

**Dynamic Partial Order Reduction
under
the Release-Acquire Semantics**

50 slides: 1 hour

Outline

- **Memory Models**
 - **SC (Sequential Consistency)**
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - **Experiments**
- **Conclusions**

Semantics:

- **Lahav, Giannarakis, Vafeiadis. Taming release-acquire consistency. **POPL, 2016.****
- **Kaiser, Dang, Dreyer, Lahav, Vafeiadis. Strong Logic for Weak Memory: Reasoning About Release-Acquire Consistency **ECOOP 2017.****

Results:

- **Parosh Aziz Abdulla, Jatin Arora, Mohamed Faouzi Atig, Shankara Narayanan Krishna. Verification of programs under the release-acquire semantics. **PLDI 2019.****
- **Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, Magnus Lång, Tuan Phong Ngo, Konstantinos Sagonas. Optimal Stateless Model Checking for Read-from Equivalence under Sequential Consistency. **OOPSLA 2019.****
- **Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, Tuan Phong Ngo. Optimal stateless model checking under the release-acquire semantics. **OOPSLA 2018.****
- **Parosh Aziz Abdulla, Stavros Aronis, Bengt Jonsson, Konstantinos Sagonas. Source Sets: A Foundation for Optimal Dynamic Partial Order Reduction. **JACM 2017.****
- **Parosh Aziz Abdulla, Stavros Aronis, Bengt Jonsson, Konstantinos Sagonas. Optimal dynamic partial order reduction. **POPL 2014.****

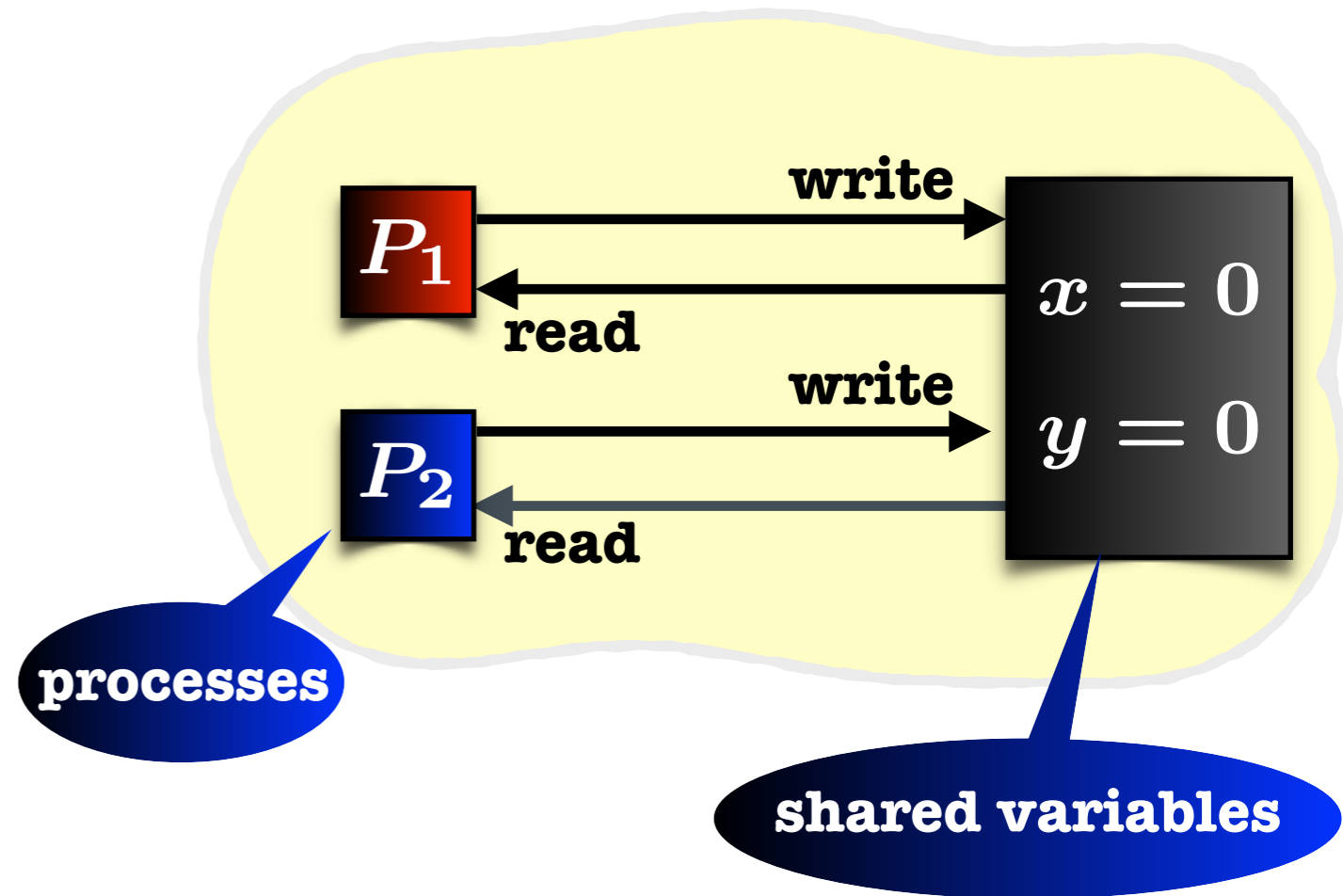
Related Work: check the reference lists in the above papers

Outline

- **Memory Models**
 - **SC (Sequential Consistency)**
 - RA (Release-Acquire)
- **Verification**
 - Dynamic Partial-Order Reduction (DPOR)
 - Experiments
- **Conclusions**

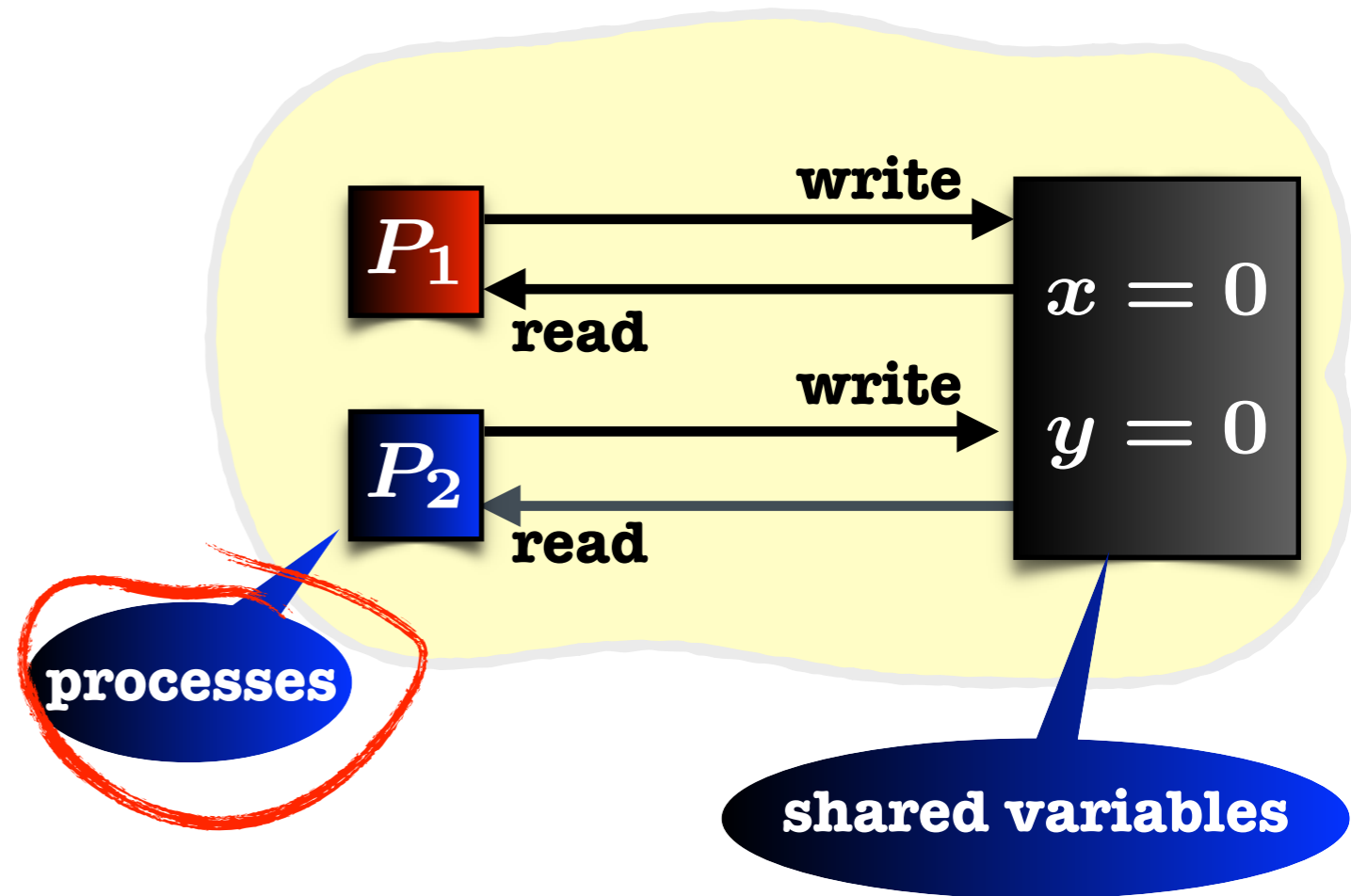
Concurrent Programs

- **Processes**
- **Shared Variables**



Concurrent Programs

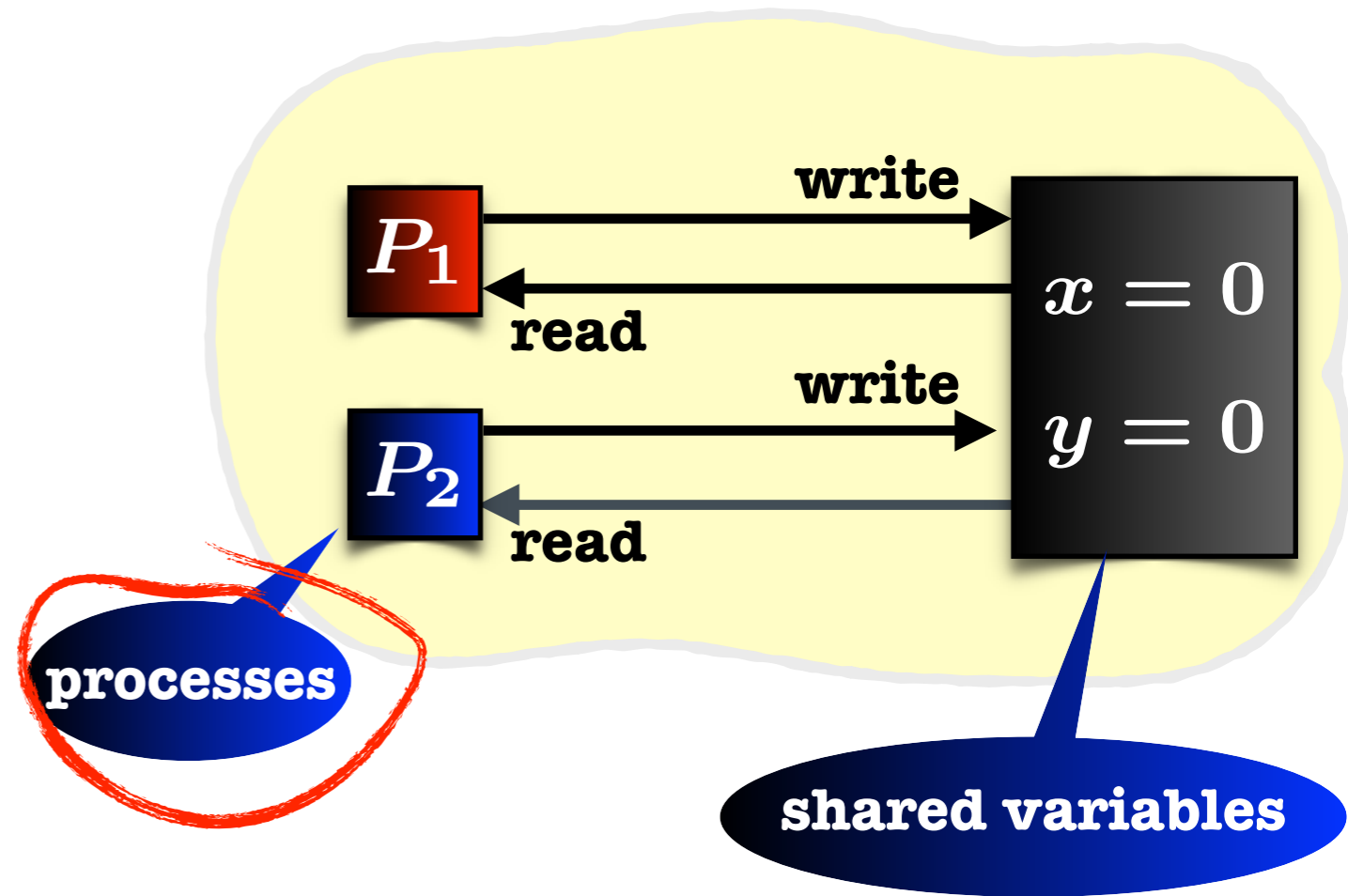
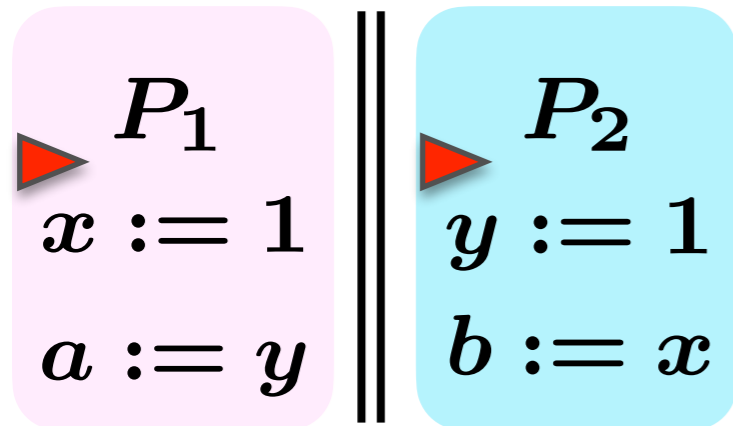
- **Processes**
- **Shared Variables**



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



Concurrent Programs

- Processes
- Shared Variables

program counter

P_1
 $x := 1$
 $a := y$

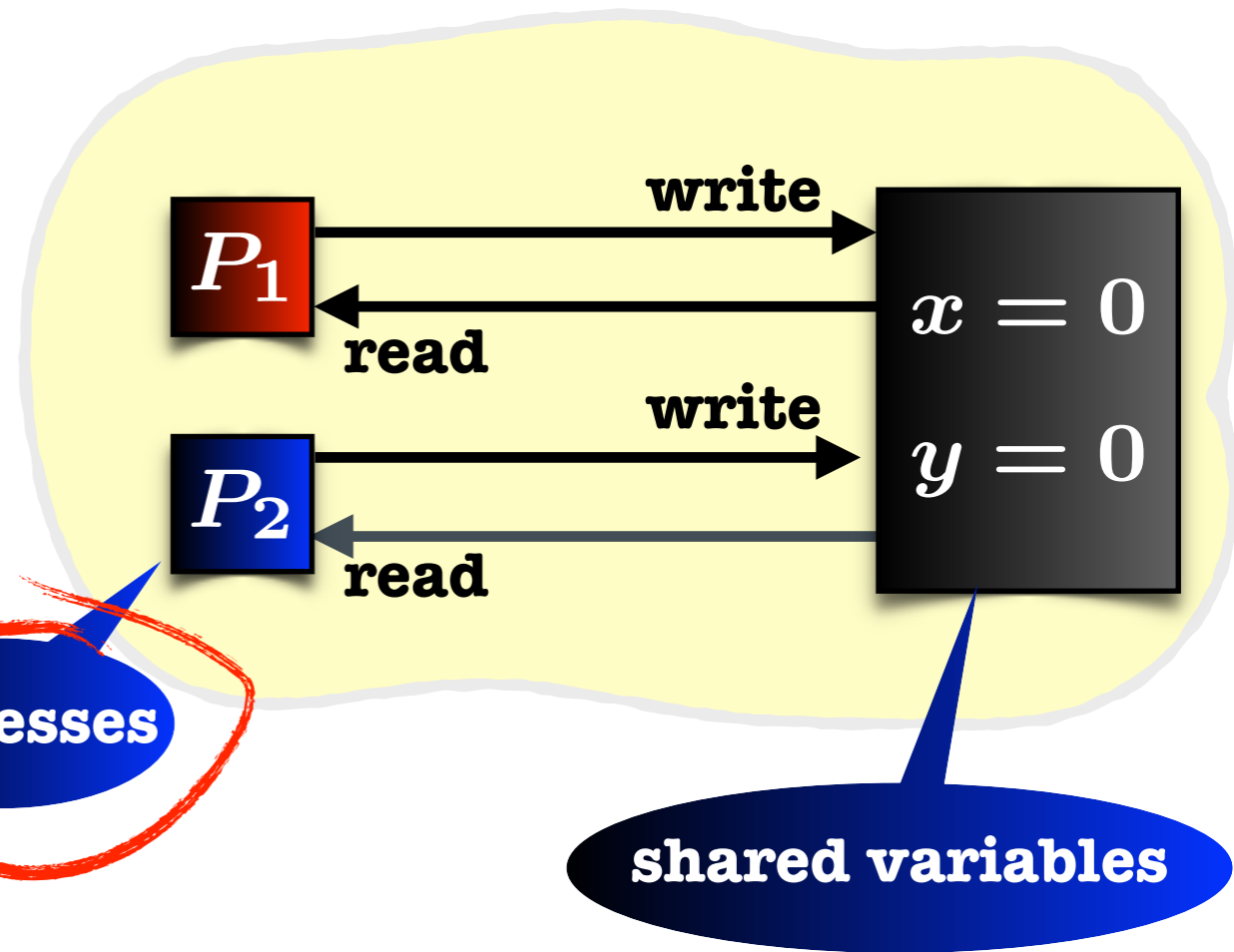
program counter

P_2
 $y := 1$
 $b := x$

0

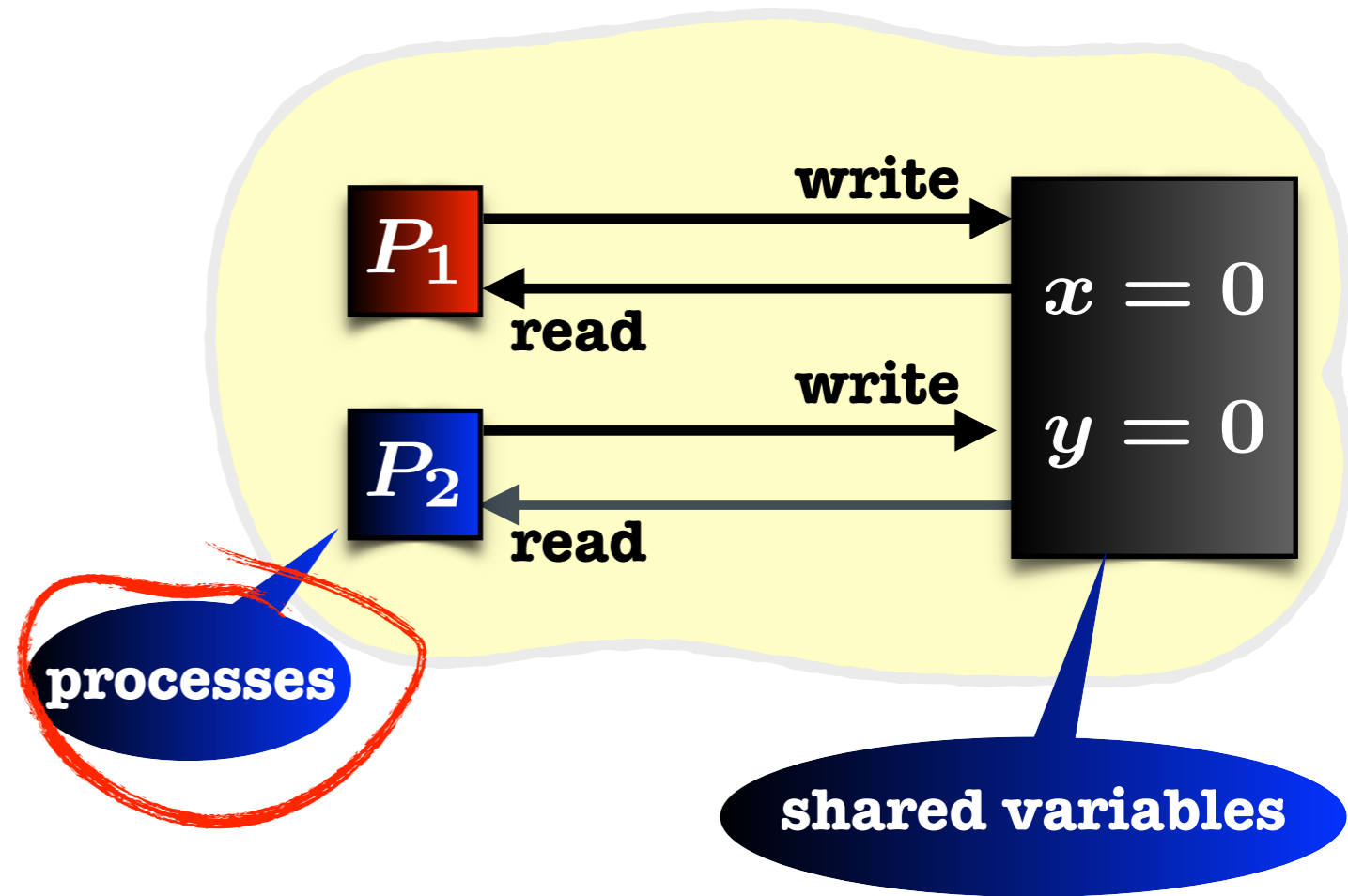
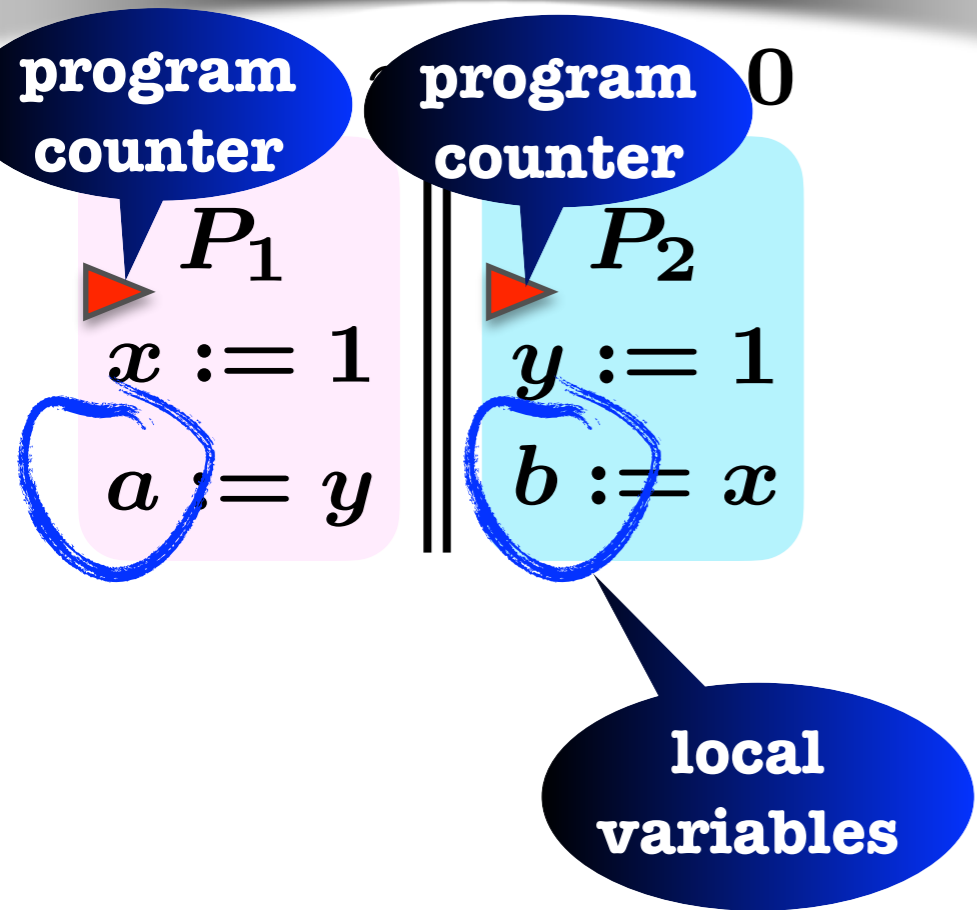
processes

shared variables



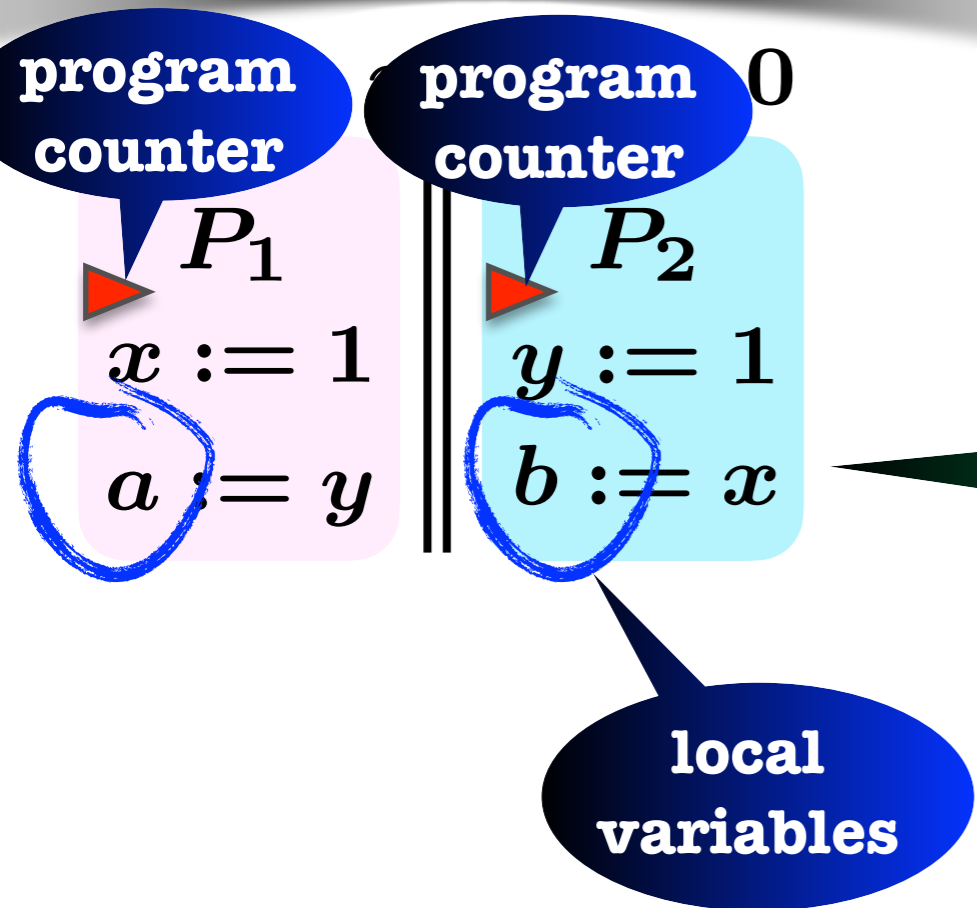
Concurrent Programs

- Processes
- Shared Variables



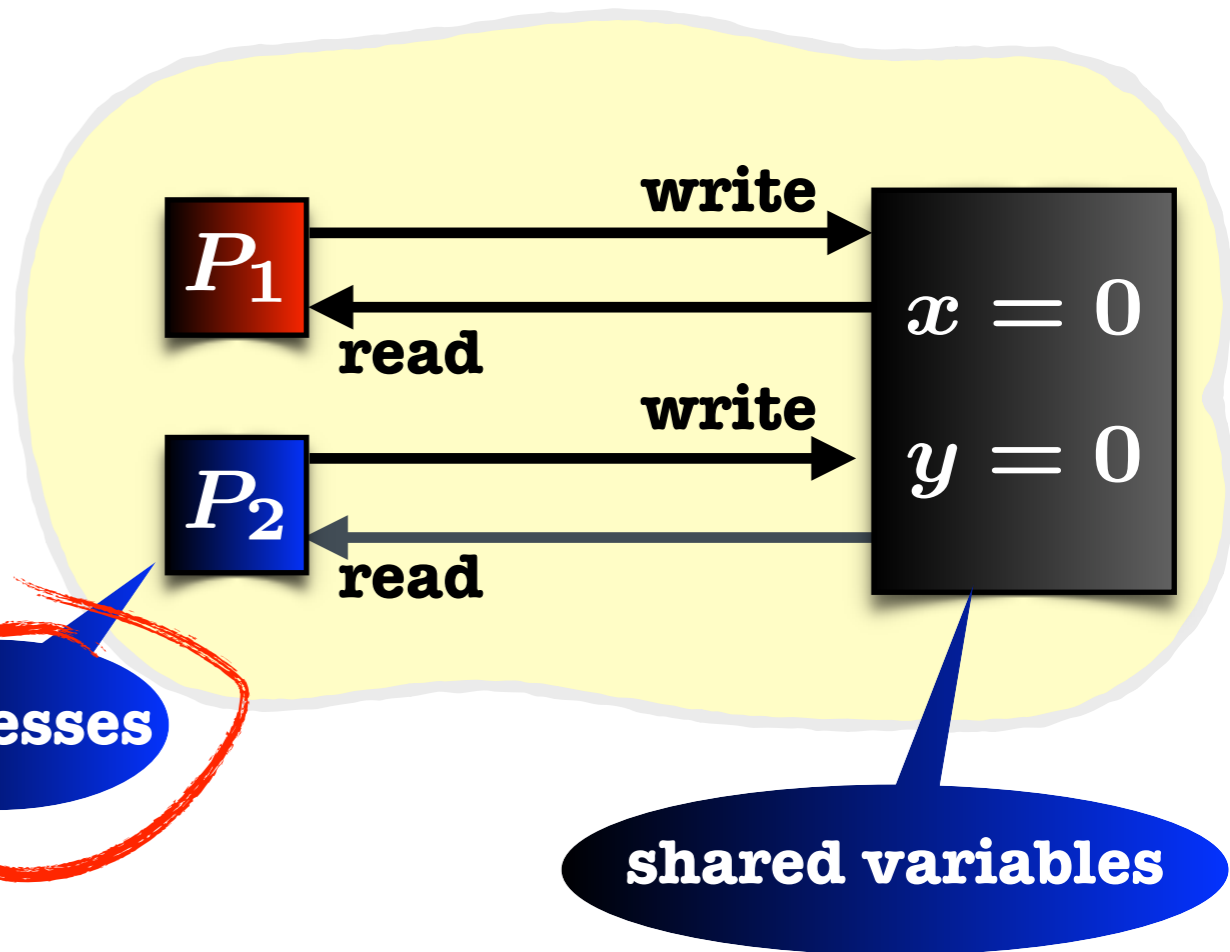
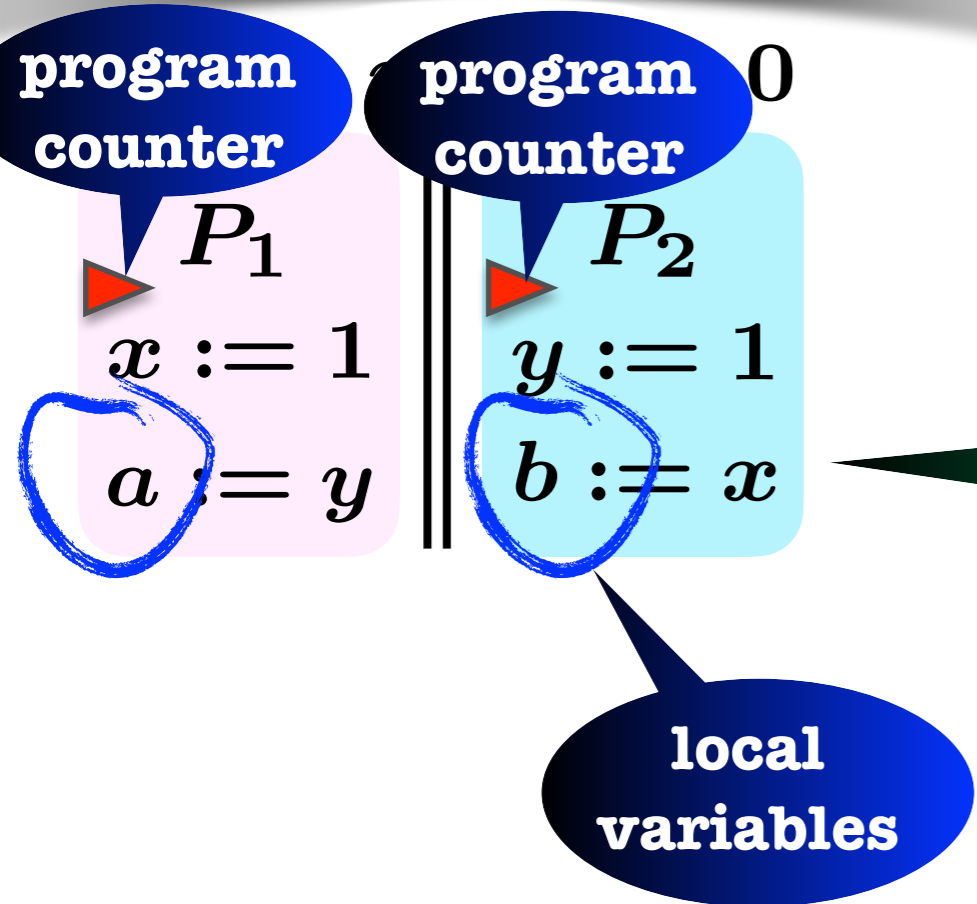
Concurrent Programs

- Processes
- Shared Variables



Concurrent Programs

- Processes
- Shared Variables



Concurrent Programs

- Processes
- Shared Variables

program counter

program counter

0

P_1

P_2

$x := 1$

$y := 1$

$a := y$

$b := x$

$(a = 0) \wedge (b = 0)?$

assertion

local variables

Dekker's protocol

Store Buffering (SB)

processes

shared variables

P_1

P_2

write

read

write

read

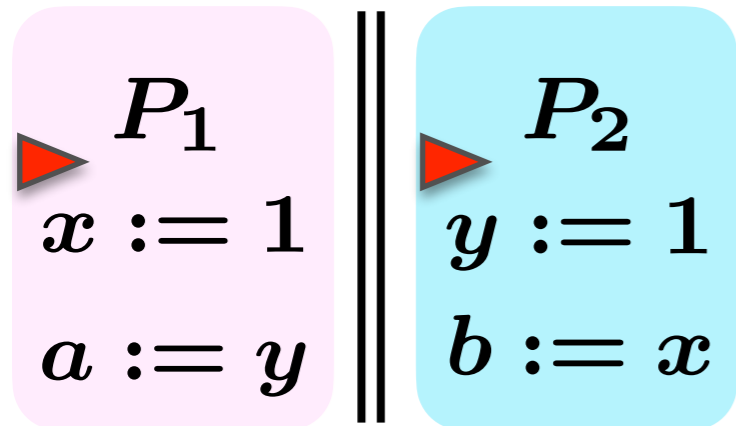
$x = 0$

$y = 0$

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

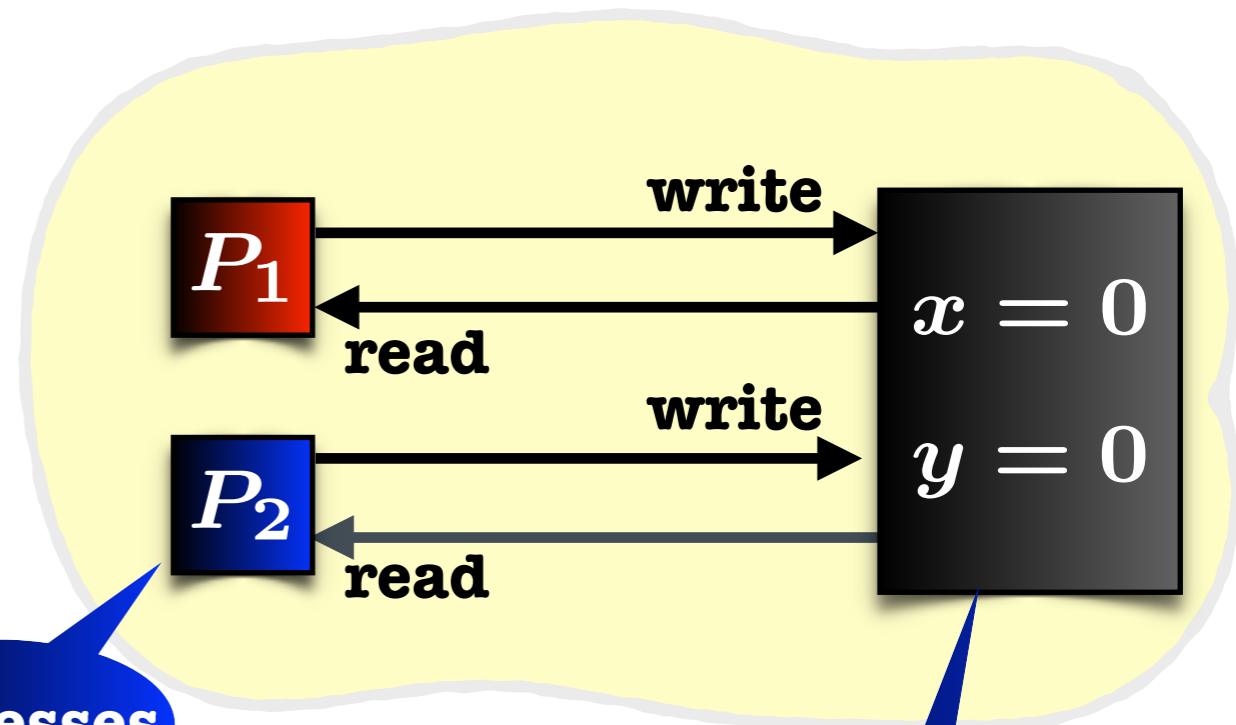
assertion

Dekker's protocol

Store Buffering (SB)

processes

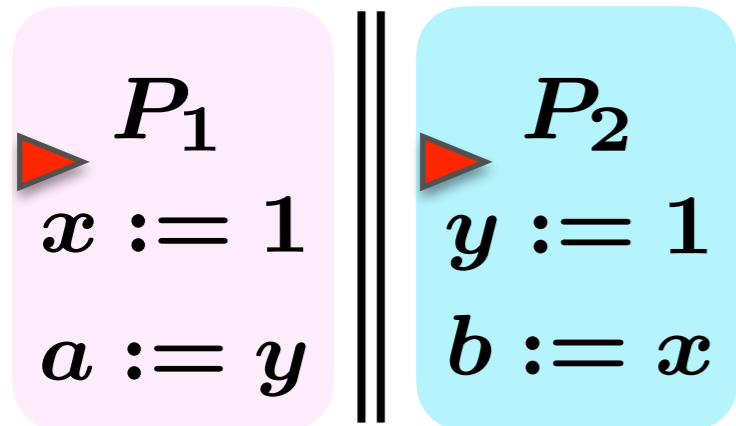
shared variables



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

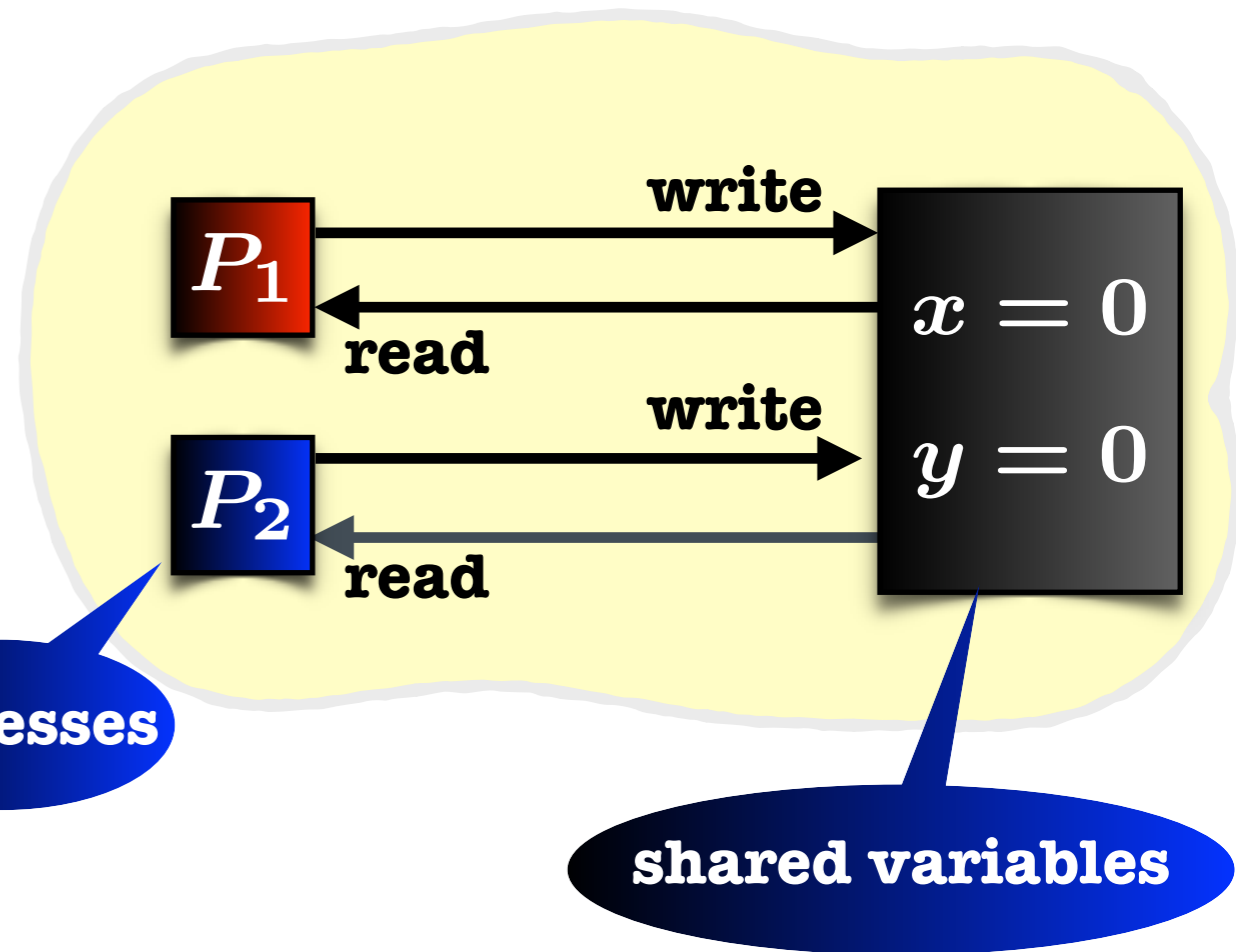
assertion

Dekker's protocol

Store Buffering (SB)

processes

shared variables



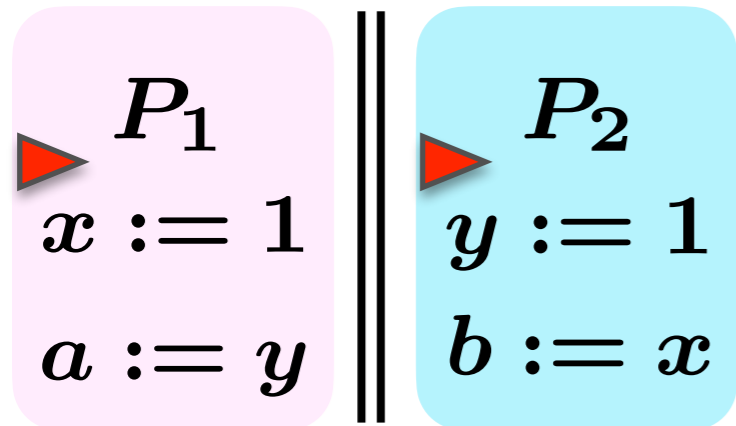
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

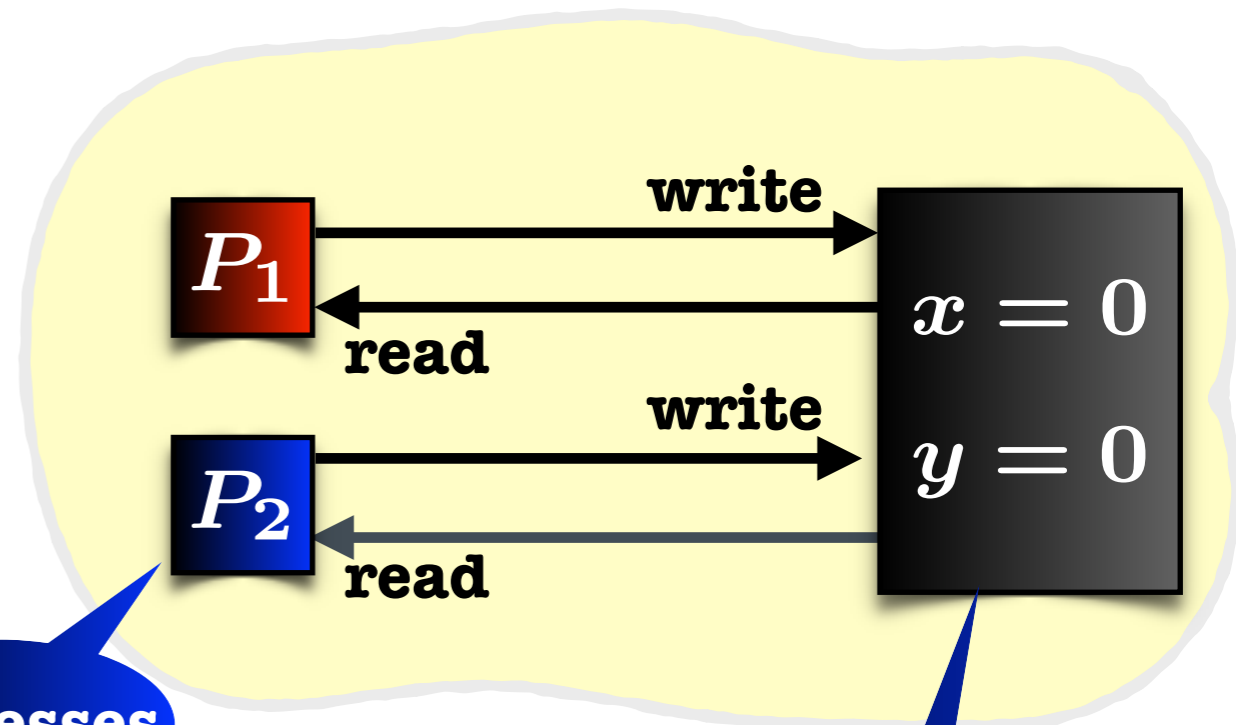
Dekker's protocol

Store Buffering (SB)

processes

shared variables

$\pi : \text{run}$



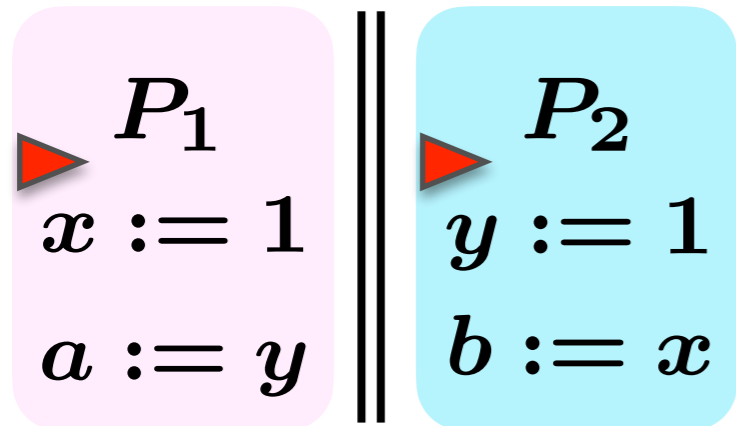
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Dekker's protocol

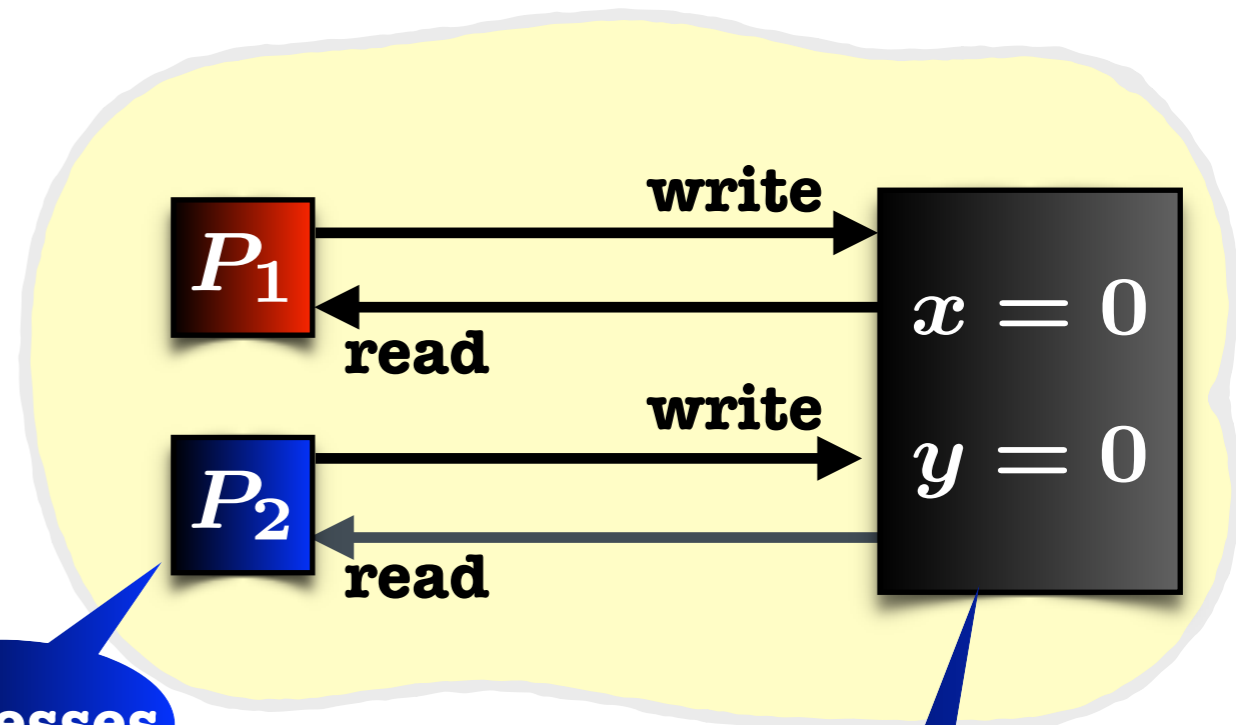
Store Buffering (SB)

processes

events

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



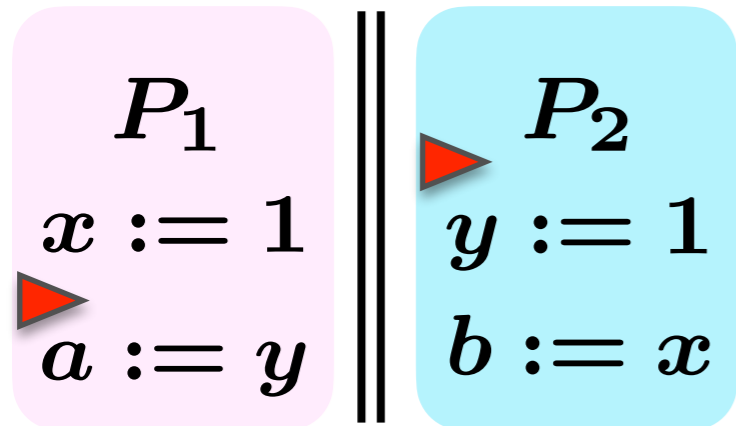
shared variables

$\pi : \text{run}$

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Dekker's protocol

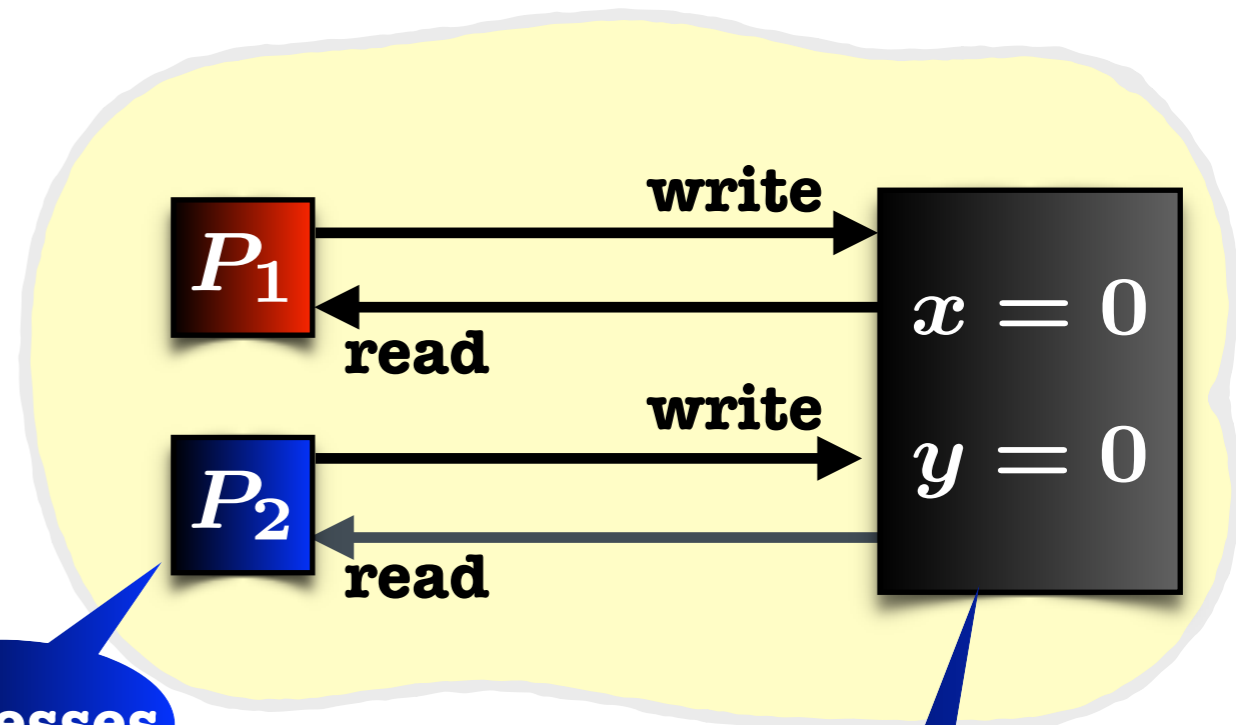
Store Buffering (SB)

processes

events

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



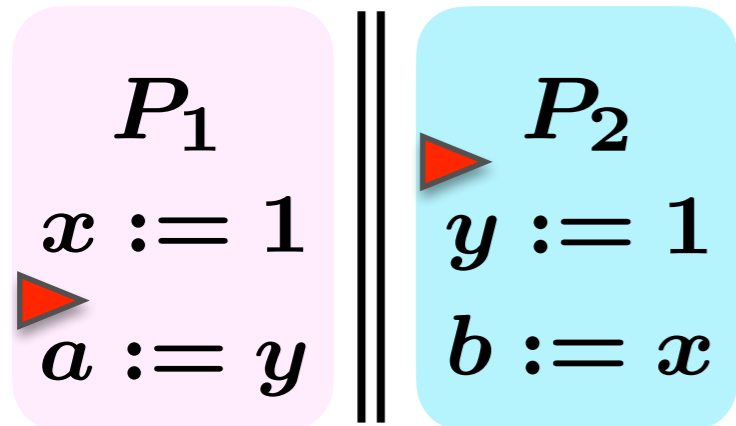
shared variables

$\pi : \text{run}$

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

Dekker's protocol

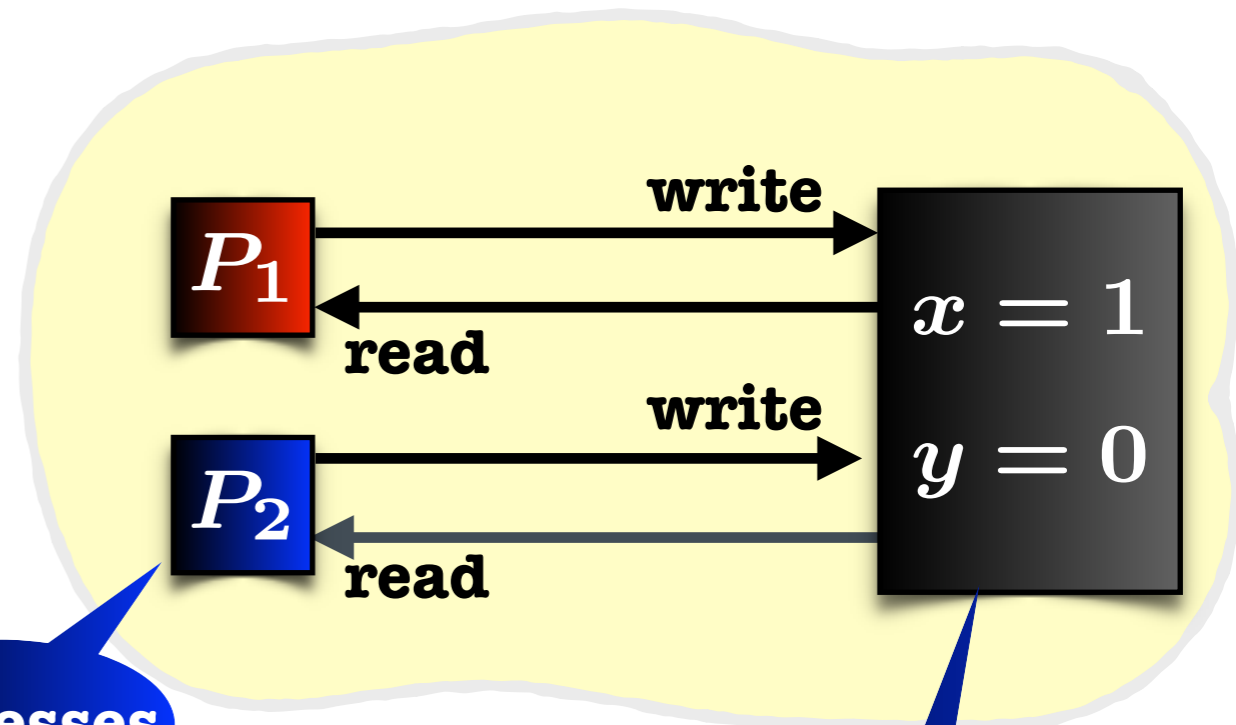
Store Buffering (SB)

processes

events

shared variables

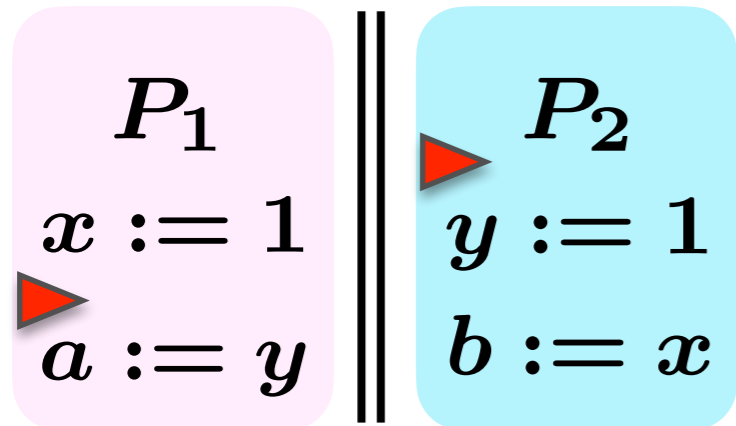
$\pi : \text{run}$



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$

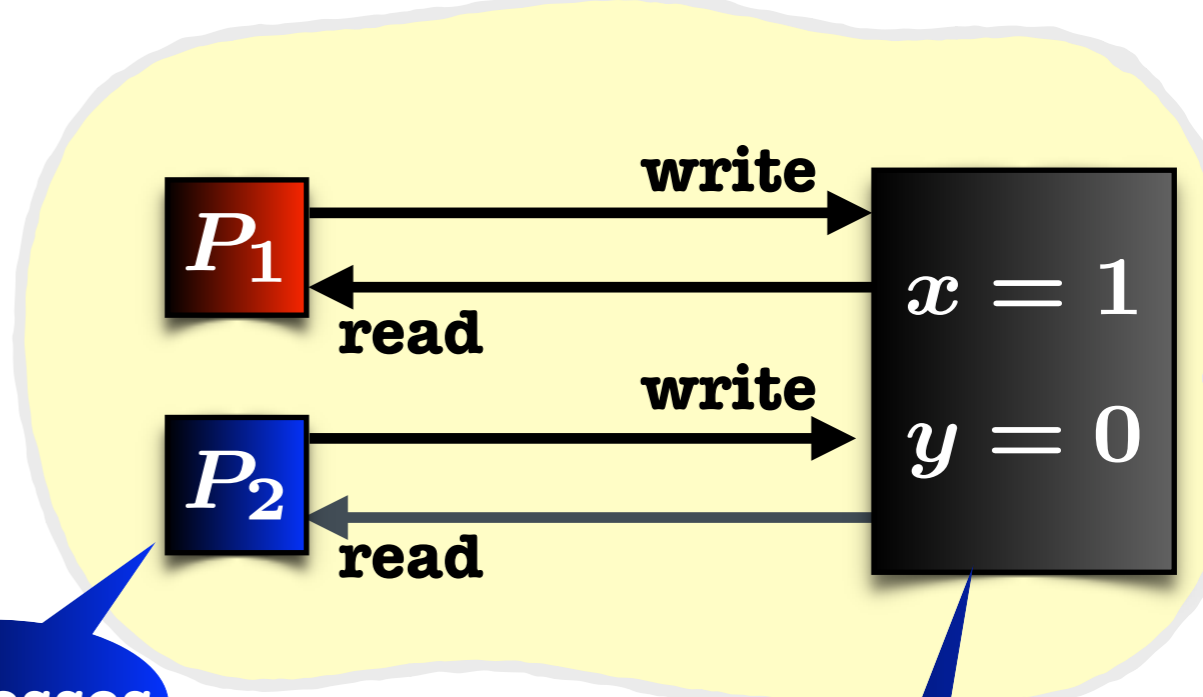


$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



processes

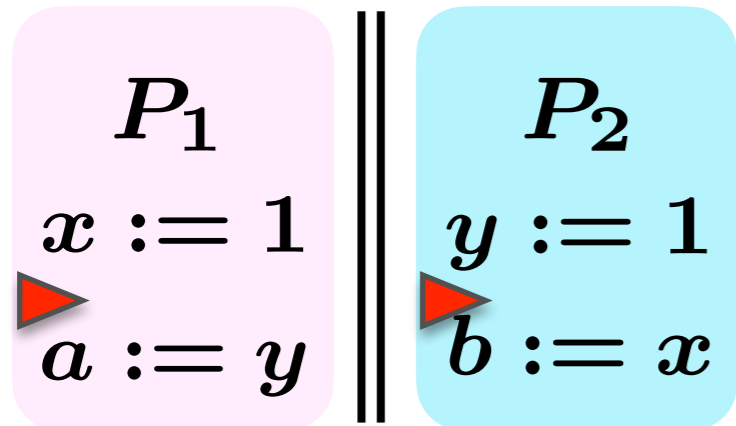
shared variables



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

Dekker's protocol

Store Buffering (SB)

processes

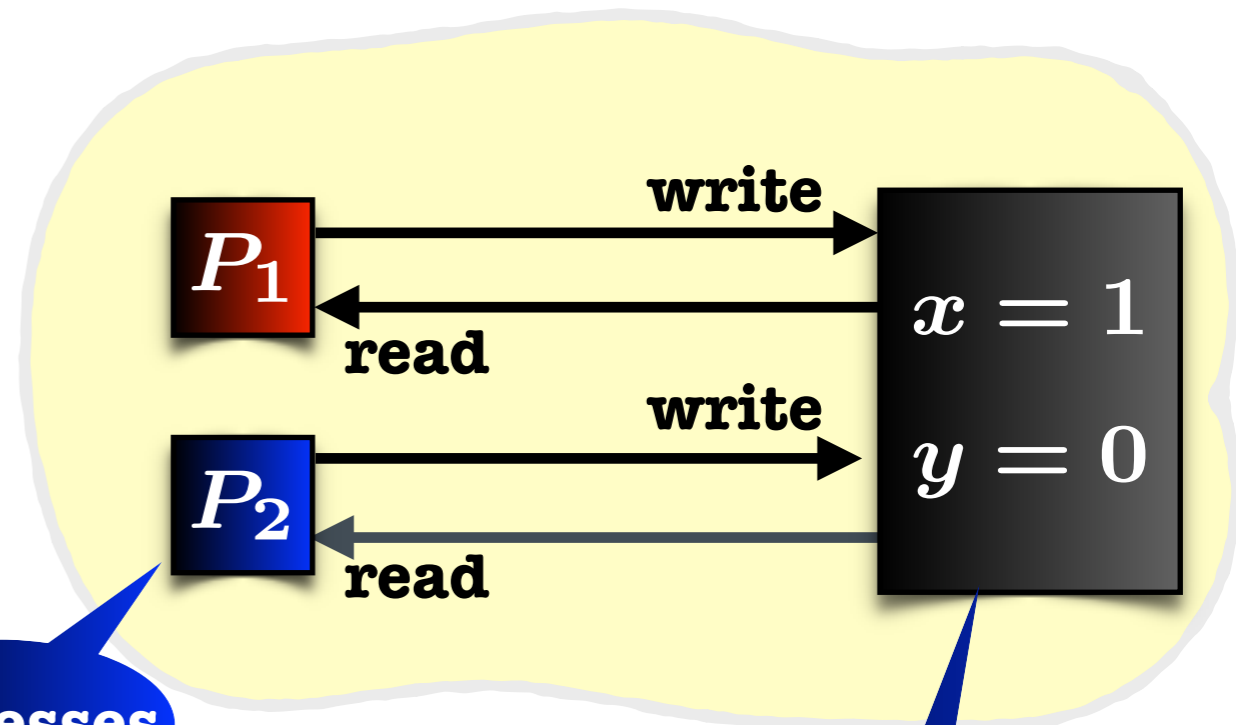
events

e_1

$x := 1$

$\pi : \text{run}$

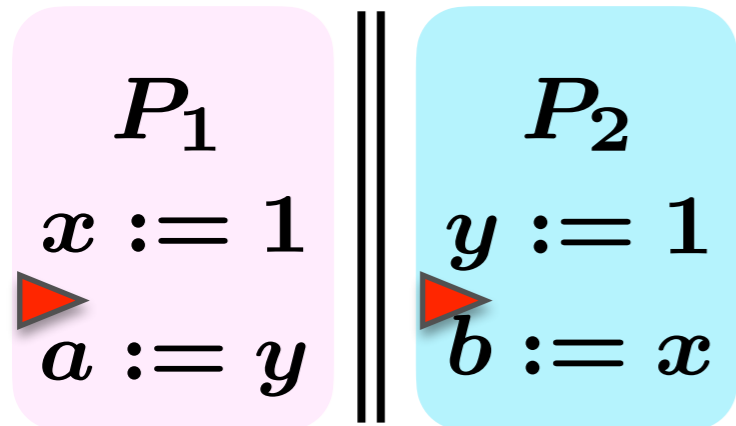
shared variables



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Dekker's protocol

Store Buffering (SB)

processes

shared variables

events

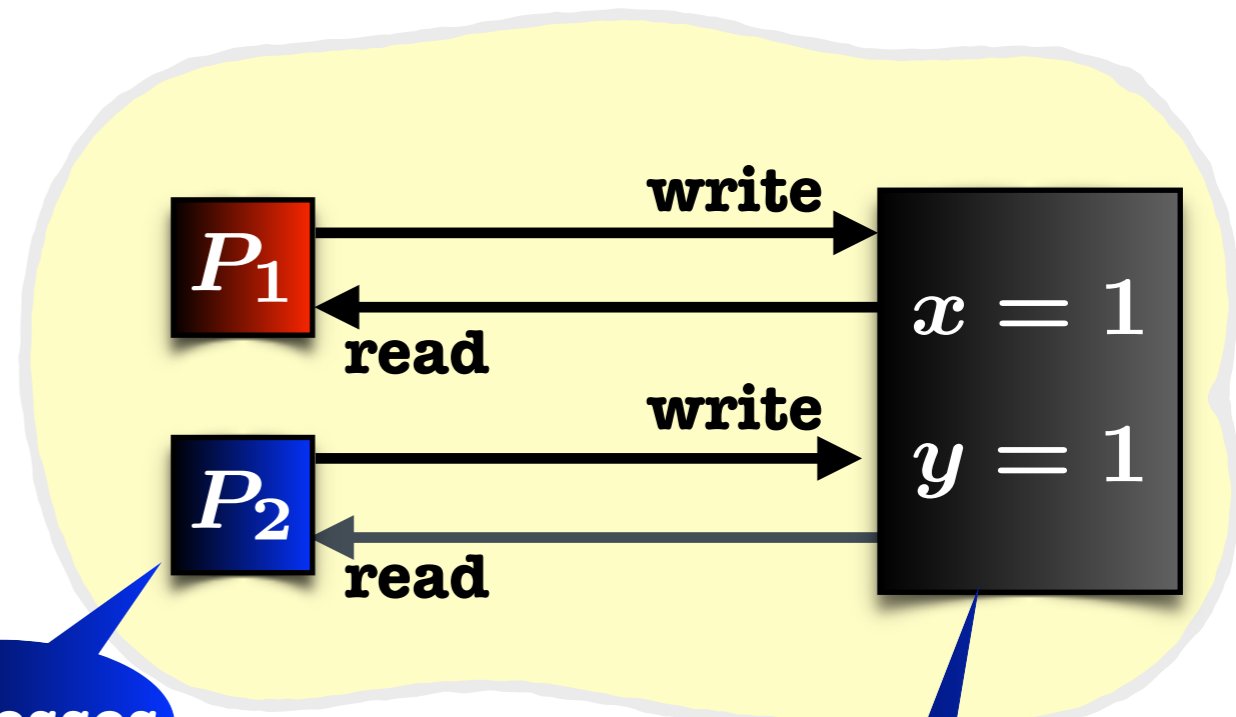
e_1

$x := 1$

$\pi : \text{run}$

Sequential Consistency (SC)

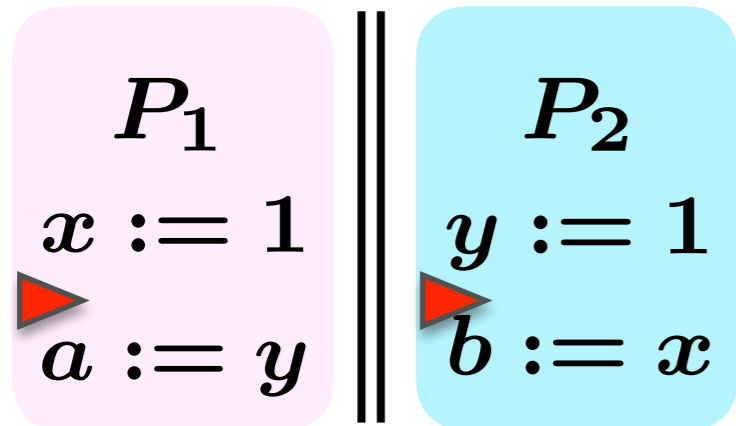
- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

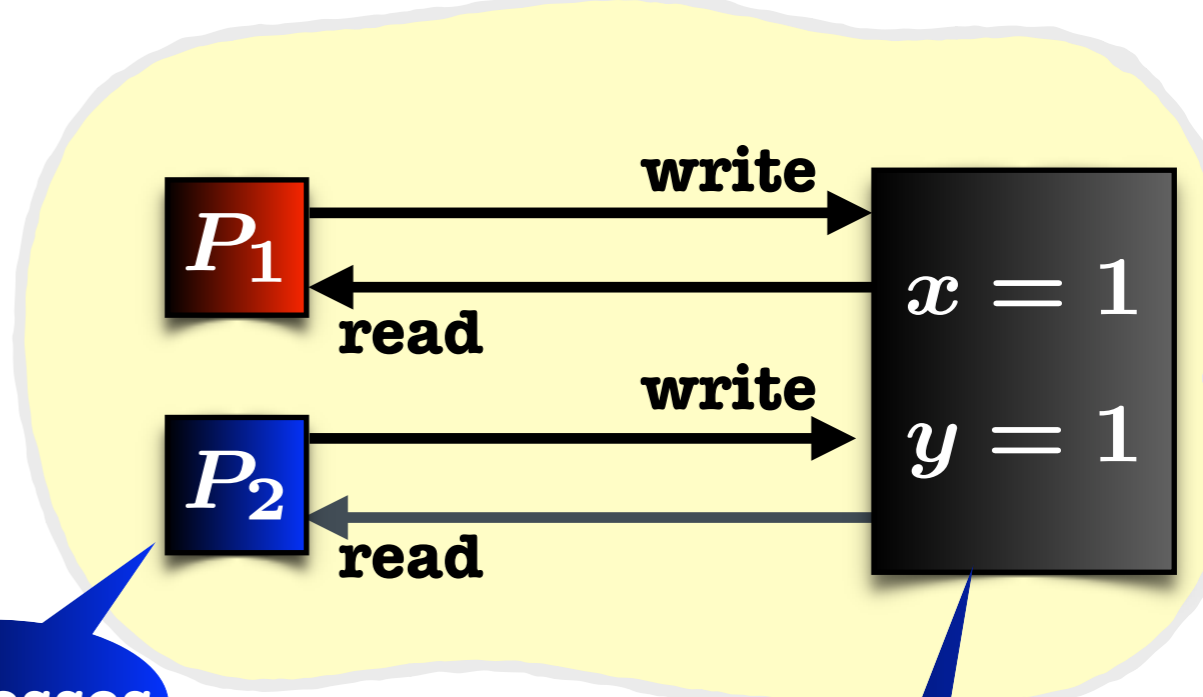
- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



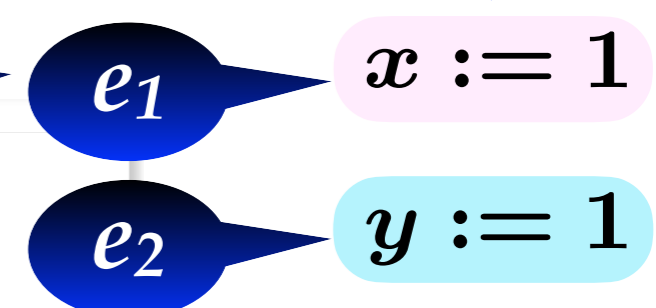
processes

events

shared variables



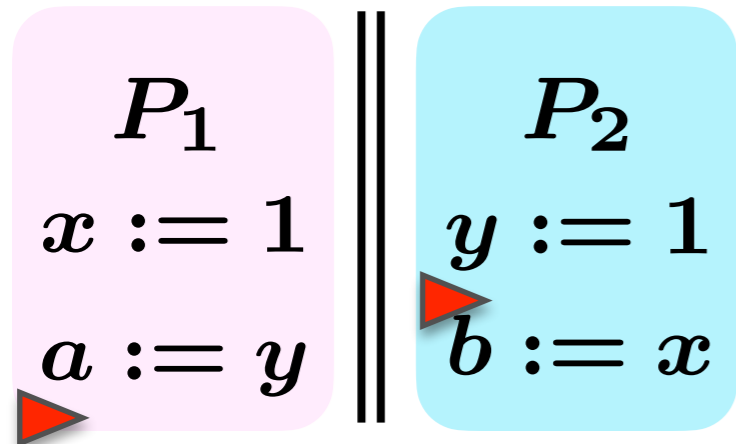
$\pi : \text{run}$



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

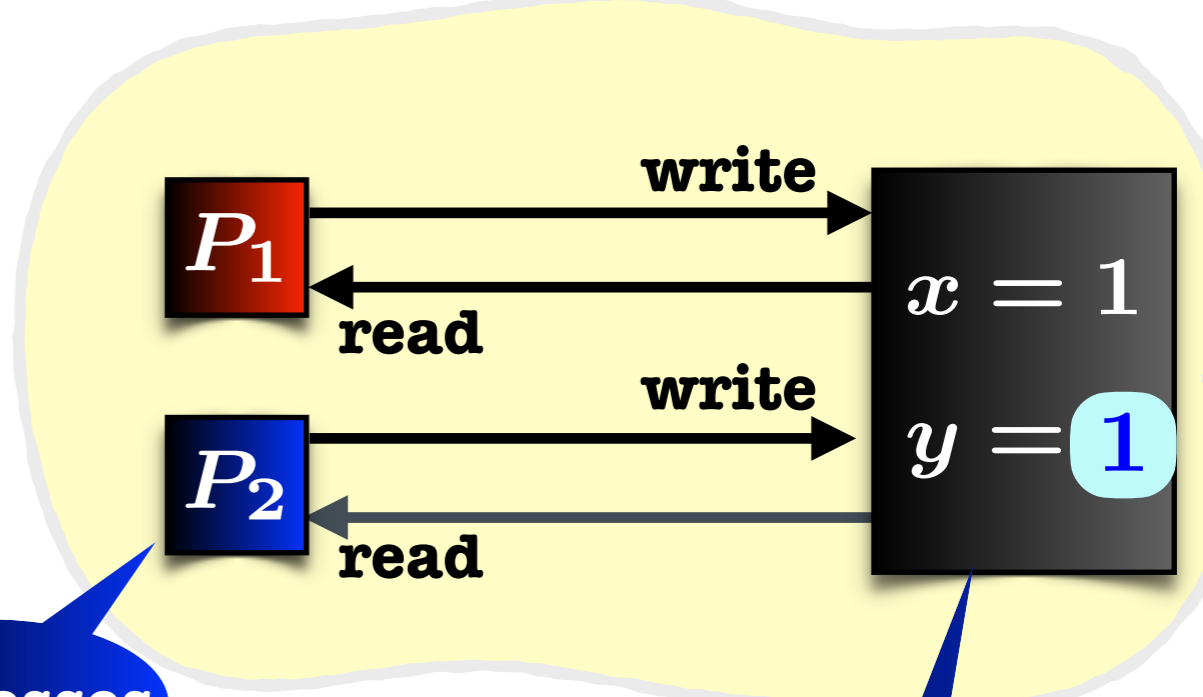
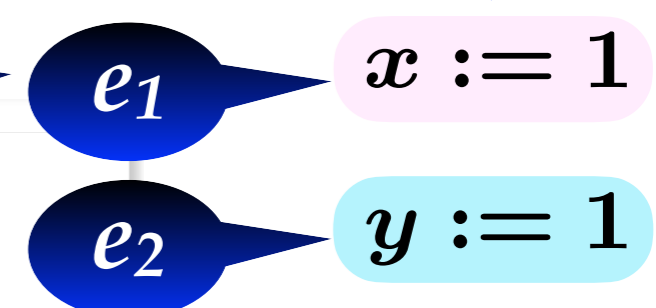
Dekker's protocol
Store Buffering (SB)

processes

events

shared variables

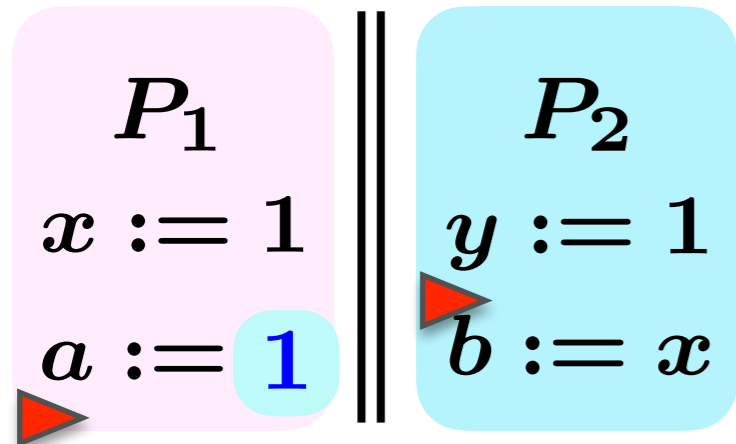
$\pi : \text{run}$



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

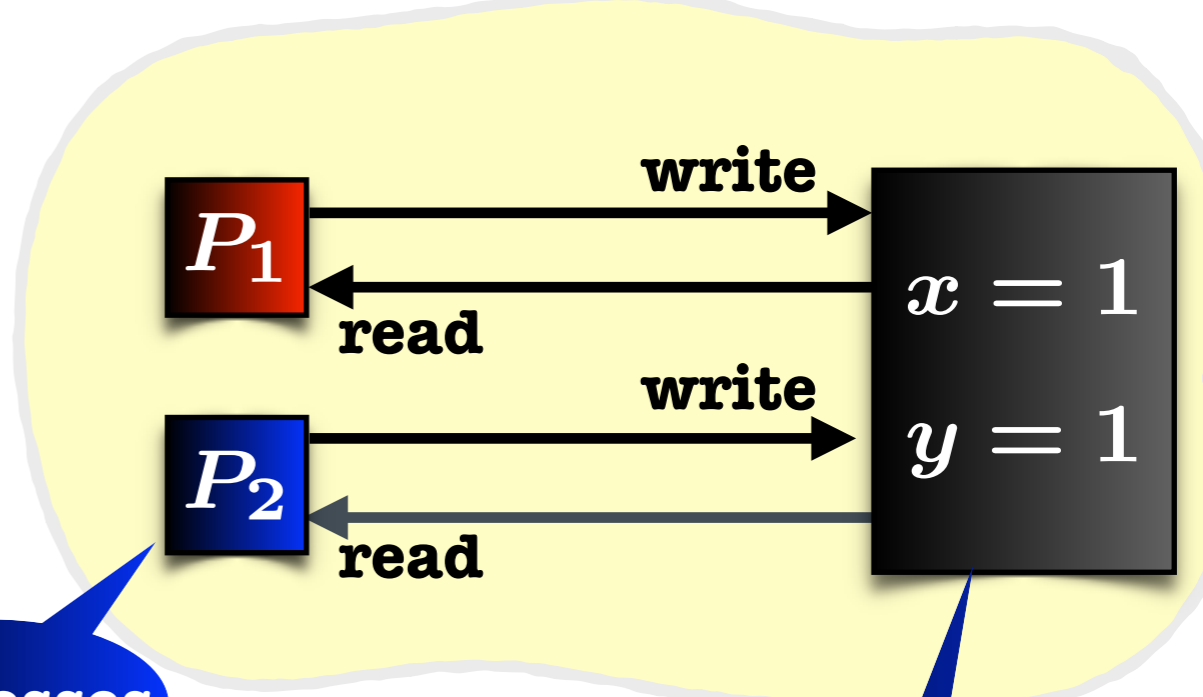
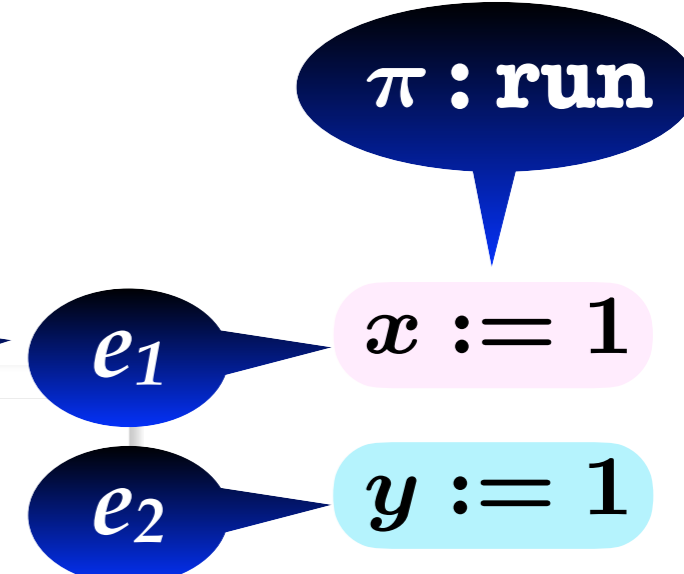
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



processes

events

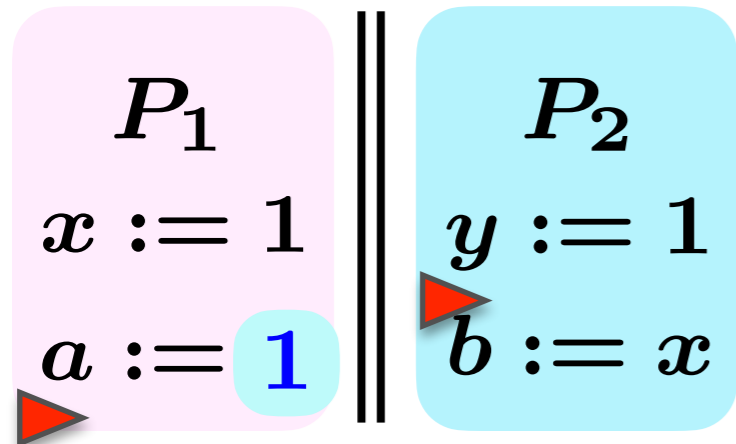


shared variables

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

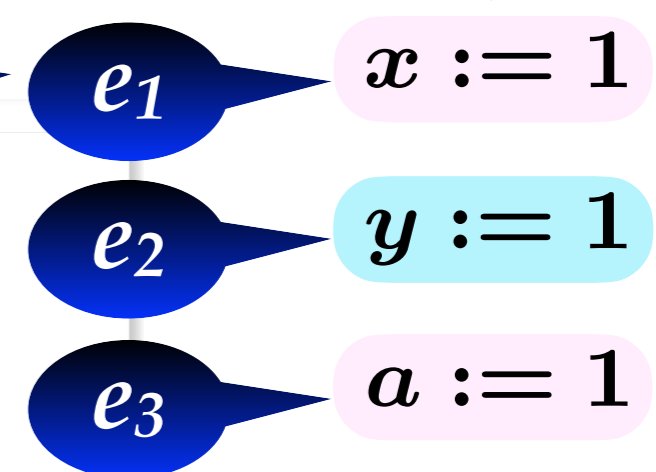
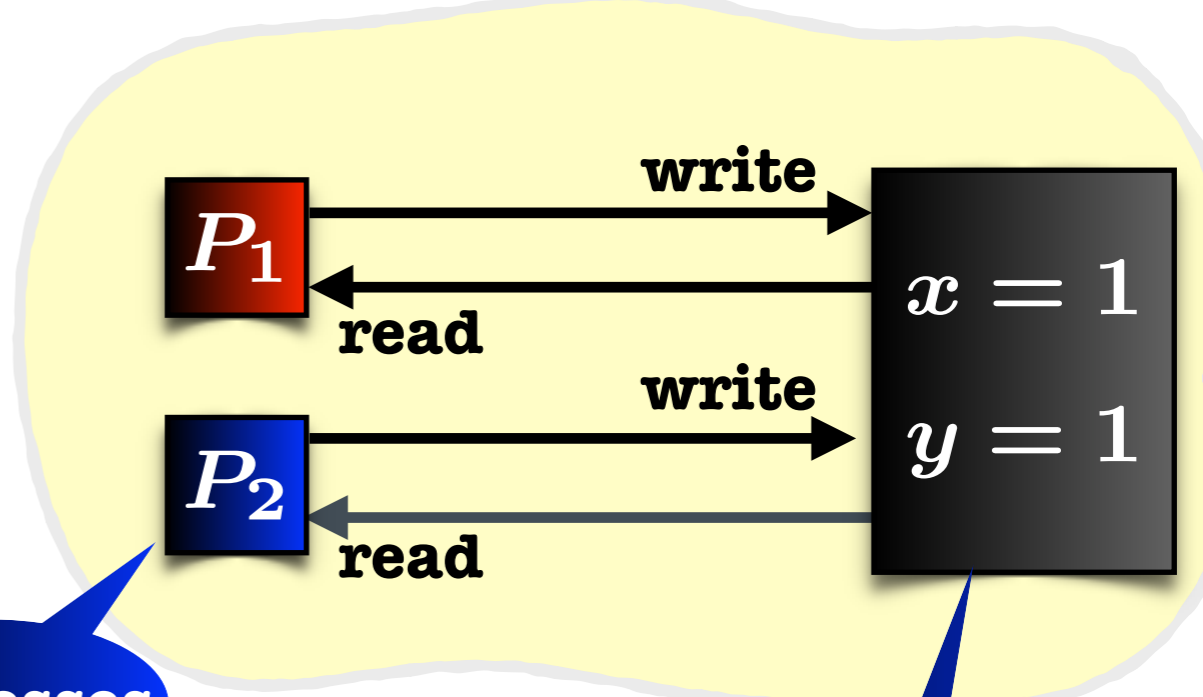
Dekker's protocol
Store Buffering (SB)

processes

events

shared variables

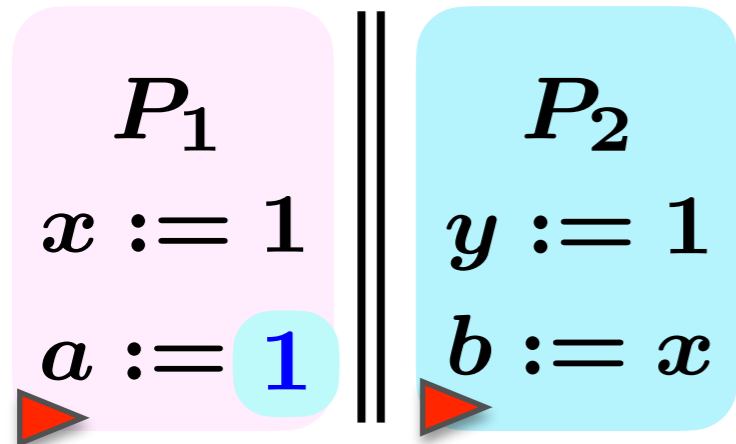
$\pi : \text{run}$



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

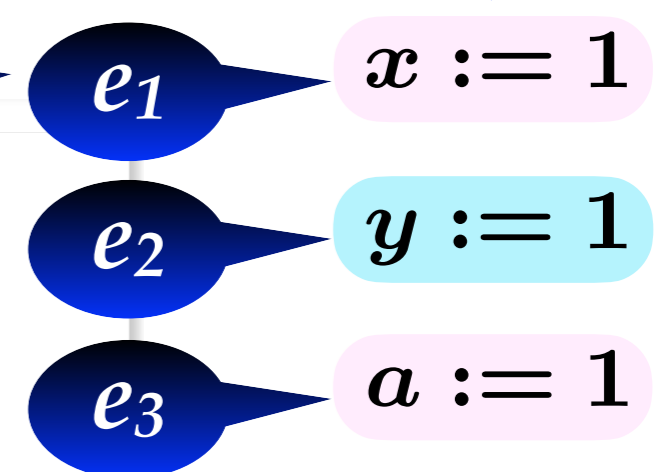
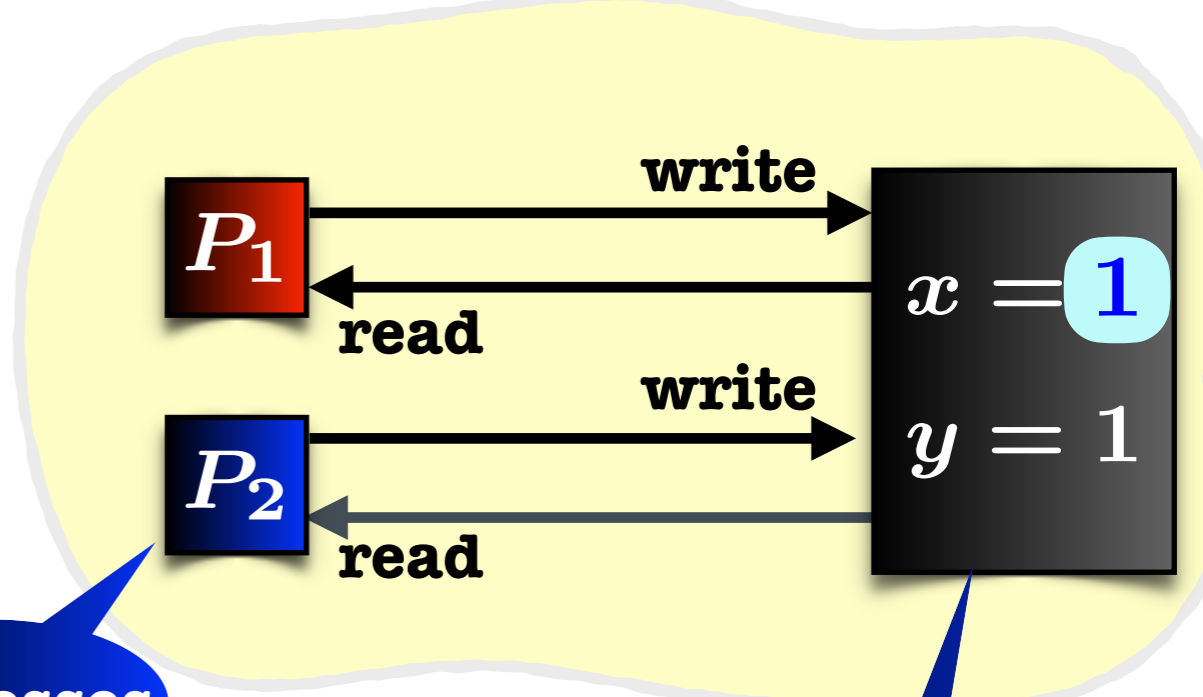
Dekker's protocol
Store Buffering (SB)

processes

events

shared variables

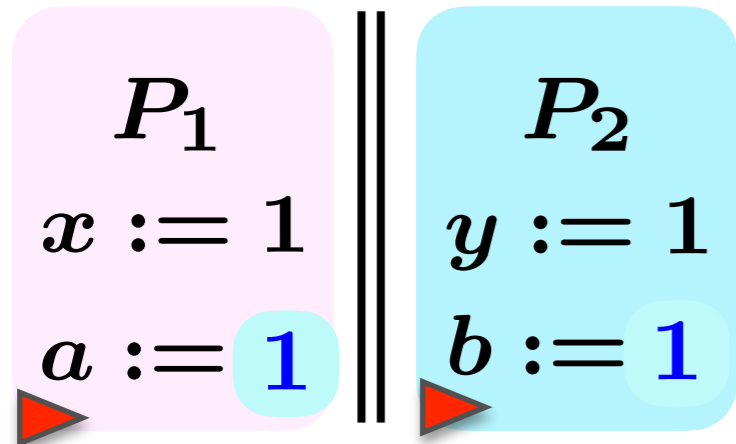
$\pi : \text{run}$



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

Dekker's protocol

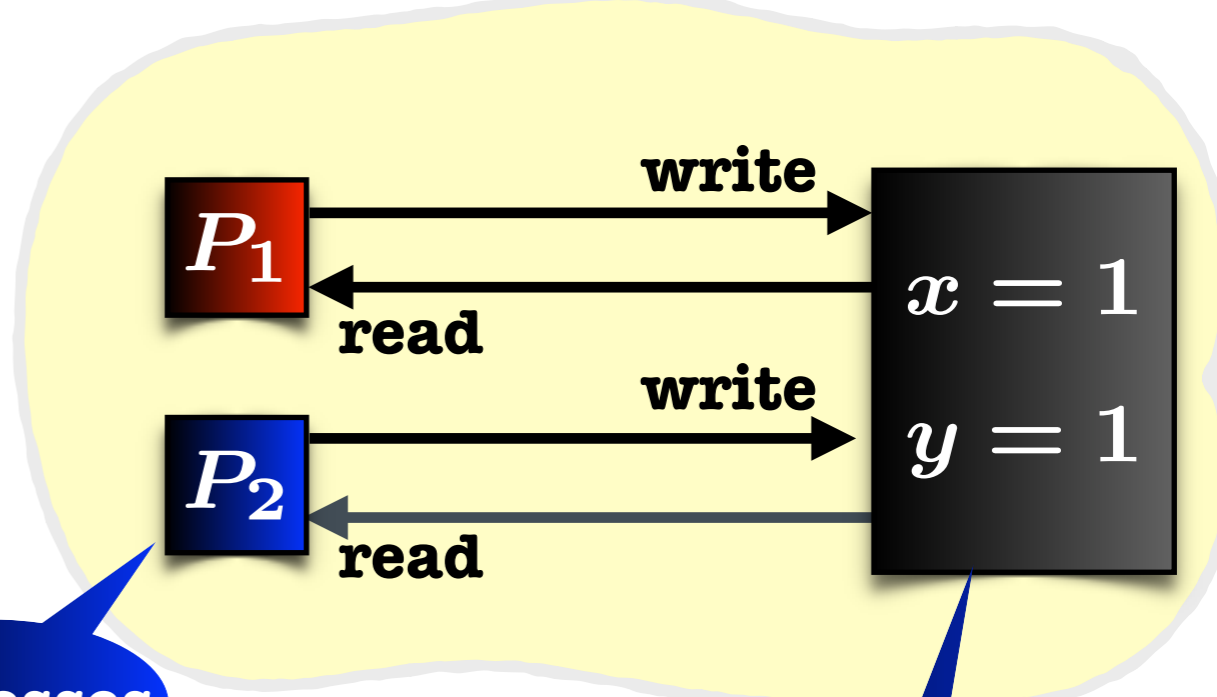
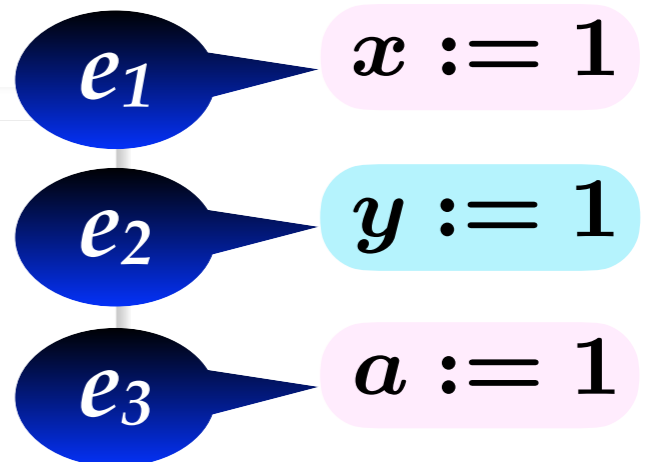
Store Buffering (SB)

processes

events

shared variables

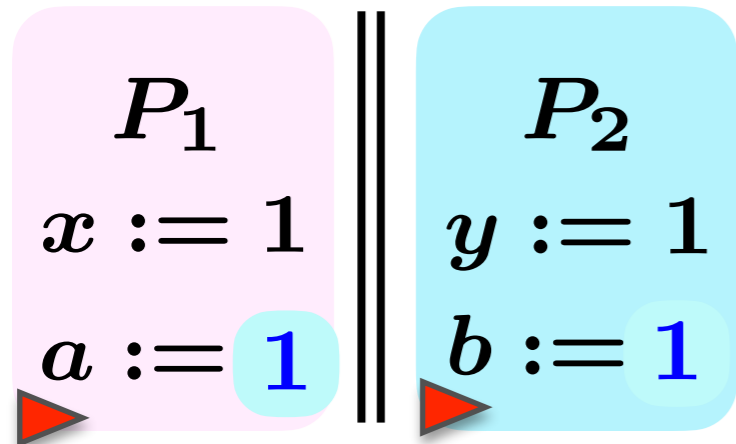
$\pi : \text{run}$



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$

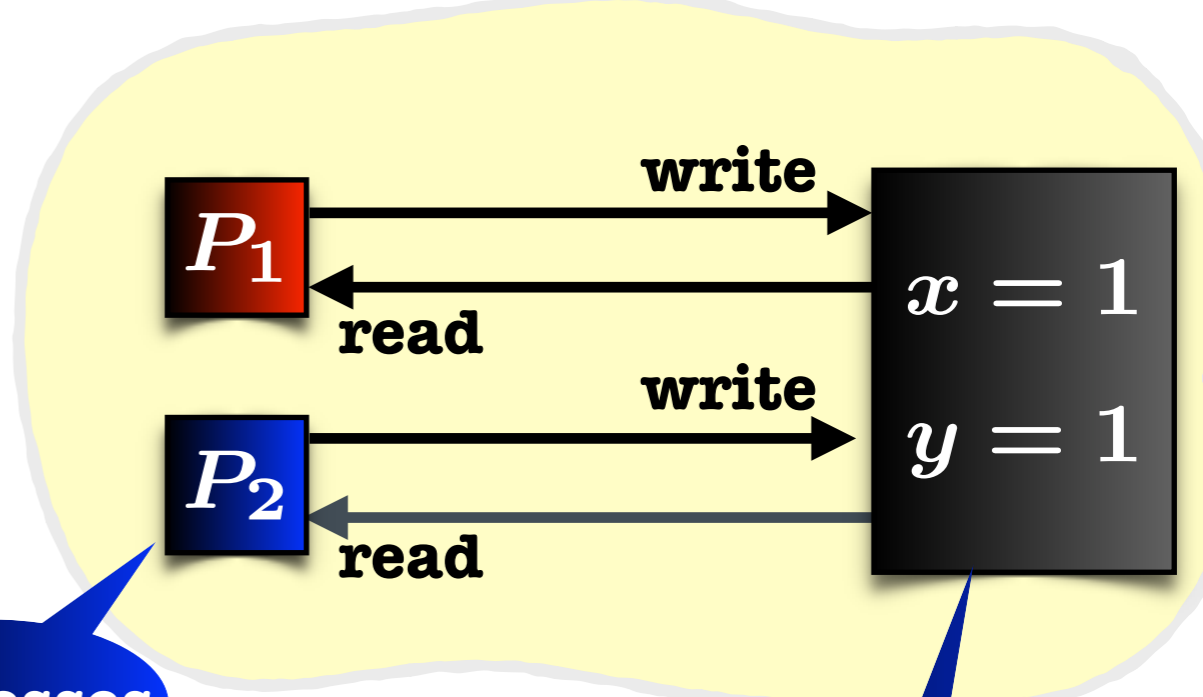


$(a = 0) \wedge (b = 0)?$

assertion

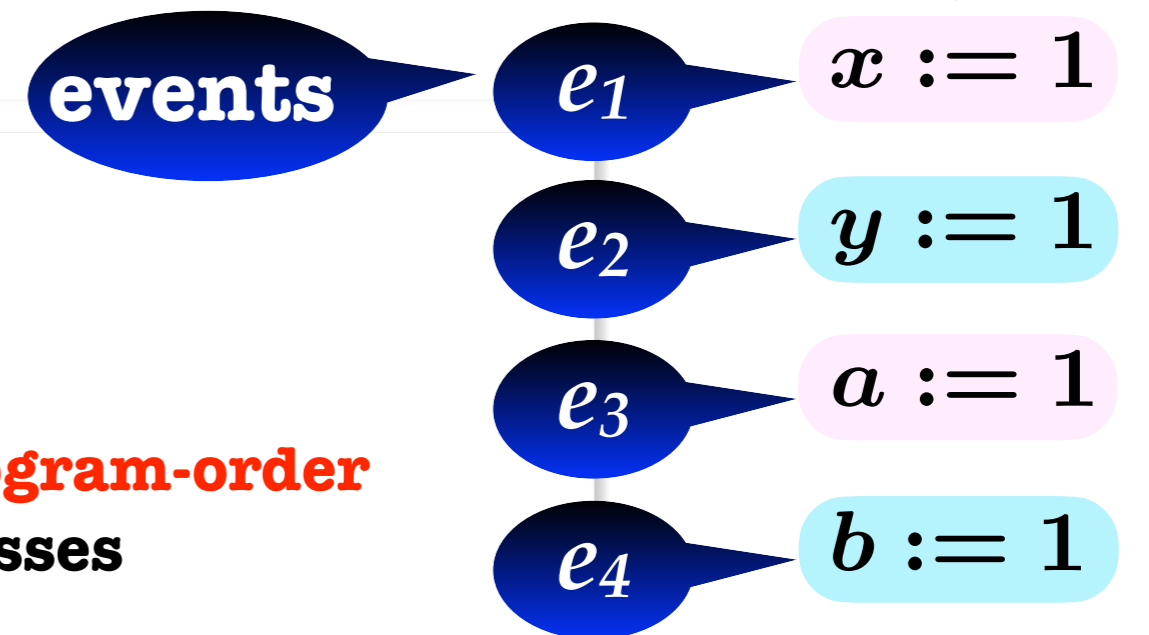
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



processes

shared variables



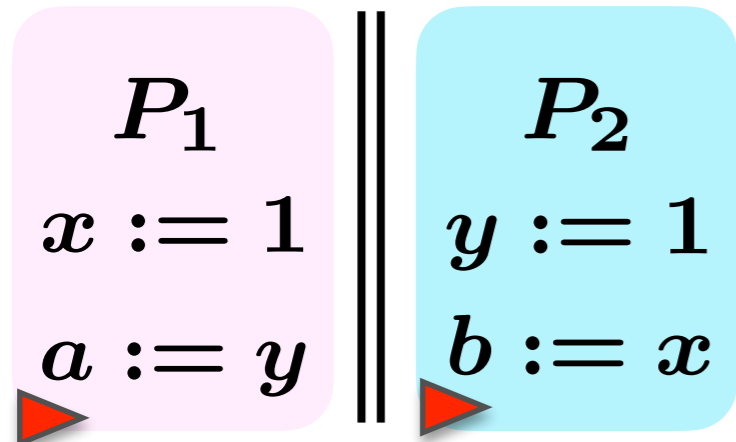
events

$\pi : \text{run}$

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

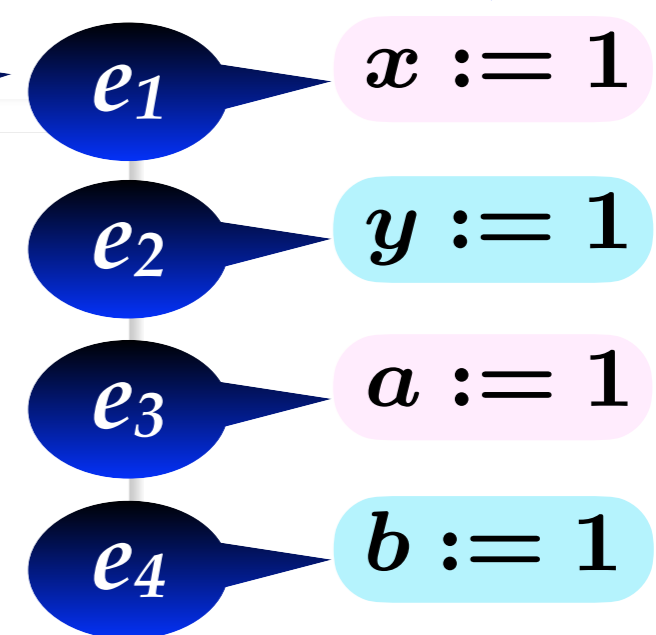
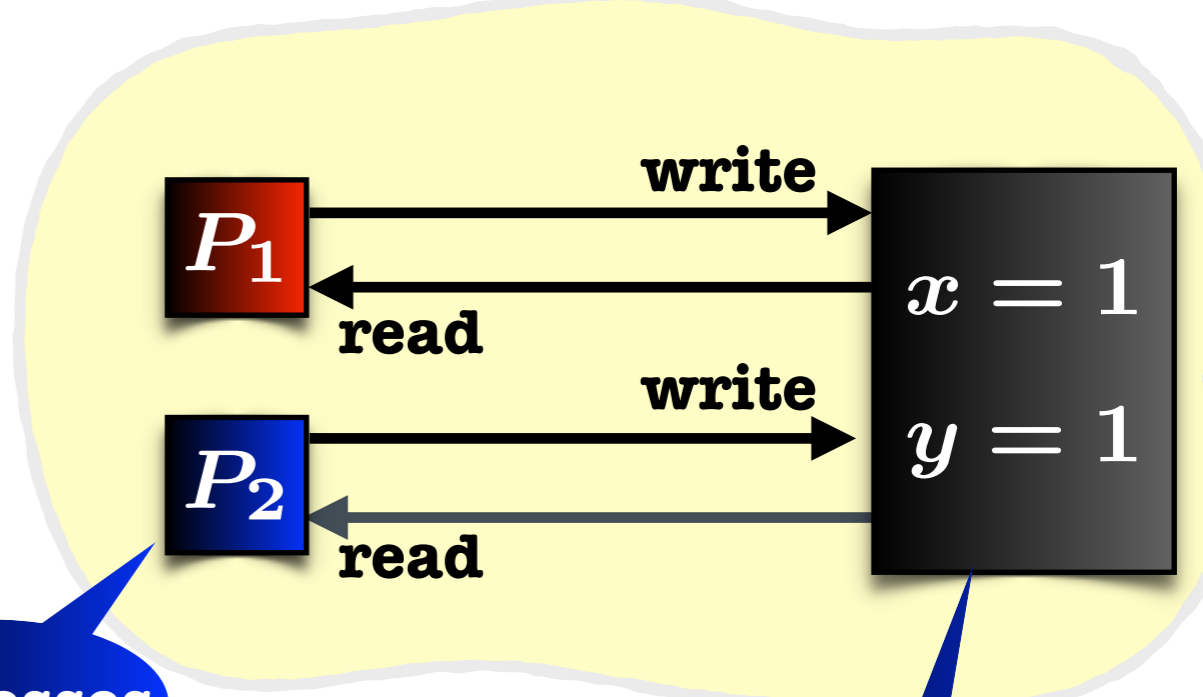
Dekker's protocol
Store Buffering (SB)

processes

events

shared variables

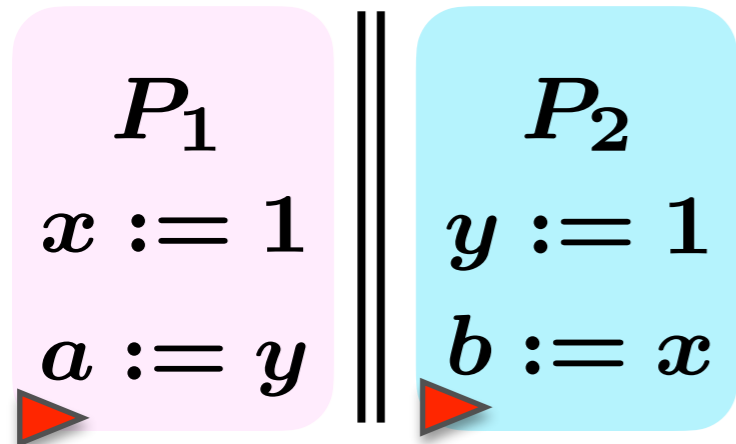
$\pi : \text{run}$



Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

assertion

unreachable

Dekker's protocol

Store Buffering (SB)

processes

shared variables

events

$\pi : \text{run}$

e_1

$x := 1$

e_2

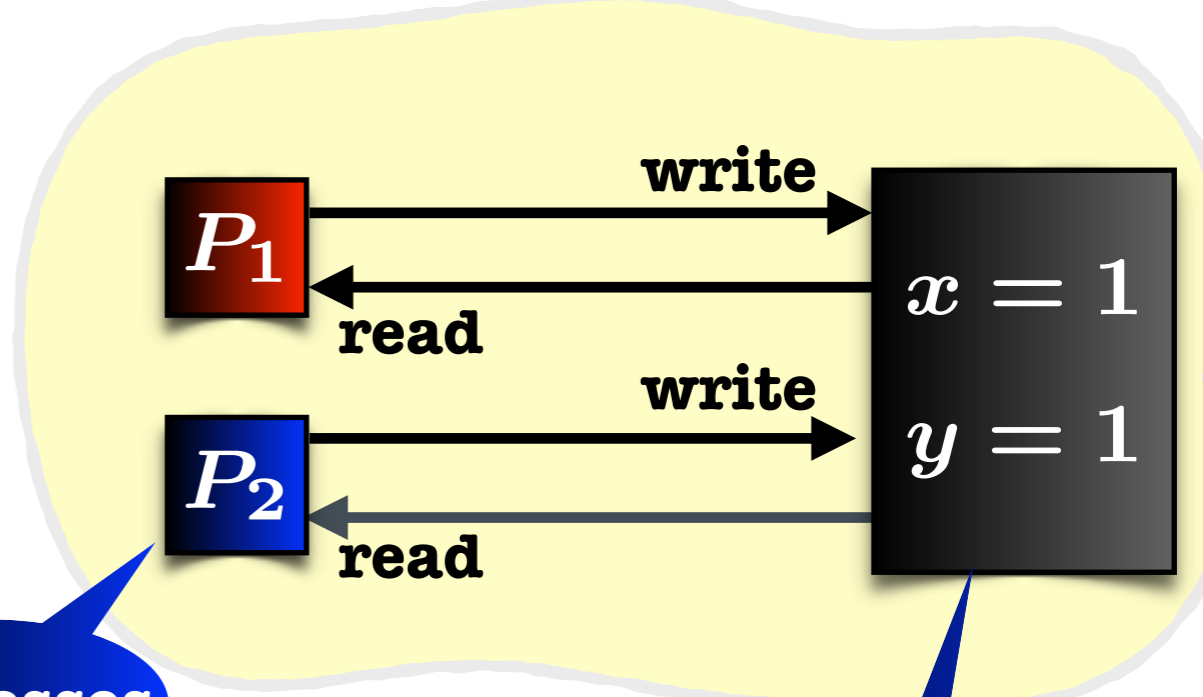
$y := 1$

e_3

$a := 1$

e_4

$b := 1$



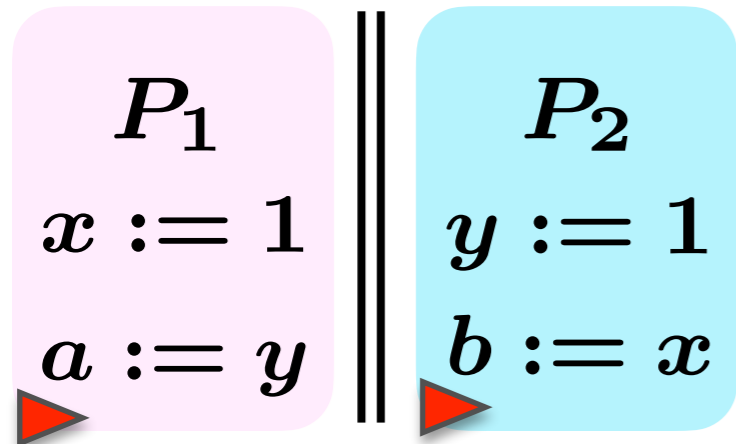
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

Concurrent Programs

- Processes
- Shared Variables

Initially: $x = y = 0$



$(a = 0) \wedge (b = 0)?$

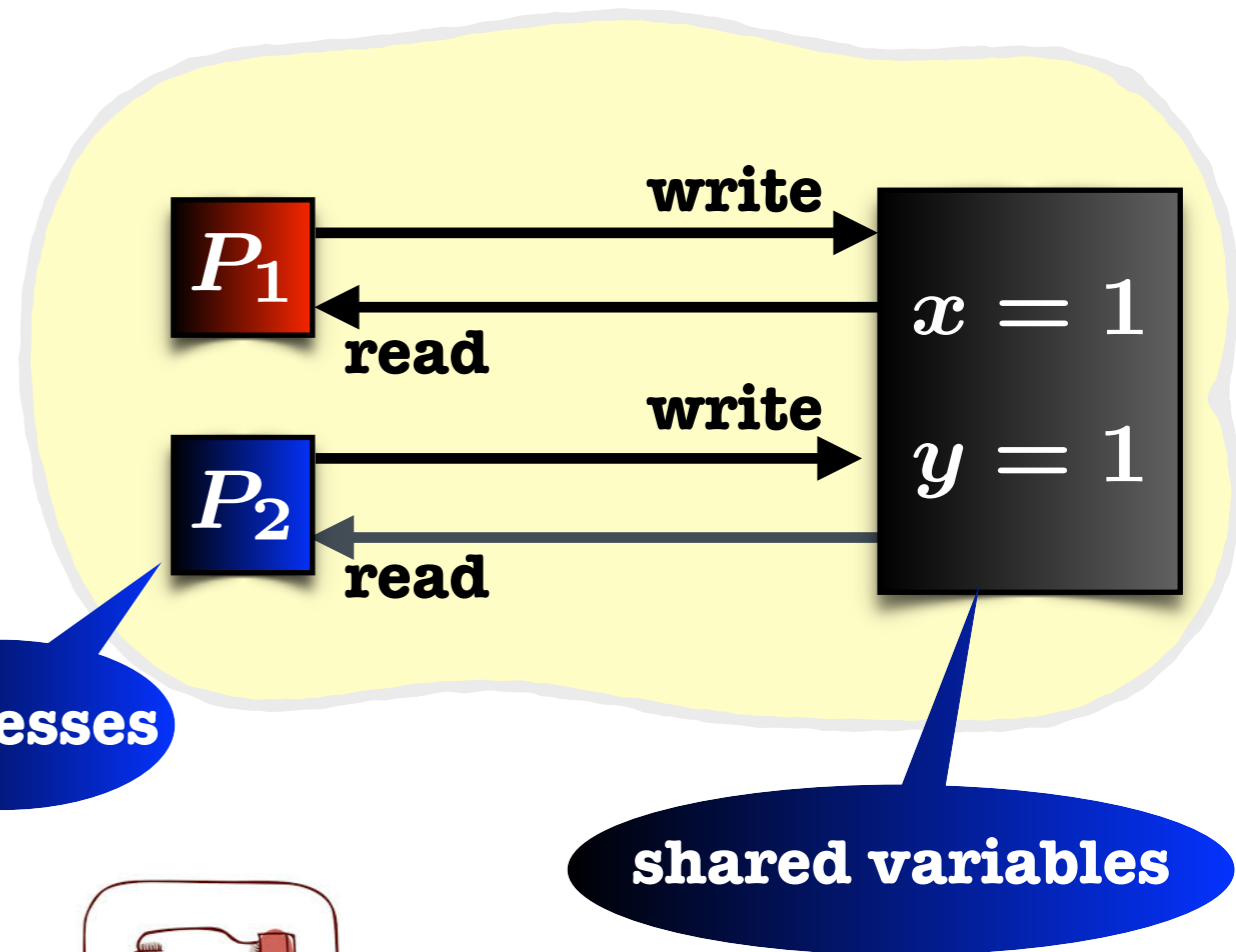
assertion

- 
- simple
 - intuitive

- 
- costly
 - unrealistic

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write



Weakly Consistent Systems

- **Microprocessors:**
 - **TSO, POWER, ARM, ...**
- **Weak Cache Protocols:**
 - **TSO-CC, Racer, SISD, ...**
- **Programming Languages:**
 - **C11, Java, ...**
- **Distributed Data Stores:**
 - **Amazon, Facebook, Google, ...**

Weakly Consistent Systems

- **Microprocessors:**
 - **TSO, POWER, ARM, ...**
- **Weak Cache Protocols:**
 - **TSO-CC, Racer, SISD, ...**
- **Programming Languages:**
 - **C11, Java, ...**
- **Distributed Data Stores:**
 - **Amazon, Facebook, Google, ...**

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - Dynamic Partial-Order Reduction (DPOR)
 - Experiments
- **Conclusions**

Release-Acquire (R.A)

Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al , POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**

Release-Acquire (R.A)

Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al , POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**

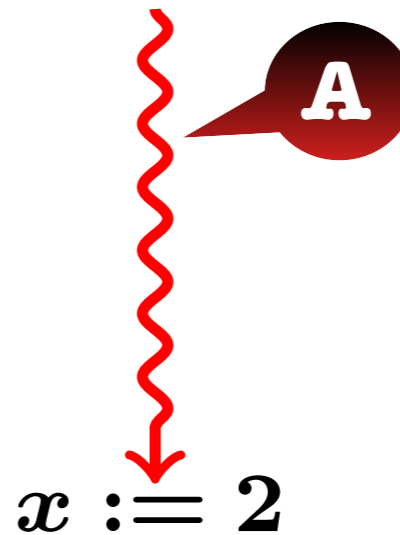
Release-Acquire (R.A)

Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al , POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**



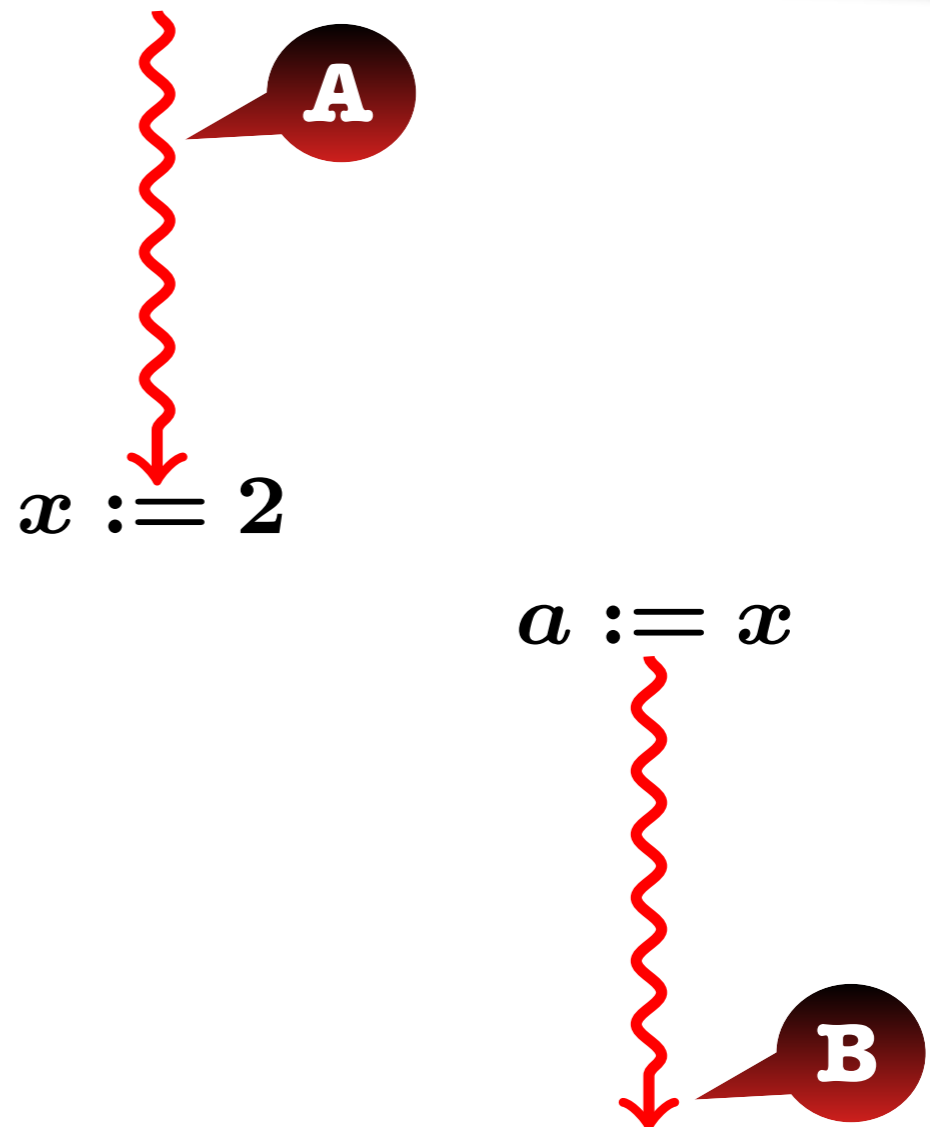
Release-Acquire (R.A)

Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al , POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**



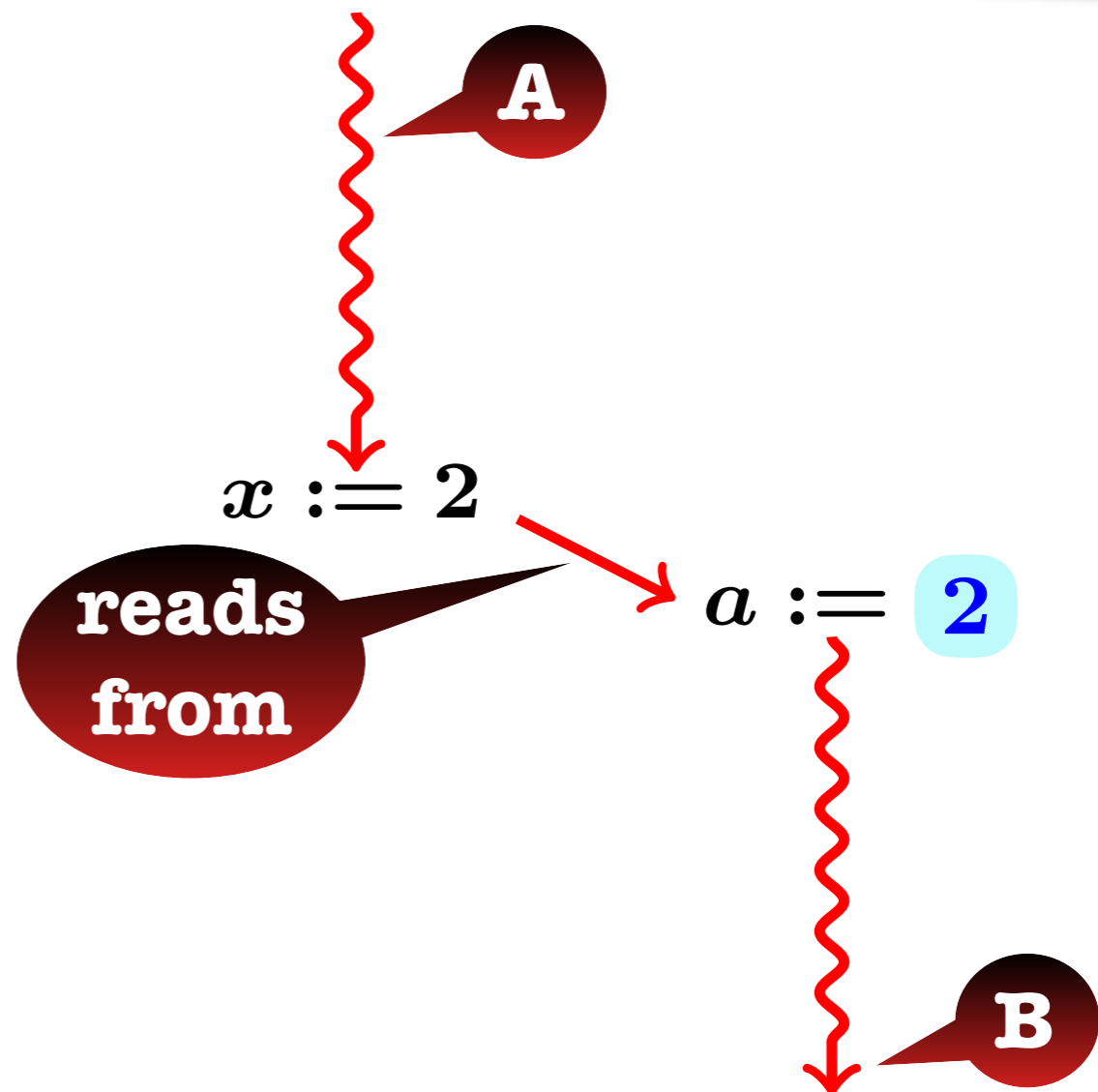
Release-Acquire (R.A)

Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al , POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**



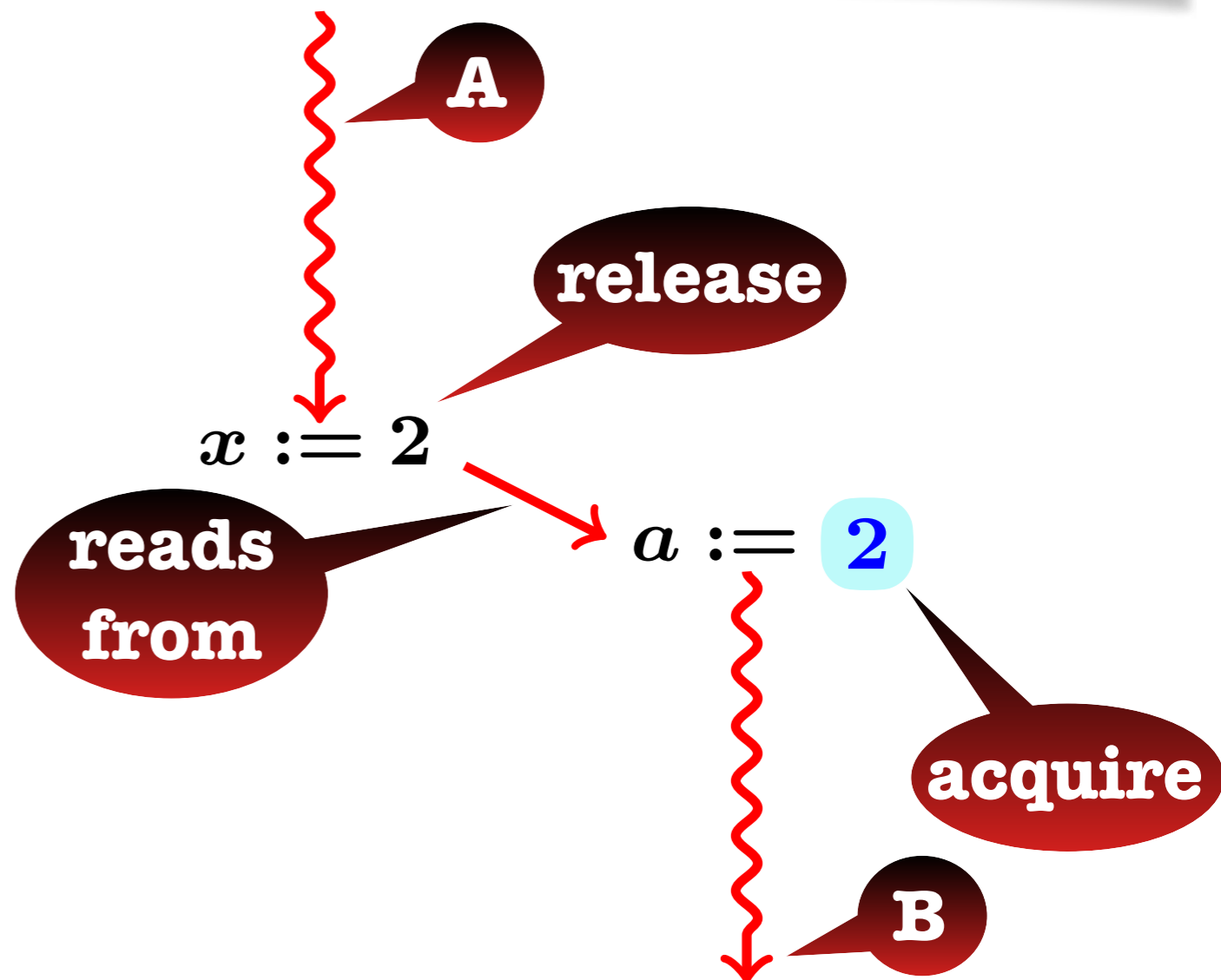
Release-Acquire (RA)

Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al, POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**



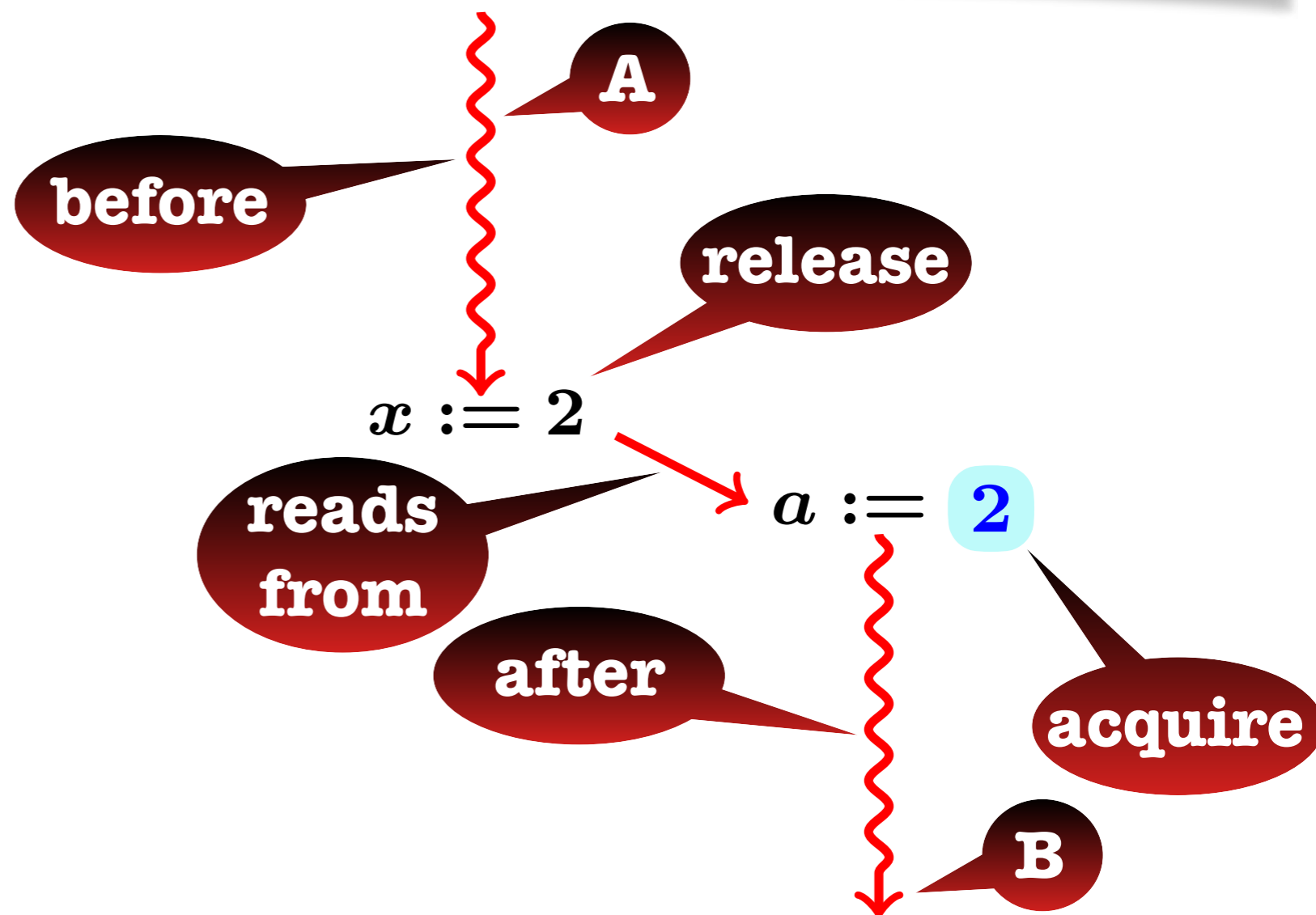
Release-Acquire (RA)

Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al, POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**



Release-Acquire (RA)

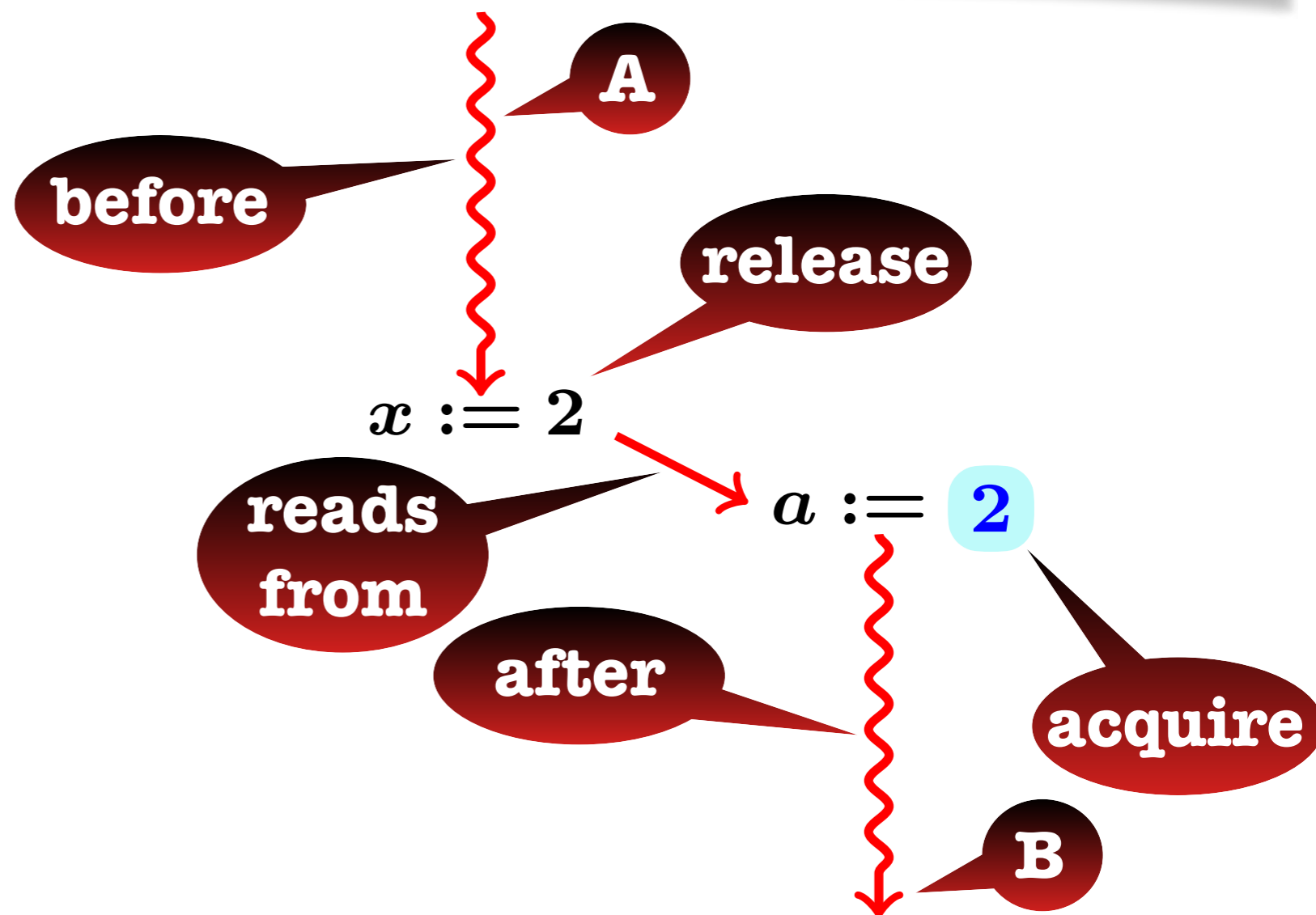
Motivation

- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al, POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

- Program order
- Synchronization
 - write: **release**
 - read: **acquire**

A happens before B



Release-Acquire (RA)

Motivation

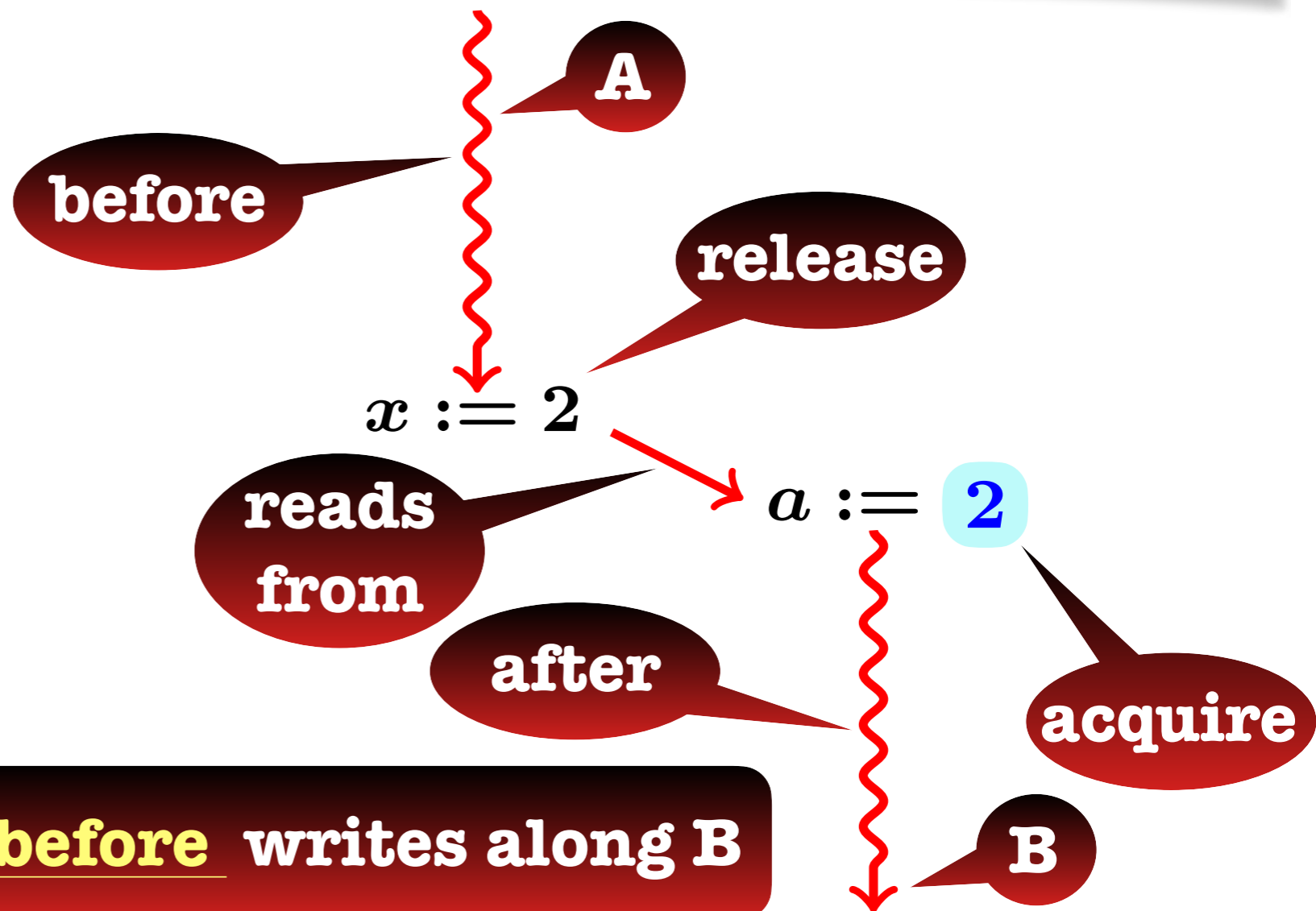
- Part of C11 semantics
- Good balance between performance and programmability [Lahav et al, POPL 2016]
 - (much) weaker than SC
 - sufficient for many concurrent algorithms
 - has verified compilation schemes to TSO and Power, ARM, etc.

Semantics

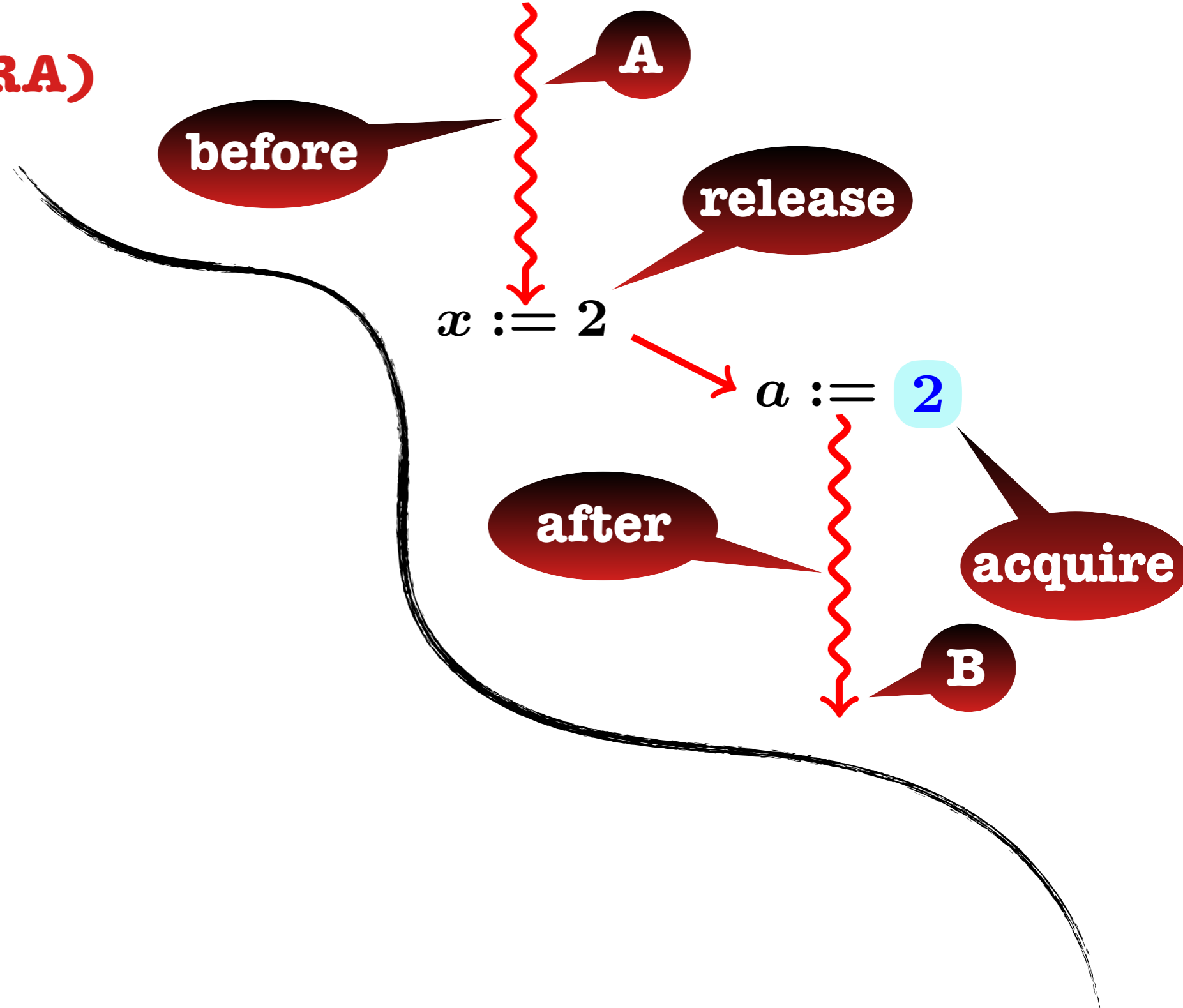
- Program order
- Synchronization
 - write: **release**
 - read: **acquire**

A **happens before** B

writes along A **happen before** writes along B



Release-Acquire (RA)



Release-Acquire (RA)

Operational Semantics

before

A

release

$x := 2$

$a := 2$

2

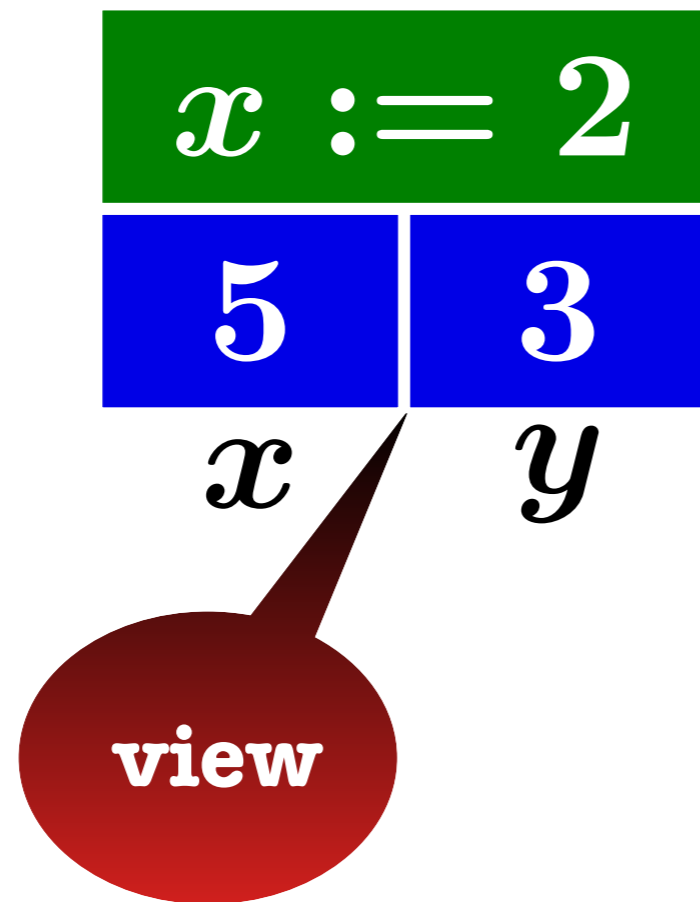
after

acquire

B

Release-Acquire (RA)

Operational Semantics



before

$x := 2$

release

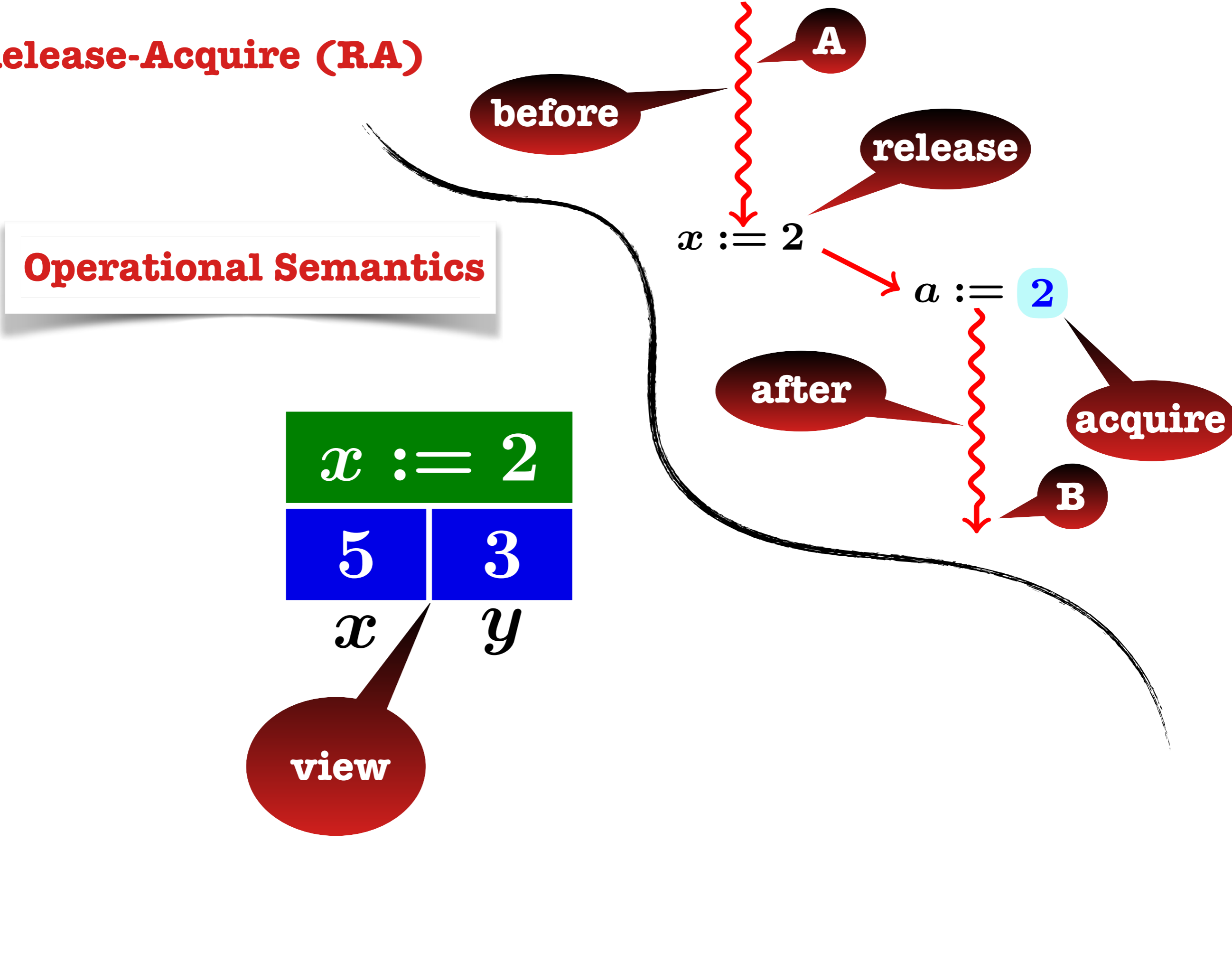
$a := 2$

after

acquire

B

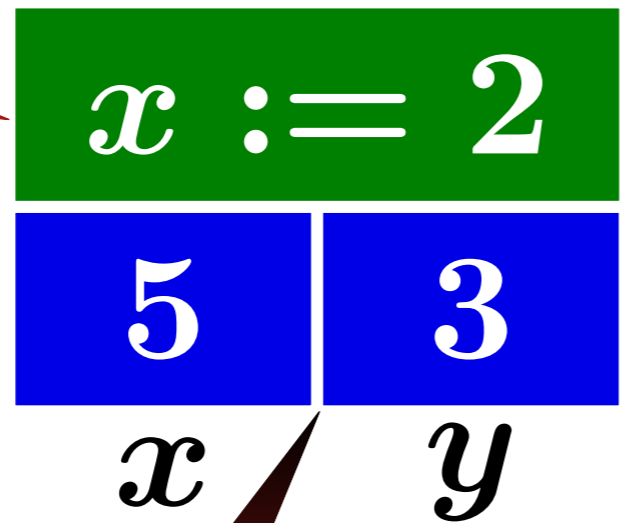
A



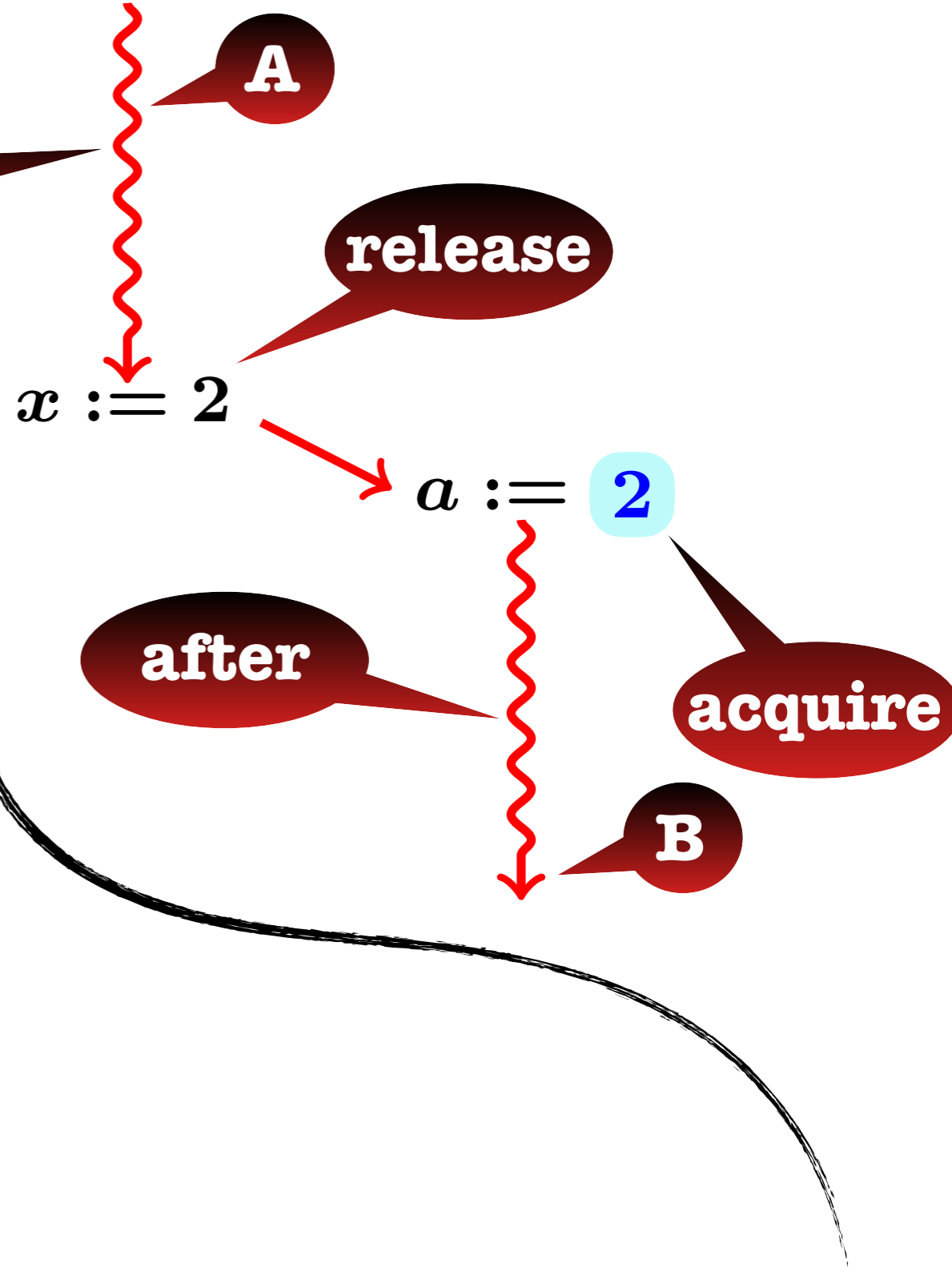
Release-Acquire (RA)

Operational Semantics

write operation

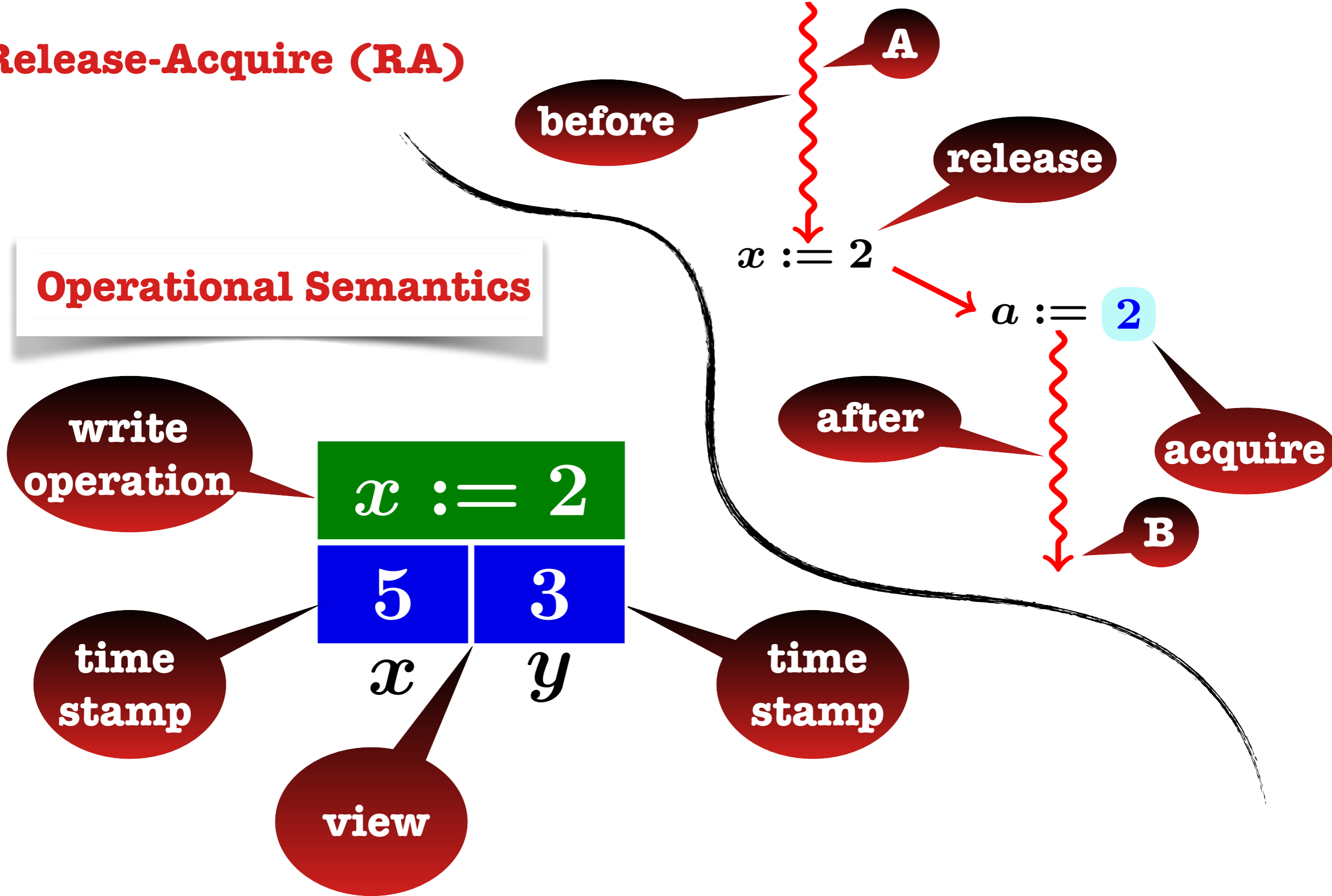


before



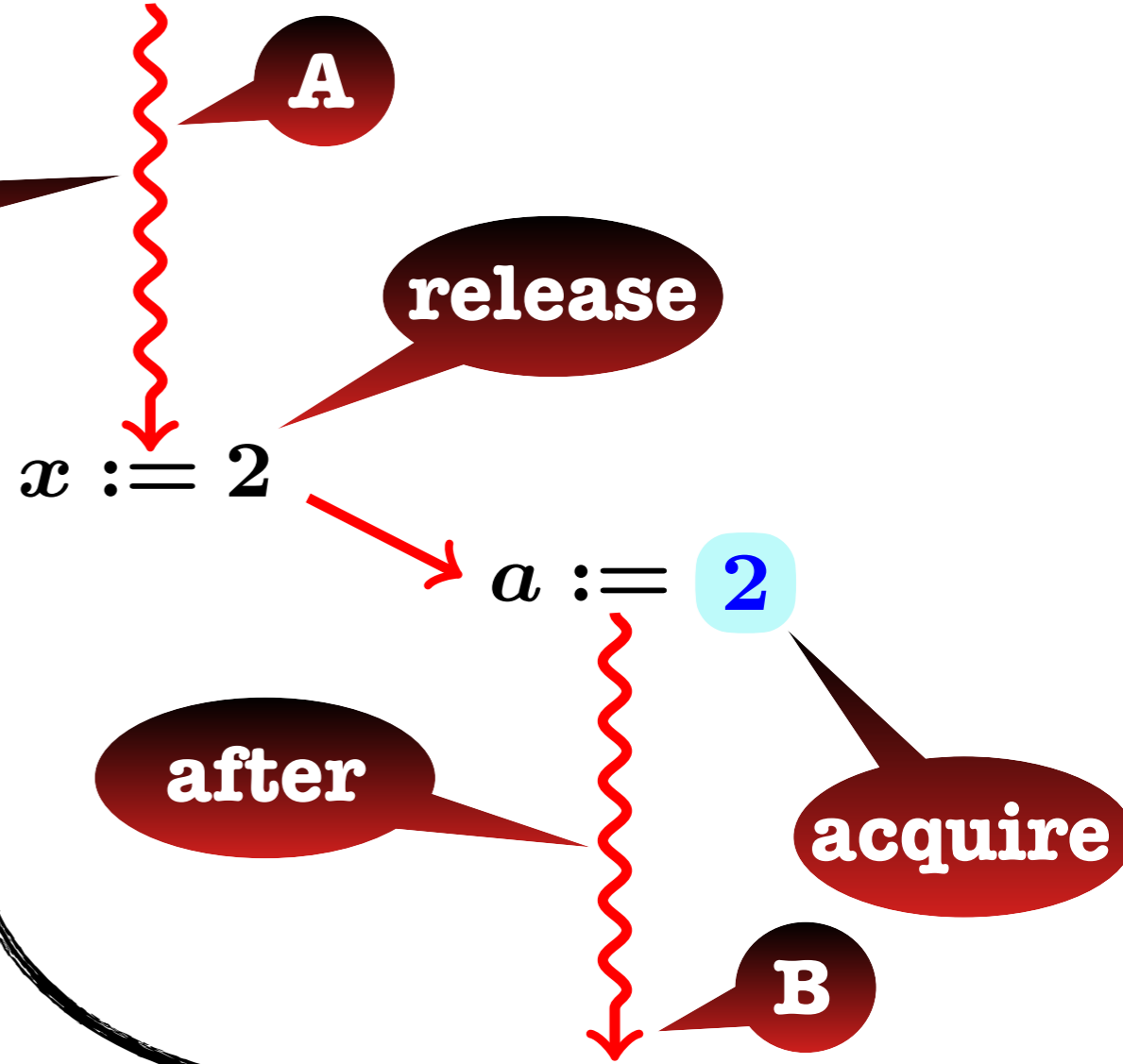
Release-Acquire (RA)

Operational Semantics



Release-Acquire (RA)

Operational Semantics



write operation

$x := 2$

5

3

x

y

time stamp

view

time stamp

logical clock

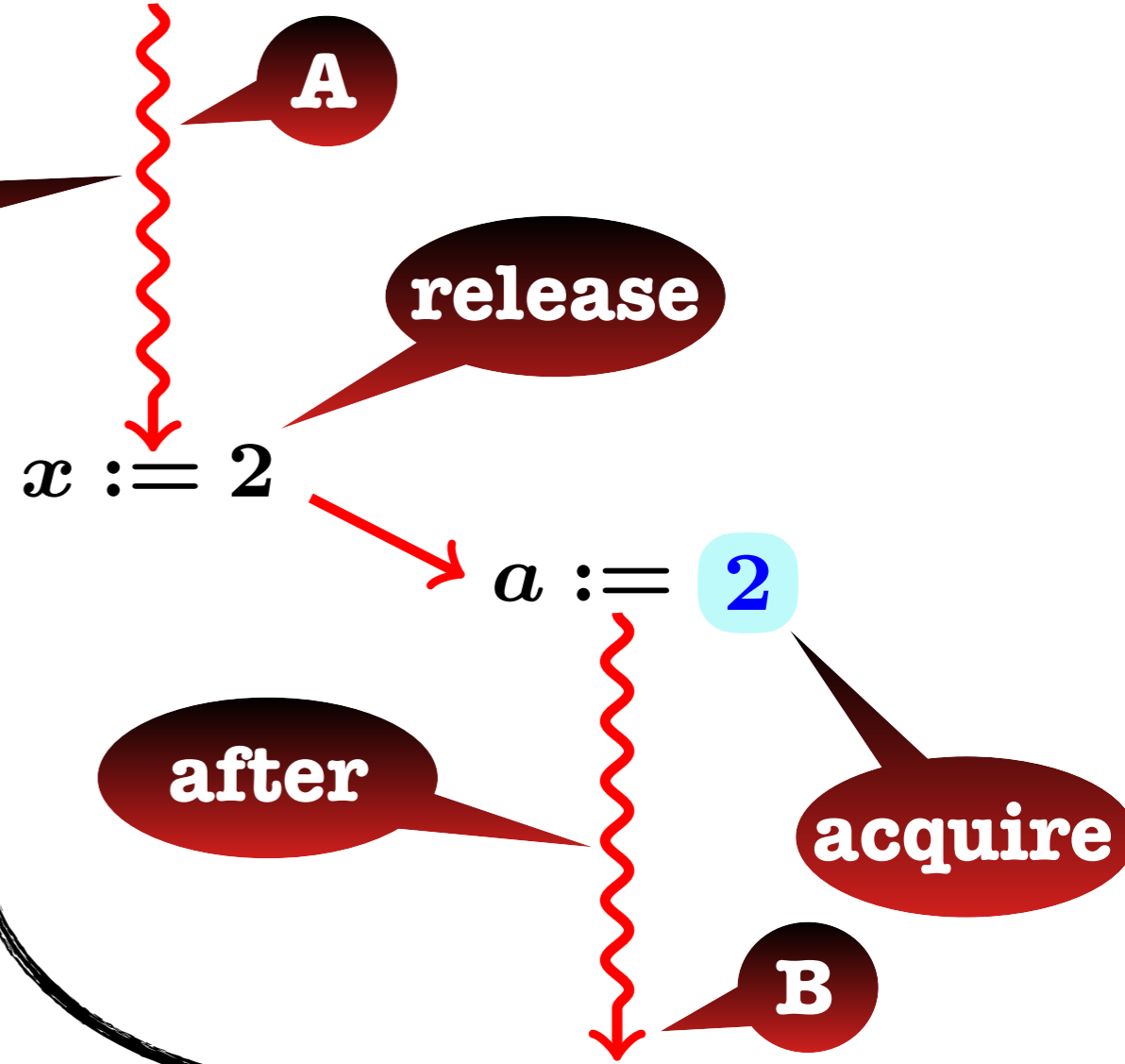
logical clock

acquire

B

Release-Acquire (RA)

Operational Semantics



write operation

$x := 2$

5 | 3

x

y

time stamp

view

logical clock

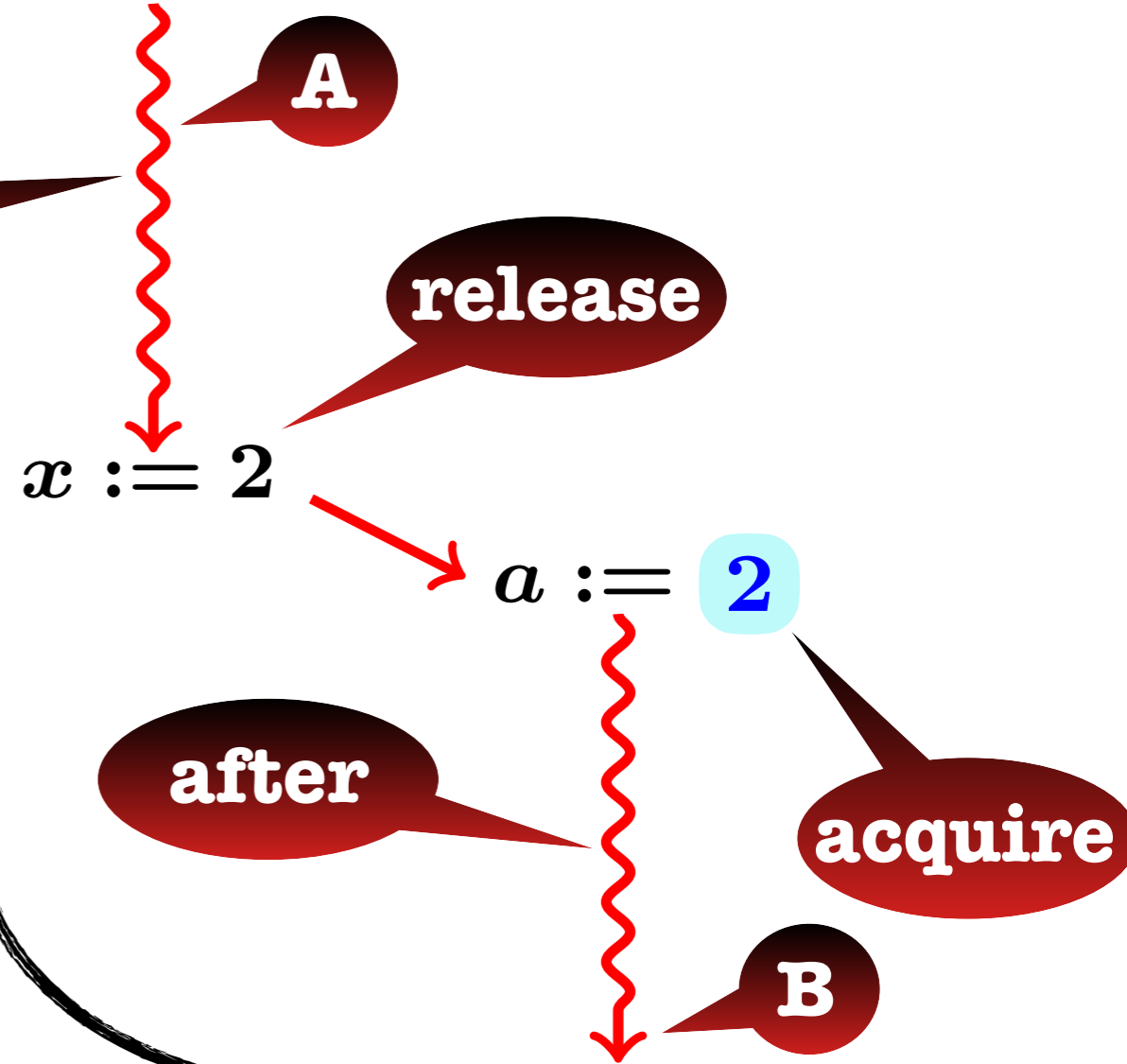
time stamp

logical clock

age of the latest write on x seen by the process

Release-Acquire (RA)

Operational Semantics



write operation

$x := 2$

5

3

x

y

time stamp

view

logical clock

time stamp

logical clock

acquire

age of the latest write on x seen by the process

age of the latest write on y seen by the process

Release-Acquire (RA)

Operational Semantics view

$x := 2$

5	3
---	---

x y

pool of views

memory

P_1

P_4

P_3

P_2

$x := 1$	
4	1

4	1
---	---

$x := 3$	
1	6

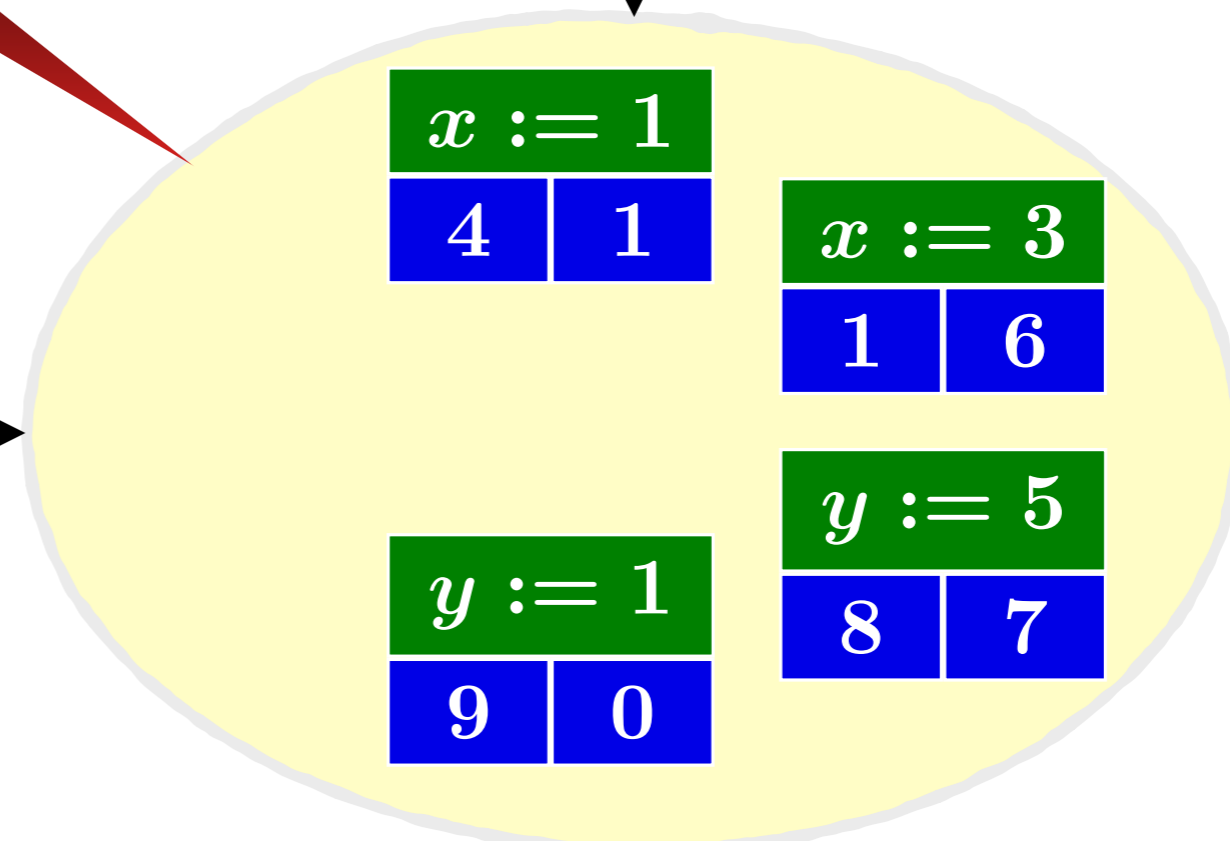
1	6
---	---

$y := 1$	
9	0

9	0
---	---

$y := 5$	
8	7

8	7
---	---



Release-Acquire (RA)

Operational Semantics view

$x := 2$	
5	3
x	y

pool of views

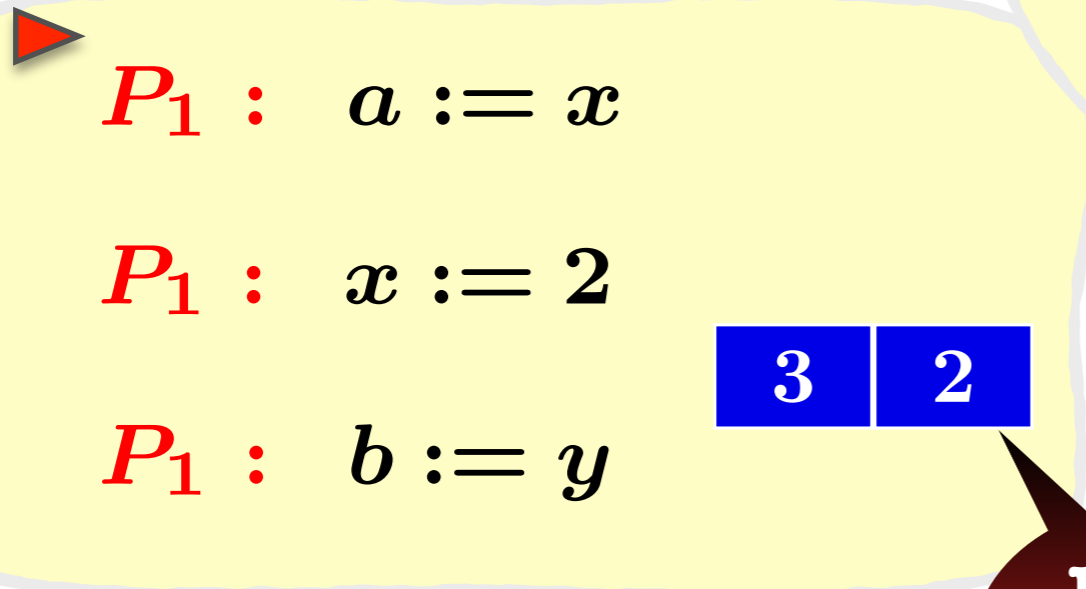
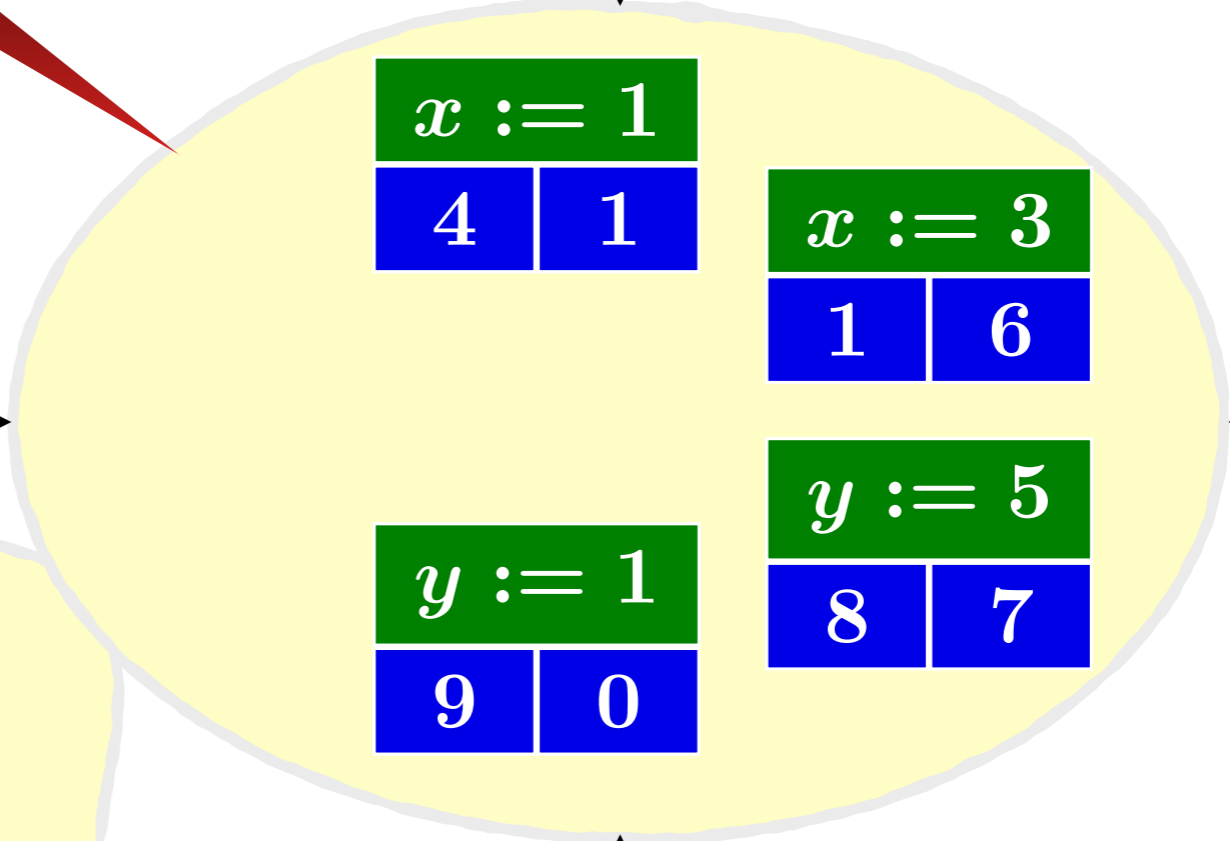
memory

P_1

P_4

P_3

P_2



local view

Release-Acquire (RA)

Operational Semantics view

$x := 2$	
5	3
x	y

pool of views

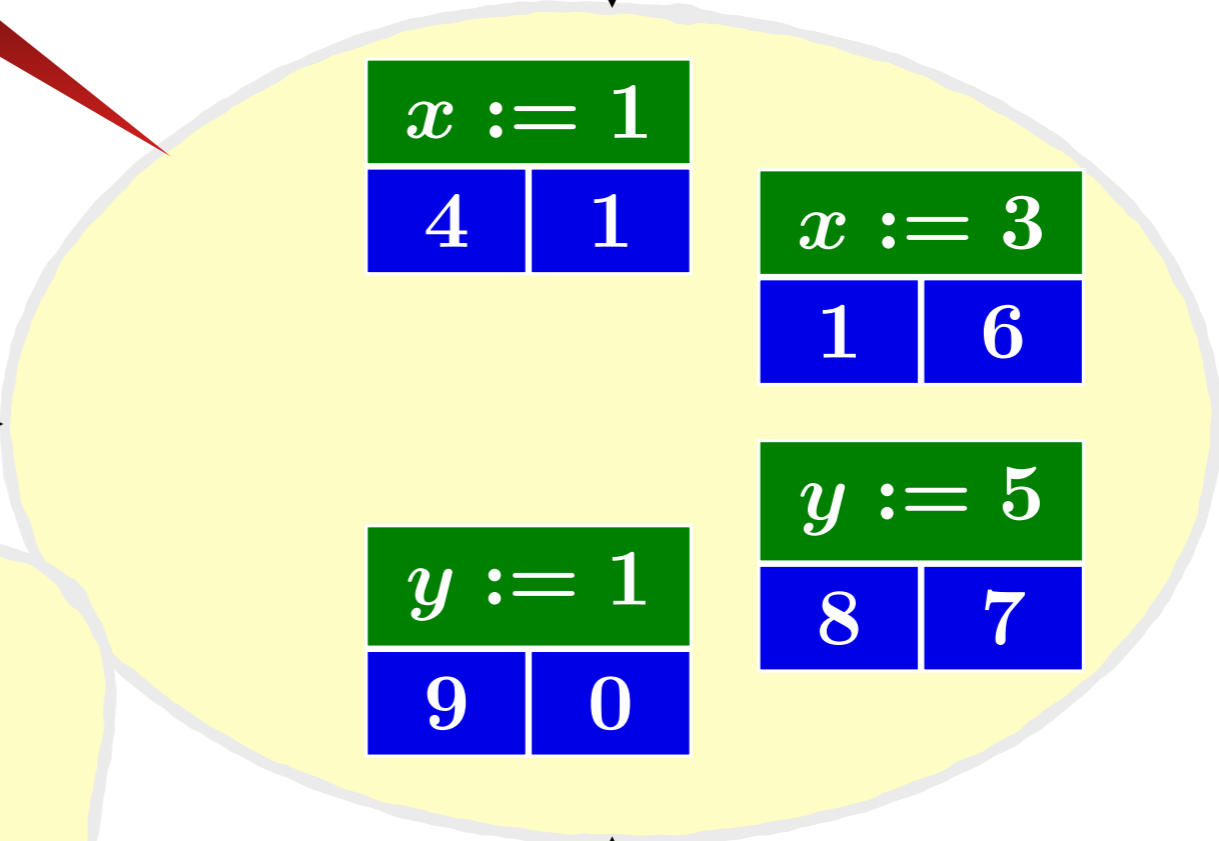
memory

P_1

P_4

P_3

P_2



$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

Release-Acquire (RA)

