Concurrent Algorithms and Data Structures

Concurrent Algorithms and Data Structures

- **Slides uploaded to “Studium”**

Concurrent Algorithms and Data Structures

- Slides uploaded to “Studium”
- Please, **download** the slides first, and only then run the animations

Concurrent Algorithms and Data Structures

- Slides uploaded to “Studium”
- Please, download the slides first, and only then run the animations
- Report problems with downloading/ watching the slides

Concurrent Algorithms and Data Structures

- Slides uploaded to “Studium”
- Please, download the slides first, and only then run the animations
- Report problems with downloading/ watching the slides
- Low-Resolution, No-Animation pdfs are available (around 50 MBs)

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Specification

- safety properties
- liveness properties

Safety Properties

"nothing bad will happen during the execution of the program"

Reachability of bad states

Liveness Properties

"something good will eventually happen during the execution of the program"

Repeated Reachability of good states

Specification

- safety properties
- liveness properties

Safety Properties

"nothing bad will happen during the execution of the program"

Reachability of bad states

Linearizability

Liveness Properties

"something good will eventually happen during the execution of the program"

Repeated Reachability of good states

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Abstract Data Type

- Definition
- Sets
- Queues
- Stacks

"atomic execution of a method"

Type	Set

- States
- initial state
- Methods
- Rules

operation

command

guard

$add(x, true)$

$S \longrightarrow S \cup \{x\}$

if $x \notin S$

source state

name

input

output

target state

rules

guarded commands

set of integers

state

2

5

- *add*
- *rmv*
- *ctn*

methods

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state

2

5

rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

"infinite set of instantiations"

$$\{2,5\} \xrightarrow{add(4,true)} \{2,5,4\} \qquad \{2,5,4\} \xrightarrow{add(7,true)} \{2,5,4,7\} \quad \cdots\cdots$$

Abstract Data Type

Set

- States
- initial state
- Methods
- Rules

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

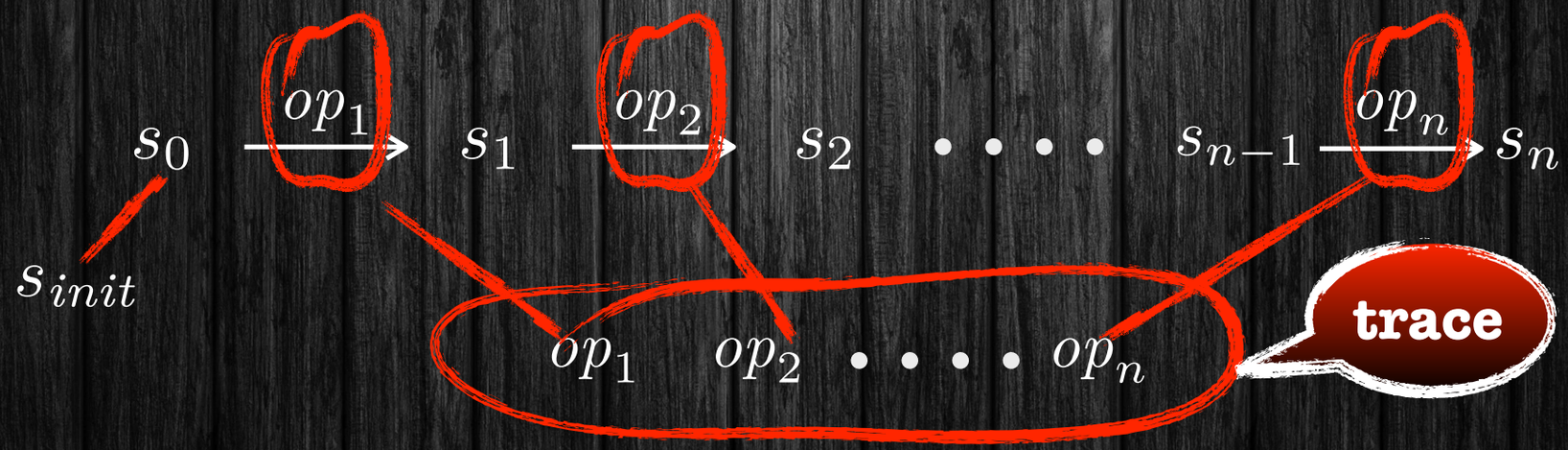
$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

Traces

- start from initial state
- apply a sequence of rules
- collect the sequence of operations



Abstract Data Type Set

- States
- initial state
- Methods
- Rules

initially empty

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state

rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

set of integers

state

2

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

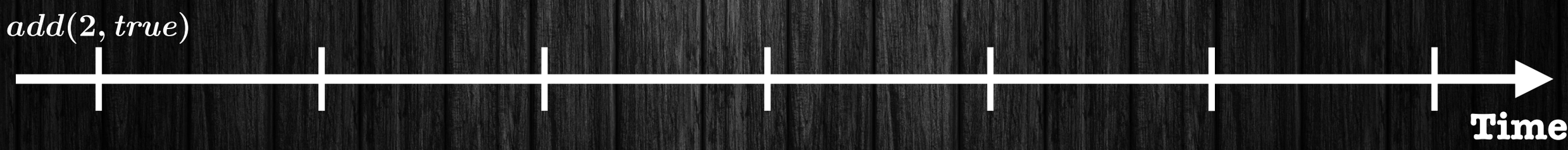
rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

$add(2, true)$? ✔



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state

2

5

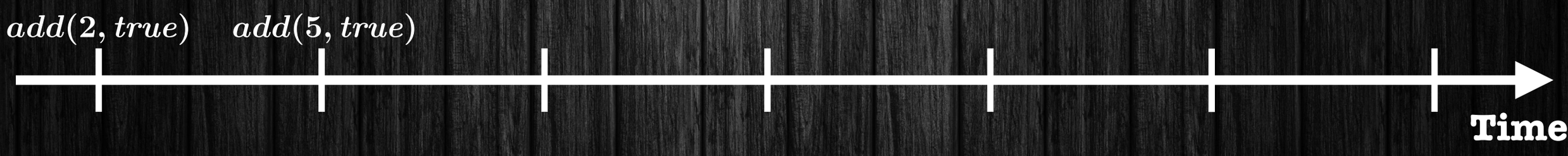
rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

$add(5, true)$? ✔



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state

2

5

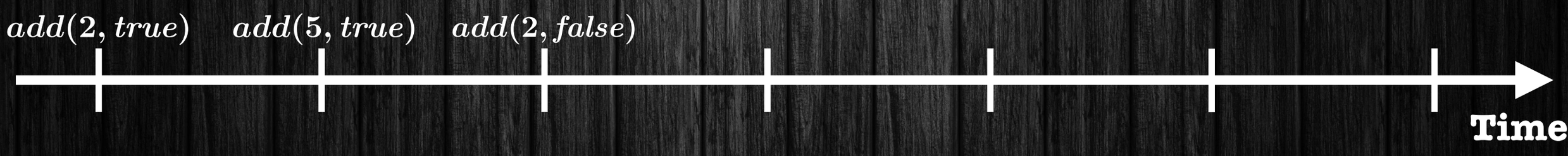
rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

$add(2, false)$? ✔



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state

2

5

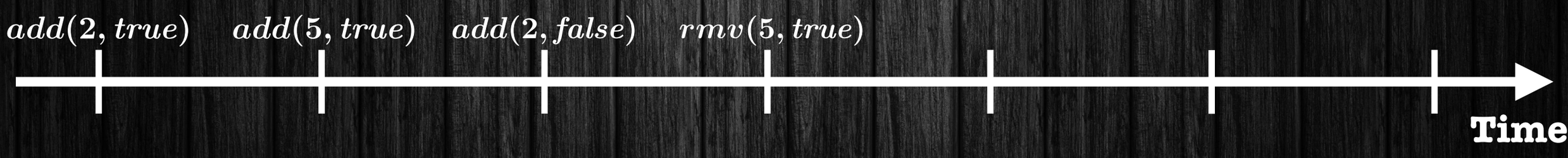
rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

$rmv(5, true)$?



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state

2

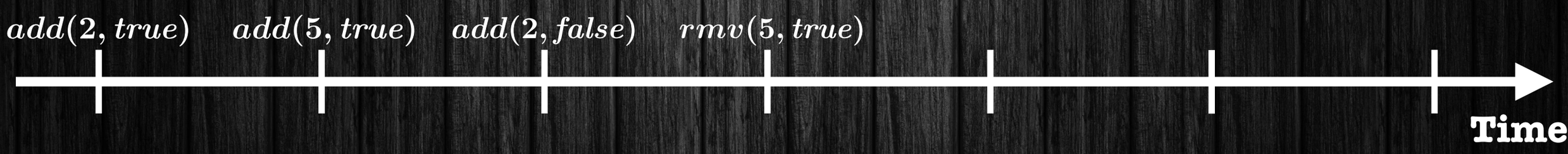
rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

$rmv(3, true)$?



Abstract Data Type Set

- States
- initial state
- Methods
- Rules

initially empty

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state

2

9

rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

$add(9, true)$?

trace

$add(2,true)$ $add(5,true)$ $add(2,false)$ $rmv(5,true)$ $rmv(3,false)$ $ctn(3,false)$ $add(9,true)$

Time

$$s_0 \xrightarrow{op_1} s_1 \xrightarrow{op_2} s_2 \bullet\bullet\bullet\bullet s_{n-1} \xrightarrow{op_n} s_n$$

Abstract Data Type — Set

- States
- initial state
- Methods
- Rules

initially empty

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

set of integers

state



rules

guarded commands

- *add*
- *rmv*
- *ctn*

methods

Traces (Set)

$add(2,true)$ $add(5,true)$ $add(2,false)$ $rmv(5,true)$ $rmv(3,false)$ $ctn(3,false)$ $add(9,true)$ ✓

$add(2,true)$ $add(5,true)$ $ctn(2,true)$ ✓

$add(2,true)$ $add(5,true)$ $ctn(2,false)$ ✗

$add(2,true)$ $add(5,true)$ $ctn(3,true)$ ✗

atomic operations

Abstract Data Type

- Definition
- Sets
- **Queues**
- Stacks

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

word over integers

state

2 5 1 2

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

initially ϵ

Abstract Data Type Queue

- States
- ~~initial state~~
- Methods
- Rules

- *enq*
- *deq*

methods

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

$$w \xrightarrow{enq(a)} w \bullet a$$

$$a \bullet w \xrightarrow{deq(a)} w$$

$$\epsilon \xrightarrow{deq(*)} \epsilon$$

rules

guarded commands

- *enq*
- *deq*

methods

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

$$w \xrightarrow{enq(a)} w \bullet a$$

$$a \bullet w \xrightarrow{deq(a)} w$$

$$\epsilon \xrightarrow{deq(*)} \epsilon$$

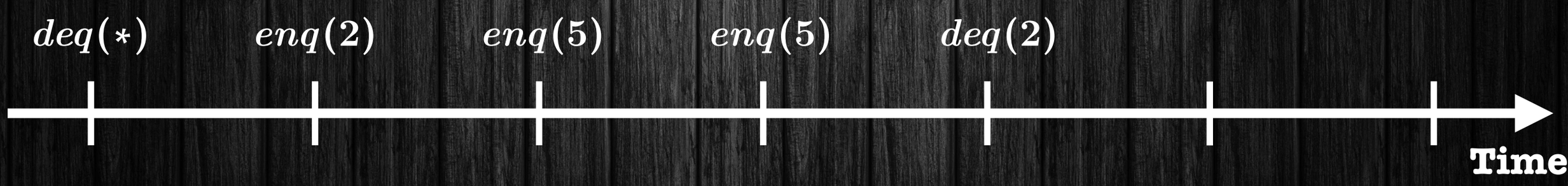
initially ϵ

5 5

Traces

- start from initial state
- apply a sequence of rules
- collect the sequence of operations

Traces (Queue)



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

$$w \xrightarrow{enq(a)} w \bullet a$$

$$a \bullet w \xrightarrow{deq(a)} w$$

$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

Traces(Queue)

$enq(3)$ $enq(5)$ $enq(6)$ $deq(3)$ $deq(5)$ $deq(6)$ $deq(*)$ ✔

$enq(3)$ $enq(5)$ $enq(6)$ $deq(3)$ $deq(5)$ $enq(5)$ $deq(6)$ ✔

$enq(3)$ $enq(5)$ $deq(5)$ ✗

$enq(3)$ $enq(5)$ $enq(6)$ $deq(3)$ $deq(5)$ $enq(5)$ $deq(6)$ $deq(*)$ ✗

atomic operations

Abstract Data Type

- Definition
- Sets
- Queues
- **Stacks**

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

push(3) push(7) push(5) pop(5) pop(7)

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

initially ϵ

$$w \xrightarrow{push(a)} a \bullet w$$

$$a \bullet w \xrightarrow{pop(a)} w$$

$$\epsilon \xrightarrow{pop(*)} \epsilon$$

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

Traces(Stack)

trace

τ

atomic operations

push(3) push(7) push(5) pop(5) pop(7)

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

word over integers

state

$$w \xrightarrow{push(a)} a \bullet w$$

$$pop(a)$$

initially ϵ

Traces = sequences of atomic operations

ush

op

Trac

ethods

trace

τ

atomic operations

push(3) push(7) push(5) pop(5) pop(7)

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Outline

- Overview
- ~~Abstract Data Types~~
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Sequential Programs

- Programs
- Sequential Histories
- Histories ⟶ Traces
- Example: implementing a set
- Program ⟶ Histories ⟶ Traces

Sequential Programs

- Programs
- Sequential Histories
- Histories → Traces
- Example: implementing a set
- Program → Histories → Traces

Sequential Programs

Program

Abstract Data Types

atomic operations

Programs

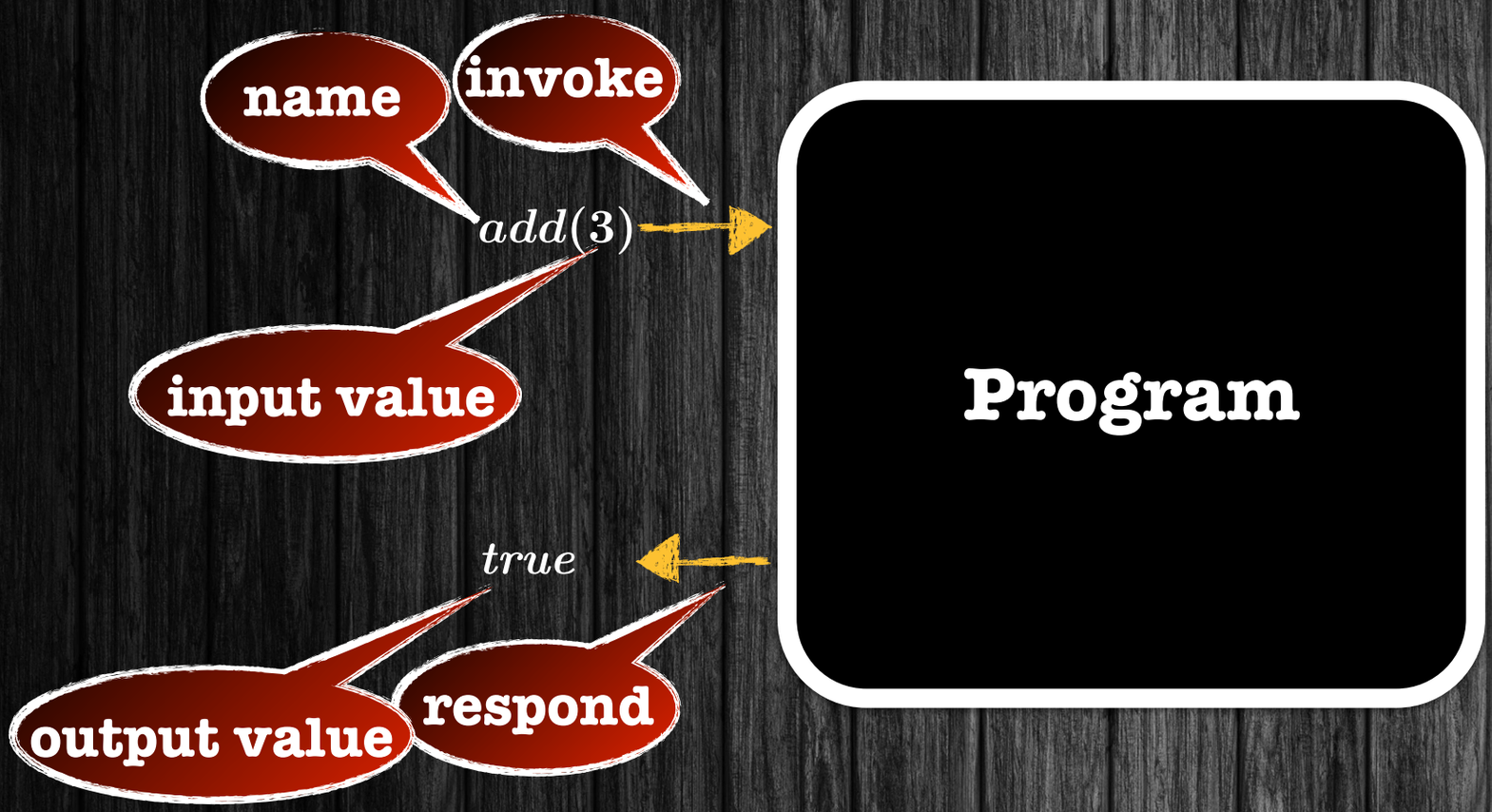
non-atomic operations

operations take time

operation = two events

- invocation
- response

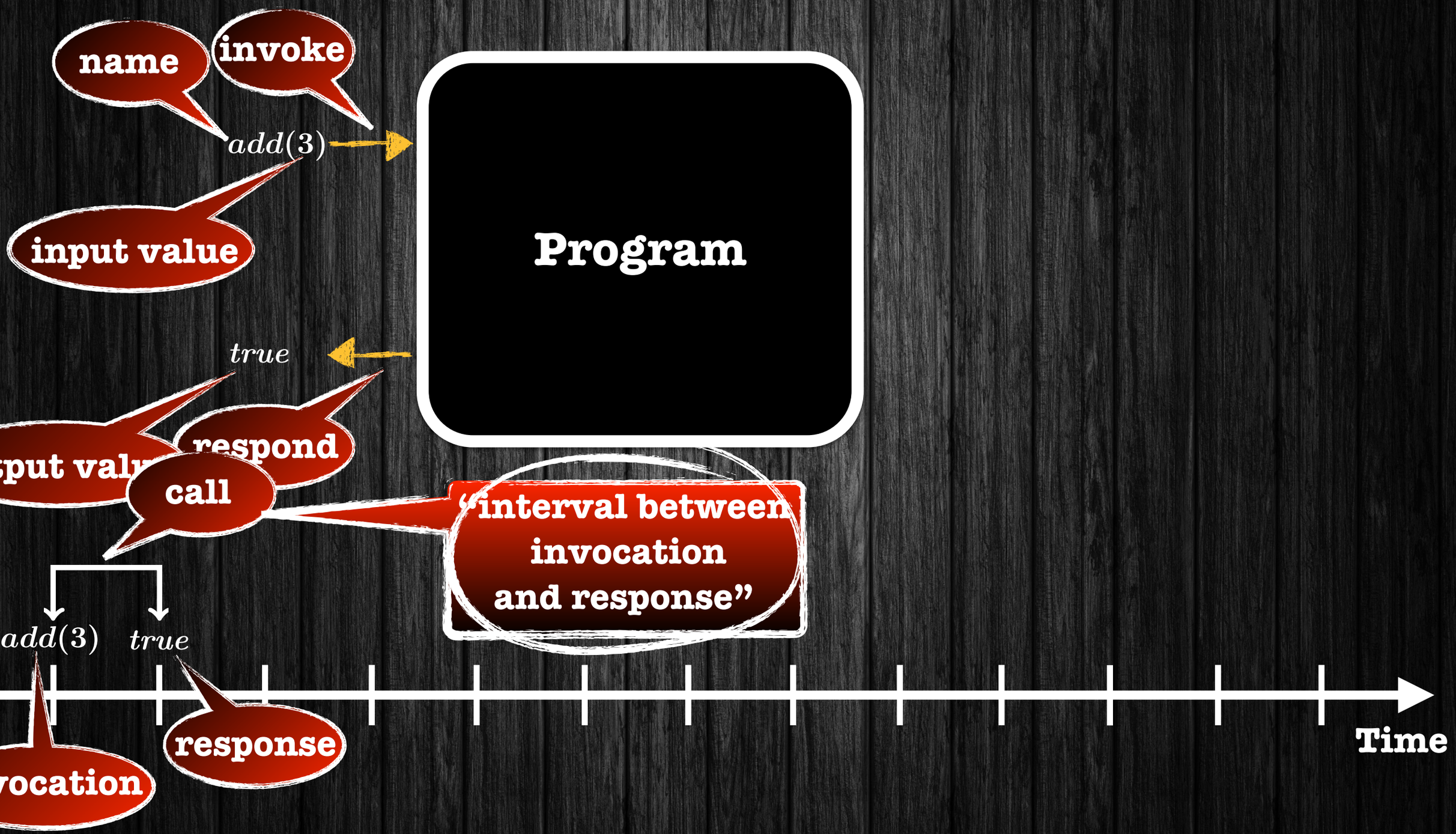
Sequential Programs | Sets



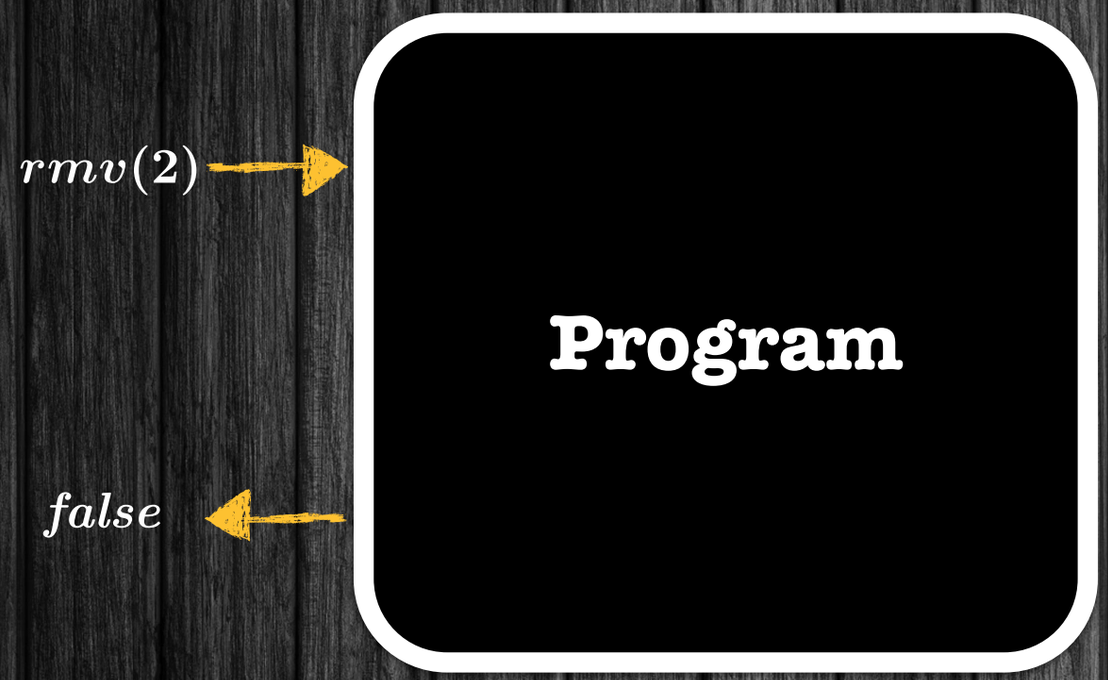
operation = two events

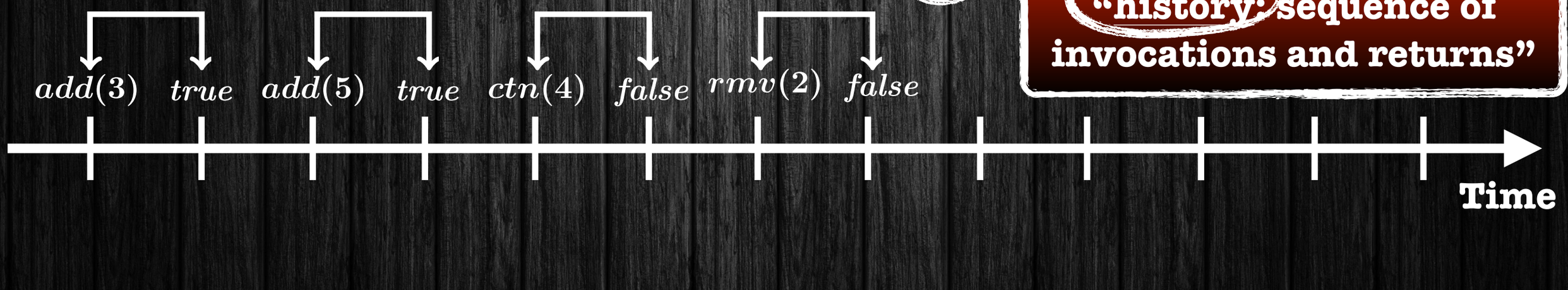
- invocation
- response

Sequential Histories



Sequential Histories





invocation followed immediately by response

method calls do not overlap

sequential

history

H

"history: sequence of invocations and returns"

Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

"a call collapses to an operation"

Program

sequential

history

H

Traceof(H)

call

respond

$add(3)$ $true$ $add(5)$ $true$ $ctn(4)$ $false$ $rmv(2)$ $false$

Time

"merge each invocation with corresponding response"

$add(3, true)$

operation

Time

Sequential Histories

invoke

respond

Program

sequential

history

H

Traceof(H)

$add(3)$ $true$ $add(5)$ $true$ $ctn(4)$ $false$ $rmv(2)$ $false$

τ

τ = Traceof(H)

Time

$add(3, true)$ $add(5, true)$ $ctn(4, false)$ $rmv(2, false)$

Time

Sequential Histories

invoke

respond

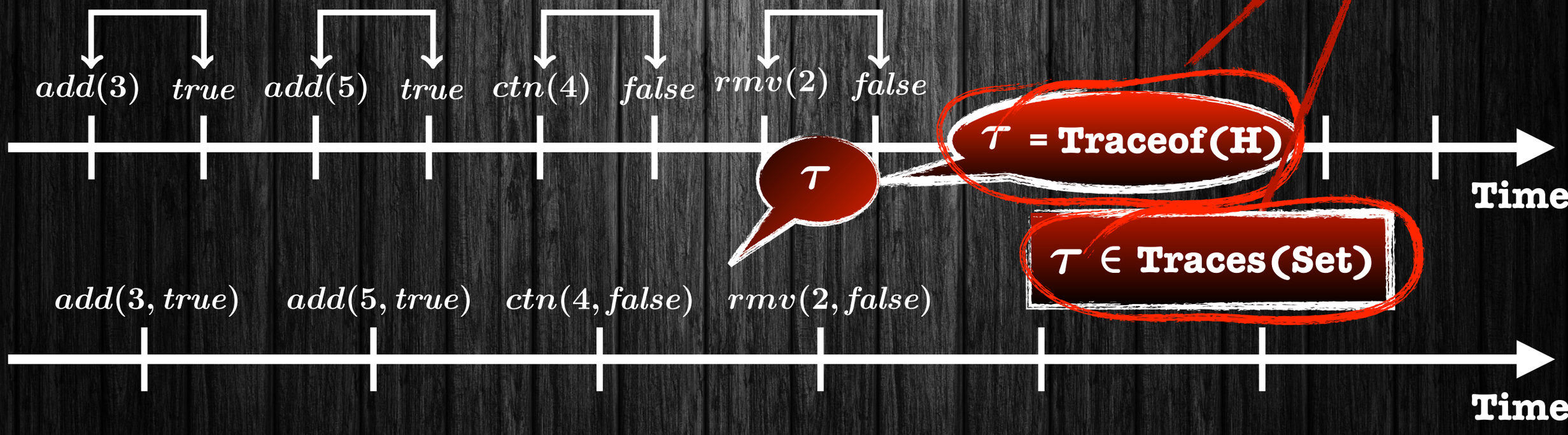
Program

A: abstract data type

H ∈ Histories(A)

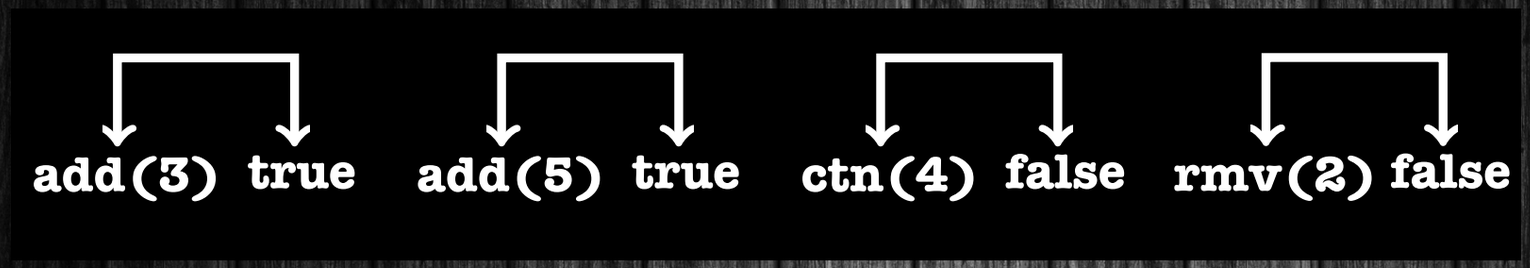
Traceof(H) ∈ Traces(A)

H ∈ Histories(Set)



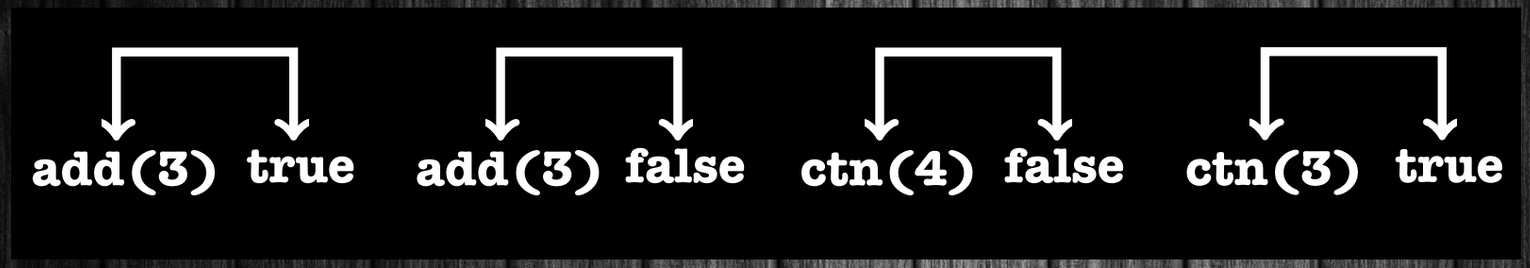
Sequential Histories

H_1



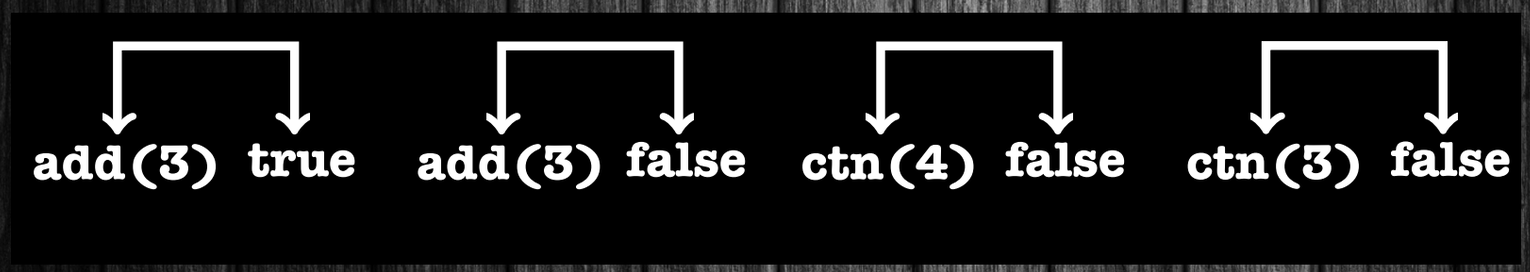
$H_1 \in$ Histories (Set)

H_2



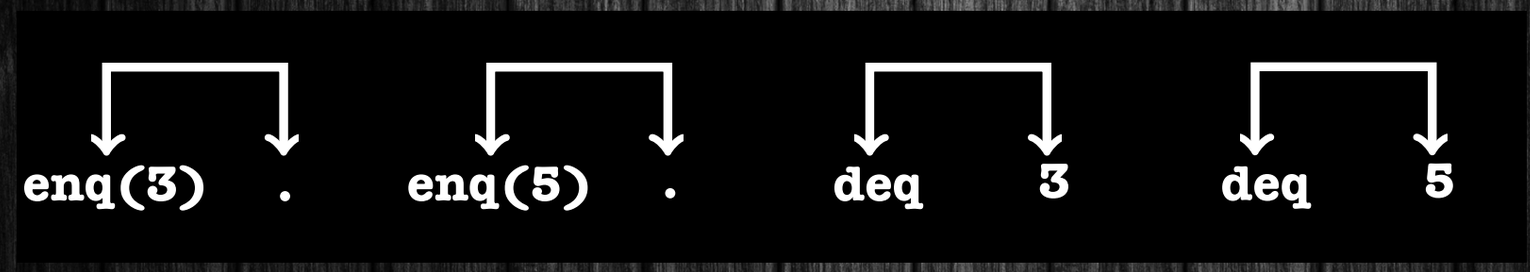
$H_2 \in$ Histories (Set)

H_3



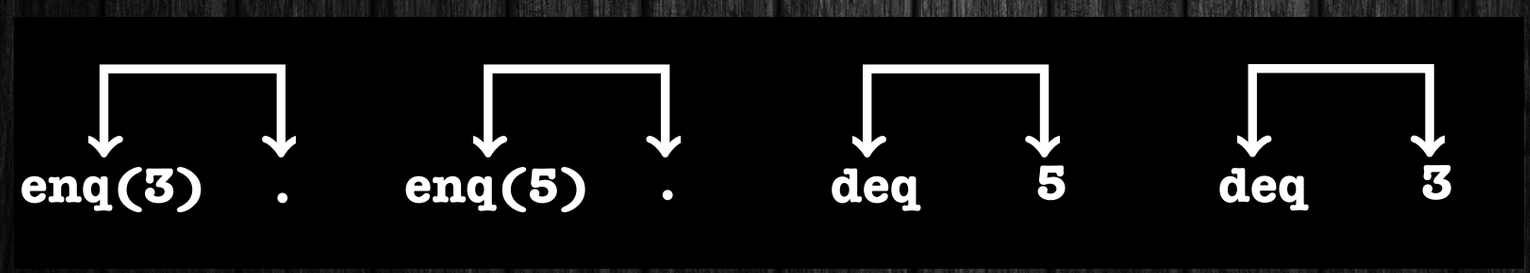
$H_3 \notin$ Histories (Set)

H_4



$H_4 \in$ Histories (Queue)

H_5



$H_5 \notin$ Histories (Queue)

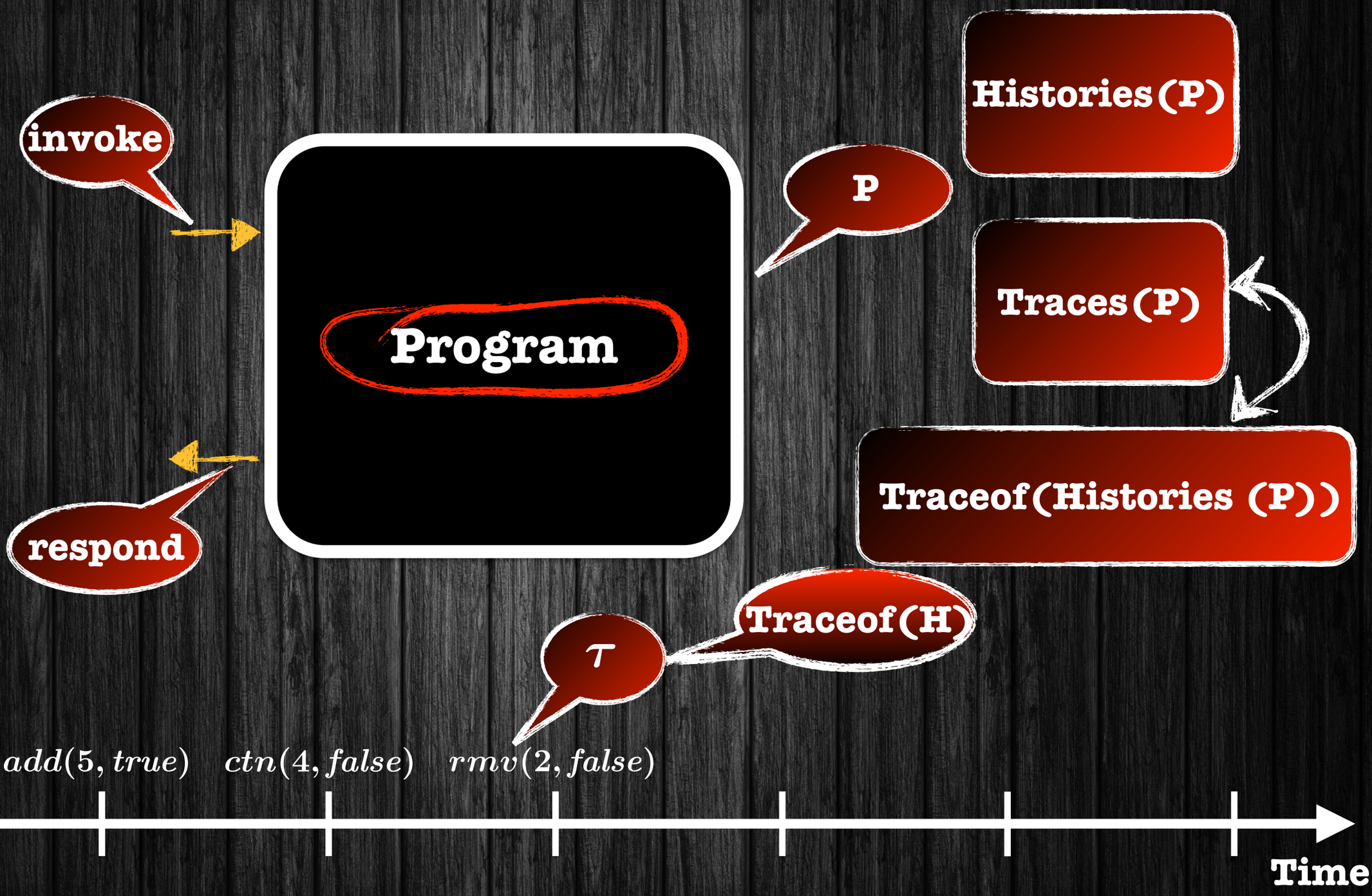
Sequential Programs

- Programs
- Sequential Histories
- Histories ⟶ Traces
- Example: implementing a set
- Program ⟶ Histories ⟶ Traces

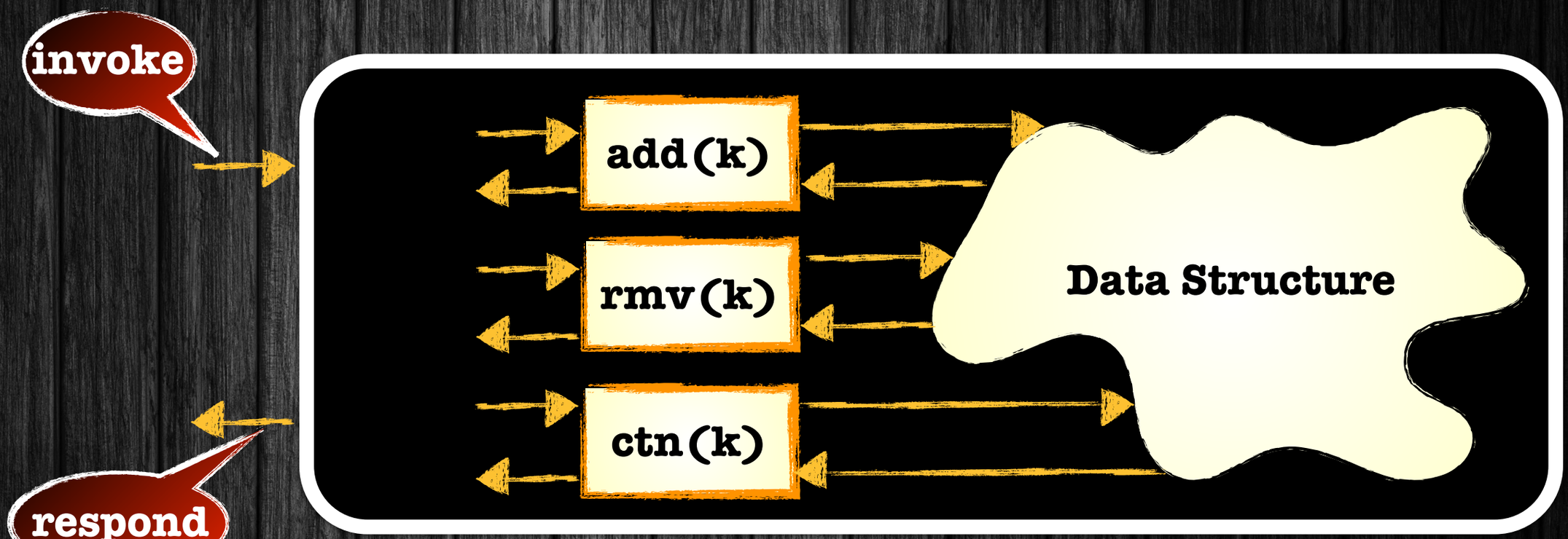
Sequential Programs

- Programs
- Sequential Histories
- Histories → Traces
- Example: implementing a set
- Program → Histories → Traces

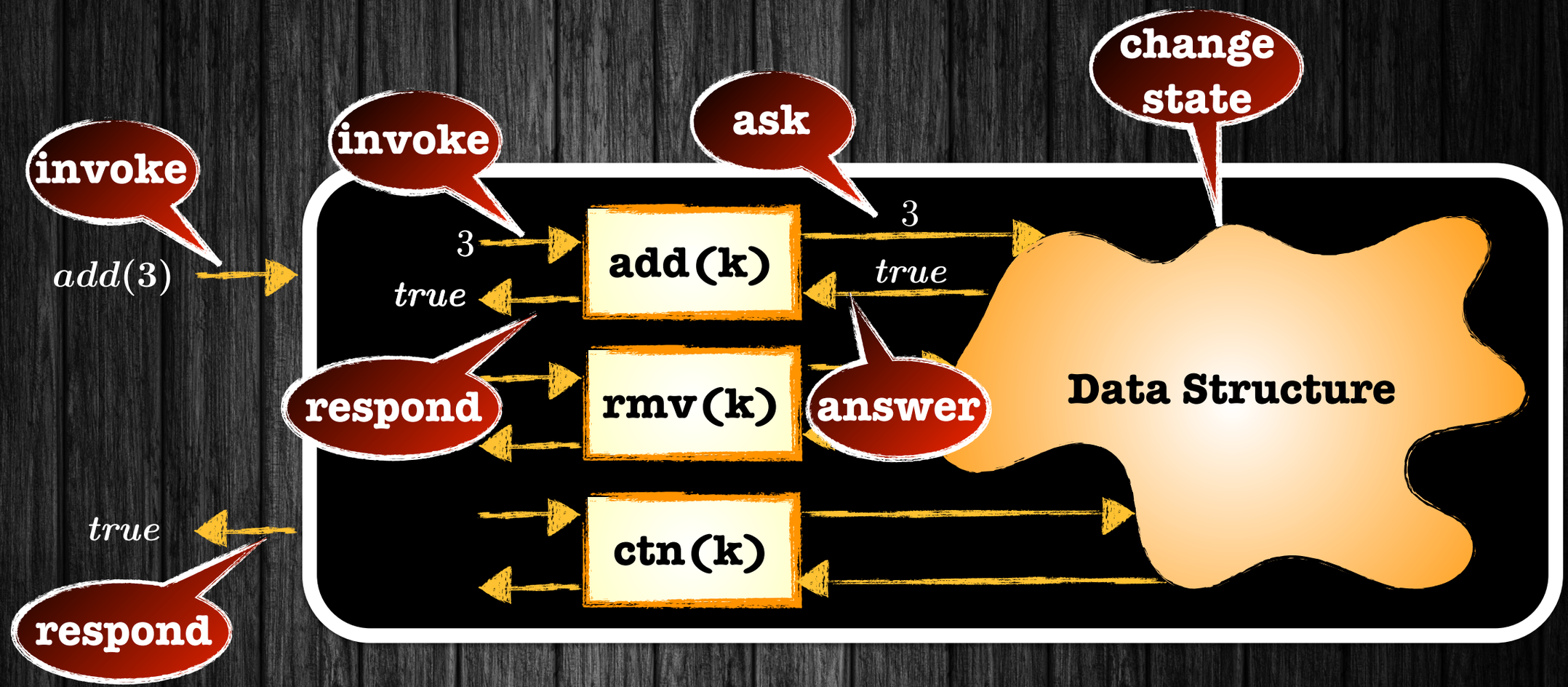
Sets | Sequential Programs



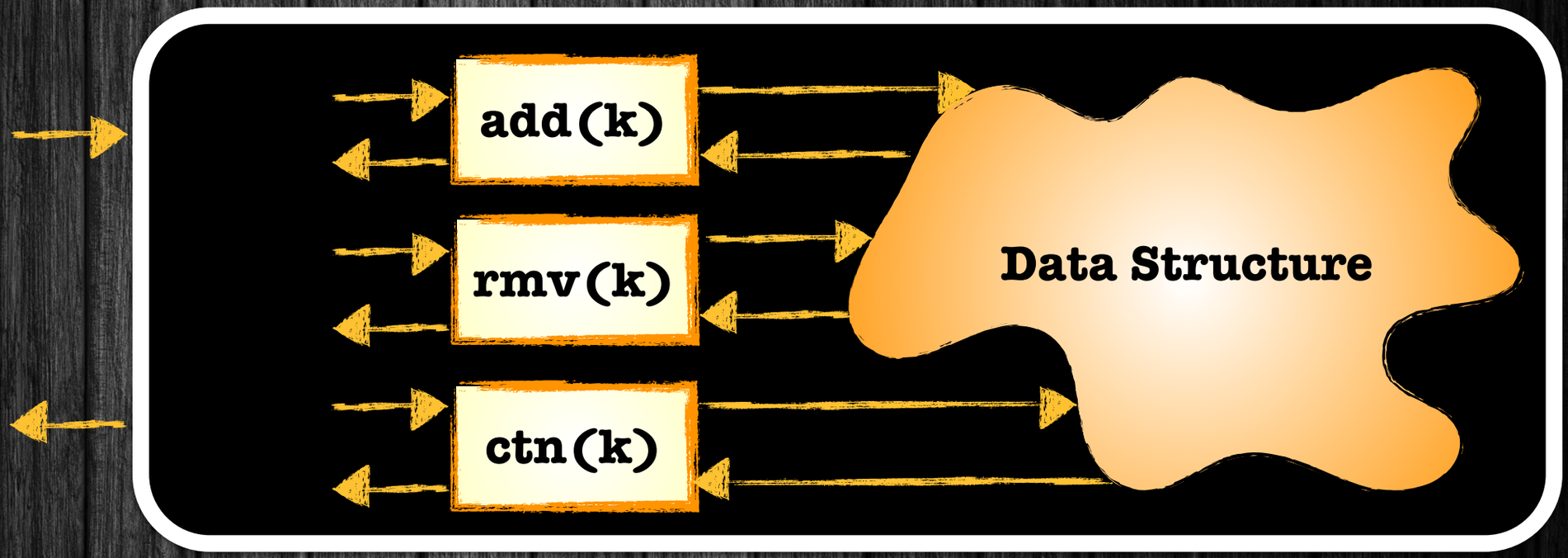
Sets | Sequential Programs



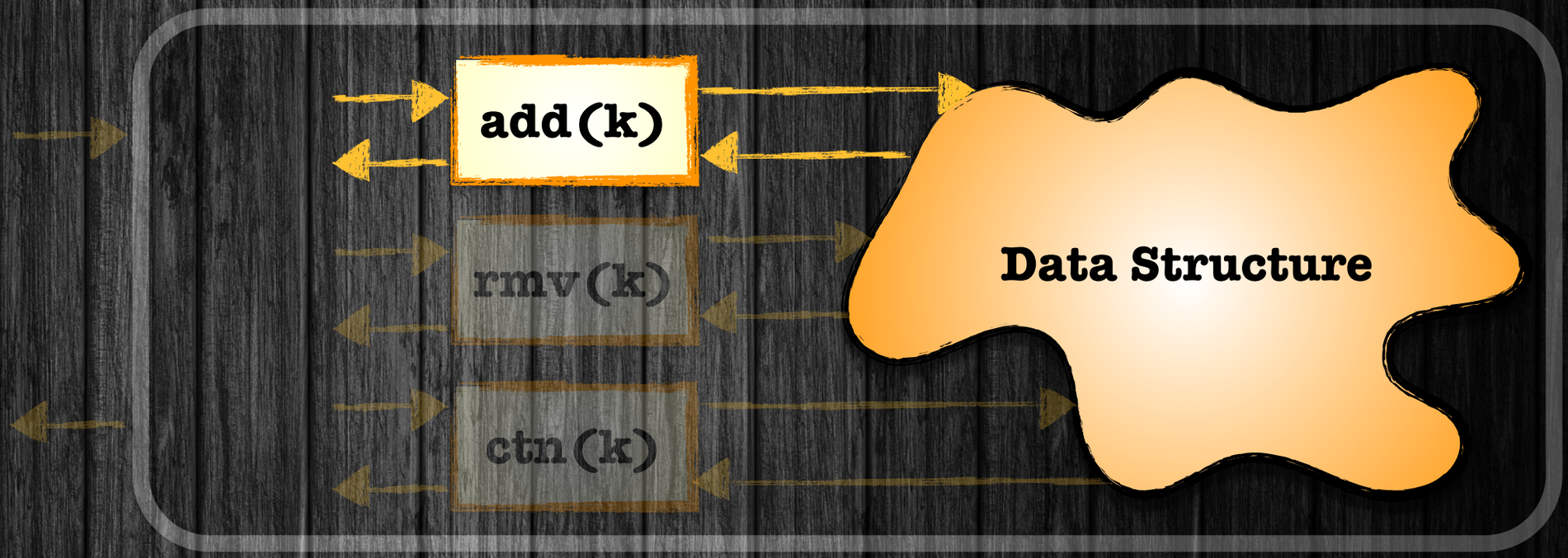
Sets | Sequential Programs



Sets | Sequential Programs



Sets | Sequential Programs



Sets | **Sequential Programs**

`add(k)`

Data Structure

Sets

Sequential Programs

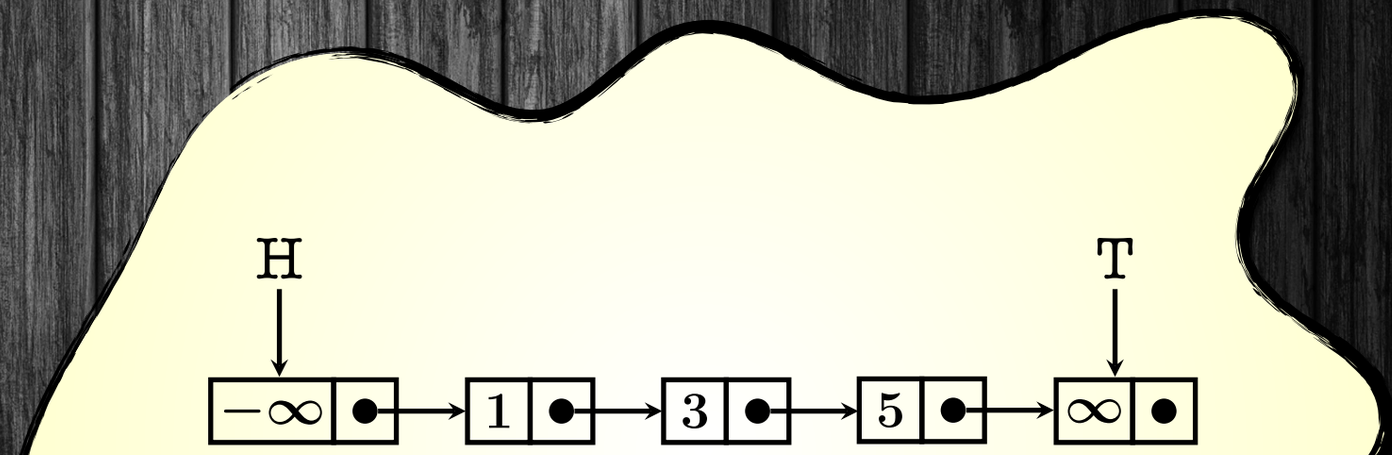
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

Data Structure

Sets

Sequential Programs

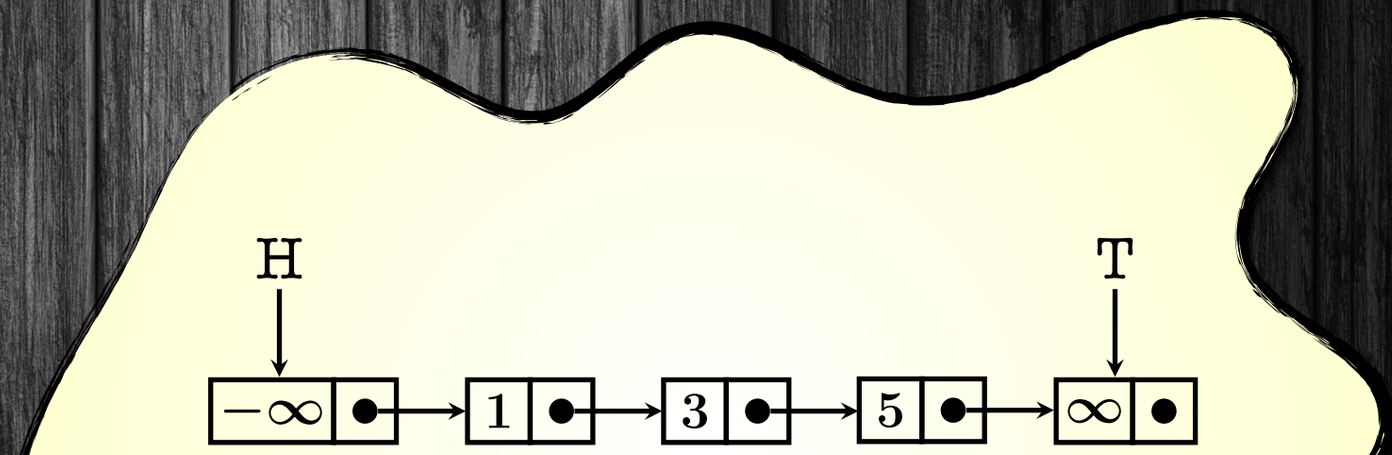
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```



Sets

Sequential Programs

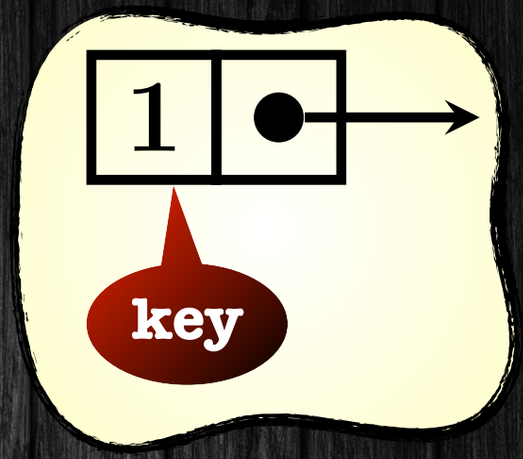
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

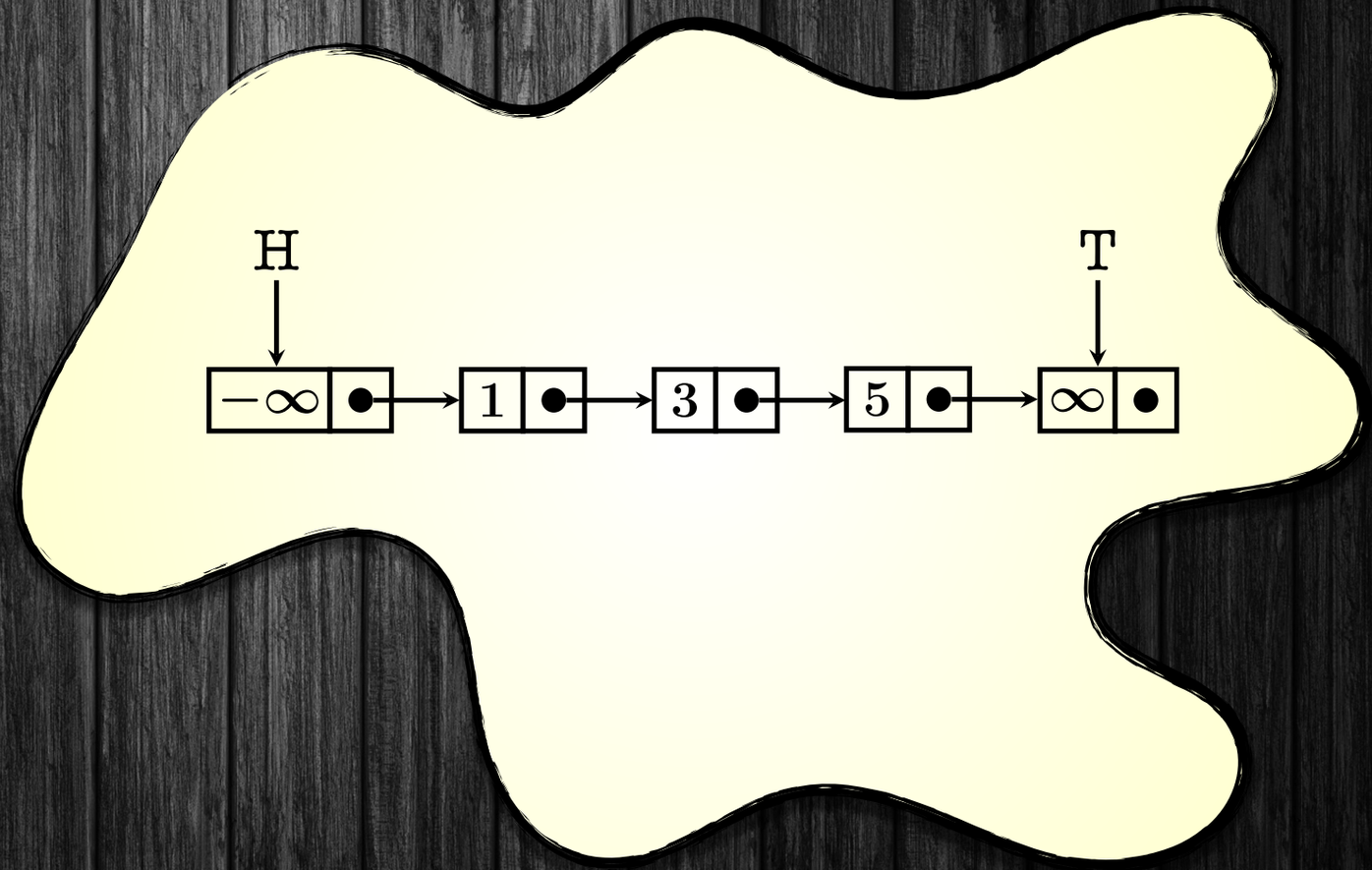


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

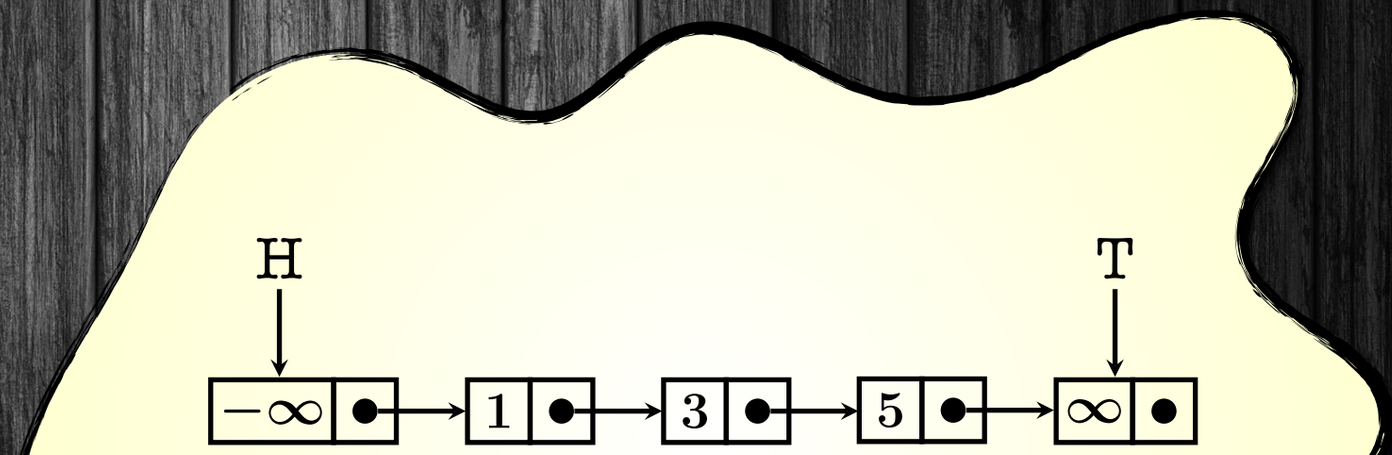




Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

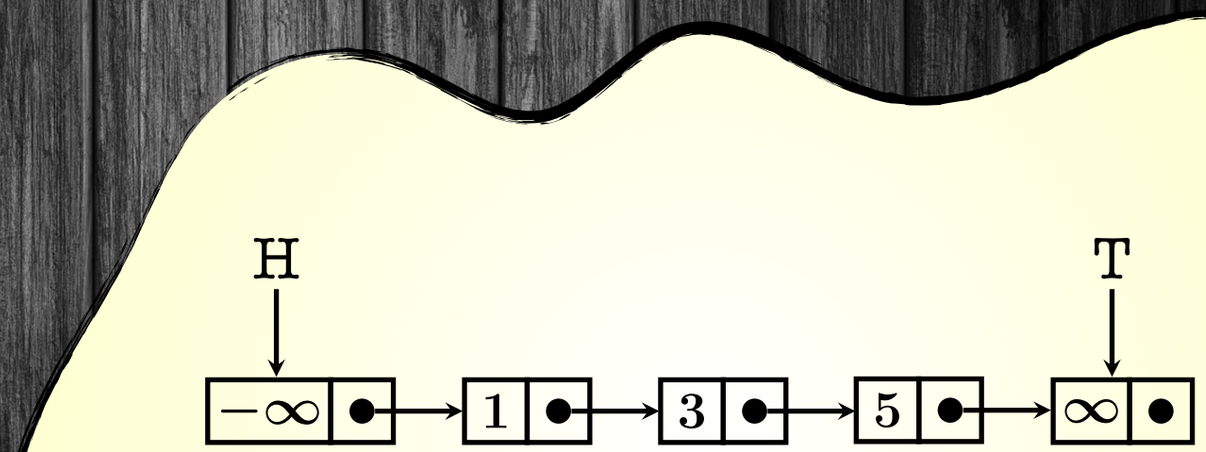


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(4)

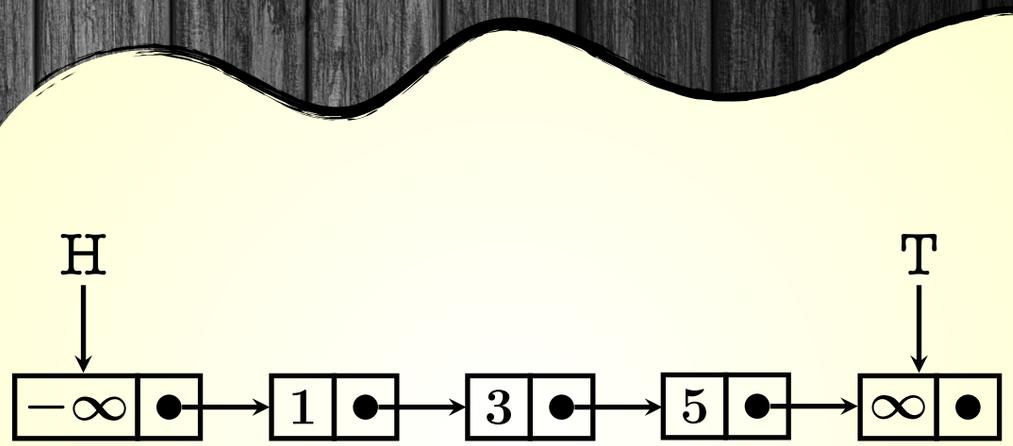


Sets

Sequential Programs

```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

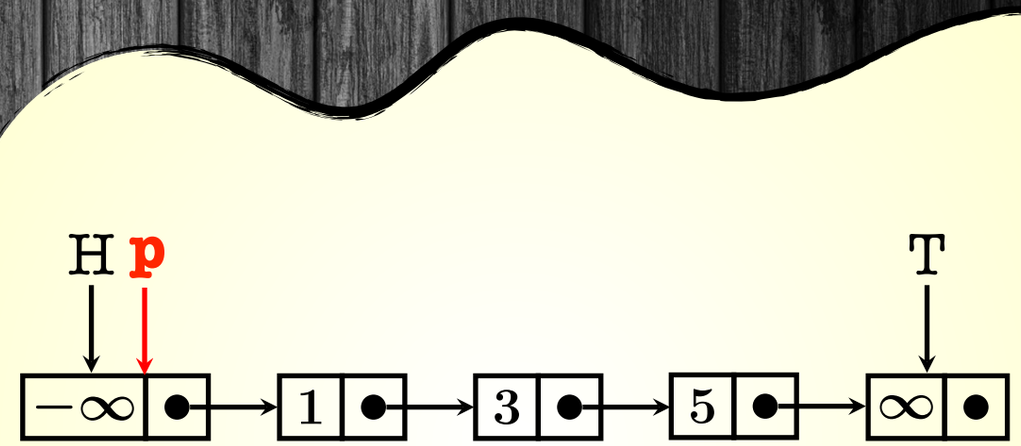
add(4)



Sets | Sequential Programs

```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

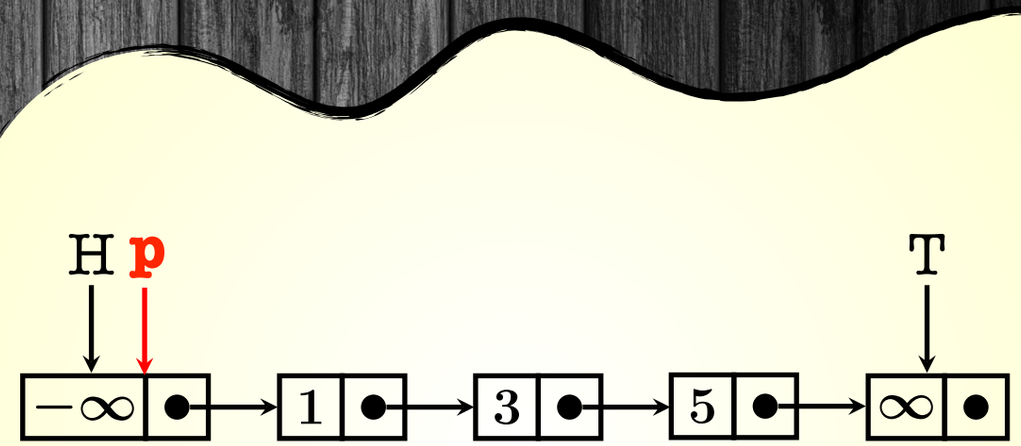
add(4)



Sets | Sequential Programs

```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

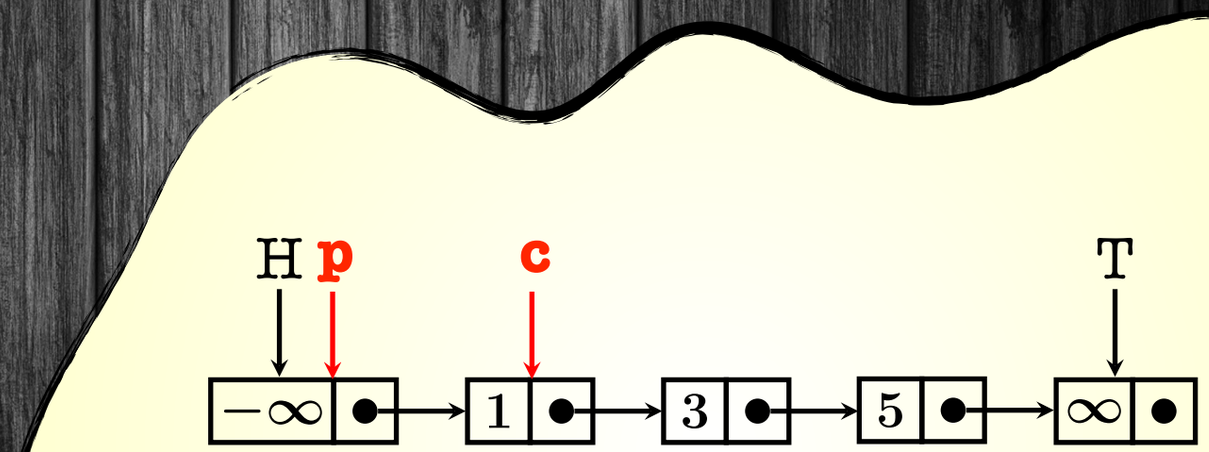
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

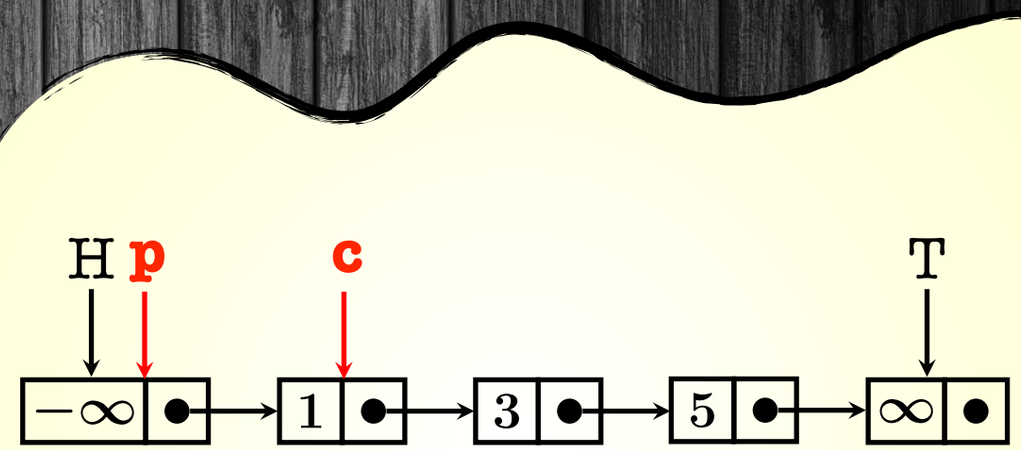
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

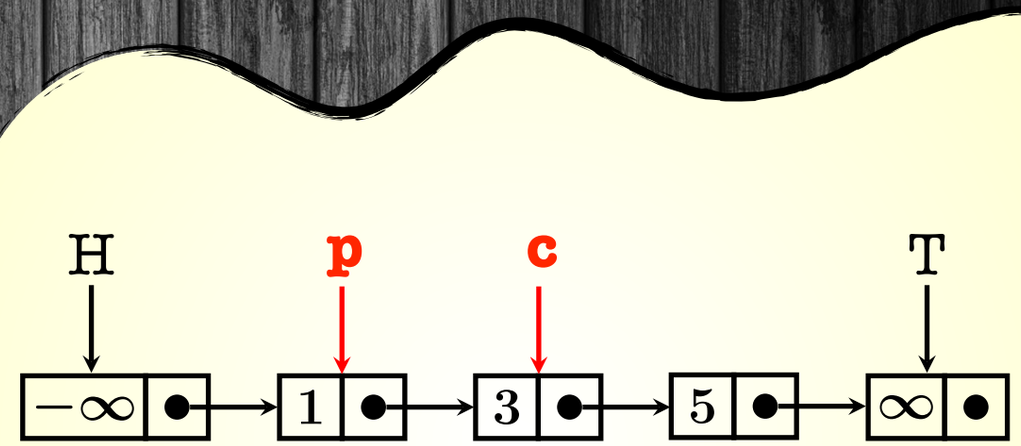
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

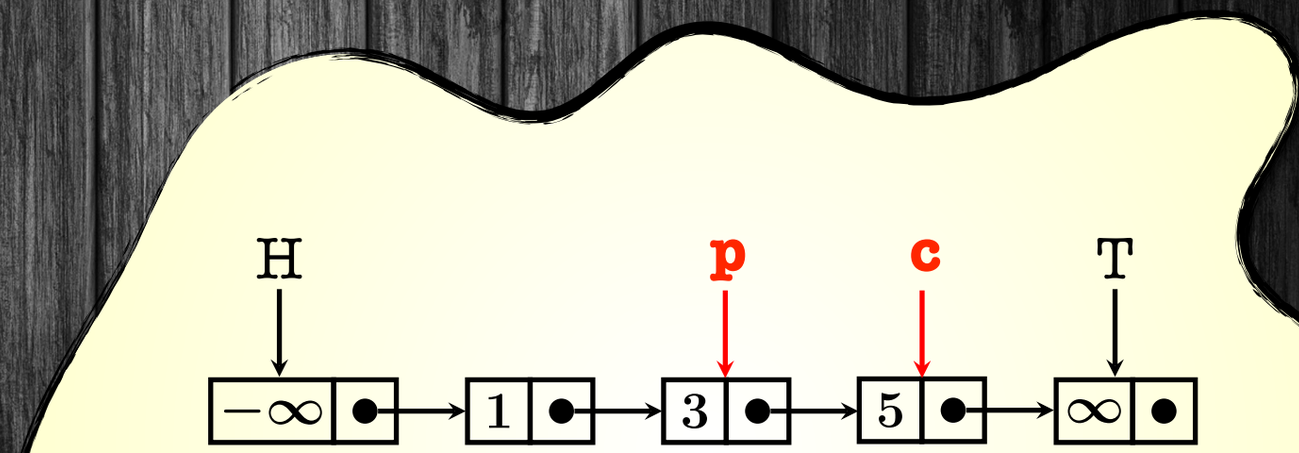
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

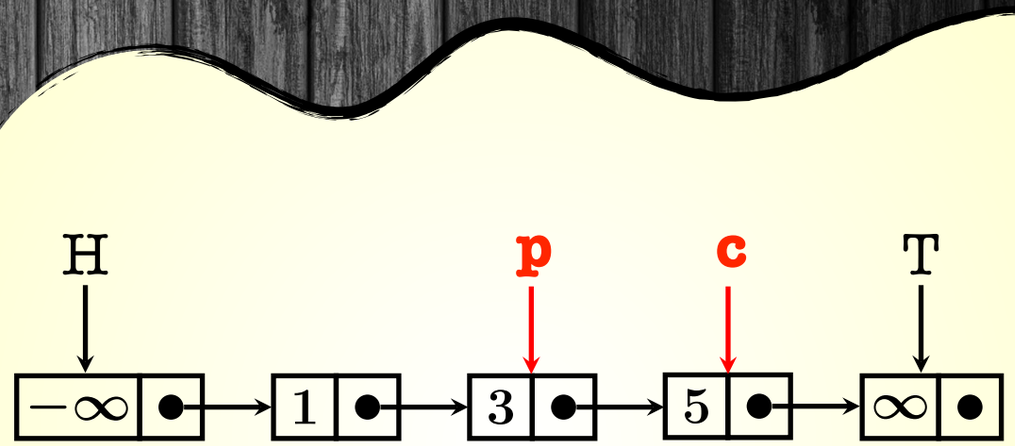
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(4)

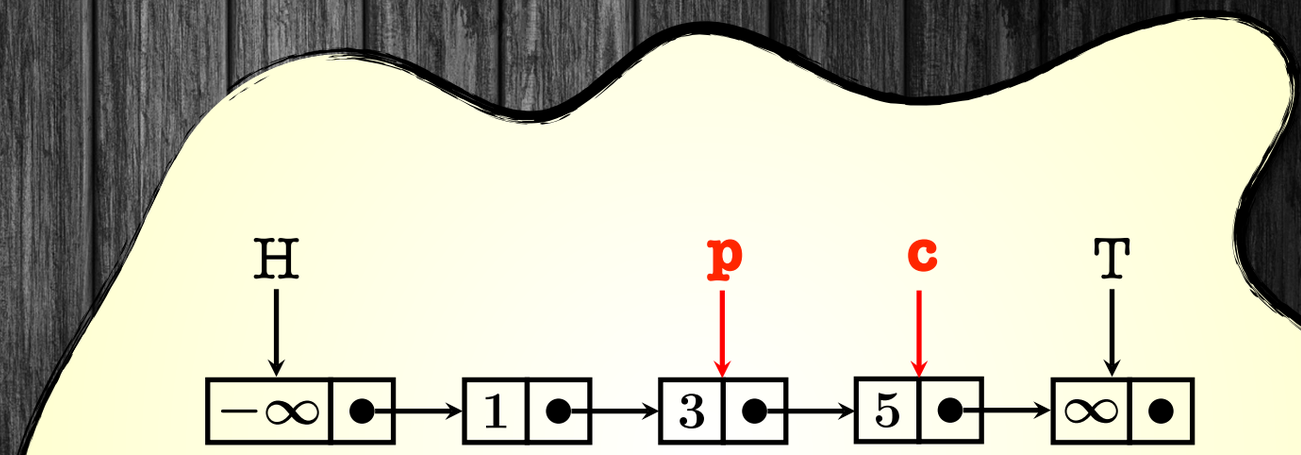


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(4)

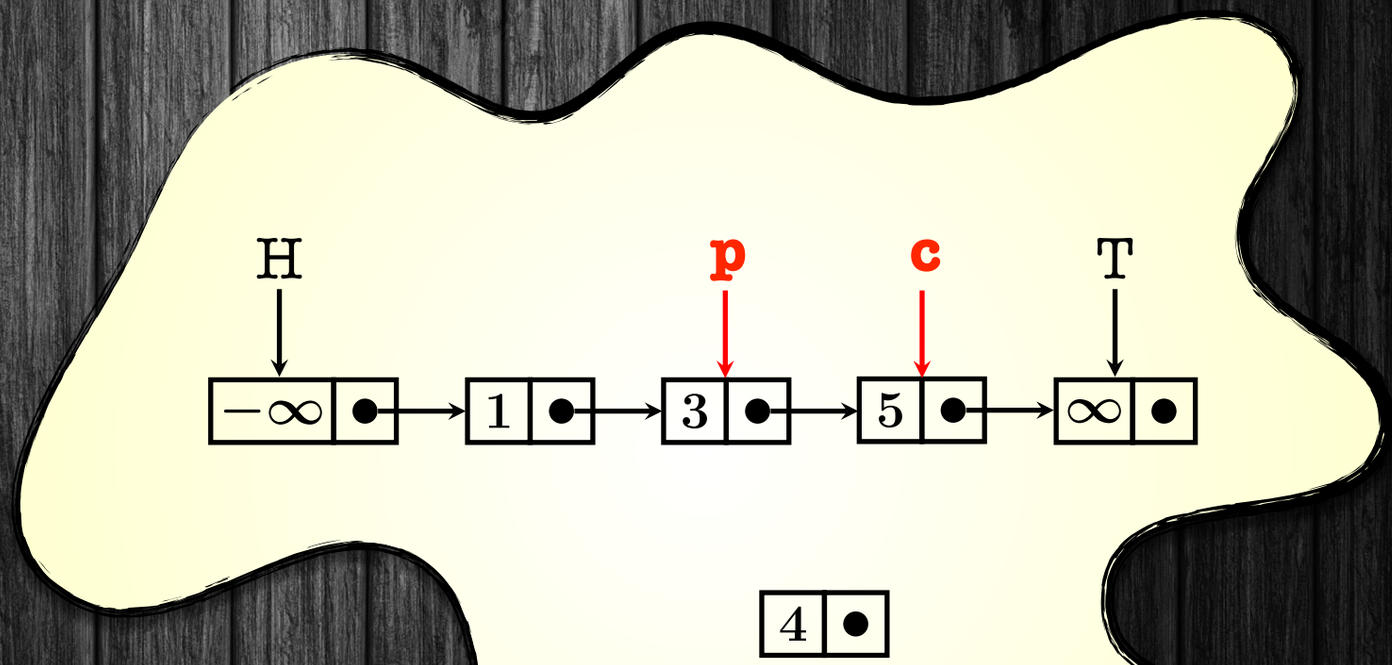


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

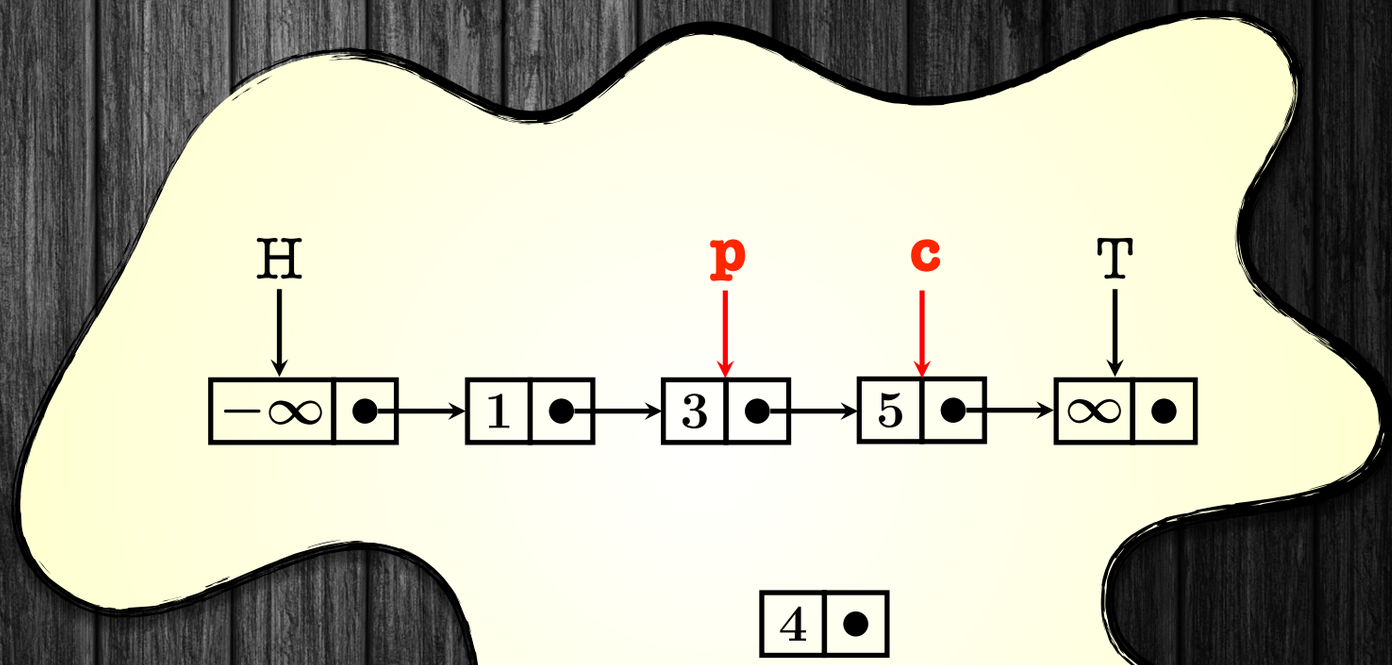
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(4)

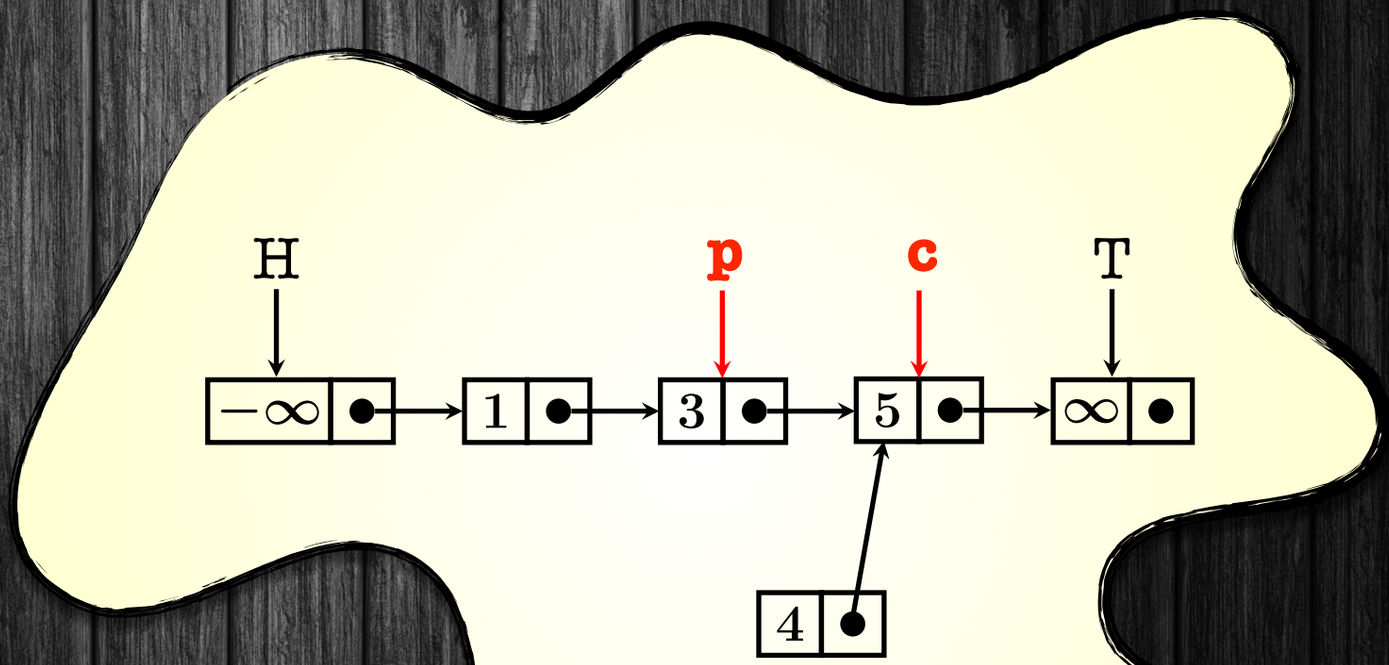


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

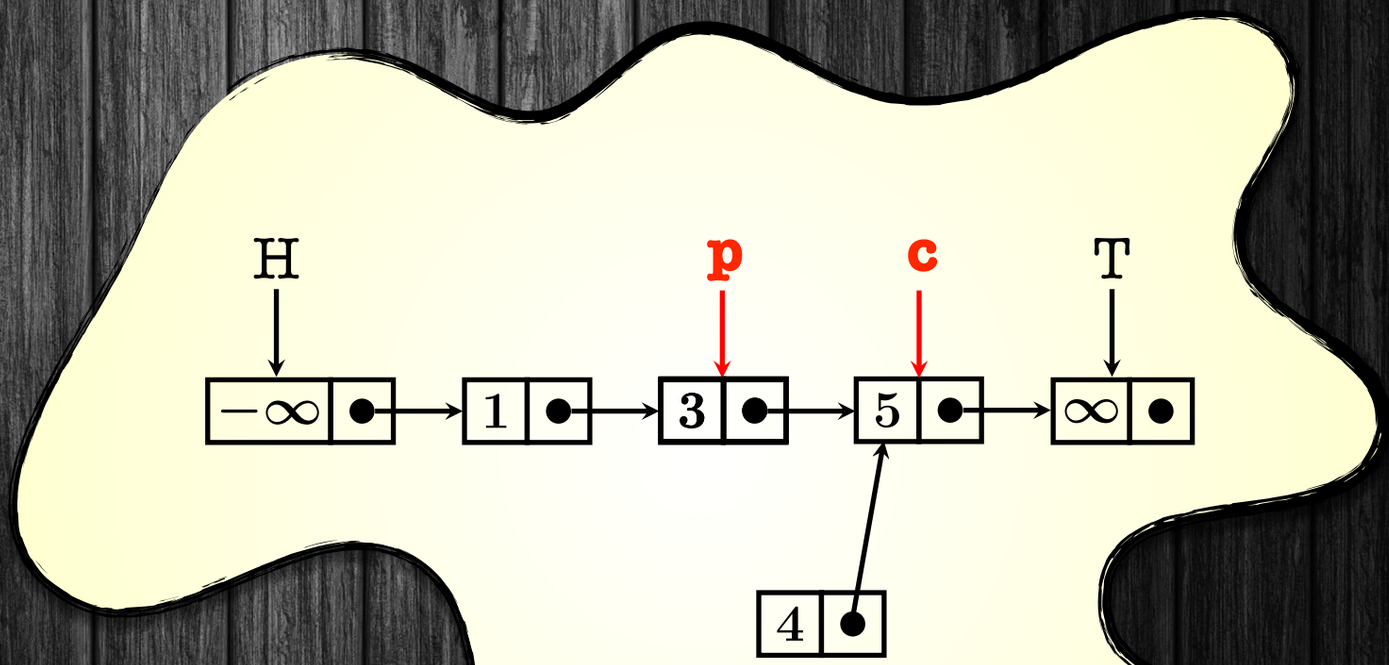
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(4)

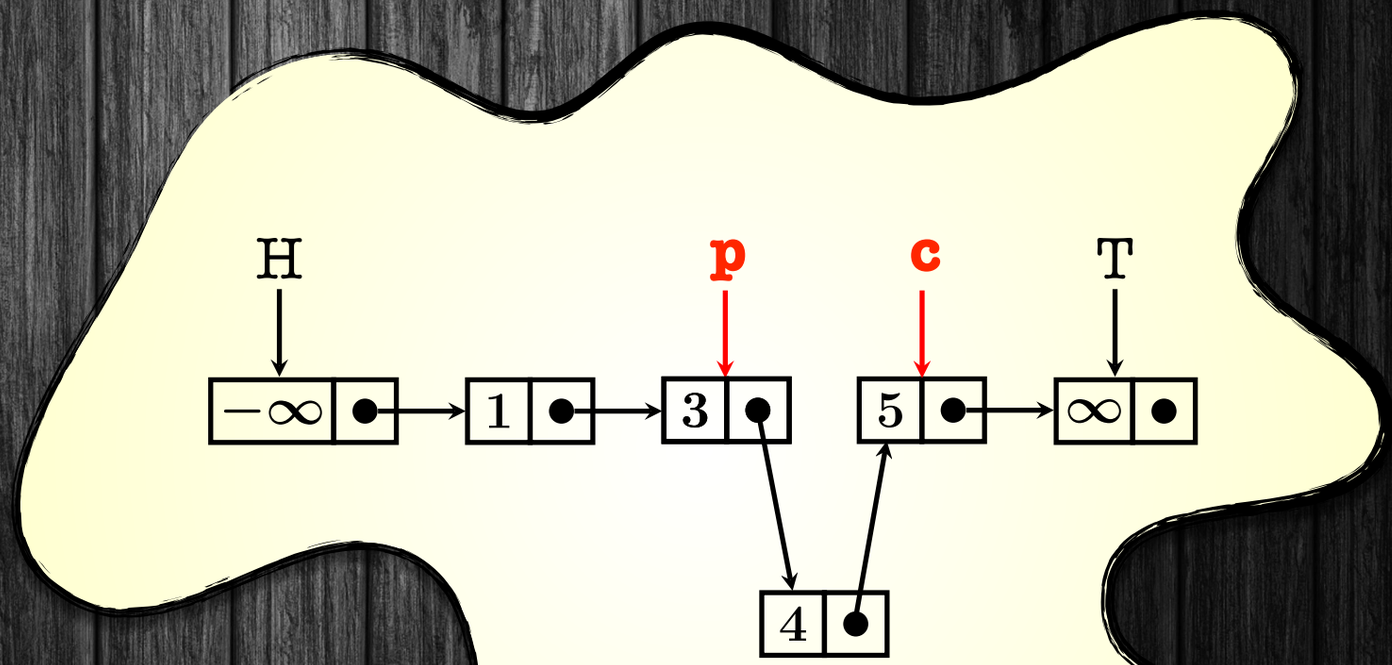


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

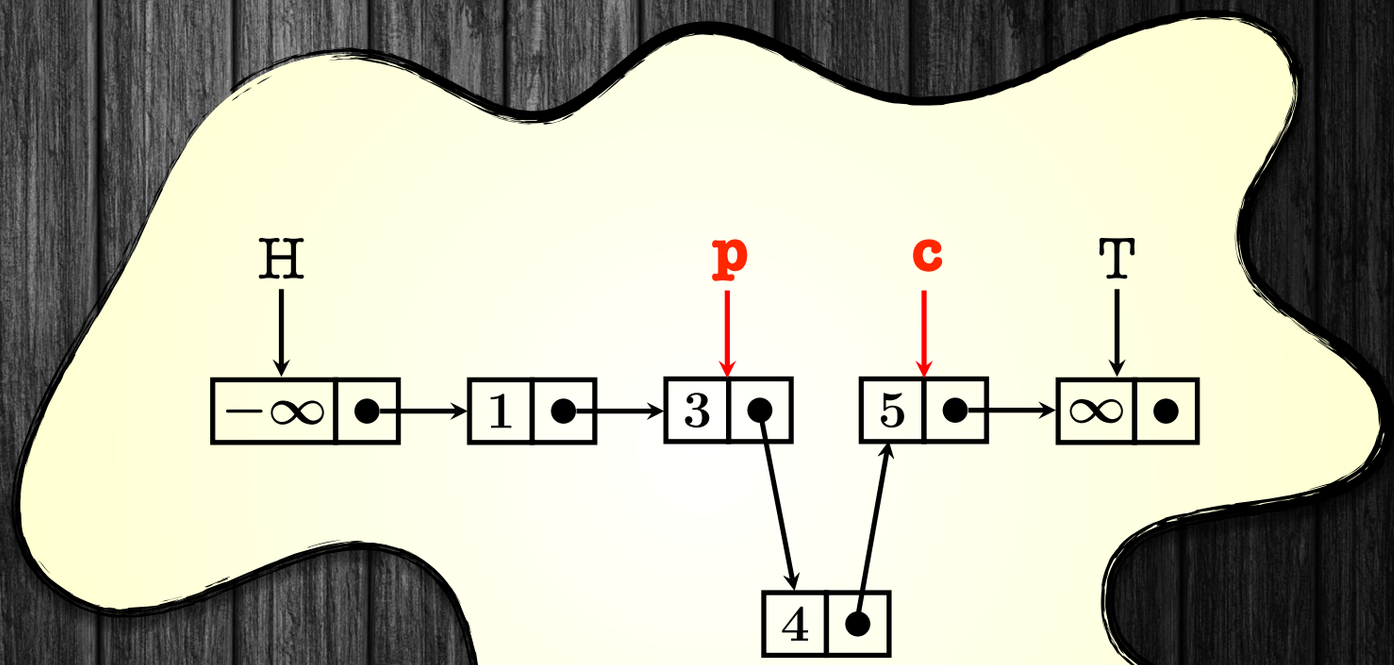
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

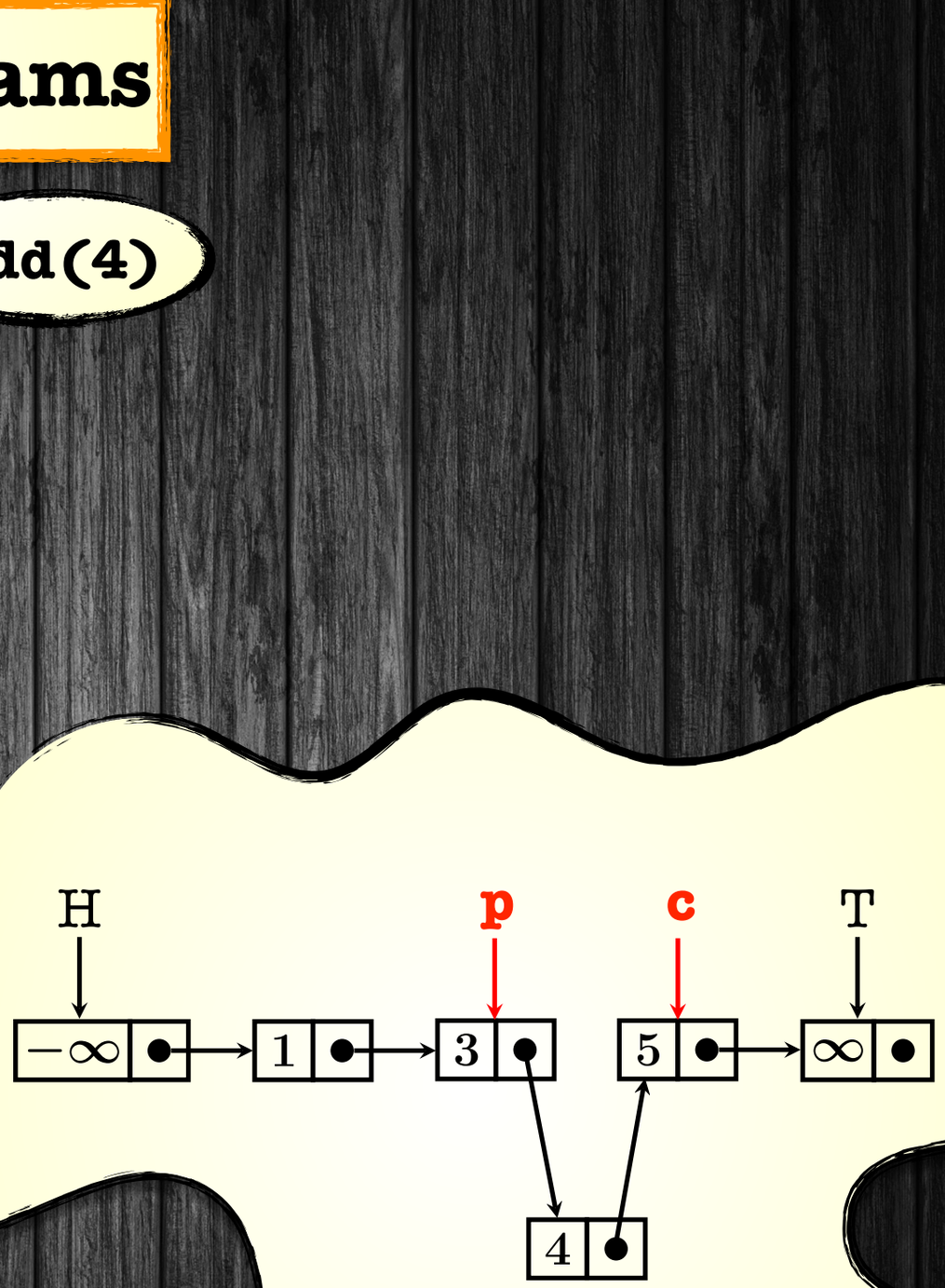
add(4)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

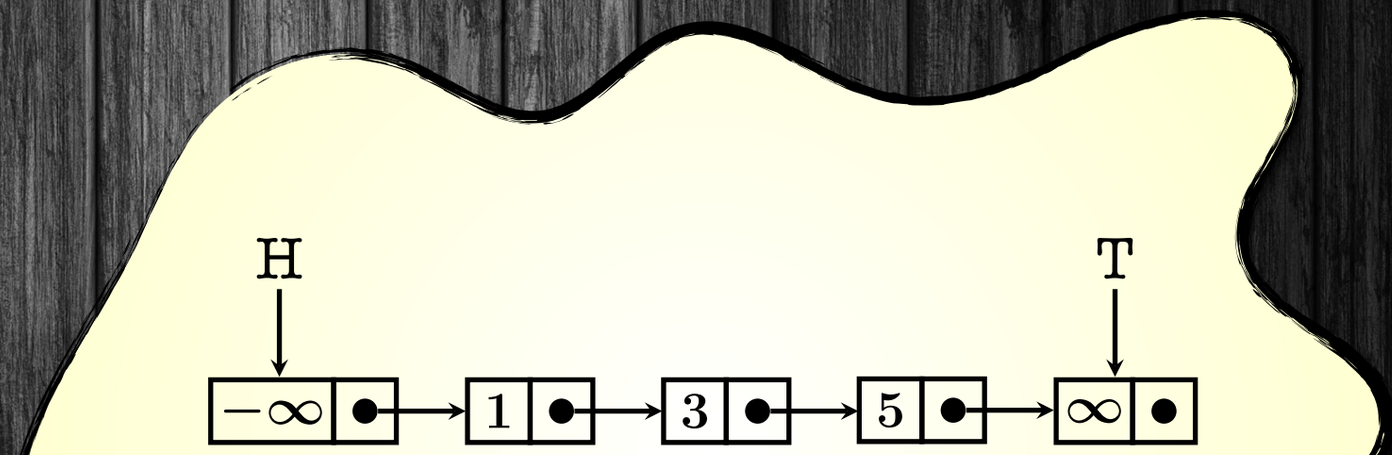
add(4)



Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

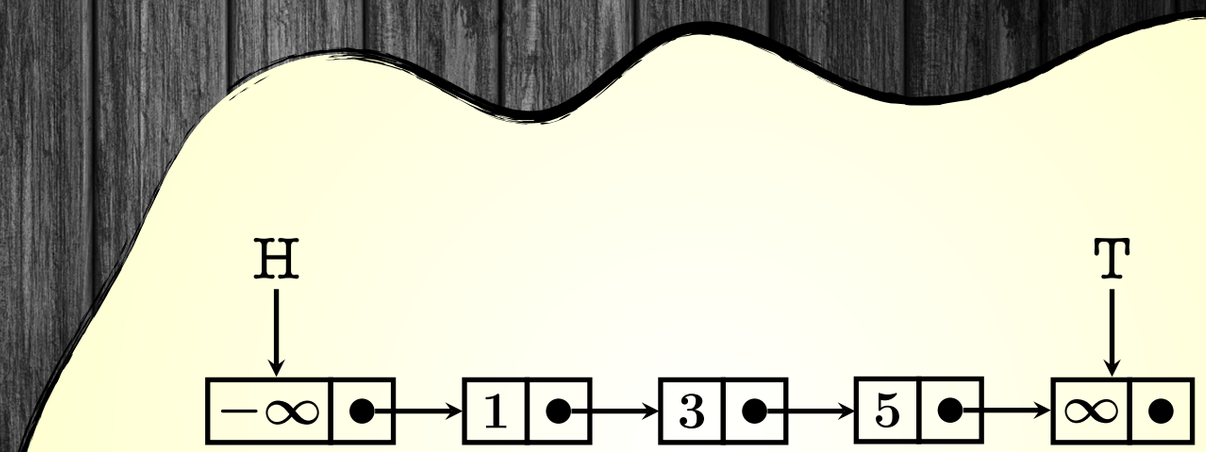


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(5)

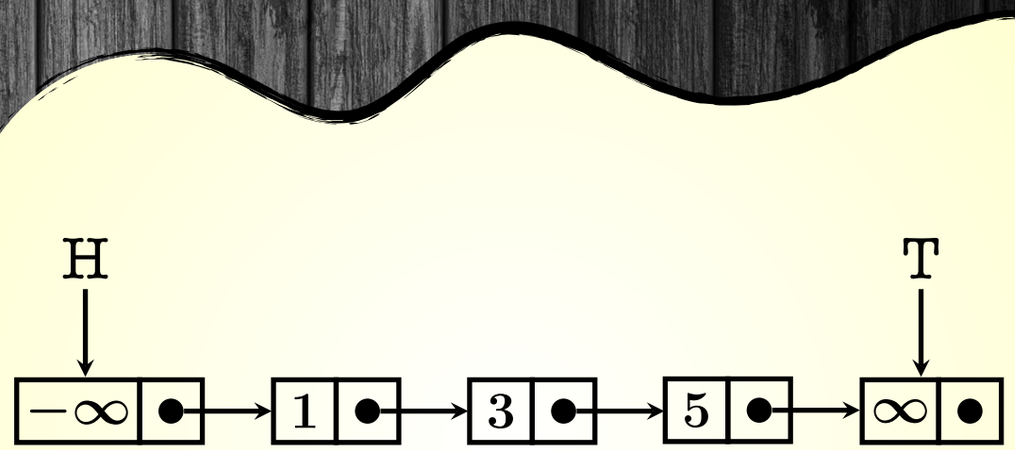


Sets

Sequential Programs

```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

add(5)

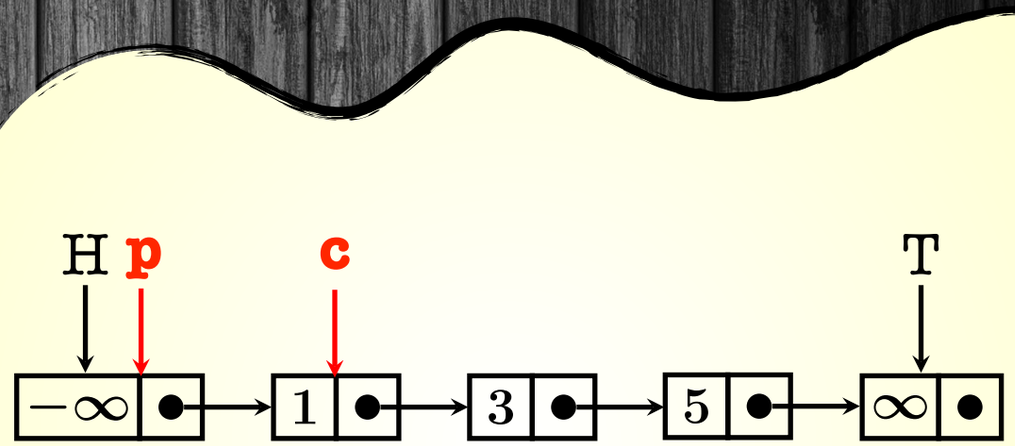


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(5)

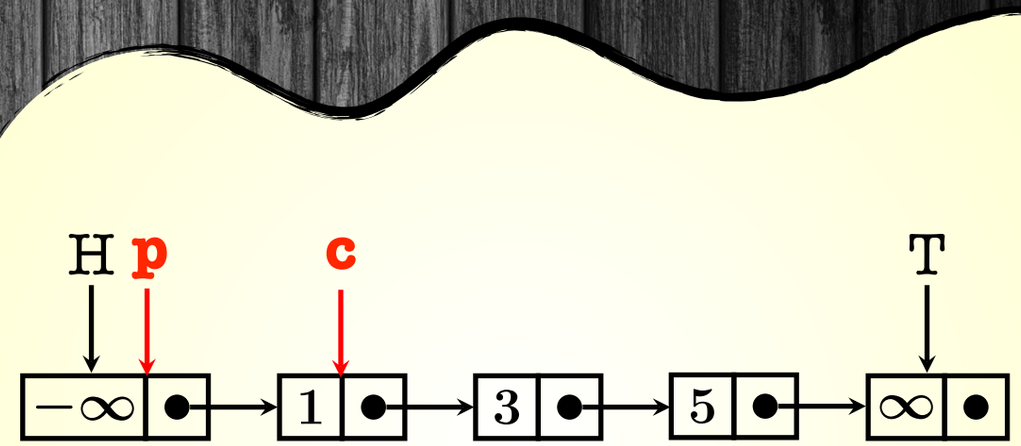


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

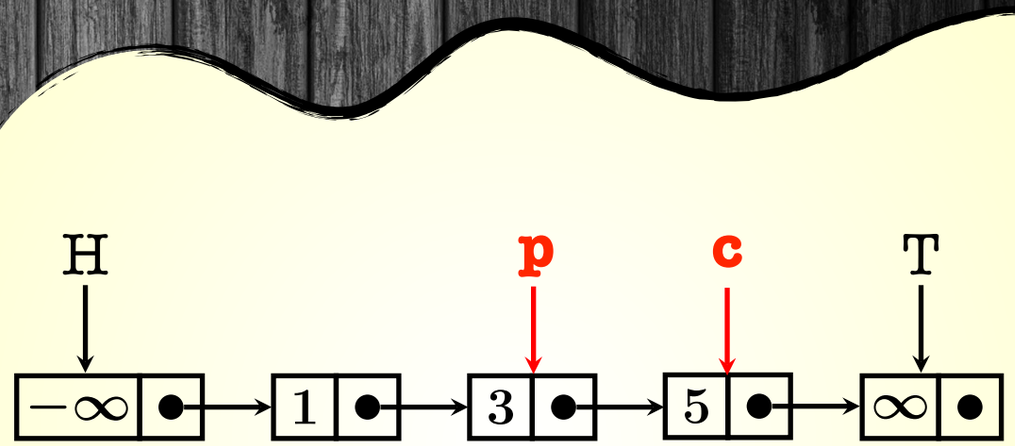
add(5)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

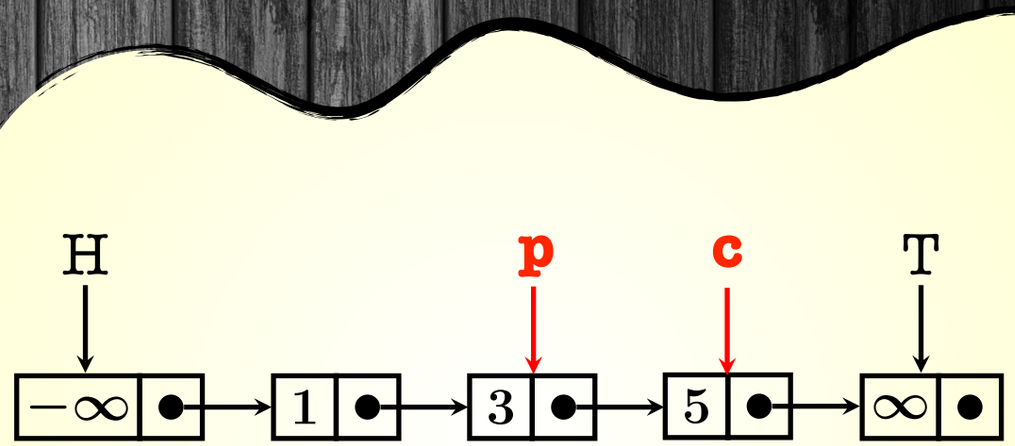
add(5)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

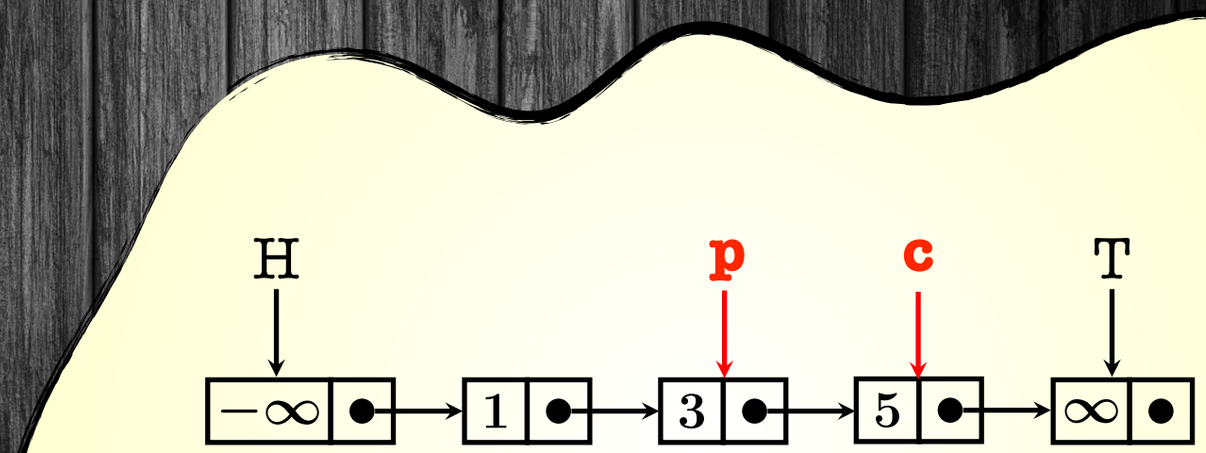
add(5)



Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(5)

