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

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

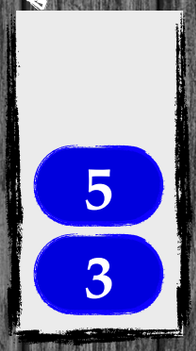
Overview

Example

Stack

(Basic) Algorithms & Data Structures

top of stack



push(4)

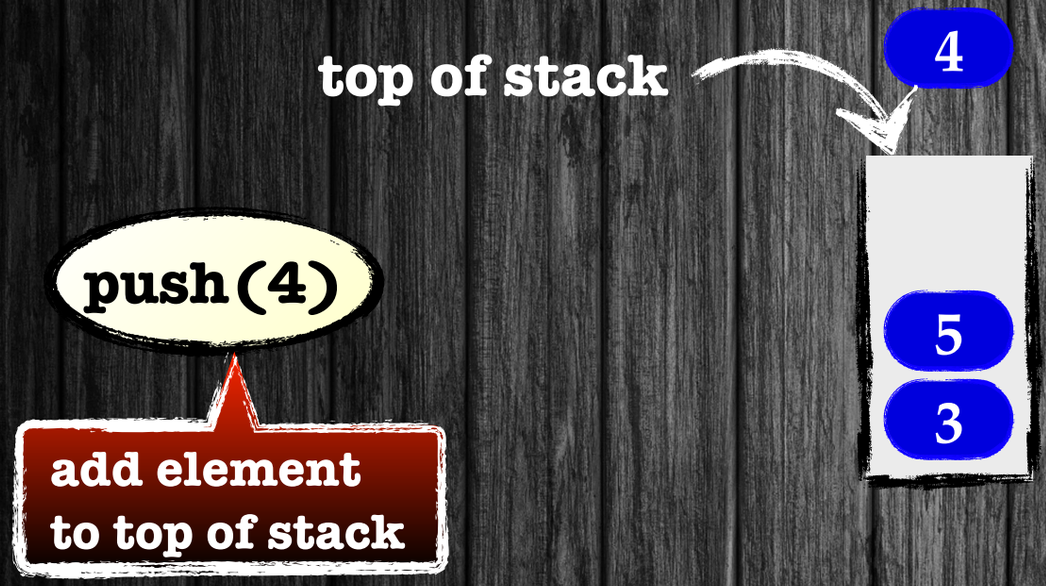
add element
to top of stack

sequential
setting

Overview

Example | Stack

(Basic) Algorithms & Data Structures



sequential setting

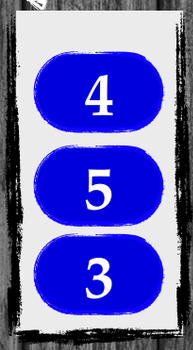
Overview

Example

Stack

(Basic) Algorithms & Data Structures

top of stack



sequential setting

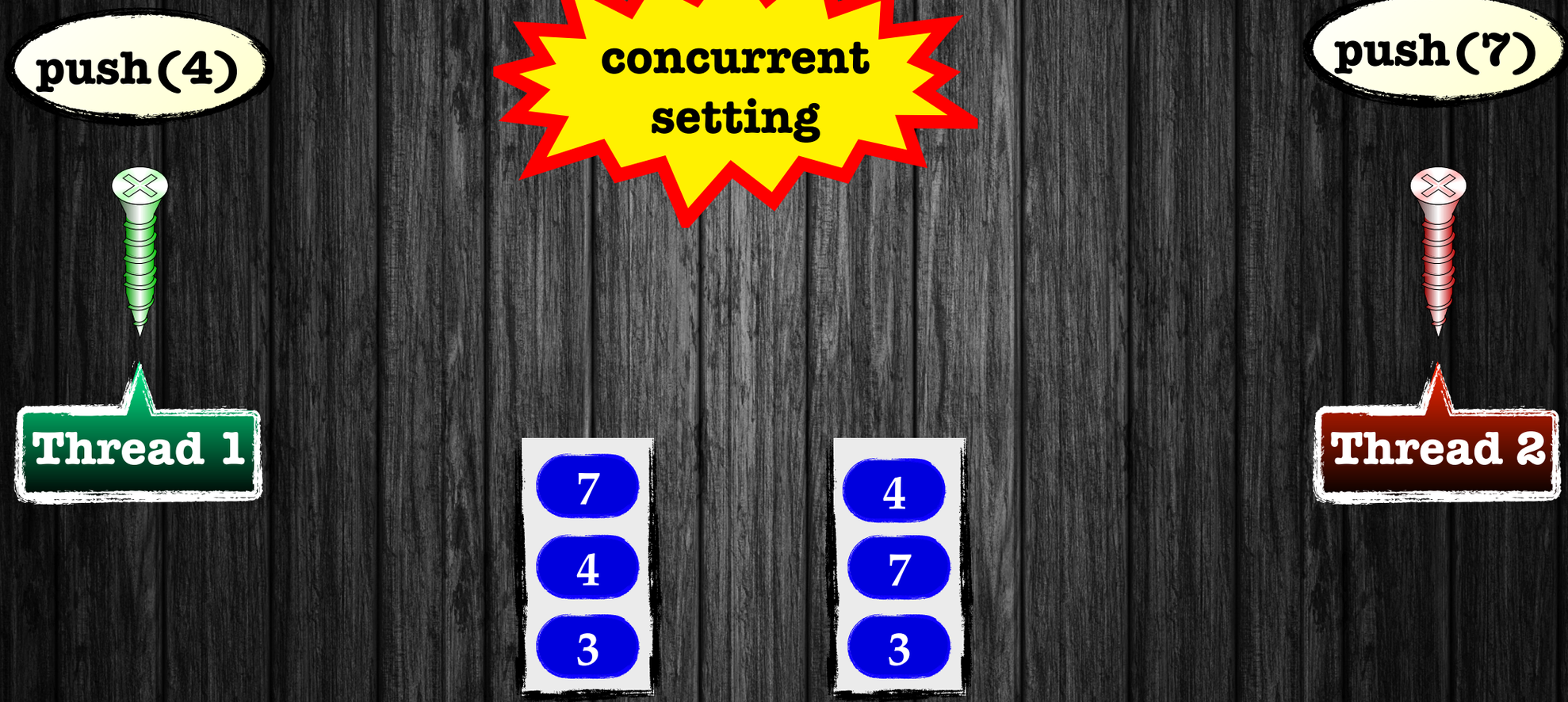
push(4)

add element to top of stack

Overview | Goal → concurrent setting

Example | Stack

(Basic) Algorithms & Data Structures



Overview **Goal** → **concurrent setting**

Example **Stack** **(Basic) Algorithms & Data Structures**

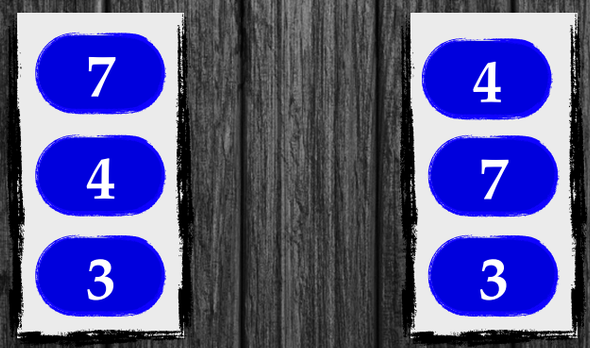
concurrent setting

push(4)

Thread 1

push(7)

Thread 2

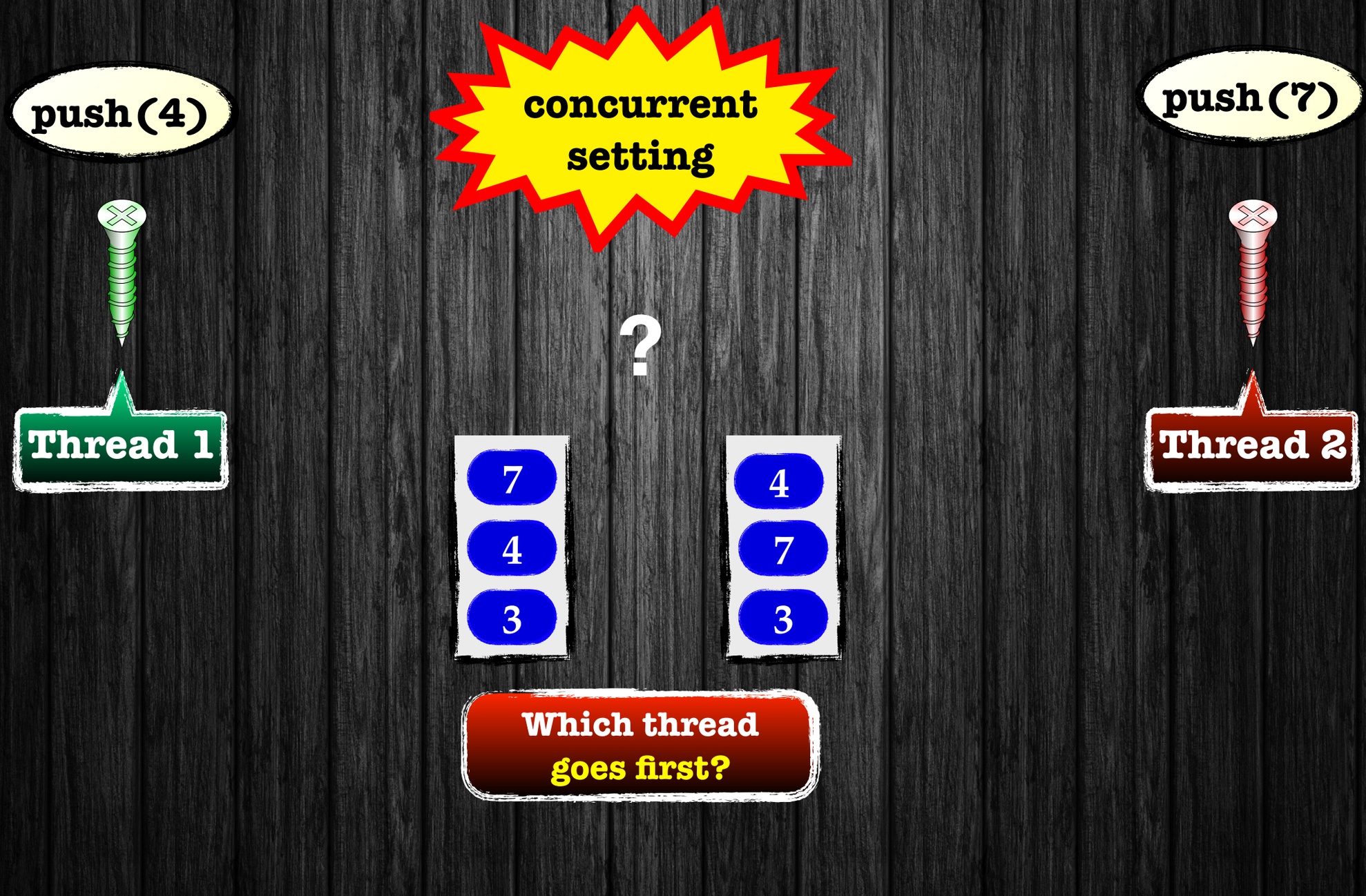


Which thread goes first?

Overview | Goal → concurrent setting

Example | Stack

(Basic) Algorithms & Data Structures



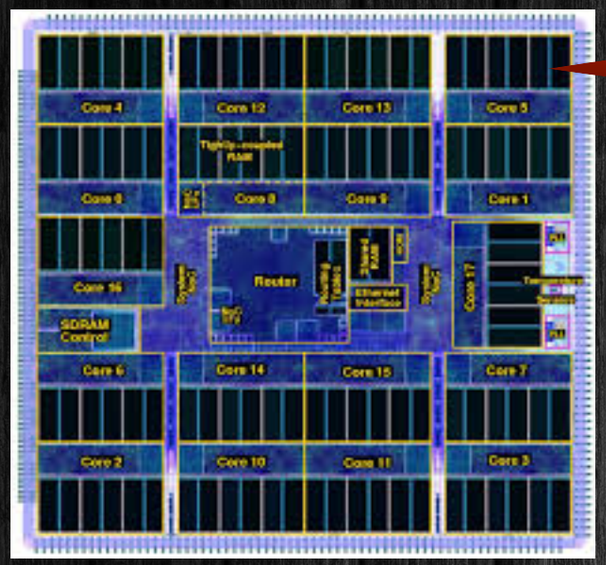
Algorithmics

Concurrency

Concurrent systems are everywhere

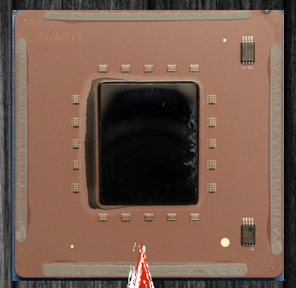
Multicore architectures

motivation









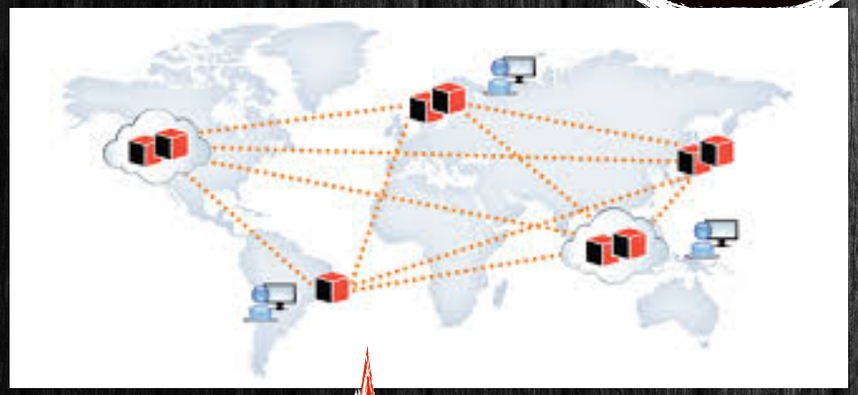




intel

ARM

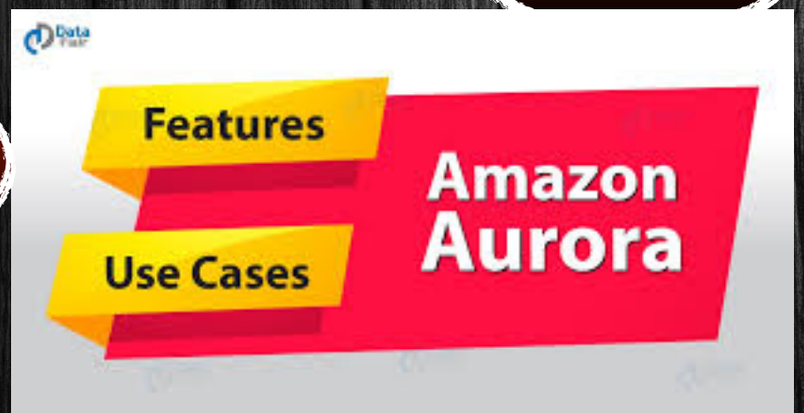
IBM Power



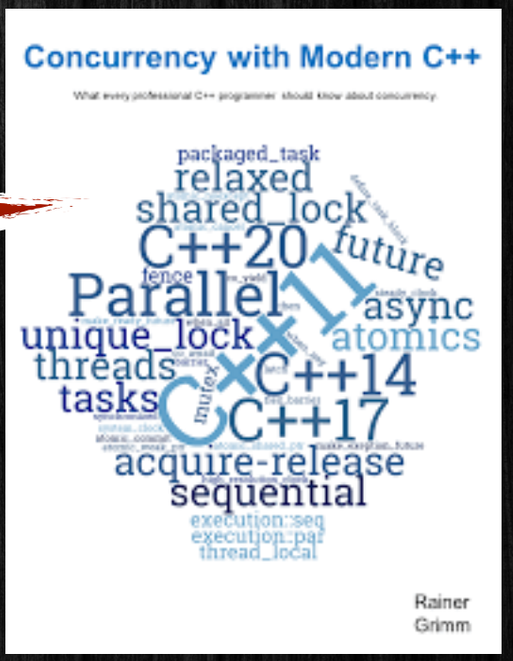
Distributed databases



Facebook



Programming languages



Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

Safety Properties

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

Examples

Sorting

list $a=[a_1,a_2,\ldots,a_n]$ of integers — input

list $b=[b_1,b_2,\ldots,b_n]$ of integers — output

- b is a permutation of a
- b is sorted

Reachability of bad states

input: [3,7,8,5,1]

output: [1,3,8,5,7]

sequential setting

Specification

- safety properties
- liveness properties

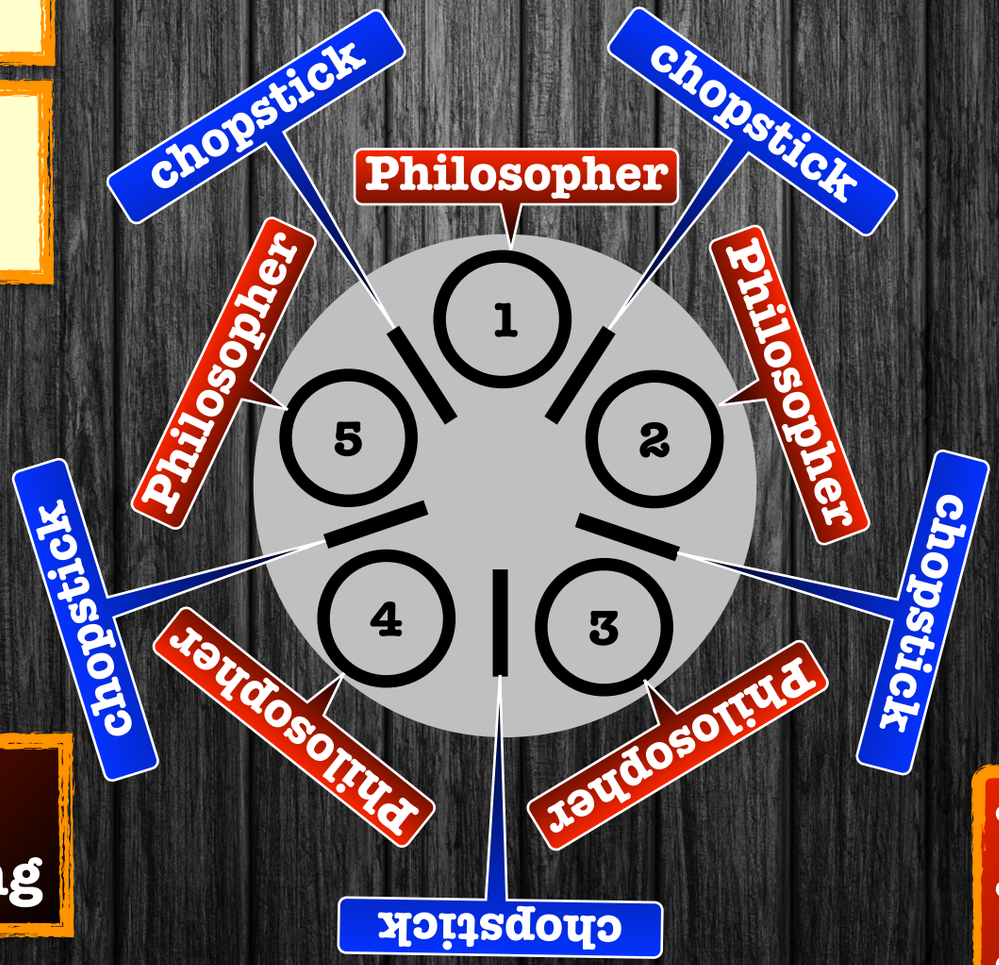
"abstract description of the expected behavior of the program"

Safety Properties

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

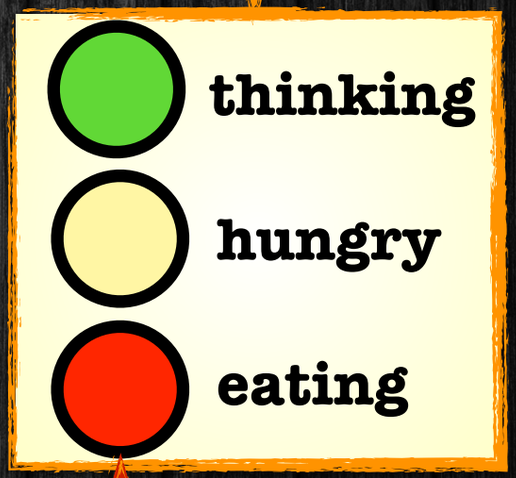
Examples

The Dining Philosophers



Initially:
all philosophers thinking

philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

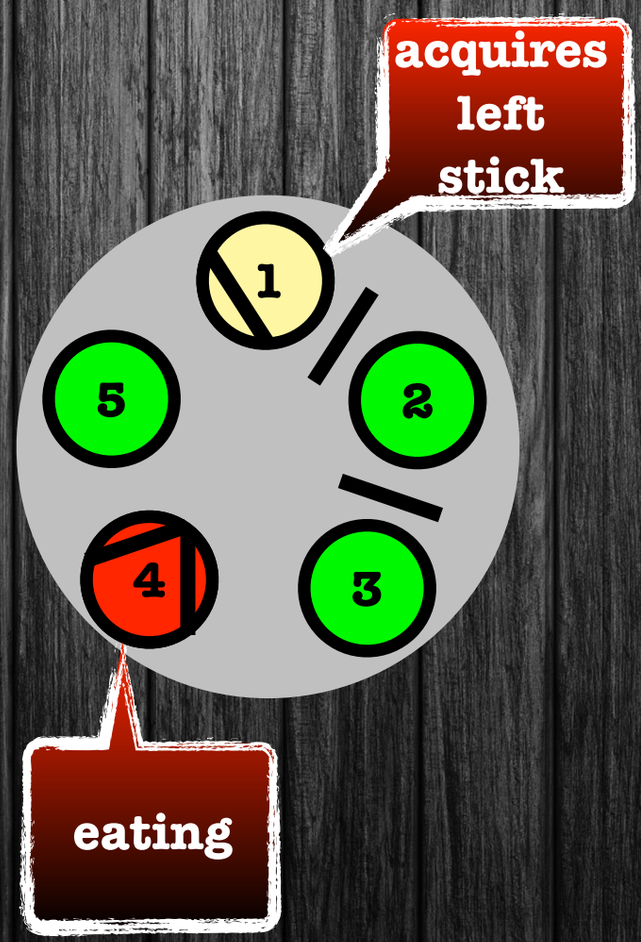
"abstract description of the expected behavior of the program"

Safety Properties

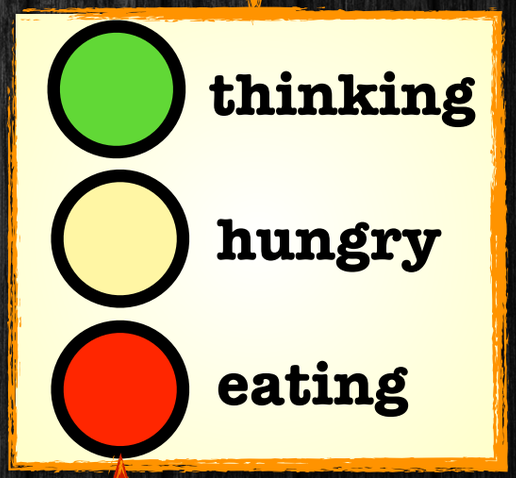
"nothing bad will happen during the execution of the program"

Examples

The Dining Philosophers



philosopher states



- acquire right stick
- acquire left stick
- start eating

Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

"we will not reach a deadlock state"

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

No Deadlock

concurrent setting

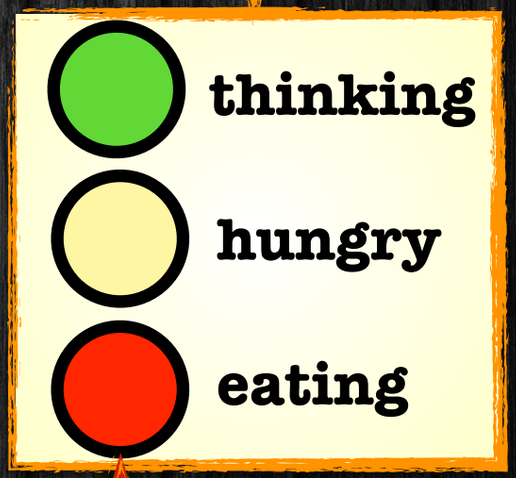
Safety Properties

Examples

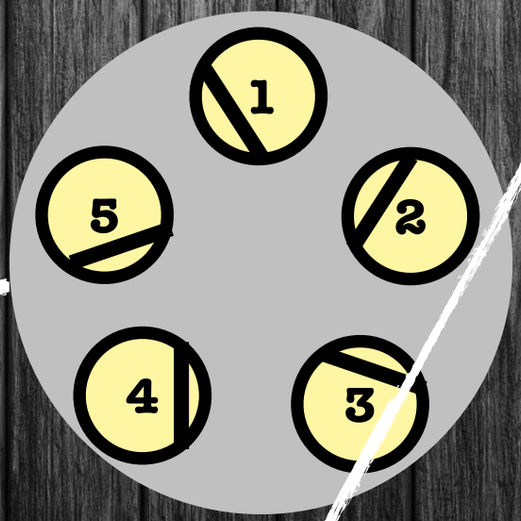
The Dining Philosophers

Reachability of bad states

philosopher states



- acquire right stick
- acquire left stick
- start eating



each philosopher waiting for the others

system cannot progress

Deadlock!

Specification

- safety properties
- liveness properties

concurrent setting

"abstract description of the expected behavior of the program"

"at most one process can access the shared resource at any point"

Safety Properties

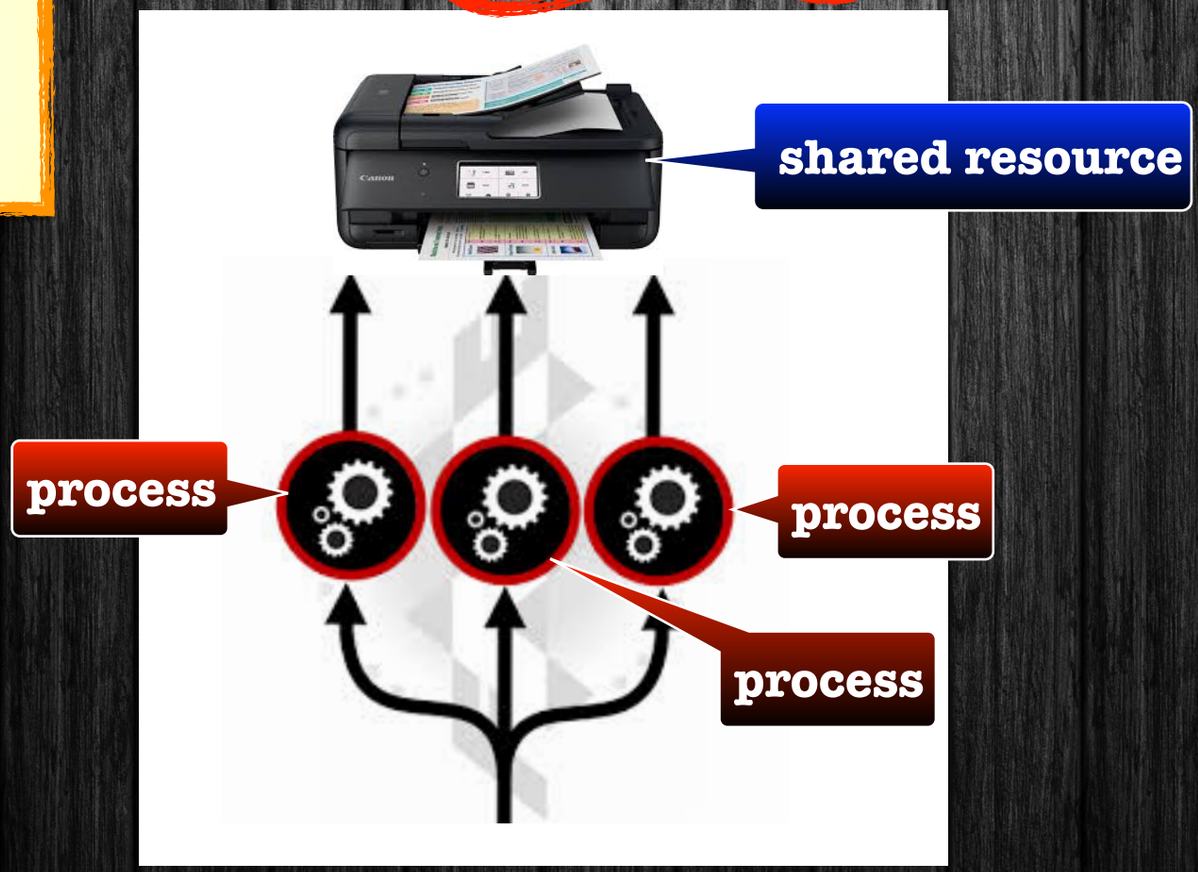
"nothing bad will happen during the execution of the program"

Examples

Mutual Exclusion

Reachability of bad states

"two or more processes access the shared resource at some point"