Operational Semantics

view

$x := 2$

5	3
---	---

x y

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

$x := 1$

4	1
---	---

$x := 3$

1	6
---	---

$y := 1$

9	0
---	---

$y := 5$

8	7
---	---

P_1

P_3

P_2

$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

Release-Acquire (RA)

Operational Semantics

view

$x := 2$

5	3
---	---

x y

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

$x := 1$

4	1
---	---

$x := 3$

1	6
---	---

$y := 5$

$y := 1$

9	0
---	---

8	7
---	---

P_1

P_3

P_2

$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

Release-Acquire (RA)

Operational Semantics view

$x := 2$	
5	3
x	y

pool of views

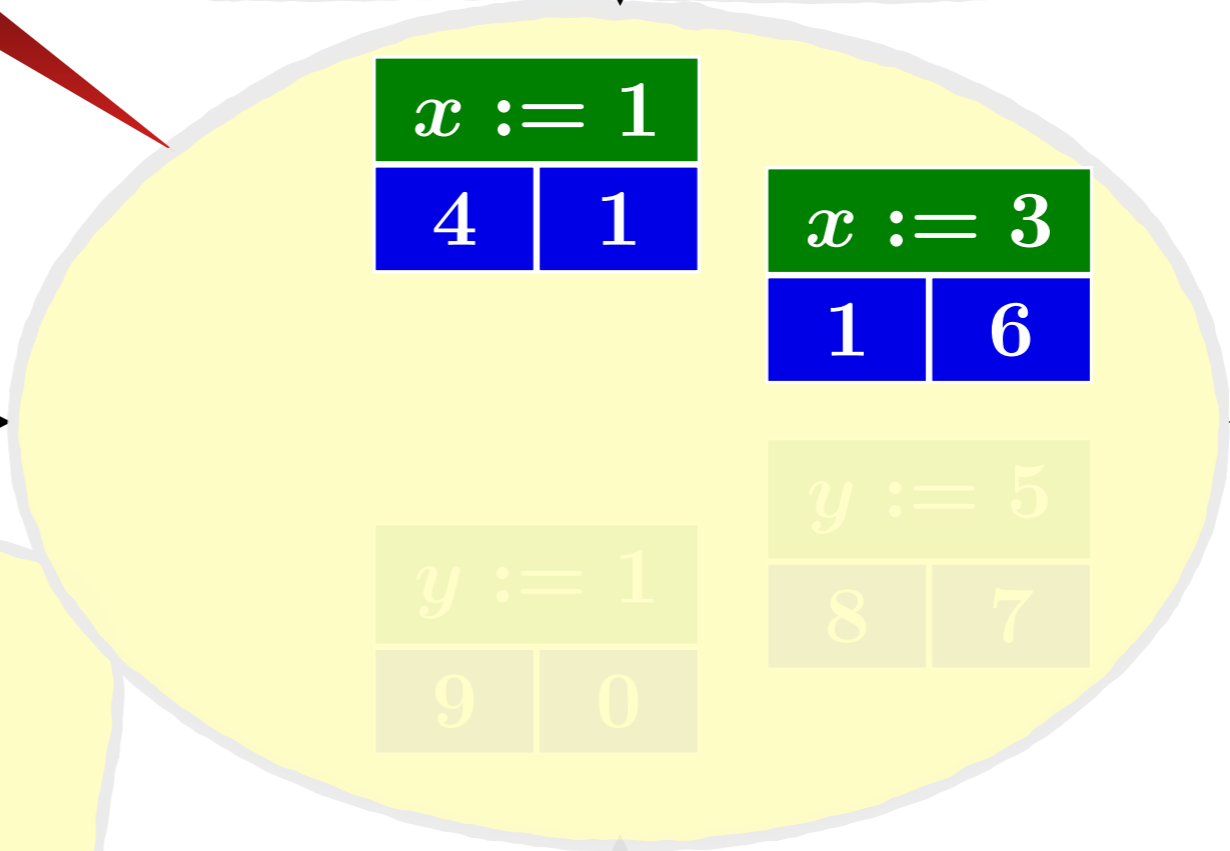
memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3



$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

P_2

Release-Acquire (RA)

Operational Semantics

view

$x := 2$

5	3
---	---

x y

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

$x := 1$

4	1
---	---

$x := 3$

1	6
---	---

P_1

P_3

$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

P_2

$y := 1$

9	0
---	---

$y := 5$

8	7
---	---

Release-Acquire (RA)

Operational Semantics

view

$x := 2$	
5	3
x	y

pool of views

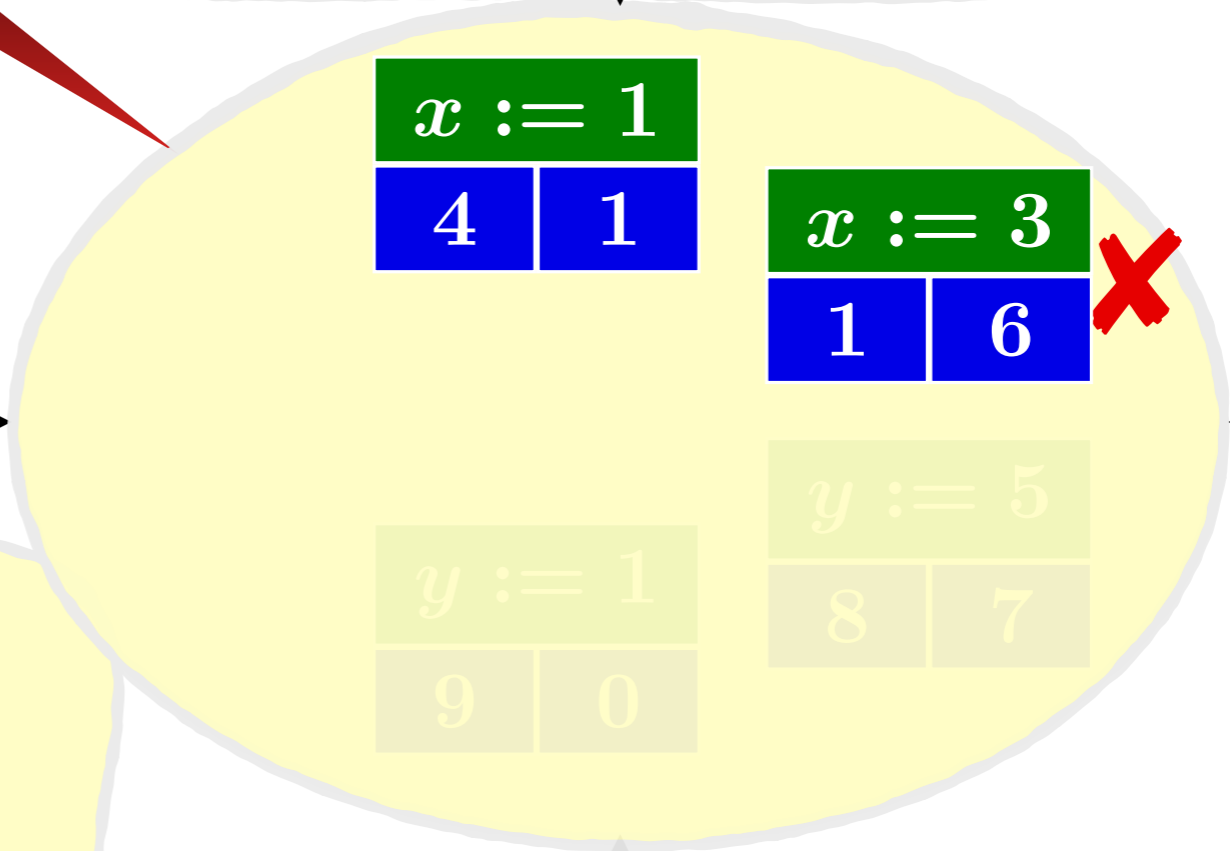
memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3



$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

P_2

Release-Acquire (RA)

Operational Semantics view

$x := 2$	
5	3
x	y

pool of views

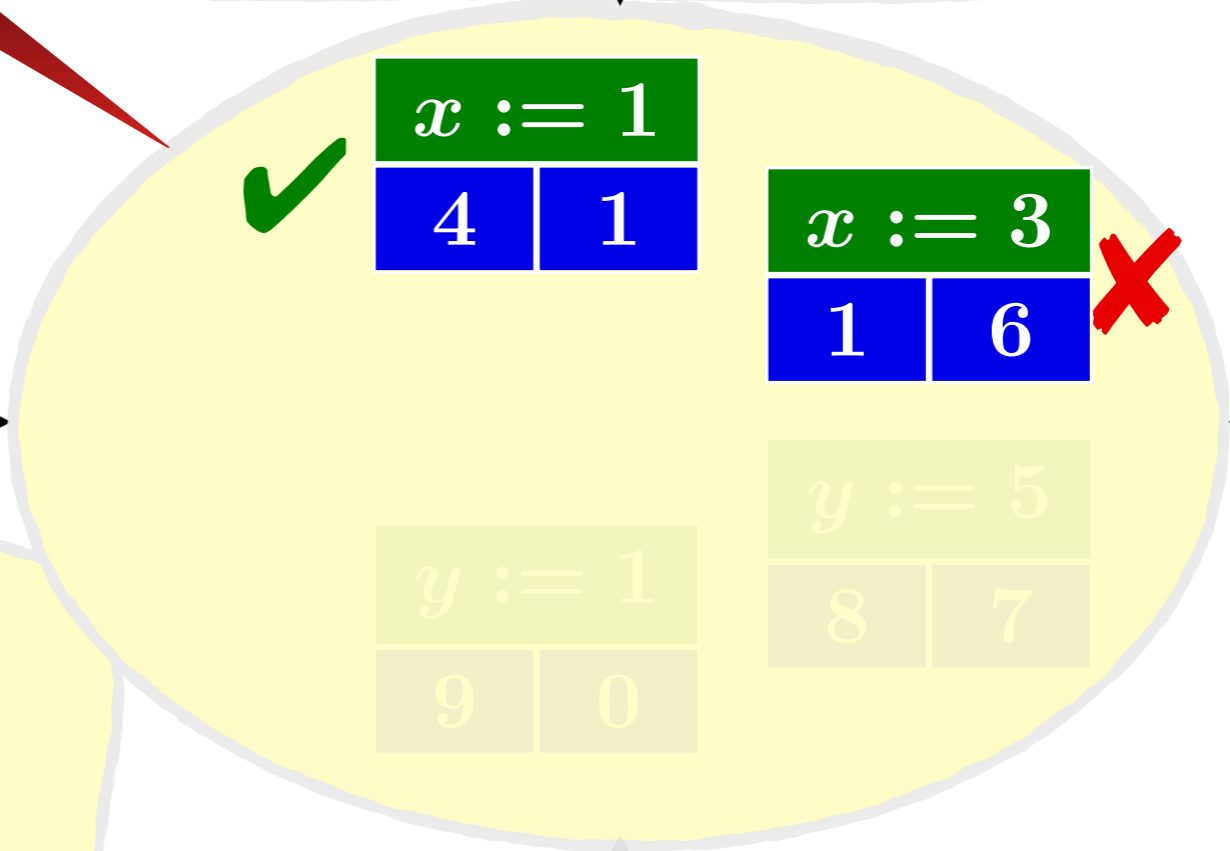
memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3



local view

P_2

$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3

P_2

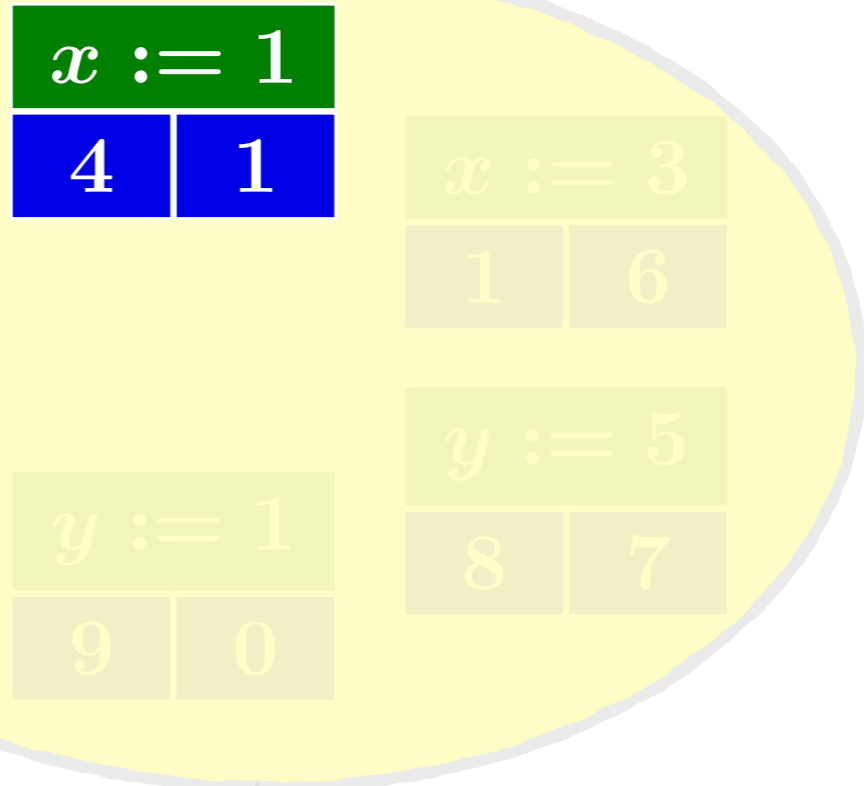
$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view



Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

$x := 1$	
4	1

$x := 3$	
1	6

$y := 5$	
8	7

$y := 1$	
9	0

P_3

$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

P_2

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

P_3

P_2

$x := 1$	
4	1

$x := 3$	
1	6

$y := 5$	
8	7

$y := 1$	
9	0

$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

P_3

$x := 1$	
4	1

$x := 3$	
1	6

$y := 5$	
8	7

$y := 1$	
3	0

max

$P_1 : a := x$

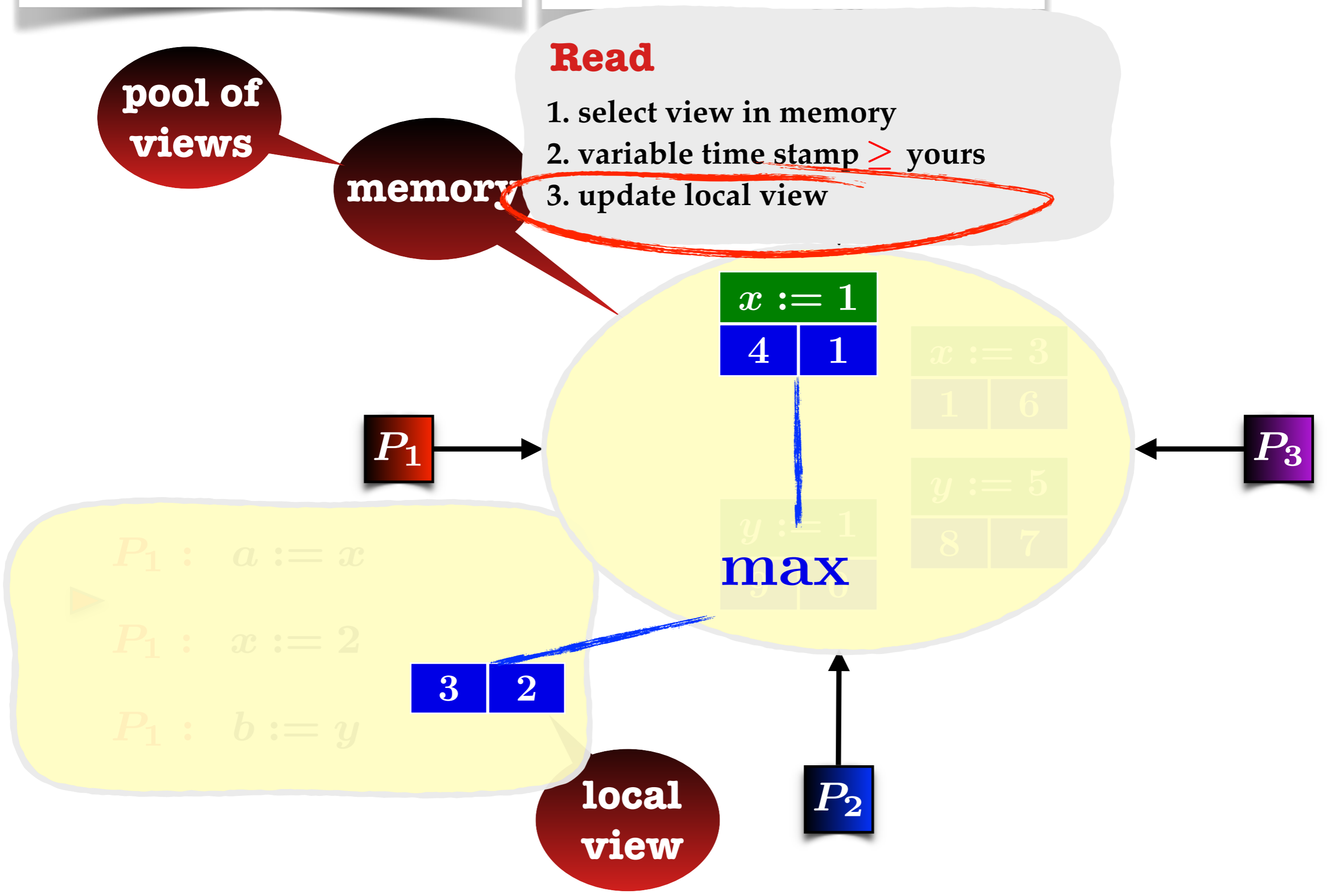
$P_1 : x := 2$

$P_1 : b := y$

3	2
---	---

local view

P_2



Release-Acquire (RA)

Operational Semantics

pool of views

memory

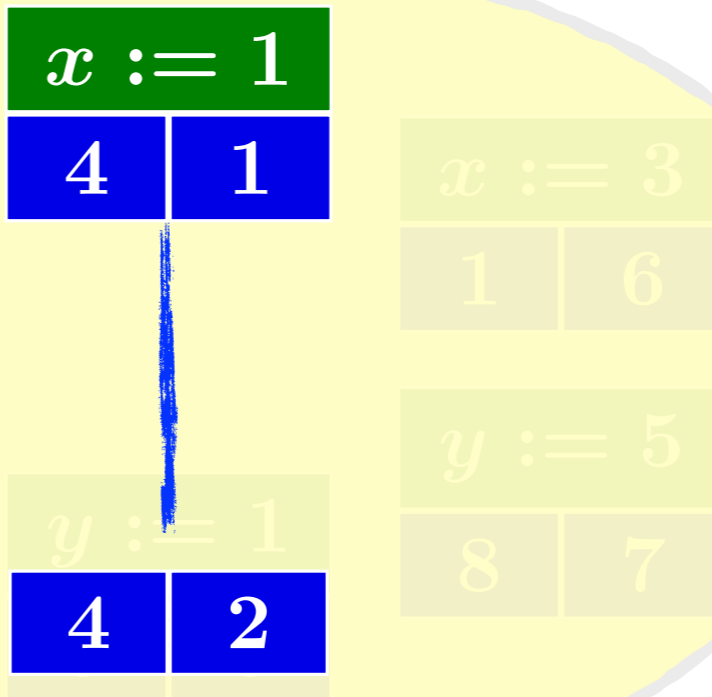
Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

P_3

P_2



$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

3 | 2

local view

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

P_3

$x := 1$	
4	1

$x := 3$	
1	6

$y := 5$	
8	7

$y := 1$	
9	6

max

$P_1 : a := x$

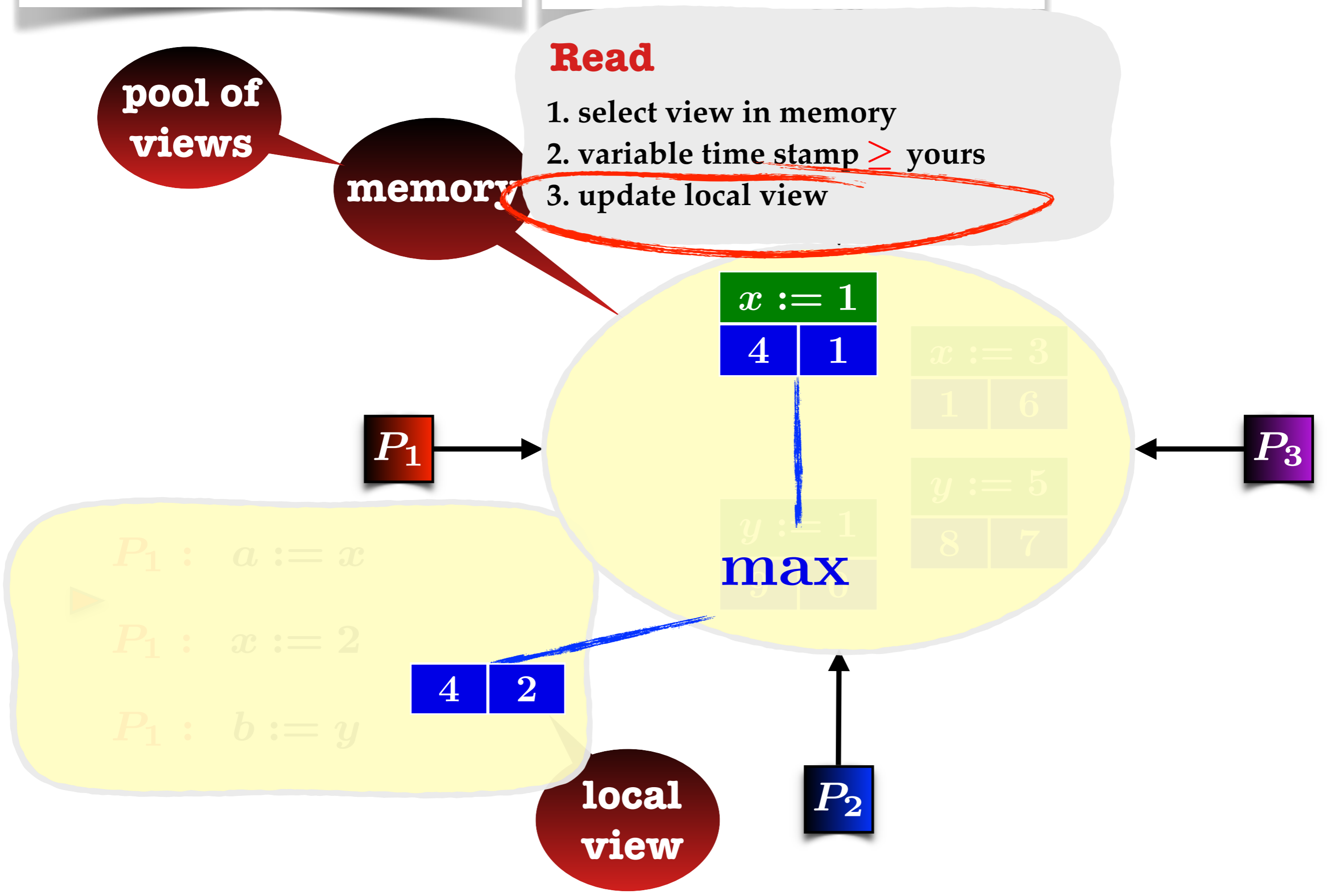
$P_1 : x := 2$

$P_1 : b := y$

4	2
---	---

local view

P_2



Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

P_3

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

$P_1 : a := x$

$P_1 : x := 2$

$P_1 : b := y$

4	2
---	---

local view

P_2

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

P_3

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

$P_1 : a := 1$

$P_1 : x := 2$

$P_1 : b := y$

4	2
---	---

local view

P_2

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp $>$ yours
3. update local view

P_1

P_3

$P_1 : a := 1$

$P_1 : x := 2$

$P_1 : b := y$

4	2
---	---

local view

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

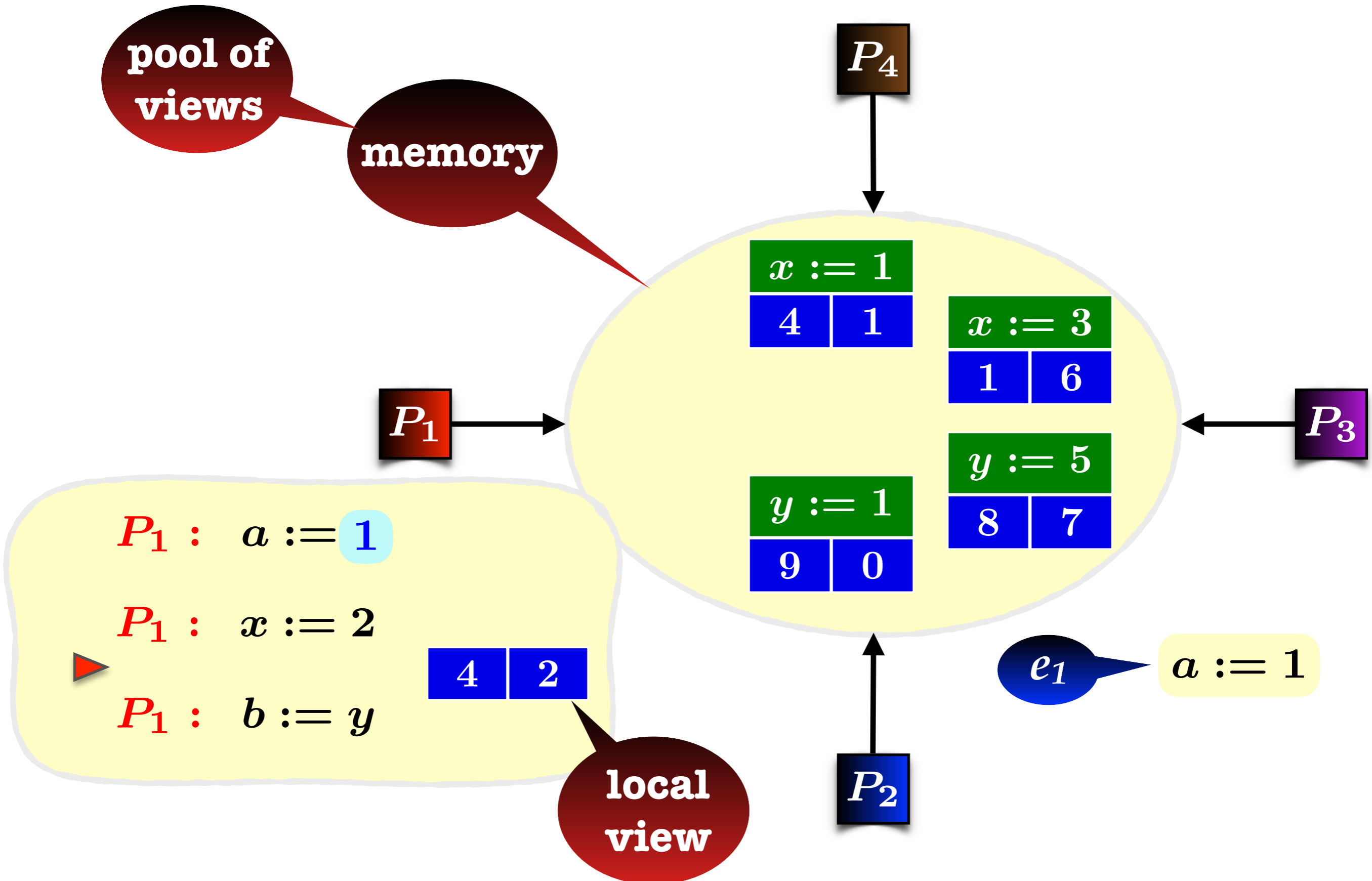
e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics



Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := y$

4	2
---	---

local view

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

$P_1 : a := 1$

$P_1 : x := 2$

$P_1 : b := y$

4	2
---	---

local view

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

$P_1 : a := 1$

$P_1 : x := 2$

$P_1 : b := y$

4	2
---	---

local view

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

$P_1 : a := 1$

$P_1 : x := 2$

$P_1 : b := y$

4 2

local view

$x := 1$
4 | 1

$x := 3$
1 | 6

$y := 1$
9 | 0

$y := 5$
8 | 7

e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := y$

$x := 2$	
5	2
4	2

local view

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

$P_1 : a := 1$
 $P_1 : x := 2$
▶ $P_1 : b := y$

$x := 2$	
5	2
5	2

local view

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := y$

$x := 2$	
5	2
5	2

local view

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

e_1

$a := 1$

P_2

Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

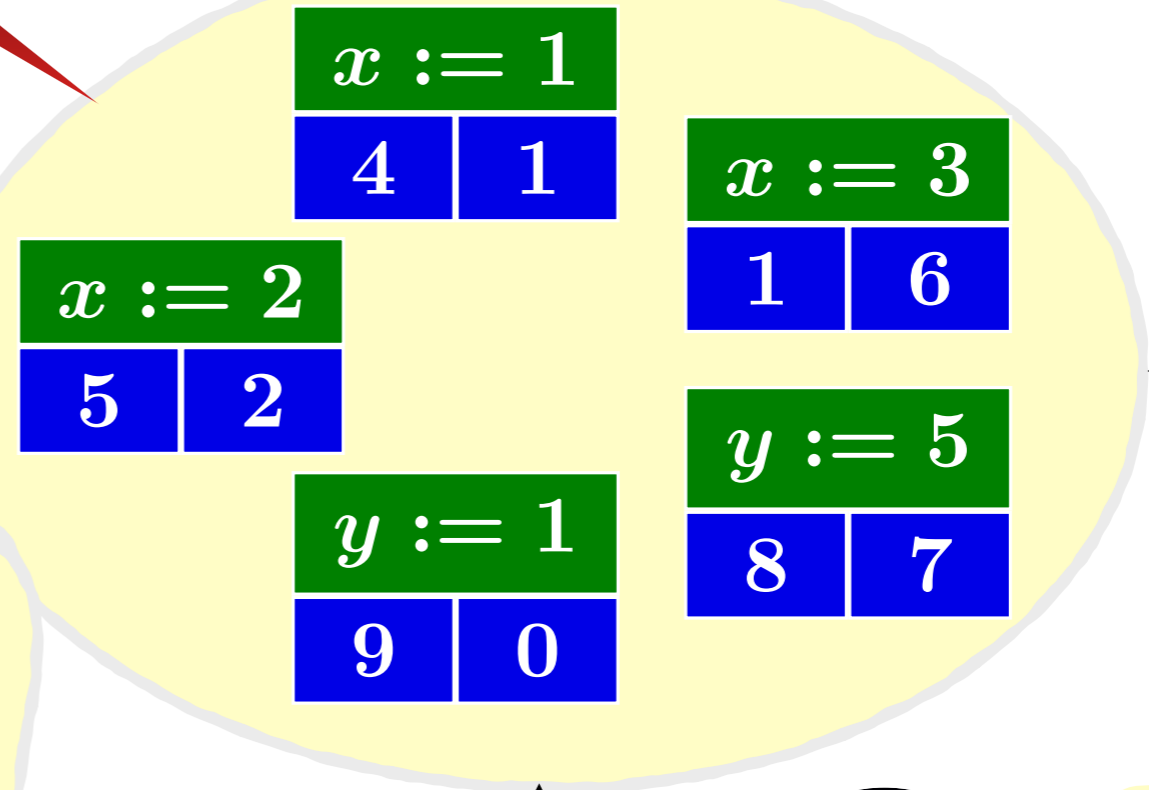
P_3

P_2

e_1

$a := 1$

local view



- $P_1 : a := 1$
- $P_1 : x := 2$
- $P_1 : b := y$



Release-Acquire (RA)

Operational Semantics

Write

1. create new local view
2. variable time stamp:
 - i. newer than yours
 - ii. not in memory
3. copy new view to memory

pool of views

memory

P_1

P_3

P_2

local view

$P_1 : a := 1$

$P_1 : x := 2$

$P_1 : b := y$

5 2

$x := 1$

4 1

$x := 2$

5 2

$y := 1$

9 0

$x := 3$

1 6

$y := 5$

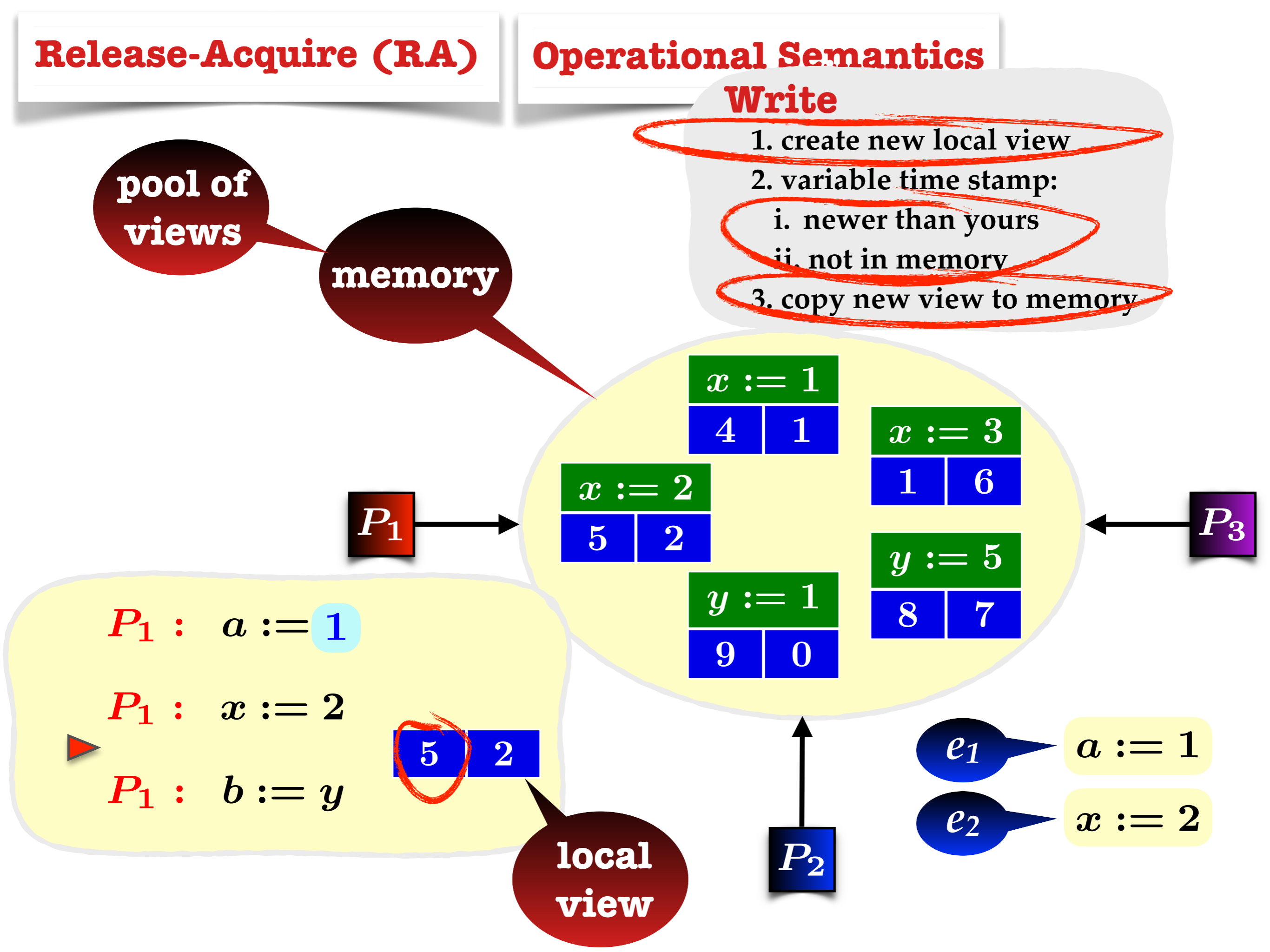
8 7

e_1

$a := 1$

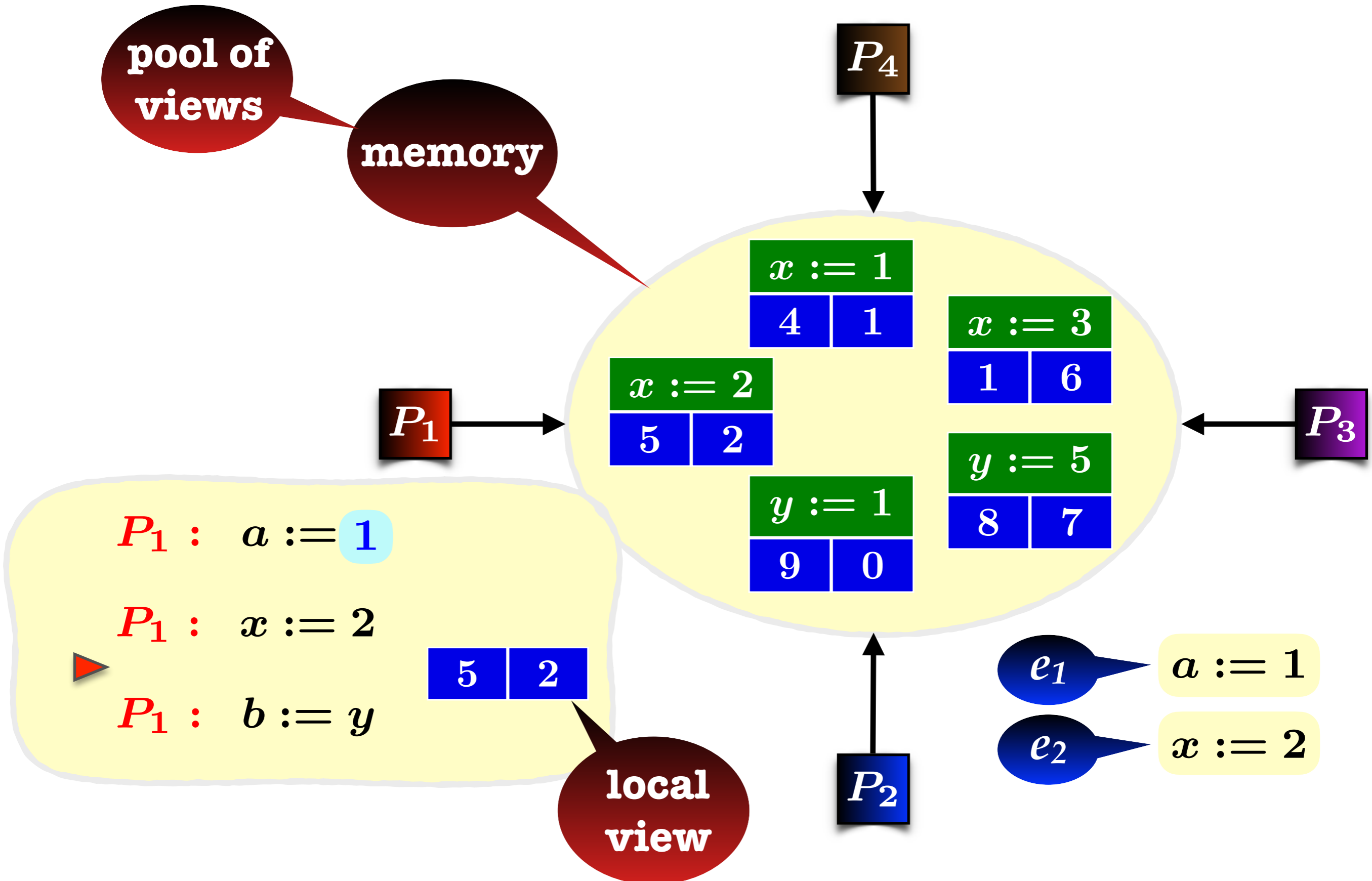
e_2

$x := 2$



Release-Acquire (RA)

Operational Semantics



Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

$x := 2$	
5	2

$x := 1$	
4	1

$x := 3$	
1	6

$y := 1$	
9	0

$y := 5$	
8	7

P_3

$P_1 : a := 1$

$P_1 : x := 2$

$P_1 : b := y$

5	2
---	---

local view

P_2

e_1

$a := 1$

e_2

$x := 2$

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3

$y := 1$	
9	0

$y := 5$	
8	7

5	2
---	---

local view

P_2

e_1

$a := 1$

e_2

$x := 2$

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3

X

$y := 1$	
9	0

$y := 5$	
8	7

5	2
---	---

local view

P_2

e_1

$a := 1$

e_2

$x := 2$

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3

\times

$y := 1$	
9	0

$y := 5$	
8	7

\checkmark

5	2
---	---

local view

P_2

e_1

$a := 1$

e_2

$x := 2$

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3

$y := 5$	
8	7



5	2
---	---

local view

P_2

e_1

$a := 1$

e_2

$x := 2$

Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3

$y := 5$
8 | 7

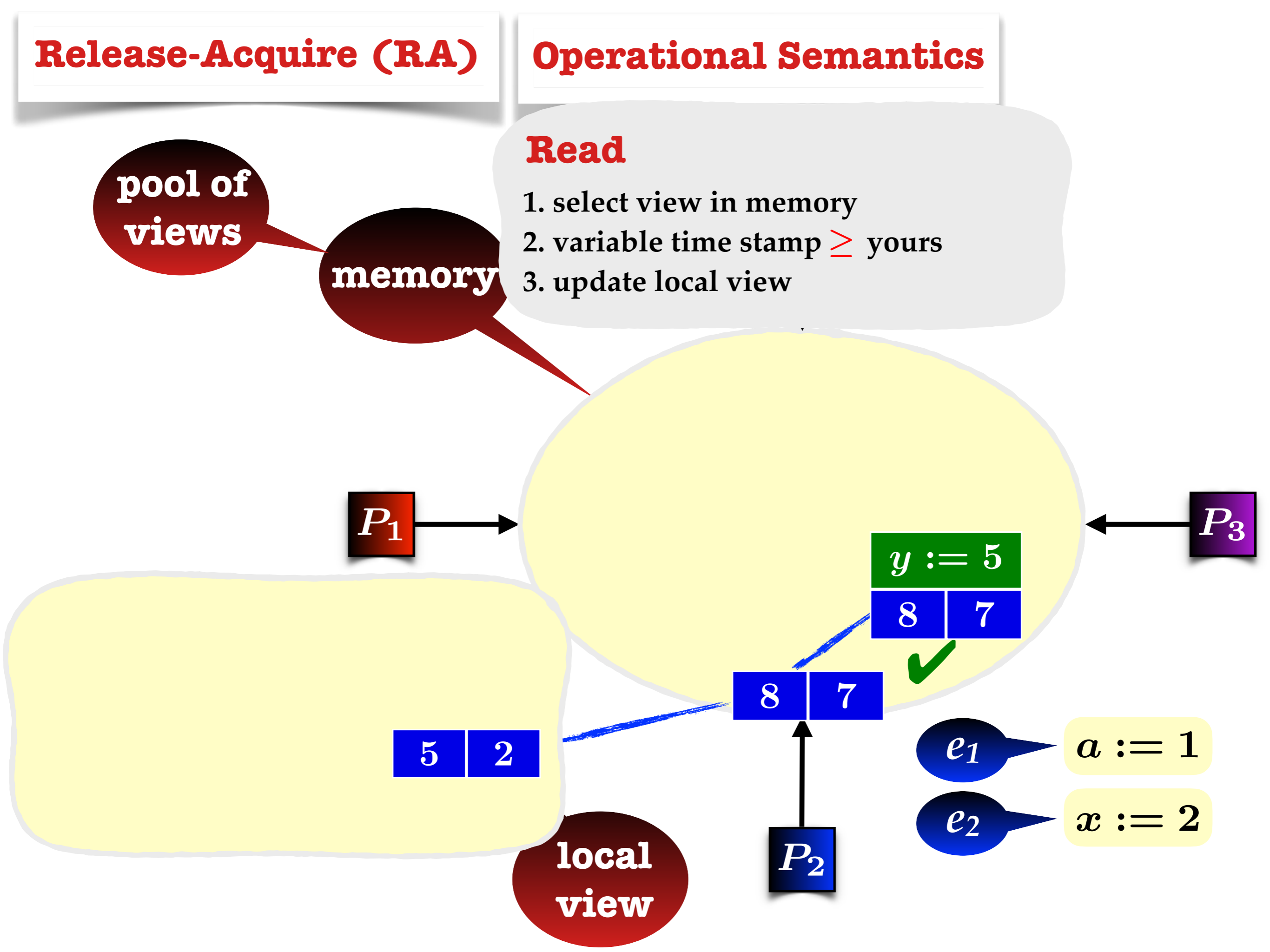
8 | 7

5 | 2

local view

P_2

e_1 $a := 1$
 e_2 $x := 2$



Release-Acquire (RA)

Operational Semantics

pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1

P_3

$y := 5$	
8	7



8	7
---	---

local view

P_2

e_1 $a := 1$

e_2 $x := 2$

Release-Acquire (RA)

Operational Semantics

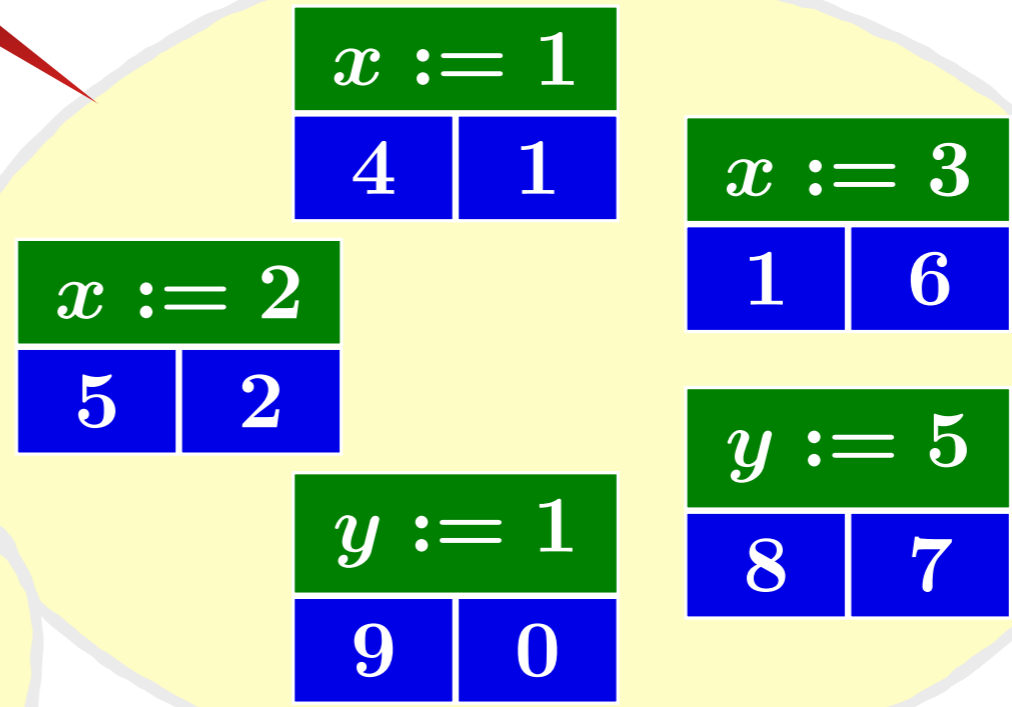
pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1



P_3

P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := y$



local view

P_2

e_1 : $a := 1$

e_2 : $x := 2$

Release-Acquire (RA)

Operational Semantics

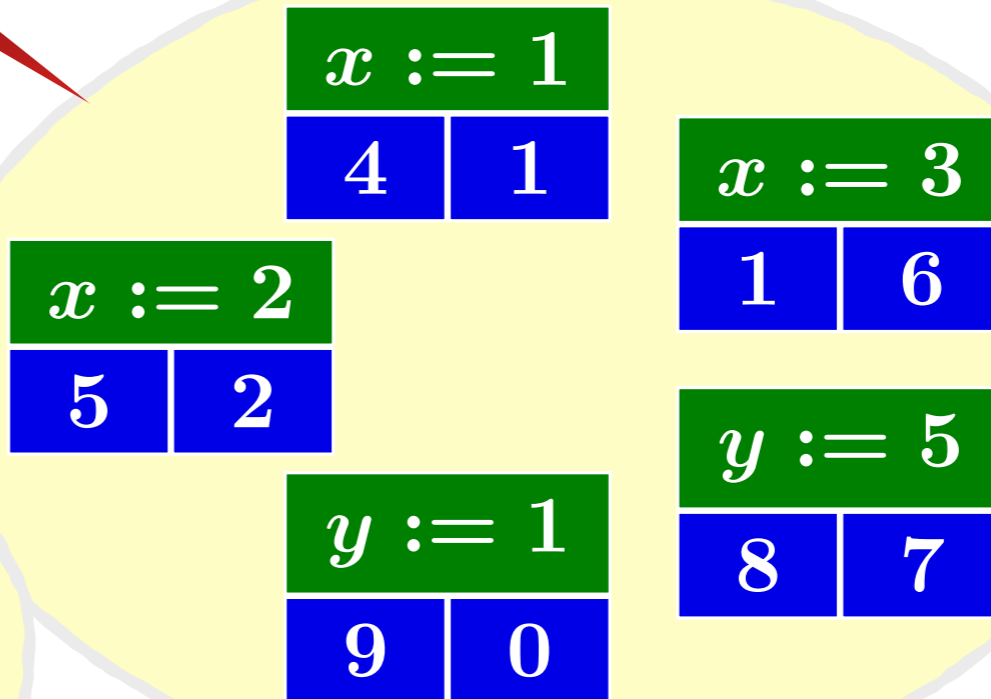
pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1



P_3

P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := 5$



local view

P_2

e_1

$a := 1$

e_2

$x := 2$

Release-Acquire (RA)

Operational Semantics

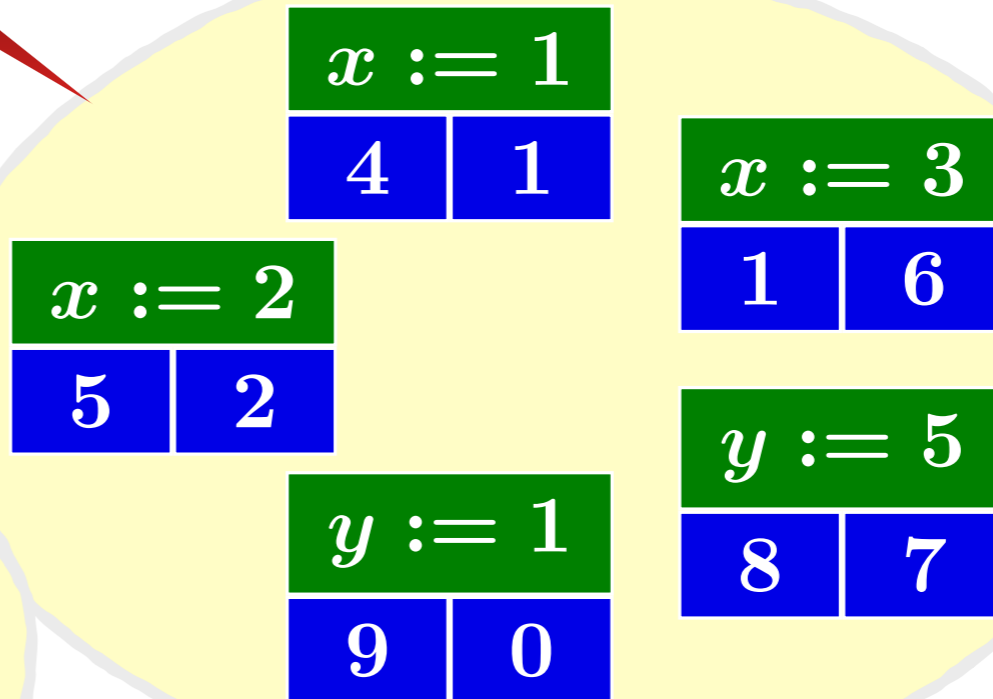
pool of views

memory

Read

1. select view in memory
2. variable time stamp \geq yours
3. update local view

P_1



P_3

P_1 : $a := 1$

P_1 : $x := 2$

P_1 : $b := 5$



local view

P_2

e_1

$a := 1$

e_2

$x := 2$


e_3

$b := 5$

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- Verification
 - Dynamic Partial-Order Reduction (DPOR)
 - Experiments
- Conclusions

Outline

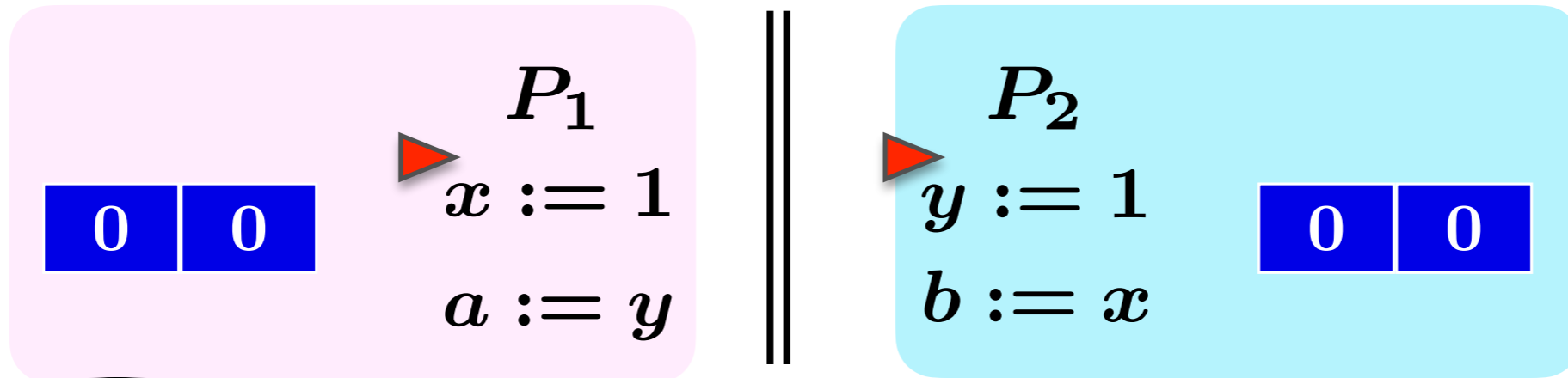
- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)** 
- Verification
 - Dynamic Partial-Order Reduction (DPOR)
 - Experiments
- Conclusions

Release-Acquire (RA)

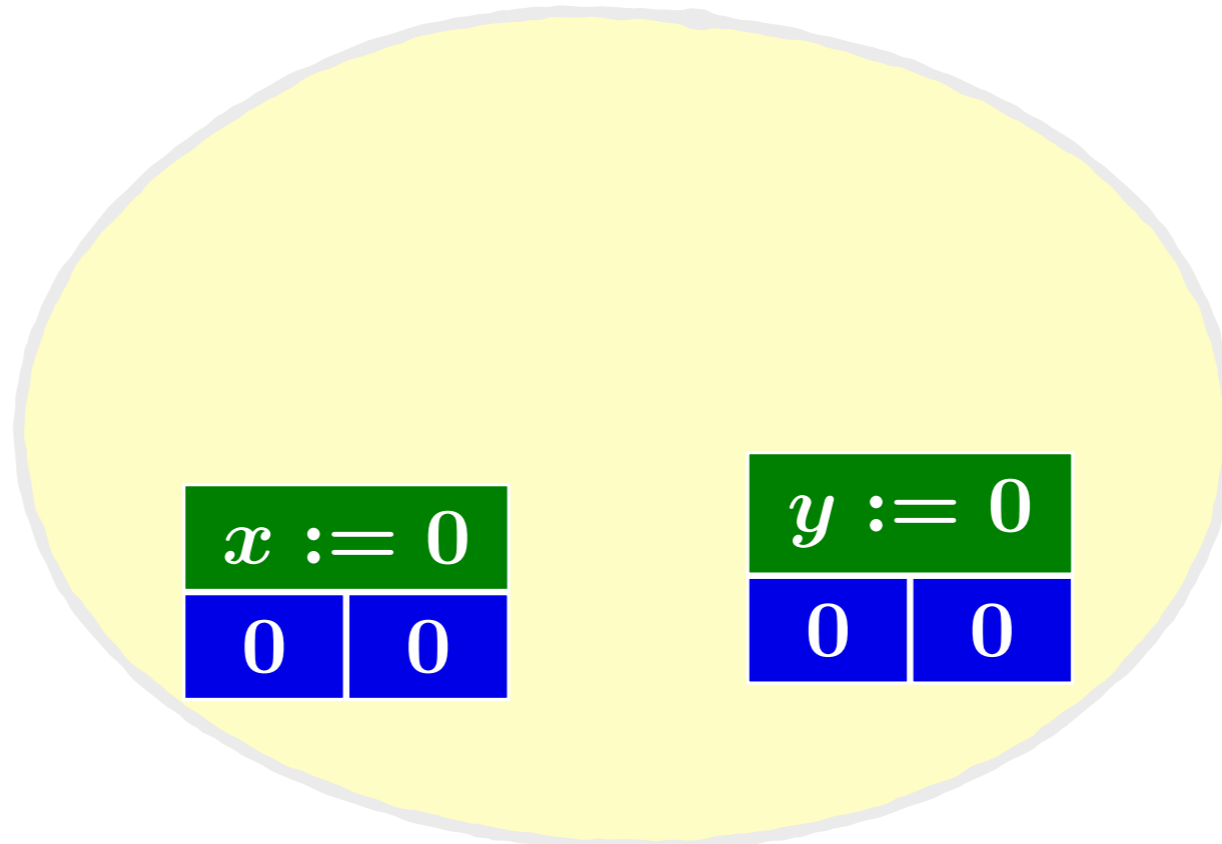
Dekker's protocol

Store Buffering (SB)

Initially: $x = y = 0$



assertion $(a = 0) \wedge (b = 0)?$

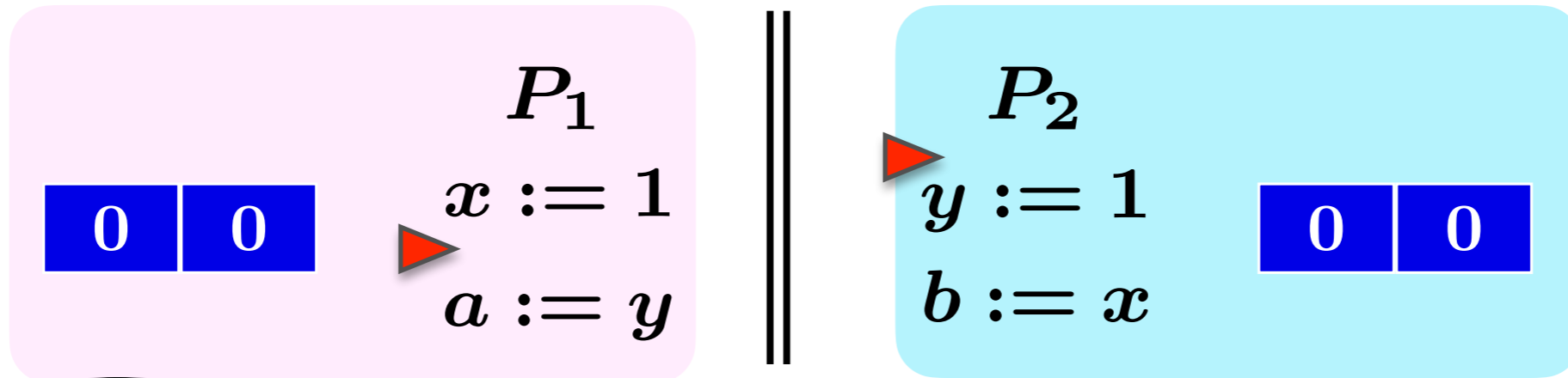


Release-Acquire (RA)

Dekker's
protocol

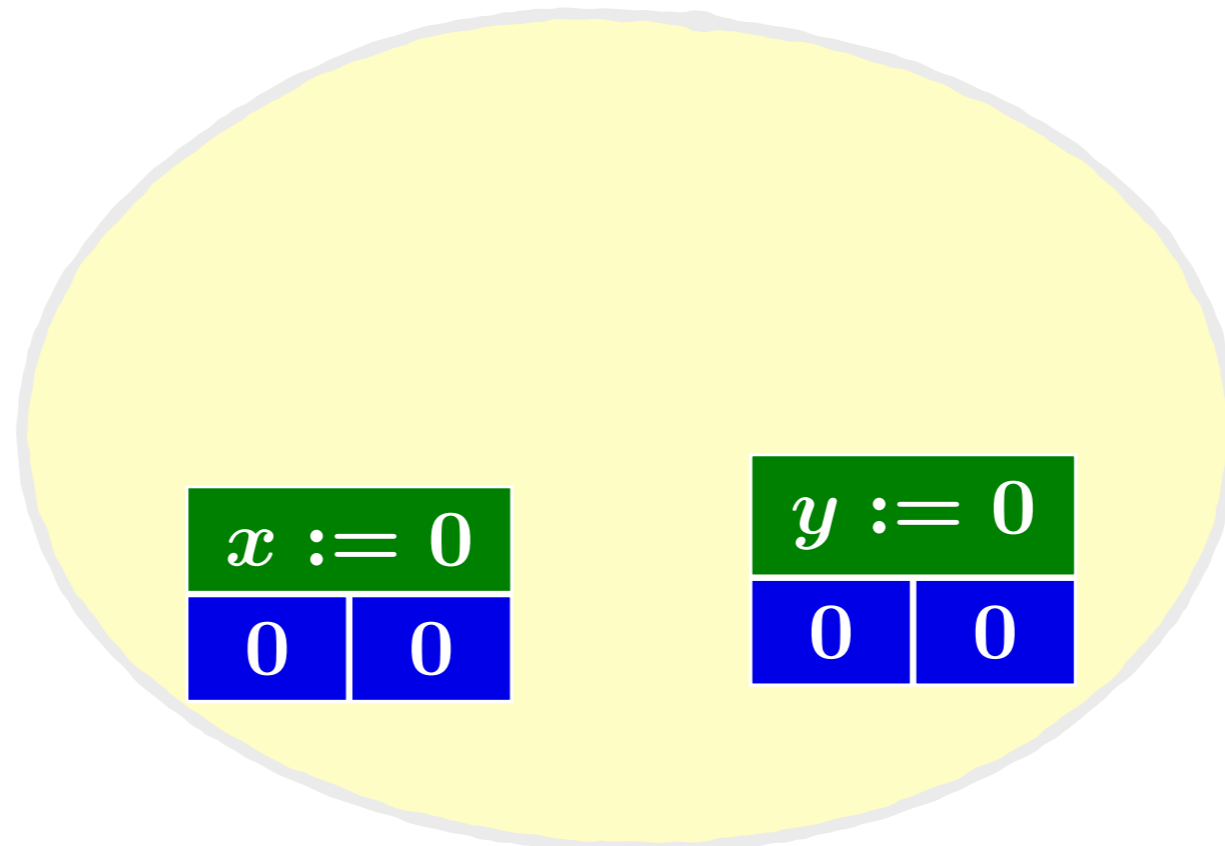
Store
Buffering
(SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$



Release-Acquire (RA)

Dekker's protocol

Store Buffering (SB)

Initially: $x = y = 0$

$x := 1$	
2	0
0	0

P_1

$x := 1$

$a := y$

P_2

$y := 1$

$b := x$

0	0
---	---

assertion

$(a = 0) \wedge (b = 0)?$

$x := 0$	
0	0

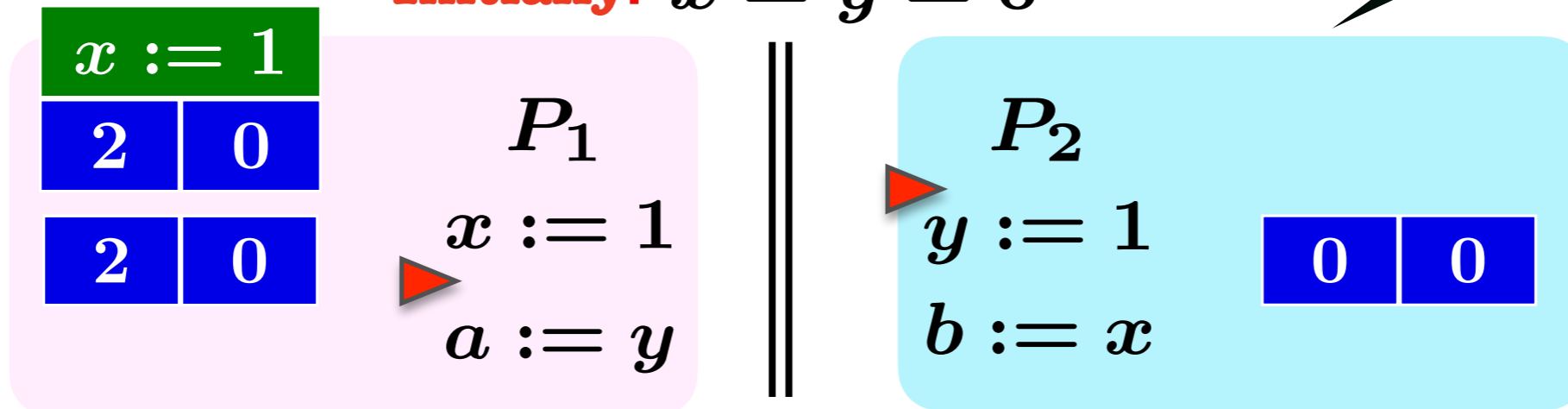
$y := 0$	
0	0

Release-Acquire (RA)

Dekker's
protocol

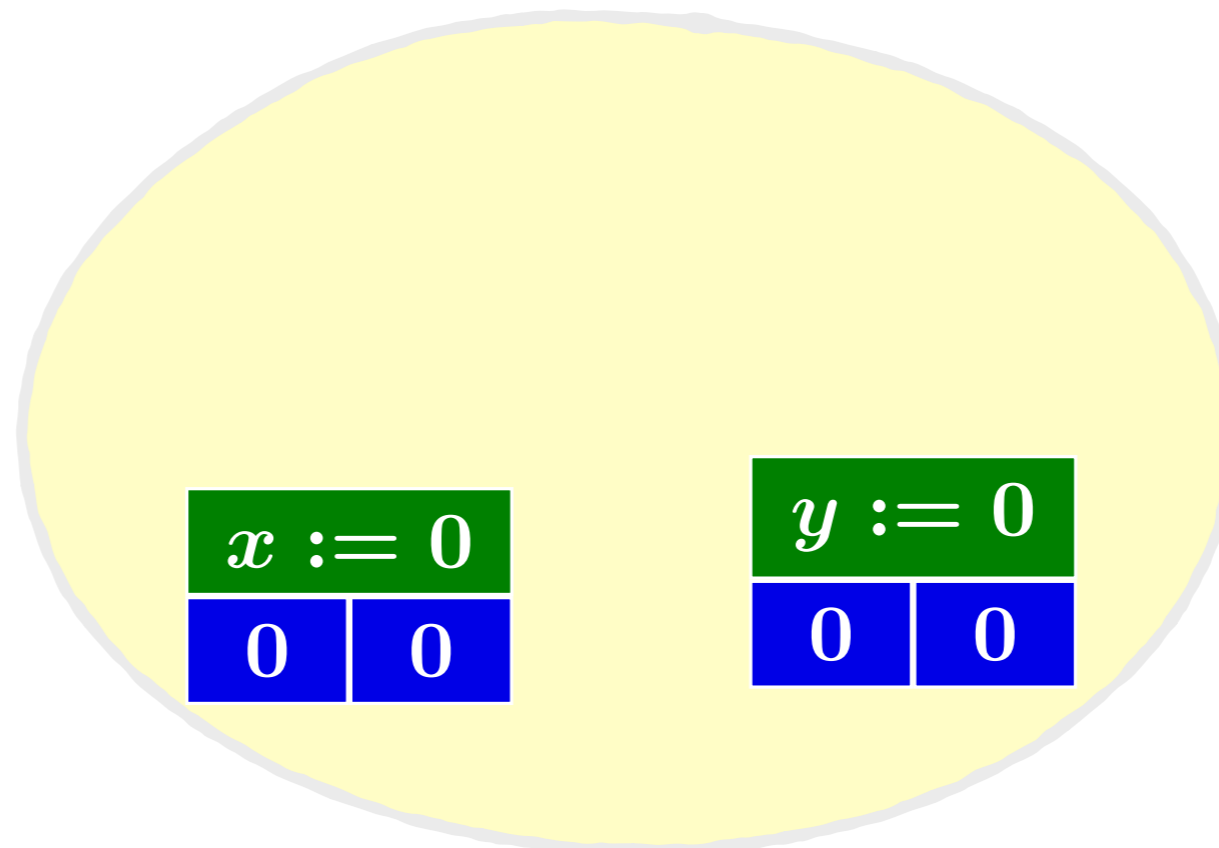
Store
Buffering
(SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

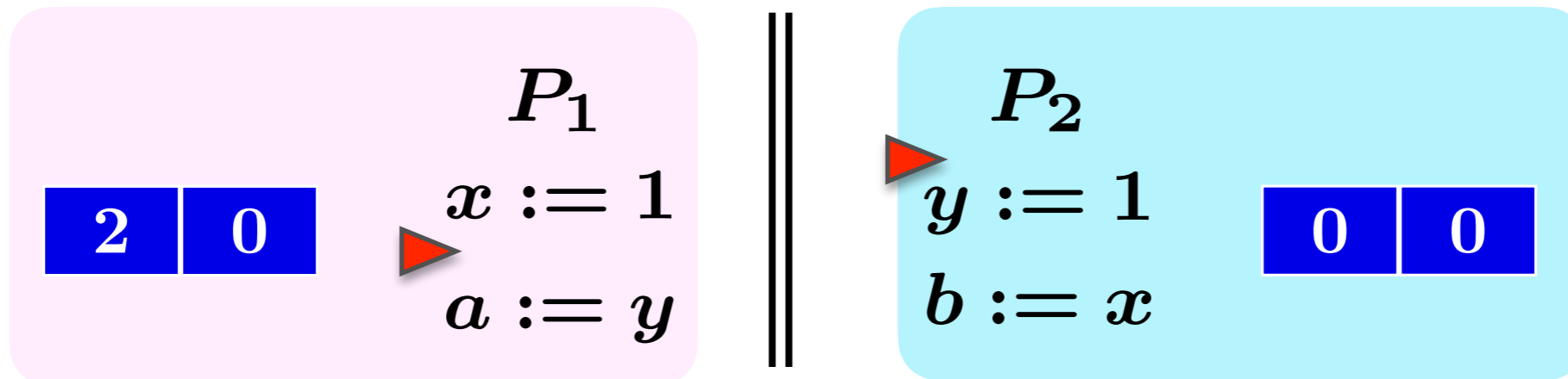


Release-Acquire (RA)

Dekker's
protocol

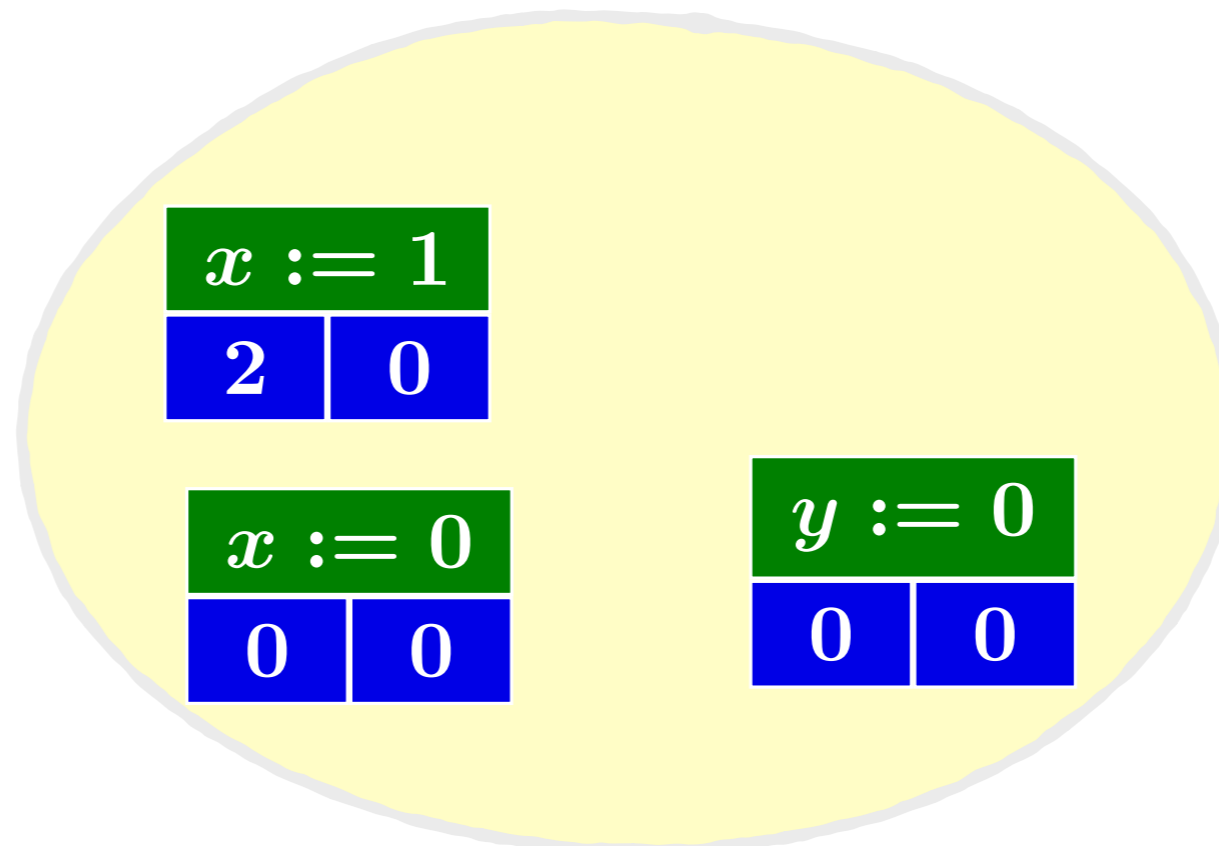
Store
Buffering
(SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

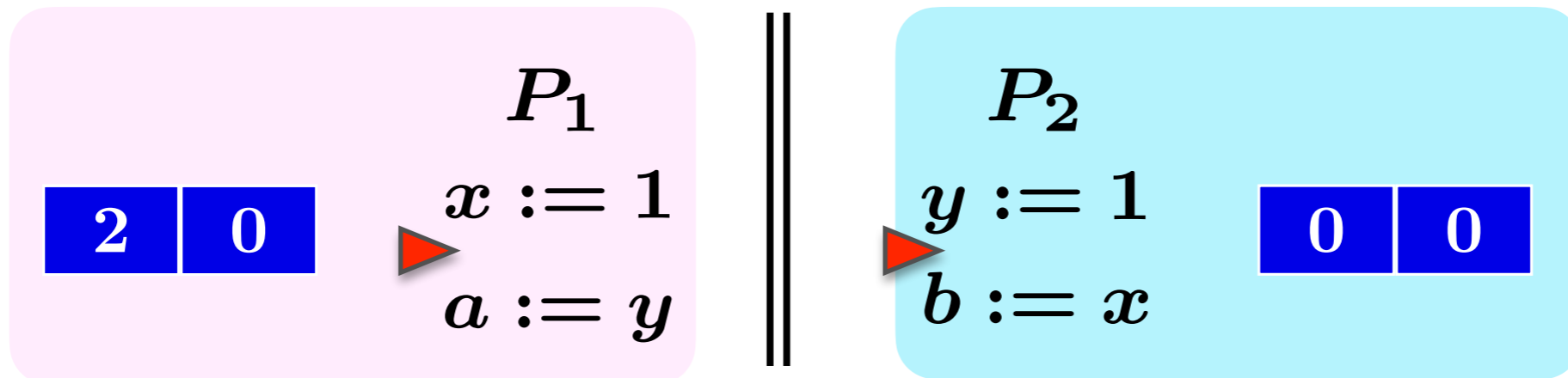


Release-Acquire (RA)

Dekker's
protocol

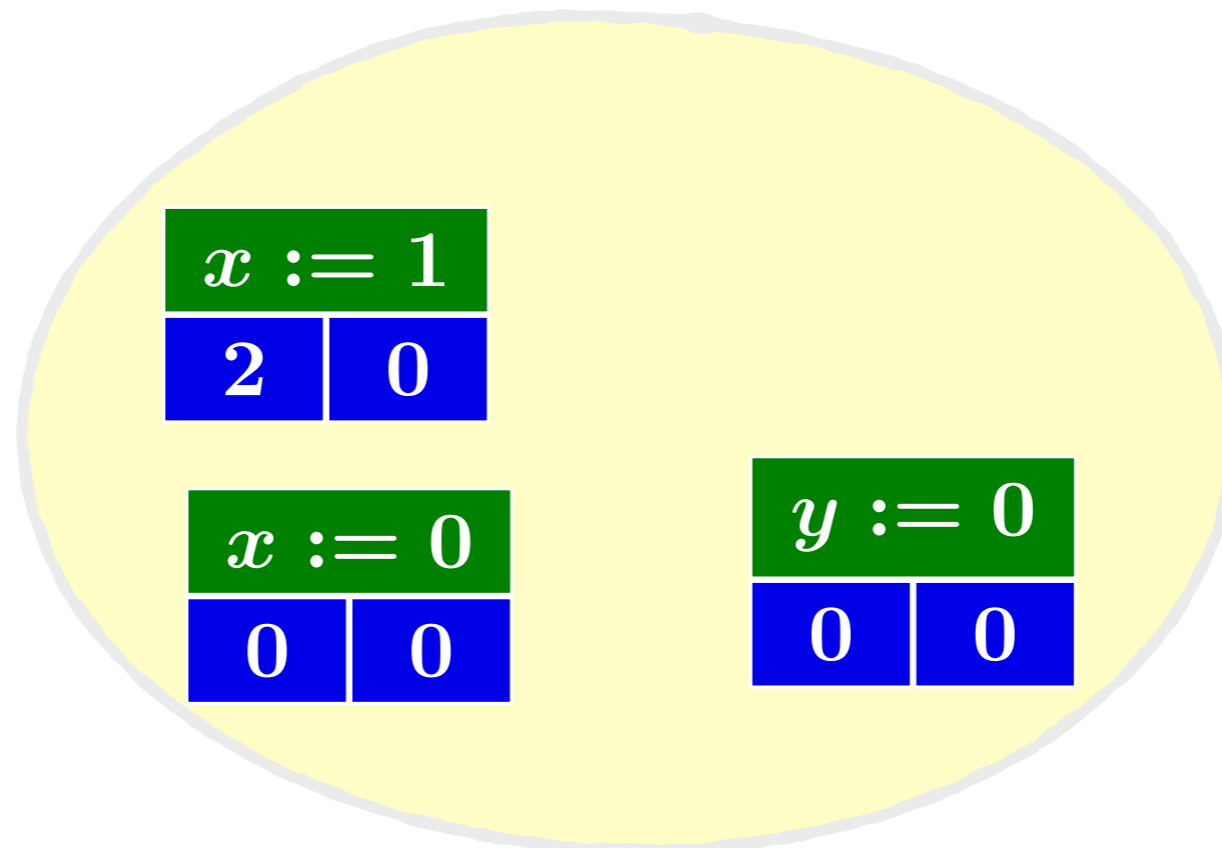
Store
Buffering
(SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$



Release-Acquire (RA)

Dekker's protocol

Store Buffering (SB)

Initially: $x = y = 0$

P_1

2	0
---	---

$x := 1$
 $a := y$

P_2

$y := 1$	
0	5
0	0

$y := 1$
 $b := x$

assertion $(a = 0) \wedge (b = 0)?$

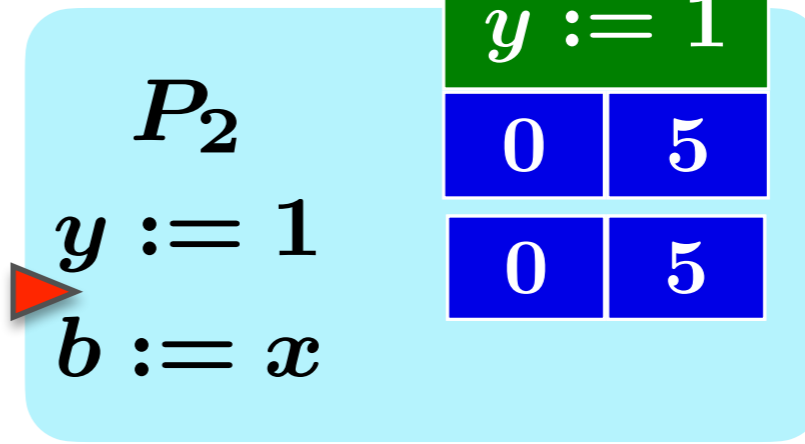
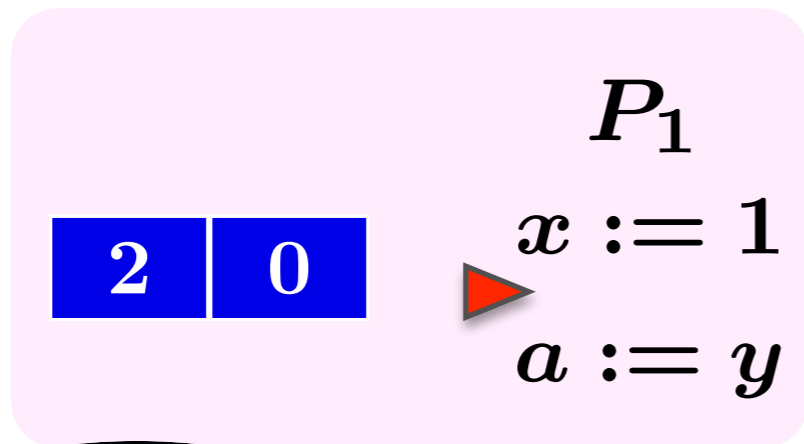
$x := 1$		
2	0	
$x := 0$		
0	0	
	$y := 0$	
	0	0

Release-Acquire (RA)

Dekker's protocol

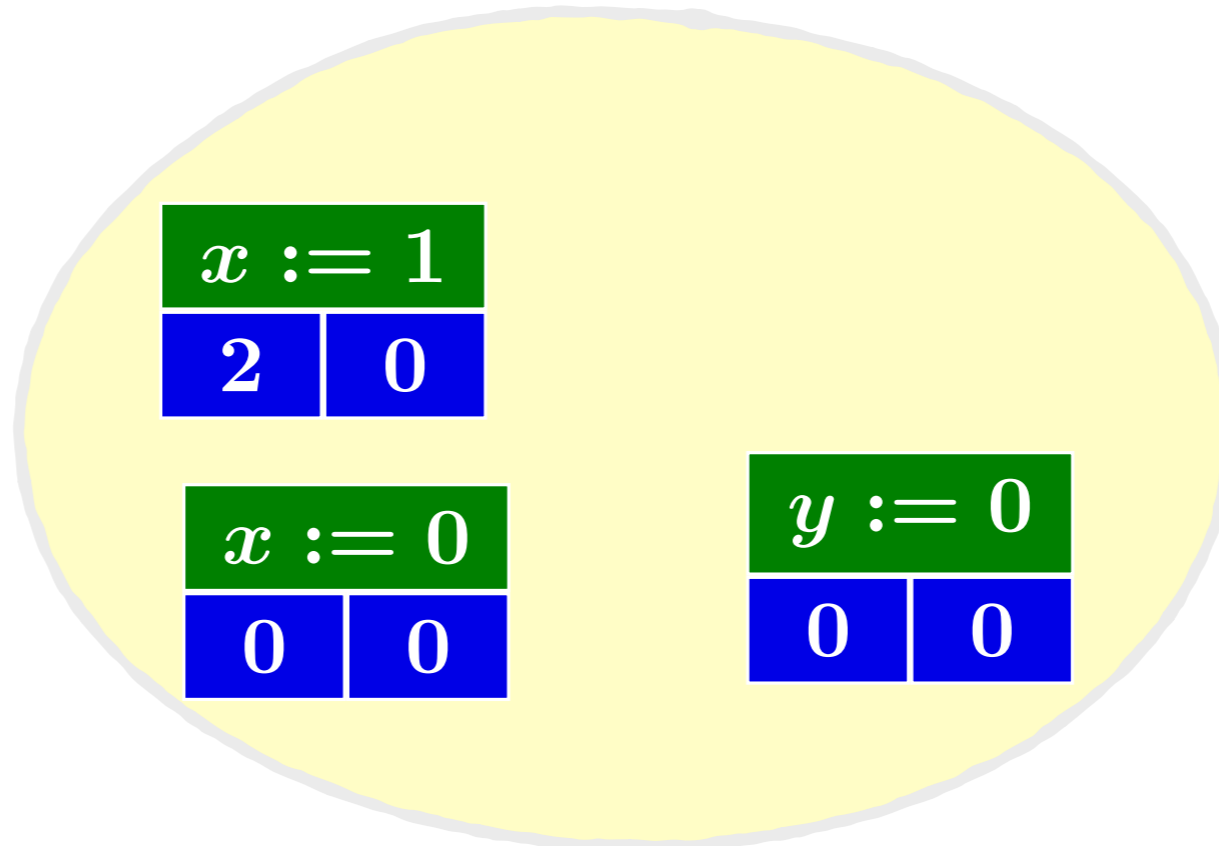
Store Buffering (SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

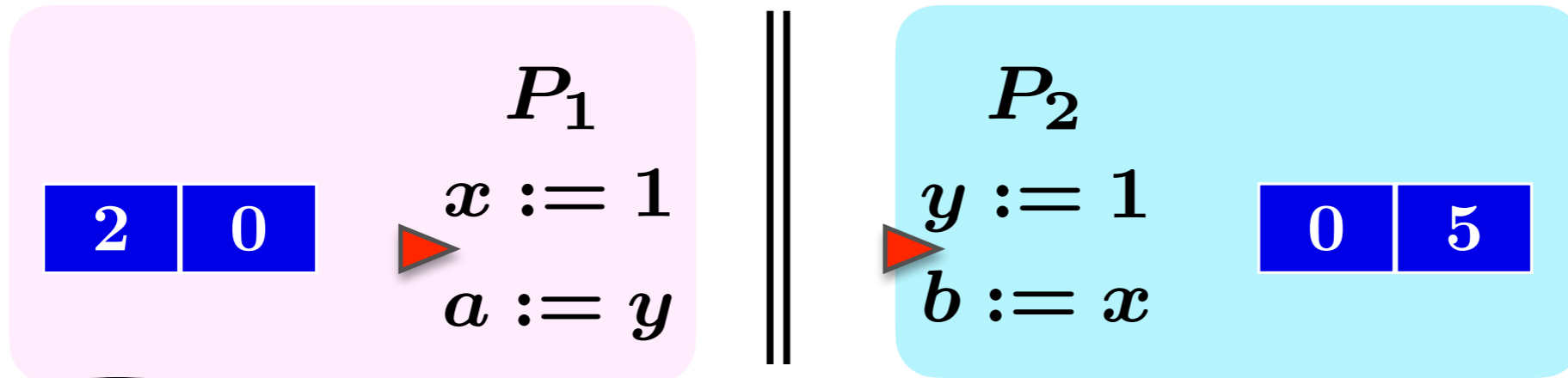


Release-Acquire (RA)

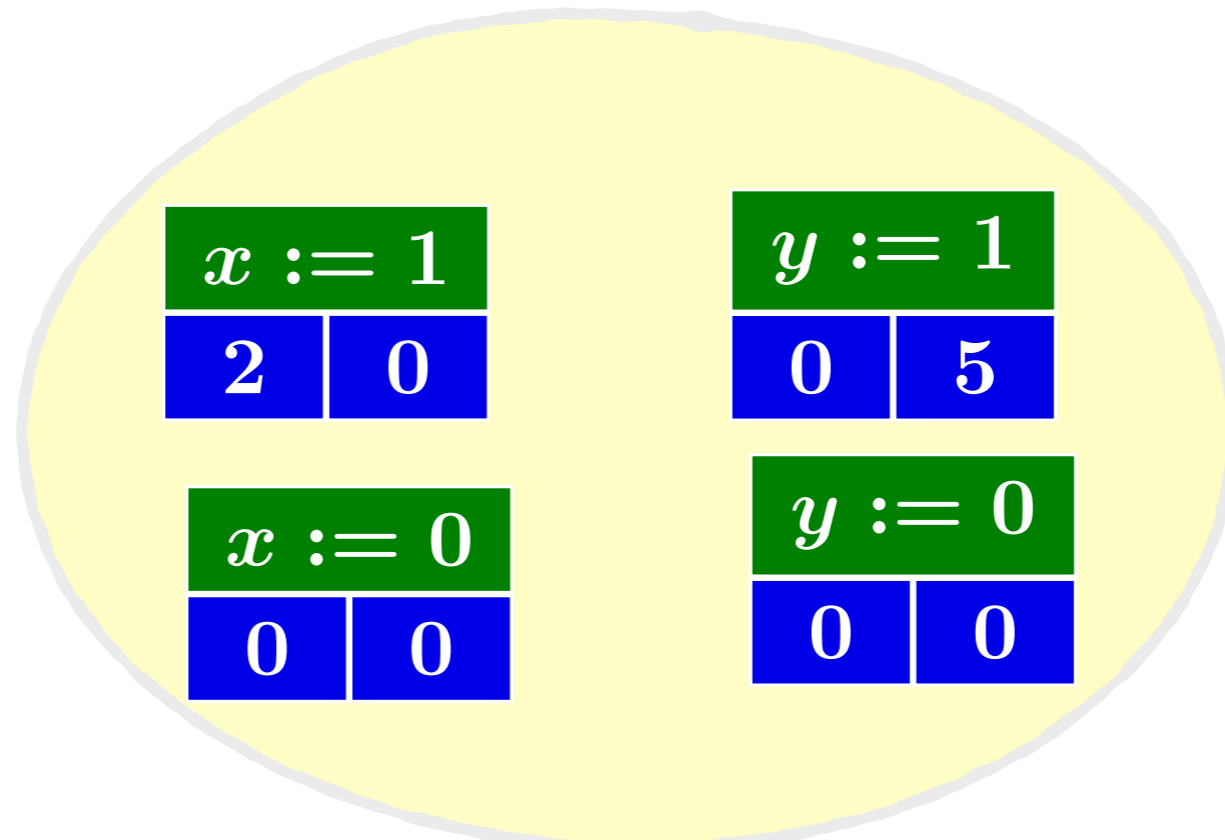
Dekker's protocol

Store Buffering (SB)

Initially: $x = y = 0$



assertion $(a = 0) \wedge (b = 0)?$

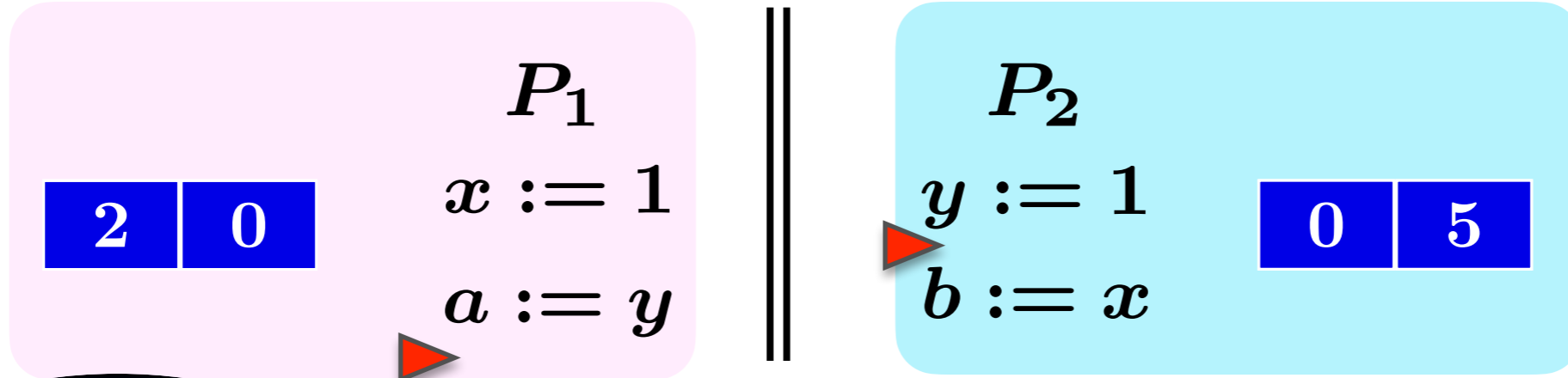


Release-Acquire (RA)

Dekker's protocol

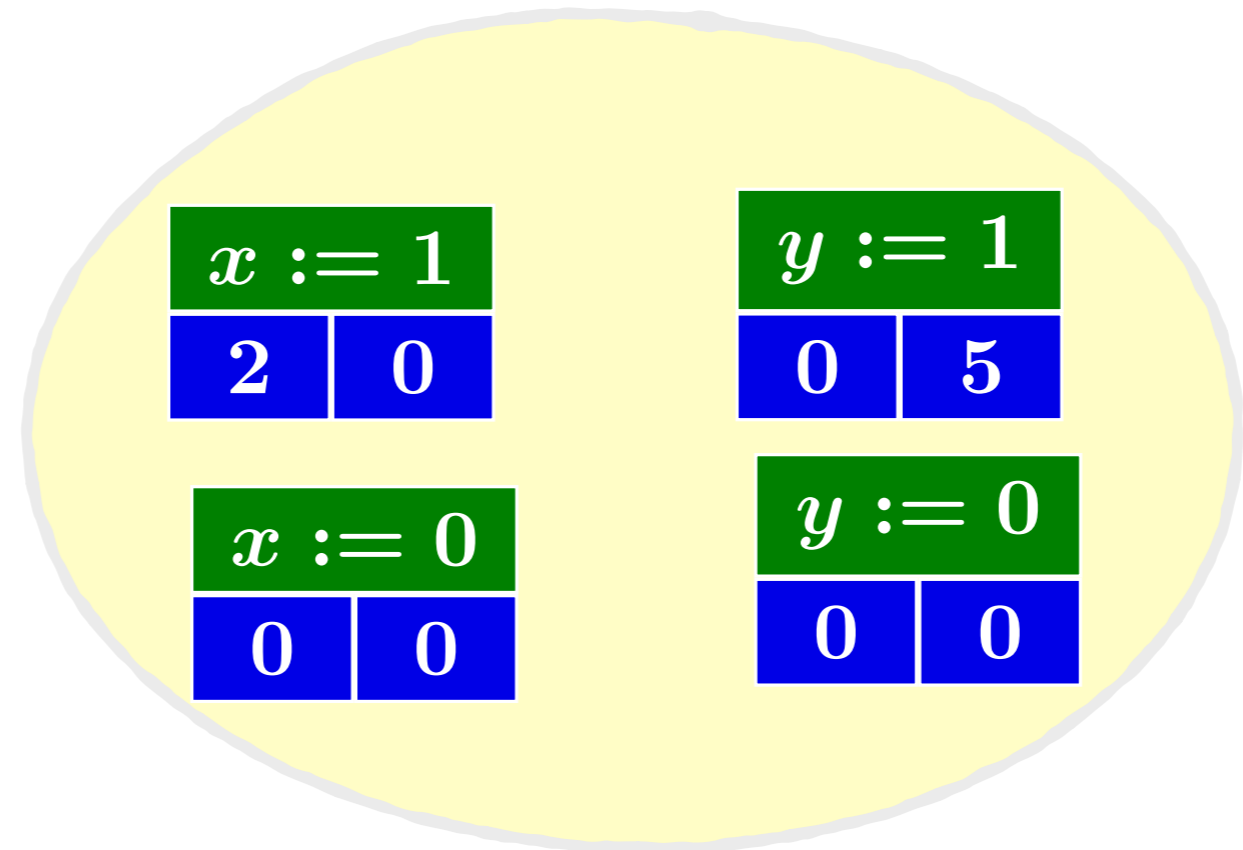
Store Buffering (SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

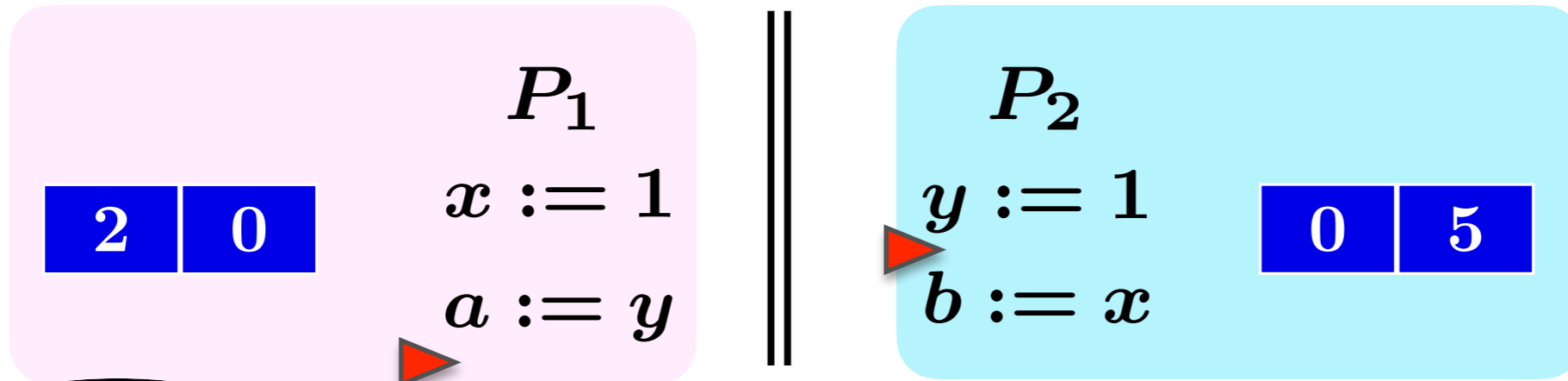


Release-Acquire (RA)

Dekker's
protocol

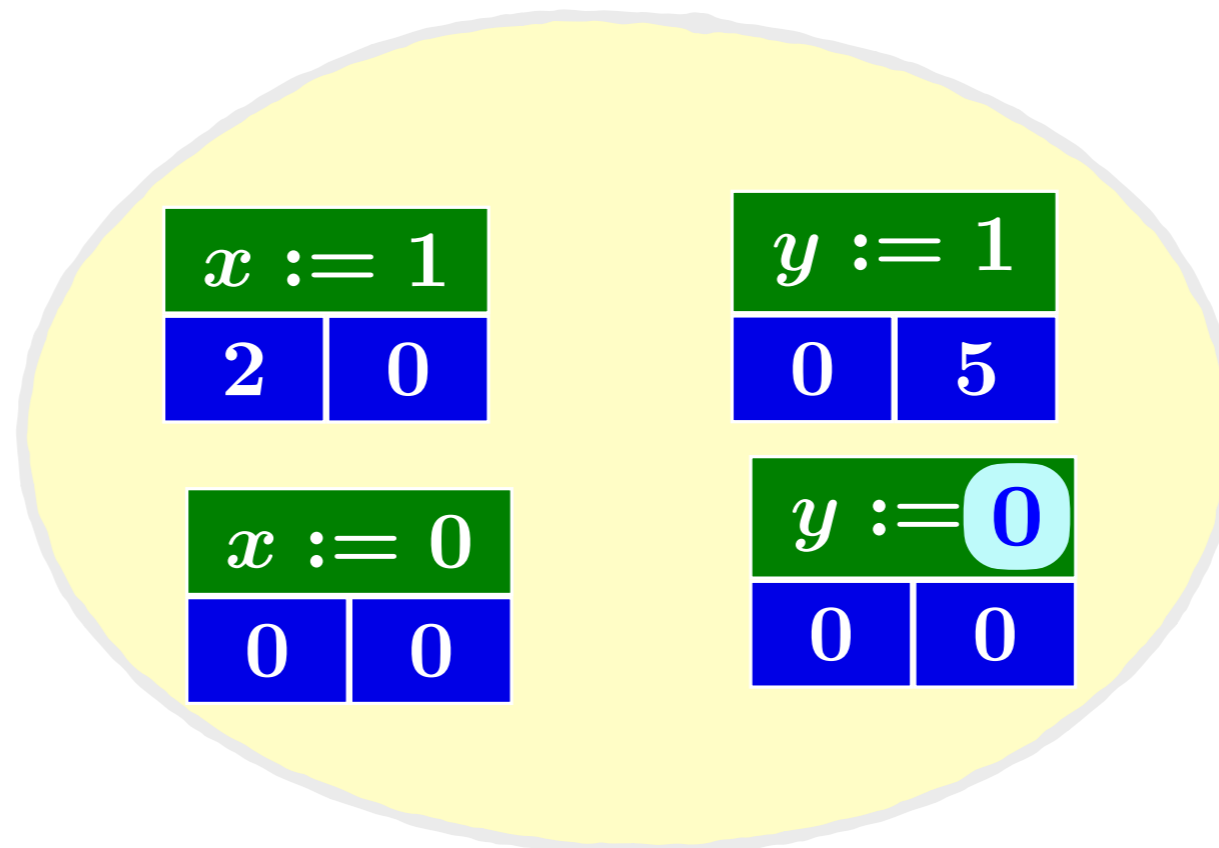
Store
Buffering
(SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

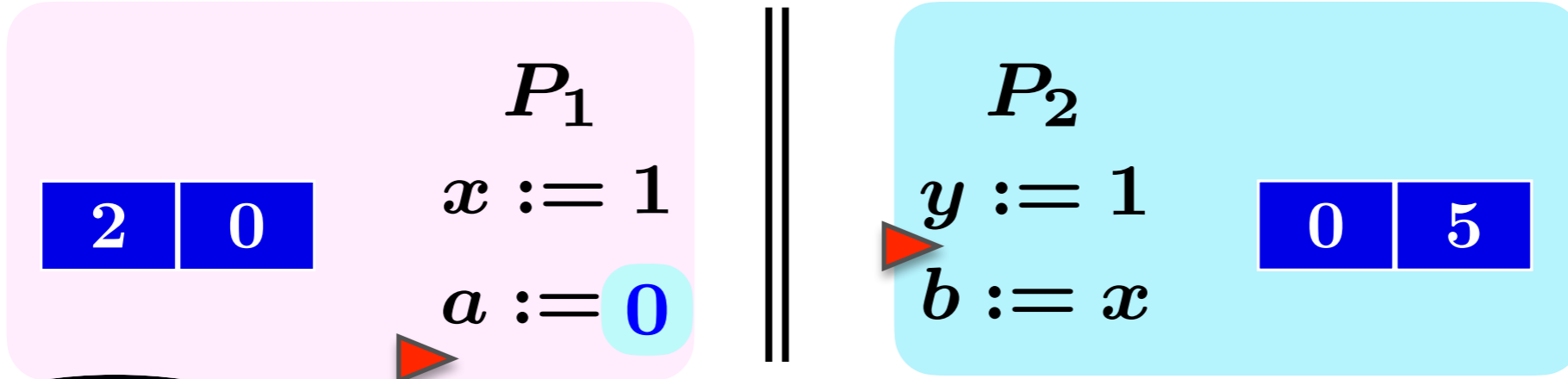


Release-Acquire (RA)

Dekker's protocol

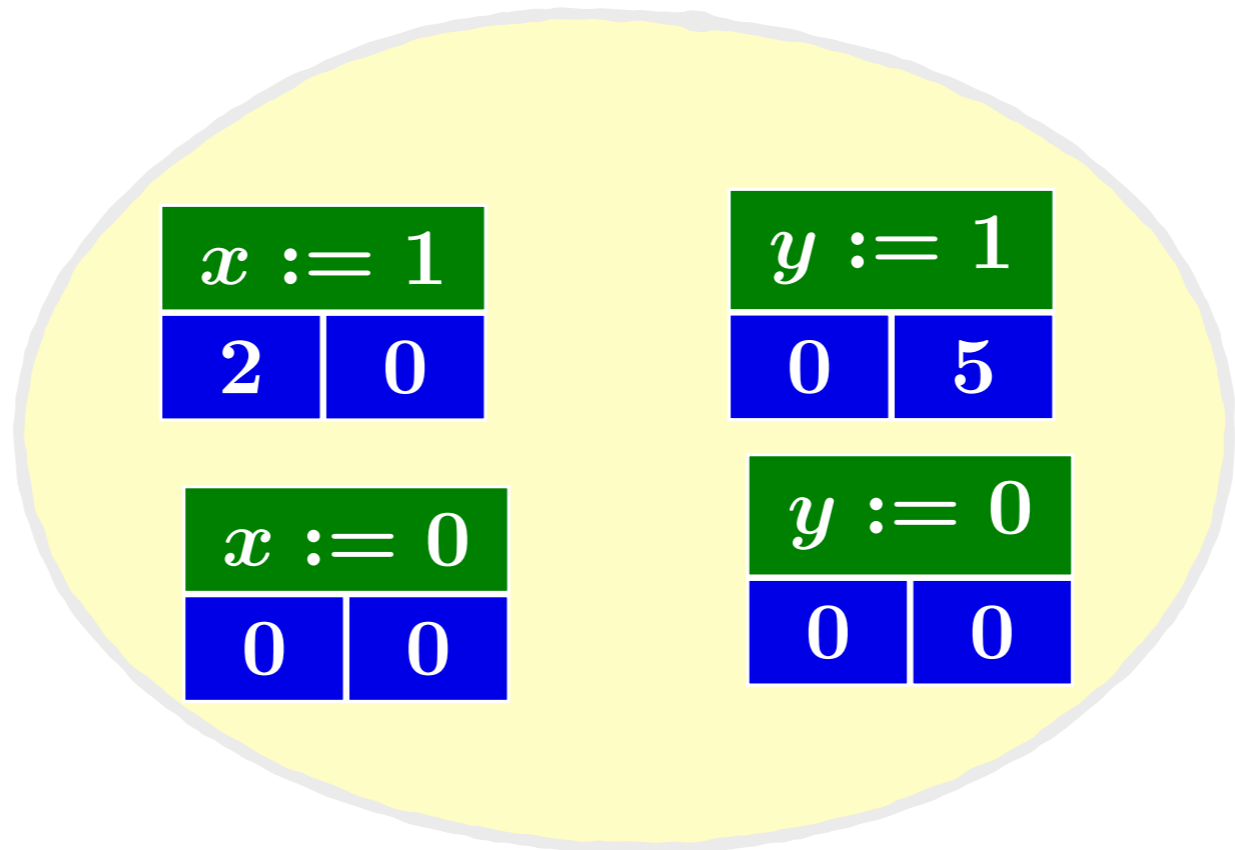
Store Buffering (SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

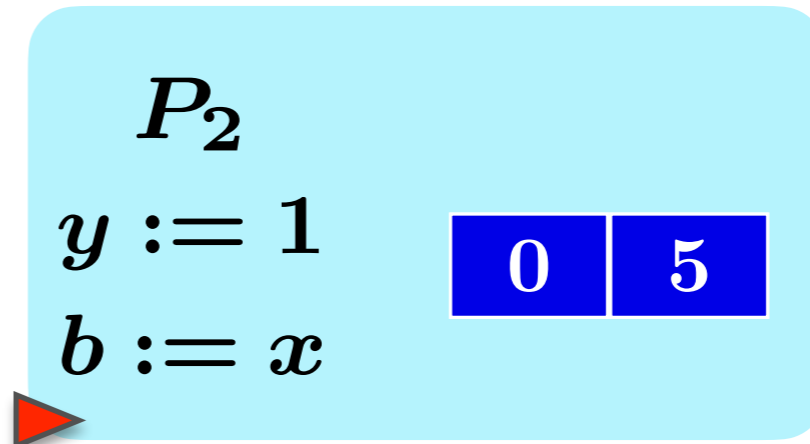
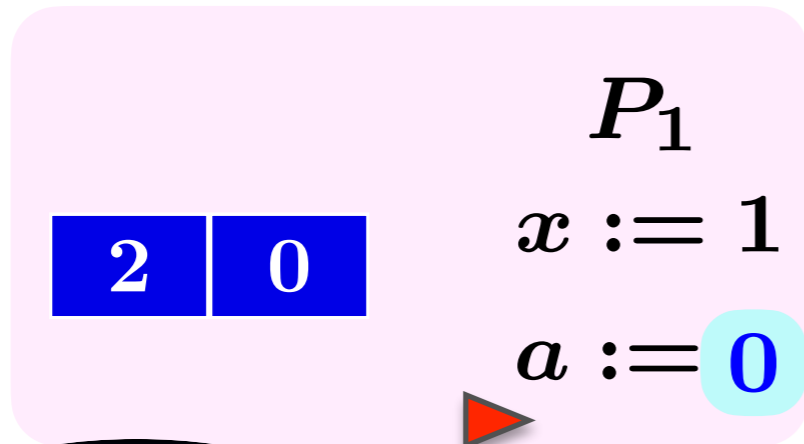


Release-Acquire (RA)

Dekker's
protocol

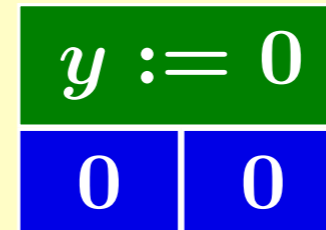
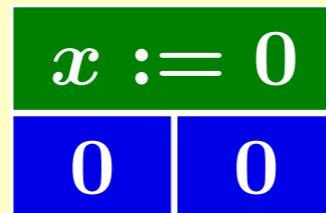
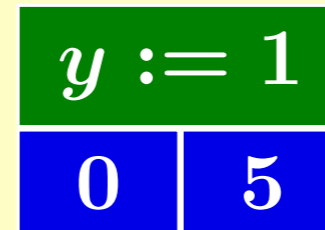
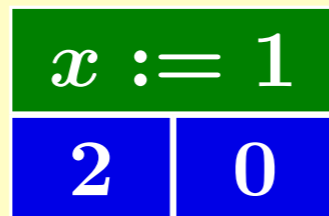
Store
Buffering
(SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

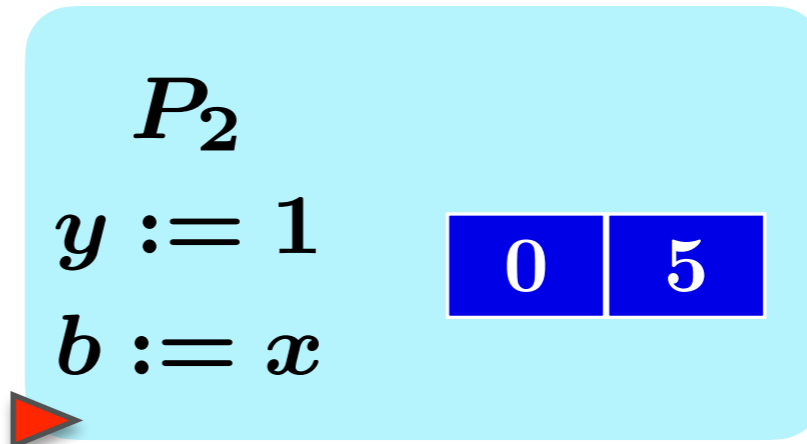
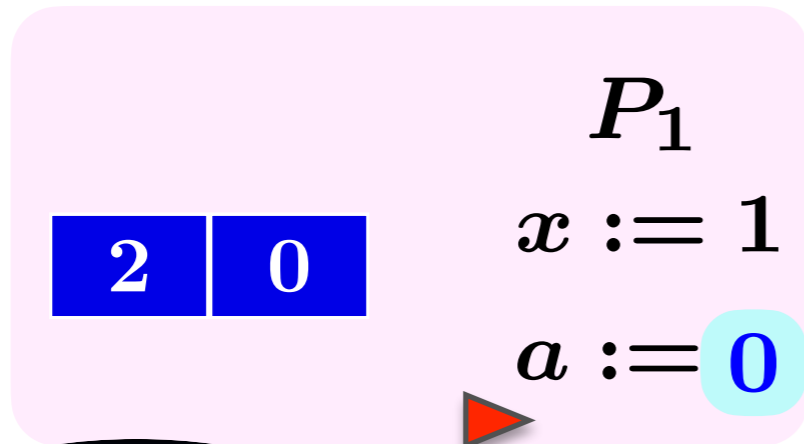


Release-Acquire (RA)

Dekker's protocol

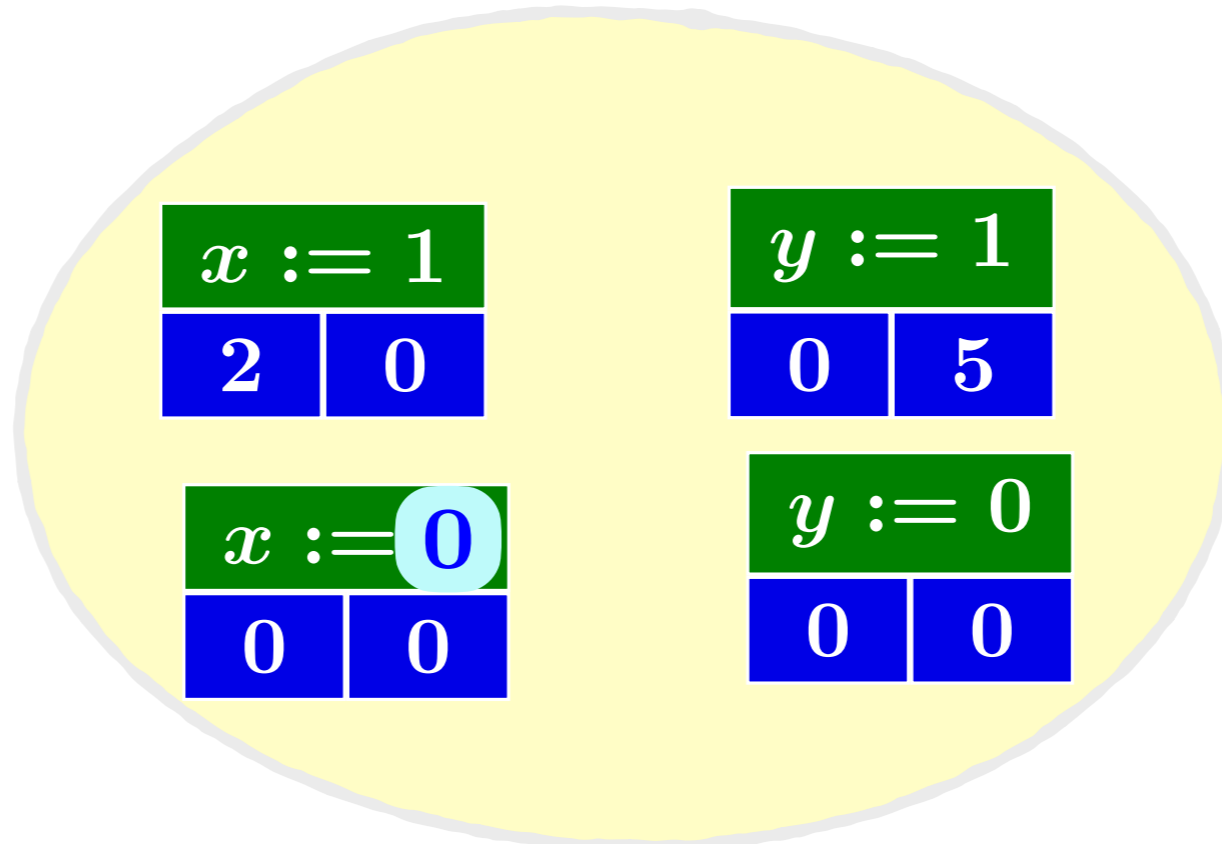
Store Buffering (SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

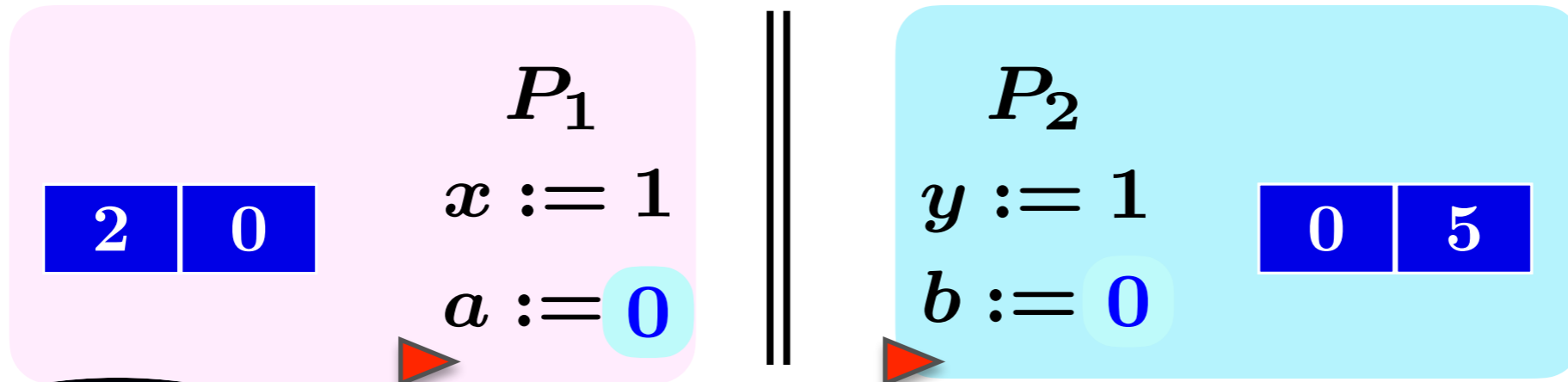


Release-Acquire (RA)

Dekker's
protocol

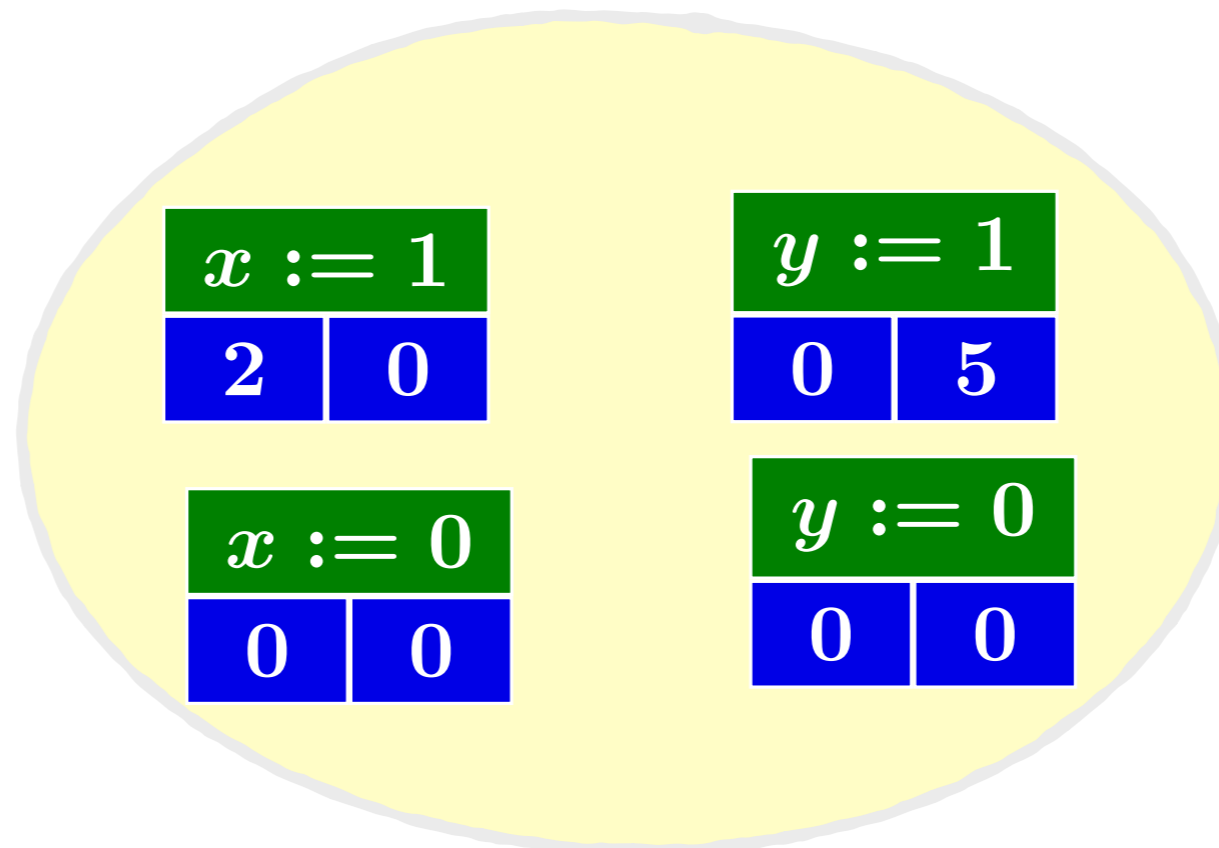
Store
Buffering
(SB)

Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

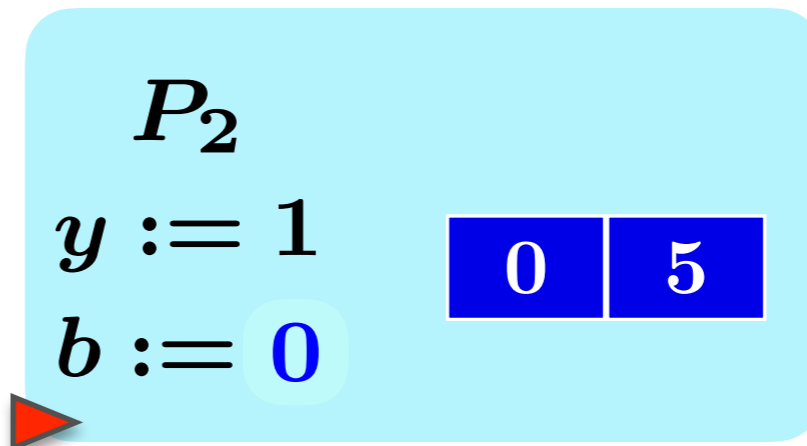
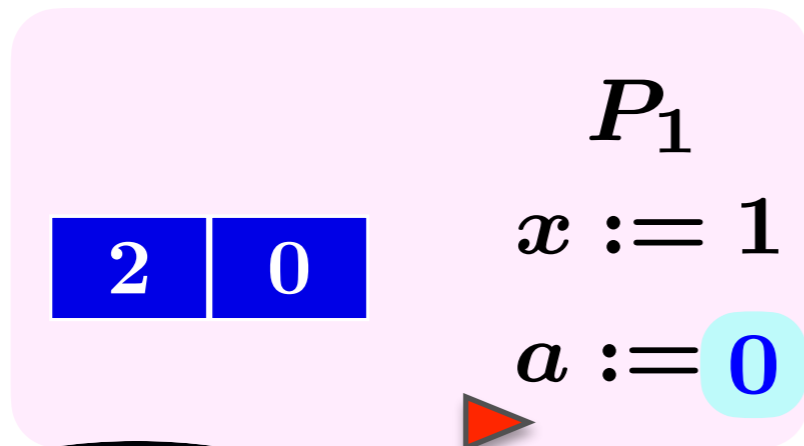


Release-Acquire (RA)

Dekker's
protocol

Store
Buffering
(SB)

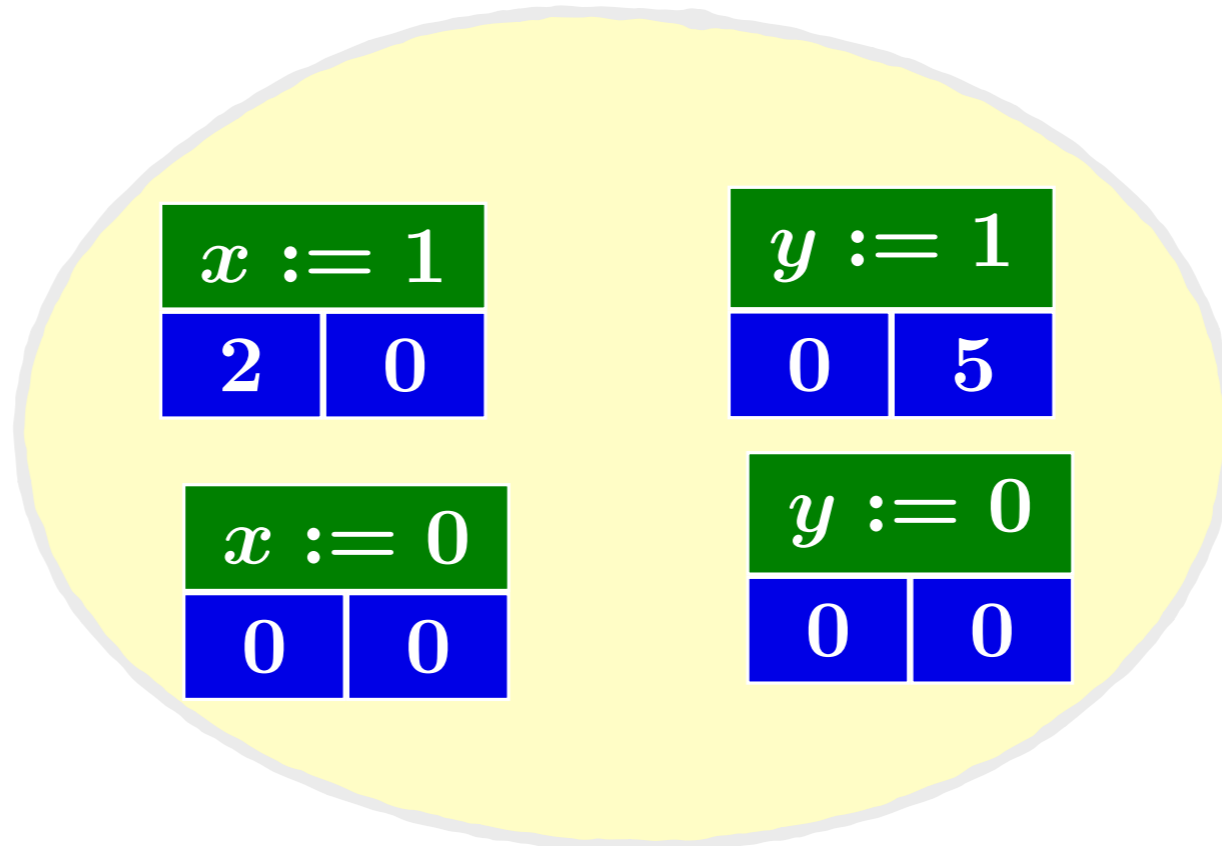
Initially: $x = y = 0$



assertion

$(a = 0) \wedge (b = 0)?$

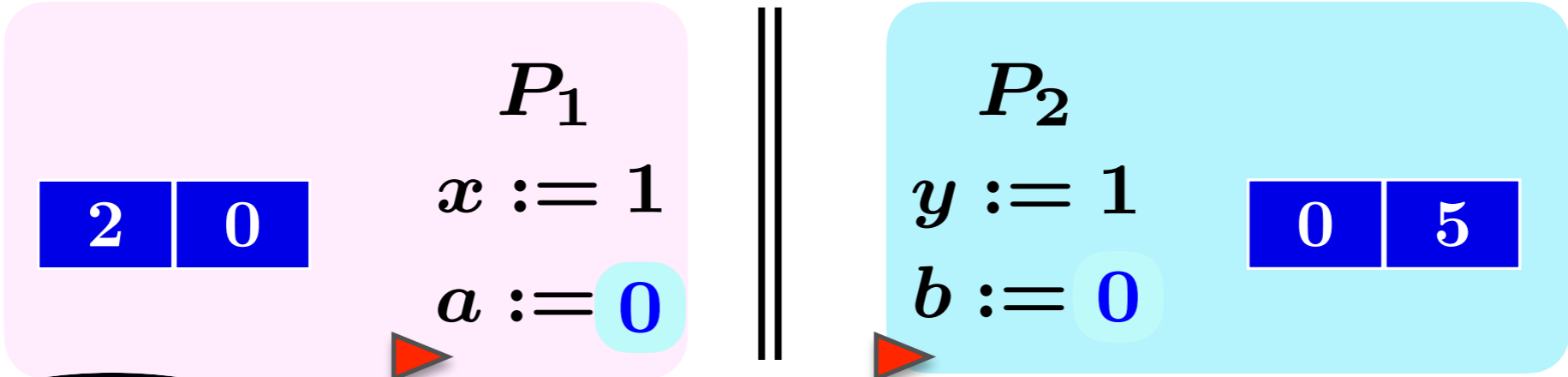
reachable



Release-Acquire (RA)

Dekker's protocol
Store Buffering (SB)

Initially: $x = y = 0$

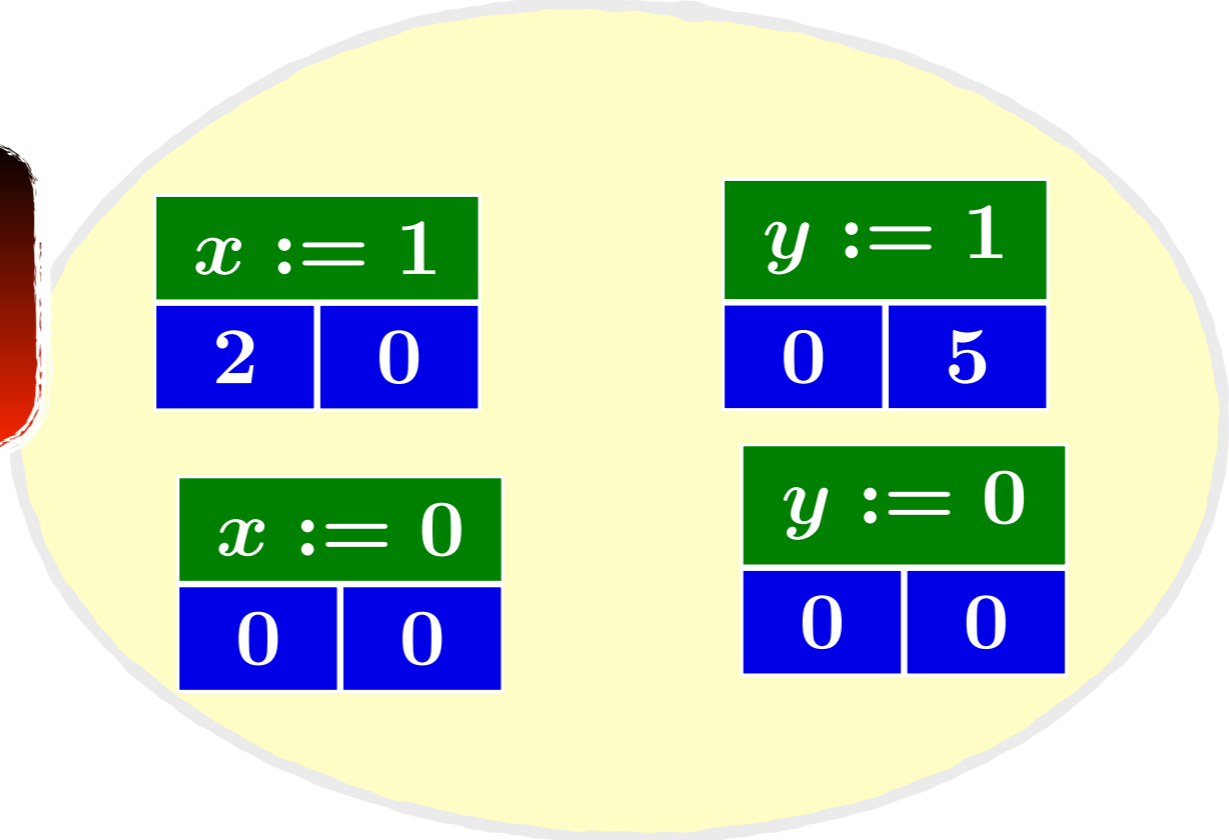


assertion

$(a = 0) \wedge (b = 0)?$

reachable

"Dekker's Protocol faulty under RA"



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

0	0
---	---

P_2
 $y := 1$

0	0
---	---

P_3
 $a_1 := x$
 $a_2 := y$

0	0
---	---

P_4
 $b_1 := y$
 $b_2 := x$

0	0
---	---

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 0$	$y := 0$
0 0	0 0

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

0	0
---	---

P_2
 $y := 1$

0	0
---	---

P_3
 $a_1 := x$
 $a_2 := y$

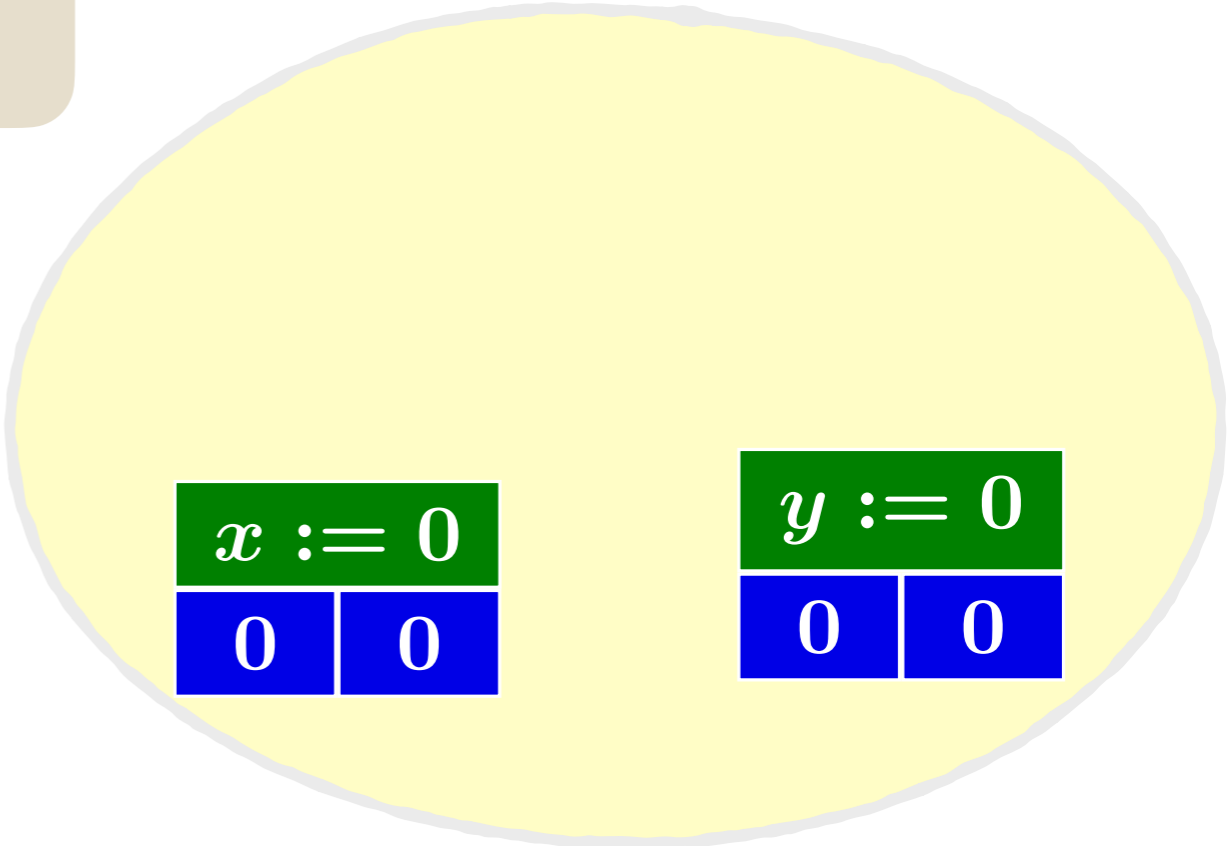
0	0
---	---

P_4
 $b_1 := y$
 $b_2 := x$

0	0
---	---

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

“can write events
be observed in different
orders by different
processes under RA?”



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

0	0
---	---

P_2
 $y := 1$

0	0
---	---

P_3
 $a_1 := x$

0	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 0$

0 0

$y := 0$

0 0

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

0	0
---	---

P_2
 $y := 1$

0	0
---	---

P_3
 $a_1 := x$

0	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	0
---	---

P_3
 $a_1 := x$

0	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 0$	$y := 0$
0 0	0 0

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	0
---	---

P_3
 $a_1 := x$

0	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 1$

3	0
---	---

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := x$

0	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 1$

3	0
---	---

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := x$

0	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 1$

3	0
---	---

$y := 1$

0	4
---	---

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := x$

0	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 1$

3	0
---	---

$y := 1$

0	4
---	---

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := x$

0	0
---	---

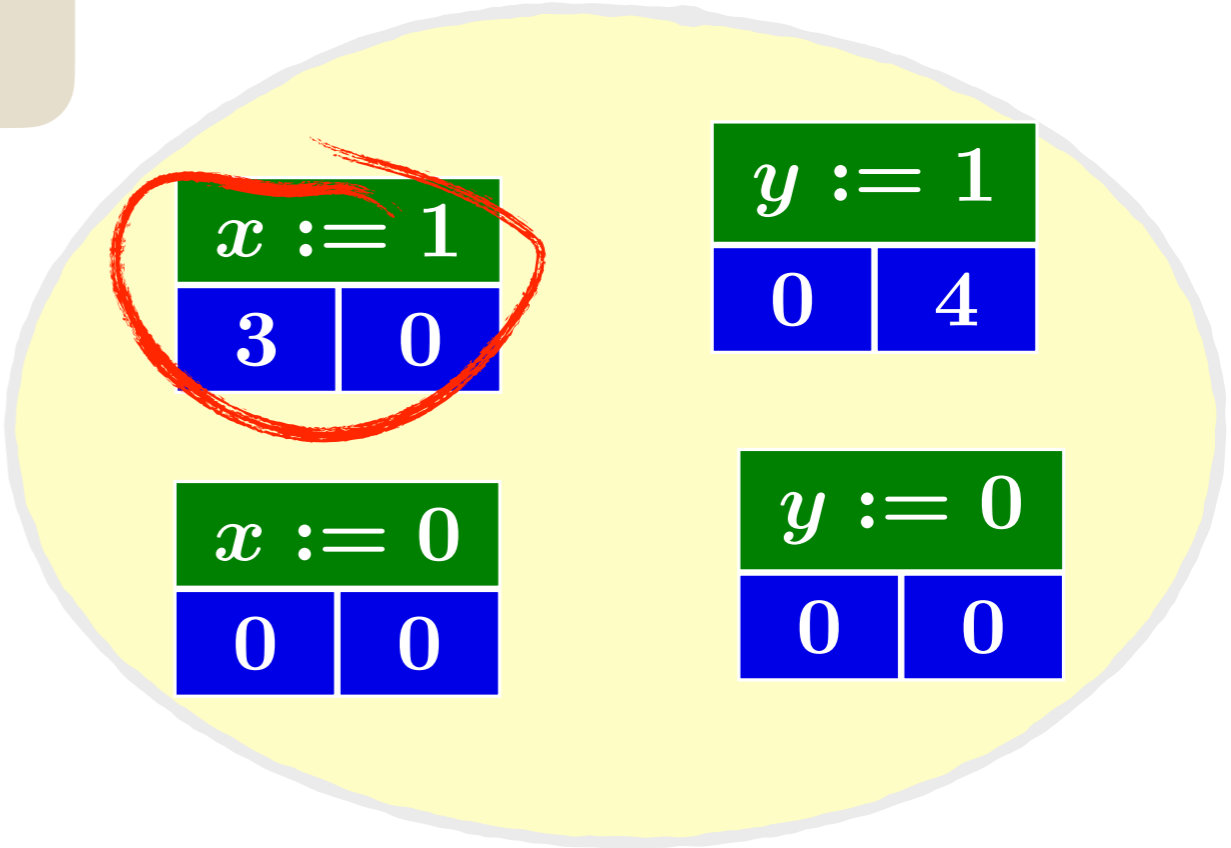
 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := x$

3	0
---	---

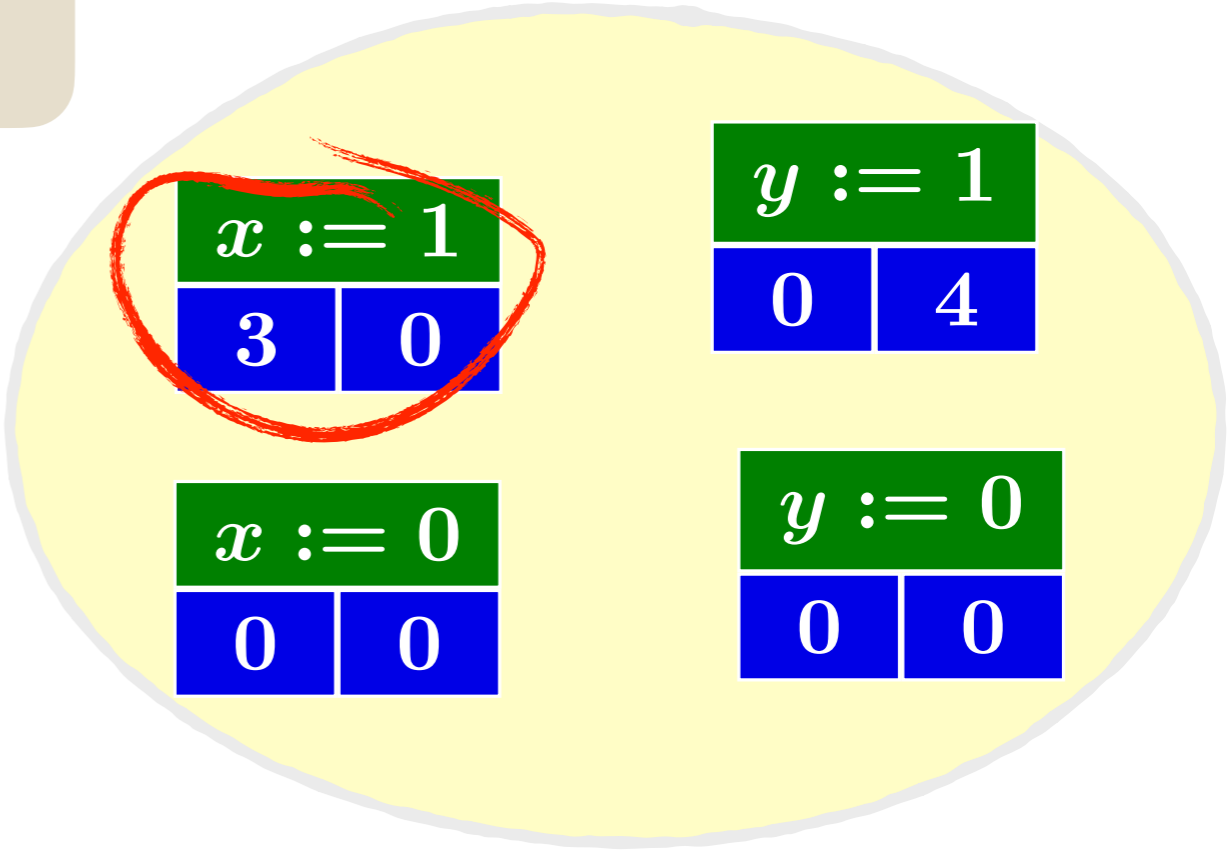
 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 1$

3	0
---	---

$y := 1$

0	4
---	---

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

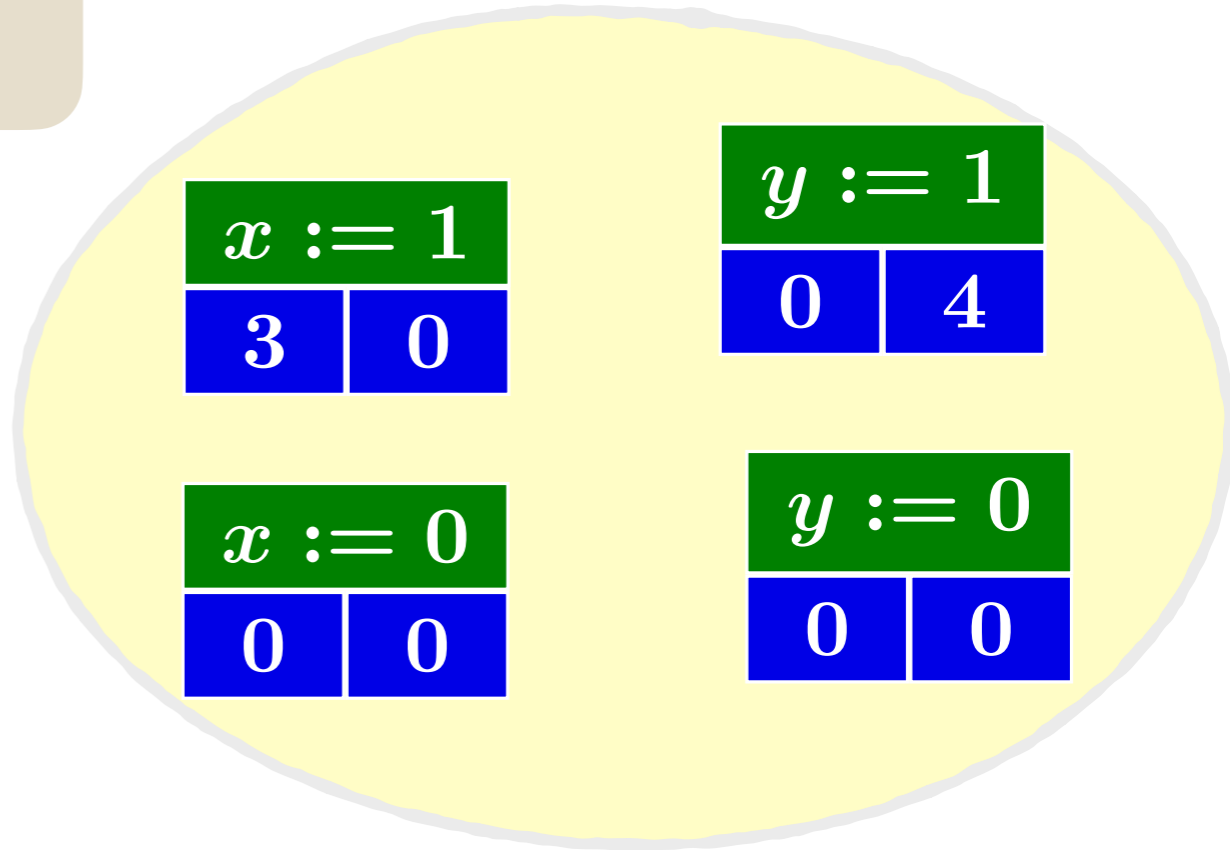
 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

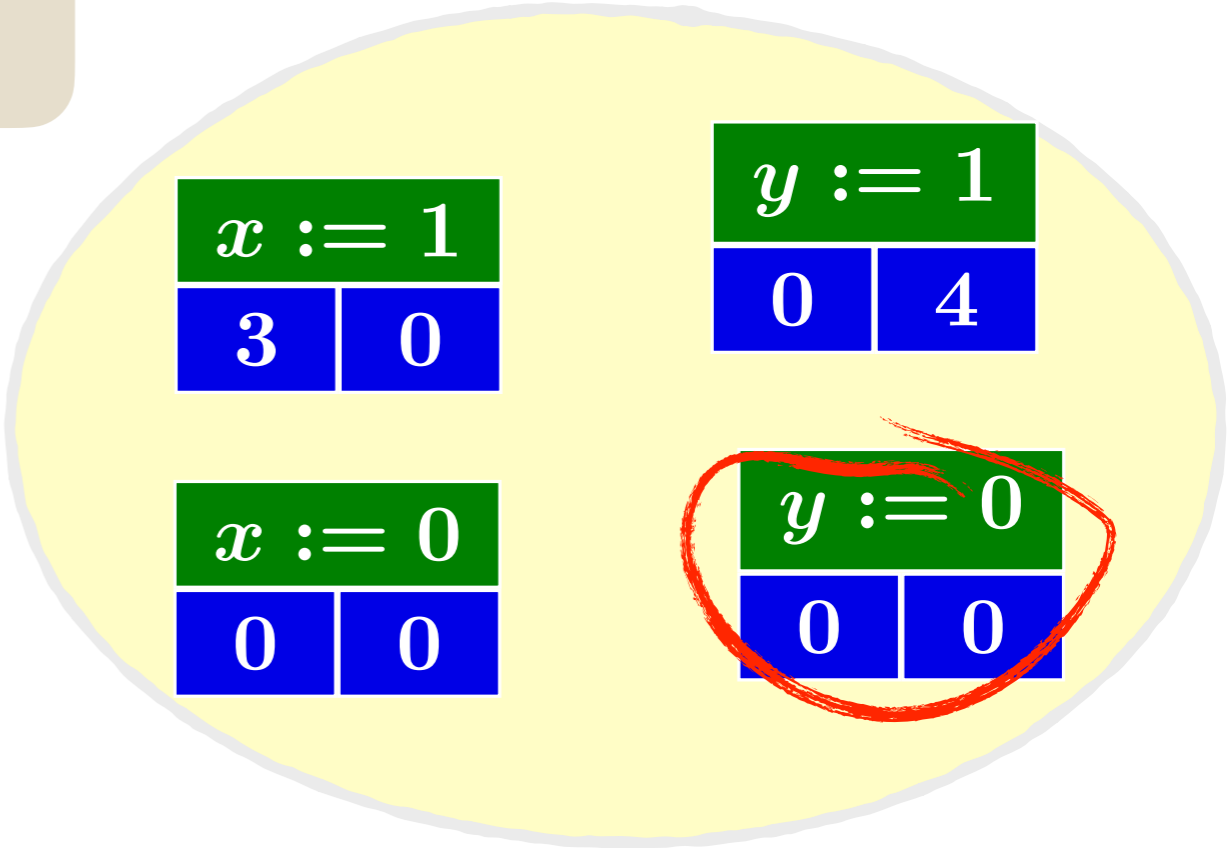
 $a_2 := y$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

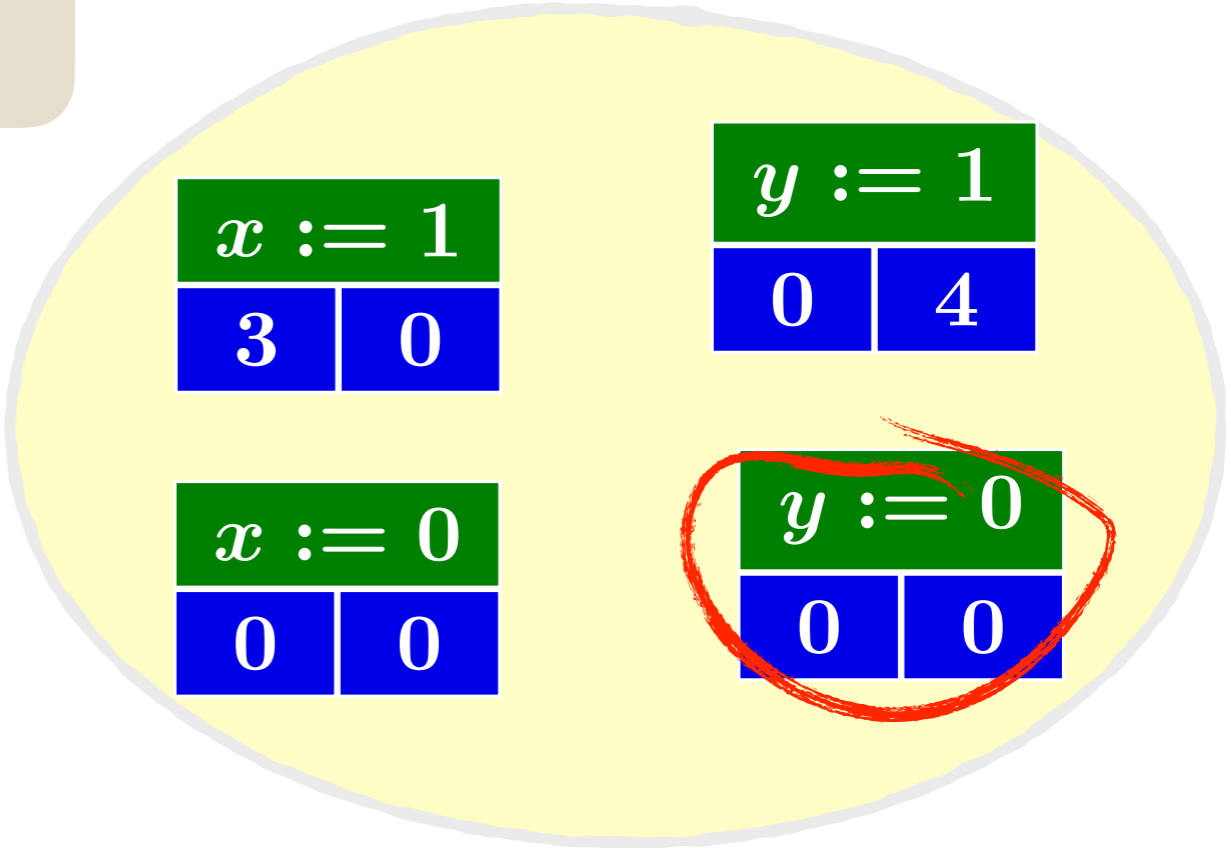
 $a_2 := 0$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge (b_1 = 1) \wedge (b_2 = 0)?$



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

 $a_2 := 0$

P_4
 $b_1 := y$

0	0
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 1$

3	0
---	---

$y := 1$

0	4
---	---

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

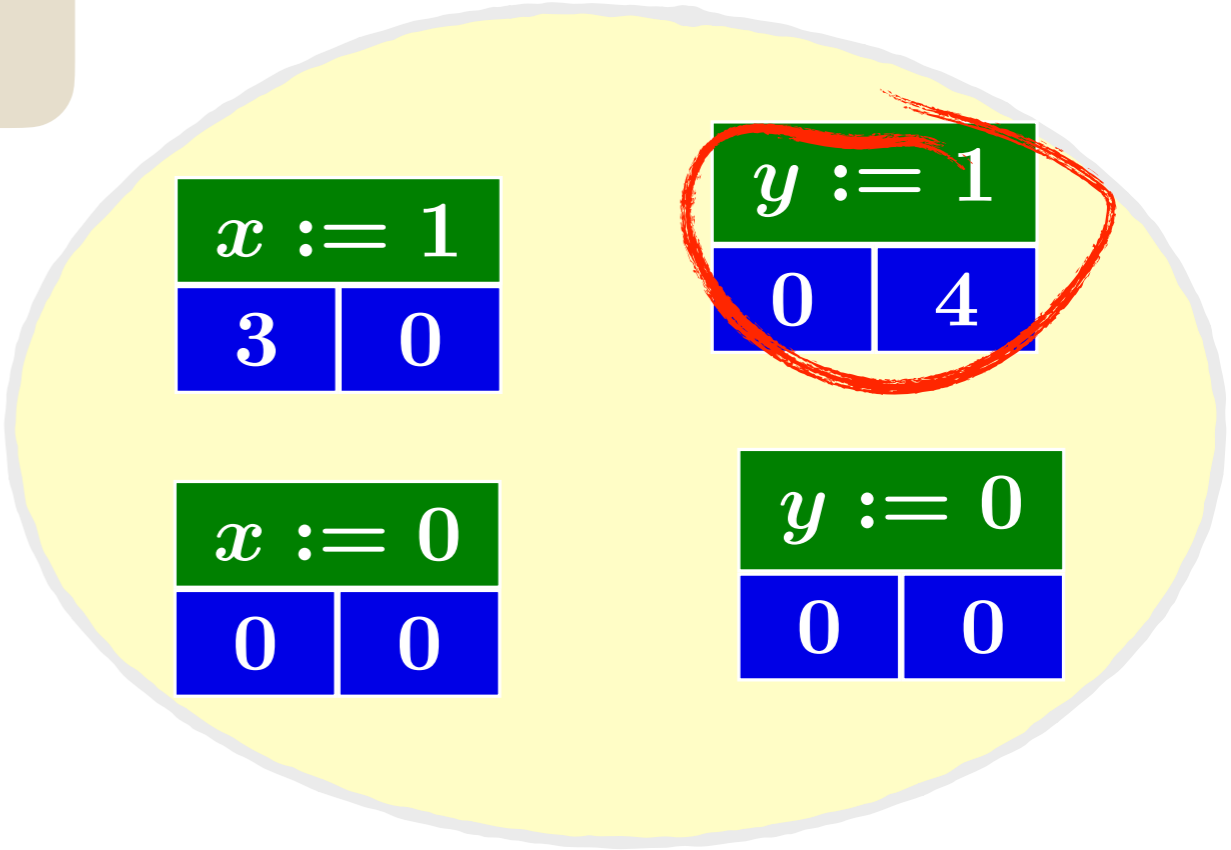
 $a_2 := 0$

P_4
 $b_1 := 1$

0	4
---	---

 $b_2 := x$

$(a_1 = 1) \wedge (a_2 = 0) \wedge (b_1 = 1) \wedge (b_2 = 0)?$



Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

 $a_2 := 0$

P_4
 $b_1 := 1$

0	4
---	---

 $b_2 := 0$

$(a_1 = 1) \wedge (a_2 = 0) \wedge$
 $(b_1 = 1) \wedge (b_2 = 0)?$

$x := 1$

3	0
---	---

$y := 1$

0	4
---	---

$x := 0$

0	0
---	---

$y := 0$

0	0
---	---

Release-Acquire (RA)

Initially: $x = y = 0$

IRIW

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

P_3
 $a_1 := 1$

3	0
---	---

 $a_2 := 0$

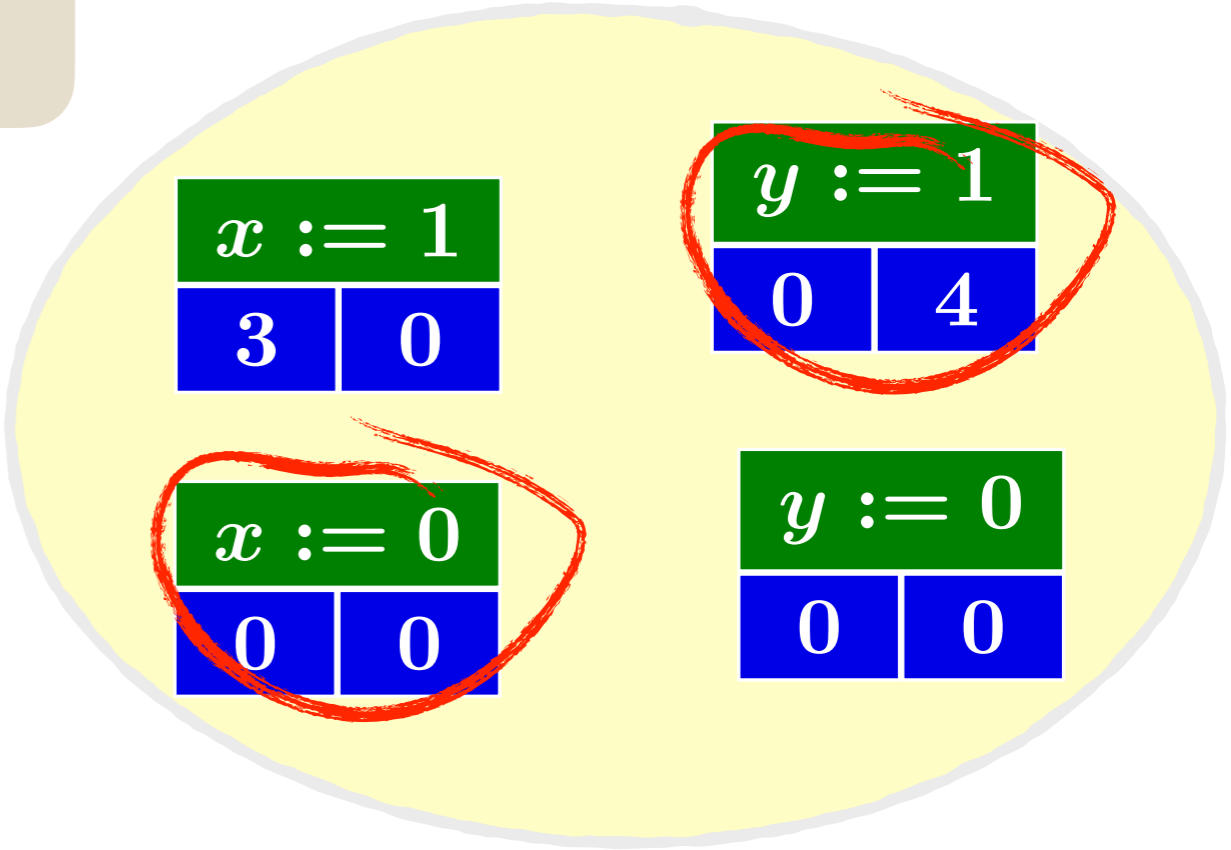
P_4
 $b_1 := 1$

0	4
---	---

 $b_2 := 0$

$(a_1 = 1) \wedge (a_2 = 0) \wedge (b_1 = 1) \wedge (b_2 = 0)?$

reachable



Release-Acquire (RA)

Initially: $x = y = 0$

P_1
 $x := 1$

3	0
---	---

P_2
 $y := 1$

0	4
---	---

IRIW

“write events can be observed in different orders by different processes under RA”

P_3
 $a_1 := 1$

3	0
---	---

 $a_2 := 0$

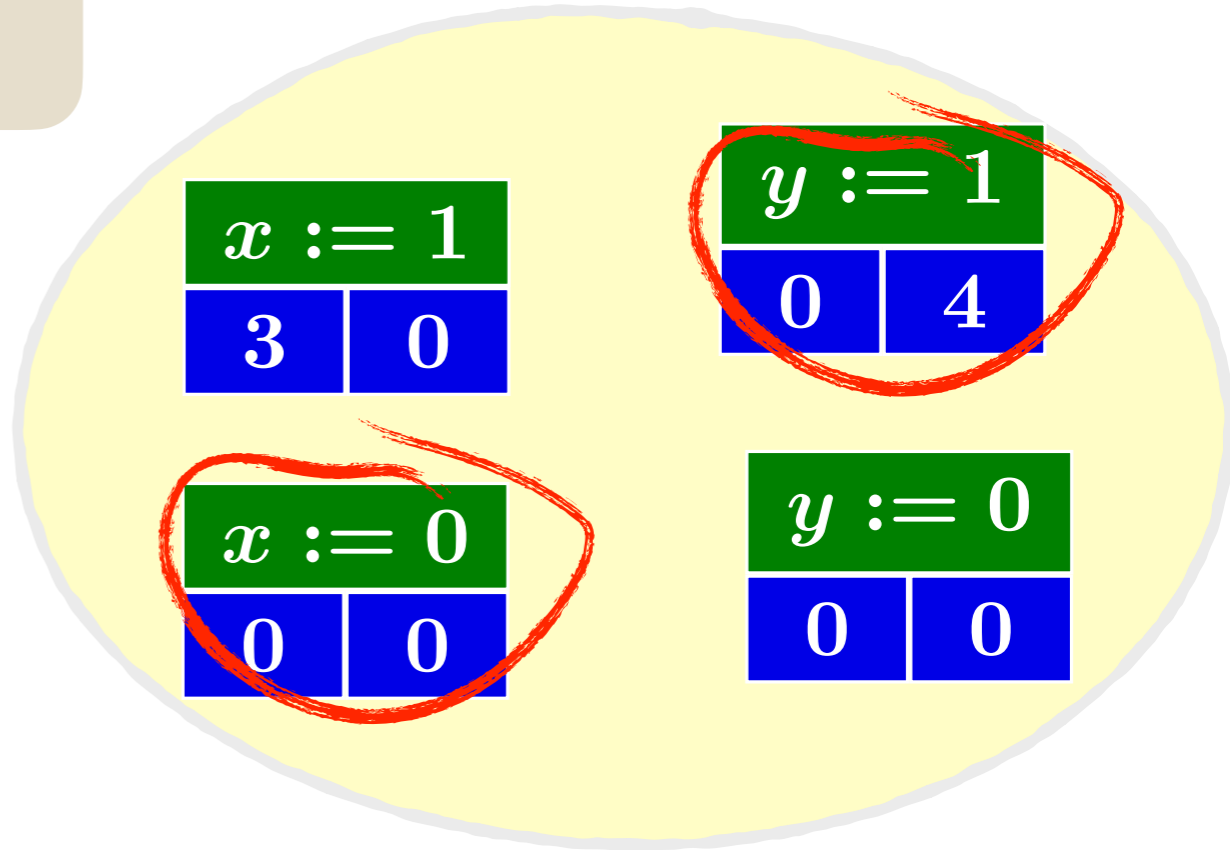
P_4
 $b_1 := 1$

0	4
---	---

 $b_2 := 0$

$(a_1 = 1) \wedge (a_2 = 0) \wedge (b_1 = 1) \wedge (b_2 = 0)$?

