Deciding Reachability under Persistent x86-TSO

Parosh Aziz Abdulla, Faouzi Atig, Ahmed Bouajjani,
K Narayan Kumar, Prakash Saivasan

Deciding Reachability under Persistent x86-TSO

(Program Verification:
from Sequential Consistency
to Weak Consistency)

Parosh Aziz Abdulla, Faouzi Atig, Ahmed Bouajjani,
K Narayan Kumar, Prakash Saivasan

Sequential Consistency (SC)

+ simple & intuitive
- expensive

Sequential Consistency (SC)

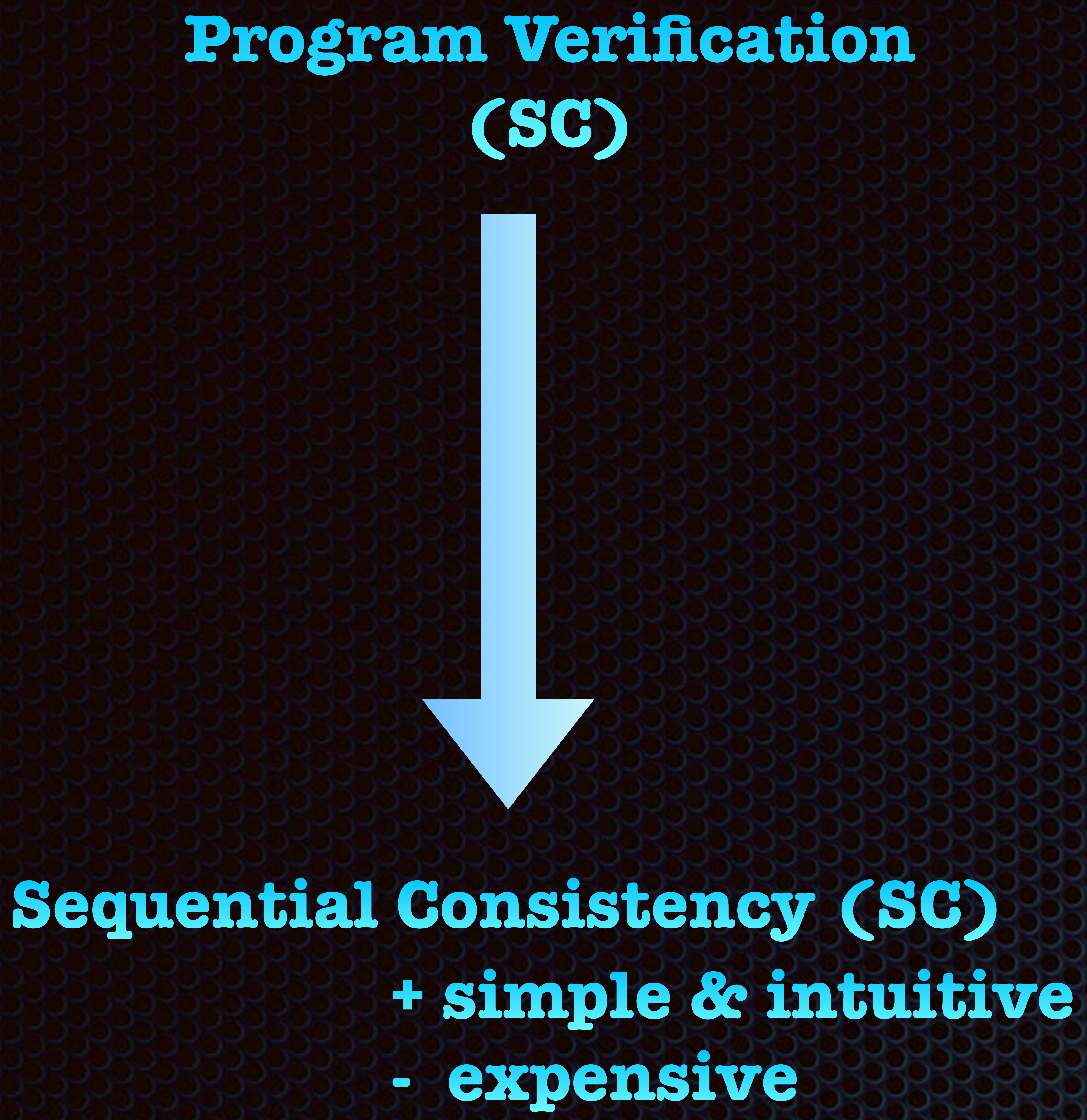
\+ simple & intuitive

\- expensive

Weak Consistency

\+ efficient, realistic

\- complicated



Program Verification
(SC)

Sequential Consistency (SC)

- \+ simple & intuitive
- \- expensive

Weak Consistency

- \+ efficient, realistic
- \- complicated

Program Verification
(SC)

Sequential Consistency (SC)

+ simple & intuitive
- expensive

Program Verification
(weak consistency)

Weak Consistency

+ efficient, realistic
- complicated

Program Verification (SC)

Program Verification (weak consistency)

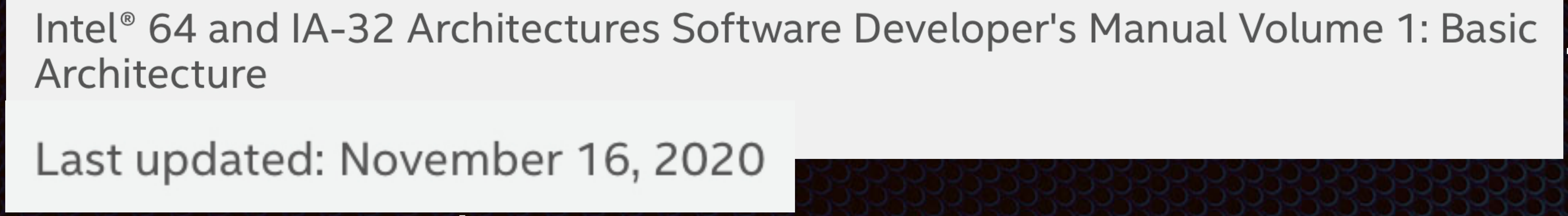
Sequential Consistency (SC)

- \+ simple & intuitive
- \- expensive

Weak Consistency

- \+ efficient, realistic
- \- complicated

main stream Intel architecture: persistent x86-TSO



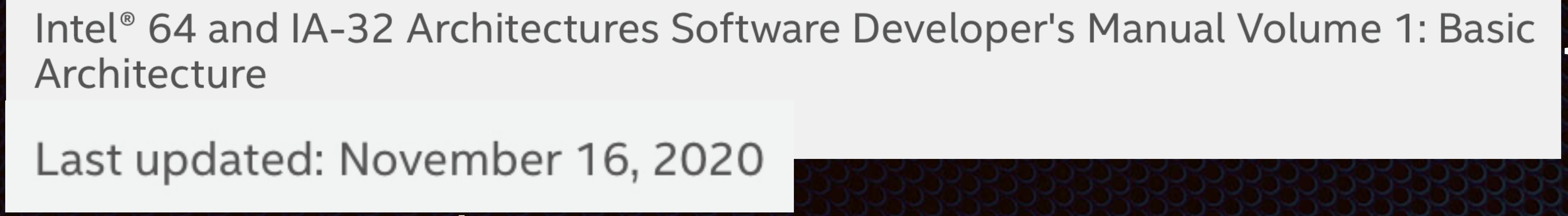
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

this work

Persistent TSO

main stream Intel architecture: persistent x86-TSO



Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

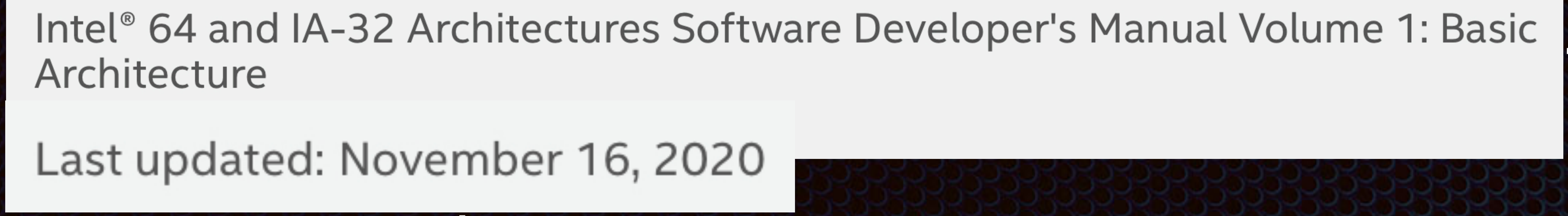
Last updated: November 16, 2020

this work

Persistent TSO = Total Store Order (TSO)

classical weak memory model

main stream Intel architecture: persistent x86-TSO



Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

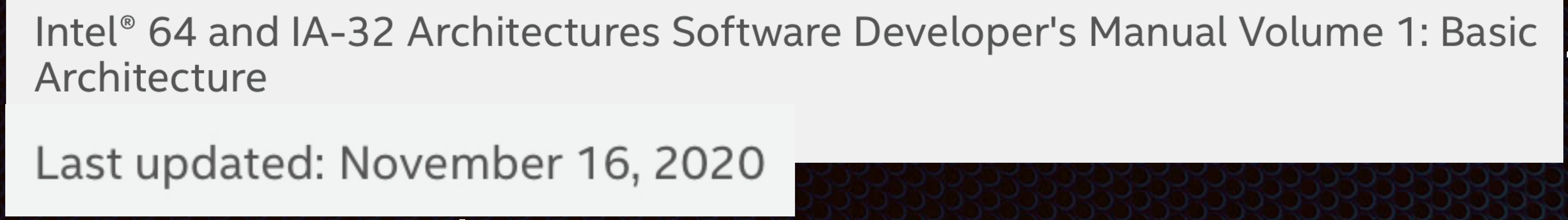
this work

Persistent TSO = **Total Store Order (TSO)** + **Persistency**

classical weak memory model

NVRAM: data persist over crashes

main stream Intel architecture: persistent x86-TSO



Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

this work

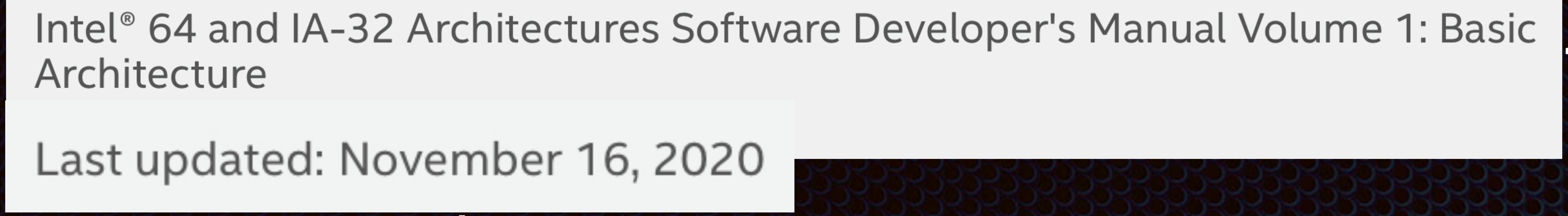
Persistent TSO = **Total Store Order (TSO)** + **Persistency**

classical weak memory model

NVRAM: data persist over crashes

Program Verification (SC) → **Program Verification (weak consistency)**

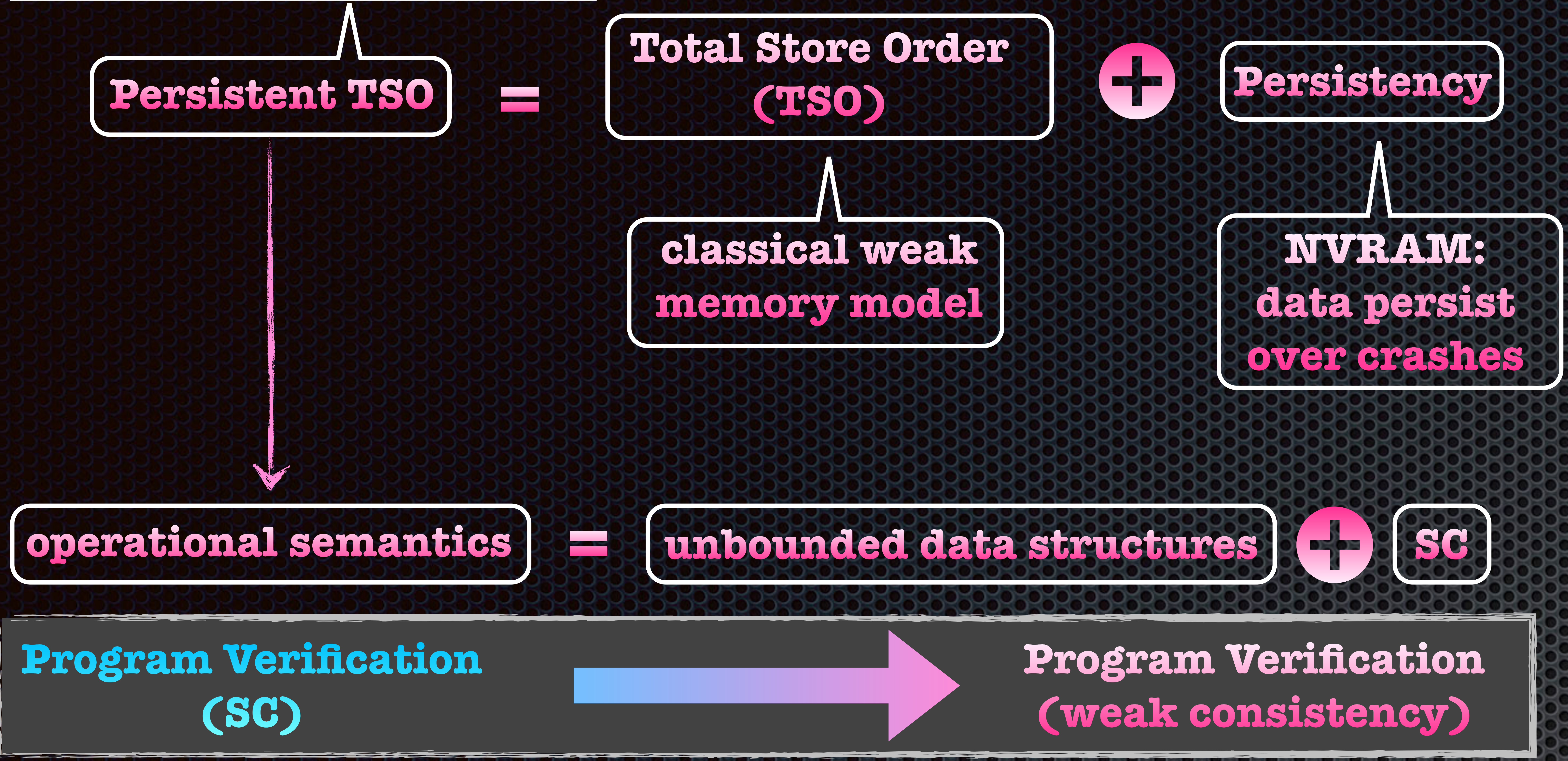
main stream Intel architecture: persistent x86-TSO



Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

this work



Persistent TSO = Total Store Order (TSO) + Persistency

classical weak memory model

NVRAM: data persist over crashes

operational semantics = unbounded data structures + SC

Program Verification (SC) → Program Verification (weak consistency)

main stream Intel architecture: persistent x86-TSO

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

this work

Persistent TSO = **Total Store Order (TSO)** + **Persistency**

classical weak memory model

NVRAM: data persist over crashes

operational semantics = **unbounded data structures** + **SC**

Program Verification (SC) → **Program Verification (weak consistency)**

main stream Intel architecture: persistent x86-TSO

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

this work

Persistent TSO = **Total Store Order (TSO)** + **Persistency**

Total Store Order (TSO): classical weak memory model

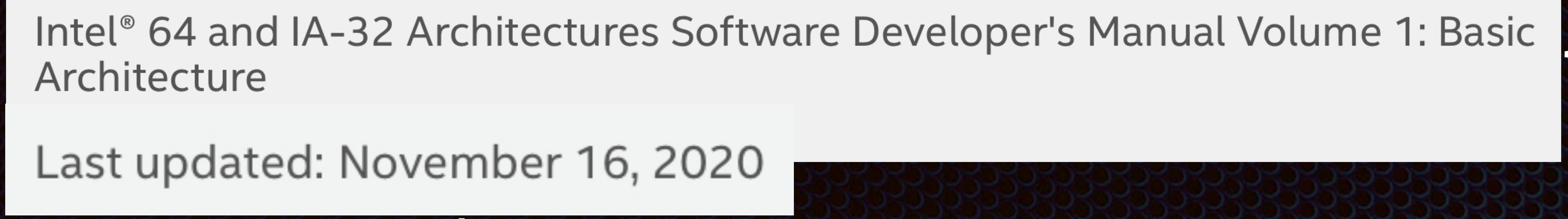
Persistency: NVRAM: data persist over crashes

operational semantics = **unbounded data structures** + **SC**

Program Verification (SC) → **Program Verification (weak consistency)**

semantics, decidability, complexity,
model checking, abstraction, under-approximation, …

main stream Intel architecture: persistent x86-TSO



Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

this work

Persistent TSO = Total Store Order (TSO) + Persistency

classical weak memory model

NVRAM: data persist over crashes

operational semantics = unbounded data structures + SC

Program Verification (SC) → Program Verification (weak consistency)

semantics, decidability, complexity,
model checking, abstraction, under-approximation, ...

main stream Intel architecture: persistent x86-TSO

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture

Last updated: November 16, 2020

this work

Persistent TSO = Total Store Order (TSO) + Persistency

classical weak memory model

NVRAM: data persist over crashes

operational semantics = unbounded data structures + SC

Program Verification (SC)

Reachability for finite-state programs is decidable

Program Verification (weak consistency)

semantics, decidability, complexity,
model checking, abstraction, under-approximation, …

unbounded
data structures

adapting SC techniques

semantics

decidability

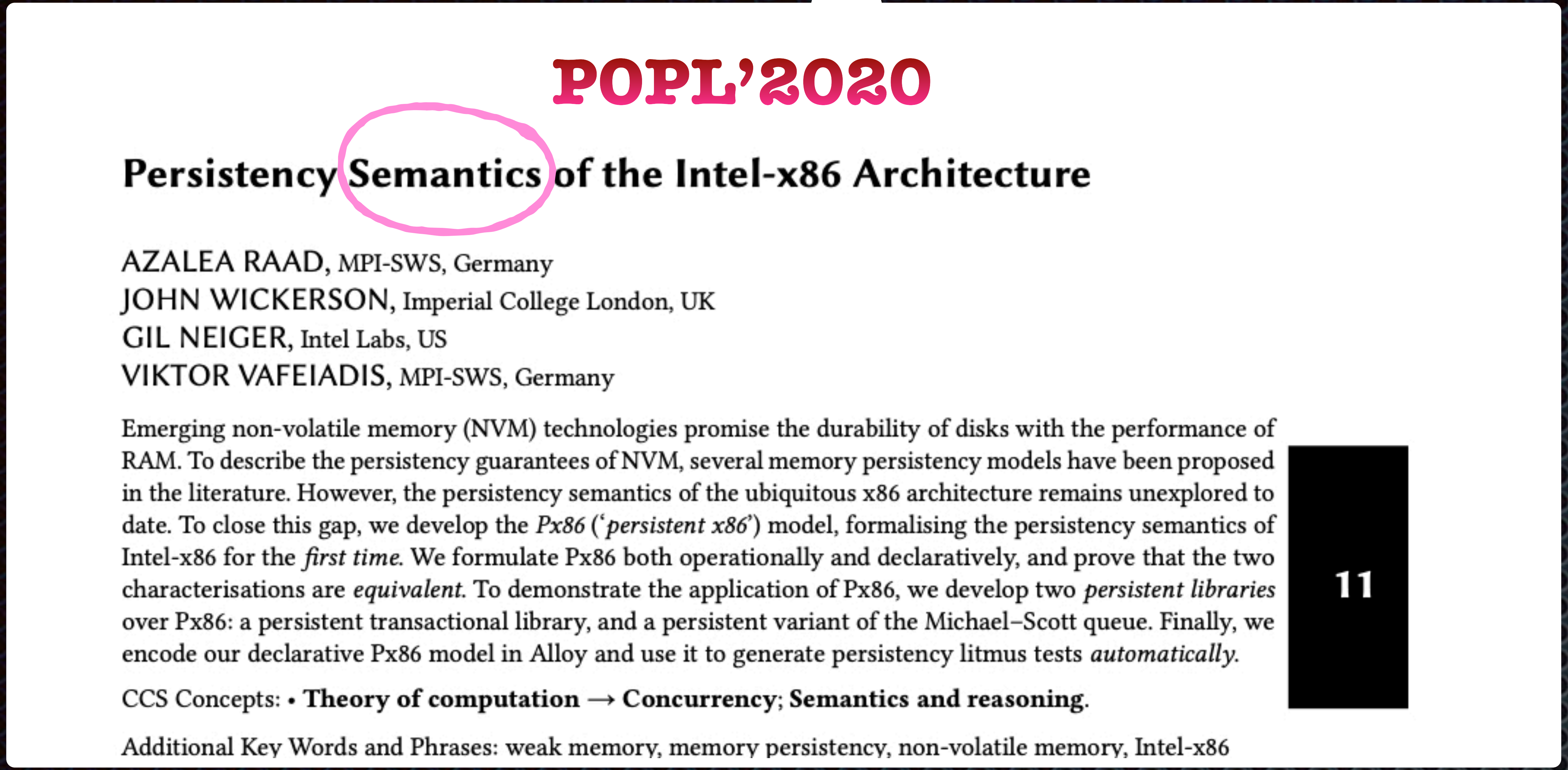
unbounded data structures

adapting SC techniques

semantics

decidability

Persistent TSO



POPL'2020

Persistency Semantics of the Intel-x86 Architecture

AZALEA RAAD, MPI-SWS, Germany
JOHN WICKERSON, Imperial College London, UK
GIL NEIGER, Intel Labs, US
VIKTOR VAFEIADIS, MPI-SWS, Germany

Emerging non-volatile memory (NVM) technologies promise the durability of disks with the performance of RAM. To describe the persistency guarantees of NVM, several memory persistency models have been proposed in the literature. However, the persistency semantics of the ubiquitous x86 architecture remains unexplored to date. To close this gap, we develop the *Px86* ('*persistent x86*') model, formalising the persistency semantics of Intel-x86 for the *first time*. We formulate Px86 both operationally and declaratively, and prove that the two characterisations are *equivalent*. To demonstrate the application of Px86, we develop two *persistent libraries* over Px86: a persistent transactional library, and a persistent variant of the Michael–Scott queue. Finally, we encode our declarative Px86 model in Alloy and use it to generate persistency litmus tests *automatically*.

CCS Concepts: • **Theory of computation** → **Concurrency**; **Semantics and reasoning**.

Additional Key Words and Phrases: weak memory, memory persistency, non-volatile memory, Intel-x86

11

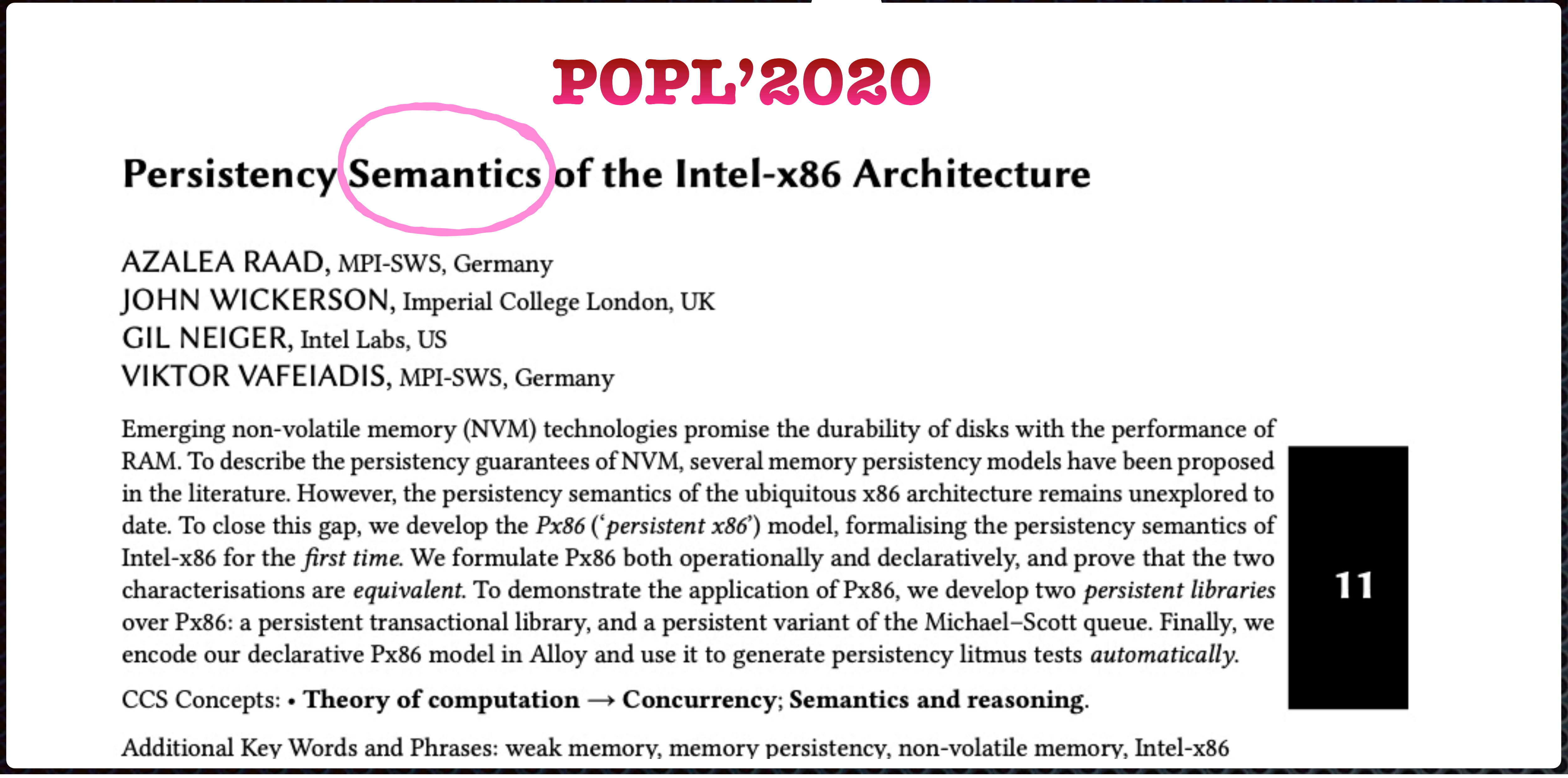
unbounded data structures

adapting SC techniques

semantics

decidability

Persistent TSO



POPL'2020

Persistency Semantics of the Intel-x86 Architecture

AZALEA RAAD, MPI-SWS, Germany
JOHN WICKERSON, Imperial College London, UK
GIL NEIGER, Intel Labs, US
VIKTOR VAFEIADIS, MPI-SWS, Germany

Emerging non-volatile memory (NVM) technologies promise the durability of disks with the performance of RAM. To describe the persistency guarantees of NVM, several memory persistency models have been proposed in the literature. However, the persistency semantics of the ubiquitous x86 architecture remains unexplored to date. To close this gap, we develop the *Px86* (‘*persistent x86*’) model, formalising the persistency semantics of Intel-x86 for the *first time*. We formulate Px86 both operationally and declaratively, and prove that the two characterisations are *equivalent*. To demonstrate the application of Px86, we develop two *persistent libraries* over Px86: a persistent transactional library, and a persistent variant of the Michael–Scott queue. Finally, we encode our declarative Px86 model in Alloy and use it to generate persistency litmus tests *automatically*.

11

CCS Concepts: • **Theory of computation** → **Concurrency**; **Semantics and reasoning**.

Additional Key Words and Phrases: weak memory, memory persistency, non-volatile memory, Intel-x86

Taming x86-TSO Persistency

Artem Khyzha Tel Aviv University, Ori Lahav Tel Aviv University

POPL'2021

PerSeVerE: Persistency Semantics for Verification under Ext4

Michalis Kokologiannakis MPI-SWS, Germany, Ilya Kaysin National Research University Higher School of Economics, JetBrains Research, Azalea Raad Imperial College London, Viktor Vafeiadis MPI-SWS

POPL'2021

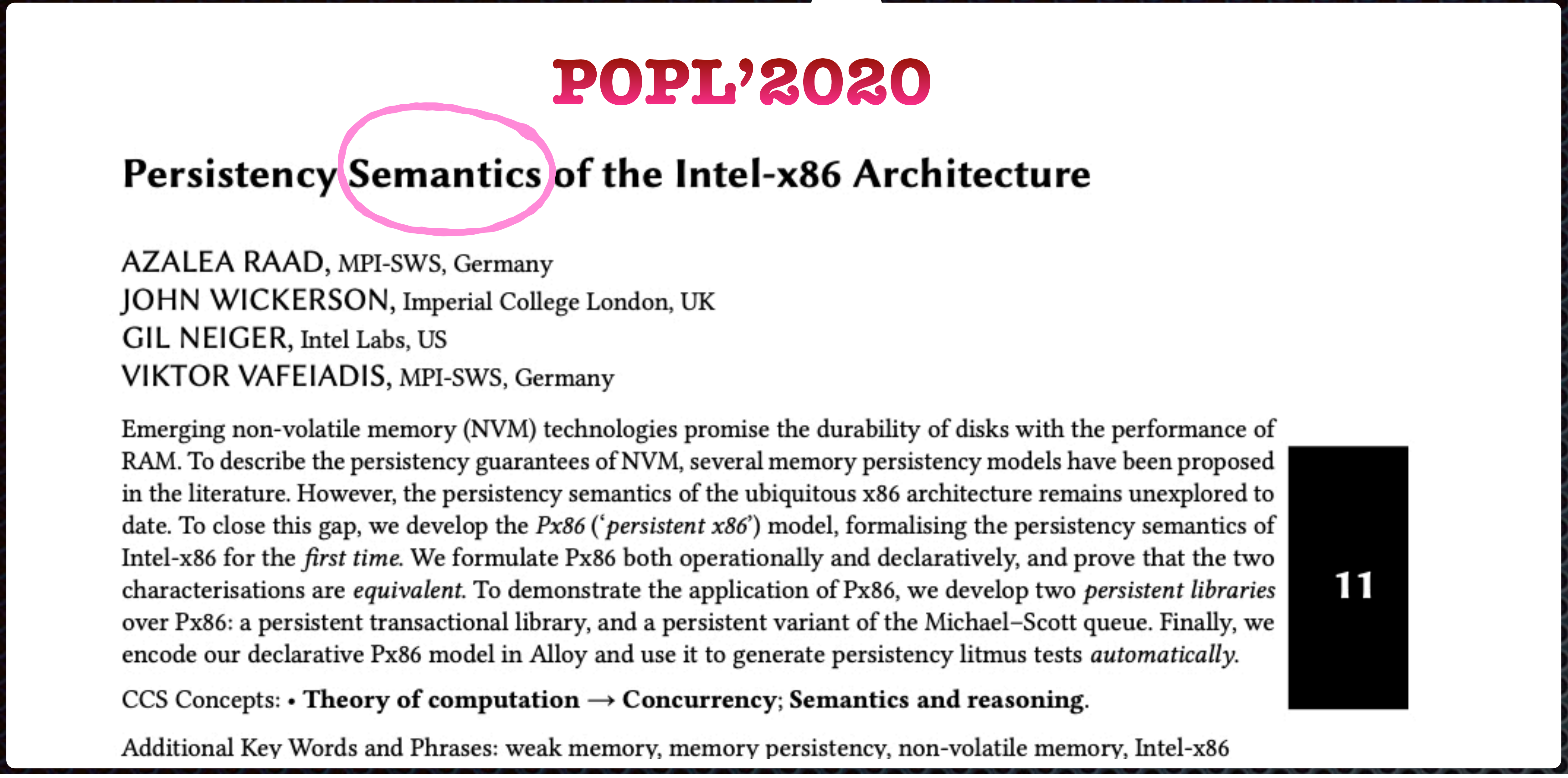
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO



POPL'2020

Persistency Semantics of the Intel-x86 Architecture

AZALEA RAAD, MPI-SWS, Germany
JOHN WICKERSON, Imperial College London, UK
GIL NEIGER, Intel Labs, US
VIKTOR VAFEIADIS, MPI-SWS, Germany

Emerging non-volatile memory (NVM) technologies promise the durability of disks with the performance of RAM. To describe the persistency guarantees of NVM, several memory persistency models have been proposed in the literature. However, the persistency semantics of the ubiquitous x86 architecture remains unexplored to date. To close this gap, we develop the *Px86* ('*persistent x86*') model, formalising the persistency semantics of Intel-x86 for the *first time*. We formulate Px86 both operationally and declaratively, and prove that the two characterisations are *equivalent*. To demonstrate the application of Px86, we develop two *persistent libraries* over Px86: a persistent transactional library, and a persistent variant of the Michael–Scott queue. Finally, we encode our declarative Px86 model in Alloy and use it to generate persistency litmus tests *automatically*.