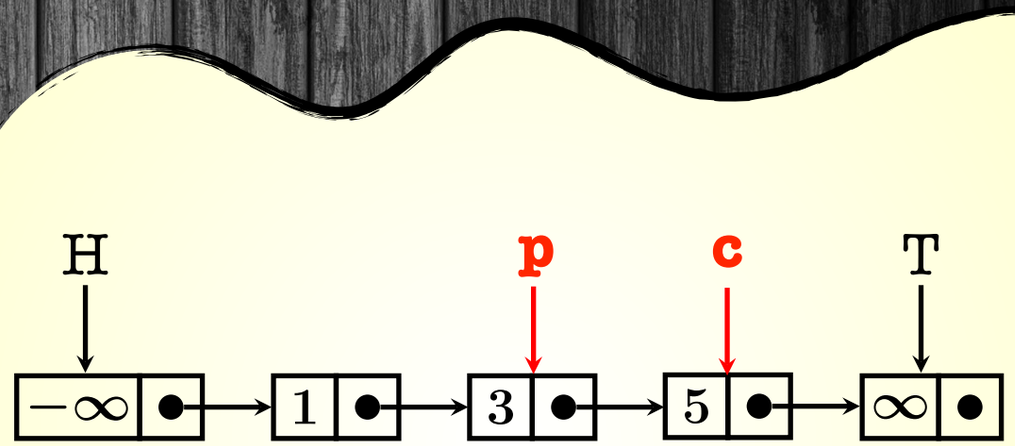


Sets

Sequential Programs

```
add(k):
Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

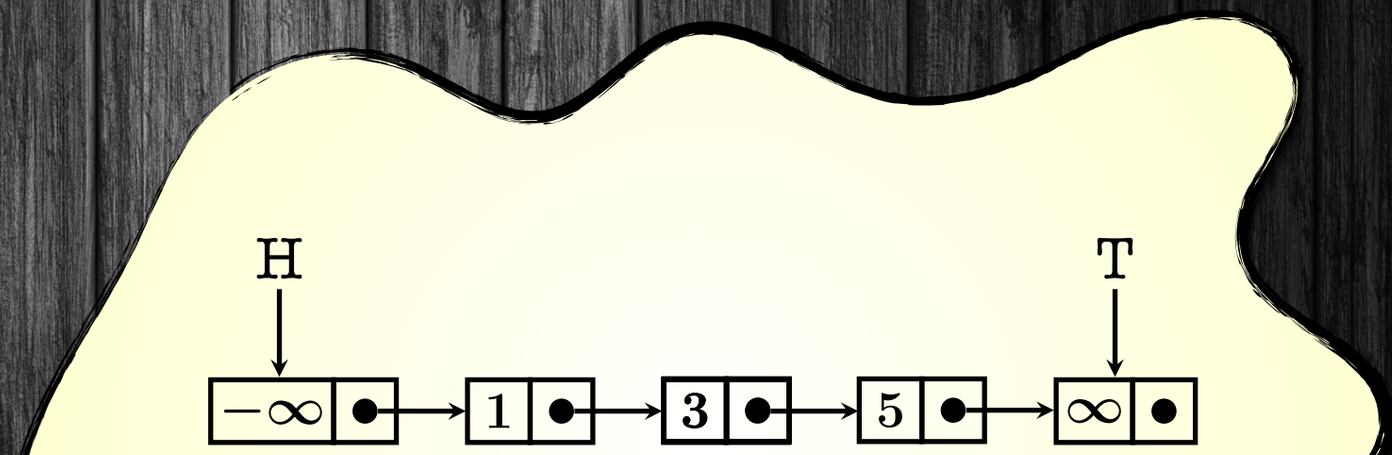
add(5)



Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

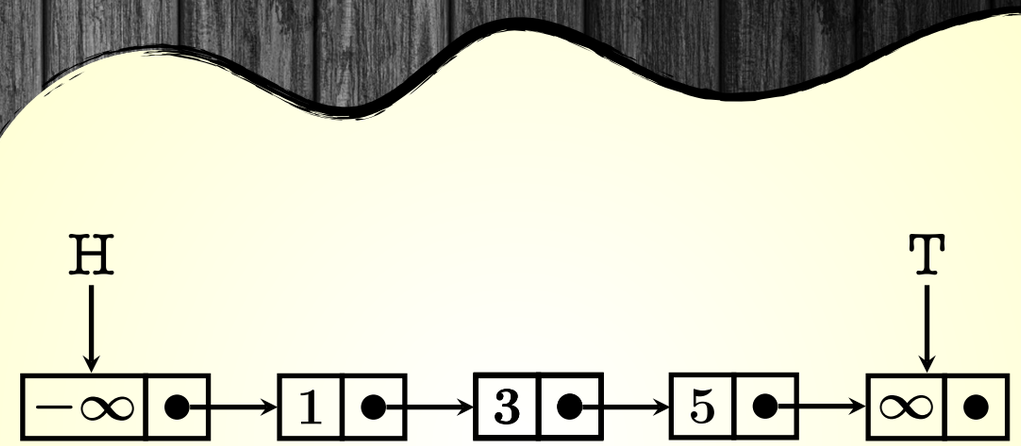


Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

add(4)



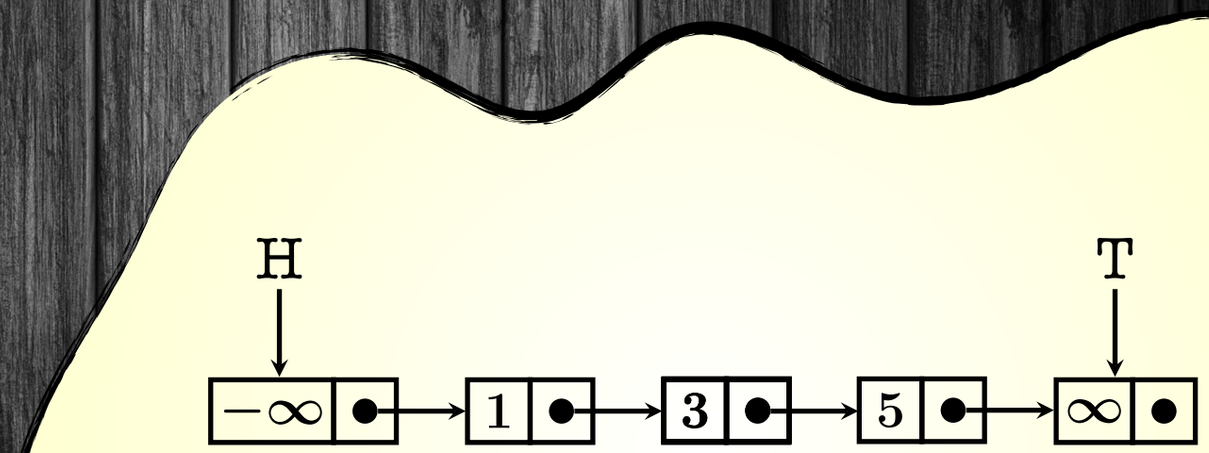
Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

add(4)

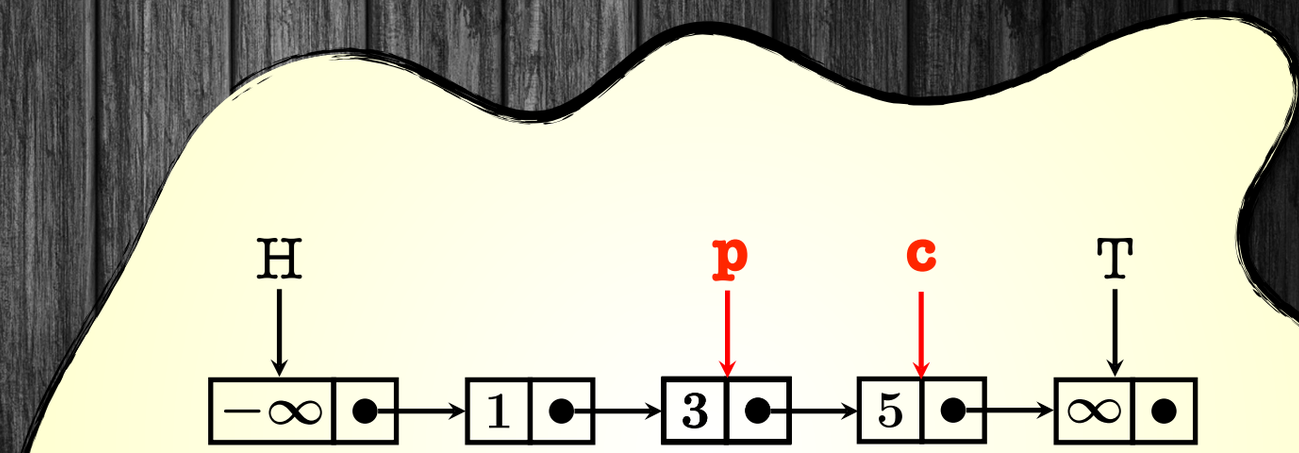


Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

add(4)

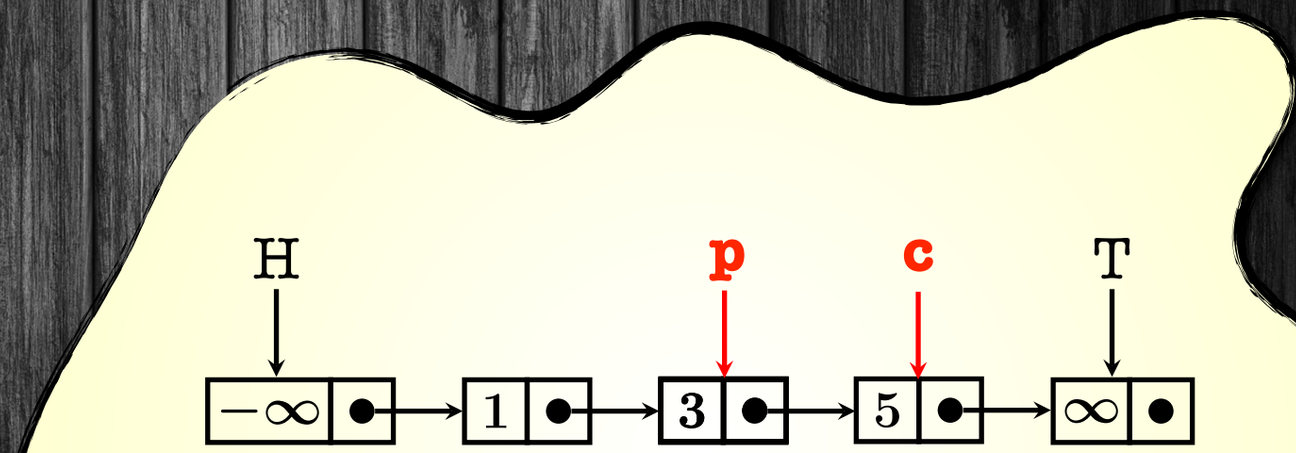


Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

add(4)

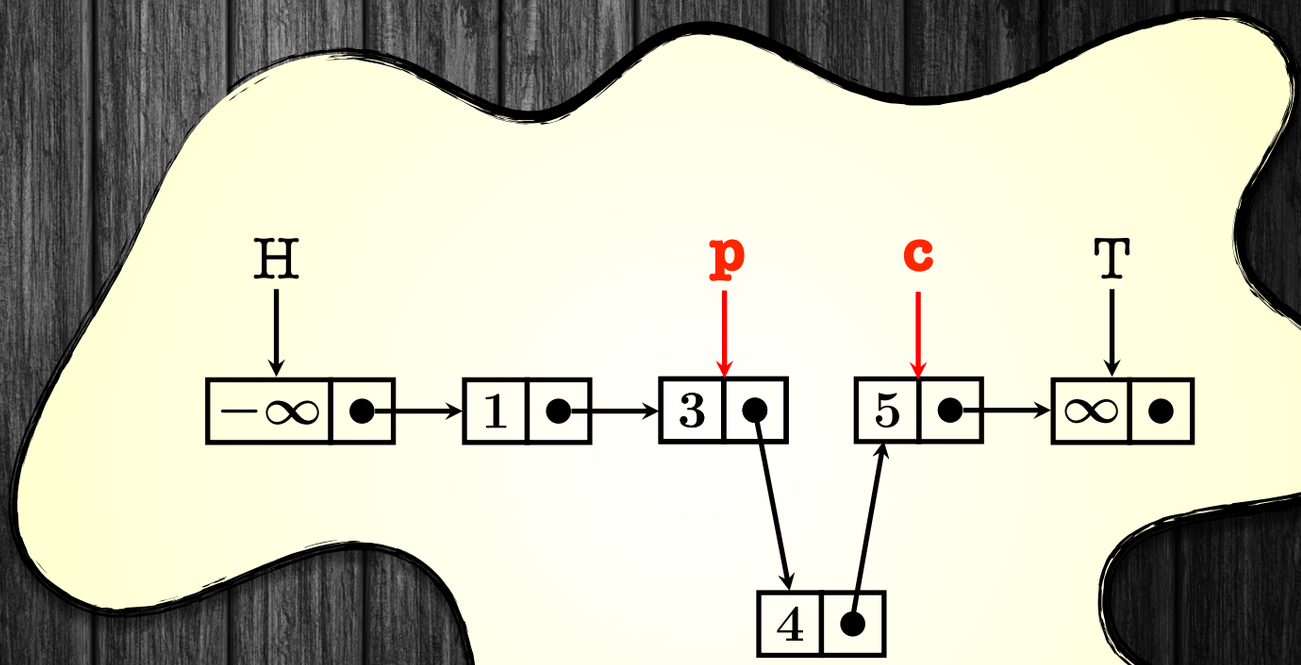


Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

add(4)

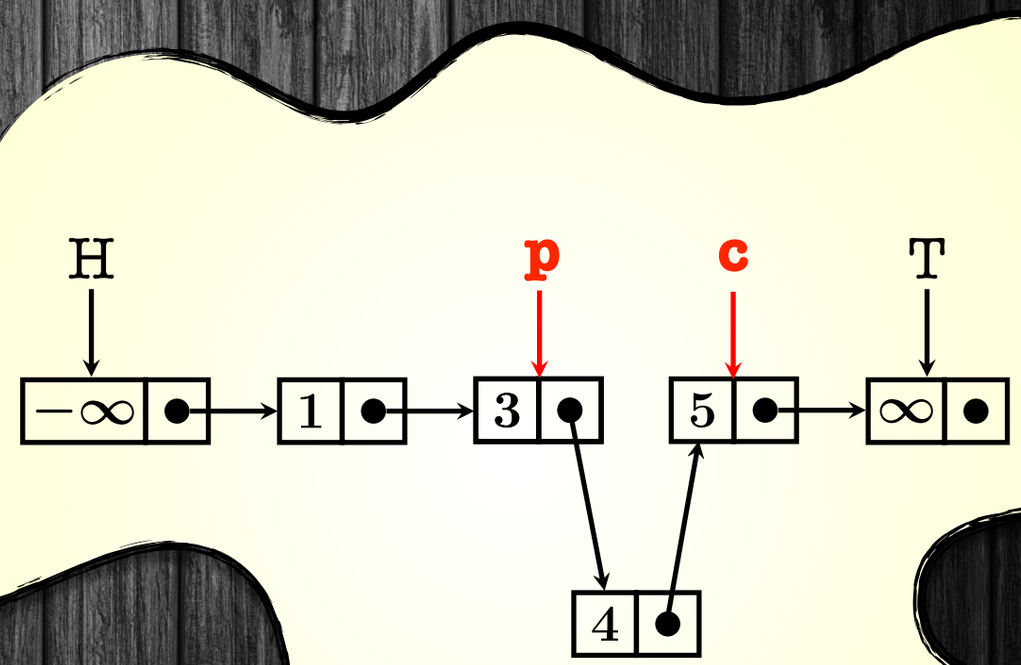


Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

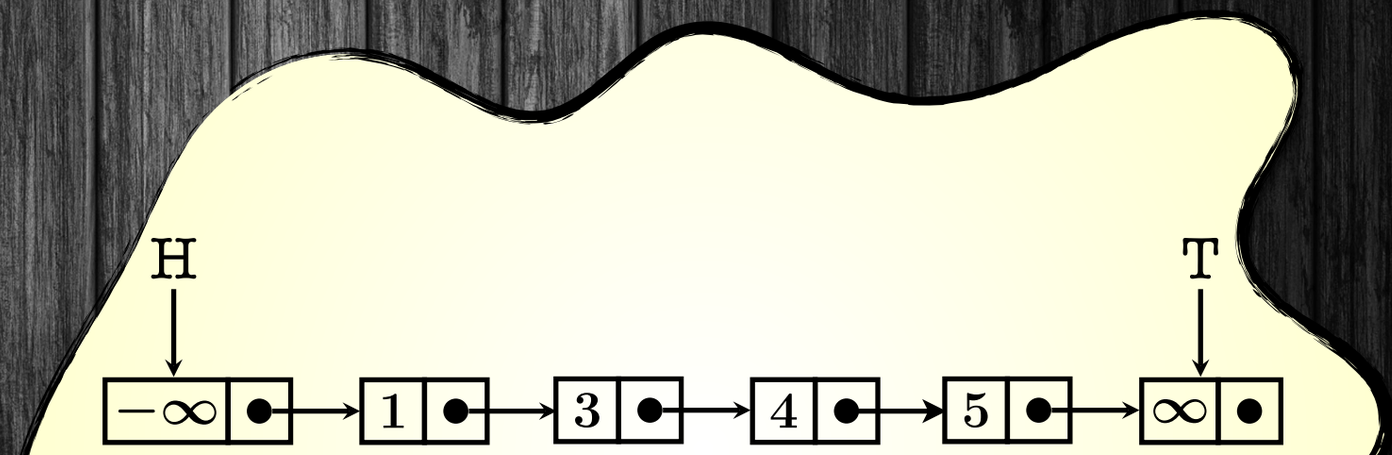
add(4)



Sets

Sequential Programs

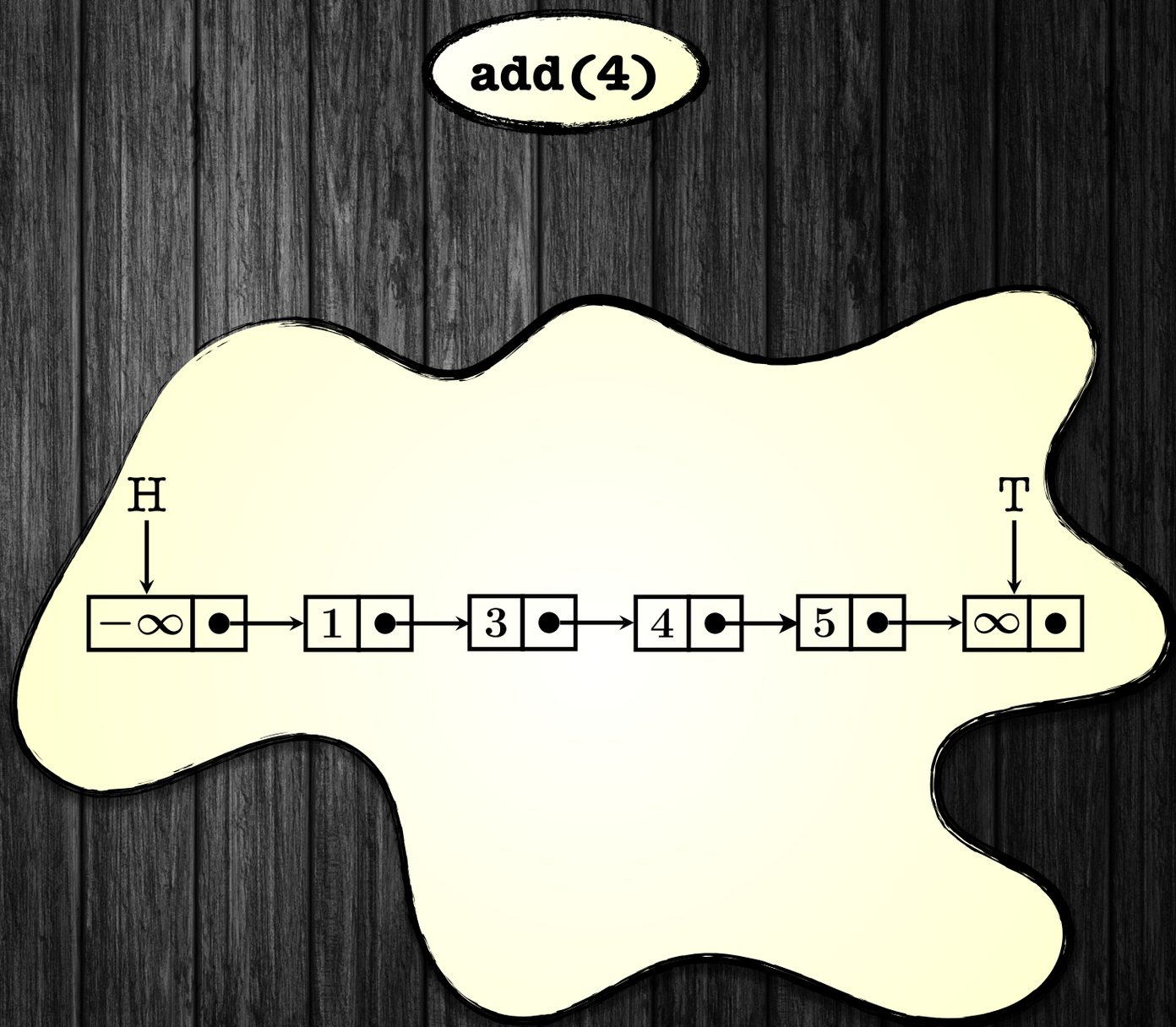
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```



Sets

Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

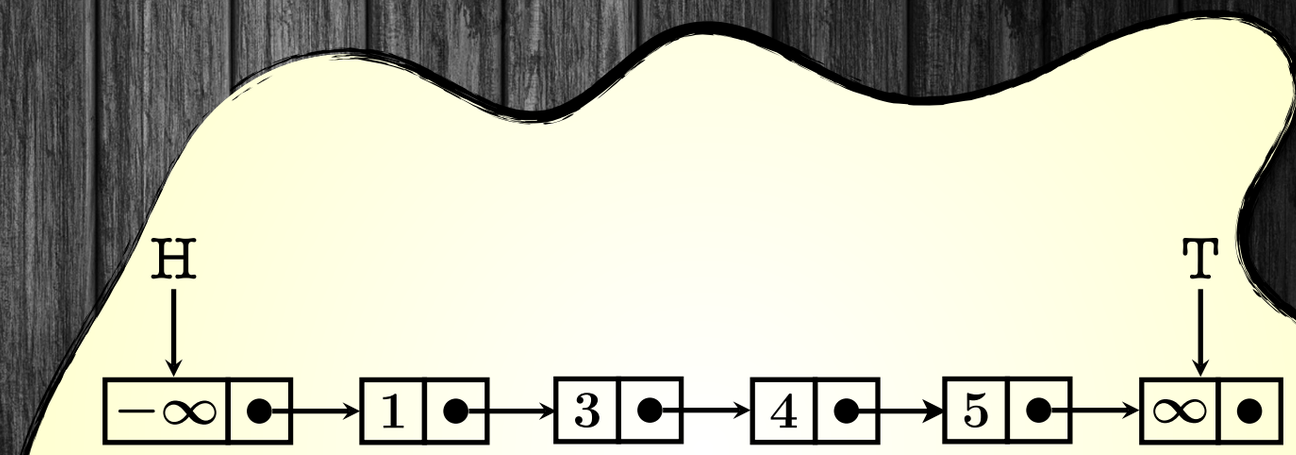


Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

add(4)

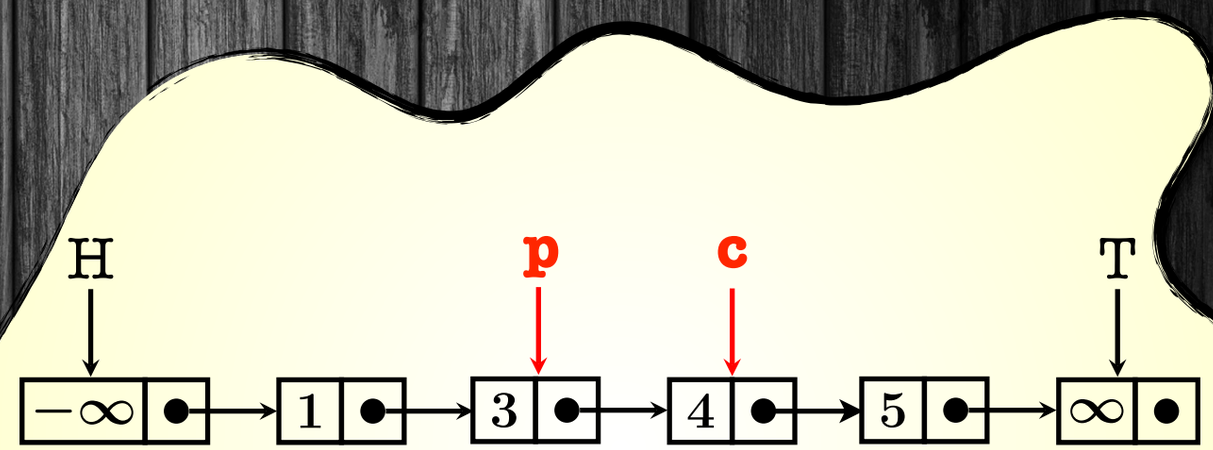


Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

add(4)

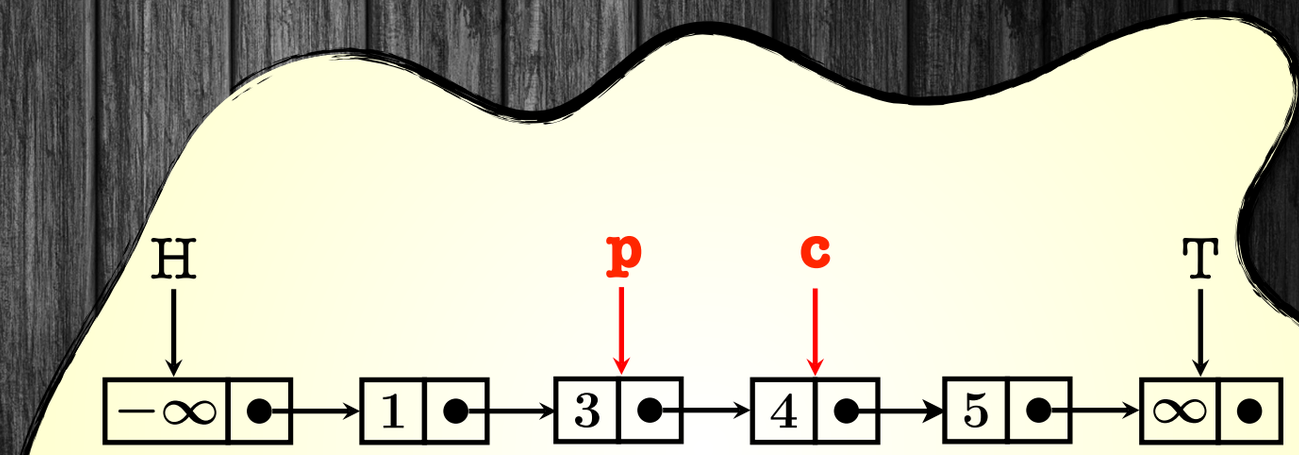


Sets | Sequential Programs

```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

add(4)

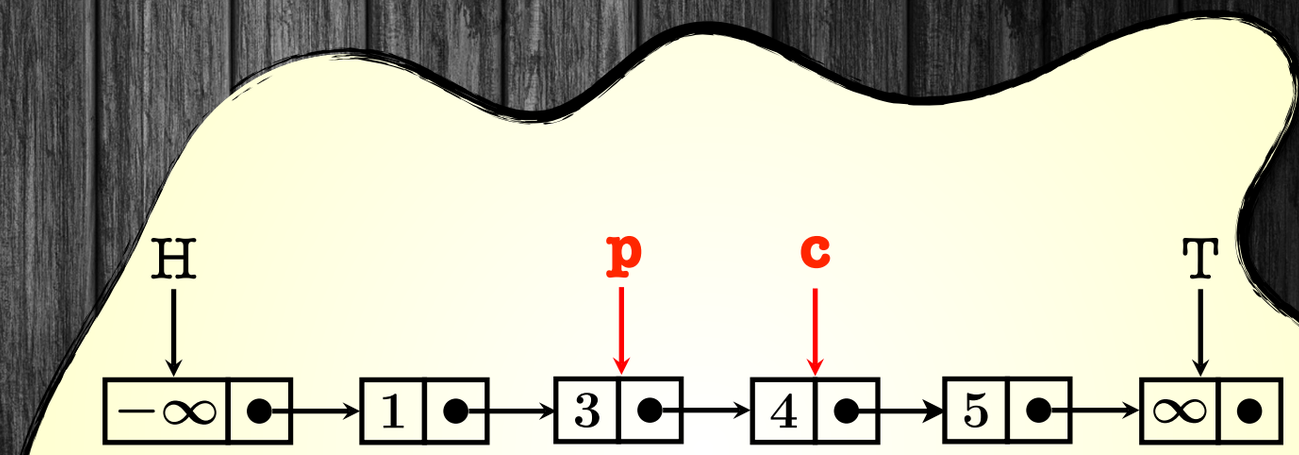


Sets | Sequential Programs

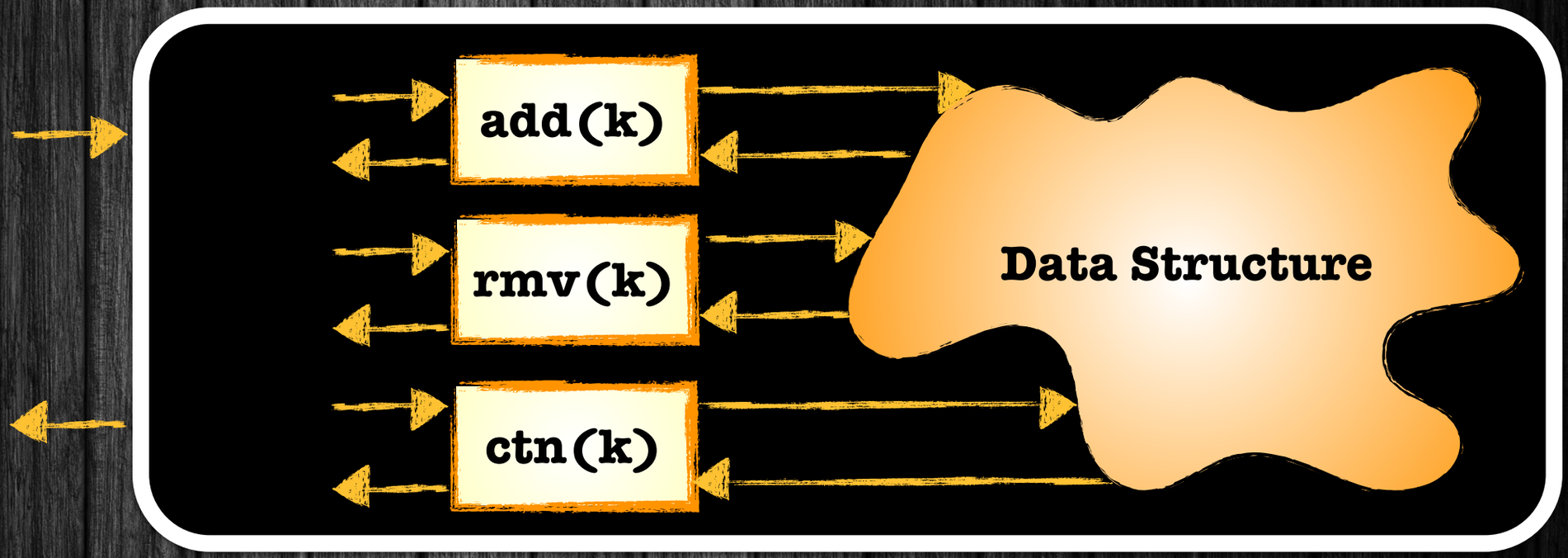
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

locate

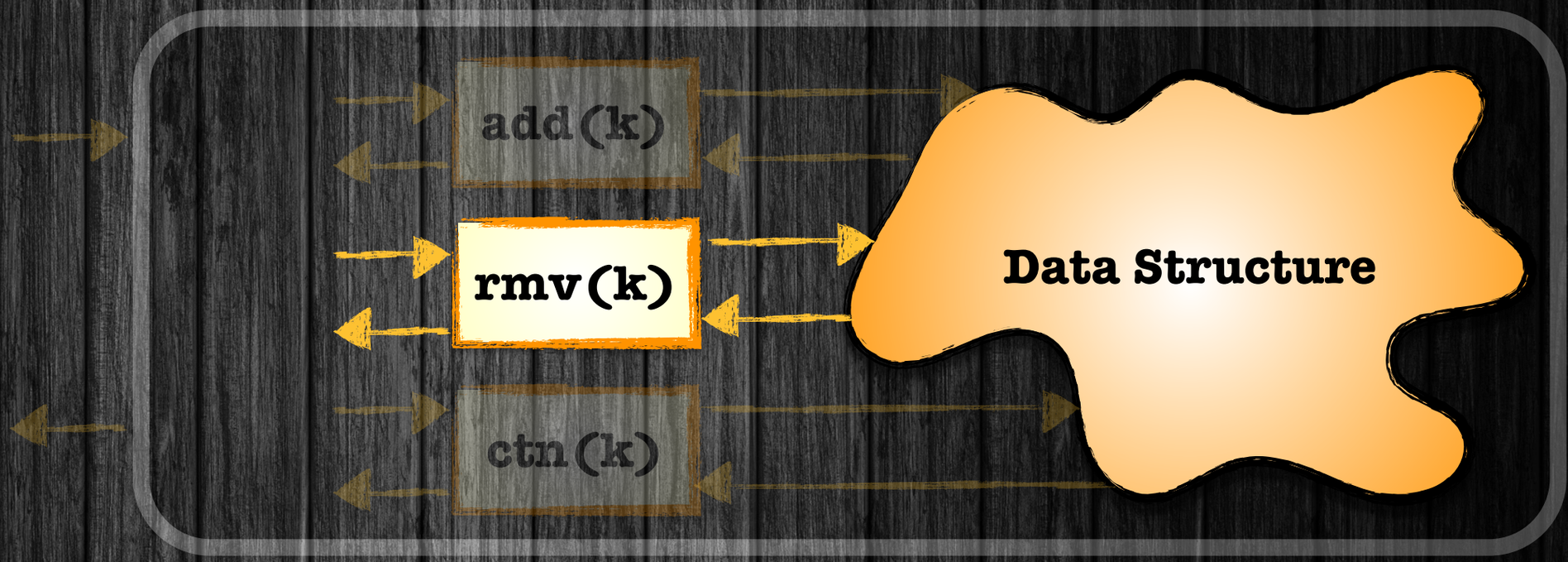
add(4)



Sets | Sequential Programs



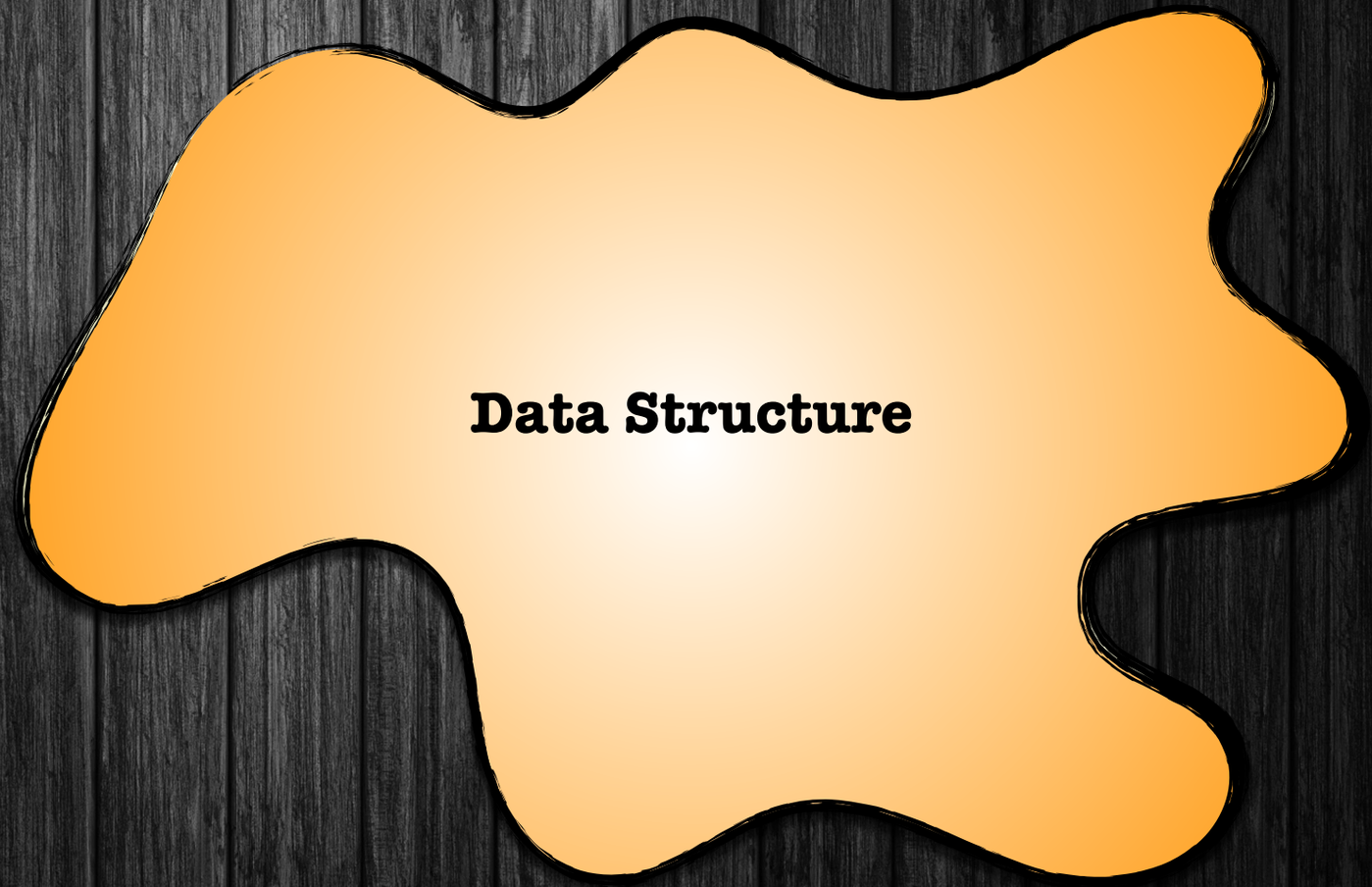
Sets | Sequential Programs



Sets

Sequential Programs

```
rmv(k)
```



Sets

Sequential Programs

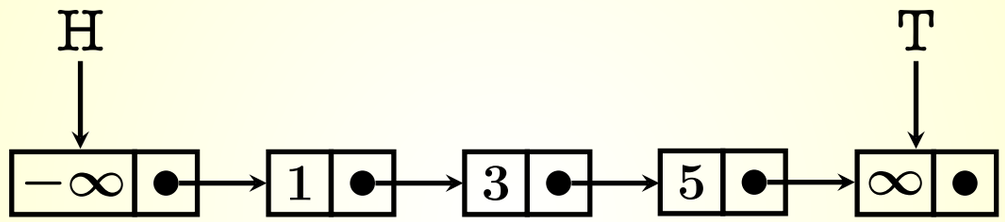
```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

Data Structure

Sets

Sequential Programs

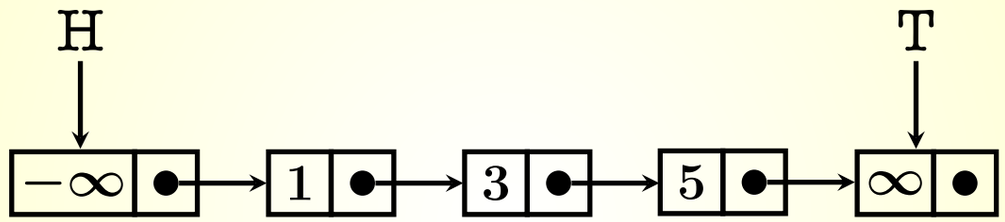
```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

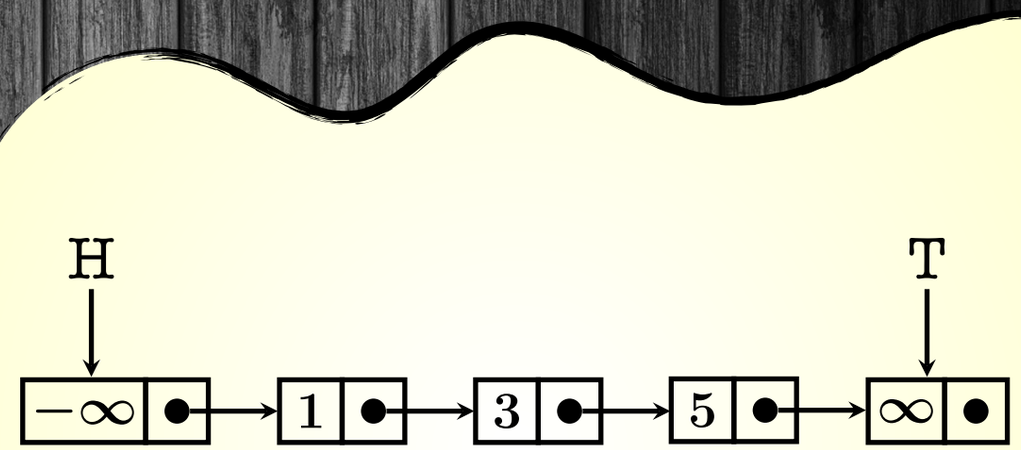


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

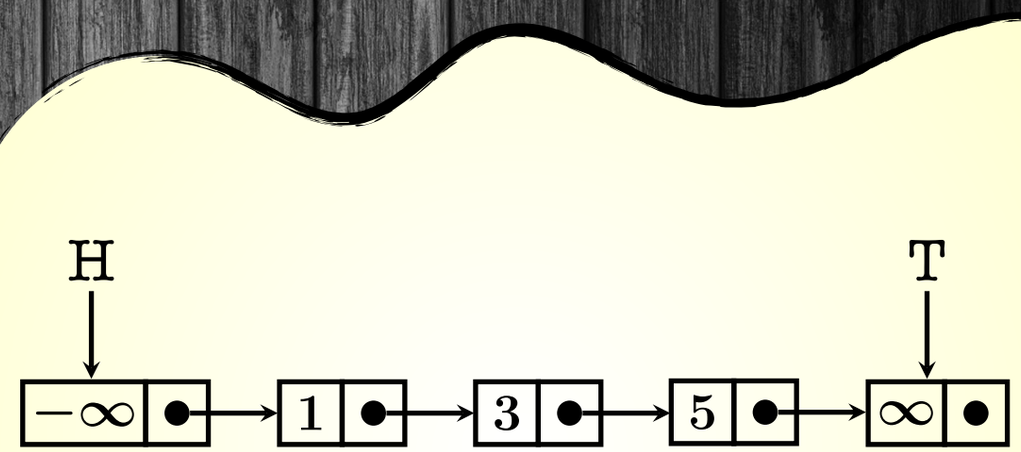


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

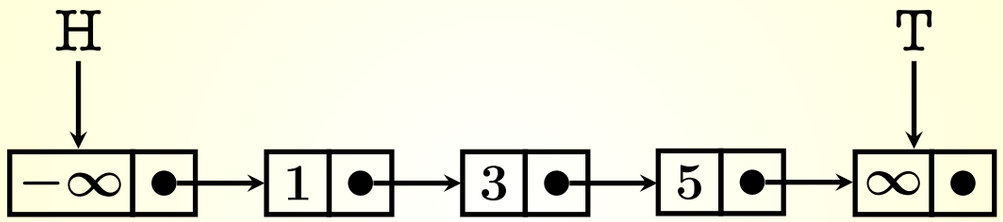


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

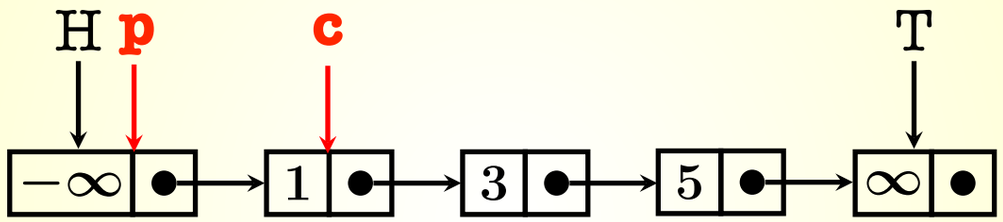


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

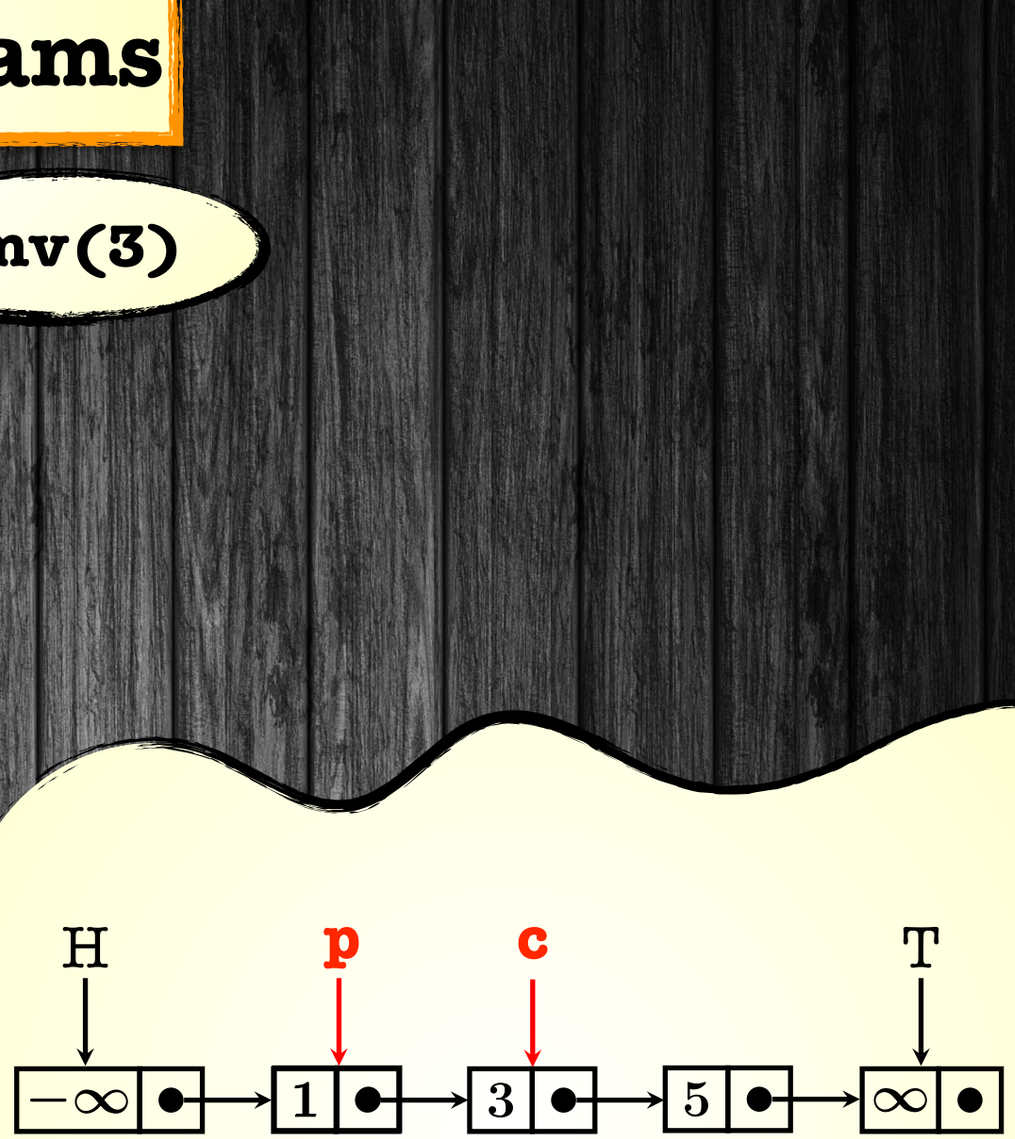


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

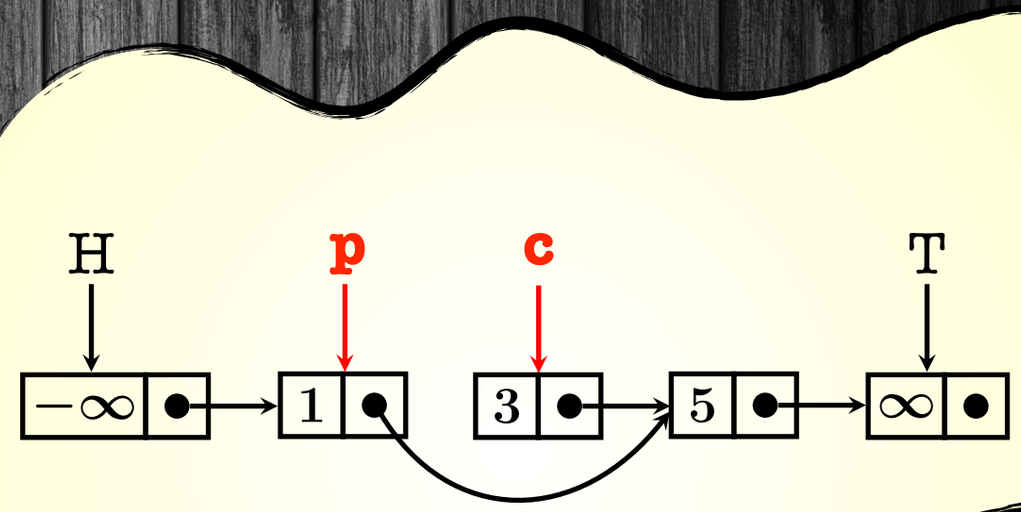


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

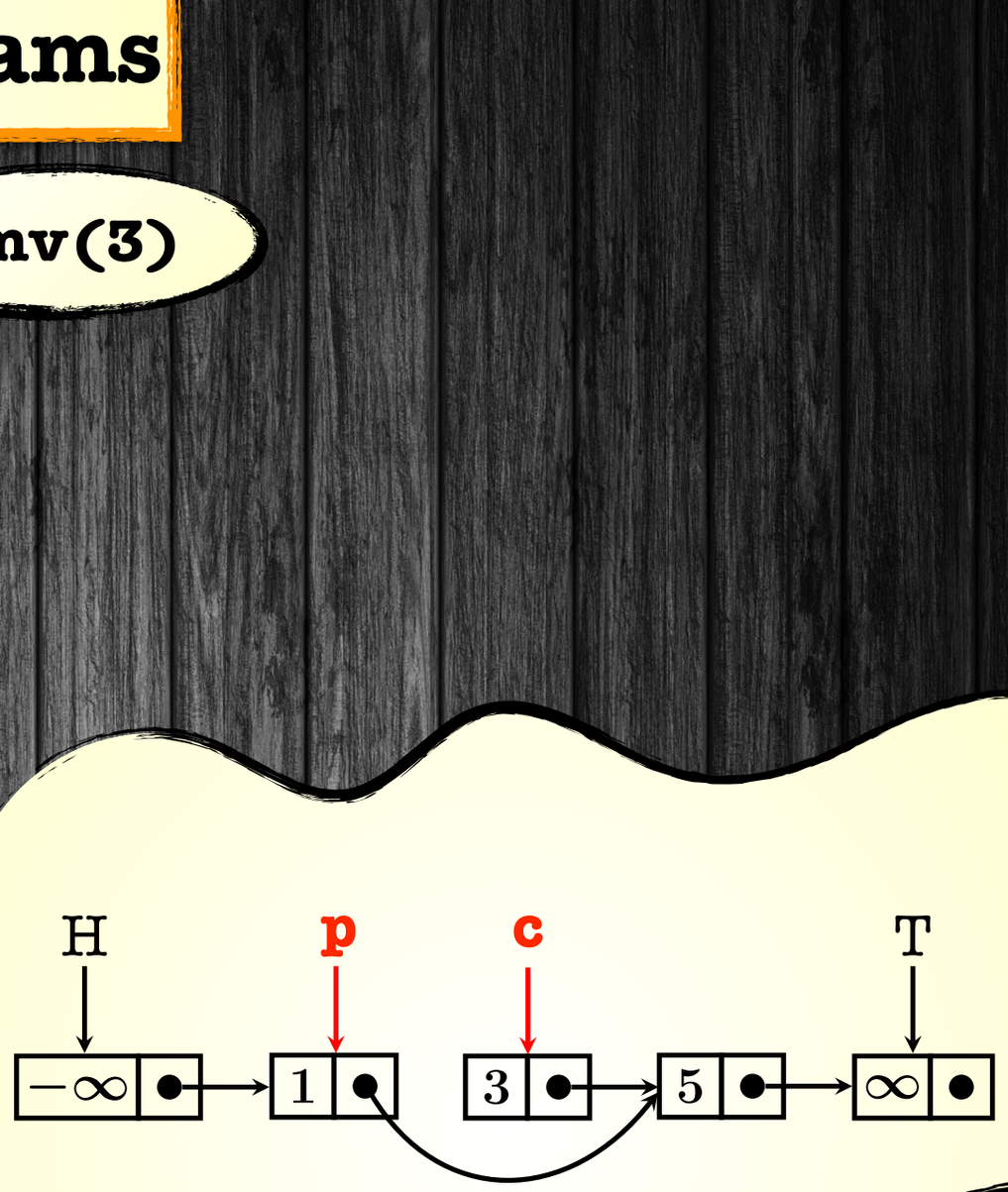


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

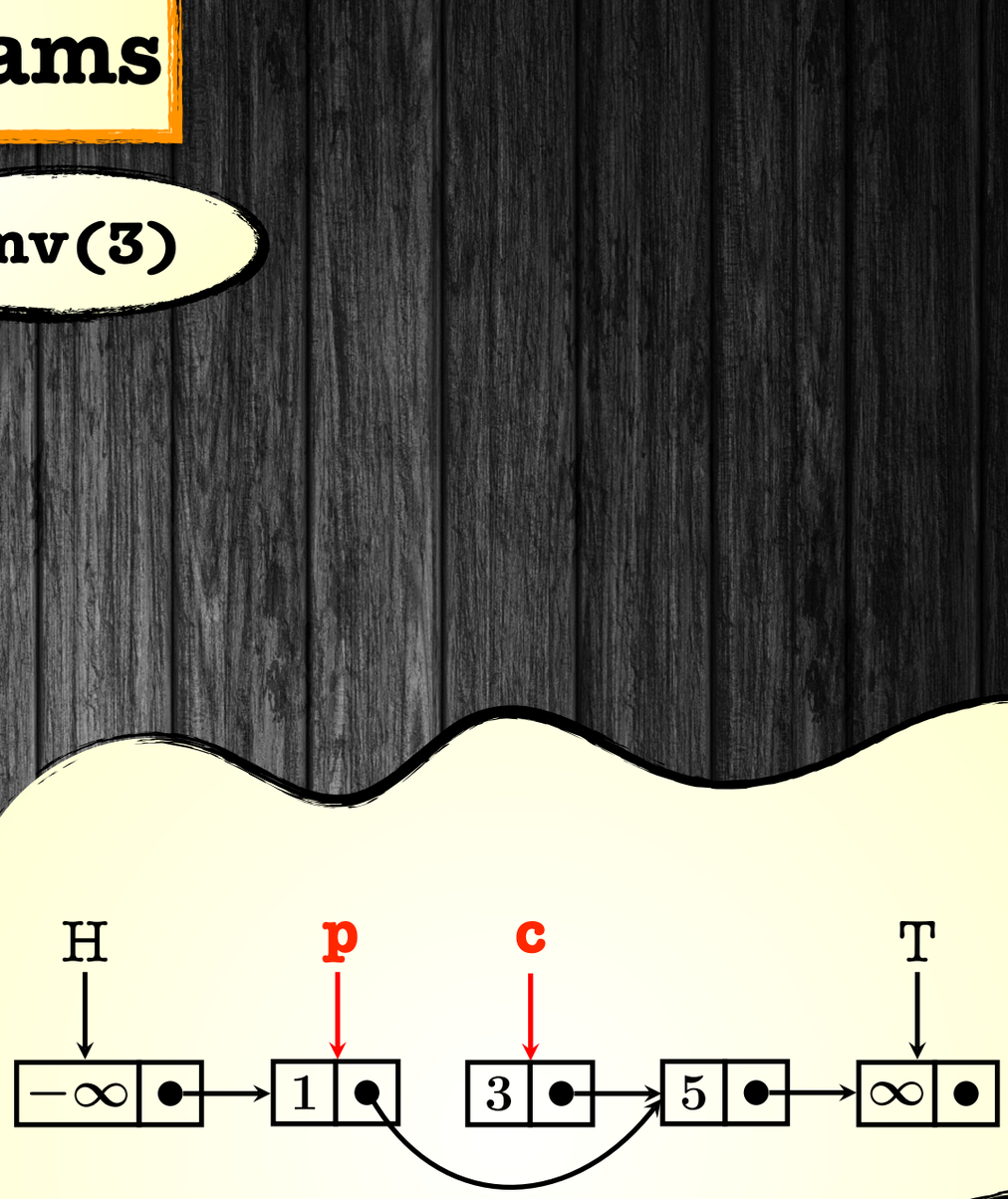


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(2)

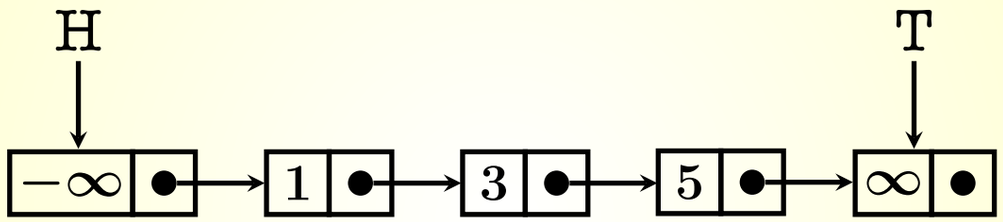


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(2)

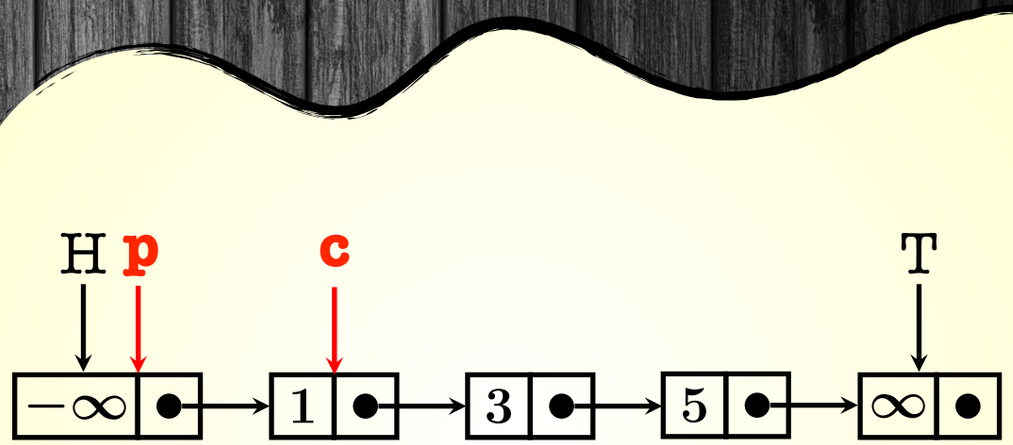


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(2)

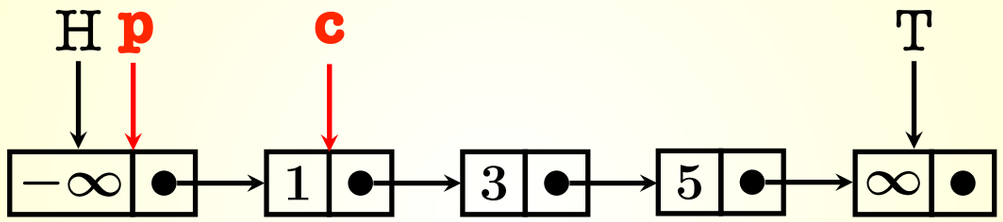


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(2)

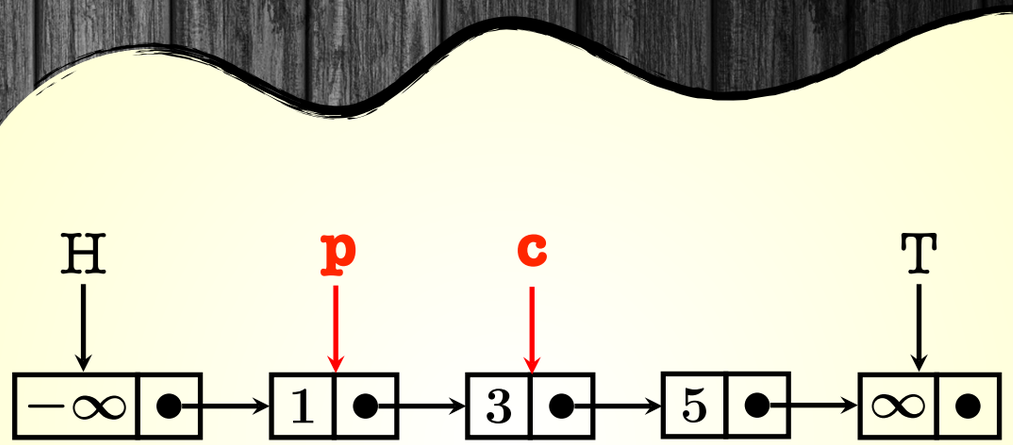


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(2)

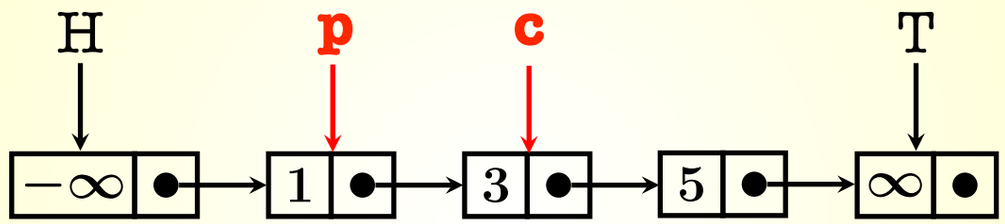


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(2)

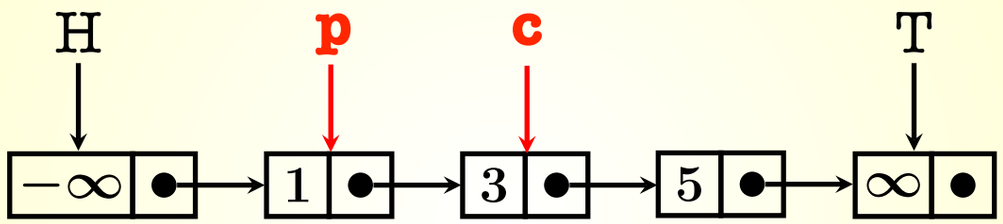


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(2)

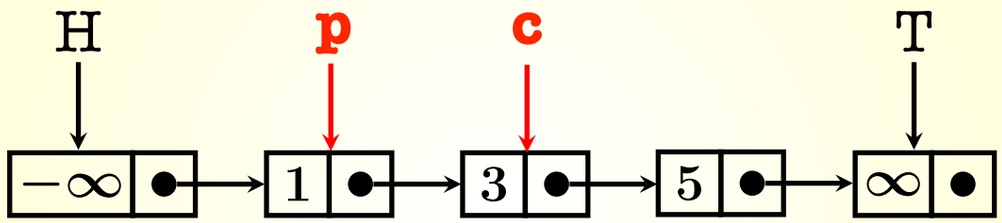


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

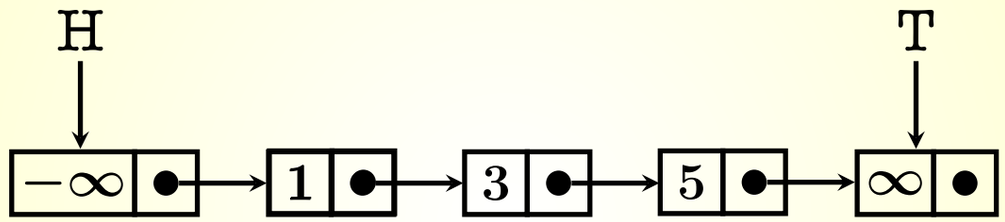
rmv(2)



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

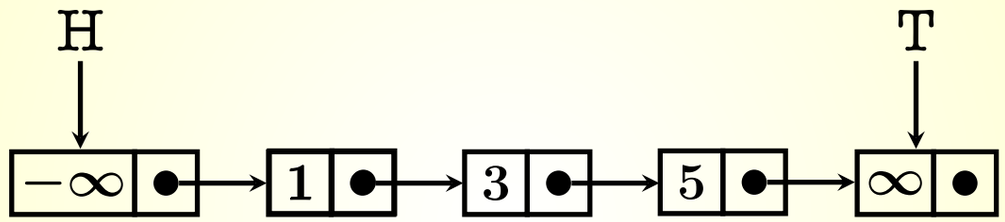


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)

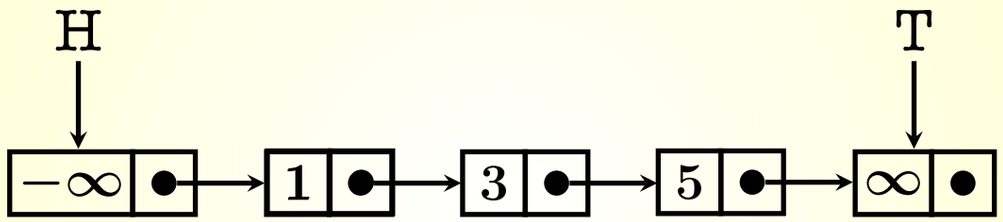


Sets | Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

locate

rmv(3)



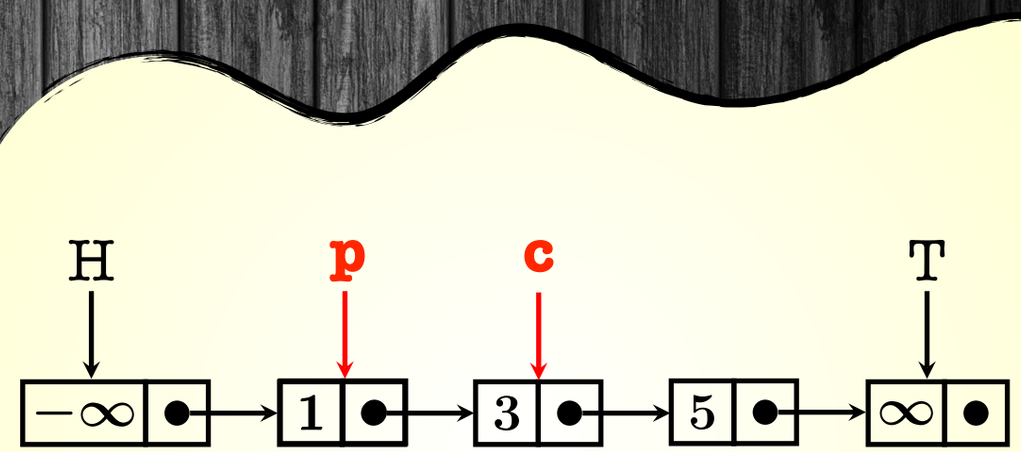
Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

locate

rmv(3)

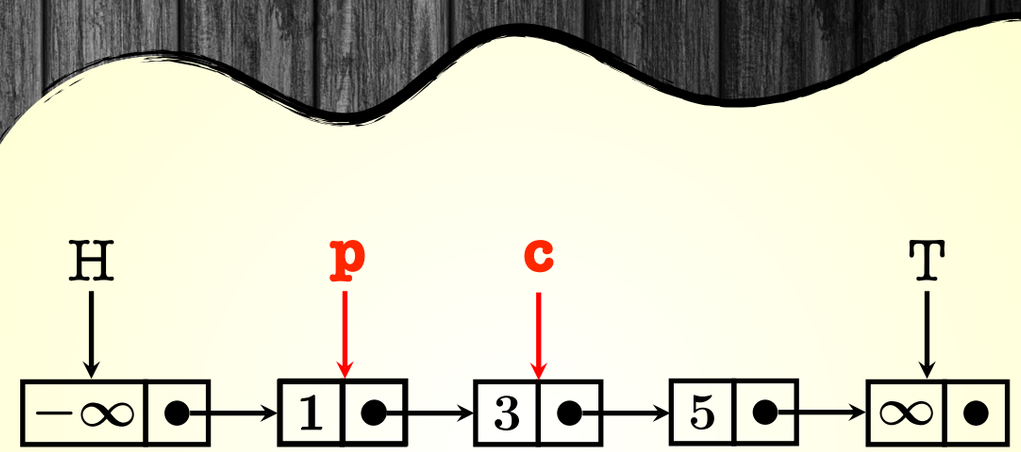


Sets | Sequential Programs

```
  rmv(k):
  Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   p.next = c.next
8   return true
9 else
10  return false
```

locate

rmv(3)

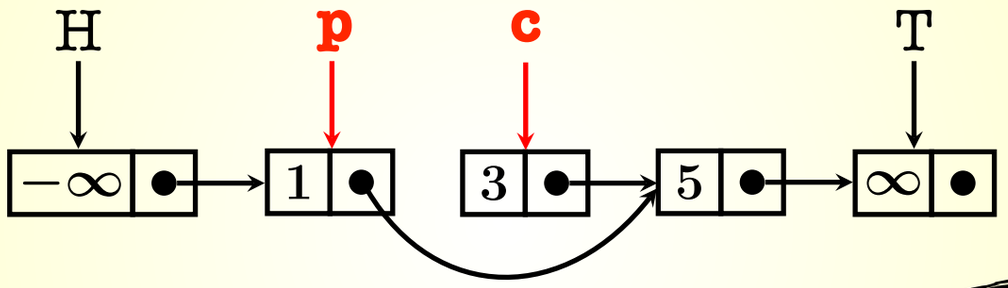


Sets | Sequential Programs

```
  rmv(k):
  Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   p.next = c.next
8   return true
9 else
10  return false
```

locate

rmv(3)



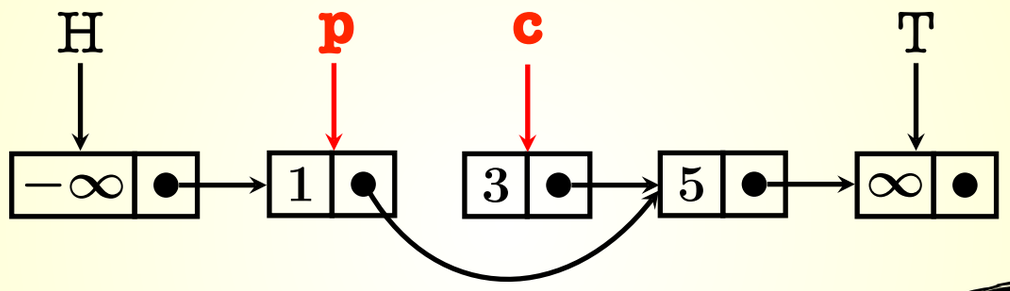
Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

locate

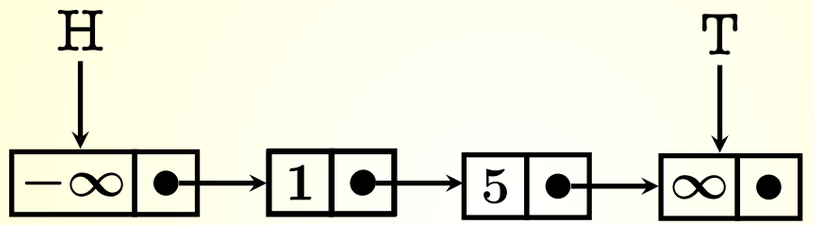
rmv(3)



Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

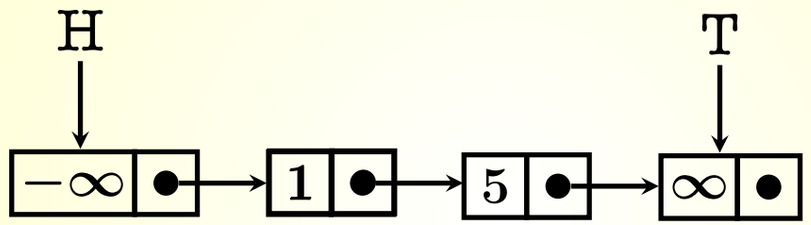


Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

rmv(3)



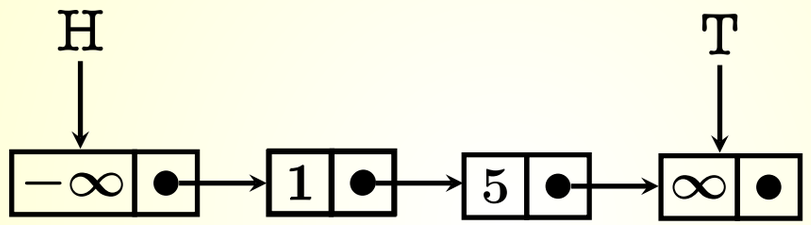
Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

locate

rmv(3)



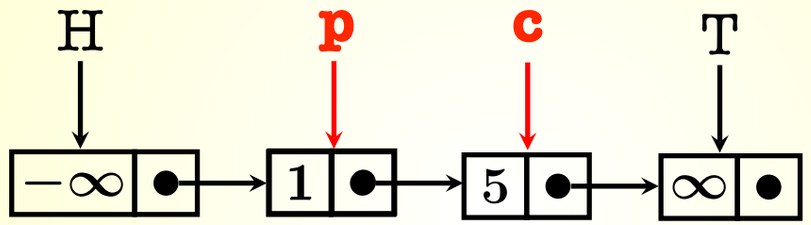
Sets

Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

locate

rmv(3)

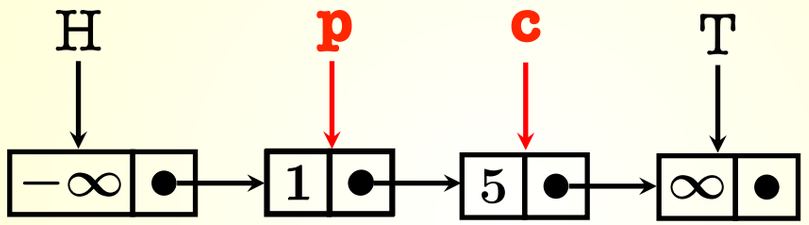


Sets | Sequential Programs

```
  rmv(k):
  Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   p.next = c.next
8   return true
9 else
10  return false
```

locate

rmv(3)

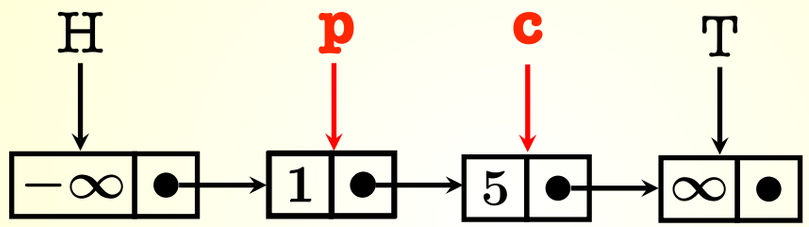


Sets | Sequential Programs

```
rmv(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    p.next = c.next
8    return true
9  else
10   return false
```

locate

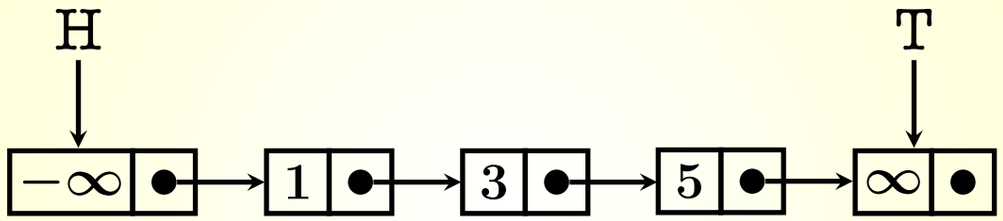
rmv(3)



Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

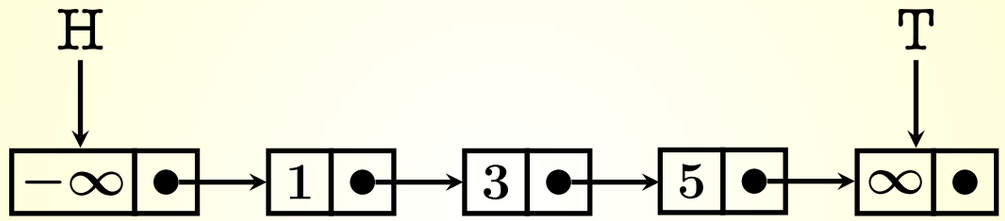


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(3)

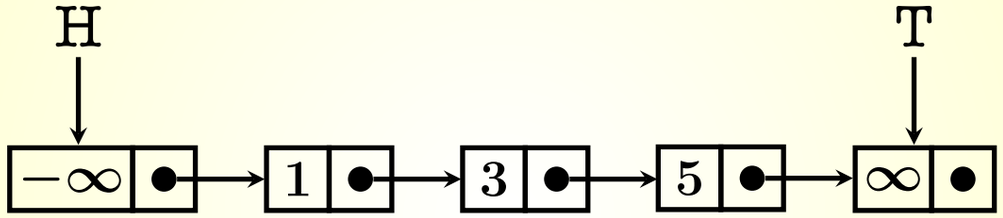


Sets

Sequential Programs

```
ctn(k):
Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   return true
8 else
9   return false
```

ctn(3)

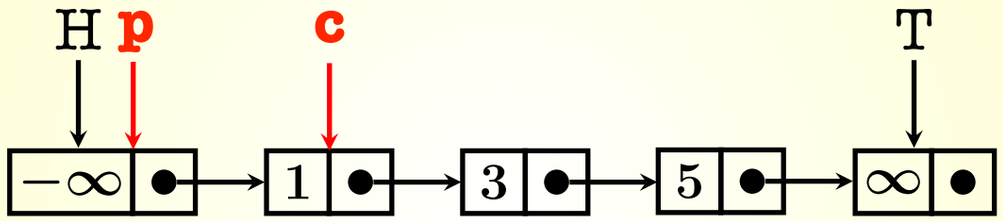


Sets

Sequential Programs

```
ctn(k):
Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   return true
8 else
9   return false
```

ctn(3)

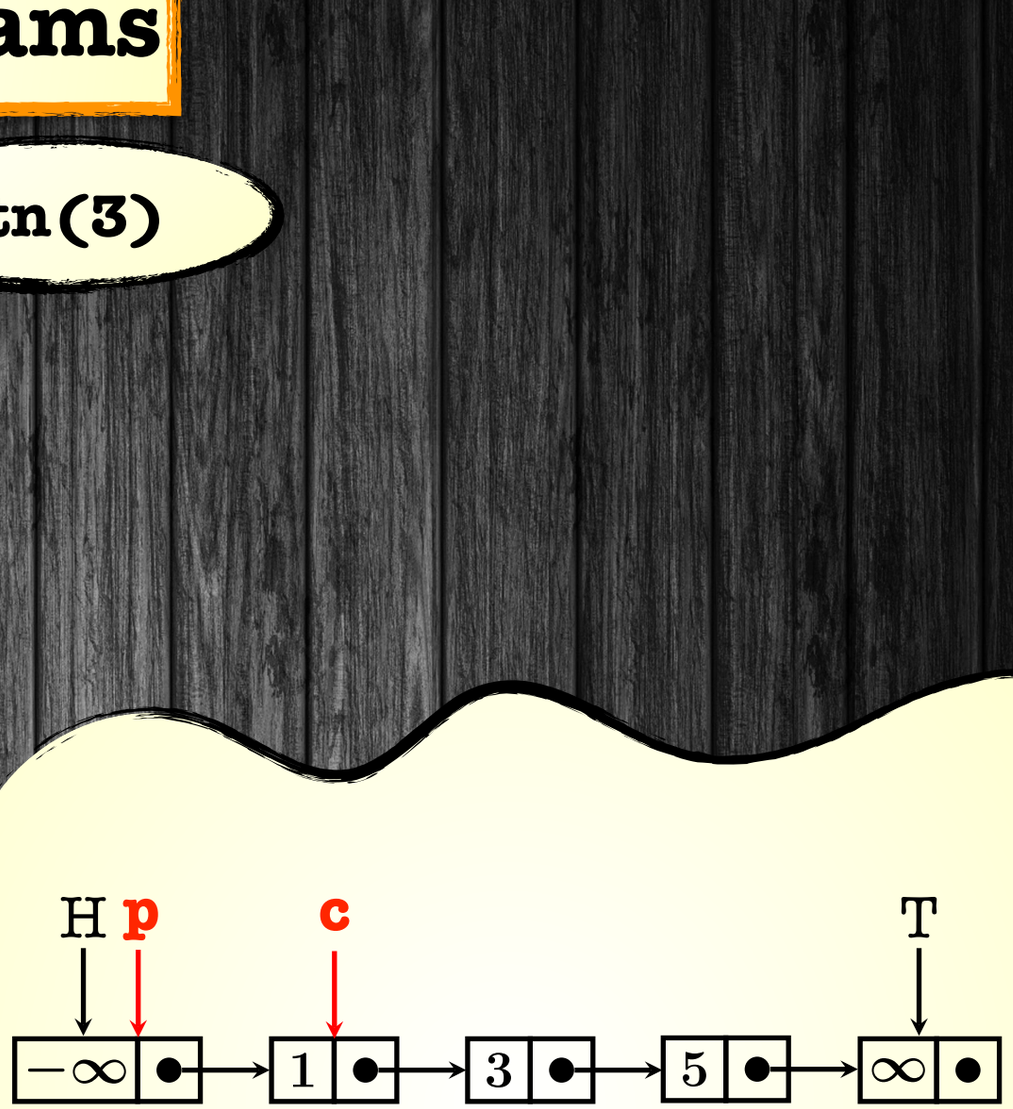


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(3)

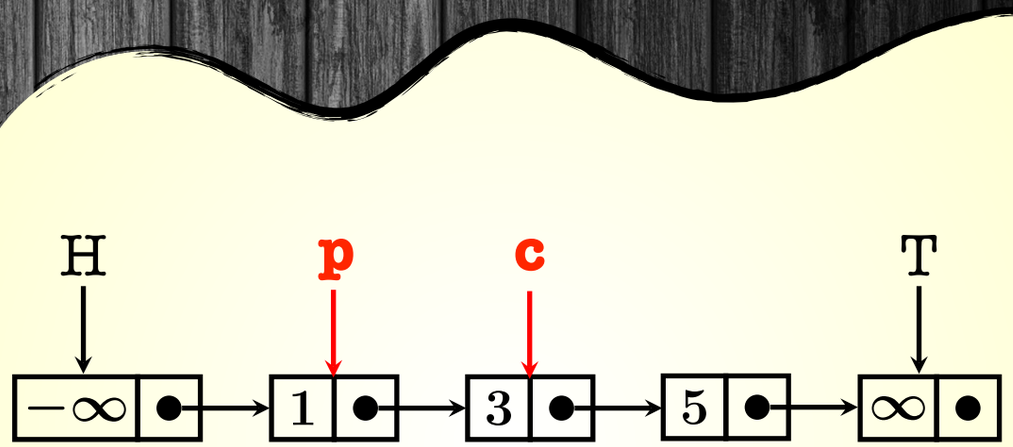


Sets

Sequential Programs

```
ctn(k):
Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   return true
8 else
9   return false
```

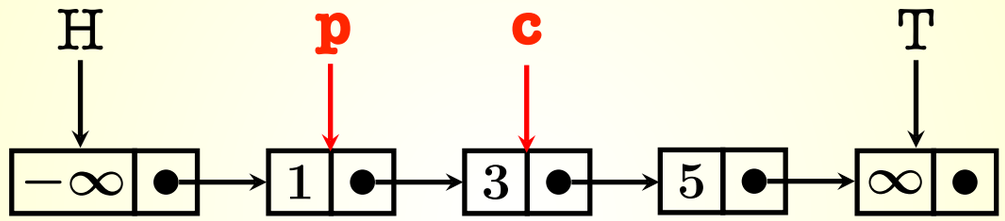
ctn(3)



Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(3)

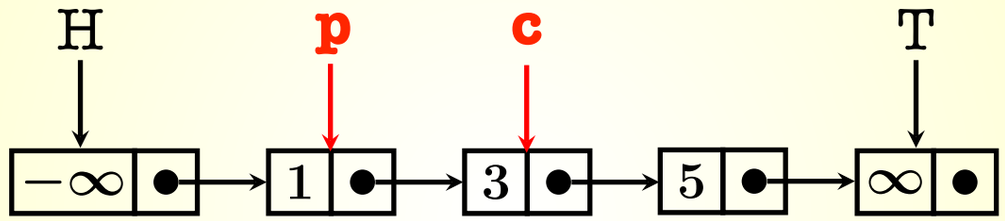


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

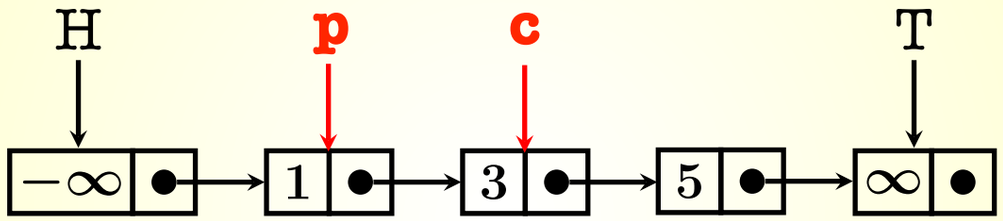
ctn(3)



Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

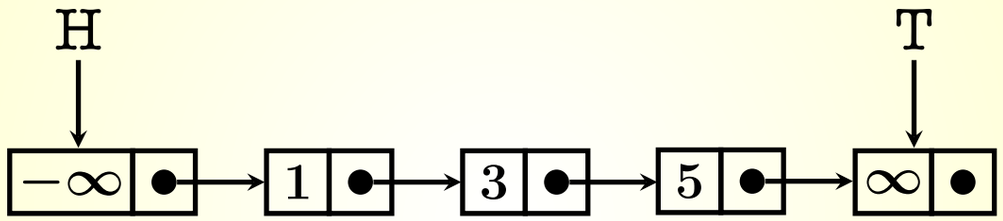
ctn(3)



Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

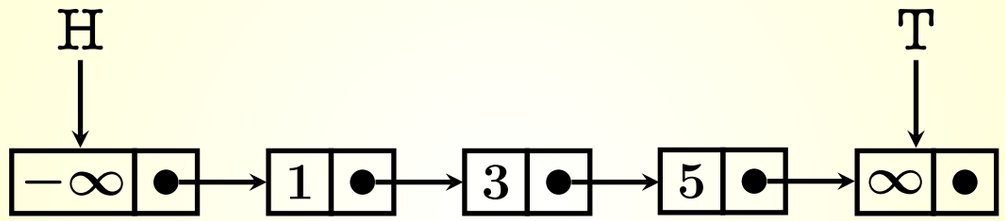


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(2)

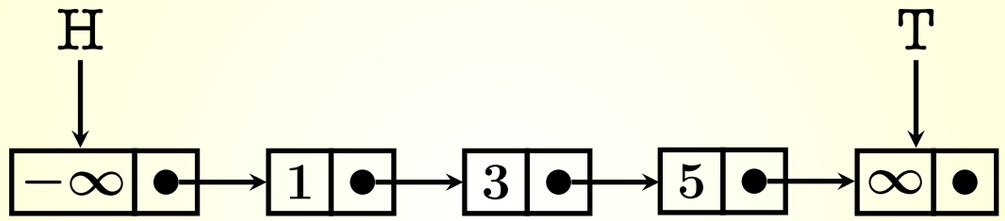


Sets

Sequential Programs

```
ctn(k):
Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   return true
8 else
9   return false
```

ctn(2)

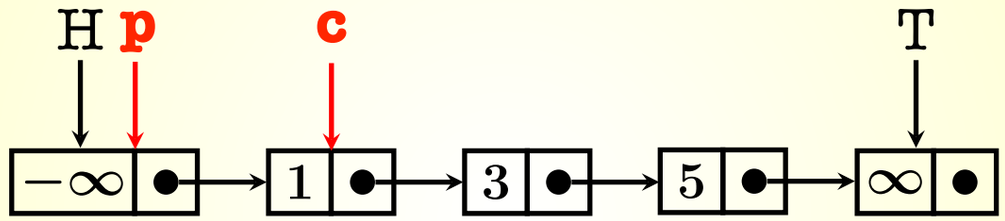


Sets

Sequential Programs

```
ctn(k):
Node p, c
1 p = H;
2 c = p.next
3 while (c.key < k)
4   p = c;
5   c = c.next;
6 if (c.key = k)
7   return true
8 else
9   return false
```

ctn(2)

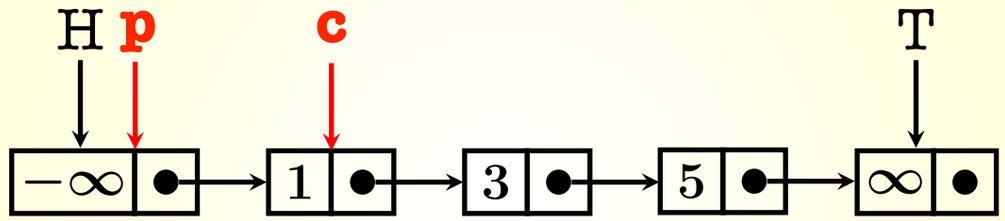


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(2)

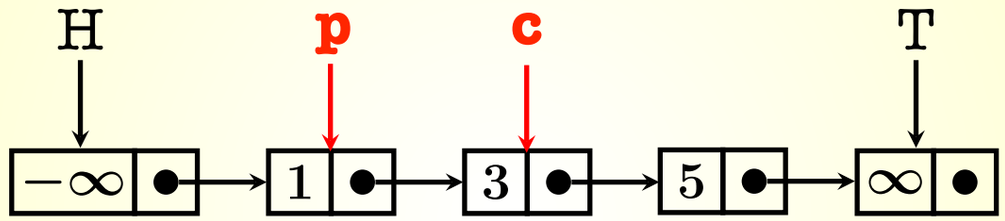


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

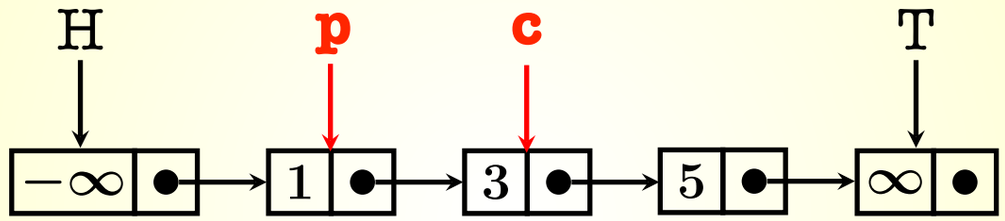
ctn(2)



Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(2)

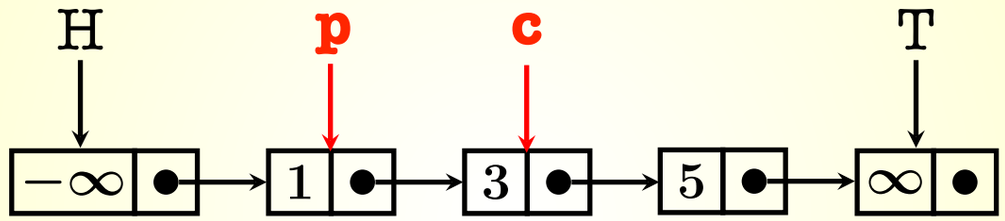


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

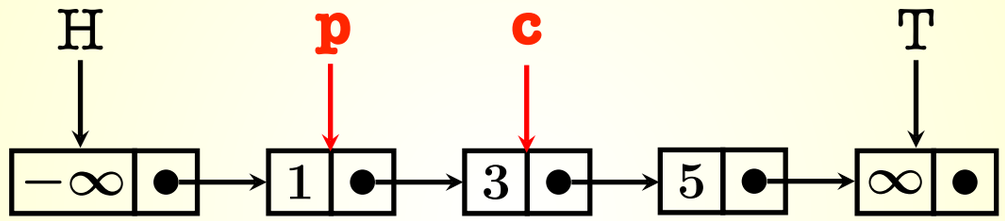
ctn(2)



Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

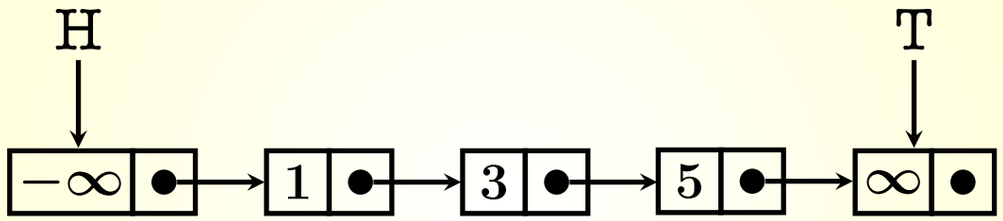
ctn(2)



Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

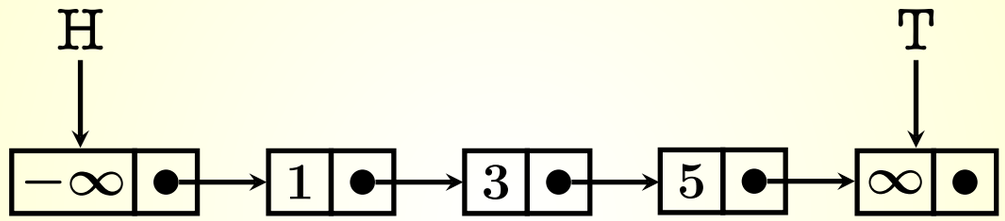


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(3)



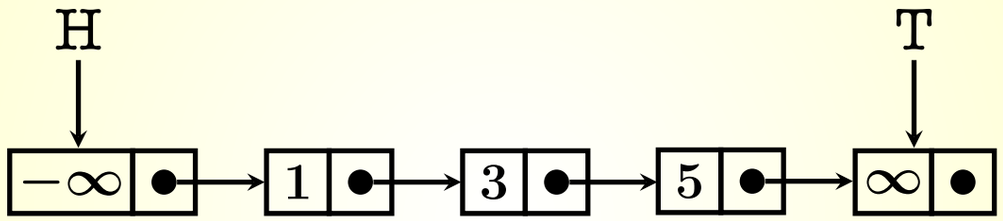
Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

ctn(3)

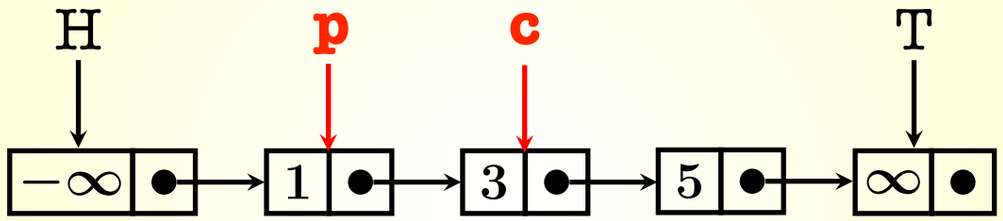


Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

ctn(3)

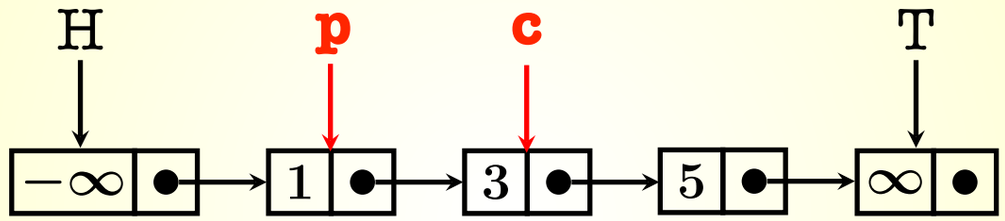


Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

ctn(3)

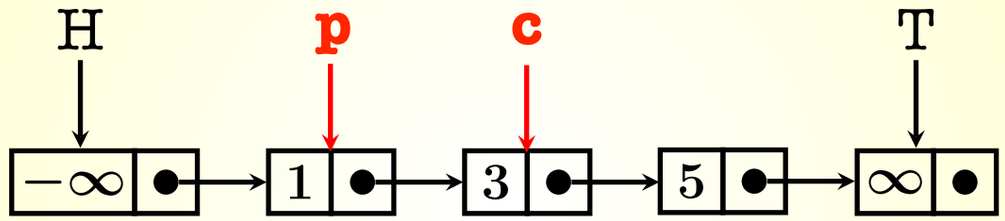


Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

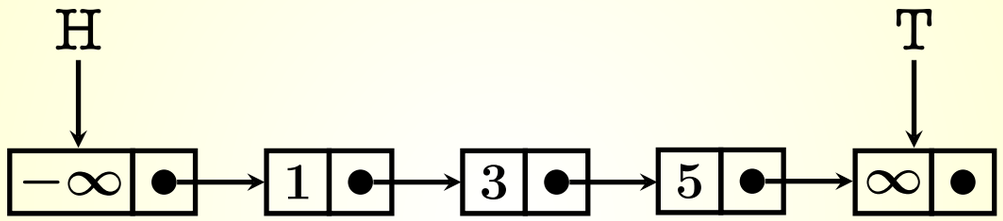
ctn(3)



Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

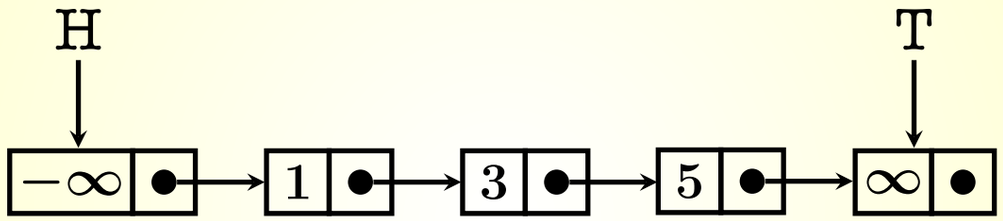


Sets

Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

ctn(2)

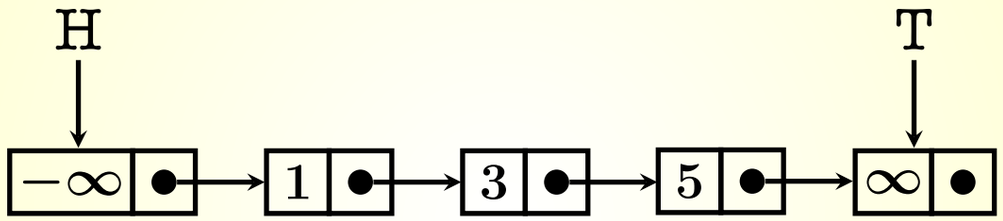


Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

ctn(2)

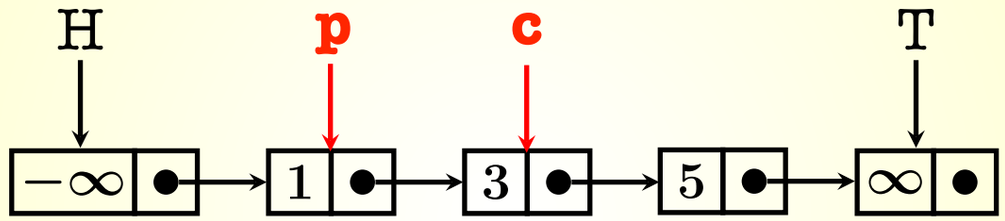


Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

ctn(2)

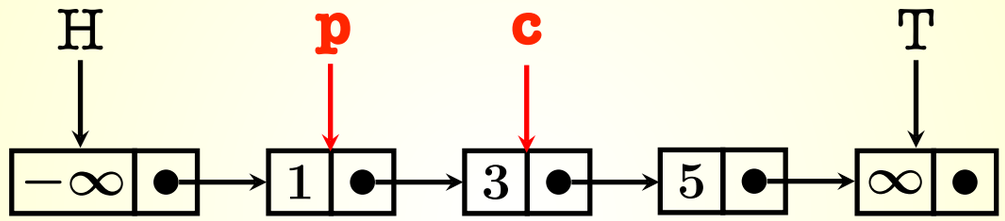


Sets | Sequential Programs

```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

ctn(2)

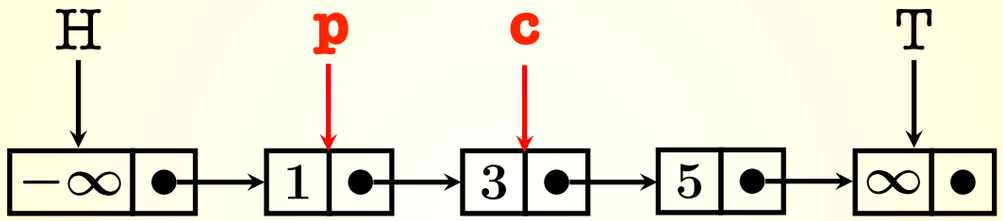


Sets | Sequential Programs

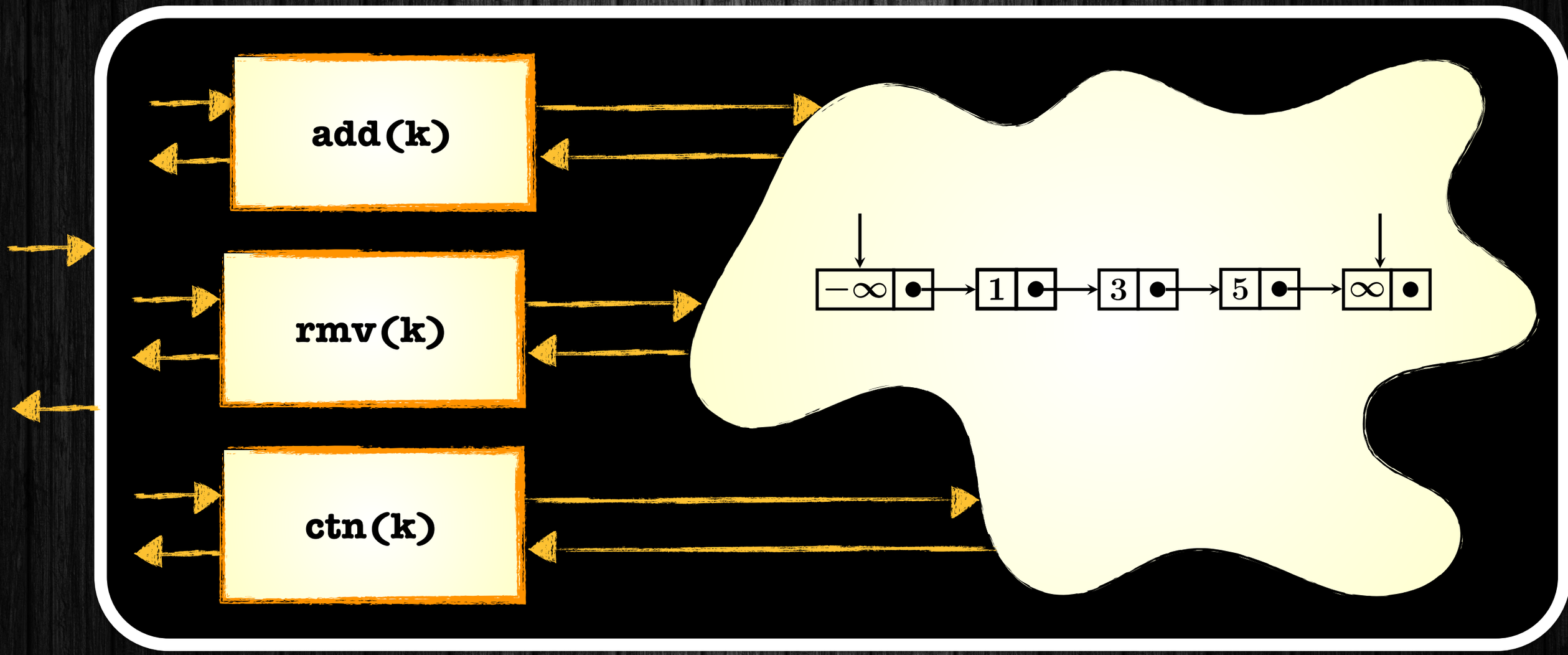
```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

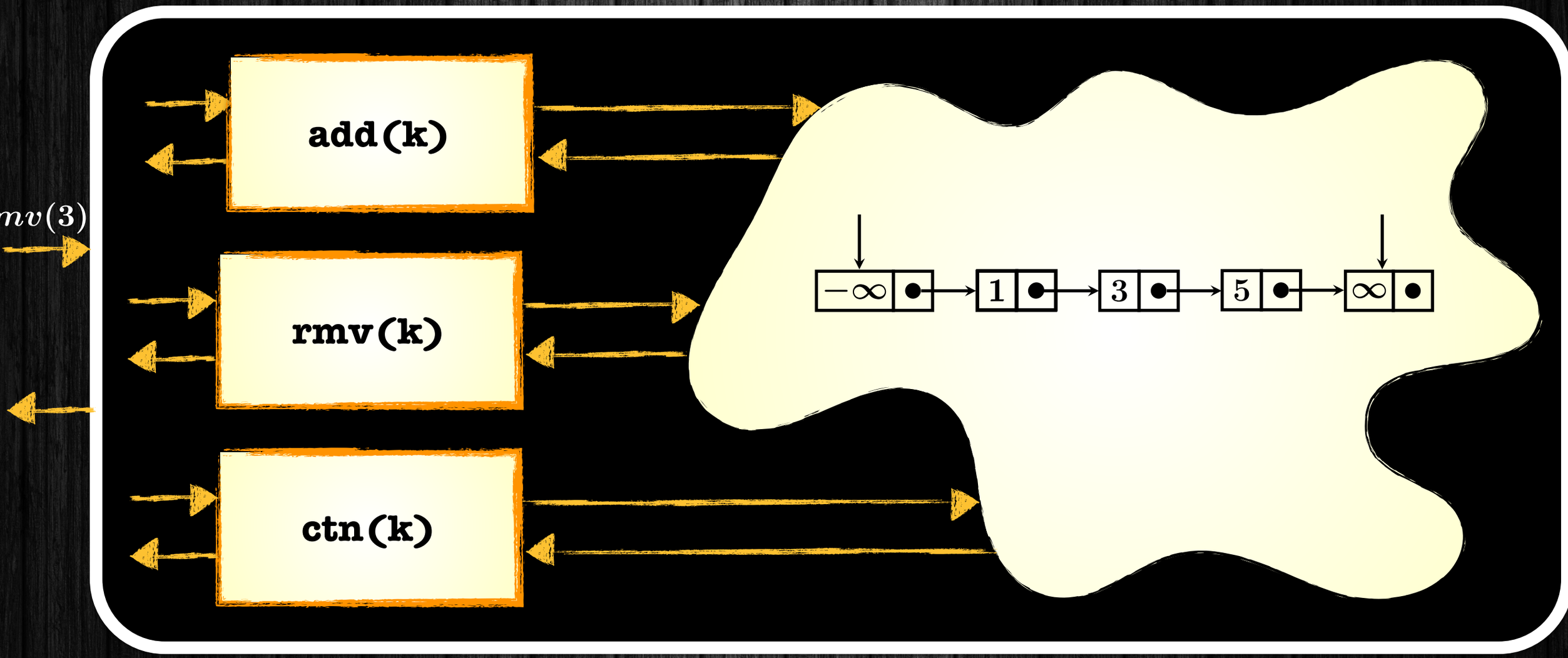
ctn(2)