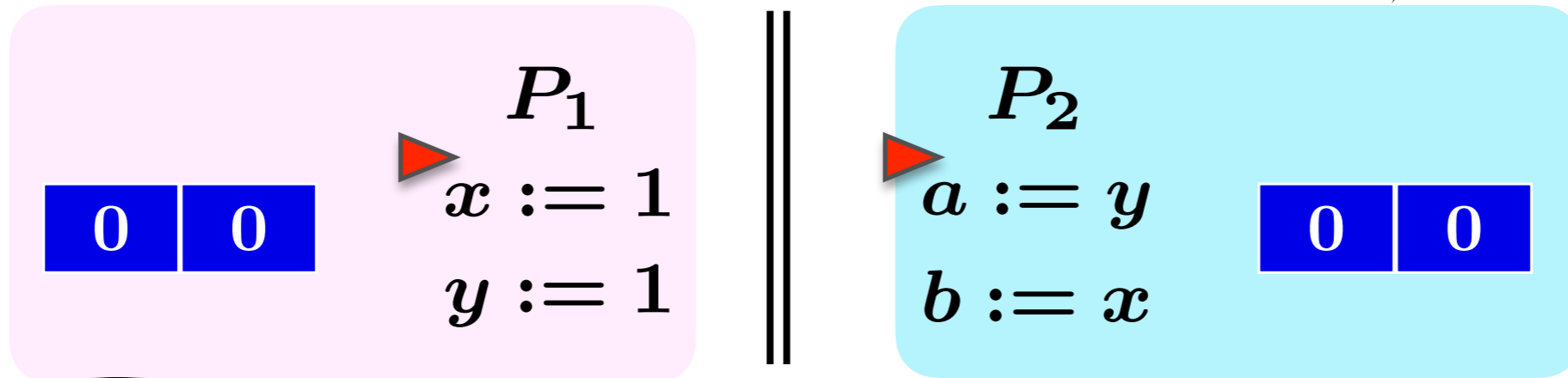
reachable



Release-Acquire (RA)

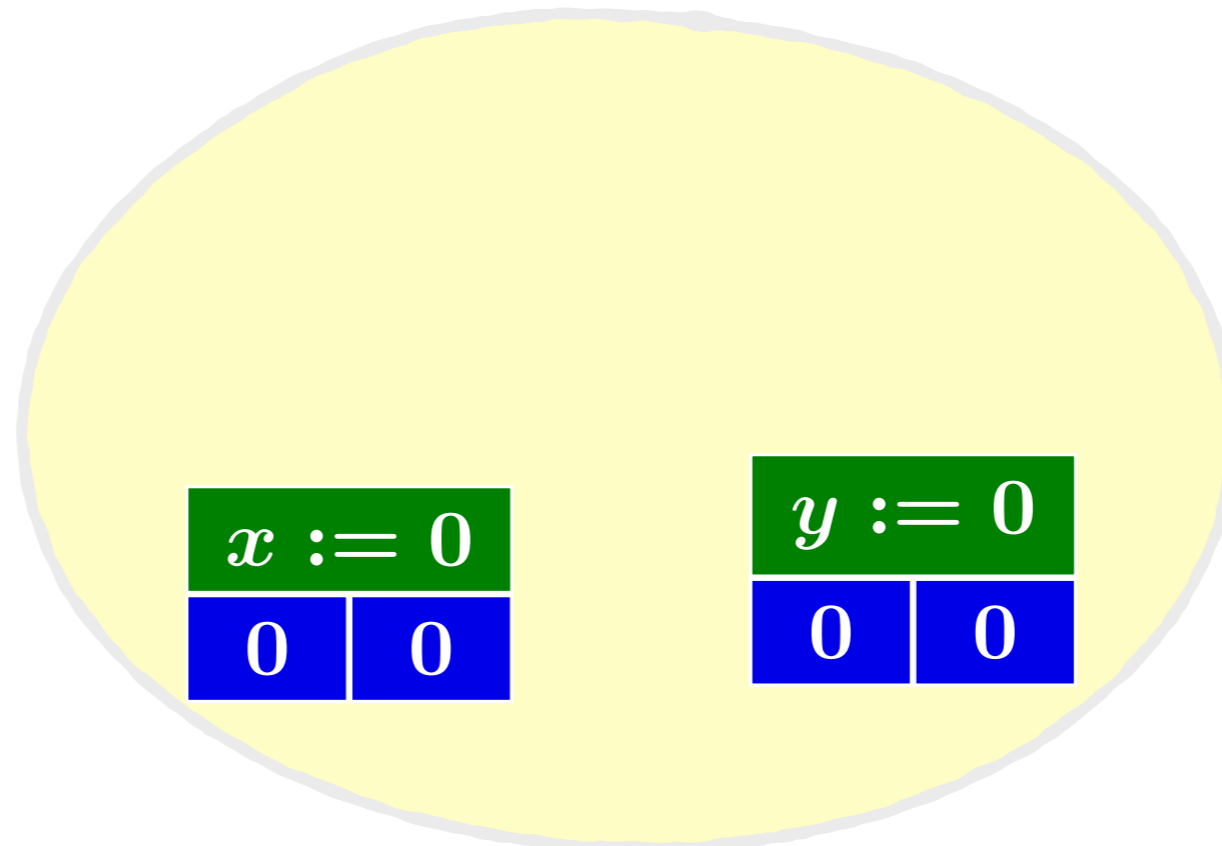
Message
Passing
(MP)

Initially: $x = y = 0$



assertion

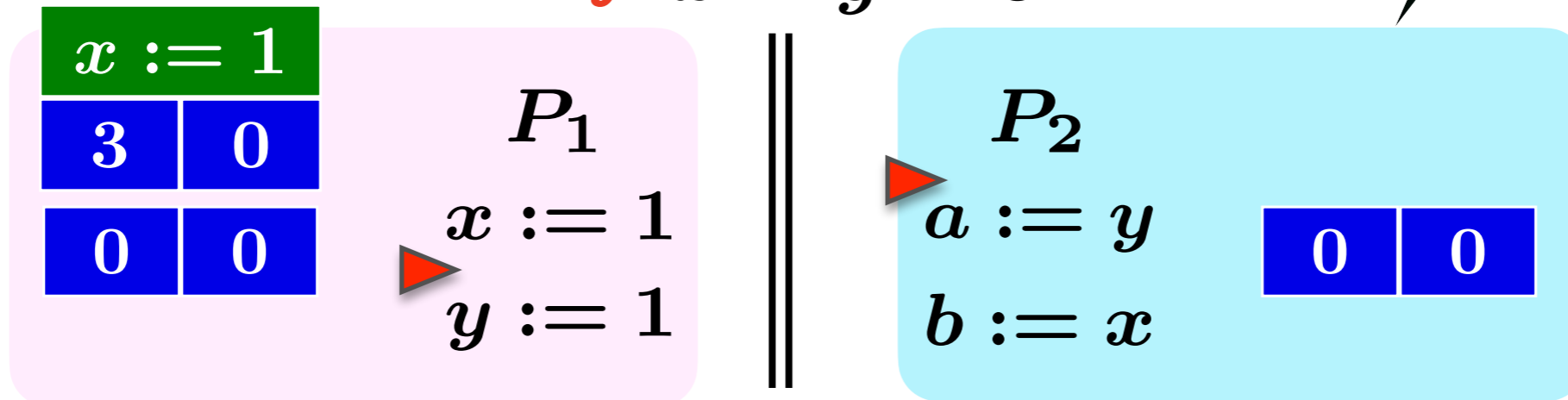
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

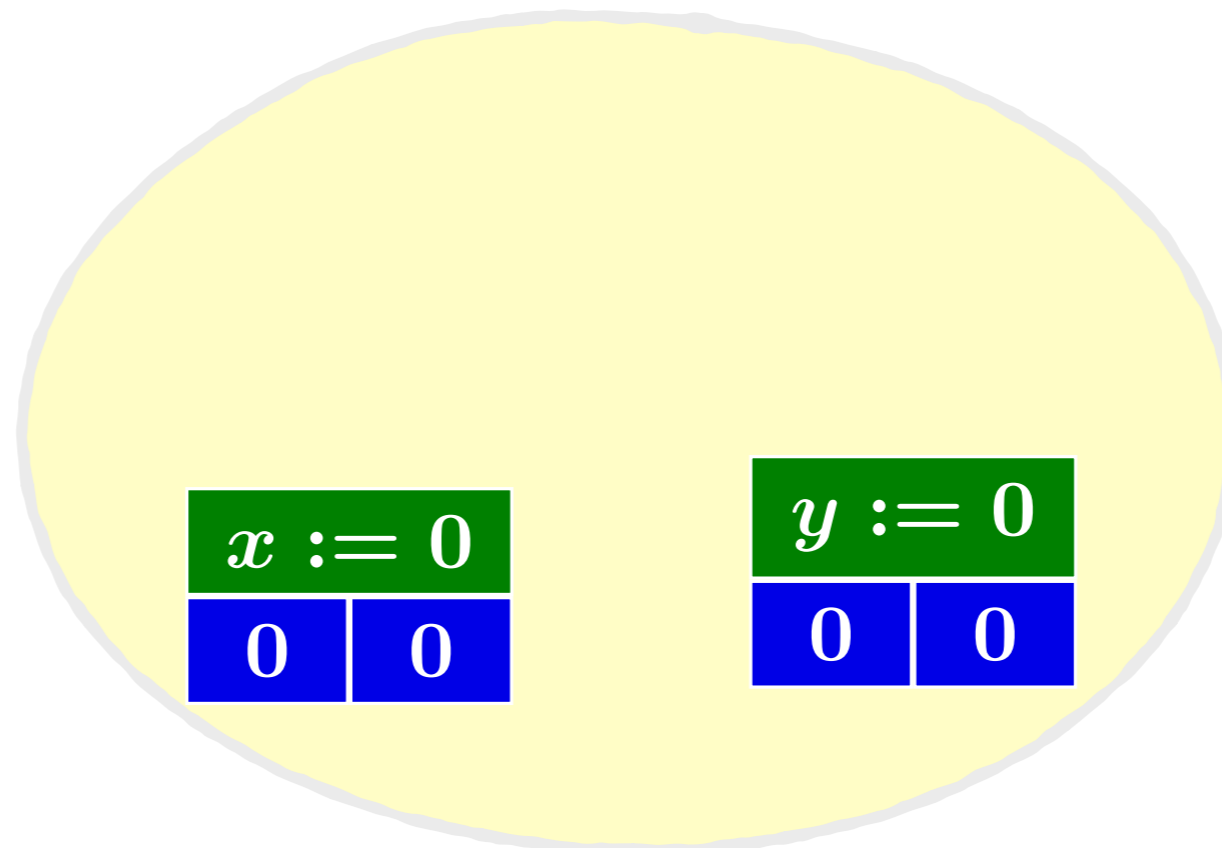
Message Passing (MP)

Initially: $x = y = 0$



assertion

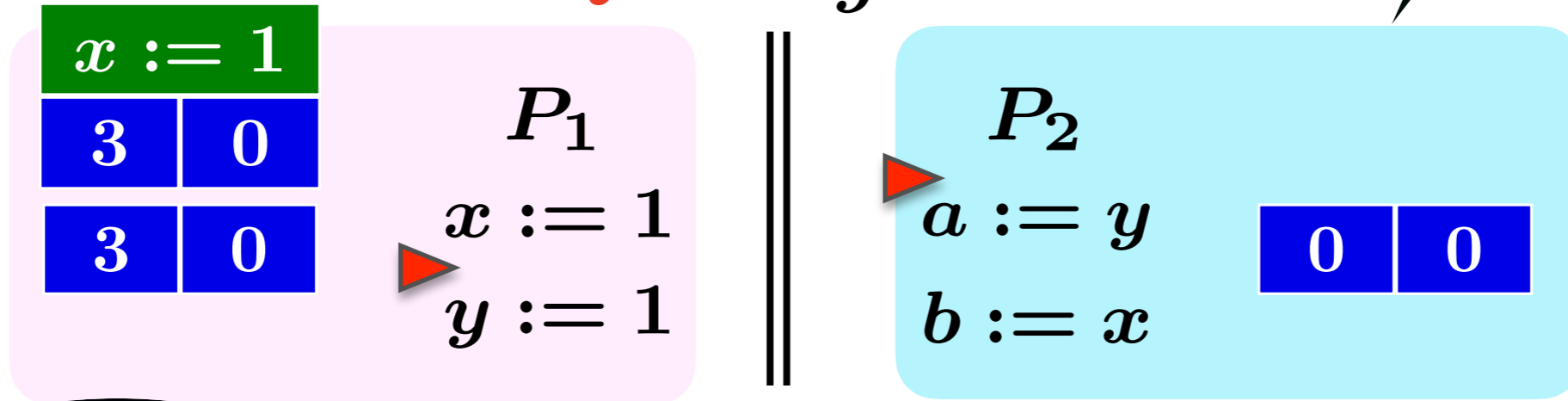
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

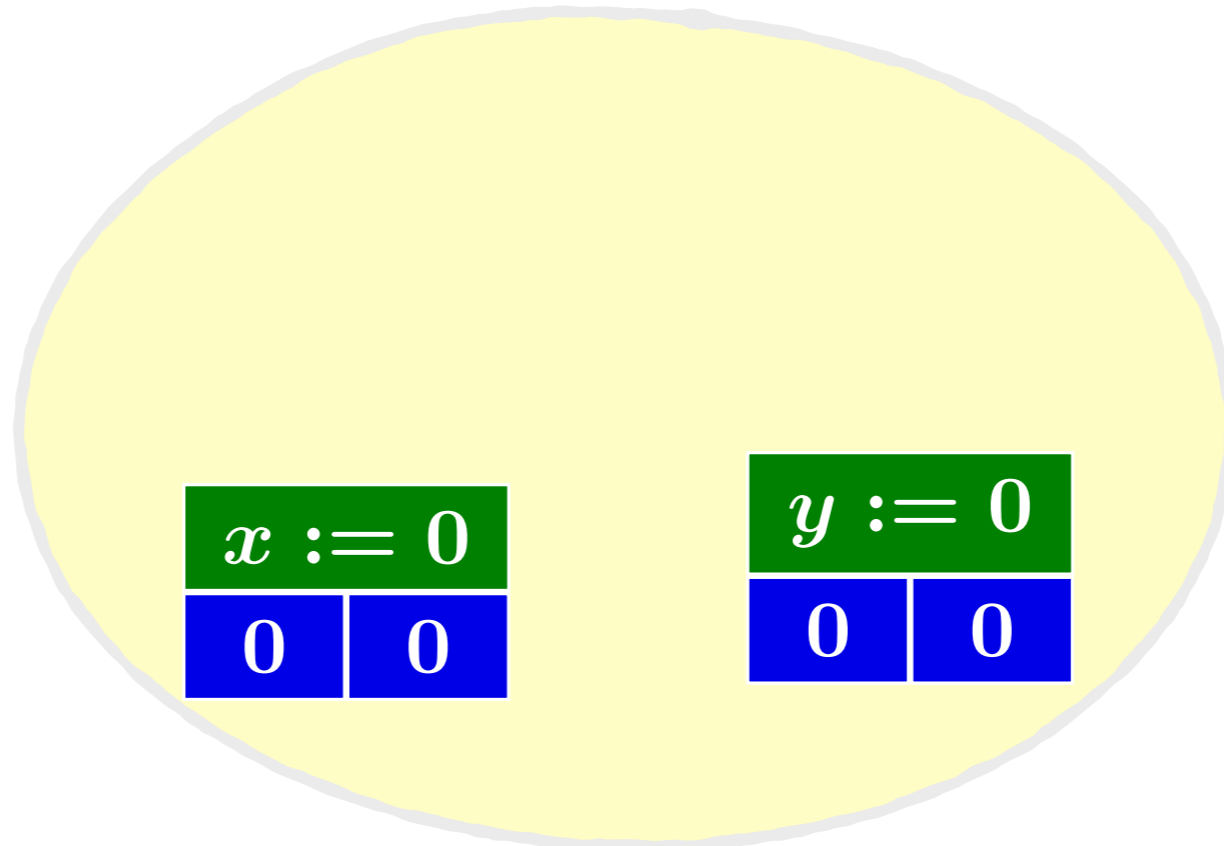
Message Passing (MP)

Initially: $x = y = 0$



assertion

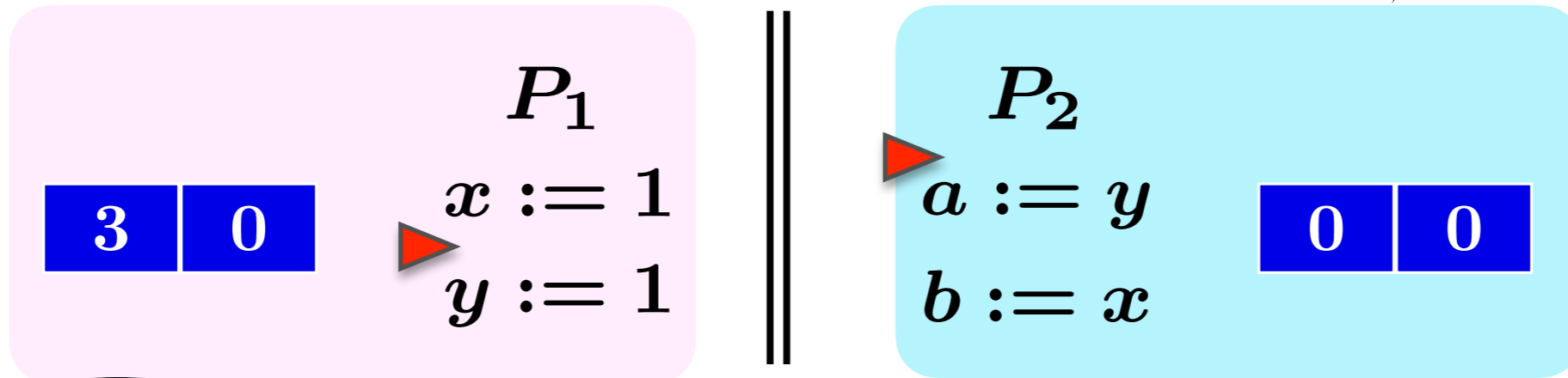
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

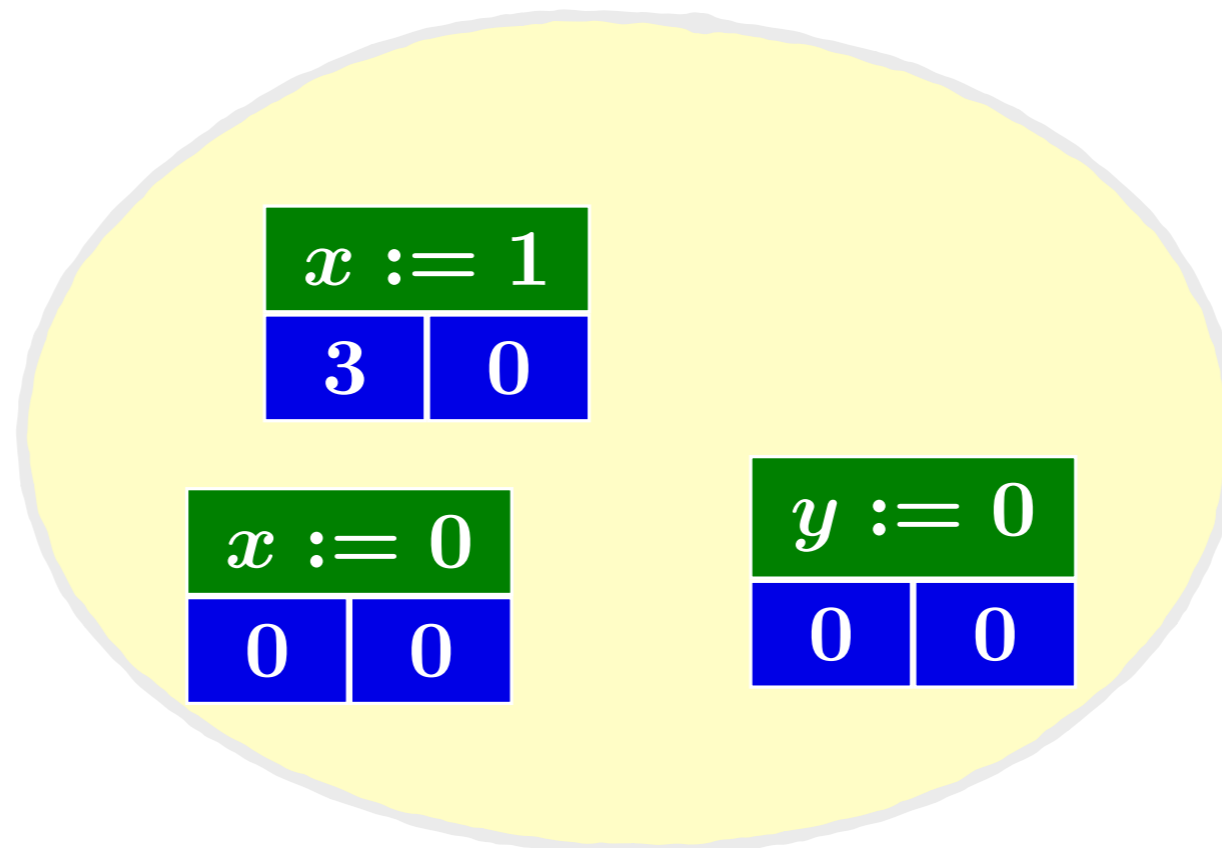
Message
Passing
(MP)

Initially: $x = y = 0$



assertion

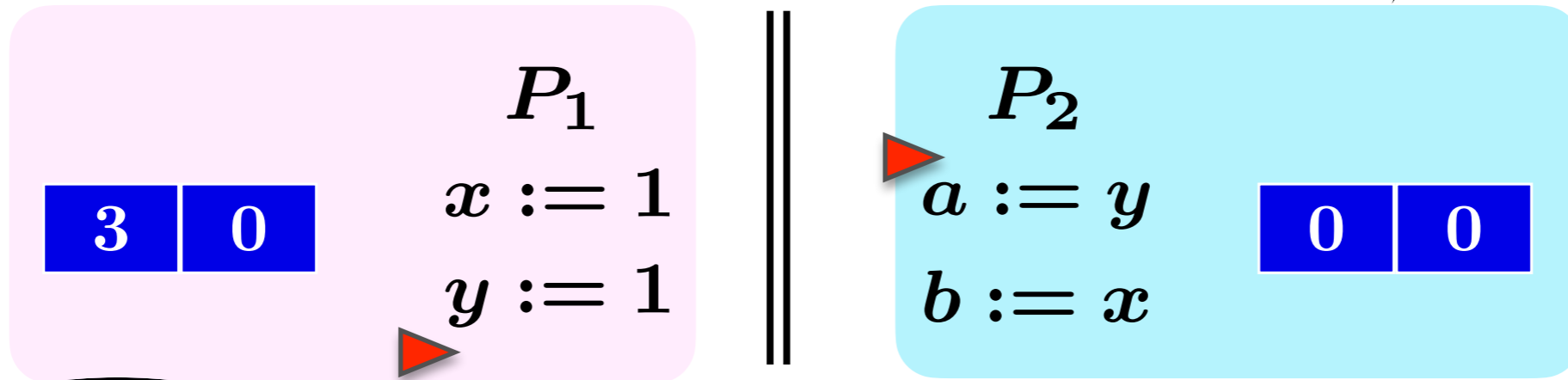
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

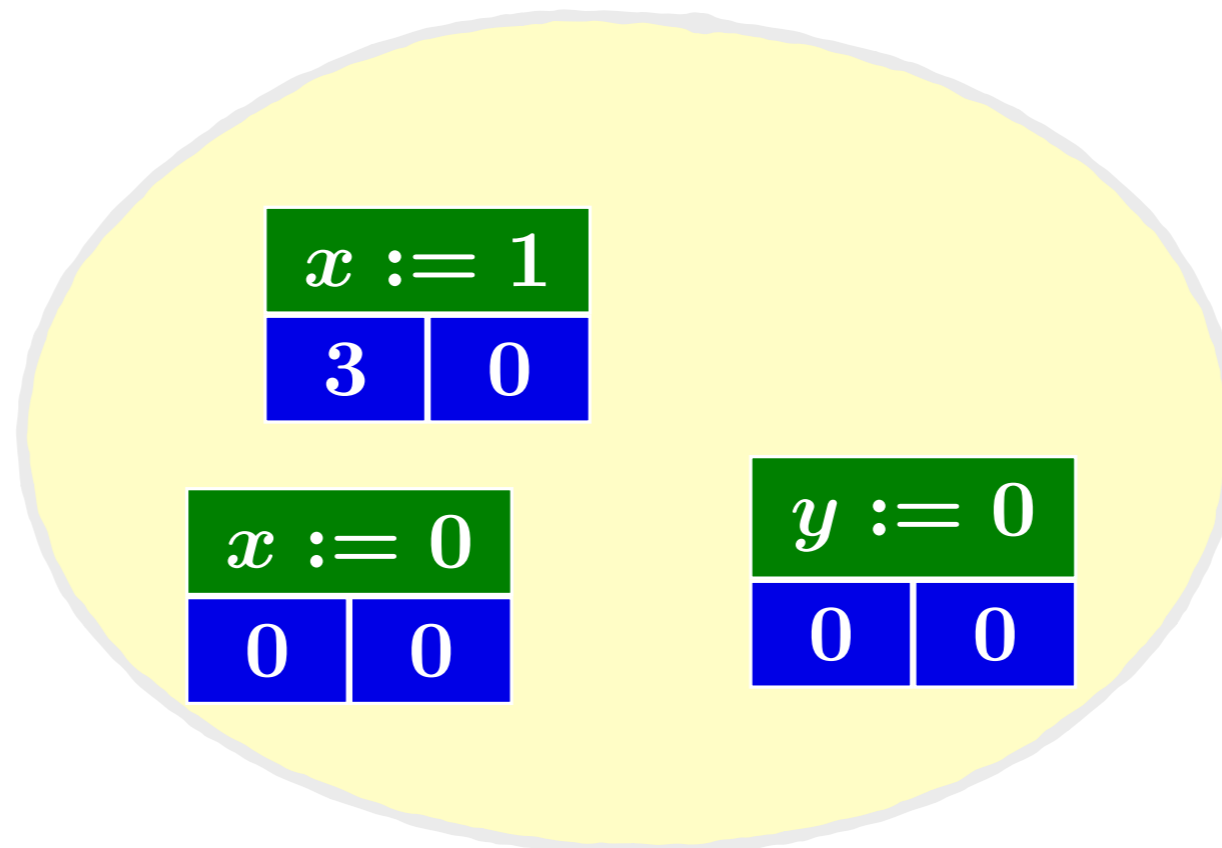
Message
Passing
(MP)

Initially: $x = y = 0$



assertion

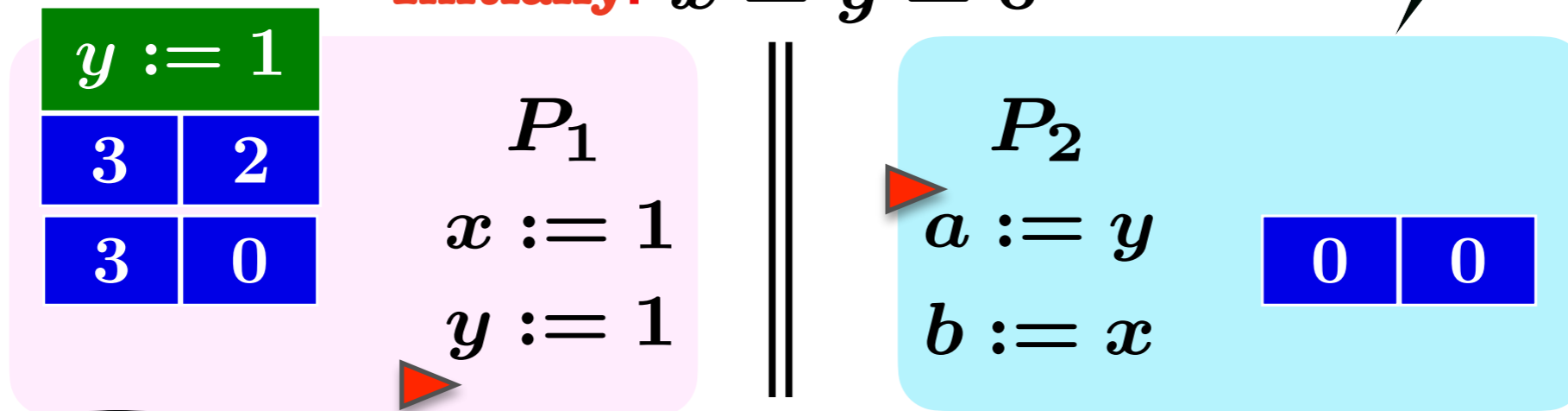
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

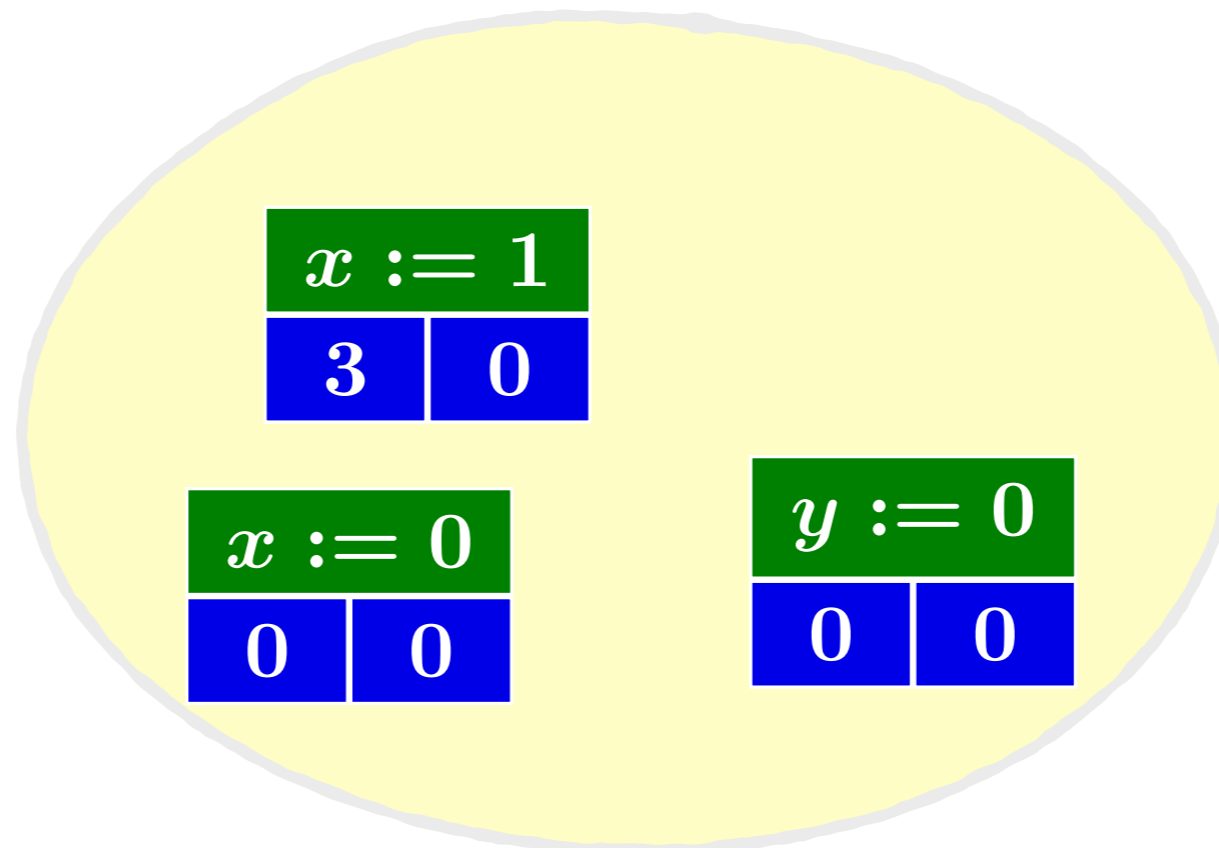
Message
Passing
(MP)

Initially: $x = y = 0$



assertion

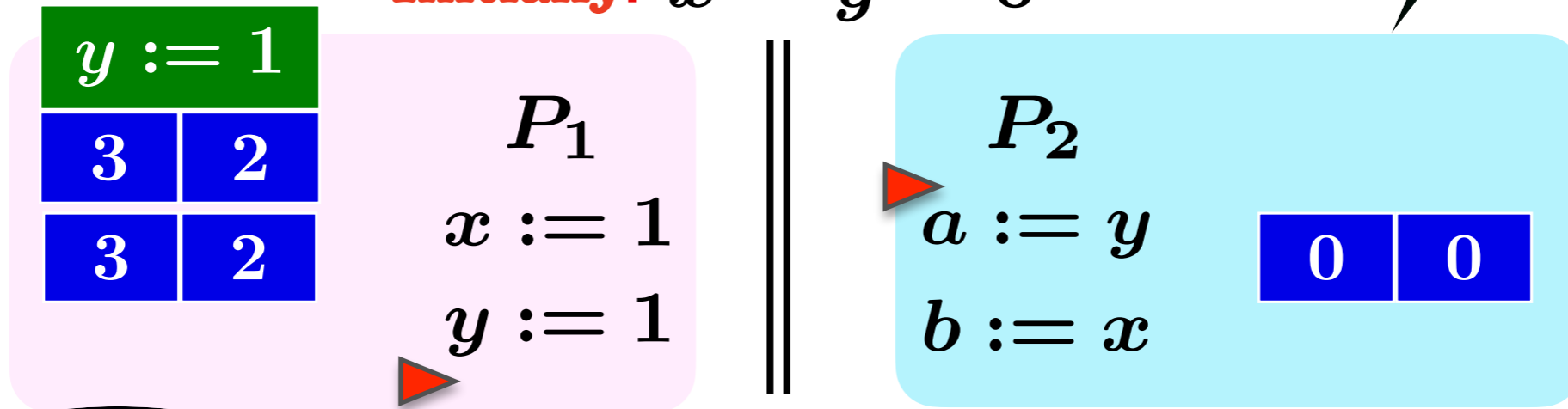
$(a = 1) \wedge (b = 0)?$



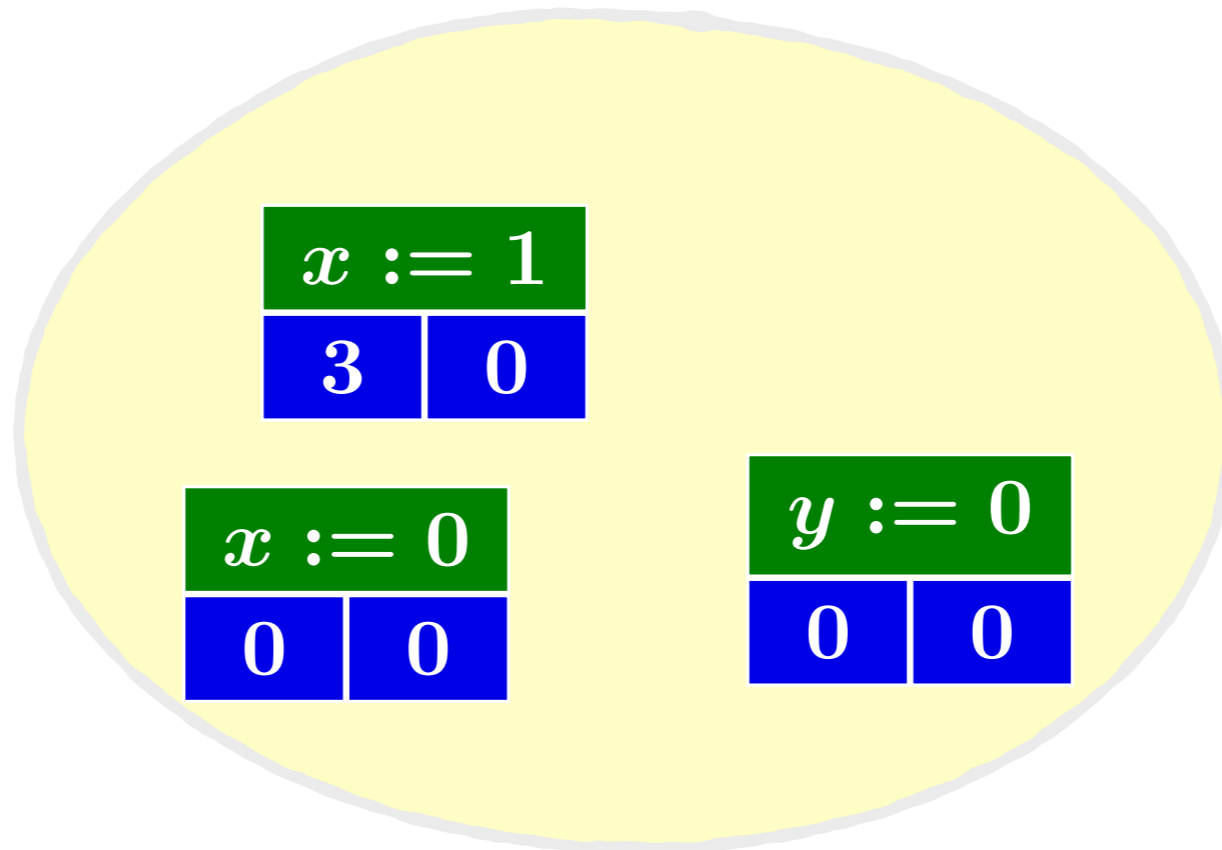
Release-Acquire (RA)

Message Passing (MP)

Initially: $x = y = 0$



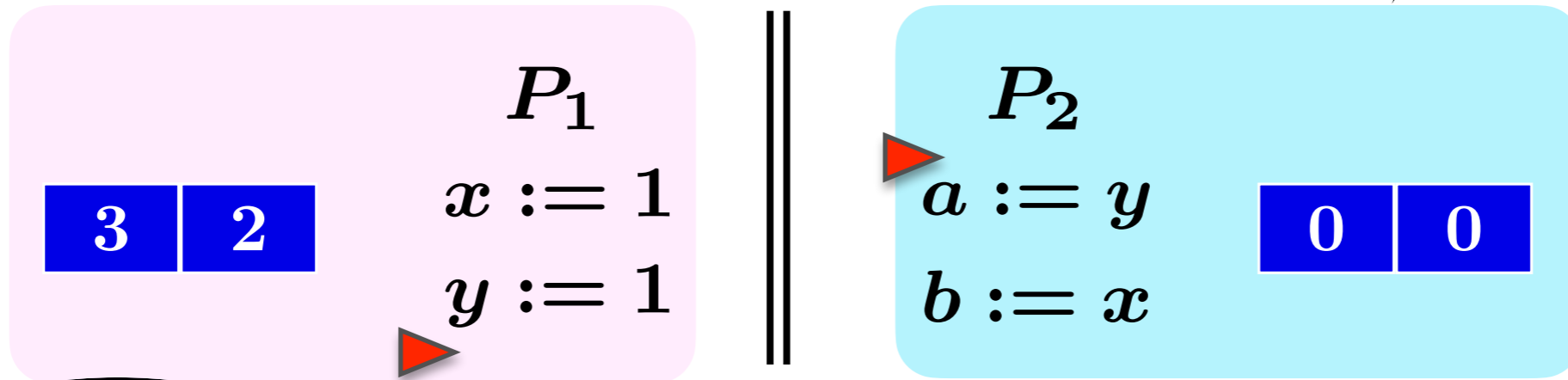
assertion $(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

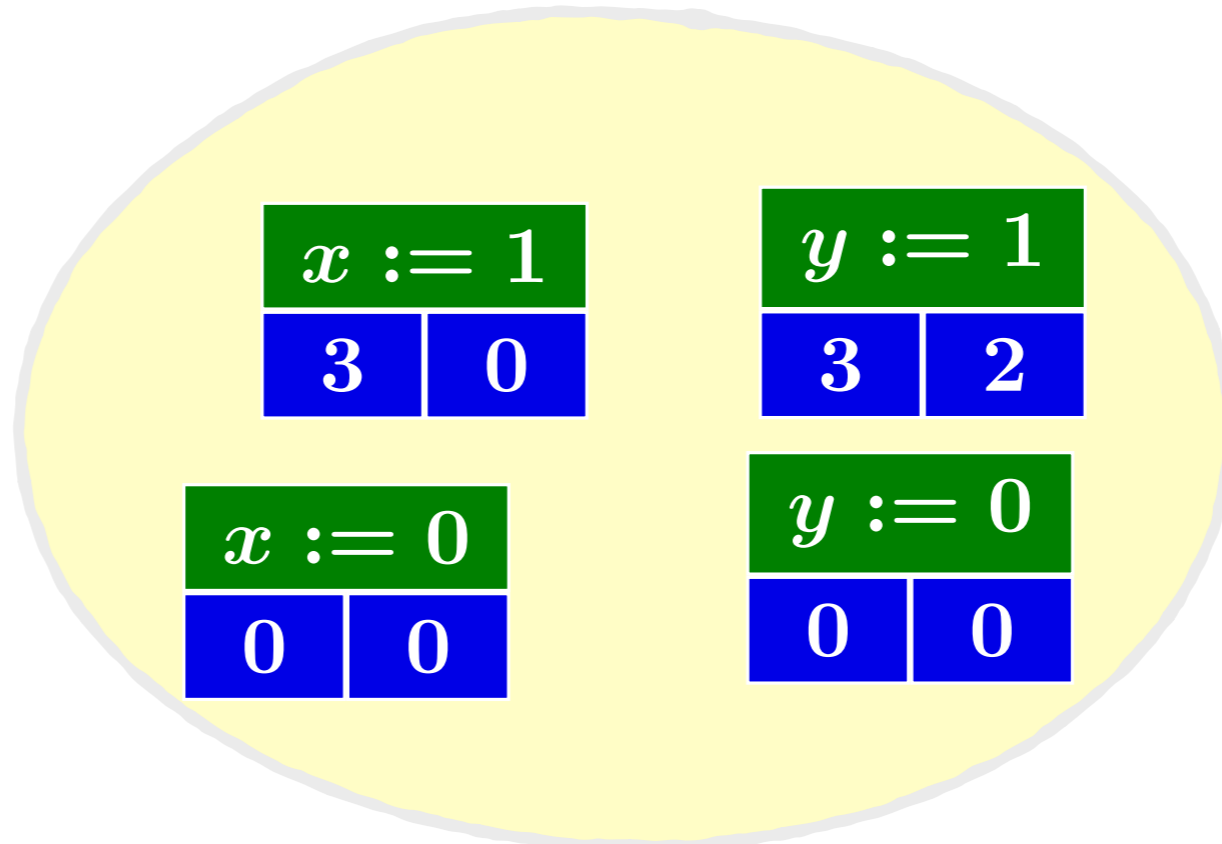
Message Passing (MP)

Initially: $x = y = 0$



assertion

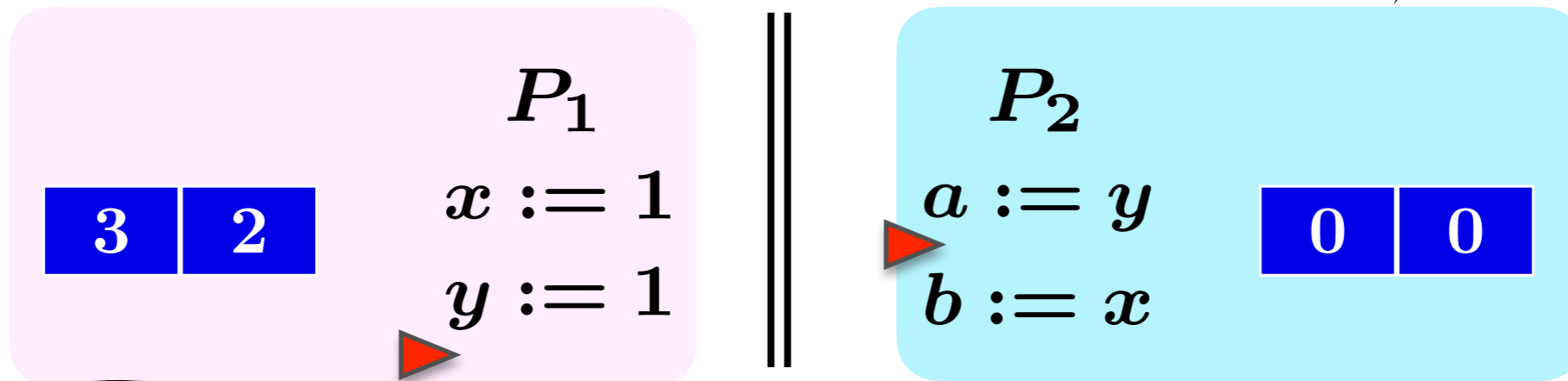
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

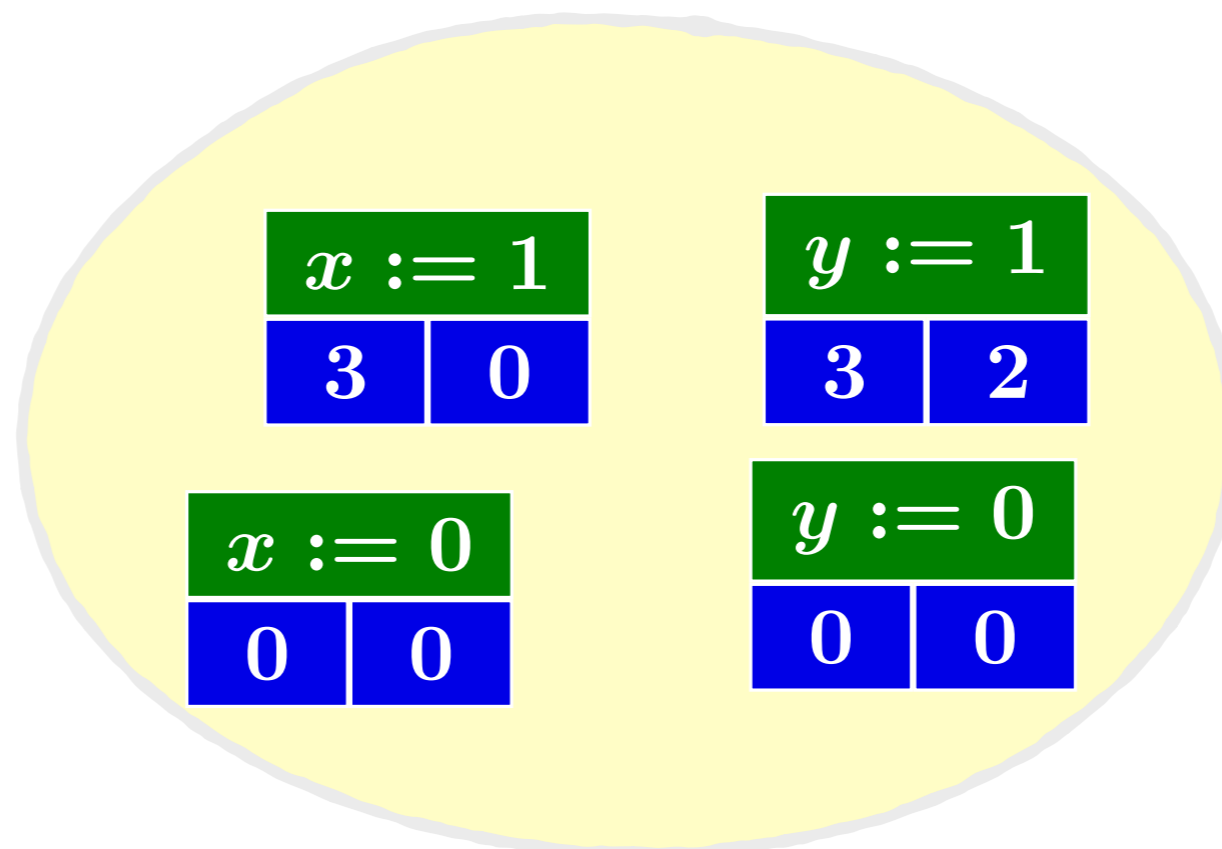
Message Passing (MP)

Initially: $x = y = 0$



assertion

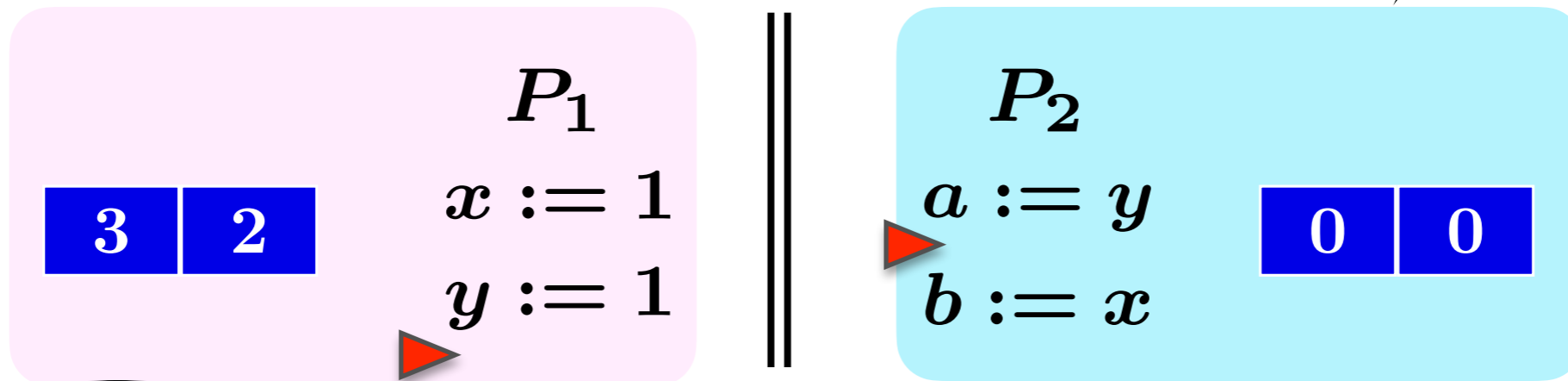
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

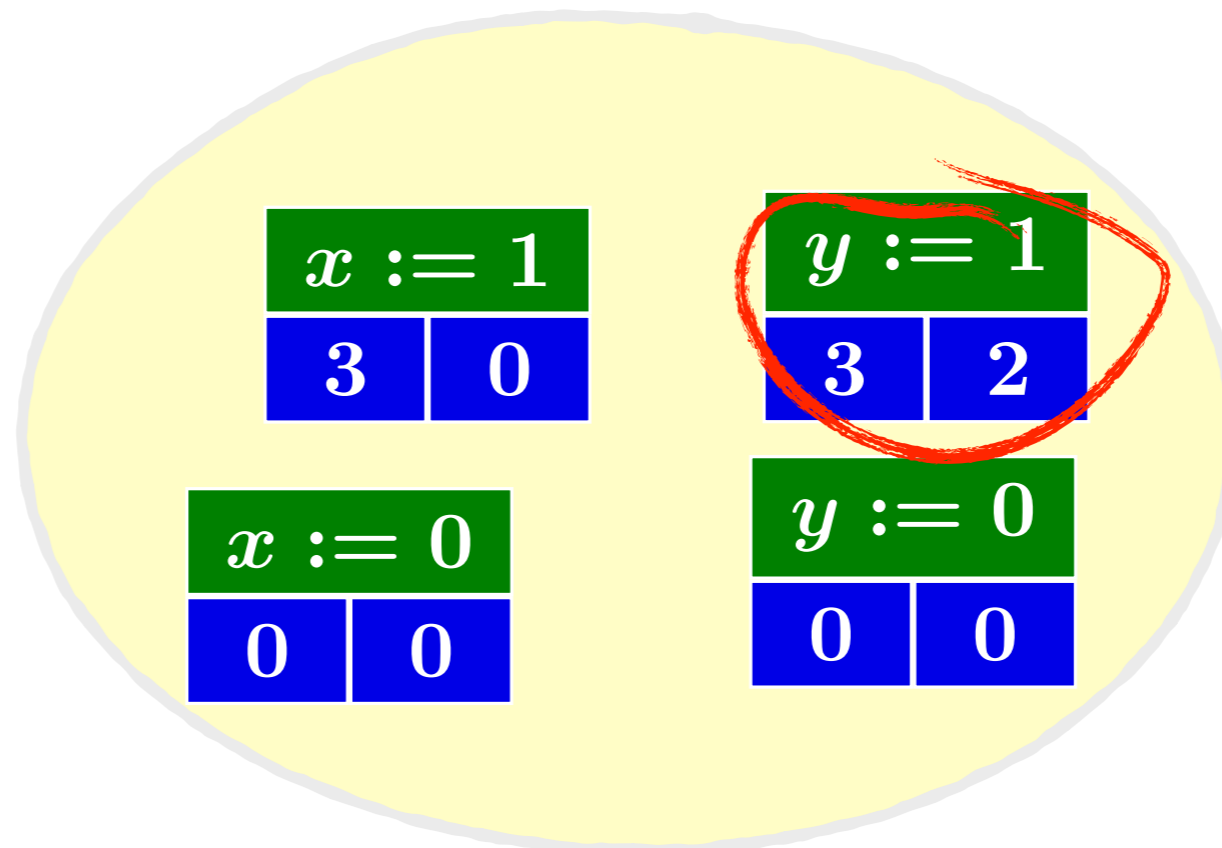
Message Passing (MP)

Initially: $x = y = 0$



assertion

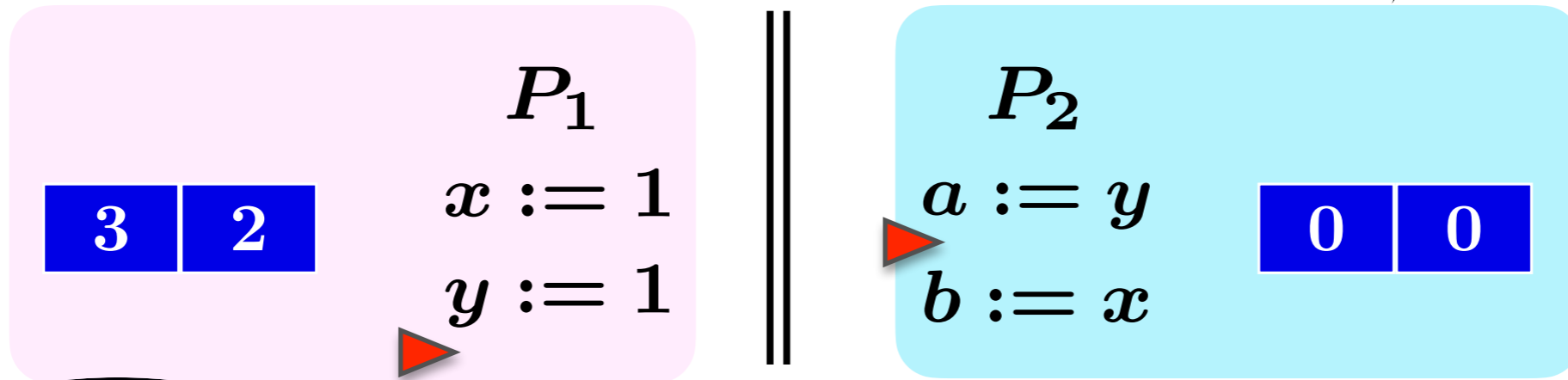
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

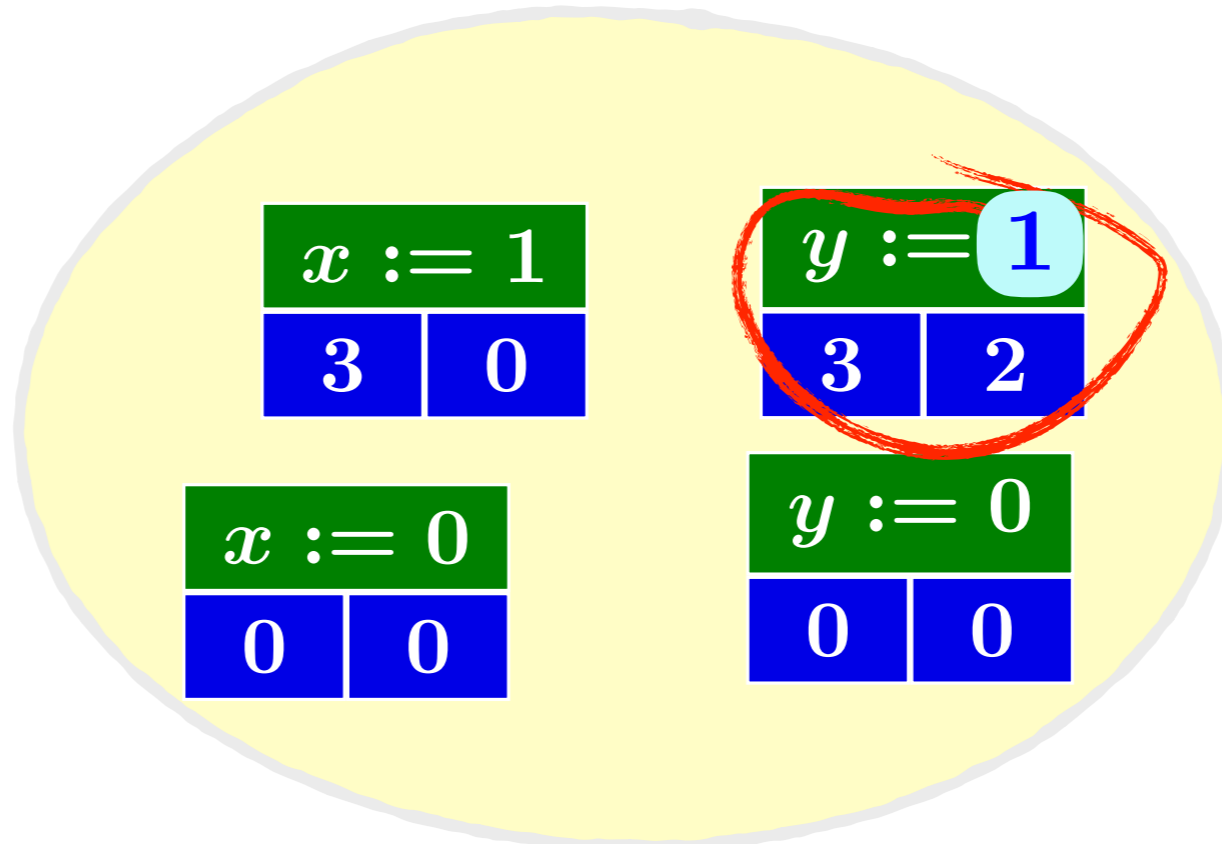
Message Passing (MP)

Initially: $x = y = 0$



assertion

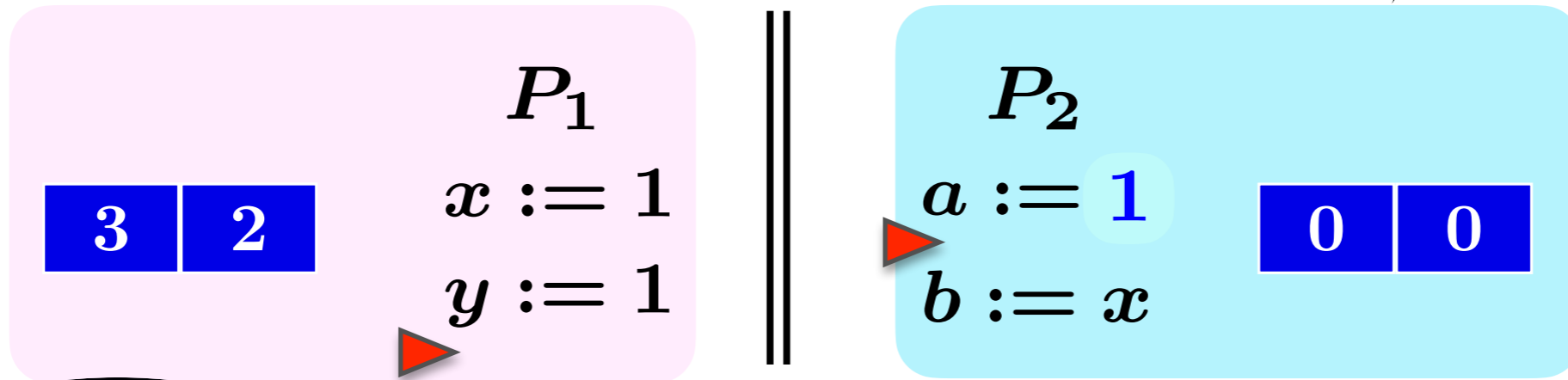
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

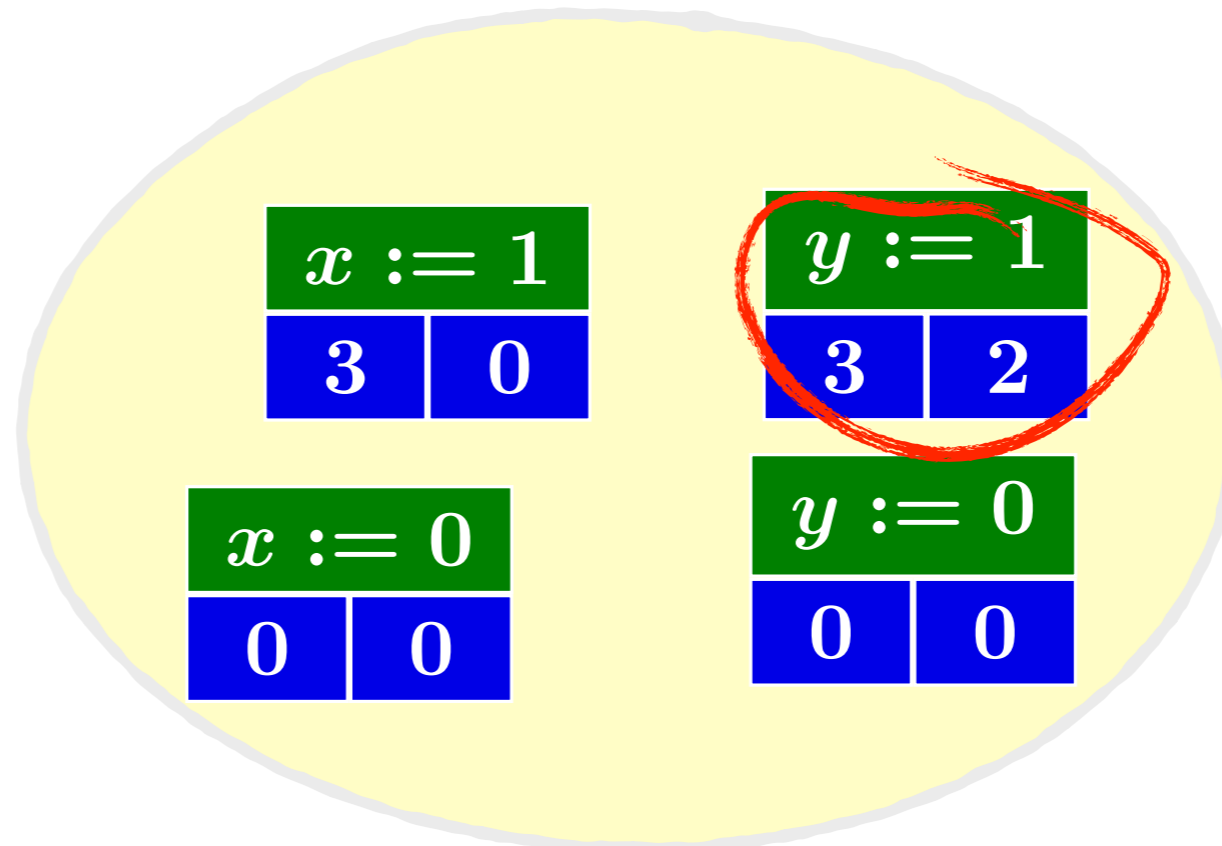
Message Passing (MP)

Initially: $x = y = 0$



assertion

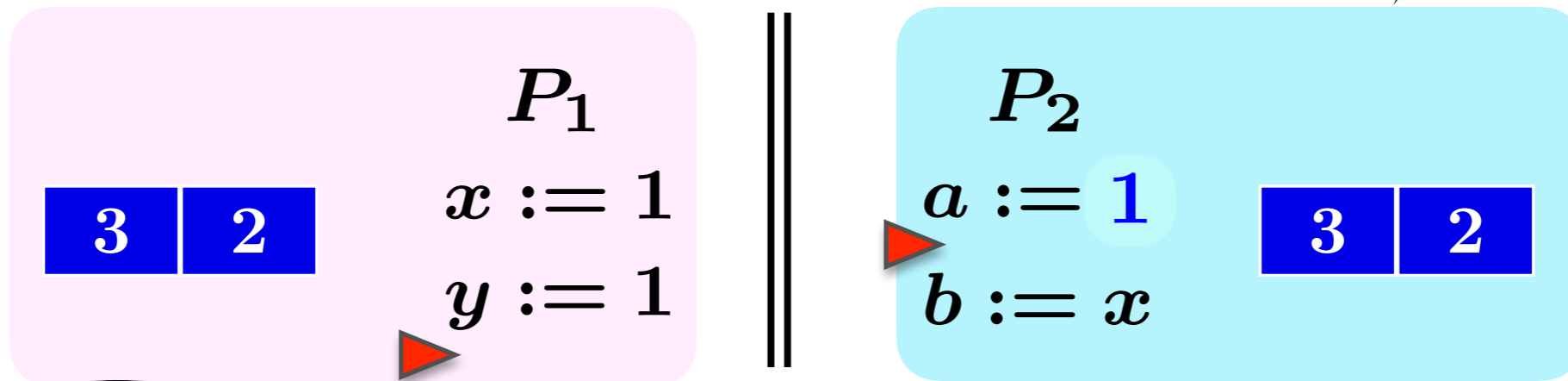
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

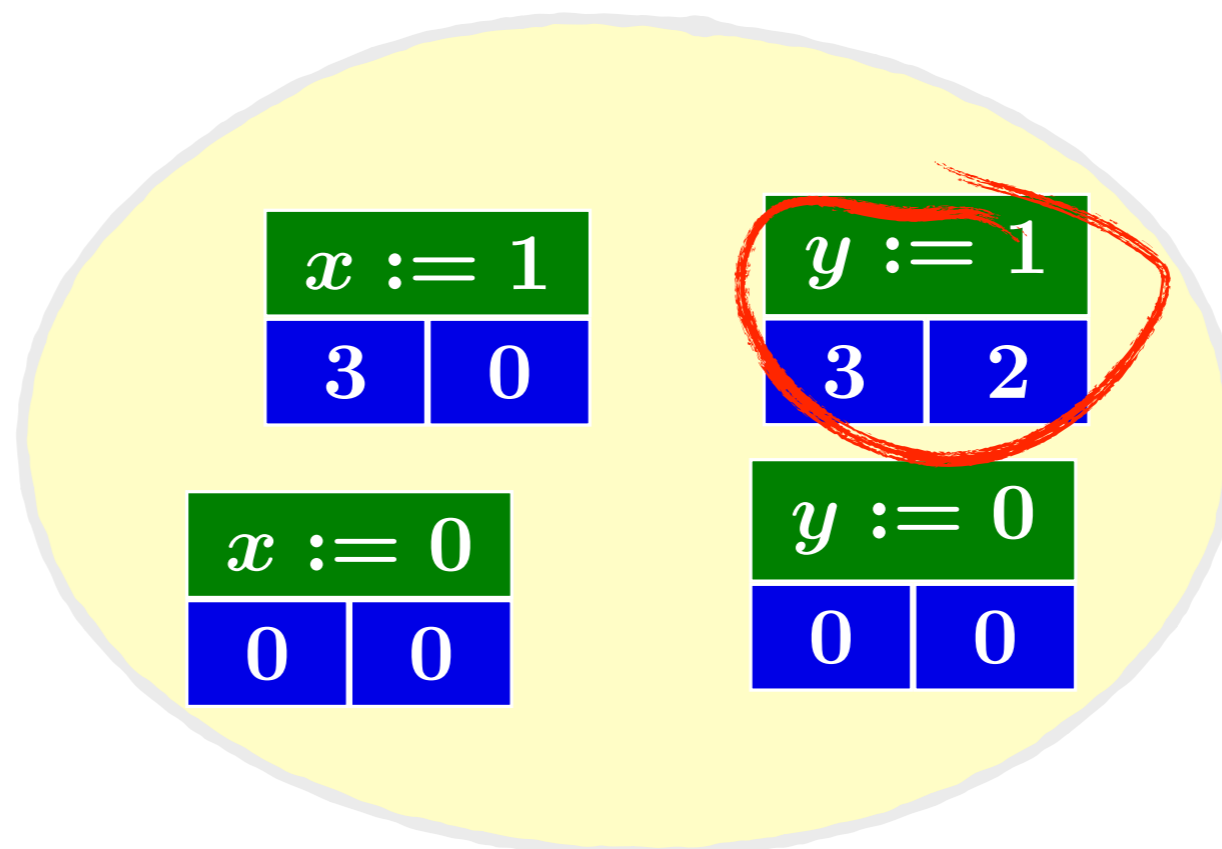
Message Passing (MP)

Initially: $x = y = 0$



assertion

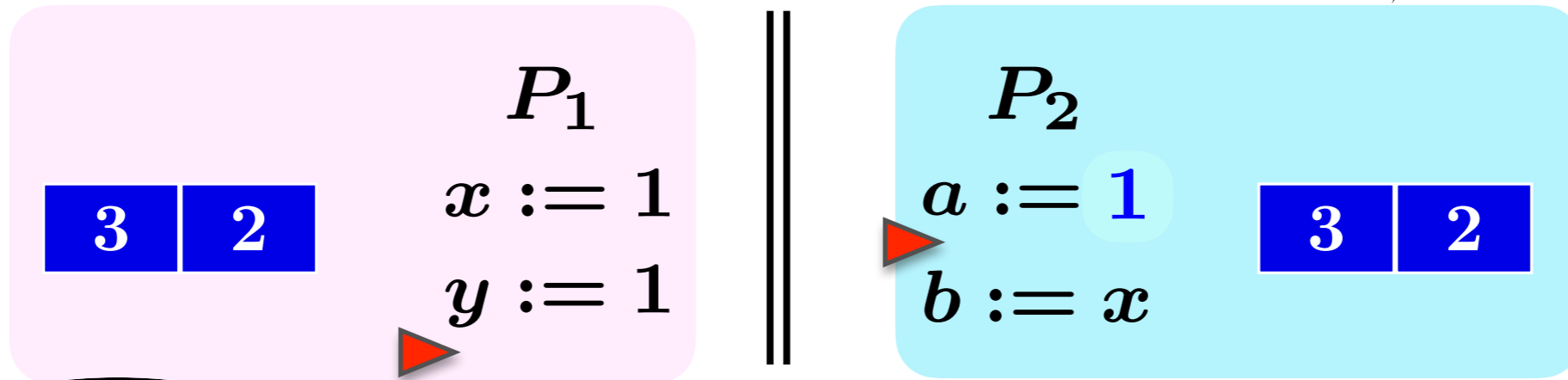
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

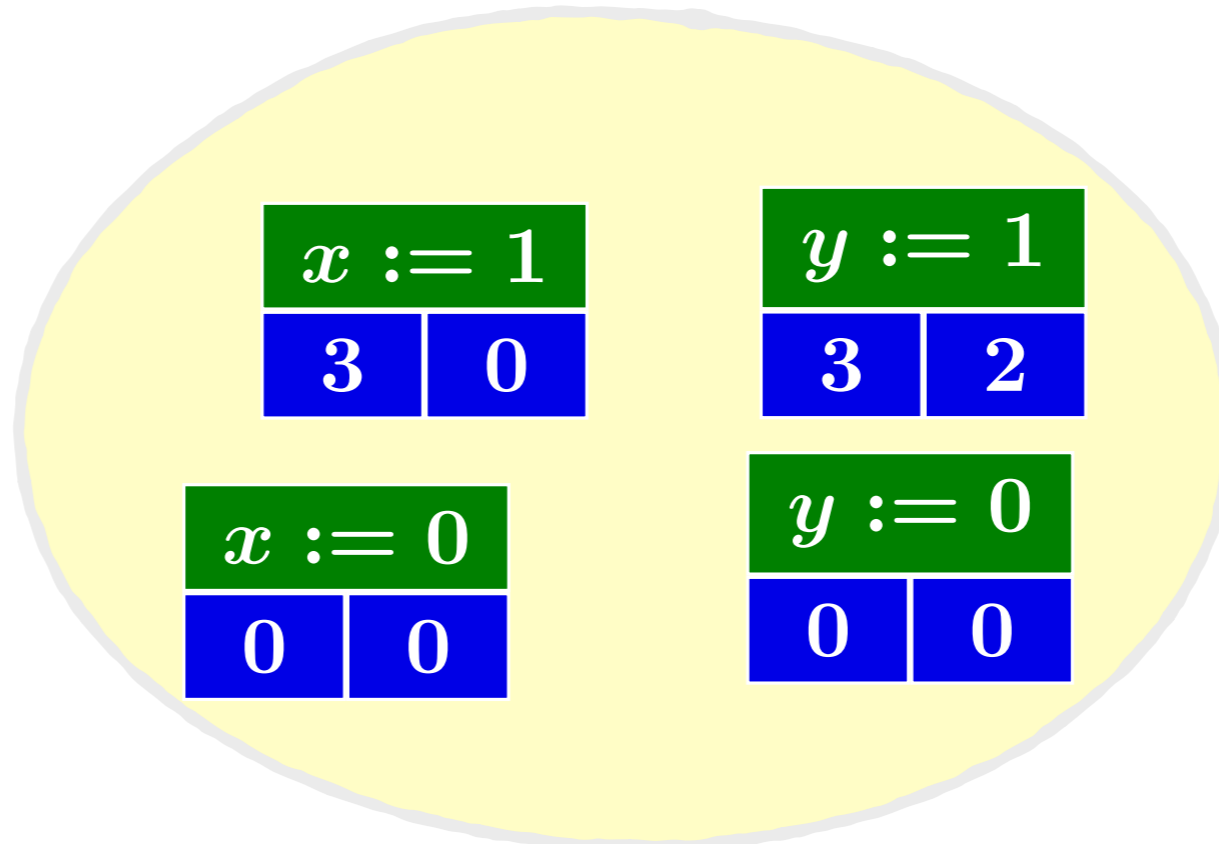
Message Passing (MP)

Initially: $x = y = 0$



assertion

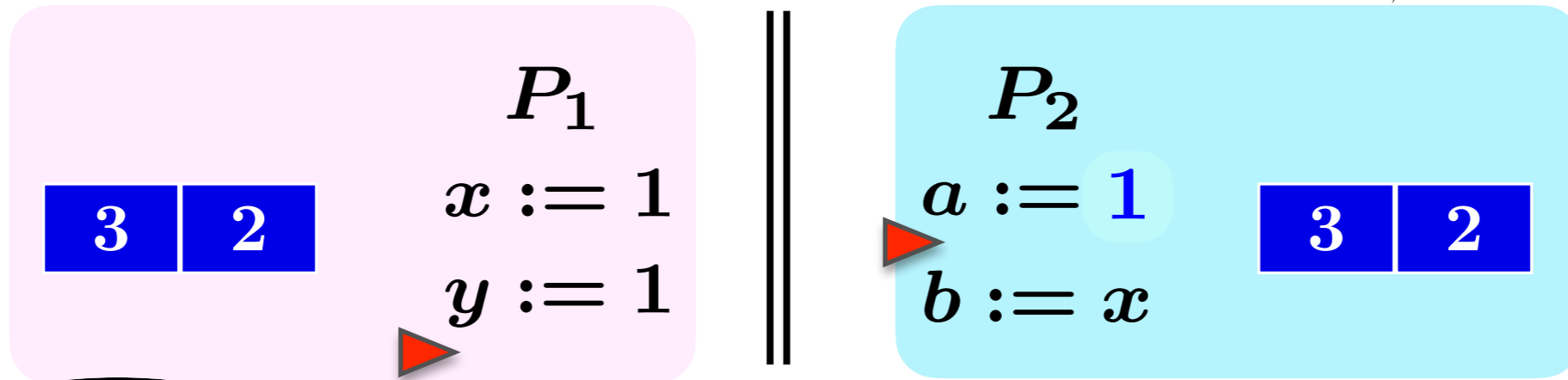
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

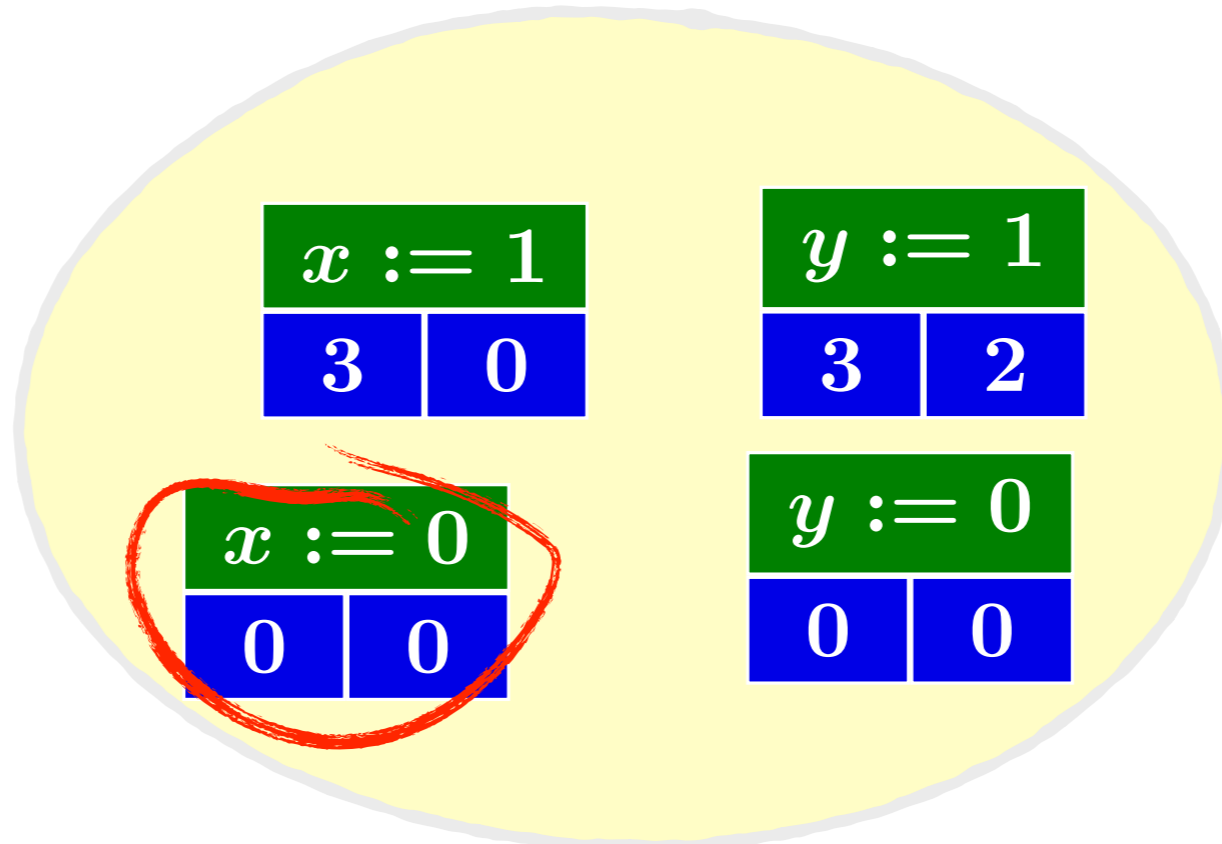
Message Passing (MP)

Initially: $x = y = 0$



assertion

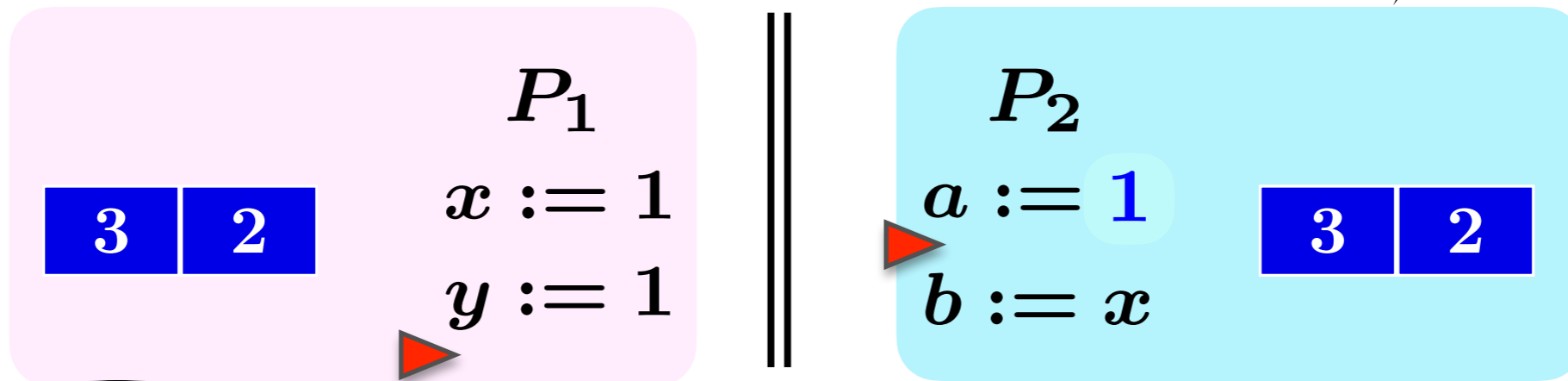
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

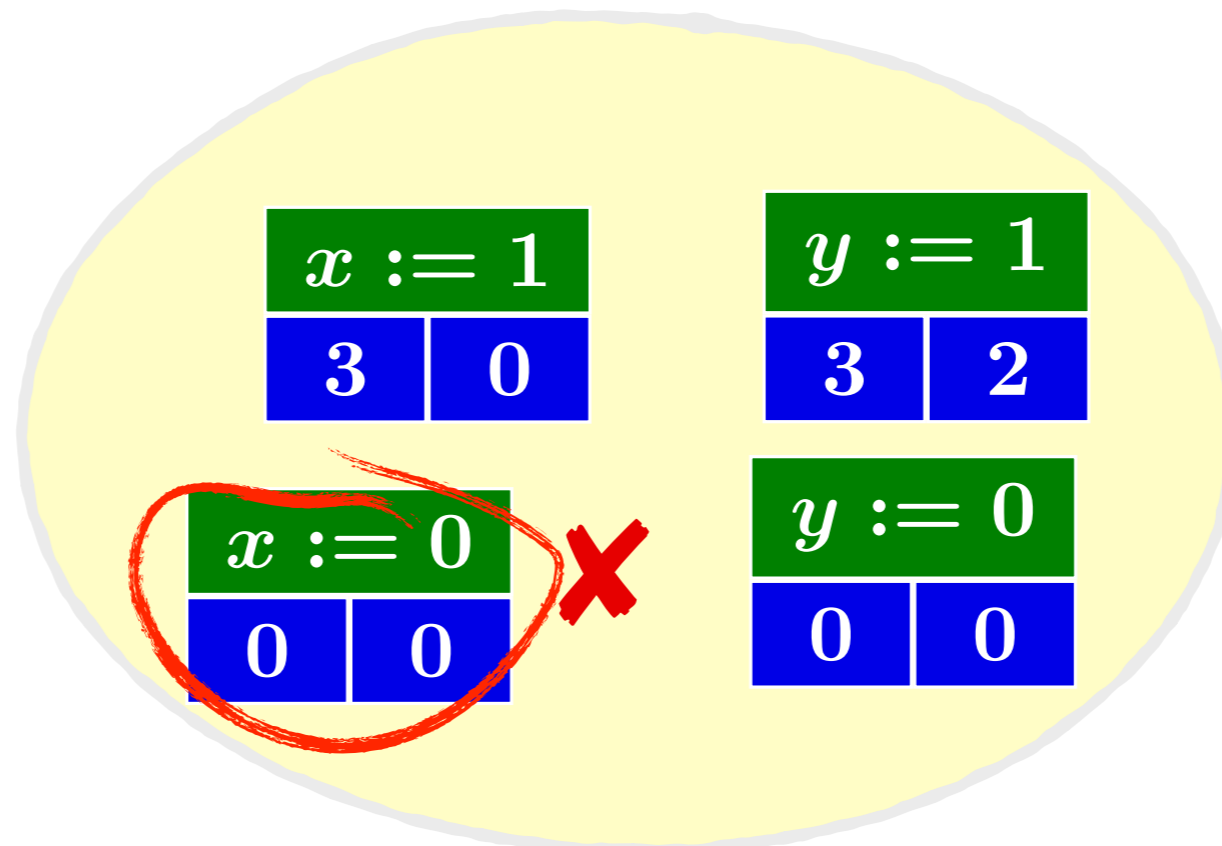
Message
Passing
(MP)

Initially: $x = y = 0$



assertion

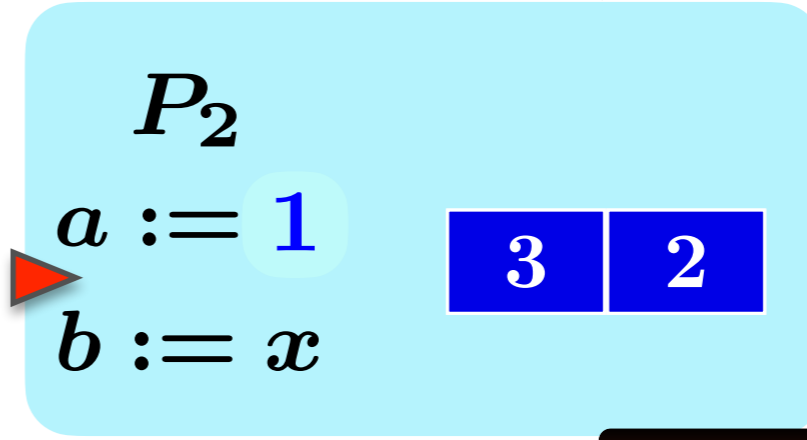
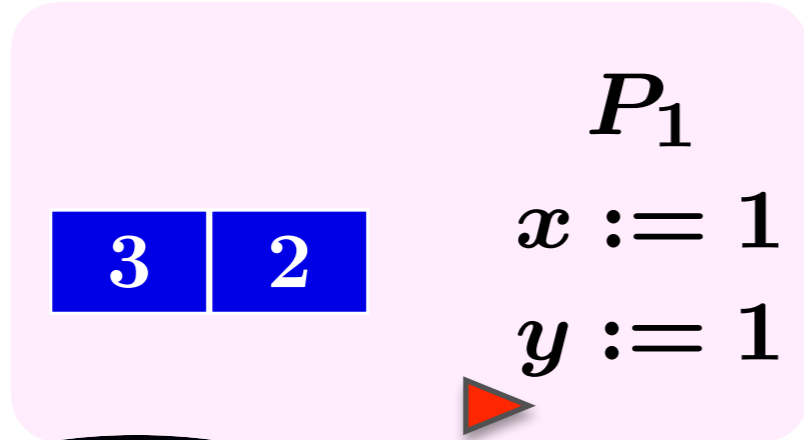
$(a = 1) \wedge (b = 0)?$



Release-Acquire (RA)

Message Passing (MP)

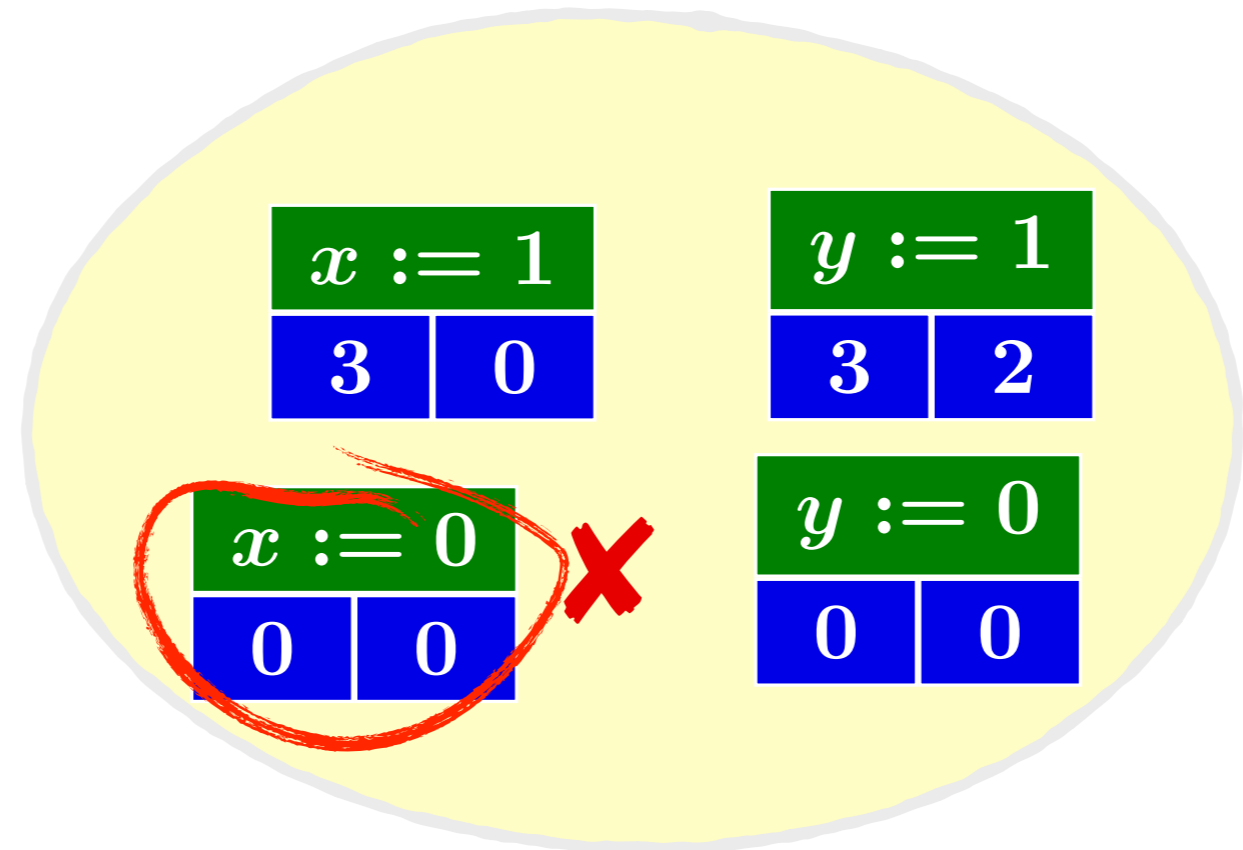
Initially: $x = y = 0$



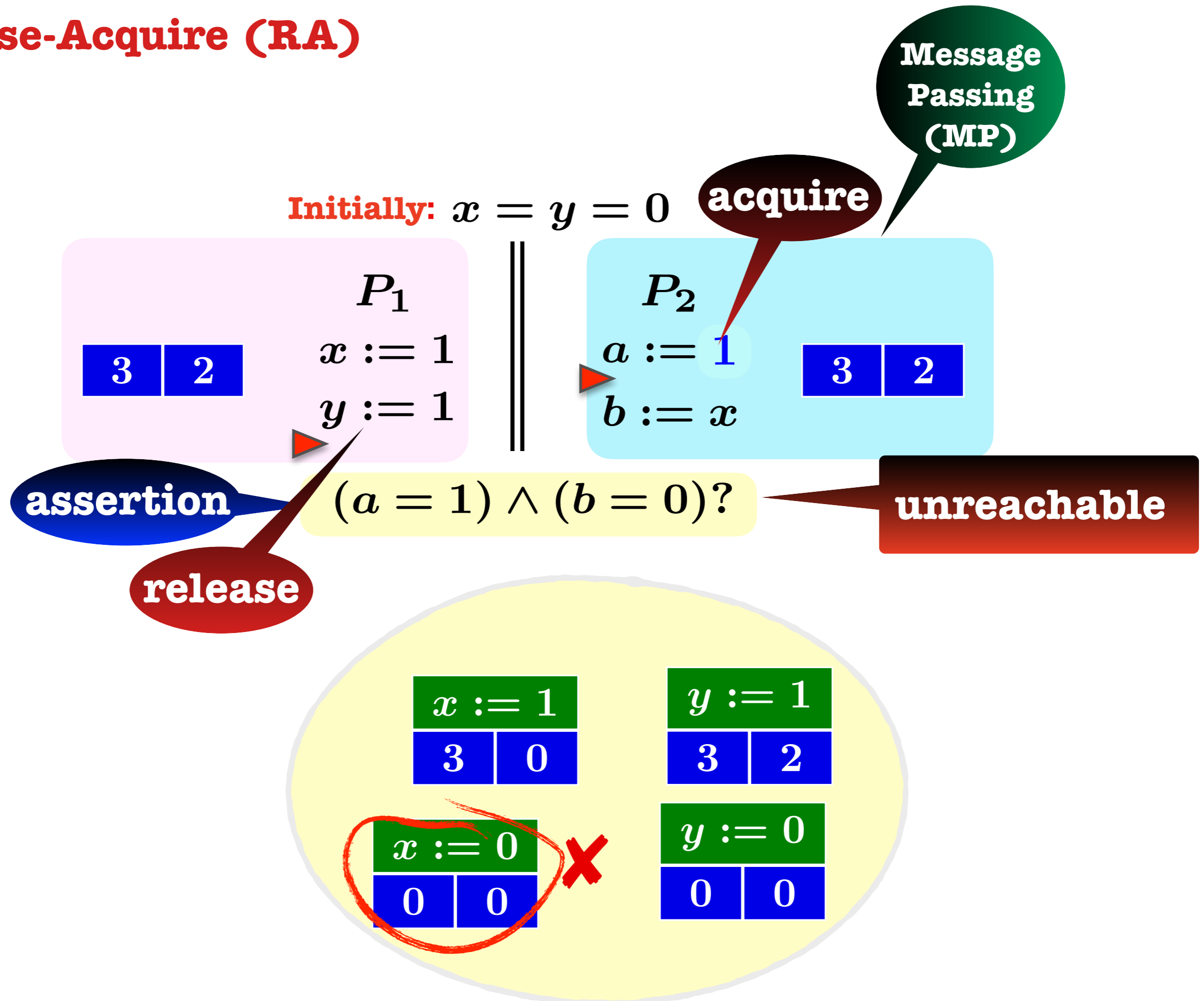
assertion

$(a = 1) \wedge (b = 0)?$

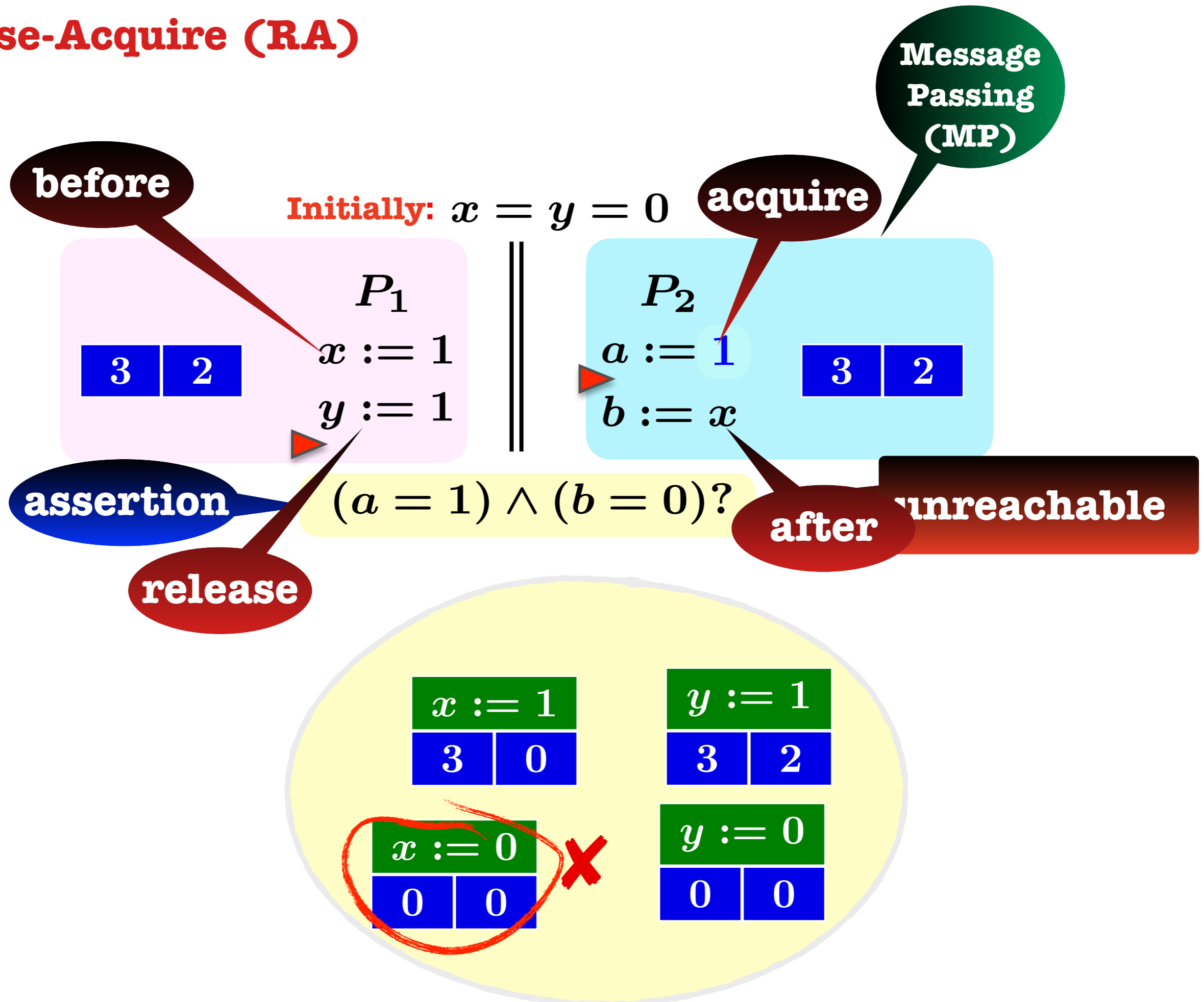
unreachable



Release-Acquire (RA)



Release-Acquire (RA)



Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - Dynamic Partial-Order Reduction (DPOR)
 - Experiments
- **Conclusions**

P_1

```
while (1)  
  write: x=1
```

P_1

```
while (1)  
  write: x=1
```

$x := 1$

$x := 1$

$x := 1$

...

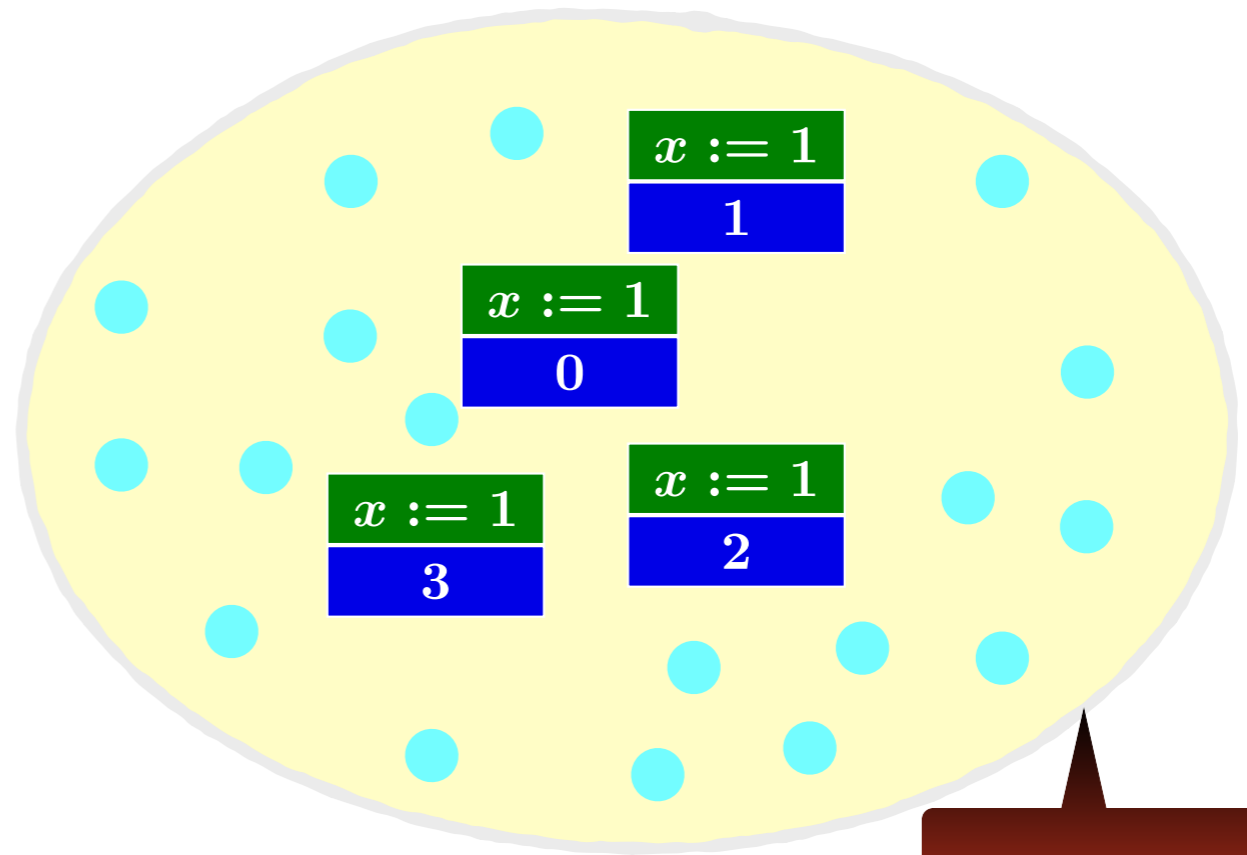
$x := 1$

...

P_1

```
while (1)
  write: x=1
```

```
x := 1
x := 1
x := 1
...
x := 1
...
```



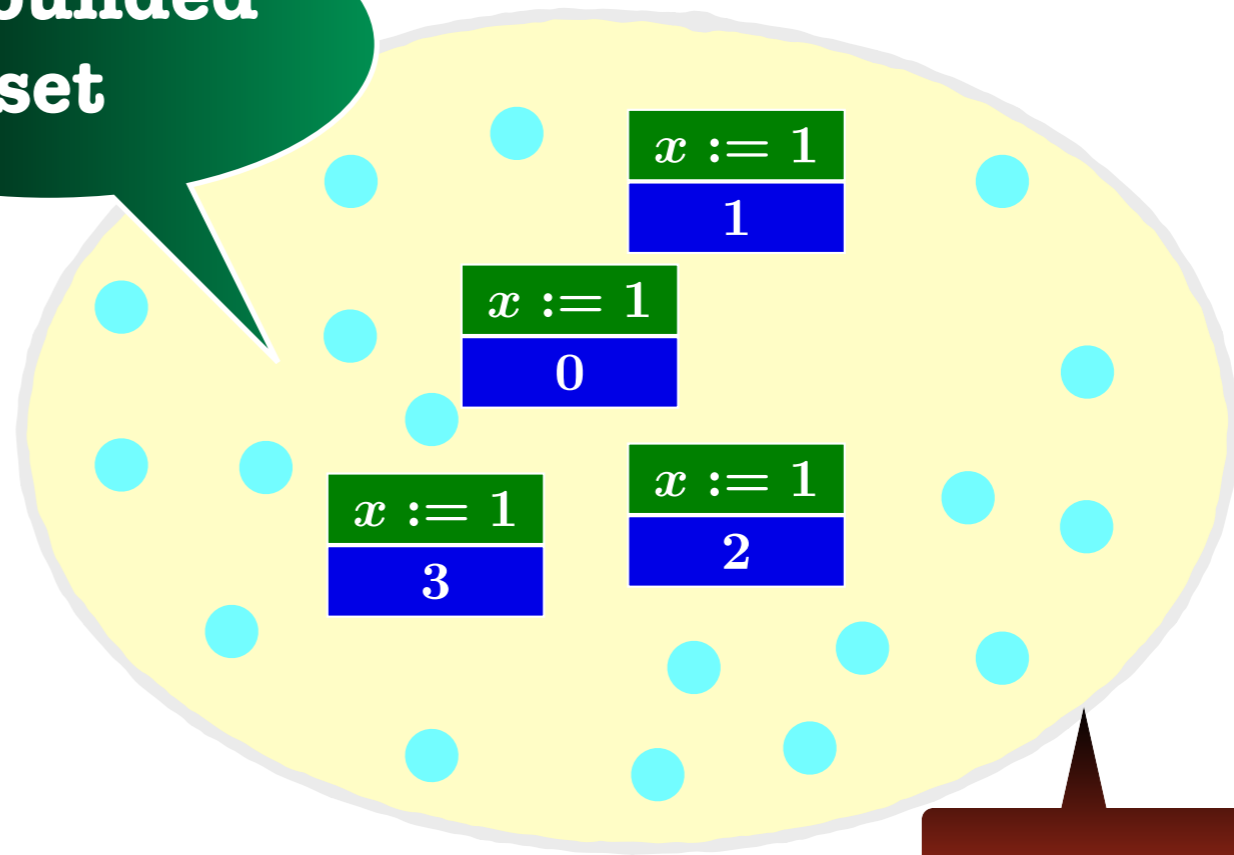
RA

P_1

```
while (1)  
write: x=1
```

```
x := 1  
x := 1  
x := 1  
...  
x := 1  
...
```

unbounded
set



RA

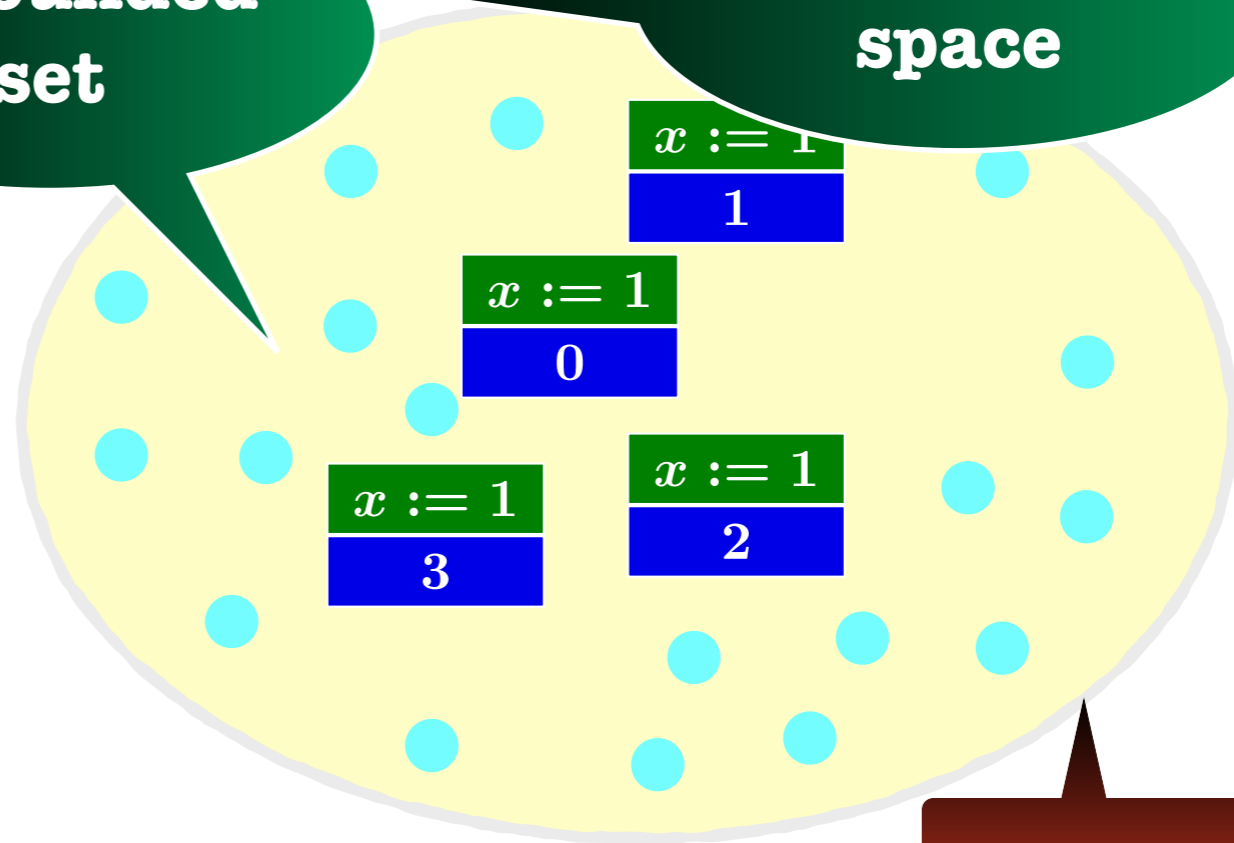
P_1

```
while (1)  
write: x=1
```

```
x := 1  
x := 1  
x := 1  
...  
x := 1  
...
```

unbounded
set

infinite state
space



RA

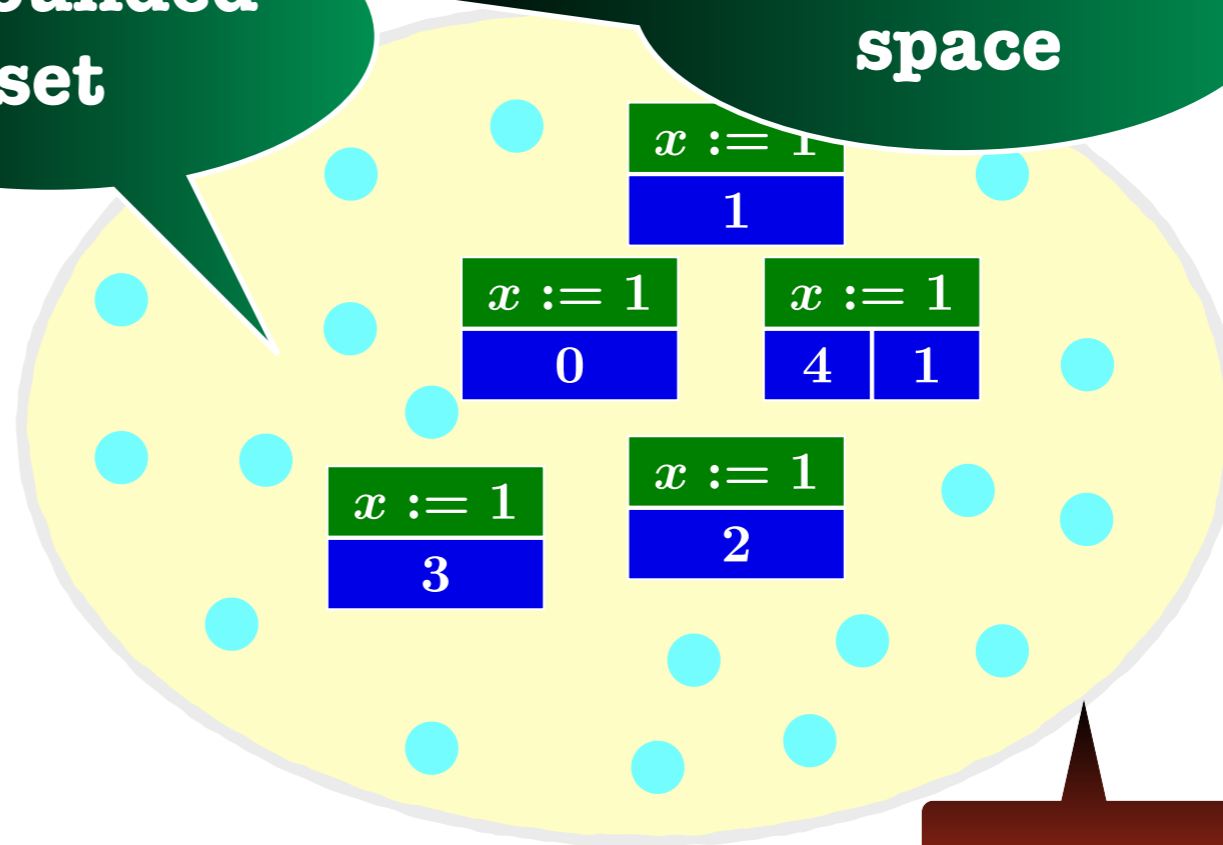
Decidability and Complexity

$x := 1$

Reachability Problem

unbounded
set

infinite state
space



RA

Decidability and Complexity

Reachability Problem

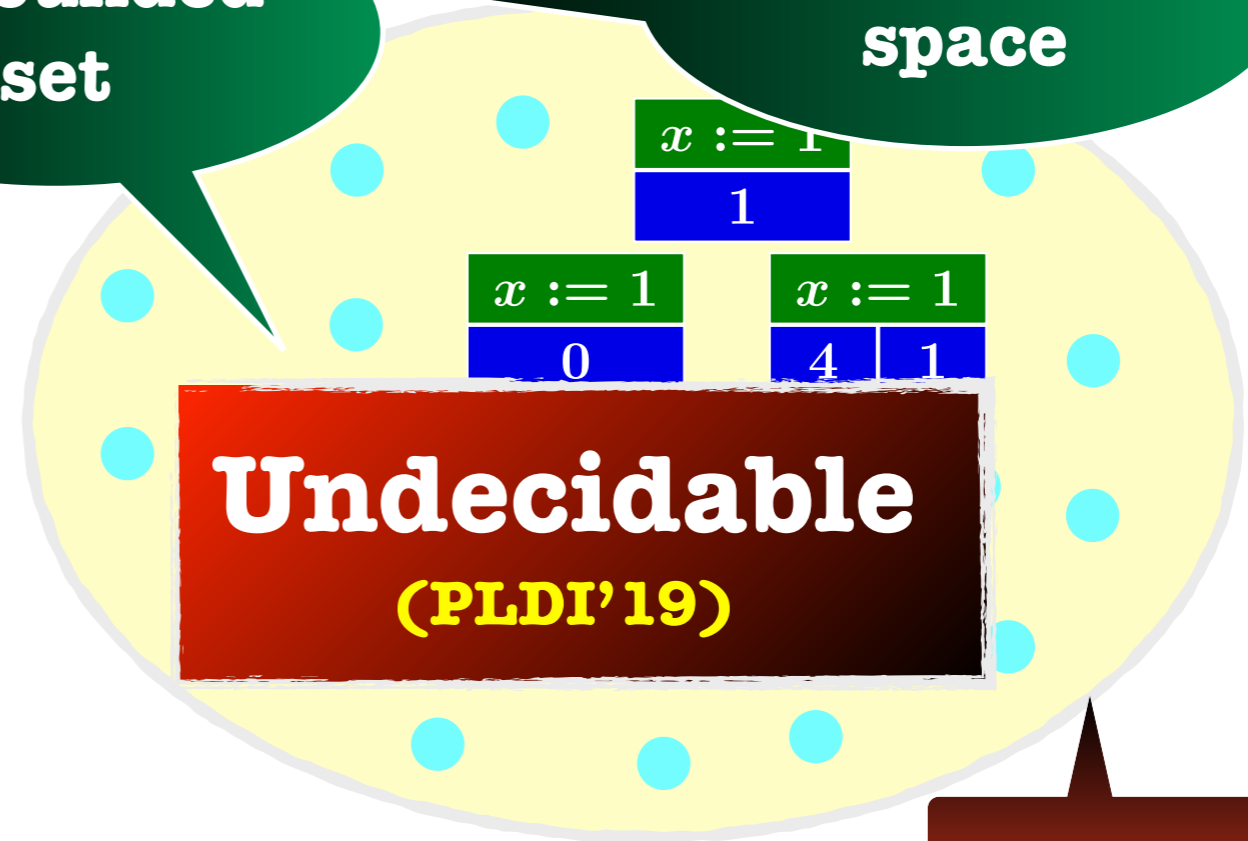
$x := 1$

unbounded
set

infinite state
space

Undecidable
(PLDI'19)

RA



Decidability and Complexity

unbounded
set

infinite state
space

$x := 1$

1

Approximation Techniques

- **Over-Approximation**
 - Abstraction
- **Under-approximation**
 - Bounded Context-Switch (PLDI'19)
 - DPOR (OOPSLA'18)

Reachability
Problem

Decidability and Complexity

unbounded
set

infinite state
space

$x := 1$

1

Approximation Techniques

- Over-Approximation
 - Abstraction
- Under-approximation
 - Bounded Context-Switch (PLDI'19)
 - DPOR (OOPSLA'18)

Reachability
Problem

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- Conclusions

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- Conclusions

Traces

Outline

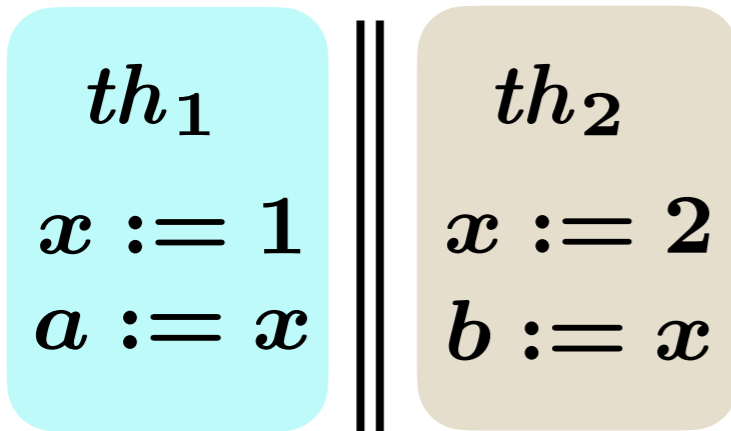
- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- **Conclusions**

Traces

“first let us consider SC”

Concurrent Programs (under SC)

Initially $x = 0$



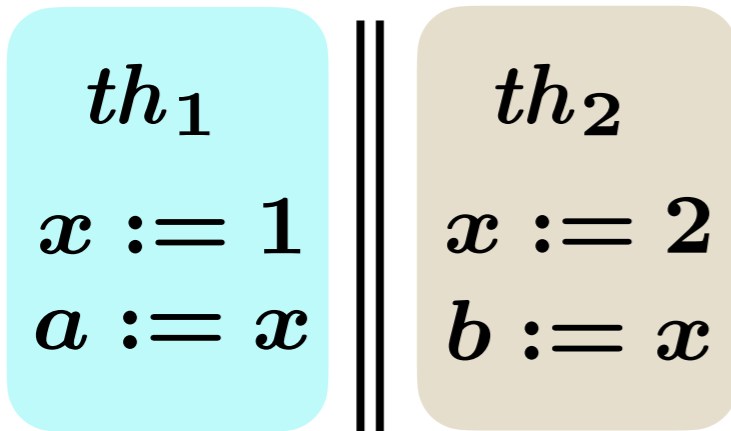
$(a = 1) \wedge (b = 2)?$

assertion

- **Threads/Processes:**
 - **Concurrent**
 - **Loop-Free**
 - **Deterministic**
- **Shared Variables:** x, y
- **Local Variables:** a, b, c, d
- **Interleaving Semantics**

Concurrent Programs (under SC)

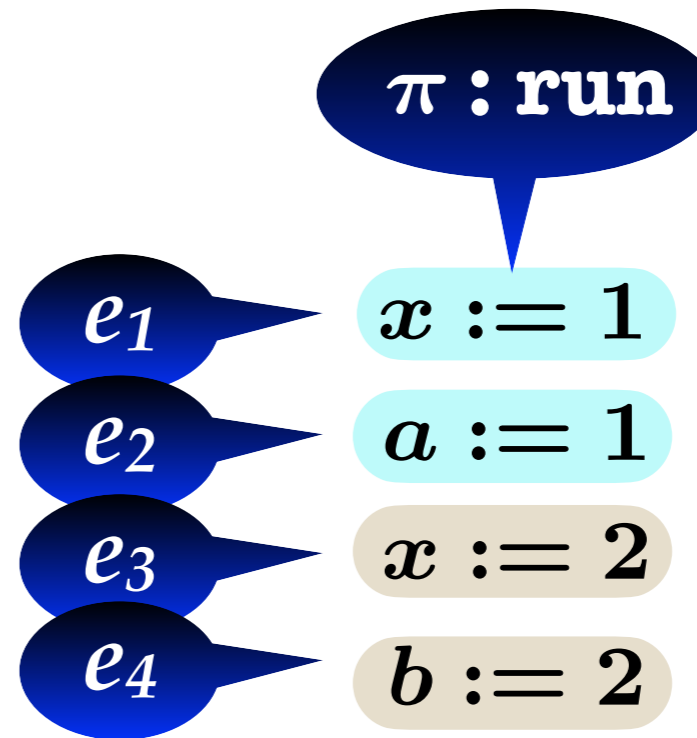
Initially $x = 0$



$(a = 1) \wedge (b = 2)?$

assertion

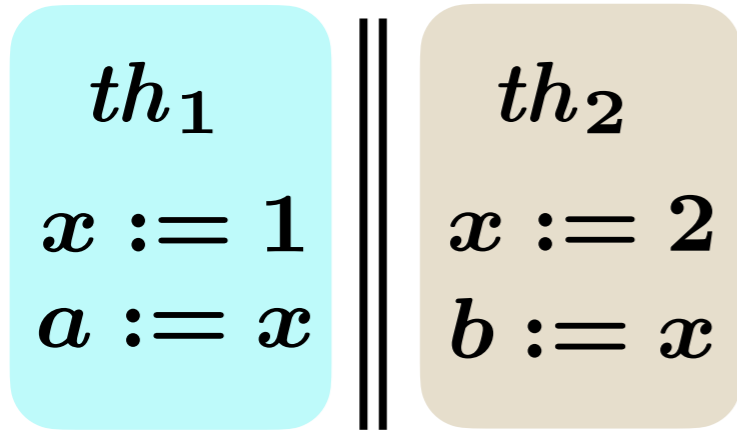
events



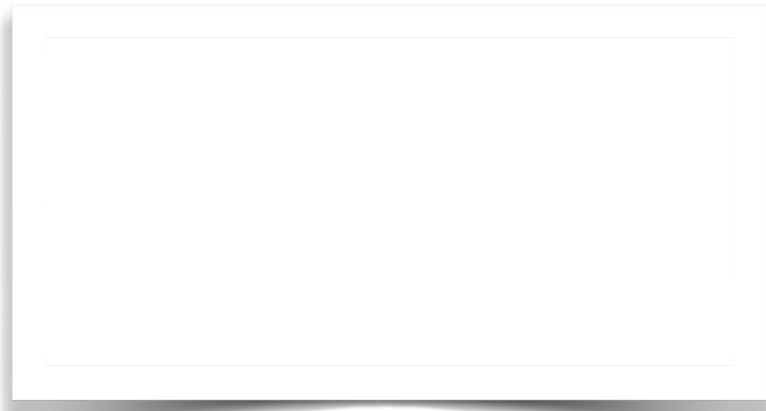
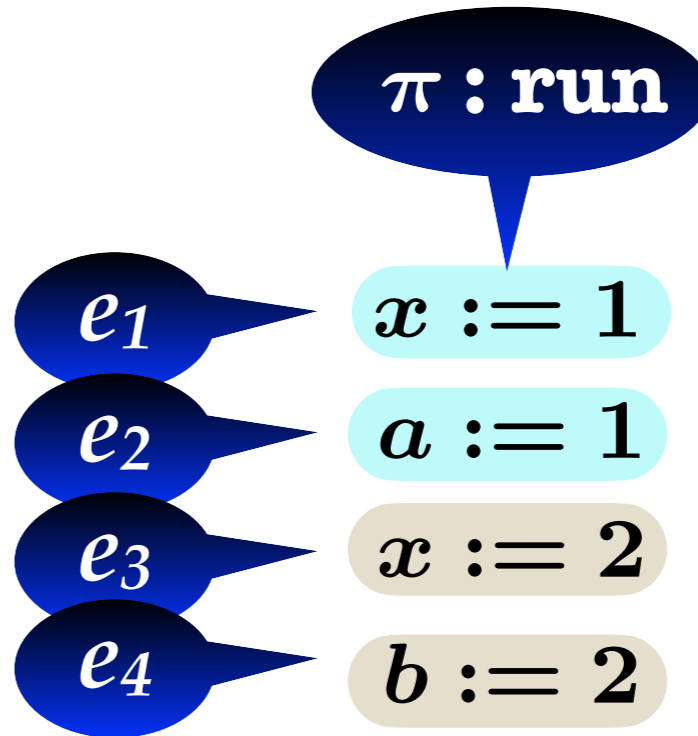
- **Threads/Processes:**
 - Concurrent
 - Loop-Free
 - Deterministic
- **Shared Variables:** x, y
- **Local Variables:** a, b, c, d
- **Interleaving Semantics**

Concurrent Programs (under SC)

Initially $x = 0$

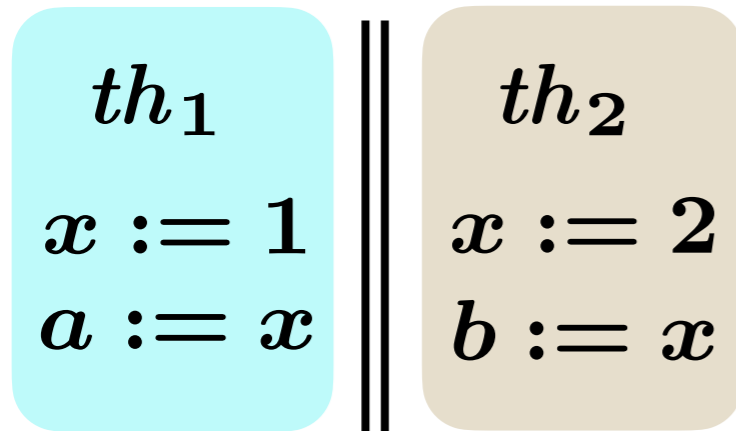


events

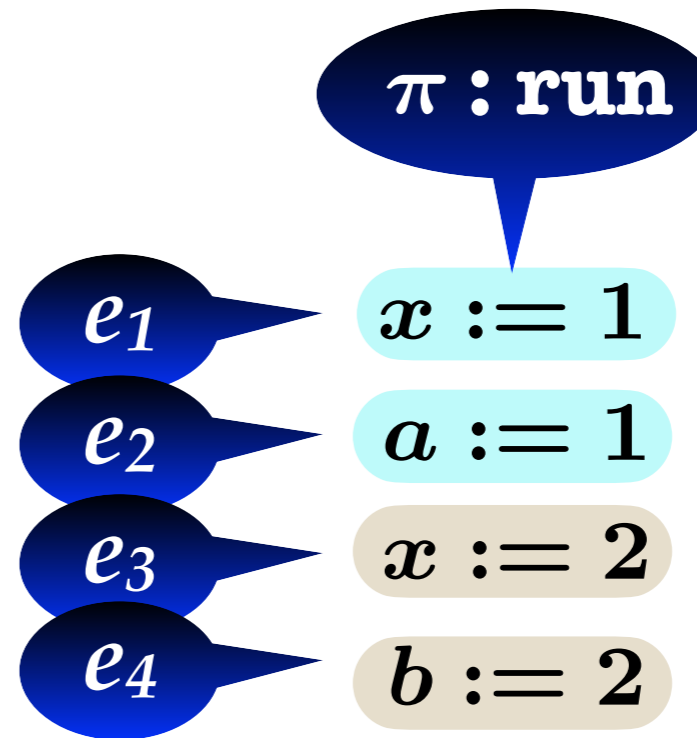


Concurrent Programs (under SC)

Initially $x = 0$



events



- **Traces**

- **efficient:**

- **more compact than runs**

- **sufficient for checking assertions**

- **abstract:**

- **allow different memory models**

Concurrent Programs (under SC)

Initially $x = 0$

th_1
 $x := 1$
 $a := x$

th_2
 $x := 2$
 $b := x$

axiomatic semantics

events

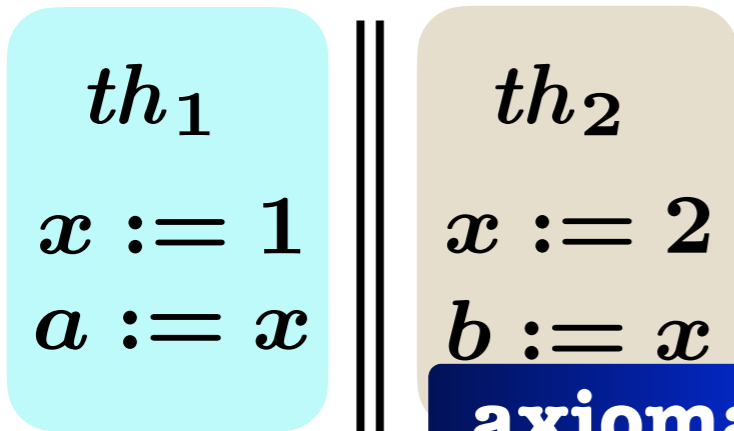
e_1
 e_2
 e_3
 e_4

$\pi : \text{run}$
 $x := 1$
 $a := 1$
 $x := 2$
 $b := 2$

- **Traces**
 - **efficient:**
 - more compact than runs
 - sufficient for checking assertions
 - **abstract:**
 - allow different memory models

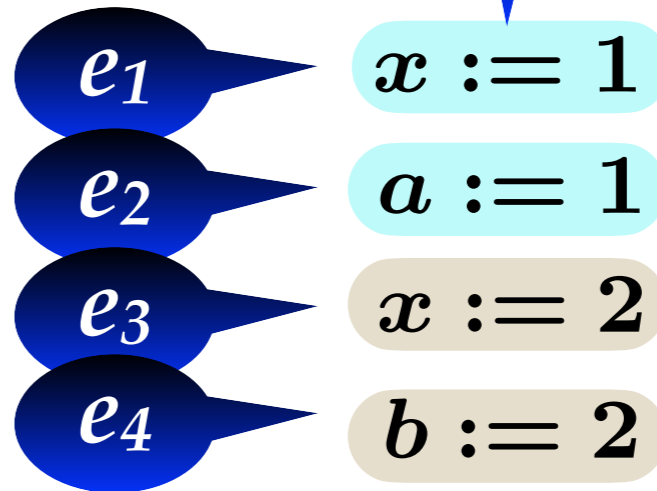
Concurrent Programs (under SC)

Initially $x = 0$



axiomatic semantics

events

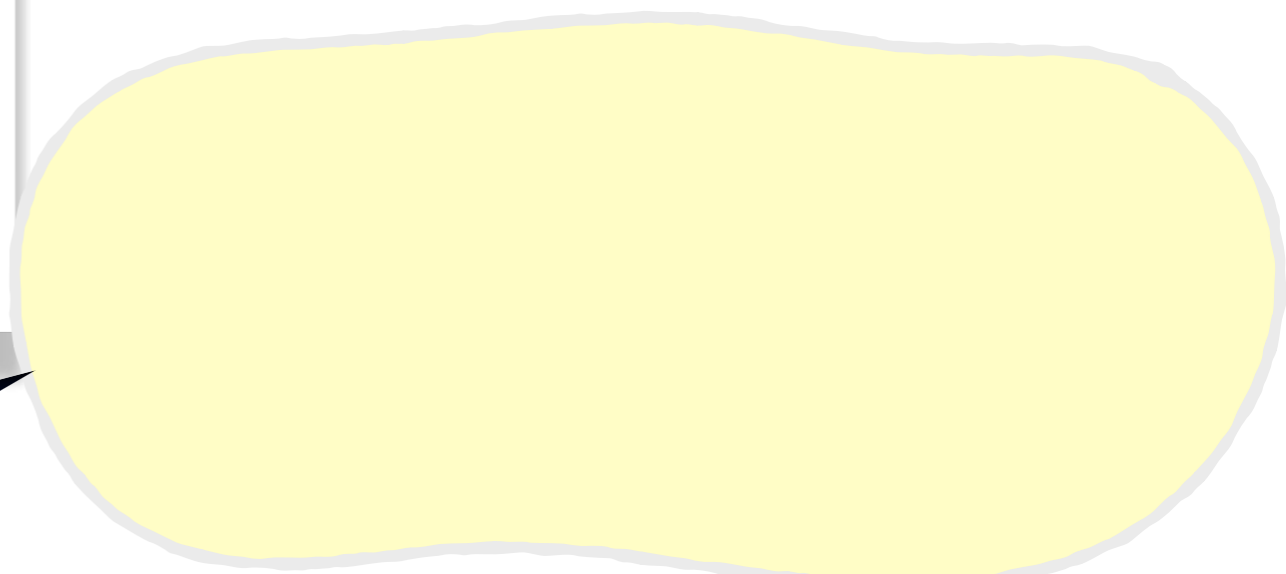


π : run

- **Traces**

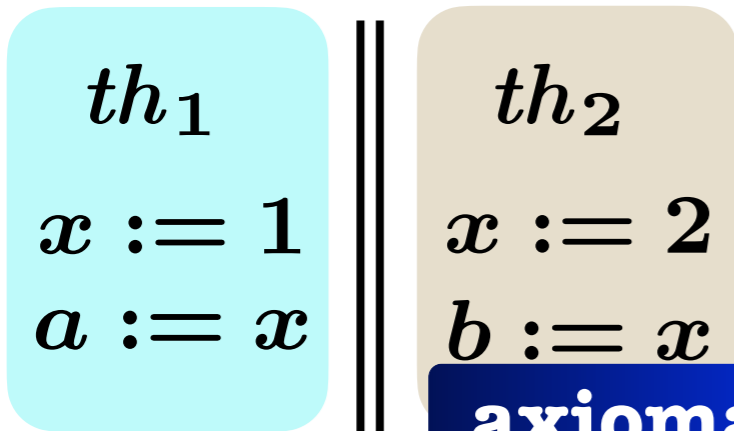
- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

τ : trace



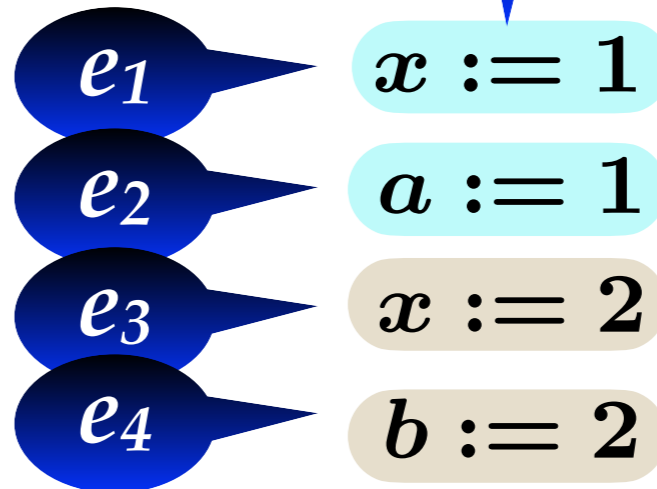
Concurrent Programs (under SC)

Initially $x = 0$



axiomatic semantics

events



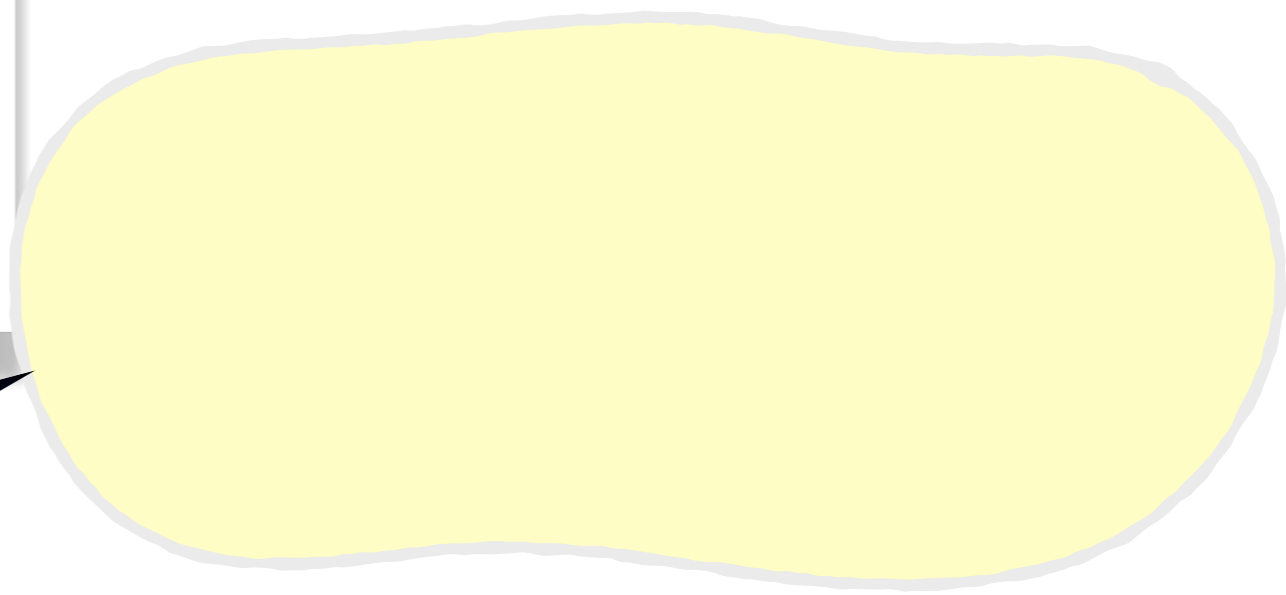
π : run

- **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

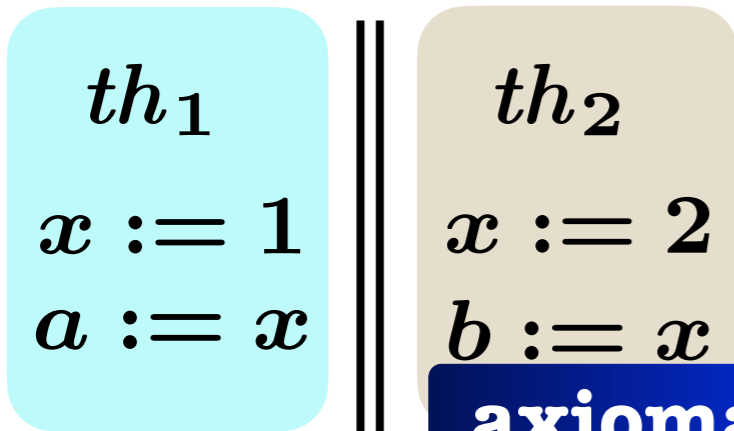
τ : trace

graph



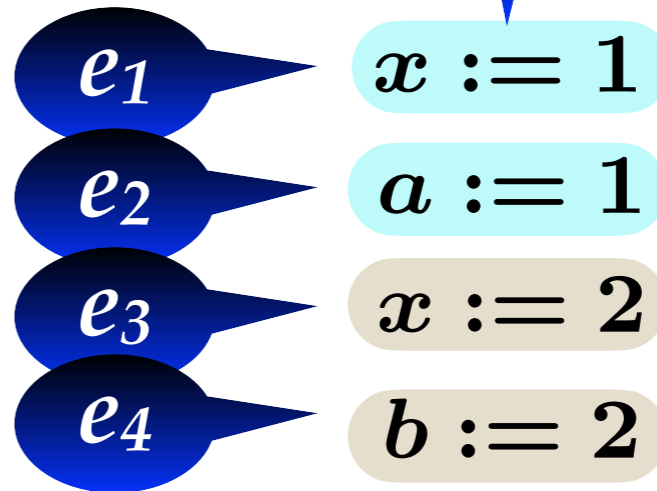
Concurrent Programs (under SC)

Initially $x = 0$



axiomatic semantics

events



π : run

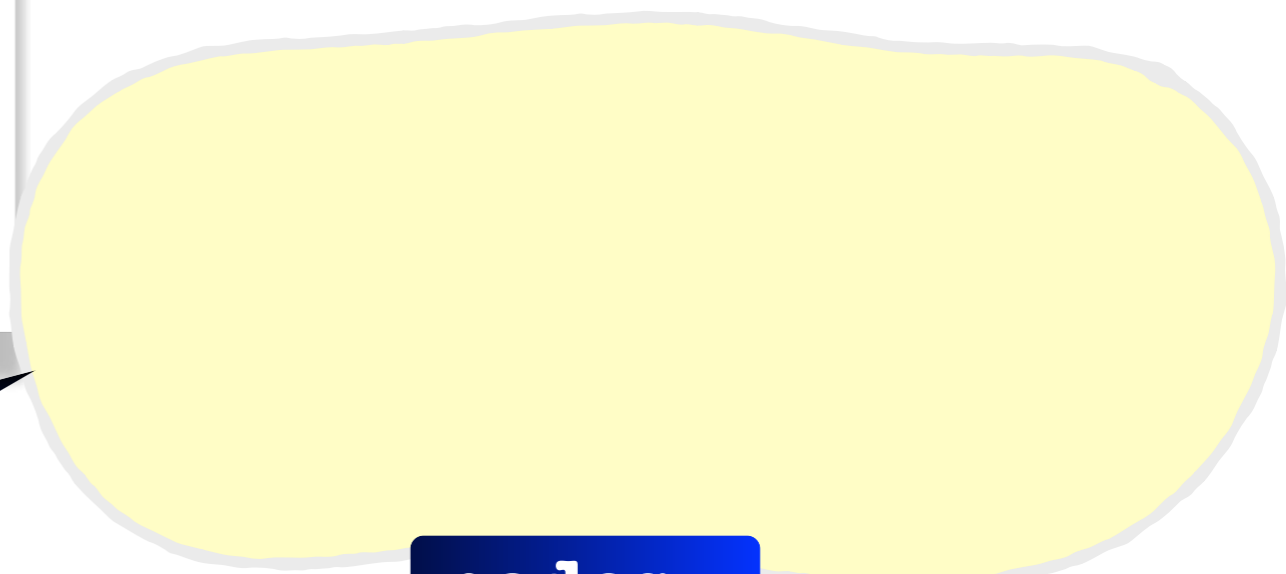
- **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

τ : trace

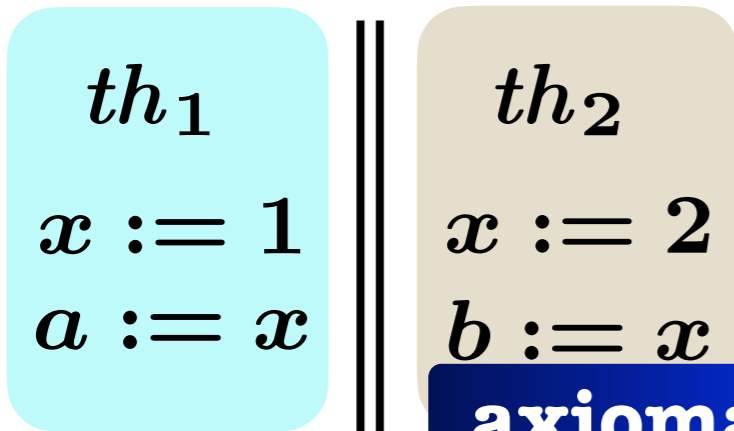
graph

nodes = events



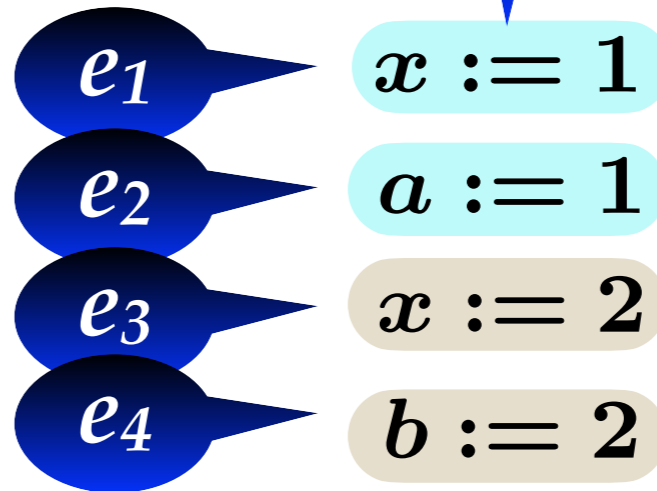
Concurrent Programs (under SC)

Initially $x = 0$



axiomatic semantics

events



- **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

$x := 1$

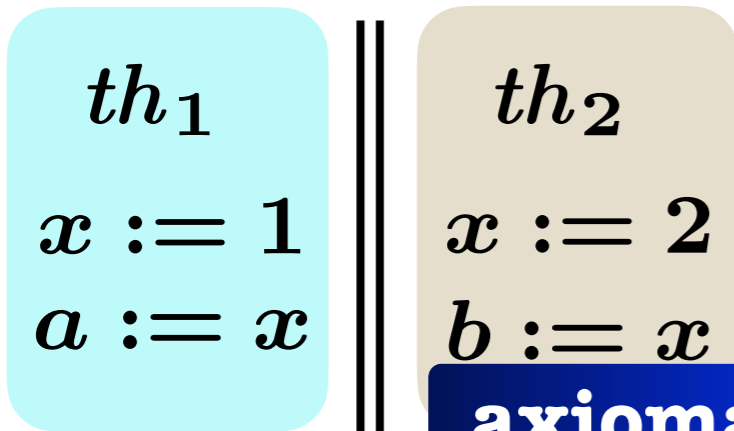
τ : trace

graph

nodes = events

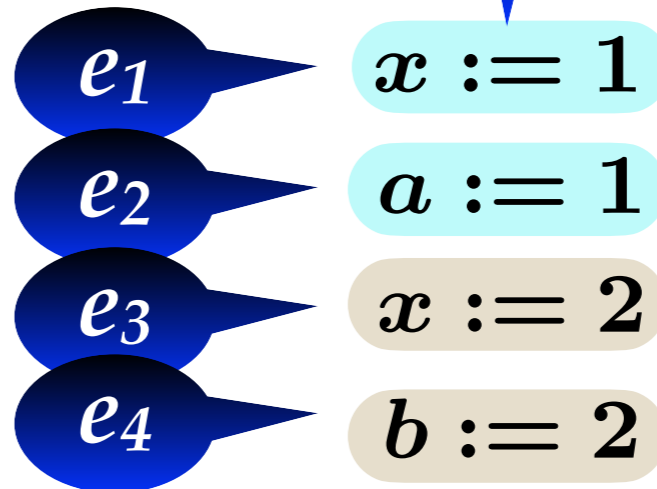
Concurrent Programs (under SC)

Initially $x = 0$



axiomatic semantics

events



- **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

$x := 1$

$a := 1$

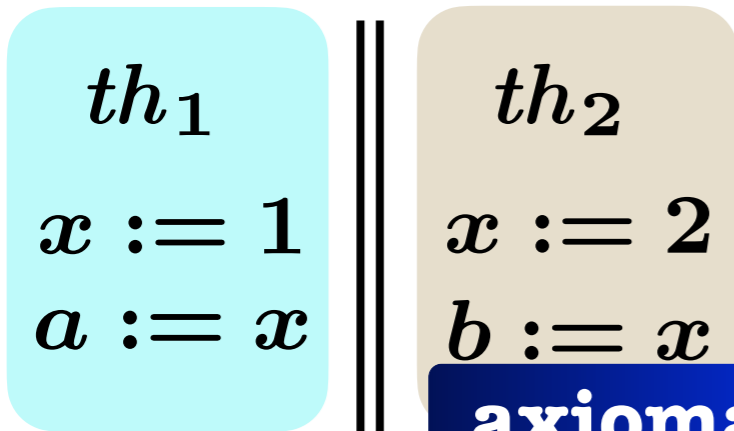
τ : trace

graph

nodes = events

Concurrent Programs (under SC)

Initially $x = 0$

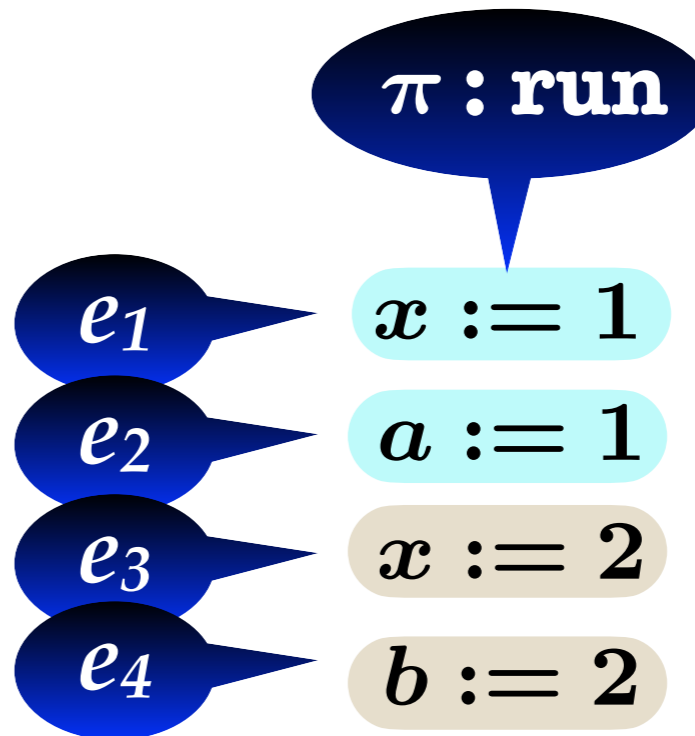


axiomatic semantics

- **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

events



$x := 1$

$x := 2$

$a := 1$

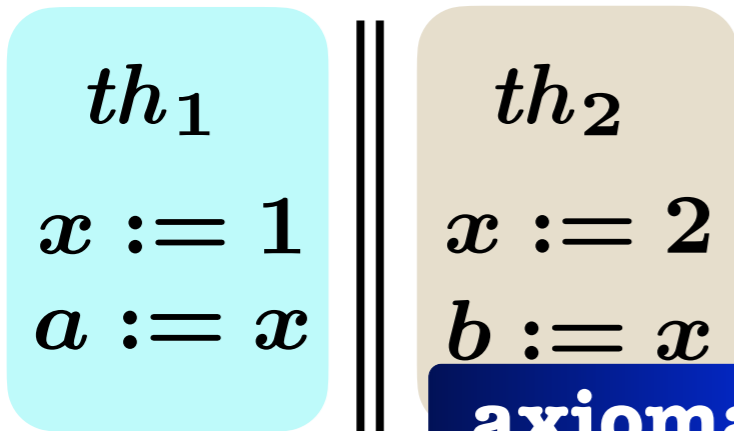
τ : trace

graph

nodes = events

Concurrent Programs (under SC)

Initially $x = 0$

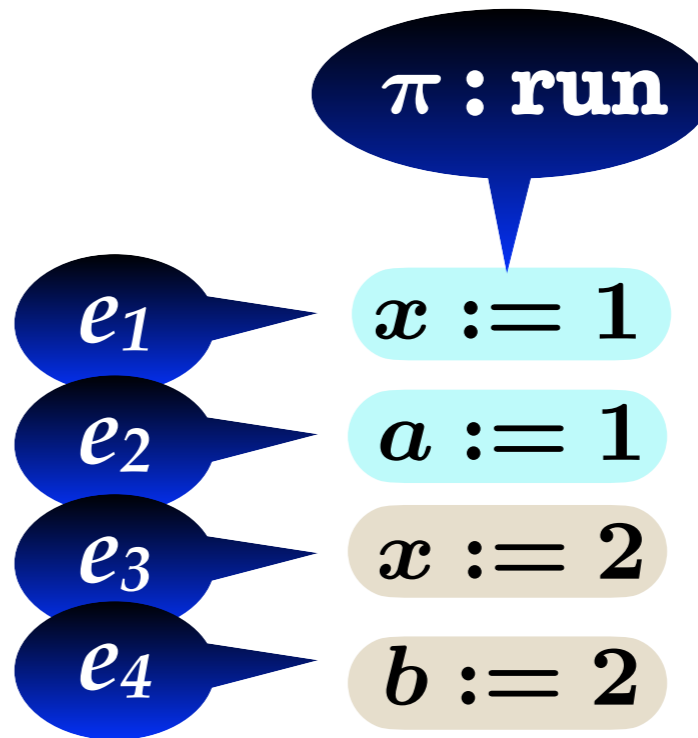


axiomatic semantics

- **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

events



$x := 1$

$x := 2$

$a := 1$

$b := 2$

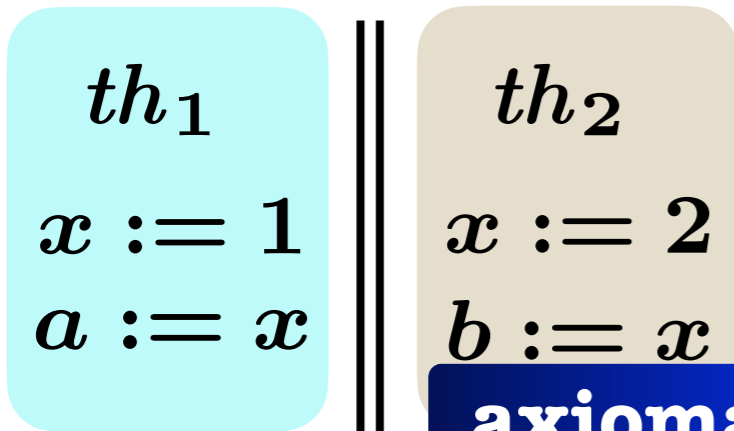
τ : trace

graph

nodes = events

Concurrent Programs (under SC)

Initially $x = 0$

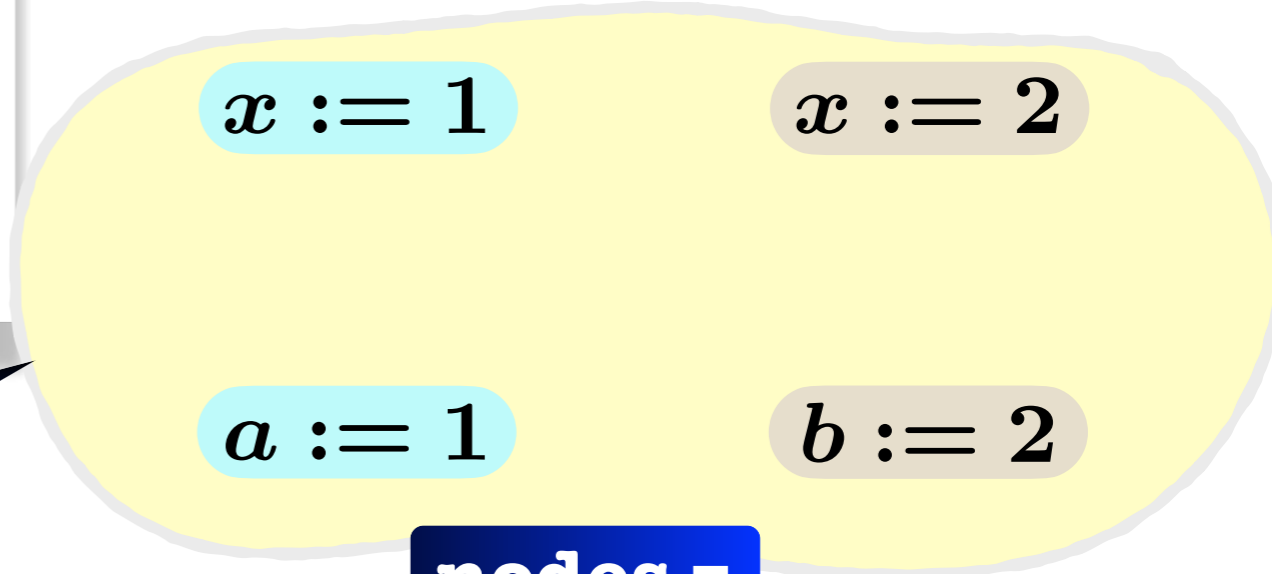
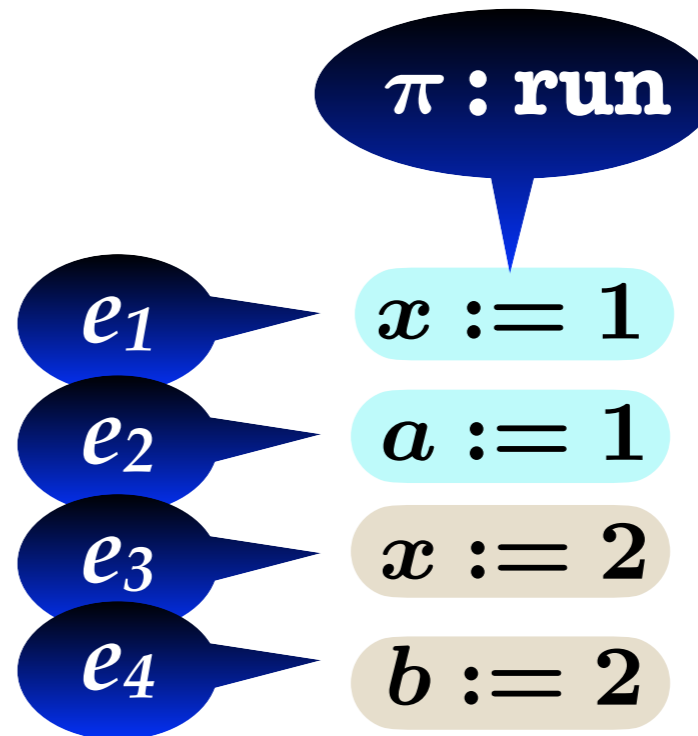


axiomatic semantics

- **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

events



τ : trace

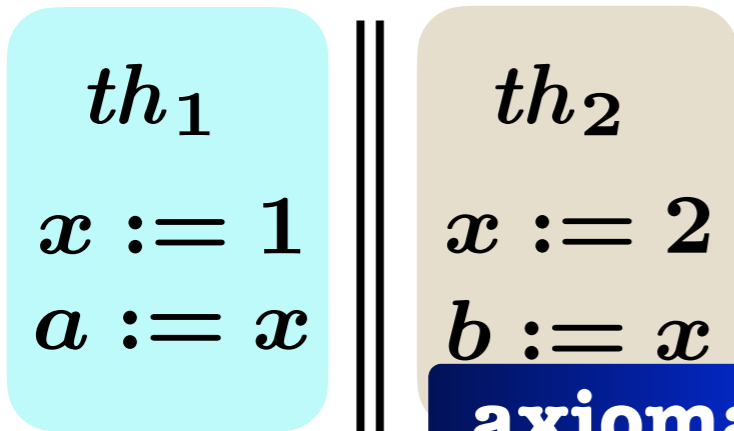
graph

nodes = events

edges = relations

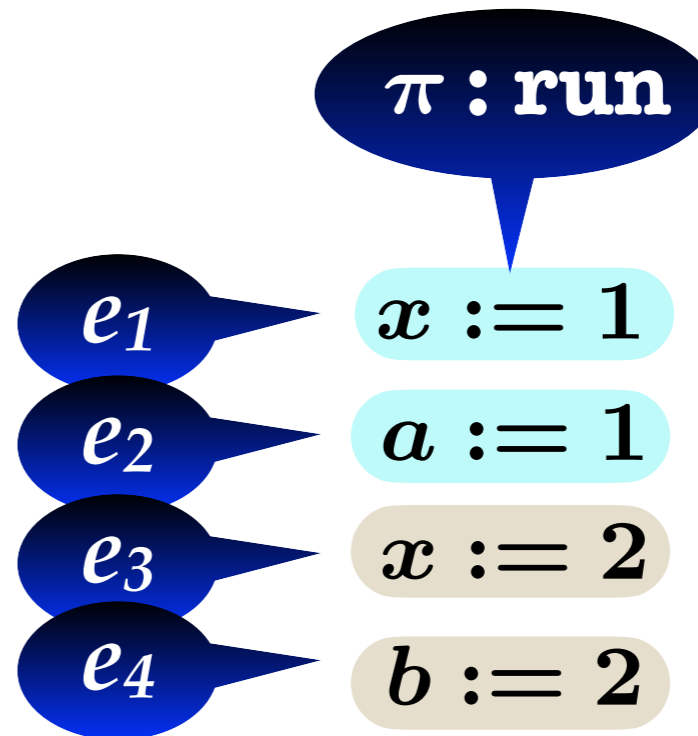
Concurrent Programs (under SC)

Initially $x = 0$



axiomatic semantics

events



Traces

- efficient:
 - more compact than runs
 - sufficient for checking assertions
- abstract:
 - allow different memory models

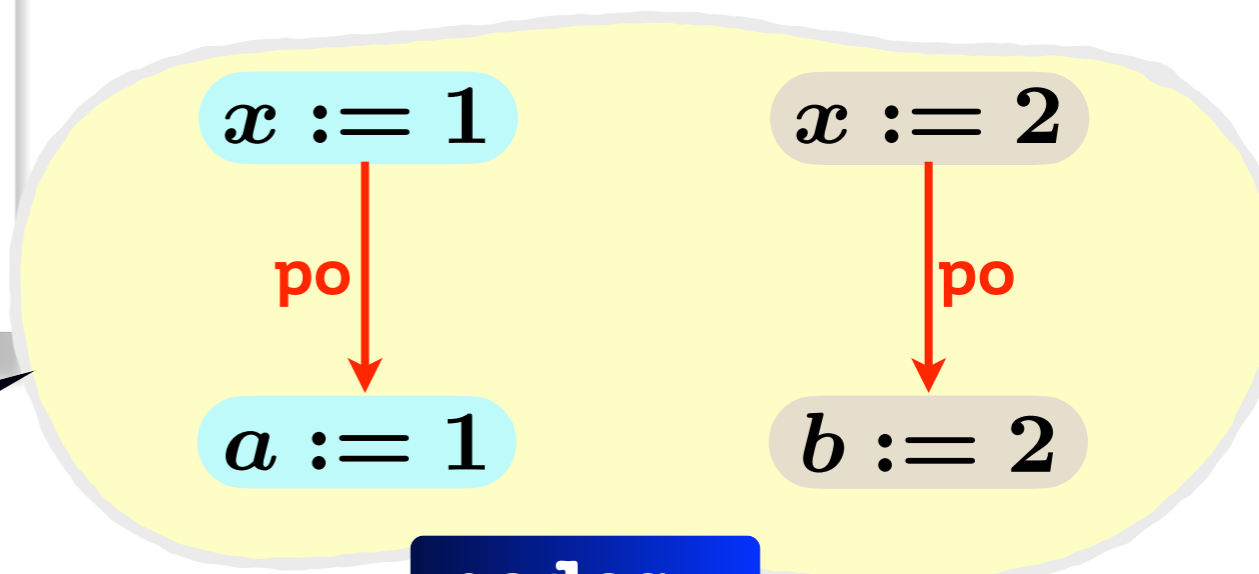
po : program order

τ : trace

graph

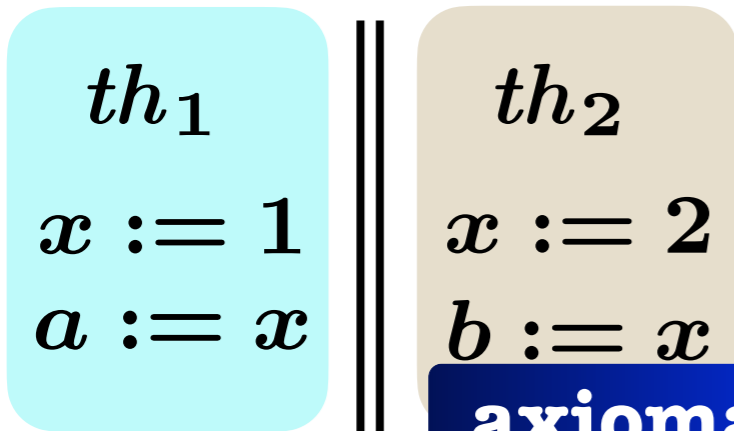
nodes = events

edges = relations



Concurrent Programs (under SC)

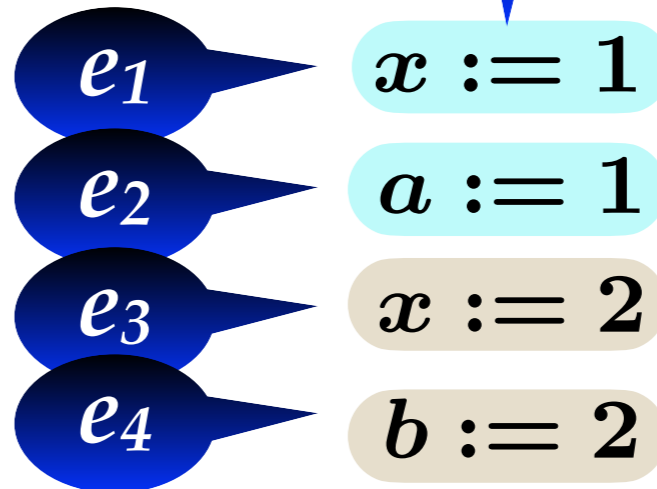
Initially $x = 0$



axiomatic semantics

events

$\pi : \text{run}$



• **Traces**

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

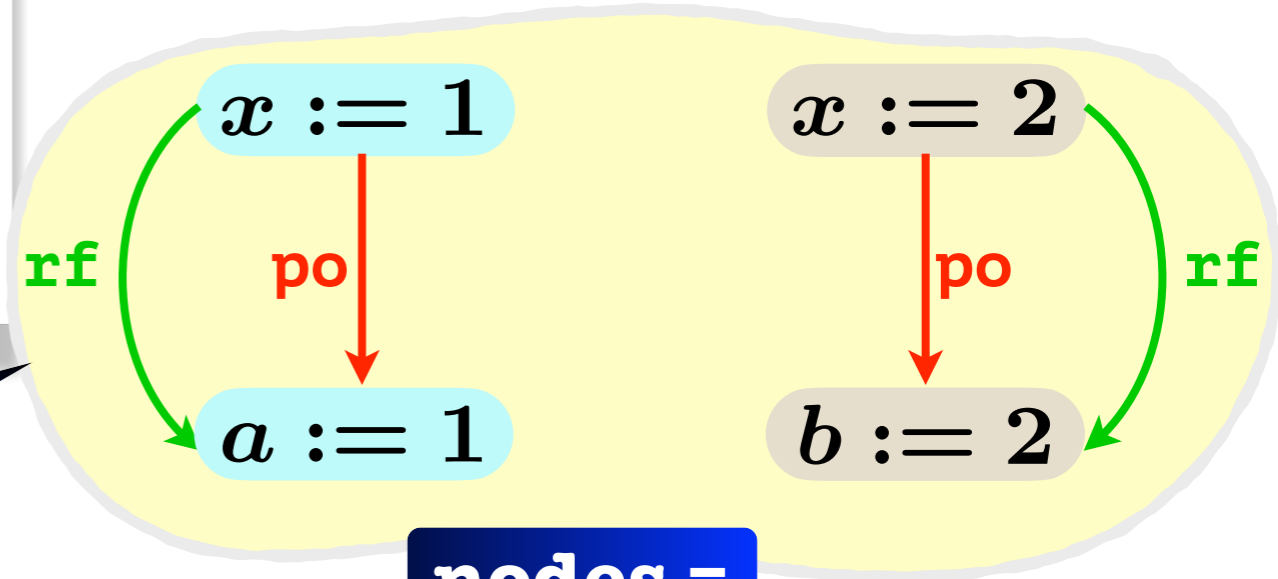
- **po**: program order
- **rf**: read-from

$\tau : \text{trace}$

graph

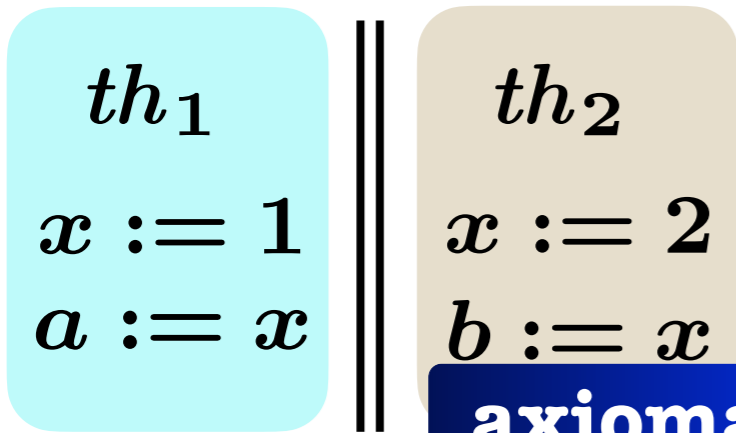
nodes = events

edges = relations



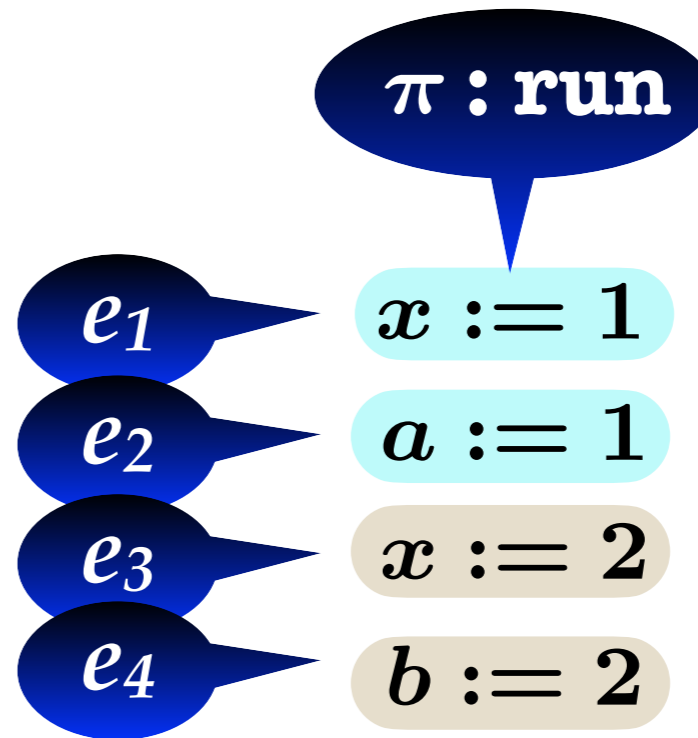
Concurrent Programs (under SC)

Initially $x = 0$



axiomatic semantics

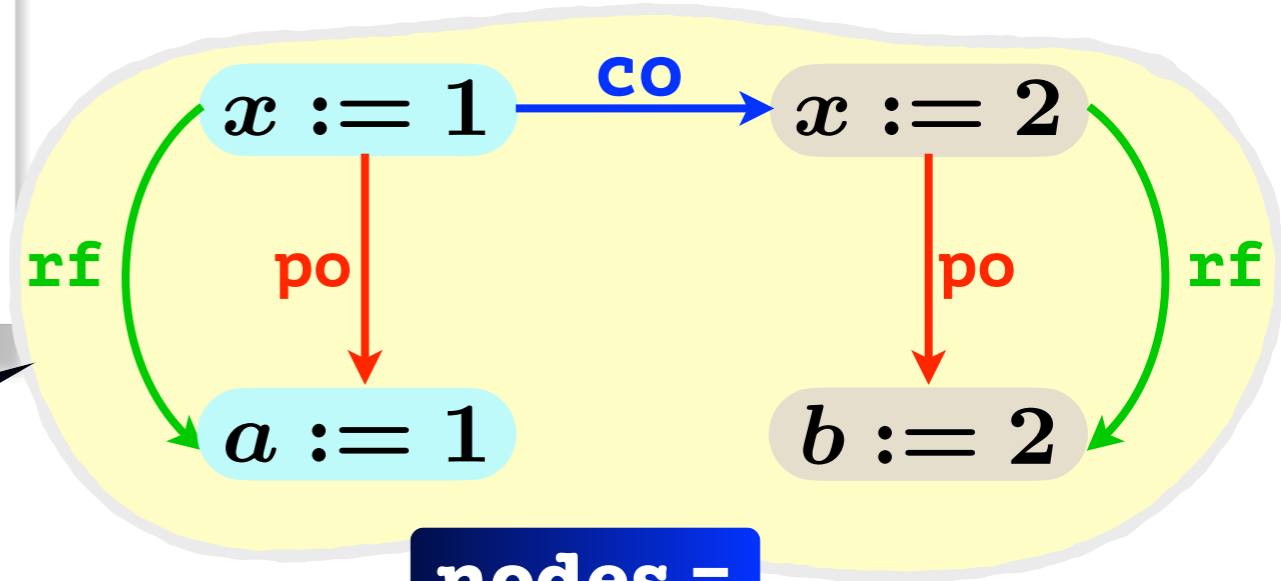
events



Traces

- efficient:
 - more compact than runs
 - sufficient for checking assertions
- abstract:
 - allow different memory models

- po**: program order
- rf**: read-from
- co**: coherence order



$\tau : \text{trace}$

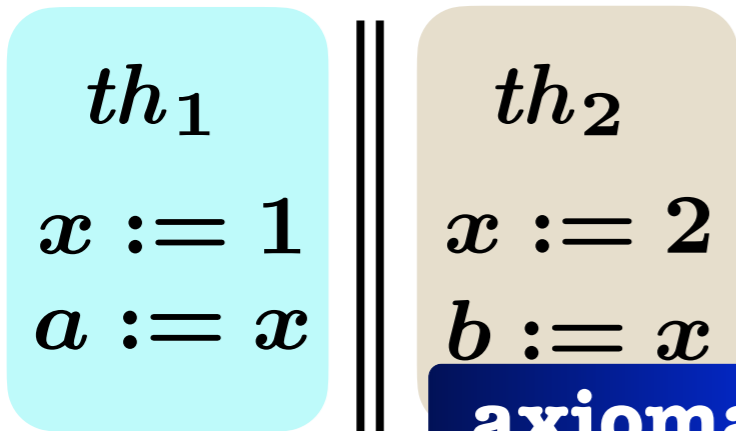
graph

nodes = events

edges = relations

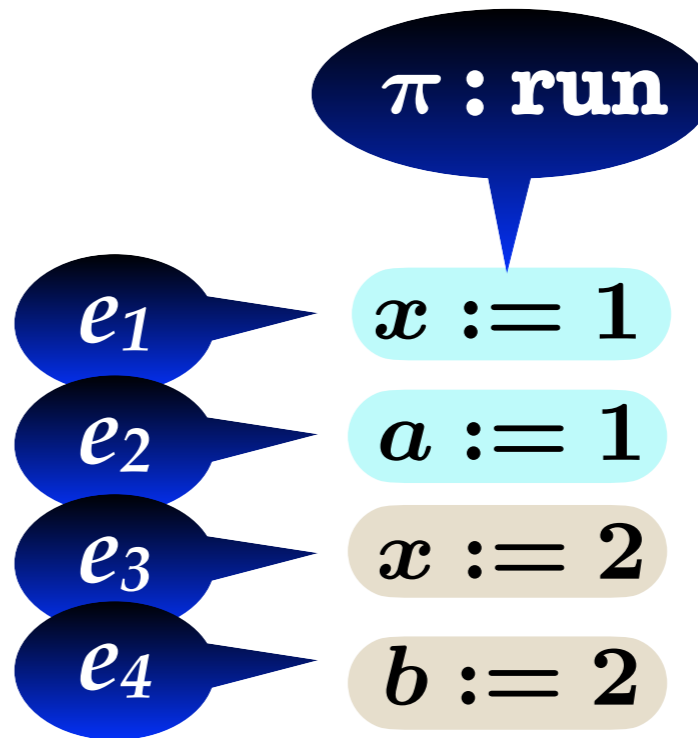
Concurrent Programs (under SC)

Initially $x = 0$



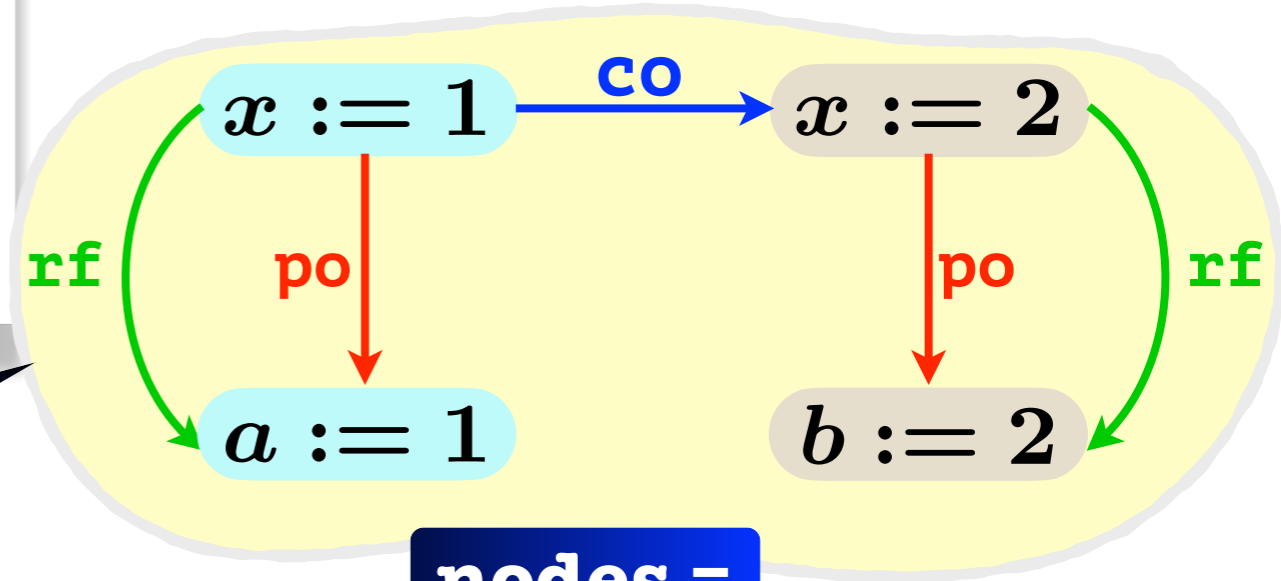
axiomatic semantics

events



Traces

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models



- **po**: program order
- **rf**: read-from
- **co**: coherence order

$\tau : \text{trace}$

graph

nodes = events

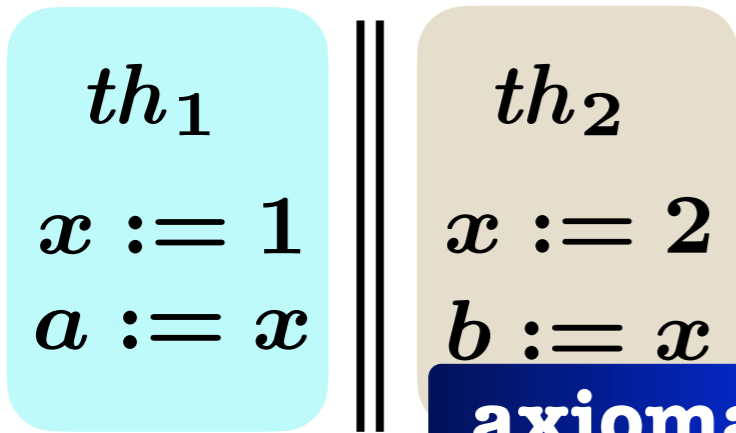
edges = relations

$$\text{co} = \bigcup_{x \in X} \text{co}^x$$

total

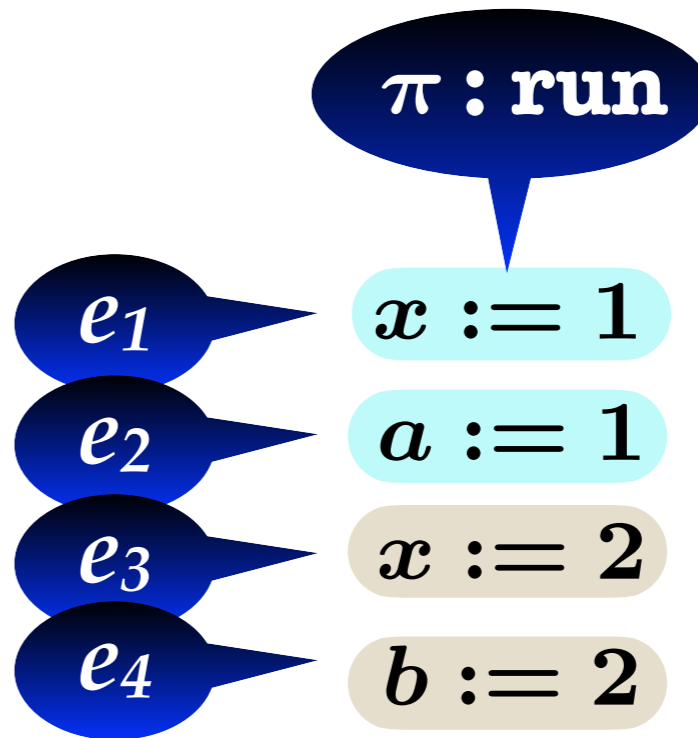
Concurrent Programs (under SC)

Initially $x = 0$



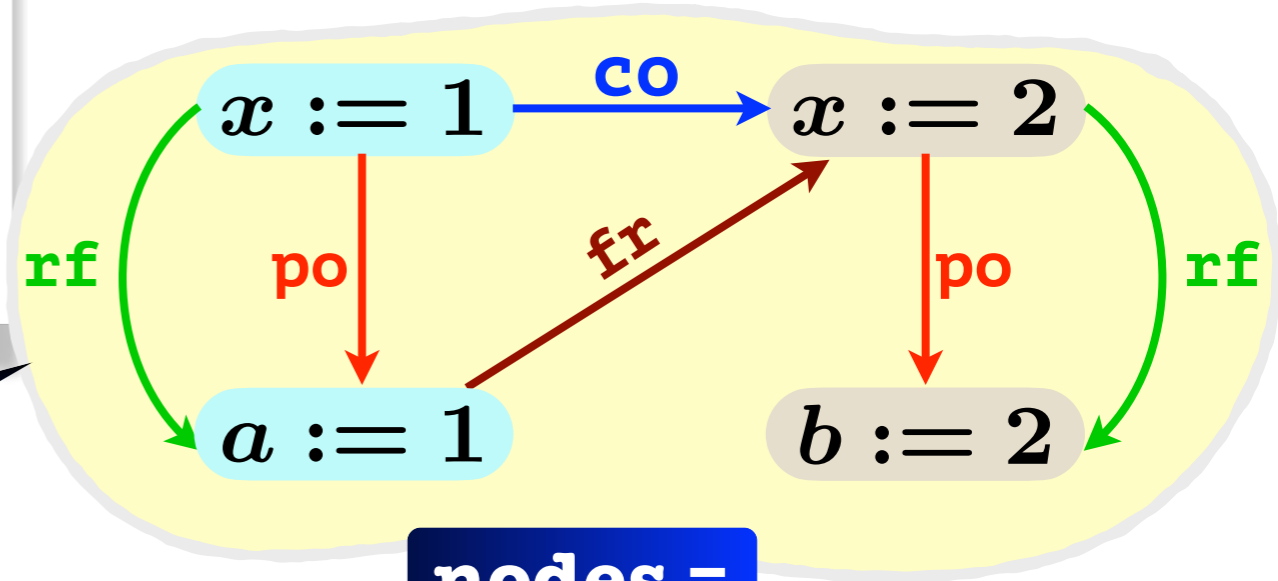
axiomatic semantics

events



Traces

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models



- **po**: program order
- **rf**: read-from
- **co**: coherence order
- **fr**: from-read

$\tau : \text{trace}$

graph

nodes = events

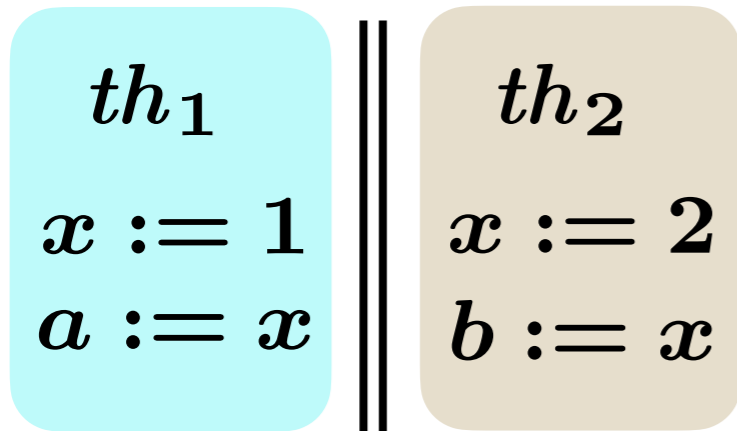
edges = relations

$$\text{co} = \bigcup_{x \in X} \text{co}^x$$

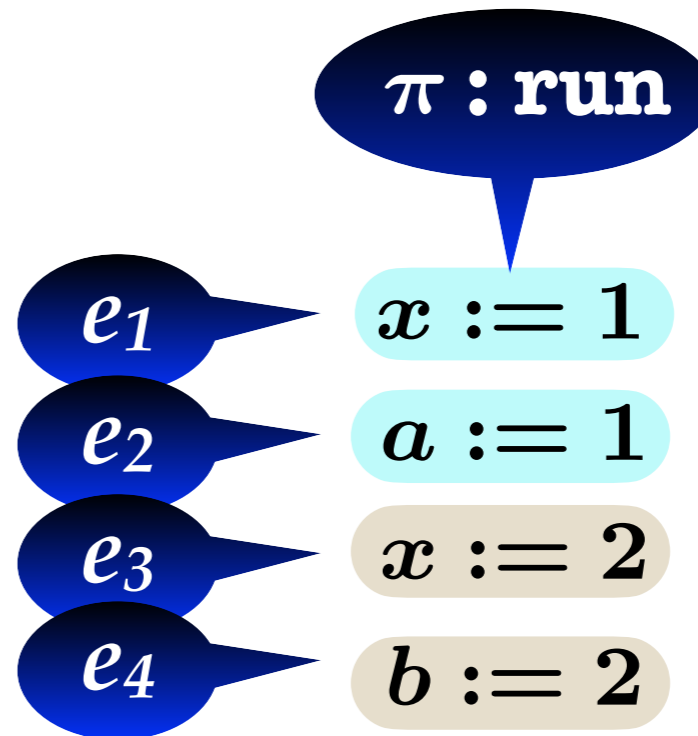
total

Concurrent Programs (under SC)

Initially $x = 0$



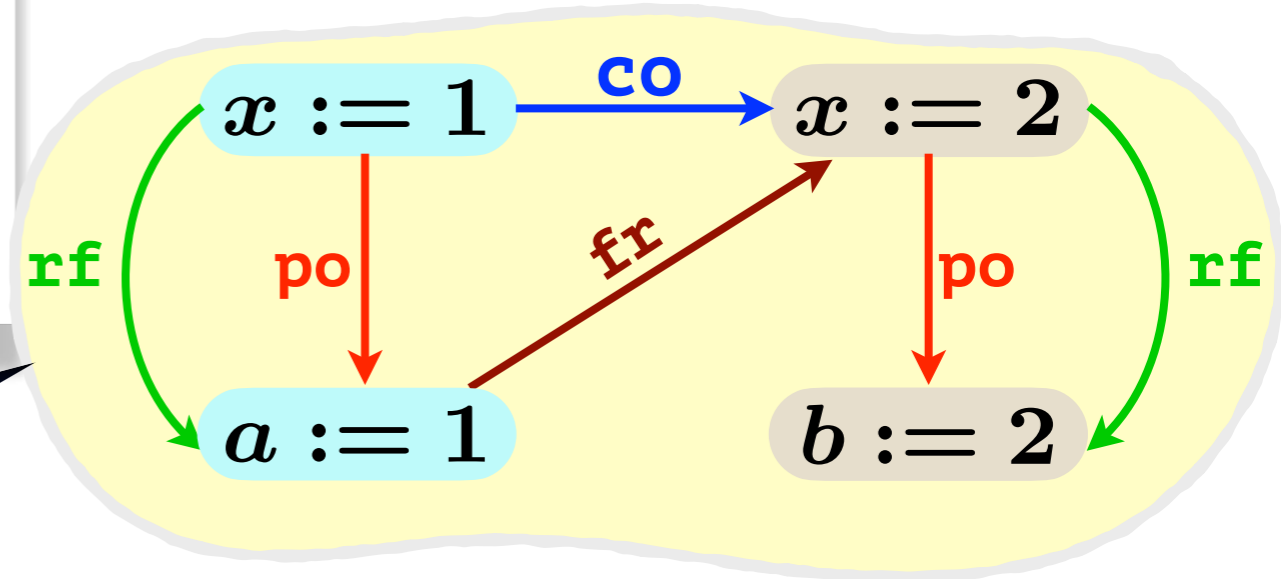
events



Traces

- efficient:
 - more compact than runs
 - sufficient for checking assertions
- abstract:
 - allow different memory models

- po**: program order
- rf**: read-from
- co**: coherence order
- fr**: from-read



$\tau : \text{trace}$

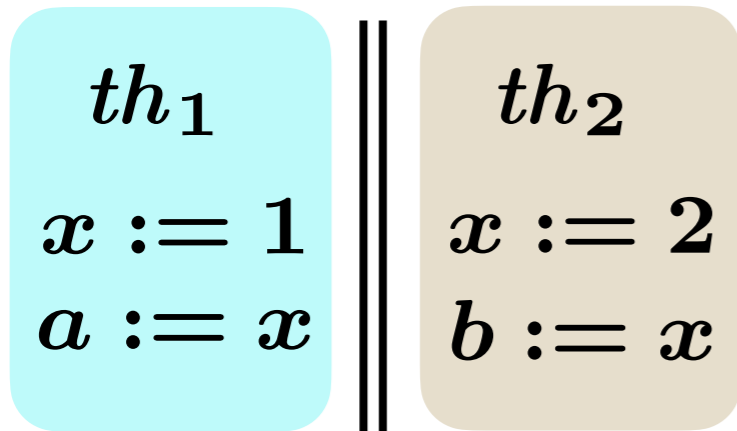
SC: $acyclic(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

$$\text{co} = \bigcup_{x \in X} \text{co}^x$$

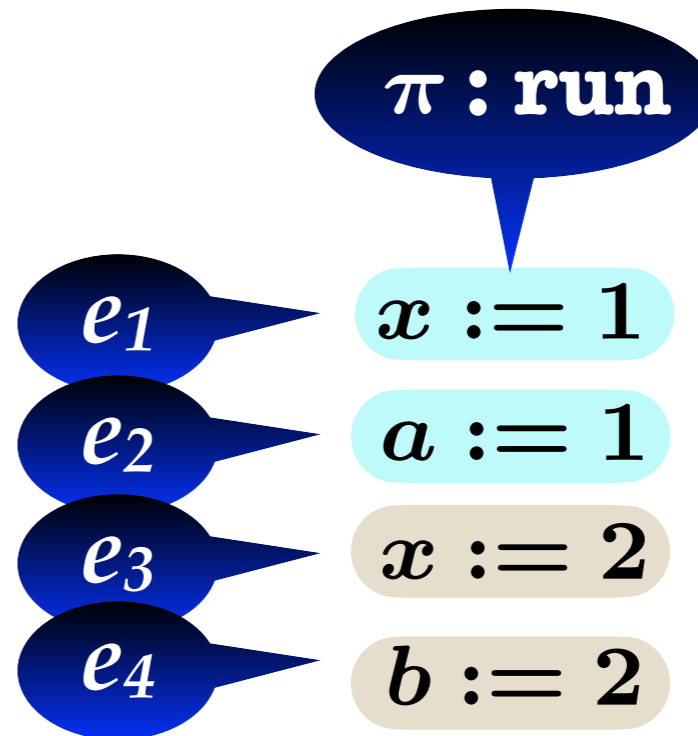
total

Concurrent Programs (under SC)

Initially $x = 0$



events

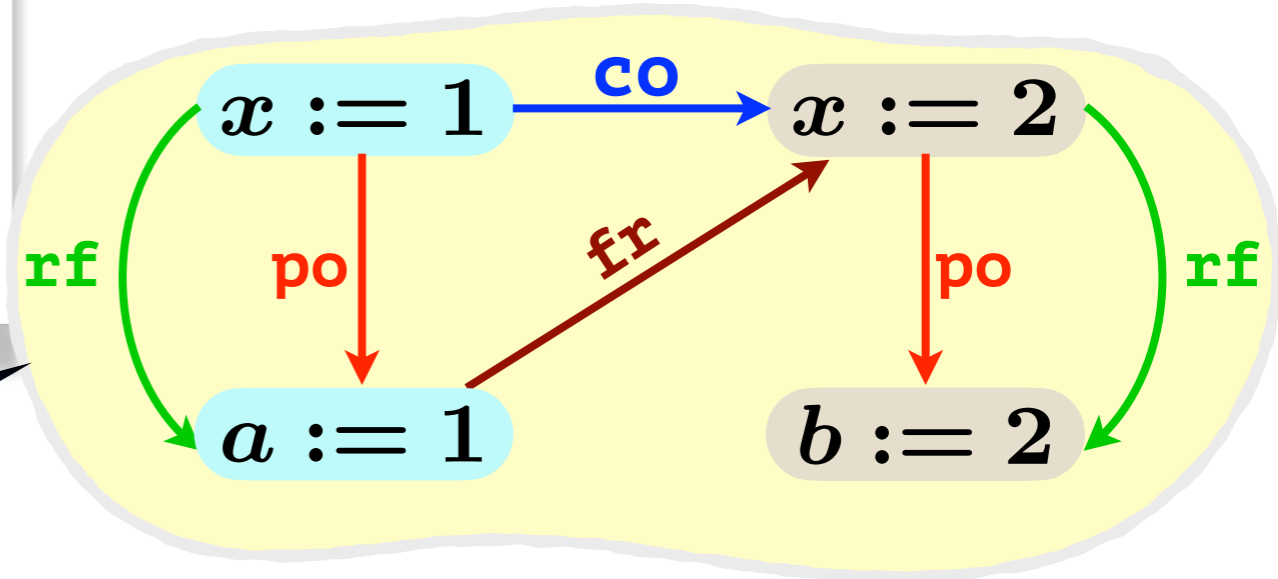


Traces

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allow different memory models

- **po**: program order
- **rf**: read-from
- **co**: coherence order
- **fr**: from-read

$\tau : \text{trace}$



SC: $acyclic(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

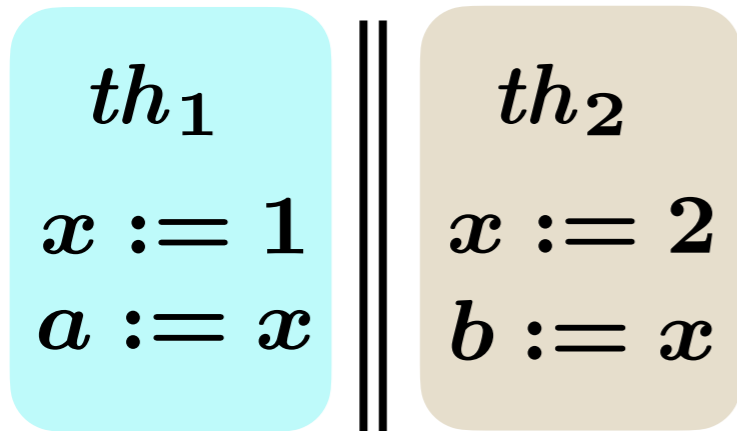
$$\text{co} = \bigcup_{x \in X} \text{co}^x$$

total

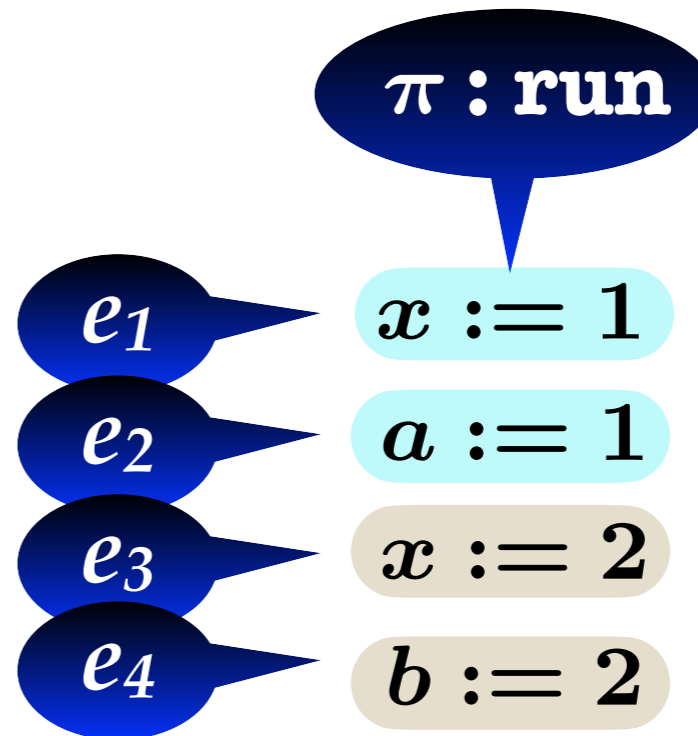
$$\tau \models \text{SC}$$

Concurrent Programs (under SC)

Initially $x = 0$



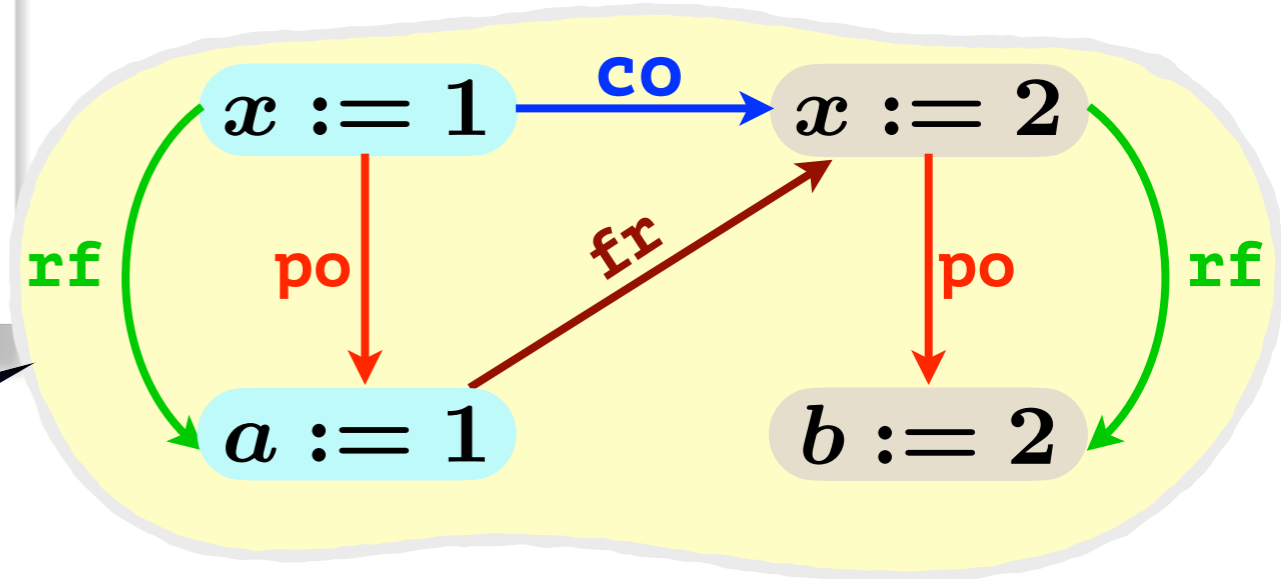
events



Traces

- efficient:
 - more compact than runs
 - sufficient for checking assertions
- abstract:
 - allow different memory models

- po**: program order
- rf**: read-from
- co**: coherence order
- fr**: from-read



$\tau : \text{trace}$

SC: $acyclic(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

$$\text{co} = \bigcup_{x \in X} \text{co}^x$$

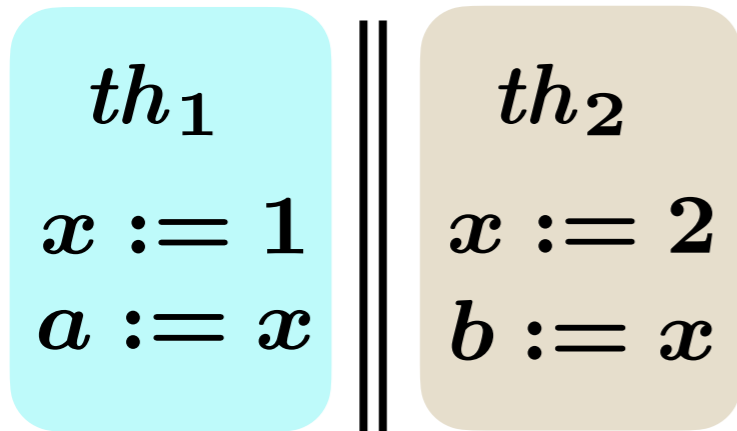
total

$$\tau \models \text{SC}$$

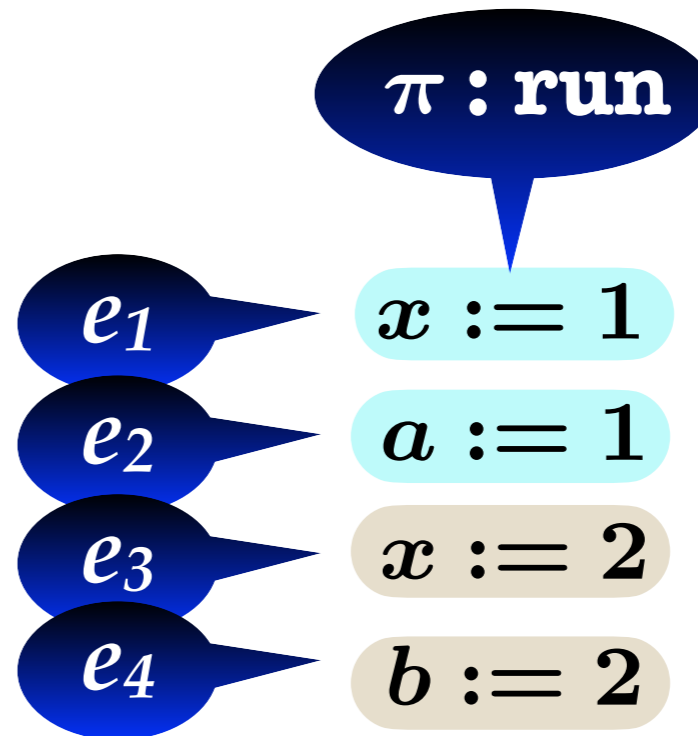
$$\pi \models \tau$$

Concurrent Programs (under SC)

Initially $x = 0$



events

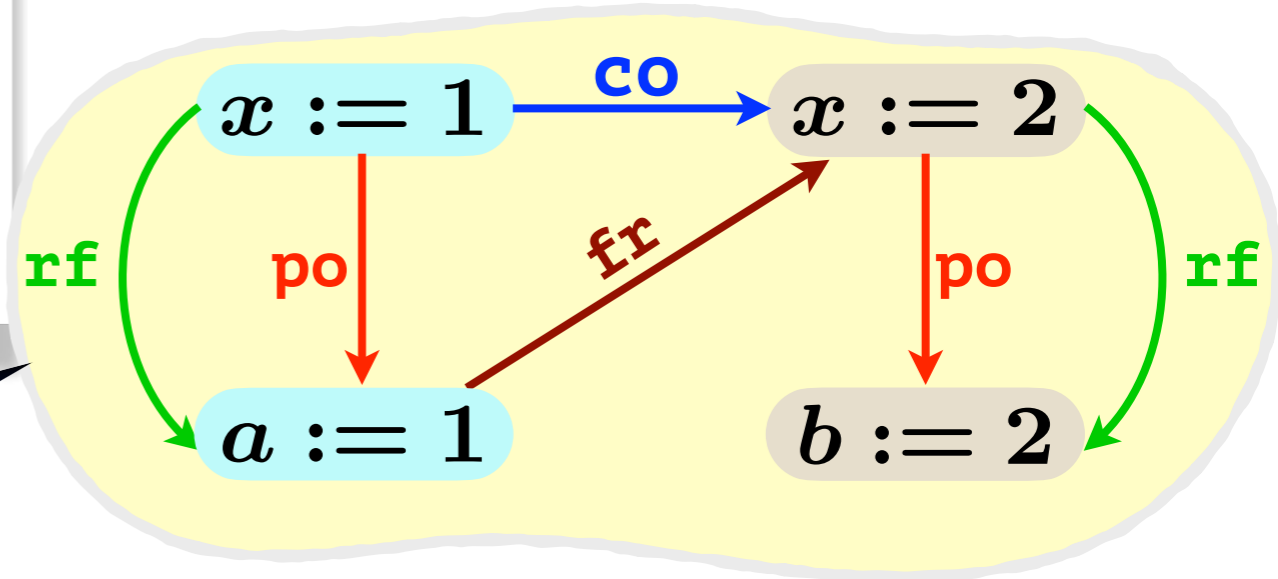


Traces

- efficient:
 - more compact than runs
 - sufficient for checking assertions
- abstract:
 - allow different memory models

- po**: program order
- rf**: read-from
- co**: coherence order
- fr**: from-read