Specification

- safety properties
- liveness properties

"abstract description of the expected behavior of the program"

concurrent setting

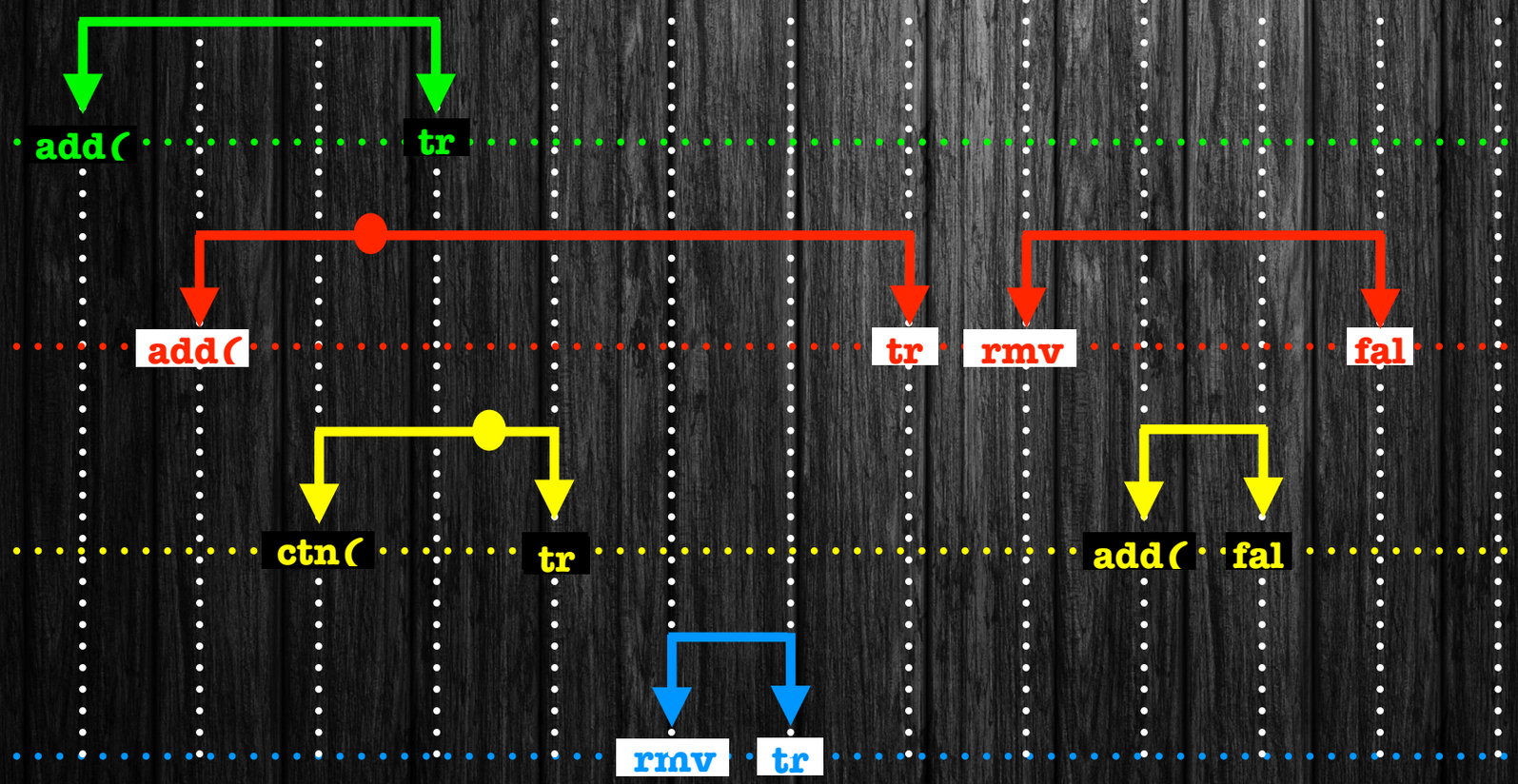
this course

Safety Properties

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

Examples

Reachability of bad states



Linearizability

Specification

- safety properties
- liveness properties

sequential setting

Liveness Properties

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

Examples

Termination

algorithms satisfying the "sorting" specification

"algorithm should terminate"

- Quick Sort
- Merge Sort
- Heap Sort
- etc

list $a=[a_1,a_2,\ldots,a_n]$ of integers

input

list $b=[b_1,b_2,\ldots,b_n]$ of integers

output

- b is a permutation of a
- b is sorted

$b_1 \leq b_2 \leq \ldots \leq b_n$

[3,0,5,8,7]

[0,3,5,7,8]

sorted permutation

Specification

- safety properties
- liveness properties

concurrent setting

Liveness Properties

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

Examples

Starvation-Freedom

"a philosopher who becomes hungry will eventually eat"



Specification

- safety properties
- liveness properties

concurrent setting

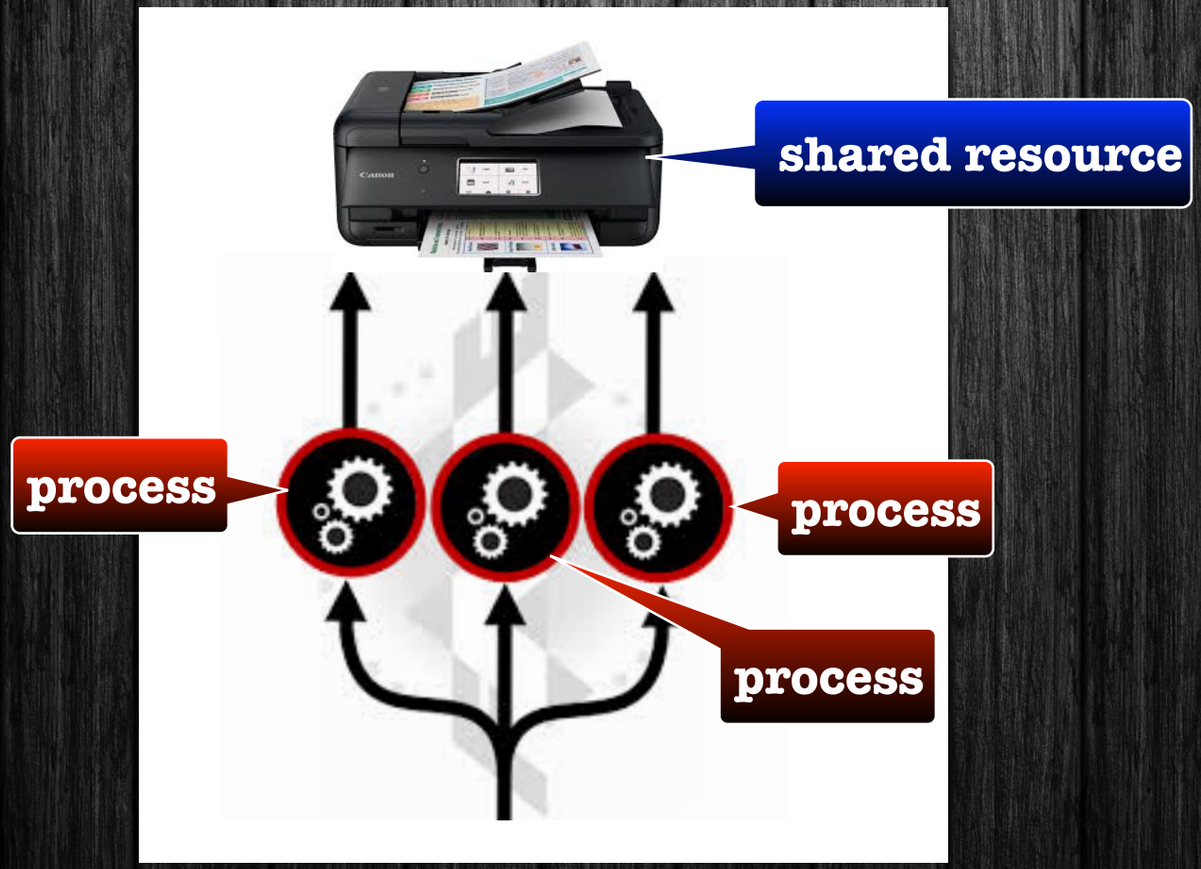
Liveness Properties

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

Examples

Starvation-Freedom

"a process which requests the printer will eventually get access"



Specification

- safety properties
- liveness properties

concurrent setting

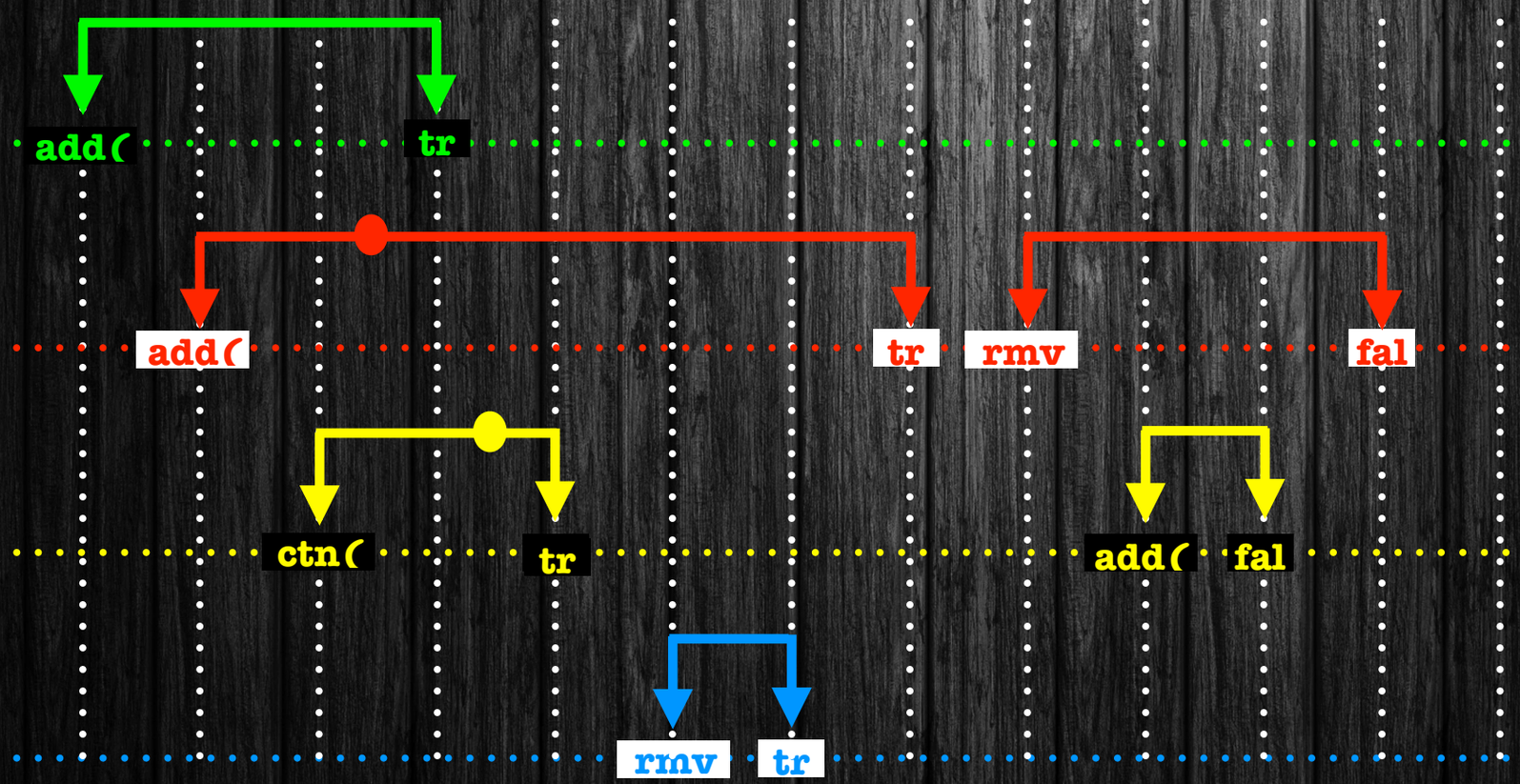
Liveness Properties

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

Examples

- non-blocking
- wait-freedom
- starvation-freedom

this course



Linearizability

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

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

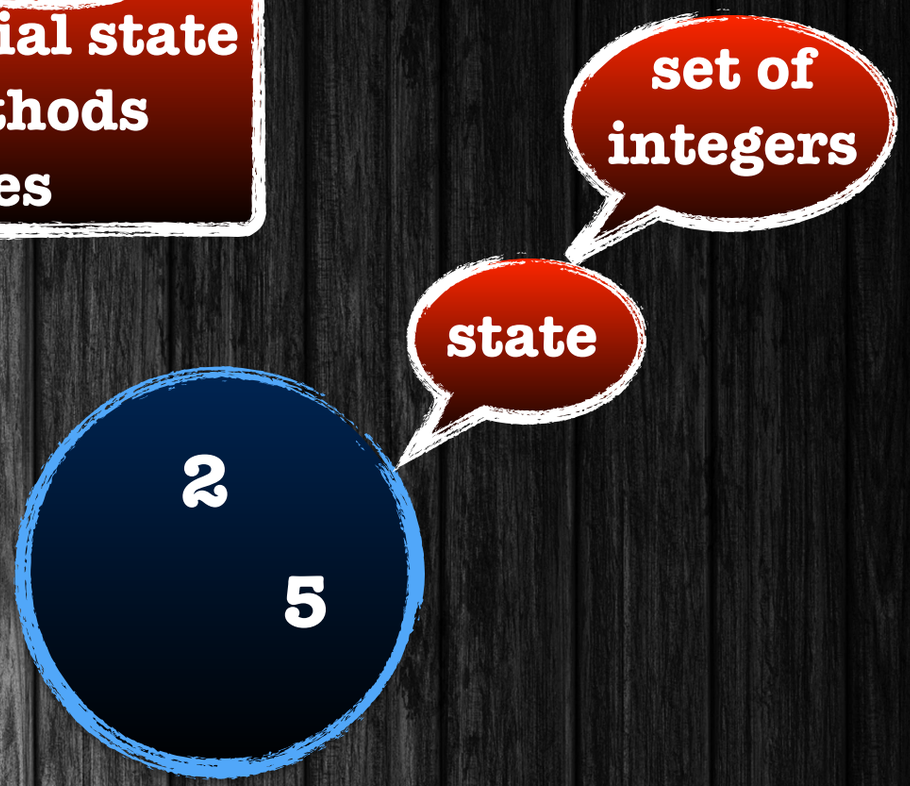
- Definition
- Sets
- Queues
- Stacks

Abstract Data Type

- Definition
- Sets
- Queues
- Stacks

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules



S : set of states

Abstract Data Type Set

- States
- initial state
- Methods
- Rules

initially empty



set of integers

state

S : set of states

$s_{init} \in S$: initial state

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

set of integers

state



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

methods

S : set of states

$s_{init} \in S$: initial state

M : set of method names

Abstract Data Type Set

- States
- initial state
- Methods
- Rules

set of integers

state



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

methods

S : set of states

$s_{init} \in S$: initial state

M : set of method names

R : set of rules

"atomic execution of a method"

operation

command

guarded command

rule

source state

target state

guard

$$s \xrightarrow{op} s' \text{ if } grd$$

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

set of integers

state

2

5

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

methods

command

guard

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

rules

guarded commands

S : set of states

$s_{init} \in S$: initial state

M : set of method names

R : set of rules

operation

"atomic execution of a method"

command

guarded command

source state

target state

guard

rule

$s \xrightarrow{op} s'$ if grd

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

command

guard

$add(x, true)$

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

source state

name

input

output

target state

operation

"atomic execution of a method"

rules

guarded commands

set of integers

state

2

5

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

methods

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

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

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

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

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

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

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

set of integers

state

2

5

rules

guarded commands

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

methods

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

set of integers

state

2

5

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

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

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

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

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

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

rules

guarded commands

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

methods

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

"infinite set of instantiations"

$$\{2,5\} \xrightarrow{add(4,true)} \{2,5,4\}$$

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

Abstract Data Type

Set

- States
- initial state
- Methods
- Rules

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

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

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

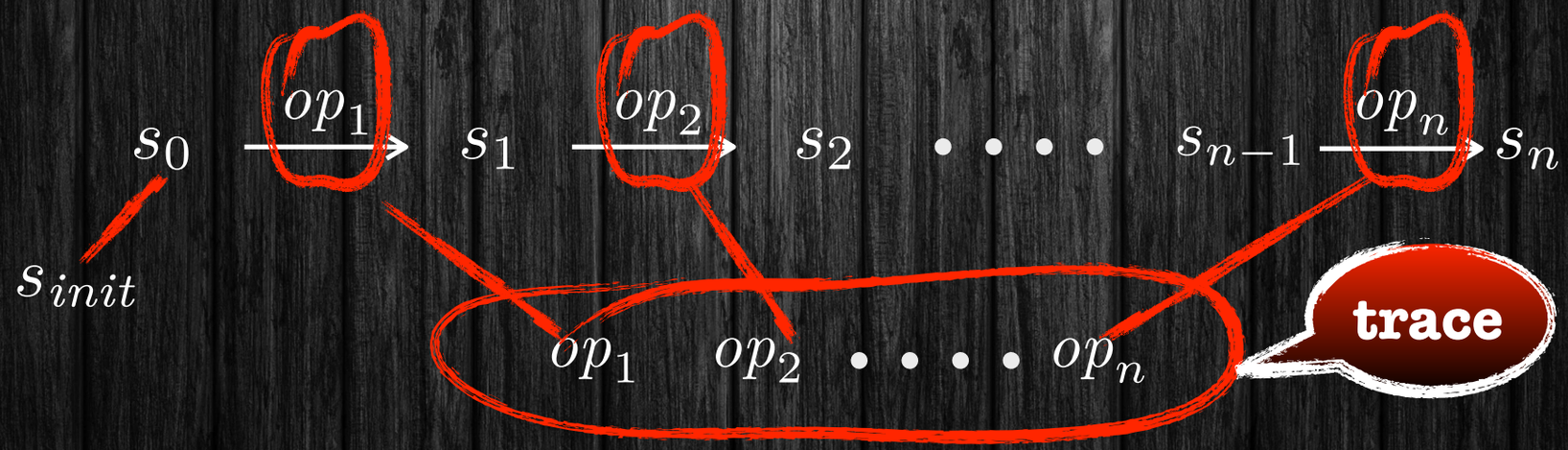
$$S \xrightarrow{rmv(x,false)} S \quad \text{if } x \notin S$$

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

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

Traces

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



Abstract Data Type Set

- States
- initial state
- Methods
- Rules

initially empty

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

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

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

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

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

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

set of integers

state

rules

guarded commands

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

methods

Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

set of integers

state

2

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

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

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

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

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

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

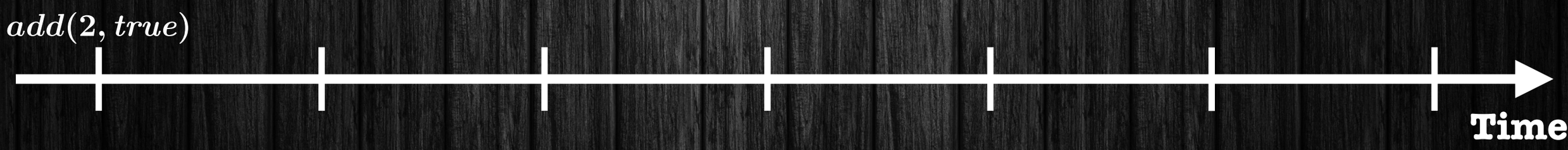
rules

guarded commands

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

methods

$add(2, true)$? ✔



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

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

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

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

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

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

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

set of integers

state

2

5

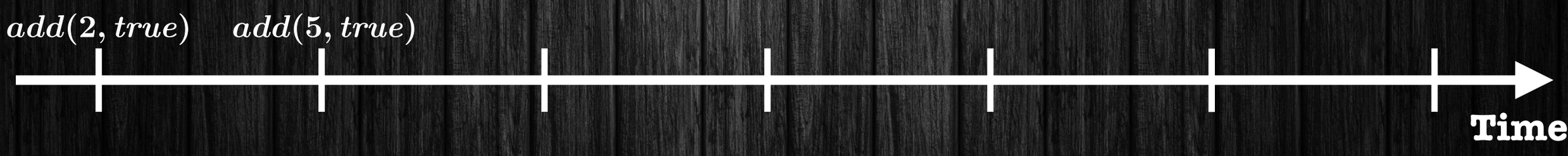
rules

guarded commands

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

methods

$add(5, true)$? ✔



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

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

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

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

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

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

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

set of integers

state

2

5

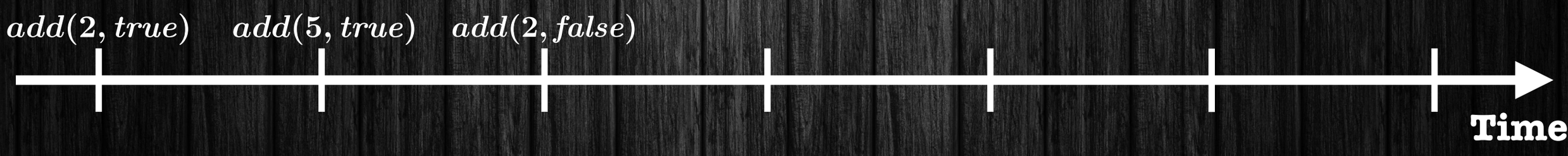
rules

guarded commands

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

methods

$add(2, false)$? ✔



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

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

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

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

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

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

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

set of integers

state

2

5

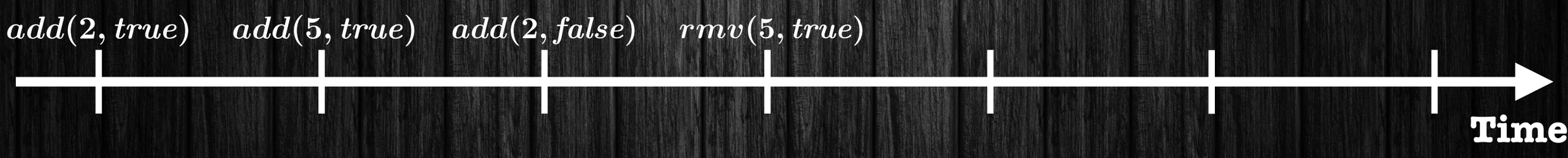
rules

guarded commands

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

methods

$rmv(5, true)$?



Abstract Data Type | Set

- States
- initial state
- Methods
- Rules

initially empty

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

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

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

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

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

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

set of integers

state

2

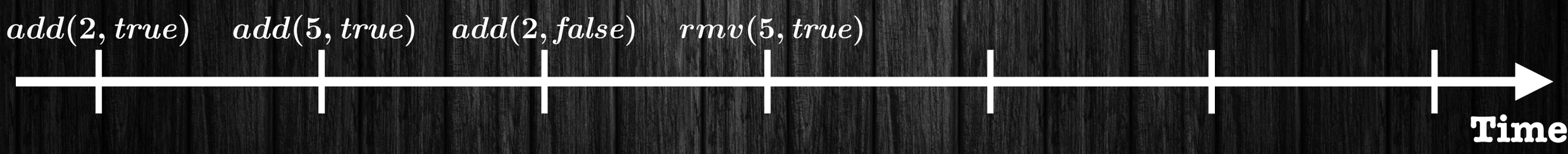
rules

guarded commands

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

methods

$rmv(3, true)$?



Abstract Data Type Set

- States
- initial state
- Methods
- Rules

initially empty

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

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

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

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

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

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

set of integers

state

2

9

rules

guarded commands

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

methods

$add(9, true)$?

trace

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

Time

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

Abstract Data Type — Set

- States
- initial state
- Methods
- Rules

initially empty

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

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

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

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

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

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

set of integers

state



rules

guarded commands

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

methods

Traces (Set)

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

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

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

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

atomic operations

Abstract Data Type

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

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

Abstract Data Type Queue

- States
- Initial state
- Methods
- Rules

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

word over integers

state

2 5 1 2

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

initially ϵ

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

initially ϵ

Abstract Data Type Queue

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

Abstract Data Type Queue

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

- *enq*
- *deq*

methods

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

- *enq*
- *deq*

methods

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

rules

guarded commands

- *enq*
- *deq*

methods

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

Traces

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

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

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

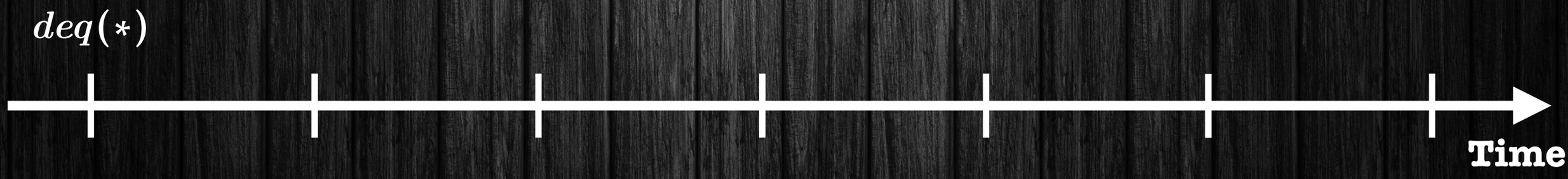
$$a \bullet w \xrightarrow{deq(a)} w$$

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

initially ϵ

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

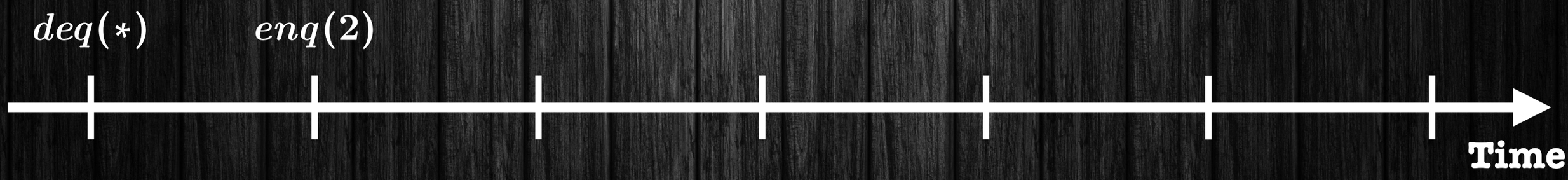
$$a \bullet w \xrightarrow{deq(a)} w$$

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

initially ϵ

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

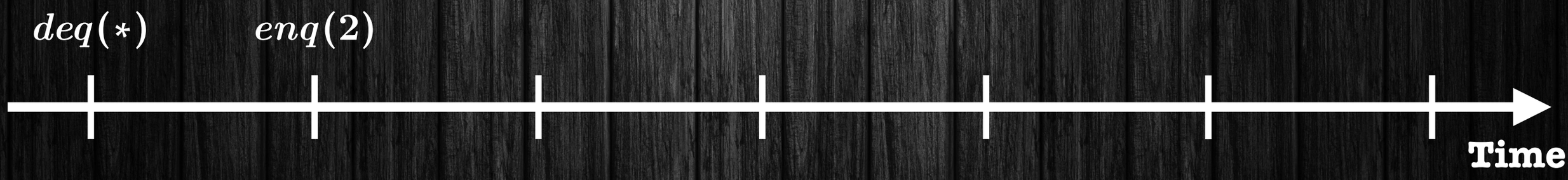
$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

2

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

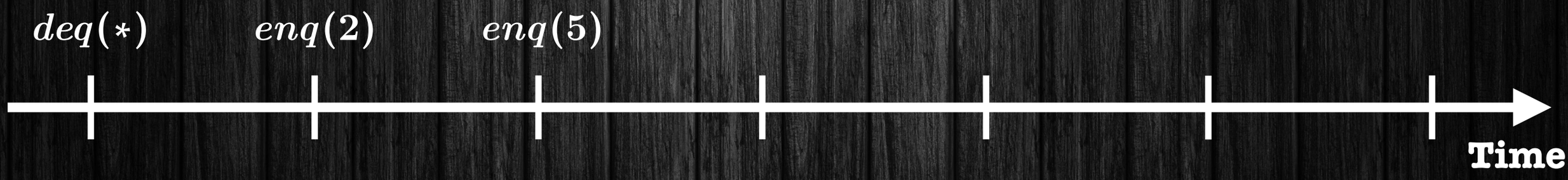
$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

2

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

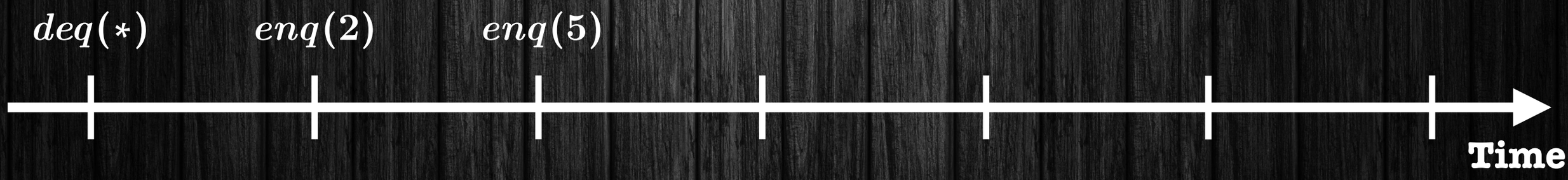
$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

2 5

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

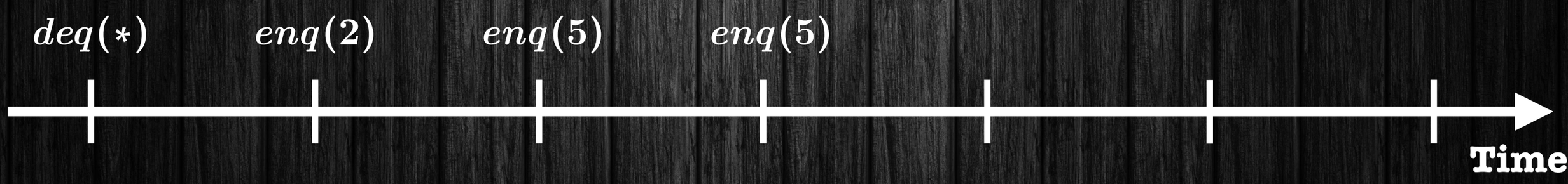
$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

2 5

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

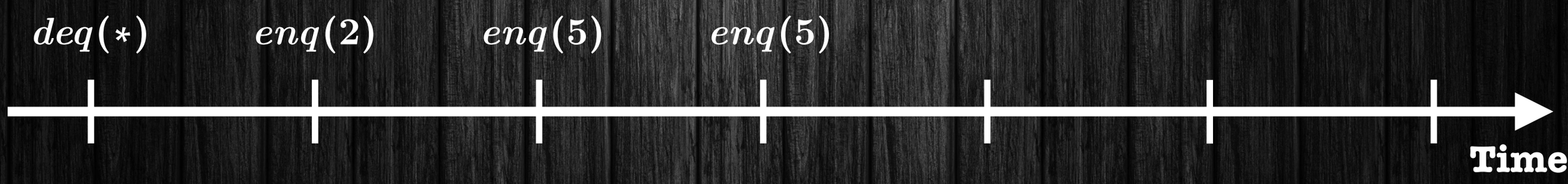
$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

2 5

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

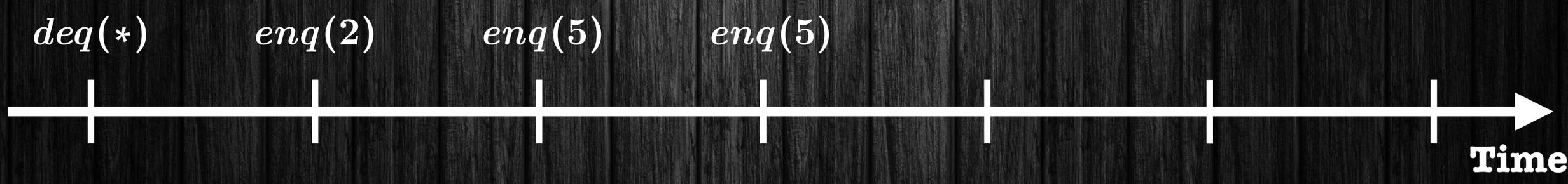
$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

