Concurrent Algorithms and Data Structures

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)

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)

- The first theory assignment will be available on Wednesday
- I will explain the problems in the assignment in Wednesday’s lecture.
- Deadline: Friday, next week.

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

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 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 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)

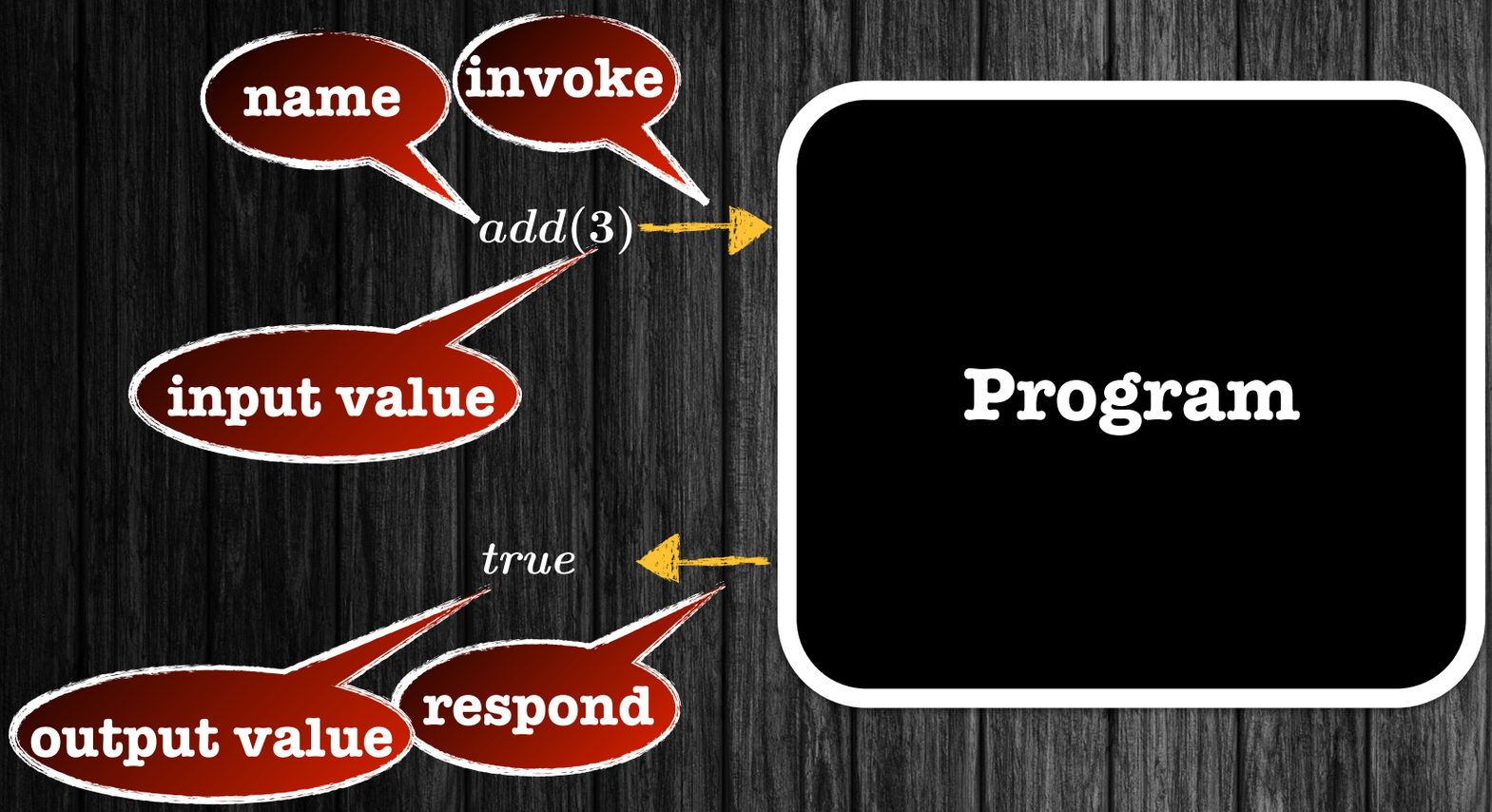
Outline

- 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

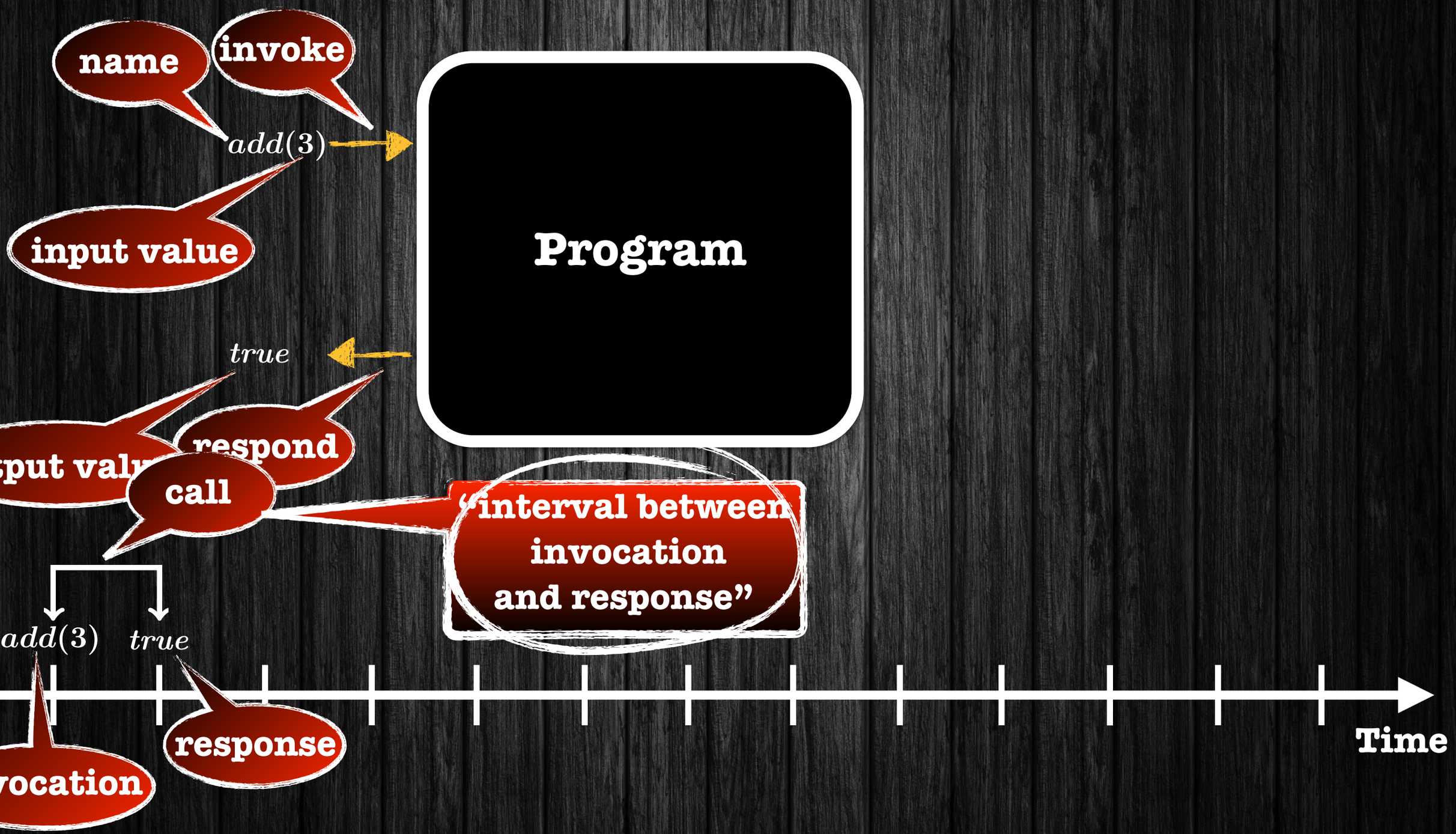
Sequential Programs | Sets



operation = two events

- invocation
- response

Sequential Histories



Sequential Histories

invoke

respond

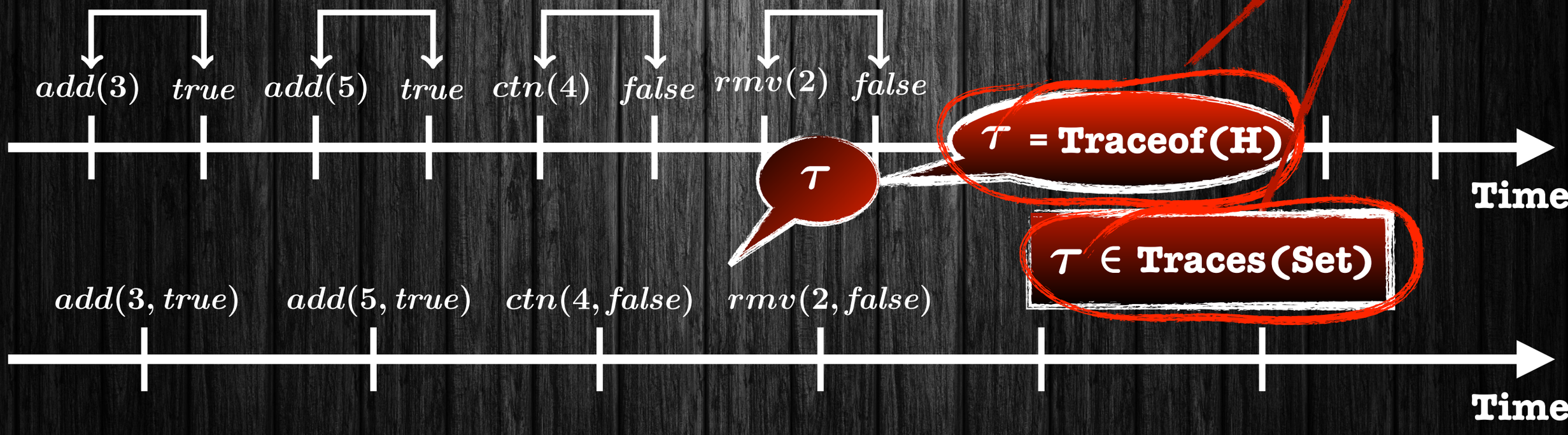
Program

A: abstract data type

H ∈ Histories(A)

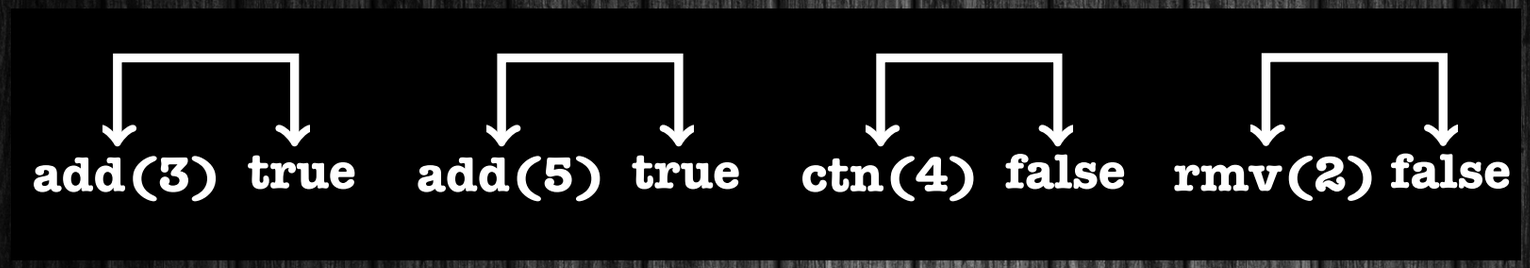
Traceof(H) ∈ Traces(A)

H ∈ Histories(Set)



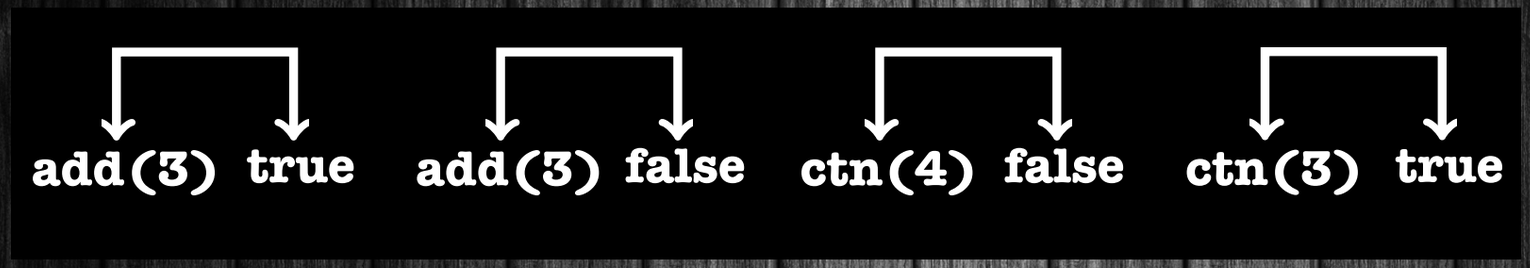
Sequential Histories

H_1



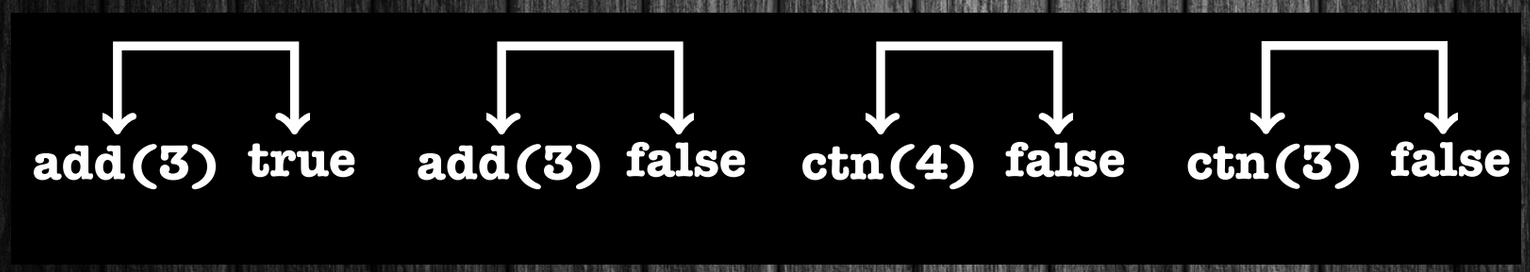
$H_1 \in$ Histories (Set)

H_2



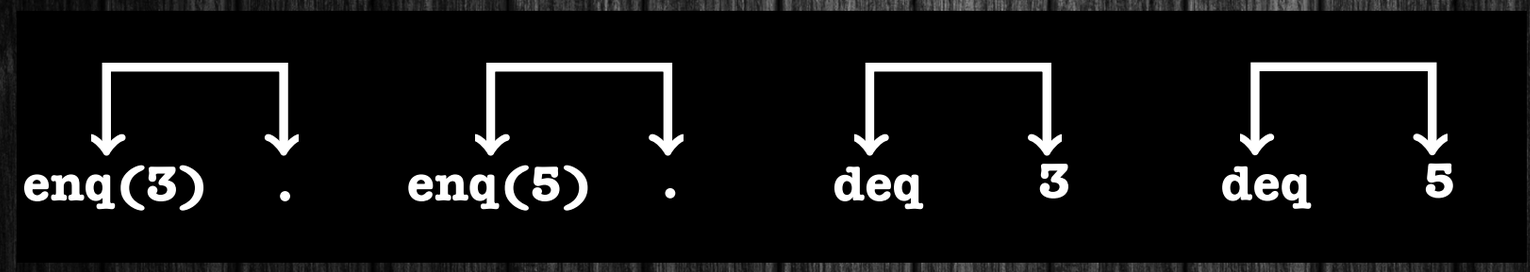
$H_2 \in$ Histories (Set)

H_3



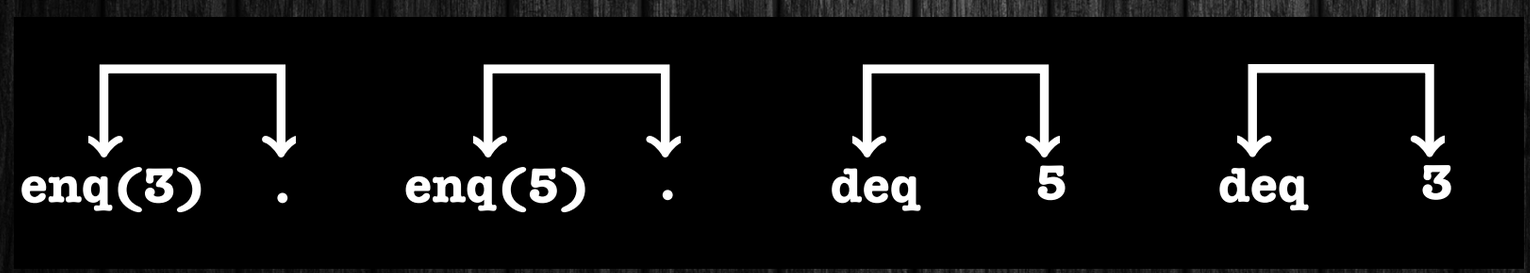
$H_3 \notin$ Histories (Set)

H_4



$H_4 \in$ Histories (Queue)

H_5



$H_5 \notin$ Histories (Queue)

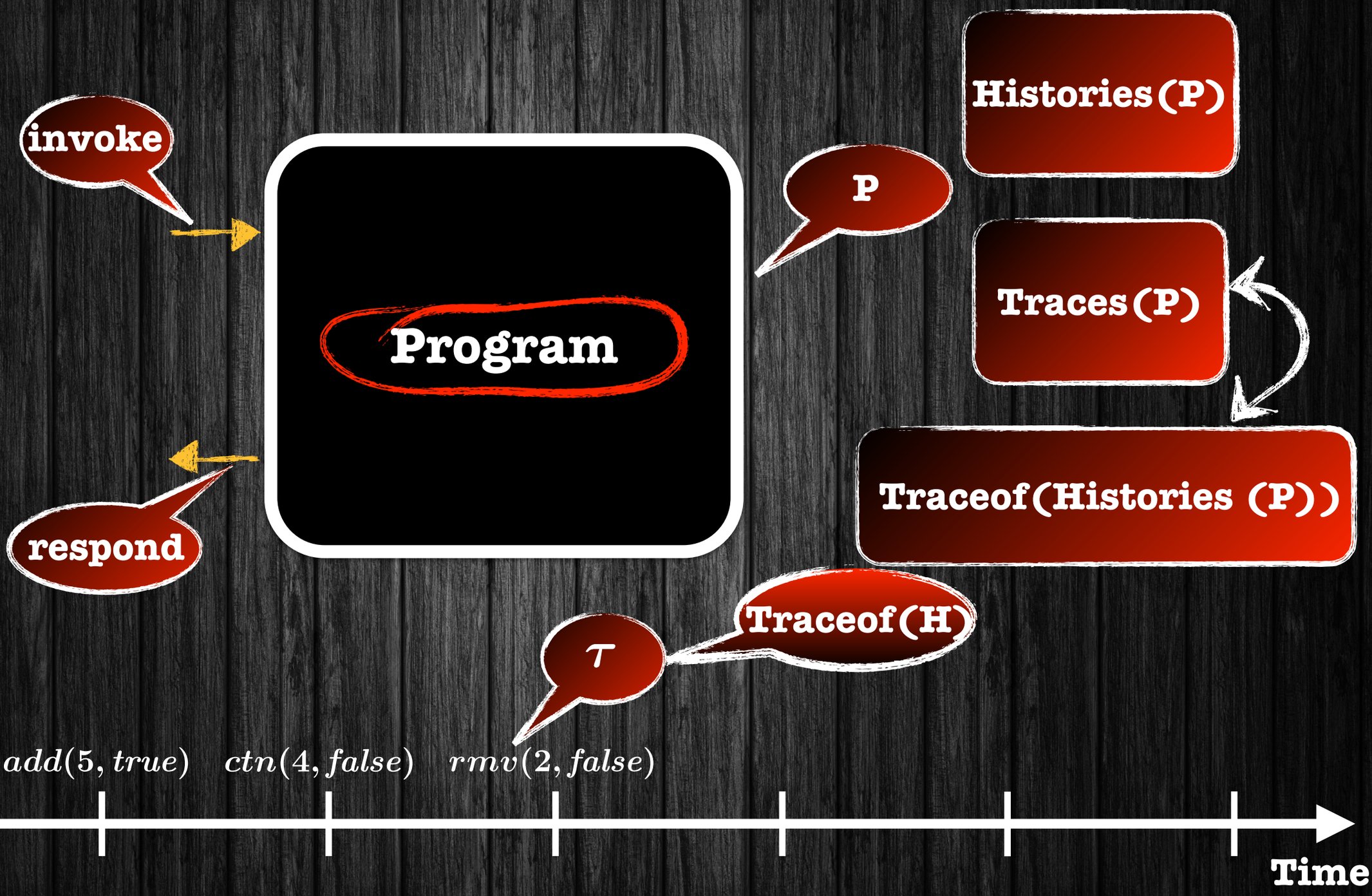
Sequential Programs

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

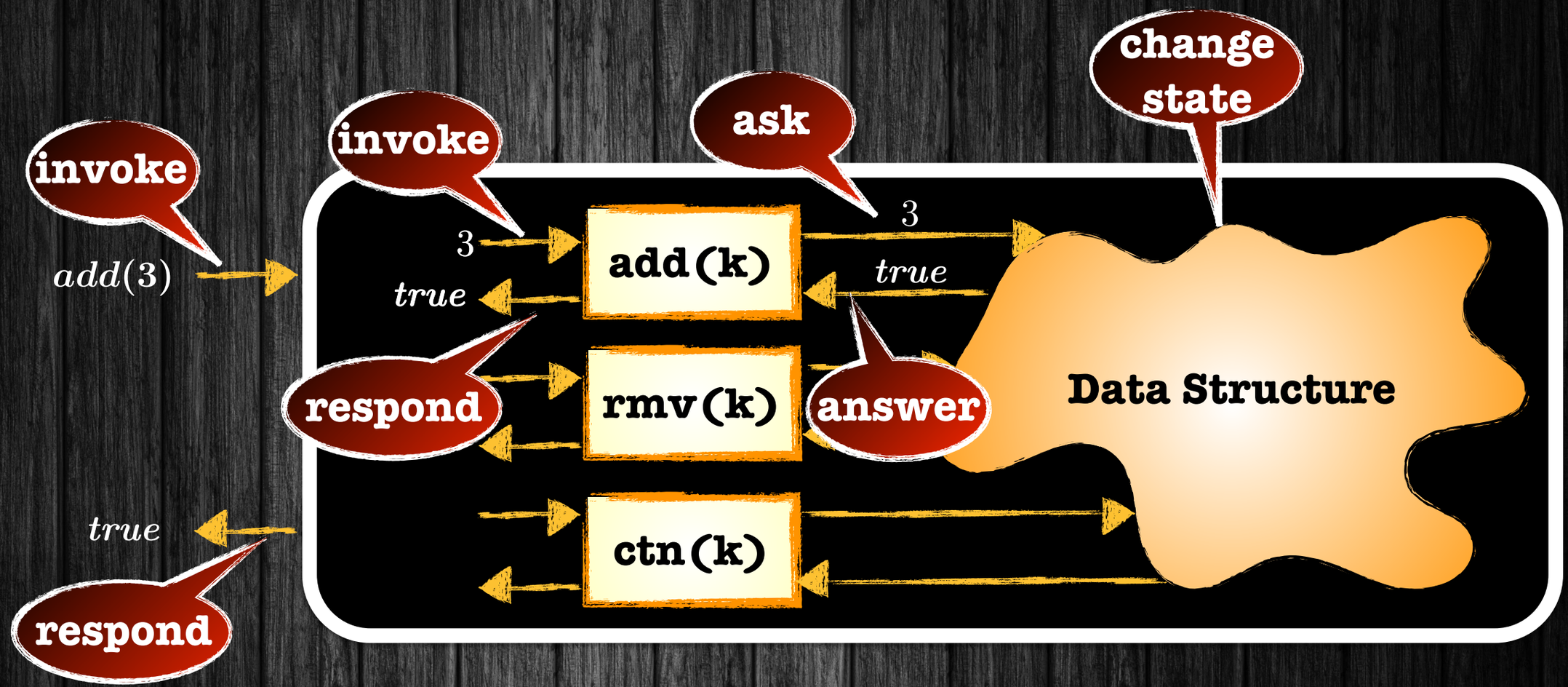
Sequential Programs

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

Sets | Sequential Programs



Sets | Sequential Programs



Sets | **Sequential Programs**

`add(k)`

Data Structure

Sets

Sequential Programs

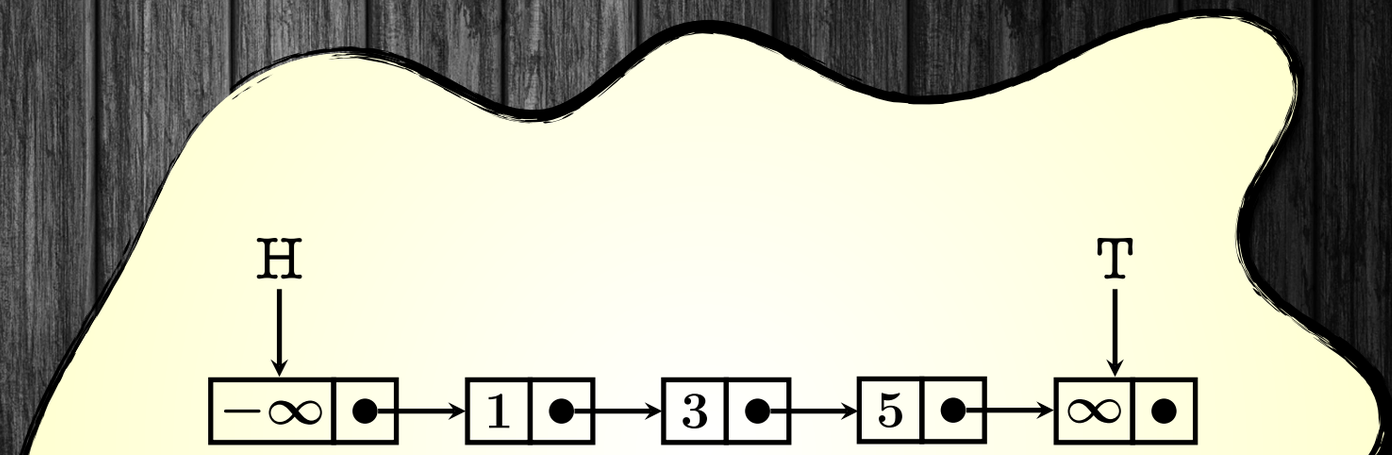
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```

Data Structure

Sets

Sequential Programs

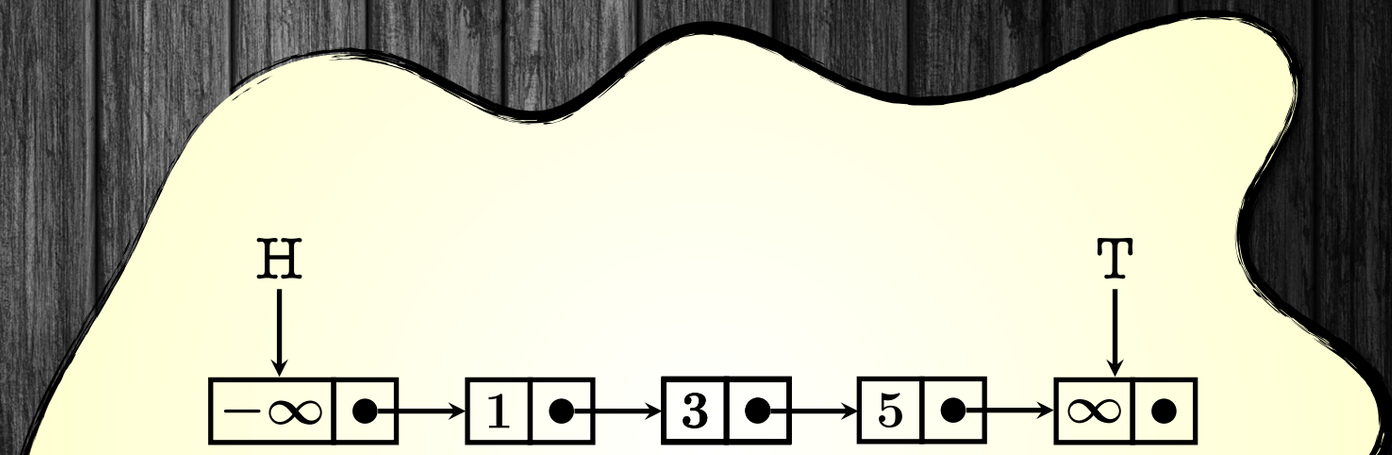
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```



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
```

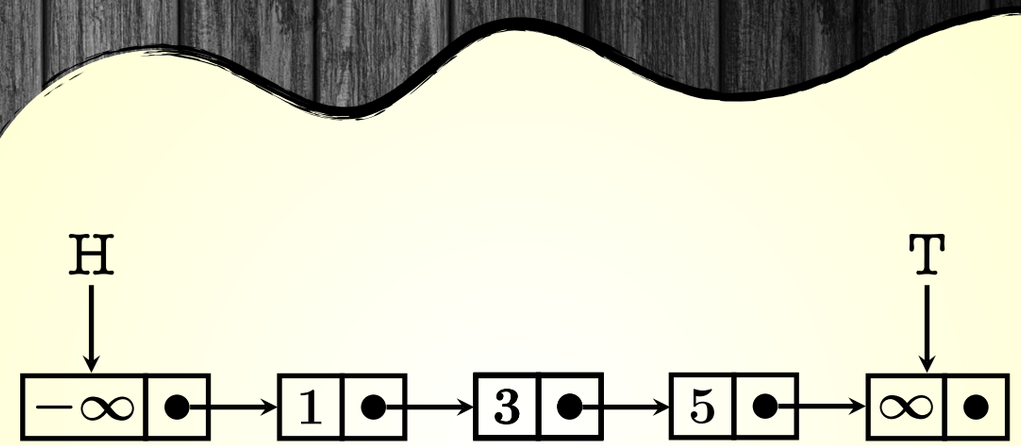


Sets

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
```

add(4)



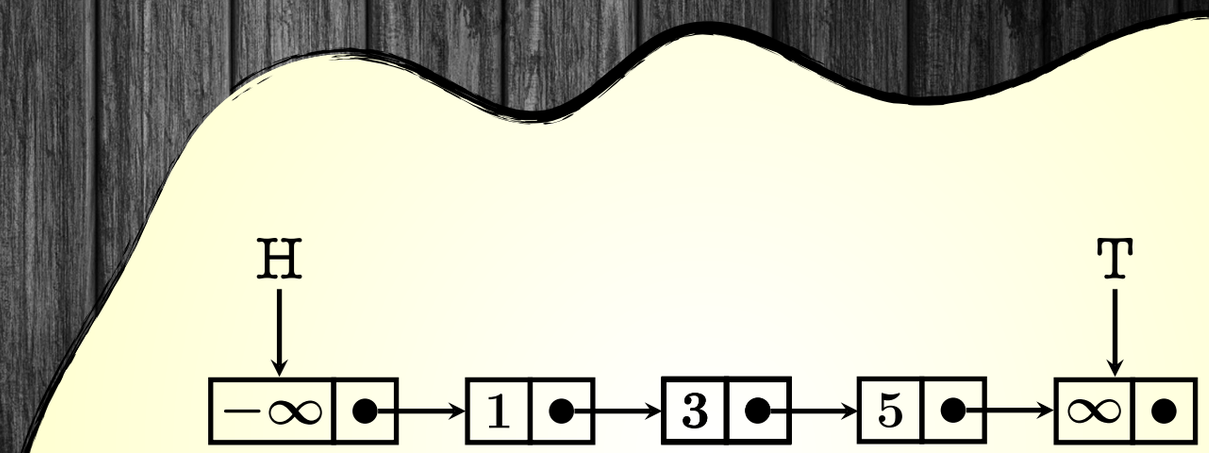
Sets

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)

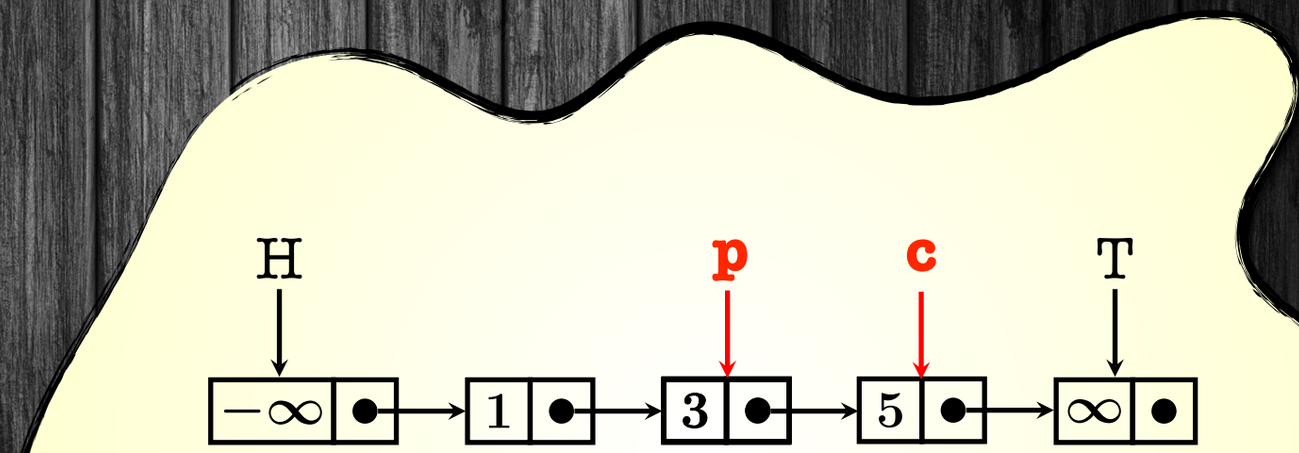


Sets | Sequential Programs

```
 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)

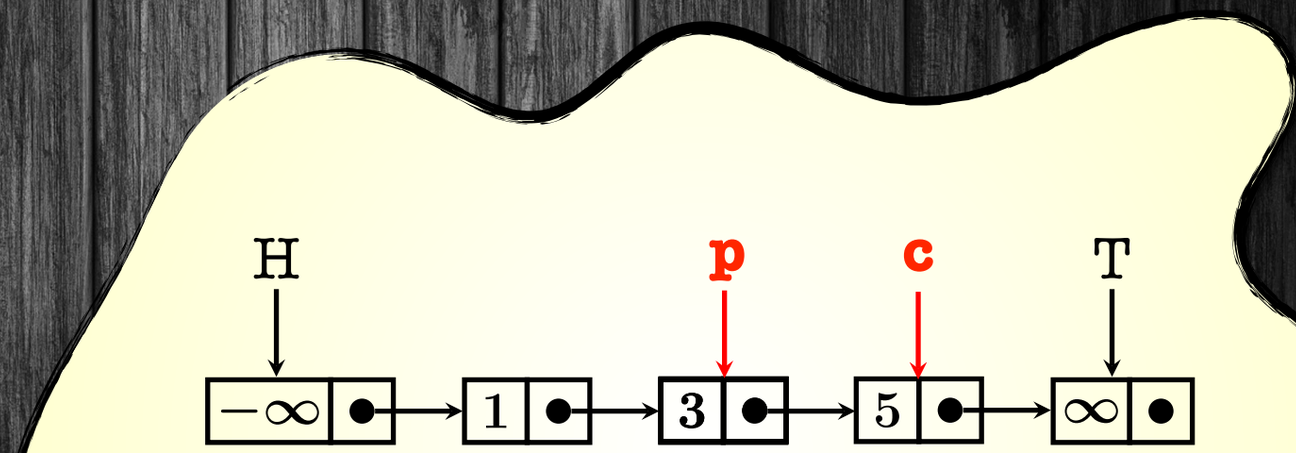


Sets | Sequential Programs

```
 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)

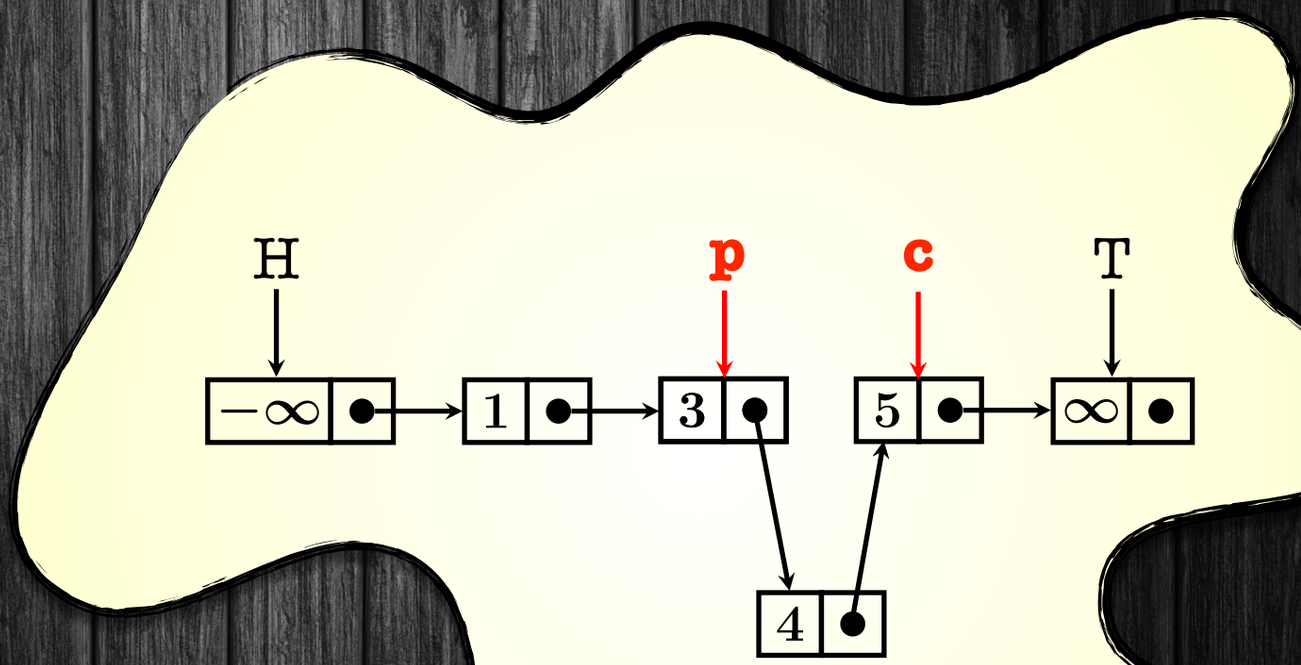


Sets | Sequential Programs

```
 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)

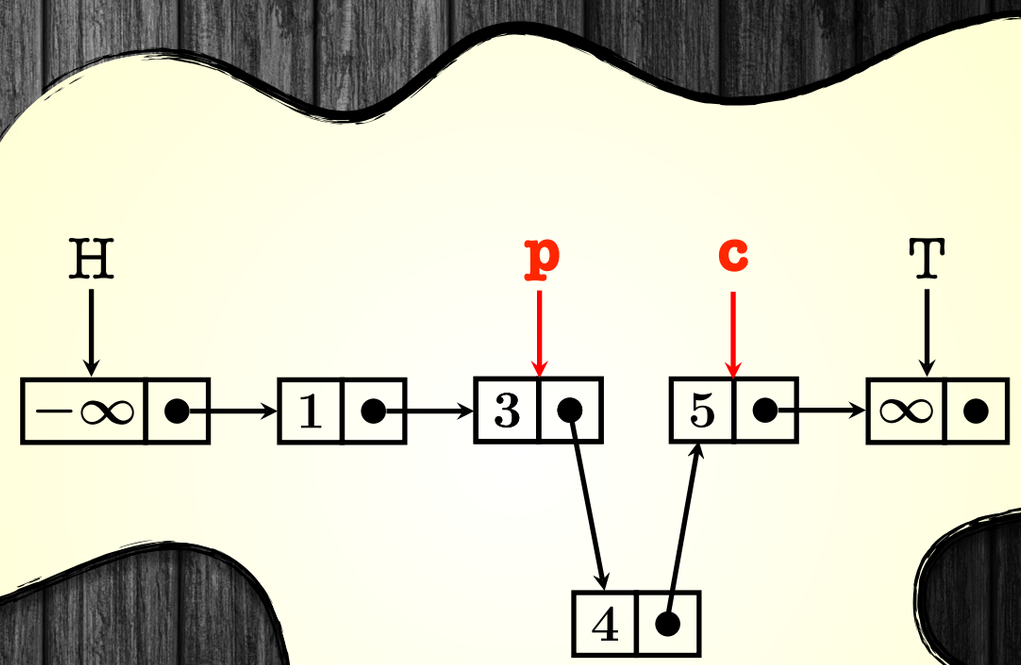


Sets | Sequential Programs

```
 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

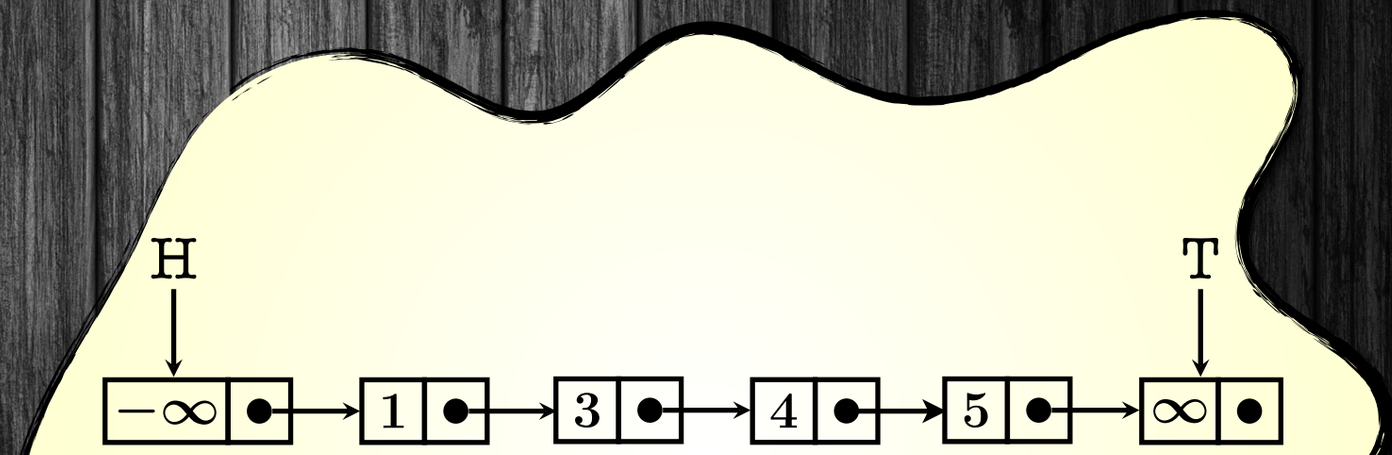
add(4)



Sets

Sequential Programs

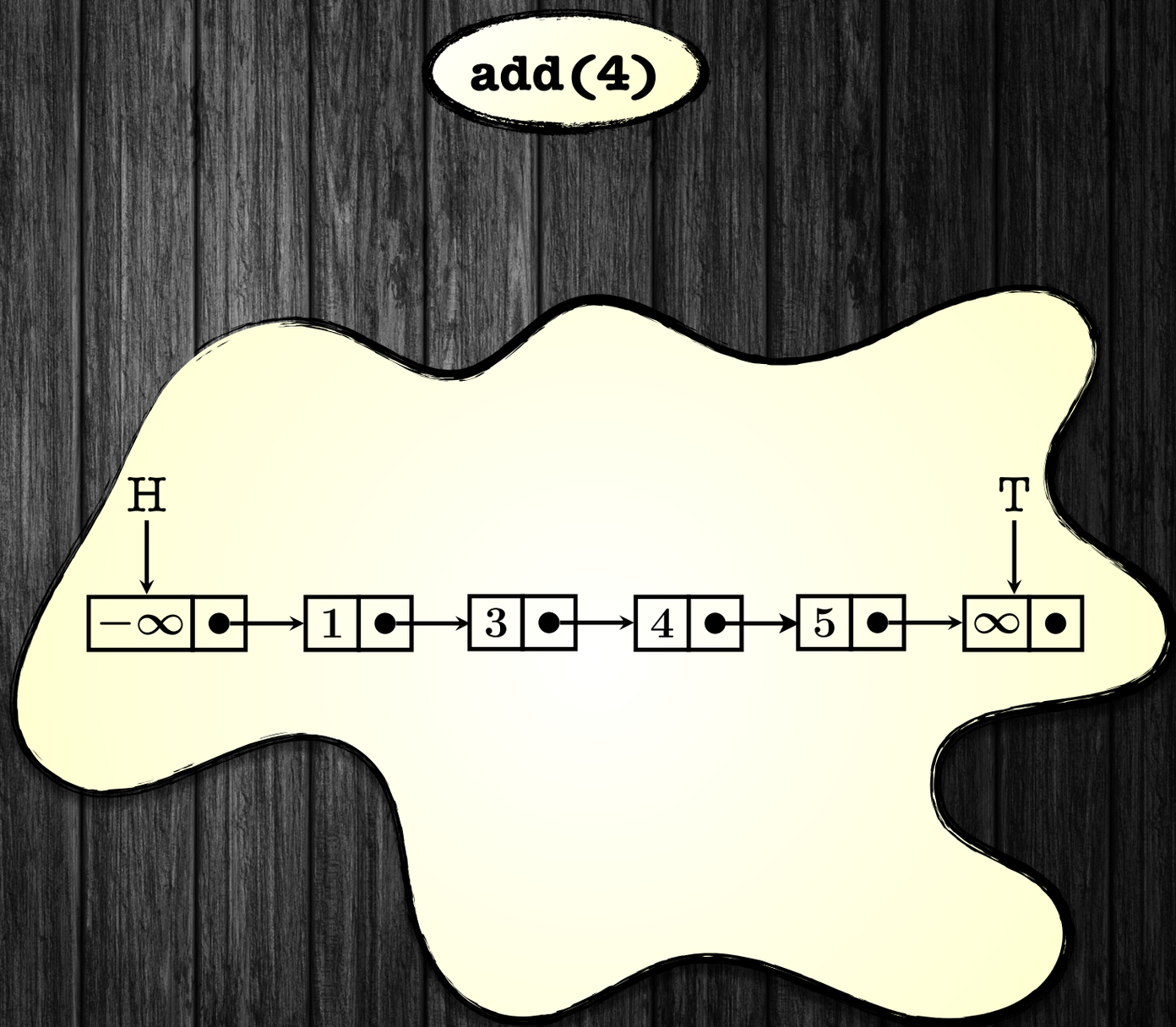
```
 add(k):
 Node p, c
 1  p = H;
 2  c = p.next
 3  while (c.key < k)
 4    p = c;
 5    c = c.next;
 6  if (c.key = k)
 7    return false
 8  else
 9    n = new Node(k,-)
10    n.next = c
11    p.next = n
12    return true
```



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
```

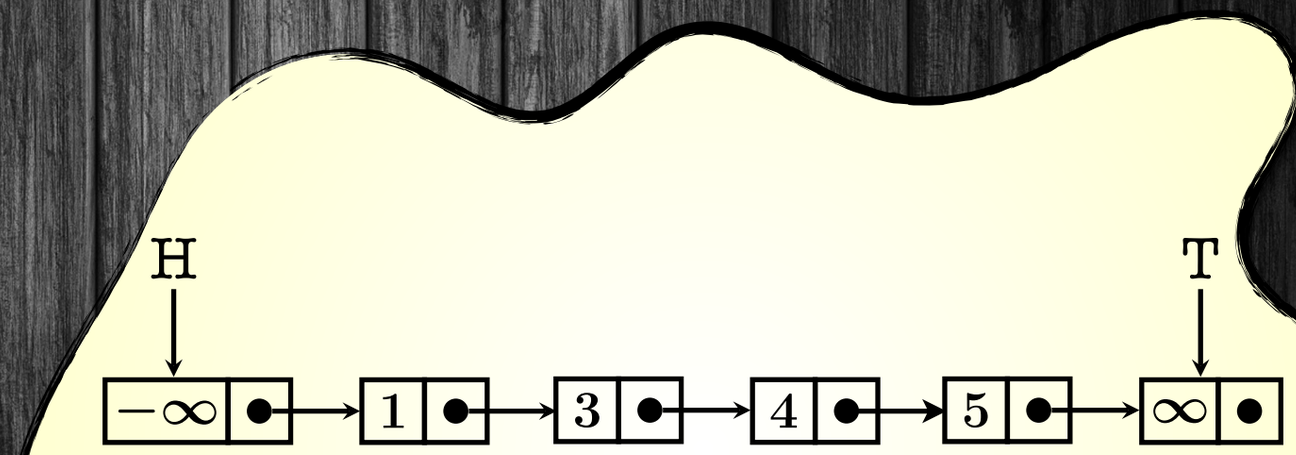


Sets | Sequential Programs

```
 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)

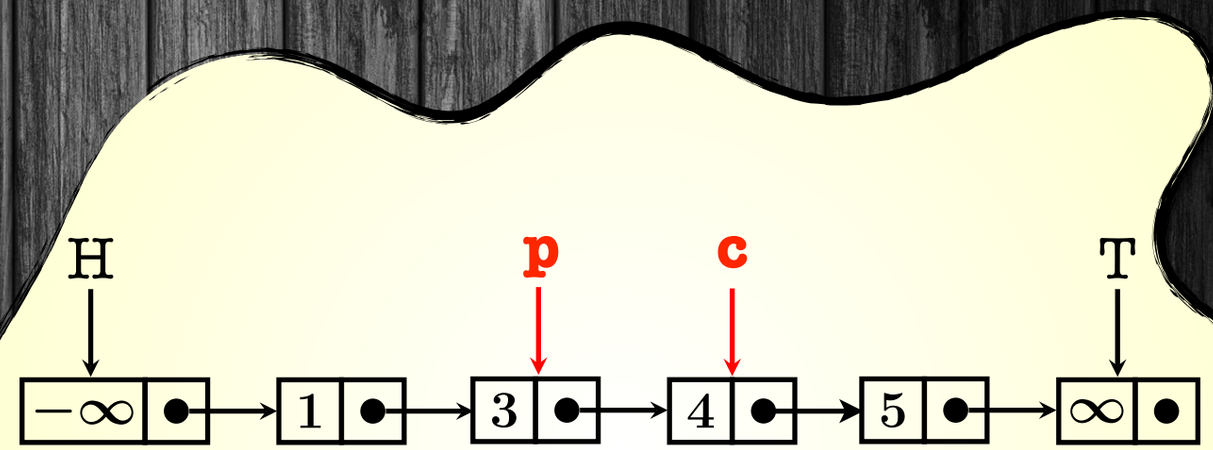


Sets | Sequential Programs

```
 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)

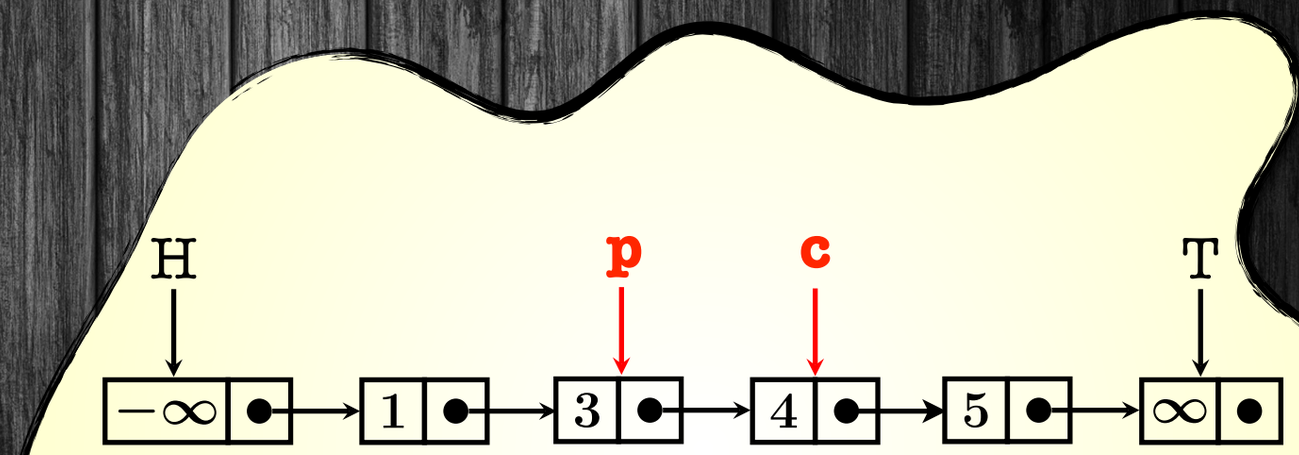


Sets | Sequential Programs

```
 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)

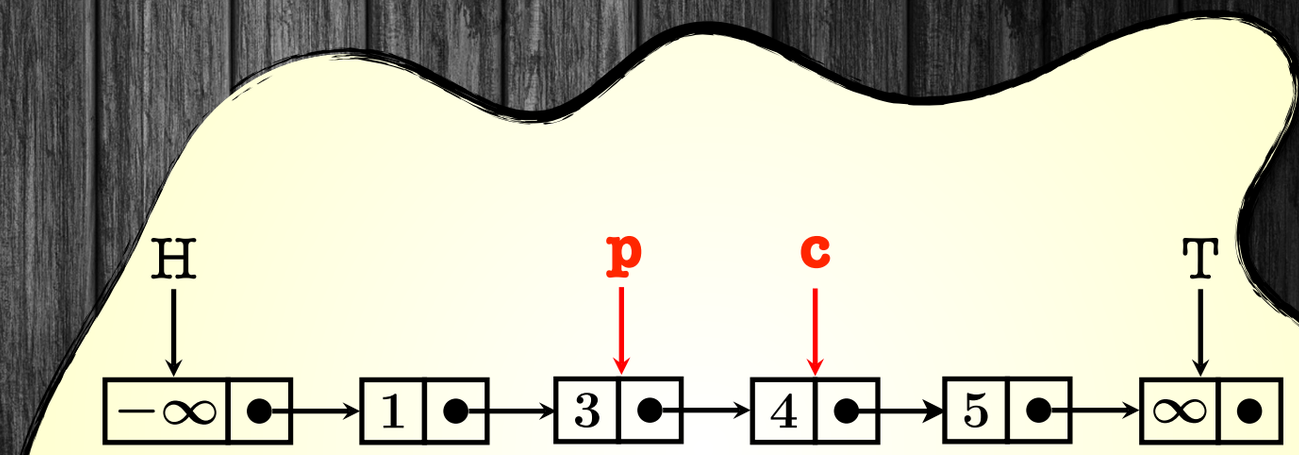


Sets | Sequential Programs

```
 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

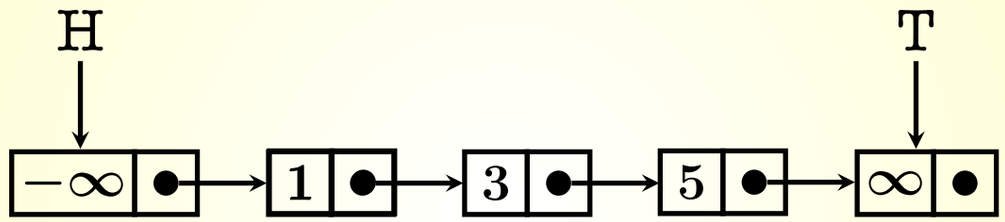
add(4)



Sets

Sequential Programs

```
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
```

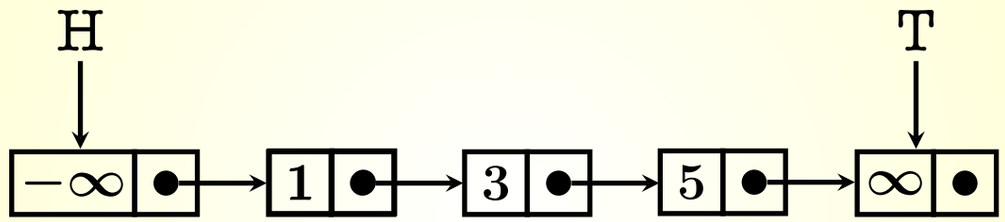


Sets

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)

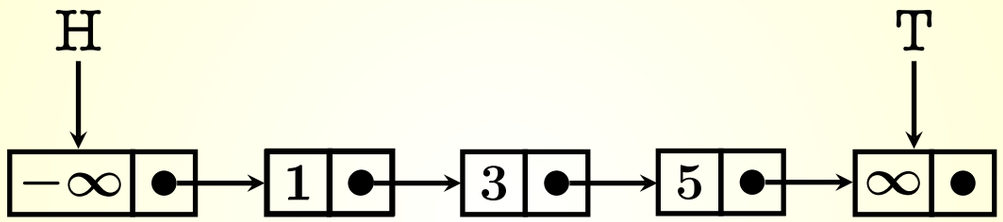


Sets | Sequential Programs

```
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)



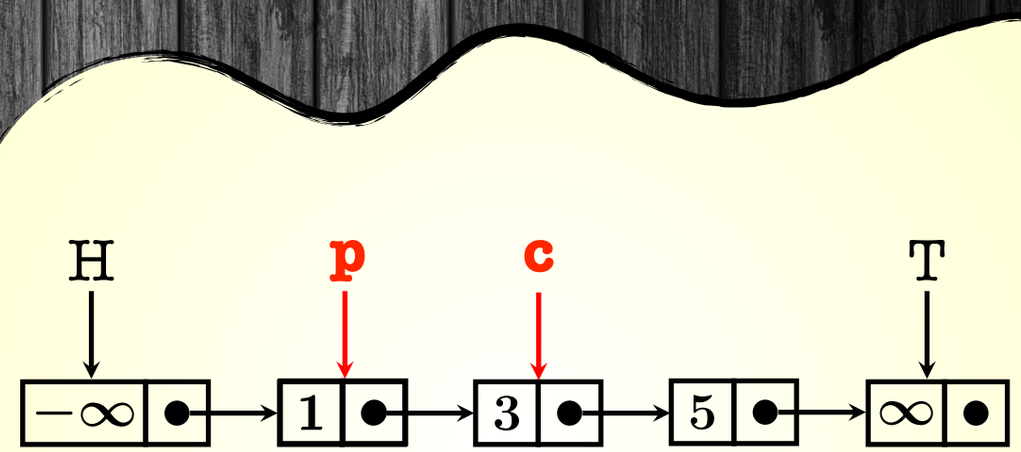
Sets

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)

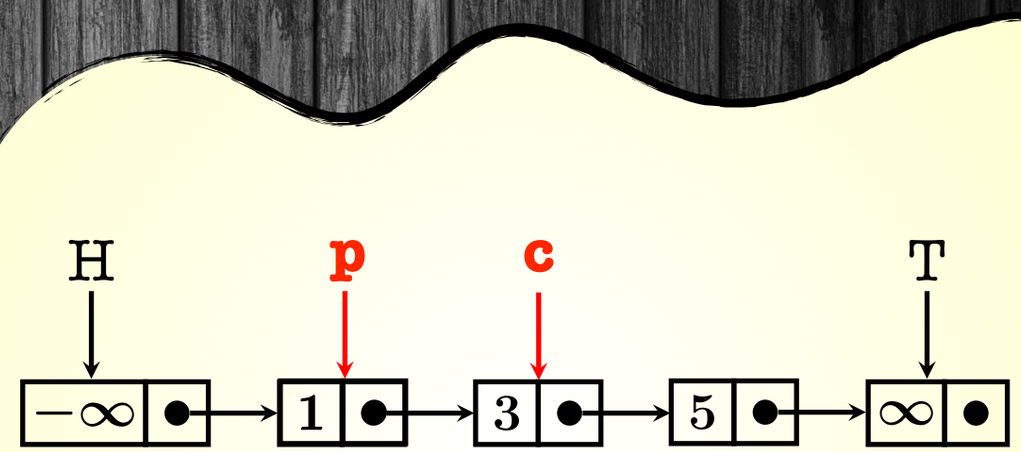


Sets | Sequential Programs

```
  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)

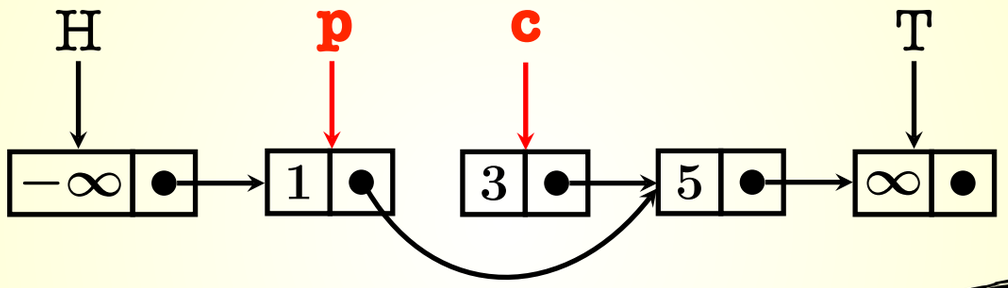


Sets | Sequential Programs

```
  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)



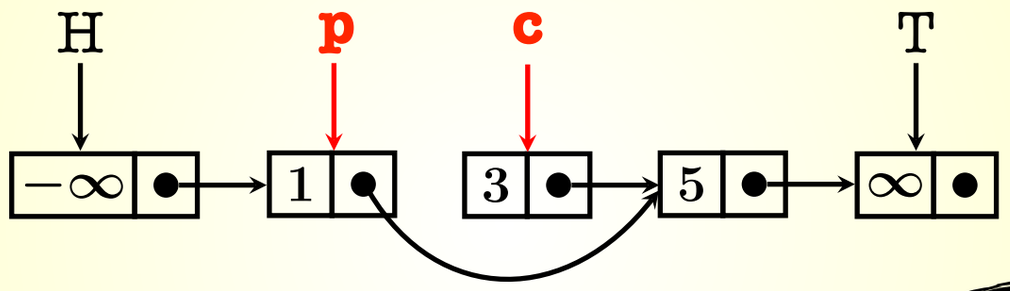
Sets

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

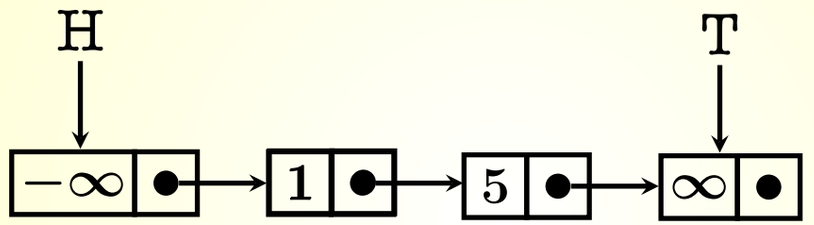
rmv(3)



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
```

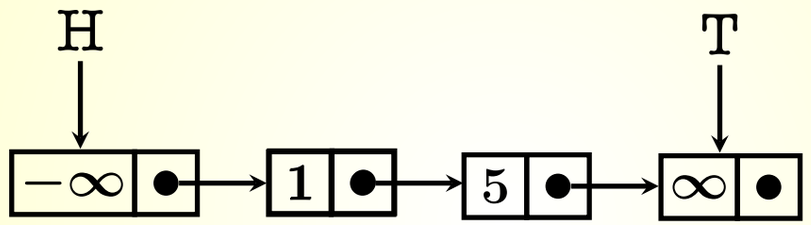


Sets

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)



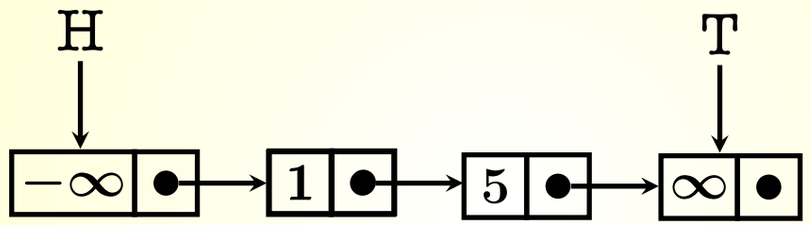
Sets

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)

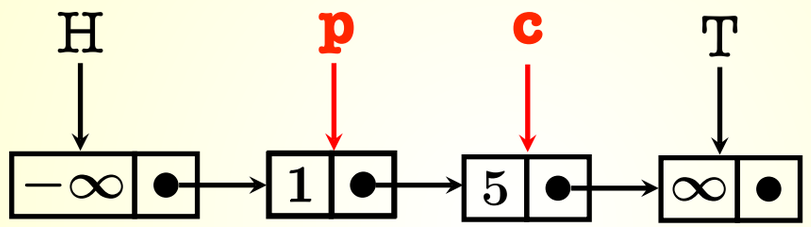


Sets | Sequential Programs

```
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)

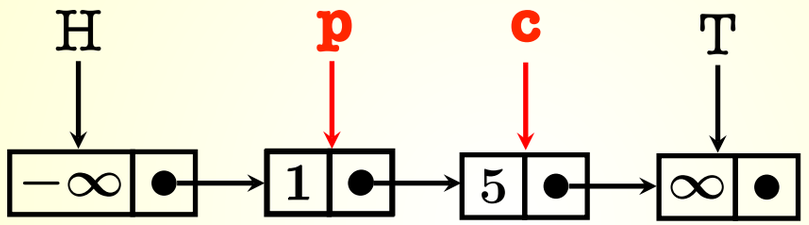


Sets | Sequential Programs

```
  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)

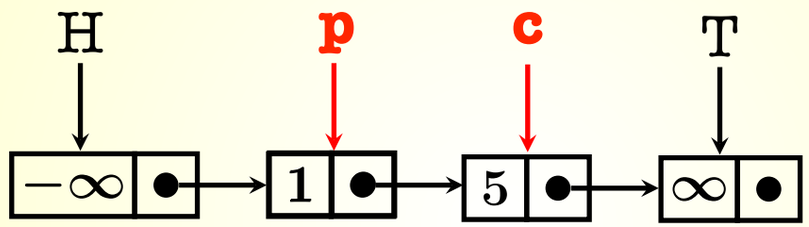


Sets | Sequential Programs

```
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

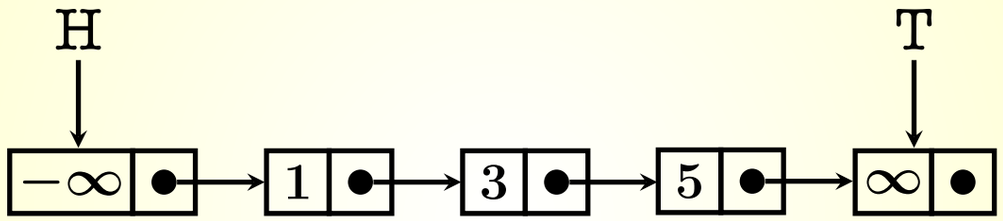
rmv(3)



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
```

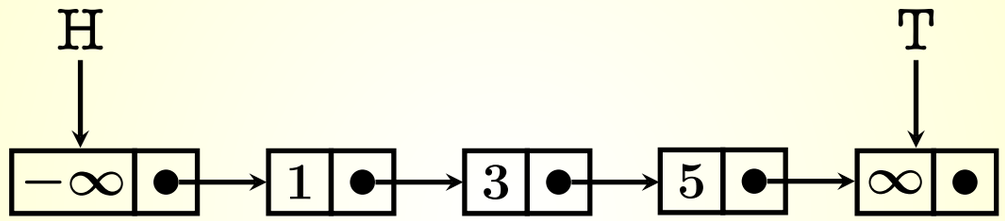


Sets

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)



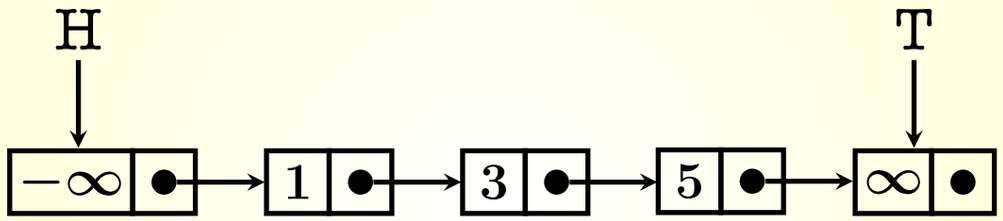
Sets

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
```

locate

ctn(3)

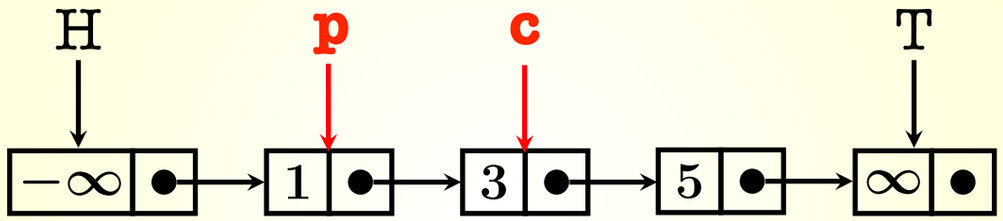


Sets | Sequential Programs

```
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)

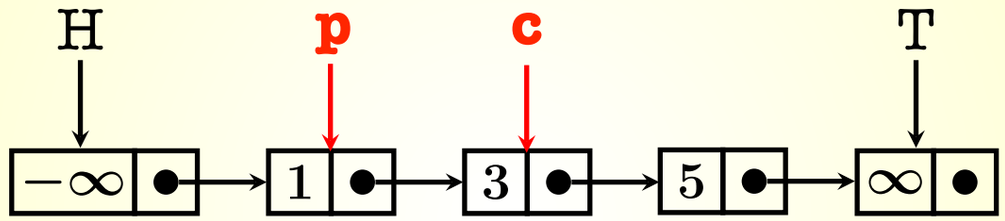


Sets | Sequential Programs

```
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)

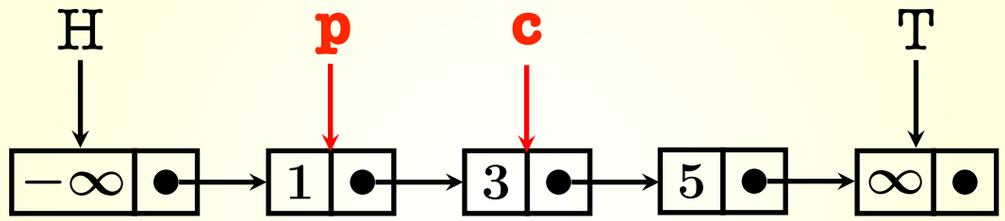


Sets | Sequential Programs

```
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

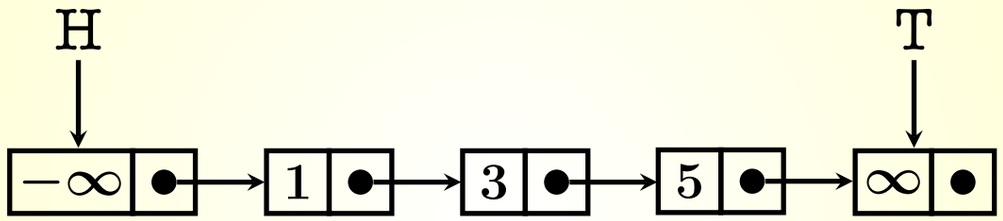
ctn(3)



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
```

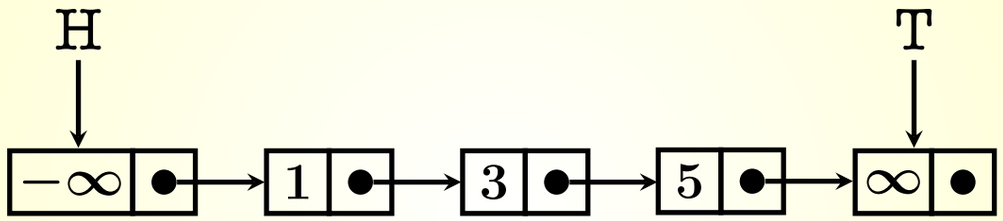


Sets

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)

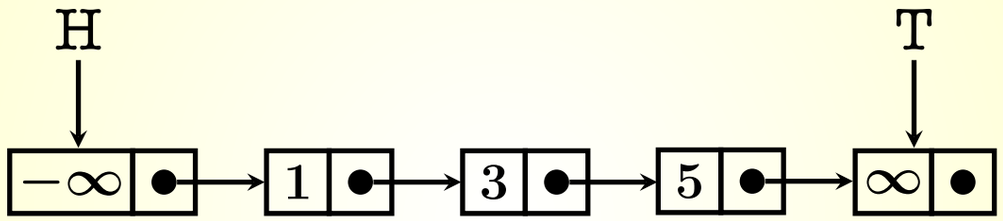


Sets | Sequential Programs

```
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)

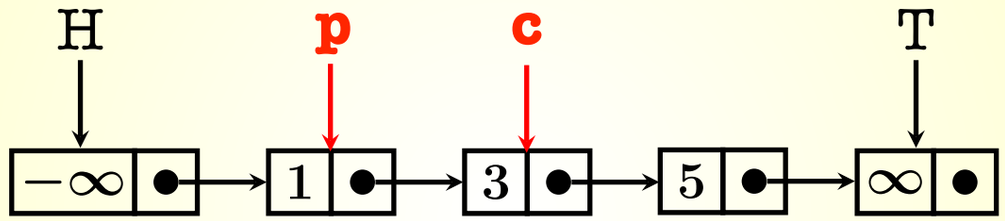


Sets | Sequential Programs

```
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)