11

CCS Concepts: • **Theory of computation** → **Concurrency**; **Semantics and reasoning**.

Additional Key Words and Phrases: weak memory, memory persistency, non-volatile memory, Intel-x86

Taming x86-TSO Persistency
Artem Khyzha Tel Aviv University, Ori Lahav Tel Aviv University

POPL'2021

PerSeVerE: Persistency Semantics for Verification under Ext4
Michalis Kokologiannakis MPI-SWS, Germany, Ilya Kaysin National Research University Higher School of Economics, JetBrains Research, Azalea Raad Imperial College London, Viktor Vafeiadis MPI-SWS

POPL'2021

- Equivelant to the Raad et al semantics
- Allows applying classical (SC) techniques
- "Nice" datastructures

this work:
new semantics

unbounded
data structures

adapting SC techniques

semantics decidability complexity

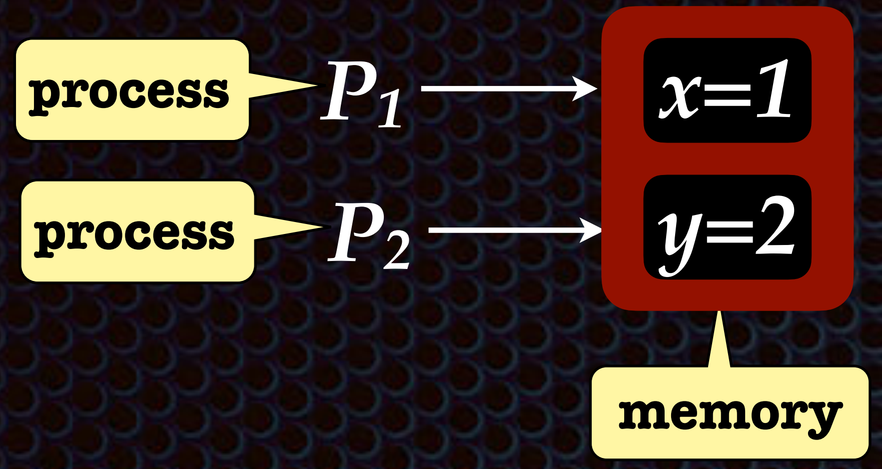
unbounded
data structures

adapting SC techniques

semantics

decidability

complexity

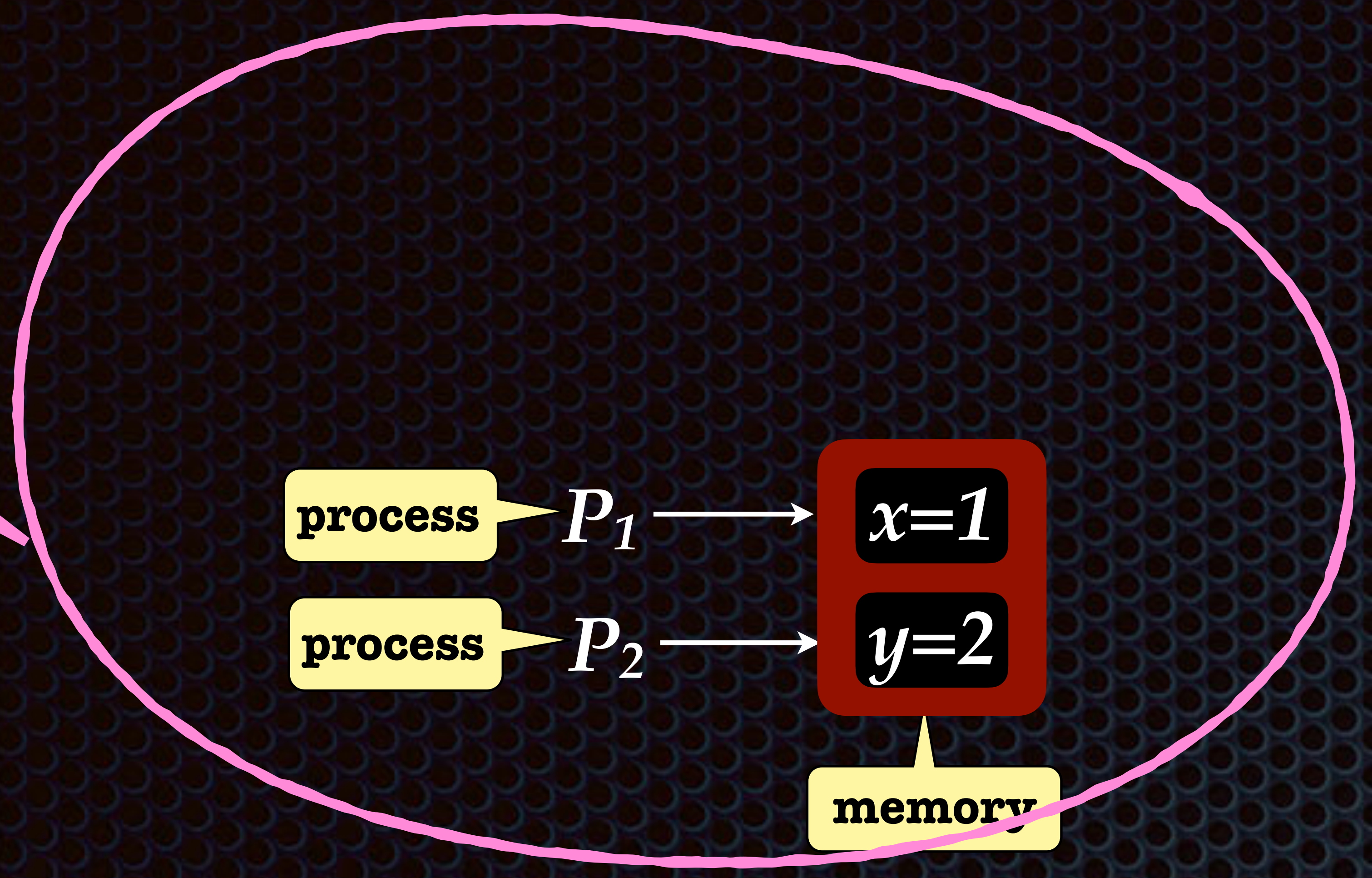

process
P_1
process
P_2
$x=1$
$y=2$
memory

unbounded data structures

adapting SC techniques

semantics **decidability** **complexity**

classical SC semantics



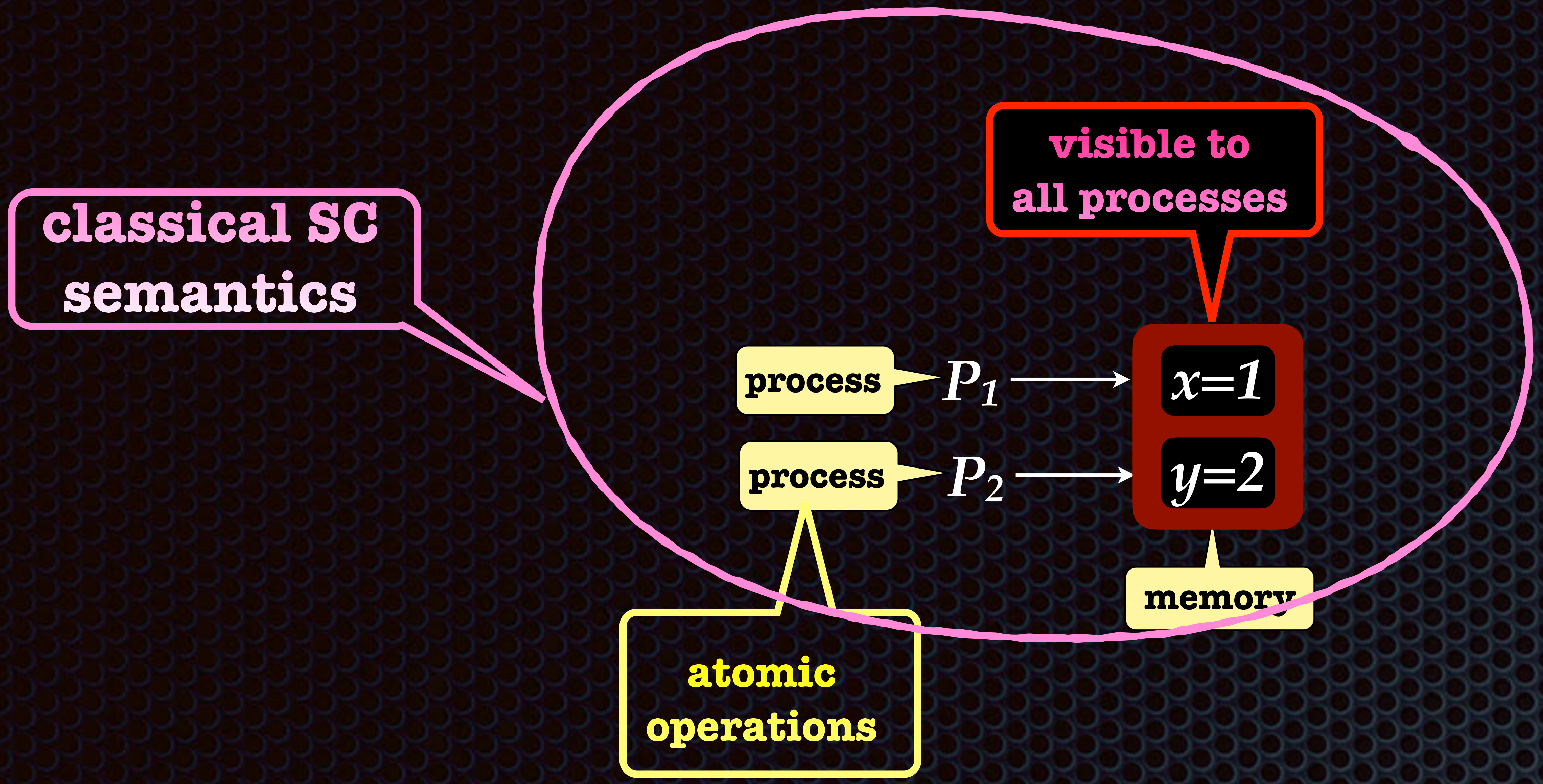
unbounded data structures

adapting SC techniques

semantics

decidability

complexity



unbounded data structures

adapting SC techniques

semantics

decidability

process P_1

process P_2

$x=1$

$y=1$

memory

unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency

unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency

process P_1

process P_2

$x=1$

$y=1$

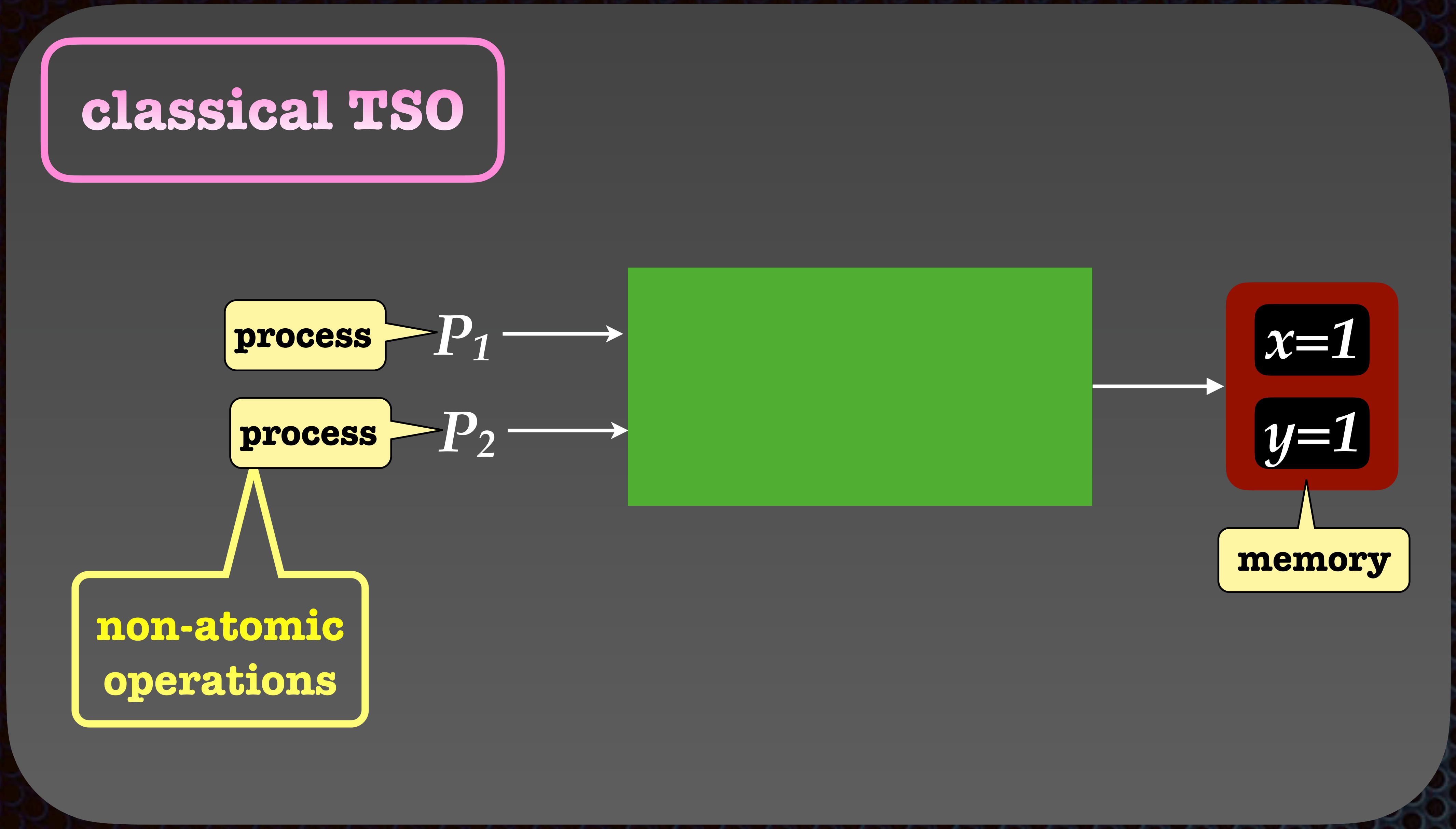
memory

unbounded data structures

adapting SC techniques

semantics **decidability**

Persistent TSO = Classical TSO + Persistency



classical TSO

process P_1

process P_2

non-atomic operations

$x=1$

$y=1$

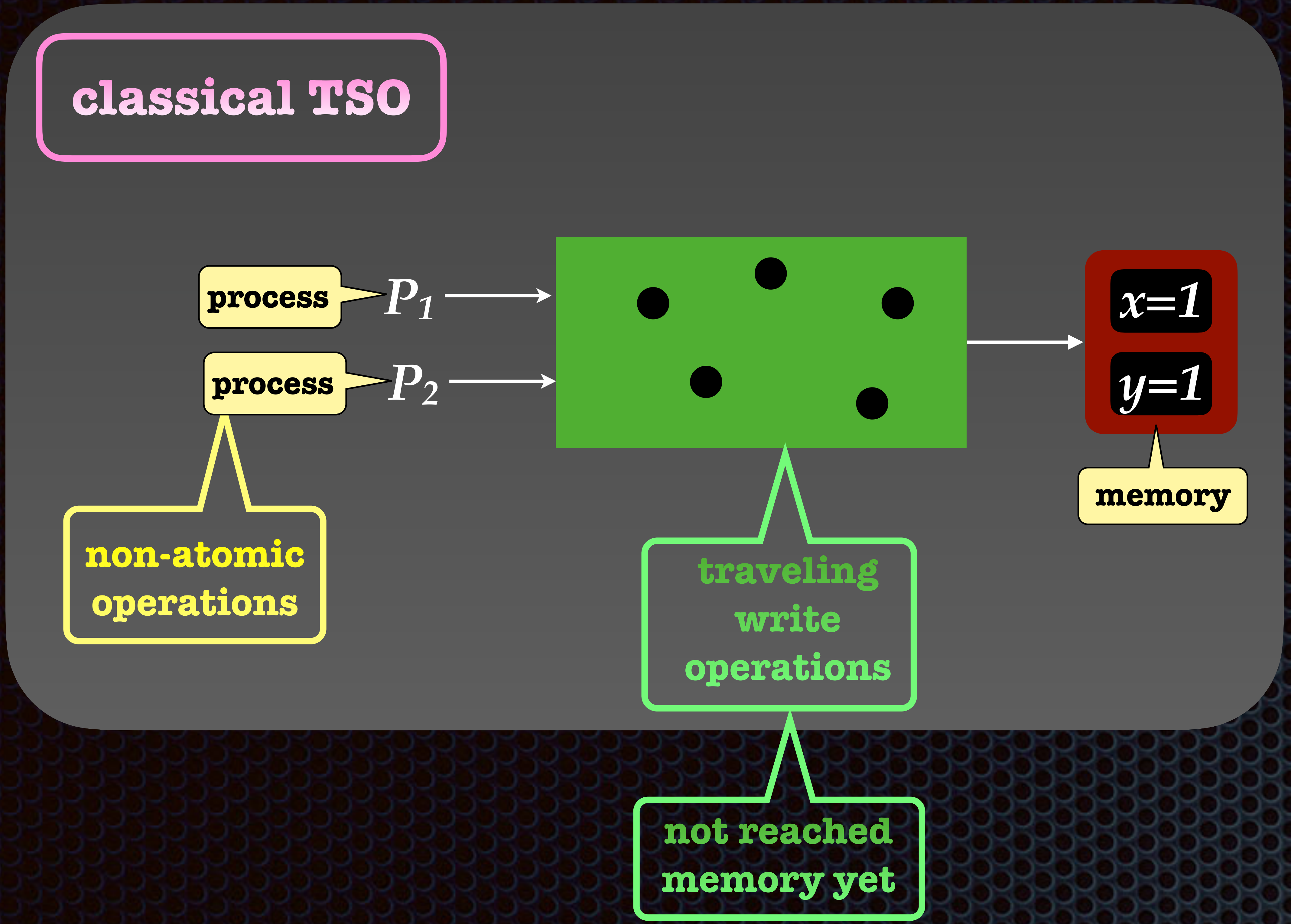
memory

unbounded
data structures

adapting SC techniques

semantics decidability

Persistent TSO = Classical TSO + Persistency

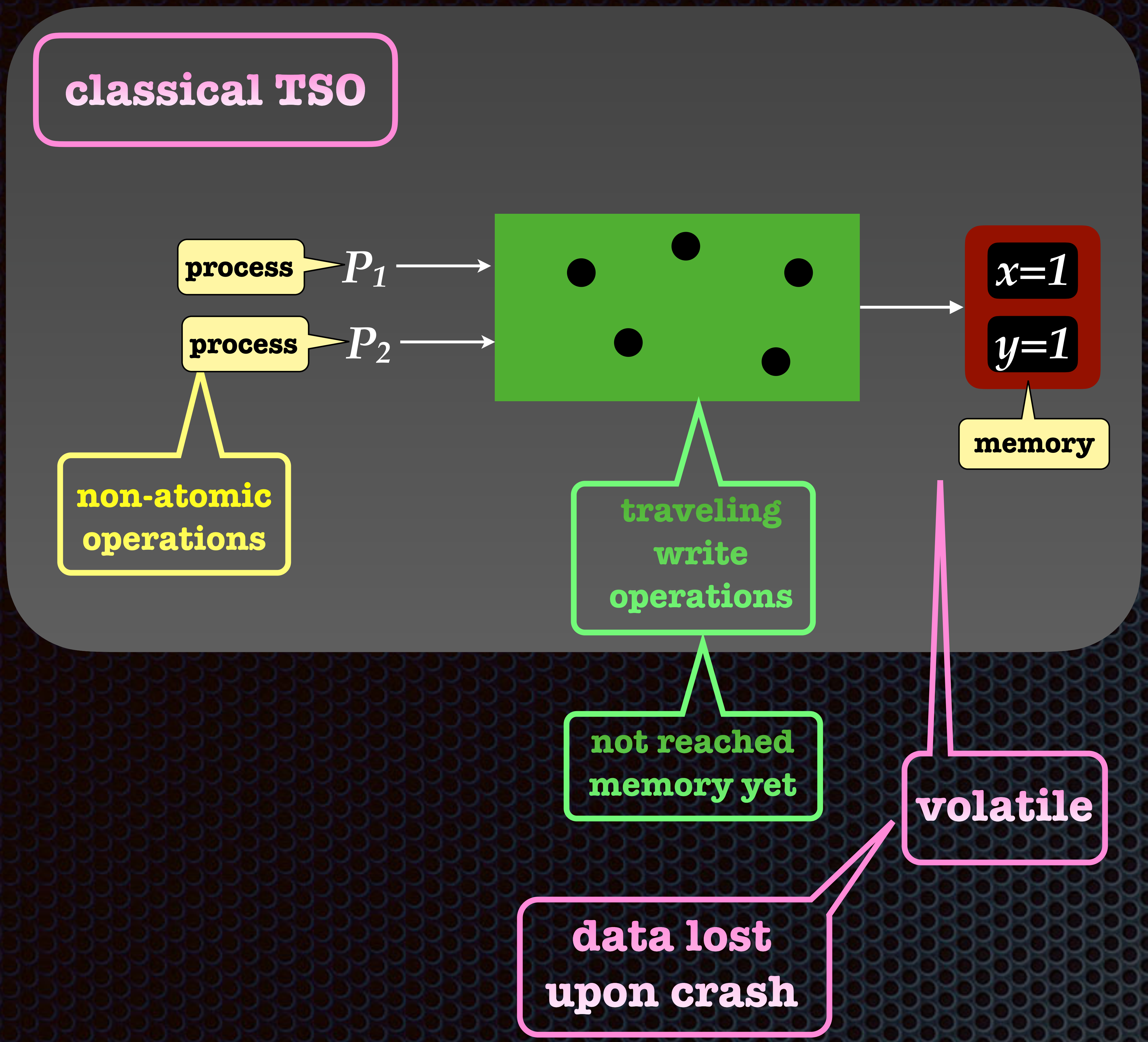


unbounded
data structures

adapting SC techniques

semantics **decidability**

Persistent TSO = Classical TSO + Persistency



classical TSO

process P_1

process P_2

x=1

y=1

memory

non-atomic operations

traveling write operations

not reached memory yet

volatile

data lost upon crash

unbounded
data structures

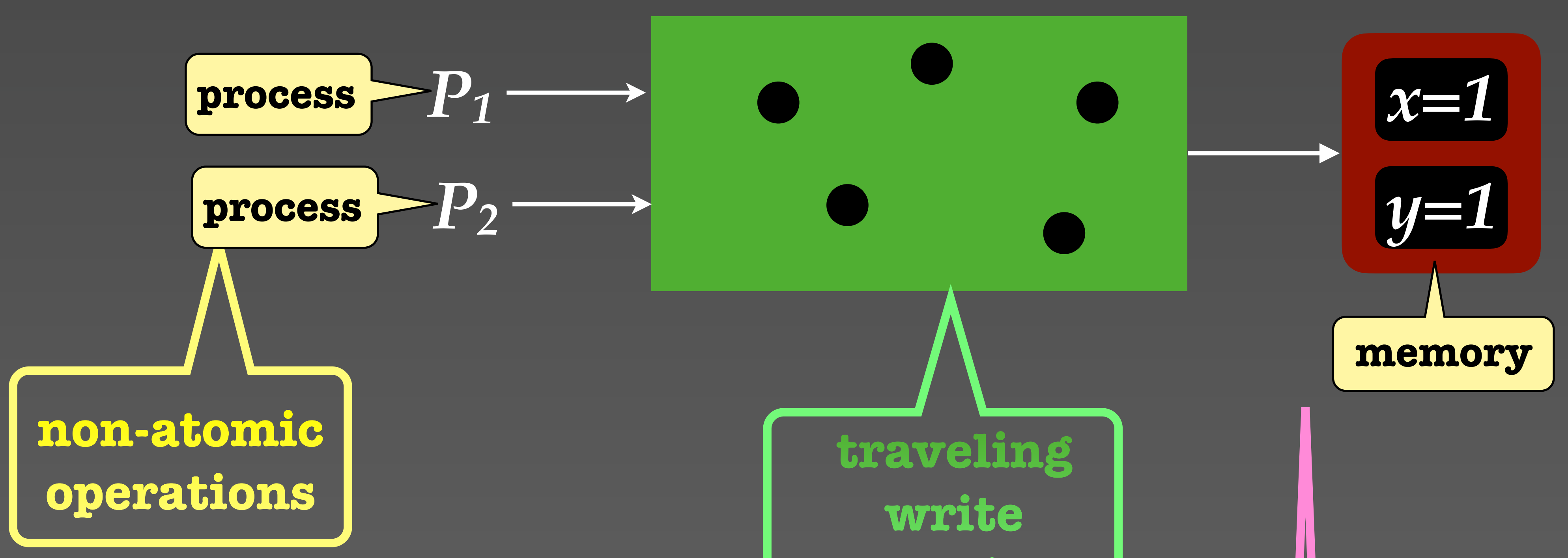
adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency

classical TSO



non-atomic
operations

traveling
write
operations

not reached
memory yet

volatile

data lost
upon crash

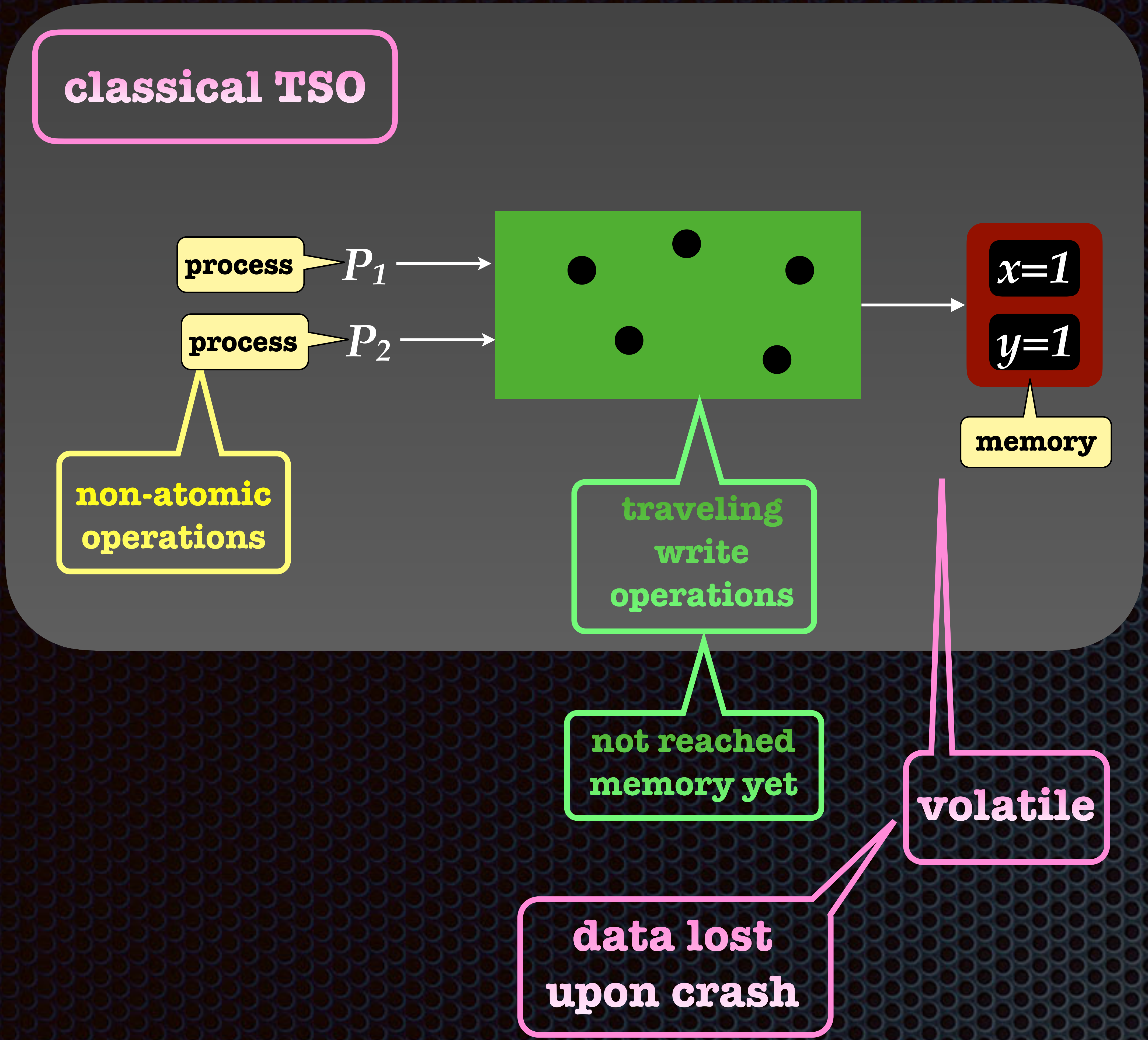
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