2 5 5

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

$$enq(a):\ w \longrightarrow w \bullet a$$

$$deq(a):\ a \bullet w \longrightarrow w$$

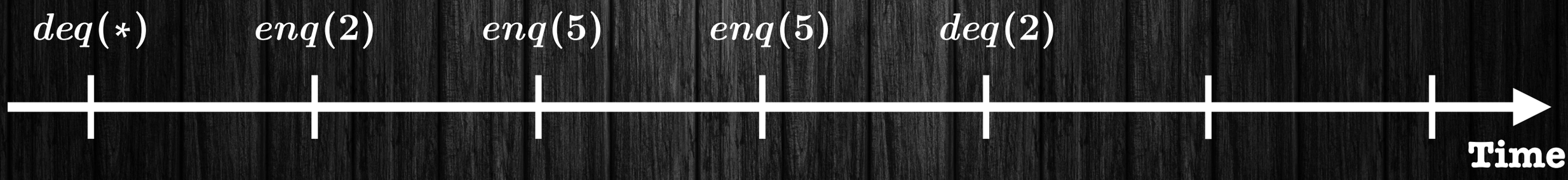
$$deq(*):\ \epsilon \longrightarrow \epsilon$$

initially ϵ

2 5 5

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

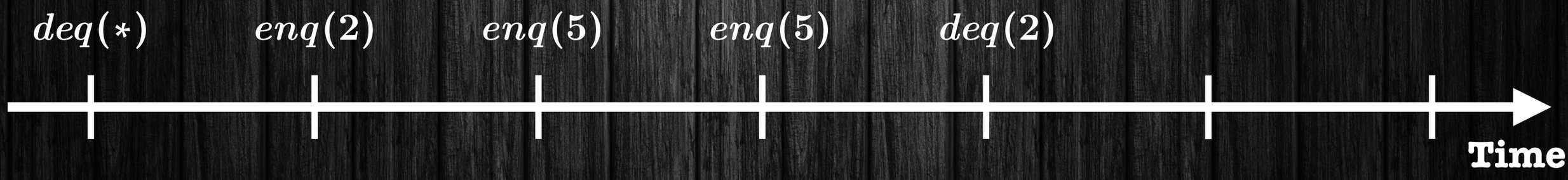
$$\epsilon \xrightarrow{deq(*)} \epsilon$$

initially ϵ

5 5

Traces

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



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

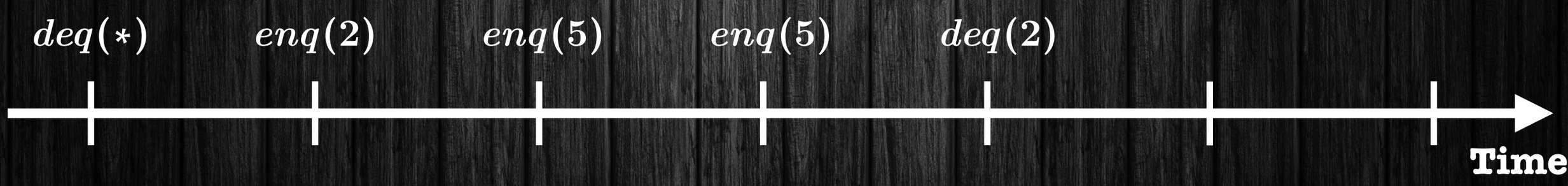
initially ϵ

5 5

Traces

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

Traces (Queue)



Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

Traces(Queue)

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)

$enq(3)$

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

3

Traces(Queue)

$enq(3)$

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

3

Traces(Queue)

$enq(3)$ $enq(5)$

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

3 5

Traces(Queue)

$enq(3)$ $enq(5)$

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

3 5

Traces(Queue)

$enq(3)$ $enq(5)$ $enq(6)$

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

3 5 6

Traces(Queue)

$enq(3)$ $enq(5)$ $enq(6)$

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

3 5 6

Traces(Queue)

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

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

5 6

Traces(Queue)

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

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

5 6

Traces(Queue)

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

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

6

Traces(Queue)

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

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

6

Traces(Queue)

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

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)

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

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)

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

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)

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

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)

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

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)

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

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

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)

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

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

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)

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

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

$enq(3)$

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

3

Traces(Queue)

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

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

$enq(3)$

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

3

Traces(Queue)

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

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

$enq(3)$ $enq(5)$

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

3 5

Traces(Queue)

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

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

$enq(3)$ $enq(5)$

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

3 5

Traces(Queue)

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

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

$enq(3)$ $enq(5)$ $deq(5)$

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

3 5

Traces(Queue)

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

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

$enq(3)$ $enq(5)$ $deq(5)$

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

3 5

Traces(Queue)

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

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

$enq(3)$ $enq(5)$ $deq(5)$ ✗

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)

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

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

$enq(3)$ $enq(5)$ $deq(5)$ ✗

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)

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

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

$enq(3)$ $enq(5)$ $deq(5)$ ✗

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

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)

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

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

$enq(3)$ $enq(5)$ $deq(5)$ ✗

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

Abstract Data Type Queue

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

Traces(Queue)

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

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

$enq(3)$ $enq(5)$ $deq(5)$ ✗

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

atomic operations

Abstract Data Type

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

Abstract Data Type **Stack**

- States
- initial state
- Methods
- Rules

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

word over integers

state



Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

rules

guarded commands

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

rules

guarded commands

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

2 5 1 2

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

- *push*
- *pop*

methods

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

- *push*
- *pop*

methods

push(3)

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

push(3)

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

push(3)

push(7)

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

push(3) push(7)

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

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

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

5 7 3

- *push*
- *pop*

methods

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

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

5 7 3

- *push*
- *pop*

methods

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

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

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

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

7 3

- *push*
- *pop*

methods

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

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

Traces(Stack)

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

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

3

- *push*
- *pop*

methods

Traces(Stack)

trace

τ

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

Abstract Data Type Stack

- States
- initial state
- Methods
- Rules

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

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

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

initially ϵ

rules

guarded commands

word over integers

state

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

Traces = sequences of atomic operations

ush

op

guarded commands

methods

Traces(Stack)

trace

τ

atomic operations

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

Outline

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

Outline

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

Sequential Programs

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

Sequential Programs

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

Sequential Programs

Program

Sequential Programs

Program

Abstract Data Types

atomic operations

Sequential Programs

Program

Abstract Data Types

atomic operations

Programs

non-atomic operations

Sequential Programs

Program

Abstract Data Types

atomic operations

Programs

non-atomic operations

operations take time

Sequential Programs

Program

Abstract Data Types

atomic operations

Programs

non-atomic operations

operations take time

operation = two events

Sequential Programs

Program

Abstract Data Types

atomic operations

Programs

non-atomic operations

operations take time

operation = two events

- invocation

Sequential Programs

Program

Abstract Data Types

atomic operations

Programs

non-atomic operations

operations take time

operation = two events

- invocation
- response

Sequential Programs

Program

operation =
two events

- invocation
- response

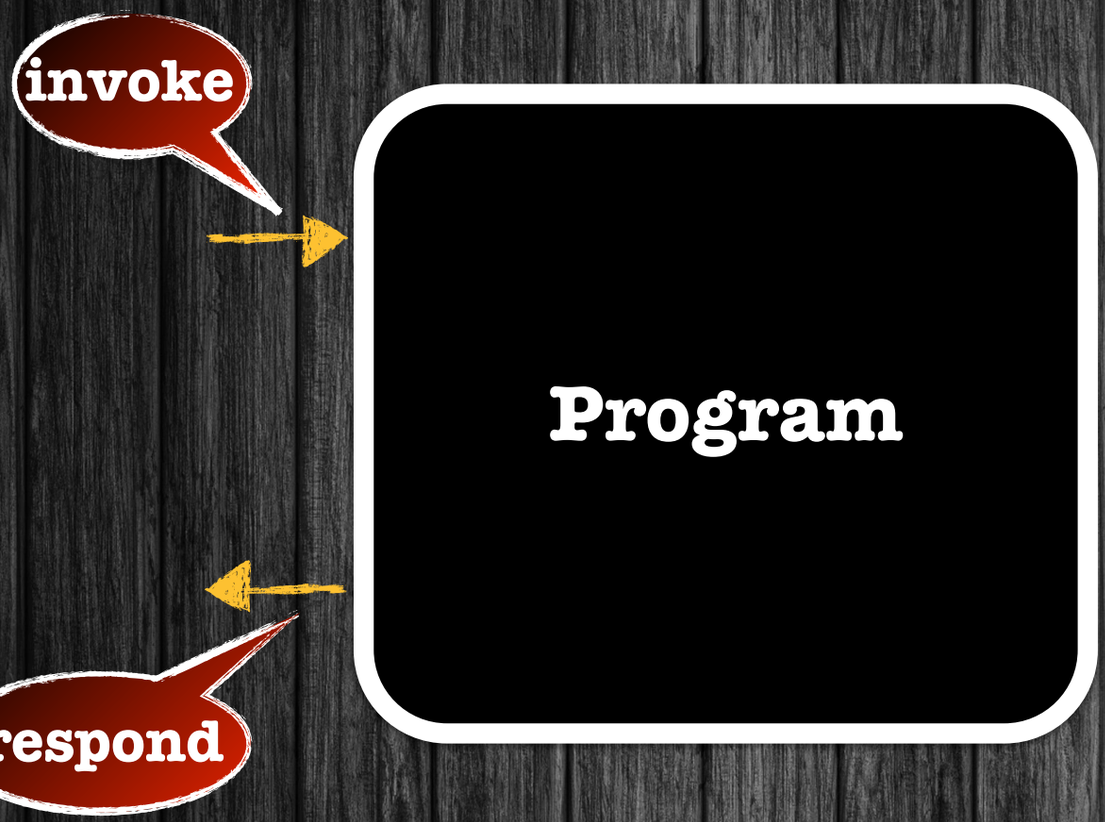
Sequential Programs



operation = two events

- invocation
- response

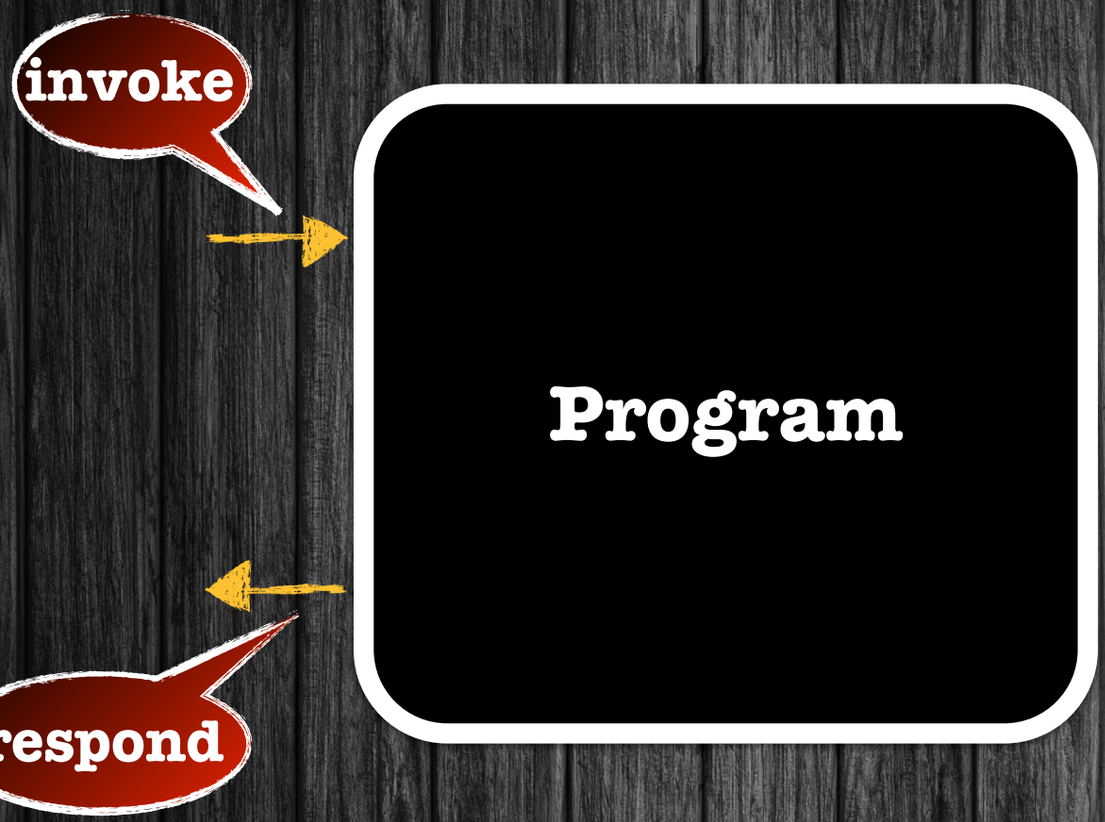
Sequential Programs



operation = two events

- invocation
- response

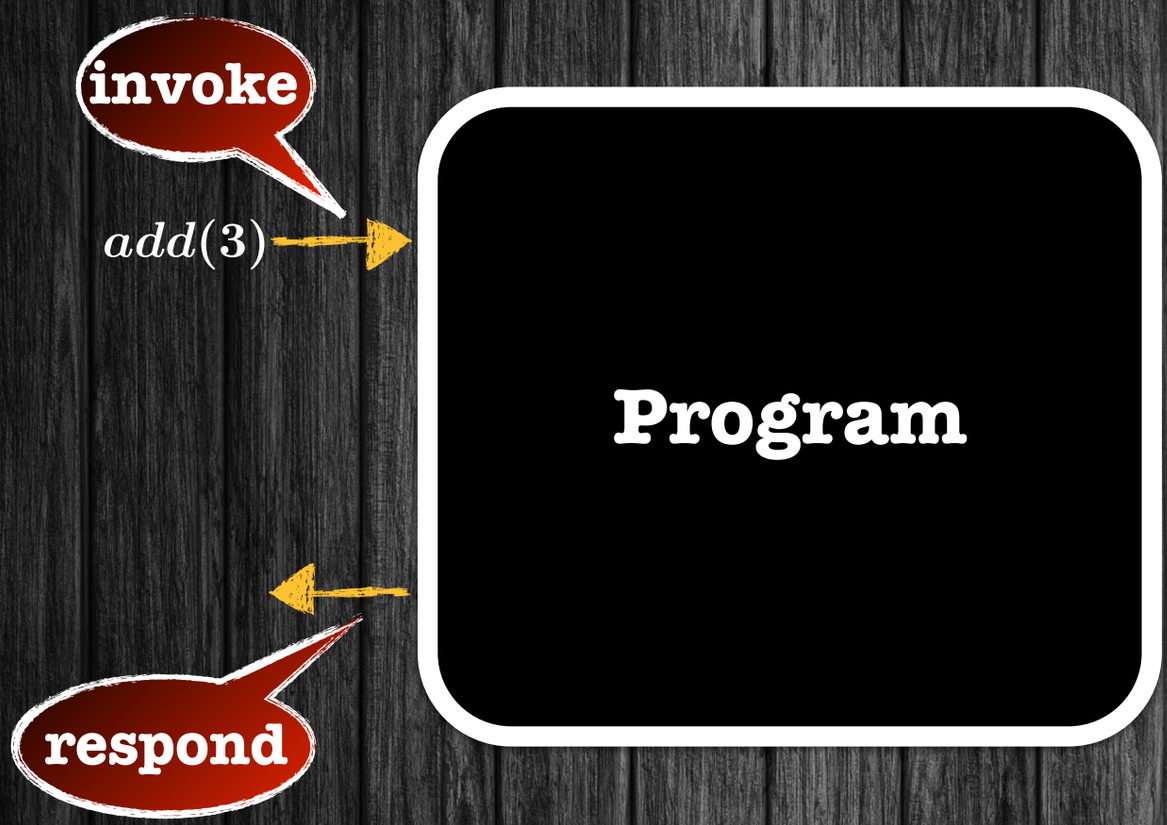
Sequential Programs | Sets



operation =
two events

- invocation
- response

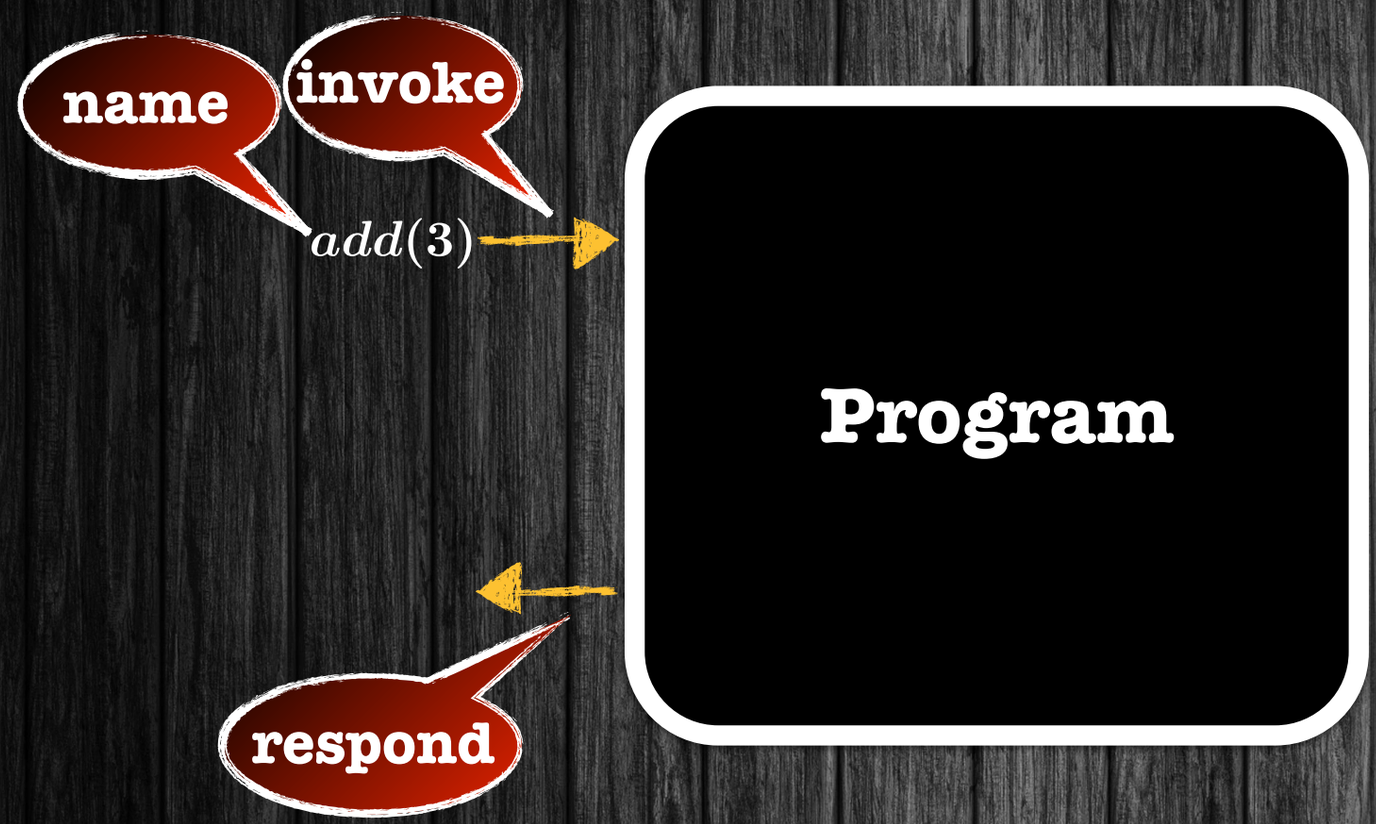
Sequential Programs | Sets



operation = two events

- invocation
- response

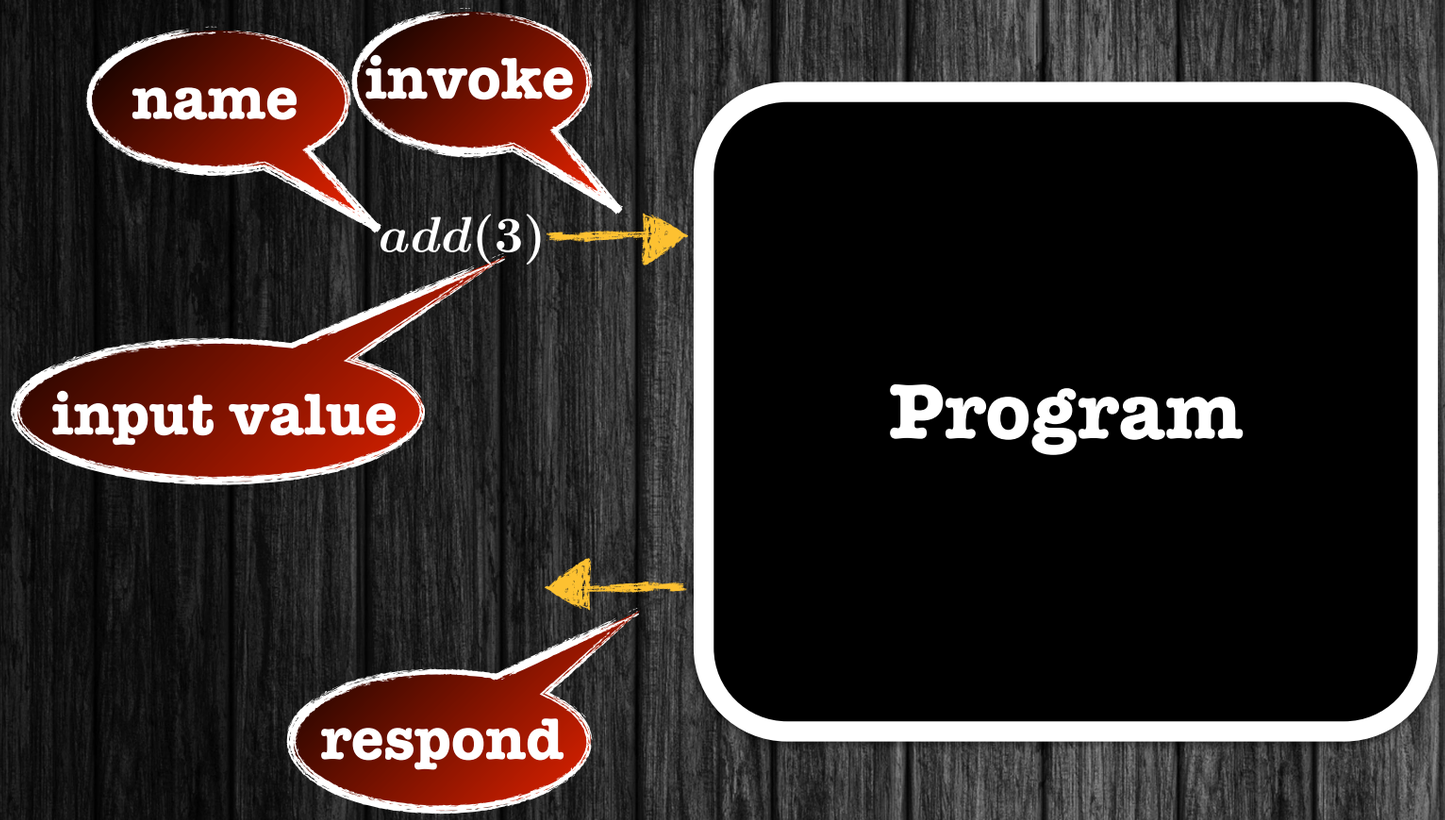
Sequential Programs | Sets



operation = two events

- invocation
- response

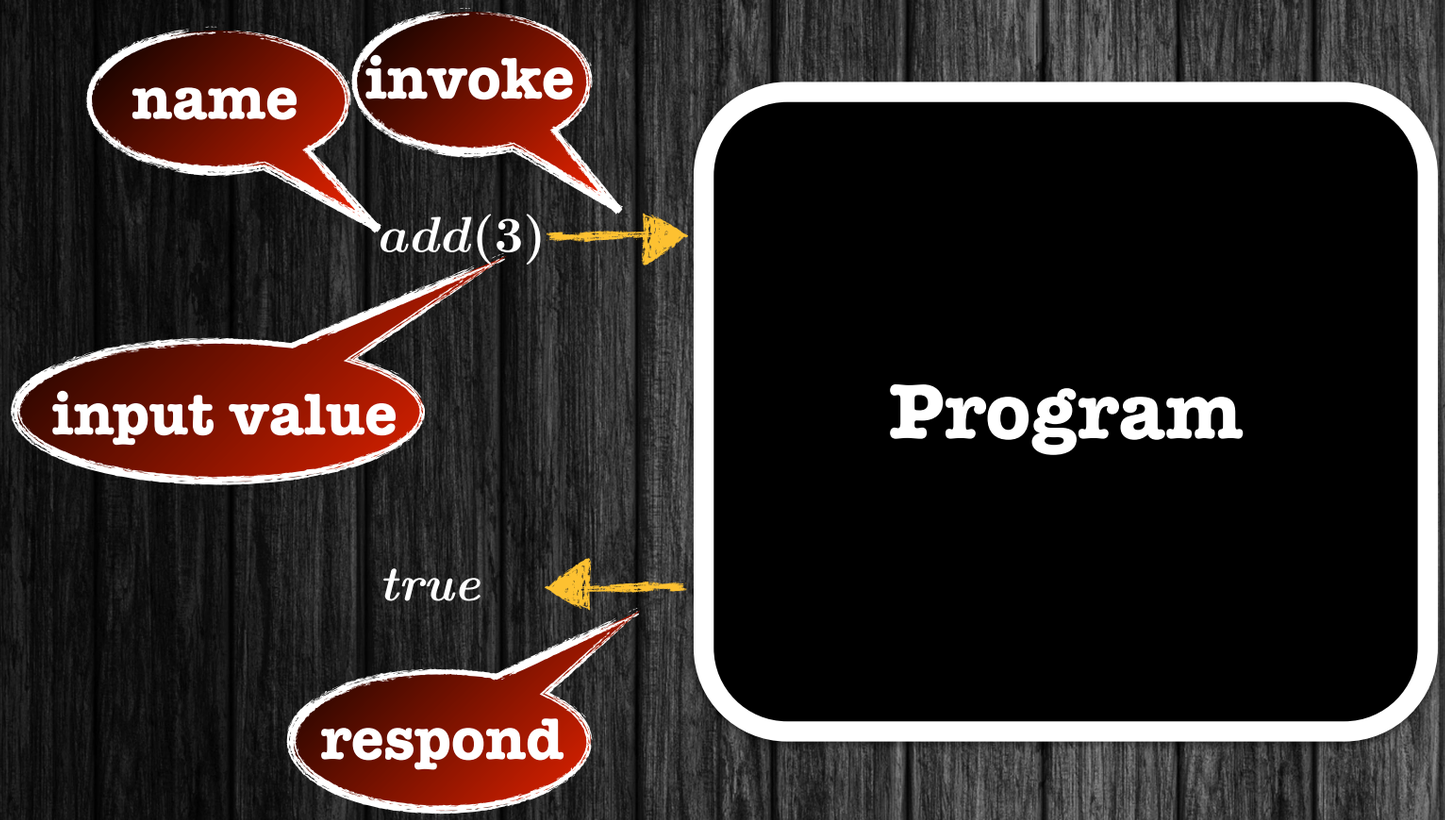
Sequential Programs | Sets



operation =
two events

- invocation
- response

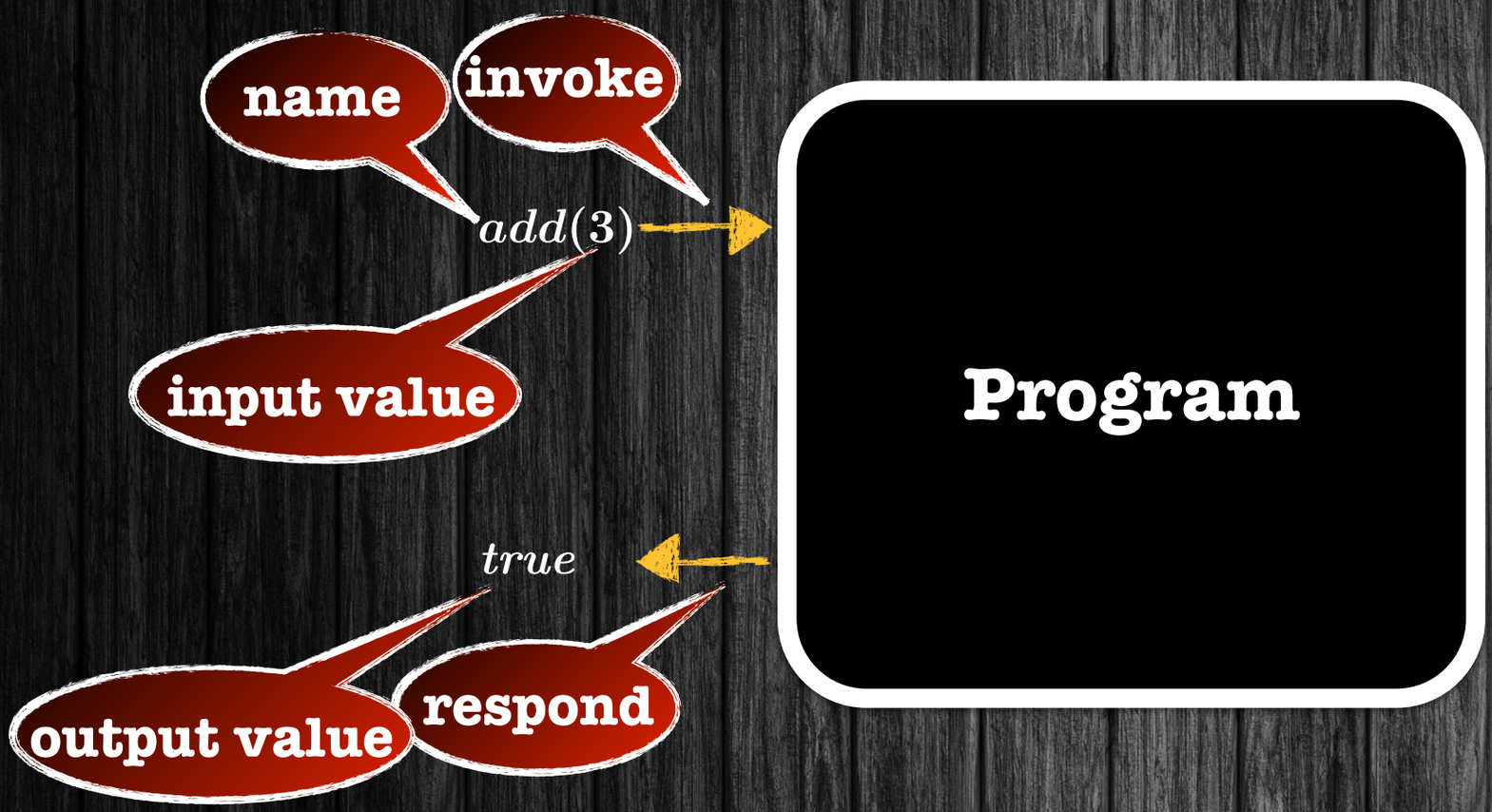
Sequential Programs | Sets



operation =
two events

- invocation
- response

Sequential Programs | Sets



operation = two events

- invocation
- response

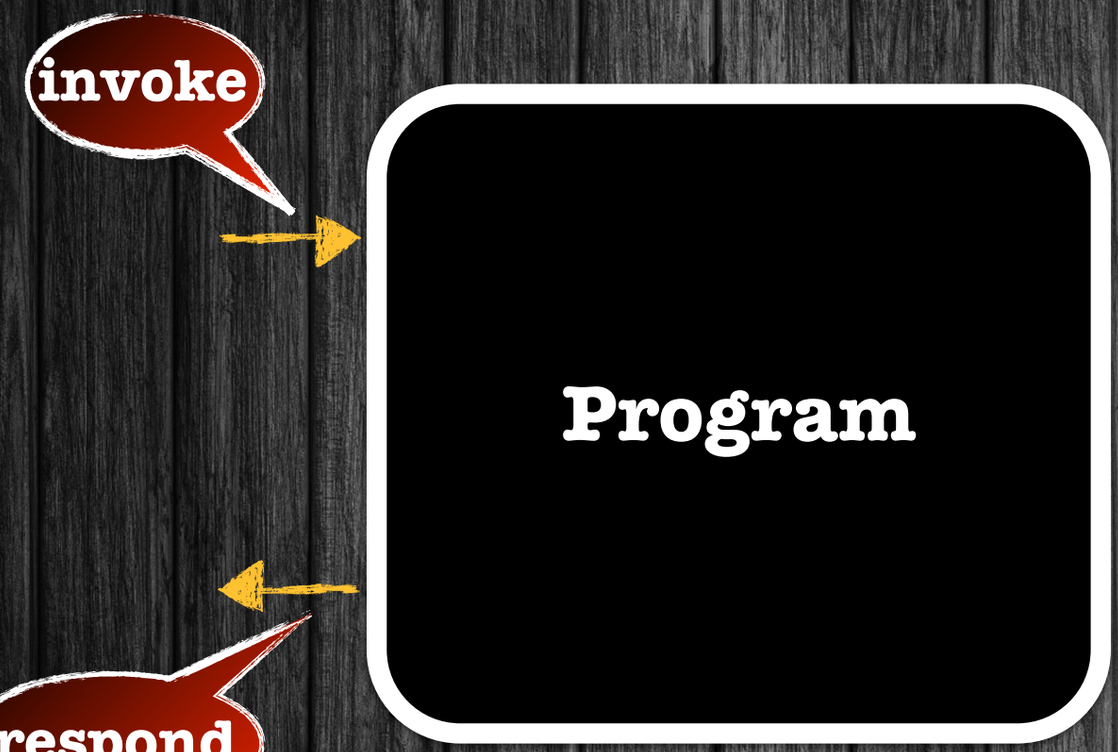
Sequential Programs

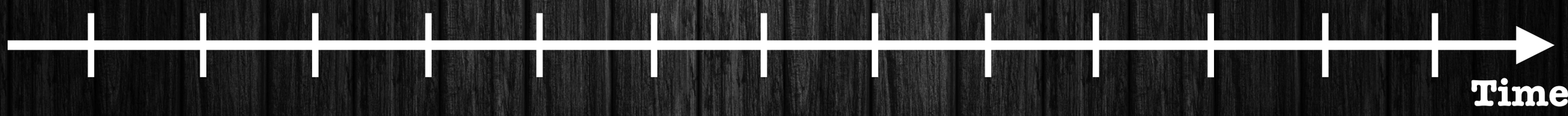
- Programs
- Sequential Histories
- Histories ⟶ Traces
- Example: implementing a set
- Program ⟶ Histories ⟶ Traces