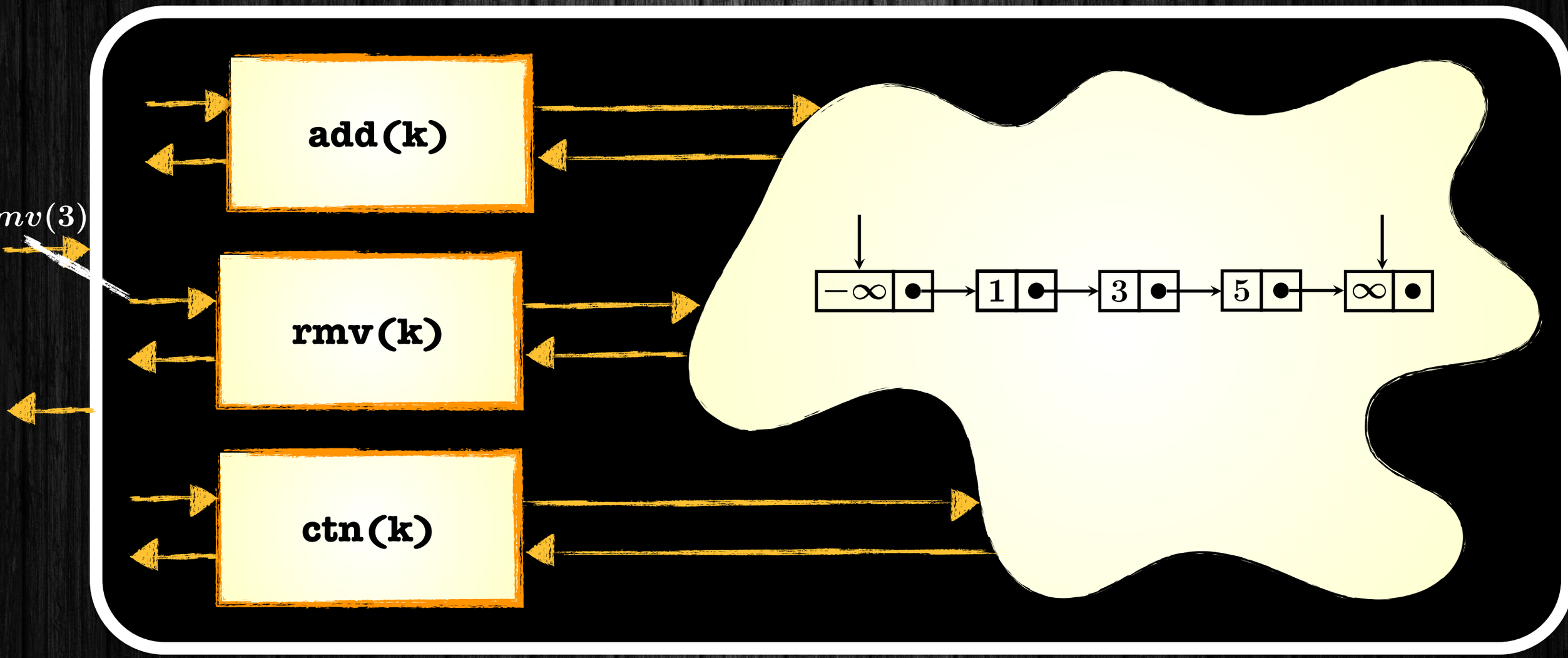
Sets | Sequential Programs



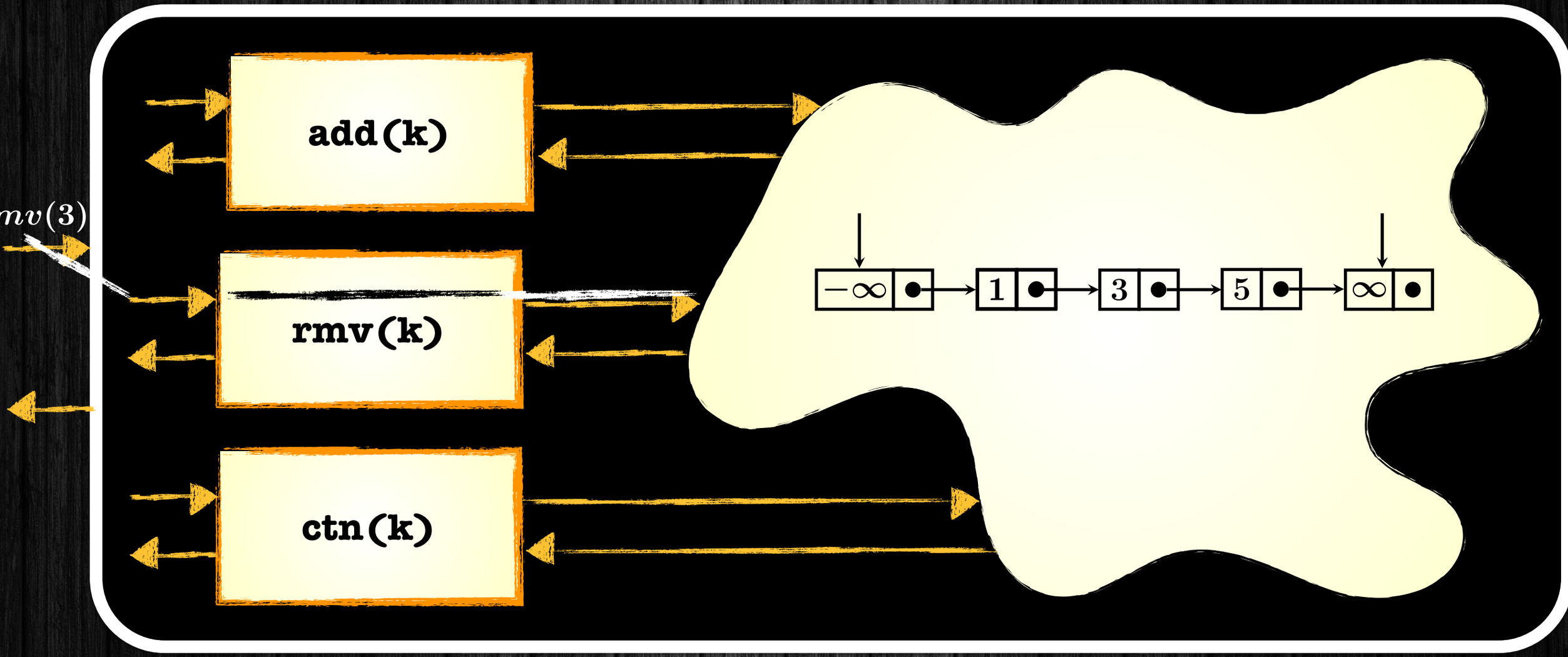
Sets | Sequential Programs



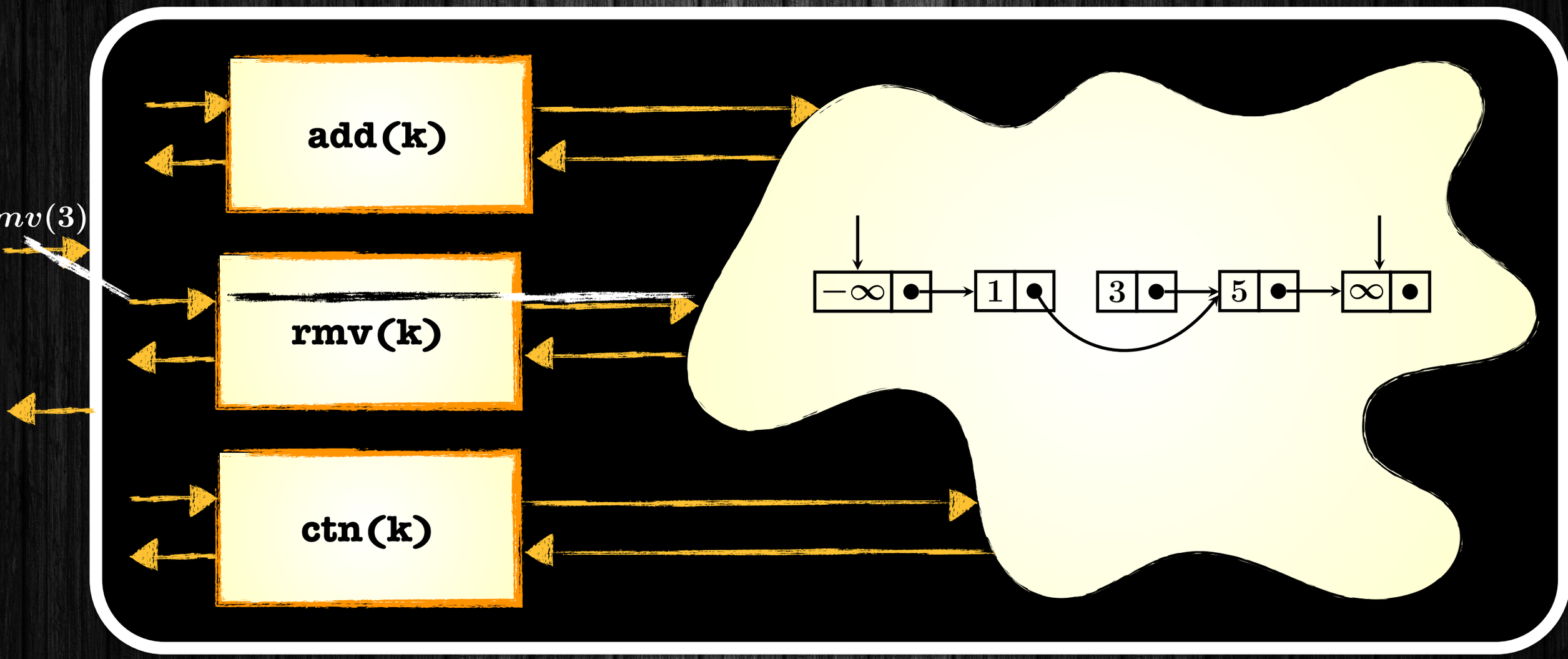
Sets | Sequential Programs



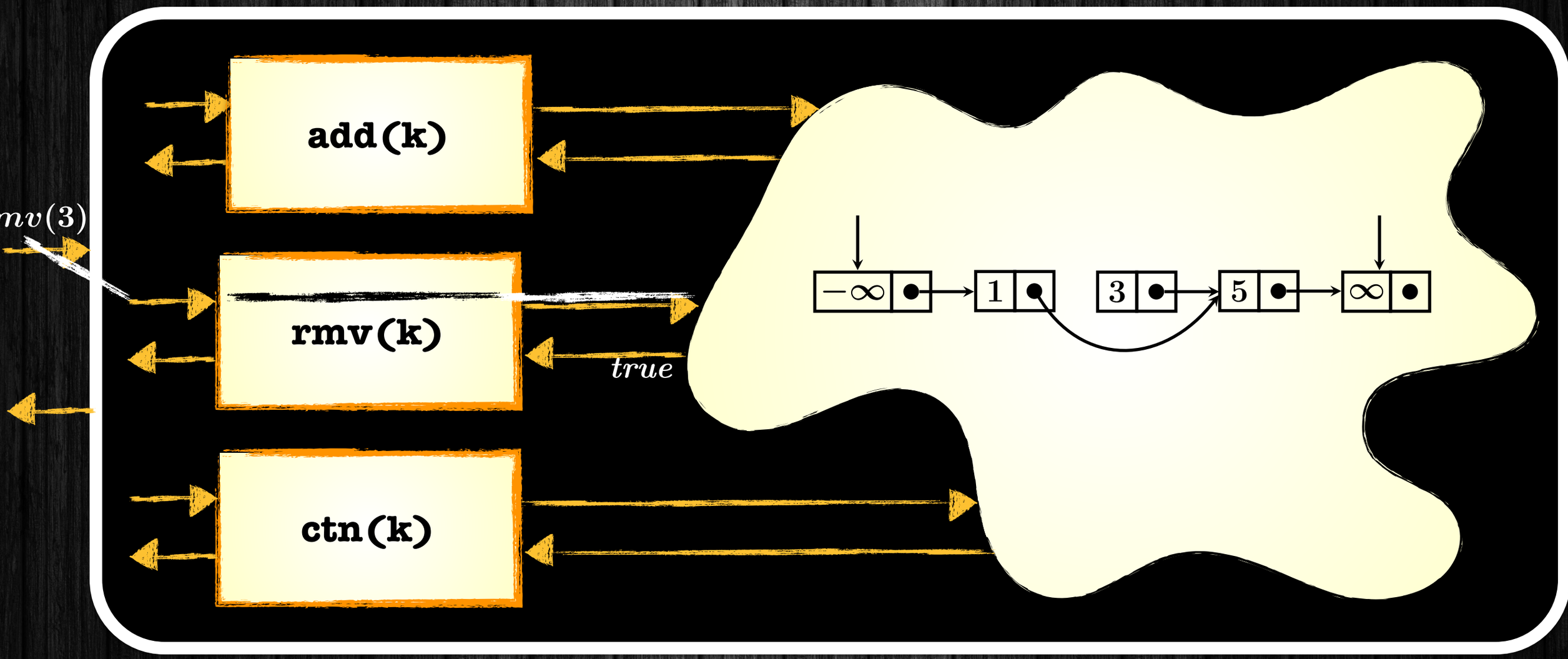
Sets | Sequential Programs



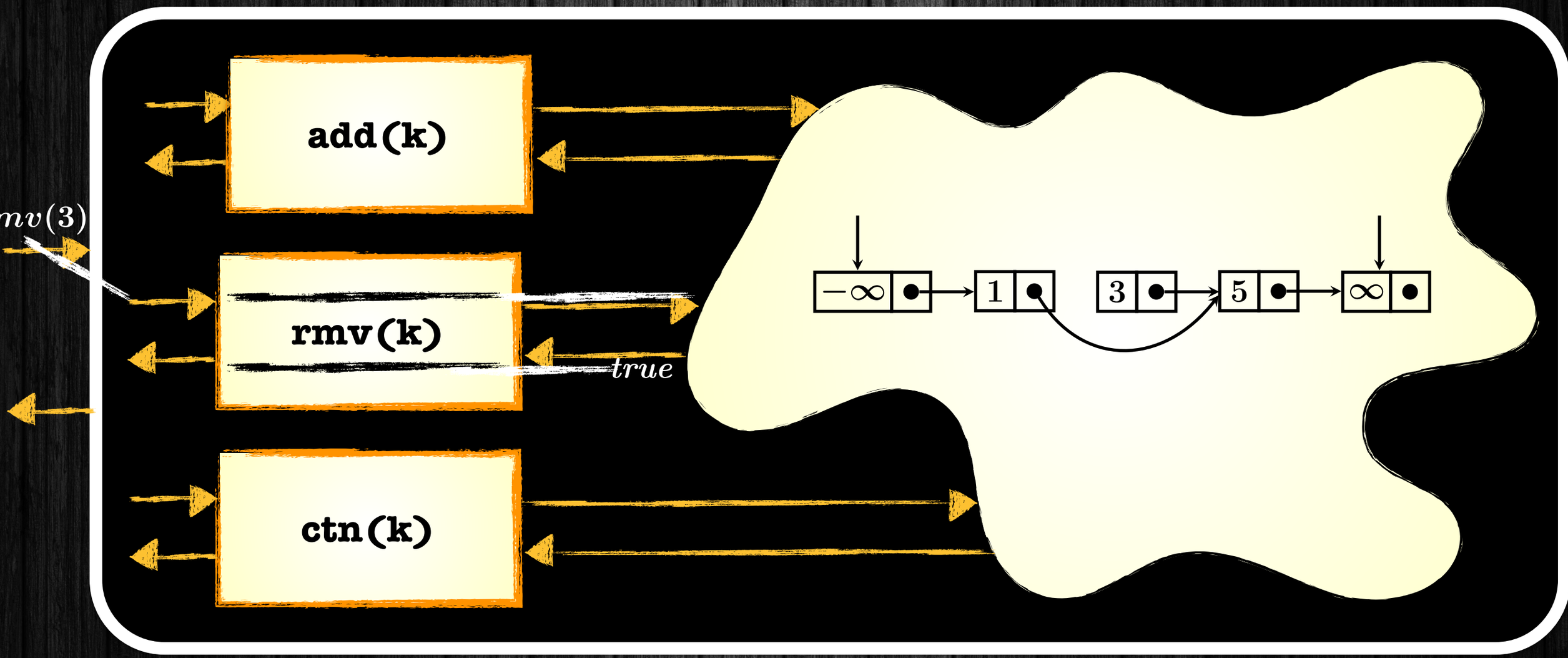
Sets | Sequential Programs



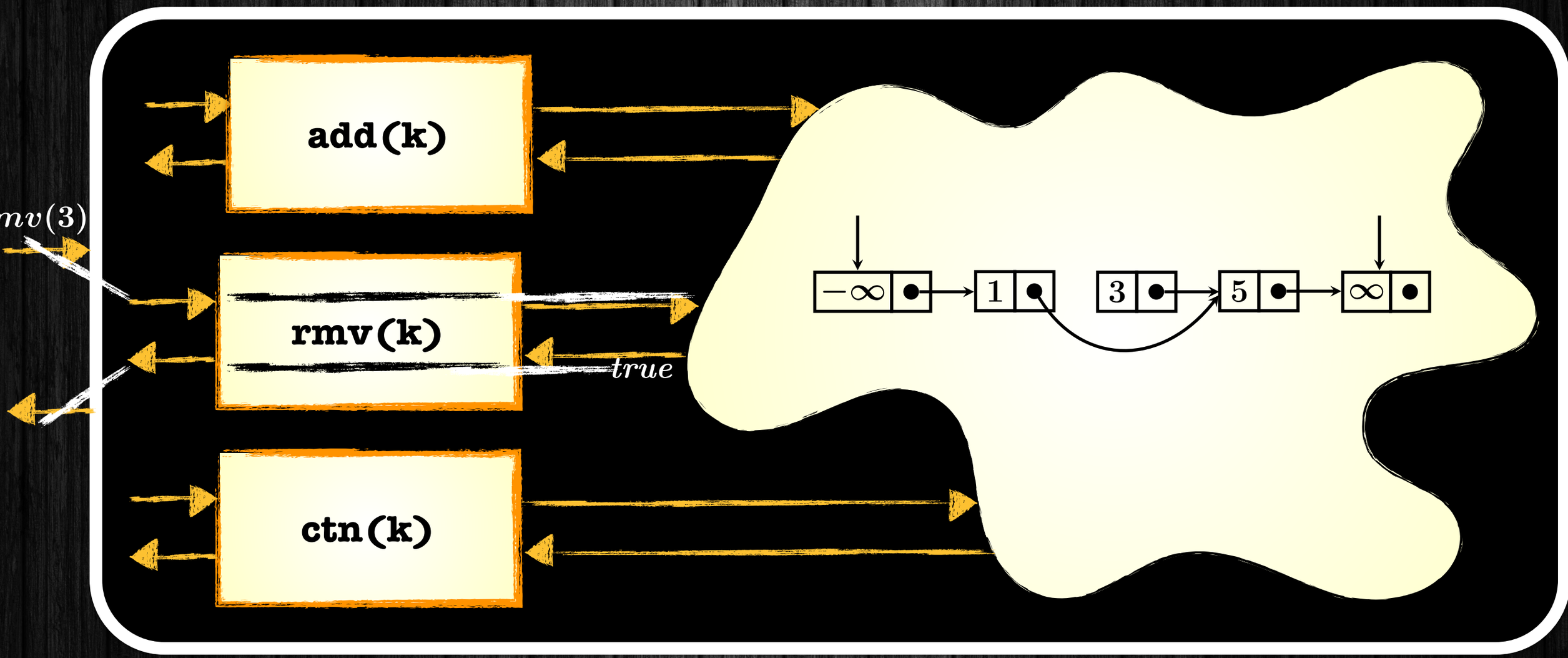
Sets | Sequential Programs



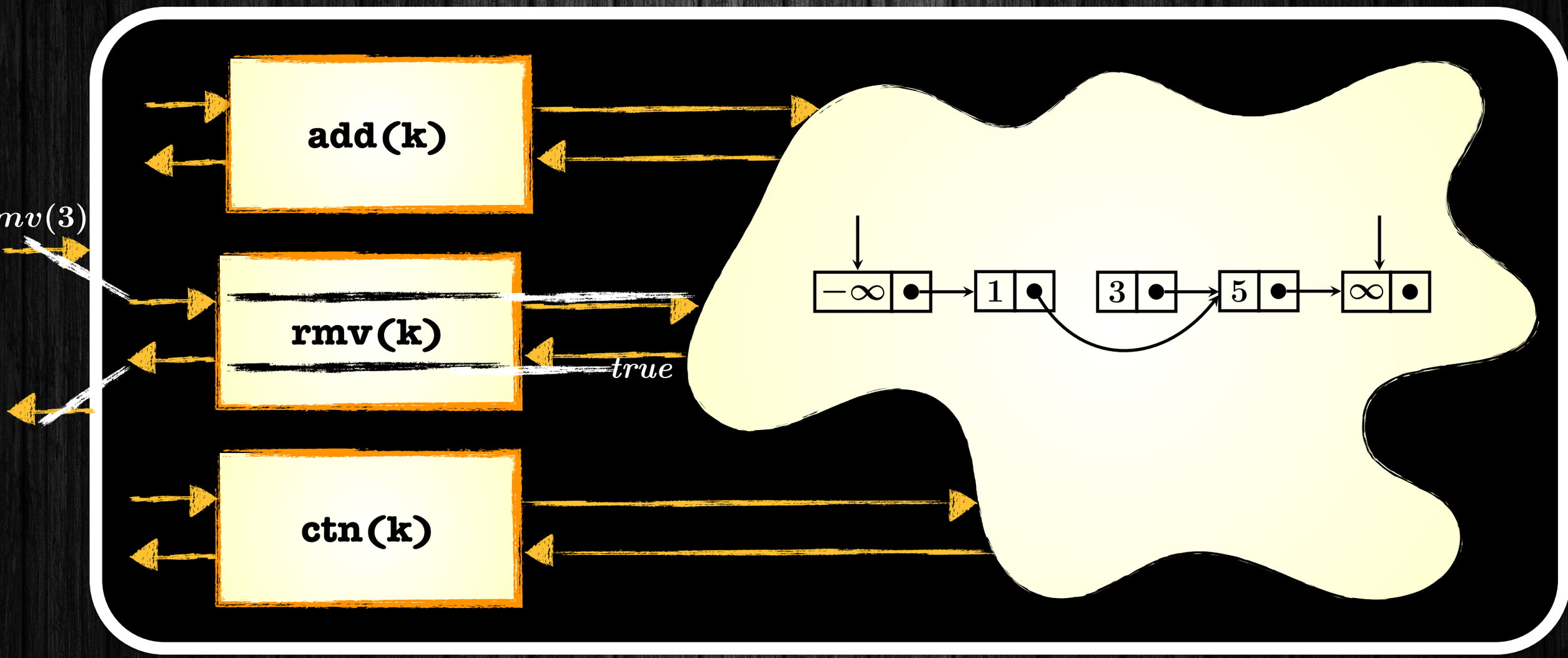
Sets | Sequential Programs



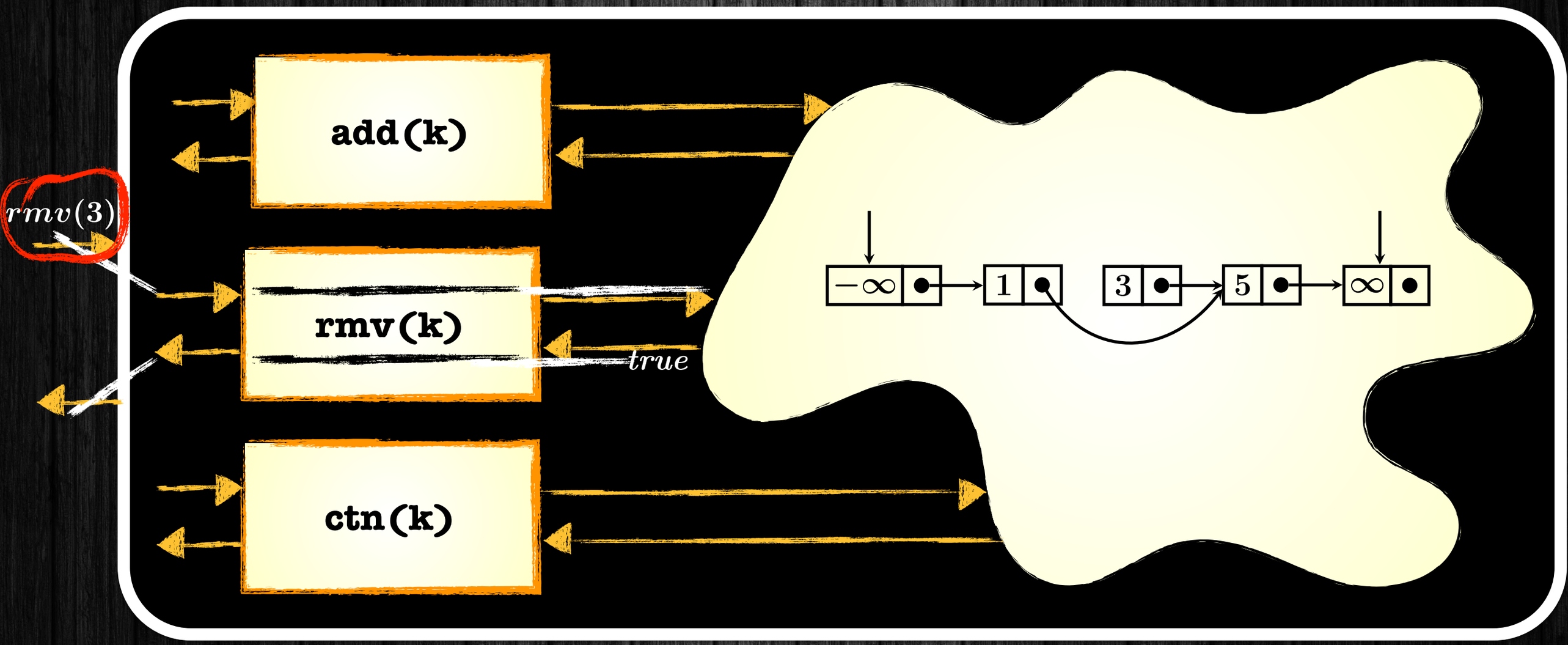
Sets | Sequential Programs



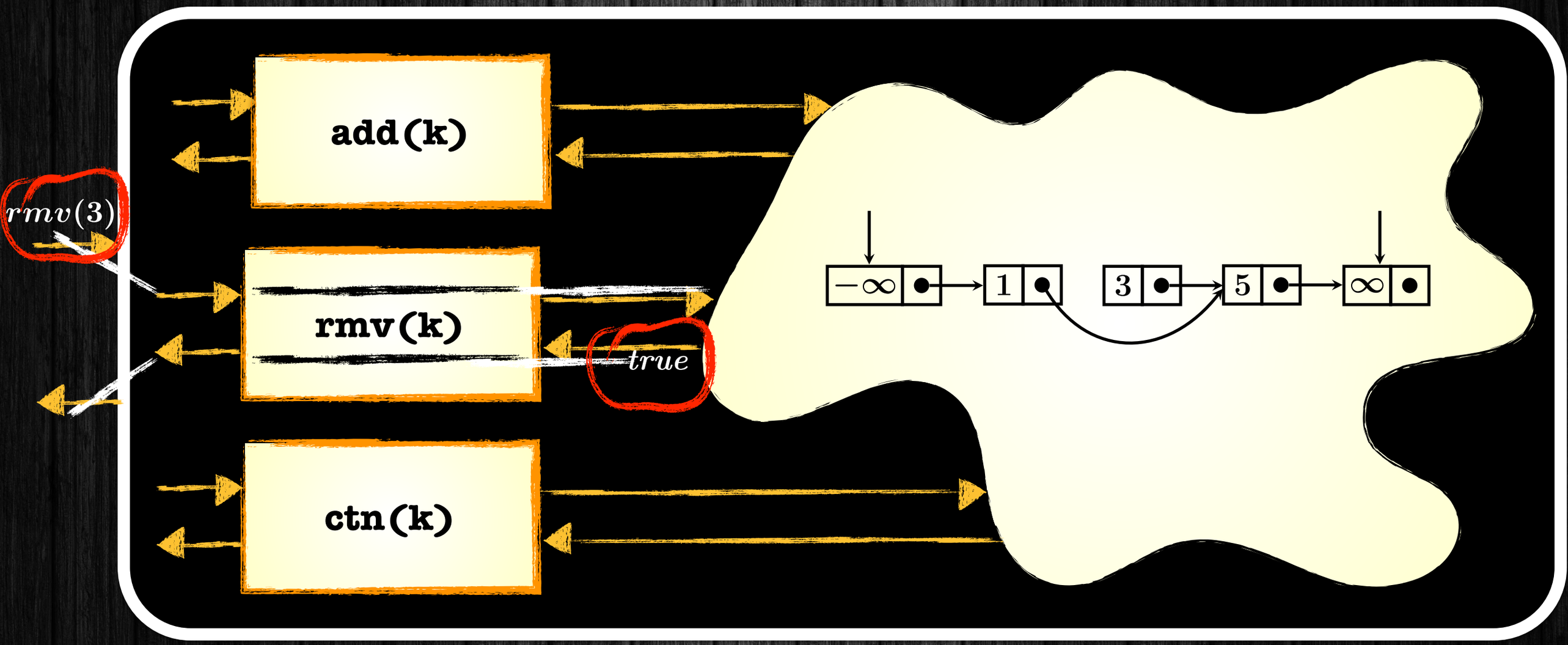
Sets | Sequential Programs



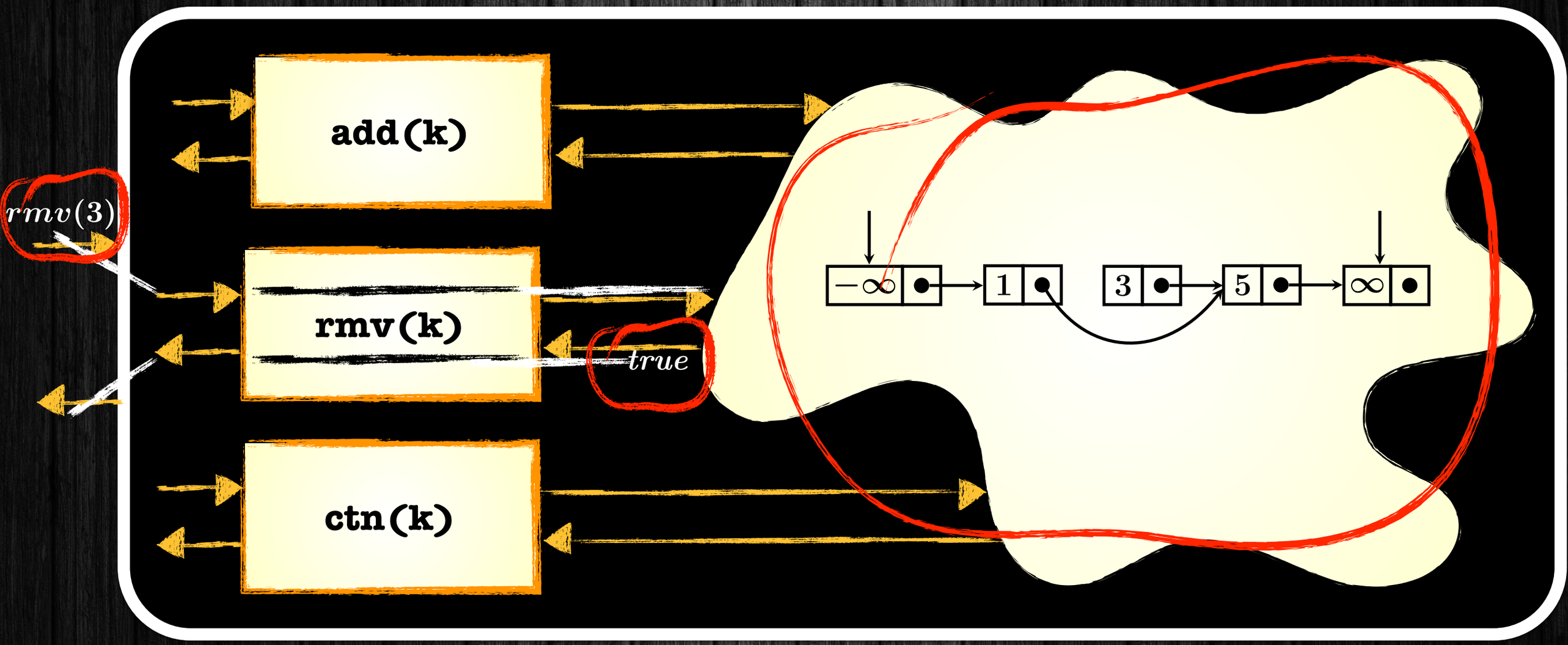
Sets | Sequential Programs



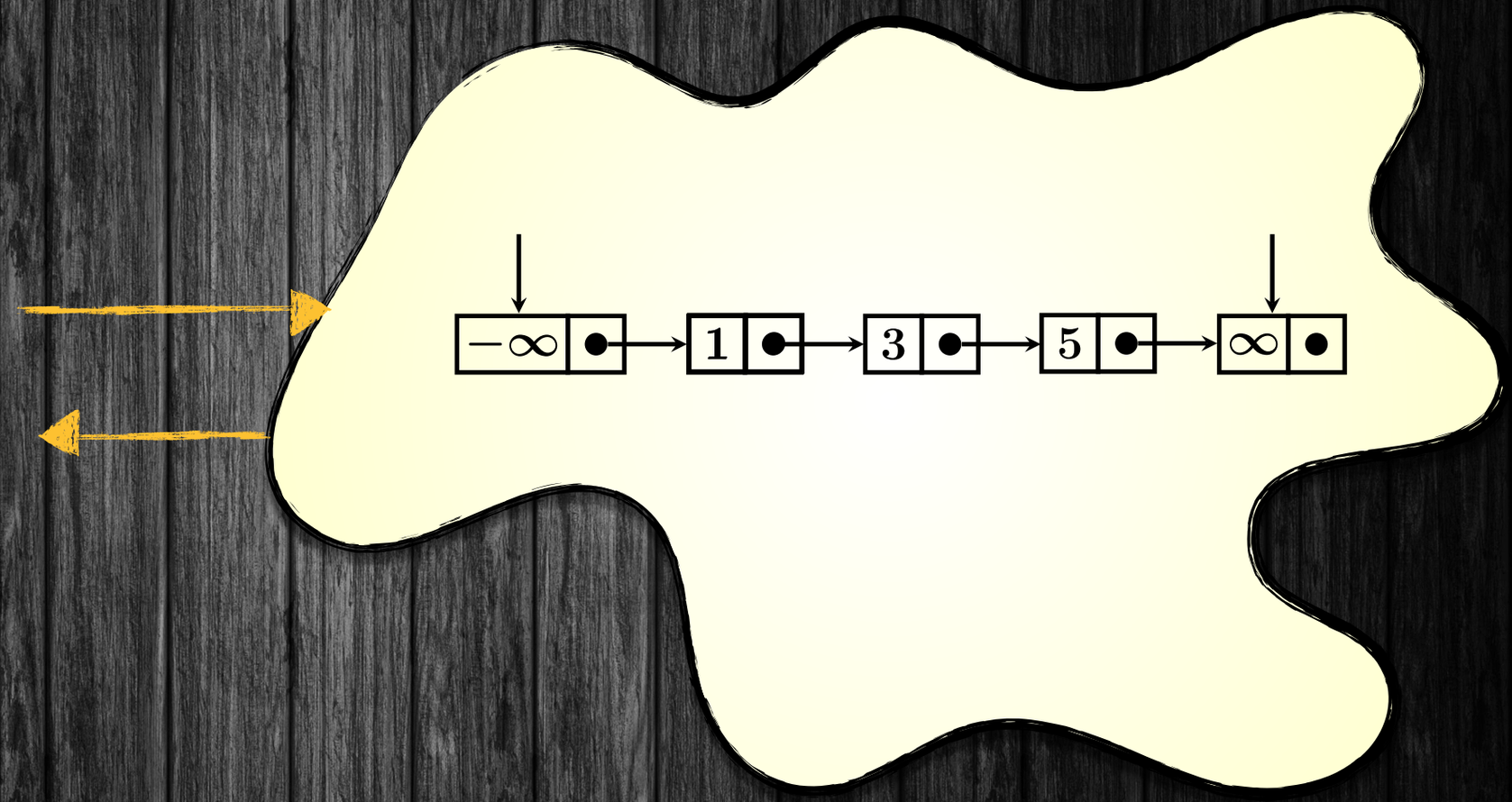
Sets | Sequential Programs



Sets | Sequential Programs

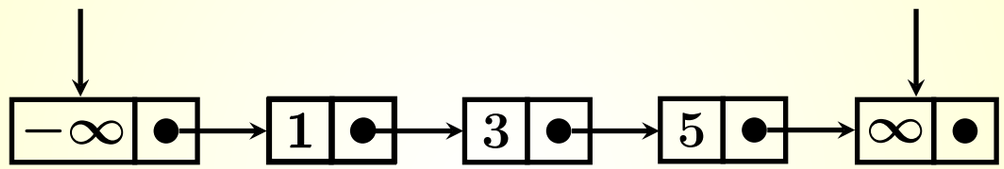


Sets | Sequential Programs

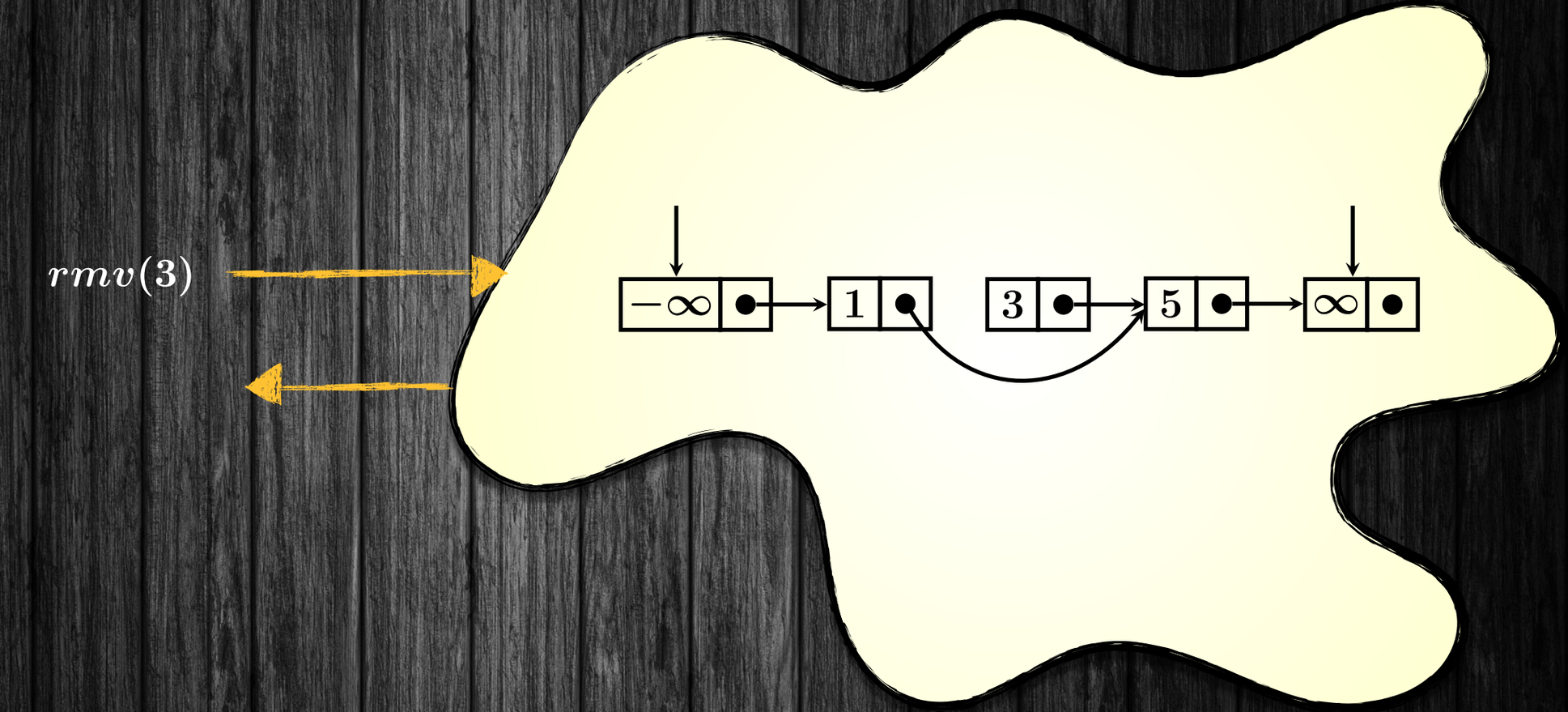


Sets | Sequential Programs

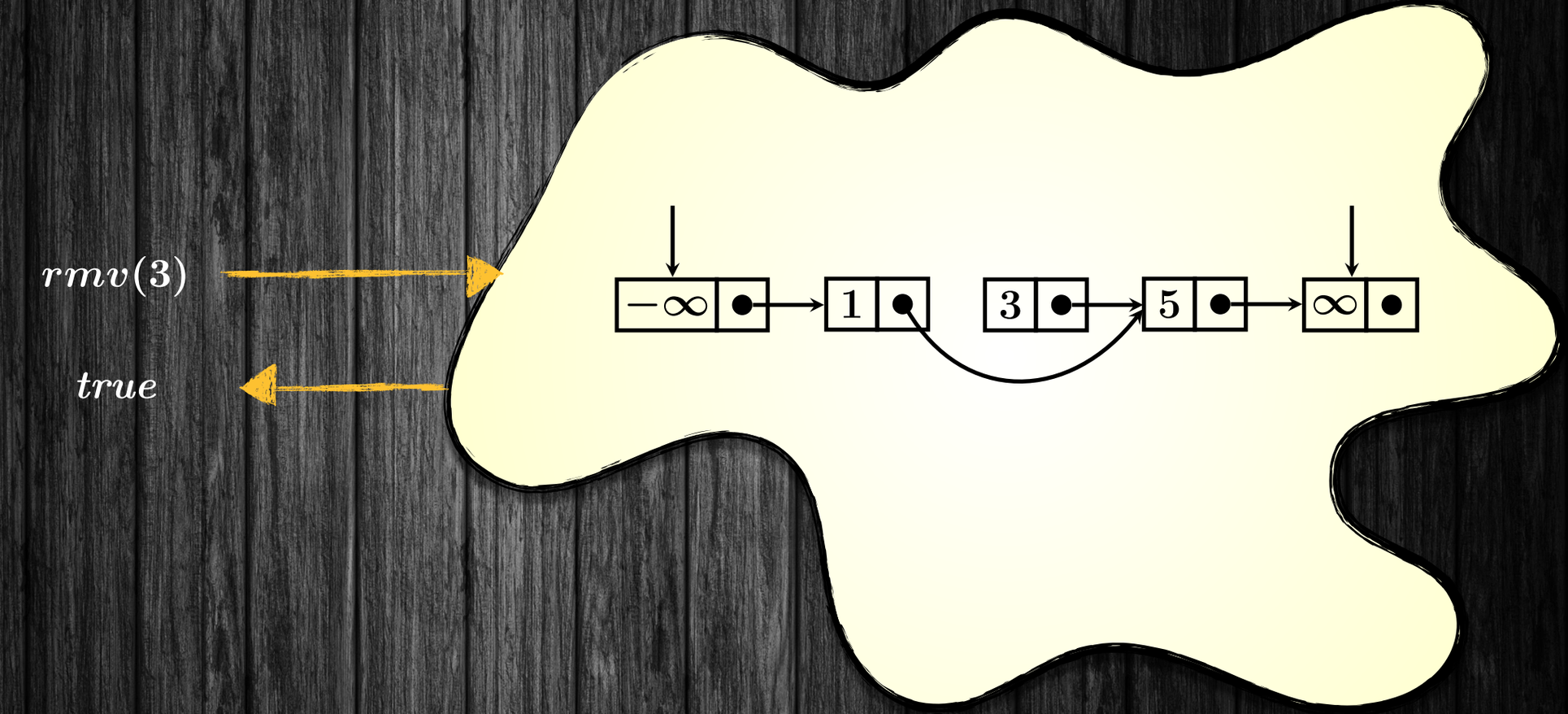
$rmv(3)$



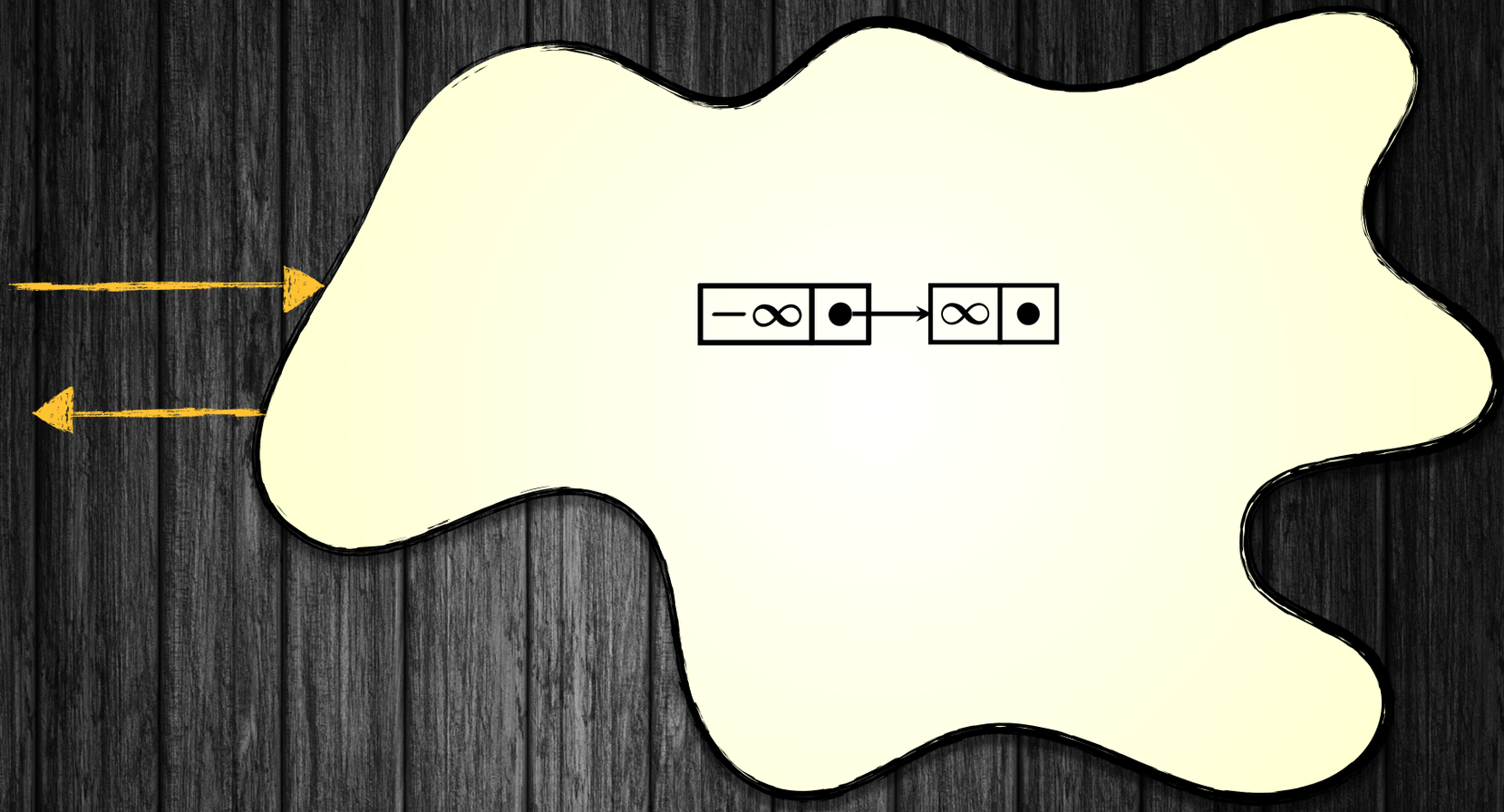
Sets | Sequential Programs

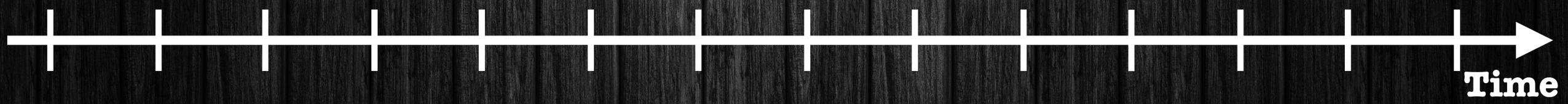


Sets | Sequential Programs



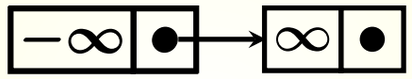
Sets | Sequential Programs





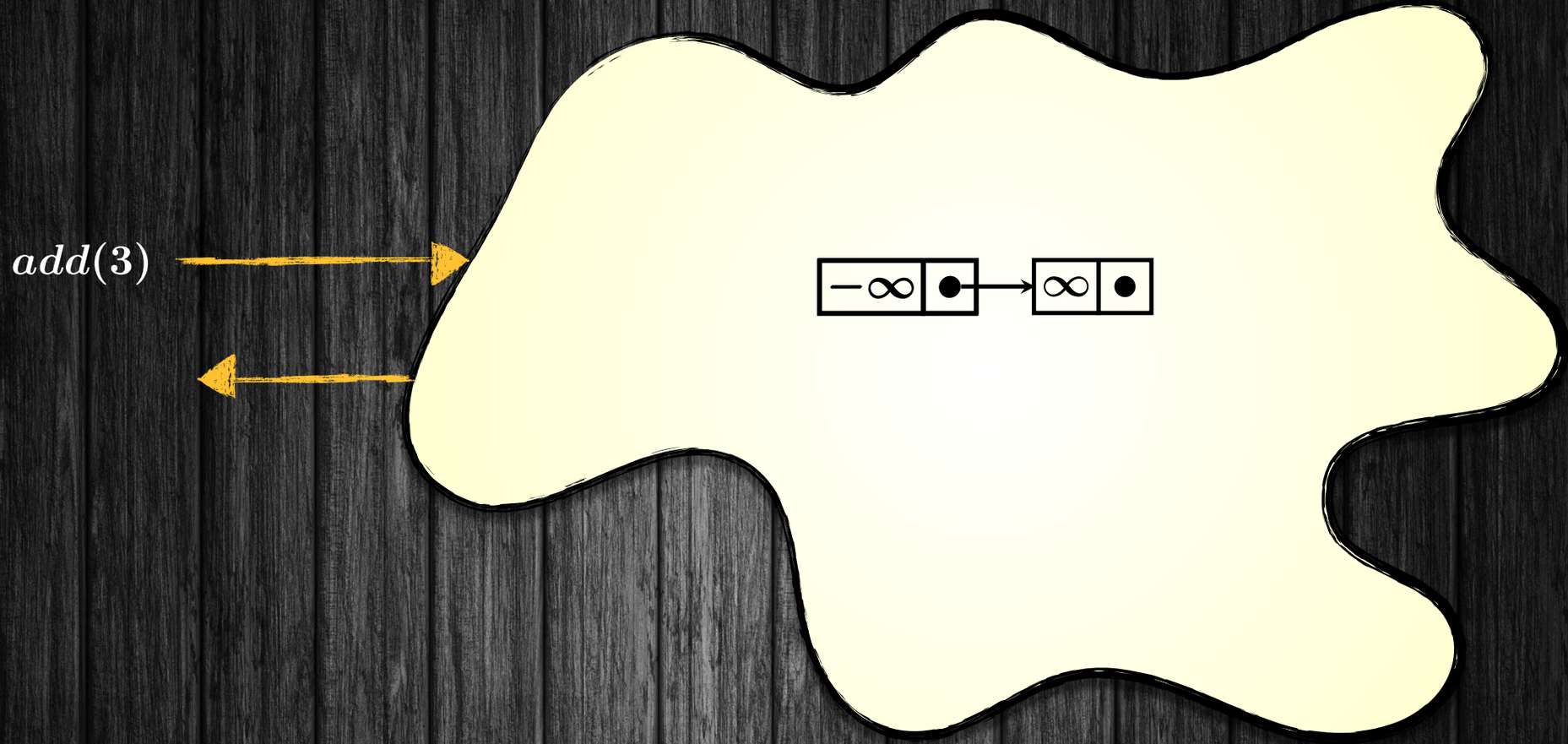
Sets | Sequential Programs

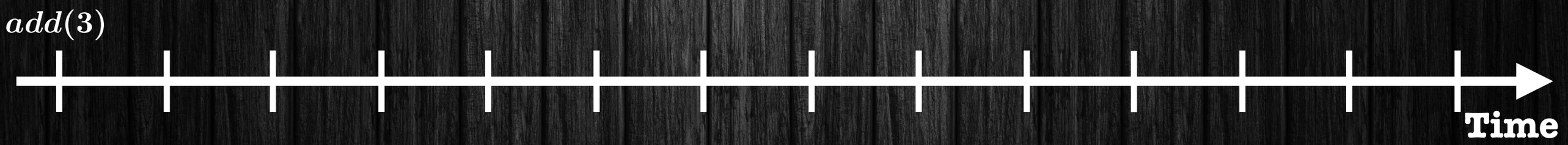
$add(3)$



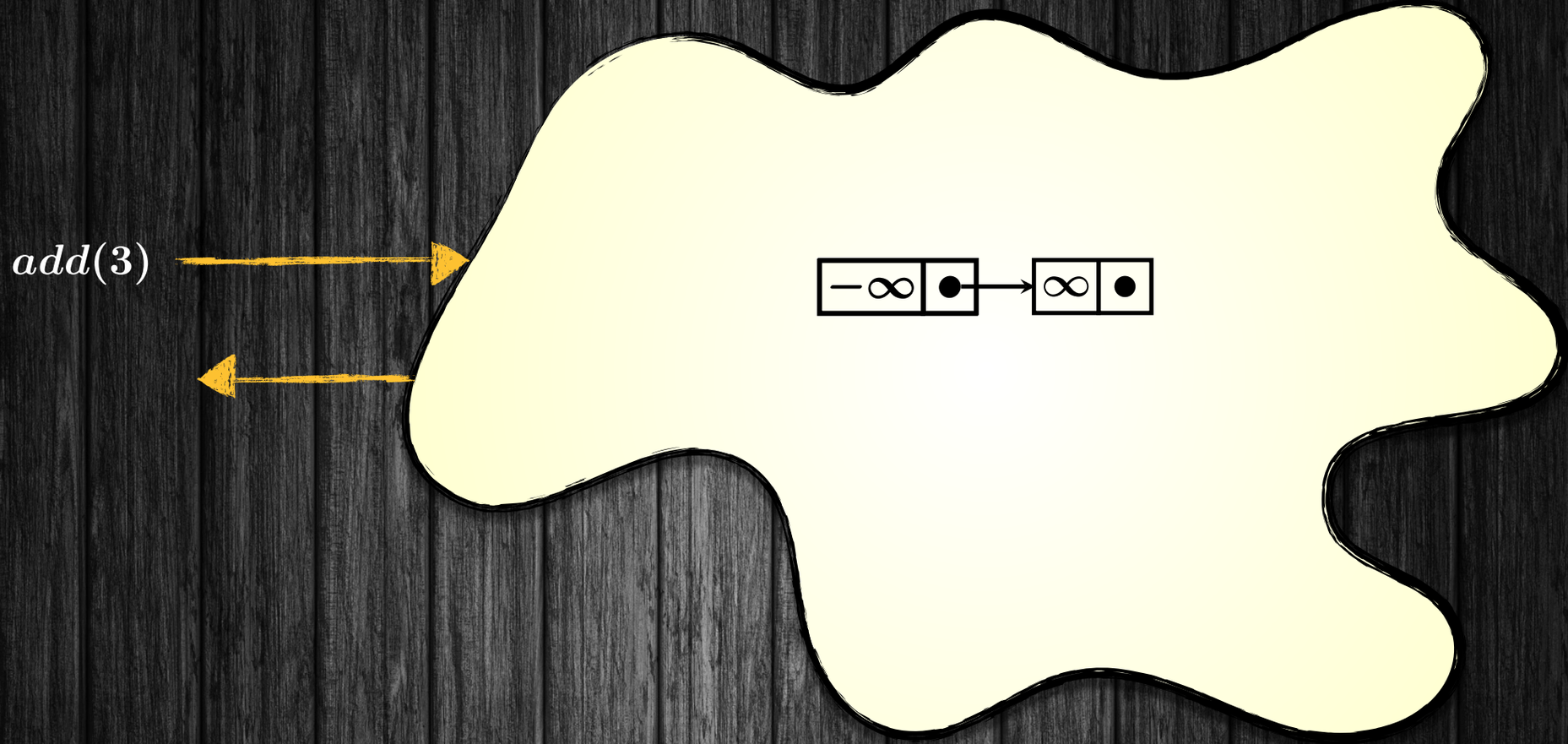


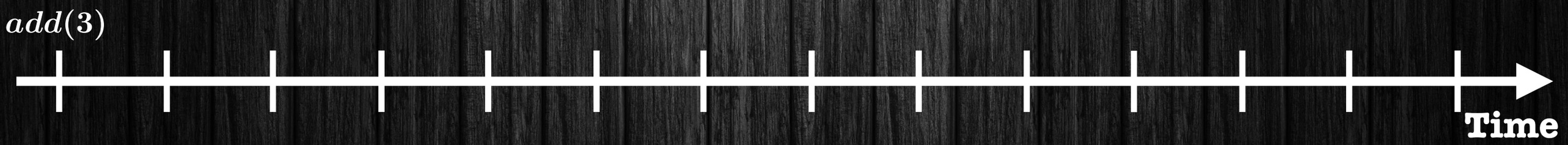
Sets | Sequential Programs



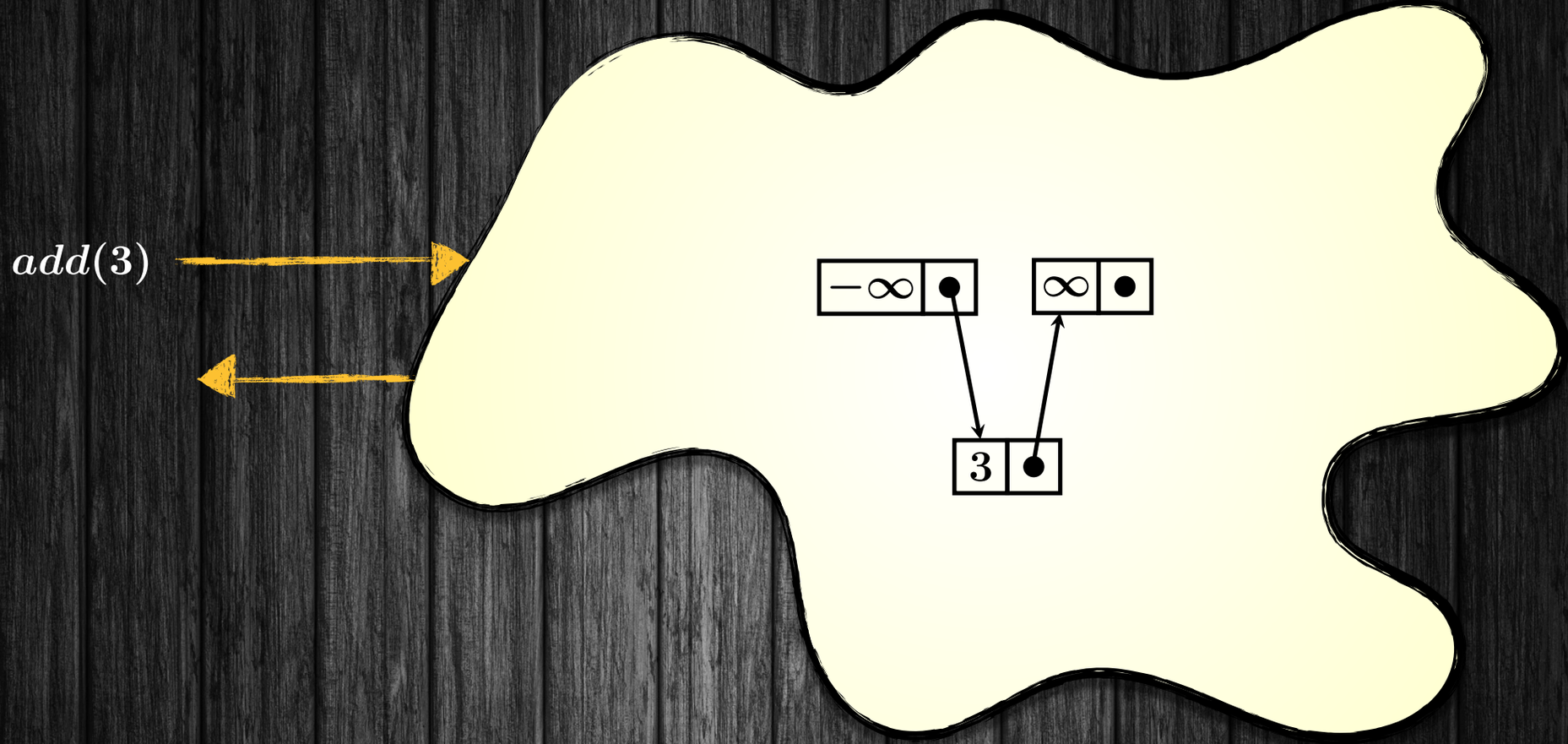


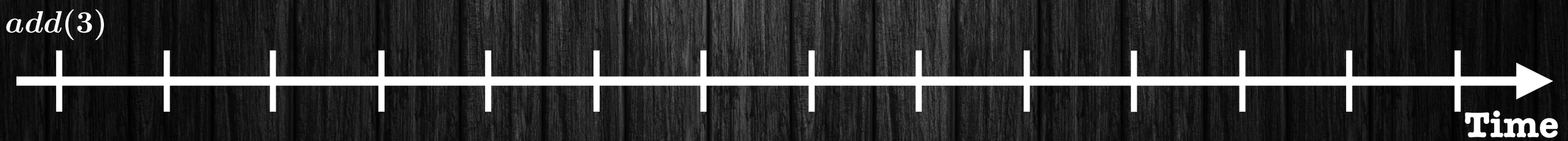
Sets | Sequential Programs



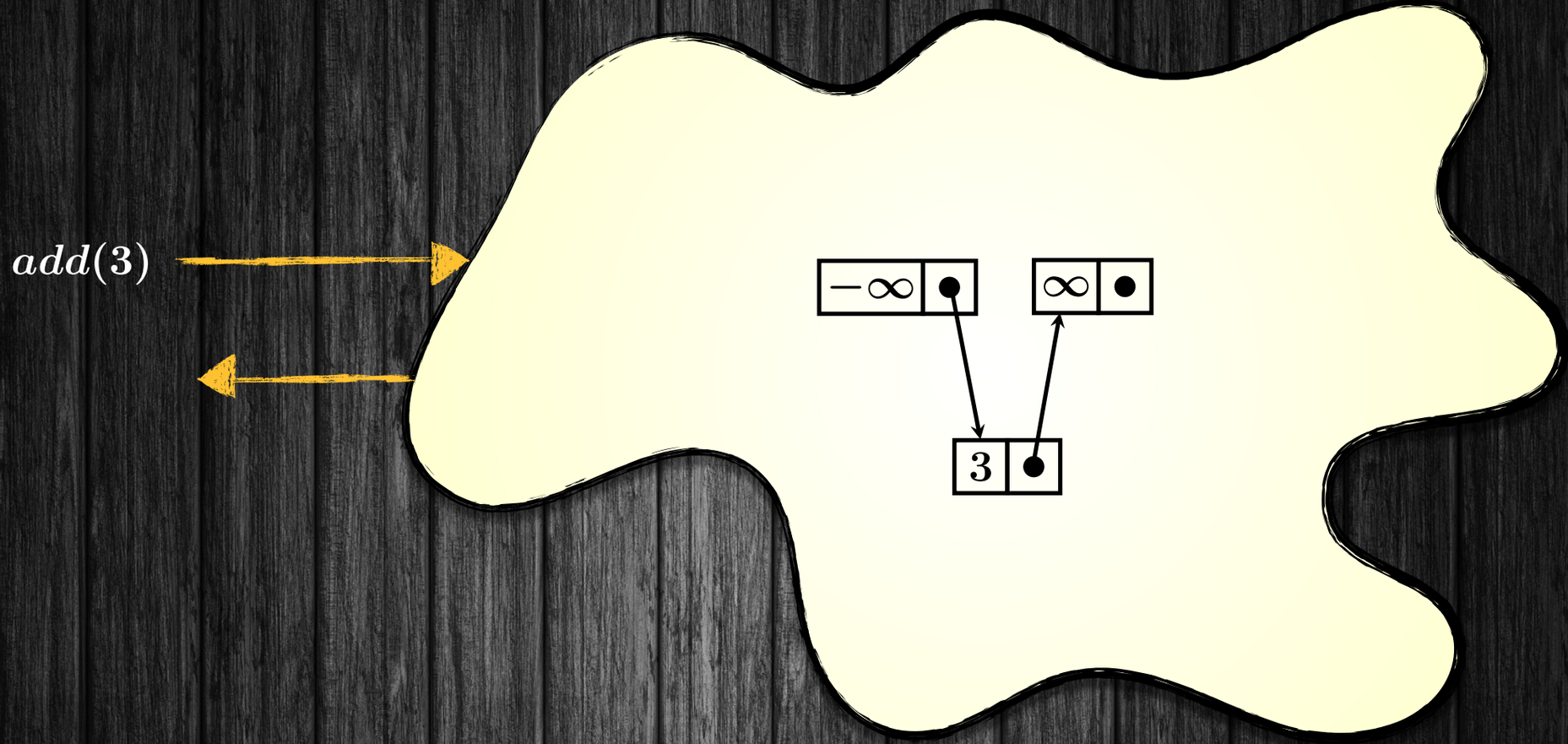


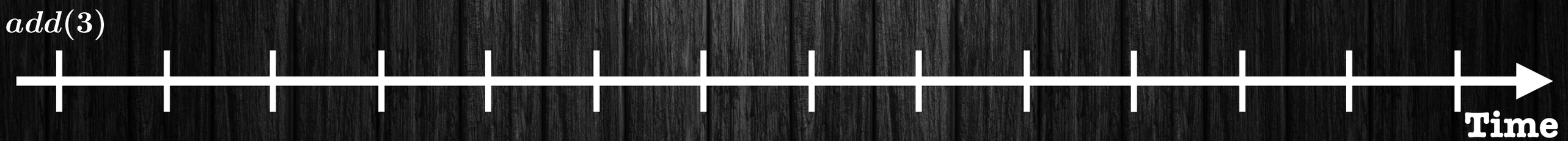
Sets | Sequential Programs



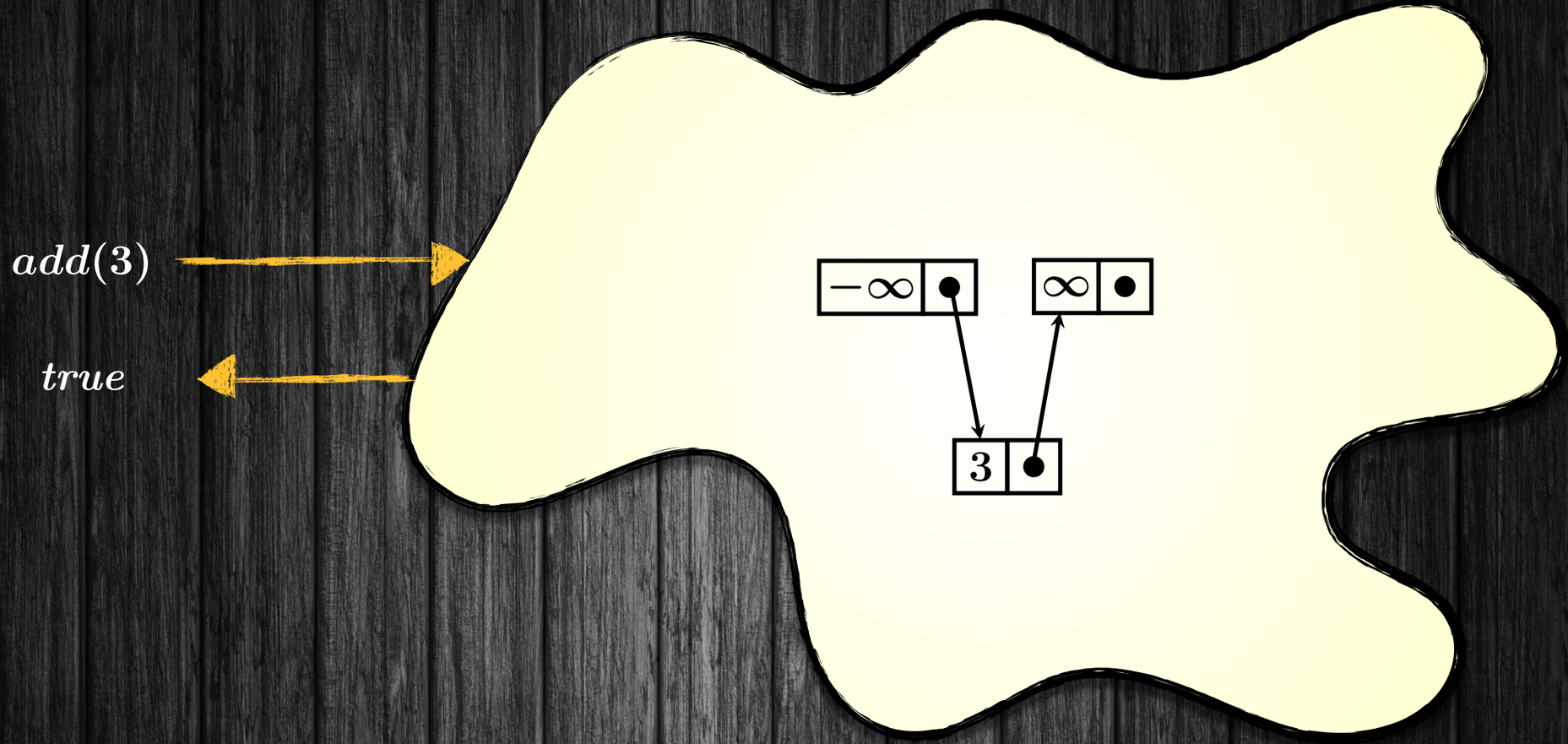


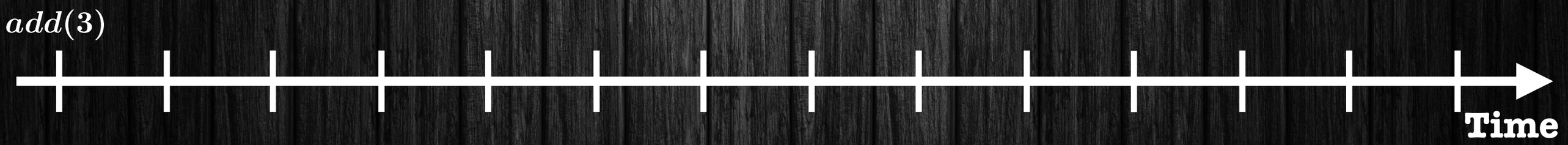
Sets | Sequential Programs



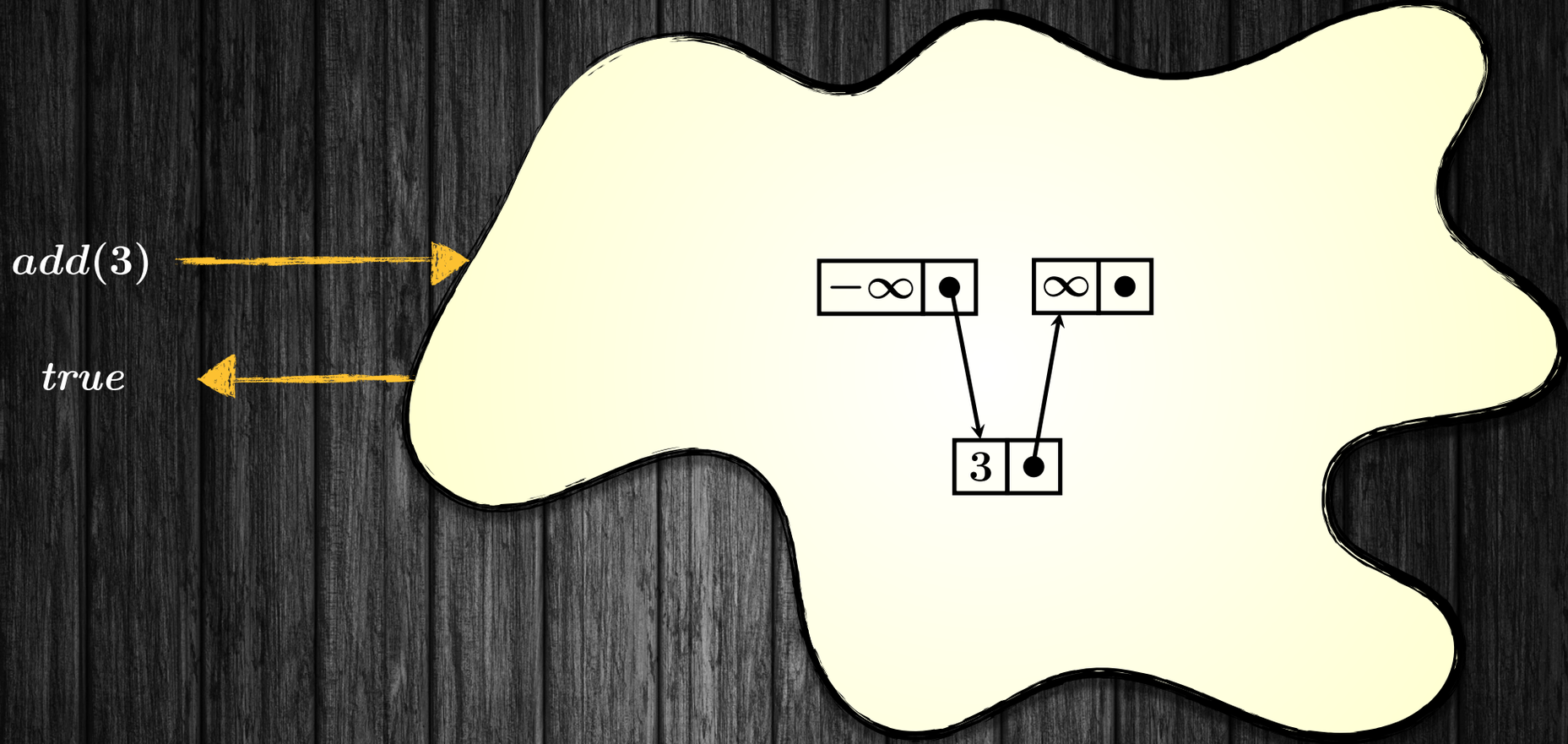


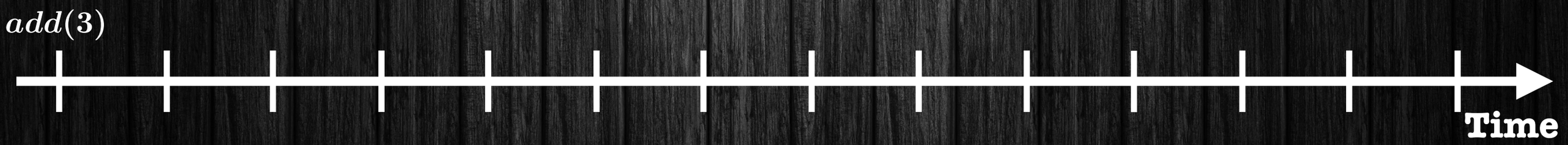
Sets | Sequential Programs



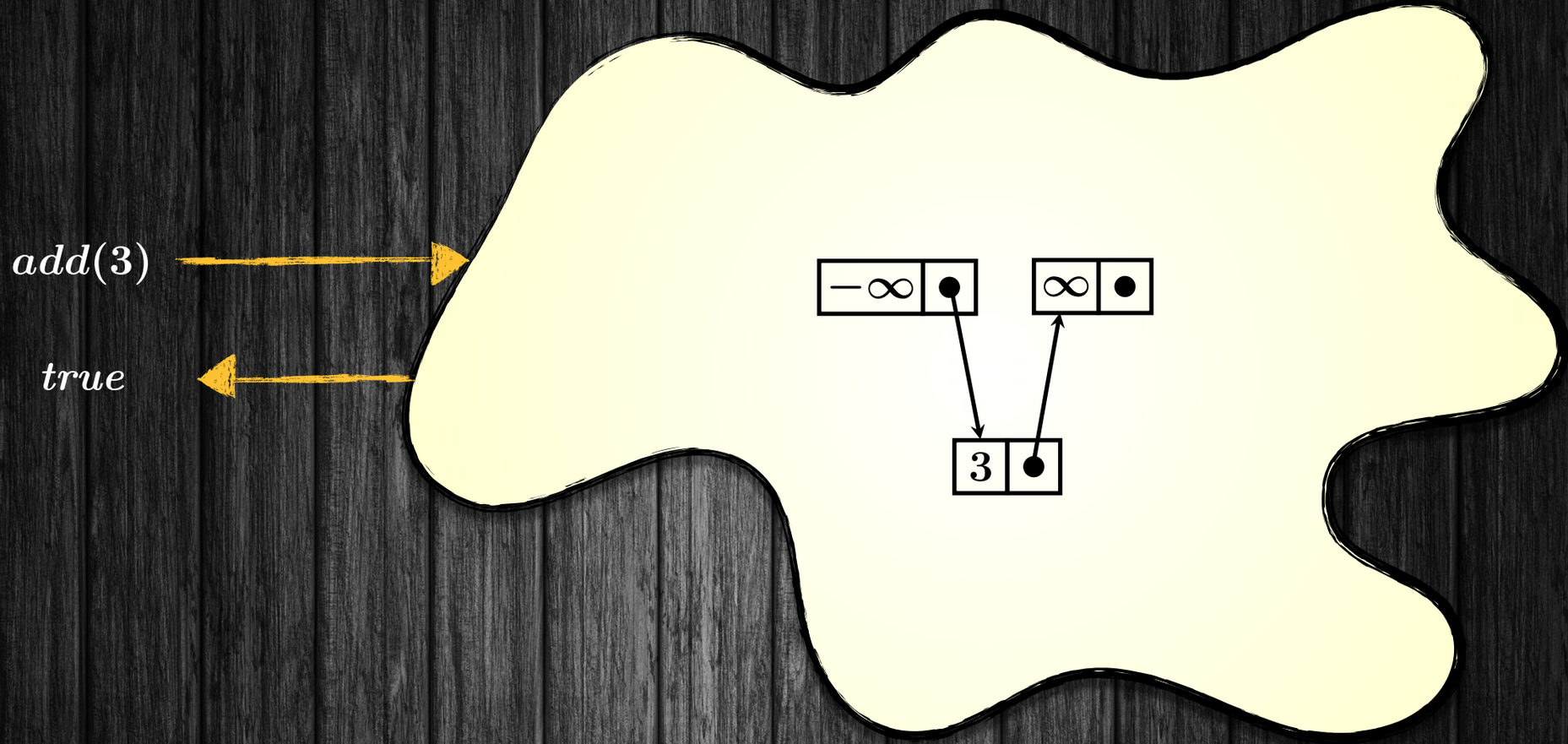


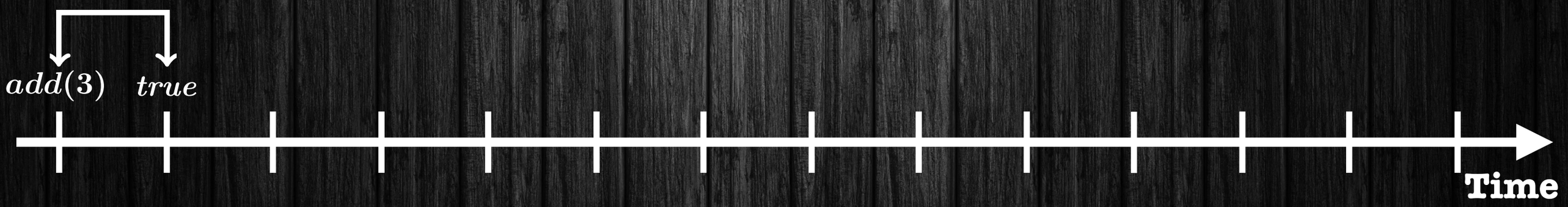
Sets | Sequential Programs



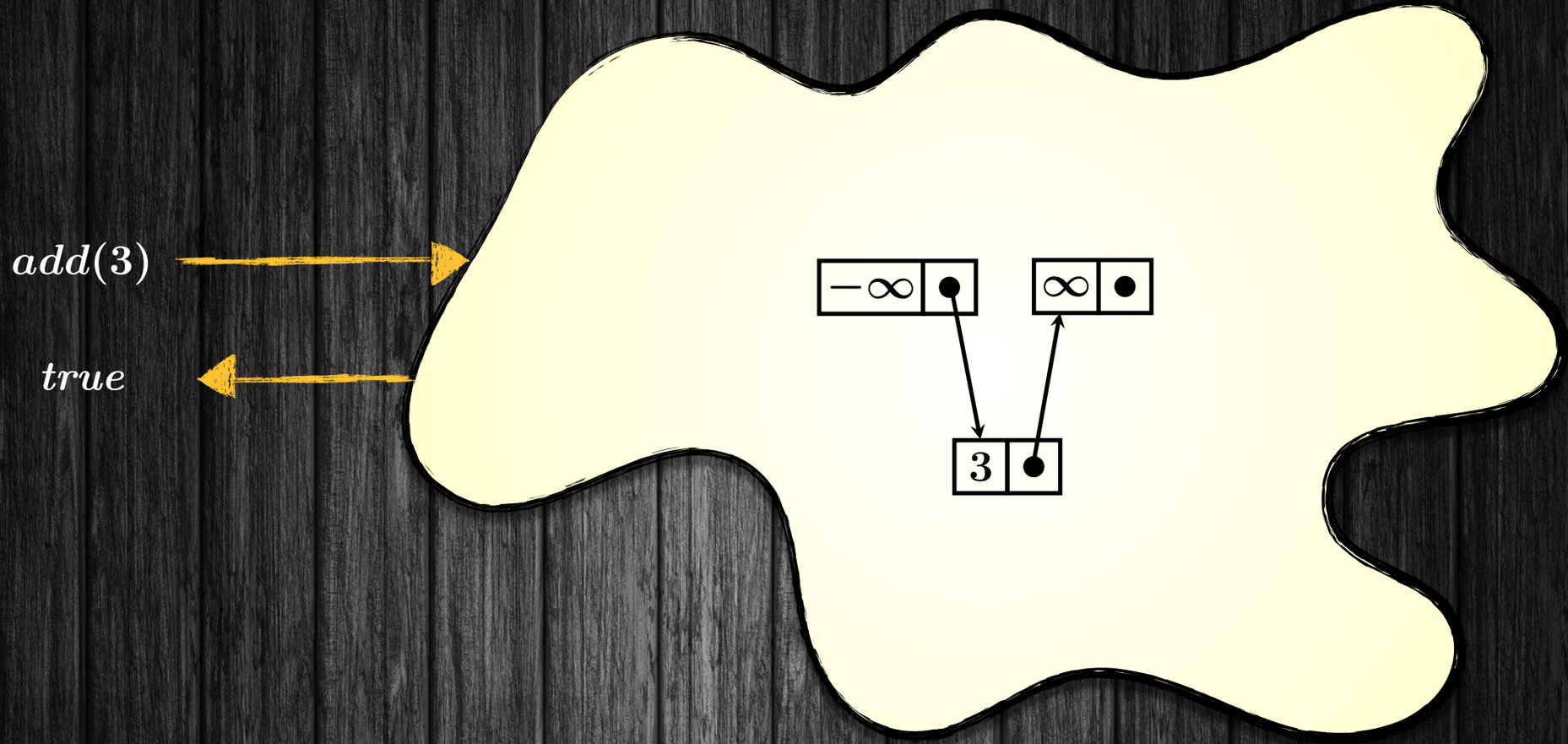


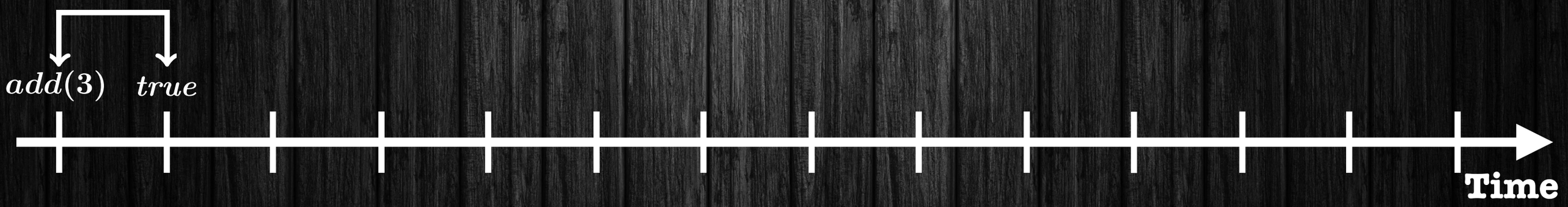
Sets | Sequential Programs



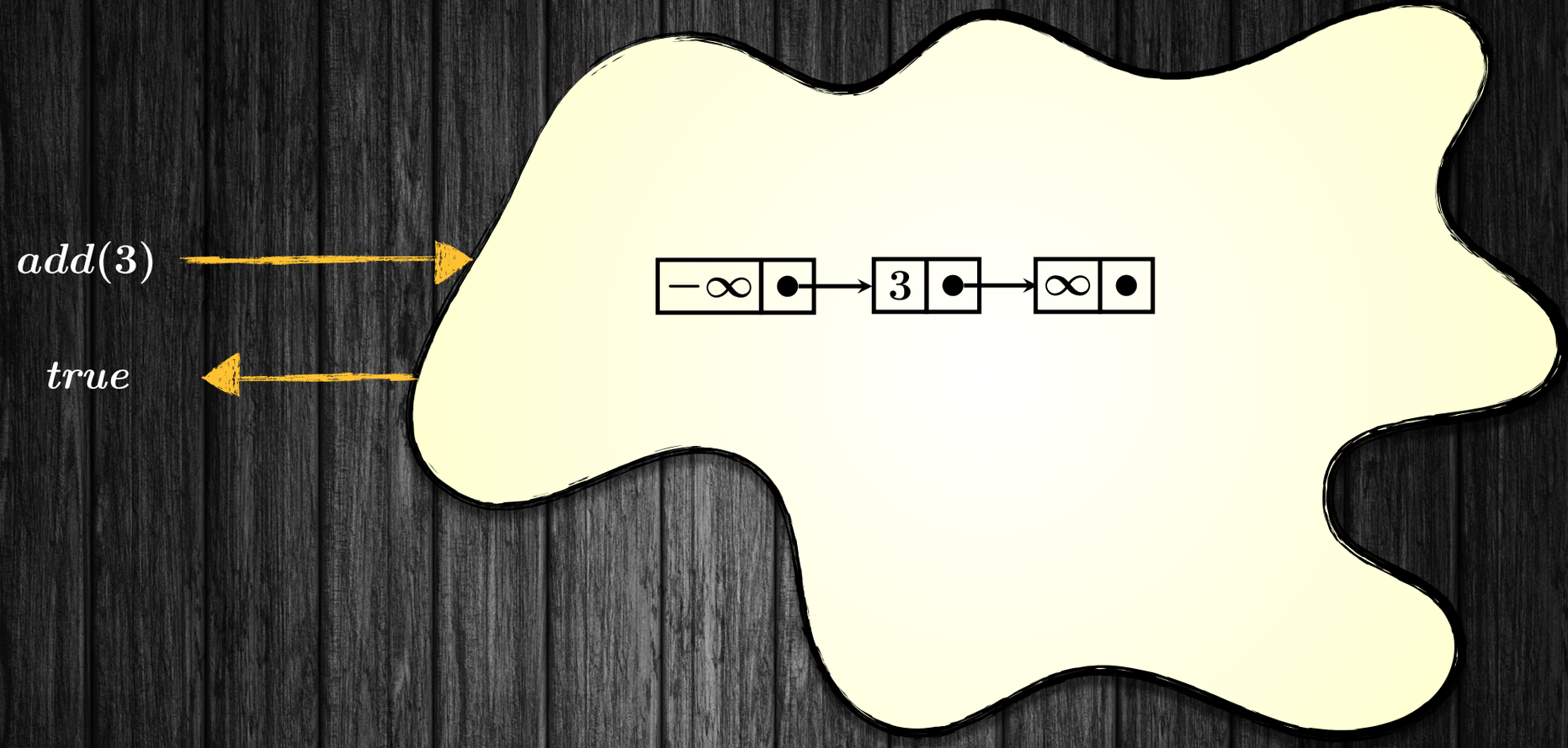


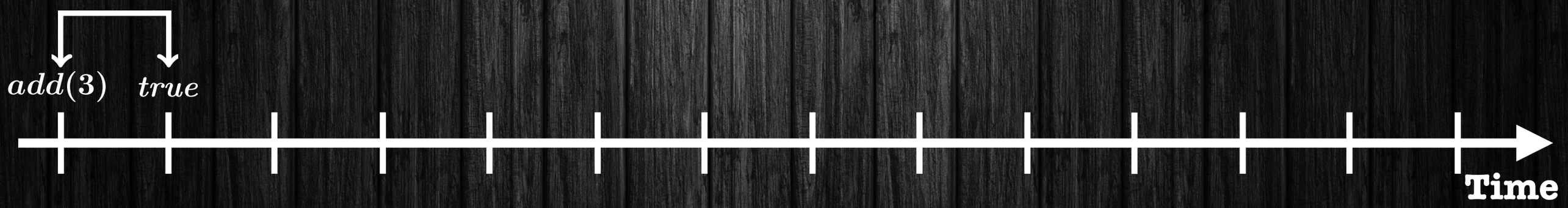
Sets | Sequential Programs



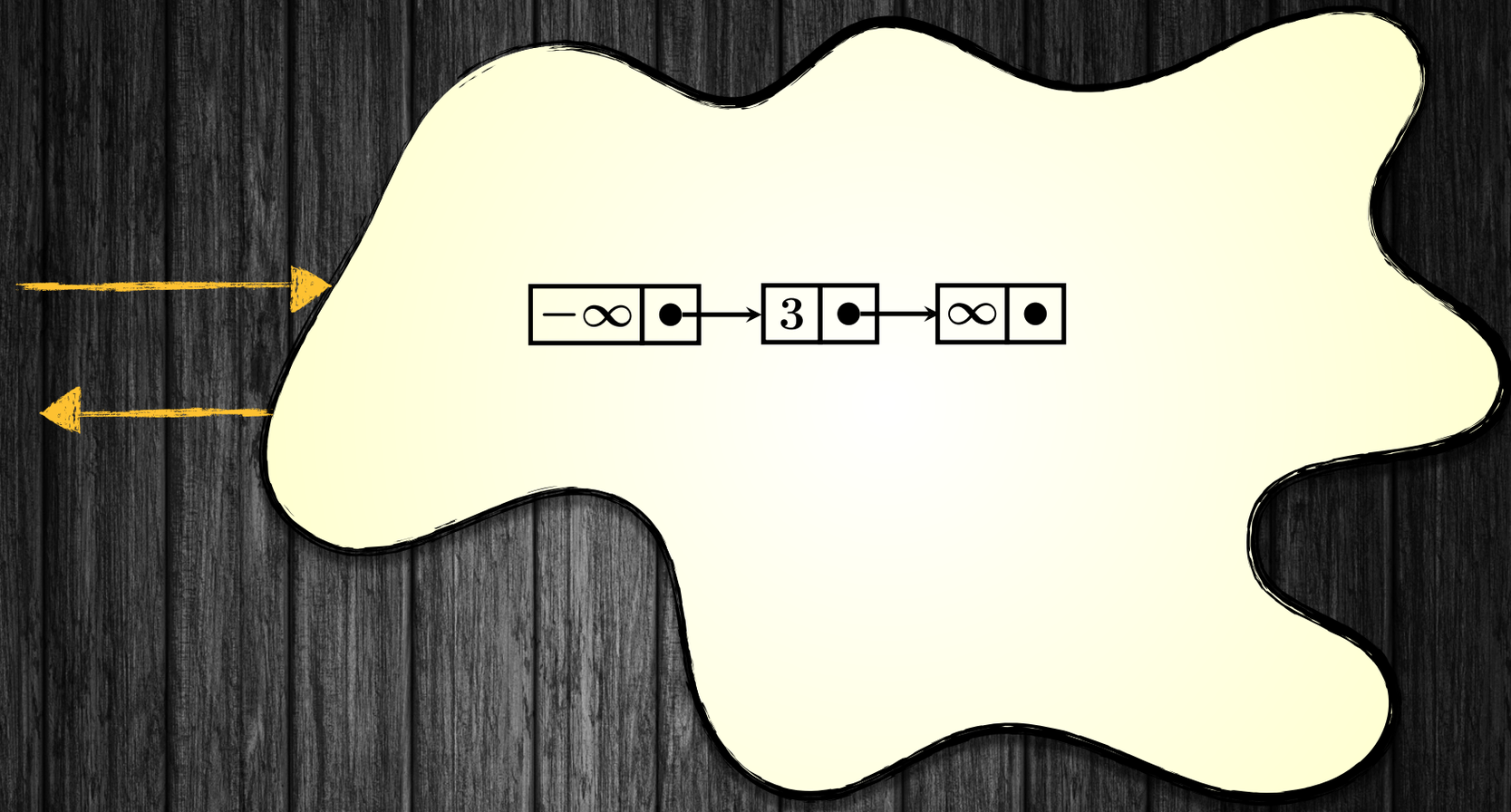


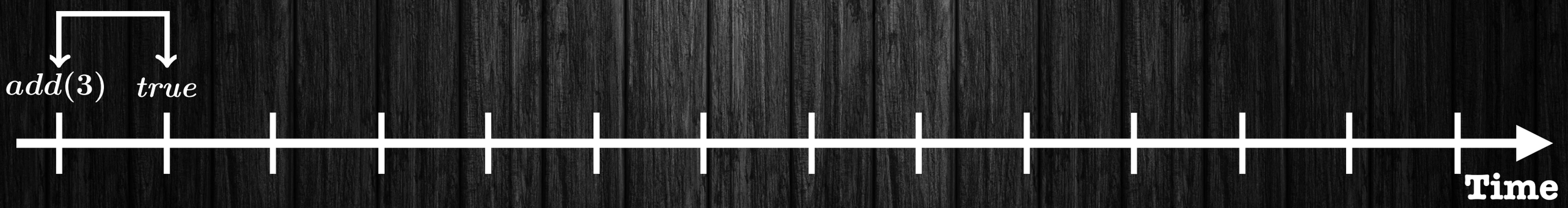
Sets | Sequential Programs



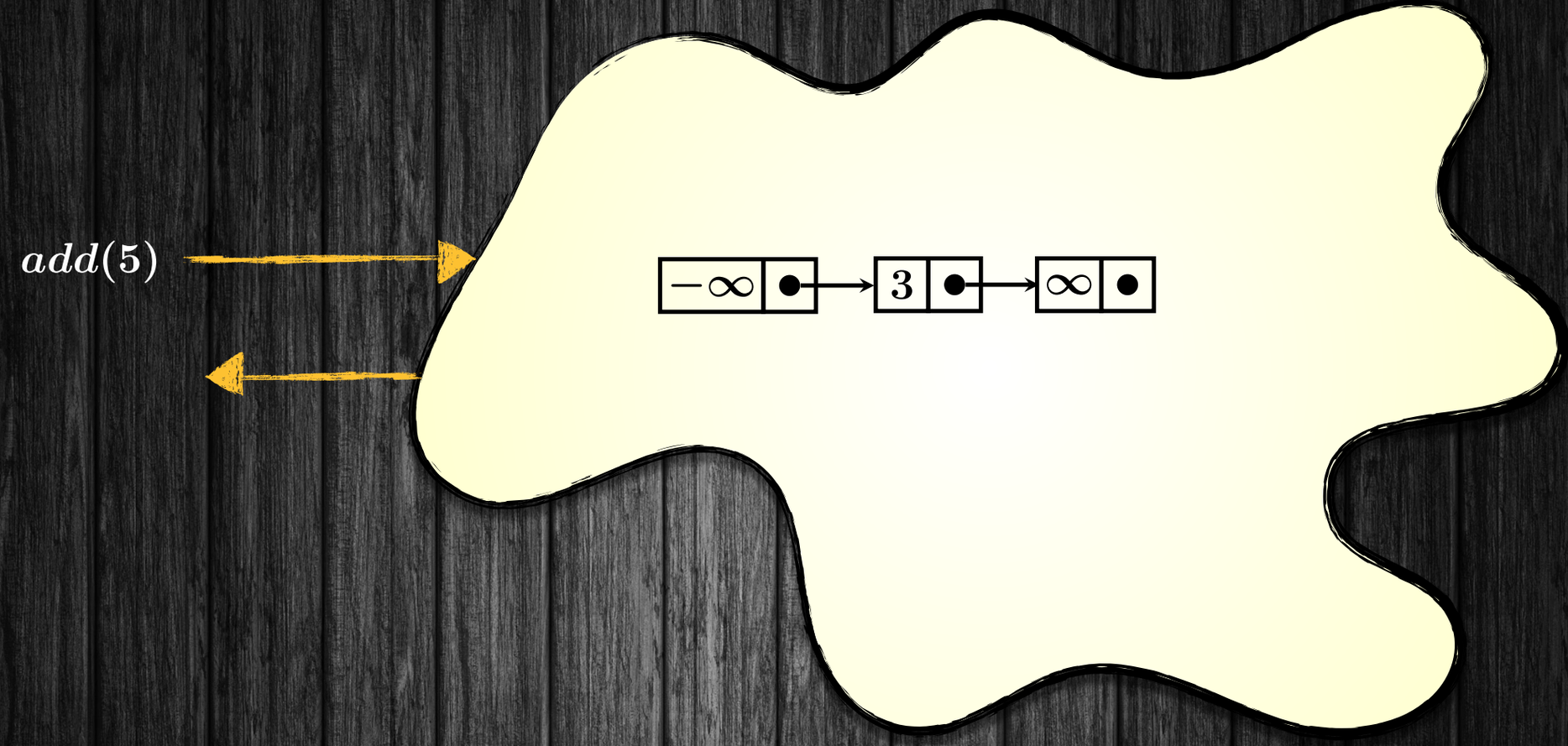


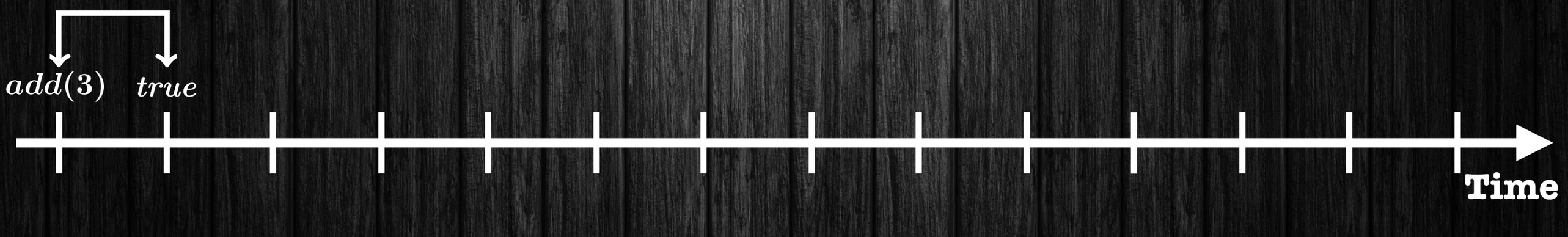
Sets | Sequential Programs



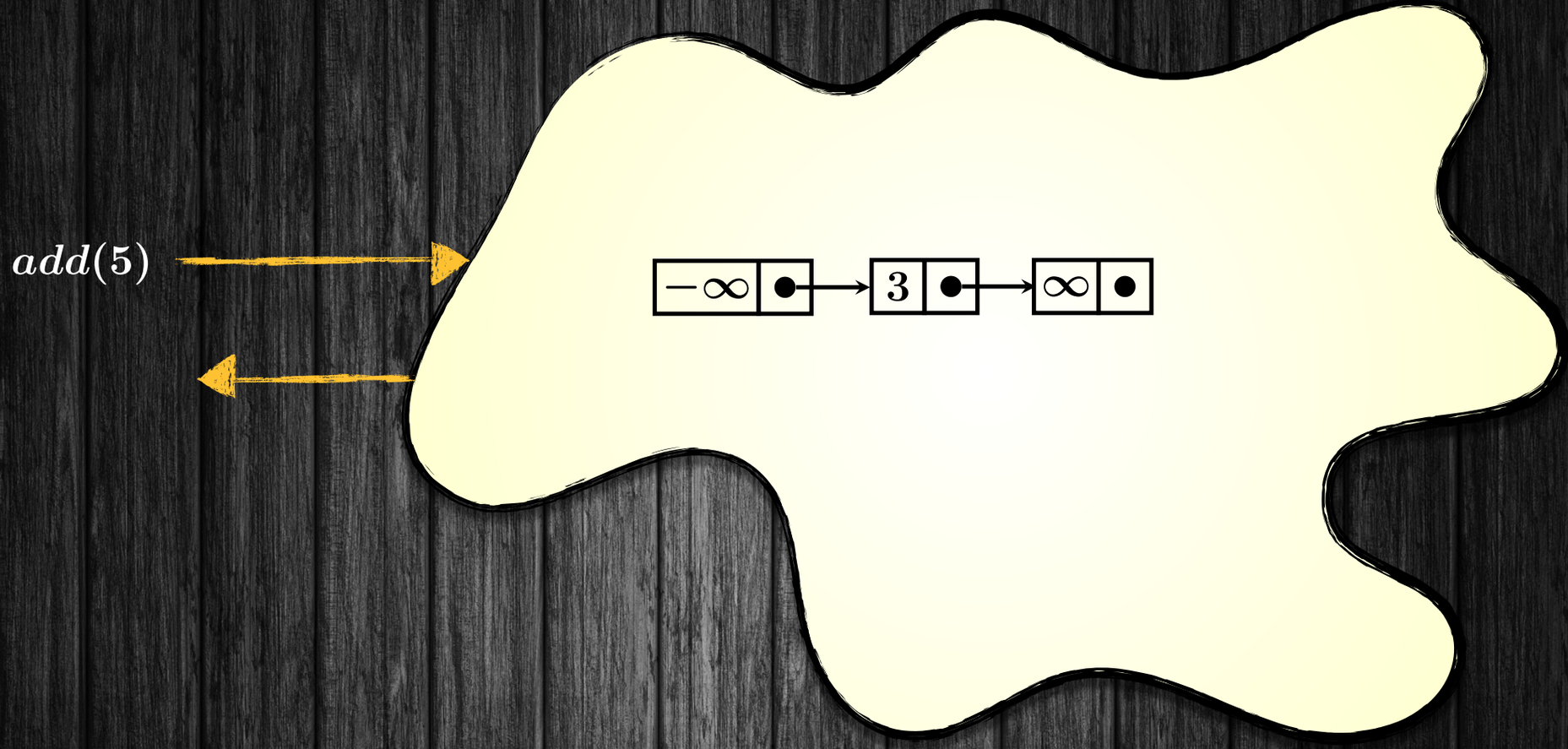


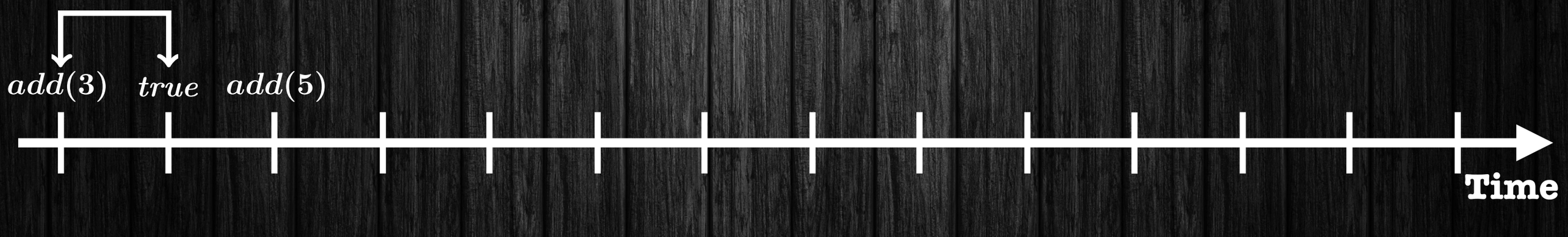
Sets | Sequential Programs



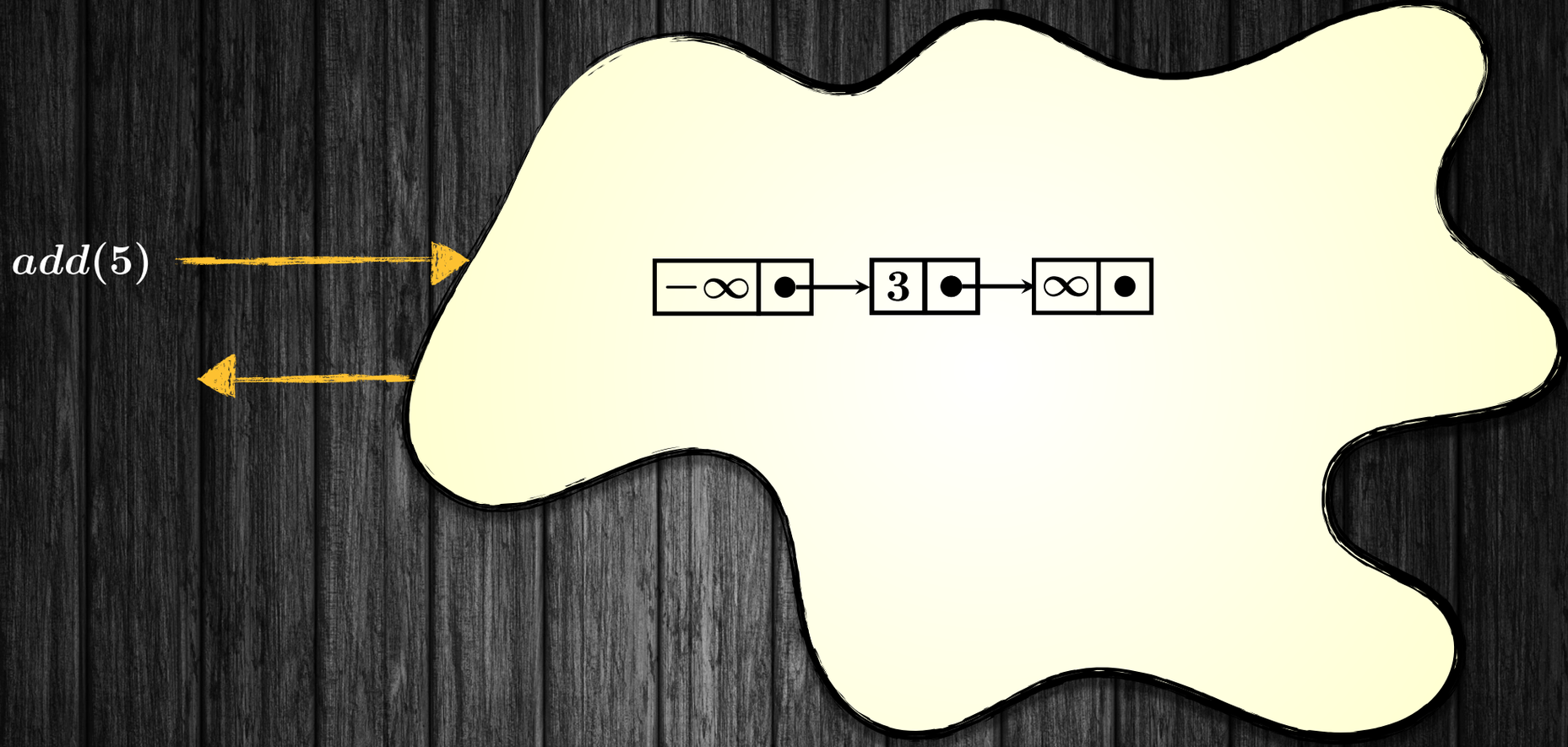


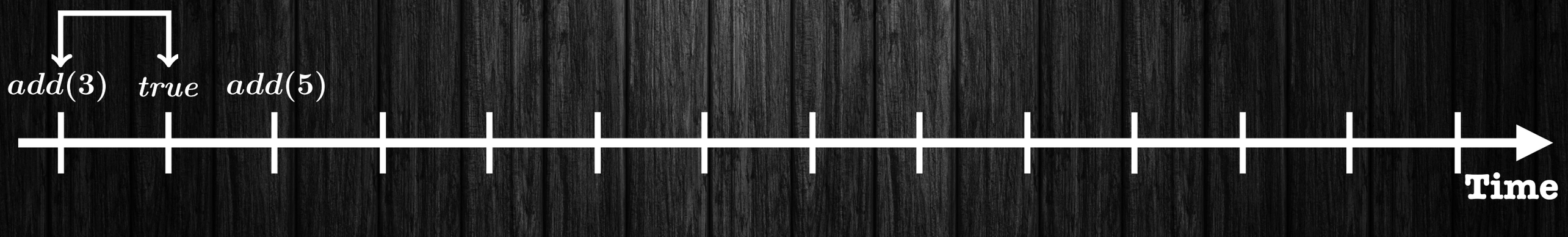
Sets | Sequential Programs



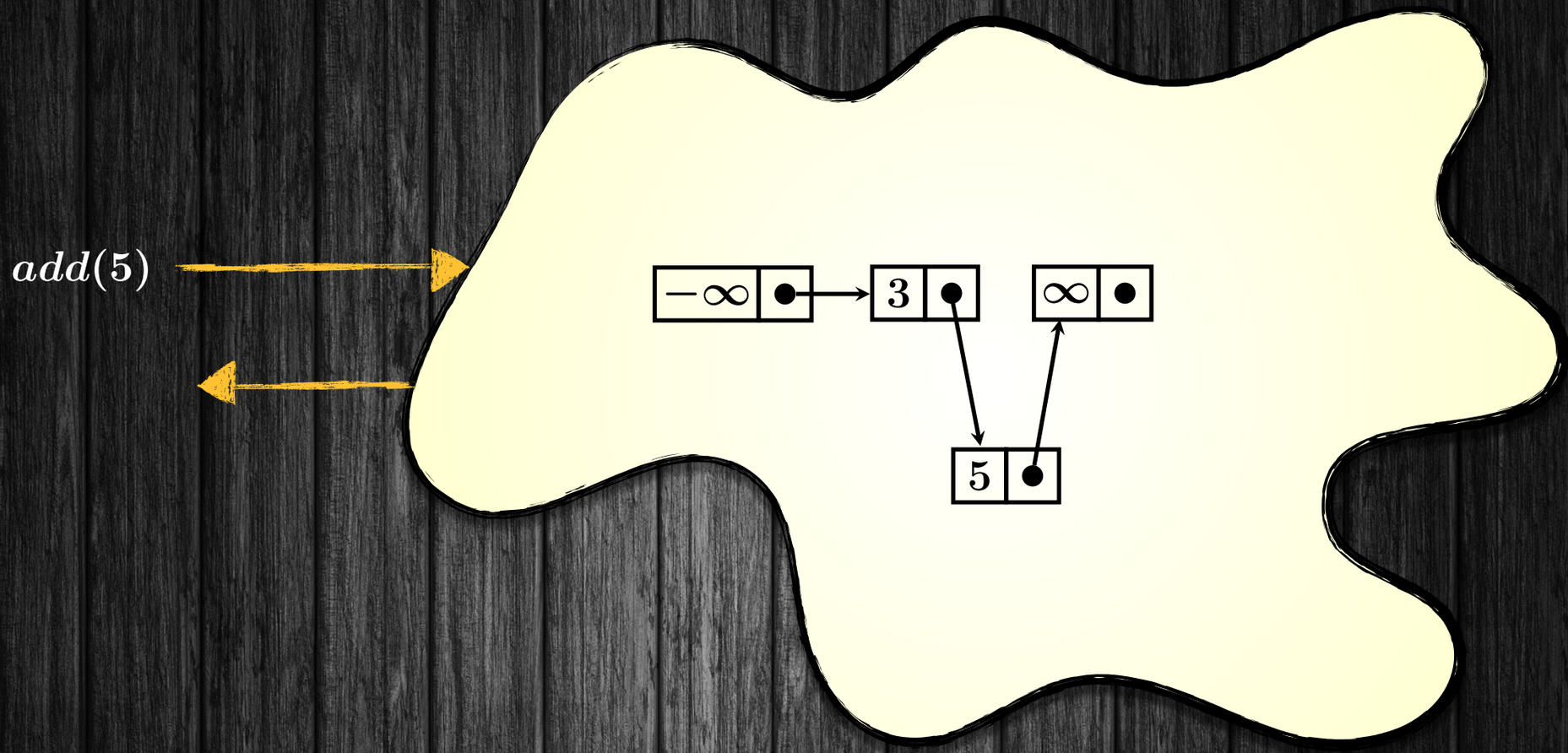


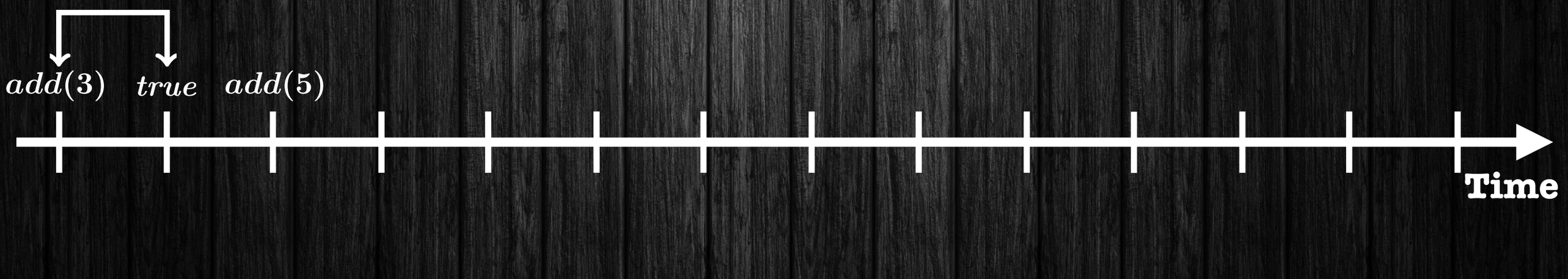
Sets | Sequential Programs



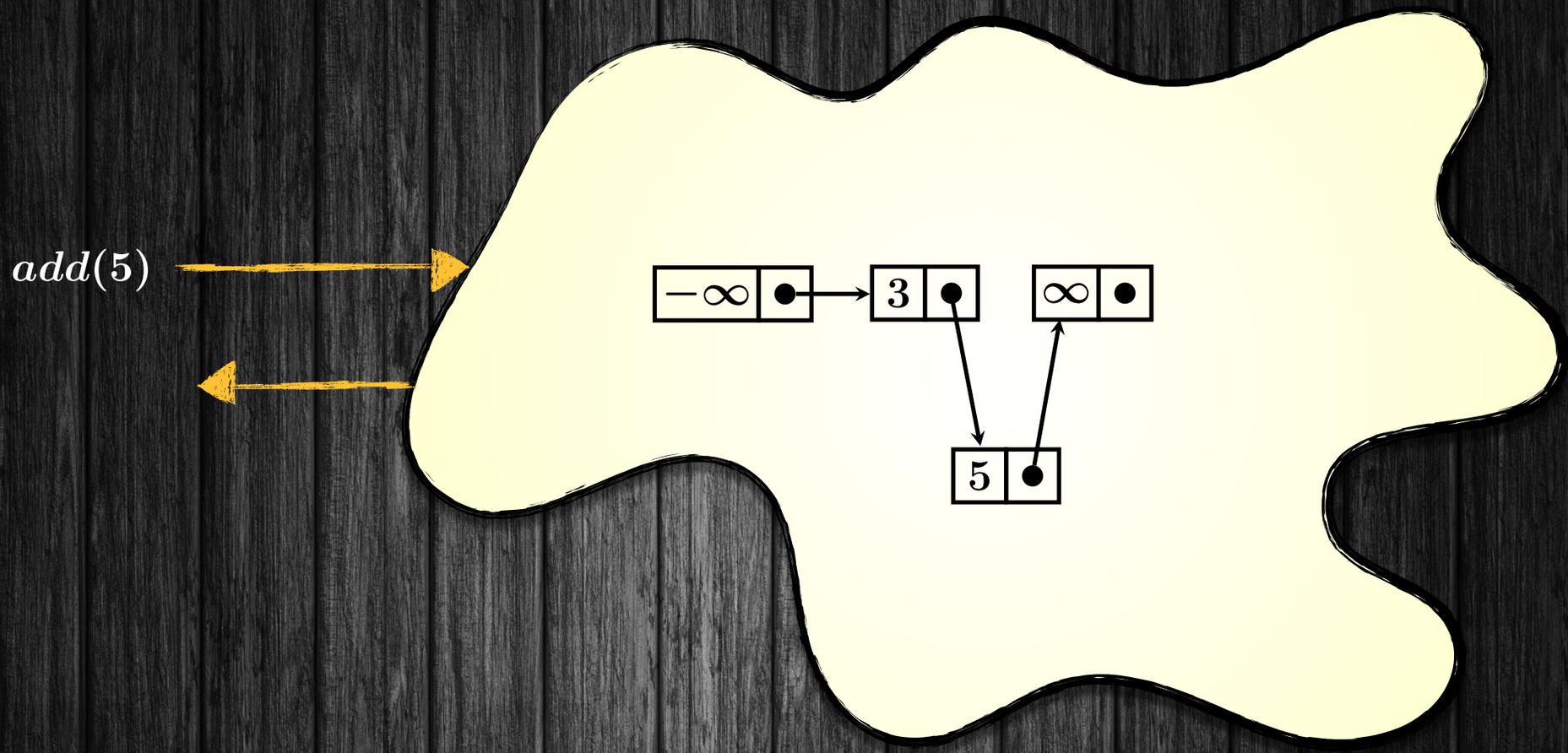


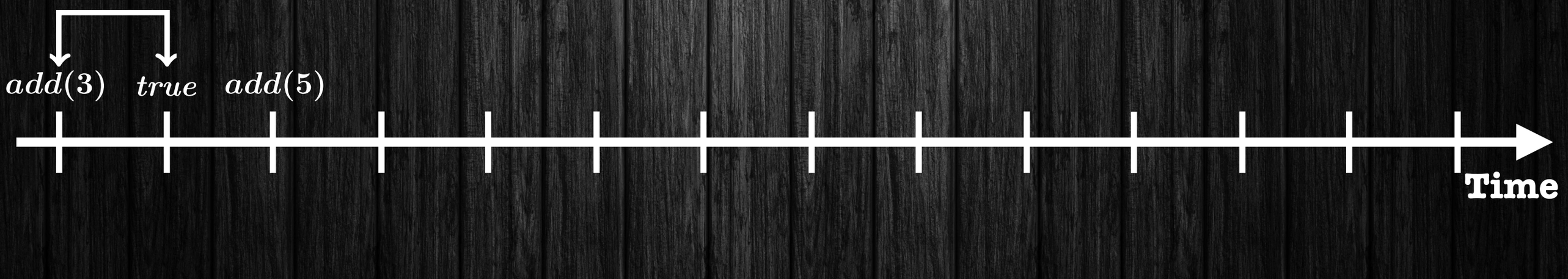
Sets Sequential Programs



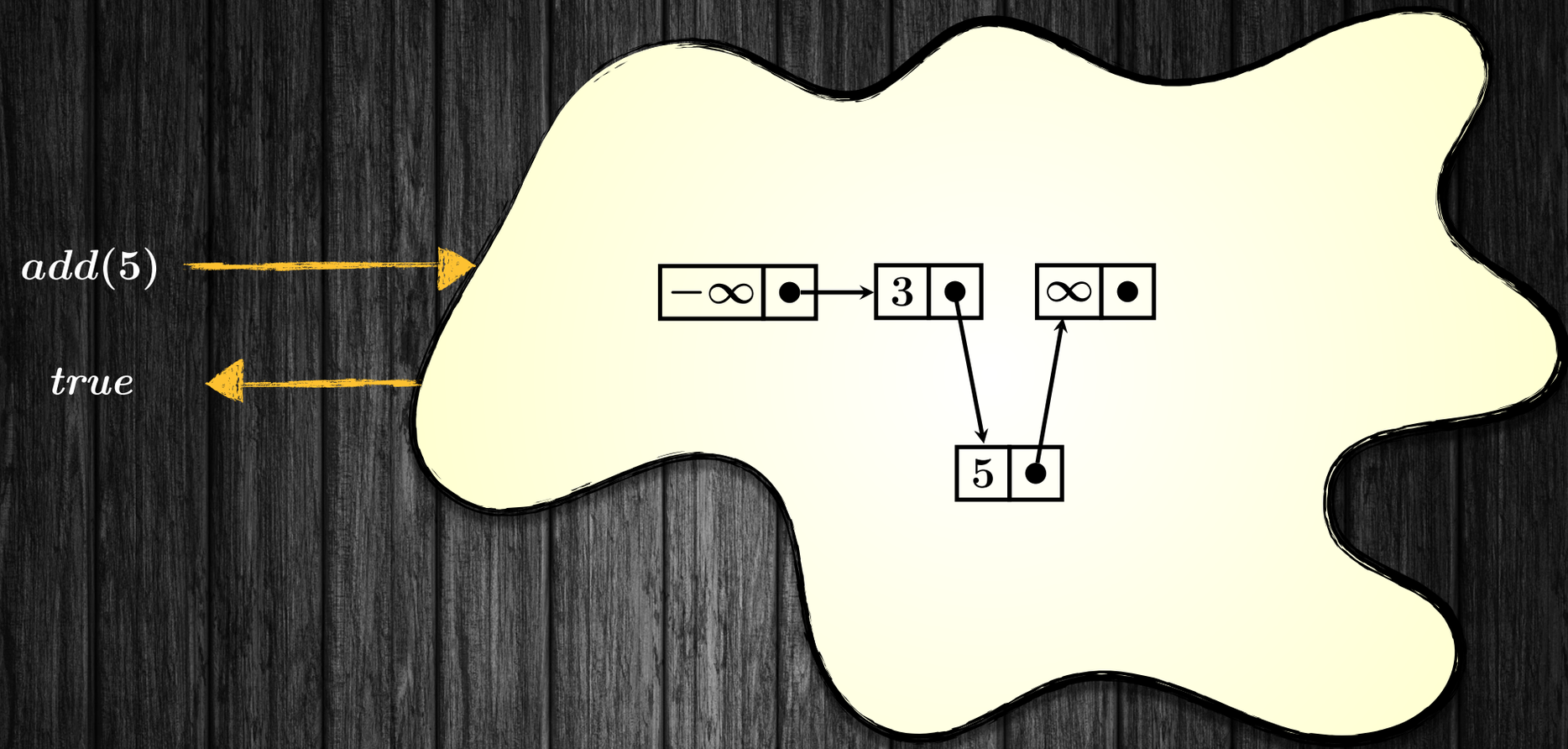


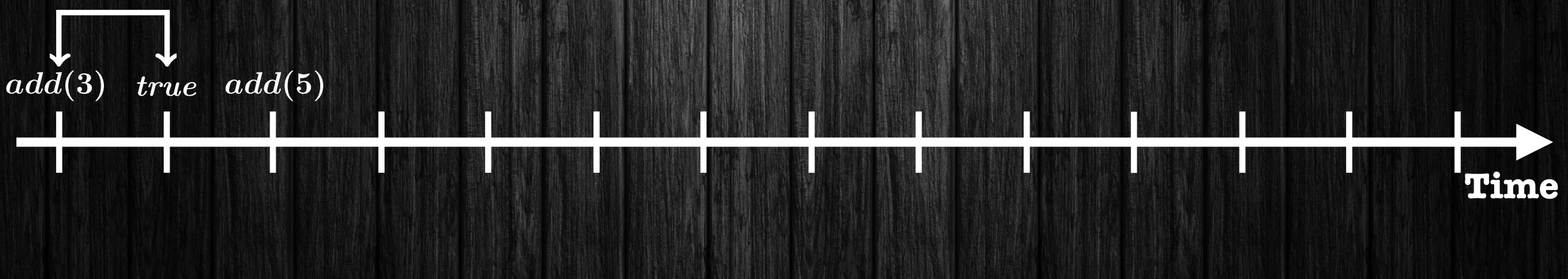
Sets Sequential Programs



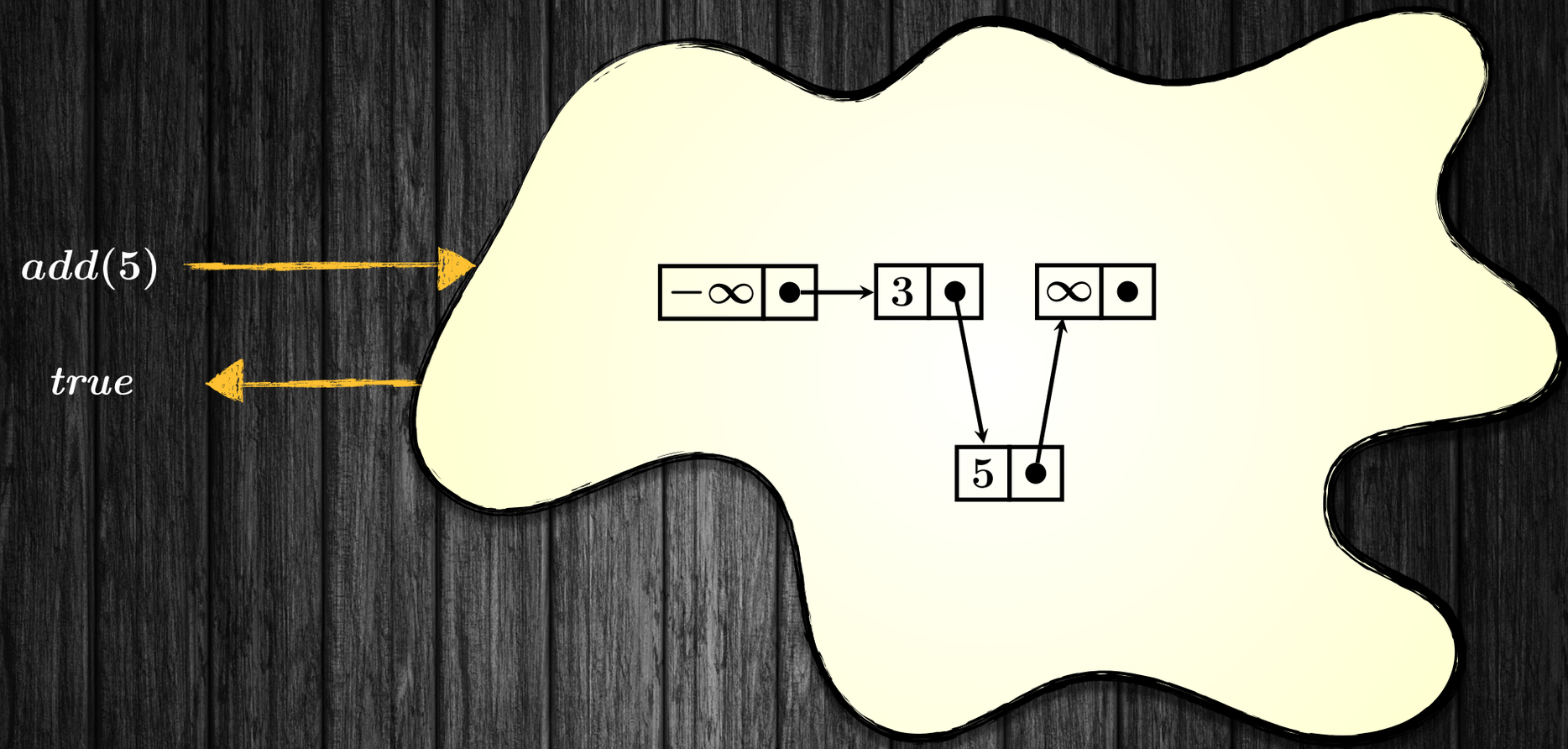


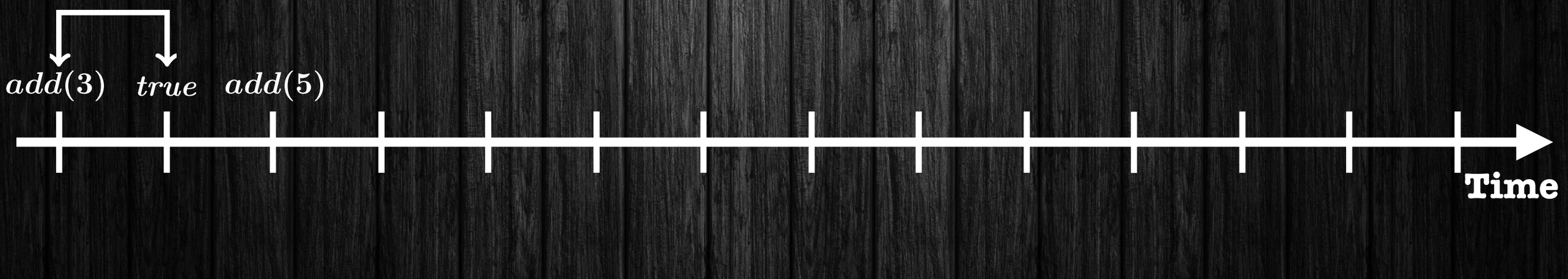
Sets | Sequential Programs



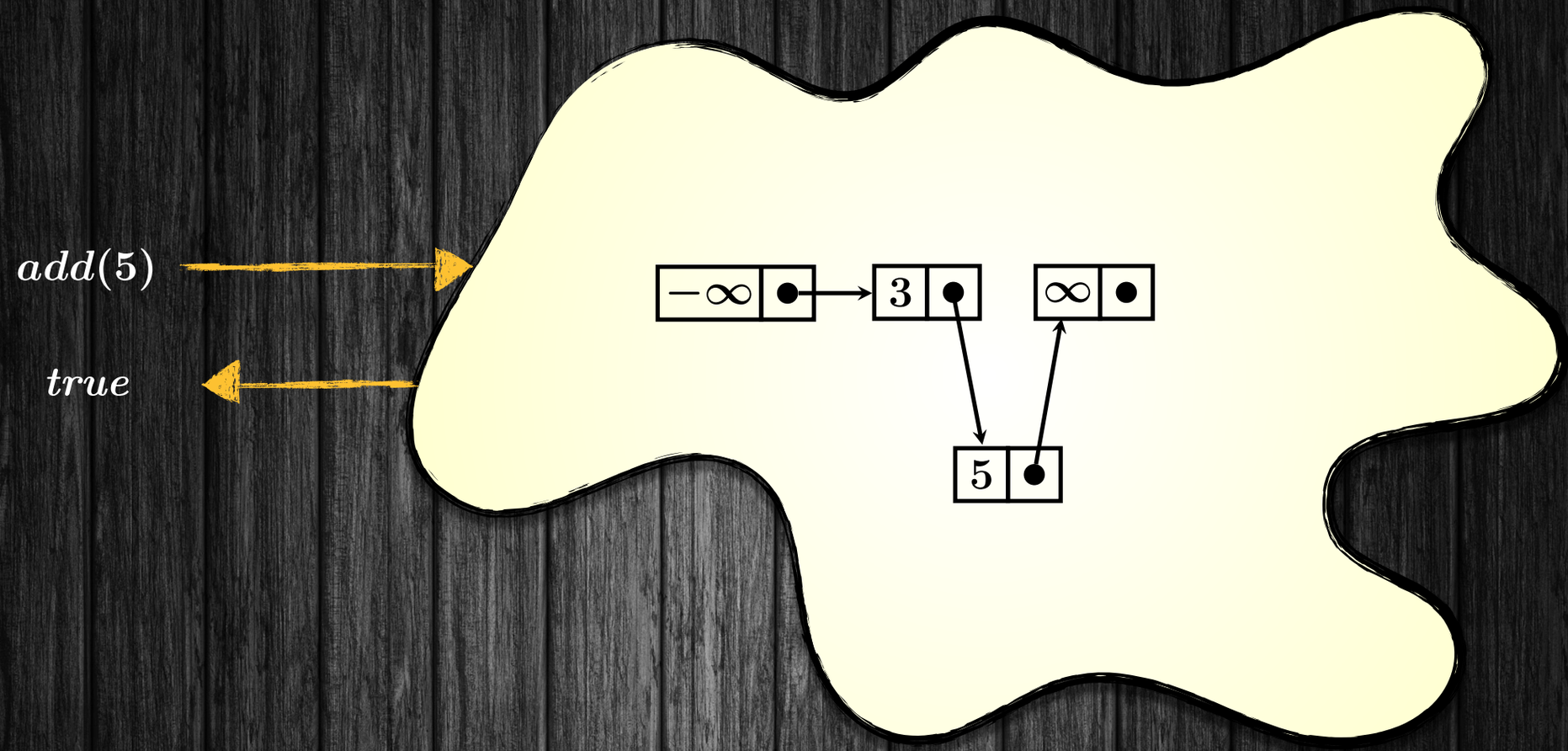


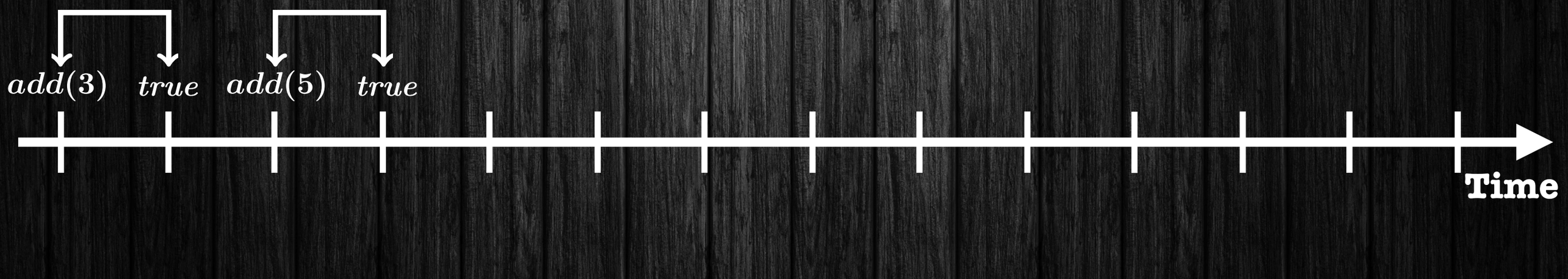
Sets | Sequential Programs



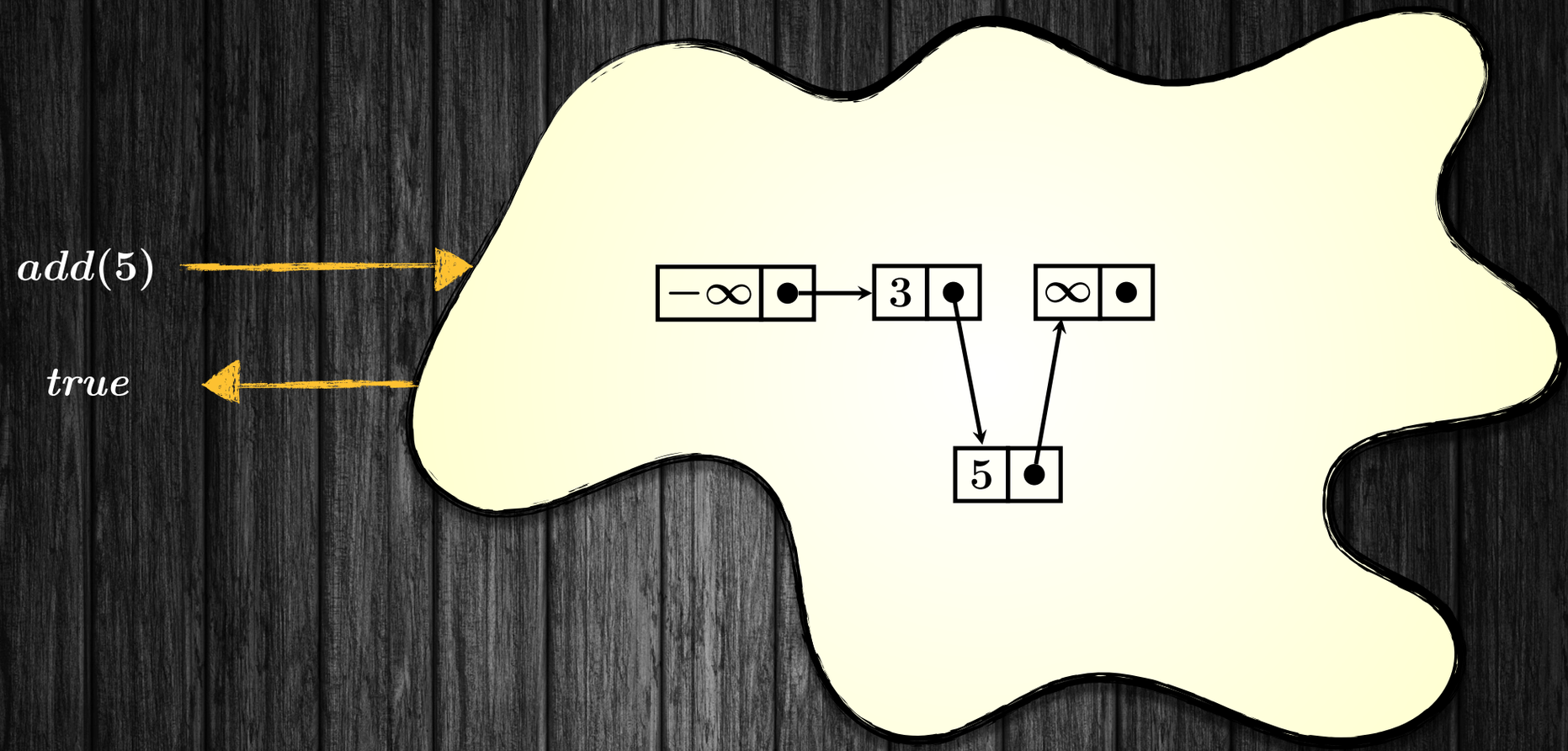


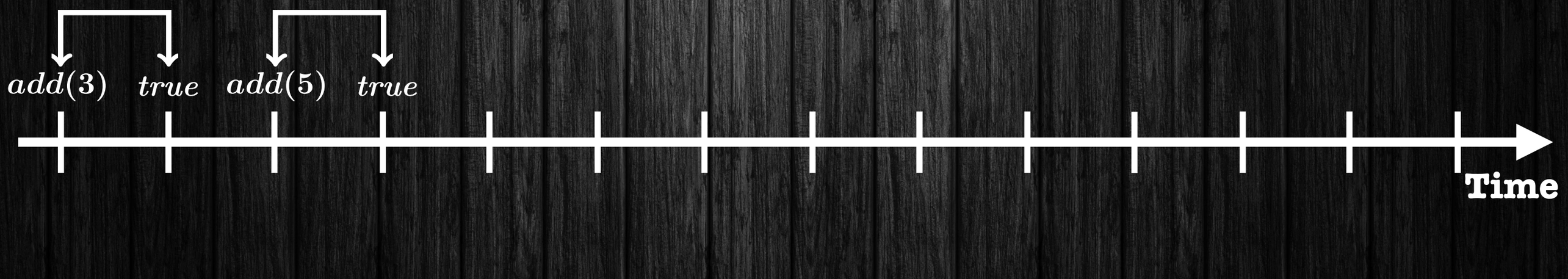
Sets | Sequential Programs



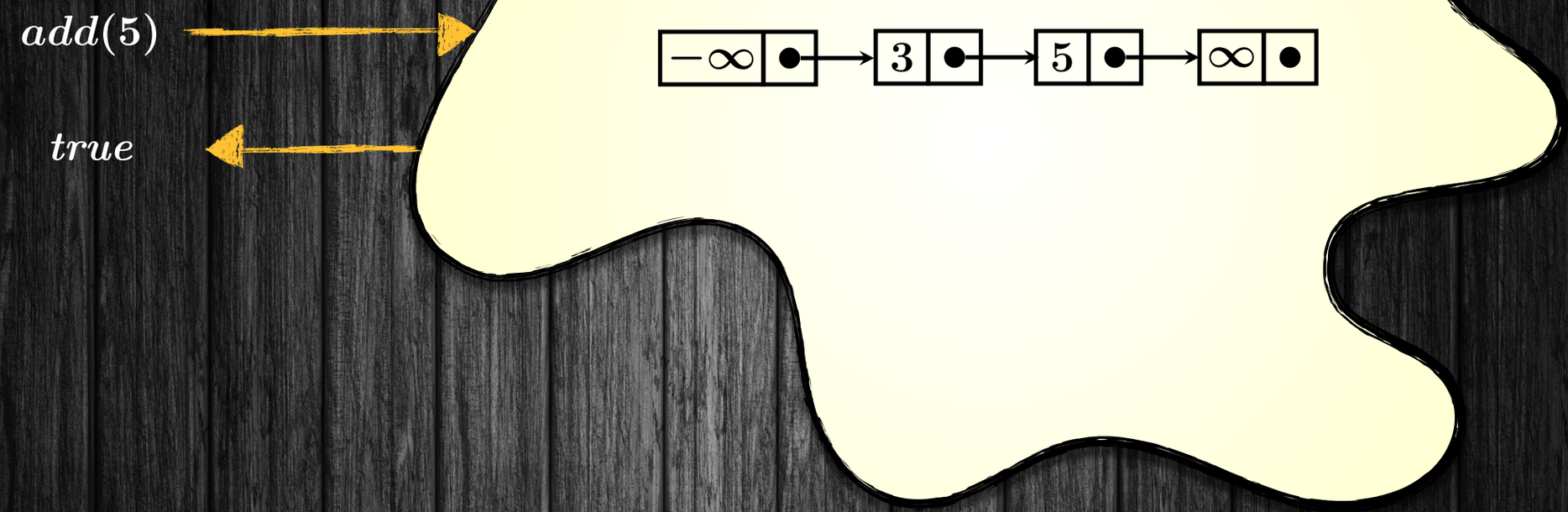


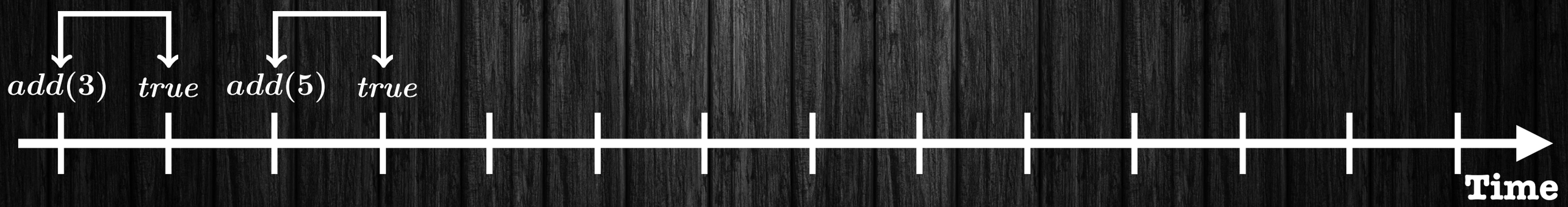
Sets | Sequential Programs



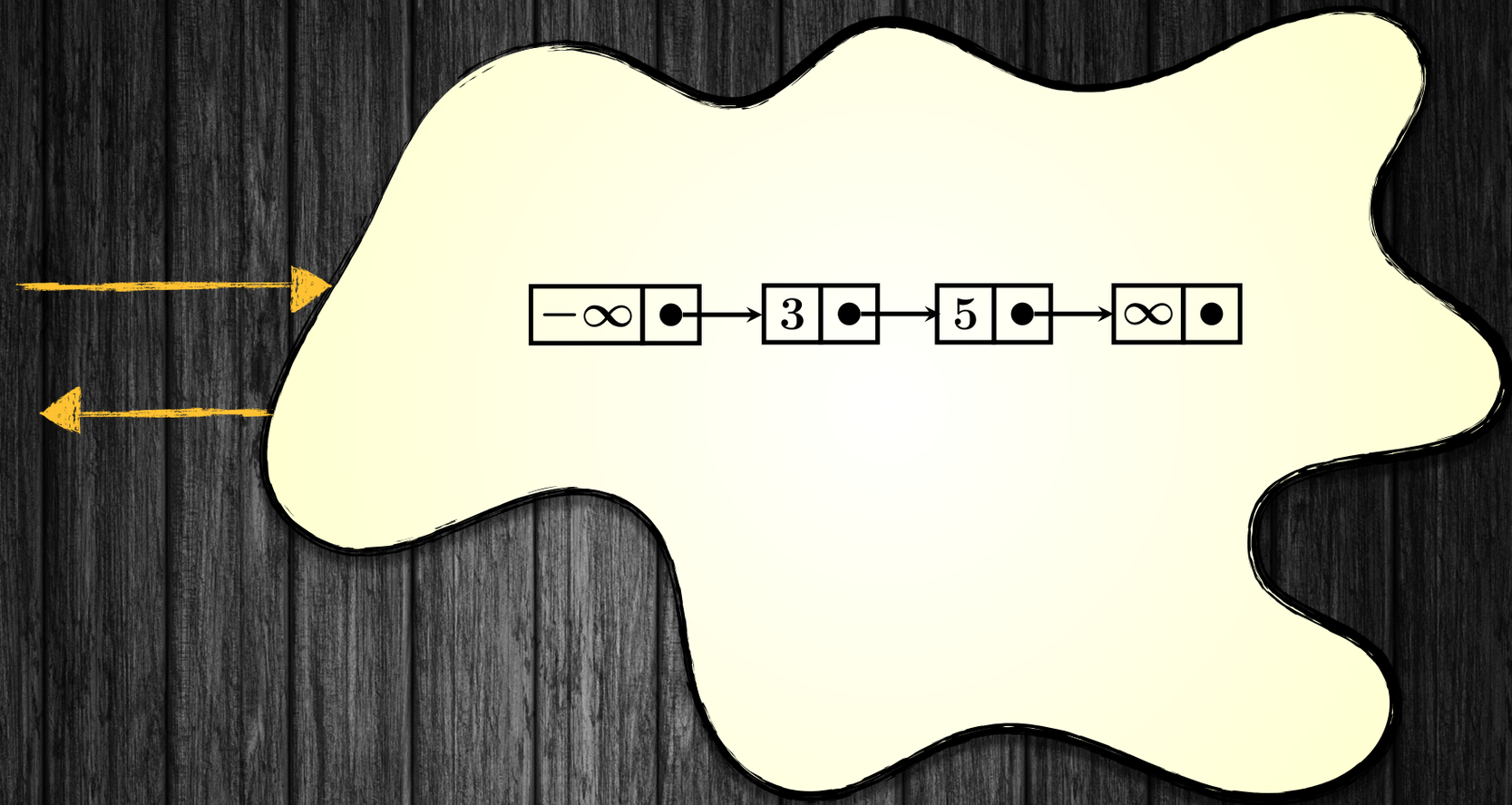


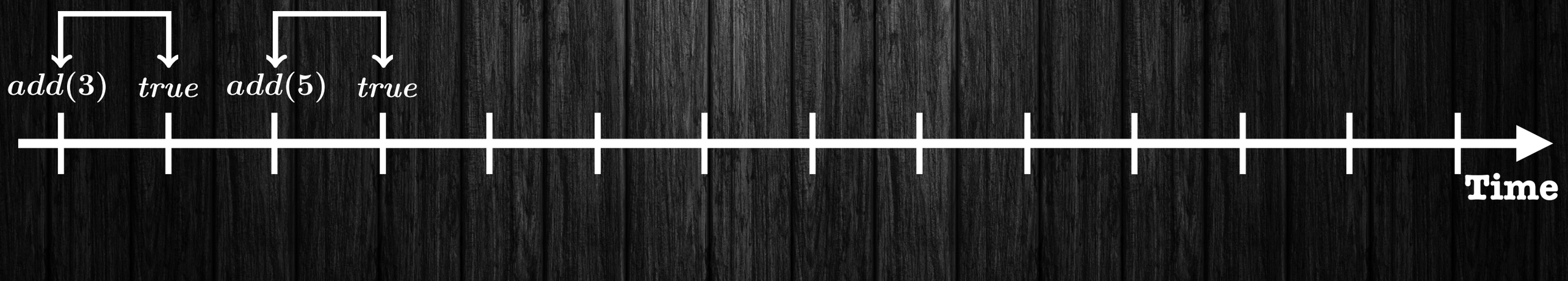
Sets | Sequential Programs



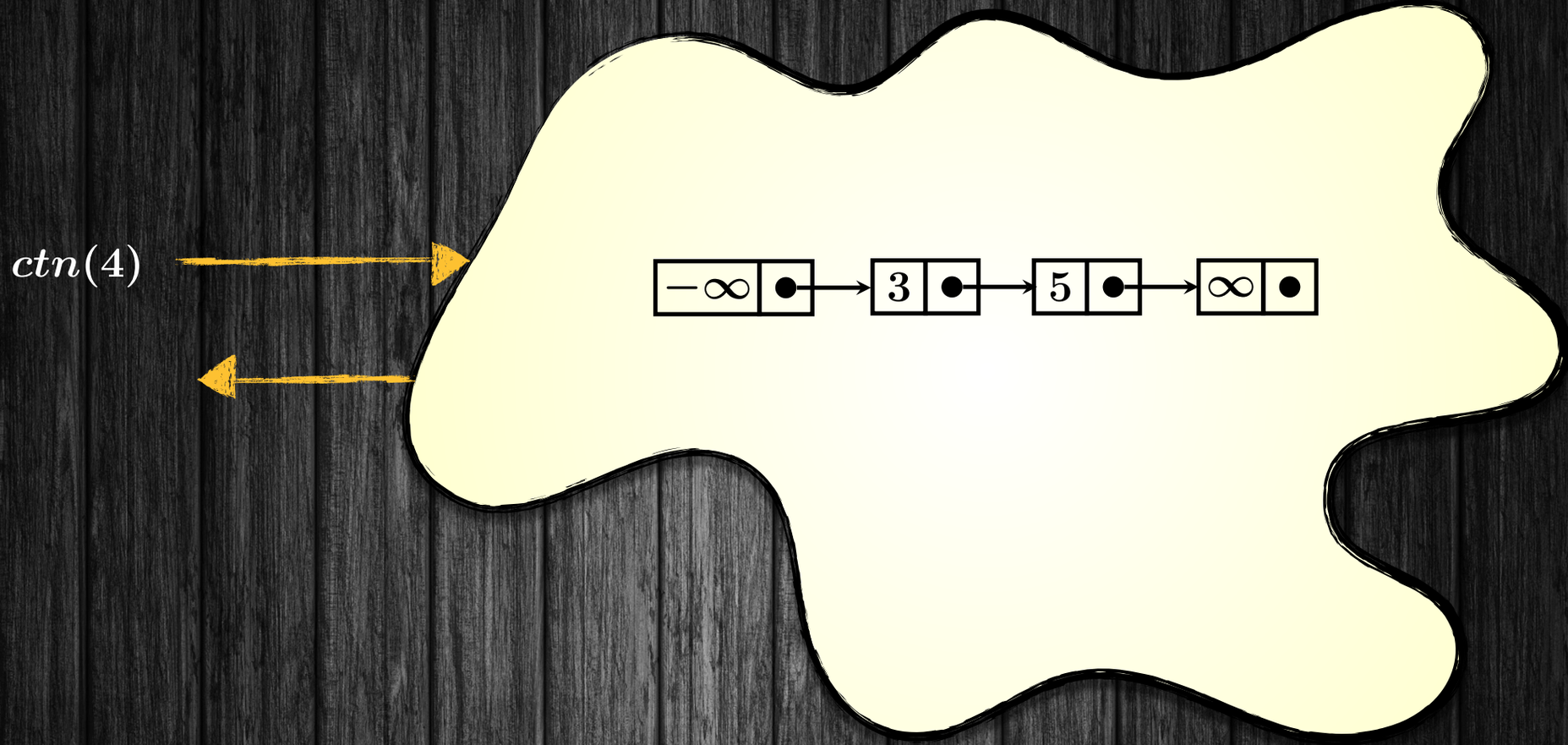


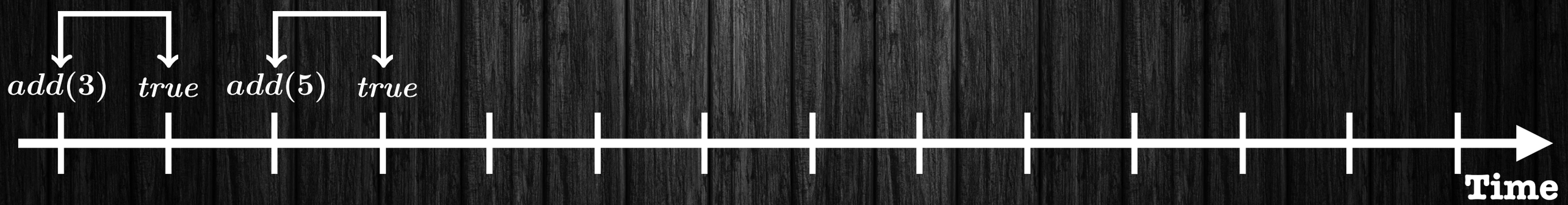
Sets | Sequential Programs



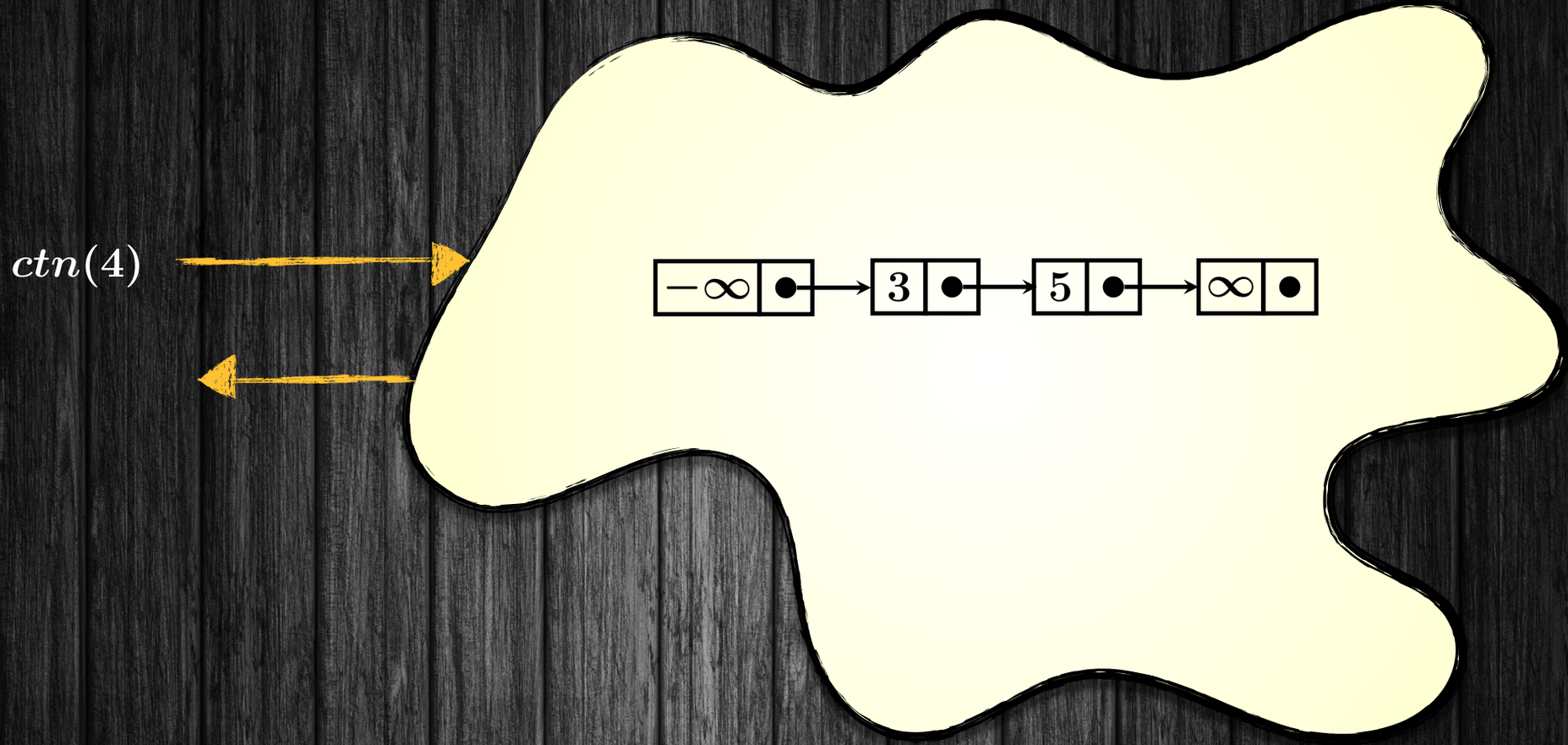


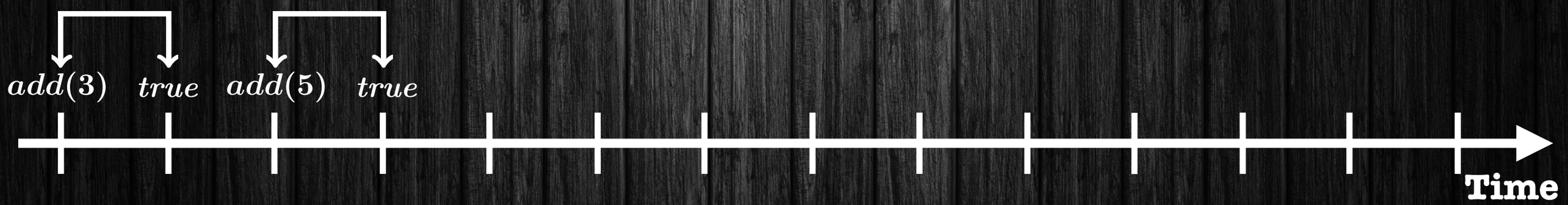
Sets | Sequential Programs