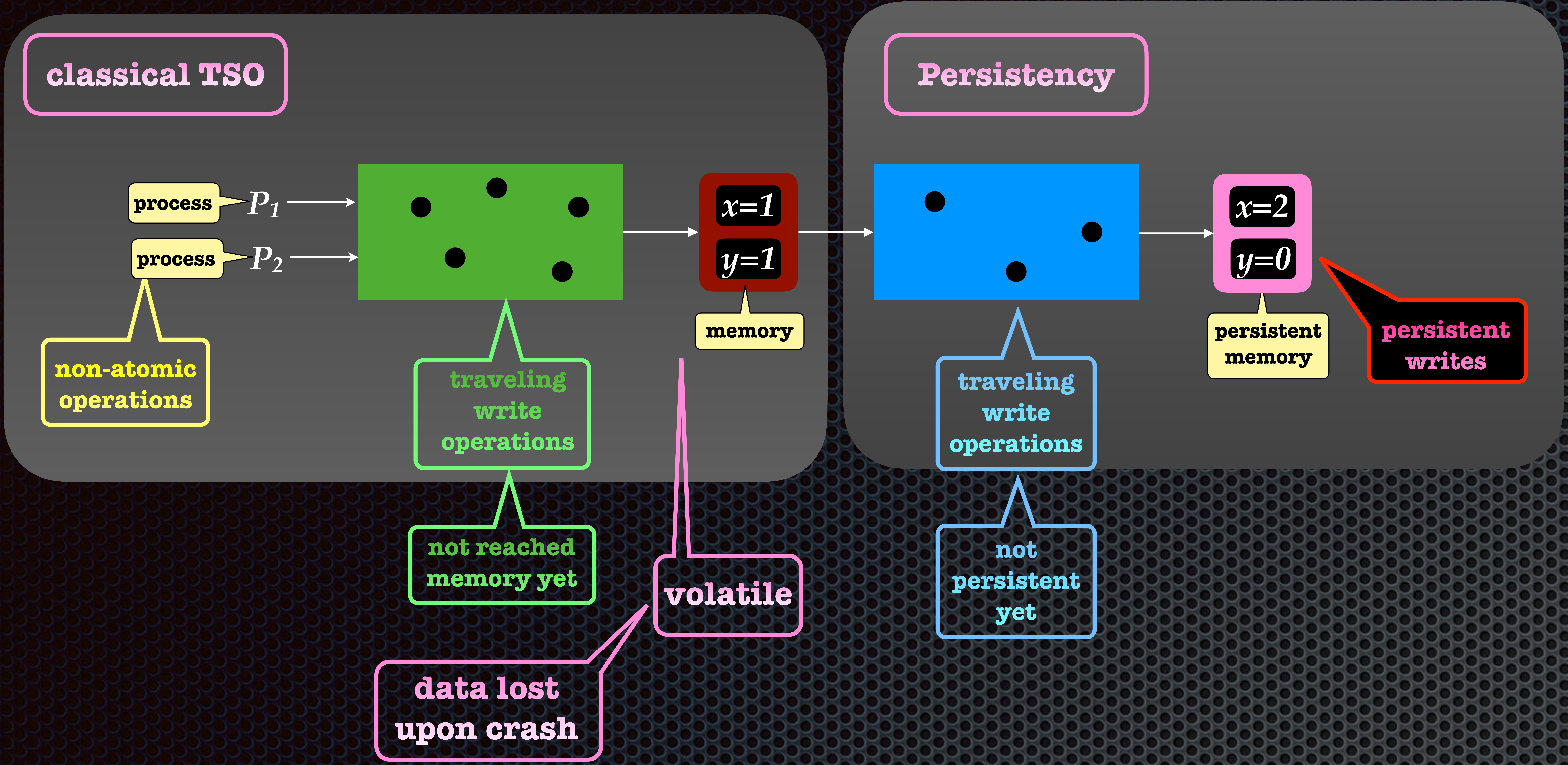
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



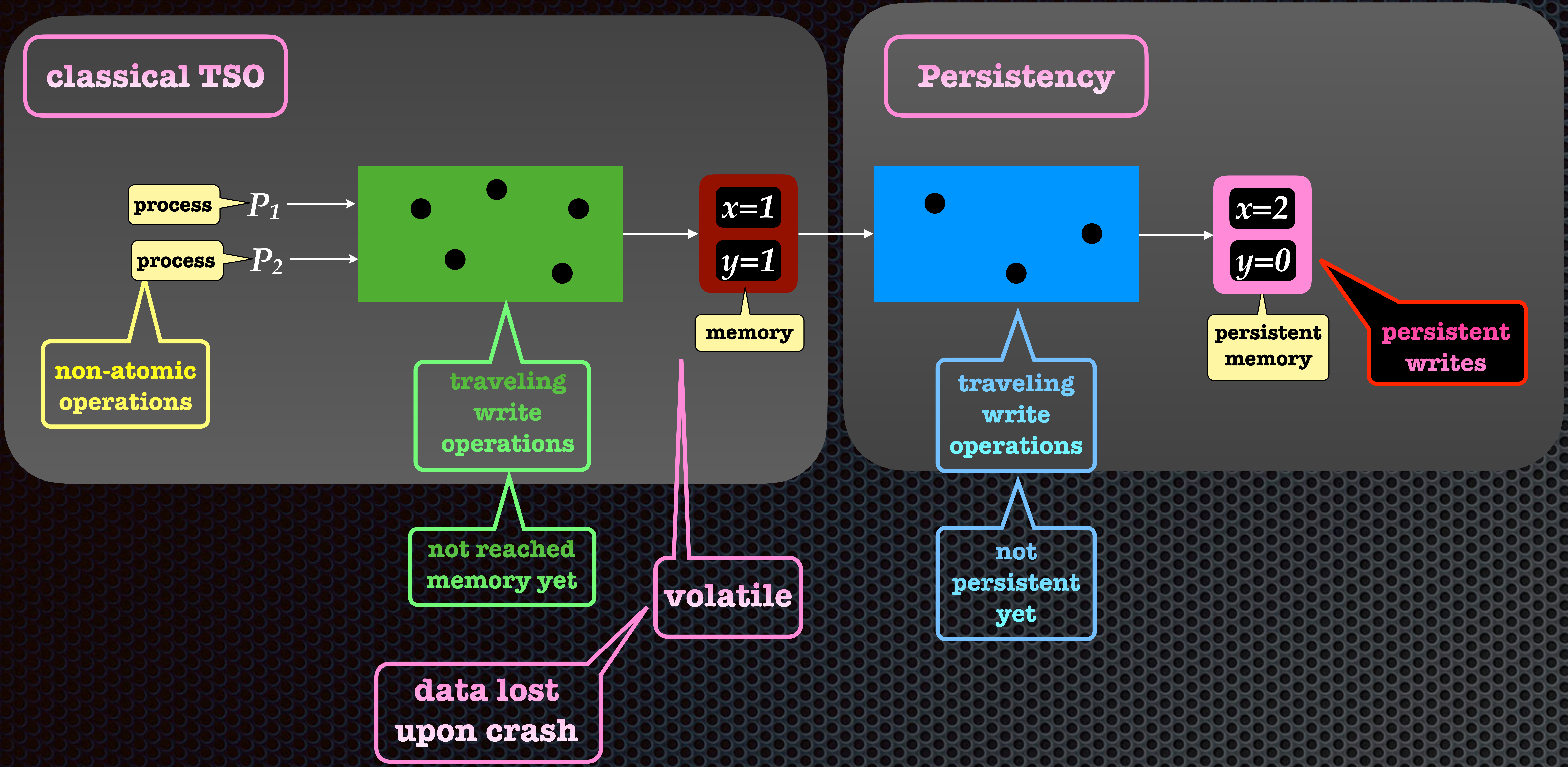
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



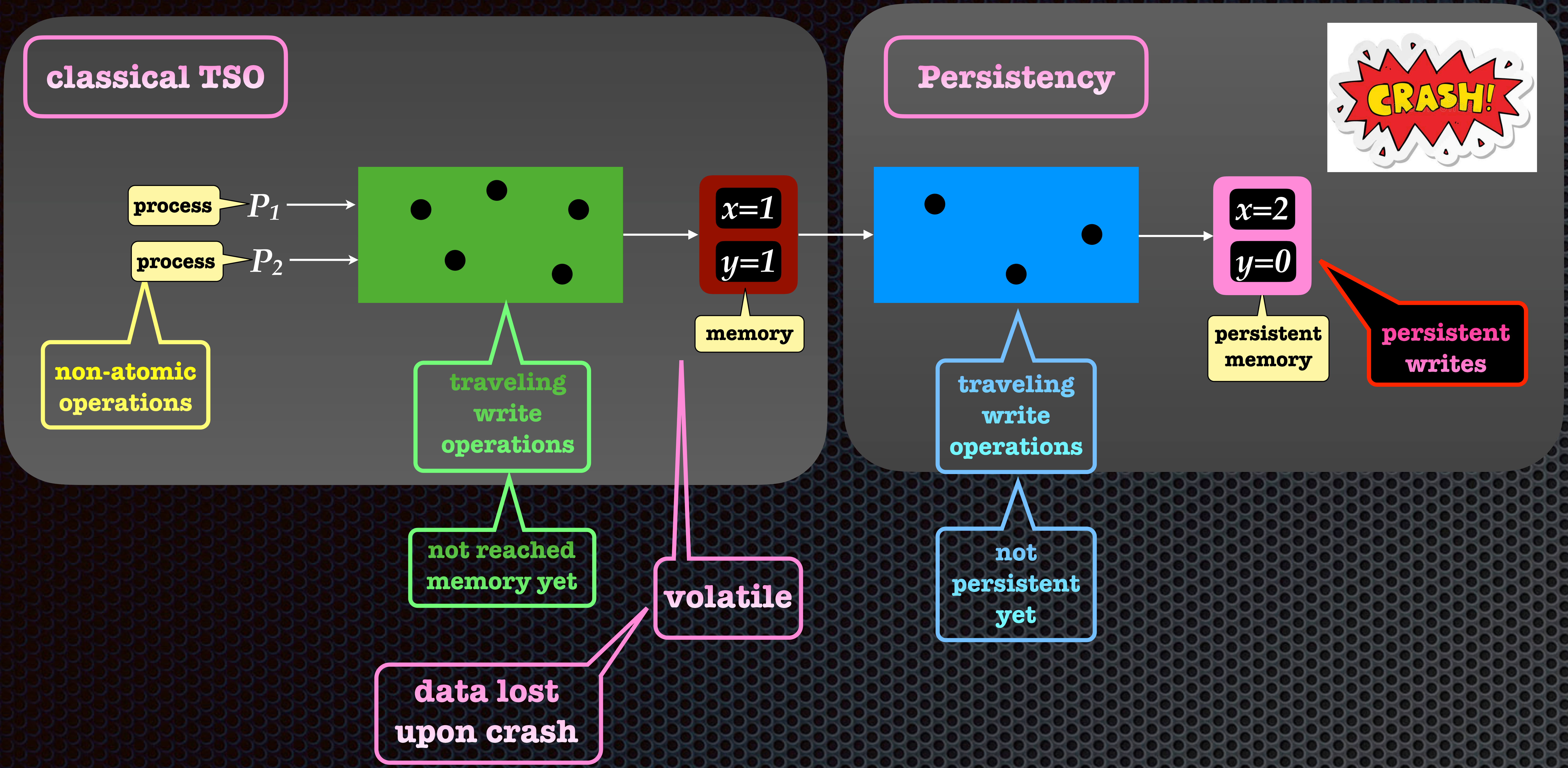
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



classical TSO

process P_1

process P_2

non-atomic operations

traveling write operations

not reached memory yet

$x=1$

$y=1$

memory

volatile

data lost upon crash

Persistency

traveling write operations

not persistent yet

$x=2$

$y=0$

persistent memory

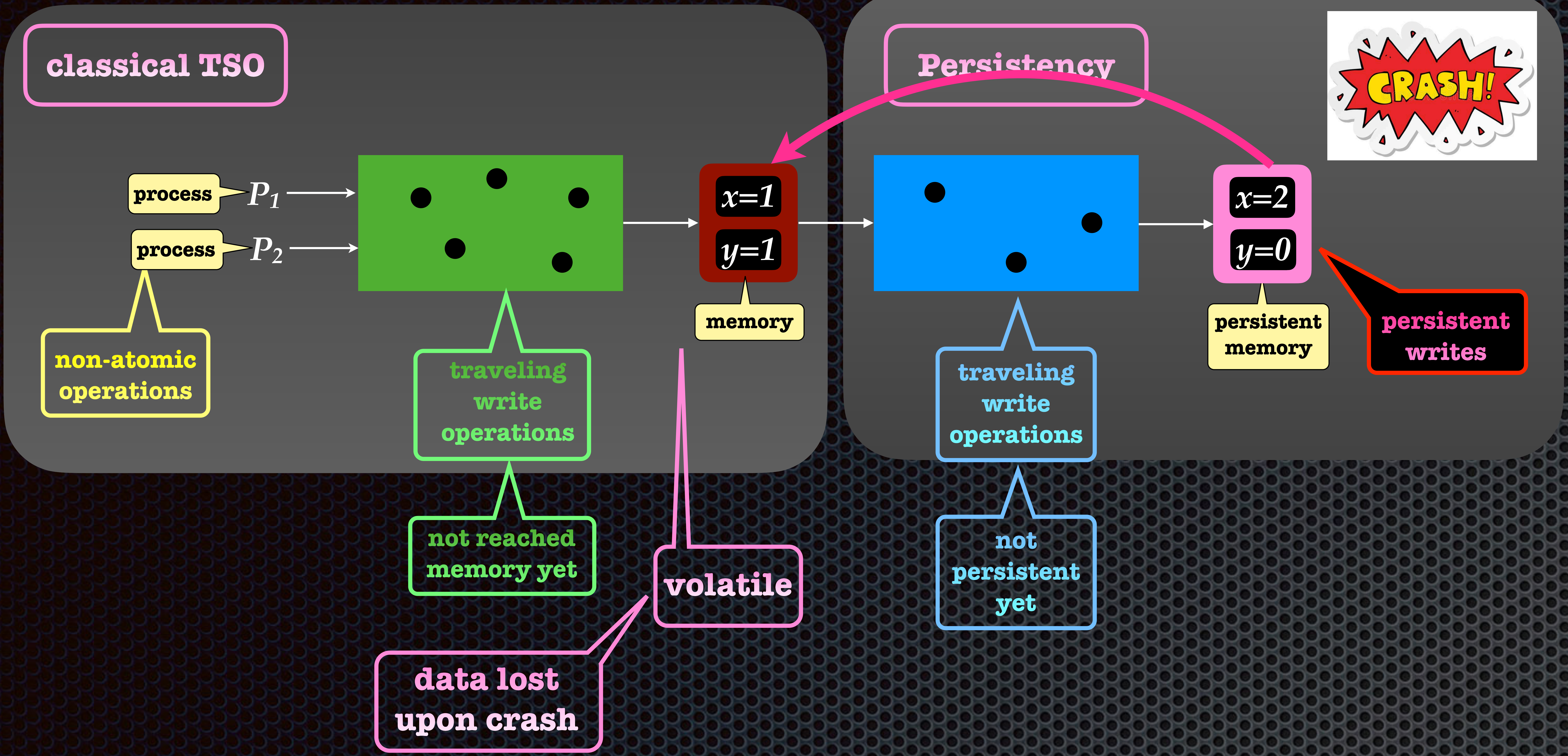
persistent writes

unbounded
data structures

adapting SC techniques

semantics decidability

Persistent TSO = Classical TSO + Persistency


classical TSO

Persistency

process P_1

process P_2

non-atomic operations

traveling write operations

not reached memory yet

$x=1$

$y=1$

memory

volatile

data lost upon crash

traveling write operations

not persistent yet

$x=2$

$y=0$

persistent memory

persistent writes

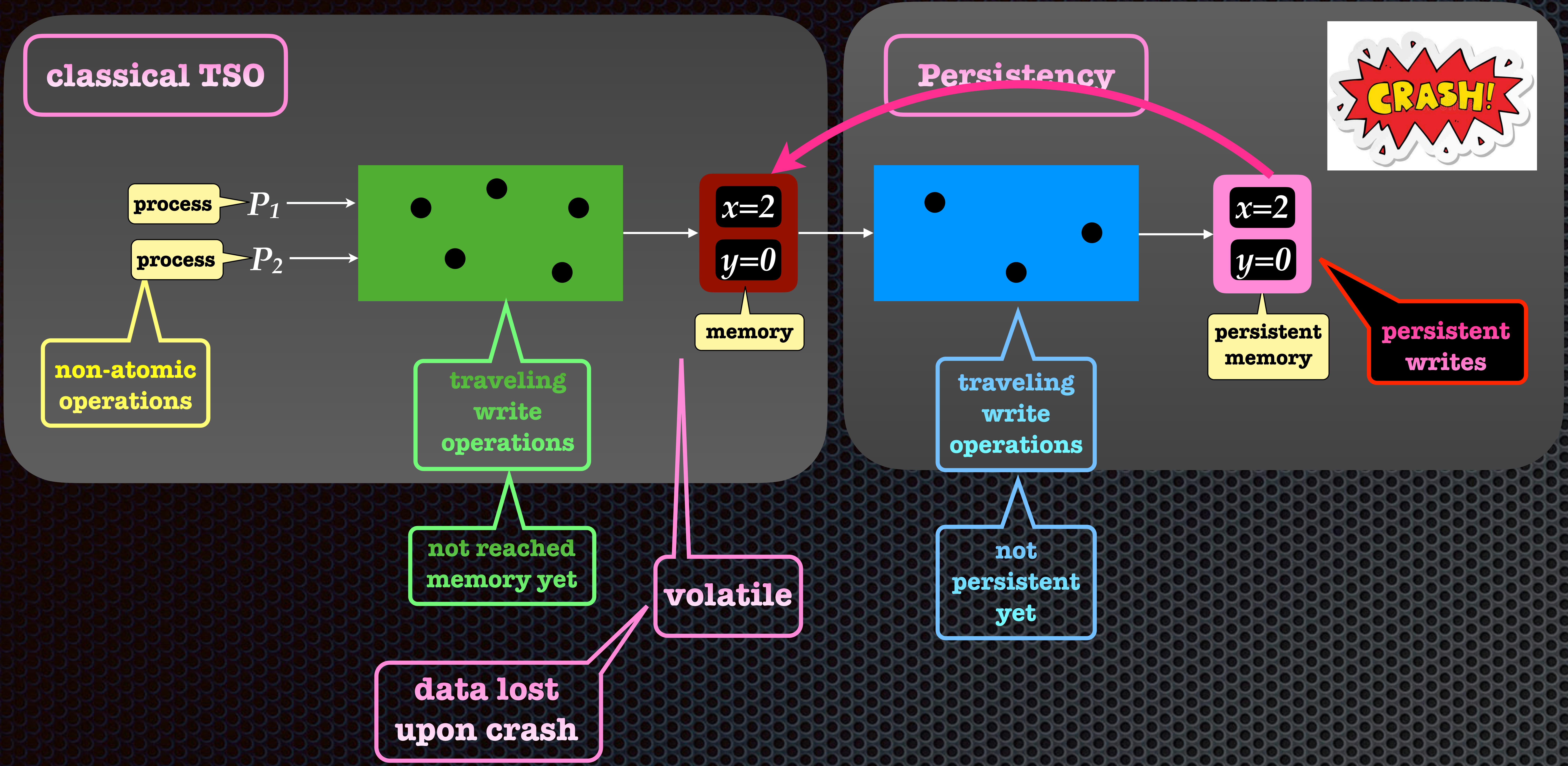
unbounded data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



classical TSO

process

P_1

process

P_2

non-atomic operations

traveling write operations

not reached memory yet

$x=2$

$y=0$

memory

volatile

data lost upon crash

Persistency

CRASH!

traveling write operations

not persistent yet

$x=2$

$y=0$

persistent memory

persistent writes

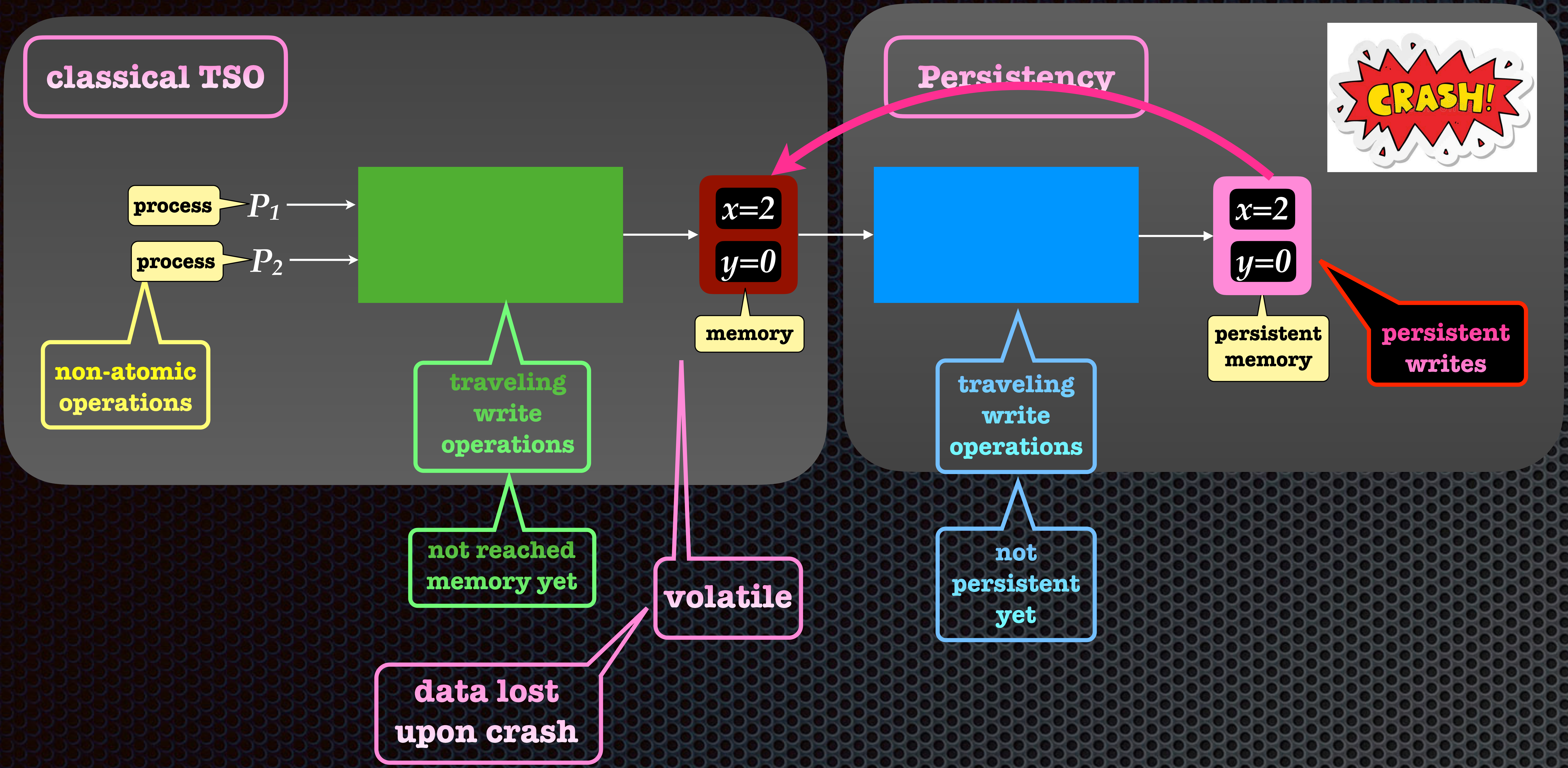
unbounded data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



classical TSO

process

P_1

process

P_2

non-atomic operations

traveling write operations

not reached memory yet

$x=2$

$y=0$

memory

volatile

data lost upon crash

Persistency

traveling write operations

not persistent yet

$x=2$

$y=0$

persistent memory

persistent writes

CRASH!

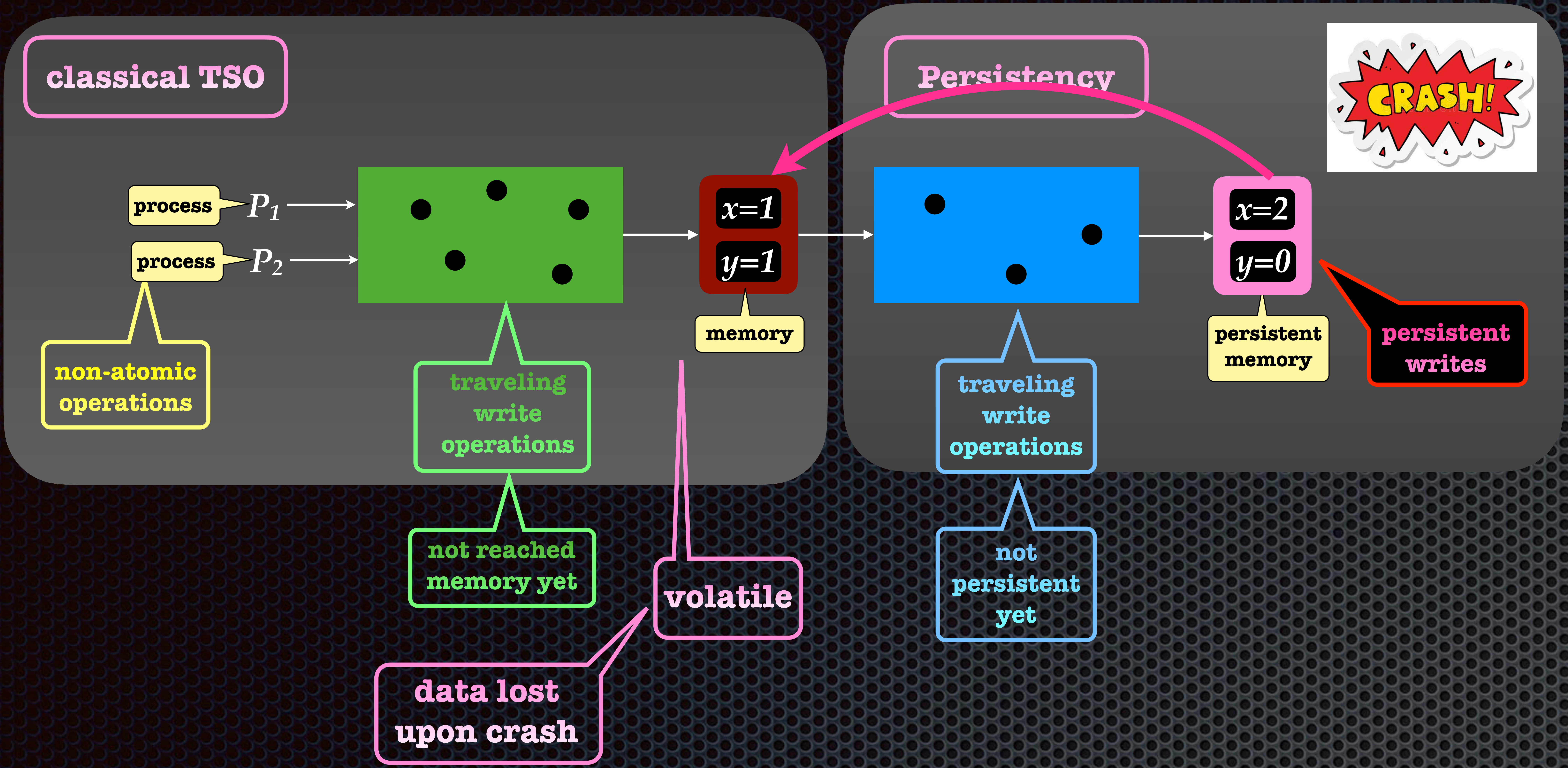
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



classical TSO

process P_1

process P_2

non-atomic operations

traveling write operations

not reached memory yet

$x=1$

$y=1$

memory

volatile

data lost upon crash

Persistency

traveling write operations

not persistent yet

$x=2$

$y=0$

persistent memory

persistent writes

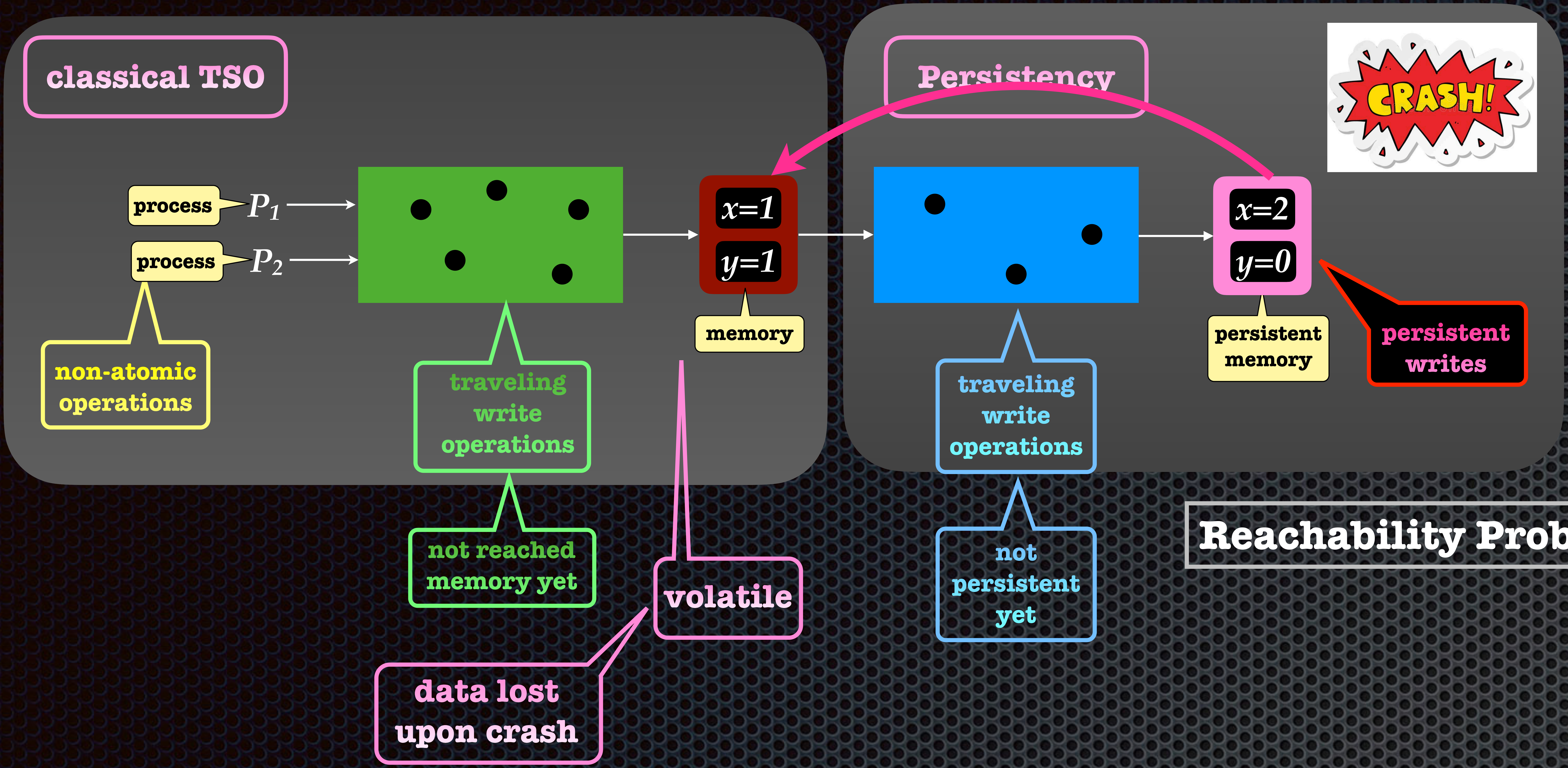
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency


classical TSO

process P_1

process P_2

non-atomic operations

traveling write operations

not reached memory yet

$x=1$

$y=1$

memory

volatile

data lost upon crash

Persistency

CRASH!

traveling write operations

not persistent yet

$x=2$

$y=0$

persistent memory

persistent writes

Reachability Problem:

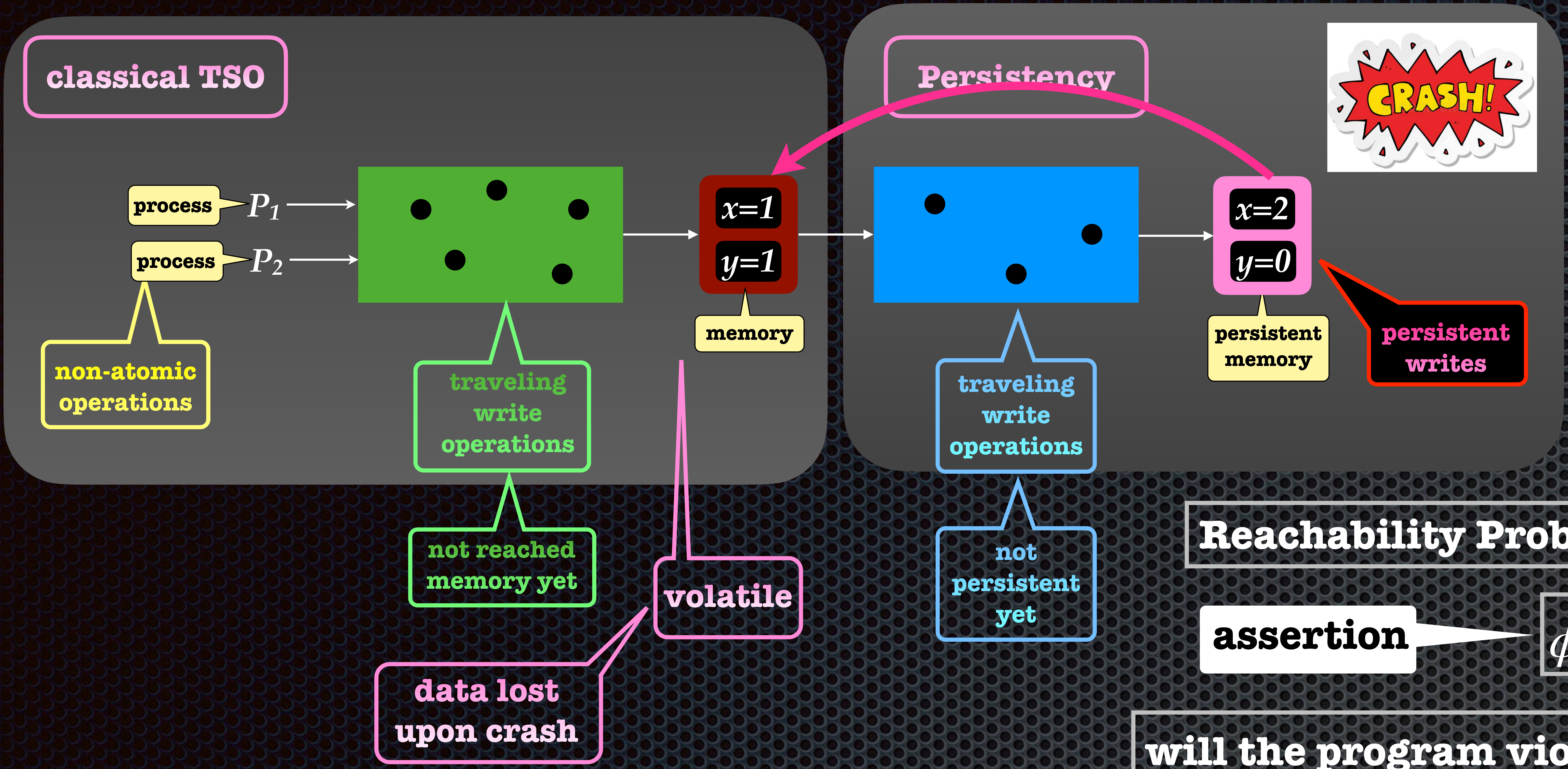
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



Reachability Problem:

assertion → ϕ

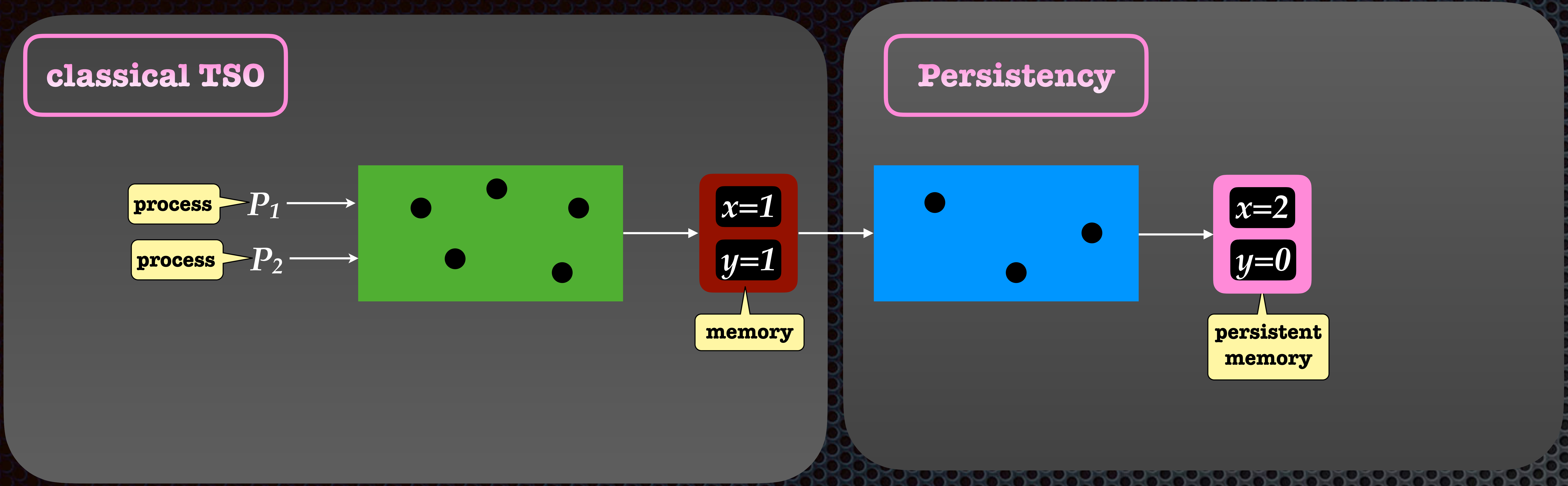
will the program violate ϕ?

unbounded
data structures

adapting SC techniques

semantics decidability

Persistent TSO = Classical TSO + Persistency



Reachability Problem:

assertion ϕ

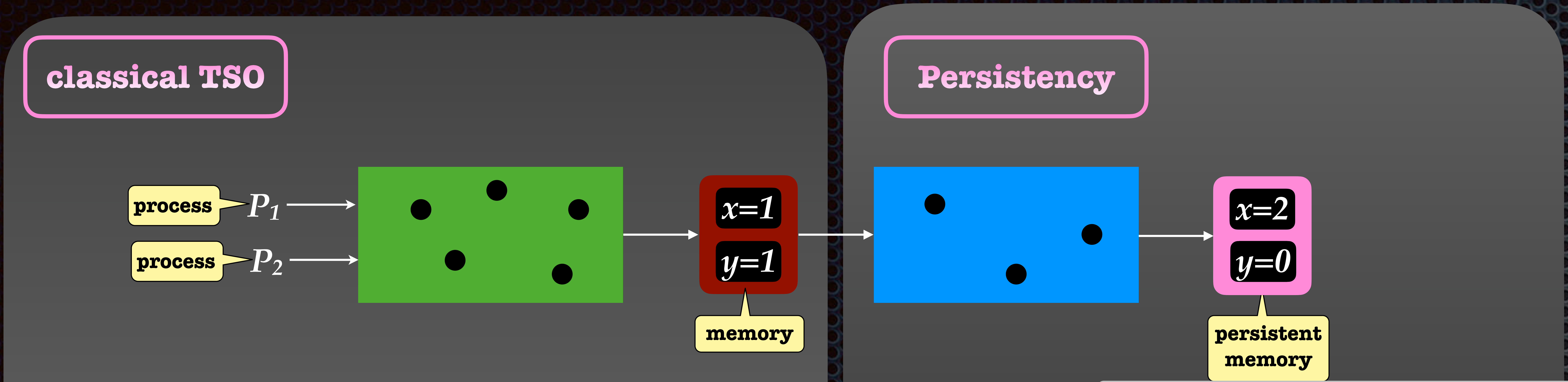
will the program violate ϕ?

unbounded
data structures

adapting SC techniques

semantics **decidability**

Persistent TSO = Classical TSO + Persistency



Main Result: Problem Decidable

Reachability Problem:

assertion ϕ

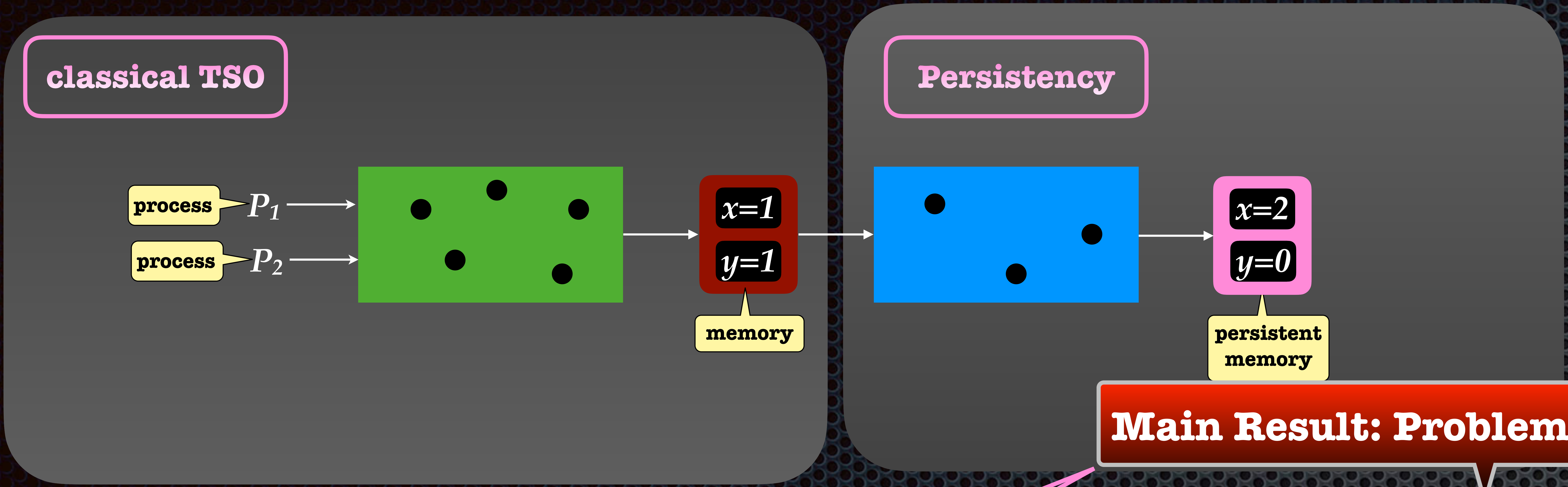
will the program violate ϕ?

unbounded
data structures

adapting SC techniques

semantics **decidability**

Persistent TSO = Classical TSO + Persistency



Main Result: Problem Decidable

challenges

Reachability Problem:

assertion ϕ

will the program violate ϕ?

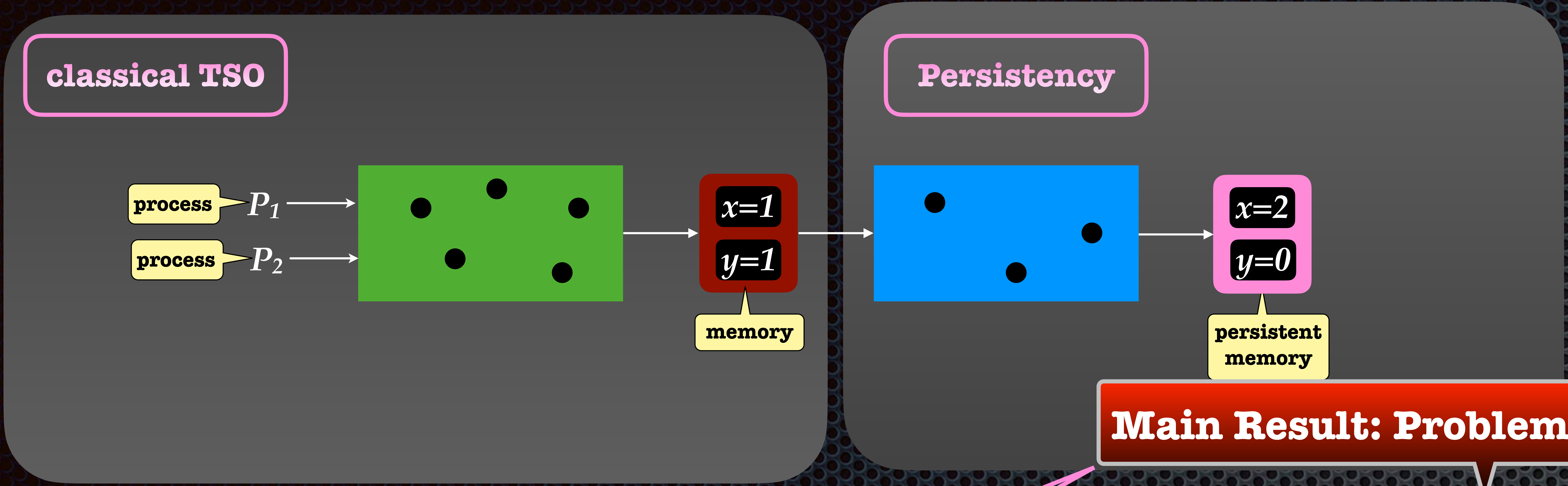
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



Main Result: Problem Decidable

challenges

Reachability Problem:

assertion ϕ

will the program violate ϕ?

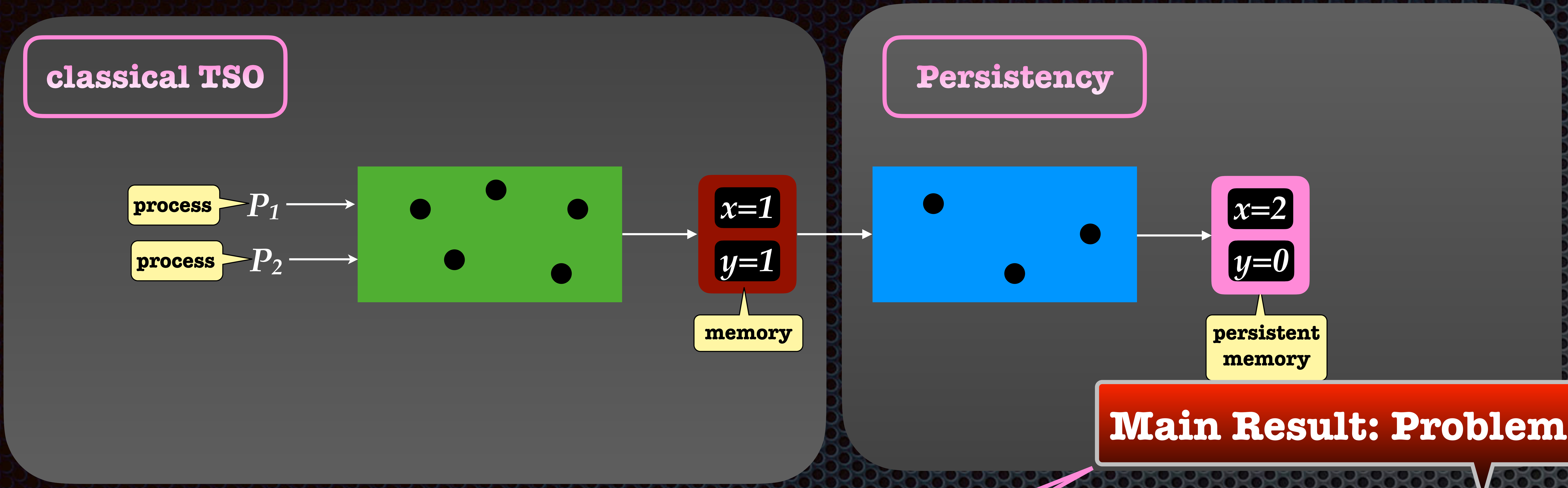
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



Main Result: Problem Decidable

Reachability Problem:

assertion ϕ

will the program violate ϕ?

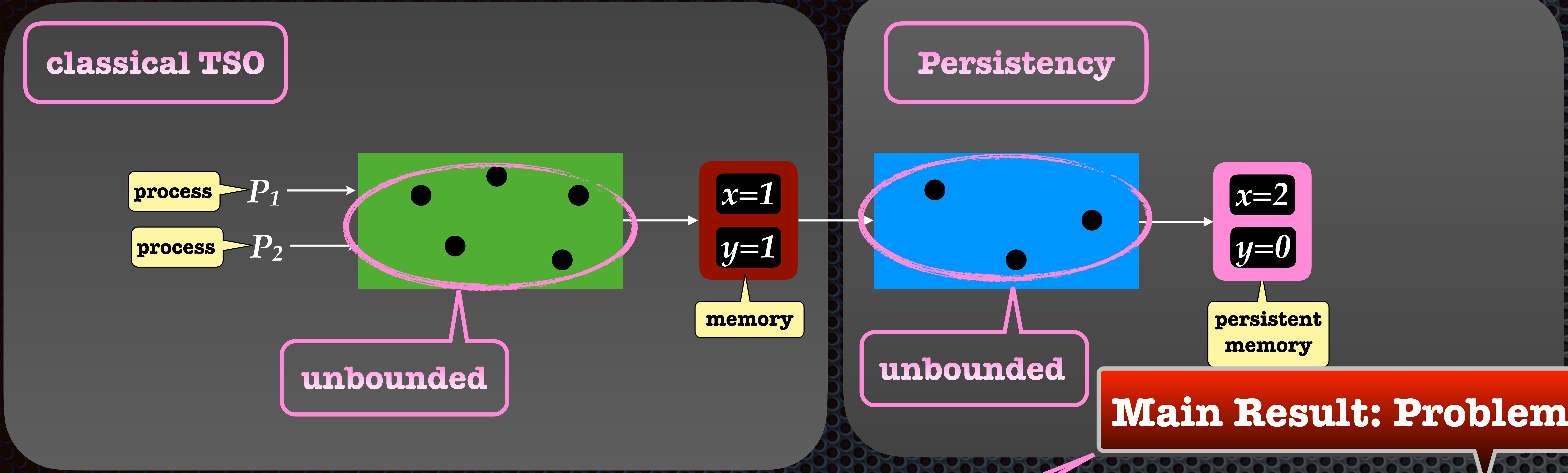
challenges

unbounded data structures

adapting SC techniques

semantics **decidability**

Persistent TSO = Classical TSO + Persistency



Main Result: Problem Decidable

challenges

Reachability Problem:

assertion ϕ

will the program violate ϕ?

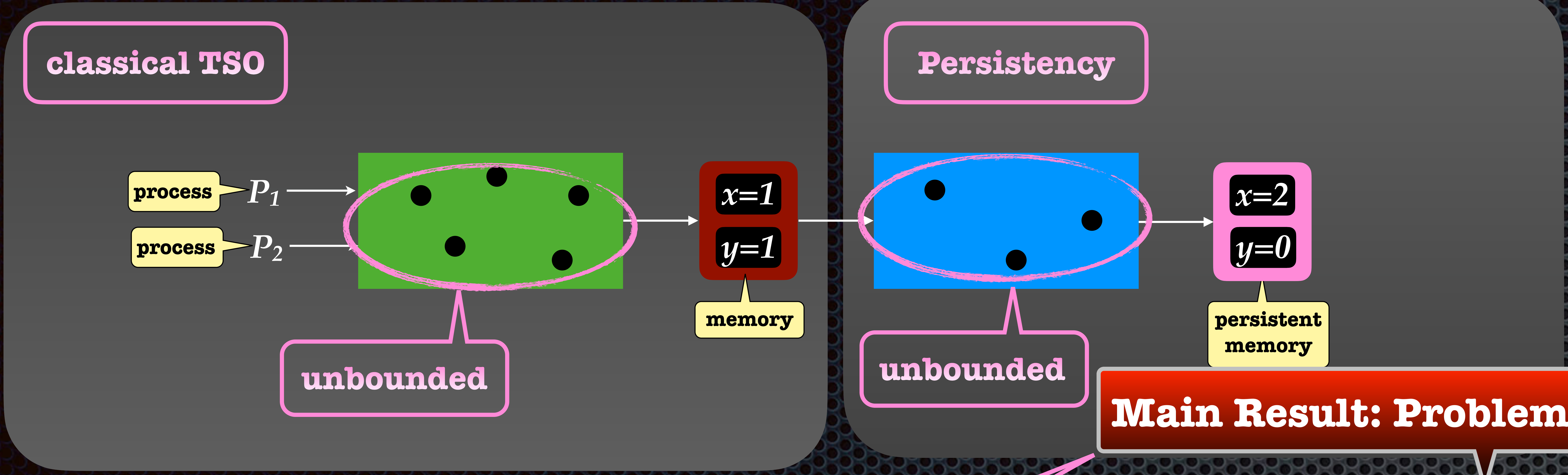
unbounded
data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



Main Result: Problem Decidable

challenges

Reachability Problem:

assertion ϕ

will the program violate ϕ?

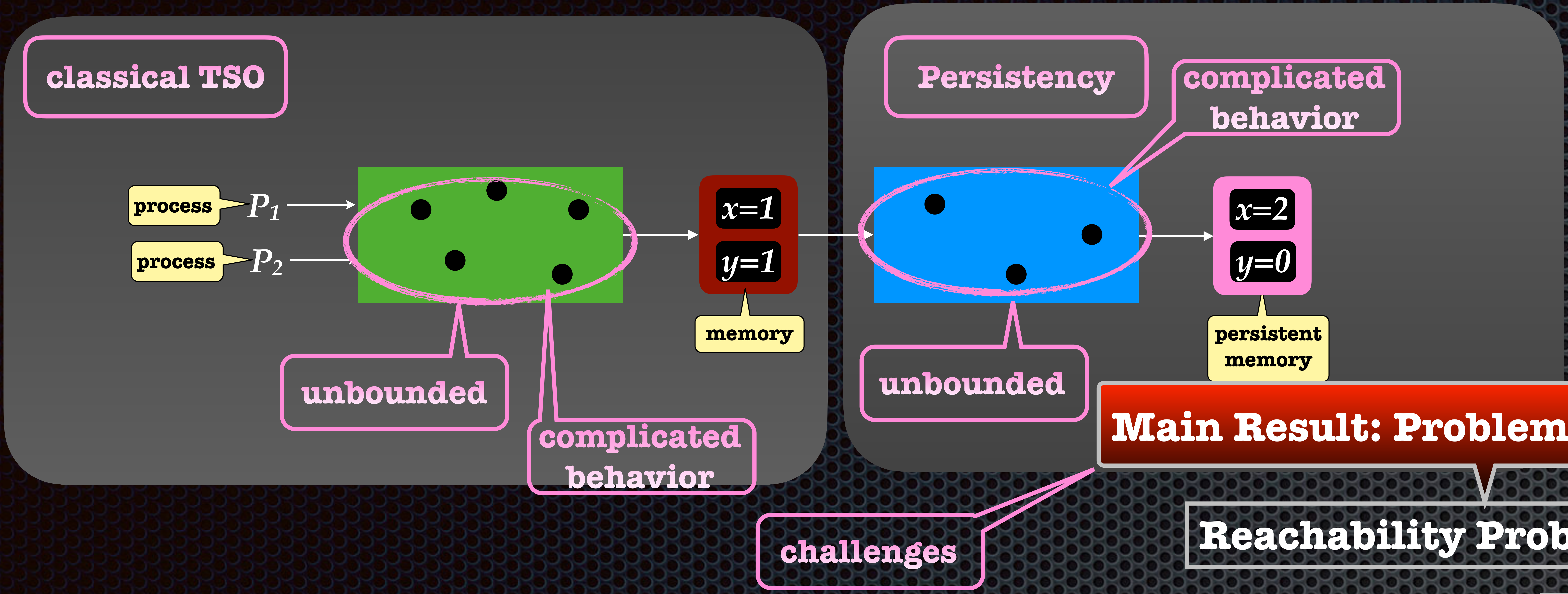
unbounded data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



Main Result: Problem Decidable

Reachability Problem:

assertion ϕ

will the program violate ϕ?

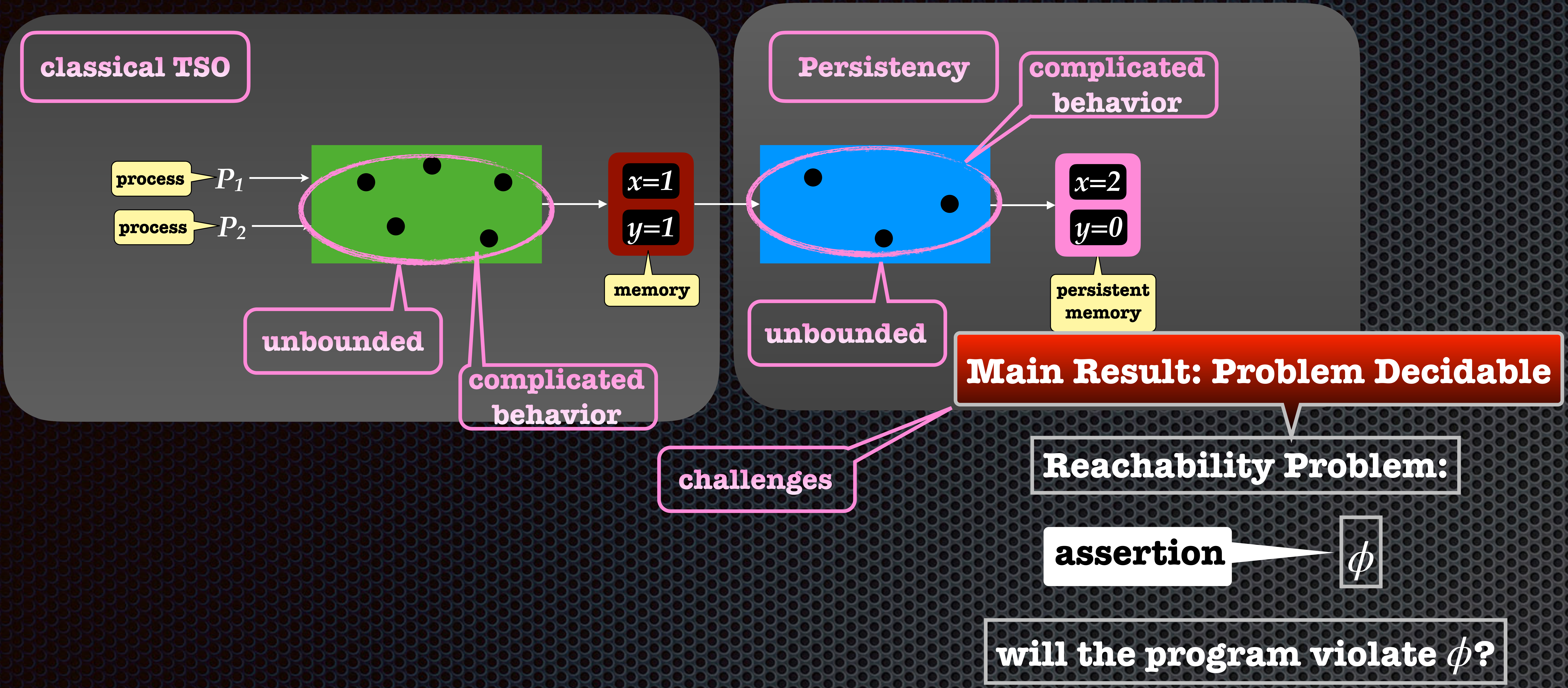
unbounded data structures

adapting SC techniques

semantics

decidability

Persistent TSO = Classical TSO + Persistency



classical TSO

process P_1

process P_2

unbounded

complicated behavior

$x=1$

$y=1$

memory

Persistency

complicated behavior

unbounded

$x=2$

$y=0$

persistent memory

challenges

Main Result: Problem Decidable

Reachability Problem:

assertion ϕ

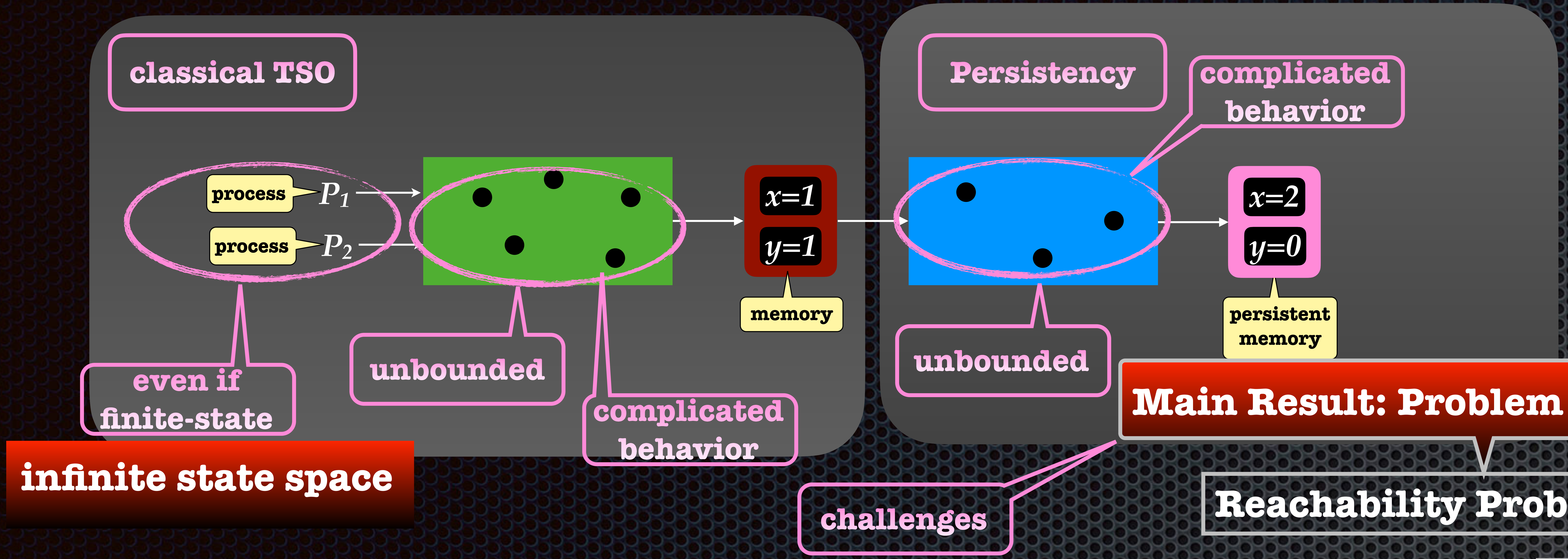
will the program violate ϕ?