Sequential Programs

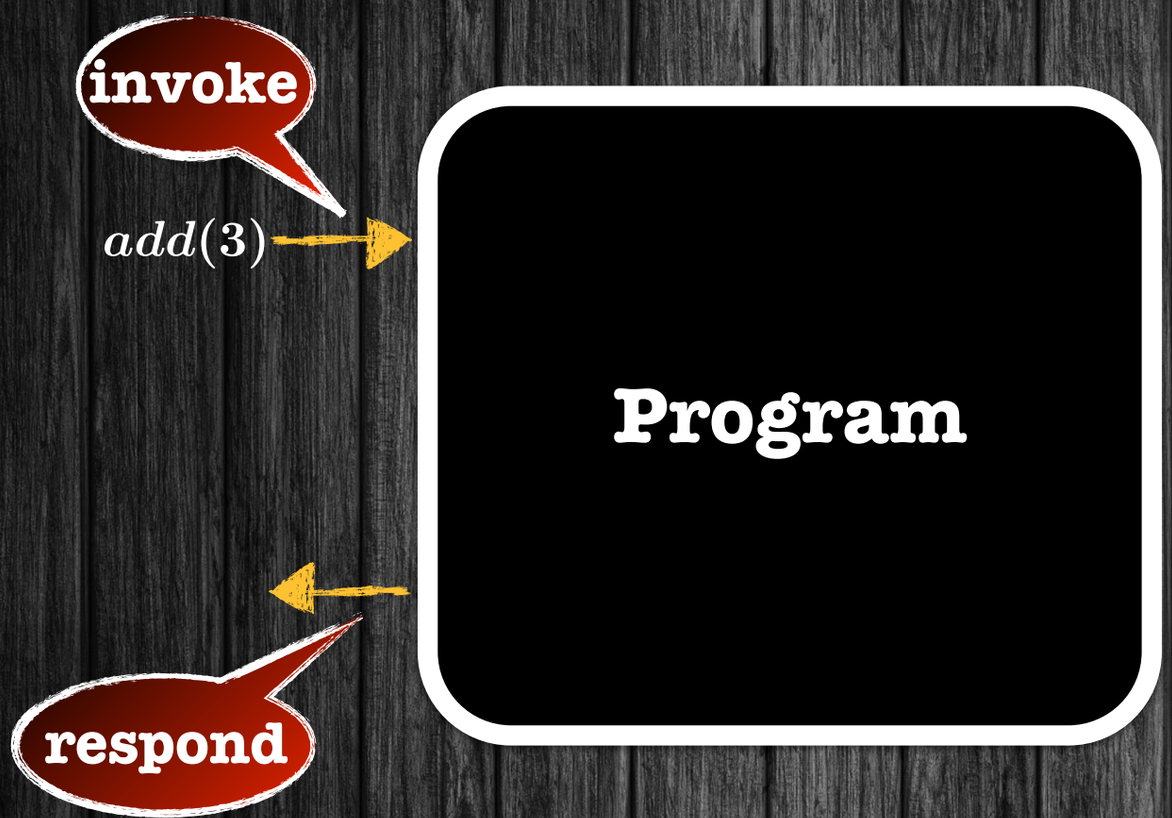
- Programs
- Sequential Histories
- Histories → Traces
- Example: implementing a set
- Program → Histories → Traces