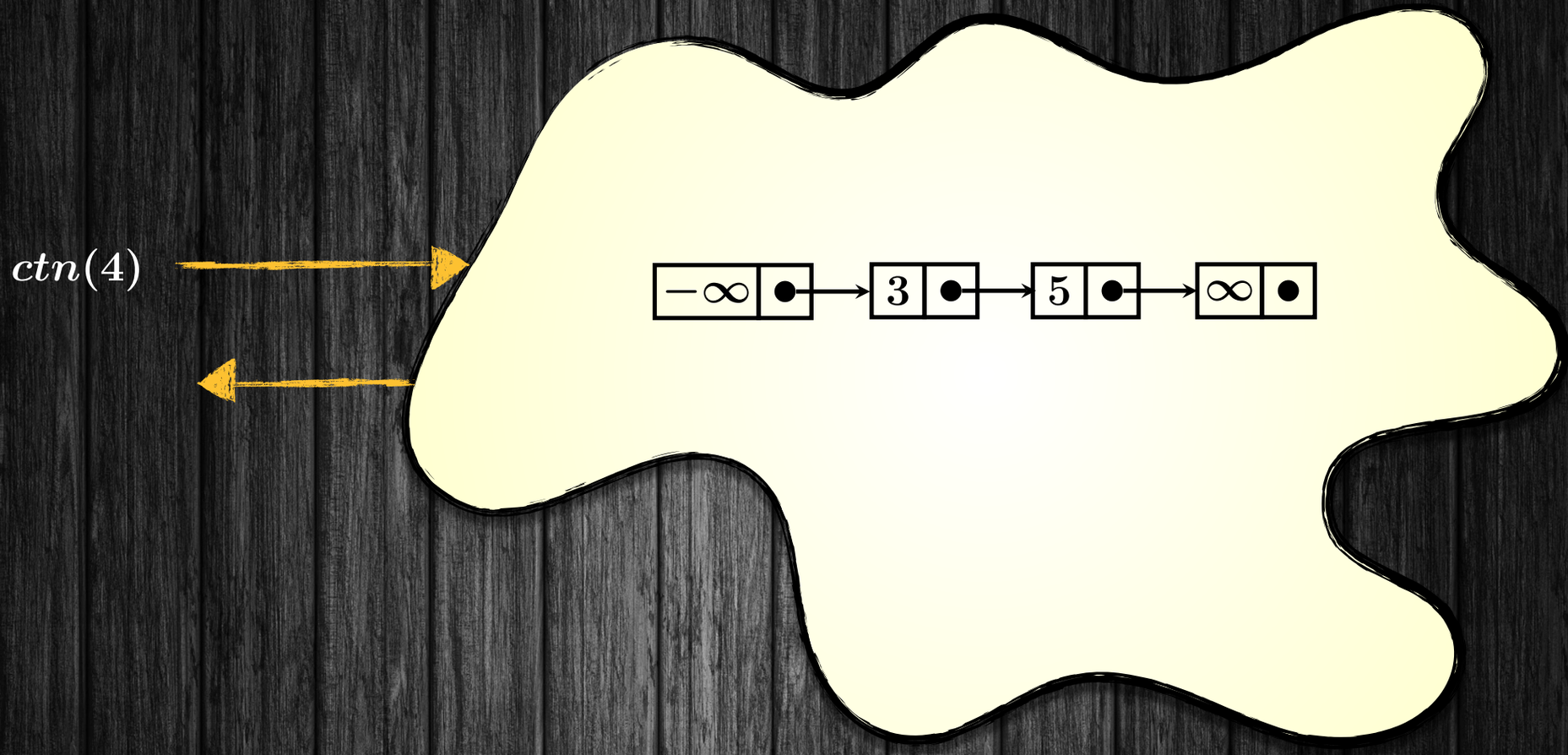


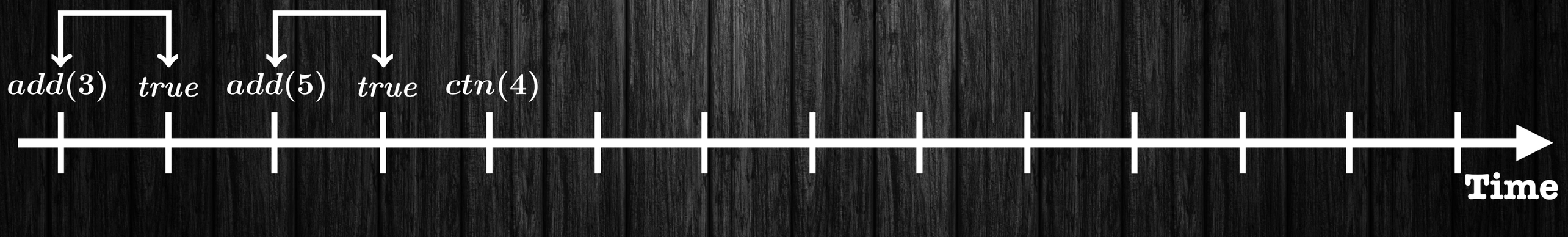
Sets | Sequential Programs



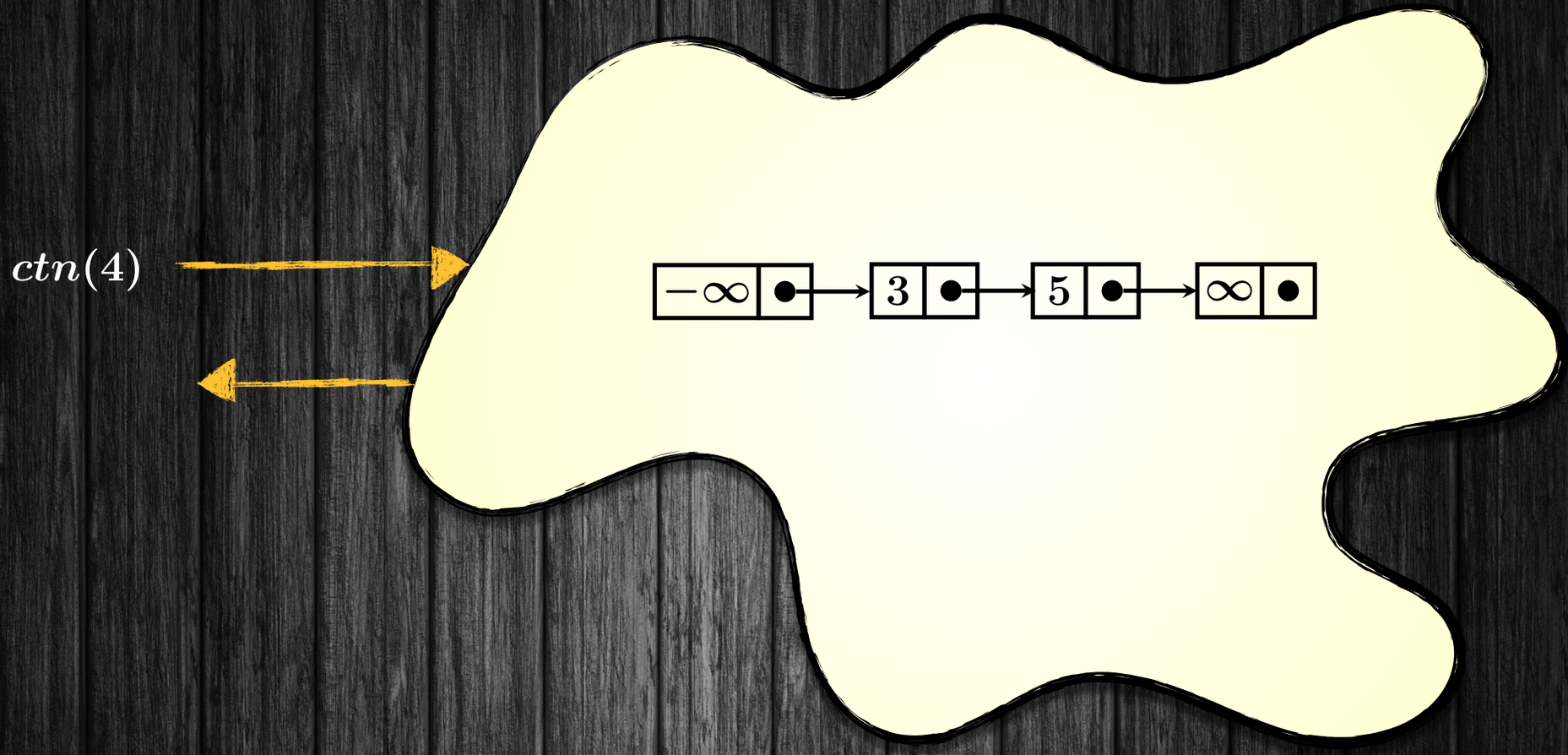


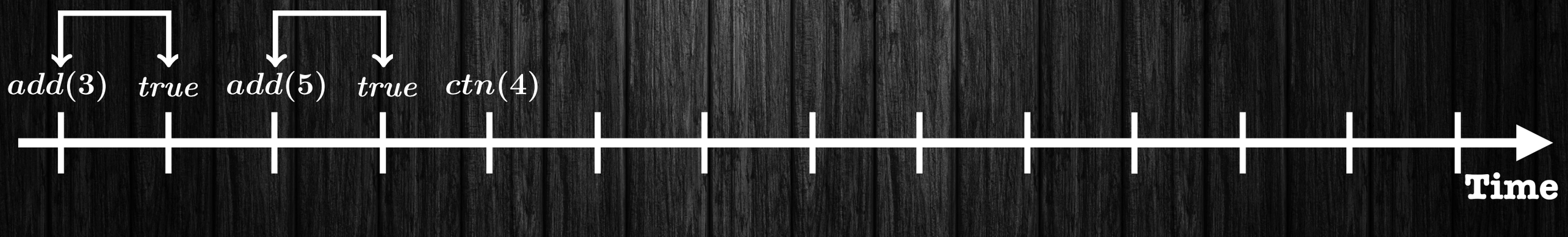
Sets | Sequential Programs



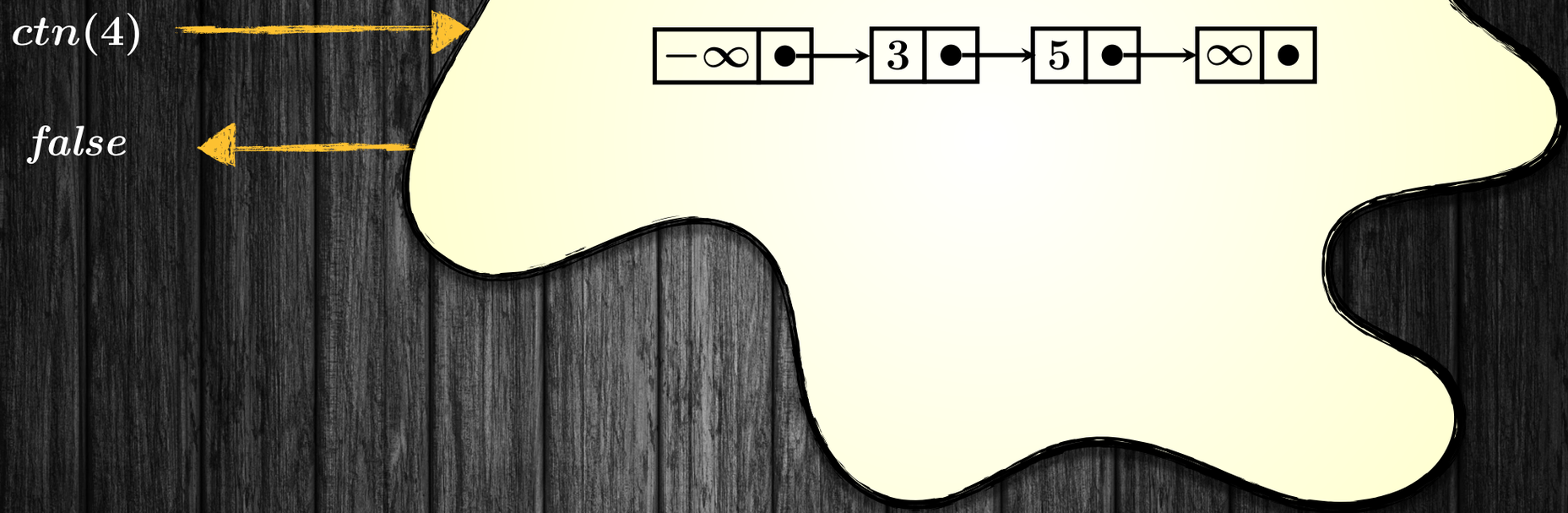


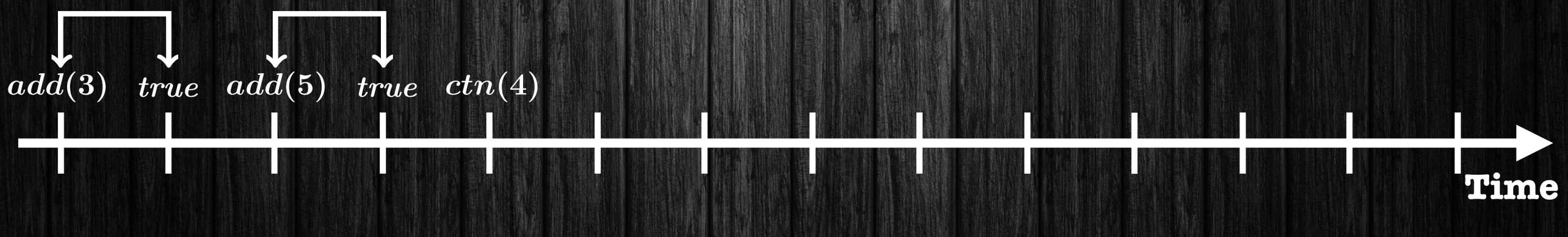
Sets | Sequential Programs



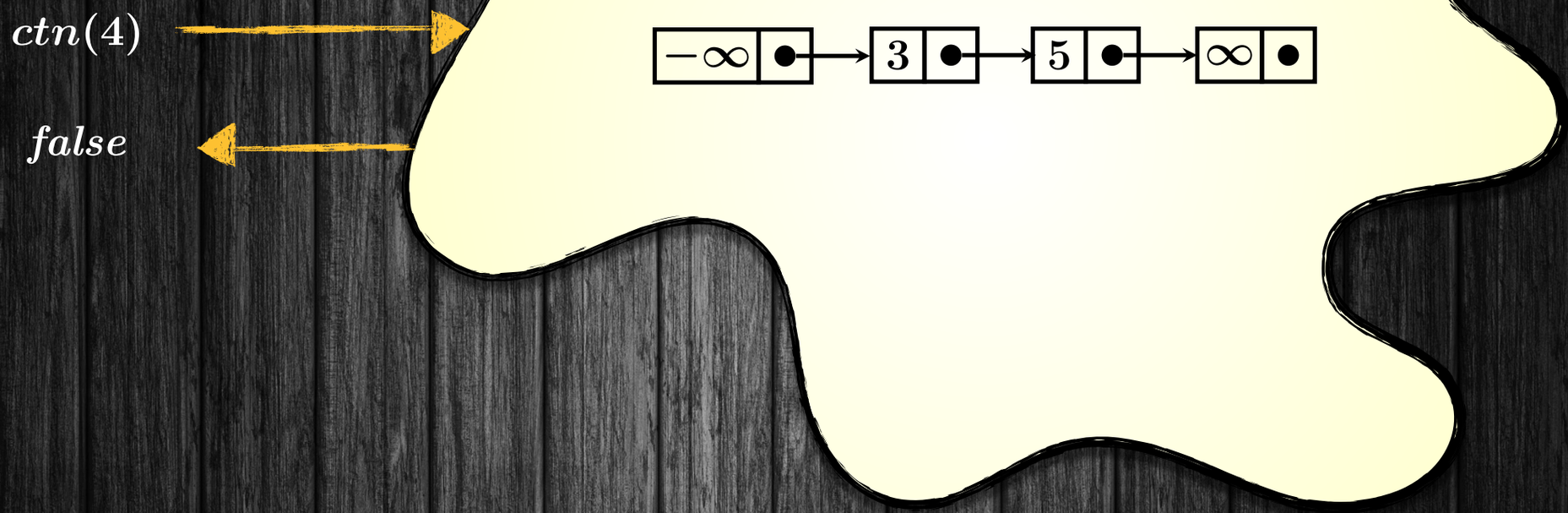


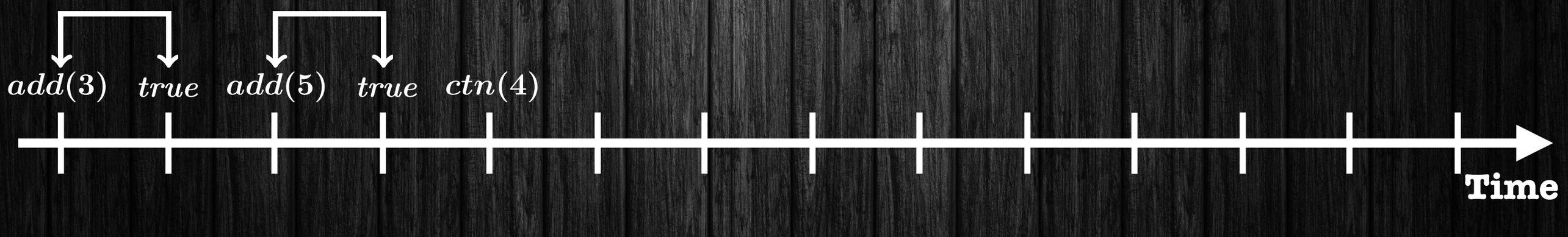
Sets | Sequential Programs



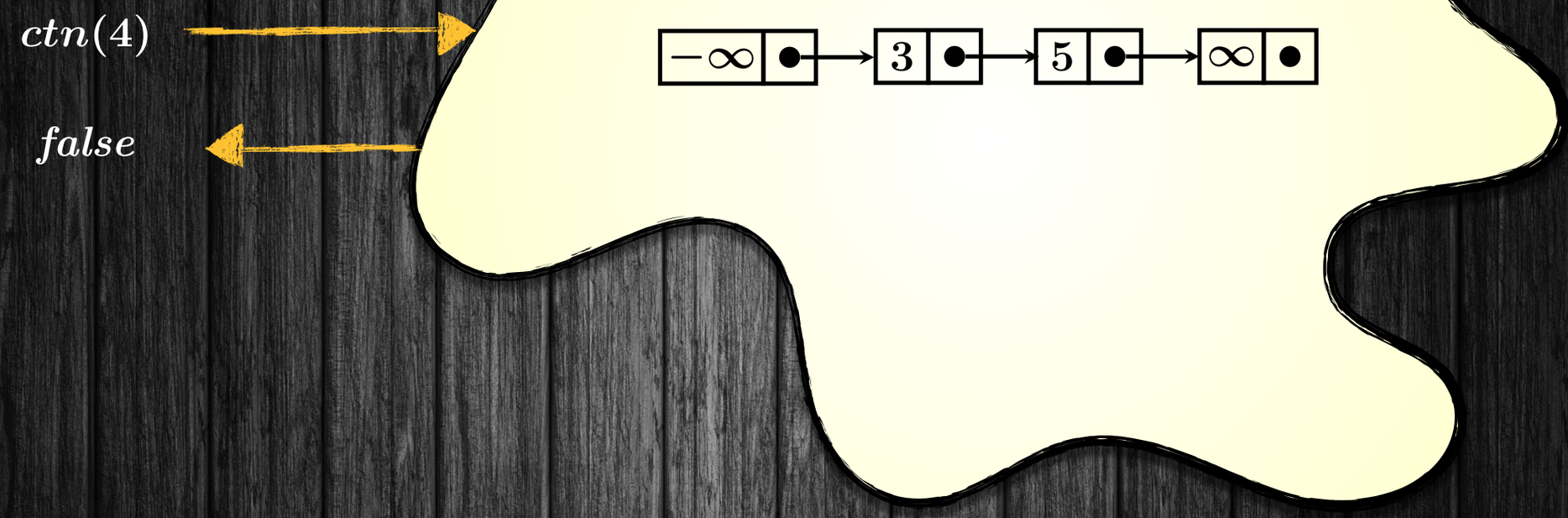


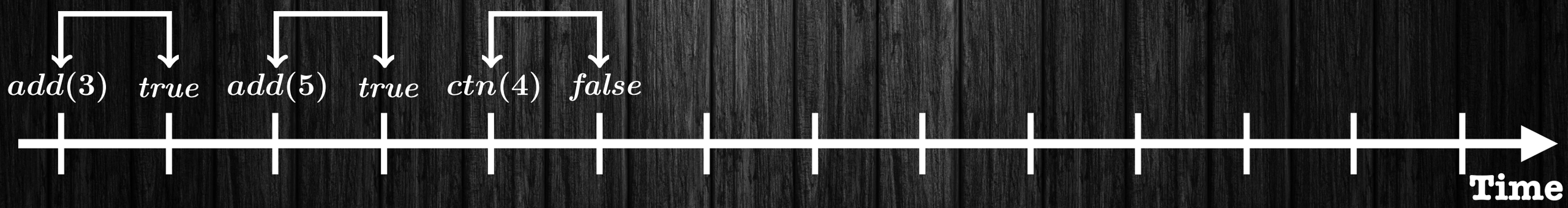
Sets | Sequential Programs



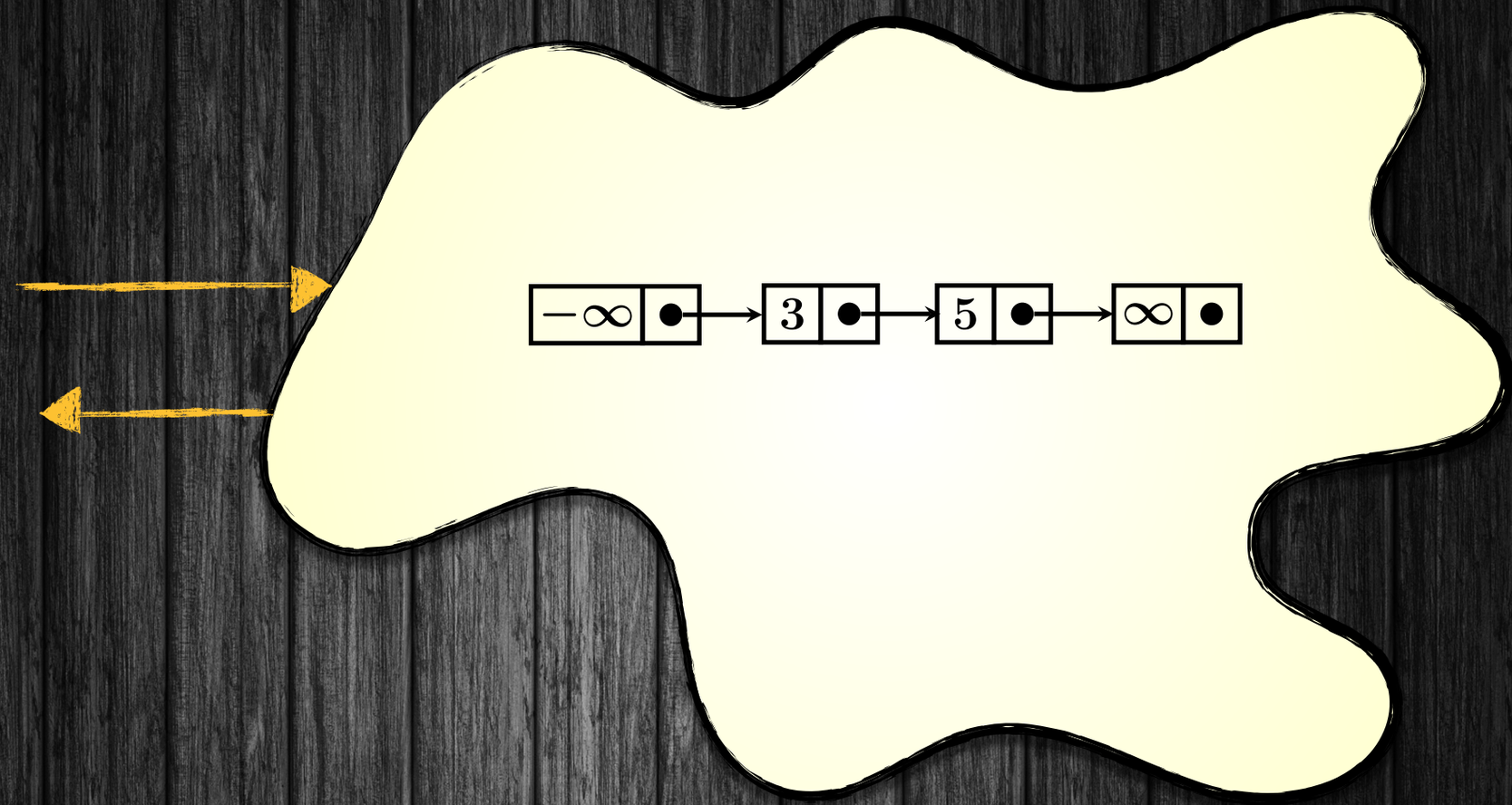


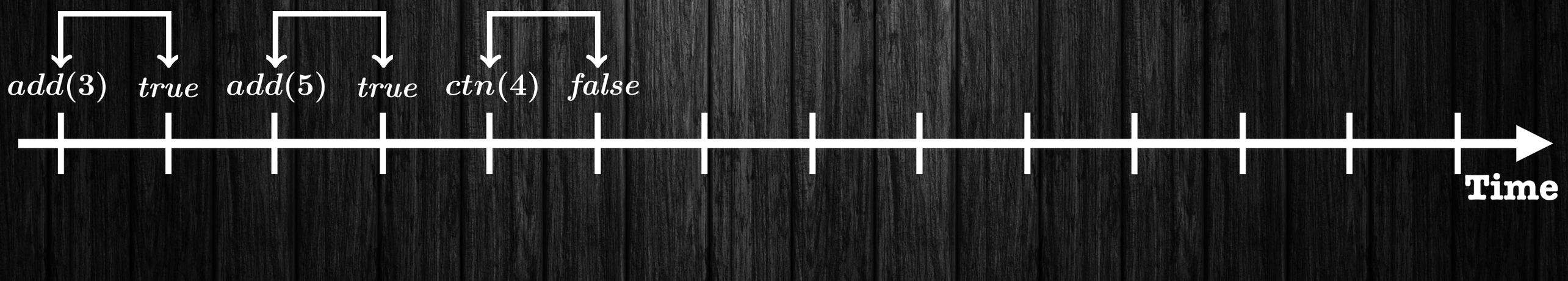
Sets | Sequential Programs



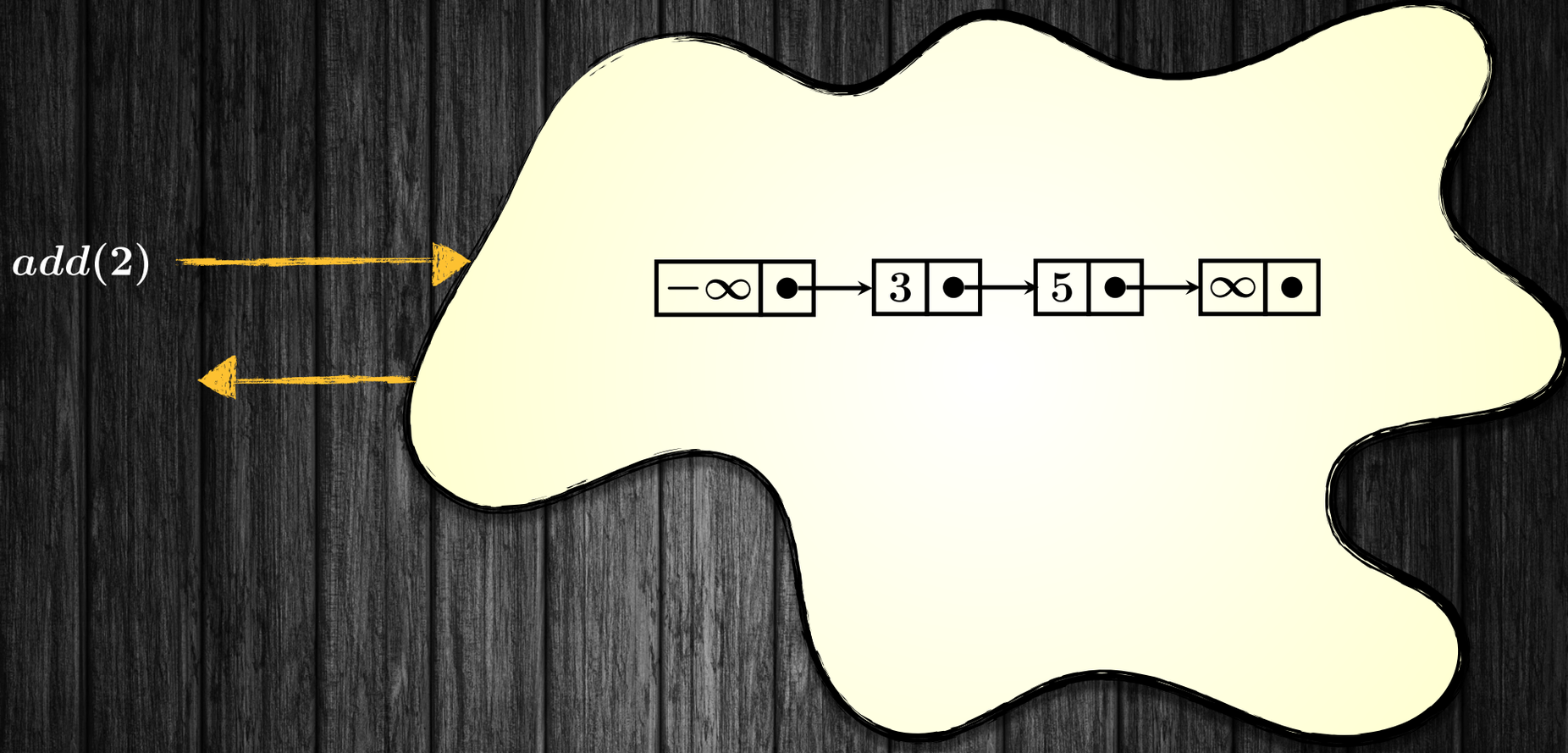


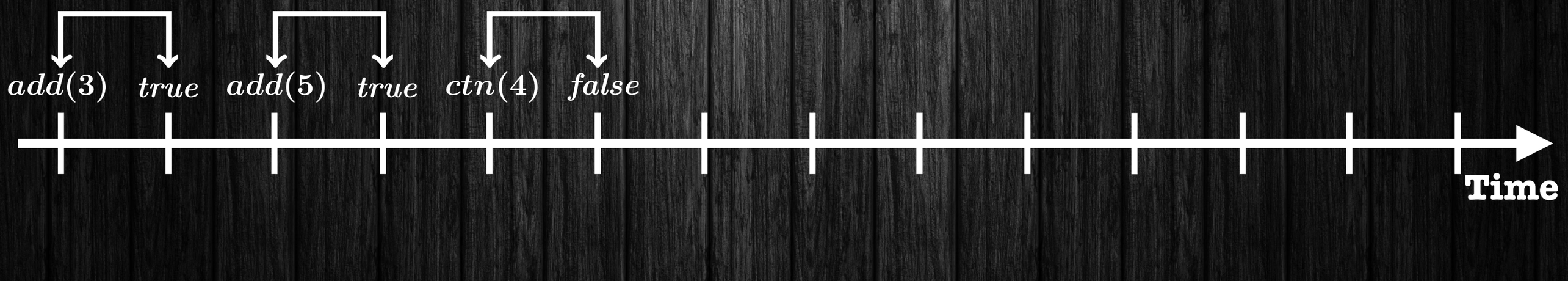
Sets Sequential Programs



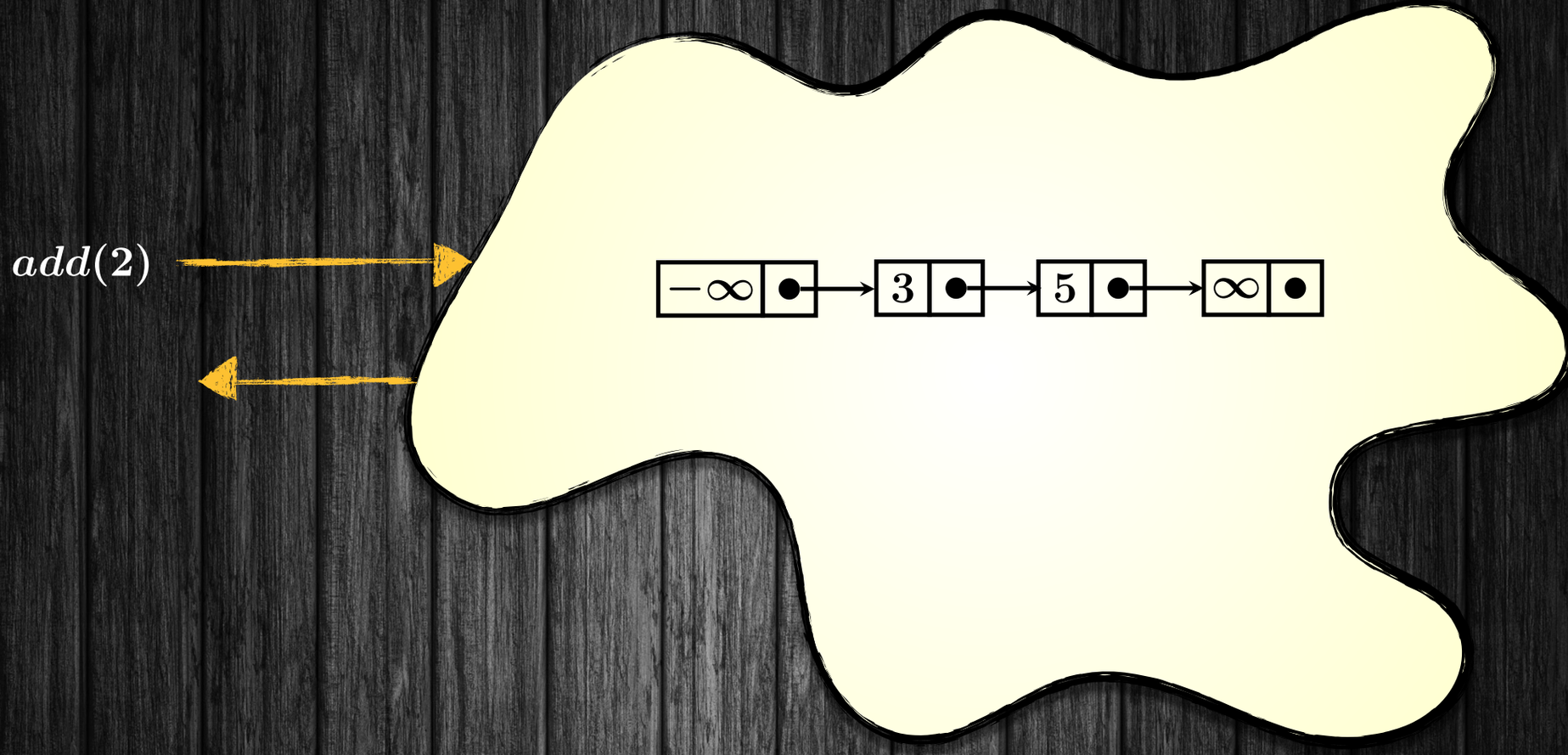


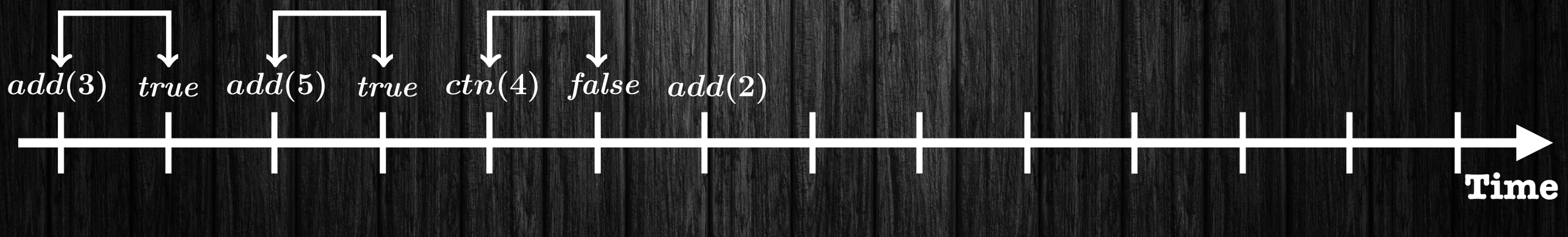
Sets | Sequential Programs



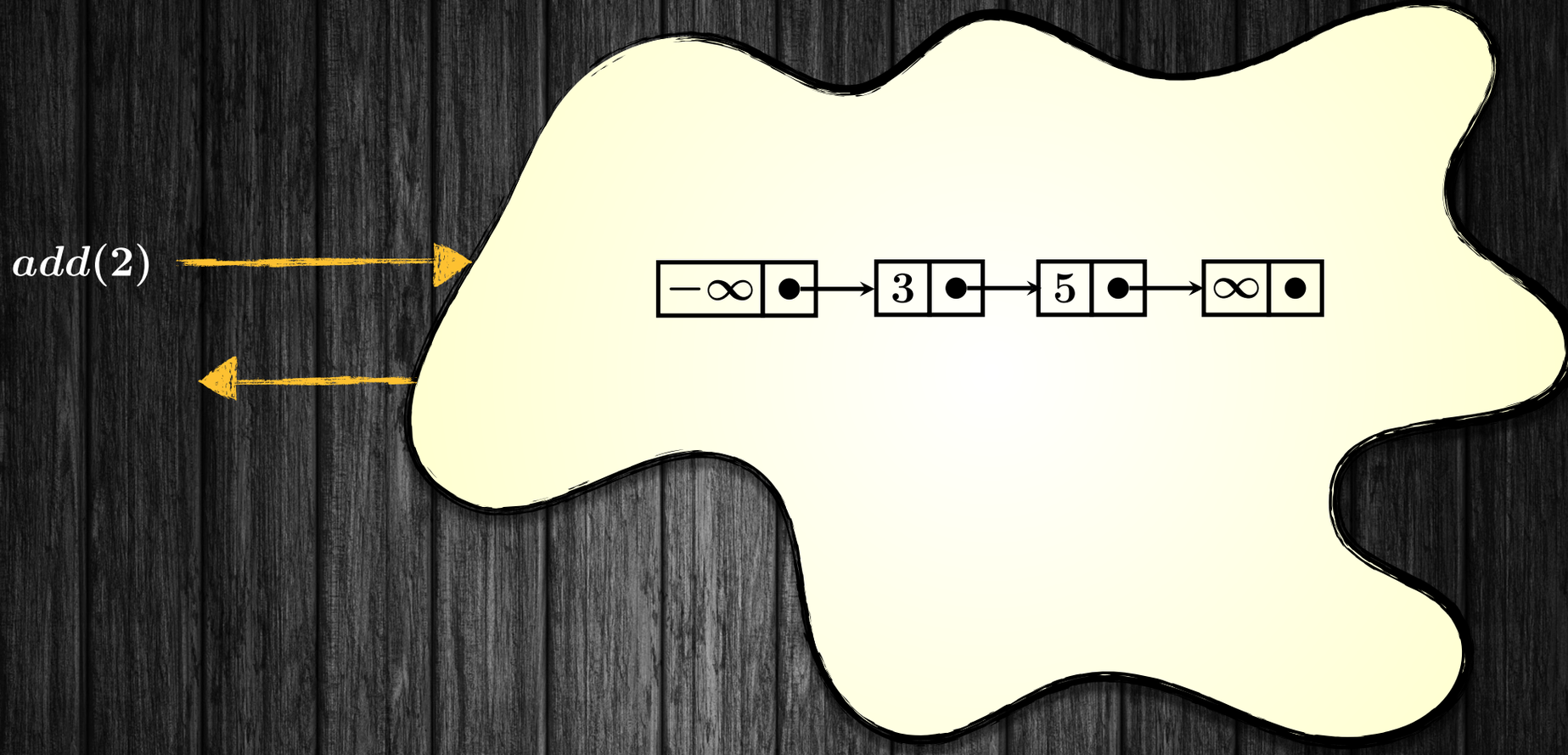


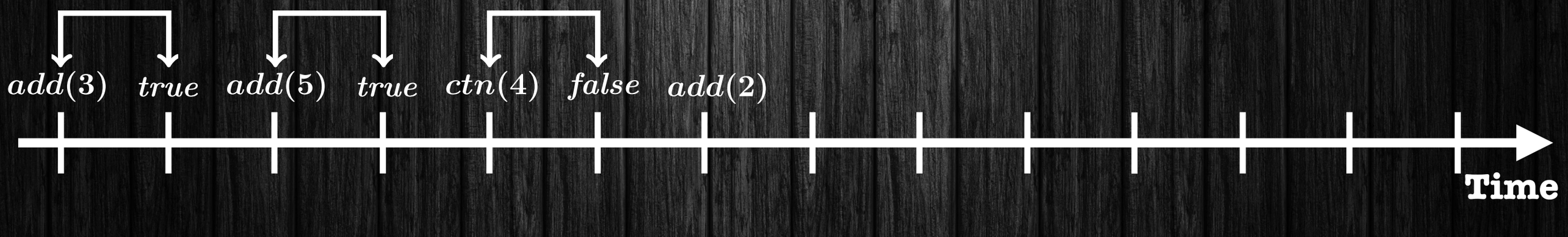
Sets | Sequential Programs



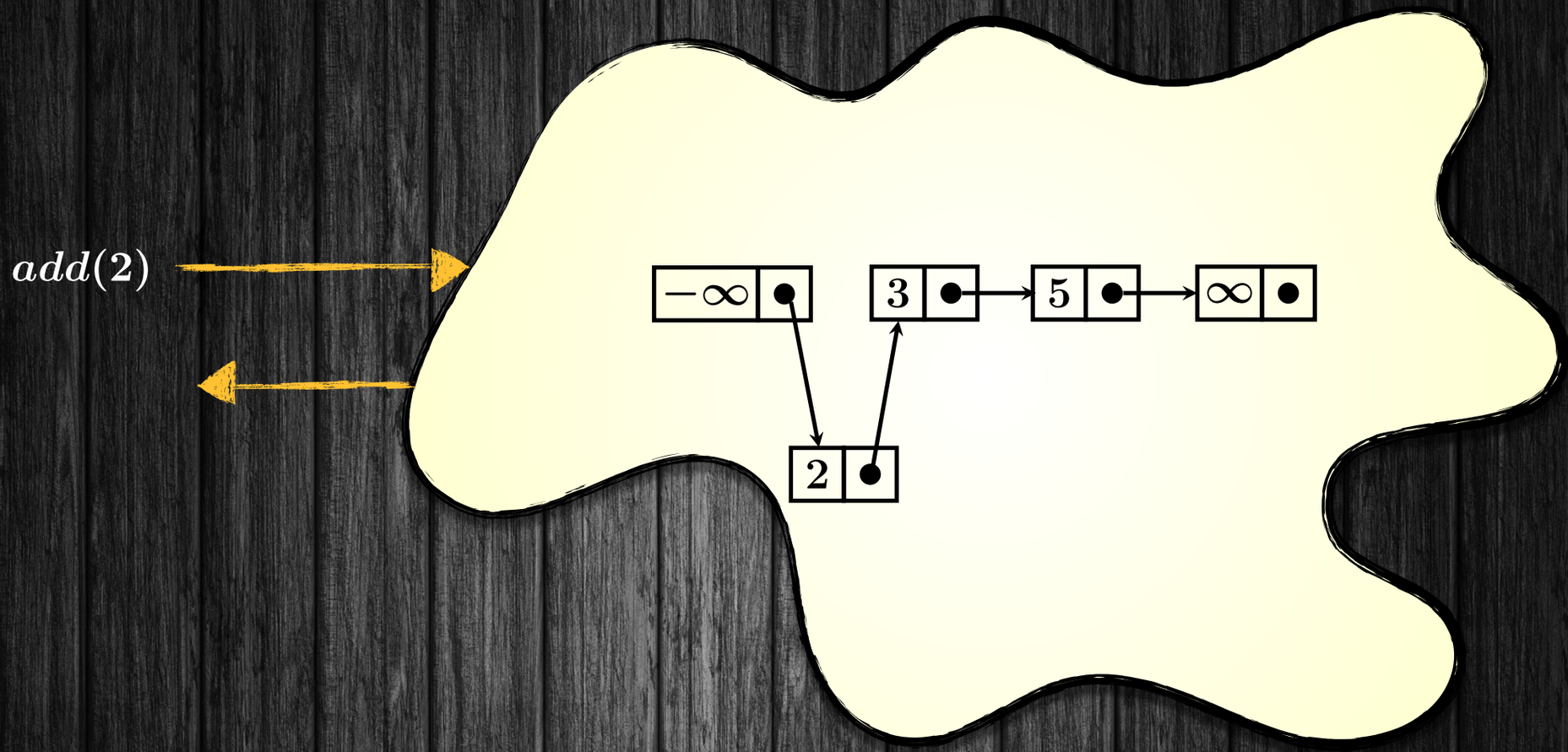


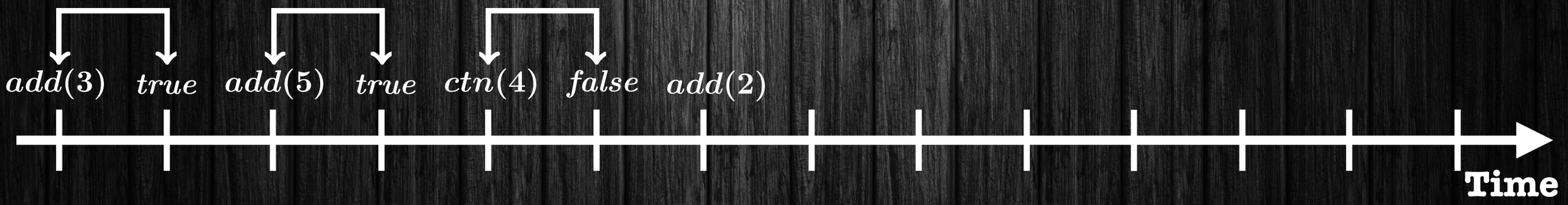
Sets | Sequential Programs



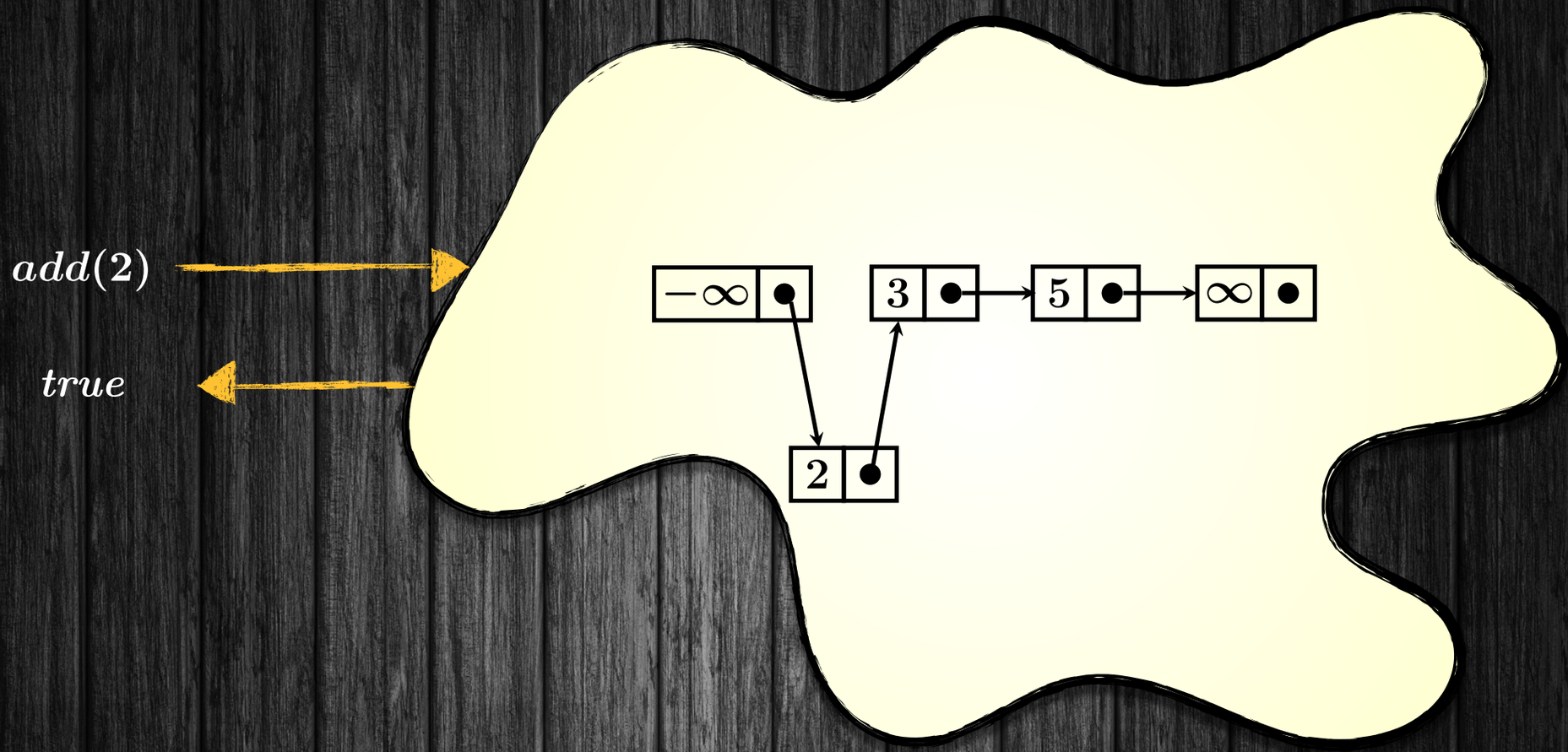


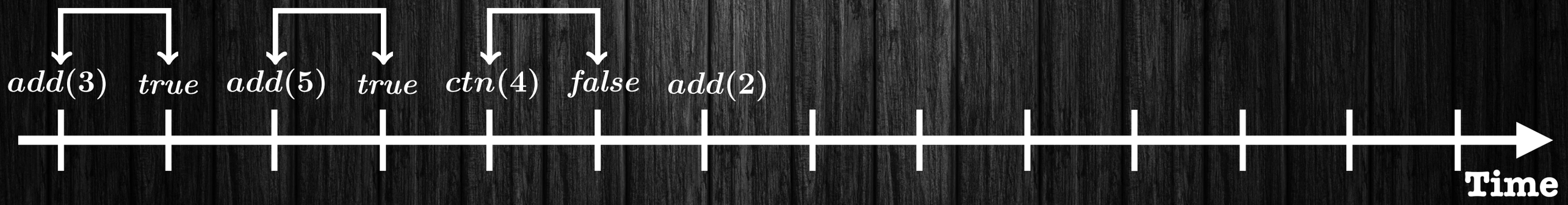
Sets | Sequential Programs



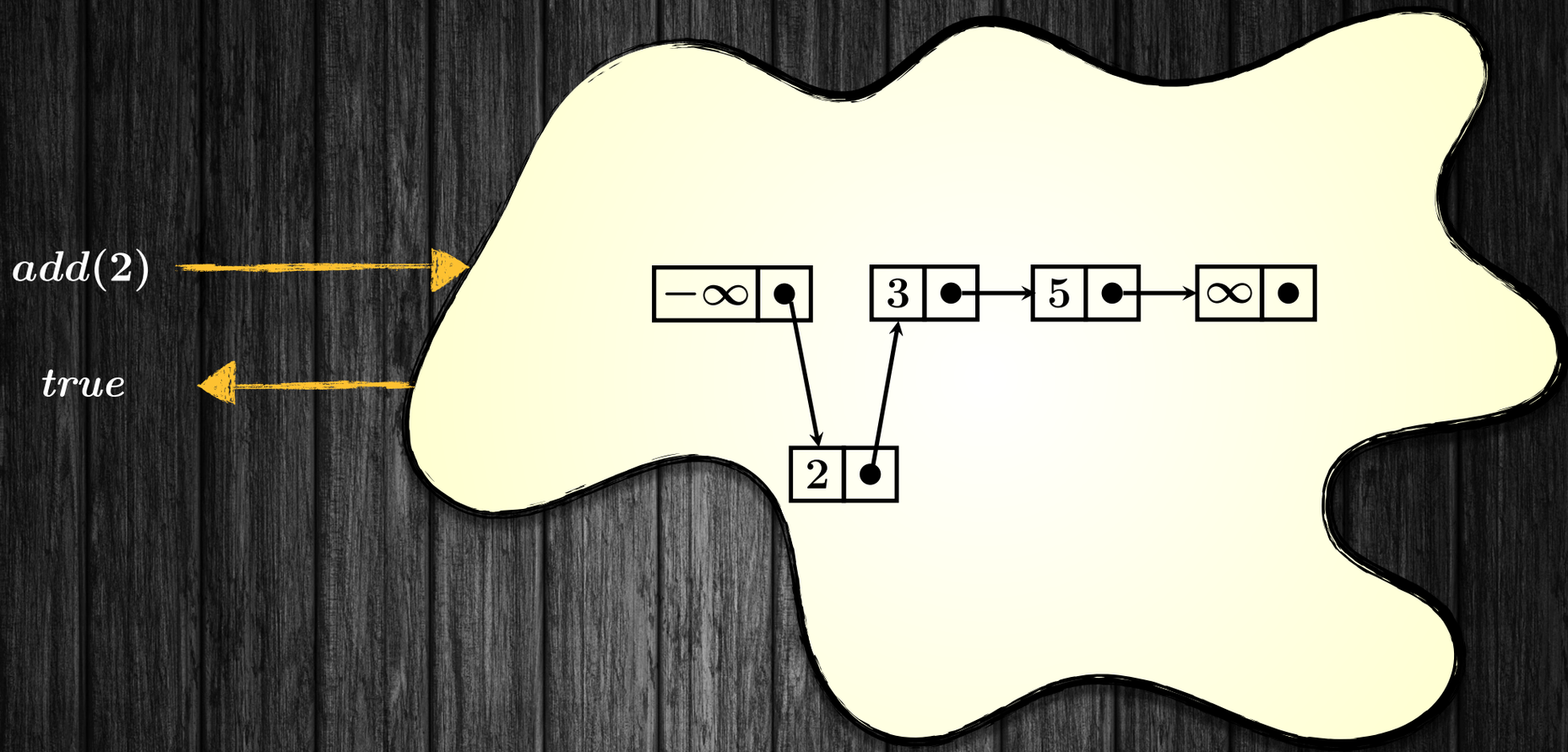


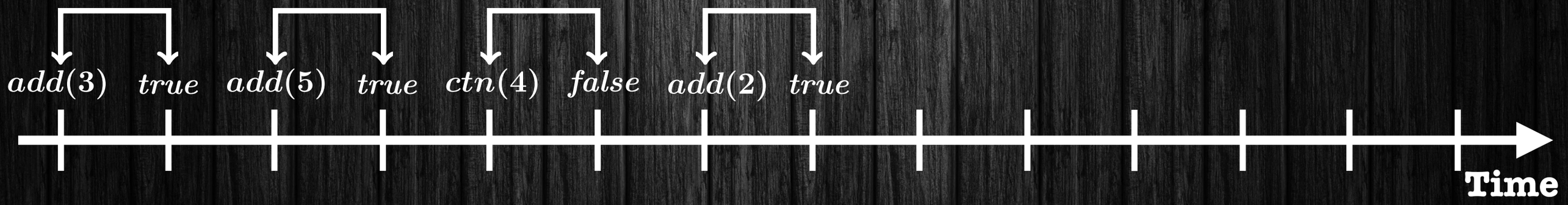
Sets | Sequential Programs



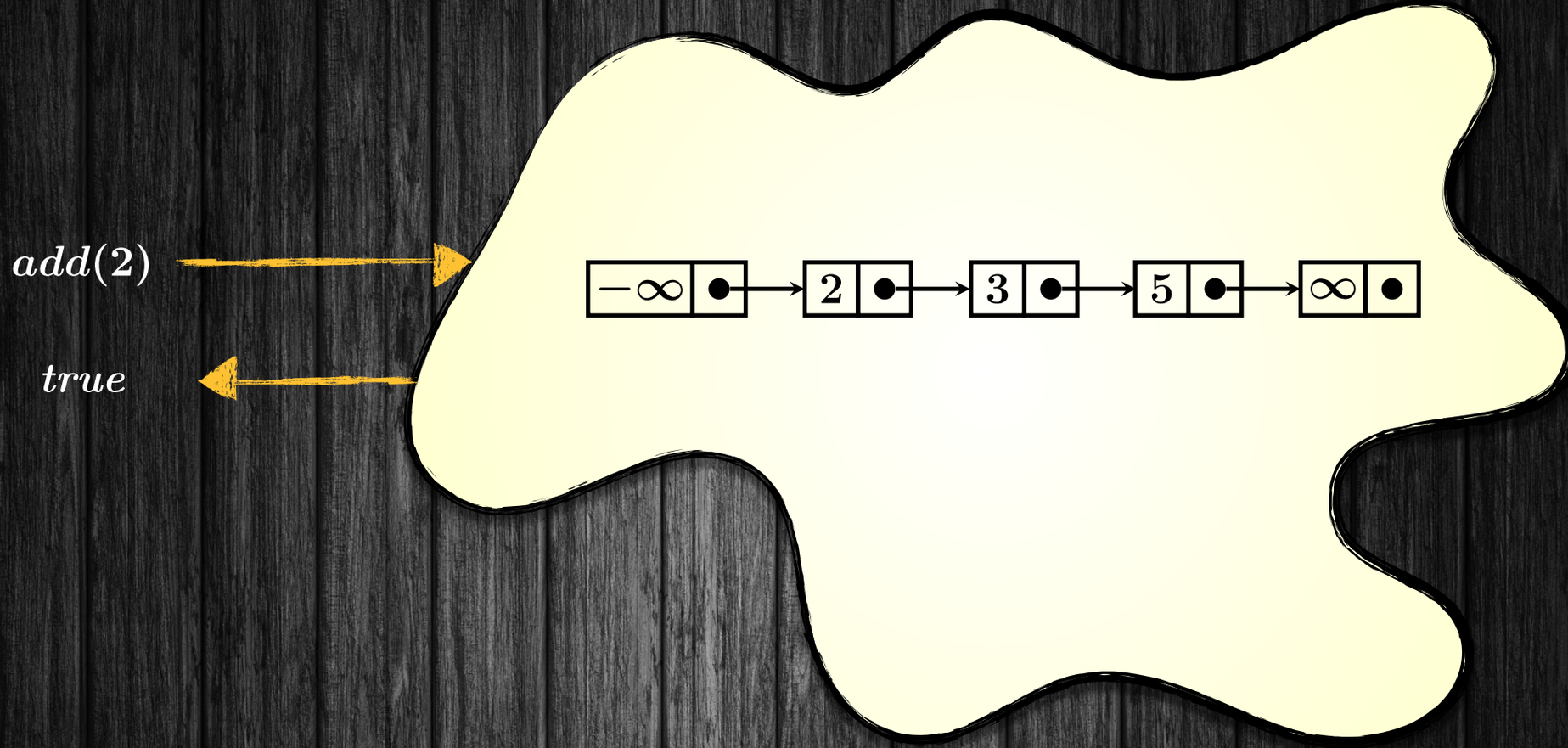


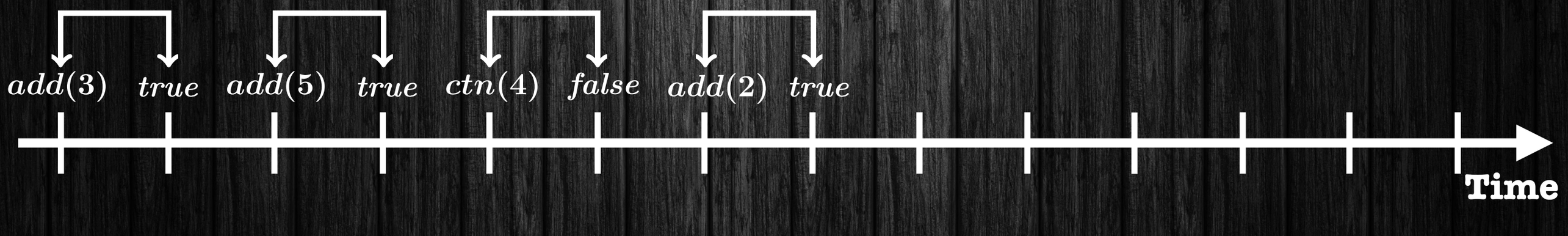
Sets | Sequential Programs



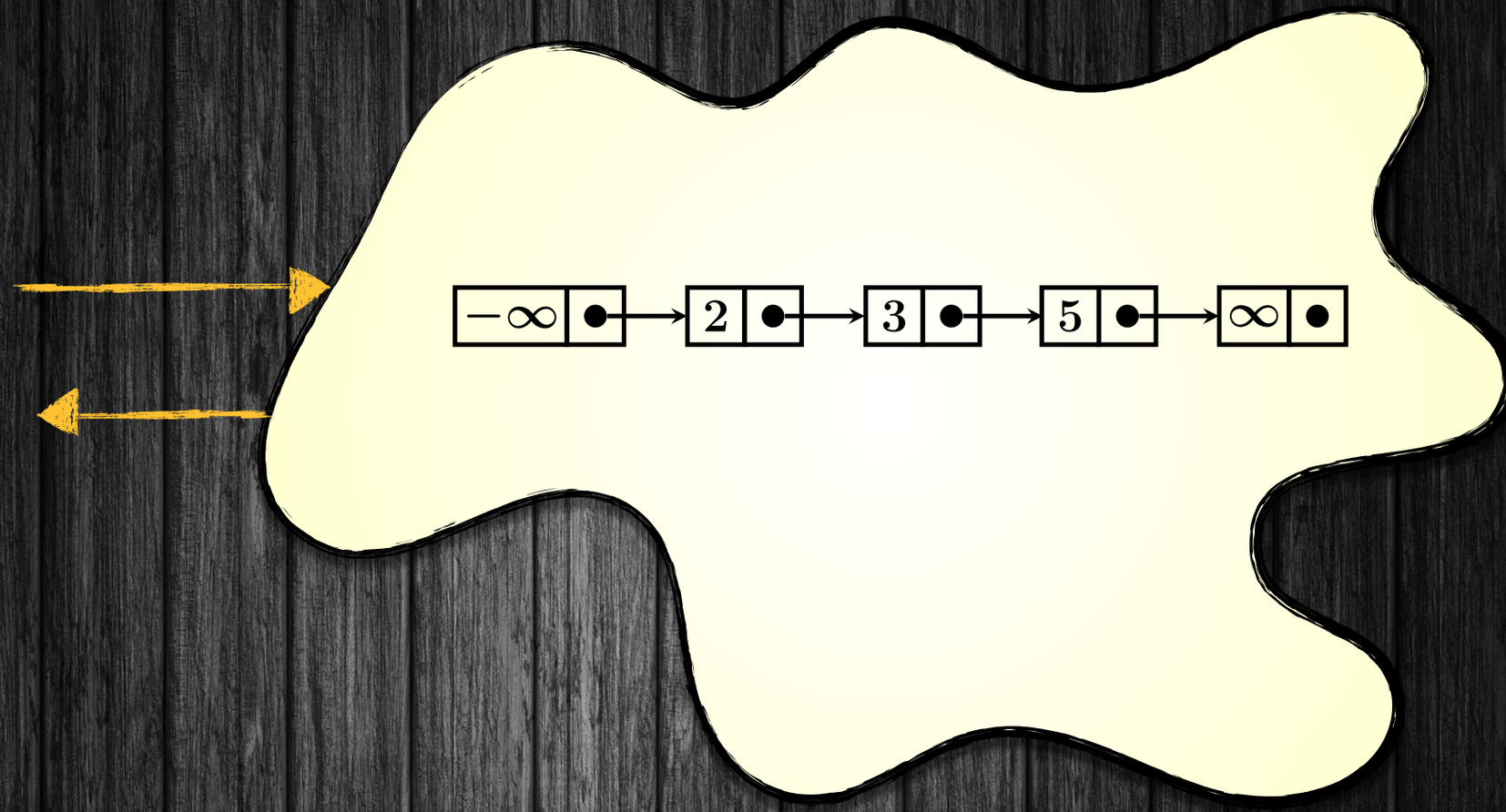


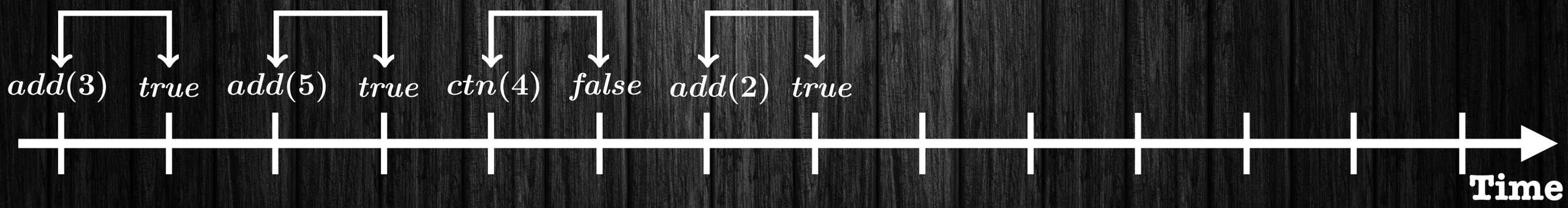
Sets | Sequential Programs



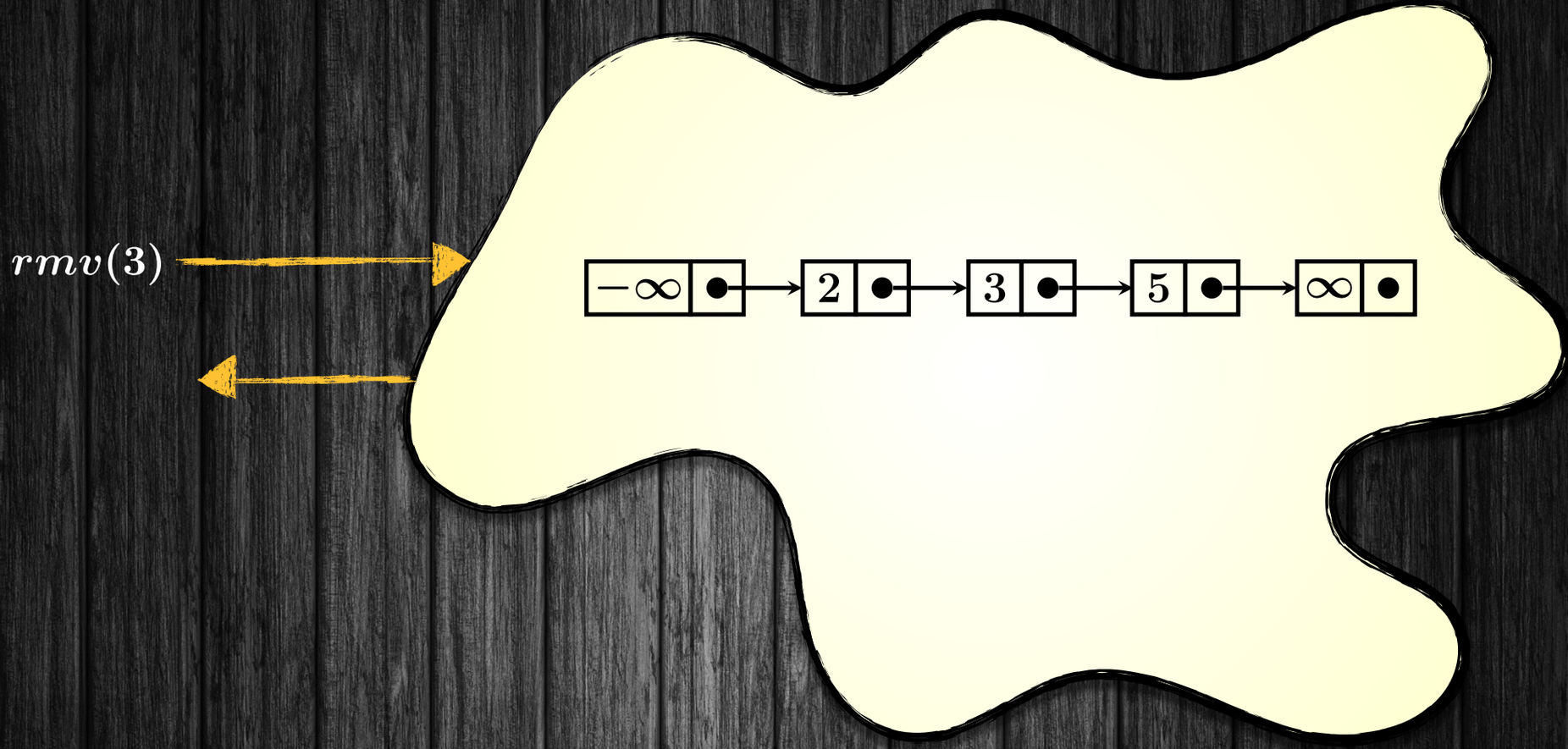


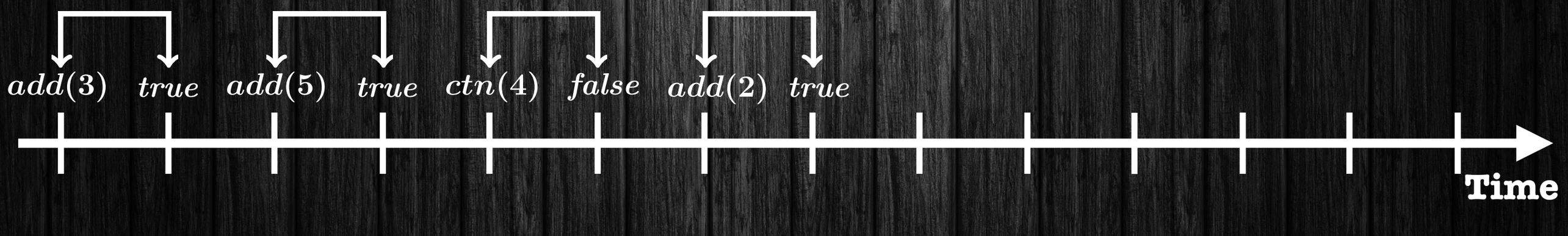
Sets | Sequential Programs



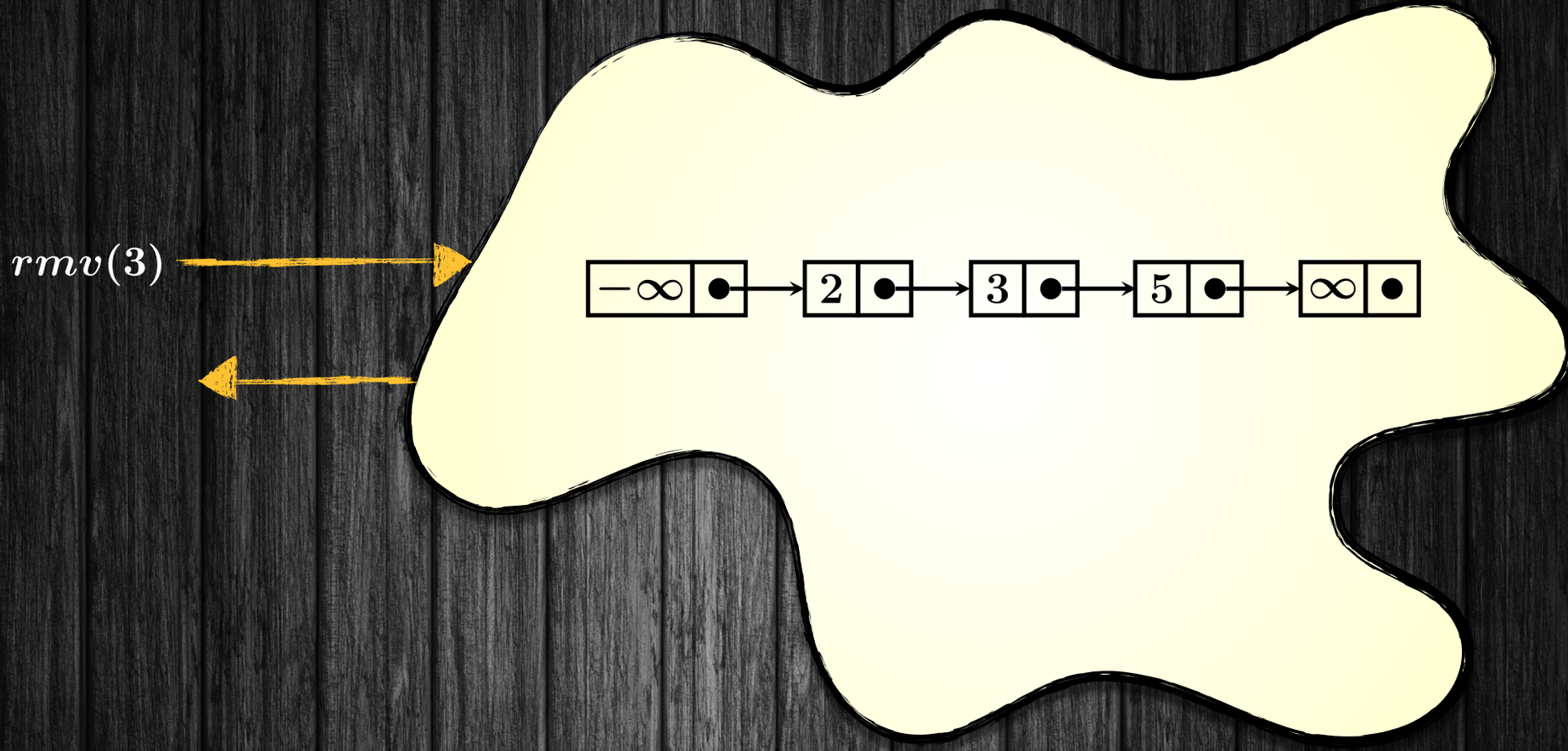


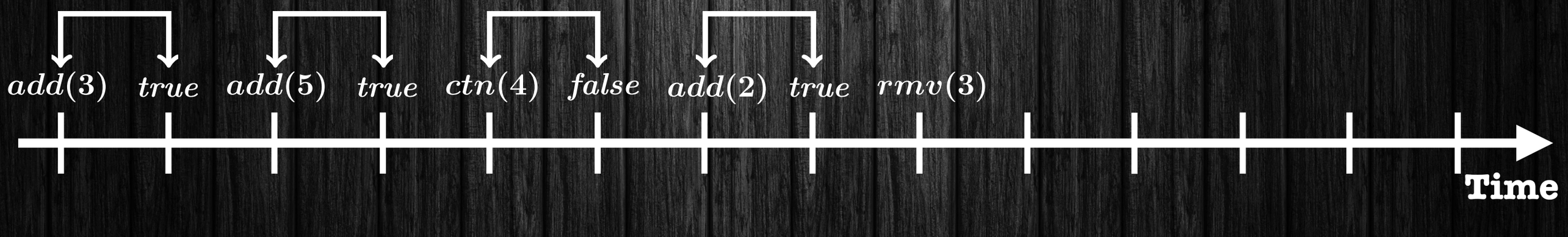
Sets | Sequential Programs



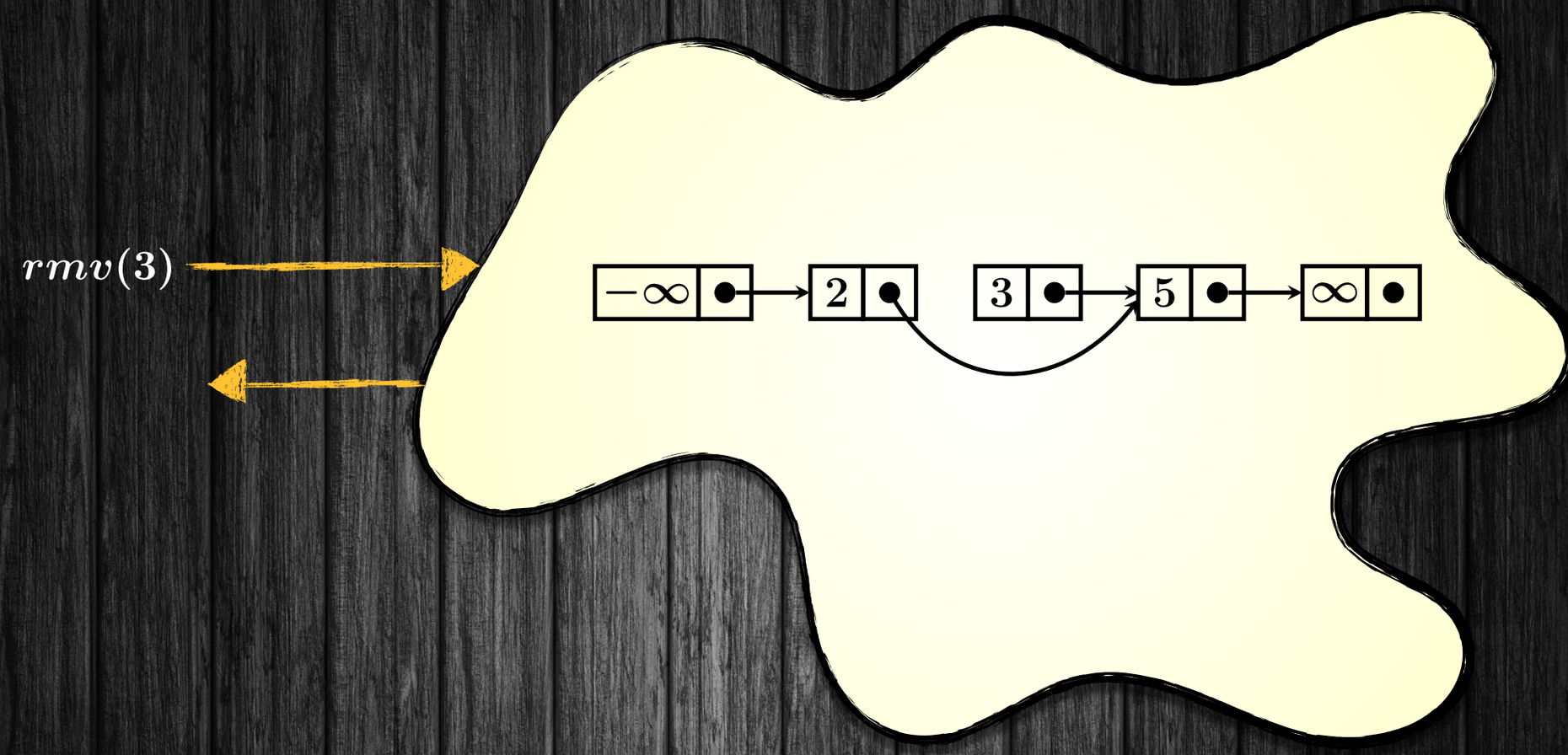


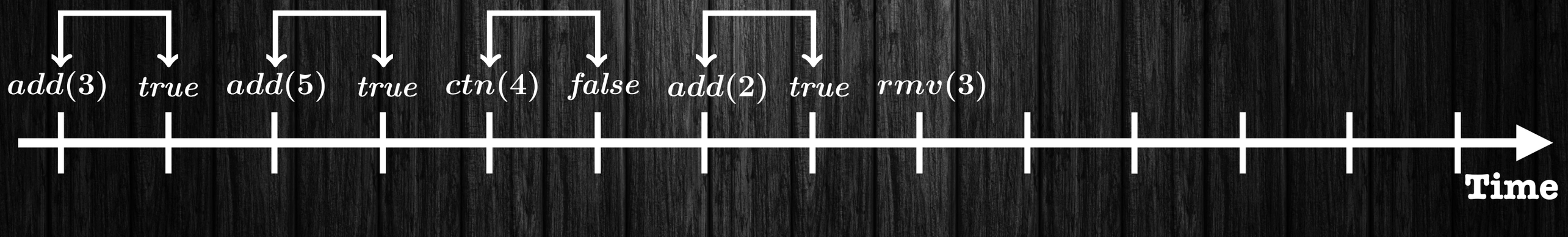
Sets | Sequential Programs



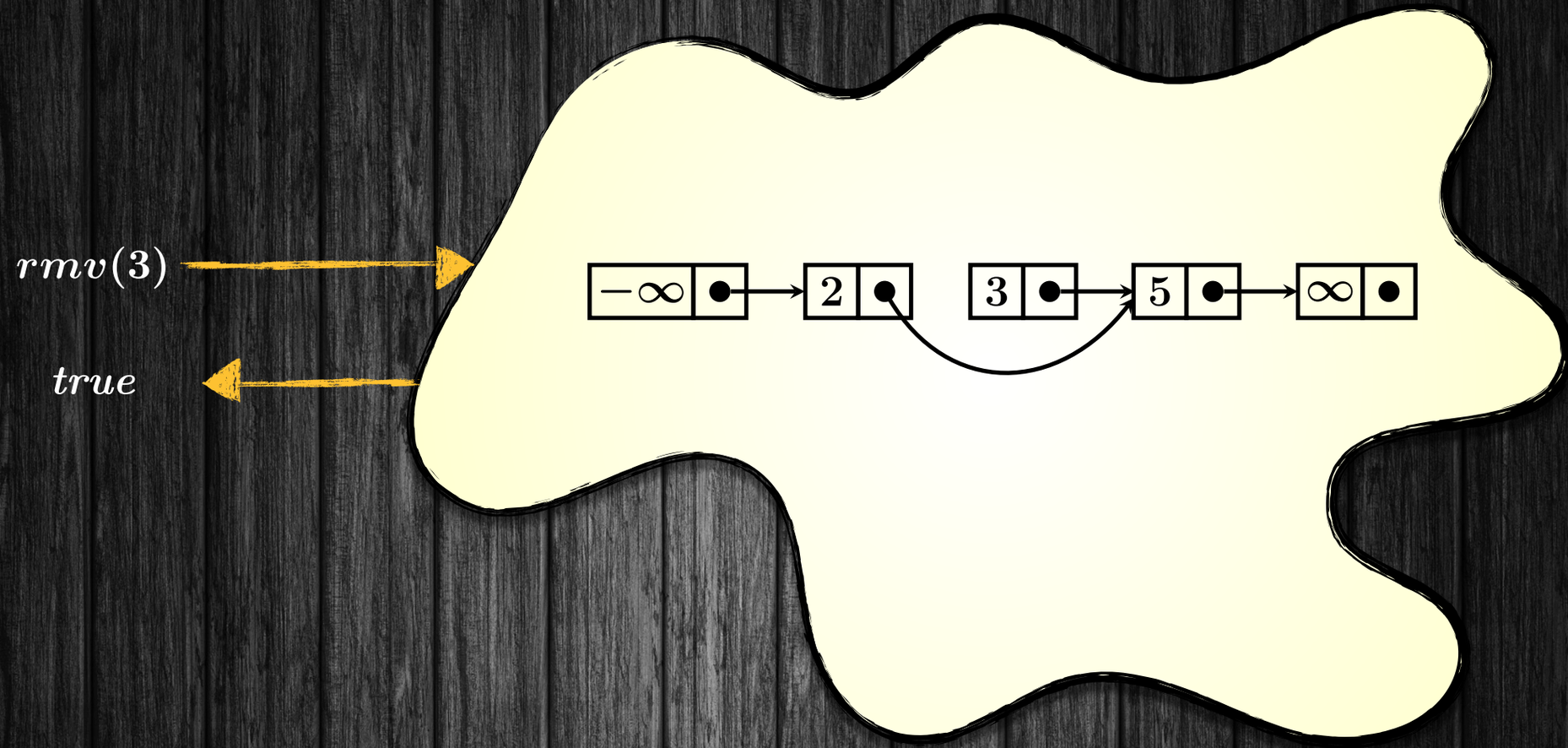


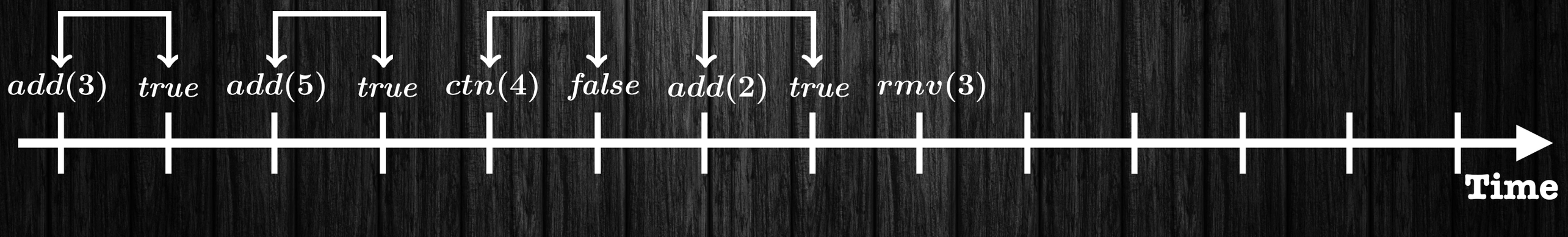
Sets | Sequential Programs



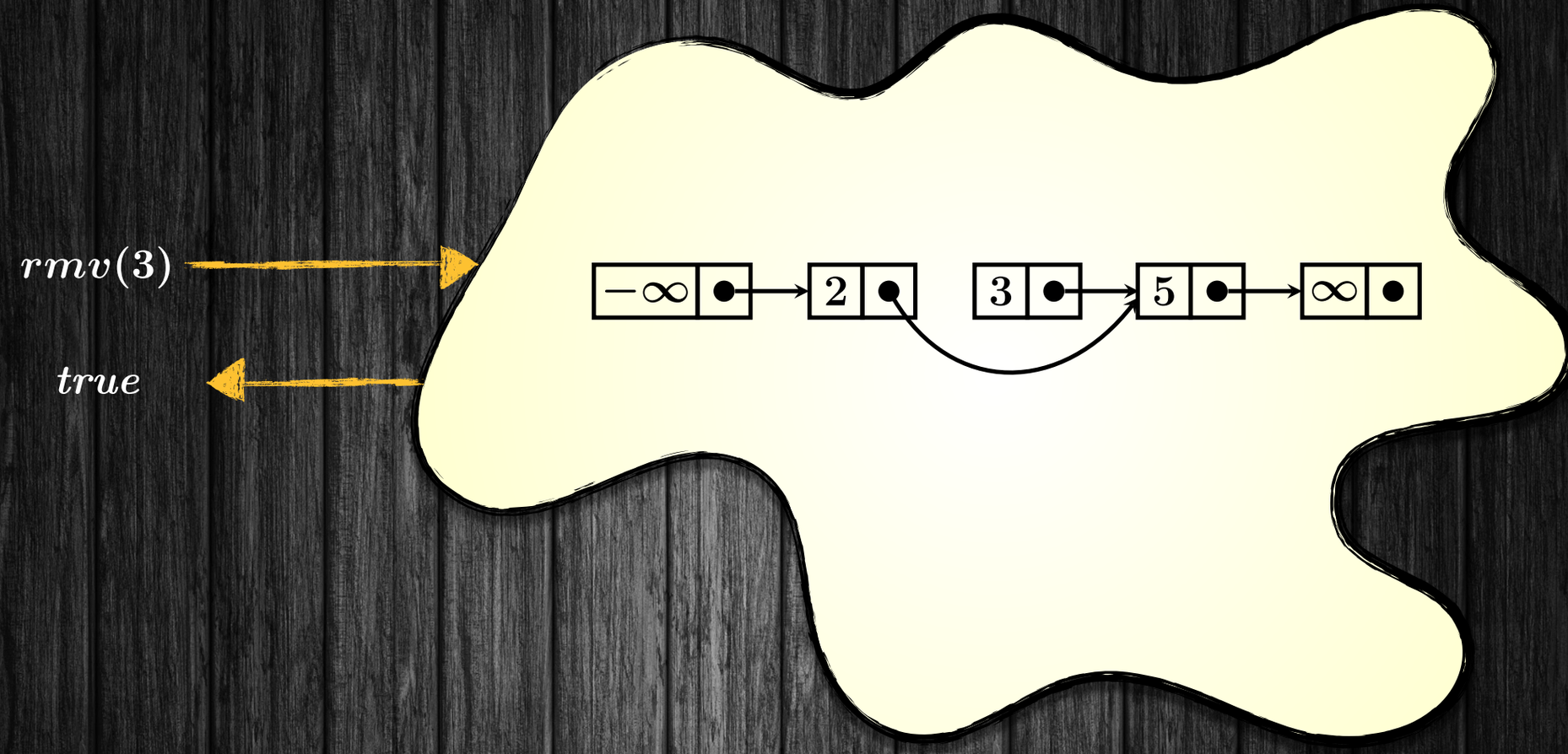


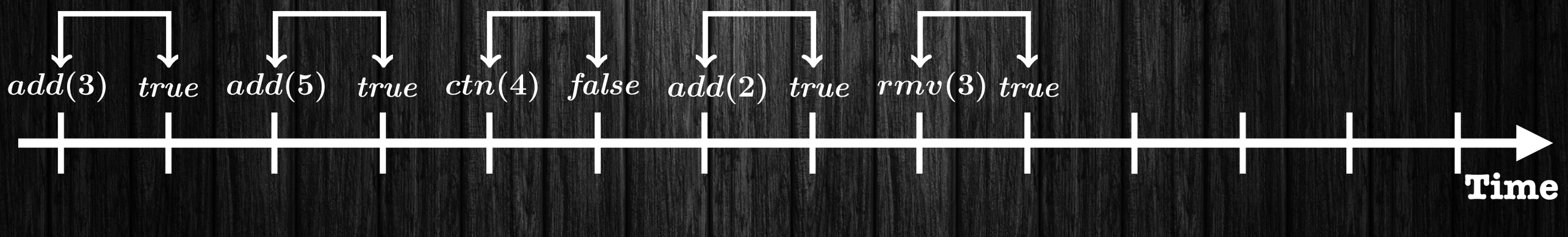
Sets | Sequential Programs



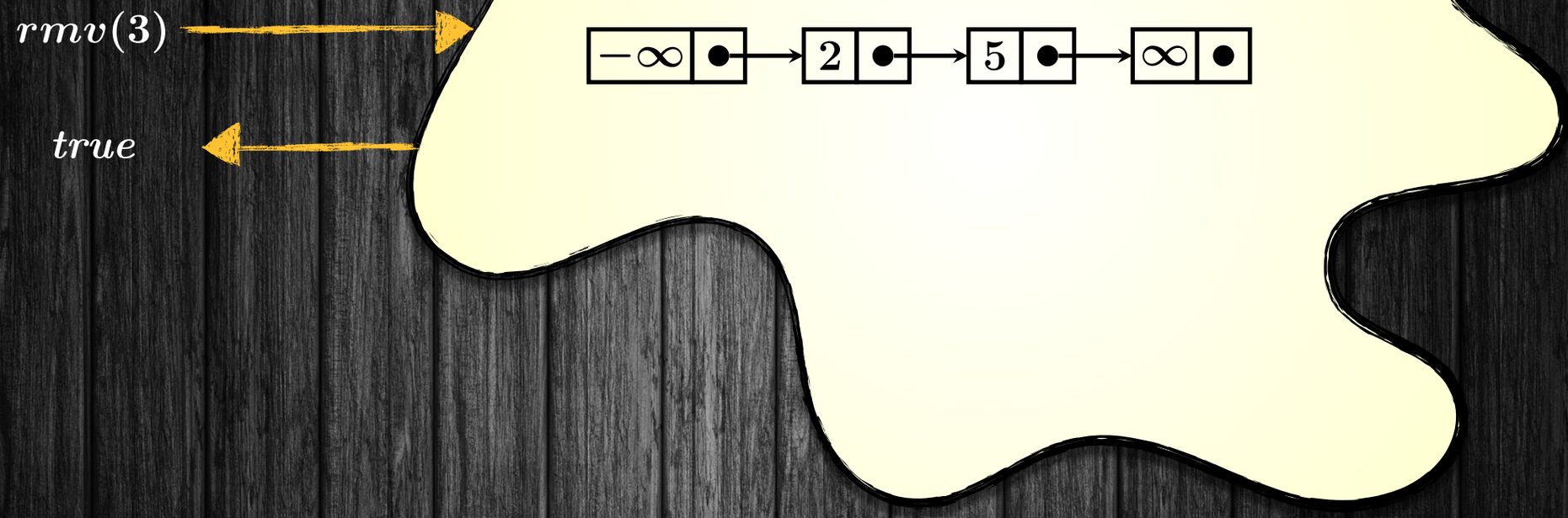


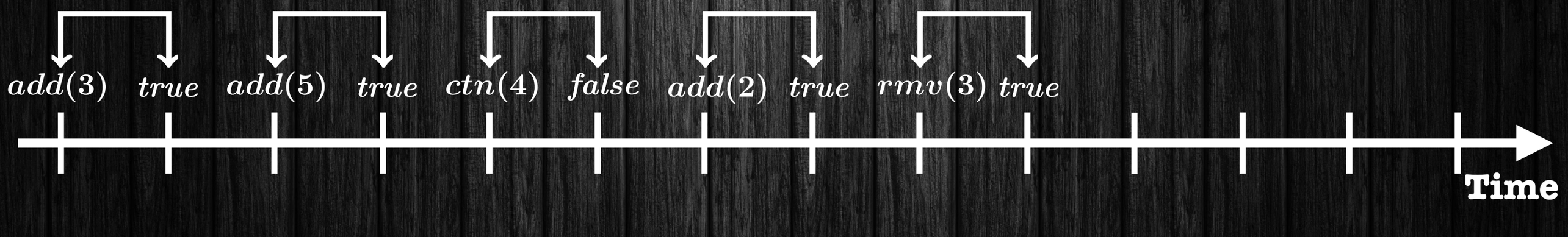
Sets | Sequential Programs



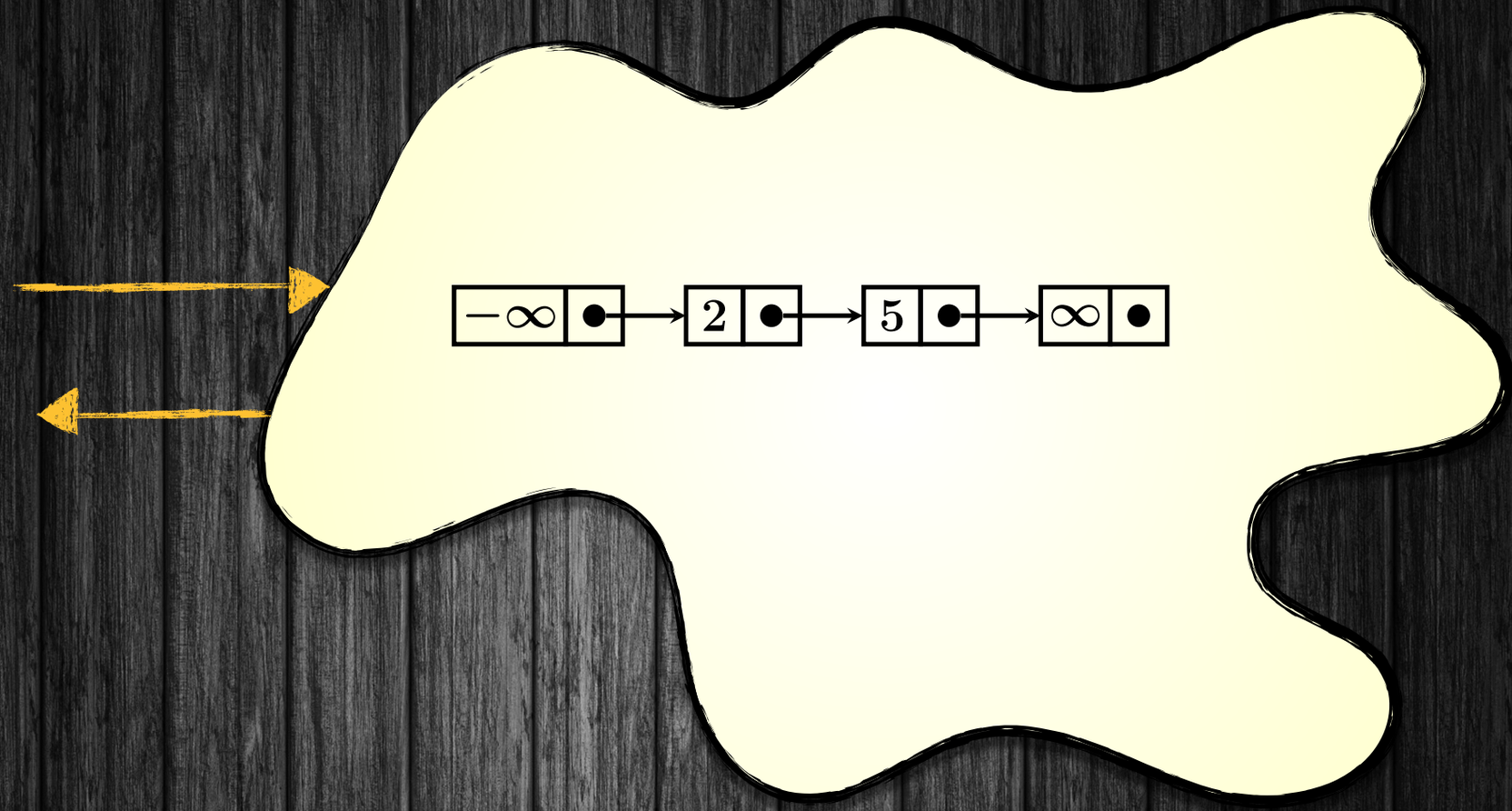


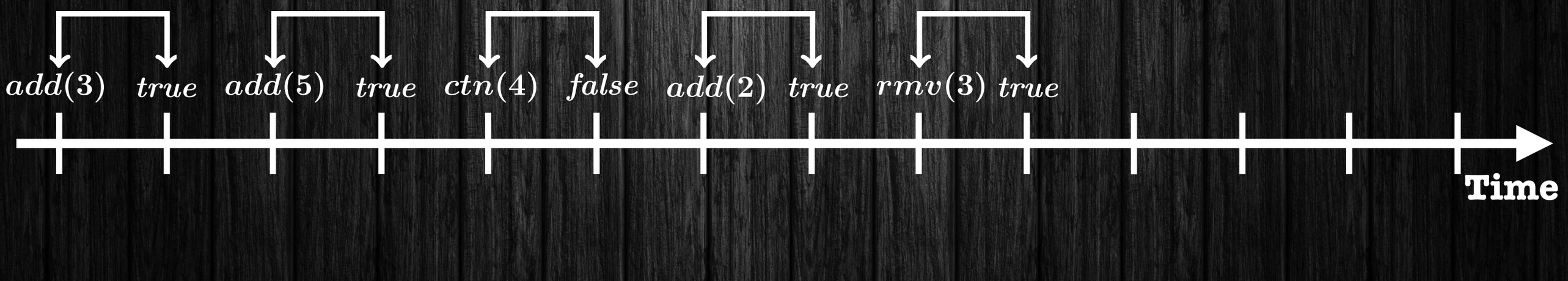
Sets | Sequential Programs



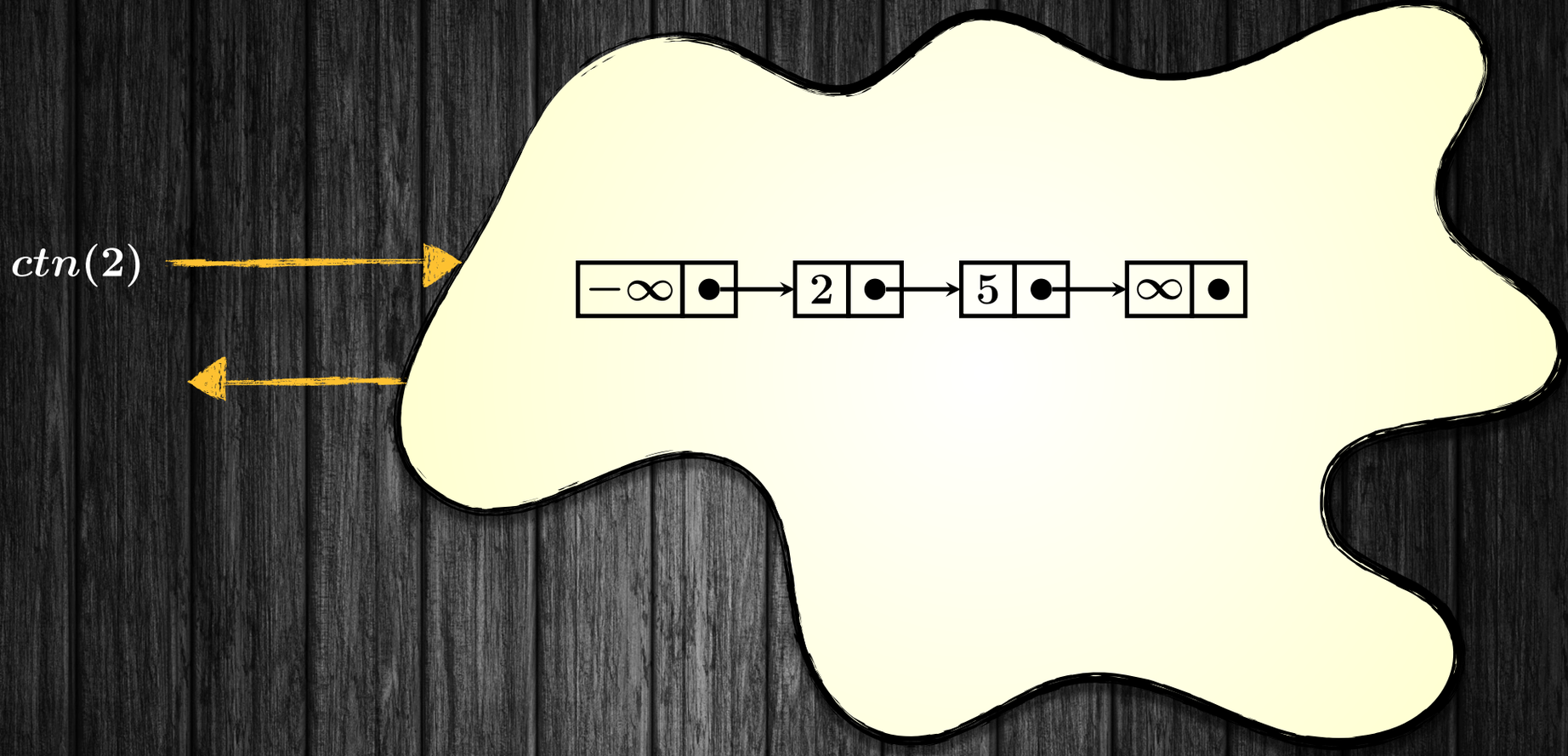


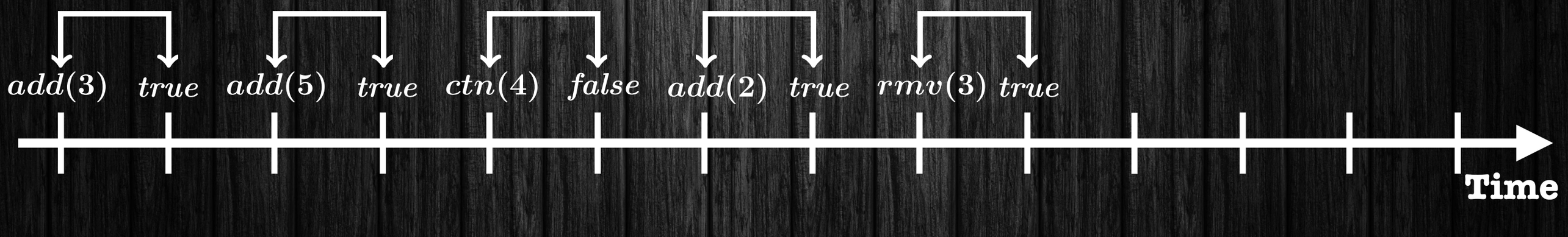
Sets | Sequential Programs



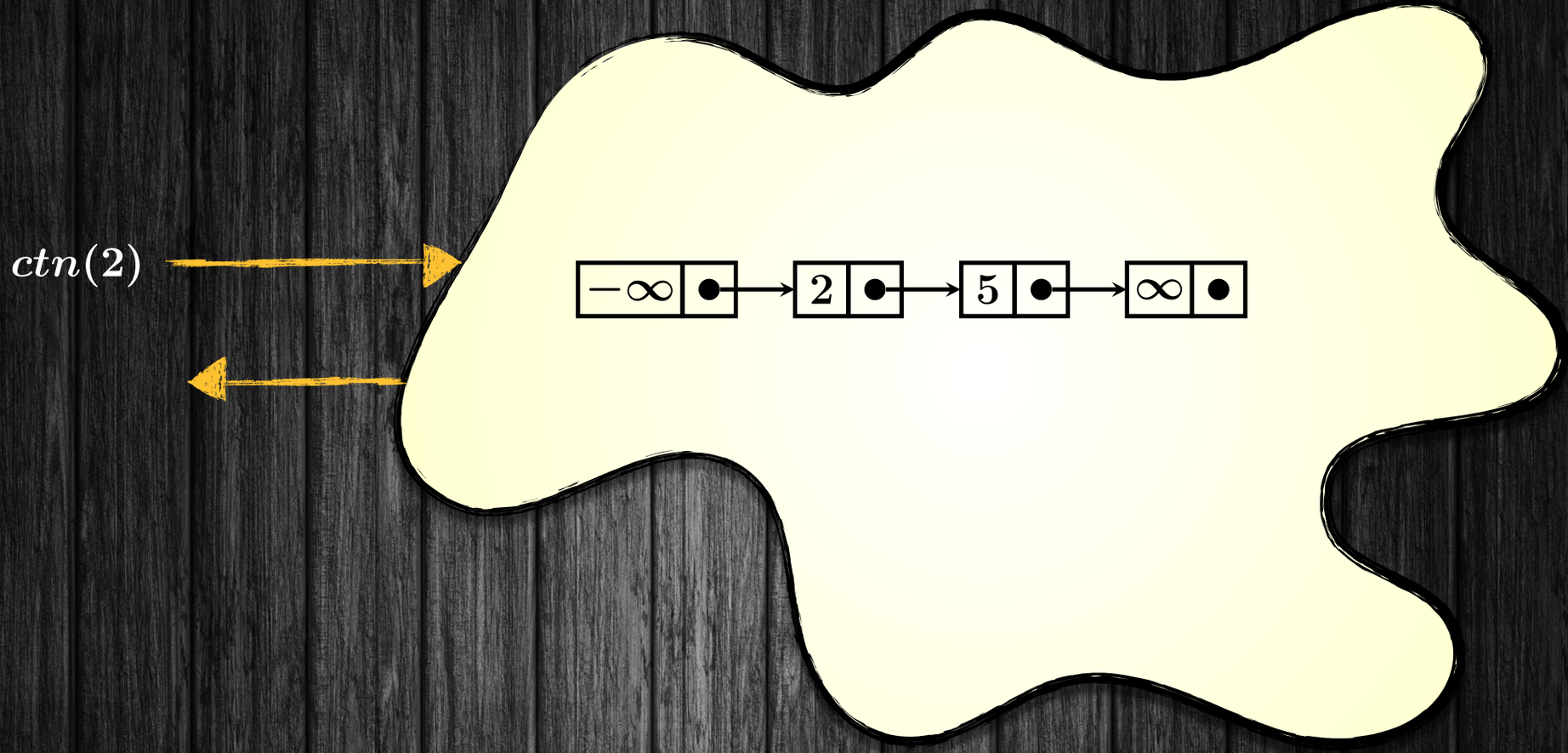


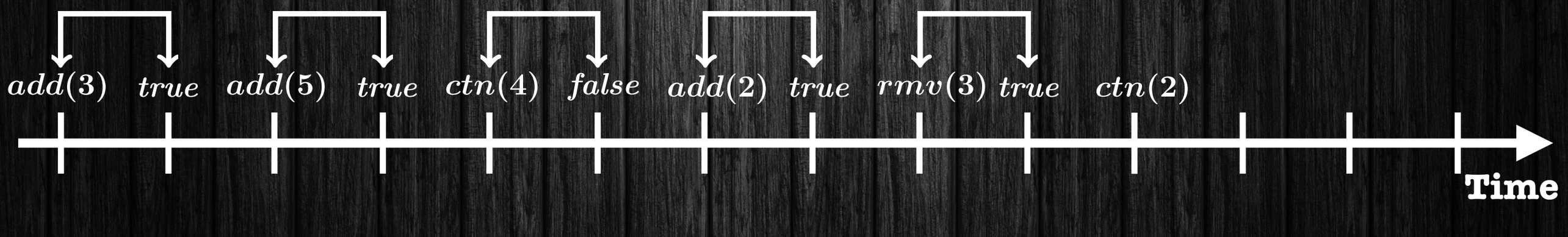
Sets | Sequential Programs



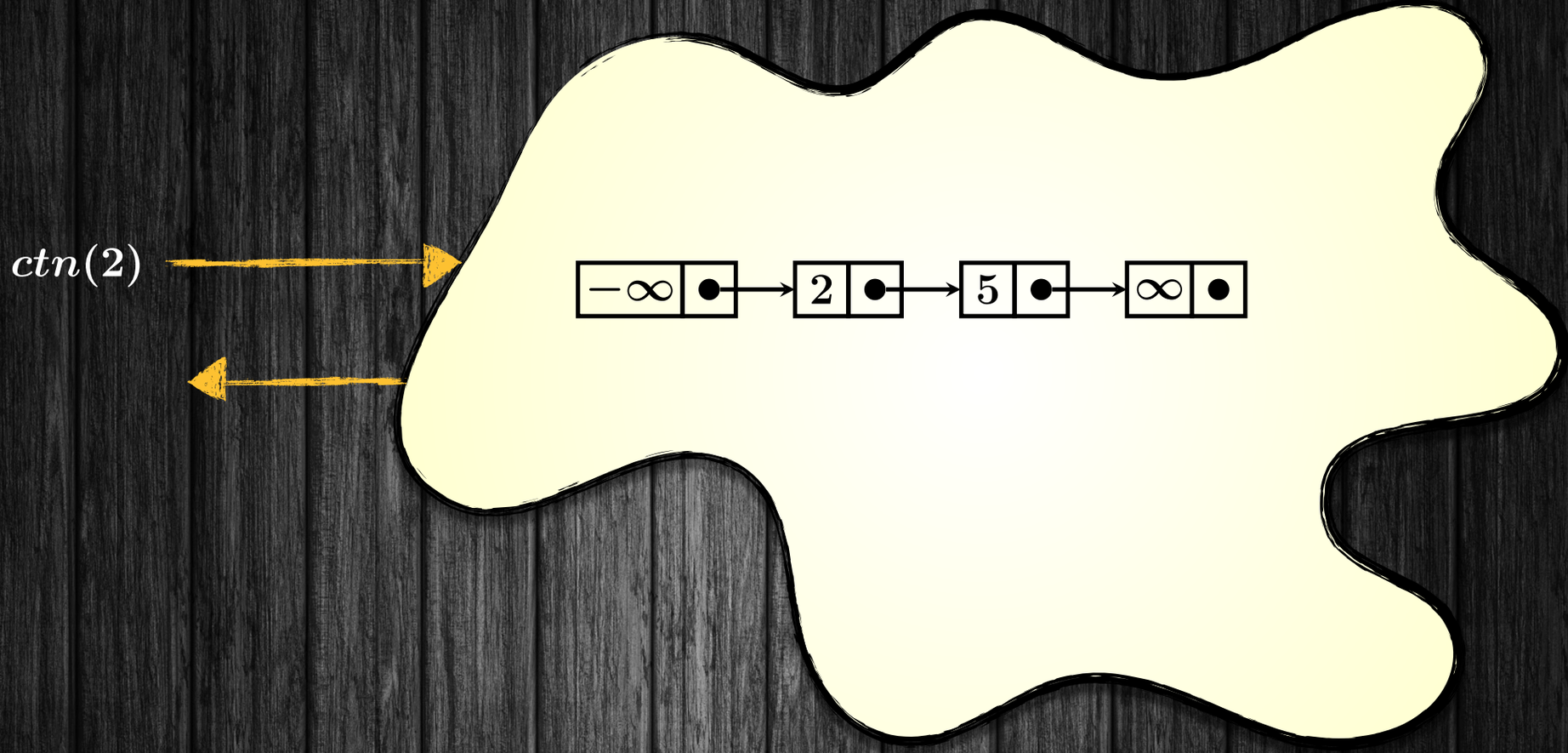


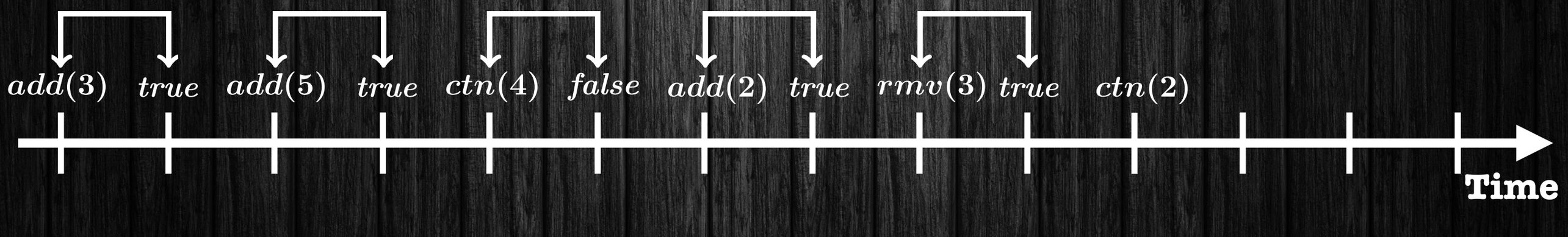
Sets | Sequential Programs



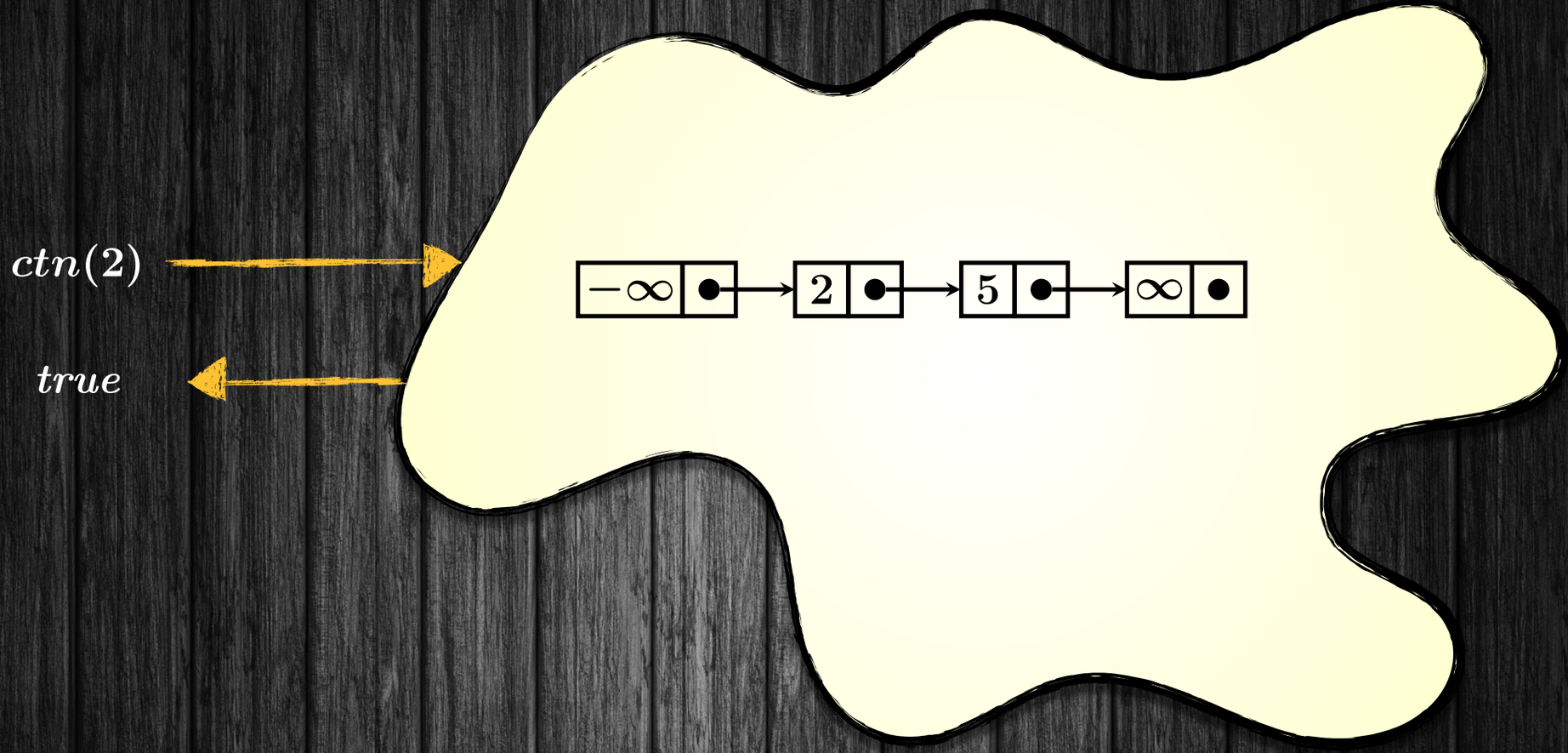


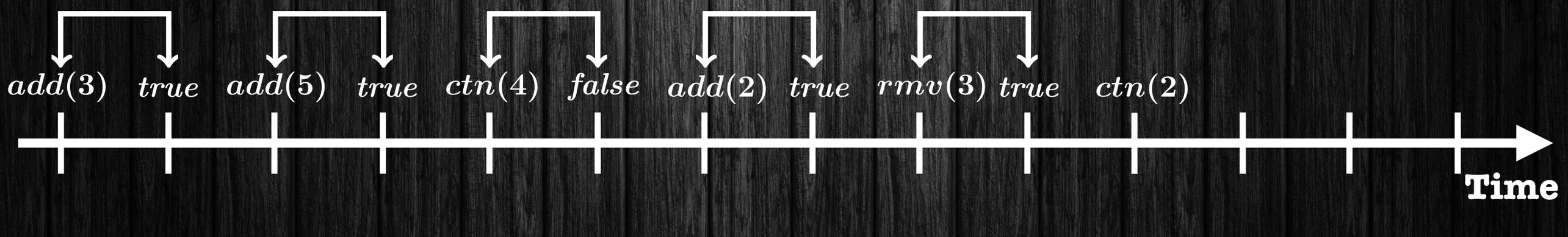
Sets | Sequential Programs



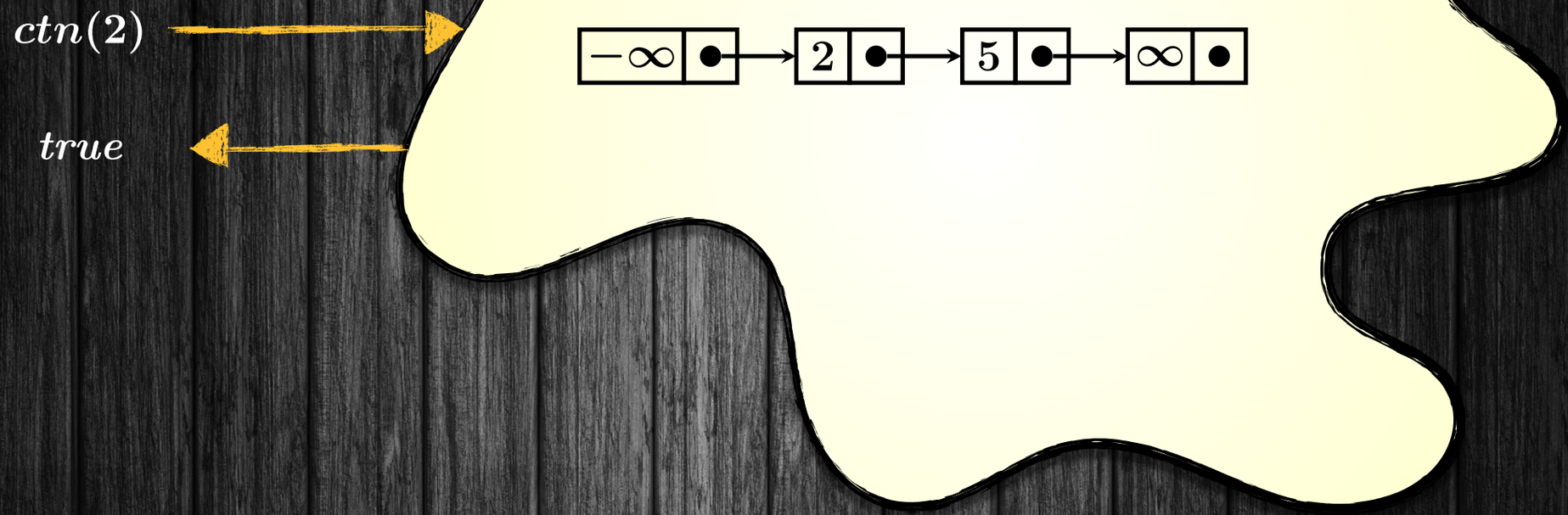


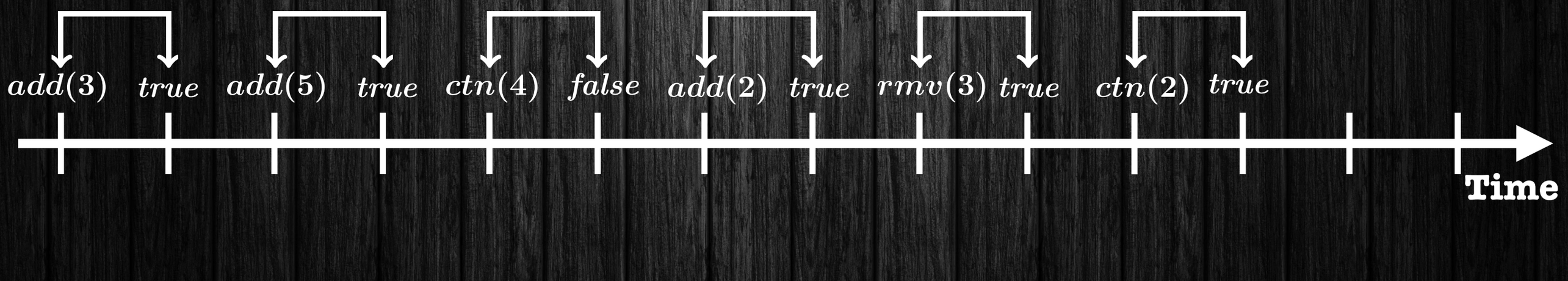
Sets | Sequential Programs



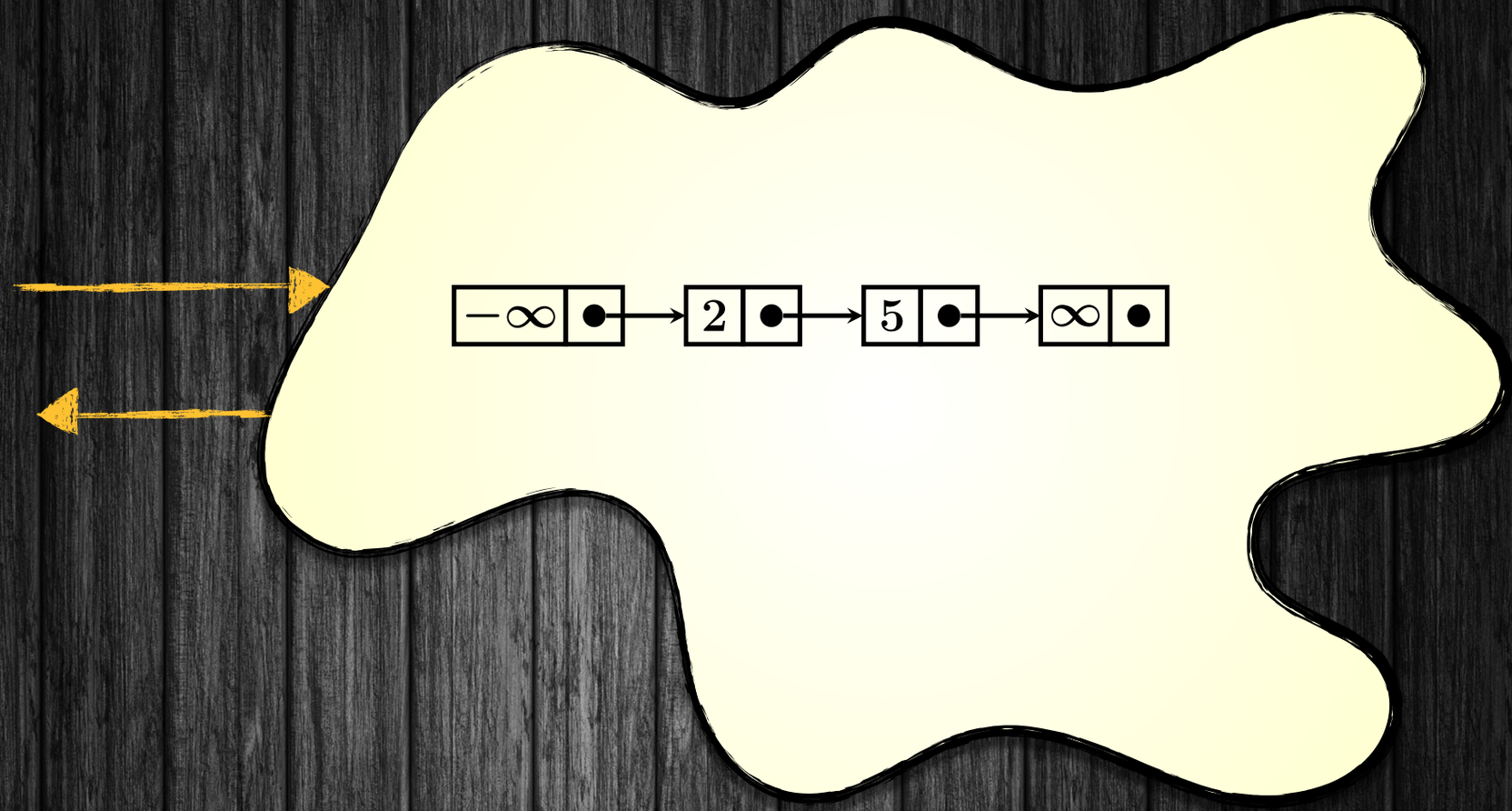


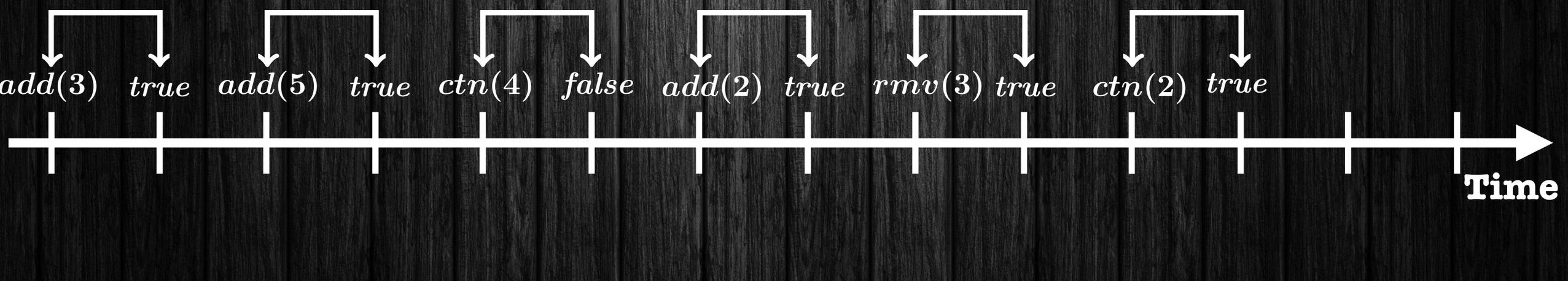
Sets | Sequential Programs



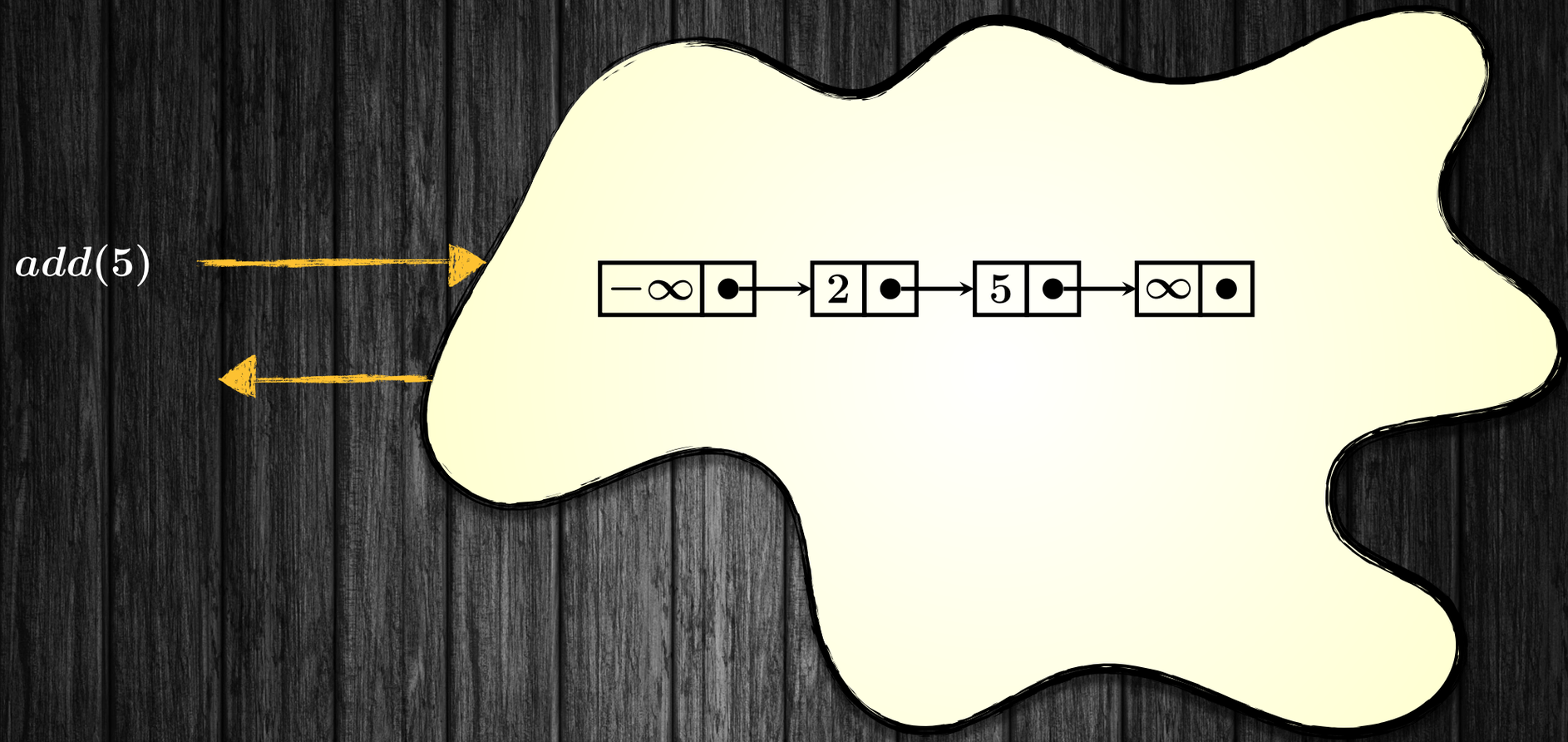


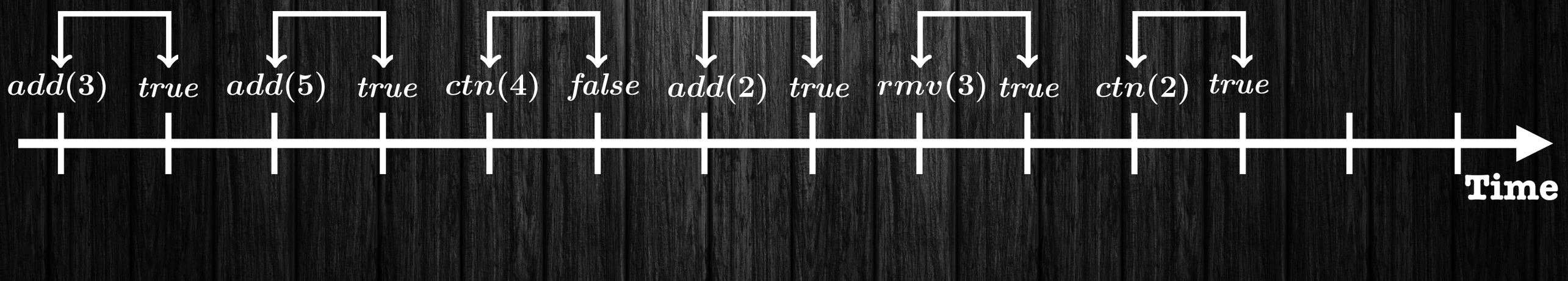
Sets | Sequential Programs



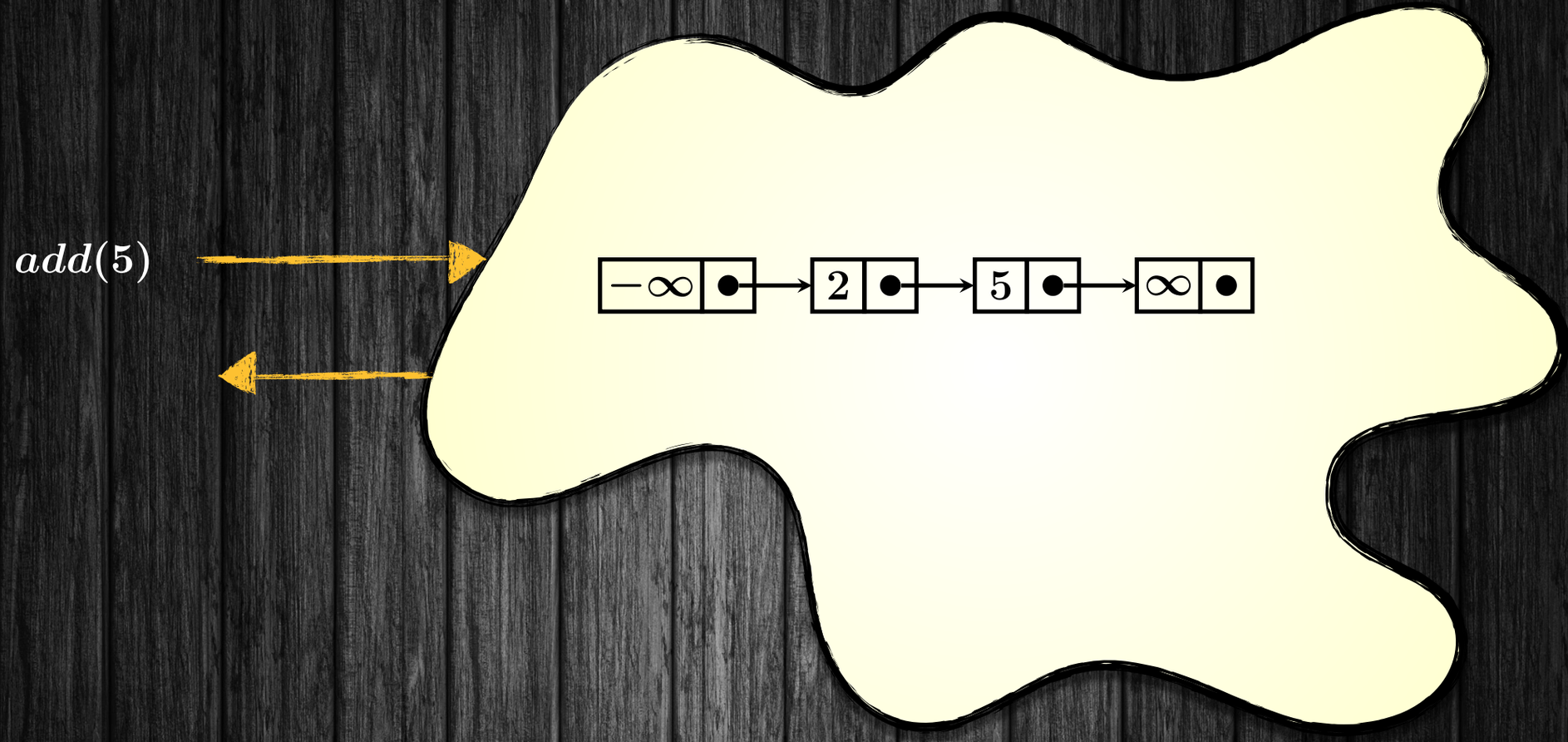


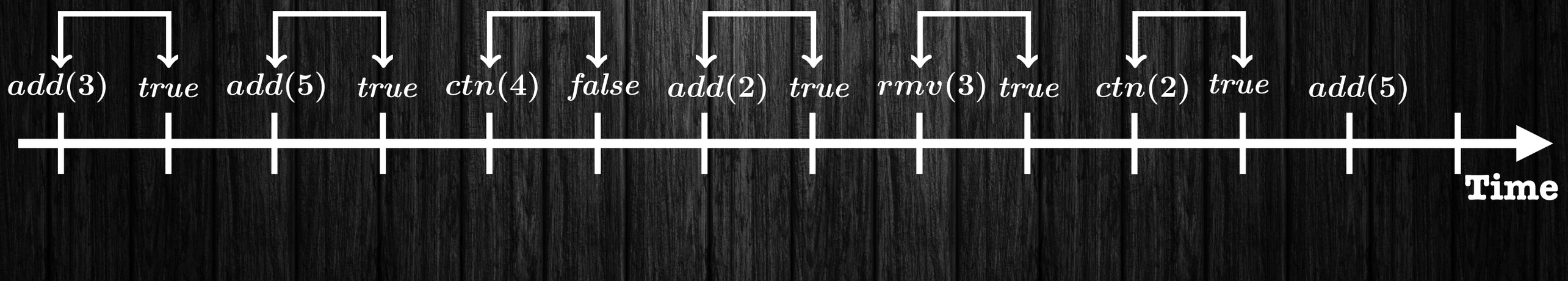
Sets | Sequential Programs



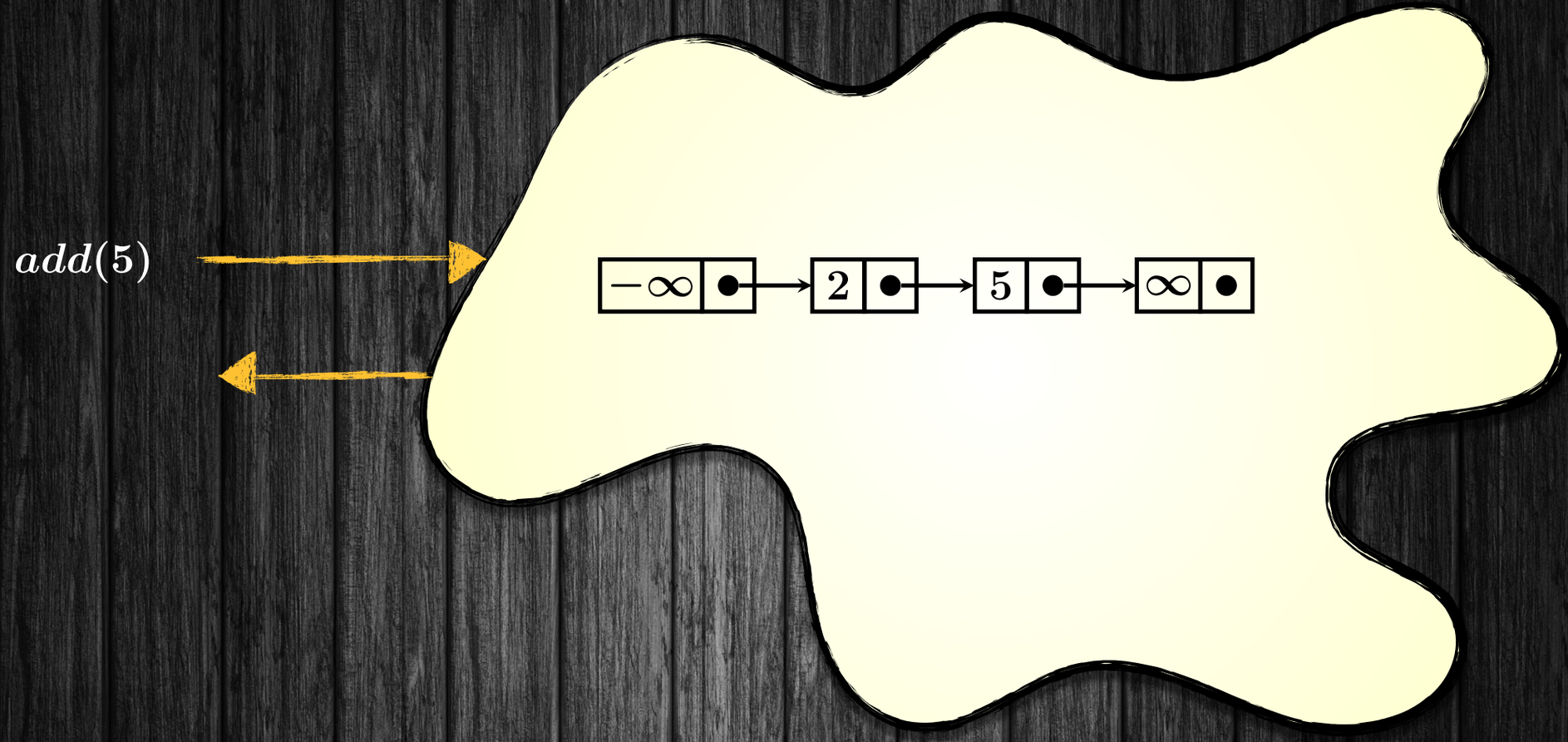


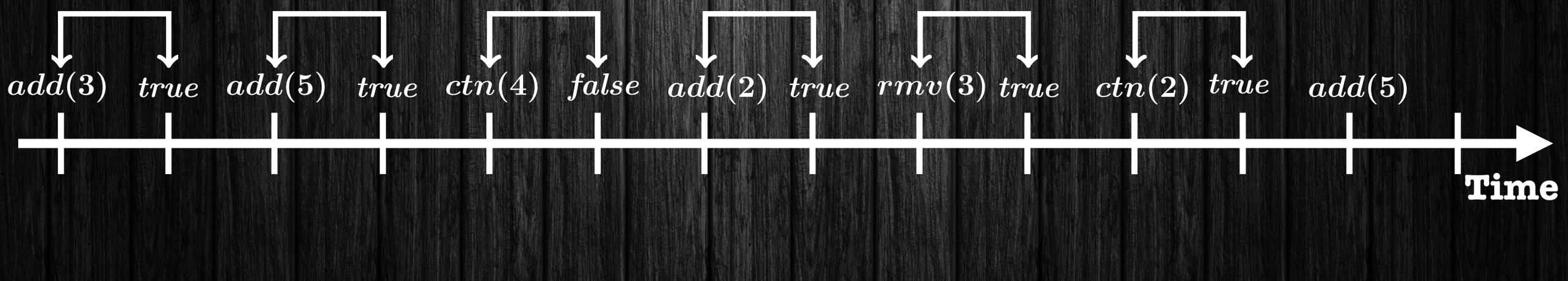
Sets | Sequential Programs



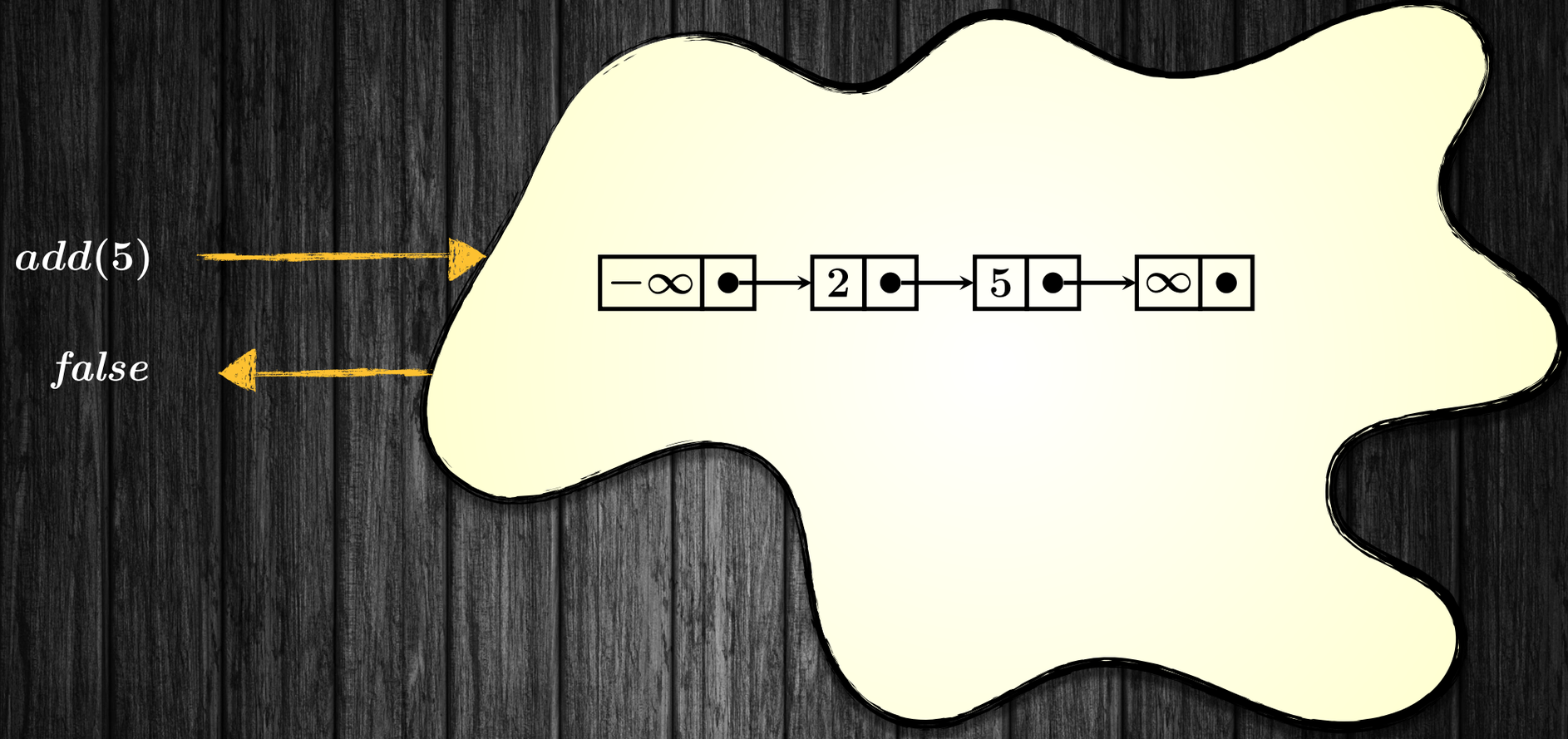


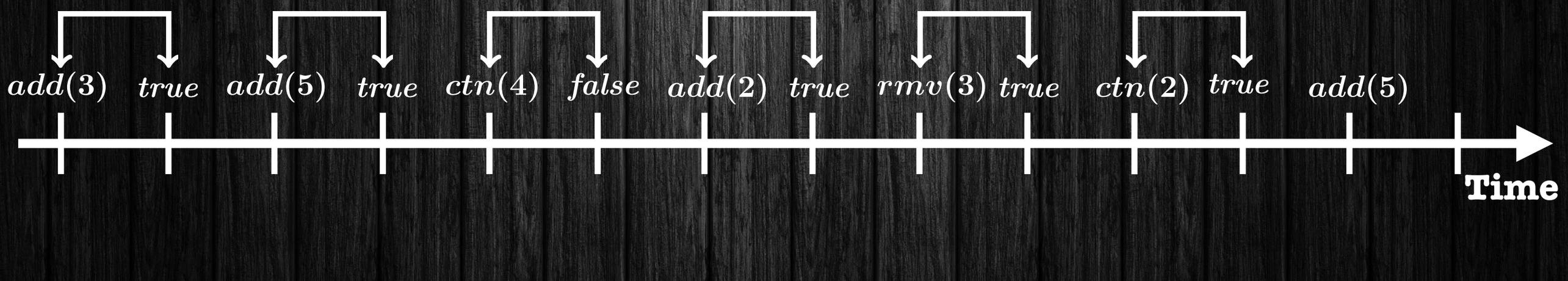
Sets | Sequential Programs



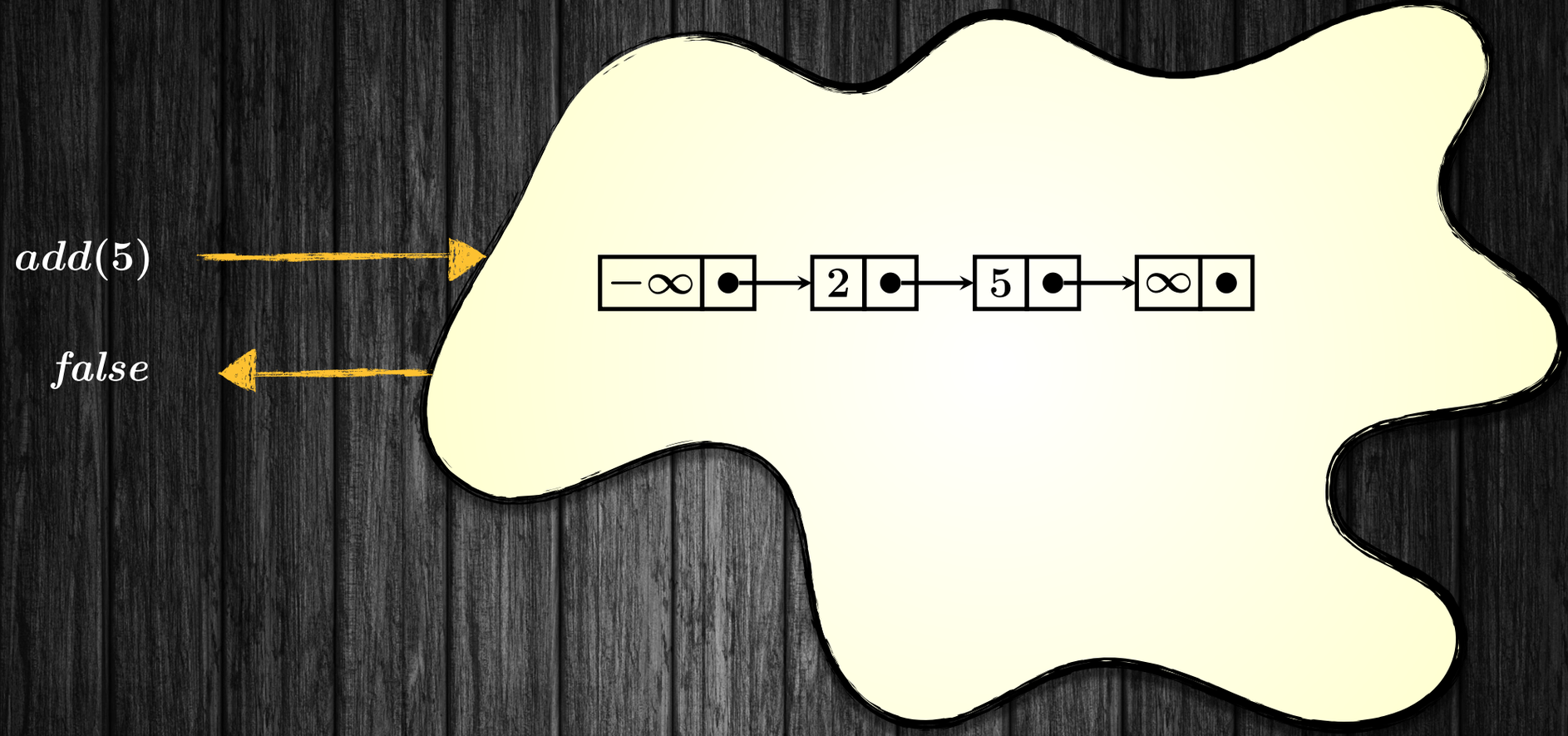


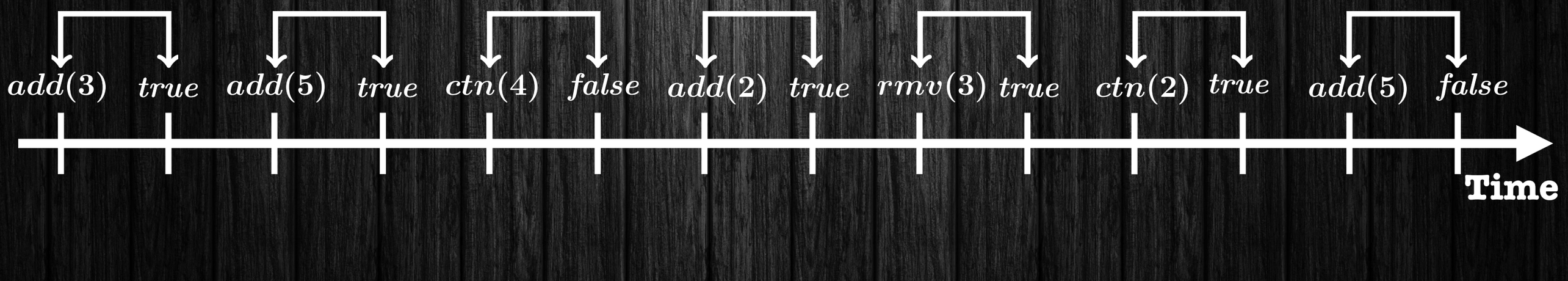
Sets | Sequential Programs



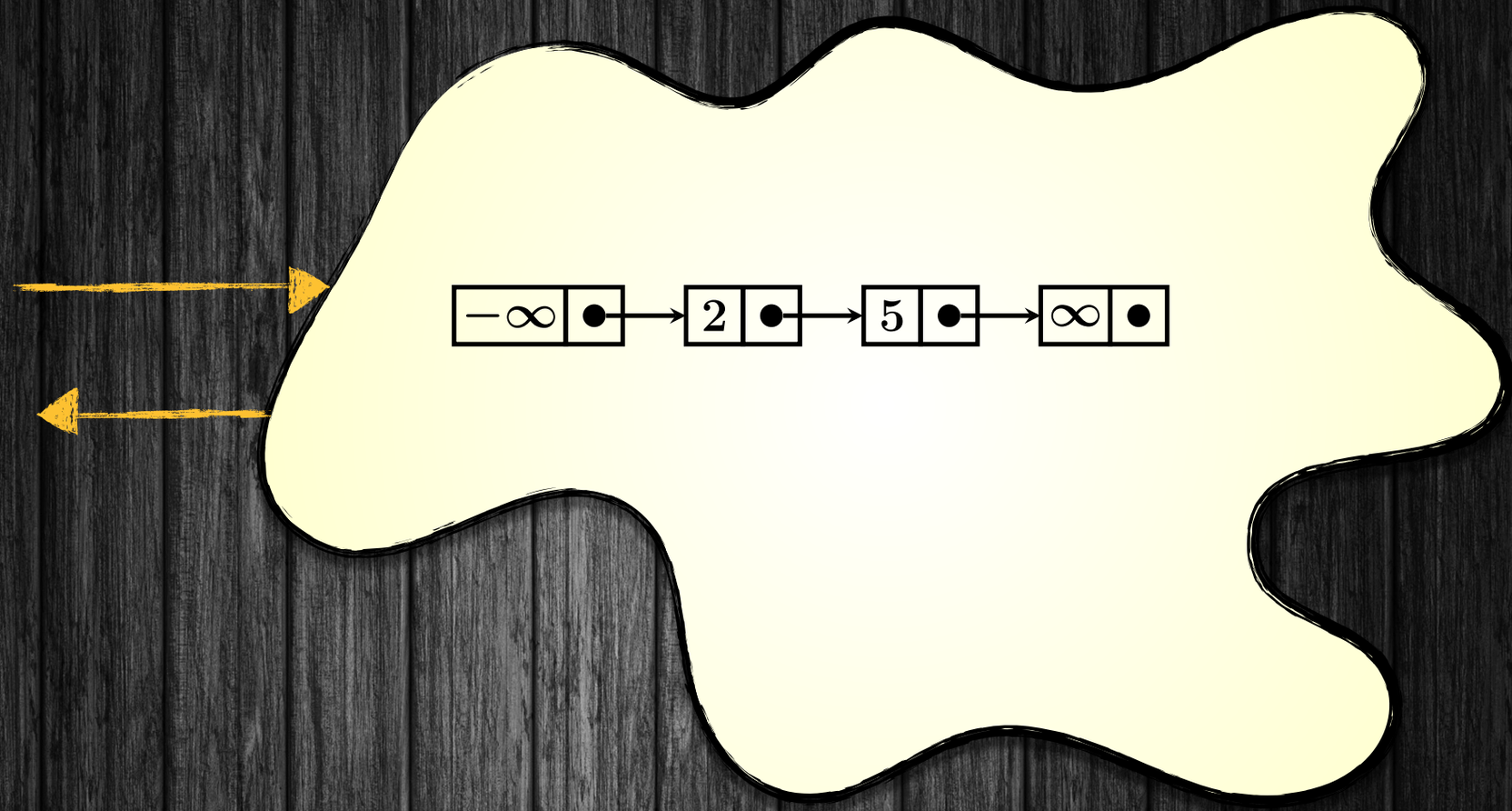


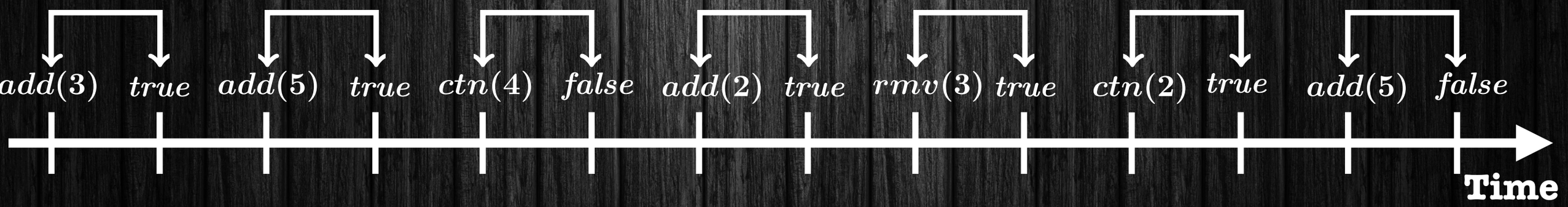
Sets | Sequential Programs



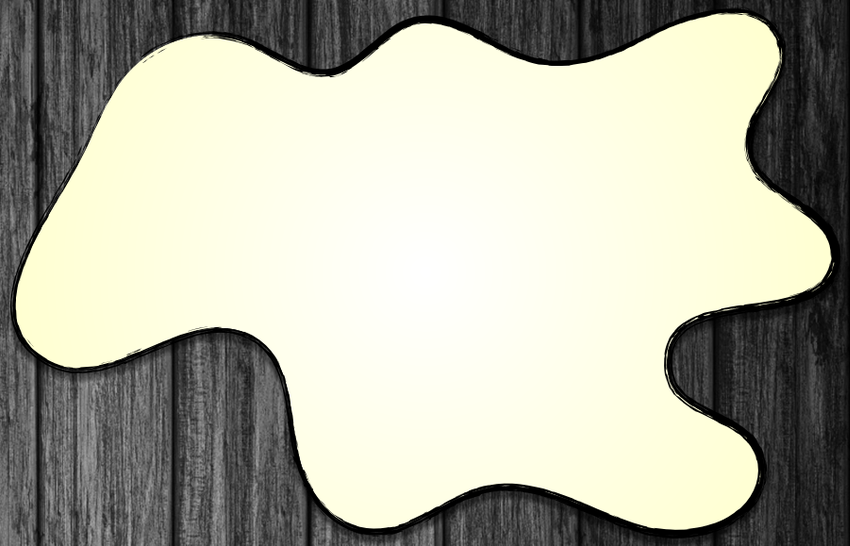


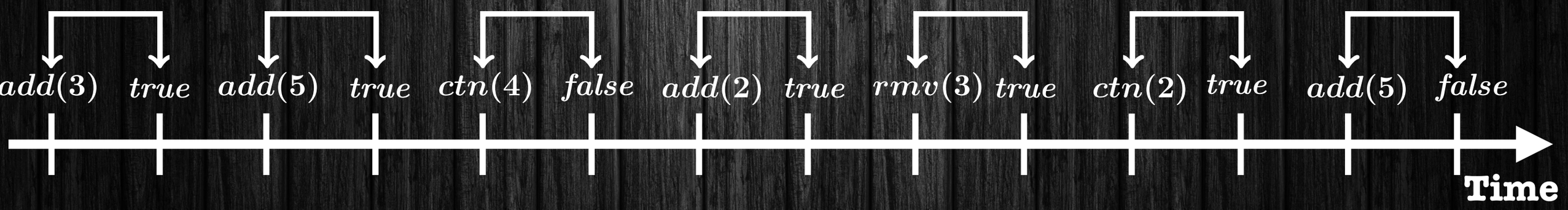
Sets | Sequential Programs