unbounded data structures

adapting SC techniques

semantics **decidability**

Persistent TSO = Classical TSO + Persistency



classical TSO

process P_1

process P_2

even if finite-state

unbounded

complicated behavior

$x=1$

$y=1$

memory

infinite state space

Persistency

complicated behavior

$x=2$

$y=0$

persistent memory

unbounded

challenges

Main Result: Problem Decidable

Reachability Problem:

assertion ϕ

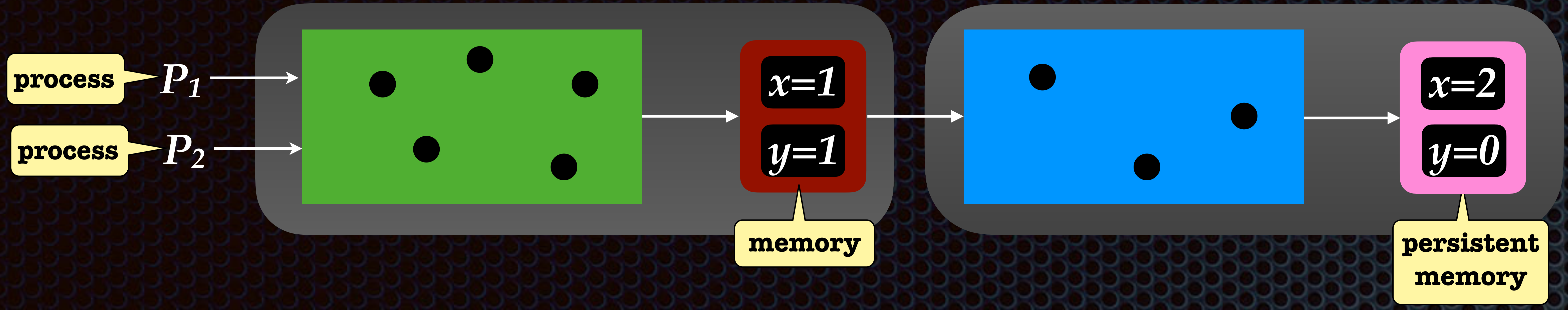
will the program violate ϕ?