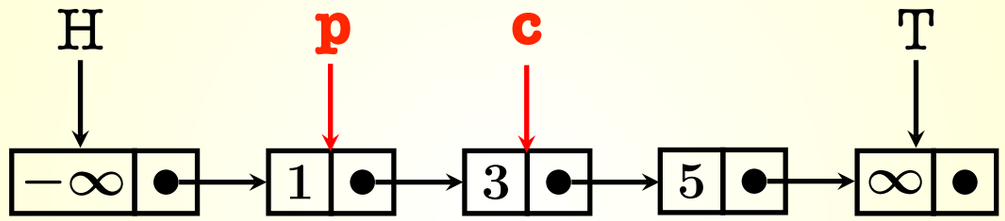


Sets | Sequential Programs

```
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)

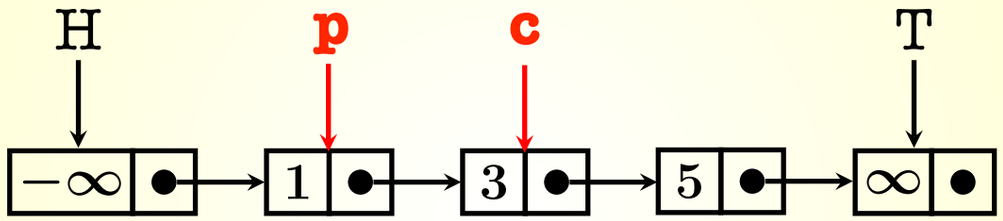


Sets | Sequential Programs

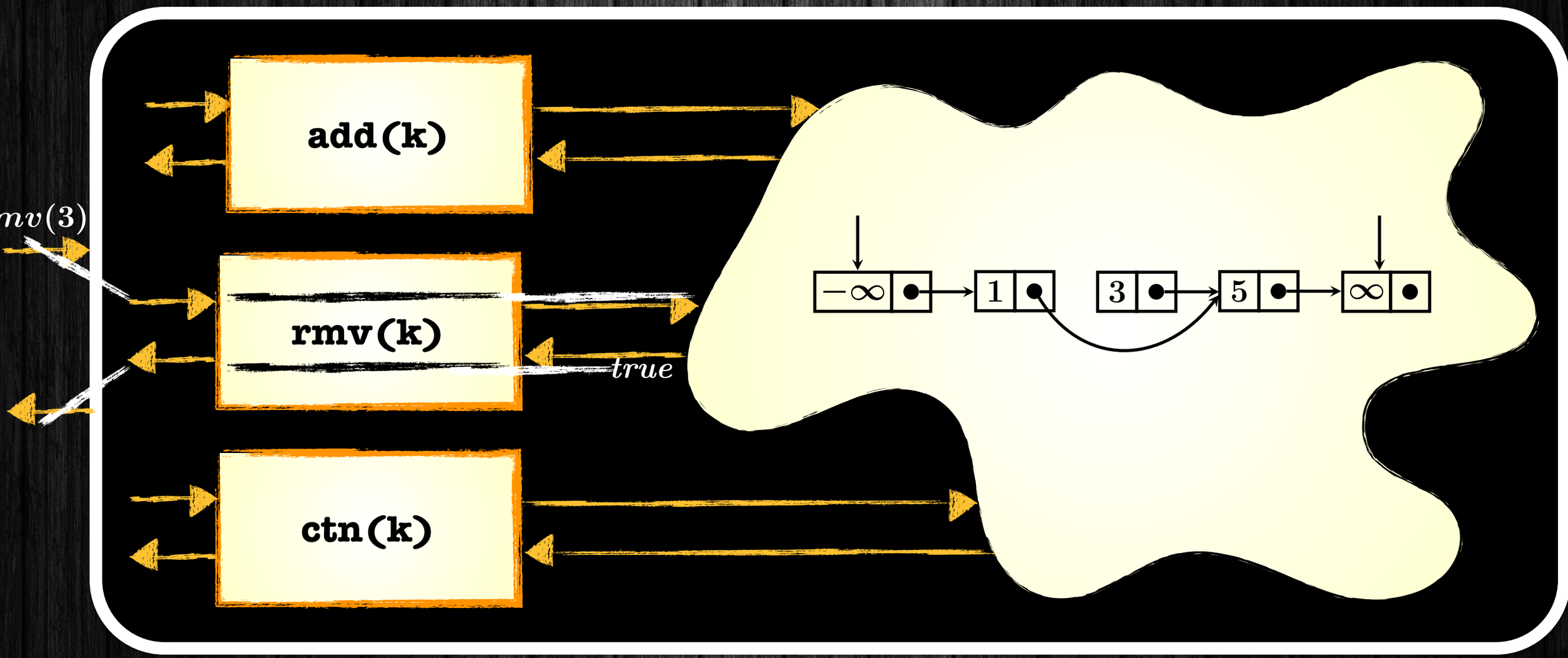
```
ctn(k):
Node p, c
1  p = H;
2  c = p.next
3  while (c.key < k)
4    p = c;
5    c = c.next;
6  if (c.key = k)
7    return true
8  else
9    return false
```

locate

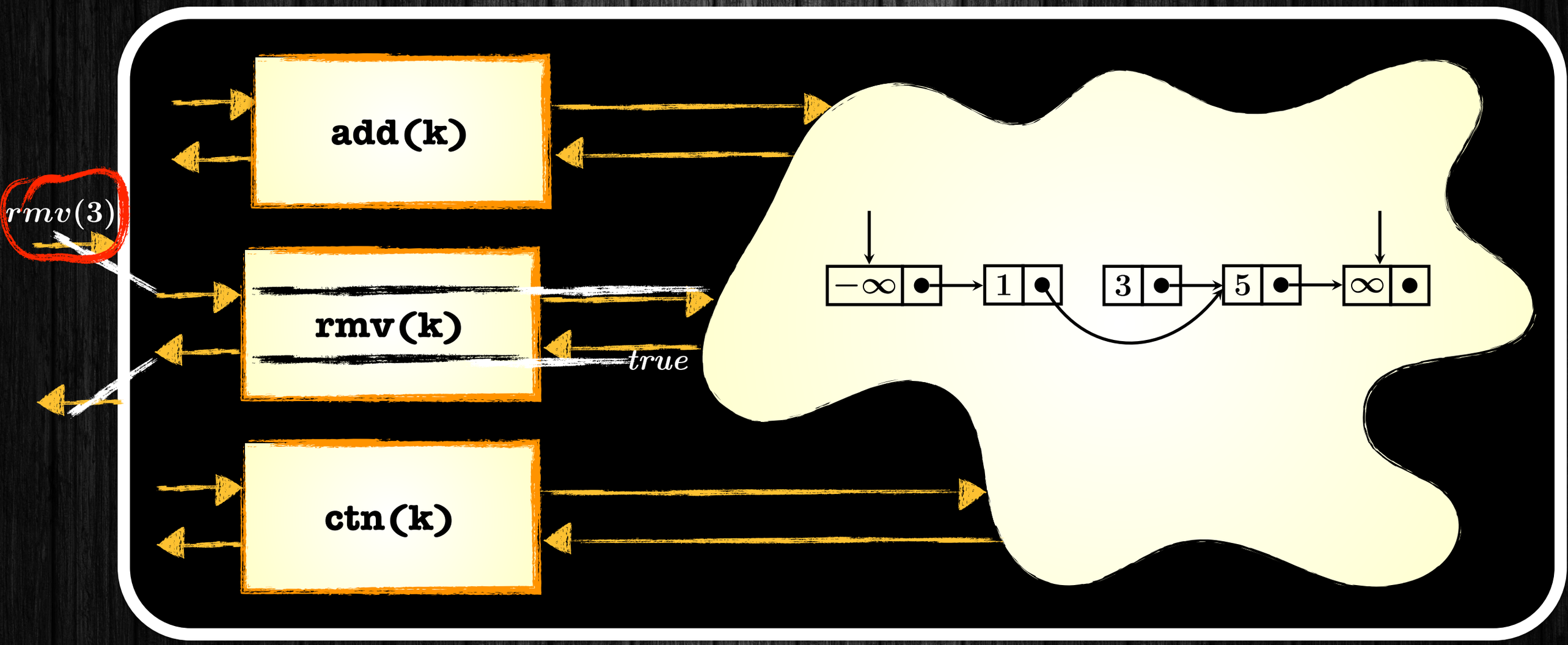
ctn(2)



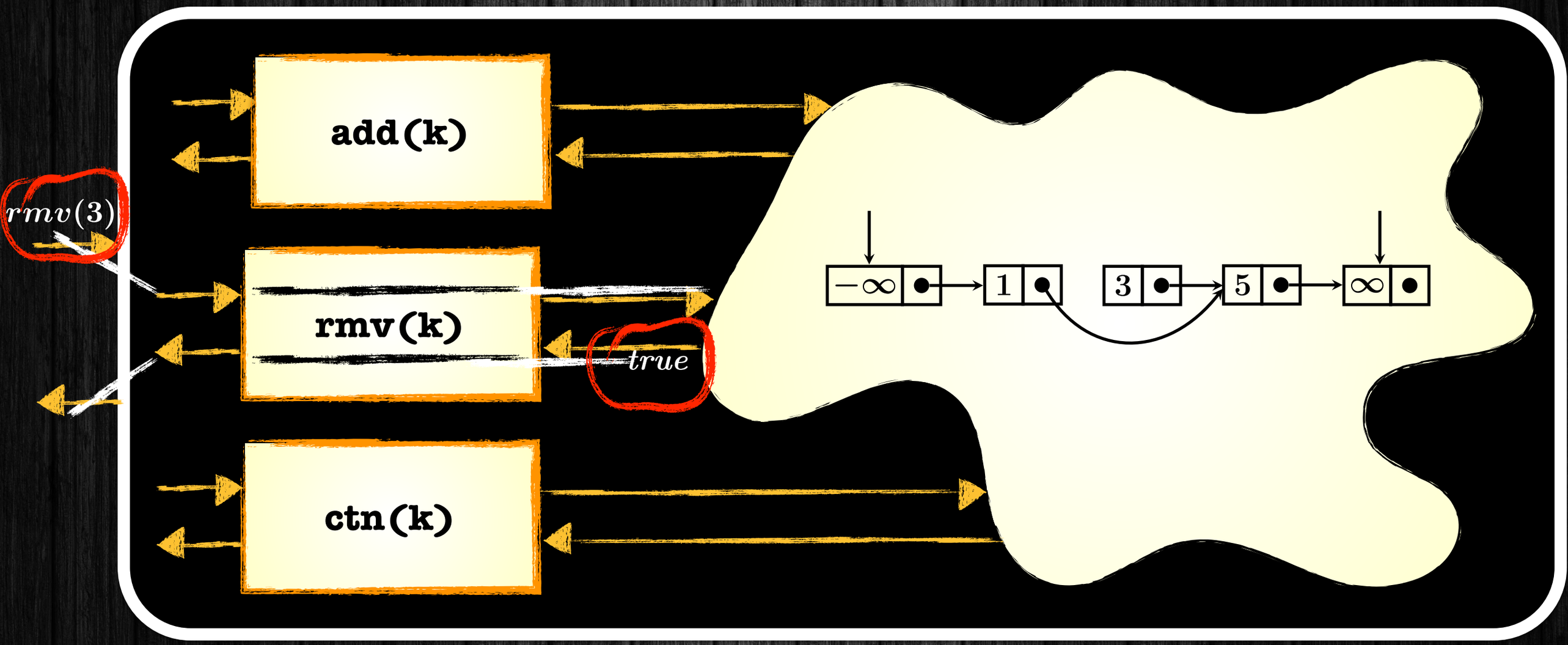
Sets | Sequential Programs



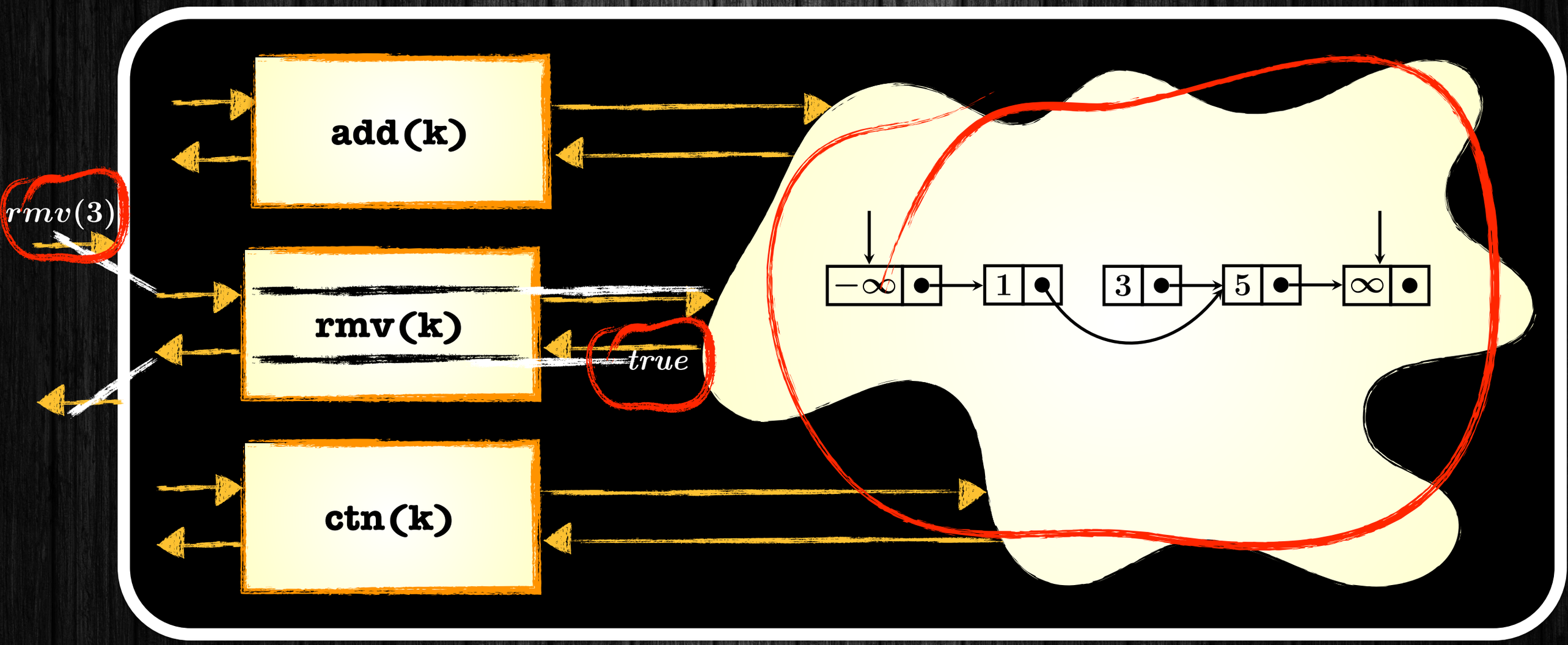
Sets | Sequential Programs



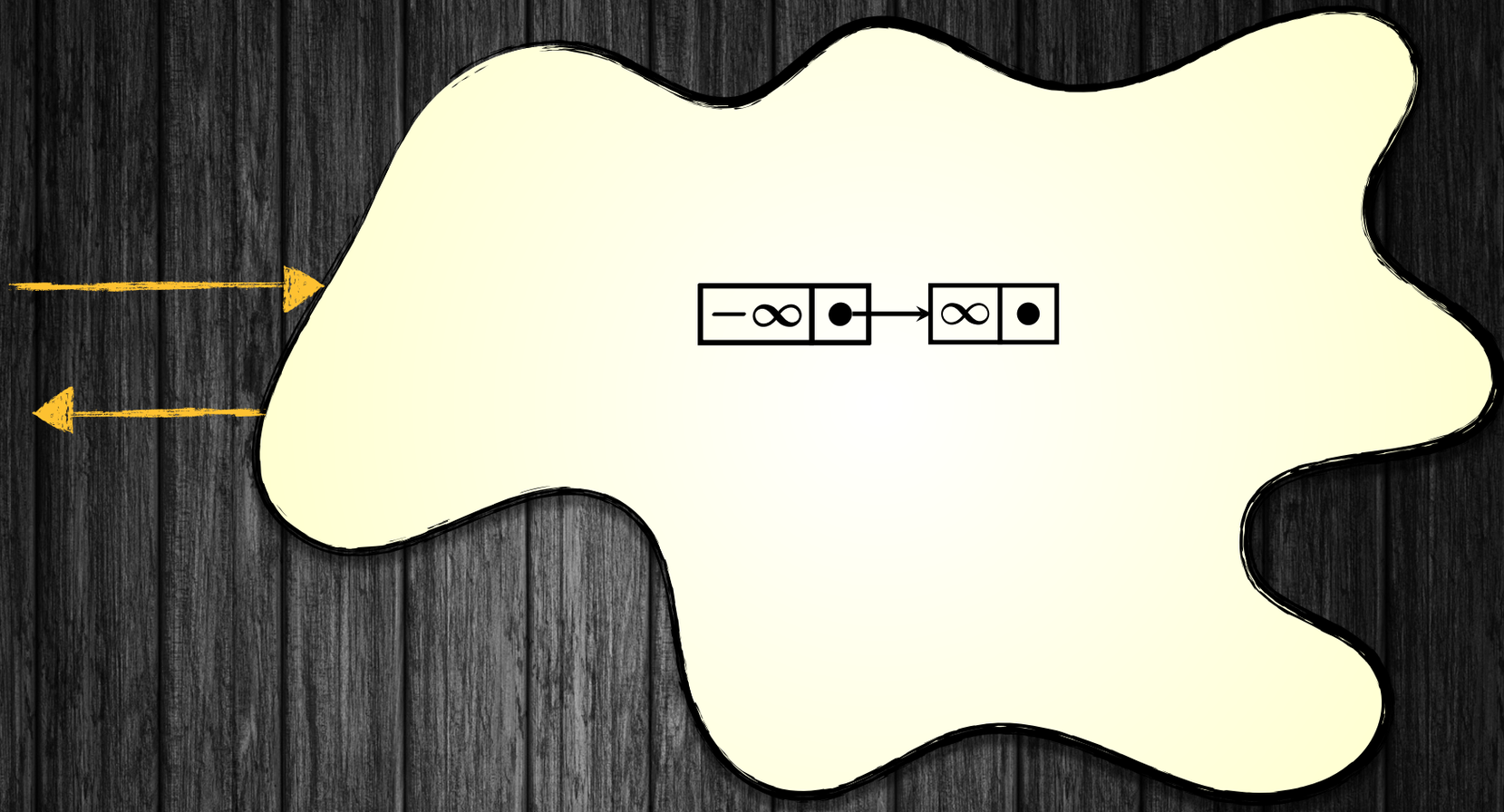
Sets | Sequential Programs

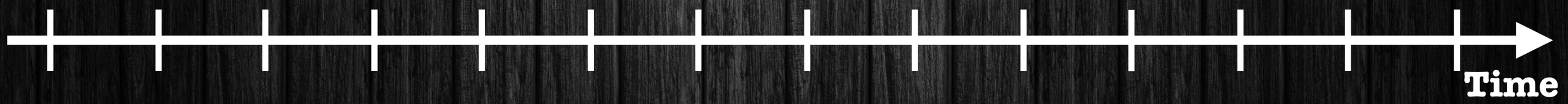


Sets | Sequential Programs



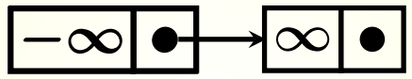
Sets | Sequential Programs





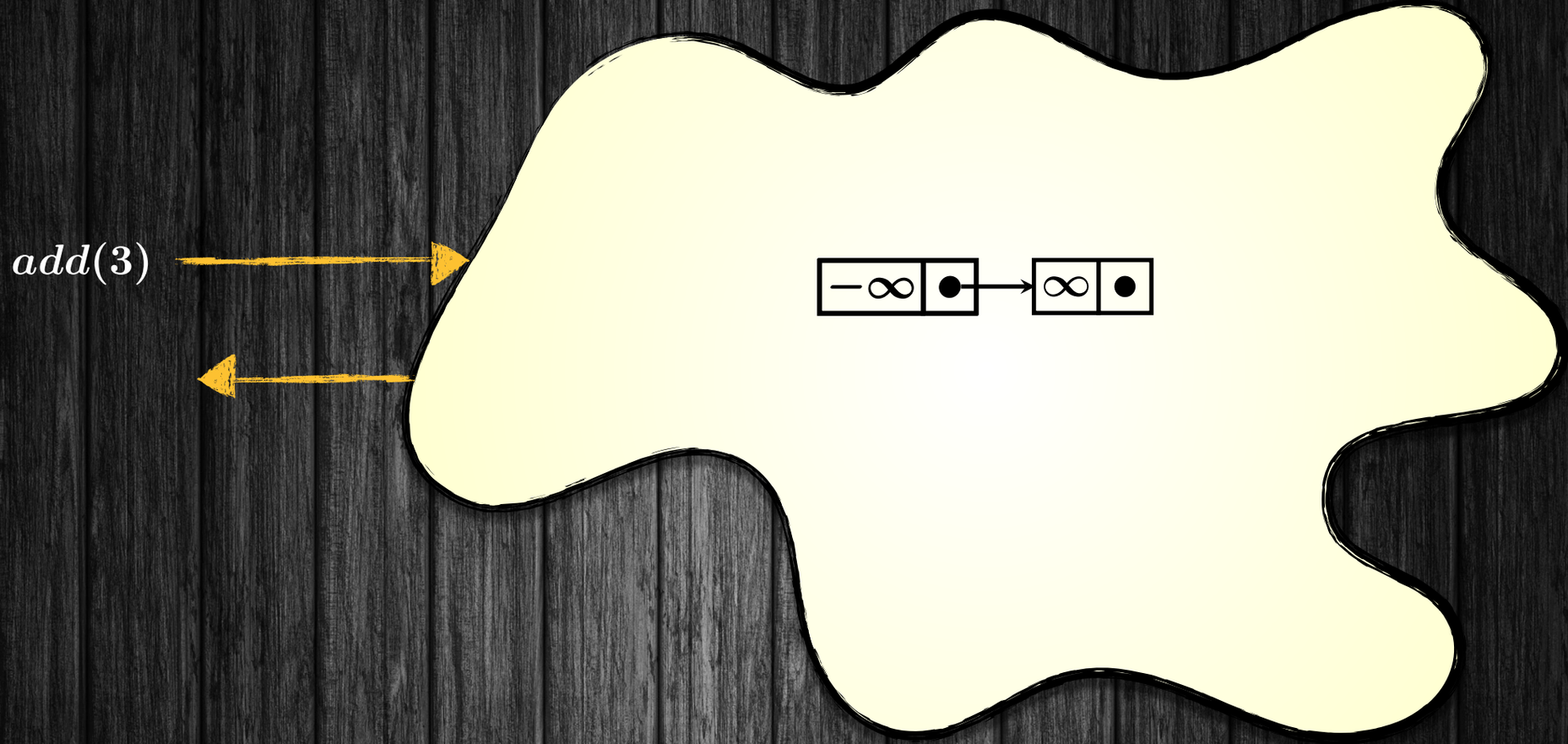
Sets | Sequential Programs

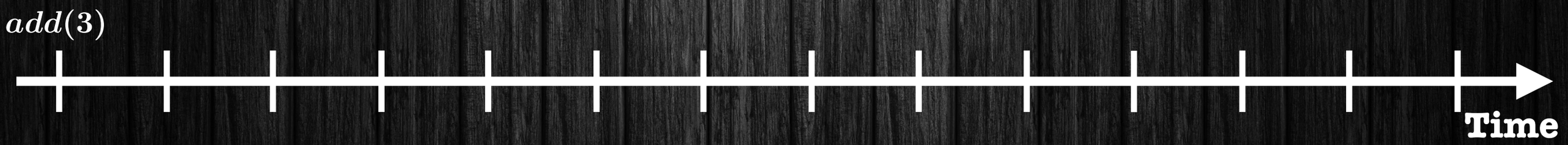
$add(3)$



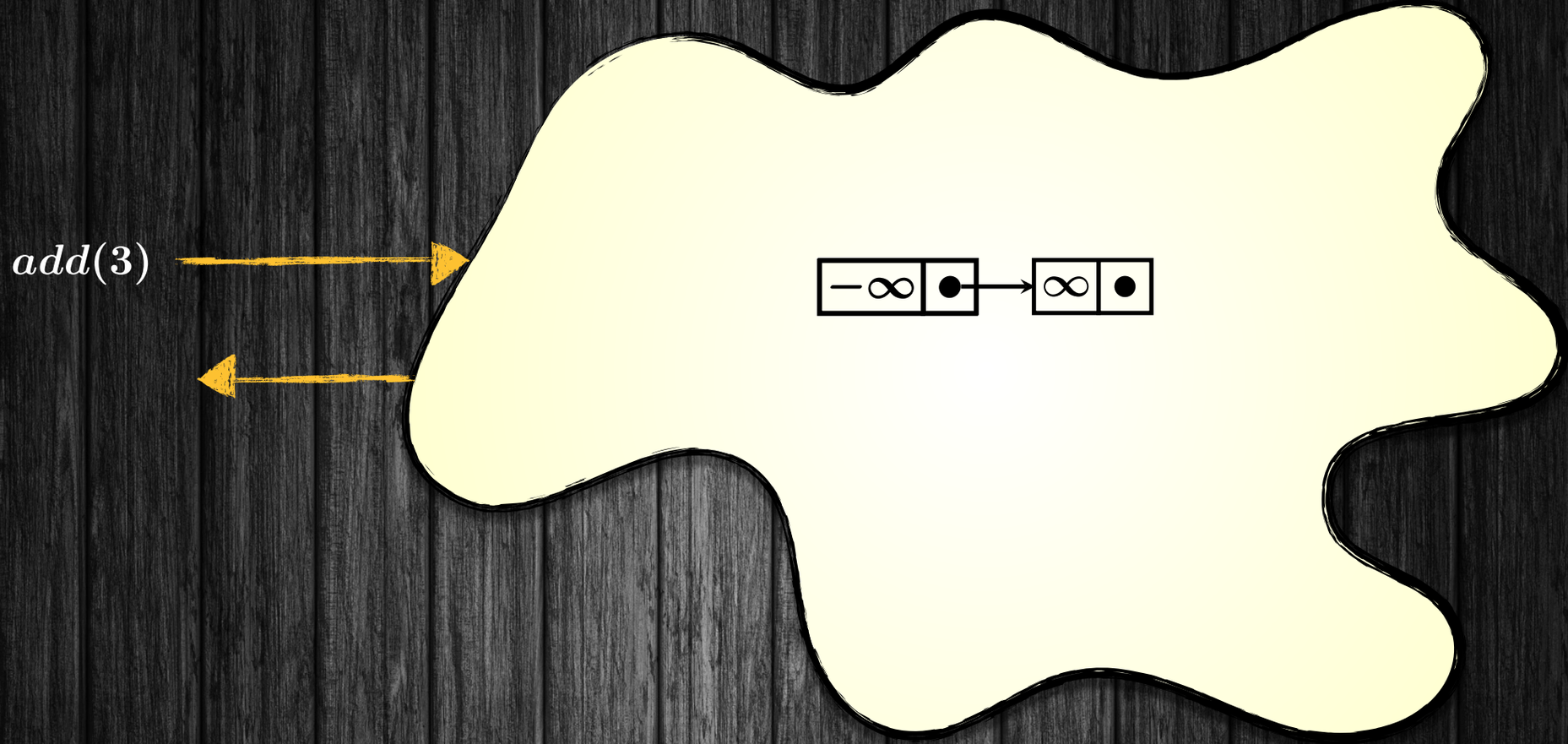


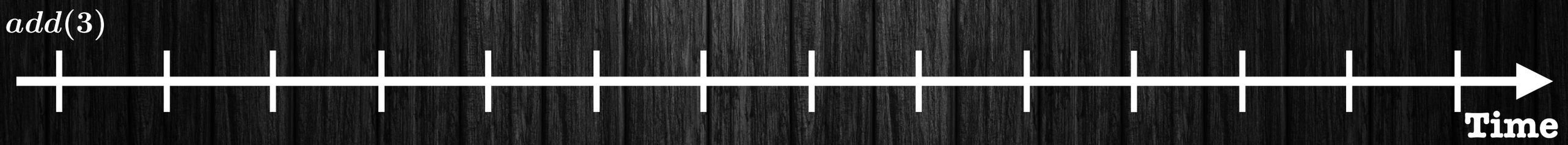
Sets | Sequential Programs



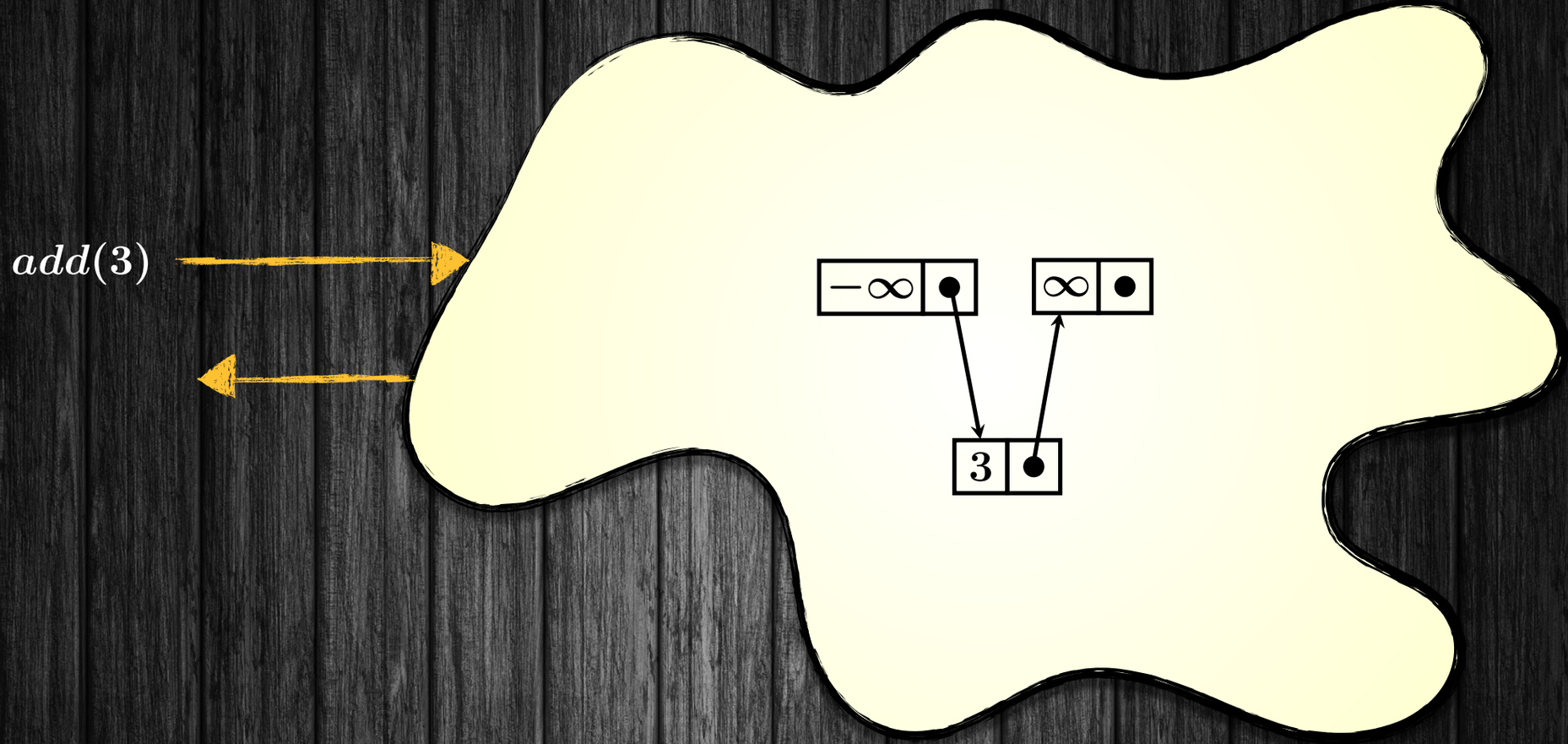


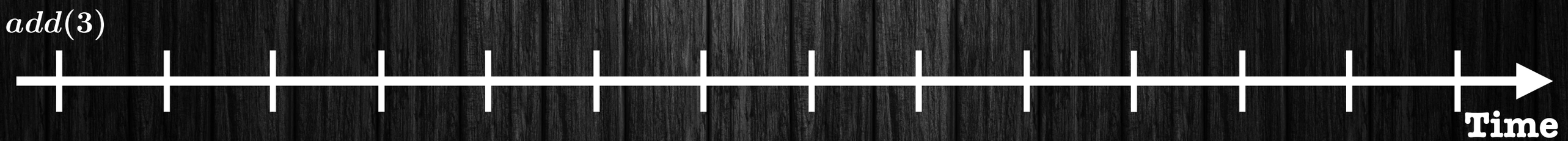
Sets | Sequential Programs



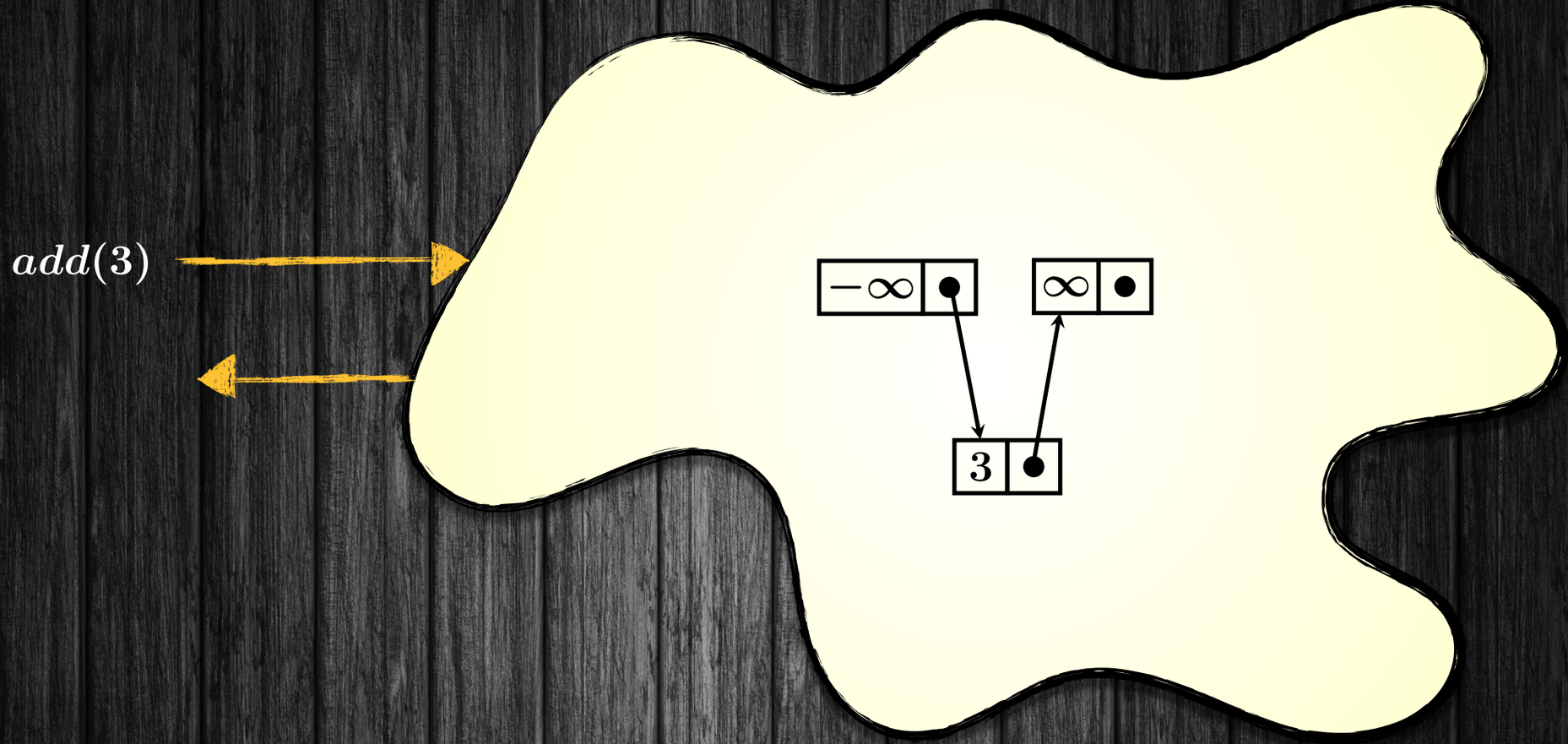


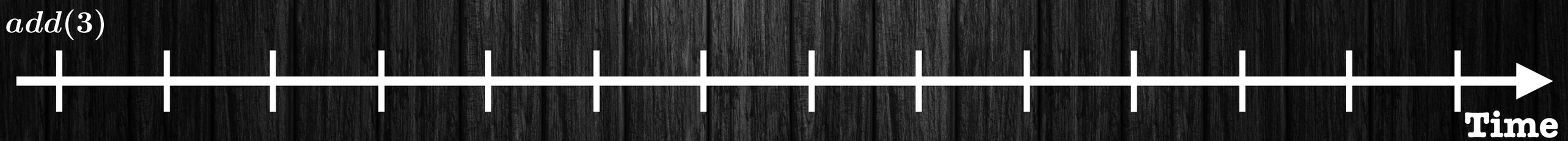
Sets | Sequential Programs



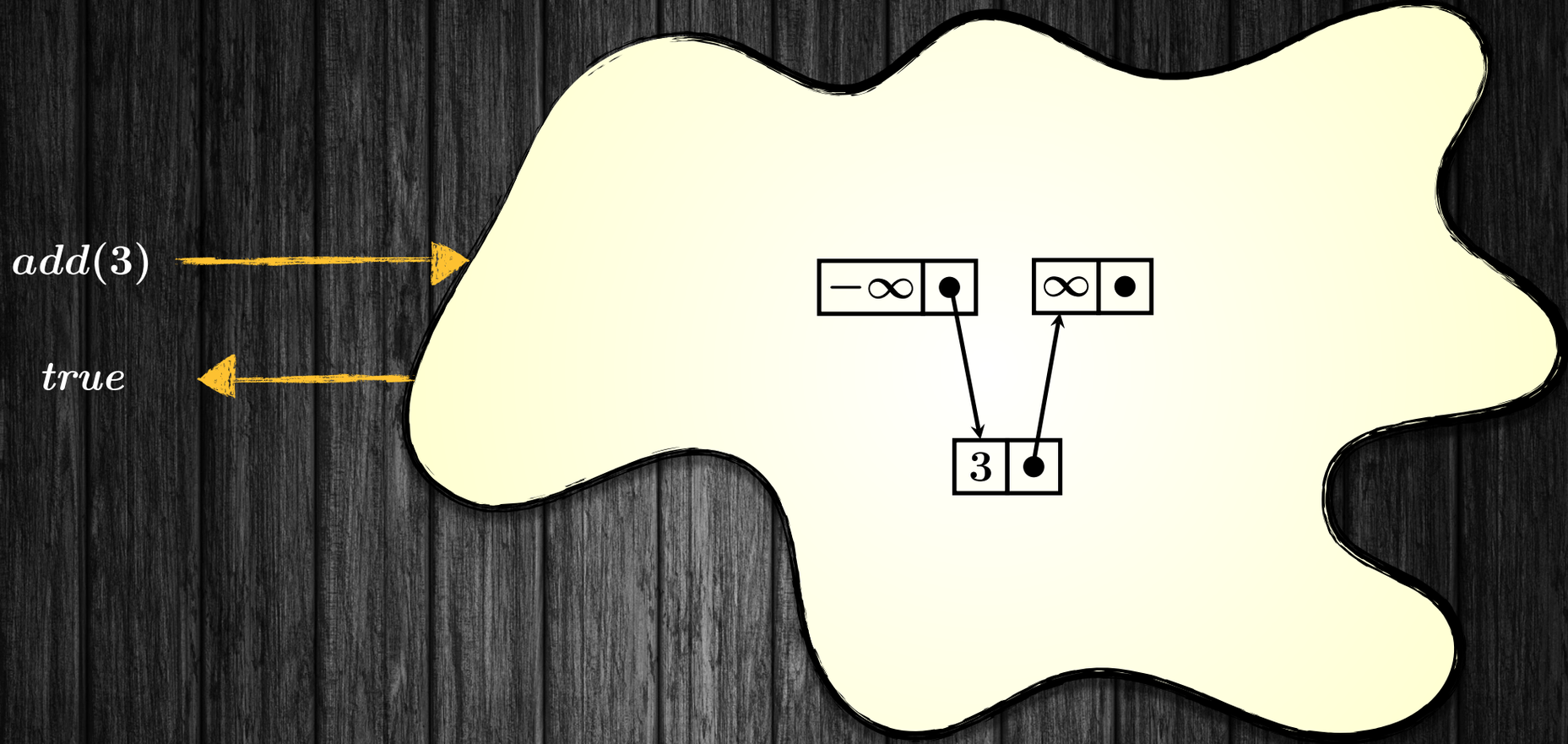


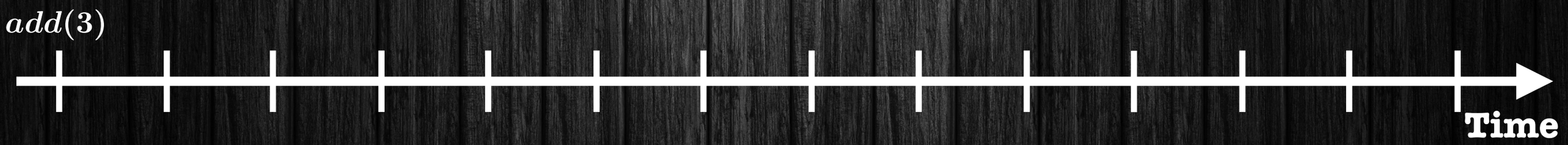
Sets | Sequential Programs



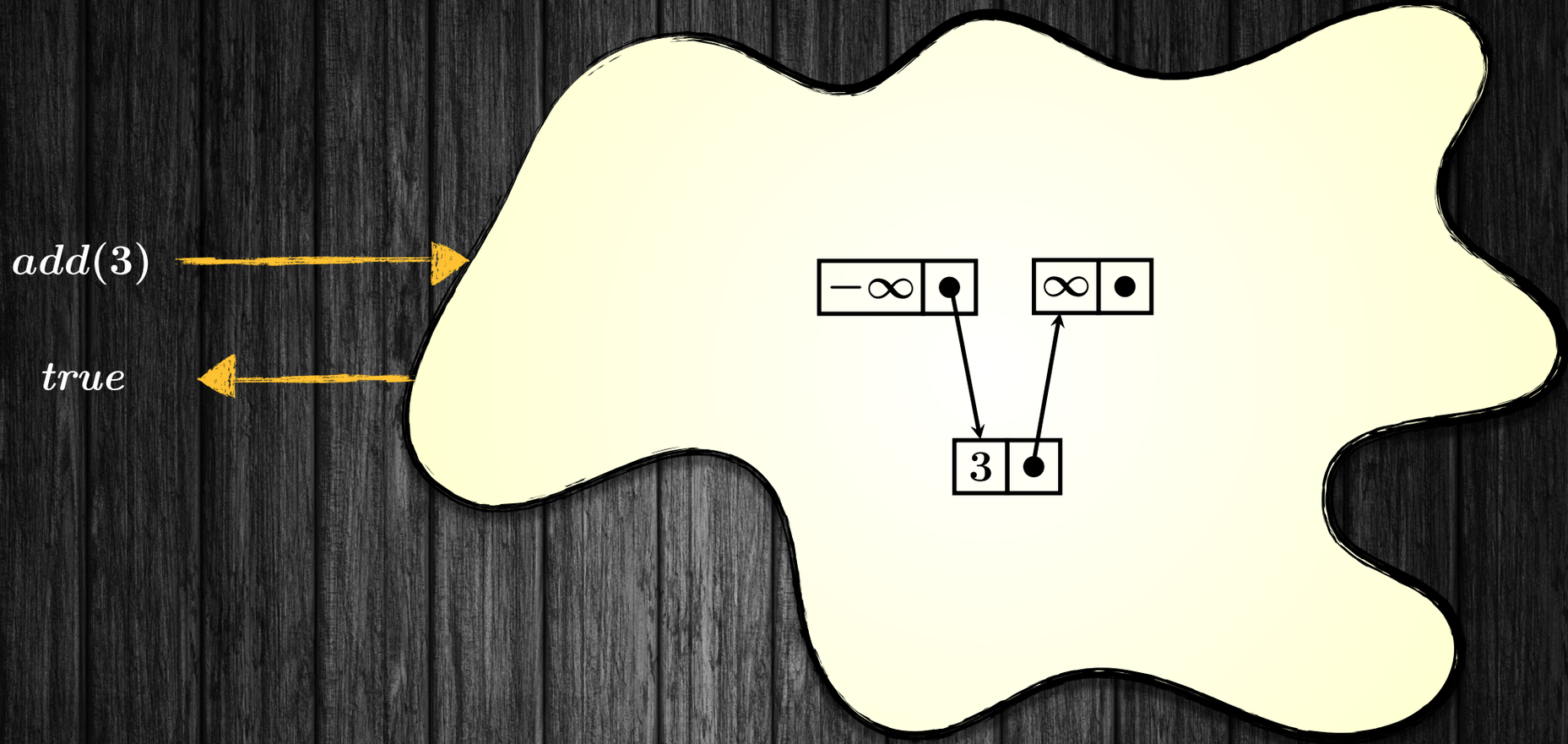


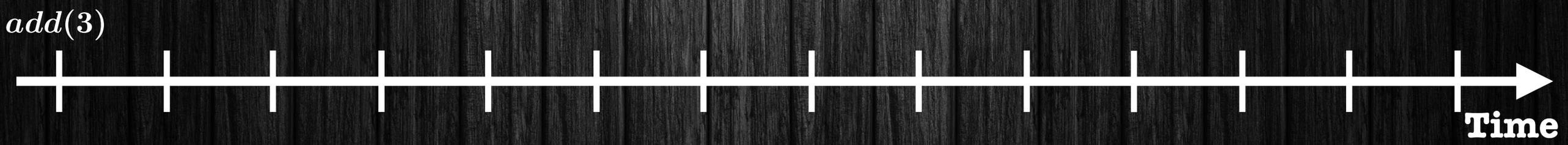
Sets | Sequential Programs



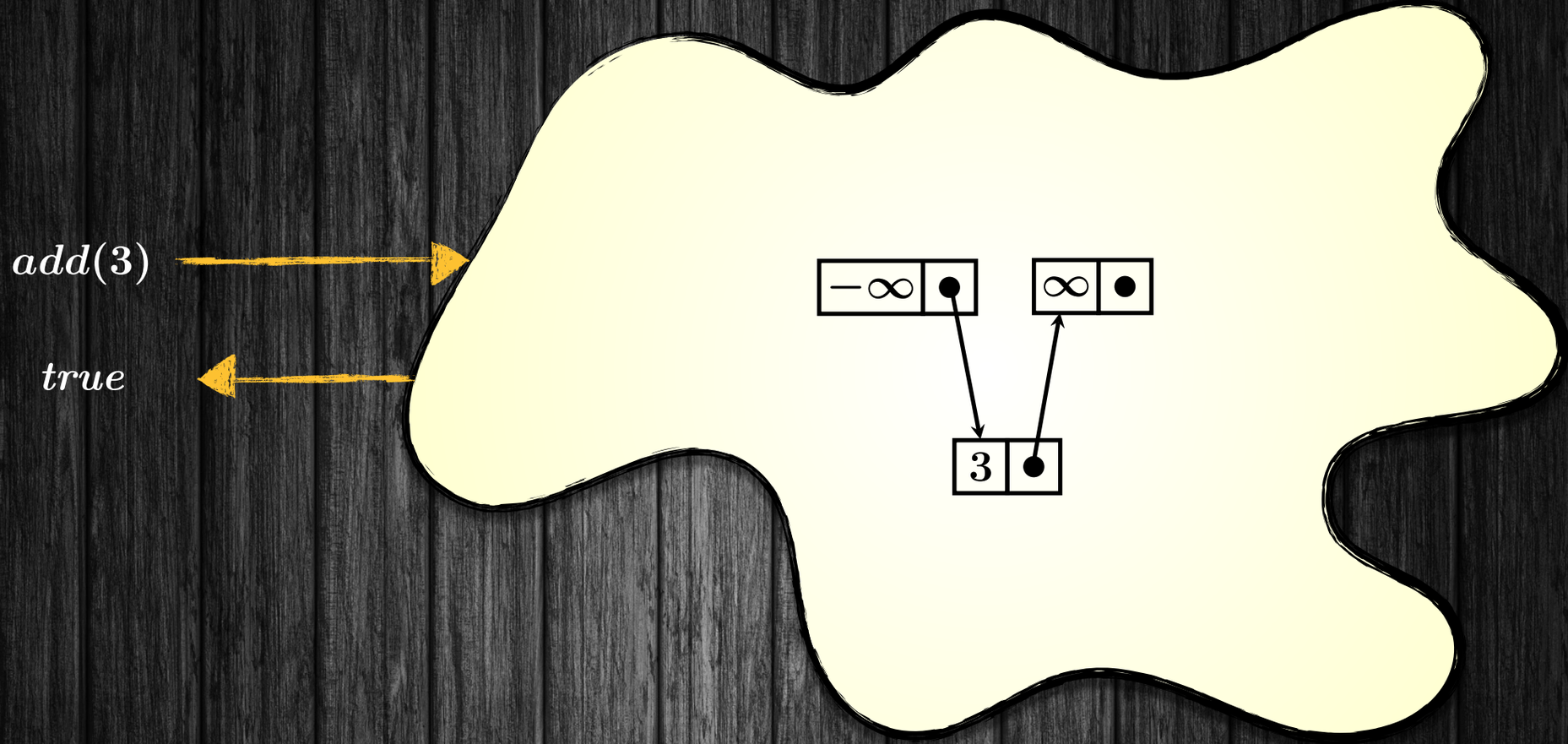


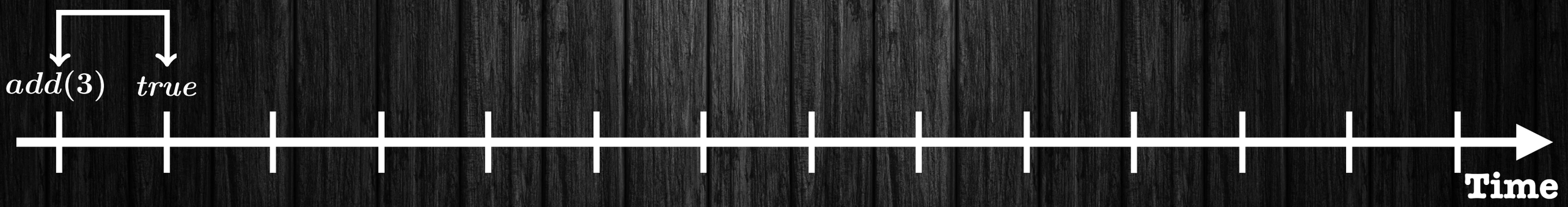
Sets | Sequential Programs



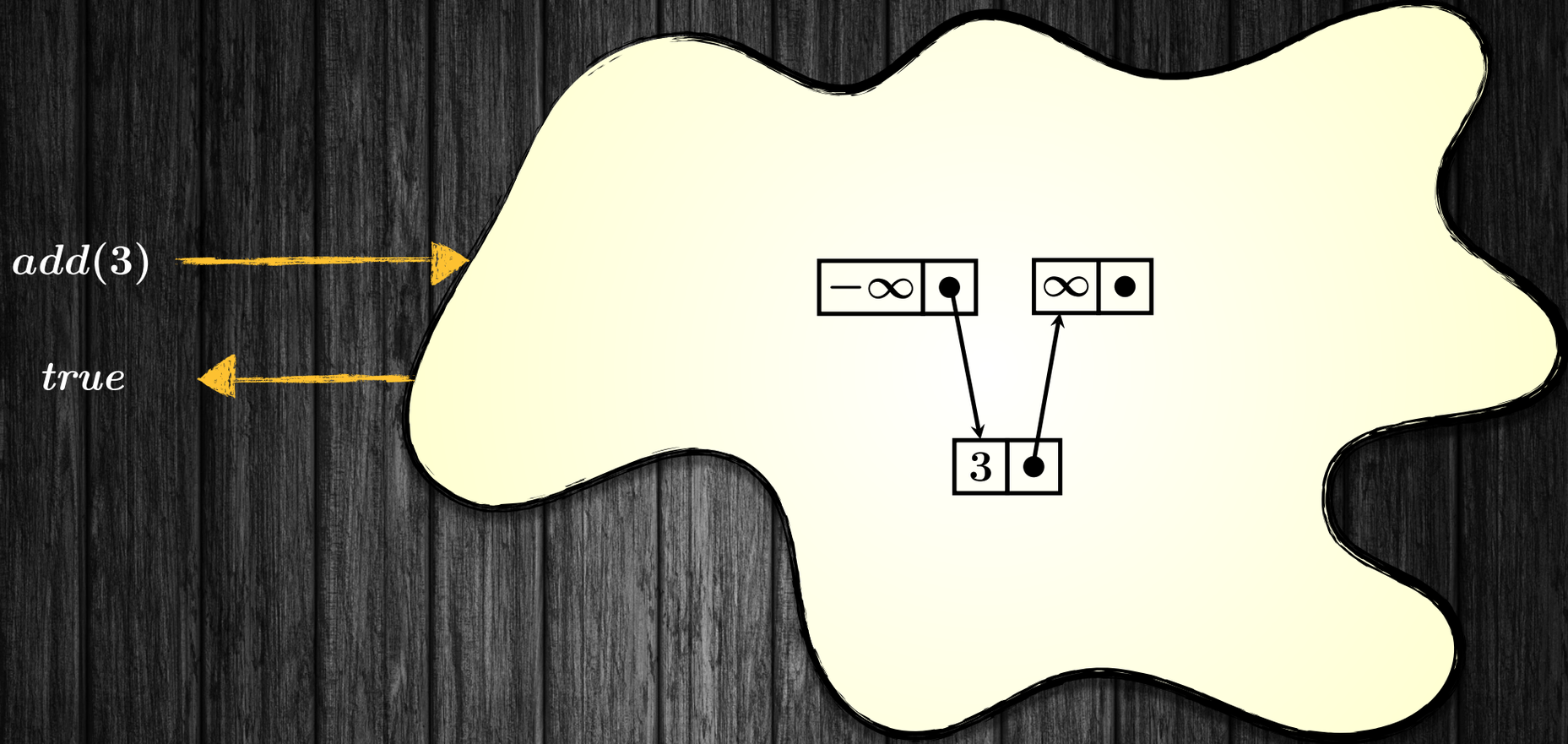


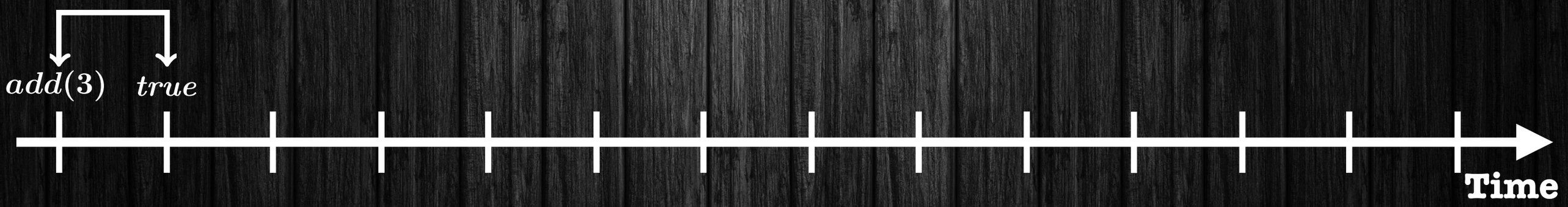
Sets | Sequential Programs



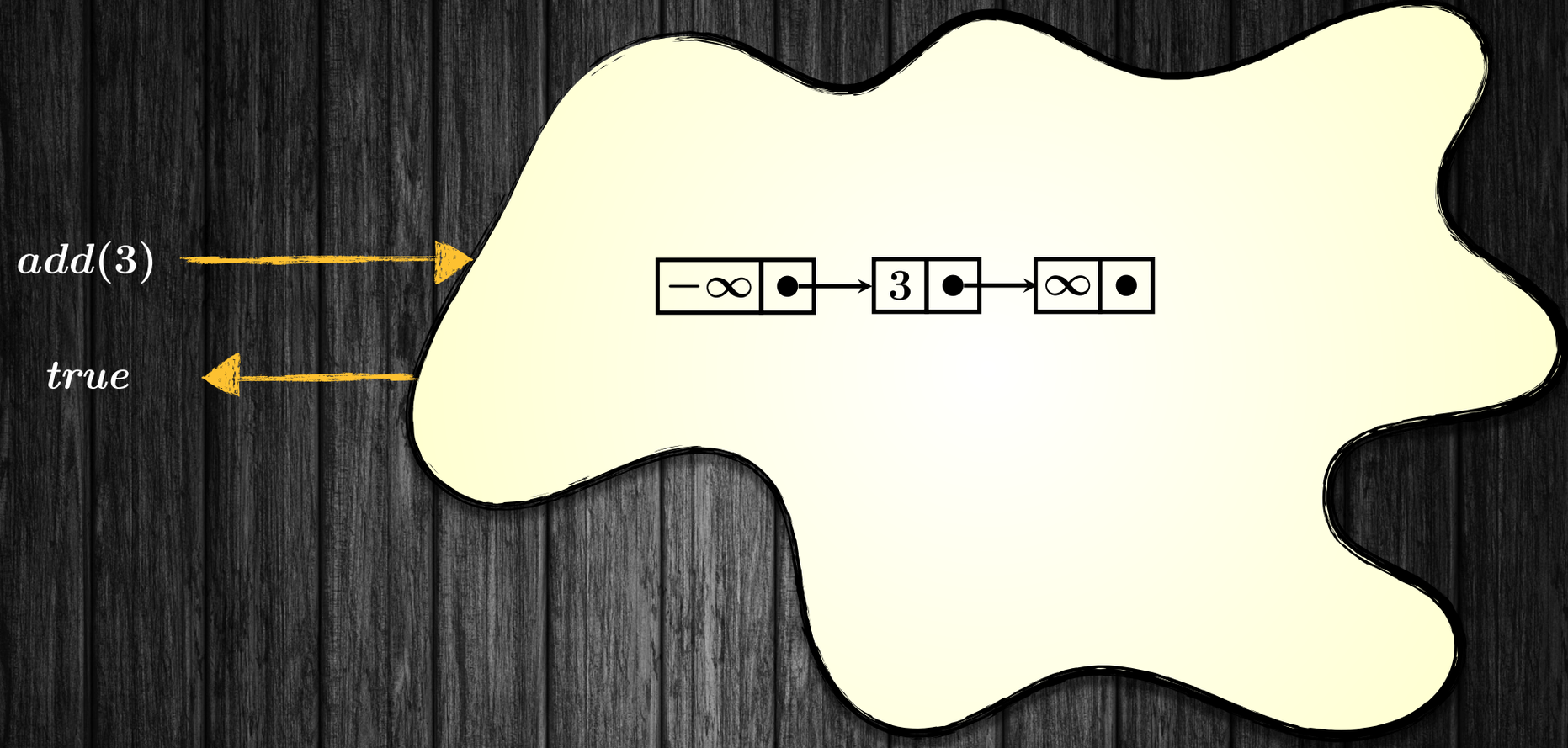


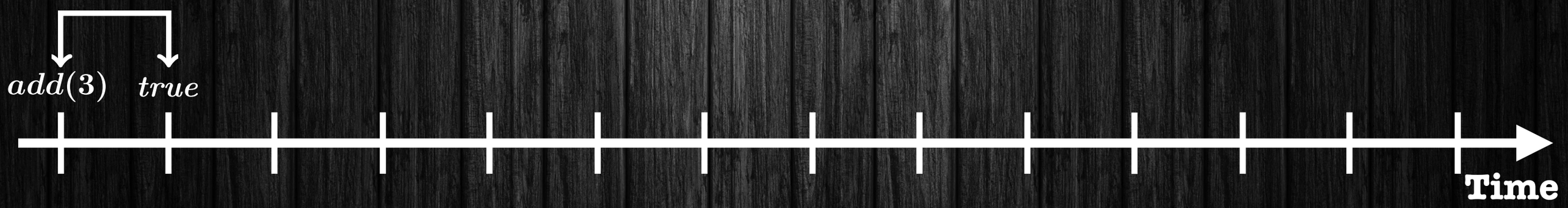
Sets | Sequential Programs



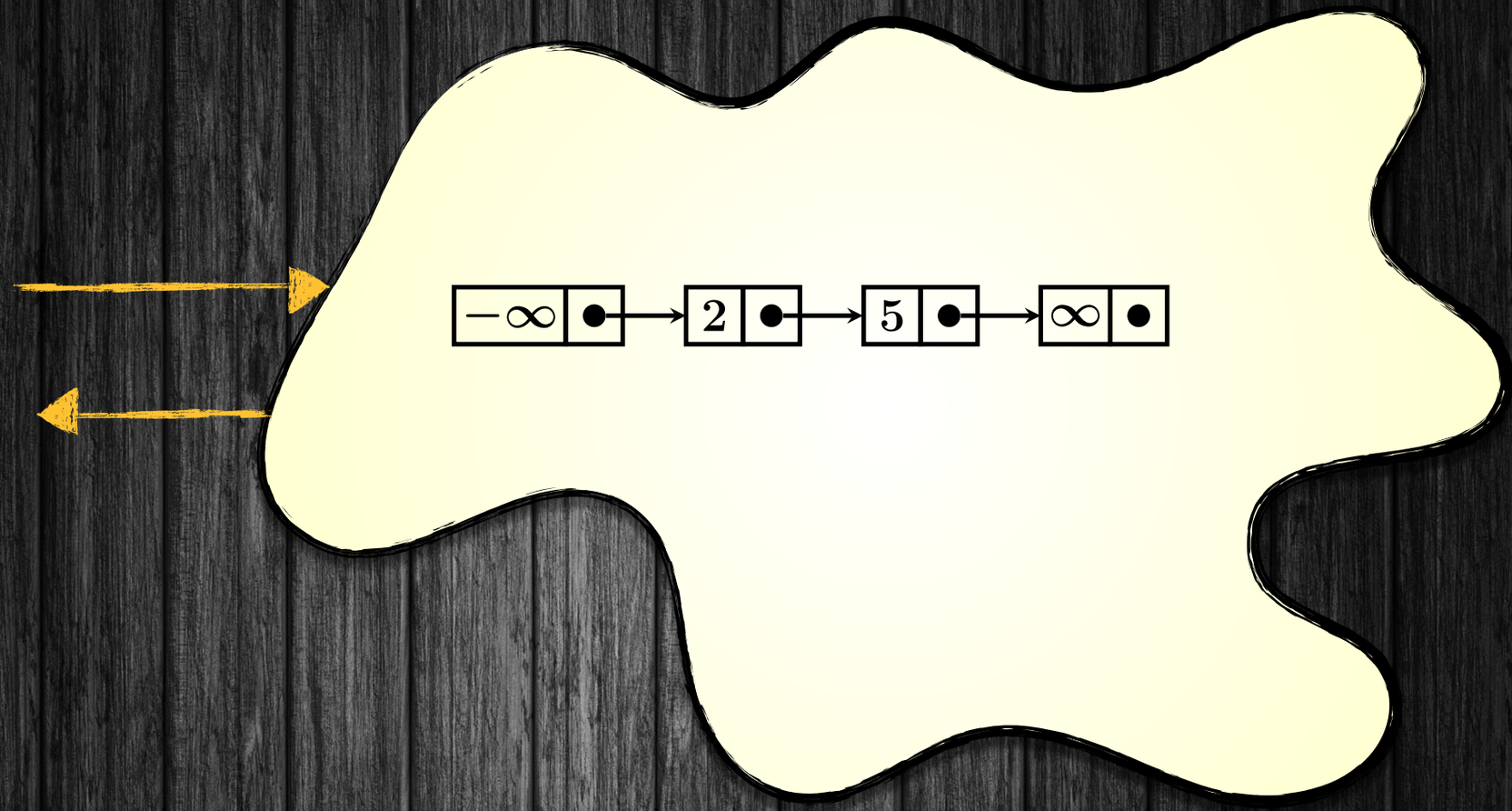


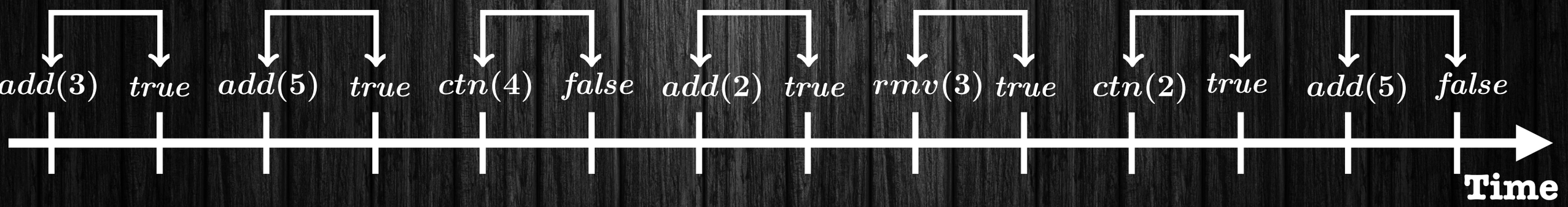
Sets | Sequential Programs





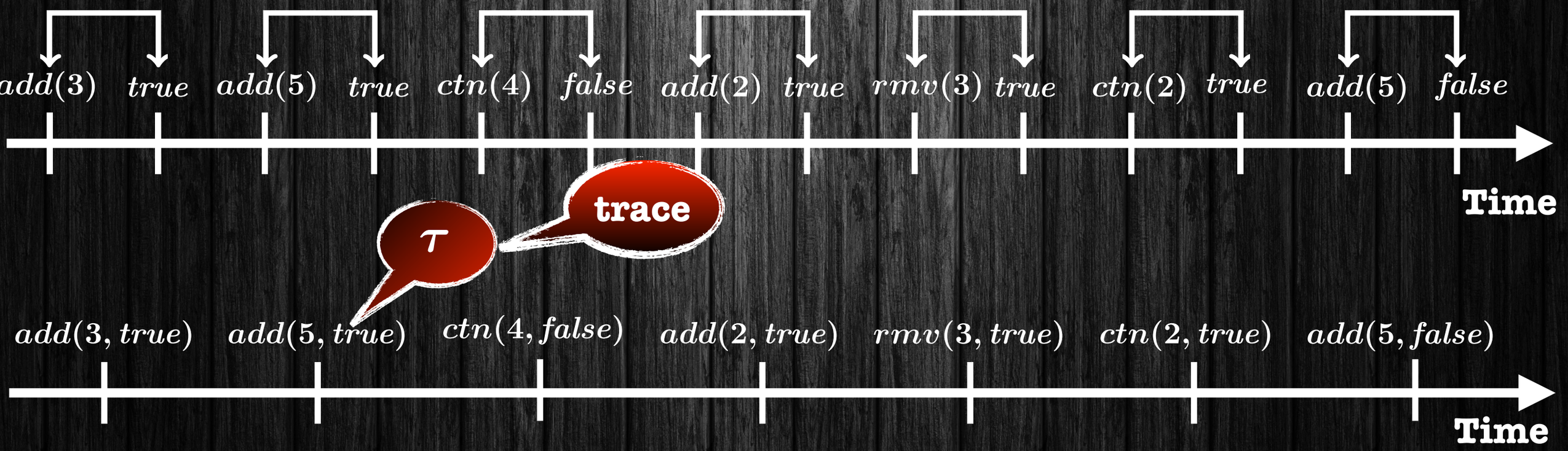
Sets | Sequential Programs





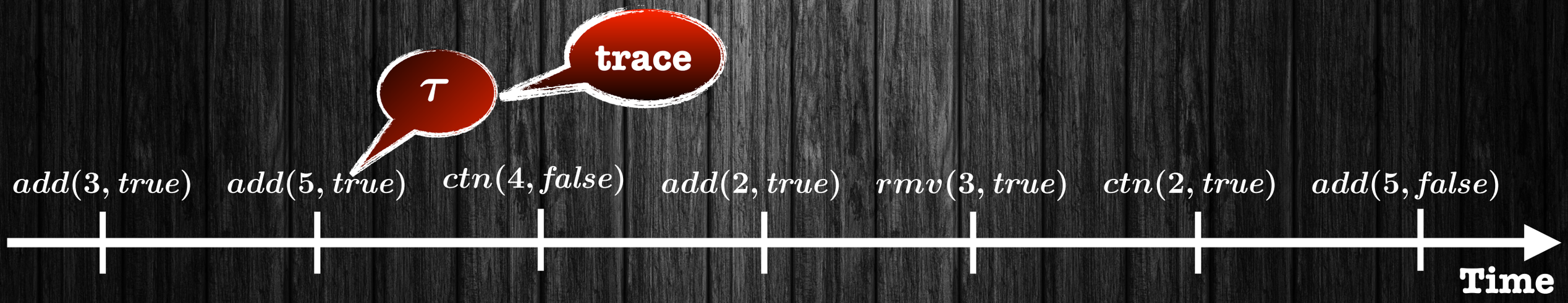
Sets | Sequential Programs





Sets | Sequential Programs





Sets | Sequential Programs | Correctness

Safety Properties

Traces(P)

Traces(Set)

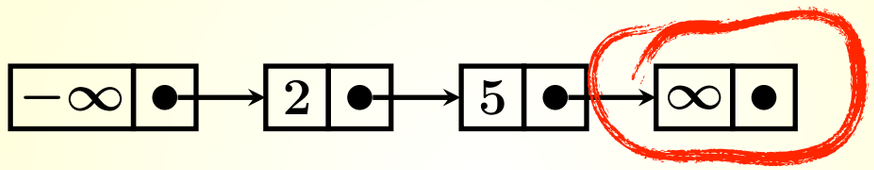
- Nothing bad will happen
- No wrong result will be delivered
- Will not violate the abstract data type
- Traces(P) $\subseteq$ Traces(Set)

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

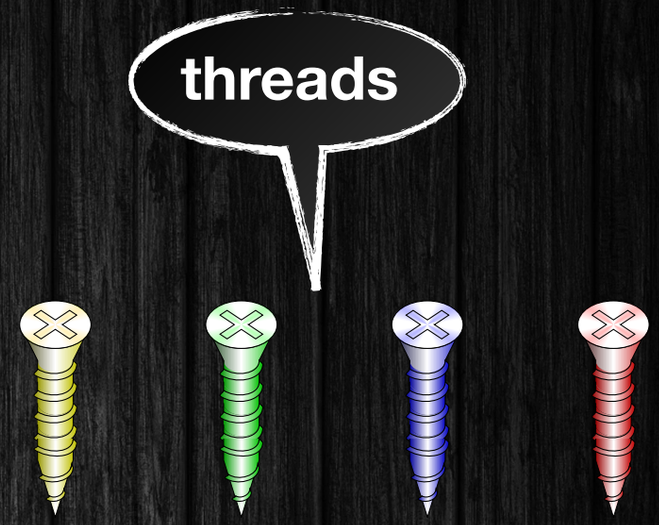
Concurrent Programs

- Concurrent Histories
- Linearizability

Concurrent Programs

- Concurrent Histories
- Linearizability

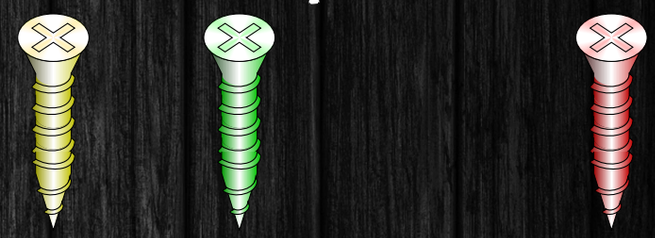
Concurrent Histories

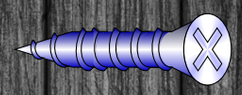


Program

Concurrent Histories

threads

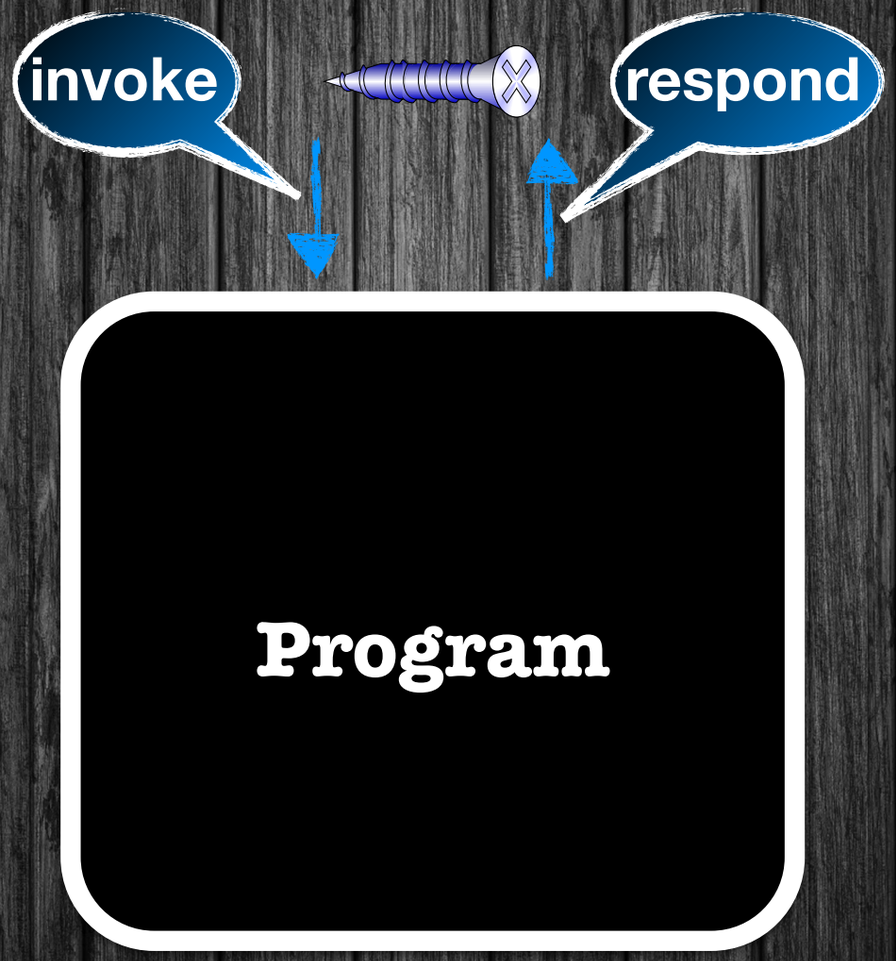




Program

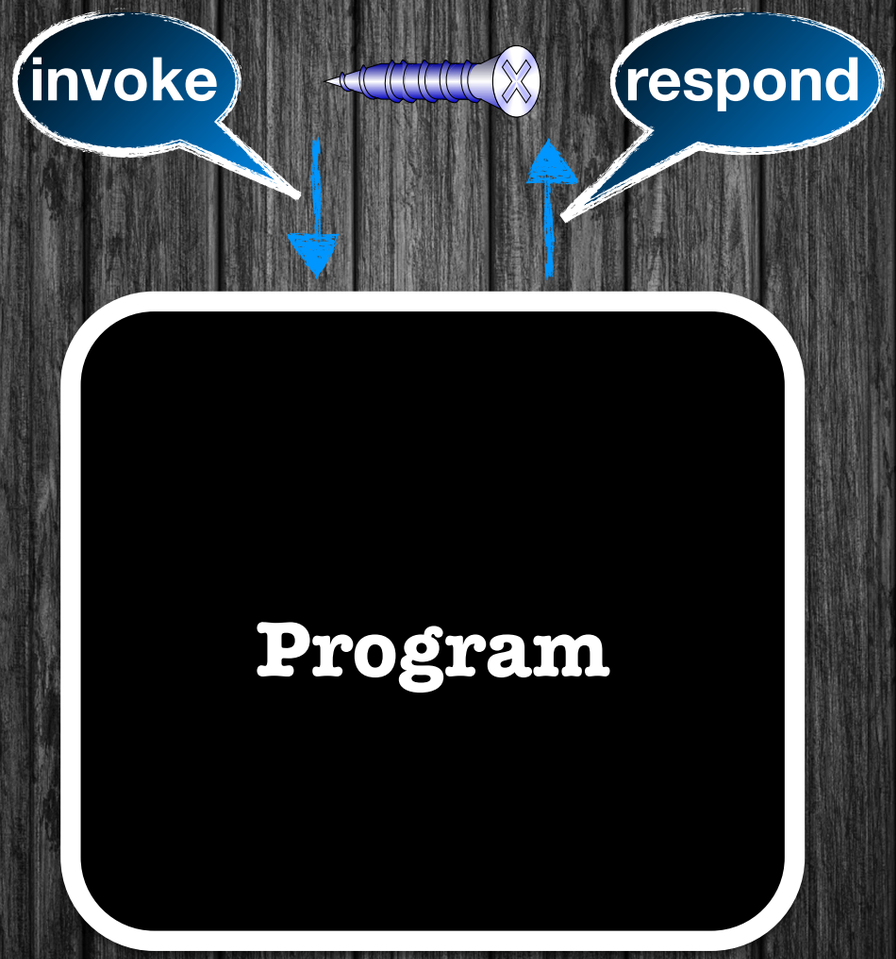
Concurrent Histories



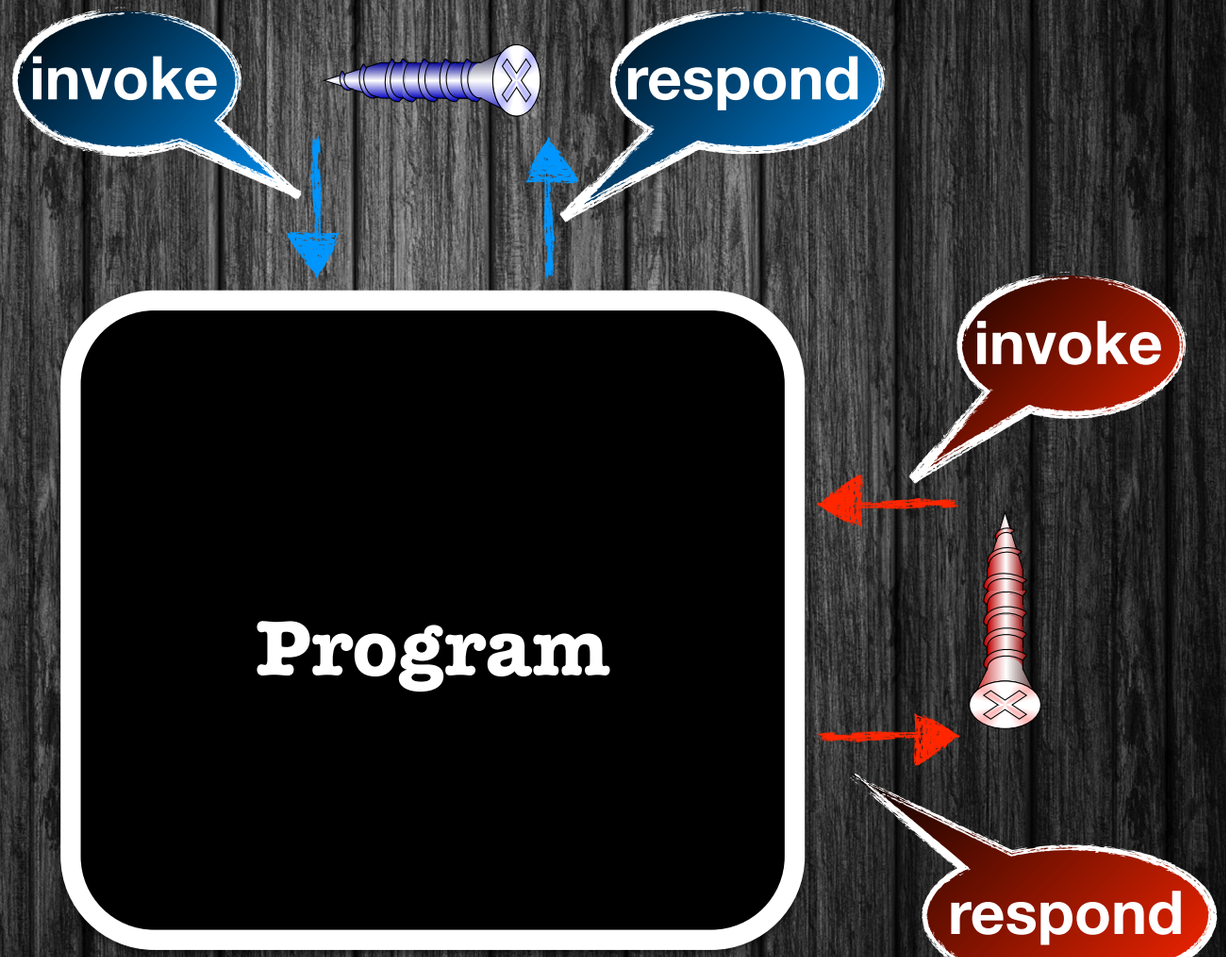


Concurrent Histories

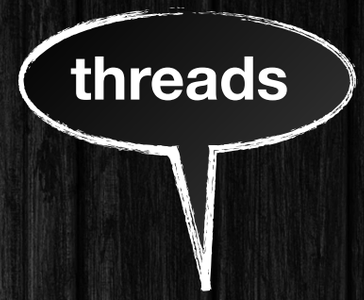




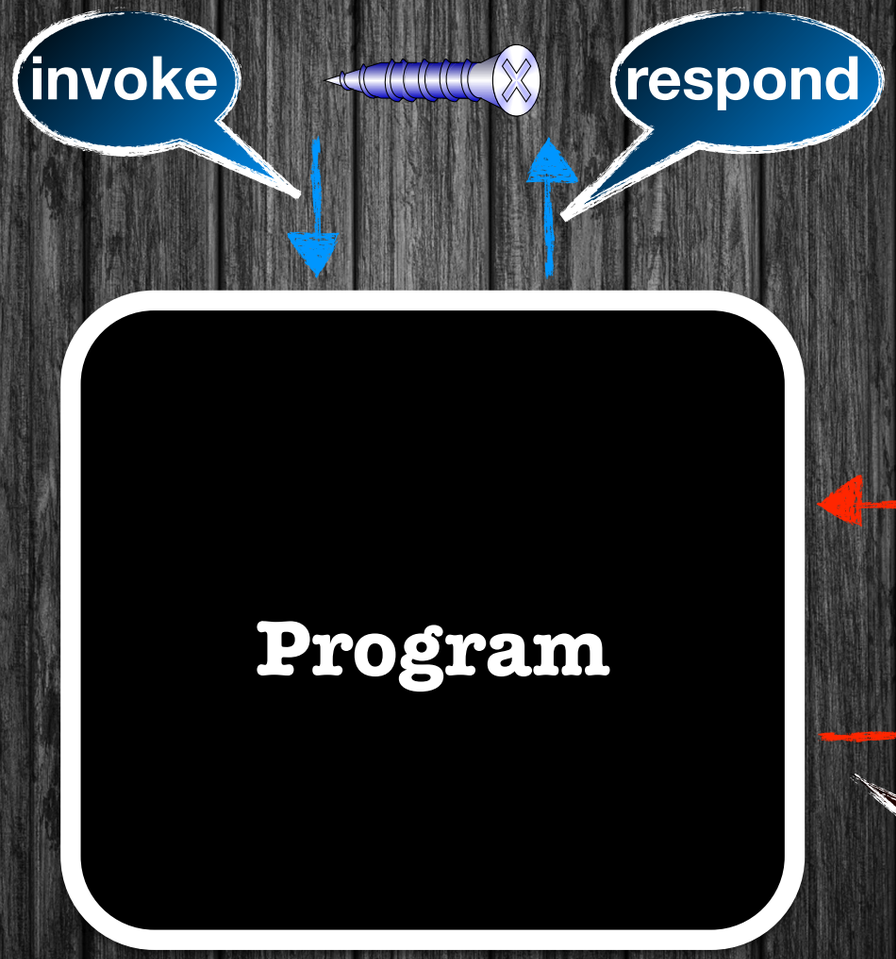
Concurrent Histories

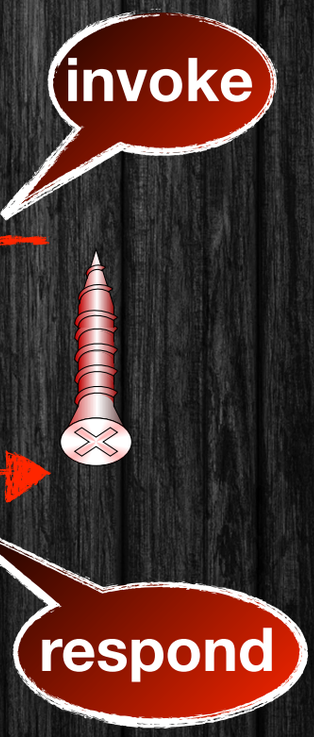


Concurrent Histories







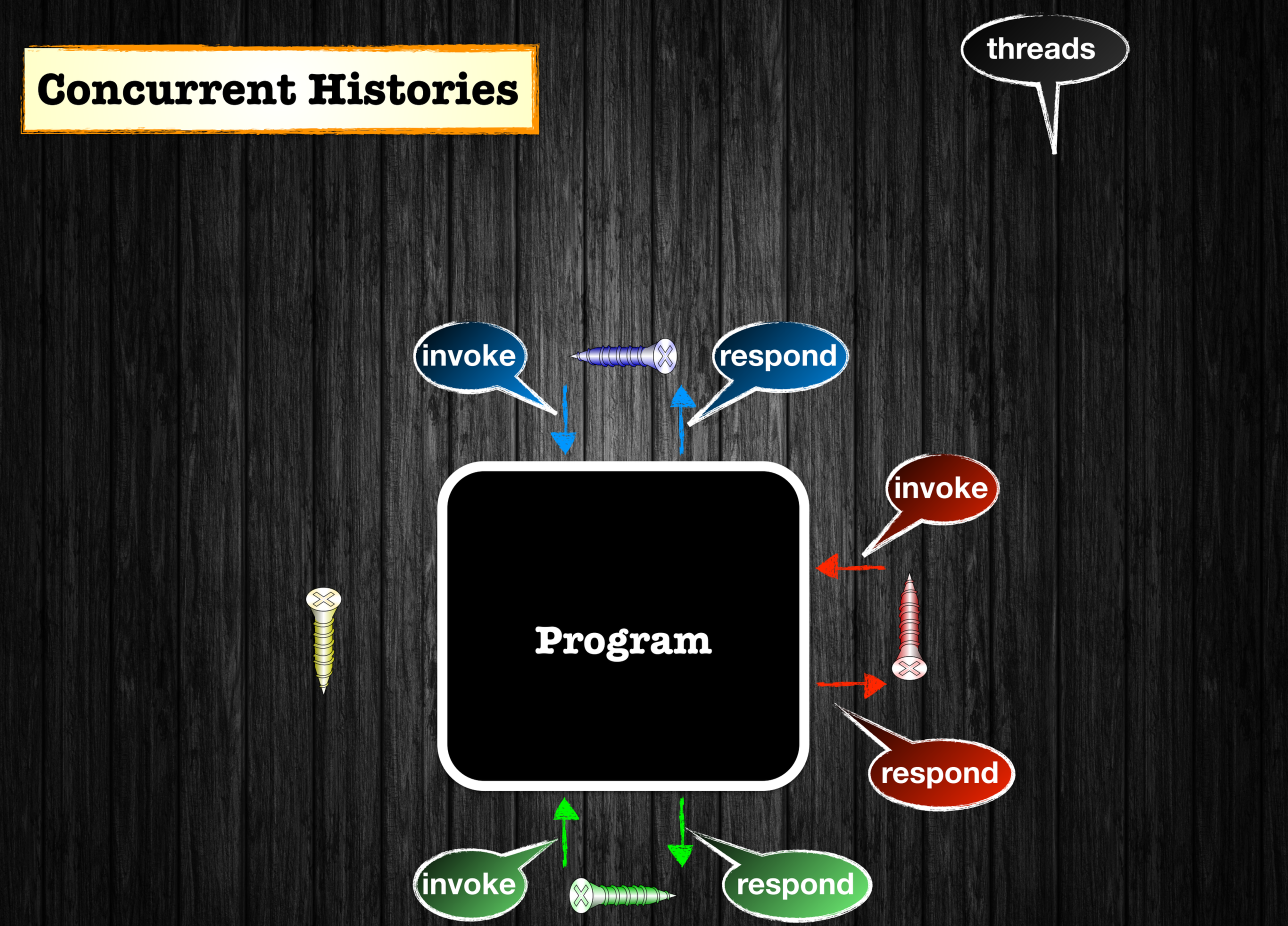




Concurrent Histories



Concurrent Histories



Concurrent Histories

threads



Concurrent Histories

threads

the threads operate concurrently



Concurrent Histories

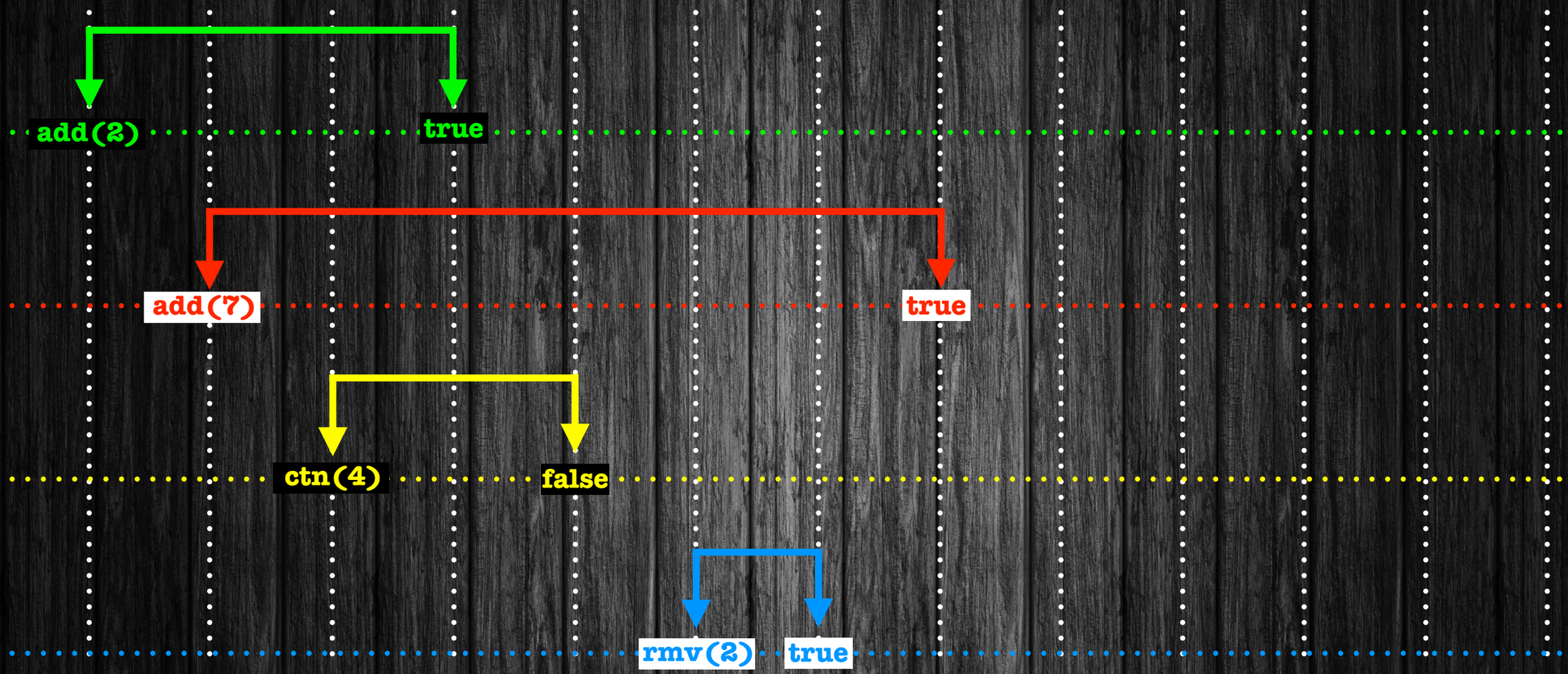
threads

the threads operate concurrently

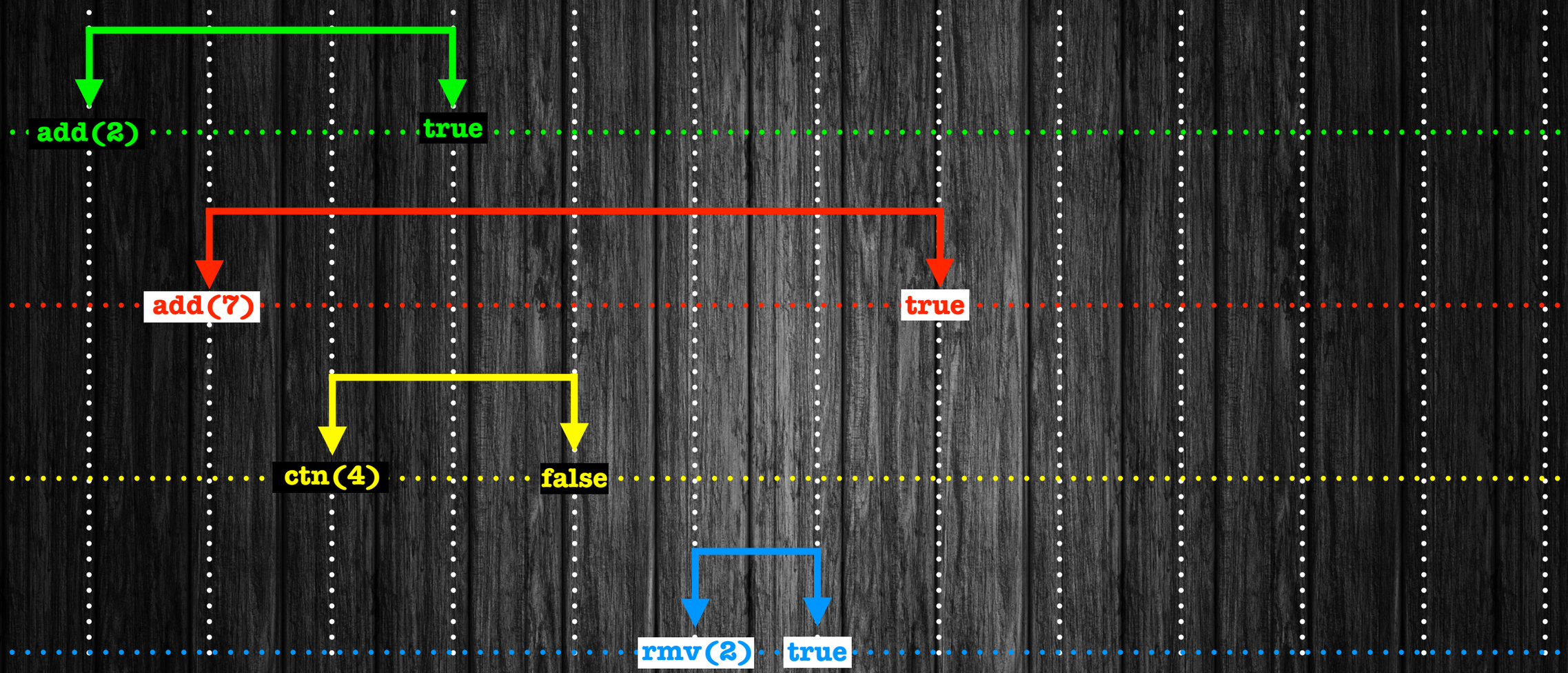
calls may overlap



Concurrent Histories | Sets

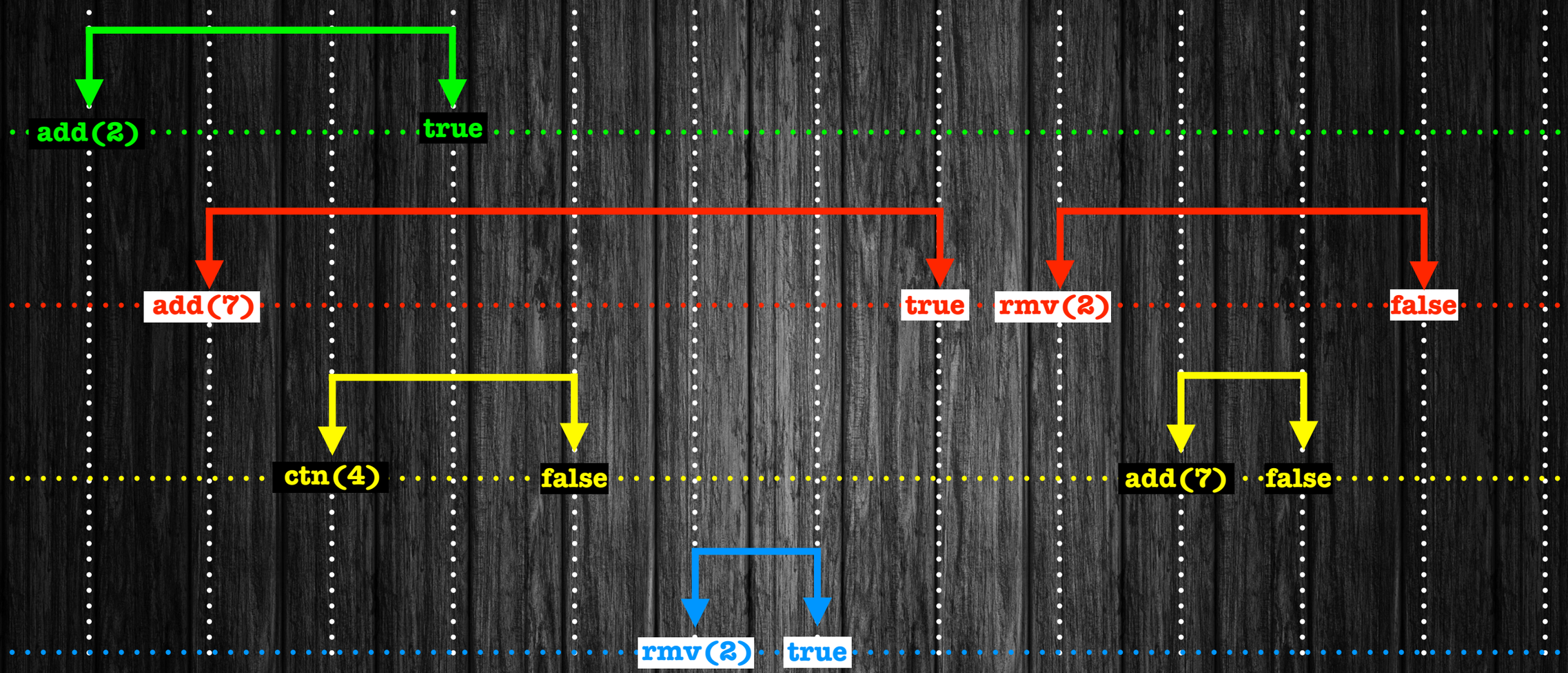


Concurrent Histories | Sets



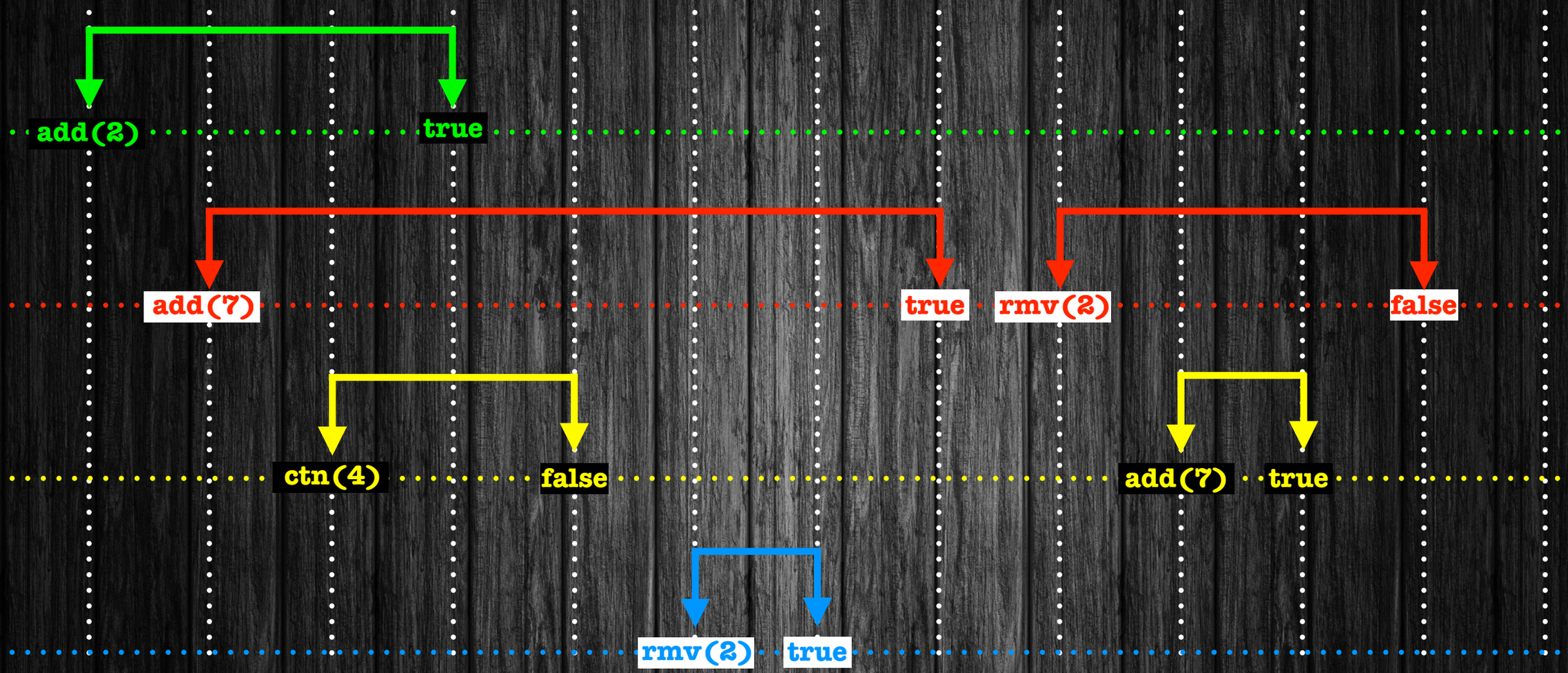
✔

Concurrent Histories | Sets

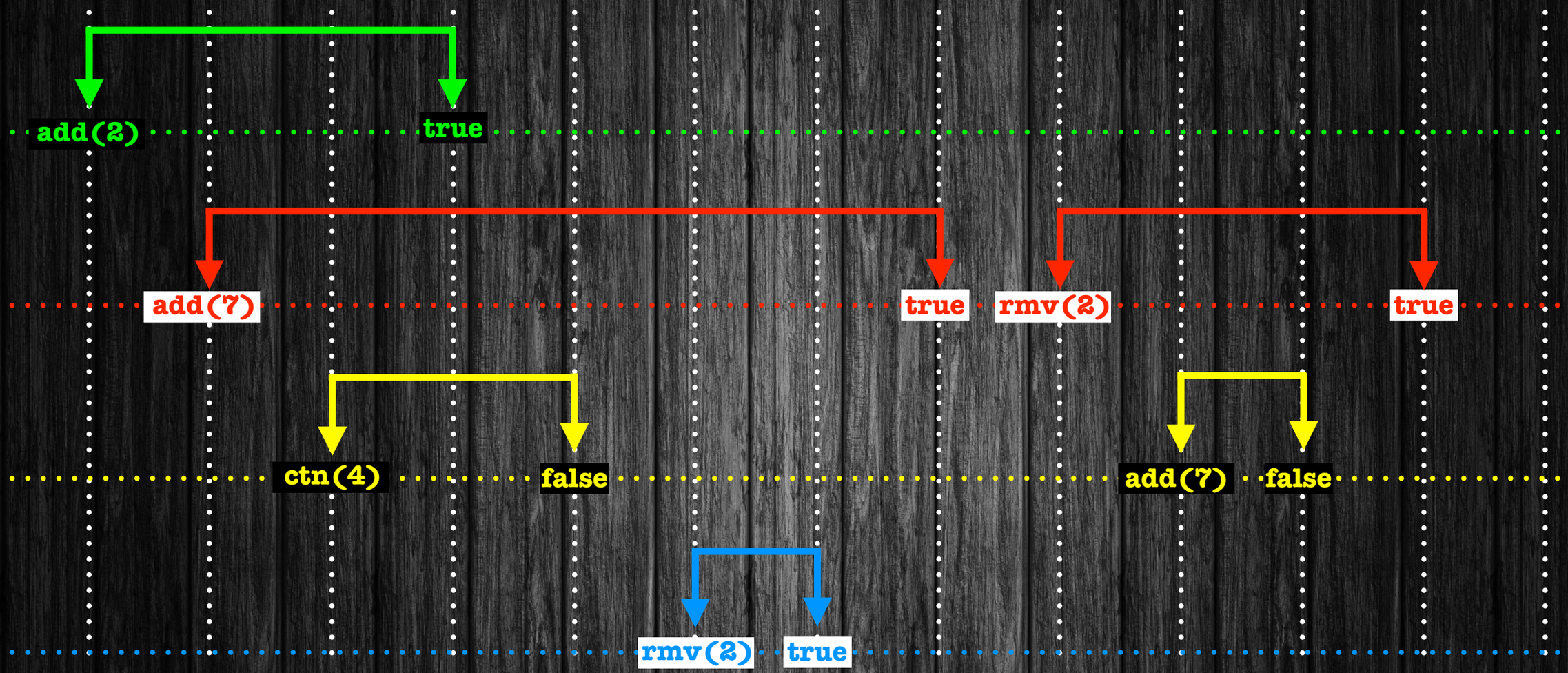


✔

Concurrent Histories | Sets

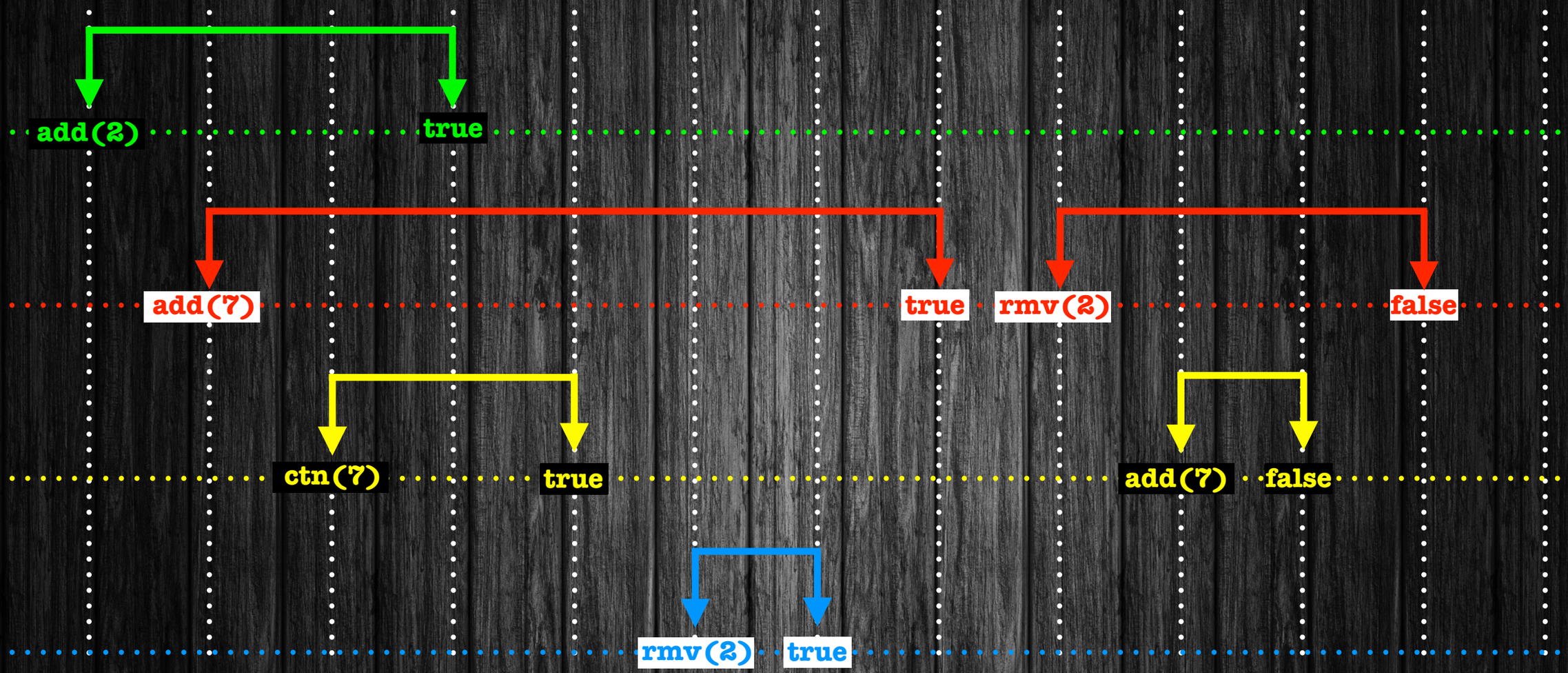


Concurrent Histories | Sets



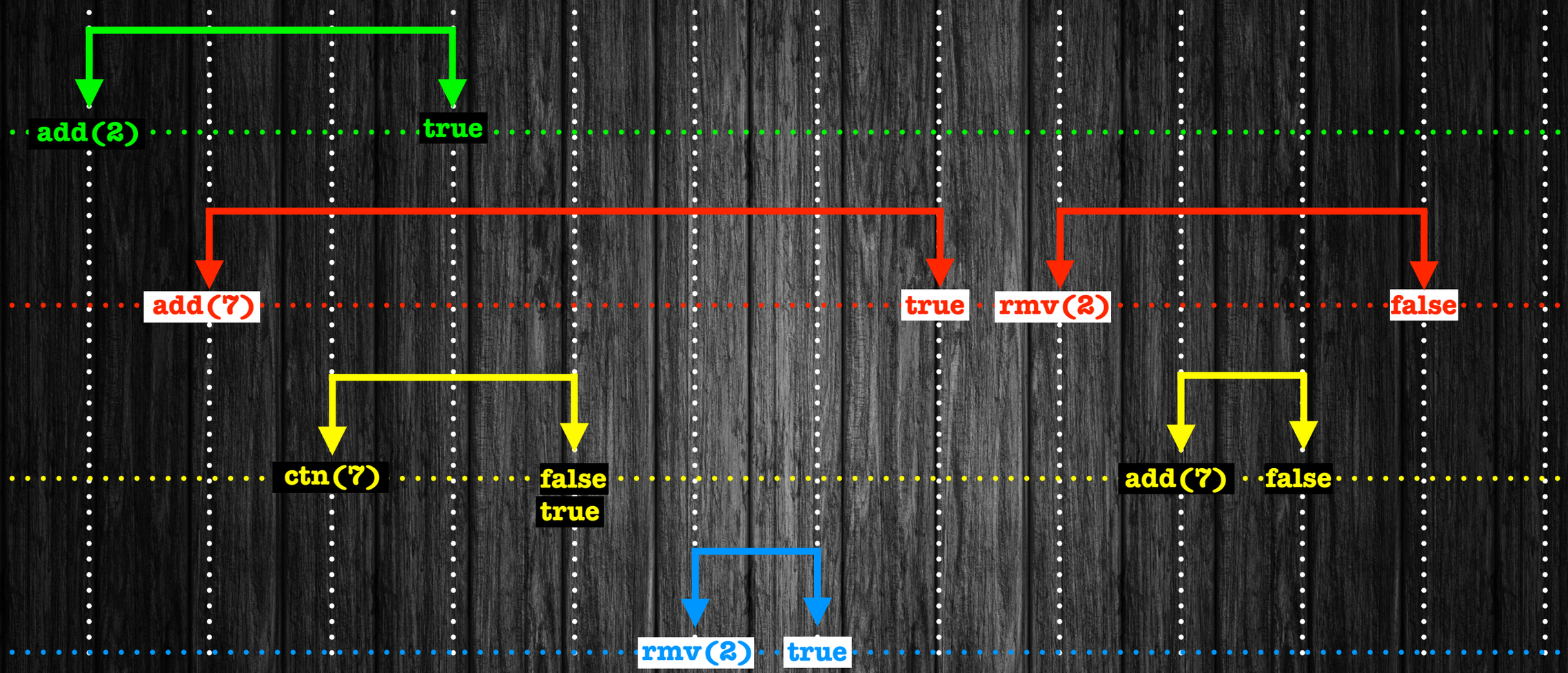


Concurrent Histories | Sets



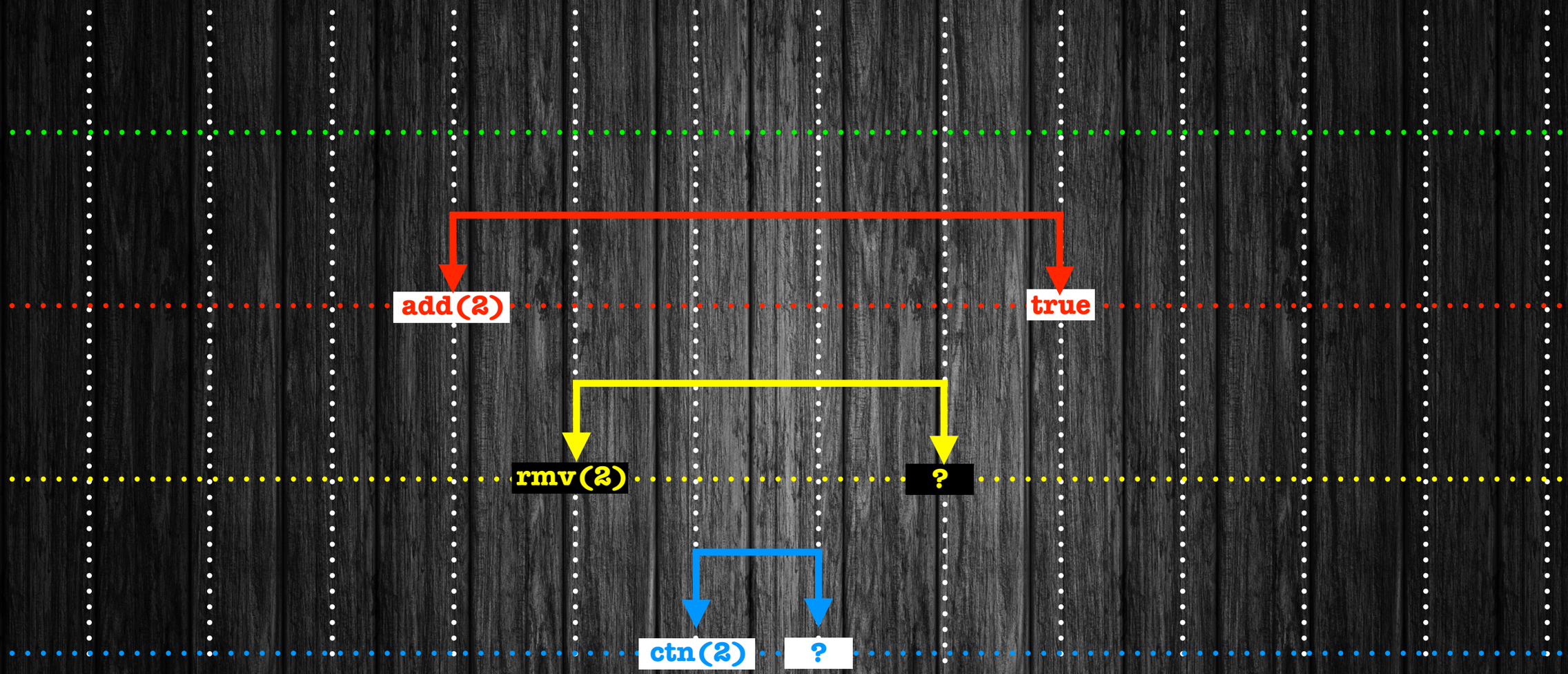
✔

Concurrent Histories | Sets



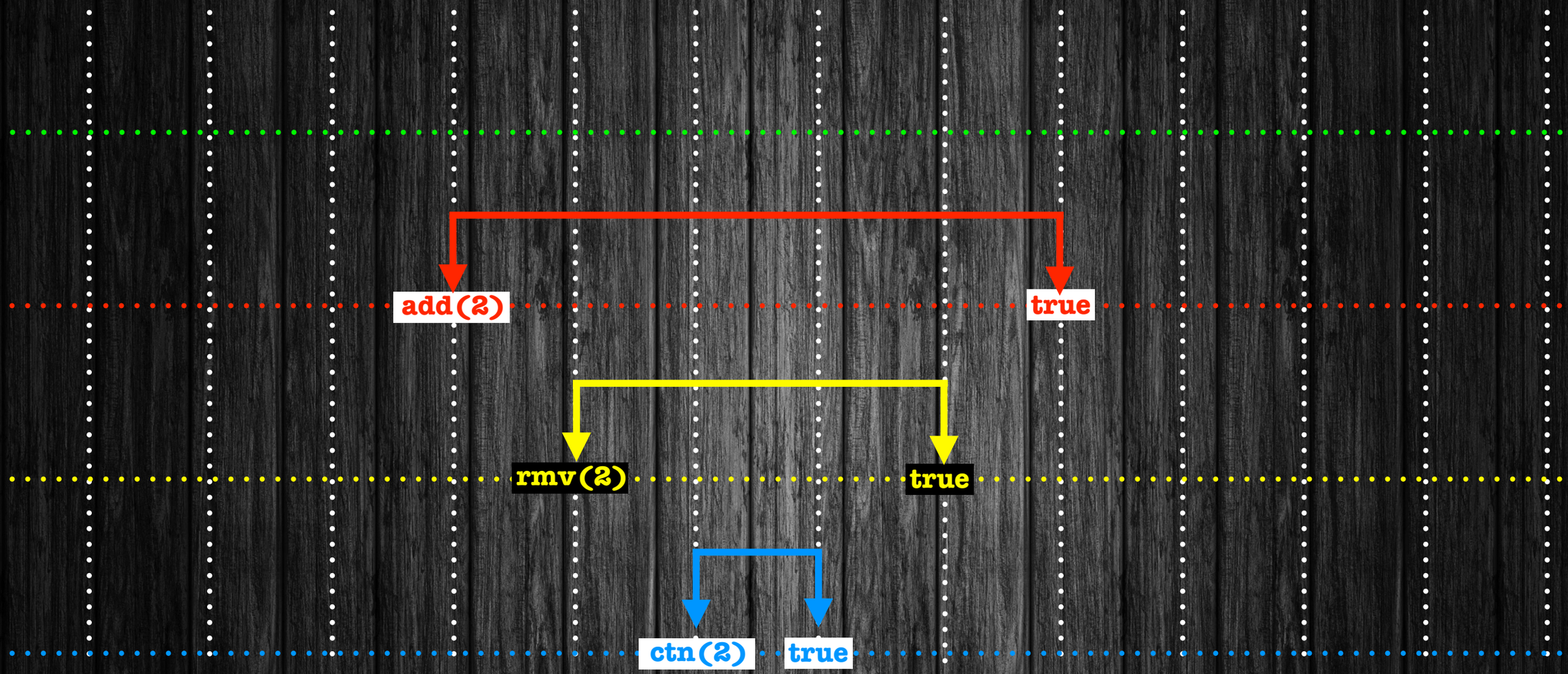
✔

Concurrent Histories | Sets



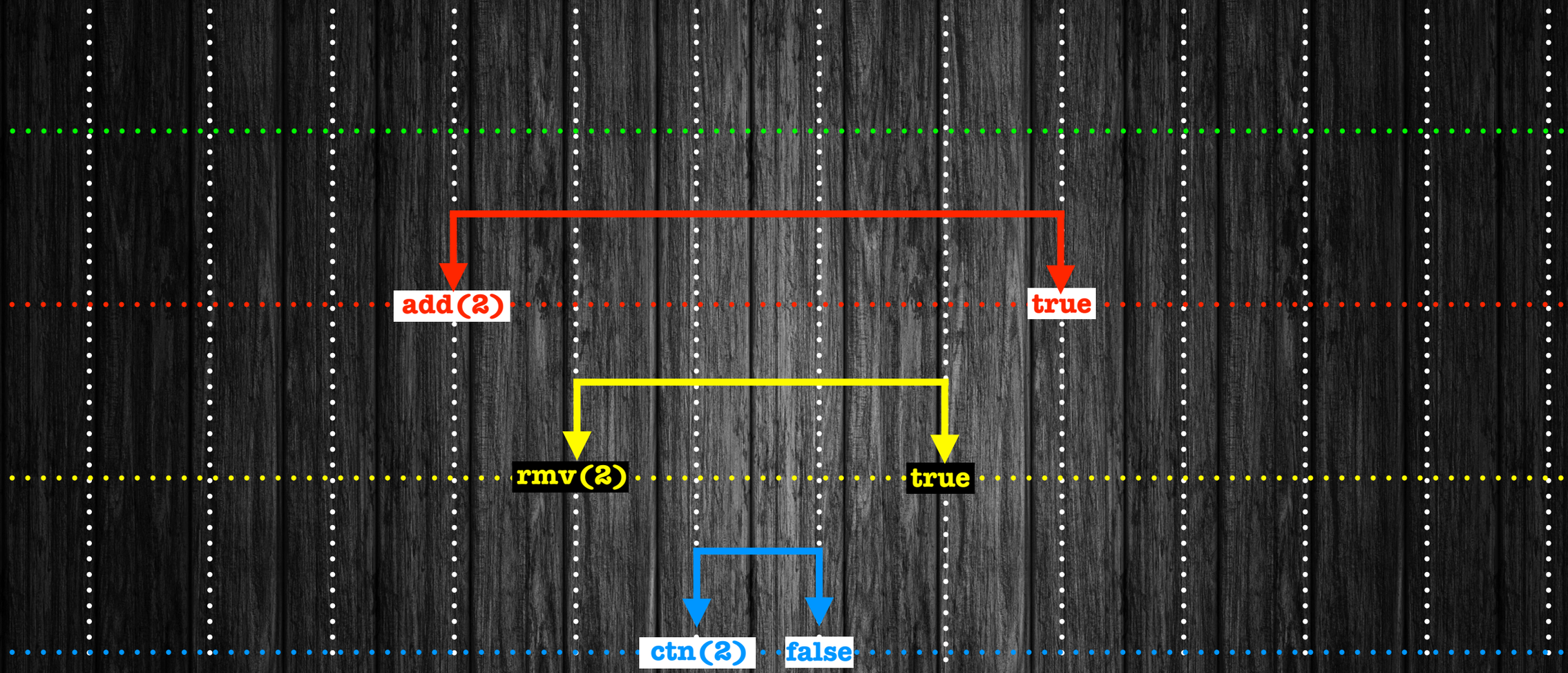
? ?

Concurrent Histories | Sets



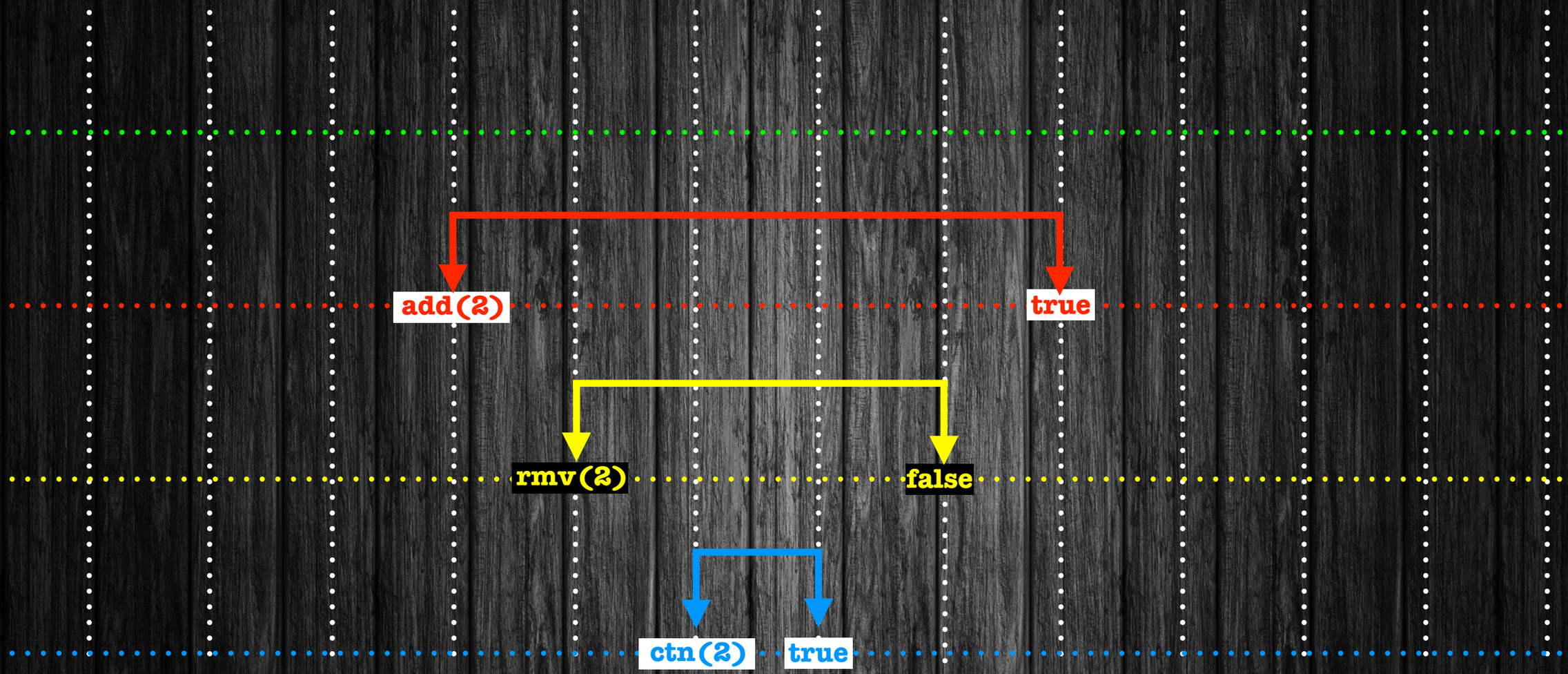
✔

Concurrent Histories Sets



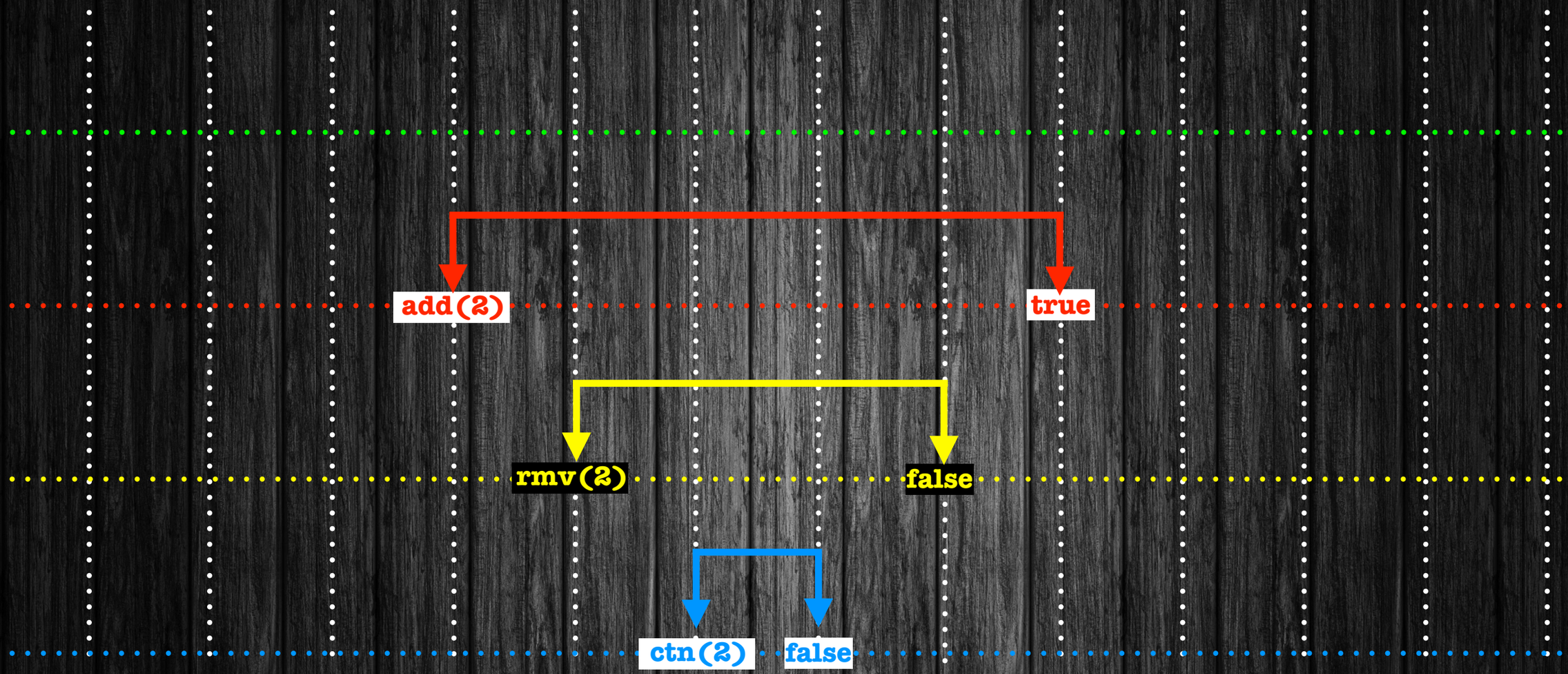
✔

Concurrent Histories | Sets



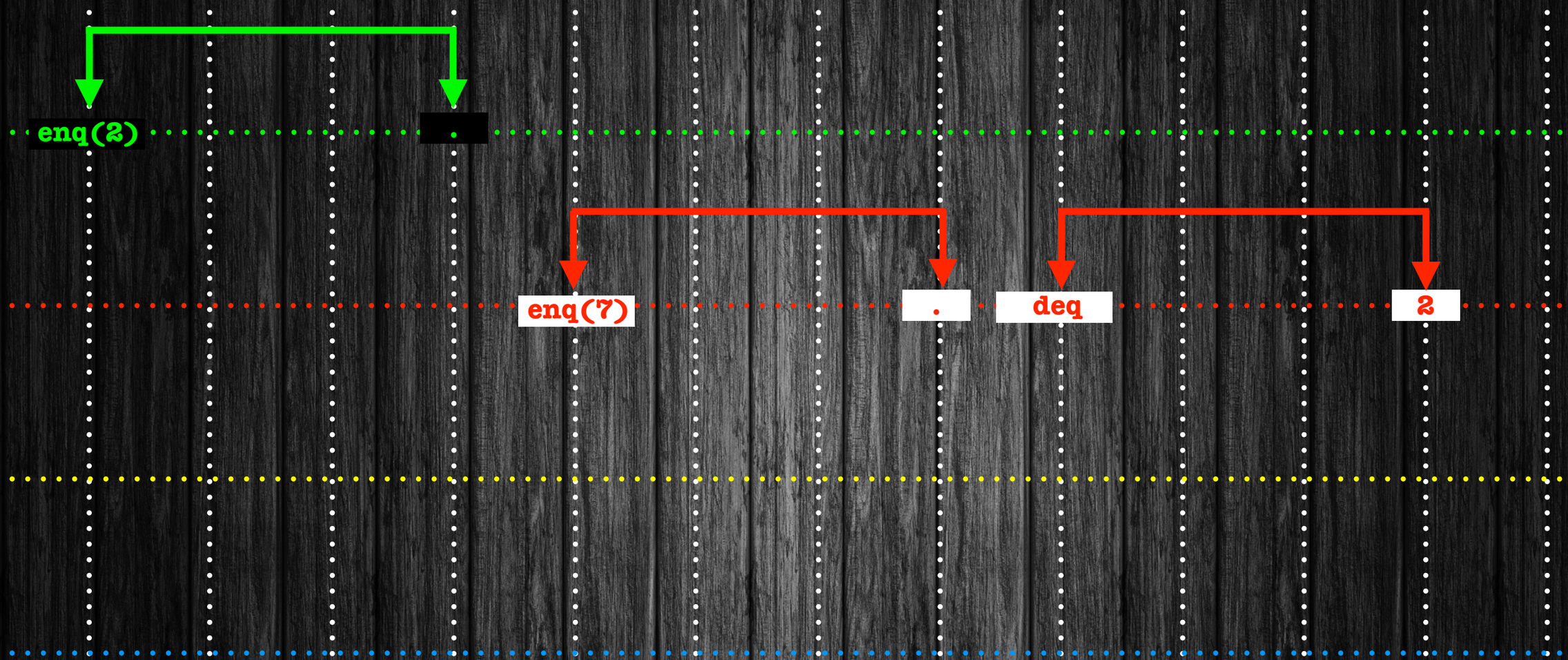
✔

Concurrent Histories | Sets



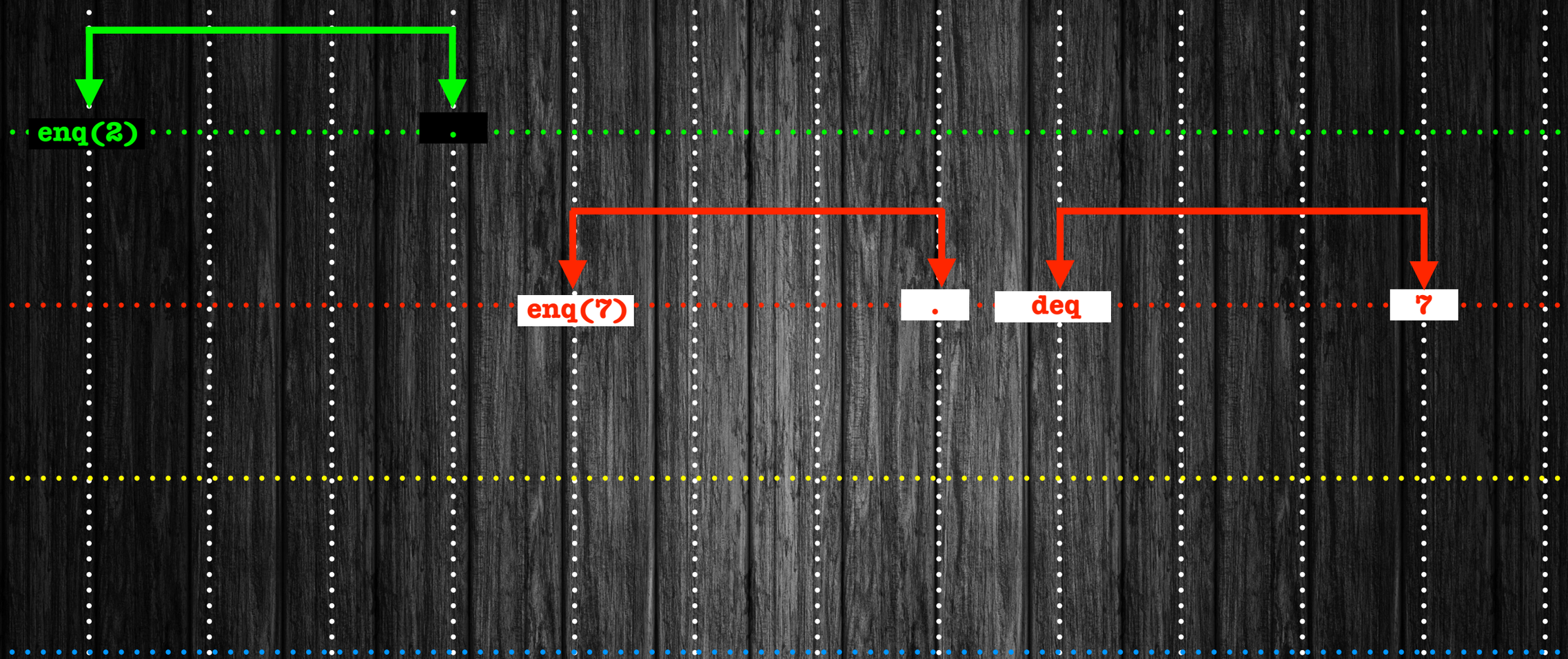
✔

Concurrent Histories | Queues



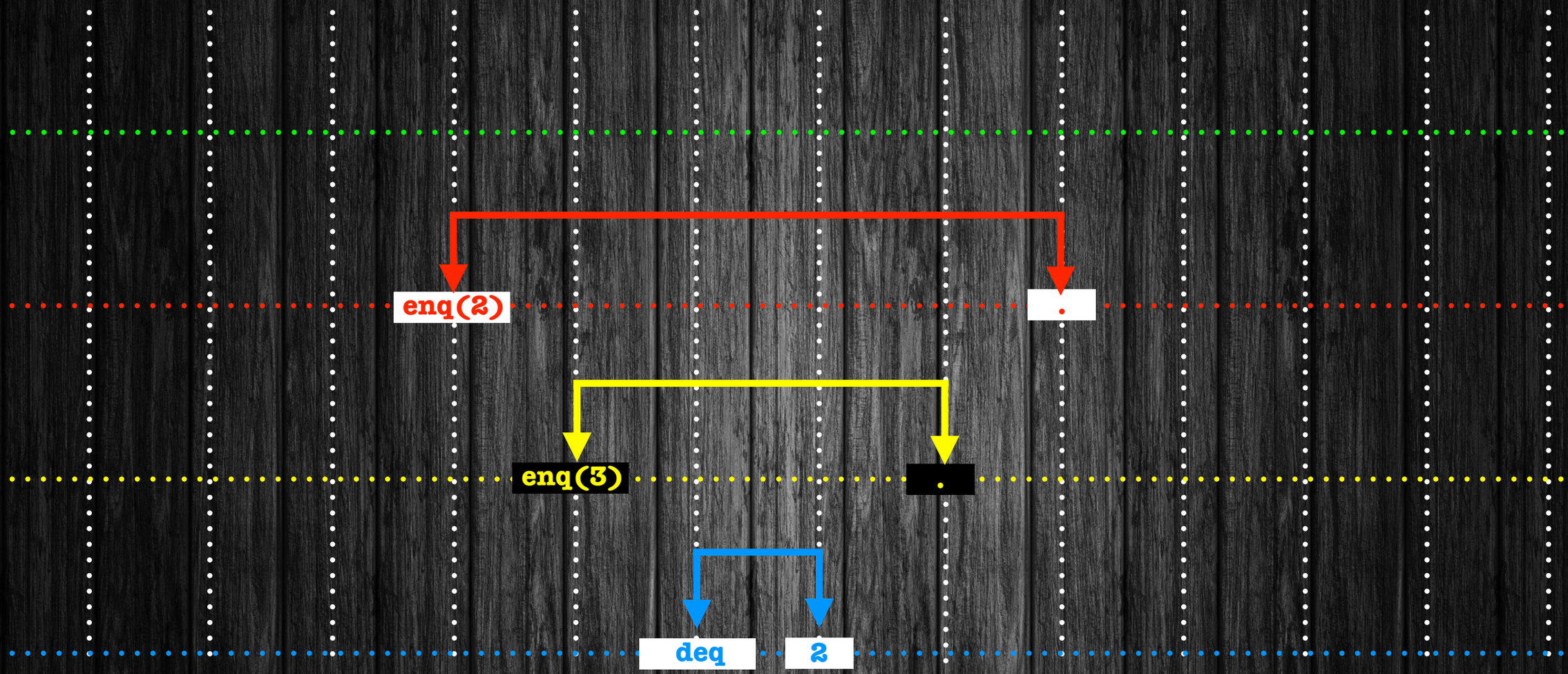
✔

Concurrent Histories | Queues