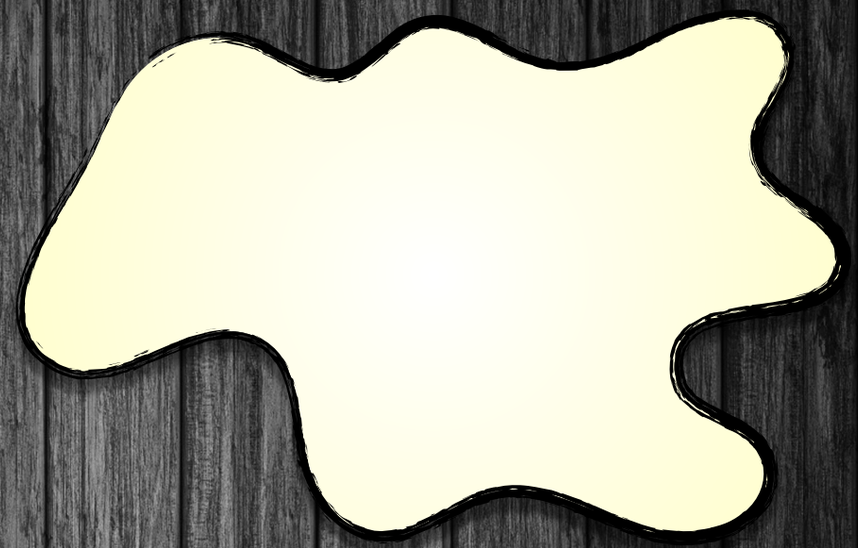


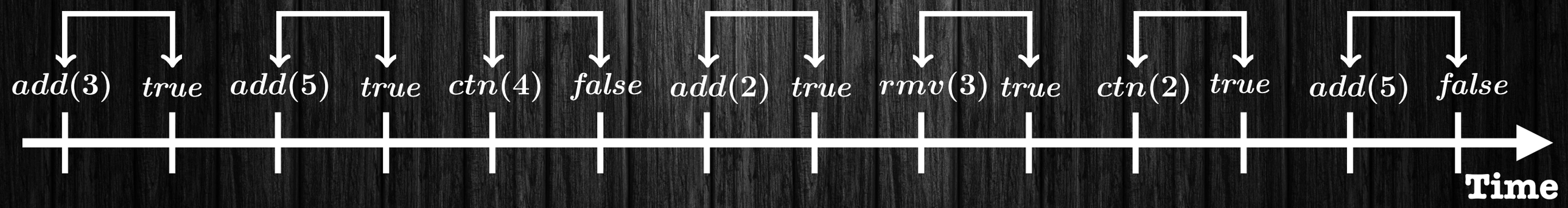
Sets | **Sequential Programs**



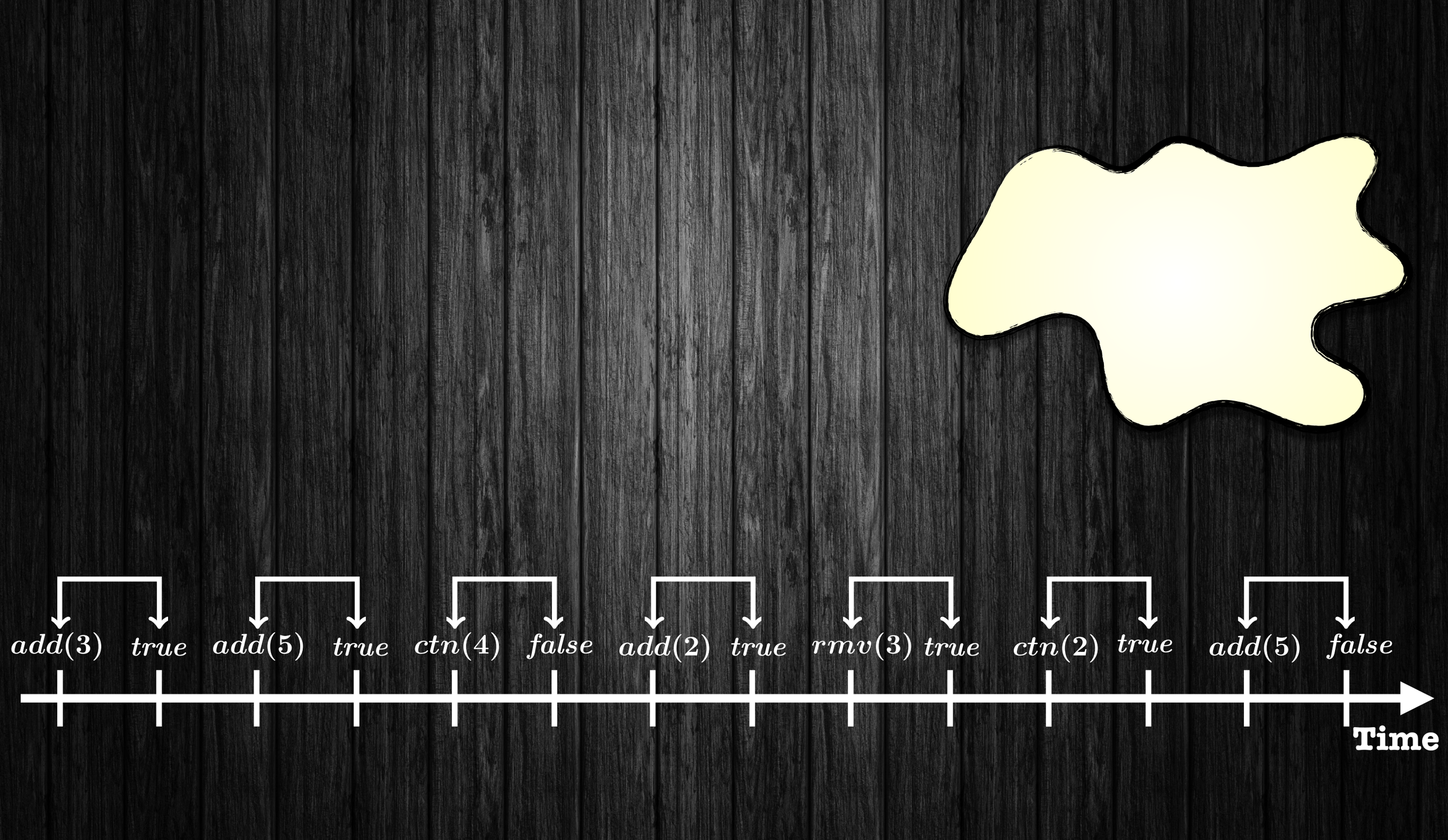


Sets | **Sequential Programs**

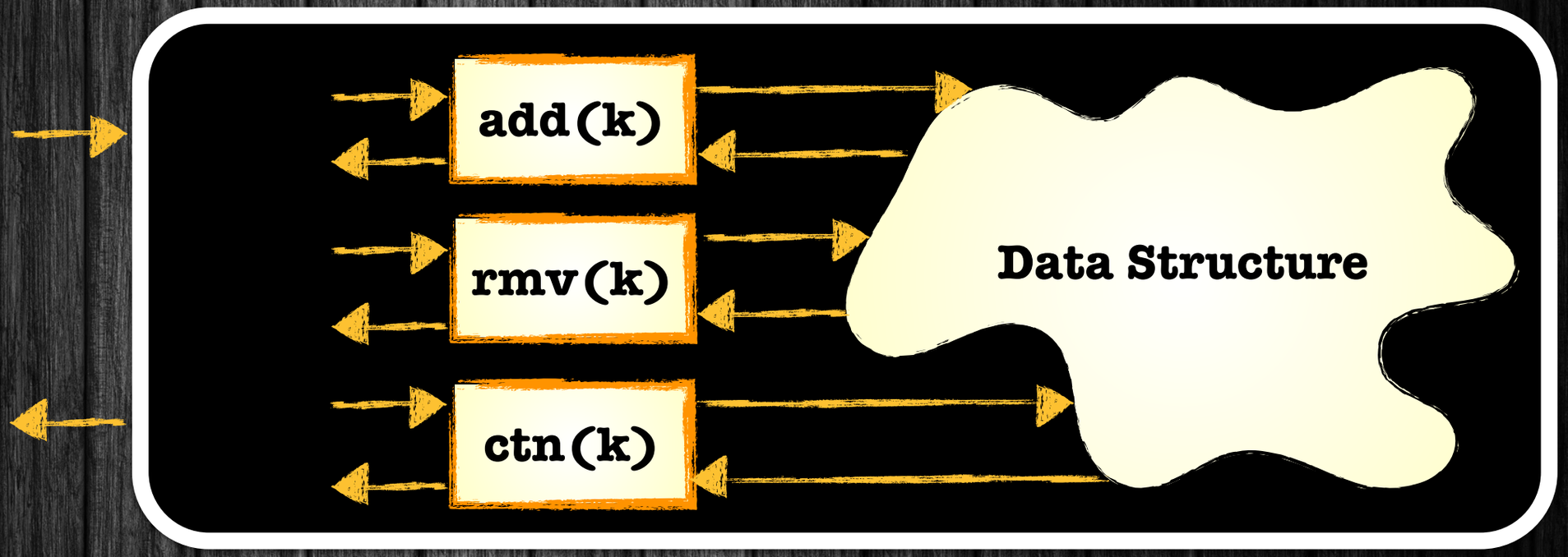


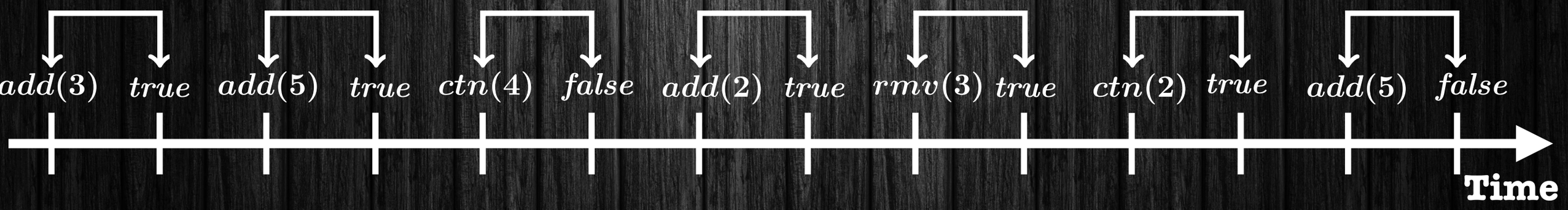


Sets Sequential Programs



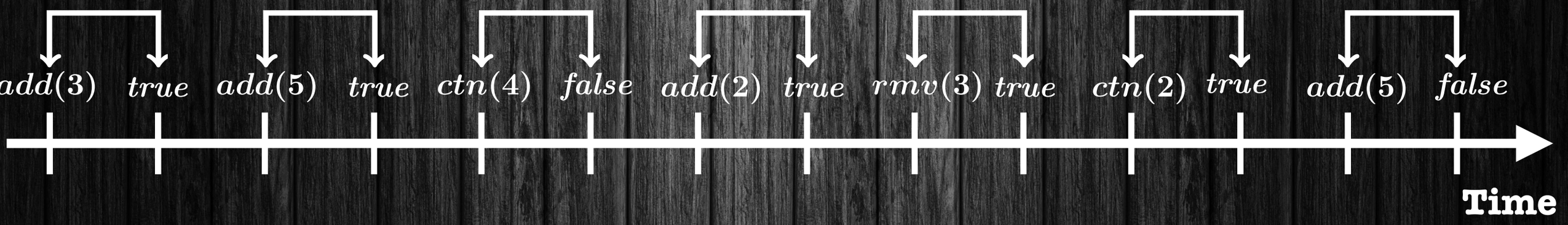
Sets | **Sequential Programs**





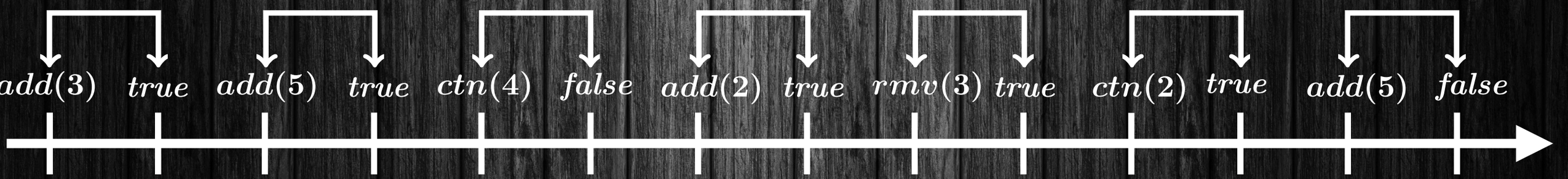
Sets | Sequential Programs

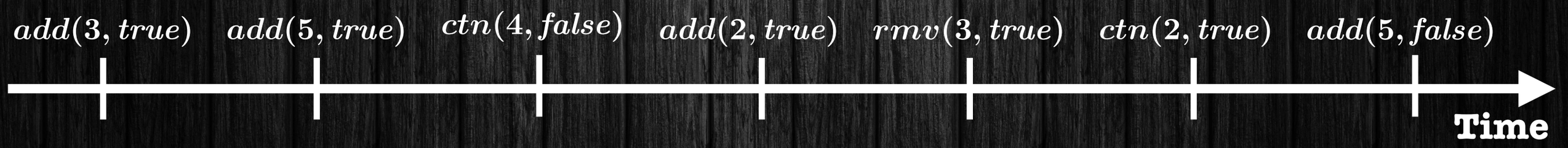




Sets | Sequential Programs

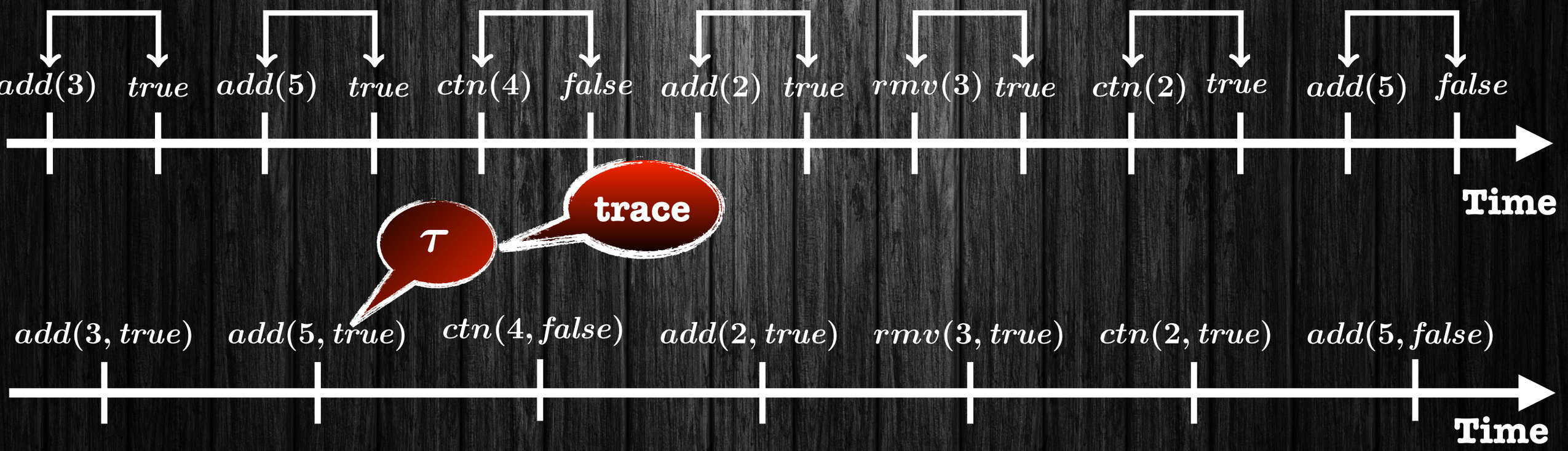






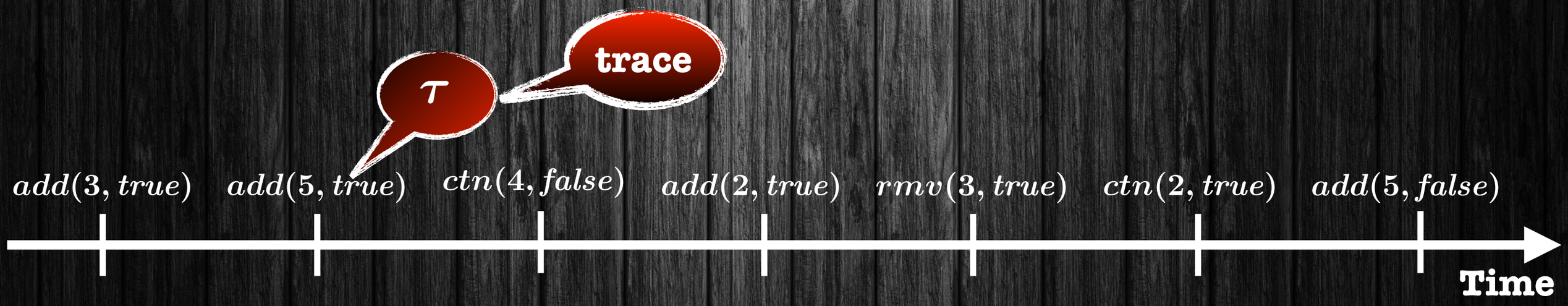
Sets | Sequential Programs





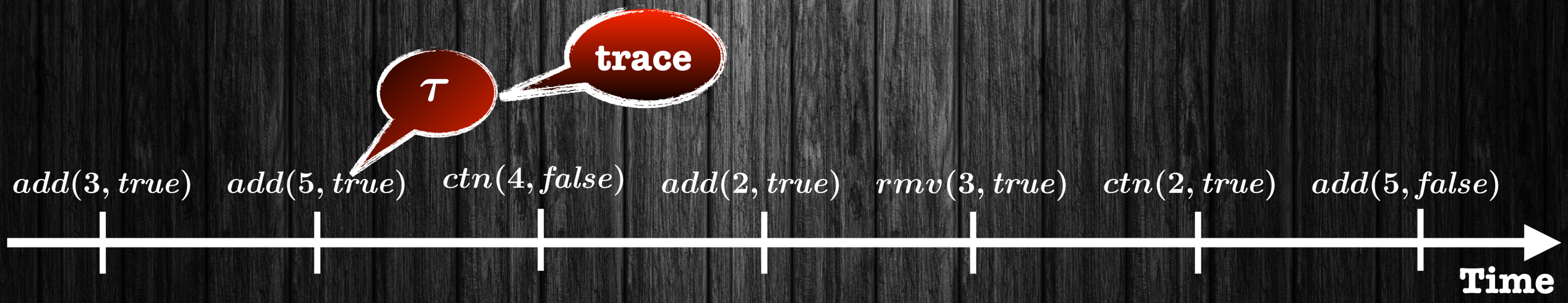
Sets | Sequential Programs





Sets | Sequential Programs





Sets

Abstract Data Type

- States
- Actions
- initial state

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

initially empty

set of integers

state



actions

rules

guarded commands

Traces (Set)

$add(2,true)$ $add(5,true)$ $add(2,false)$ $rmv(5,true)$ $rmv(3,false)$ $ctn(3,false)$ $add(9,true)$ ✓

$add(2,true)$ $add(5,true)$ $ctn(2,true)$ ✓

$add(2,true)$ $add(5,true)$ $ctn(2,false)$ ✗

$add(2,true)$ $add(5,true)$ $ctn(3,true)$ ✗

Sets

Abstract Data Type

- States
- Actions
- initial state

$$S \xrightarrow{add(x,true)} S \cup \{x\} \quad \text{if } x \notin S$$

$$S \xrightarrow{add(x,false)} S \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,true)} S - \{x\} \quad \text{if } x \in S$$

$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

$$S \xrightarrow{ctn(x,true)} S \quad \text{if } x \in S$$

$$S \xrightarrow{ctn(x,false)} S \quad \text{if } x \notin S$$

initially empty

set of integers

state



actions

rules

guarded commands

Traces (Set)

$add(2,true)$ $add(5,true)$ $add(2,false)$ $rmv(5,true)$ $rmv(3,false)$ $ctn(3,false)$ $add(9,true)$ ✓

$add(2,true)$ $add(5,true)$ $ctn(2,true)$ ✓

$add(2,true)$ $add(5,true)$ $ctn(2,false)$ ✗

$add(2,true)$ $add(5,true)$ $ctn(3,true)$ ✗

Sets | Sequential Programs | Correctness

Safety Properties

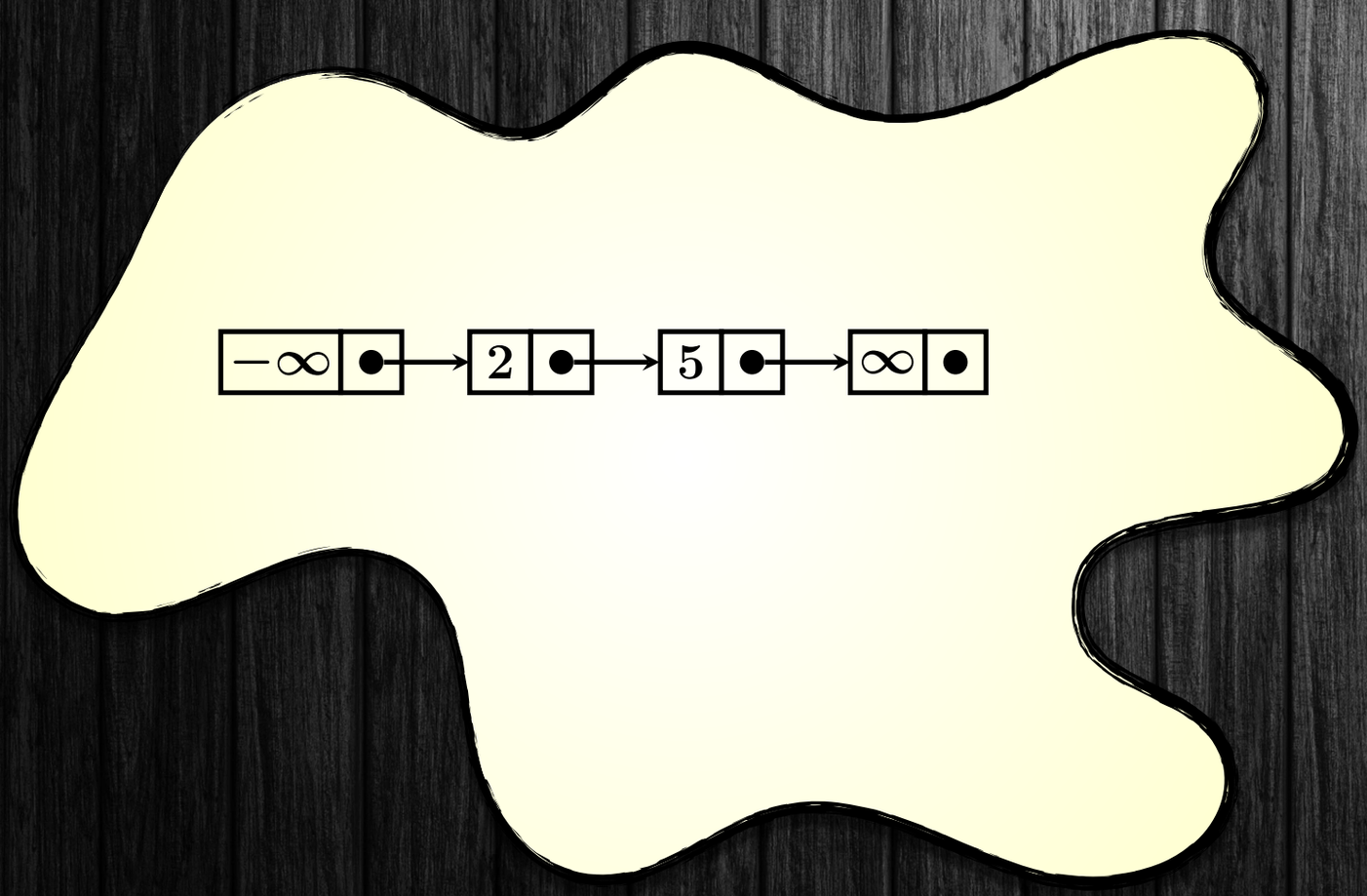
Traces(P)

Traces(Set)

- Nothing bad will happen
- No wrong result will be delivered
- Will not violate the abstract data type
- Traces(P) $\subseteq$ Traces(Set)

Liveness Properties

- Something good will happen
- Every operation will eventually terminate
- Every call will return

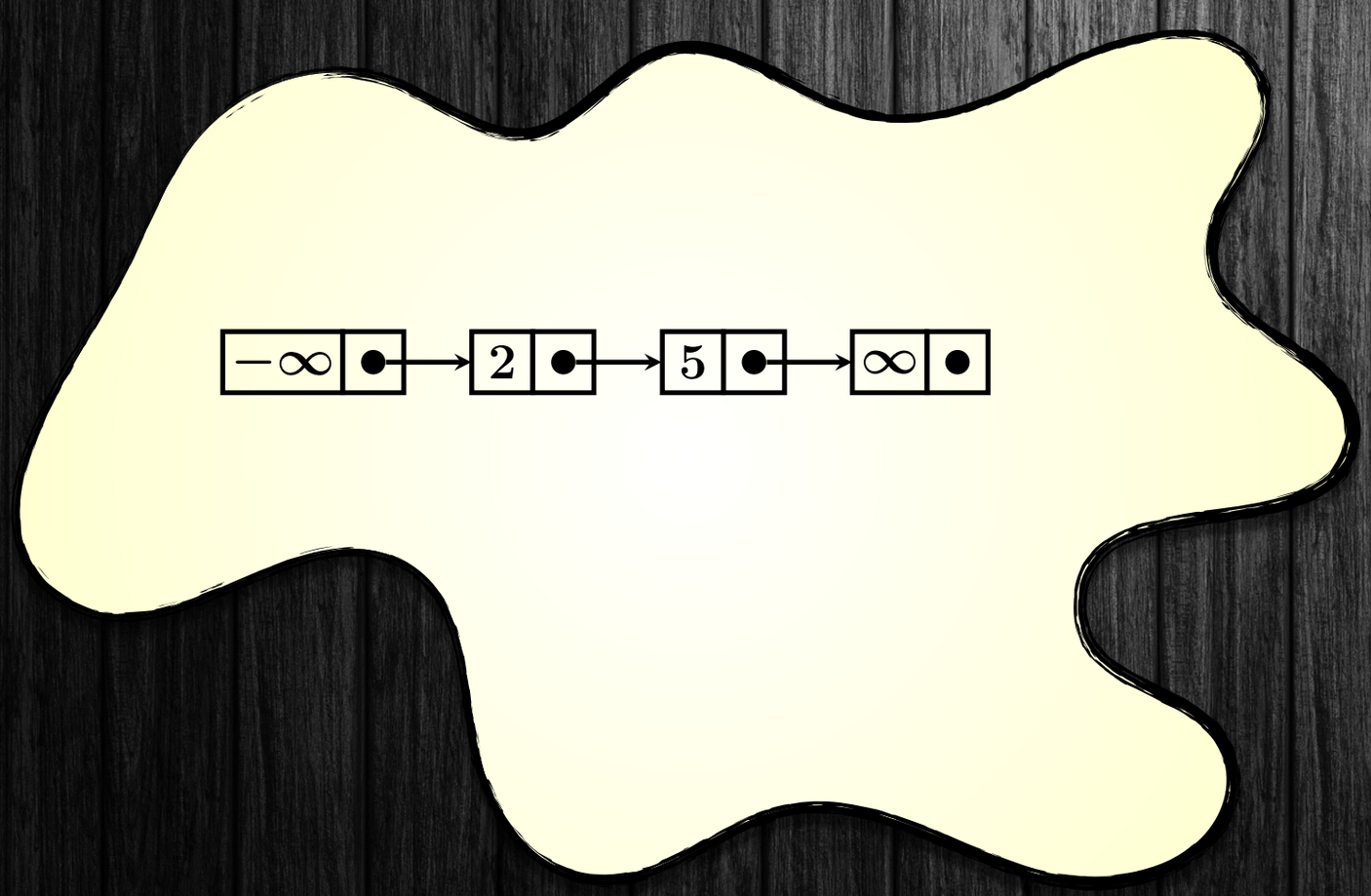



```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

Sets | Sequential Programs | Correctness

Liveness Properties

- Something good will happen
- Every operation will eventually terminate
- Every call will return

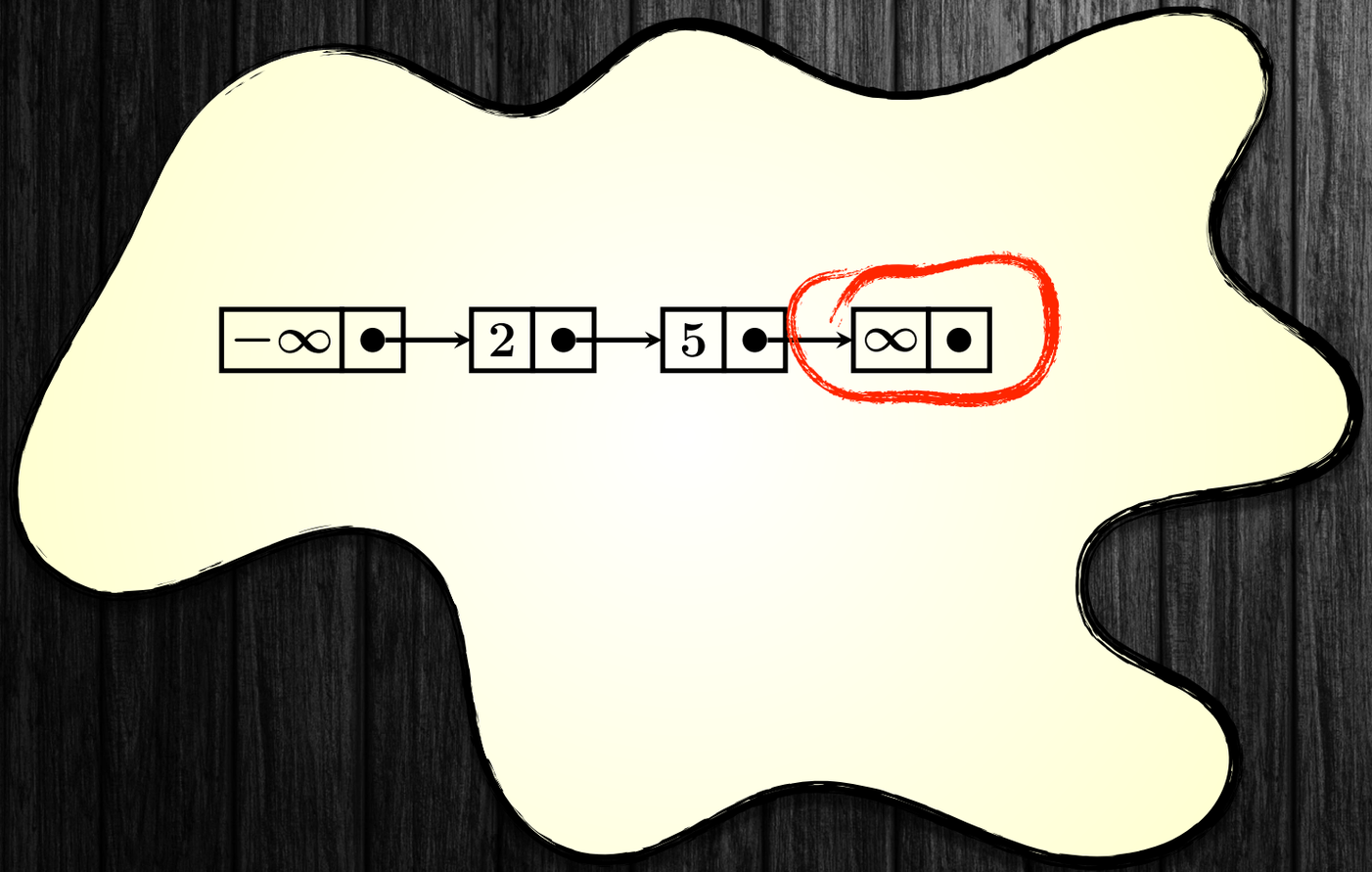



```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

Sets | Sequential Programs | Correctness

Liveness Properties

- Something good will happen
- Every operation will eventually terminate
- Every call will return

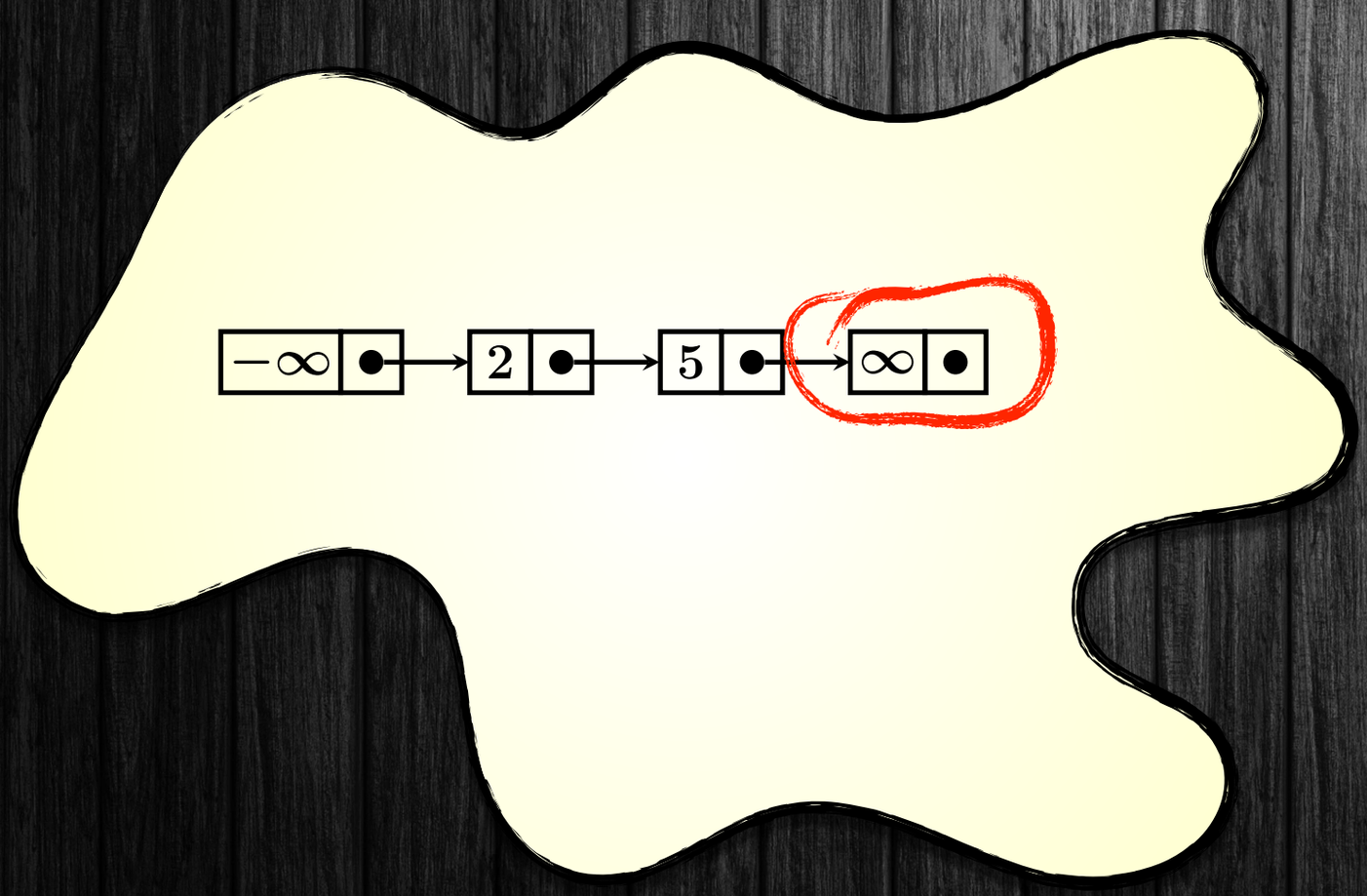



```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

Sets | Sequential Programs | Correctness

Liveness Properties

- Something good will happen
- Every operation will eventually terminate
- Every call will return



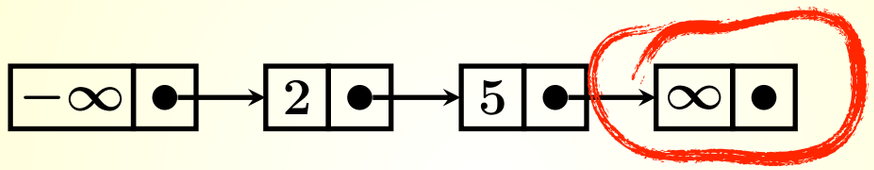

```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```

Sets | Sequential Programs | Correctness

Liveness Properties

- Something good will happen
- Every operation will eventually terminate
- Every call will return

```
add(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return false
8  else
9    n = new Node(k,-)
10   n.next = c
11   p.next = n
12   return true
```



Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

Outline

- Overview
- Abstract Data Types
- Sequential Programs
- Concurrent Programs
- Coarse-Grained Locking
- Fine-Grained Locking
- Lazy Lists
- Optimistic Lists
- Treiber Stack Algorithm
- Michael&Scott Queue Algorithm