unbounded
data structures

adapting SC techniques

semantics

decidability



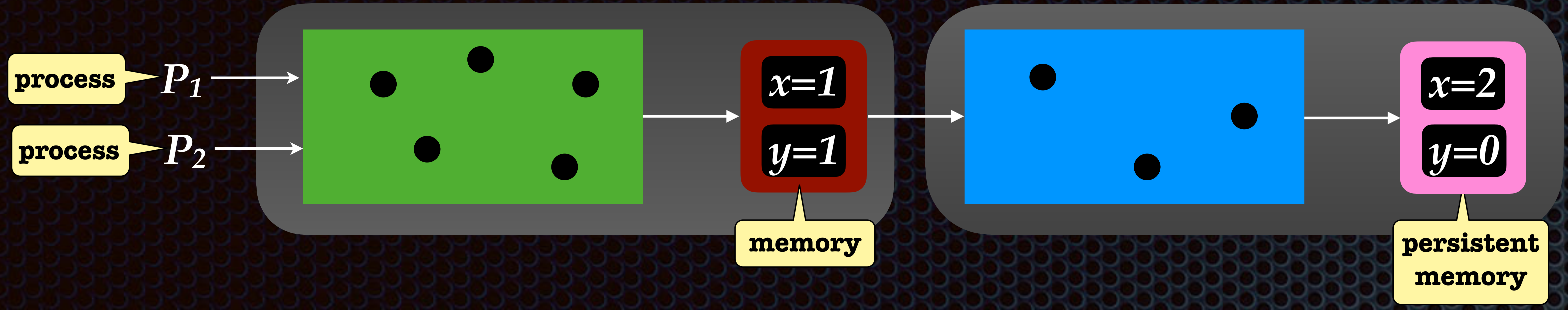
Persistent TSO

unbounded data structures

adapting SC techniques

semantics

decidability



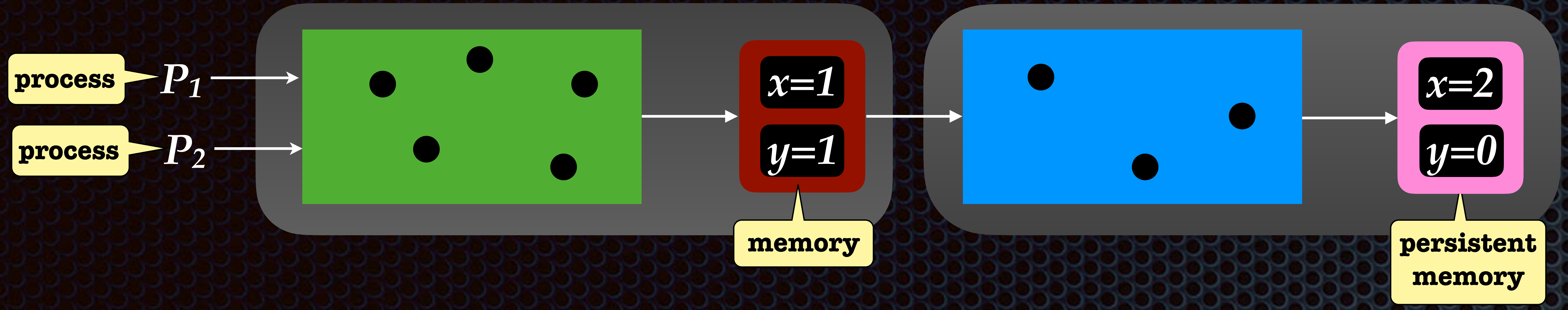
Persistent TSO

unbounded data structures

adapting SC techniques

semantics

decidability



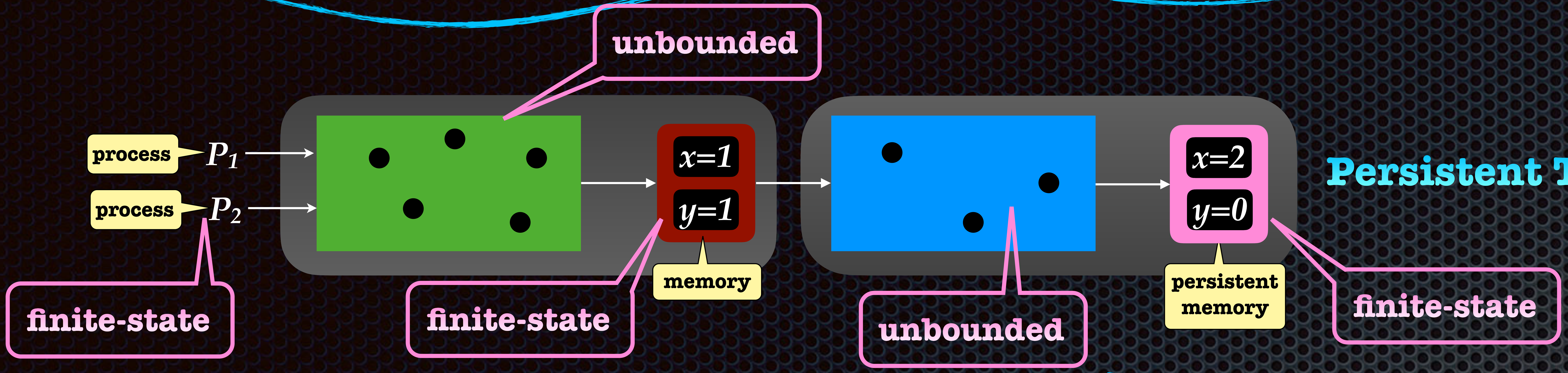
Persistent TSO

unbounded
data structures

adapting SC techniques

semantics

decidability



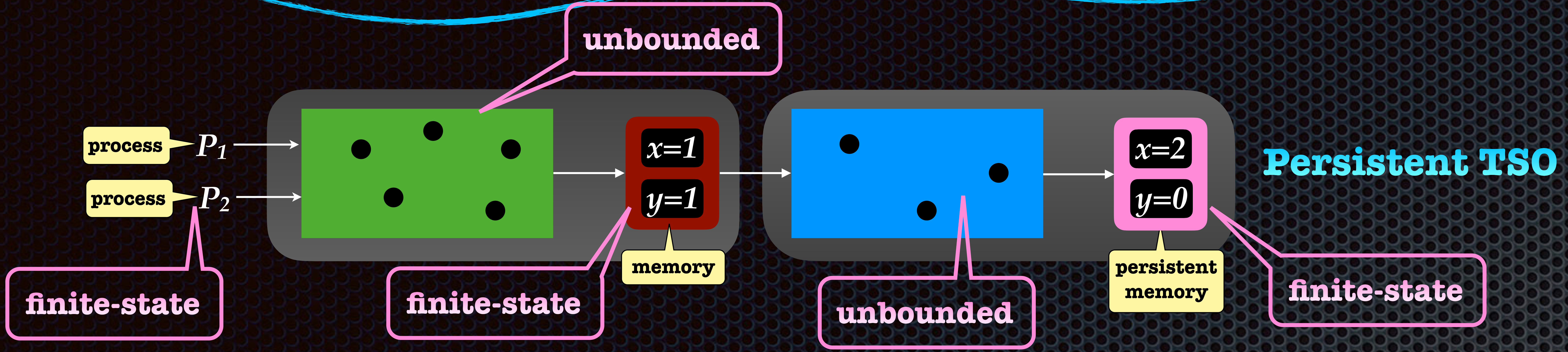
Persistent TSO

unbounded
data structures

adapting SC techniques

semantics

decidability

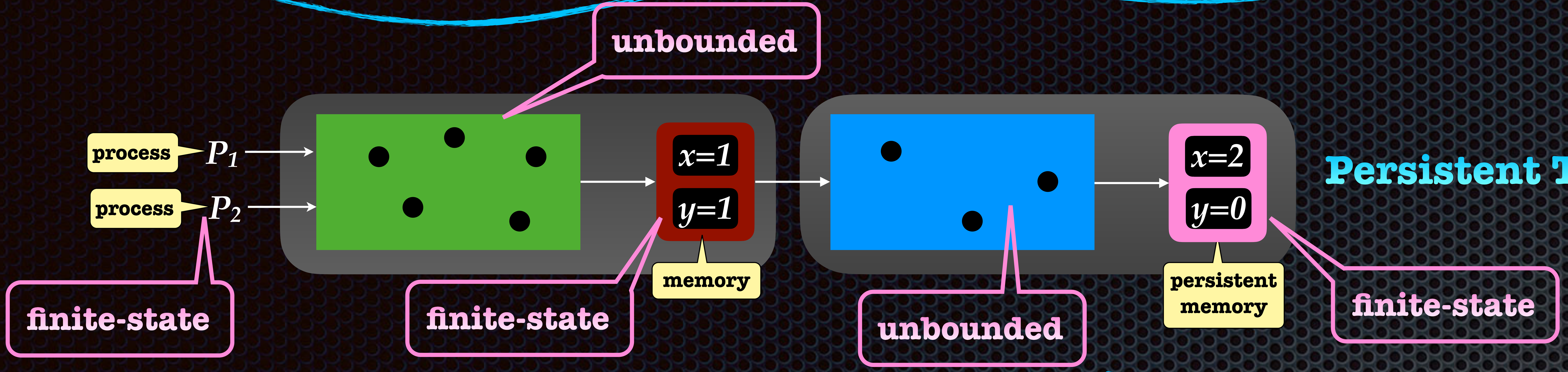


unbounded
data structures

adapting SC techniques

semantics

decidability



Persistent TSO

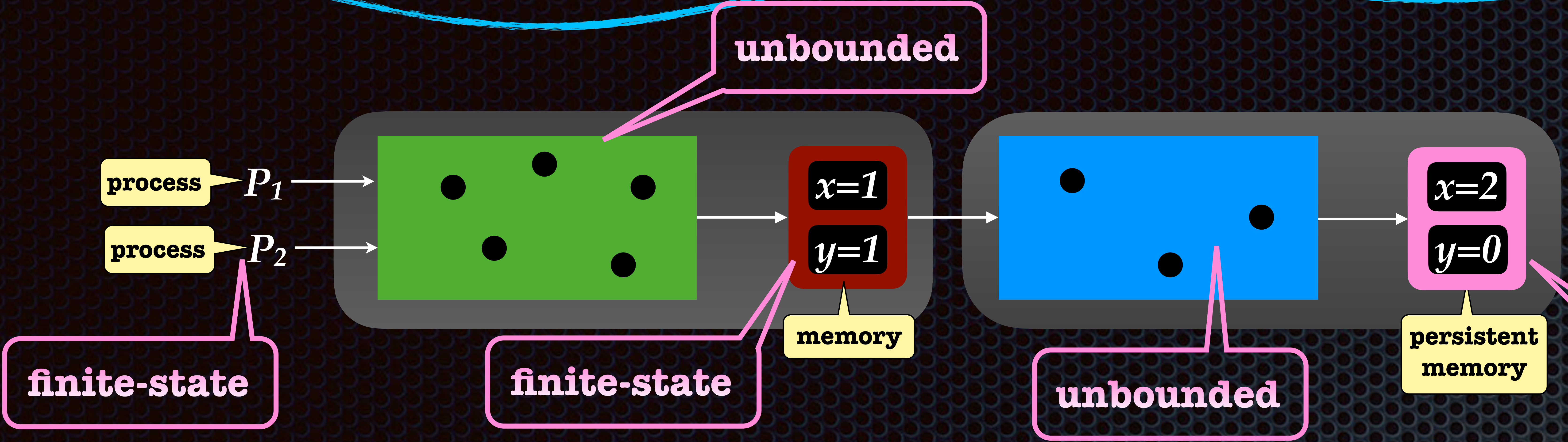
Well-Structured Systems

unbounded data structures

adapting SC techniques

semantics

decidability



Persistent TSO

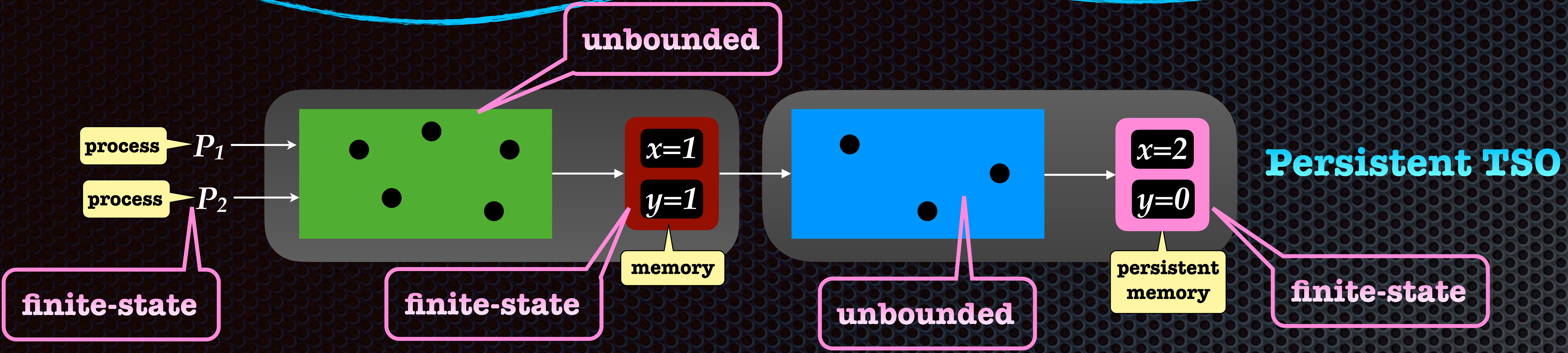
Well-Structured Systems

unbounded
data structures

adapting SC techniques

semantics

decidability



Persistent TSO

Well-Structured Systems

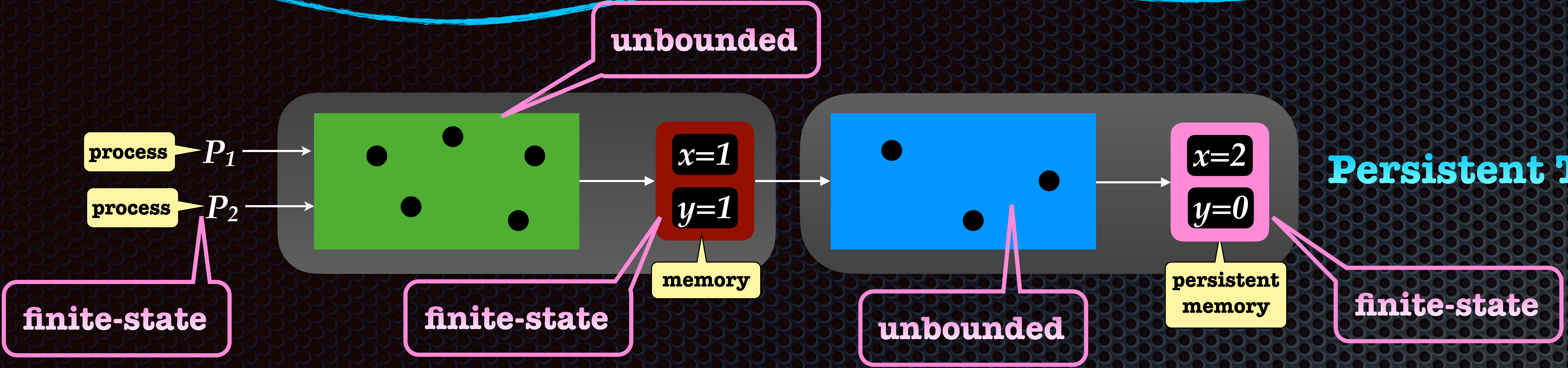
- unbounded
- FIFO
- monotone

unbounded data structures

adapting SC techniques

semantics

decidability



Persistent TSO

Well-Structured Systems

finite-state part

channel part

- unbounded
- FIFO
- monotone

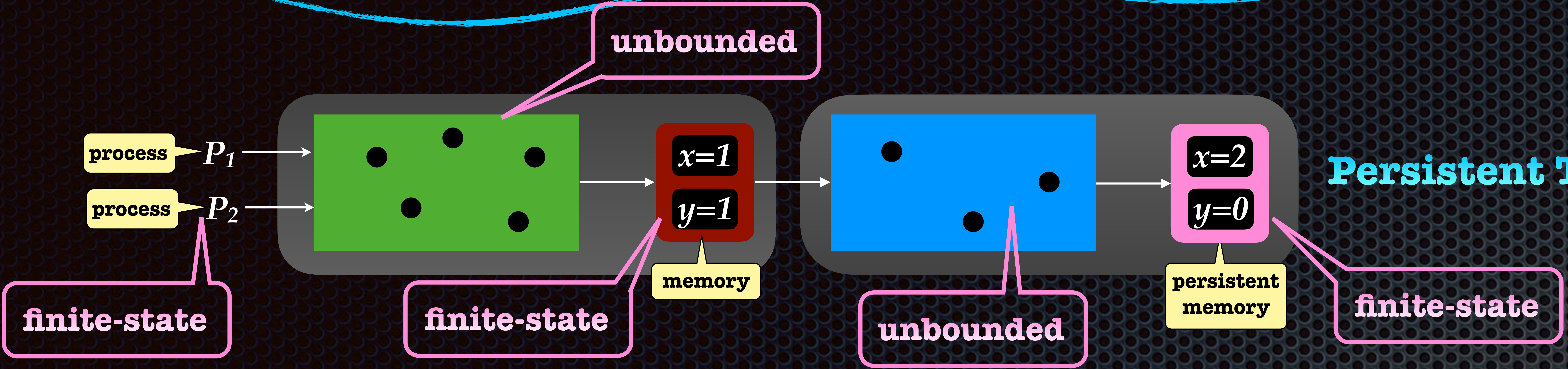
Reachability Problem Decidable

unbounded
data structures

adapting SC techniques

semantics

decidability


unbounded

process P_1

process P_2

$x=1$

$y=1$

memory

$x=2$

$y=0$

persistent memory

finite-state

finite-state

unbounded

finite-state

Persistent TSO

Well-Structured Systems


finite-state part

channel part

- unbounded
- FIFO
- monotone

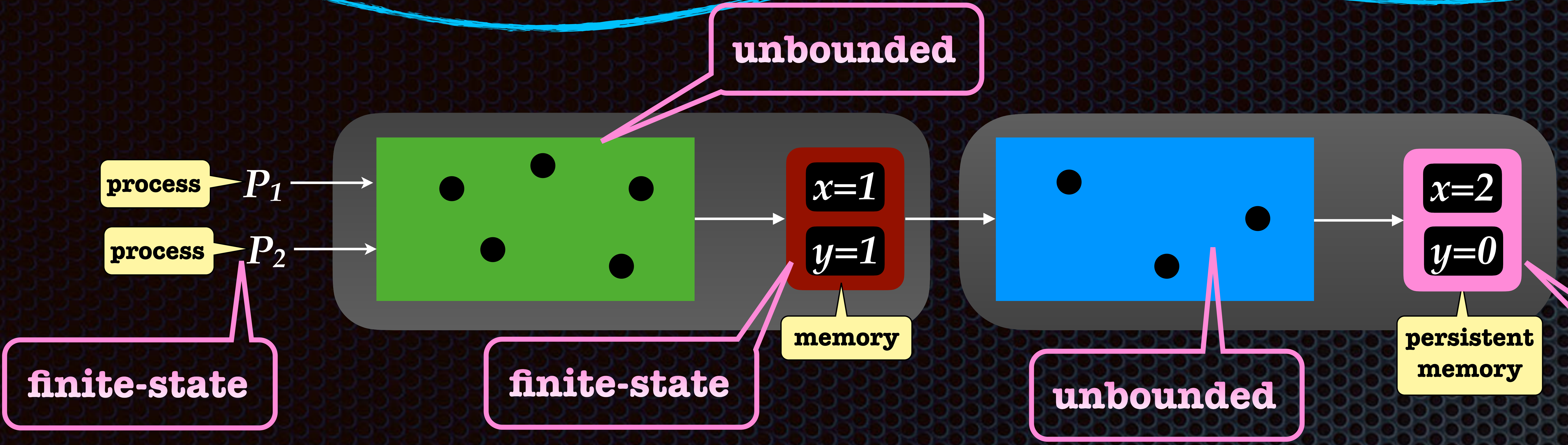
Reachability Problem Decidable

unbounded data structures

adapting SC techniques

semantics

decidability



Persistent TSO

Well-Structured Systems

- unbounded
- FIFO
- monotone

Reachability Problem Decidable

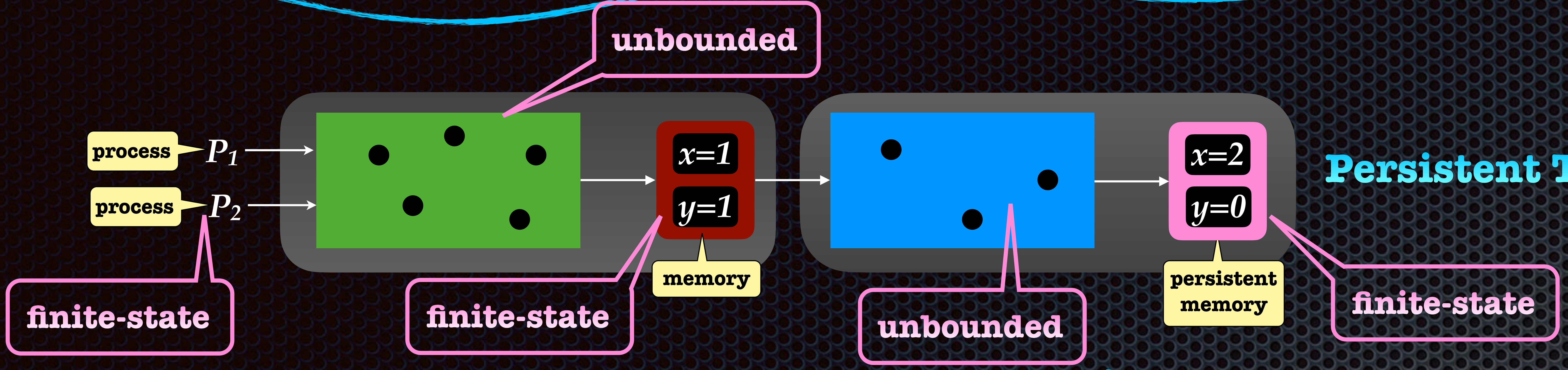
unbounded data structures

adapting SC techniques

semantics

decidability

Persistent TSO



Well-Structured Systems

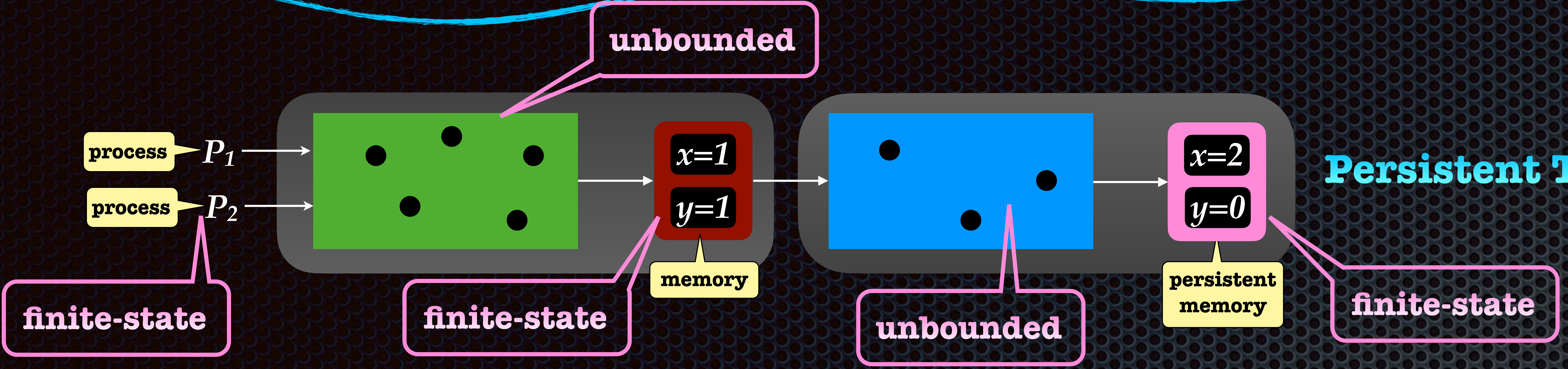
Monotonicity

unbounded data structures

adapting SC techniques

semantics

decidability



unbounded

process P_1

process P_2

$x=1$

$y=1$

memory

$x=2$

$y=0$

persistent memory

finite-state

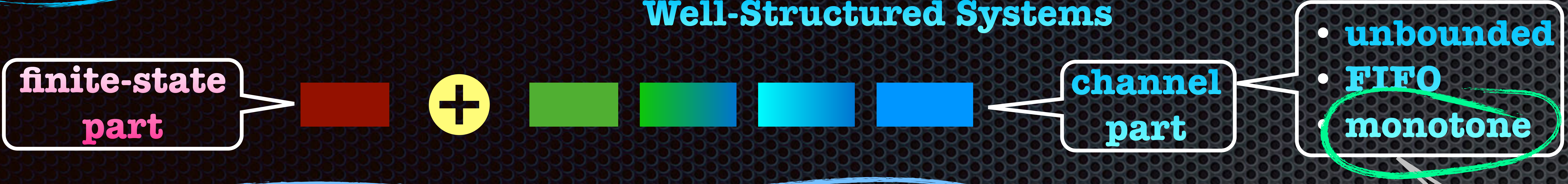
finite-state

unbounded

finite-state

Persistent TSO

Well-Structured Systems



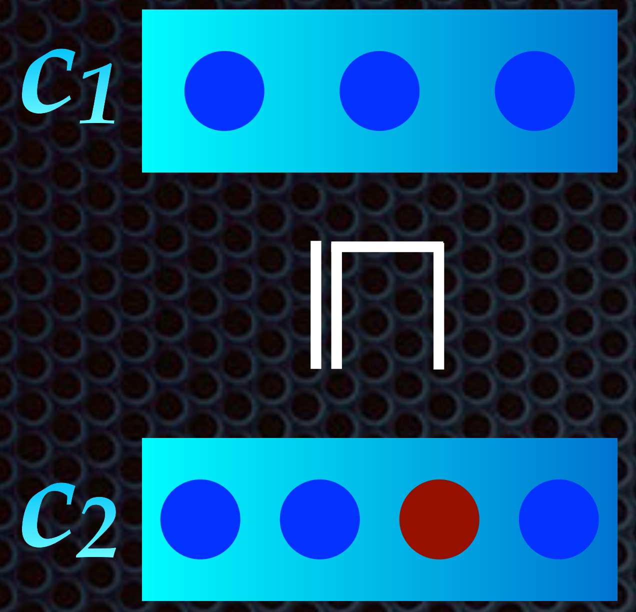
finite-state part

channel part

- unbounded
- FIFO
- monotone

Reachability Problem Decidable

Monotonicity

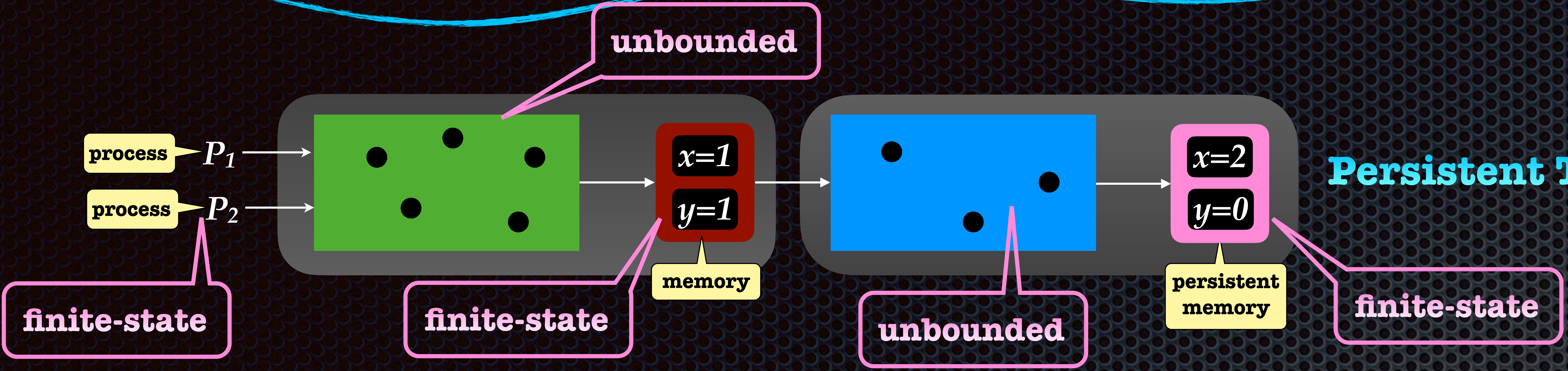


unbounded
data structures

adapting SC techniques

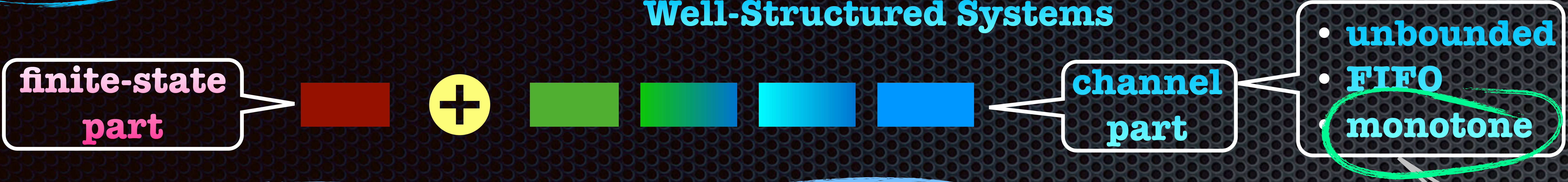
semantics

decidability



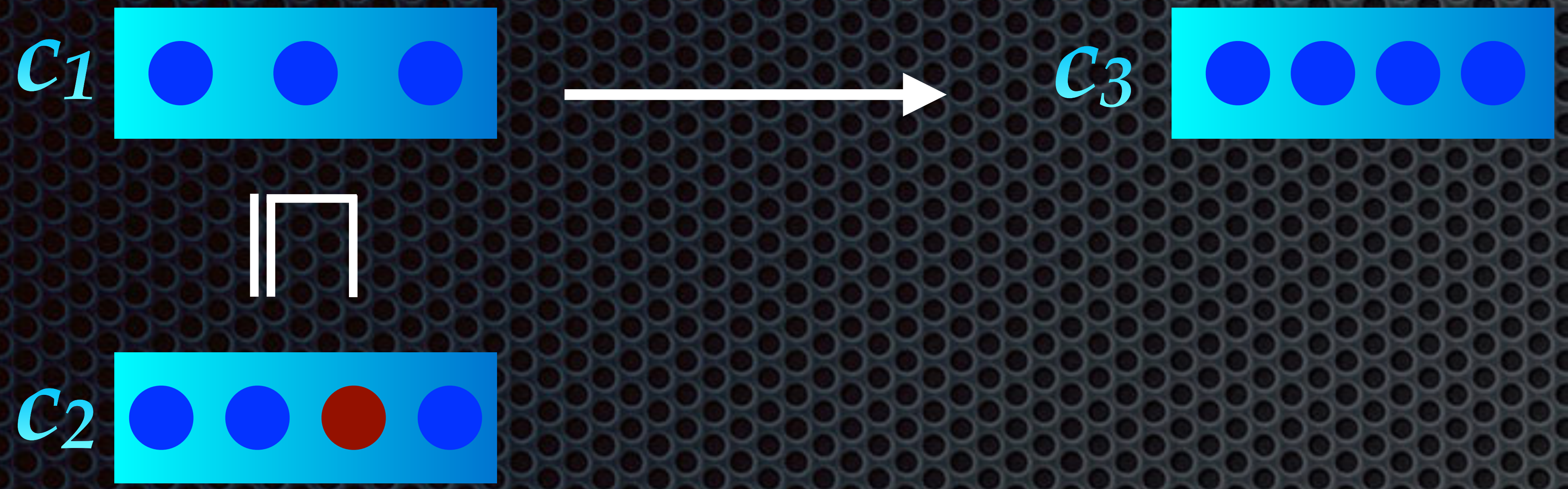
Persistent TSO

Well-Structured Systems

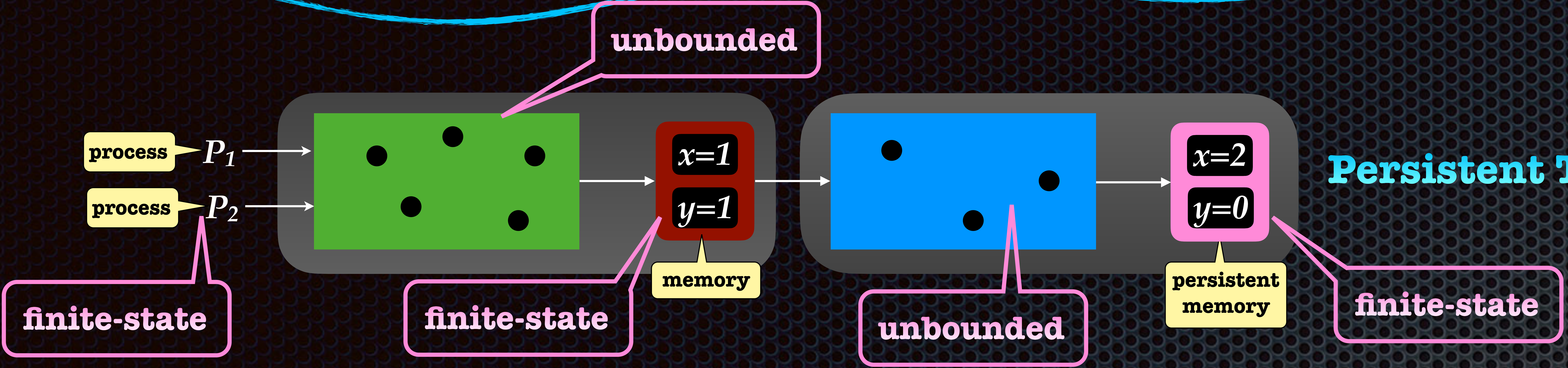


Reachability
Problem
Decidable

Monotonicity

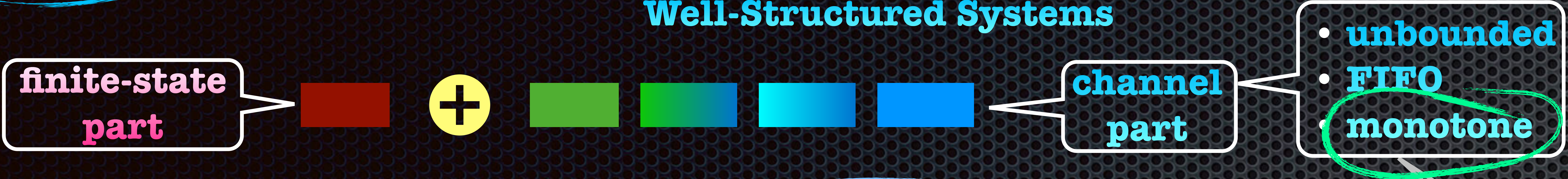


unbounded data structures **adapting SC techniques** **semantics** **decidability**



Persistent TSO

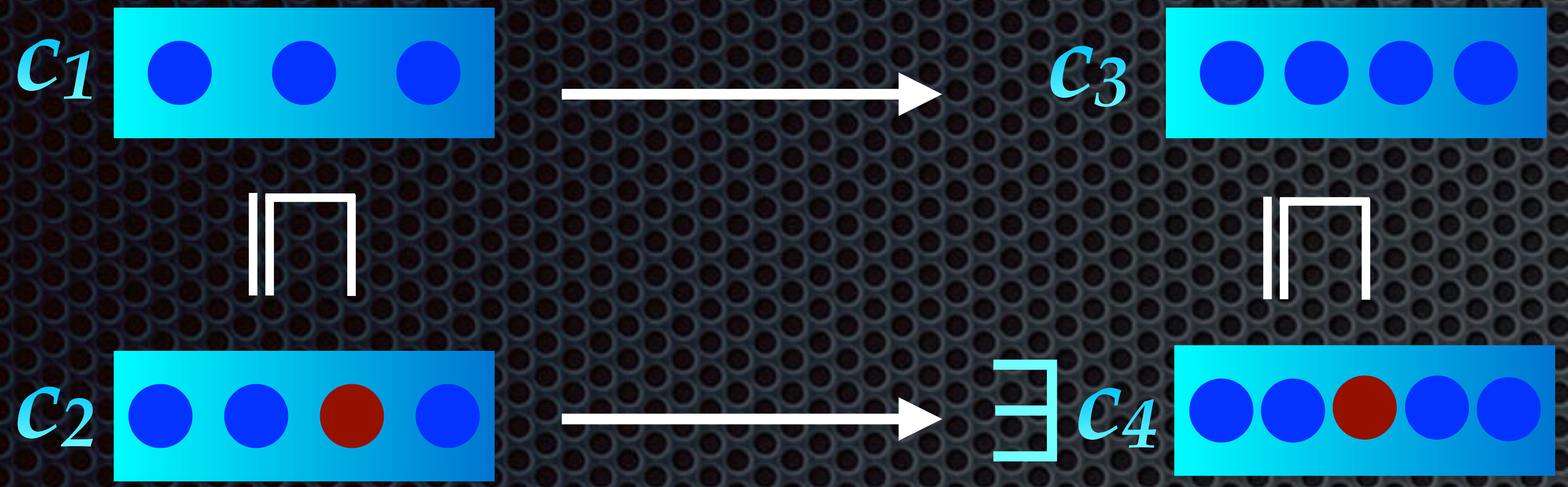
Well-Structured Systems



- unbounded
- FIFO
- monotone

Reachability Problem Decidable

Monotonicity

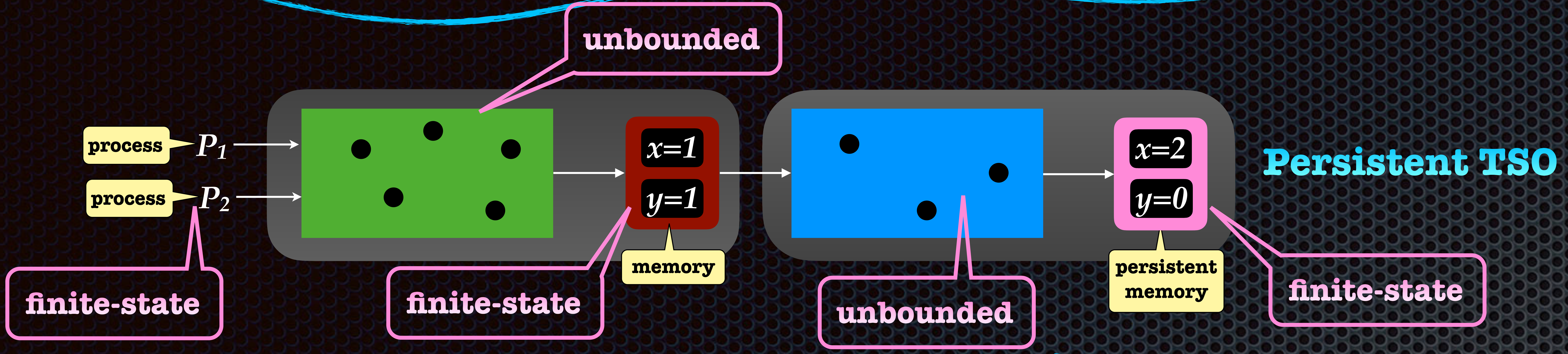


unbounded data structures

adapting SC techniques

semantics

decidability



Persistent TSO

process P_1

process P_2

unbounded

$x=1$

$y=1$

memory

$x=2$

$y=0$

persistent memory

finite-state

finite-state

unbounded

finite-state

Well-Structured Systems

finite-state part

+

channel part

- unbounded
- FIFO
- monotone

Reachability Problem Decidable

Monotonicity

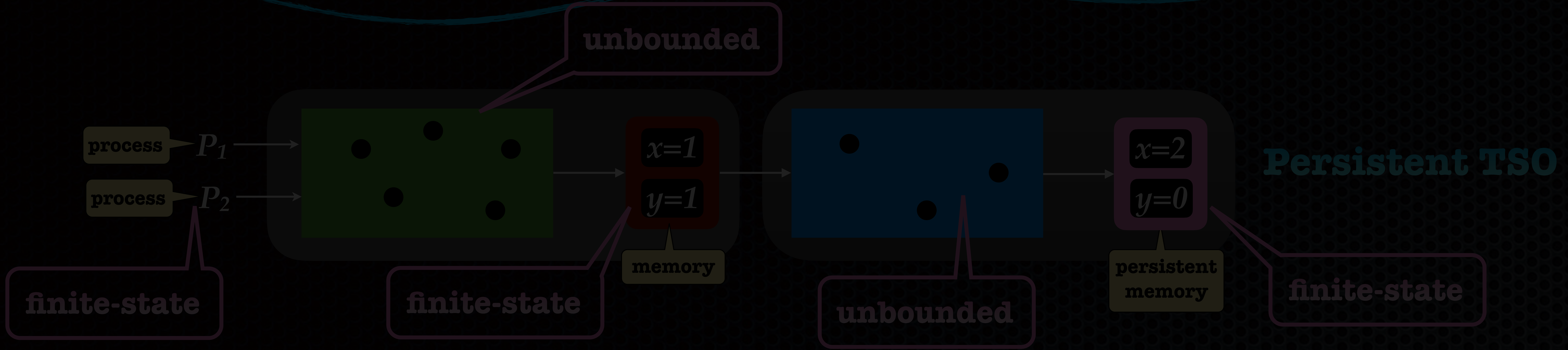
simulation

$c_1 \longrightarrow c_3$

$\sqsubseteq \qquad \sqsubseteq$

$c_2 \longrightarrow \exists c_4$

unbounded data structures **adapting SC techniques** **semantics** **decidability**



Well-Structured Systems

- **unbounded**
- **FIFO**
- **monotone**

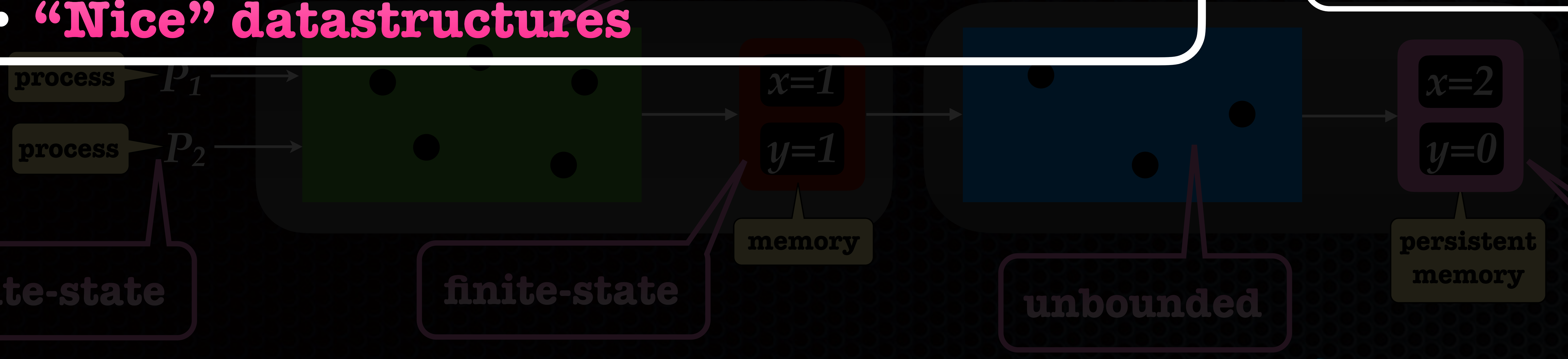
Monotonicity

Reachability Problem Decidable

unbounded data structures | **adapting SC techniques** | **semantics** | **decidability**

- **Equivelant to the Raad et al semantics**
- **Allows applying classical (SC) techniques**
- **“Nice” datastructures**

this work:
new semantics



Persistent TSO

Well-Structured Systems

- **unbounded**
- **FIFO**
- **monotone**

Reachability Problem Decidable

Monotonicity

unbounded data structures

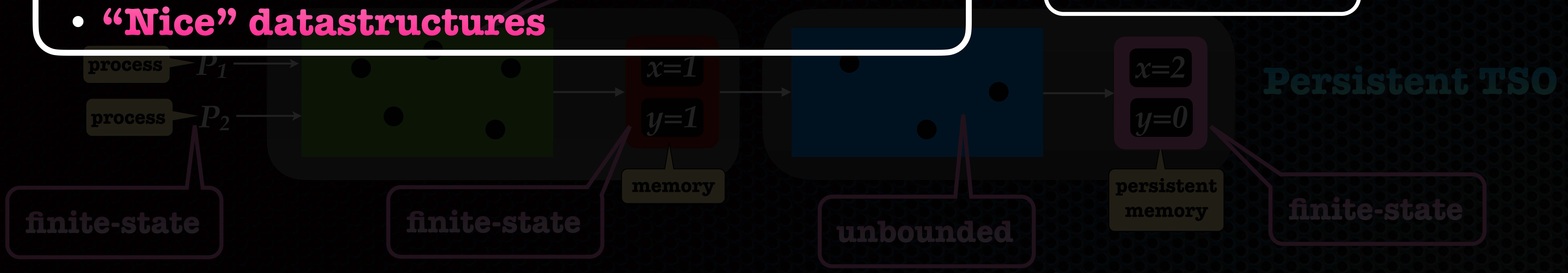
adapting SC techniques

semantics

decidability

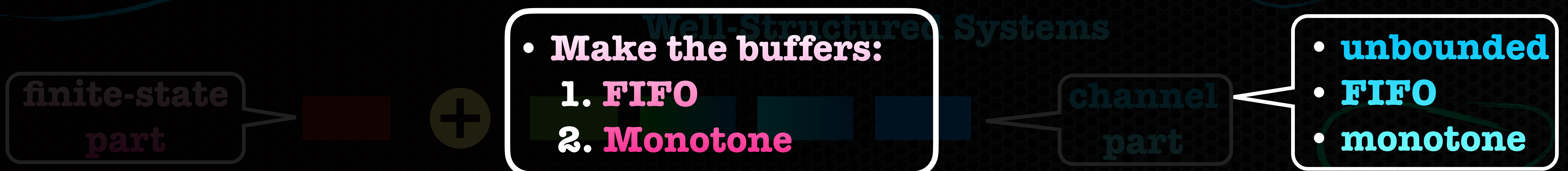
- **Equivelant to the Raad et al semantics**
- **Allows applying classical (SC) techniques**
- **“Nice” datastructures**

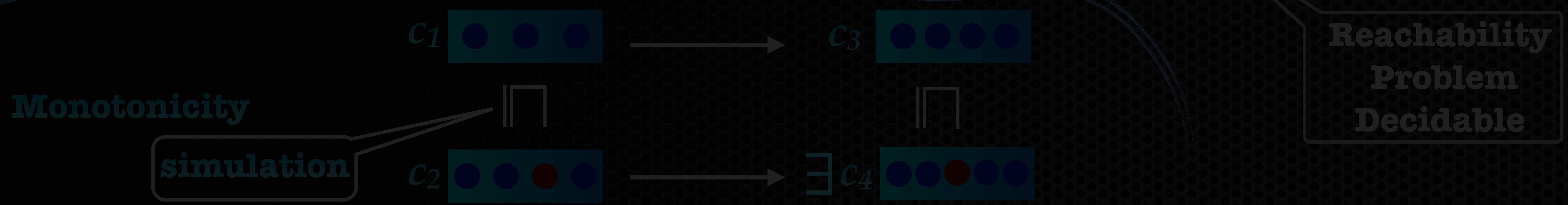
this work:
new semantics



- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**

- **unbounded**
- **FIFO**
- **monotone**

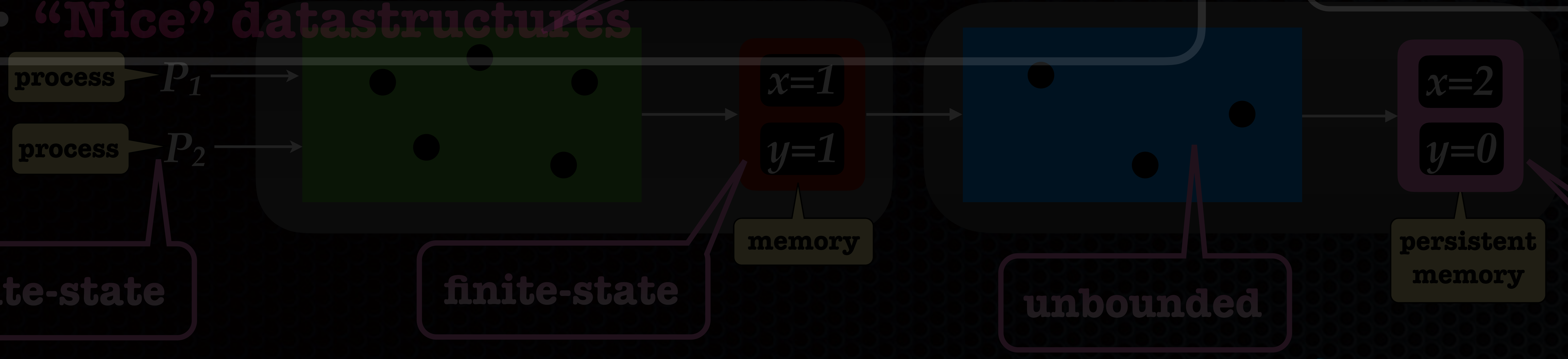




unbounded data structures **adapting SC techniques** **semantics** **decidability**

- Equivelant to the Raad et al semantics
- Allows applying classical (SC) techniques
- "Nice" datastructures

this work:
new semantics



Persistent TSO

Well-Structured Systems

- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**

- unbounded
- FIFO
- monotone

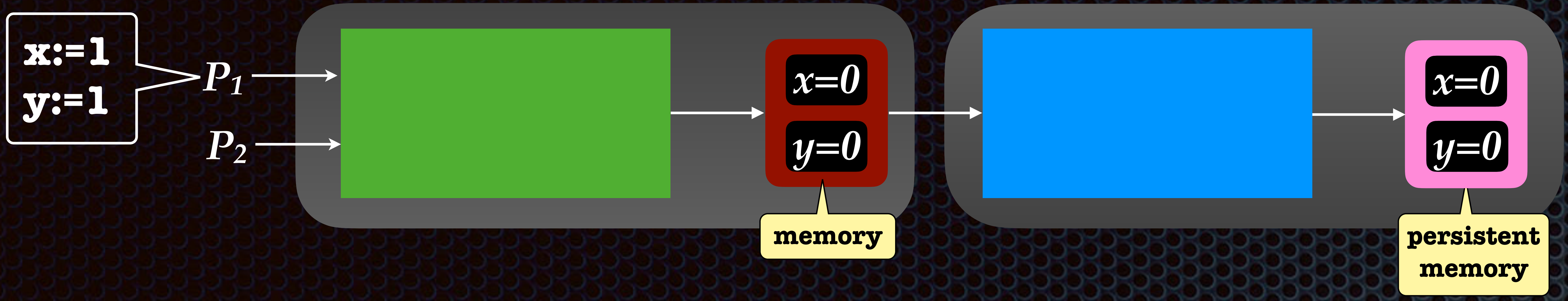
Reachability Problem Decidable

Monotonicity

- Make the buffers:
  1. FIFO
  2. Monotone

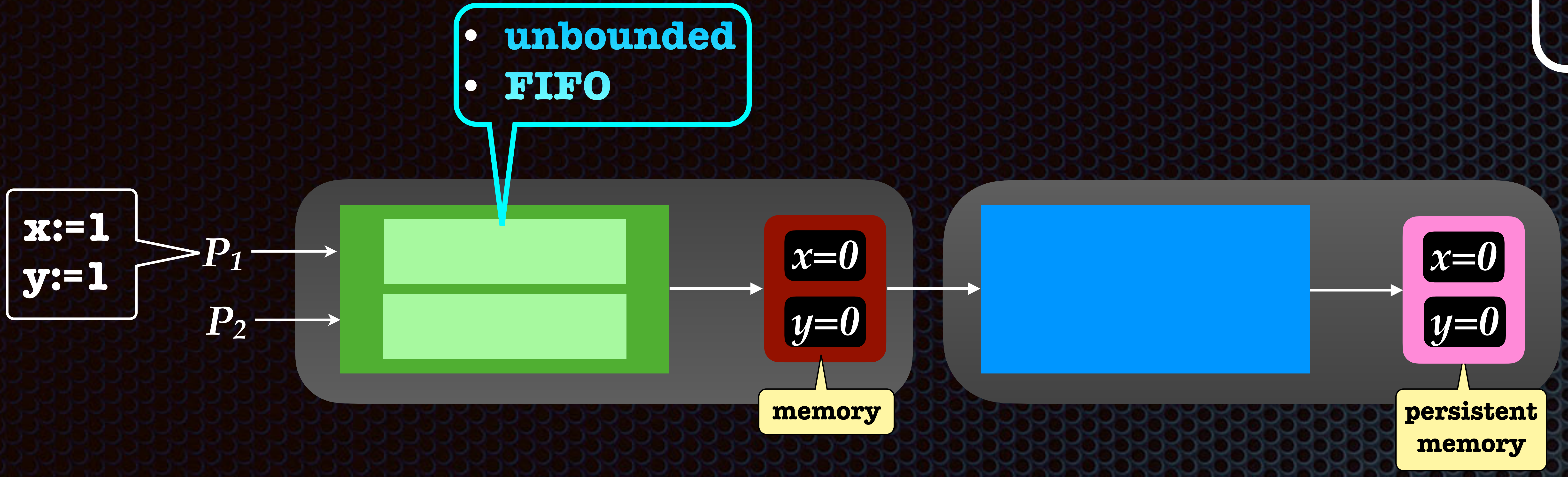
- Make the buffers:
  1. FIFO
  2. Monotone