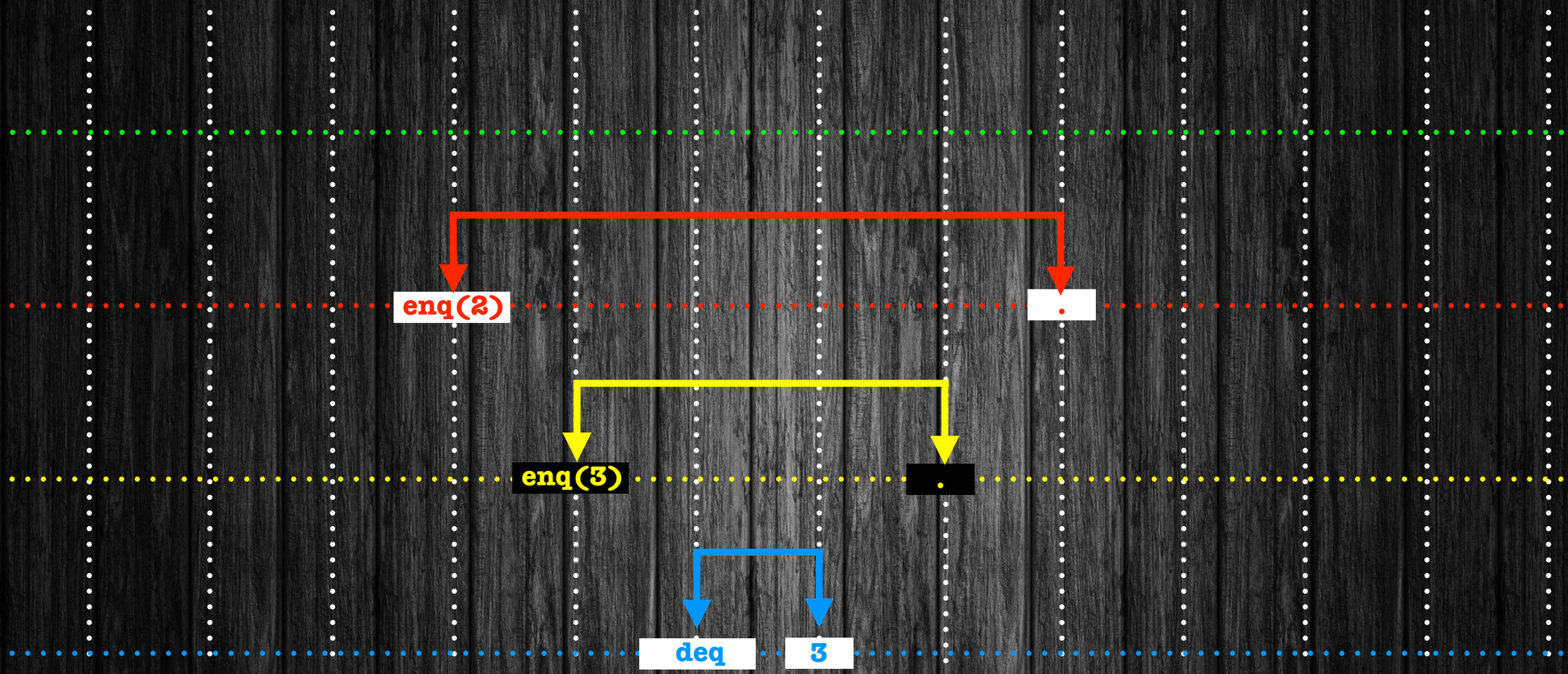
X

Concurrent Histories | Queues



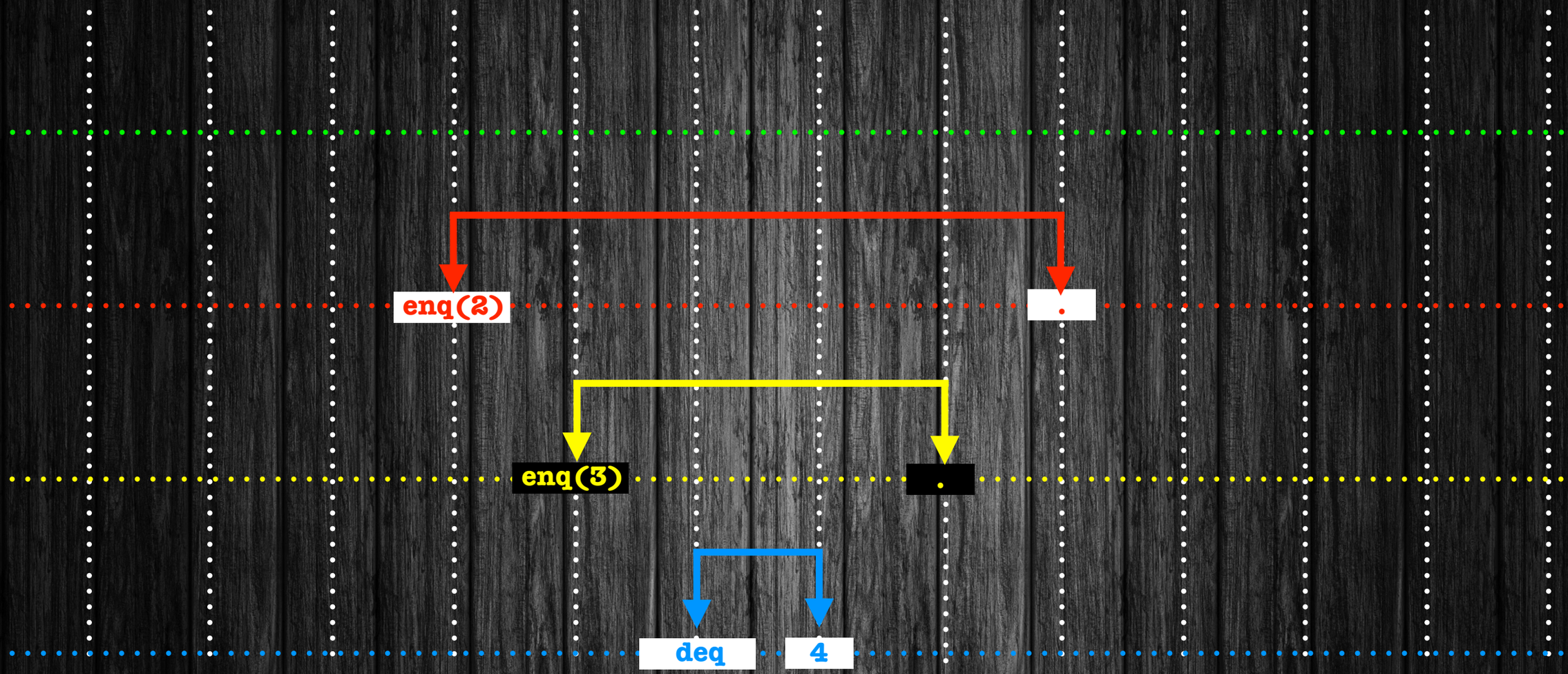
✔

Concurrent Histories | Queues

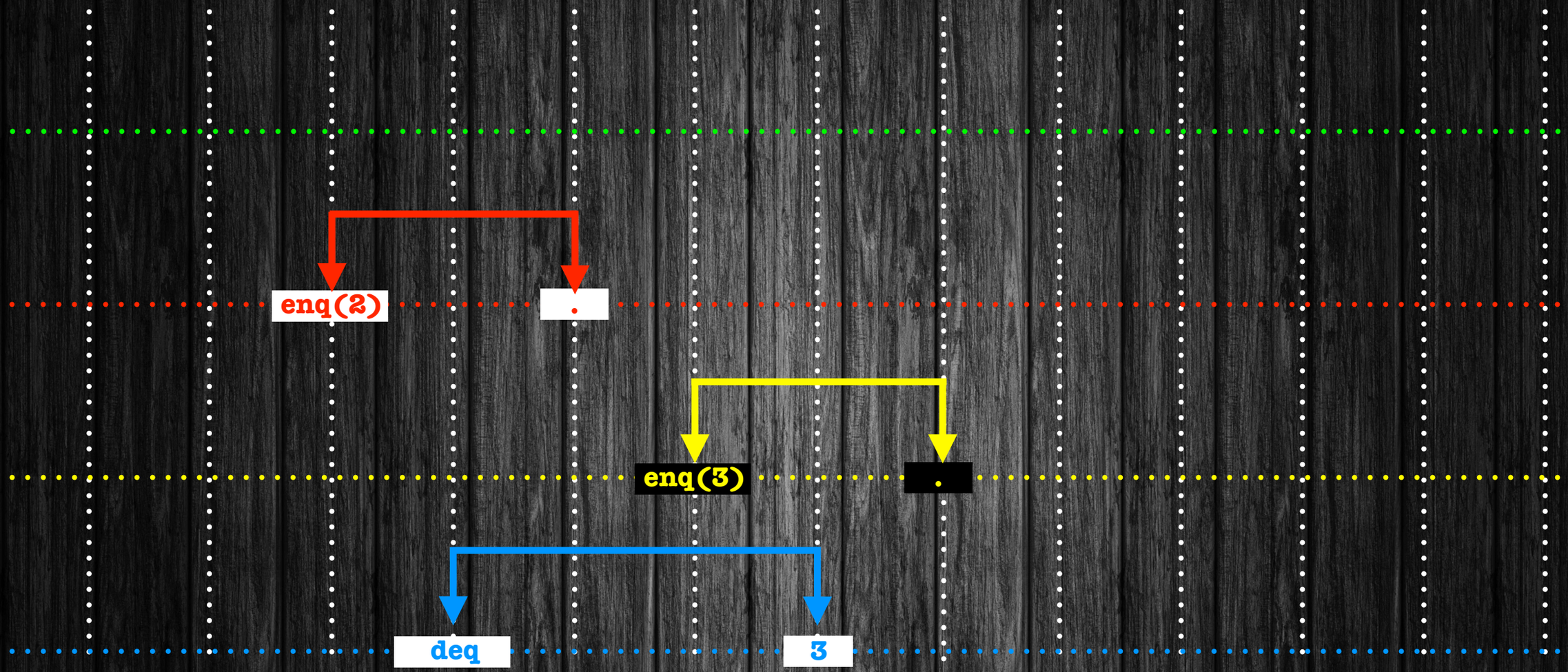


✔

Concurrent Histories | Queues

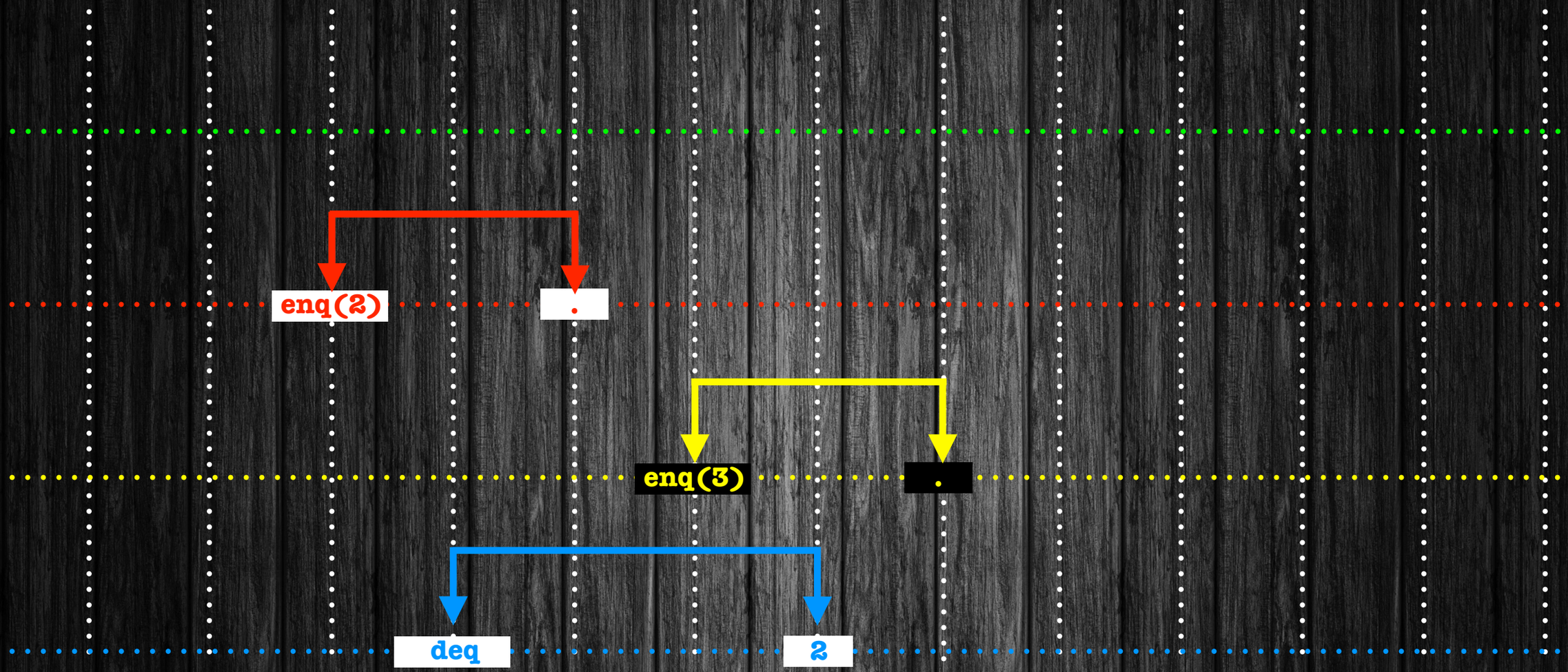


Concurrent Histories | Queues



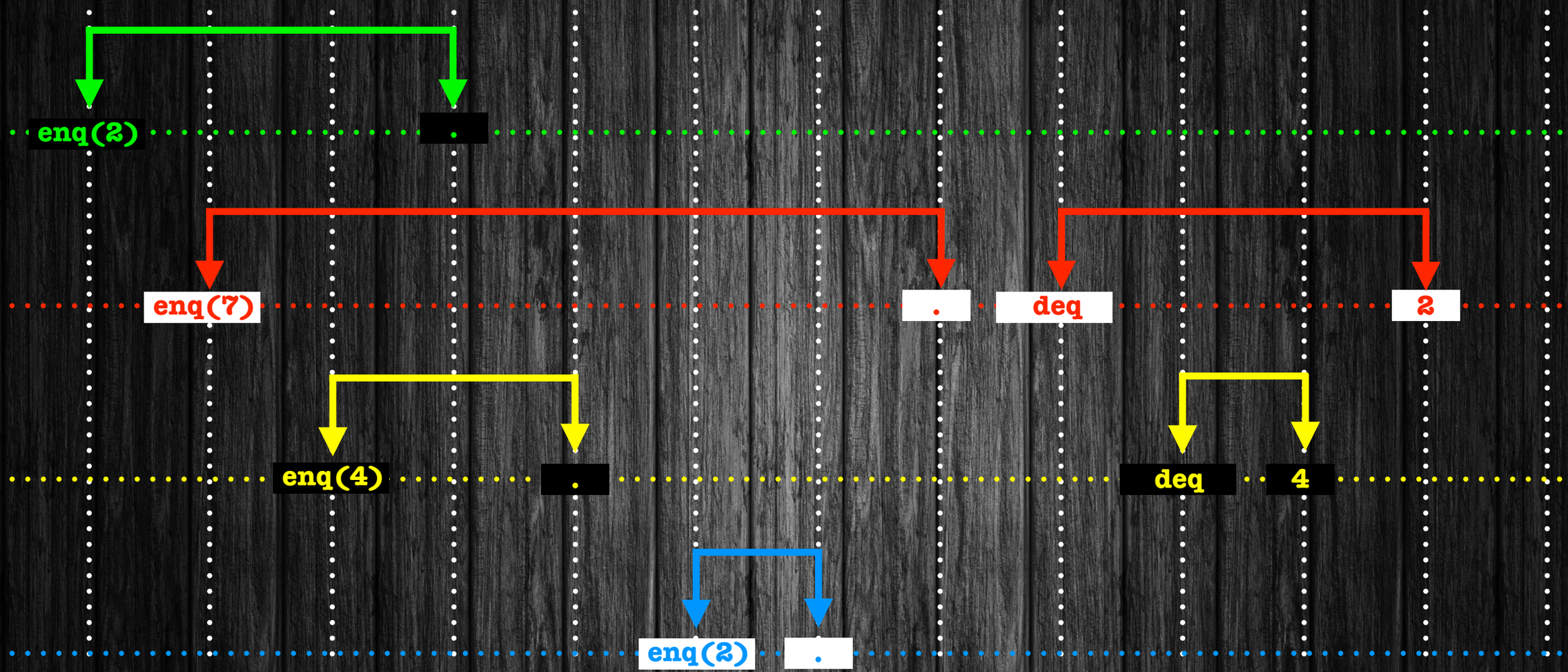


Concurrent Histories | Queues



✔

Concurrent Histories | Queues



✔

Concurrent Programs

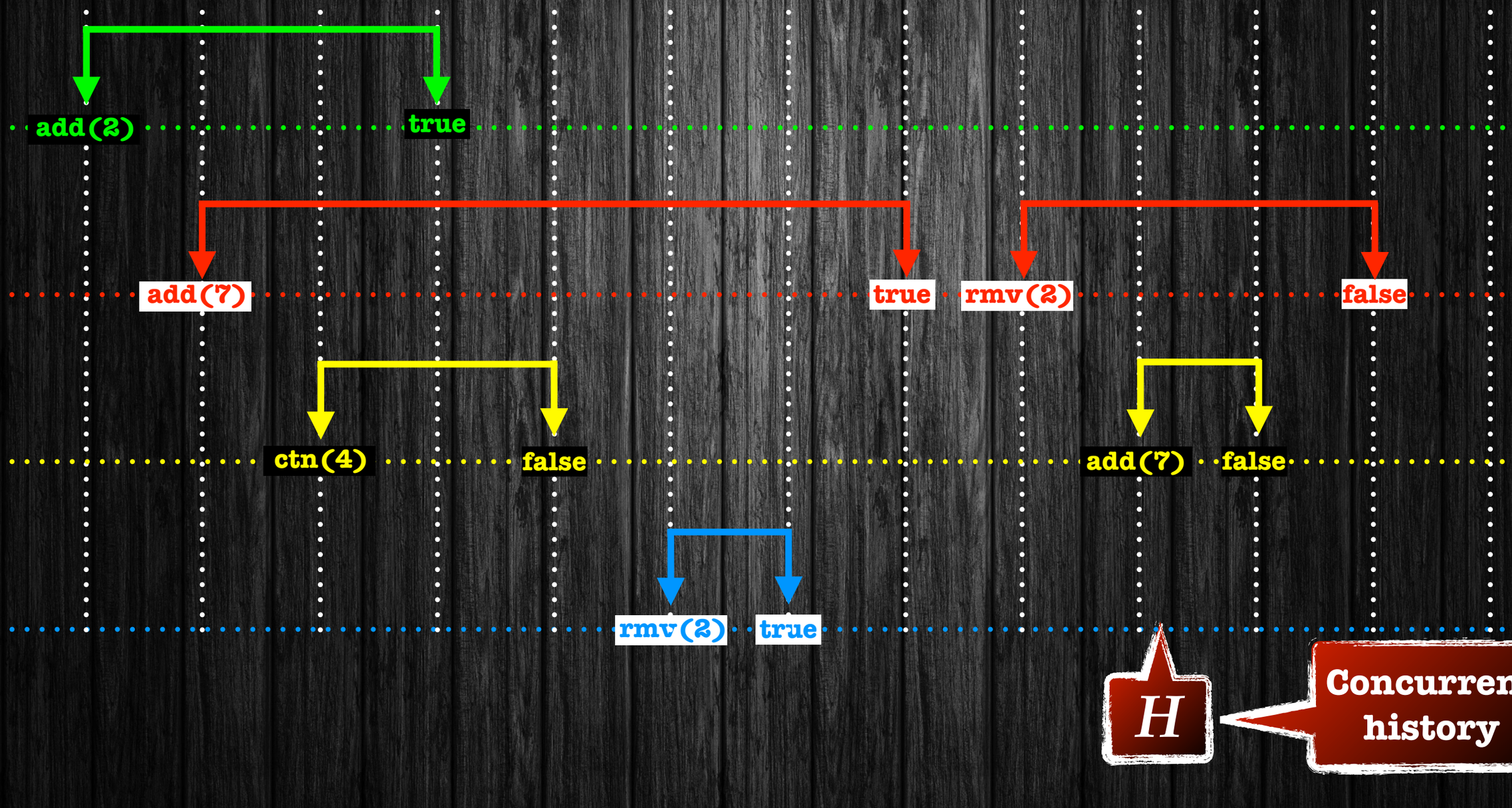
- Concurrent Histories
- Linearizability

Concurrent Programs

- Concurrent Histories
- Linearizability

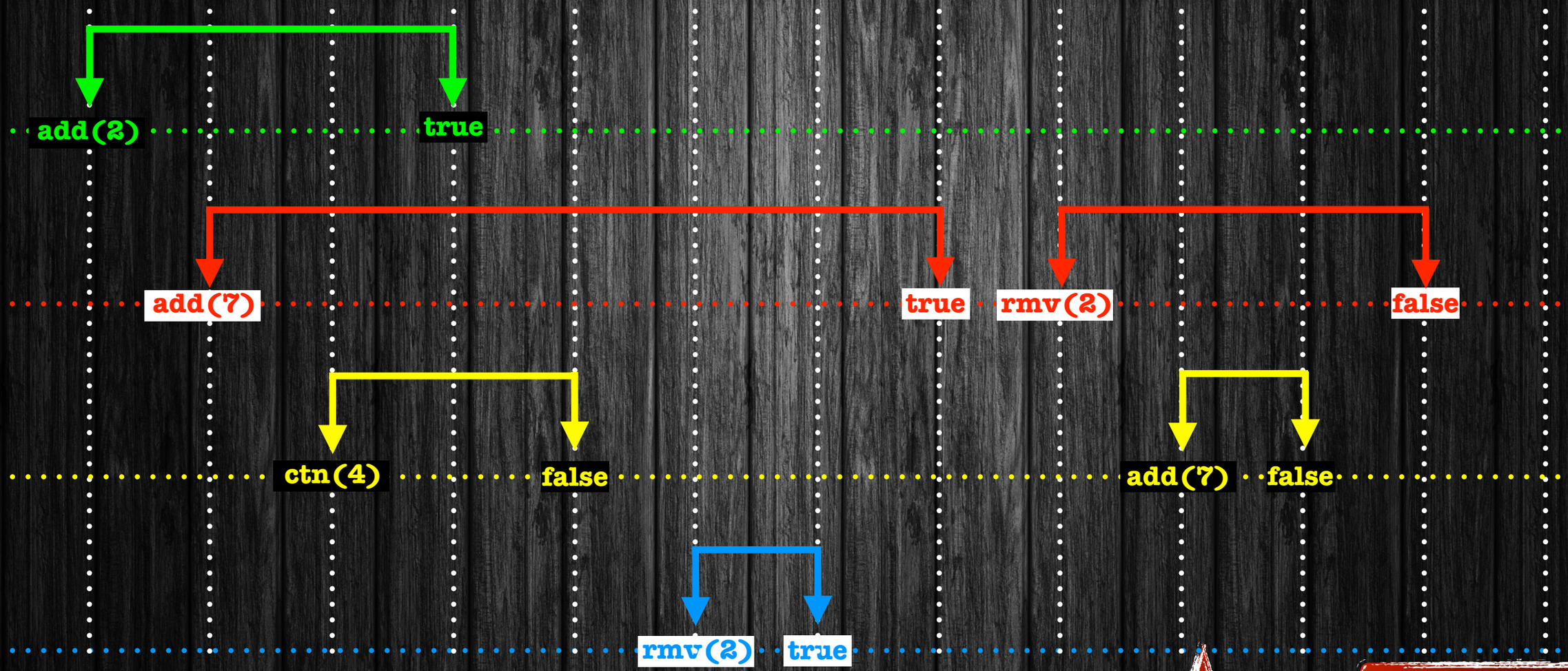
Concurrent Histories | **Linearizability**

Concurrent Histories | Linearizability



Concurrent Histories | Linearizability

ordering



H

Concurrent history

Concurrent Histories | Linearizability

ordering

precedence

add(2)

true

add(7)

true

rmv(2)

false

ctn(4)

false

add(7)

false

rmv(2)

true

H

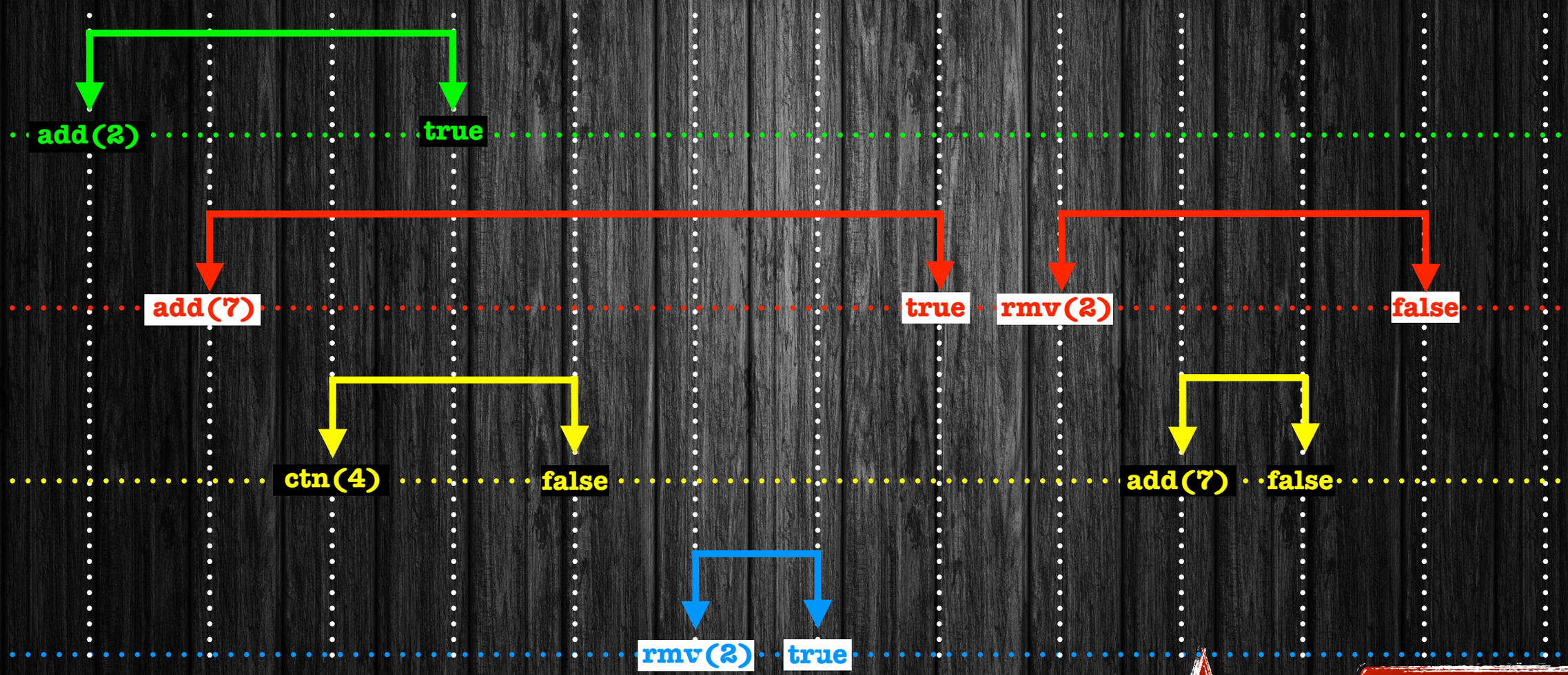
Concurrent history

Concurrent Histories | Linearizability

ordering

precedence

$call_1$ precedes $call_2$



H

Concurrent history

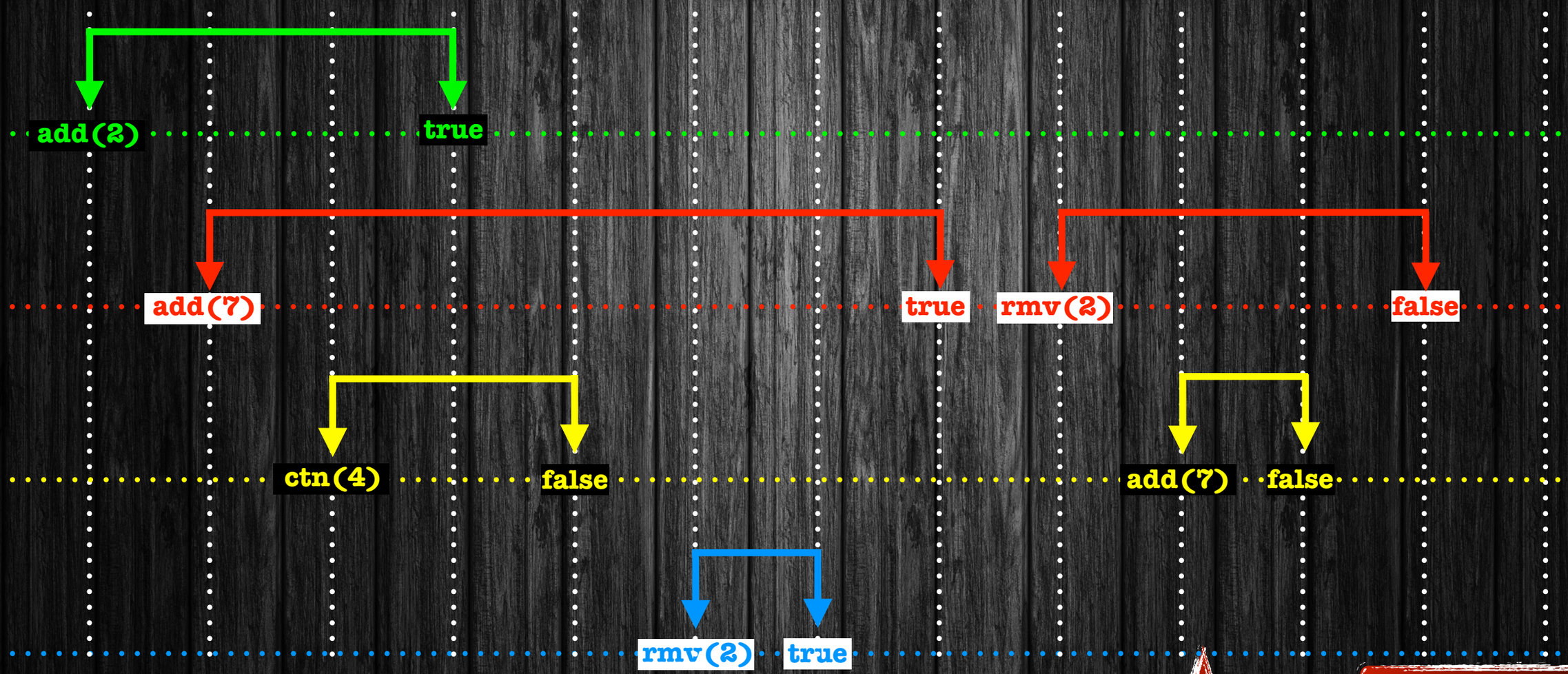
Concurrent Histories | Linearizability

ordering

precedence

$call_1$ precedes $call_2$

$call_1$ ends before $call_2$ starts



H

Concurrent history

Concurrent Histories

Linearizability

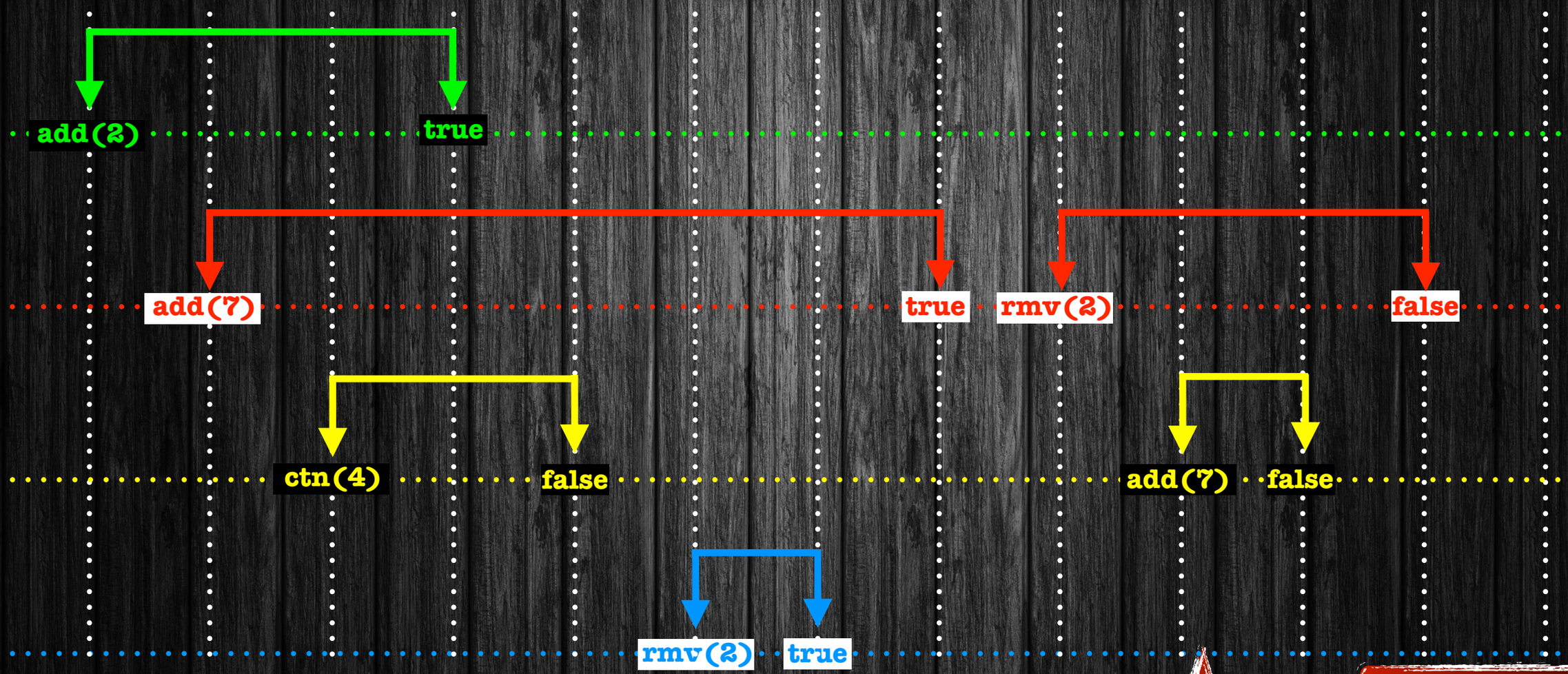
interval of $call_1$ precedes interval of $call_2$

$call_1$ ends before $call_2$ starts

$call_1$ precedes $call_2$

precedence

ordering



H

Concurrent history

Concurrent Histories

Linearizability

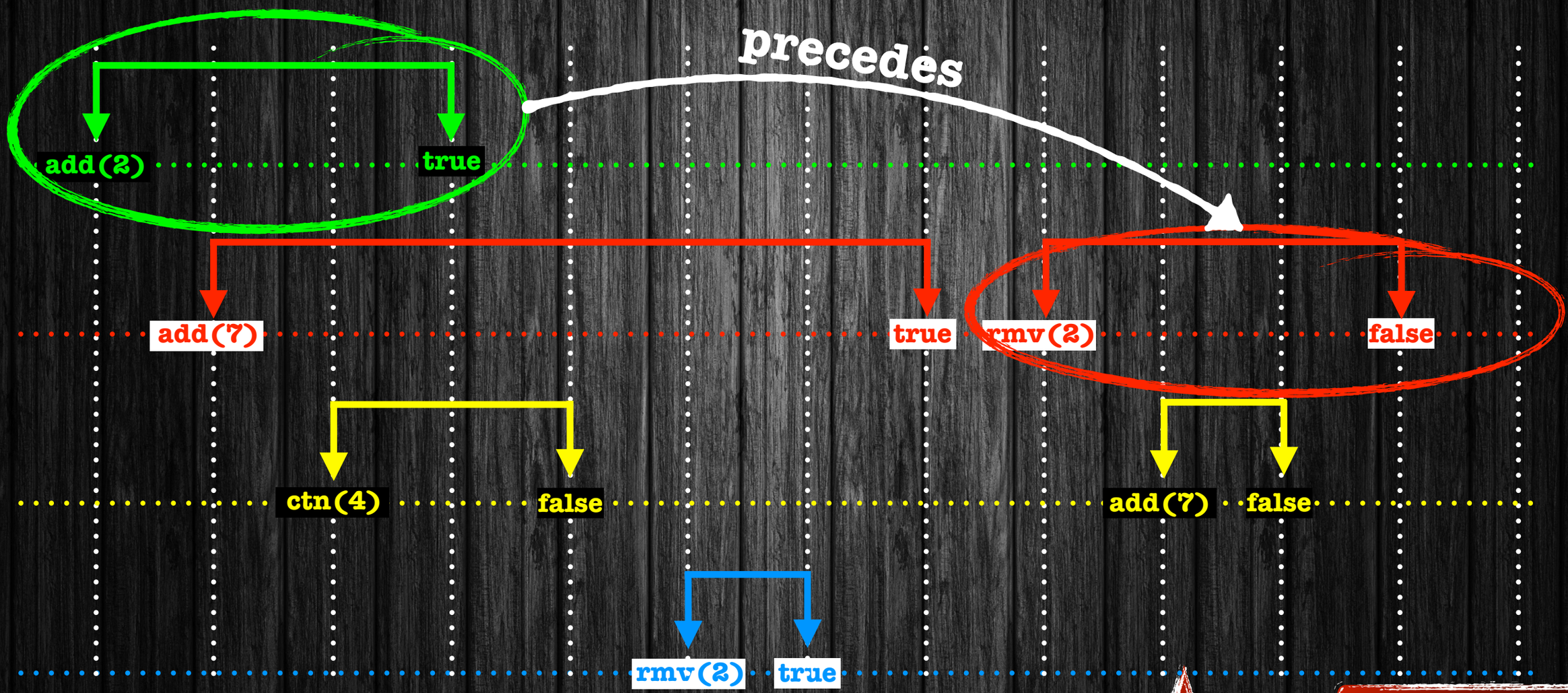
ordering

precedence

$call_1$ precedes $call_2$

$call_1$ ends before $call_2$ starts

interval of $call_1$ precedes interval of $call_2$



H

Concurrent history

Concurrent Histories

Linearizability

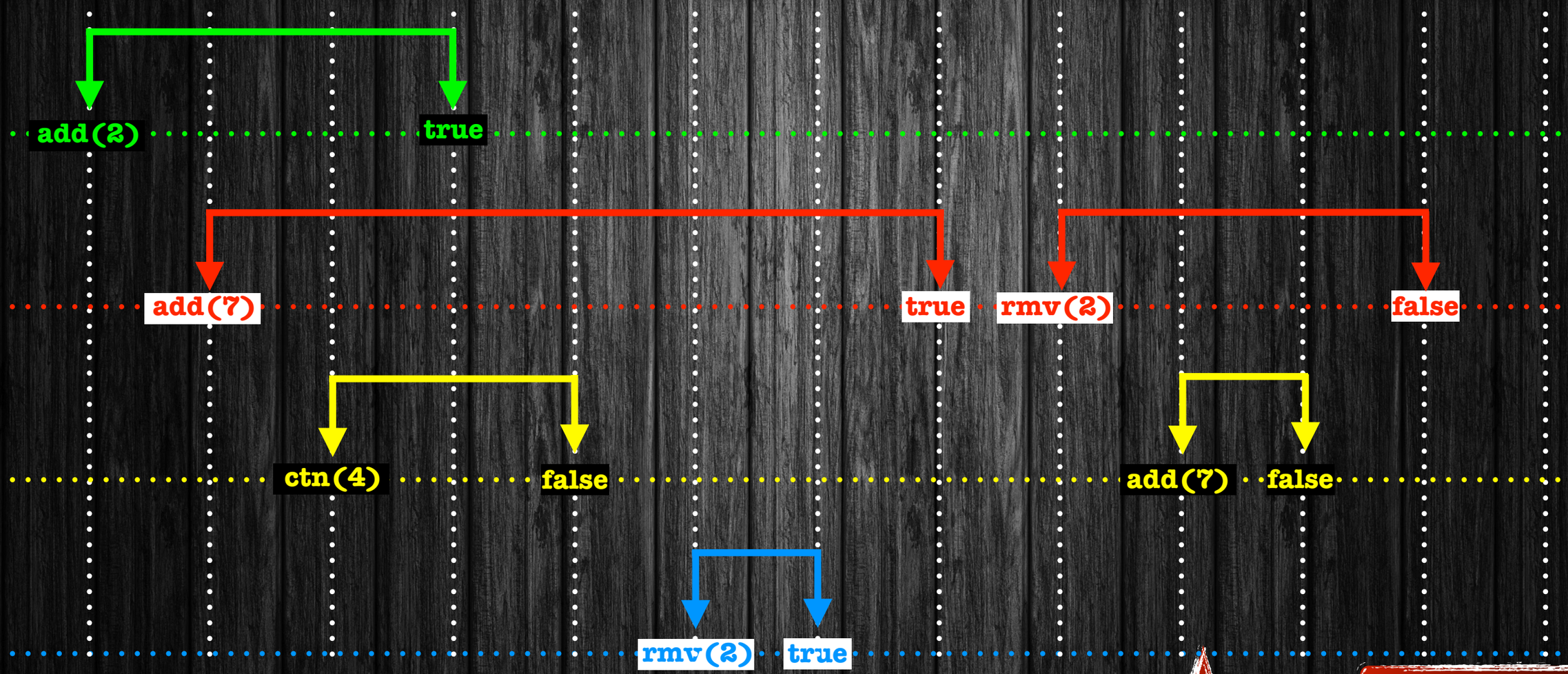
interval of $call_1$ precedes interval of $call_2$

$call_1$ ends before $call_2$ starts

$call_1$ precedes $call_2$

precedence

ordering



H

Concurrent history

Concurrent Histories

Linearizability

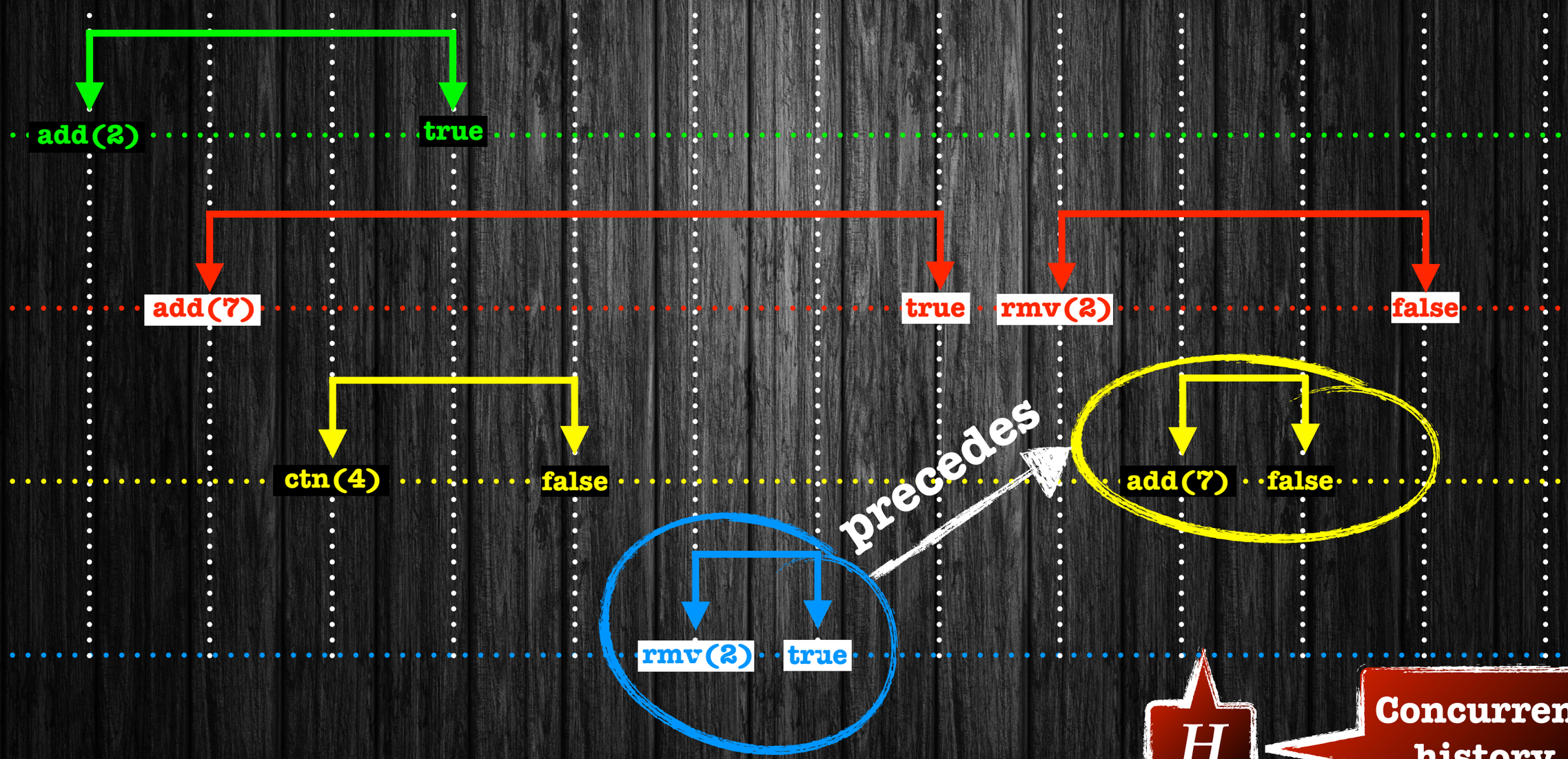
interval of $call_1$ precedes interval of $call_2$

$call_1$ ends before $call_2$ starts

$call_1$ precedes $call_2$

precedence

ordering



Concurrent Histories

Linearizability

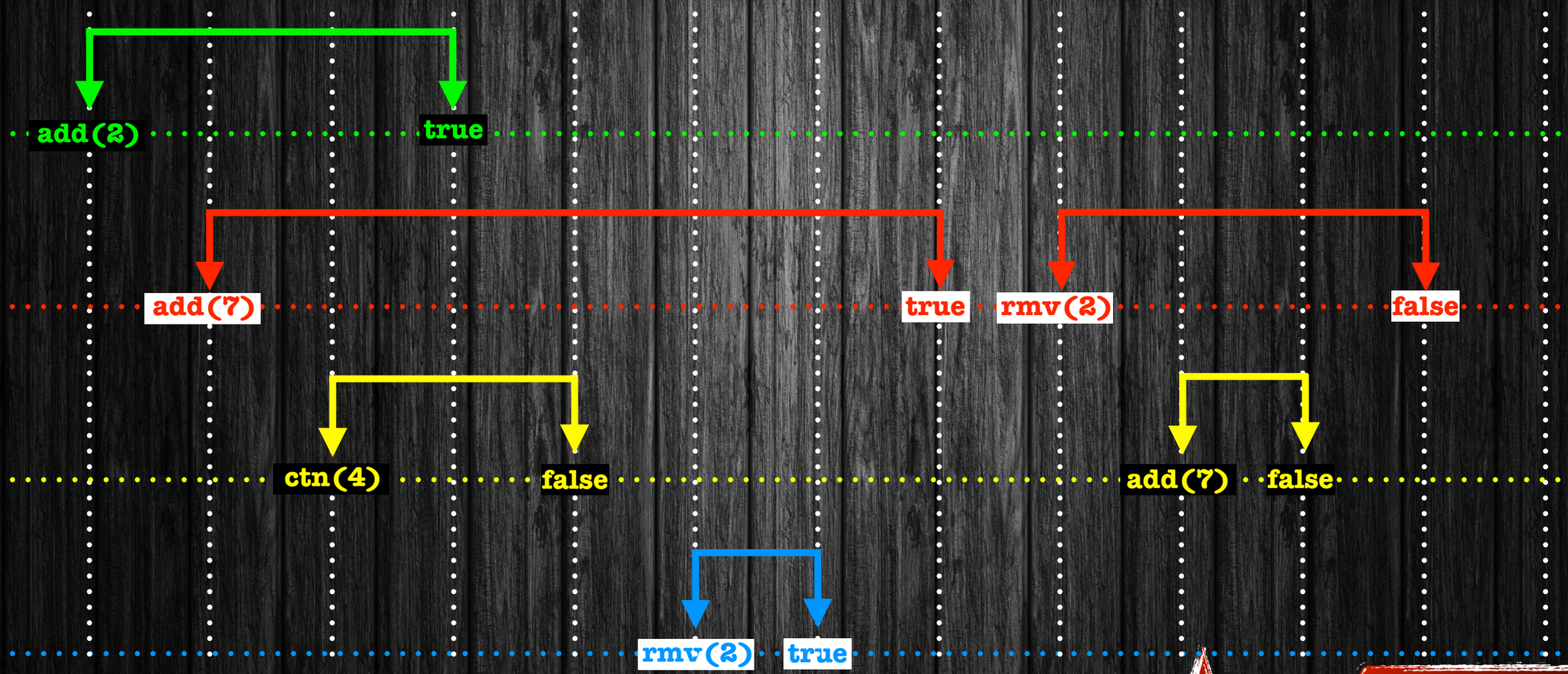
interval of $call_1$ precedes interval of $call_2$

$call_1$ ends before $call_2$ starts

$call_1$ precedes $call_2$

precedence

ordering



H

Concurrent history

Concurrent Histories

Linearizability

interval of $call_1$ precedes interval of $call_2$

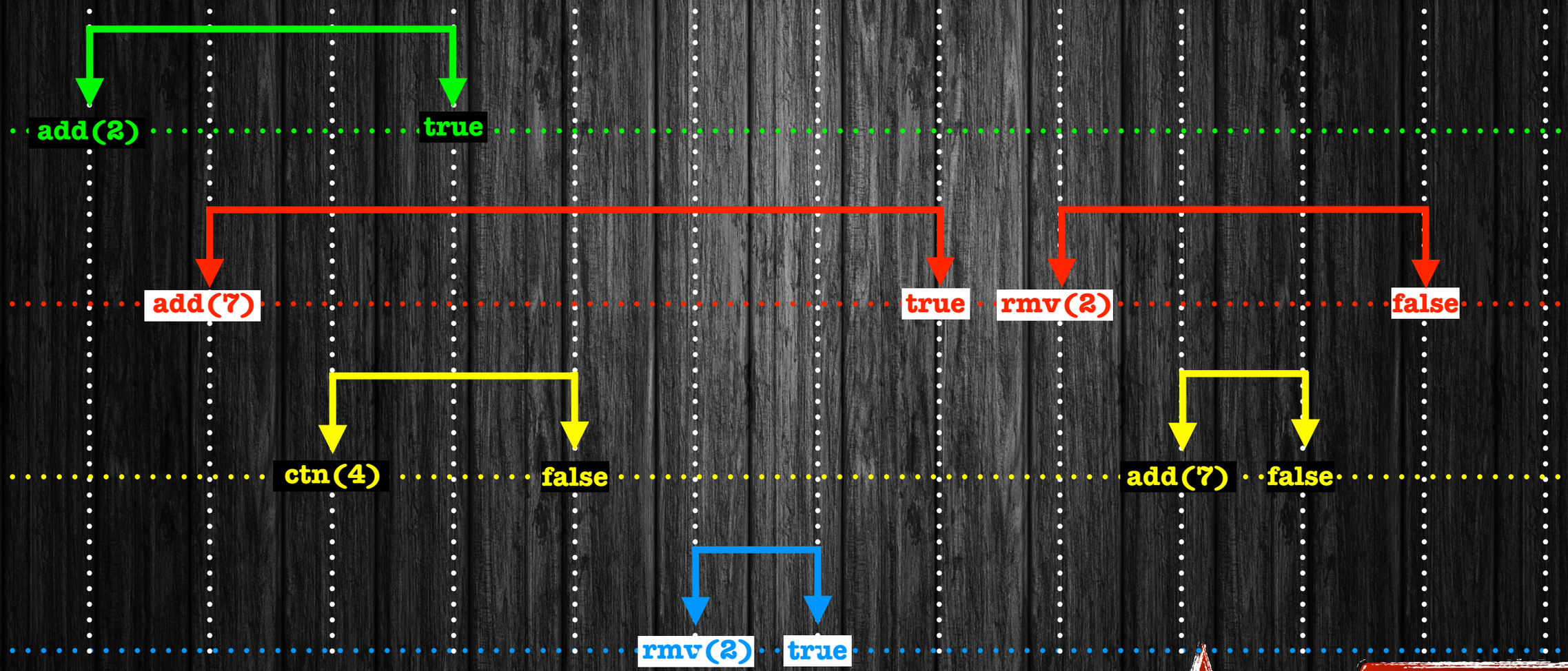
$call_1$ ends before $call_2$ starts

ordering

precedence

$call_1$ precedes $call_2$

overlapping



H

Concurrent history

Concurrent Histories

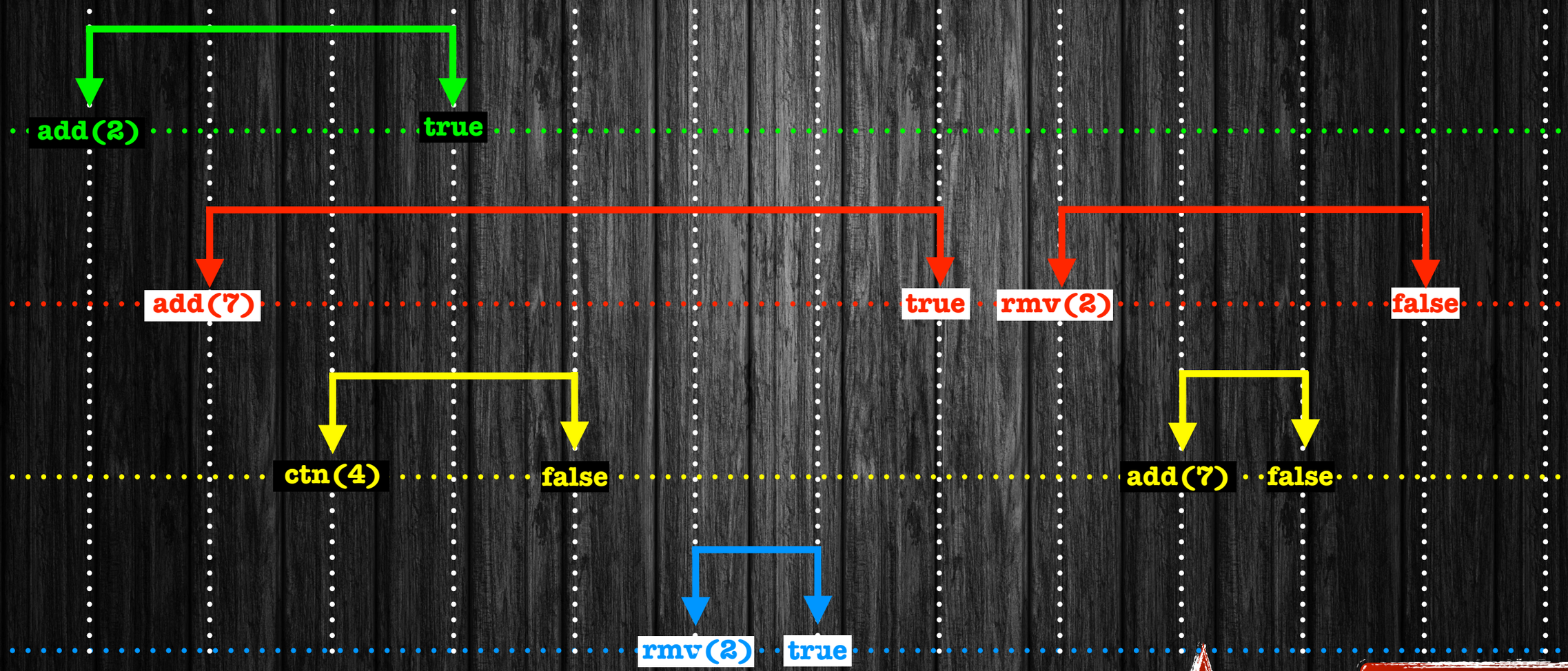
Linearizability

interval of $call_1$ precedes interval of $call_2$

$call_1$ ends before $call_2$ starts

ordering

- precedence → $call_1$ precedes $call_2$
- overlapping → $call_1$ and $call_2$ overlap



H — Concurrent history

Concurrent Histories

Linearizability

ordering

precedence

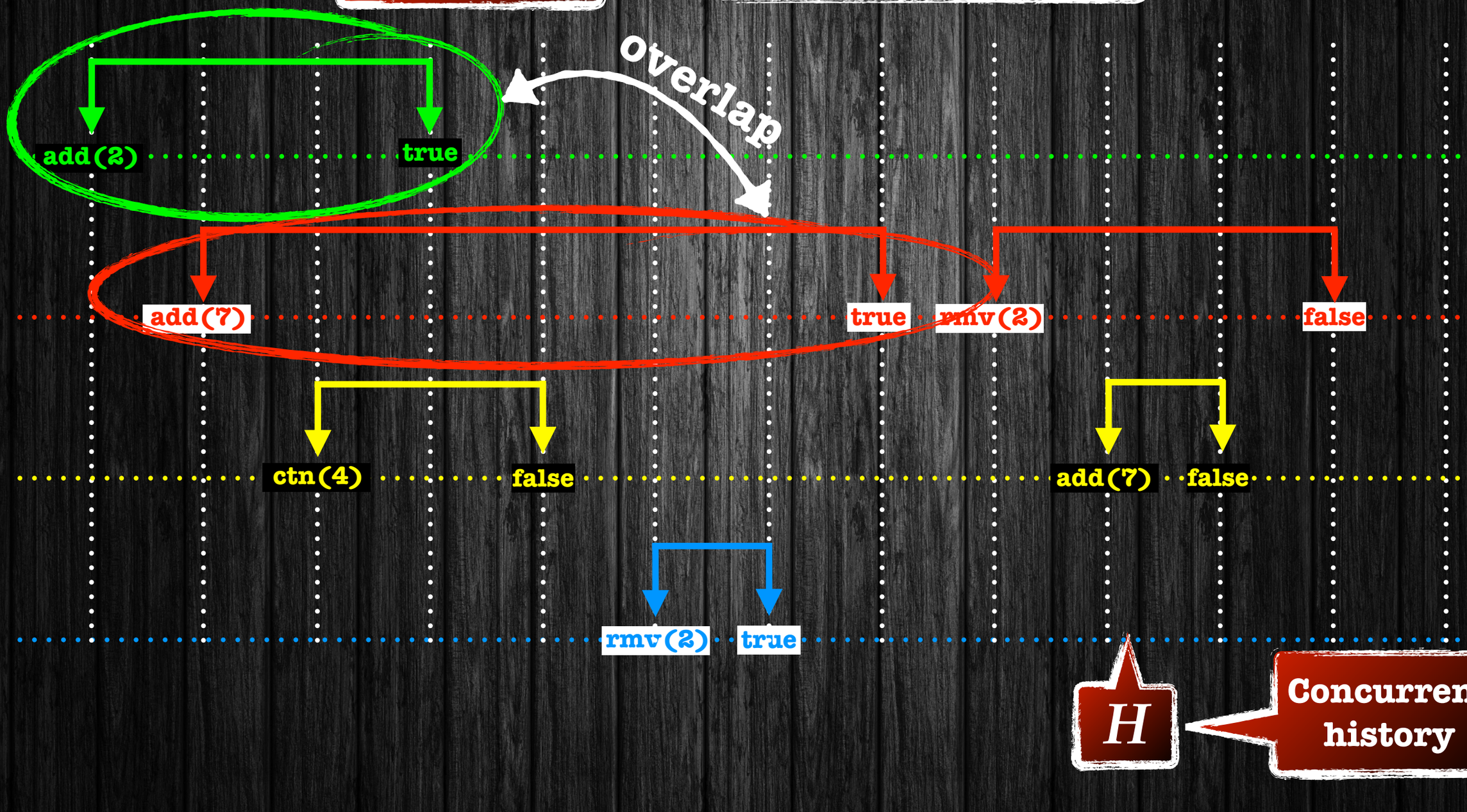
$call_1$ precedes $call_2$

$call_1$ ends before $call_2$ starts

overlapping

$call_1$ and $call_2$ overlap

interval of $call_1$ precedes interval of $call_2$



H

Concurrent history

Concurrent Histories

Linearizability

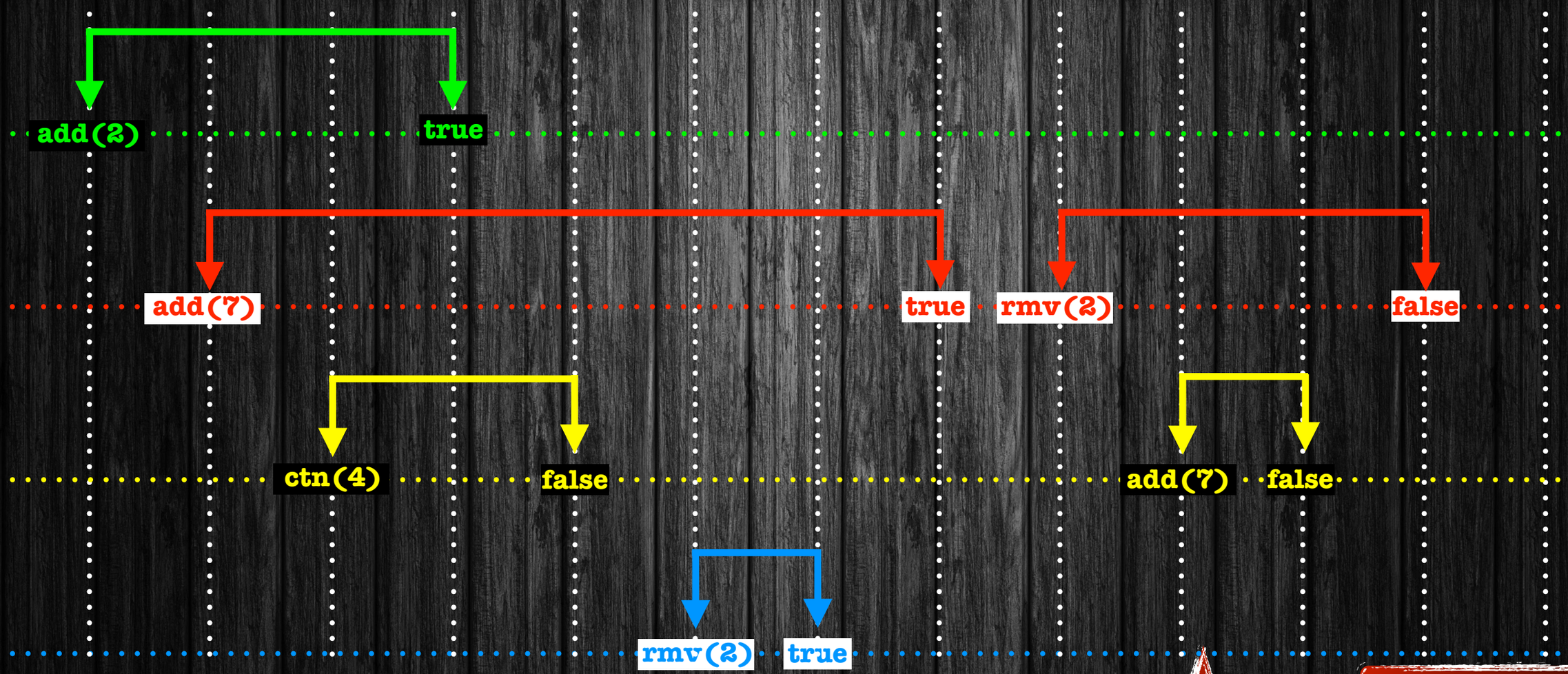
interval of $call_1$ precedes interval of $call_2$

$call_1$ ends before $call_2$ starts

$call_1$ precedes $call_2$

precedence

ordering



H

Concurrent history

Concurrent Histories

Linearizability

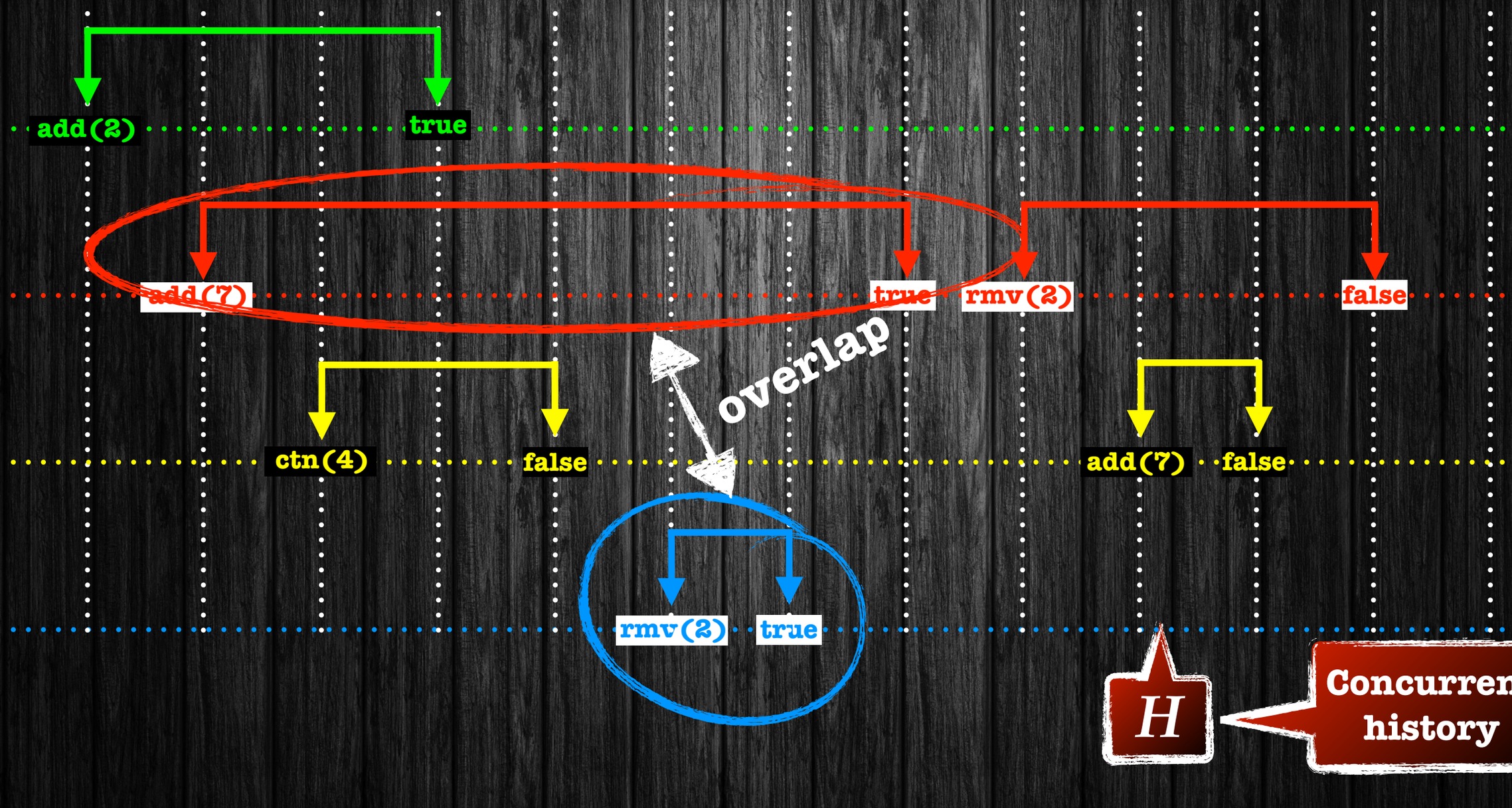
ordering

precedence

$call_1$ precedes $call_2$

$call_1$ ends before $call_2$ starts

interval of $call_1$ precedes interval of $call_2$

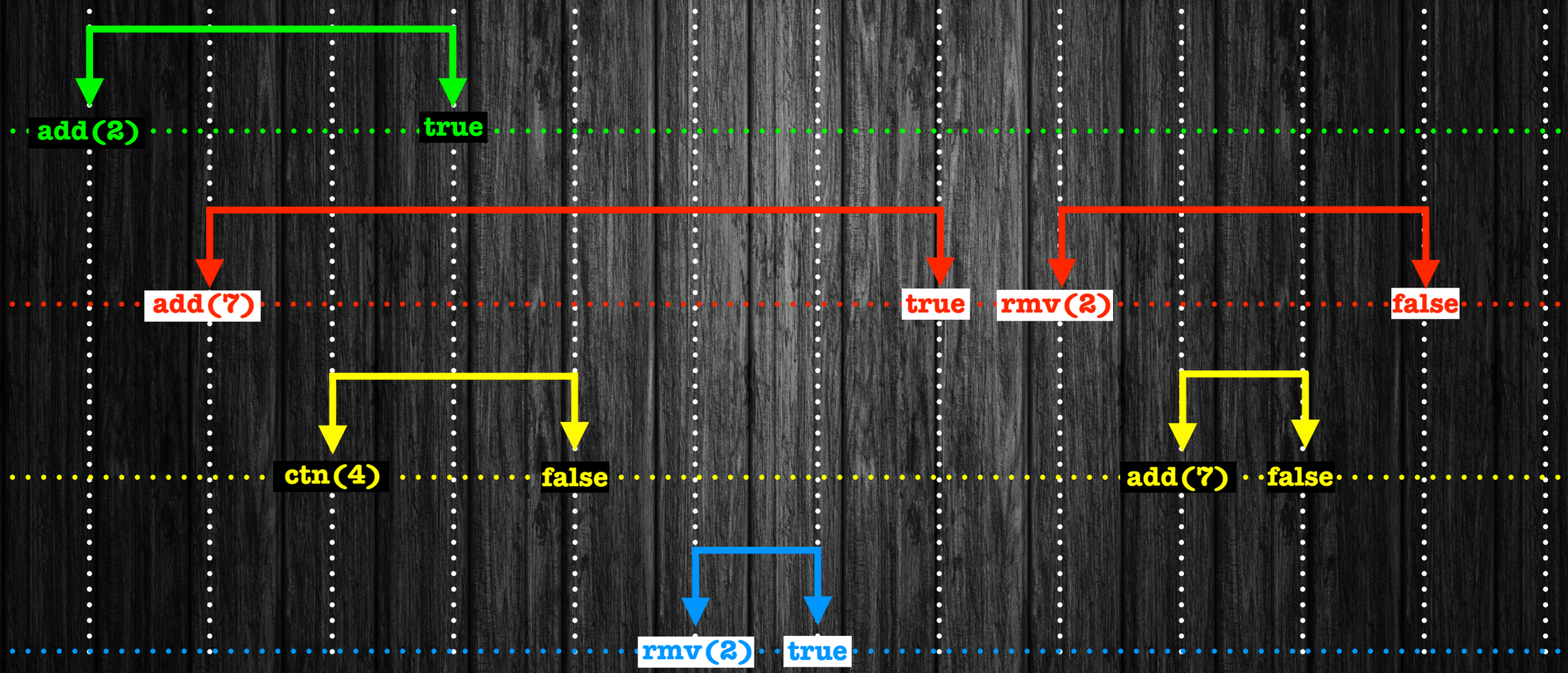


H

Concurrent history

Concurrent Histories | Linearizability

H — Concurrent history



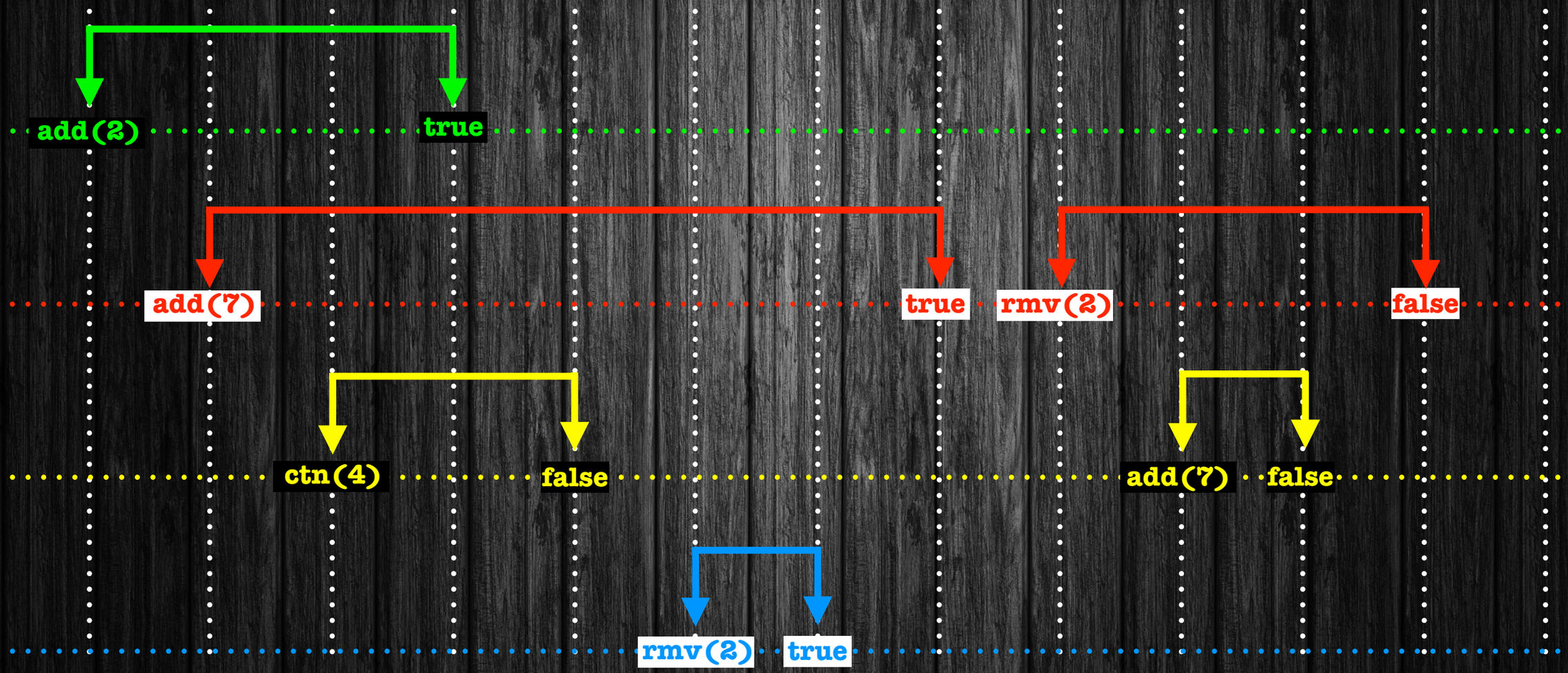
Concurrent Histories

Linearizability

Linearization Points

H

Concurrent history

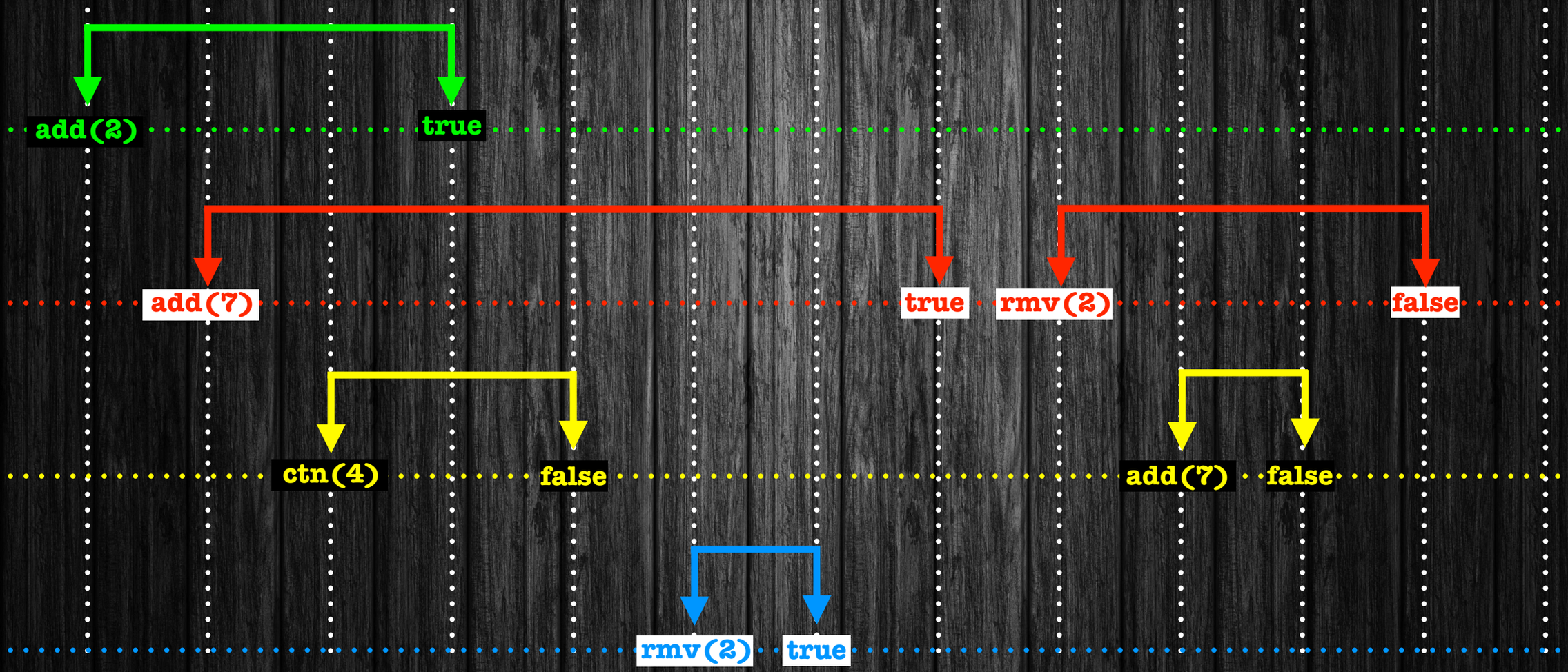


Concurrent Histories

Linearizability

Linearization Points — a point inside each call

H — Concurrent history



Concurrent Histories

Linearizability

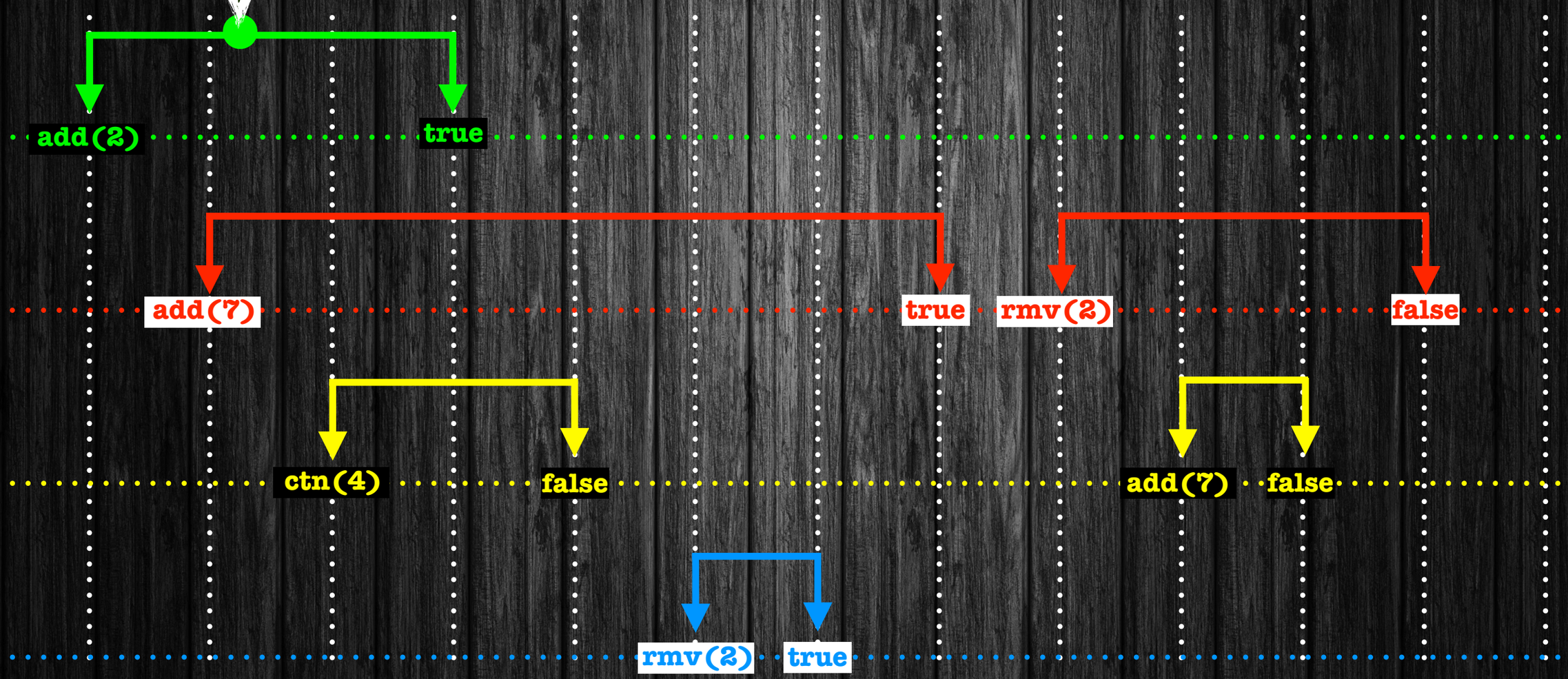
Linearization Points

a point inside each call

linearization point

H

Concurrent history

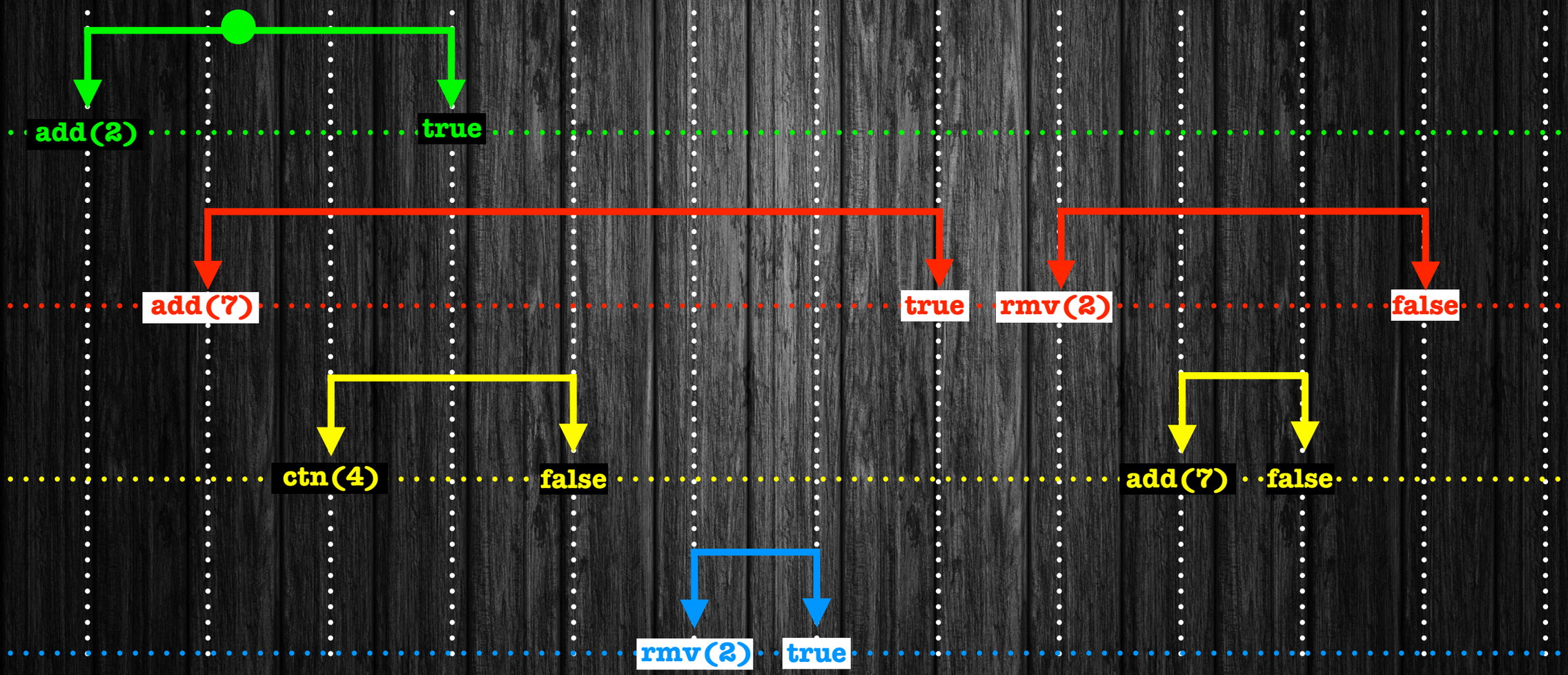


Concurrent Histories

Linearizability