$\tau : \text{trace}$



SC: $acyclic(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

$co = \bigcup_{x \in X} co^x$

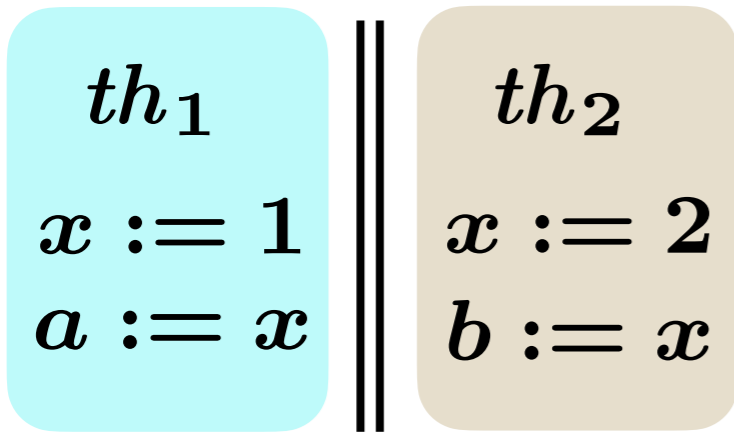
total

$\tau \models SC$

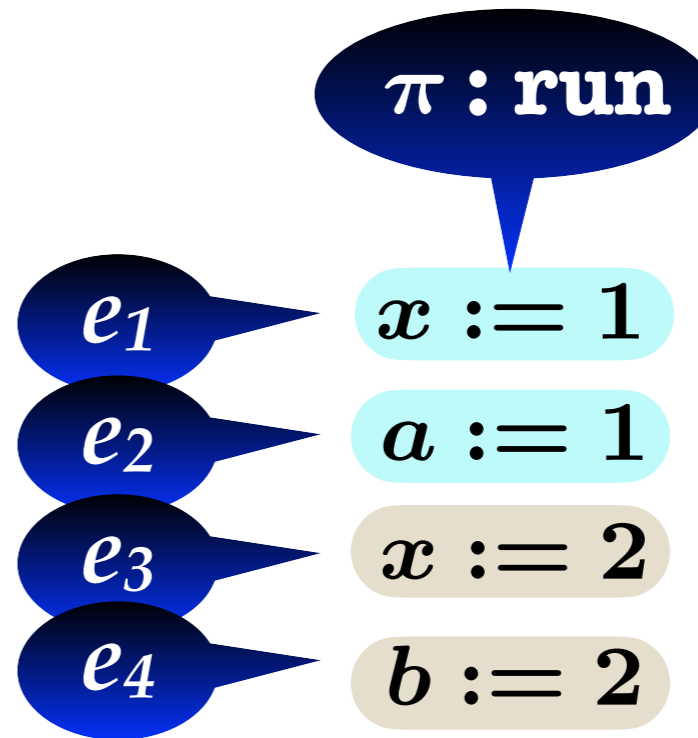
$\pi \models \tau$

Concurrent Programs (under SC)

Initially $x = 0$



events



$(a = 1) \wedge (b = 2)?$

efficient:

are compact than runs

assertion

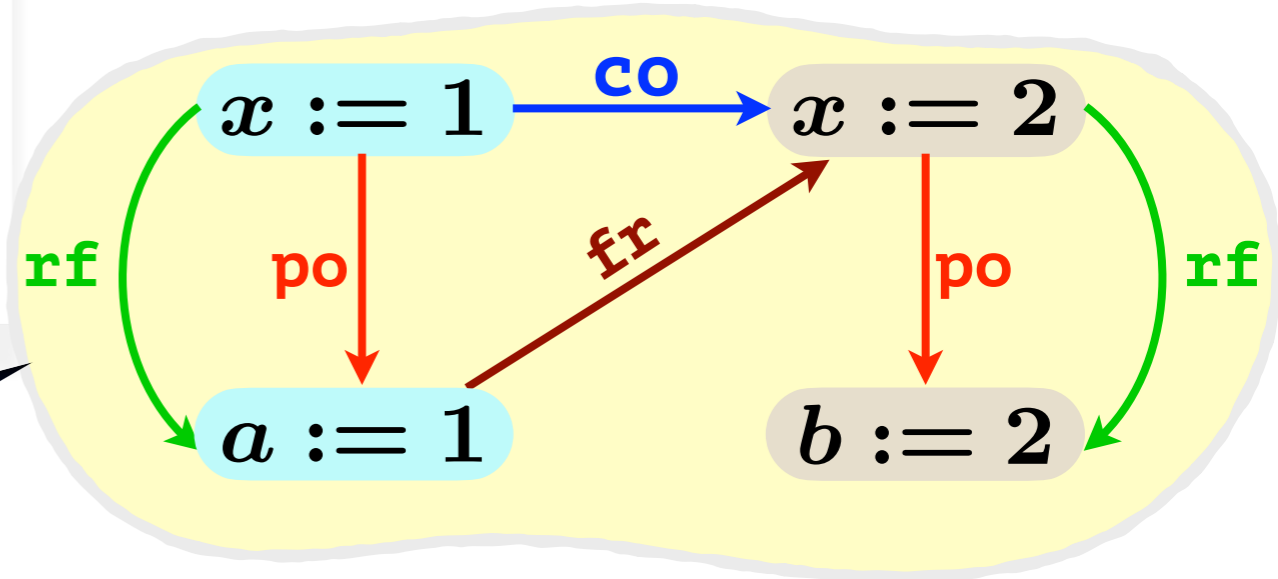
are convenient for checking assertions

abstract:

allows different memory models

- **po**: program order
- **rf**: read-from
- **co**: coherence order
- **fr**: from-read

$\tau : \text{trace}$



SC: $acyclic(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

$\text{co} = \bigcup_{x \in X} \text{co}^x$

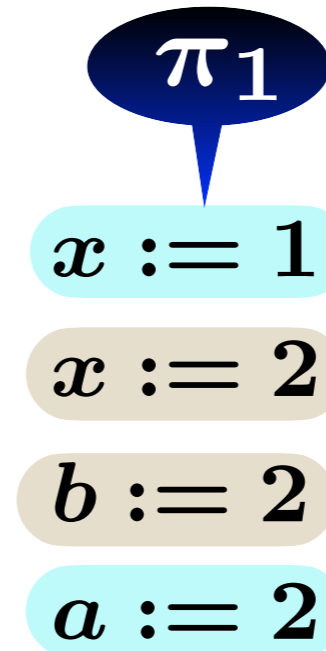
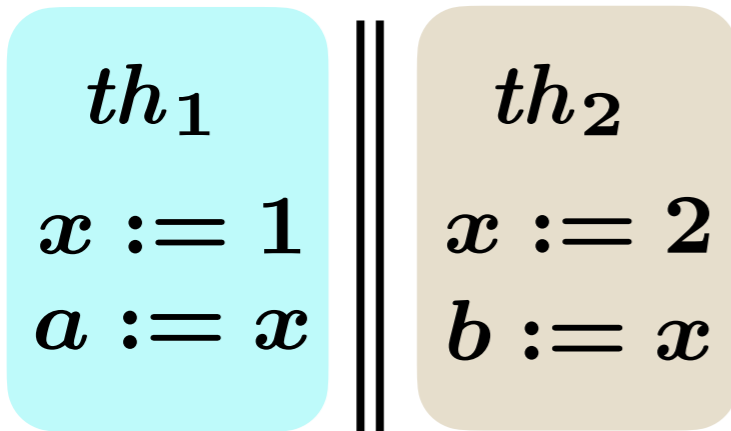
total

$\tau \models \text{SC}$

$\pi \models \tau$

Concurrent Programs (under SC)

Initially $x = 0$

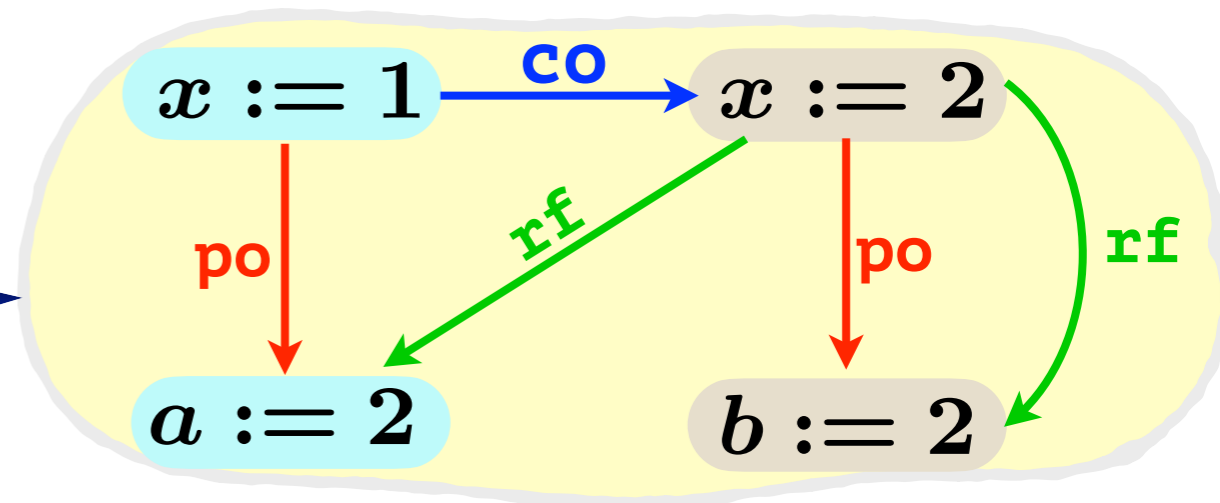


• Traces

- **efficient:**
 - more compact than runs
 - sufficient for checking assertions
- **abstract:**
 - allows different memory models

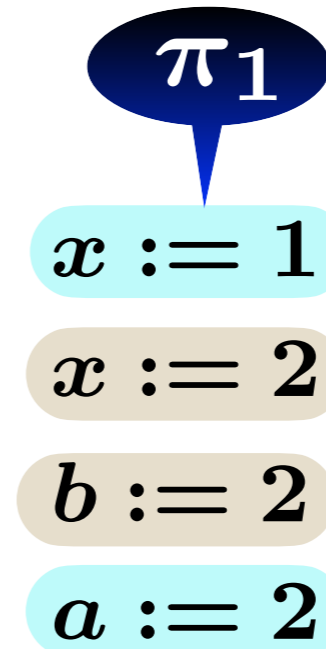
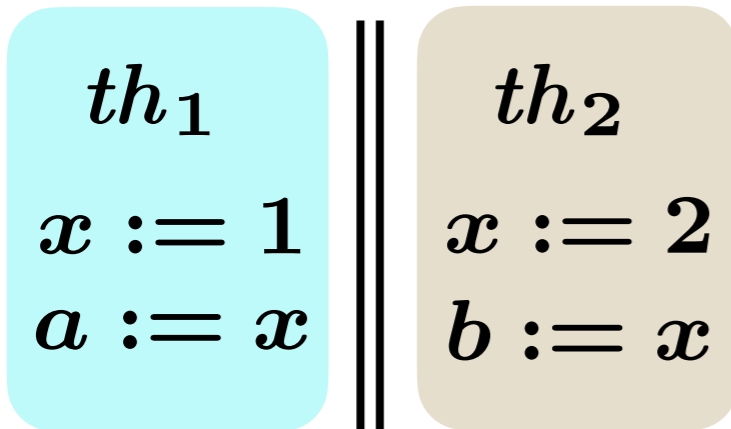
- **po**: program order
- **rf**: read-from
- **co**: coherence order
- **fr**: from-read

τ



Concurrent Programs (under SC)

Initially $x = 0$

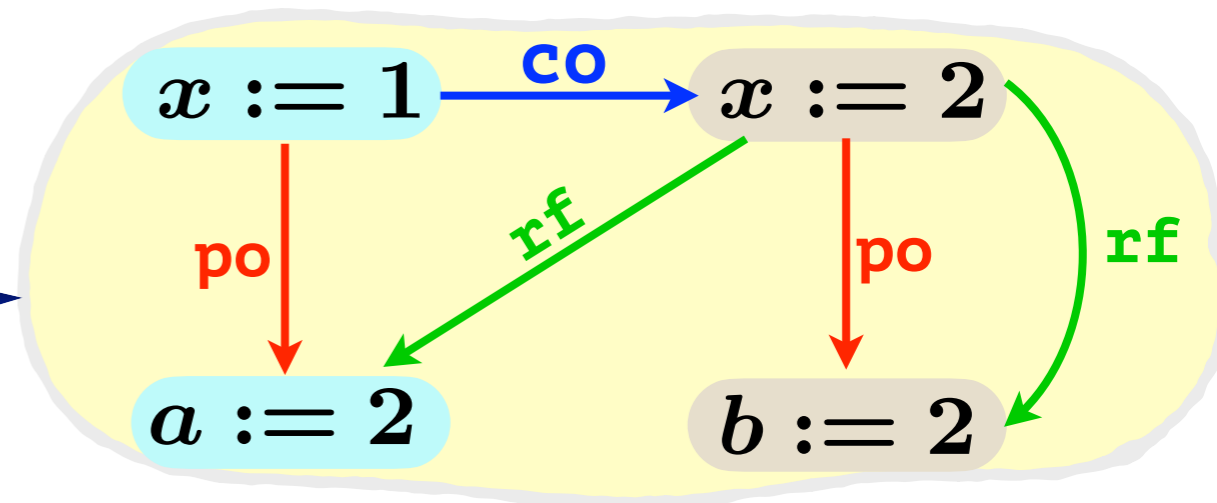


• Traces

- **efficient:**
 - **more compact than runs**
 - **sufficient for checking assertions**
- **abstract:**
 - **allows different memory models**

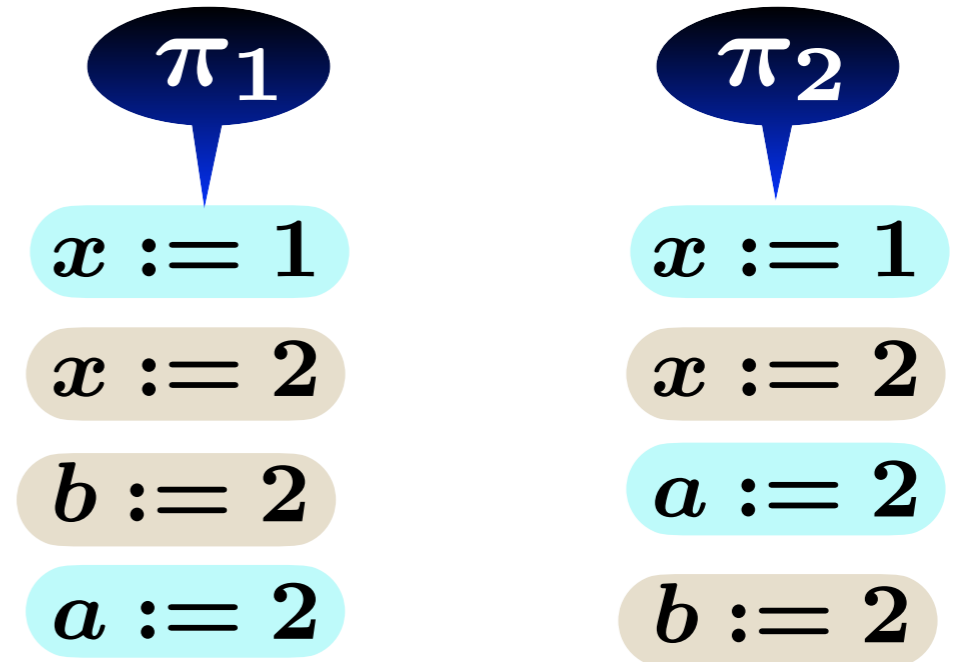
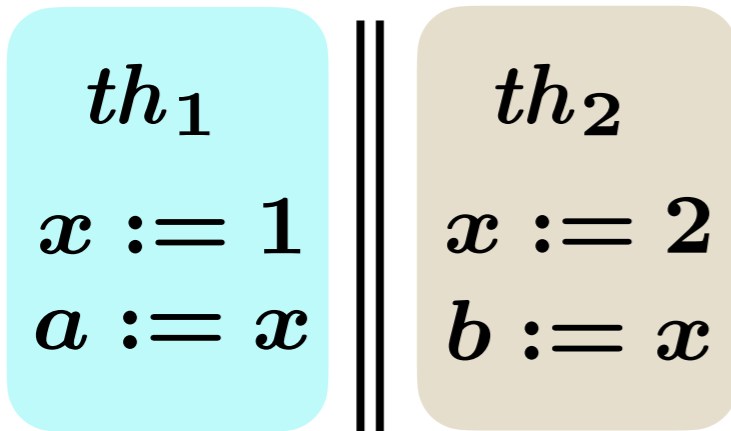
- **po**: program order
- **rf**: read-from
- **co**: coherence order
- **fr**: from-read

τ



Concurrent Programs (under SC)

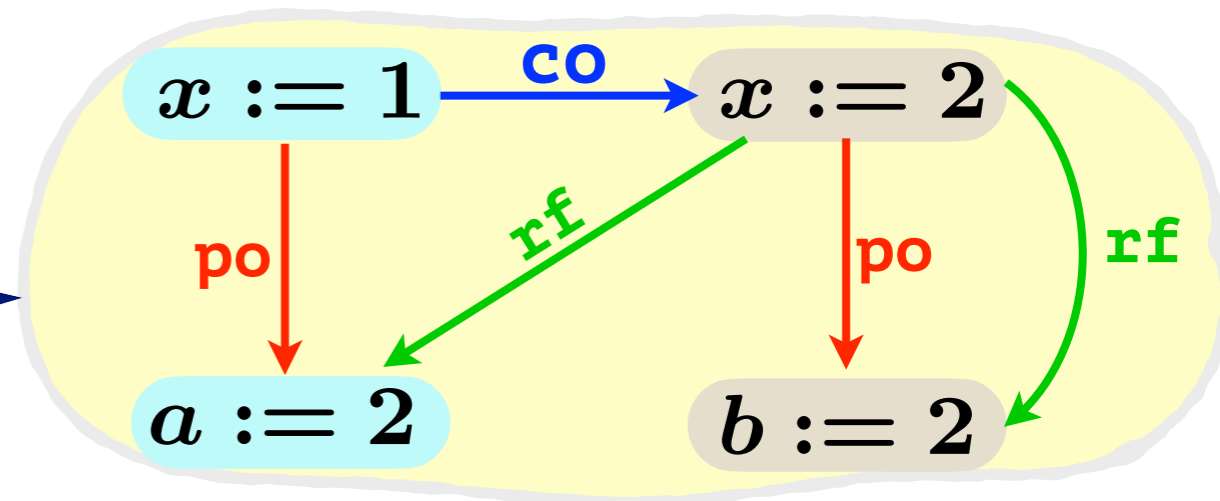
Initially $x = 0$



• Traces

- **efficient:**
 - **more compact than runs**
 - **sufficient for checking assertions**
- **abstract:**
 - **allows different memory models**

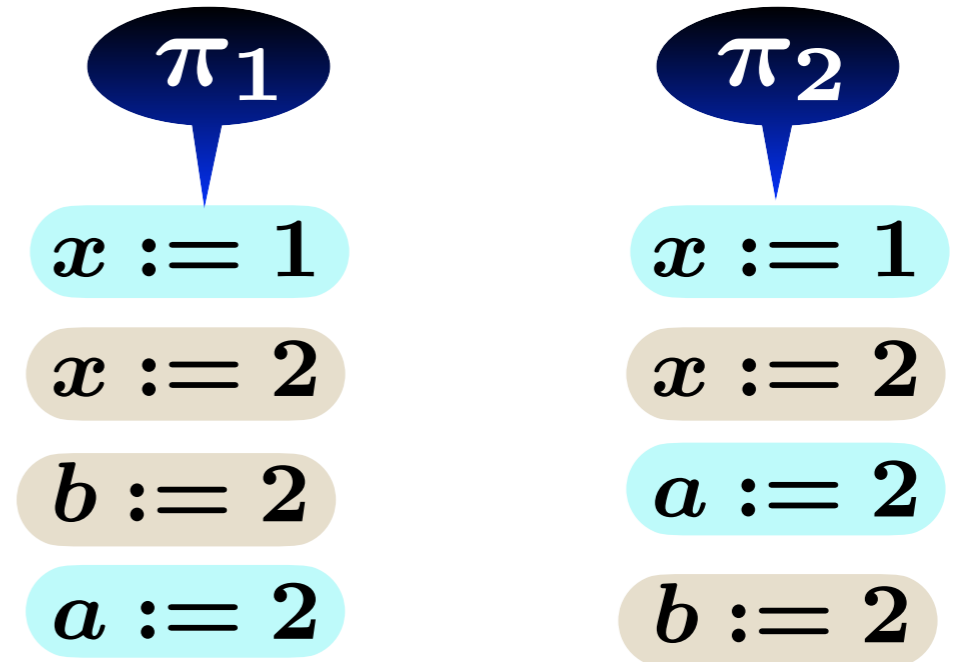
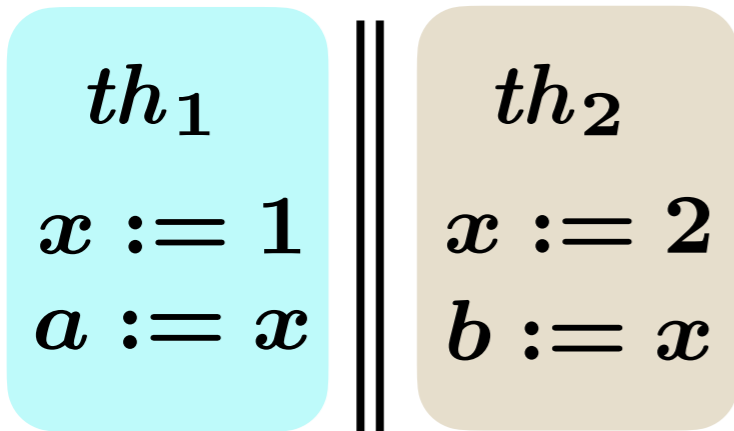
τ



- **po**: program order
- **rf**: read-from
- **co**: coherence order
- **fr**: from-read

Concurrent Programs (under SC)

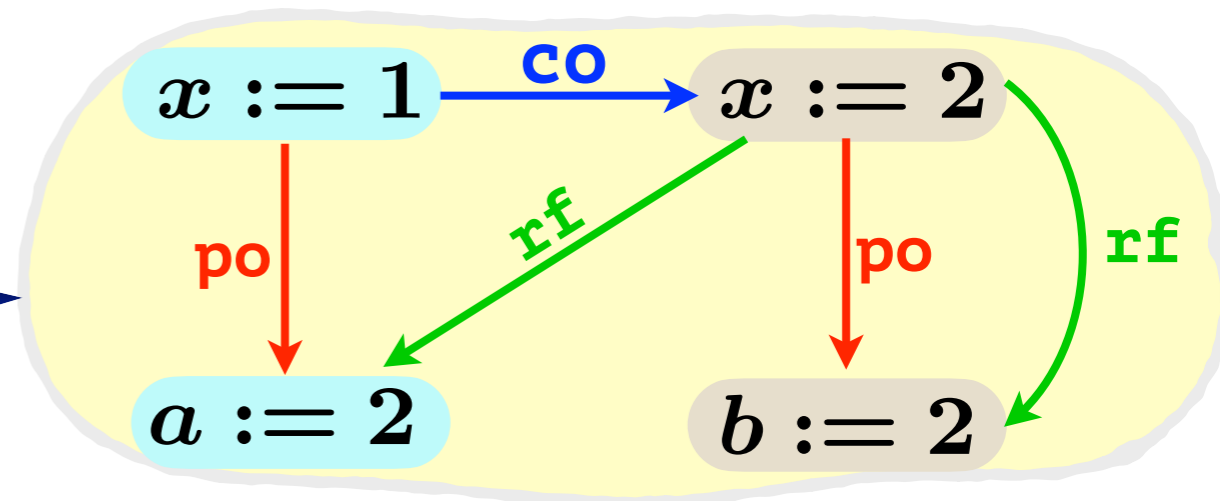
Initially $x = 0$



• Traces

- **efficient:**
 - more compact than runs
 - **sufficient for checking assertions**
- **abstract:**
 - **allows different memory models**

τ



- **po**: program order
- **rf**: read-from
- **co**: coherence order
- **fr**: from-read

$$\pi_1 \models \tau$$

$$\pi_2 \models \tau$$

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- **Conclusions**

“first let us consider SC”

Outline

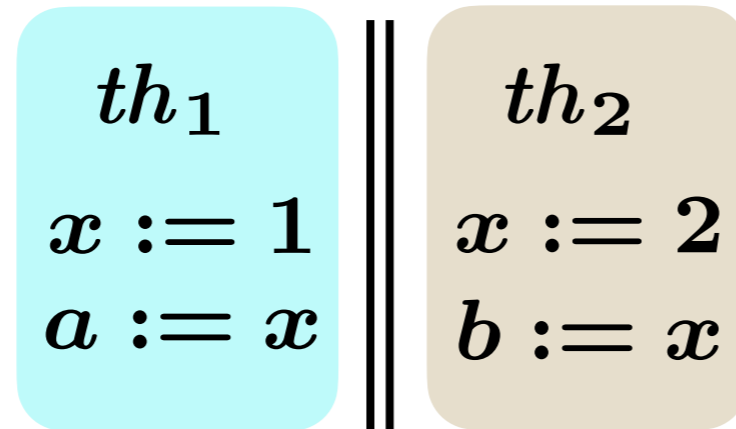
- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- **Conclusions**

Traces

“first let us consider SC”

DPOR: Dynamic Partial Order Reduction

Initially $x = 0$

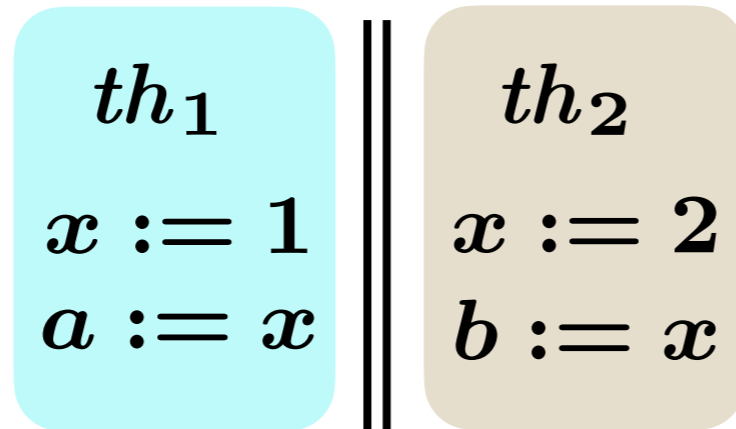


(optimal)
DPOR

(POPL'14)

DPOR: Dynamic Partial Order Reduction

Initially $x = 0$



Prog

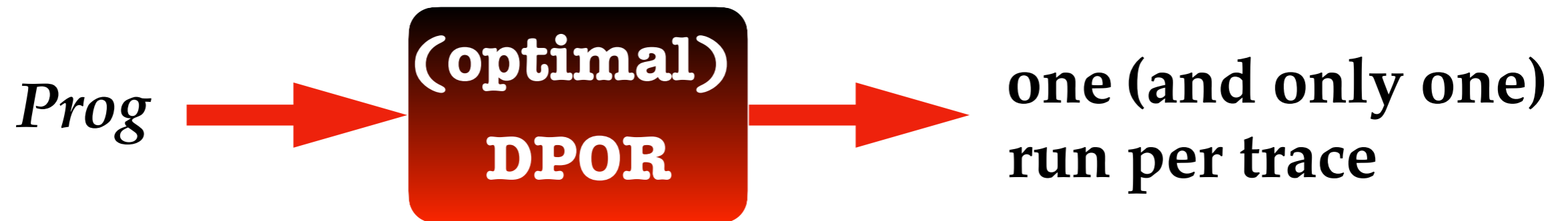
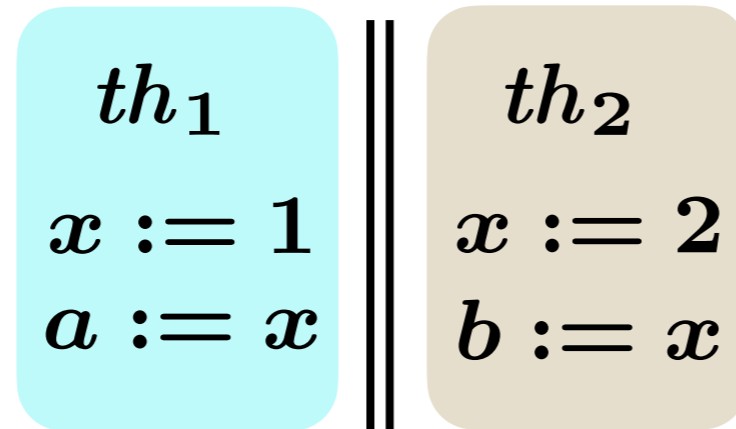


(optimal)
DPOR

(POPL'14)

DPOR: Dynamic Partial Order Reduction

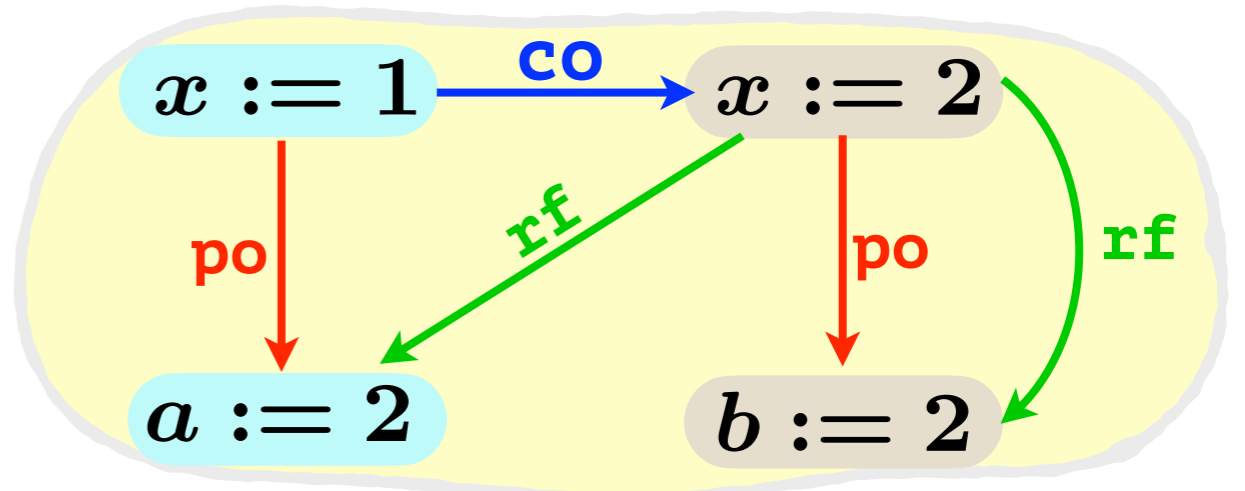
Initially $x = 0$



(POPL'14)

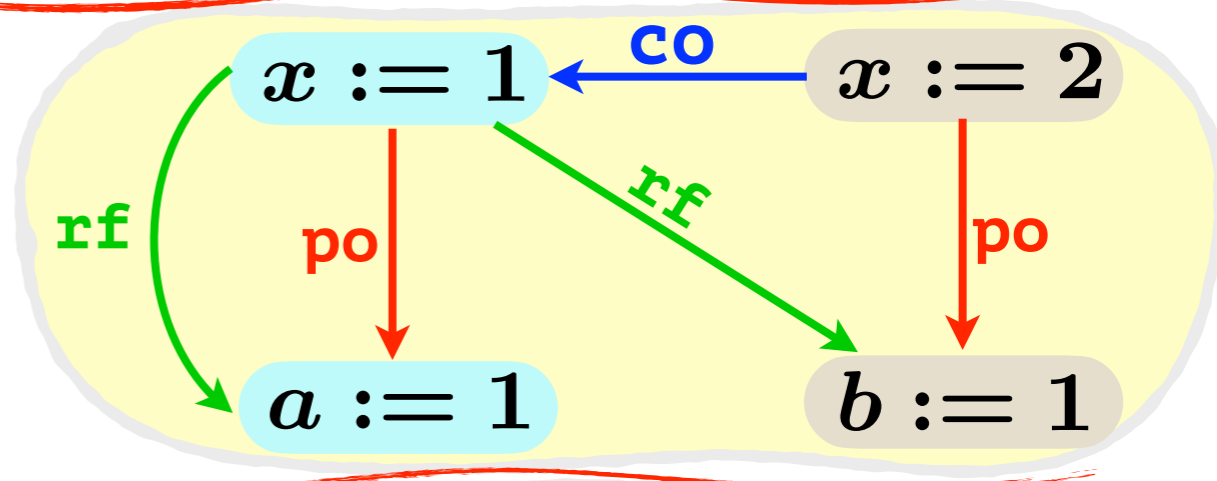
$x := 1$
 $x := 2$
 $a := 2$
 $b := 2$

$x := 1$
 $x := 2$
 $b := 2$
 $a := 2$

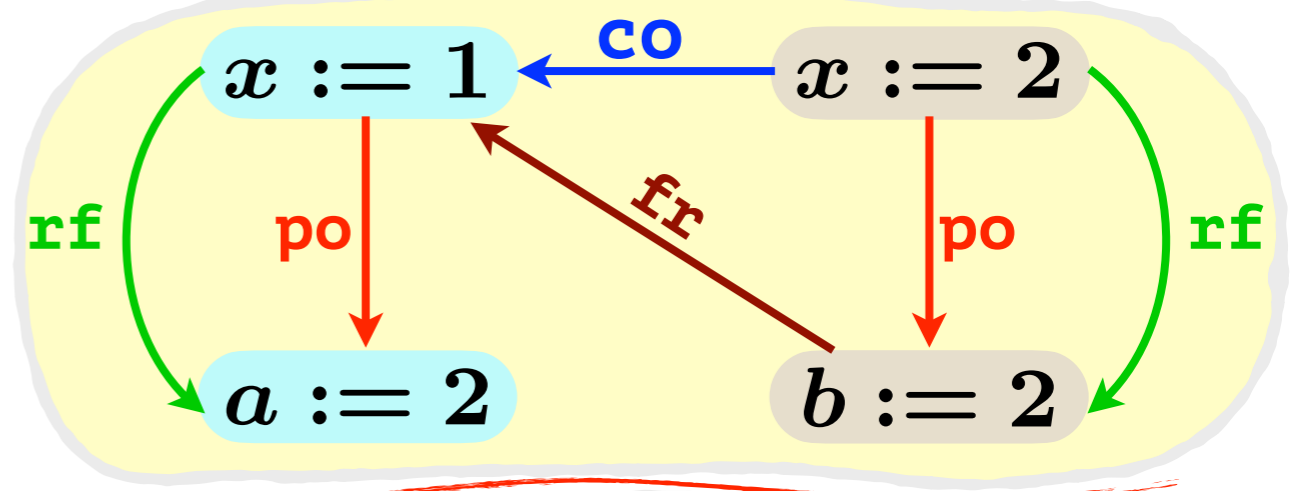


$x := 2$
 $x := 1$
 $b := 1$
 $a := 1$

$x := 2$
 $x := 1$
 $a := 1$
 $b := 1$

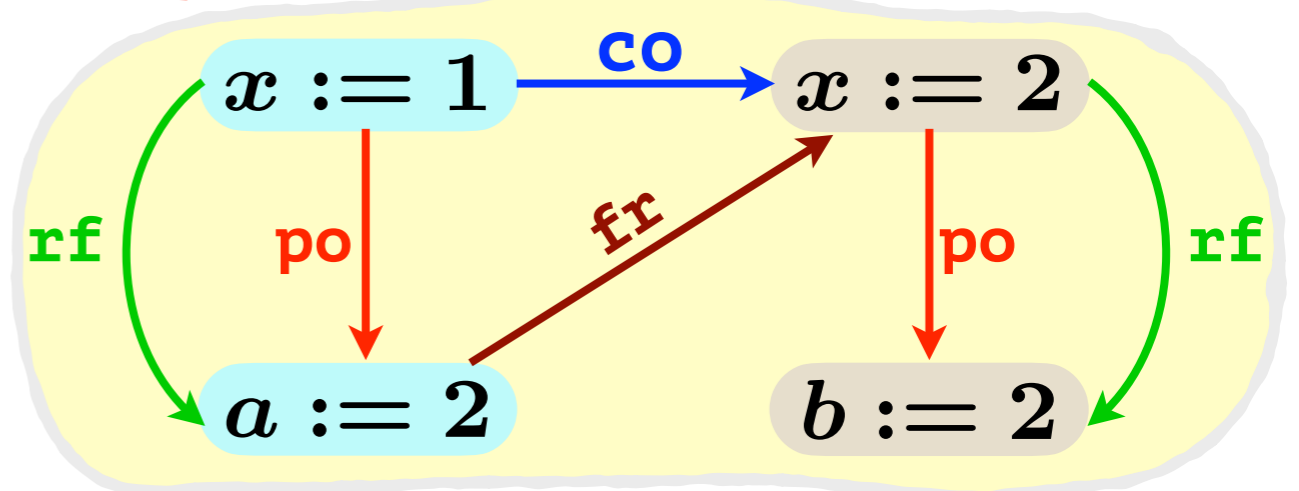


$x := 1$
 $a := 1$
 $x := 2$
 $b := 2$



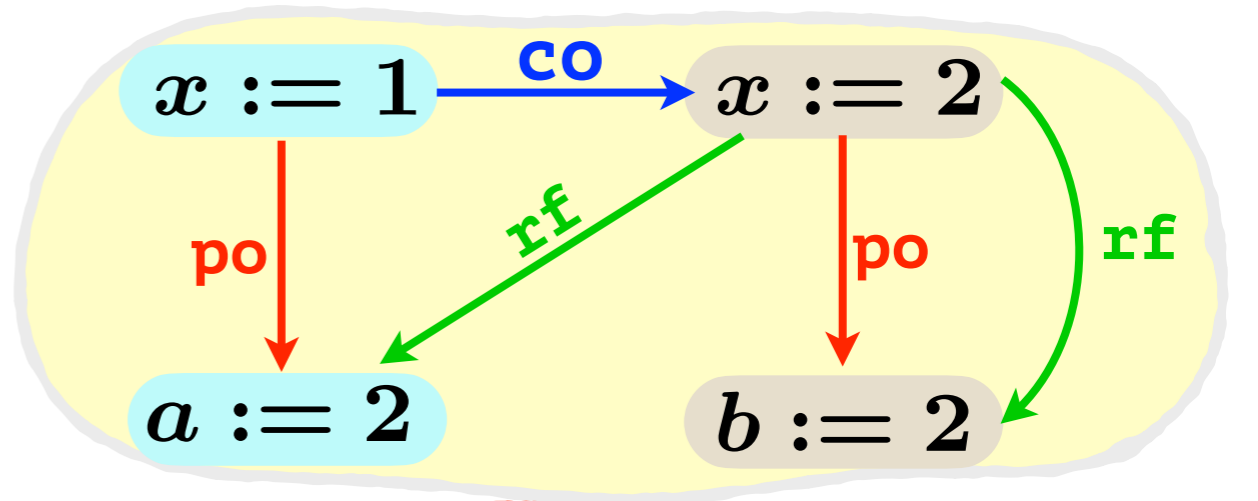
**(optimal)
 DPOR**

$x := 2$
 $b := 2$
 $x := 1$
 $a := 1$



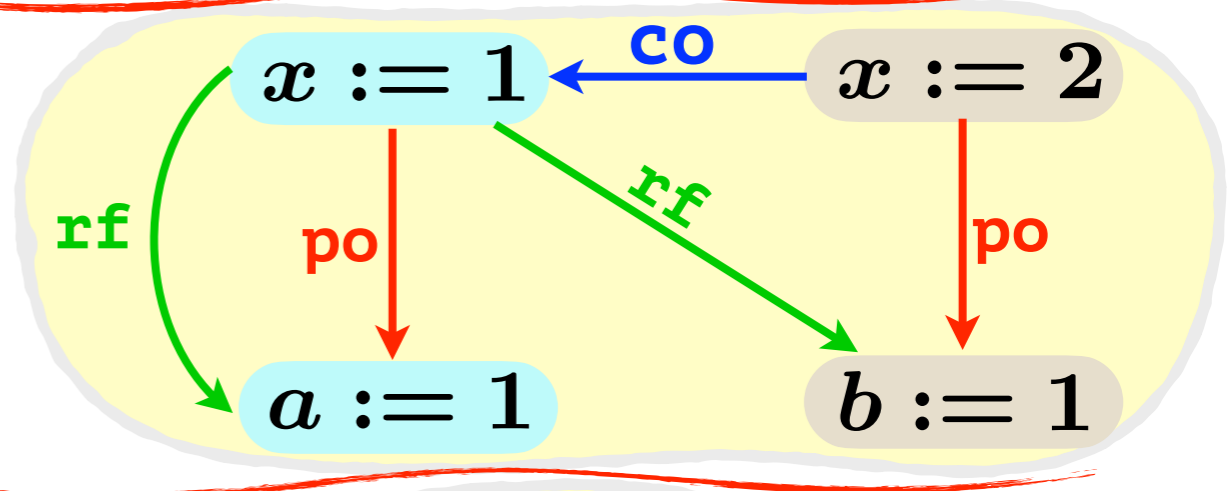
$x := 1$
 $x := 2$
 $a := 2$
 $b := 2$

$x := 1$
 $x := 2$
 $b := 2$
 $a := 2$

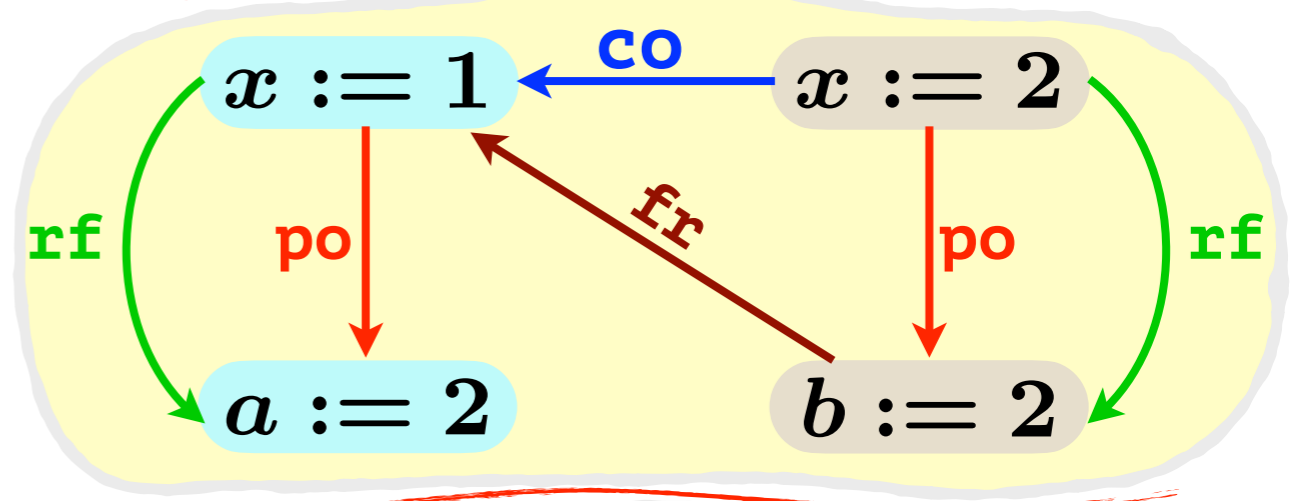


$x := 2$
 $x := 1$
 $b := 1$
 $a := 1$

$x := 2$
 $x := 1$
 $a := 1$
 $b := 1$

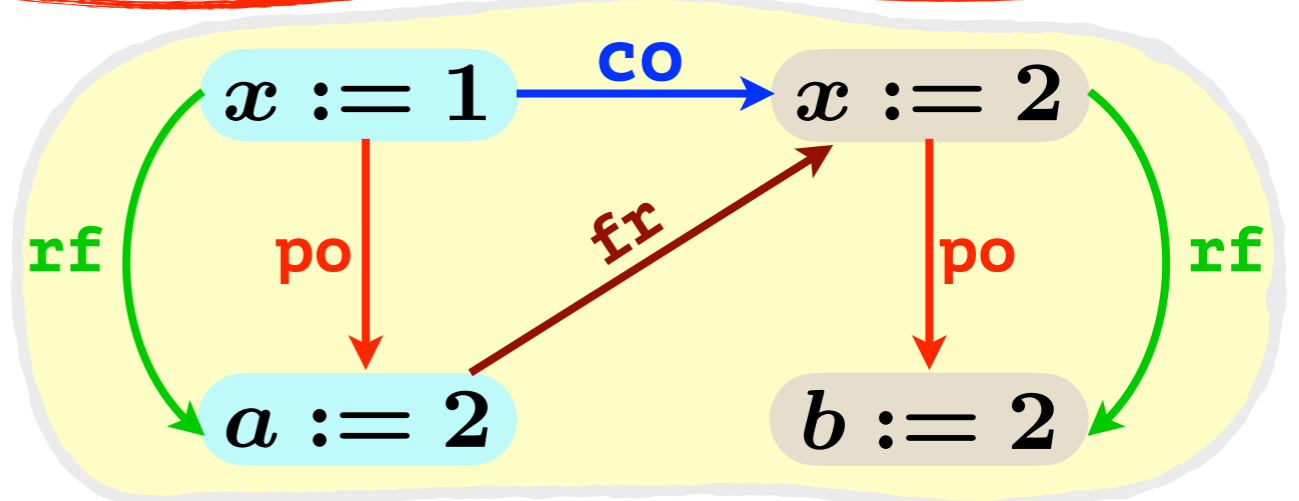


$x := 1$
 $a := 1$
 $x := 2$
 $b := 2$



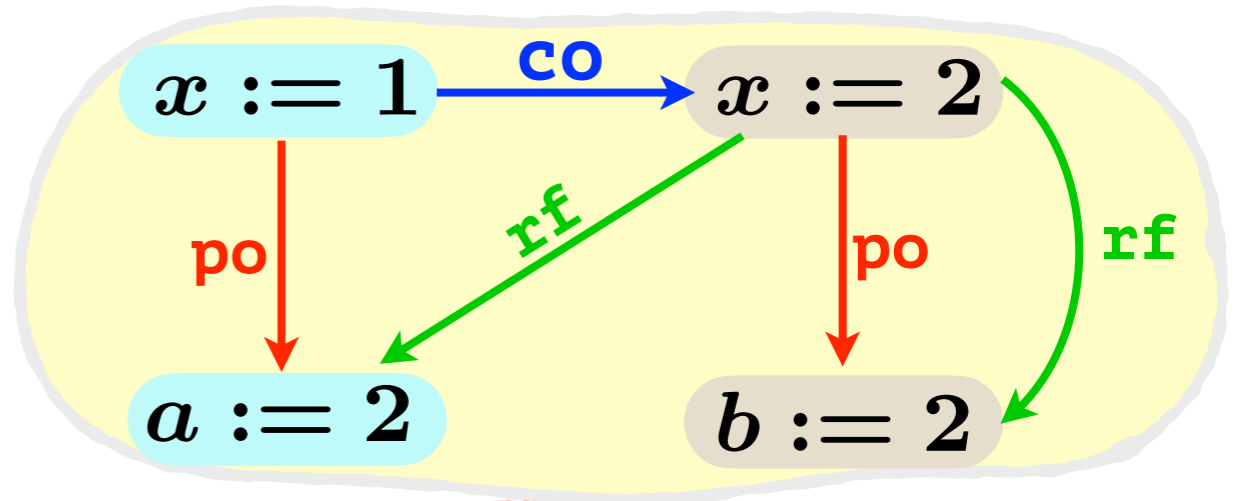
(optimal)
DPOR

$x := 2$
 $b := 2$
 $x := 1$
 $a := 1$



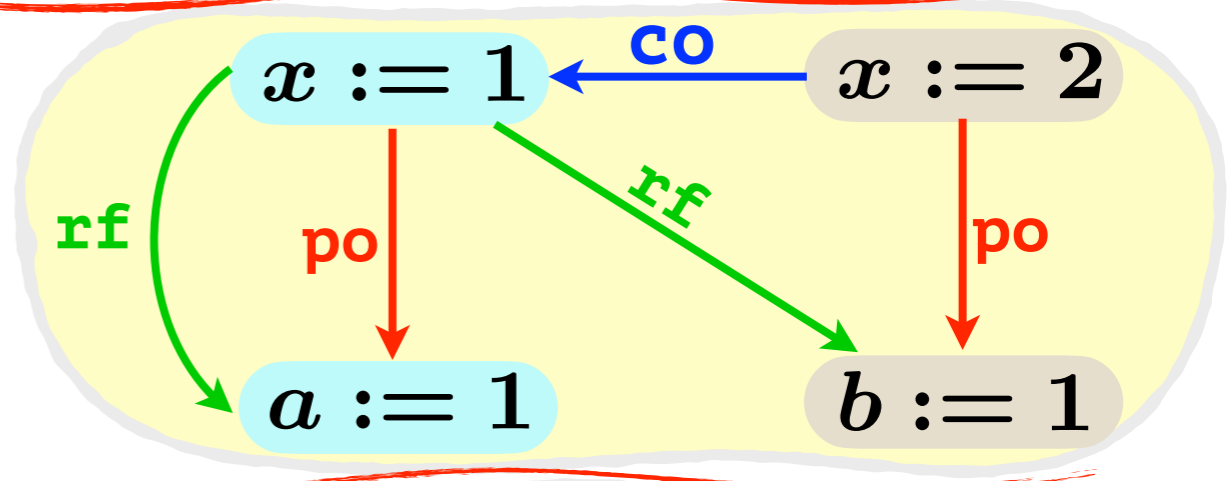
$x := 1$
 $x := 2$
 $a := 2$
 $b := 2$

$x := 1$
 $x := 2$
 $b := 2$
 $a := 2$

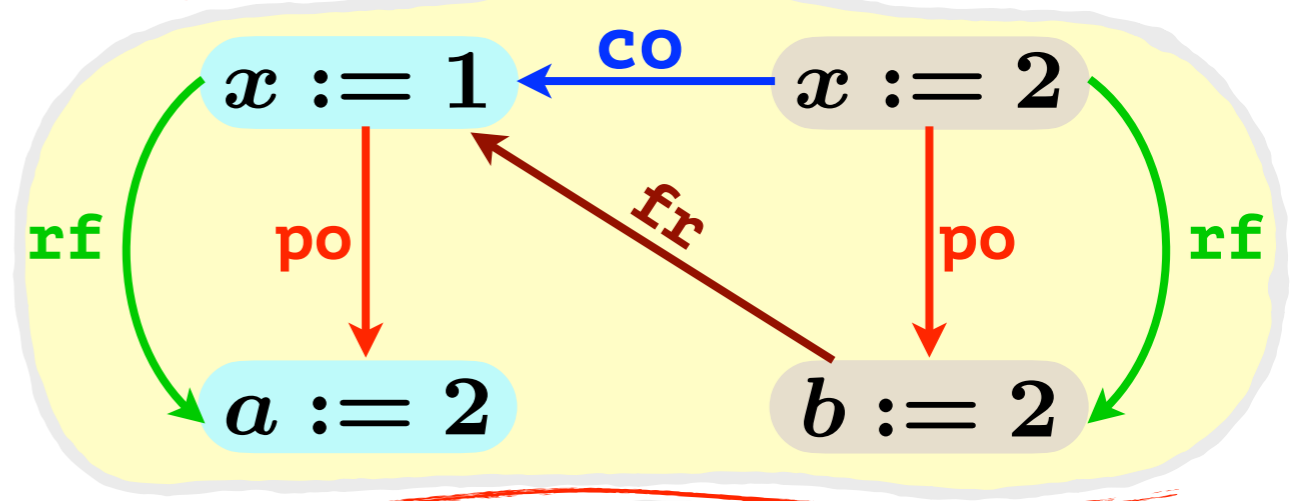


$x := 2$
 $x := 1$
 $b := 1$
 $a := 1$

$x := 2$
 $x := 1$
 $a := 1$
 $b := 1$

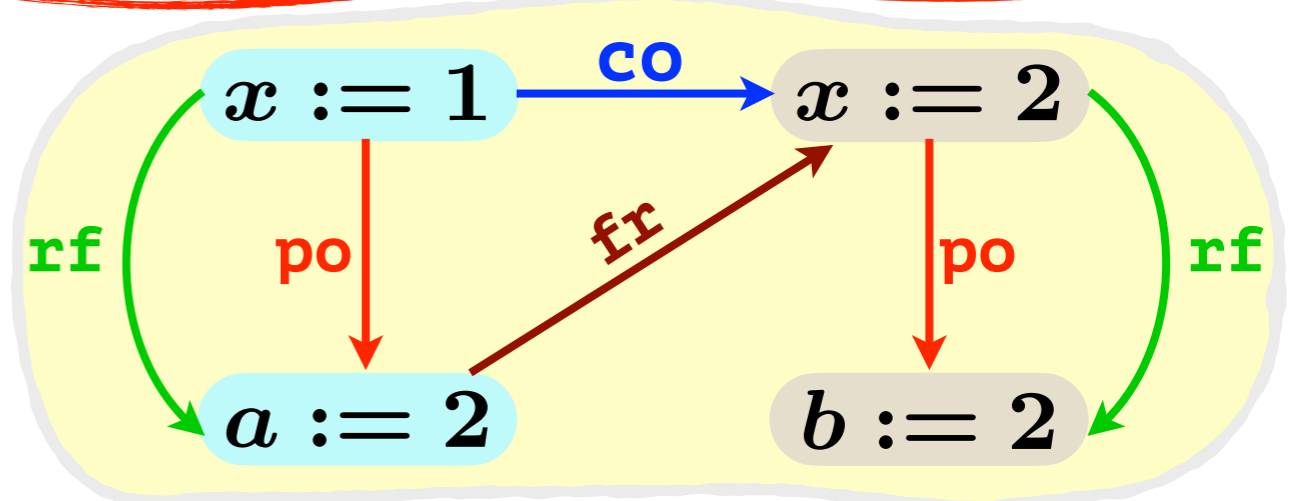


$x := 1$
 $a := 1$
 $x := 2$
 $b := 2$



**(optimal)
 DPOR**

$x := 2$
 $b := 2$
 $x := 1$
 $a := 1$

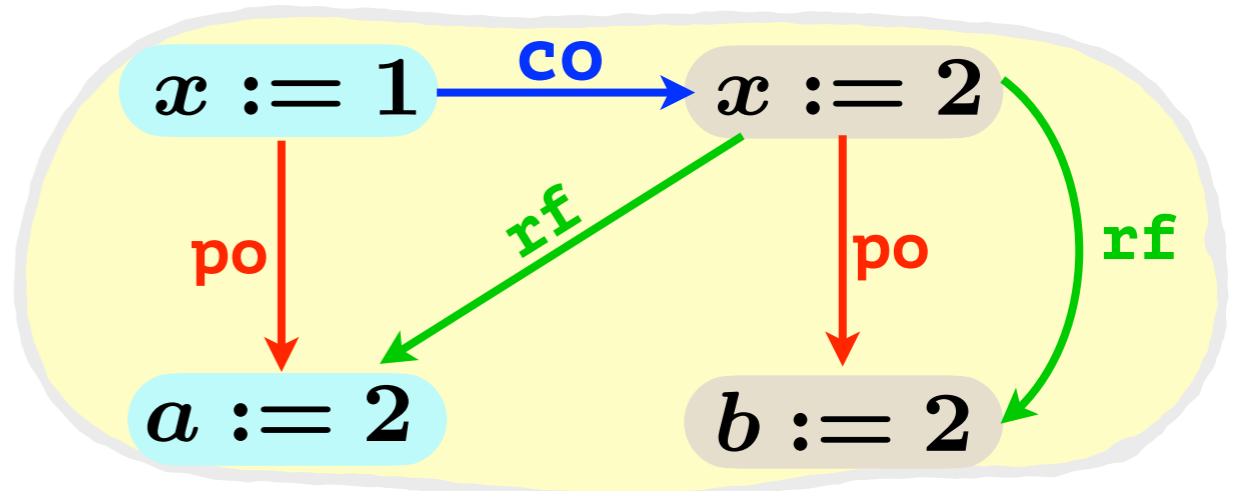


$x := 1$

$x := 2$

$a := 2$

$b := 2$

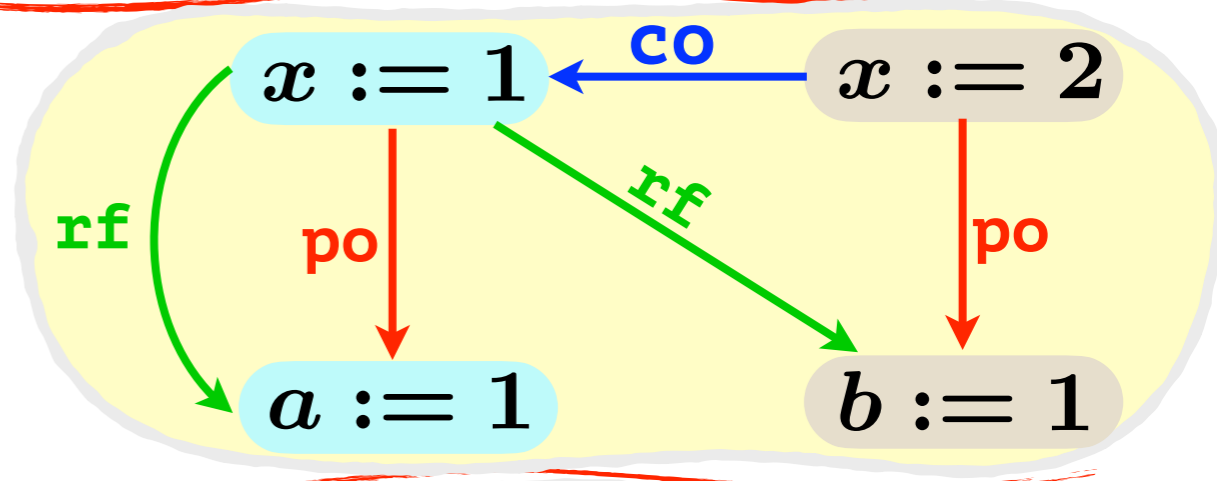


$x := 2$

$x := 1$

$b := 1$

$a := 1$

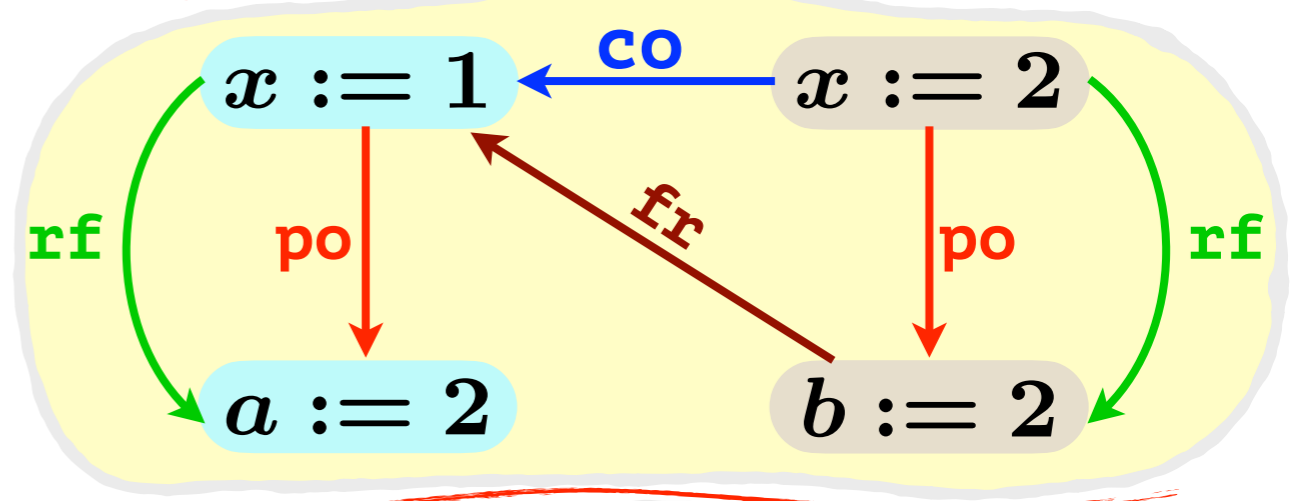


$x := 1$

$a := 1$

$x := 2$

$b := 2$



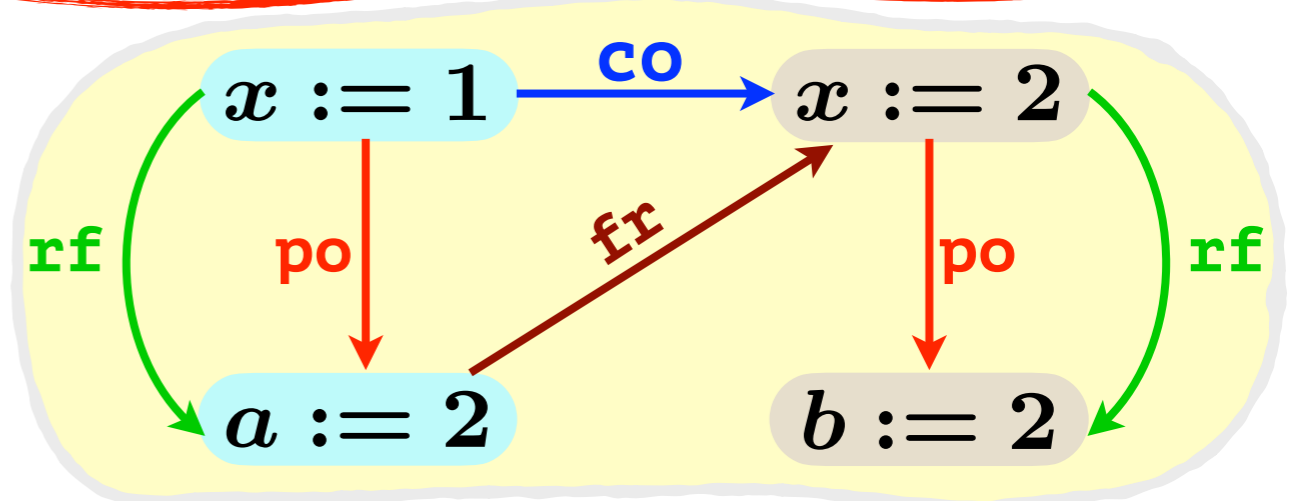
**(super-optimal)
DPOR**

$x := 2$

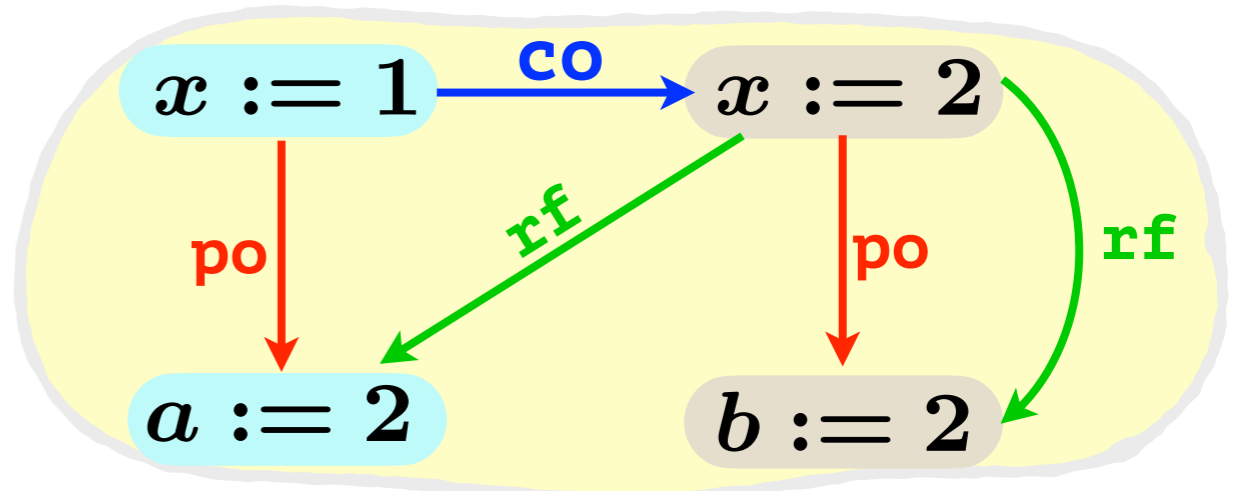
$b := 2$

$x := 1$

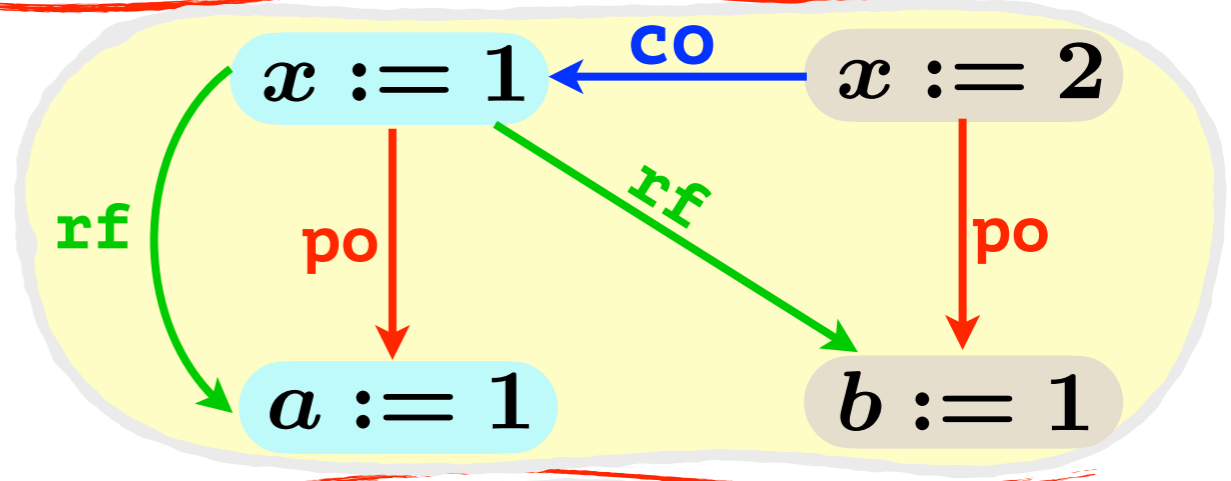
$a := 1$



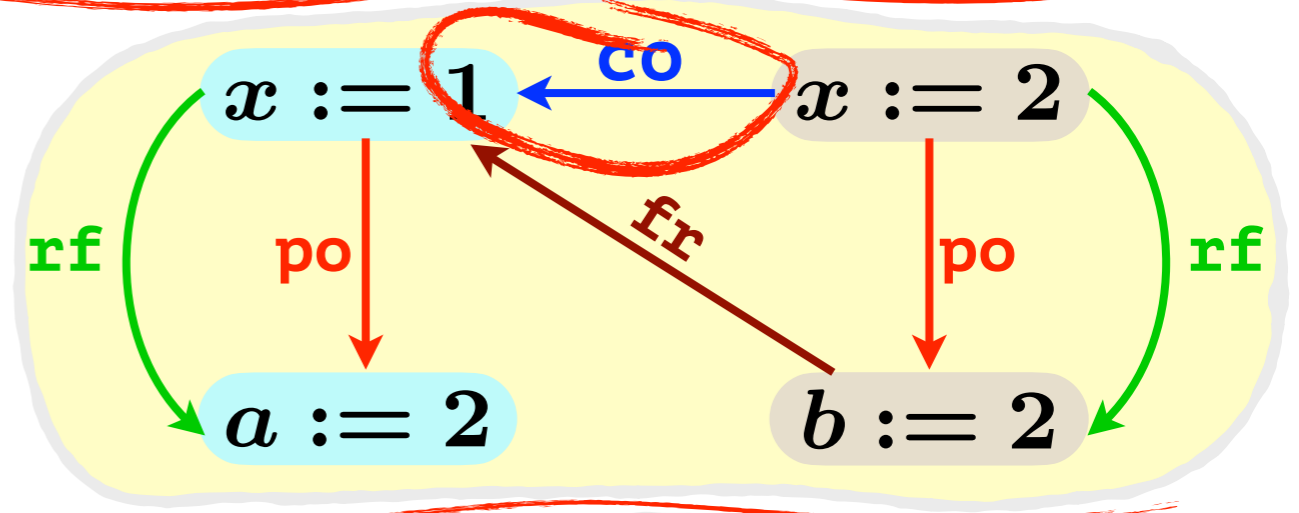
$x := 1$
 $x := 2$
 $a := 2$
 $b := 2$



$x := 2$
 $x := 1$
 $b := 1$
 $a := 1$

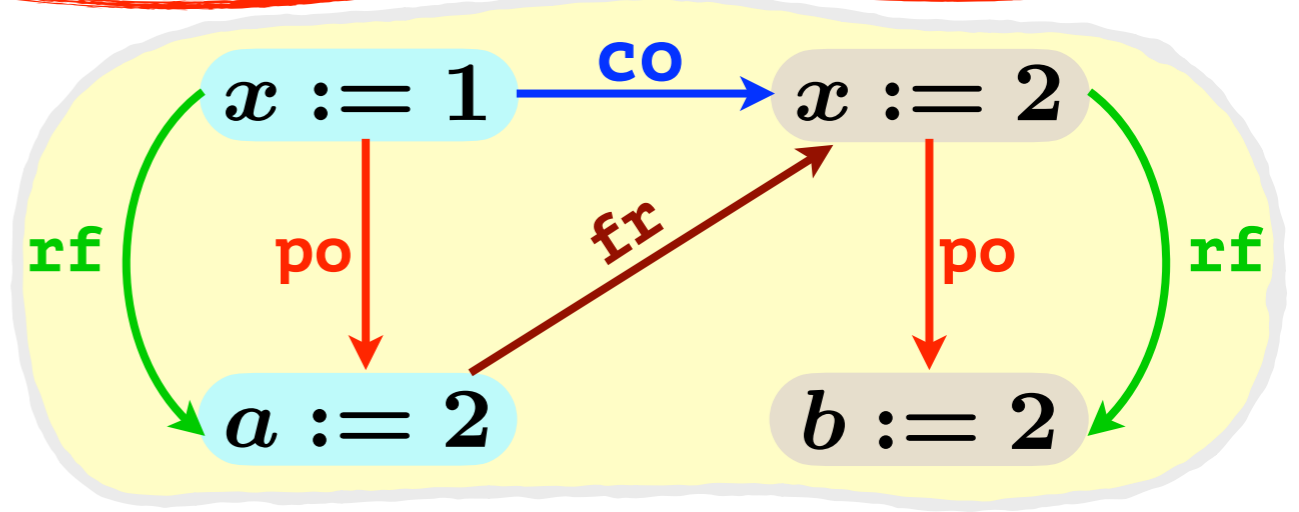


$x := 1$
 $a := 1$
 $x := 2$
 $b := 2$

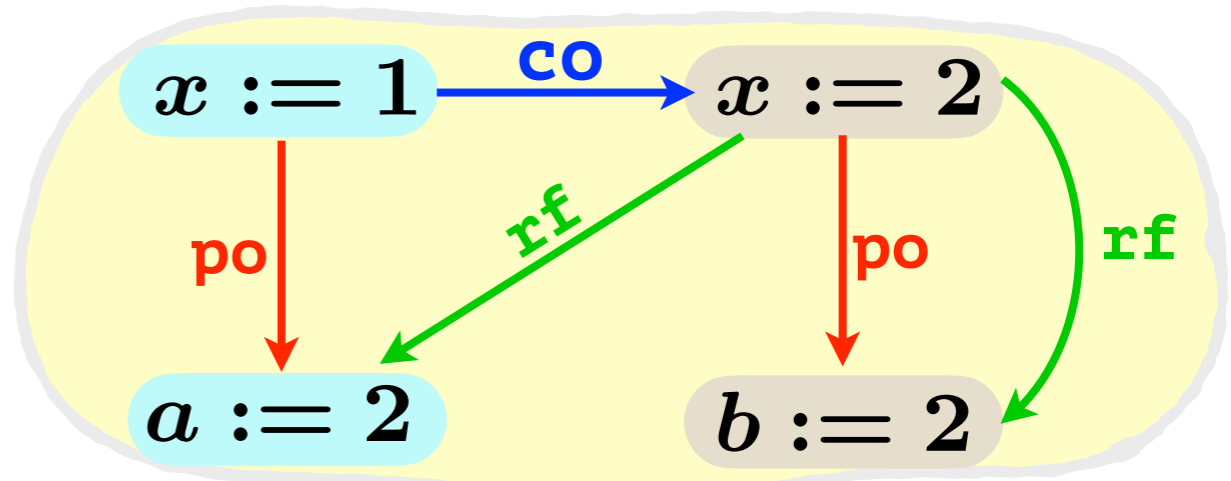


**(super-optimal)
 DPOR**

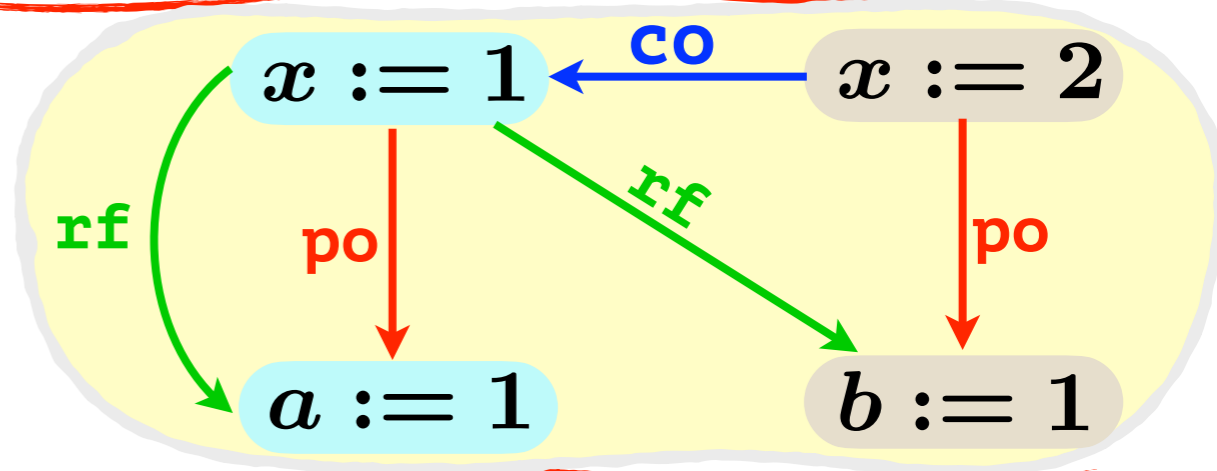
$x := 2$
 $b := 2$
 $x := 1$
 $a := 1$



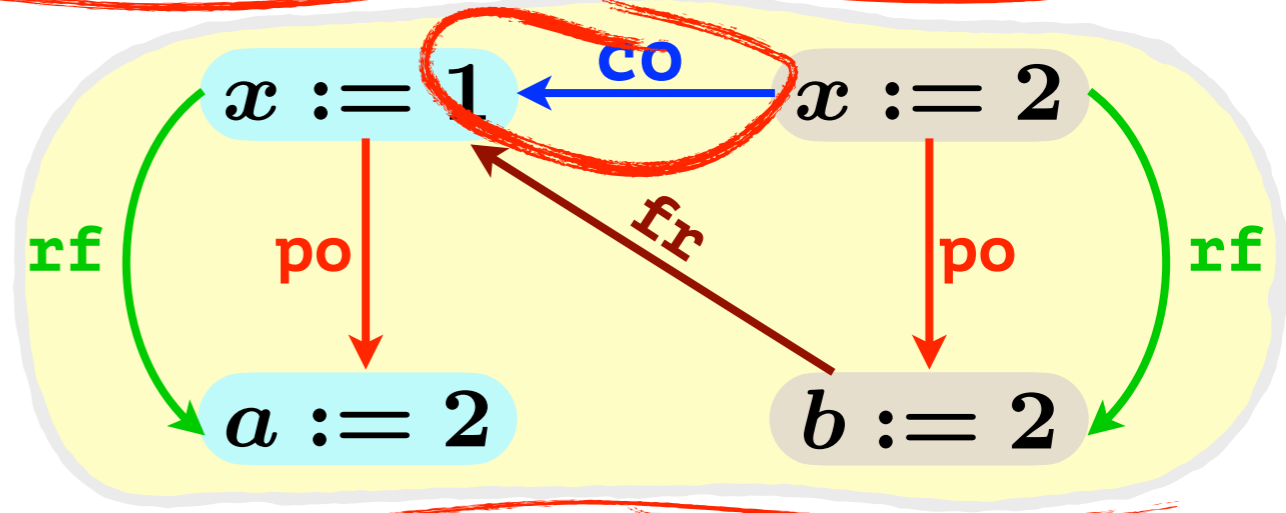
$x := 1$
 $x := 2$
 $a := 2$
 $b := 2$



$x := 2$
 $x := 1$
 $b := 1$
 $a := 1$

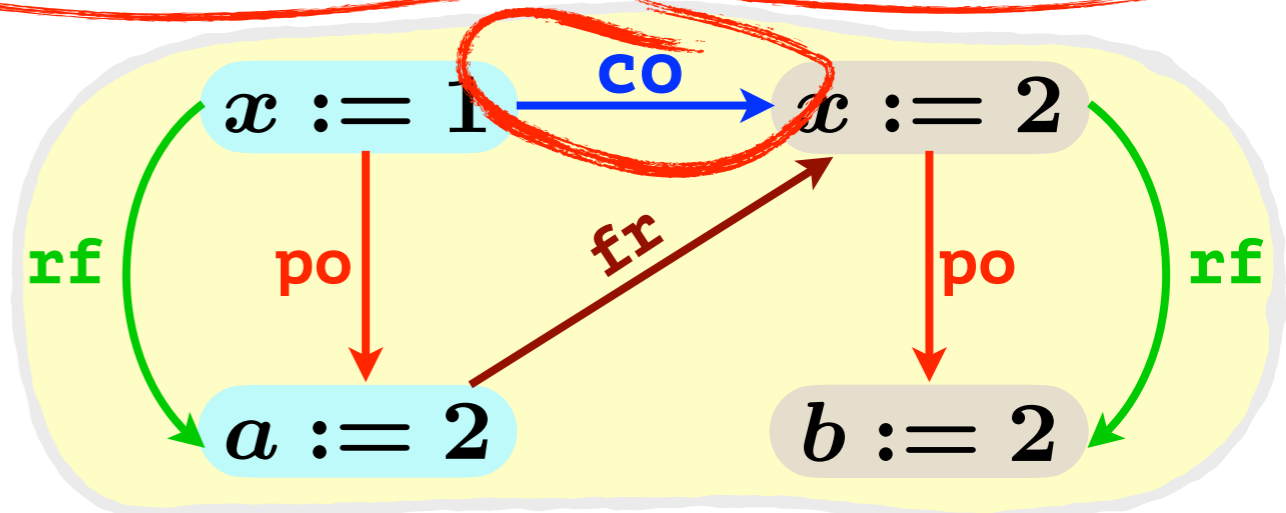


$x := 1$
 $a := 1$
 $x := 2$
 $b := 2$

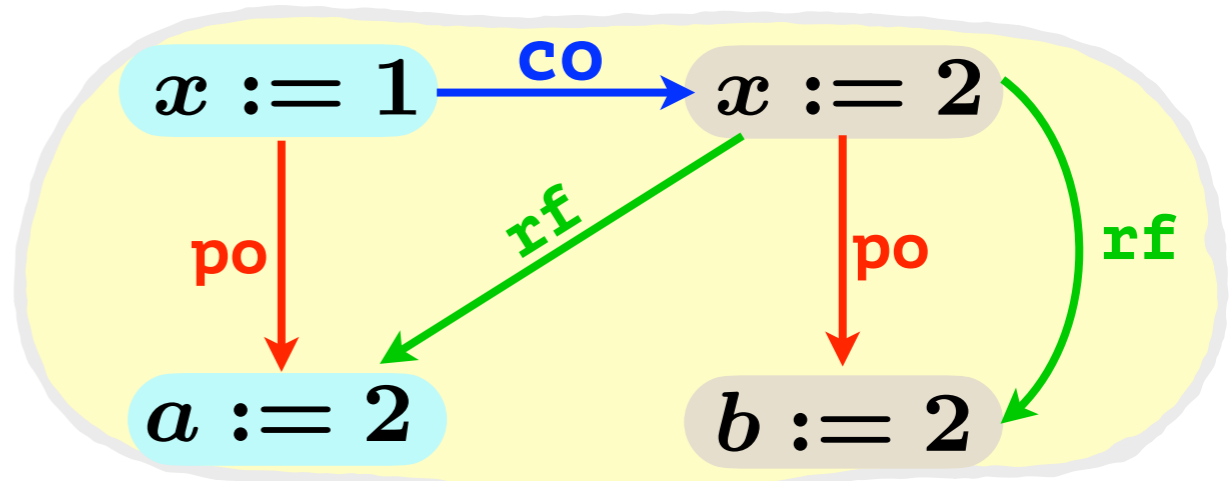


**(super-optimal)
 DPOR**

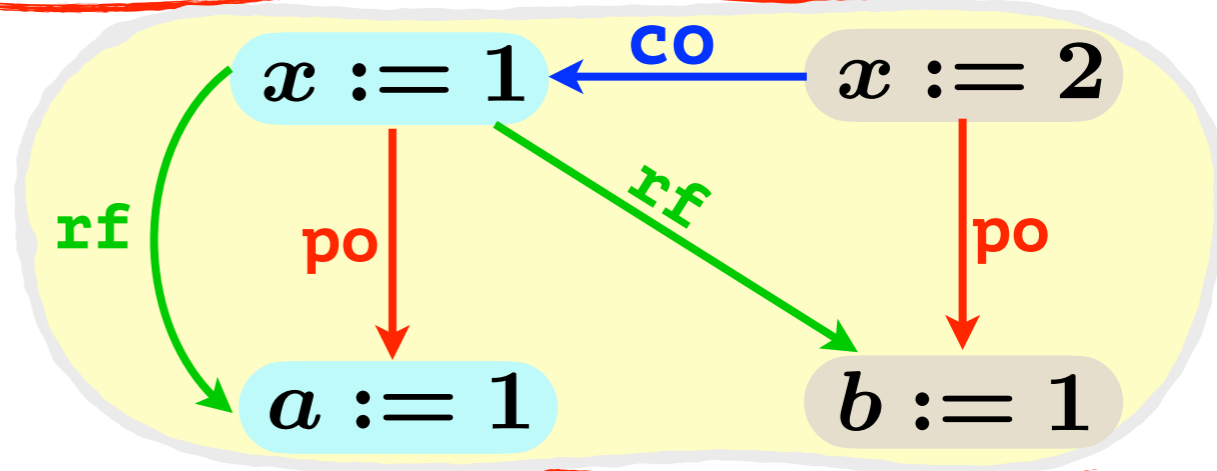
$x := 2$
 $b := 2$
 $x := 1$
 $a := 1$



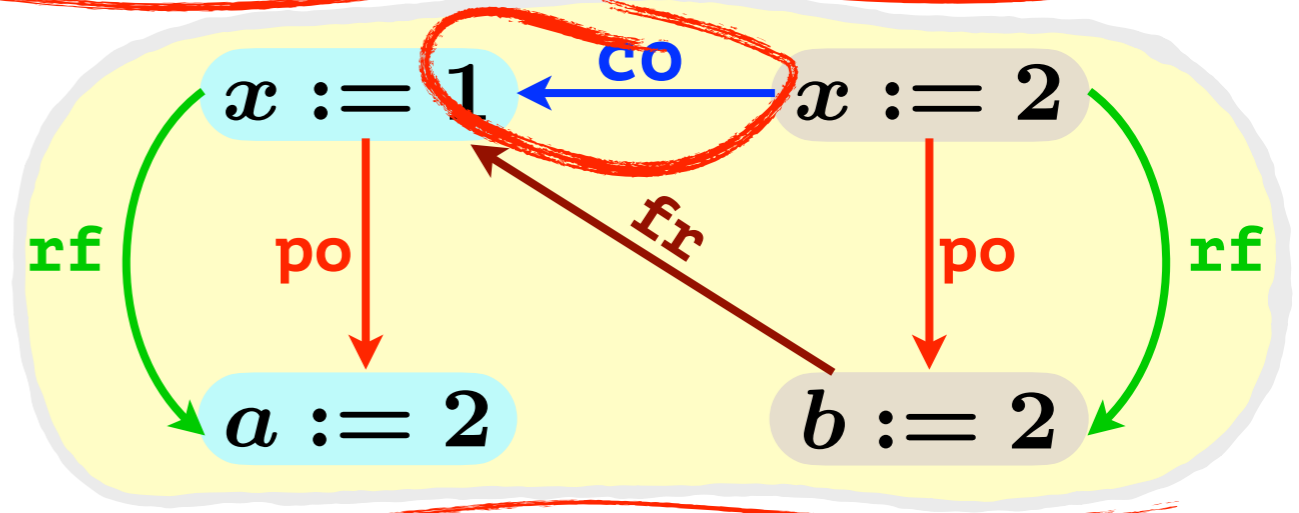
$x := 1$
 $x := 2$
 $a := 2$
 $b := 2$



$x := 2$
 $x := 1$
 $b := 1$
 $a := 1$



$x := 1$
 $a := 1$
 $x := 2$
 $b := 2$



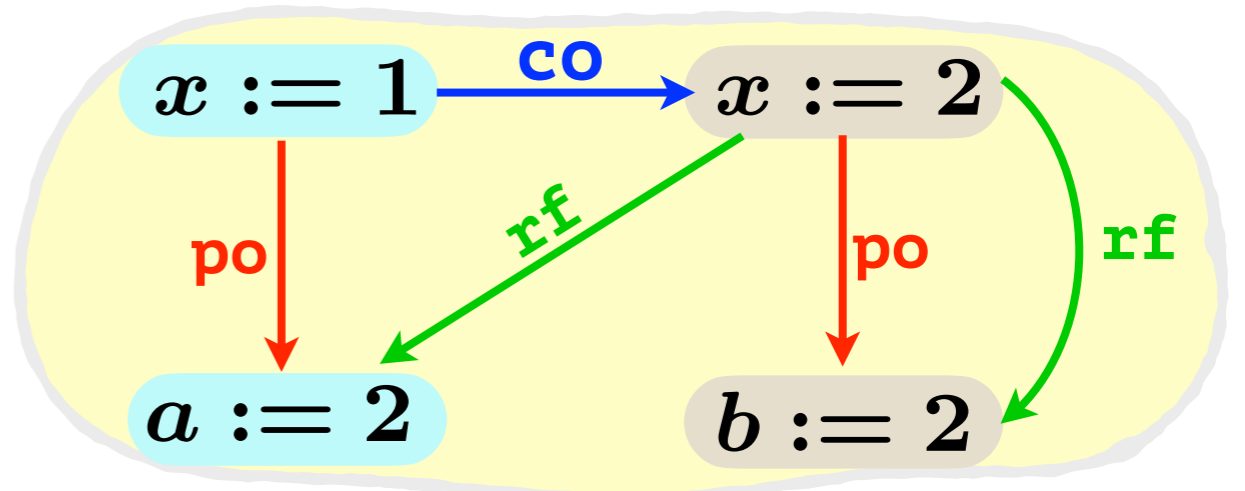
**(super-optimal)
 DPOR**

$x := 1$

$x := 2$

$a := 2$

$b := 2$

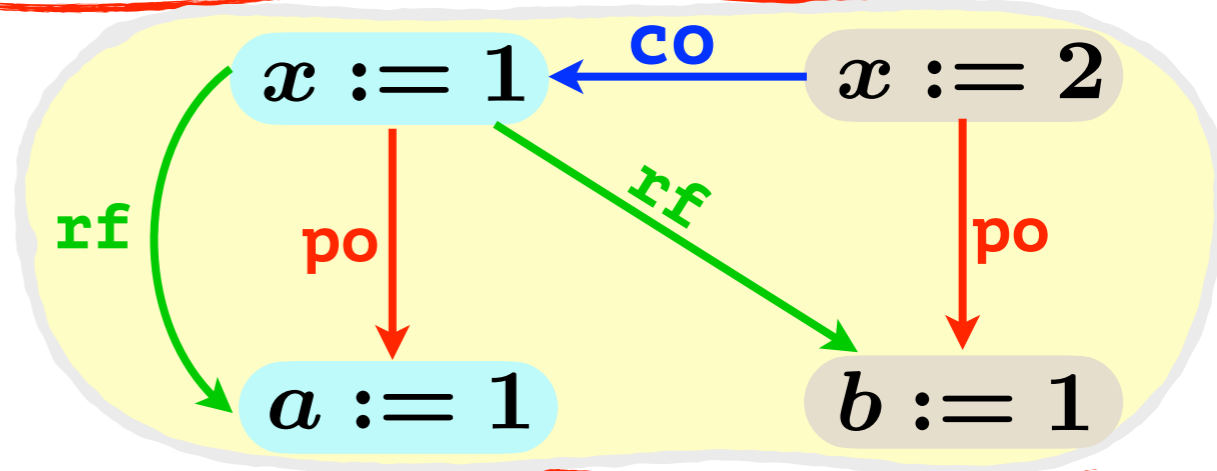


$x := 2$

$x := 1$

$b := 1$

$a := 1$

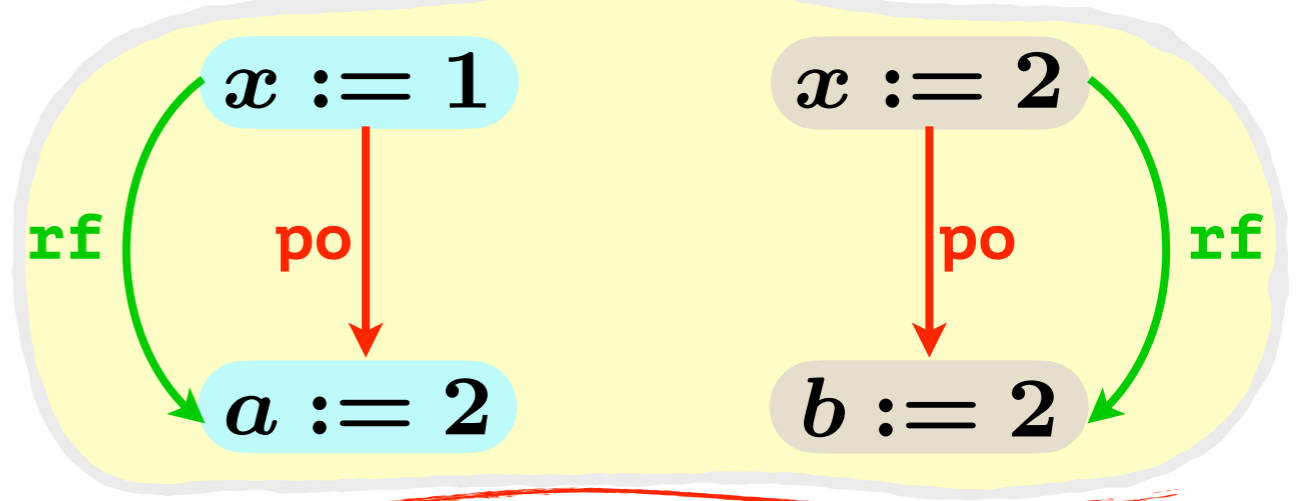


$x := 1$

$a := 1$

$x := 2$

$b := 2$



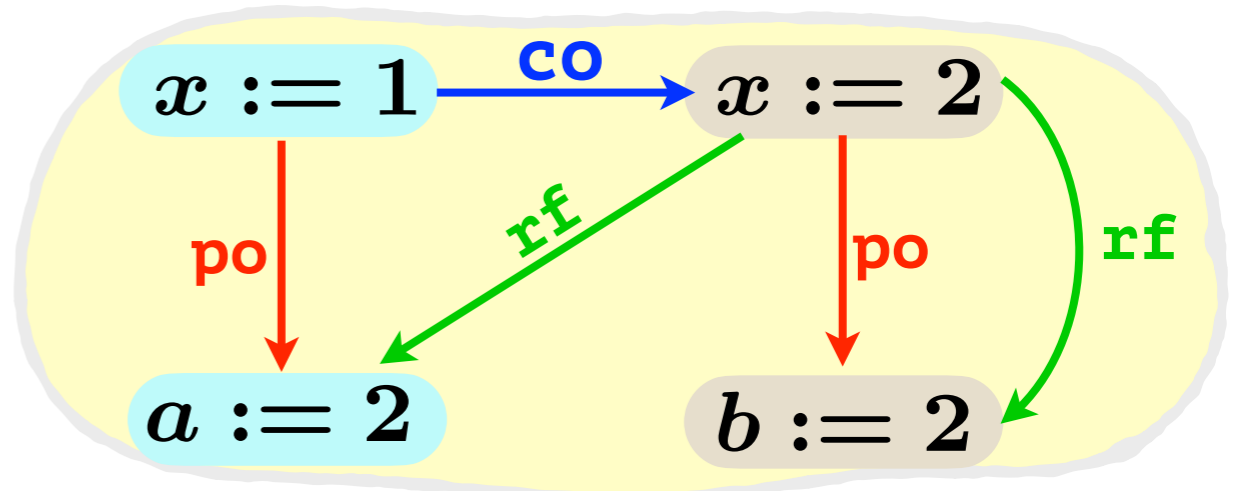
(super-optimal)
DPOR

$x := 1$

$x := 2$

$a := 2$

$b := 2$

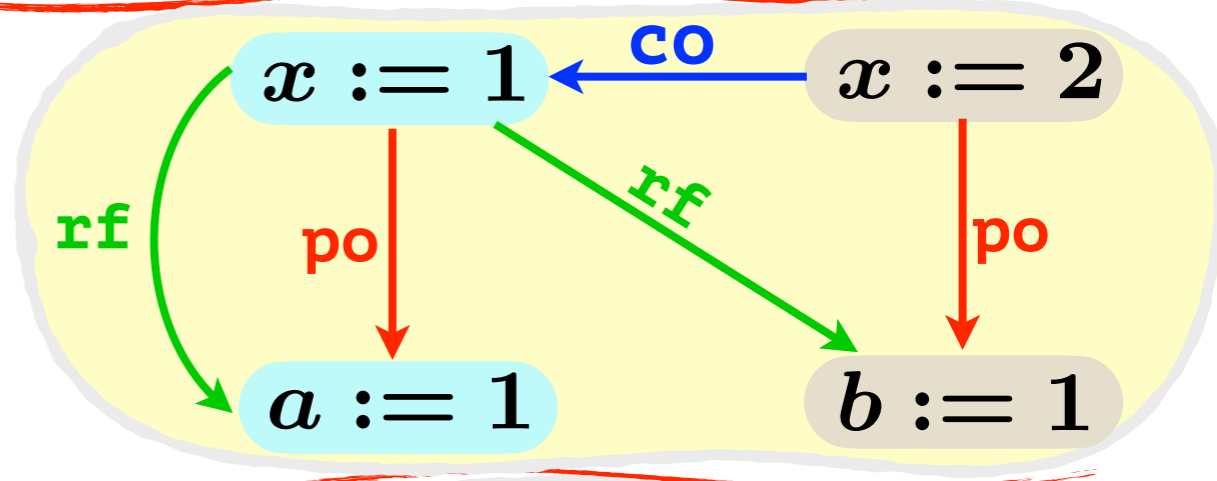


$x := 2$

$x := 1$

$b := 1$

$a := 1$

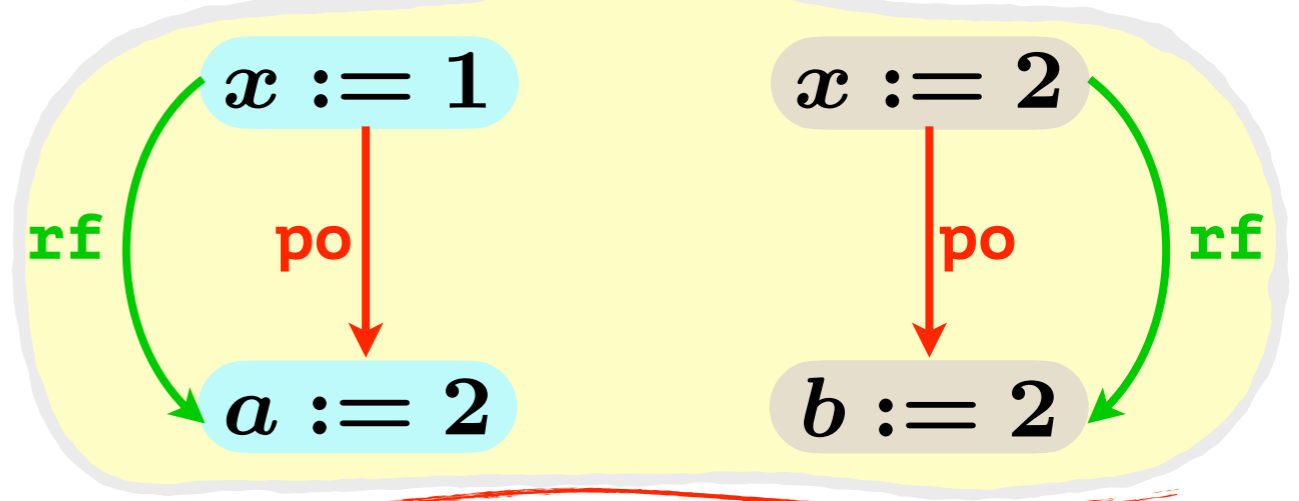


$x := 1$

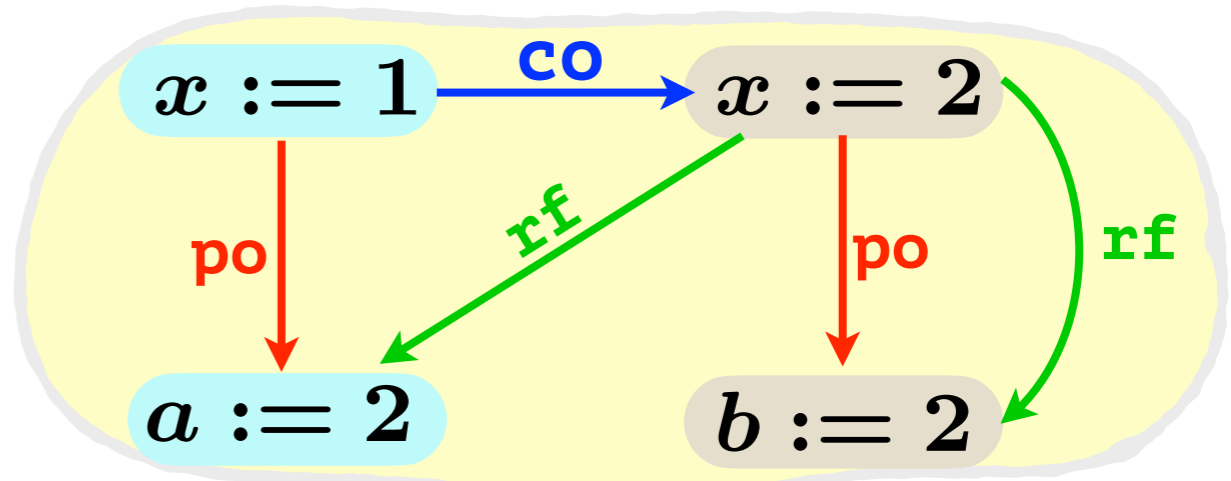
$a := 1$

$x := 2$

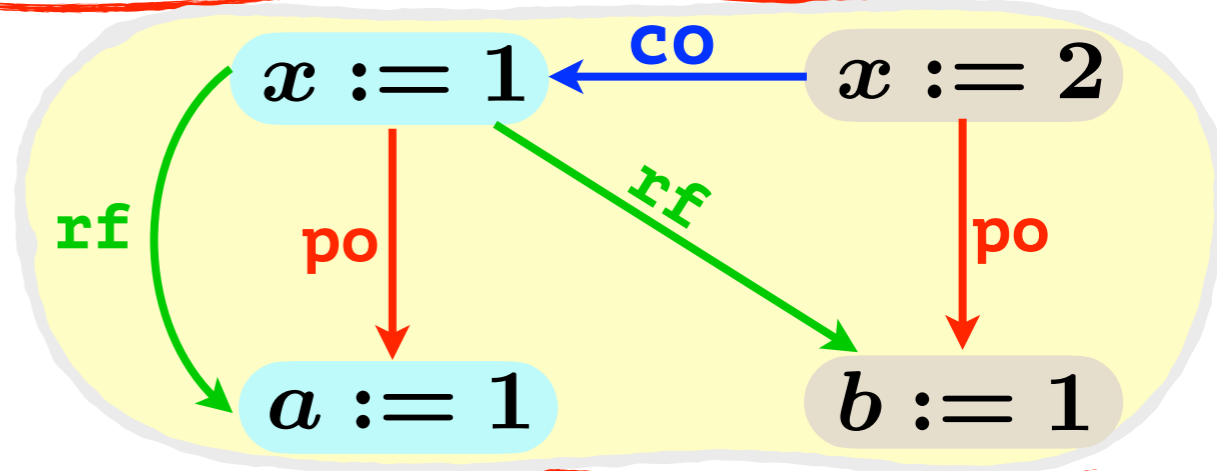
$b := 2$



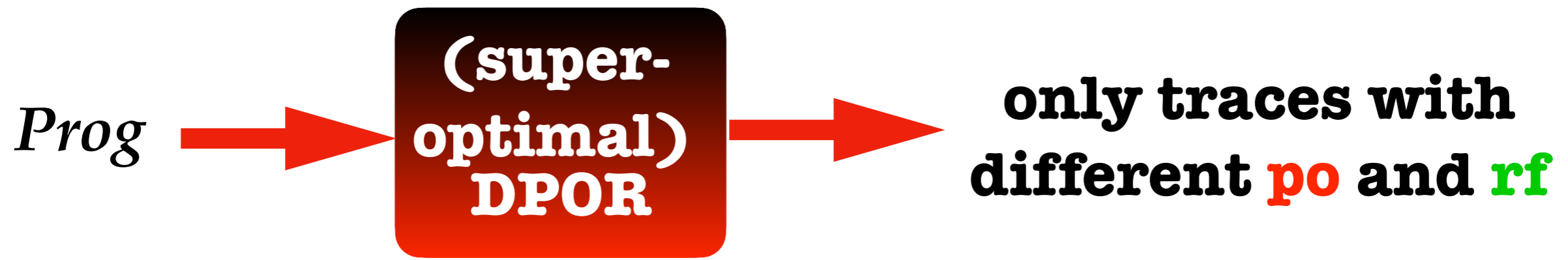
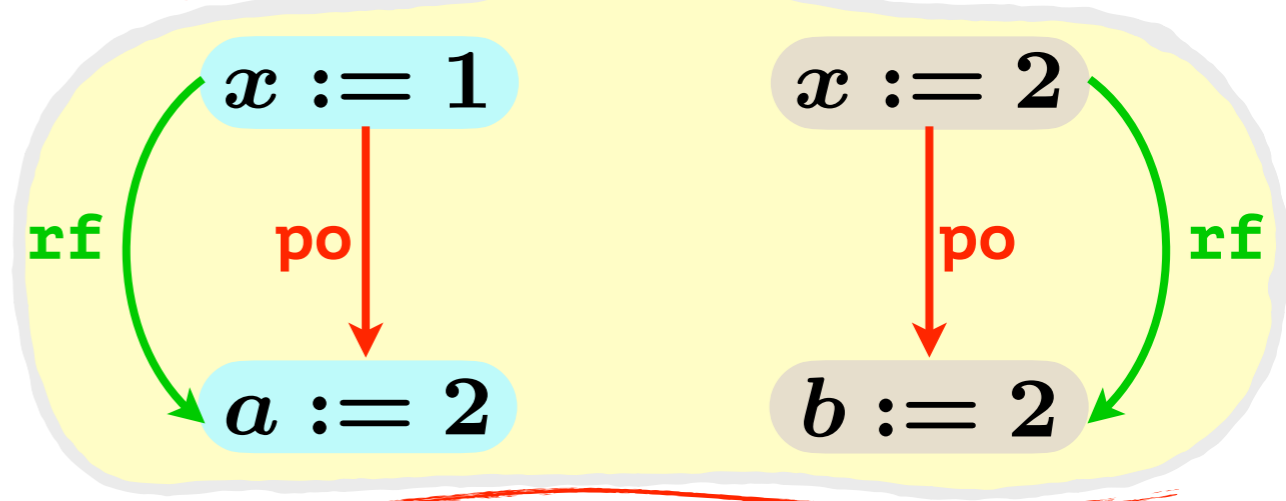
$x := 1$
 $x := 2$
 $a := 2$
 $b := 2$



$x := 2$
 $x := 1$
 $b := 1$
 $a := 1$



$x := 1$
 $a := 1$
 $x := 2$
 $b := 2$



Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- **Conclusions**

“first let us consider SC”

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- **Conclusions**

“now let us consider RA”

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- **Conclusions**

“now let us consider RA”

axiomatic
semantics

Outline

- **Memory Models**
 - SC (Sequential Consistency)
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - Experiments
- **Conclusions**

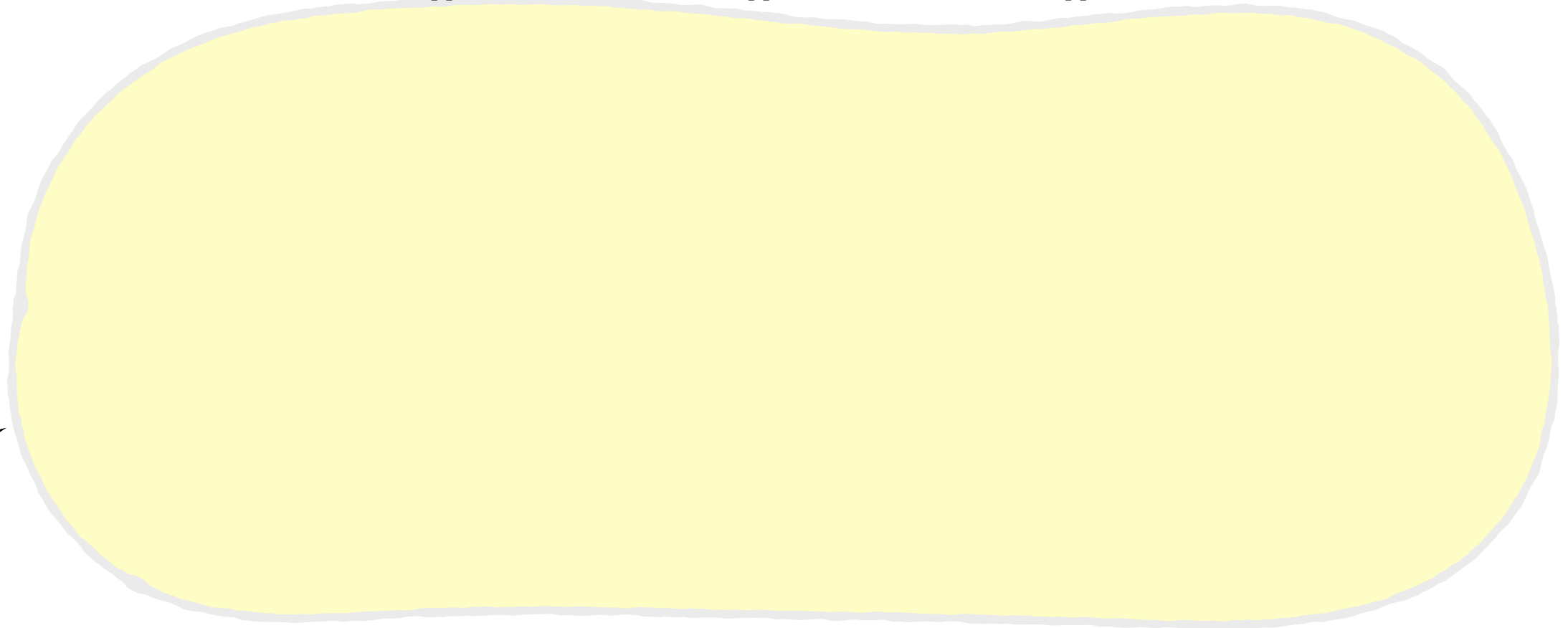
“now let us consider RA”

Traces

axiomatic
semantics

Release-Acquire (RA)

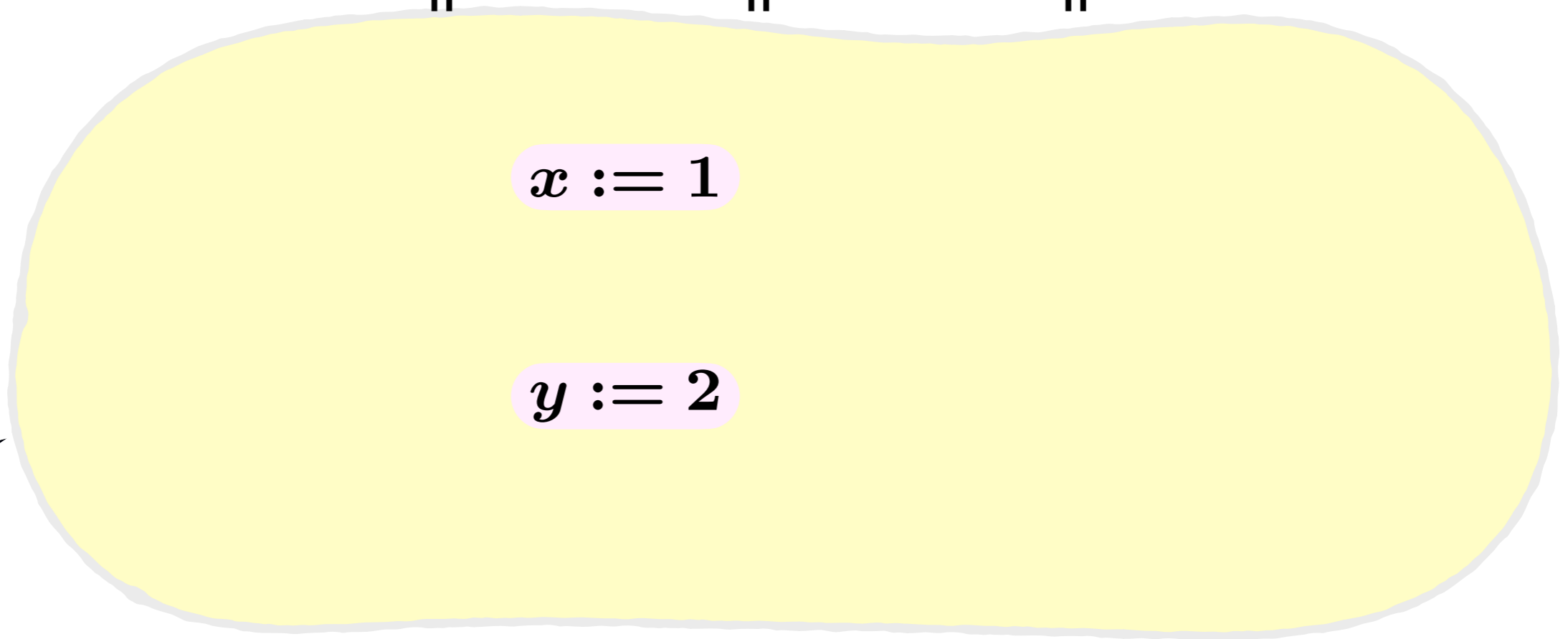
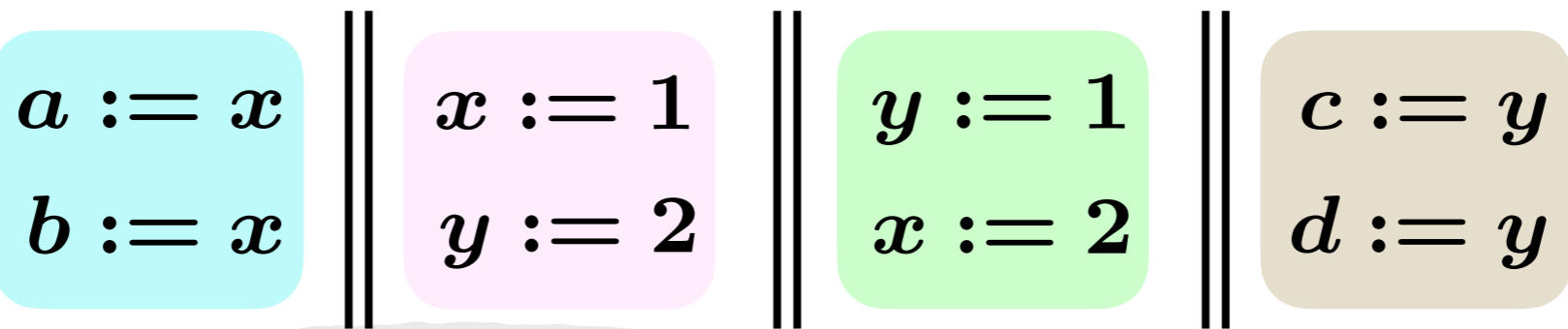
$a := x$		$x := 1$		$y := 1$		$c := y$
$b := x$		$y := 2$		$x := 2$		$d := y$



τ

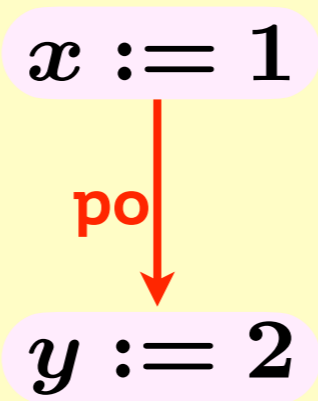
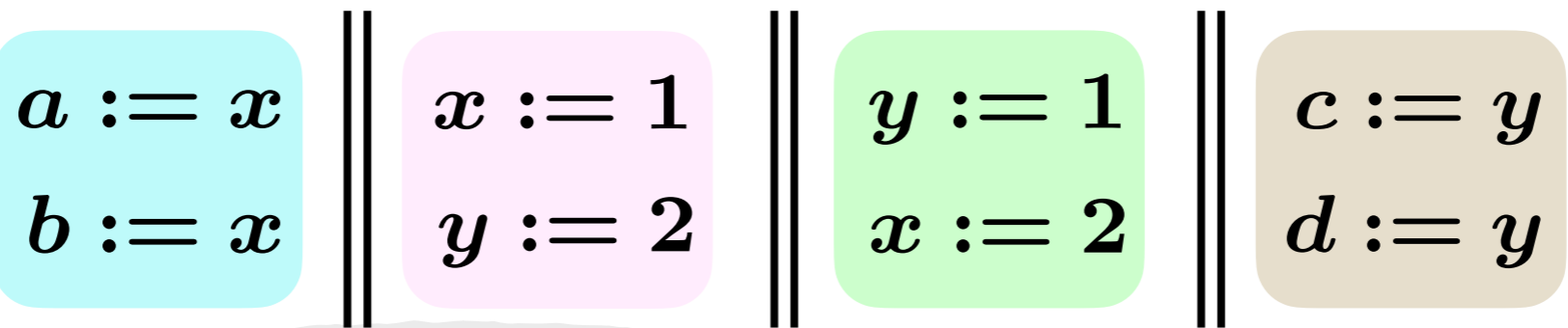
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

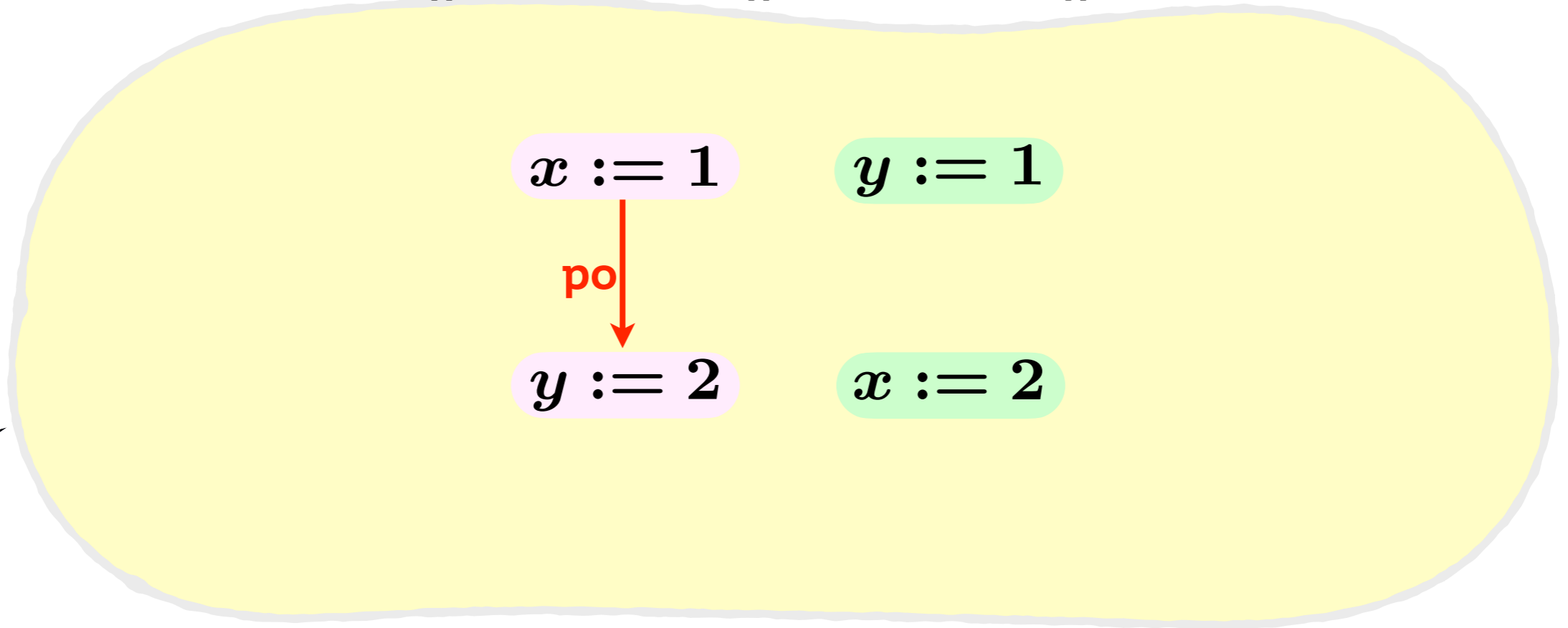
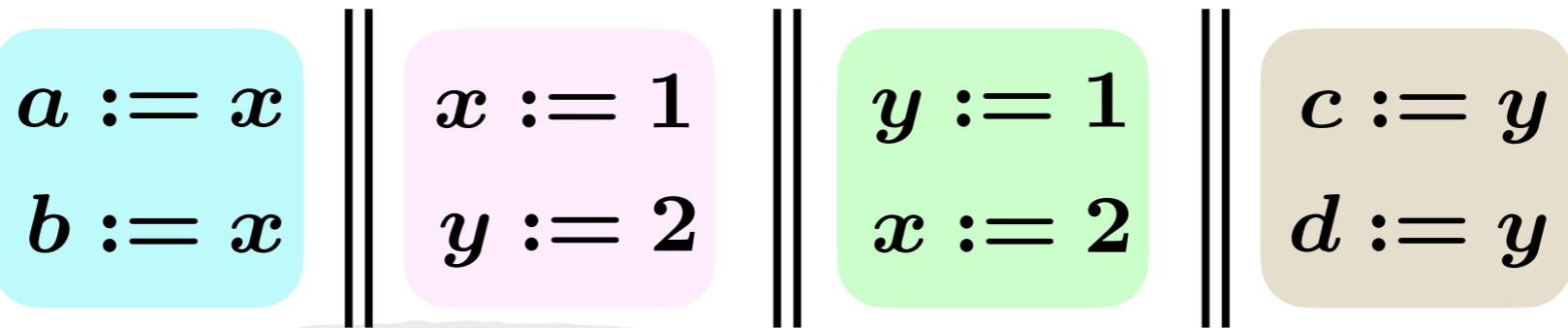
Release-Acquire (RA)



τ

RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

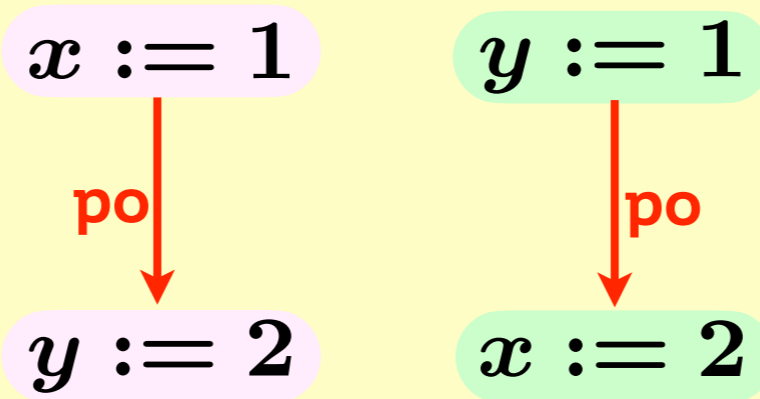
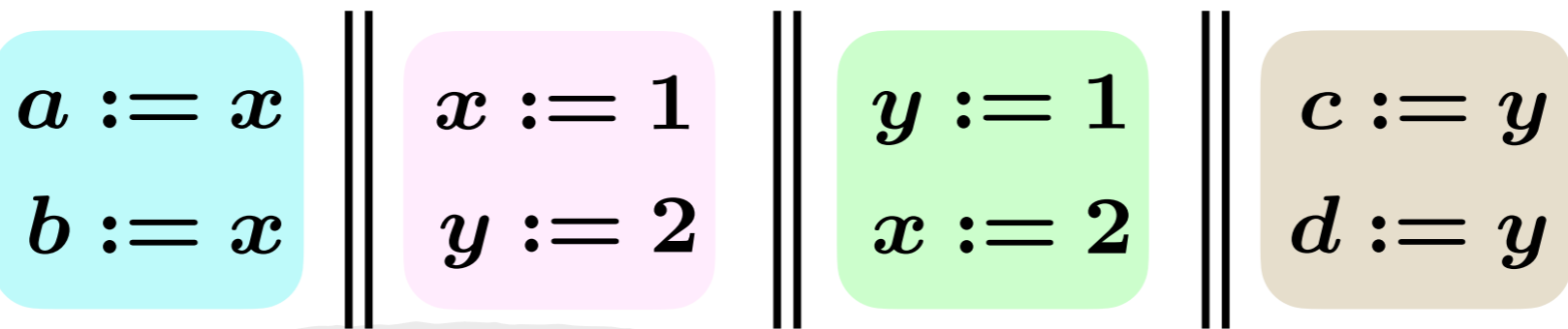
Release-Acquire (RA)



τ

RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

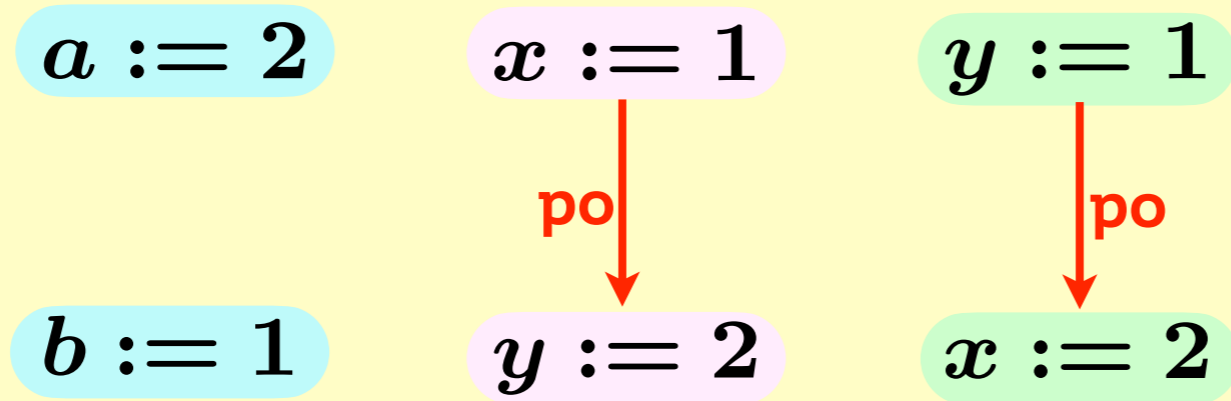
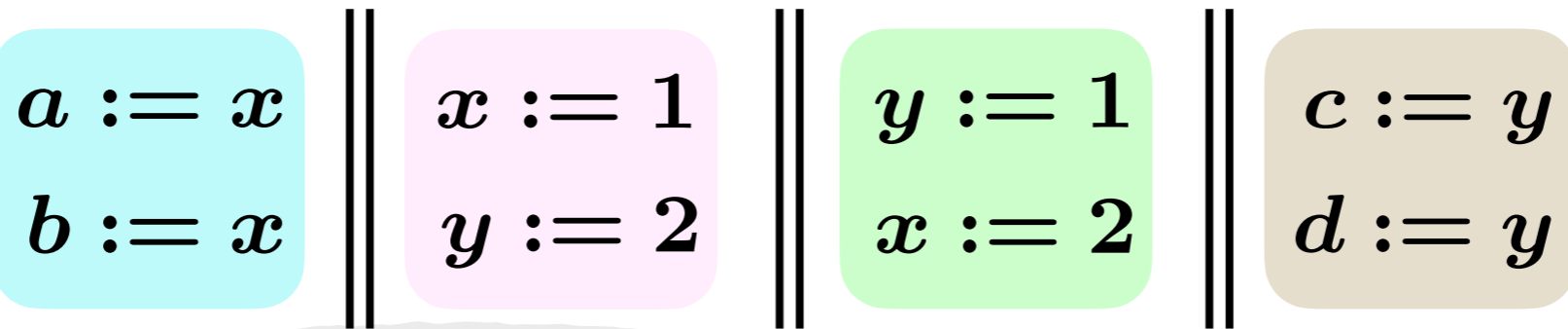
Release-Acquire (RA)



τ

RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

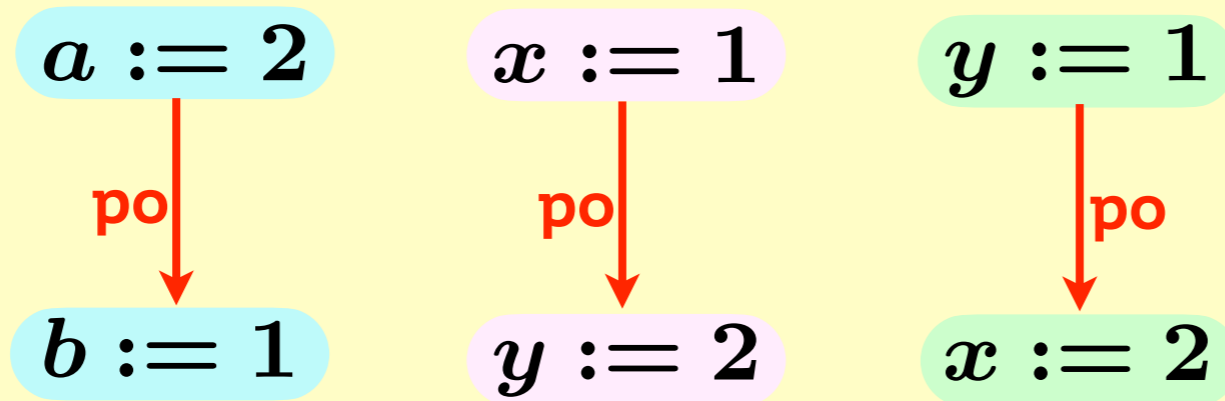
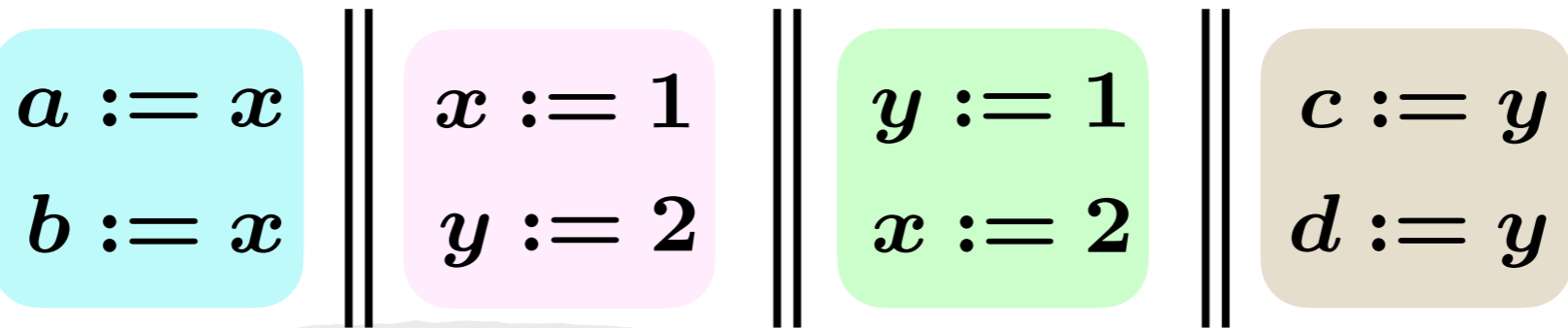
Release-Acquire (RA)



τ

RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

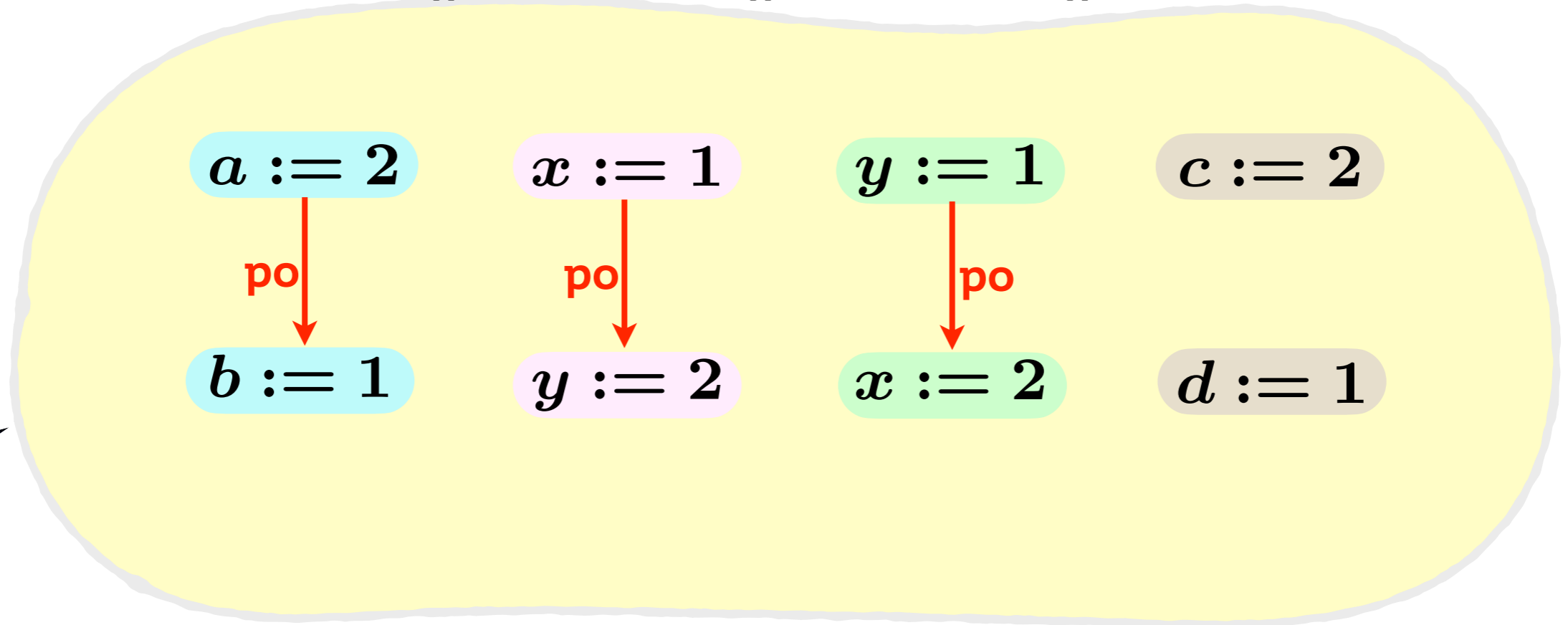
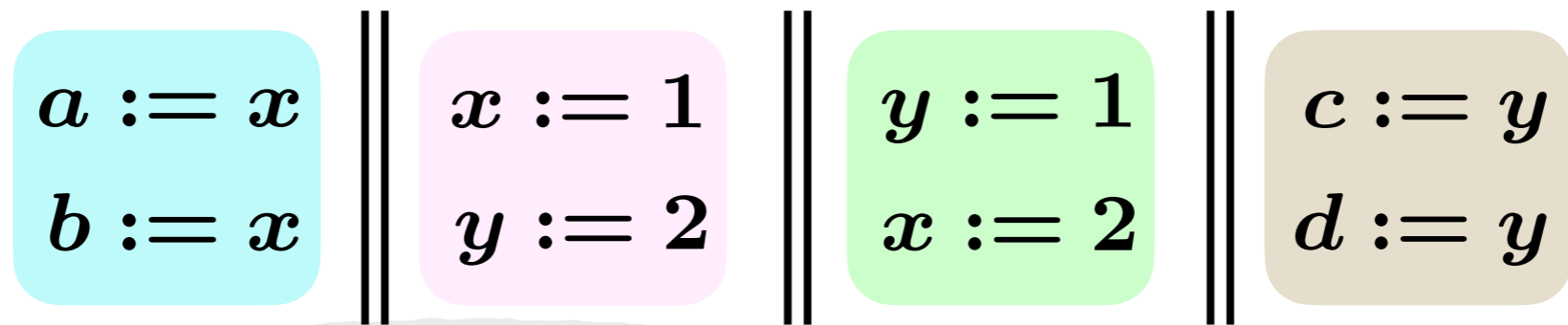
Release-Acquire (RA)



τ

RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

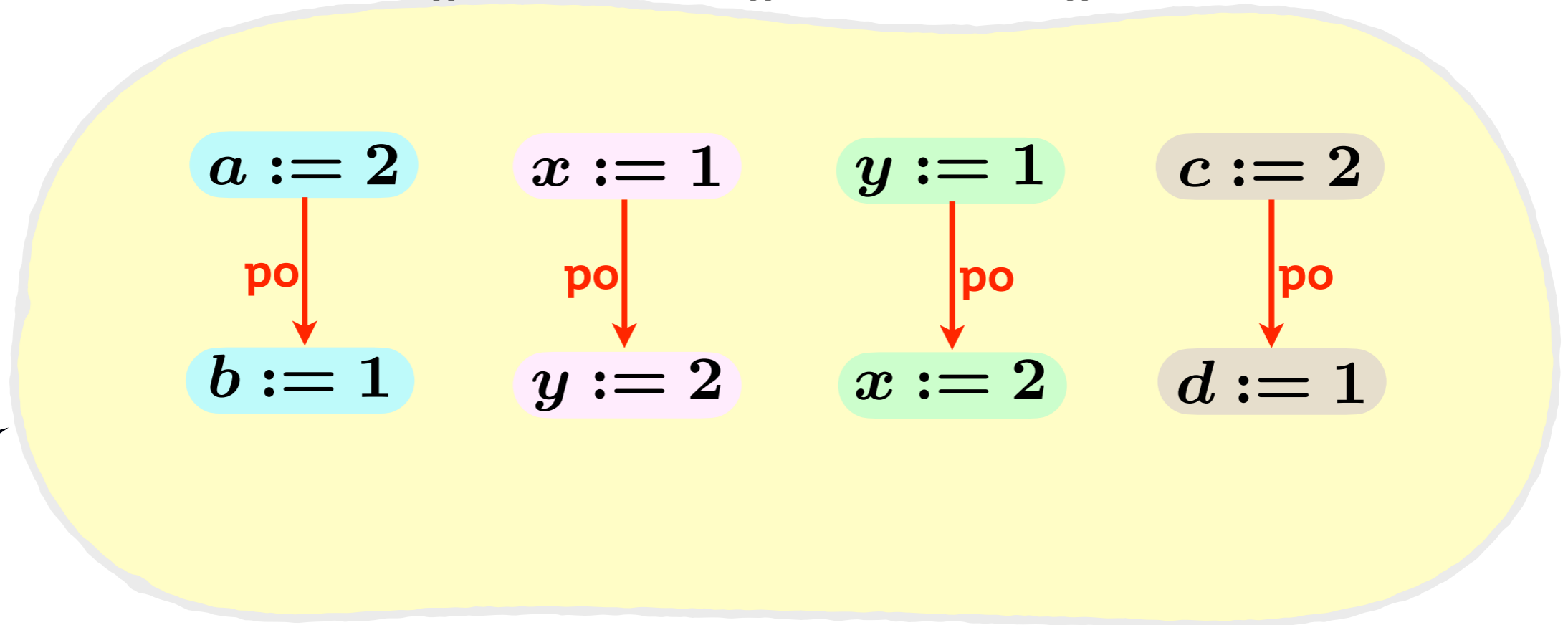
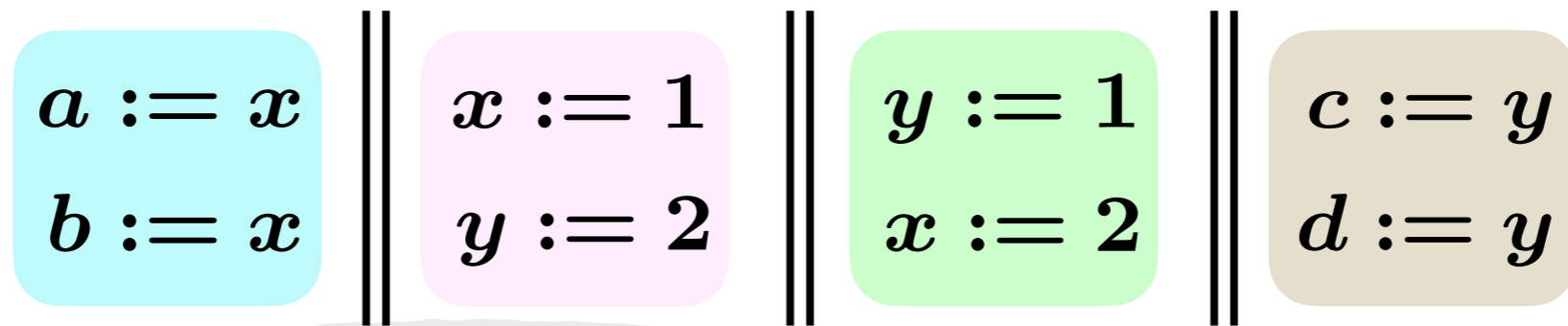
Release-Acquire (RA)



τ

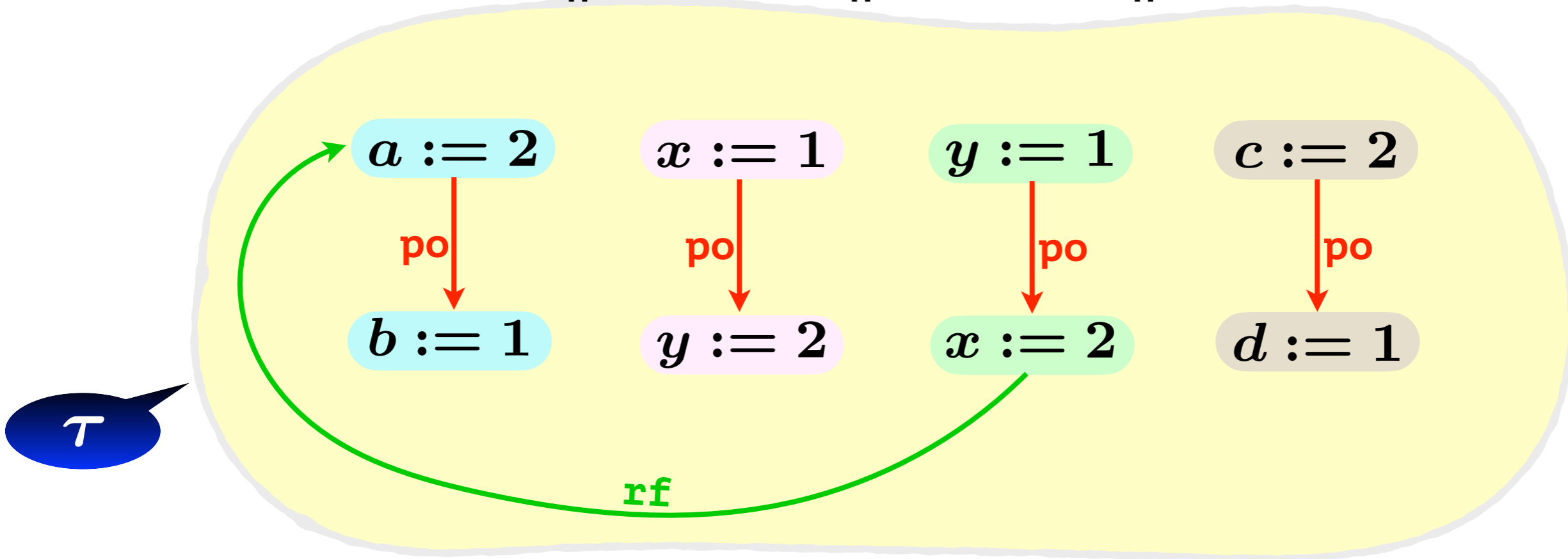
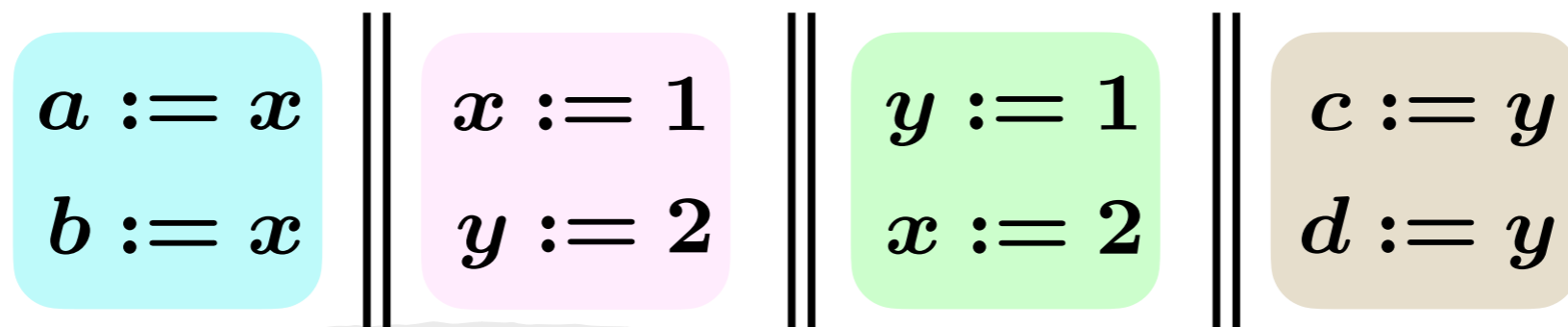
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



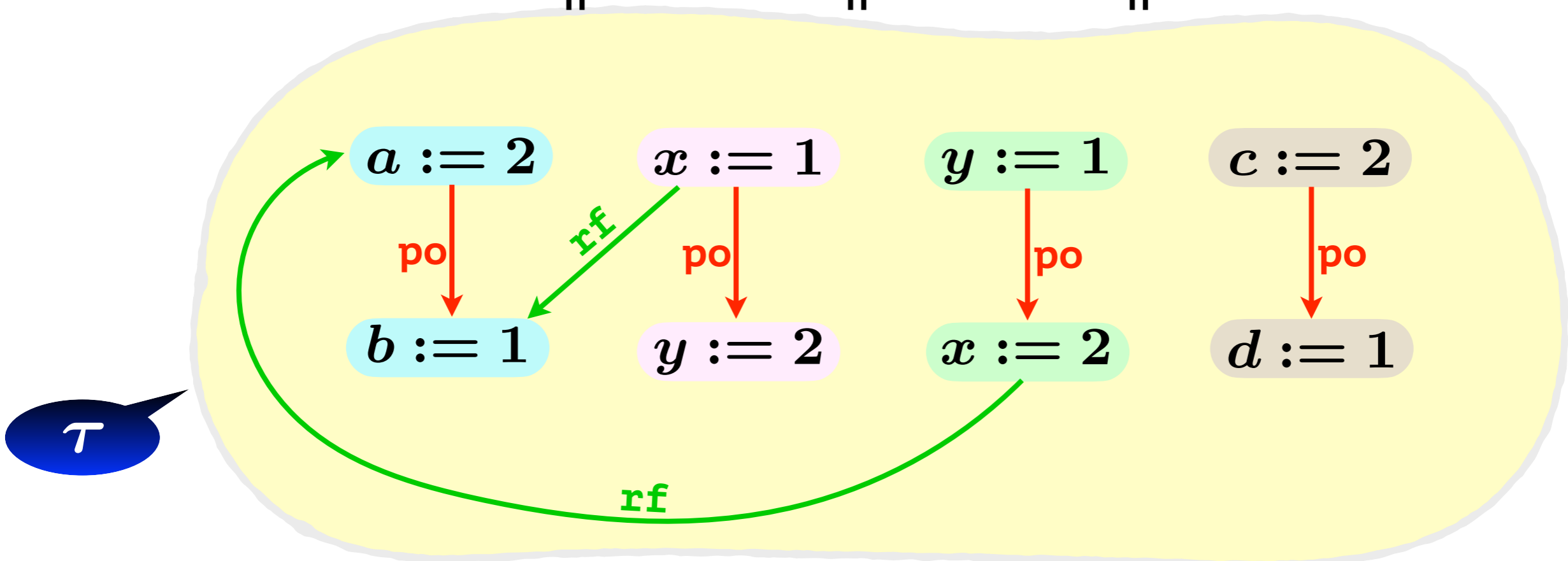
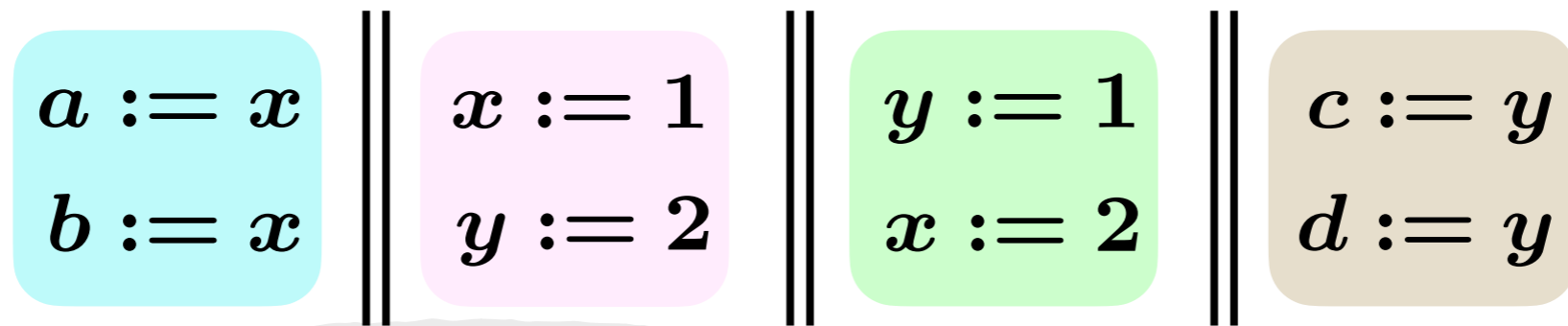
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



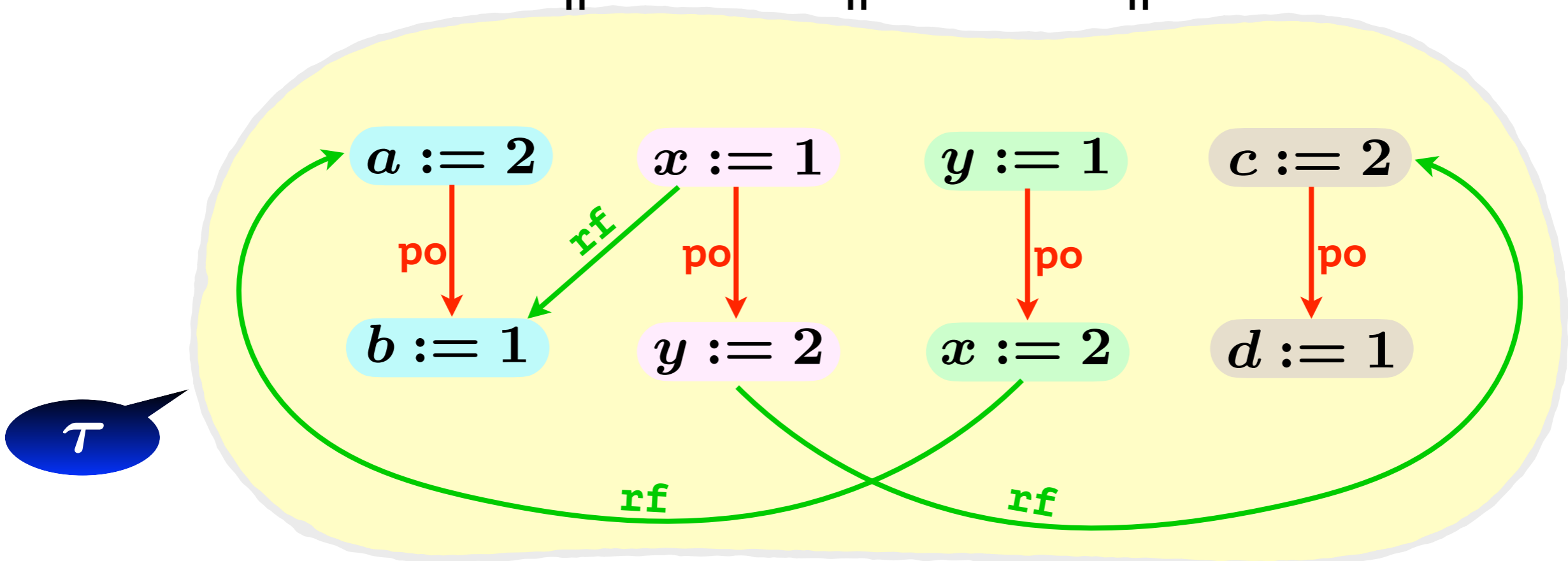
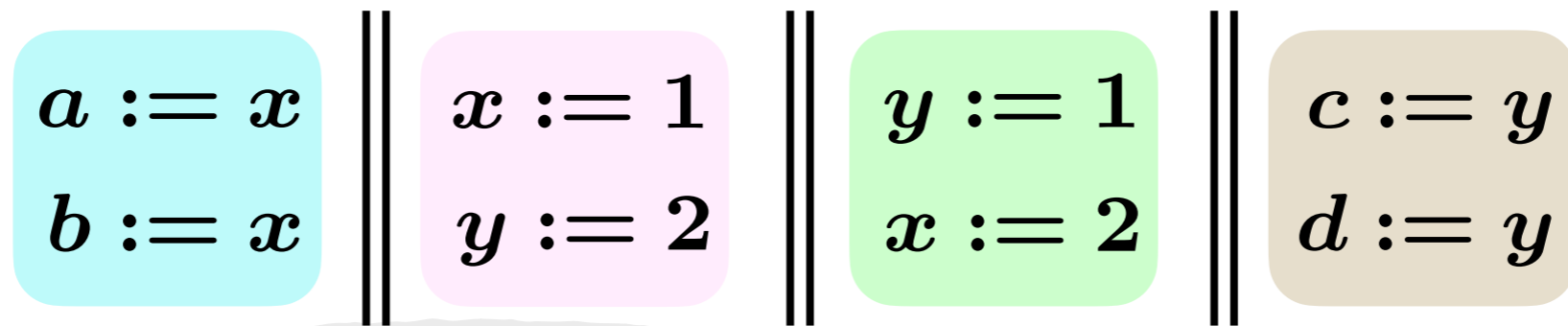
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



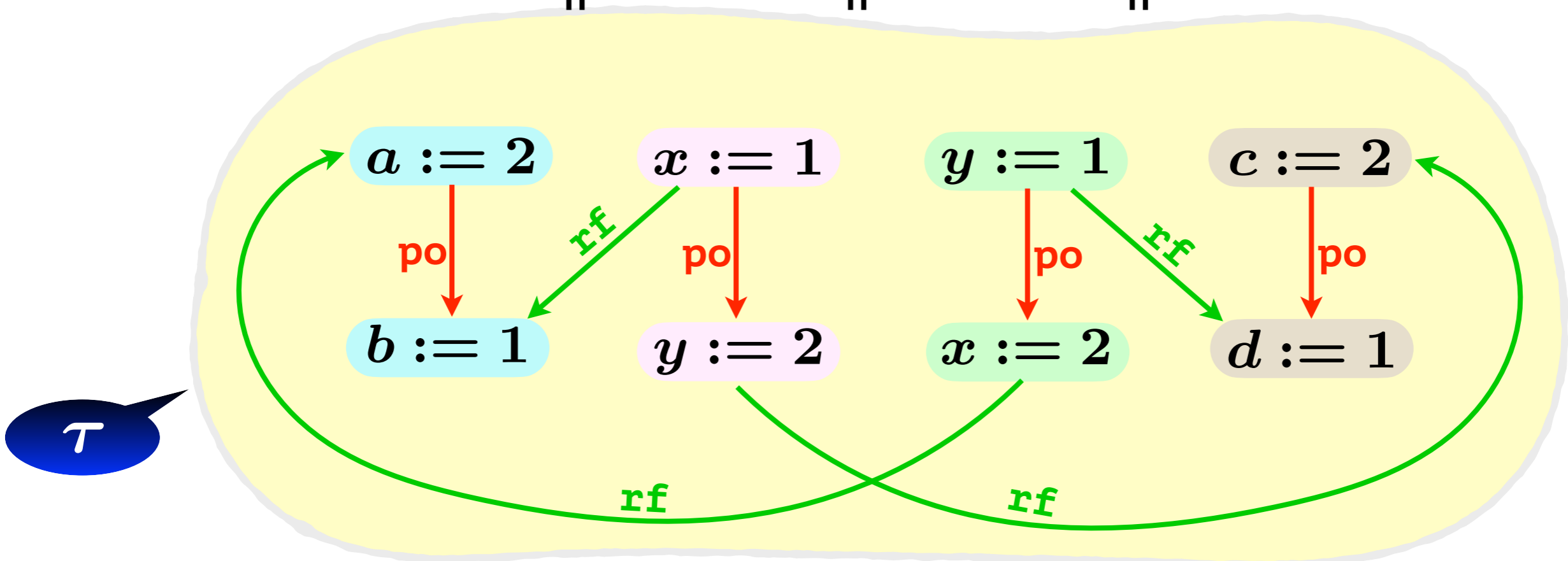
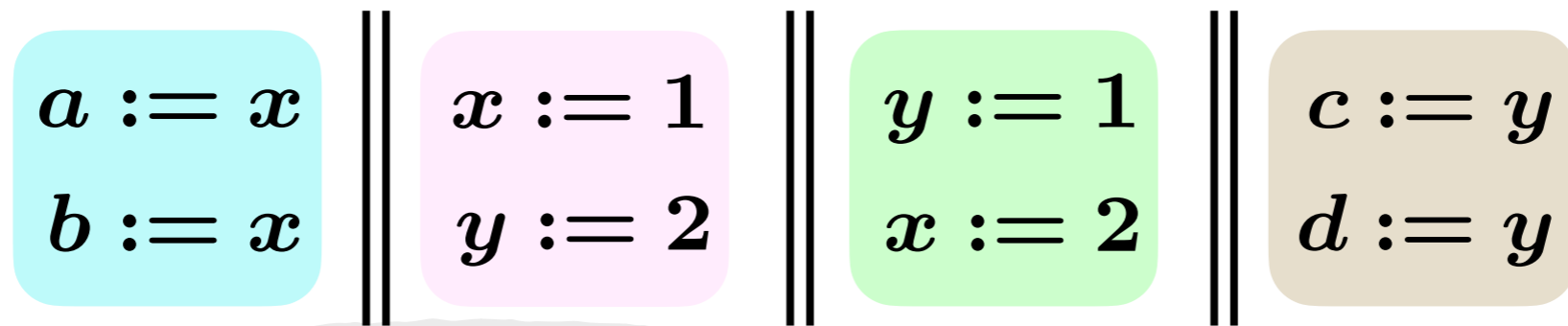
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



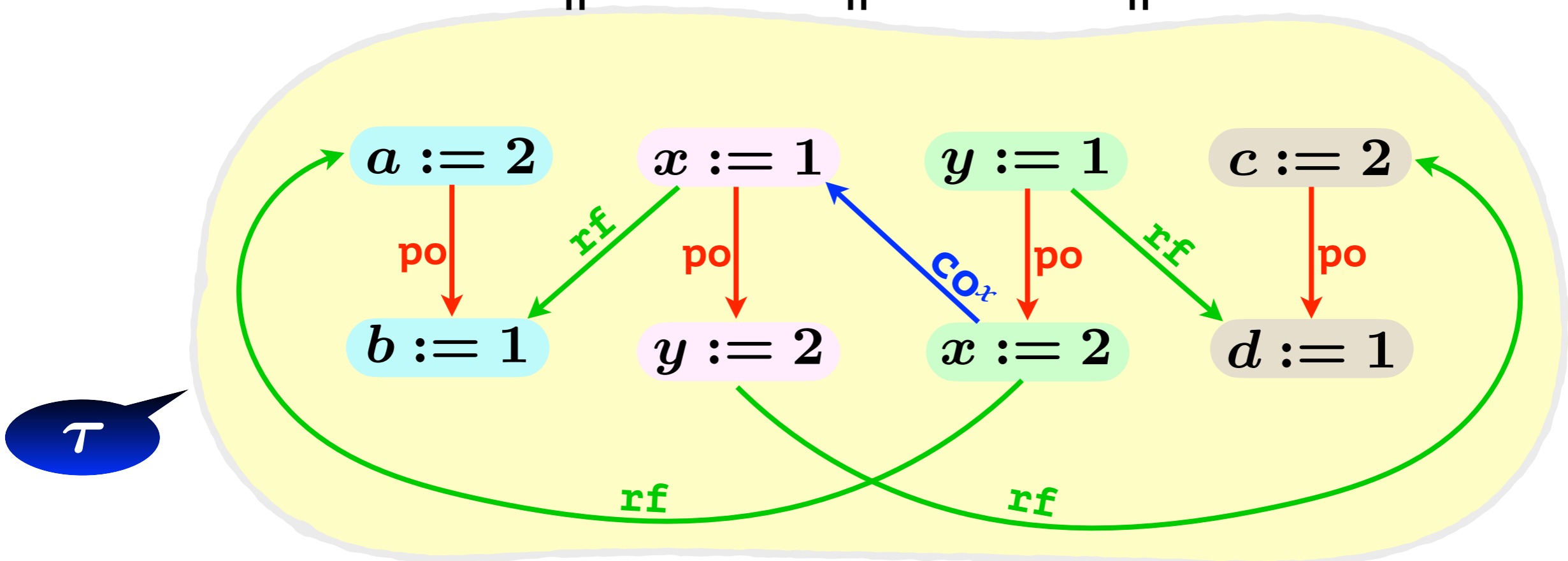
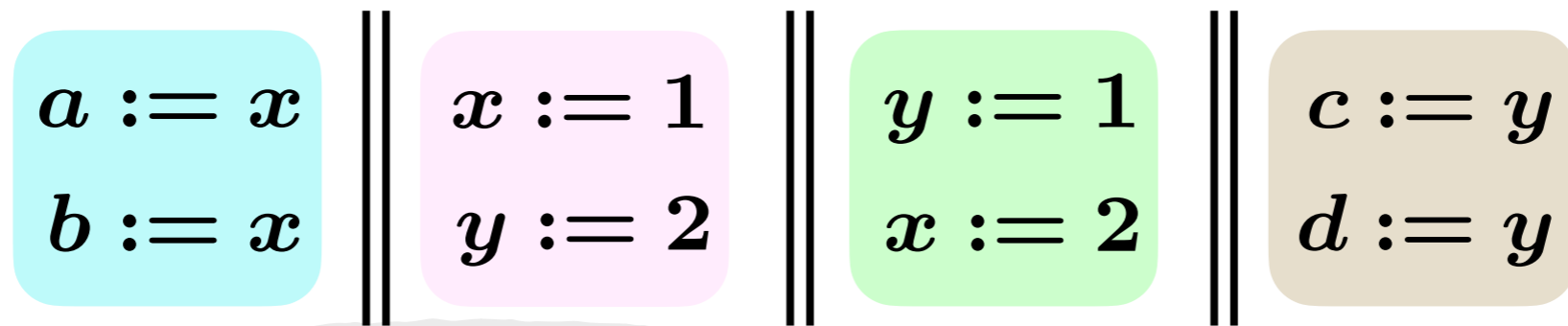
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



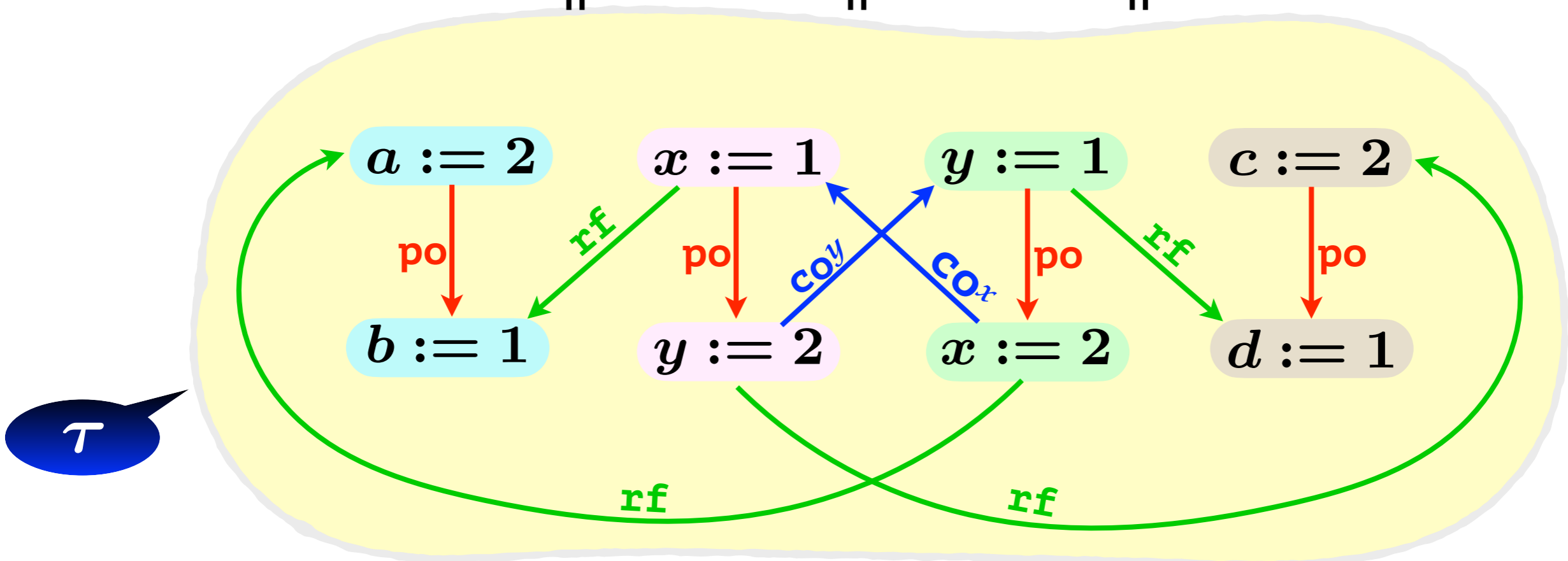
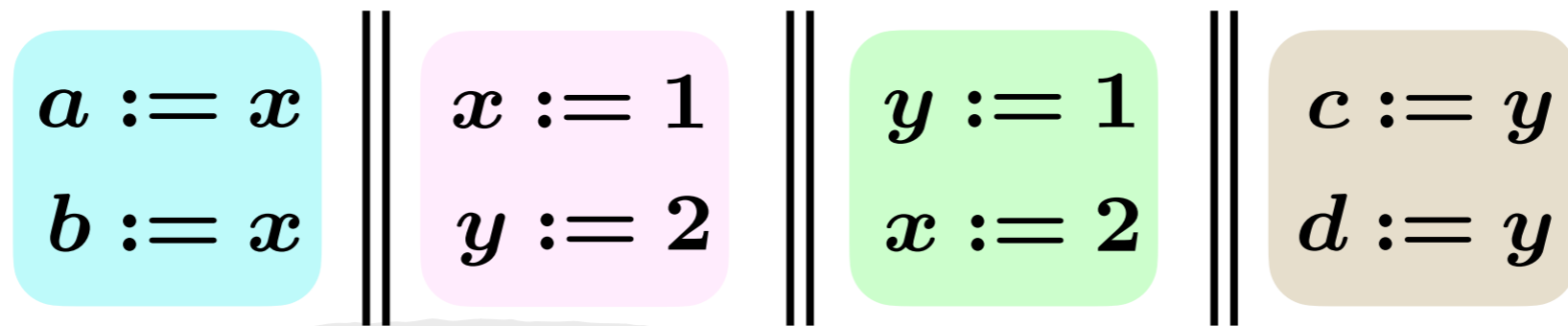
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



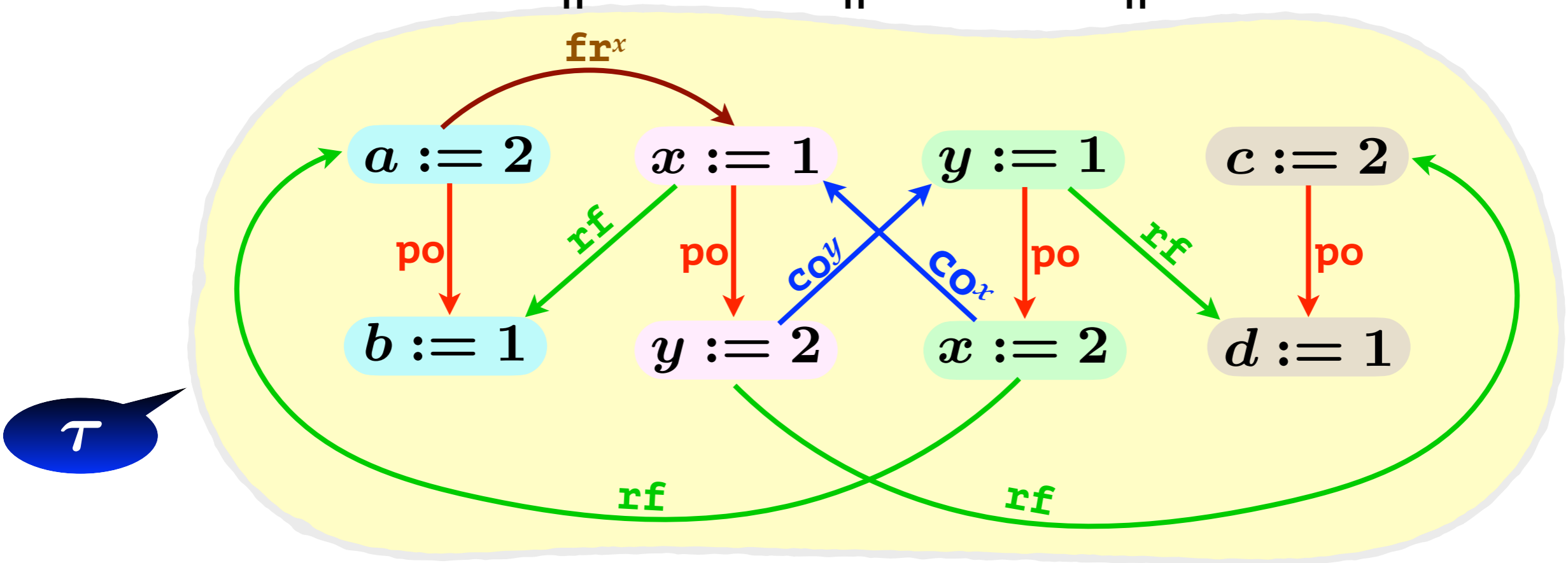
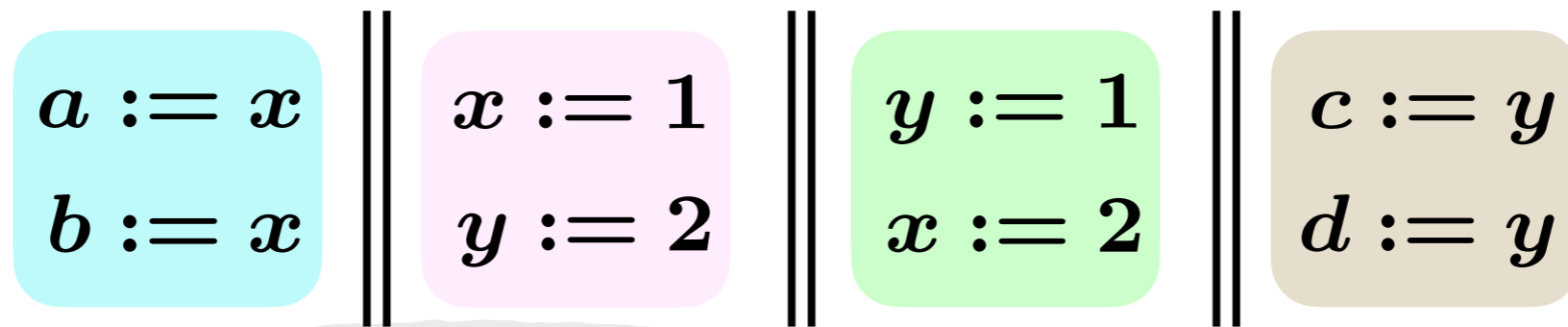
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{cox} \cup \text{fr}^x)$

Release-Acquire (RA)



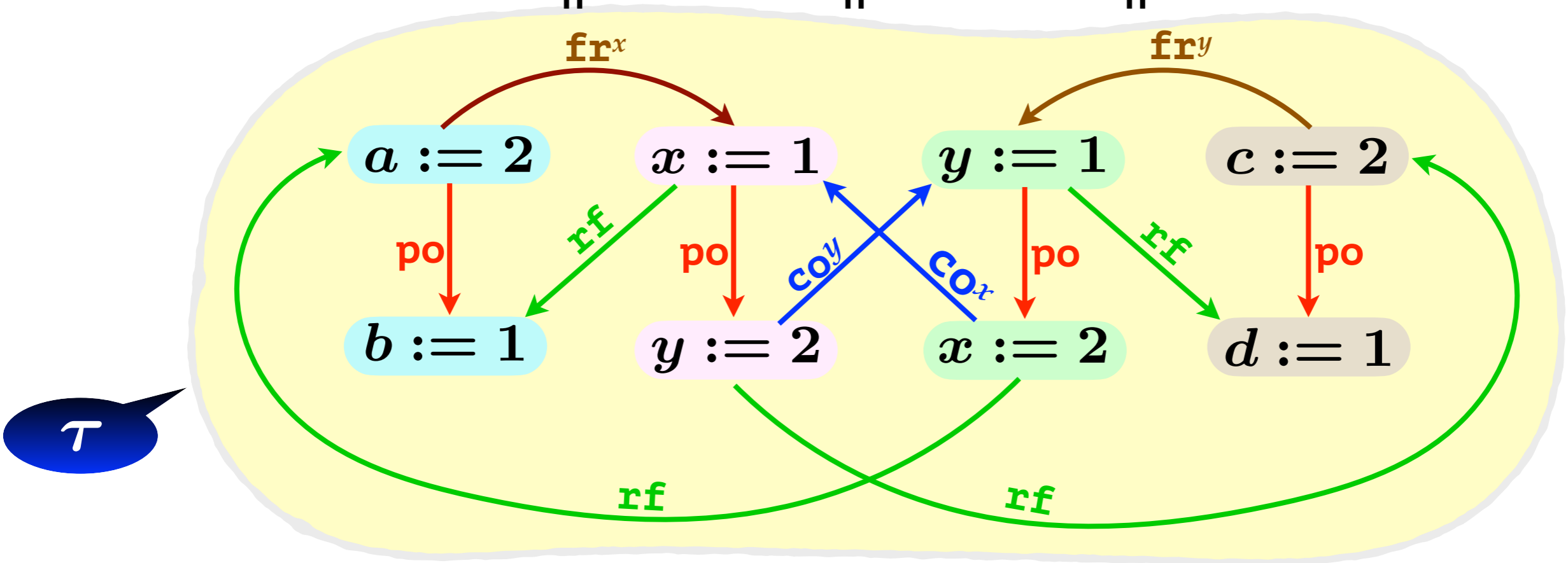
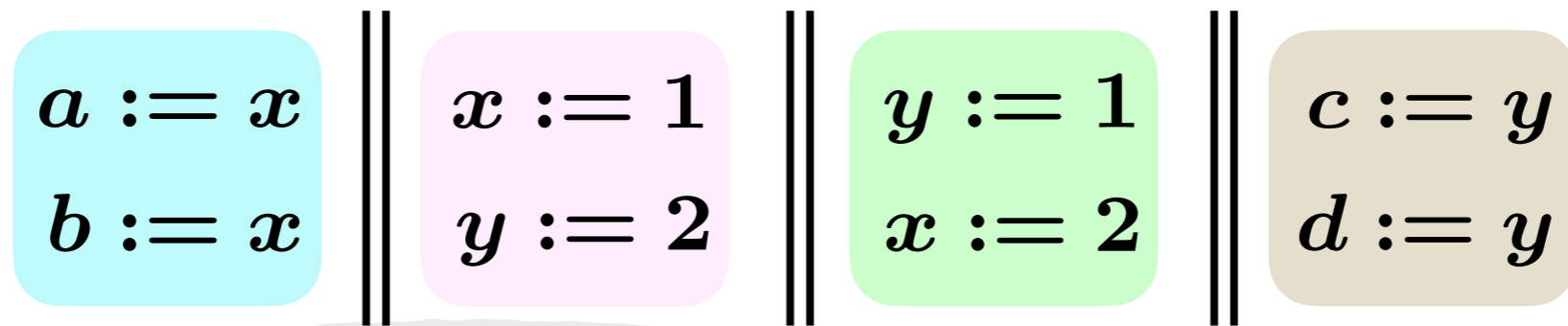
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



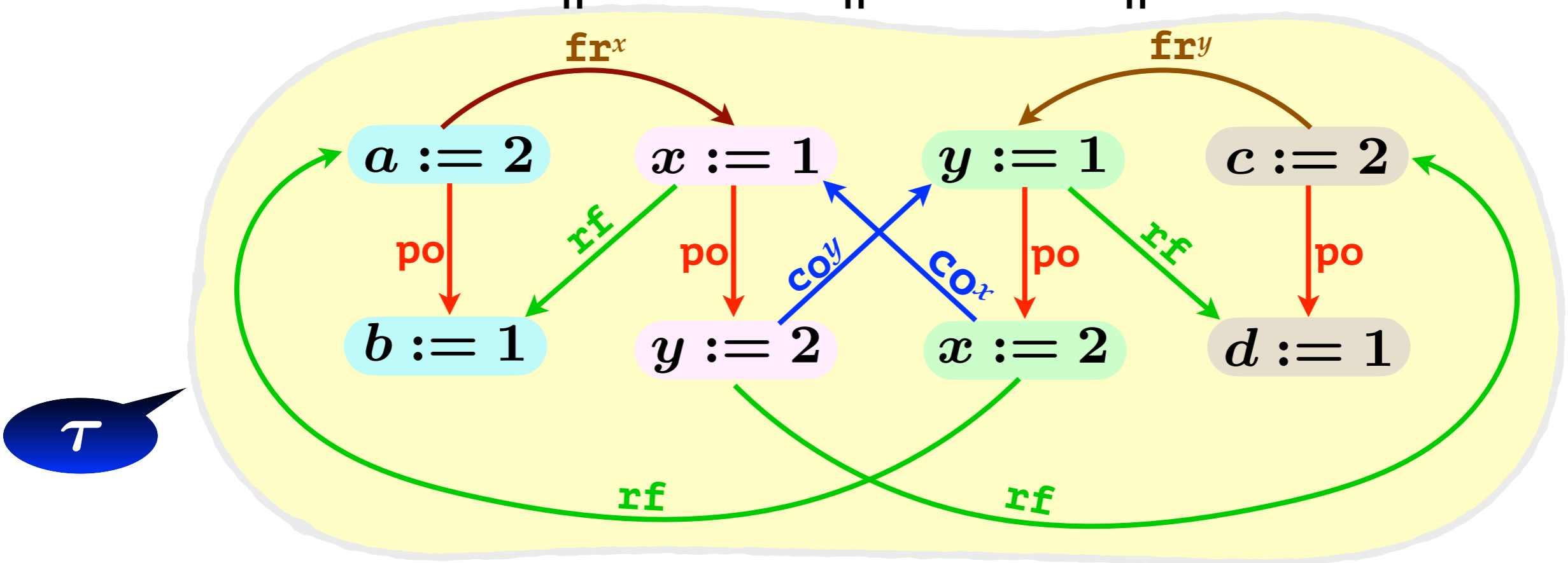
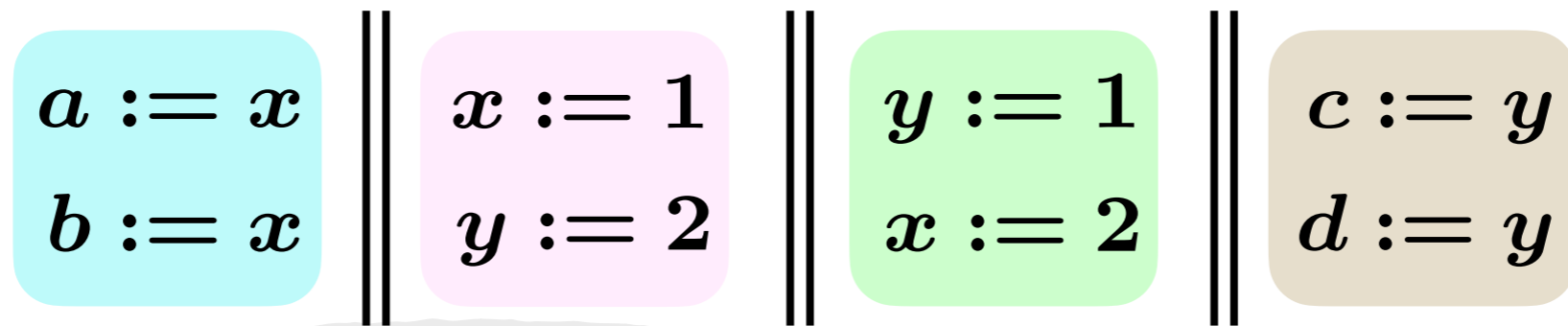
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



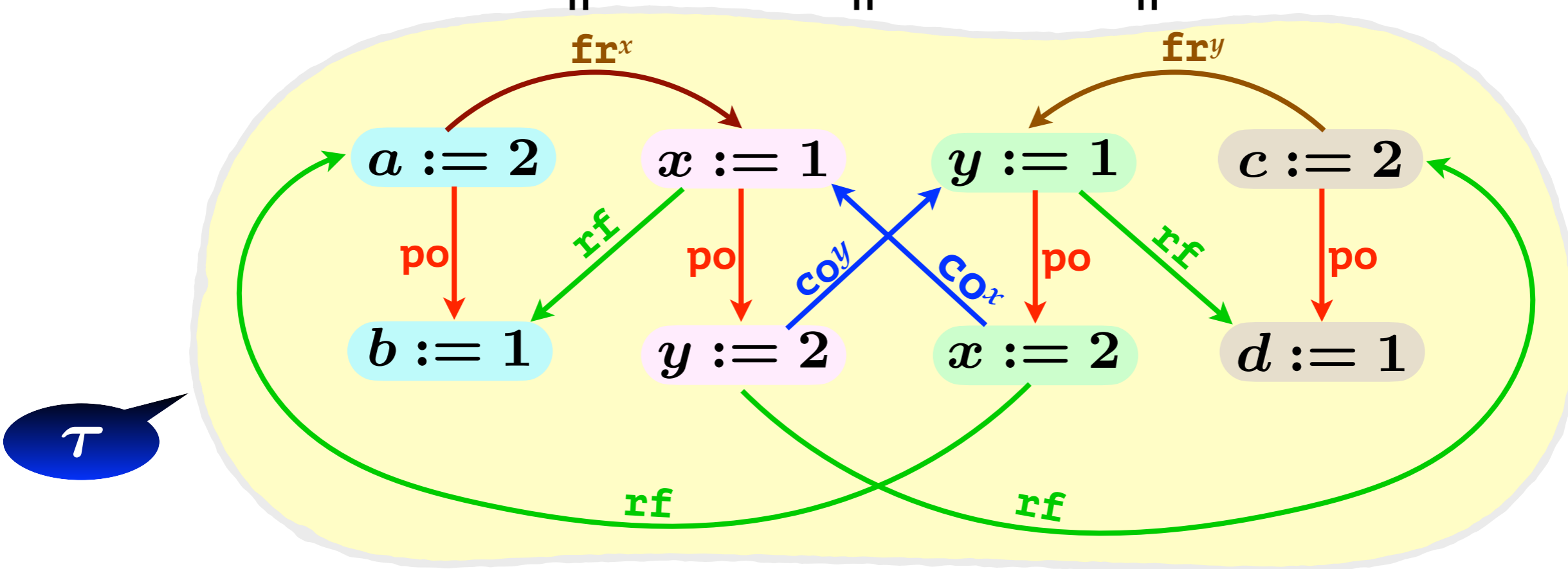
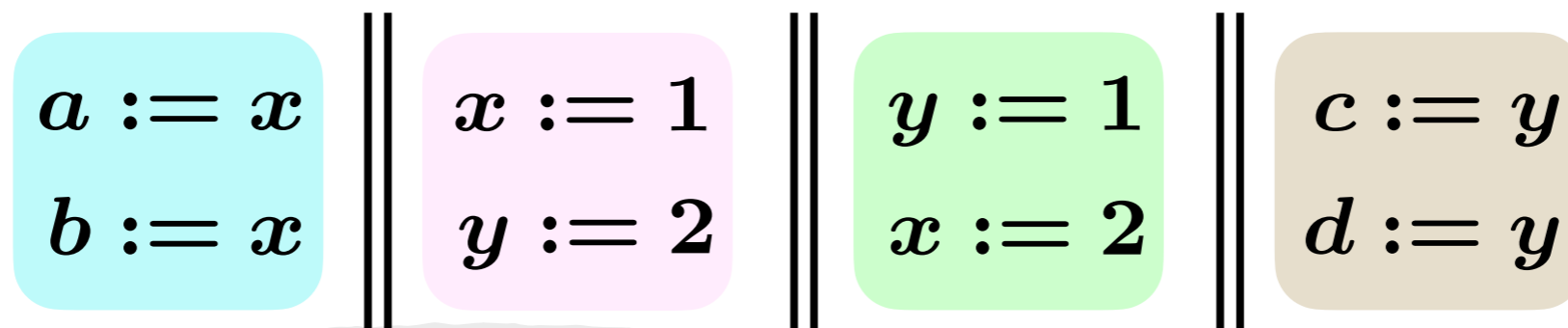
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Release-Acquire (RA)



RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$ $\tau \models \text{RA}$

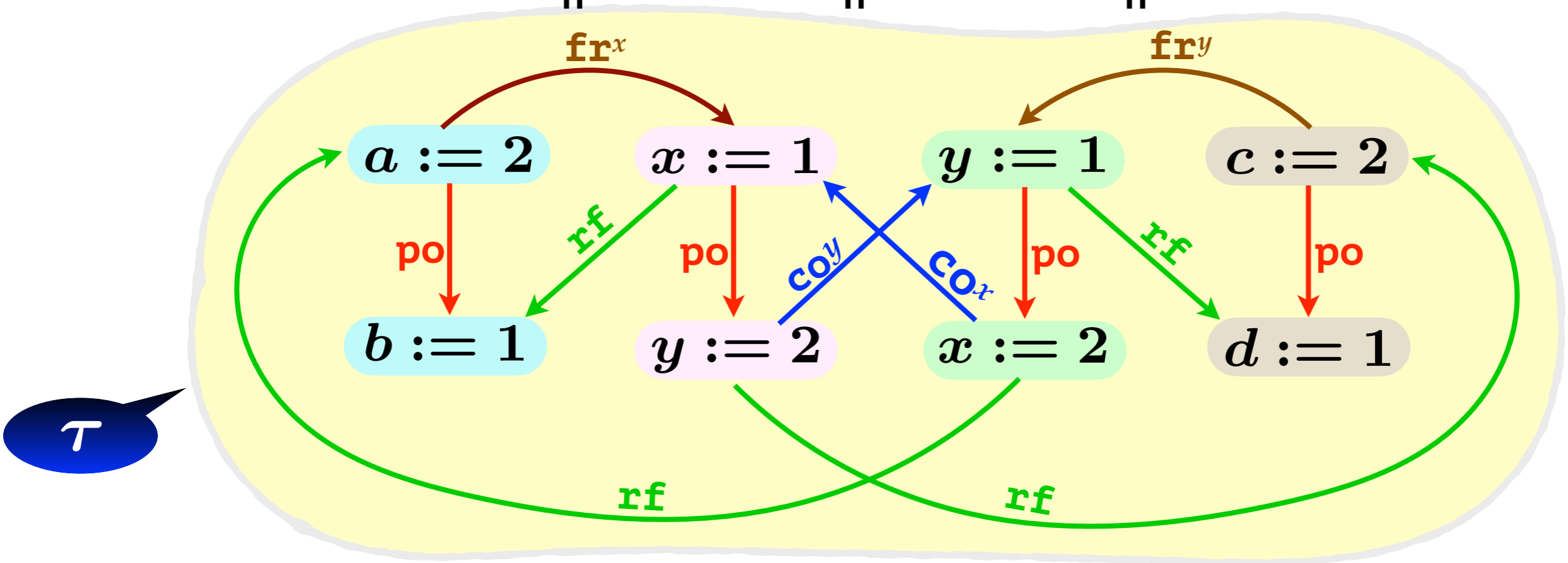
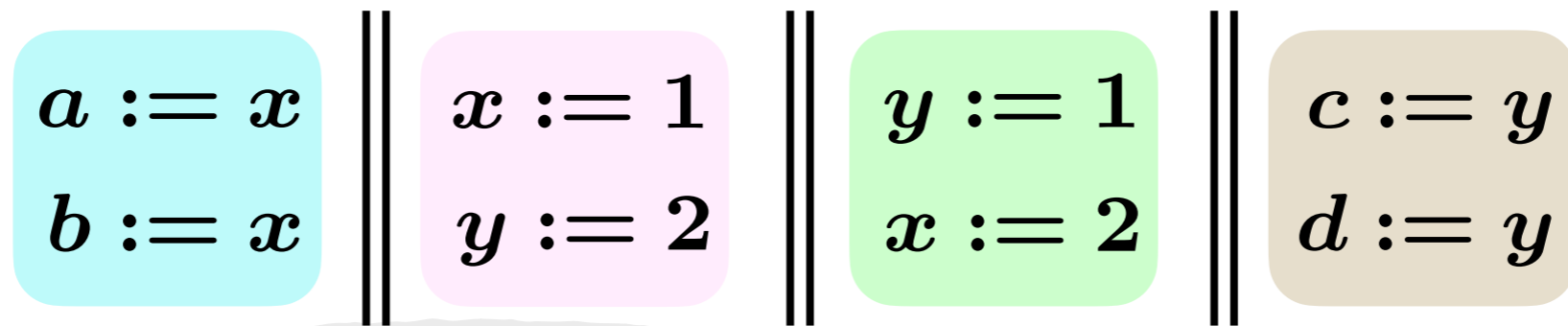
Release-Acquire (RA)



RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$ $\tau \models \text{RA}$

SC: $\text{acyclic}(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

Release-Acquire (RA)



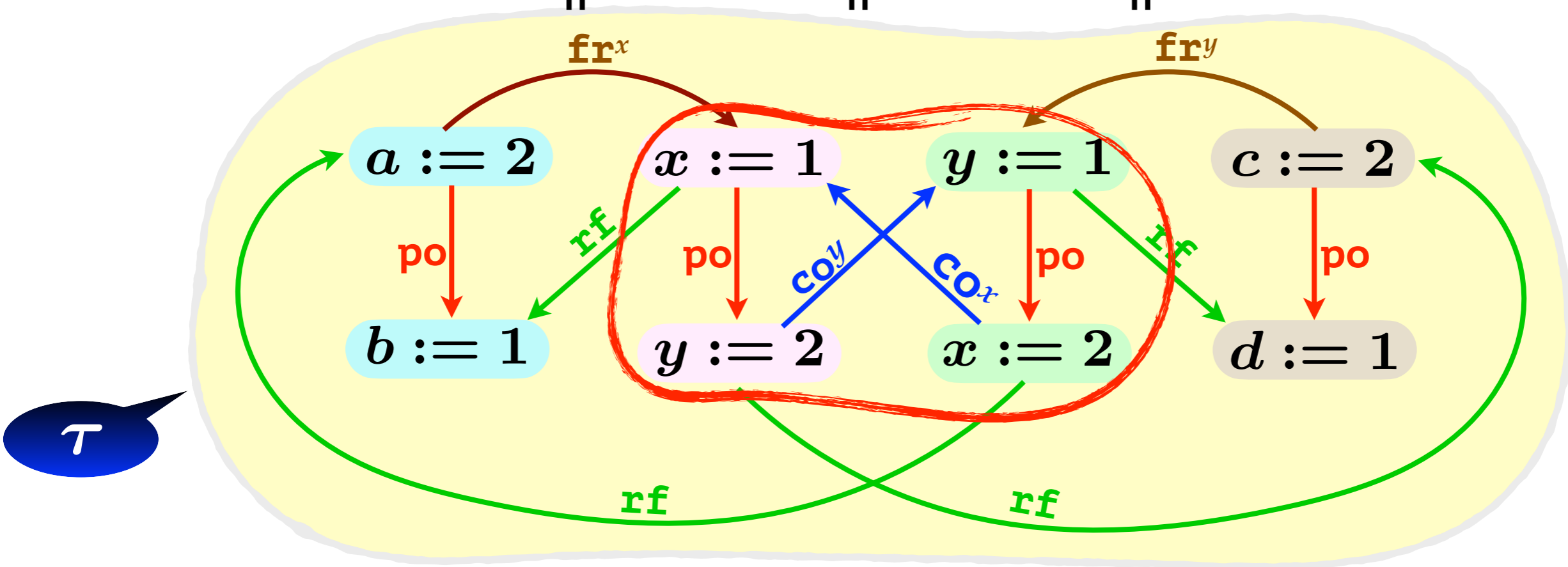
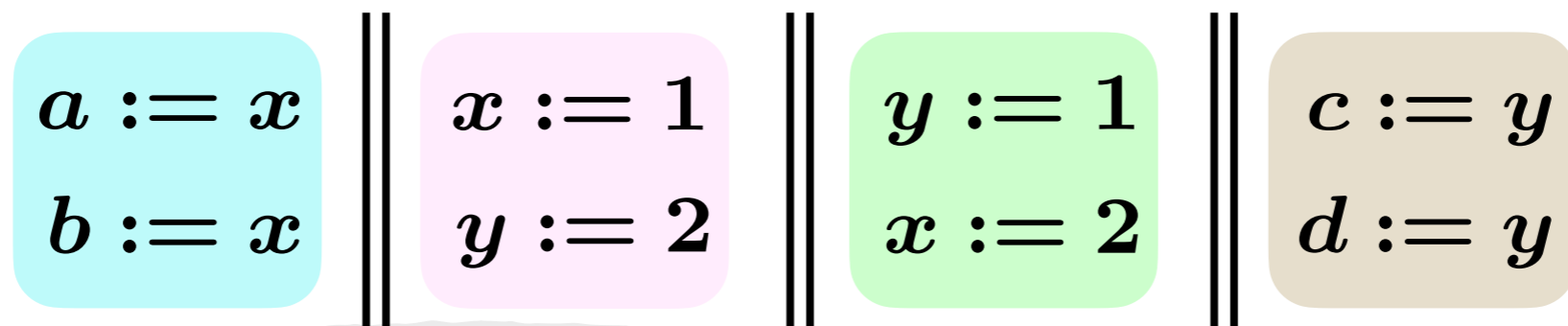
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

$\tau \models \text{RA}$

SC: $\text{acyclic}(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

$\tau \not\models \text{SC}$

Release-Acquire (RA)



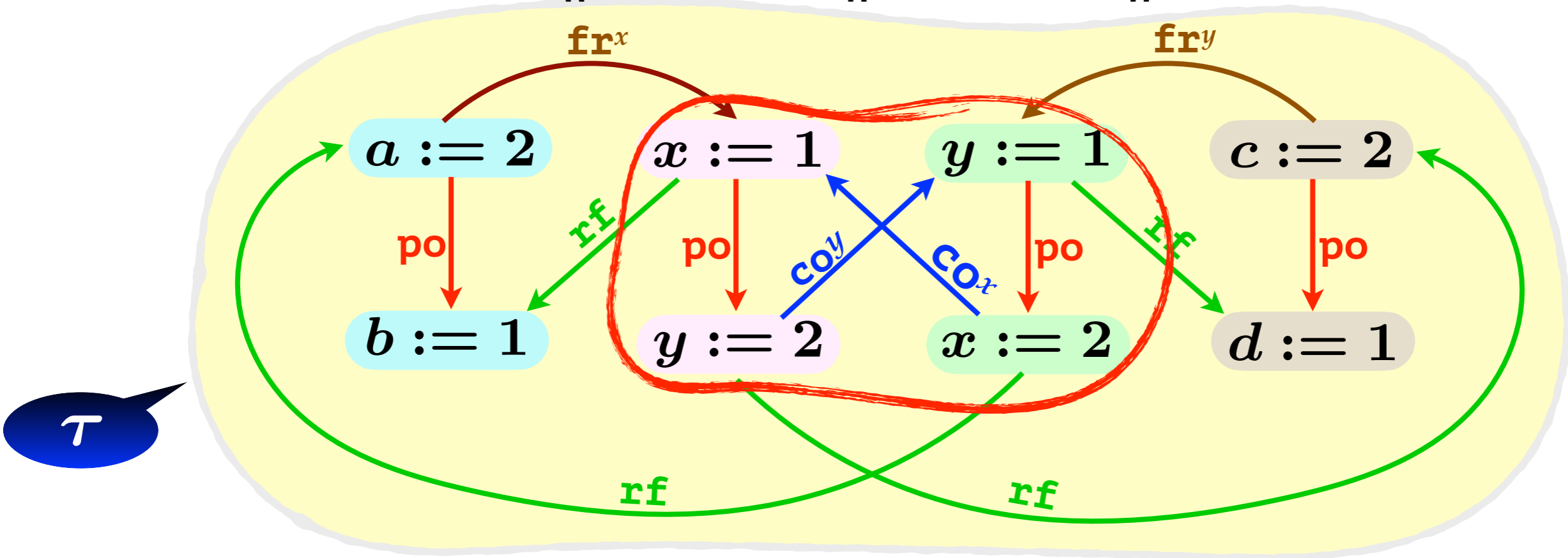
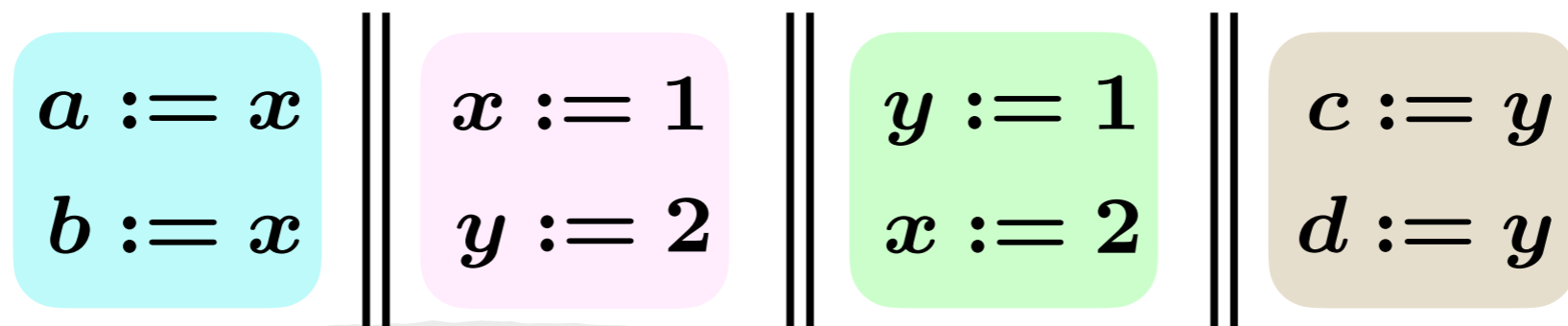
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

$\tau \models \text{RA}$

SC: $\text{acyclic}(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$

$\tau \not\models \text{SC}$

Release-Acquire (RA)

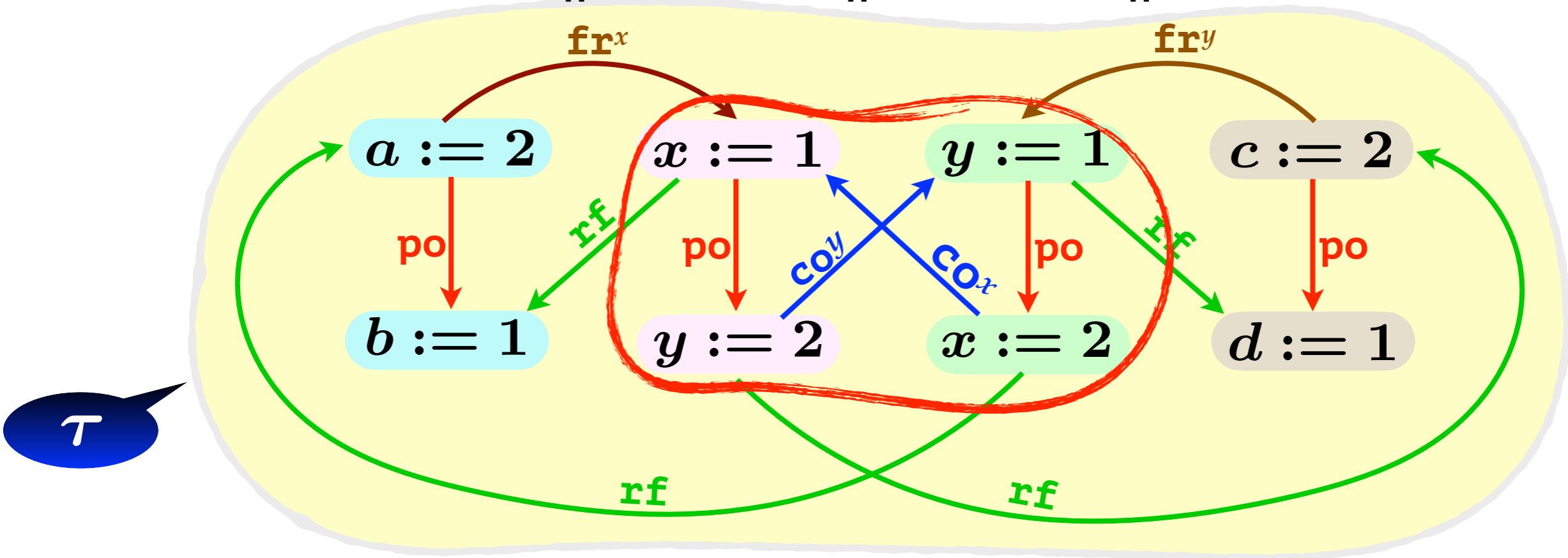
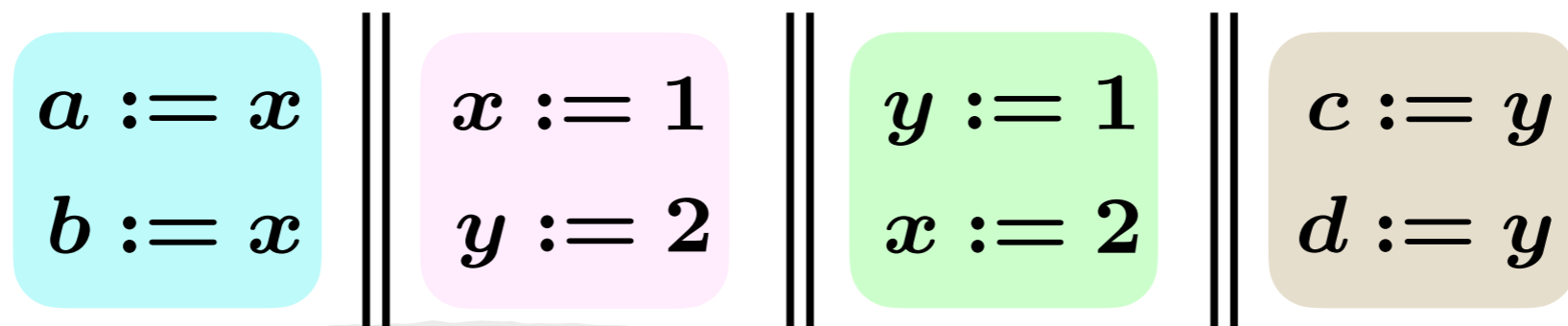


RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$ $\tau \models \text{RA}$

SC: $\text{acyclic}(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$ $\tau \not\models \text{SC}$

$(a = 2) \wedge (b = 1) \wedge (c = 2) \wedge (d = 1)?$

Release-Acquire (RA)



RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$ $\tau \models \text{RA}$

SC: $\text{acyclic}(\text{po} \cup \text{rf} \cup \text{co} \cup \text{fr})$ $\tau \not\models \text{SC}$

$(a = 2) \wedge (b = 1) \wedge (c = 2) \wedge (d = 1)?$

DPOR under the RA-semantics

(OOPSLA'2018)

- **Sound**
 - **only valid traces**
- **Complete**
 - **all valid traces**
- **Optimal**
 - **different po+rf**
- **Deadlock-free**

DPOR under the RA-semantics

- Build **traces** one after one
- For a given trace \mathcal{T} :
 - Build \mathcal{T} incrementally
 - extend \mathcal{T} by one event e at a time

- Sound
 - only valid traces
- Complete
 - all valid traces
- Optimal
 - different po+rf
- Deadlock-free

DPOR under the RA-semantics

- Build **traces** one after one
- For a given trace \mathcal{T} :
 - Build \mathcal{T} incrementally
 - extend \mathcal{T} by one event e at a time

DPOR under the RA-semantics

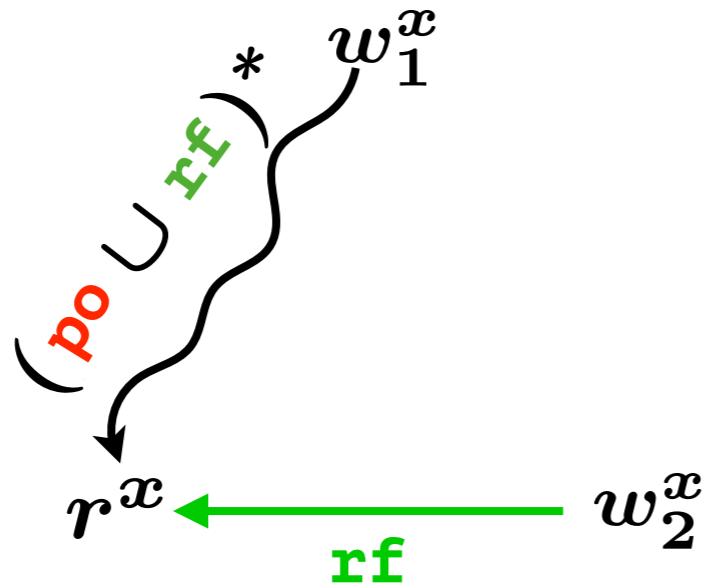
- Build **traces** one after one
- For a given trace \mathcal{T} :
 - Build \mathcal{T} incrementally
 - extend \mathcal{T} by one event e at a time

DPOR under the RA-semantics

- Build **traces** one after one
- For a given trace τ :
 - Build τ incrementally
 - extend τ by one event e at a time

$$\tau \oplus e$$

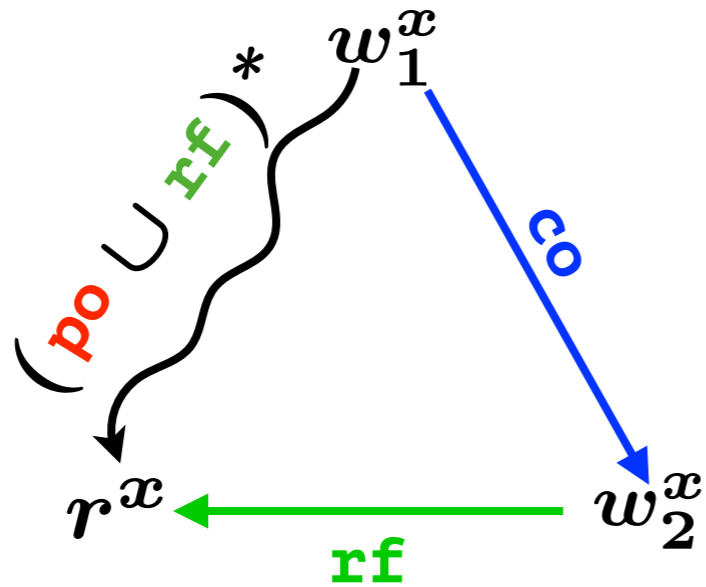
Saturation



- Add **co**-edges by demand
- never generate two traces with identical **po** and **rf**

RA: $\forall x. \text{acyclic}(po \cup rf \cup co^x \cup fr^x)$

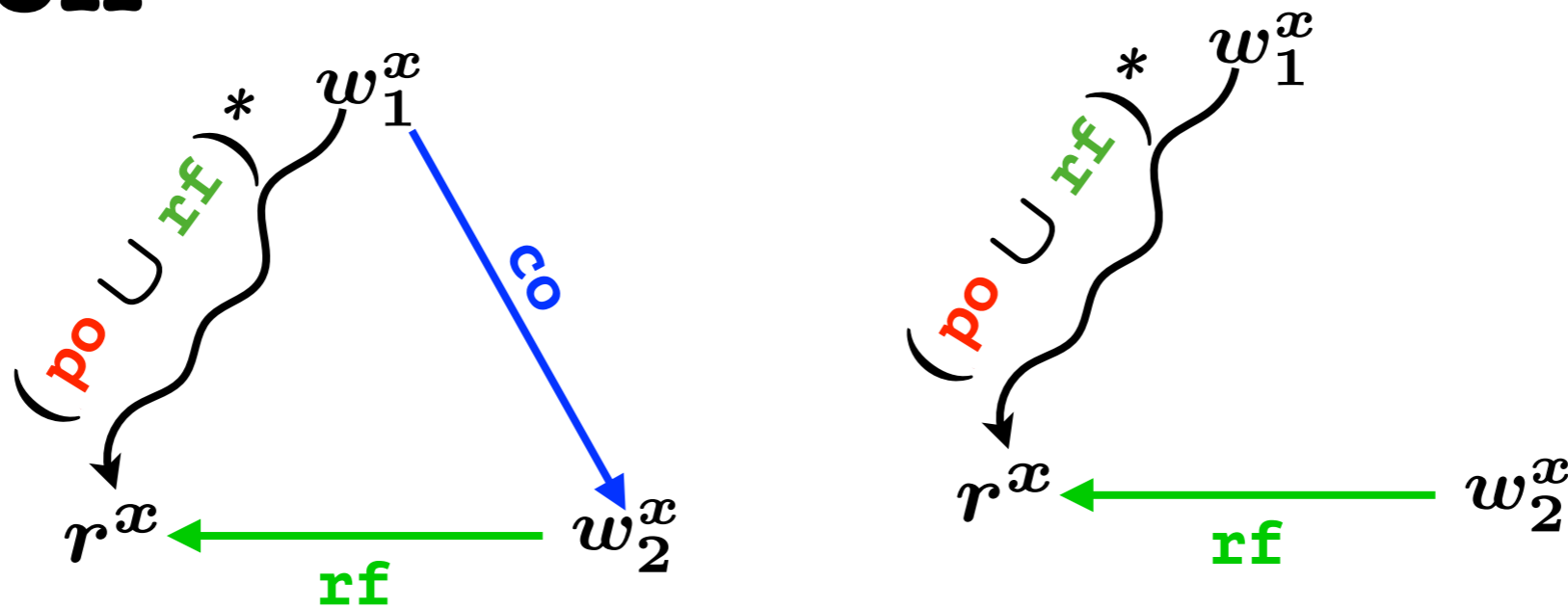
Saturation



- Add **co**-edges by demand
- never generate two traces with identical **po** and **rf**

RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

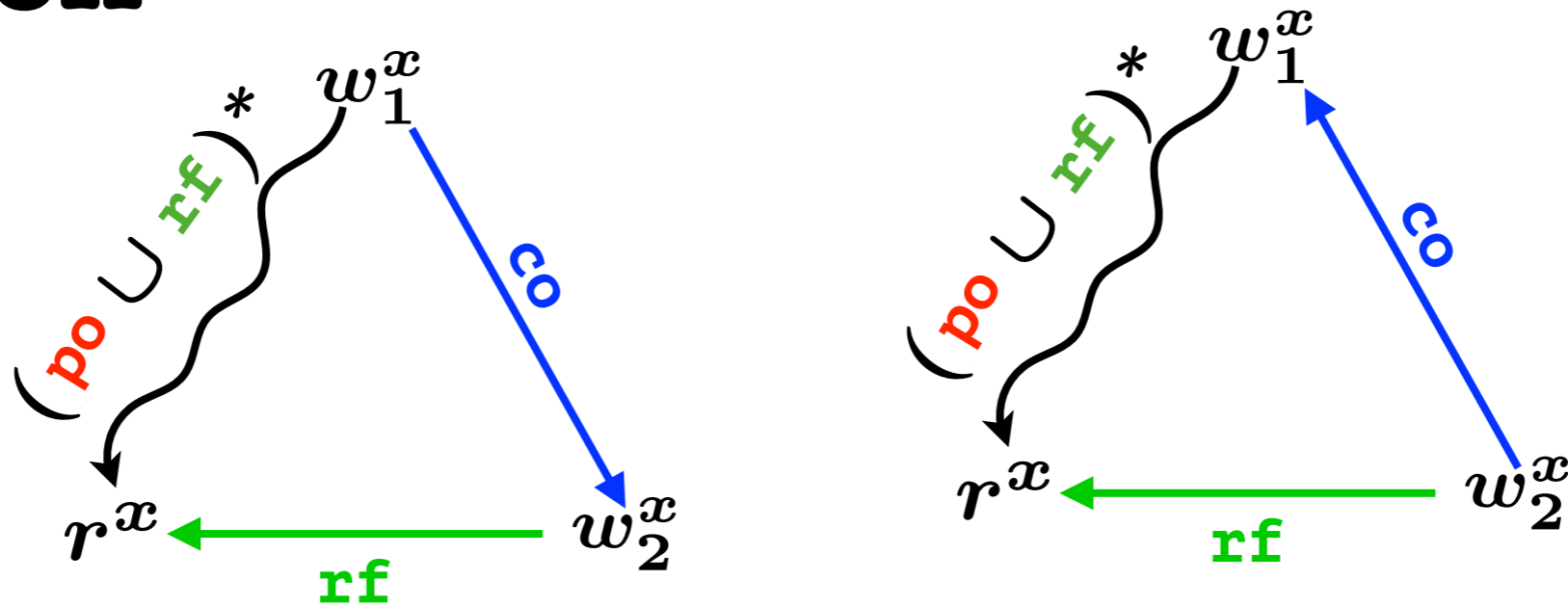
Saturation



- Add **co**-edges by demand
- never generate two traces with identical **po** and **rf**

RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

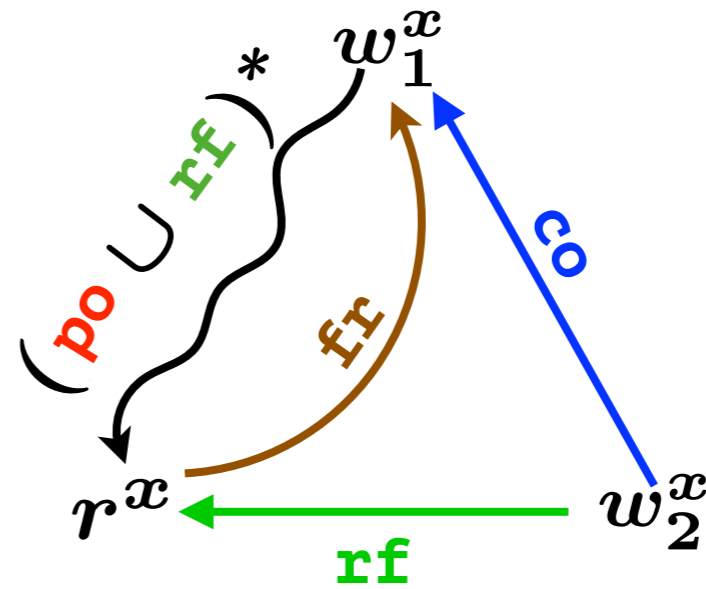
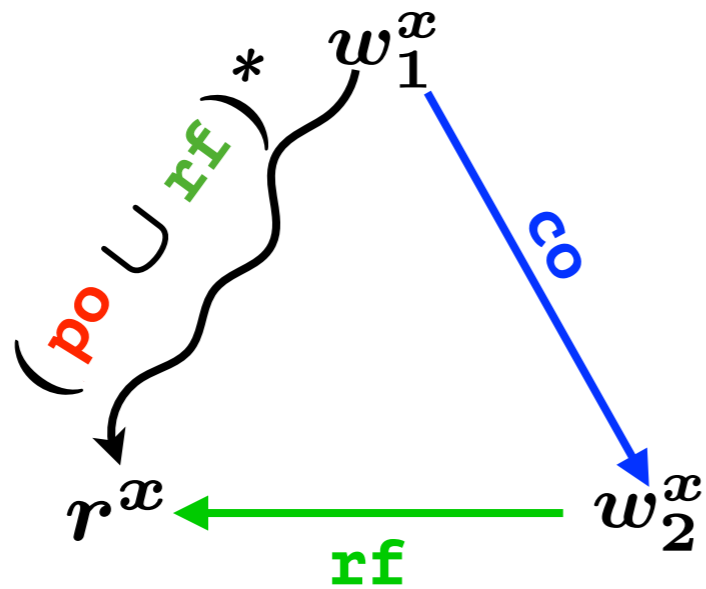
Saturation



- Add **co**-edges by demand
- never generate two traces with identical **po** and **rf**

RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

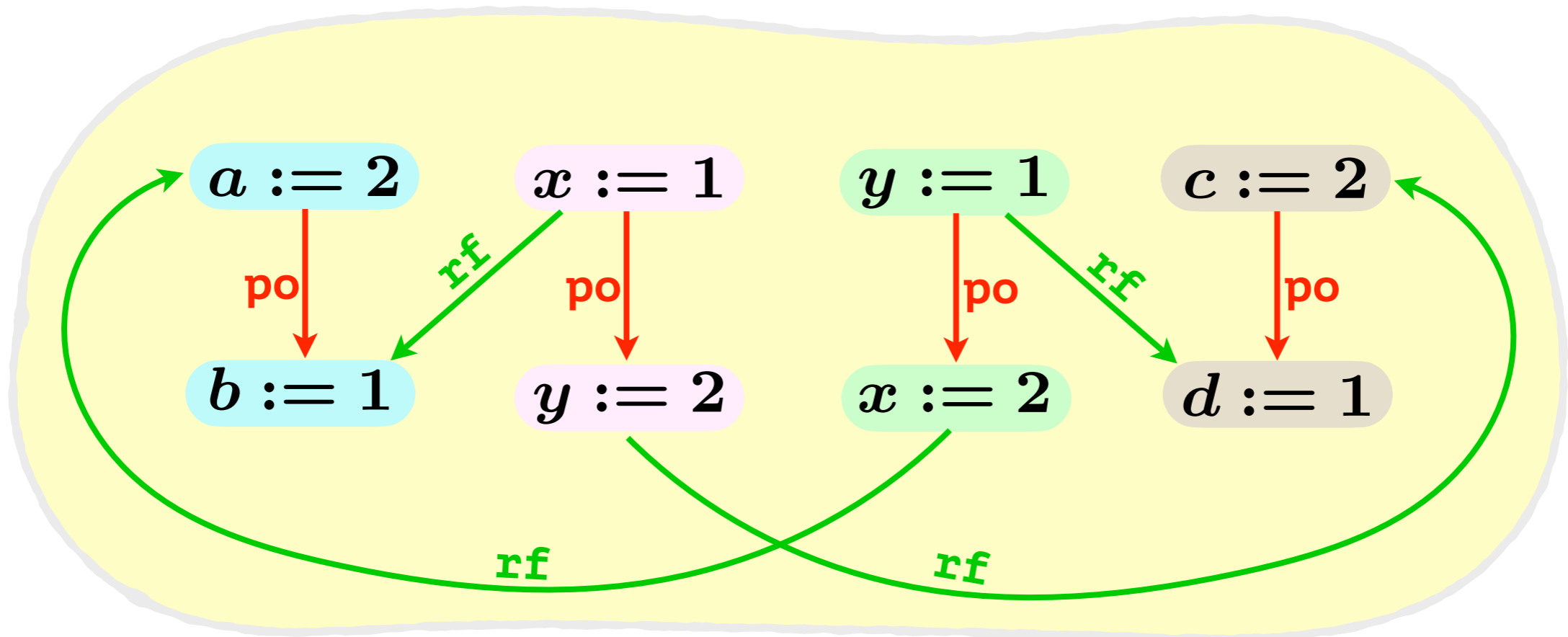
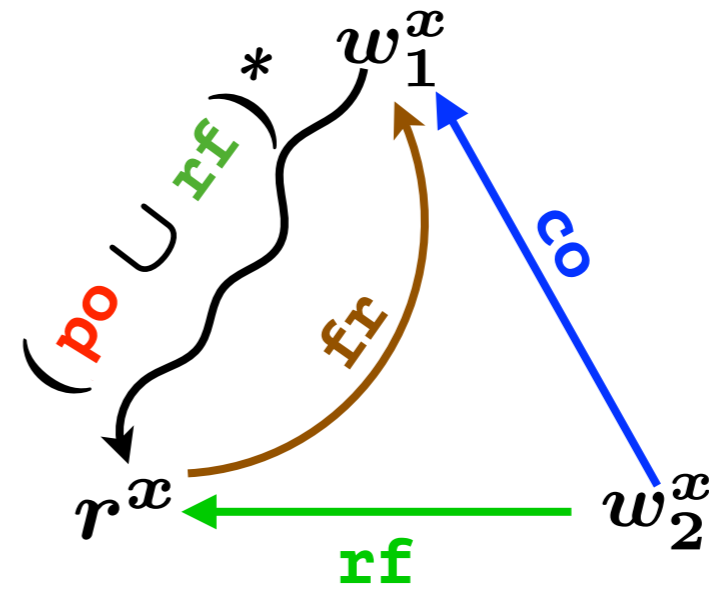
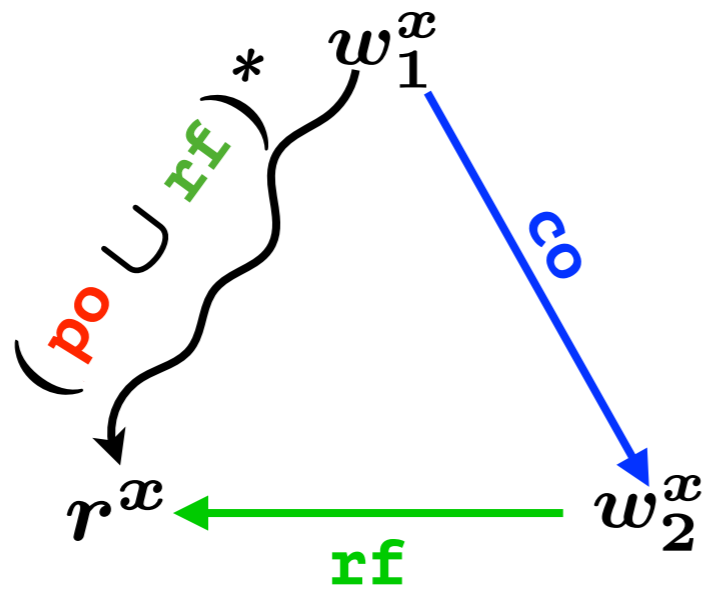
Saturation



- Add **co**-edges by demand
- never generate two traces with identical **po** and **rf**

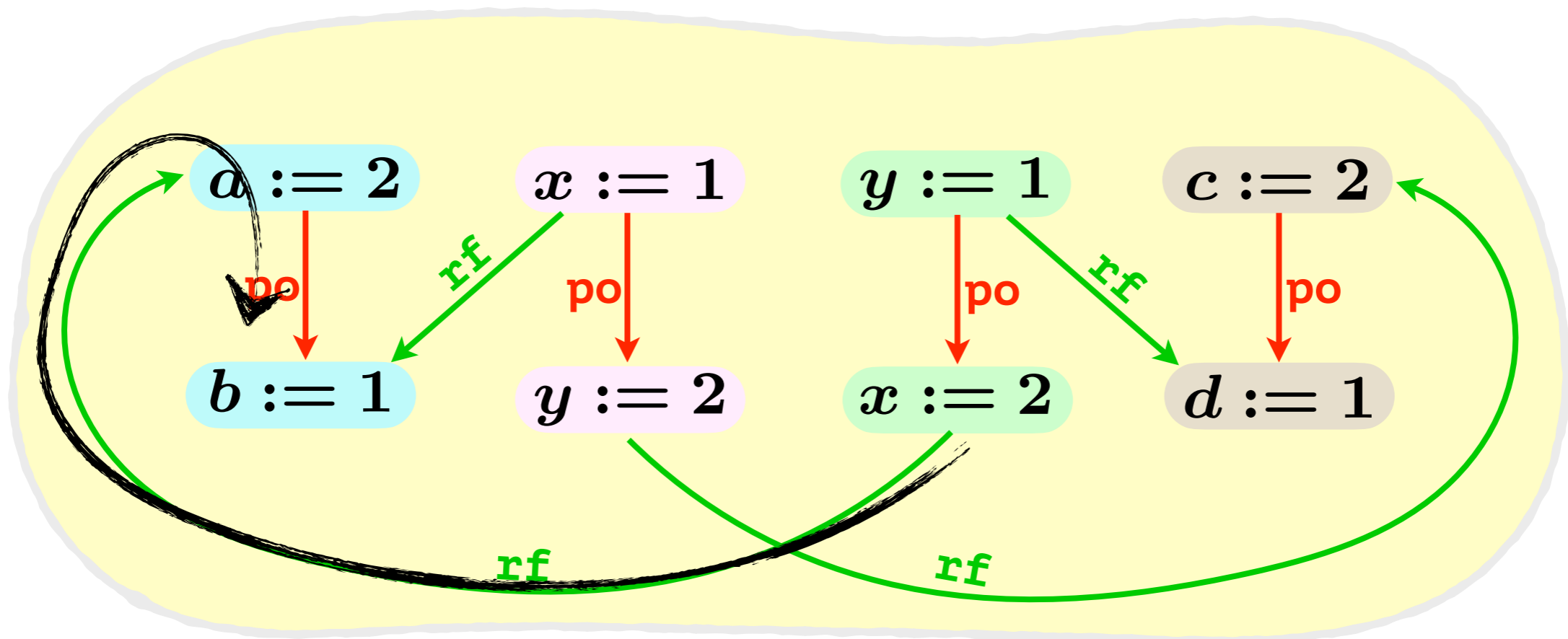
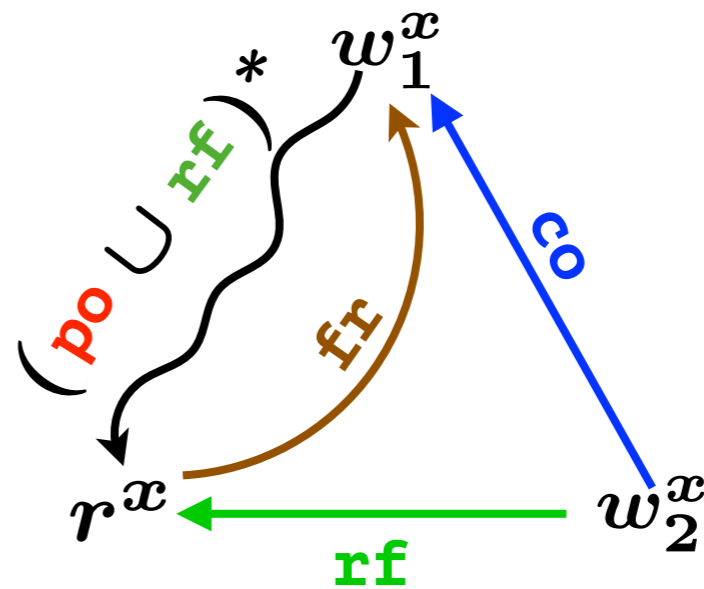
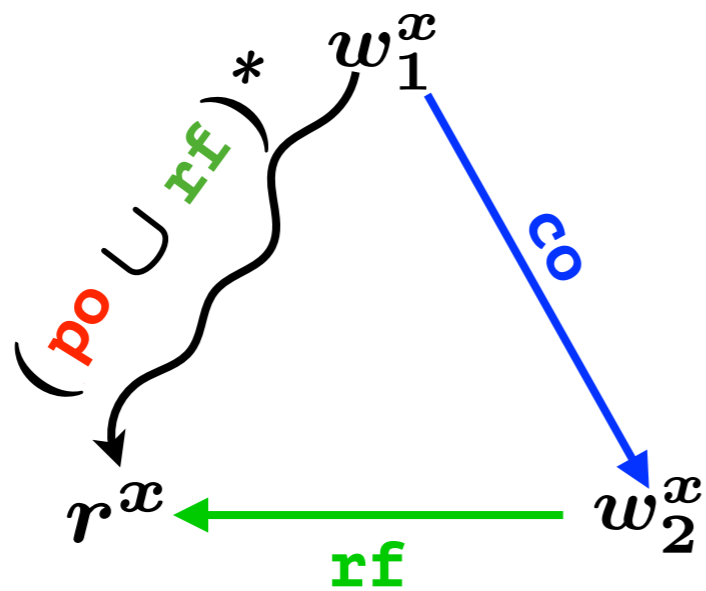
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Saturation



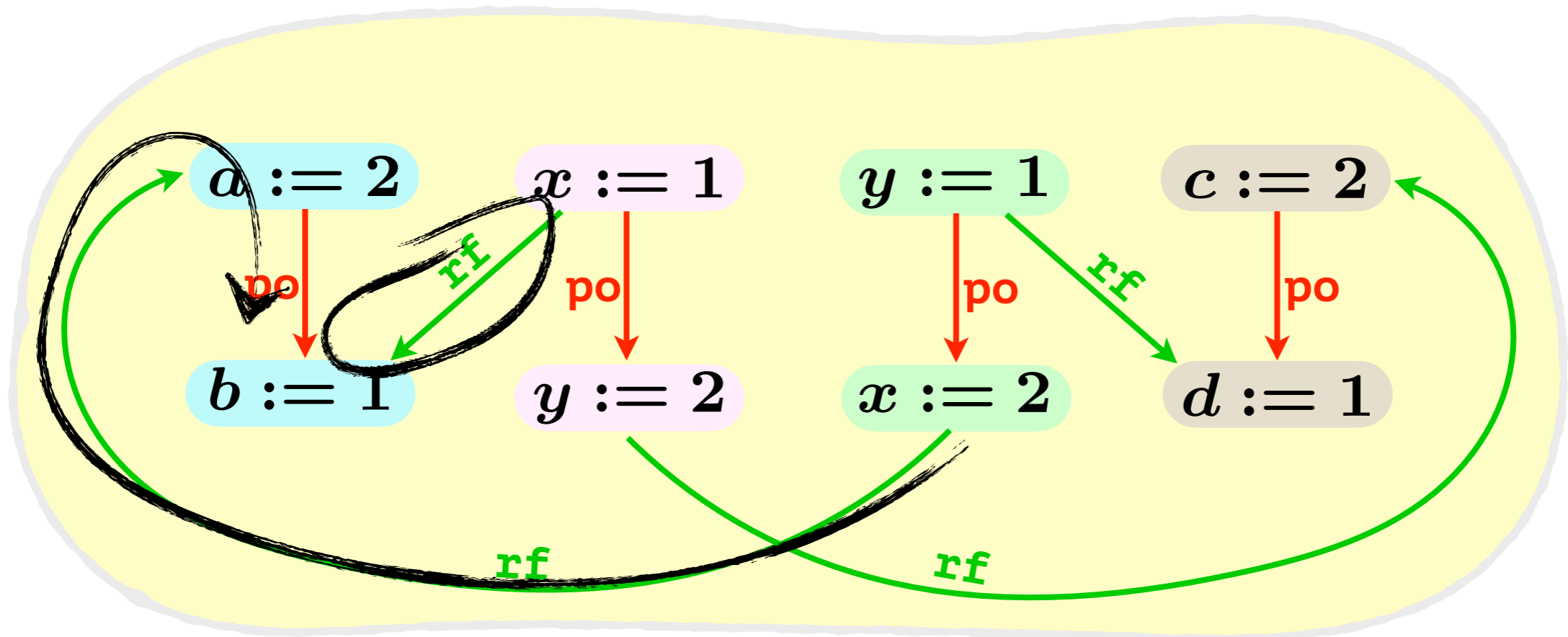
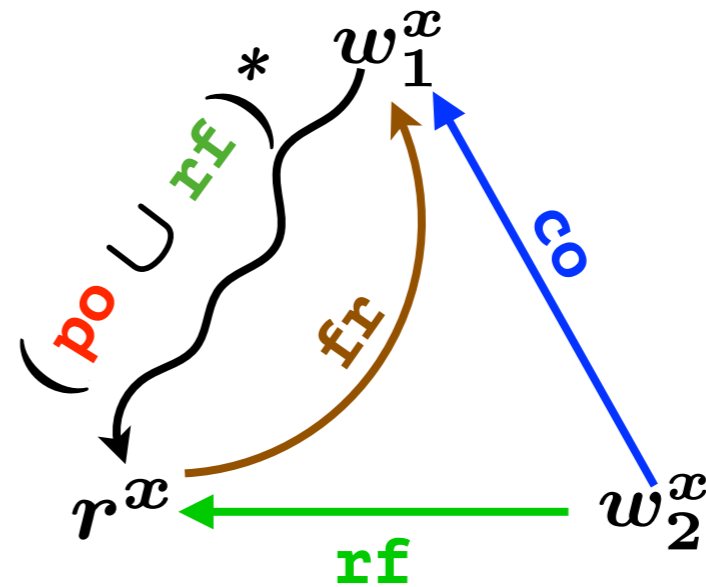
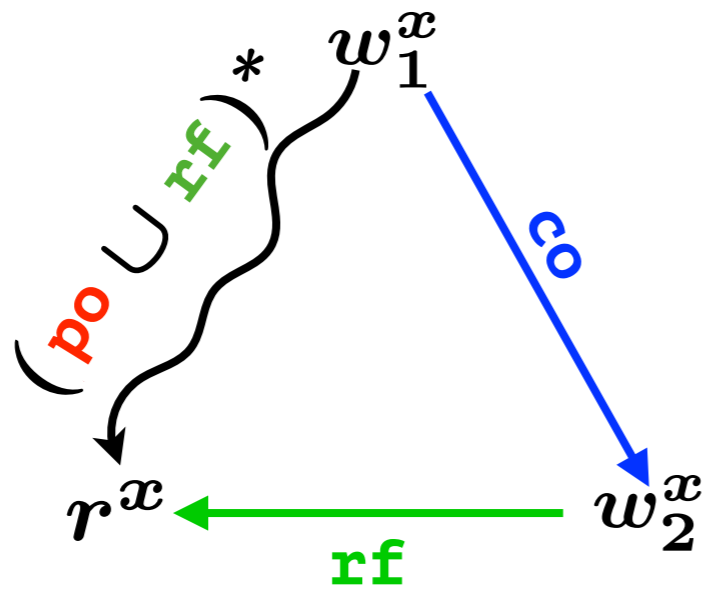
RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

Saturation



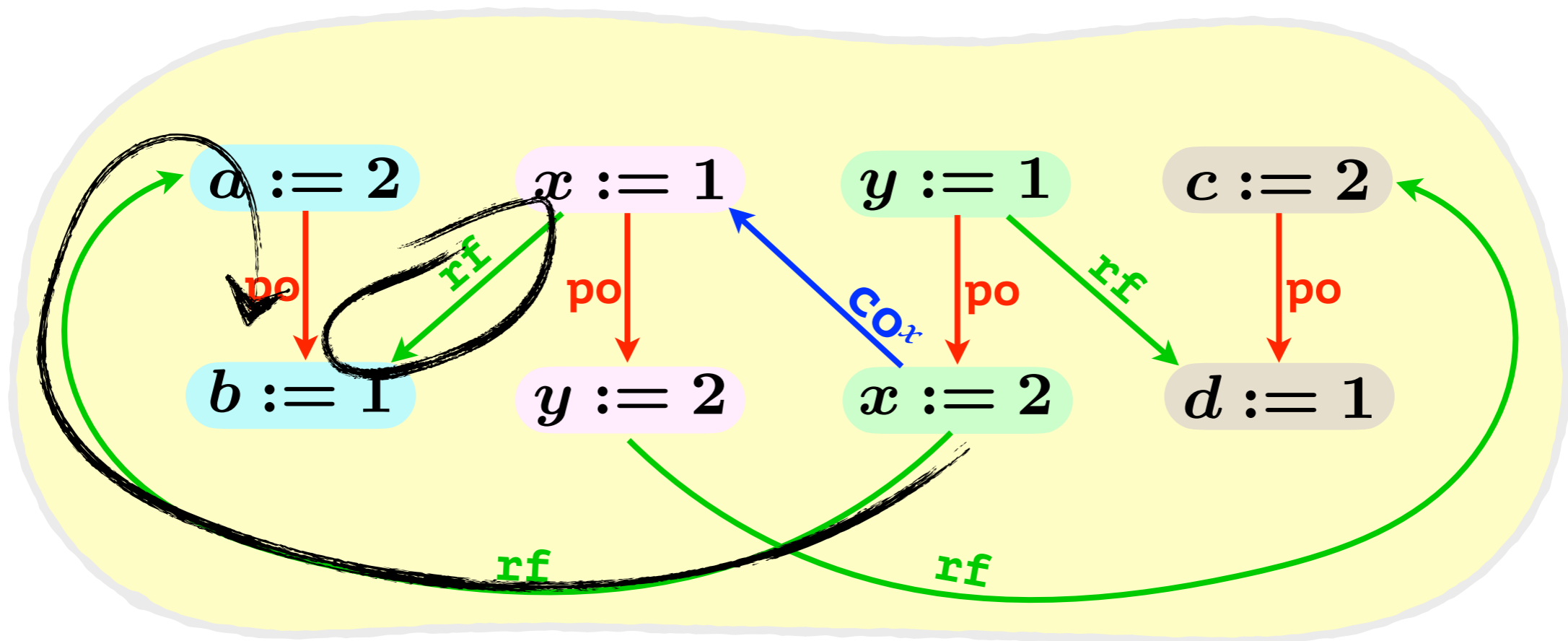
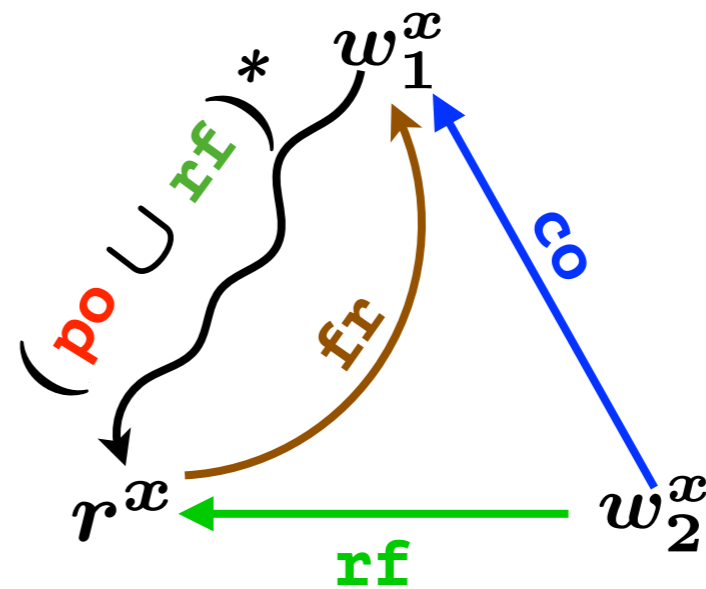
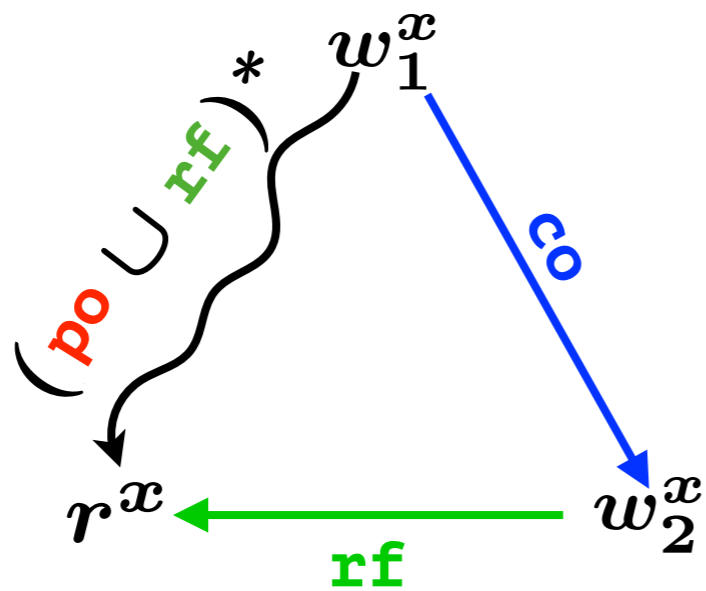
RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

Saturation



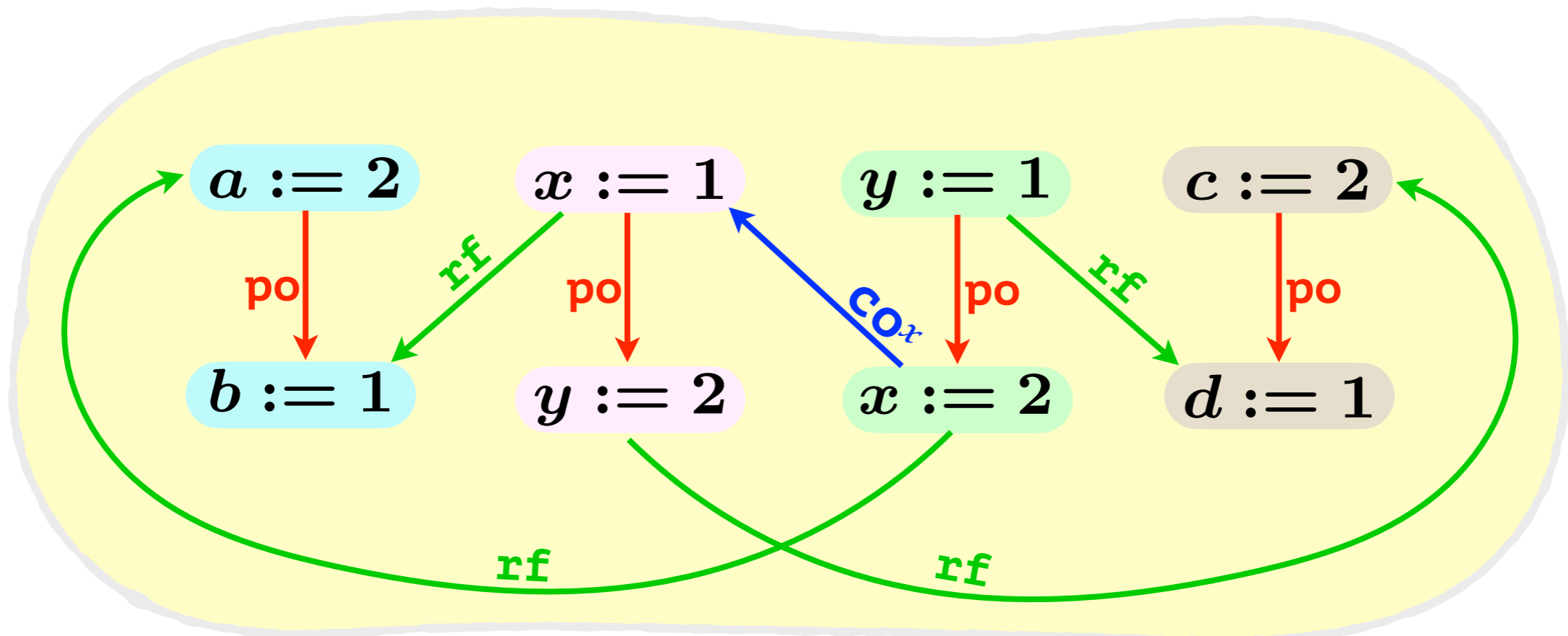
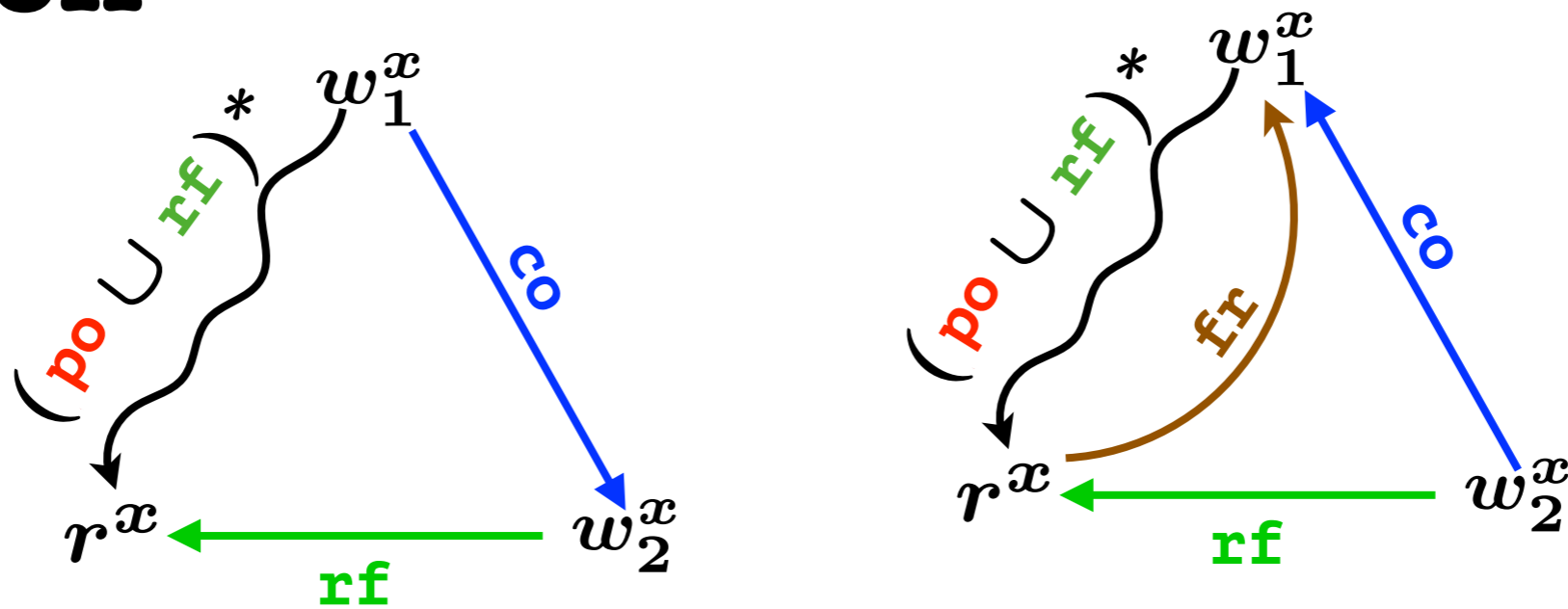
RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

Saturation



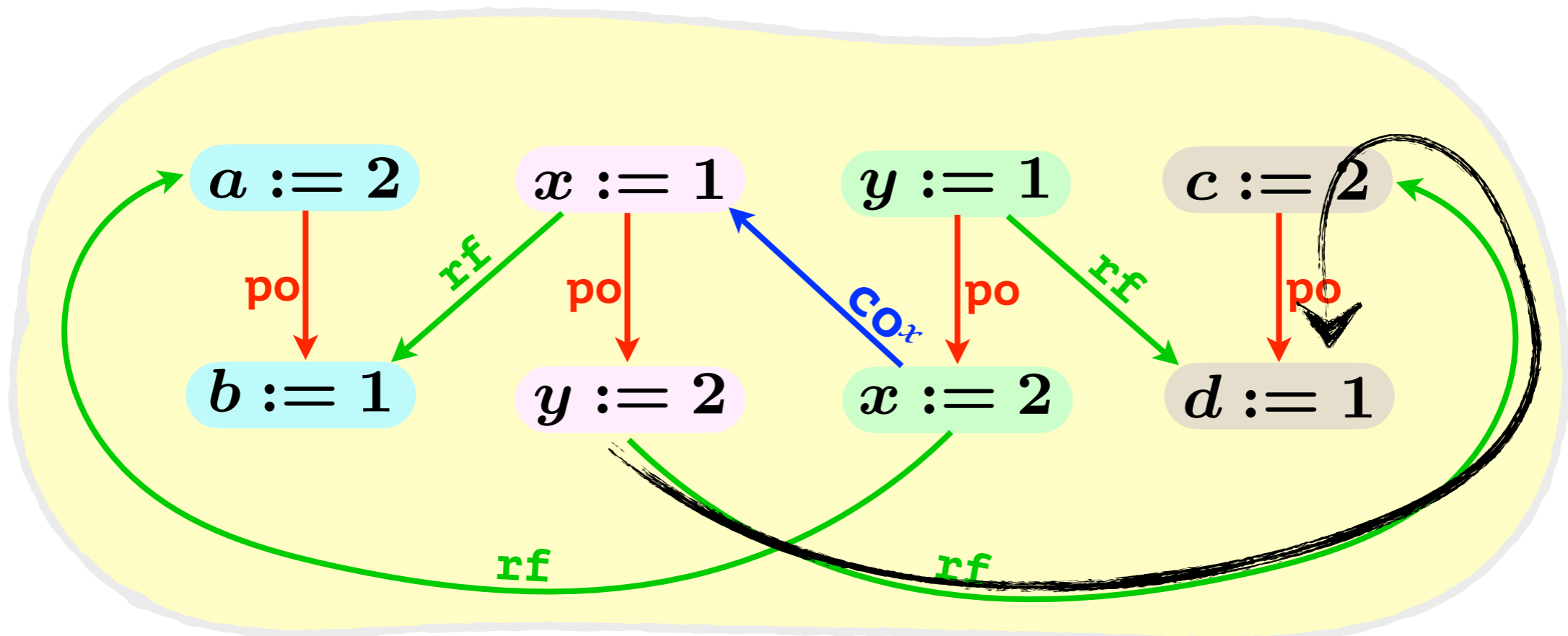
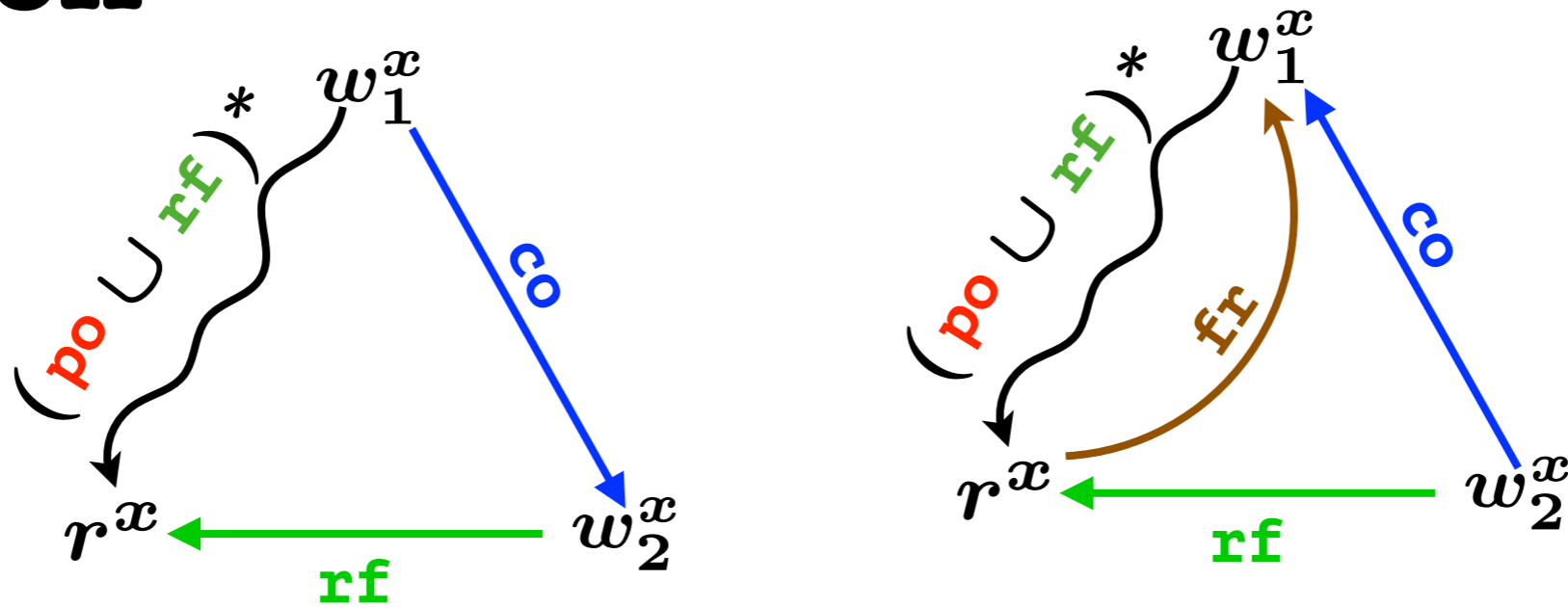
RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

Saturation



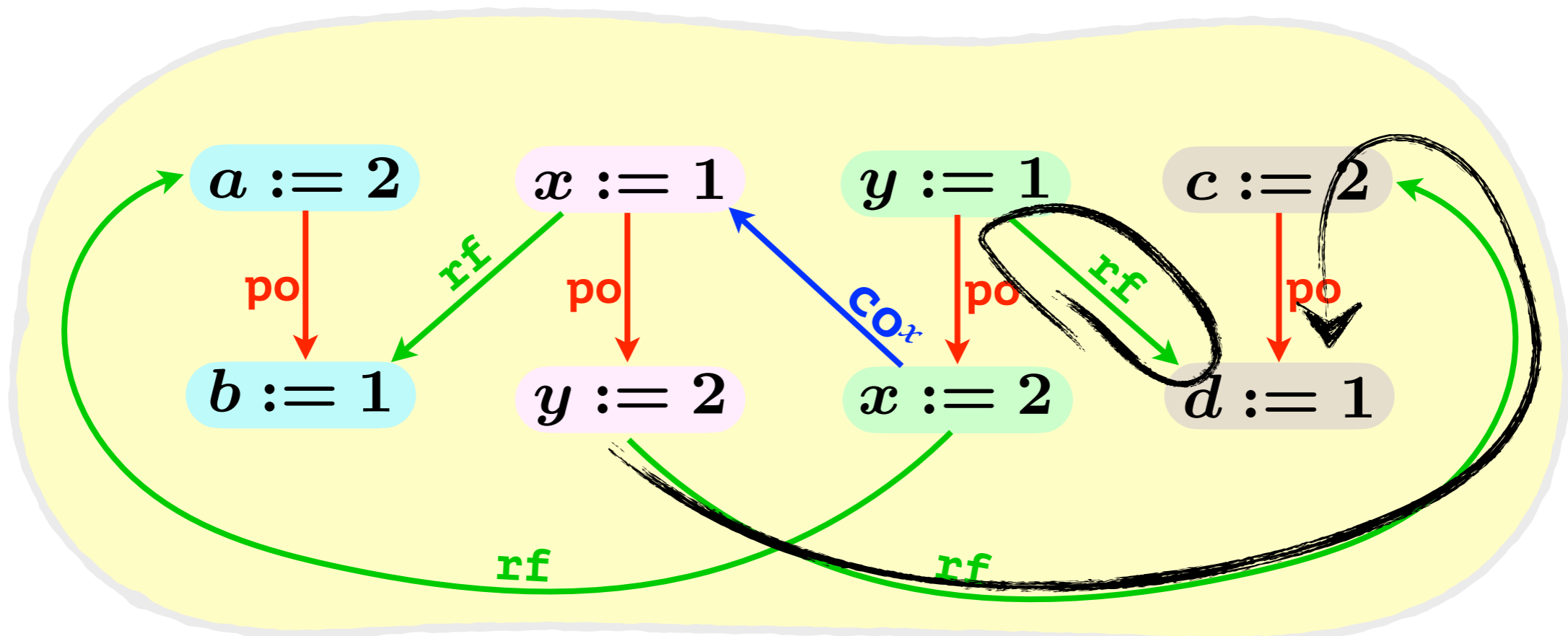
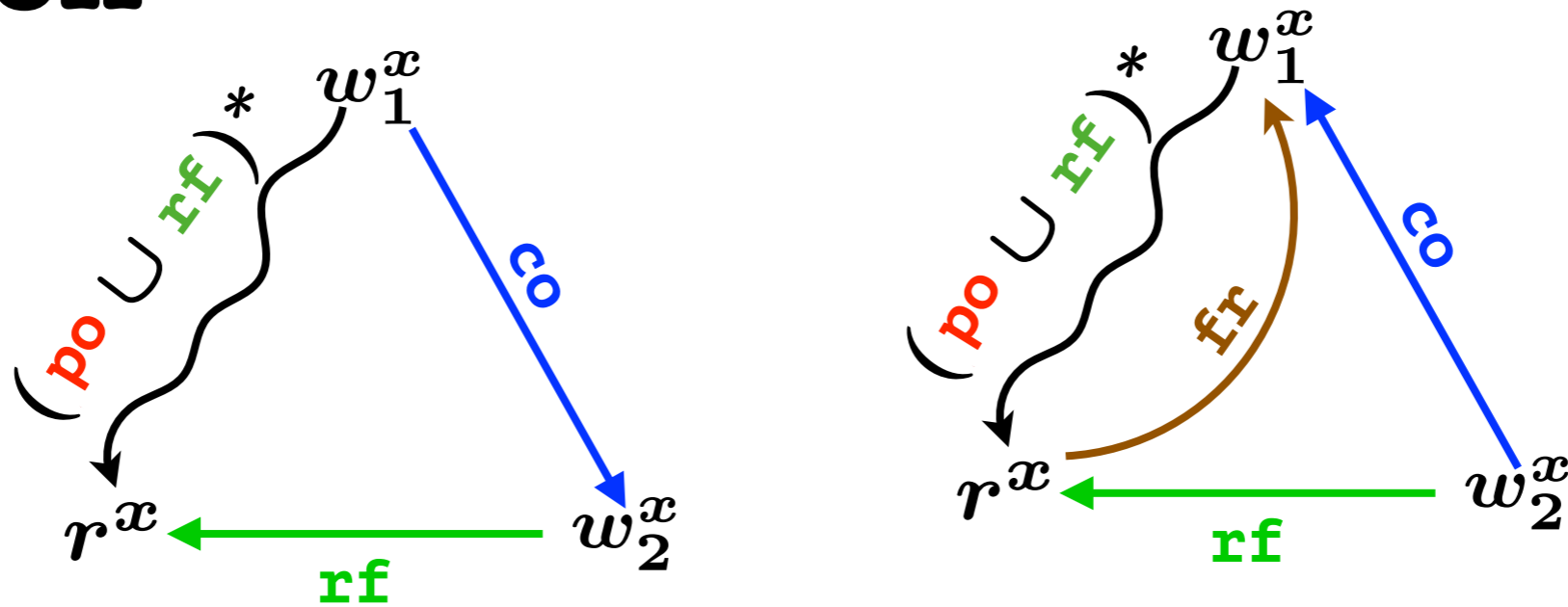
RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

Saturation



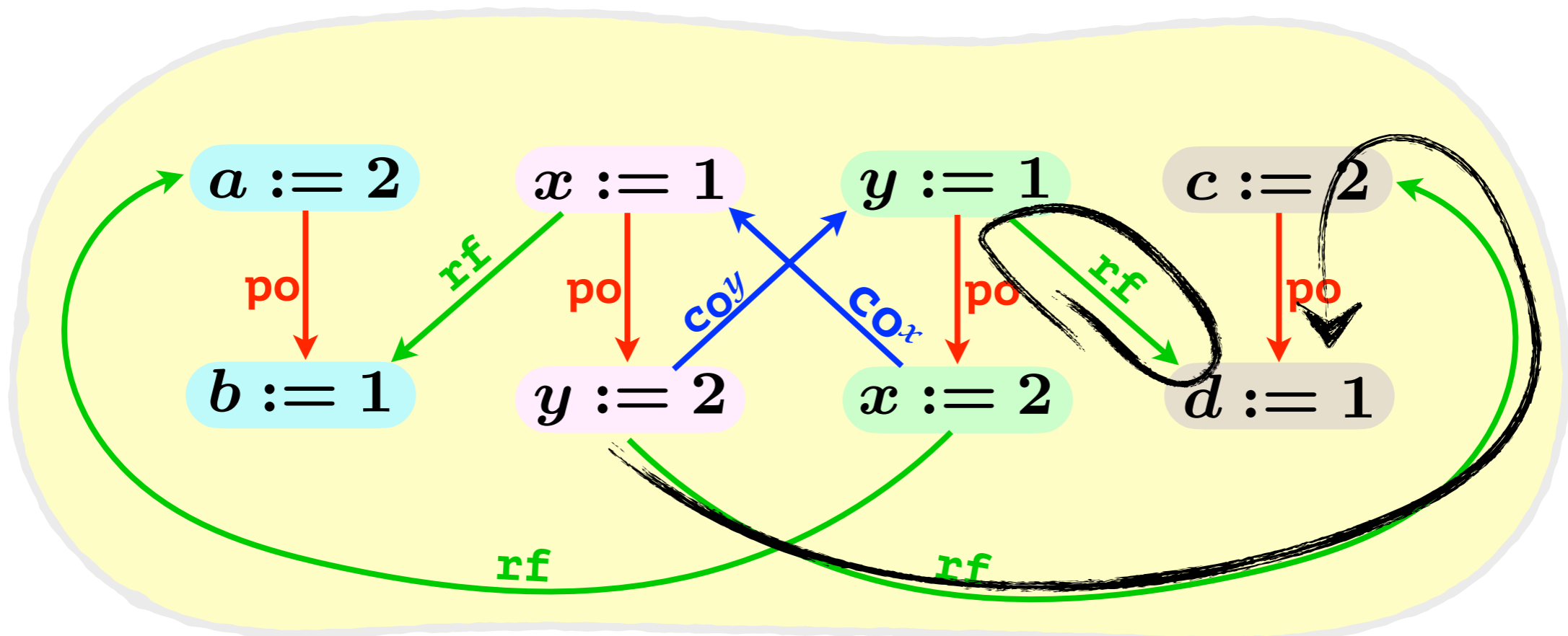
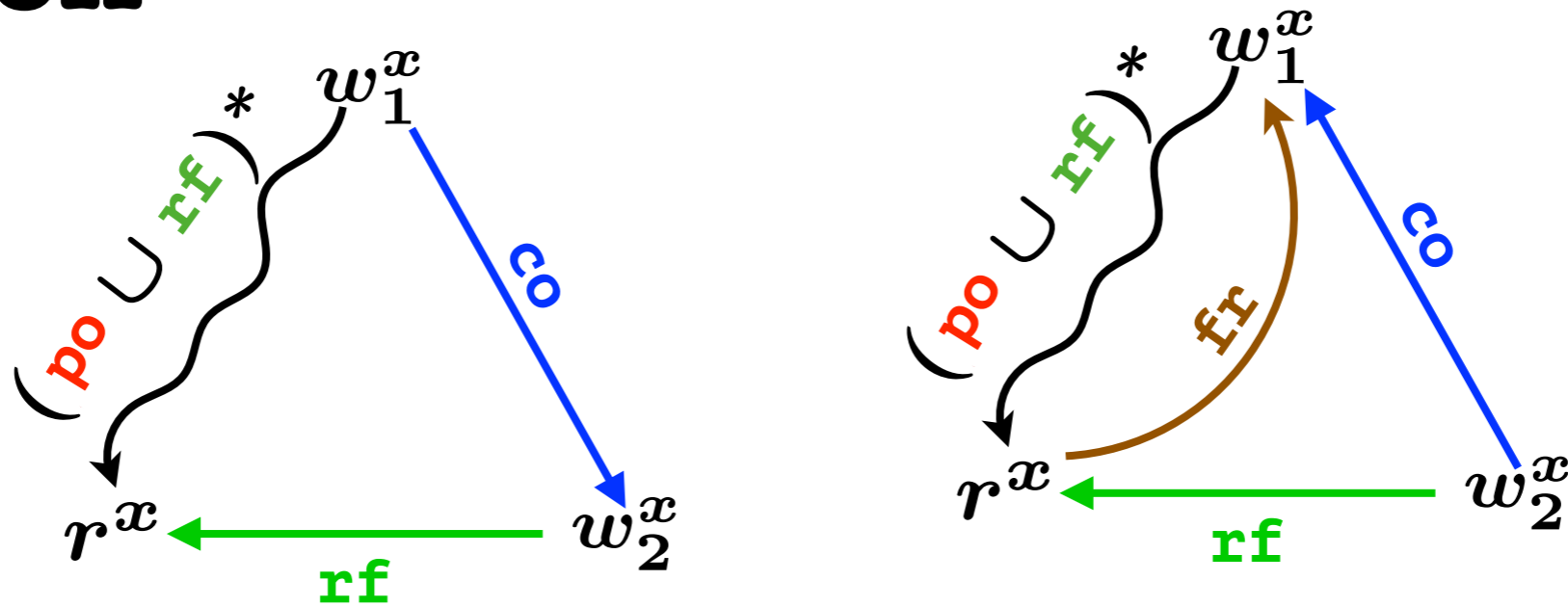
RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

Saturation



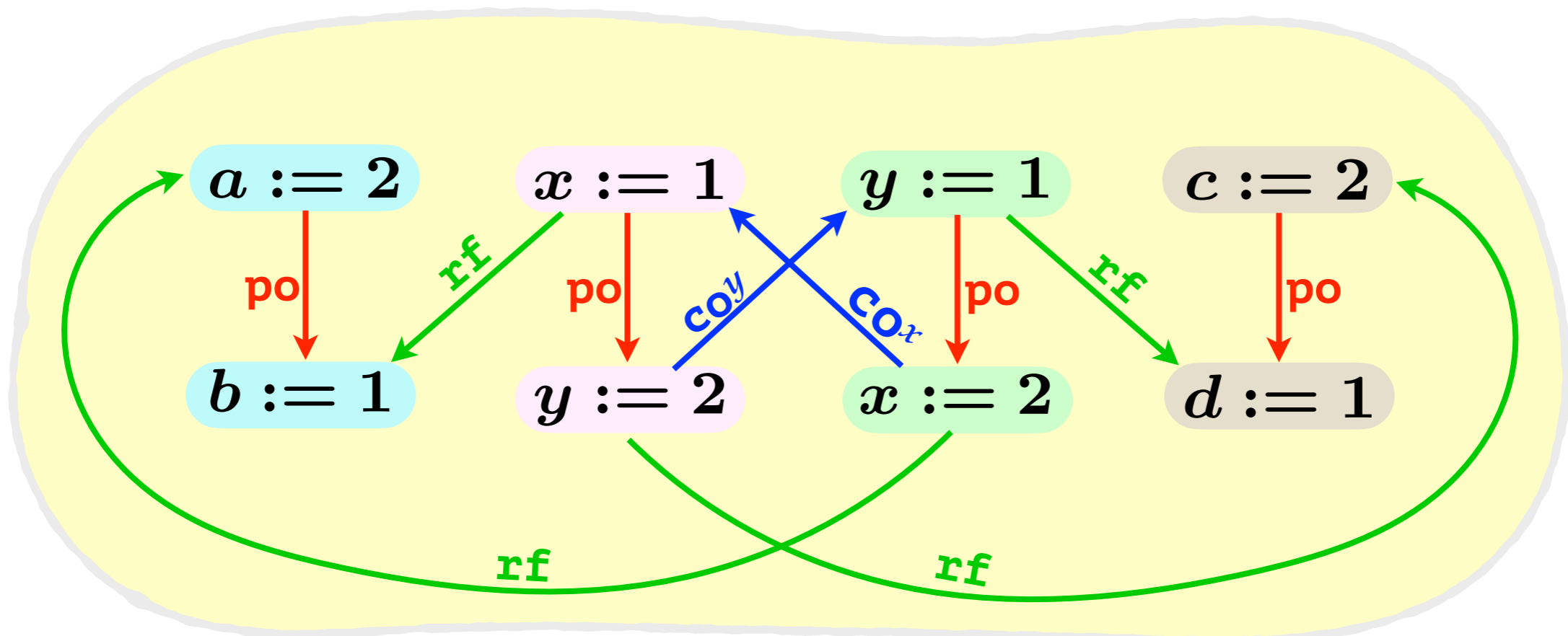
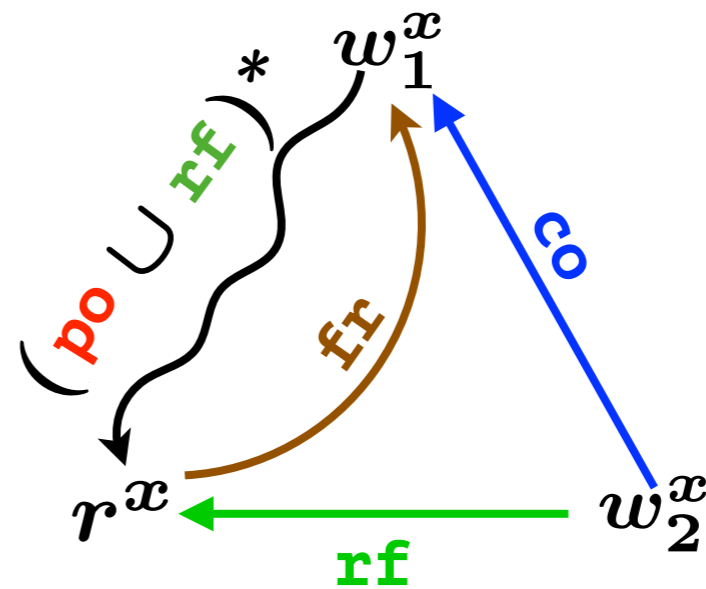
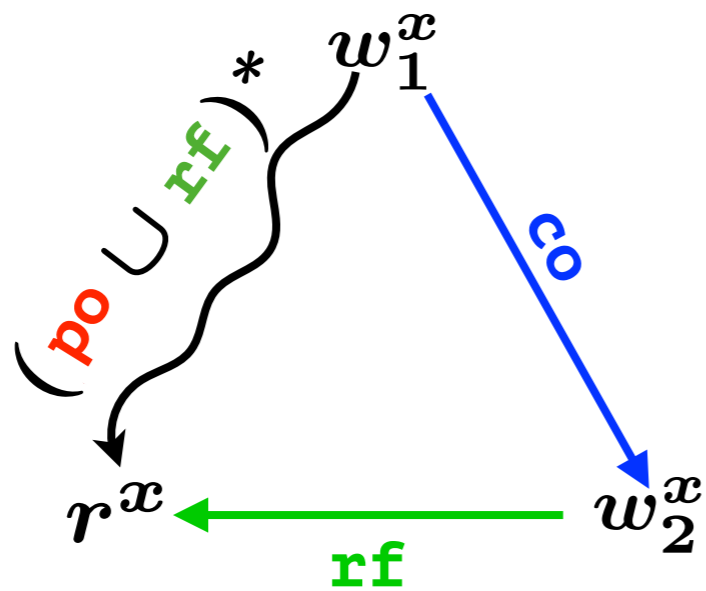
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Saturation



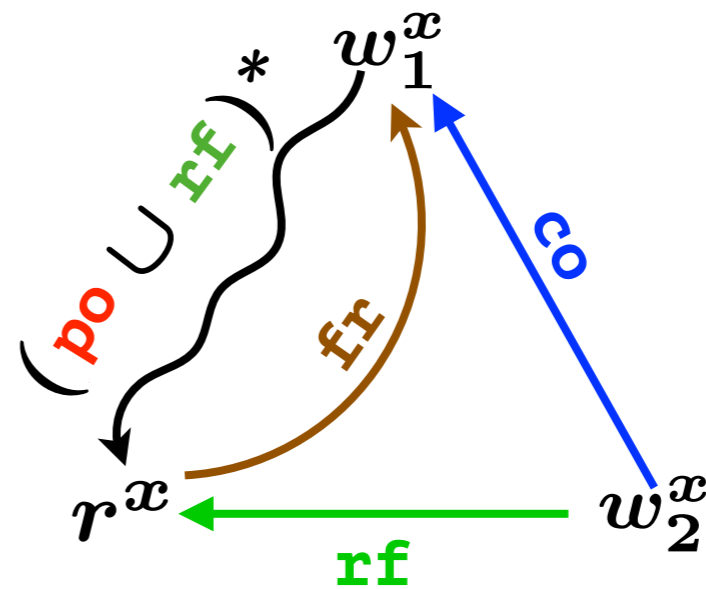
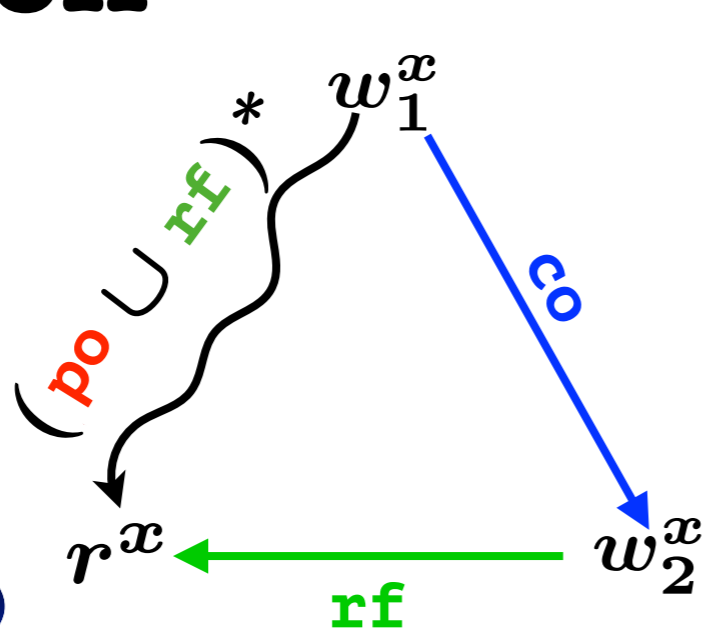
RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Saturation

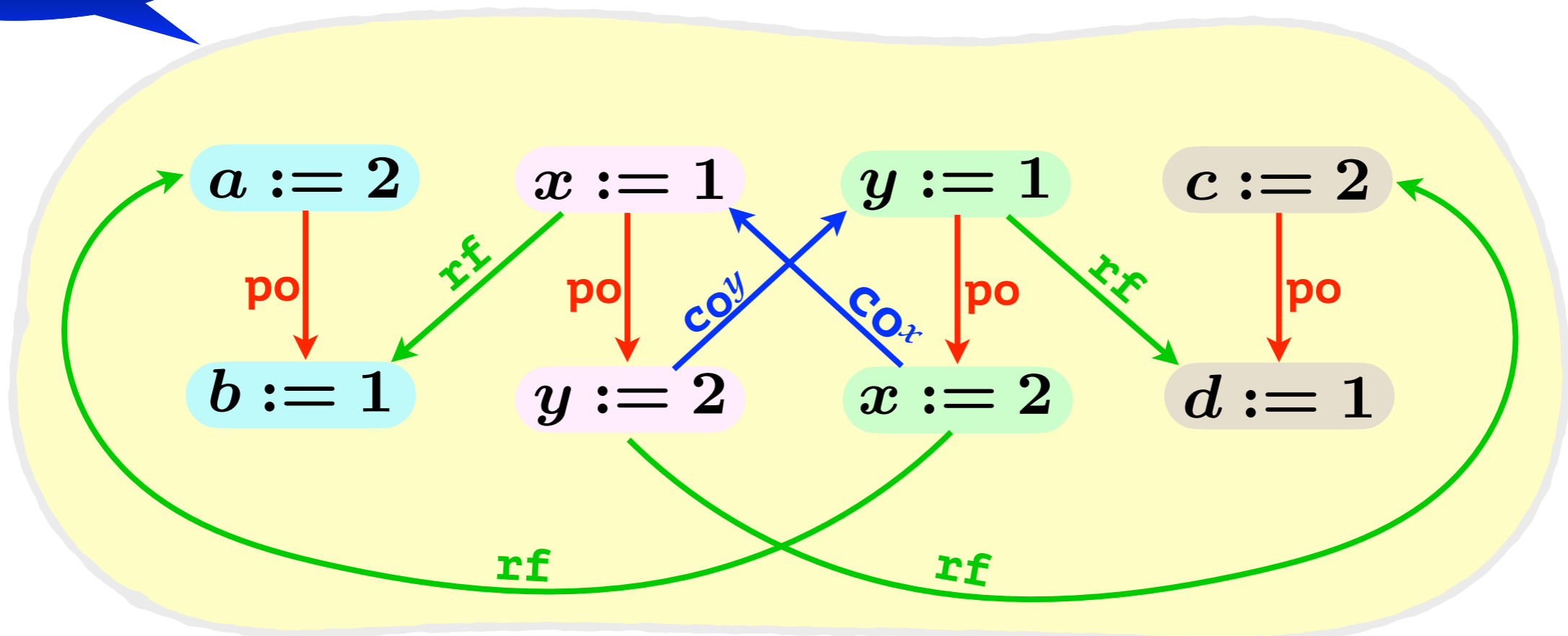


RA: $\forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$

Saturation

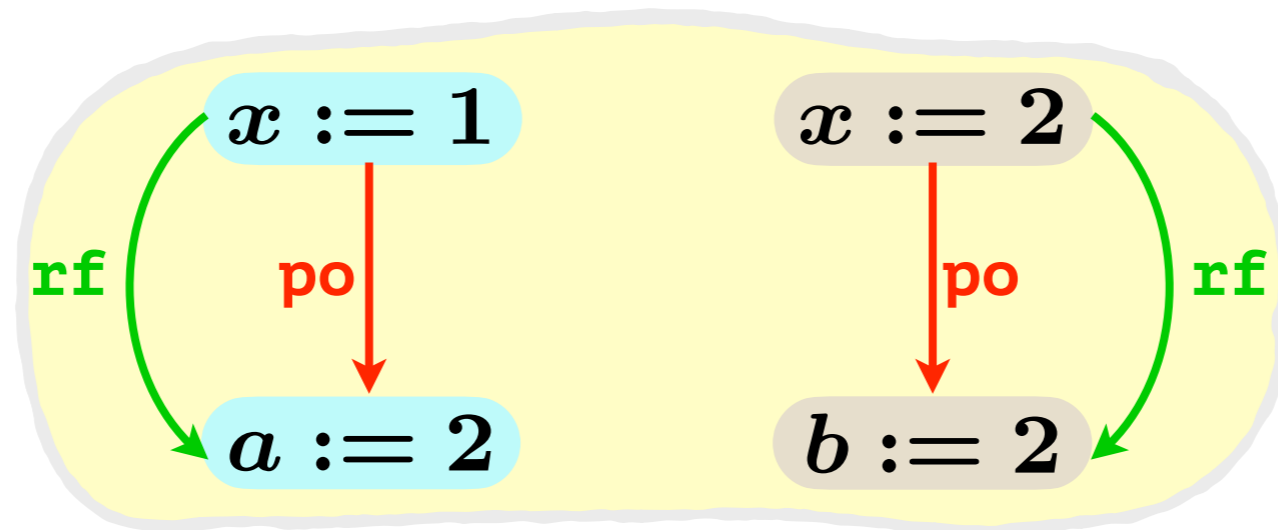
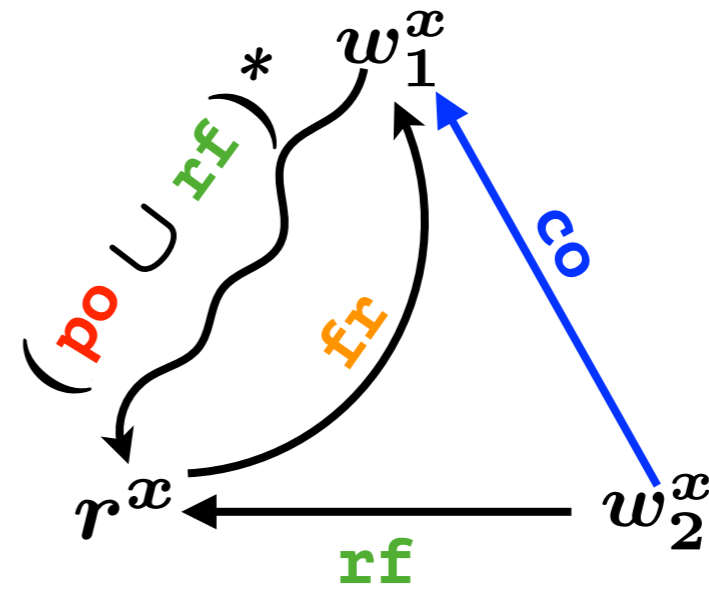
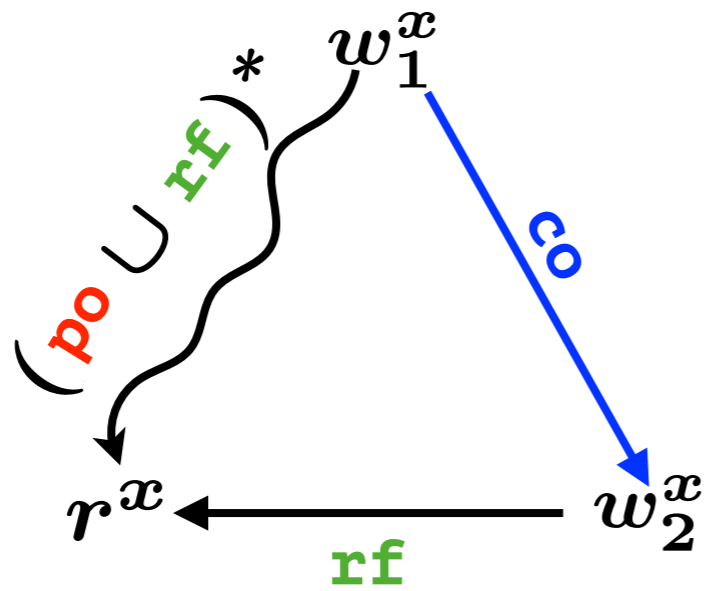


saturated



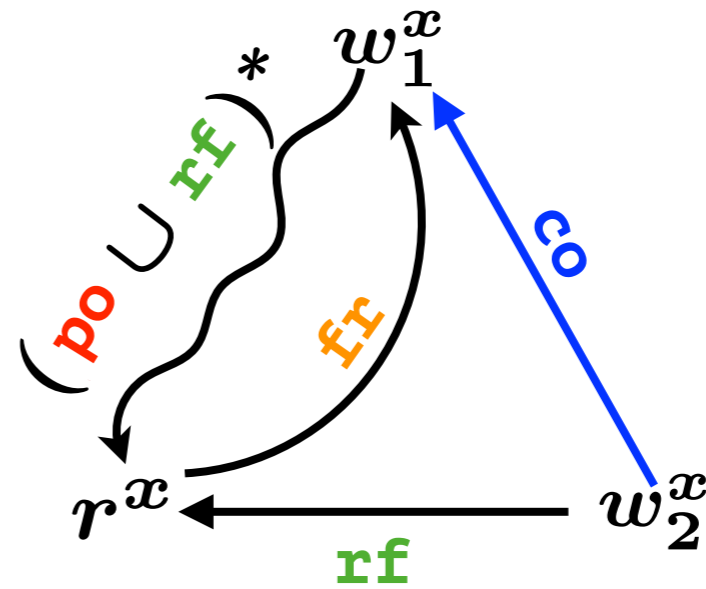
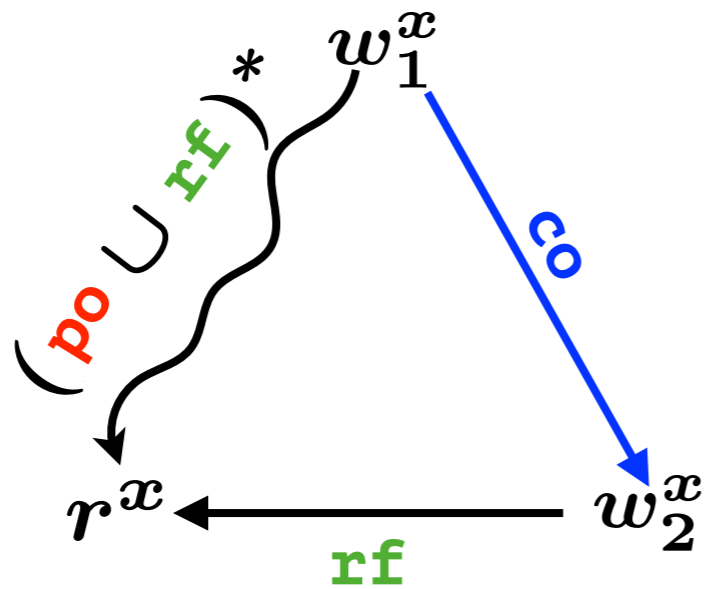
$$RA: \forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$$

Saturation

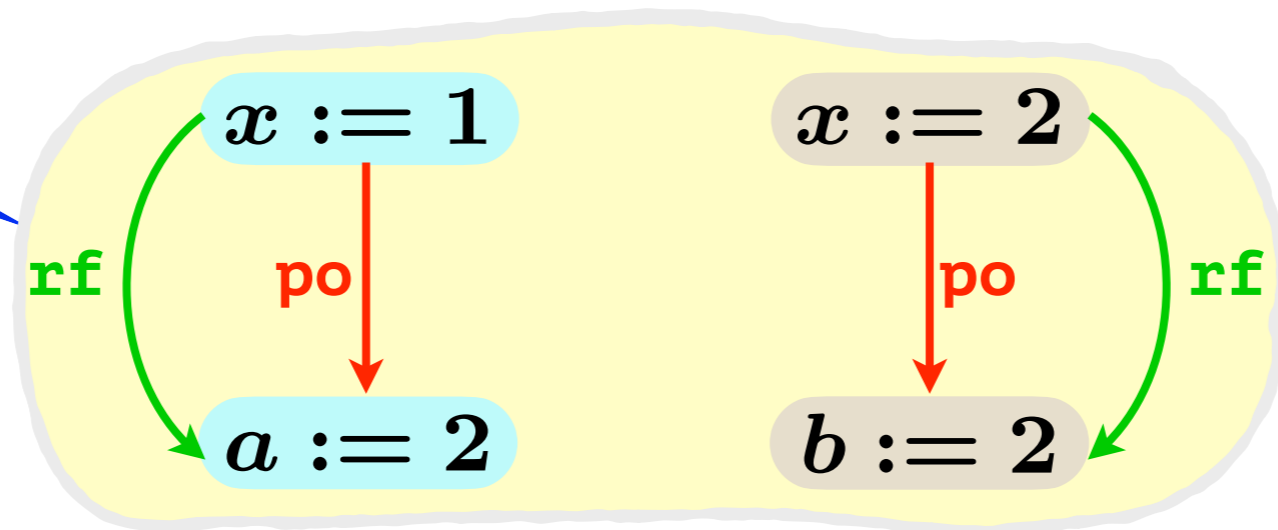


RA: $\forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$

Saturation

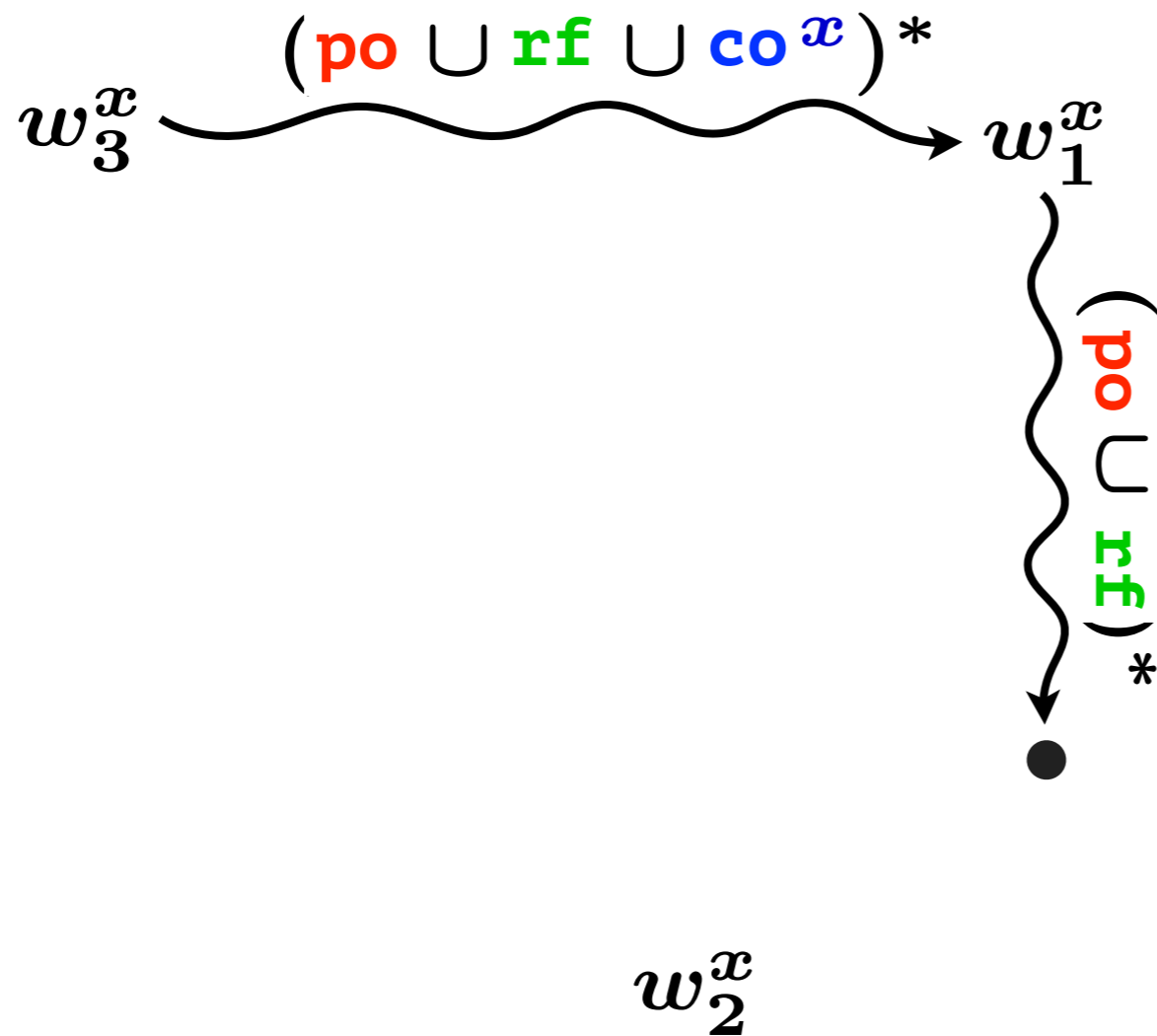


saturated

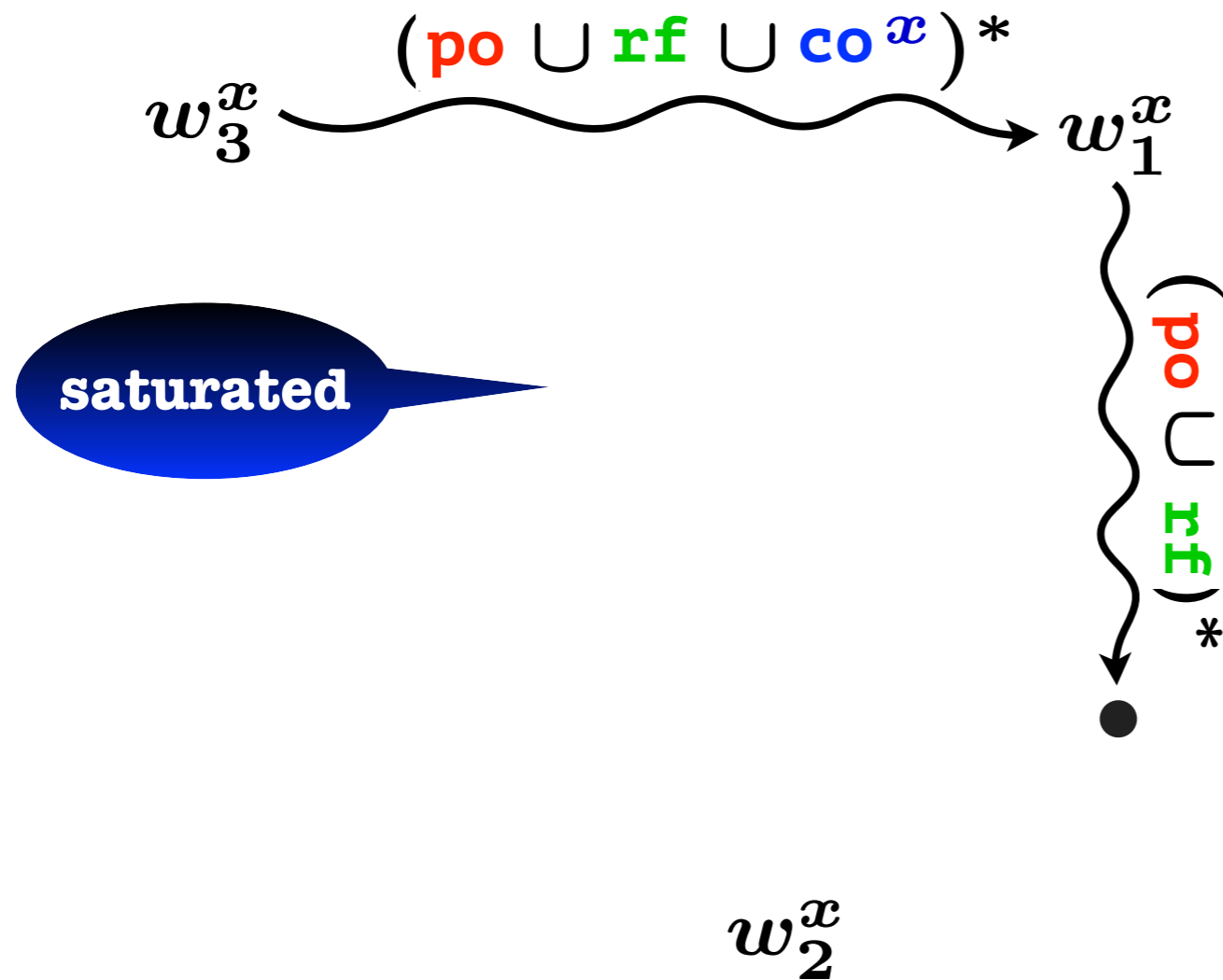


$$RA: \forall x. acyclic(po \cup rf \cup co^x \cup fr^x)$$

Adding Events to Traces

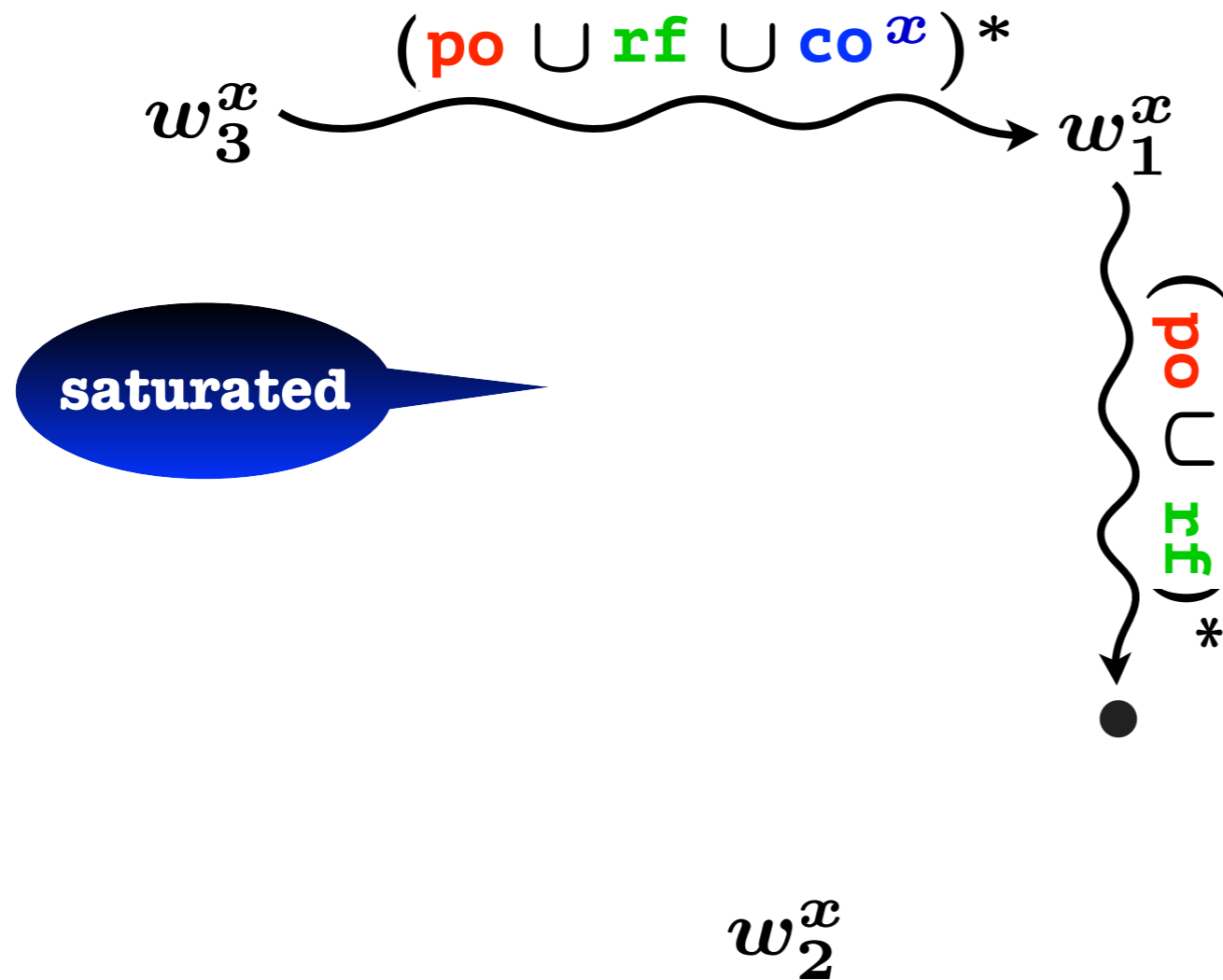


Adding Events to Traces

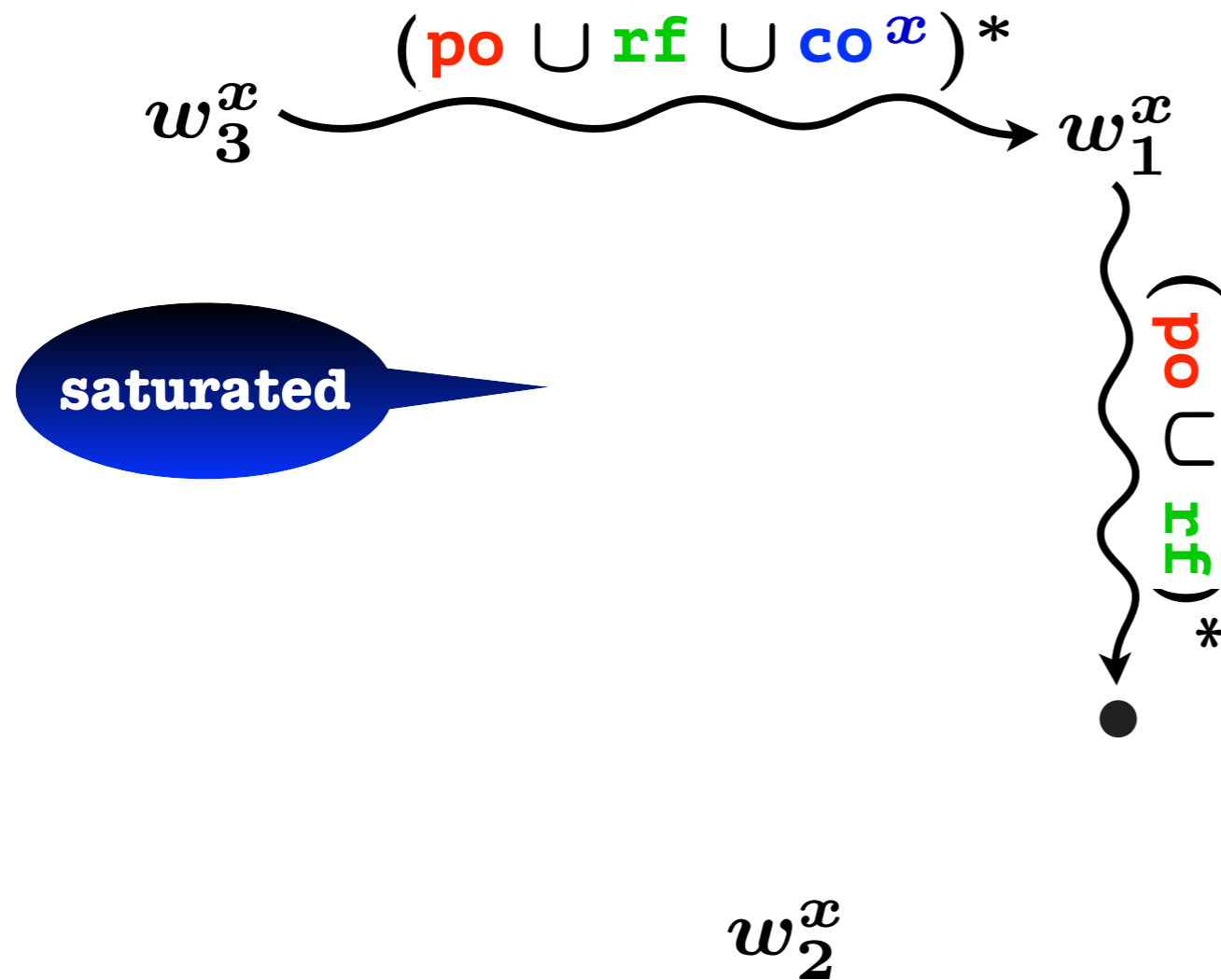


Adding Events to Traces

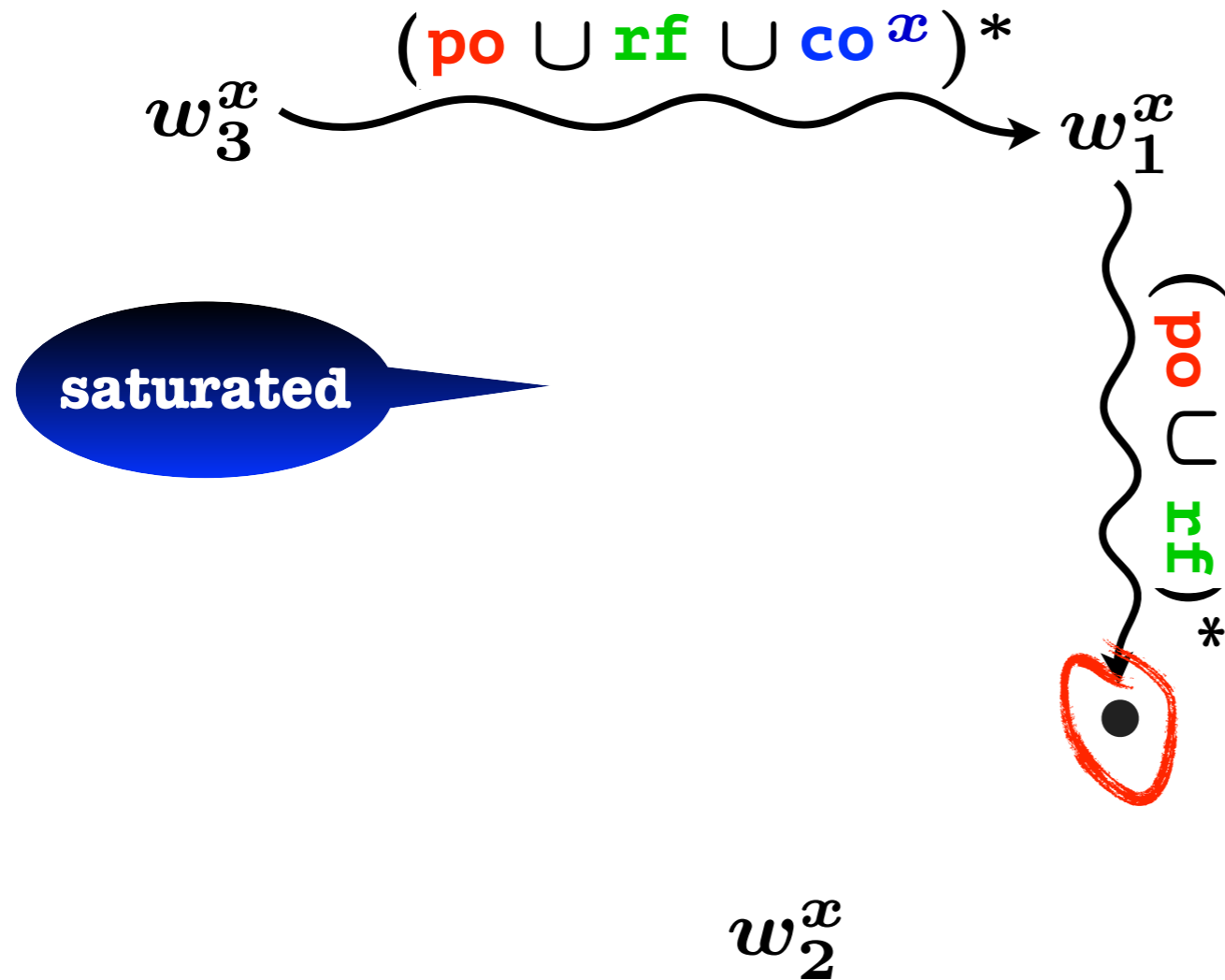
$$\tau \oplus e$$



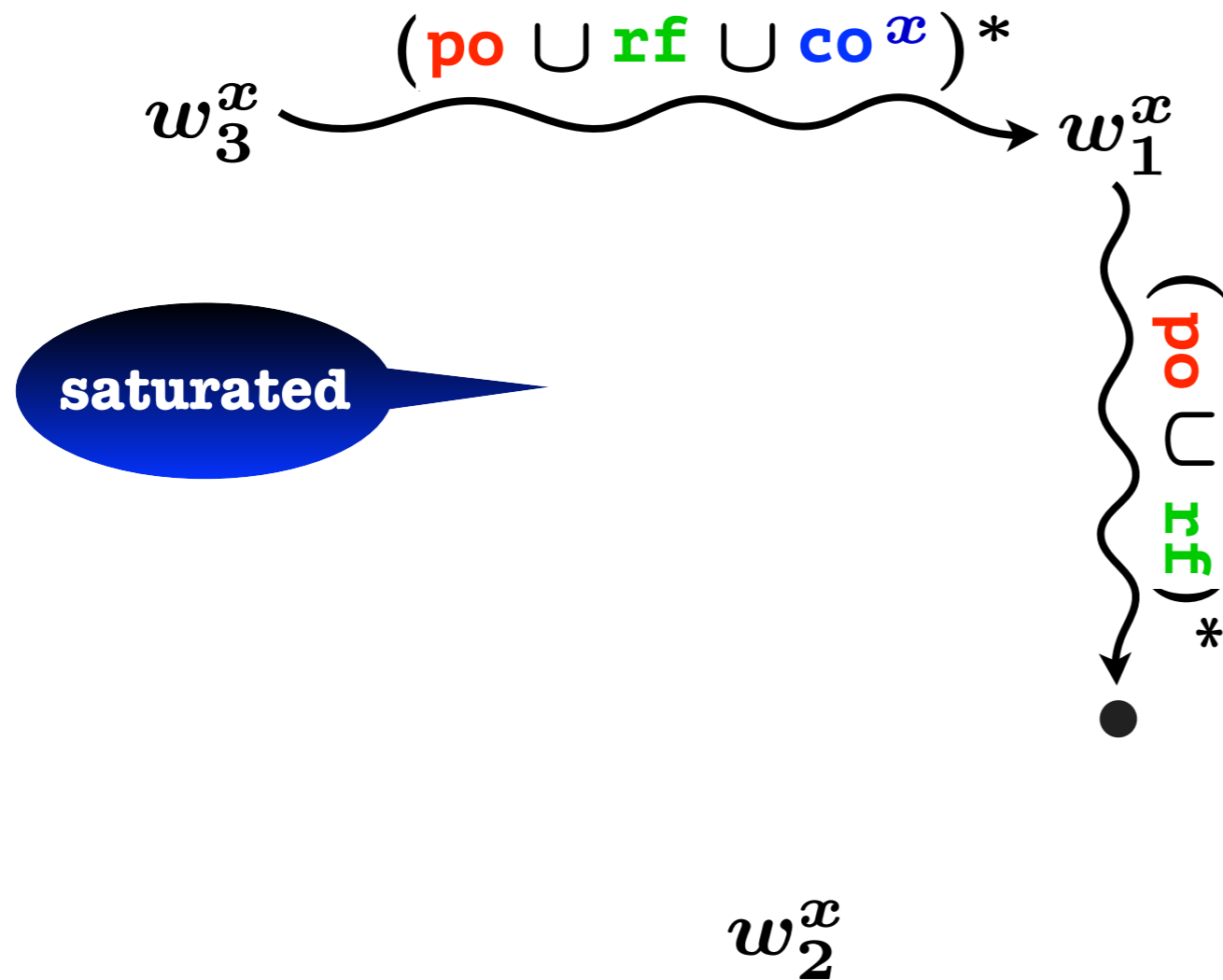
Adding Events to Traces



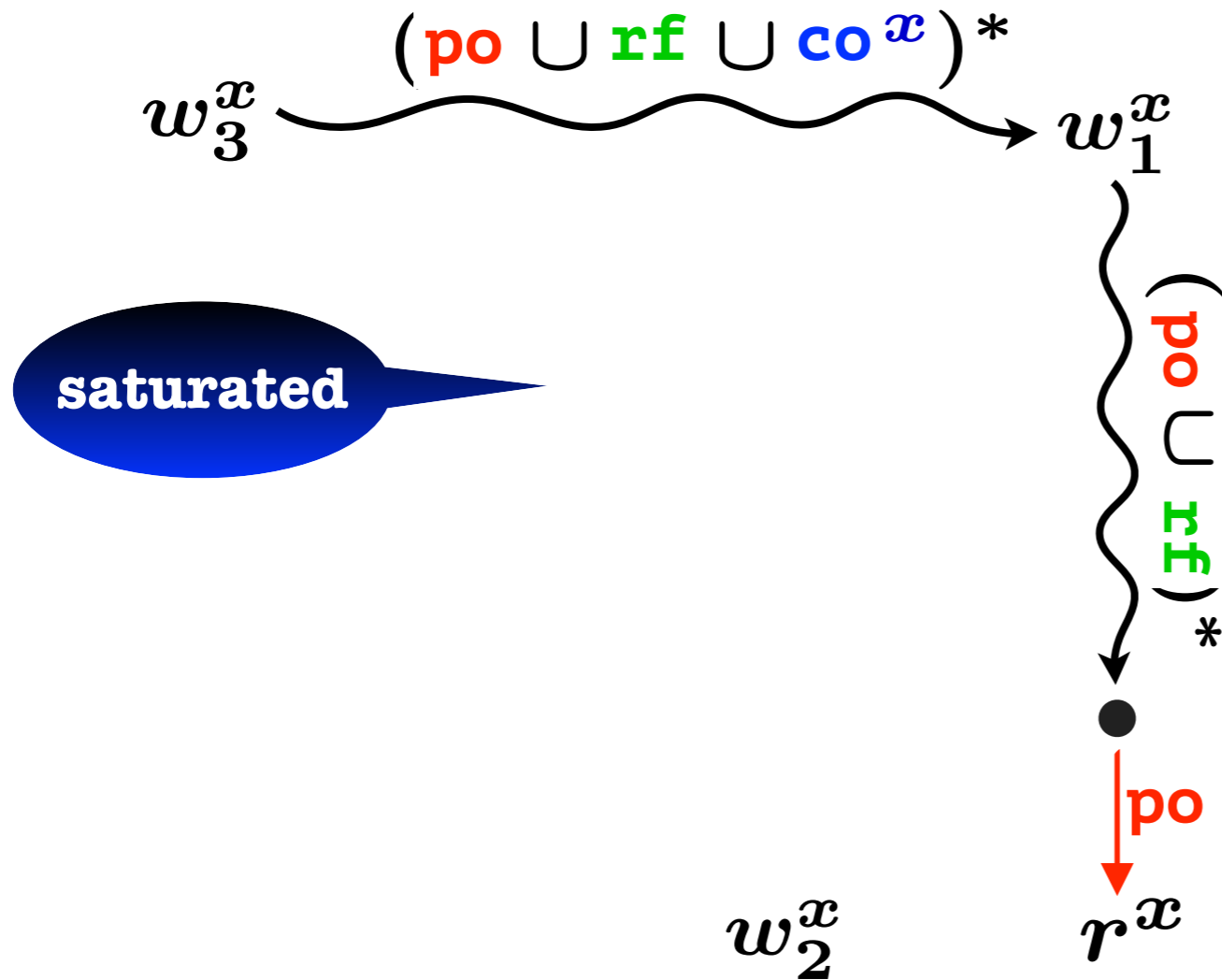
Adding Events to Traces



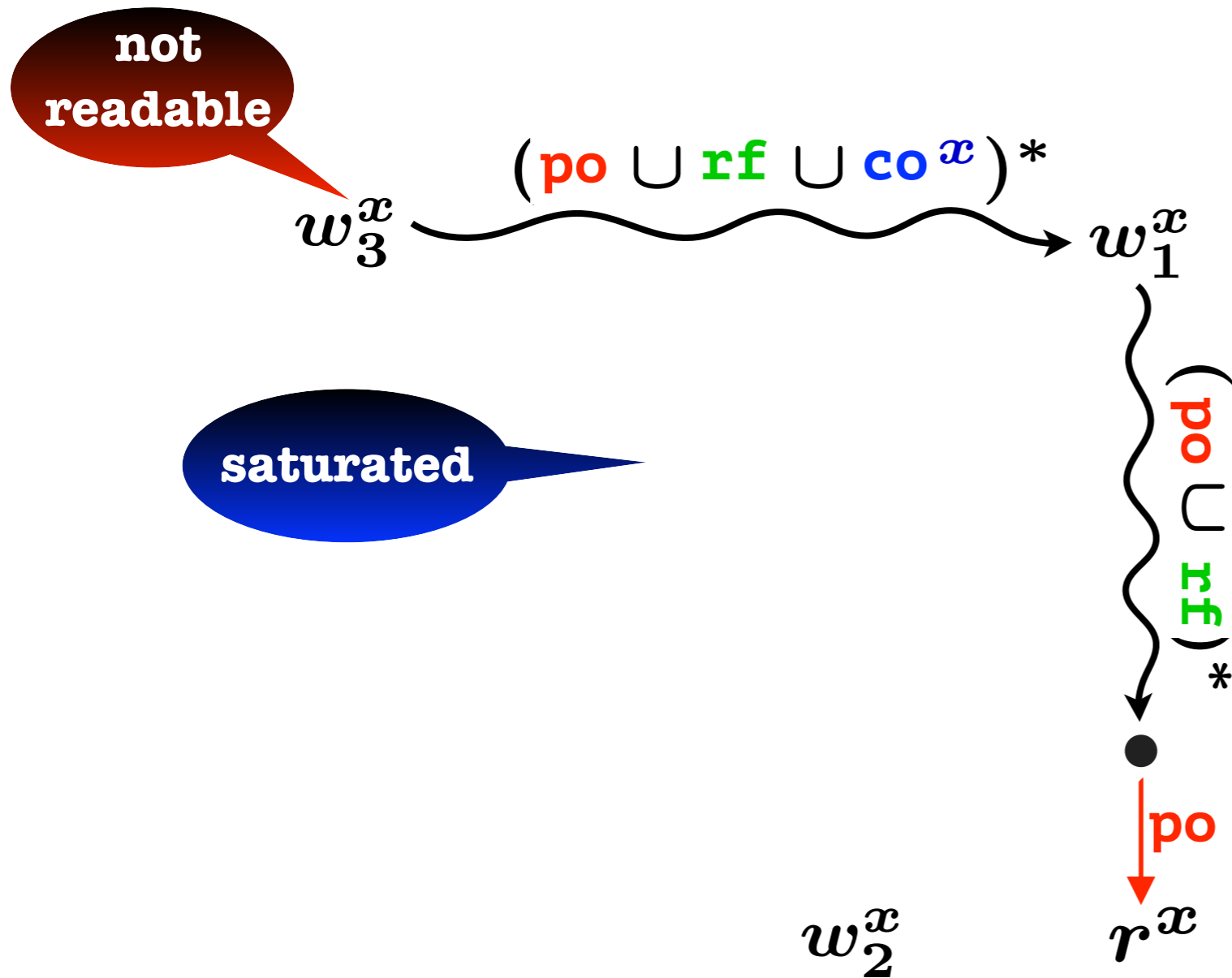
Adding Events to Traces



Adding Events to Traces



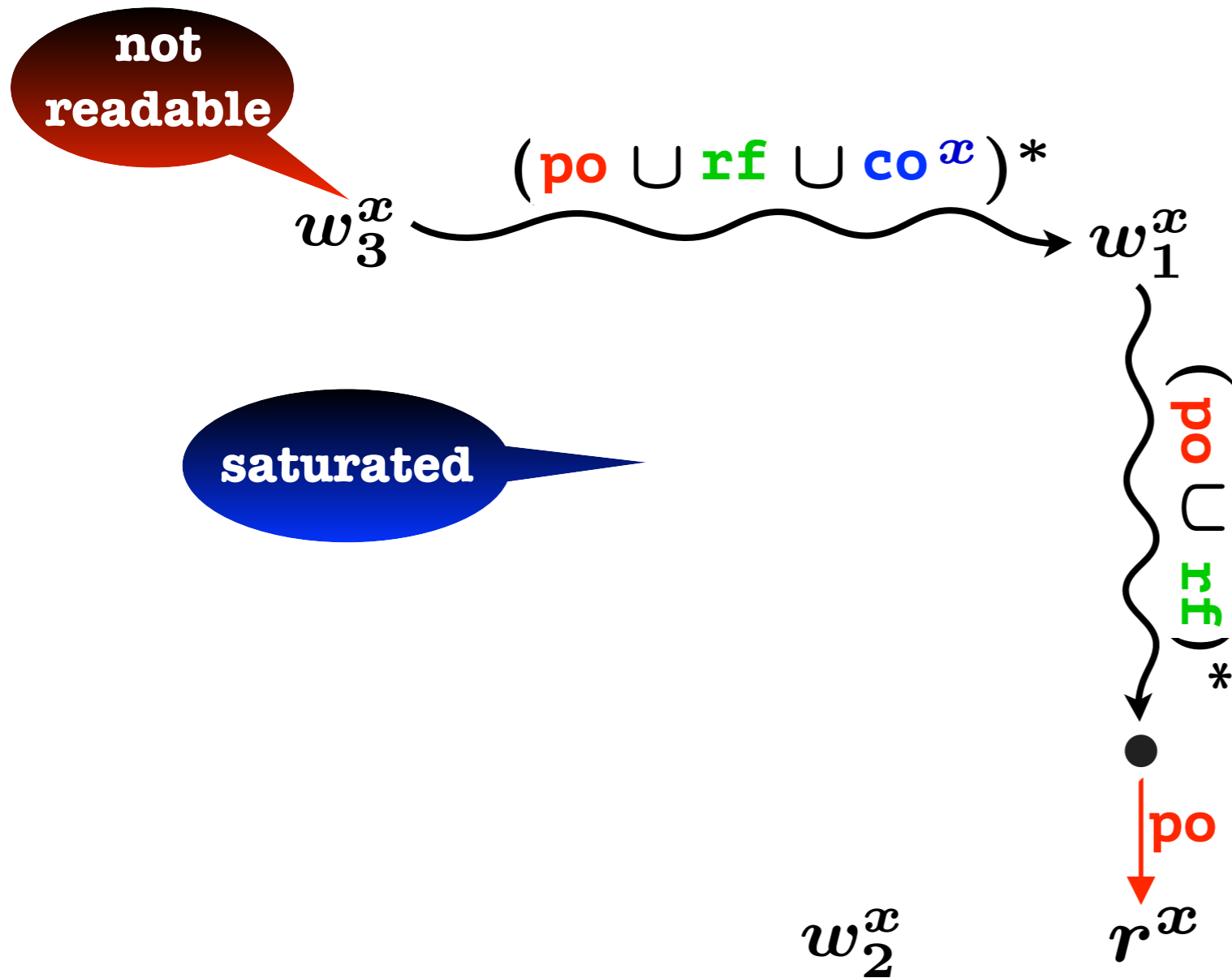
Adding Events to Traces



Adding Events to Traces



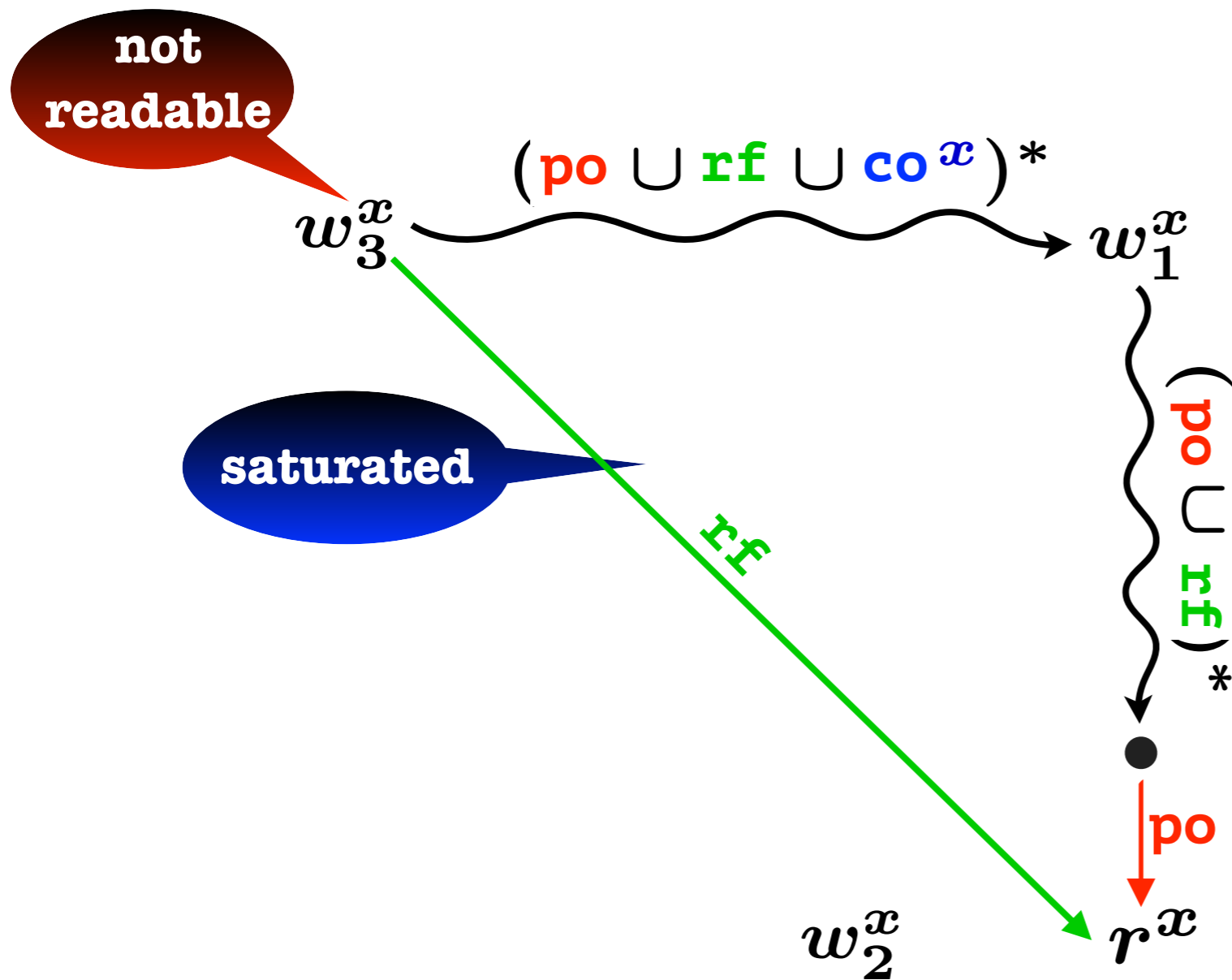
$$\text{RA: } \forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$$



Adding Events to Traces



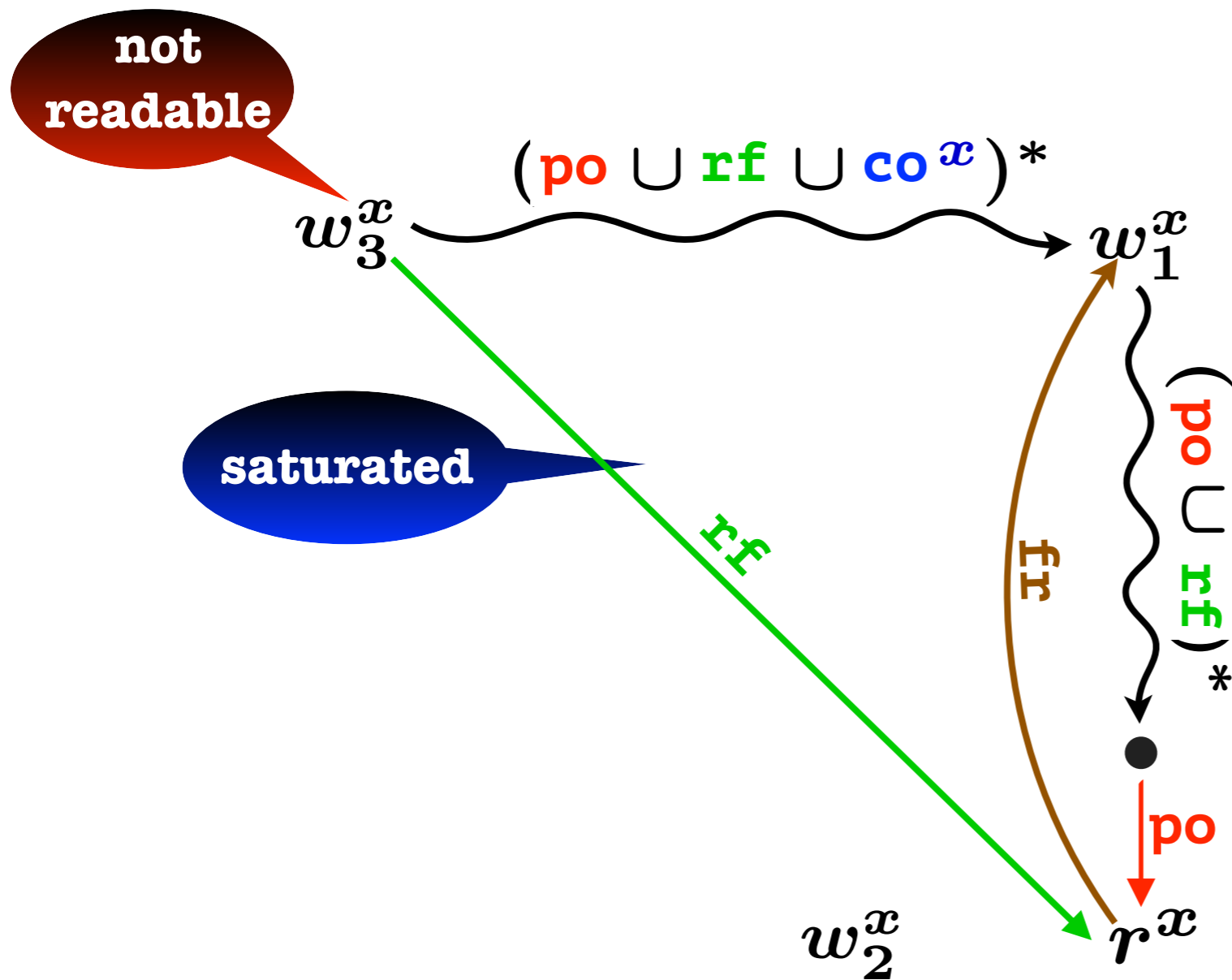
$$\text{RA: } \forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$$



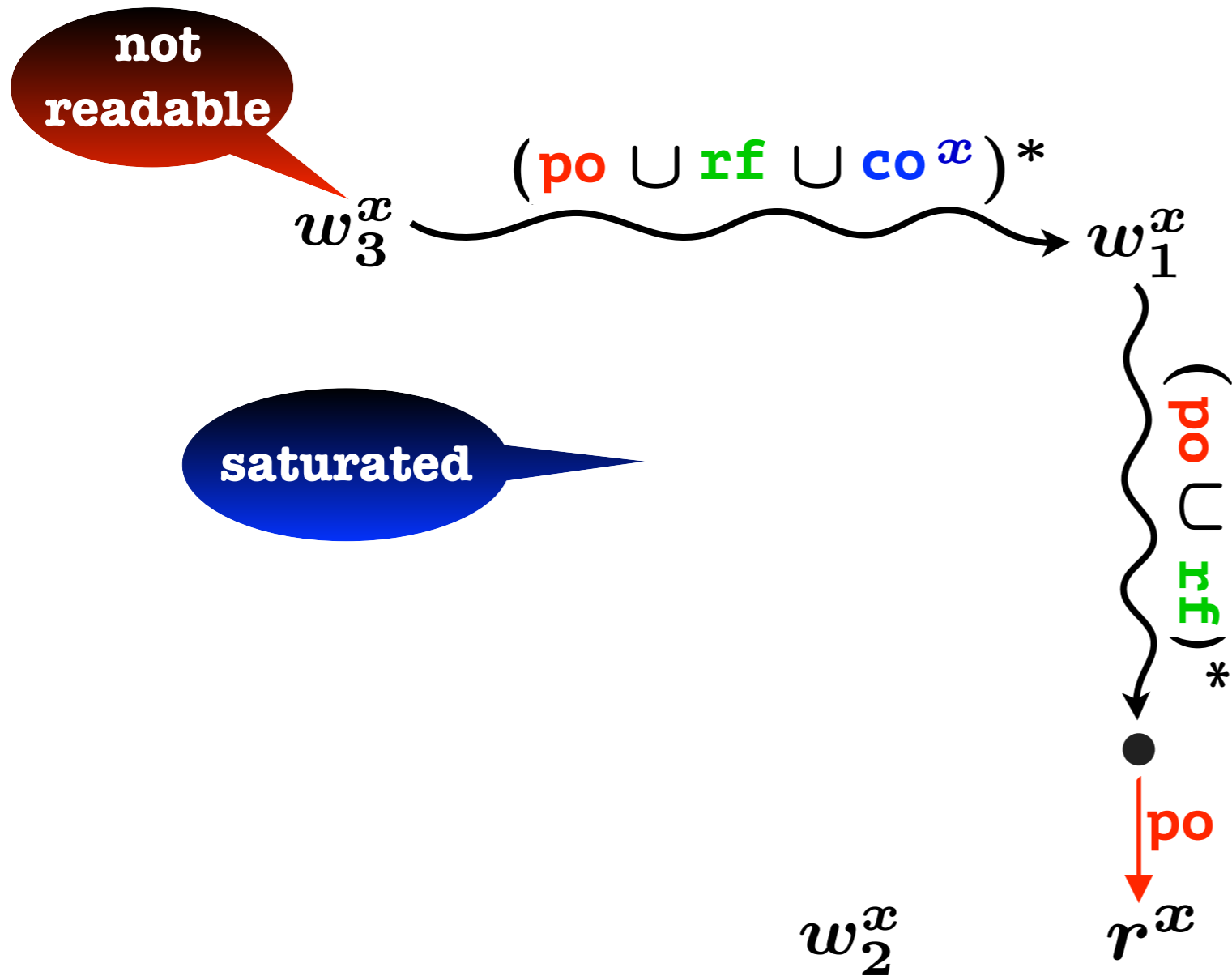
Adding Events to Traces



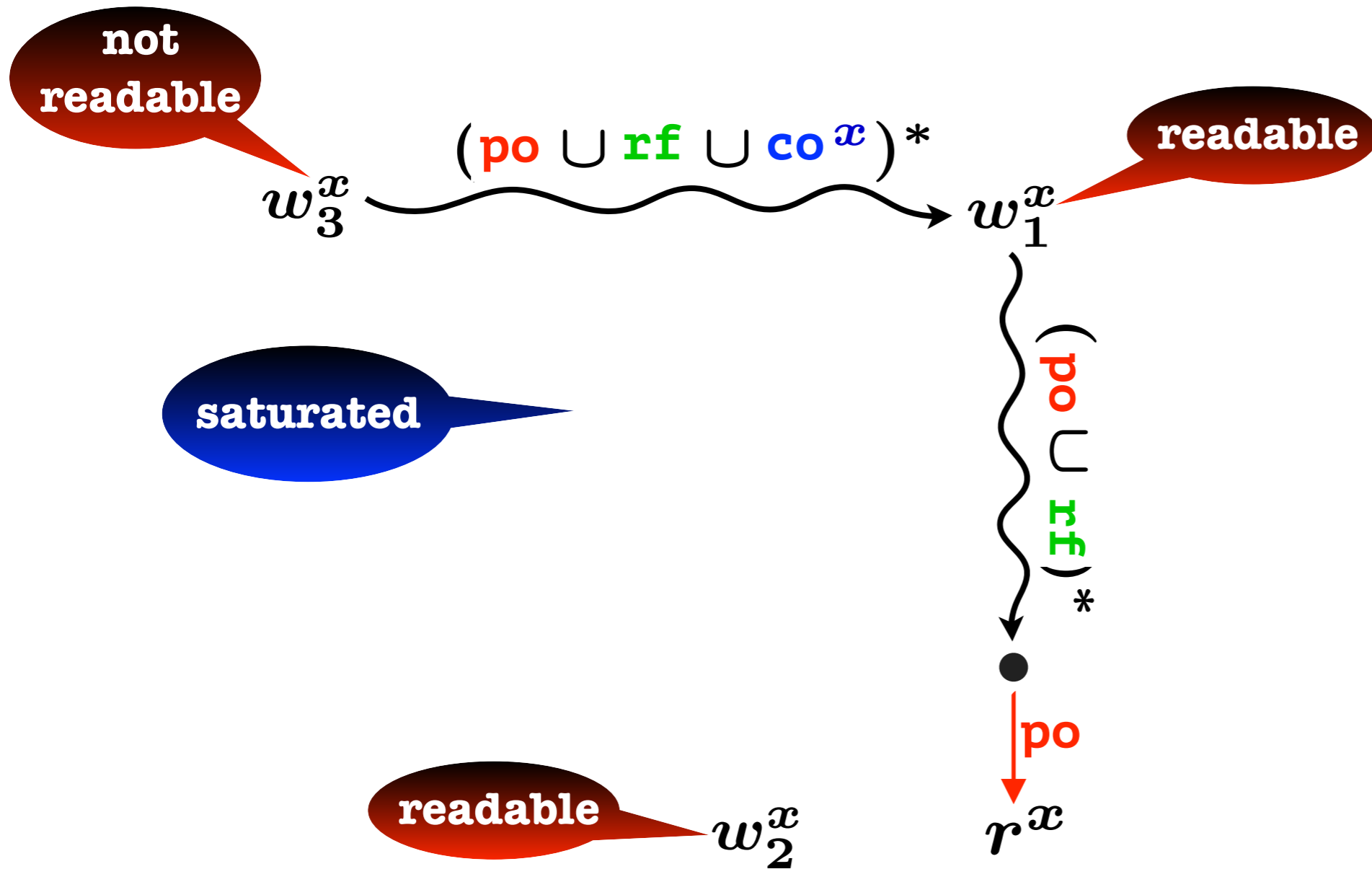
$$\text{RA: } \forall x. \text{acyclic}(\text{po} \cup \text{rf} \cup \text{co}^x \cup \text{fr}^x)$$



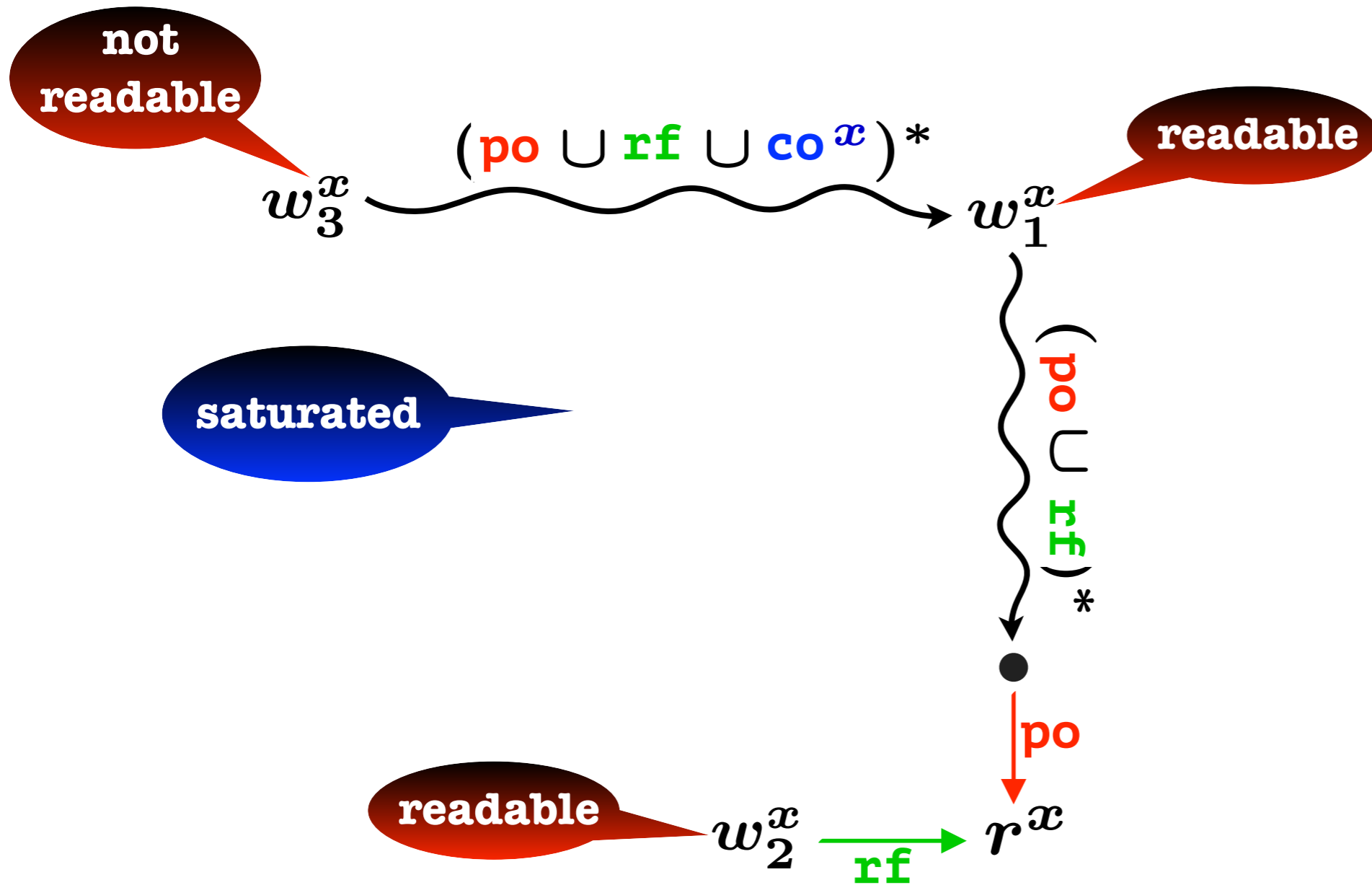
Adding Events to Traces



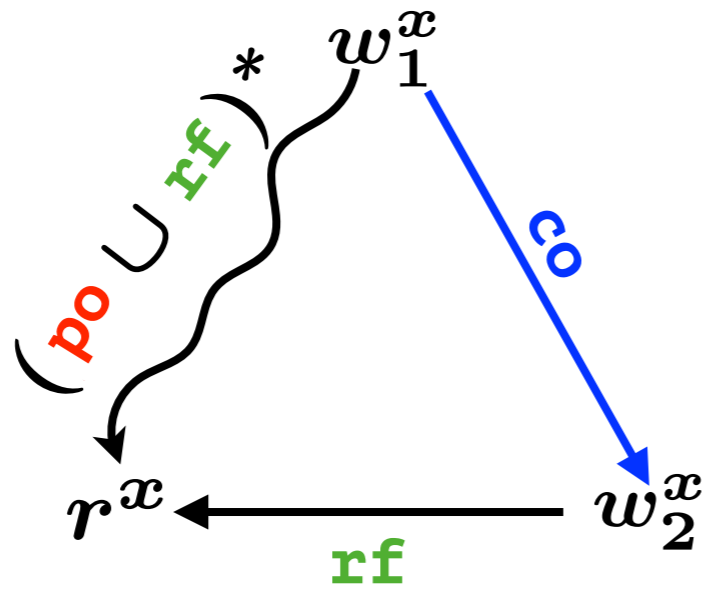
Adding Events to Traces



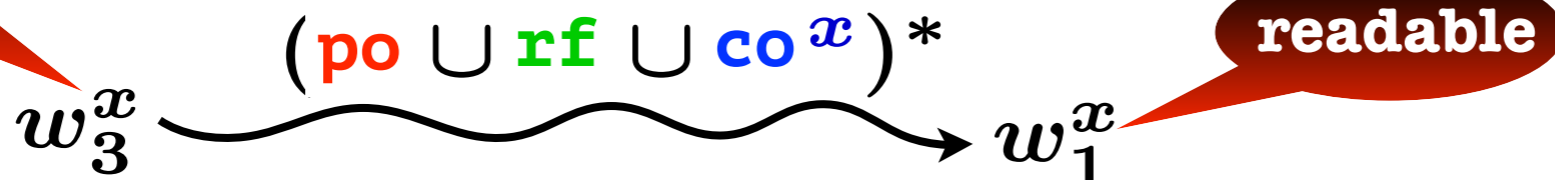
Adding Events to Traces



Adding Events to Traces



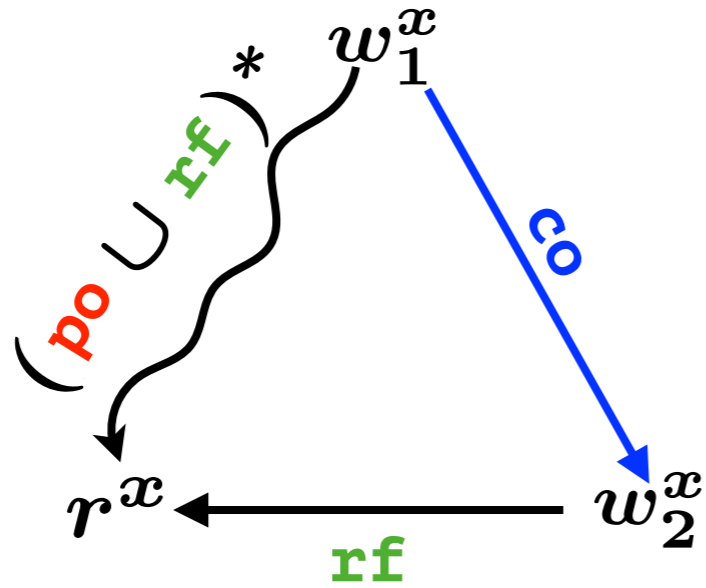
not readable



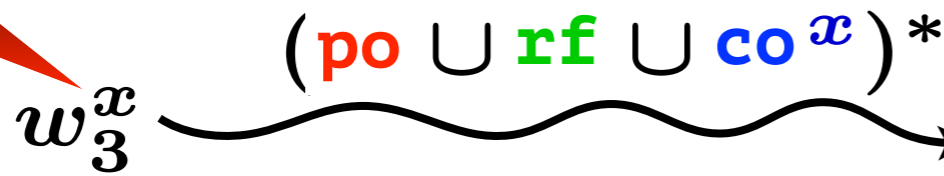
saturated



Adding Events to Traces



not readable



readable

visible

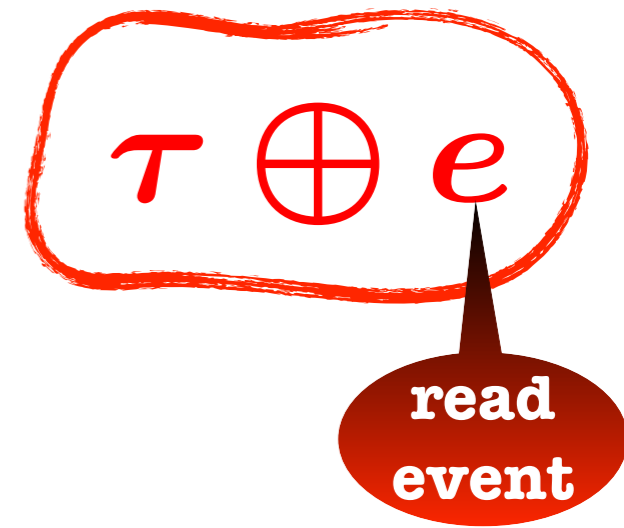
saturated



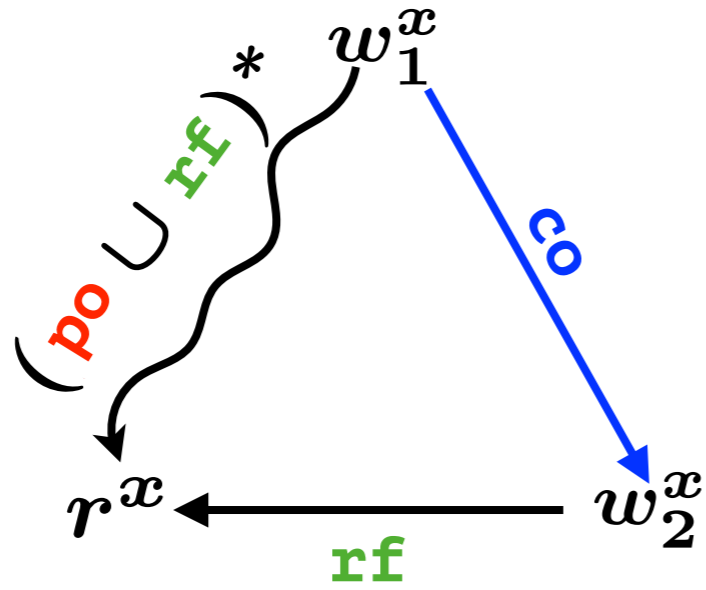
readable



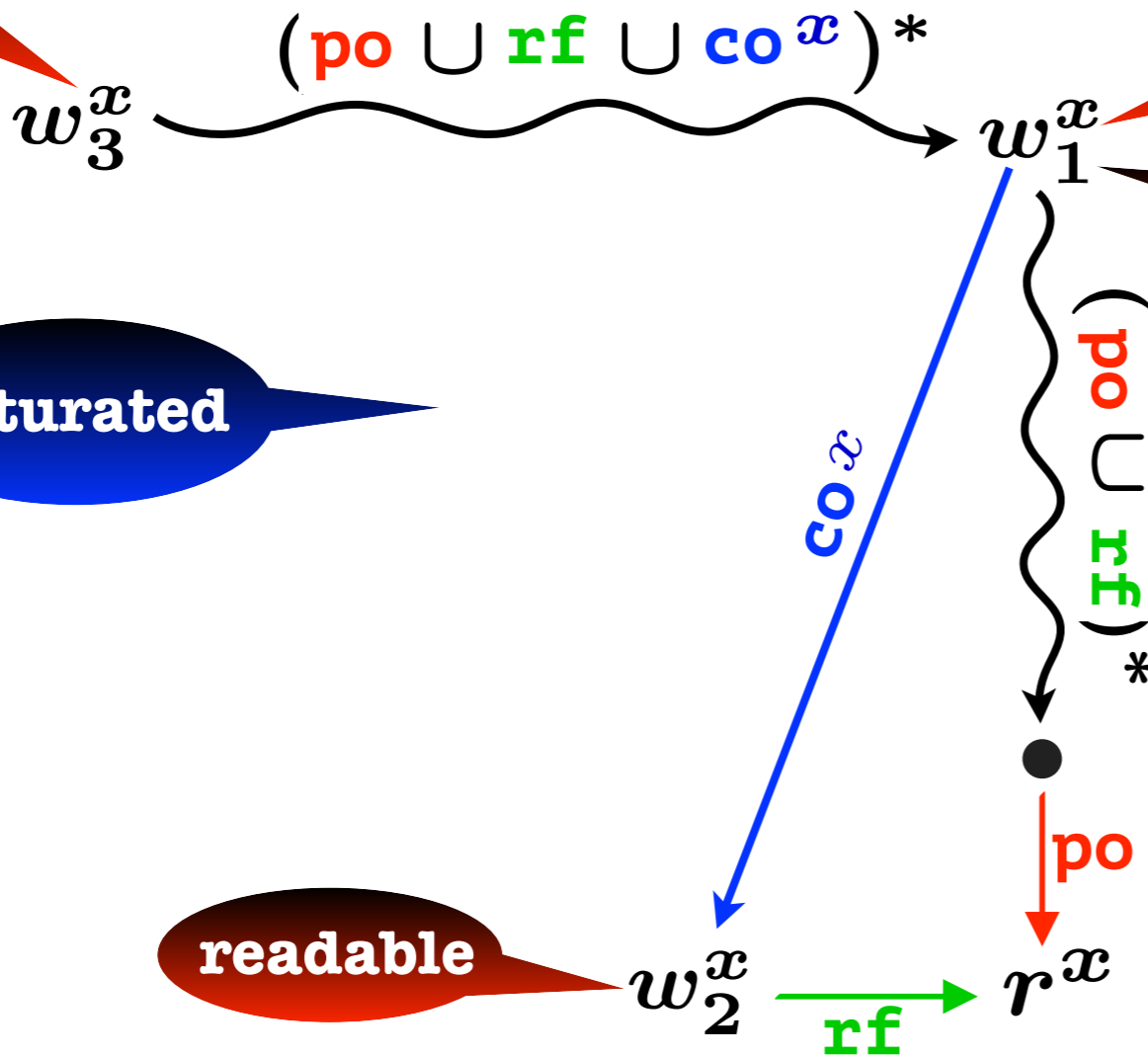
Adding Events to Traces



not readable



saturated

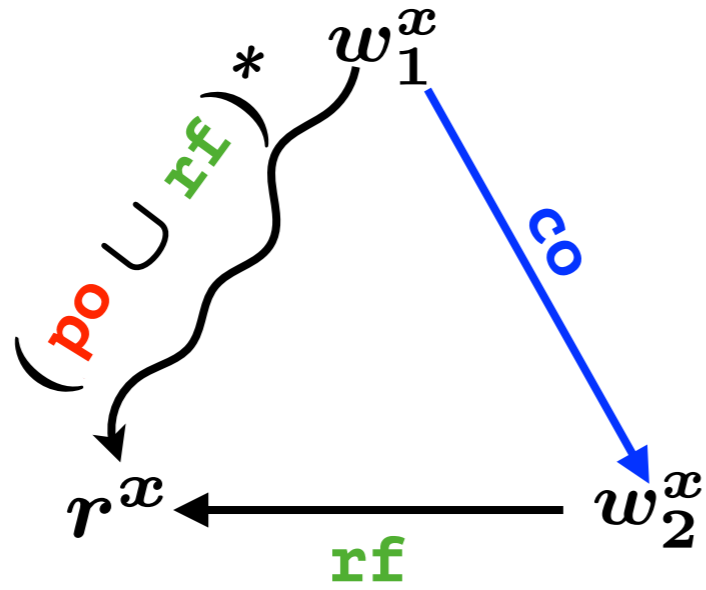
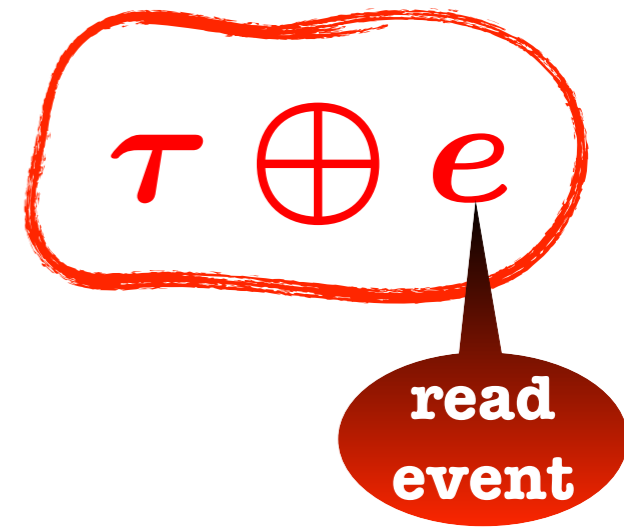


readable

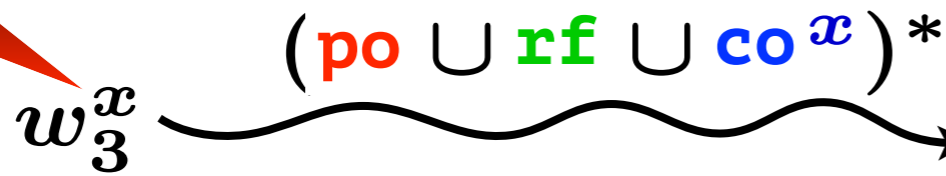
visible

readable

Adding Events to Traces



not readable



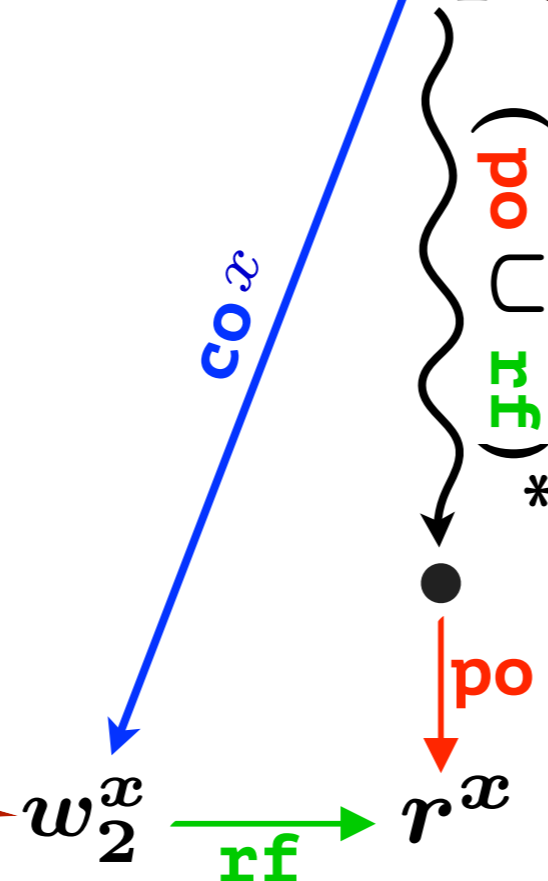
readable

visible

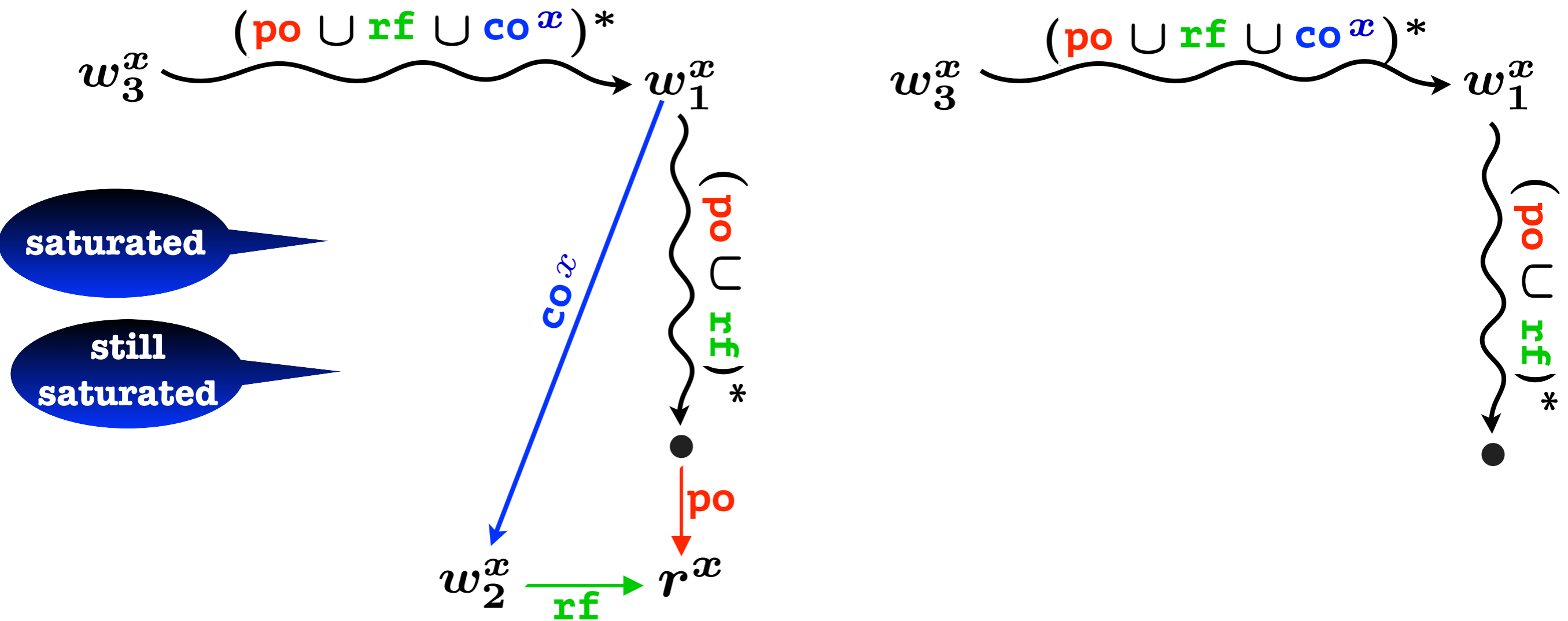
saturated

still saturated

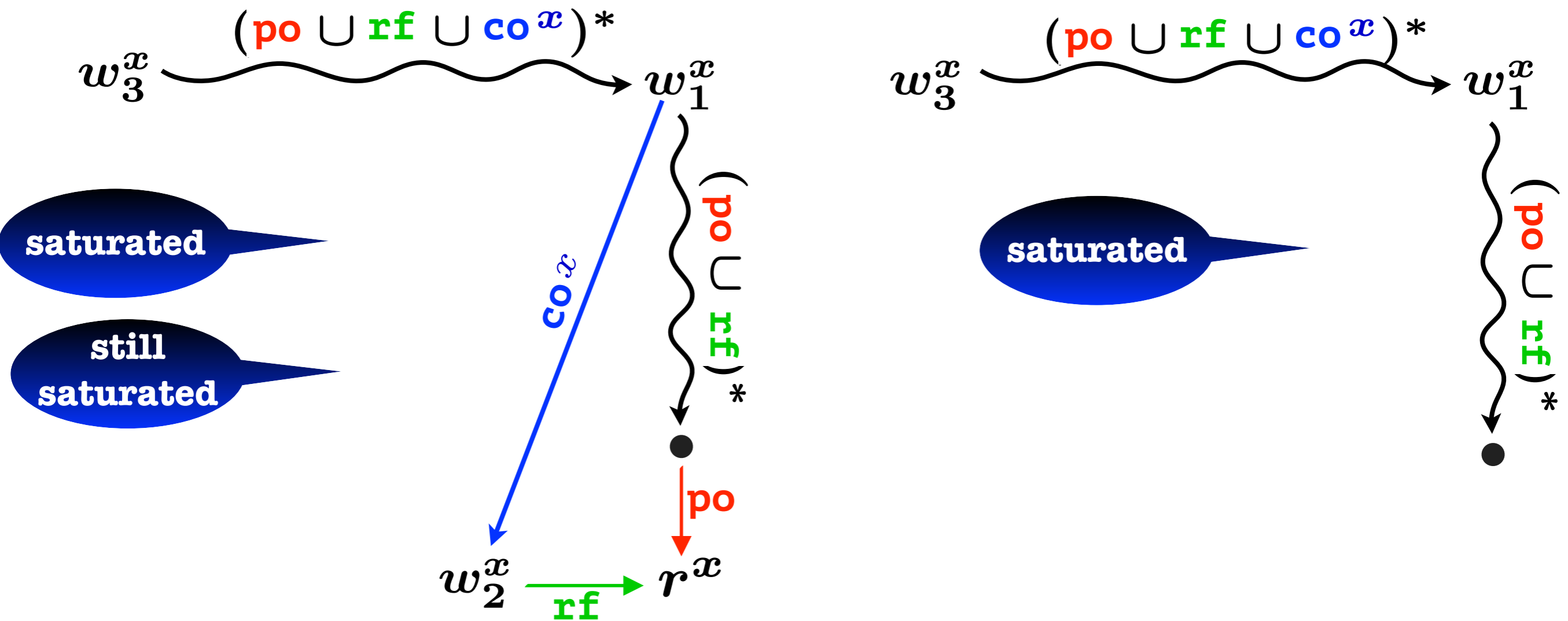
readable



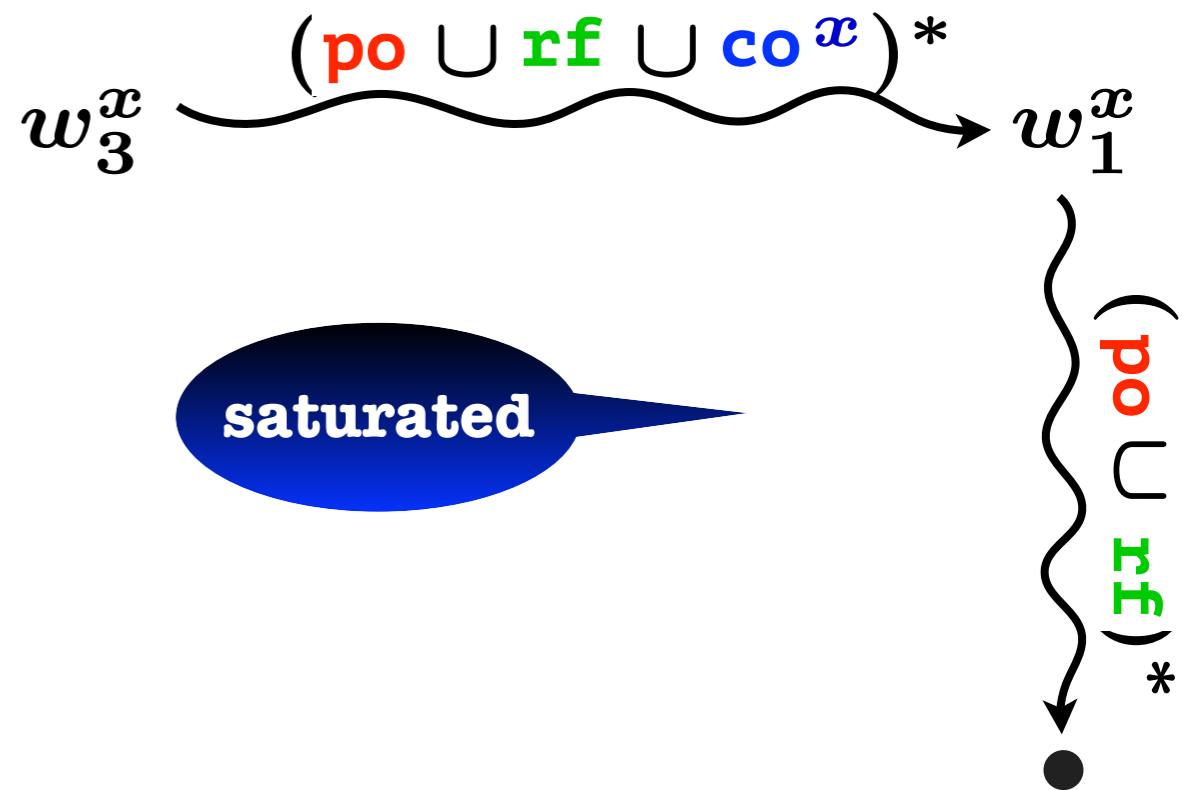
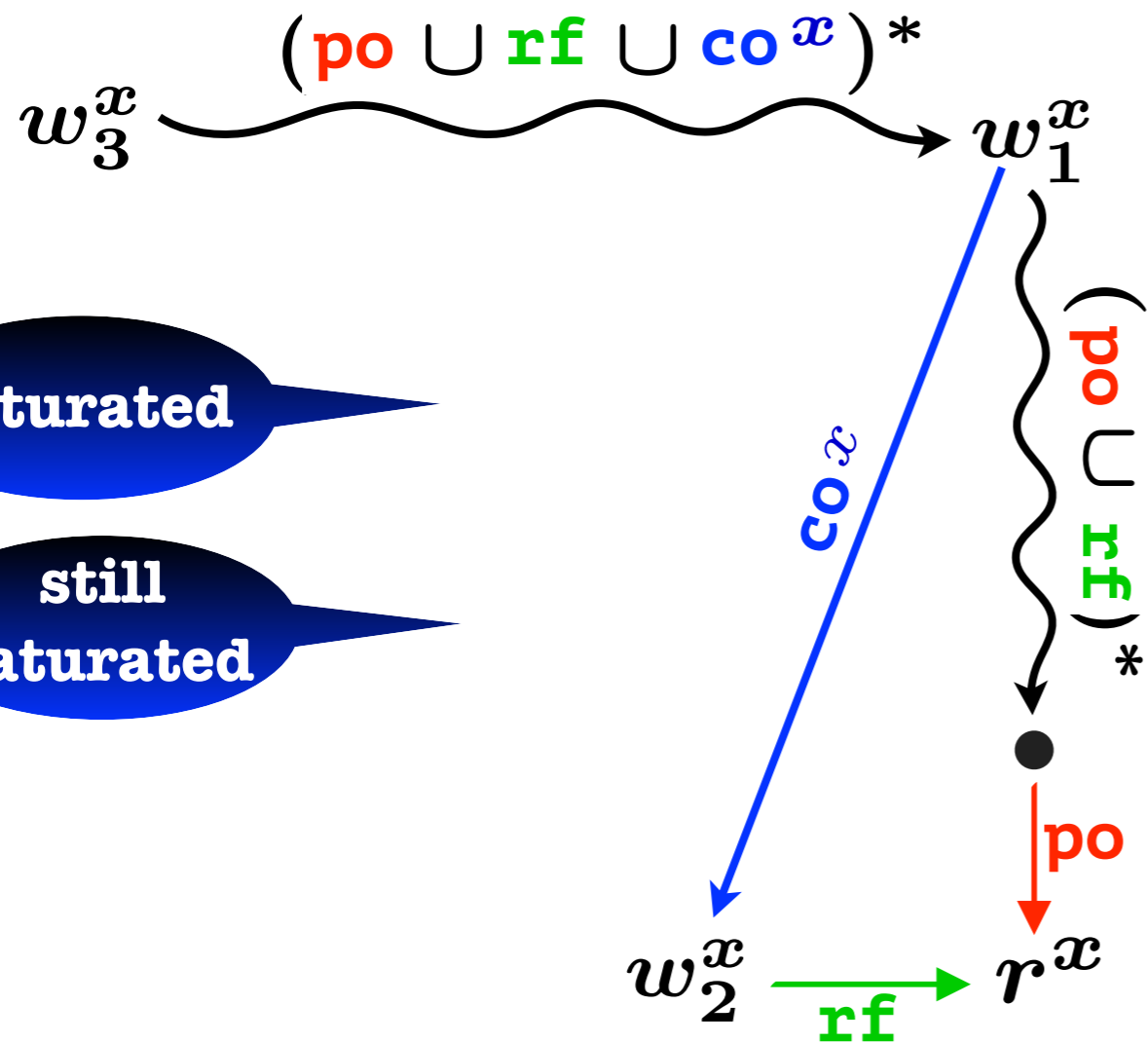
Adding Events to Traces