- Make the buffers:
  1. FIFO
  2. Monotone



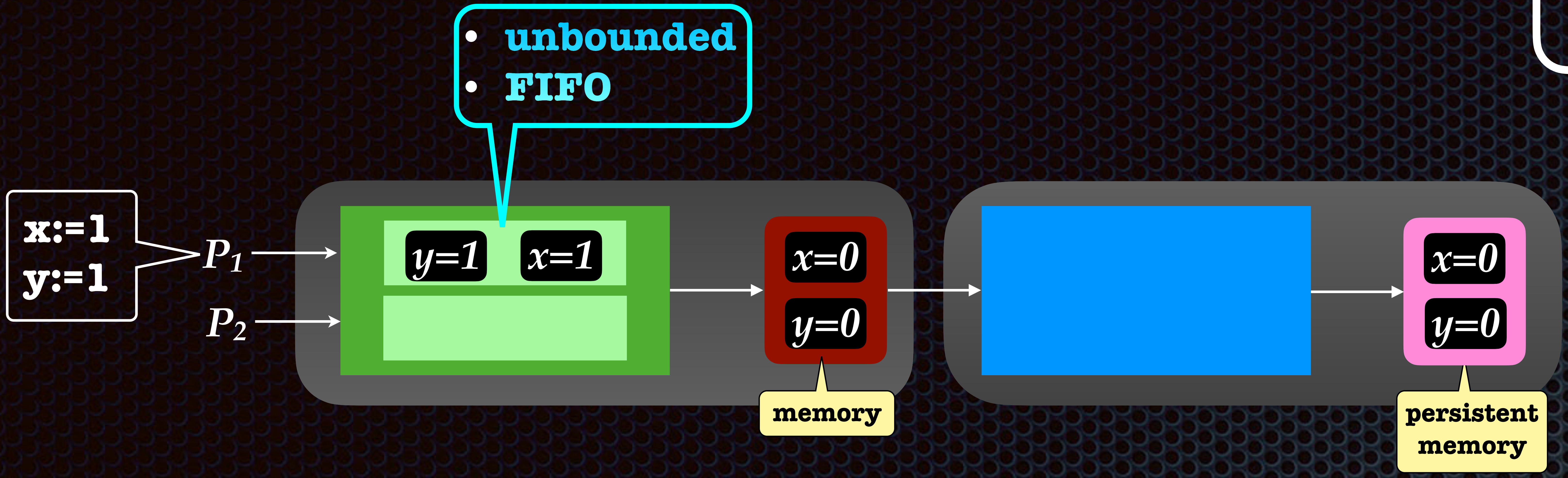
Persistent TSO

- Make the buffers:
  1. FIFO
  2. Monotone



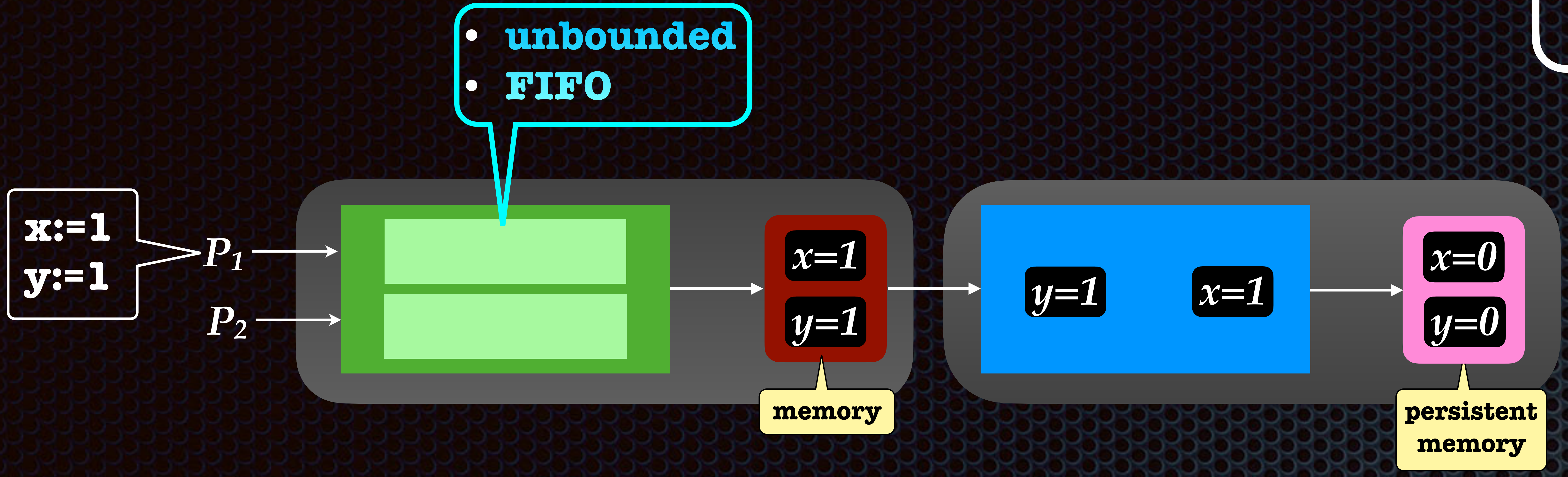
Persistent TSO

- Make the buffers:
  1. FIFO
  2. Monotone



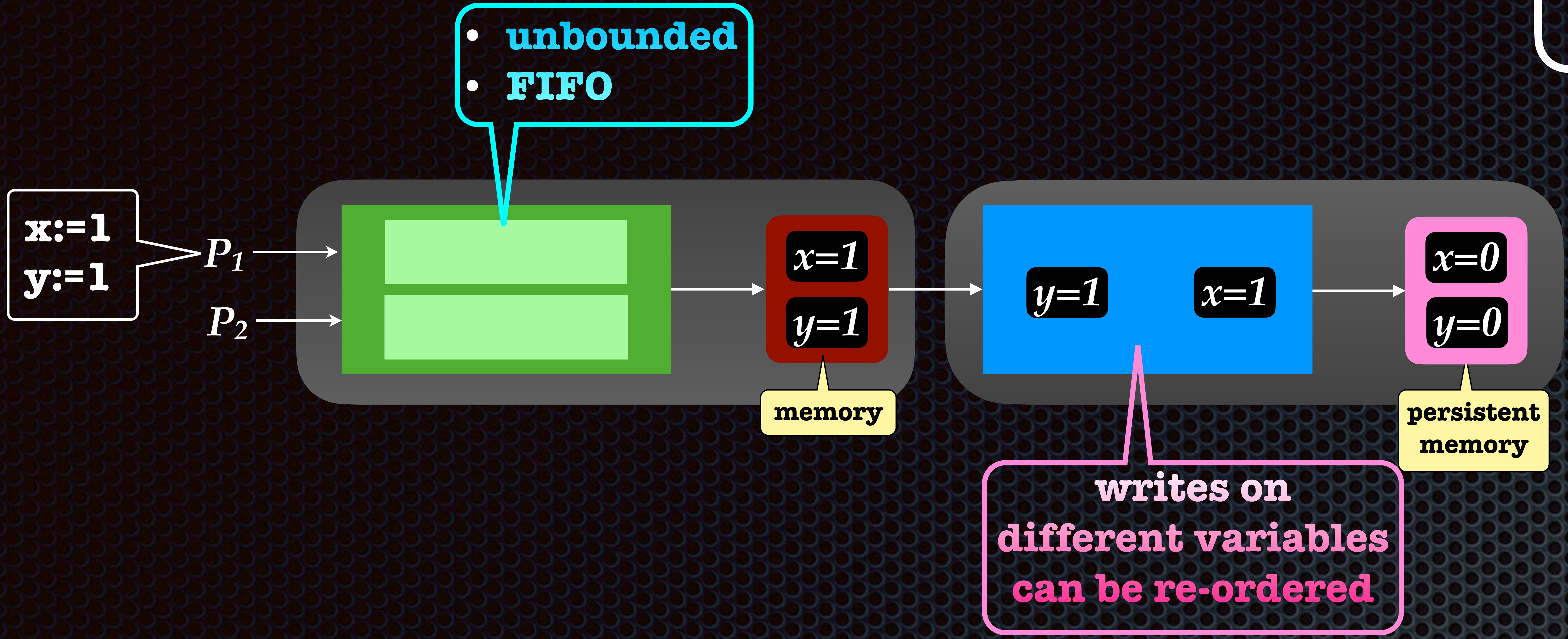
Persistent TSO

- Make the buffers:
  1. FIFO
  2. Monotone



Persistent TSO

- Make the buffers:
  1. FIFO
  2. Monotone

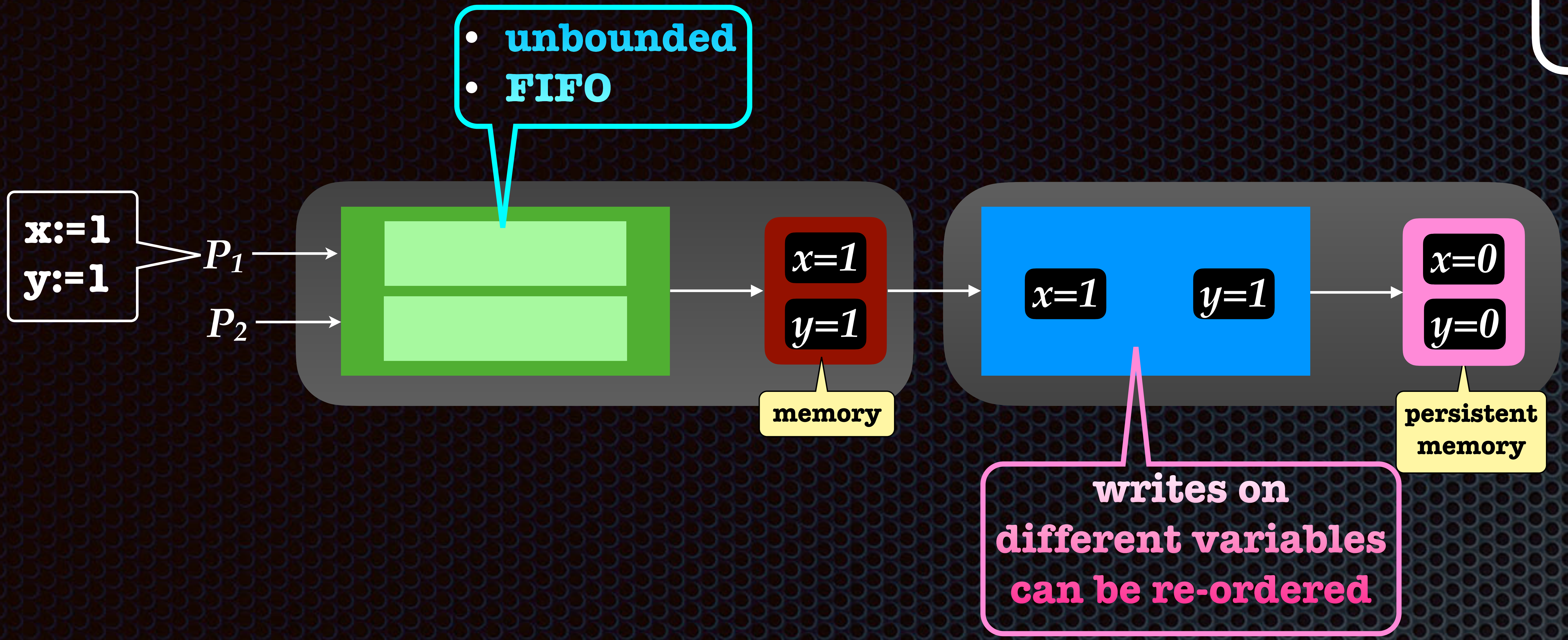


- unbounded
- FIFO

writes on different variables can be re-ordered

Persistent TSO

- **Make the buffers:**
  1. FIFO
  2. Monotone

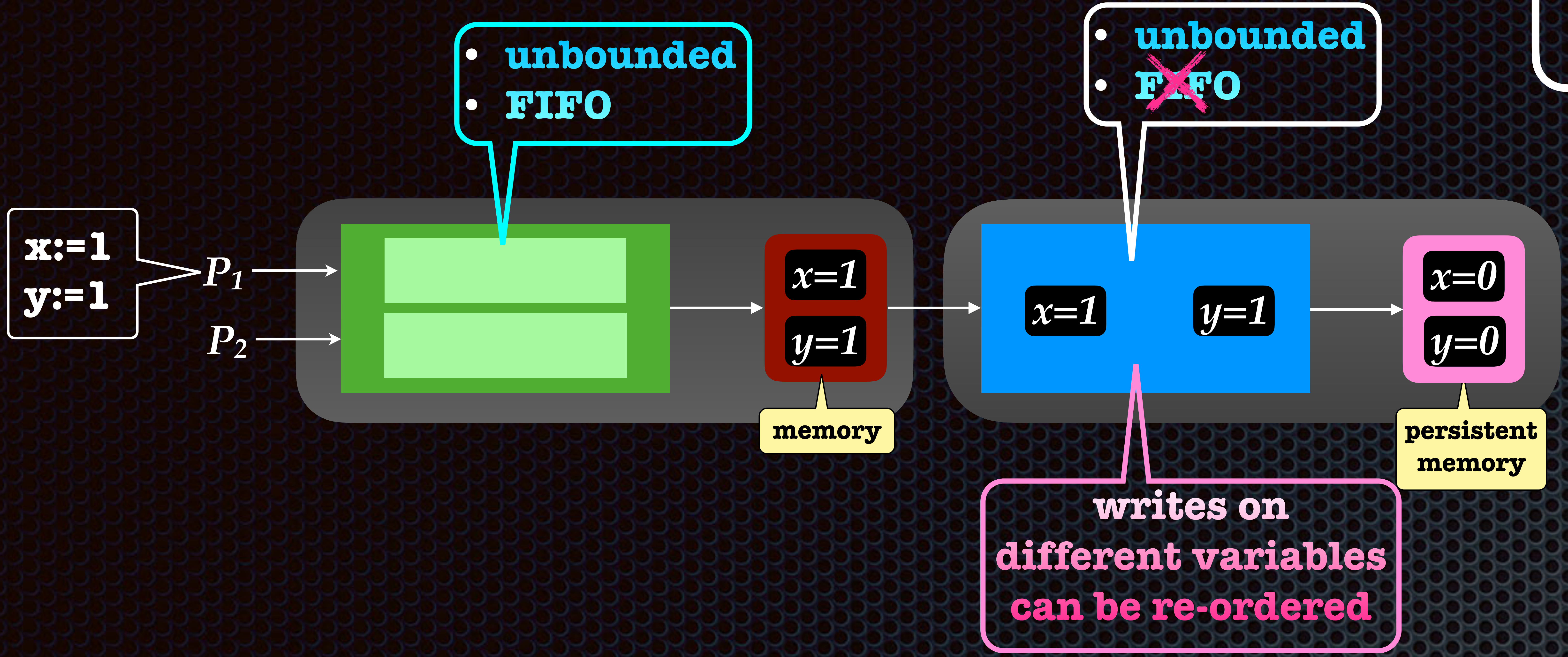


- unbounded
- FIFO

writes on different variables can be re-ordered

Persistent TSO

- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



x:=1
y:=1

P_1

P_2

- unbounded
- FIFO

$x=1$
$y=1$

memory

- unbounded
- F~~I~~FO

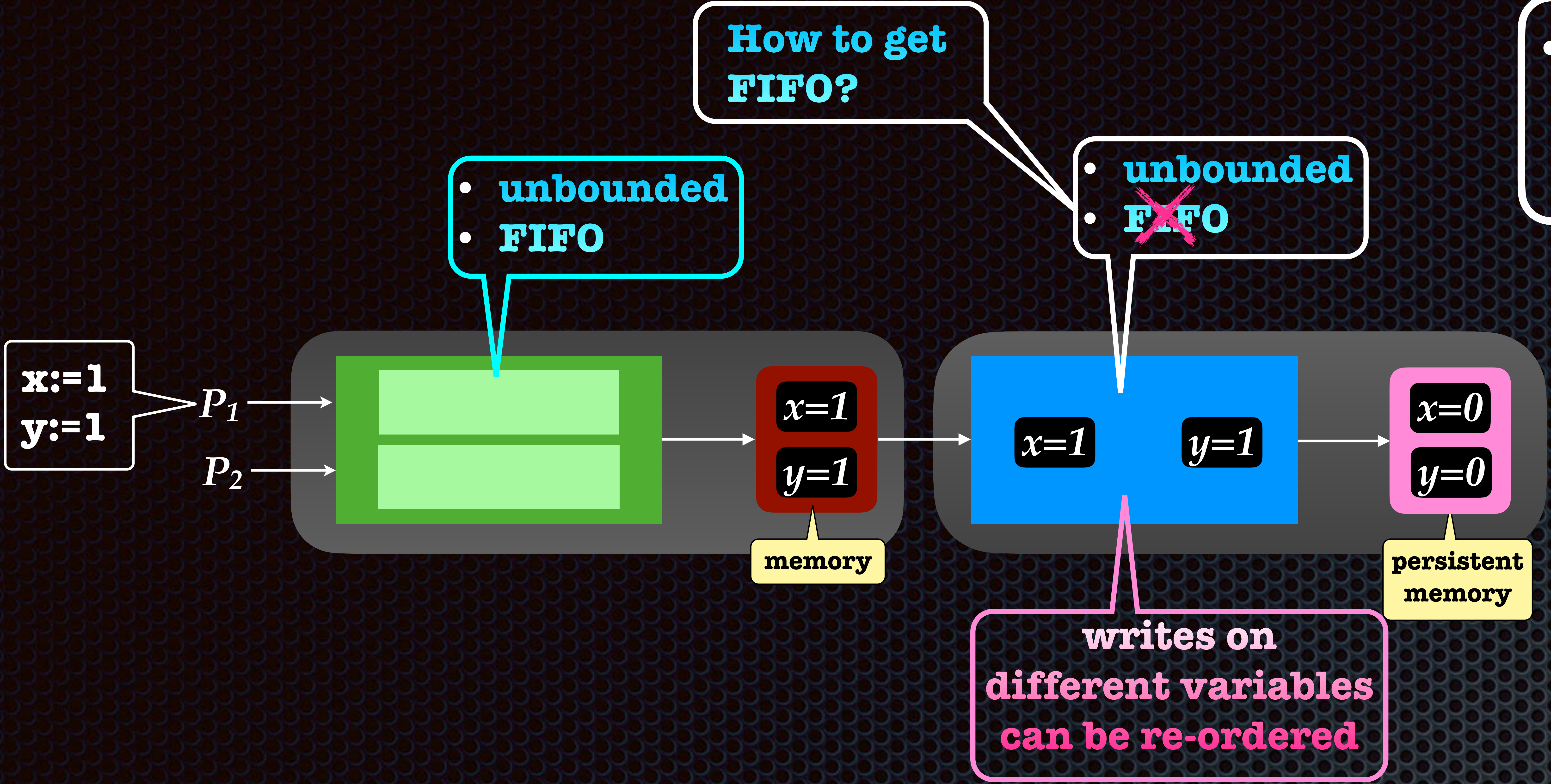
$x=1$ $y=1$

writes on different variables can be re-ordered

$x=0$
$y=0$

persistent memory

Persistent TSO



How to get FIFO?

- unbounded
- FIFO

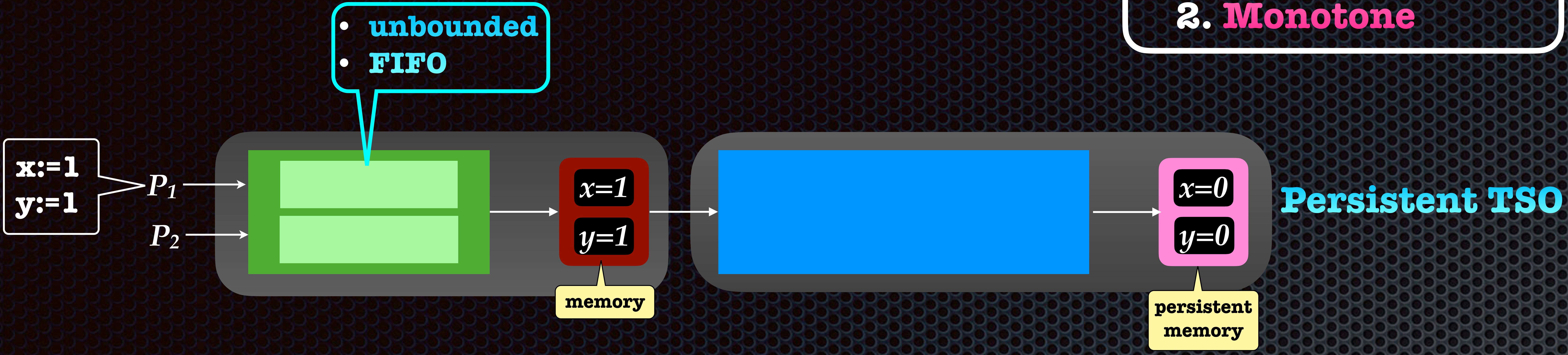
- unbounded
- F~~I~~FO

writes on different variables can be re-ordered

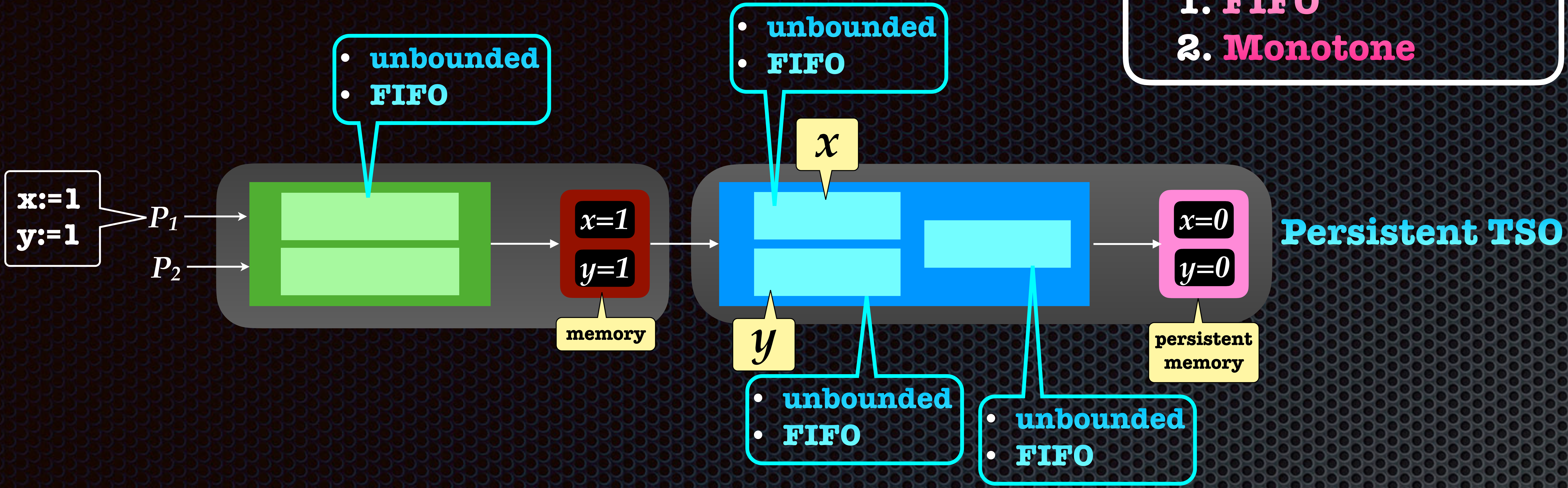
- Make the buffers:
  1. FIFO
  2. Monotone

Persistent TSO

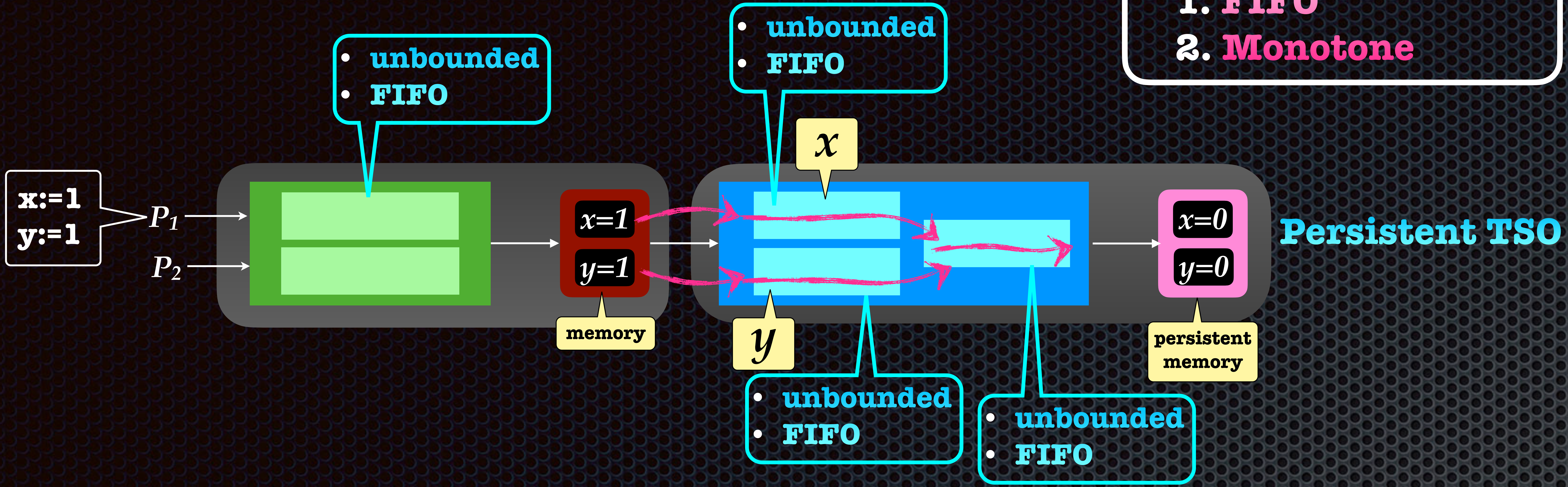
- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



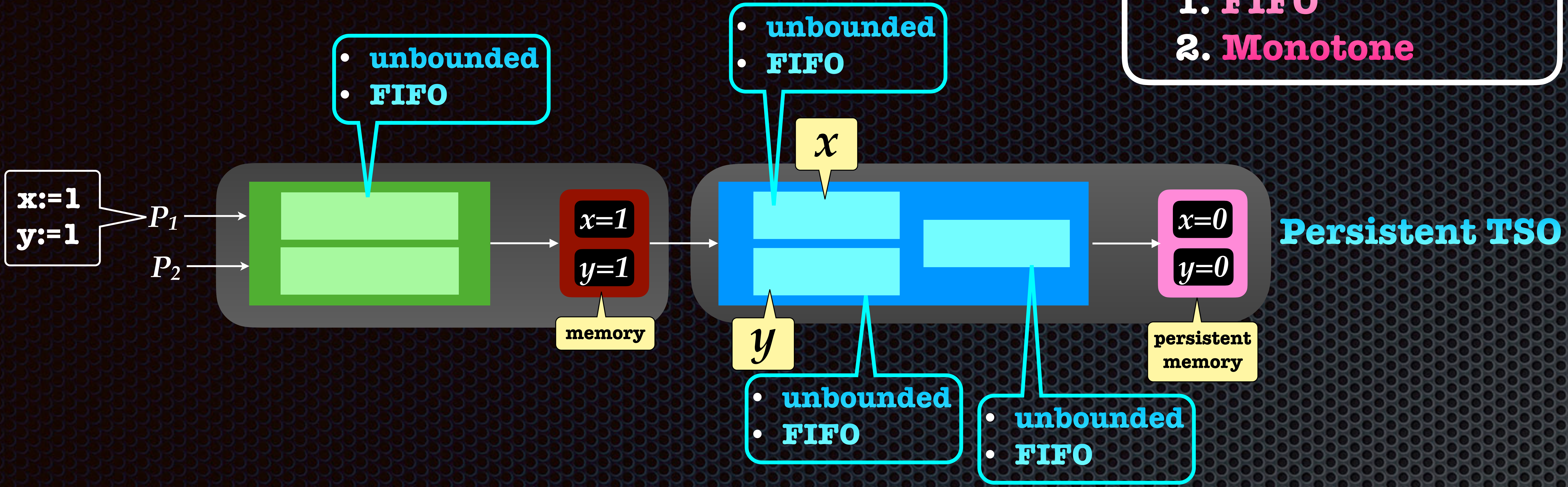
- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



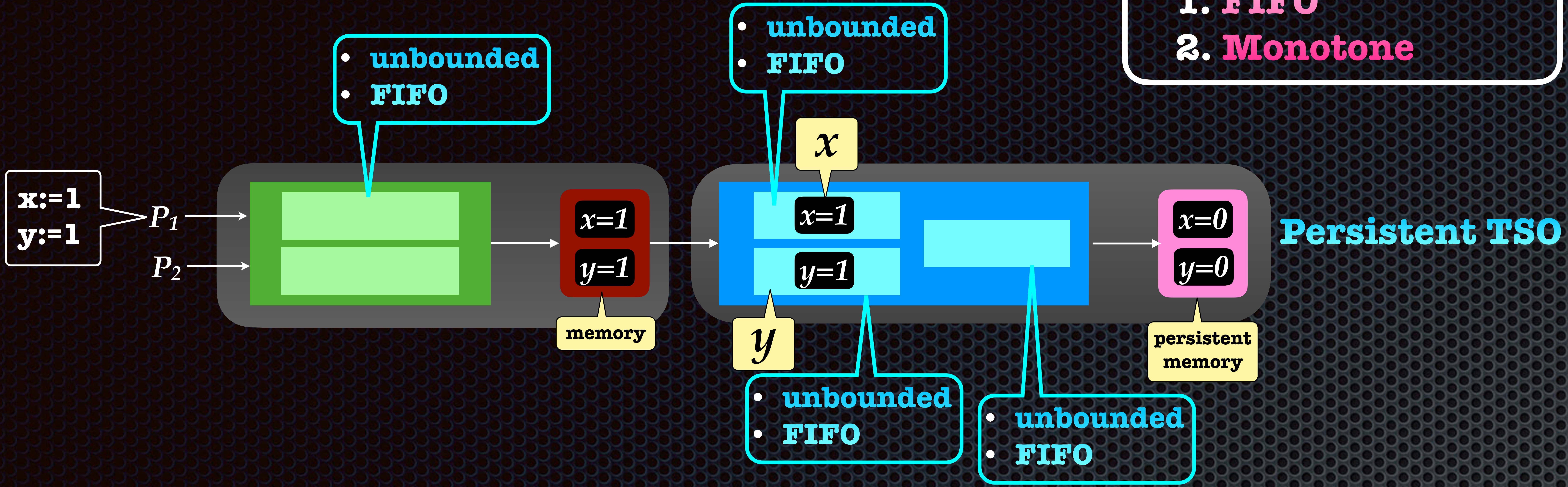
- Make the buffers:
  1. FIFO
  2. Monotone



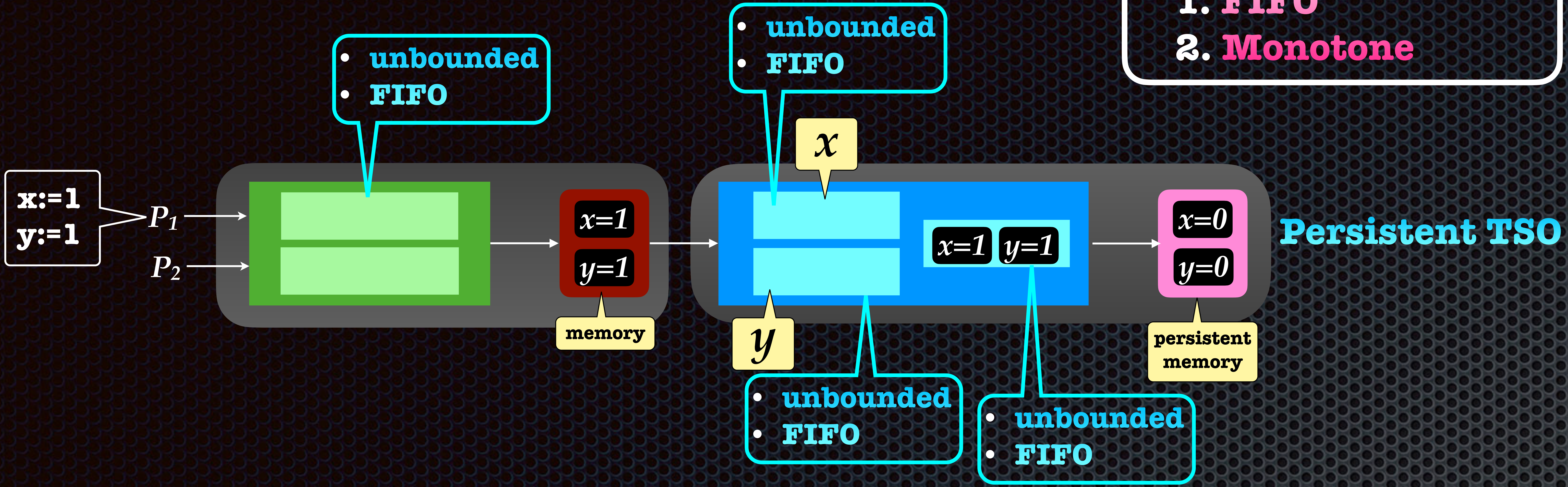
- Make the buffers:
  1. FIFO
  2. Monotone



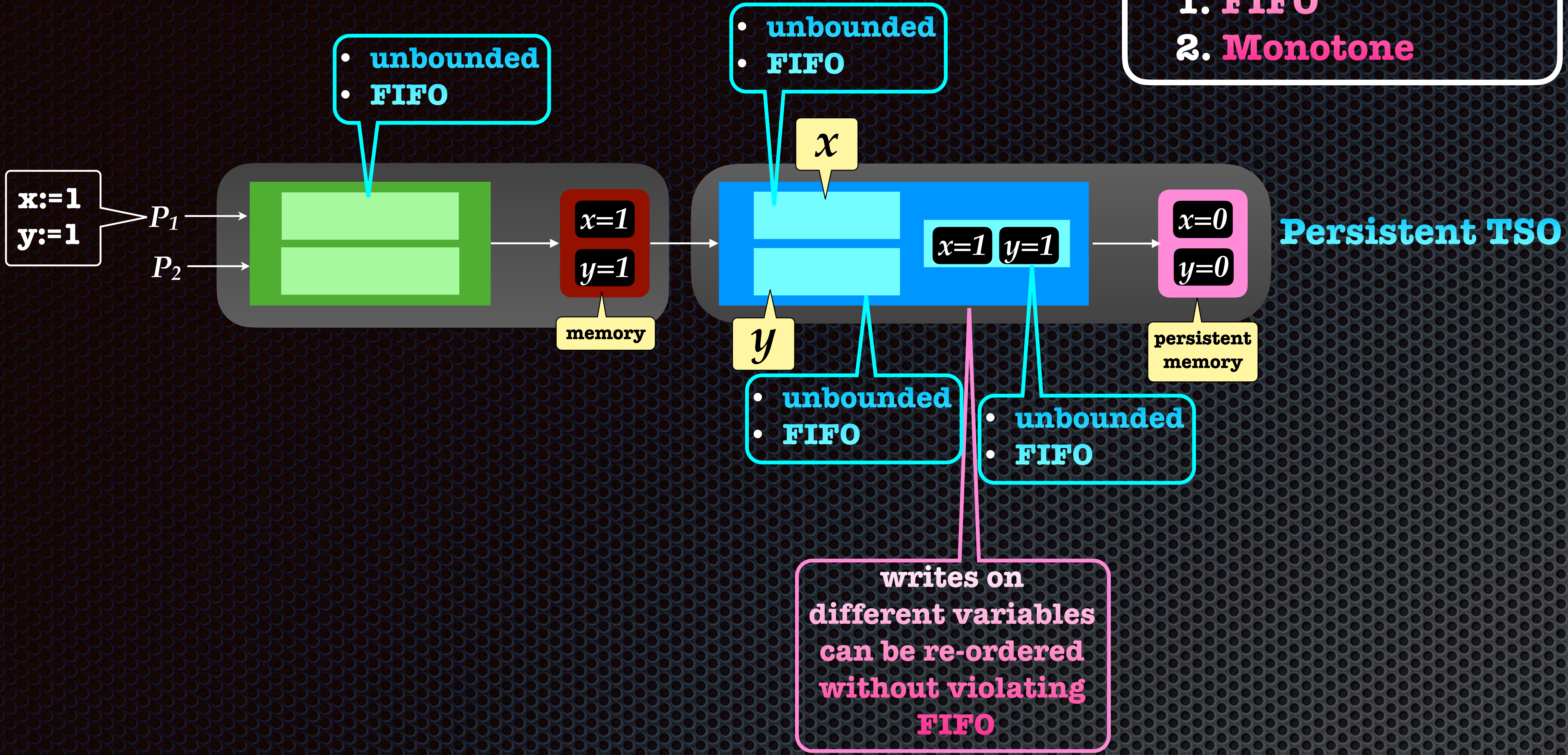
- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