Linearization Points — a point inside each call

H — Concurrent history



Concurrent Histories

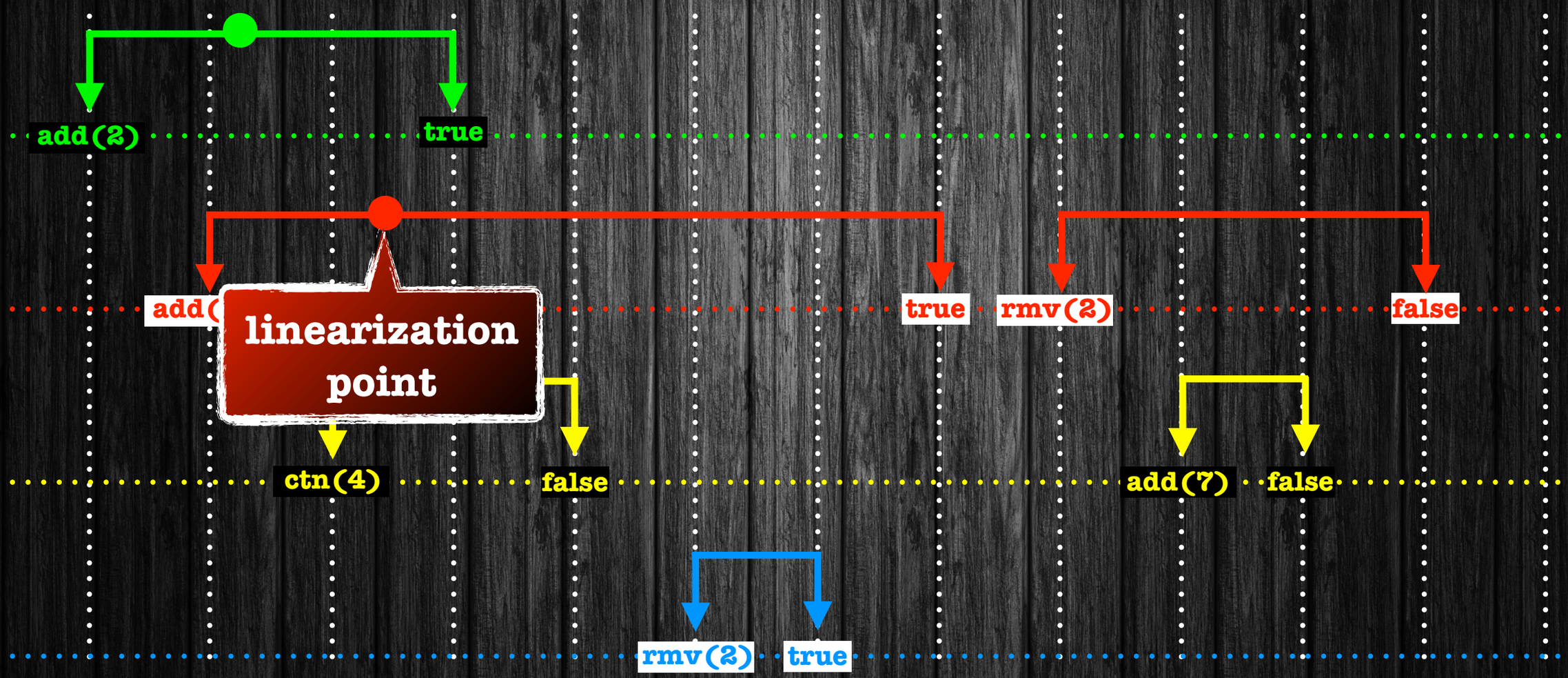
Linearizability

Linearization Points

a point inside each call

H

Concurrent history

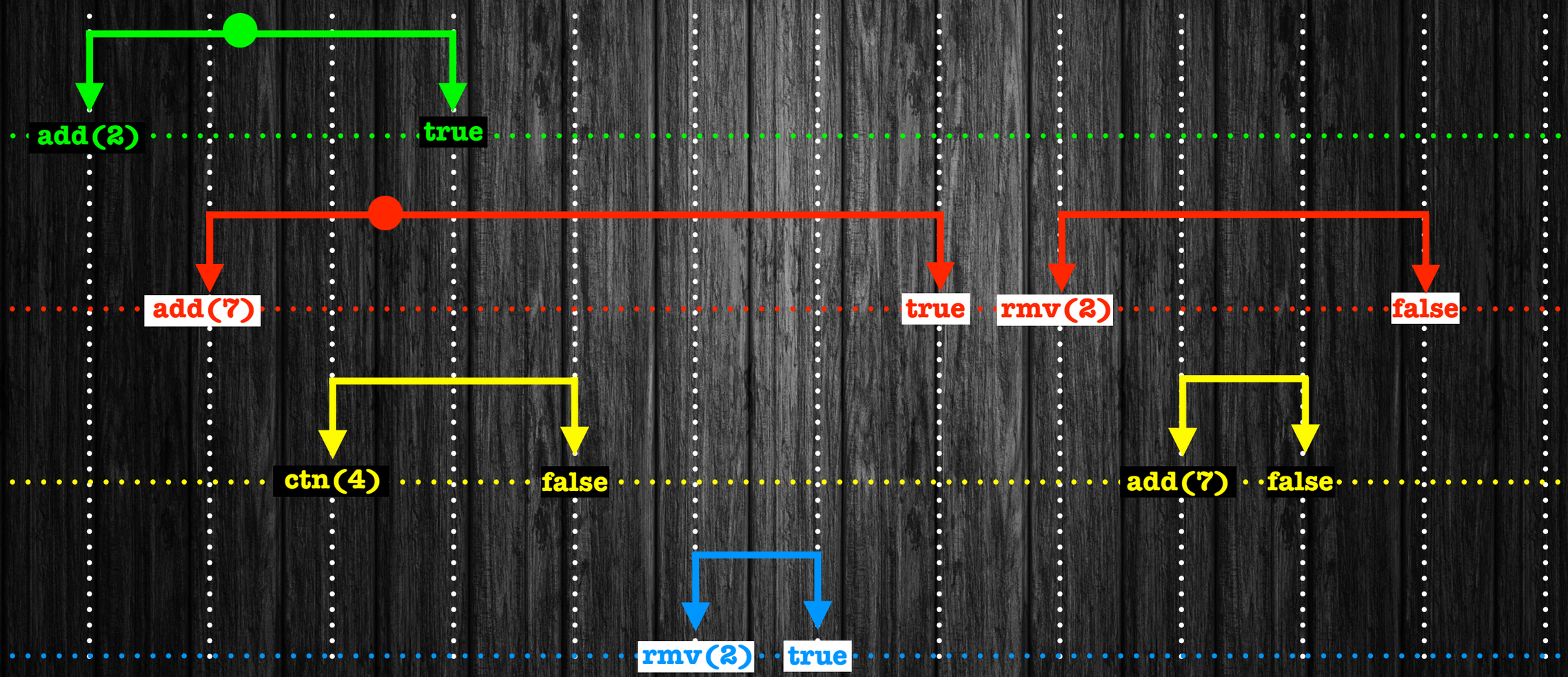


Concurrent Histories

Linearizability

Linearization Points — a point inside each call

H — Concurrent history



Concurrent Histories

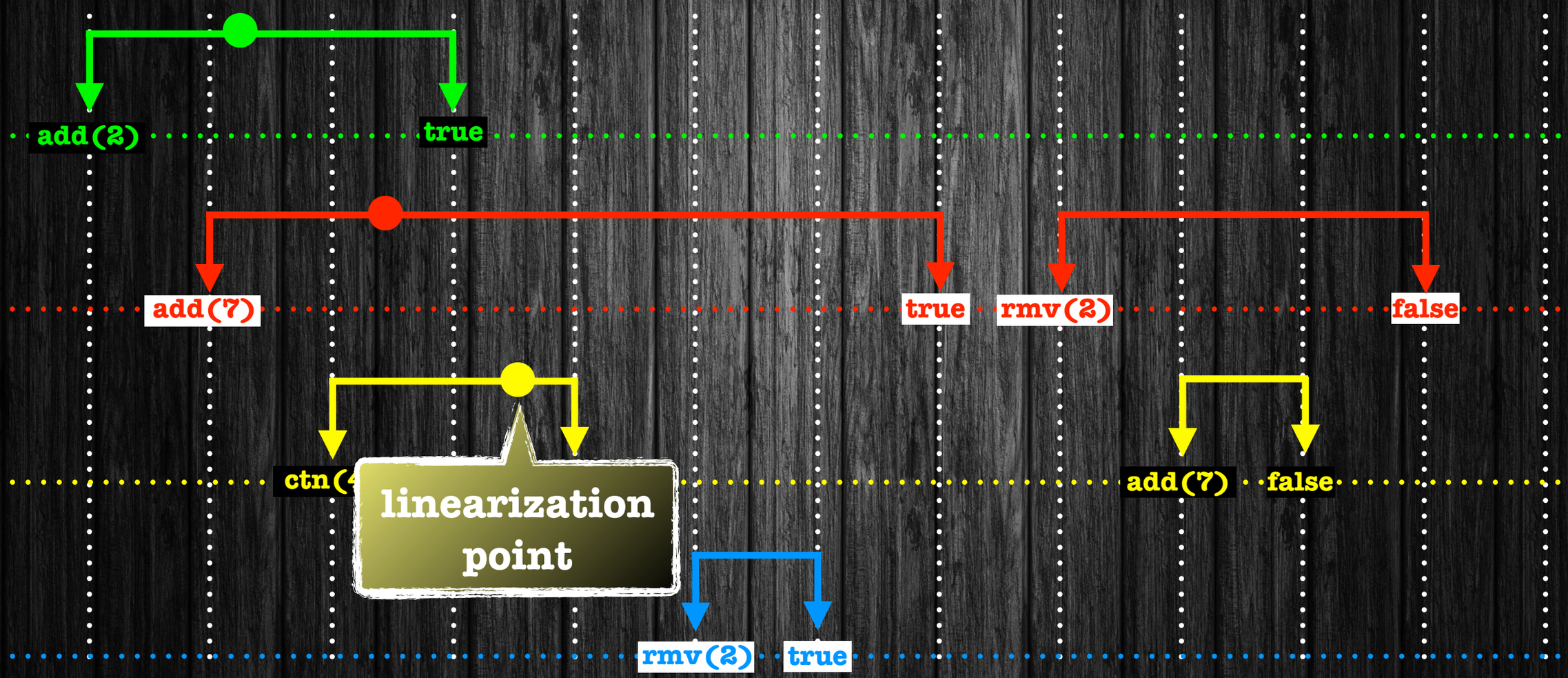
Linearizability

Linearization Points

a point inside each call

H

Concurrent history



Concurrent Histories

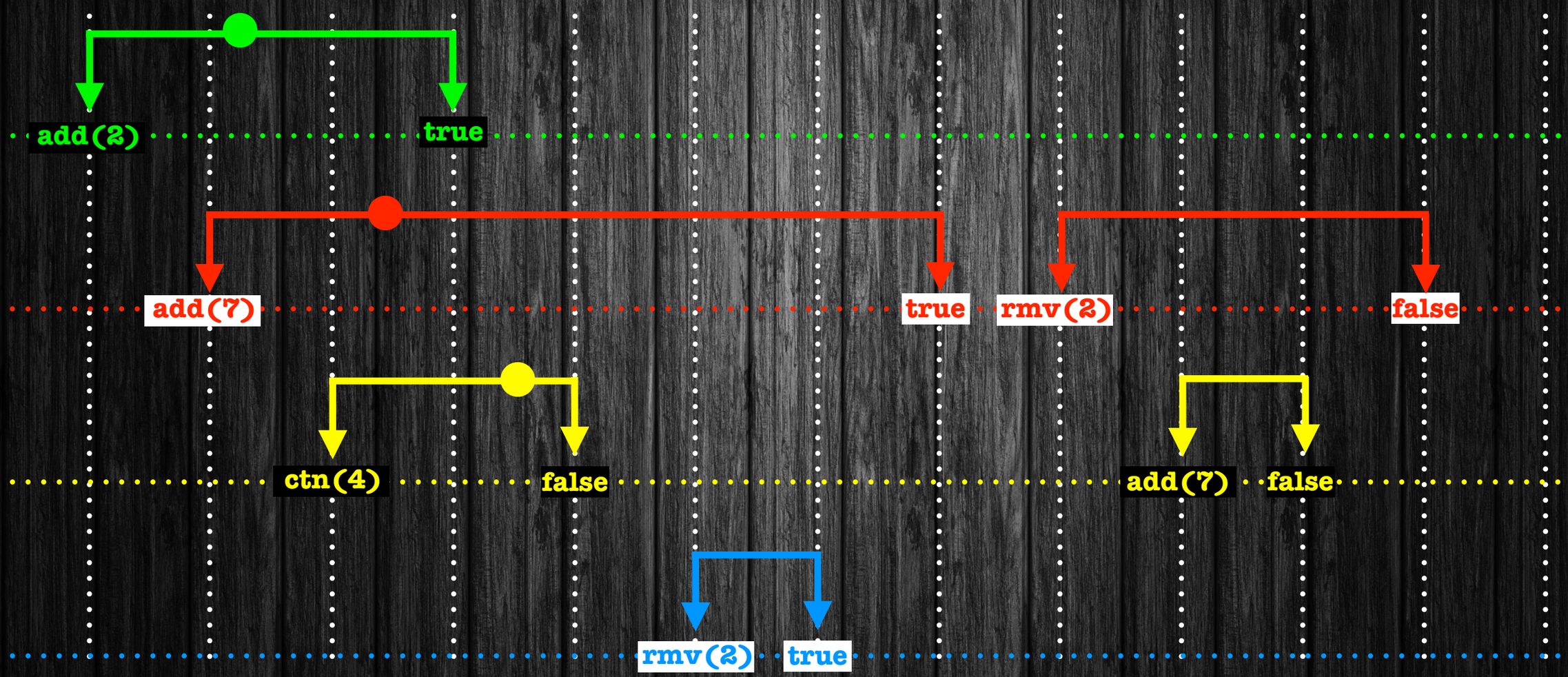
Linearizability

Linearization Points

a point inside each call

H

Concurrent history



Concurrent Histories

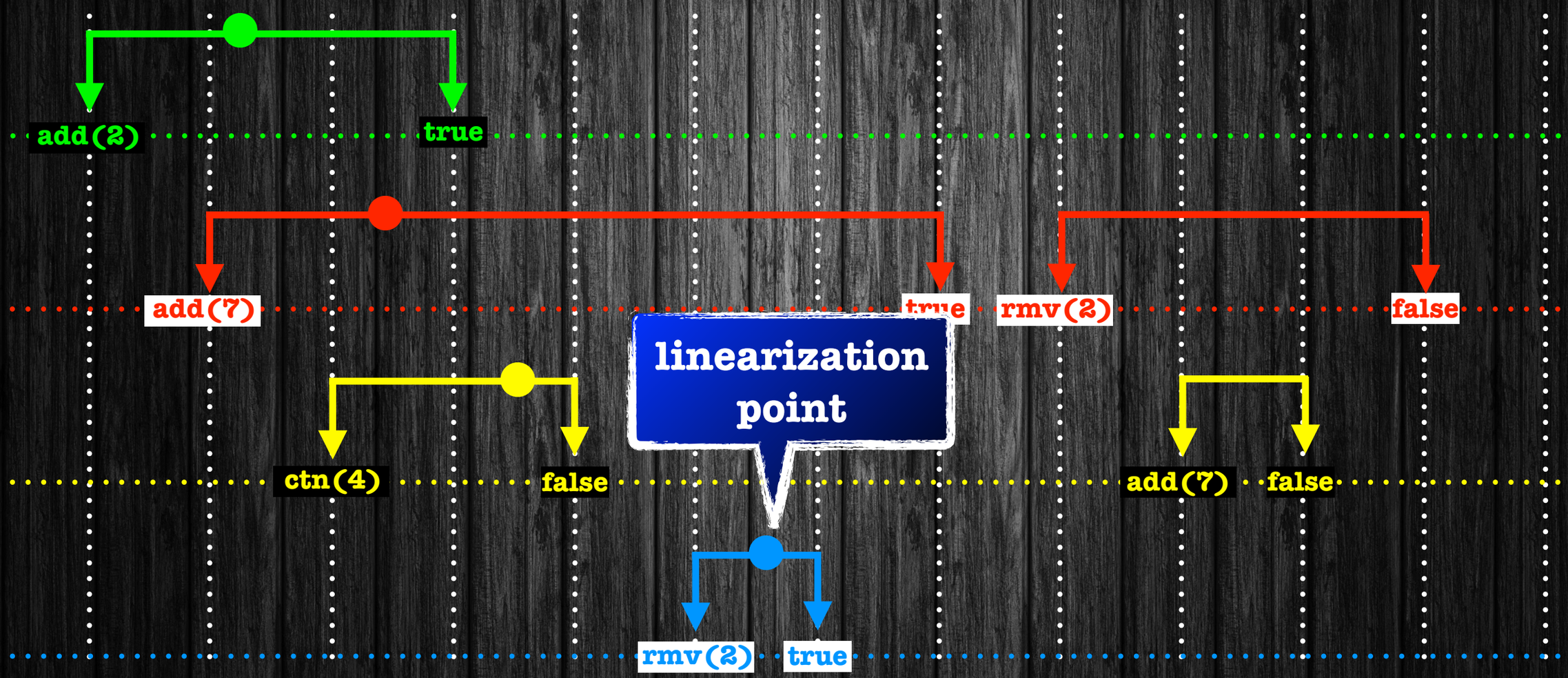
Linearizability

Linearization Points

a point inside each call

H

Concurrent history

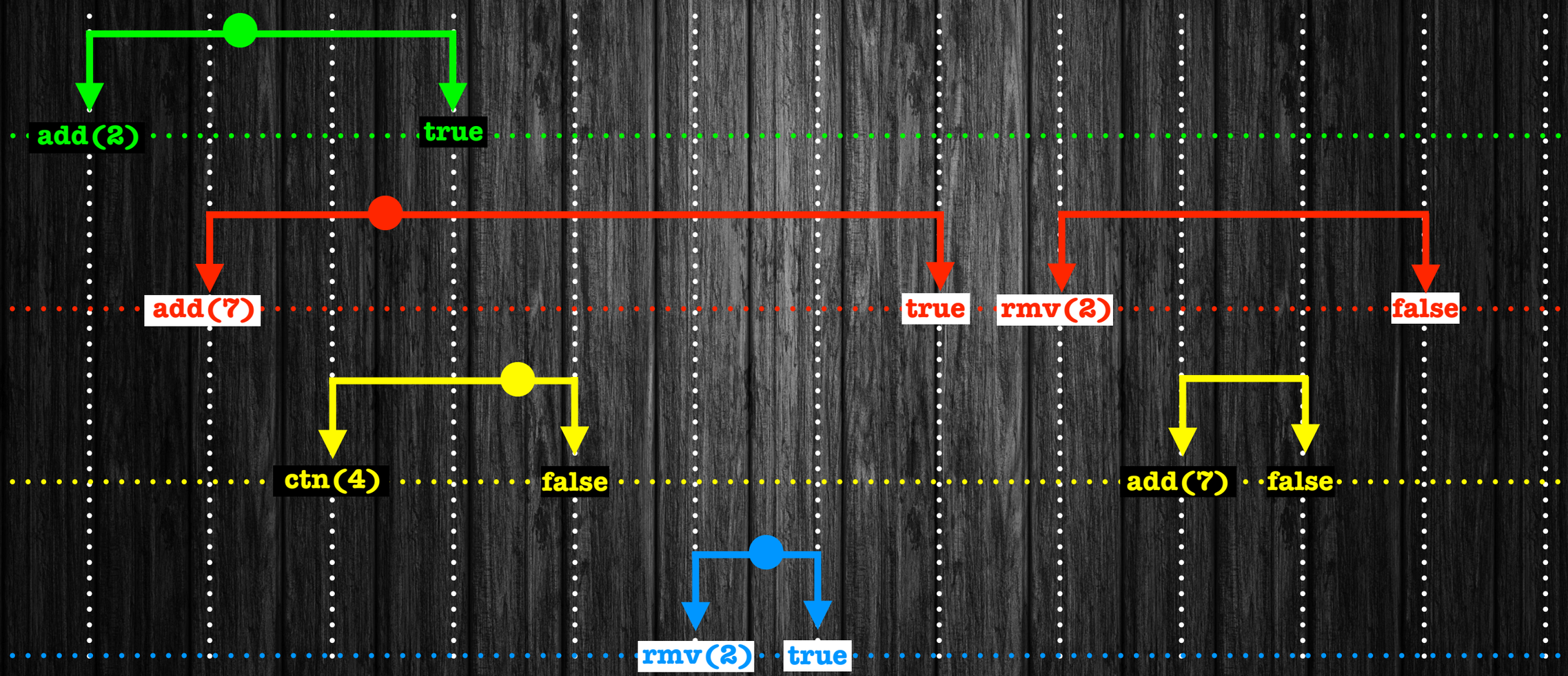


Concurrent Histories

Linearizability

Linearization Points — a point inside each call

H — Concurrent history



Concurrent Histories

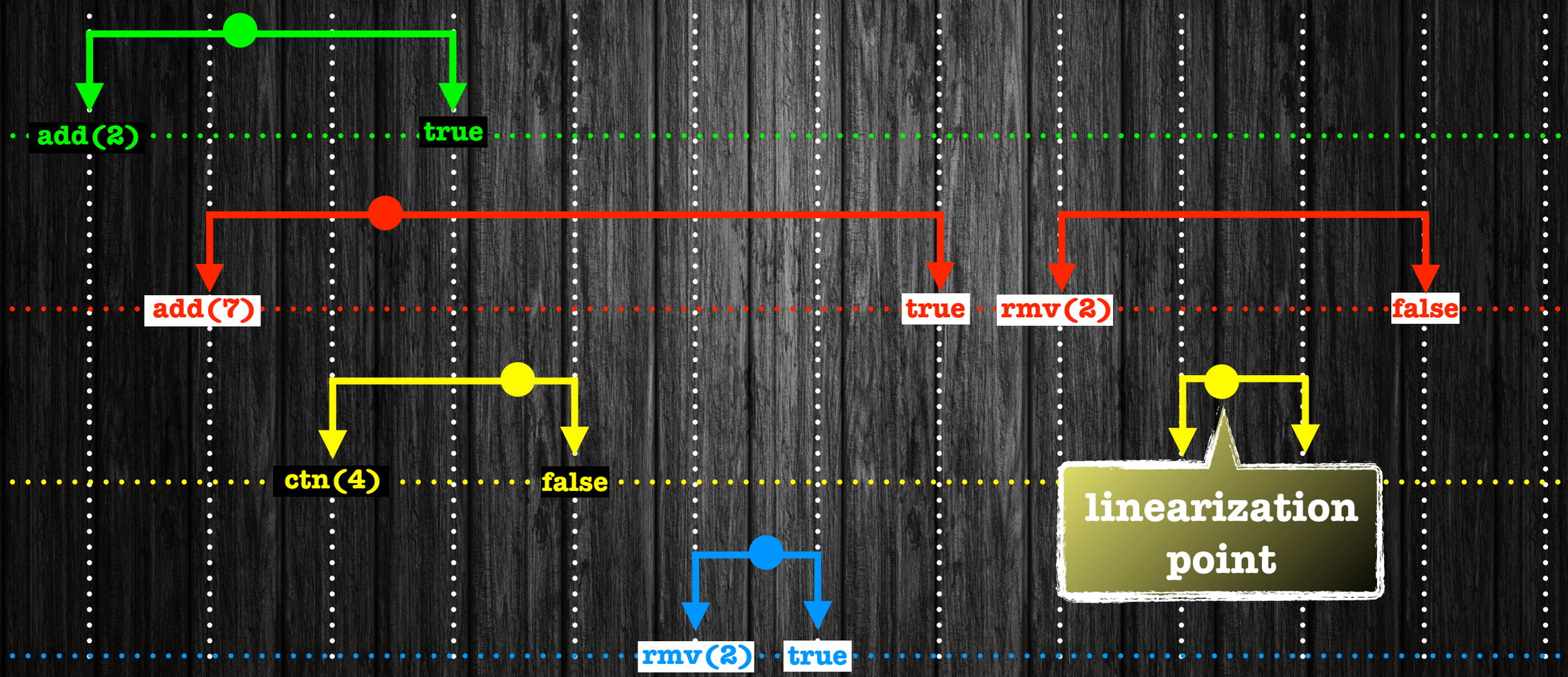
Linearizability

Linearization Points

a point inside each call

H

Concurrent history

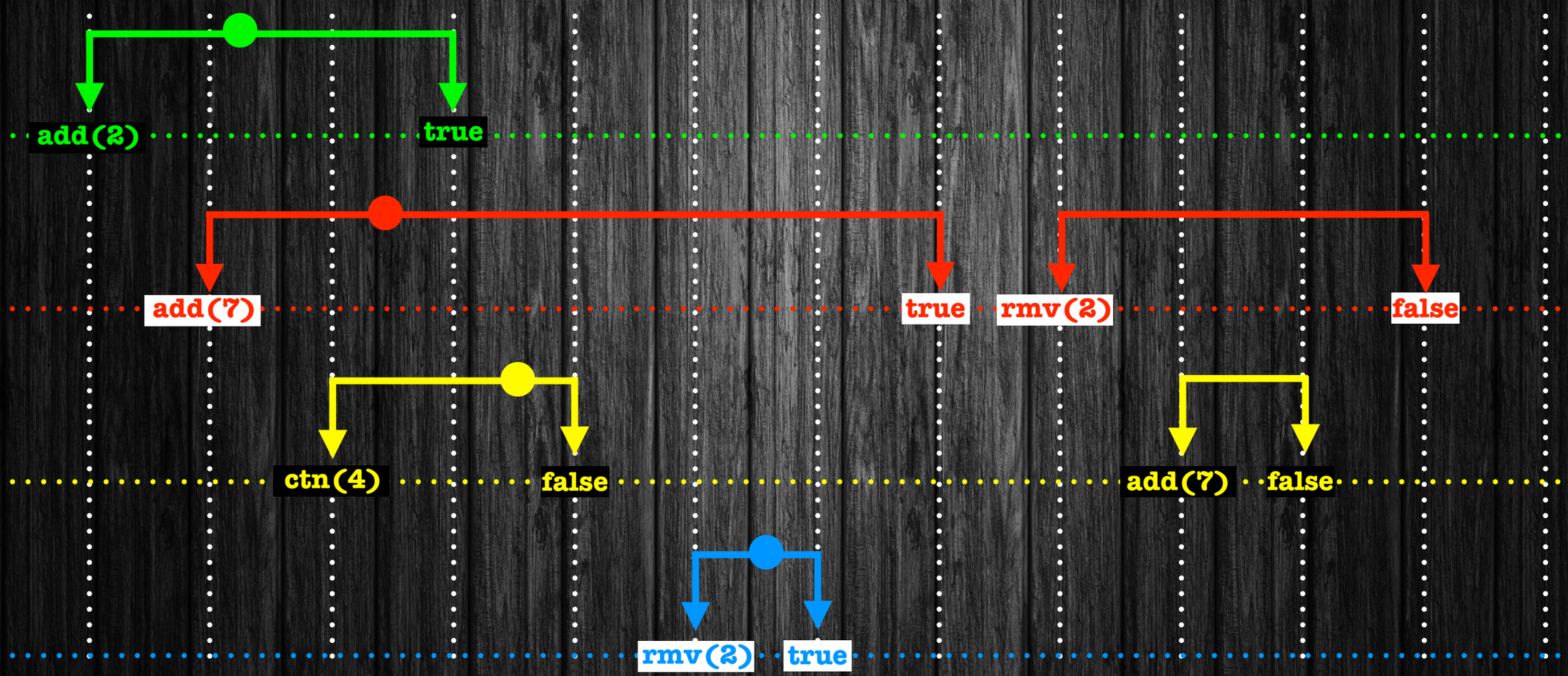


Concurrent Histories

Linearizability

Linearization Points — a point inside each call

H — Concurrent history



Concurrent Histories

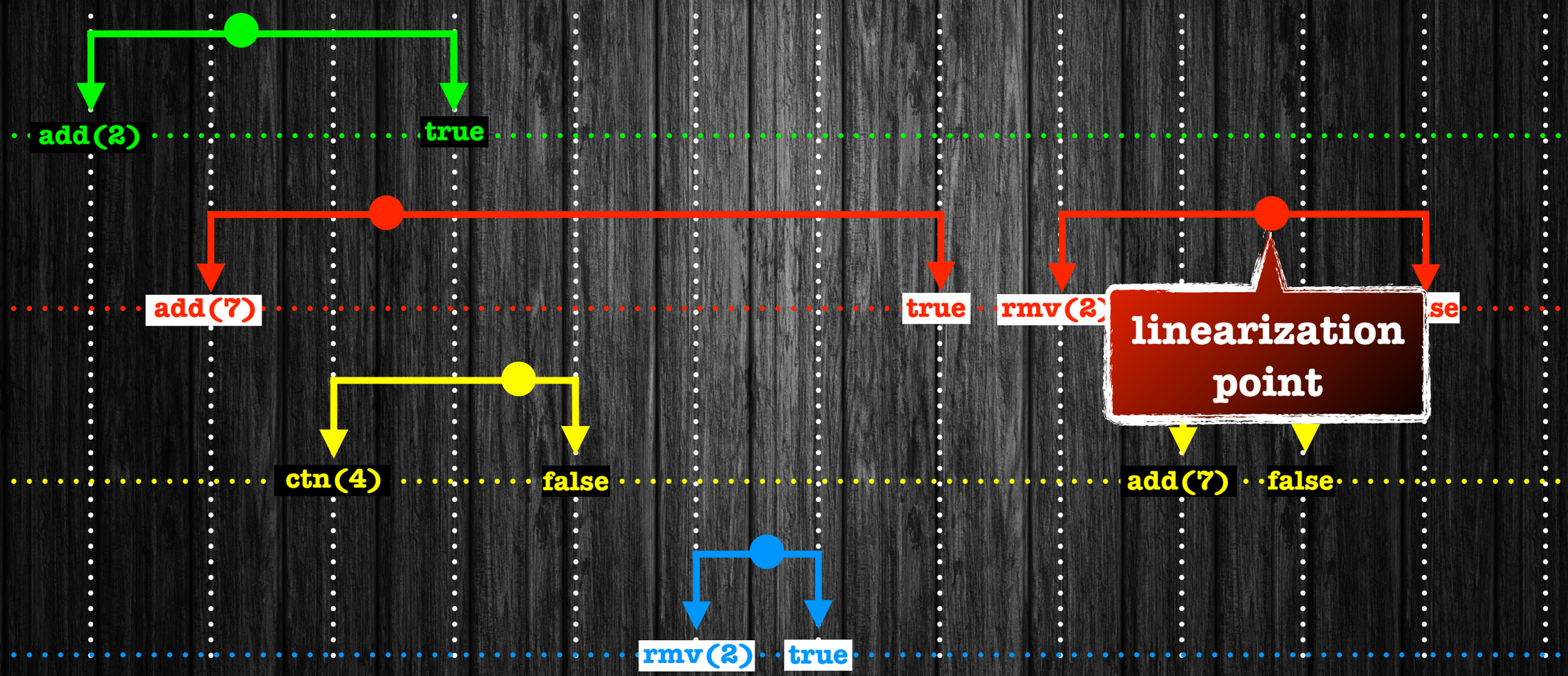
Linearizability

Linearization Points

a point inside each call

H

Concurrent history



Concurrent Histories | **Linearizability**

Linearization Operation

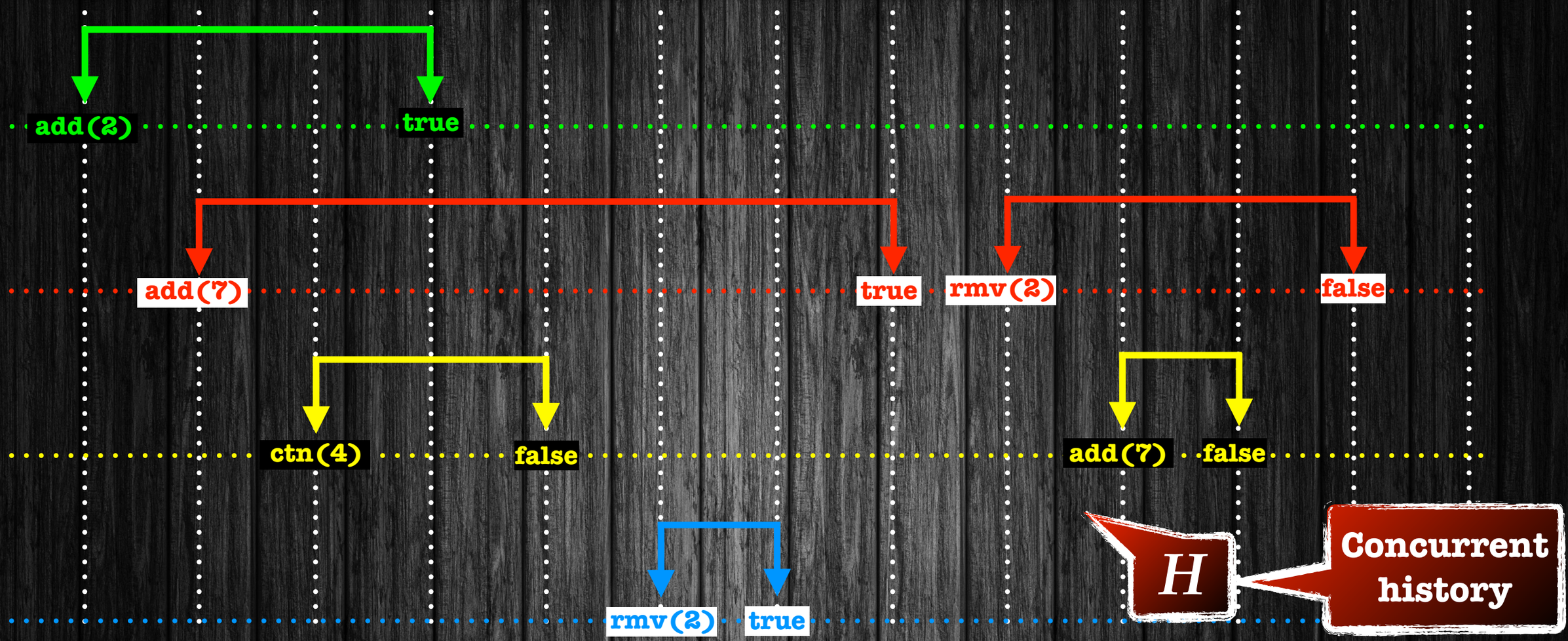
Concurrent Histories

Linearizability

Concurrent Histories

Linearizability

Linearization Operation



Concurrent Histories

Linearizabilitv

Linearization Operation

add(2)

true

add(7)

true

rmv(2)

false

ctn(4)

false

add(7)

false

rmv(2)

true

H

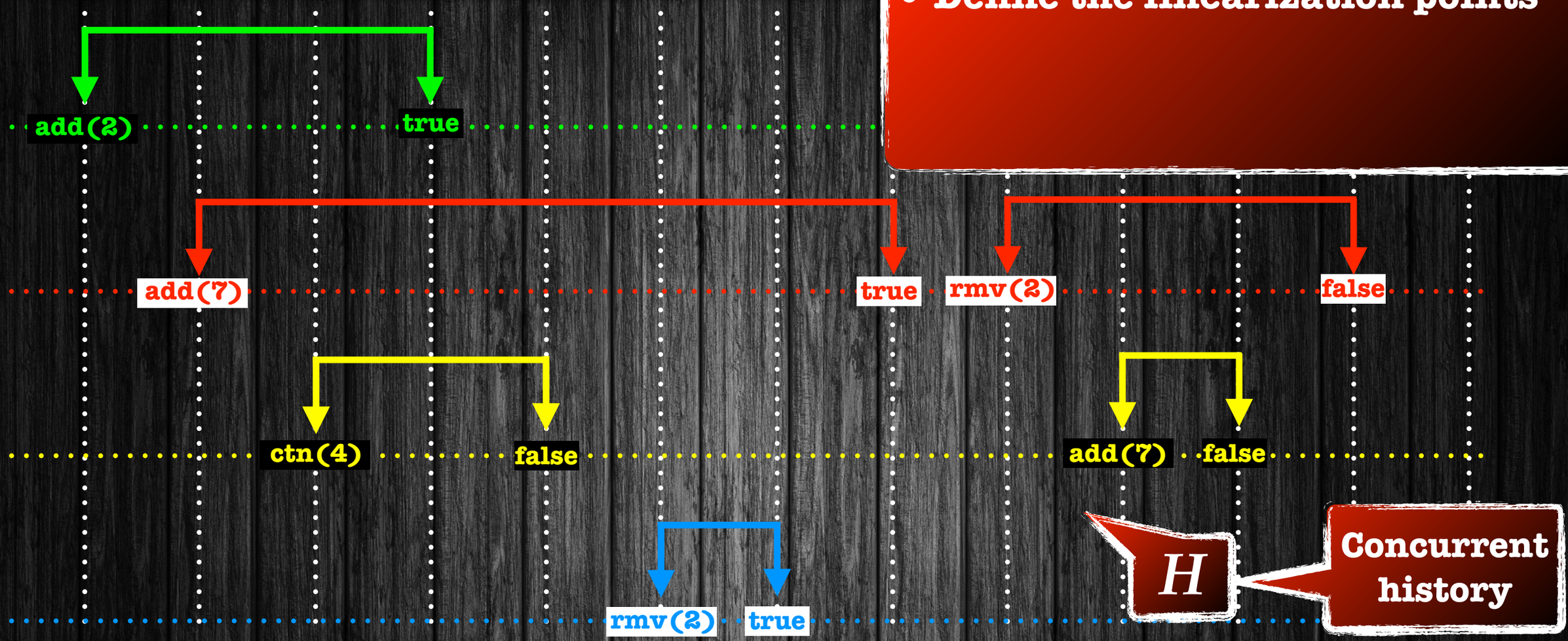
Concurrent history

Concurrent Histories

Linearizabilitv

Linearization Operation

- **Define the linearization points**



Concurrent Histories

Linearizabilitv

Linearization Operation

- **Define the linearization points**

add(2) true

add(7) true rmv(2) false

ctn(4) false add(7) false

rmv(2) true

H

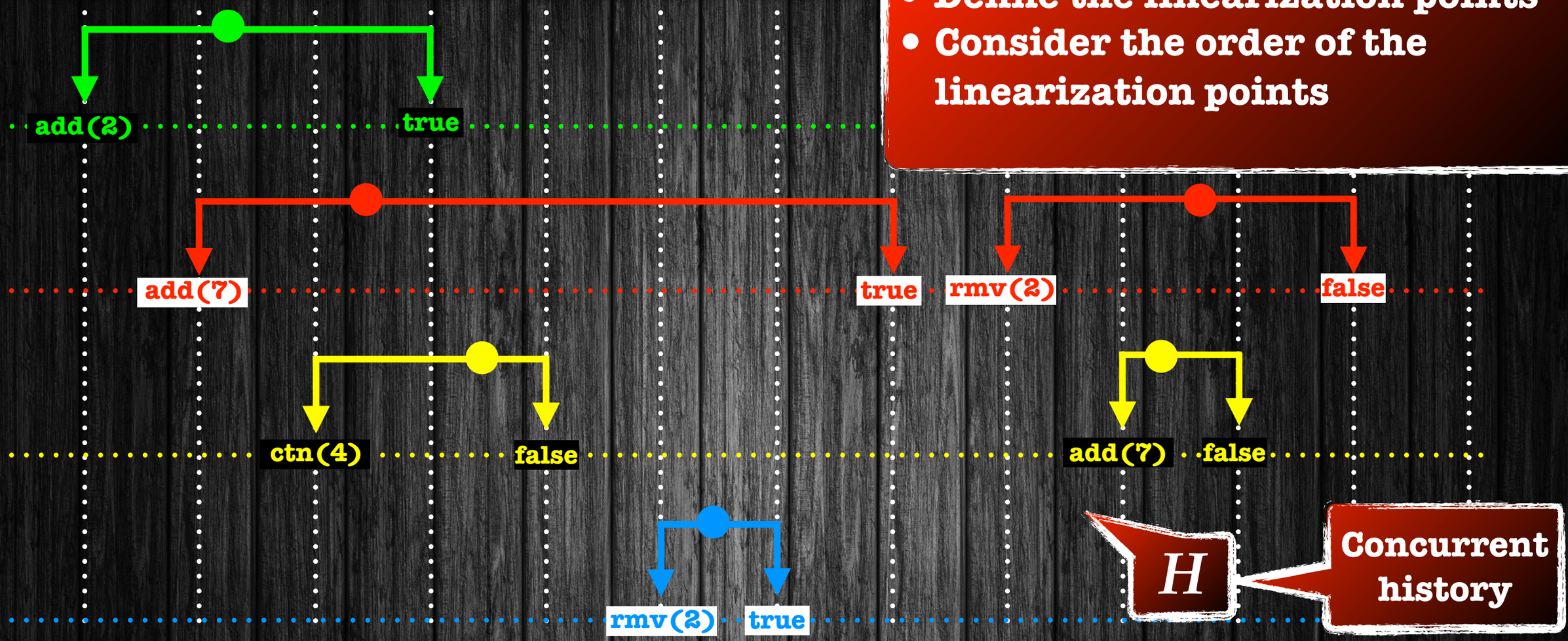
Concurrent history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points



H

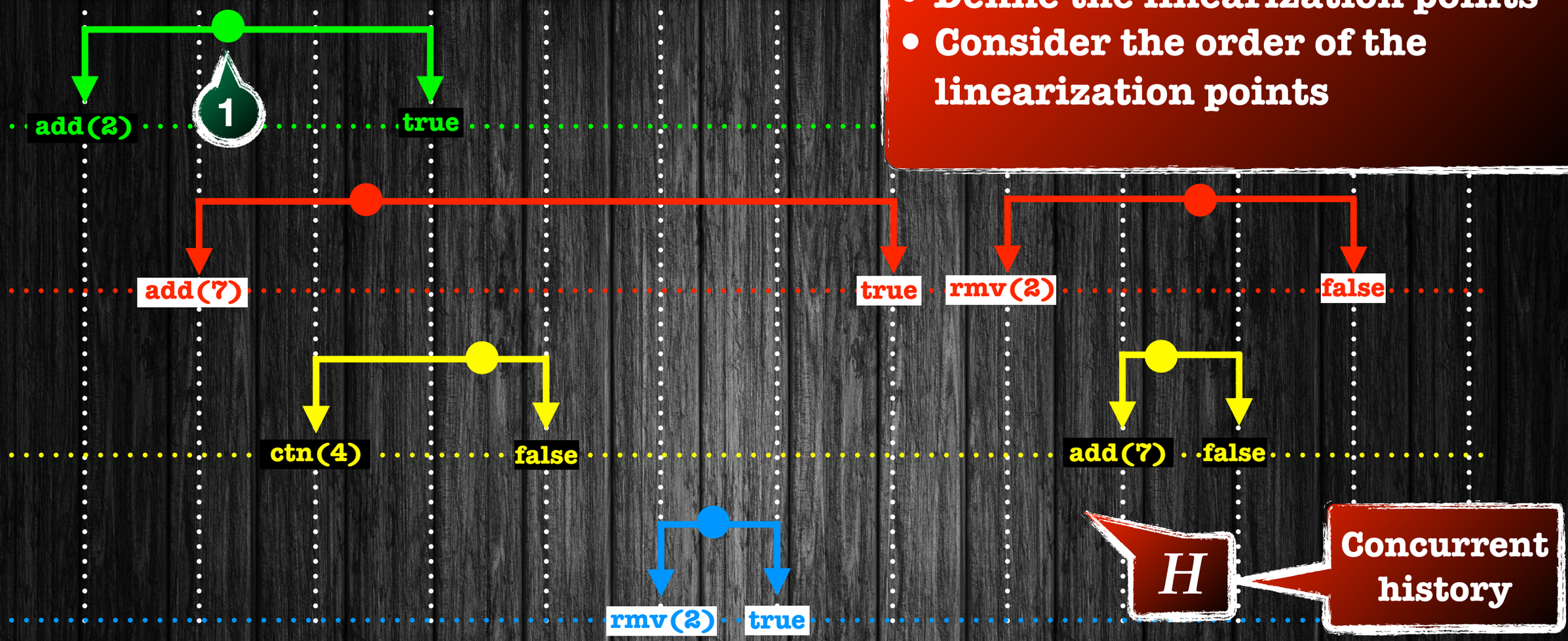
Concurrent history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points

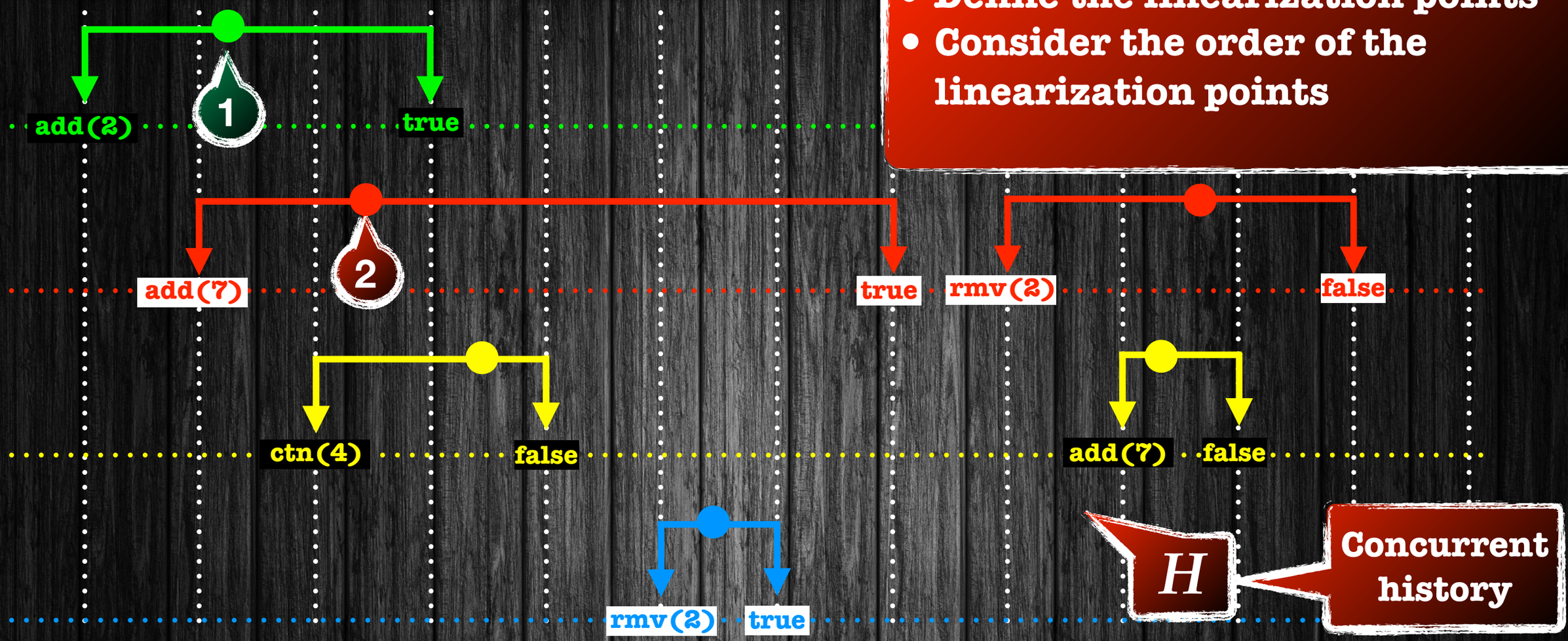


Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points

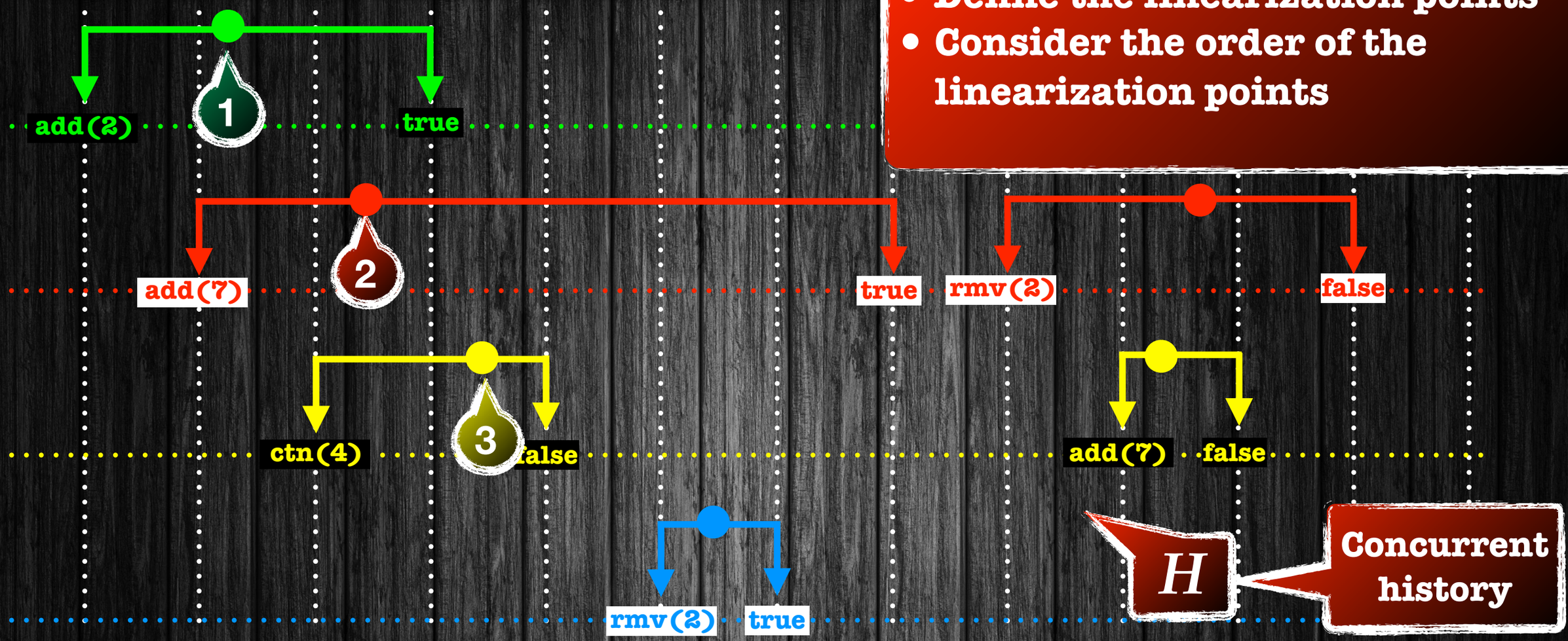


Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points



Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) false

add(7) false

H

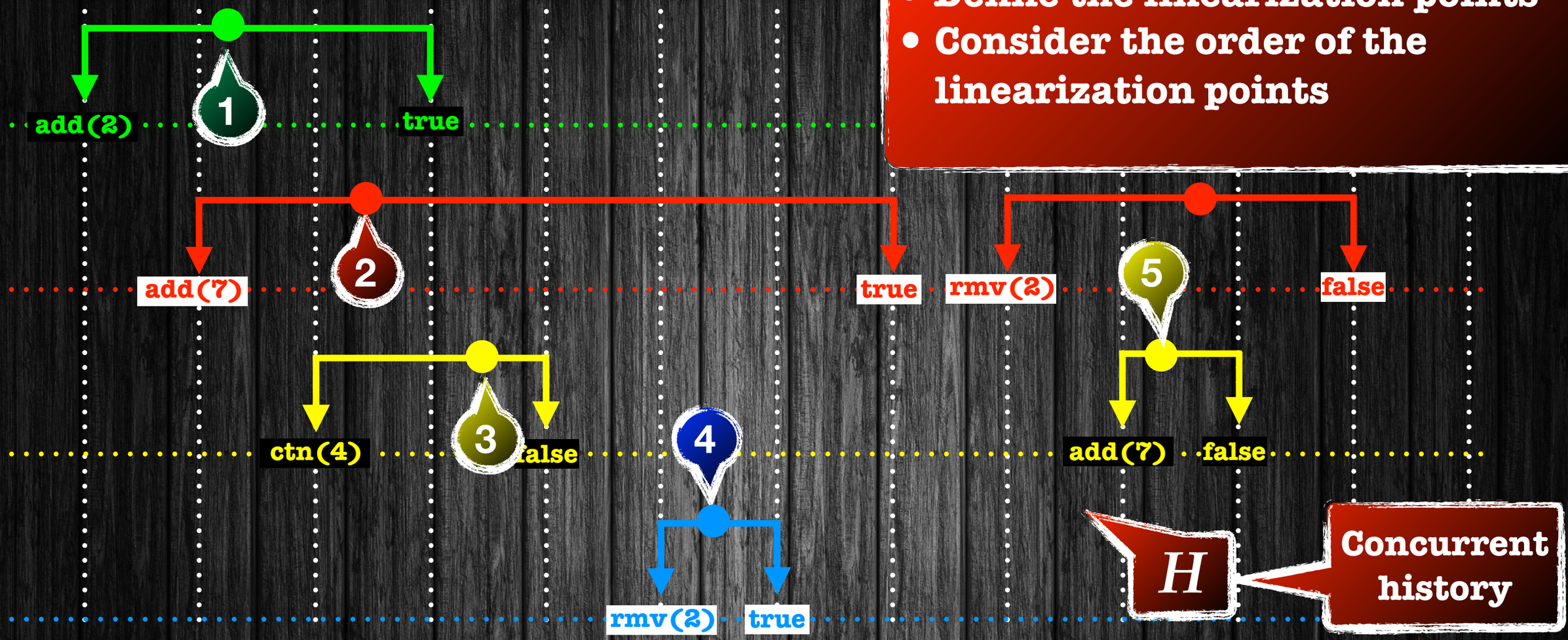
Concurrent history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points



H

Concurrent history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

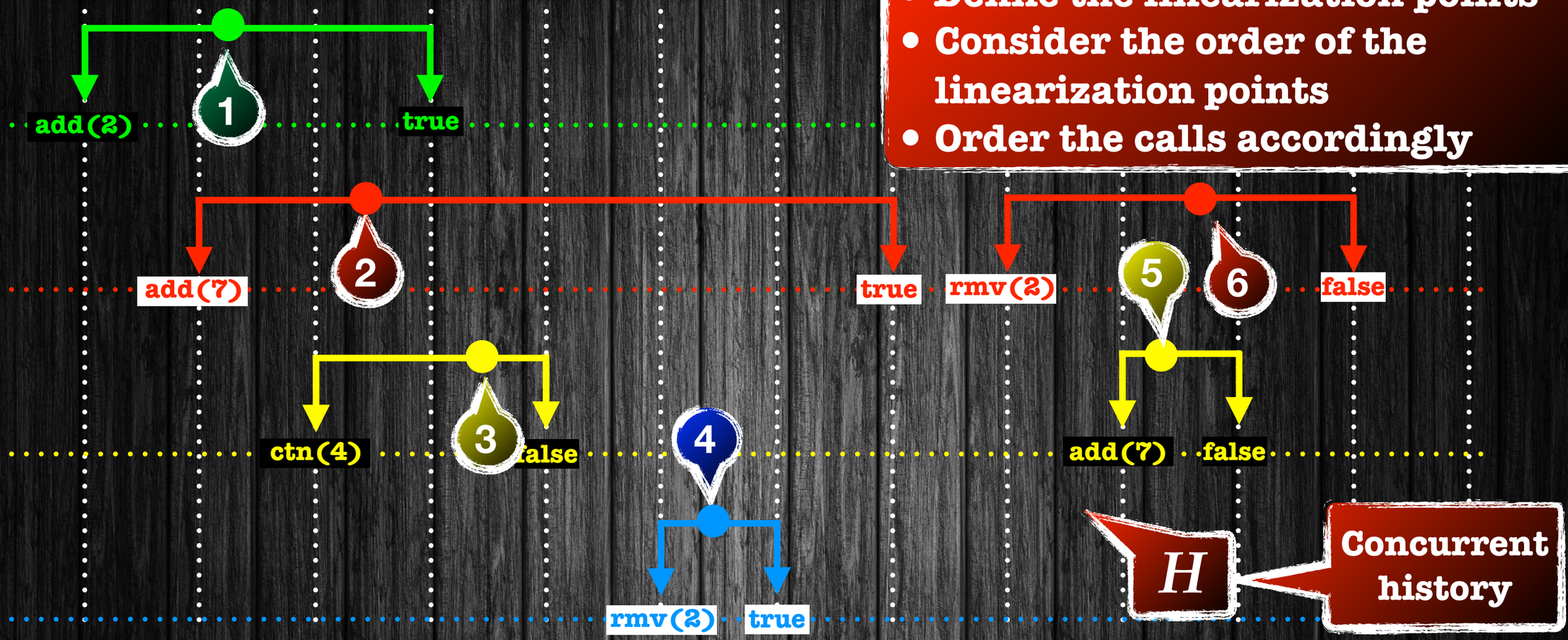
Concurrent history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

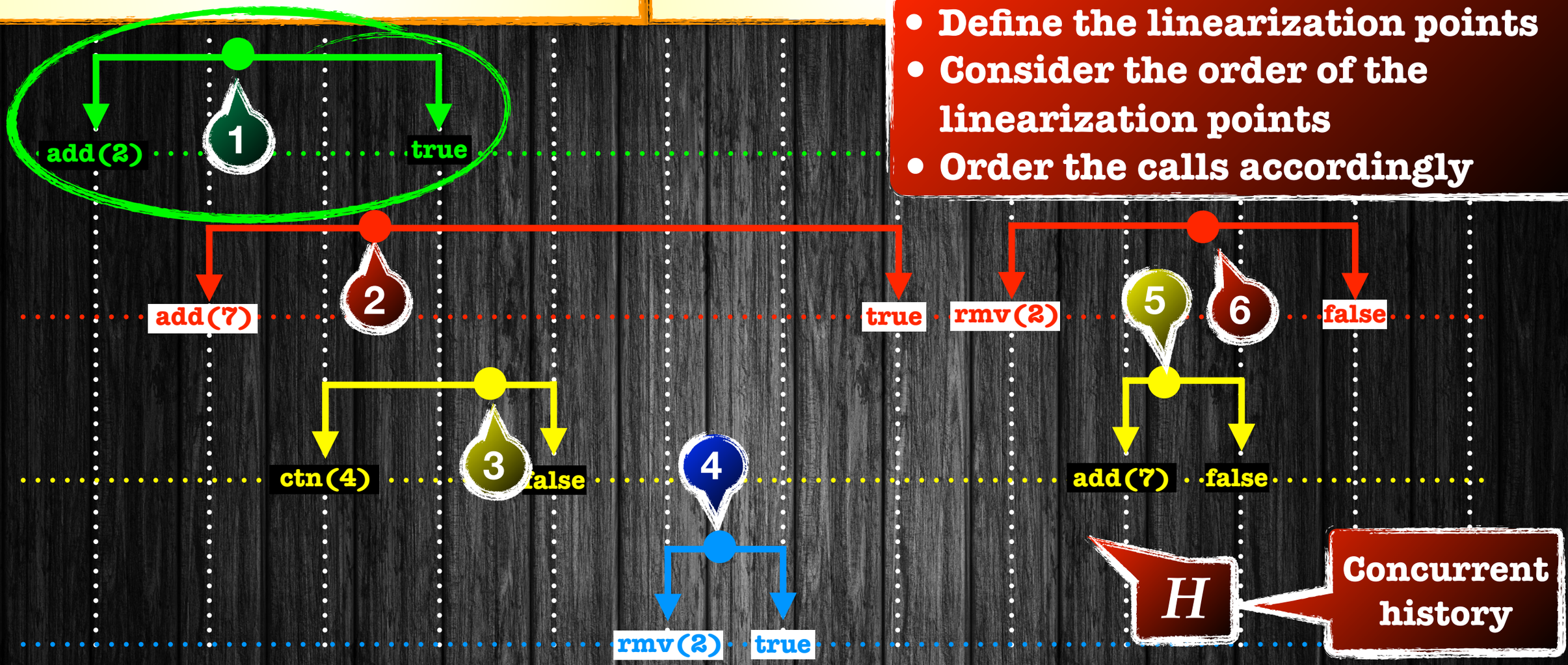
Concurrent history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

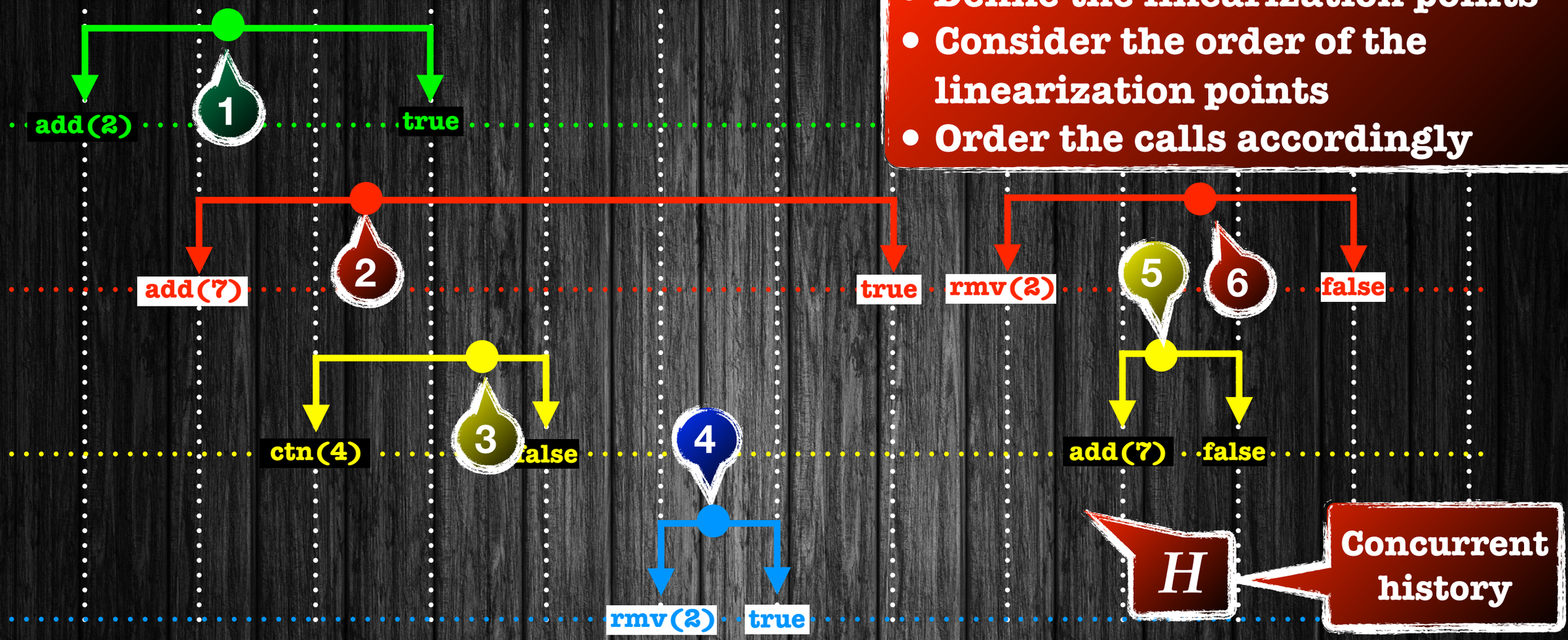
add(2) true

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



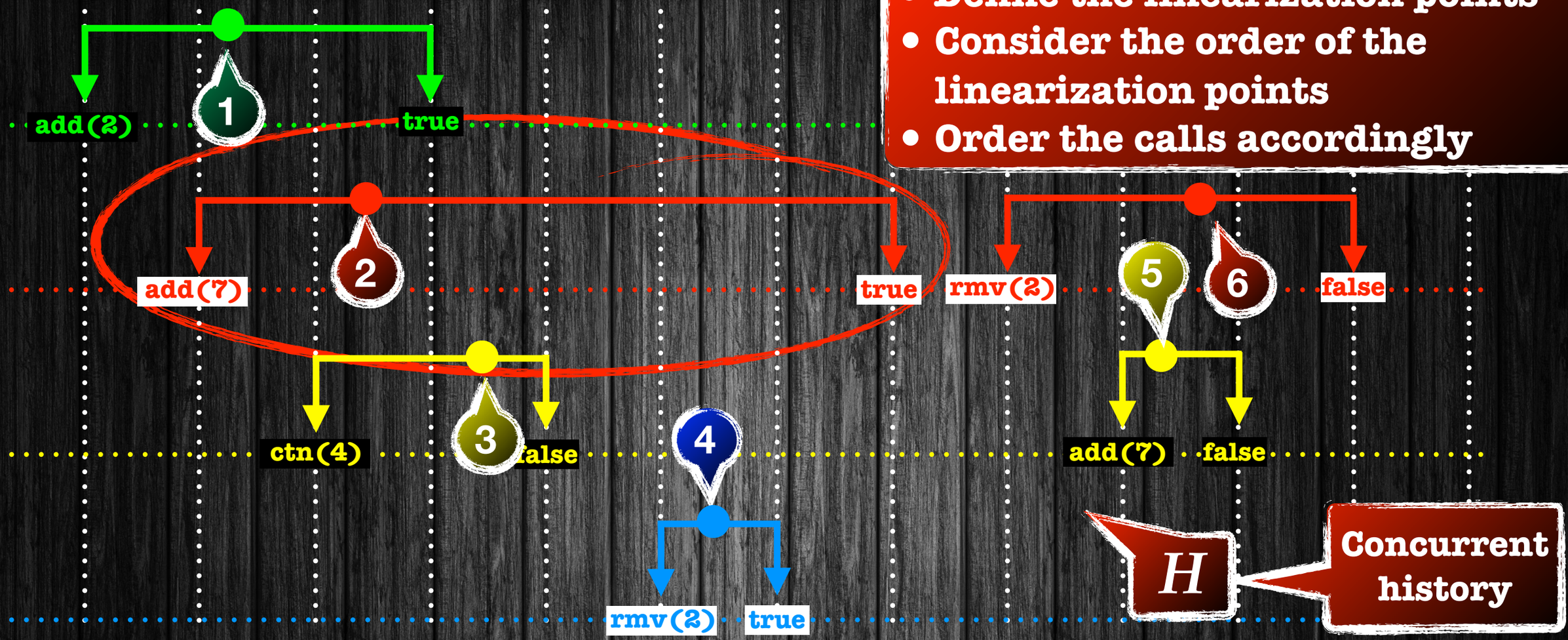


Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



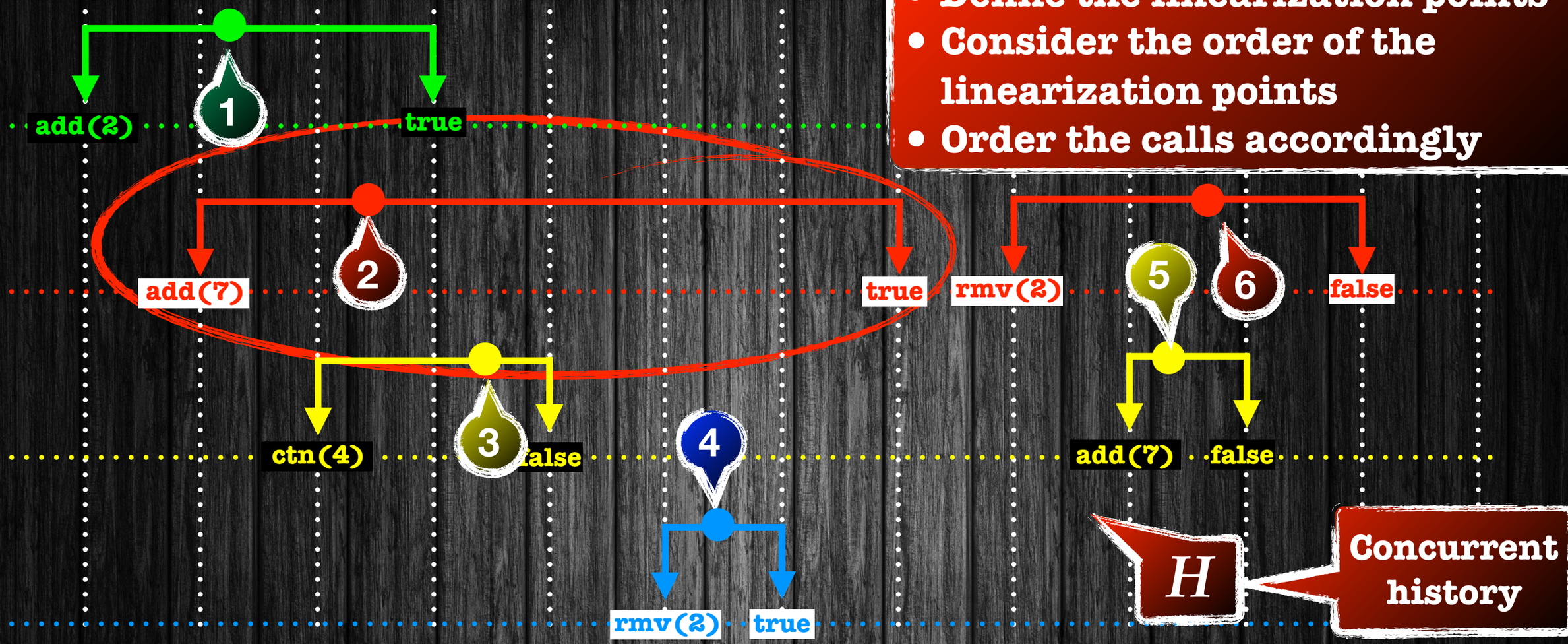


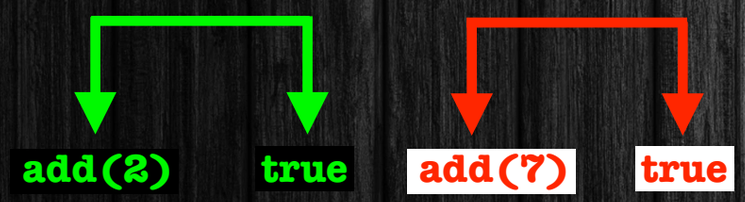
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



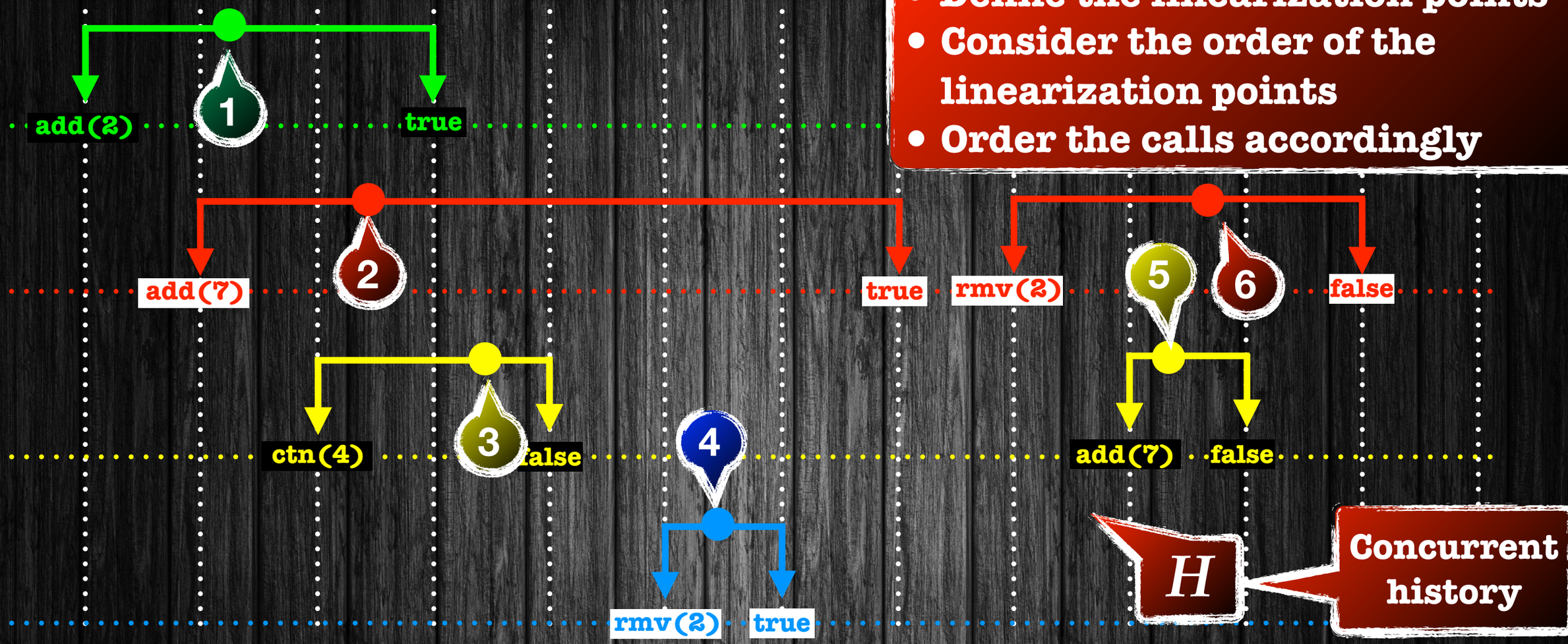


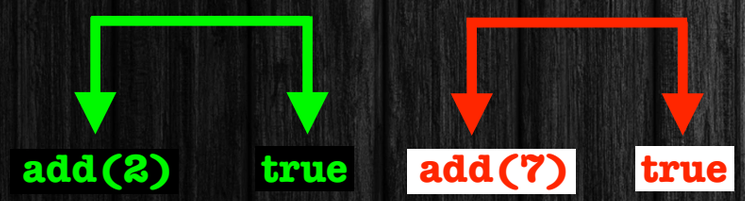
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



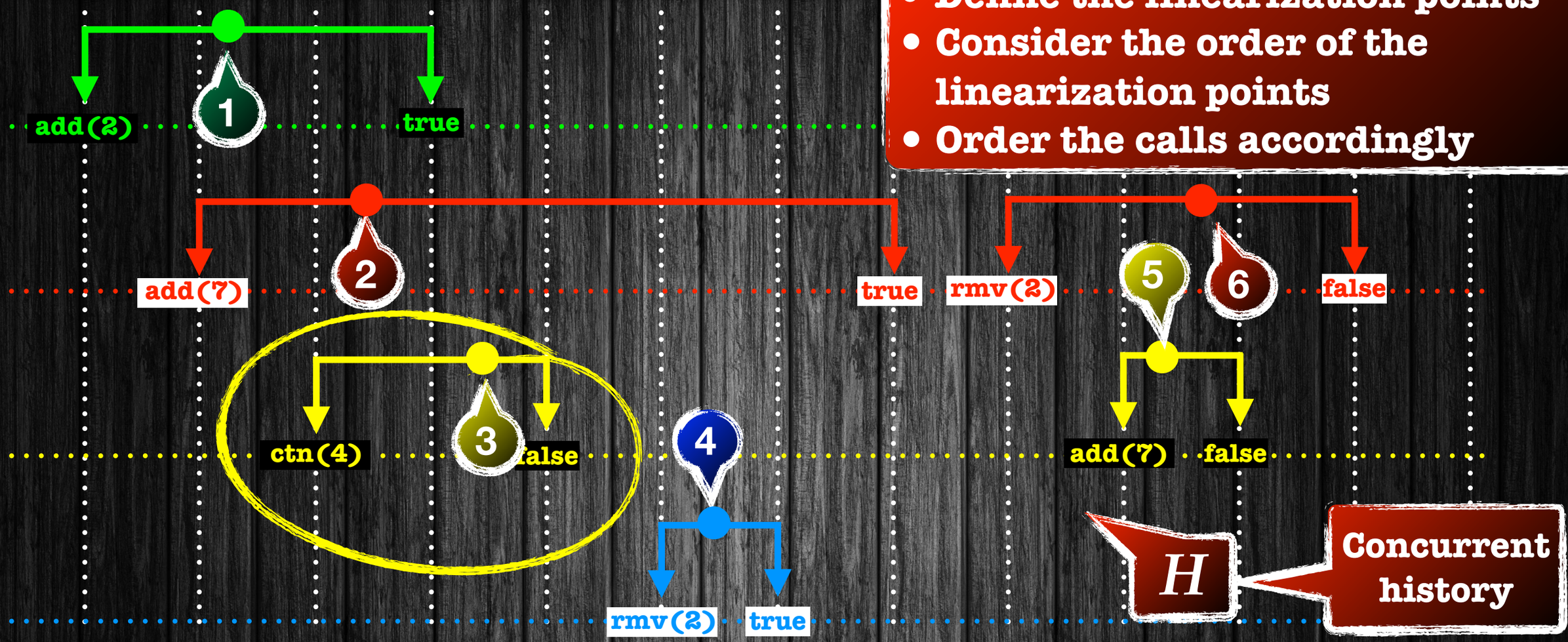


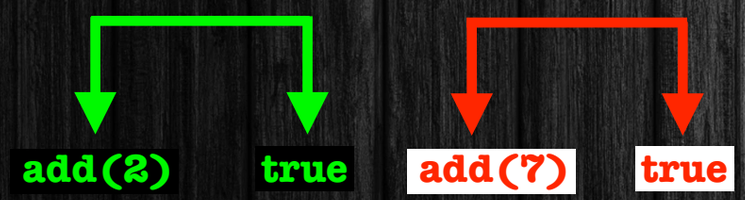
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



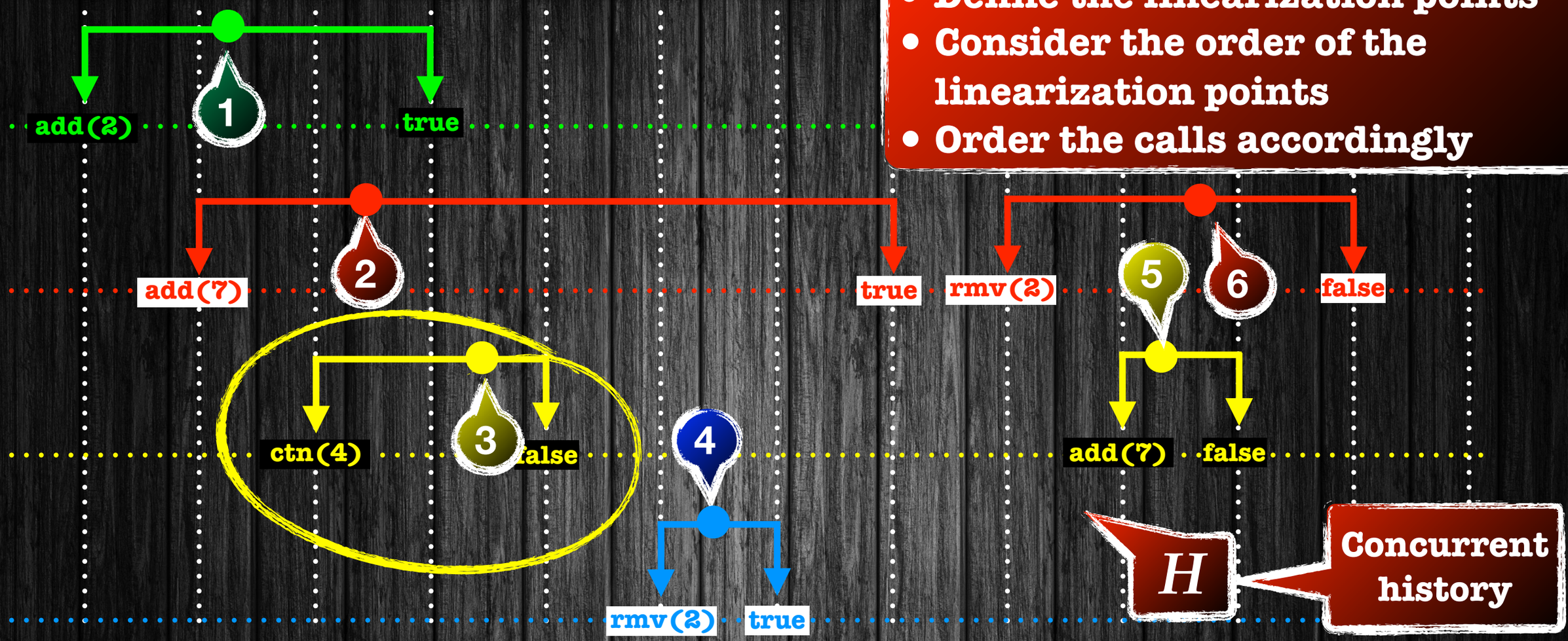


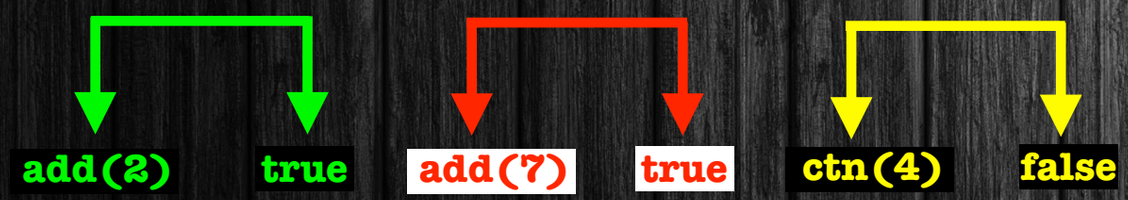
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



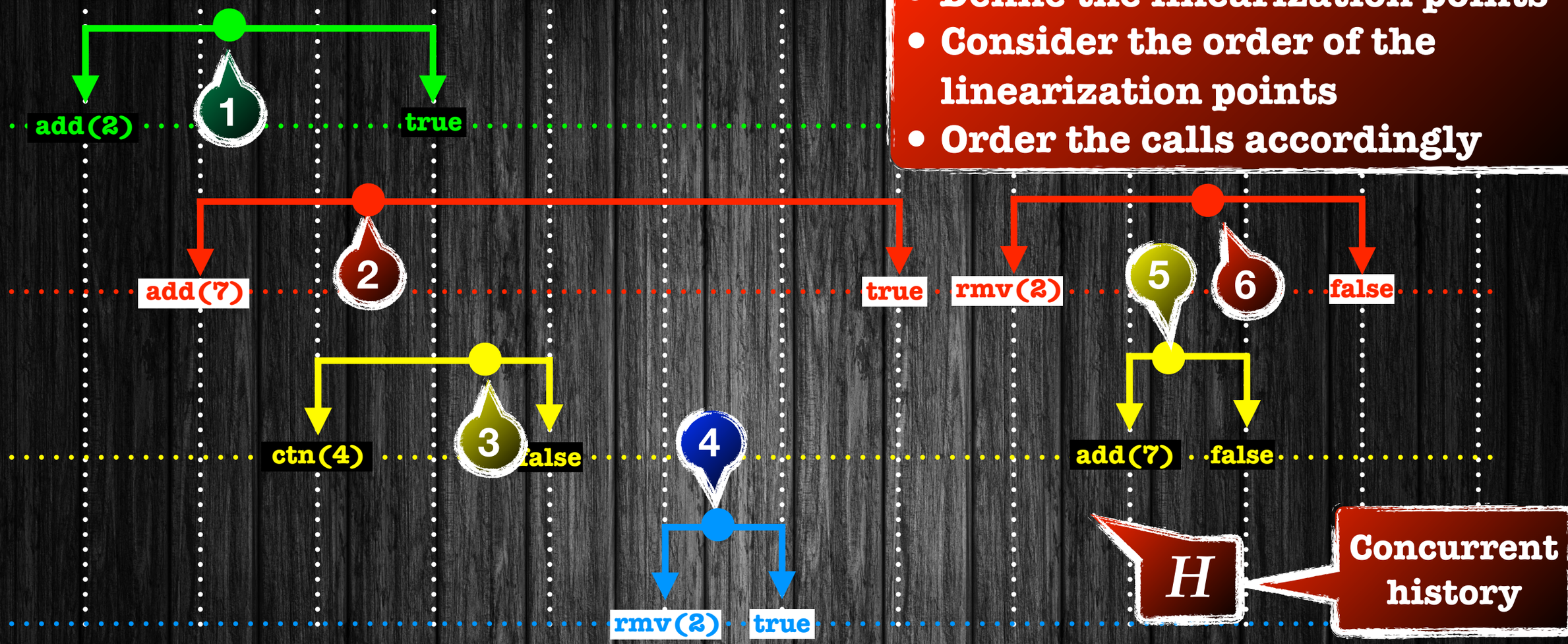


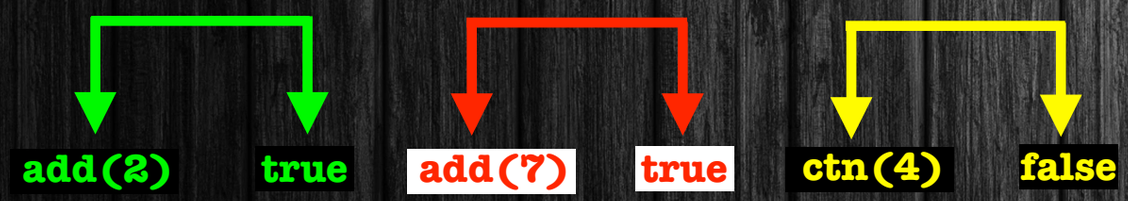
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly





Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

add(2) true

add(7) true

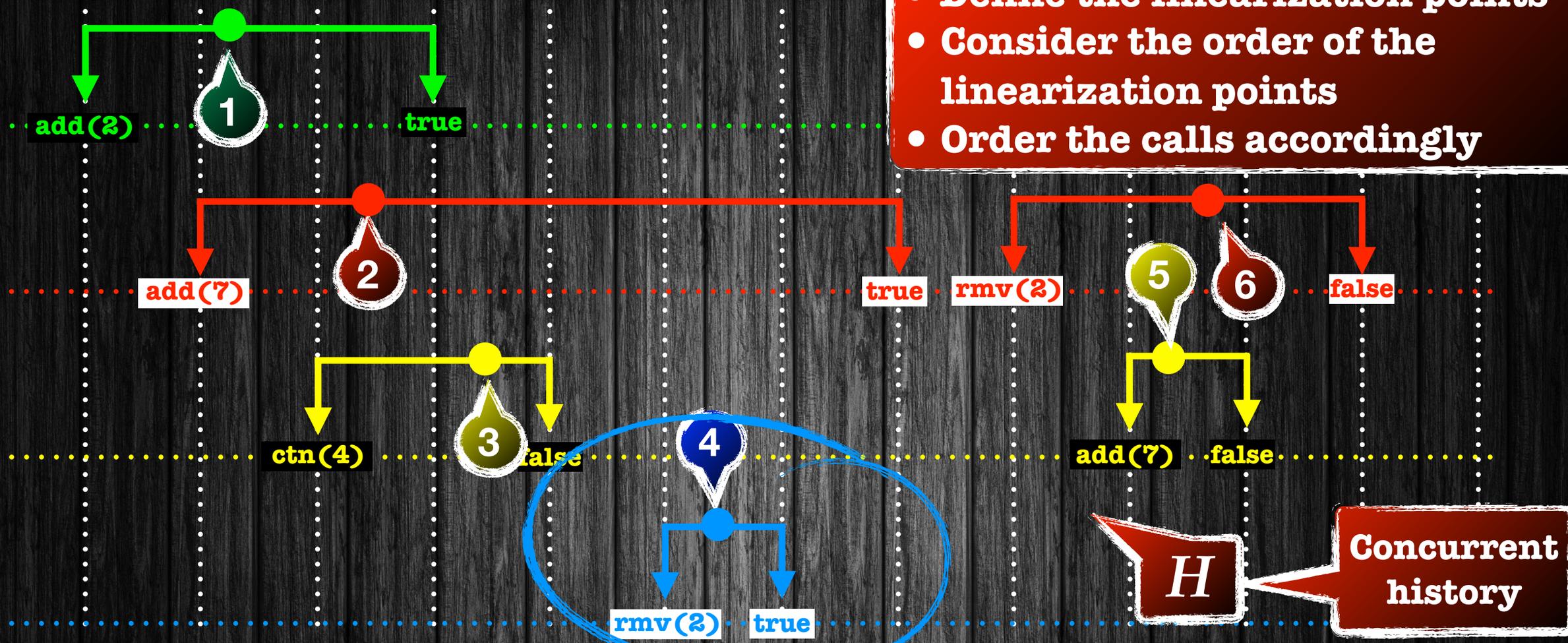
ctn(4) false

Concurrent Histories

Linearizability

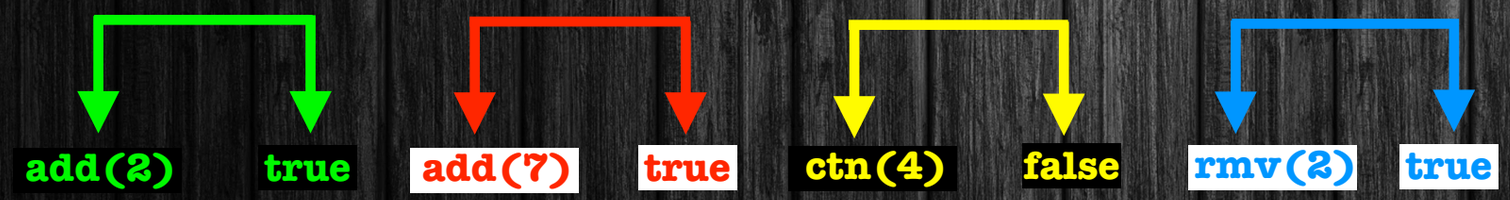
Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history

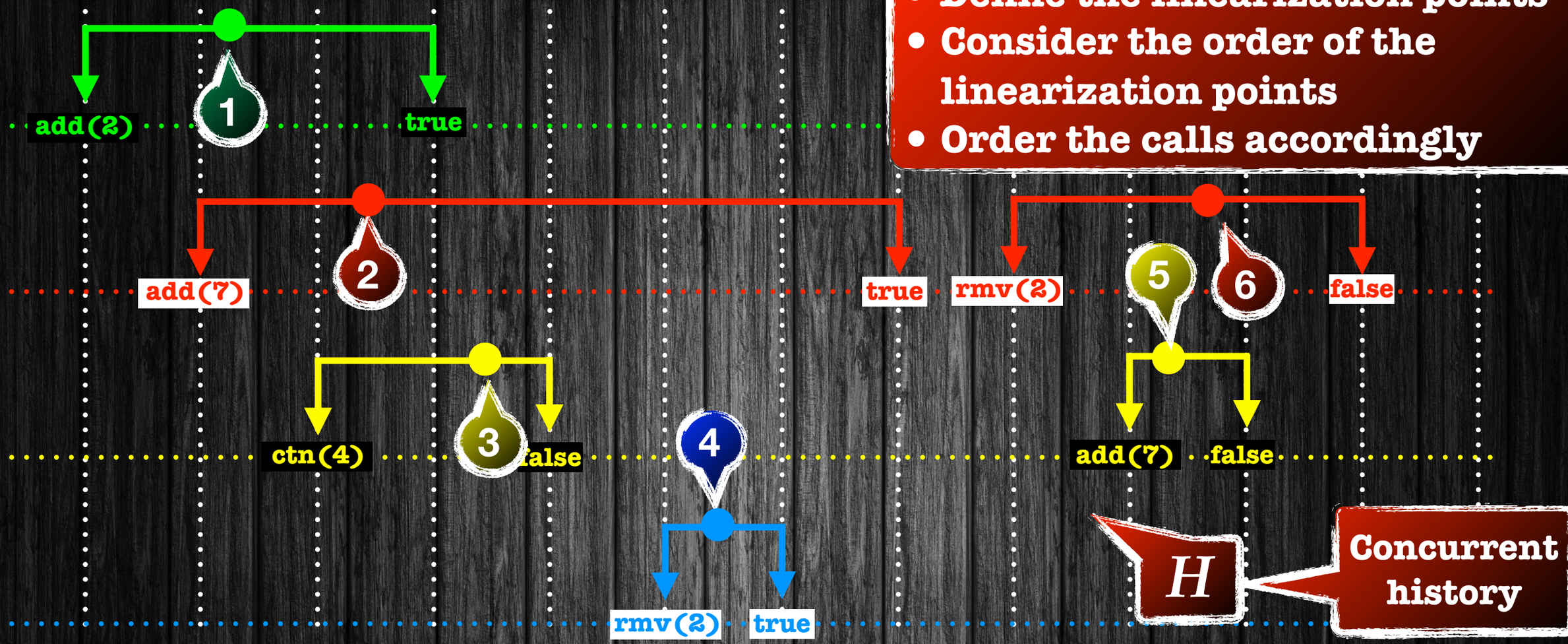


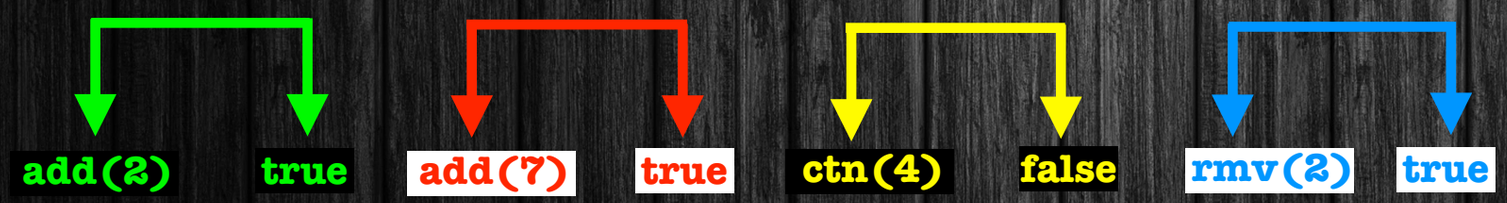
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



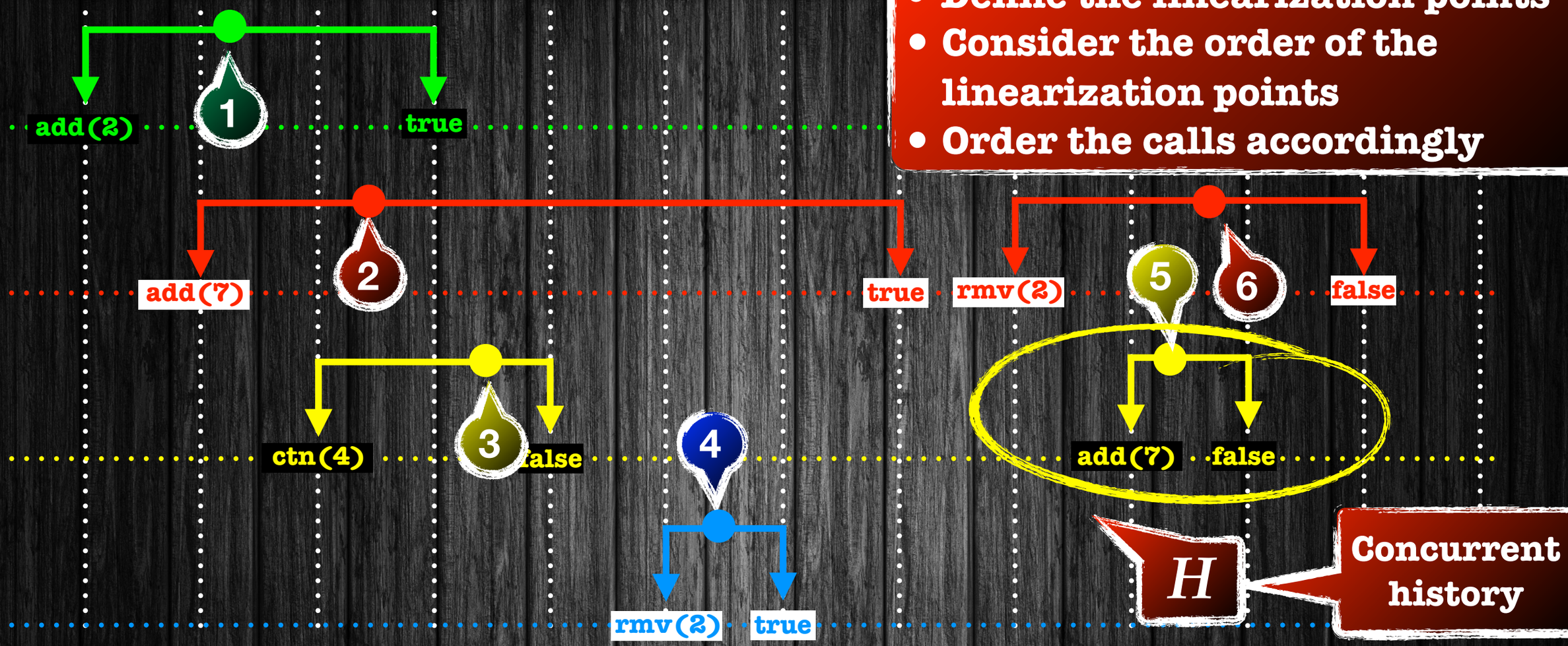