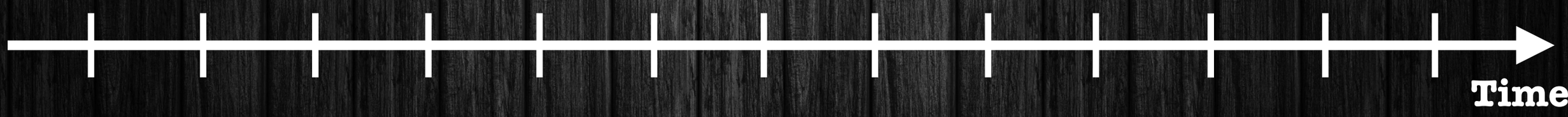
Sequential Histories



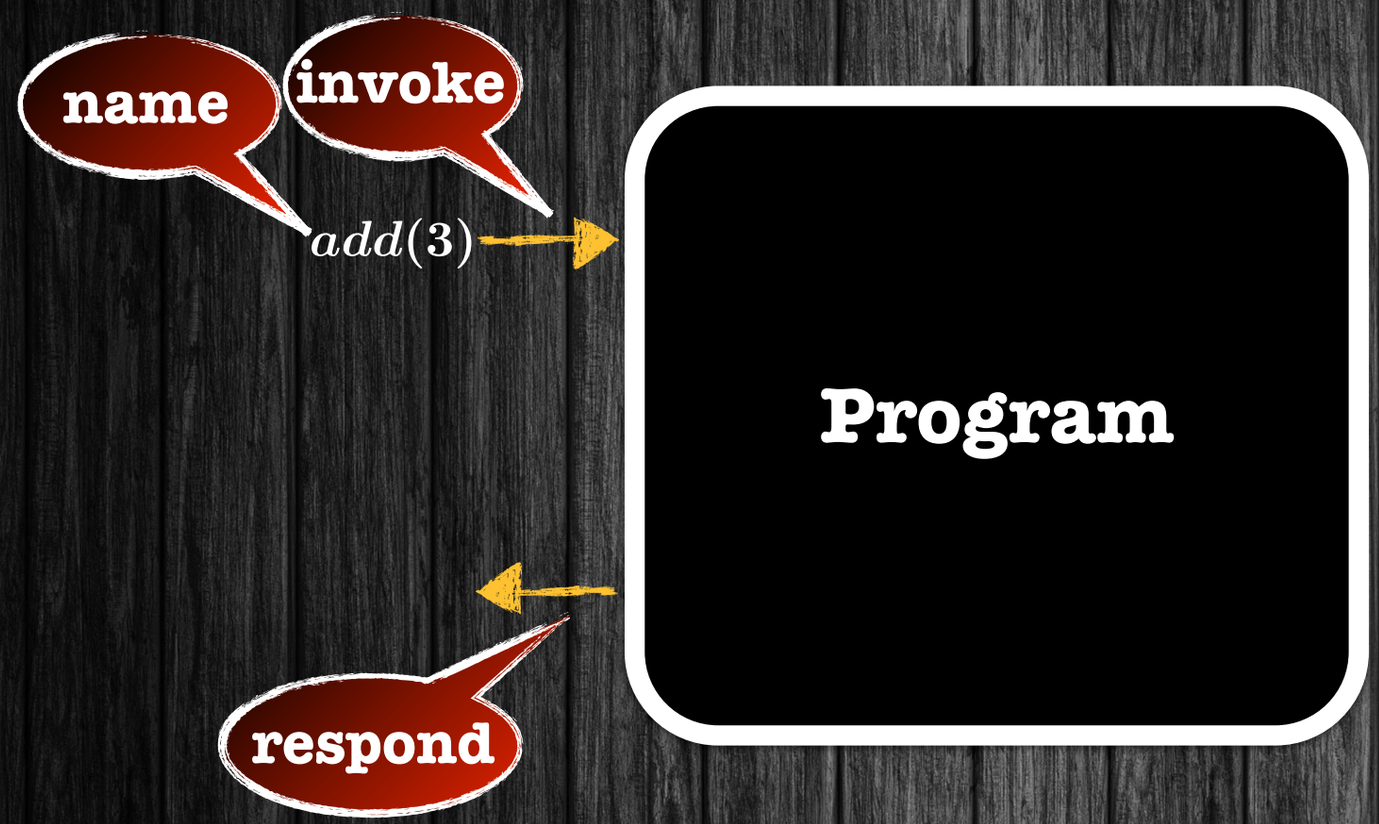


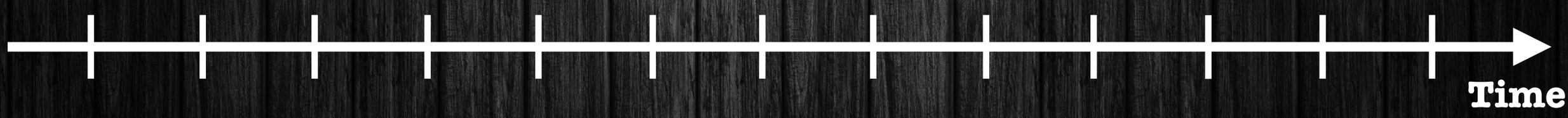
Sequential Histories



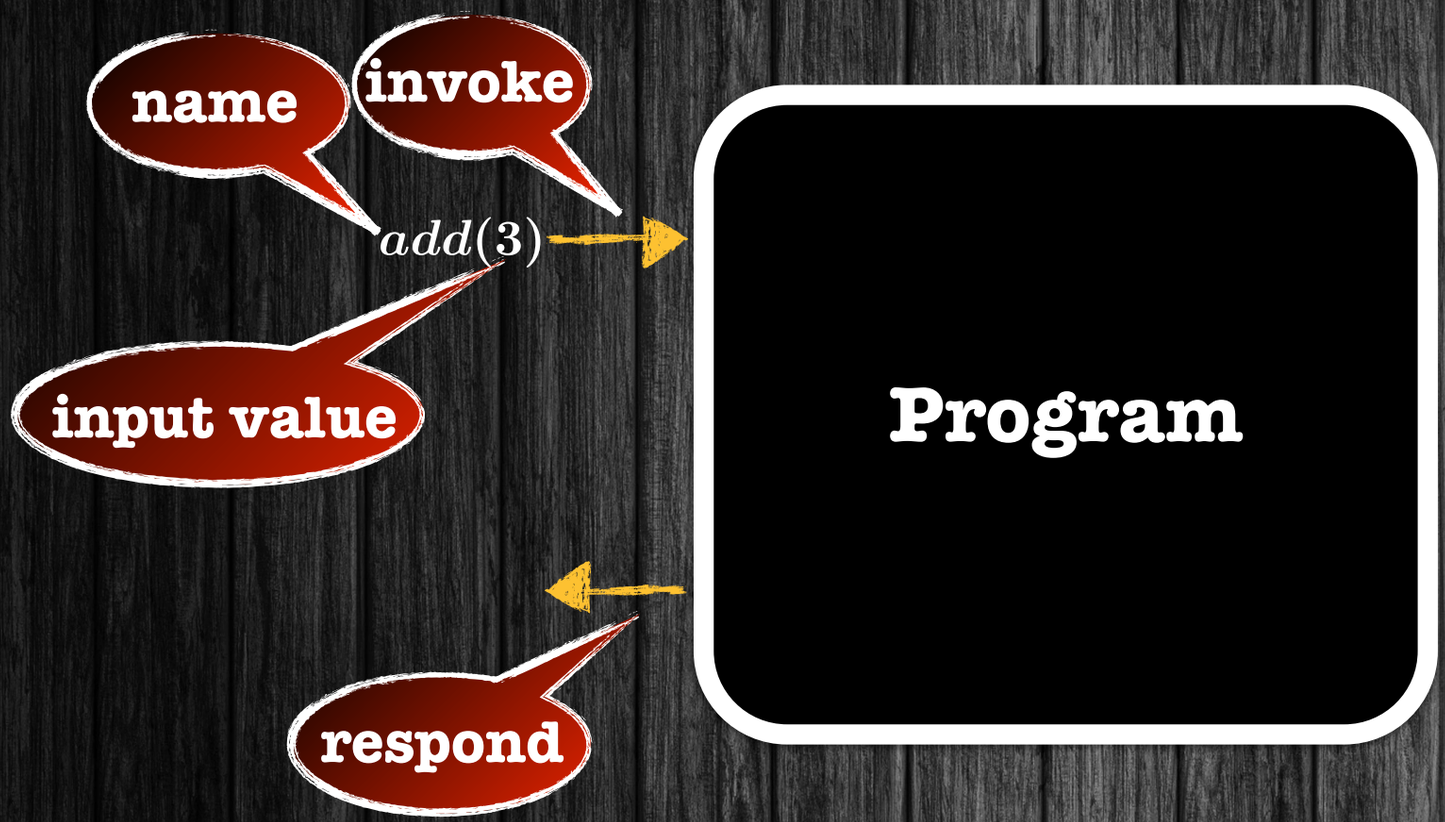


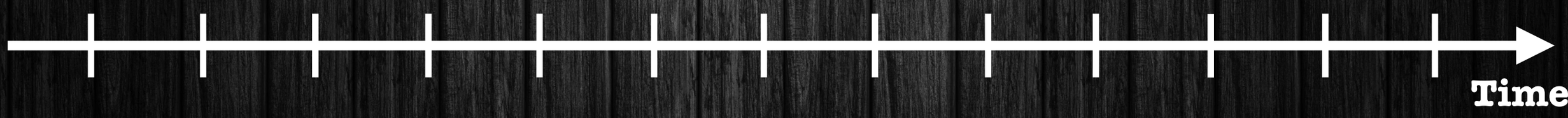
Sequential Histories



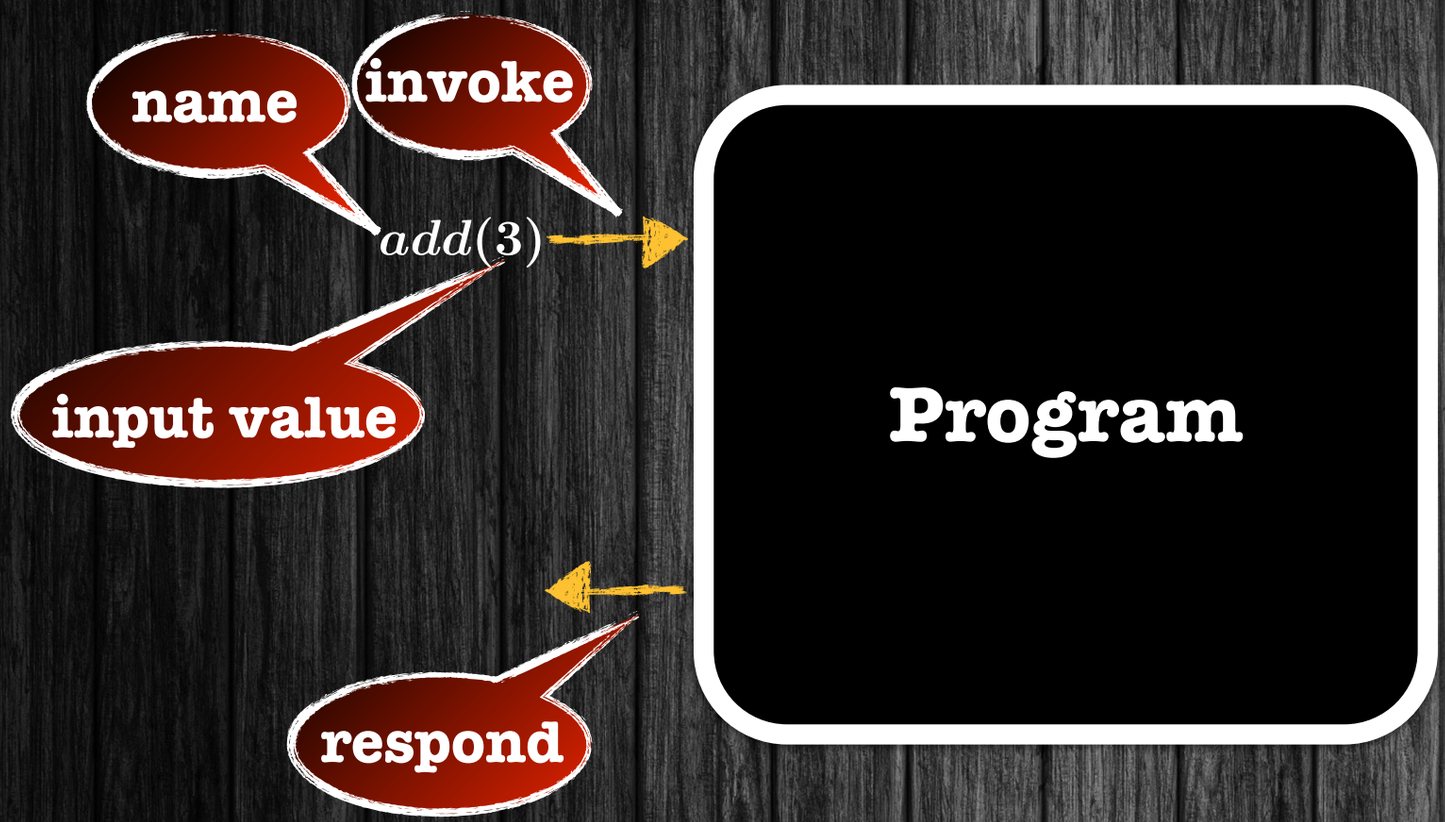


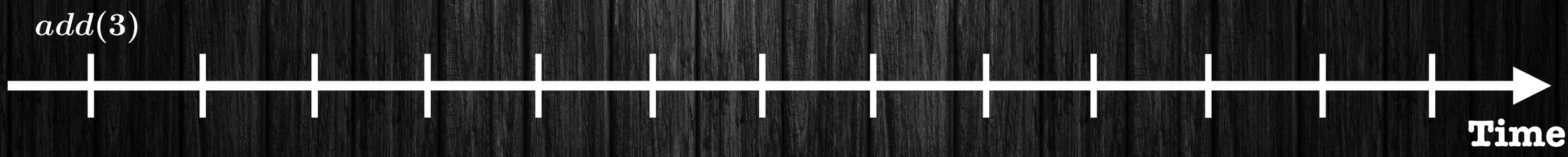
Sequential Histories



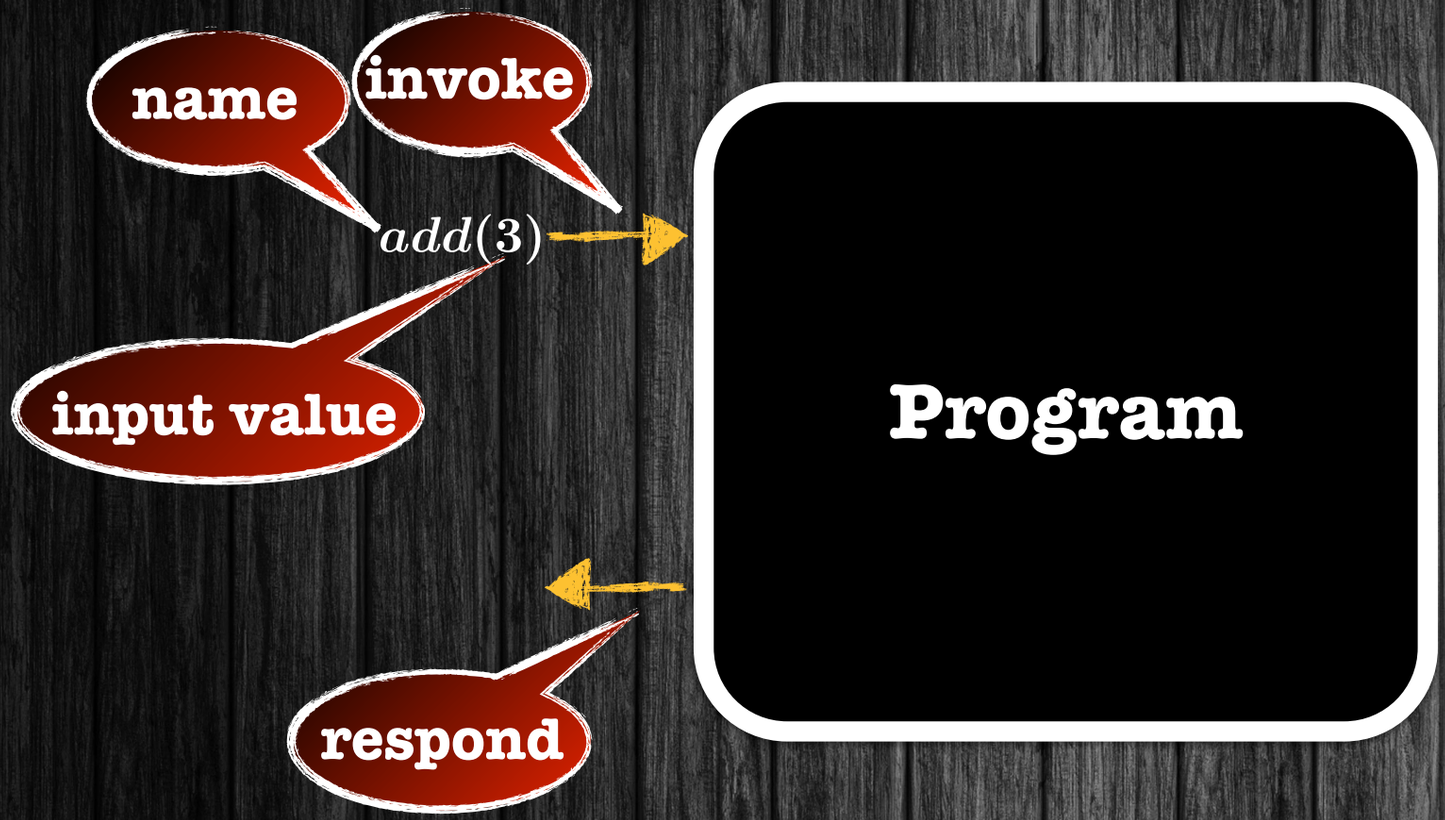


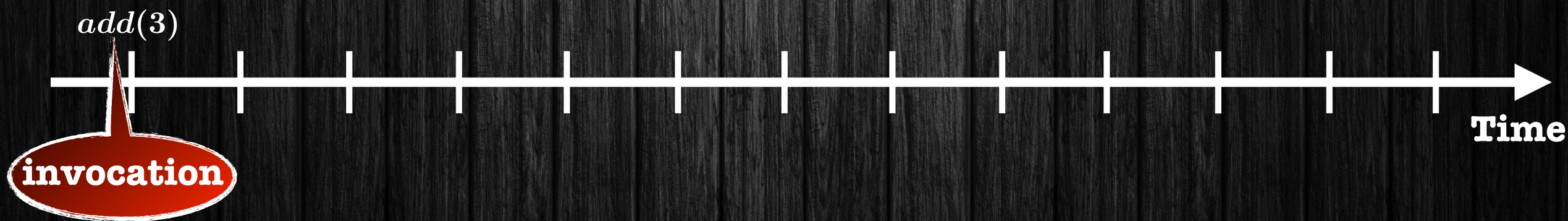
Sequential Histories



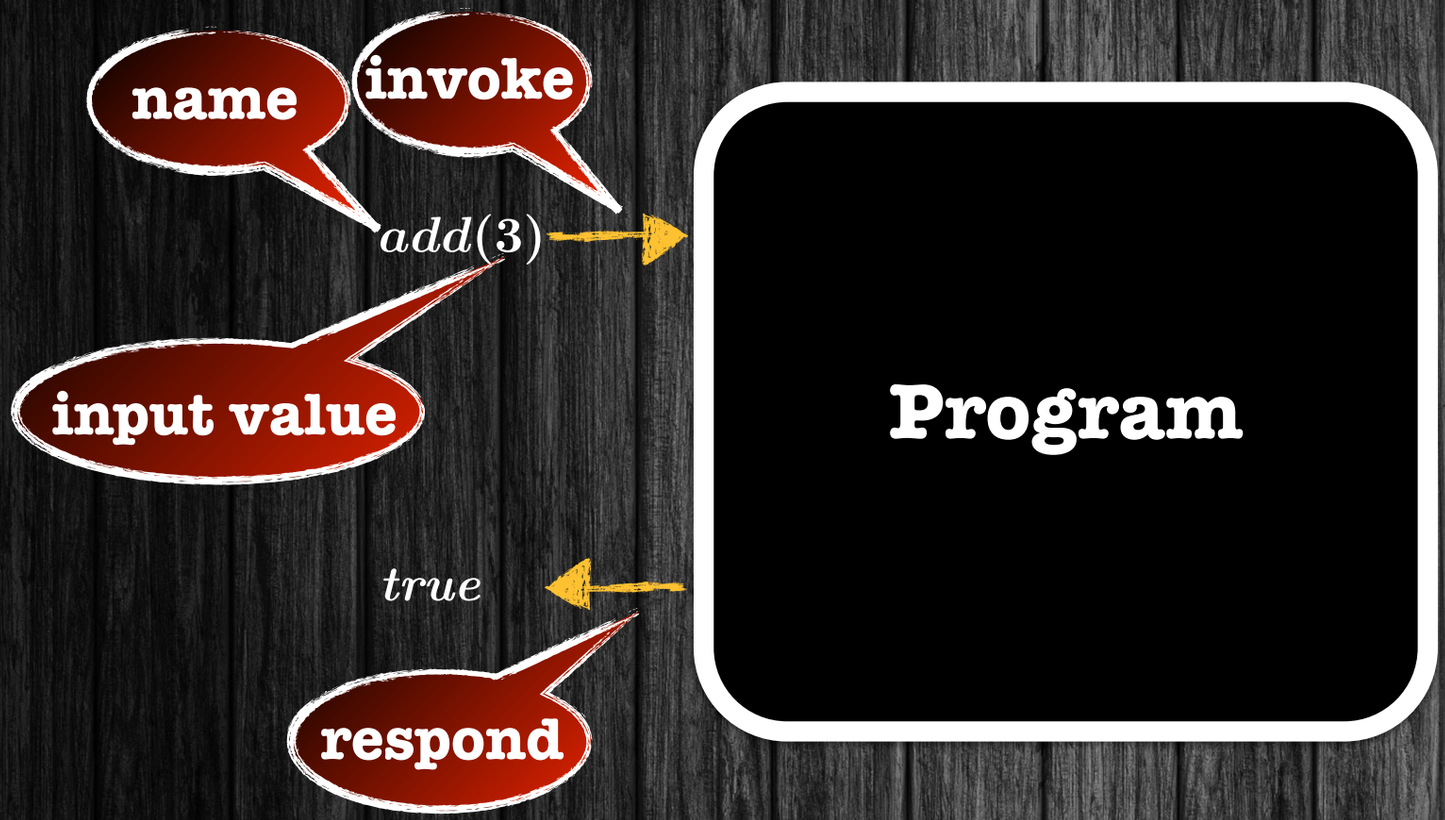


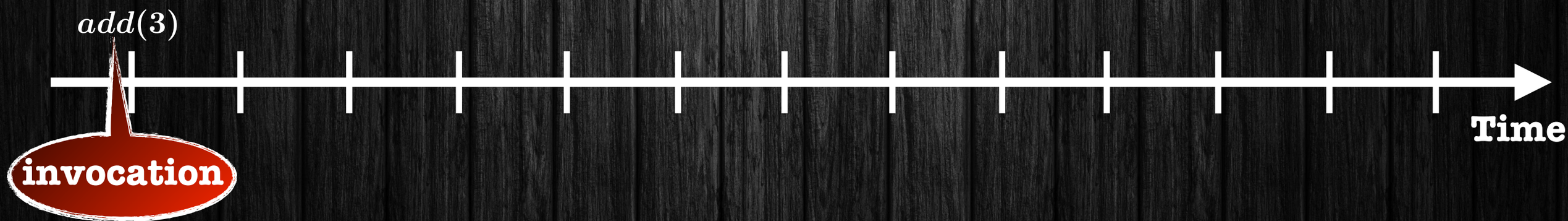
Sequential Histories



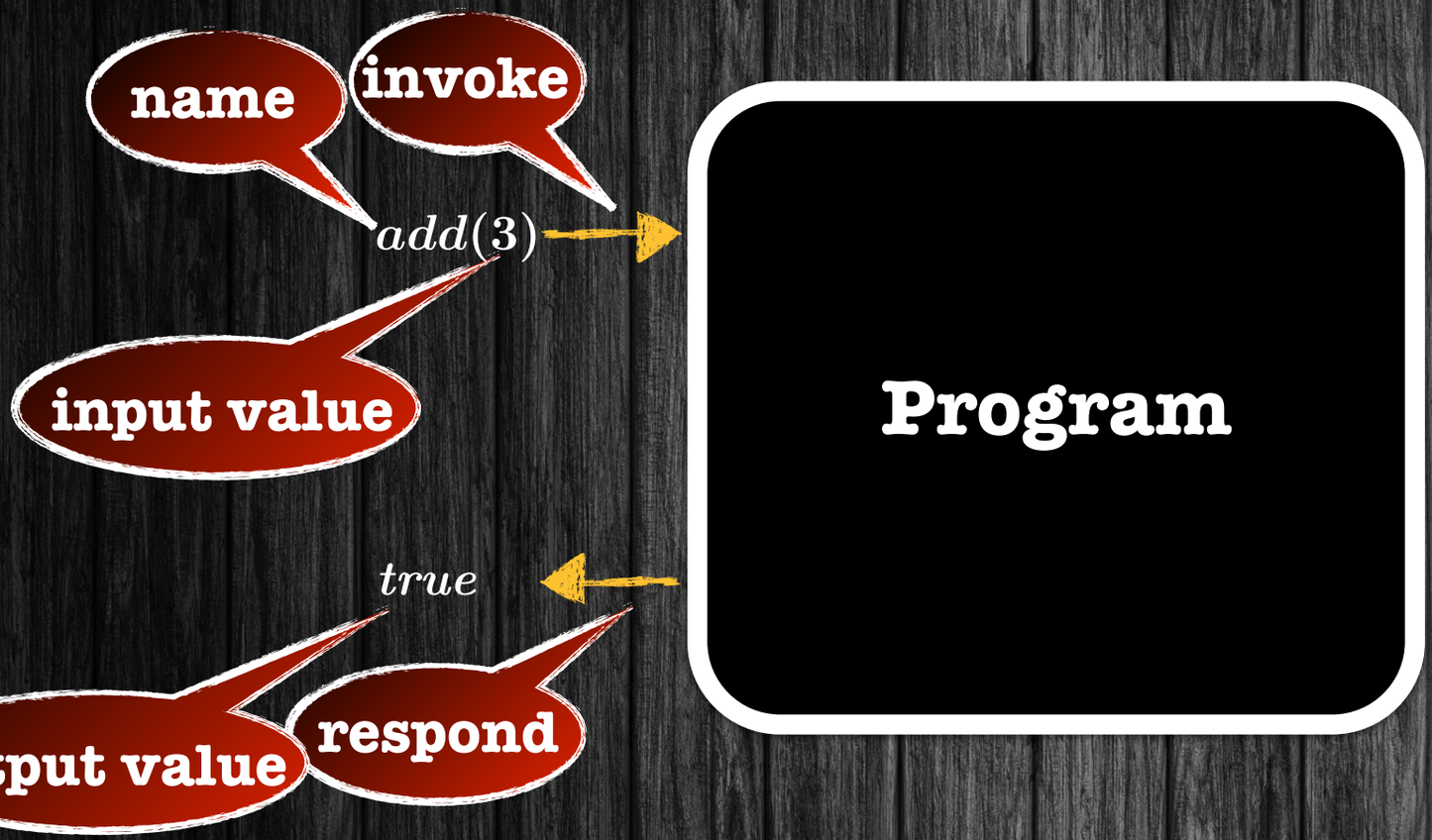


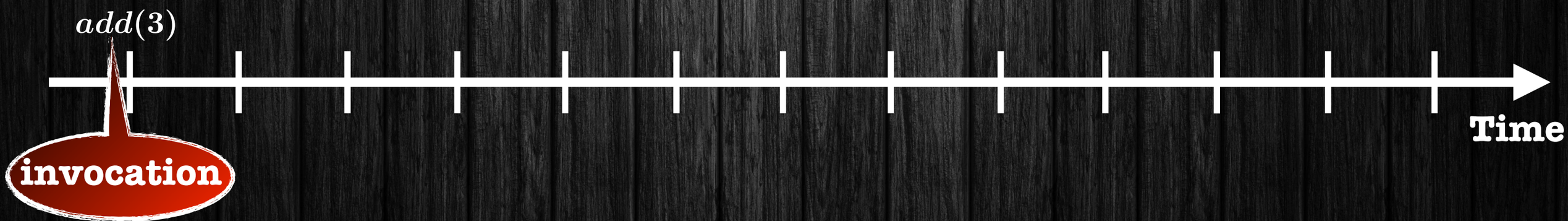
Sequential Histories



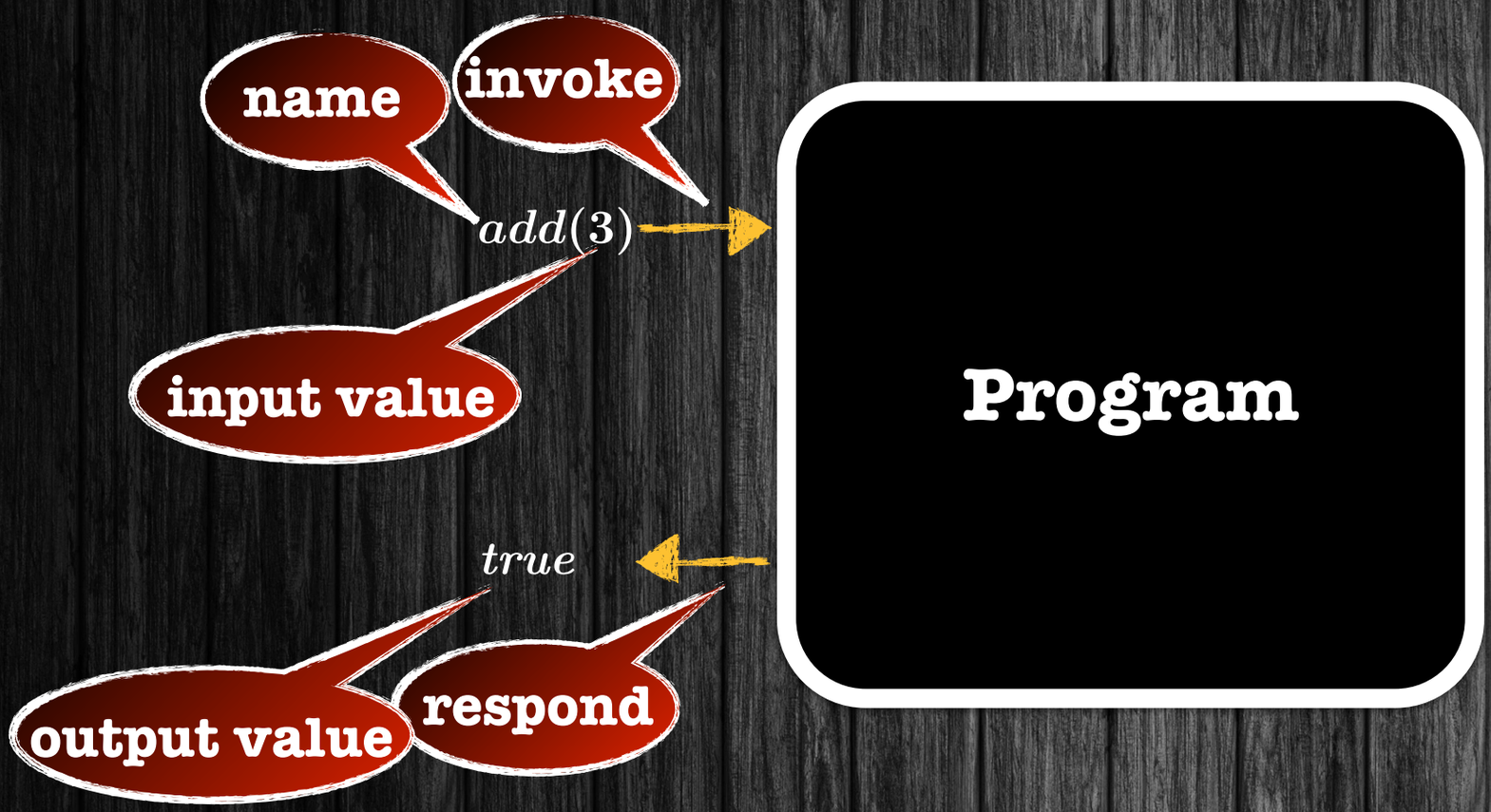


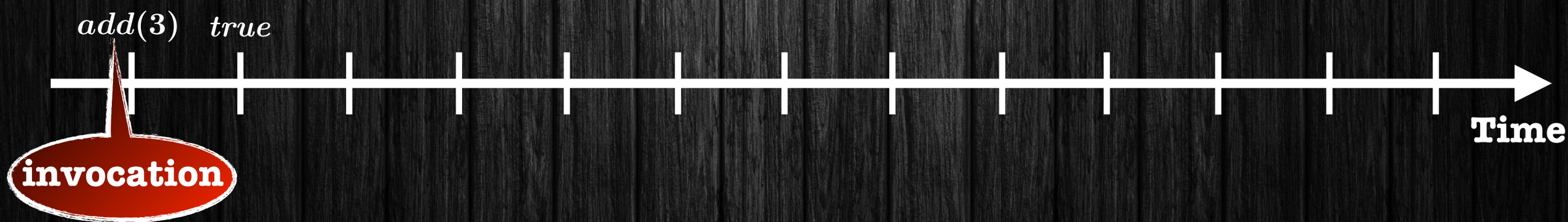
Sequential Histories



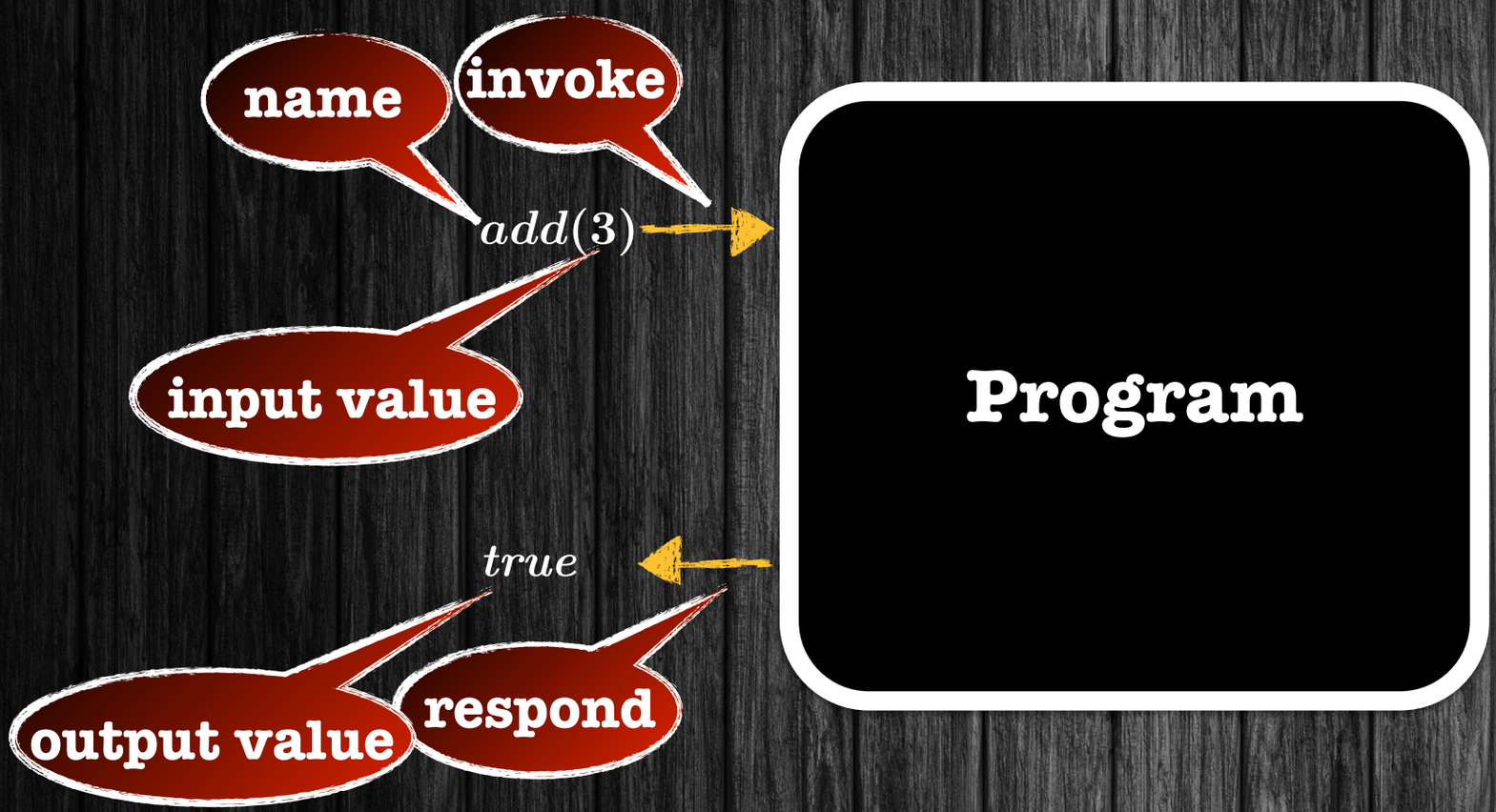


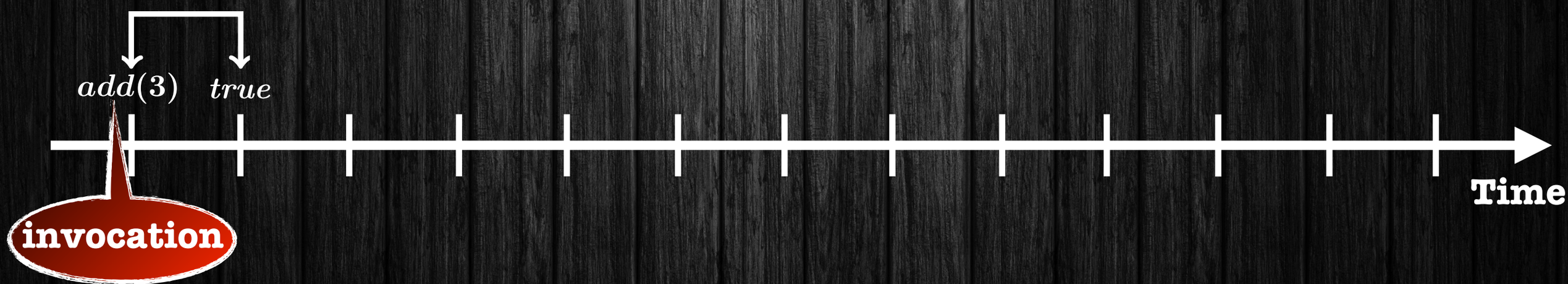
Sequential Histories



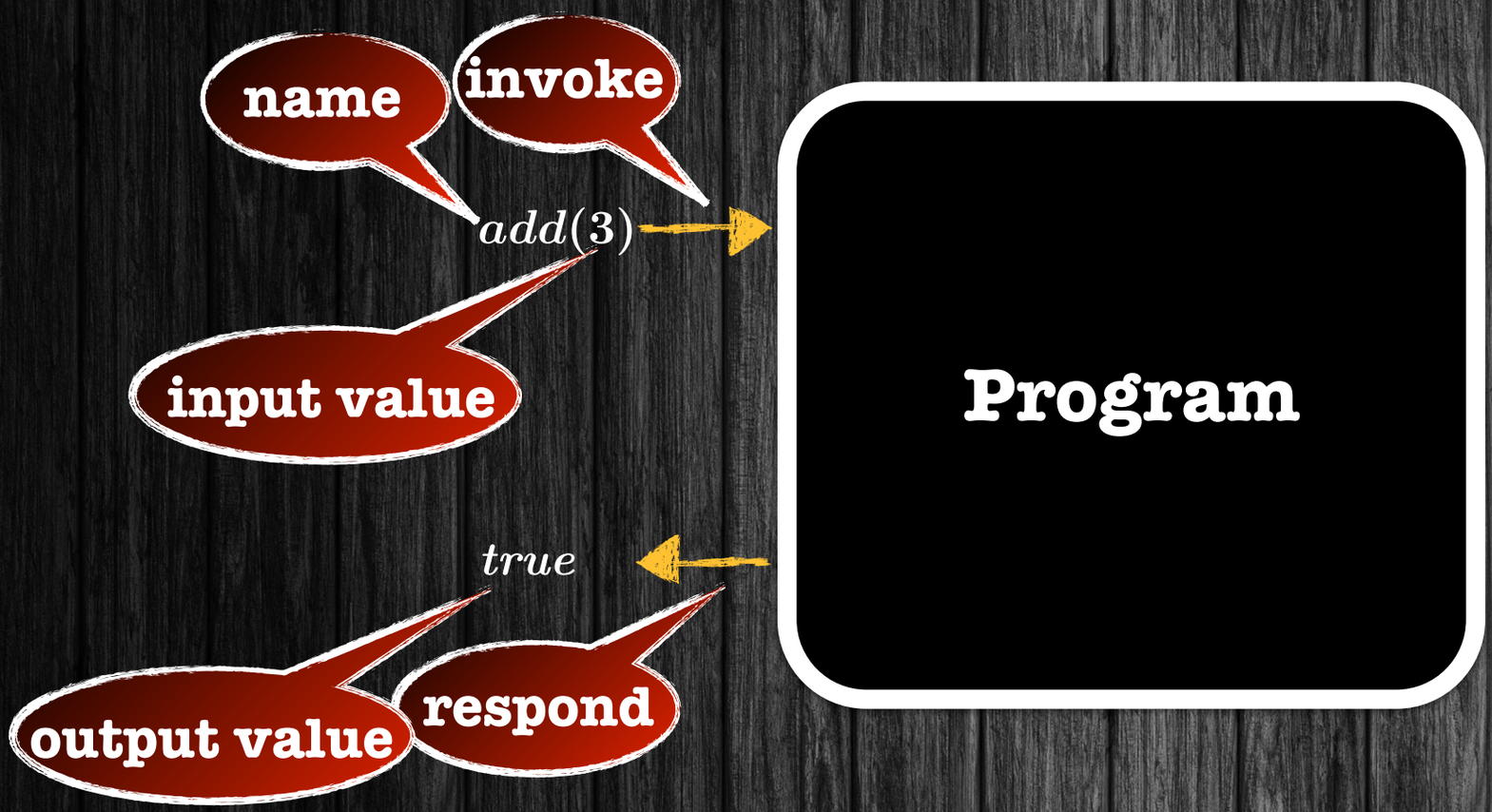


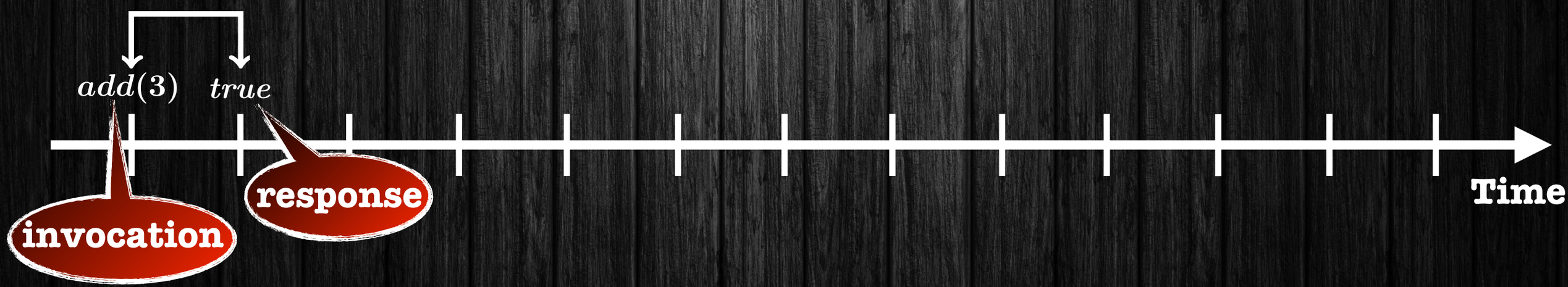
Sequential Histories



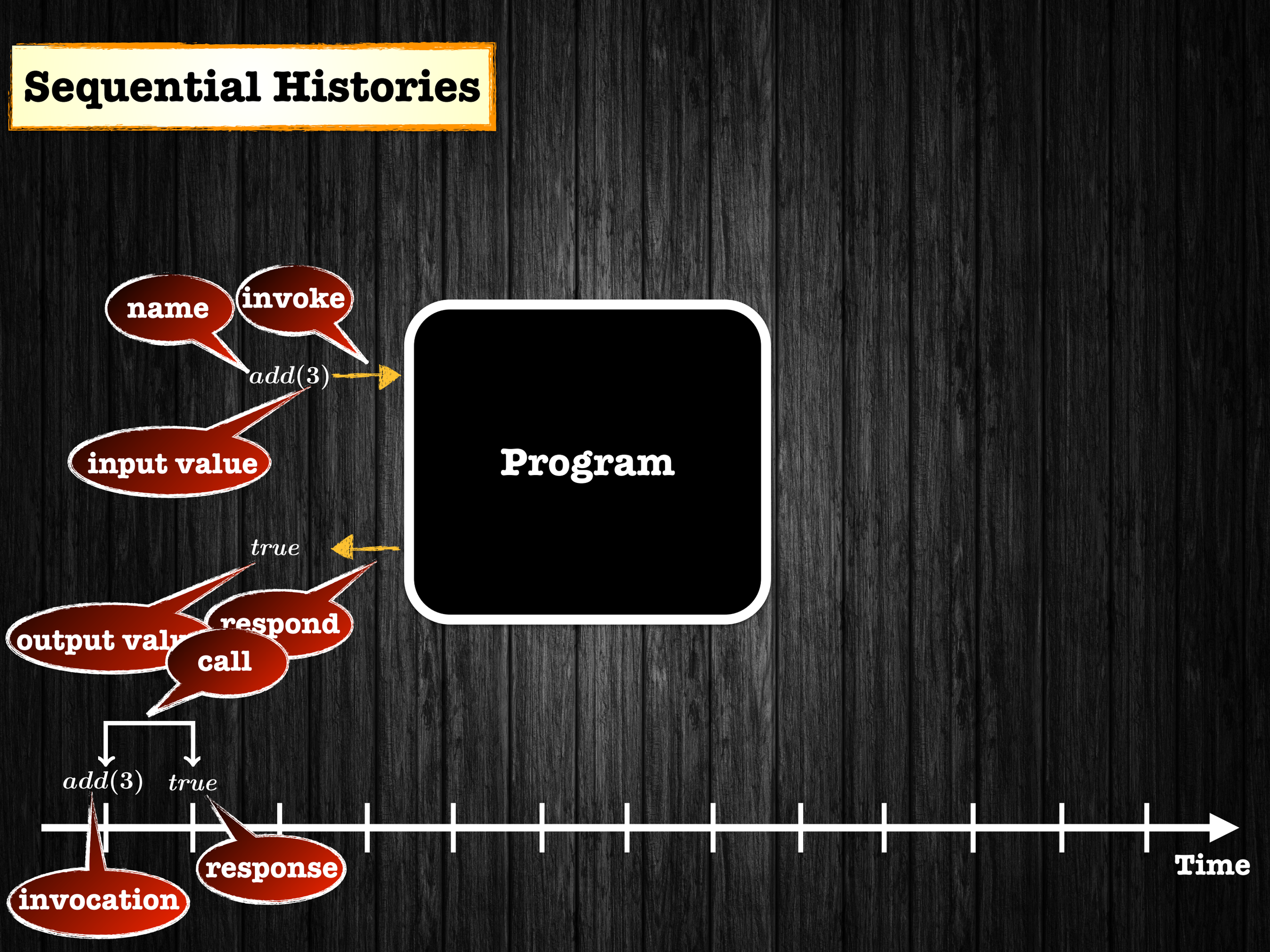


Sequential Histories

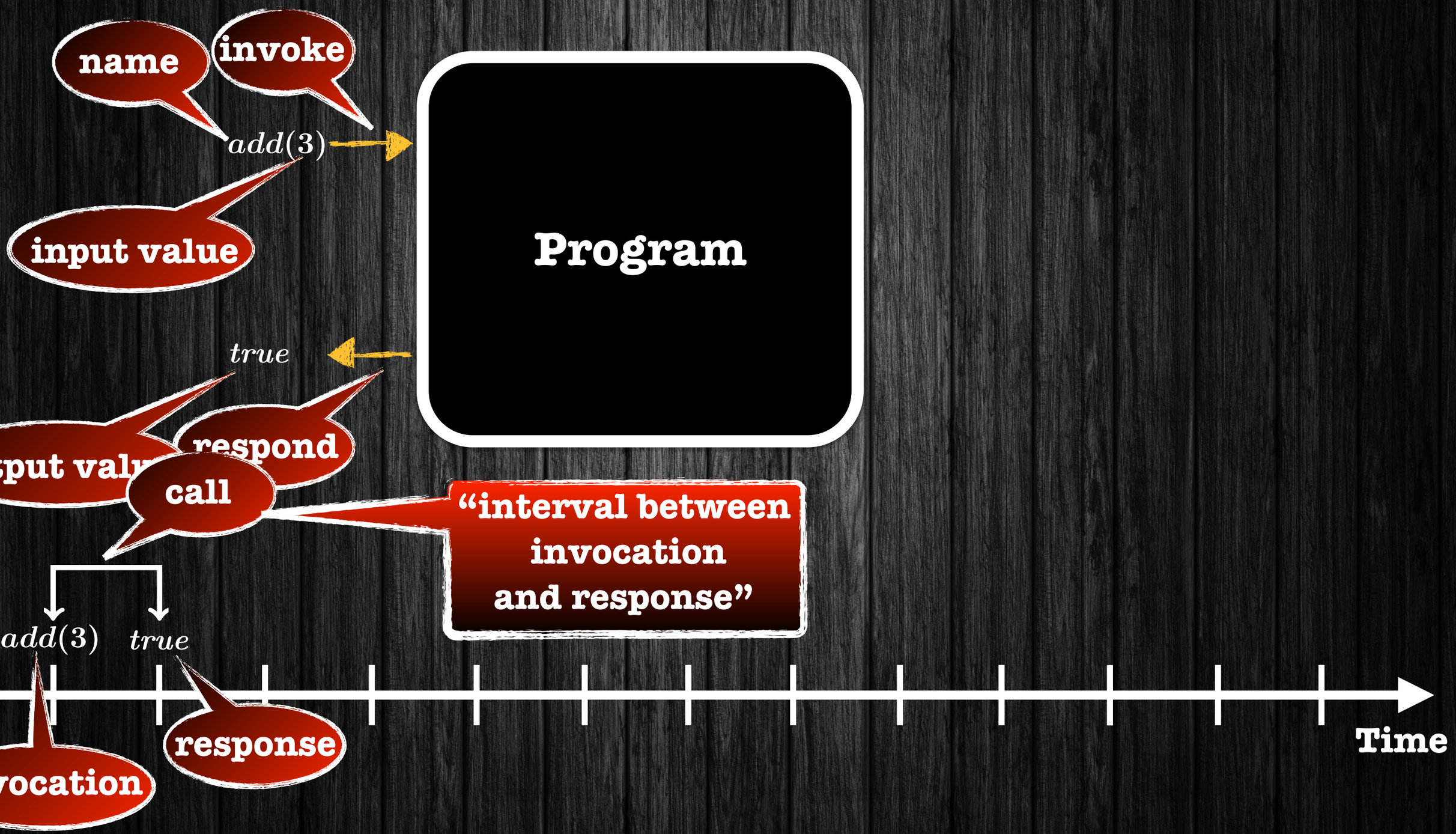




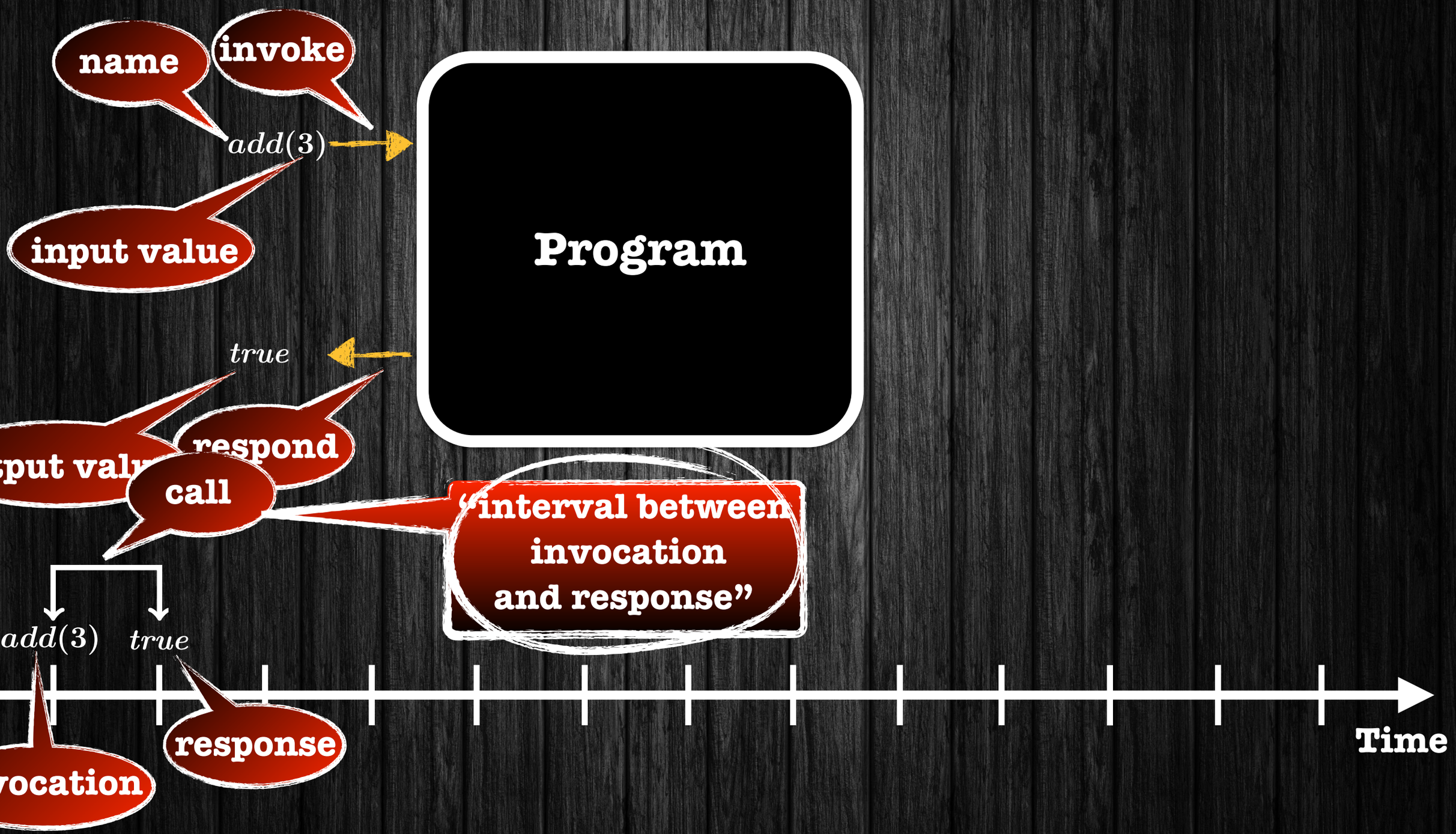
Sequential Histories



Sequential Histories

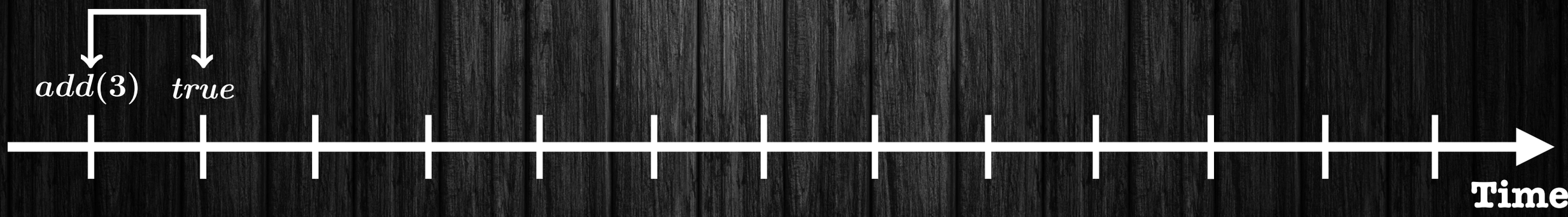


Sequential Histories

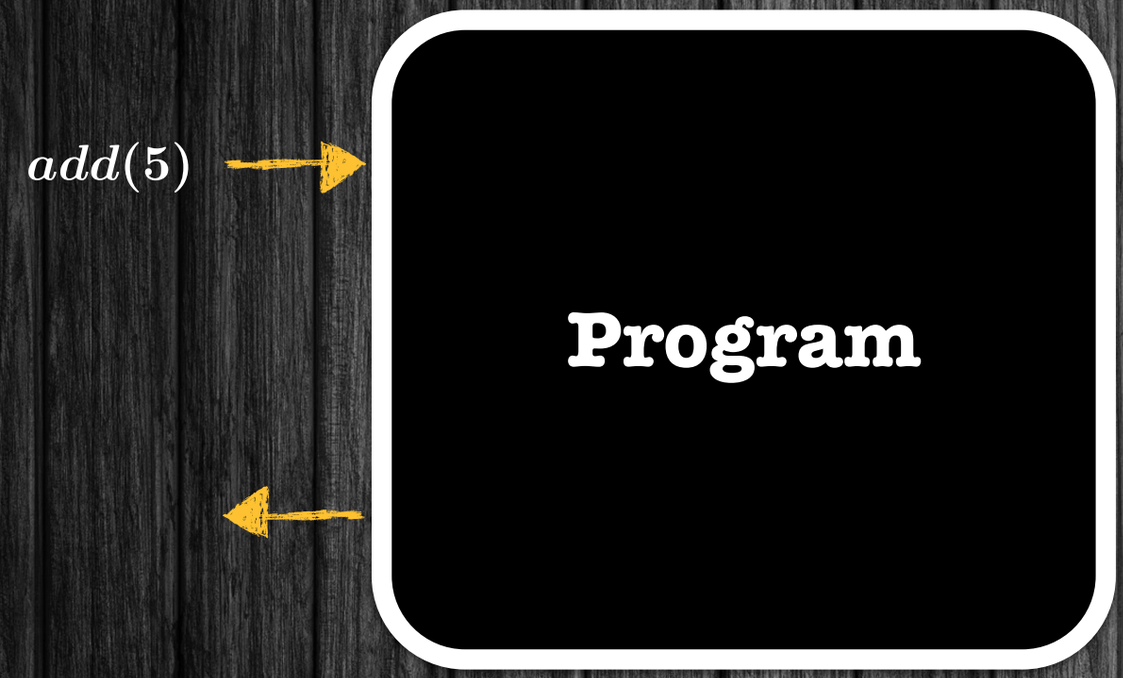