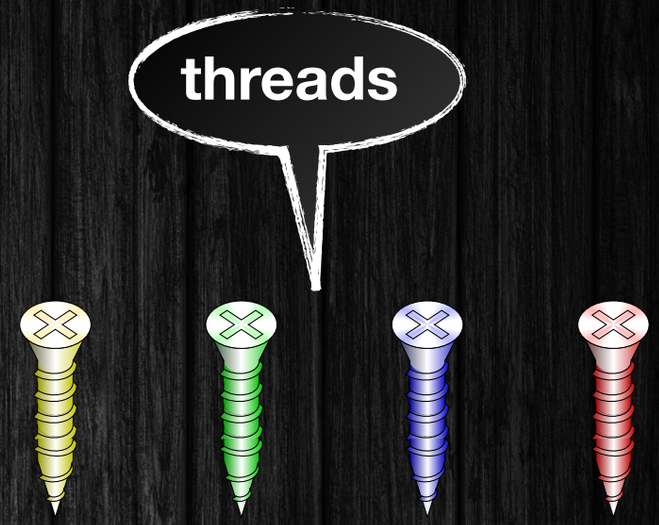
Concurrent Programs

- Concurrent Histories
- Linearizability

Concurrent Programs

- Concurrent Histories
- Linearizability

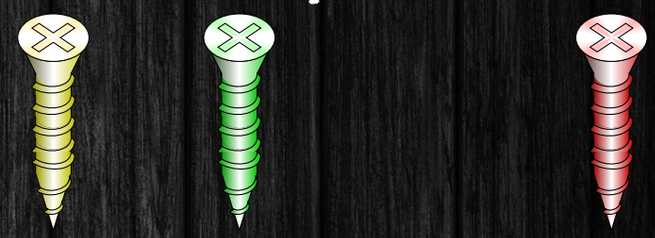
Concurrent Histories

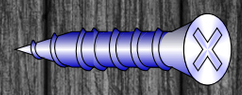


Program

Concurrent Histories

threads

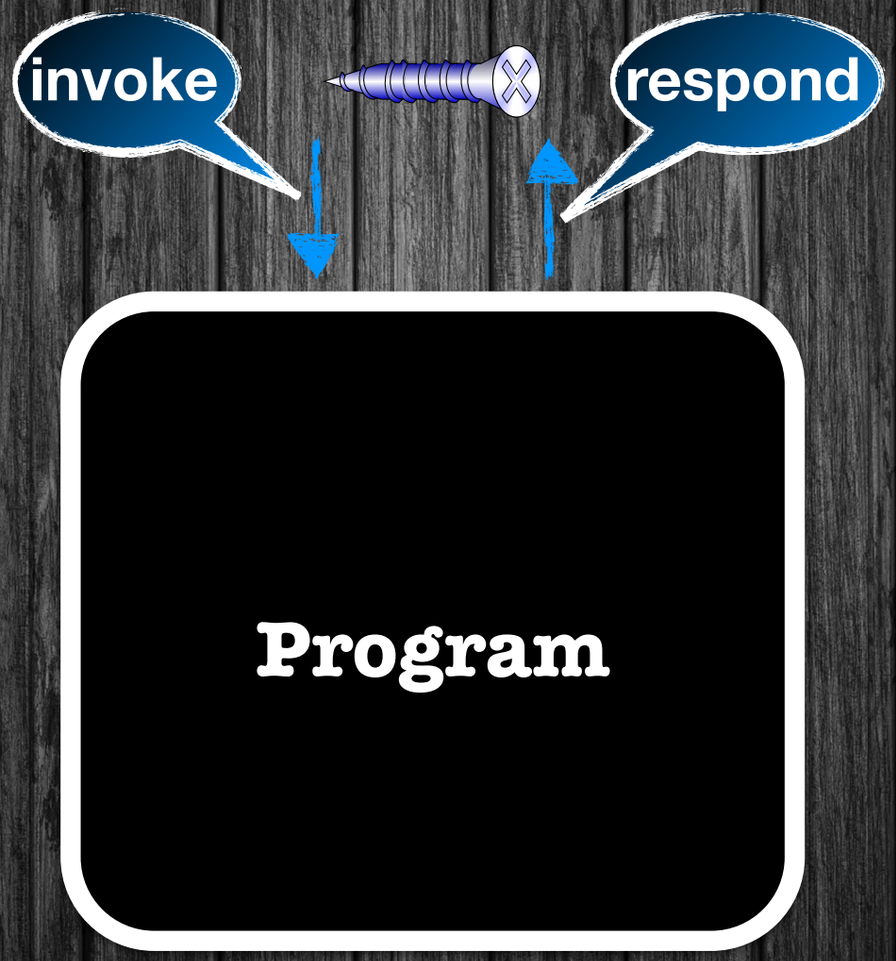




Program

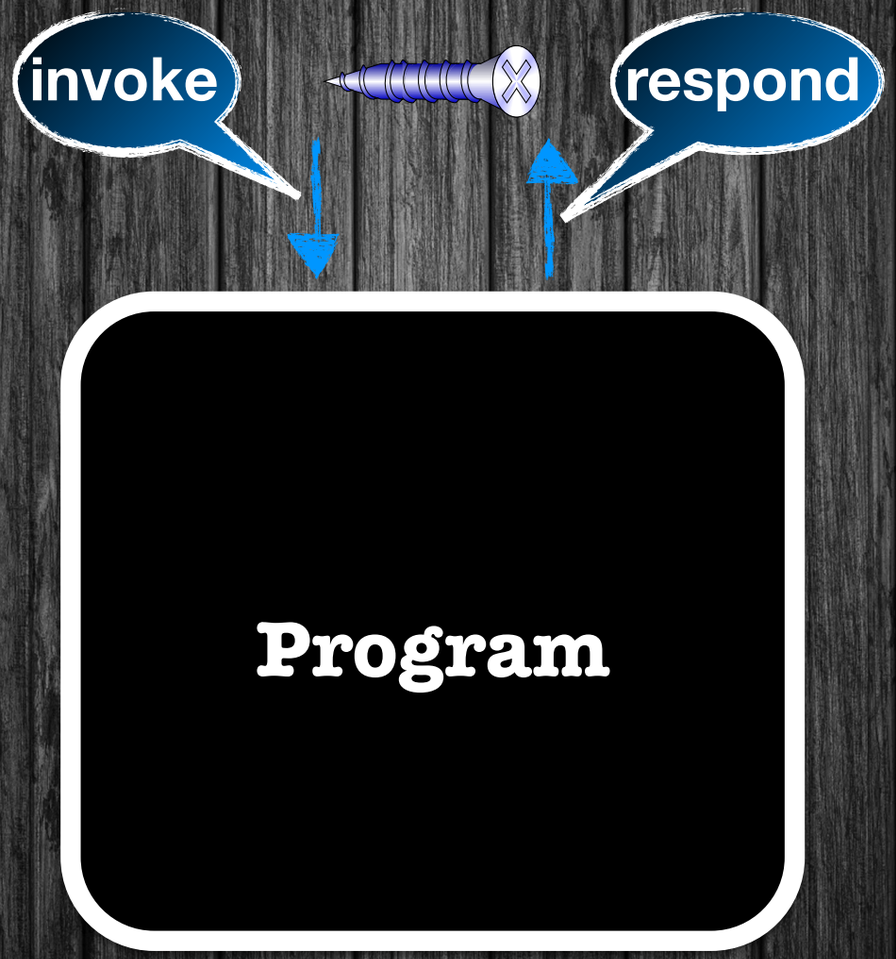
Concurrent Histories



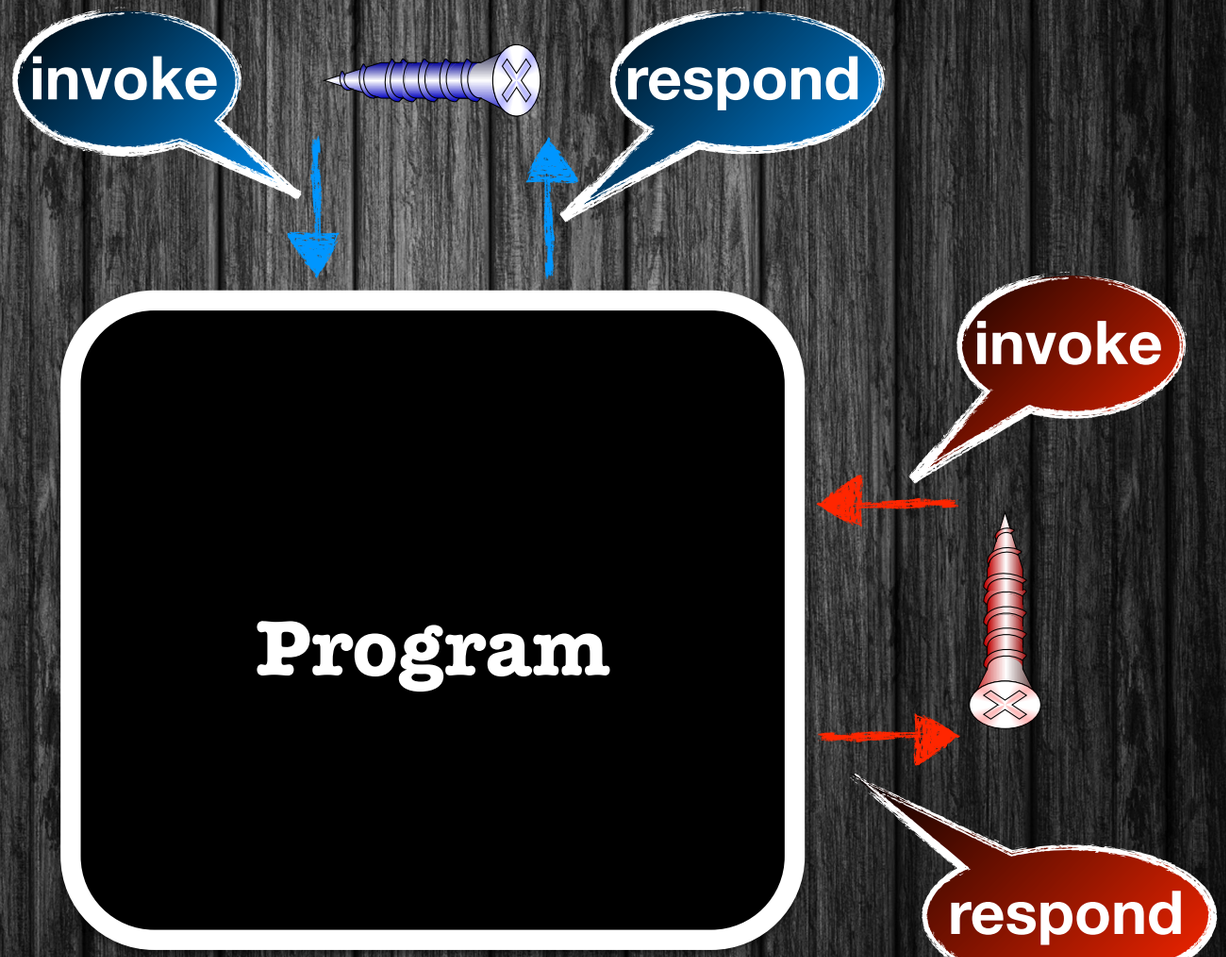


Concurrent Histories

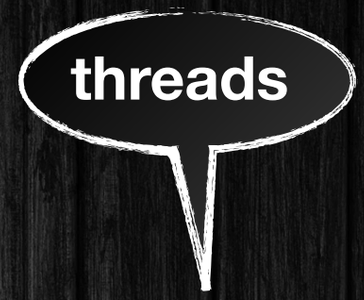




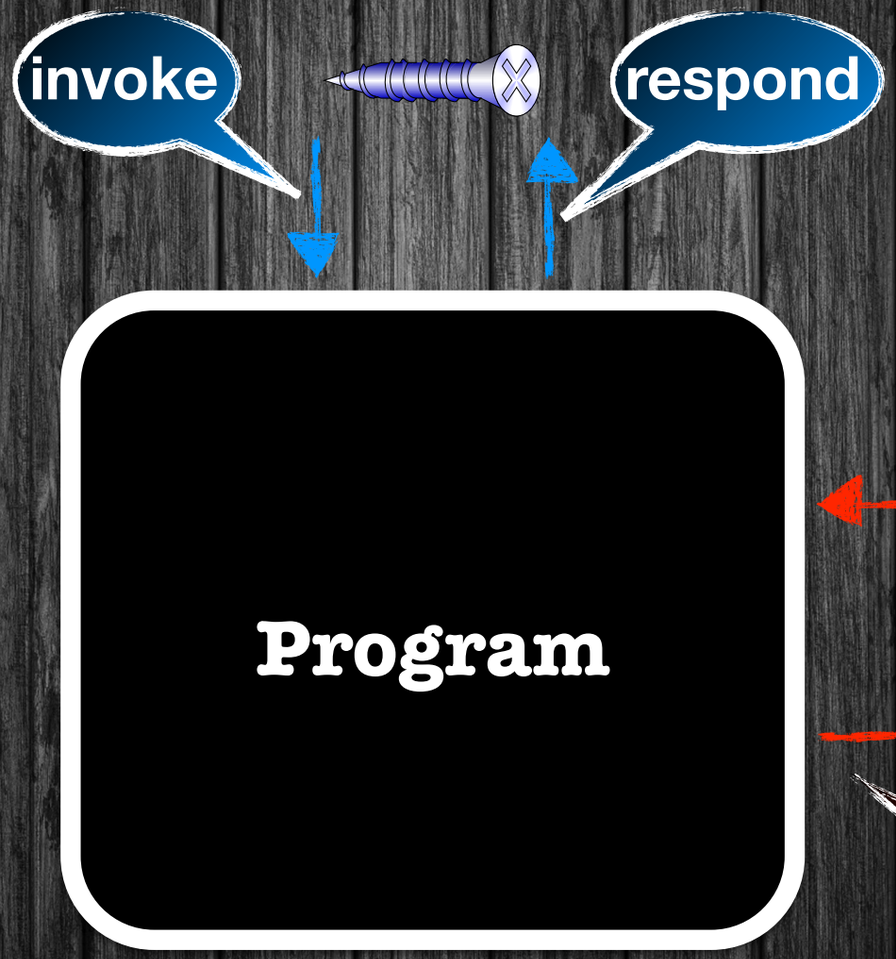
Concurrent Histories

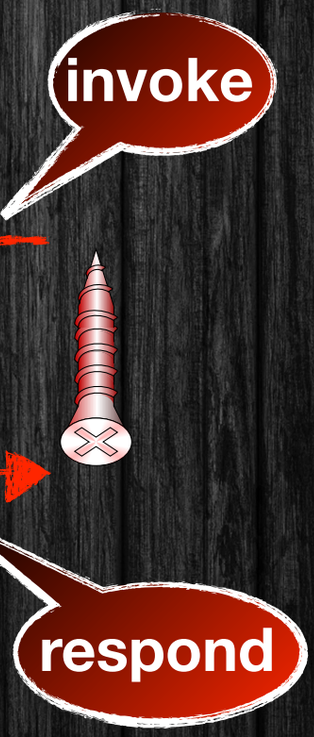


Concurrent Histories







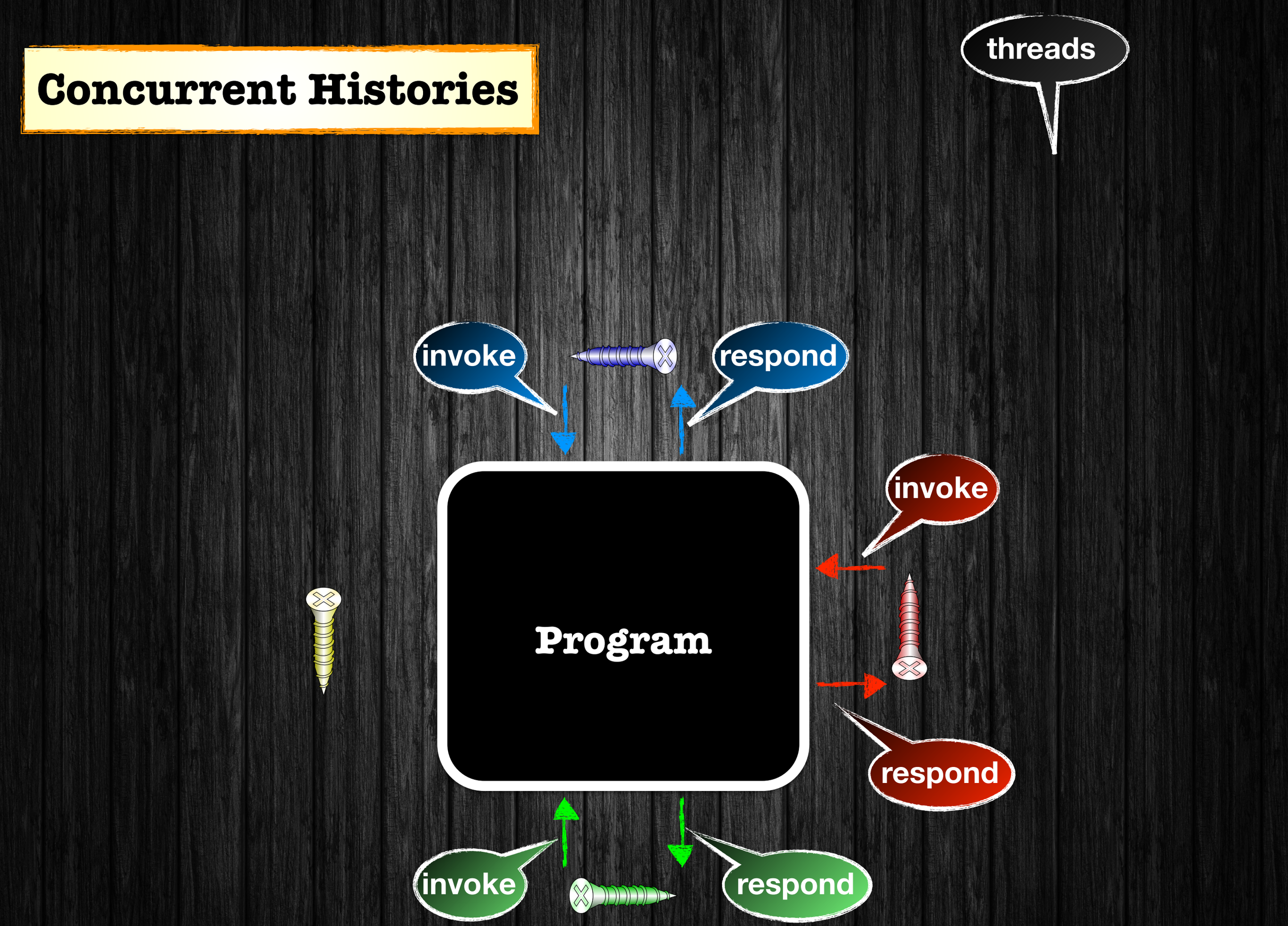




Concurrent Histories



Concurrent Histories



Concurrent Histories

threads



Concurrent Histories

threads

the threads operate concurrently



Concurrent Histories

threads

the threads operate concurrently

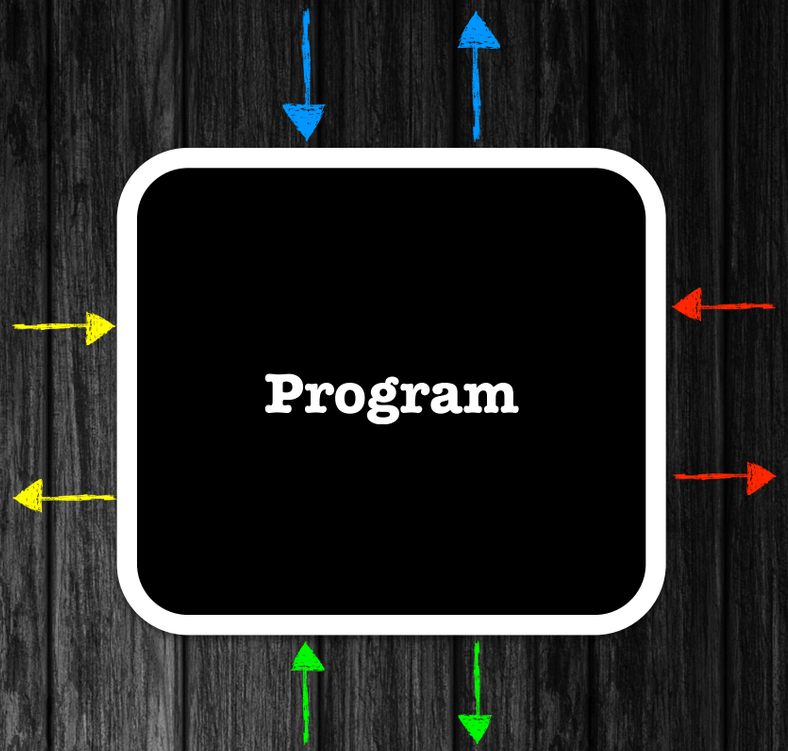
calls may overlap



Concurrent Histories

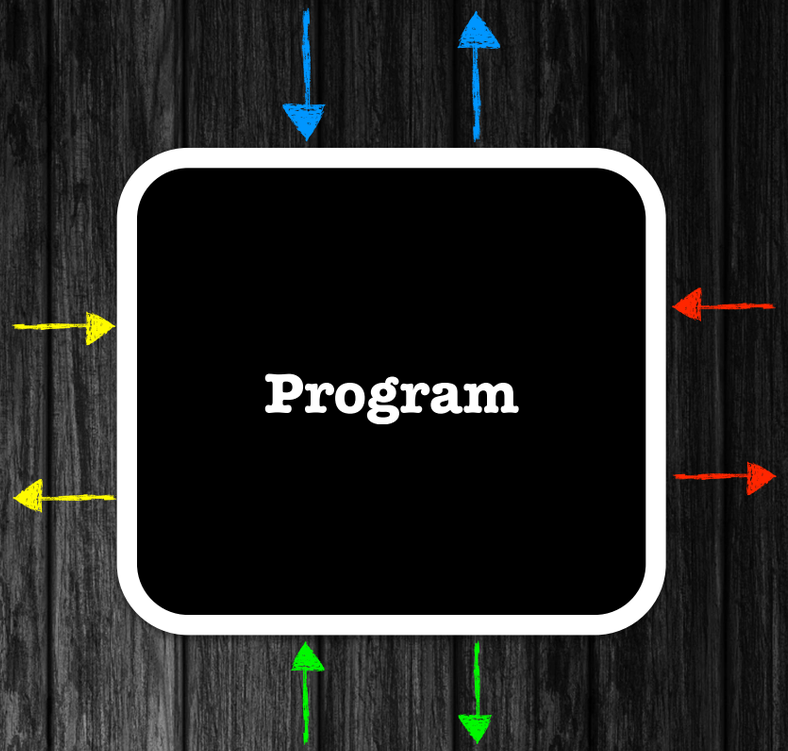


Concurrent Histories Sets



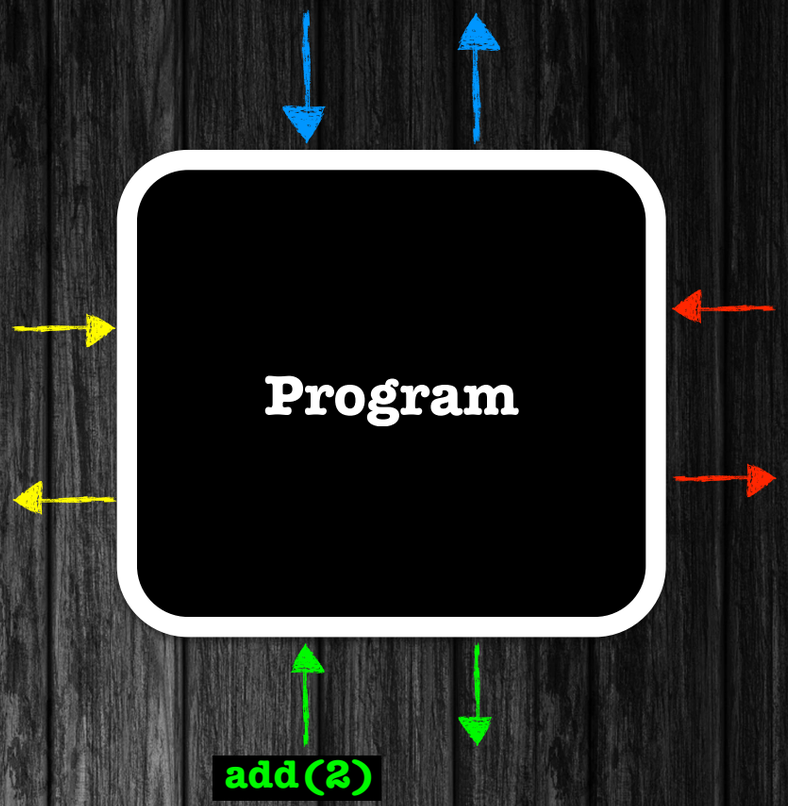
Time

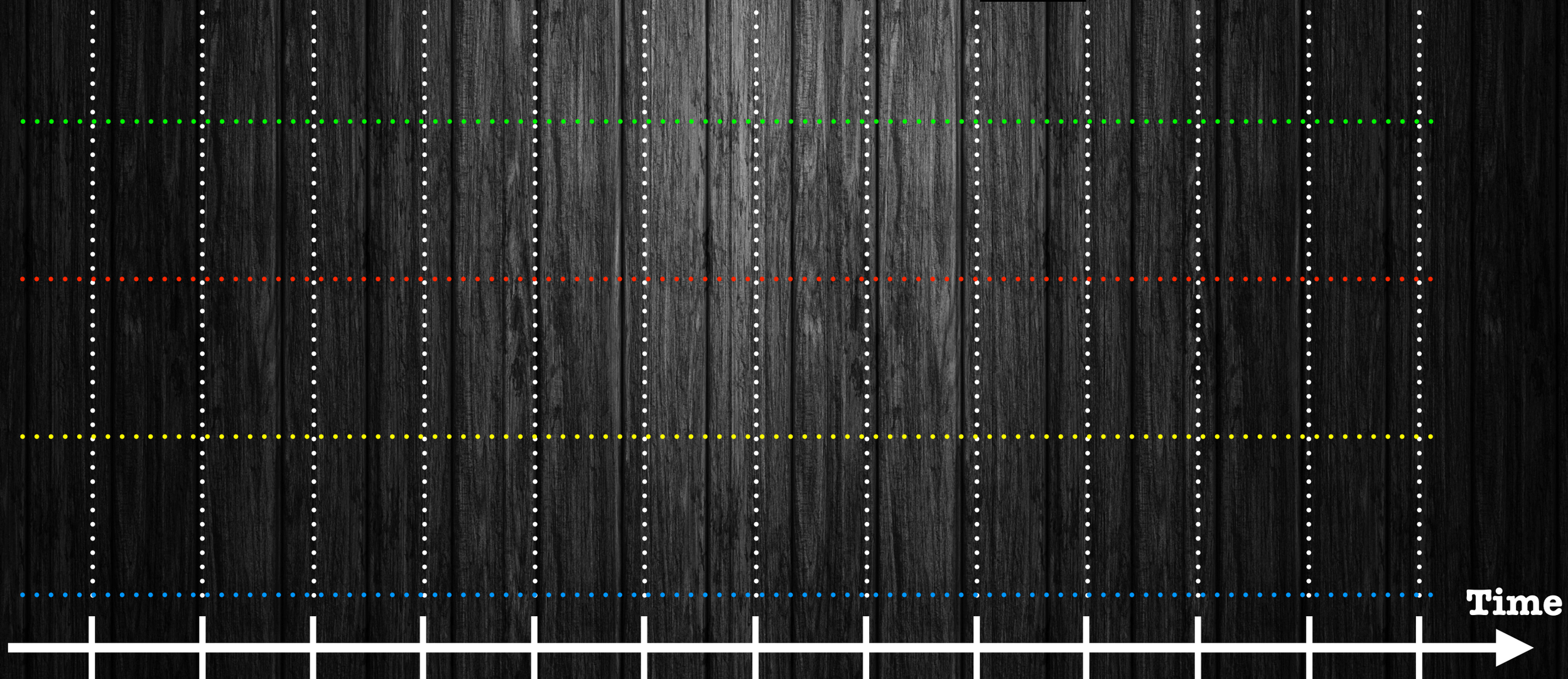
Concurrent Histories Sets



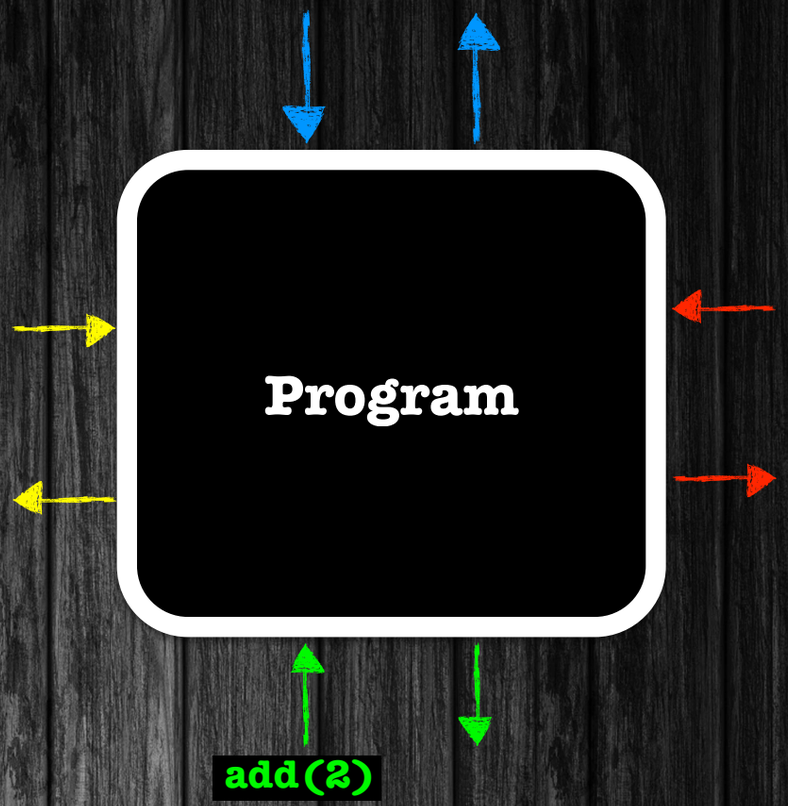
Time

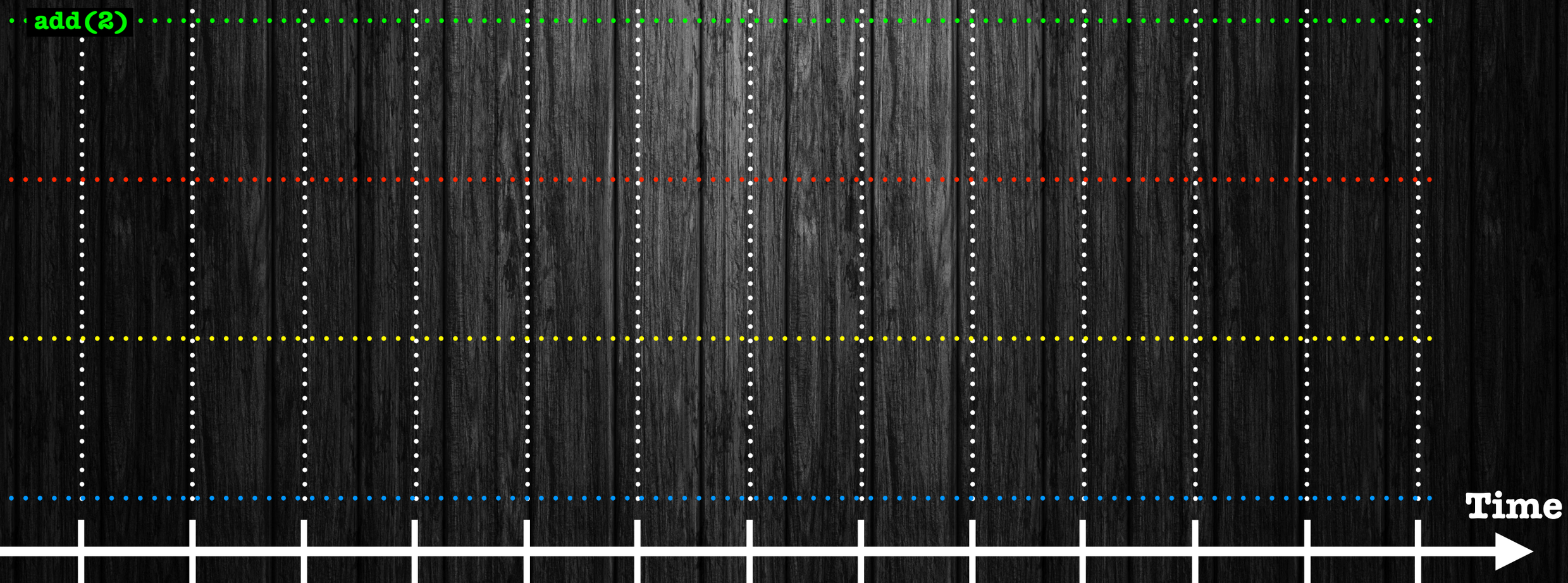
Concurrent Histories Sets



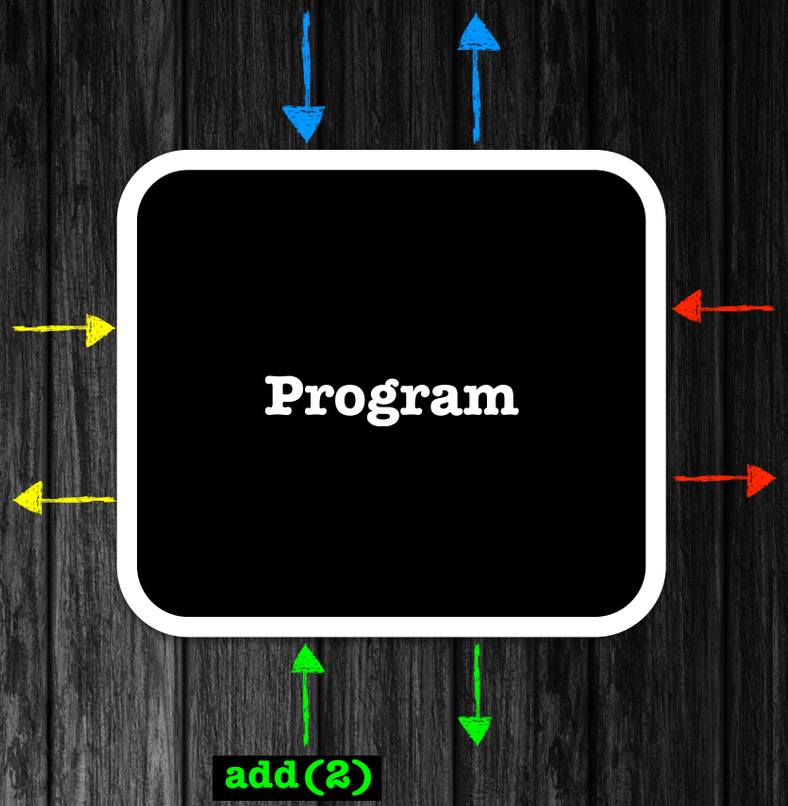


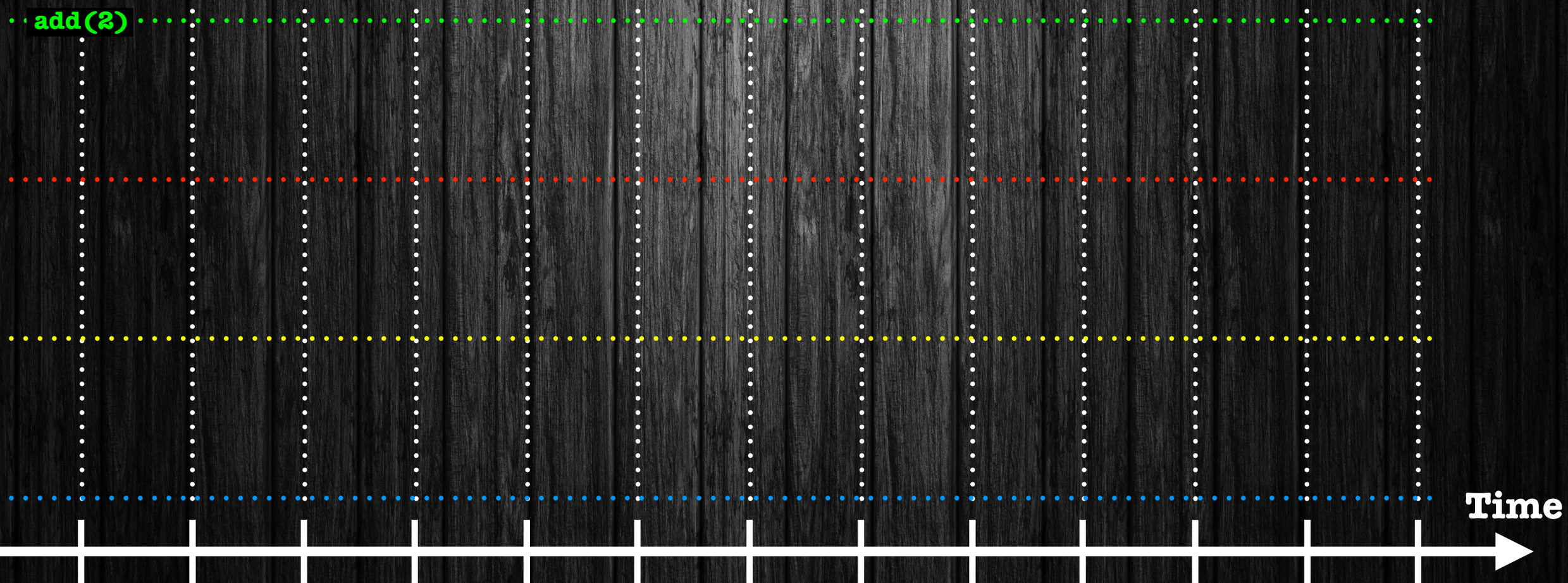
Concurrent Histories Sets



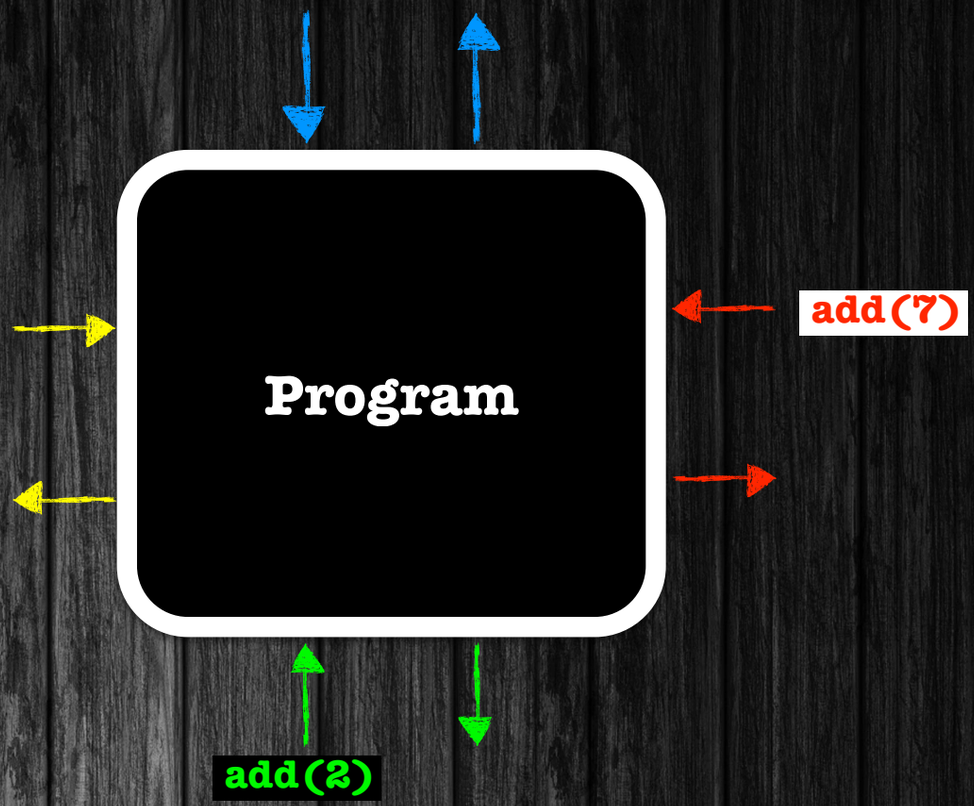


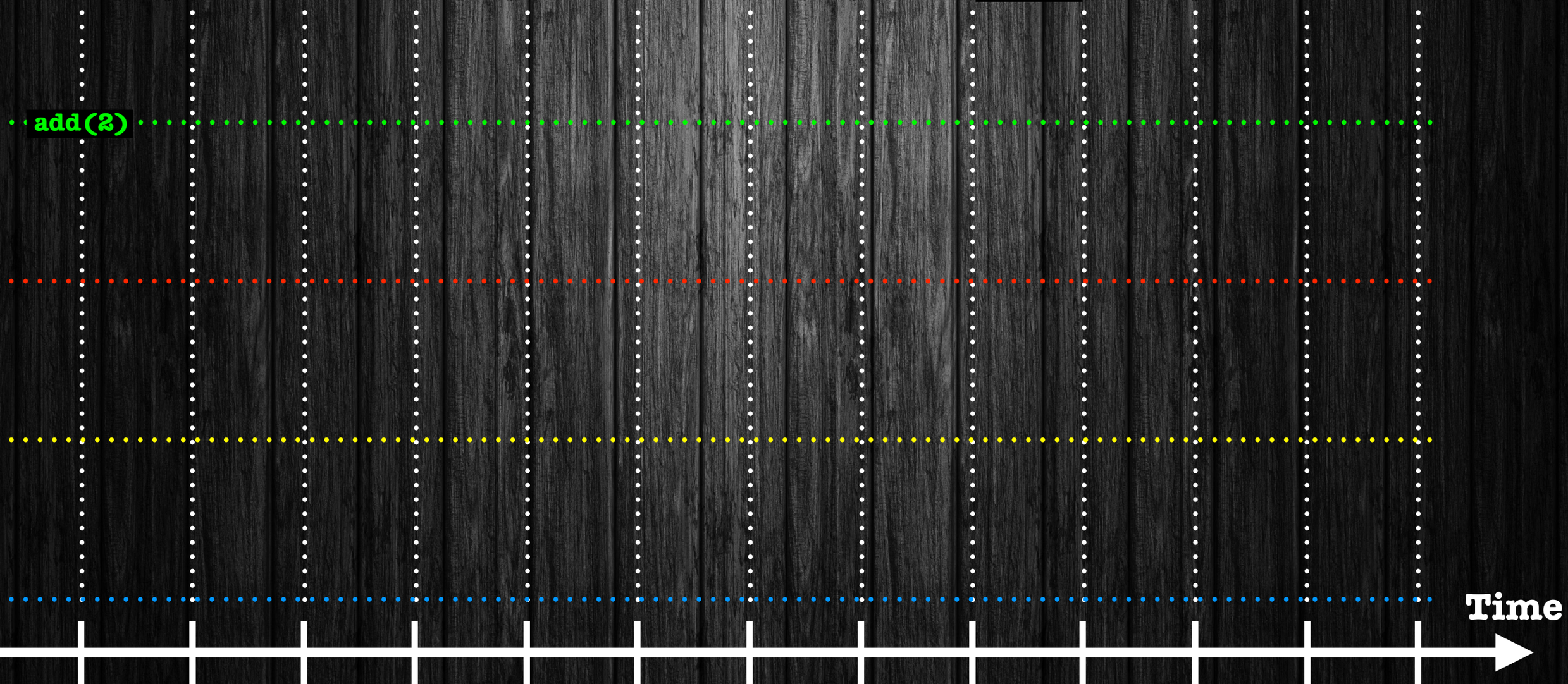
Concurrent Histories Sets



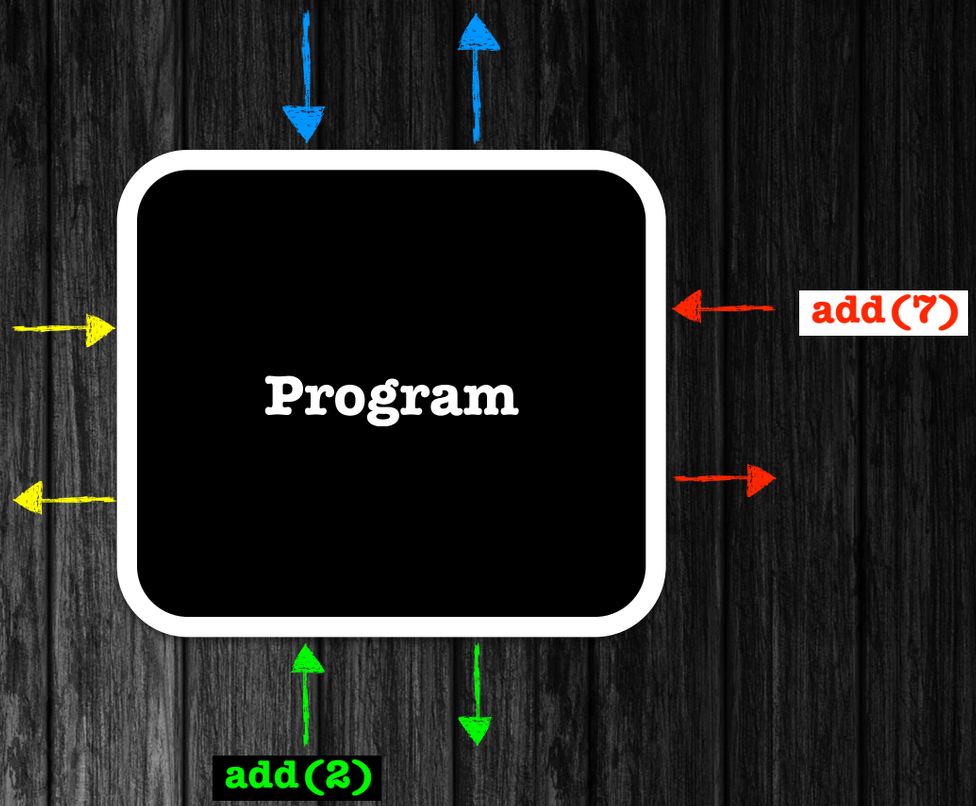


Concurrent Histories Sets





Concurrent Histories Sets

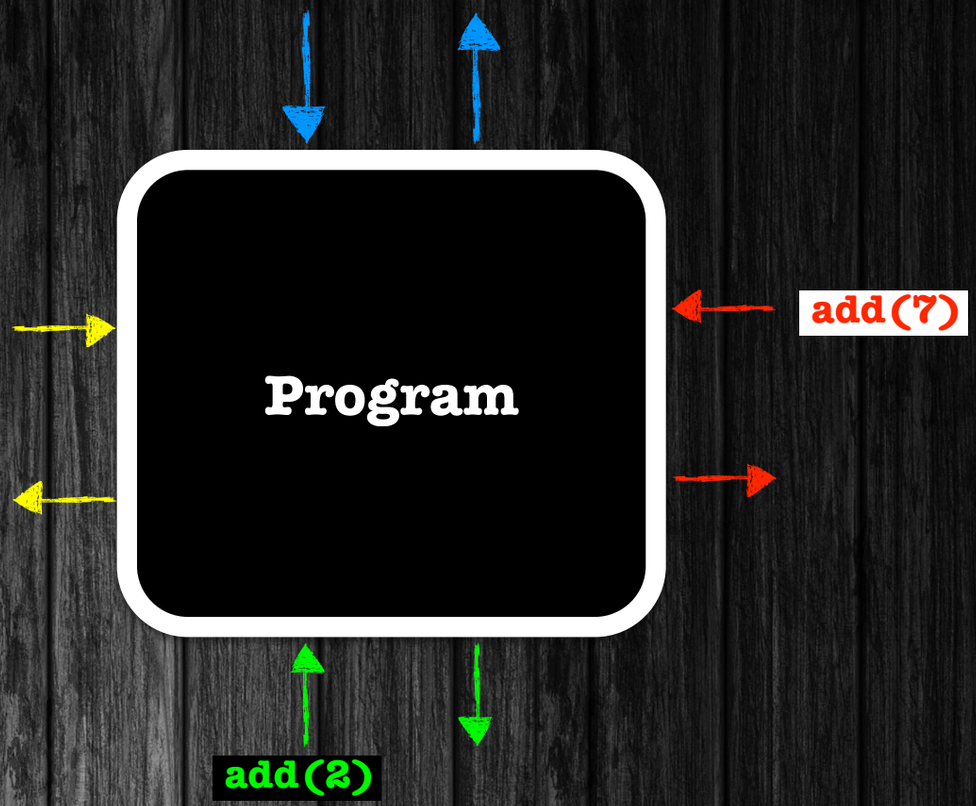


add(2)

add(7)

Time

Concurrent Histories Sets

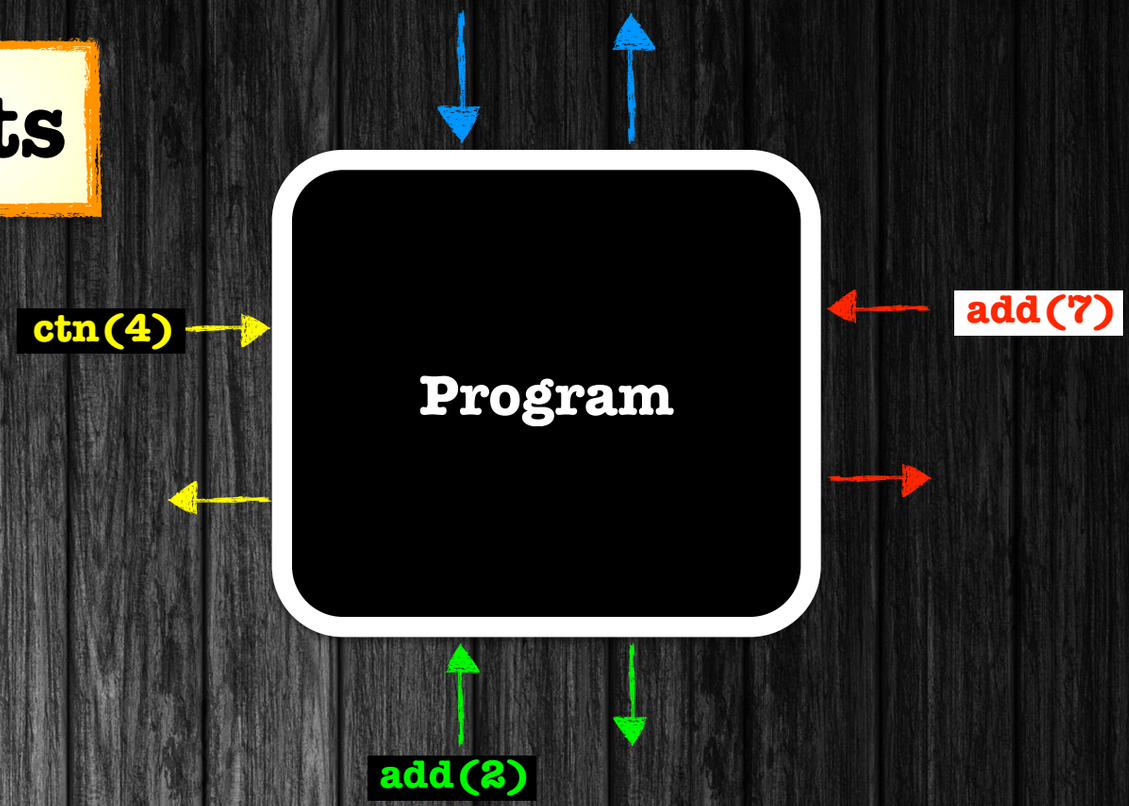


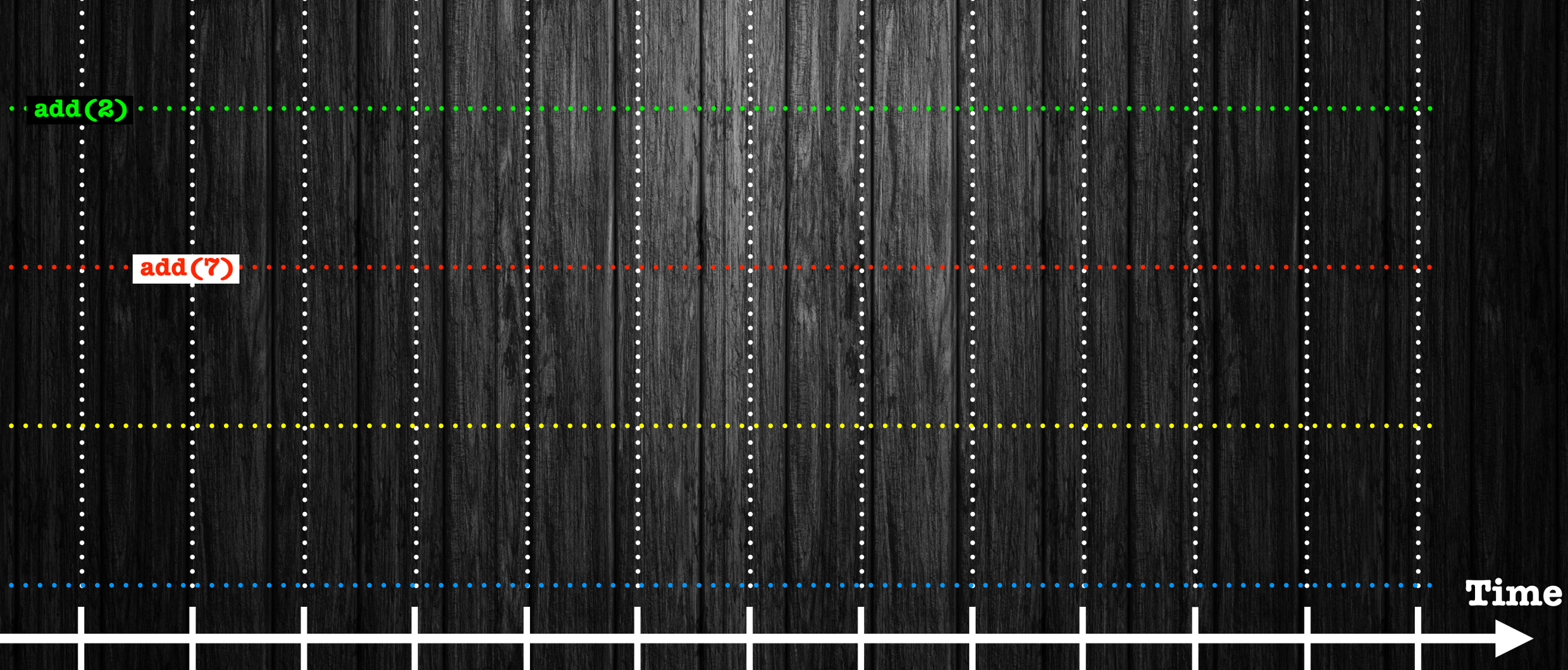
add(2)

add(7)