Concurrent Histories

Linearizability

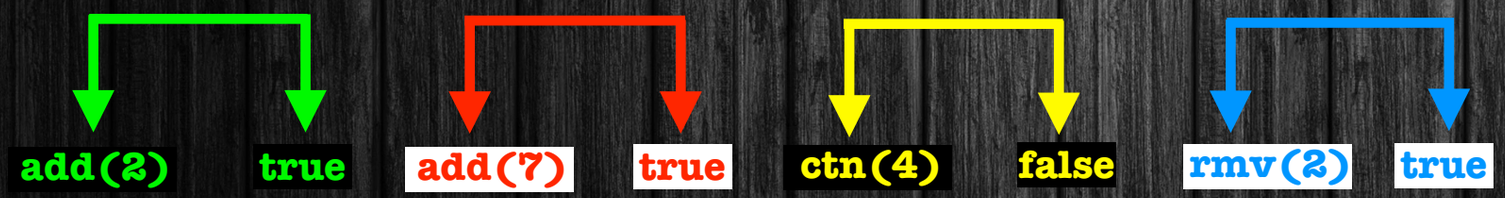
Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history



Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) 5 6 false

add(7) false

rmv(2) true

H

Concurrent history

add(2) true

add(7) true

ctn(4) false

rmv(2) true

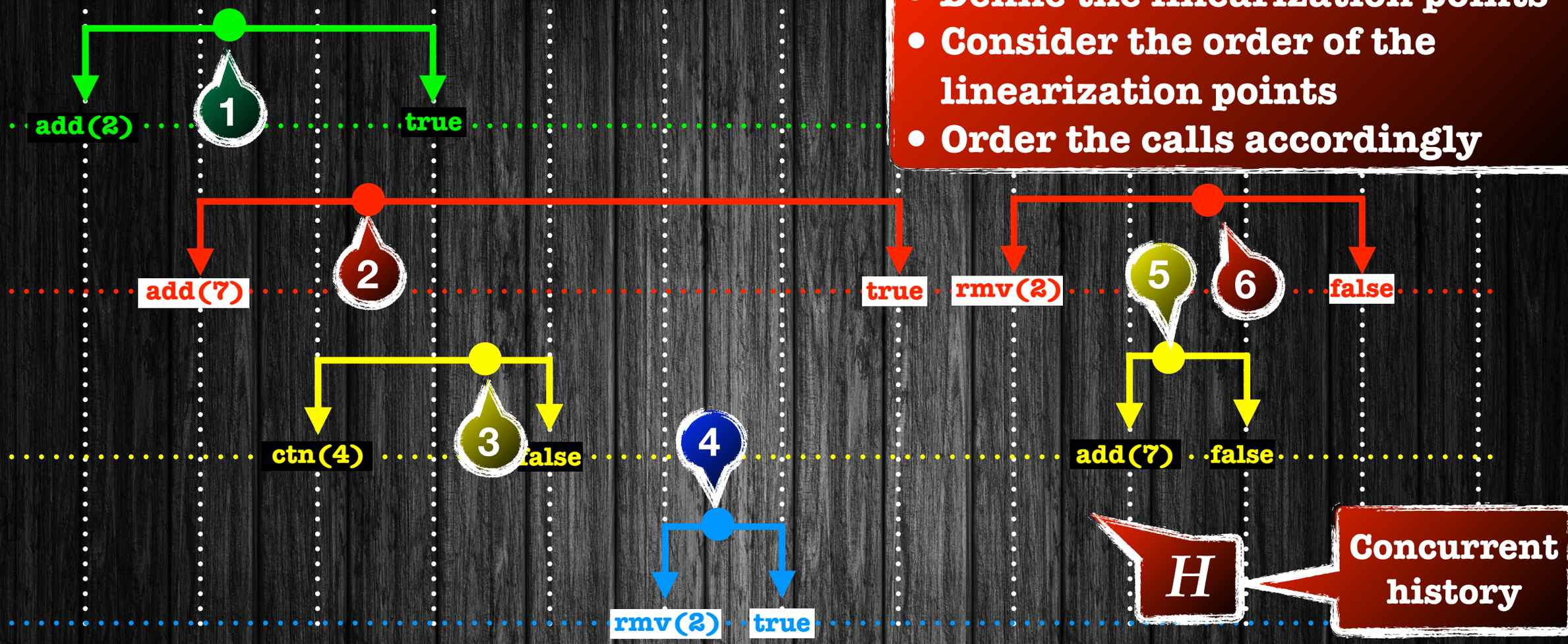
add(7) false

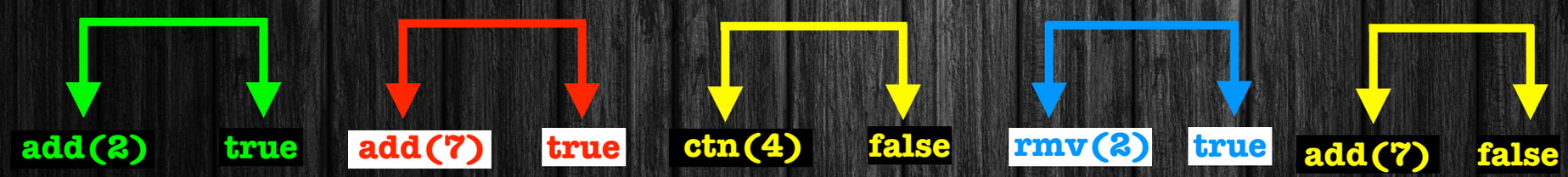
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



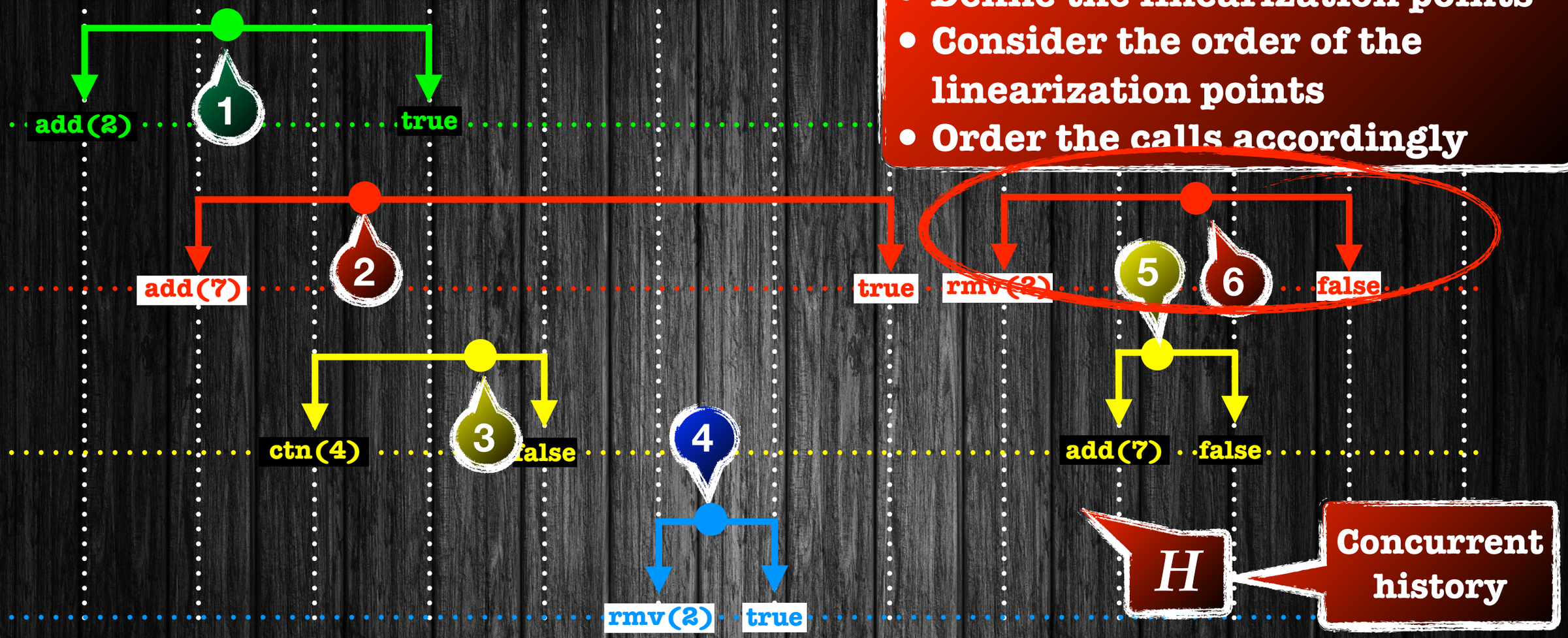


Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



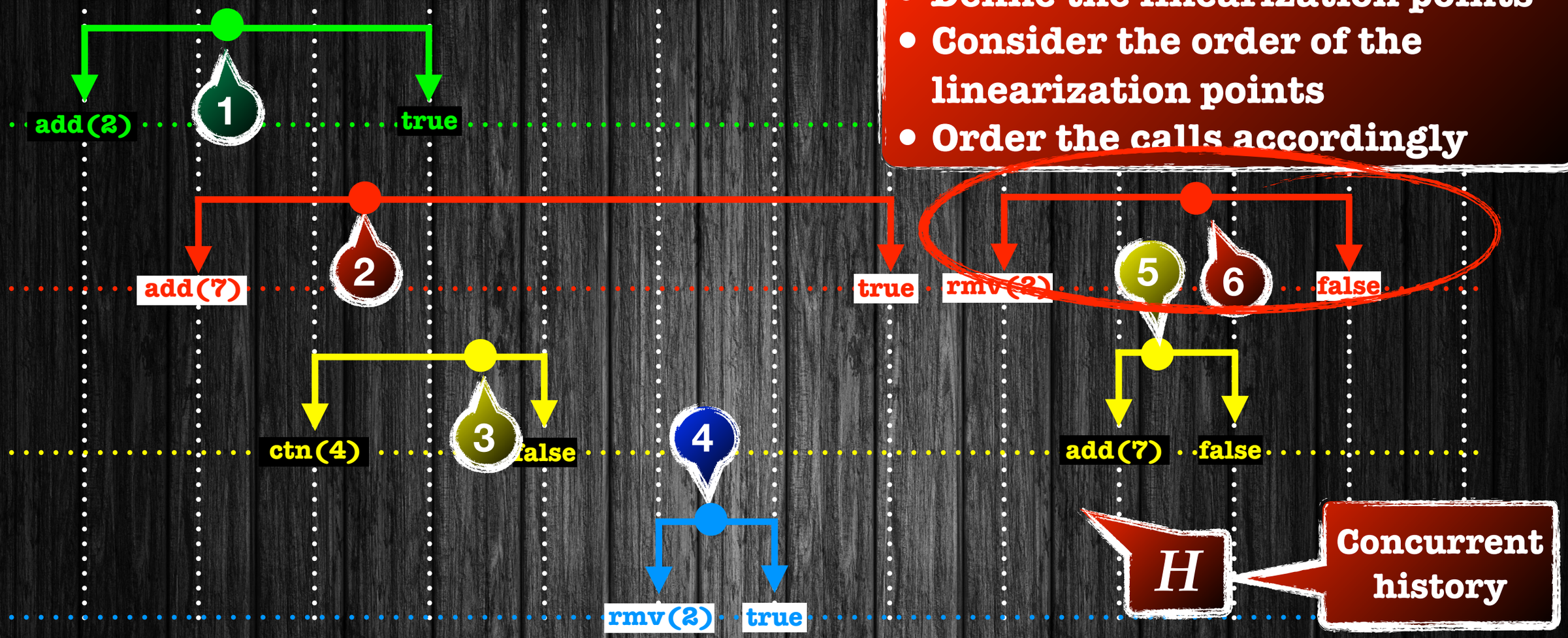
add(2) true add(7) true ctn(4) false rmv(2) true add(7) false

Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history

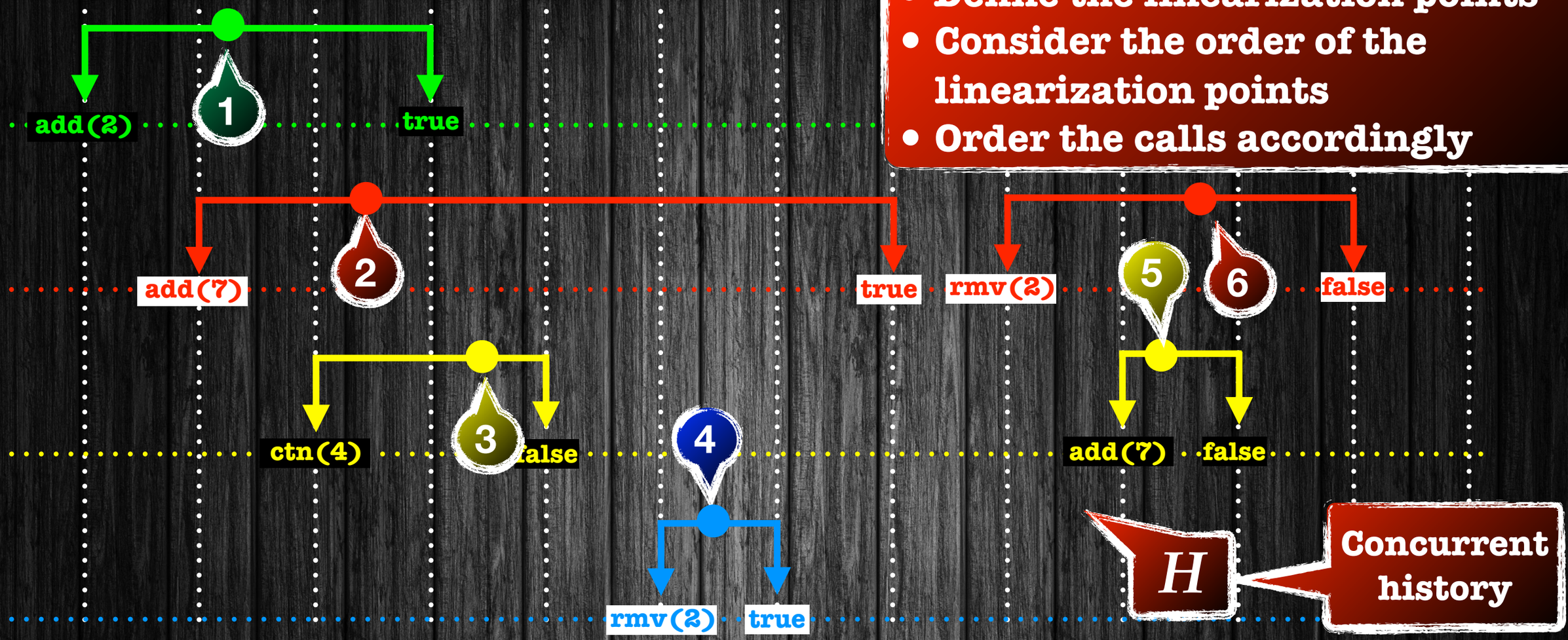
add(2) true | add(7) true | ctn(4) false | rmv(2) true | add(7) false | rmv(2) false

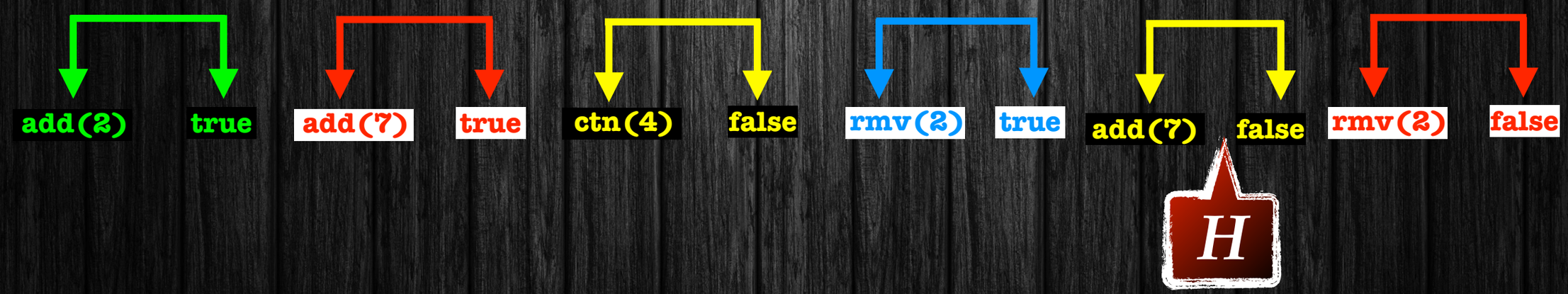
Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



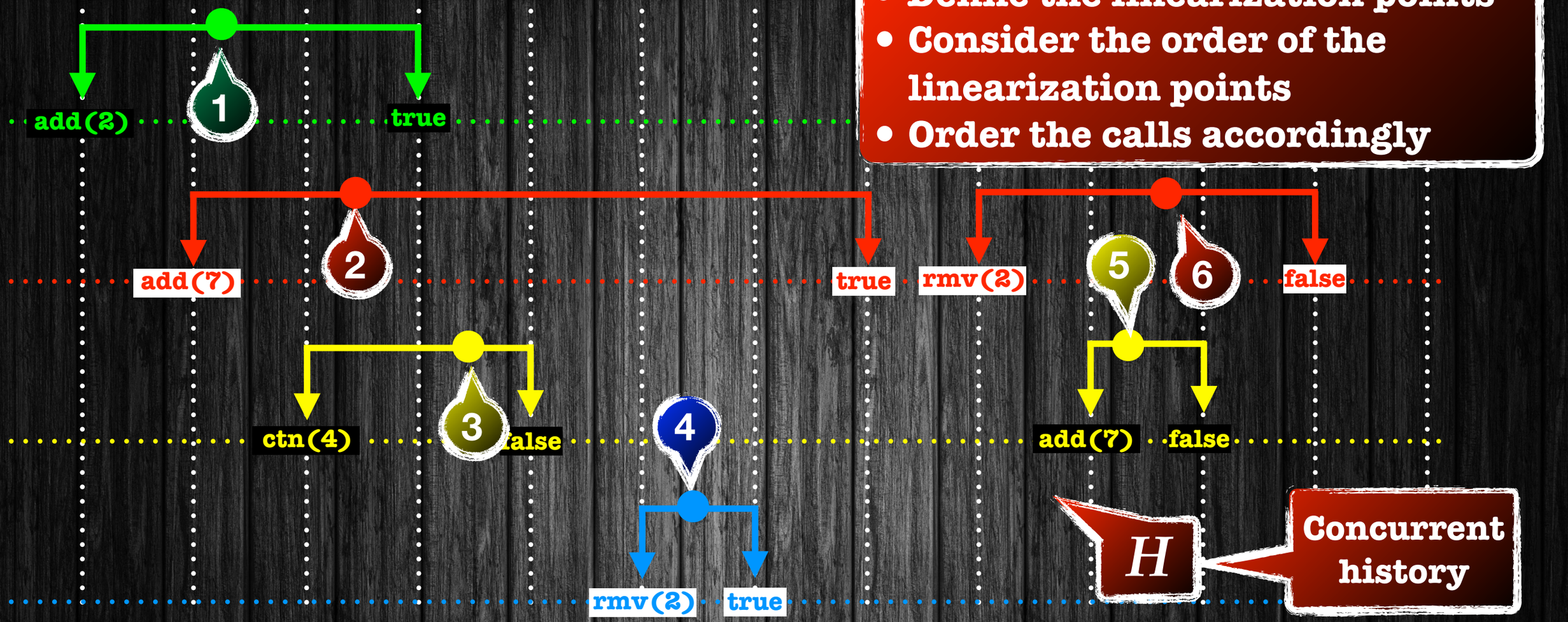


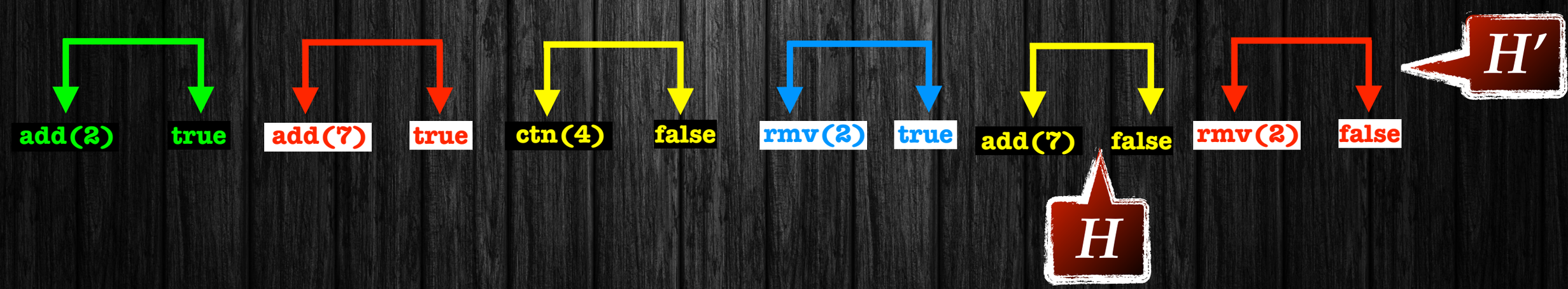
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



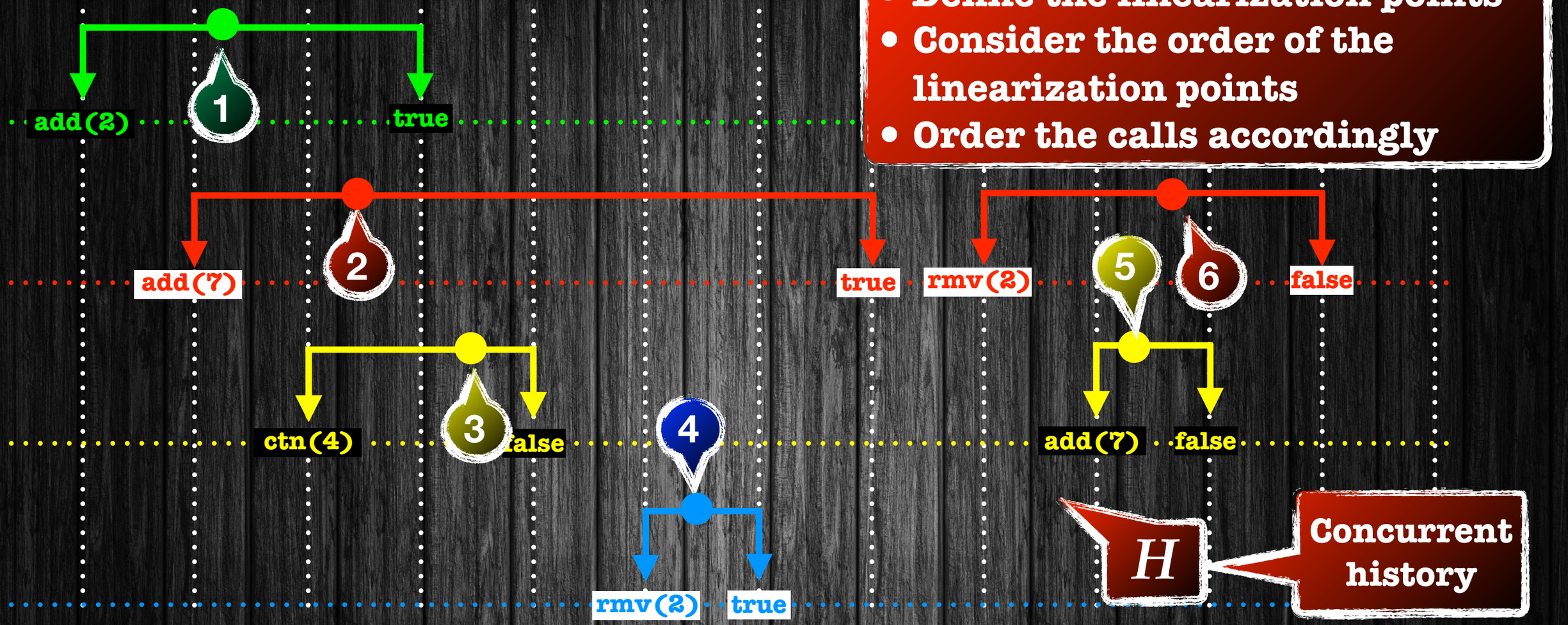


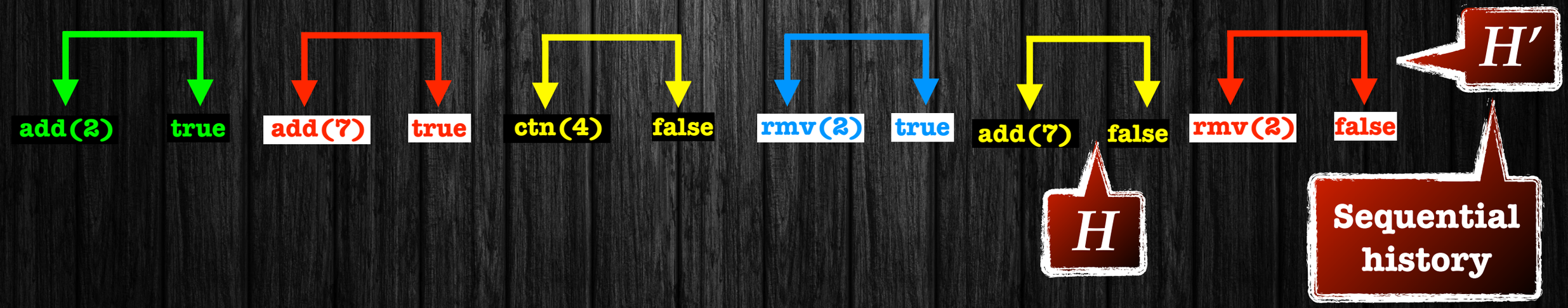
Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



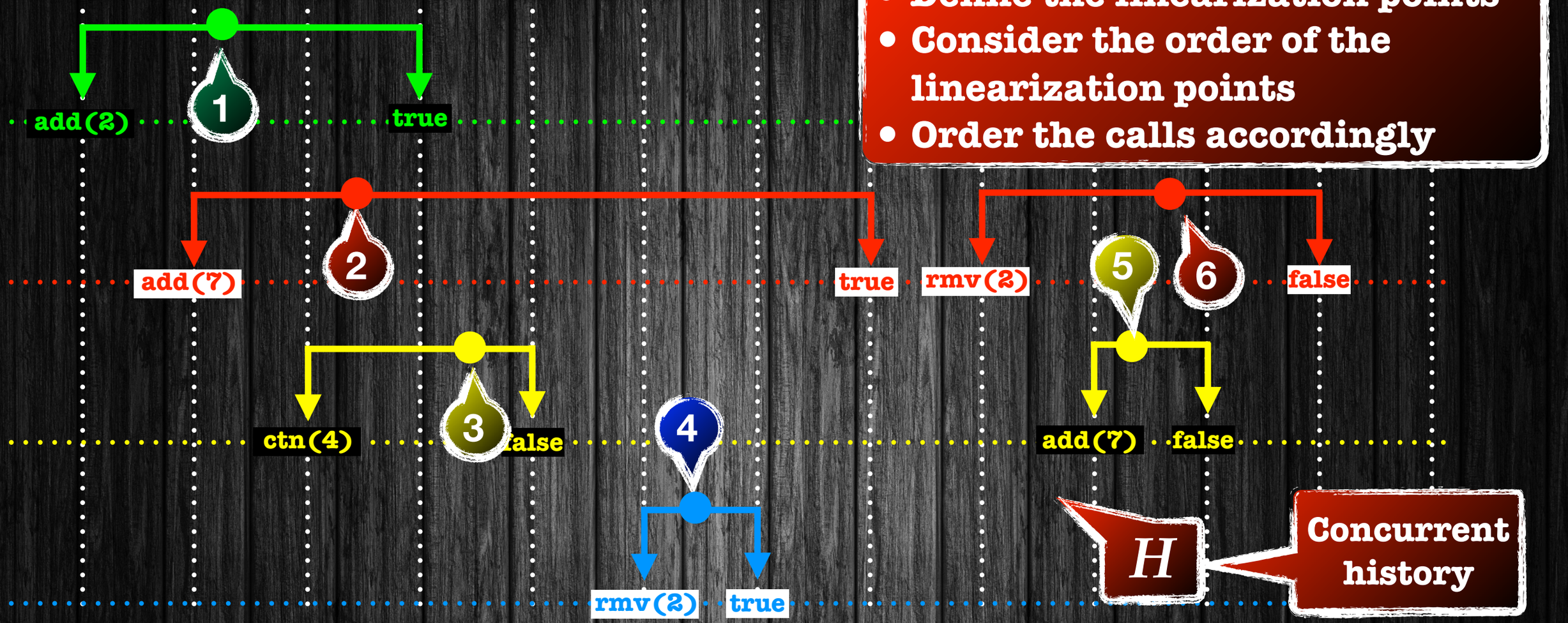


Concurrent Histories

Linearizabilitv

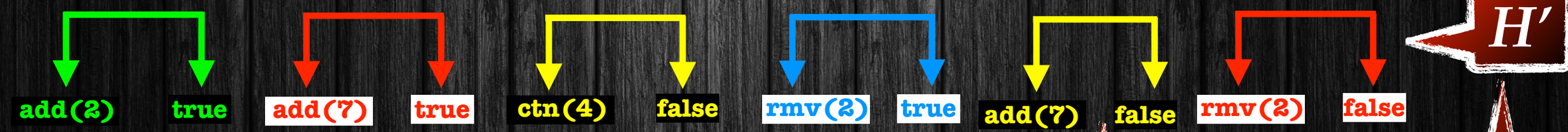
Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history



H′

Sequential history

H′ ∈ linearizations(*H*)

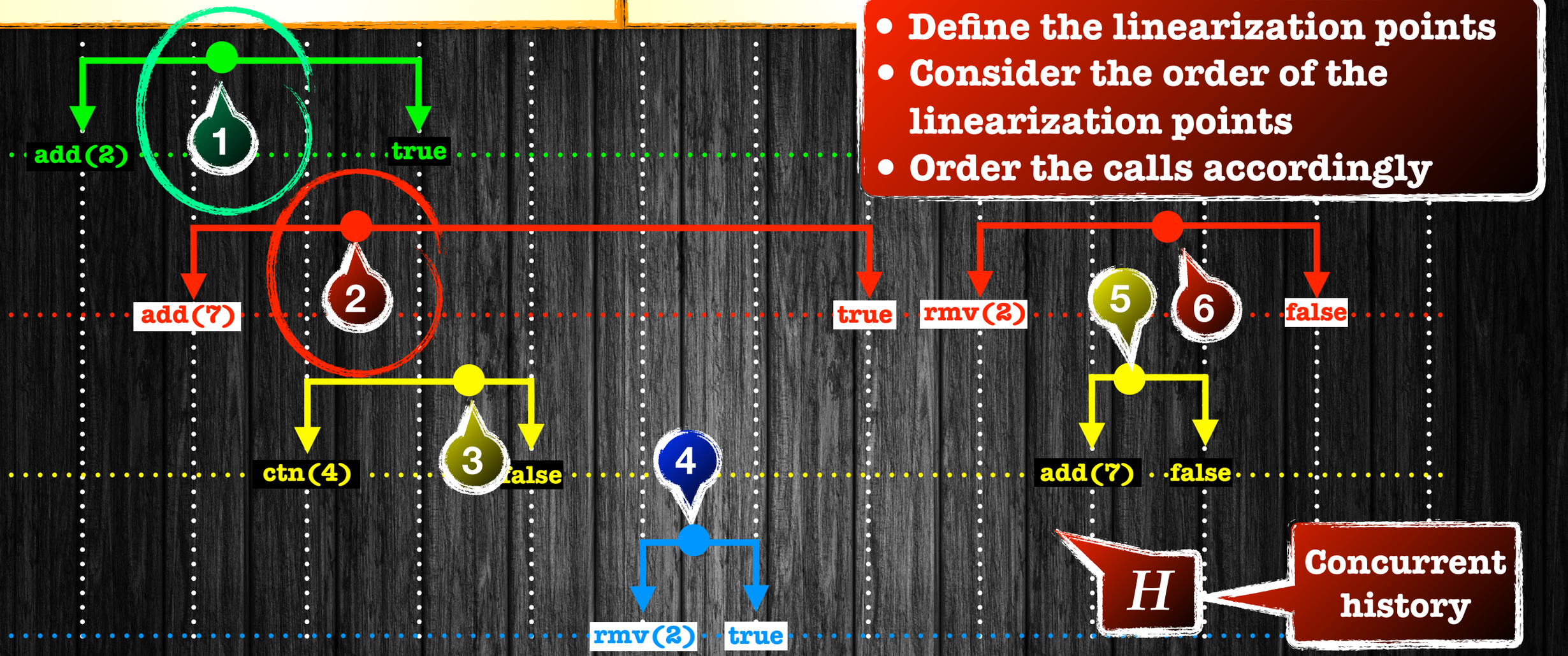
H

Concurrent Histories

Linearizabilitv

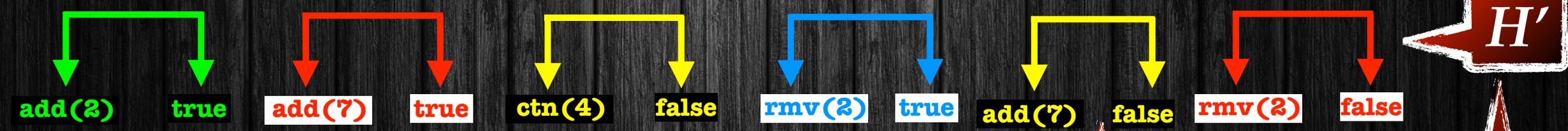
Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history



H′

H′ ∈ linearizations(*H*)

H

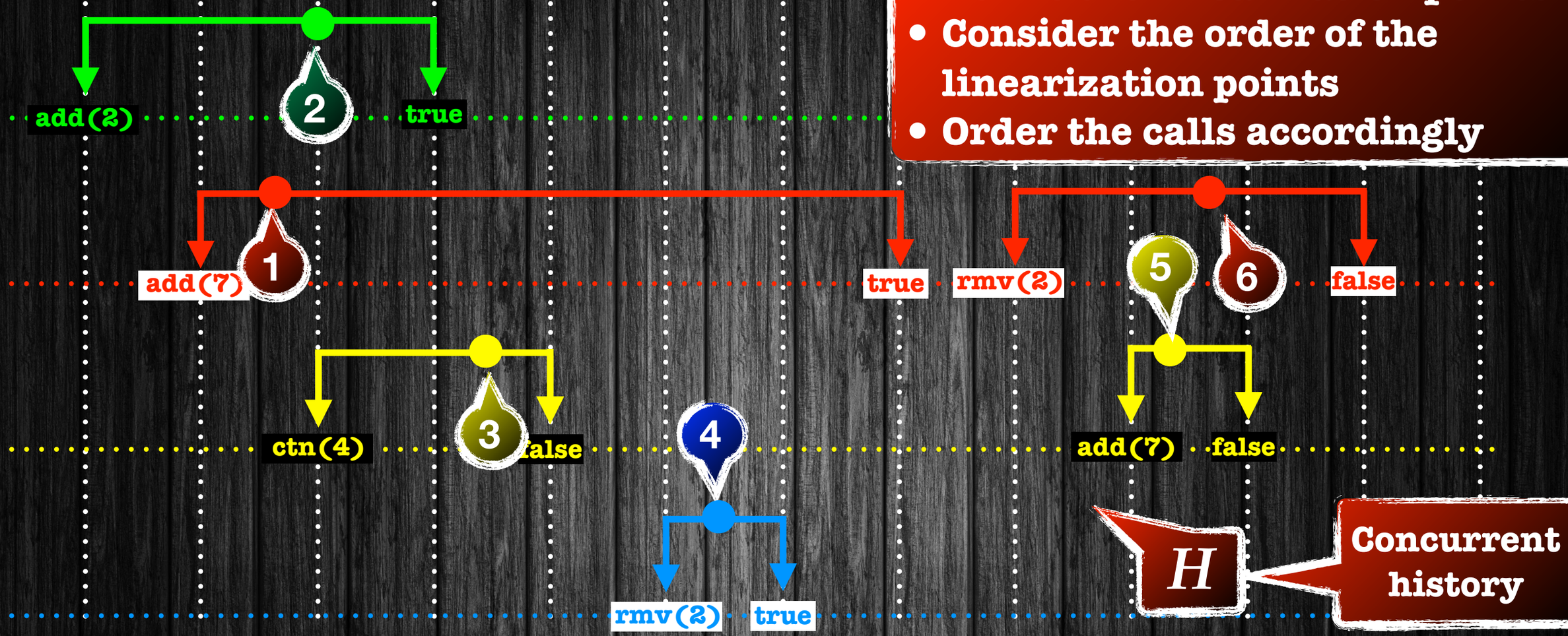
Sequential history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history



H′

Sequential history

$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 2 true

add(7) 1 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

H′

Sequential history

$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 2 true

add(7) 1 true

ctn(4) 3 false

4

rmv(2) 5 6 false

add(7) false

rmv(2) true

H

Concurrent history

add(7) true

add(2) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

H″

Sequential history

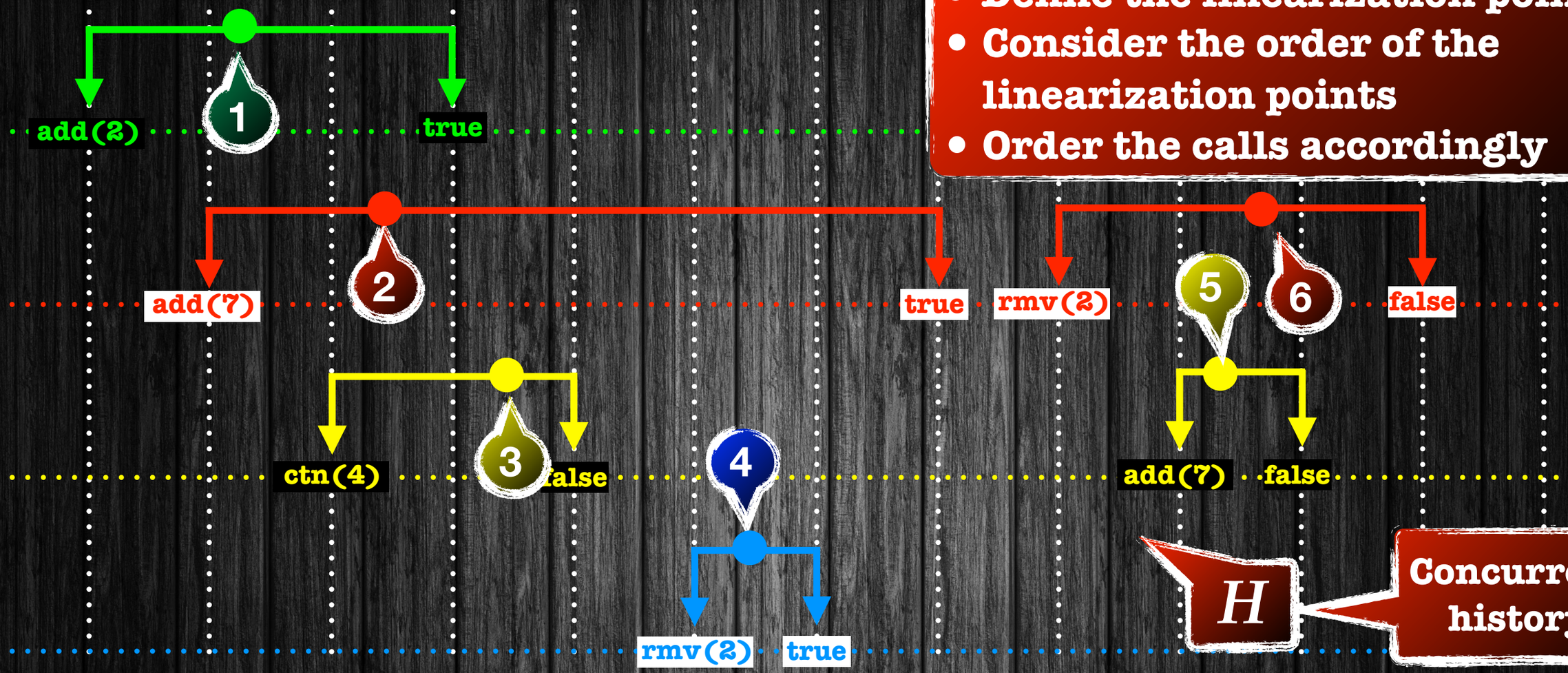
$H'' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

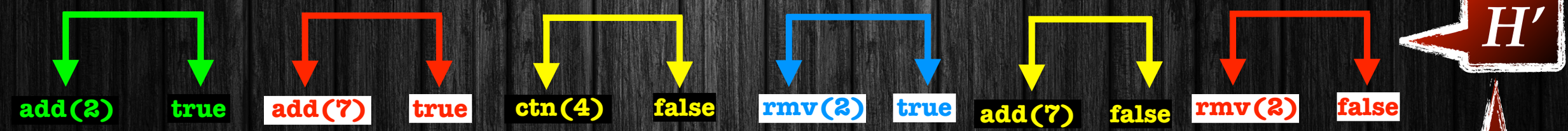
Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history



H′

Sequential history

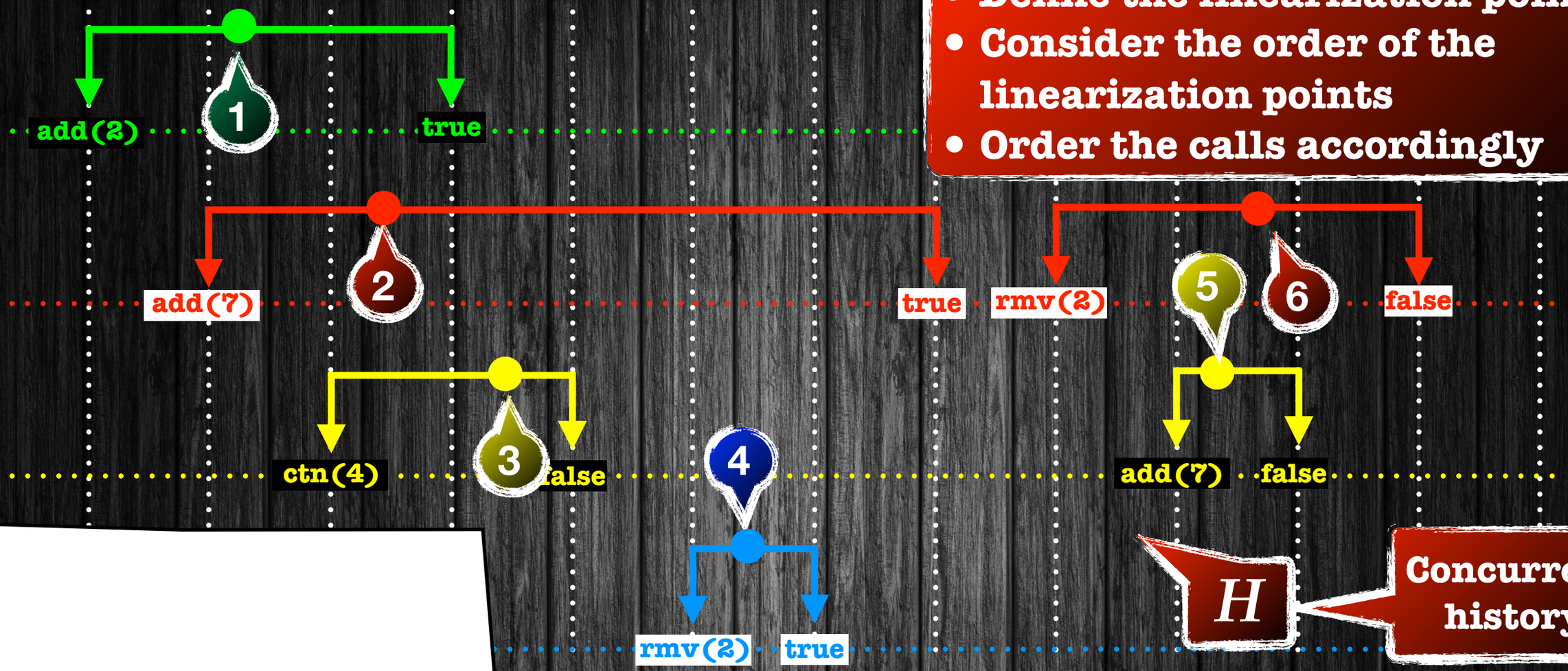
$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

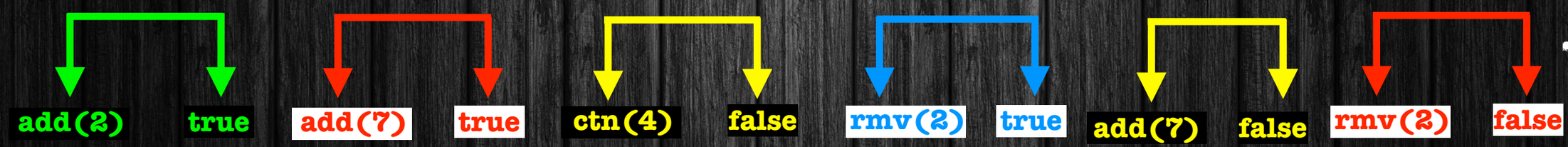
Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



H

Concurrent history



H′

Sequential history

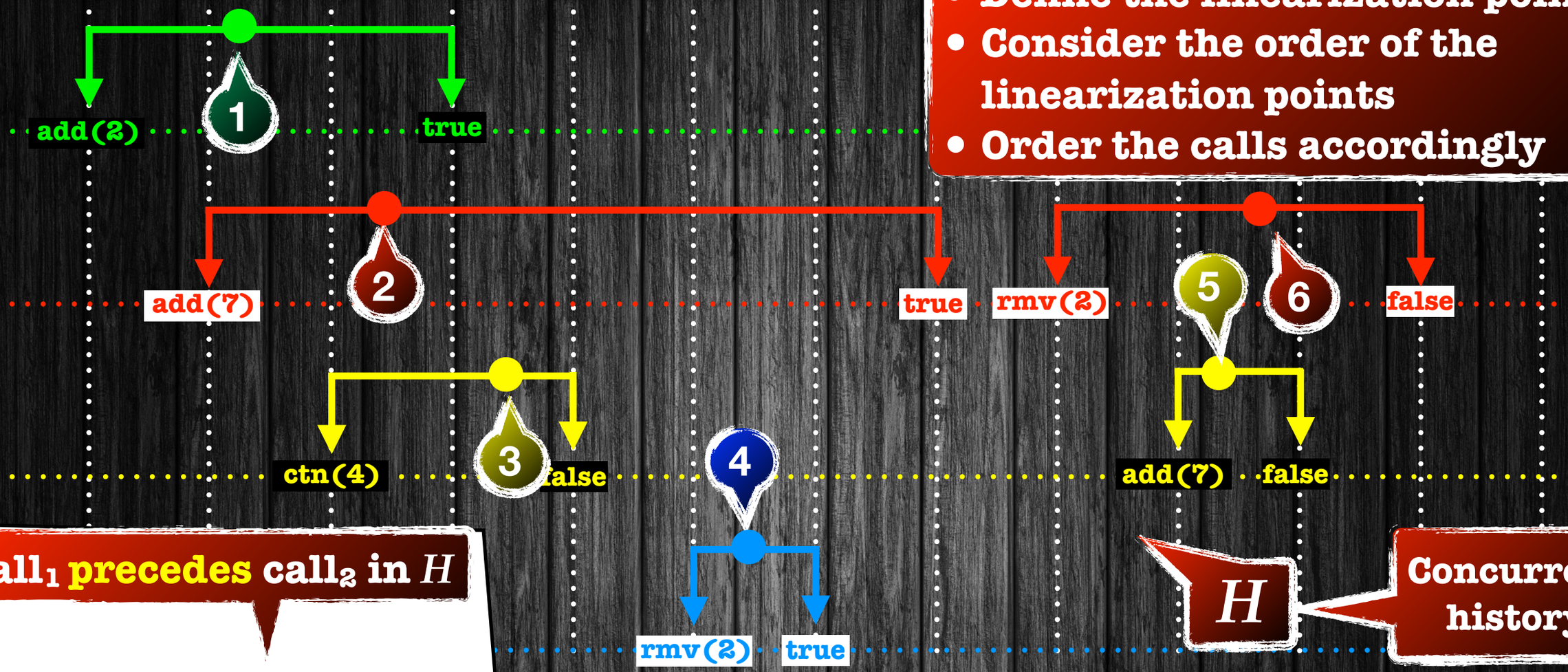
$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

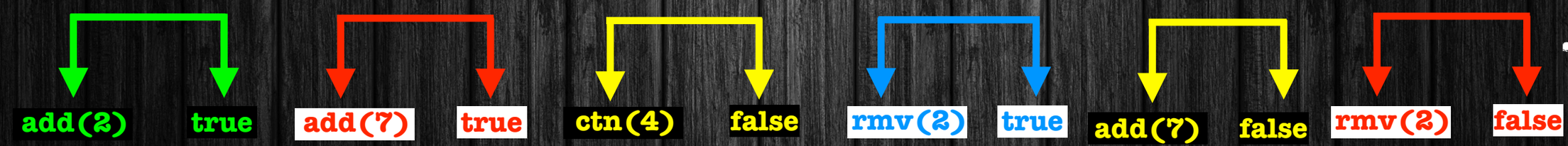
- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly



$call_1$ **precedes** $call_2$ **in** H

H

Concurrent history



H'

Sequential history

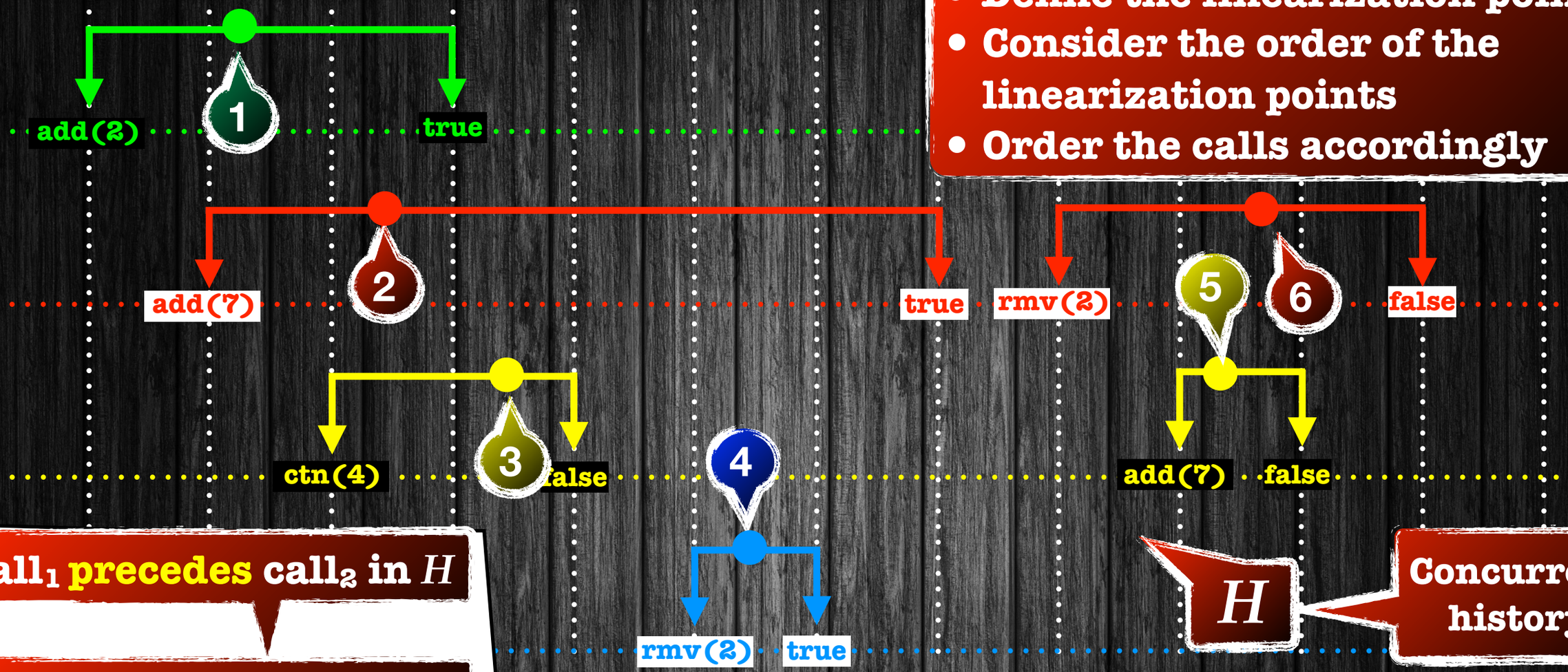
$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

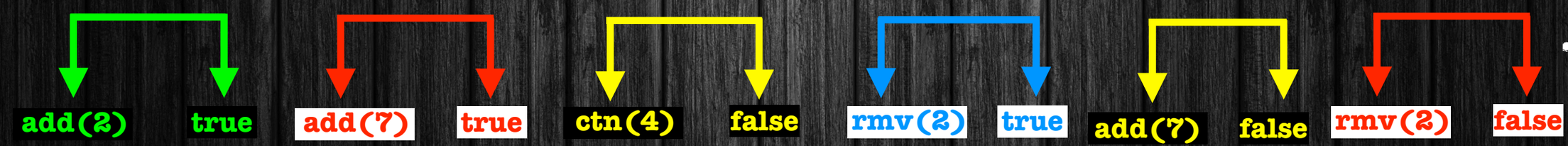


$call_1$ **precedes** $call_2$ **in** H

$call_1$ **precedes** $call_2$ **in** H'

H

Concurrent history



$H' \in$ linearizations(H)

H'

Sequential history

Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

call_1 precedes call_2 in H

call_1 precedes call_2 in H'

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

H'

Sequential history

$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizability

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

call_1 precedes call_2 in H

call_1 precedes call_2 in H'

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

H'

Sequential history

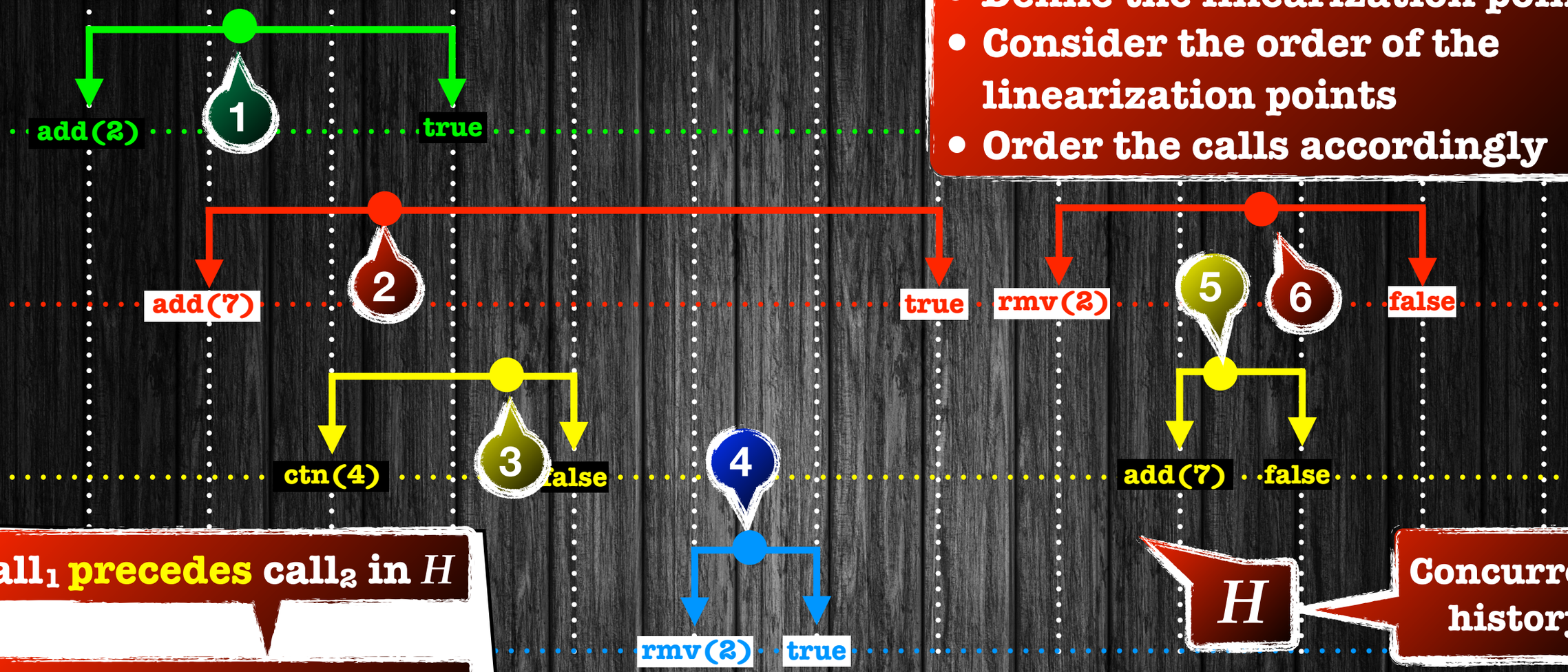
$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

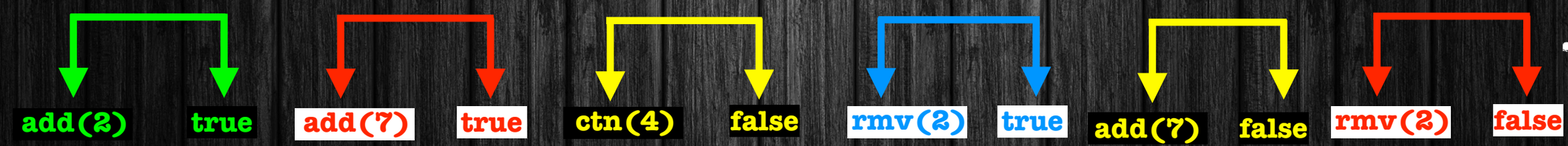


$call_1$ **precedes** $call_2$ **in** H

$call_1$ **precedes** $call_2$ **in** H'

H

Concurrent history



$H' \in$ linearizations(H)

H'

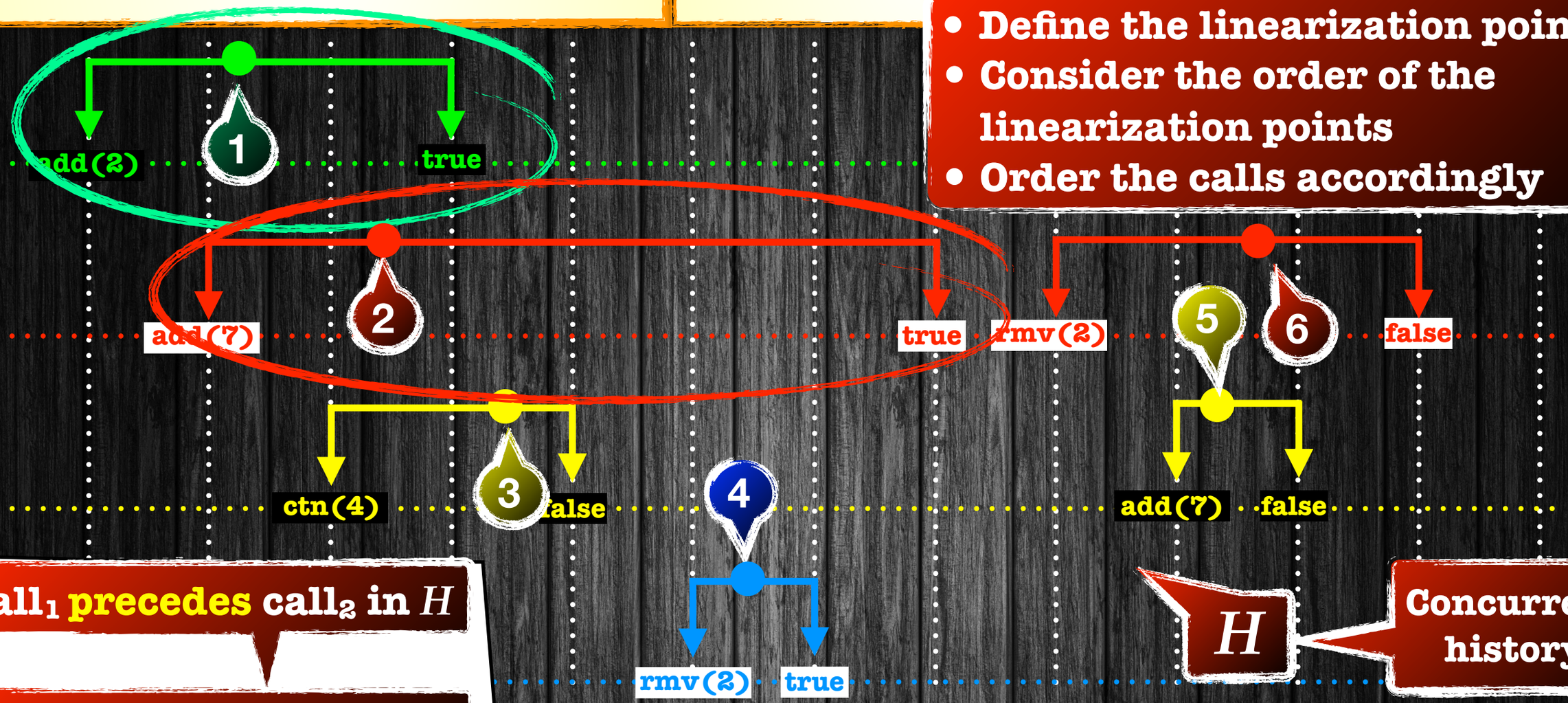
Sequential history

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

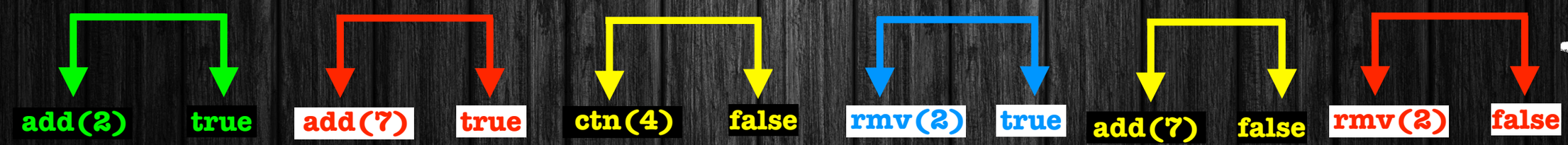


$call_1$ **precedes** $call_2$ **in** H

$call_1$ **precedes** $call_2$ **in** H'

H

Concurrent history



H'

Sequential history

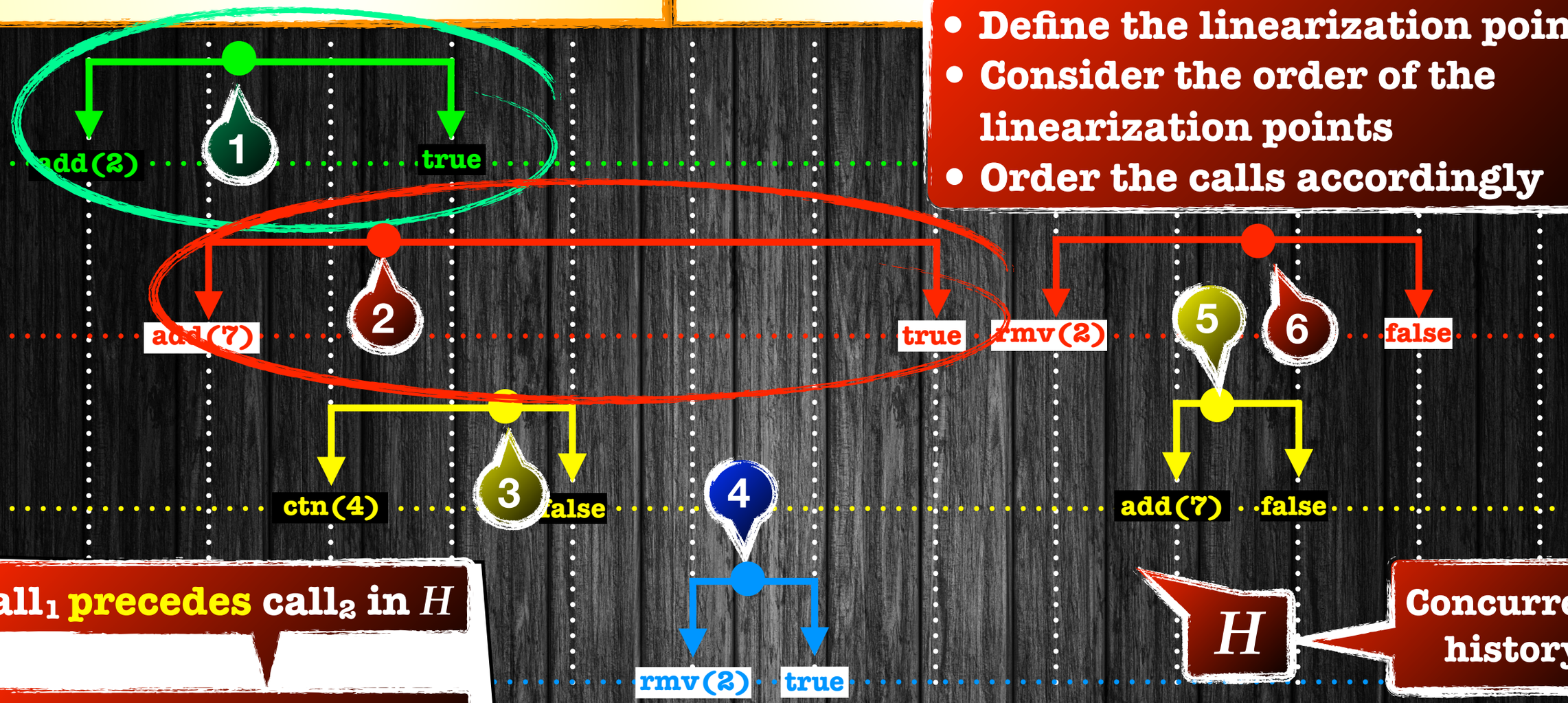
$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

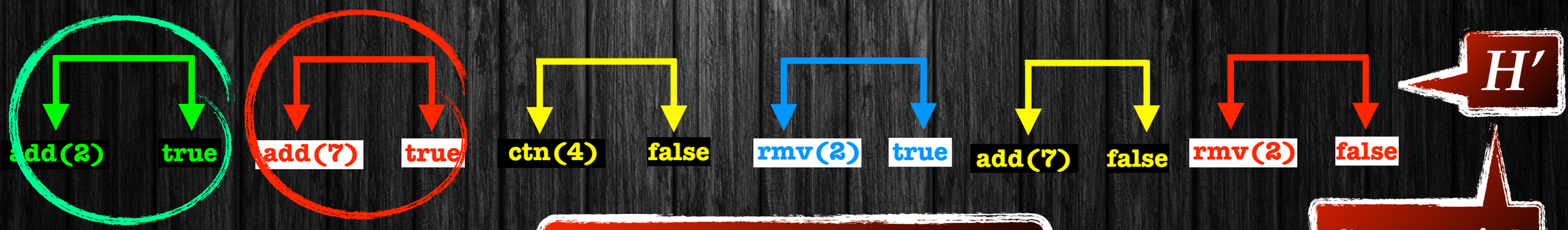


H

Concurrent history