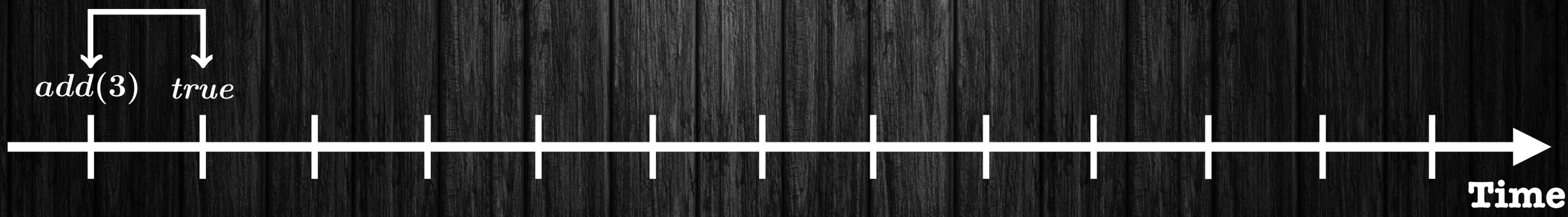
Sequential Histories



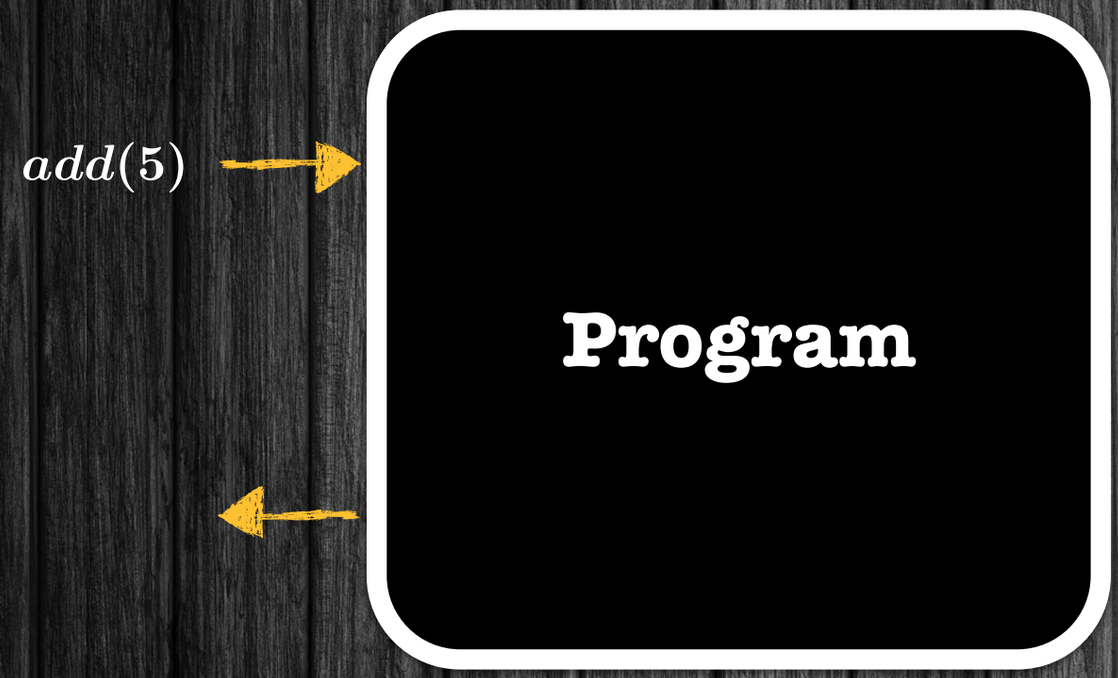


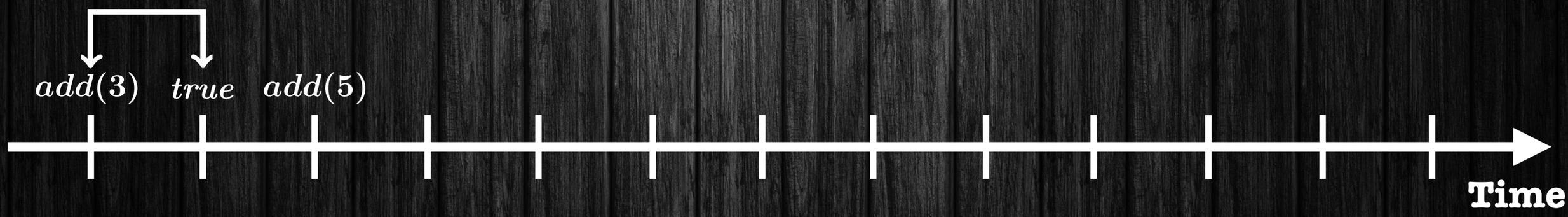
Sequential Histories



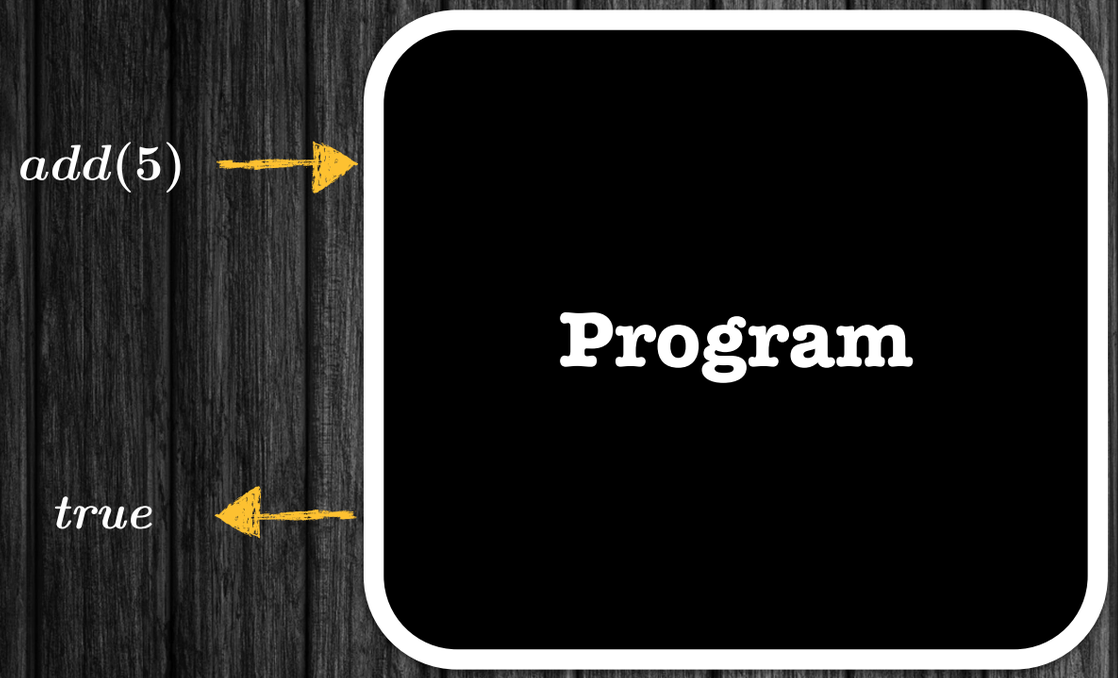


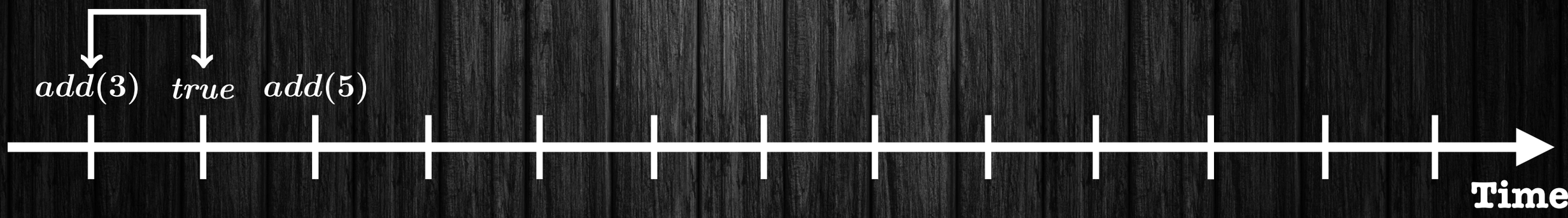
Sequential Histories



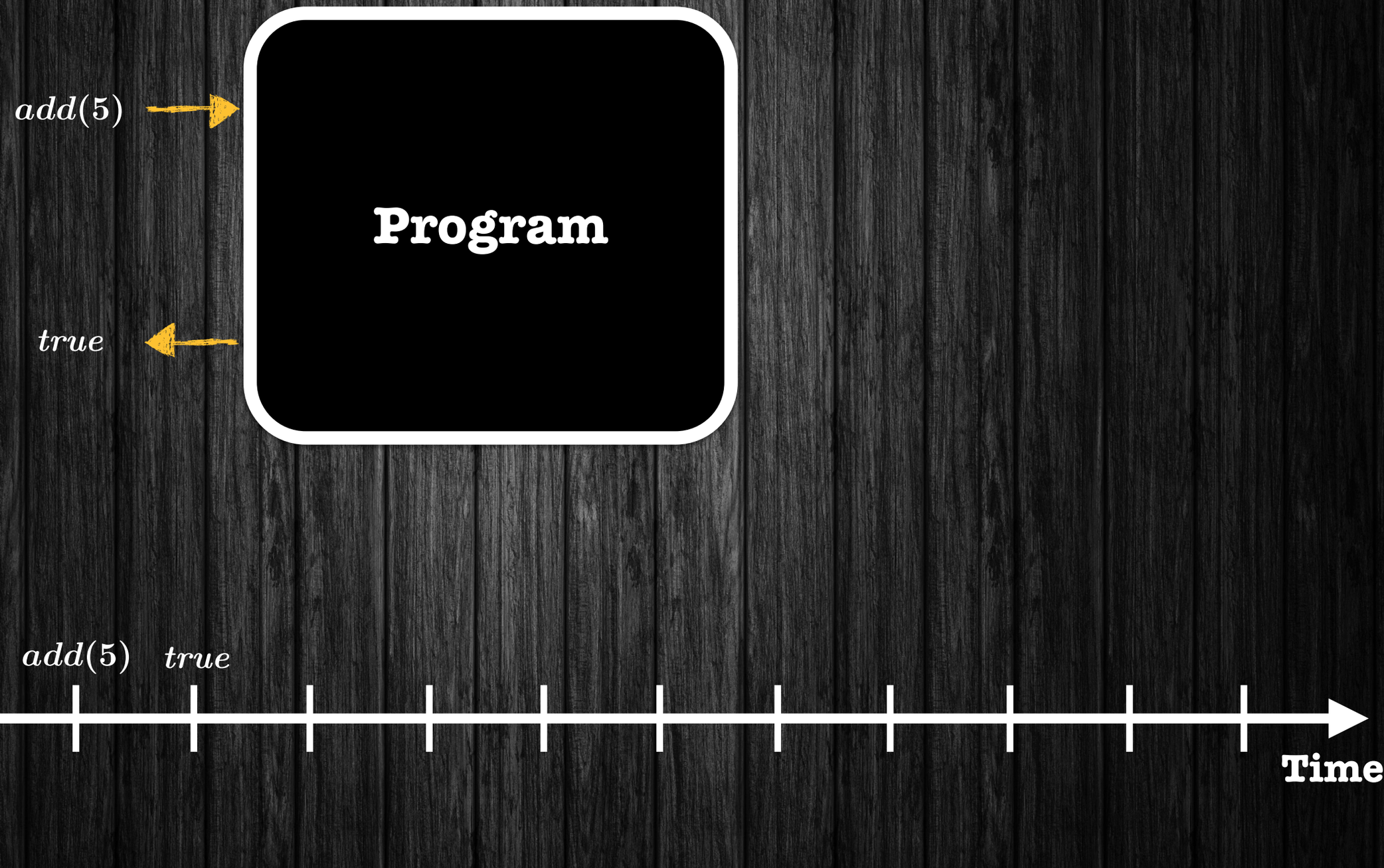


Sequential Histories

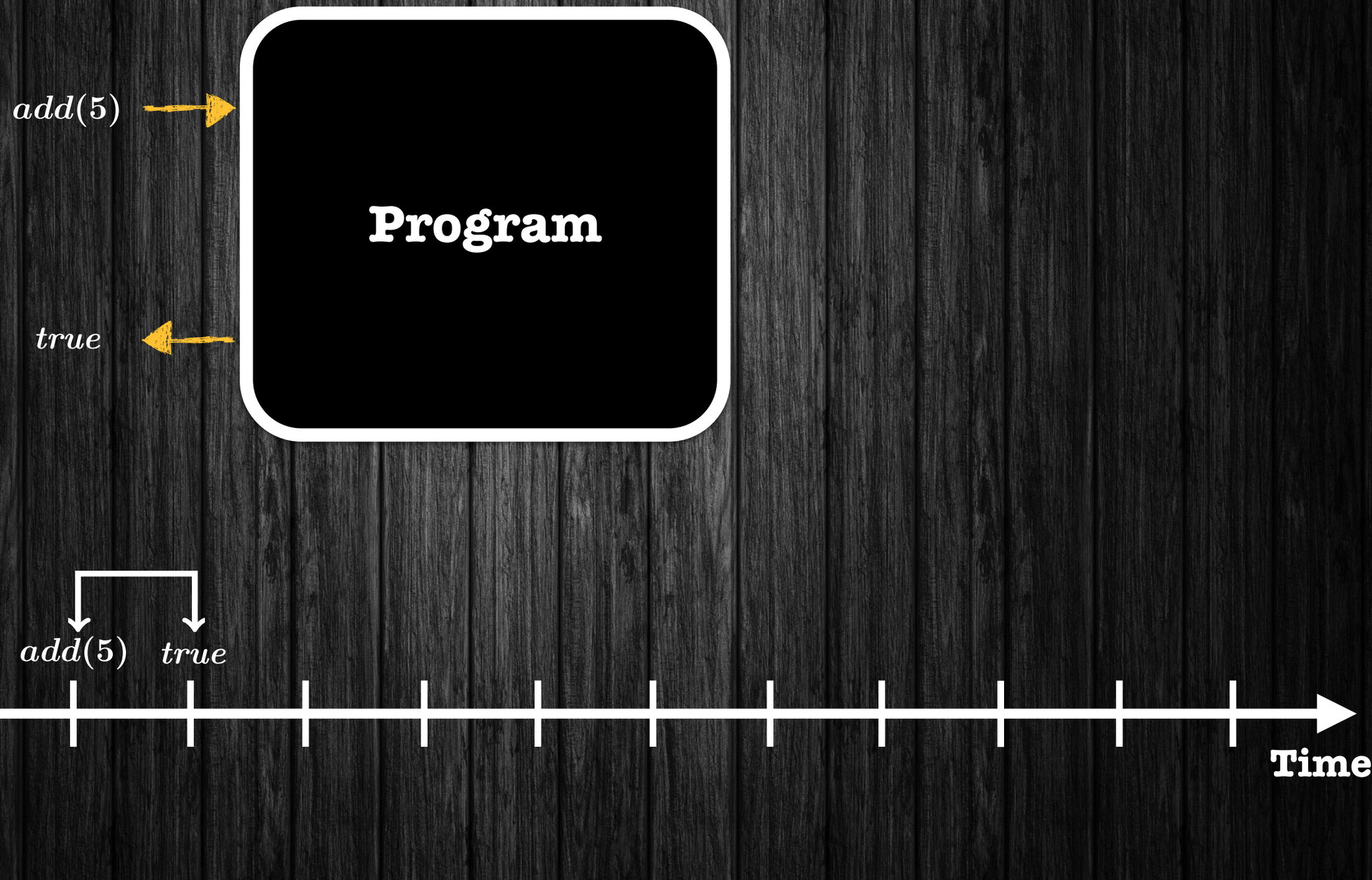




Sequential Histories

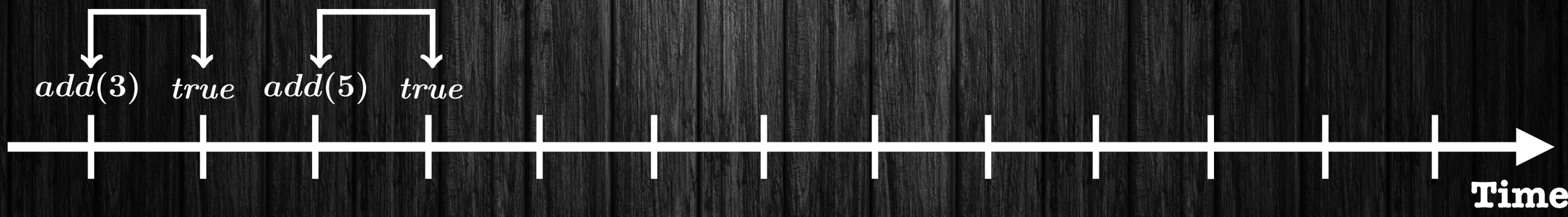


Sequential Histories

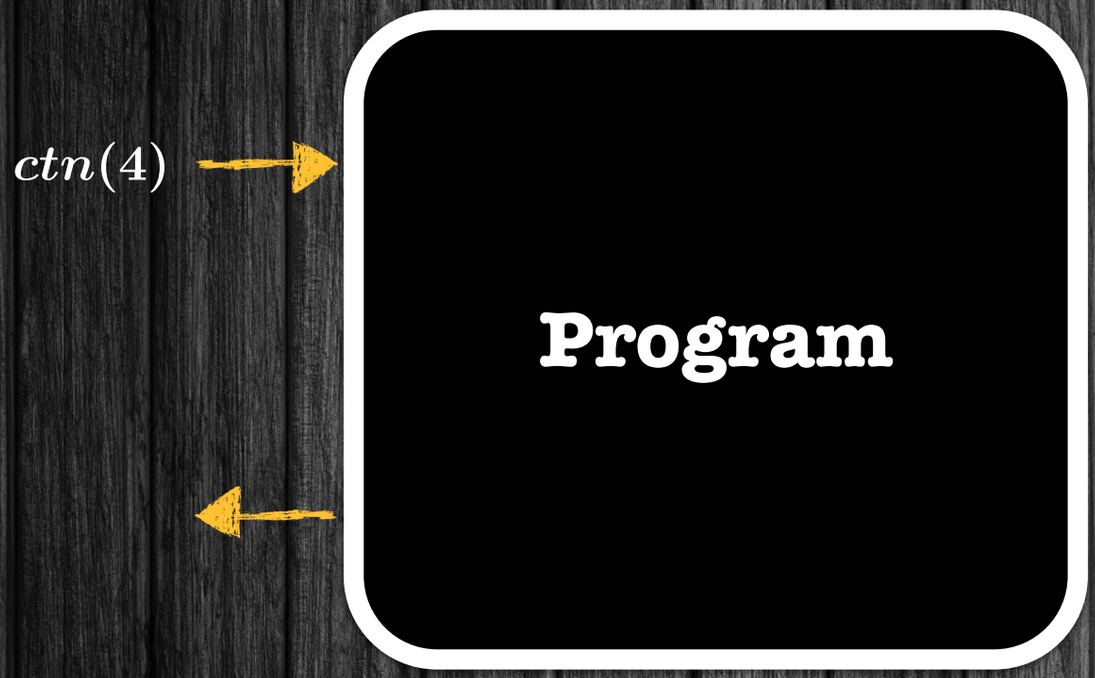


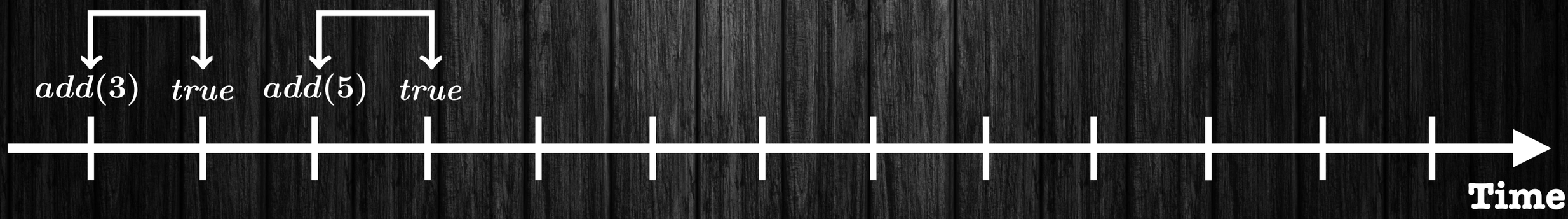
Sequential Histories



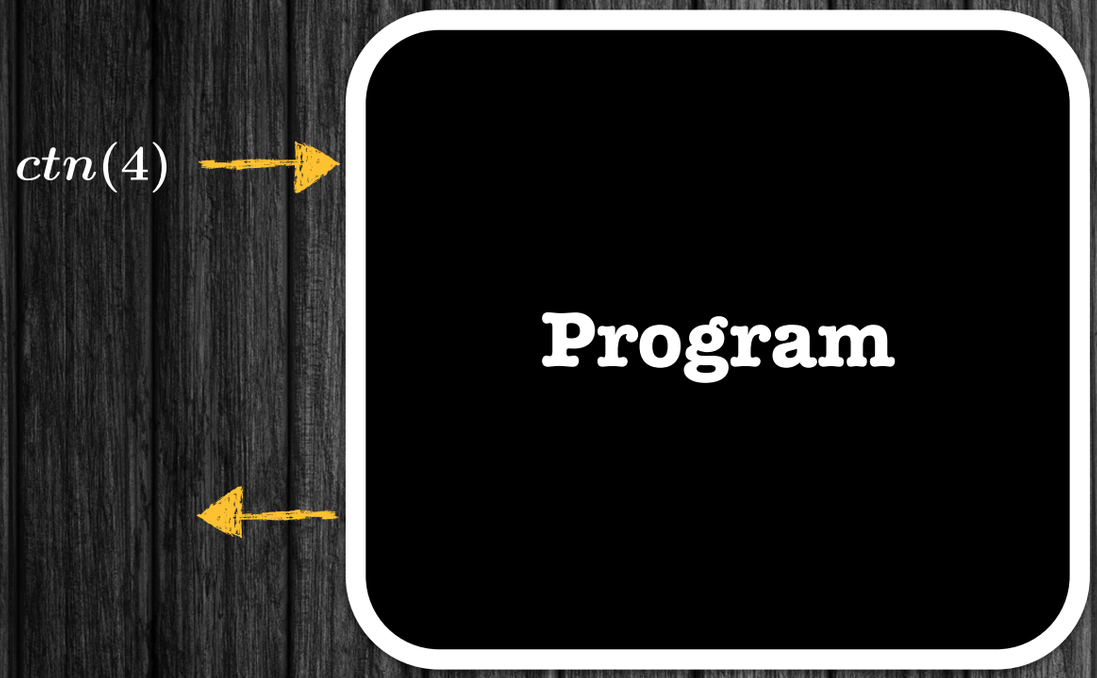


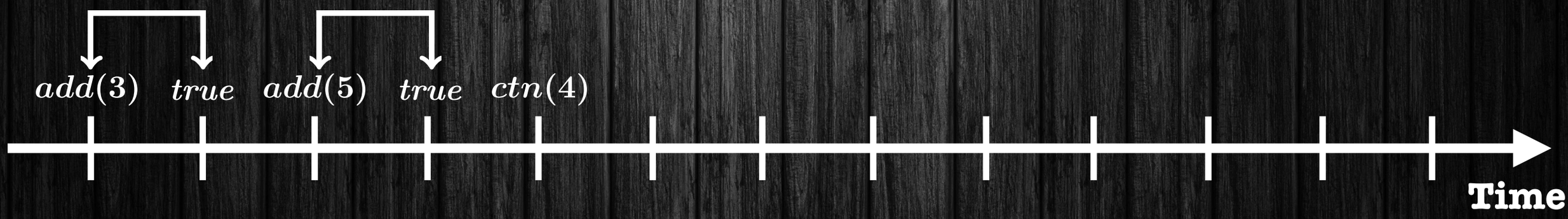
Sequential Histories



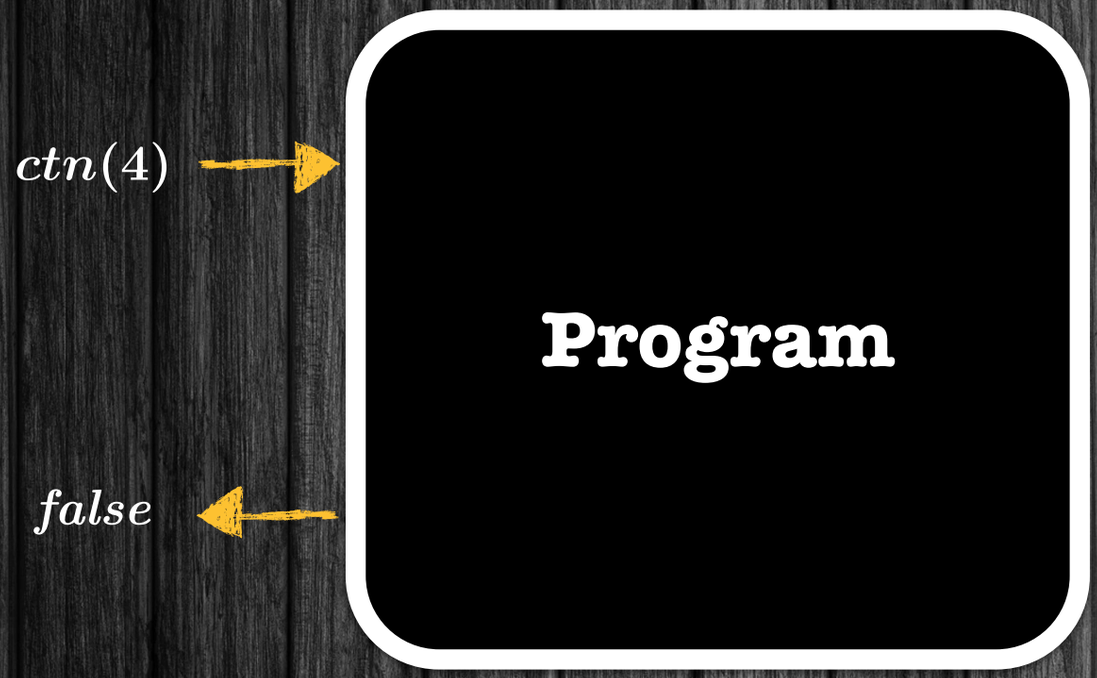


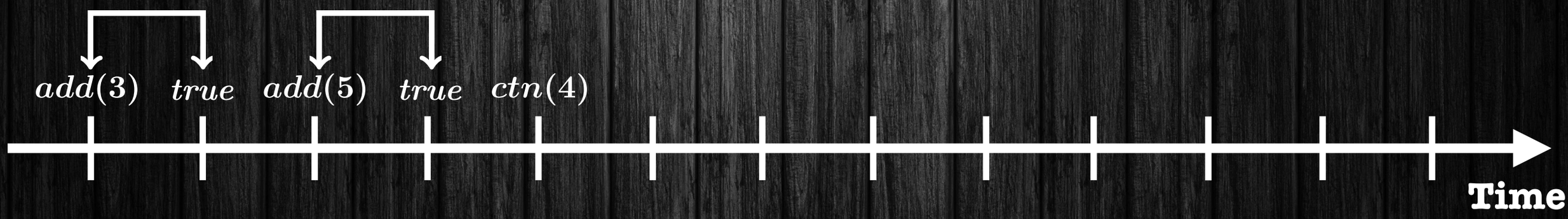
Sequential Histories



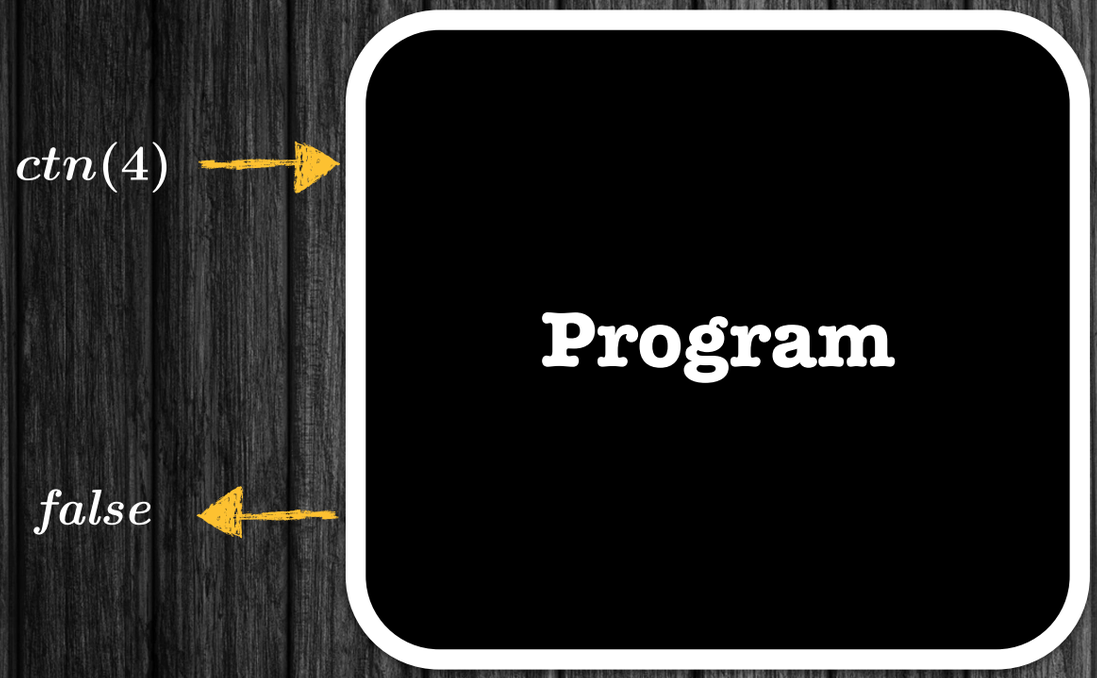


Sequential Histories



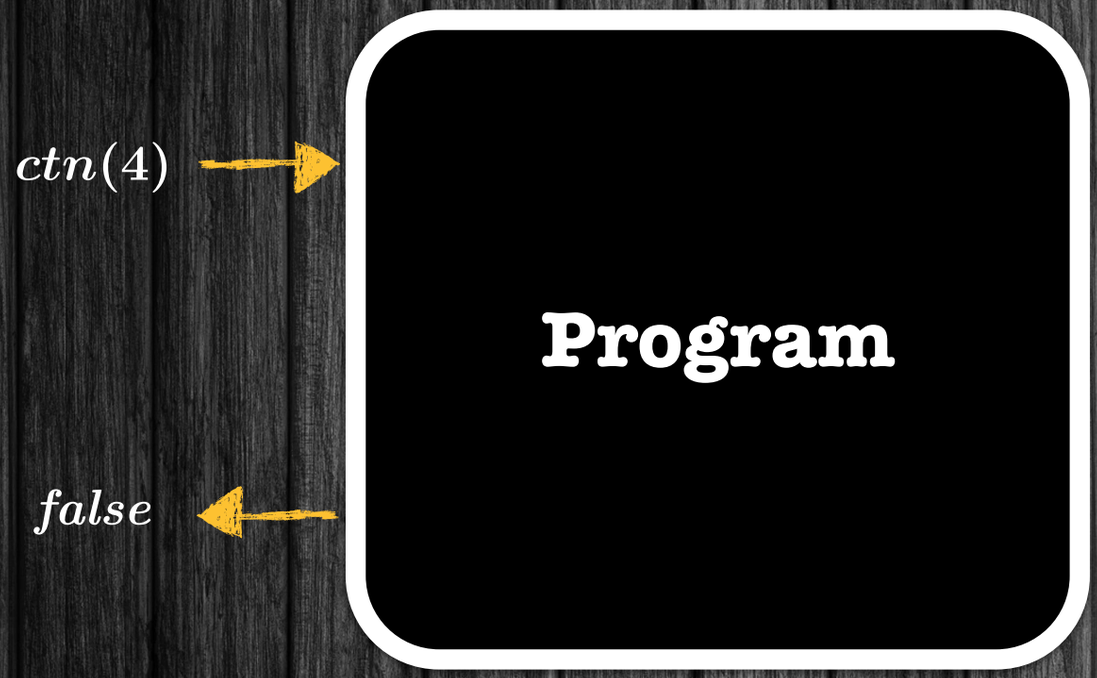


Sequential Histories





Sequential Histories



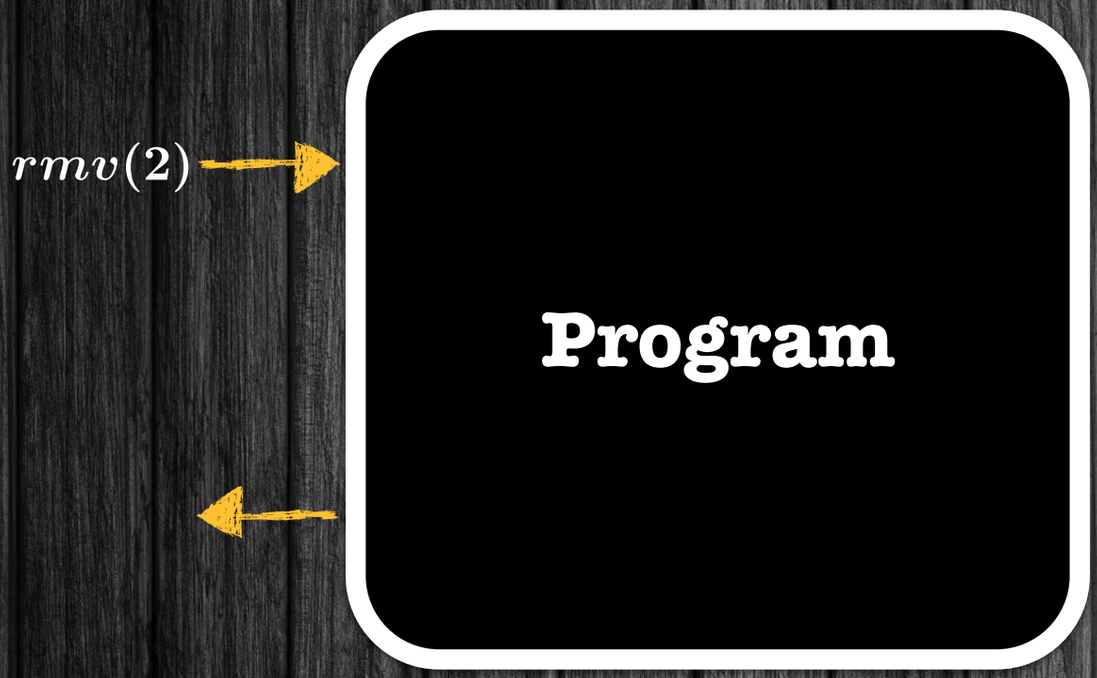


Sequential Histories



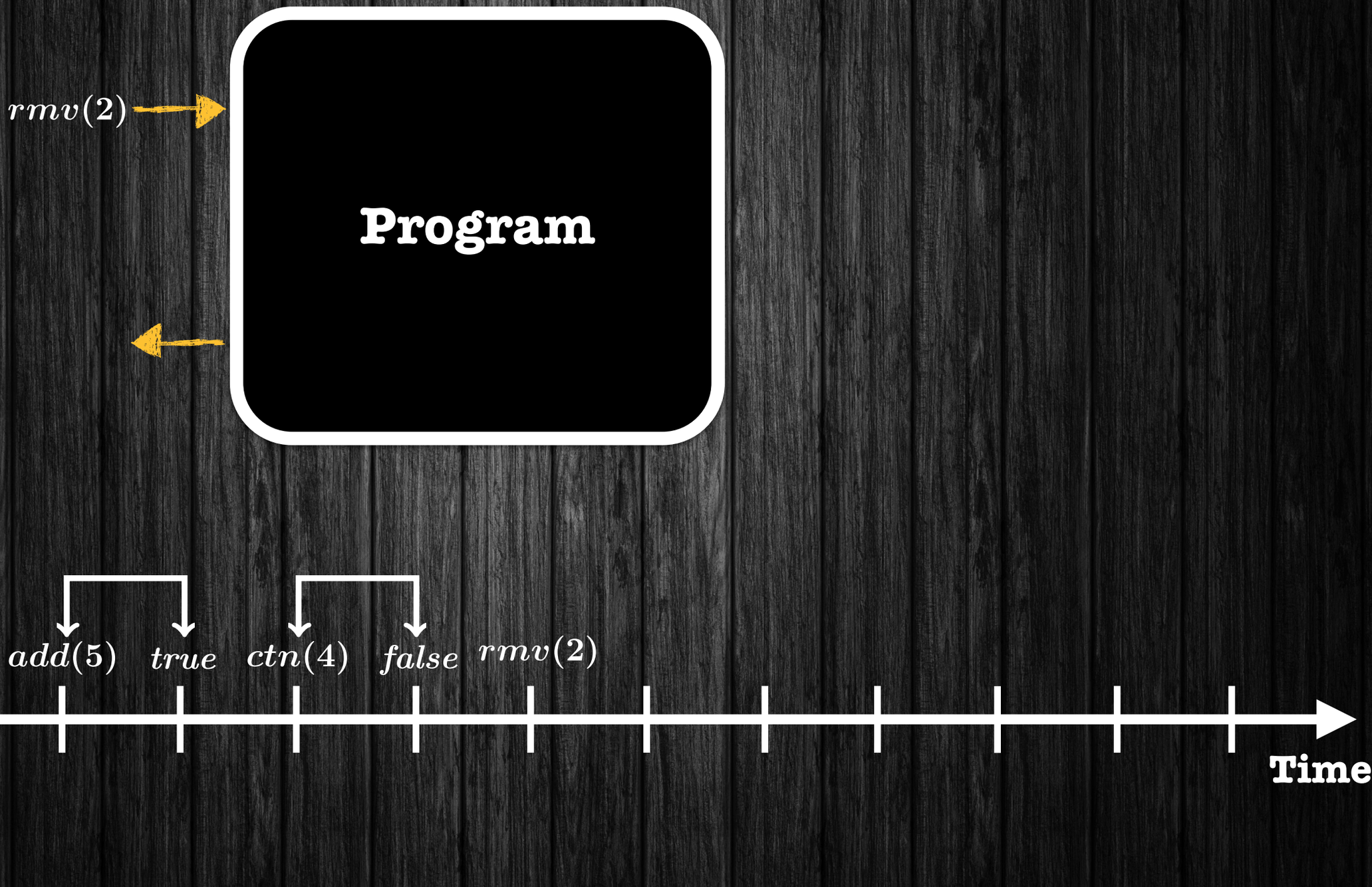


Sequential Histories

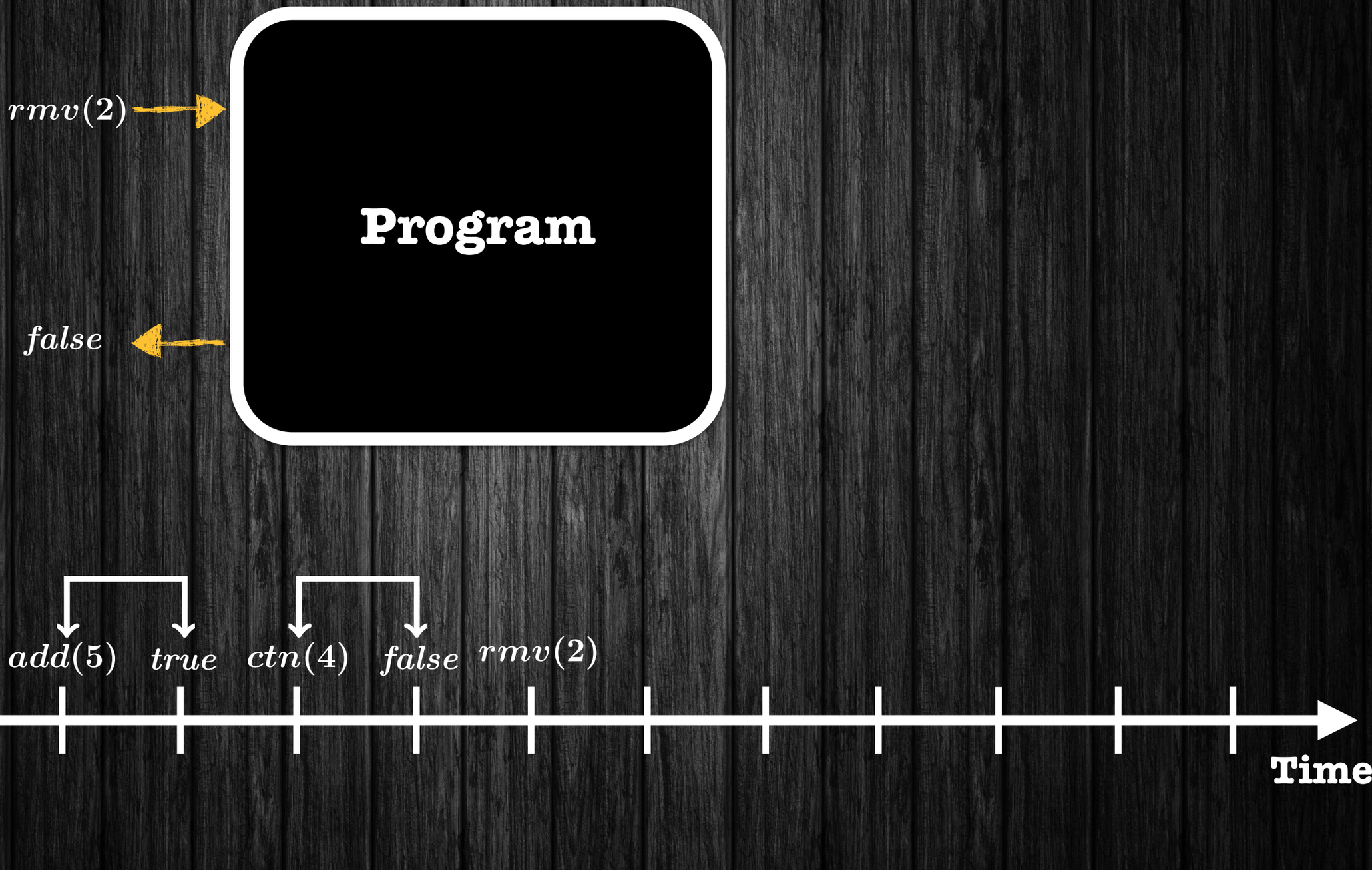




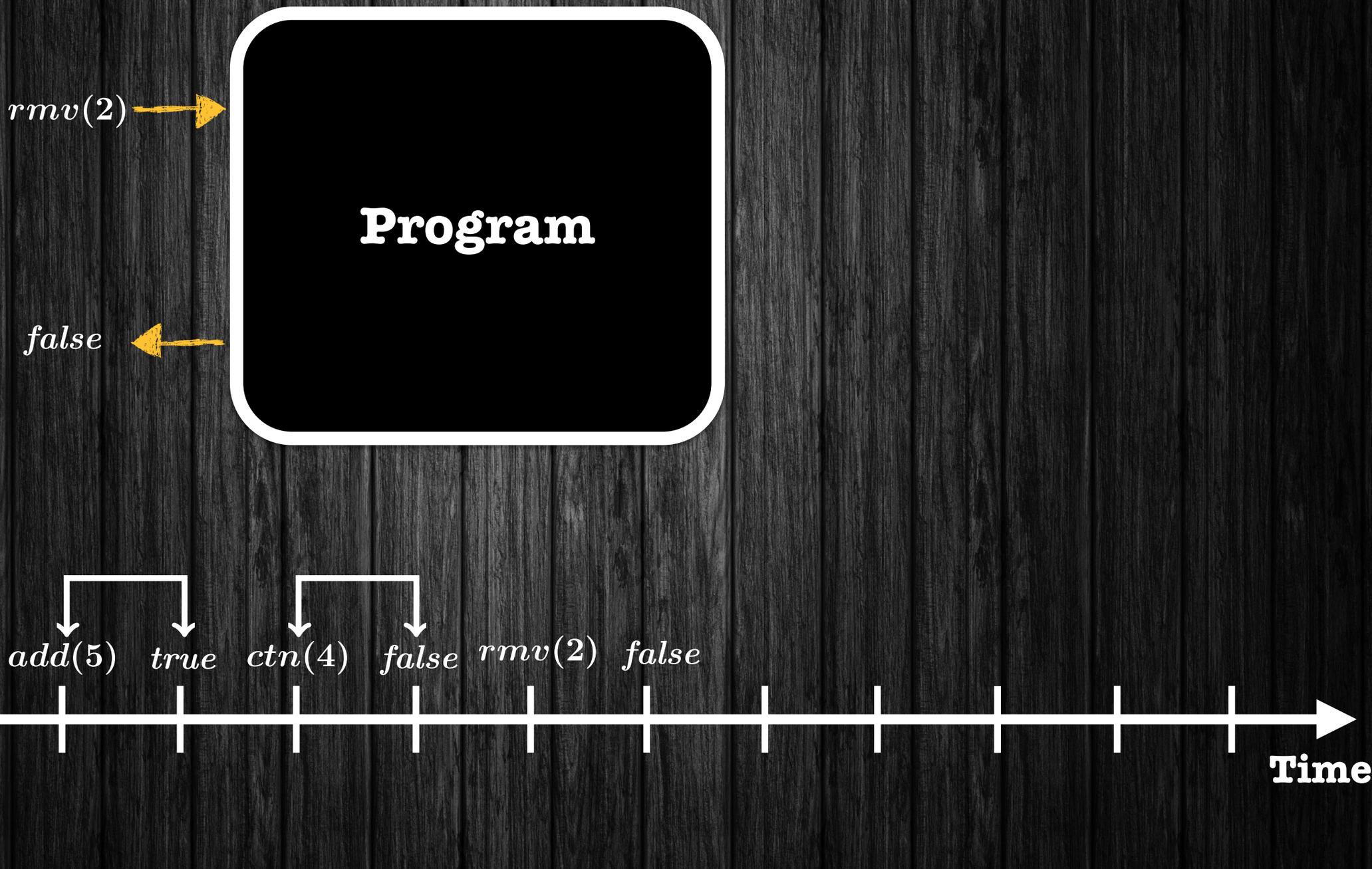
Sequential Histories



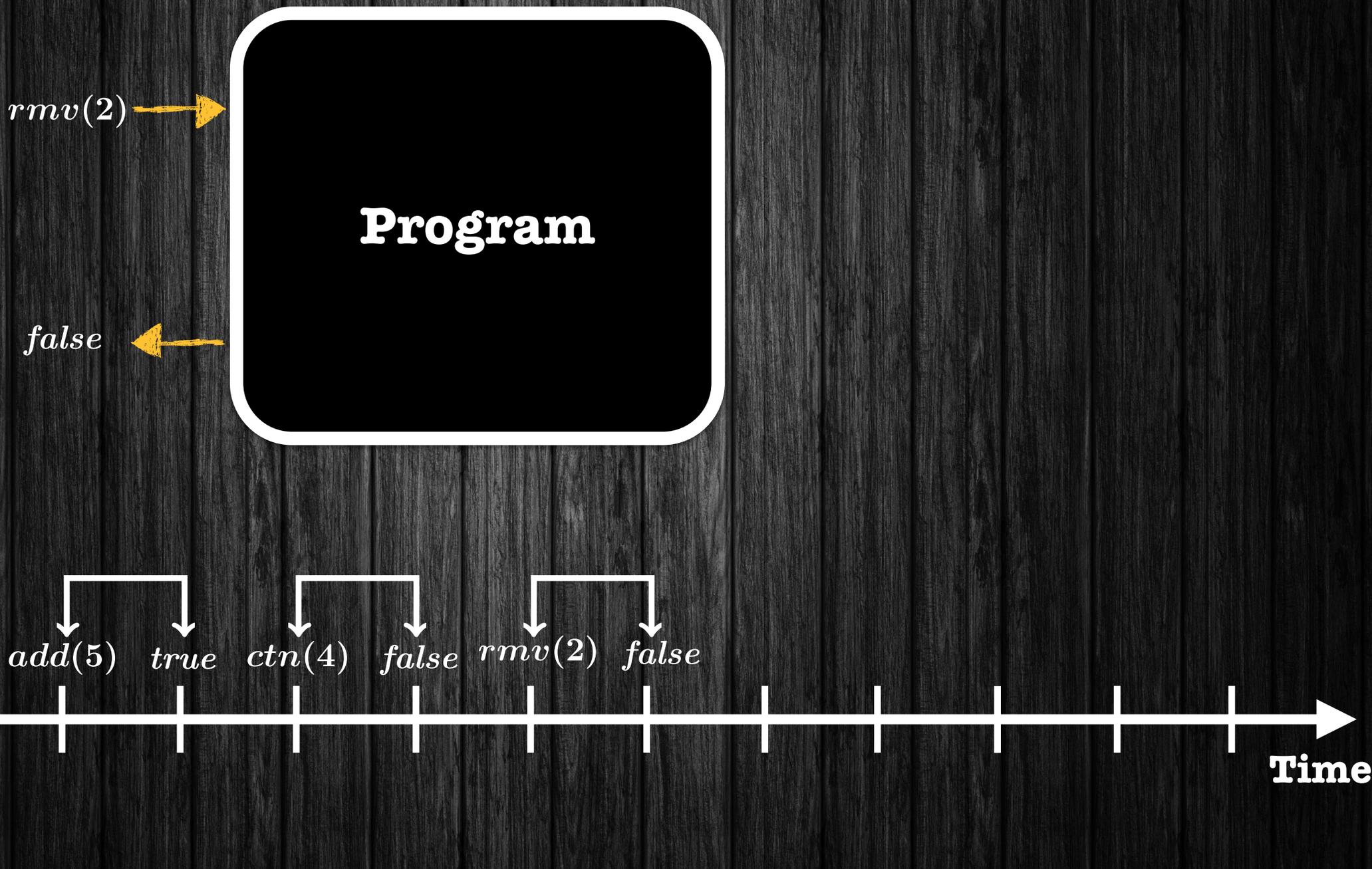
Sequential Histories



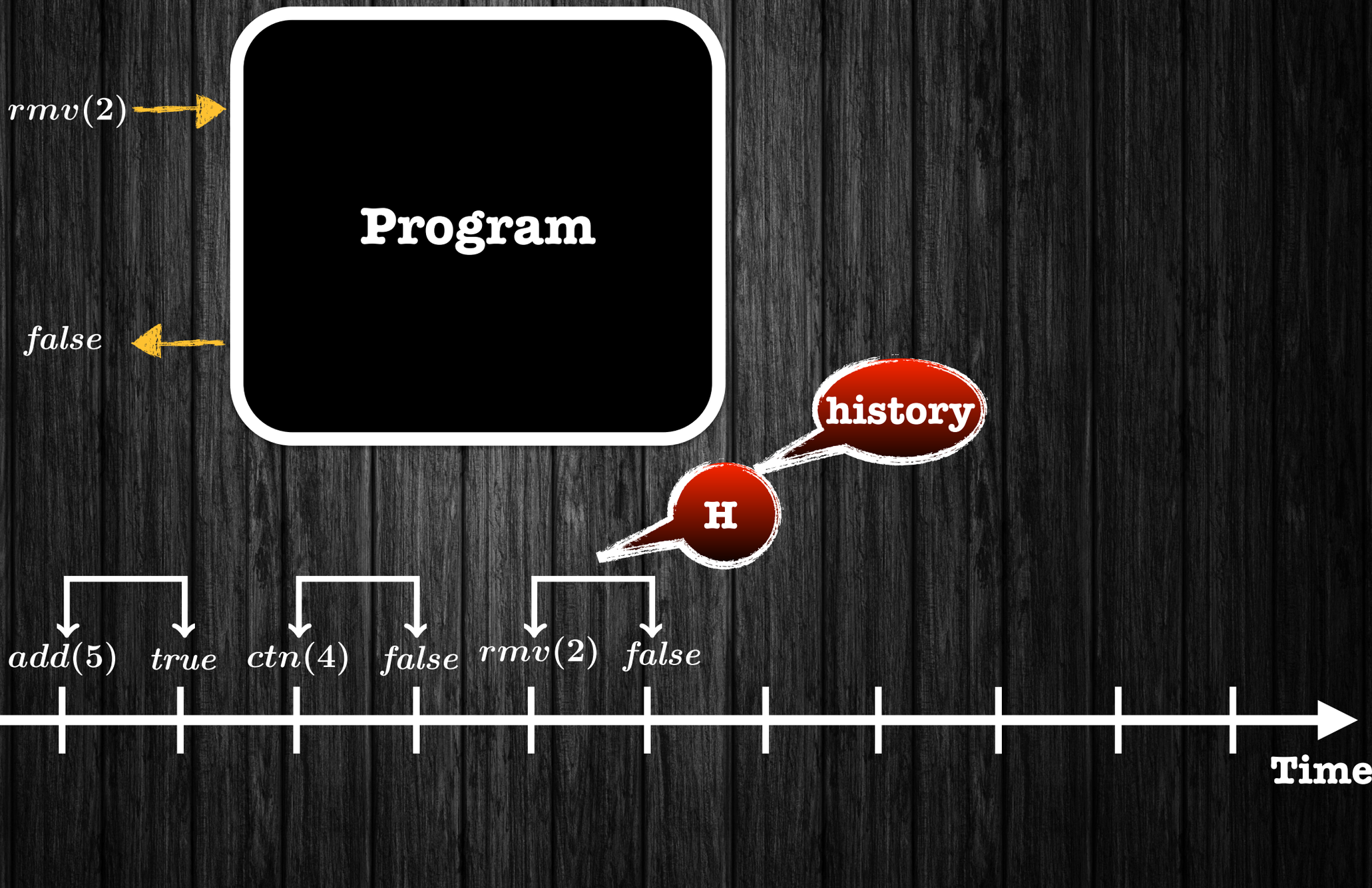