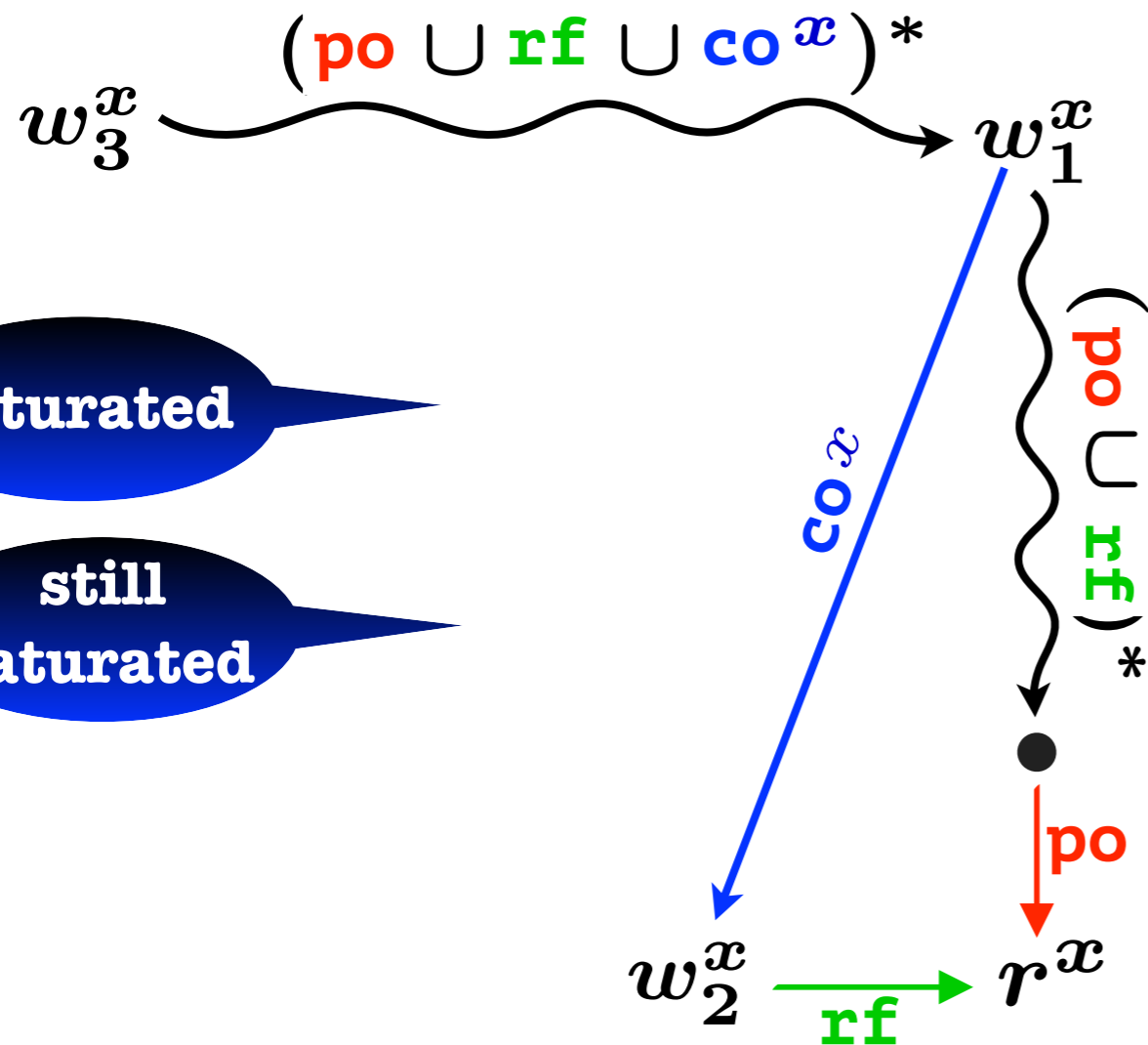
Adding Events to Traces



Adding Events to Traces

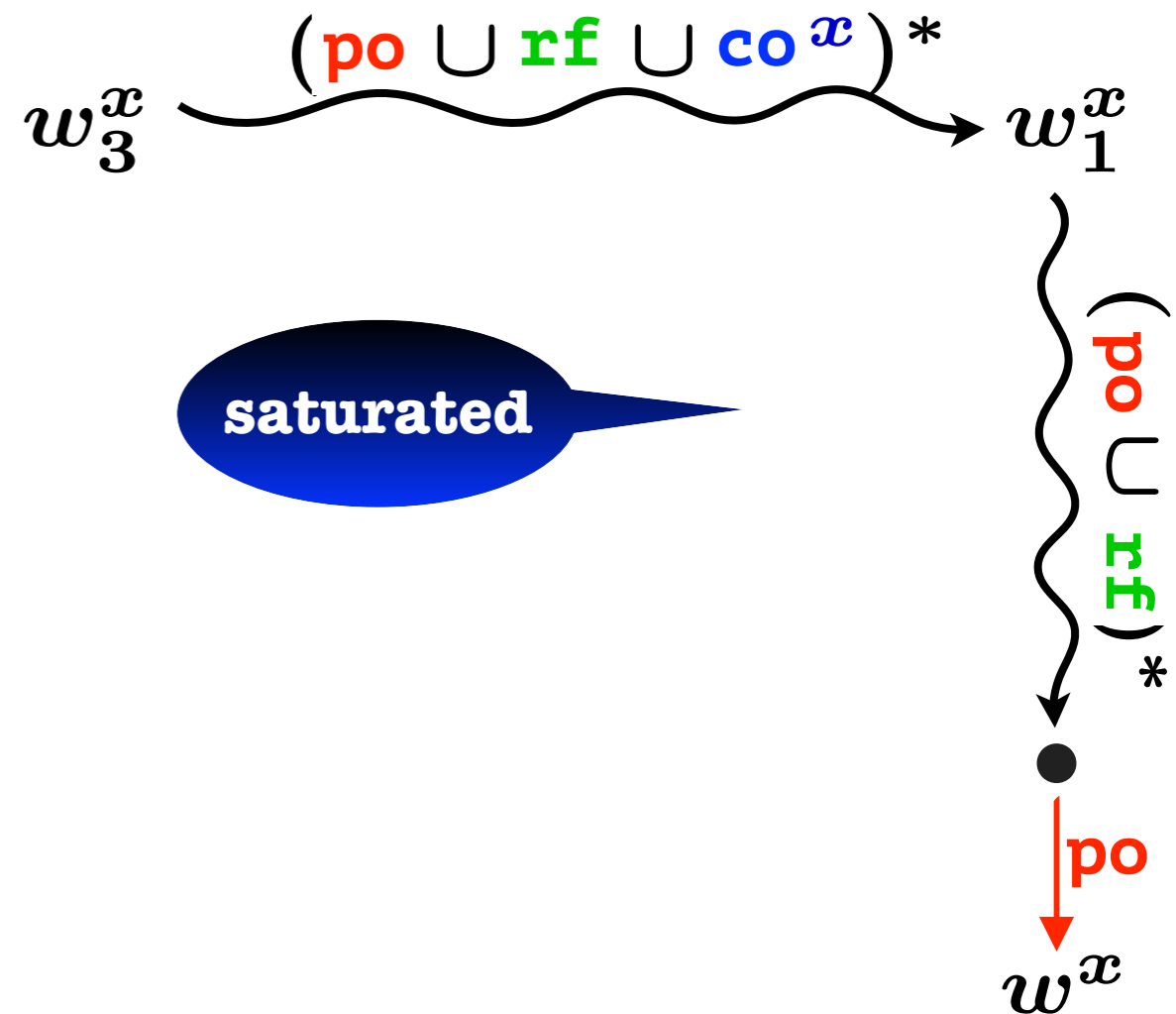


Adding Events to Traces



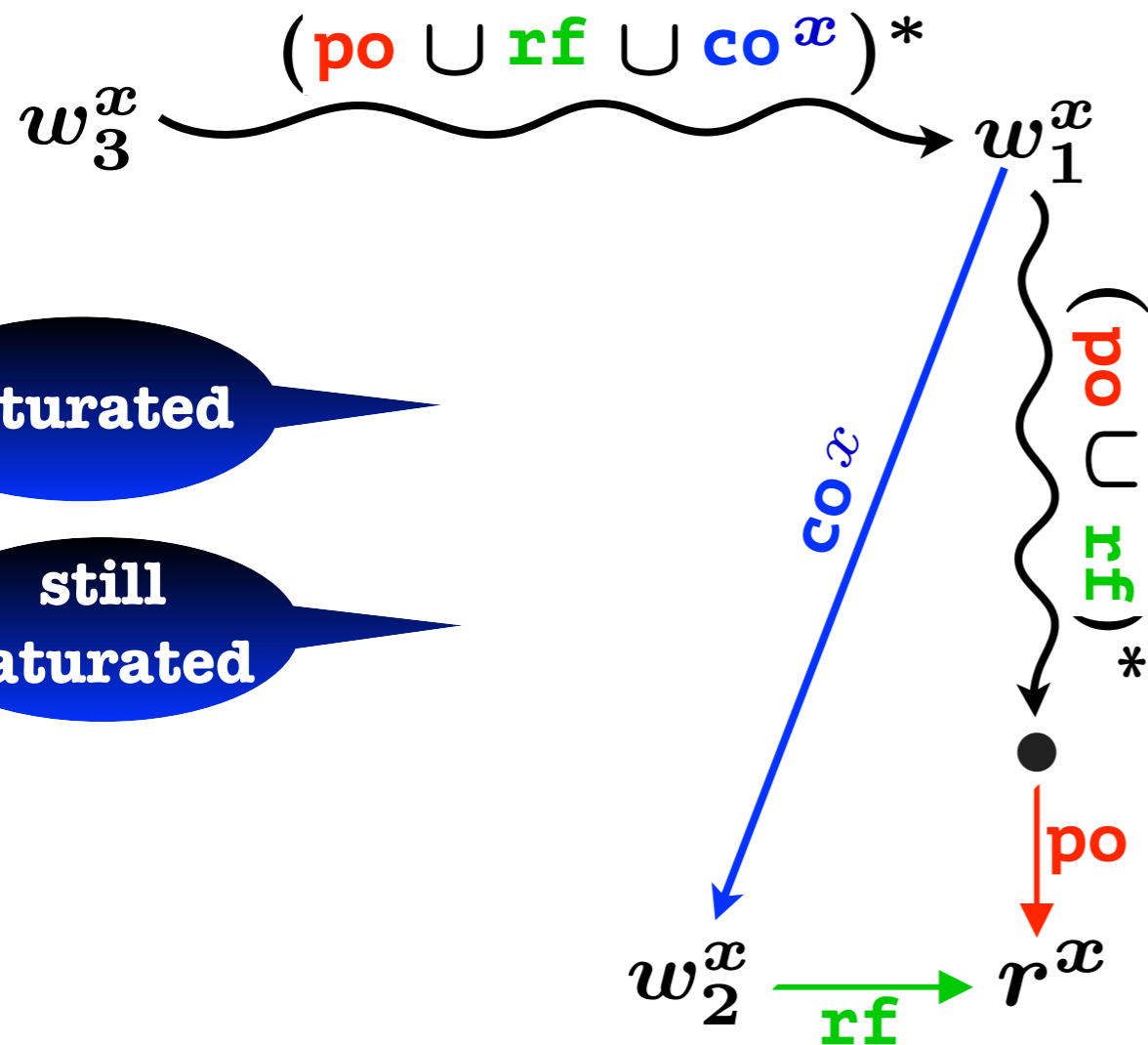
saturated

still saturated



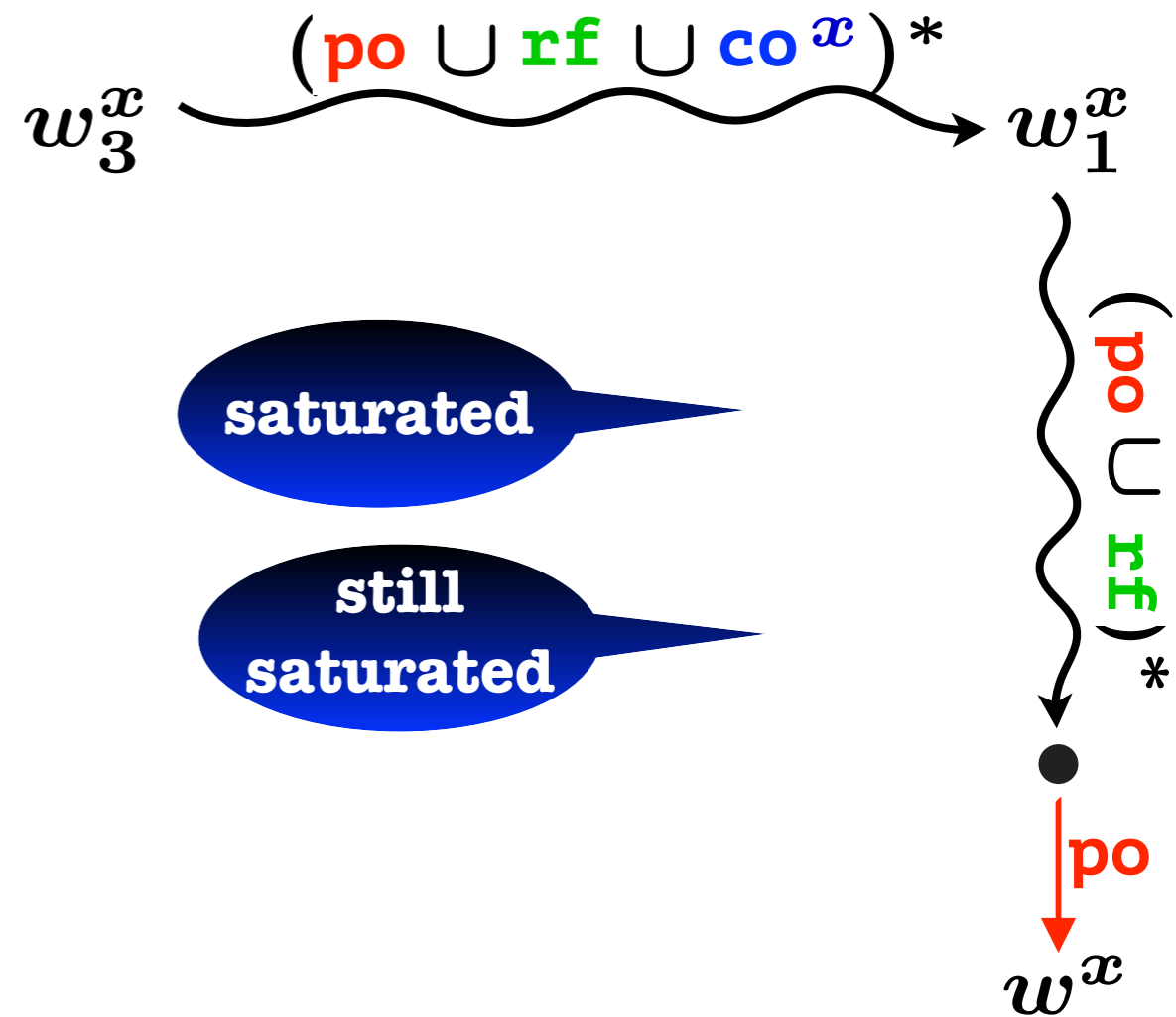
saturated

Adding Events to Traces



saturated

still saturated



saturated

still saturated

DPOR under the RA-semantics

- Build **traces** one after one
- For a given trace \mathcal{T} :
 - Build \mathcal{T} incrementally
 - extend \mathcal{T} by one event e at a time

Branch

&

Postpone

DPOR under the RA-semantics

- Build **traces** one after one
- For a given trace \mathcal{T} :
 - Build \mathcal{T} incrementally
 - extend \mathcal{T} by one event e at a time

Branch

&

Postpone

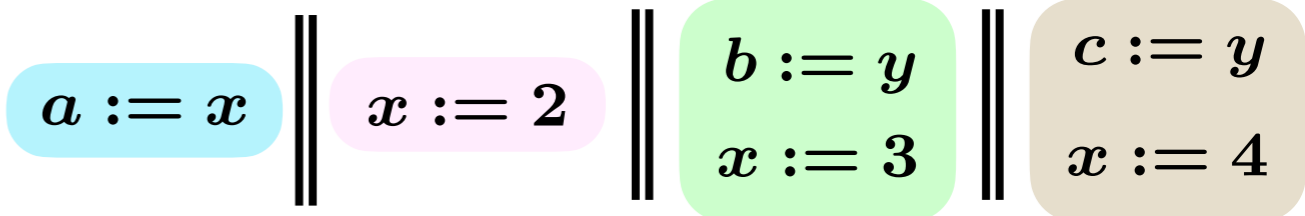
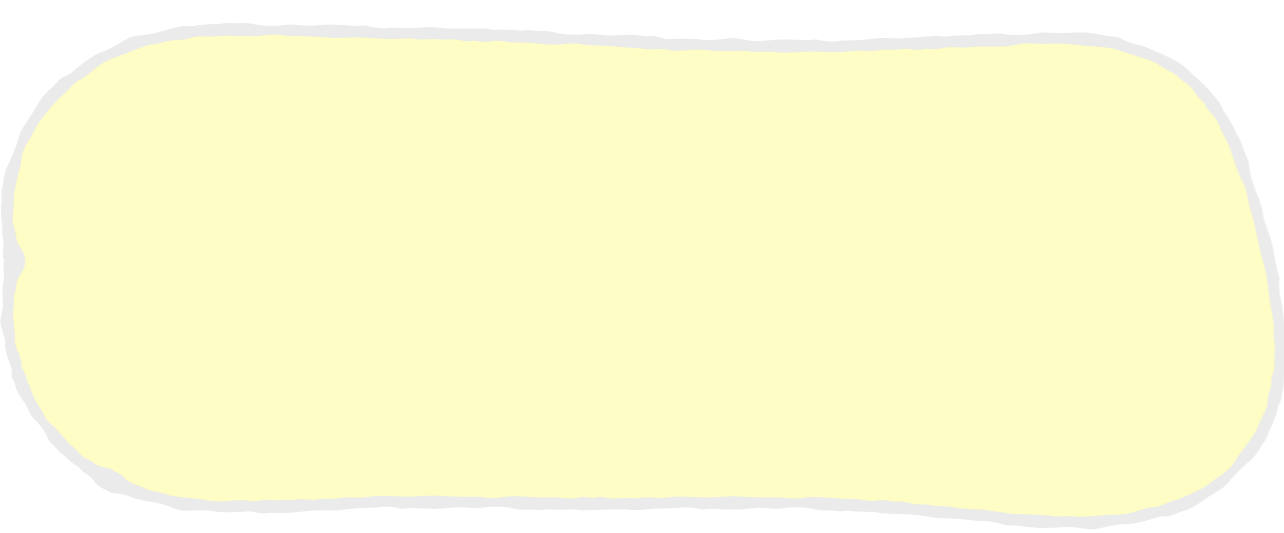
DPOR under the RA-semantics

- Build **traces** one after one
- For a given trace τ :
 - Build τ incrementally
 - extend τ by one event e at a time

Branch

&

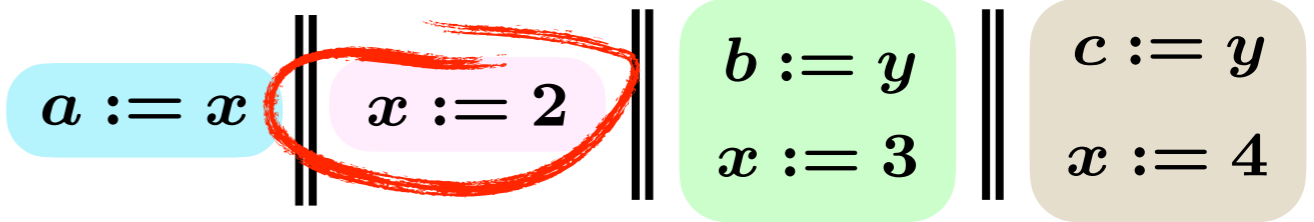
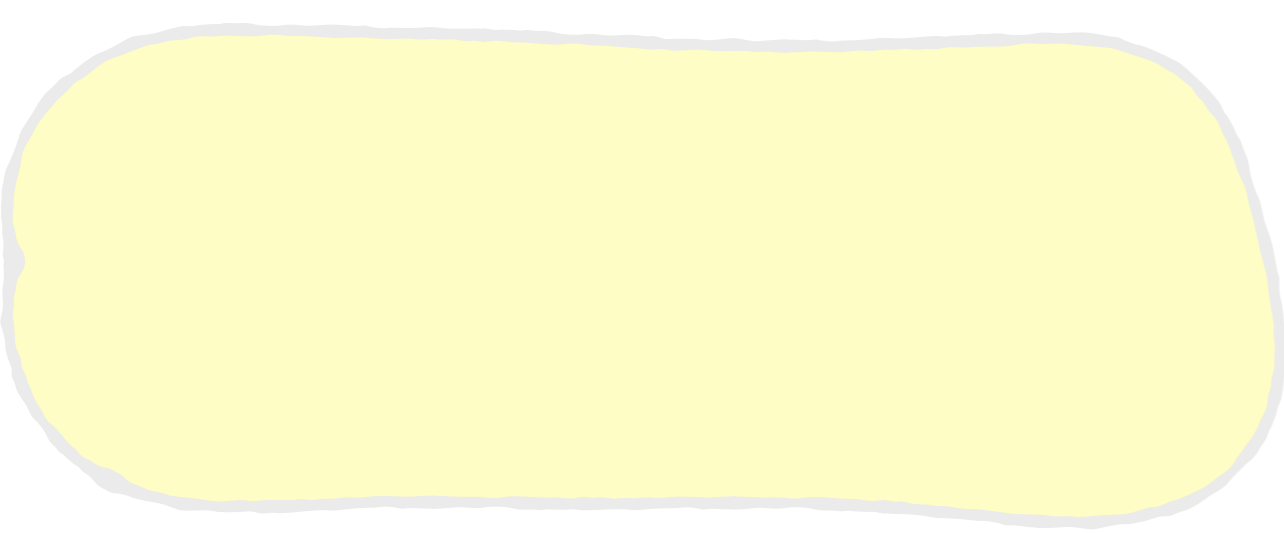
Postpone



Branch

&

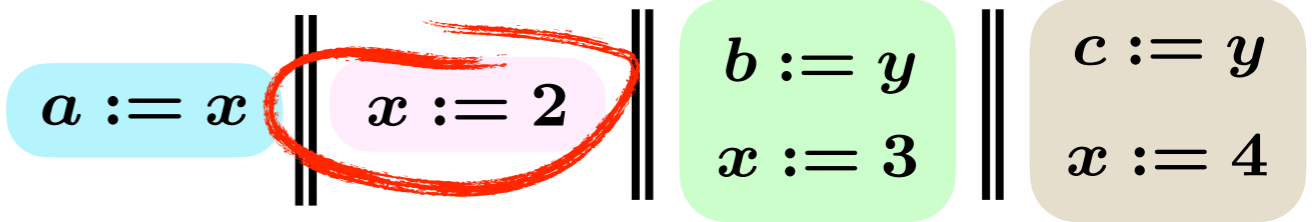
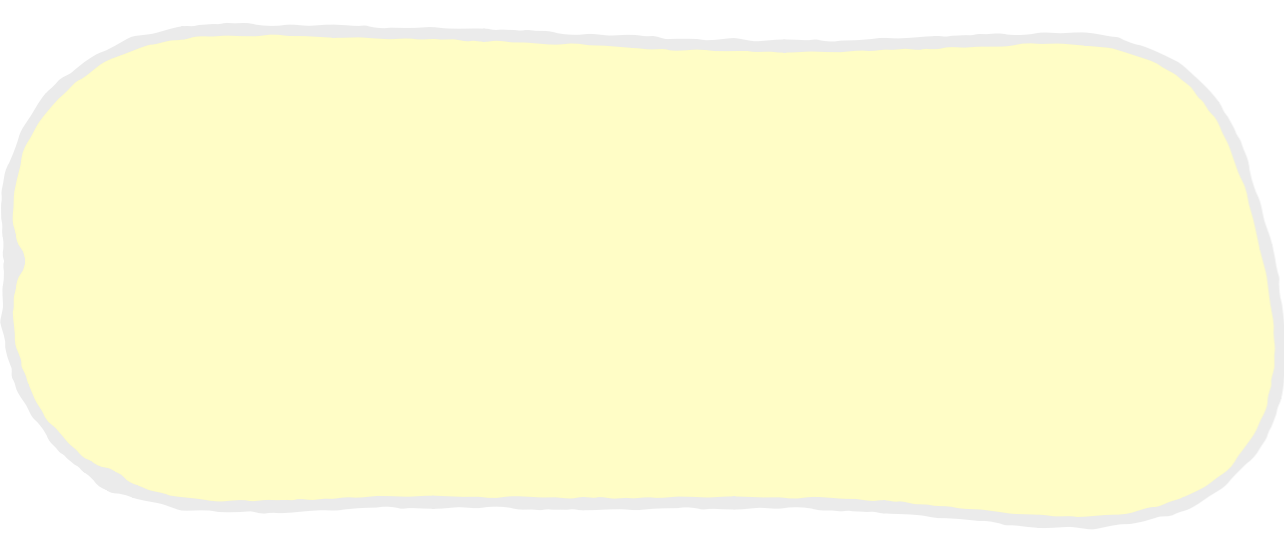
Postpone



Branch

&

Postpone

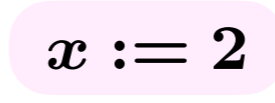
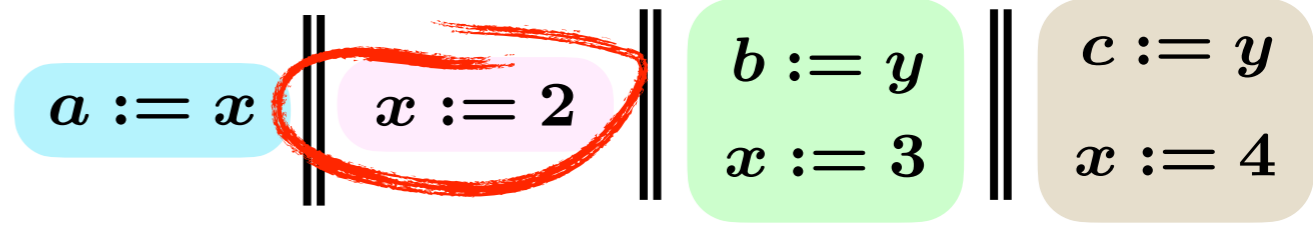
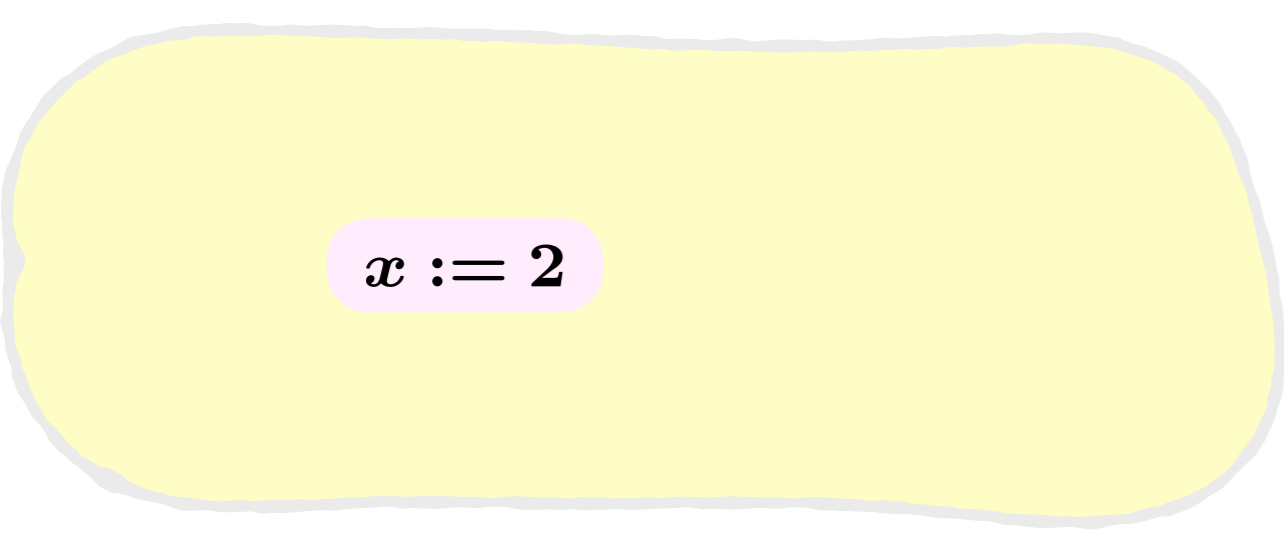


$x := 2$

Branch

&

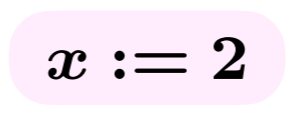
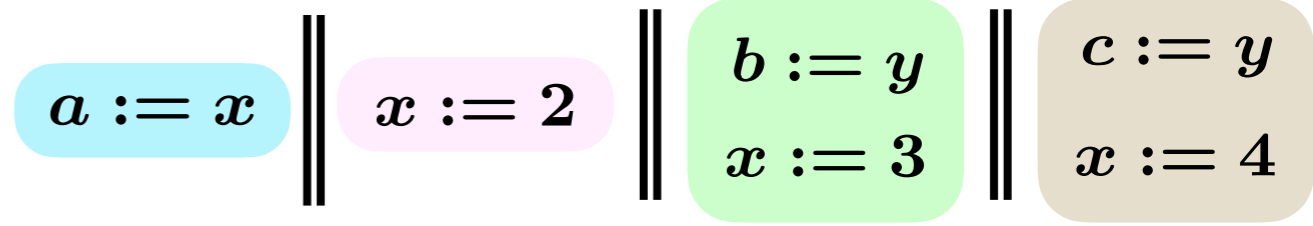
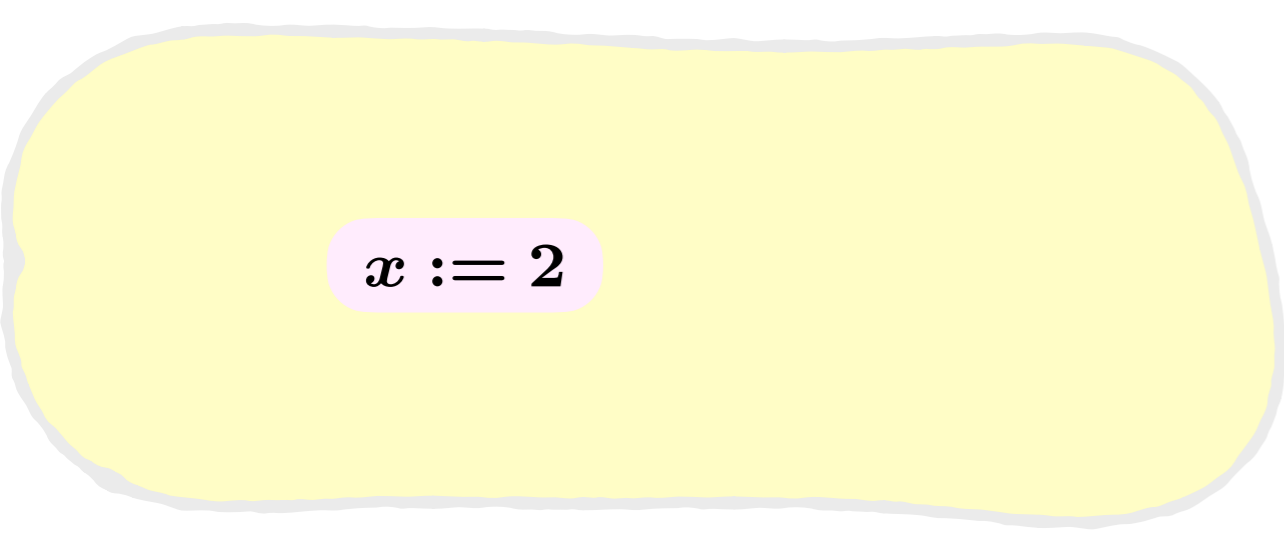
Postpone



Branch

&

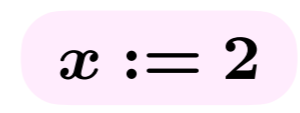
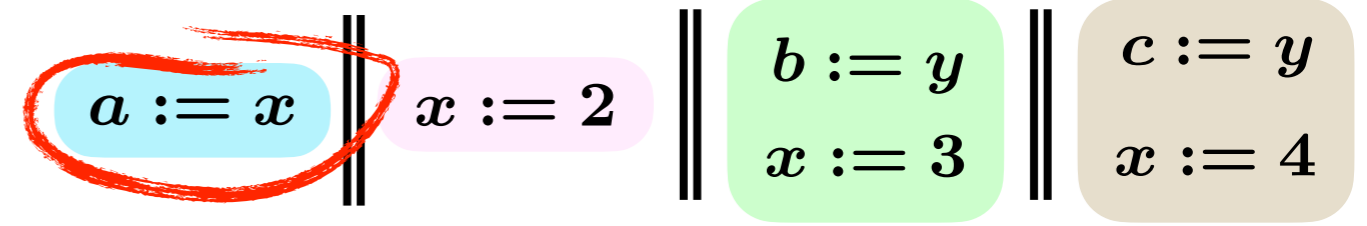
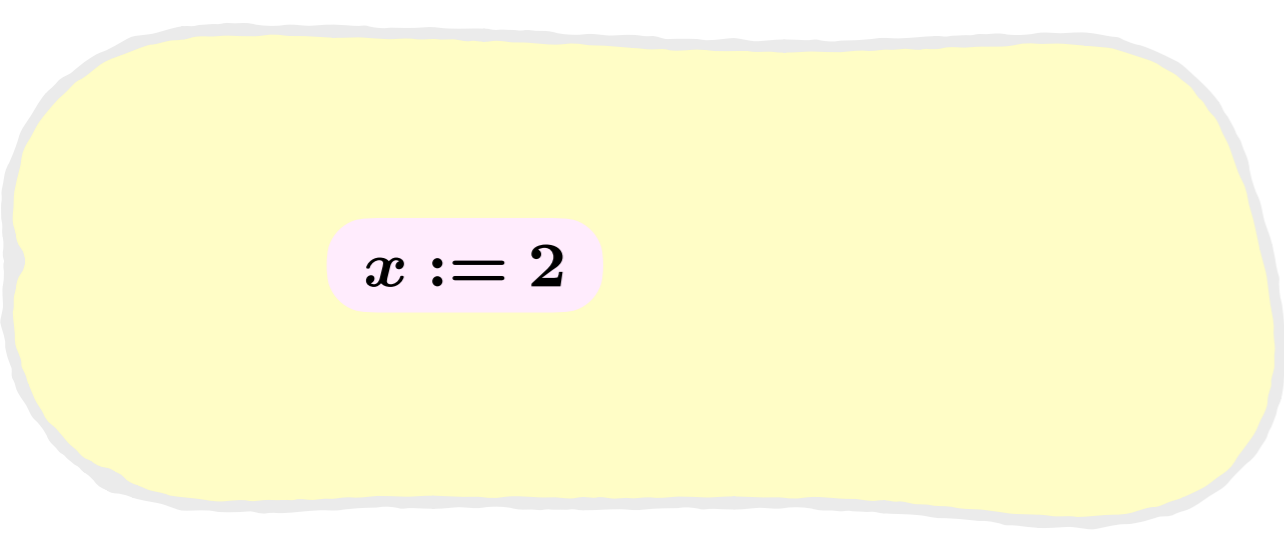
Postpone



Branch

&

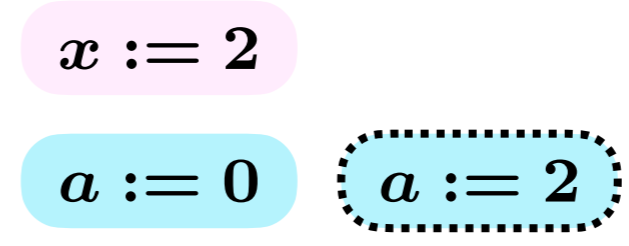
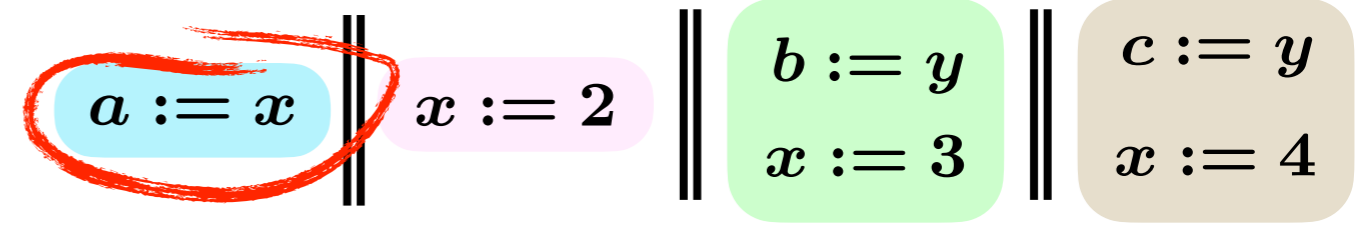
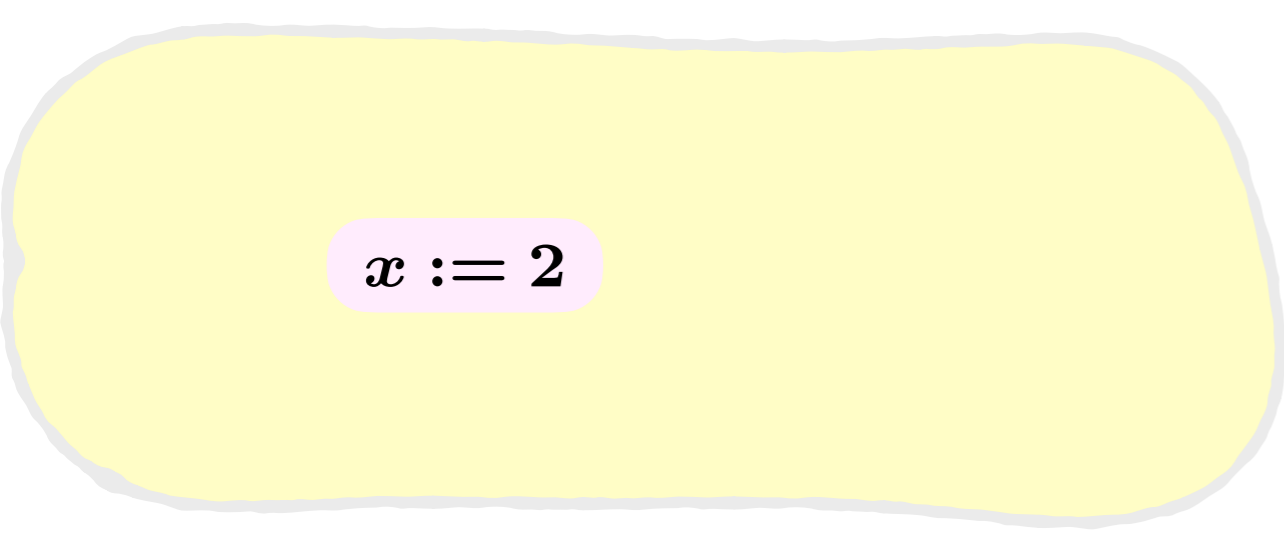
Postpone



Branch

&

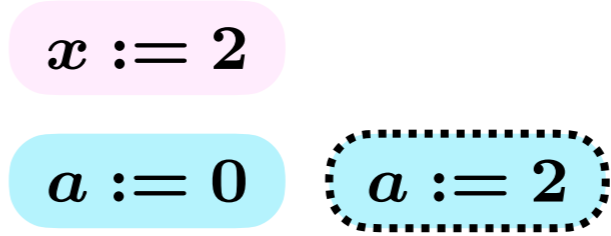
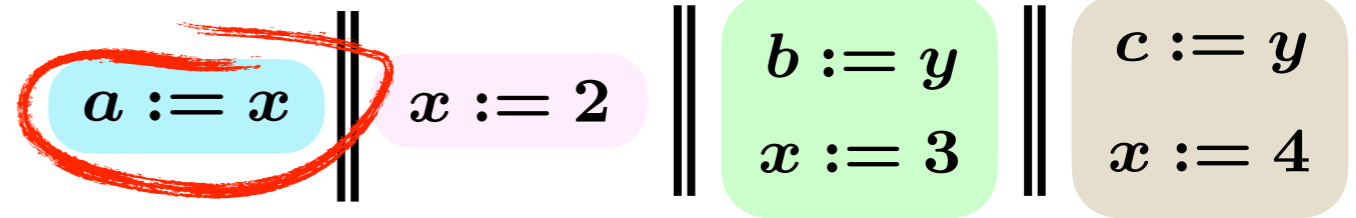
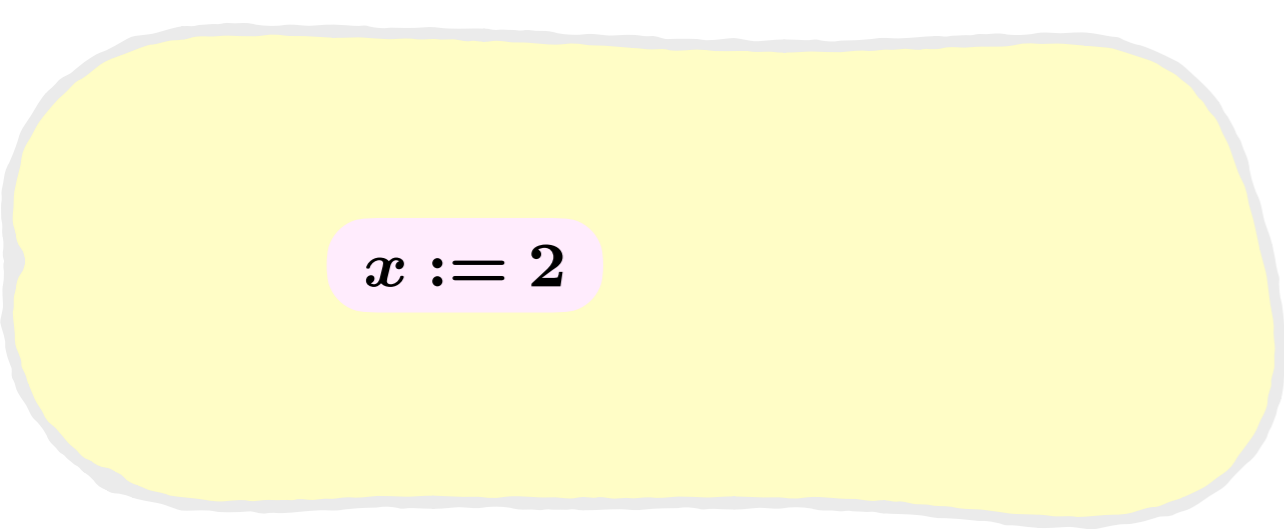
Postpone



Branch

&

Postpone

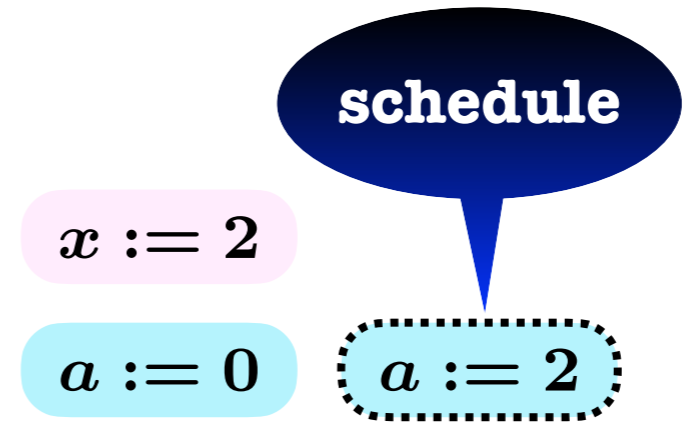
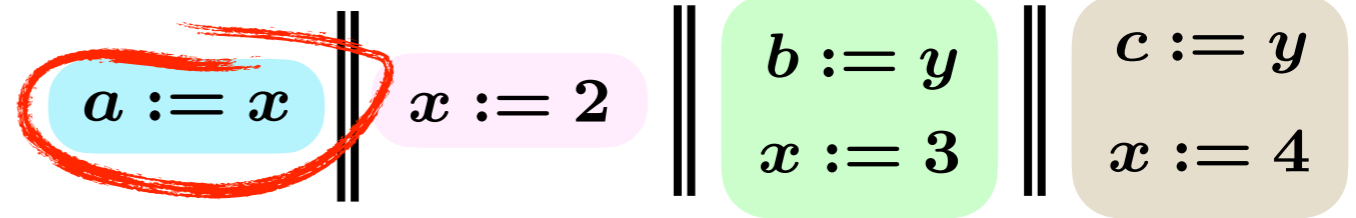
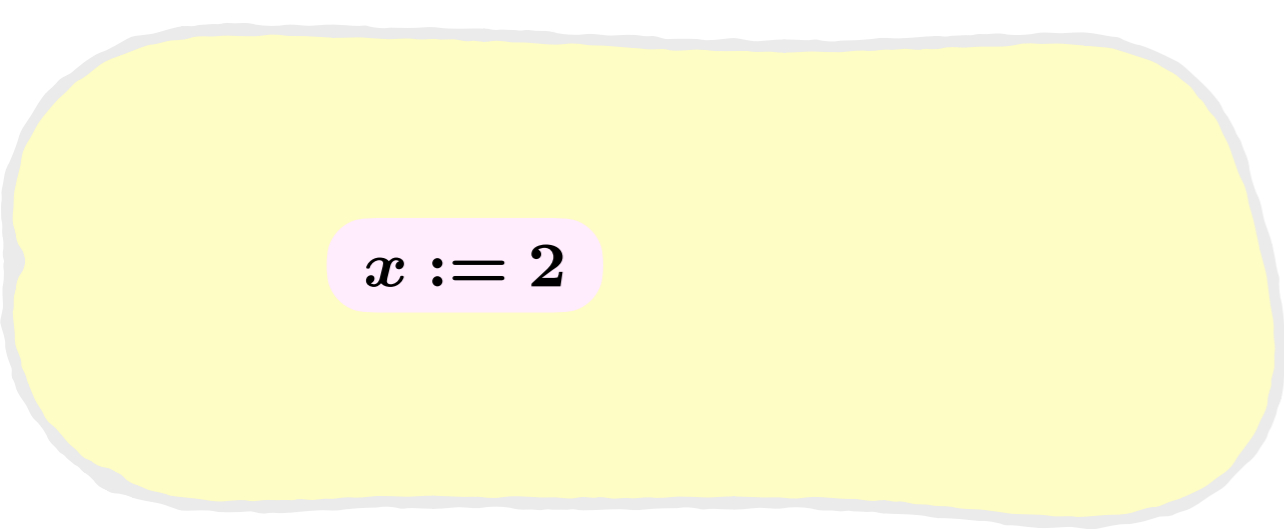


**read-
branching**

Branch

&

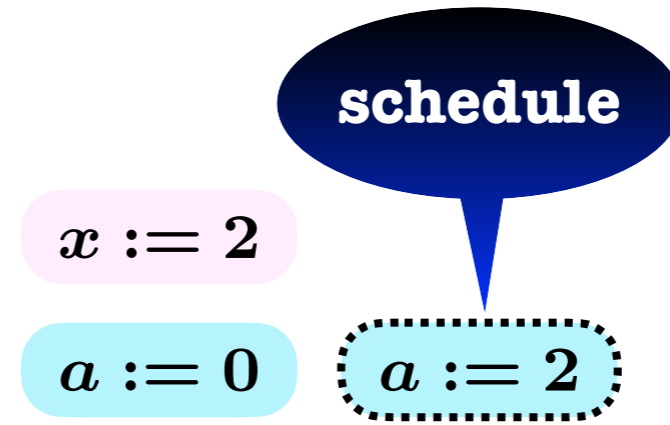
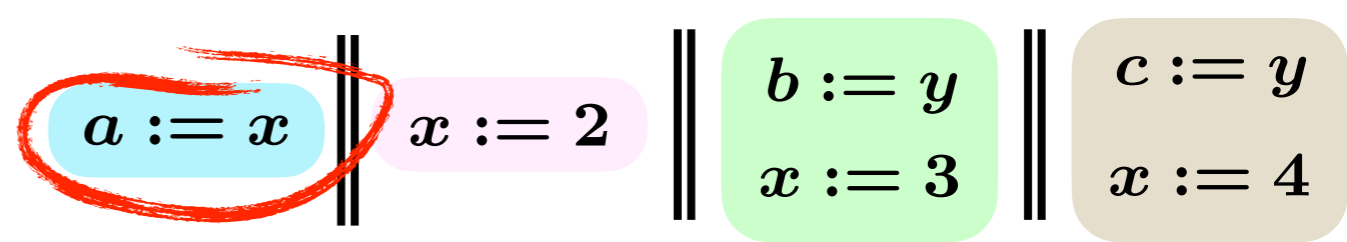
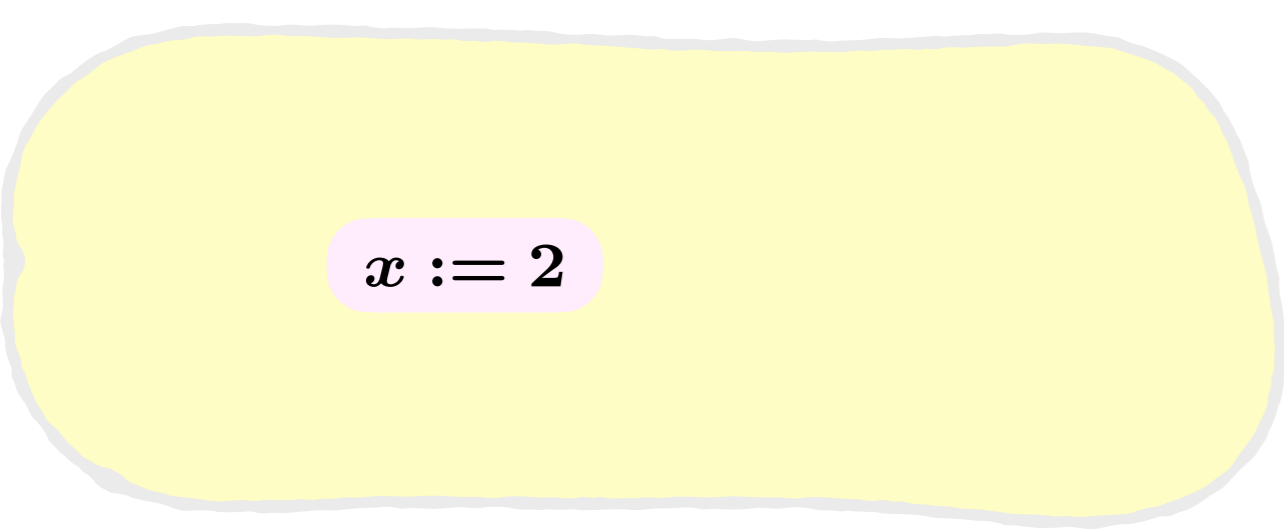
Postpone



Branch

&

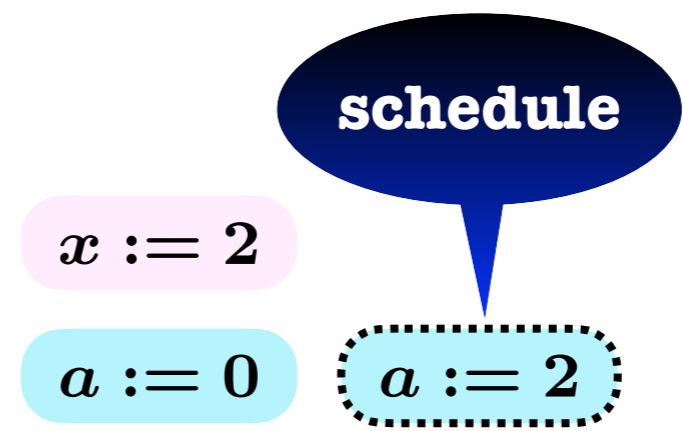
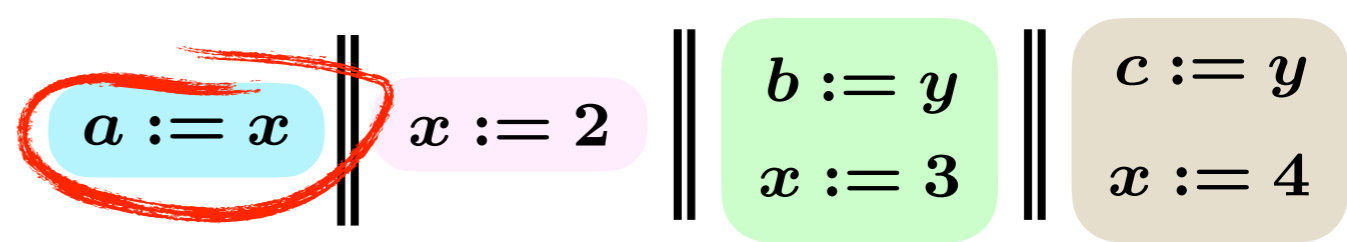
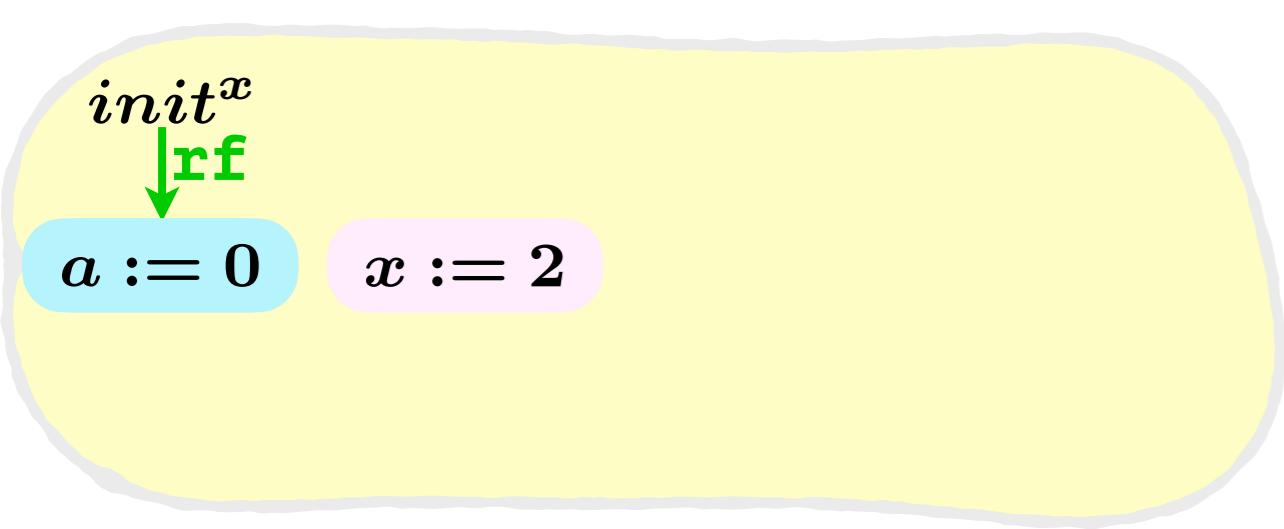
Postpone



Branch

&

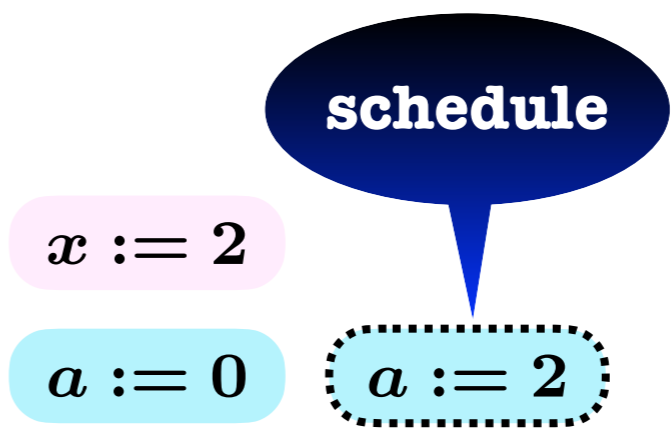
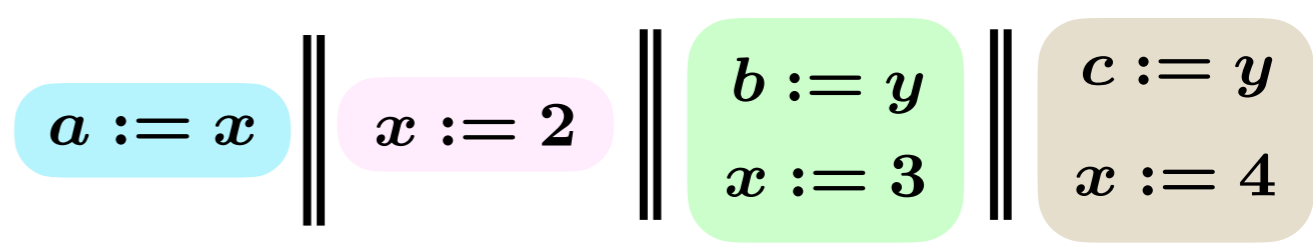
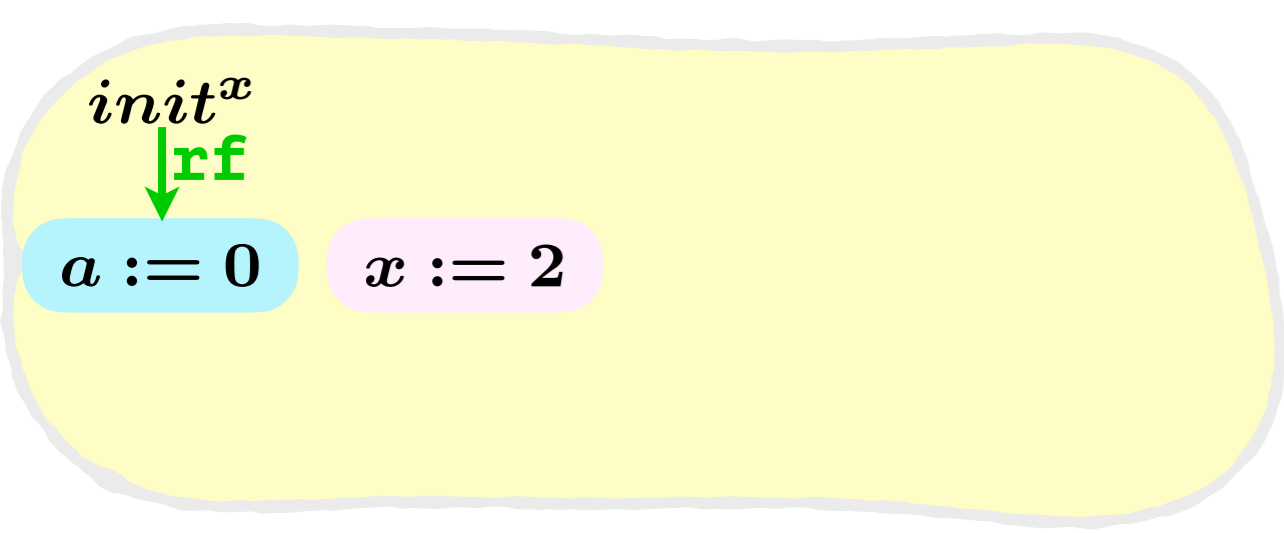
Postpone



Branch

&

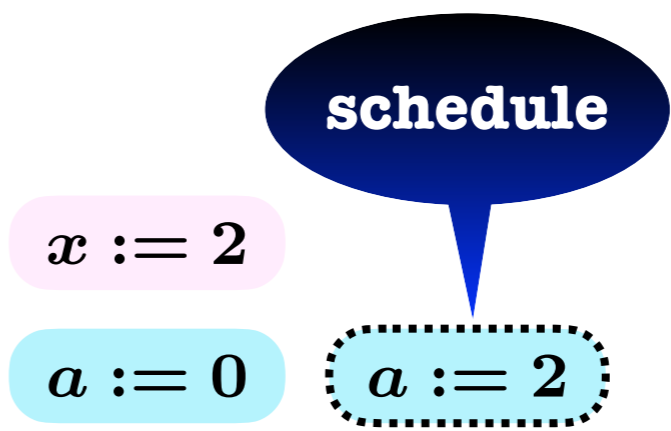
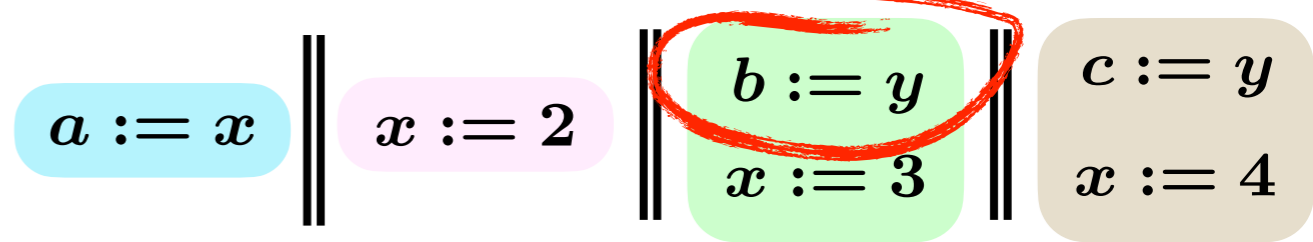
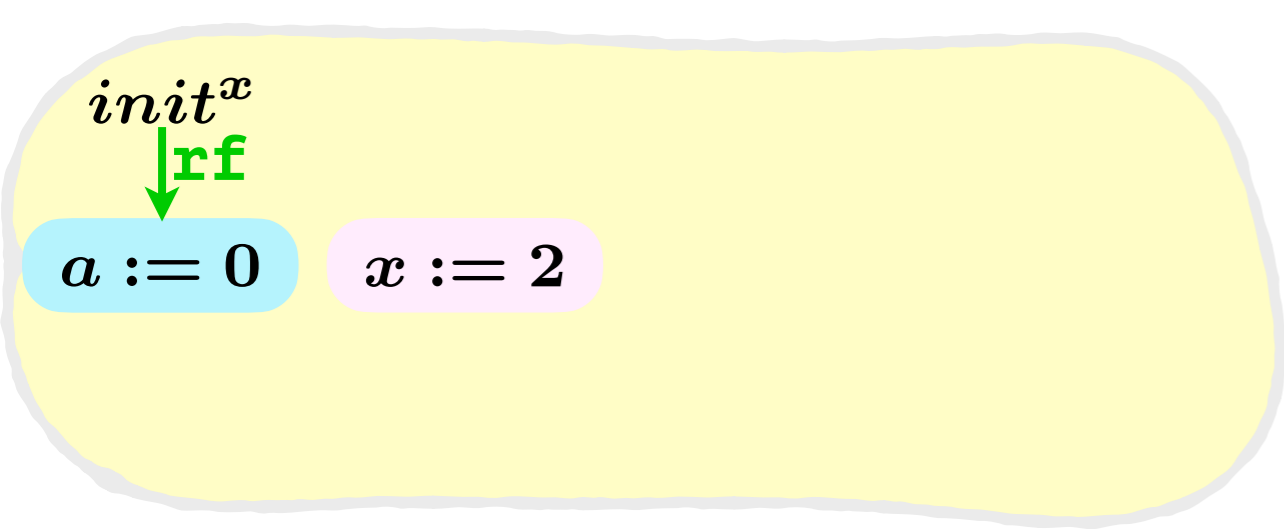
Postpone



Branch

&

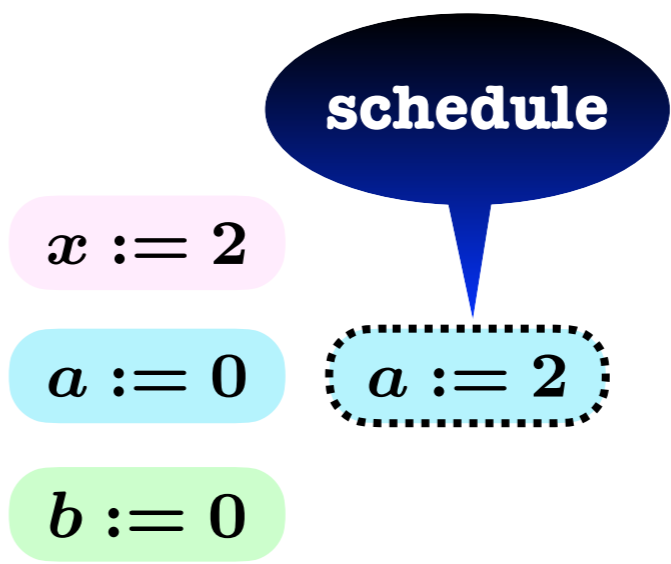
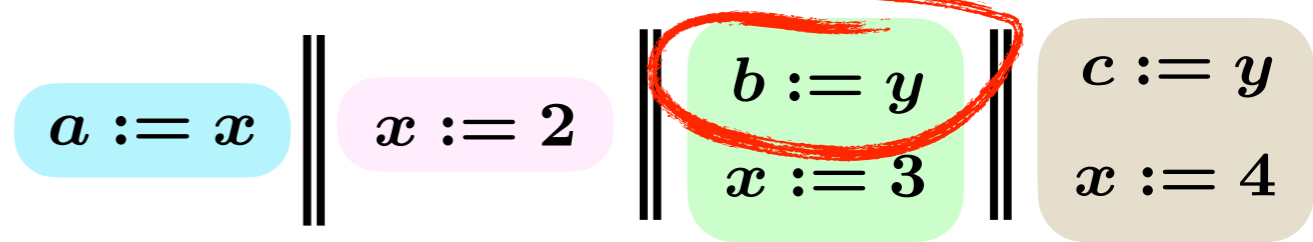
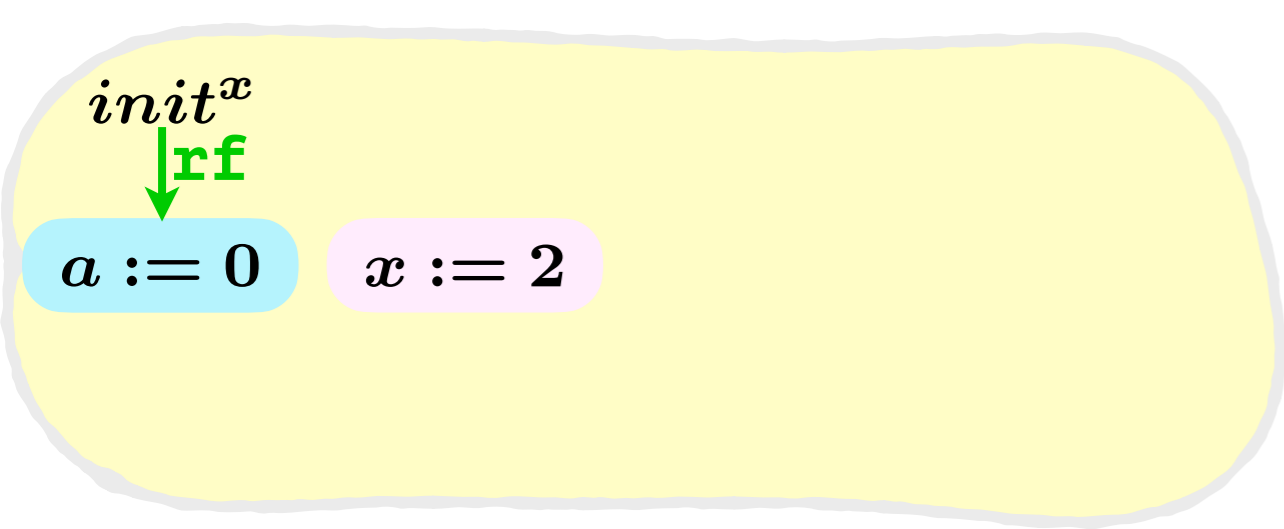
Postpone



Branch

&

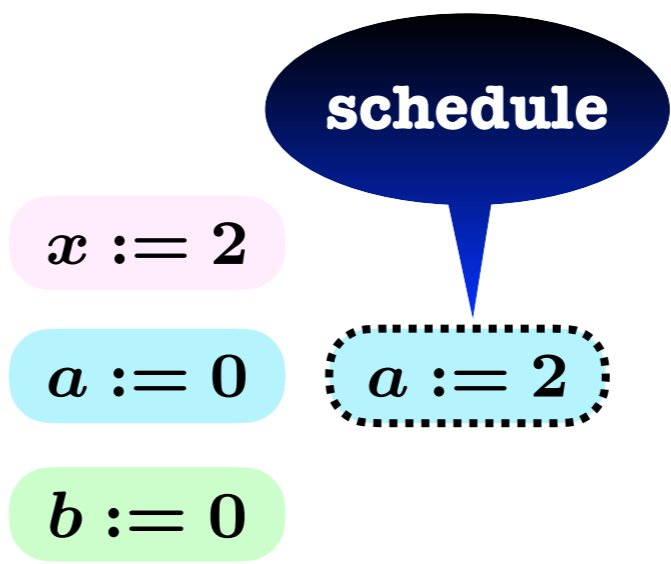
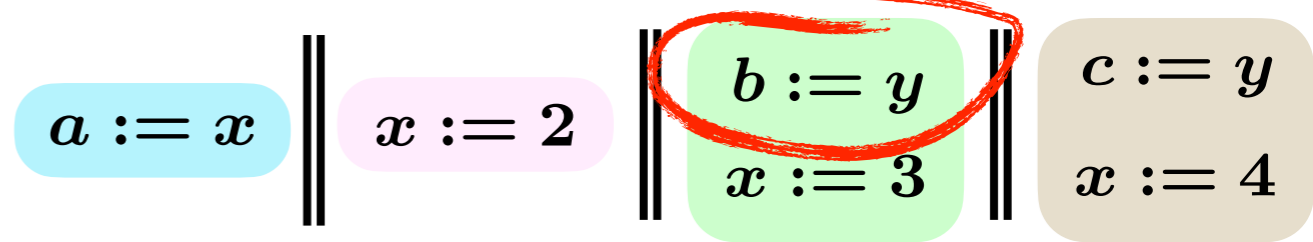
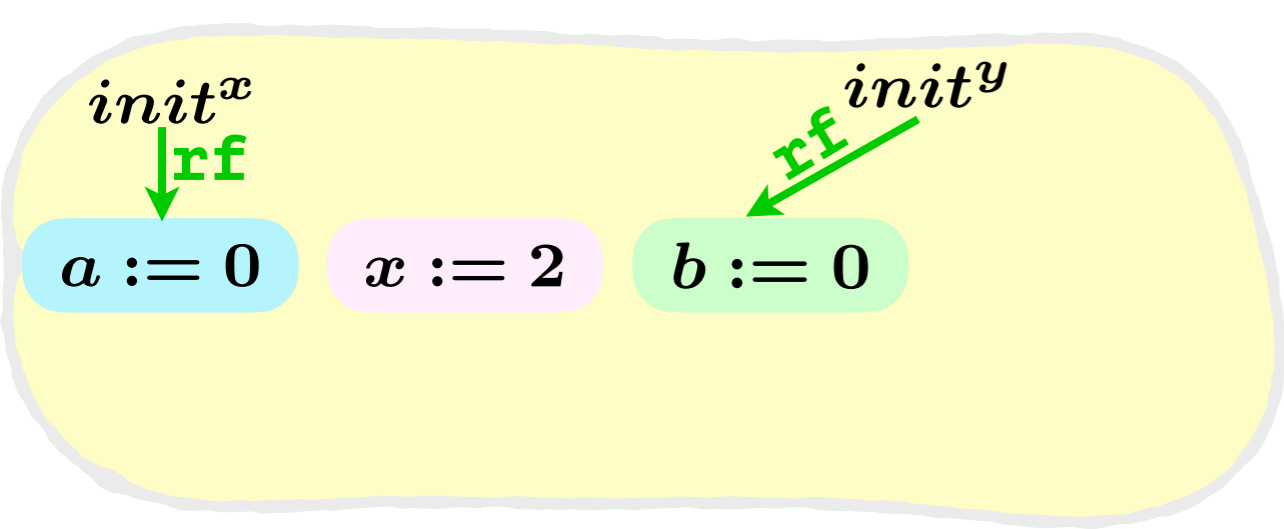
Postpone



Branch

&

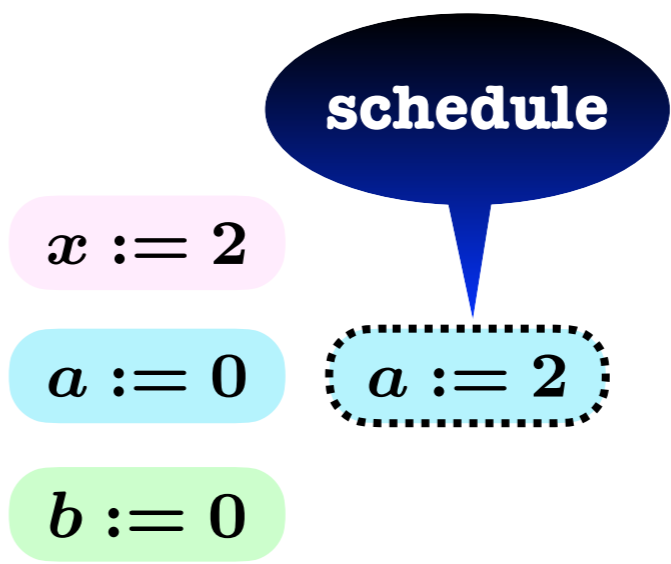
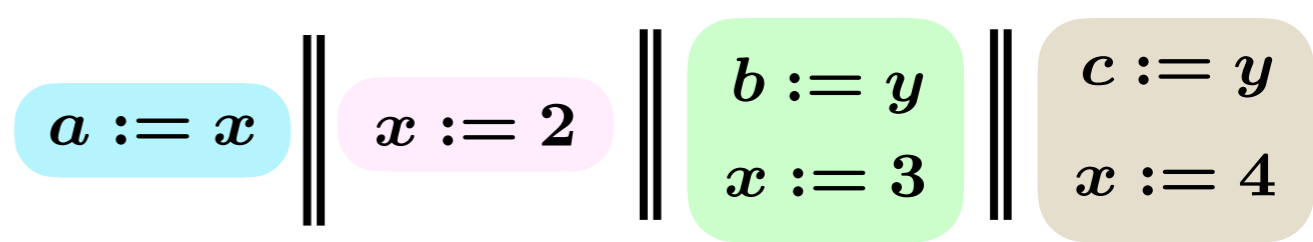
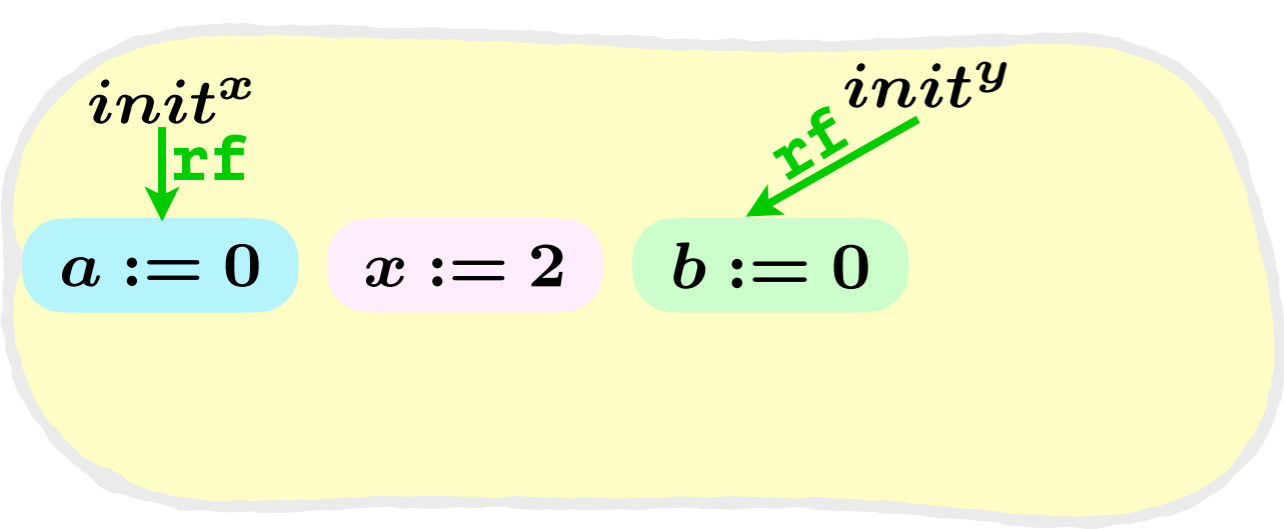
Postpone



Branch

&

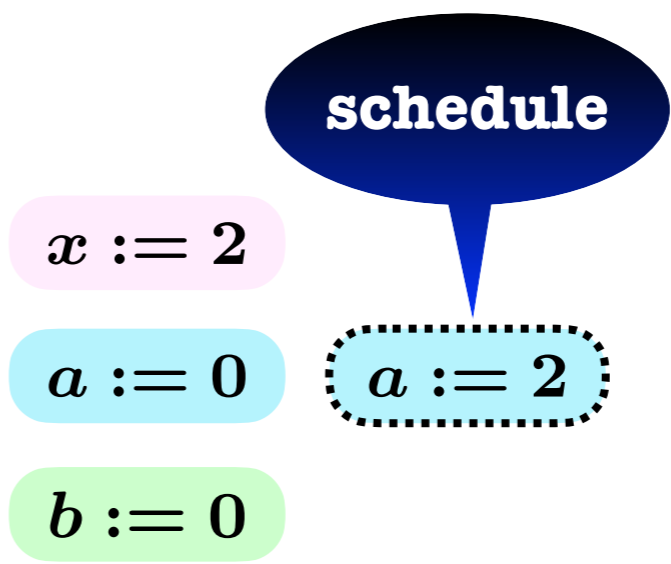
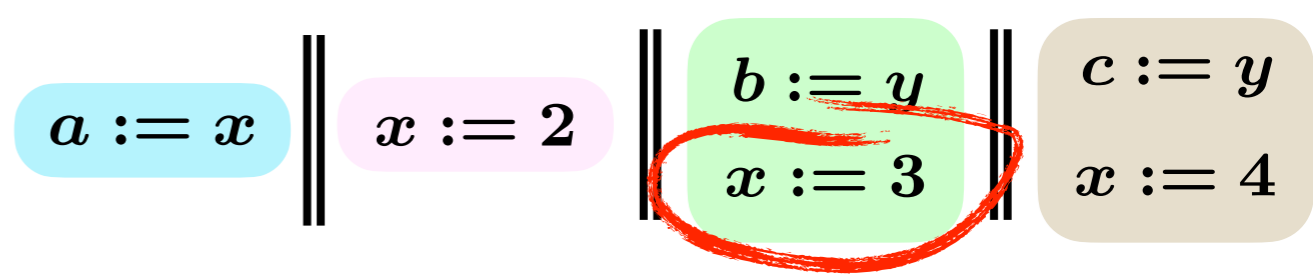
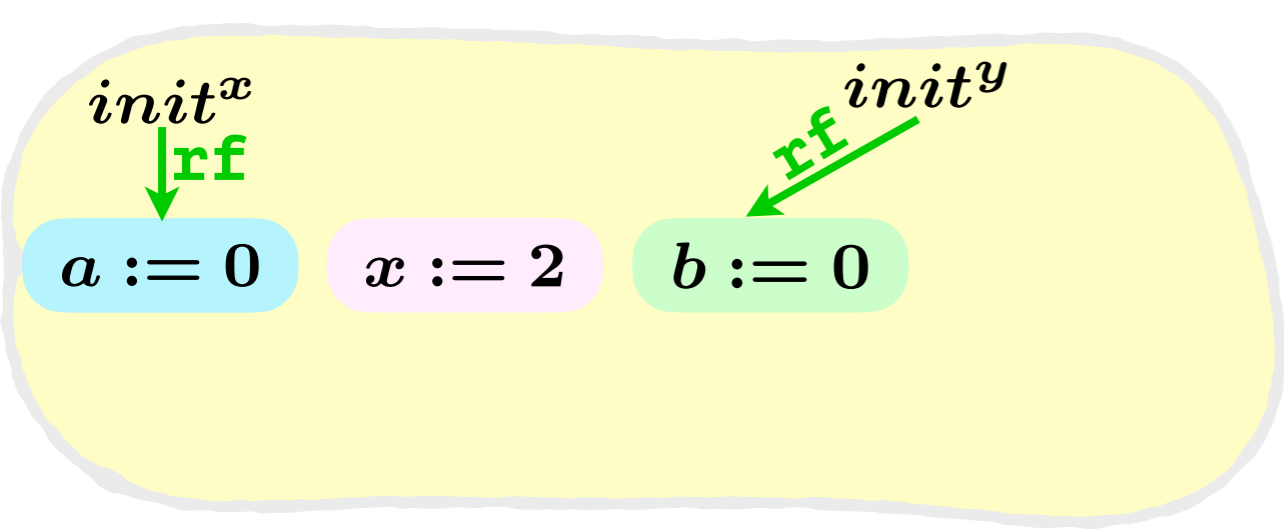
Postpone



Branch

&

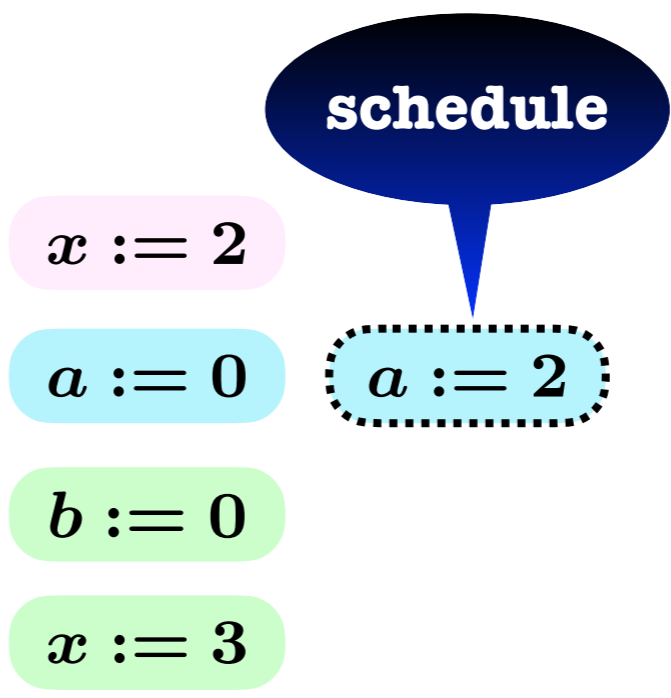
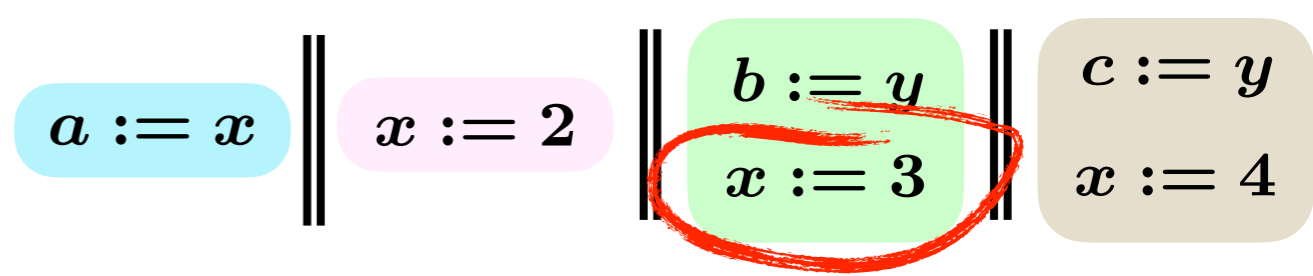
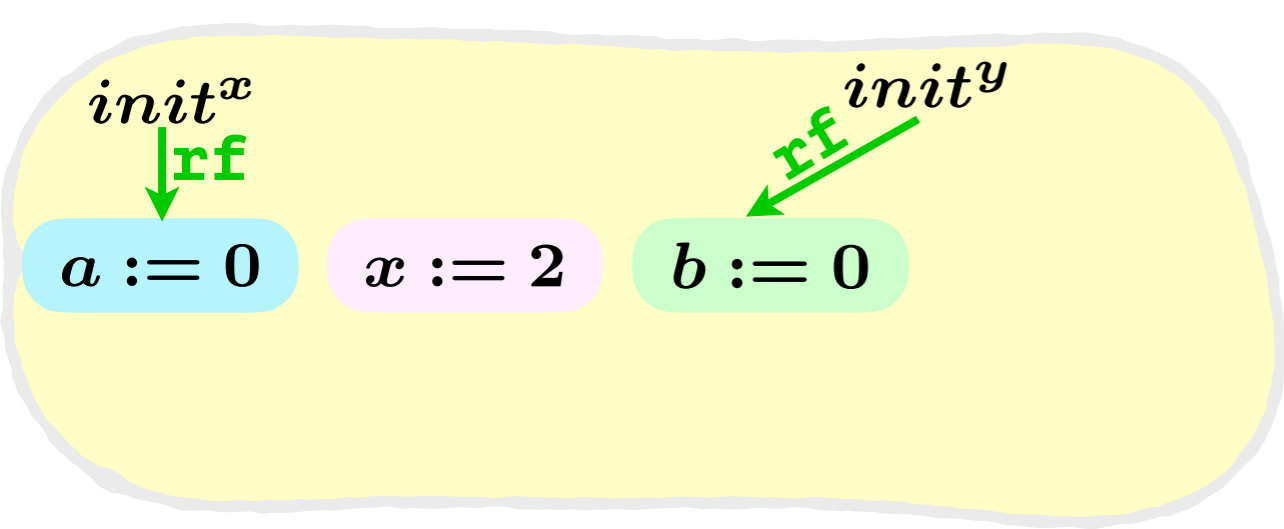
Postpone



Branch

&

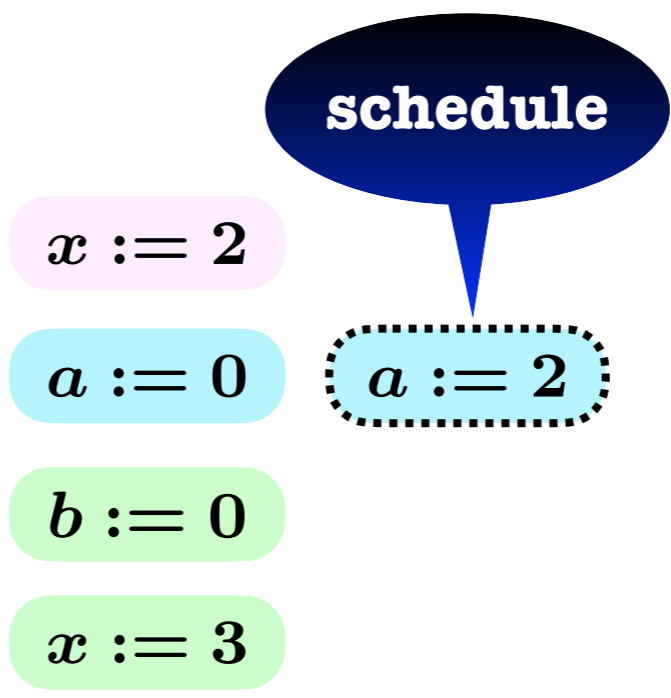
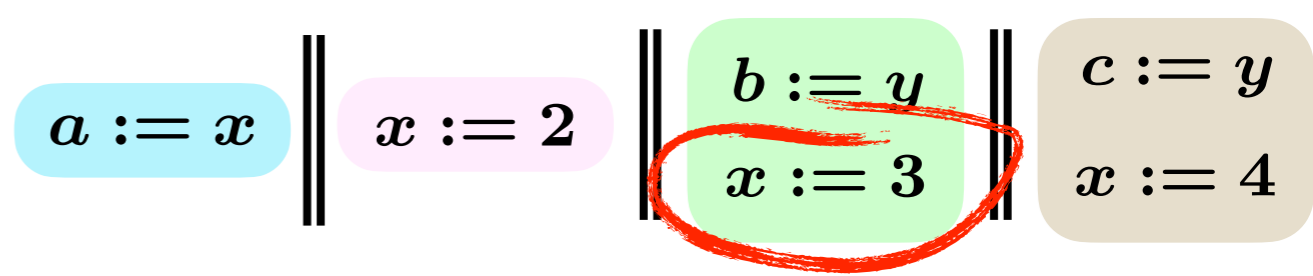
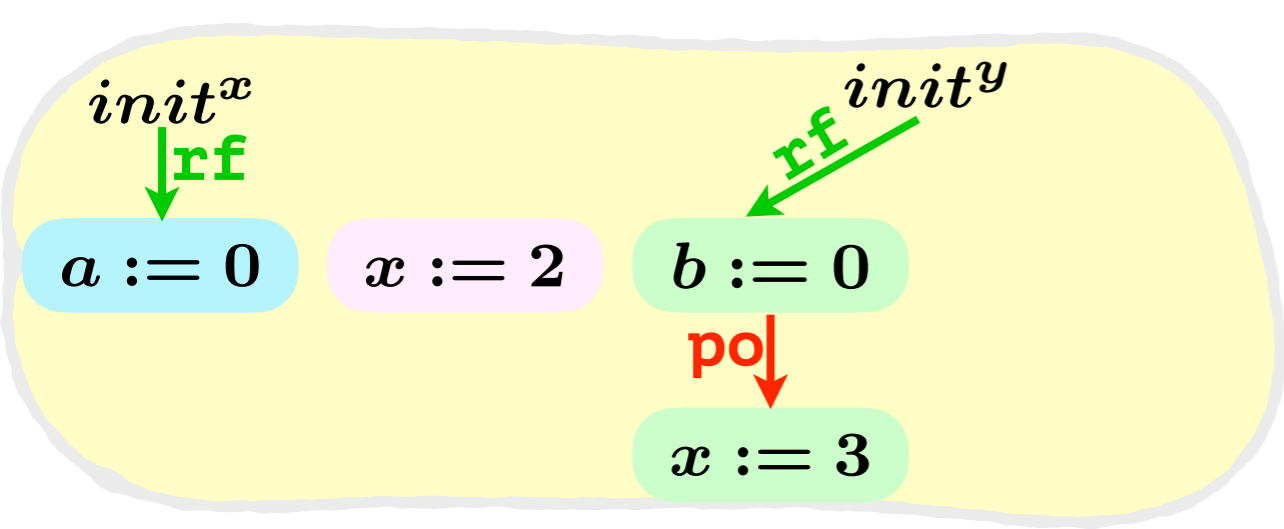
Postpone



Branch

&

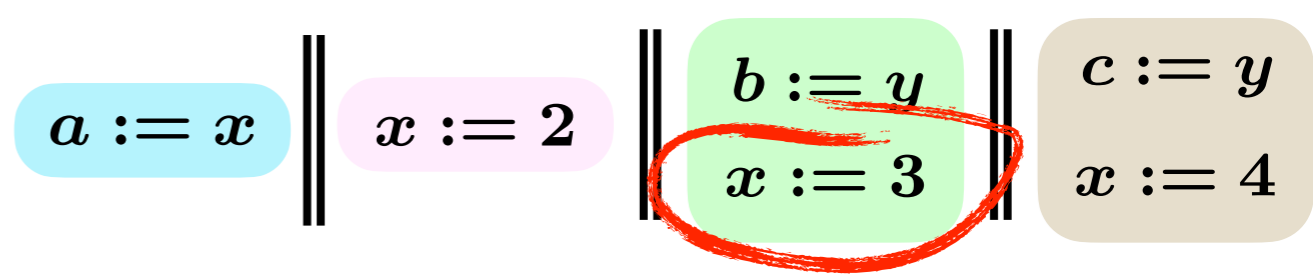
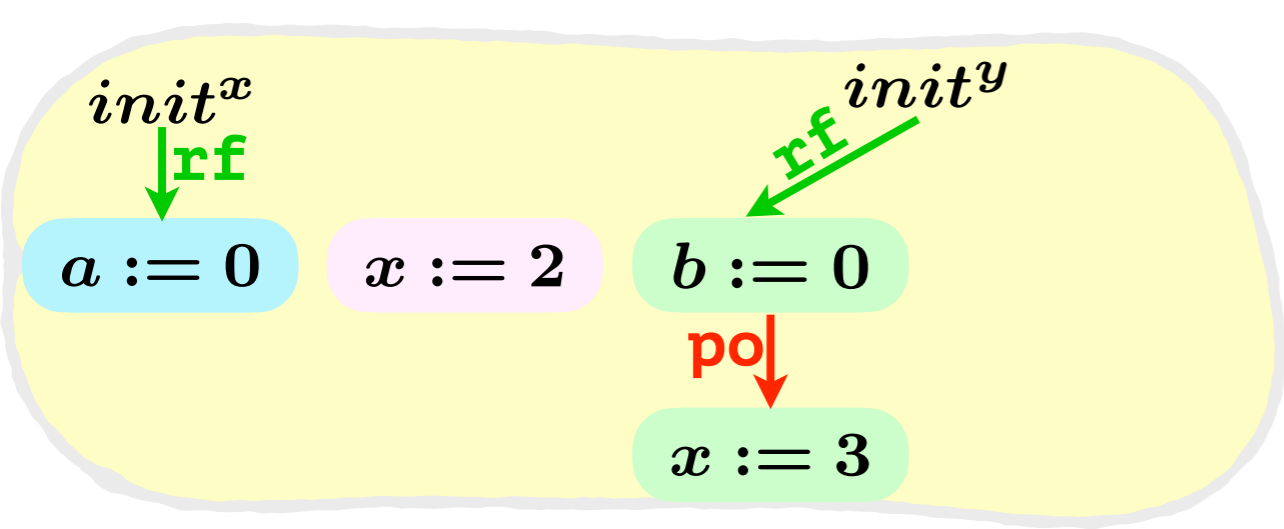
Postpone



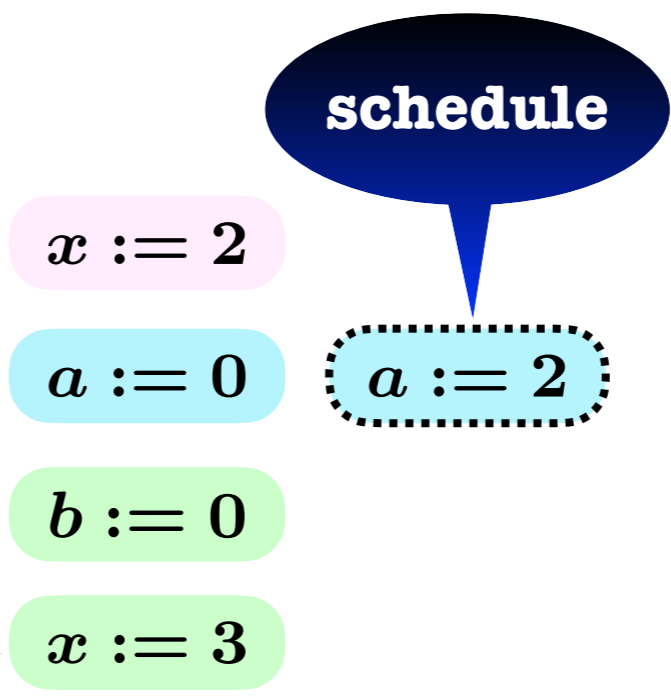
Branch

&

Postpone



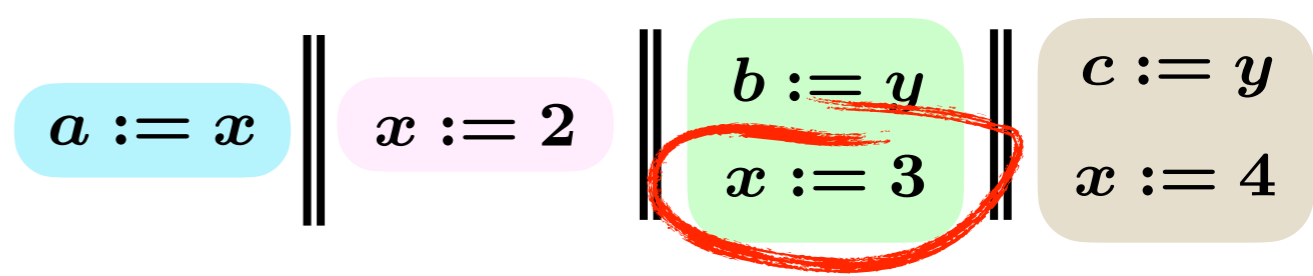
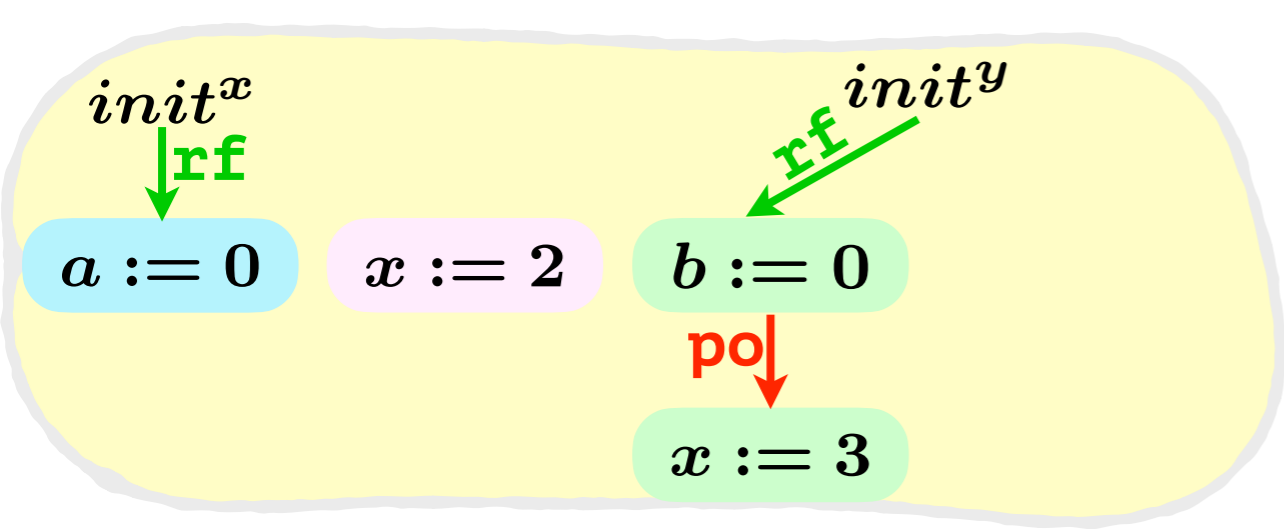
postponed write



Branch

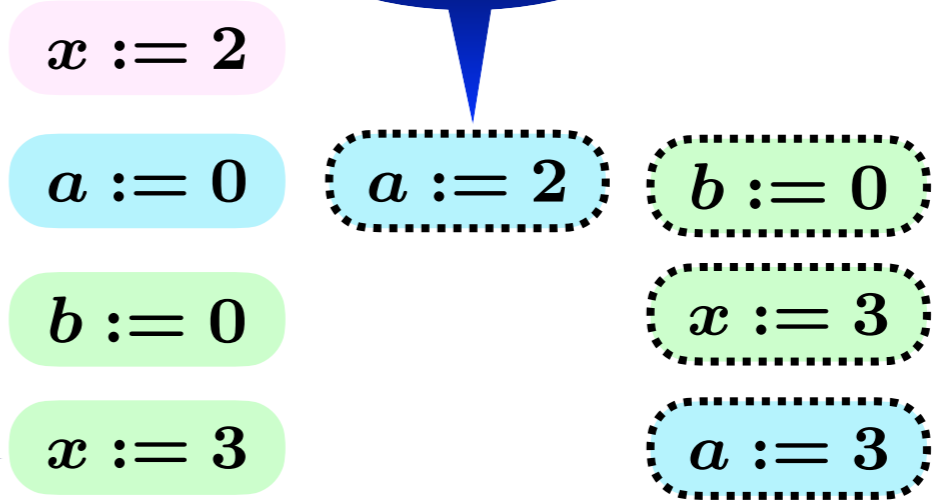
&

Postpone



postponed write

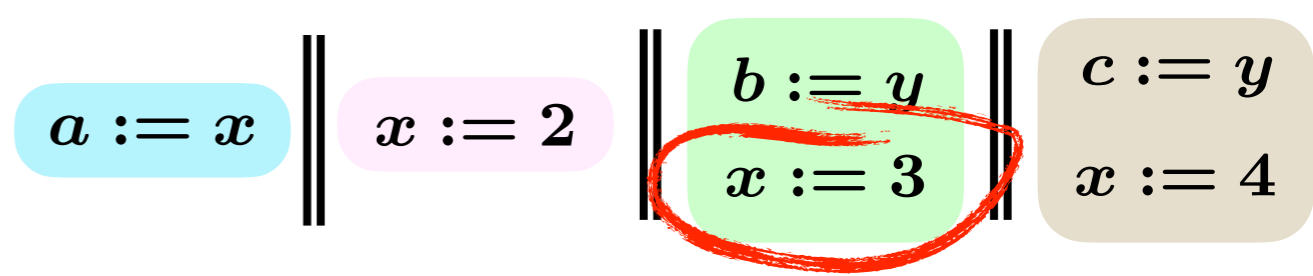
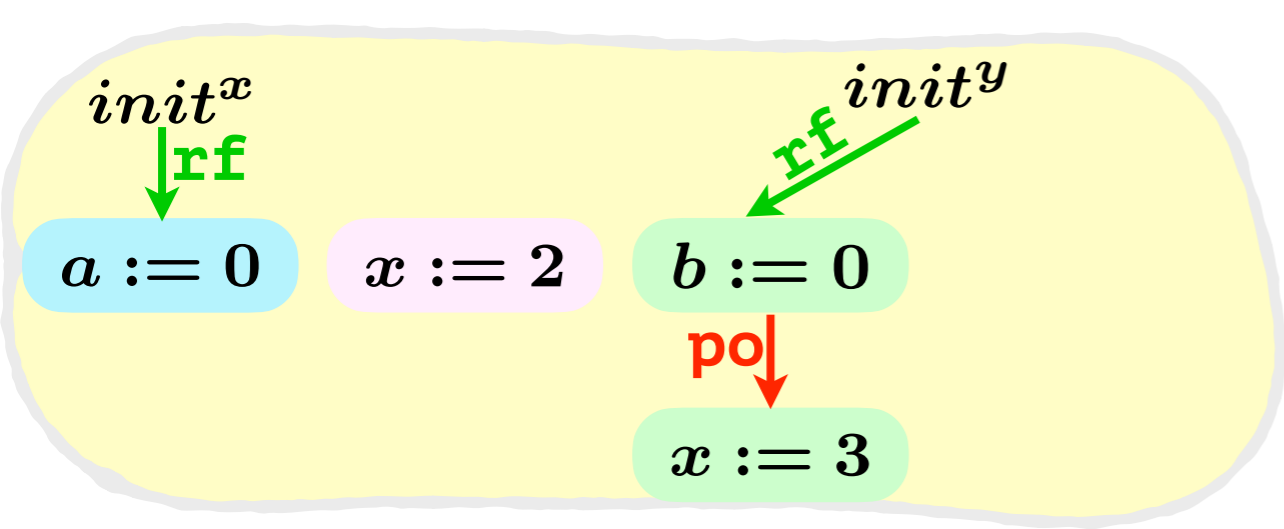
schedule



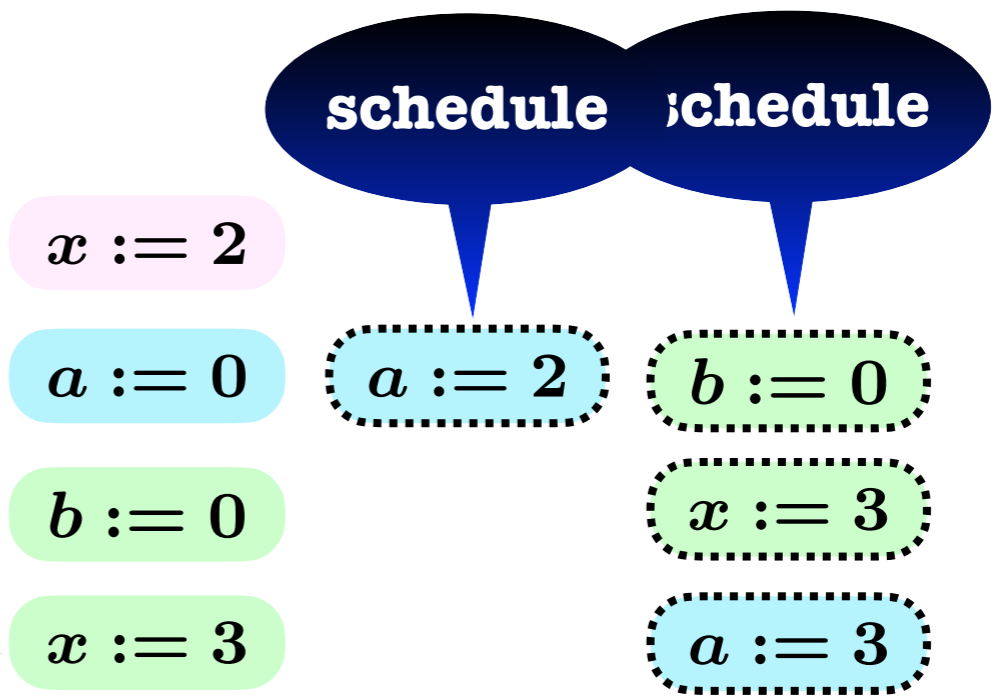
Branch

&

Postpone



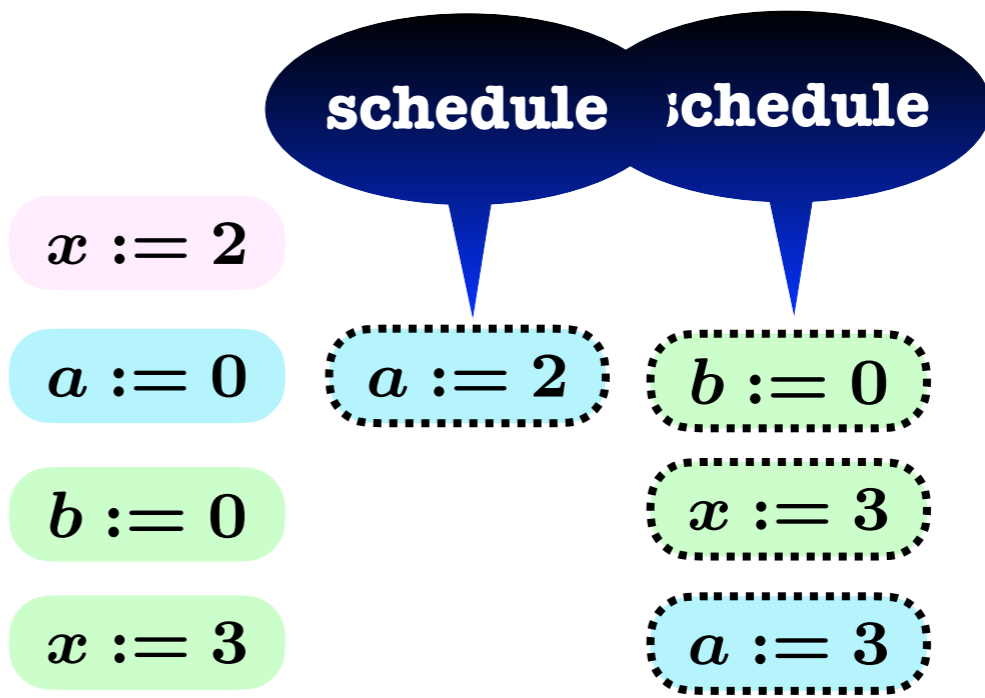
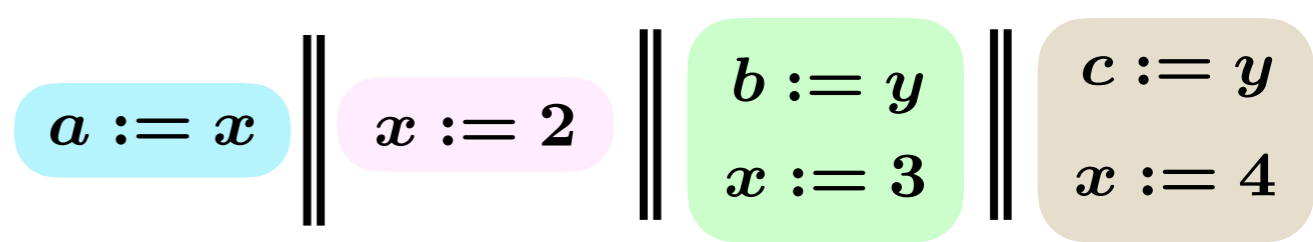
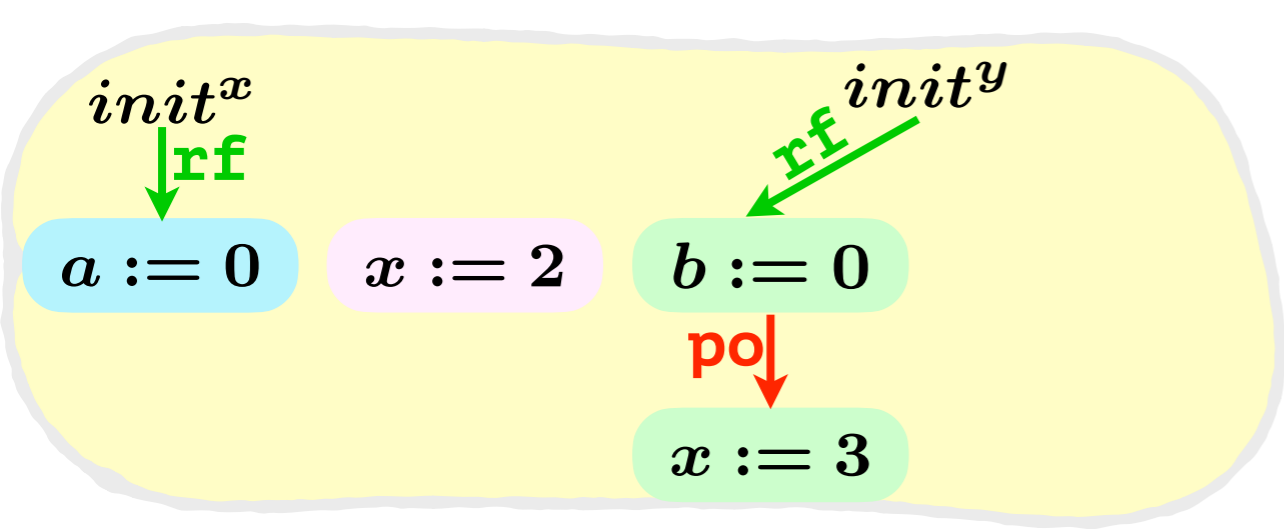
postponed write



Branch

&

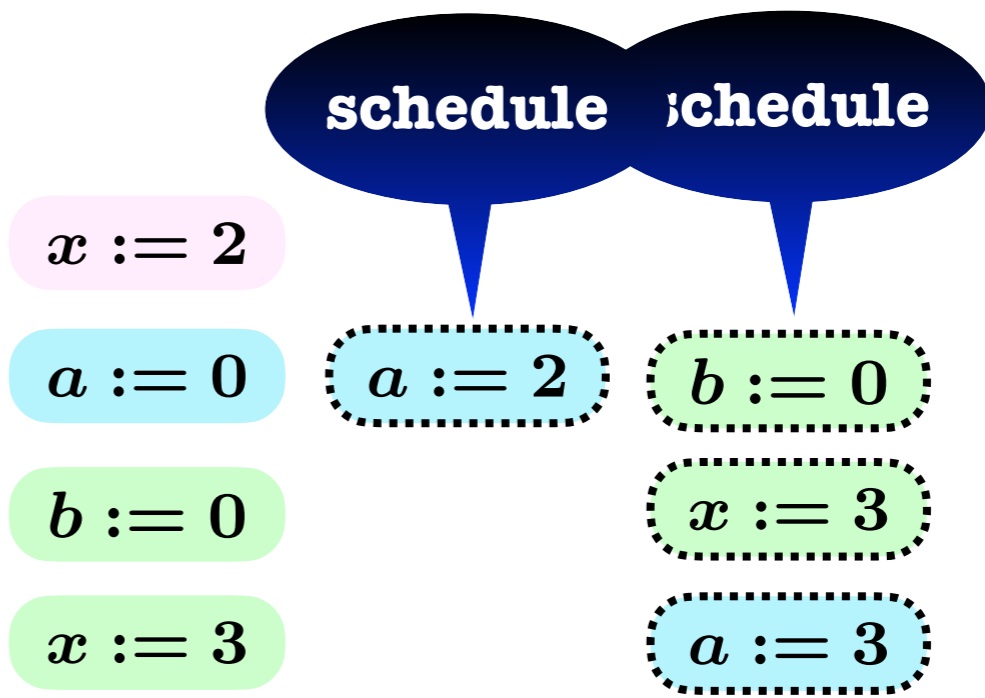
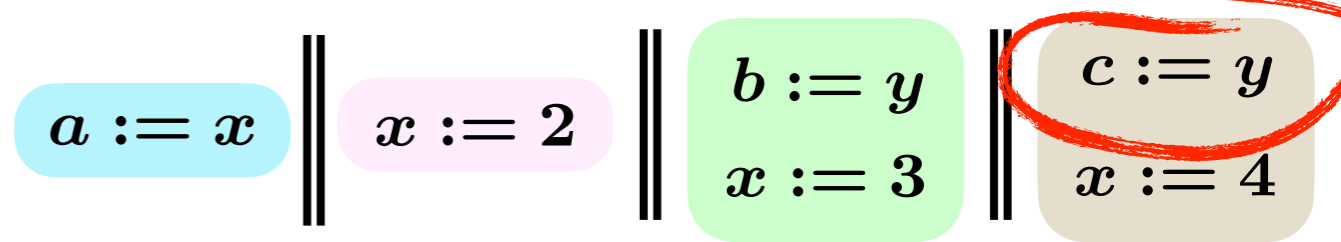
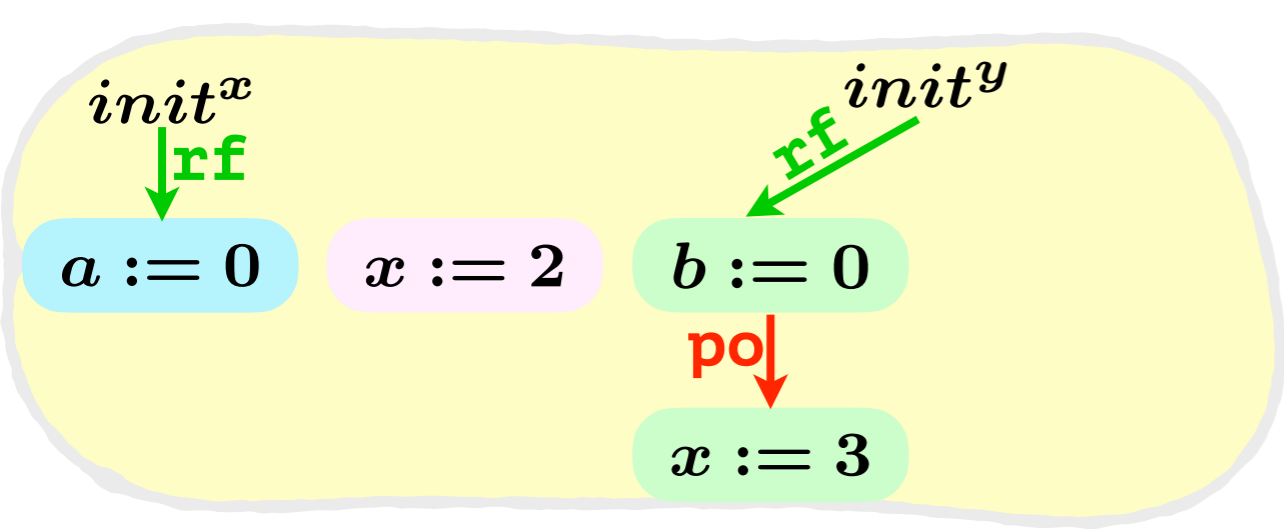
Postpone



Branch

&

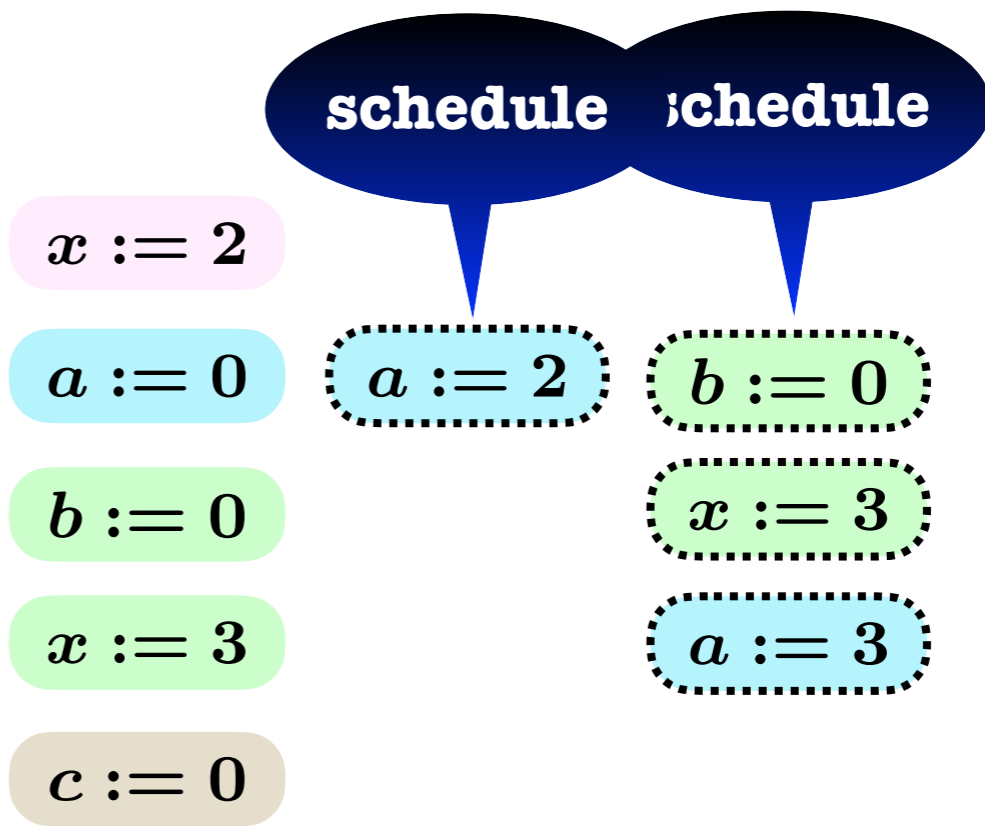
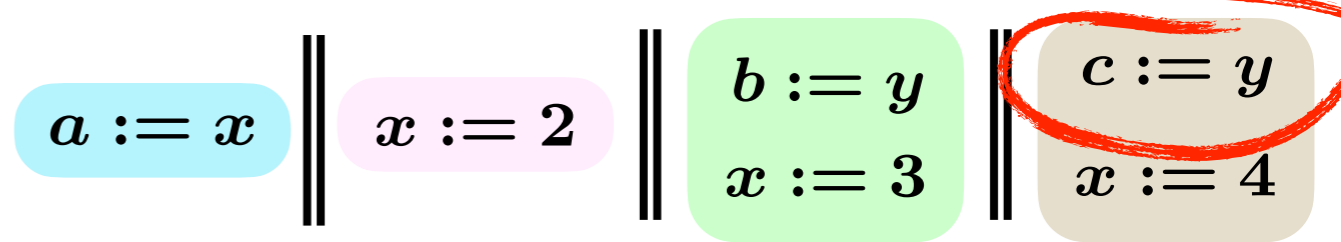
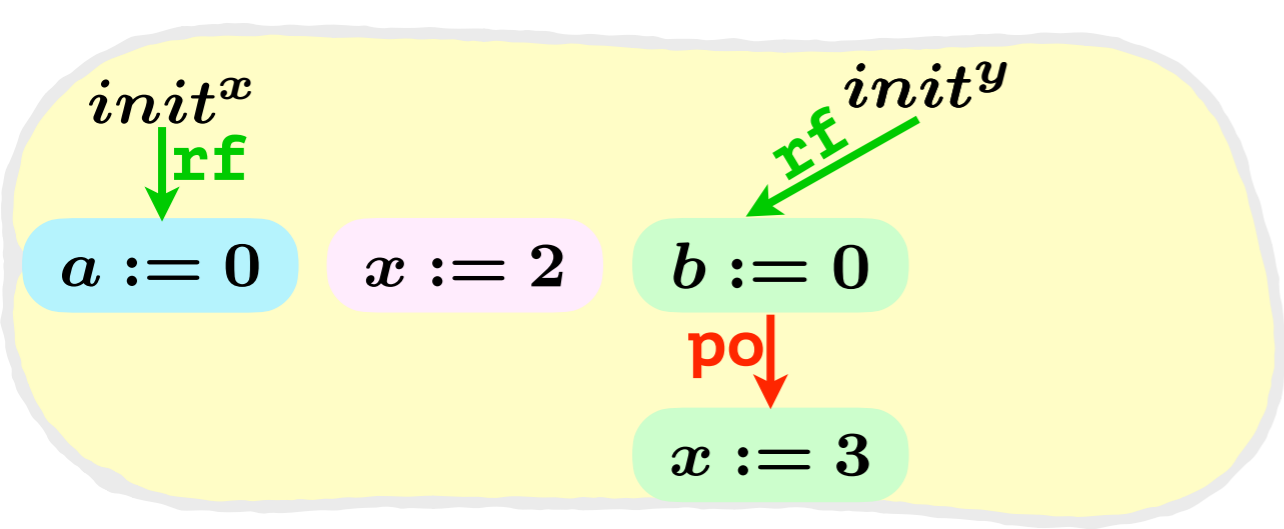
Postpone



Branch

&

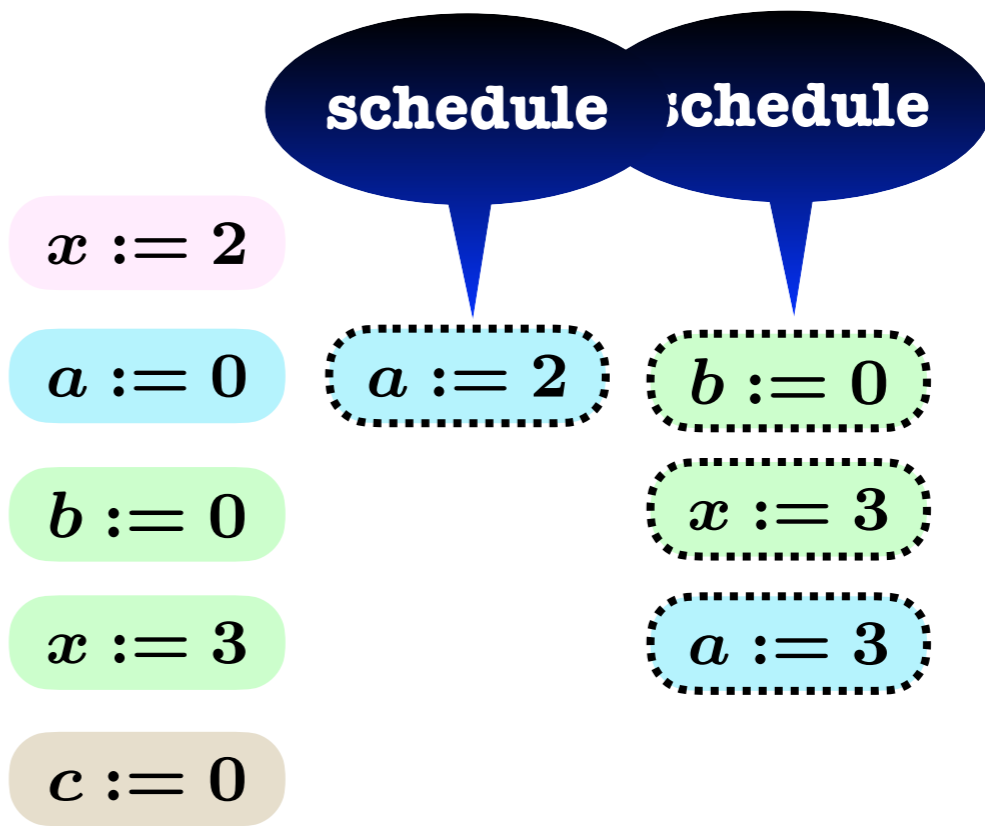
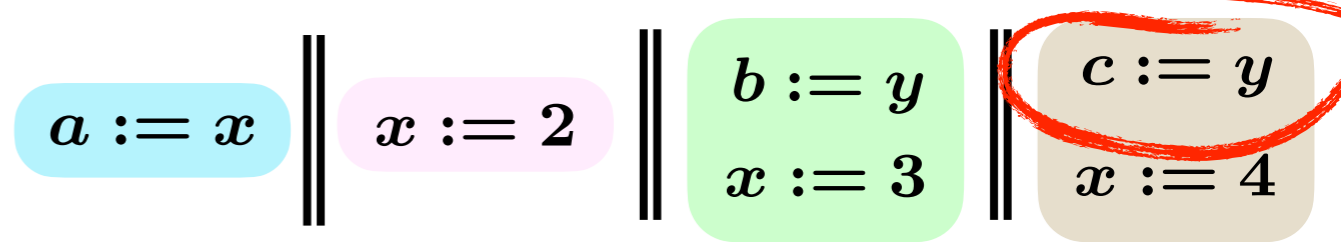
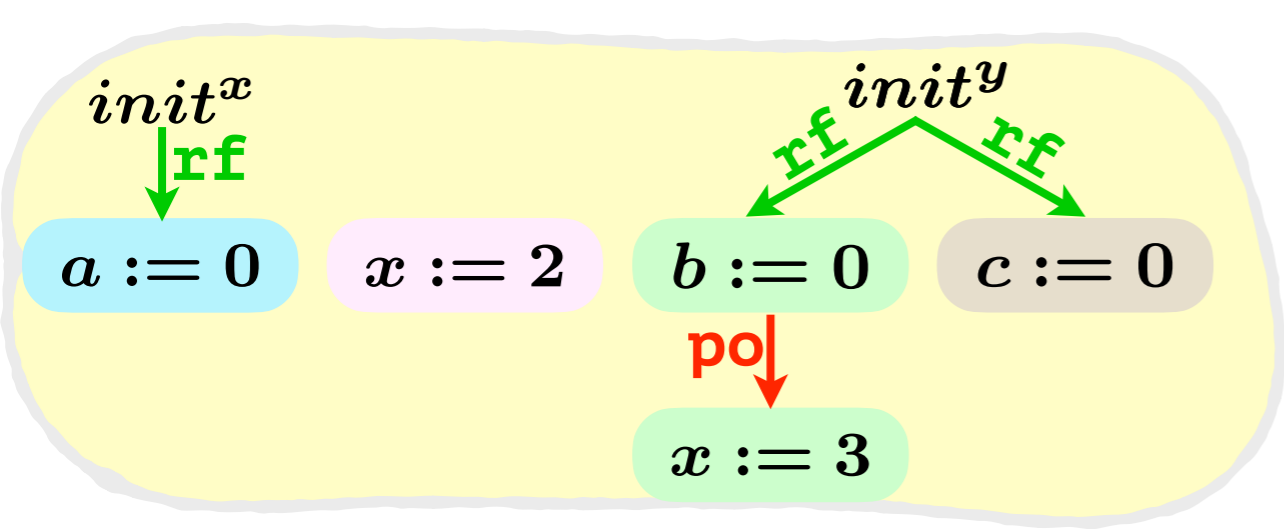
Postpone



Branch

&

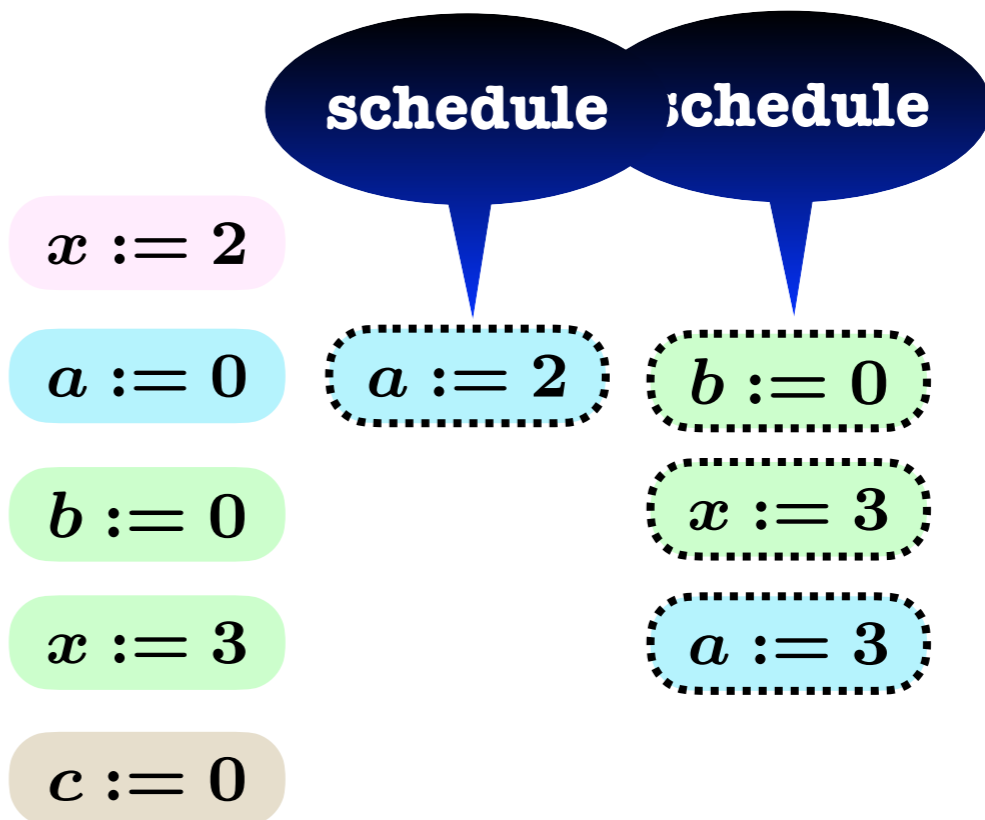
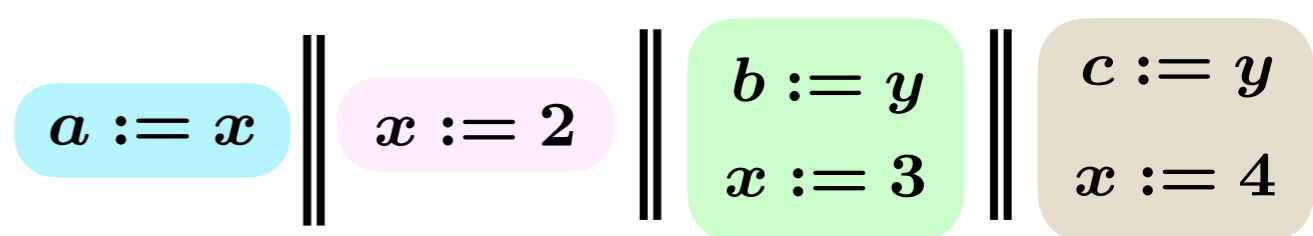
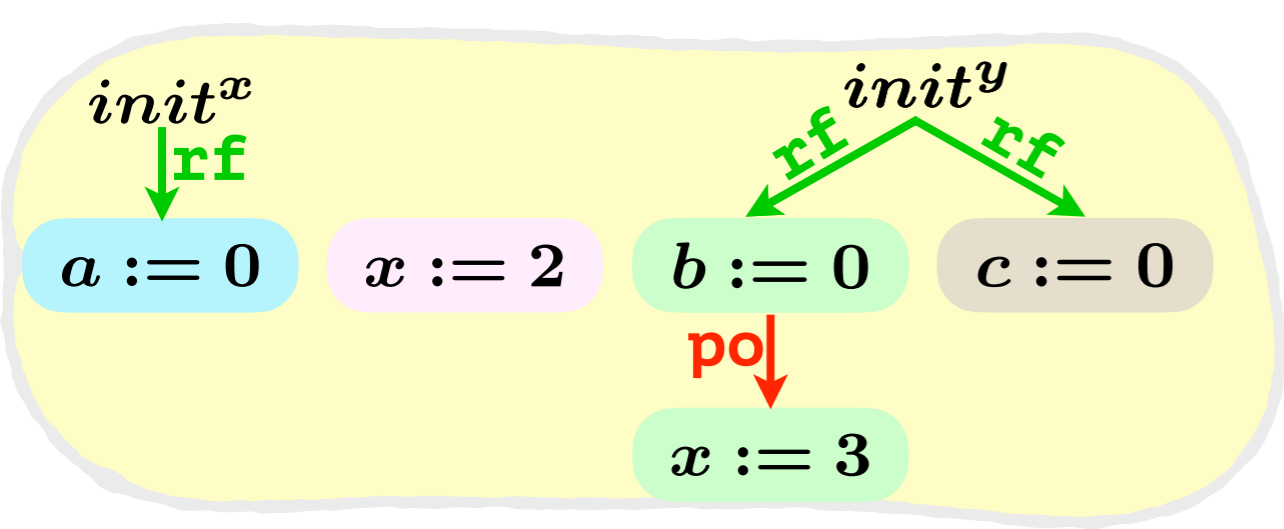
Postpone



Branch

&

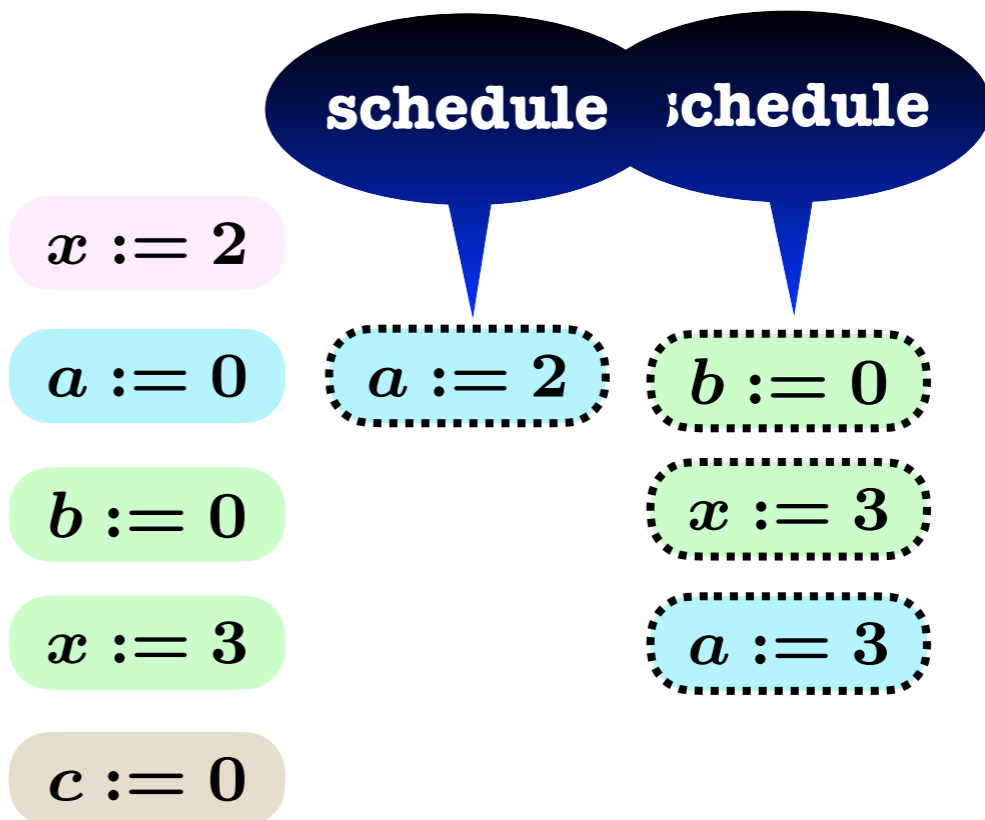
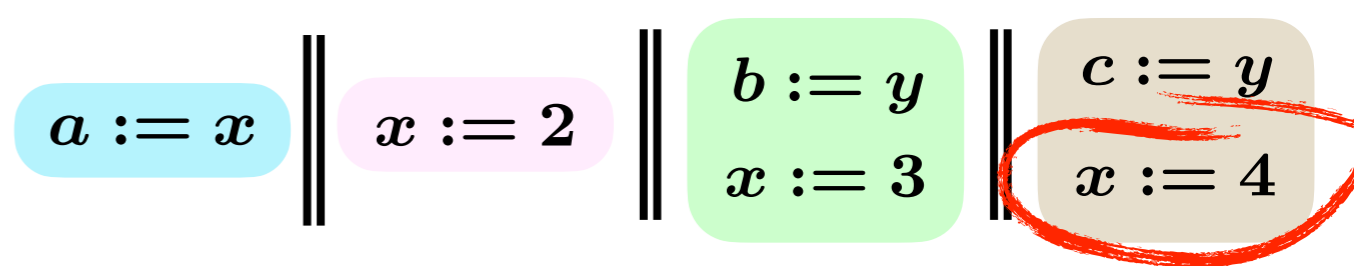
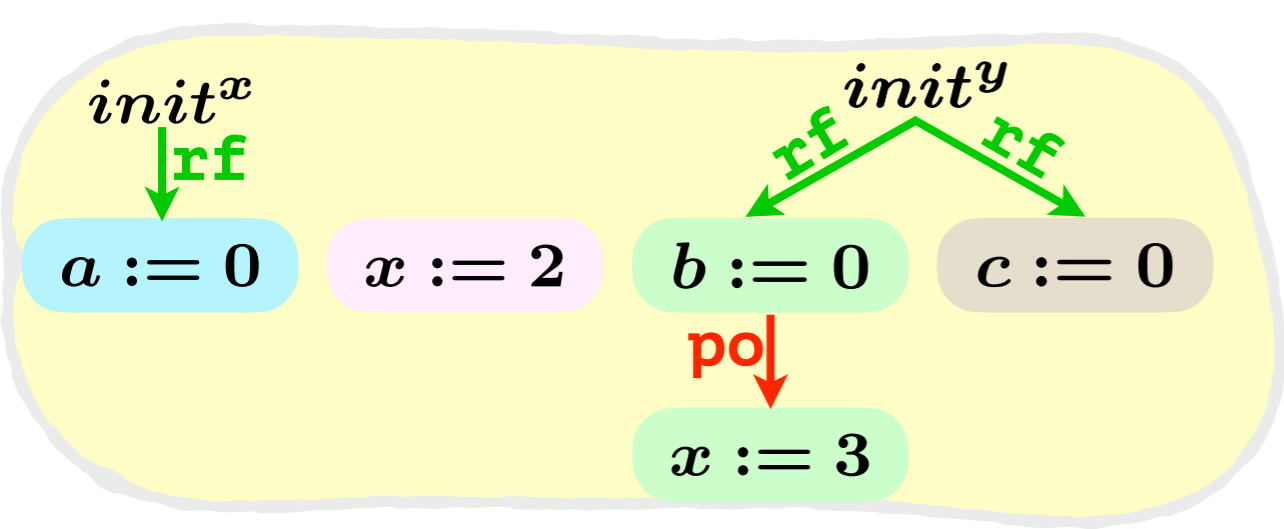
Postpone



Branch

&

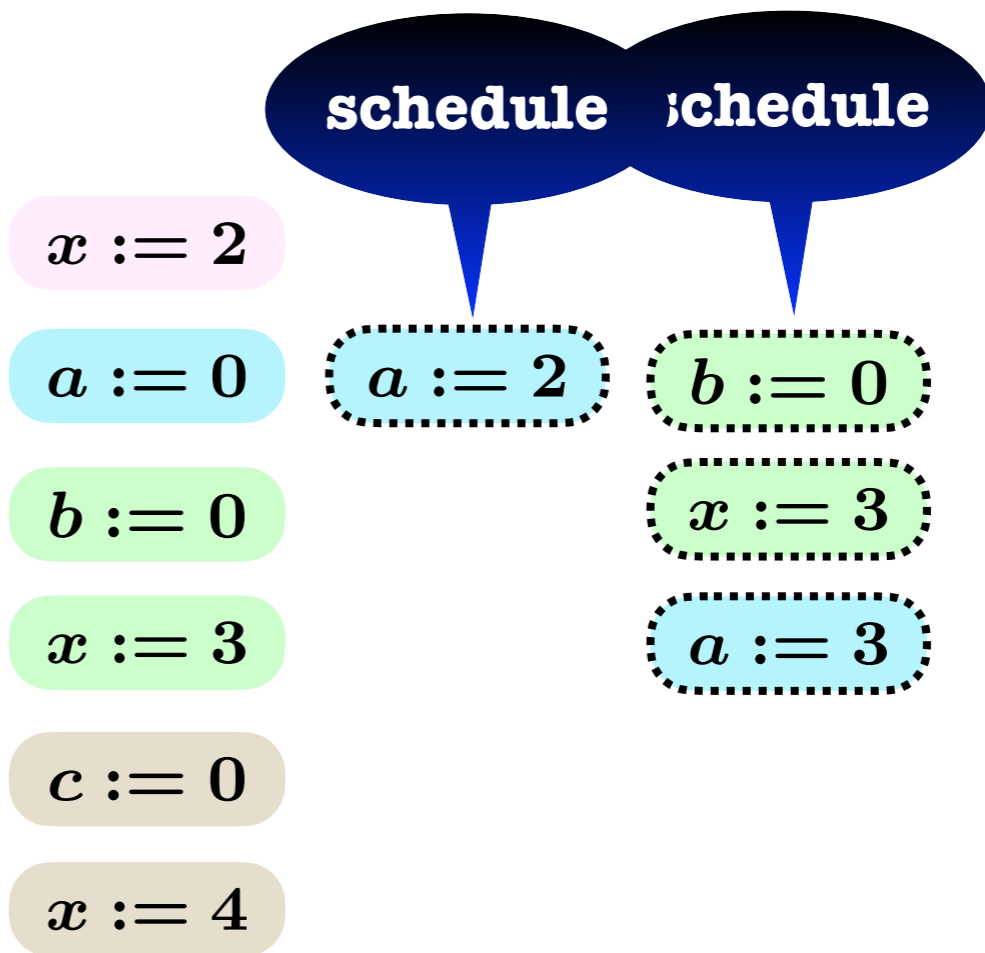
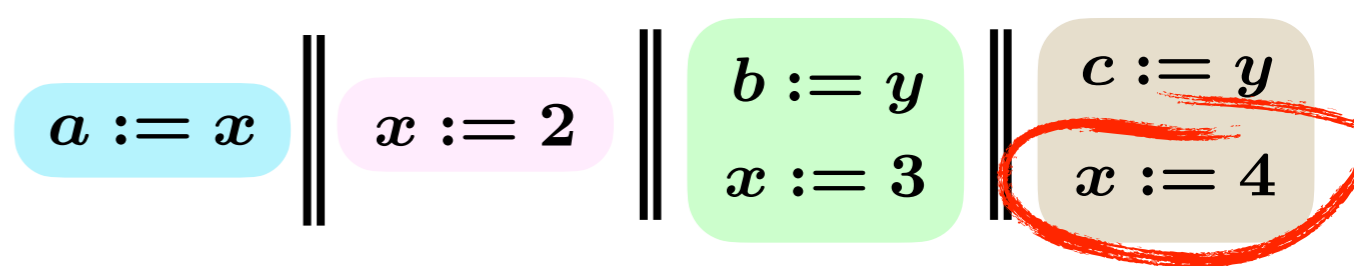
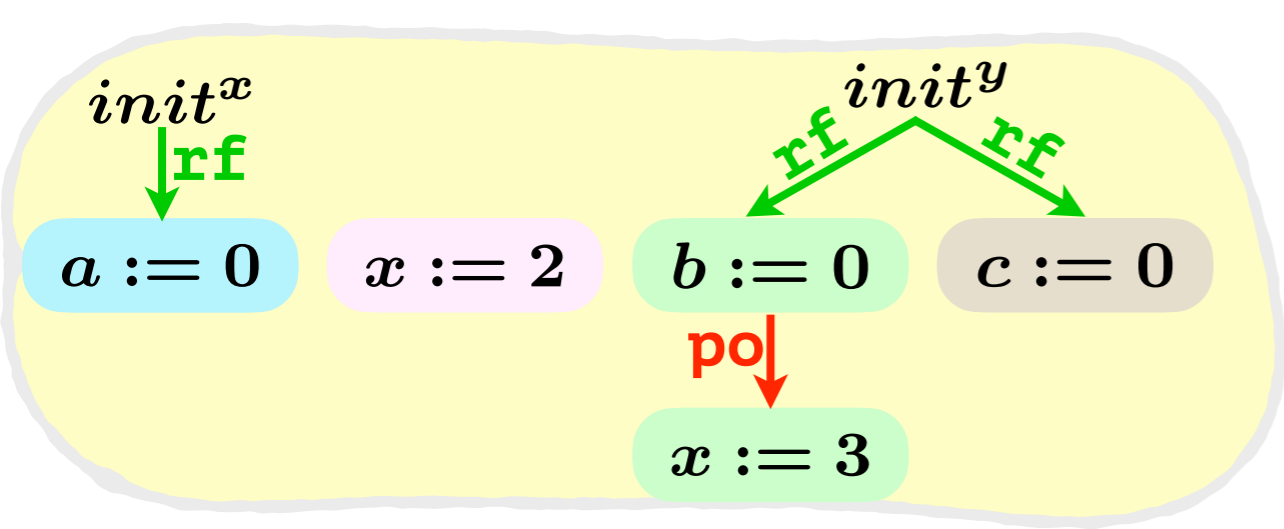
Postpone



Branch

&

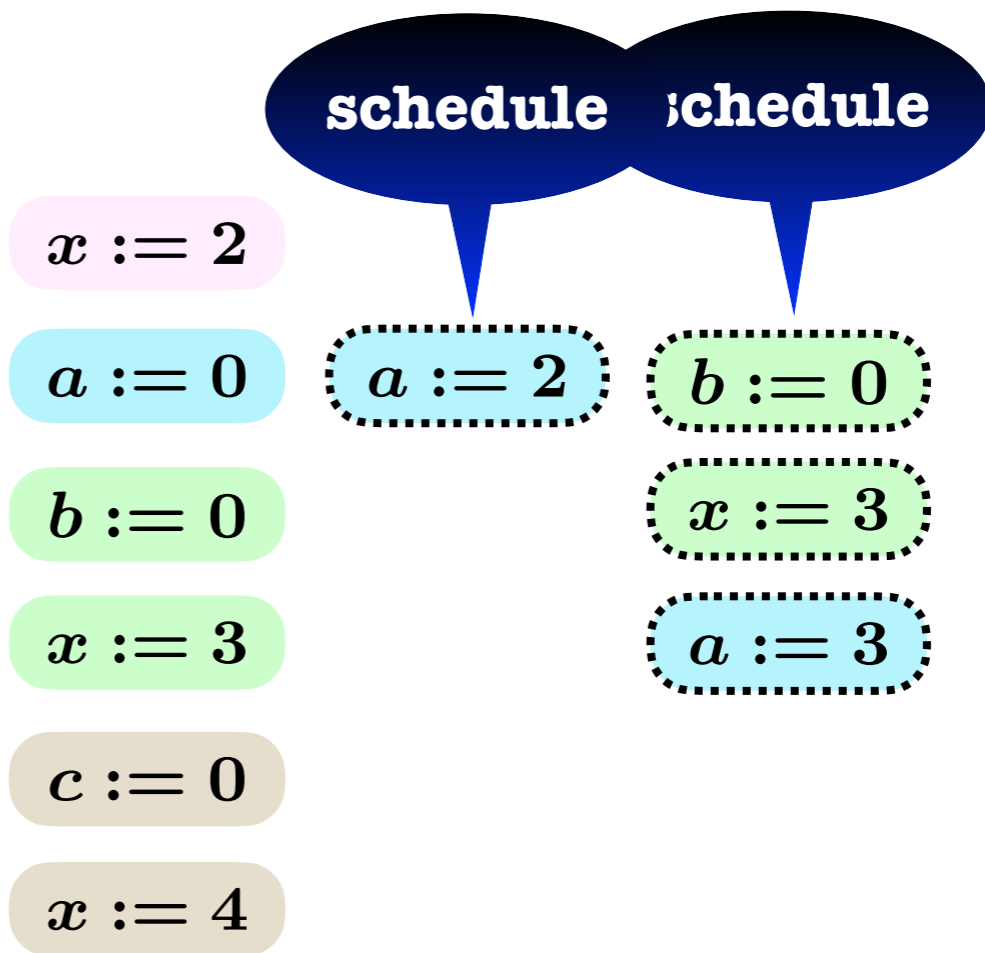
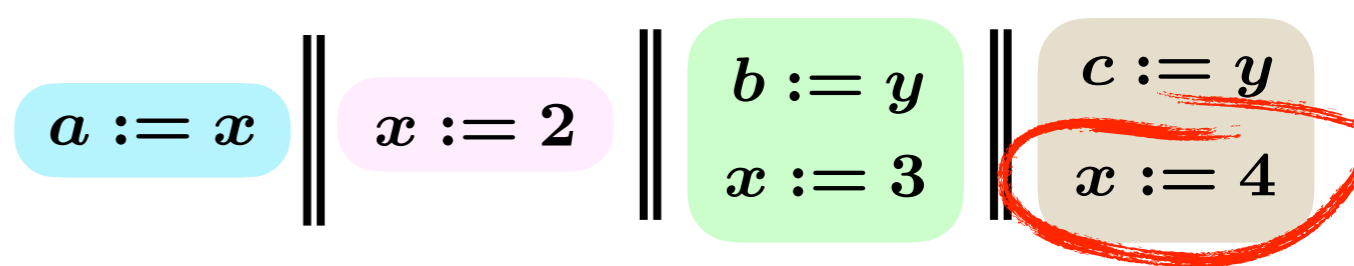
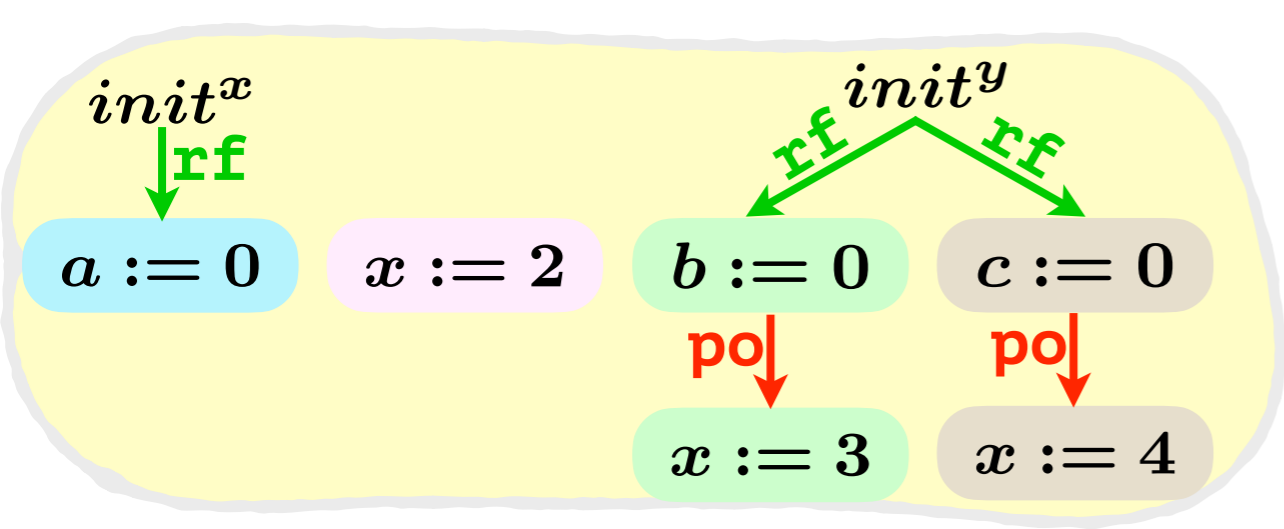
Postpone



Branch

&

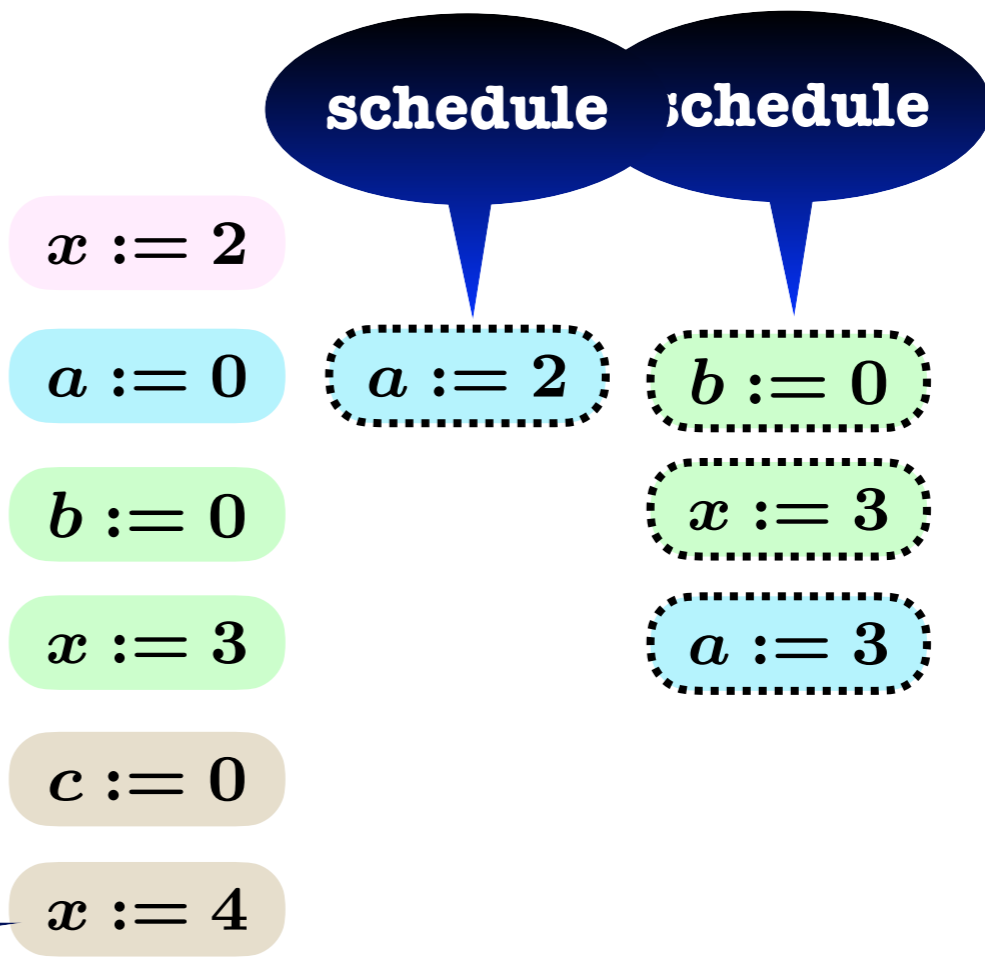
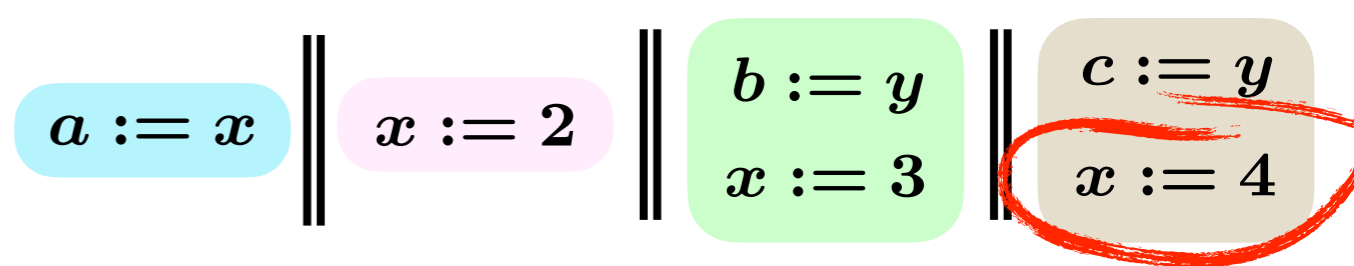
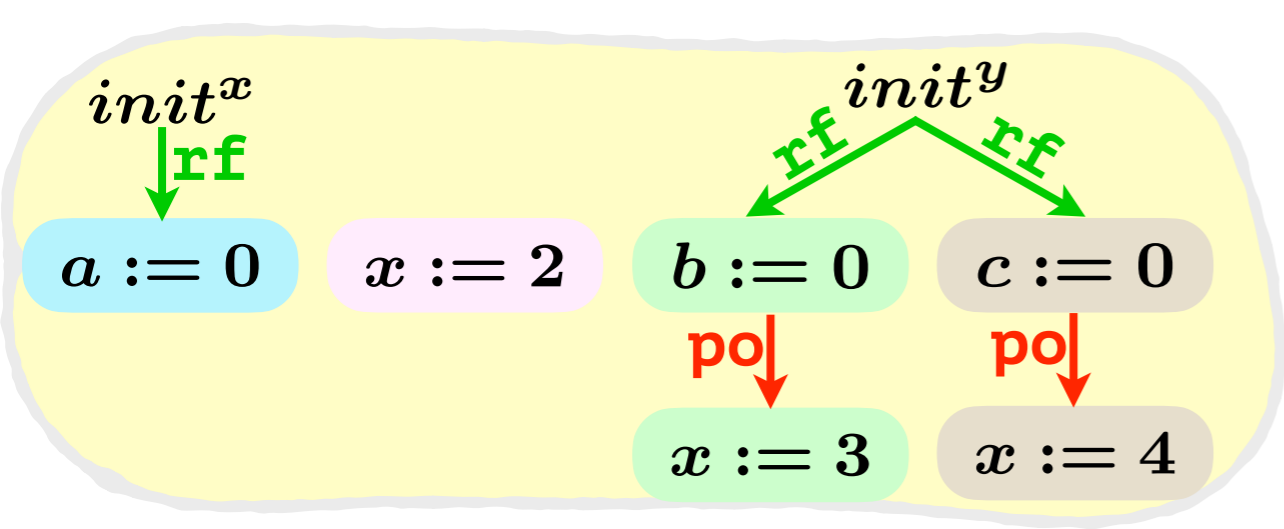
Postpone



Branch

&

Postpone

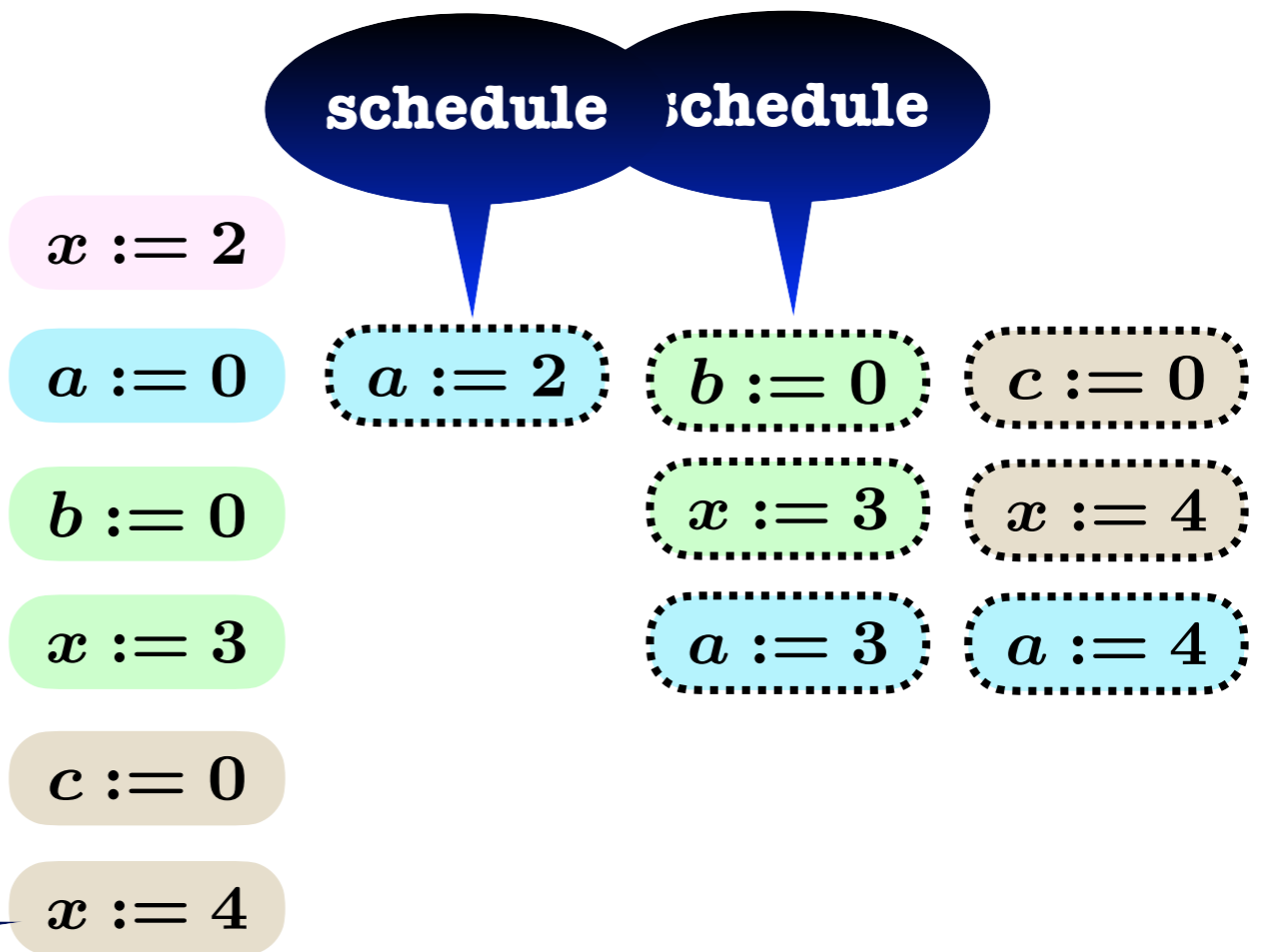
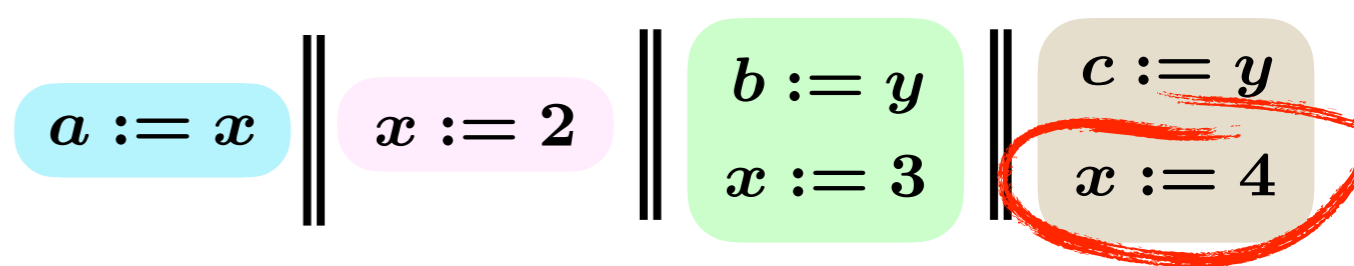
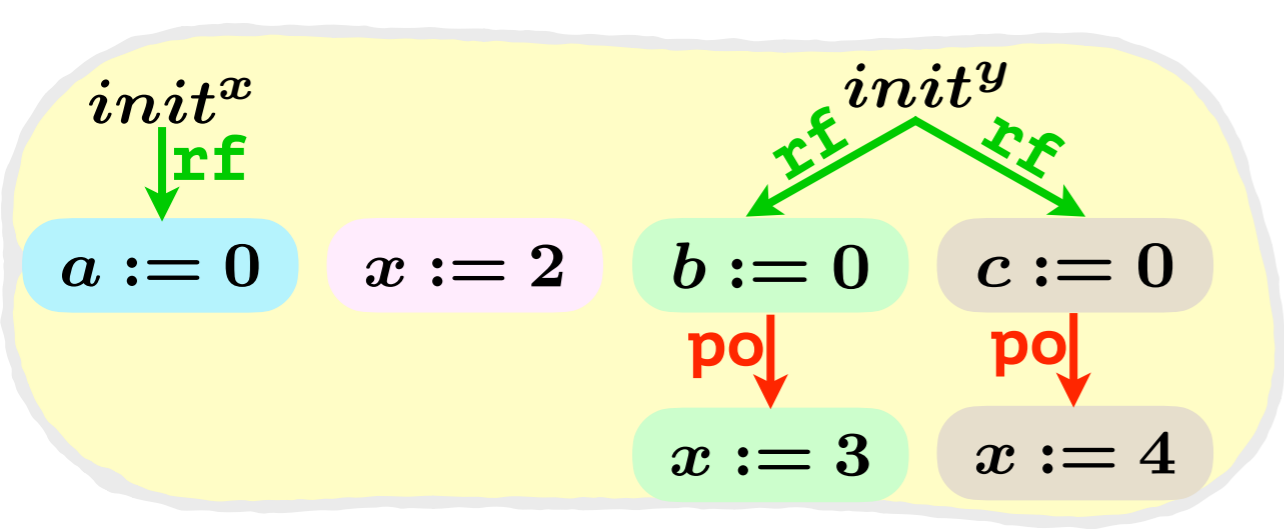


postponed write

Branch

&

Postpone

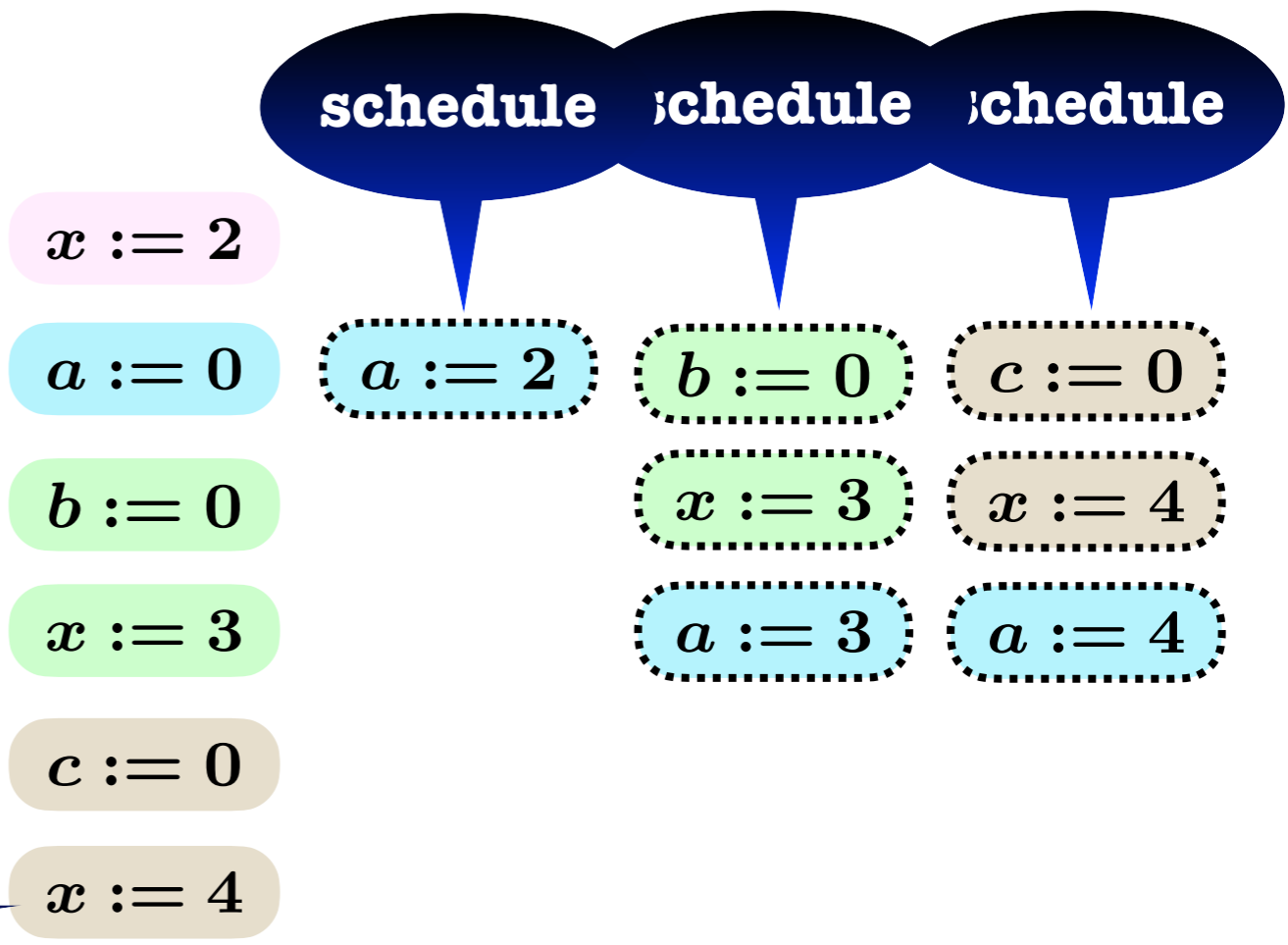
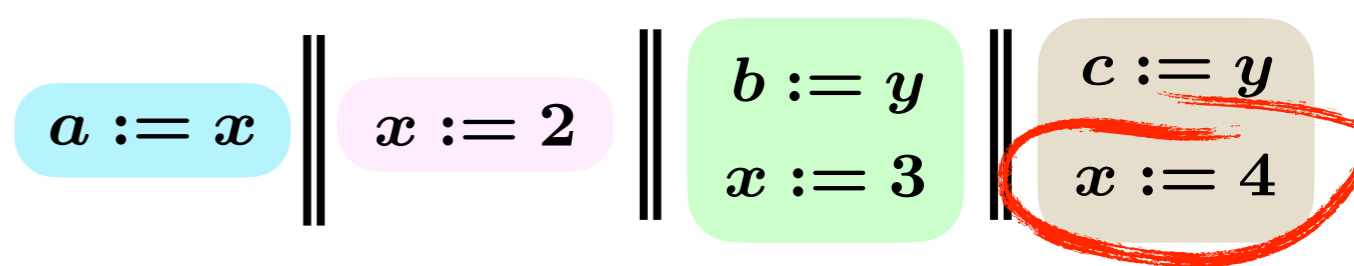
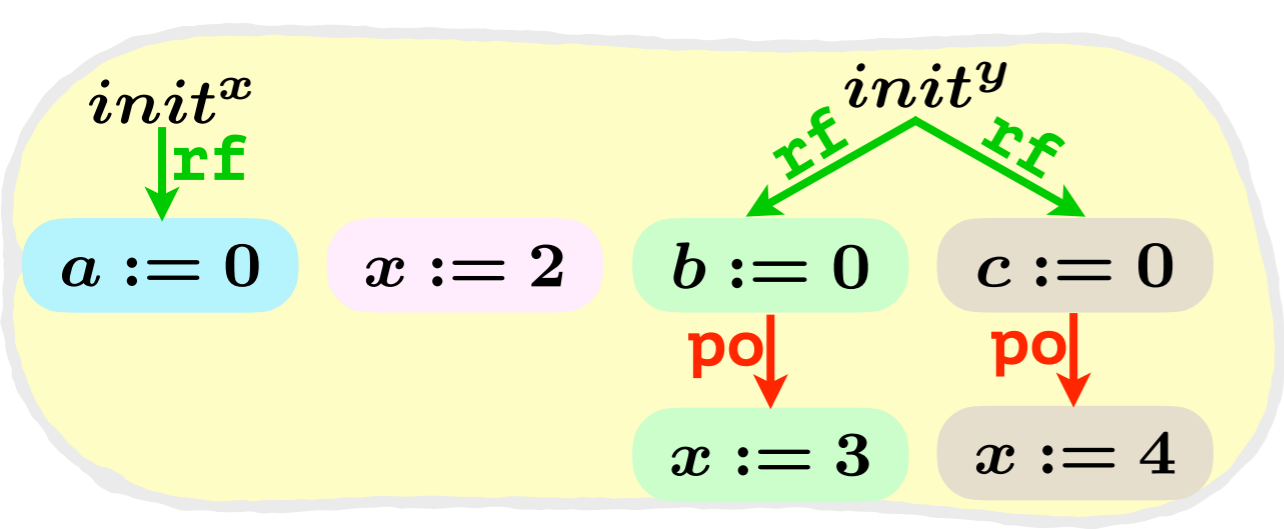


postponed write

Branch

&c

Postpone

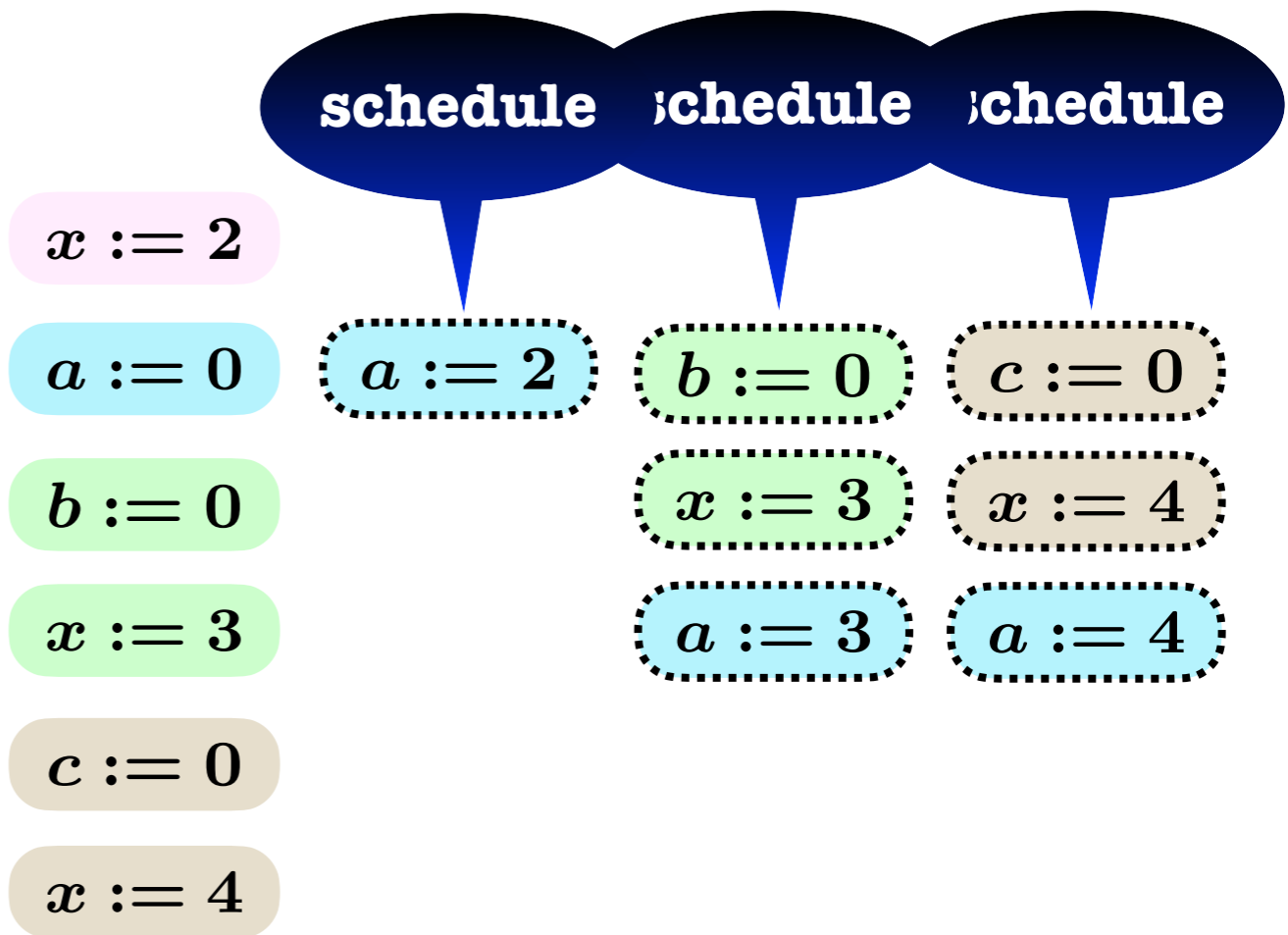
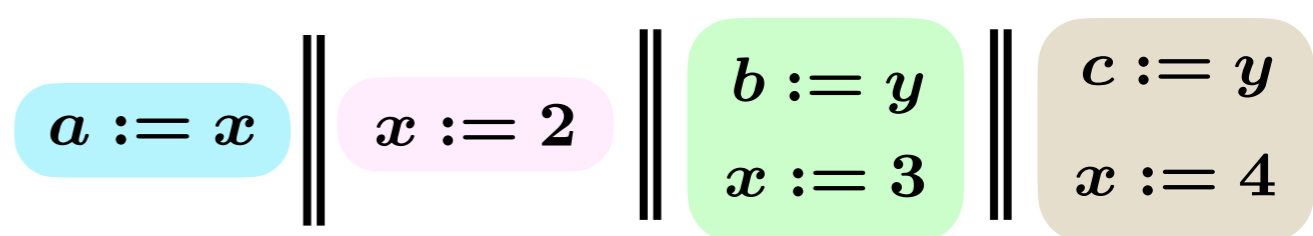
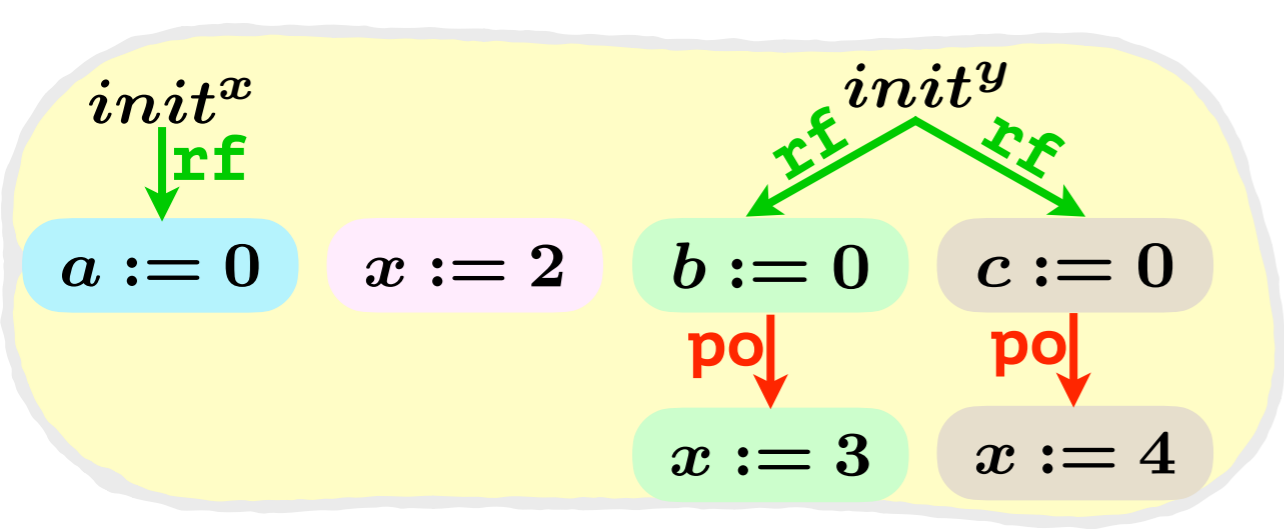


postponed write

Branch

&c

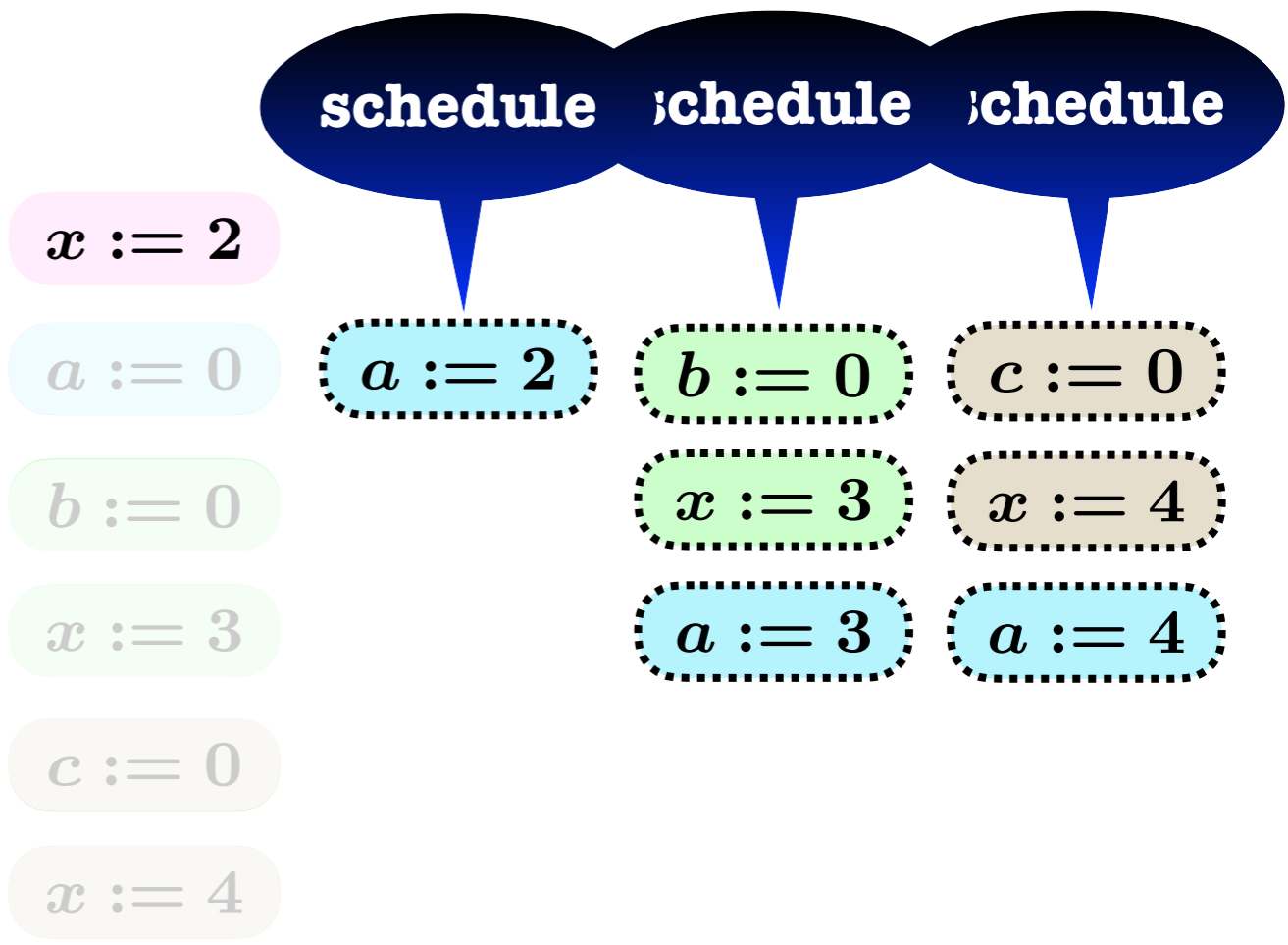
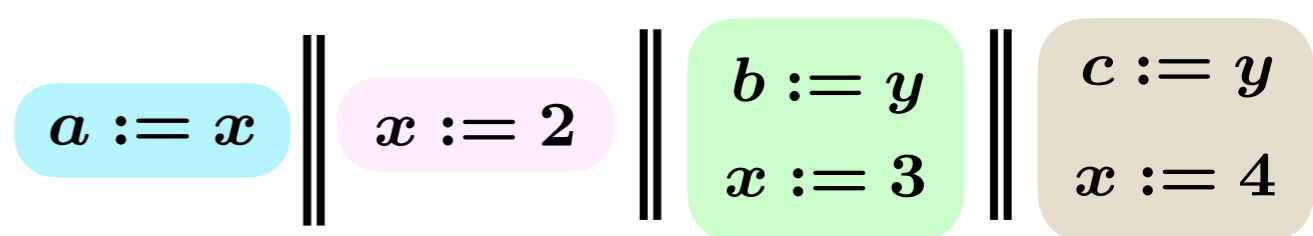
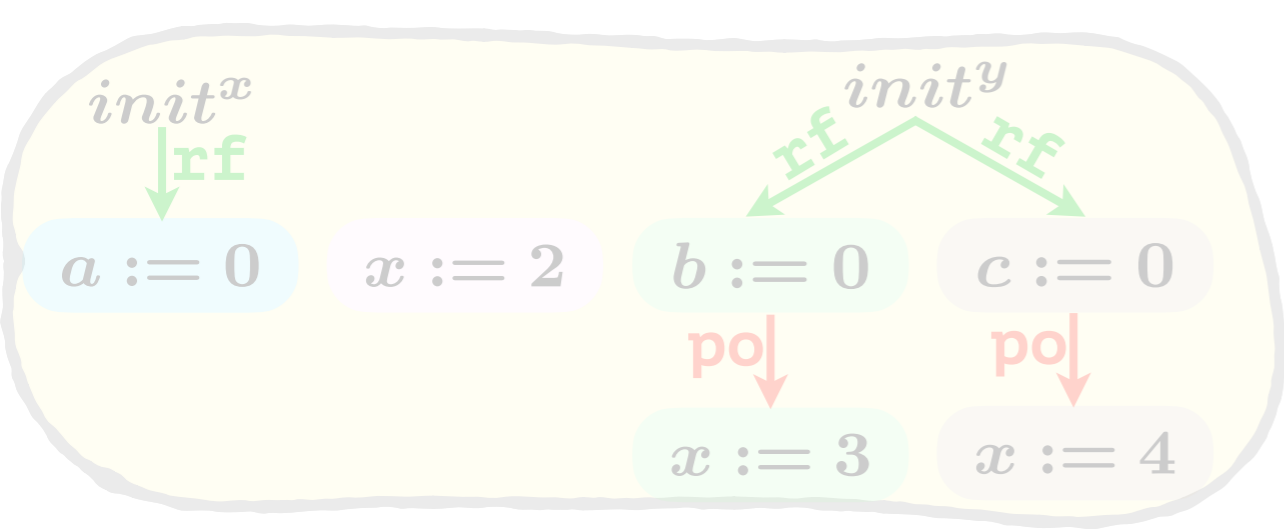
Postpone



Branch

&

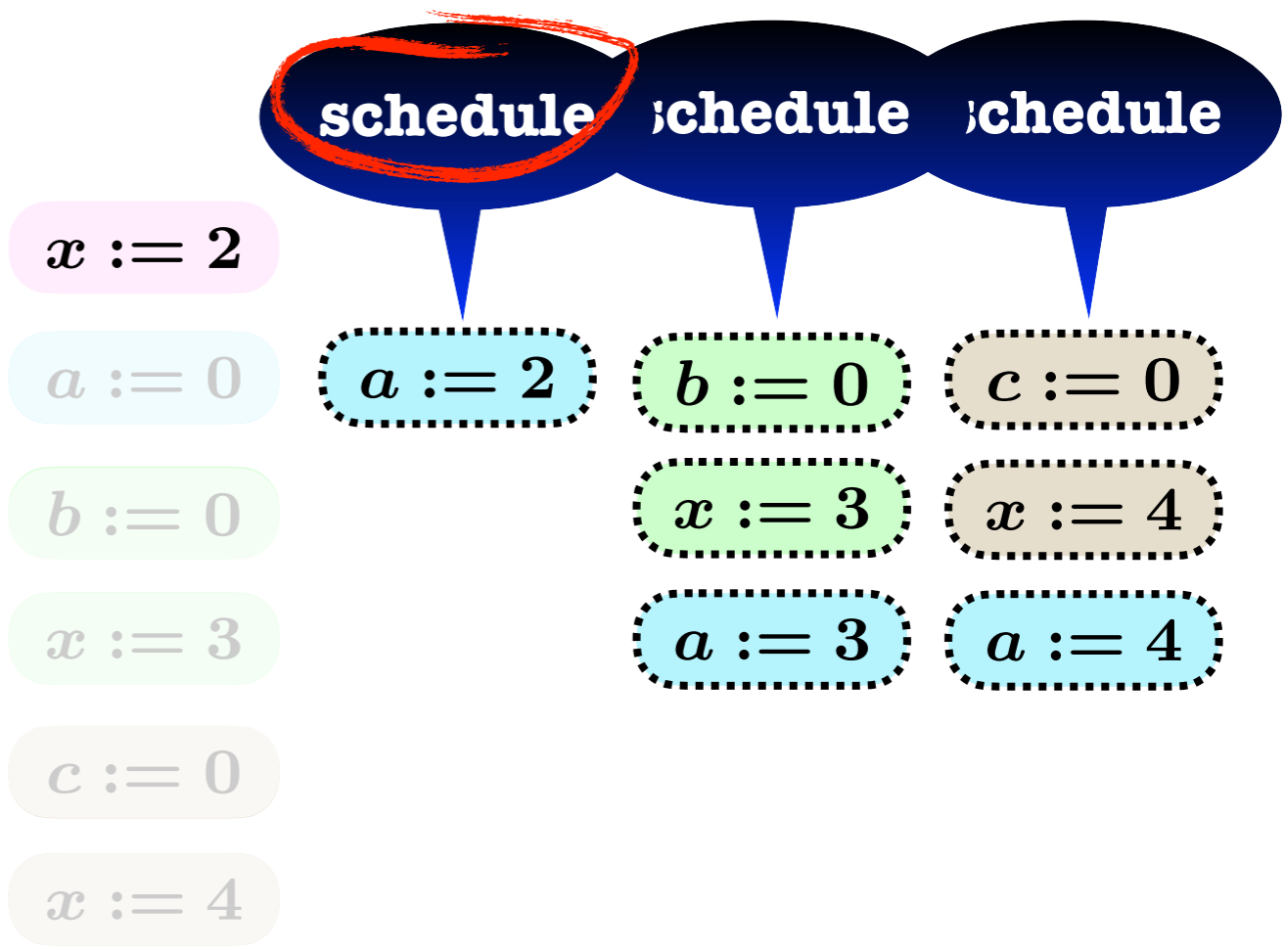
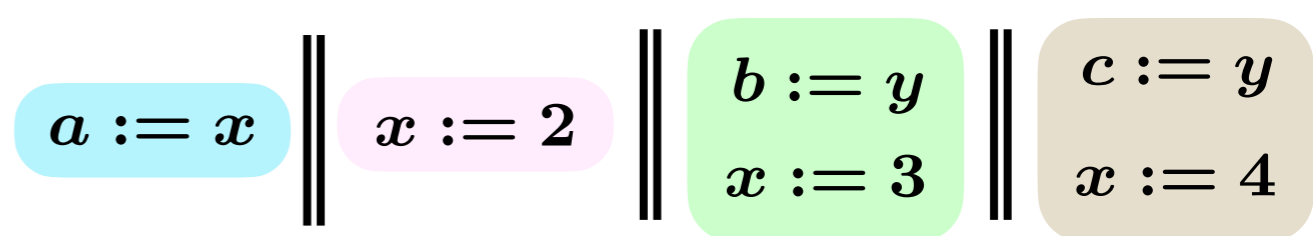
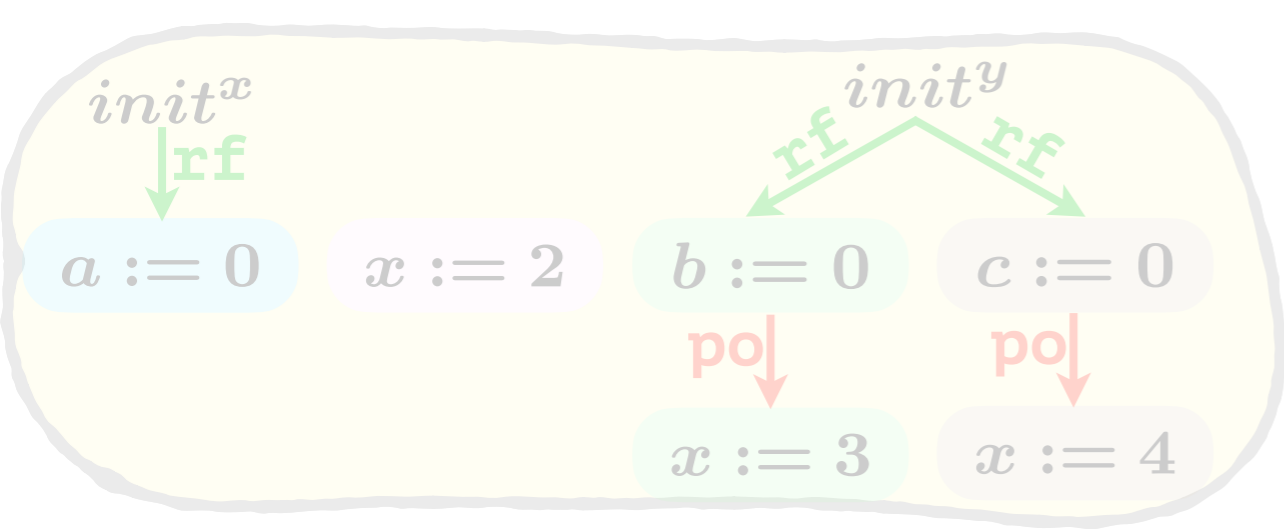
Postpone