$call_1$ precedes $call_2$ in H

$call_1$ precedes $call_2$ in H'



H'

Sequential history

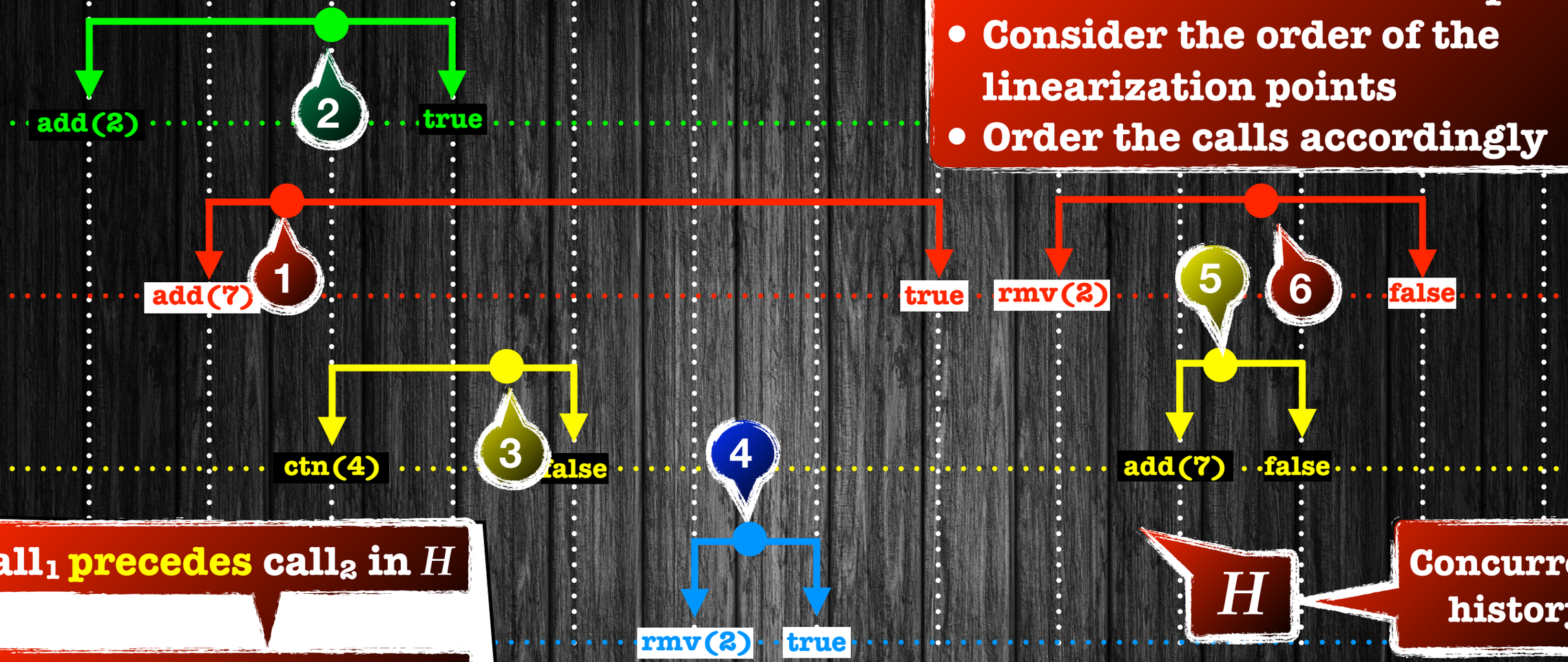
$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

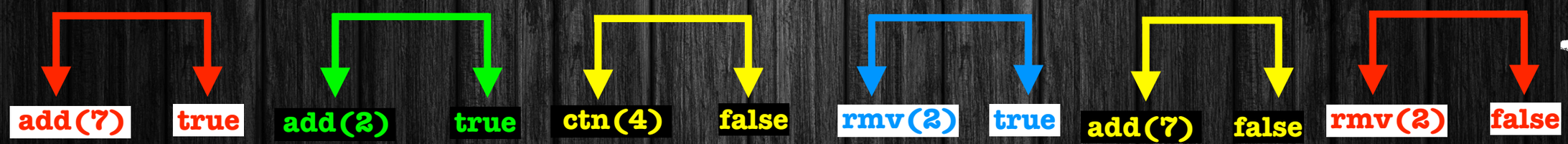


$call_1$ **precedes** $call_2$ **in** H

$call_1$ **precedes** $call_2$ **in** H'

H

Concurrent history



H'

Sequential history

$H' \in \text{linearizations}(H)$

Concurrent Histories

Linearizabilitv

Linearization Operation

- Define the linearization points
- Consider the order of the linearization points
- Order the calls accordingly

add(2) 2 true

add(7) 1 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

call_1 precedes call_2 in H

call_1 precedes call_2 in H'

add(7) true

add(2) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

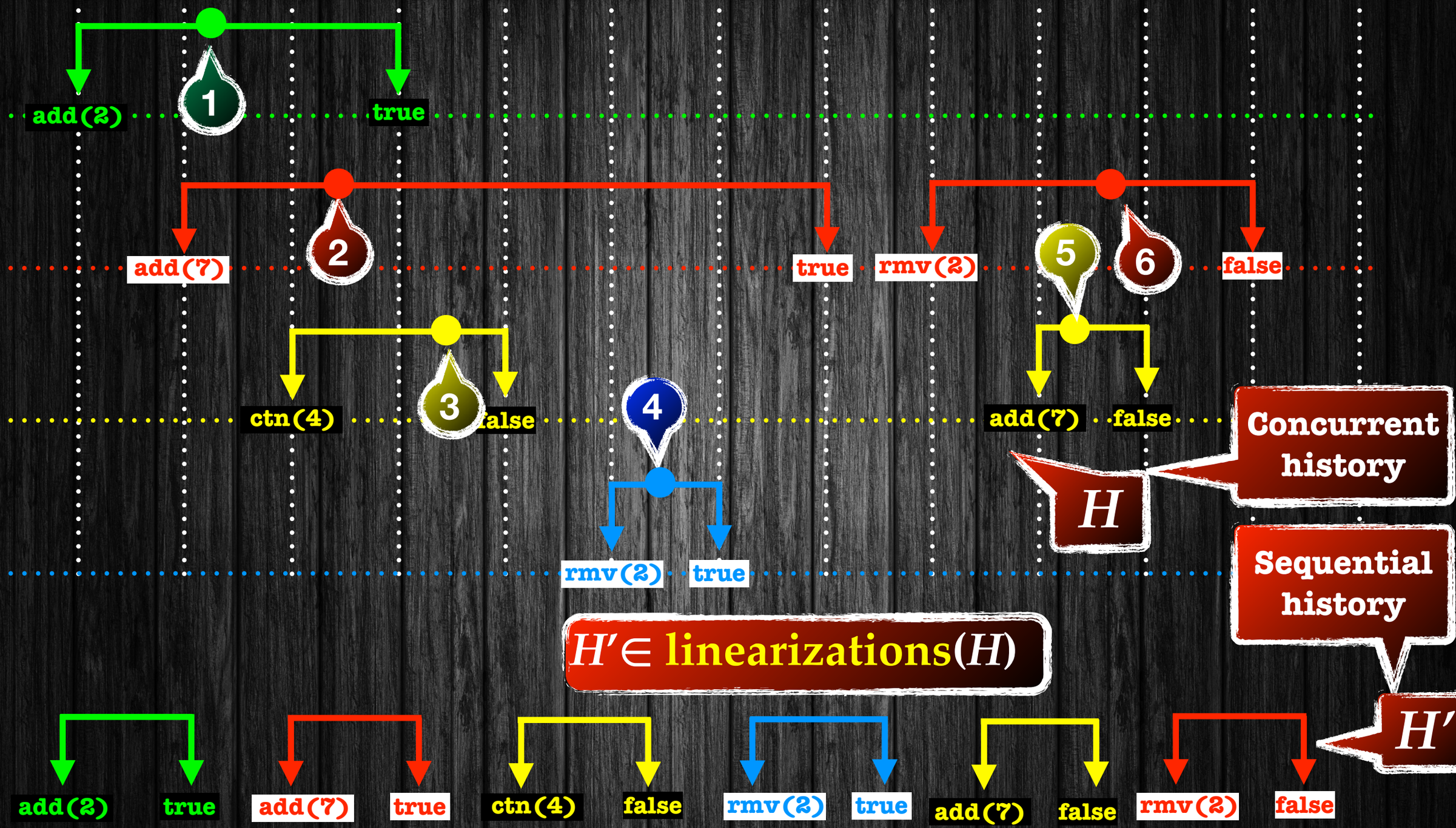
H'

Sequential history

$H' \in \text{linearizations}(H)$

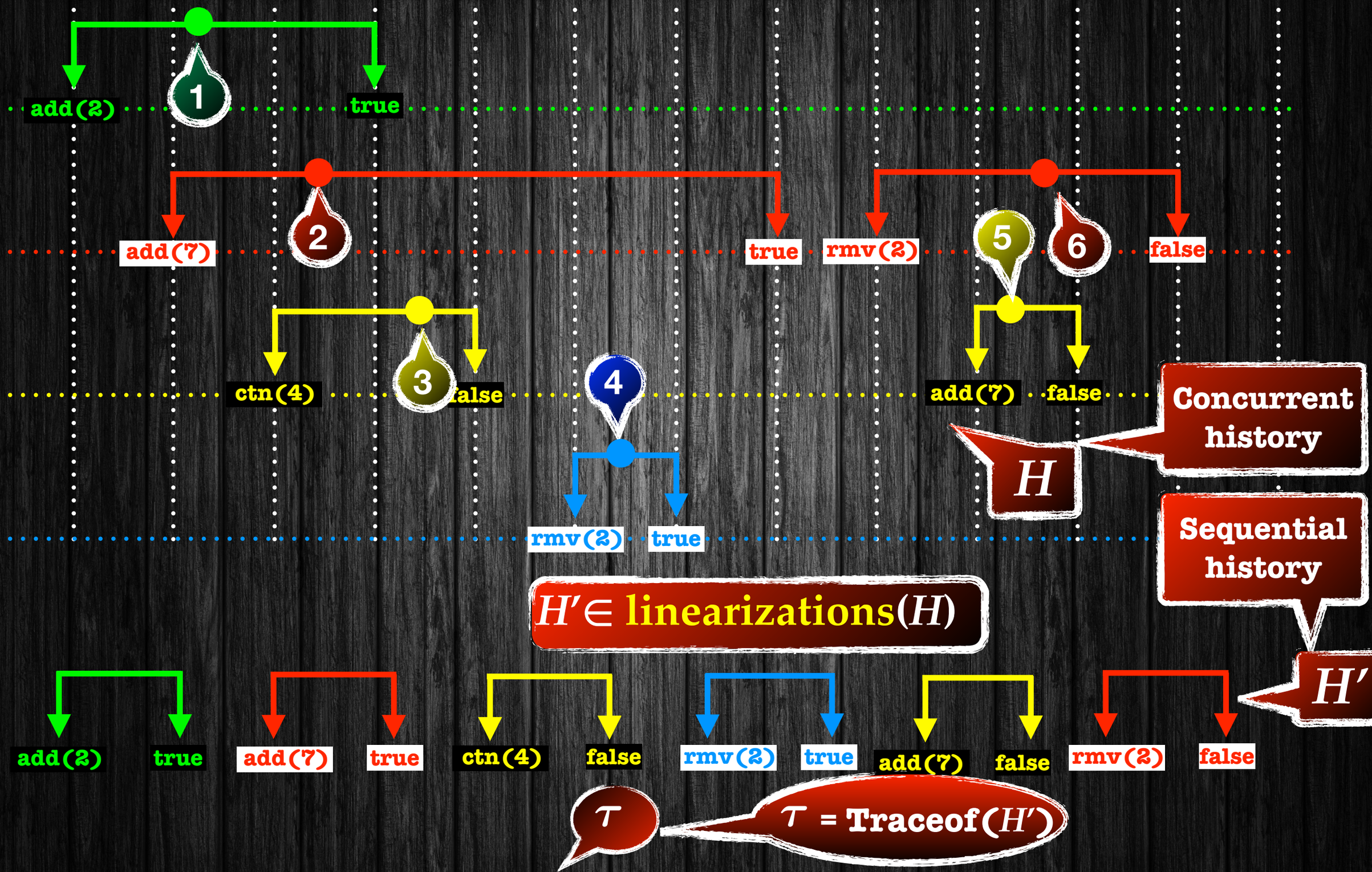
Concurrent Histories

Linearizability



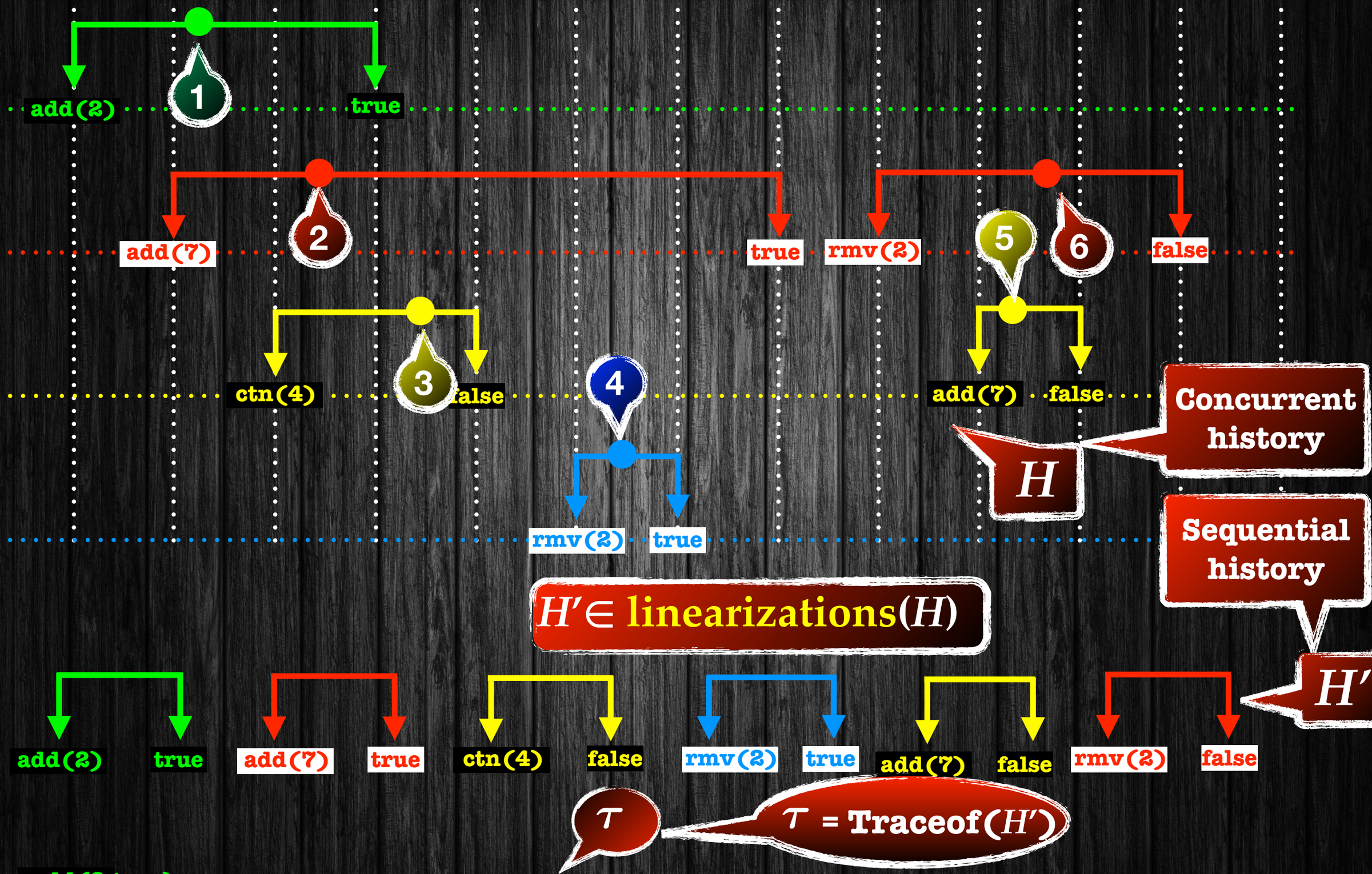
Concurrent Histories

Linearizability



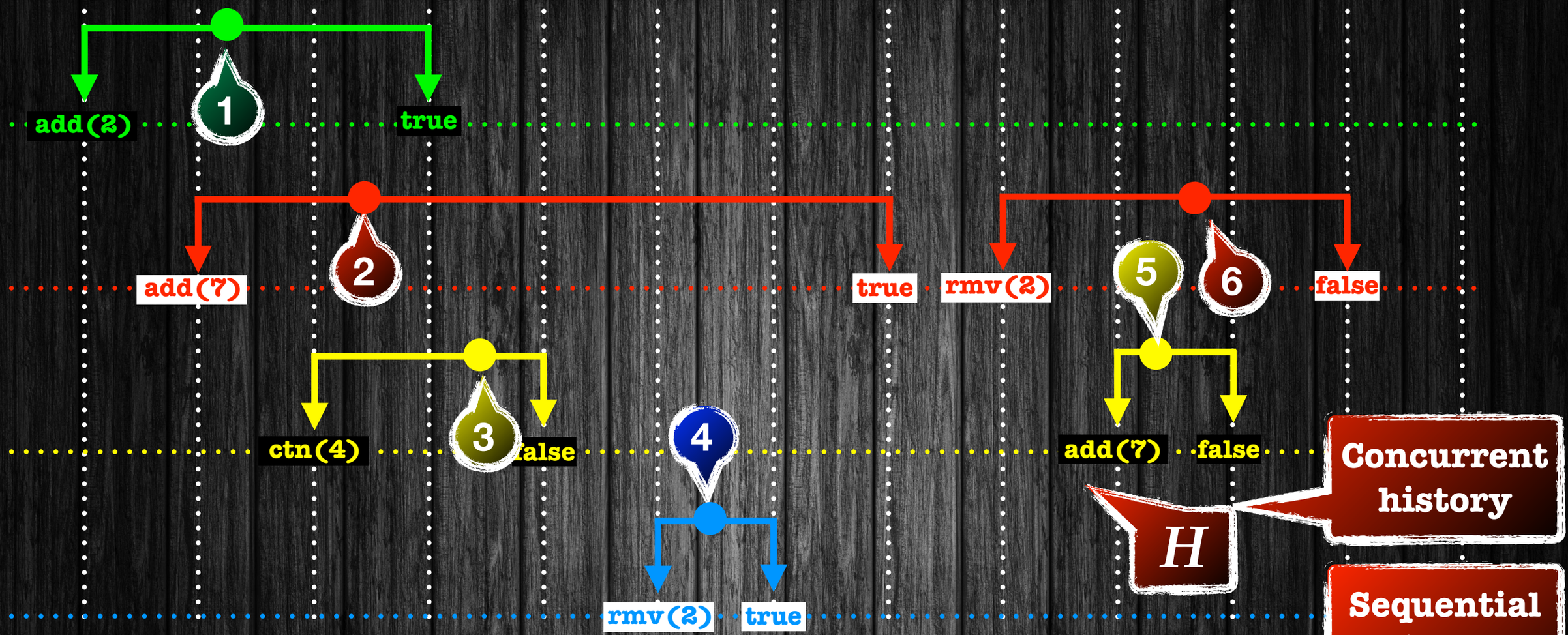
Concurrent Histories

Linearizability



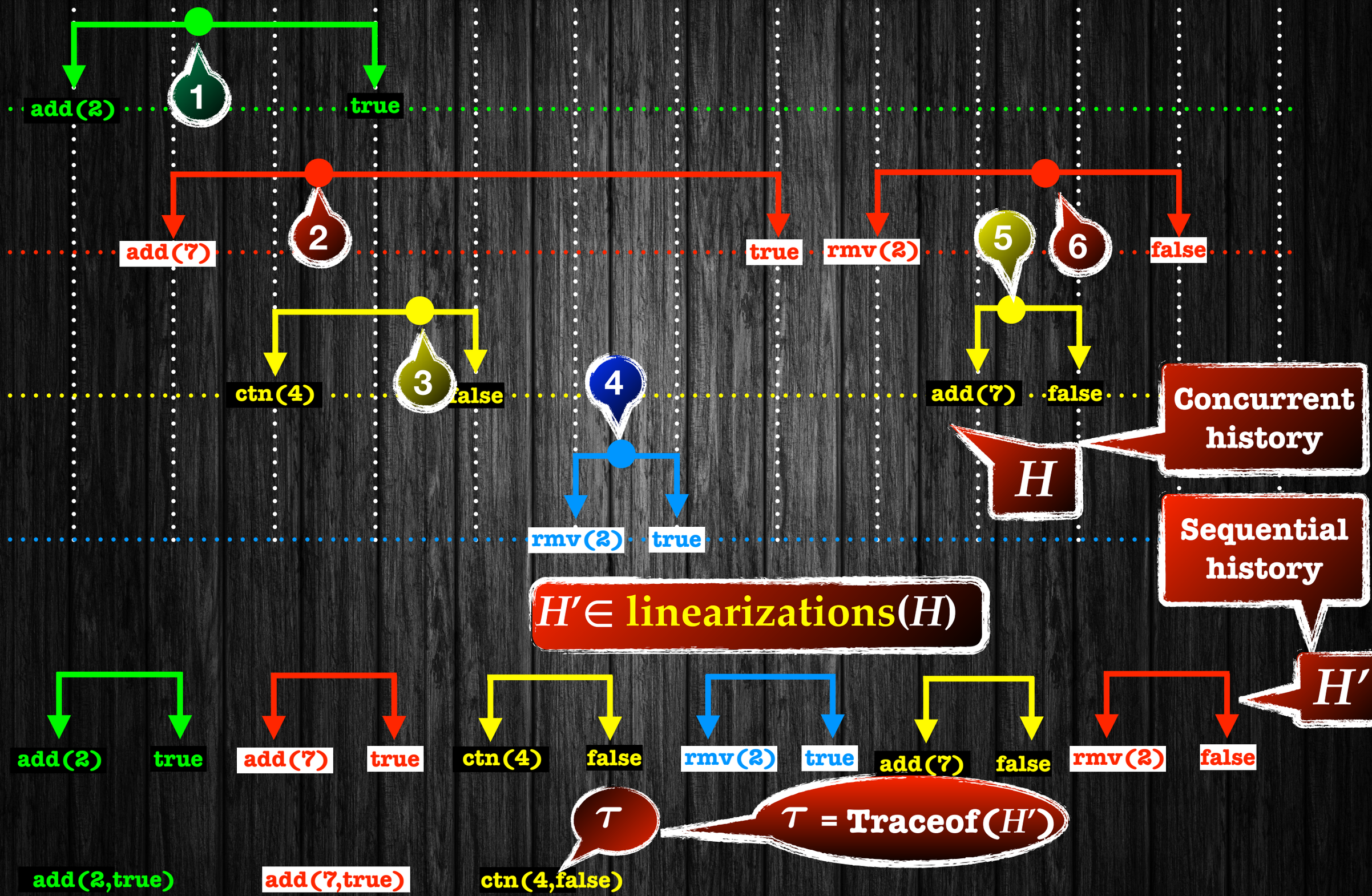
Concurrent Histories

Linearizability



Concurrent Histories

Linearizability



Concurrent Histories

Linearizability

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

Concurrent history

H

Sequential history

H′

H′ ∈ linearizations(*H*)

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

τ

τ = Traceof(*H′*)

add(2,true)

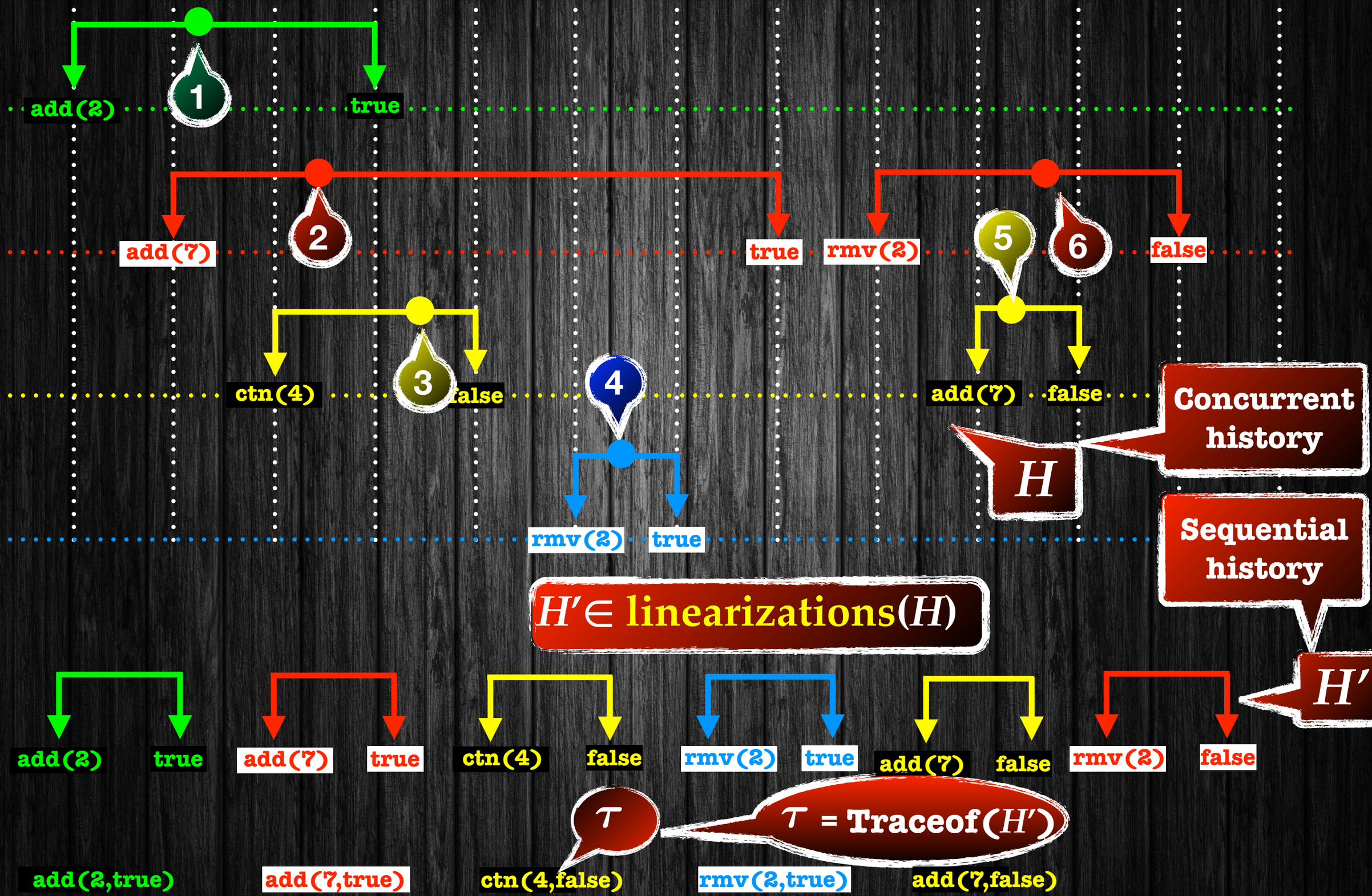
add(7,true)

ctn(4,false)

rmv(2,true)

Concurrent Histories

Linearizability



Concurrent Histories

Linearizability

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

Concurrent history

H

Sequential history

H′

$H' \in$ linearizations(H)

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

τ

τ = Traceof(H')

add(2,true)

add(7,true)

ctn(4,false)

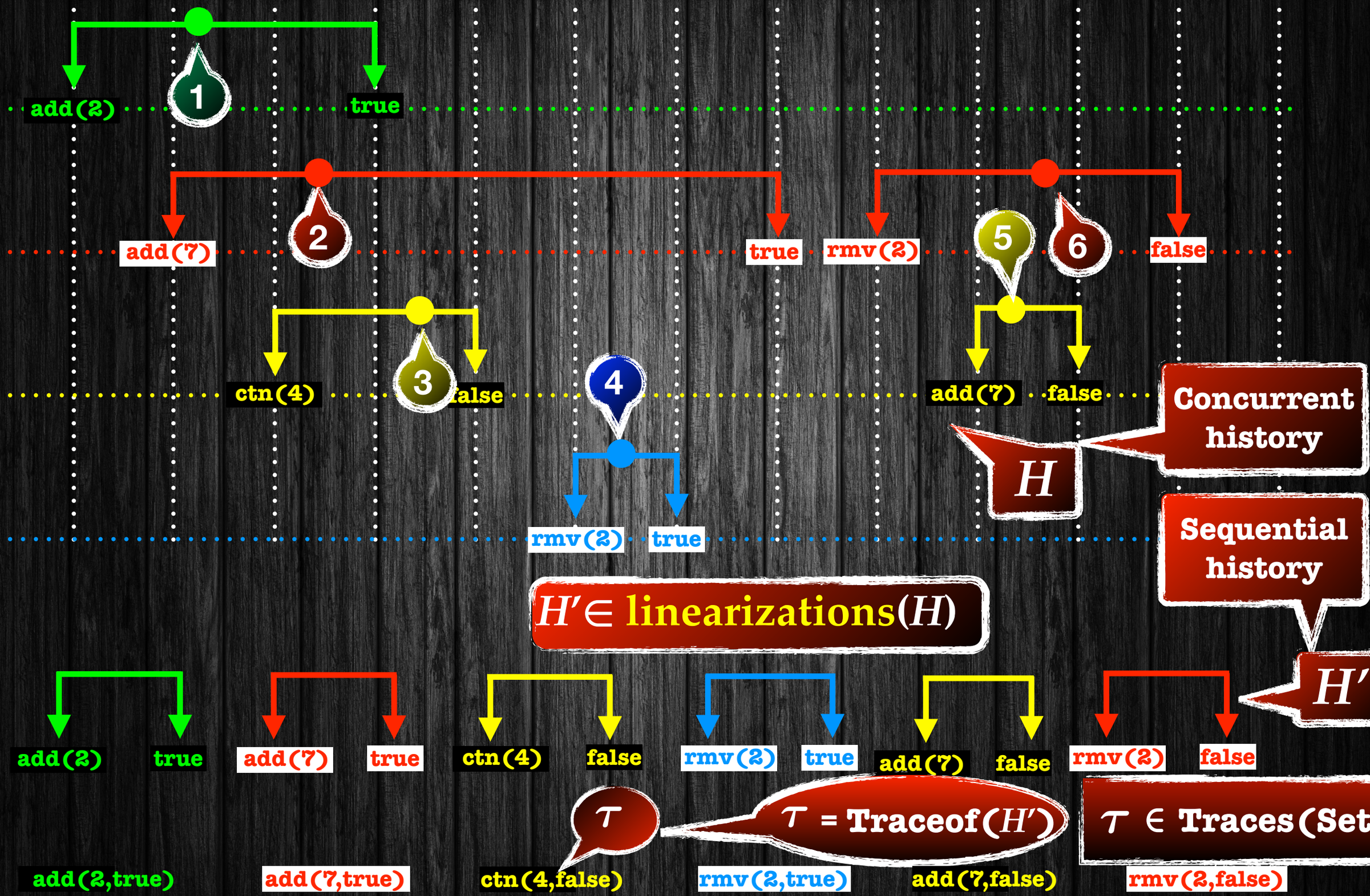
rmv(2,true)

add(7,false)

rmv(2,false)

Concurrent Histories

Linearizability



Concurrent Histories

Linearizability

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

Concurrent history

H

Sequential history

H′

$H' \in$ linearizations(H)

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

add(2,true)

add(7,true)

ctn(4,false)

rmv(2,true)

add(7,false)

rmv(2,false)

τ

τ = Traceof(H')

$\tau \in$ Traces(Set)

Concurrent Histories

Linearizability

$H' \in$ Histories (Set)

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

Sequential history

$H' \in$ linearizations(H)

H'

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

τ

τ = Traceof(H')

$\tau \in$ Traces (Set)

add(2,true)

add(7,true)

ctn(4,false)

rmv(2,true)

add(7,false)

rmv(2,false)

Concurrent Histories

Linearizability

$H' \in$ Histories (Set)

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

H

Concurrent history

Sequential history

$H' \in$ linearizations(H)

H'

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

τ

τ = Traceof(H')

$\tau \in$ Traces (Set)

add(2,true)

add(7,true)

ctn(4,false)

rmv(2,true)

add(7,false)

rmv(2,false)

Concurrent Histories

Linearizability

$H'\in$ Histories(Set)

H linearizable wrt. SET

add(2) 1 true

add(7) 2 true

ctn(4) 3 false

4

rmv(2) true

rmv(2) 5 6 false

add(7) false

Concurrent history

H

Sequential history

H'

$H'\in$ linearizations(H)

add(2) true

add(7) true

ctn(4) false

rmv(2) true

add(7) false

rmv(2) false

τ

τ = Traceof(H')

$\tau \in$ Traces(Set)

add(2,true)

add(7,true)

ctn(4,false)

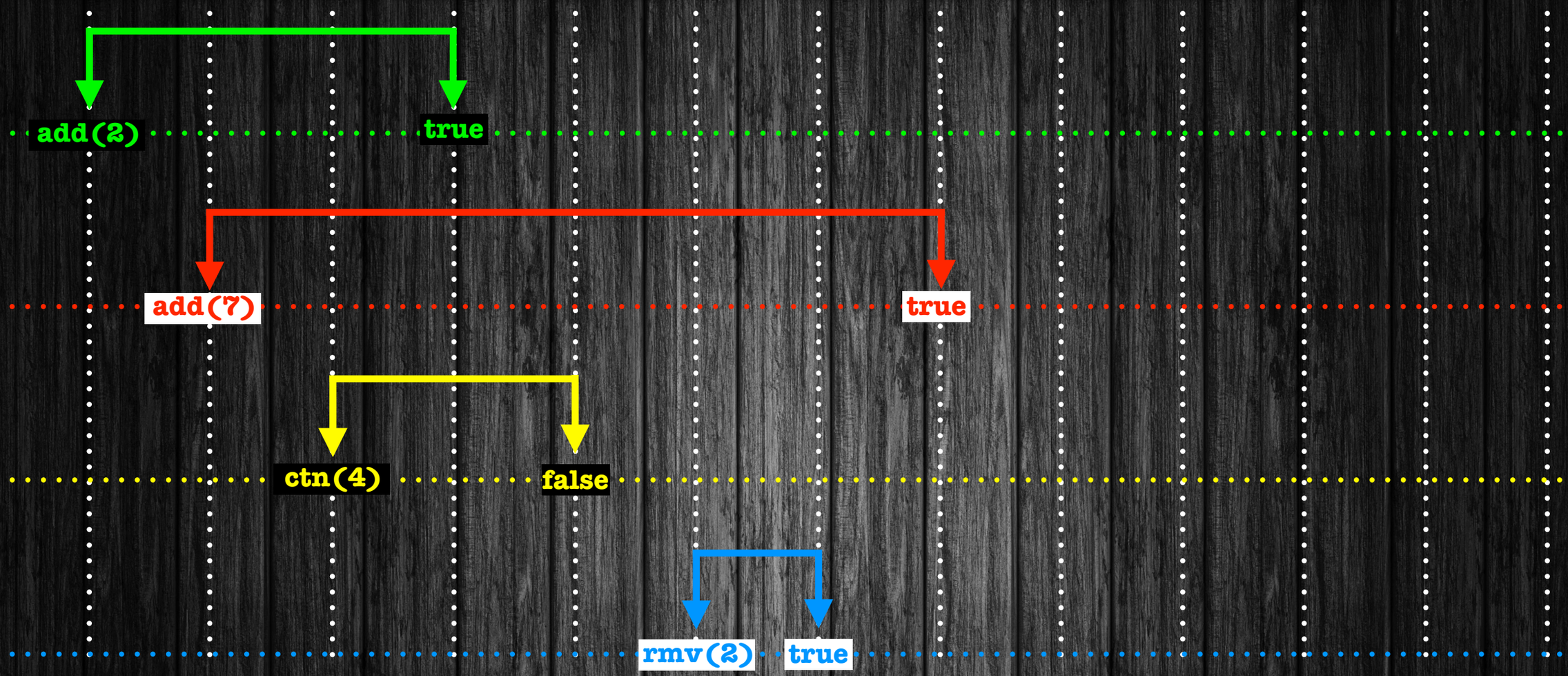
rmv(2,true)

add(7,false)

rmv(2,false)

Concurrent Histories | Linearizability | Sets

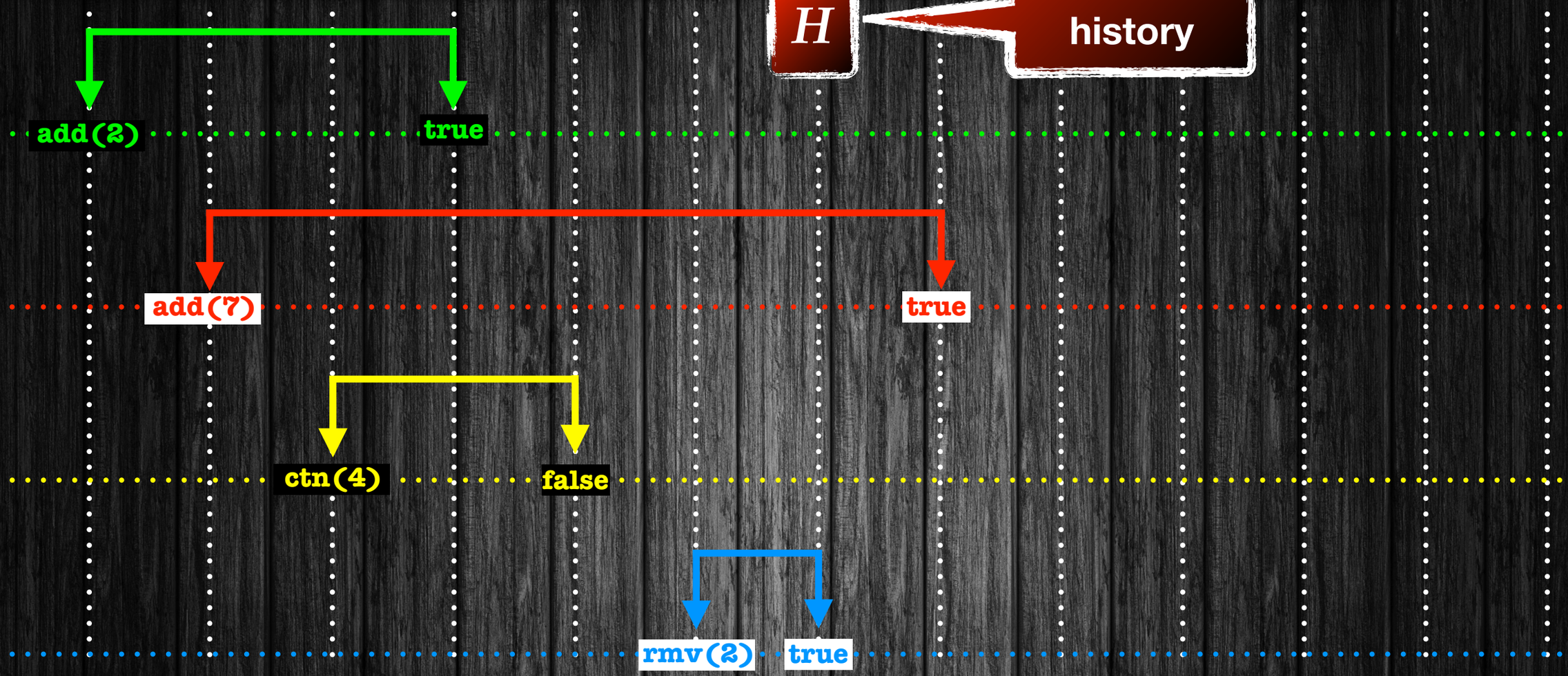
Concurrent Histories | Linearizability | Sets



Concurrent Histories | Linearizability | Sets

H

Concurrent history

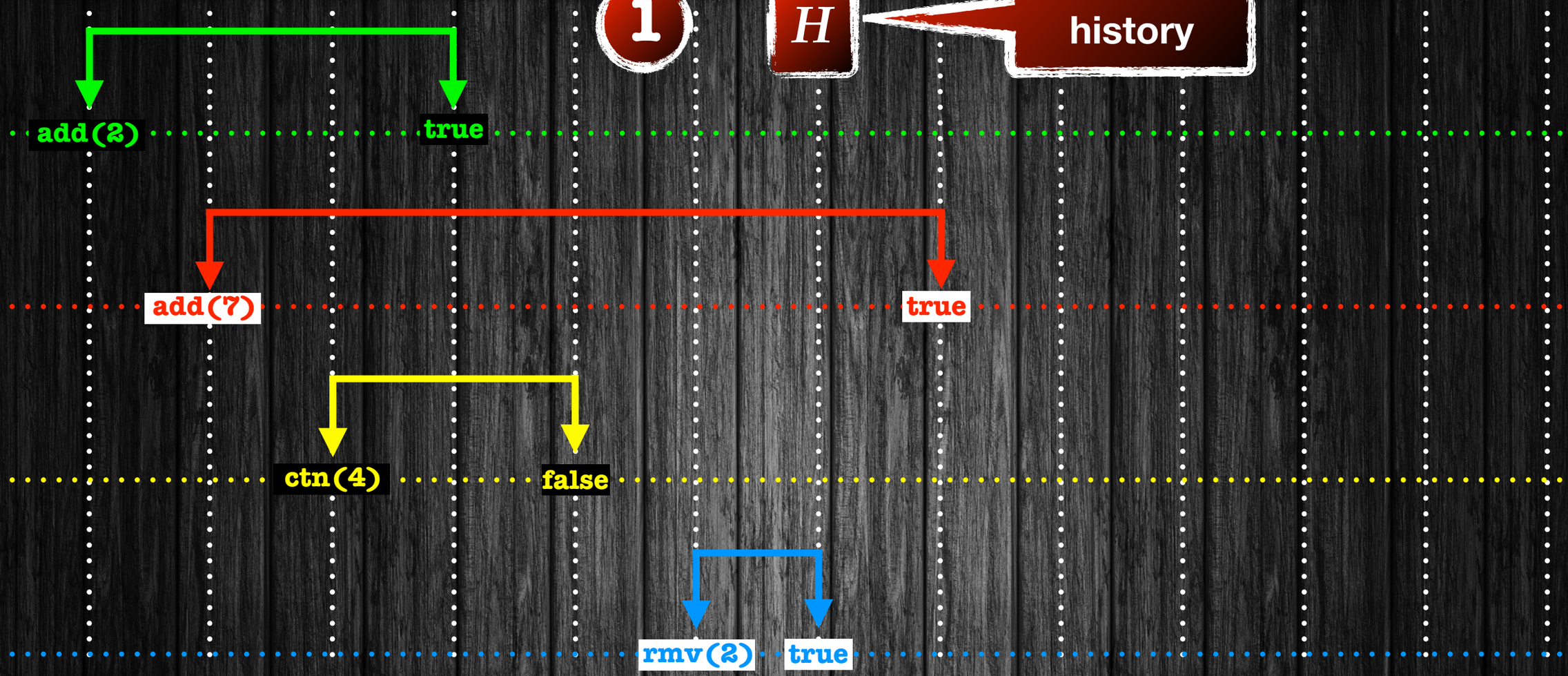


Concurrent Histories | Linearizability | Sets

1

H

Concurrent history

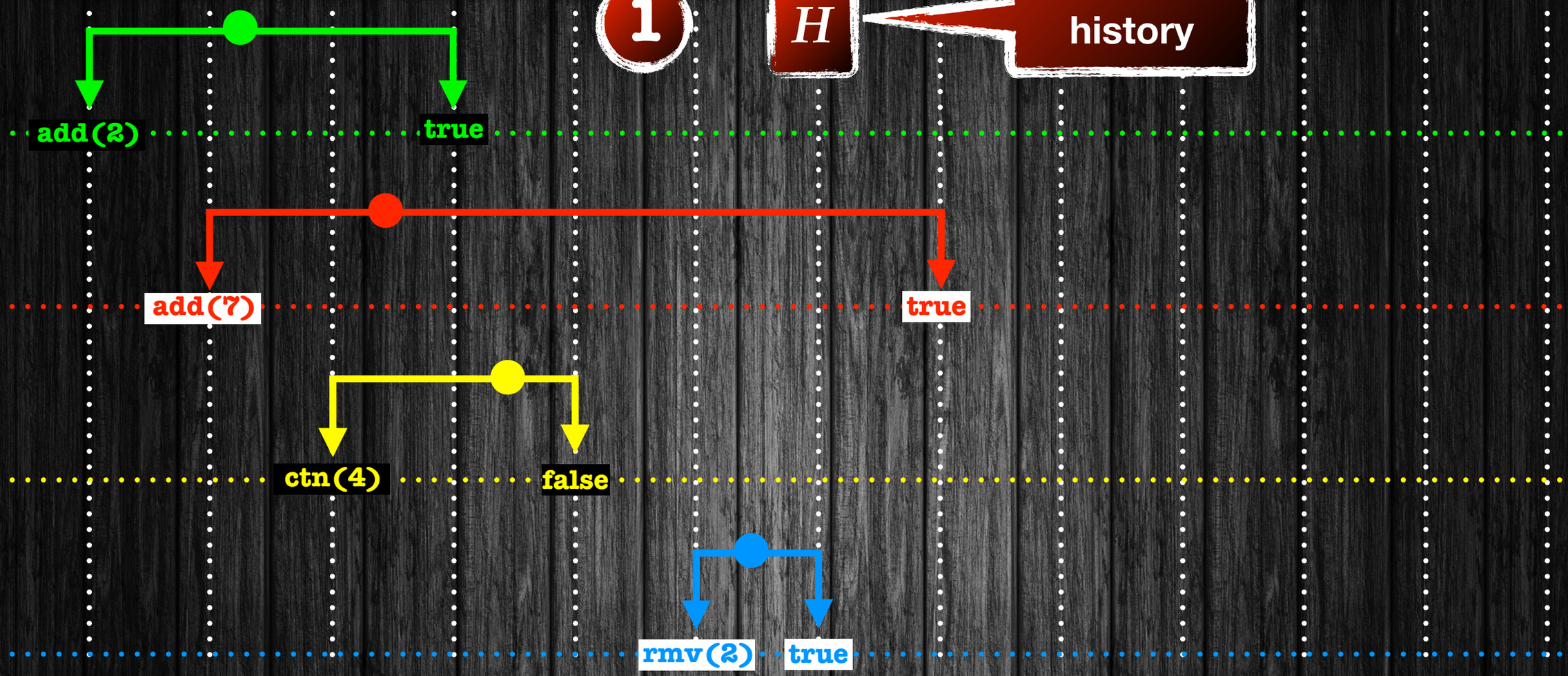


Concurrent Histories | Linearizability | Sets

1

H

Concurrent history



Concurrent Histories | Linearizability | Sets

1

H

Concurrent history

add(2)

true

add(7)

true

ctn(4)

false

rmv(2)

true

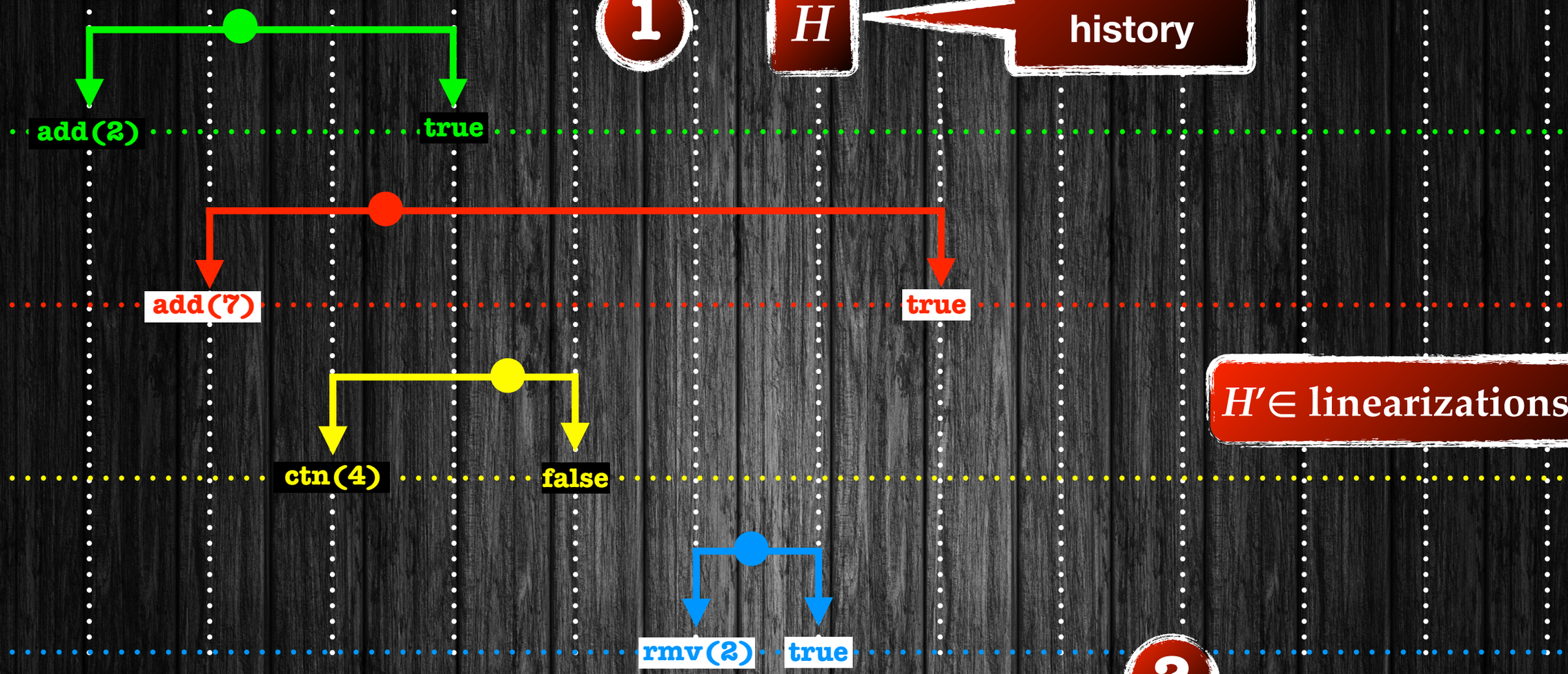
2

Concurrent Histories | Linearizability | Sets

1

H

Concurrent history

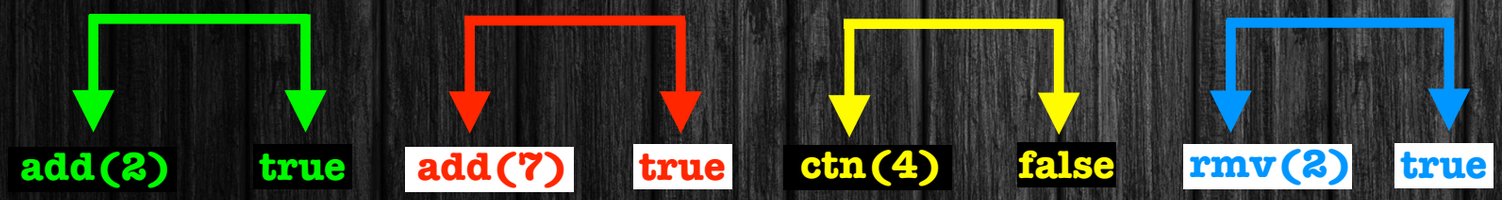


$H' \in$ linearizations(H)

2

H'

sequential history

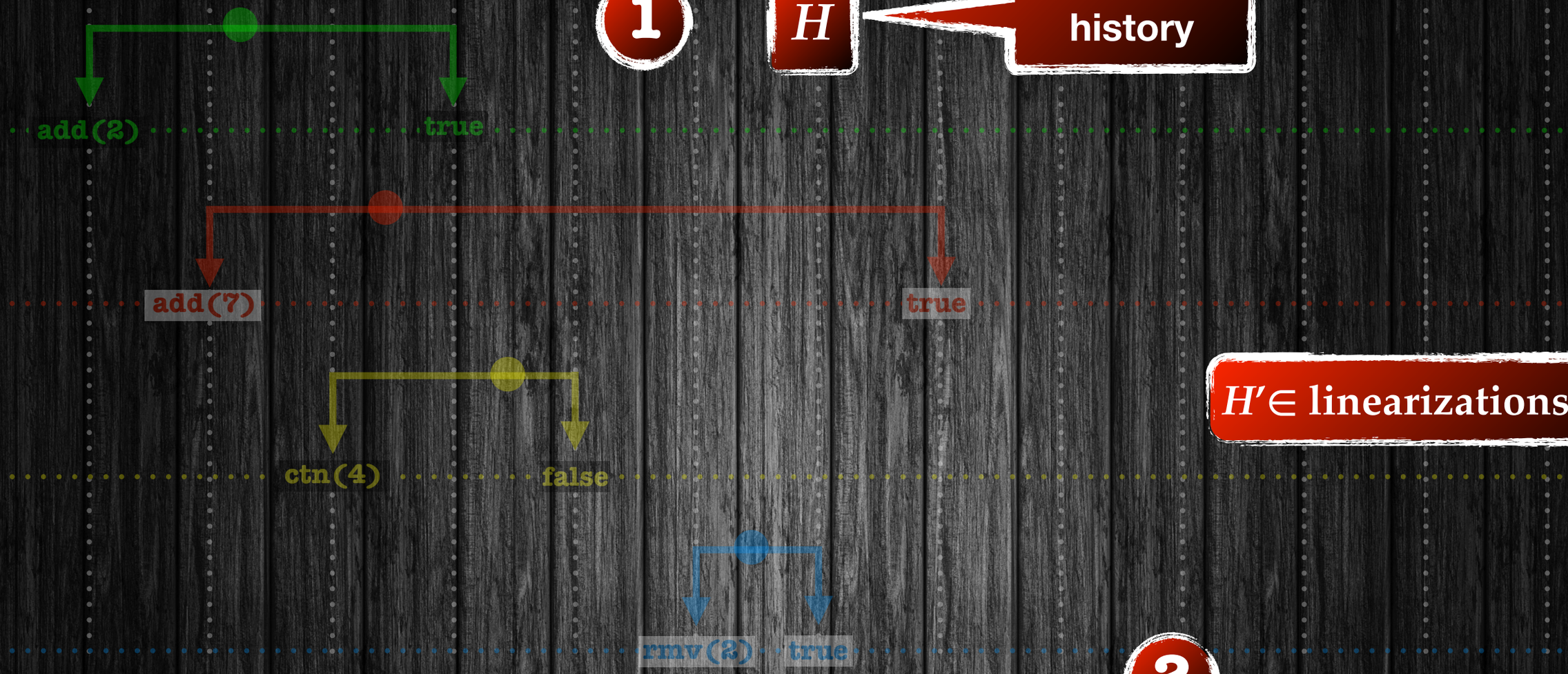


Concurrent Histories | Linearizability | Sets

1

H

Concurrent history