Sequential Histories



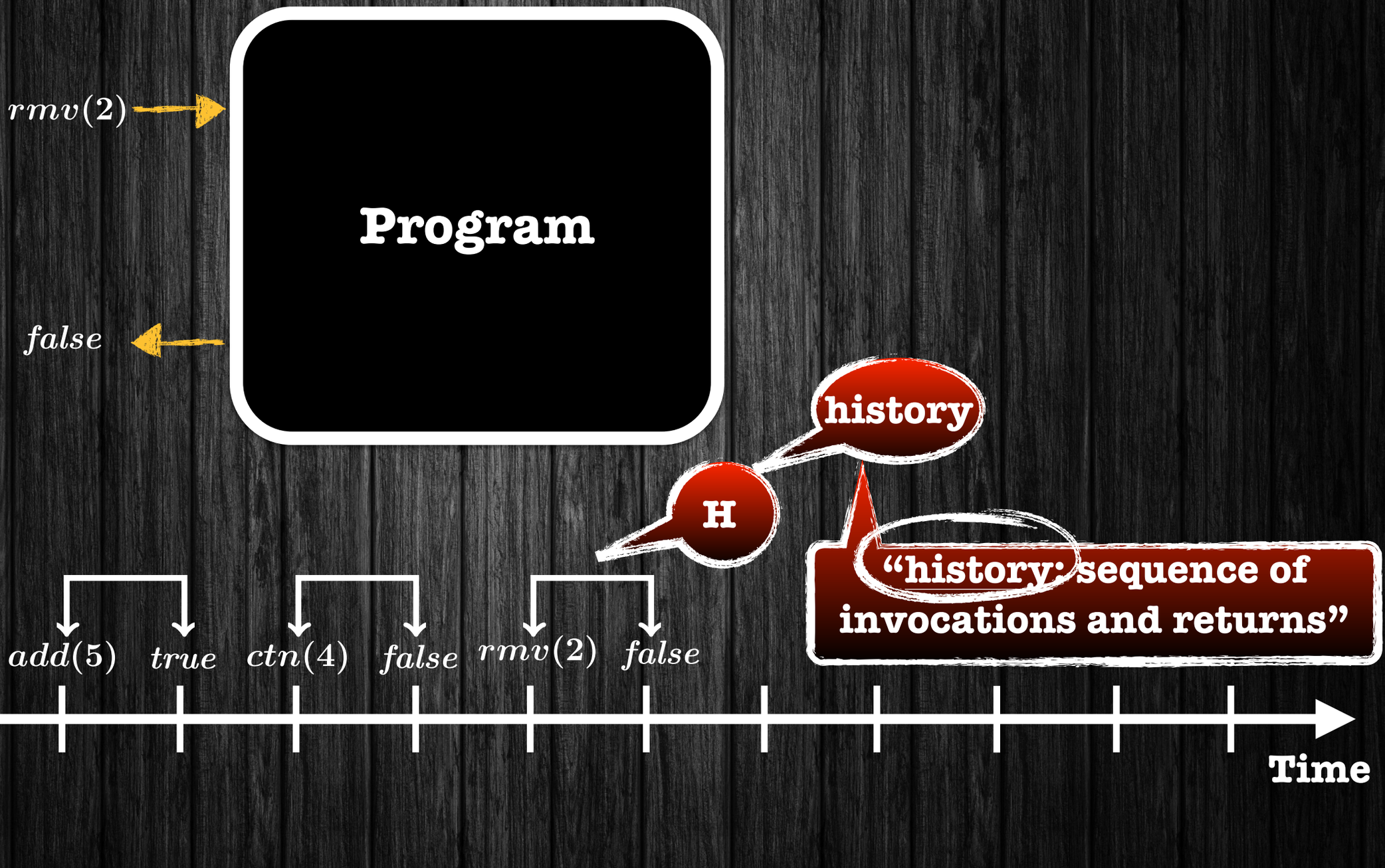
Sequential Histories



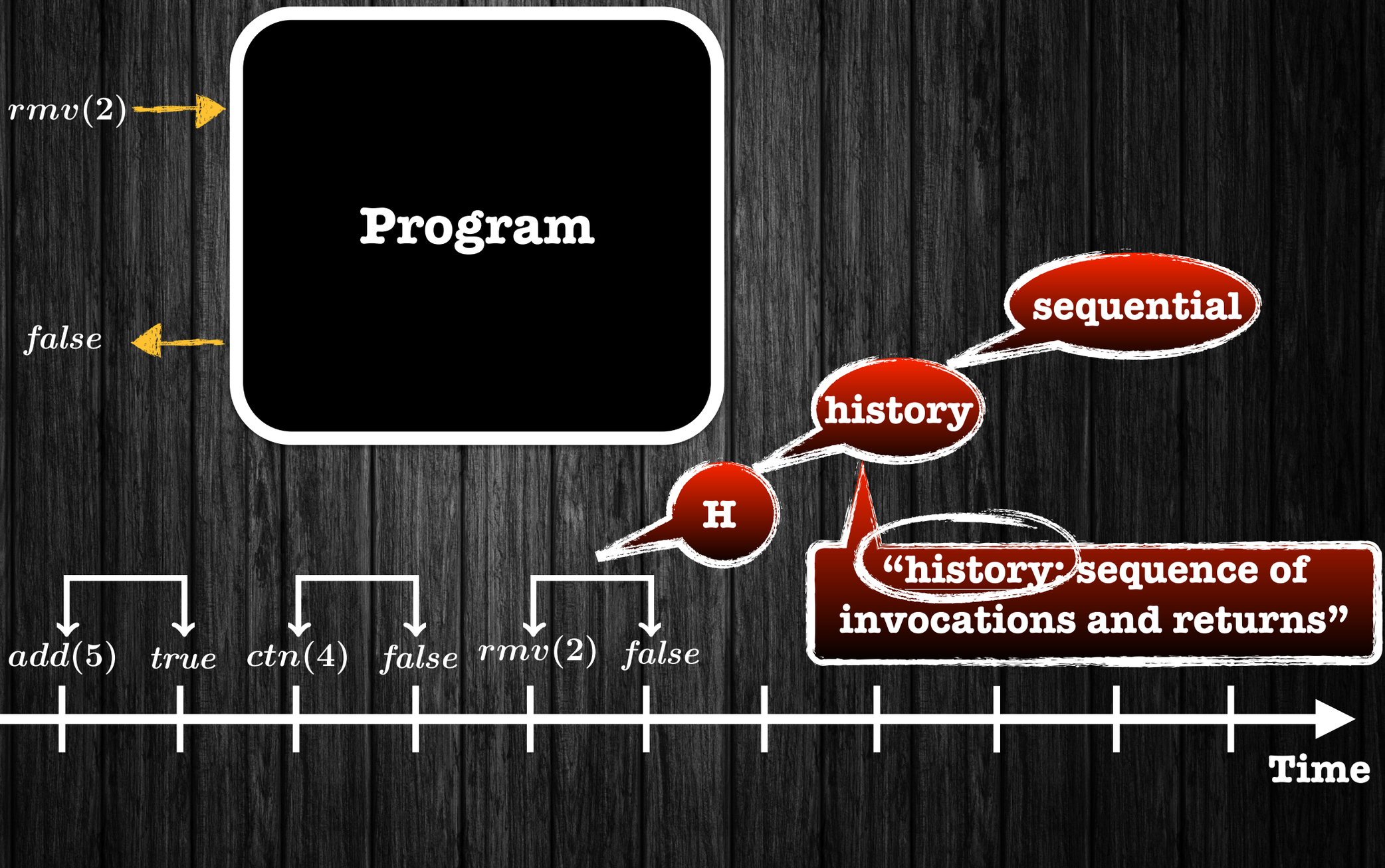
Sequential Histories



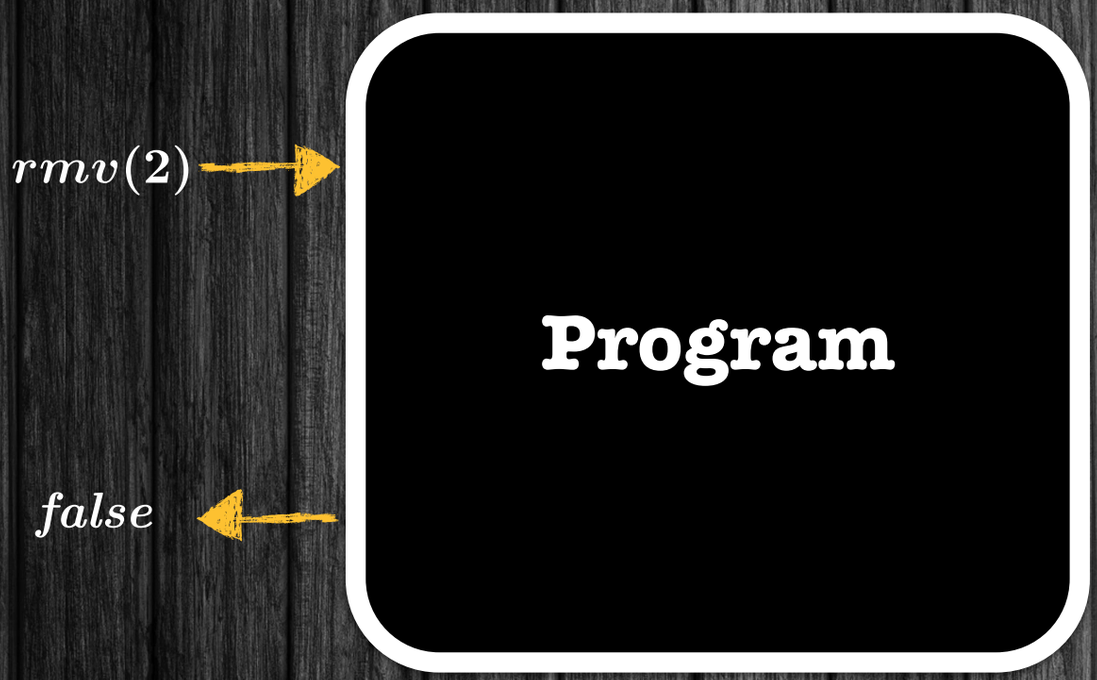
Sequential Histories



Sequential Histories



Sequential Histories



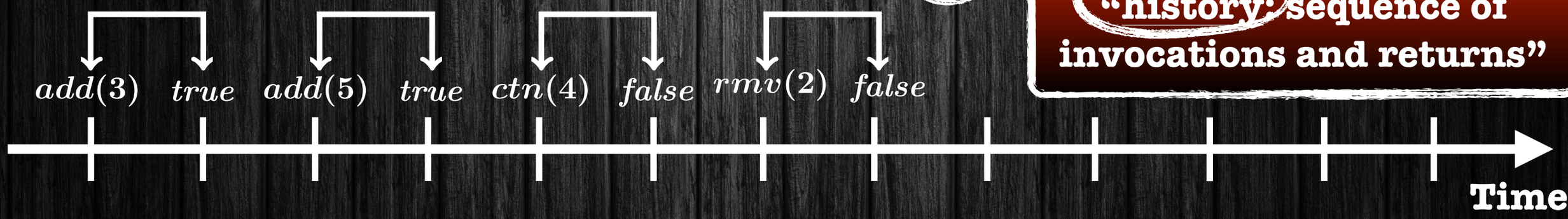
method calls do not overlap

sequential

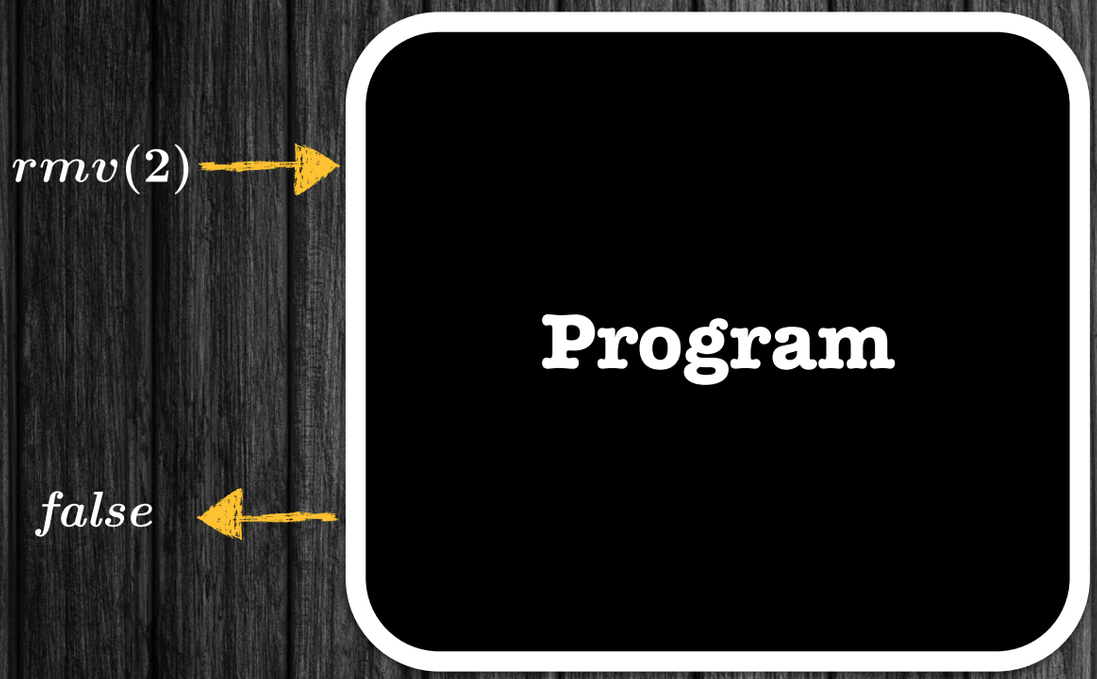
history

H

"history: sequence of invocations and returns"



Sequential Histories



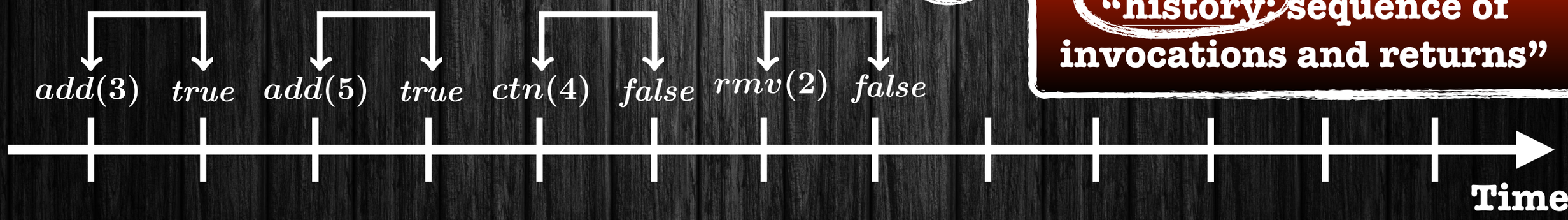
method calls do not overlap

sequential

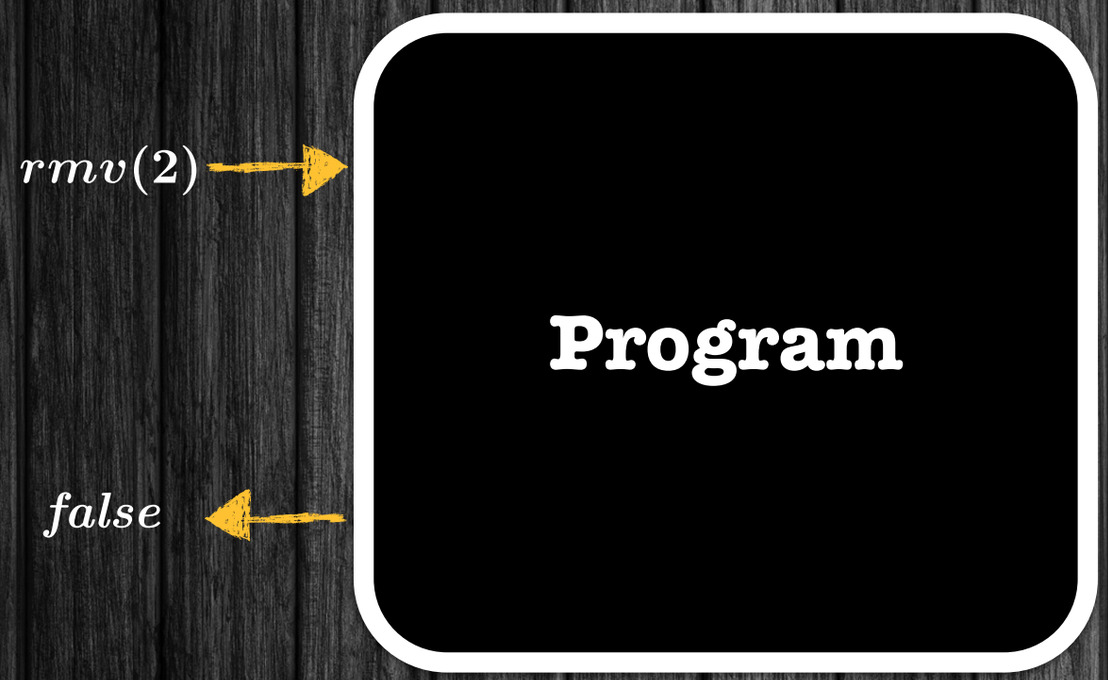
history

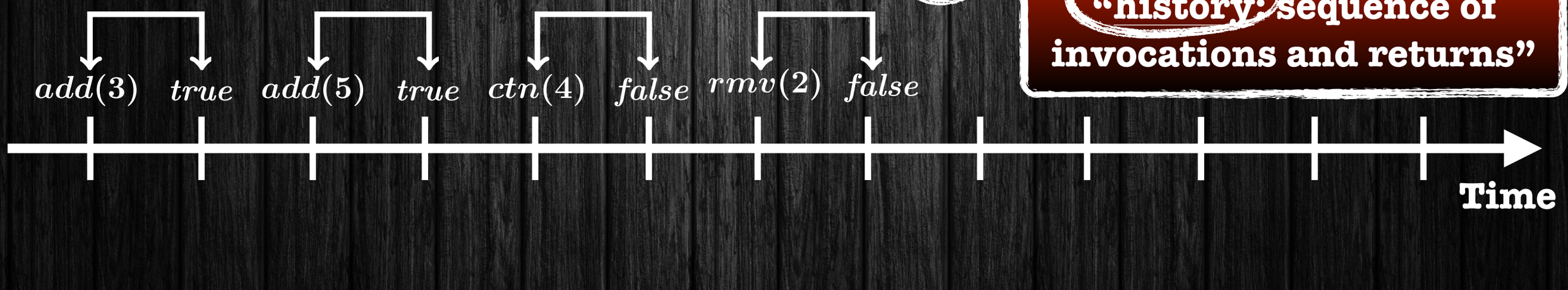
H

"history: sequence of invocations and returns"



Sequential Histories





invocation followed immediately by response

method calls do not overlap

sequential

history

H

"history: sequence of invocations and returns"

Sequential Histories

invocation followed immediately by response

method calls do not overlap

sequential

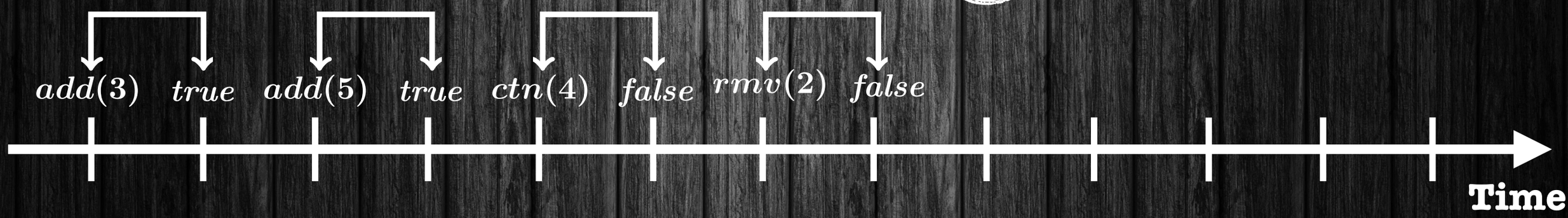
history

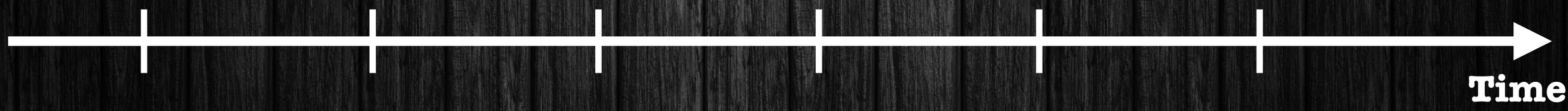
H

invoke

respond

Program





Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

respond

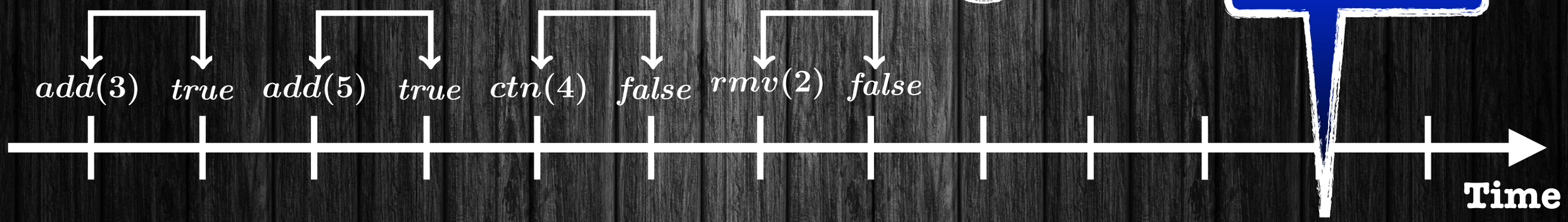
Program

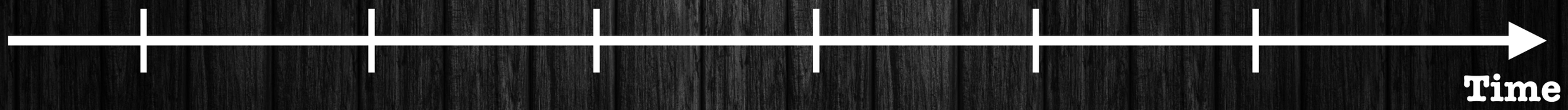
sequential

history

H

Traceof(H)





Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