- Make the buffers:
  1. FIFO
  2. Monotone



x:=1
y:=1

P_1

P_2

- unbounded
- FIFO

$x=1$

$y=1$

memory

- unbounded
- FIFO

x

y

$x=1$ $y=1$

- unbounded
- FIFO

- unbounded
- FIFO

$x=0$

$y=0$

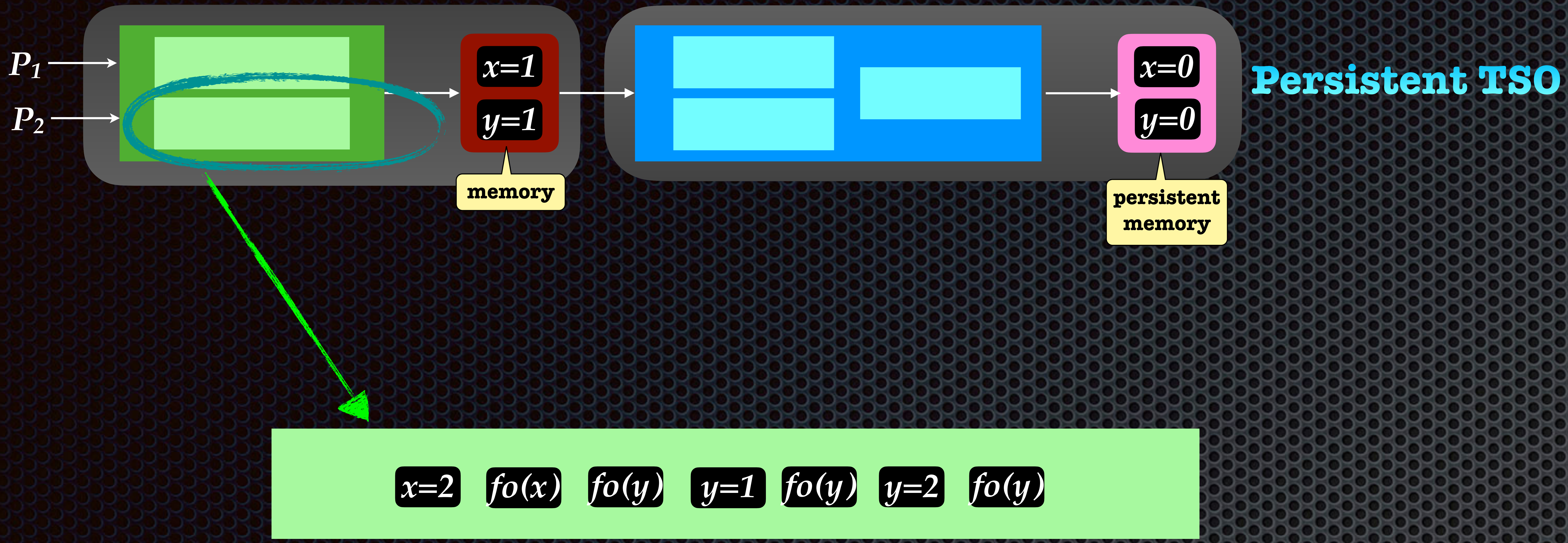
persistent memory

Persistent TSO

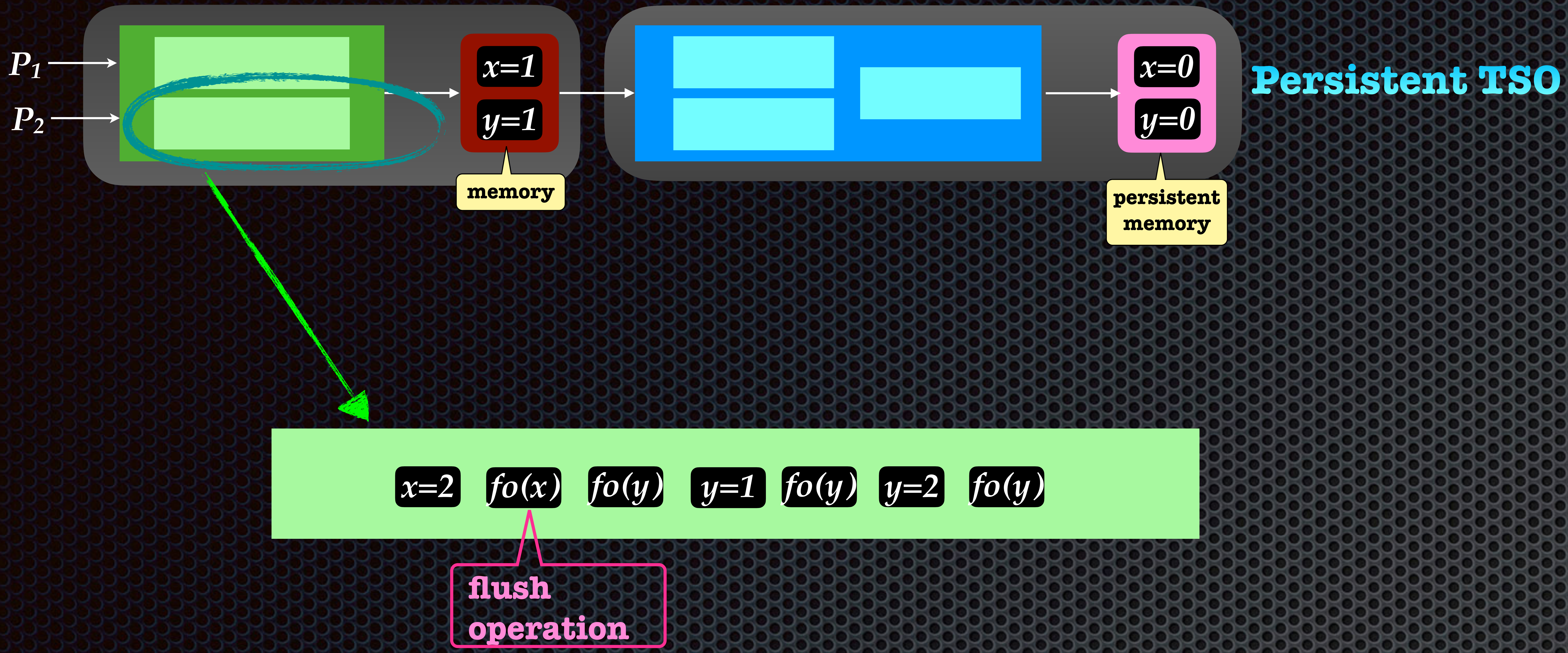
writes on different variables can be re-ordered without violating FIFO

- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**

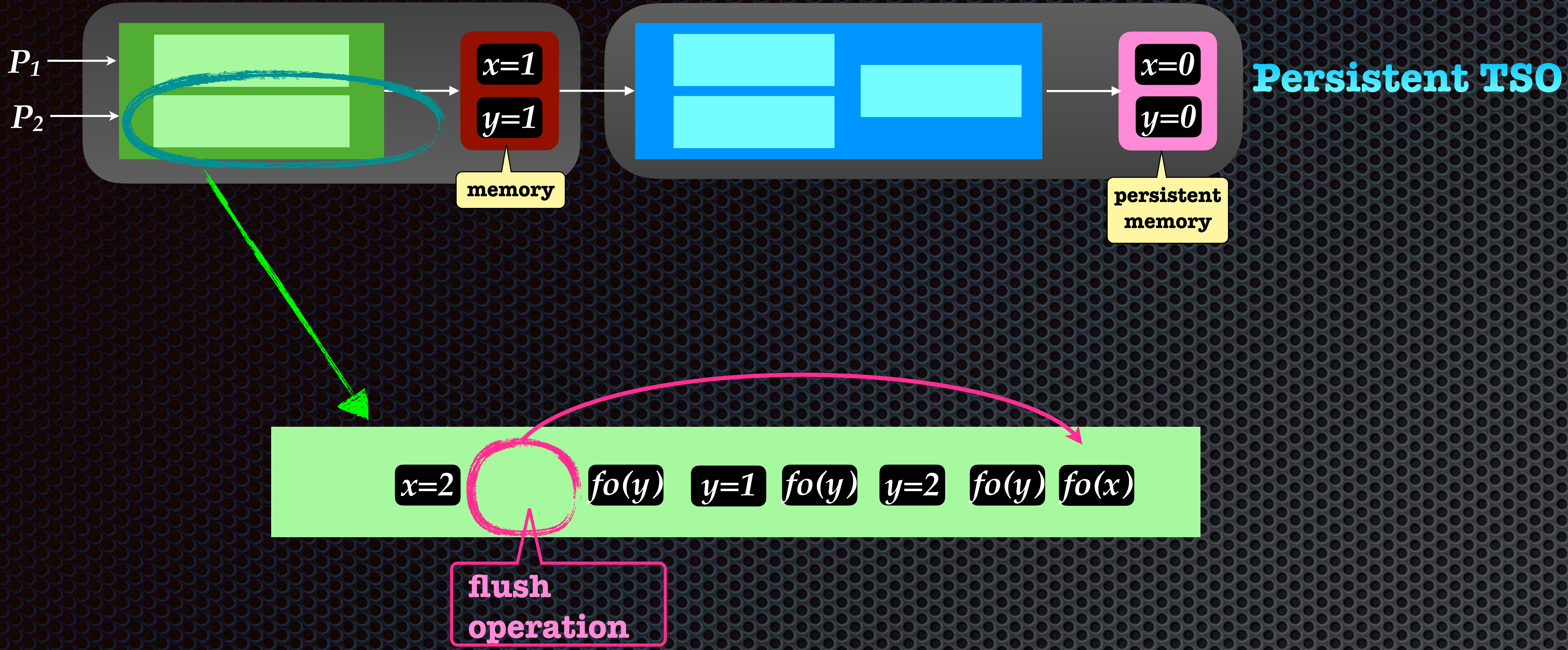
- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



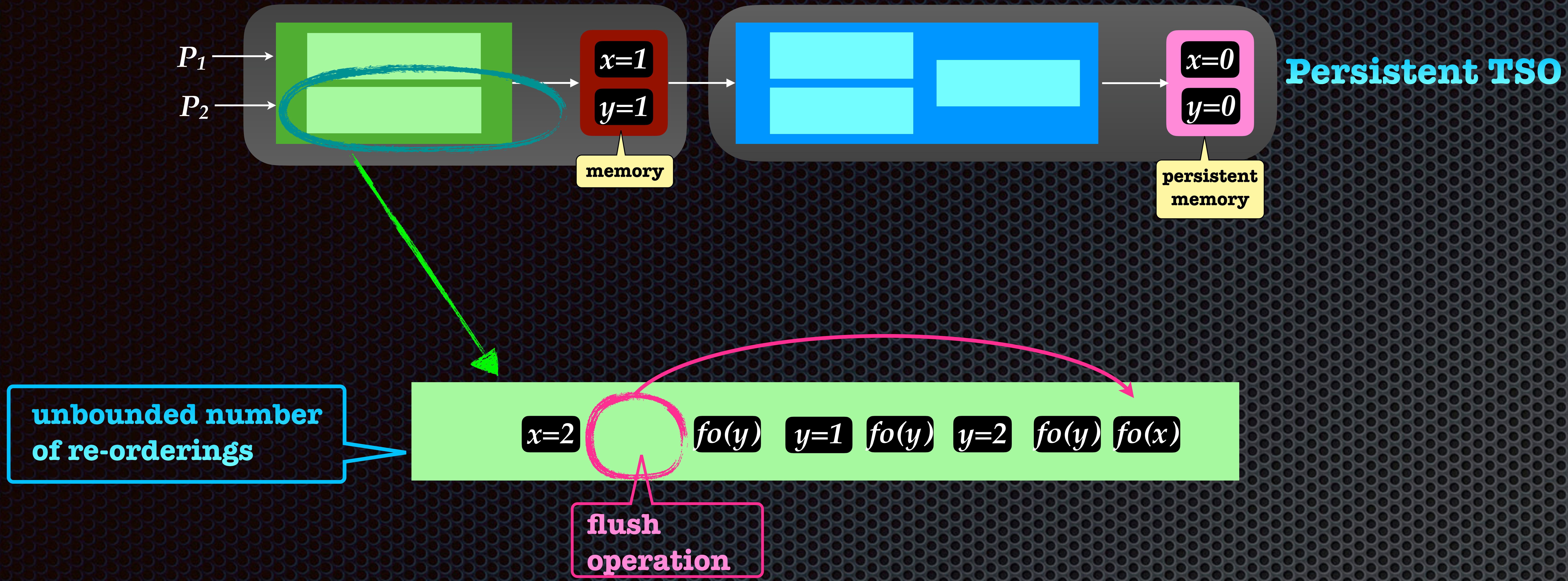
- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



P_1

P_2

$x=1$

$y=1$

memory

$x=0$

$y=0$

persistent memory

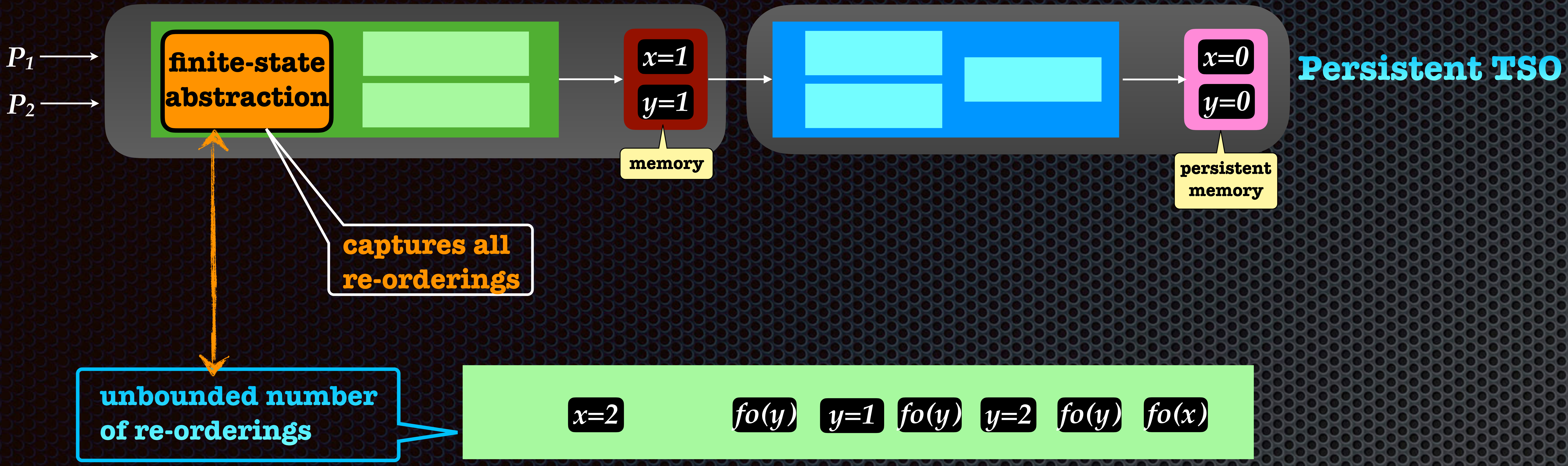
Persistent TSO

unbounded number of re-orderings

$x=2$ $fo(y)$ $y=1$ $fo(y)$ $y=2$ $fo(y)$ $fo(x)$

flush operation

- Make the buffers:
  1. **FIFO**
  2. **Monotone**



P_1

P_2

finite-state abstraction

$x=1$

$y=1$

memory

$x=0$

$y=0$

persistent memory

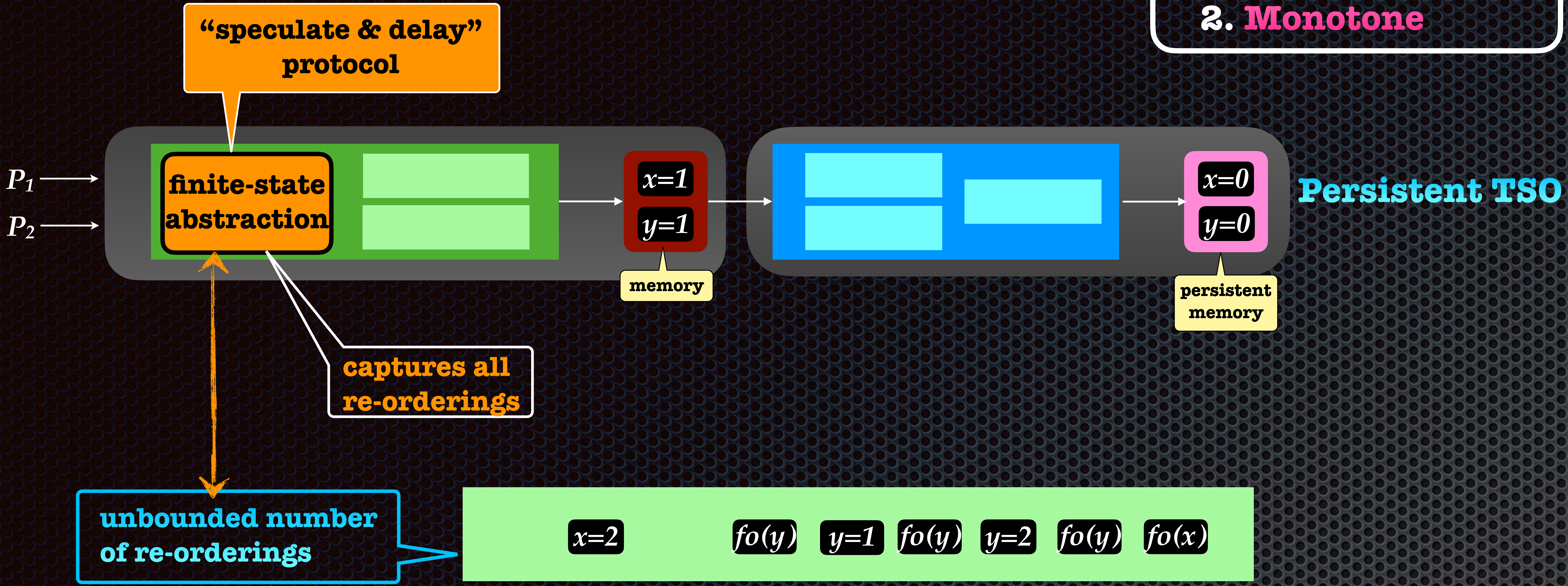
Persistent TSO

captures all re-orderings

unbounded number of re-orderings

$x=2$ $fo(y)$ $y=1$ $fo(y)$ $y=2$ $fo(y)$ $fo(x)$

- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**



"speculate & delay" protocol

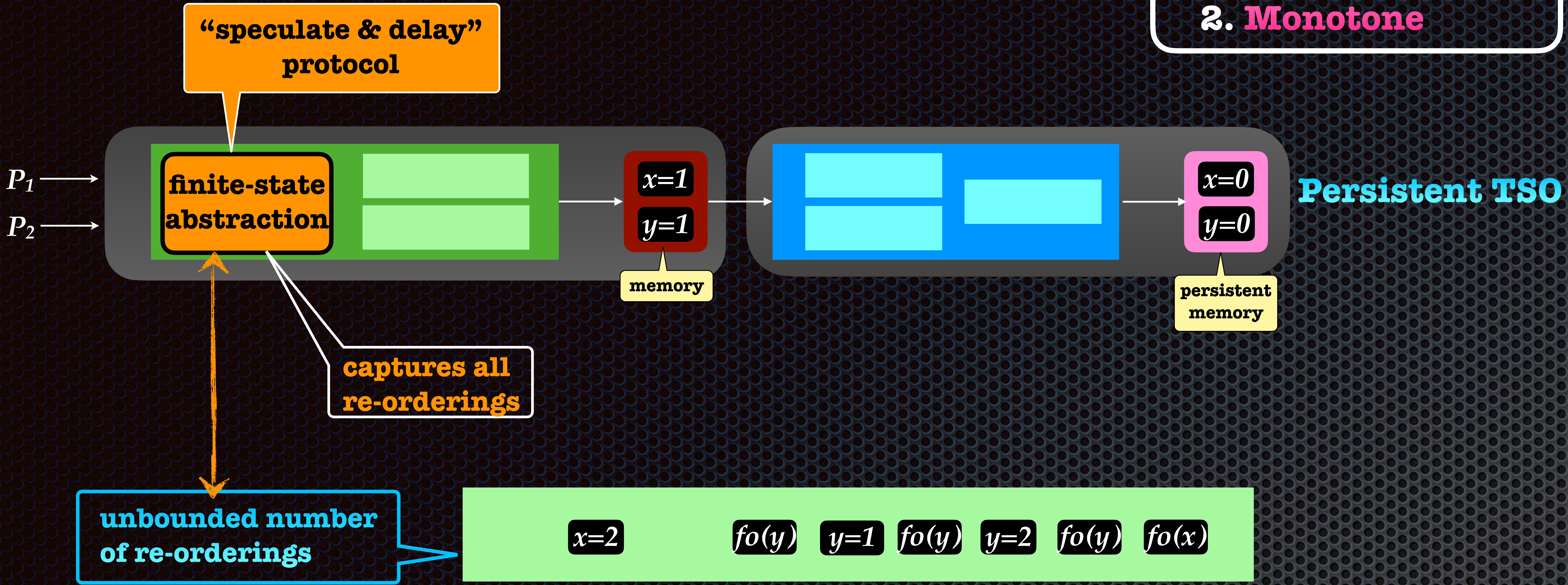
finite-state abstraction

captures all re-orderings

unbounded number of re-orderings

Persistent TSO

- **Make the buffers:**
  1. **FIFO** ✓
  2. **Monotone**



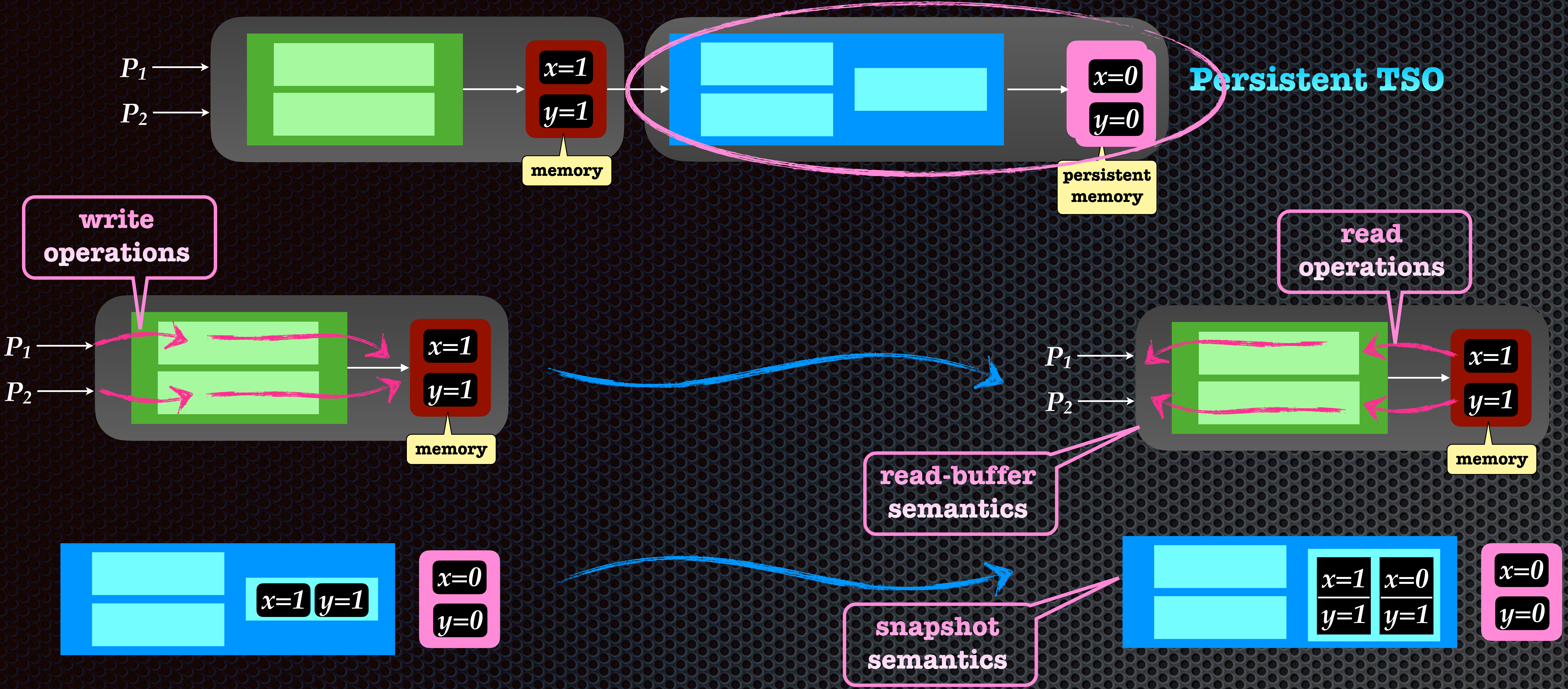
“speculate & delay” protocol

Persistent TSO

captures all re-orderings

unbounded number of re-orderings

- Make the buffers:
  1. FIFO
  2. Monotone ✓



unbounded data structures

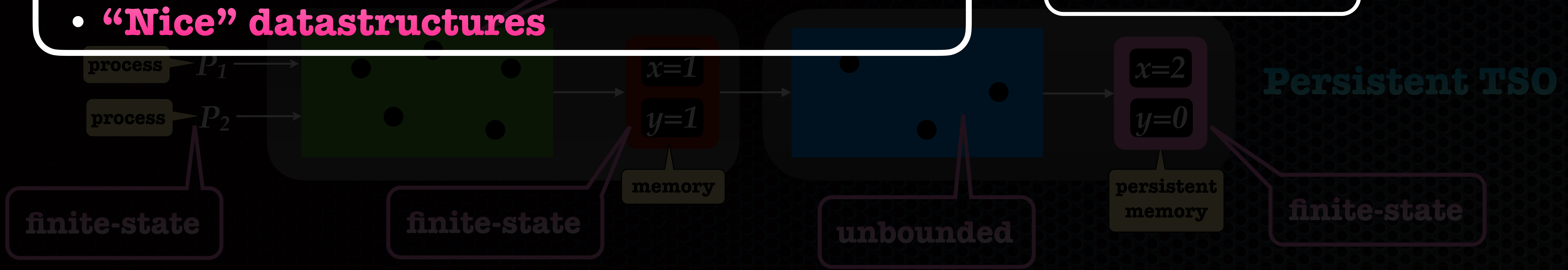
adapting SC techniques

semantics

decidability

- Equivelant to the Raad et al semantics
- Allows applying classical (SC) techniques
- “Nice” datastructures

this work:
new semantics



Well-Structured Systems

- Make the buffers:
  1. FIFO
  2. Monotone

- unbounded
- FIFO
- monotone

Reachability Problem Decidable

Monotonicity

- **Equivelant to the Raad et al semantics**
- **Allows applying classical (SC) techniques**
- **“Nice” datastructures**

this work:
new semantics

- **Make the buffers:**
  1. **FIFO**
  2. **Monotone**

- **unbounded**
- **FIFO**
- **monotone**

unbounded data structures

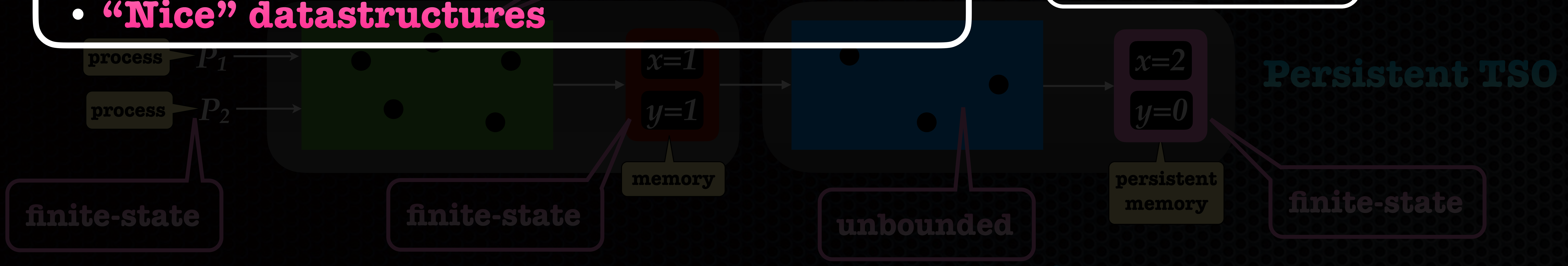
adapting SC techniques

semantics

decidability

- Equivelant to the Raad et al semantics
- Allows applying classical (SC) techniques
- “Nice” datastructures

this work:
new semantics



Well-Structured Systems

- Make the buffers:
  1. FIFO
  2. Monotone

- unbounded
- FIFO
- monotone

well-structured system

“we are facing big challenges”

“we are facing big challenges”

“solutions require interaction
across reach communities”

“we are facing big challenges”

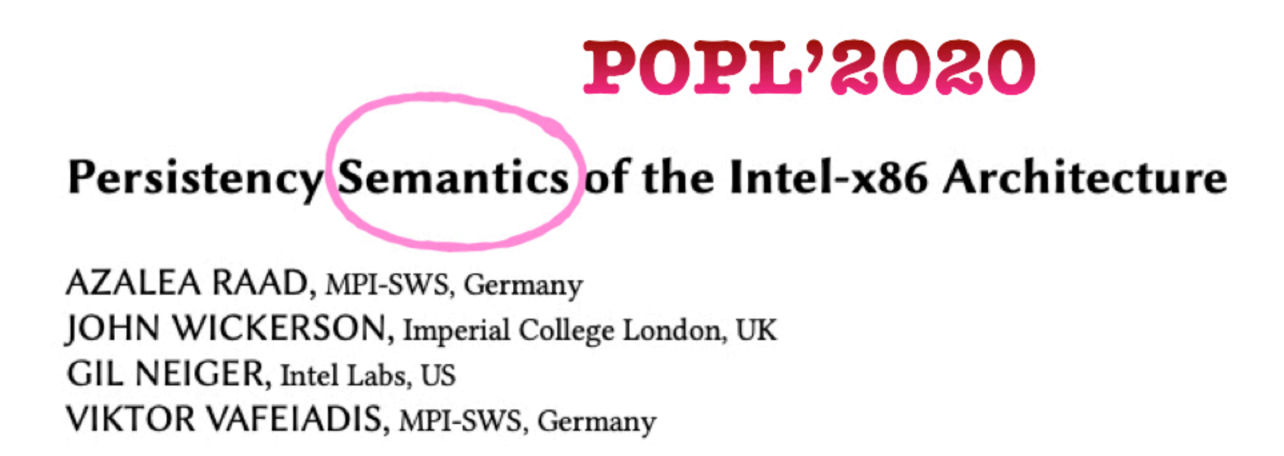
“solutions require interaction
across reach communities”

“POPL is an excellent forum to achieve that”

“we are facing big challenges”

“solutions require interaction
across reach communities”

“POPL is an excellent forum to achieve that”



POPL’2020

Persistency Semantics of the Intel-x86 Architecture

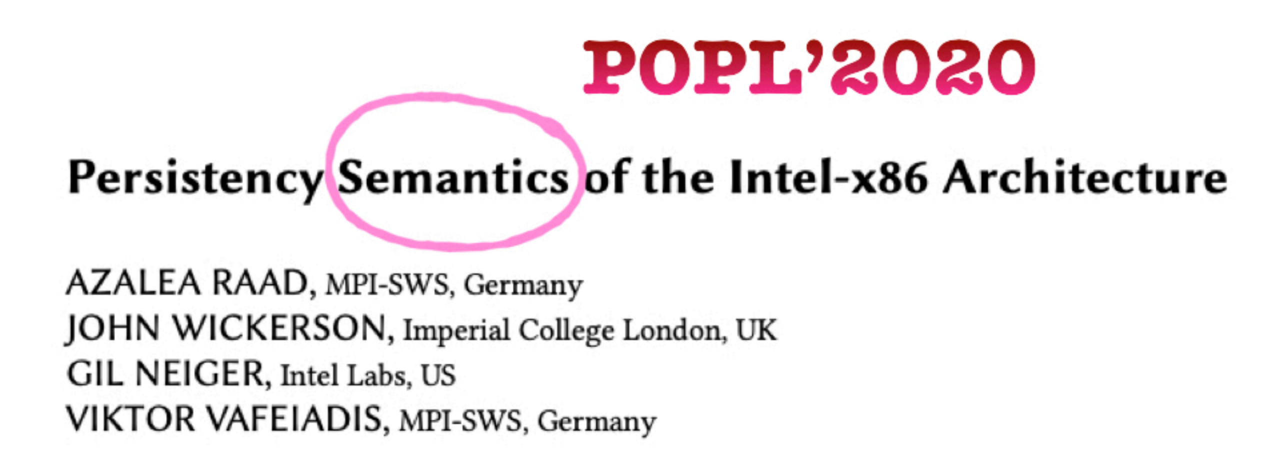
AZALEA RAAD, MPI-SWS, Germany
JOHN WICKERSON, Imperial College London, UK
GIL NEIGER, Intel Labs, US
VIKTOR VAFEIADIS, MPI-SWS, Germany

architecture + programming languages

"we are facing big challenges"

"solutions require interaction
across reach communities"

"POPL is an excellent forum to achieve that"



POPL'2020

Persistency Semantics of the Intel-x86 Architecture

AZALEA RAAD, MPI-SWS, Germany
JOHN WICKERSON, Imperial College London, UK
GIL NEIGER, Intel Labs, US
VIKTOR VAFEIADIS, MPI-SWS, Germany

architecture + programming languages

this work: programming languages + program verification