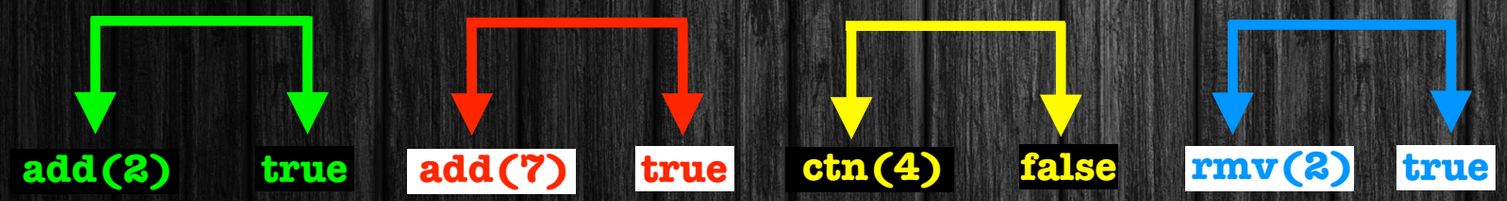


$H' \in$ linearizations(H)

2

H'

sequential history

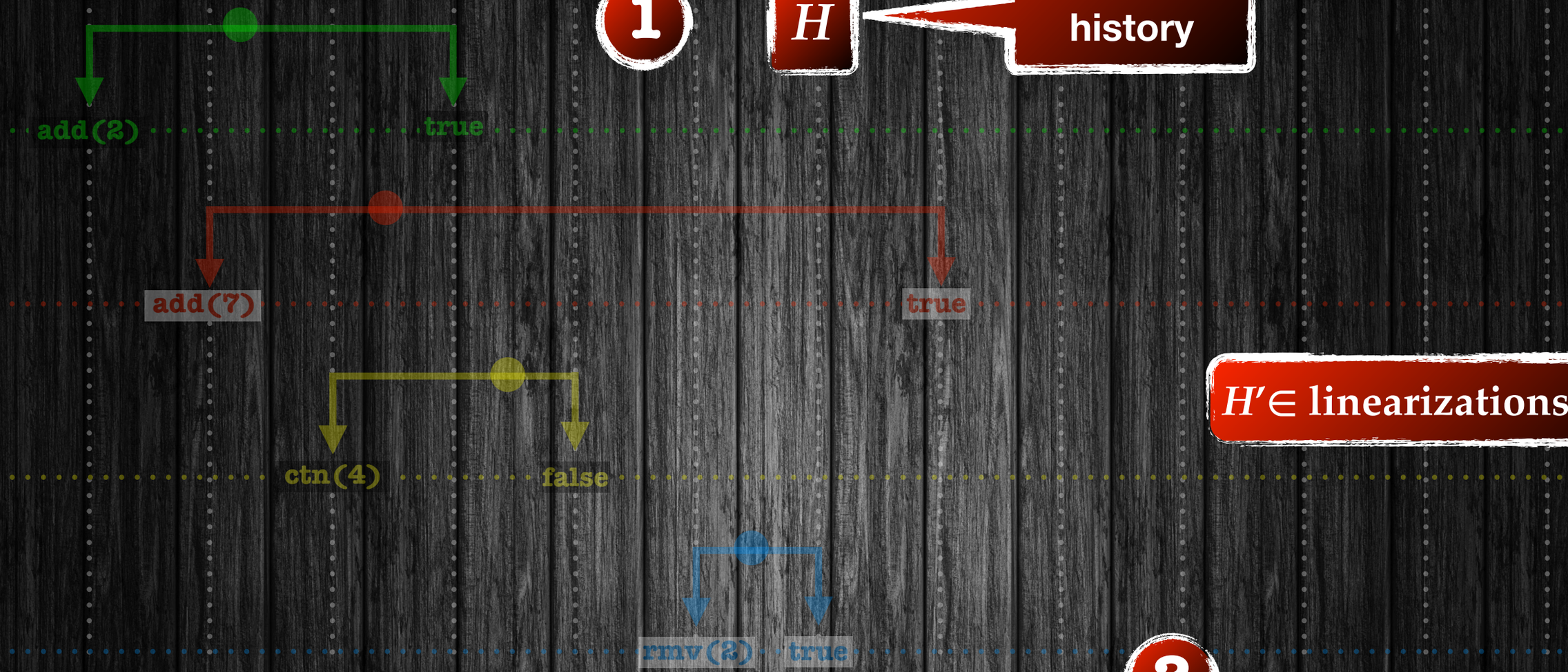


Concurrent Histories | Linearizability | Sets

1

H

Concurrent history

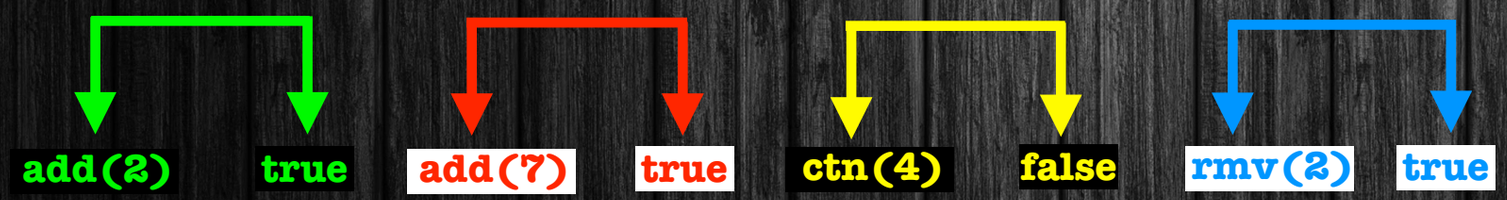


$H' \in$ linearizations(H)

2

H'

sequential history



3

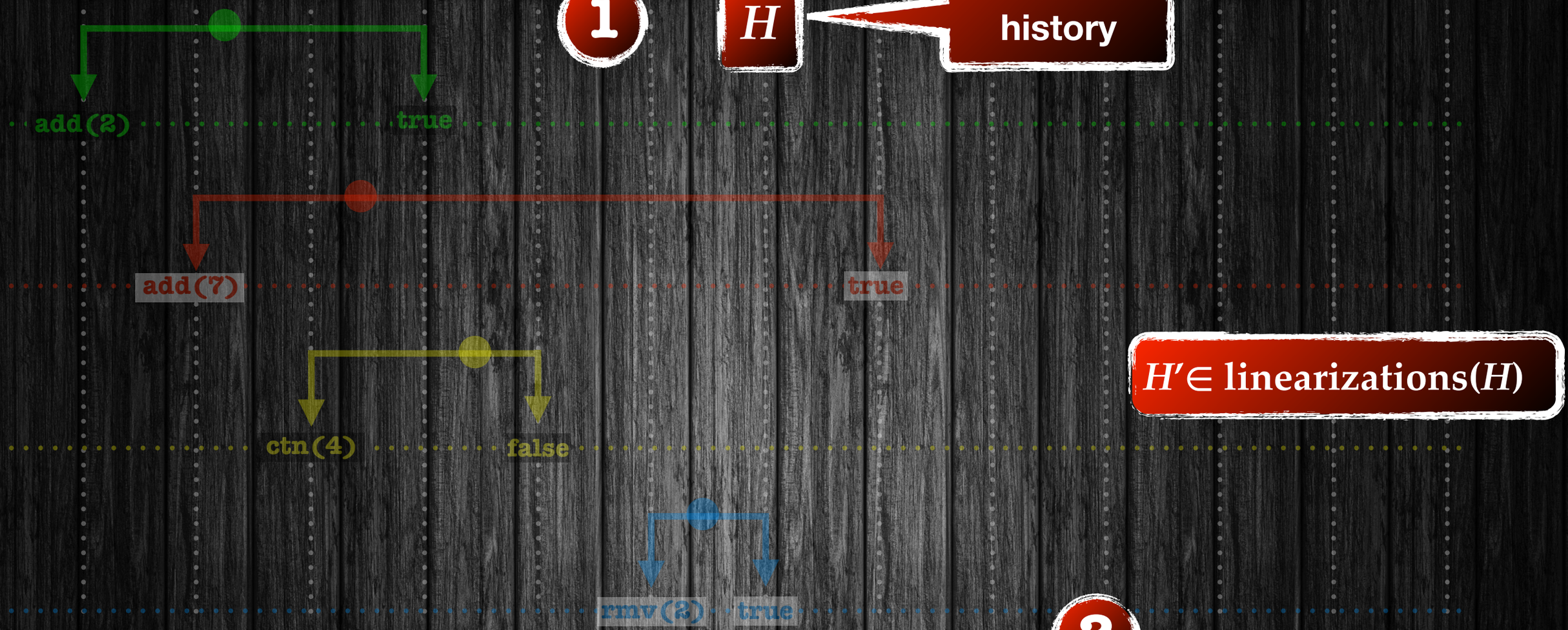
τ

τ = Traceof(H')

$\tau \in$ Traces(Set)

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

Concurrent Histories | Linearizability | Sets



1

H

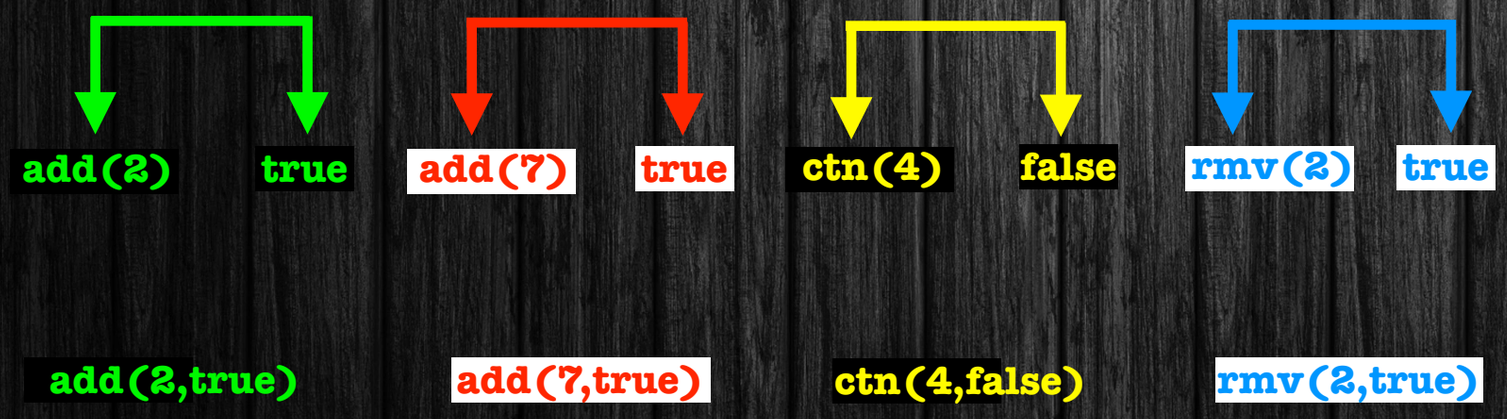
Concurrent history

$H' \in$ linearizations(H)

2

H'

sequential history



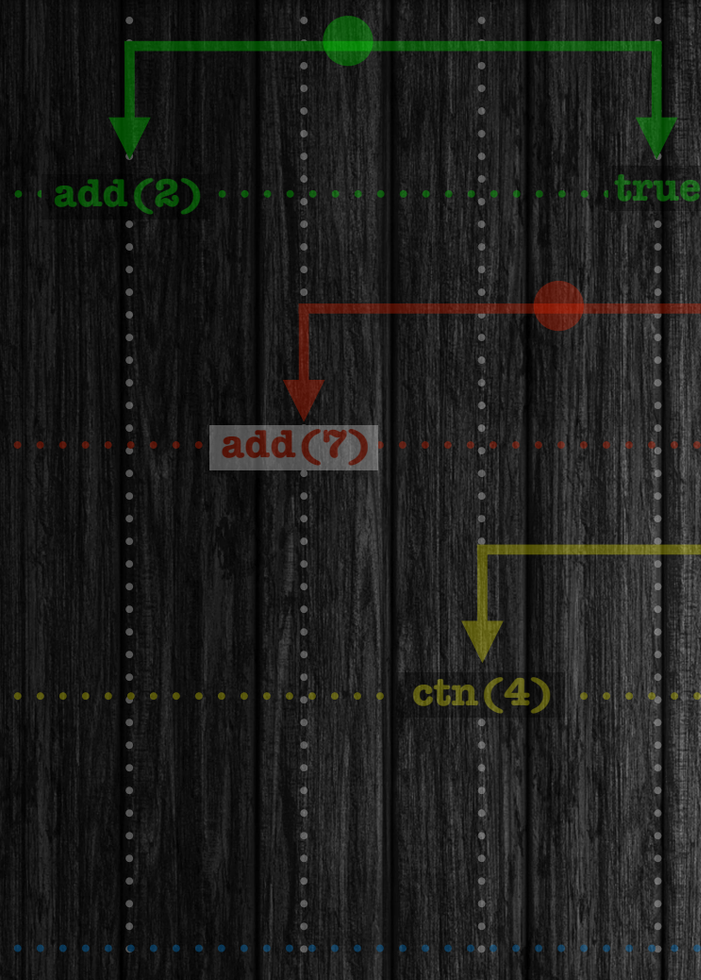
3

τ

τ = Traceof(H')

$\tau \in$ Traces(Set)

Concurrent Histories | Linearizability | Sets



1

H

Concurrent history

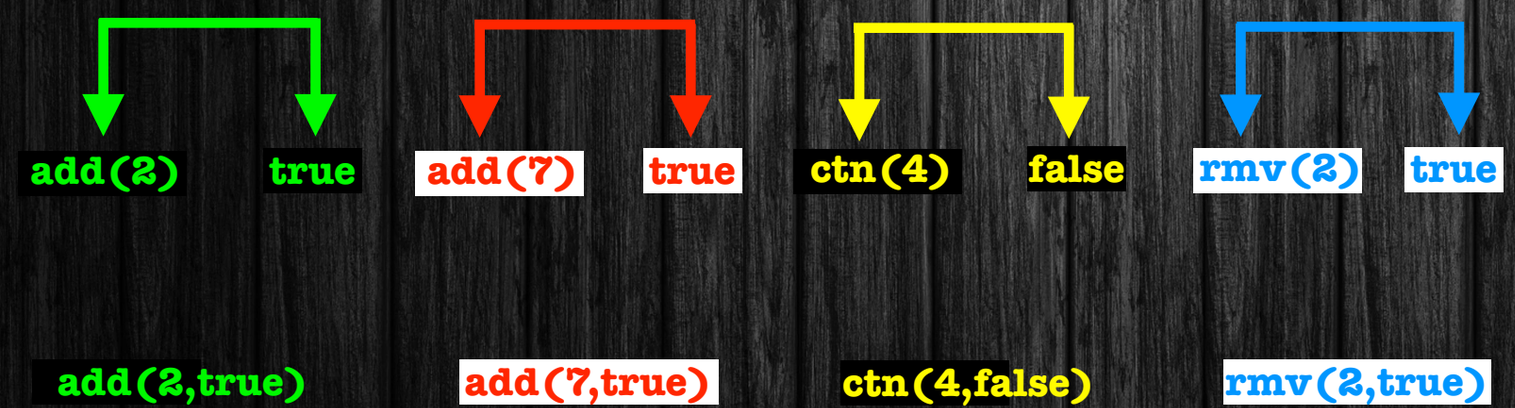
$H' \in$ linearizations(H)

$H' \in$ Histories(Set)

2

H'

sequential history



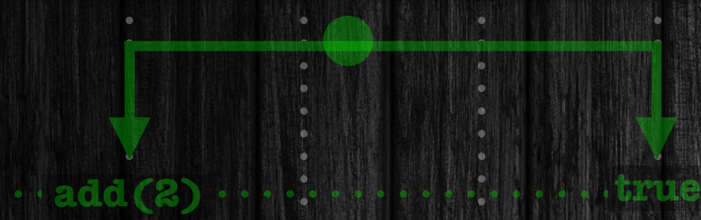
3

τ

τ = Traceof(H')

$\tau \in$ Traces(Set)

Concurrent Histories | Linearizability | Sets

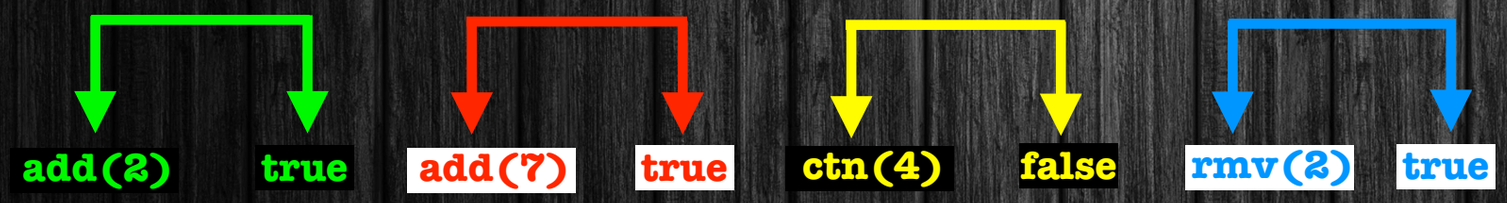








1 H — Concurrent history



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

2 H' — sequential history

3 τ

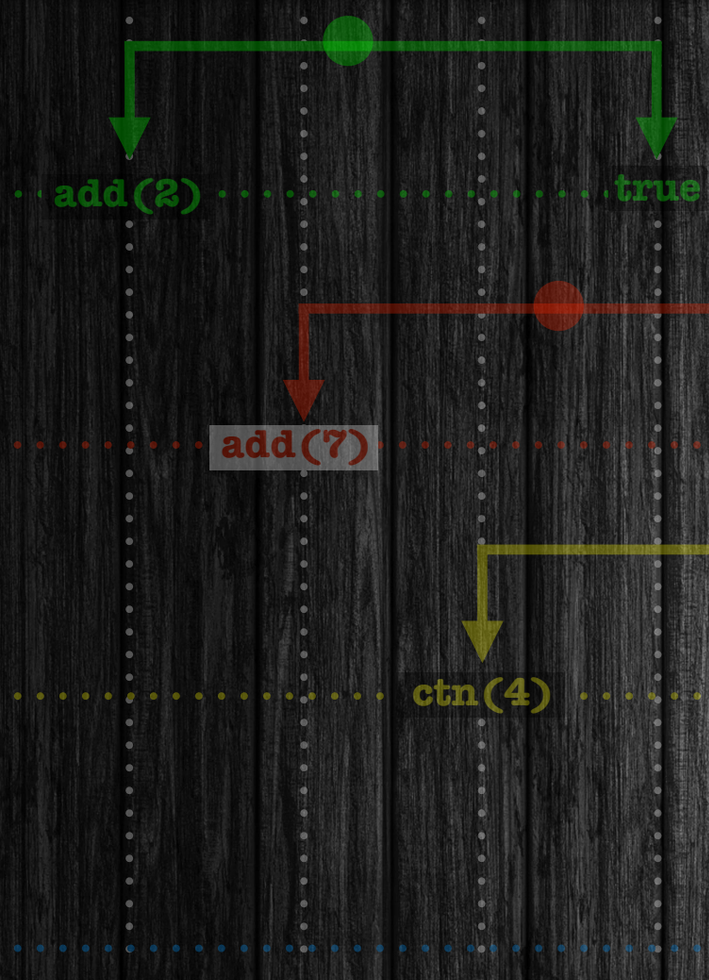
$H' \in$ linearizations(H)

$H' \in$ Histories(Set)

τ = Traceof(H')

$\tau \in$ Traces(Set)

Concurrent Histories | Linearizability | Sets



1

H

Concurrent history

H linearizable wrt. Set

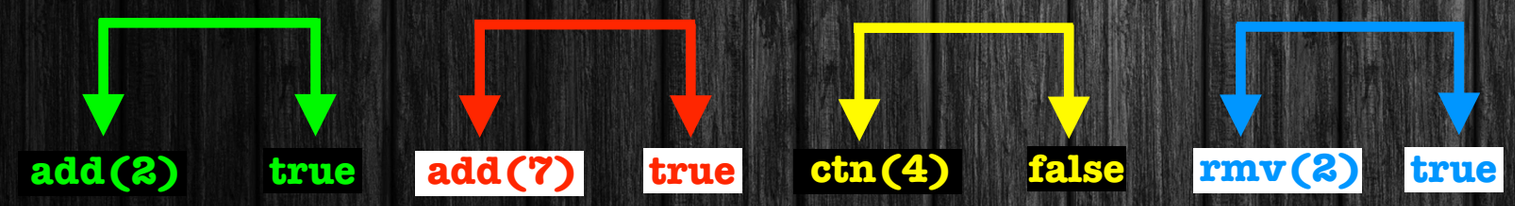
$H' \in$ linearizations(H)

$H' \in$ Histories(Set)

2

H'

sequential history



3

τ

τ = Traceof(H')

$\tau \in$ Traces(Set)

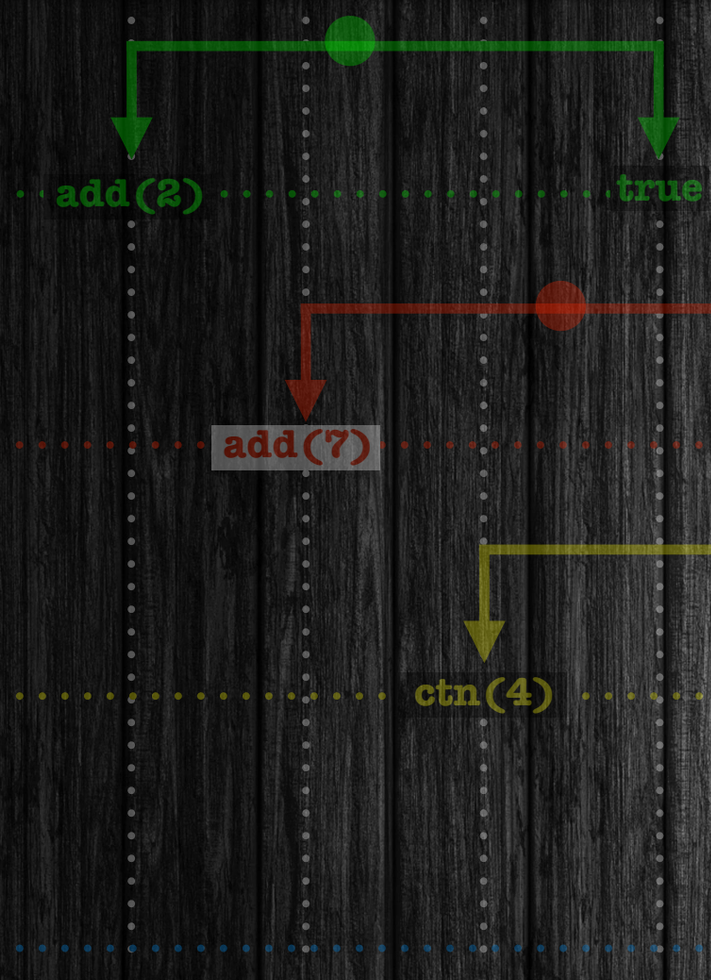
add(2,true)

add(7,true)

ctn(4,false)

rmv(2,true)

Concurrent Histories | Linearizability | Sets



1

H

Concurrent history

H linearizable wrt. Set

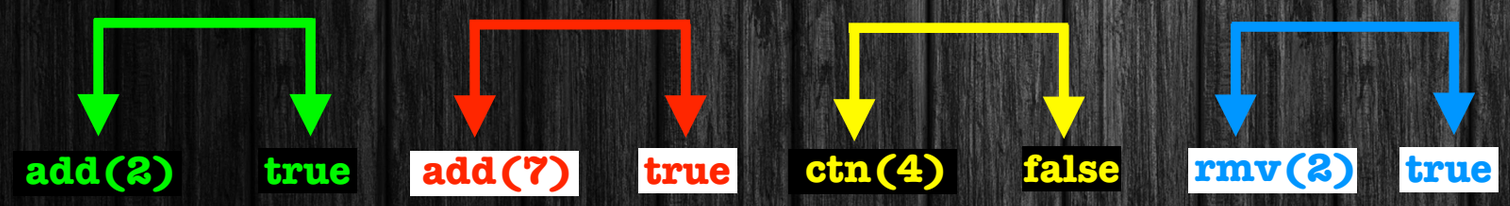
$H' \in$ linearizations(H)

$H' \in$ Histories(Set)

2

H'

sequential history



3

τ

τ = Traceof(H')

$\tau \in$ Traces(Set)

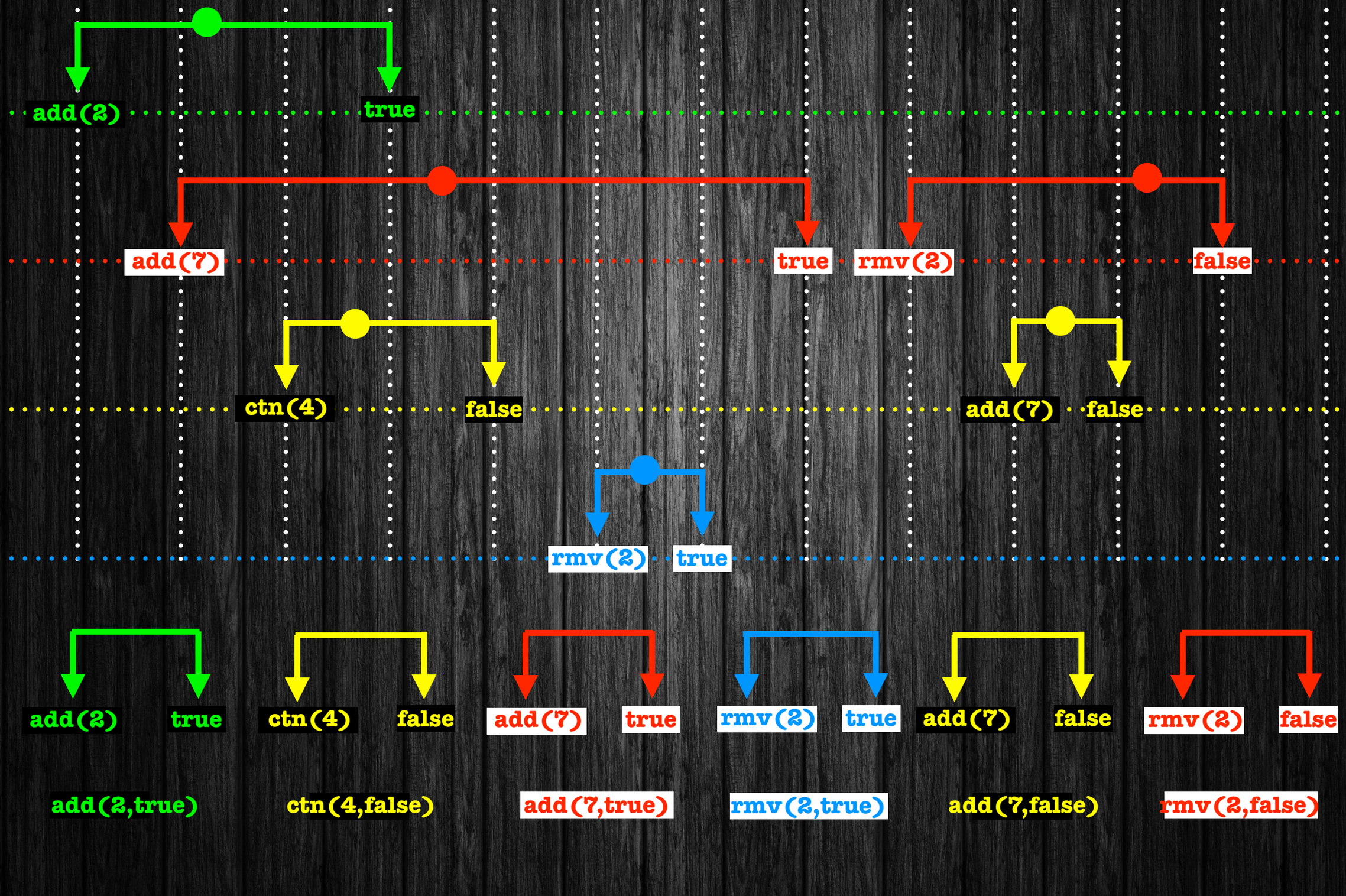
add(2,true)

add(7,true)

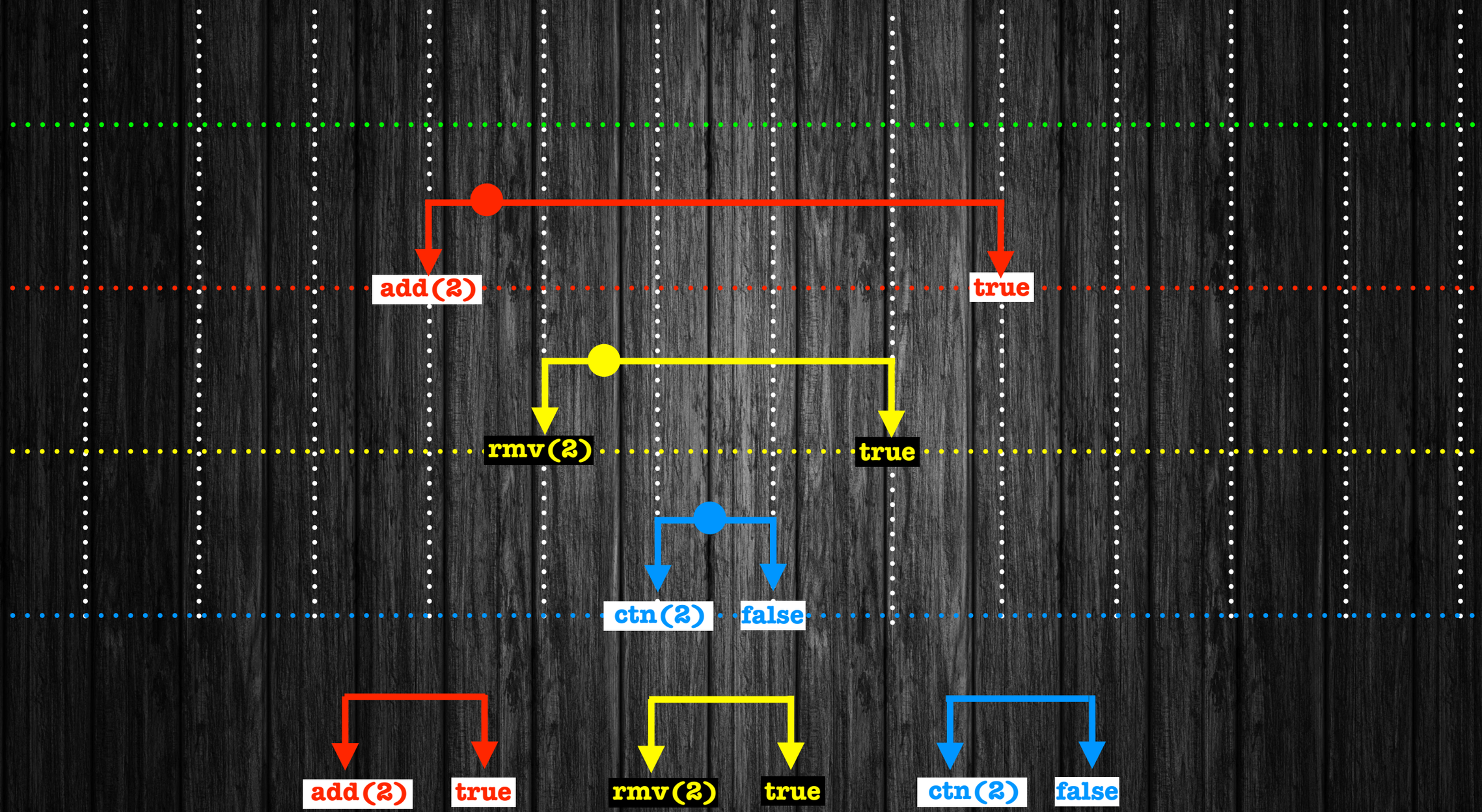
ctn(4,false)

rmv(2,true)

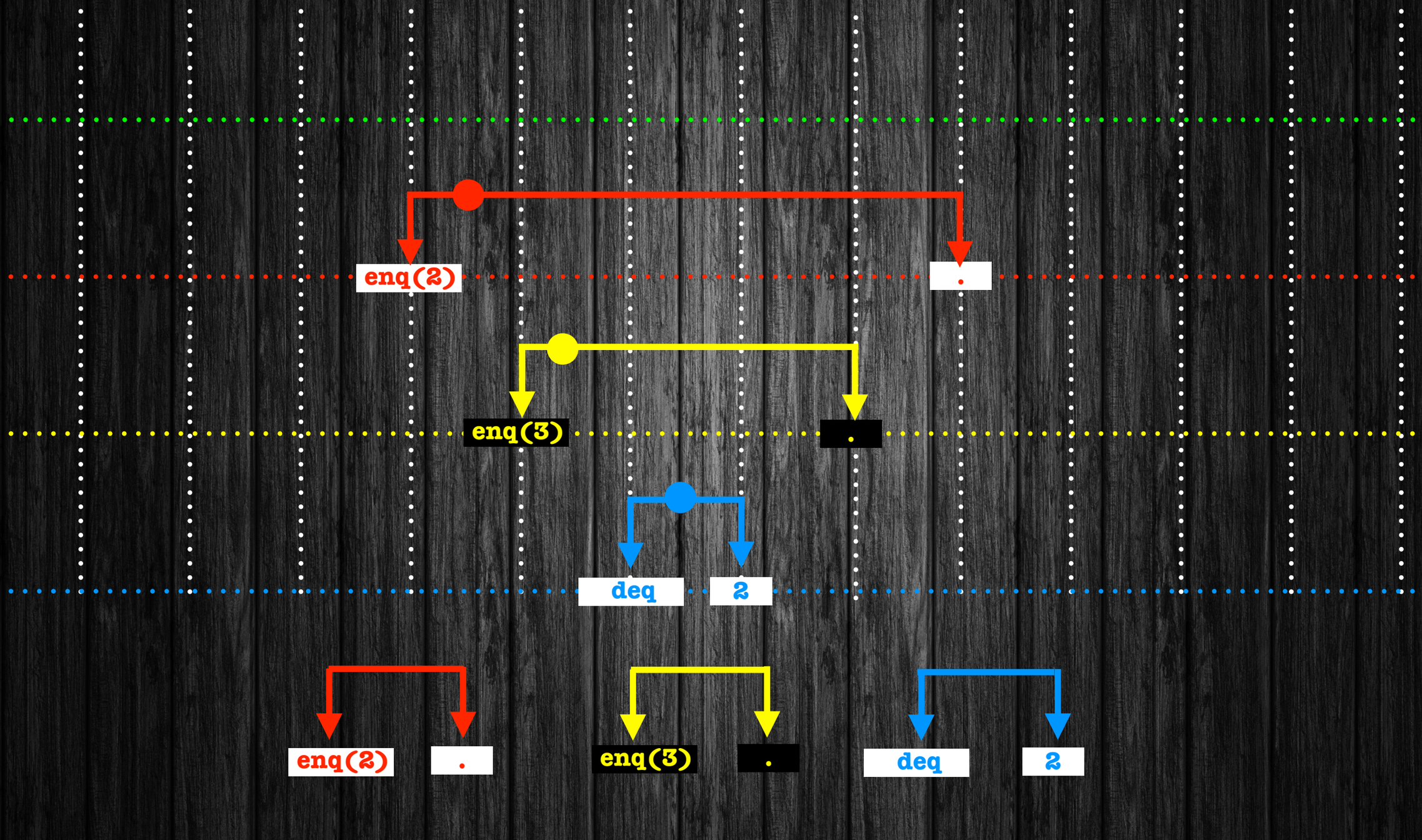
Concurrent Histories | Linearizability | Sets



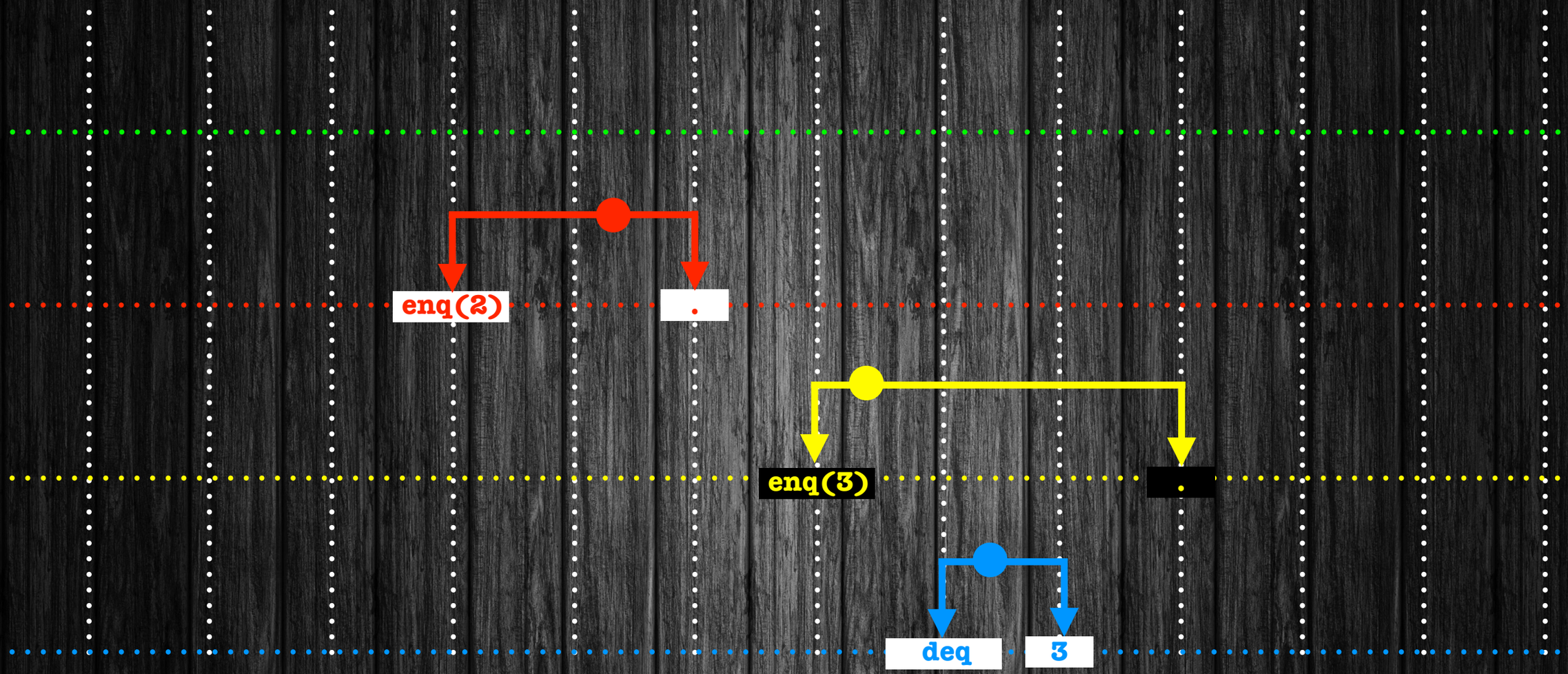
Concurrent Histories | Linearizability | Sets



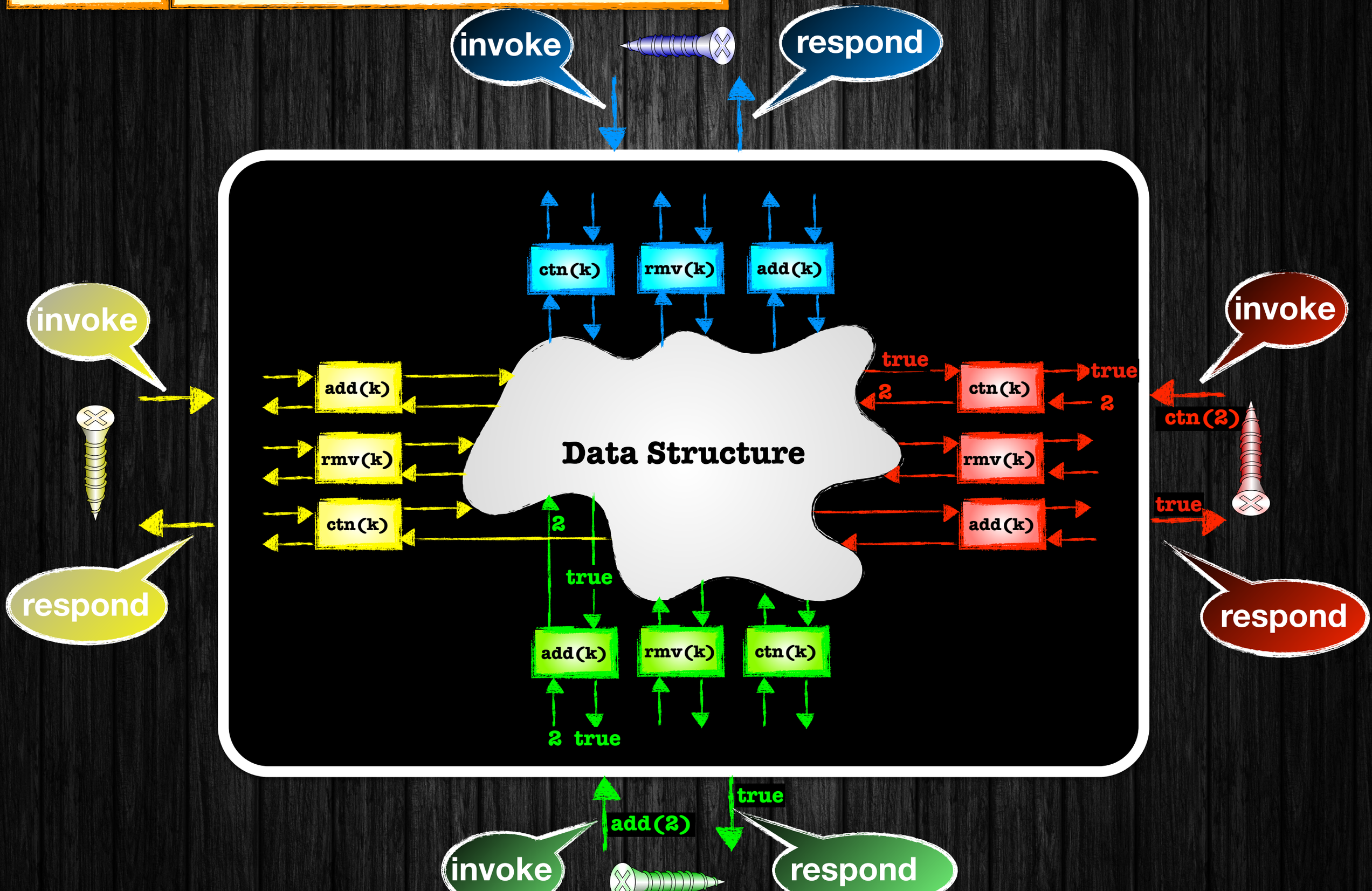
Concurrent Histories | Linearizability | Queues



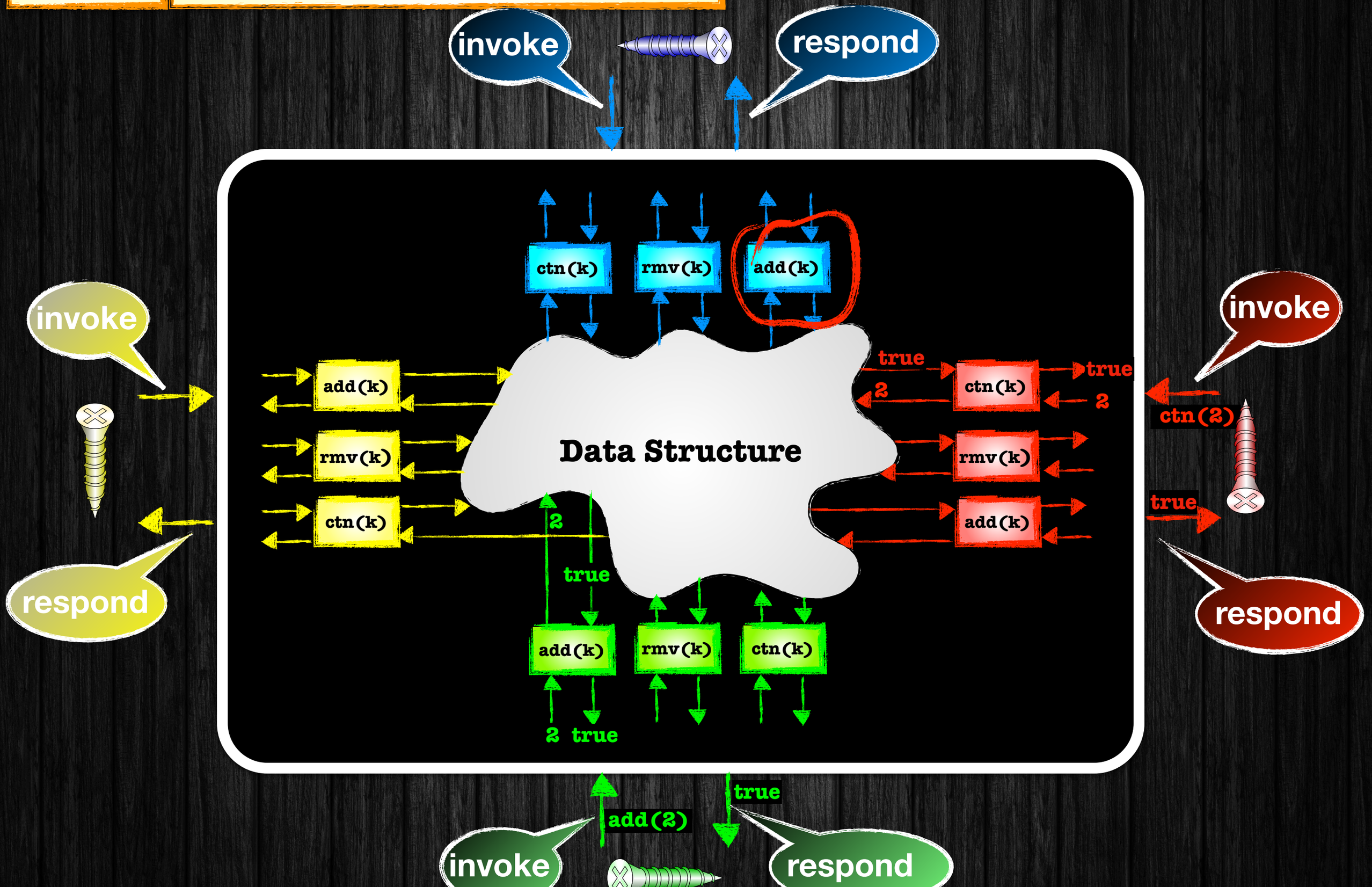
Concurrent Histories | Linearizability | Queues



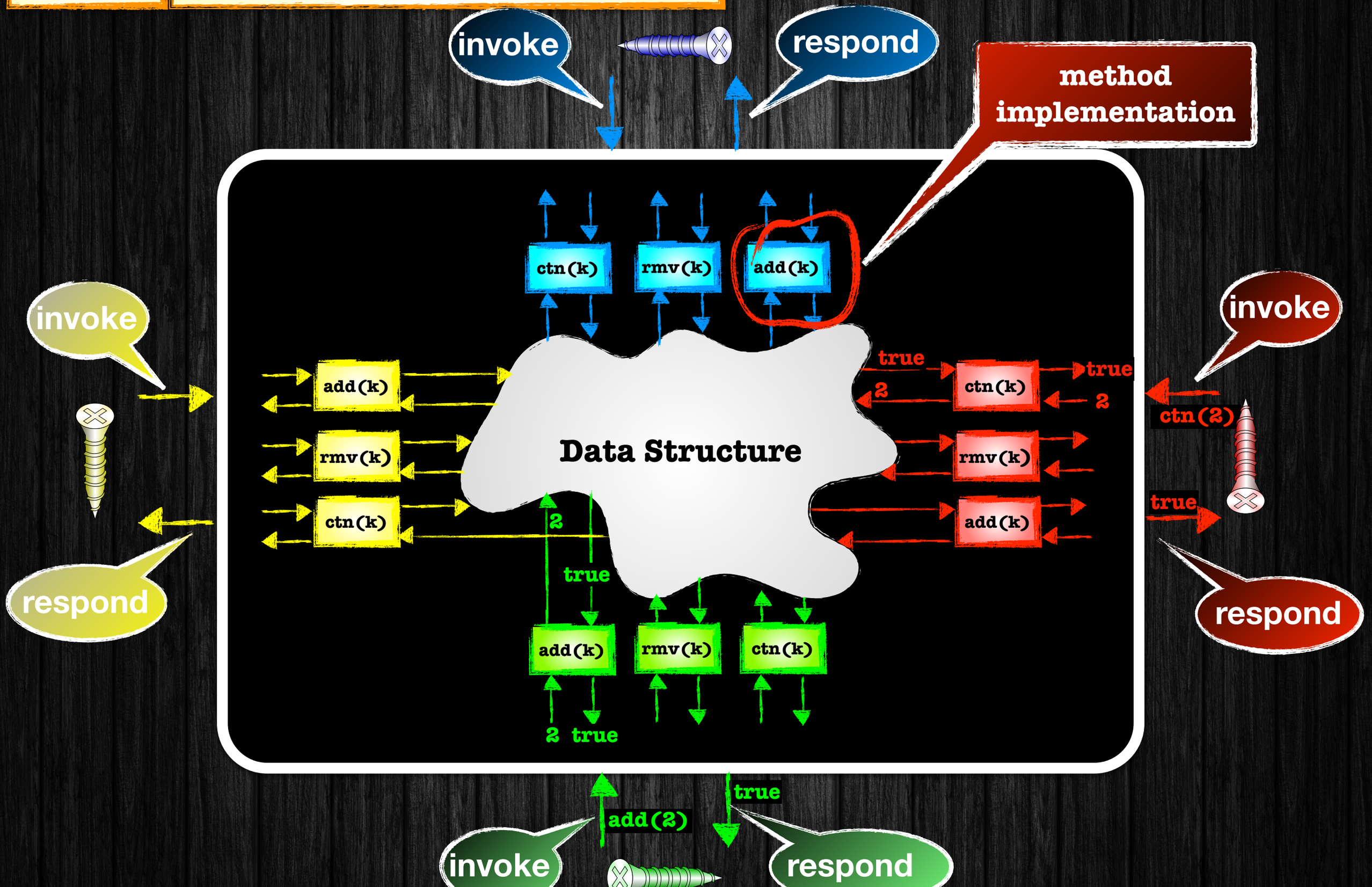
Sets Concurrent Programs



Sets Concurrent Programs



Sets Concurrent Programs



Sets | Concurrent Programs

code + linearization point

method implementation

invoke

respond

ctn(k) rmv(k) add(k)

add(k)

rmv(k)

ctn(k)

Data Structure

true

2

true

2

ctn(k)

rmv(k)

add(k)

2

true

add(k) rmv(k) ctn(k)

2 true

invoke

respond

invoke

ctn(2)

true

respond

add(2)

true

invoke

respond

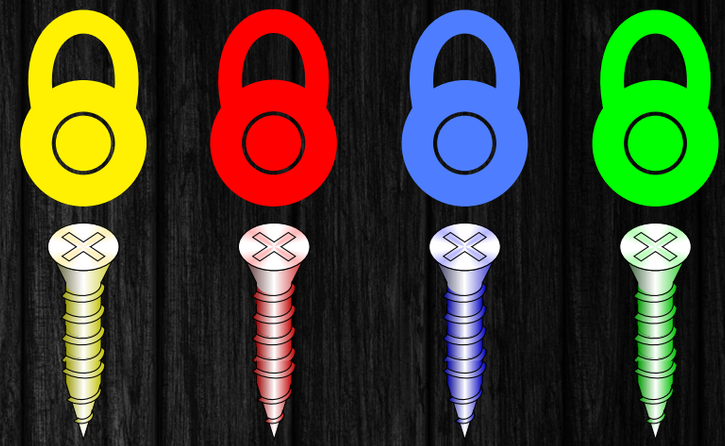
Outline

- 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

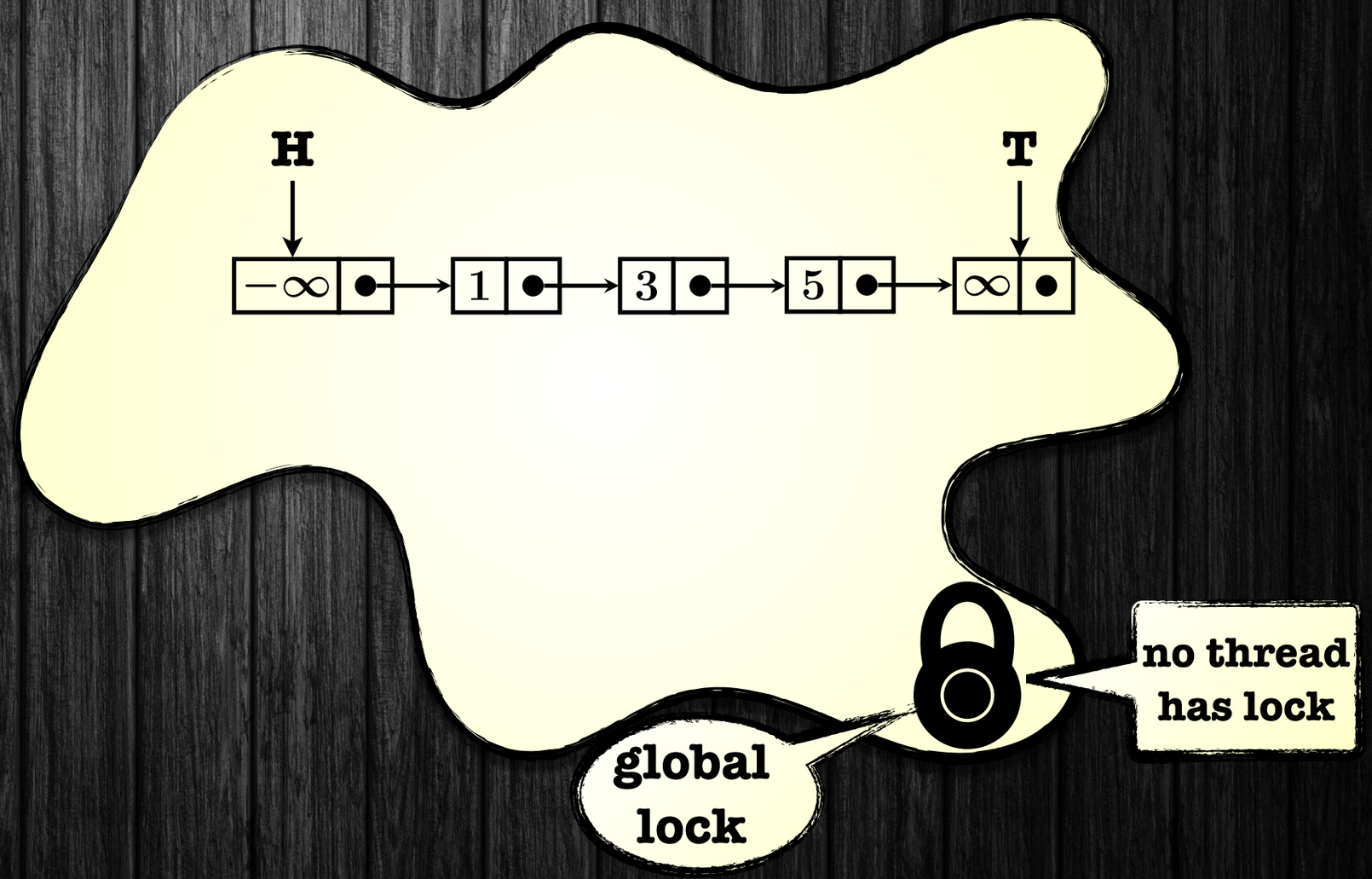
Outline

- 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

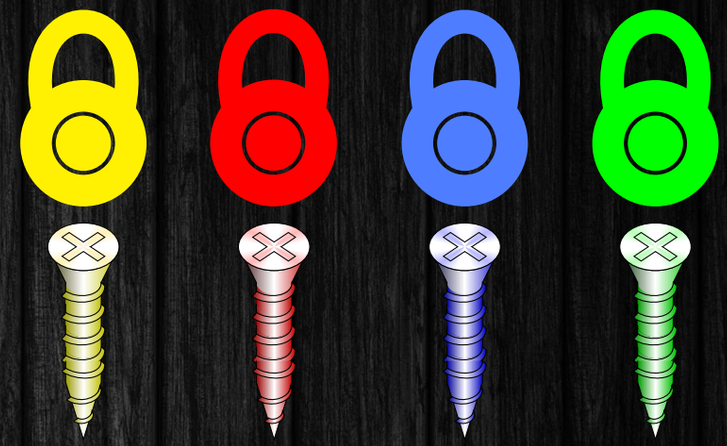
Sets Coarse-Grained Synchronization



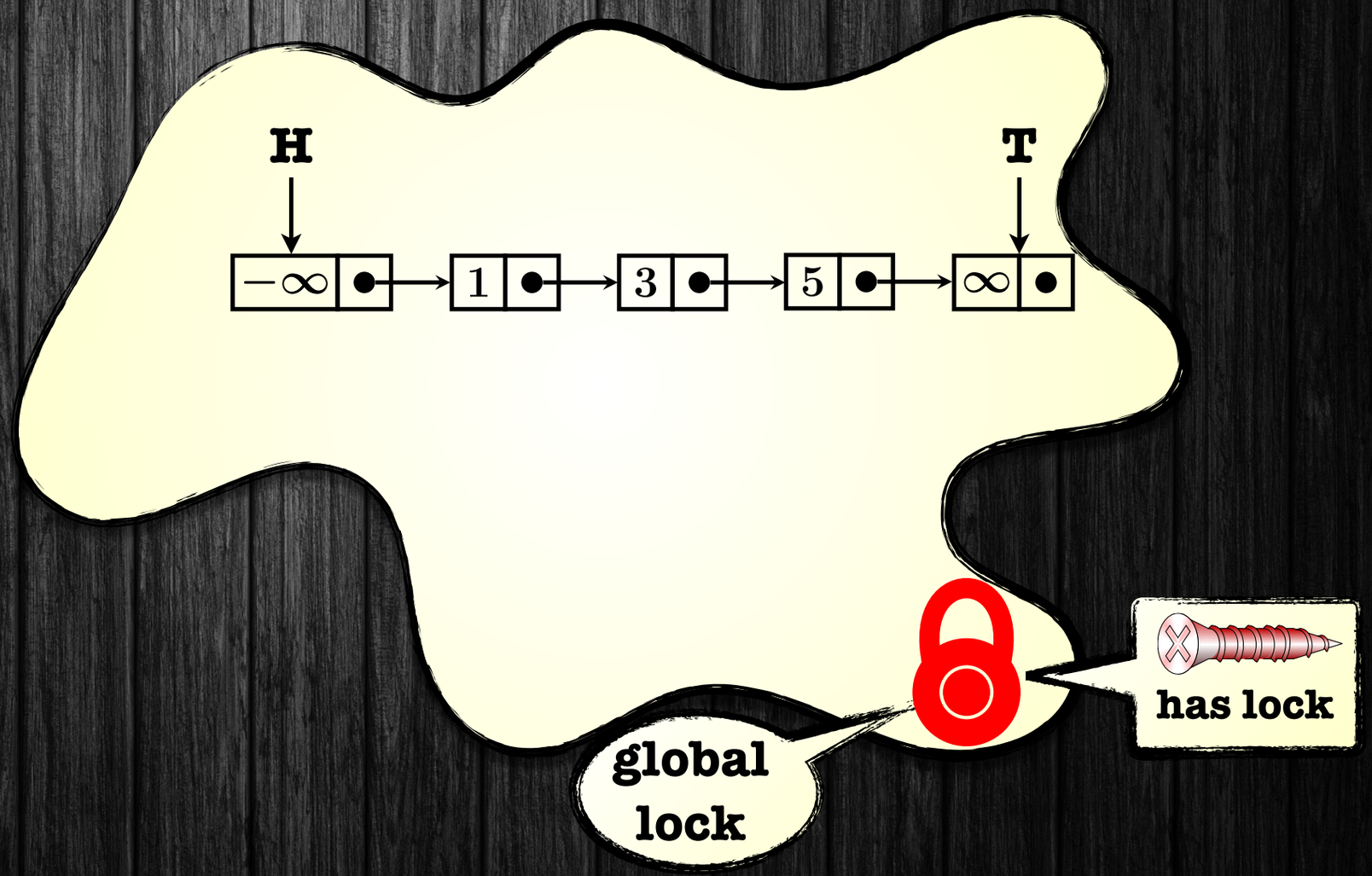
At most one thread has lock



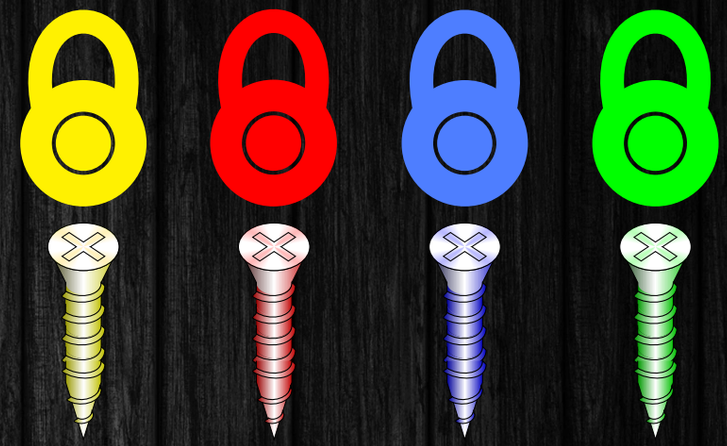
Sets Coarse-Grained Synchronization



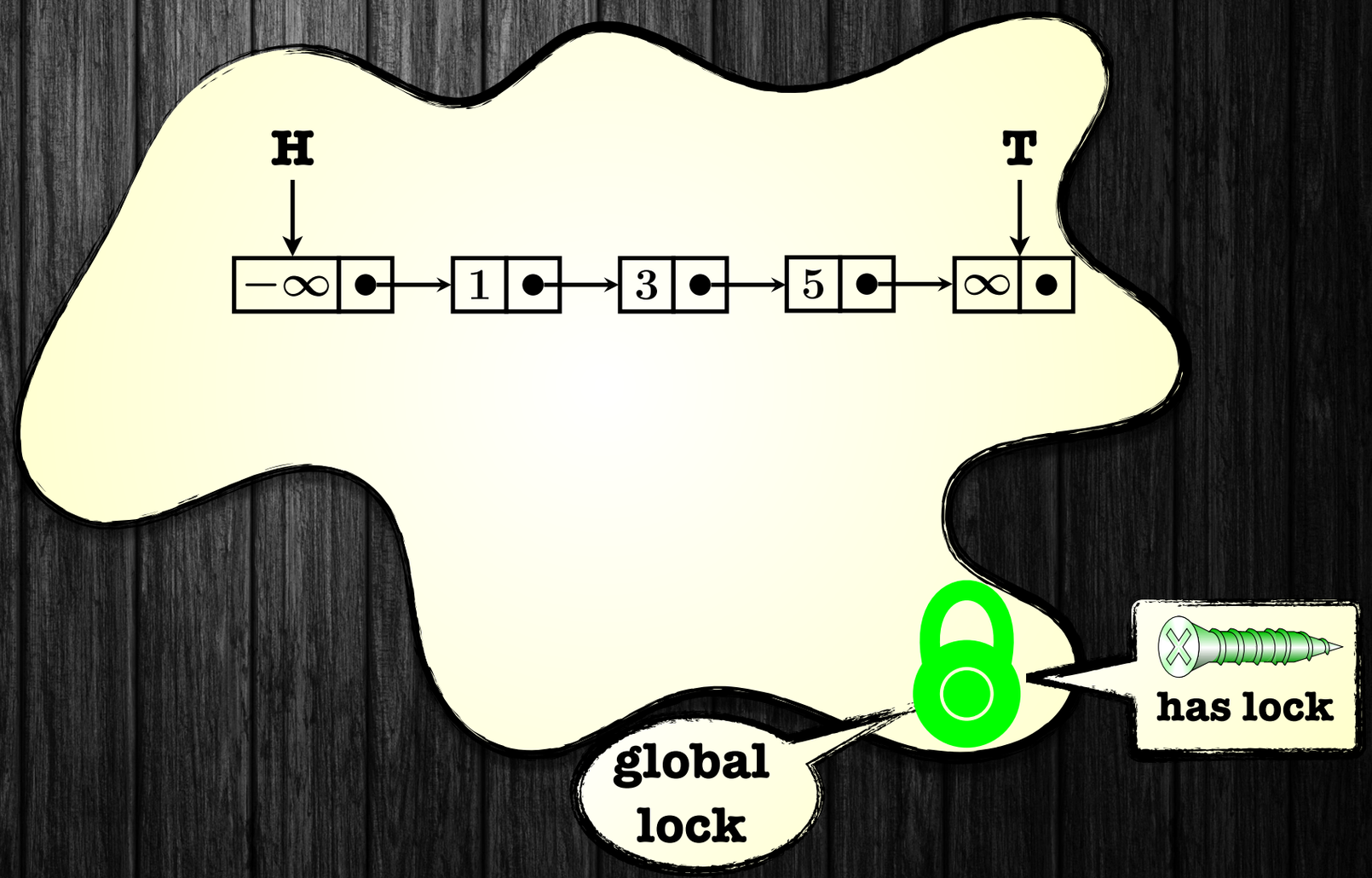
At most one thread has lock



Sets Coarse-Grained Synchronization

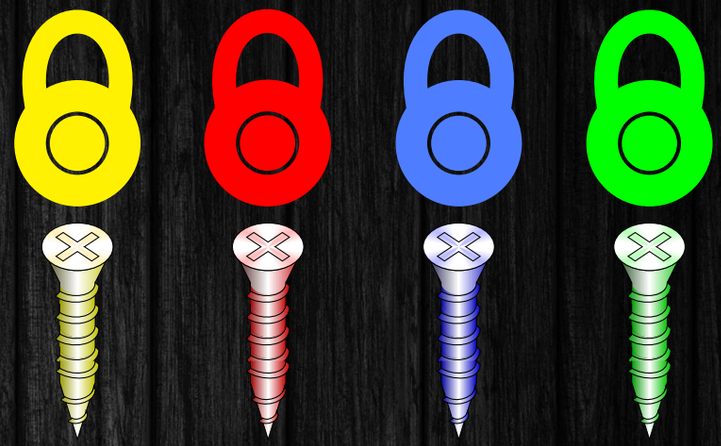


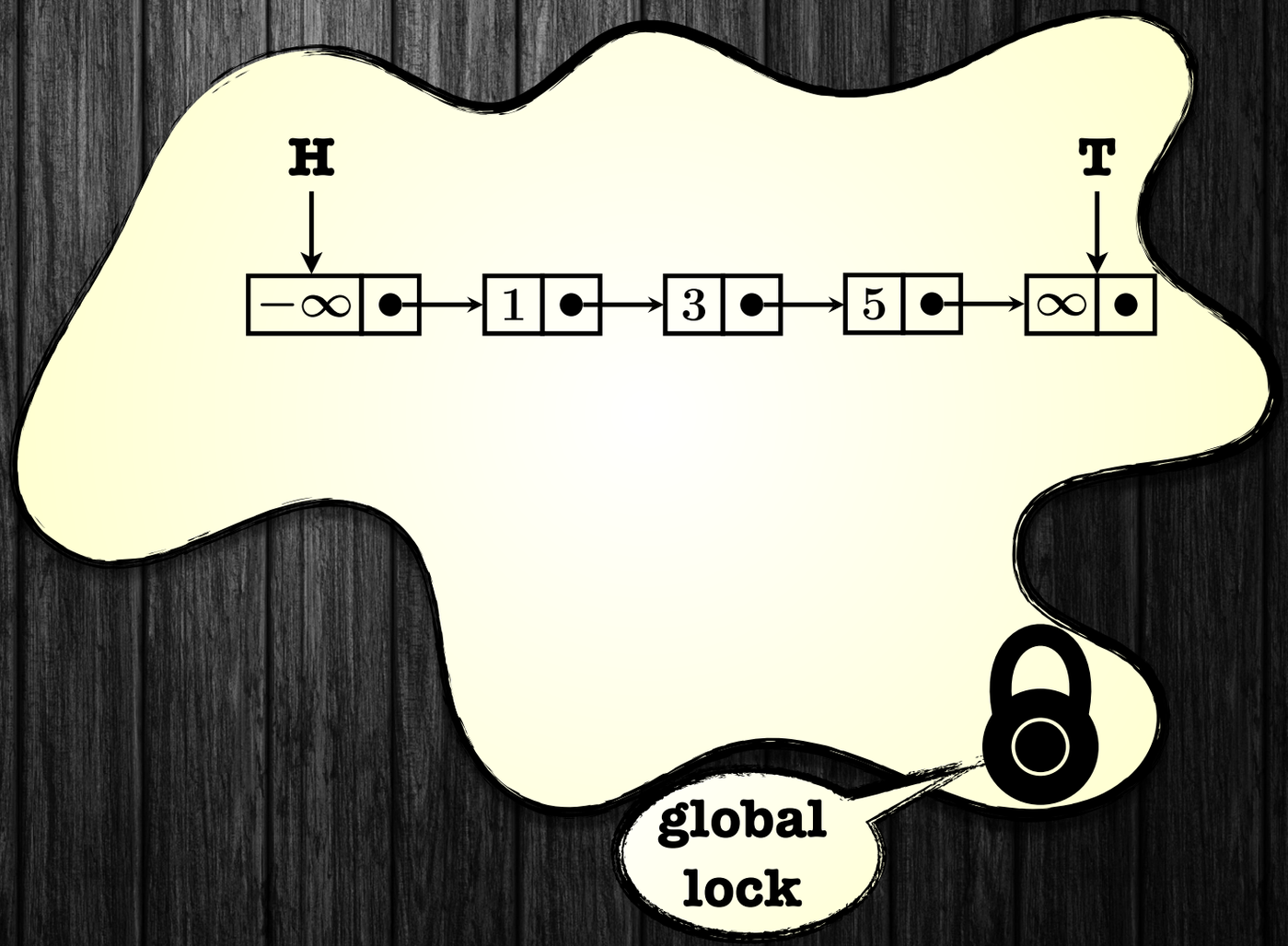
At most one thread has lock



Sets

Coarse-Grained Synchronization

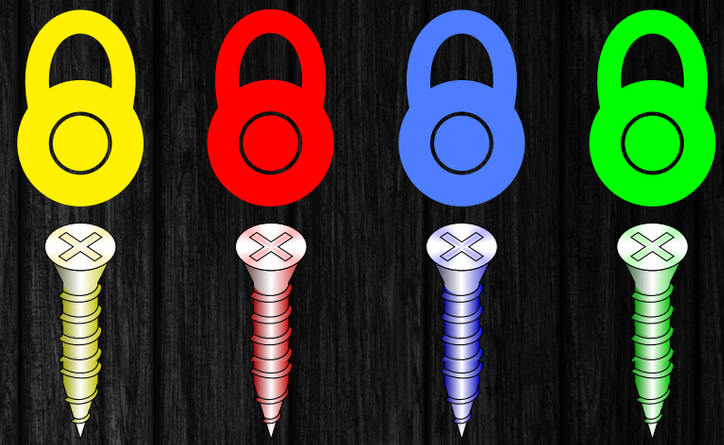


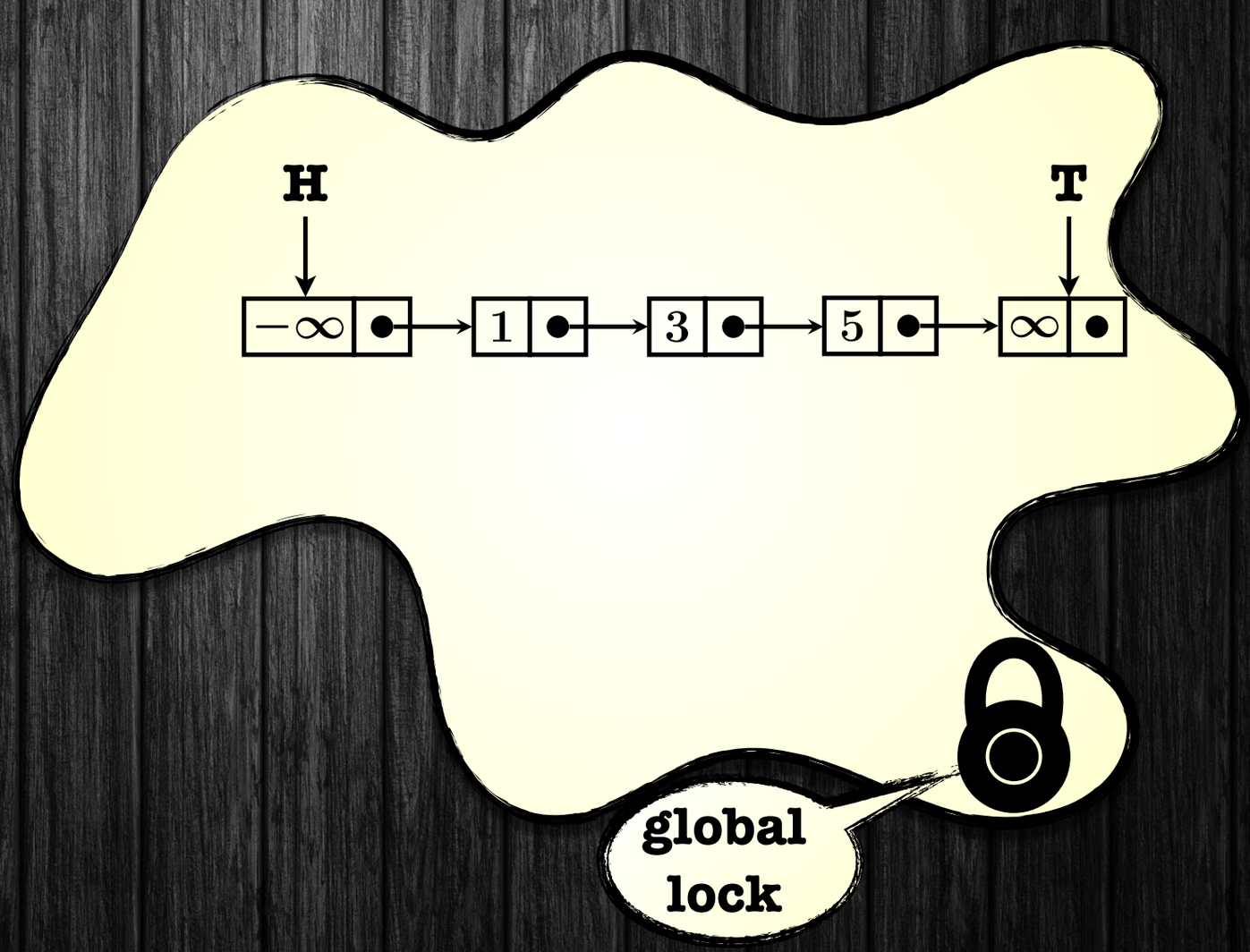


Sets

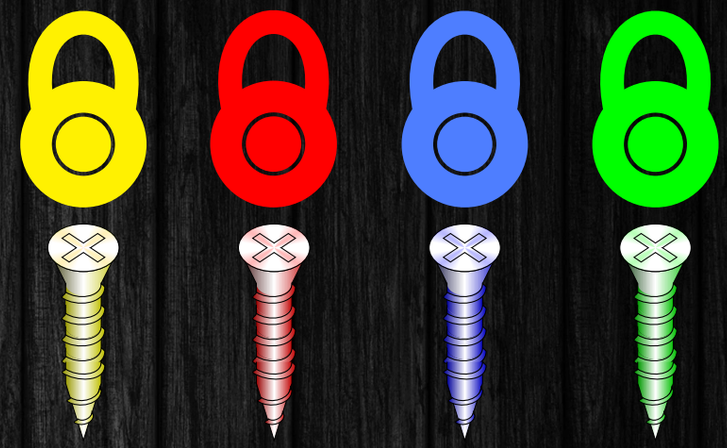
Coarse-Grained Synchronization

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



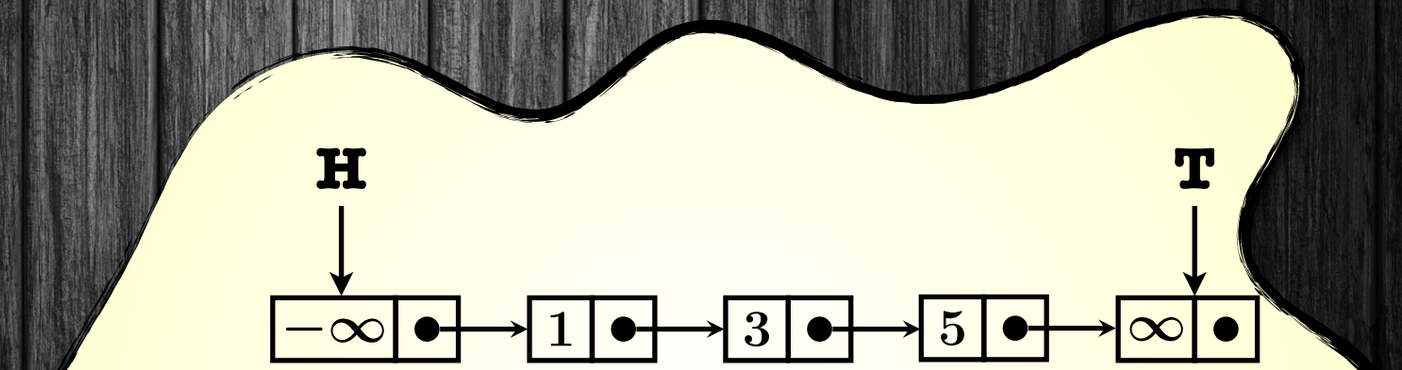


Sets | Coarse-Grained Synchronization



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

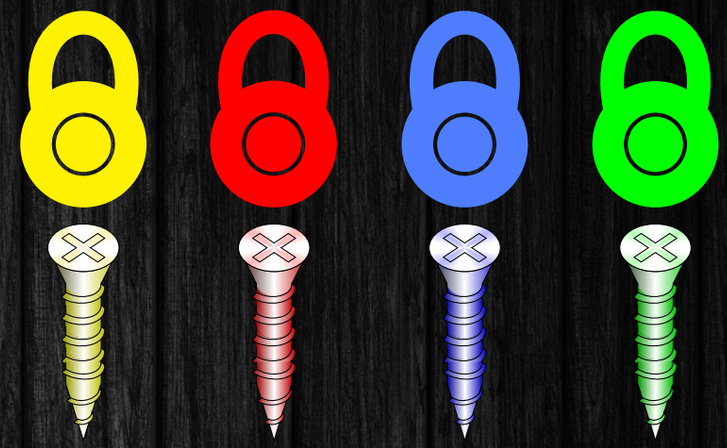
sequential code





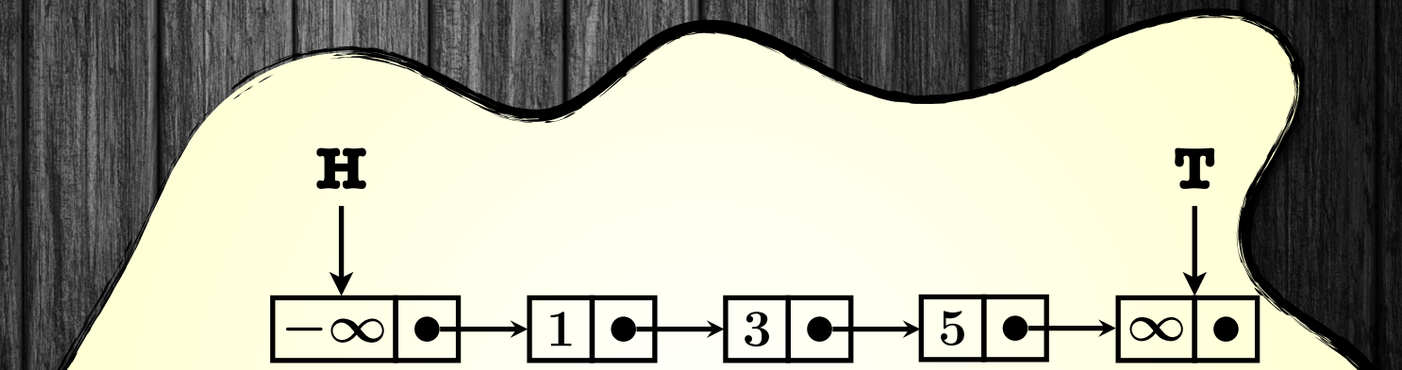
global lock

Sets | Coarse-Grained Synchronization



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

sequential code



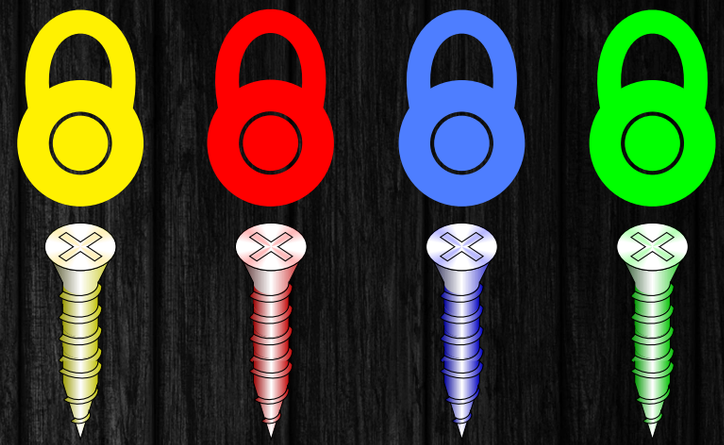


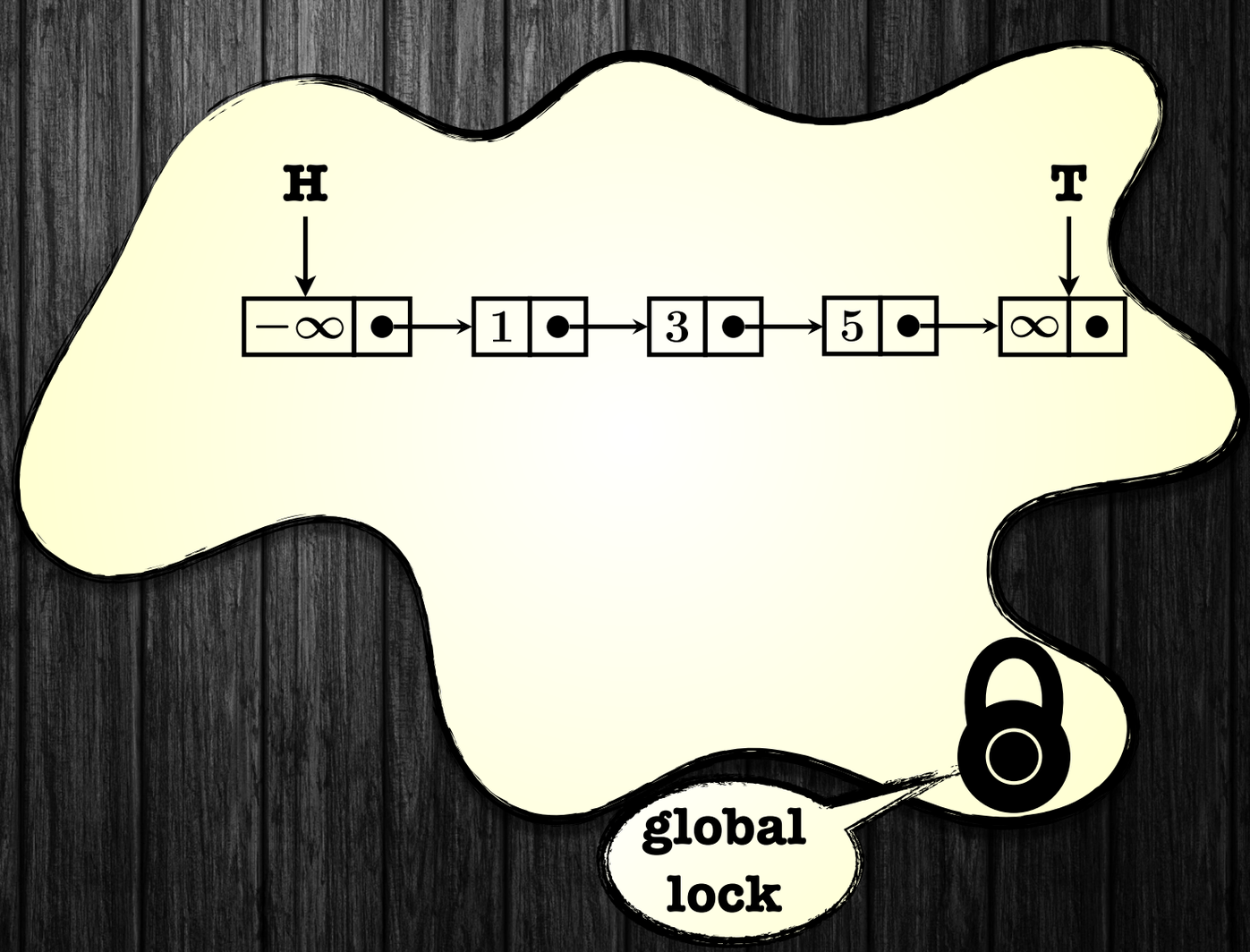
global lock

Sets

Coarse-Grained Synchronization

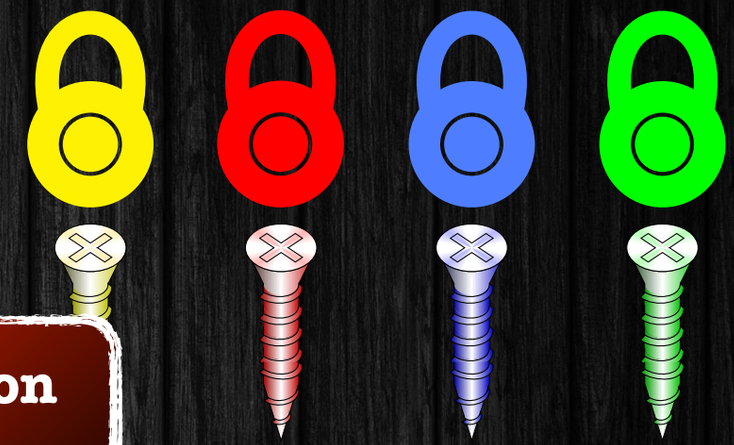
```
add(k):
Node p, c
Lock l
1  lock l;
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```