Branch

&

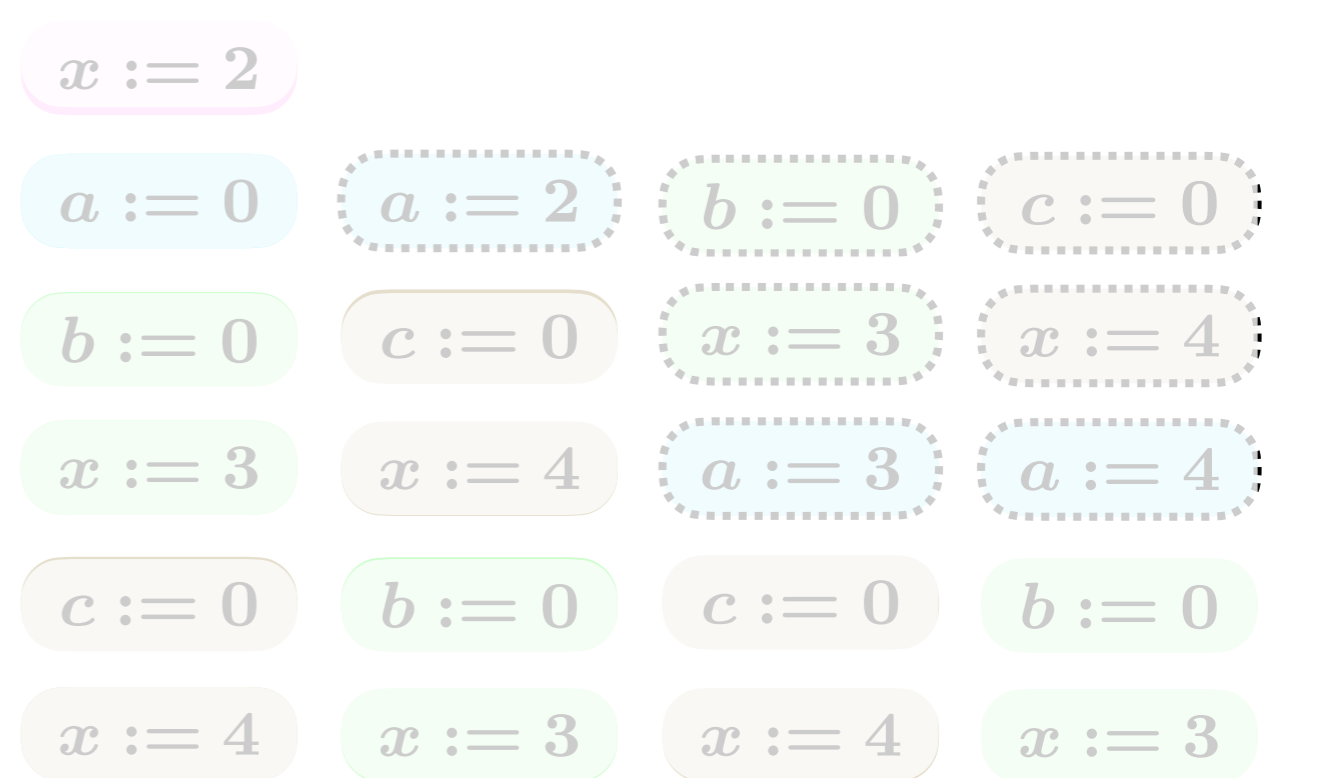
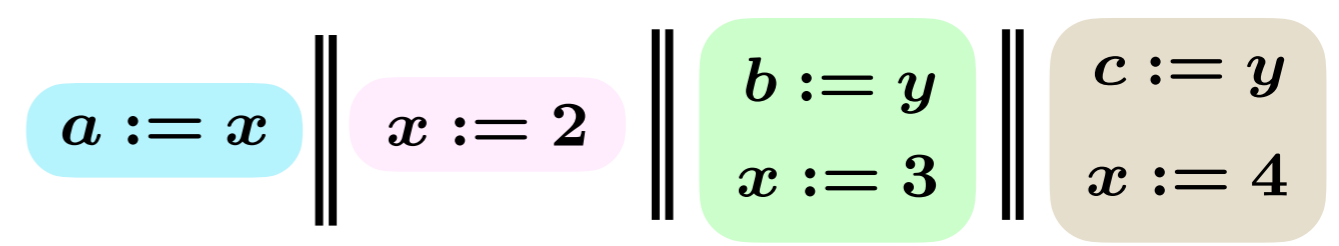
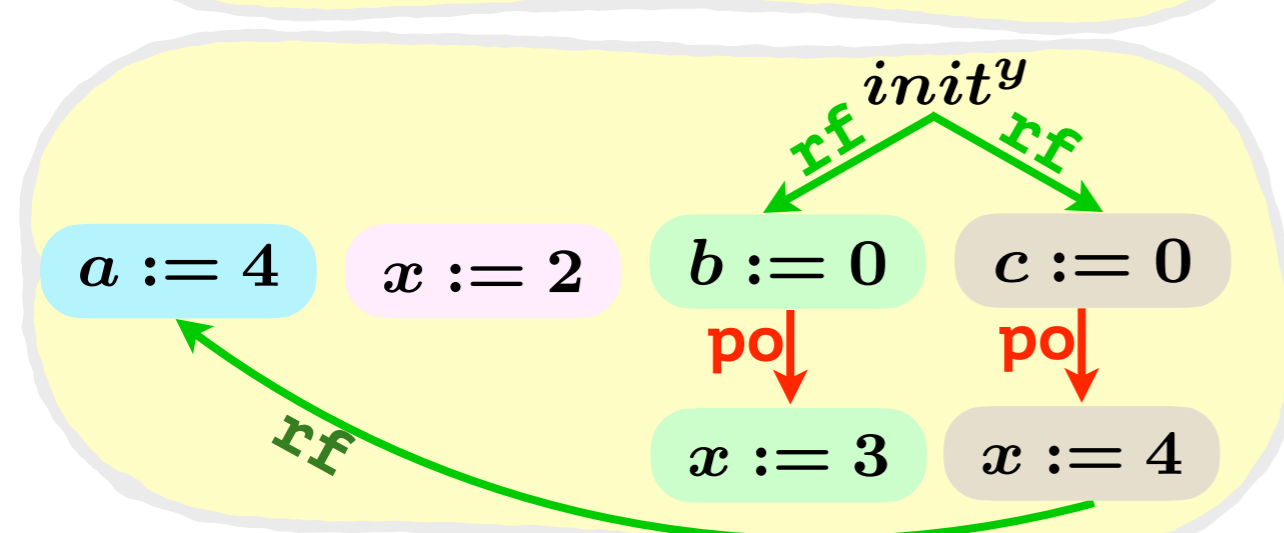
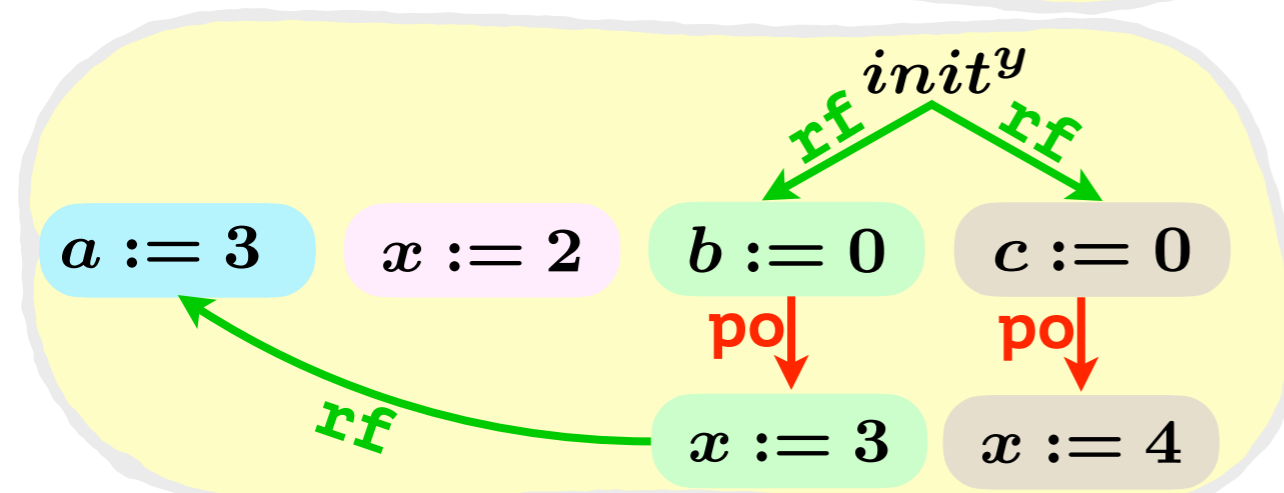
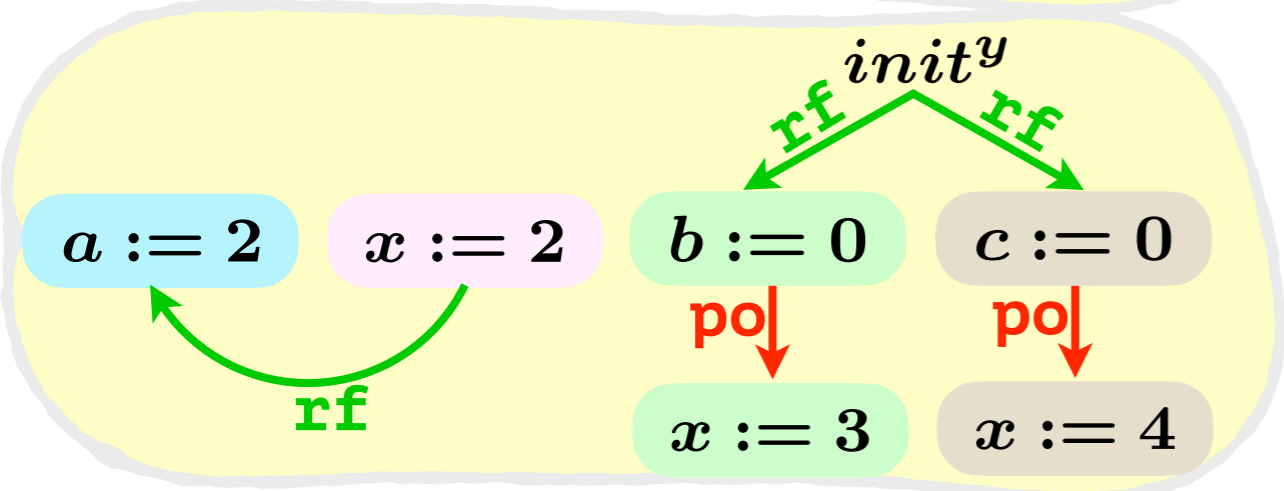
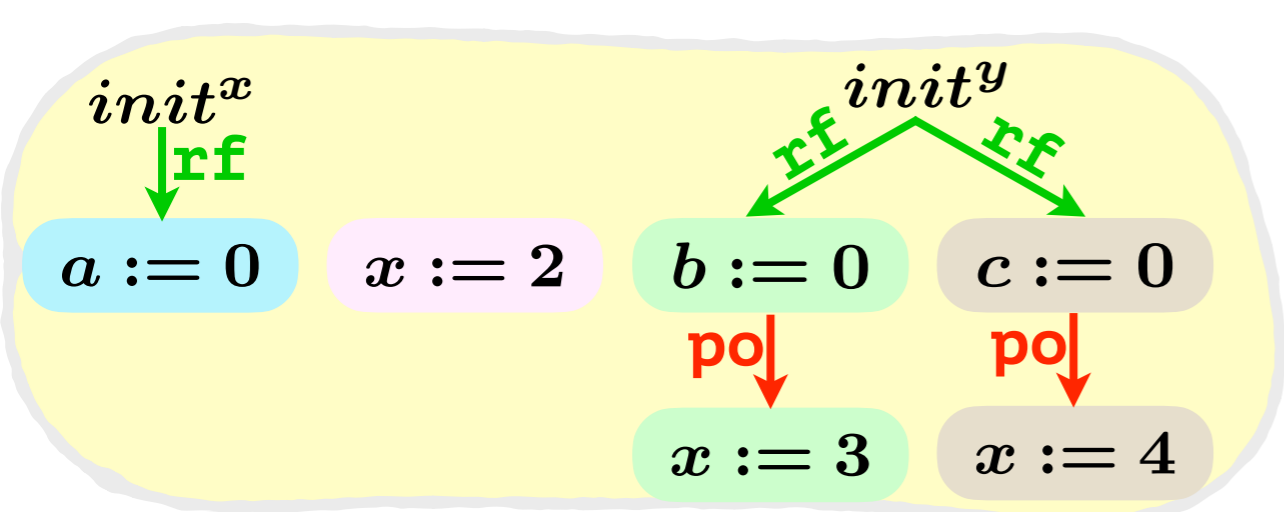
Postpone

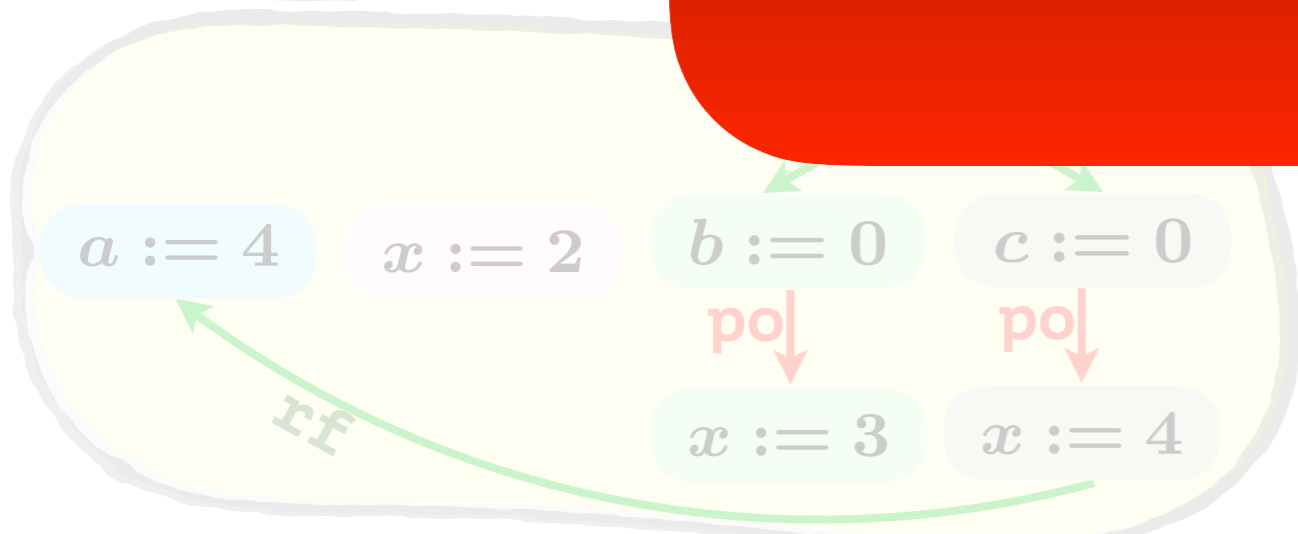
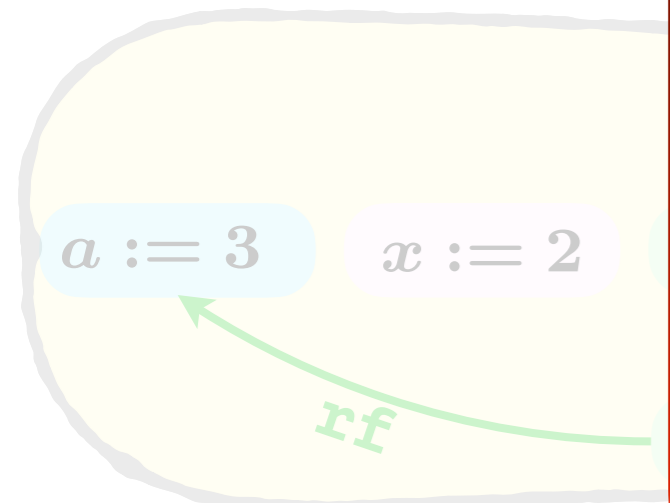
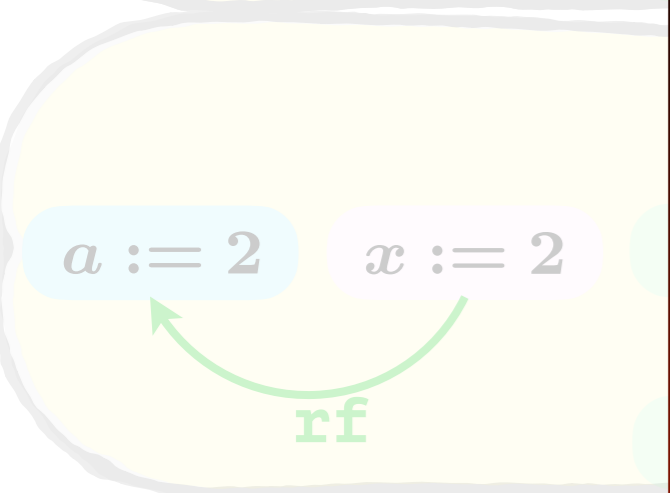
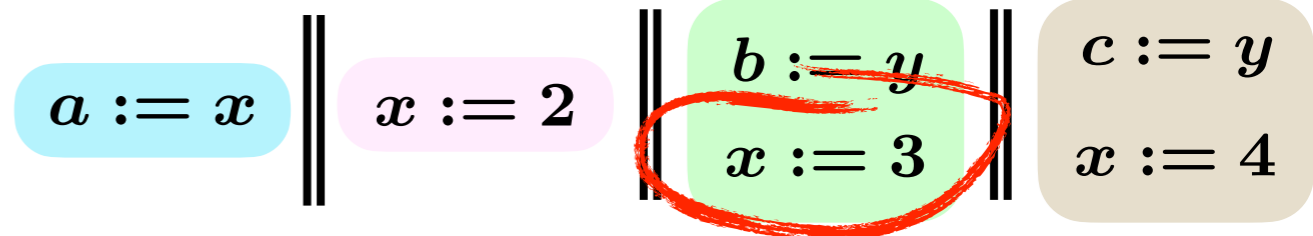
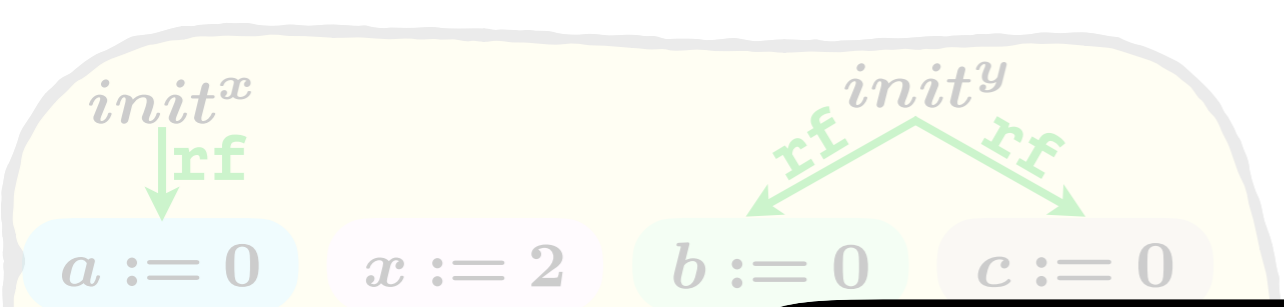


Branch

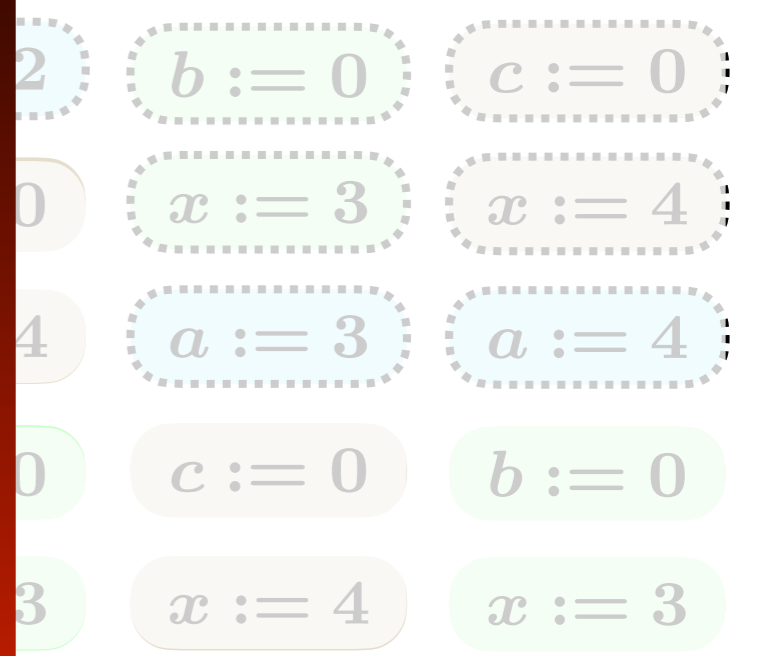
&

Postpone





- Sound
 - only valid traces
- Complete
 - all valid traces
- Optimal
 - different traces
- Deadlock-free



Outline

- **Memory Models**
 - **SC (Sequential Consistency)**
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - **Experiments**
- **Conclusions**

Outline

- **Memory Models**
 - **SC (Sequential Consistency)**
 - **RA (Release-Acquire)**
- **Verification**
 - **Dynamic Partial-Order Reduction (DPOR)**
 - **Experiments** <https://github.com/nidhugg/nidhugg>
- **Conclusions**

Tool	Wrong answer	Execs	Time	Average Execs <i>(Execs/Nums of tests)</i>	Average Time <i>(Time/Execs)</i>
CDSCHECKER	0	2 383 704	6m28s	295.59	0.16ms
RCMC/Rc11	0	727 942	14m00s	90.27	1.15ms
RCMC/WRC11	81	668 574	12m15s	82.91	1.10ms
TRACER	0	521 288	3m03s	64.64	0.35ms

Litmus Tests

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Indexer(13)	20	190	64	64	0.15s	0.62s	0.04s
Indexer(14)	20	3 075	512	512	2.72s	11.68s	0.38s
Indexer(15)	20	48 261	4 096	4 096	45.32s	4m22s	2.98s
Indexer(16)	20	740 889	<i>t/o</i>	32 768	12m21s	<i>t/o</i>	25.78s
Filesystem(16)	20	27	8	8	0.04s	0.05s	0.01s
Filesystem(19)	20	728	64	64	0.52s	2.02s	0.03s
Filesystem(22)	20	19 678	512	512	13.53s	2m29s	0.33s
Filesystem(25)	20	531 415	<i>t/o</i>	4096	7m04s	<i>t/o</i>	3.12s

Standard Benchmarks

Tool	Wrong answer	Execs	Time	Average Execs (Execs/Nums of tests)	Average Time (Time/Execs)
CDSCHECKER	0	2 383 704	6m28s	295.59	0.16ms
RCMC/Rc11	0	727 942	14m00s	90.27	1.15ms
RCMC/WRC11	81	668 574	12m15s	82.91	1.10ms
TRACER	0	521 288	3m03s	64.64	0.35ms

Litmus Tests

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Indexer(13)	20	190	64	64	0.15s	0.62s	0.04s
Indexer(14)	20	3 075	512	512	2.72s	11.68s	0.38s
Indexer(15)	20	48 261	4 096	4 096	45.32s	4m22s	2.98s
Indexer(16)	20	740 889	<i>t/o</i>	32 768	12m21s	<i>t/o</i>	25.78s
Filesystem(16)	20	27	8	8	0.04s	0.05s	0.01s
Filesystem(19)	20	728	64	64	0.52s	2.02s	0.03s
Filesystem(22)	20	19 678	512	512	13.53s	2m29s	0.33s
Filesystem(25)	20	531 415	<i>t/o</i>	4096	7m04s	<i>t/o</i>	3.12s

Standard Benchmarks

Tool	Wrong answer	Execs	Time	Average Execs <i>(Execs/Nums of tests)</i>	Average Time <i>(Time/Execs)</i>
CDSCHECKER	0	2 383 704	6m28s	295.59	0.16ms
RCMC/Rc11	0	727 942	14m00s	90.27	1.15ms
RCMC/WRC11	81	668 574	12m15s	82.91	1.10ms
TRACER	0	521 288	3m03s	64.64	0.35ms

Litmus Tests

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Indexer(13)	20	190	64	64	0.15s	0.62s	0.04s
Indexer(14)	20	3 075	512	512	2.72s	11.68s	0.38s
Indexer(15)	20	48 261	4 096	4 096	45.32s	4m22s	2.98s
Indexer(16)	20	740 889	<i>t/o</i>	32 768	12m21s	<i>t/o</i>	25.78s
Filesystem(16)	20	27	8	8	0.04s	0.05s	0.01s
Filesystem(19)	20	728	64	64	0.52s	2.02s	0.03s
Filesystem(22)	20	19 678	512	512	13.53s	2m29s	0.33s
Filesystem(25)	20	531 415	<i>t/o</i>	4096	7m04s	<i>t/o</i>	3.12s

Standard Benchmarks

Tool	Wrong answer	Execs	Time	Average Execs <i>(Execs/Nums of tests)</i>	Average Time <i>(Time/Execs)</i>
CDSCHECKER	0	2 383 704	6m28s	295.59	0.16ms
RCMC/Rc11	0	727 942	14m00s	90.27	1.15ms
RCMC/WRC11	81	668 574	12m15s	82.91	1.10ms
TRACER	0	521 288	3m03s	64.64	0.35ms

Litmus Tests

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Indexer(13)	20	190	64	64	0.15s	0.62s	0.04s
Indexer(14)	20	3 075	512	512	2.72s	11.68s	0.38s
Indexer(15)	20	48 261	4 096	4 096	45.32s	4m22s	2.98s
Indexer(16)	20	740 889	<i>t/o</i>	32 768	12m21s	<i>t/o</i>	25.78s
Filesystem(16)	20	27	8	8	0.04s	0.05s	0.01s
Filesystem(19)	20	728	64	64	0.52s	2.02s	0.03s
Filesystem(22)	20	19 678	512	512	13.53s	2m29s	0.33s
Filesystem(25)	20	531 415	<i>t/o</i>	4096	7m04s	<i>t/o</i>	3.12s

Standard Benchmarks

Tool	Wrong answer	Execs	Time	Average Execs (Execs/Nums of tests)	Average Time (Time/Execs)
CDSCHECKER	0	2 383 704	6m28s	295.59	0.16ms
RCMC/Rc11	0	727 942	14m00s	90.27	1.15ms
RCMC/WRC11	81	668 574	12m15s	82.91	1.10ms
TRACER	0	521 288	3m03s	64.64	0.35ms

Litmus Tests

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Indexer(13)	20	190	64	64	0.15s	0.62s	0.04s
Indexer(14)	20	3 075	512	512	2.72s	11.68s	0.38s
Indexer(15)	20	48 261	4 096	4 096	45.32s	4m22s	2.98s
Indexer(16)	20	740 889	t/o	32 768	12m21s	t/o	25.78s
Filesystem(16)	20	27	8	8	0.04s	0.05s	0.01s
Filesystem(19)	20	728	64	64	0.52s	2.02s	0.03s
Filesystem(22)	20	19 678	512	512	13.53s	2m29s	0.33s
Filesystem(25)	20	531 415	t/o	4096	7m04s	t/o	3.12s

Standard Benchmarks

Tool	Wrong answer	Execs	Time	Average Execs (Execs/Nums of tests)	Average Time (Time/Execs)
CDSCHECKER	0	2 383 704	6m28s	295.59	0.16ms
RCMC/Rc11	0	727 942	14m00s	90.27	1.15ms
RCMC/WRC11	81	668 574	12m15s	82.91	1.10ms
TRACER	0	521 288	3m03s	64.64	0.35ms

Litmus Tests

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Indexer(13)	20	190	64	64	0.15s	0.62s	0.04s
Indexer(14)	20	3 075	512	512	2.72s	11.68s	0.38s
Indexer(15)	20	48 261	4 096	4 096	45.32s	4m22s	2.98s
Indexer(16)	20	740 889	t/o	32 768	12m21s	t/o	25.78s
Filesystem(16)	20	27	8	8	0.04s	0.05s	0.01s
Filesystem(19)	20	728	64	64	0.52s	2.02s	0.03s
Filesystem(22)	20	19 678	512	512	13.53s	2m29s	0.33s
Filesystem(25)	20	531 415	t/o	4096	7m04s	t/o	3.12s

Standard Benchmarks

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Pthread_demo	10	184 758	184 756	184 756	24.96s	1m03s	18.00s
Gcd	8	8 814 044	1 162 333	1 162 333	38m20s	7m07s	2m48s
Fibonacci	6	<i>t/o</i>	525 630	525 630	<i>t/o</i>	31.06s	57.95s
Szymanski	6	<i>t/o</i>	26 037 490	12 209 410	<i>t/o</i>	44m52s	14m54s
Dekker	10	7 306 447	3 121 870	3 121 870	15m25s	5m12s	4m52s
Lamport	8	<i>t/o</i>	6 580 870	3 372 868	<i>t/o</i>	14m40s	6m58s
Sigma(5)	5	1 279	945	120	0.09s	0.16s	0.01s
Peterson	6	<i>t/o</i>	1 897 228	1 897 228	<i>t/o</i>	3m16s	3m15s
Stack_true	12	2 704 157	2 704 156	2 704 156	19m03s	54m25s	10m12s
Queue_ok	12	581 790	<i>t/o</i>	362 880	33m27s	<i>t/o</i>	12m52s

SV-COMP

Program	Executions Explored			Total Running Time		
	CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
N_writers_a_reader(7)	8	40 320	8	0.01s	0.46s	0.00s
N_writers_a_reader(8)	9	362 880	9	0.01s	4.19s	0.00s
N_writers_a_reader(9)	10	3 628 800	10	0.01s	46.13s	0.00s
N_writers_a_reader(10)	11	39 916 800	11	0.01s	9m35s	0.00s
Redundant_co(5)	581	16 632	91	0.03s	0.39s	0.01s
Redundant_co(10)	10 631	42 678 636	331	0.64s	23m56s	0.02s
Redundant_co(15)	59 056	<i>t/o</i>	721	4.57s	<i>t/o</i>	0.06s
Redundant_co(20)	197 231	<i>t/o</i>	1 261	20.27s	<i>t/o</i>	0.14s

Synthetic Programs

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Pthread_demo	10	184 758	184 756	184 756	24.96s	1m03s	18.00s
Gcd	8	8 814 044	1 162 333	1 162 333	38m20s	7m07s	2m48s
Fibonacci	6	<i>t/o</i>	525 630	525 630	<i>t/o</i>	31.06s	57.95s
Szymanski	6	<i>t/o</i>	26 037 490	12 209 410	<i>t/o</i>	44m52s	14m54s
Dekker	10	7 306 447	3 121 870	3 121 870	15m25s	5m12s	4m52s
Lamport	8	<i>t/o</i>	6 580 870	3 372 868	<i>t/o</i>	14m40s	6m58s
Sigma(5)	5	1 279	945	120	0.09s	0.16s	0.01s
Peterson	6	<i>t/o</i>	1 897 228	1 897 228	<i>t/o</i>	3m16s	3m15s
Stack_true	12	2 704 157	2 704 156	2 704 156	19m03s	54m25s	10m12s
Queue_ok	12	581 790	<i>t/o</i>	362 880	33m27s	<i>t/o</i>	12m52s

SV-COMP

Program	Executions Explored			Total Running Time		
	CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
N_writers_a_reader(7)	8	40 320	8	0.01s	0.46s	0.00s
N_writers_a_reader(8)	9	362 880	9	0.01s	4.19s	0.00s
N_writers_a_reader(9)	10	3 628 800	10	0.01s	46.13s	0.00s
N_writers_a_reader(10)	11	39 916 800	11	0.01s	9m35s	0.00s
Redundant_co(5)	581	16 632	91	0.03s	0.39s	0.01s
Redundant_co(10)	10 631	42 678 636	331	0.64s	23m56s	0.02s
Redundant_co(15)	59 056	<i>t/o</i>	721	4.57s	<i>t/o</i>	0.06s
Redundant_co(20)	197 231	<i>t/o</i>	1 261	20.27s	<i>t/o</i>	0.14s

Synthetic Programs

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
Pthread_demo	10	184 758	184 756	184 756	24.96s	1m03s	18.00s
Gcd	8	8 814 044	1 162 333	1 162 333	38m20s	7m07s	2m48s
Fibonacci	6	<i>t/o</i>	525 630	525 630	<i>t/o</i>	31.06s	57.95s
Szymanski	6	<i>t/o</i>	26 037 490	12 209 410	<i>t/o</i>	44m52s	14m54s
Dekker	10	7 306 447	3 121 870	3 121 870	15m25s	5m12s	4m52s
Lamport	8	<i>t/o</i>	6 580 870	3 372 868	<i>t/o</i>	14m40s	6m58s
Sigma(5)	5	1 279	945	120	0.09s	0.16s	0.01s
Peterson	6	<i>t/o</i>	1 897 228	1 897 228	<i>t/o</i>	3m16s	3m15s
Stack_true	12	2 704 157	2 704 156	2 704 156	19m03s	54m25s	10m12s
Queue_ok	12	581 790	<i>t/o</i>	362 880	33m27s	<i>t/o</i>	12m52s

SV-COMP

Program	Executions Explored			Total Running Time		
	CDSCHECKER	RCMC	TRACER	CDSCHECKER	RCMC	TRACER
N_writers_a_reader(7)	8	40 320	8	0.01s	0.46s	0.00s
N_writers_a_reader(8)	9	362 880	9	0.01s	4.19s	0.00s
N_writers_a_reader(9)	10	3 628 800	10	0.01s	46.13s	0.00s
N_writers_a_reader(10)	11	39 916 800	11	0.01s	9m35s	0.00s
Redundant_co(5)	581	16 632	91	0.03s	0.39s	0.01s
Redundant_co(10)	10 631	42 678 636	331	0.64s	23m56s	0.02s
Redundant_co(15)	59 056	<i>t/o</i>	721	4.57s	<i>t/o</i>	0.06s
Redundant_co(20)	197 231	<i>t/o</i>	1 261	20.27s	<i>t/o</i>	0.14s

Synthetic Programs

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RMC	TRACER	CDSCHECKER	RMC	TRACER
Sigma(6)	6	25 357	10 395	720	2.20s	1.96s	0.04s
Sigma(7)	7	605 714	135 135	5 040	1m02s	29.06s	0.40s
Sigma(8)	8	16 667 637	2 027 025	40 320	33m18s	8m02s	3.28s
Sigma(9)	9	<i>t/o</i>	<i>t/o</i>	362 880	<i>t/o</i>	<i>t/o</i>	33.59s
Control_flow(6)	0	896	55 440	77	0.09s	1.97s	0.01s
Control_flow(8)	0	4 608	11 007 360	273	0.53s	7m58s	0.03s
Control_flow(10)	0	22 528	<i>t/o</i>	1 045	3.27s	<i>t/o</i>	0.16s
Control_flow(12)	0	106 496	<i>t/o</i>	4 121	19.10s	<i>t/o</i>	0.79s
Exponential_bug(6)	6	983 386	1 203 446	15 601	1m18s	56.75s	0.96s
Exponential_bug(7)	7	2 250 290	2 833 112	22 841	3m13s	2m26s	1.46s
Exponential_bug(8)	8	4 844 378	6 158 718	32 313	7m15s	5m28s	2.23s
Exponential_bug(9)	9	9 896 954	12 526 576	44 428	15m29s	11m48s	3.22s

Parameterized Programs

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RMC	TRACER	CDSCHECKER	RMC	TRACER
Linux_locks(2)	6	47	21	21	0.03s	0.02s	0.01s
Linux_locks(3)	6	14 187 799	412 814	412 814	16m01s	36.36s	33.21s
Ticket_mutex_c11(3)	10	8 054	4 026	4 026	0.69s	50.76s	0.25s
Correia_Ramalhete(3)	5	5 355	5 355	5 355	1.10s	0.77s	0.92s

Concurrent Data Structures

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RMC	TRACER	CDSCHECKER	RMC	TRACER
Sigma(6)	6	25 357	10 395	720	2.20s	1.96s	0.04s
Sigma(7)	7	605 714	135 135	5 040	1m02s	29.06s	0.40s
Sigma(8)	8	16 667 637	2 027 025	40 320	33m18s	8m02s	3.28s
Sigma(9)	9	<i>t/o</i>	<i>t/o</i>	362 880	<i>t/o</i>	<i>t/o</i>	33.59s
Control_flow(6)	0	896	55 440	77	0.09s	1.97s	0.01s
Control_flow(8)	0	4 608	11 007 360	273	0.53s	7m58s	0.03s
Control_flow(10)	0	22 528	<i>t/o</i>	1 045	3.27s	<i>t/o</i>	0.16s
Control_flow(12)	0	106 496	<i>t/o</i>	4 121	19.10s	<i>t/o</i>	0.79s
Exponential_bug(6)	6	983 386	1 203 446	15 601	1m18s	56.75s	0.96s
Exponential_bug(7)	7	2 250 290	2 833 112	22 841	3m13s	2m26s	1.46s
Exponential_bug(8)	8	4 844 378	6 158 718	32 313	7m15s	5m28s	2.23s
Exponential_bug(9)	9	9 896 954	12 526 576	44 428	15m29s	11m48s	3.22s

Parameterized Programs

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RMC	TRACER	CDSCHECKER	RMC	TRACER
Linux_locks(2)	6	47	21	21	0.03s	0.02s	0.01s
Linux_locks(3)	6	14 187 799	412 814	412 814	16m01s	36.36s	33.21s
Ticket_mutex_c11(3)	10	8 054	4 026	4 026	0.69s	50.76s	0.25s
Correia_Ramalhete(3)	5	5 355	5 355	5 355	1.10s	0.77s	0.92s

Concurrent Data Structures

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RMC	TRACER	CDSCHECKER	RMC	TRACER
Sigma(6)	6	25 357	10 395	720	2.20s	1.96s	0.04s
Sigma(7)	7	605 714	135 135	5 040	1m02s	29.06s	0.40s
Sigma(8)	8	16 667 637	2 027 025	40 320	33m18s	8m02s	3.28s
Sigma(9)	9	<i>t/o</i>	<i>t/o</i>	362 880	<i>t/o</i>	<i>t/o</i>	33.59s
Control_flow(6)	0	896	55 440	77	0.09s	1.97s	0.01s
Control_flow(8)	0	4 608	11 007 360	273	0.53s	7m58s	0.03s
Control_flow(10)	0	22 528	<i>t/o</i>	1 045	3.27s	<i>t/o</i>	0.16s
Control_flow(12)	0	106 496	<i>t/o</i>	4 121	19.10s	<i>t/o</i>	0.79s
Exponential_bug(6)	6	983 386	1 203 446	15 601	1m18s	56.75s	0.96s
Exponential_bug(7)	7	2 250 290	2 833 112	22 841	3m13s	2m26s	1.46s
Exponential_bug(8)	8	4 844 378	6 158 718	32 313	7m15s	5m28s	2.23s
Exponential_bug(9)	9	9 896 954	12 526 576	44 428	15m29s	11m48s	3.22s

Parameterized Programs

Program	LB	Executions Explored			Total Running Time		
		CDSCHECKER	RMC	TRACER	CDSCHECKER	RMC	TRACER
Linux_locks(2)	6	47	21	21	0.03s	0.02s	0.01s
Linux_locks(3)	6	14 187 799	412 814	412 814	16m01s	36.36s	33.21s
Ticket_mutex_c11(3)	10	8 054	4 026	4 026	0.69s	50.76s	0.25s
Correia_Ramalhete(3)	5	5 355	5 355	5 355	1.10s	0.77s	0.92s

Concurrent Data Structures

Program	LB	Executions Explored		Total Running Time	
		CDSCHECKER	TRACER	CDSCHECKER	TRACER
Barrier(3)	10	62 649	31 944	5.45s	1.88s
Seqlock(3)	5	17 792	14 864	1.17s	0.77s
MPMC_queue(3)	5	621 882	239 254	2m11s	44.50s
MCS_lock(2)	5	92 210	70 072	8.91s	5.19s
Cliffc_hashtable(4)	0	9 520	4 576	2.68s	0.94s
Concurrent_hashmap(4)	0	110	42	0.01s	0.00s
Chase_Lev_deque(2) (deadlock)	0	162 306	20 852	20.65s	2.10s
SPSC_queue(2) (deadlock)	3	754	57	0.18s	0.01s

Concurrent Data Structures

Program	LB	Executions Explored		Total Running Time	
		CDSCHECKER	TRACER	CDSCHECKER	TRACER
Tidex(3)	5	4 676	748	0.37s	0.04s
Tidex_nps(2)	5	10 257	10 254	6m15s	22.58s
CLH_c11(3)	10	2 562	732	0.22s	0.06s
CLH_rwlock_c11(3)	10	20	6	0.01s	0.00s
MPSC_c11(3)	10	12 937	4 824	1.20s	0.36s
Correia_Ramalhete_turn(3) (mutex broken)	2	441 494	96 184	1m17s	12.78s

Concurrent Data Structures

Program	LB	Executions Explored		Total Running Time	
		CDSCHECKER	TRACER	CDSCHECKER	TRACER
Barrier(3)	10	62 649	31 944	5.45s	1.88s
Seqlock(3)	5	17 792	14 864	1.17s	0.77s
MPMC_queue(3)	5	621 882	239 254	2m11s	44.50s
MCS_lock(2)	5	92 210	70 072	8.91s	5.19s
Cliffc_hashtable(4)	0	9 520	4 576	2.68s	0.94s
Concurrent_hashmap(4)	0	110	42	0.01s	0.00s
Chase_Lev_deque(2) (deadlock)	0	162 306	20 852	20.65s	2.10s
SPSC_queue(2) (deadlock)	3	754	57	0.18s	0.01s

Concurrent Data Structures

Program	LB	Executions Explored		Total Running Time	
		CDSCHECKER	TRACER	CDSCHECKER	TRACER
Tidex(3)	5	4 676	748	0.37s	0.04s
Tidex_nps(2)	5	10 257	10 254	6m15s	22.58s
CLH_c11(3)	10	2 562	732	0.22s	0.06s
CLH_rwlock_c11(3)	10	20	6	0.01s	0.00s
MPSC_c11(3)	10	12 937	4 824	1.20s	0.36s
Correia_Ramalhete_turn(3) (mutex broken)	2	441 494	96 184	1m17s	12.78s

Concurrent Data Structures

Program	LB	Executions Explored		Total Running Time	
		CDSCHECKER	TRACER	CDSCHECKER	TRACER
Barrier(3)	10	62 649	31 944	5.45s	1.88s
Seqlock(3)	5	17 792	14 864	1.17s	0.77s
MPMC_queue(3)	5	621 882	239 254	2m11s	44.50s
MCS_lock(2)	5	92 210	70 072	8.91s	5.19s
Cliffc_hashtable(4)	0	9 520	4 576	2.68s	0.94s
Concurrent_hashmap(4)	0	110	42	0.01s	0.00s
Chase_Lev_deque(2) (deadlock)	0	162 306	20 852	20.65s	2.10s
SPSC_queue(2) (deadlock)	3	754	57	0.18s	0.01s

Concurrent Data Structures

Program	LB	Executions Explored		Total Running Time	
		CDSCHECKER	TRACER	CDSCHECKER	TRACER
Tidex(3)	5	4 676	748	0.37s	0.04s
Tidex_nps(2)	5	10 257	10 254	6m15s	22.58s
CLH_c11(3)	10	2 562	732	0.22s	0.06s
CLH_rwlock_c11(3)	10	20	6	0.01s	0.00s
MPSC_c11(3)	10	12 937	4 824	1.20s	0.36s
Correia_Ramalhete_turn(3) (mutex broken)	2	441 494	96 184	1m17s	12.78s

Concurrent Data Structures

DPOR

Conclusions

- **(Optimal) DPOR**
- **Release-Acquire Semantics**
- **Experiments**

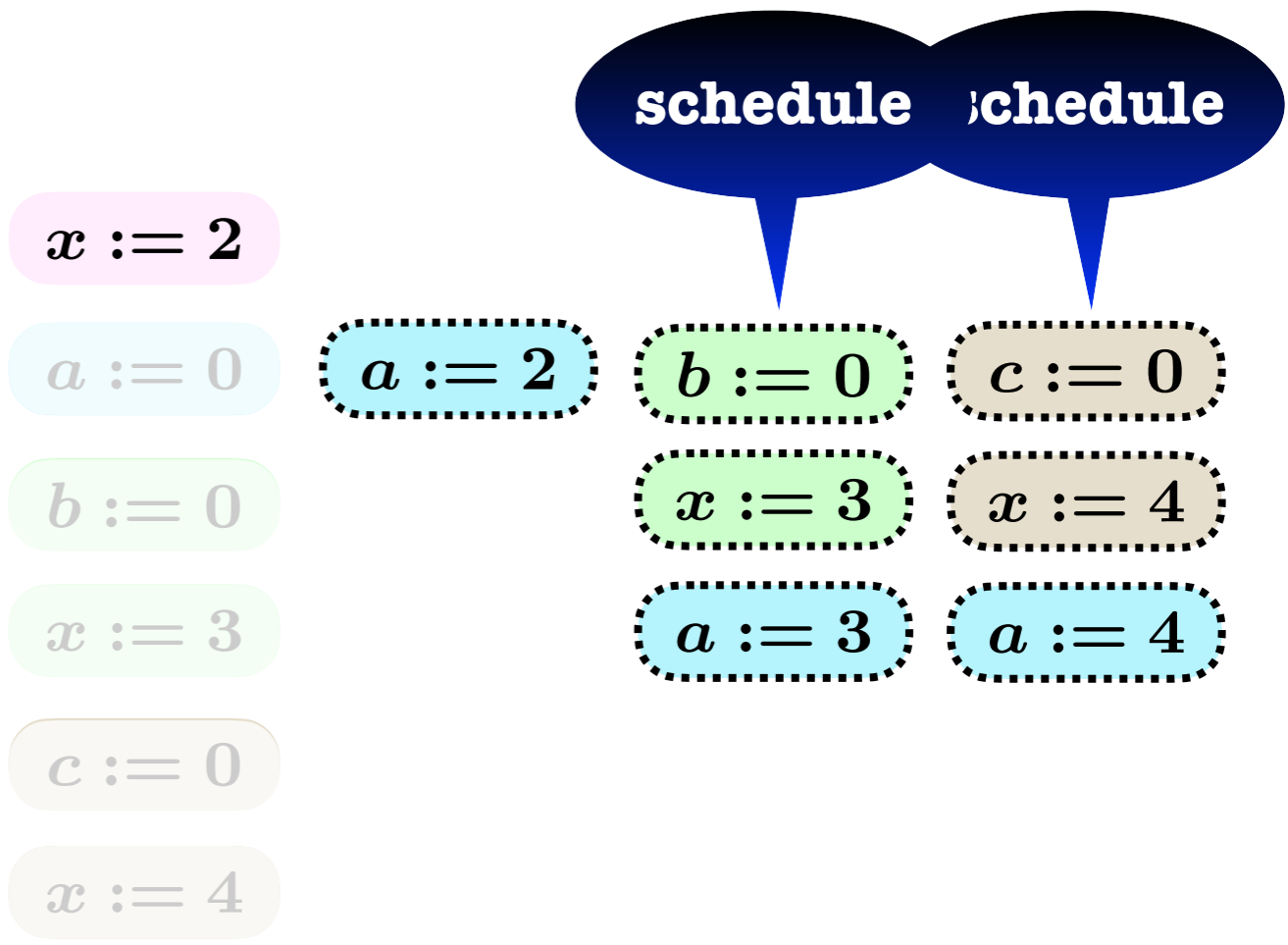
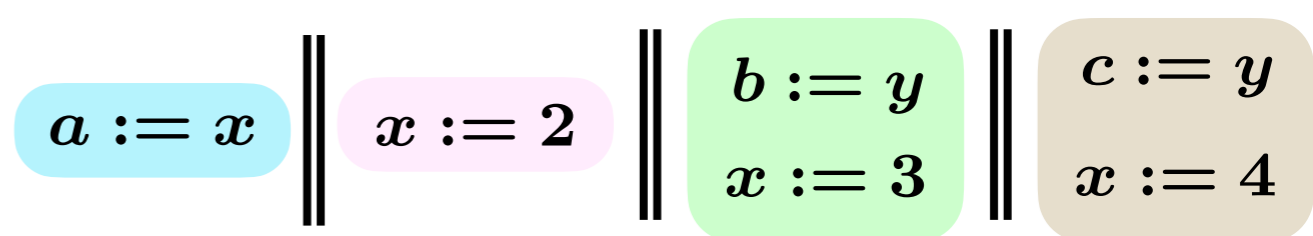
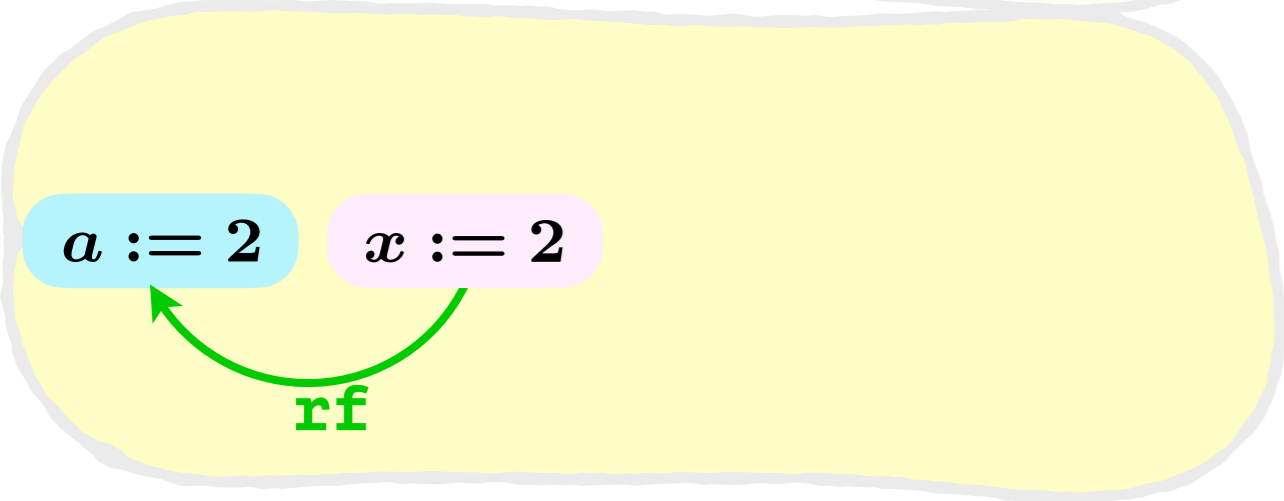
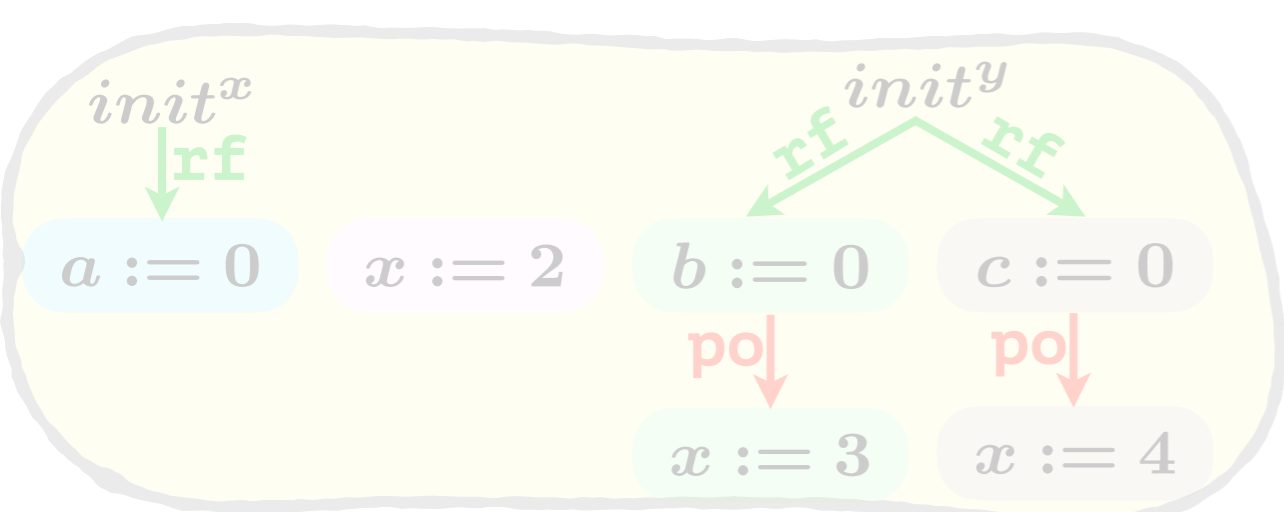
DPOR

Conclusions

- **(Optimal) DPOR**
- **Release-Acquire Semantics**
- **Experiments**

Future Work

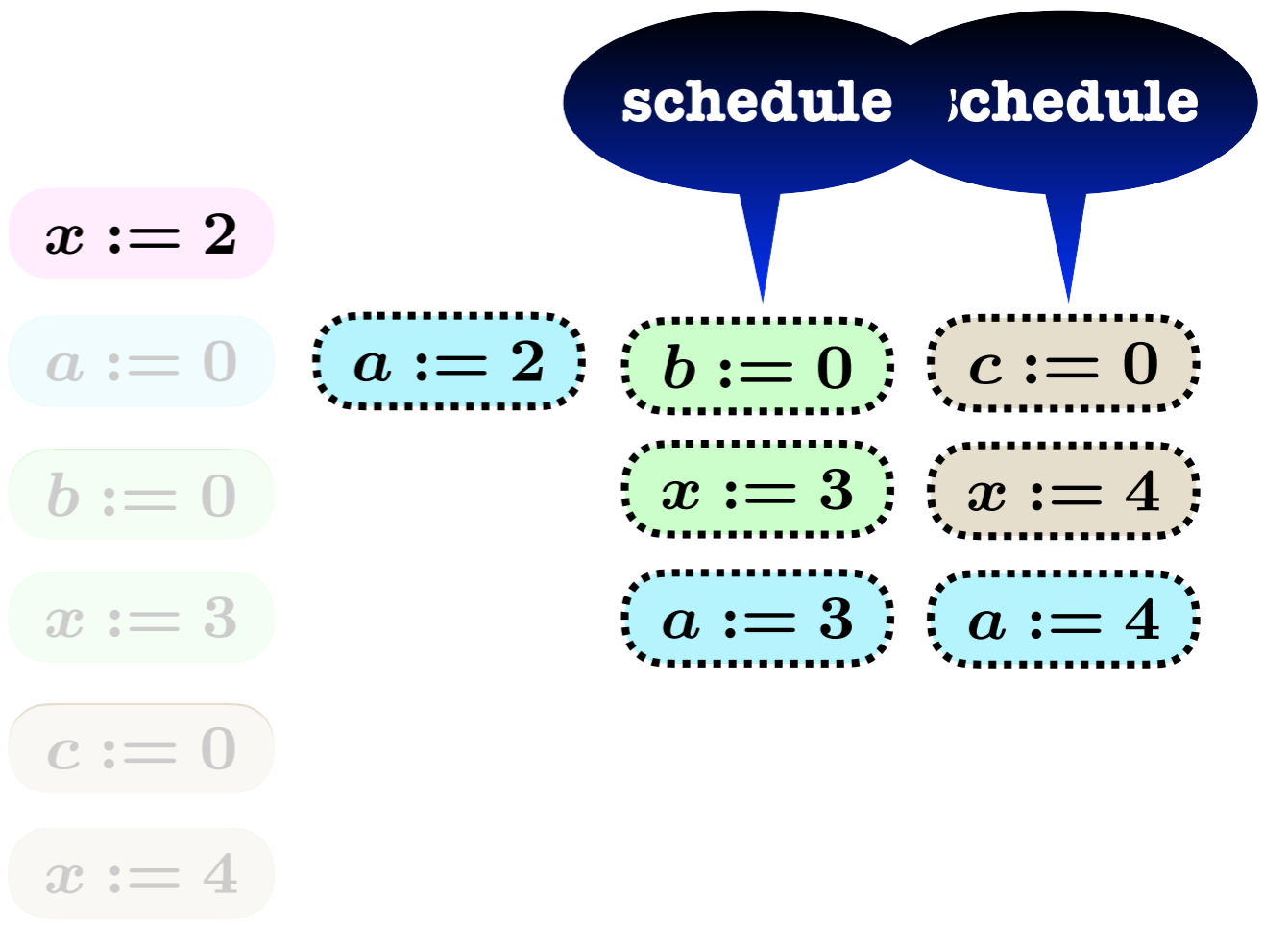
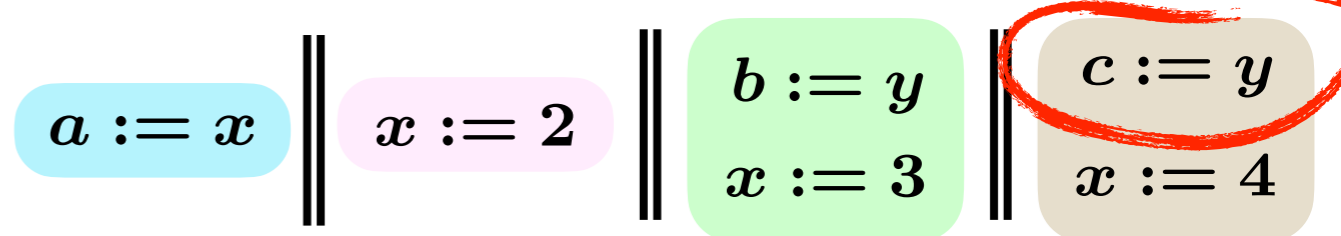
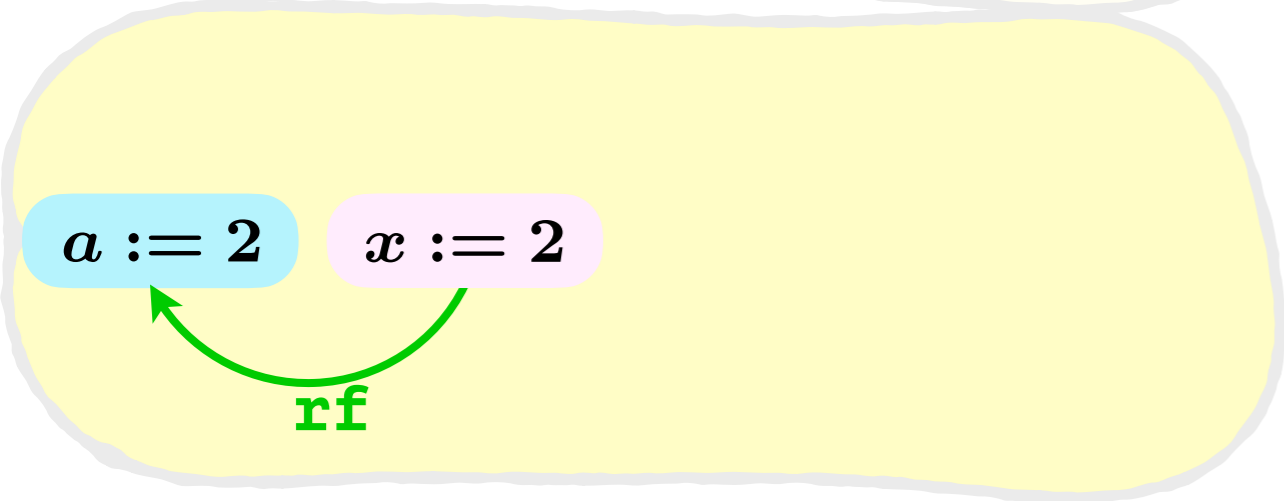
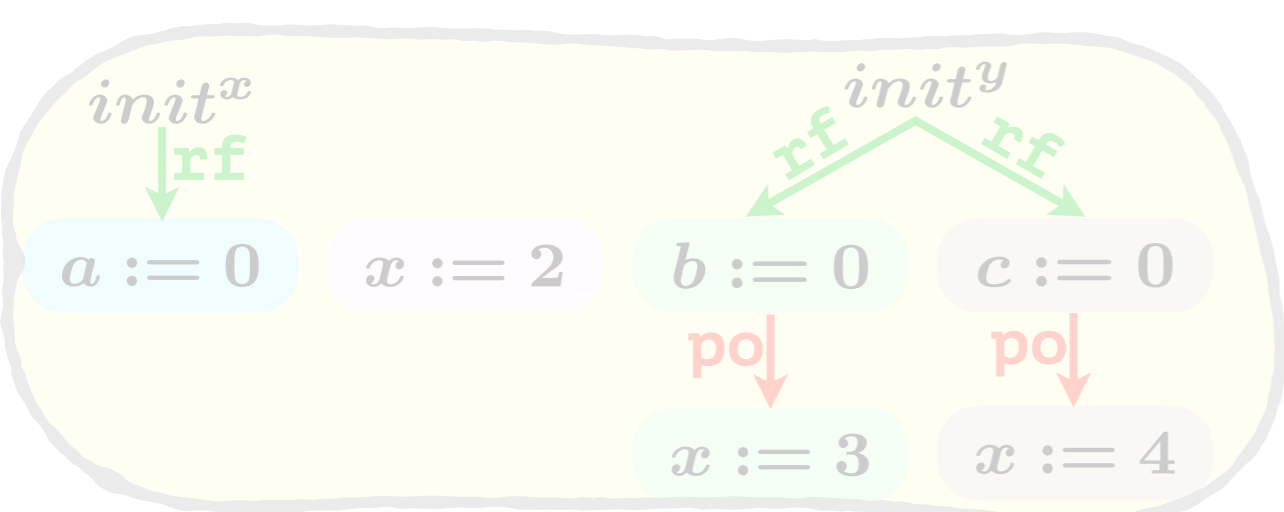
- **Other Memory Models**
 - **Full C11, SC, TSO, POWER, ARM, Causal Consistency, ...**
- **Weaker Notions of Traces**
- **Other Properties:**
 - **Race Detection**
 - **Liveness: Progress, Termination, ...**



Branch

&

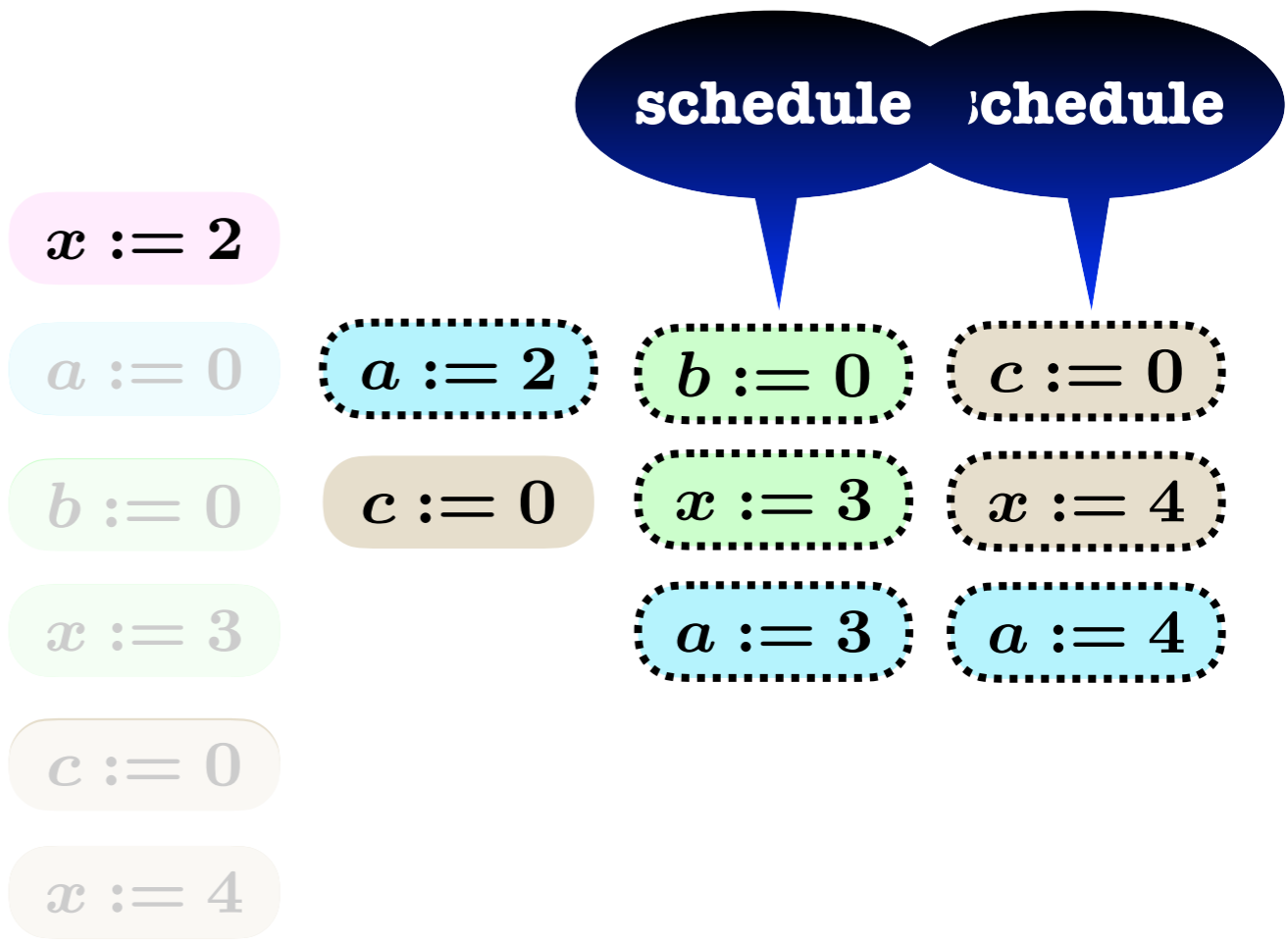
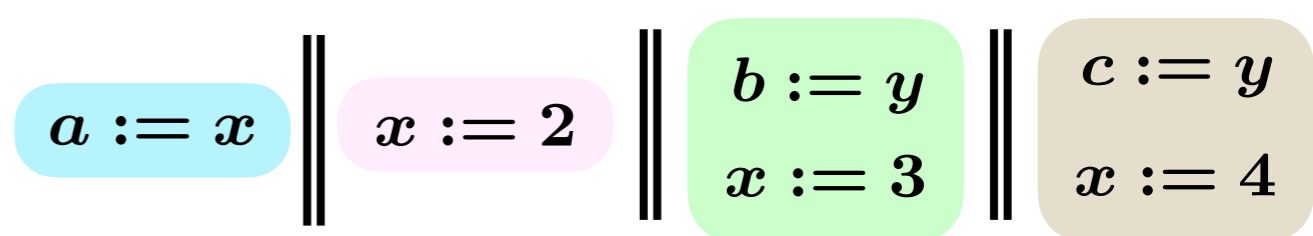
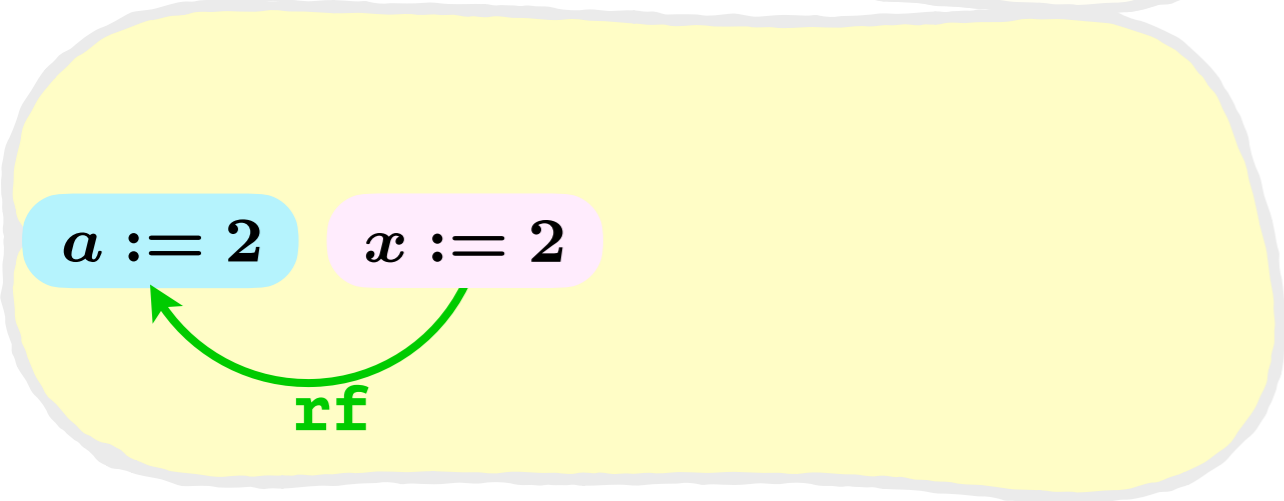
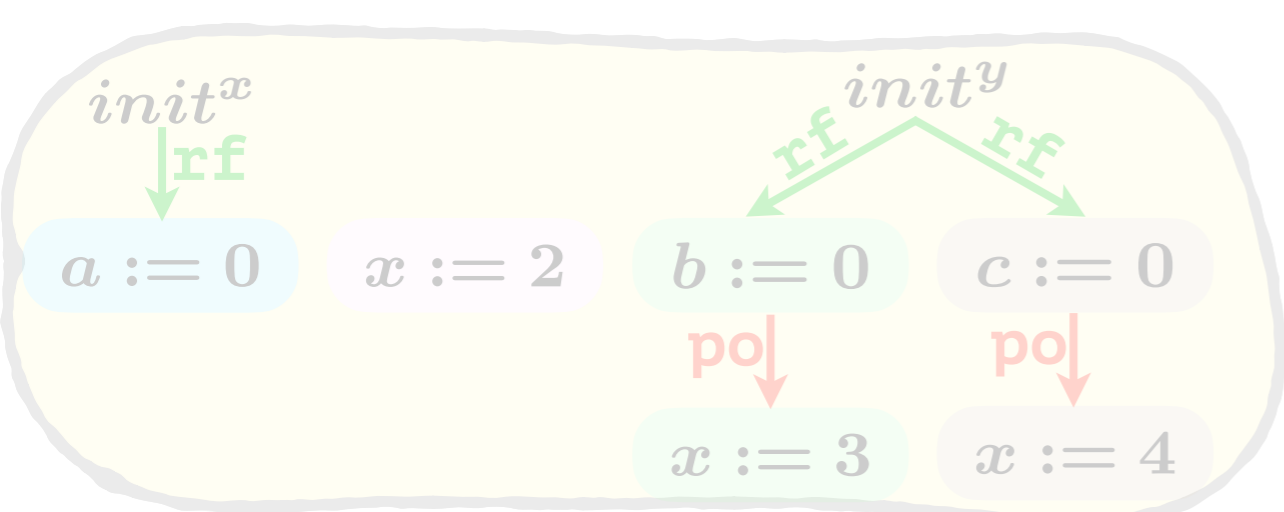
Postpone



Branch

&

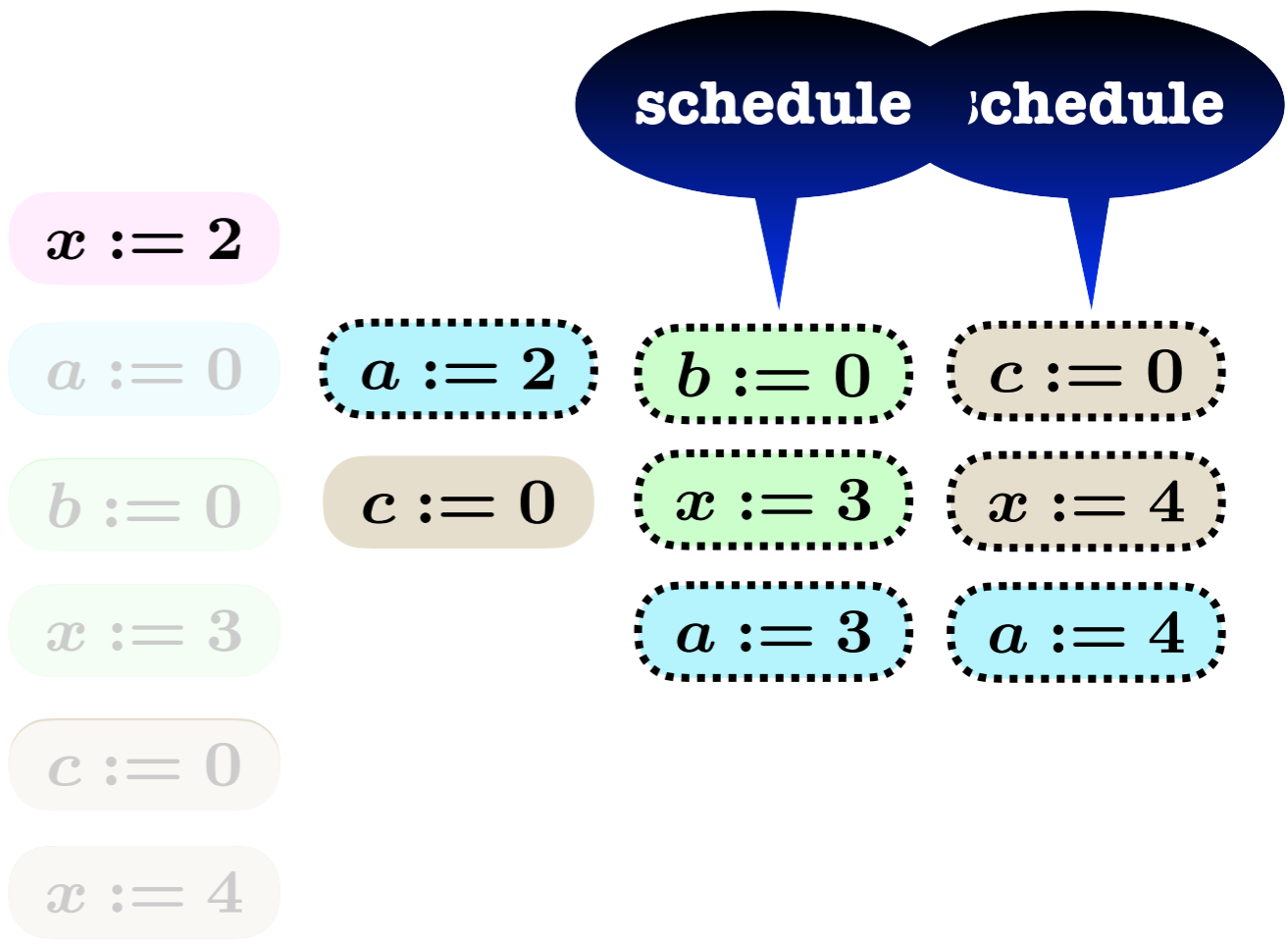
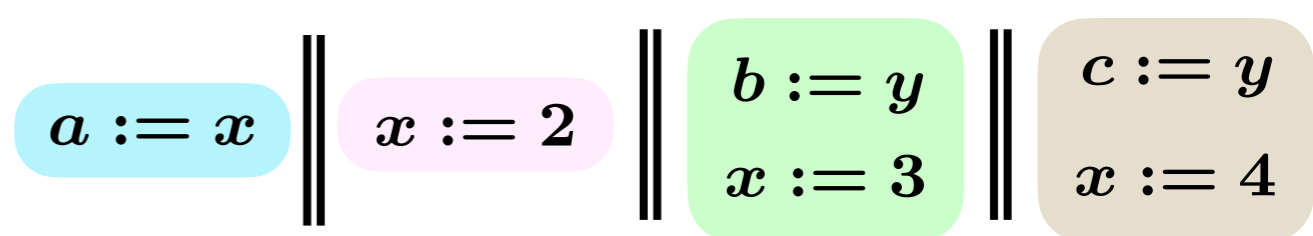
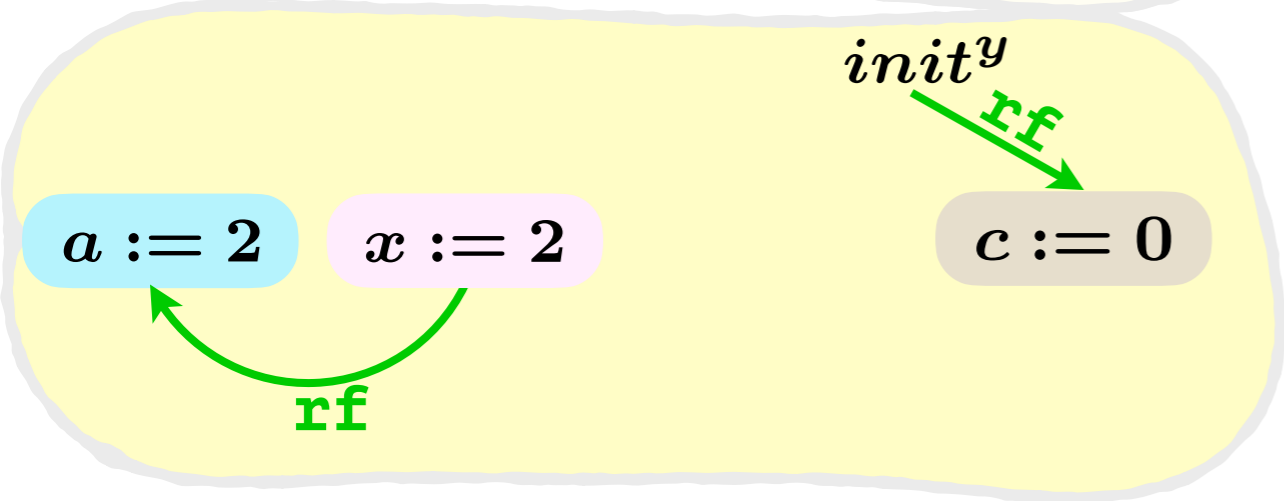
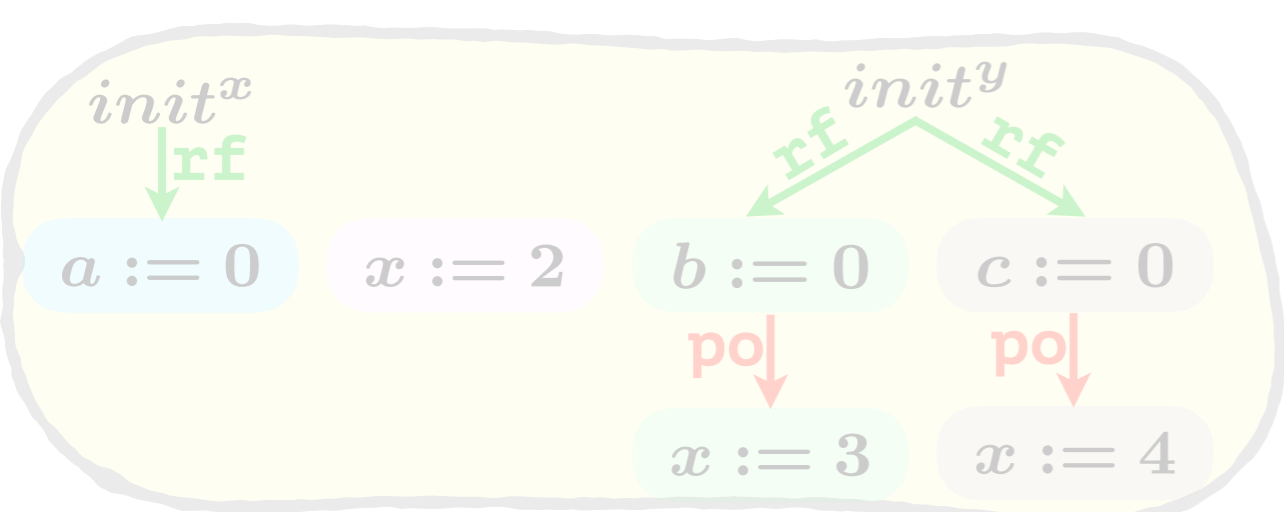
Postpone



Branch

&

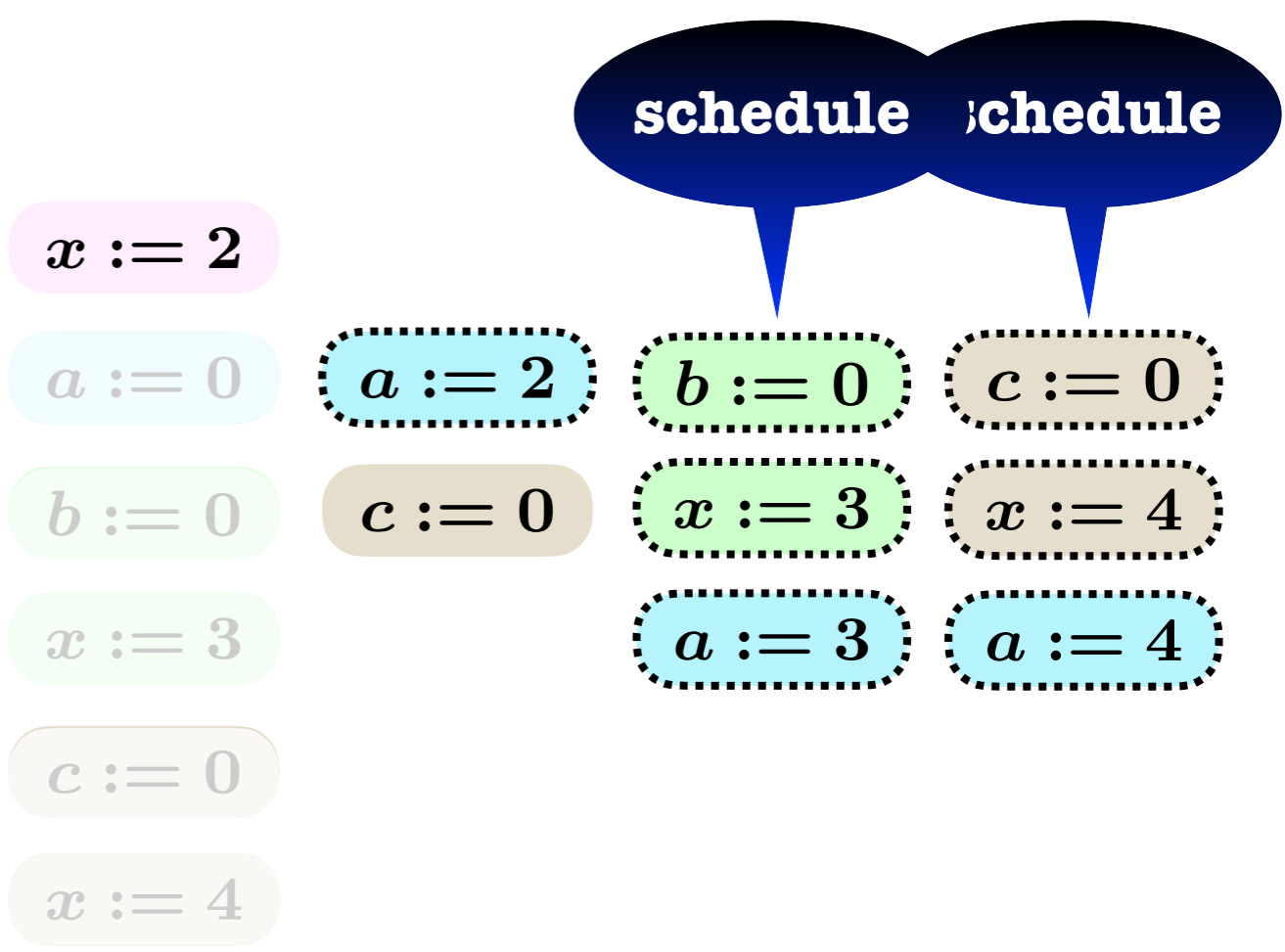
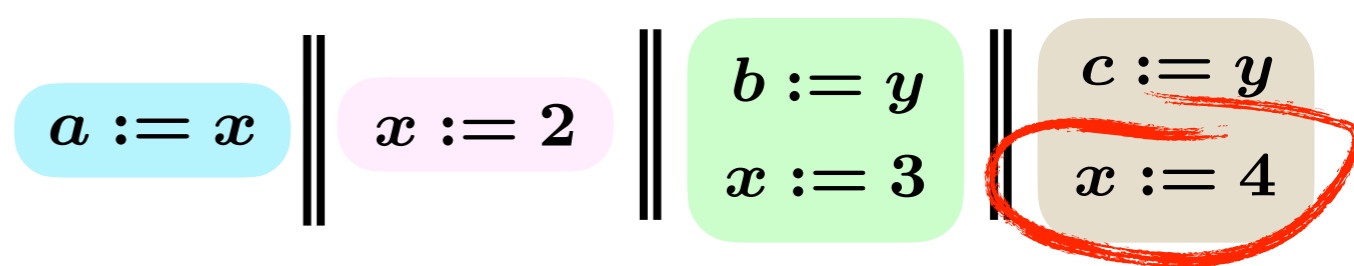
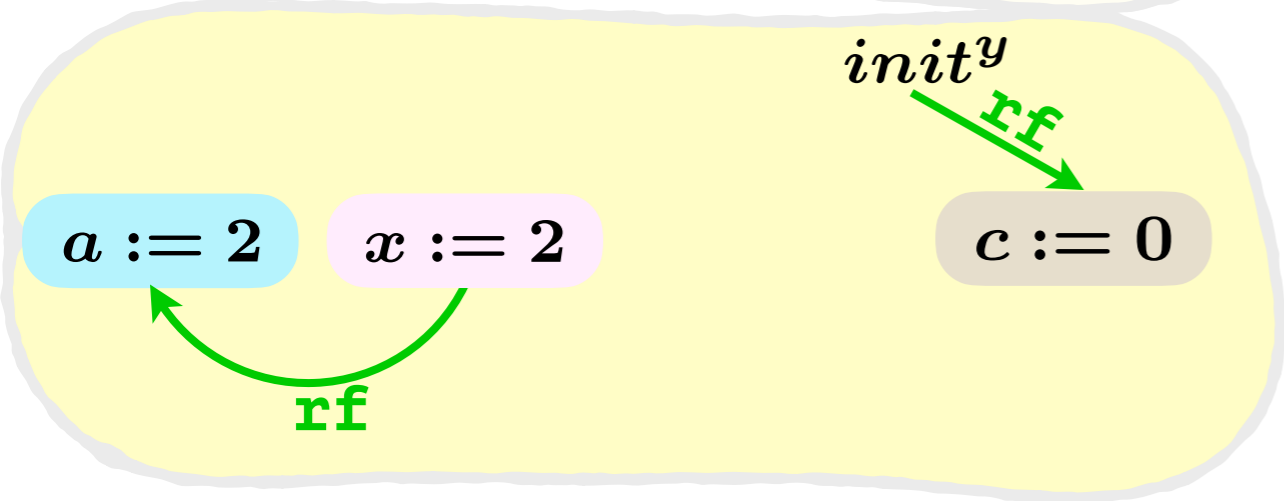
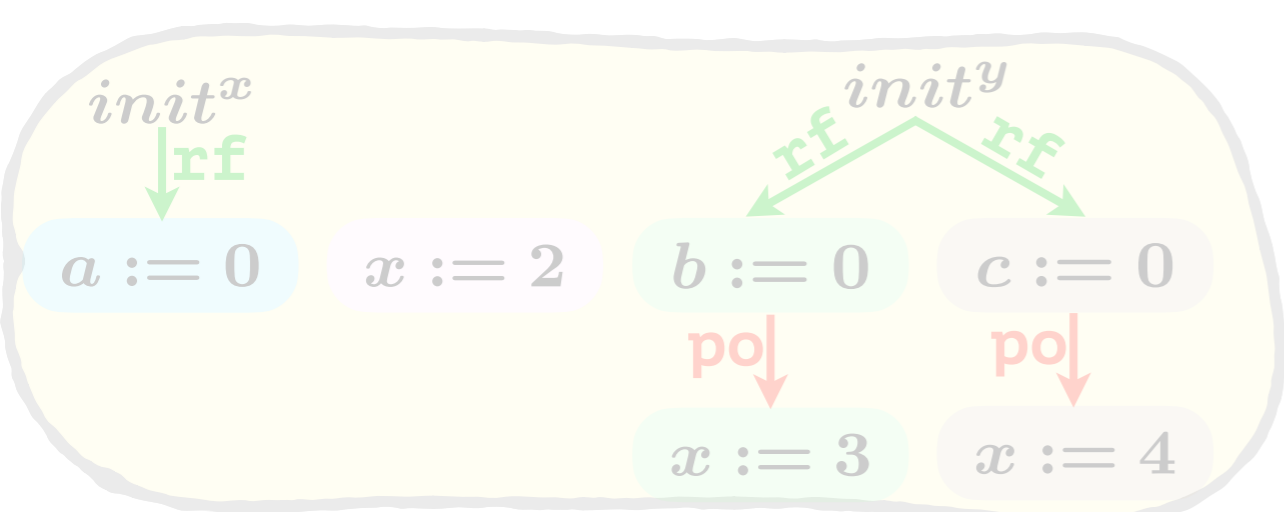
Postpone



Branch

&

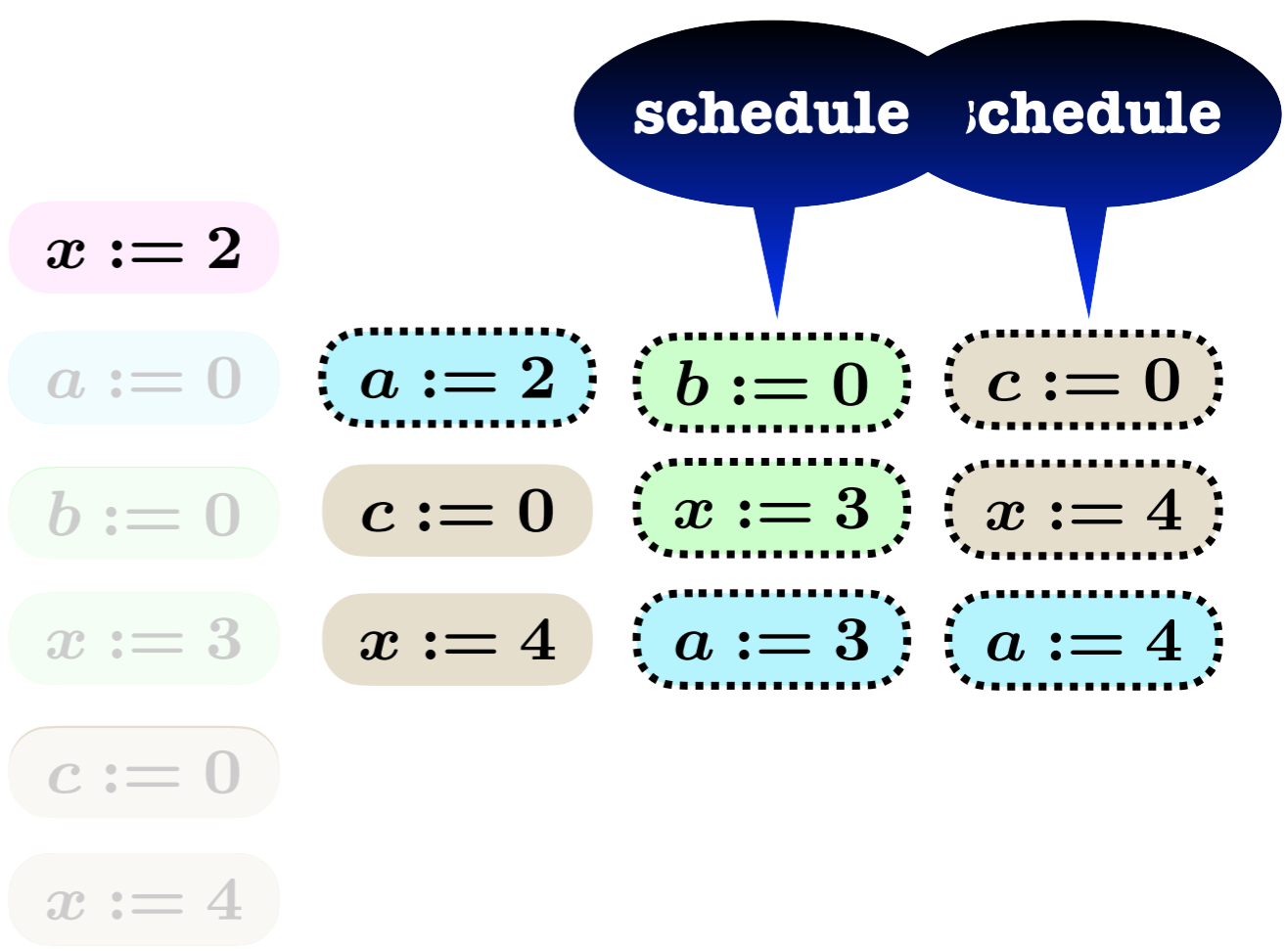
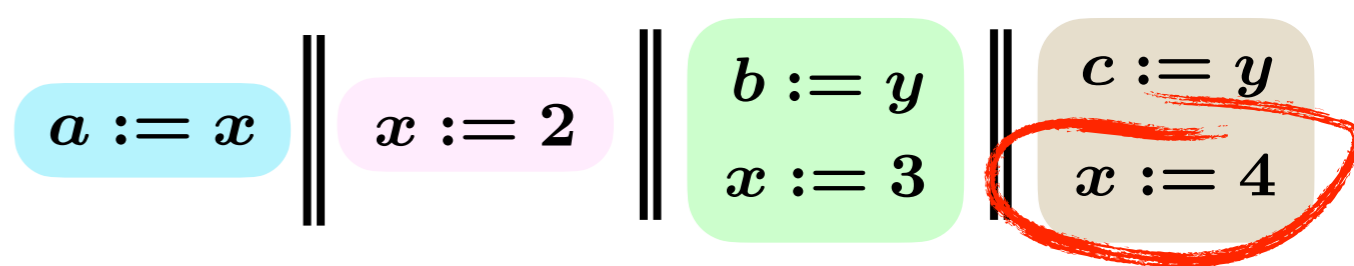
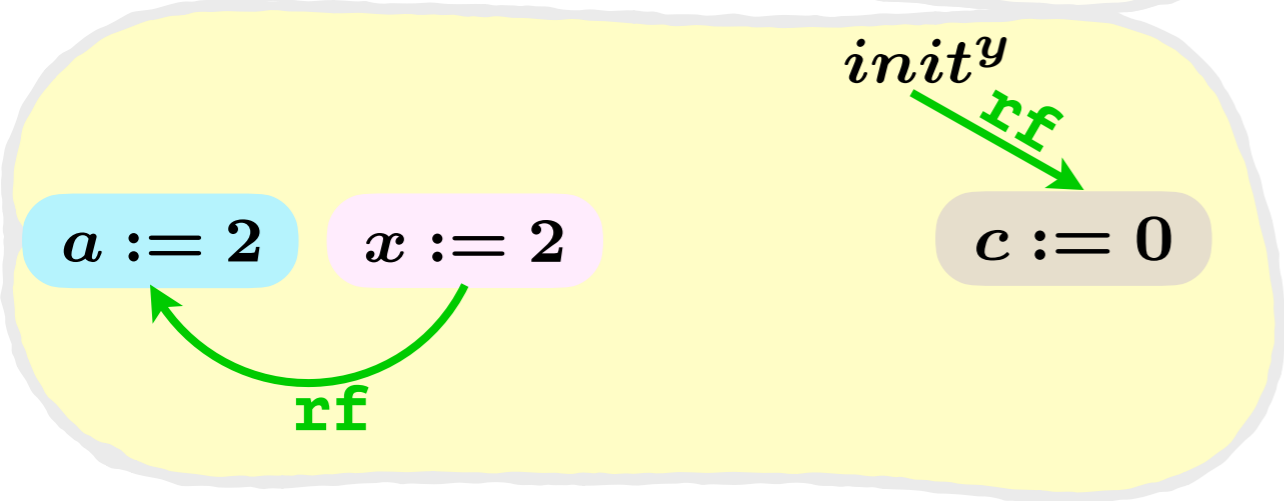
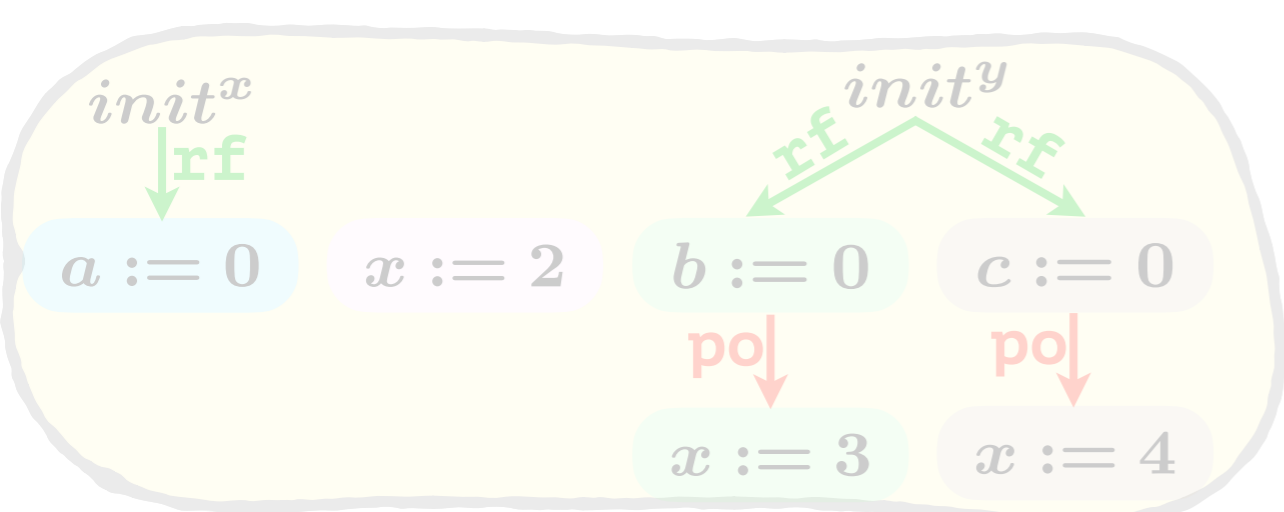
Postpone



Branch

&

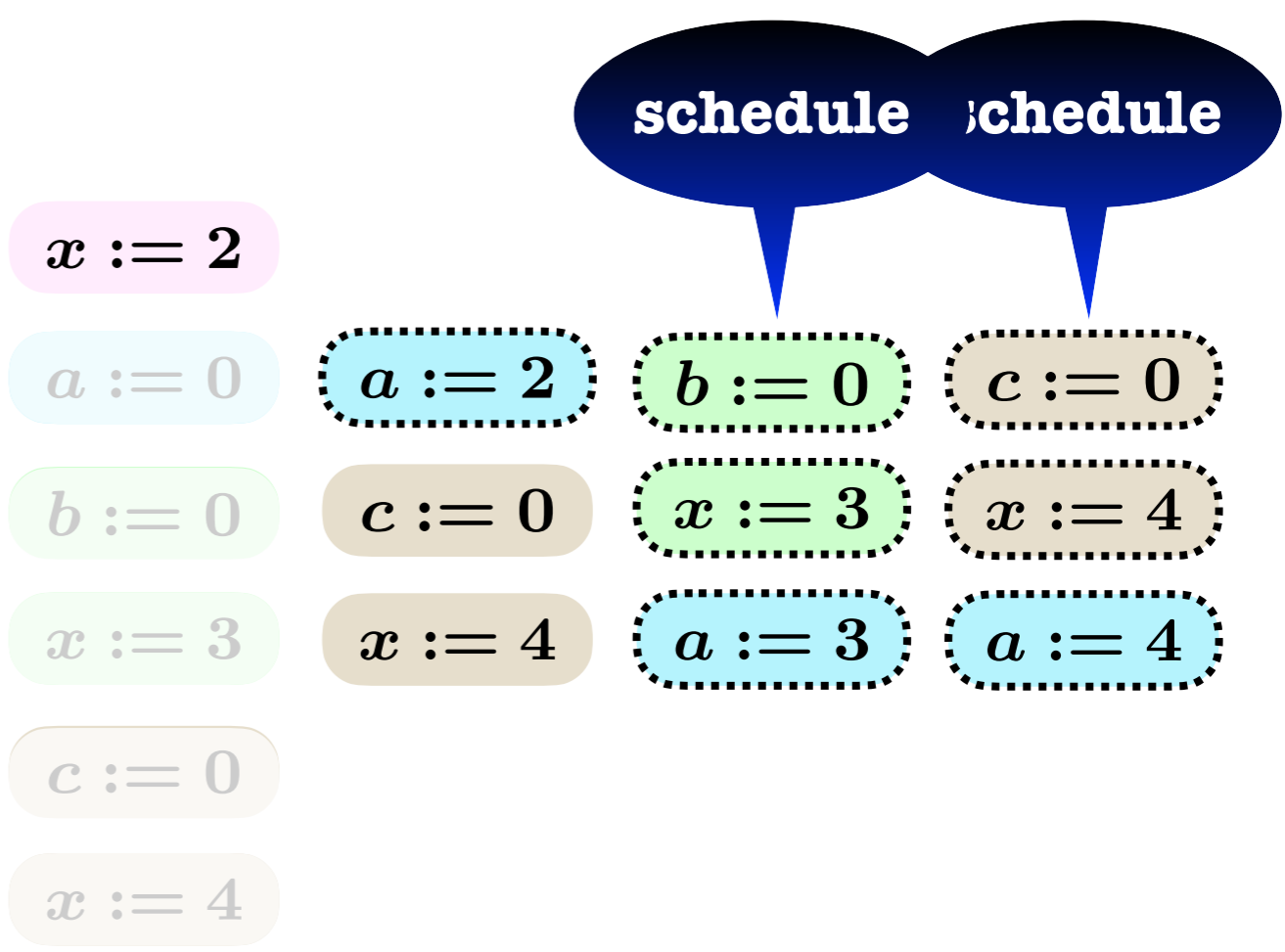
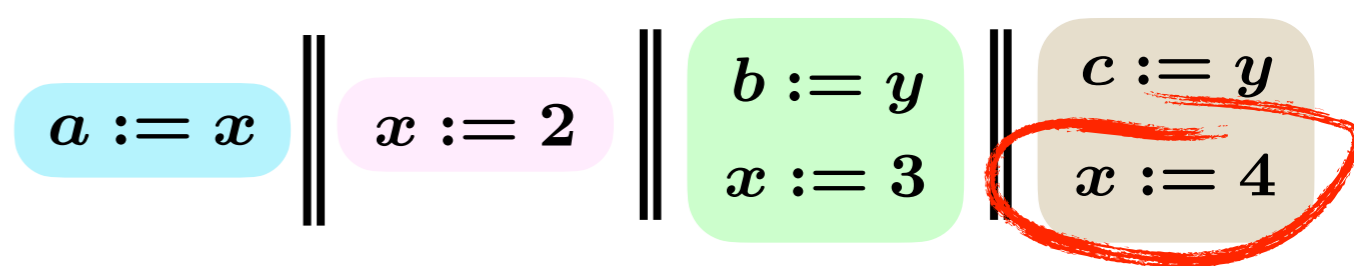
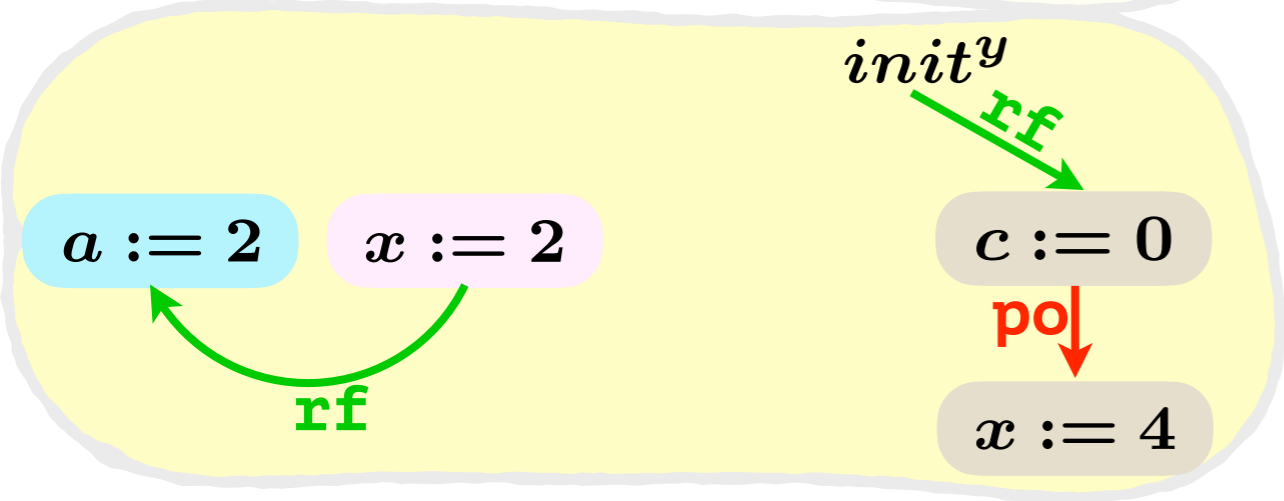
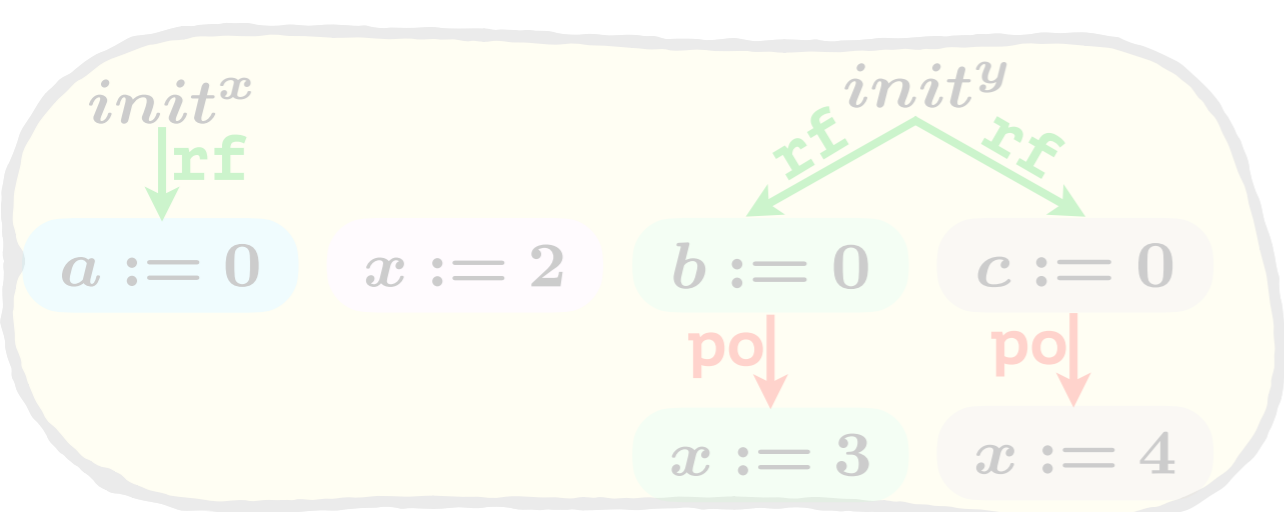
Postpone



Branch

&

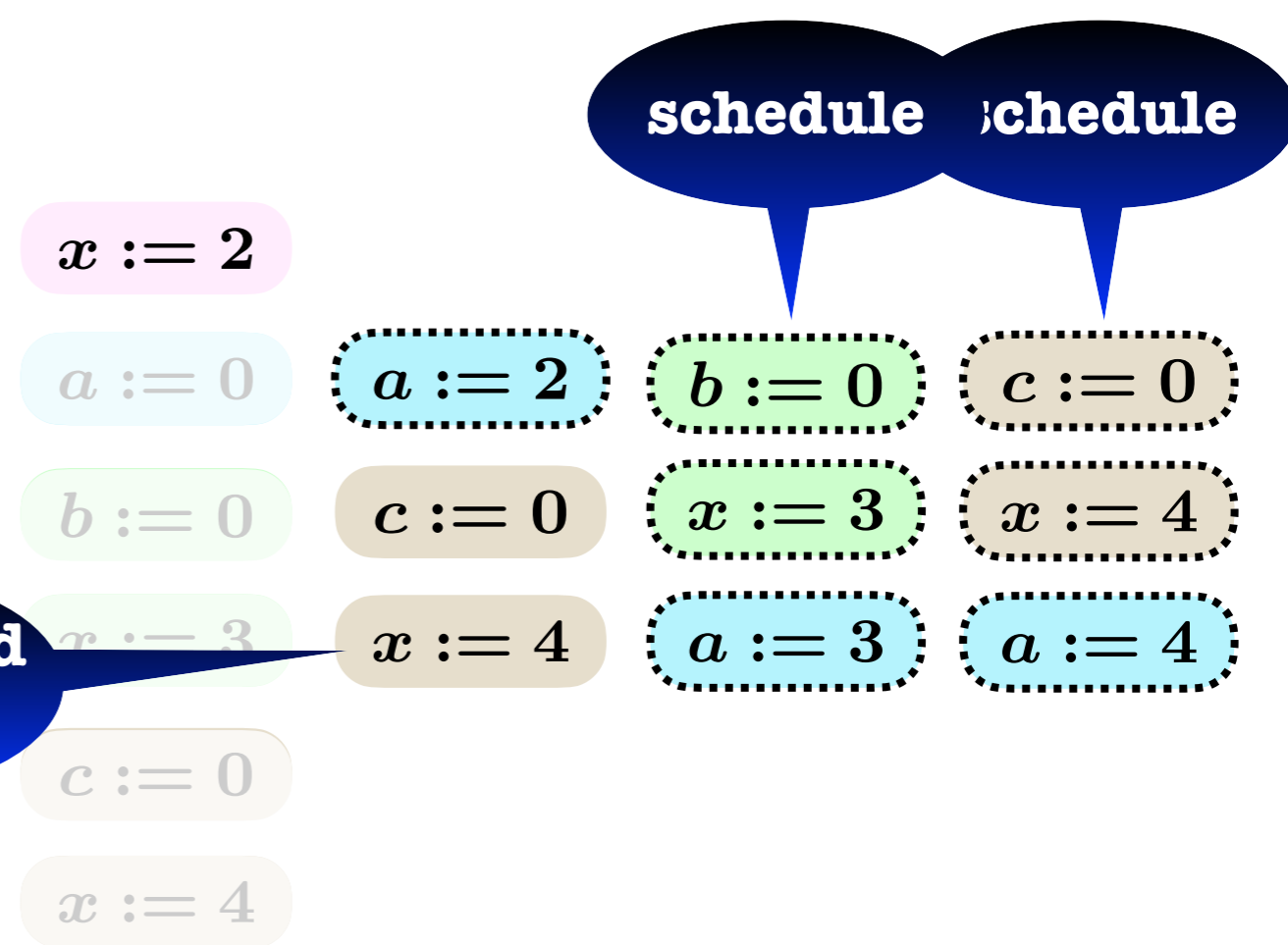
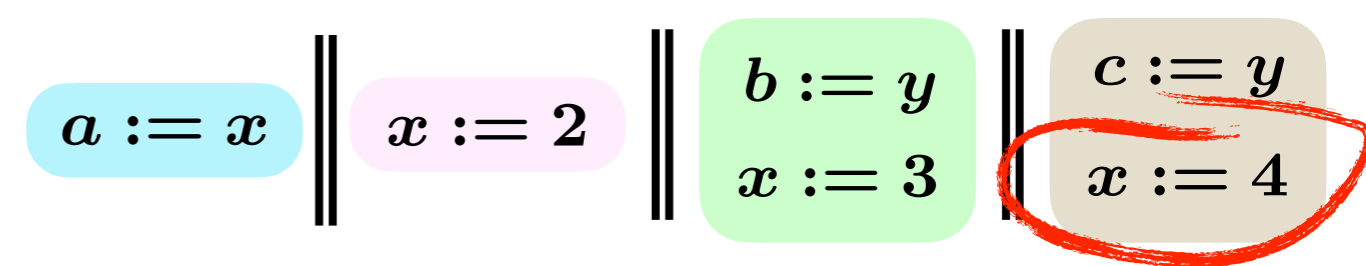
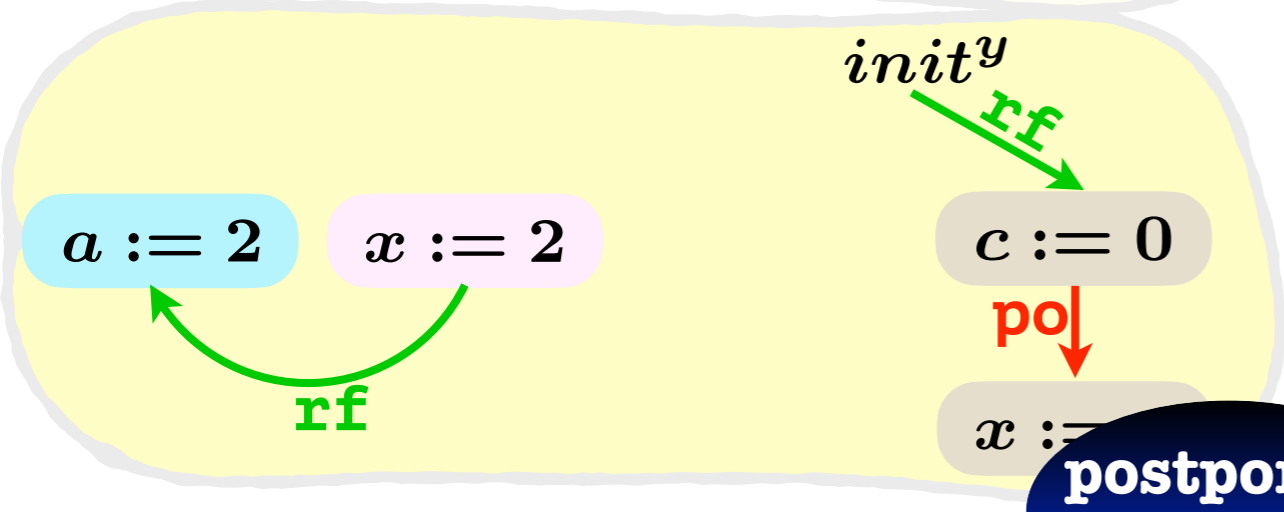
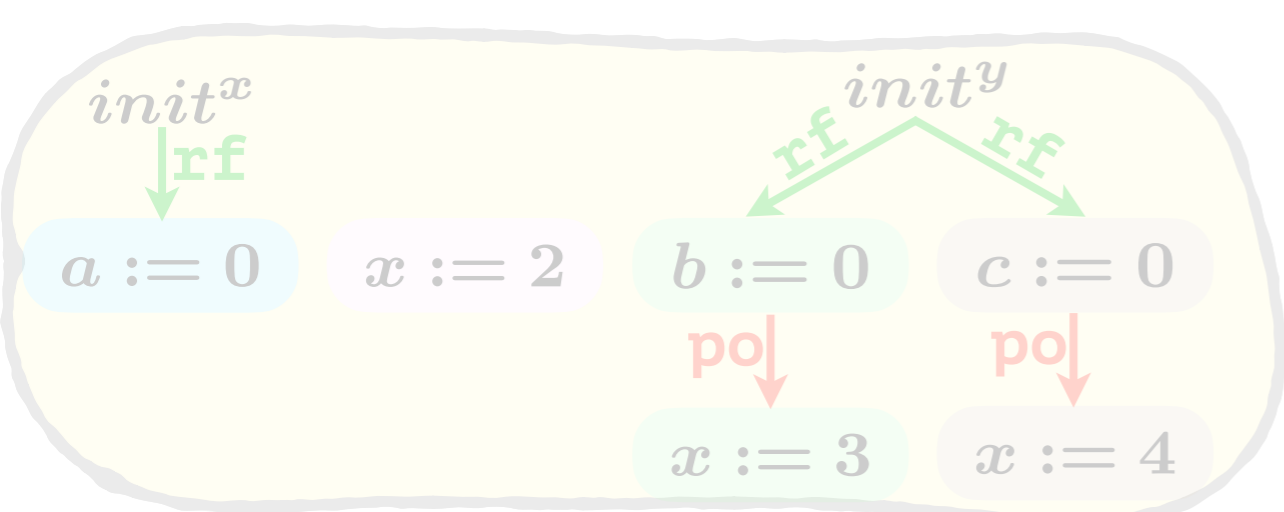
Postpone



Branch

&

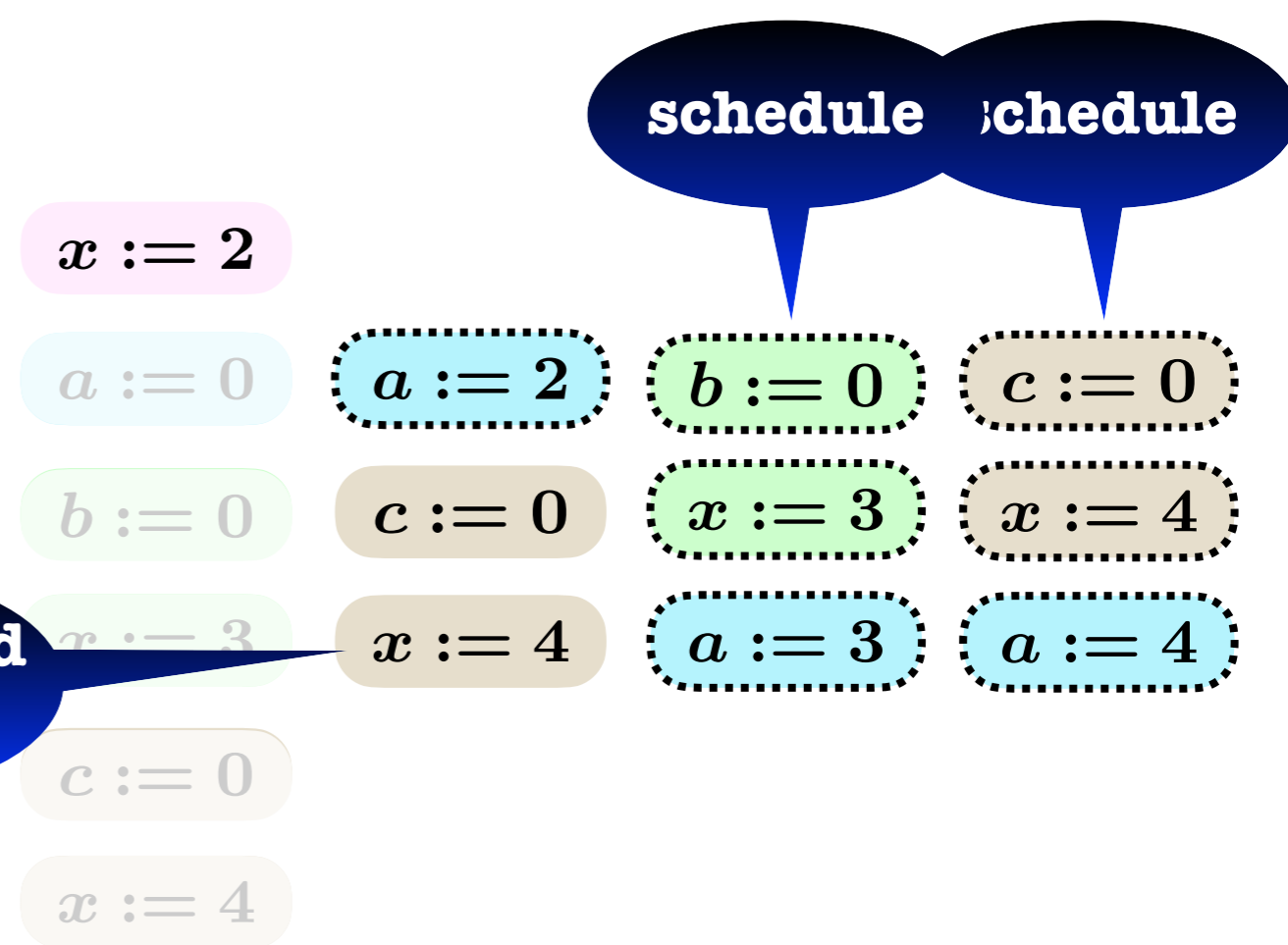
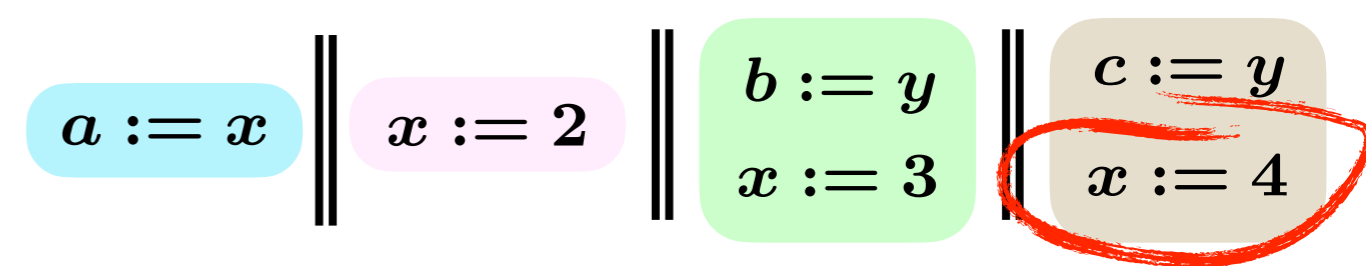
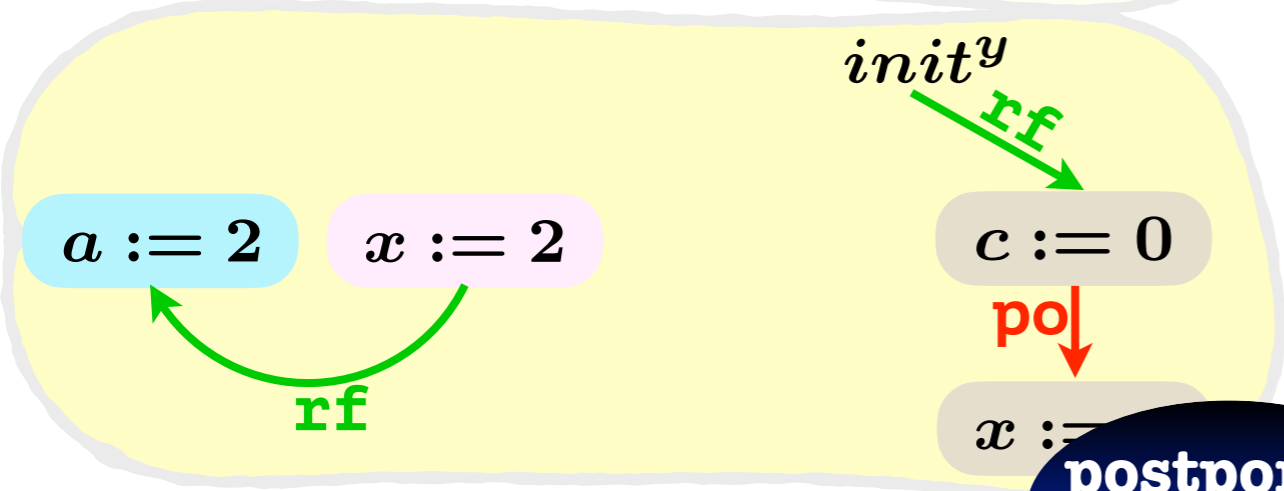
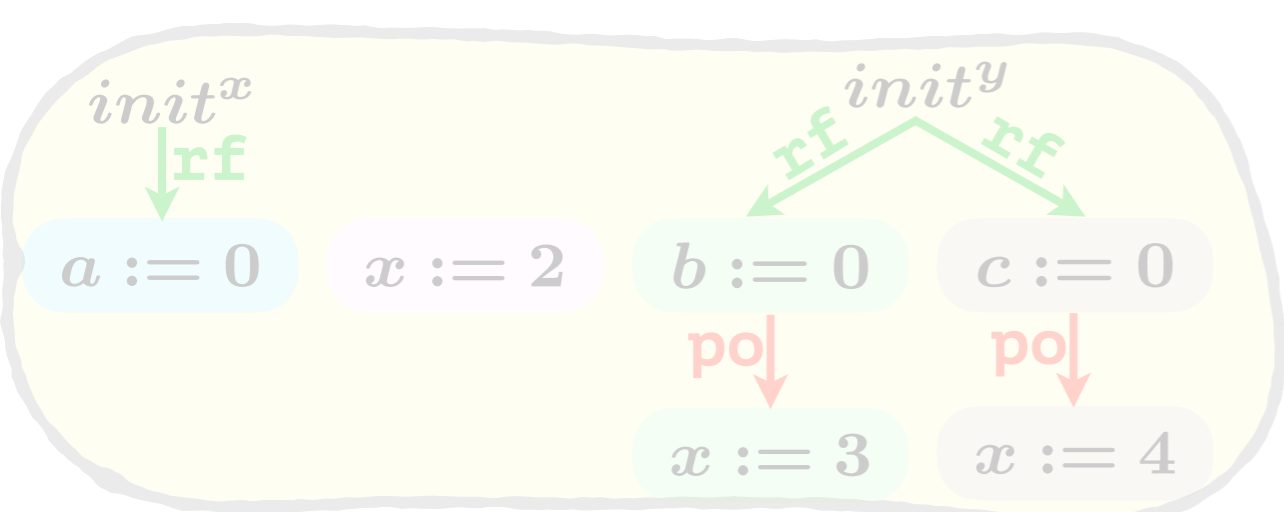
Postpone



Branch

&

Postpone



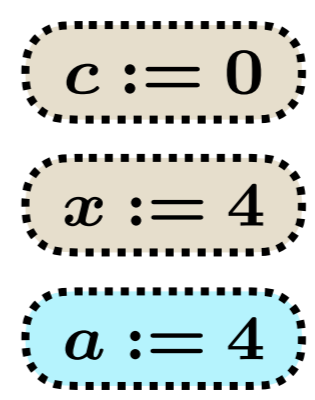
postponed write

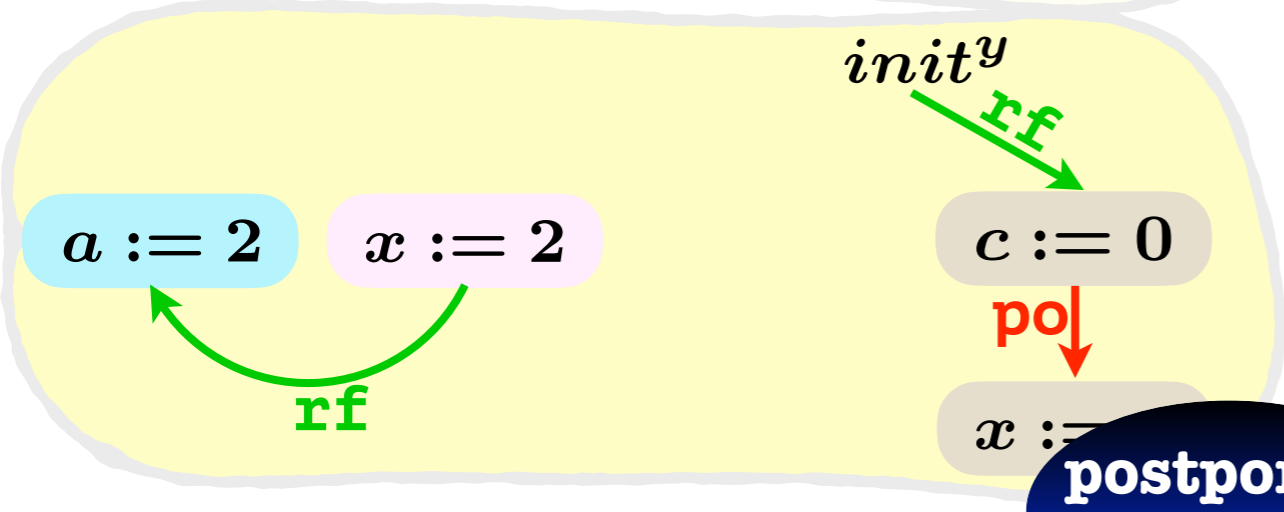
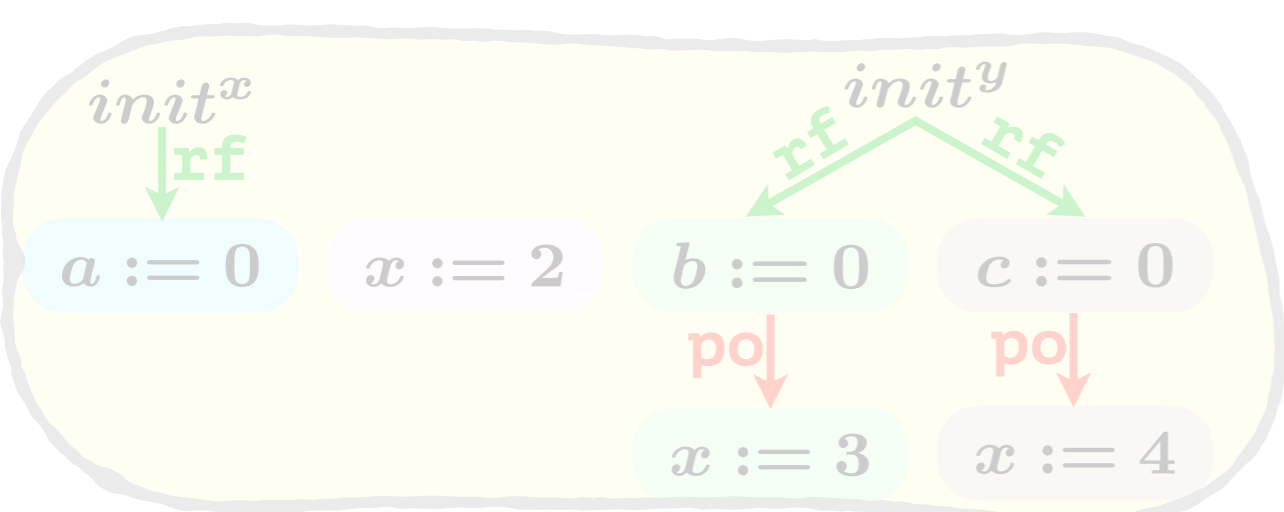
schedule schedule

Branch

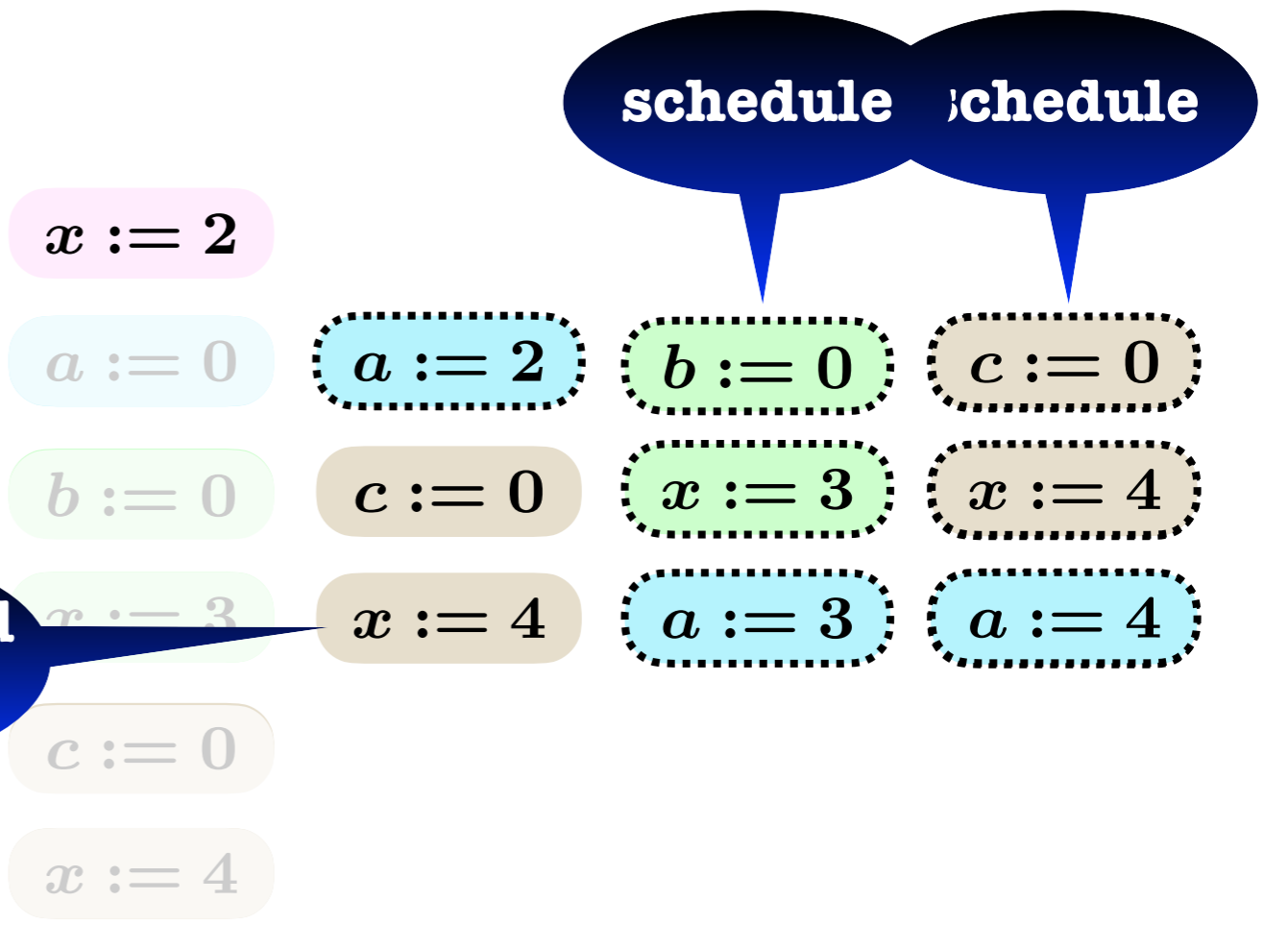
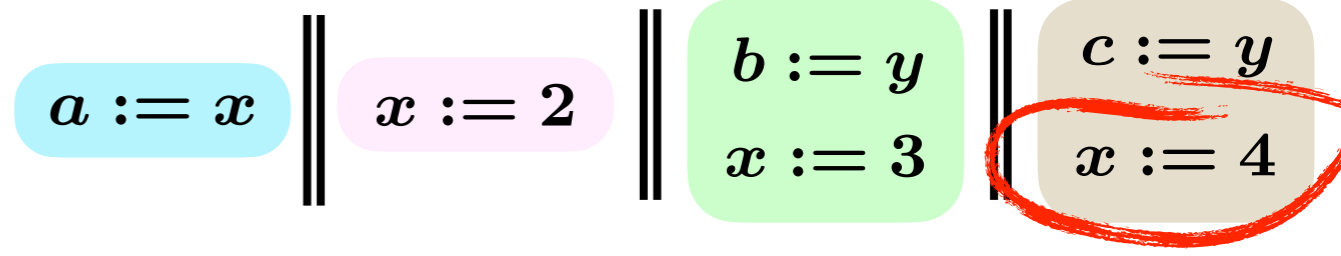
&

Postpone





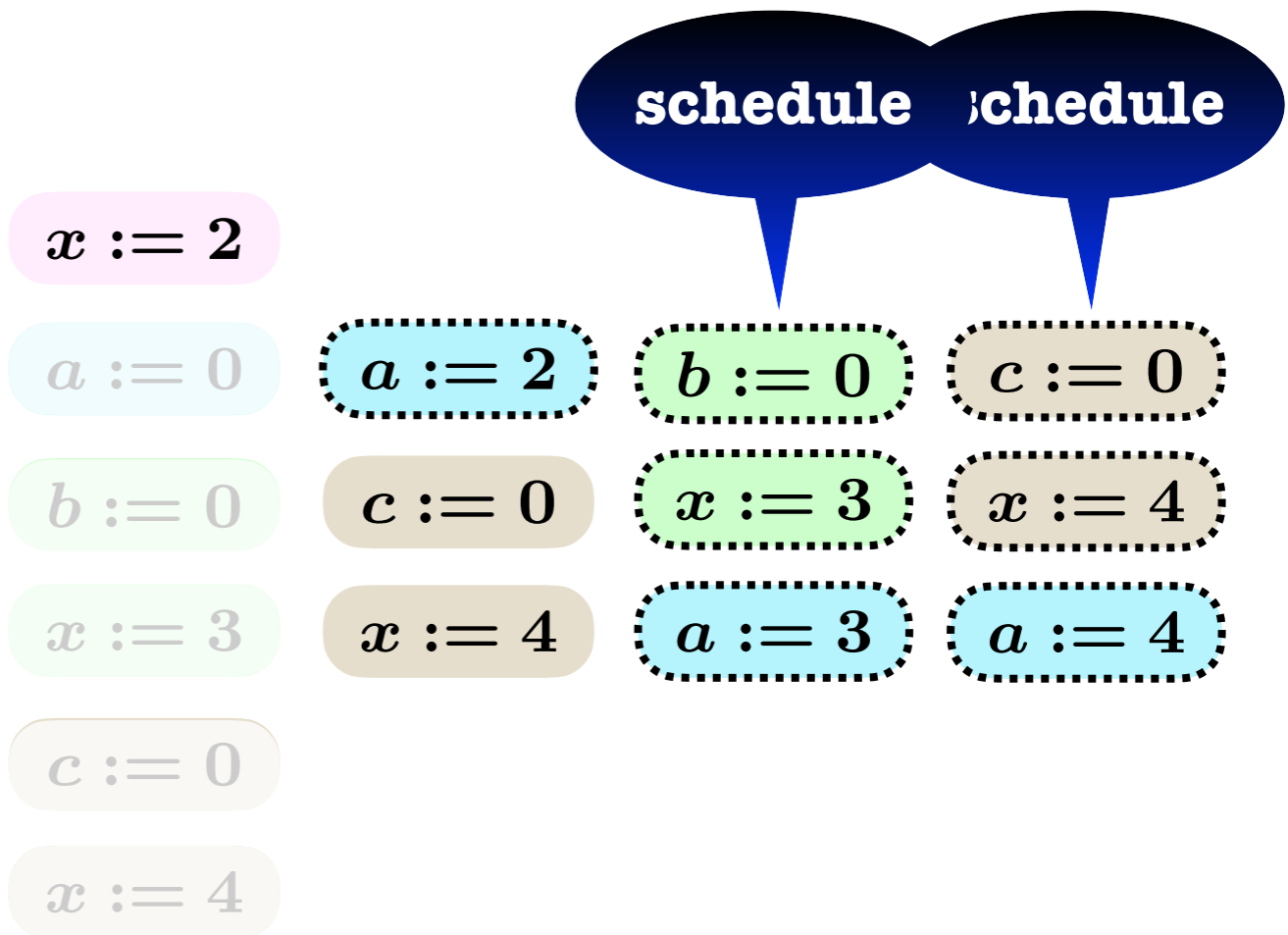
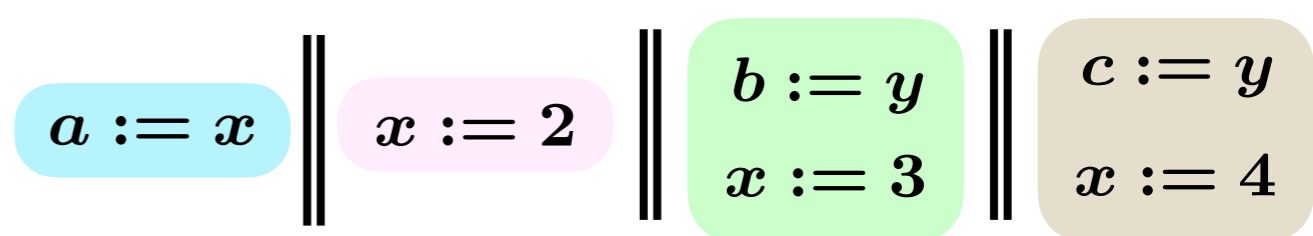
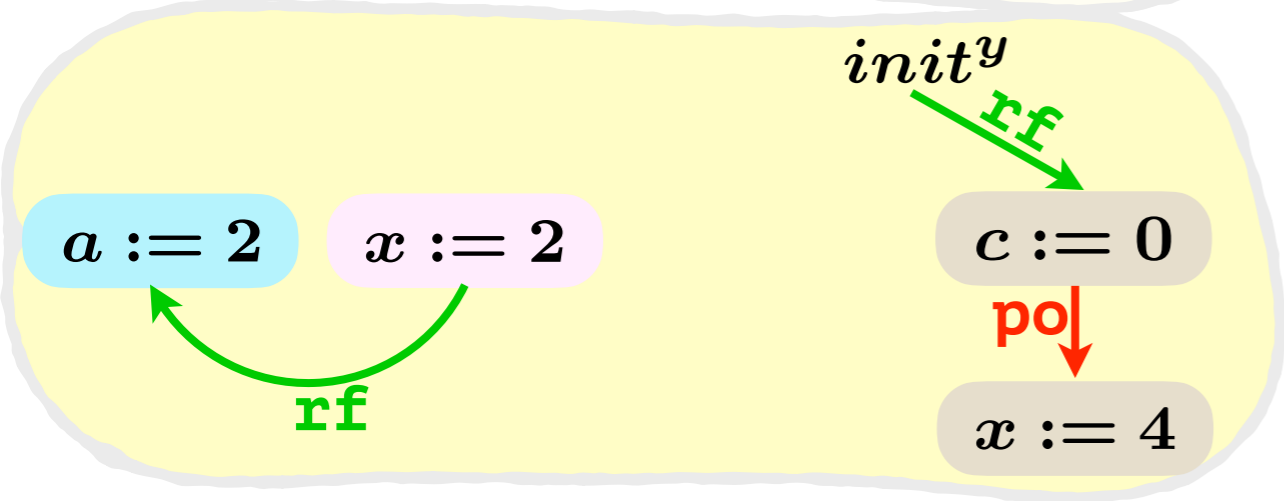
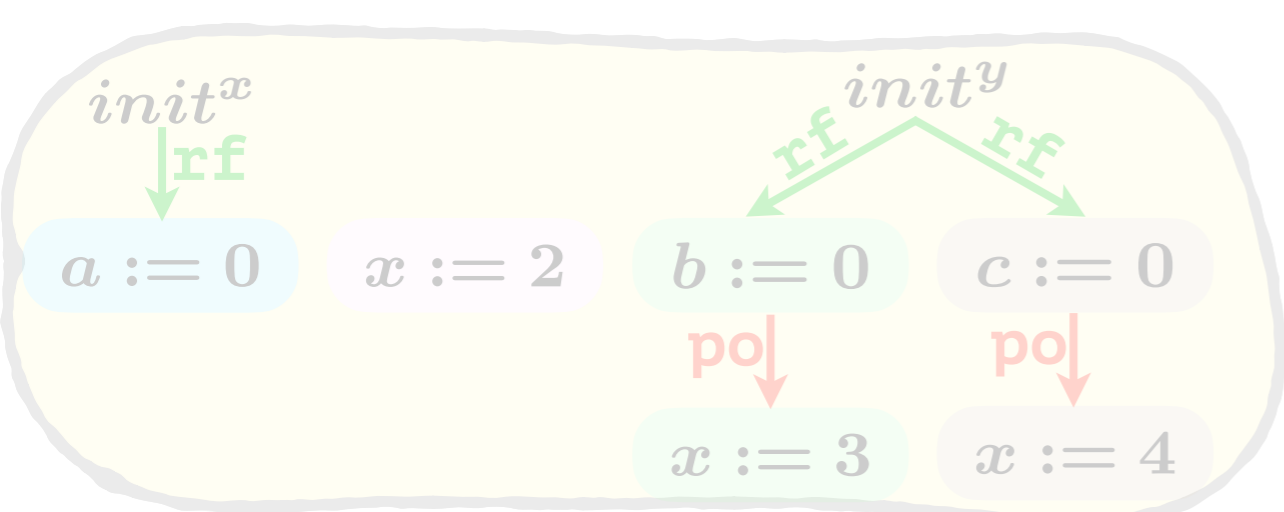
postponed write



Branch

&

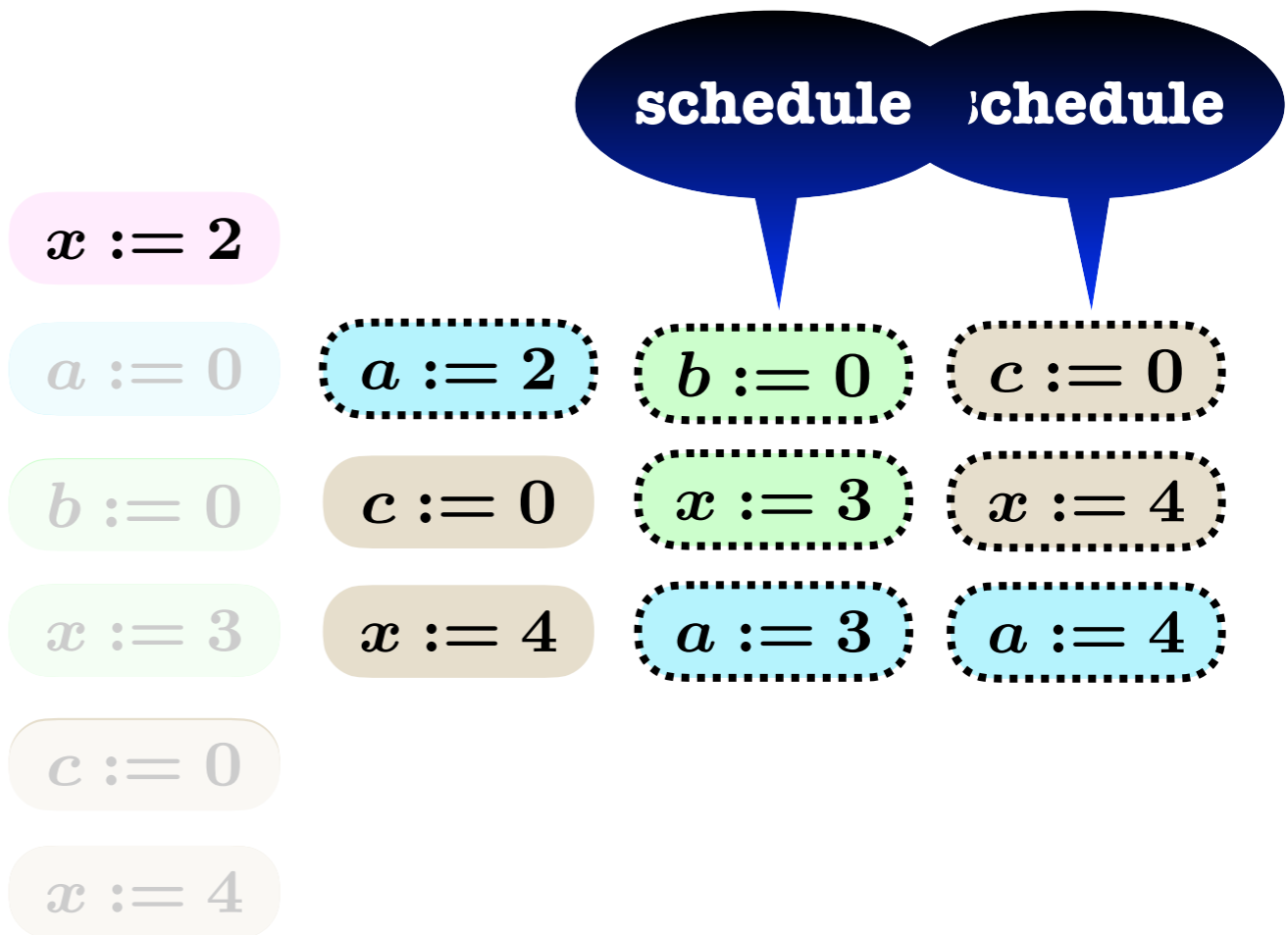
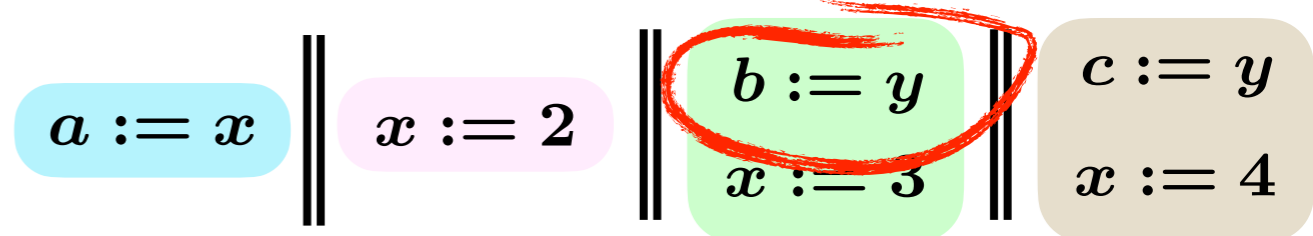
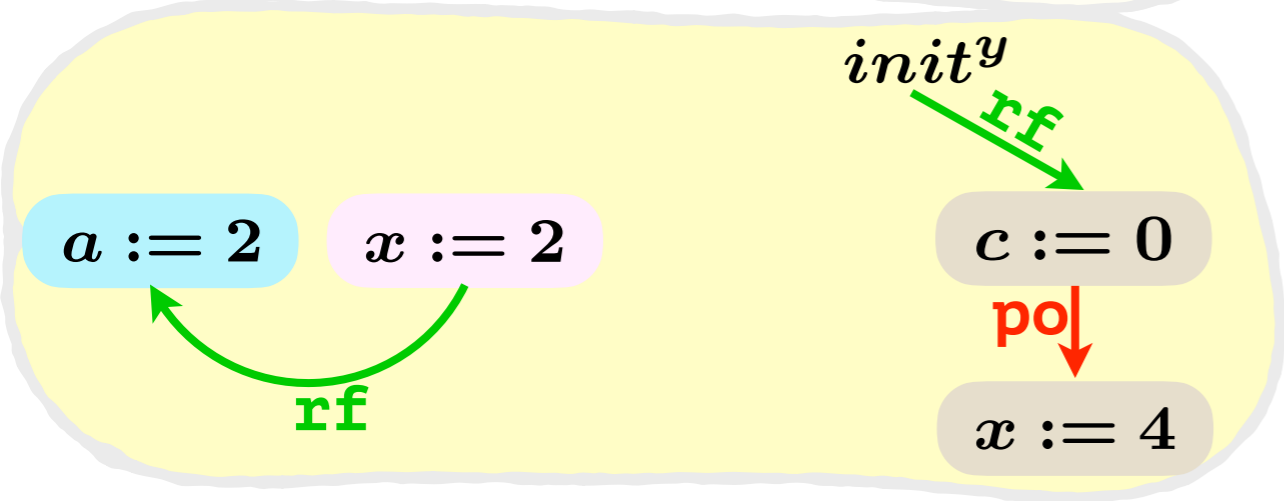
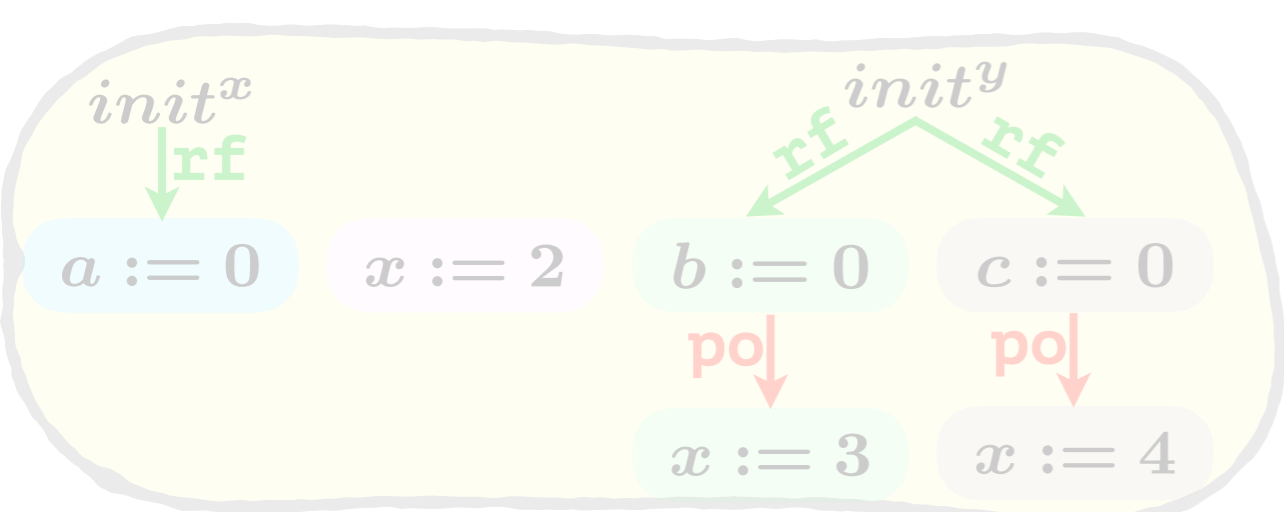
Postpone



Branch

&

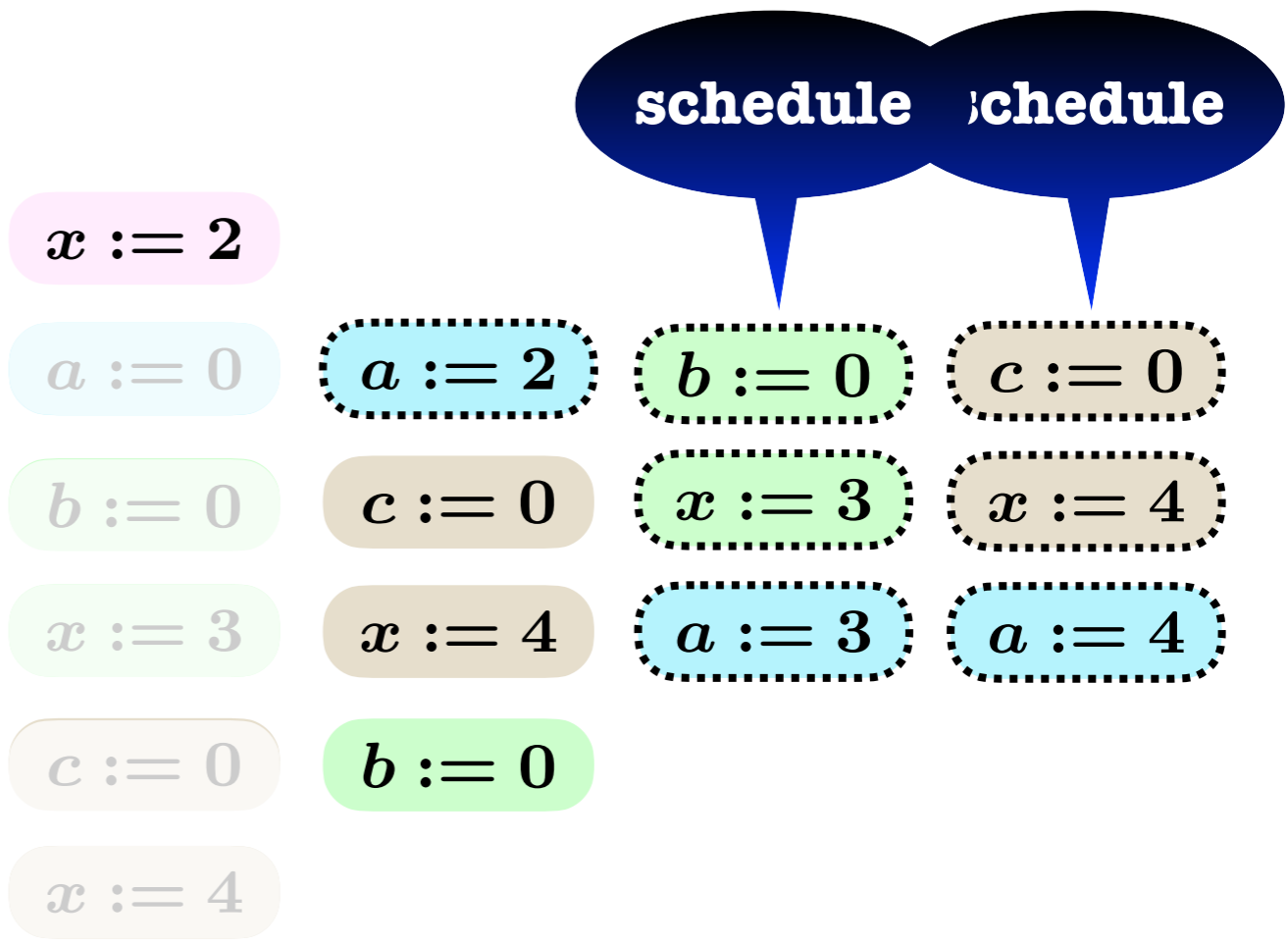
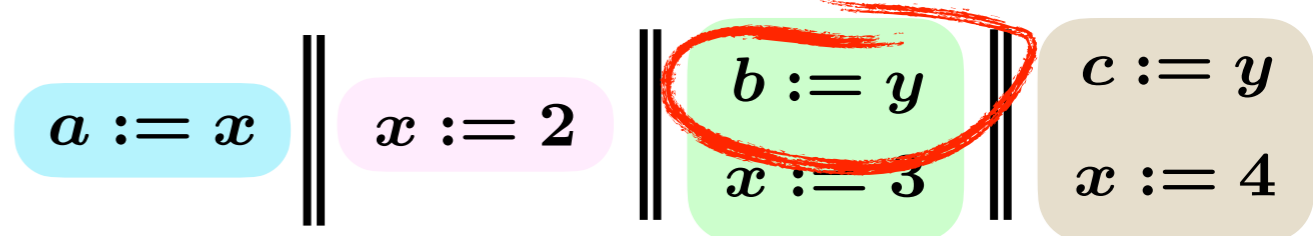
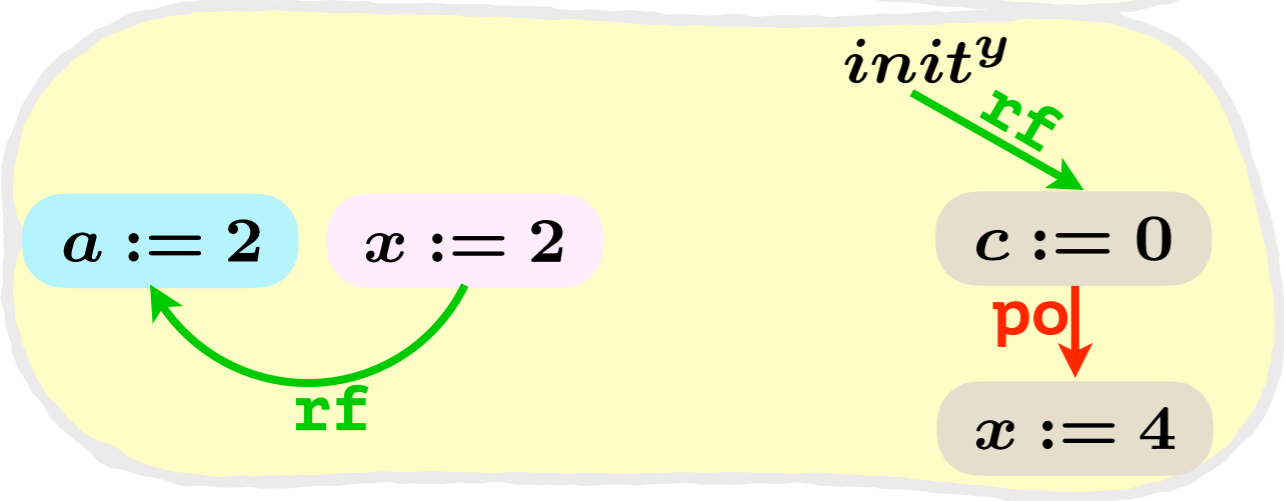
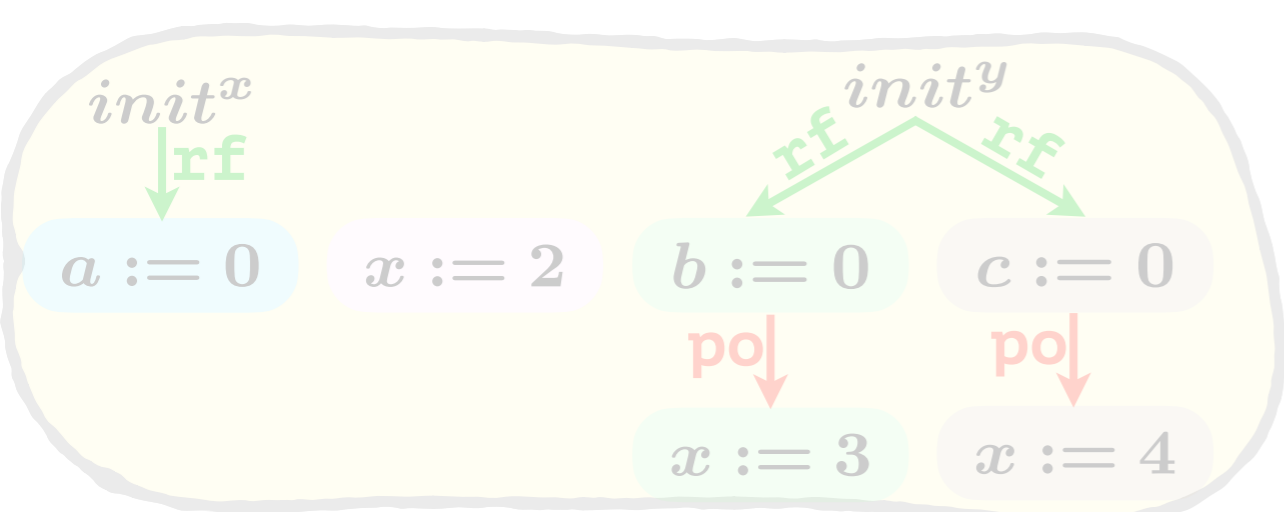
Postpone



Branch

&

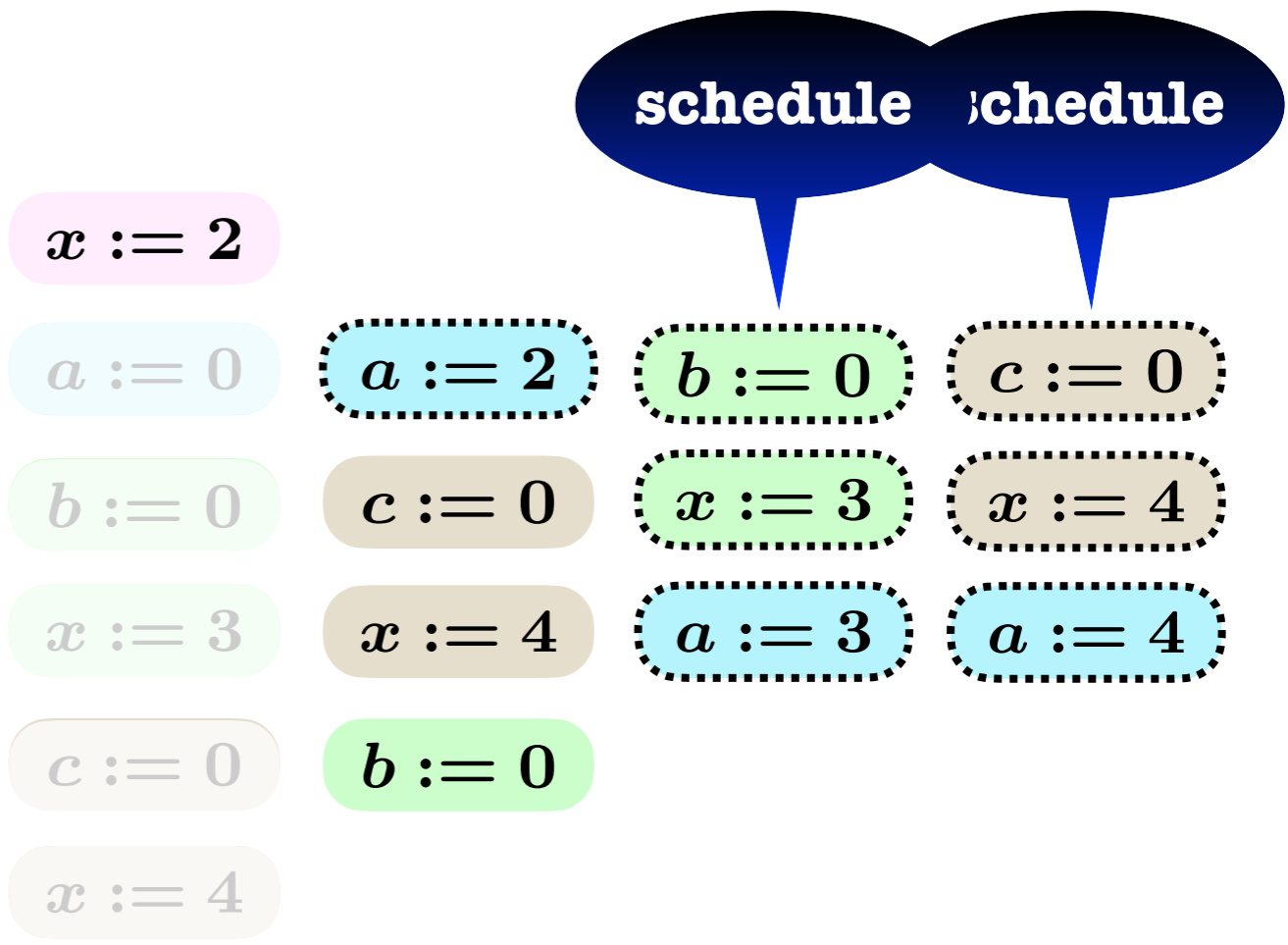
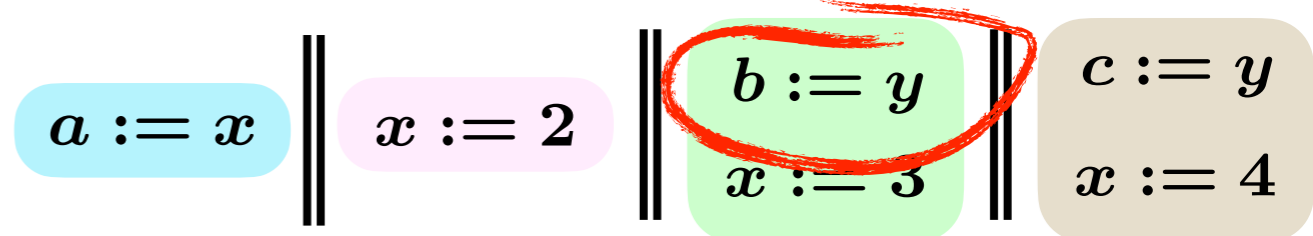
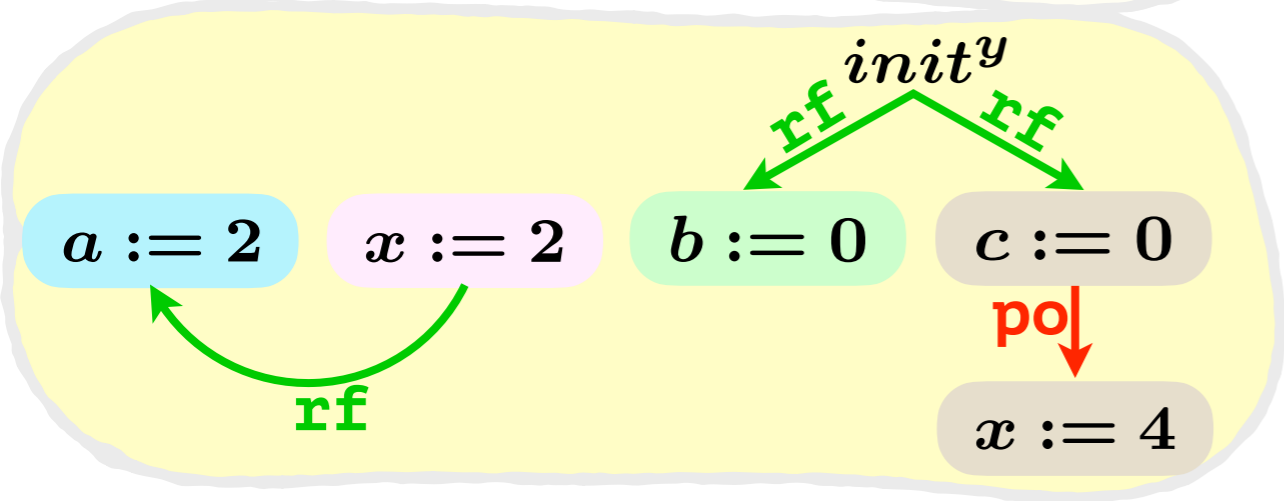
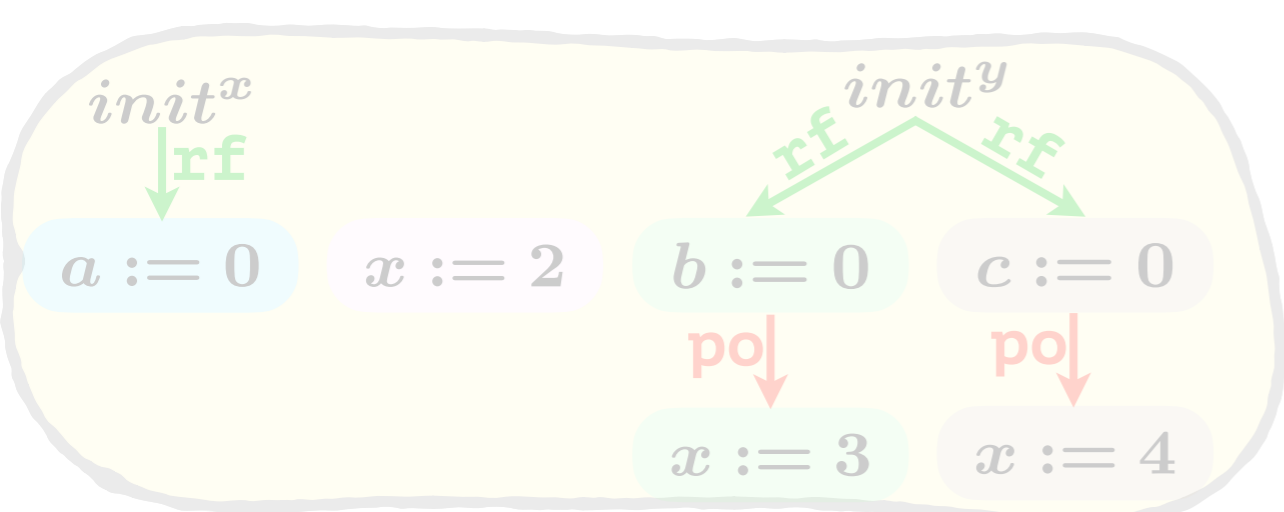
Postpone



Branch

&

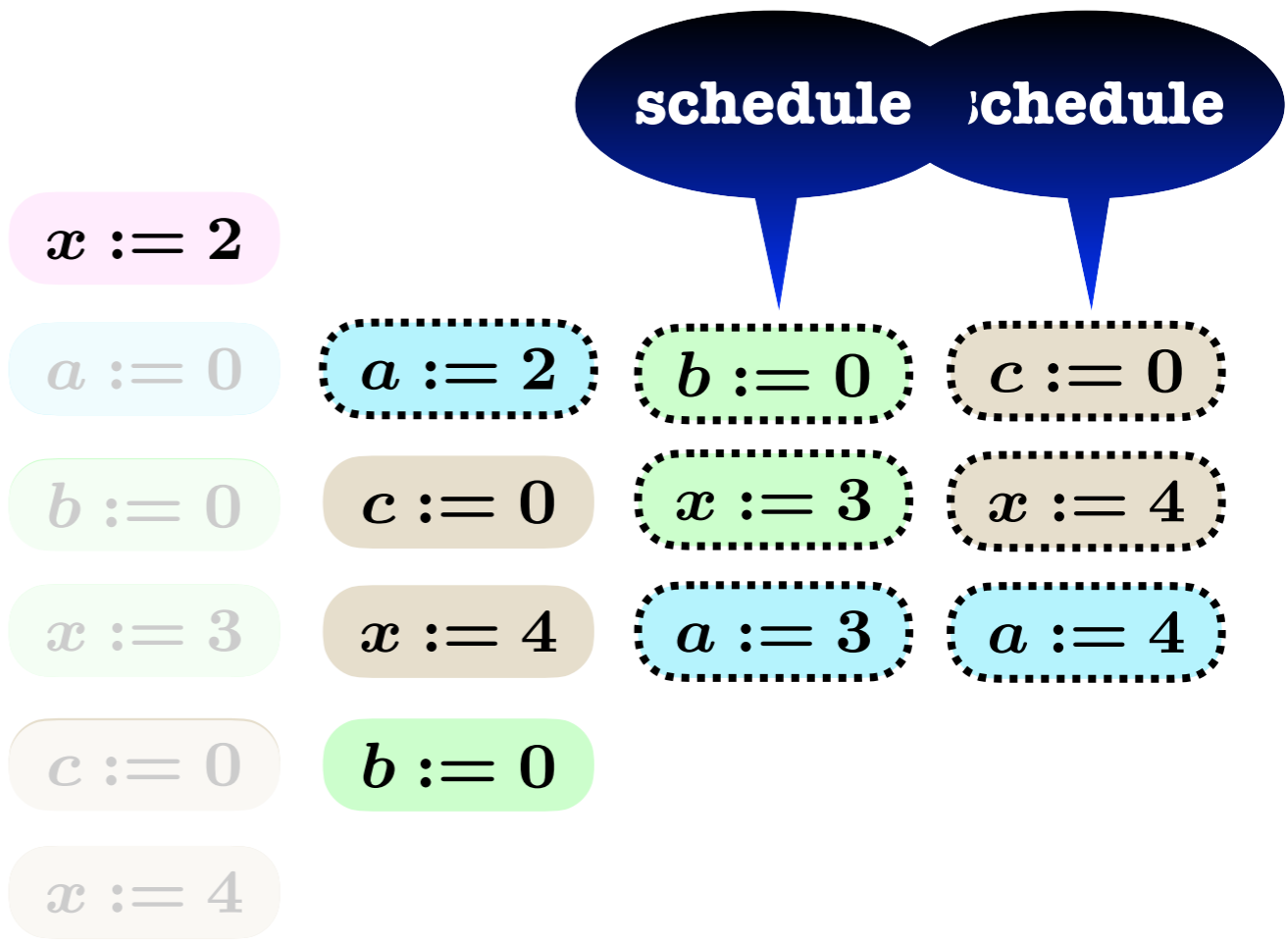
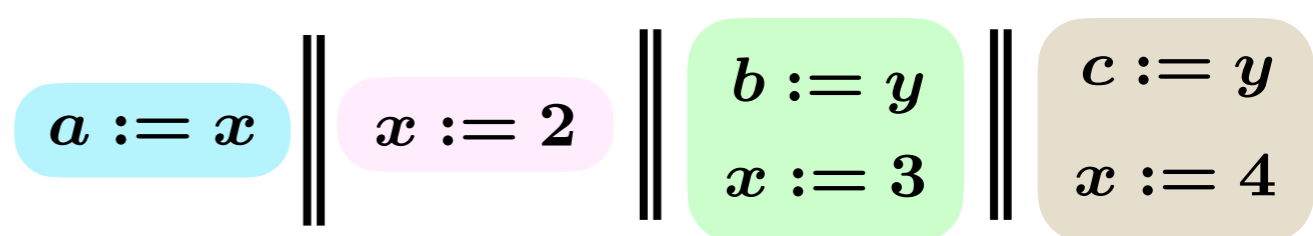
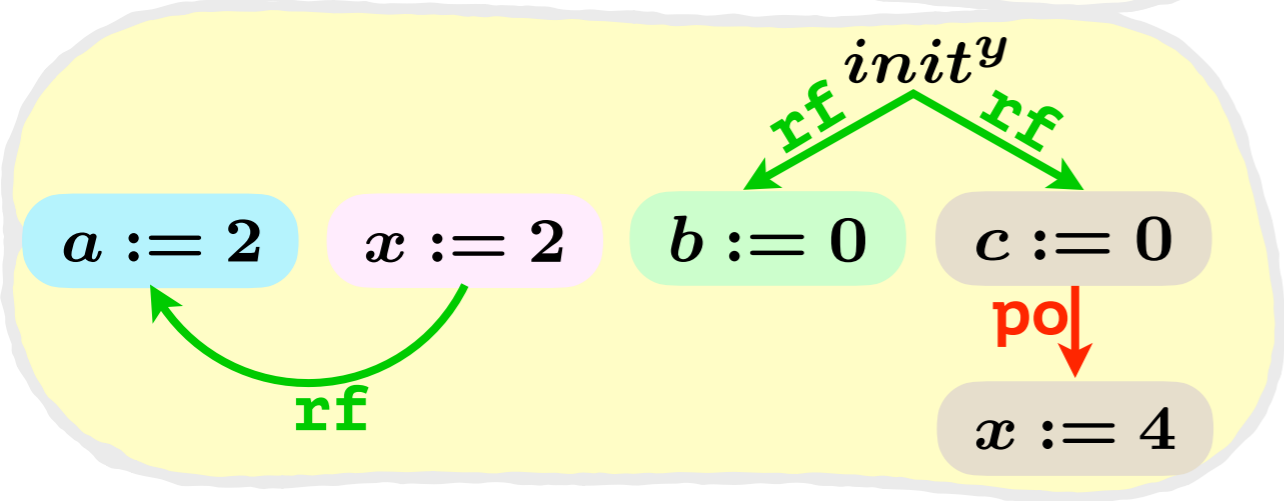
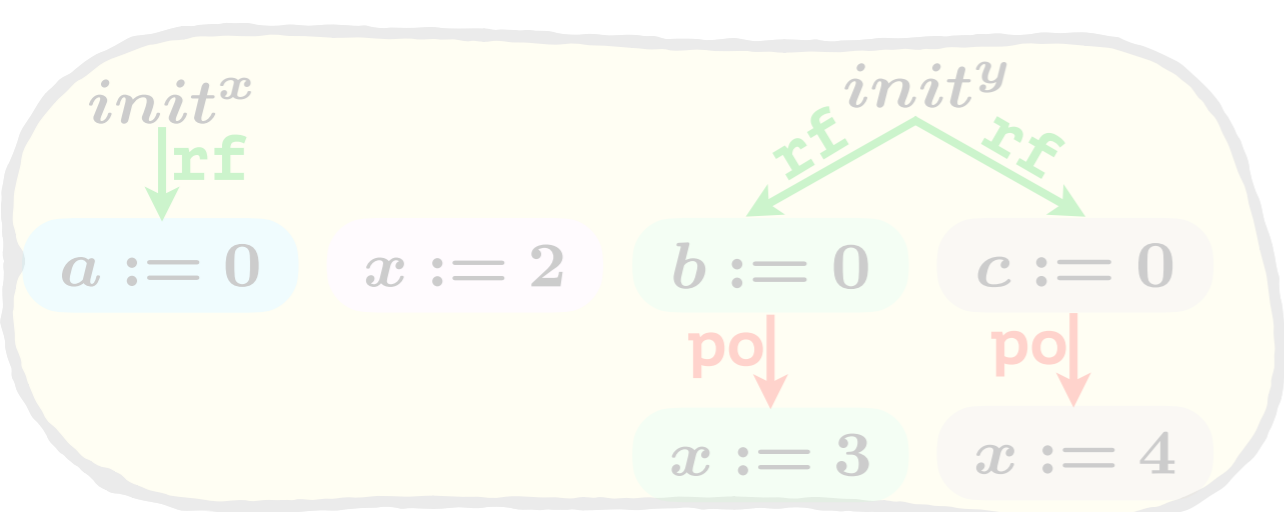
Postpone