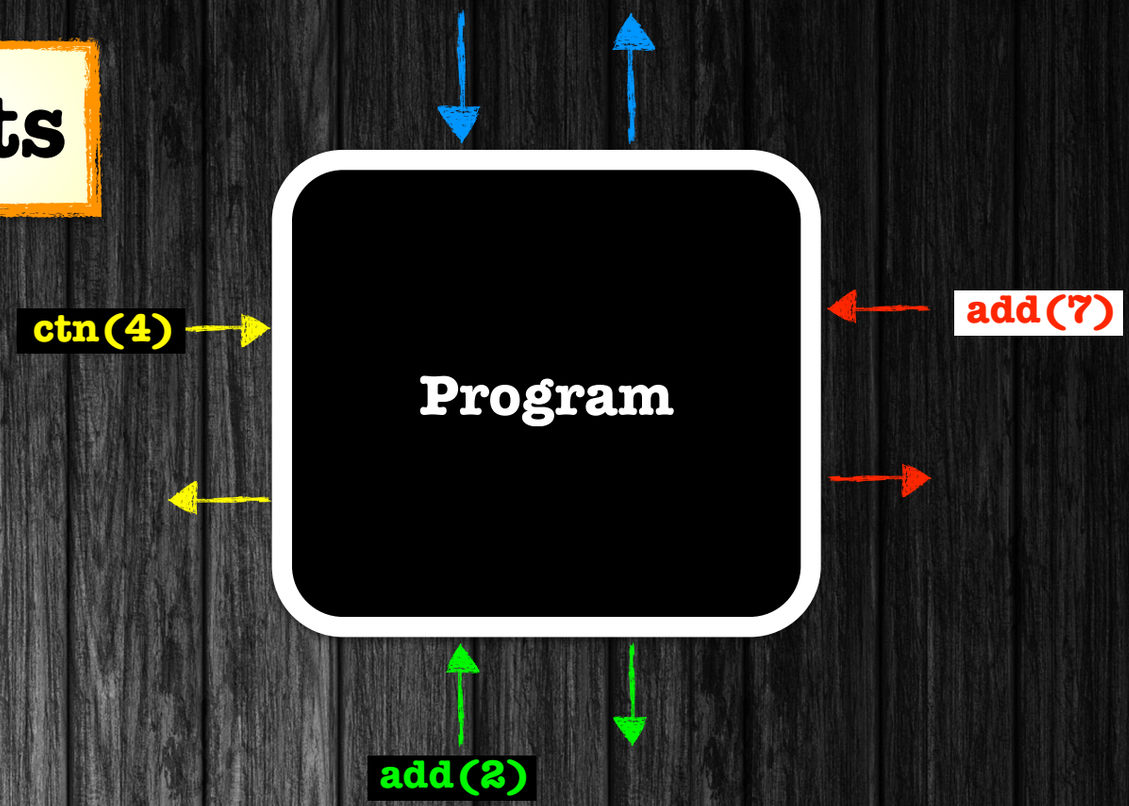
Time

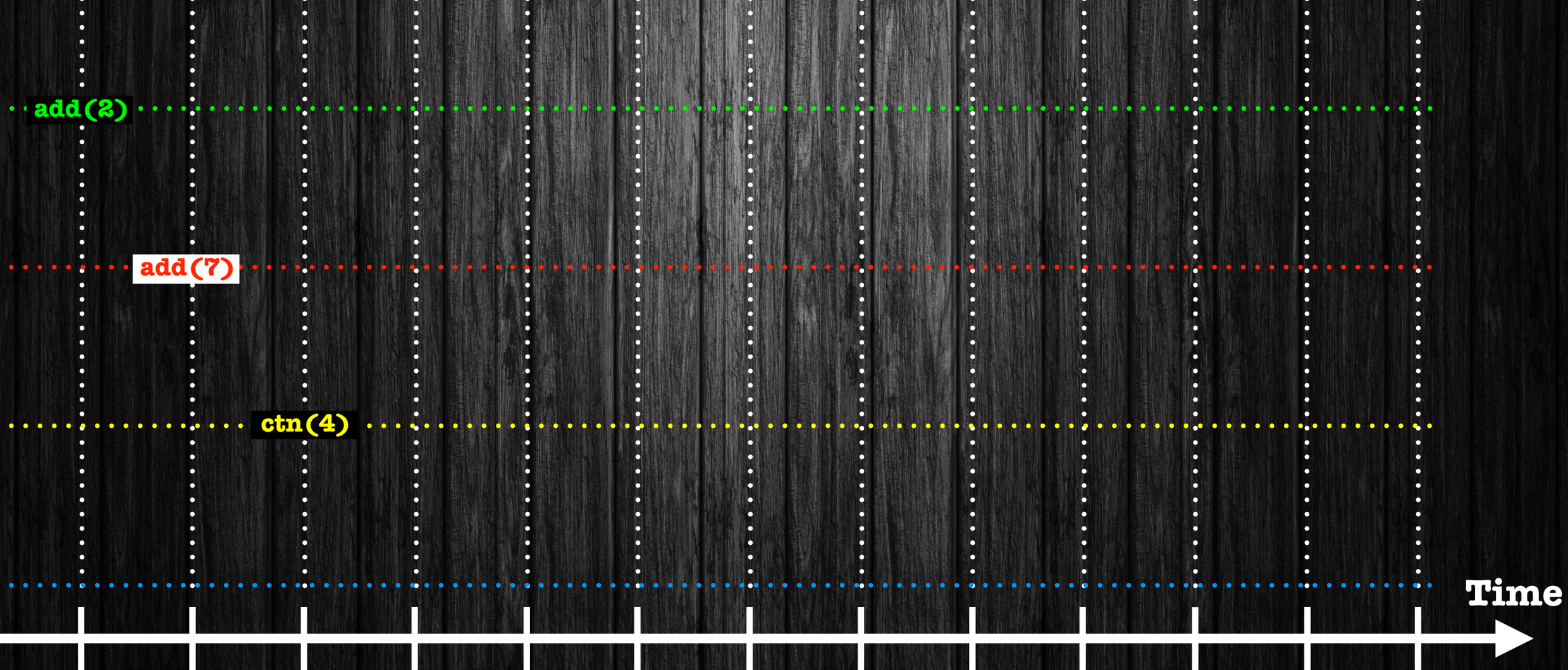
Concurrent Histories Sets



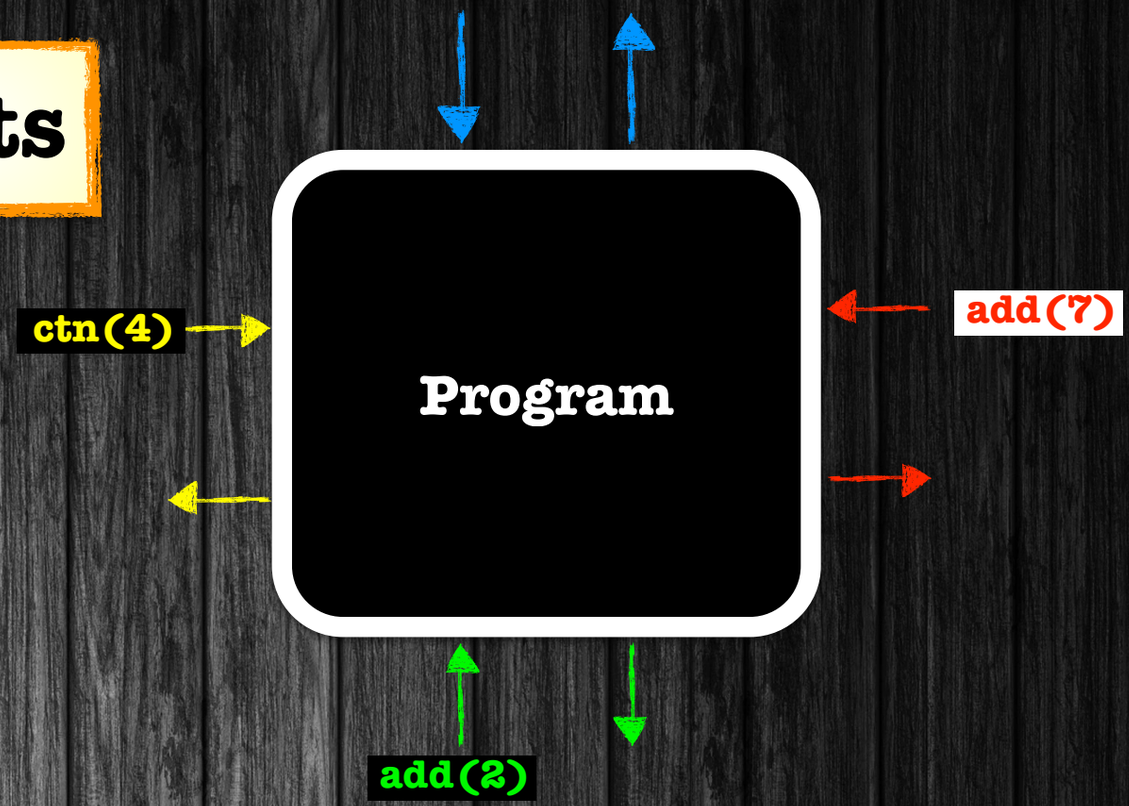


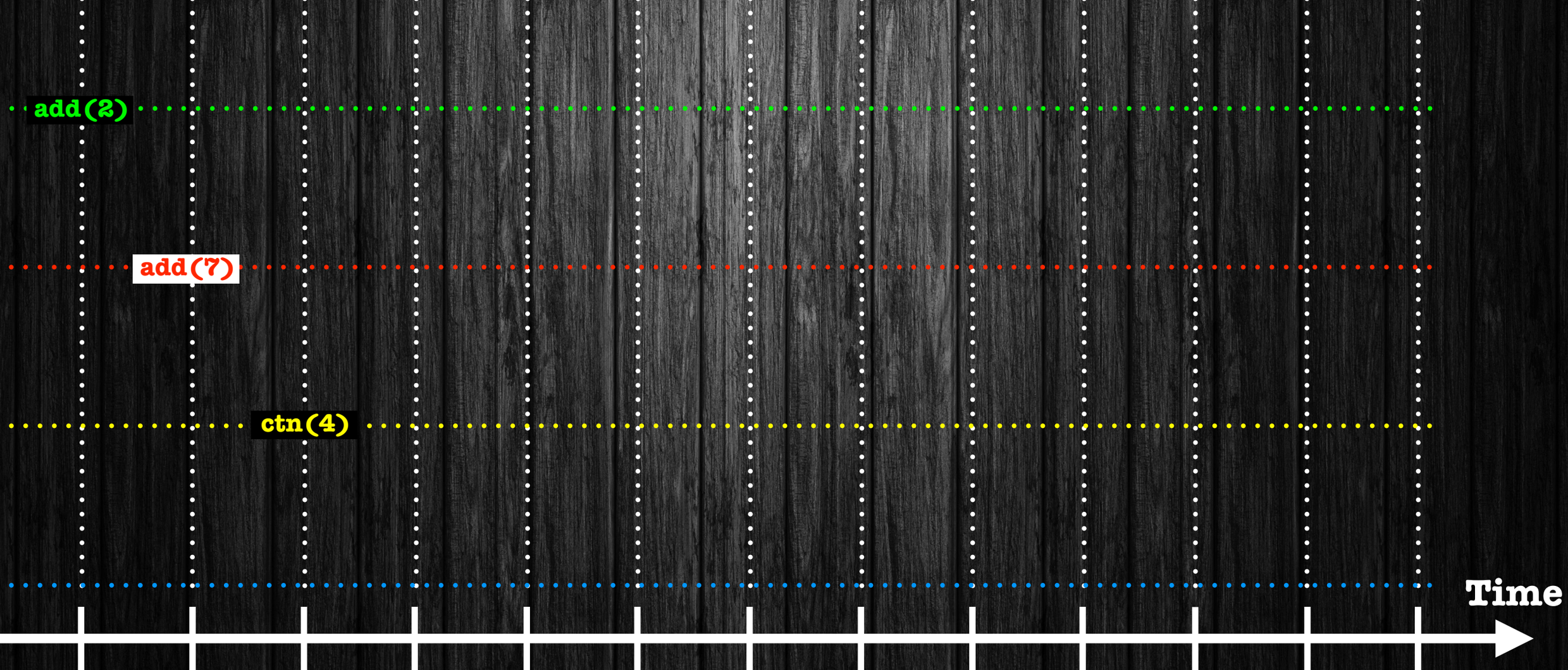
Concurrent Histories Sets



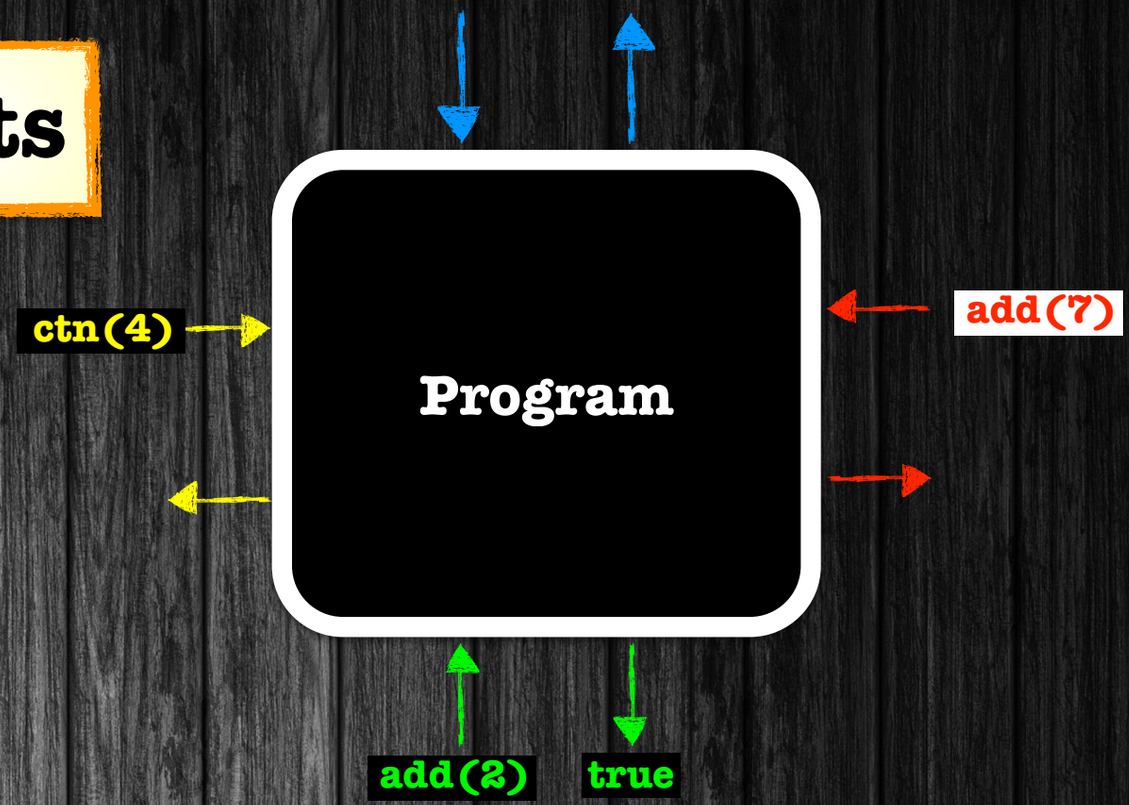


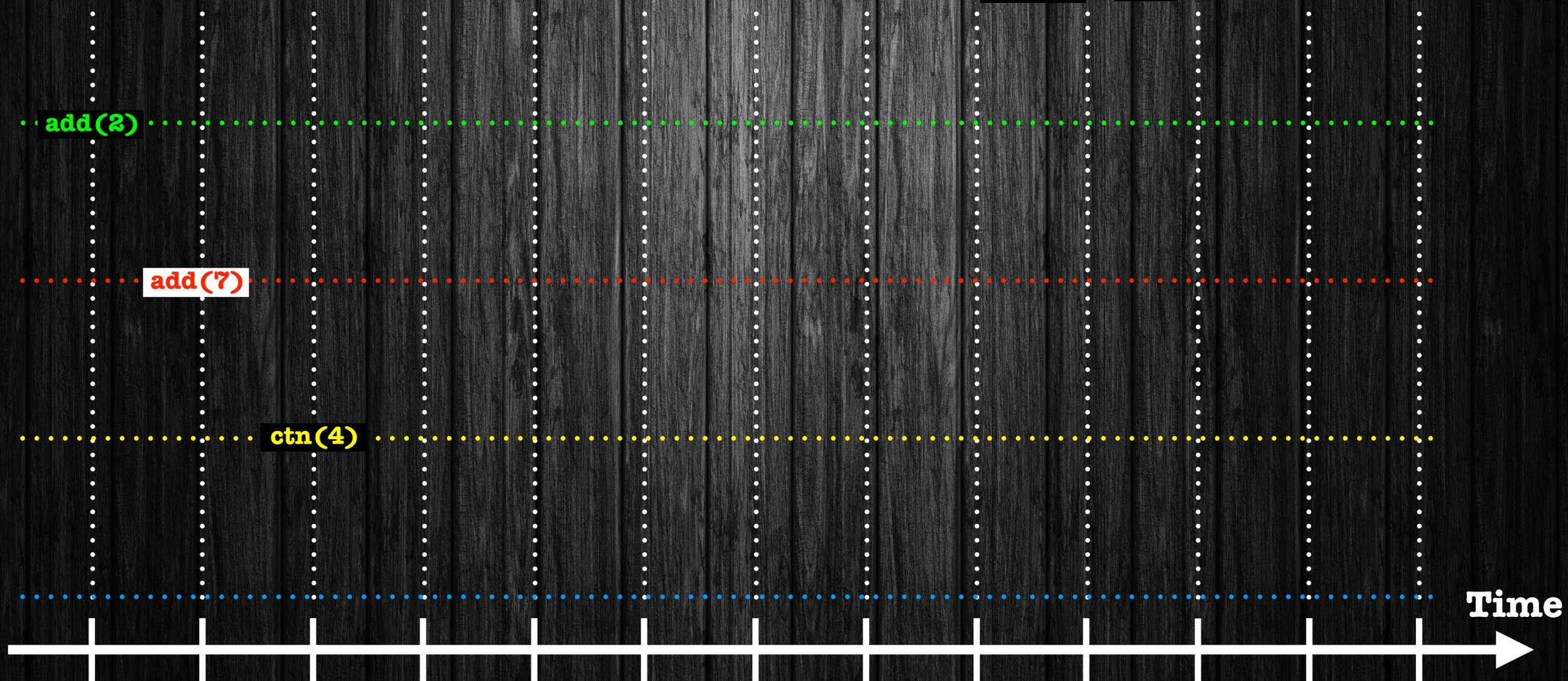
Concurrent Histories Sets



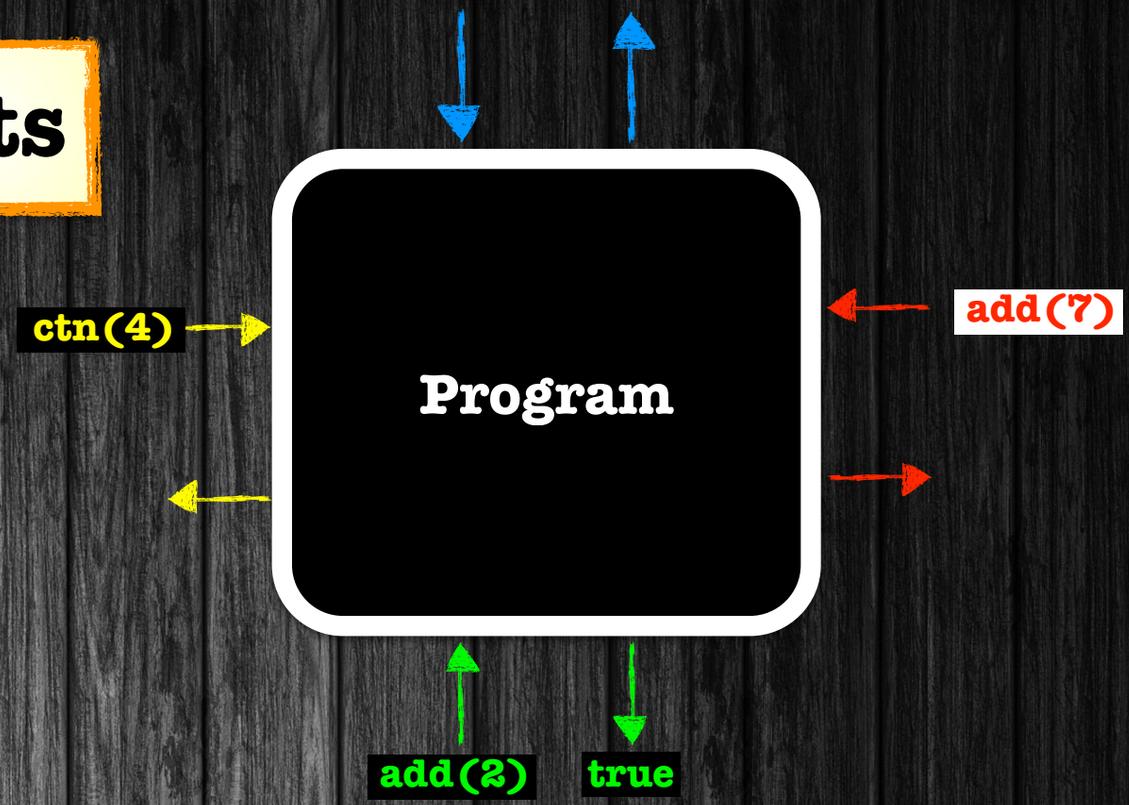


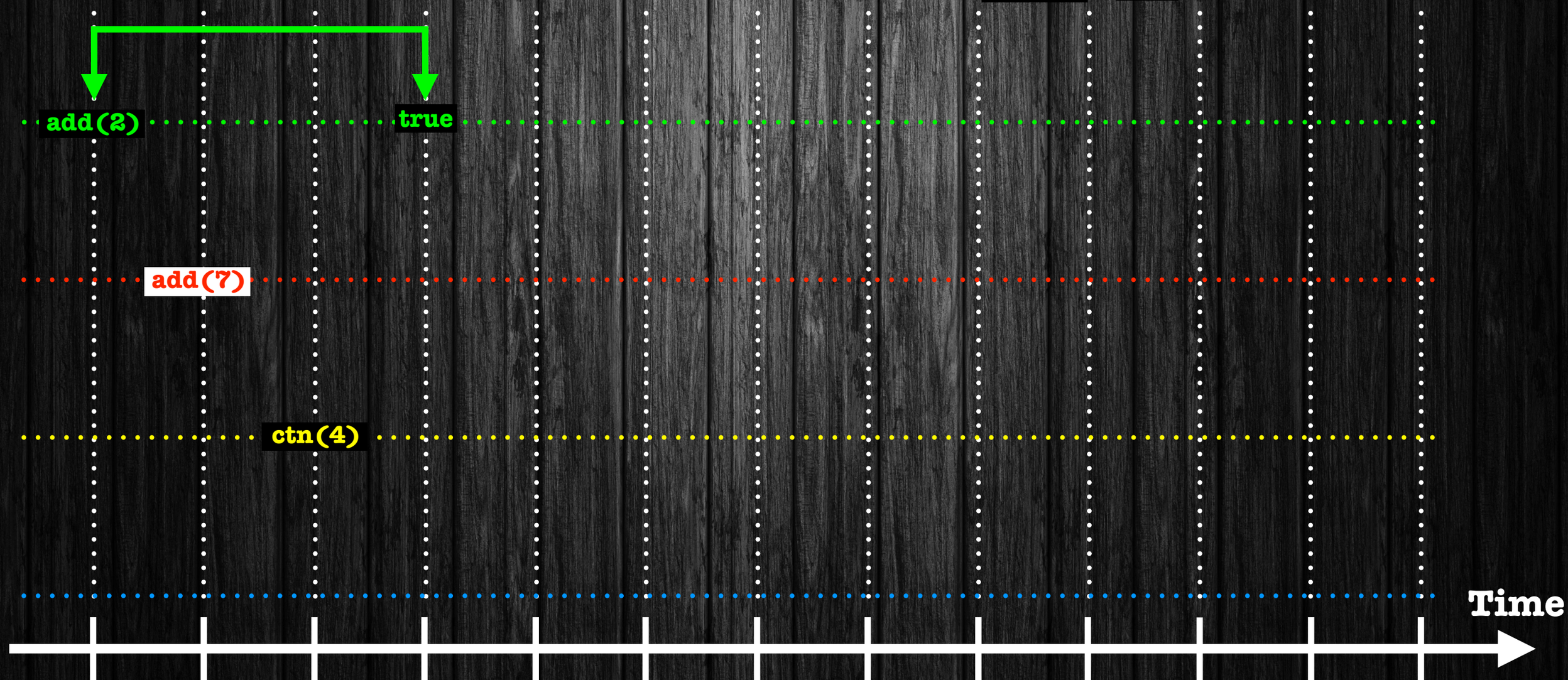
Concurrent Histories Sets



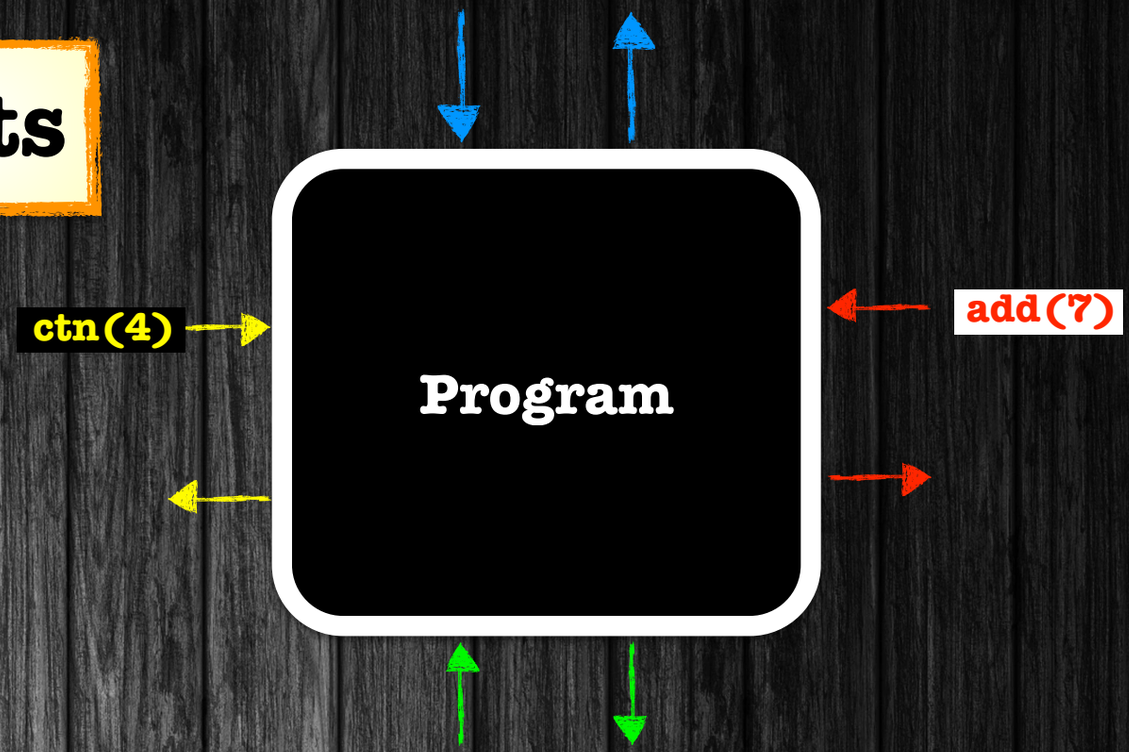


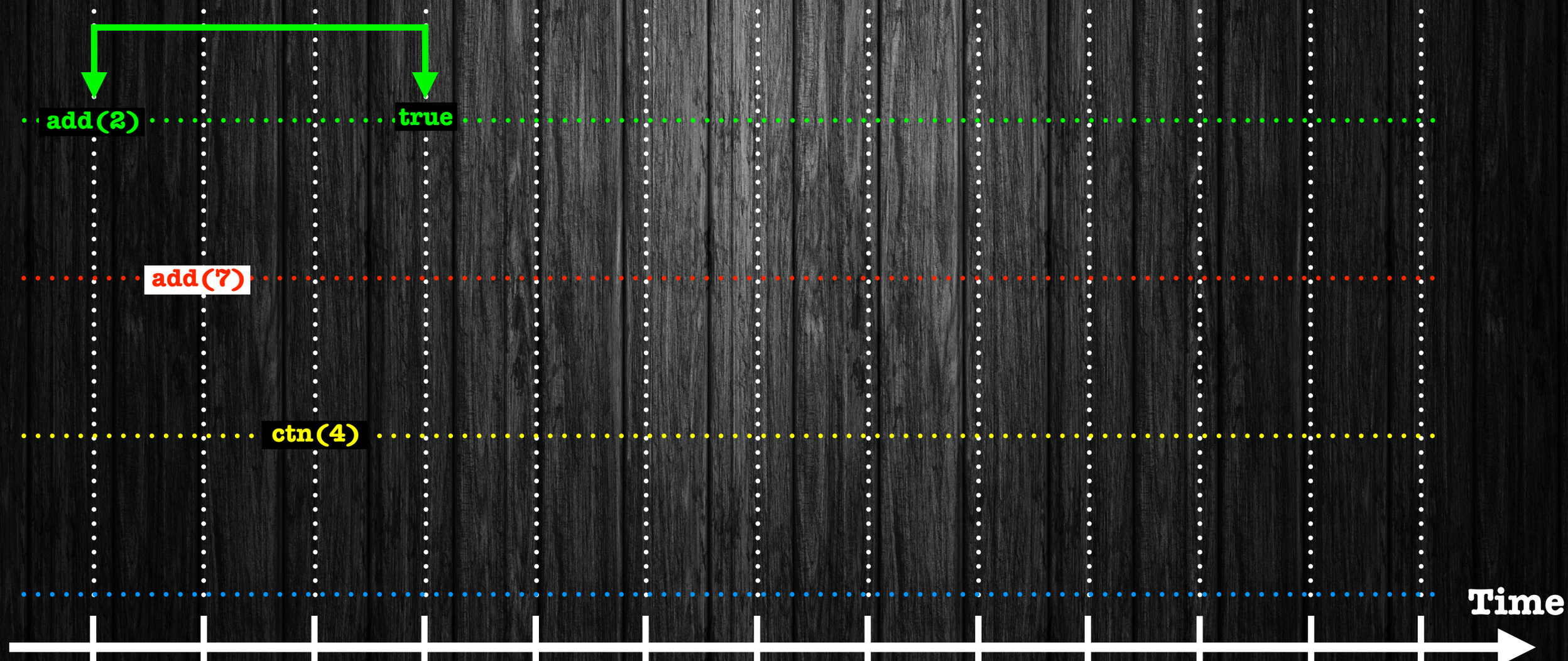
Concurrent Histories Sets



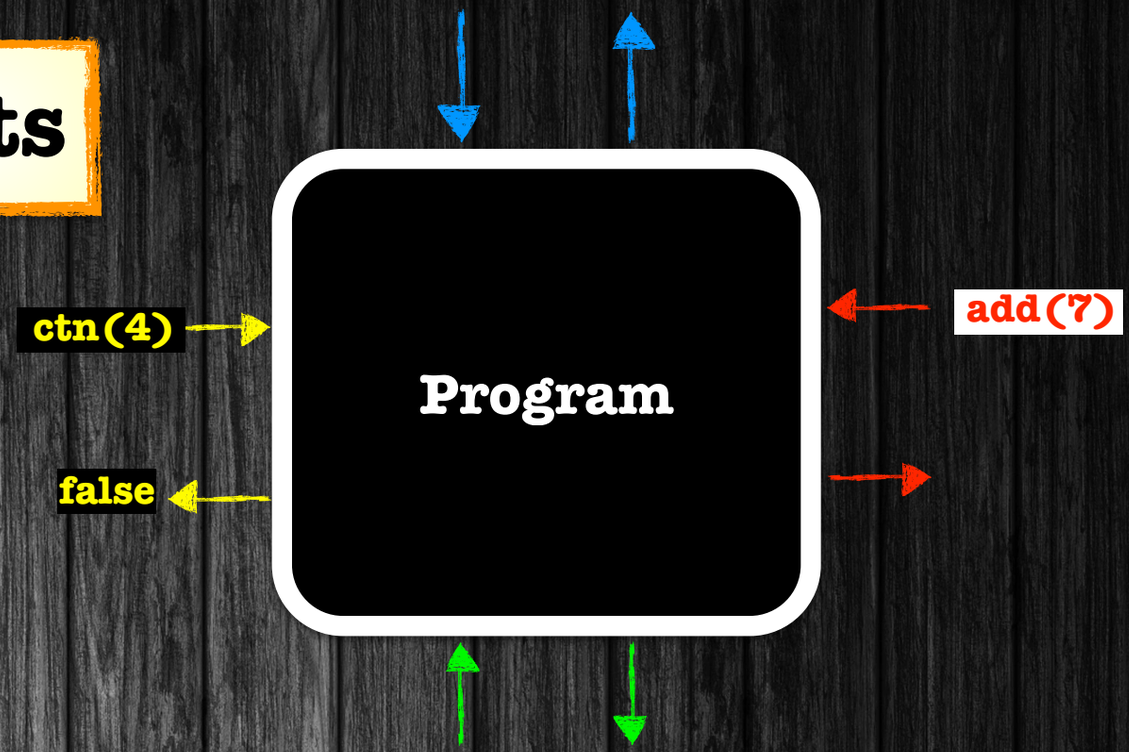


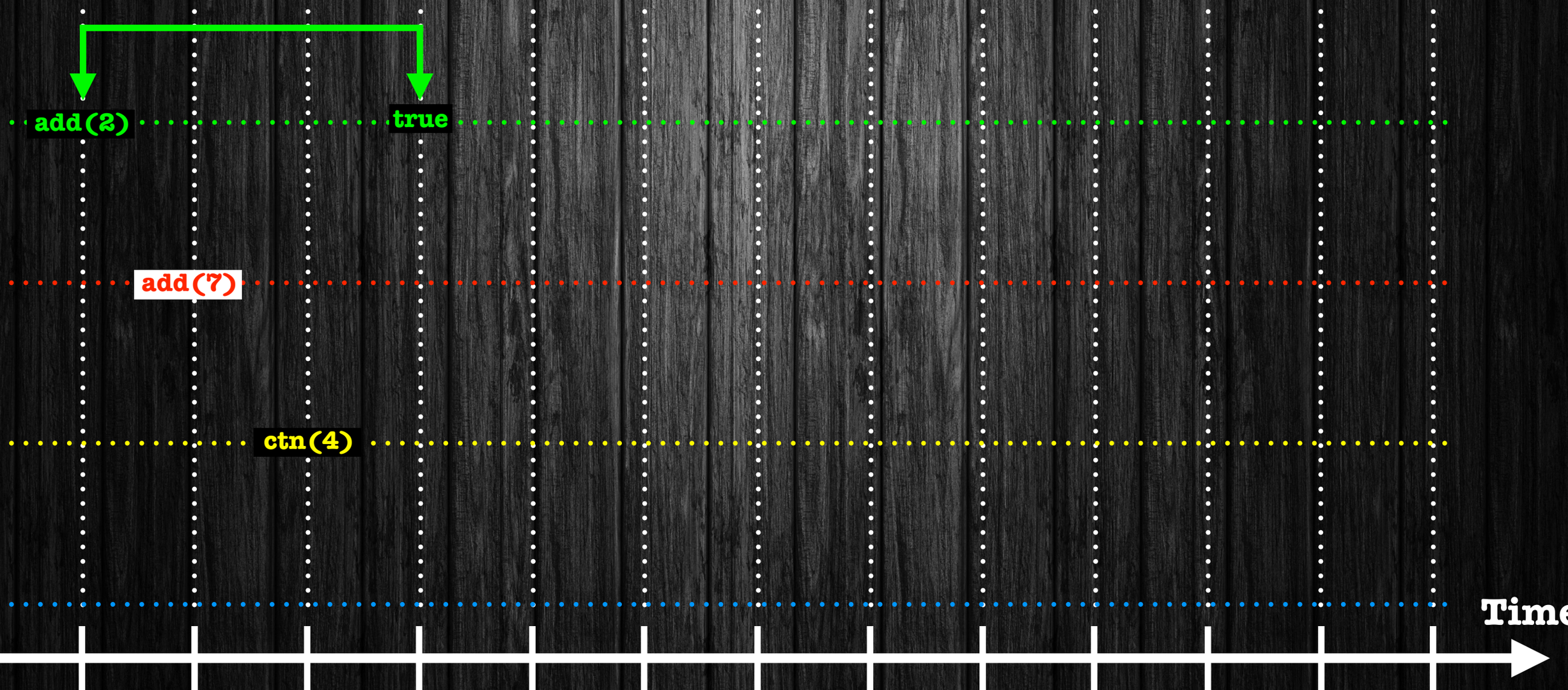
Concurrent Histories Sets



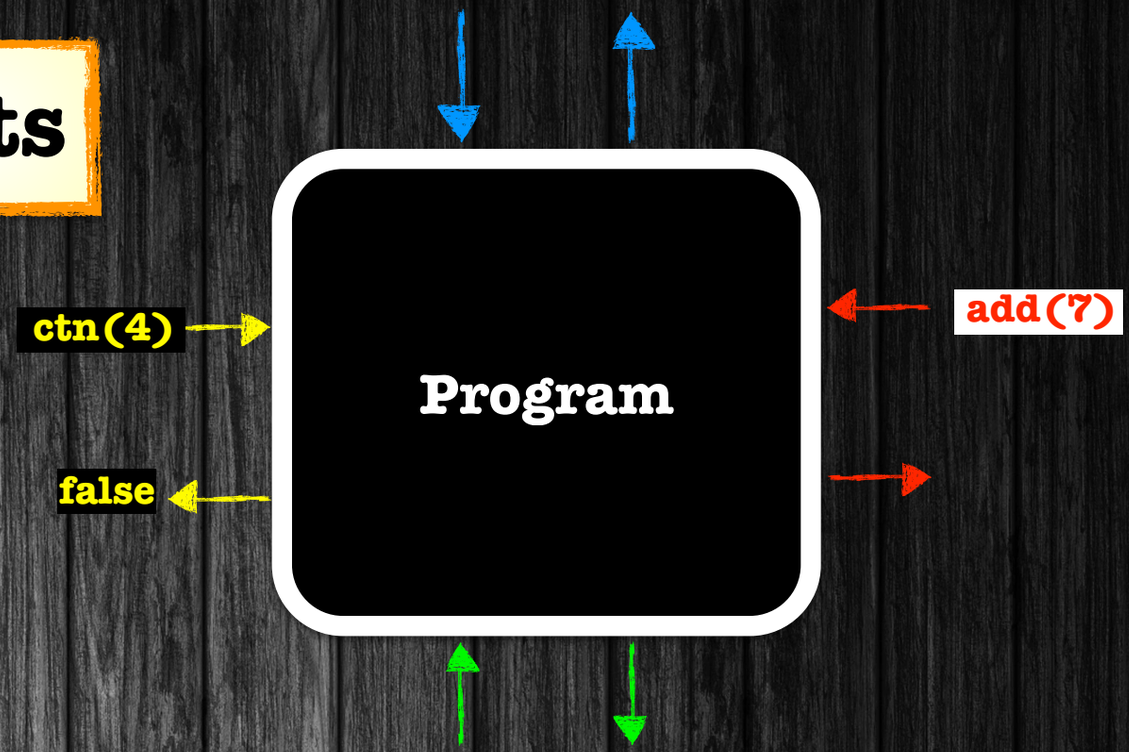


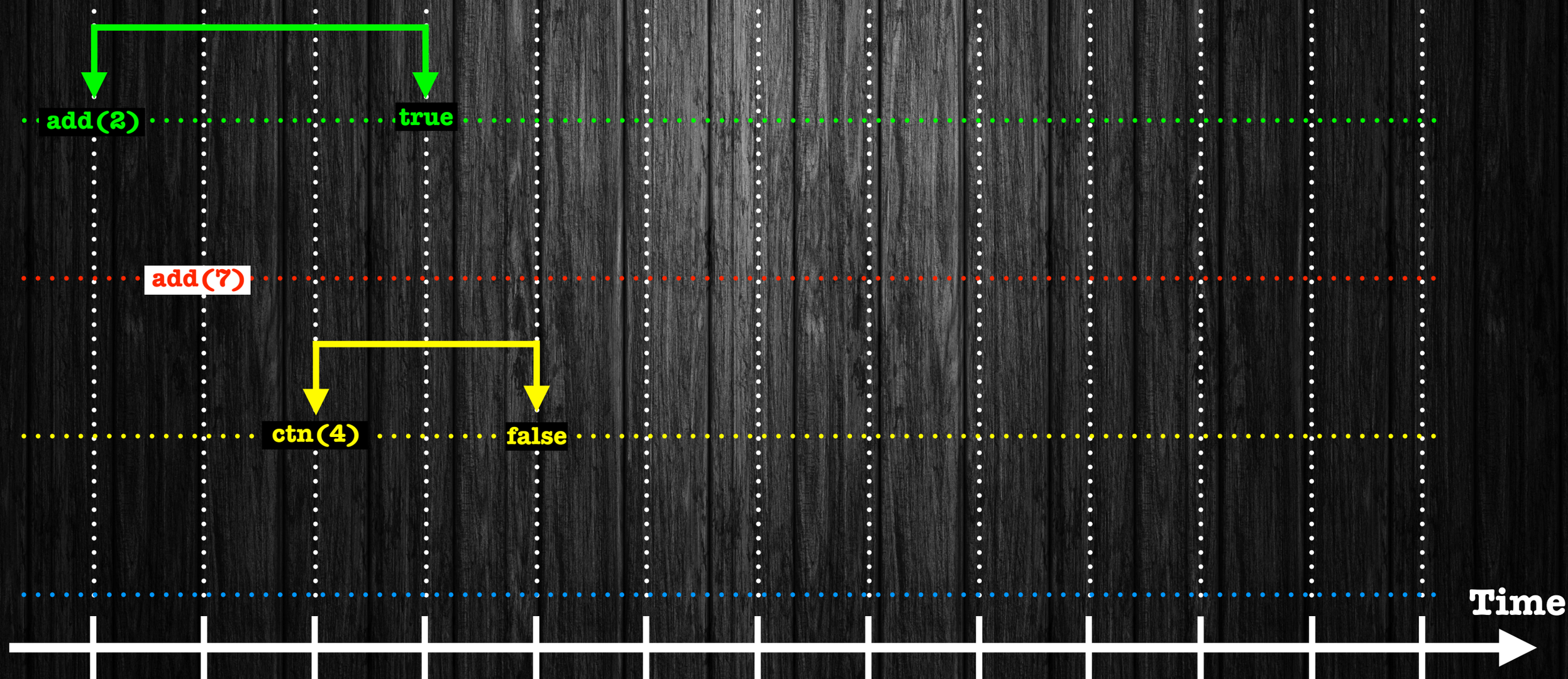
Concurrent Histories Sets



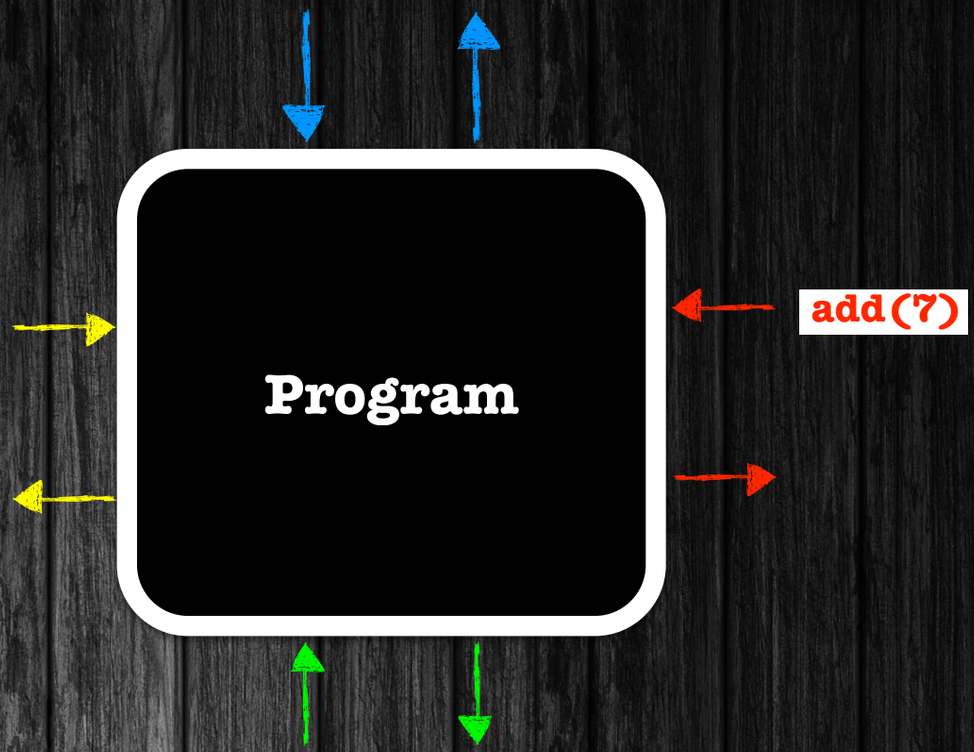


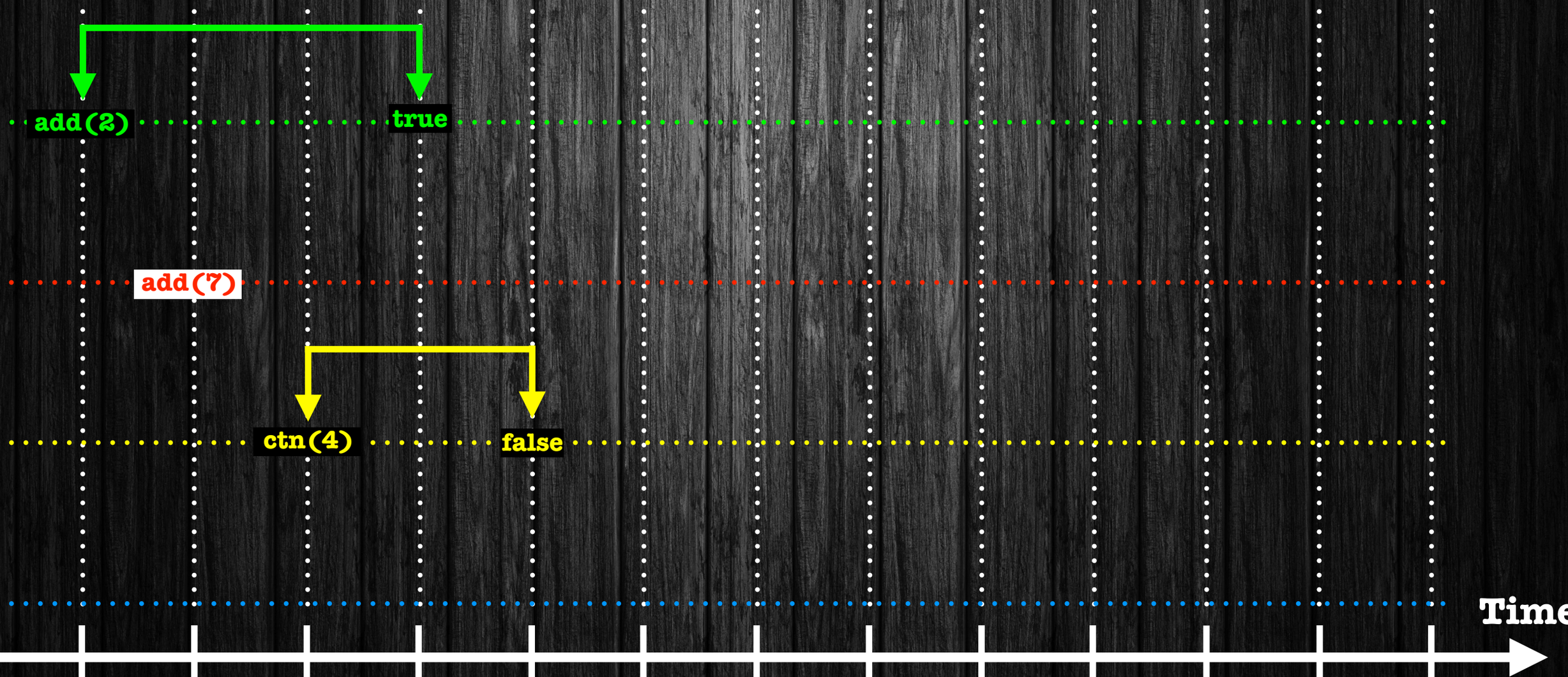
Concurrent Histories Sets



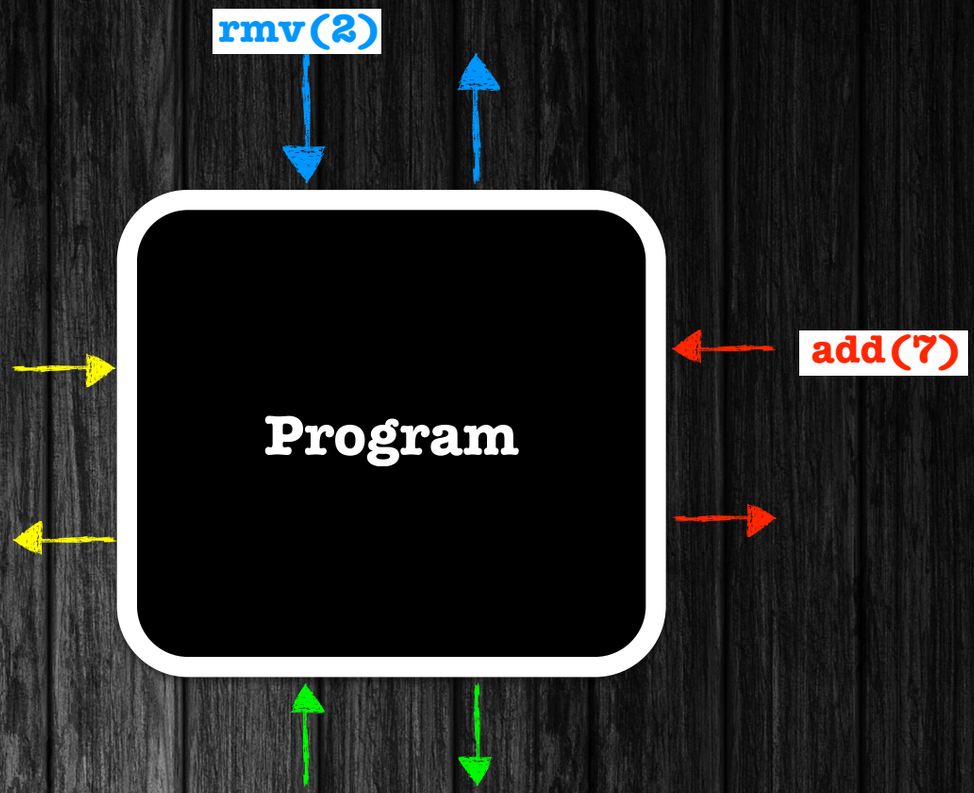


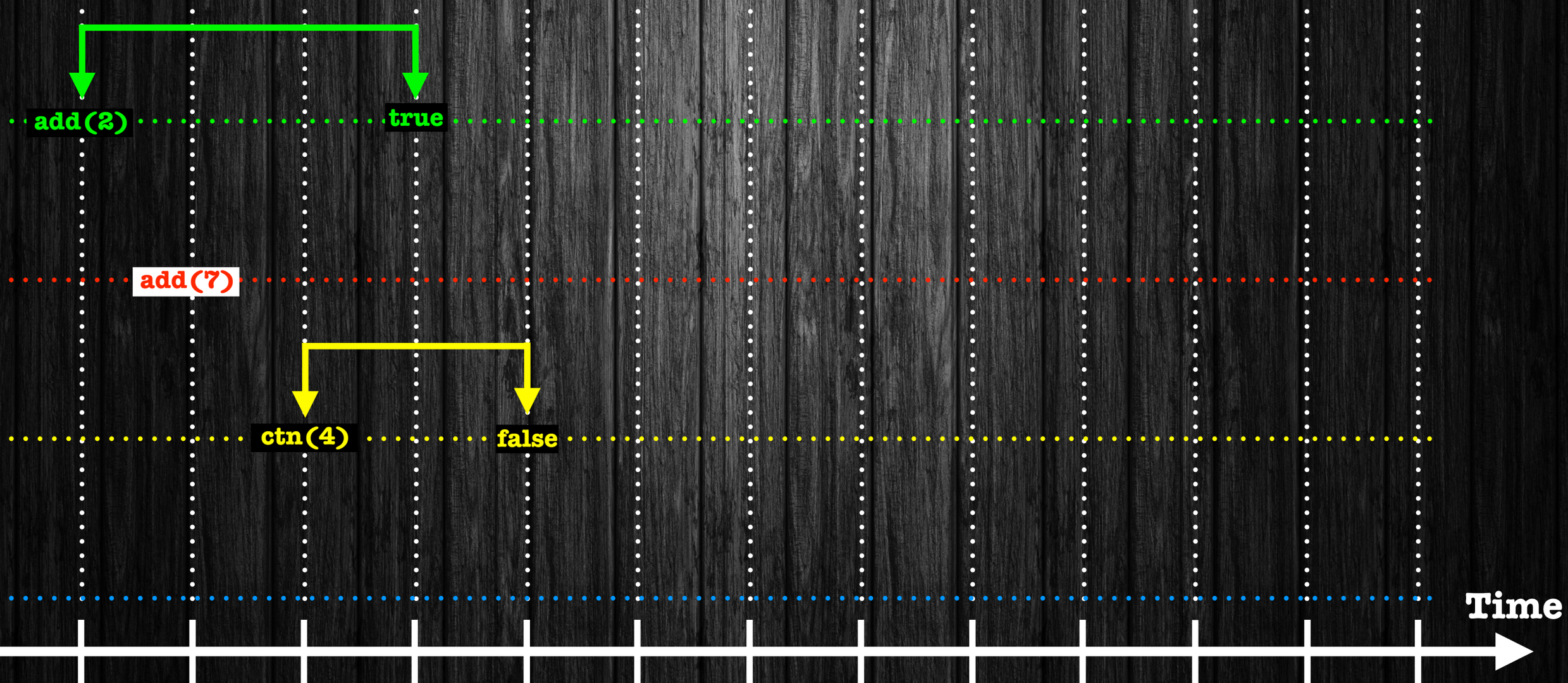
Concurrent Histories Sets



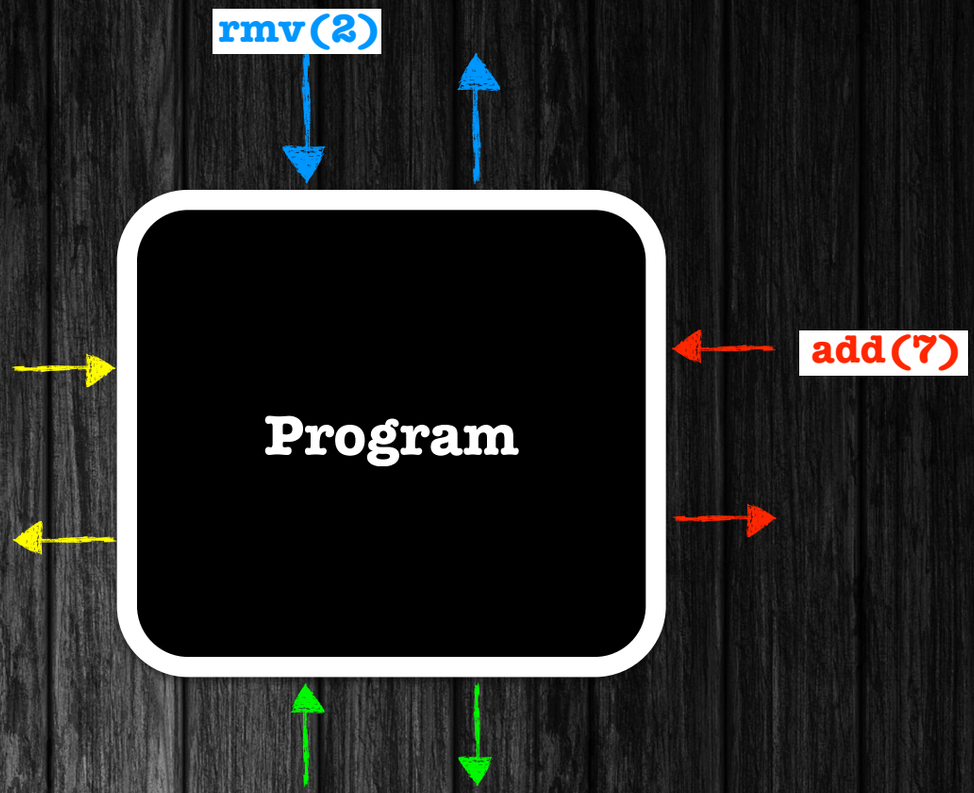


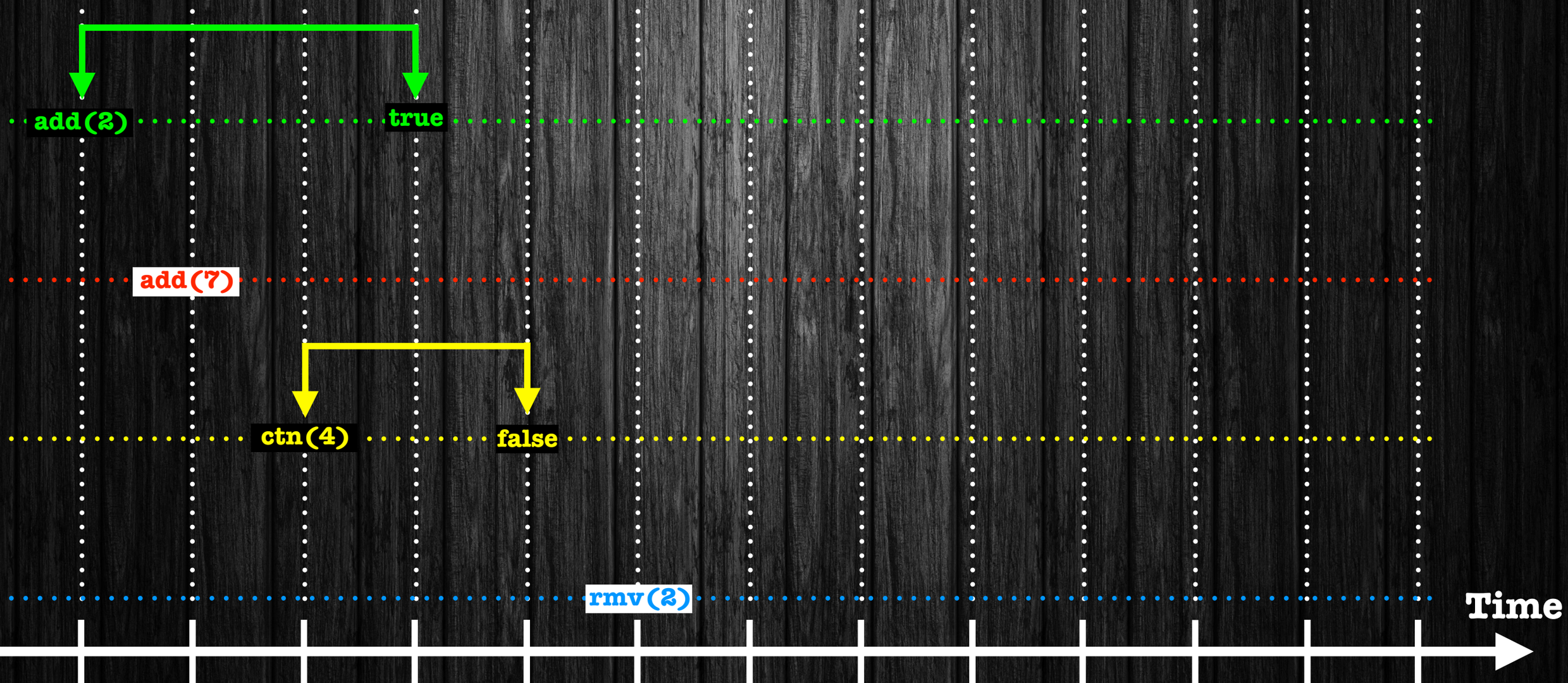
Concurrent Histories Sets



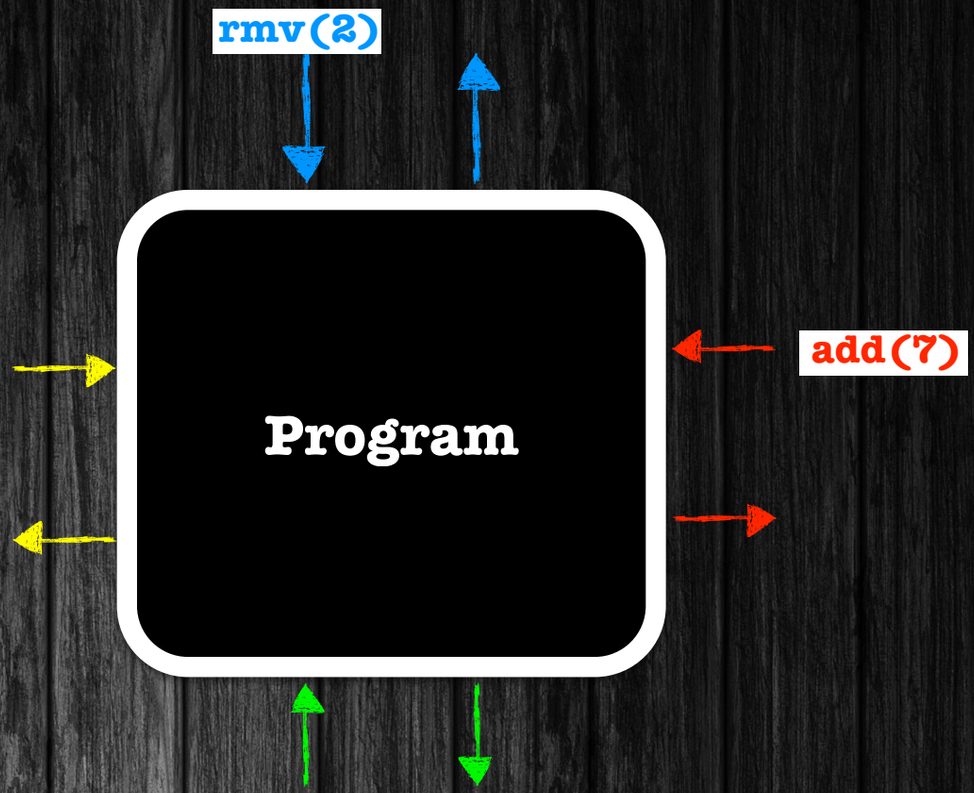


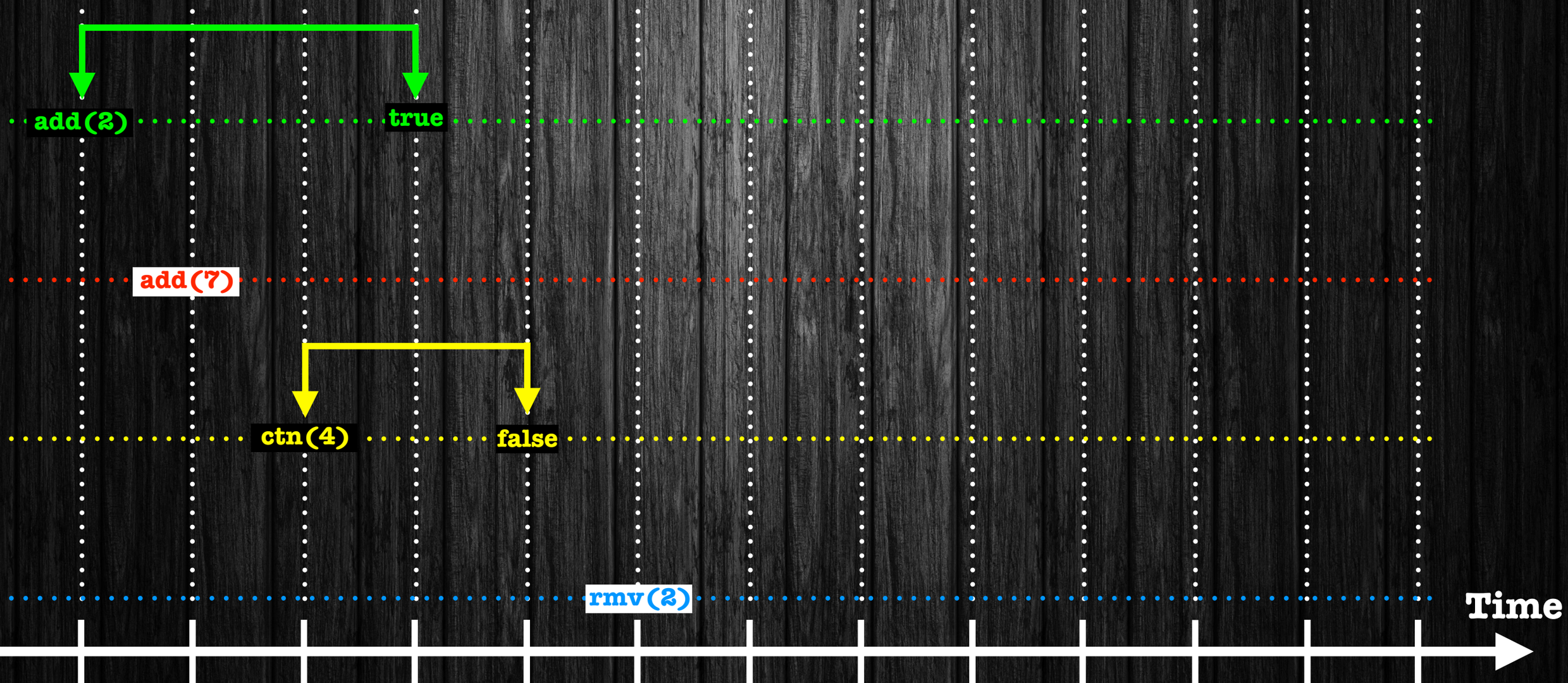
Concurrent Histories Sets



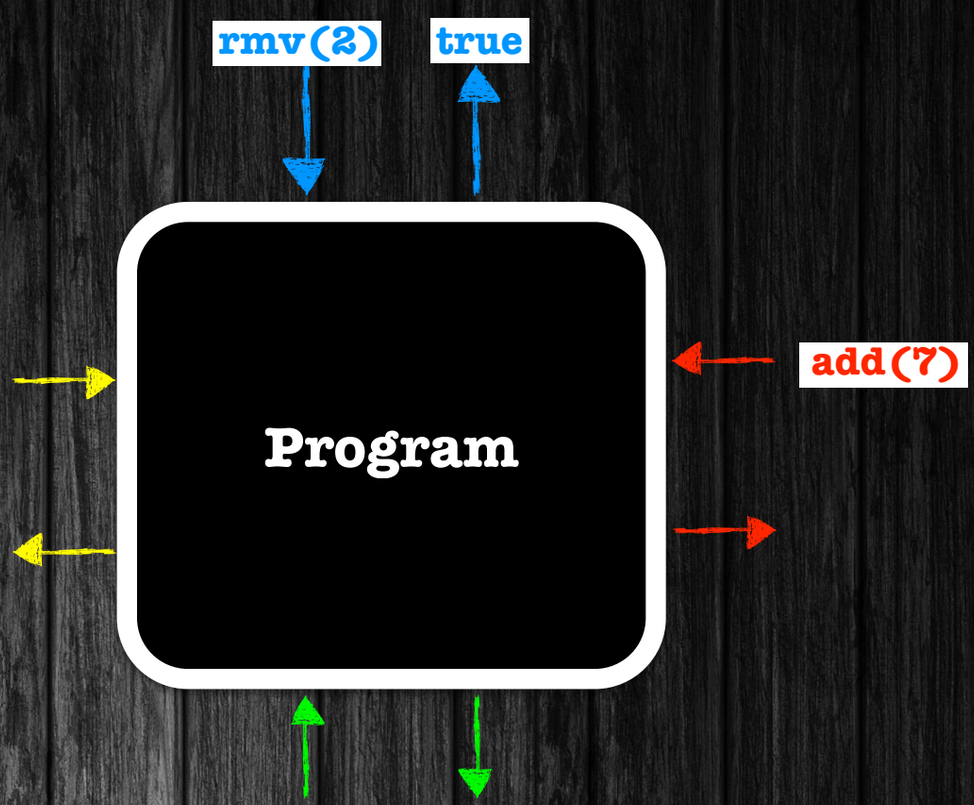


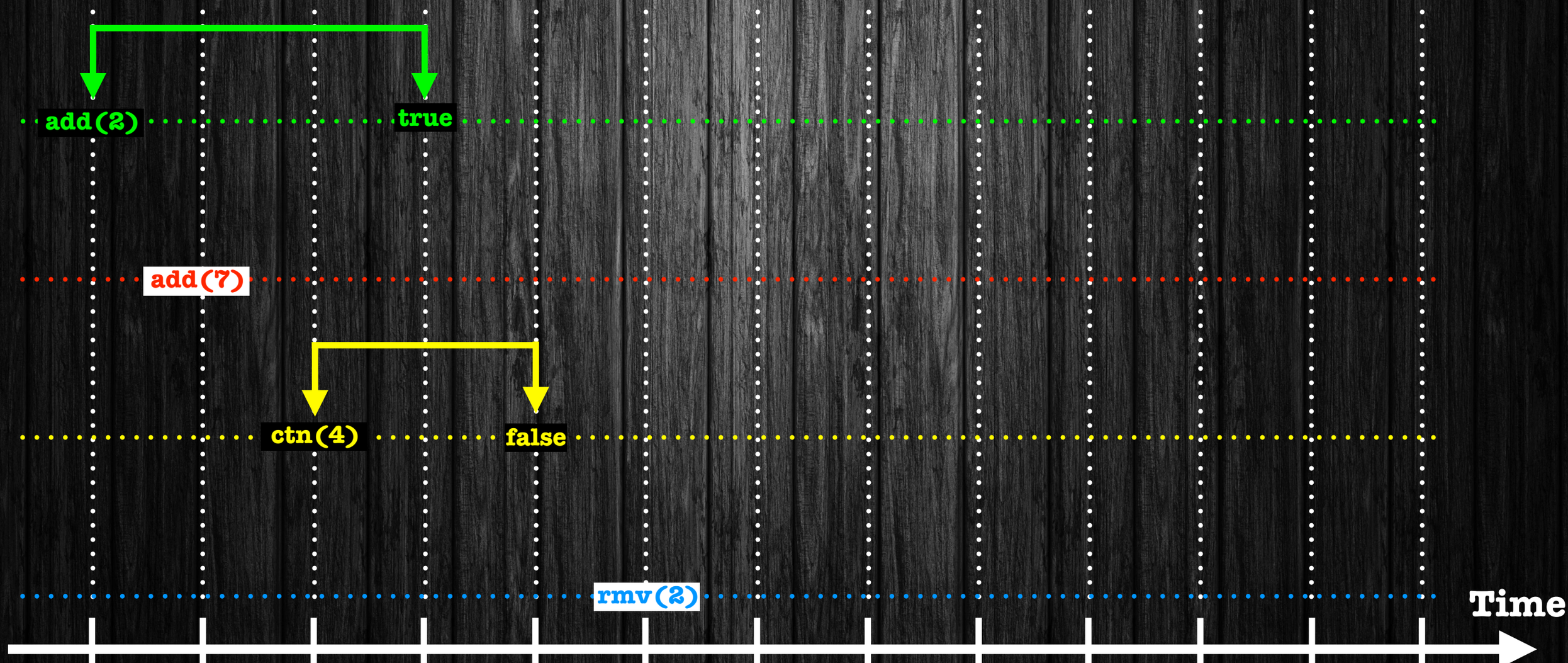
Concurrent Histories Sets



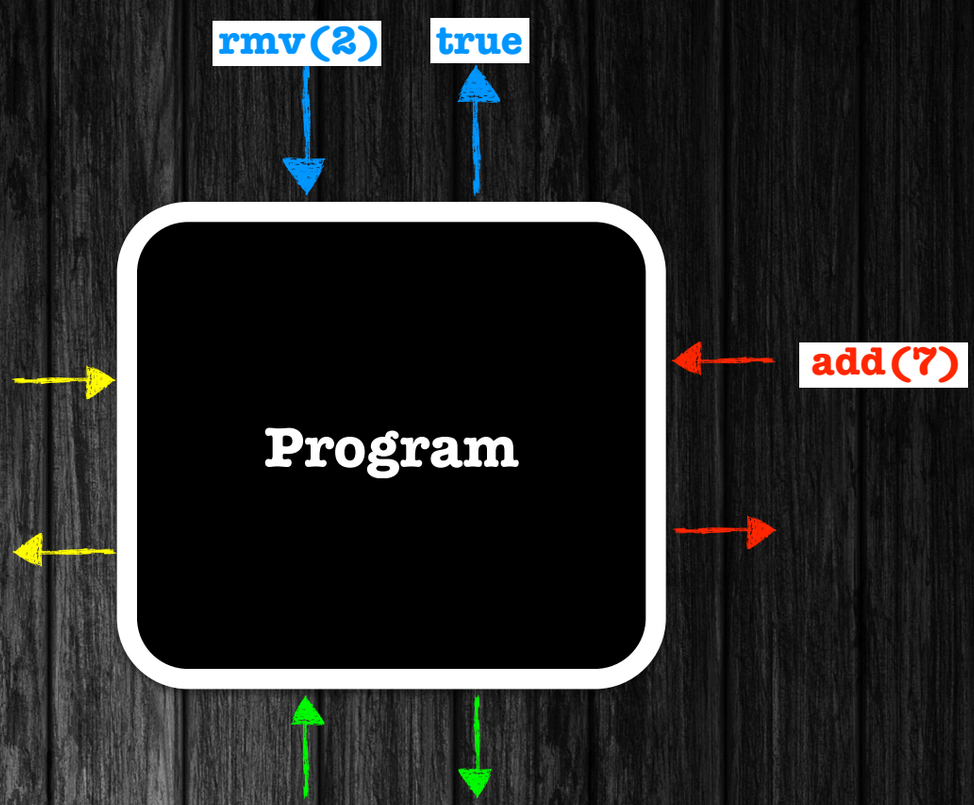


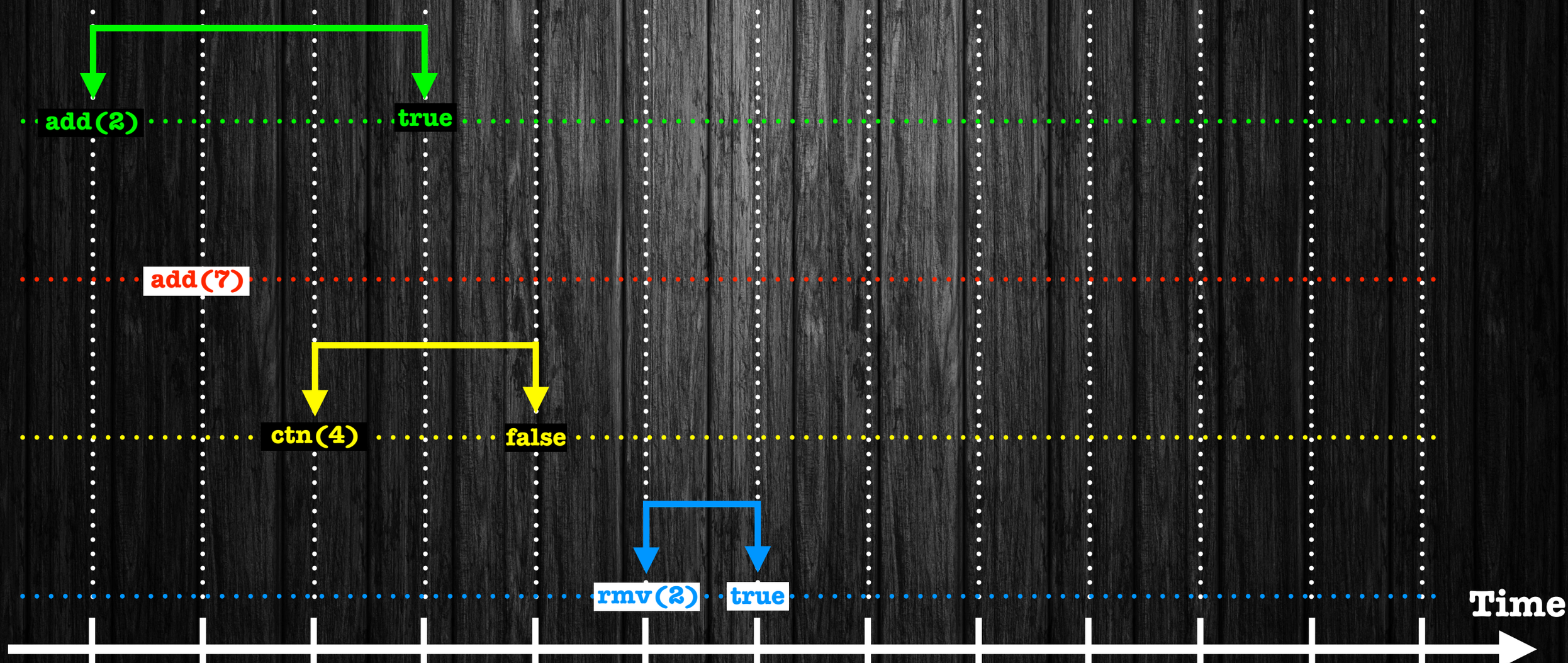
Concurrent Histories | Sets



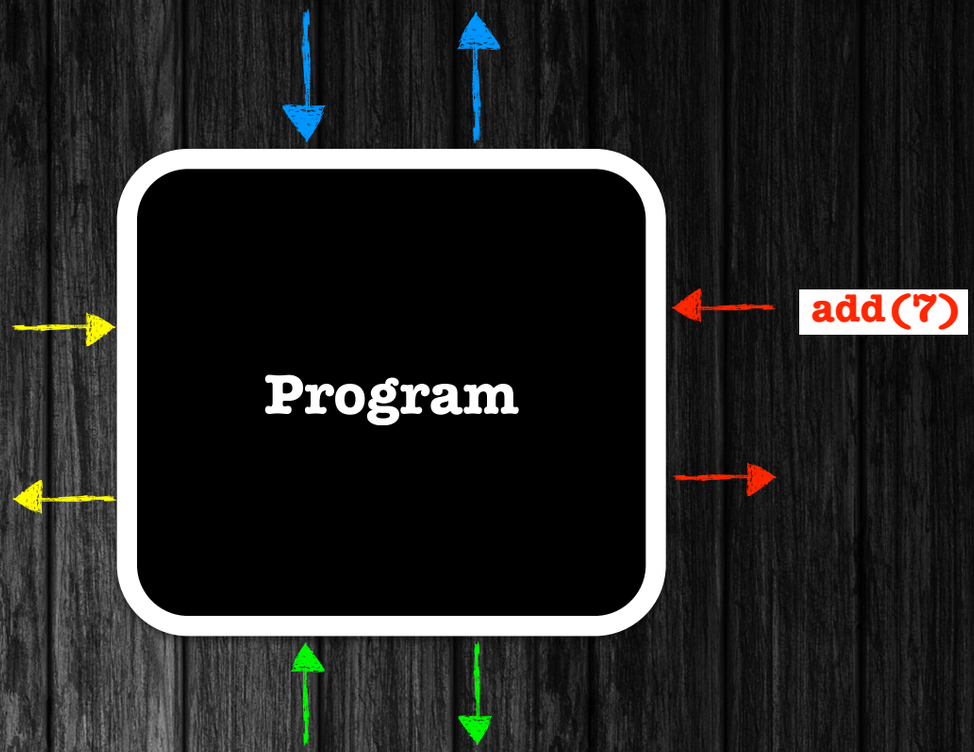


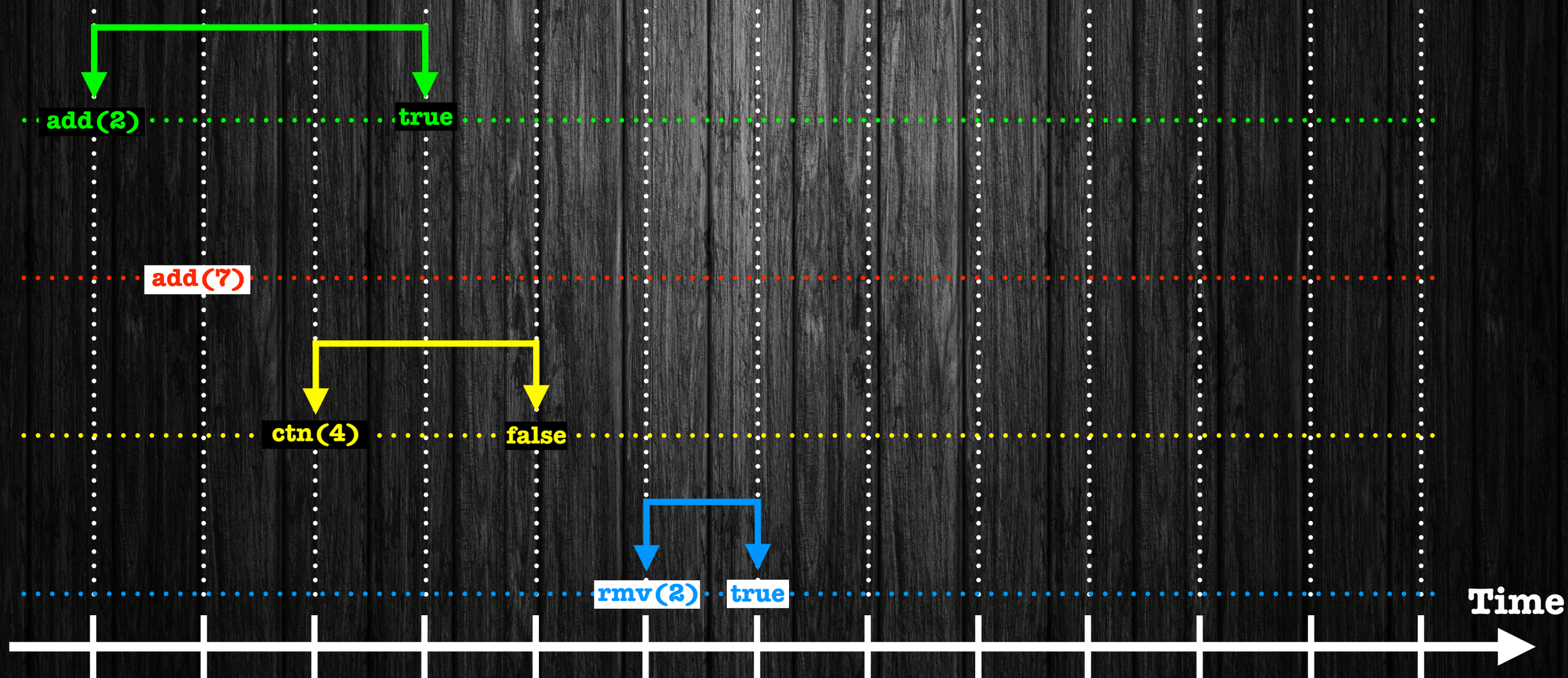
Concurrent Histories | Sets



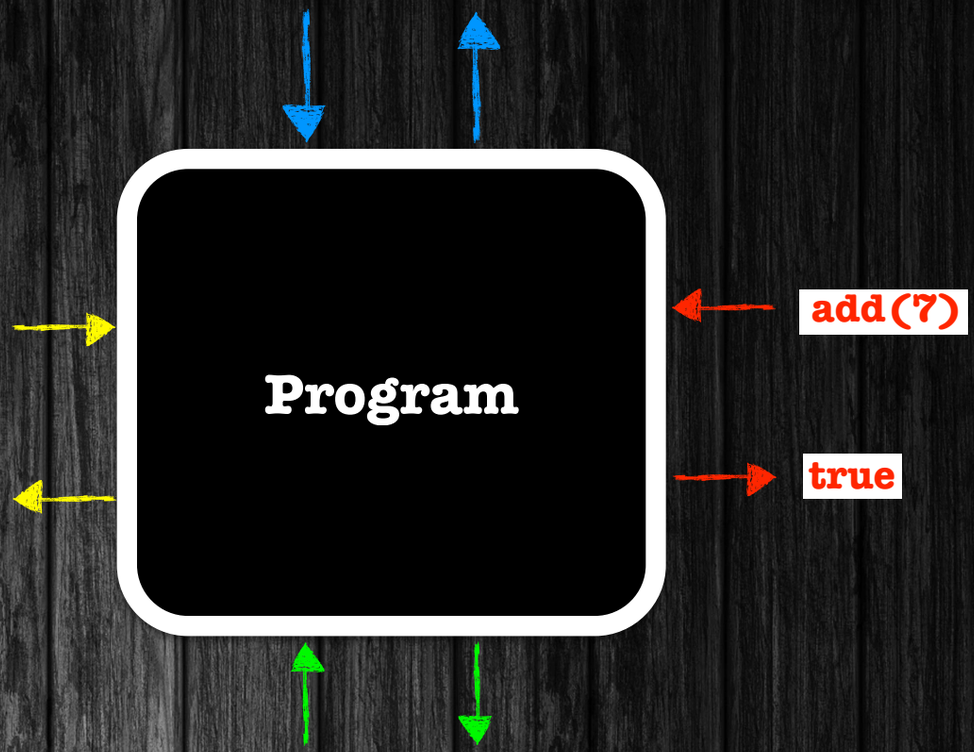


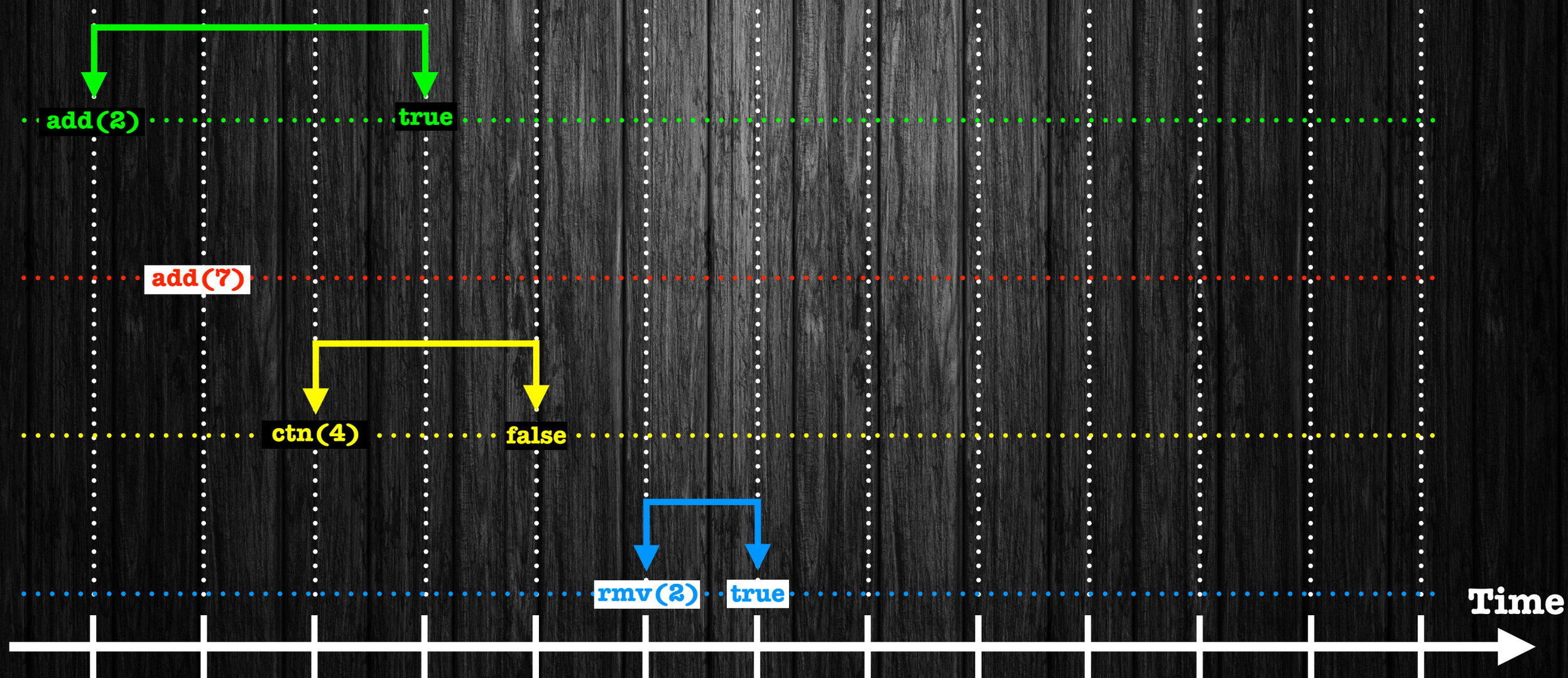
Concurrent Histories Sets



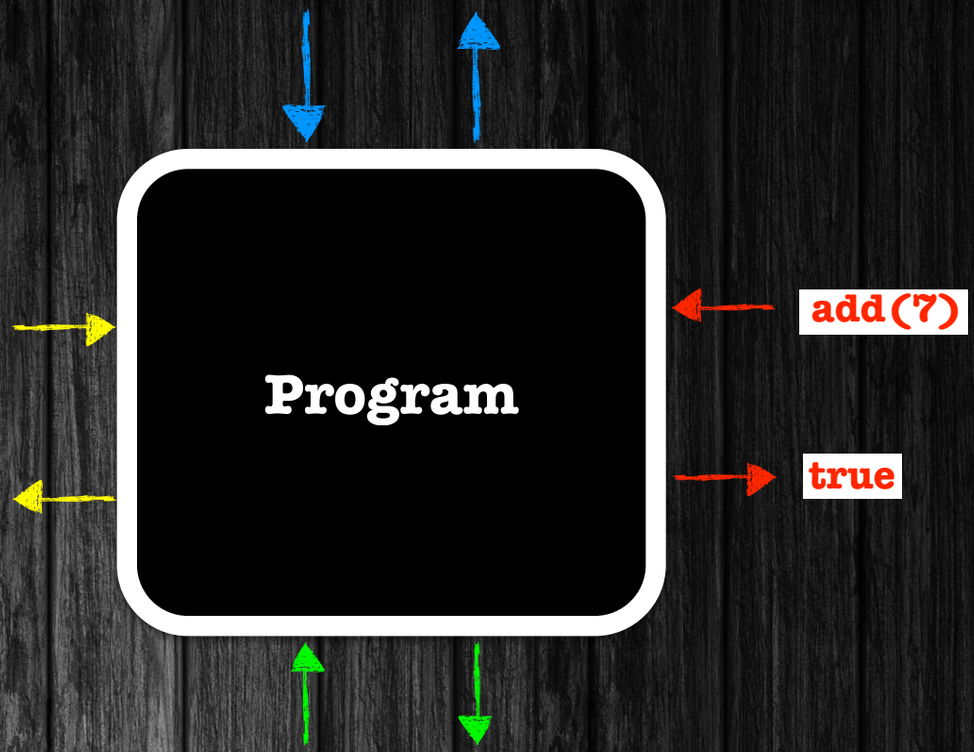


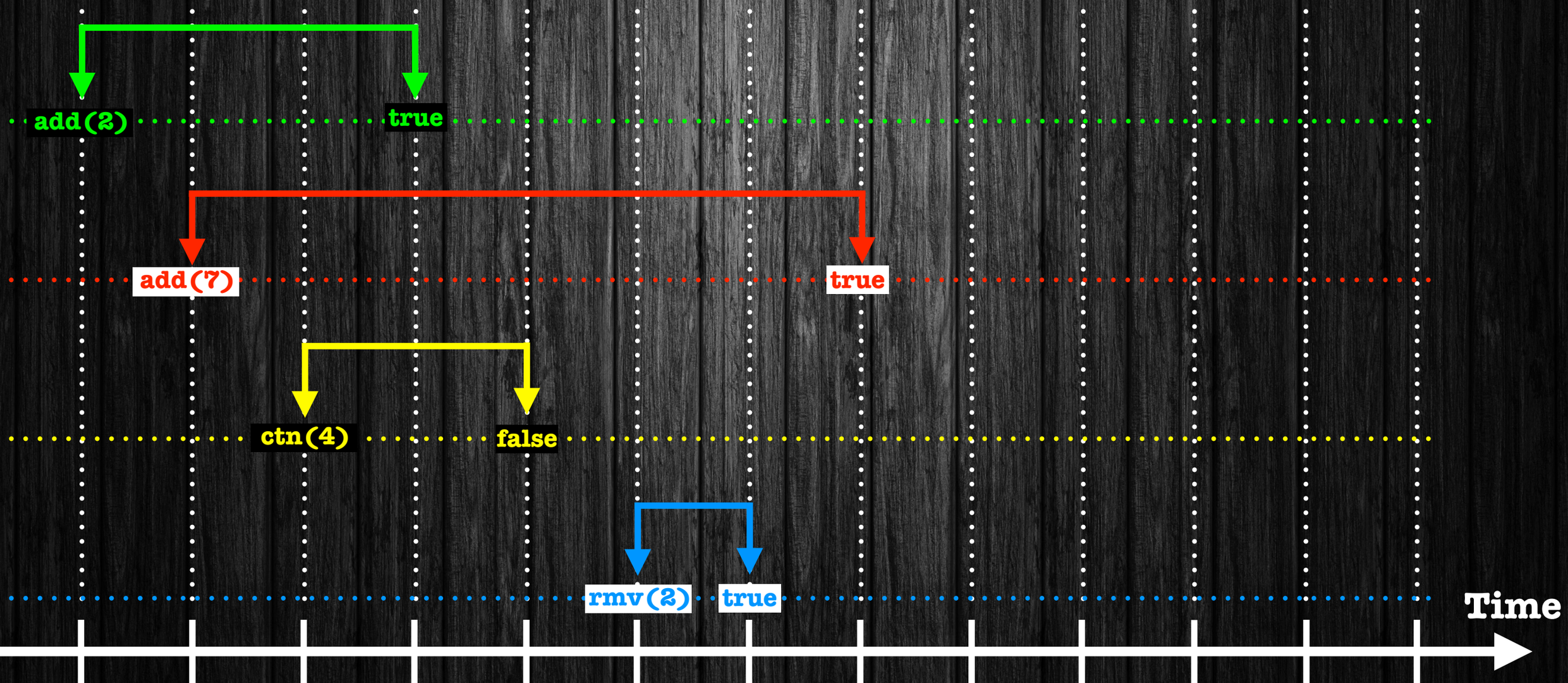
Concurrent Histories Sets





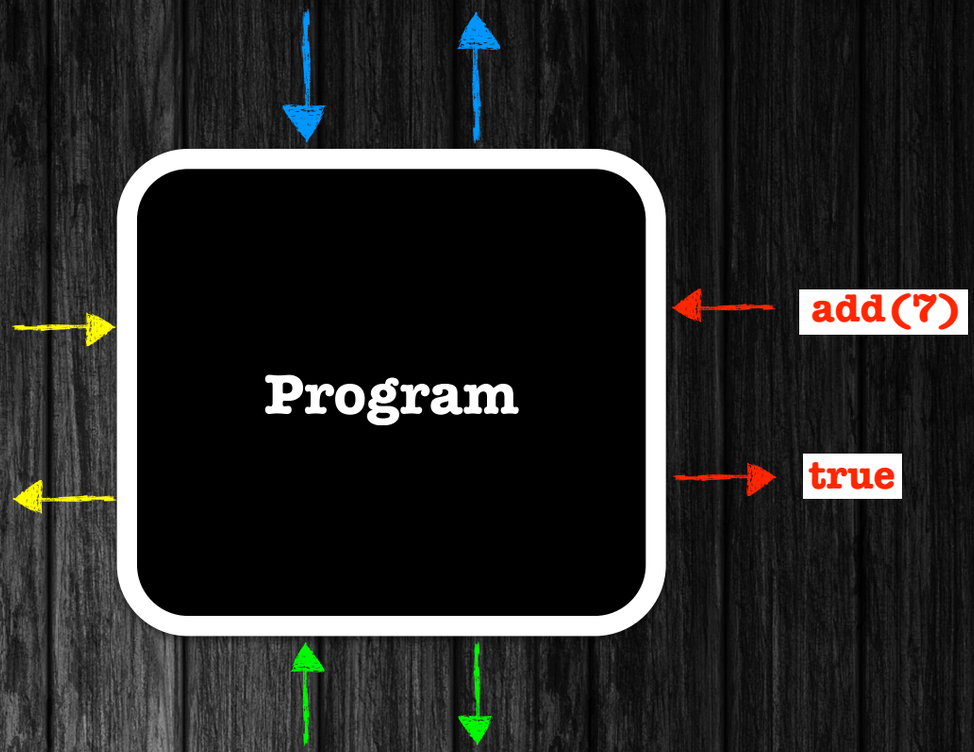
Concurrent Histories Sets

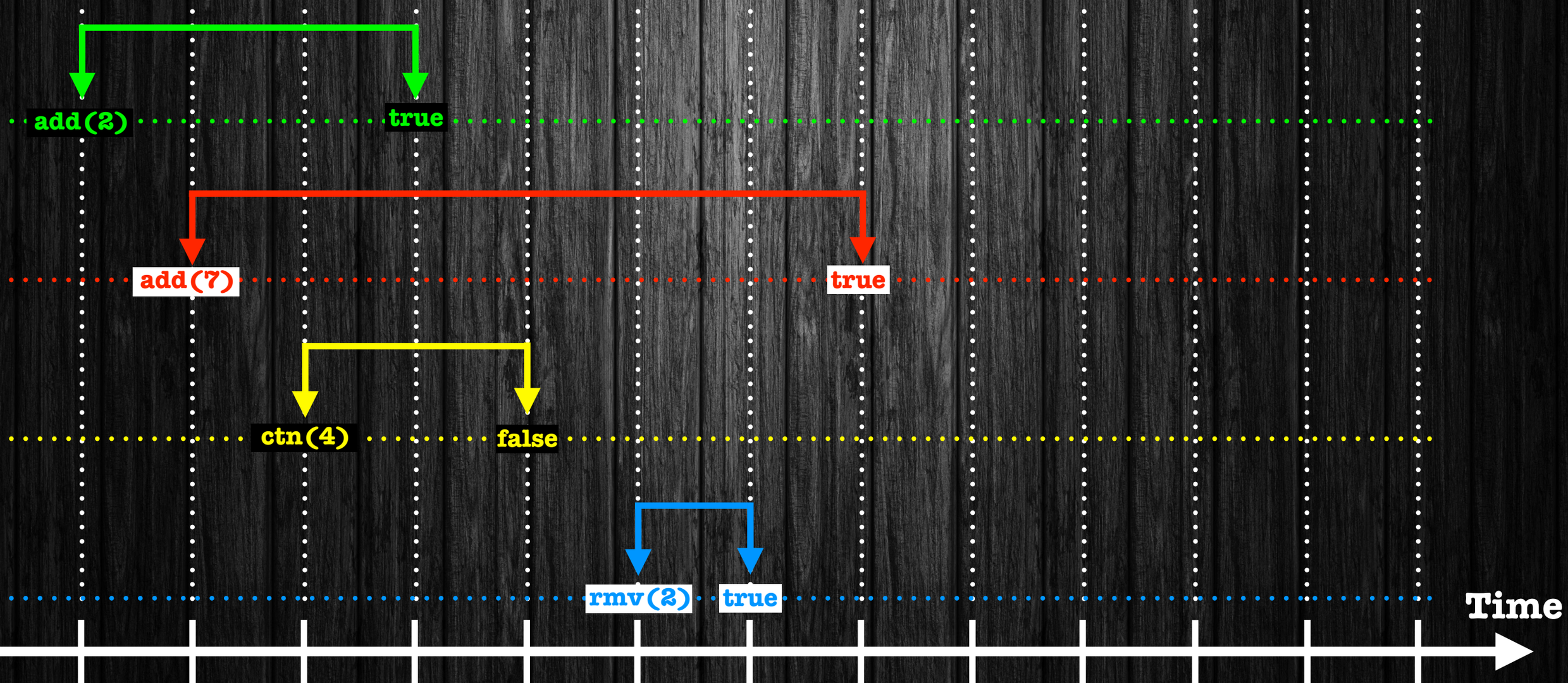




Concurrent Histories Sets

the threads operate concurrently

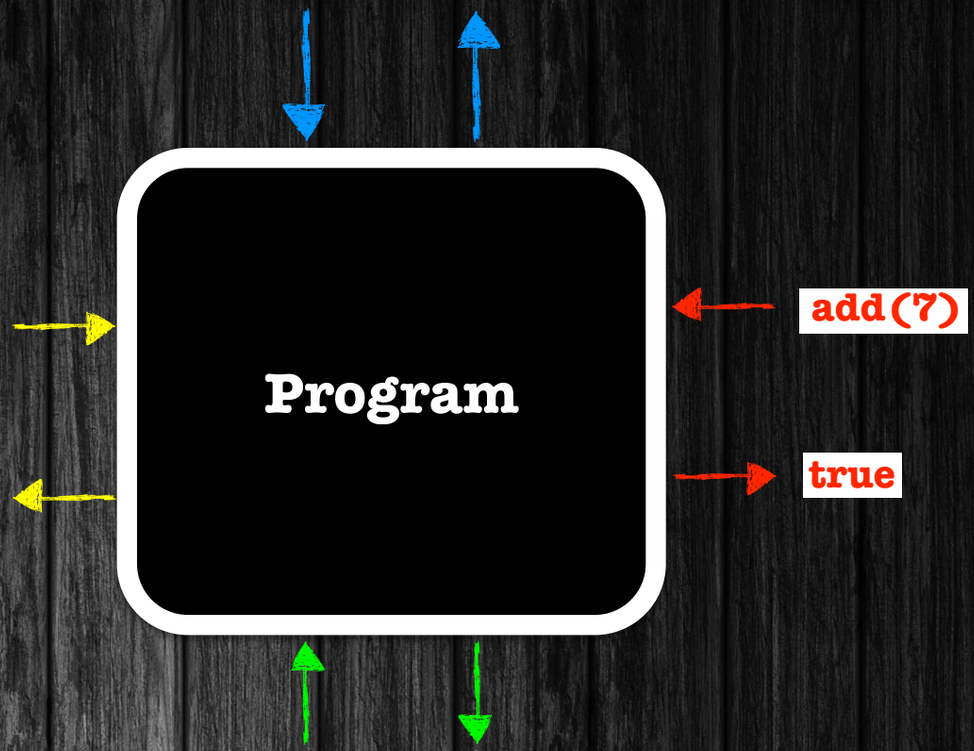


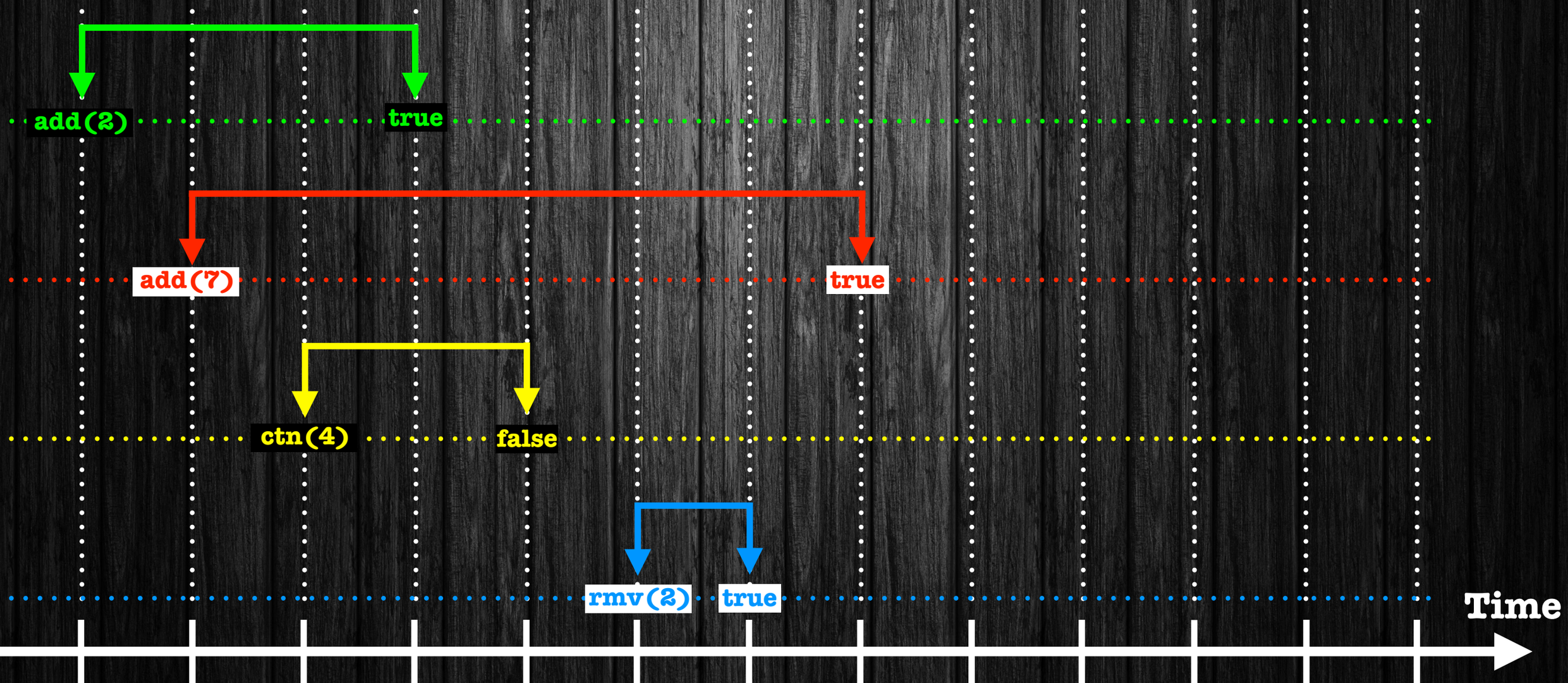


Concurrent Histories Sets

the threads operate concurrently

calls may overlap

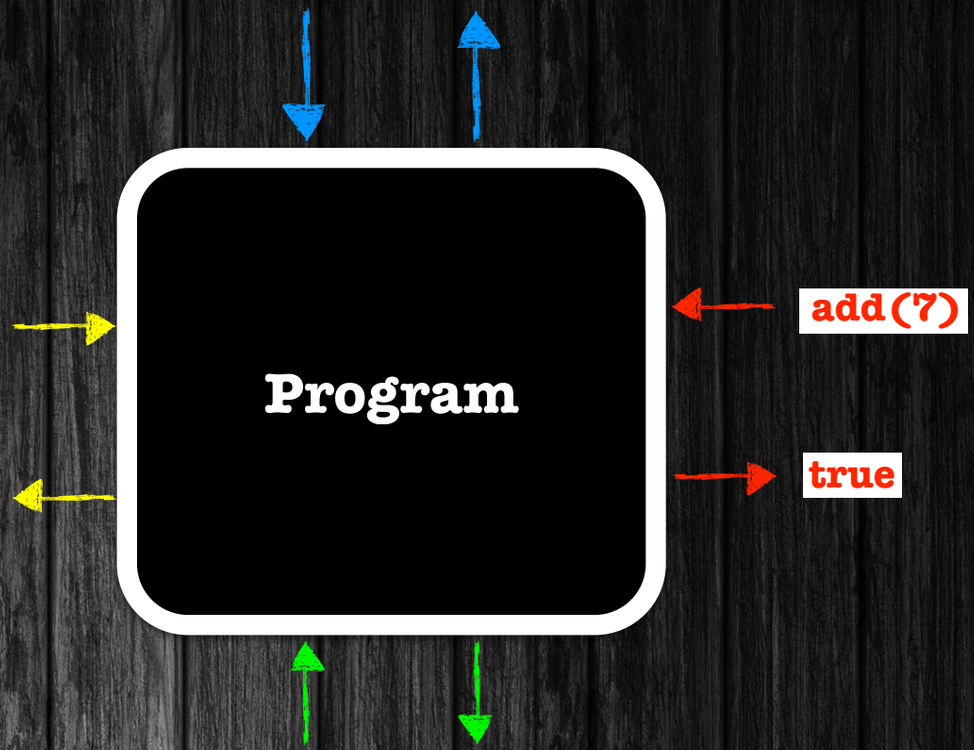


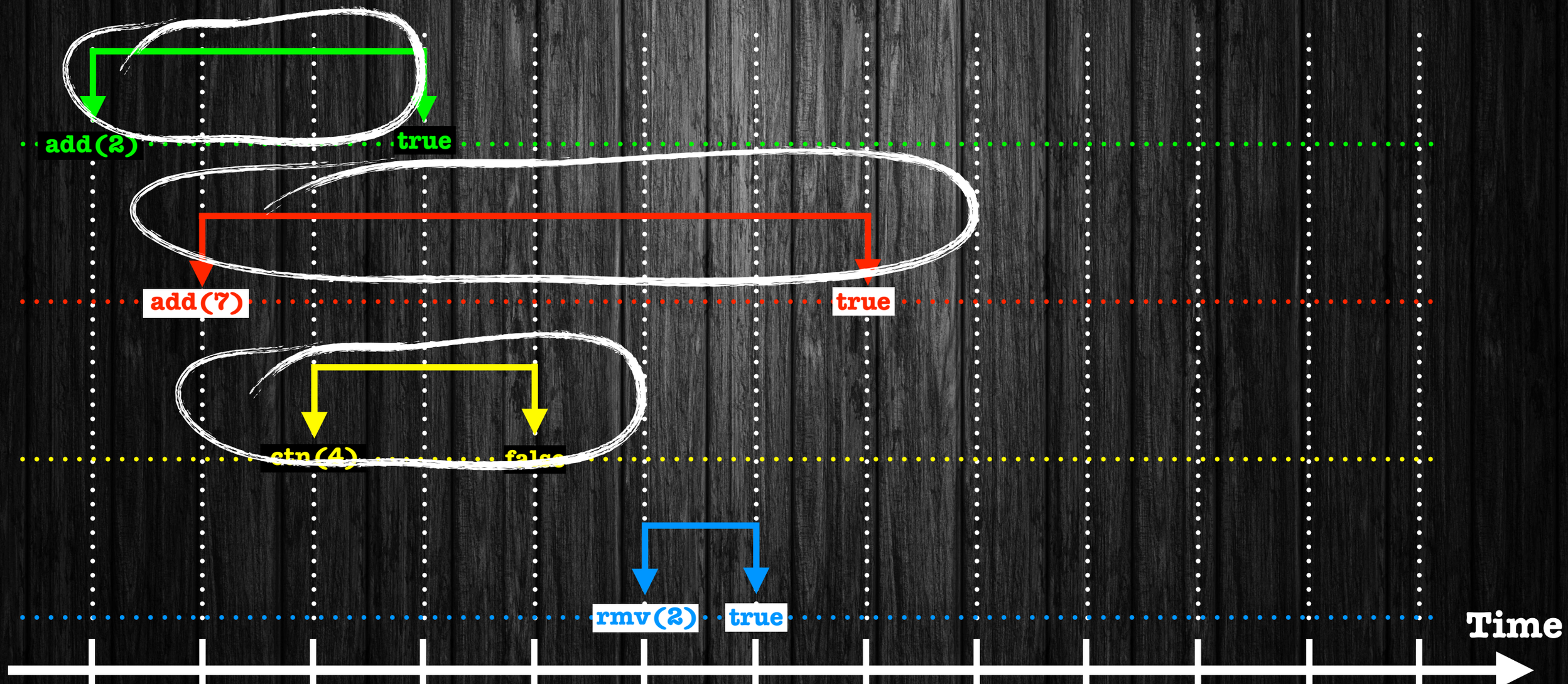


Concurrent Histories Sets

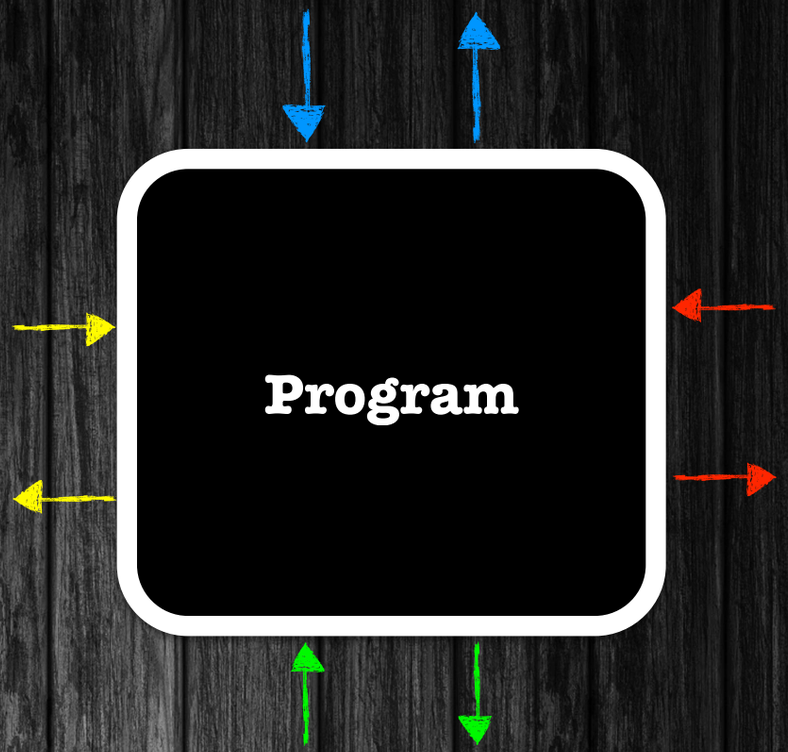
the threads operate concurrently

calls may overlap



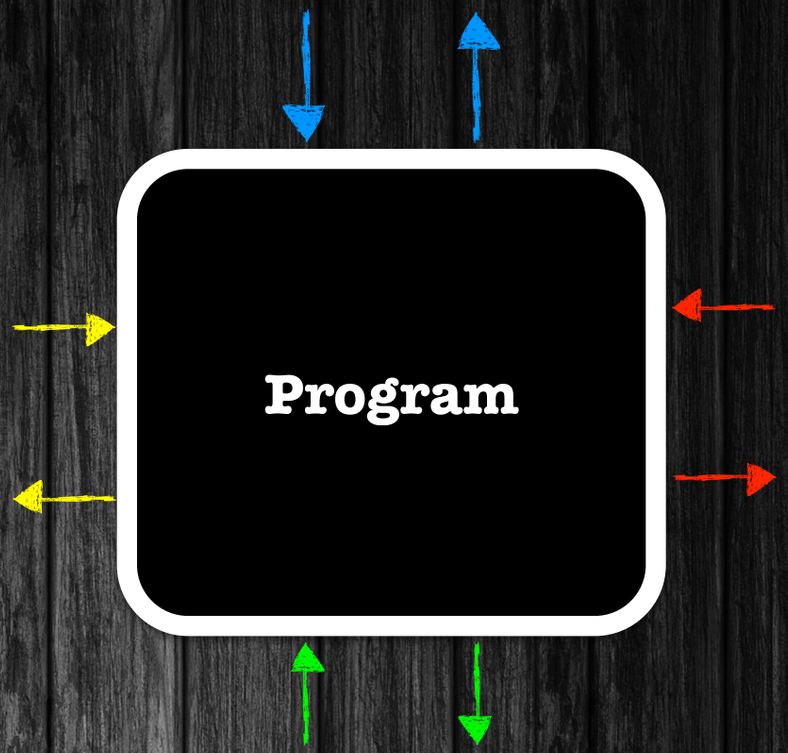


Concurrent Histories Sets

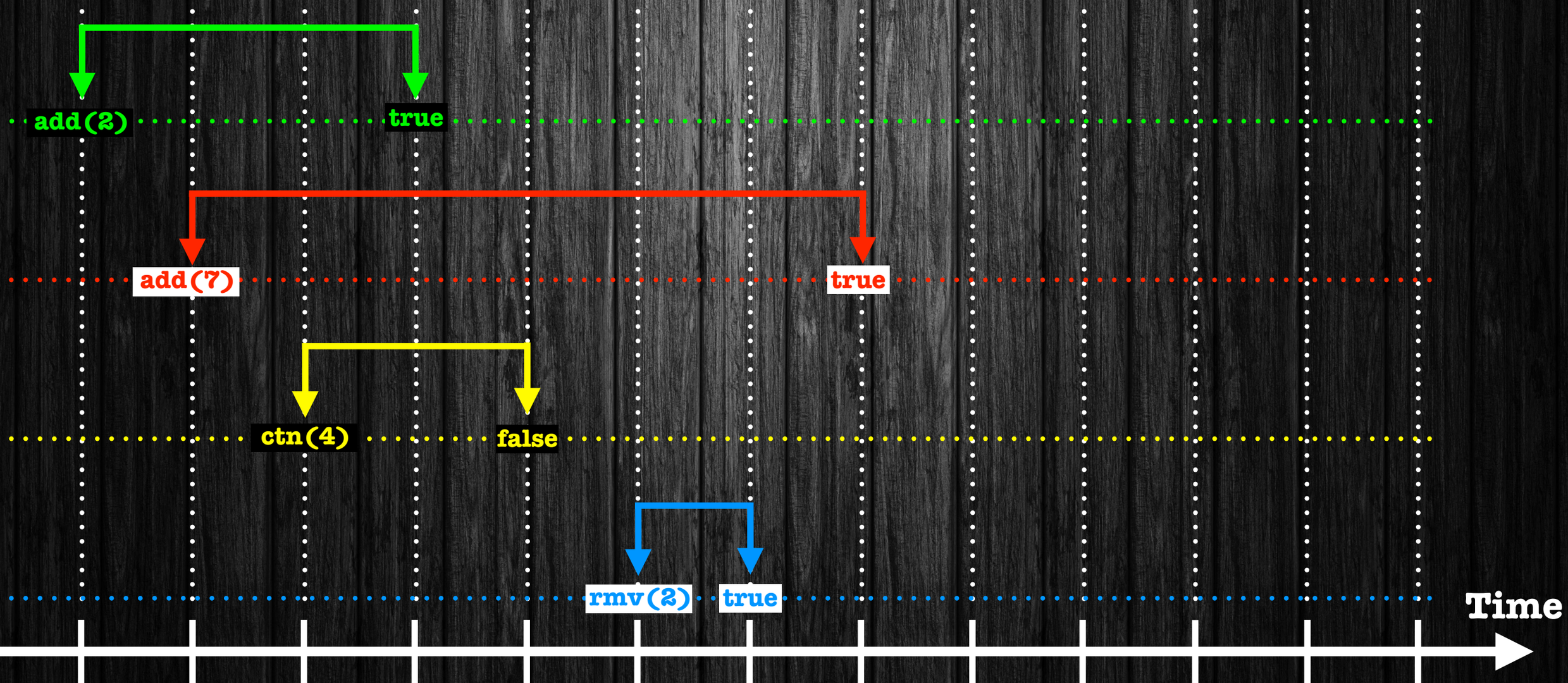


add(2)
true
add(7)
true
ctn(4)
false
rmv(2)
true
Time

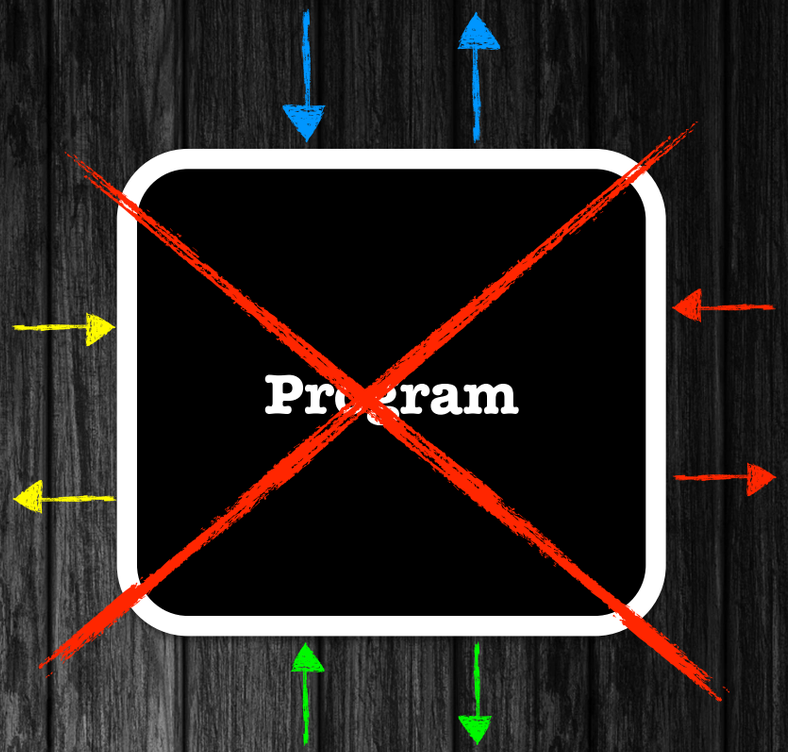
Concurrent Histories Sets



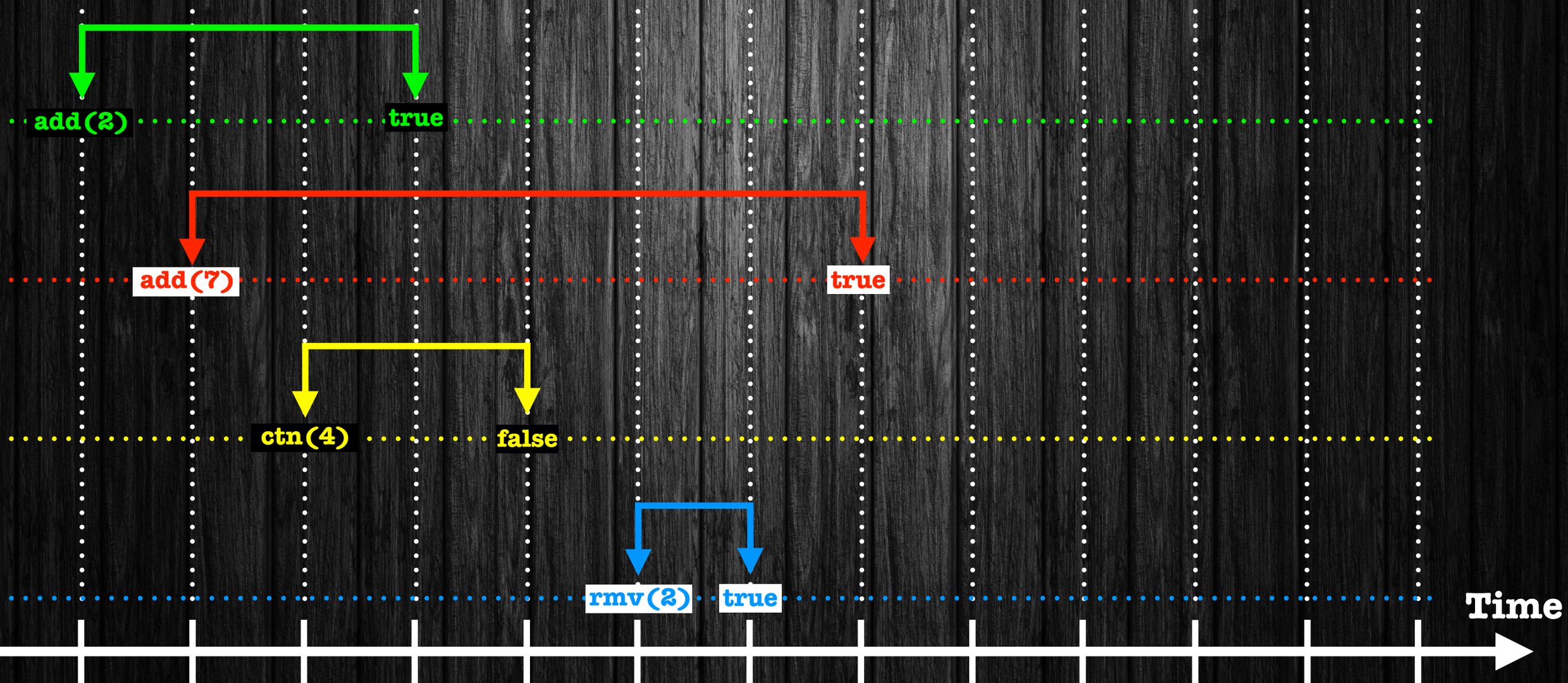
from now on



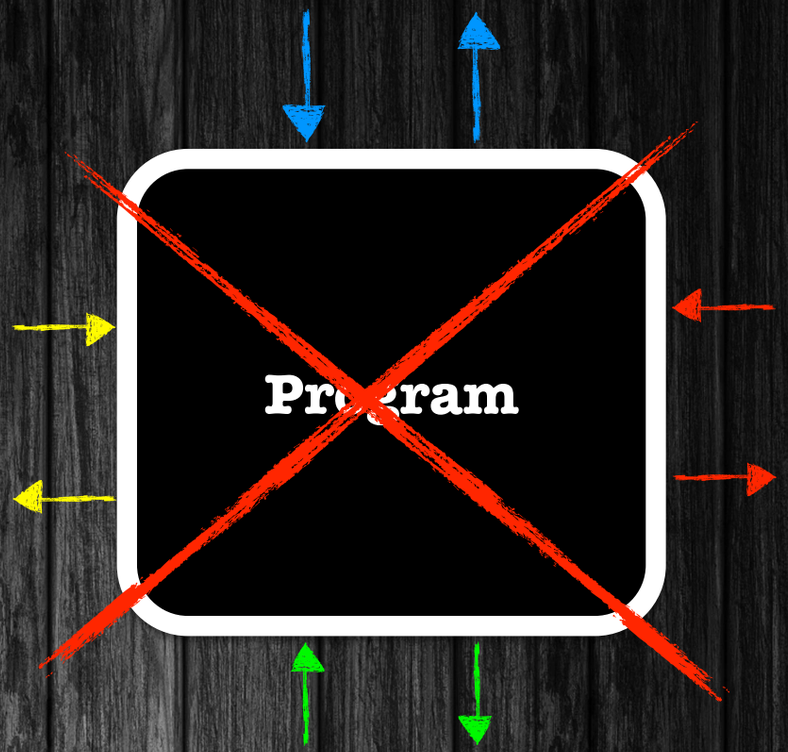
Concurrent Histories Sets



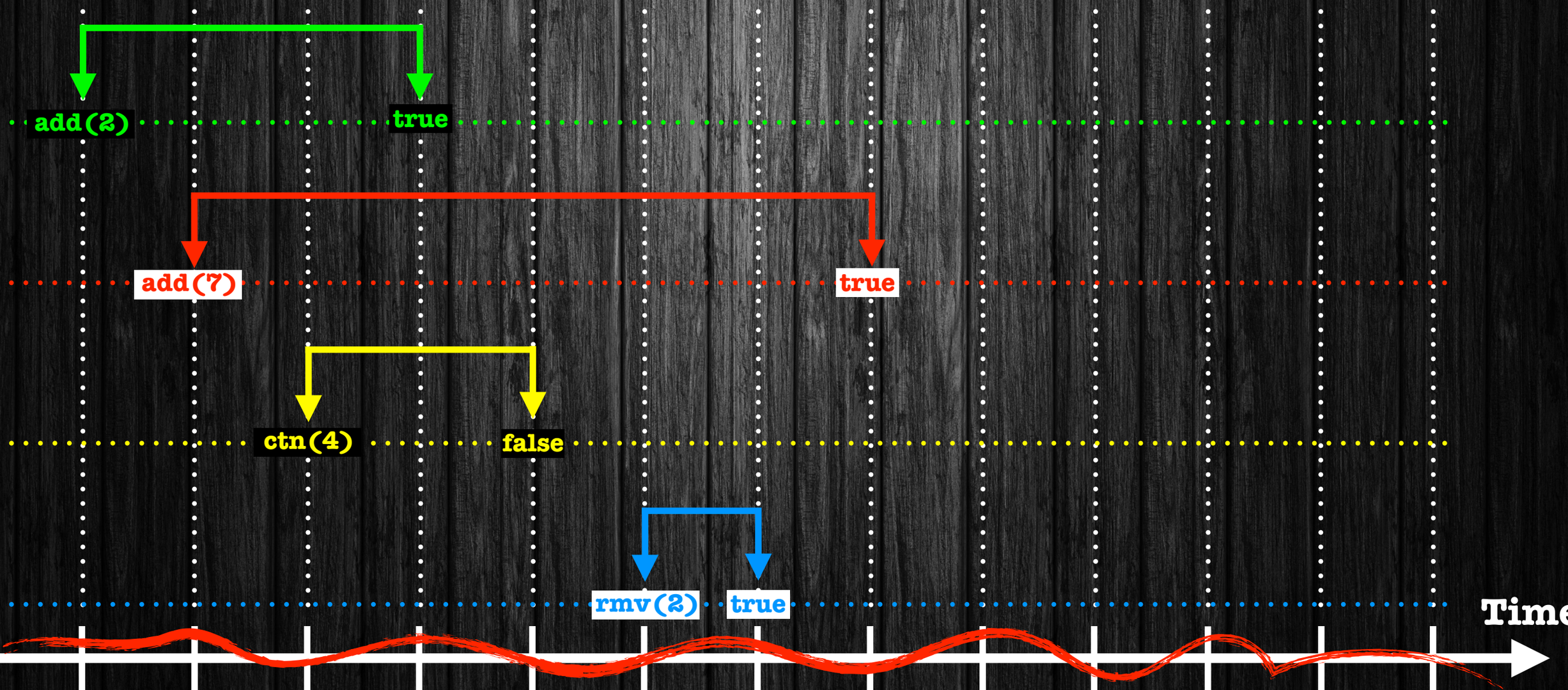
from now on



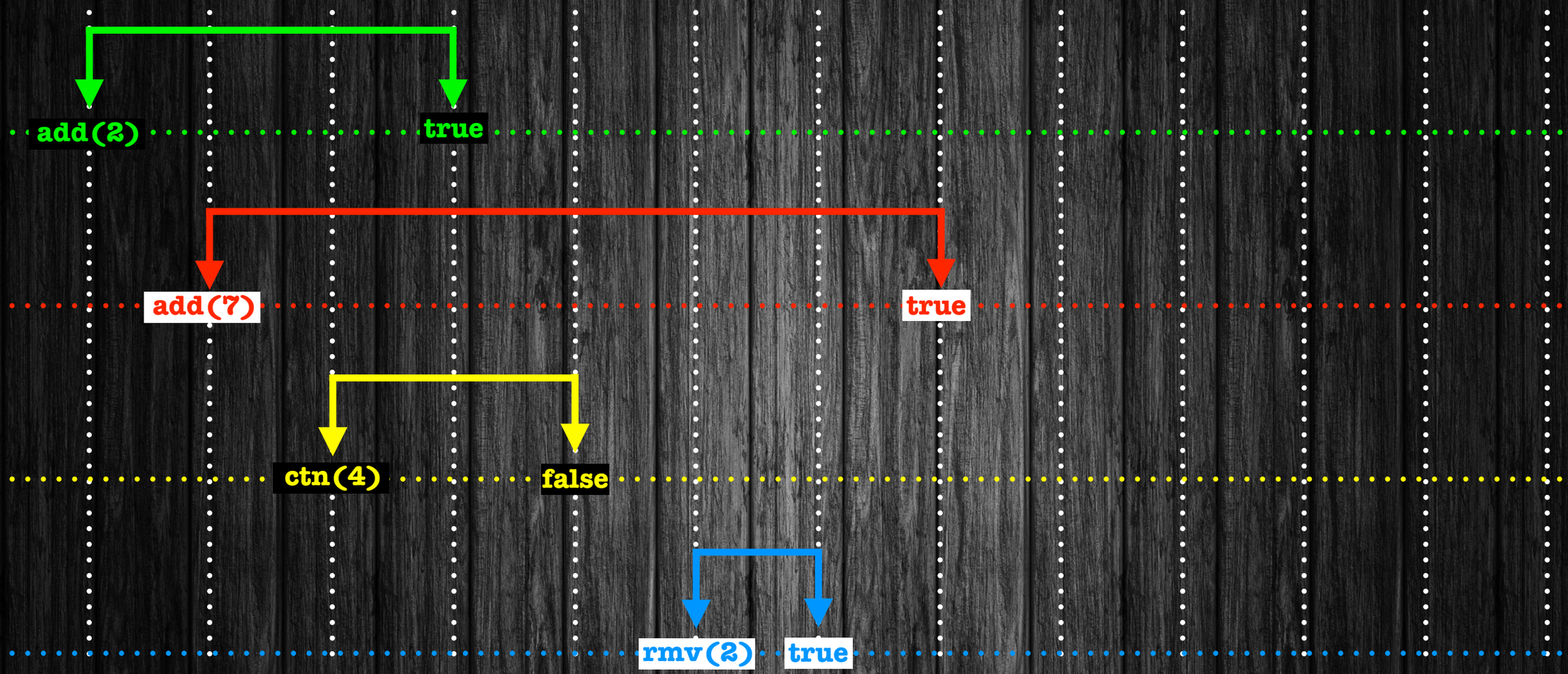
Concurrent Histories Sets



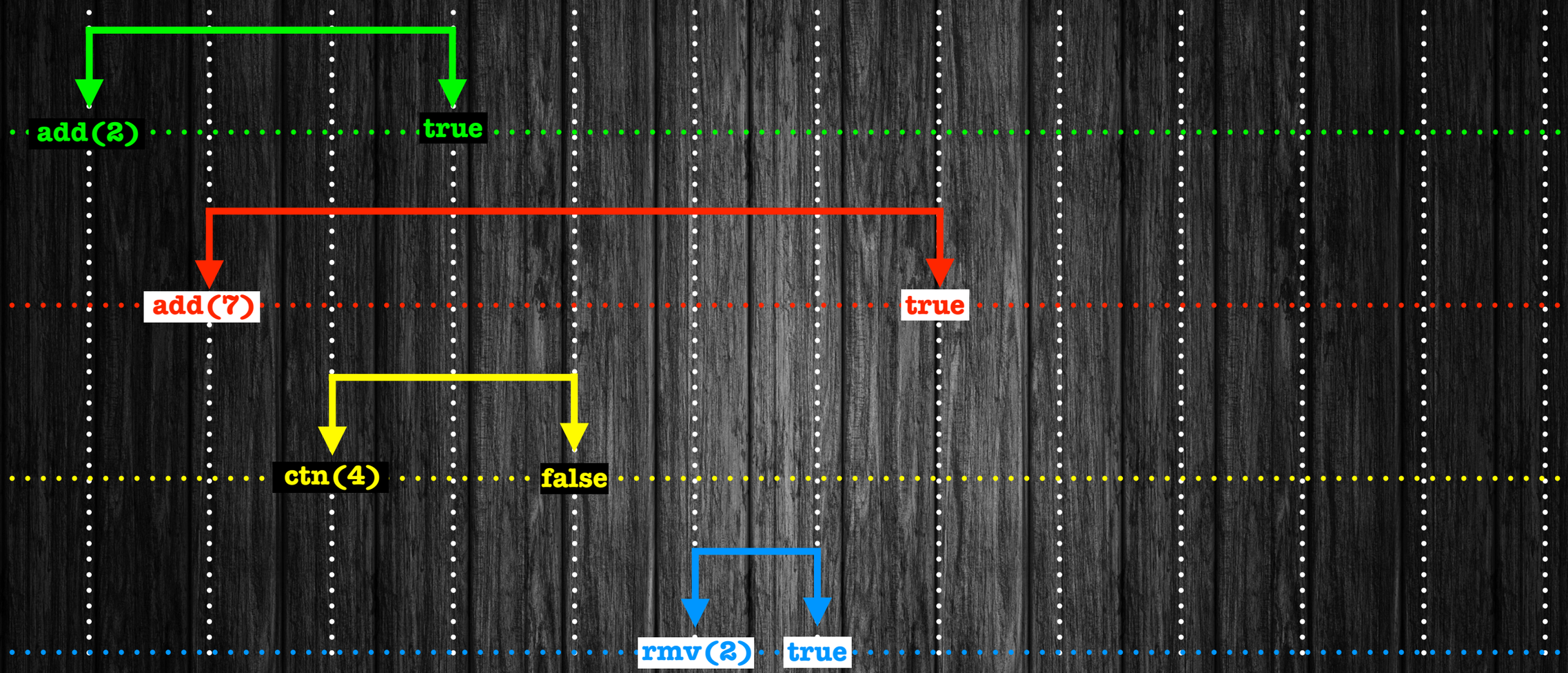
from now on



Concurrent Histories | Sets

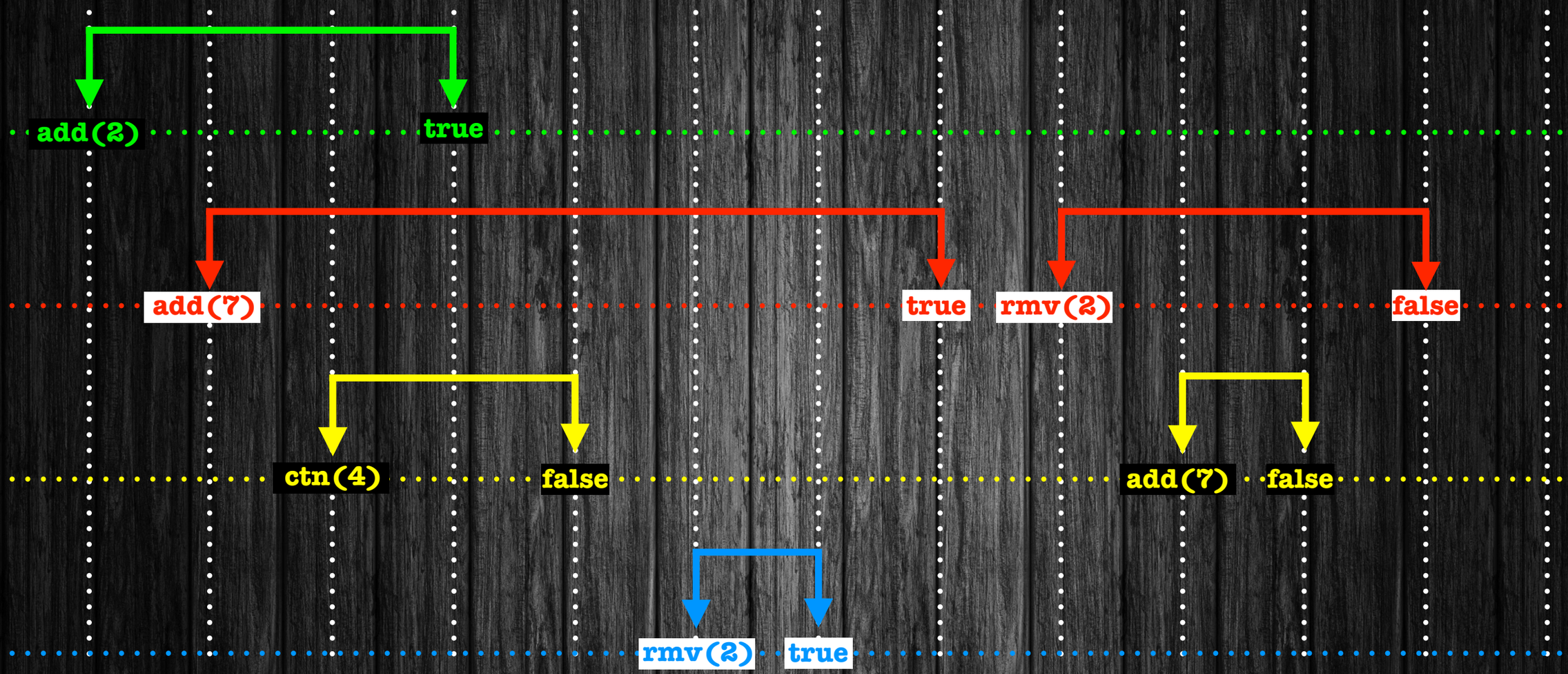


Concurrent Histories | Sets

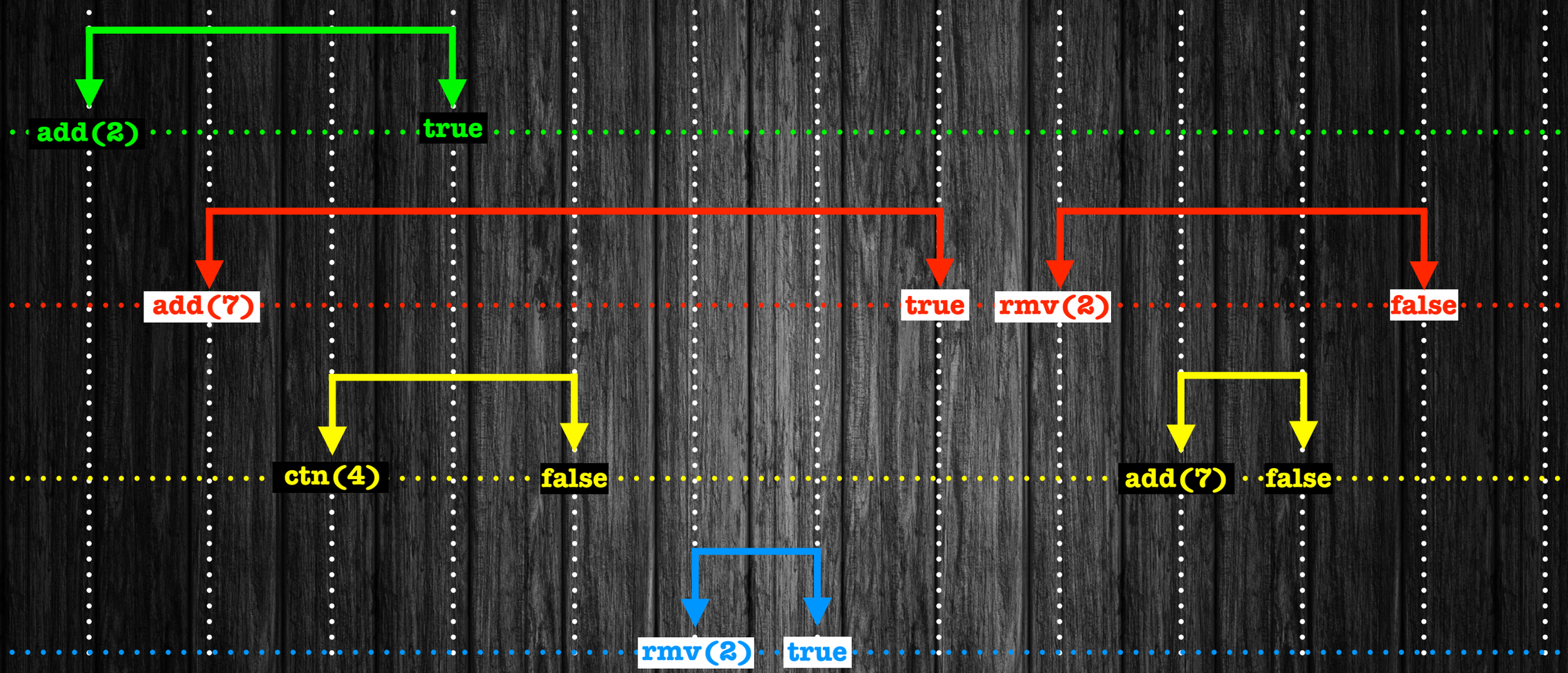


✔

Concurrent Histories | Sets

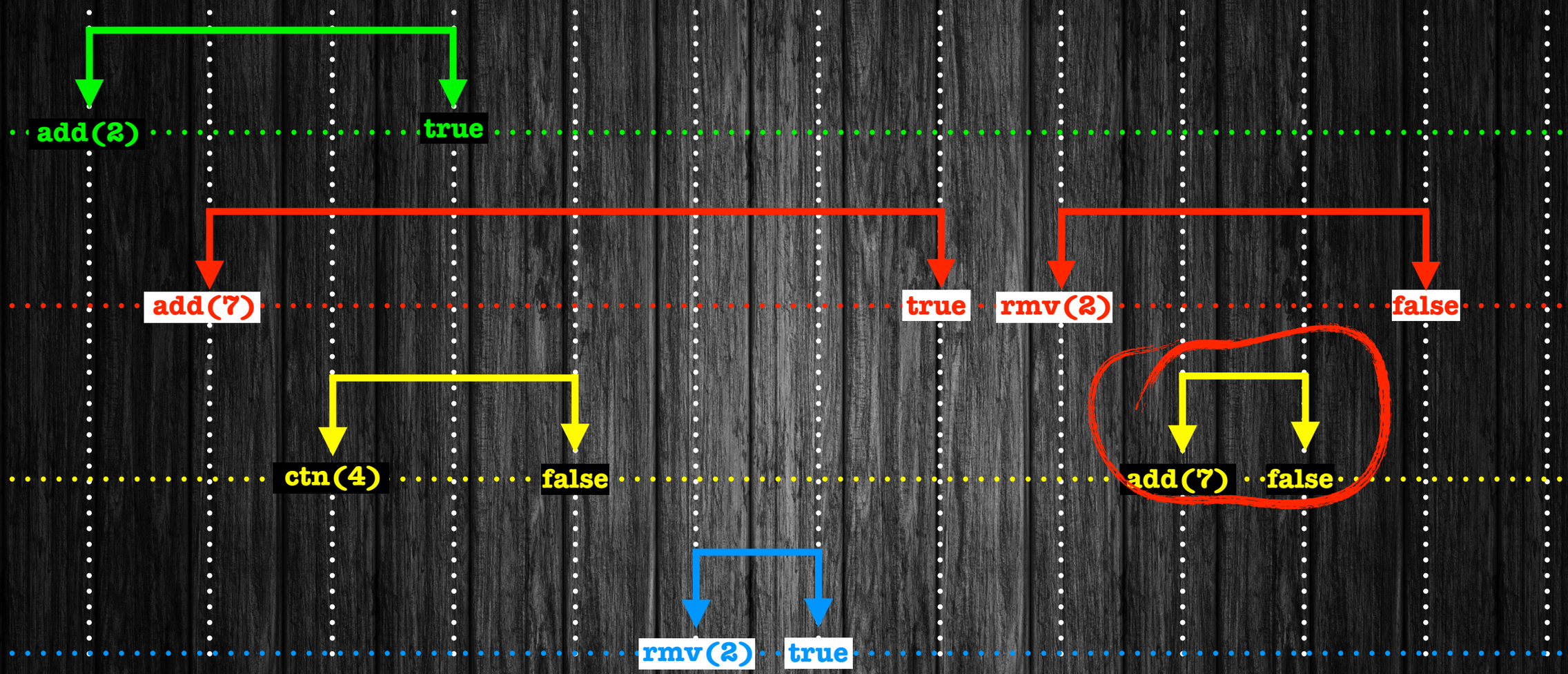


Concurrent Histories | Sets



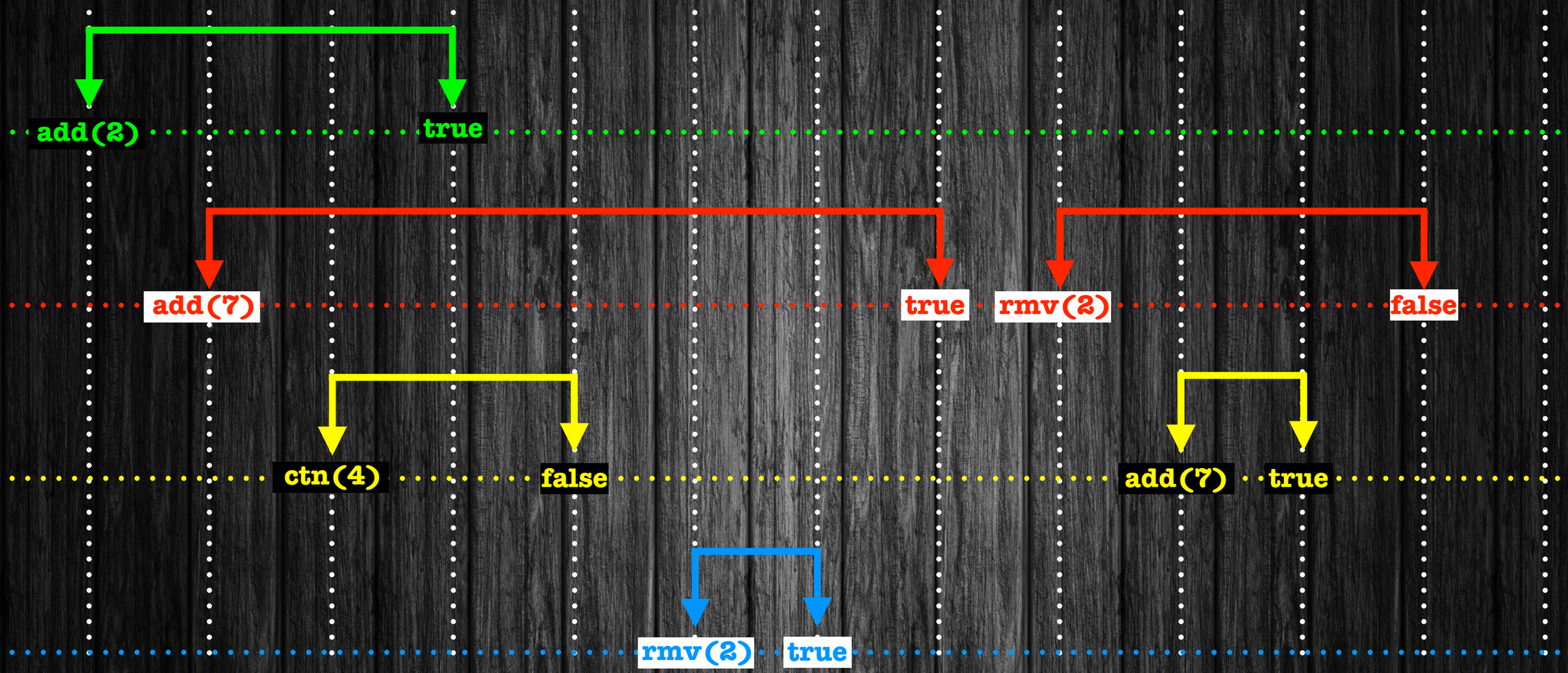
✔

Concurrent Histories | Sets

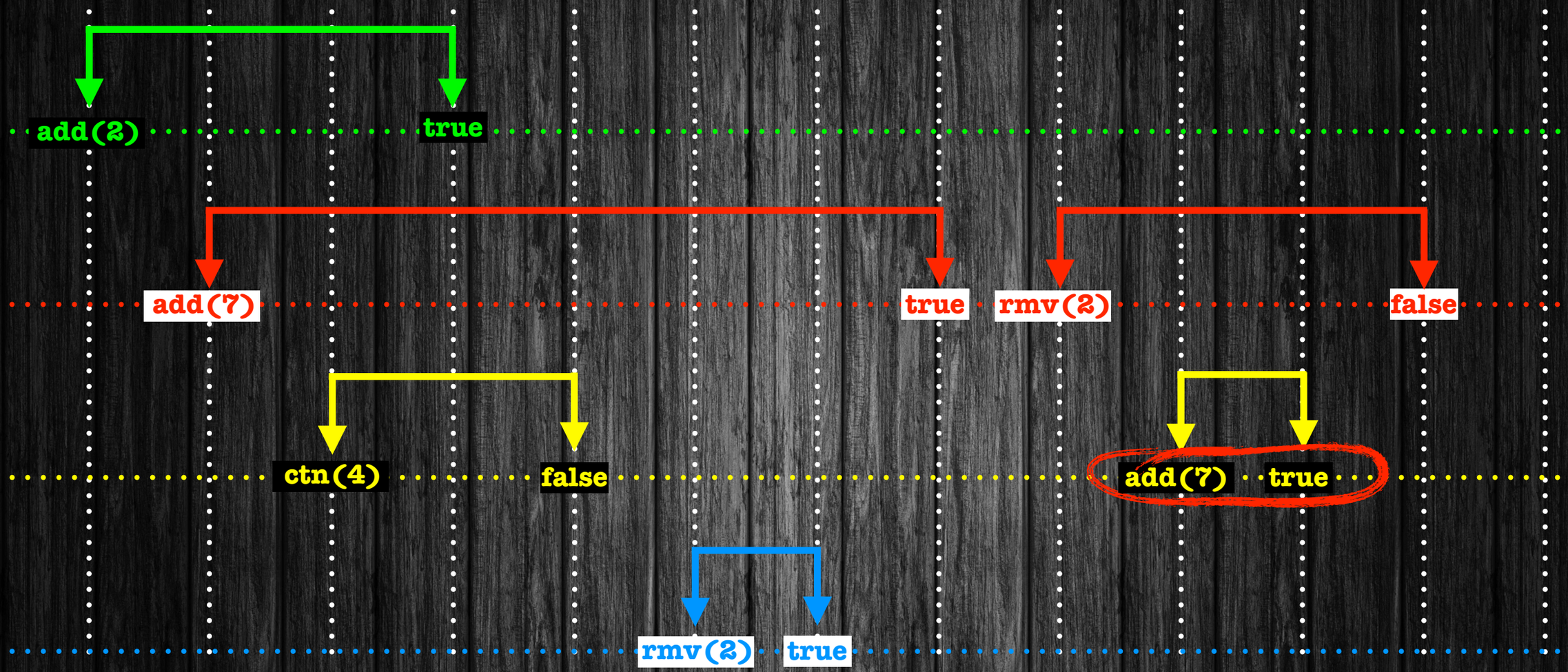


✔

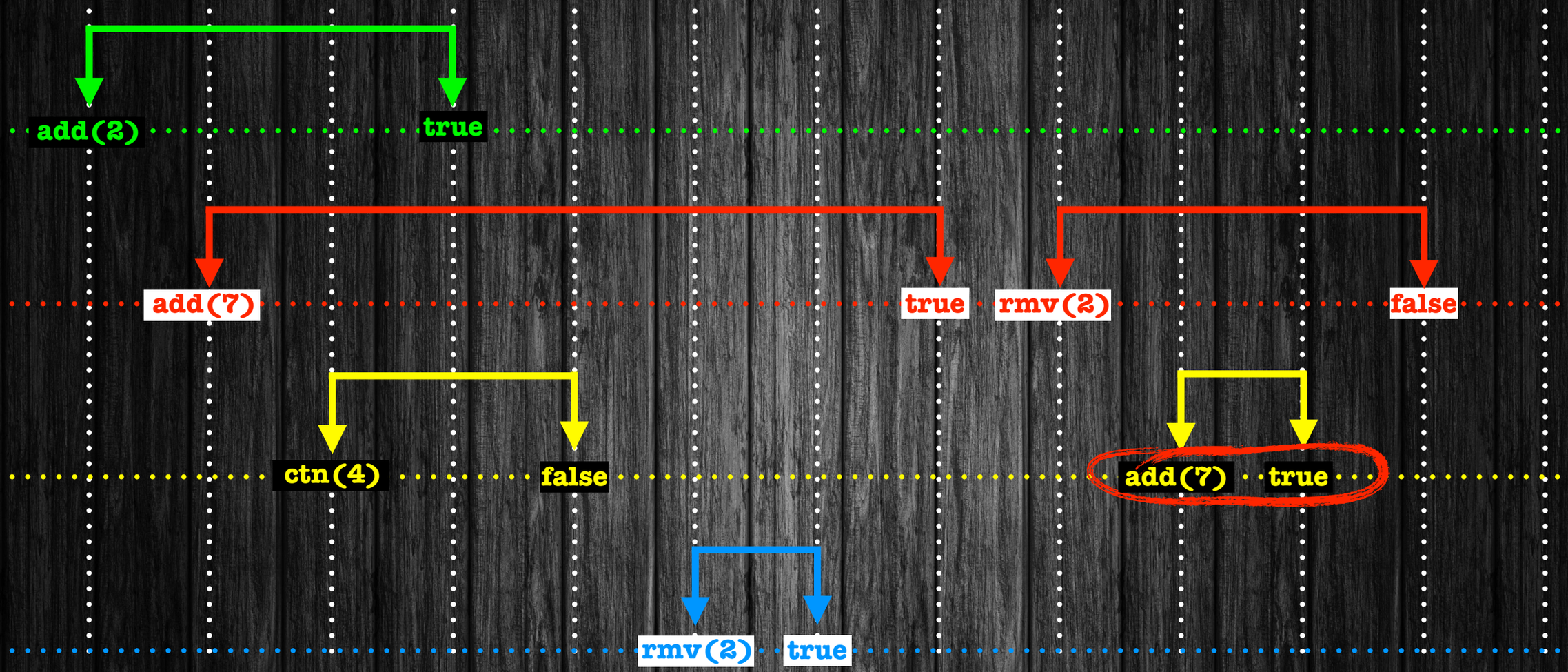
Concurrent Histories | Sets



Concurrent Histories | Sets

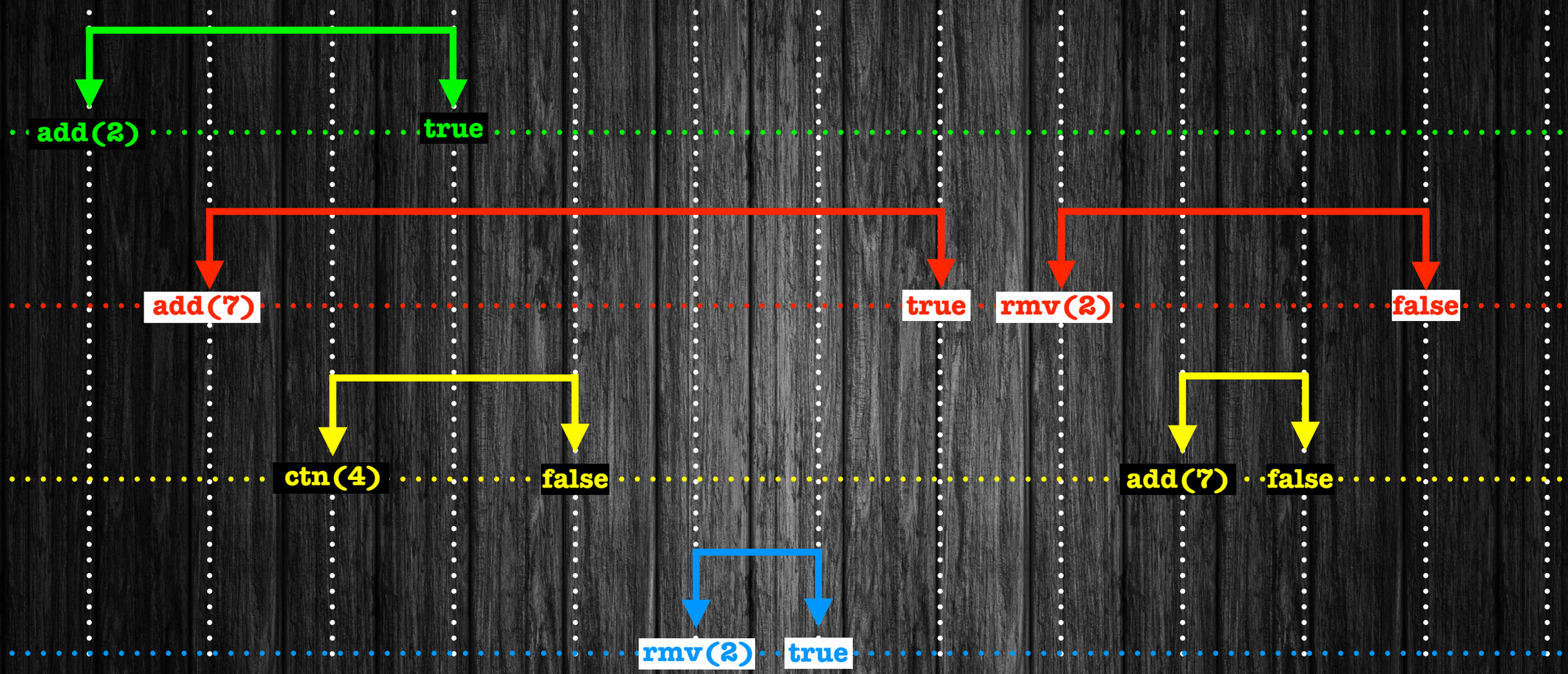


Concurrent Histories | Sets

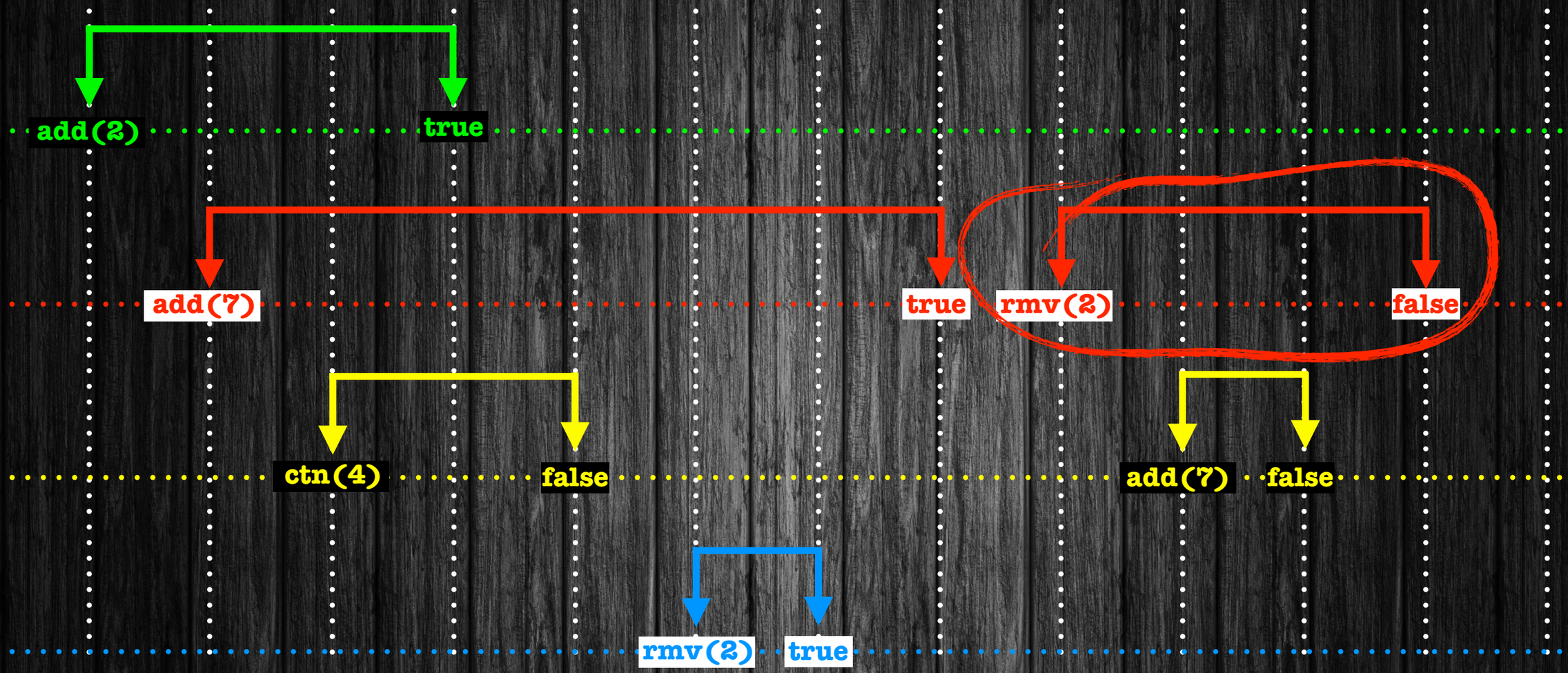




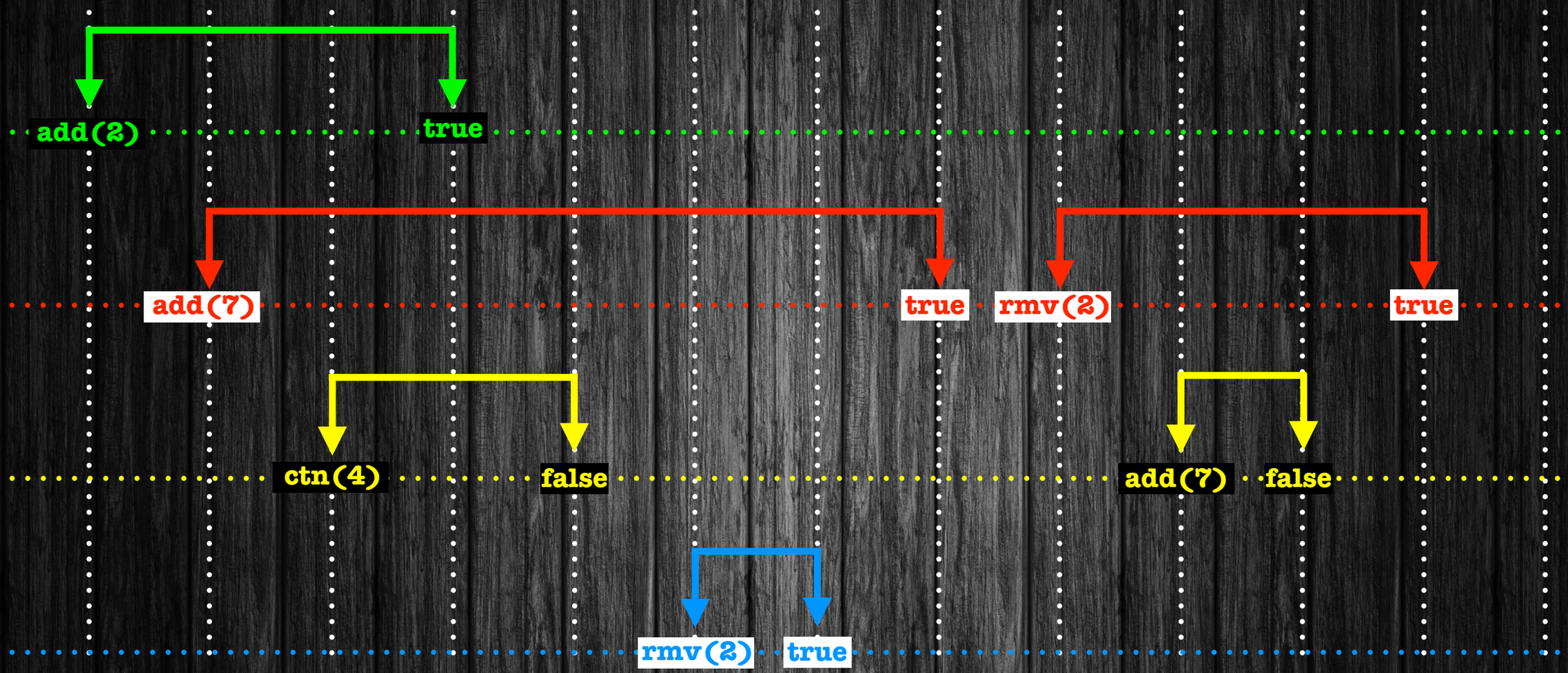
Concurrent Histories | Sets



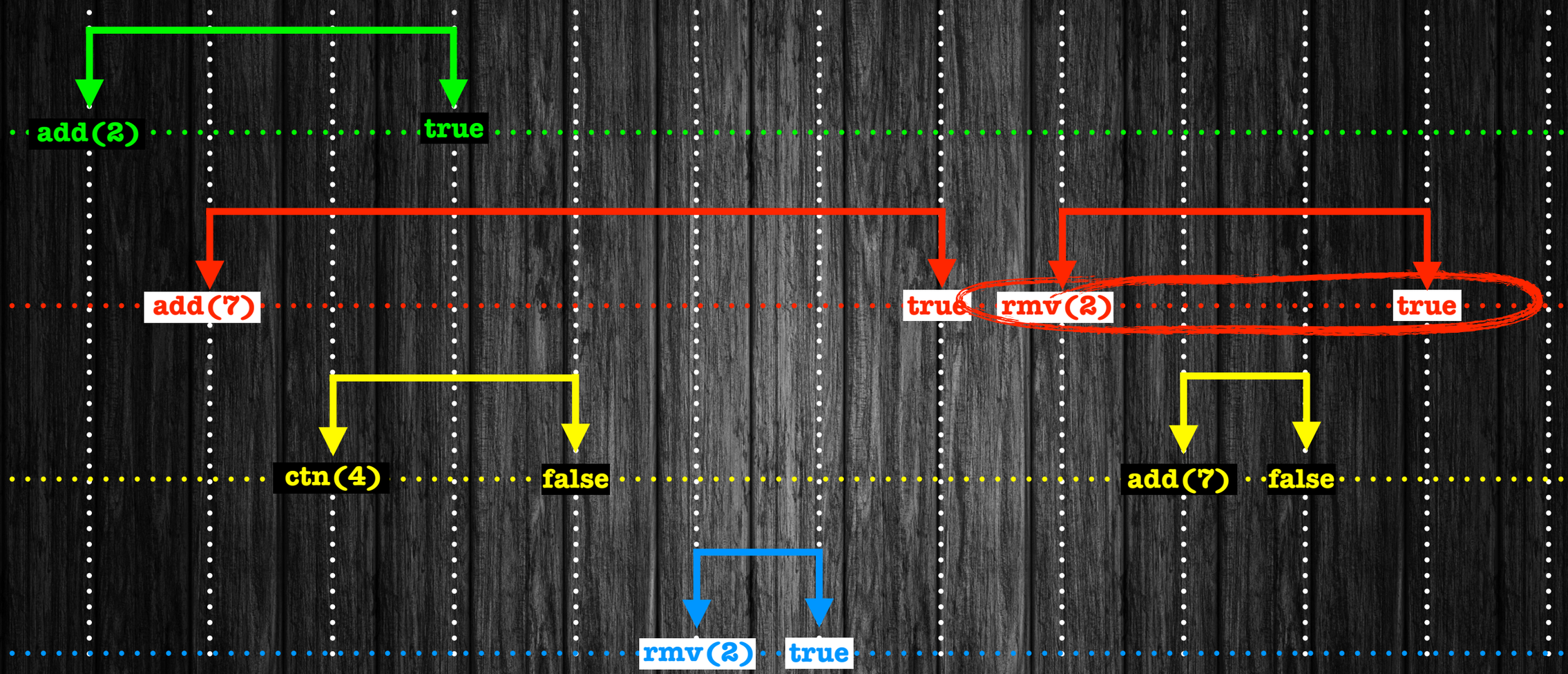
Concurrent Histories | Sets



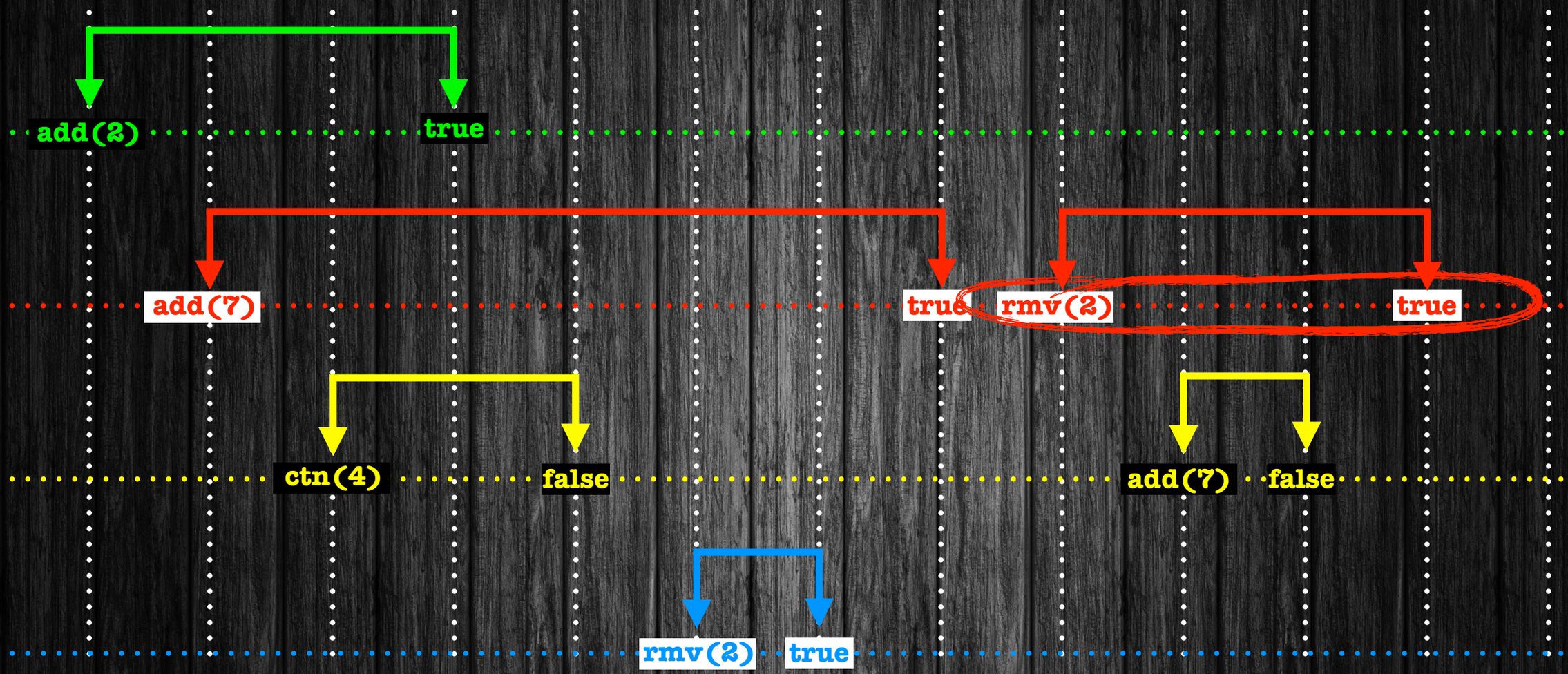
Concurrent Histories | Sets



Concurrent Histories | Sets

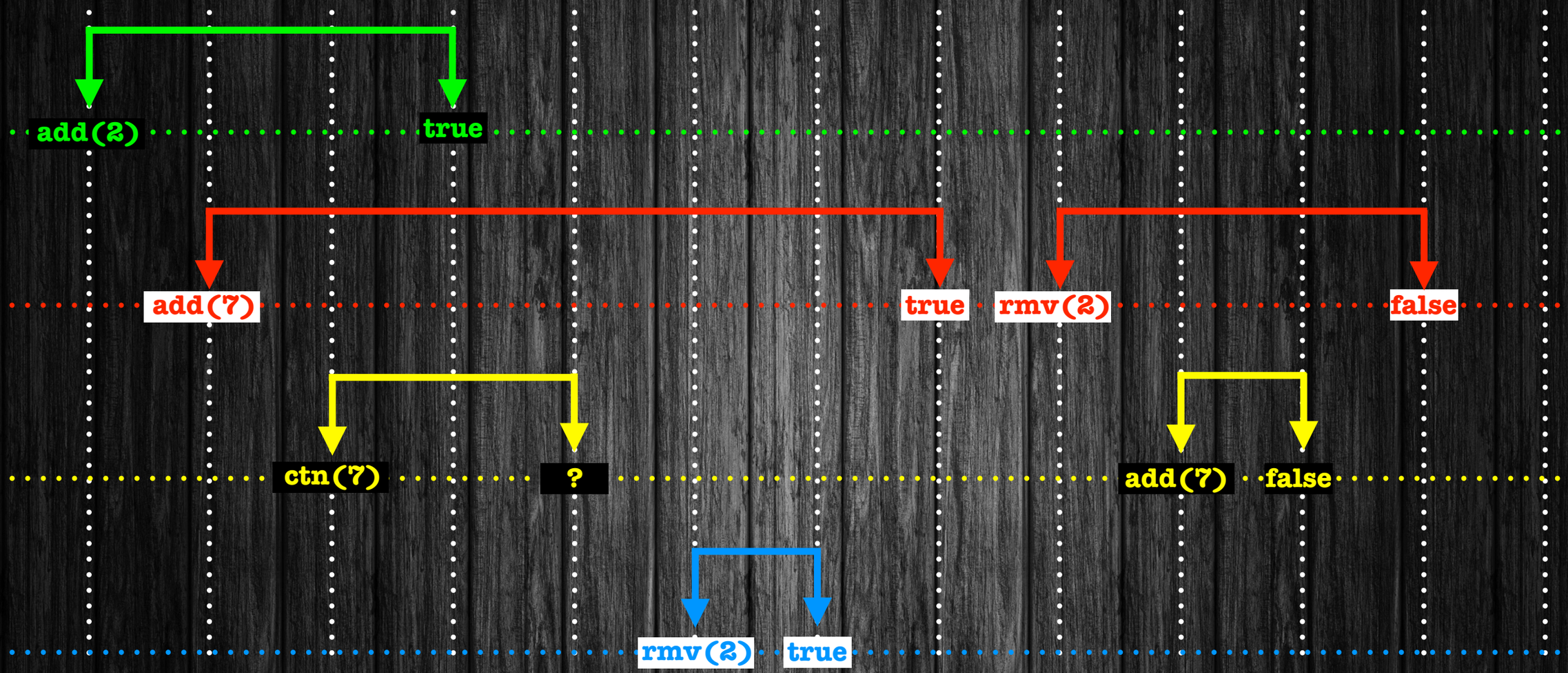


Concurrent Histories | Sets



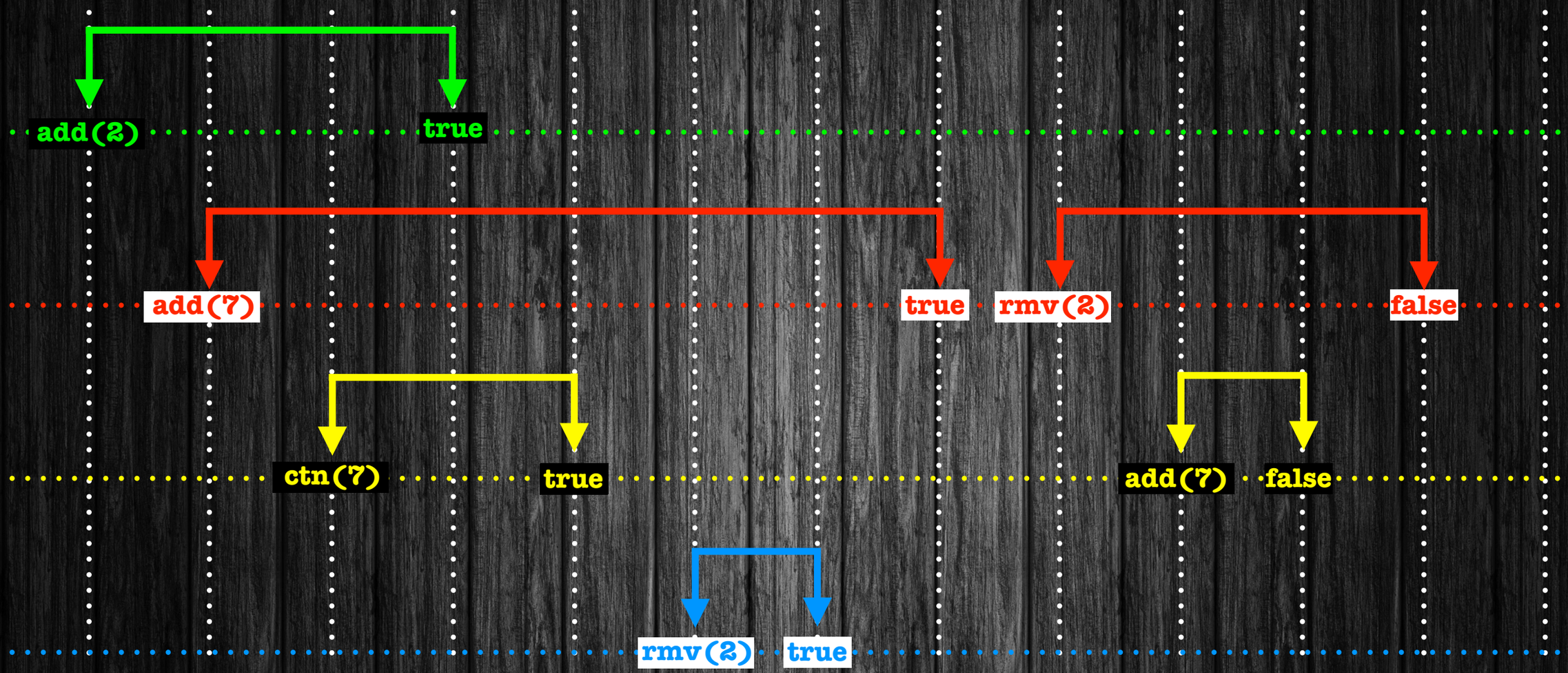


Concurrent Histories | Sets



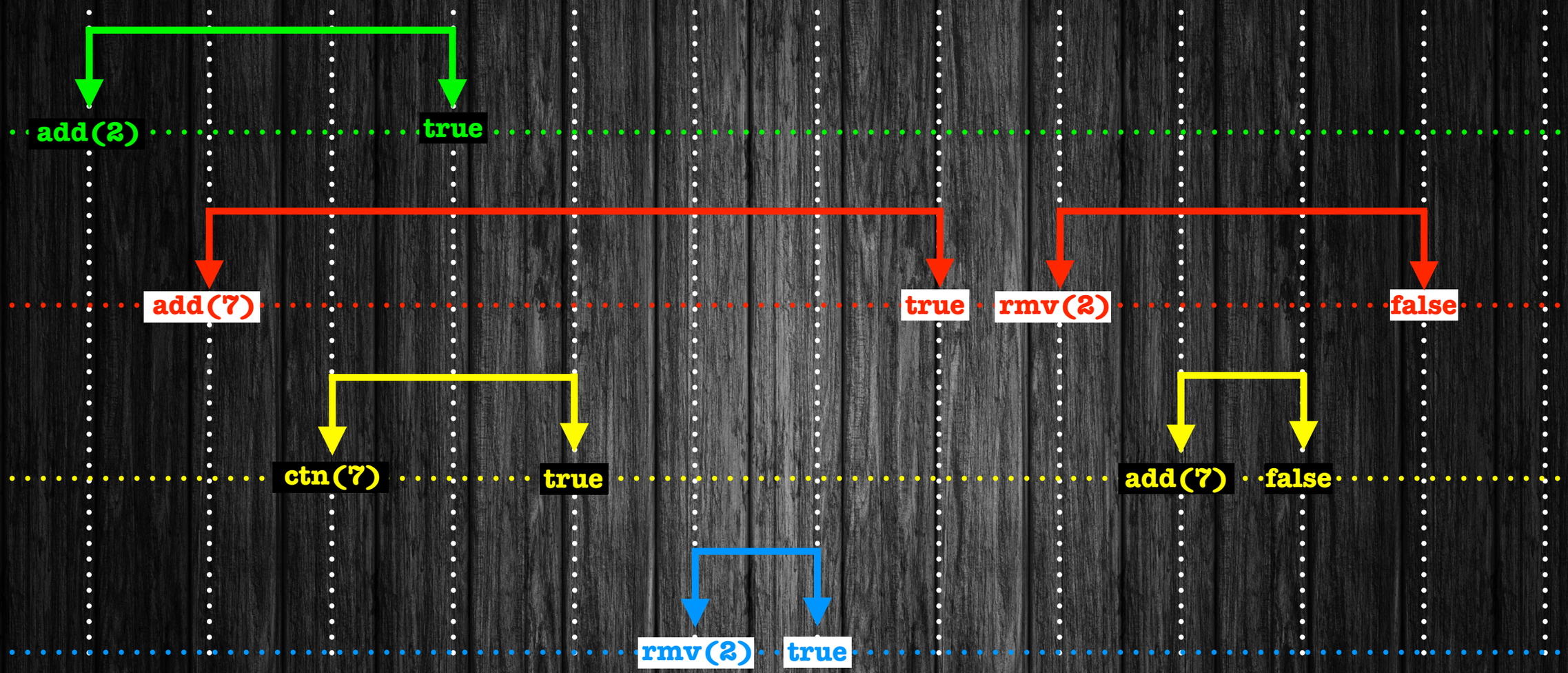
?