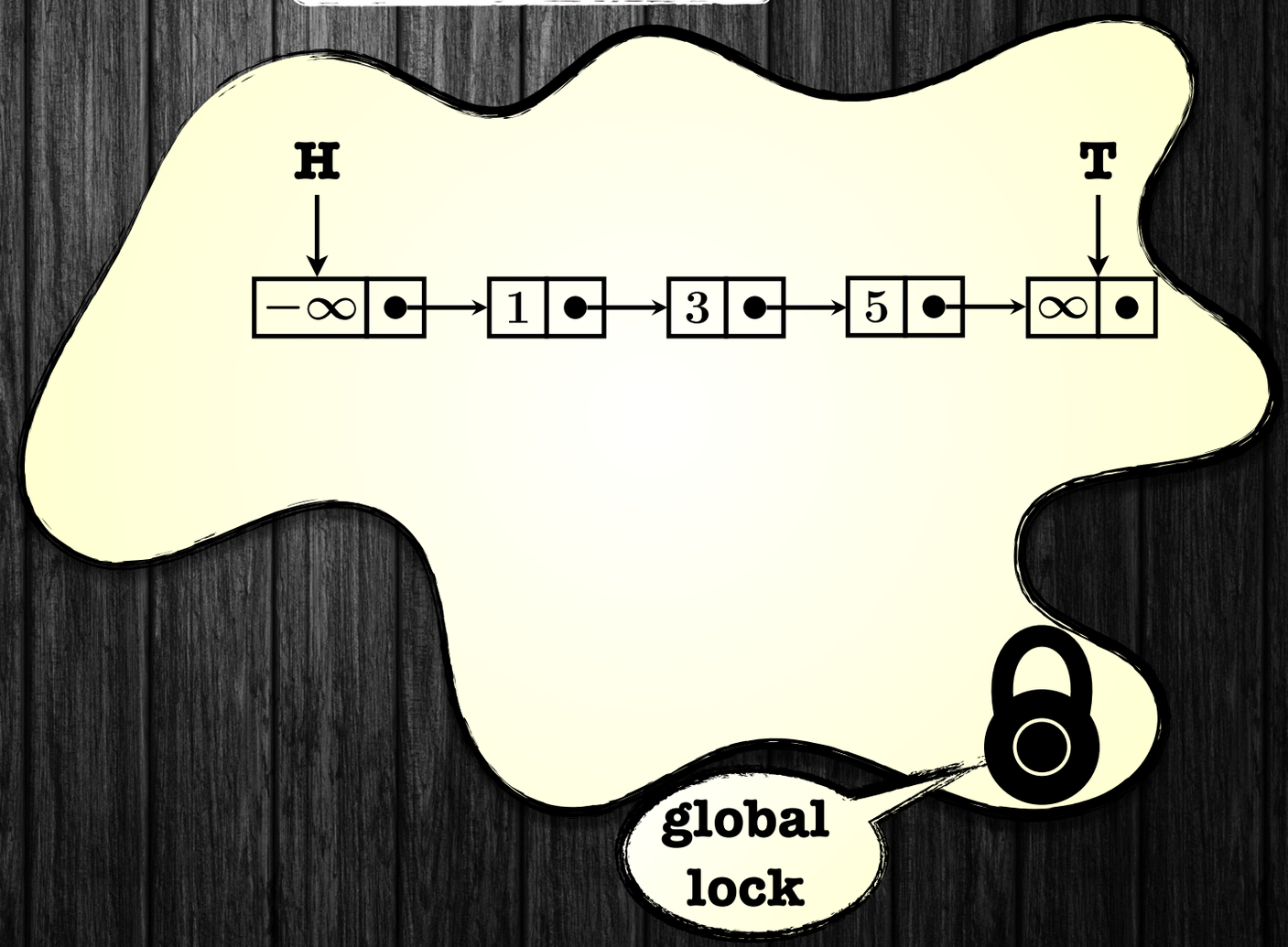
Sets | Coarse-Grained Synchronization

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



code + linearization point

method implementation



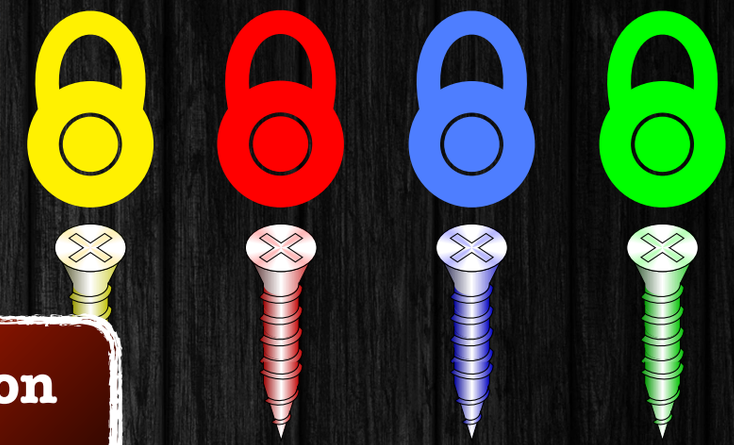
global lock

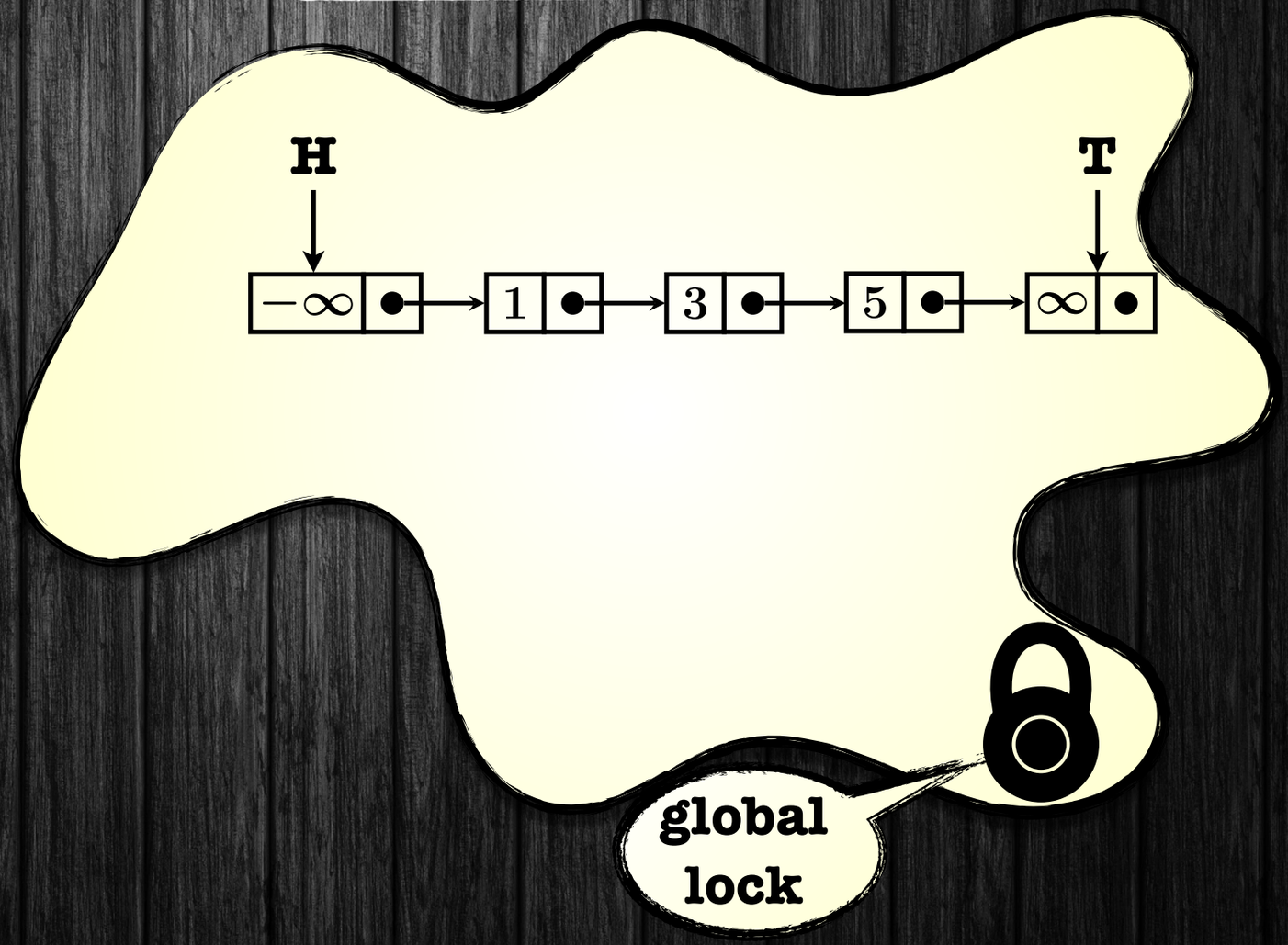
Sets | Coarse-Grained Synchronization

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

code + linearization point

method implementation





global lock

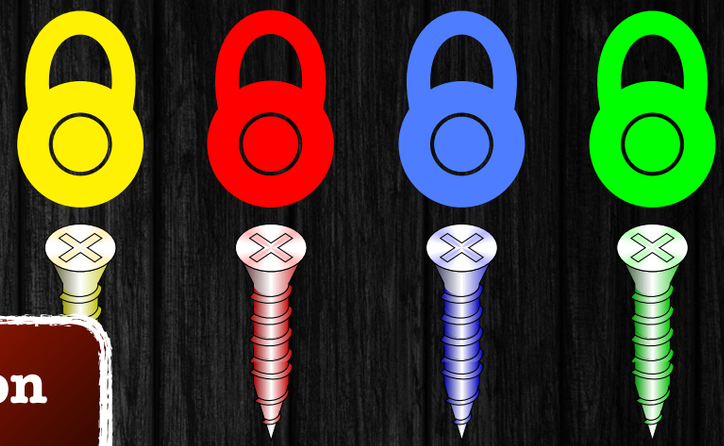
Sets | Coarse-Grained Synchronization

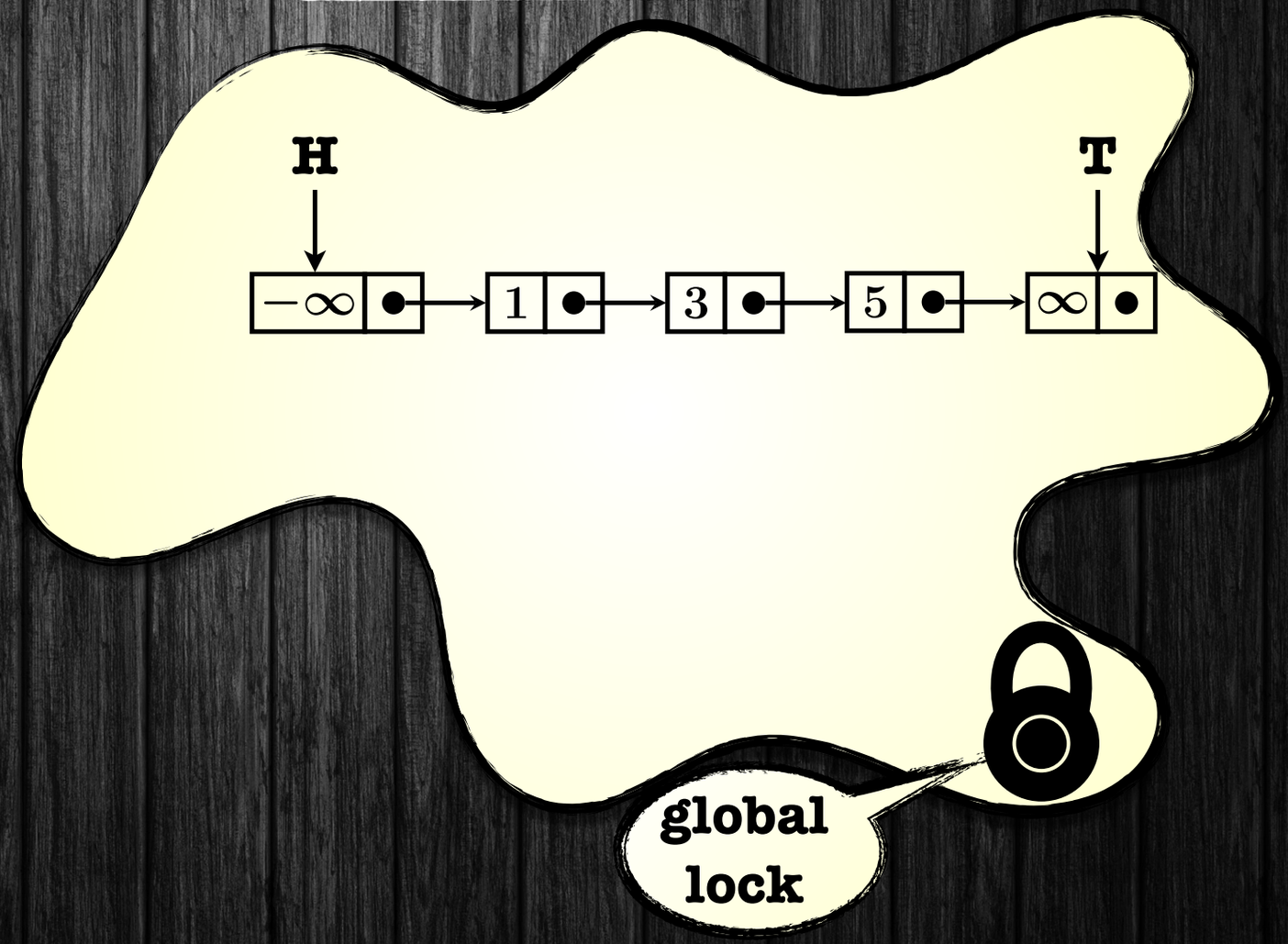
```
add(k):
Node p, c
Lock l
1  lock l        LP
2  p = H;
3  c = p.next
4  while (c.key < k)
5    p = c;
6    c = c.next;
7  if (c.key = k)
8    return false
9  else
10   n = new Node(k,-)
11   n.next = c
12   p.next = n
13   return true
14 unlock l
```

linearization point

code + linearization point

method implementation

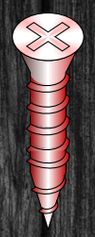




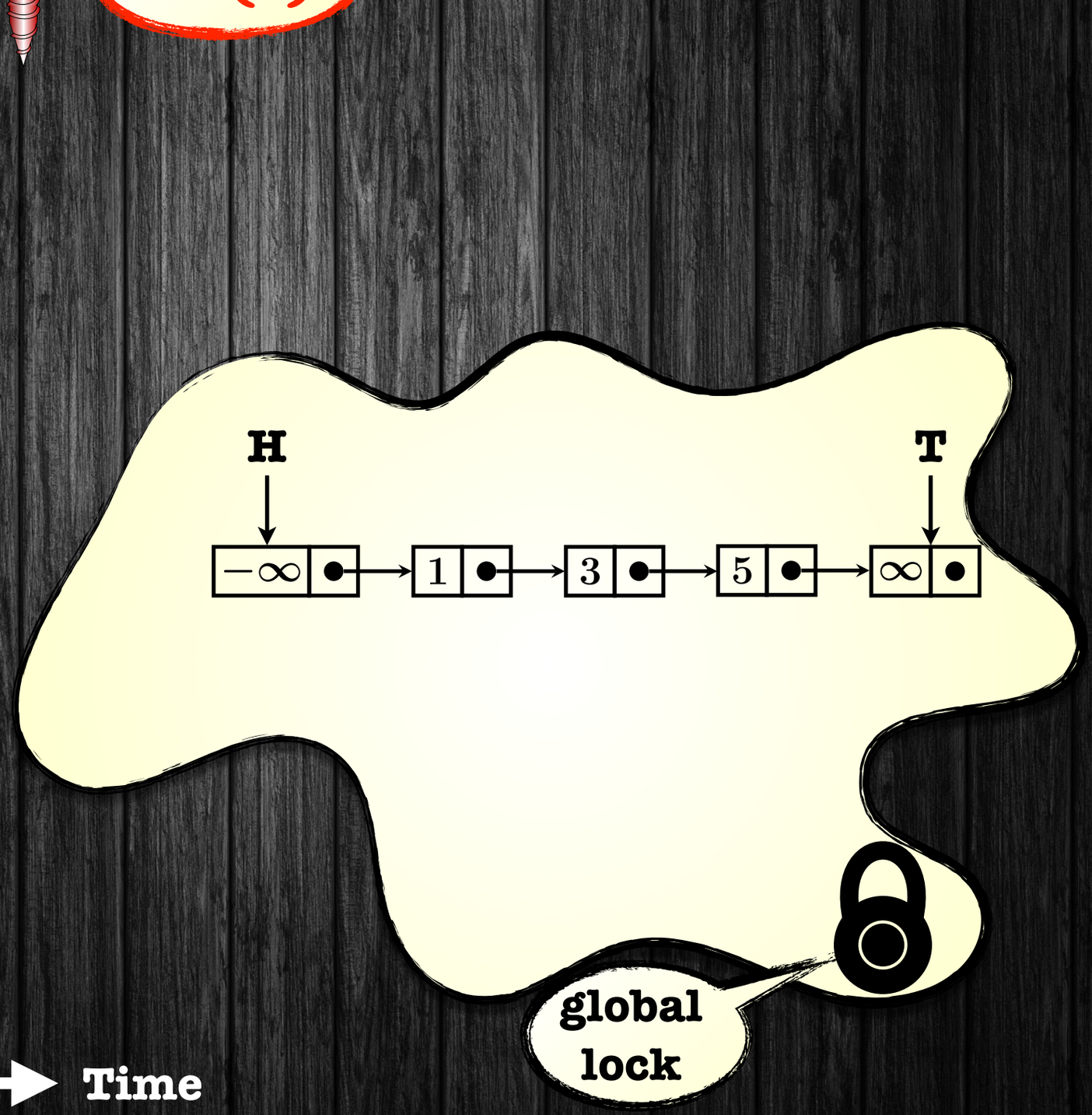
global lock

Sets | Coarse-Grained Synchronization

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



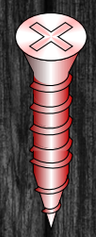
add(4)



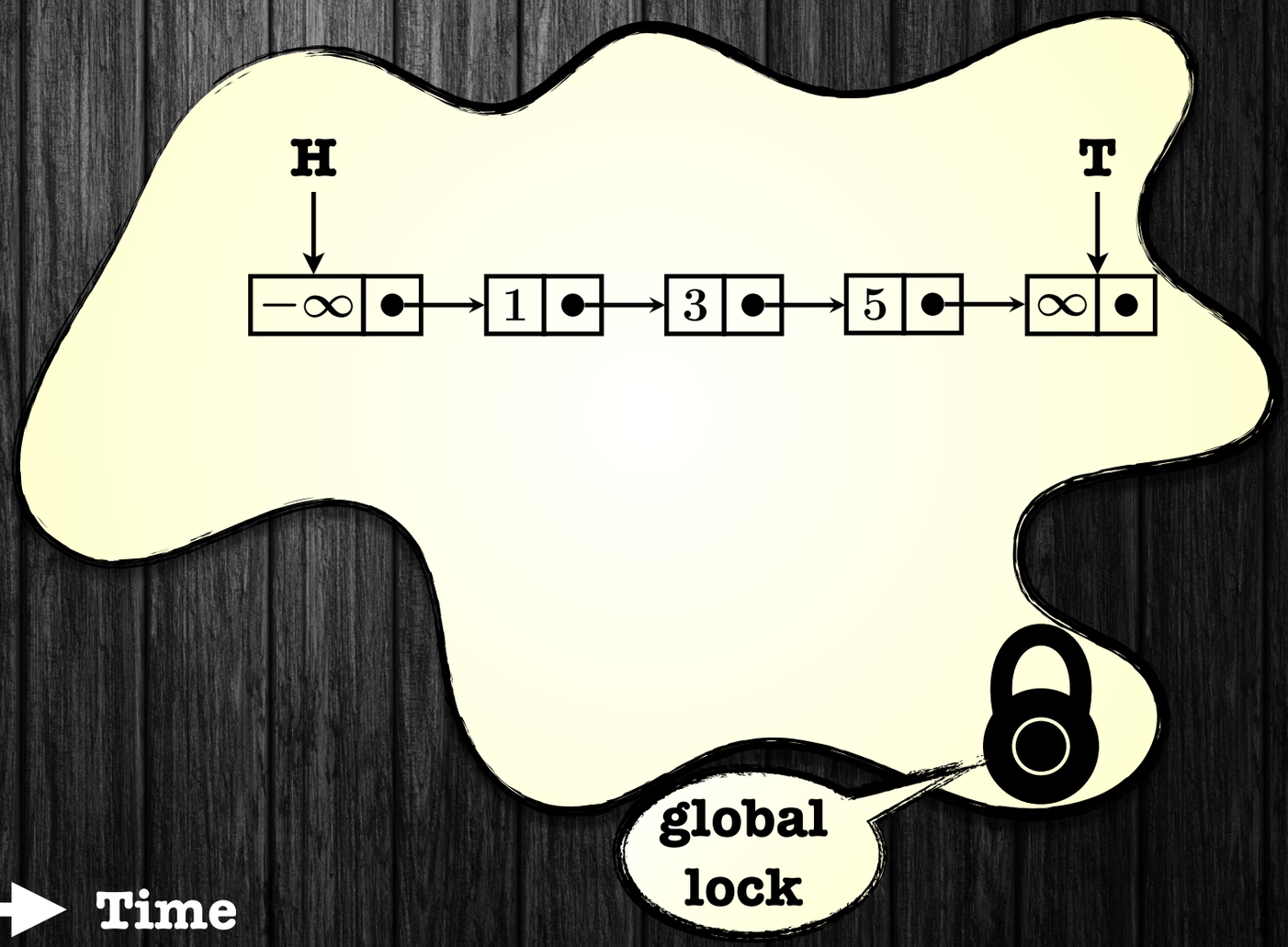


Sets | Coarse-Grained Synchronization

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



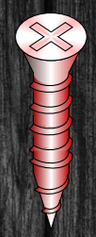
add(4)



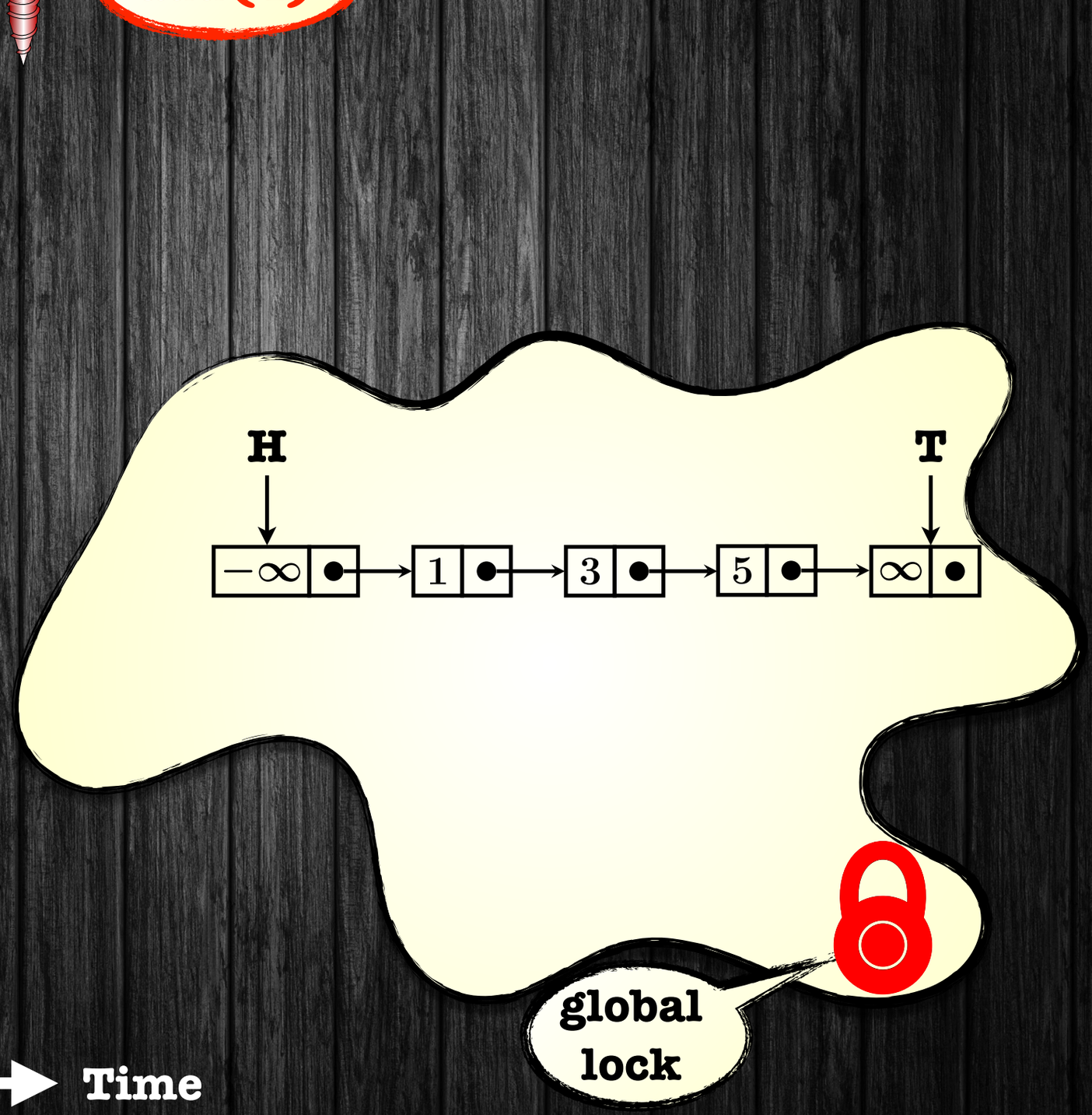


Sets | Coarse-Grained Synchronization

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



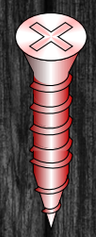
add(4)



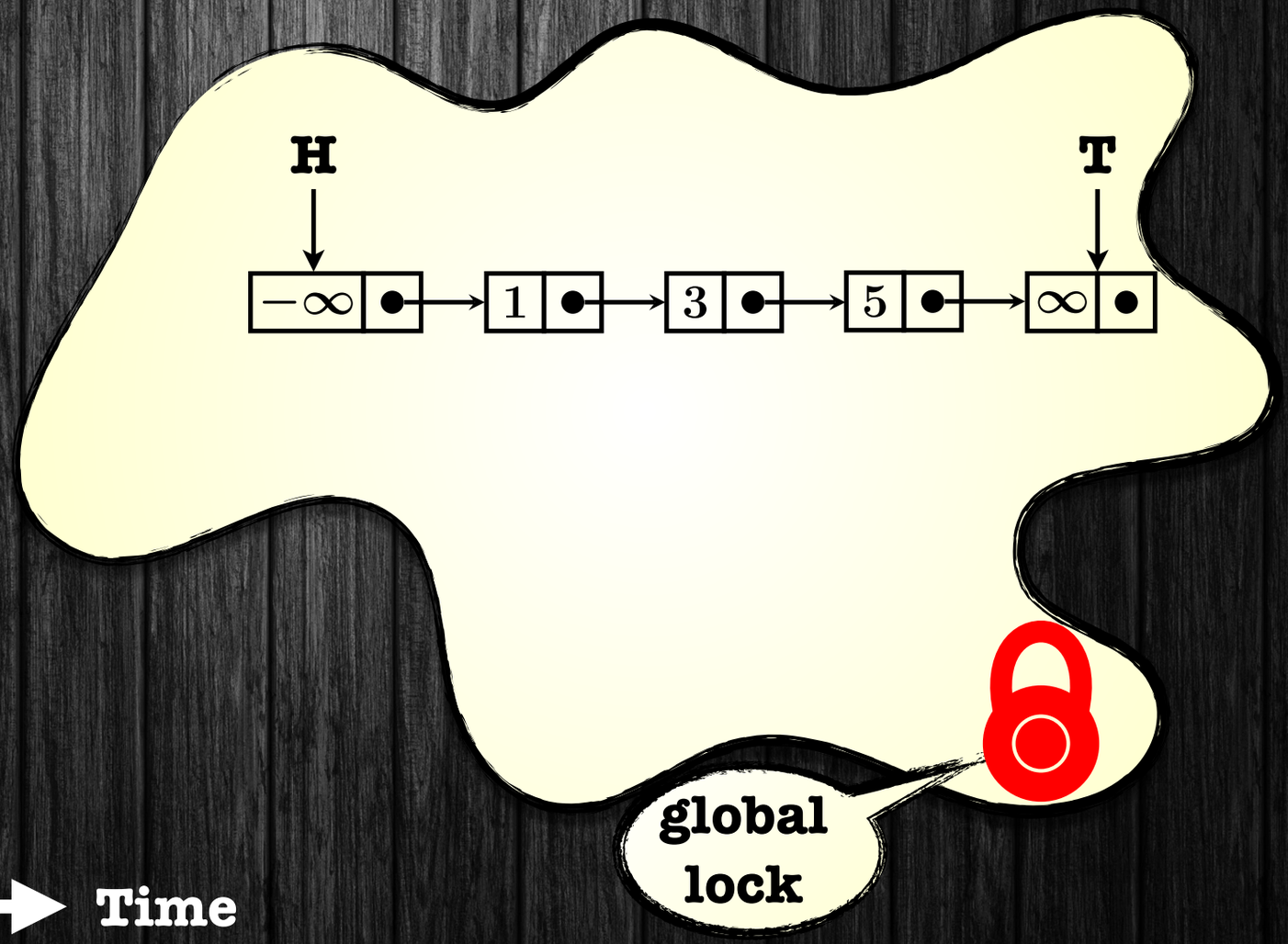


Sets | Coarse-Grained Synchronization

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



add(4)



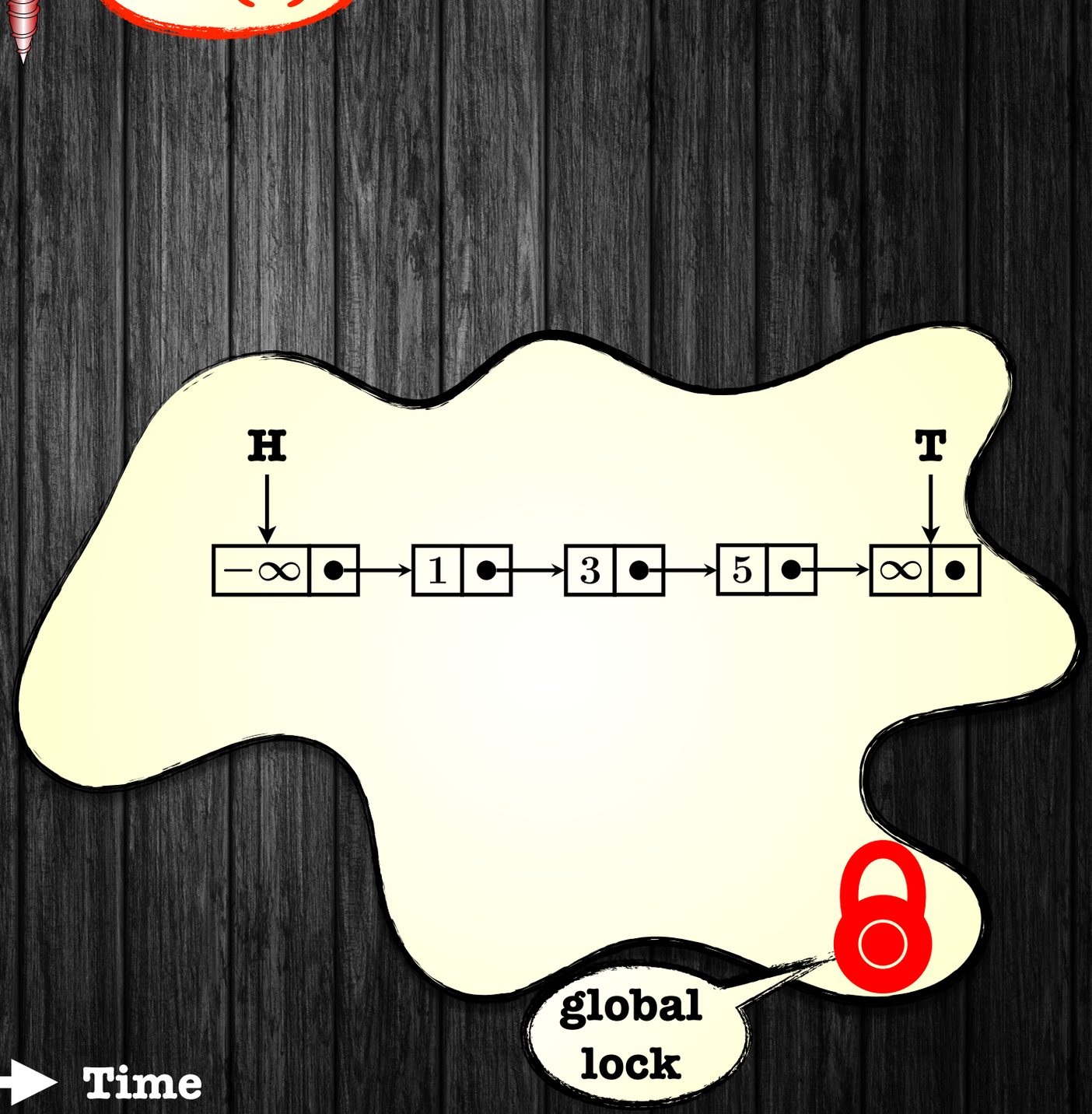


Sets | Coarse-Grained Synchronization

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



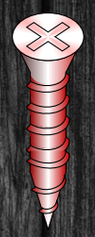
add(4)



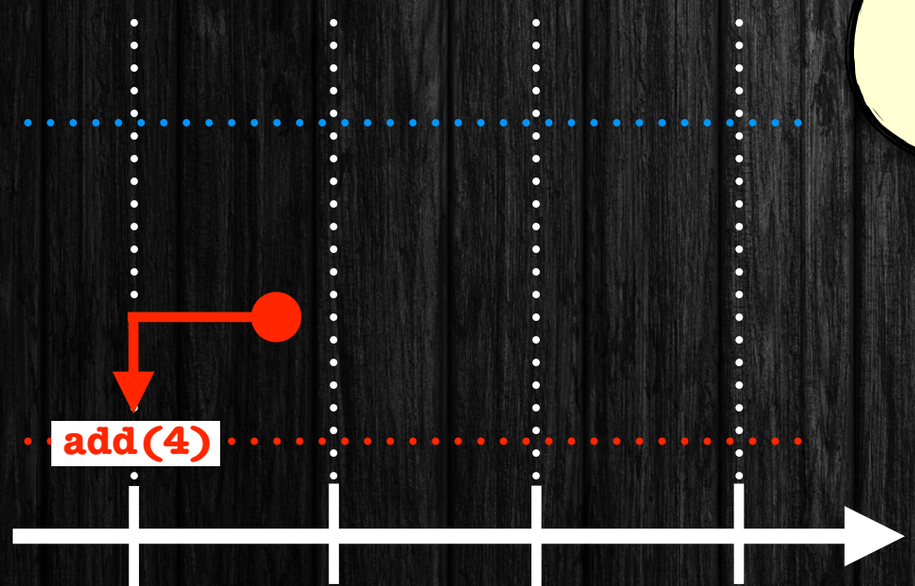


Sets | Coarse-Grained Synchronization

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

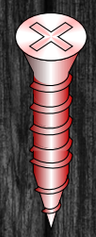


add(4)

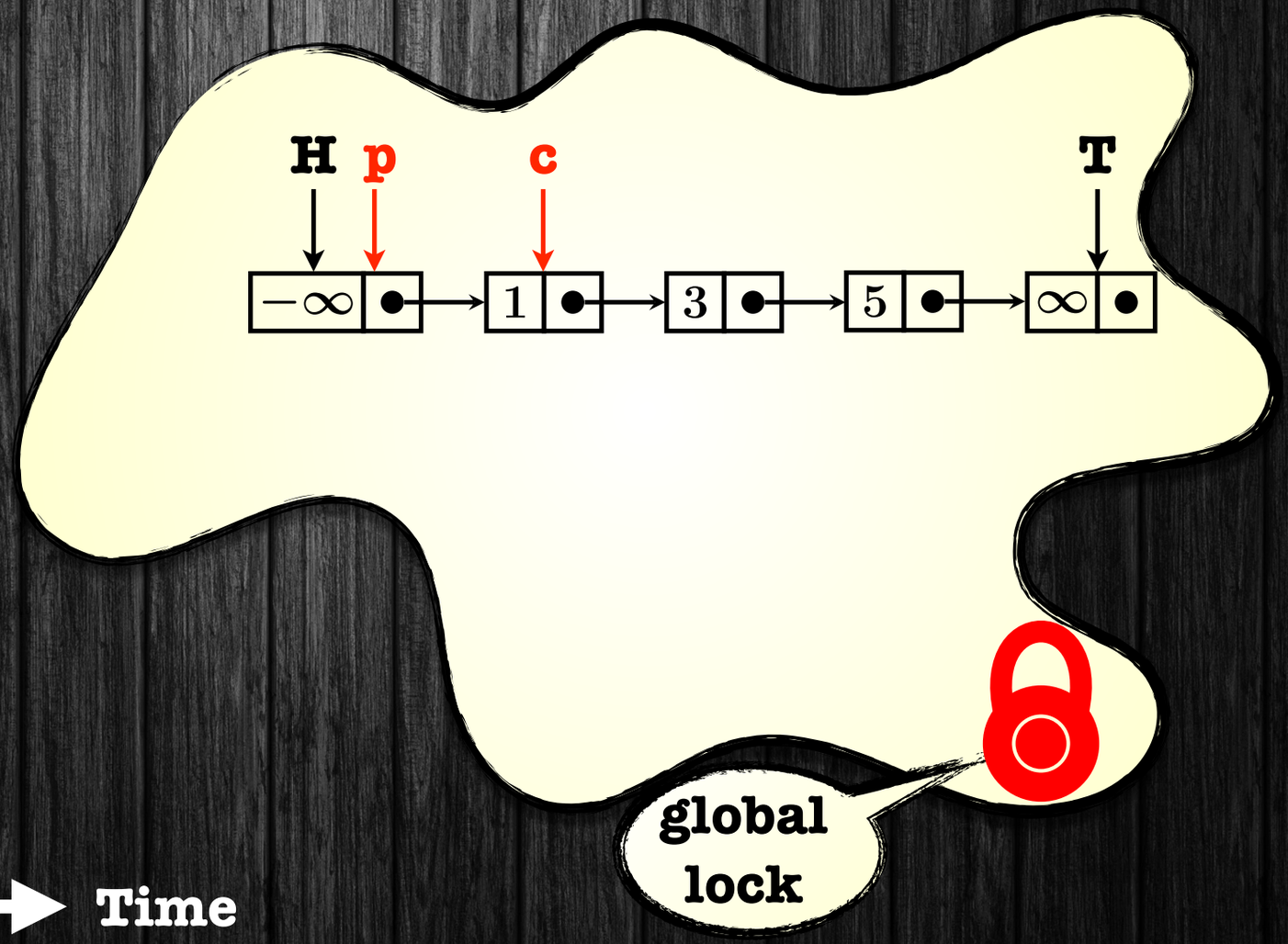


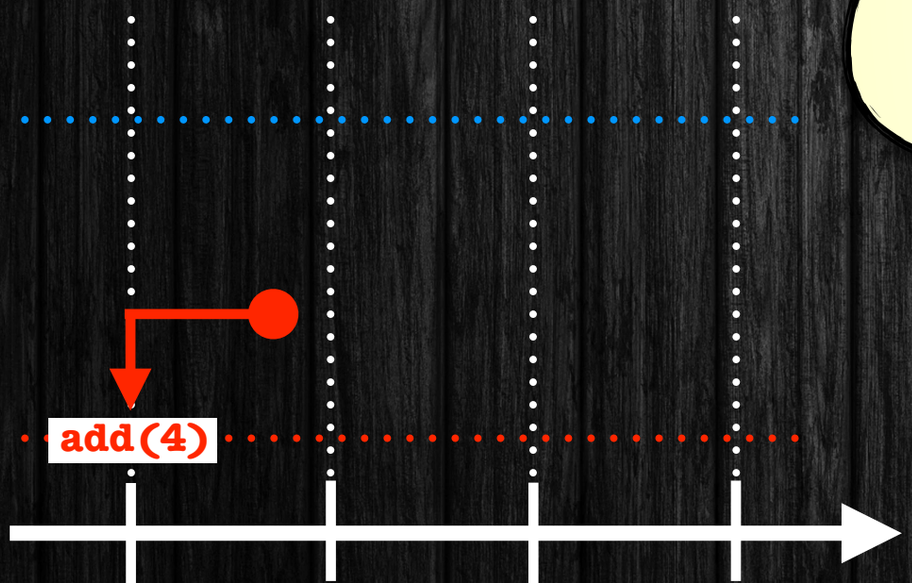
Sets | Coarse-Grained Synchronization

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



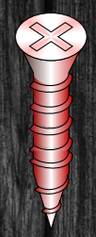
add(4)



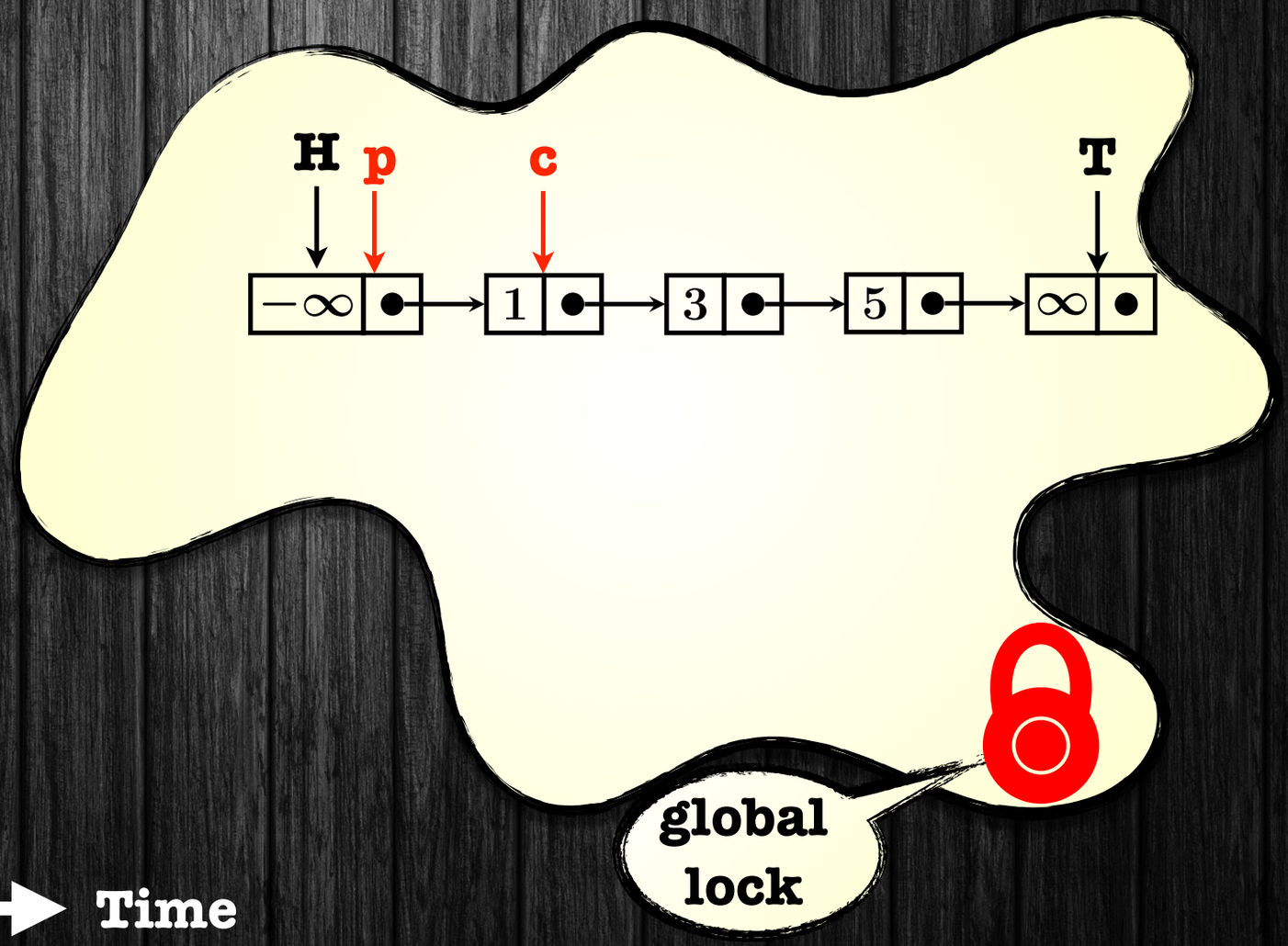


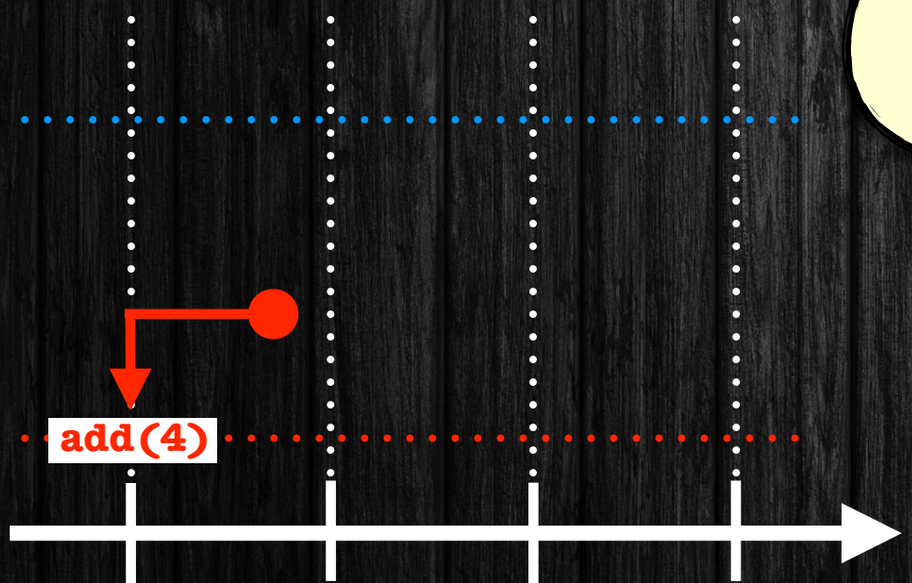
Sets | Coarse-Grained Synchronization

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



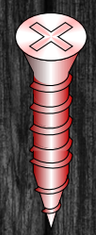
add(4)



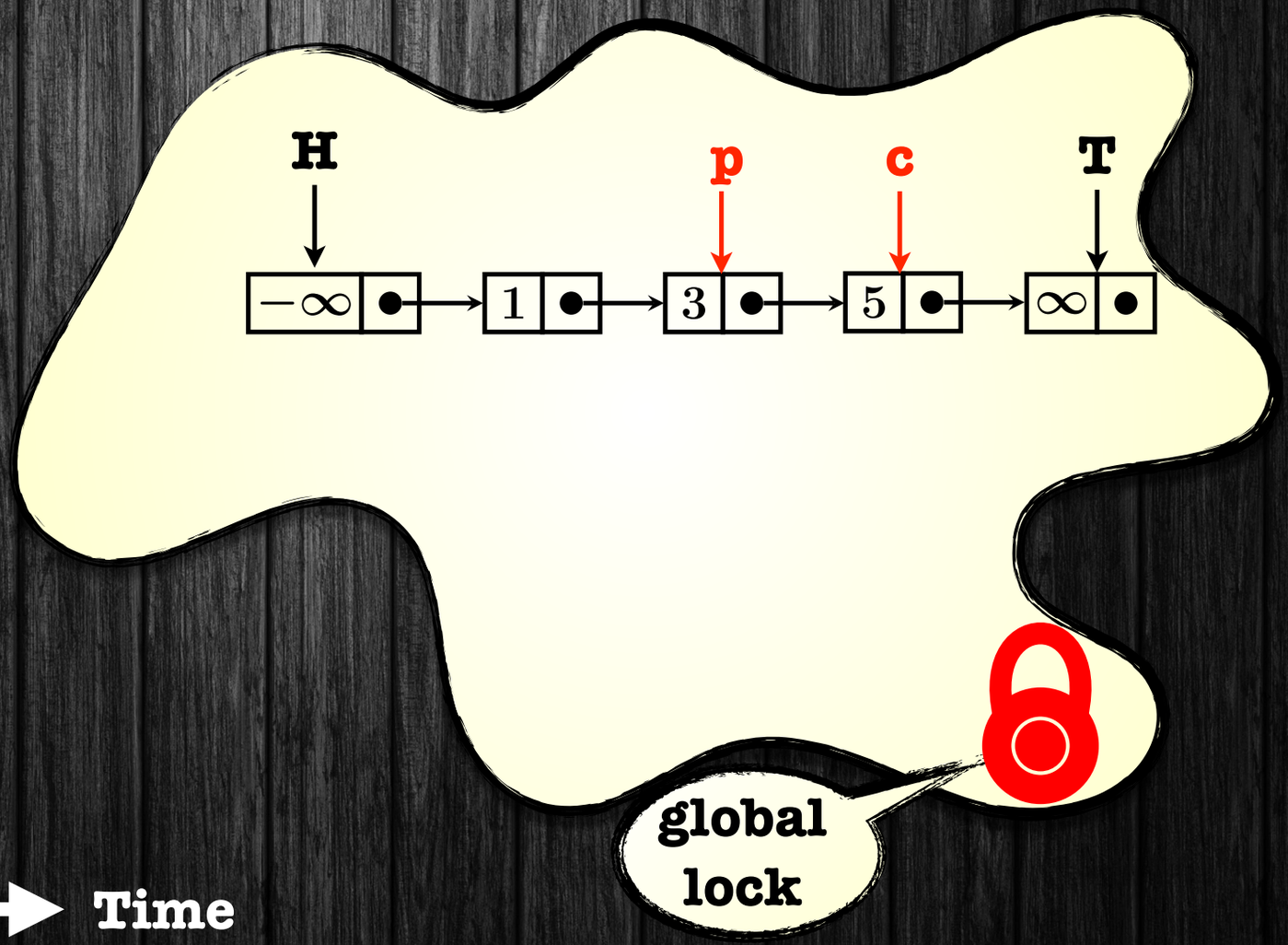


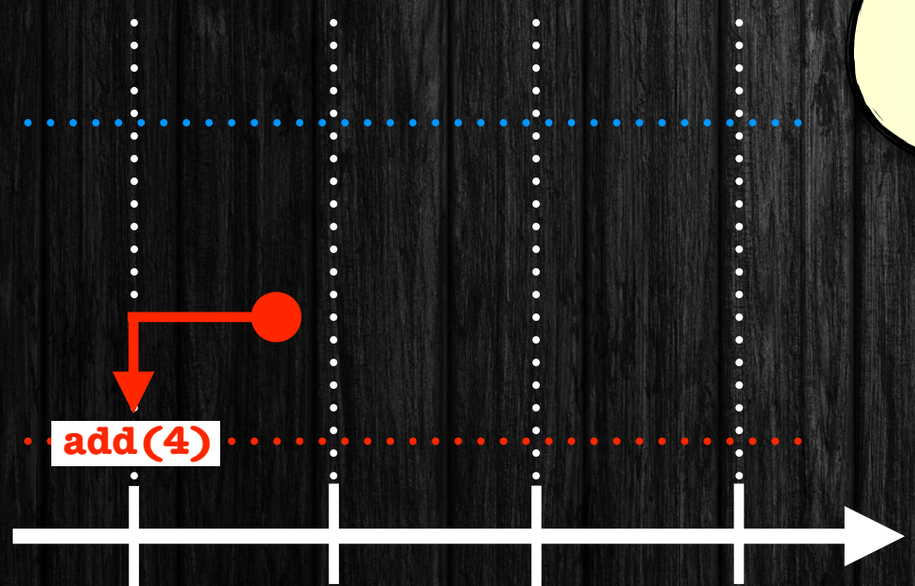
Sets | Coarse-Grained Synchronization

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



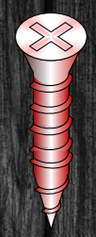
add(4)



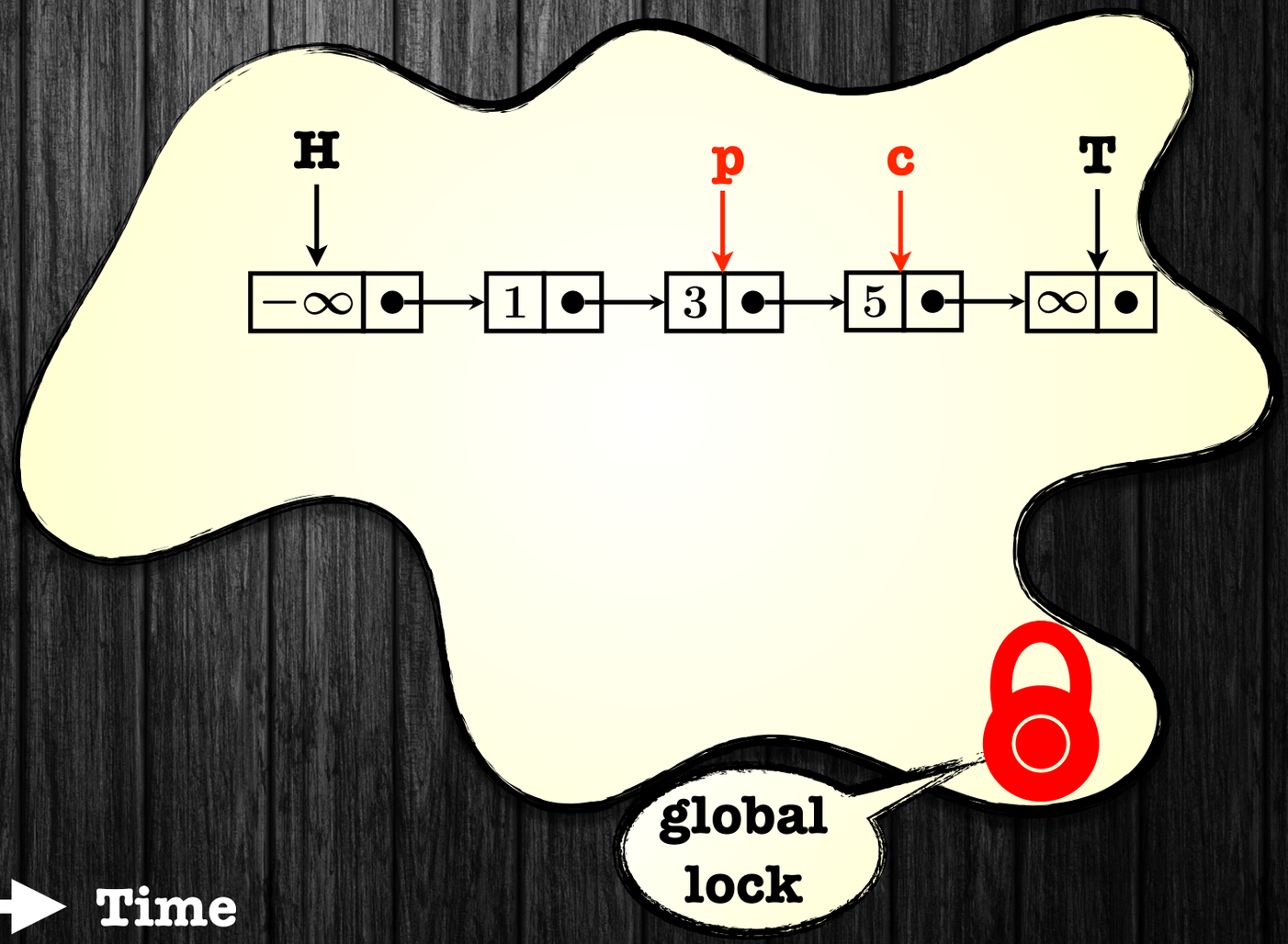


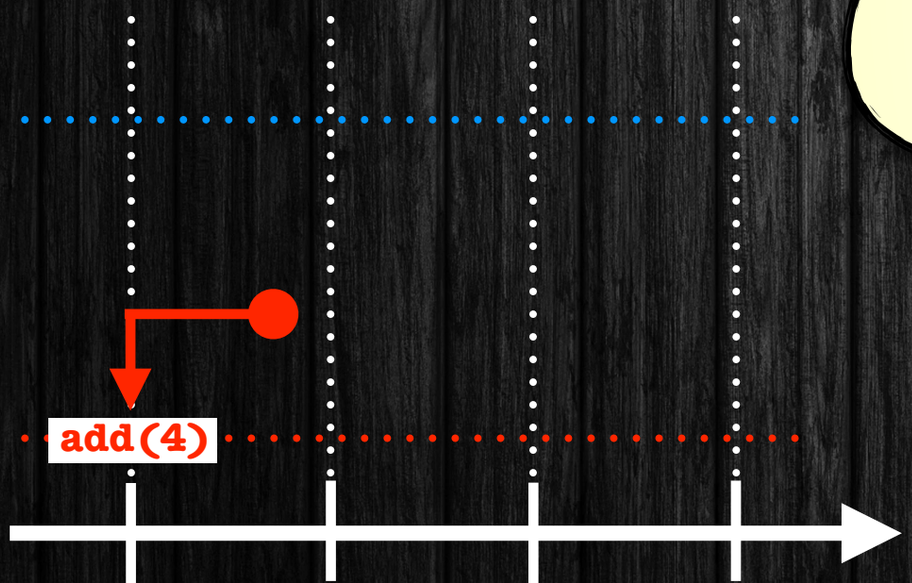
Sets | Coarse-Grained Synchronization

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



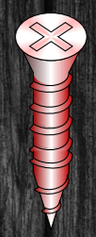
add(4)





Sets | Coarse-Grained Synchronization

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

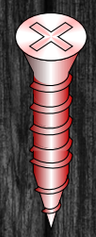


add(4)



Sets | Coarse-Grained Synchronization

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



add(4)

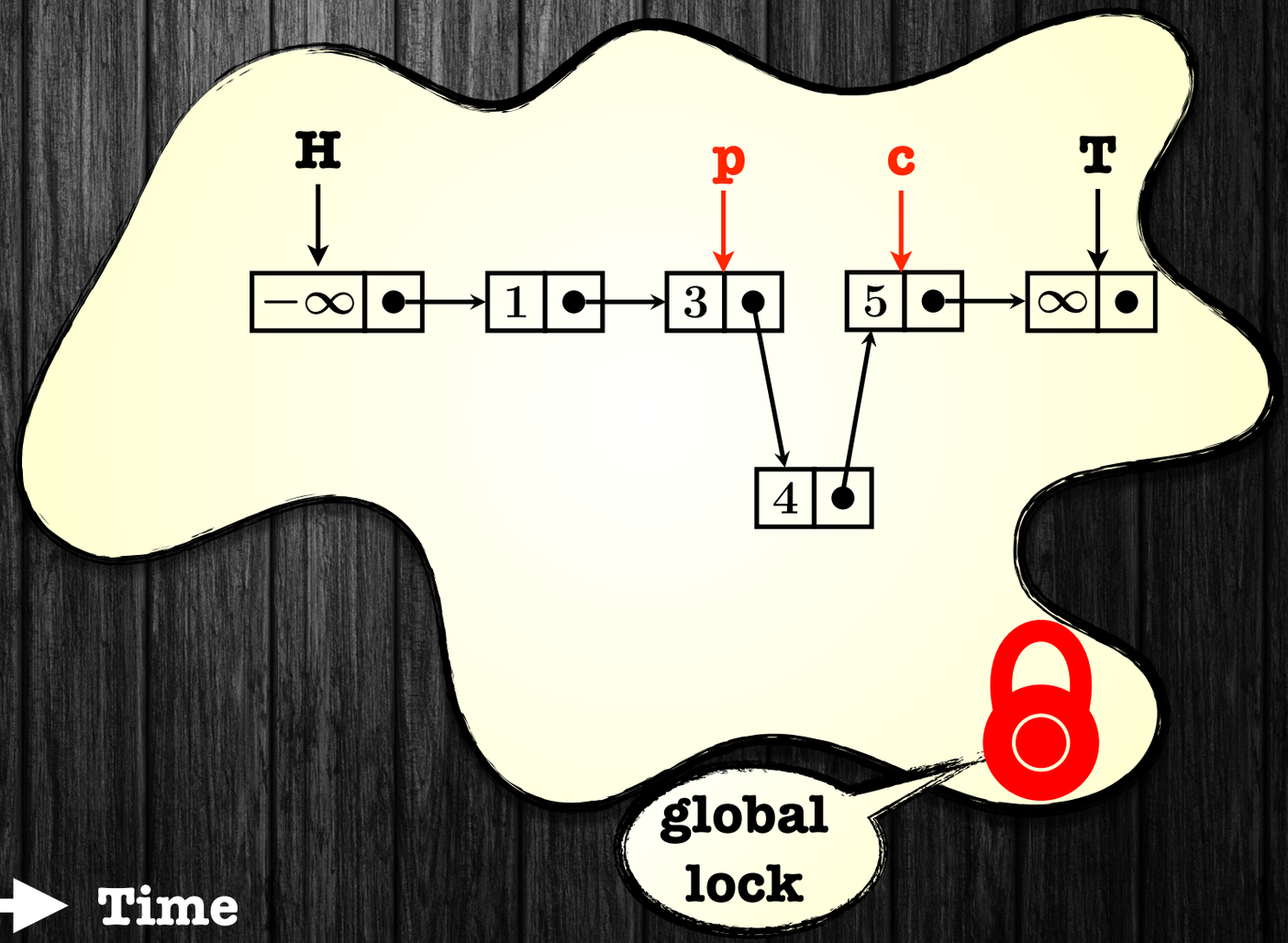


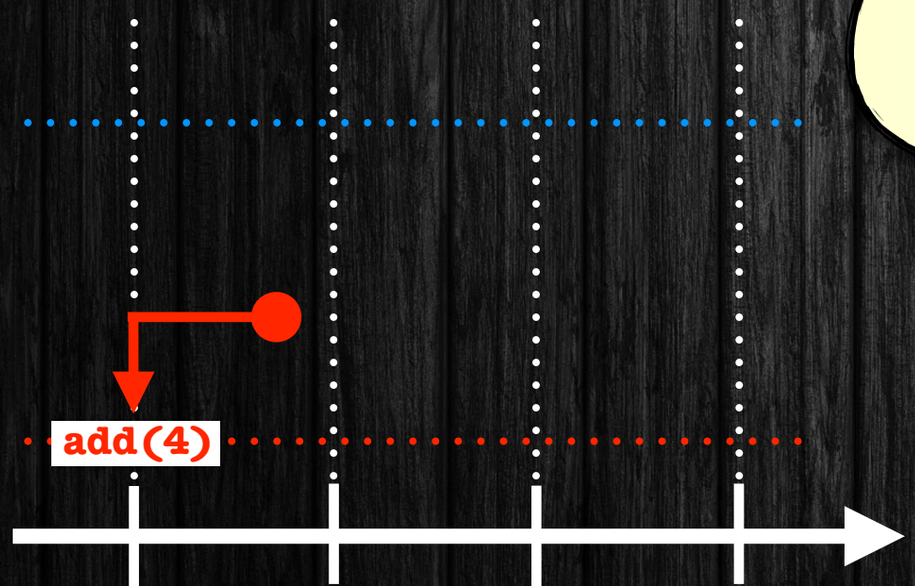
Sets | Coarse-Grained Synchronization

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



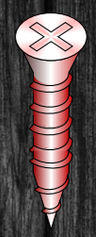
add(4)



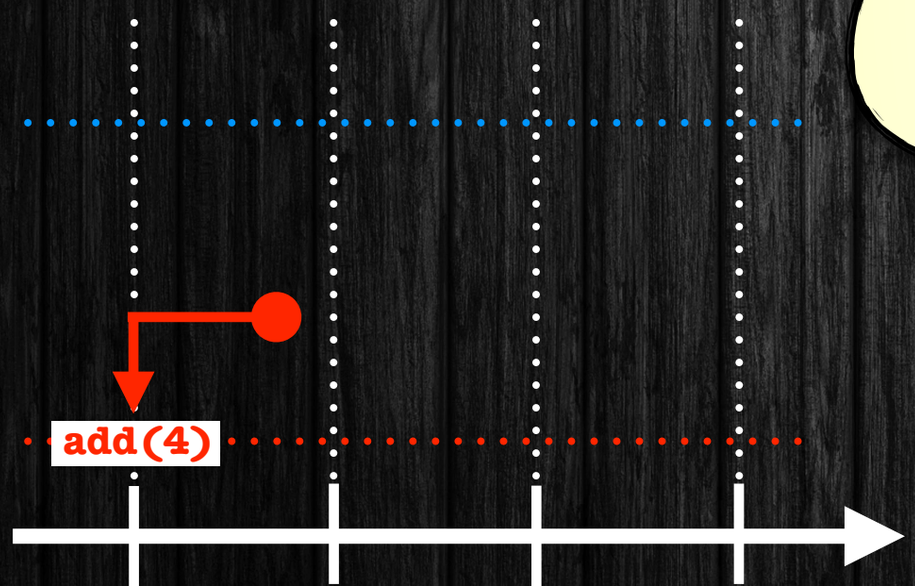


Sets | Coarse-Grained Synchronization

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

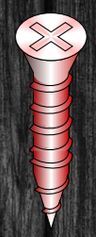


add(4)

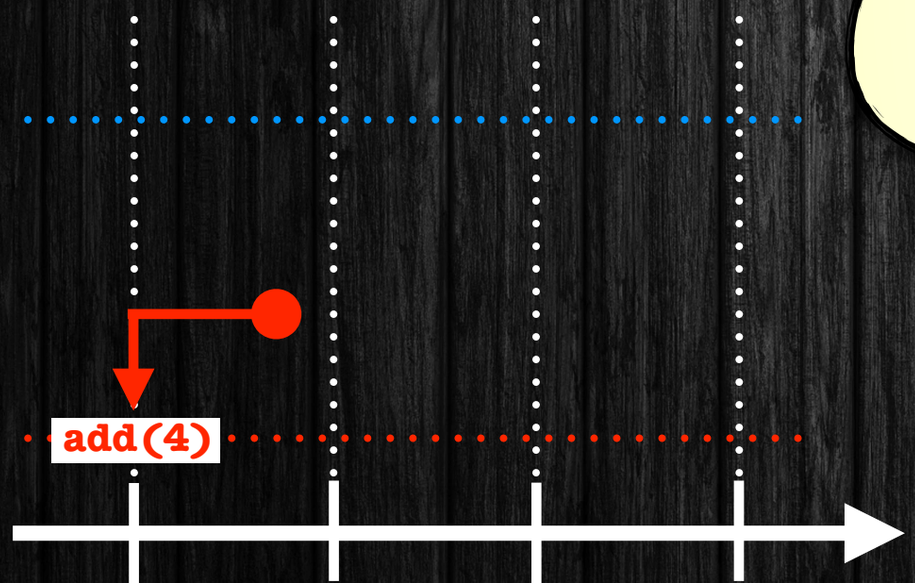


Sets | Coarse-Grained Synchronization

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

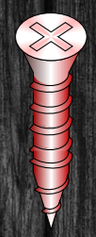


add(4)

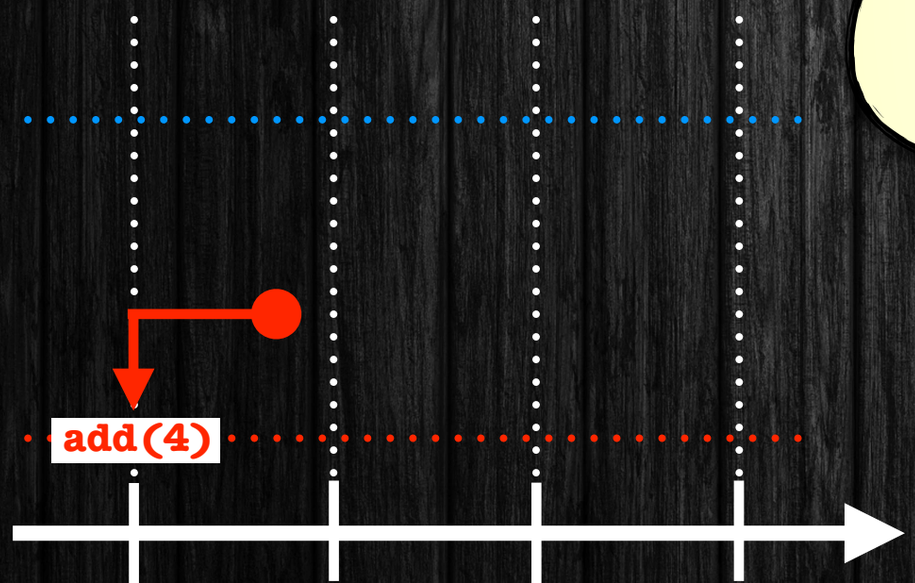


Sets | Coarse-Grained Synchronization

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

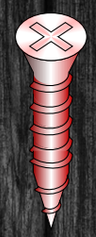


add(4)

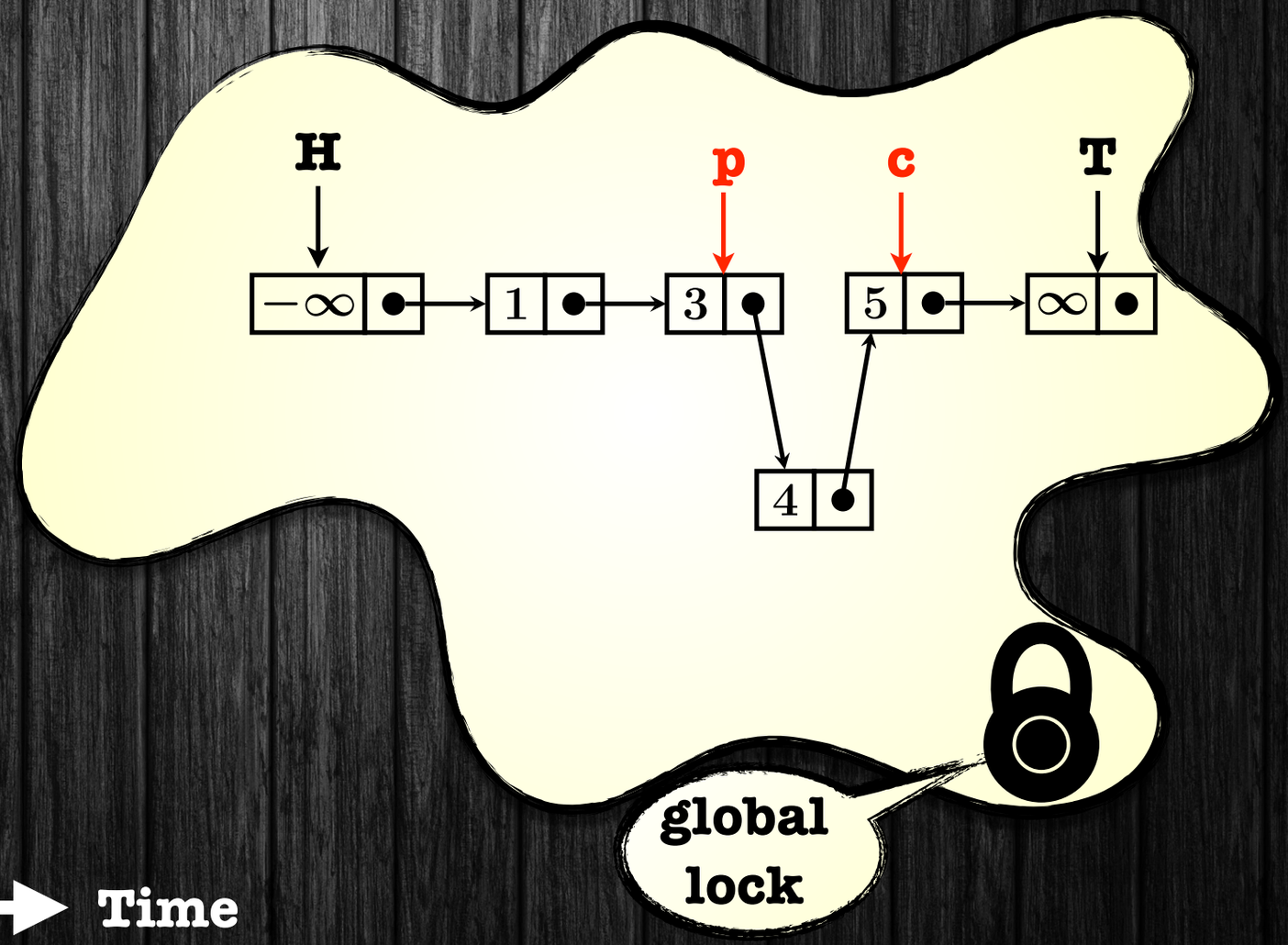


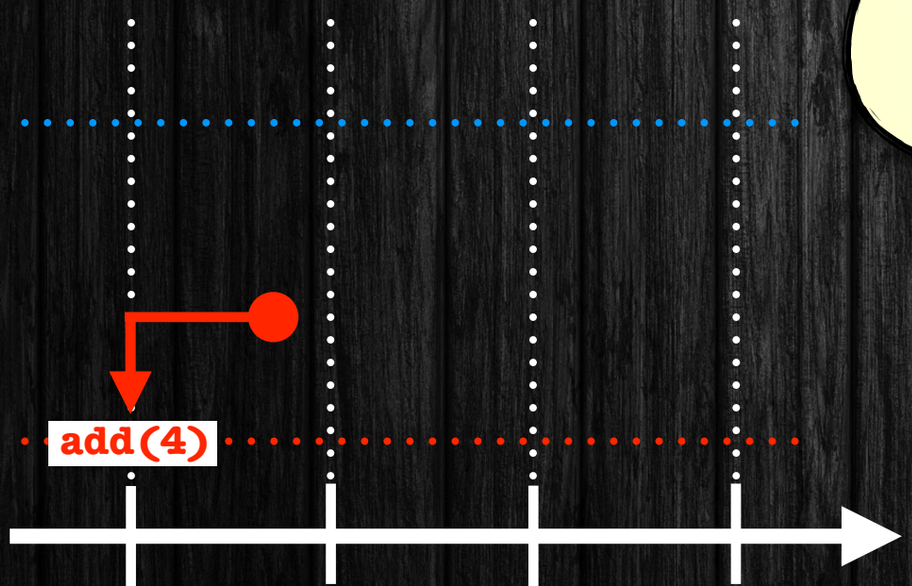
Sets | Coarse-Grained Synchronization

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



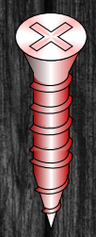
add(4)



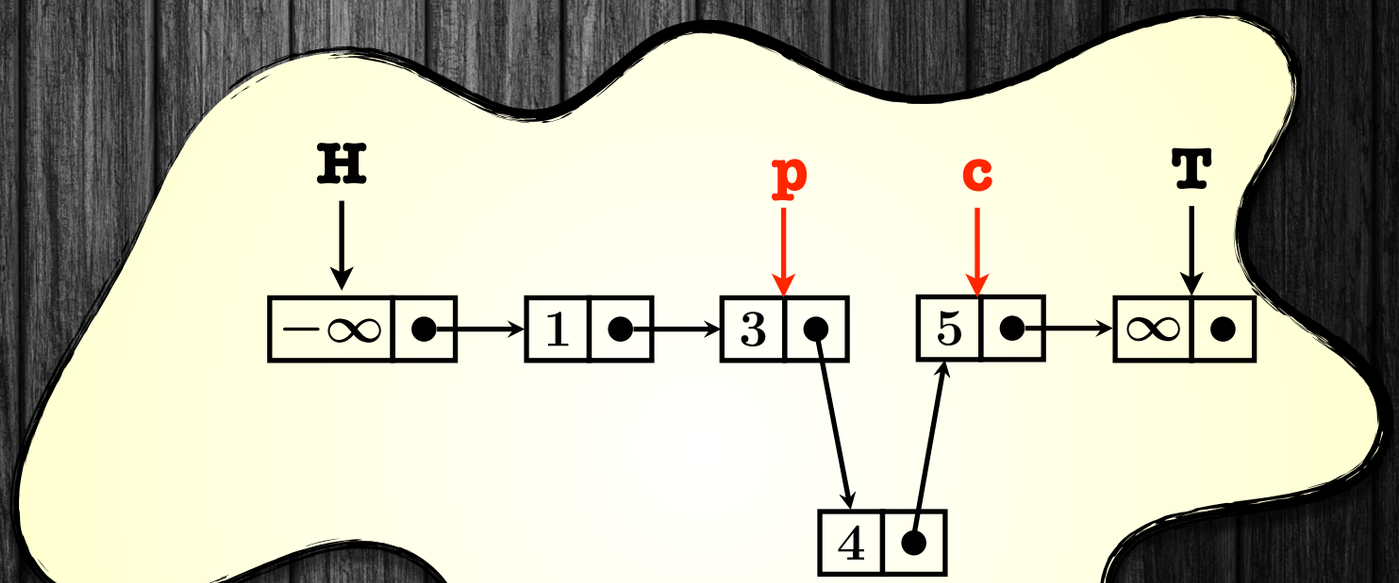


Sets | Coarse-Grained Synchronization

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



add(4)



global lock