Branch

&

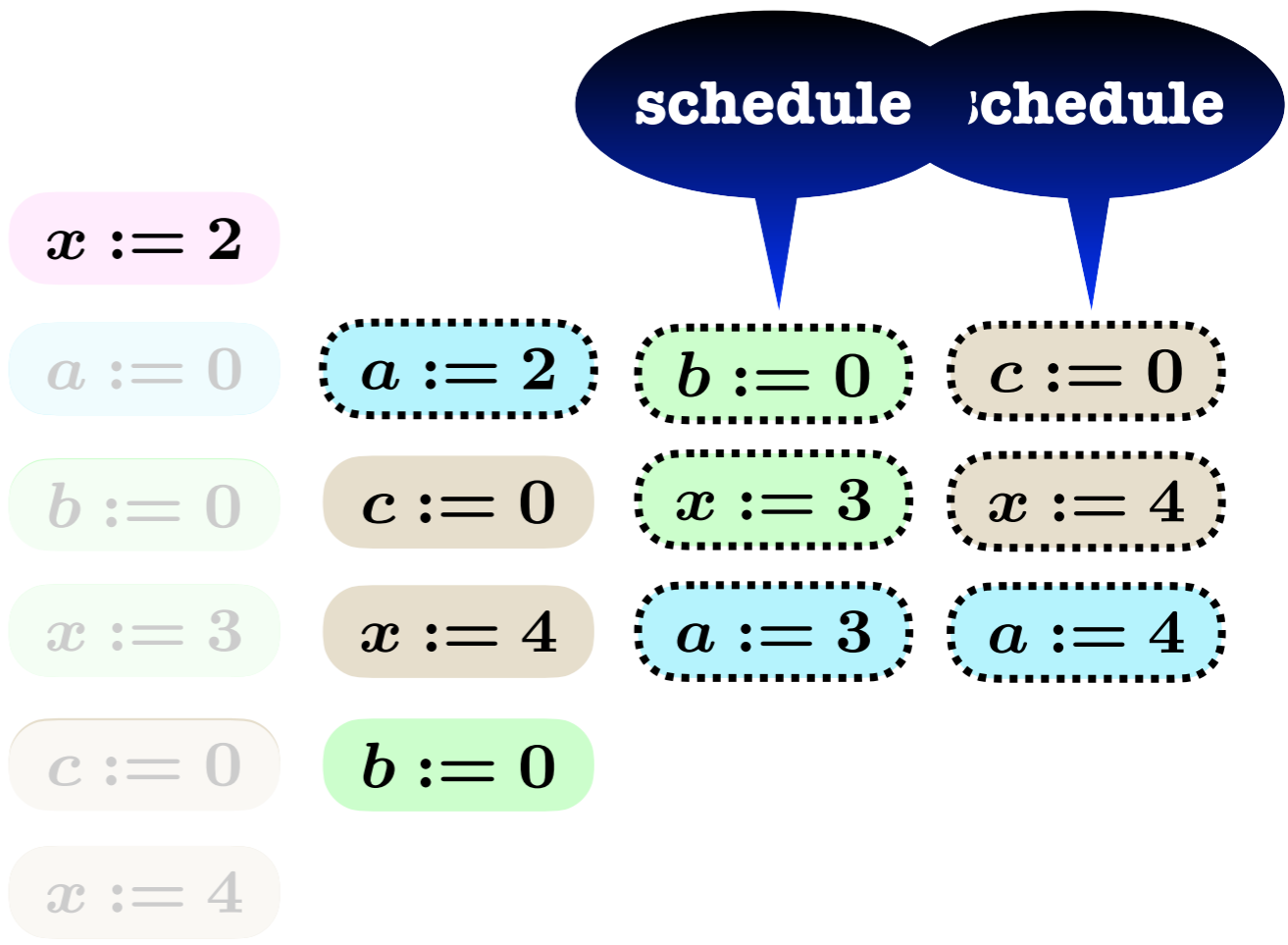
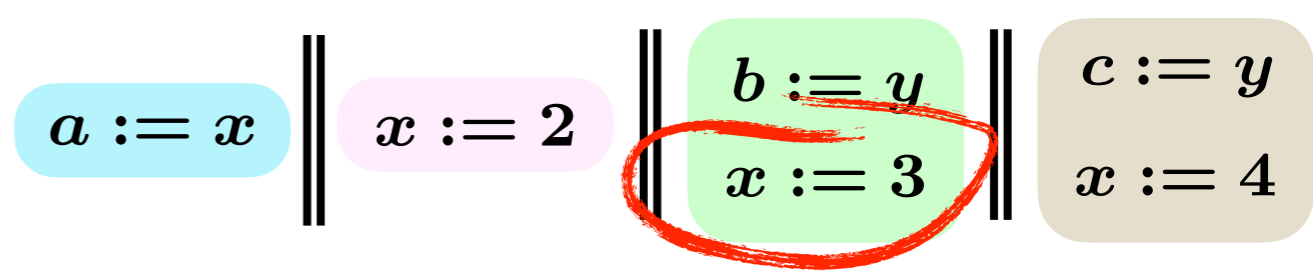
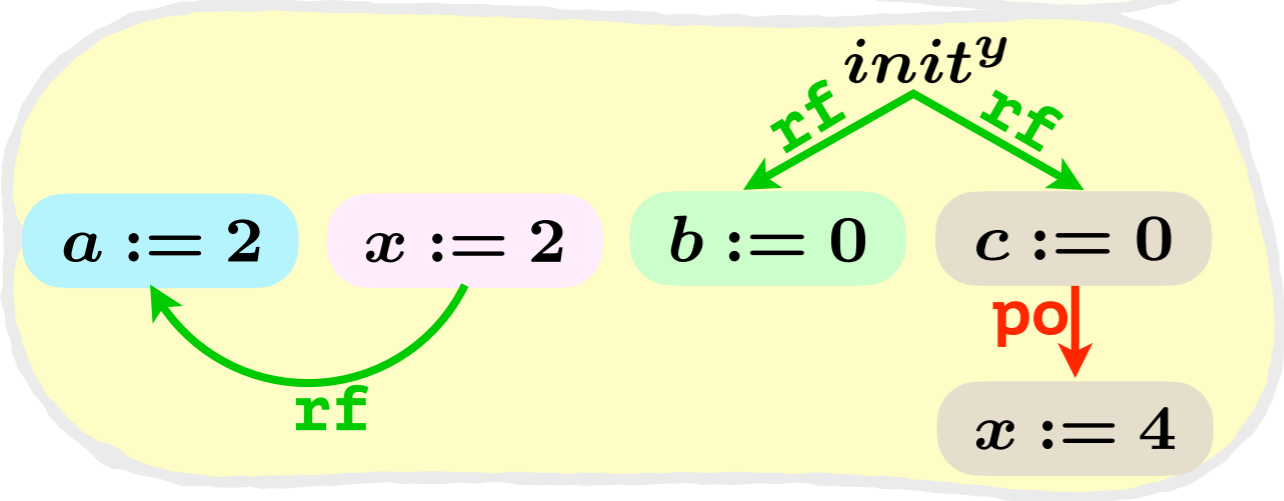
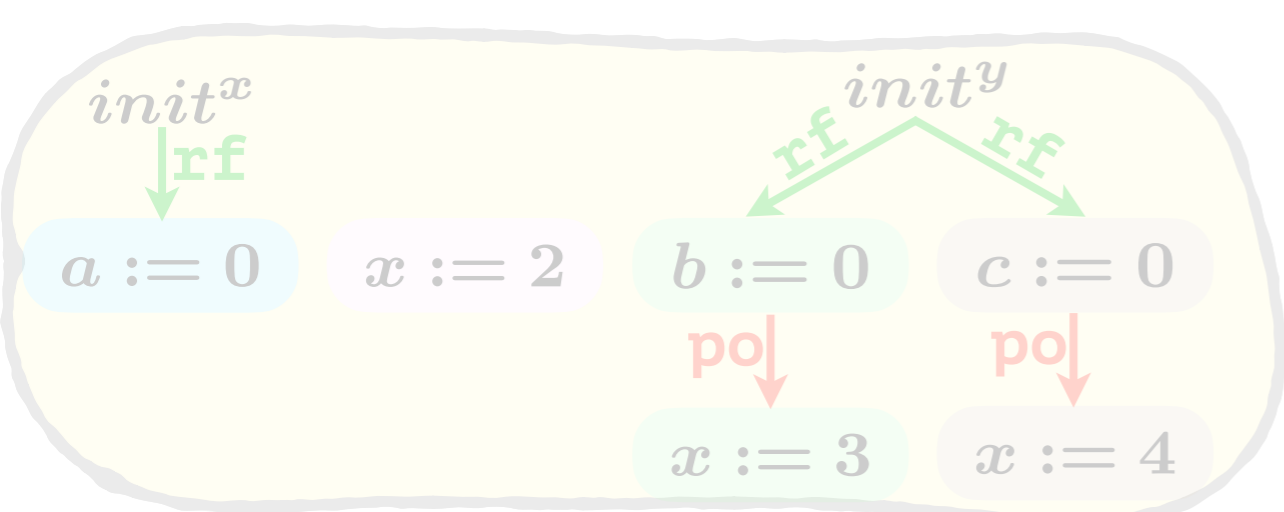
Postpone



Branch

&

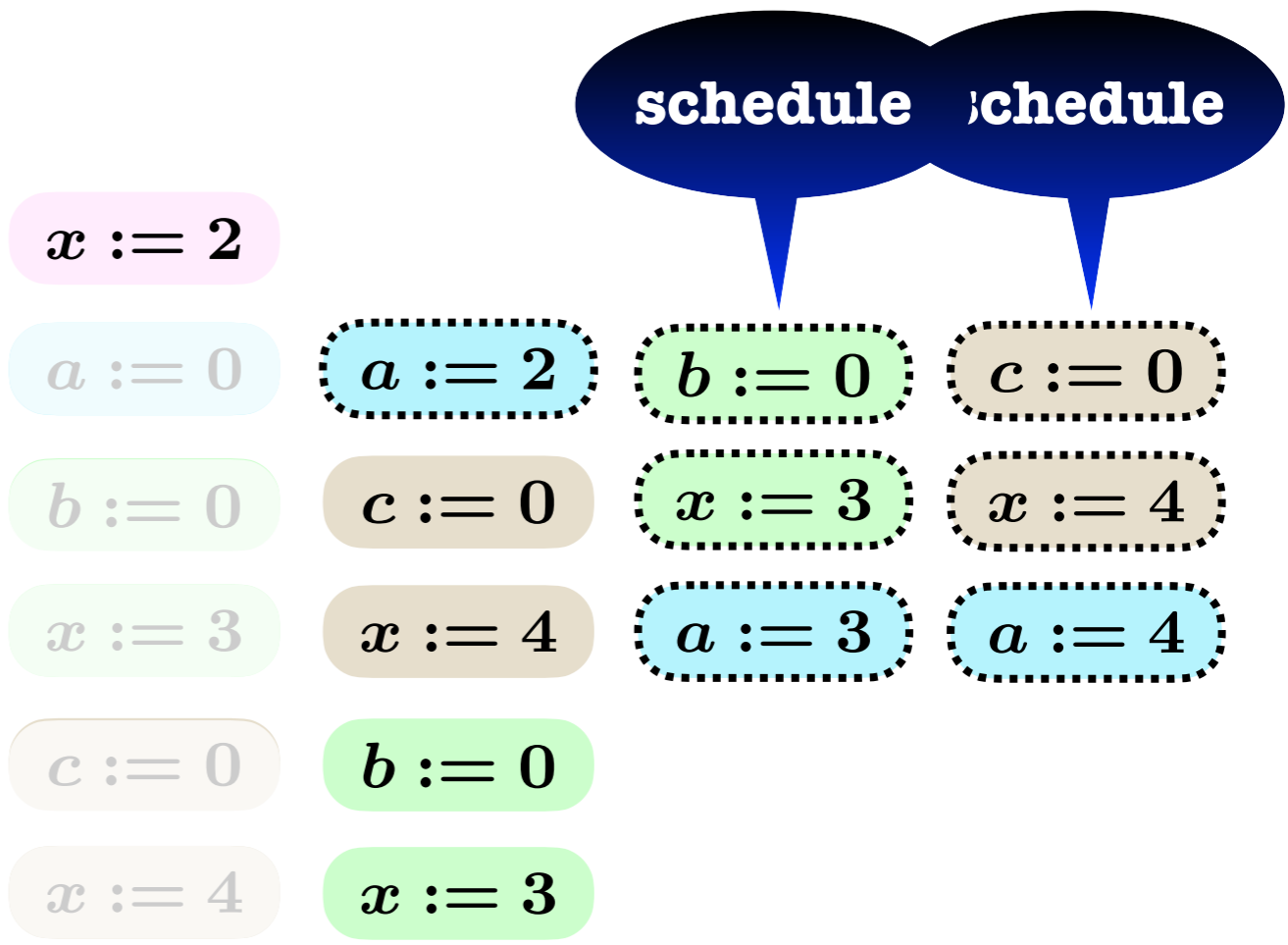
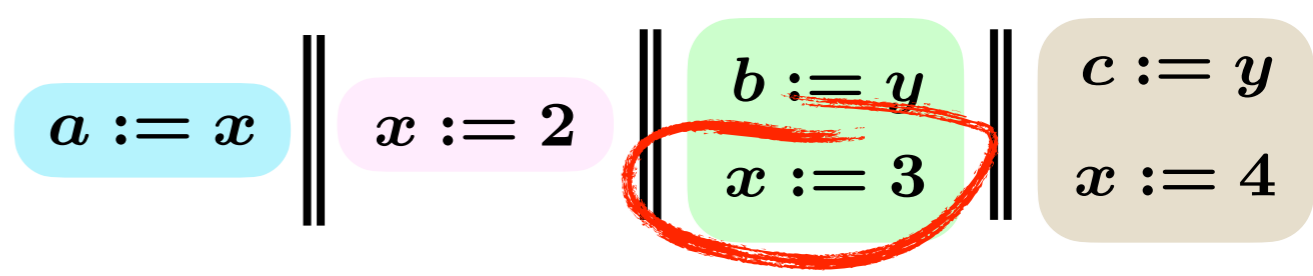
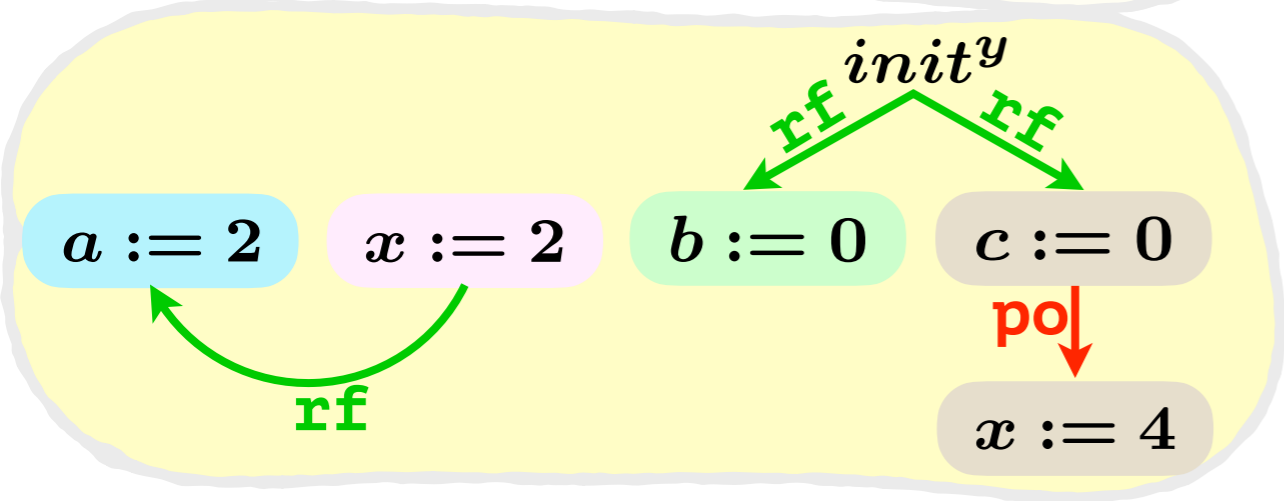
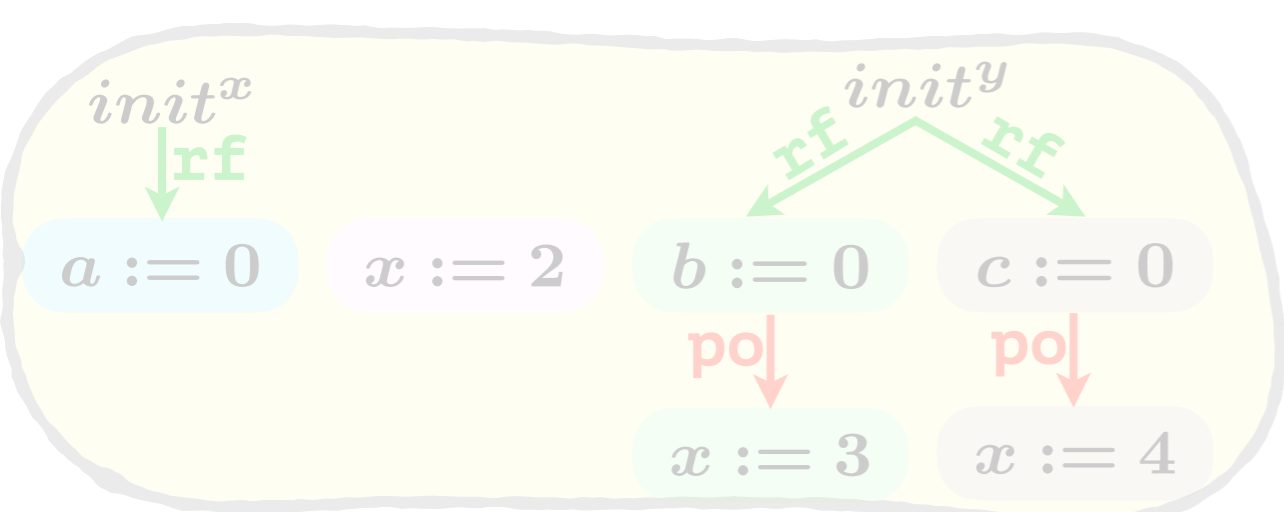
Postpone



Branch

&

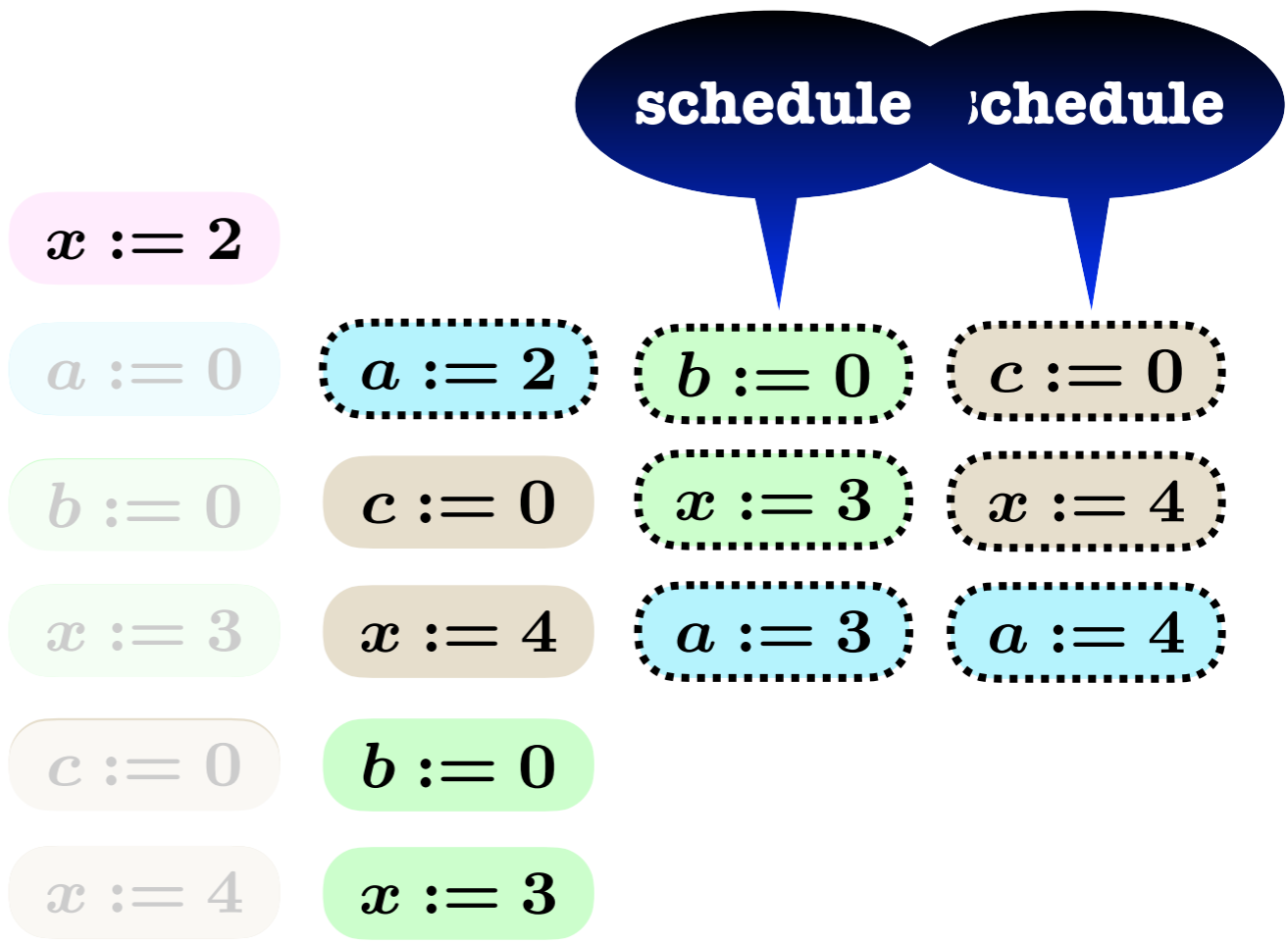
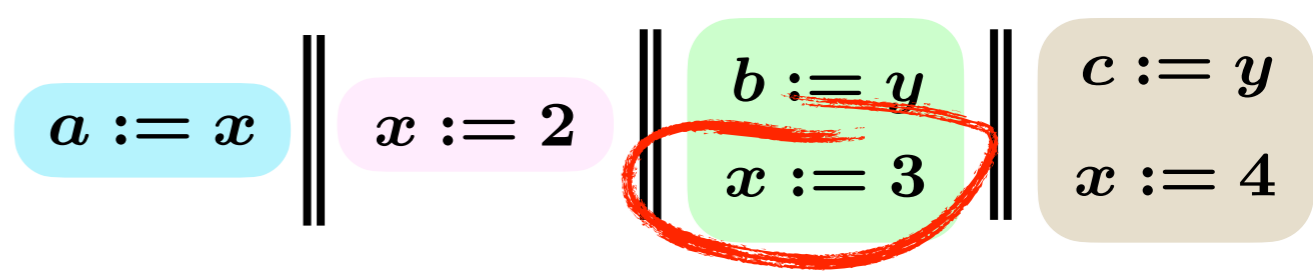
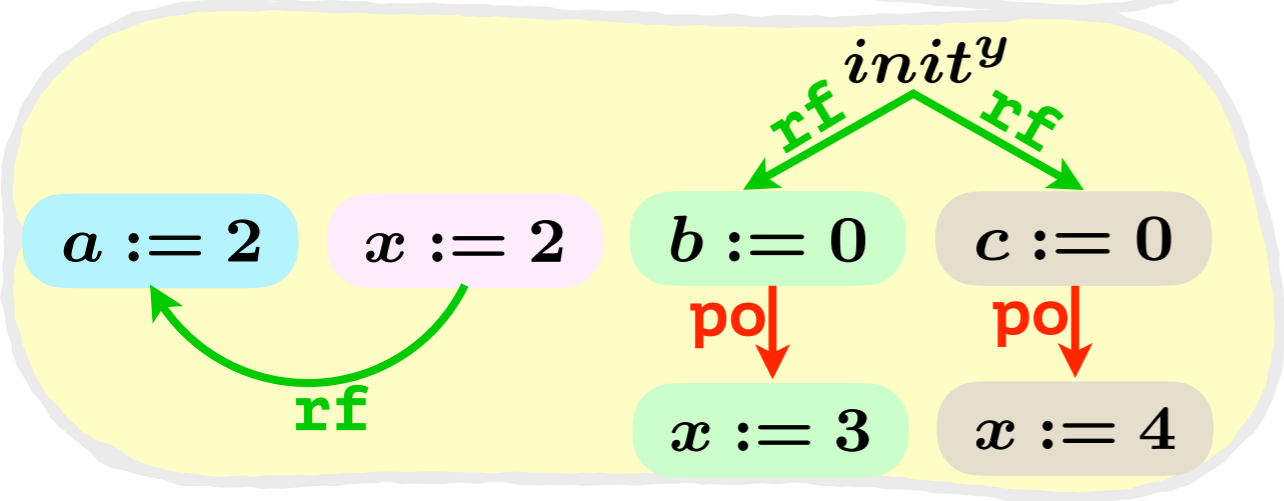
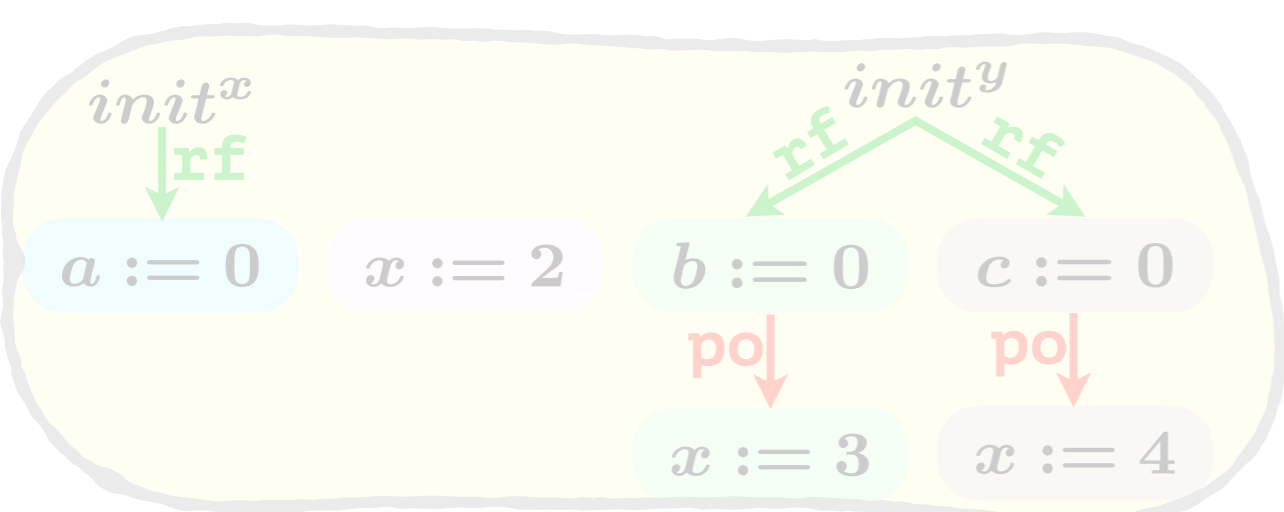
Postpone



Branch

&

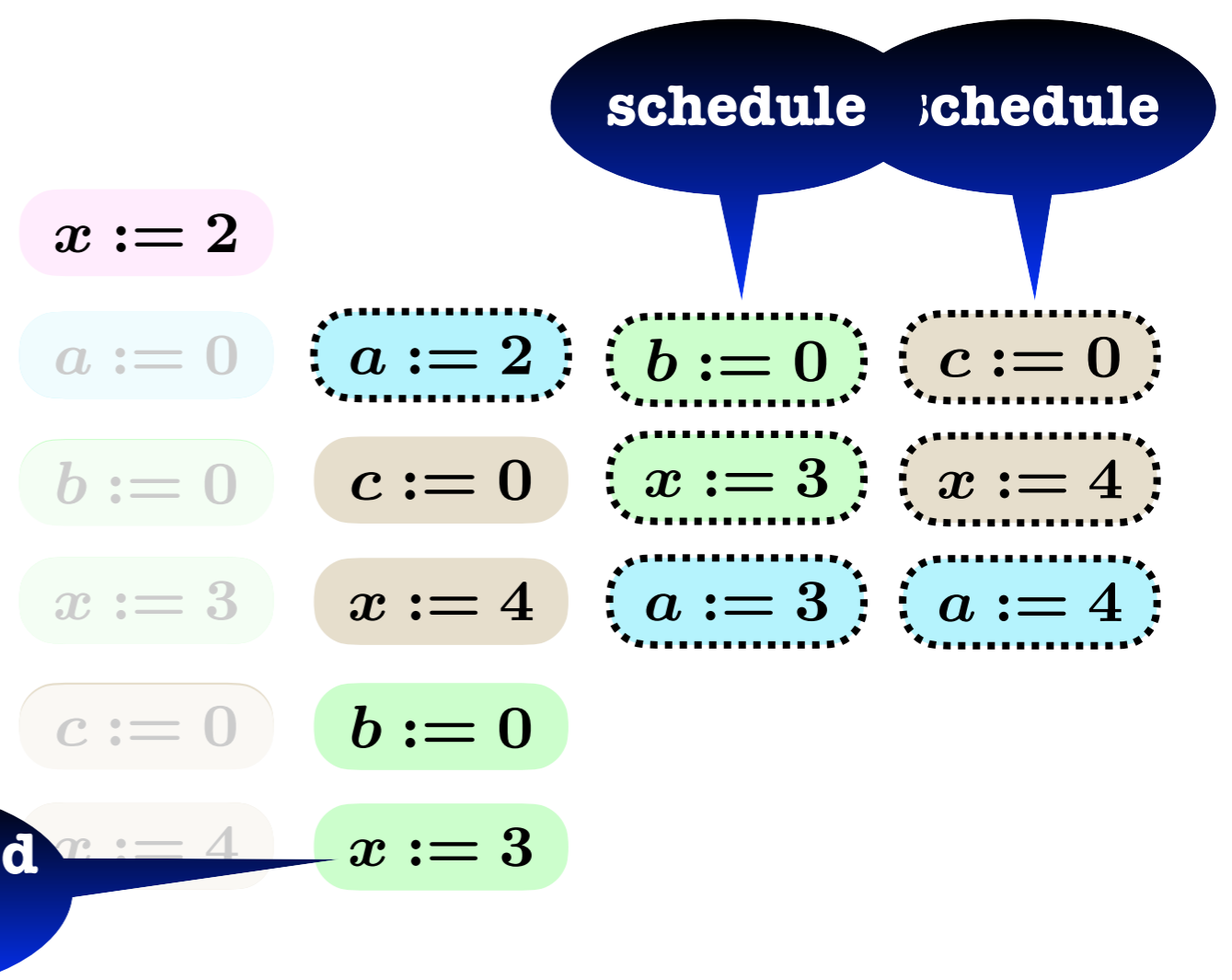
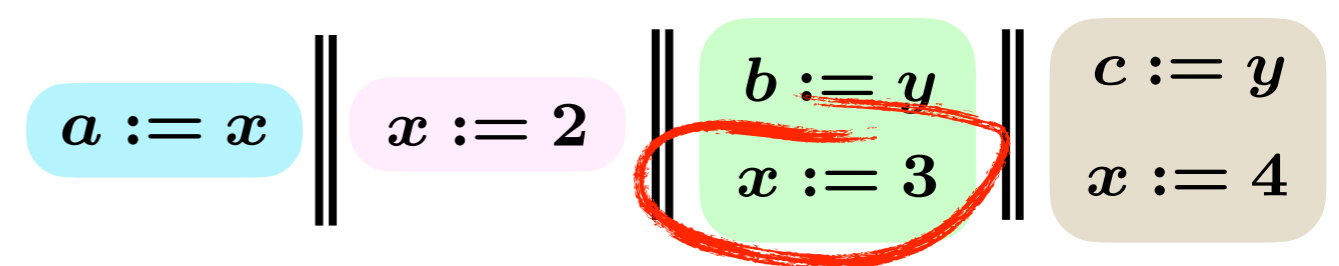
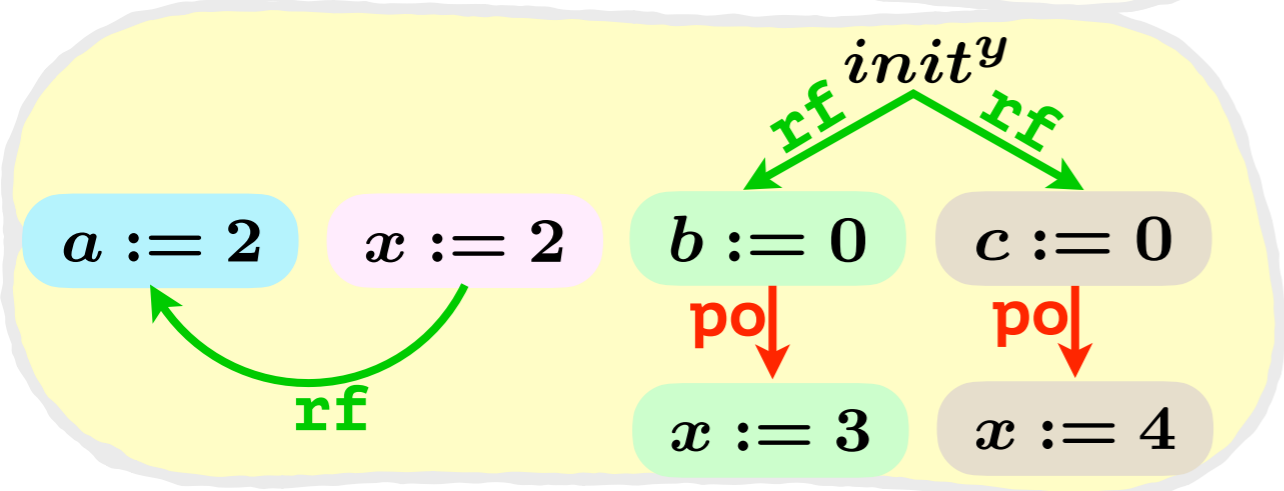
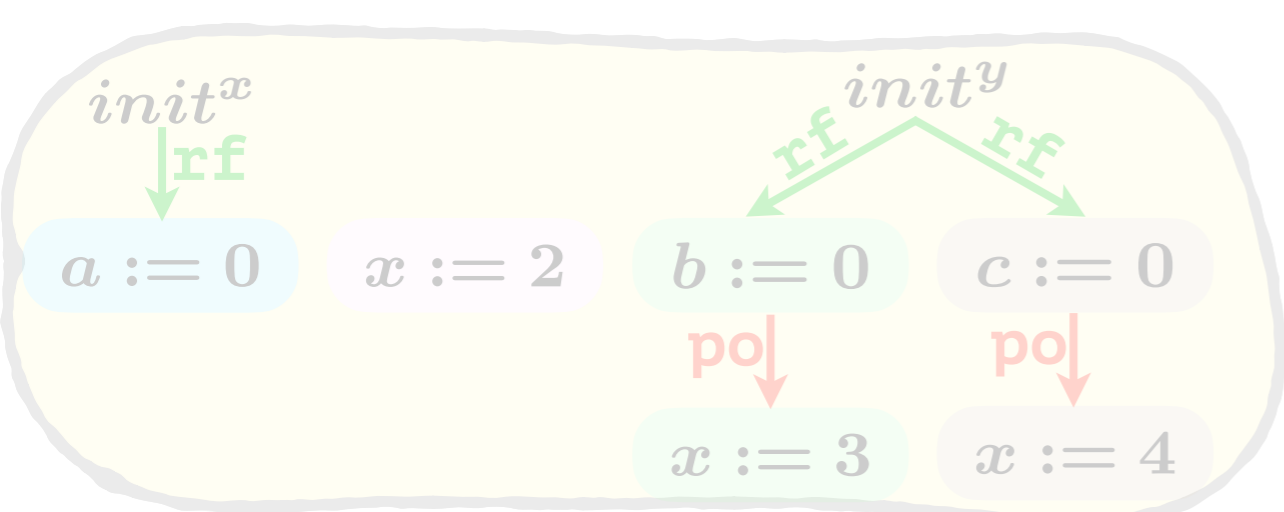
Postpone



Branch

&

Postpone

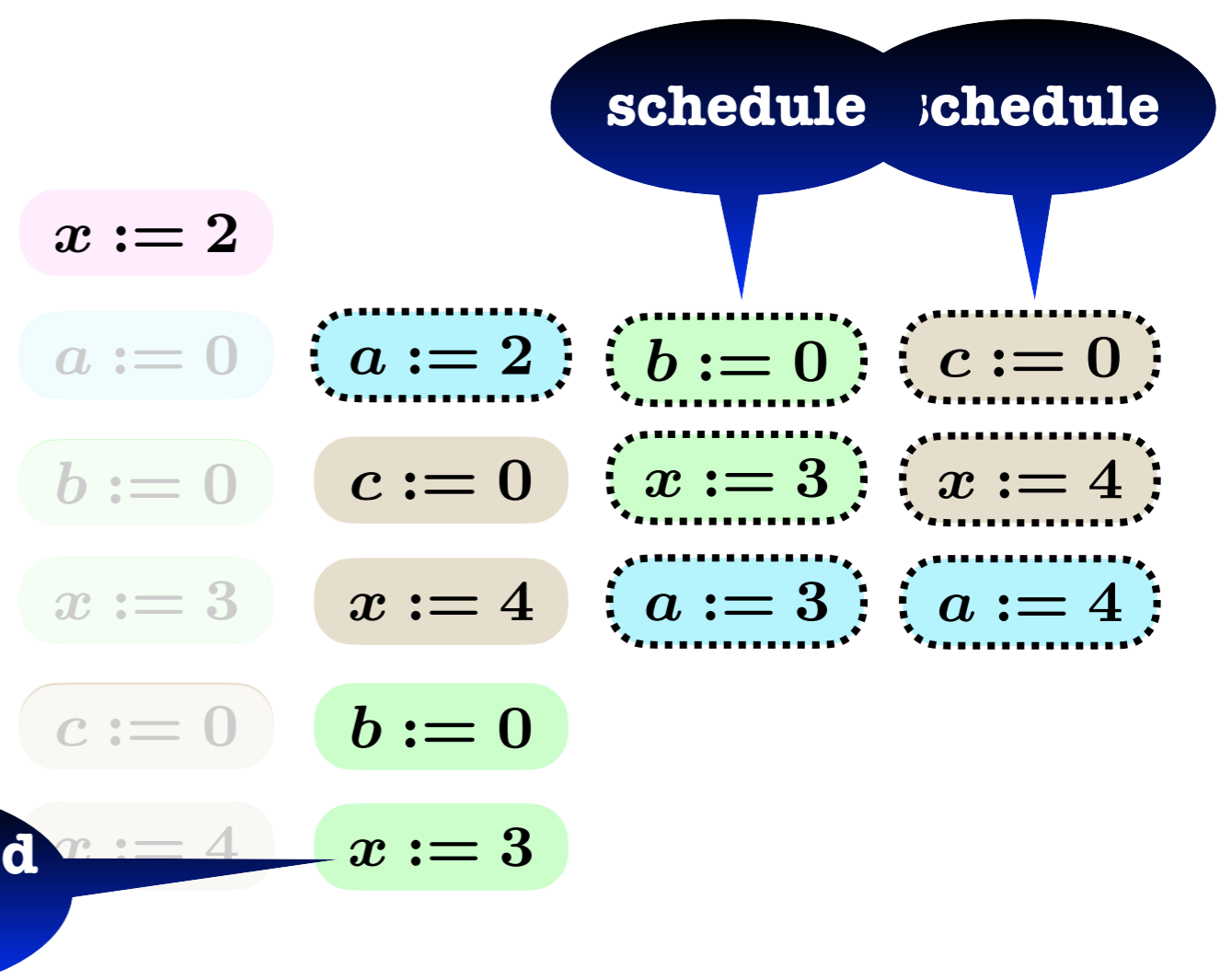
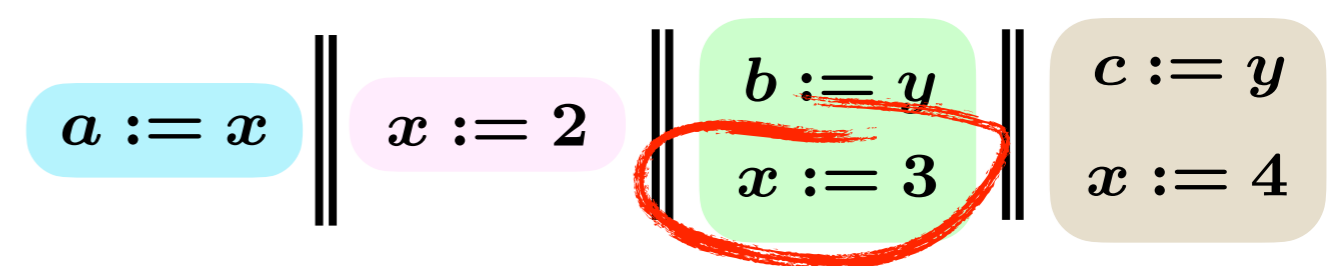
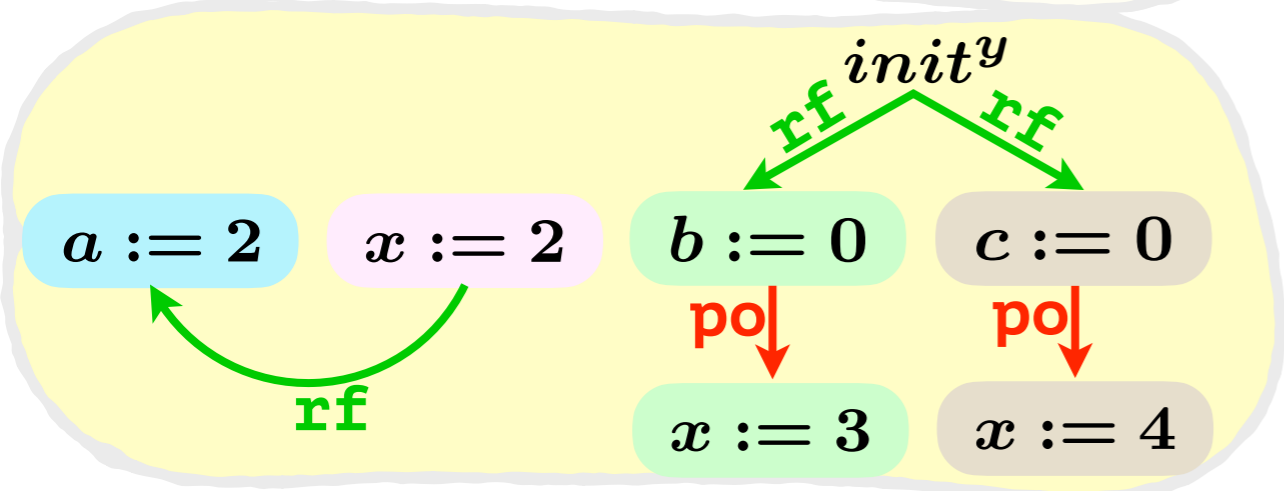
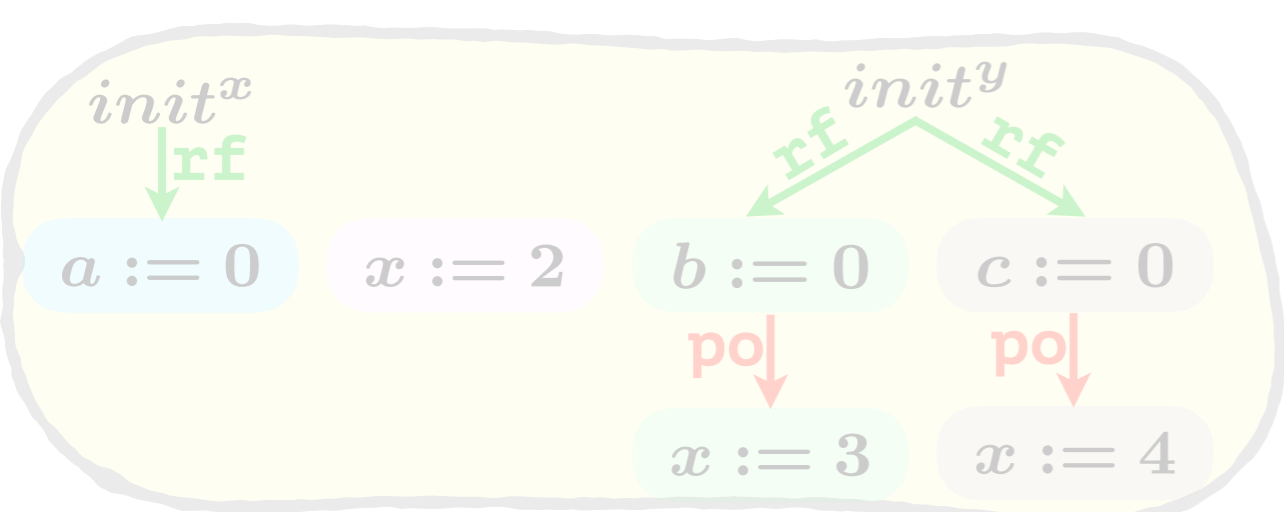


postponed write

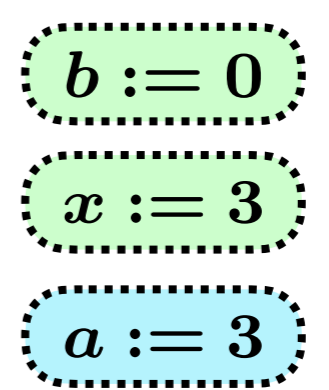
Branch

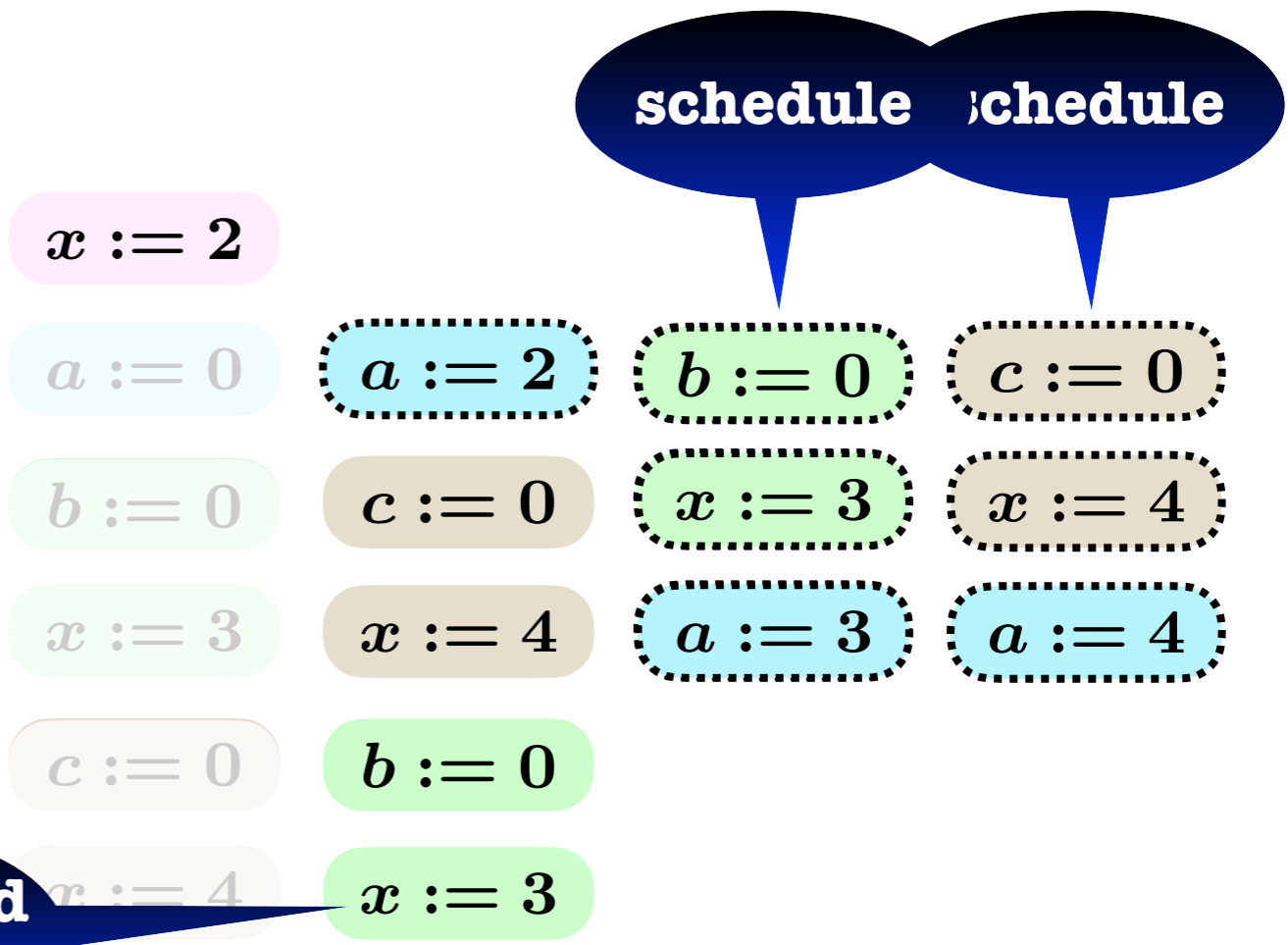
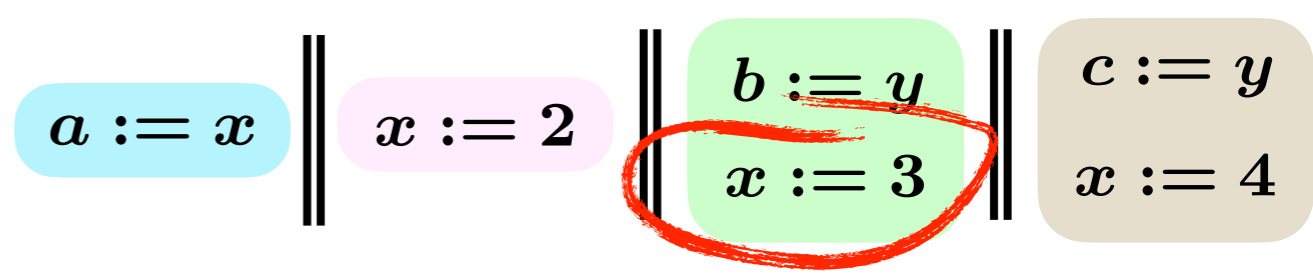
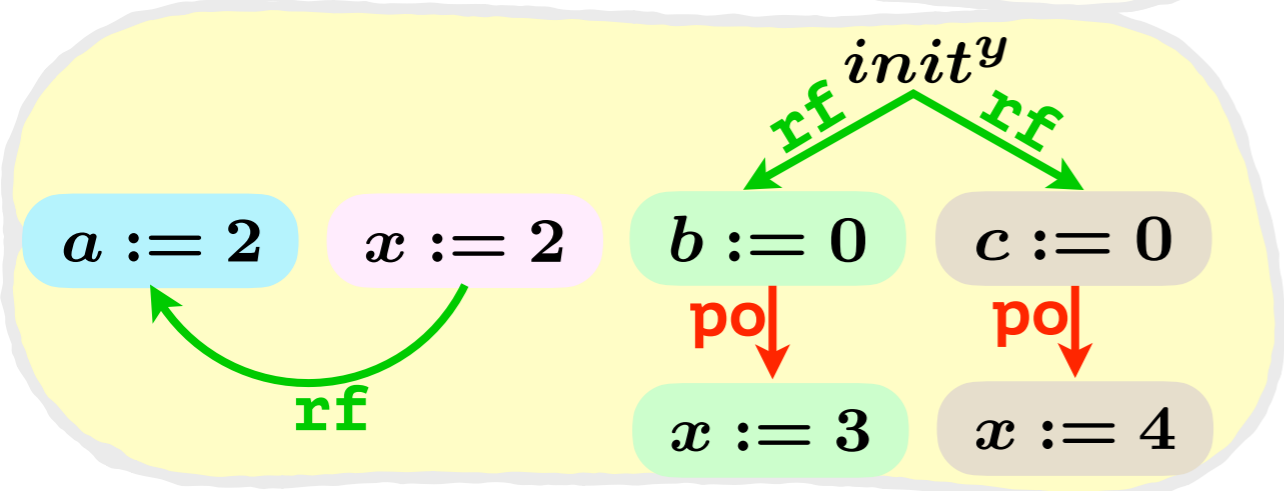
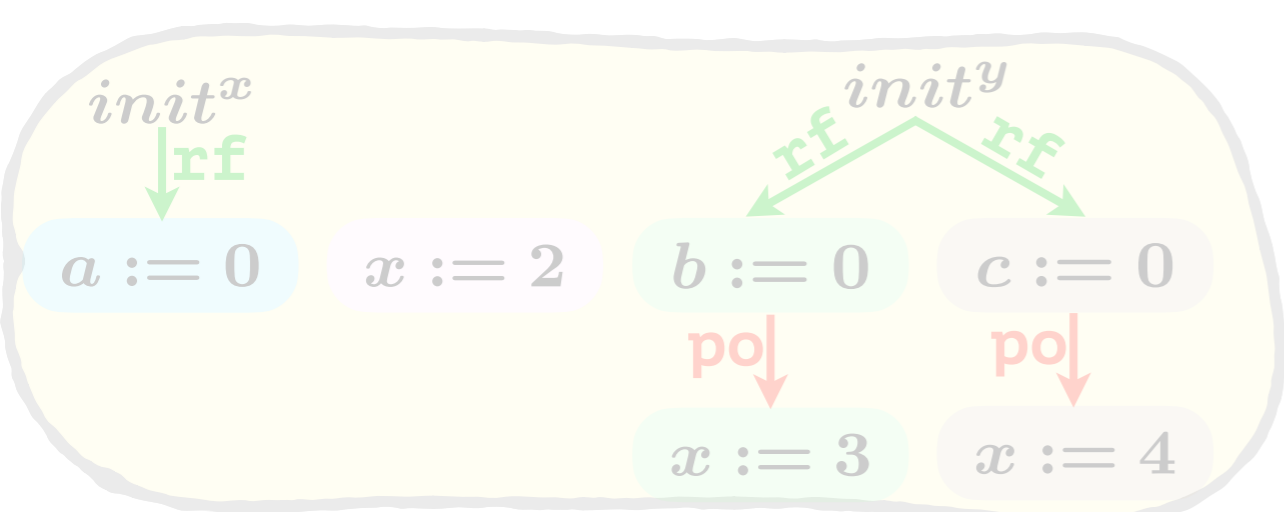
&

Postpone



Branch **&** **Postpone**



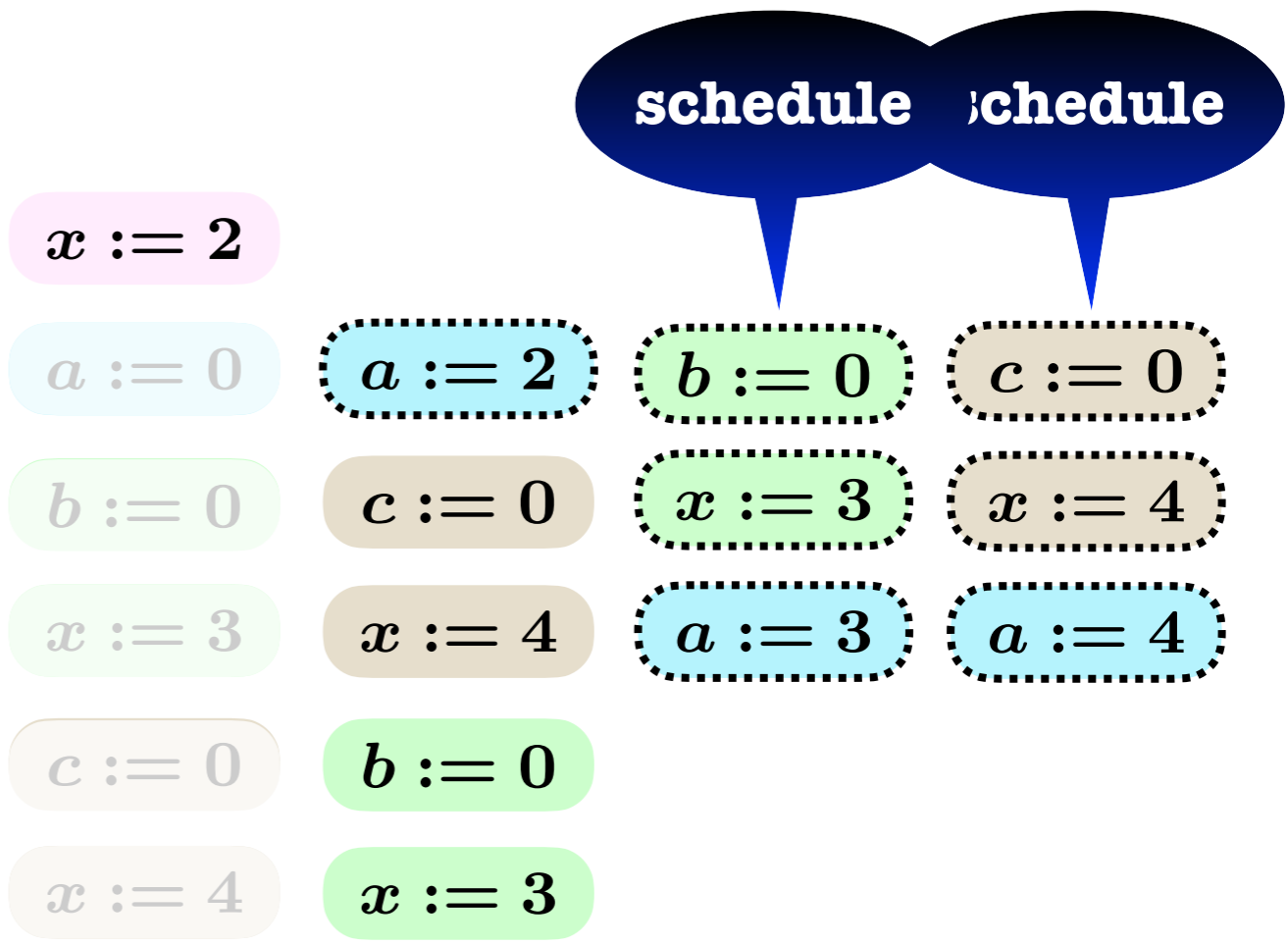
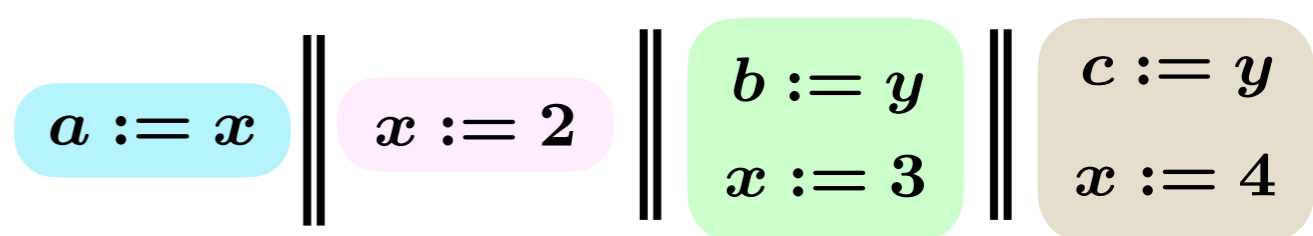
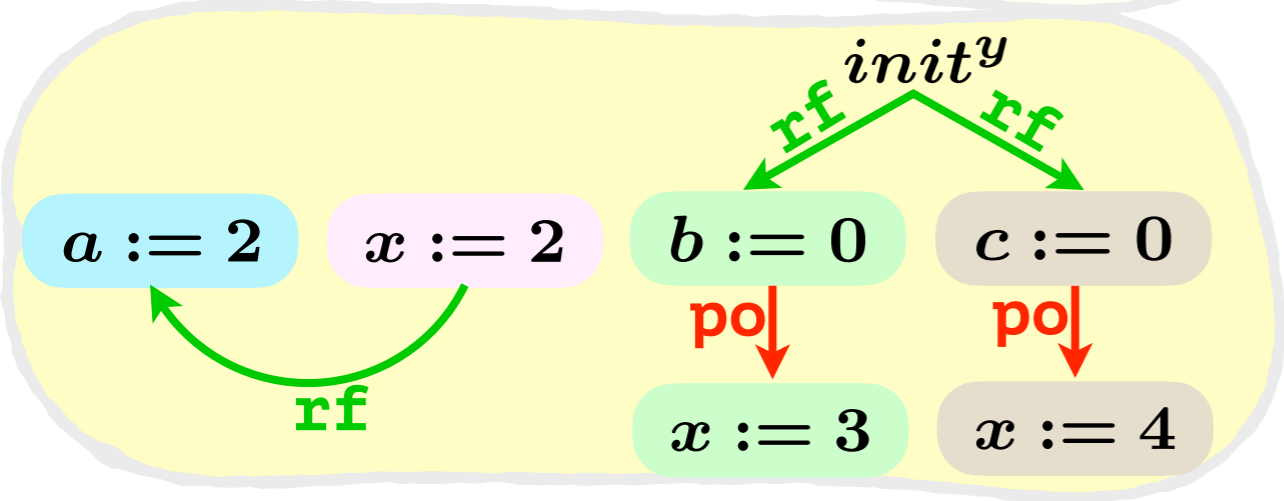
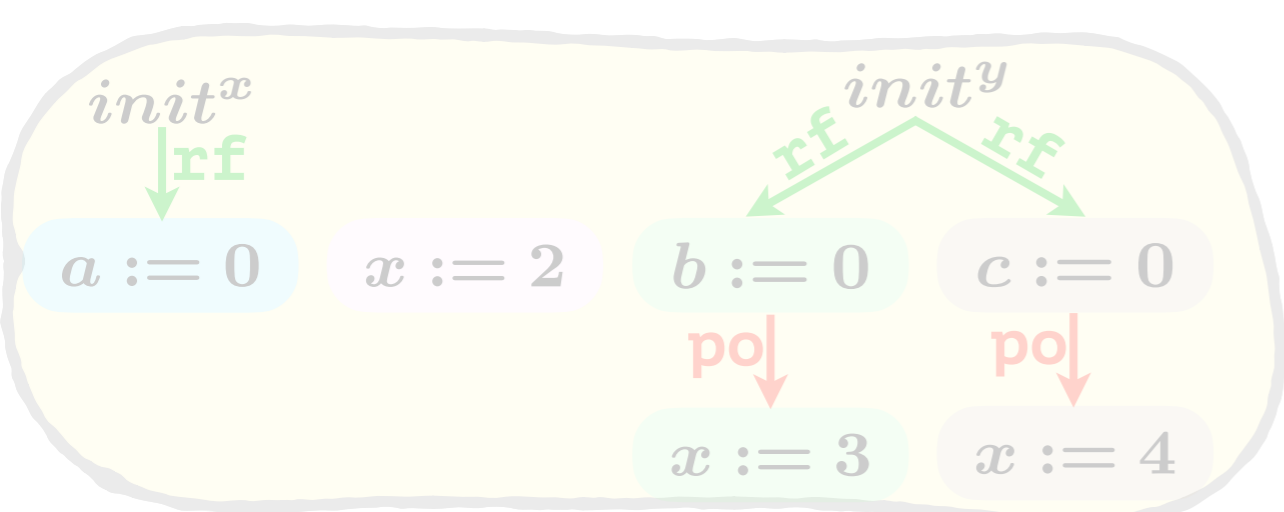


postponed write

Branch

&

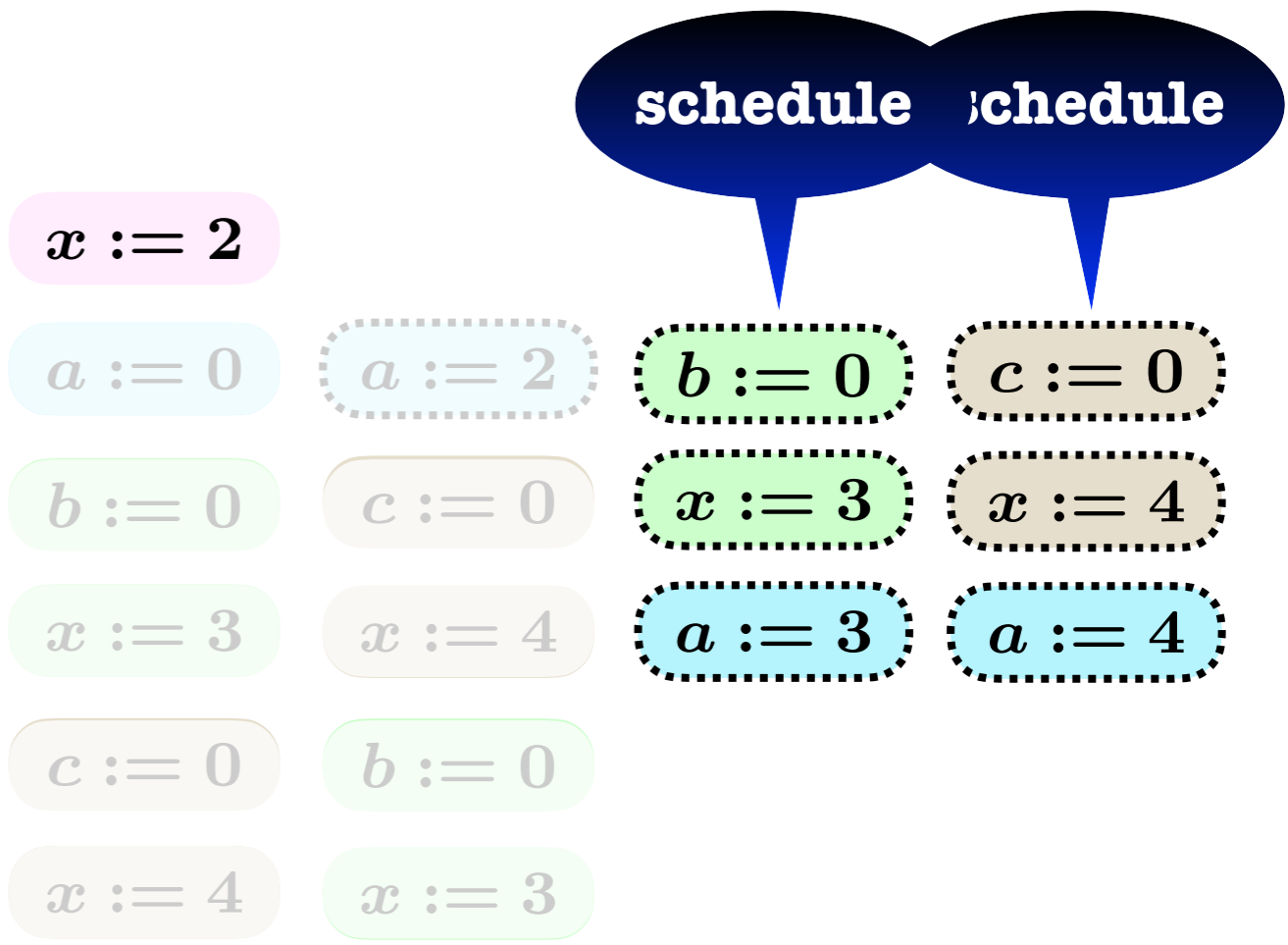
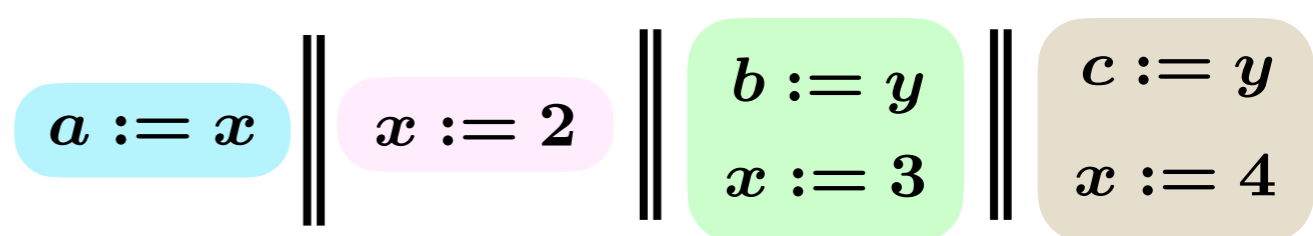
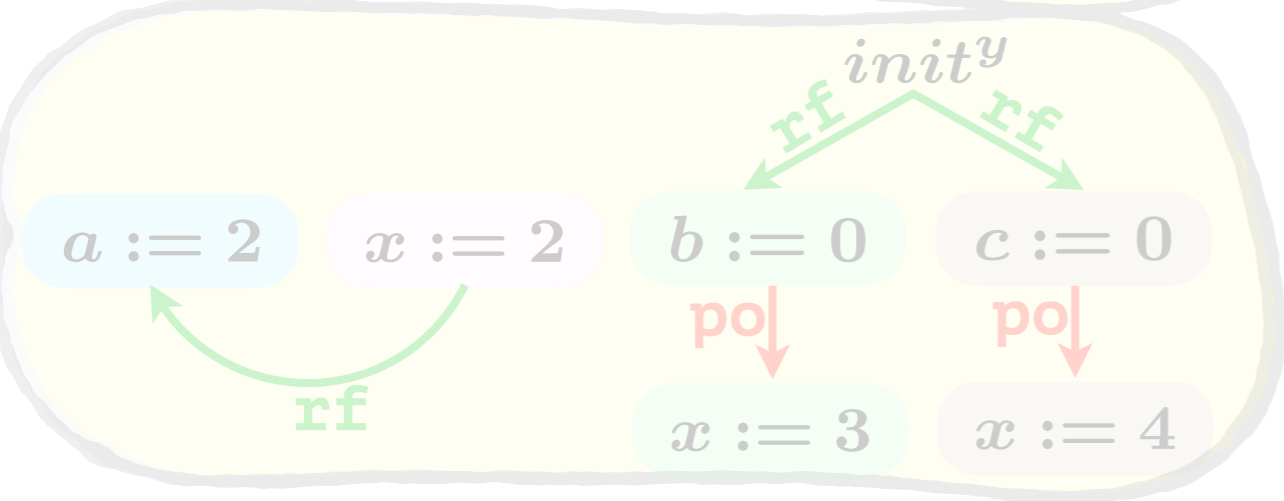
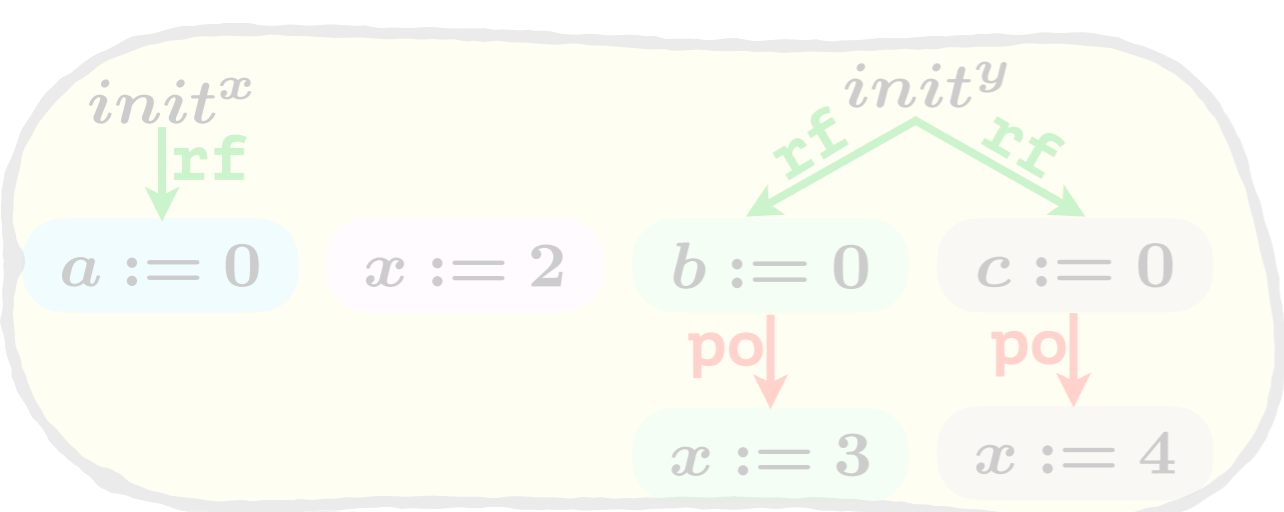
Postpone



Branch

&

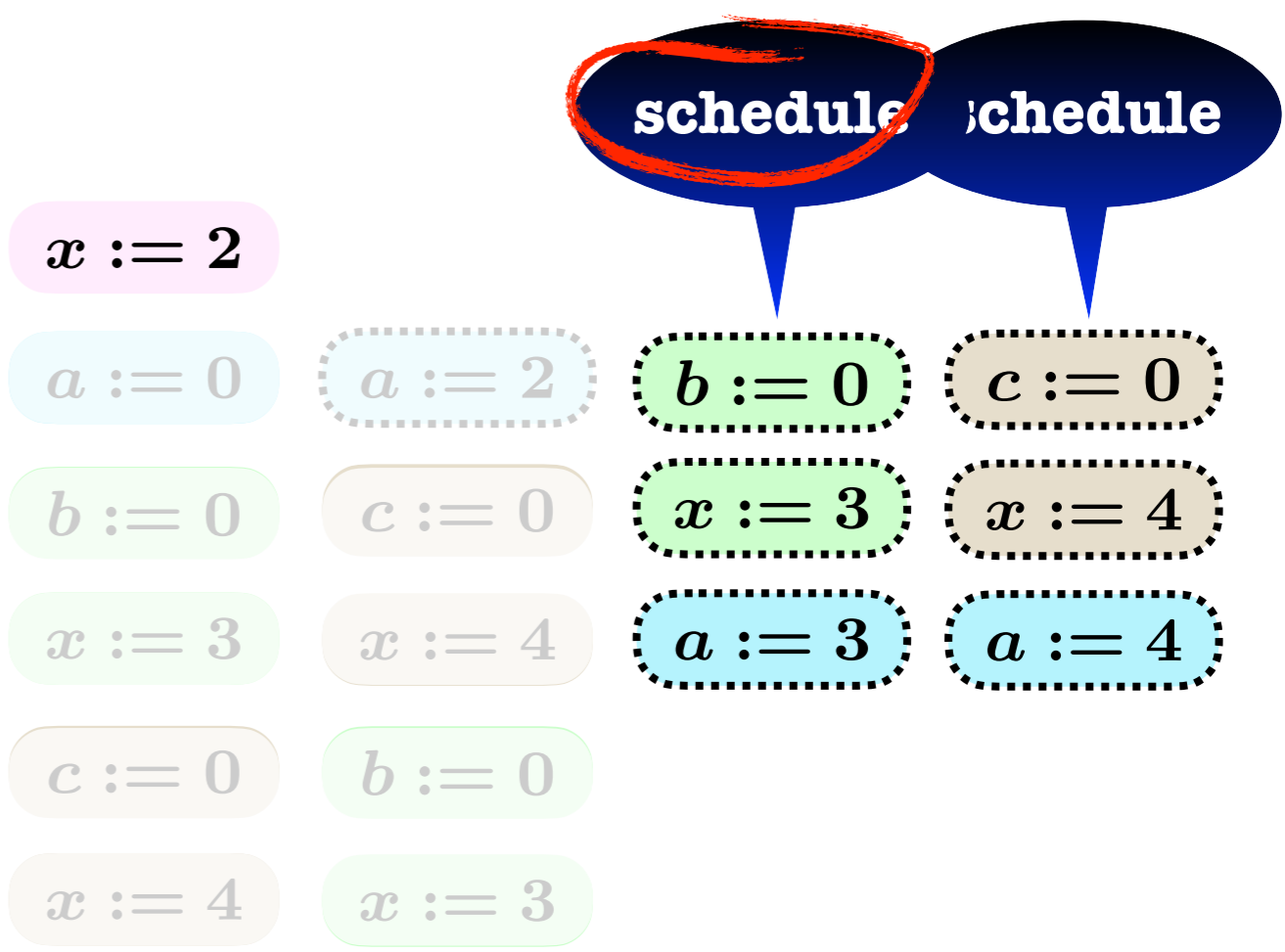
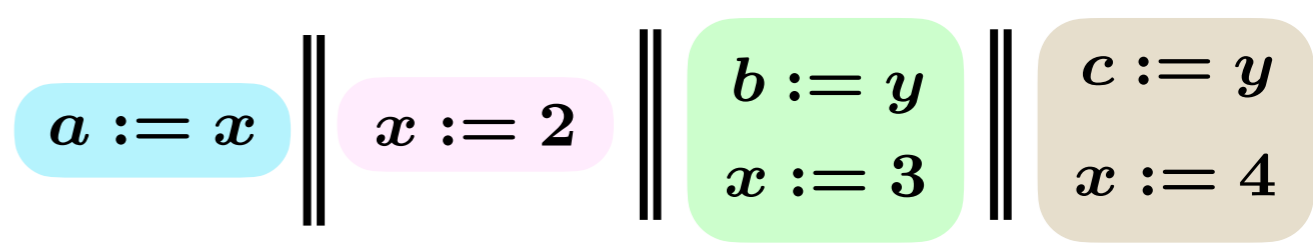
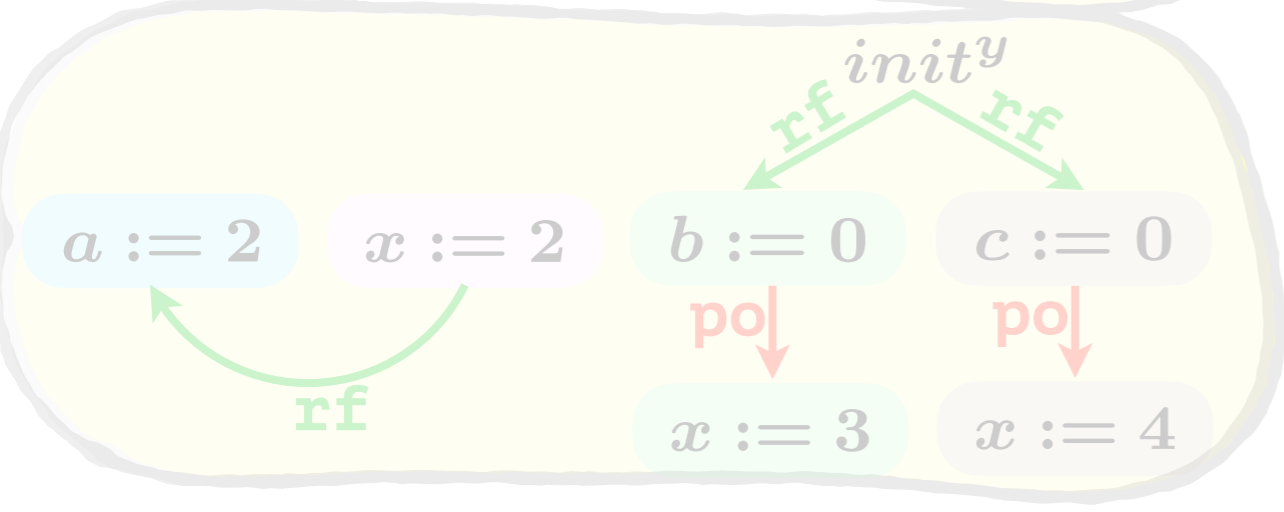
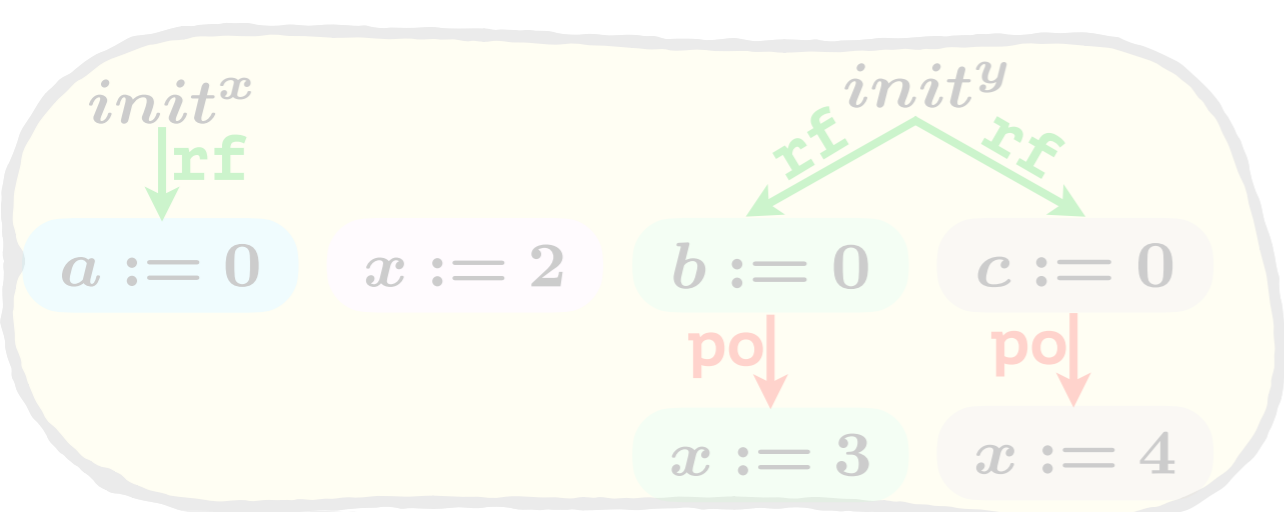
Postpone



Branch

&

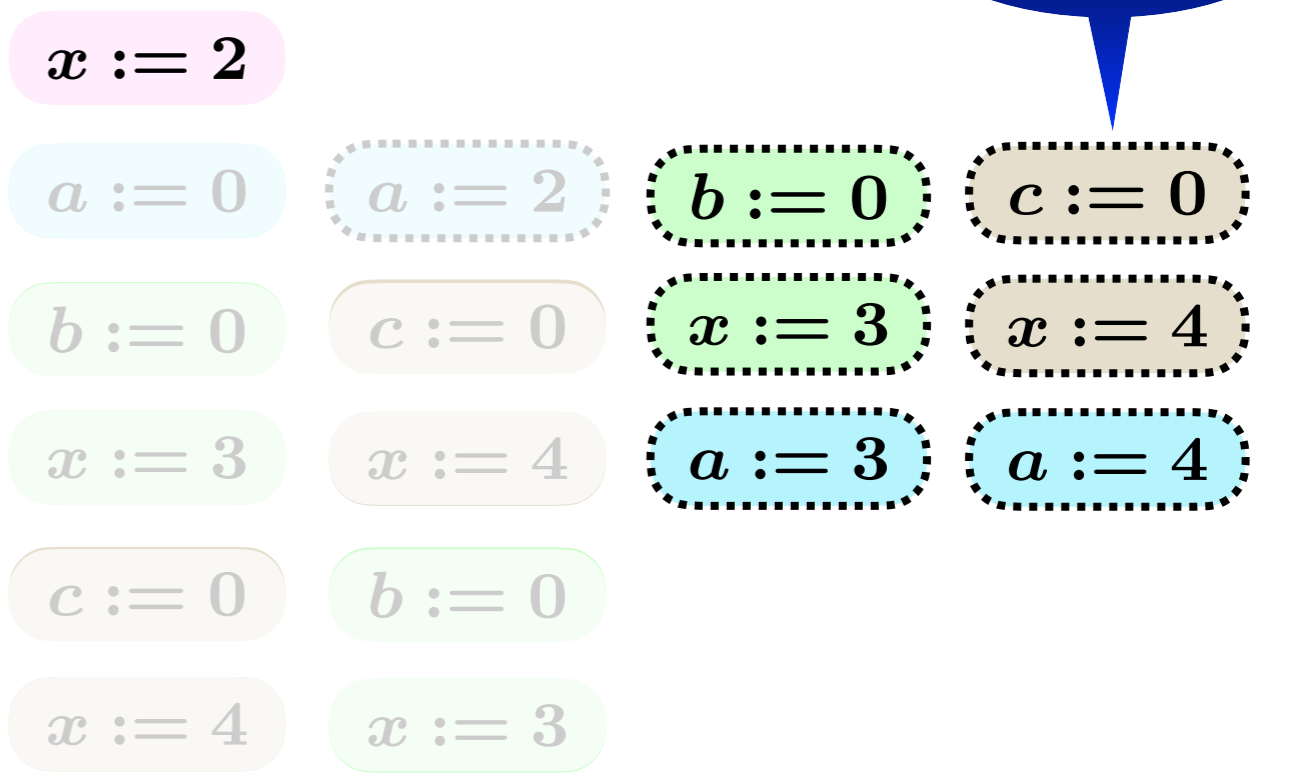
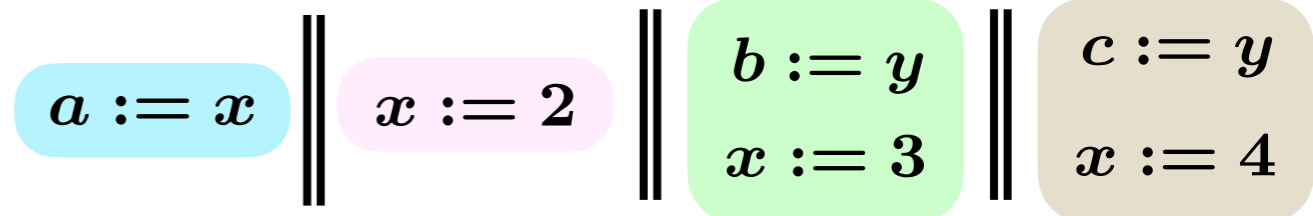
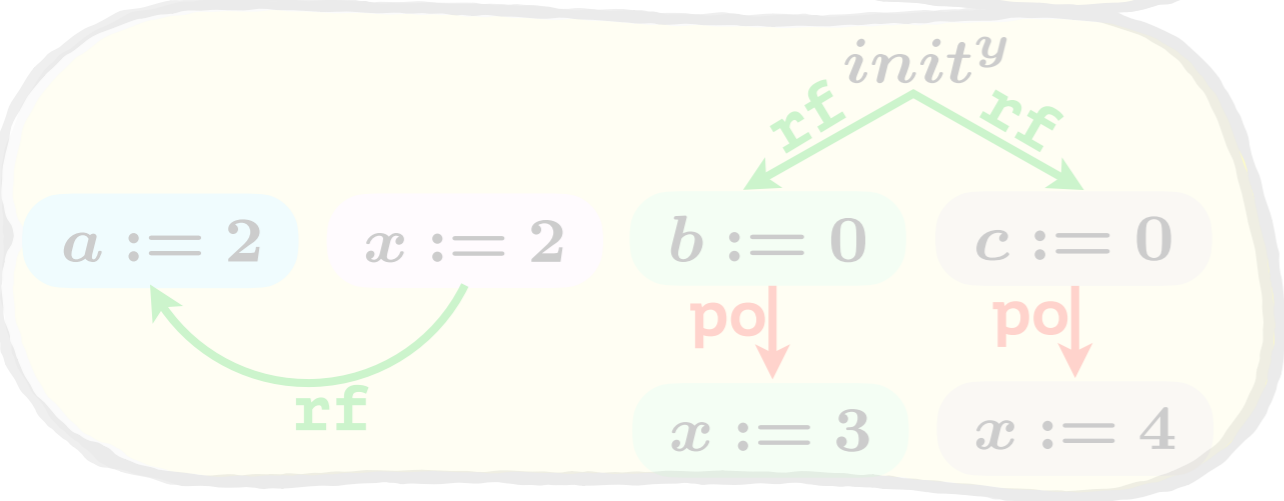
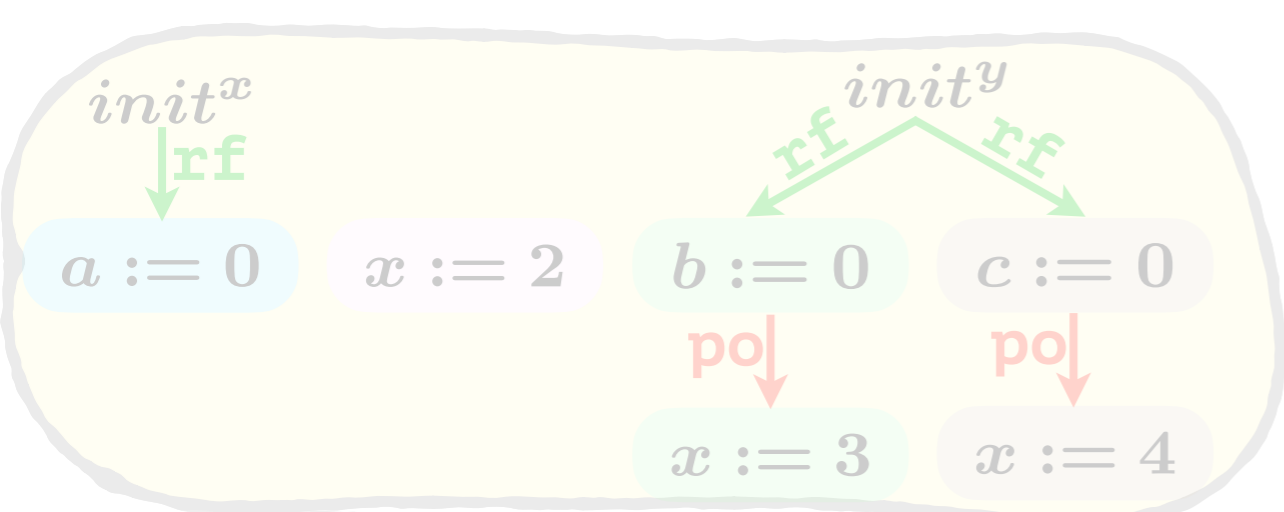
Postpone



Branch

&

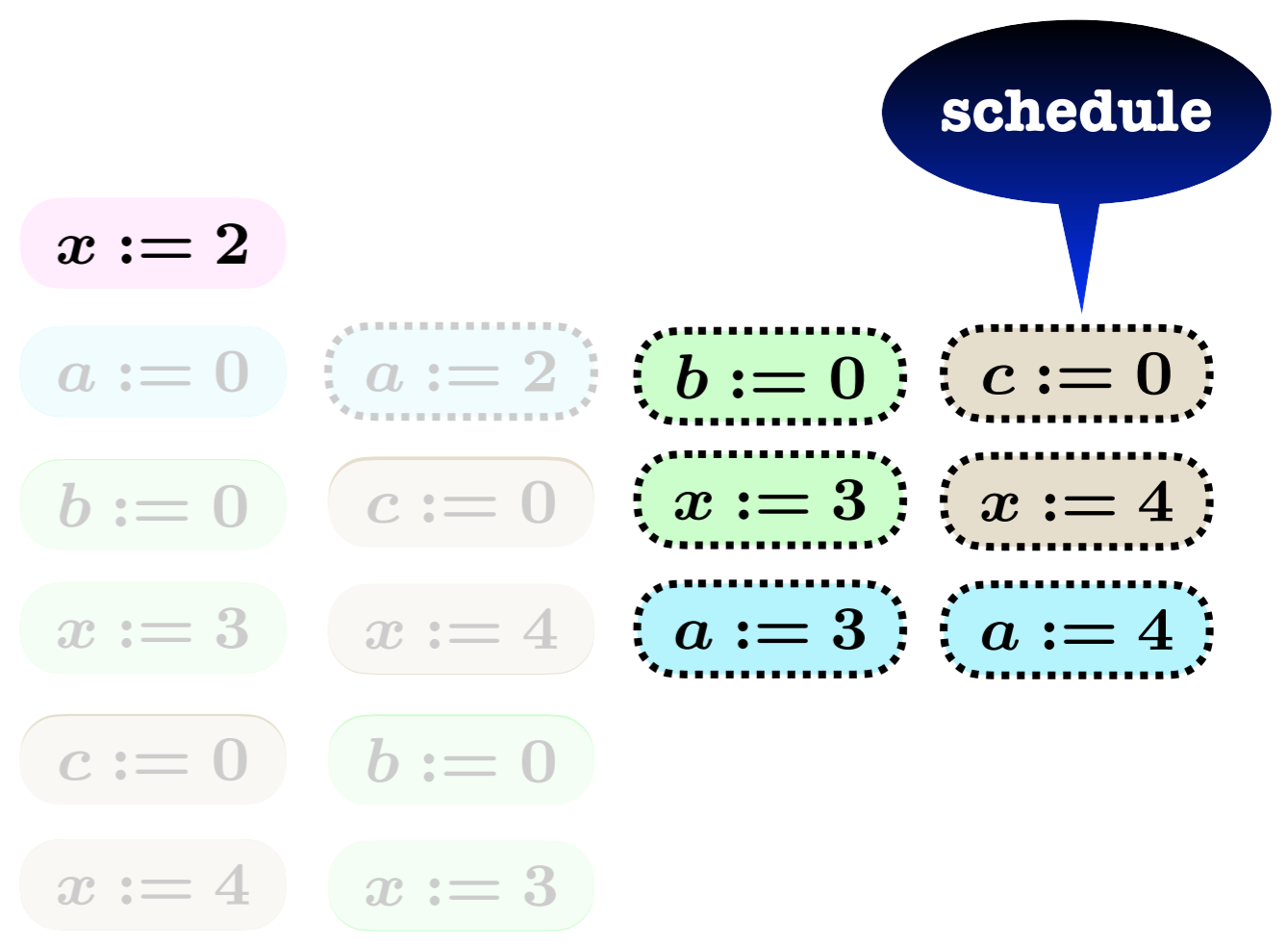
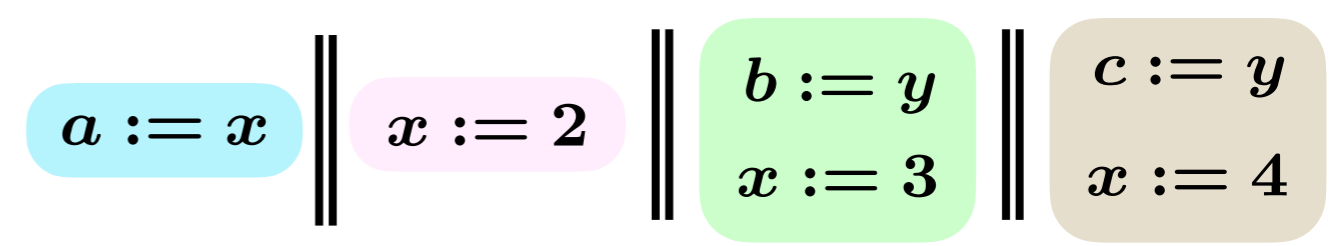
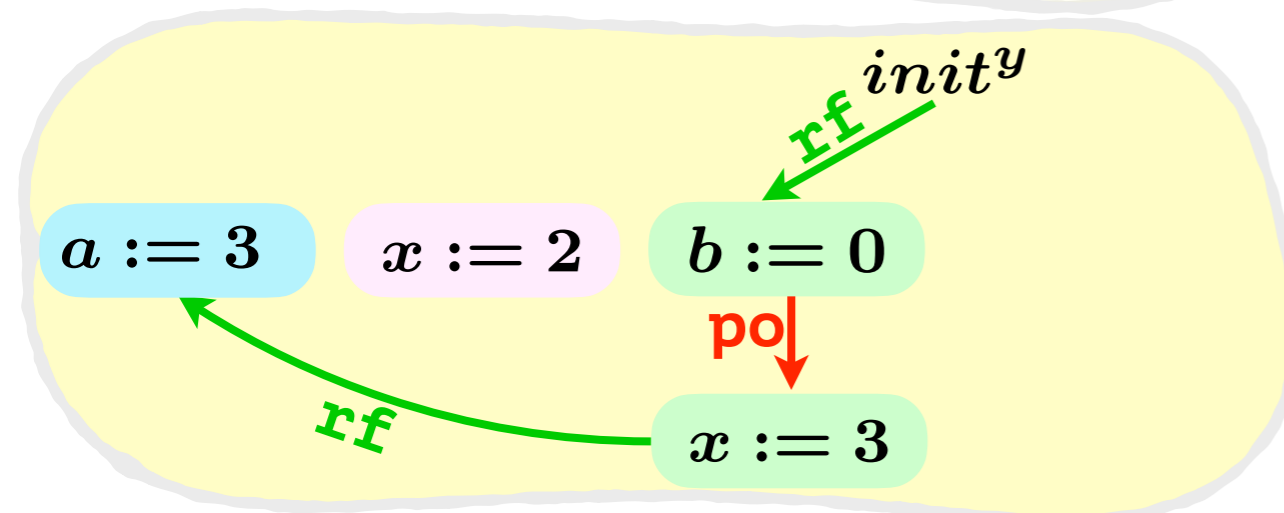
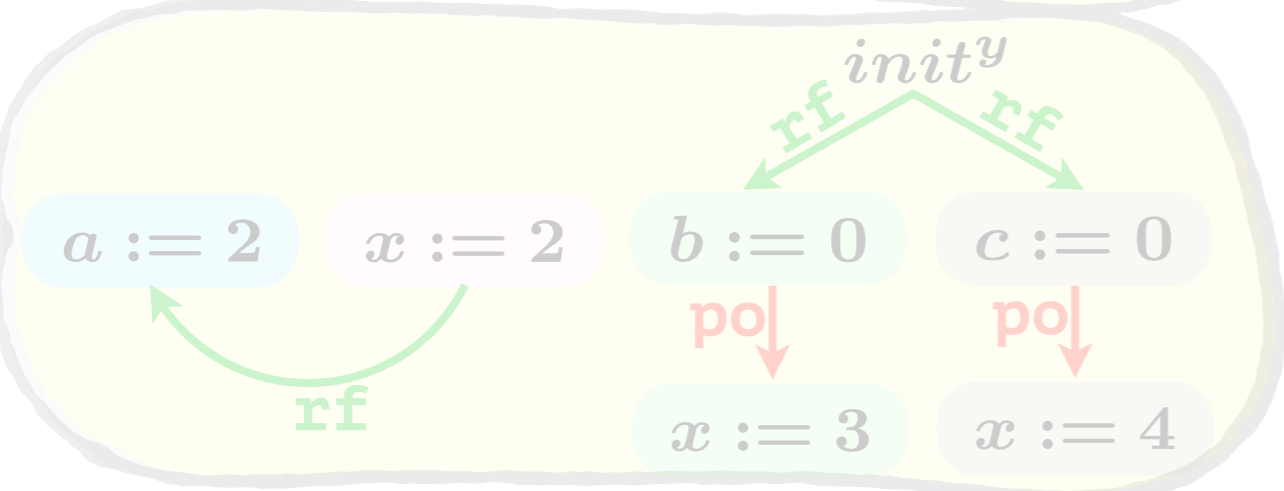
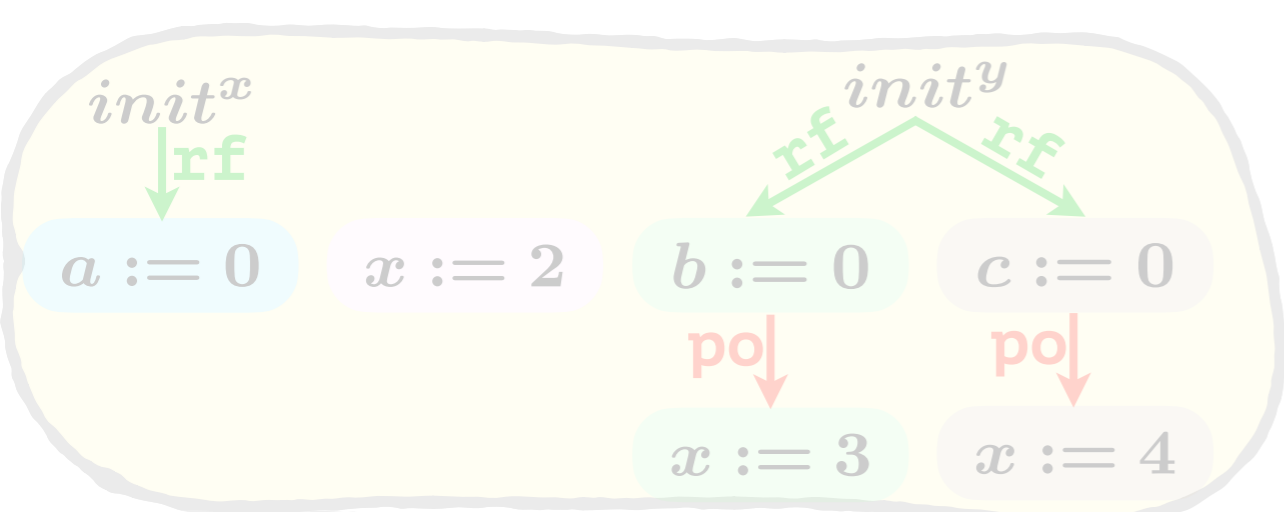
Postpone



Branch

&

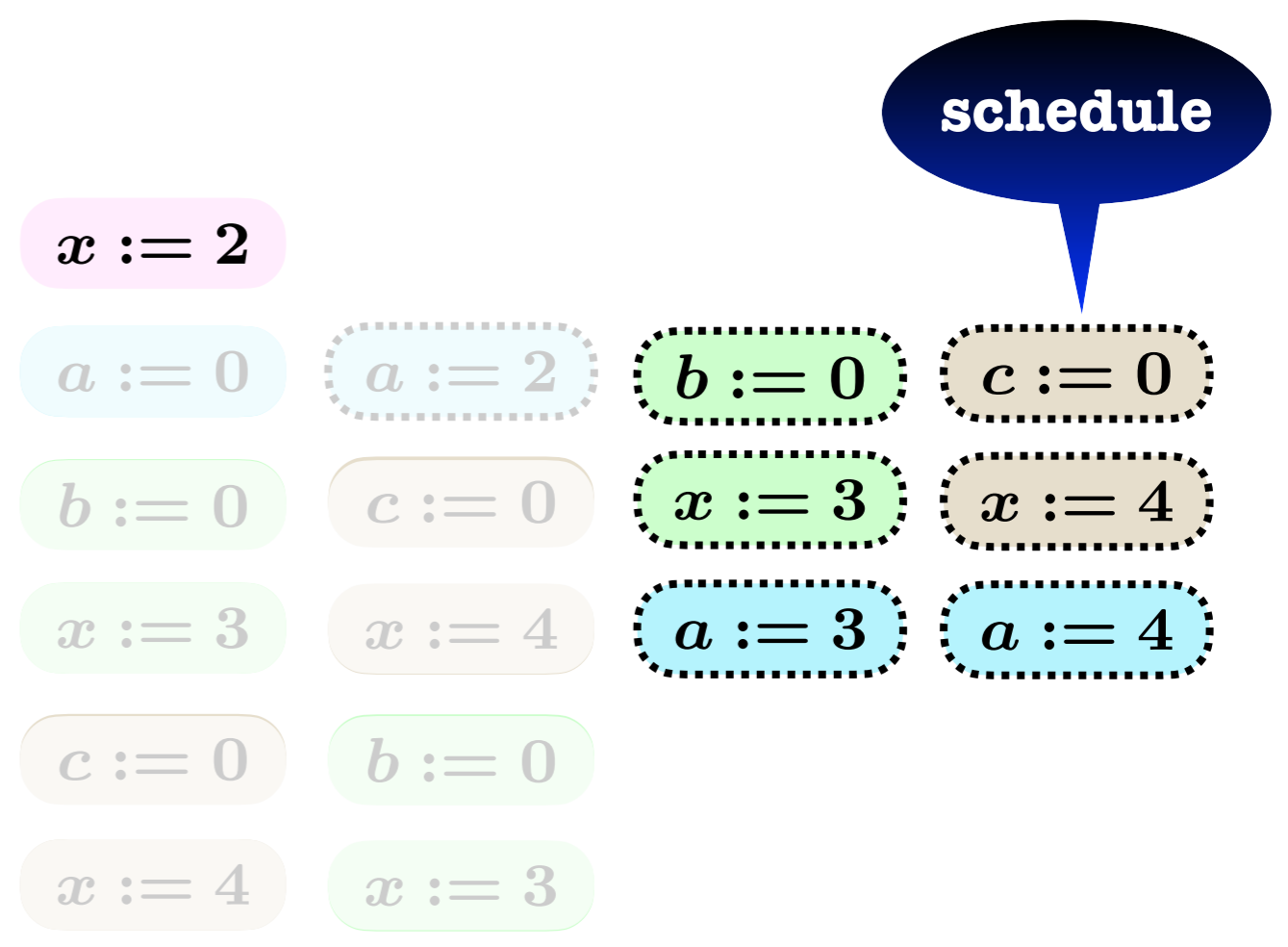
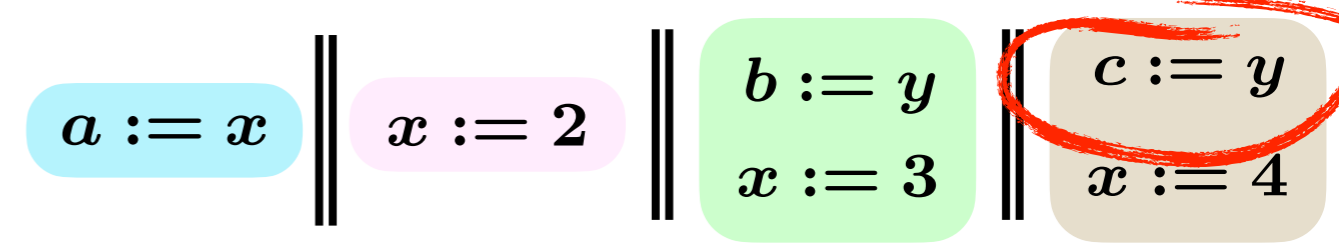
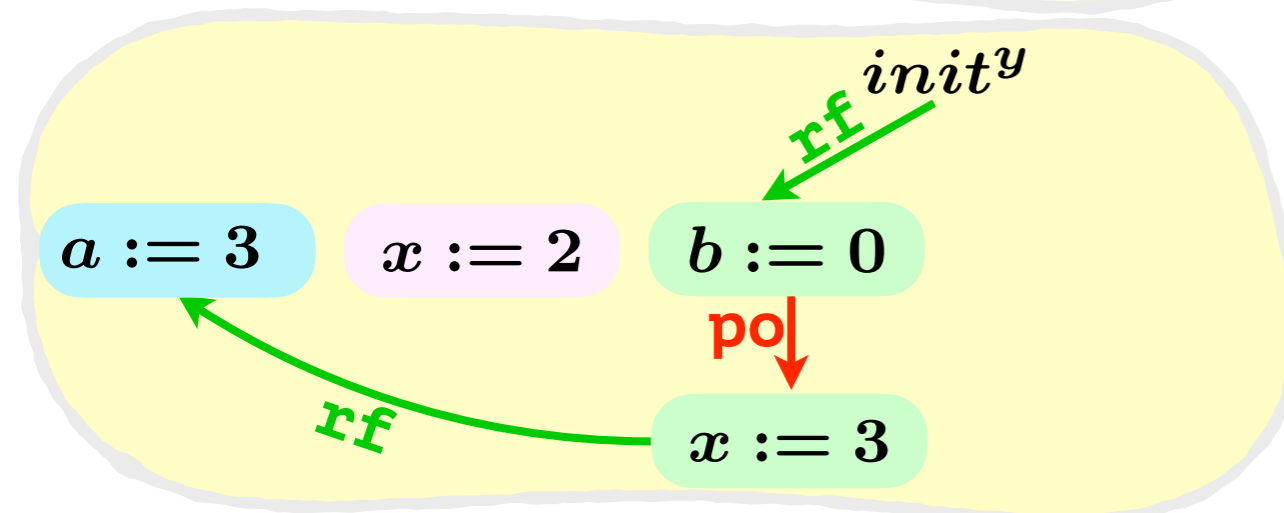
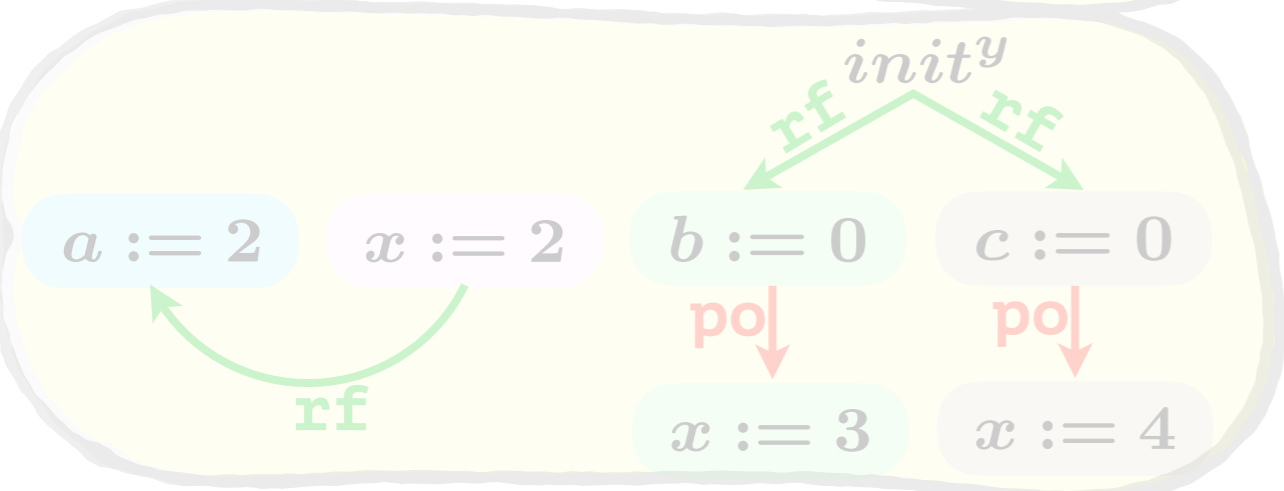
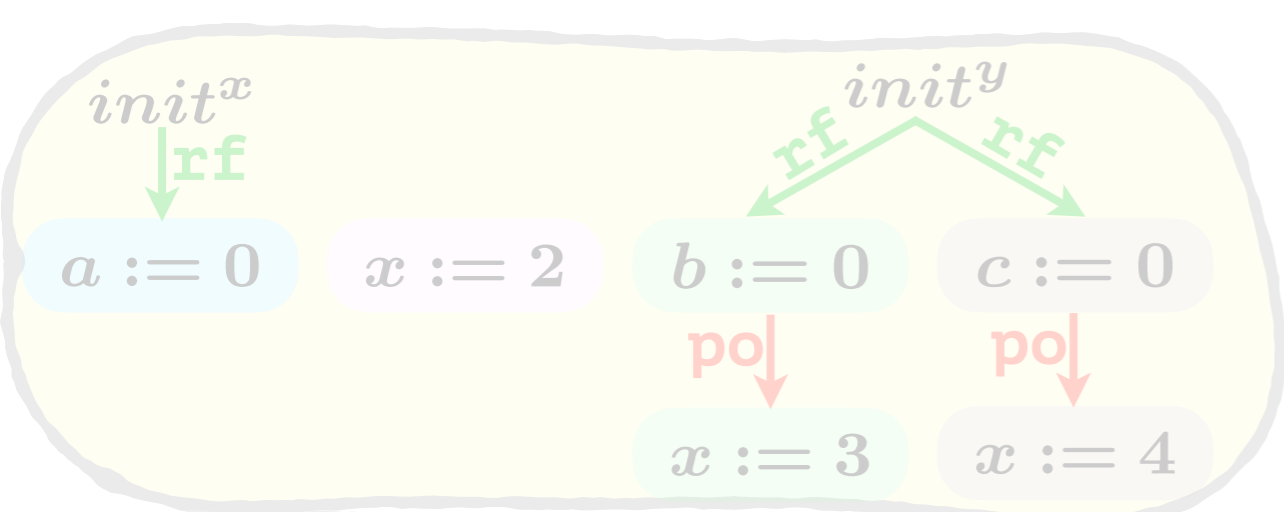
Postpone



Branch

&

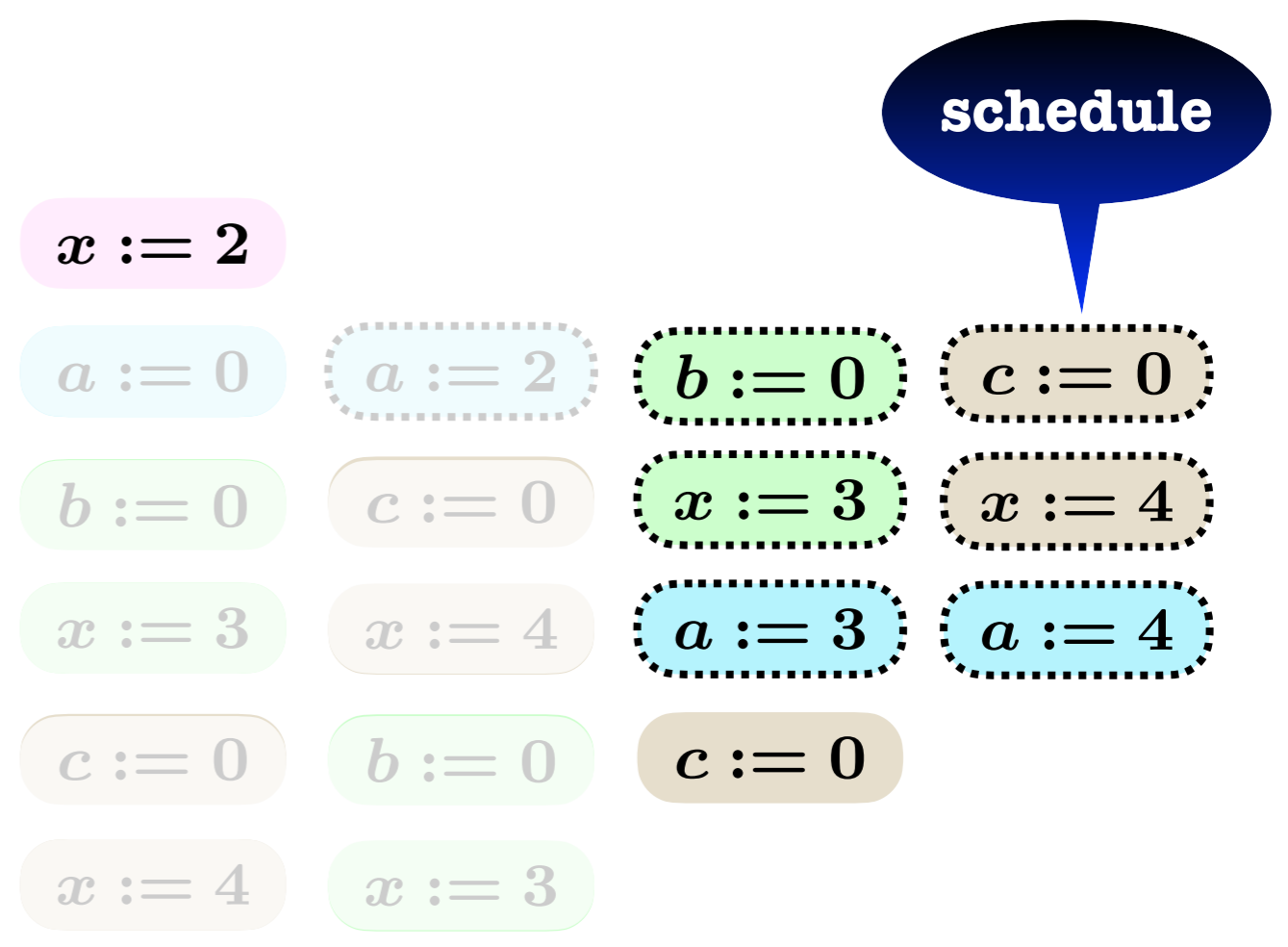
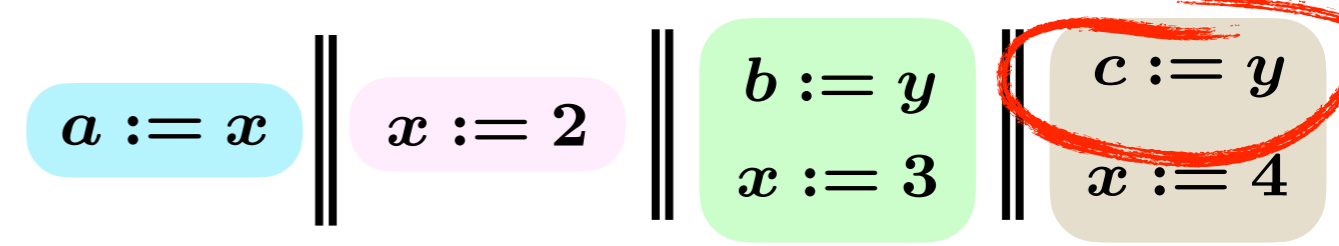
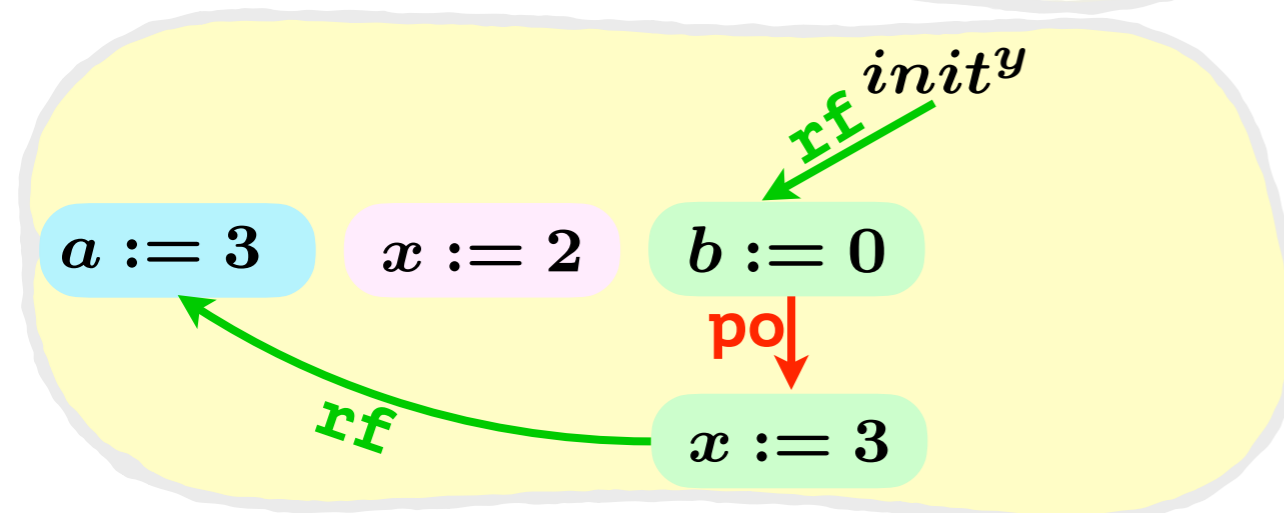
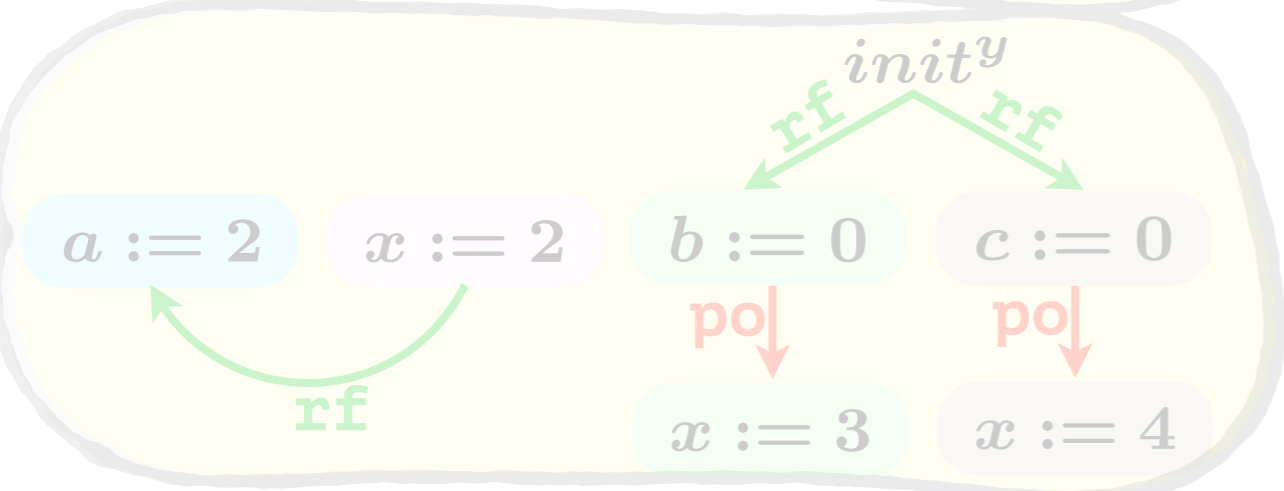
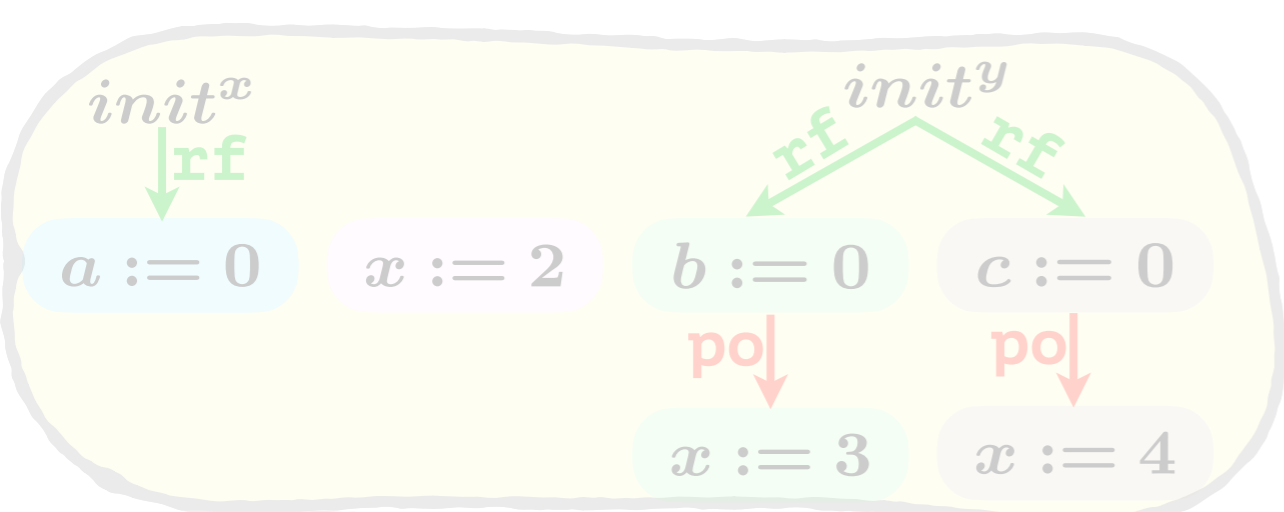
Postpone



Branch

&

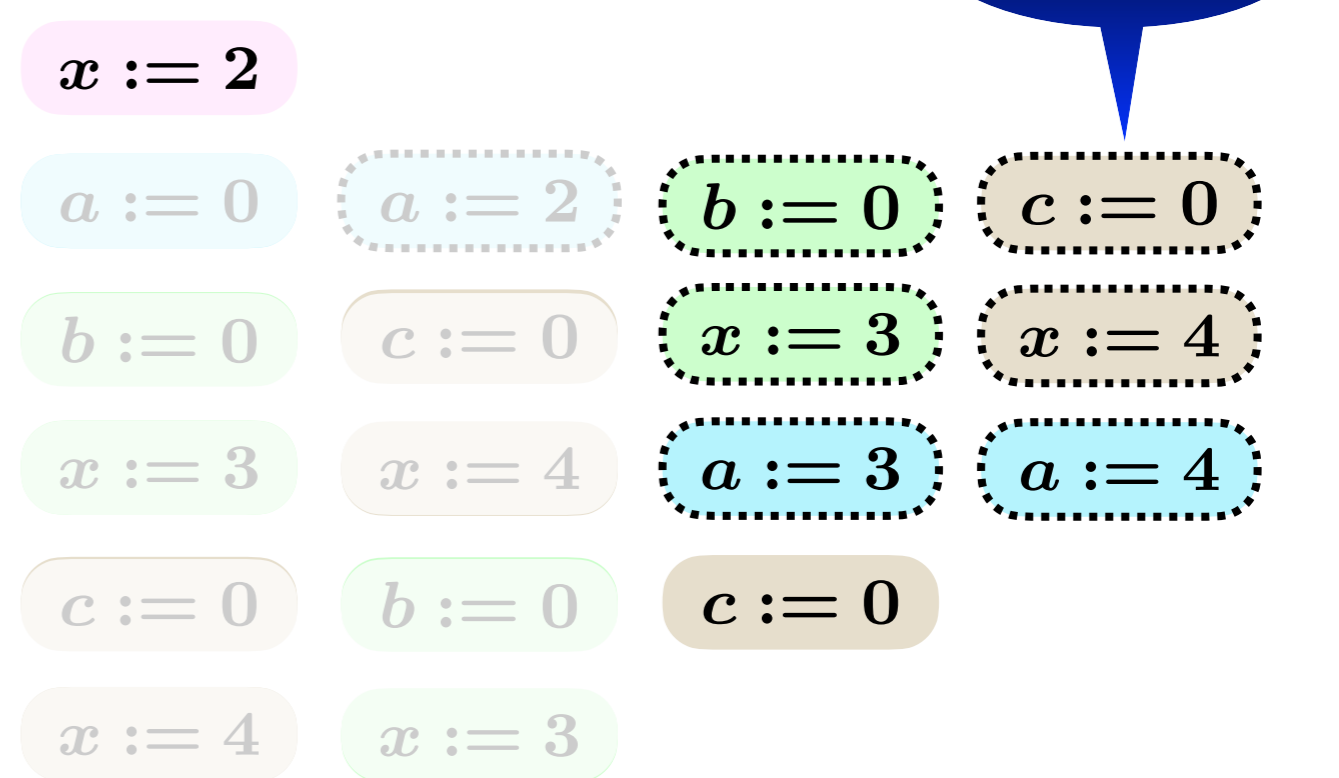
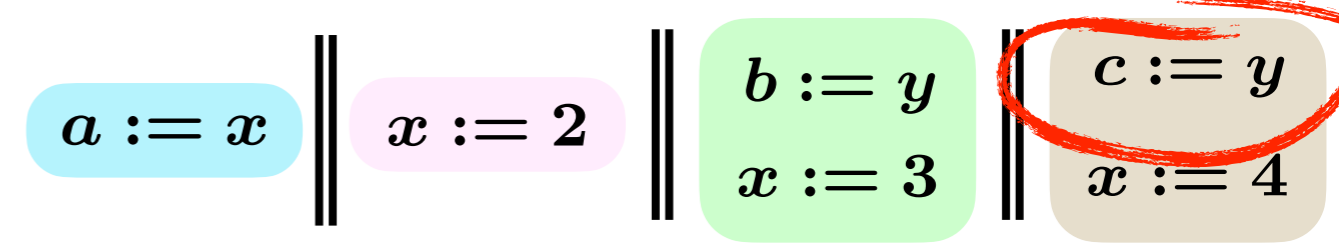
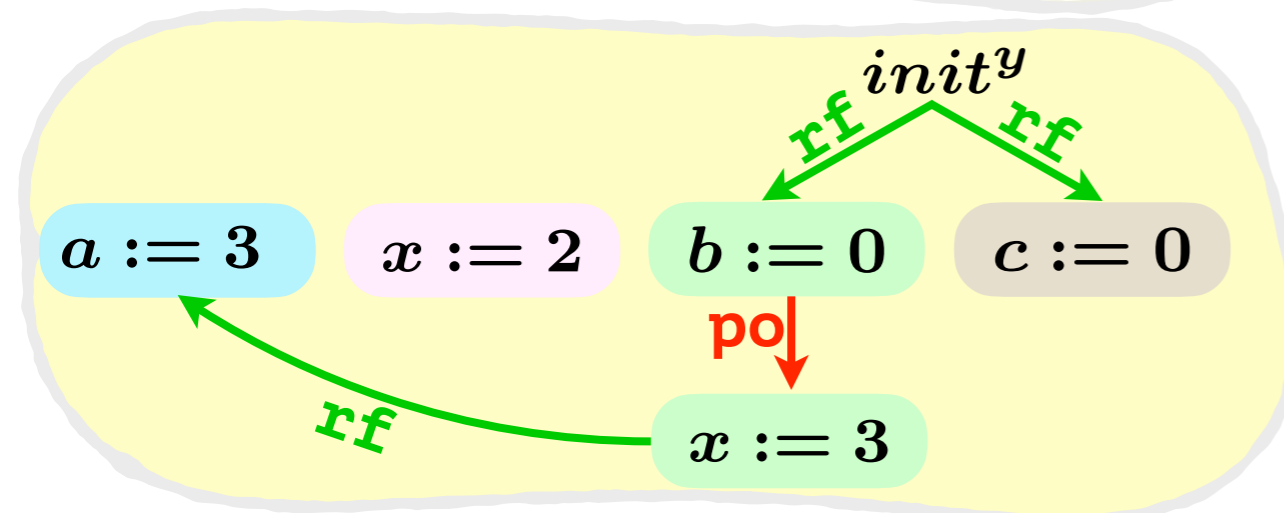
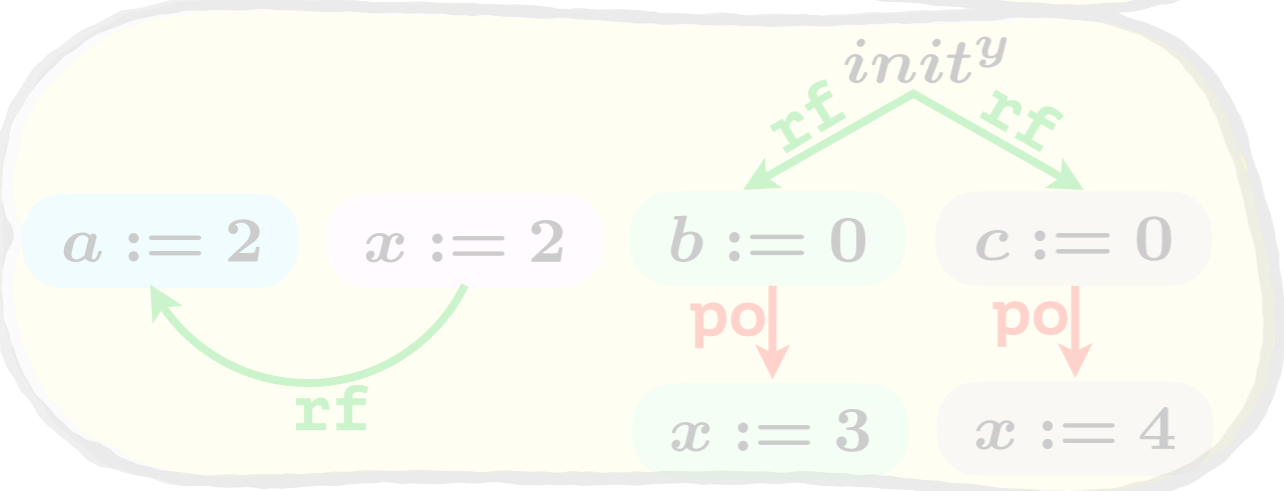
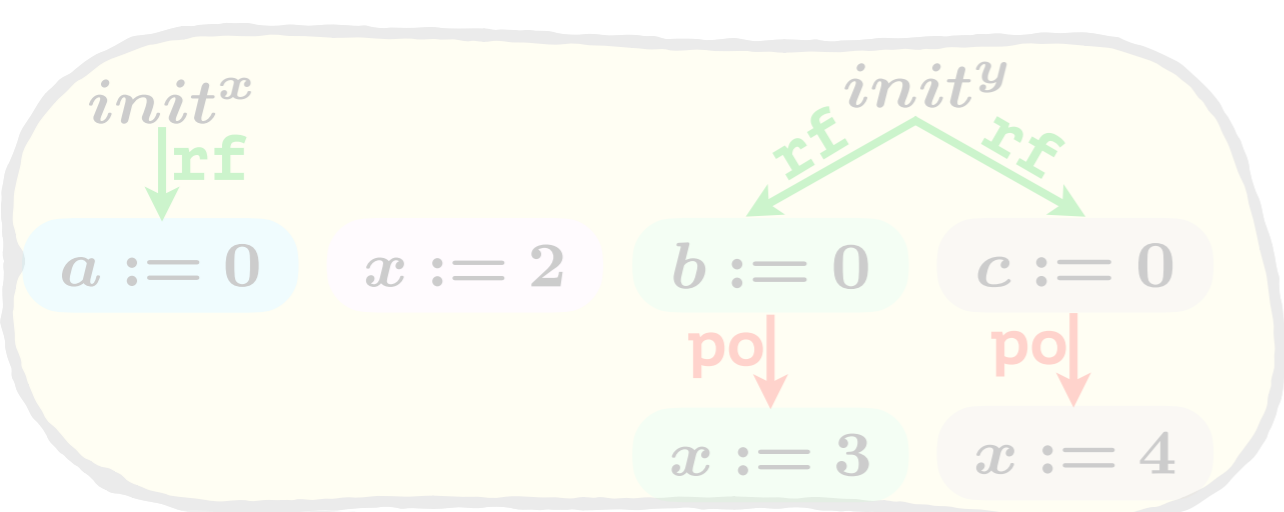
Postpone



Branch

&

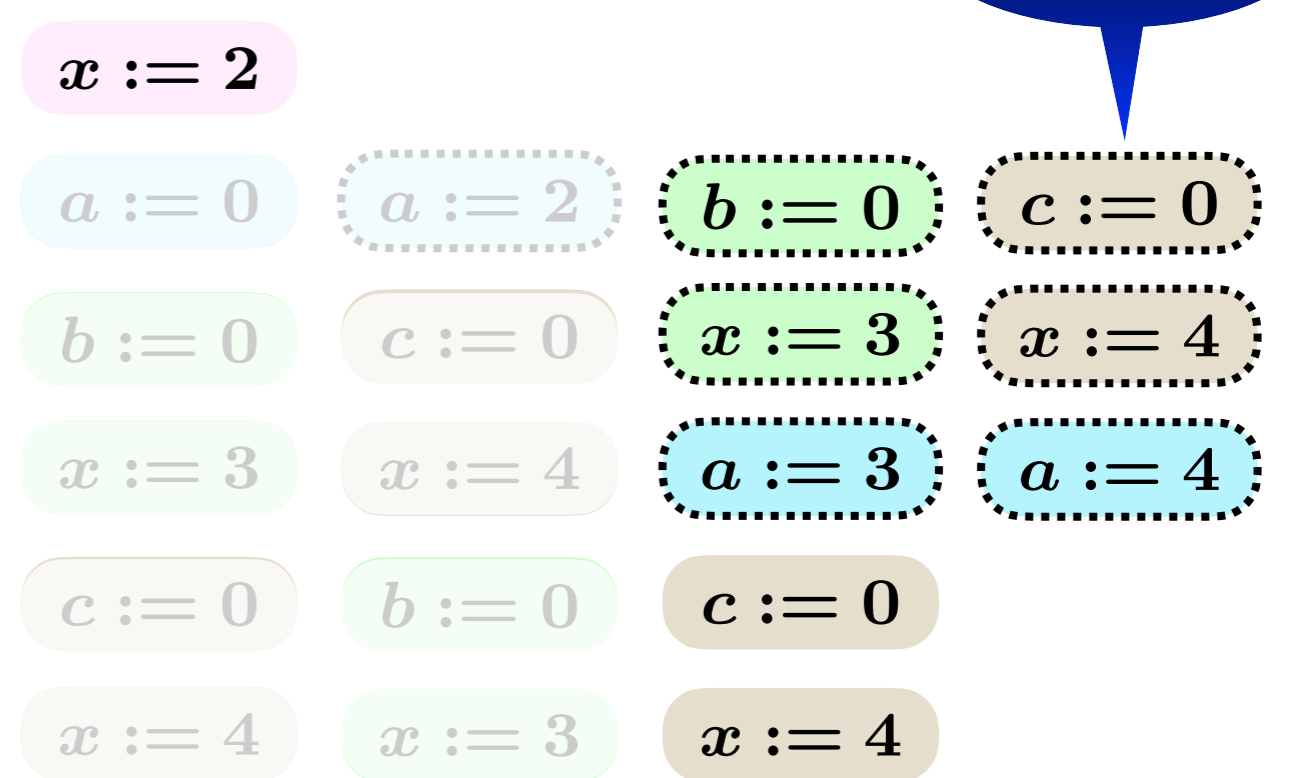
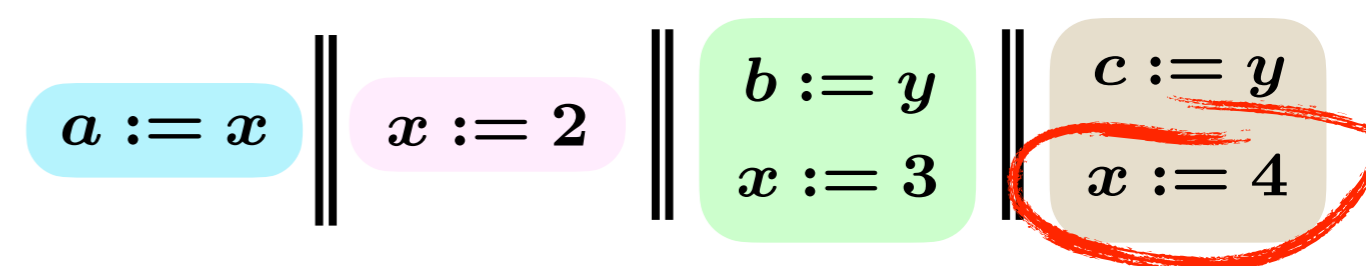
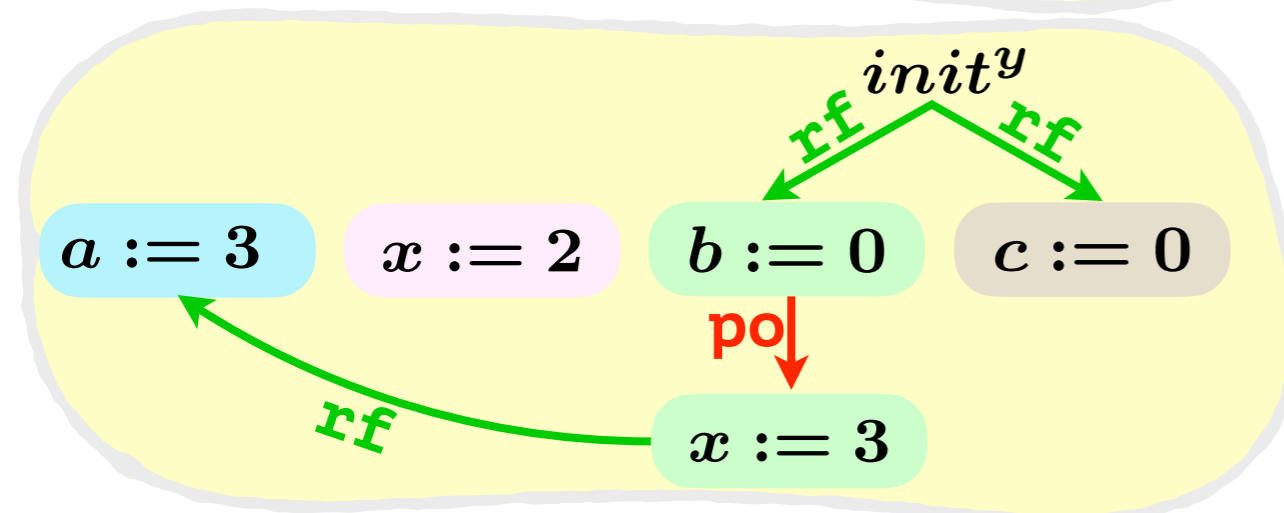
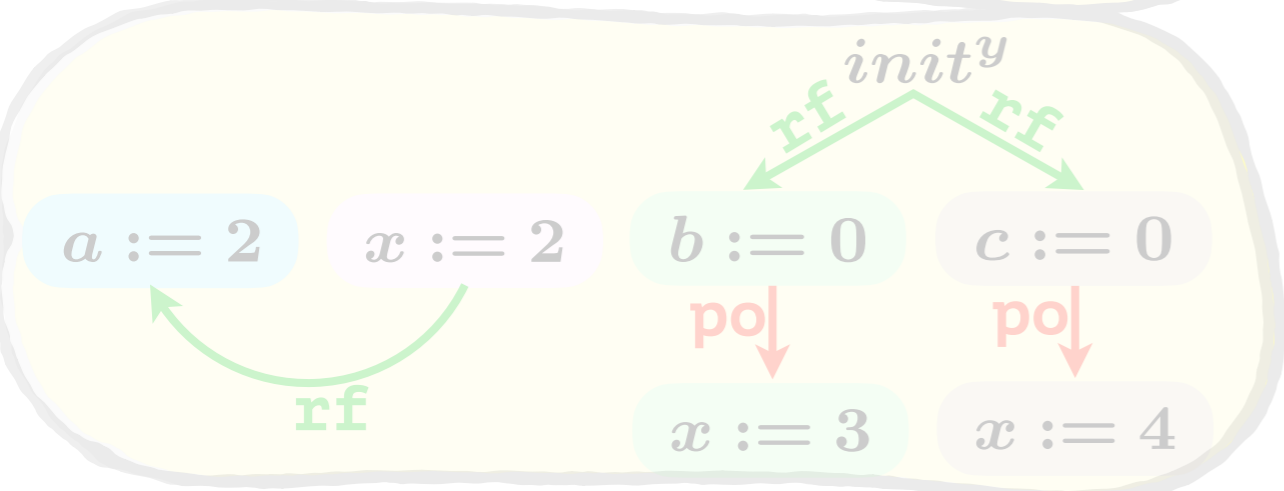
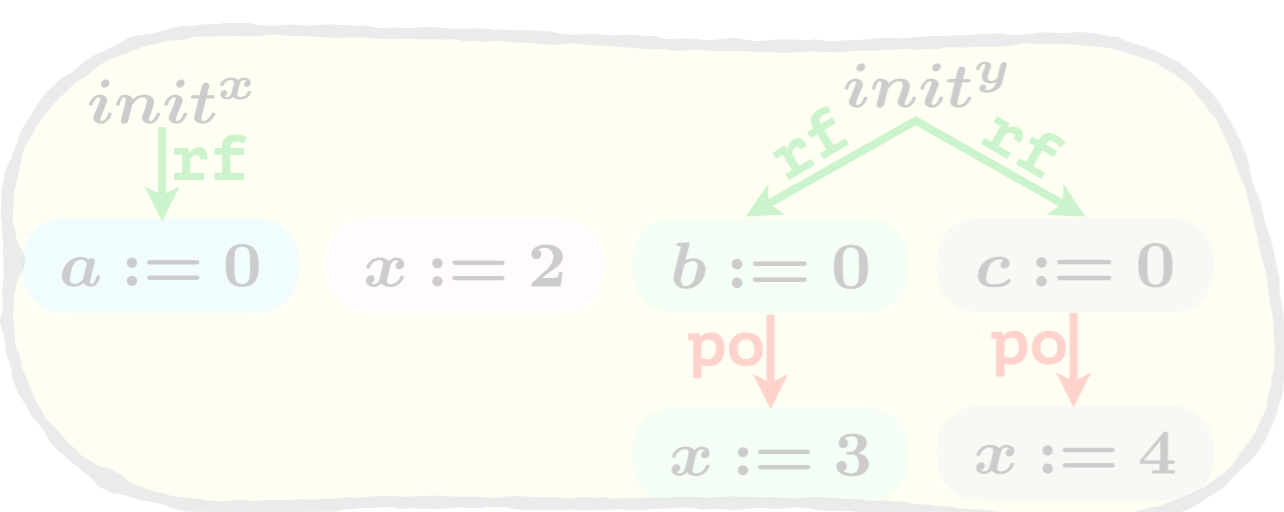
Postpone



Branch

&

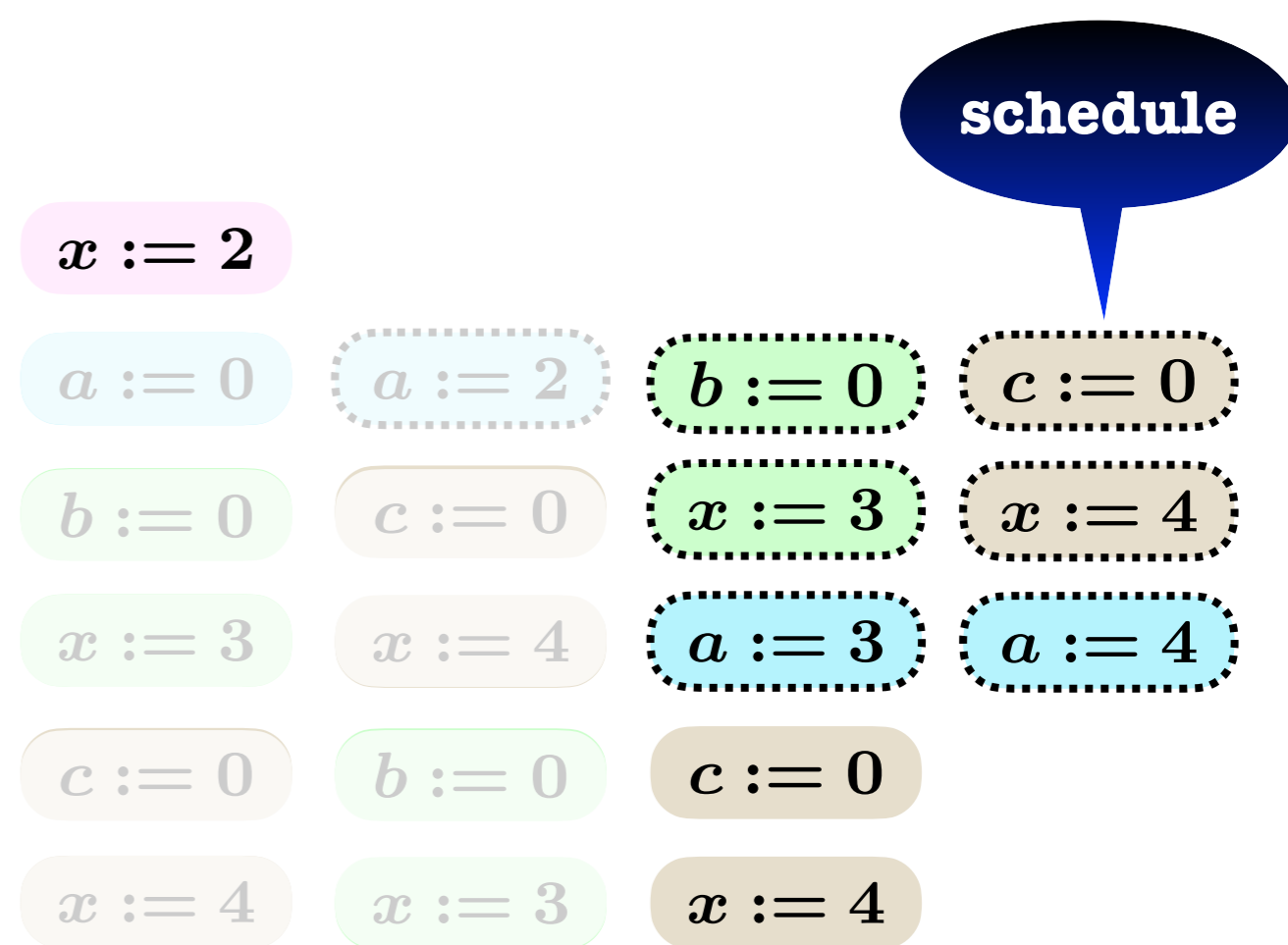
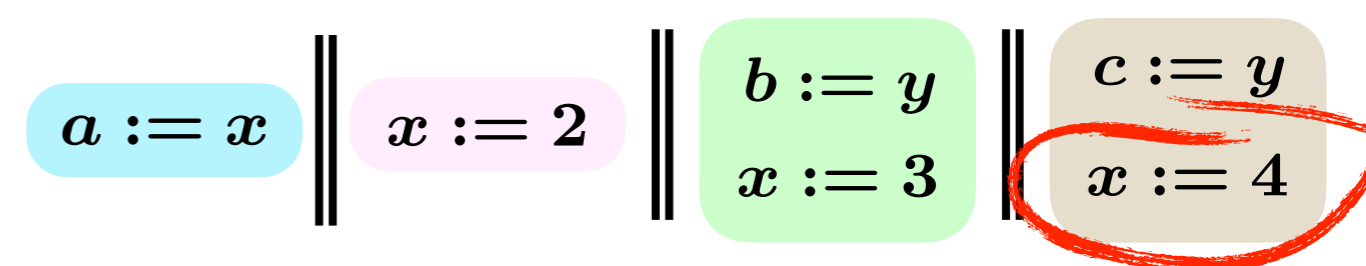
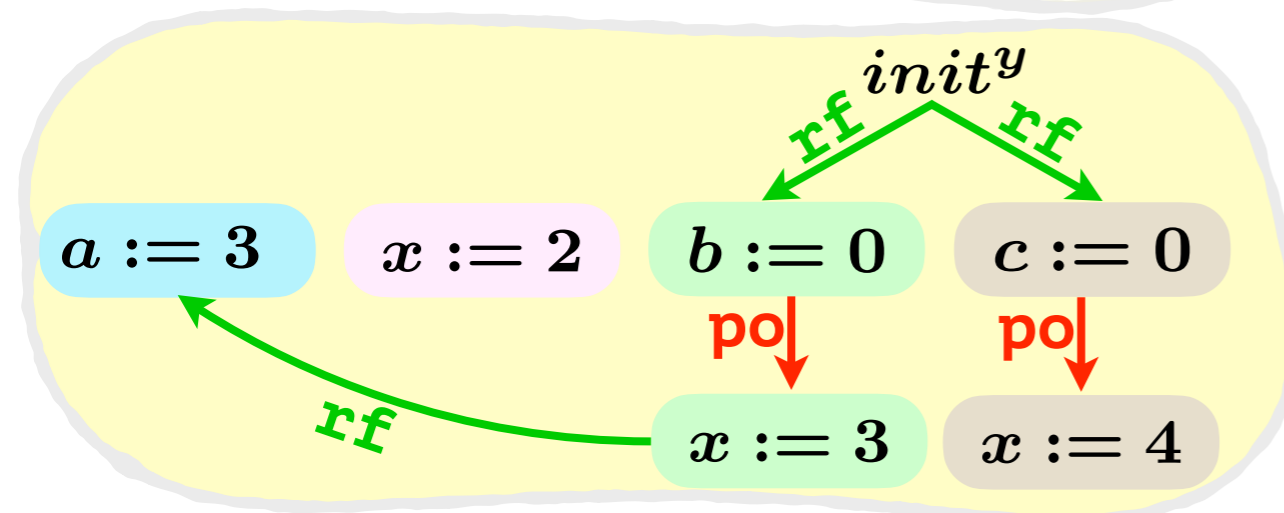
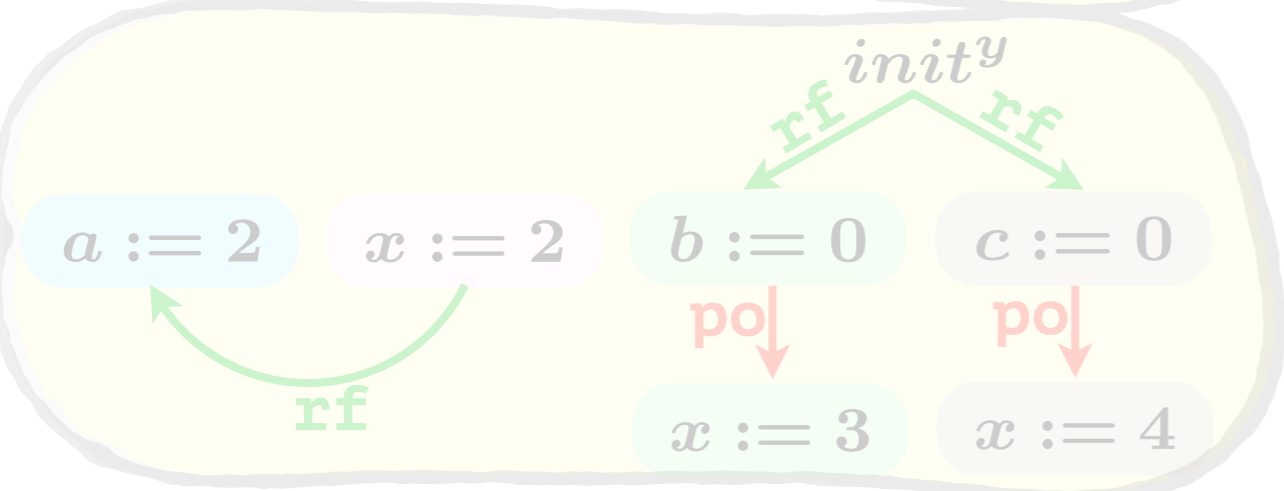
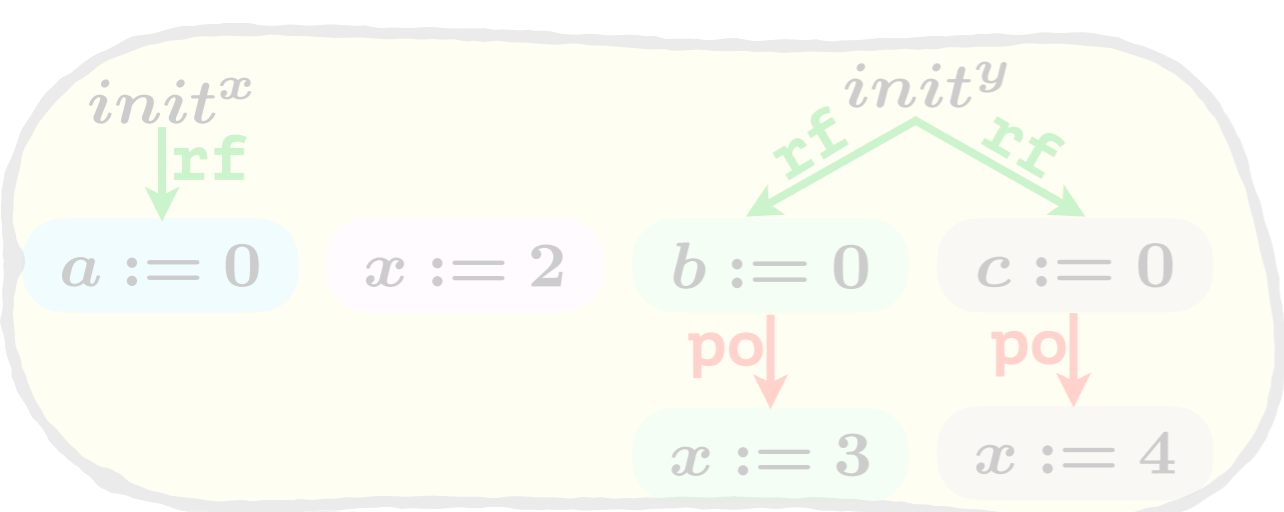
Postpone



Branch

&

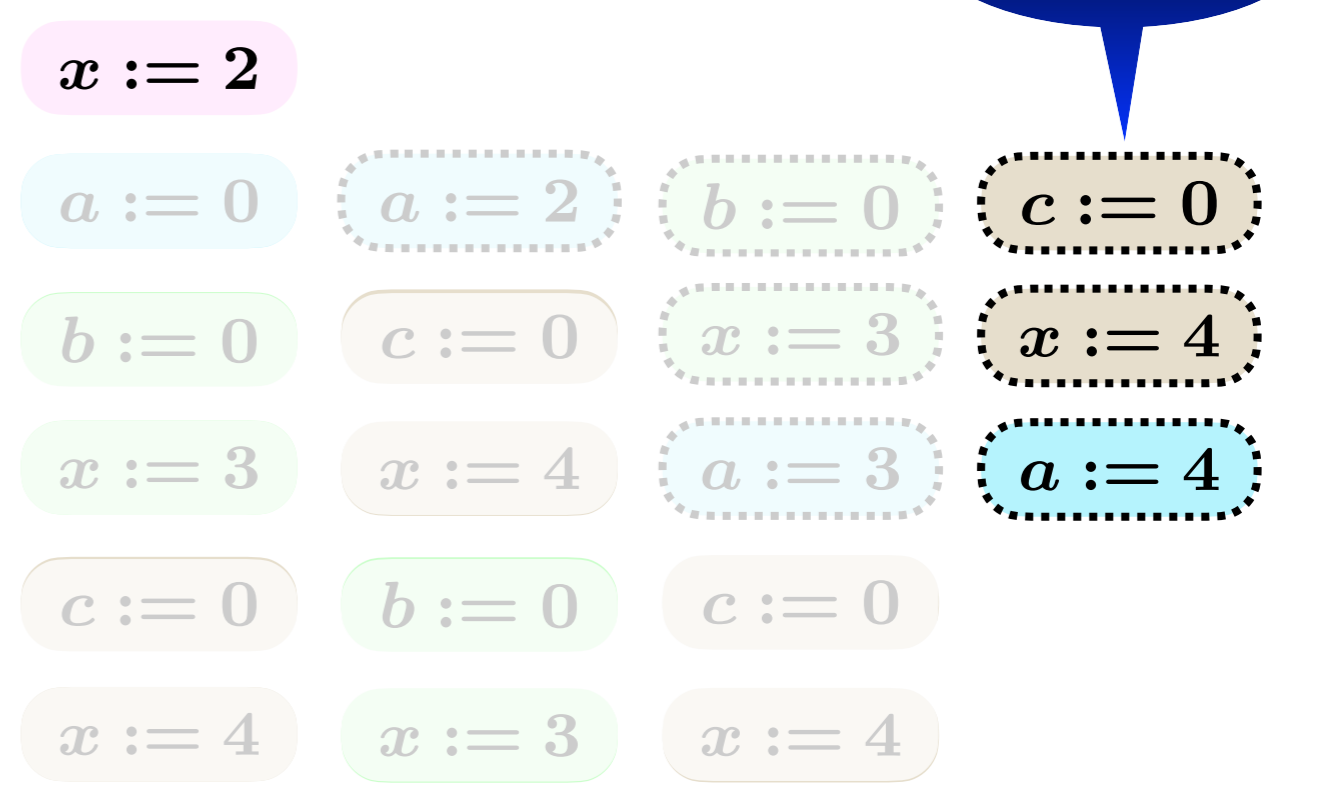
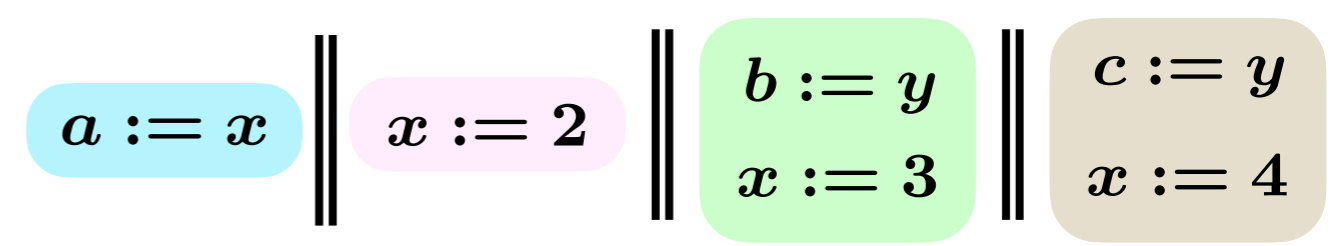
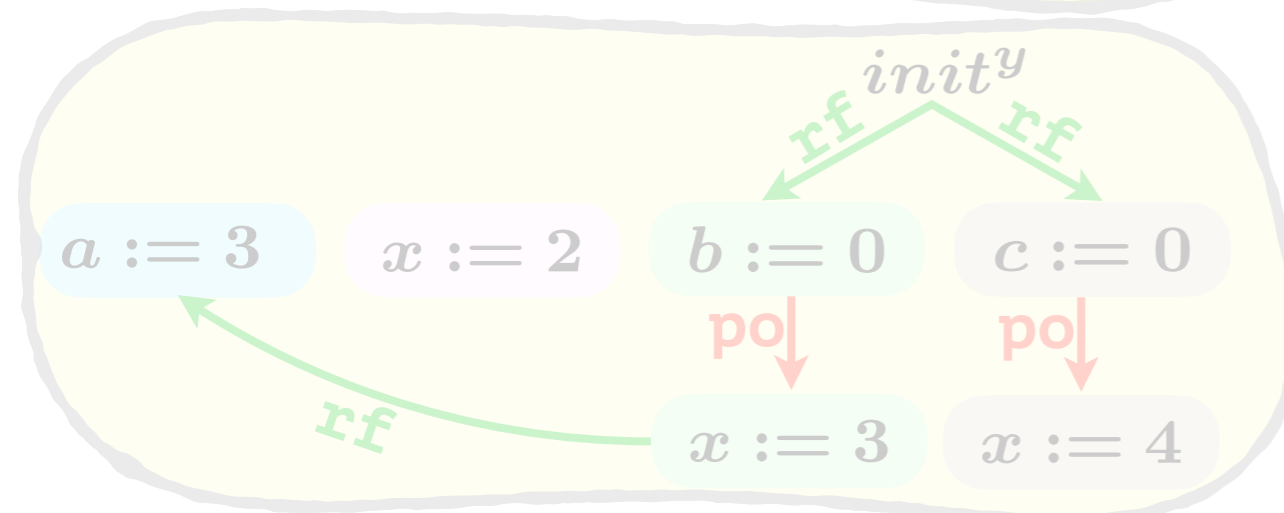
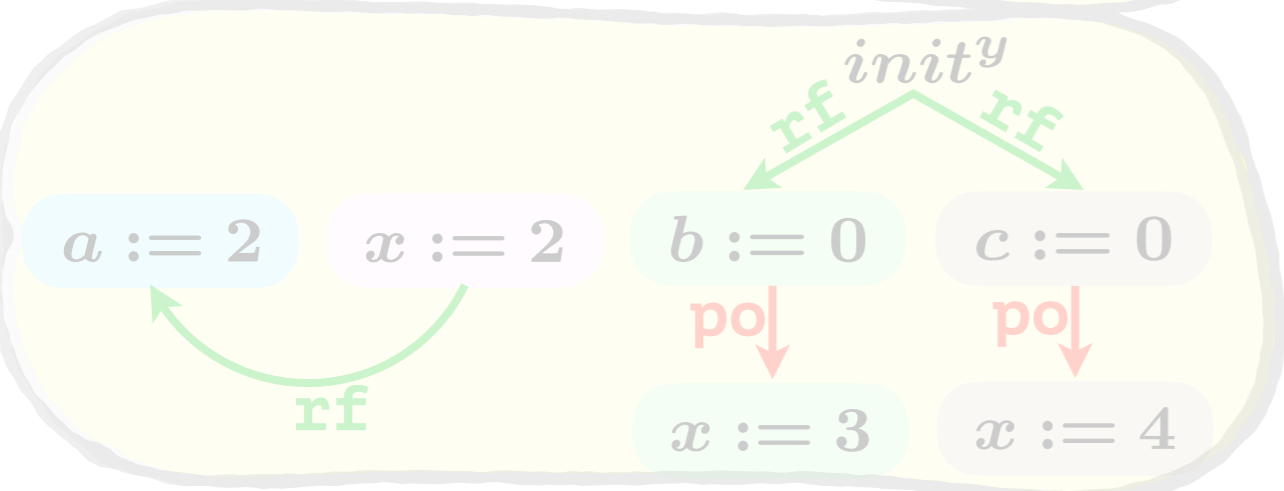
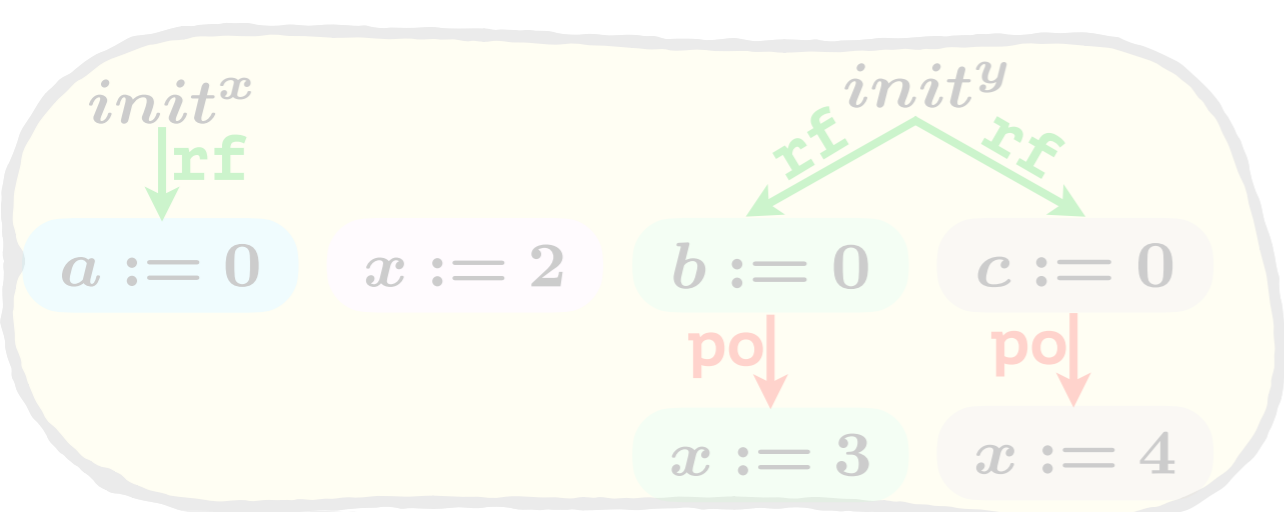
Postpone



Branch

&

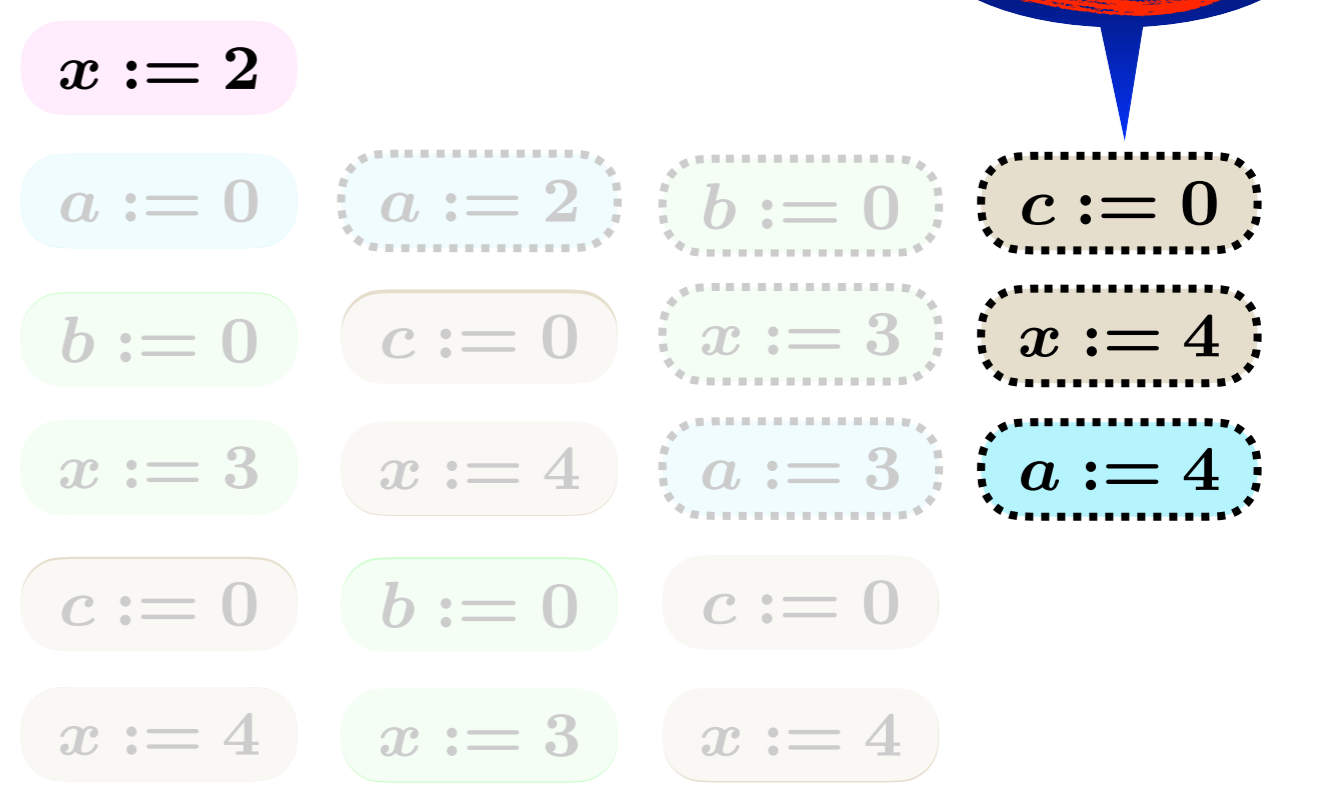
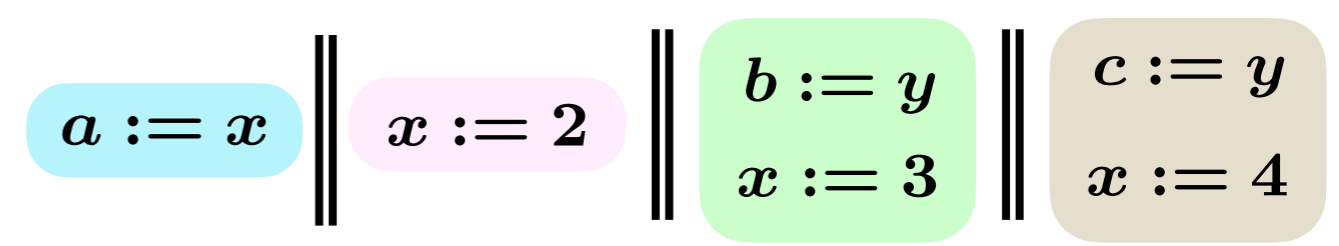
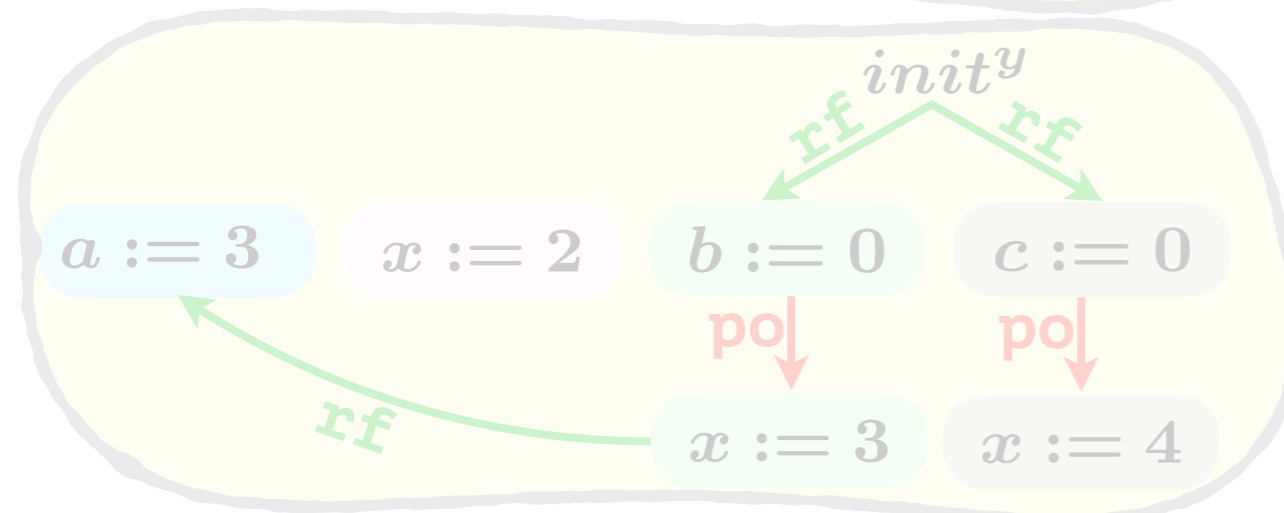
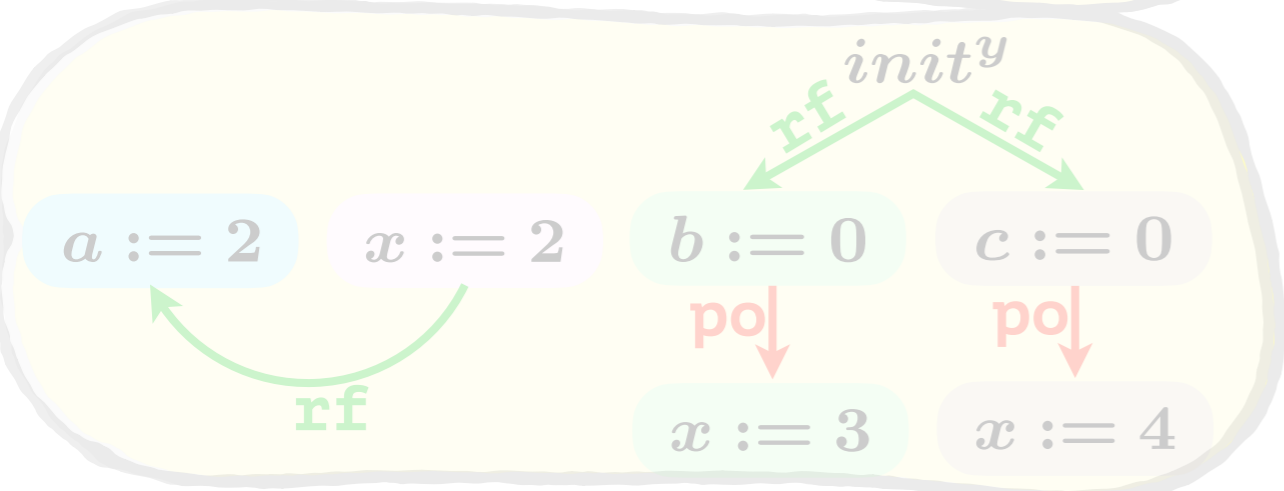
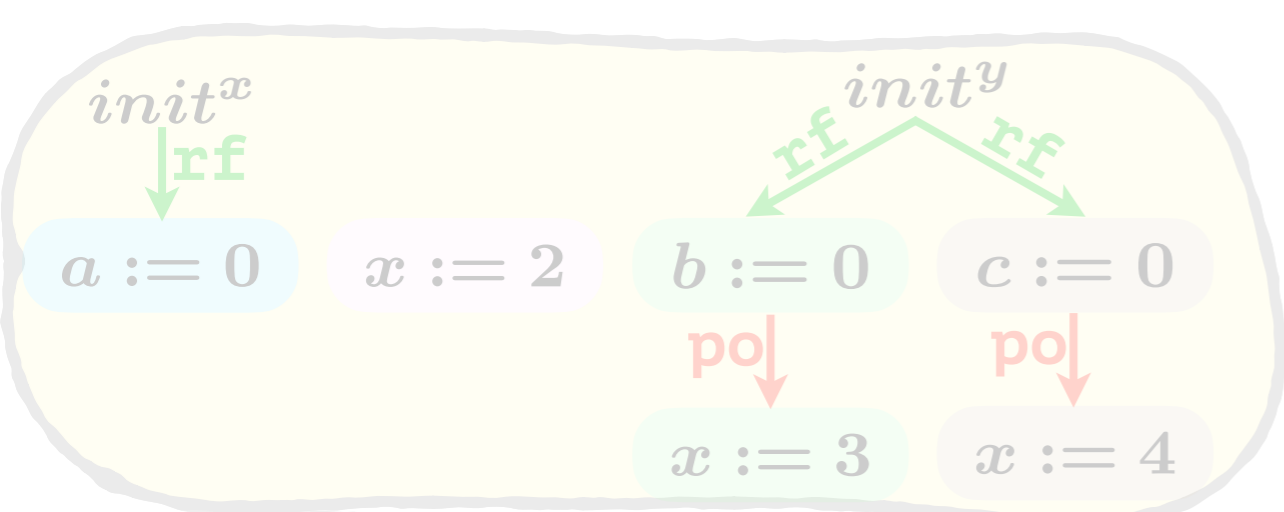
Postpone



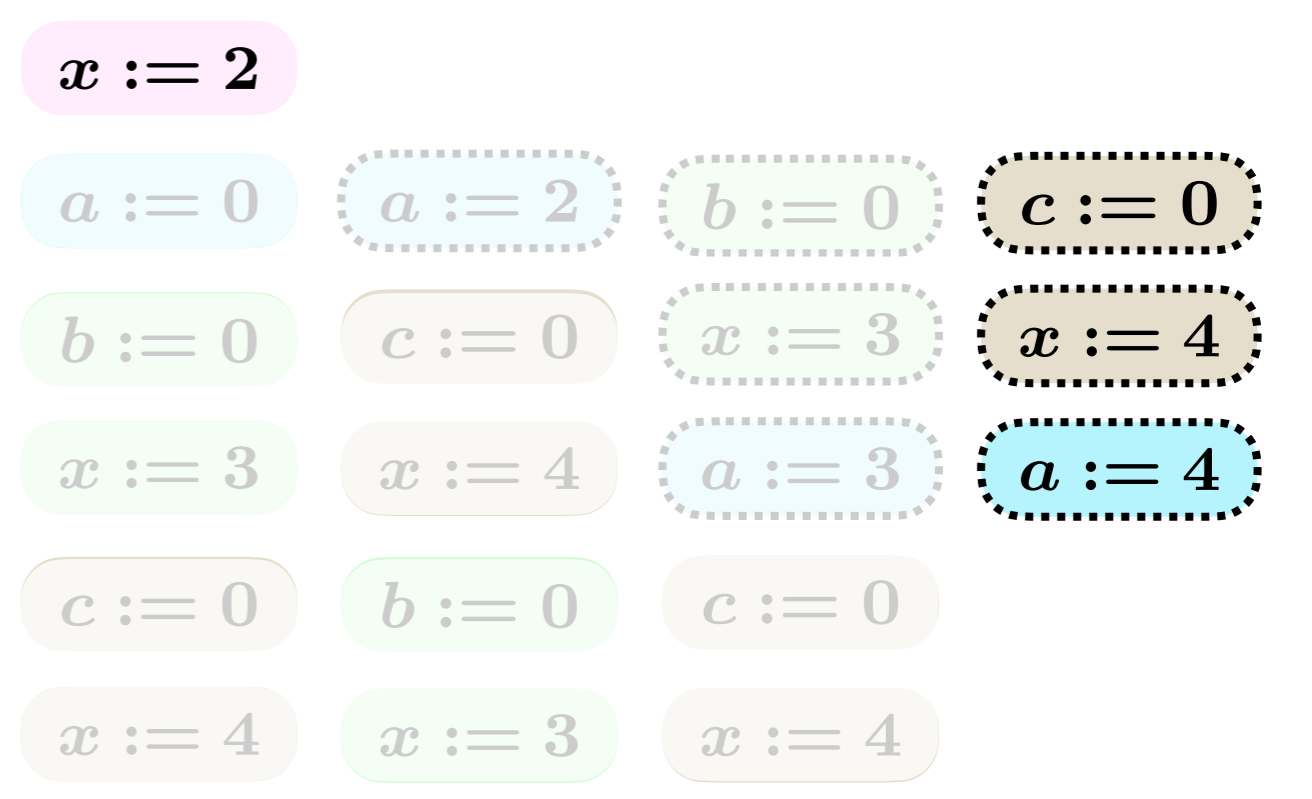
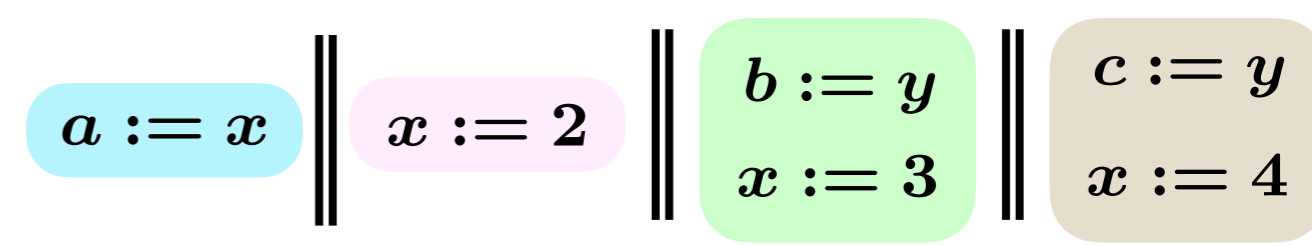
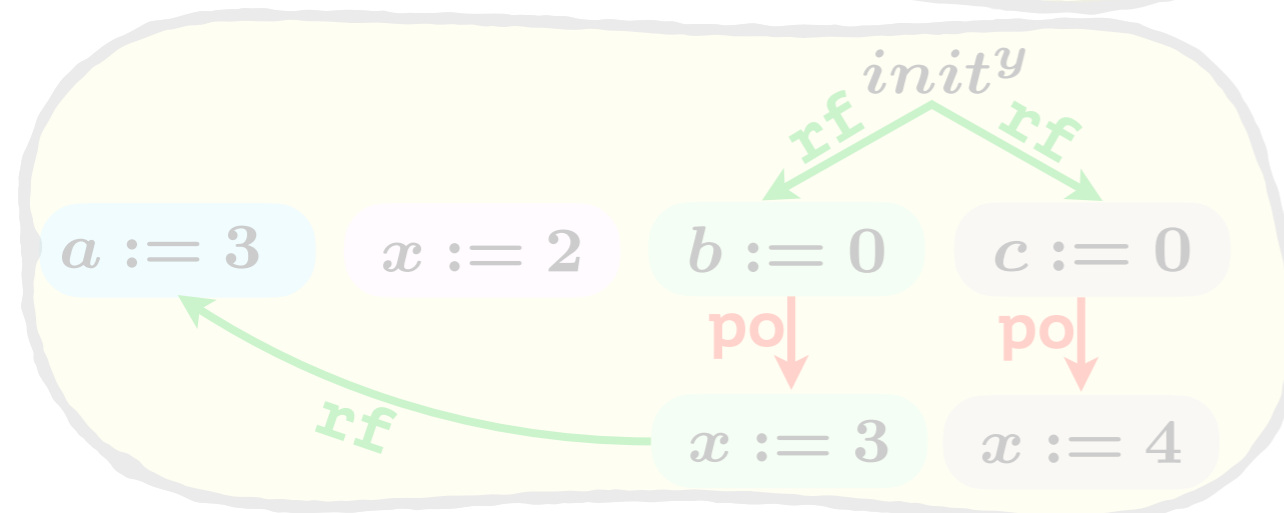
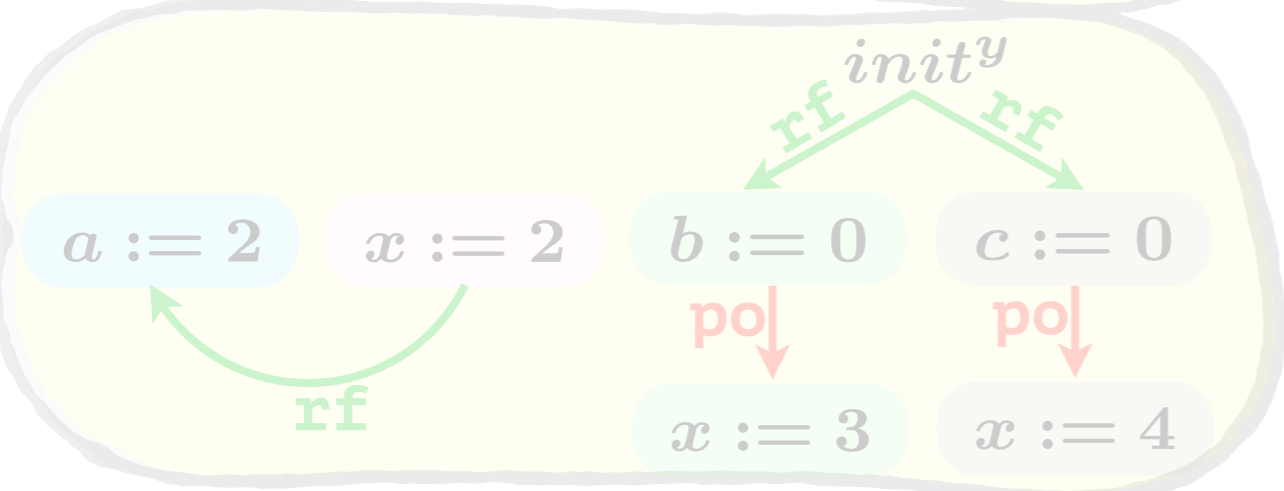
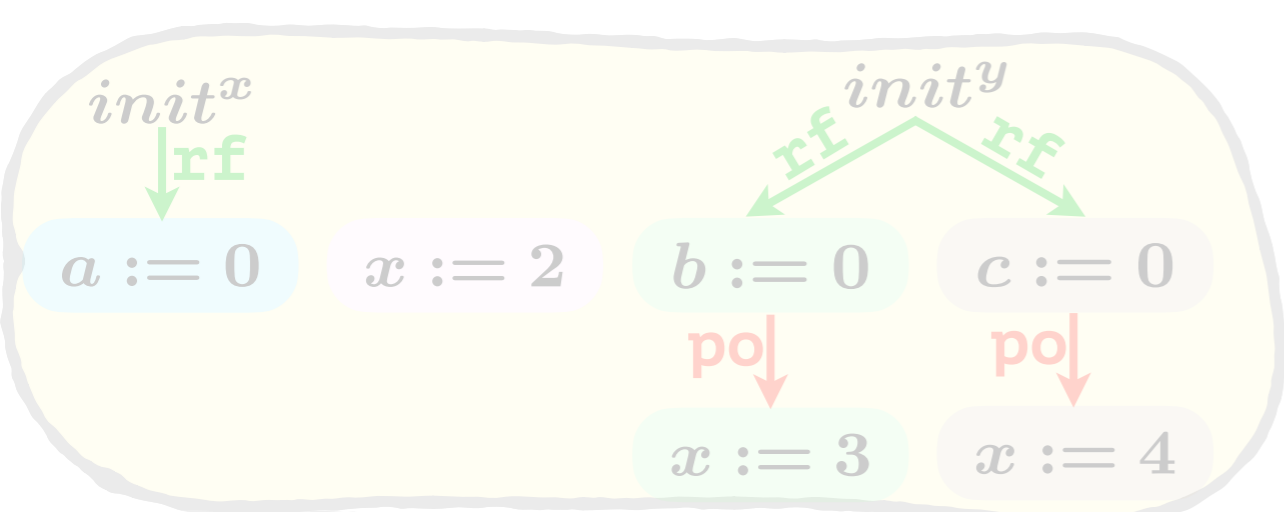
Branch

&

Postpone



Branch & **Postpone**



Branch

&

Postpone