Concurrent Histories | Sets



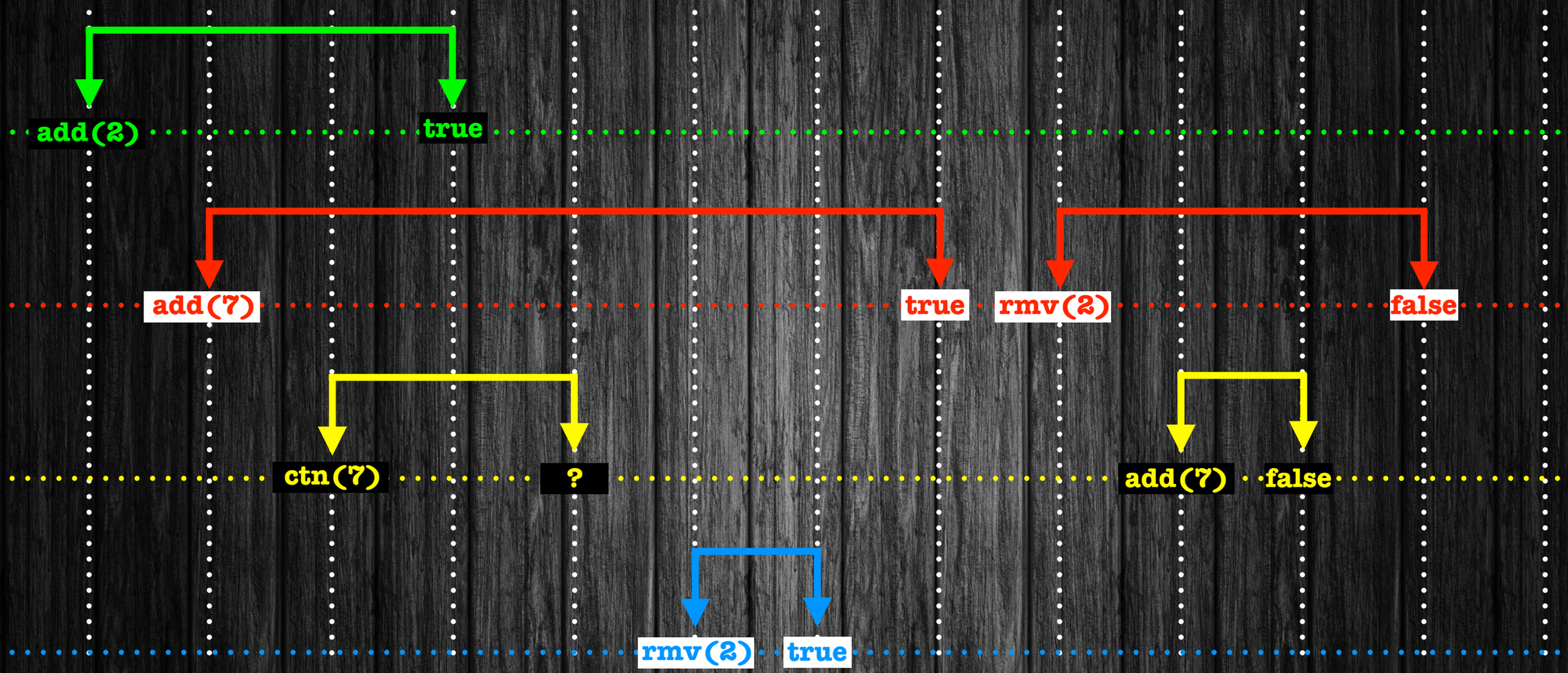
?

Concurrent Histories | Sets



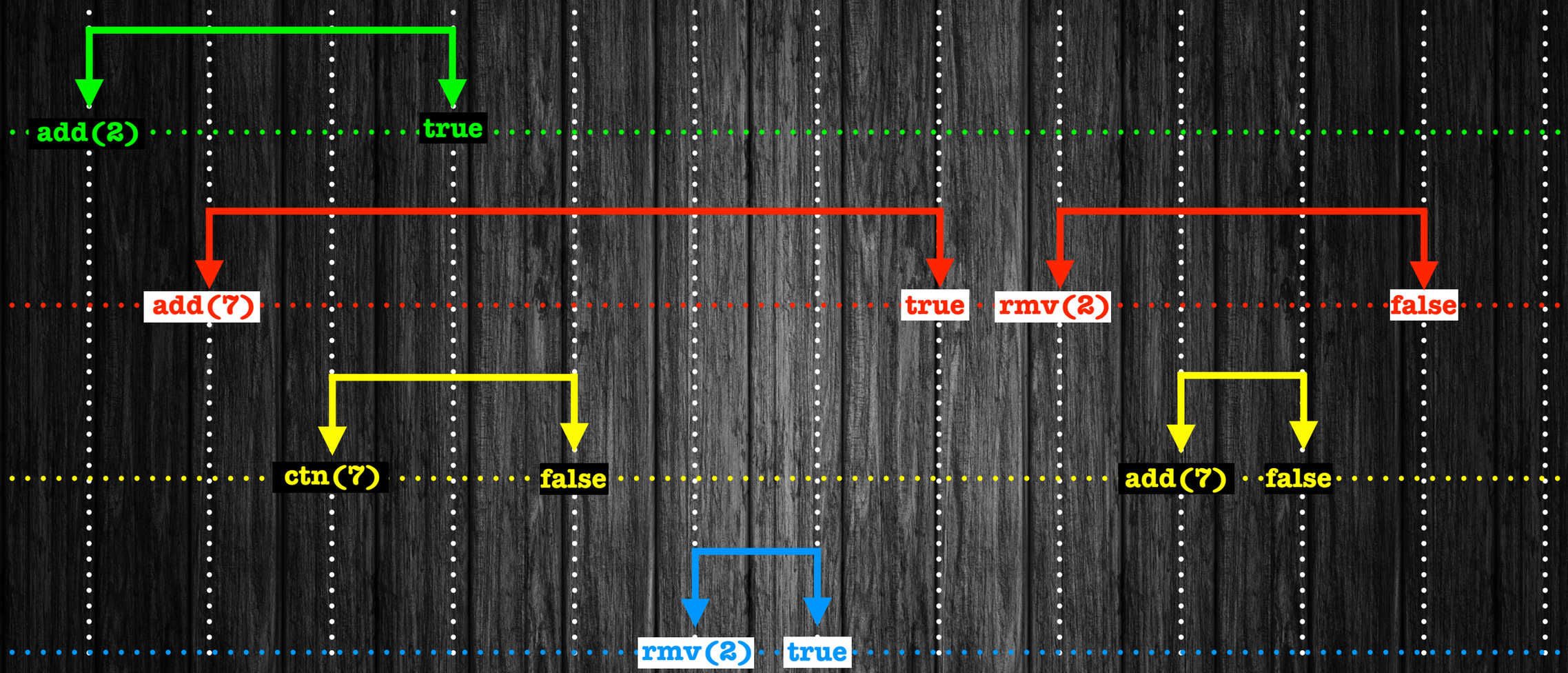
✔

Concurrent Histories | Sets



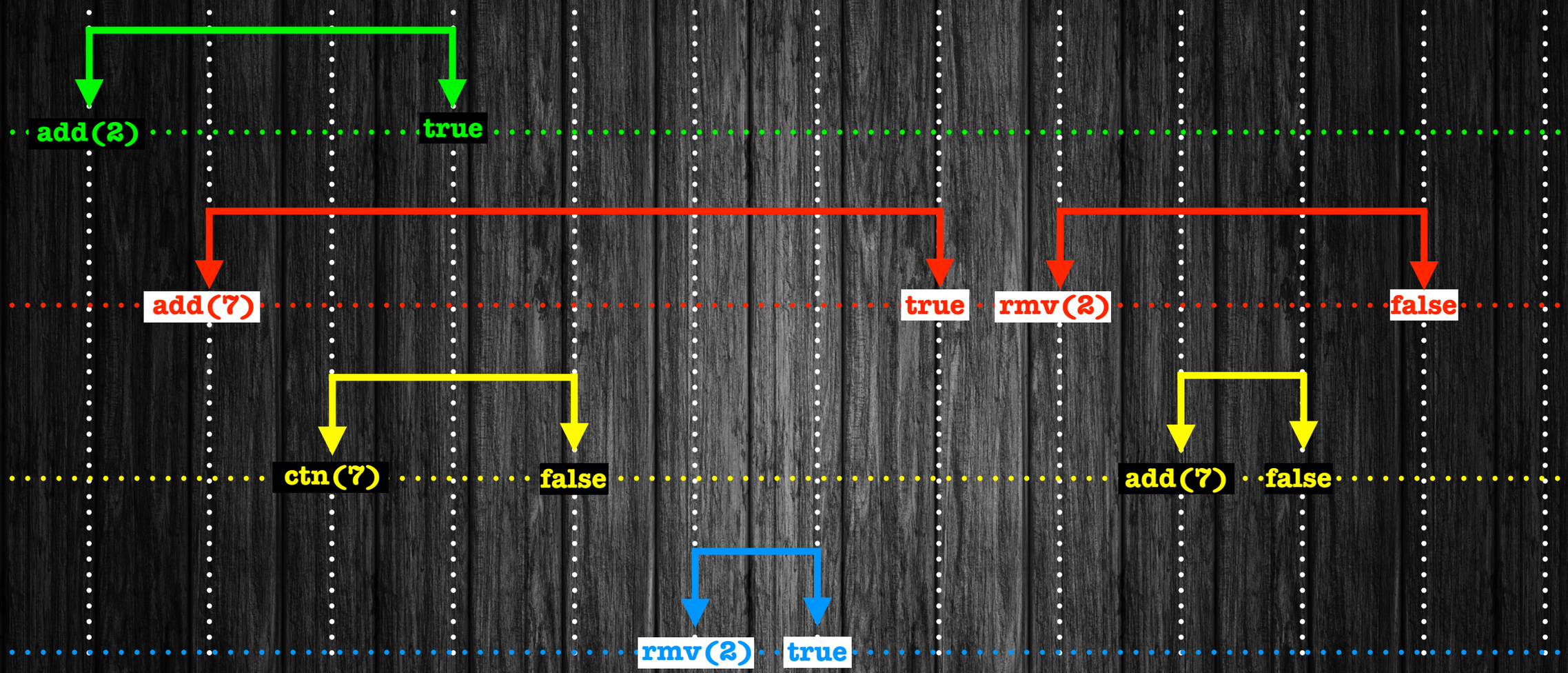
?

Concurrent Histories | Sets



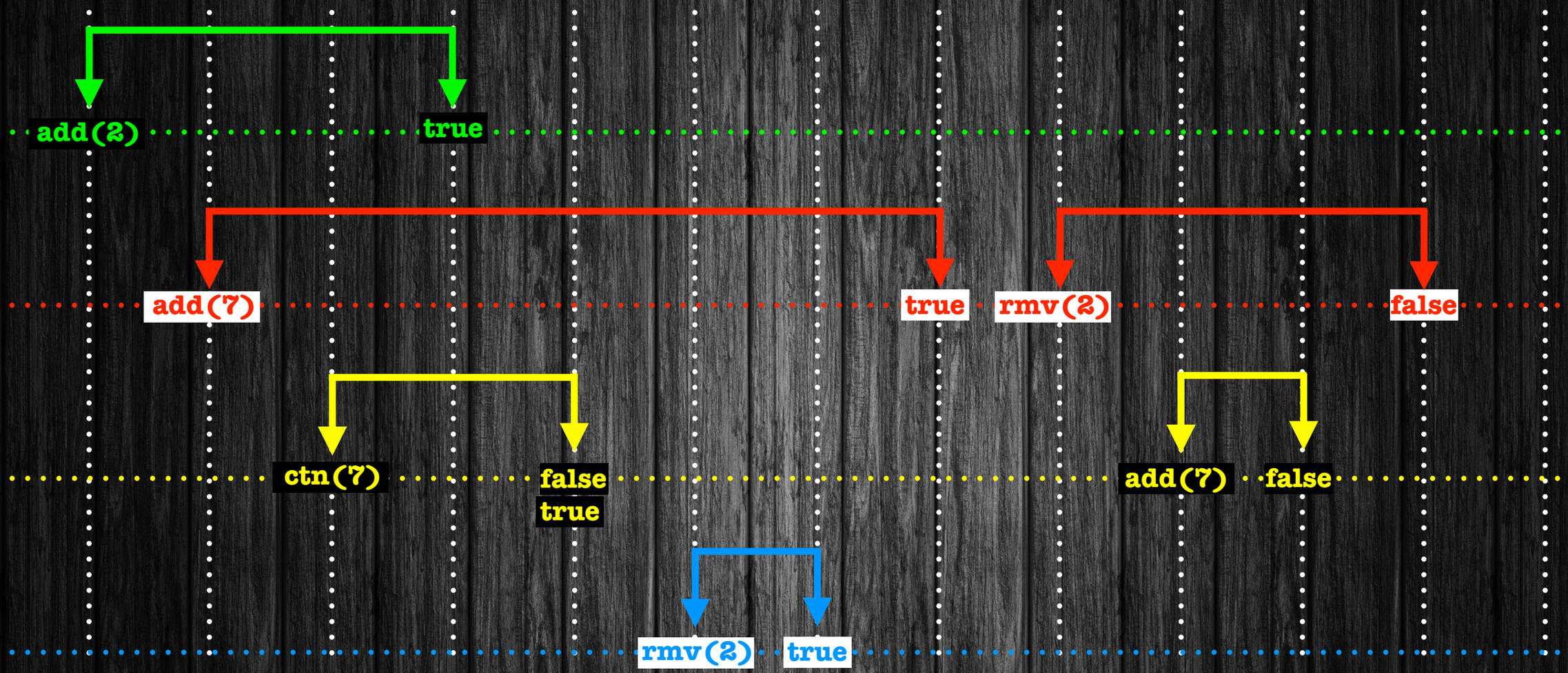
?

Concurrent Histories | Sets



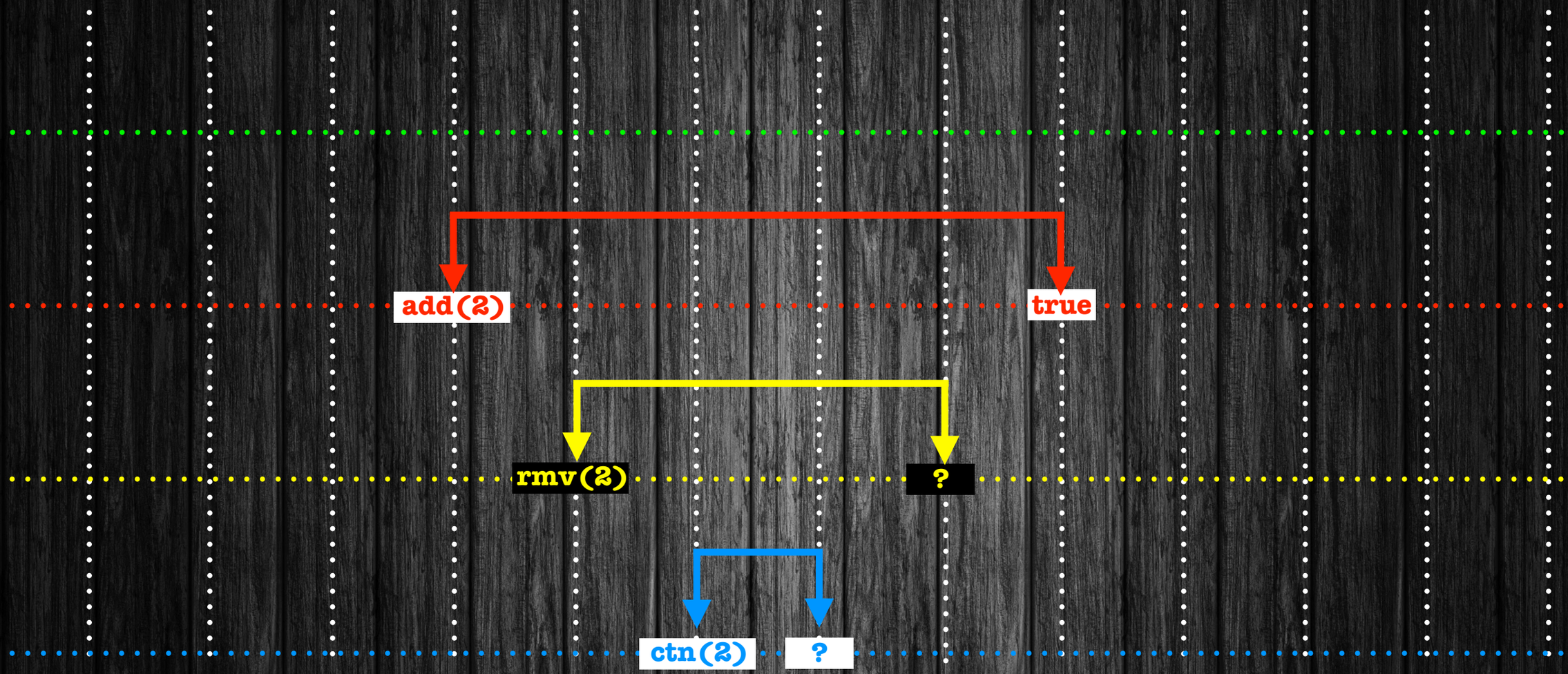
✔

Concurrent Histories | Sets

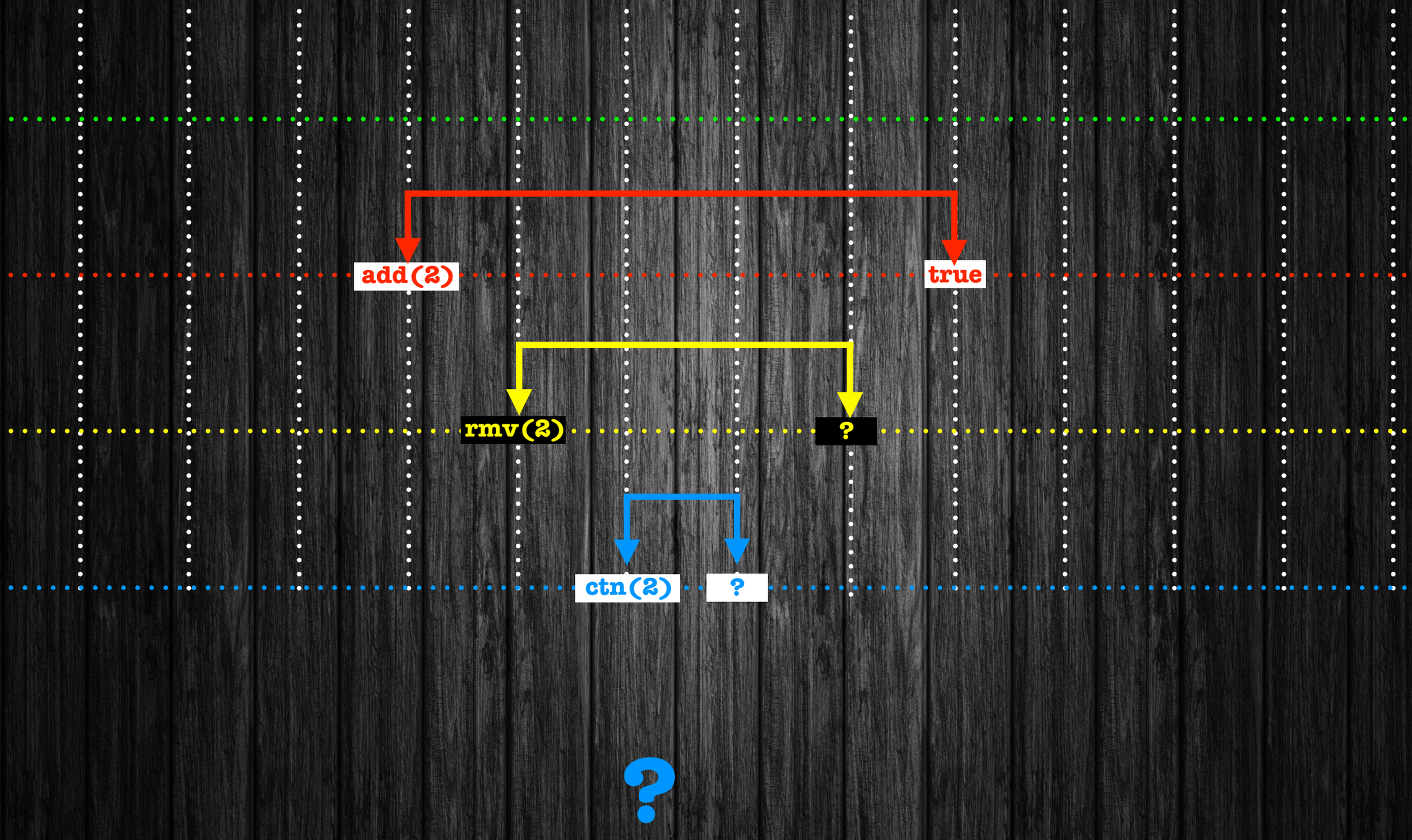


✔

Concurrent Histories | Sets

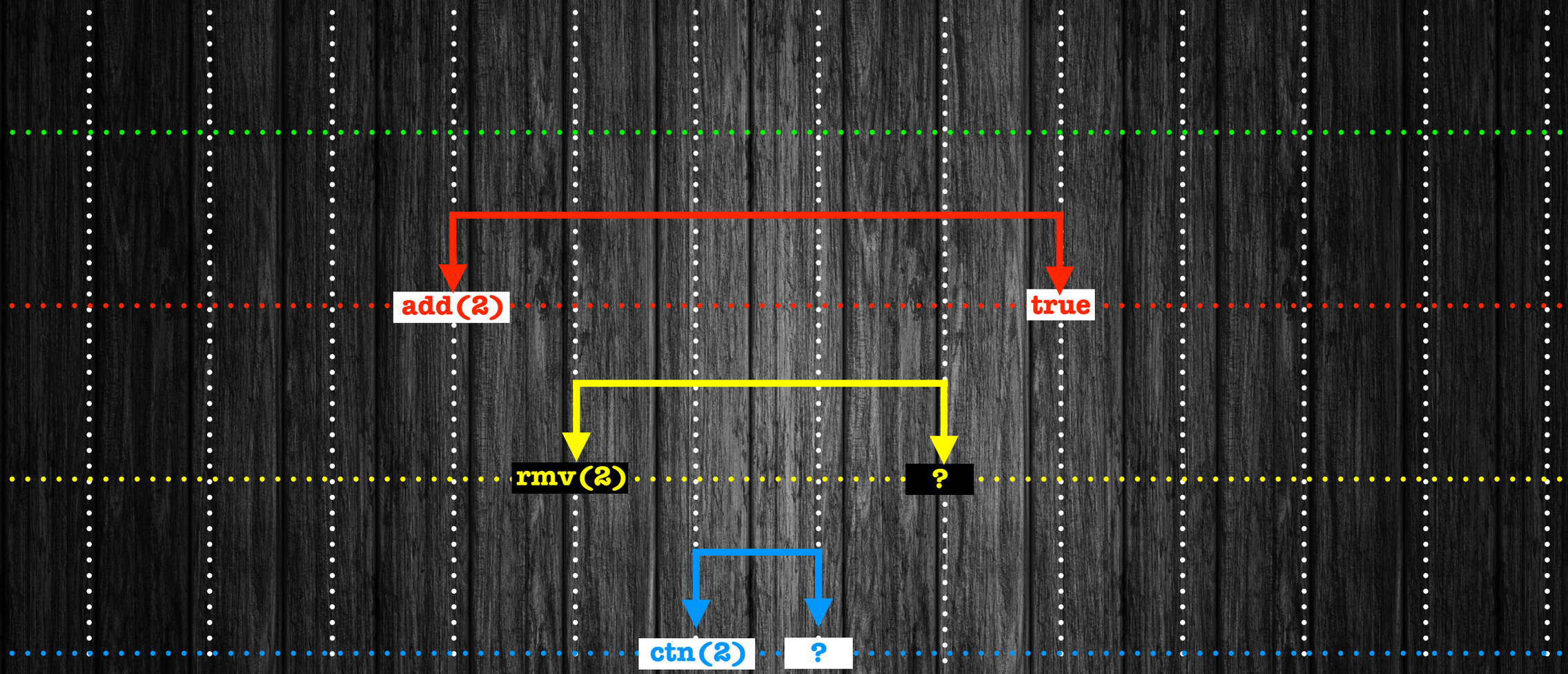


Concurrent Histories | Sets



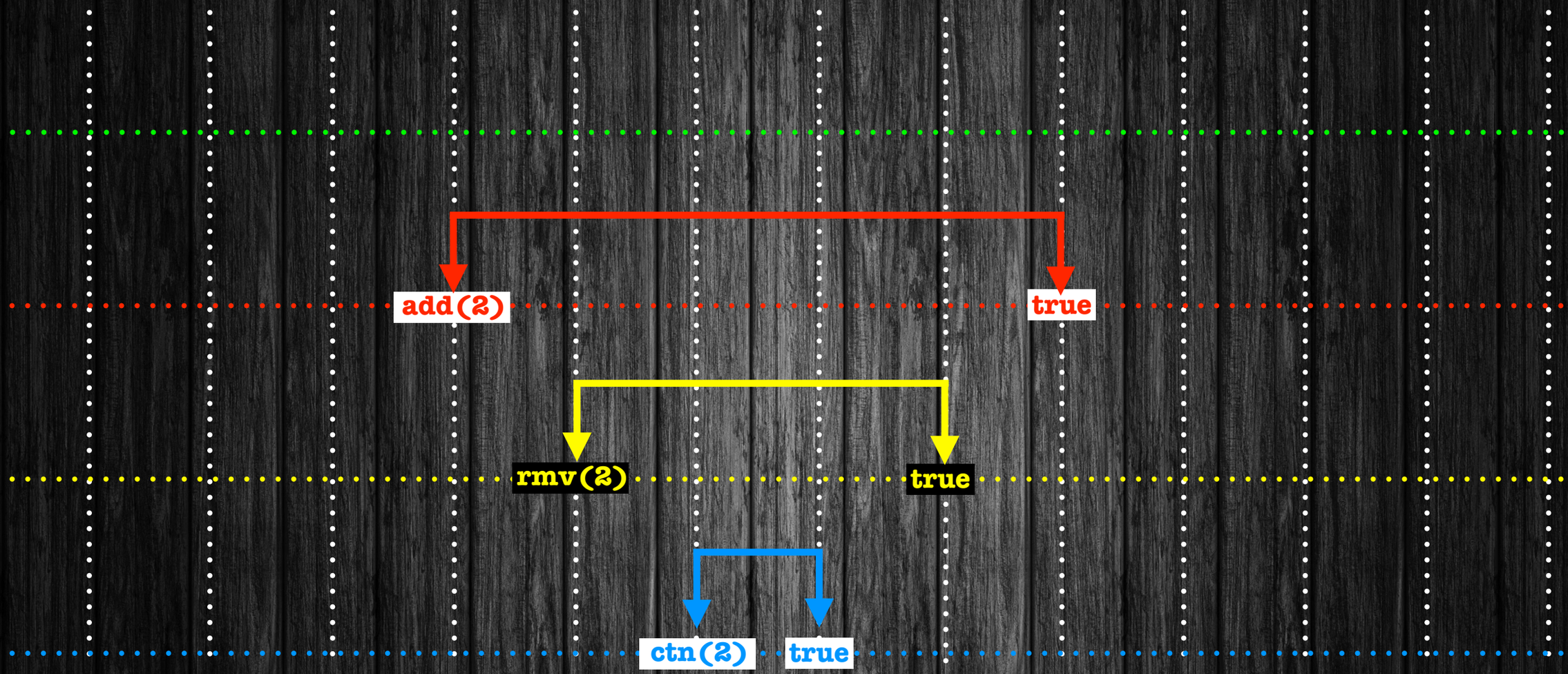
?

Concurrent Histories | Sets

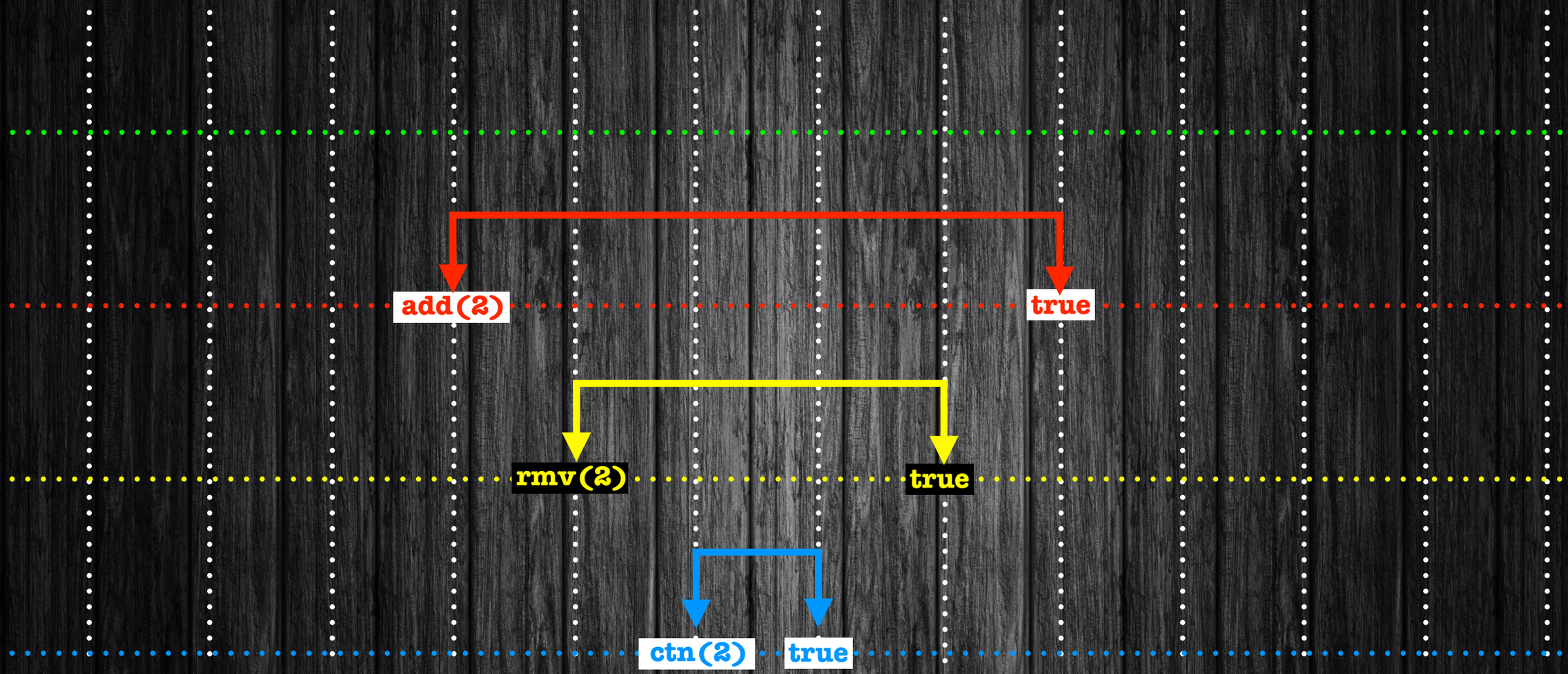


? ?

Concurrent Histories | Sets

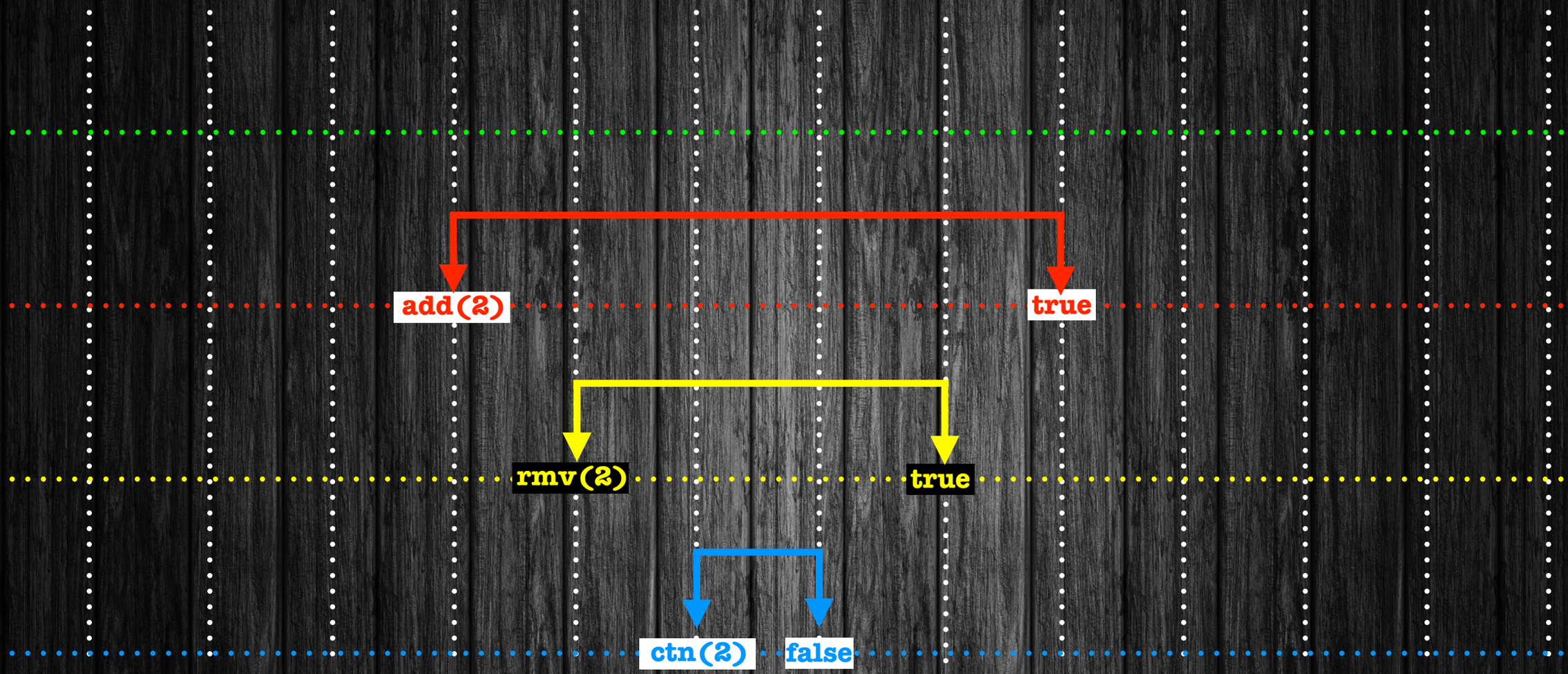


Concurrent Histories | Sets

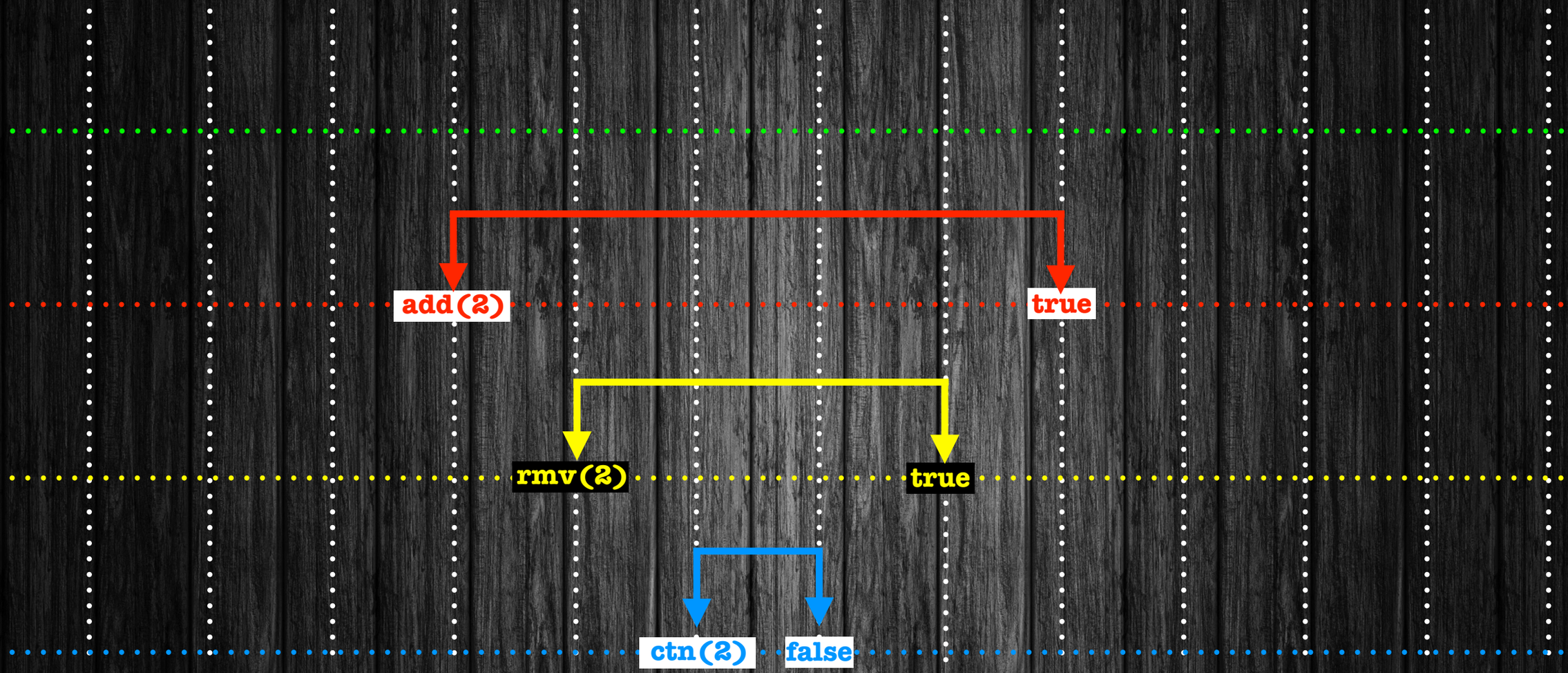


✔

Concurrent Histories | Sets

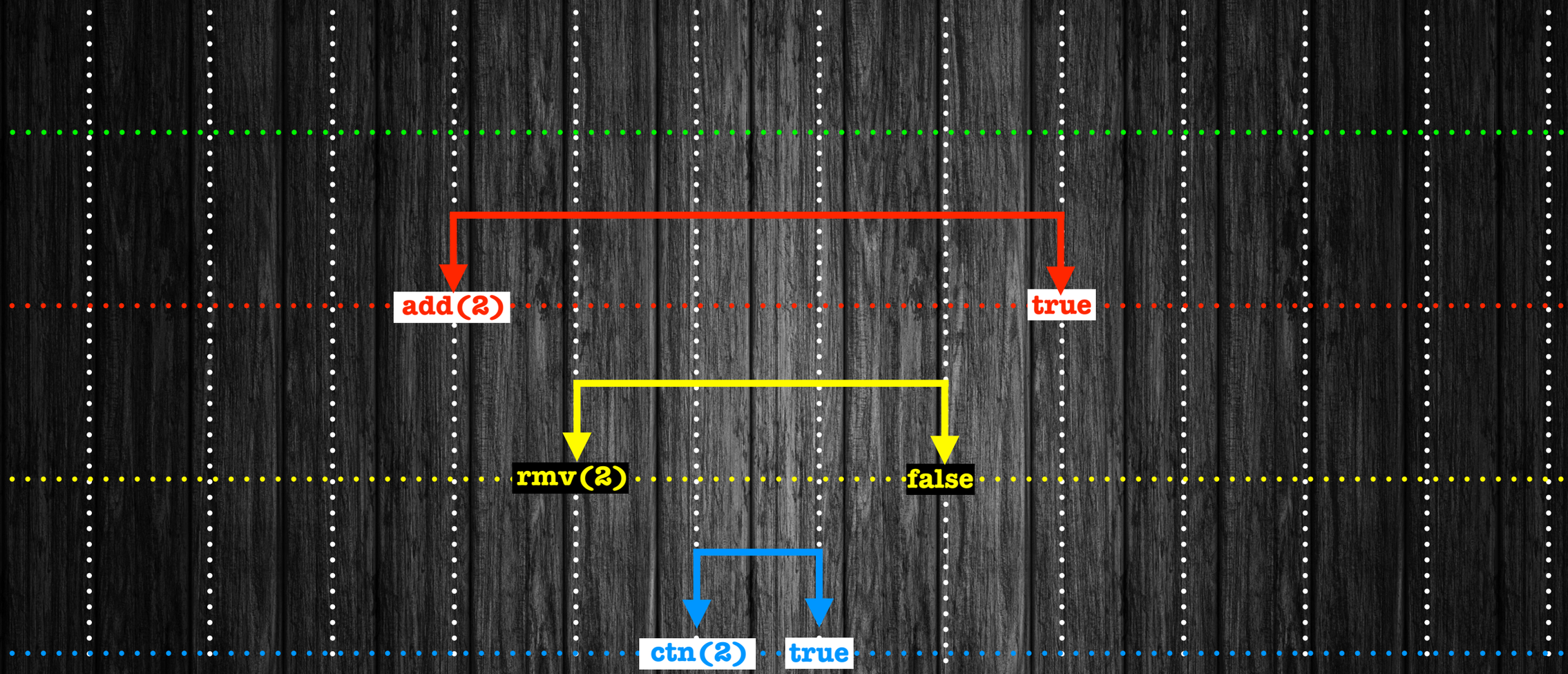


Concurrent Histories Sets

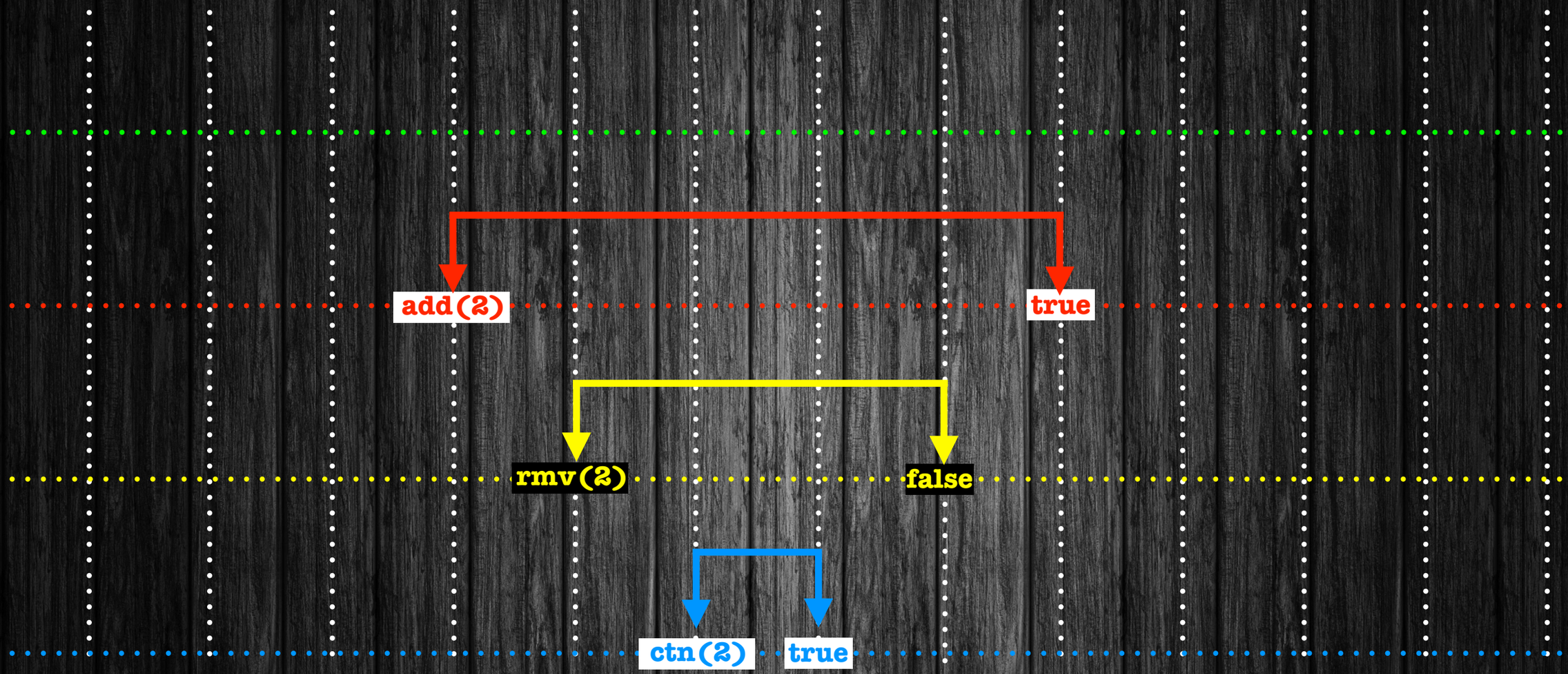


✔

Concurrent Histories | Sets

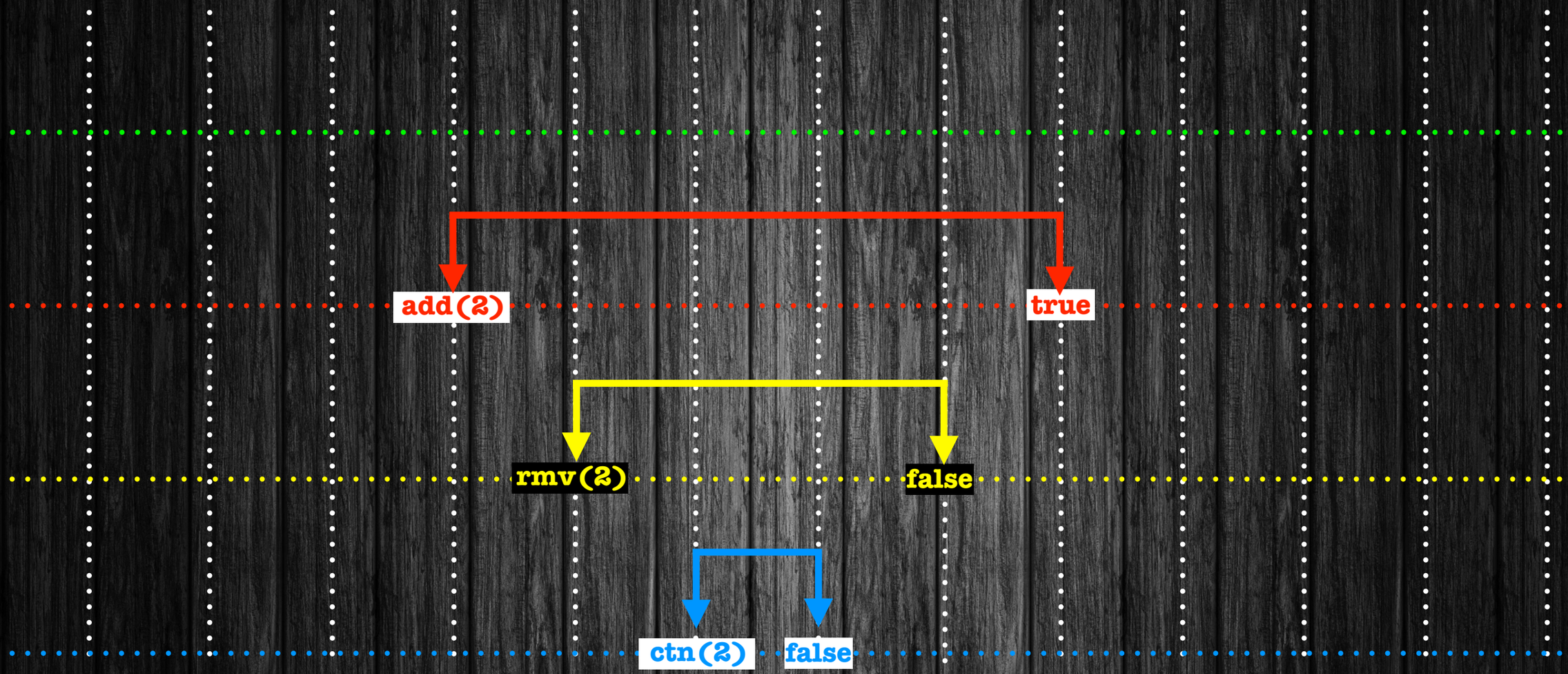


Concurrent Histories | Sets

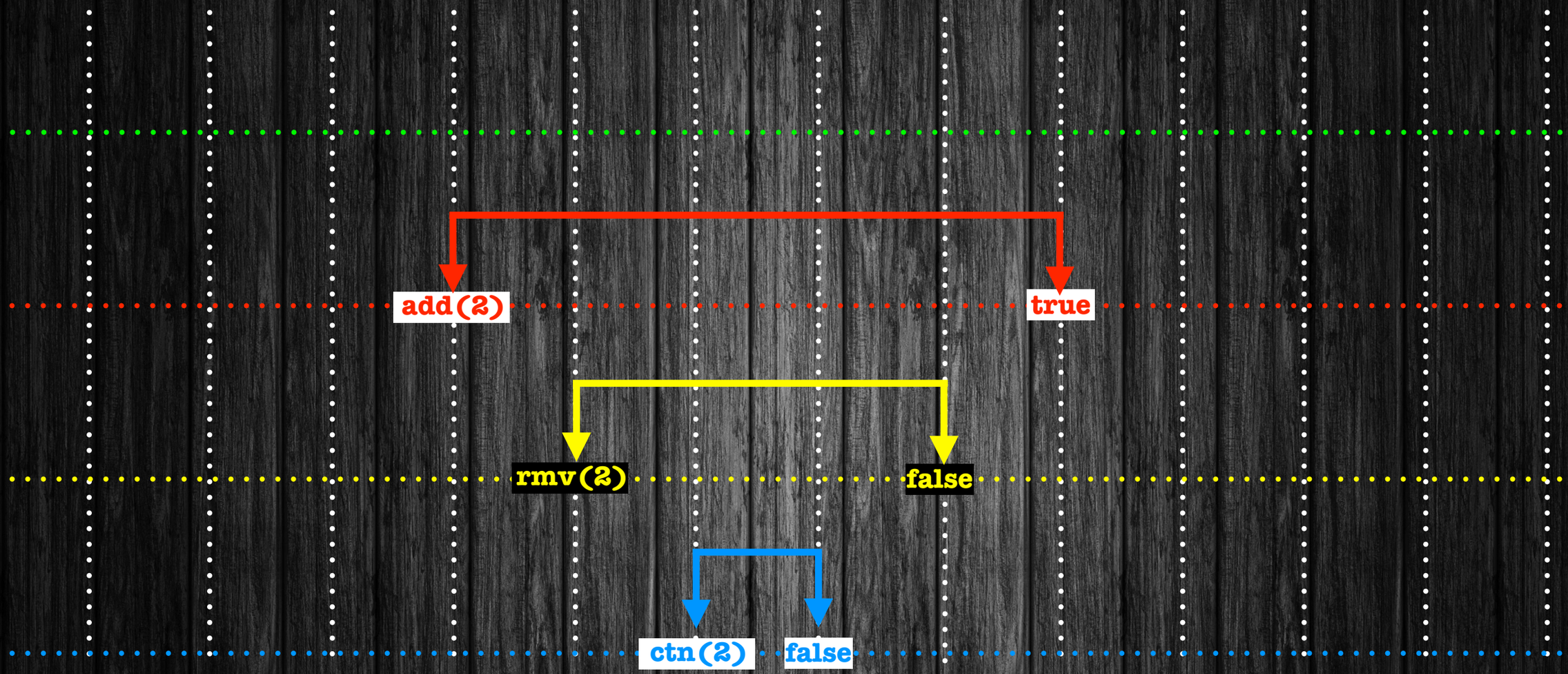


✔

Concurrent Histories | Sets

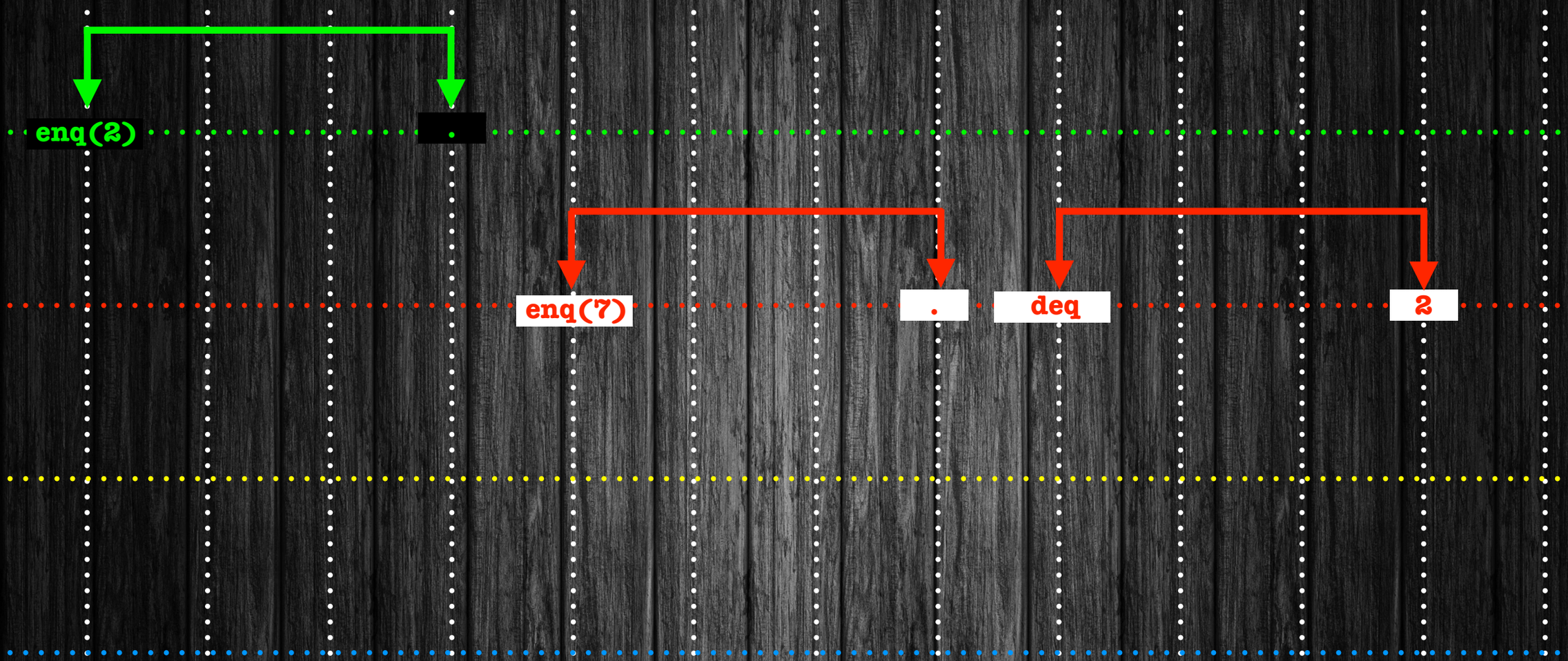


Concurrent Histories | Sets

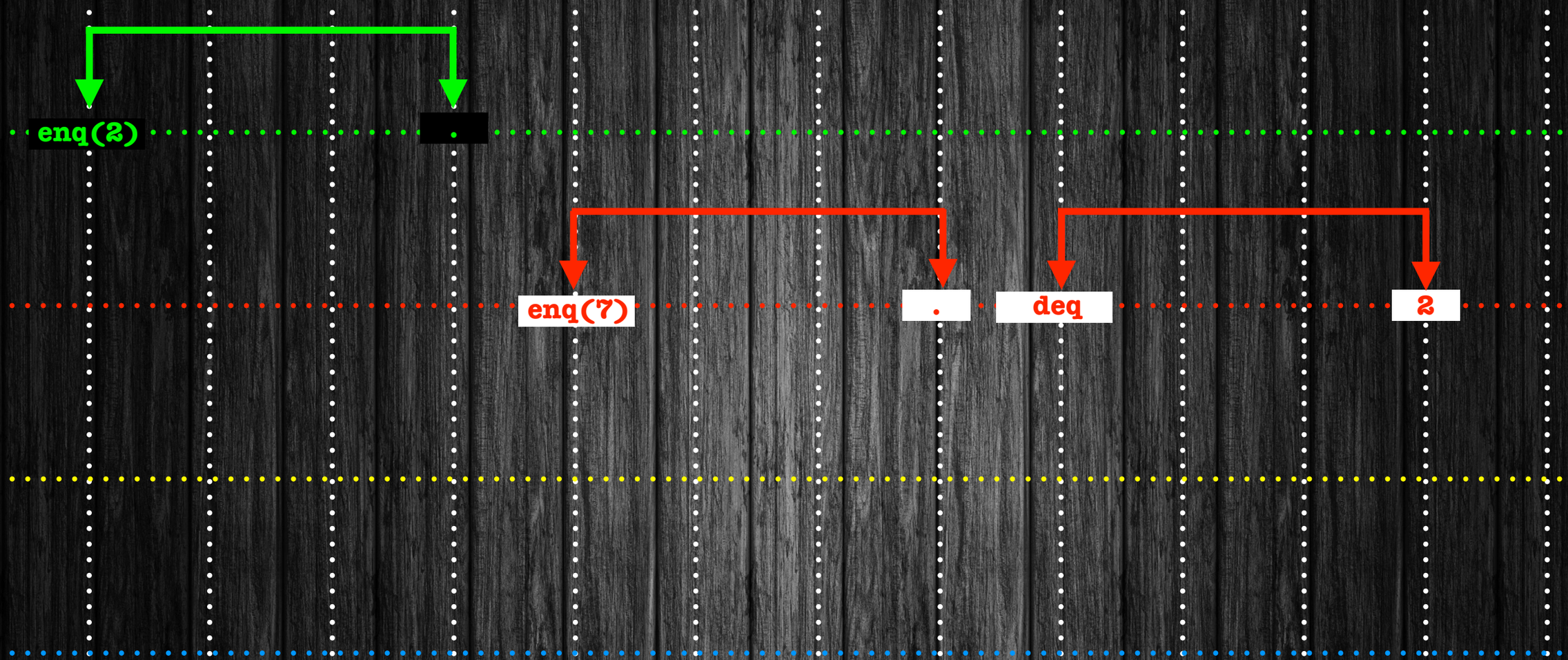


✔

Concurrent Histories | Queues

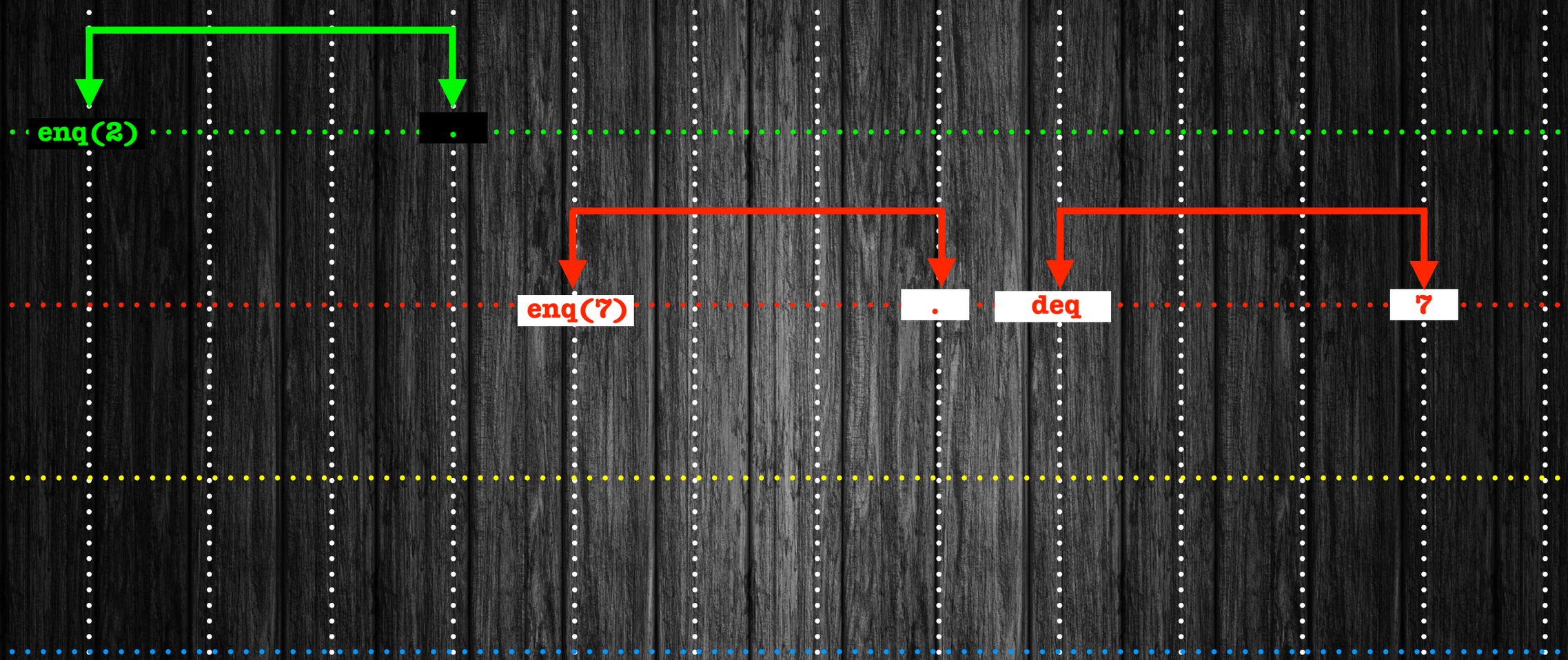


Concurrent Histories | Queues

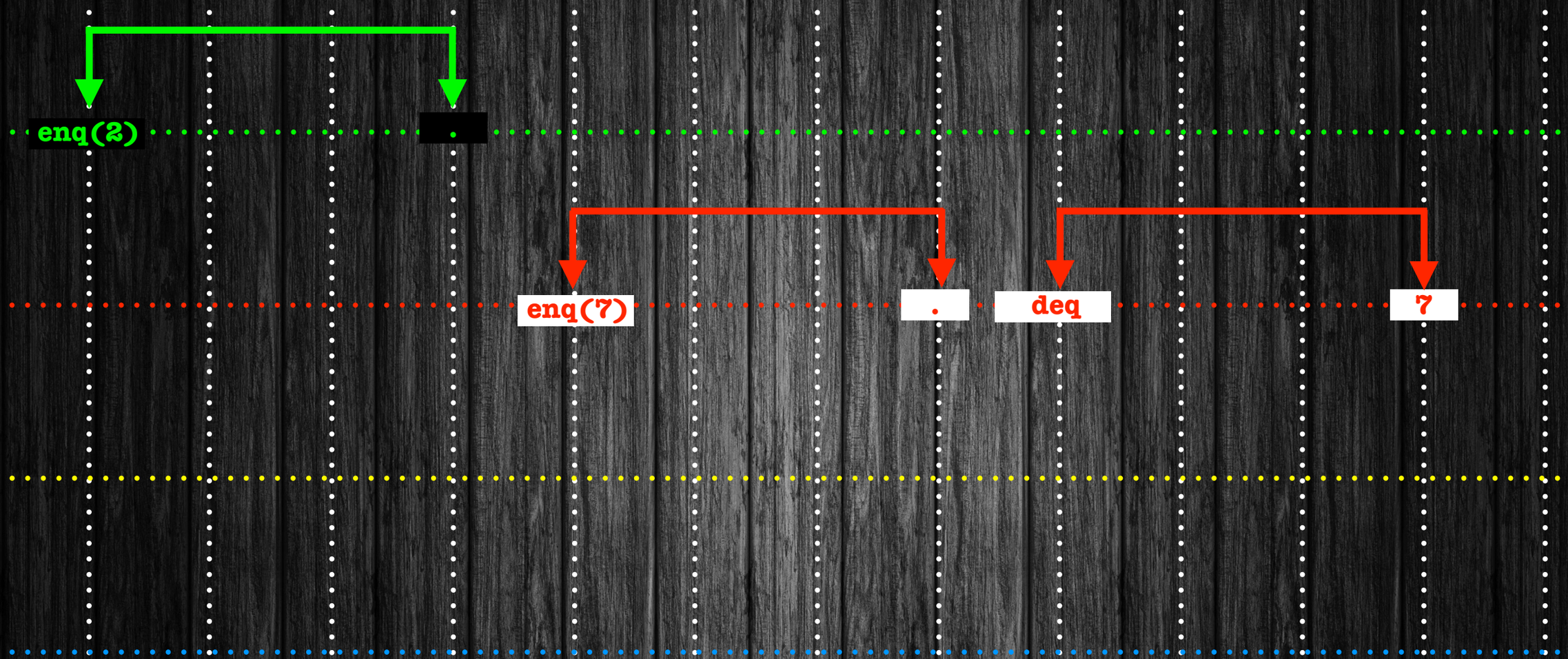


✔

Concurrent Histories | Queues

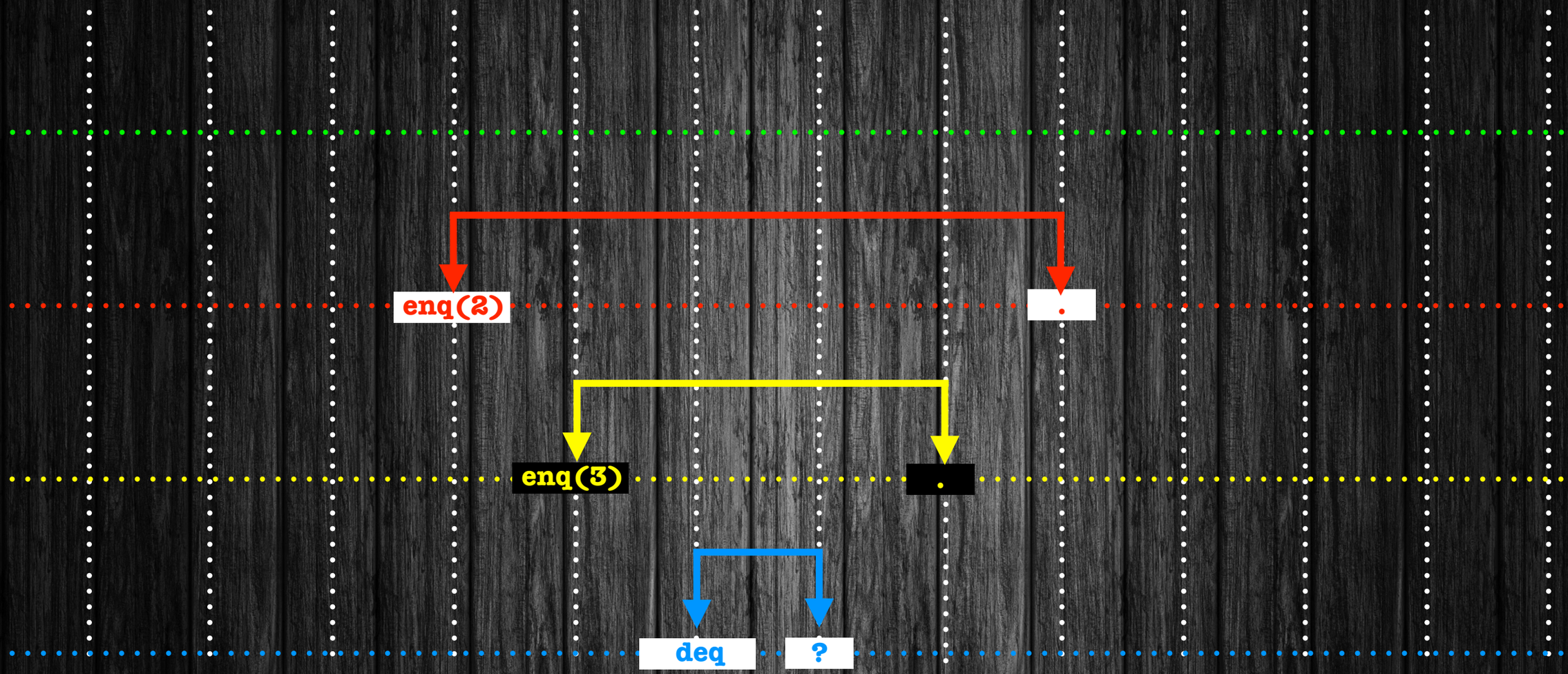


Concurrent Histories | Queues

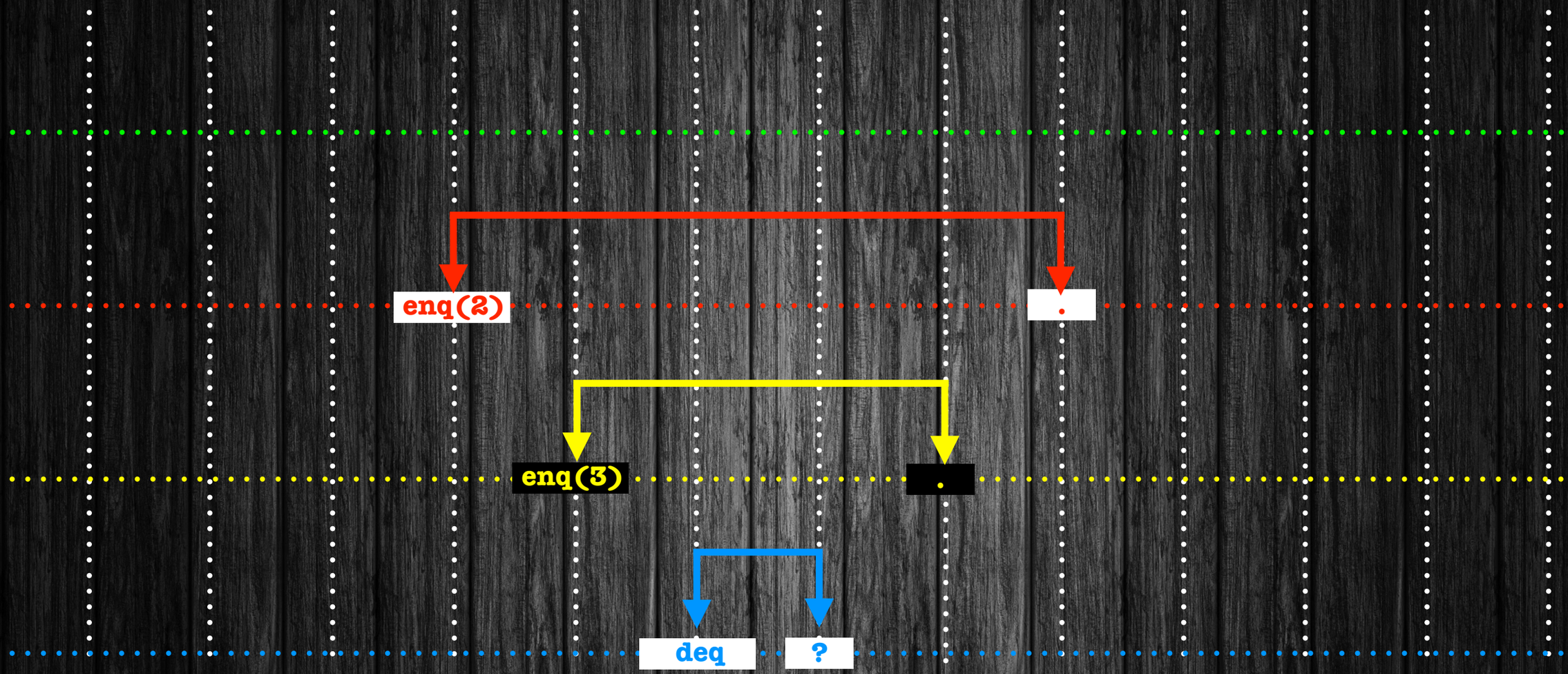


X

Concurrent Histories | Queues

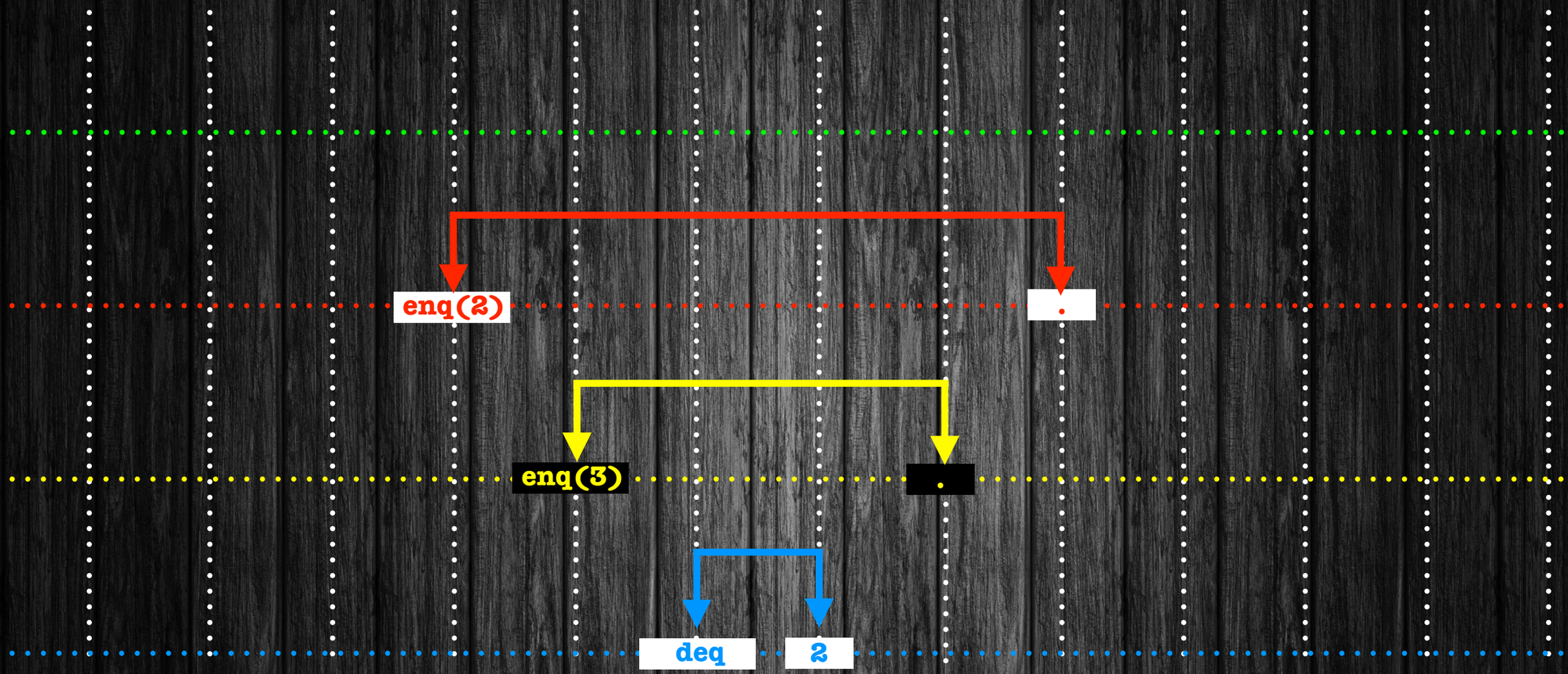


Concurrent Histories | Queues

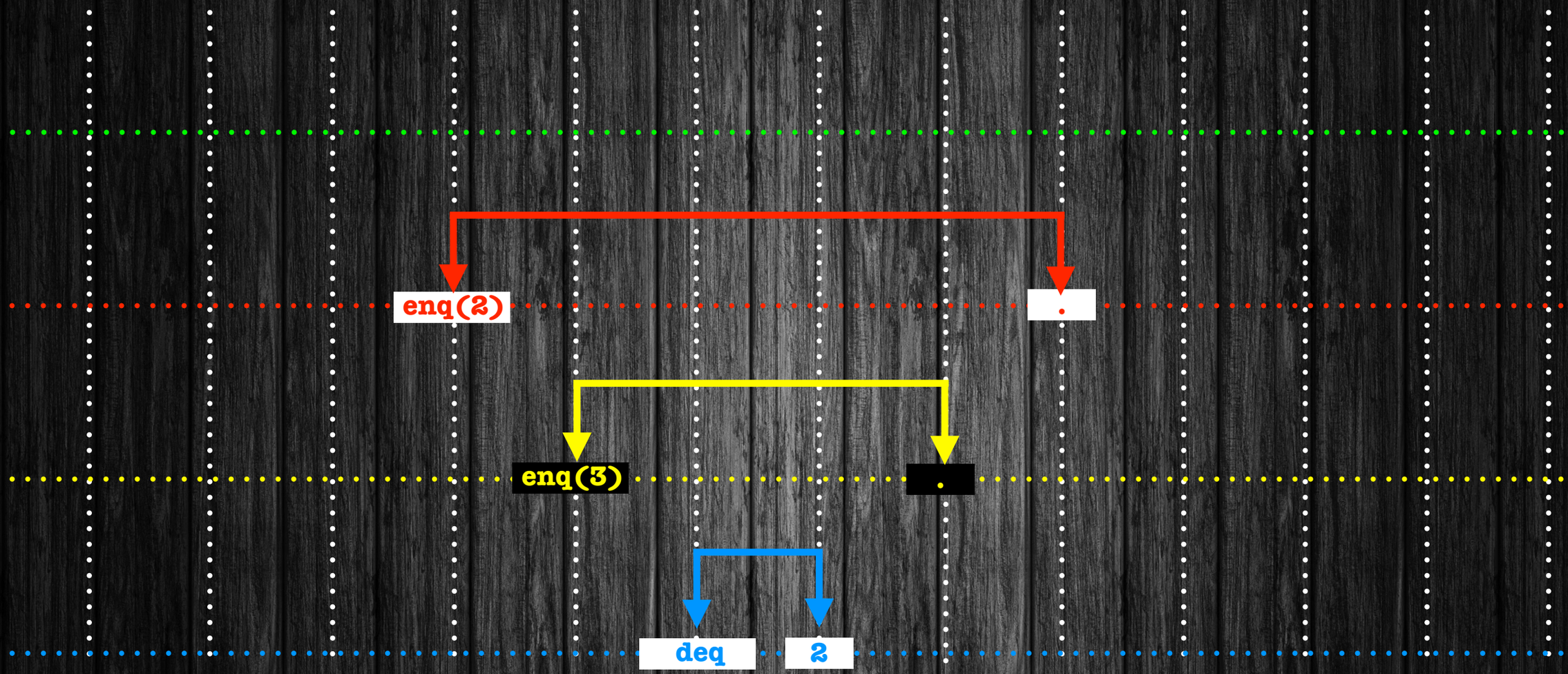


?

Concurrent Histories | Queues

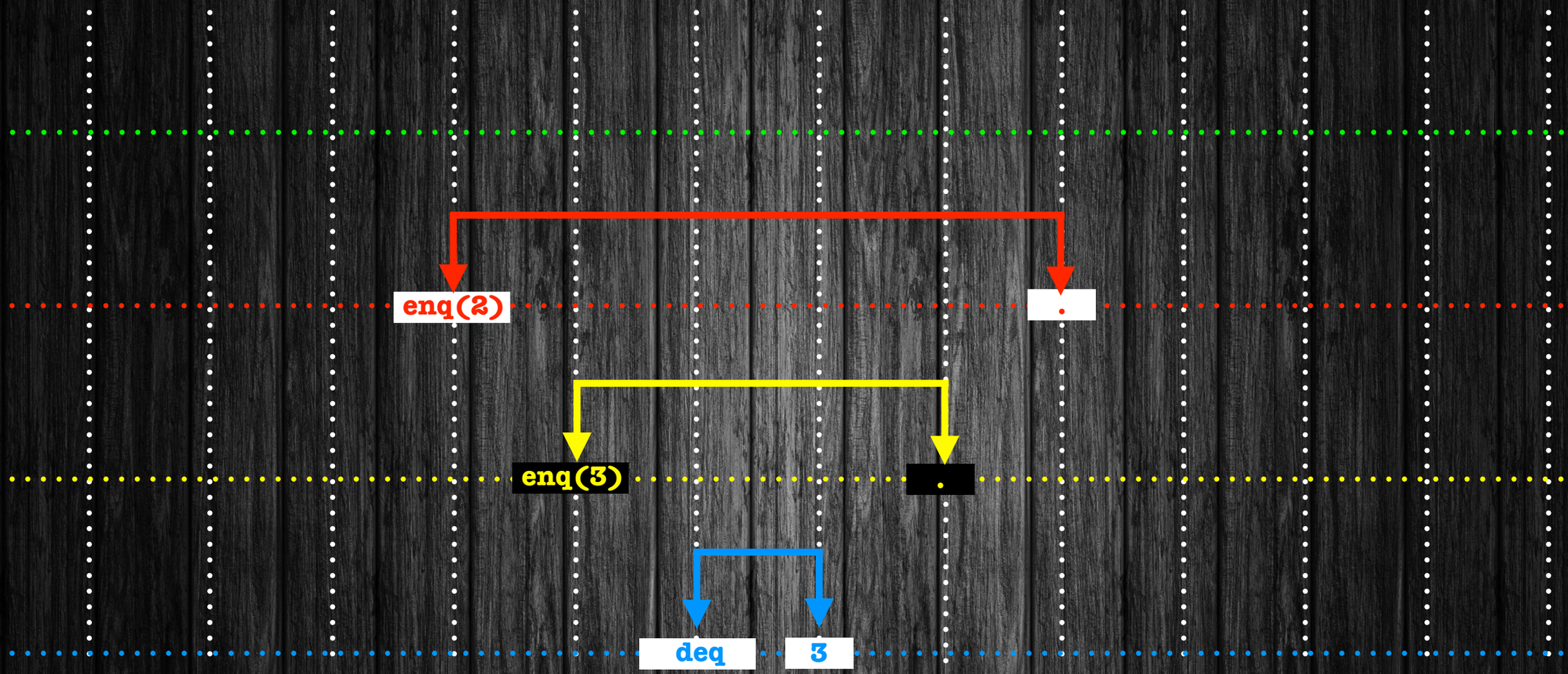


Concurrent Histories | Queues

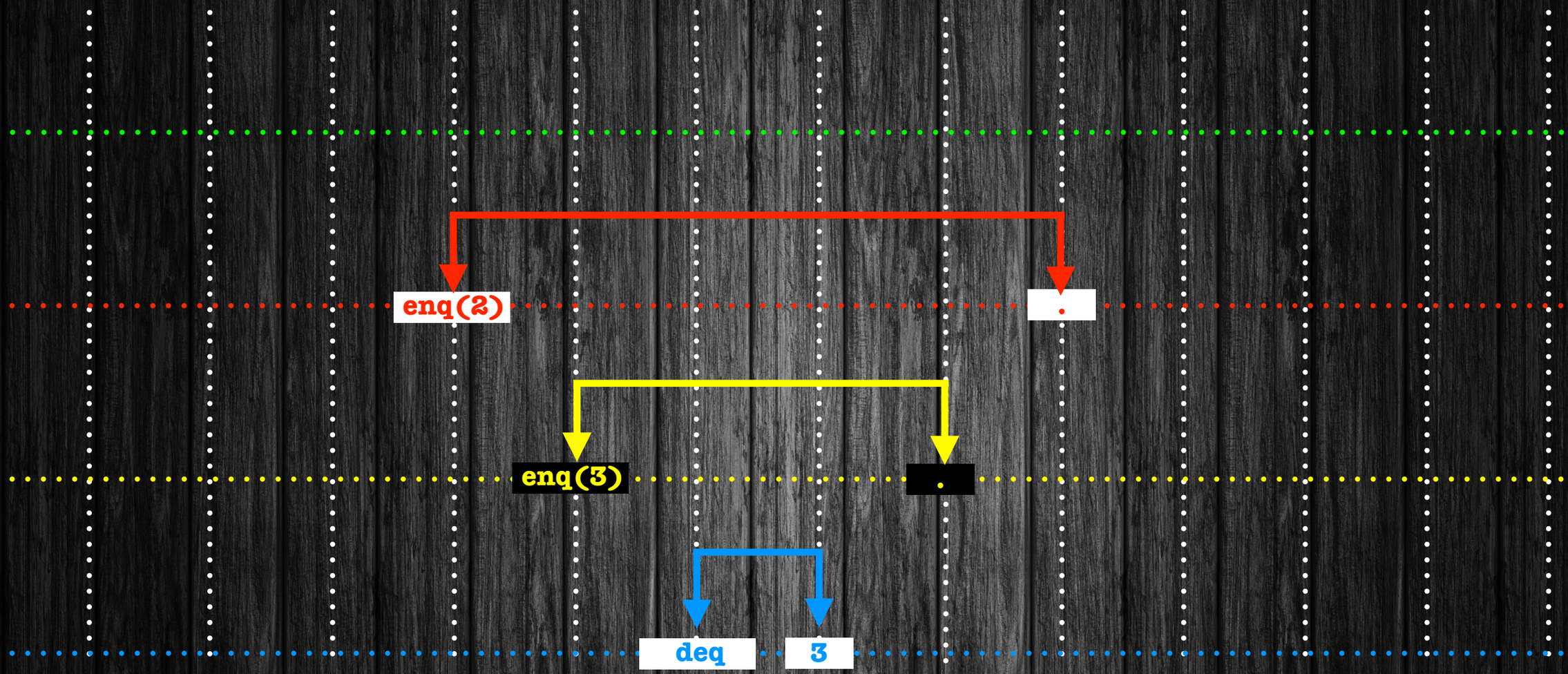


✔

Concurrent Histories | Queues

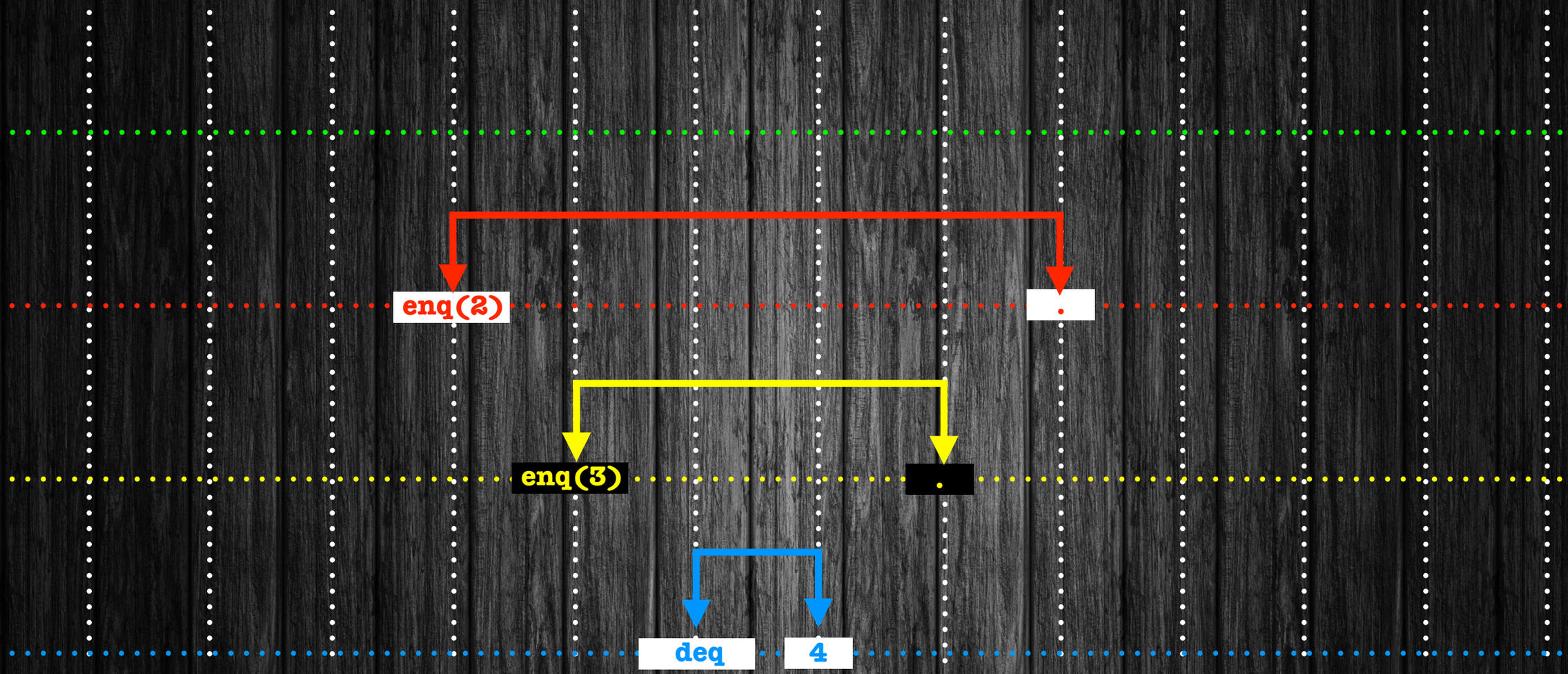


Concurrent Histories | Queues

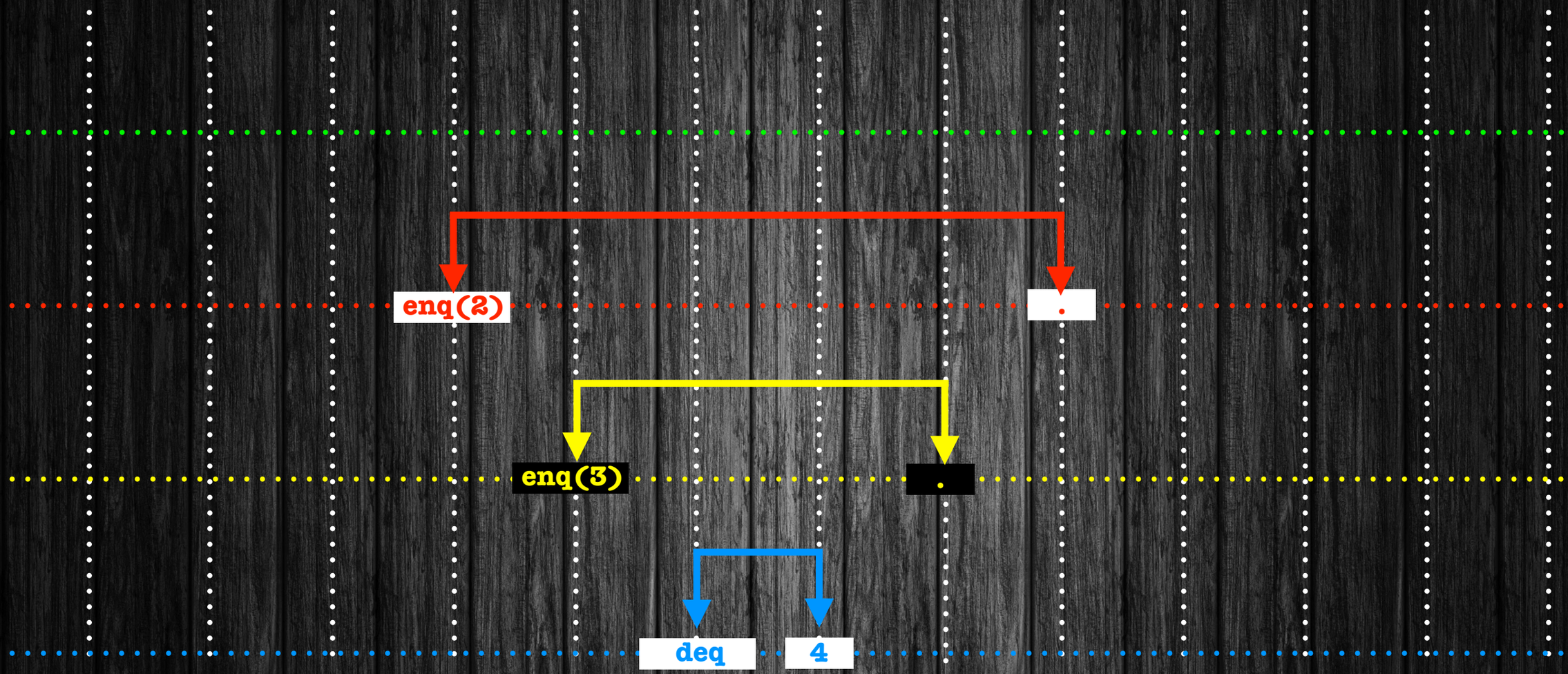


✔

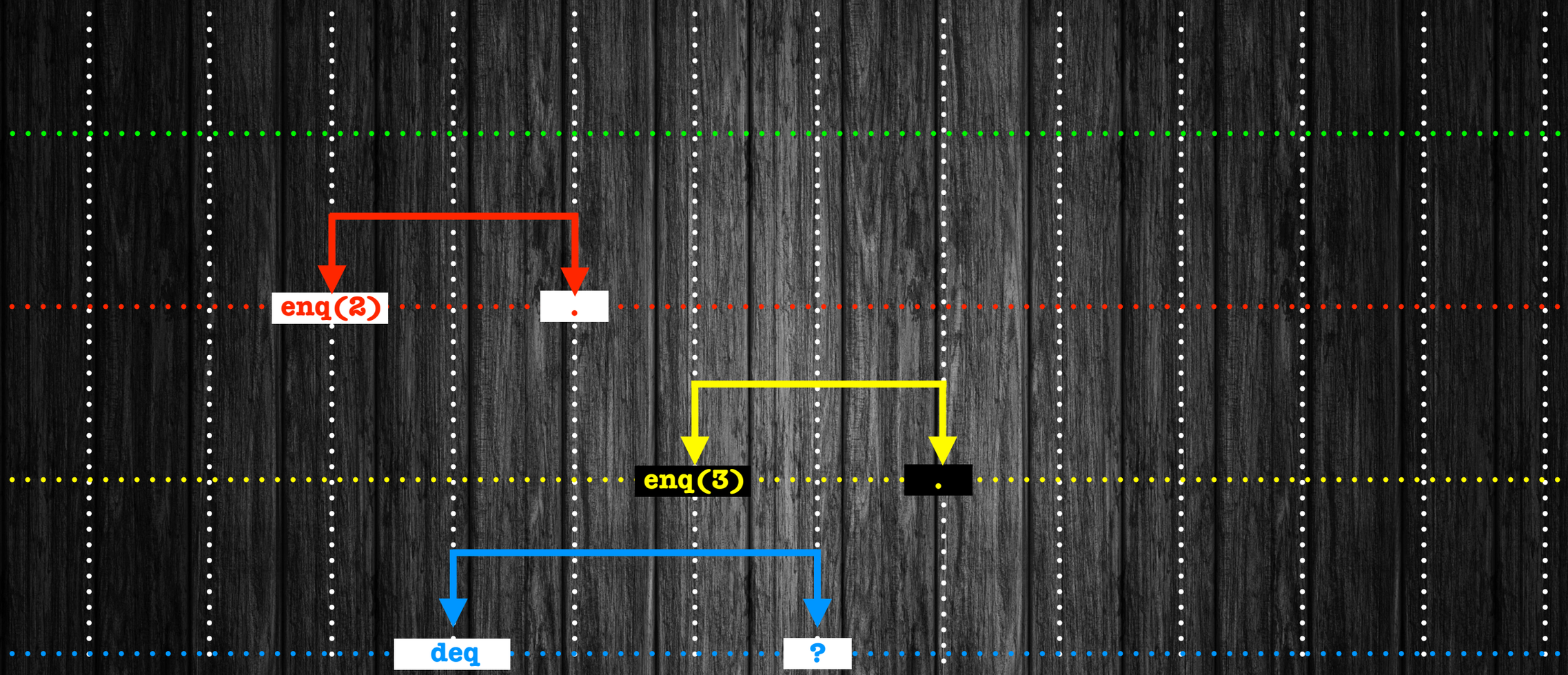
Concurrent Histories | Queues



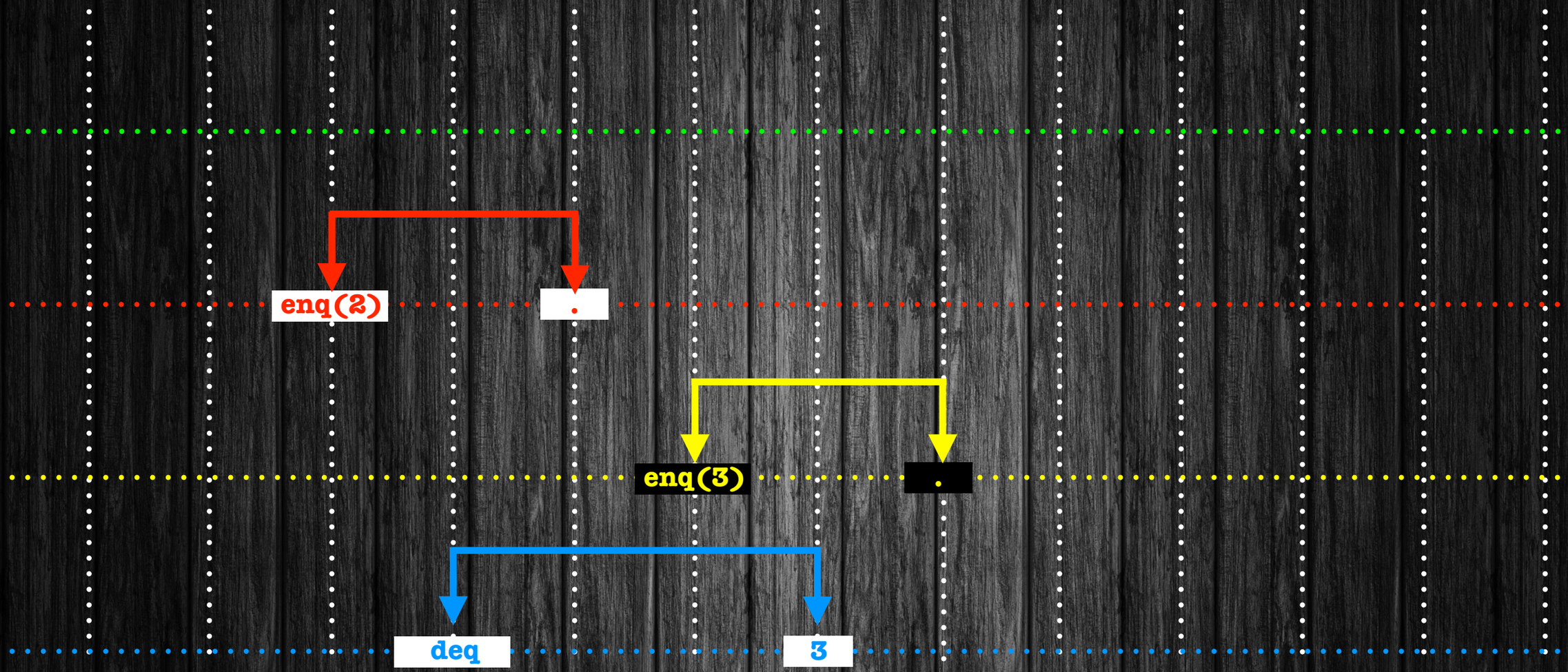
Concurrent Histories | Queues



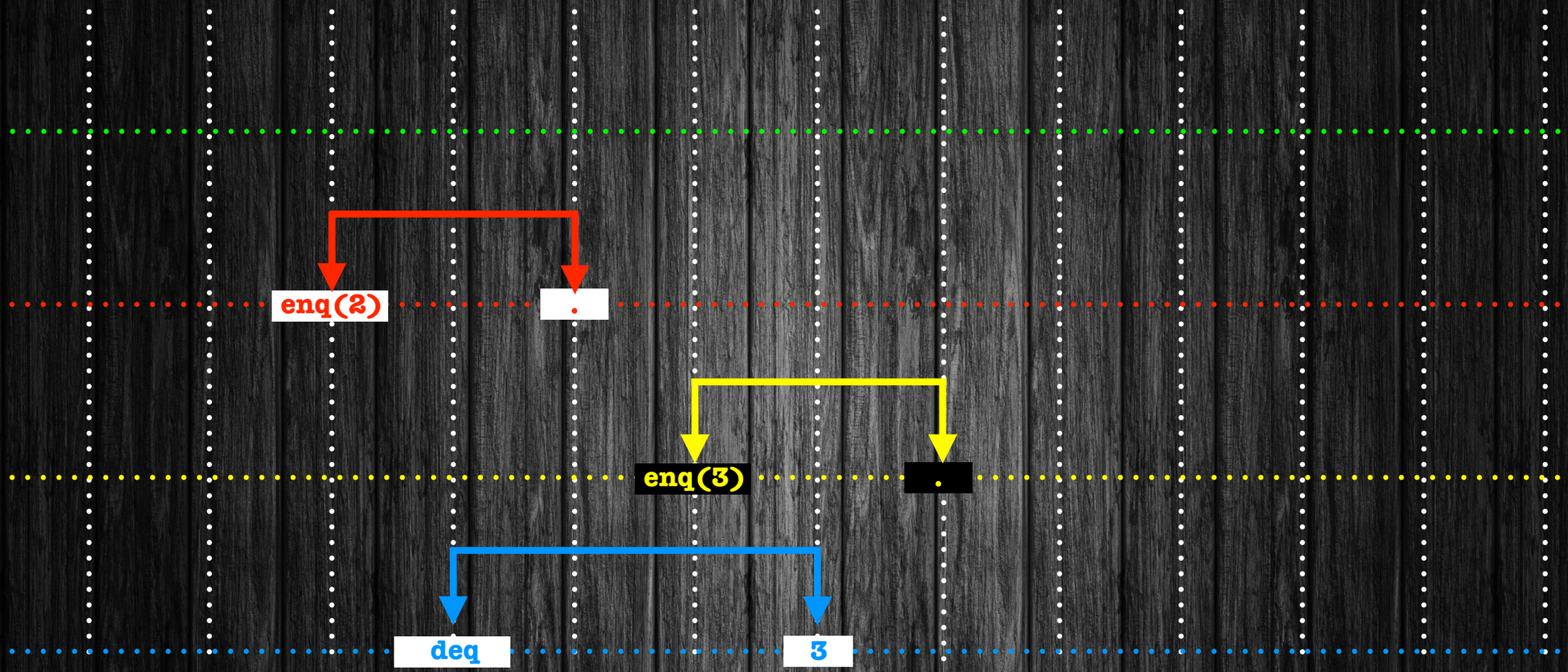
Concurrent Histories | Queues



Concurrent Histories | Queues

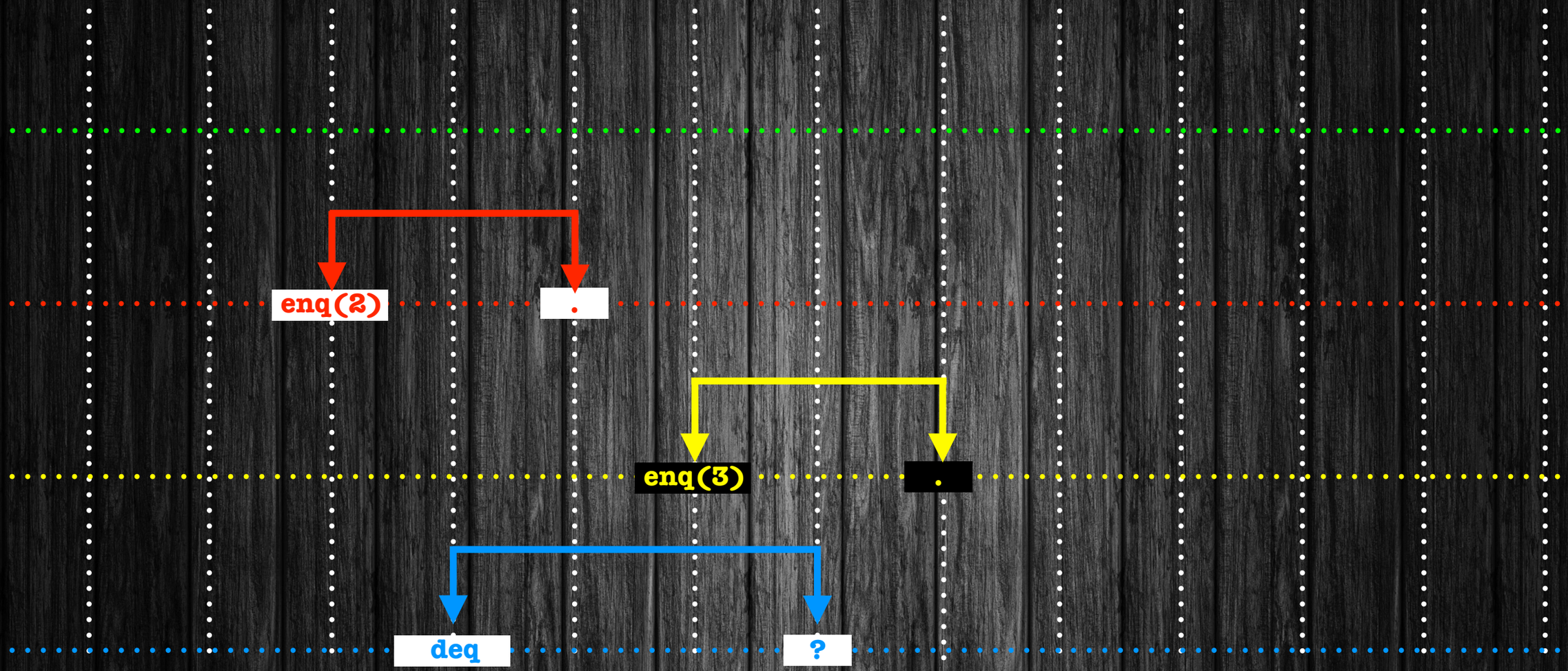


Concurrent Histories | Queues



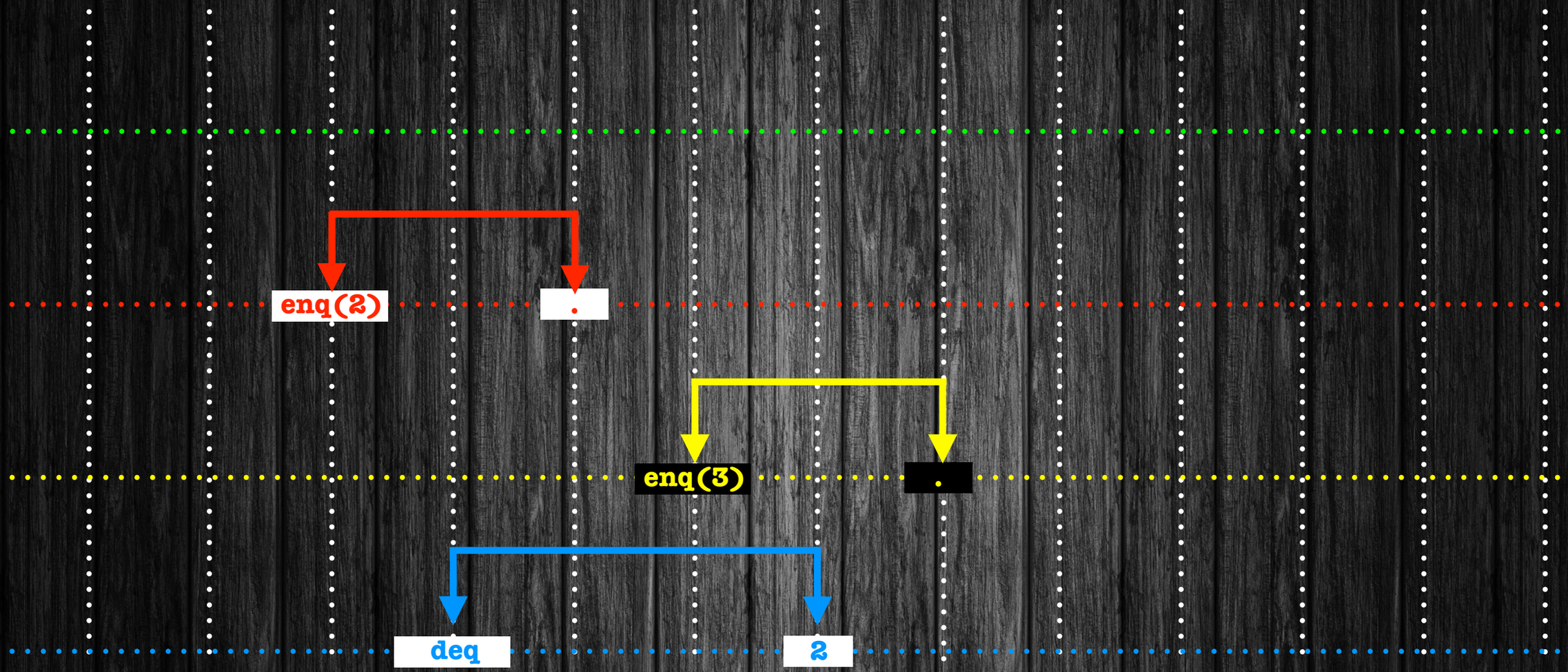


Concurrent Histories | Queues



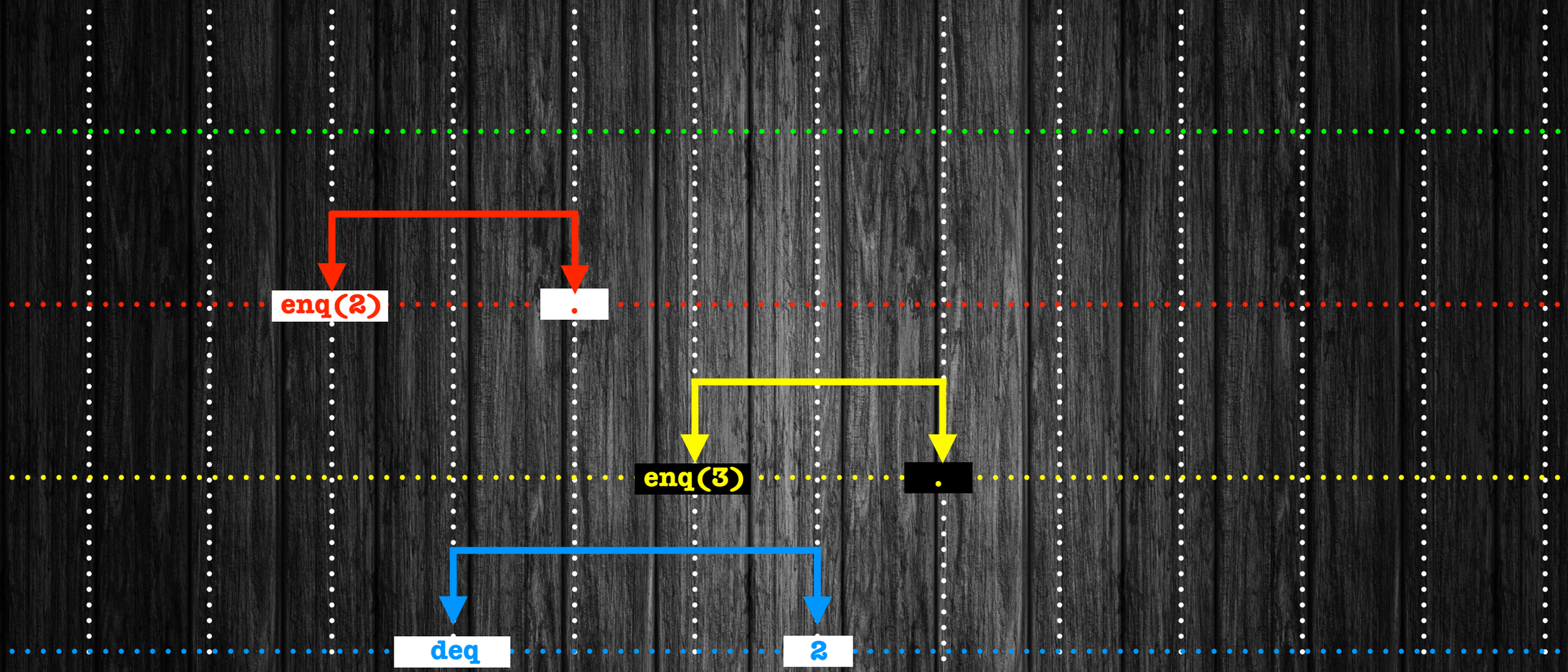
?

Concurrent Histories | Queues



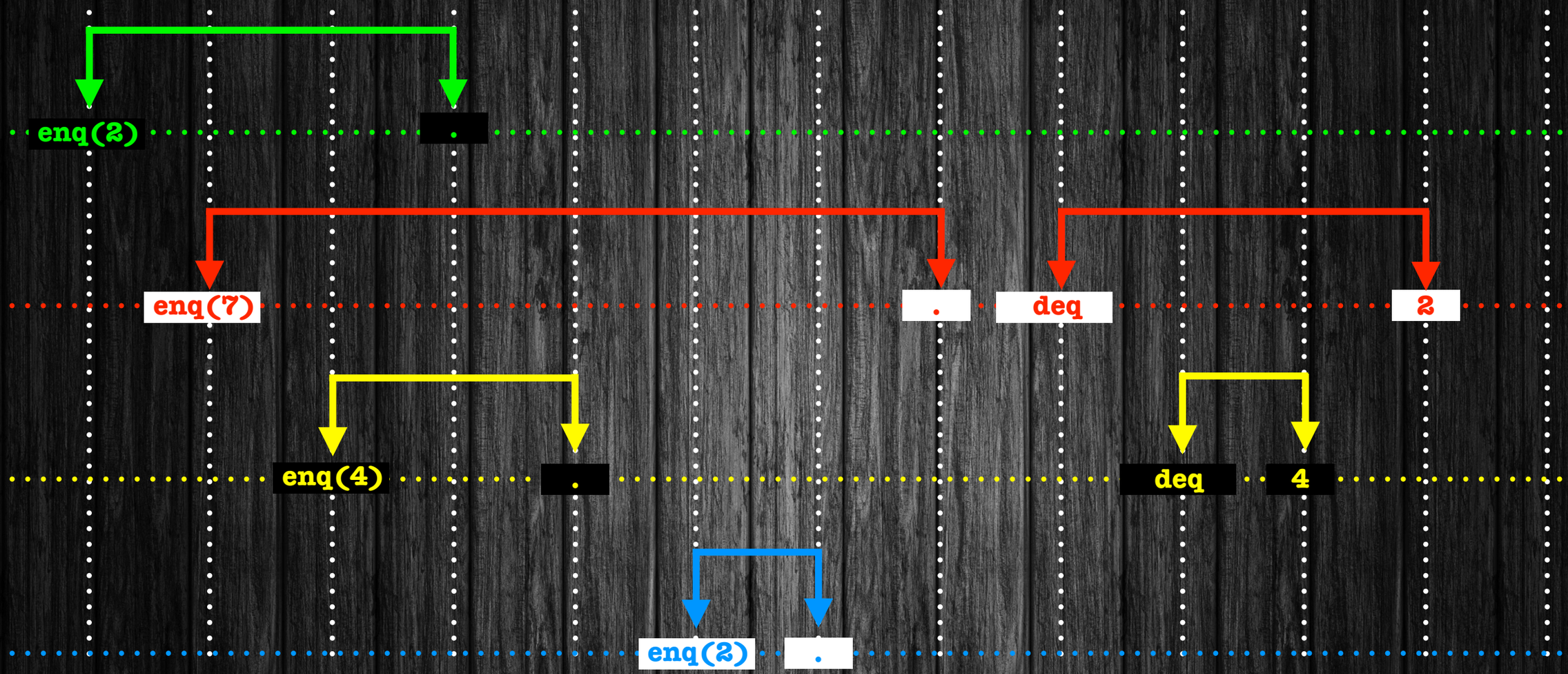
?

Concurrent Histories | Queues

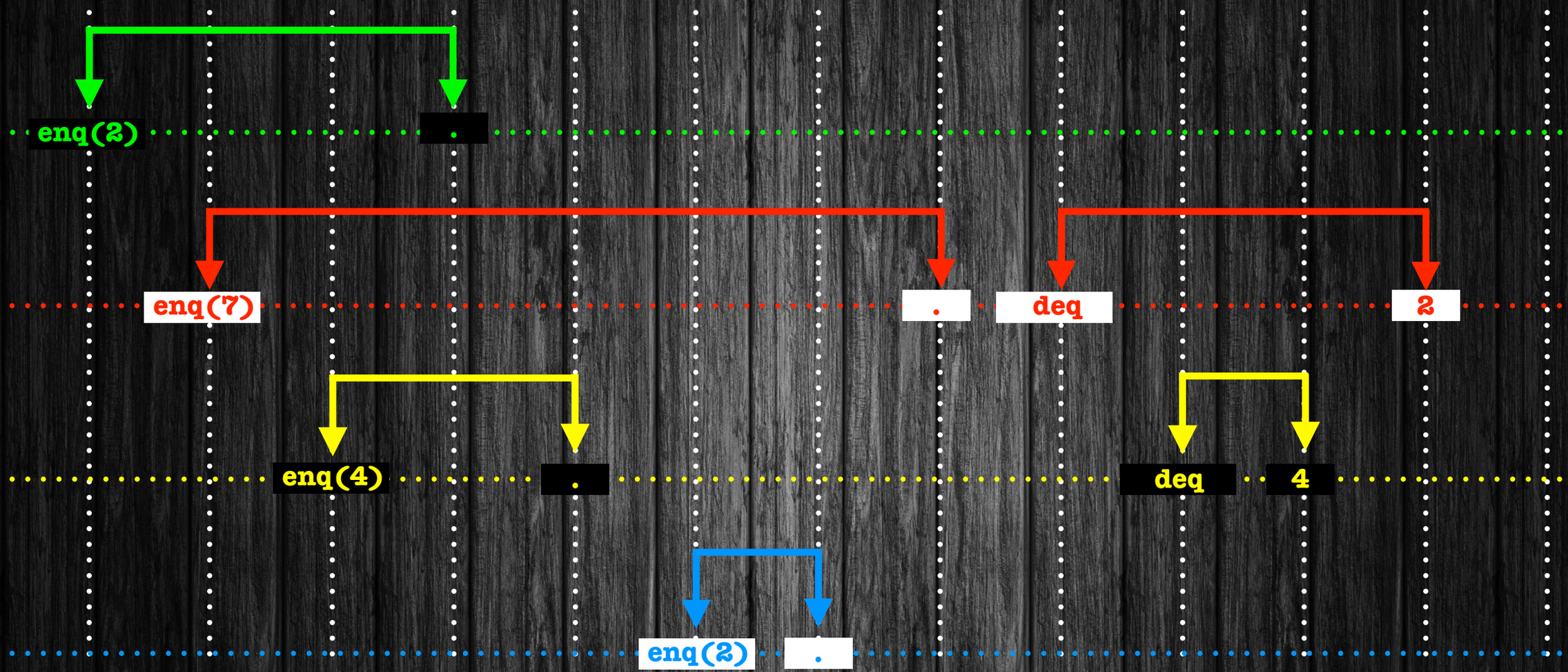


✔

Concurrent Histories | Queues



Concurrent Histories | Queues



✔