respond

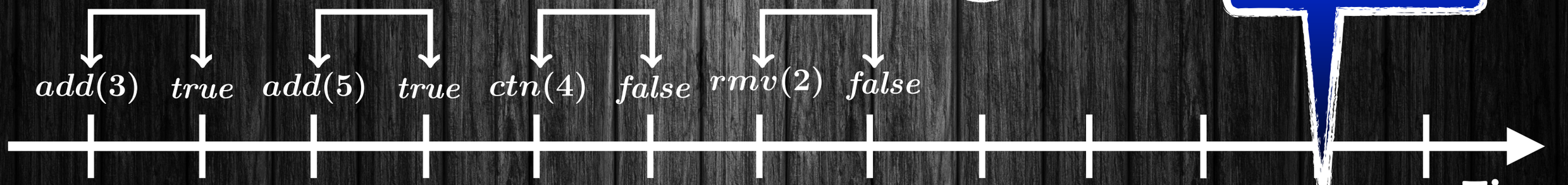
Program

sequential

history

H

Traceof(H)



"merge each invocation with corresponding response"



Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

respond

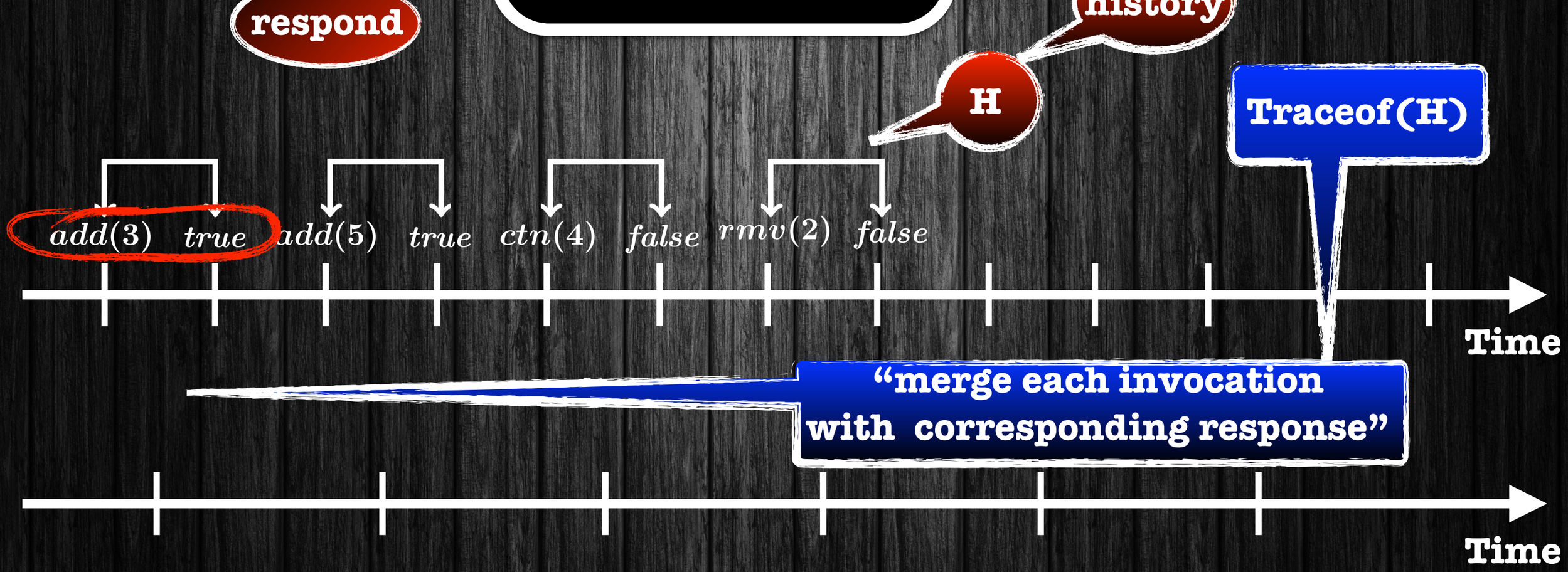
Program

sequential

history

H

Traceof(H)



"merge each invocation with corresponding response"

Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

Program

respond

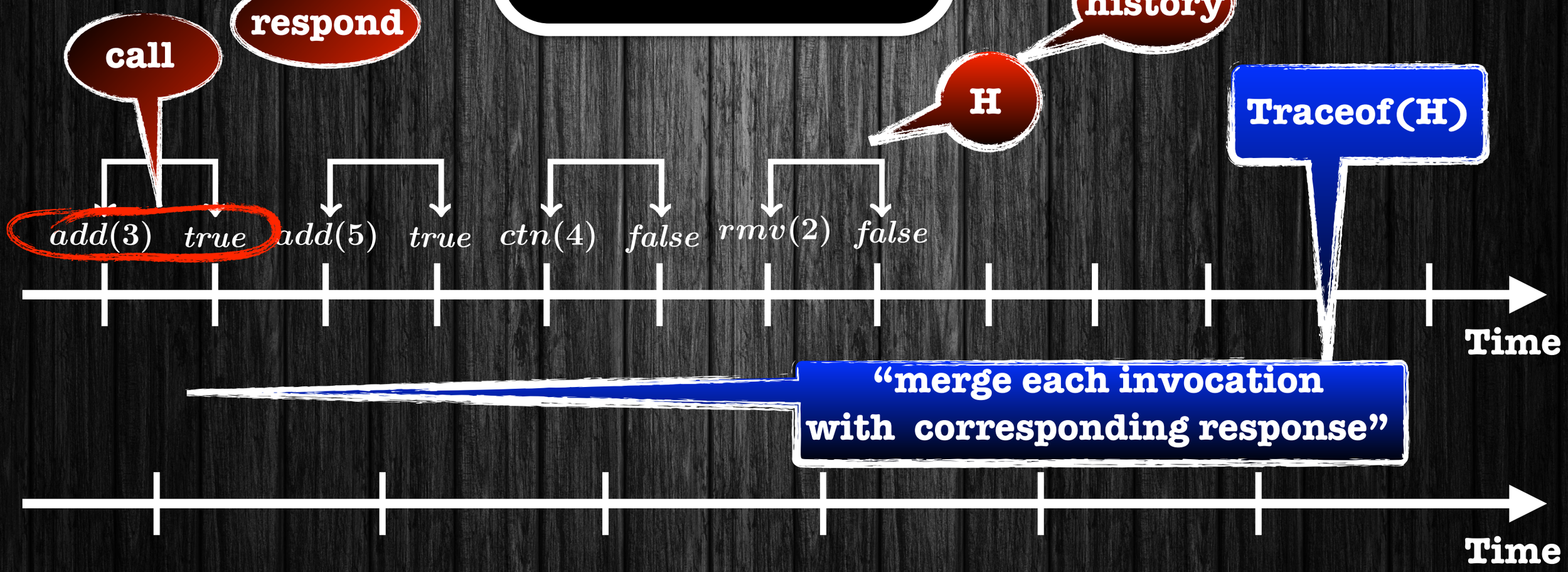
call

sequential

history

H

Traceof(H)



"merge each invocation with corresponding response"

Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

Program

respond

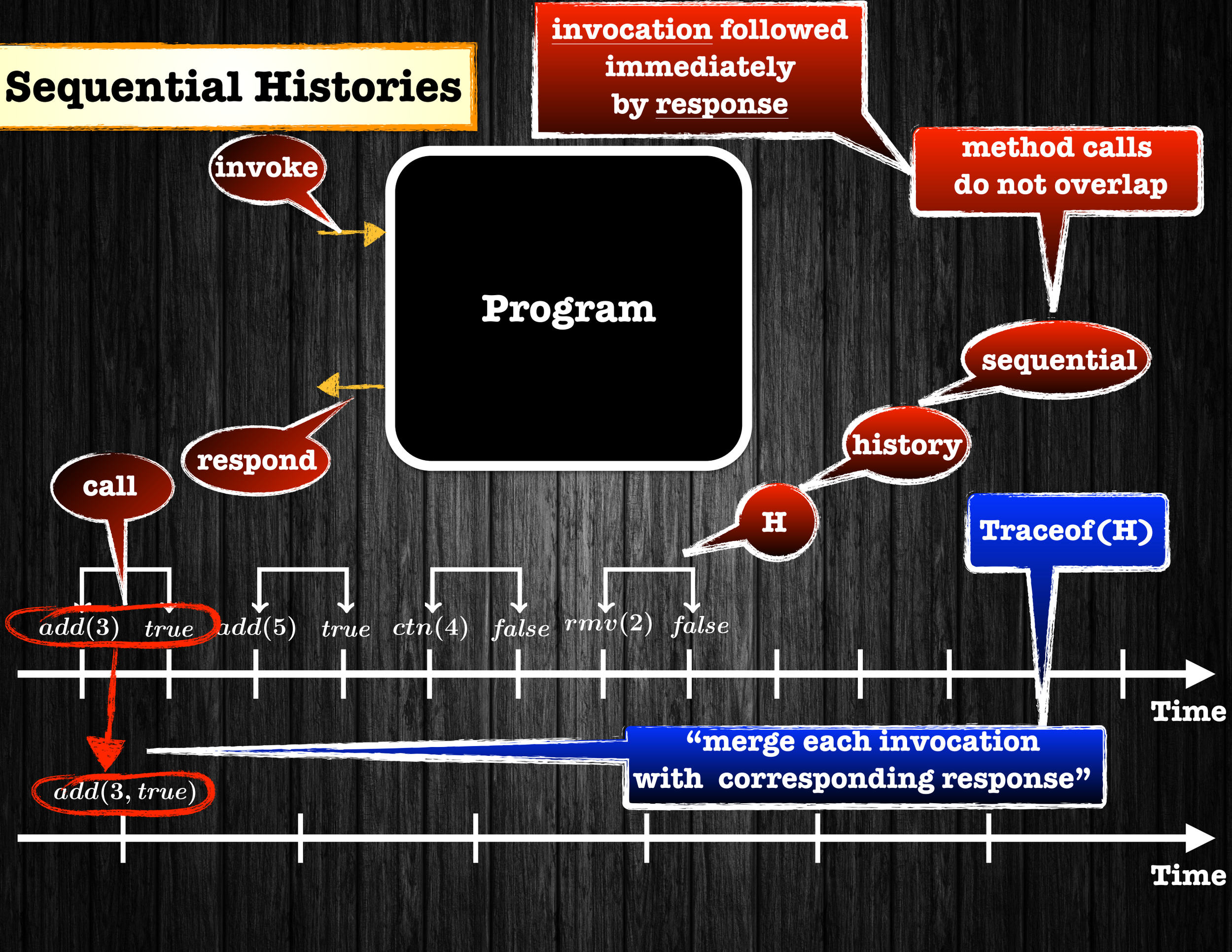
call

sequential

history

H

Traceof(H)



"merge each invocation with corresponding response"

Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

Program

respond

call

sequential

history

H

Traceof(H)

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

Time

"merge each invocation with corresponding response"

$add(3, true)$

operation

Time

Sequential Histories

invocation followed immediately by response

method calls do not overlap

invoke

"a call collapses to an operation"

Program

sequential

history

H

Traceof(H)

call

respond

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

Time

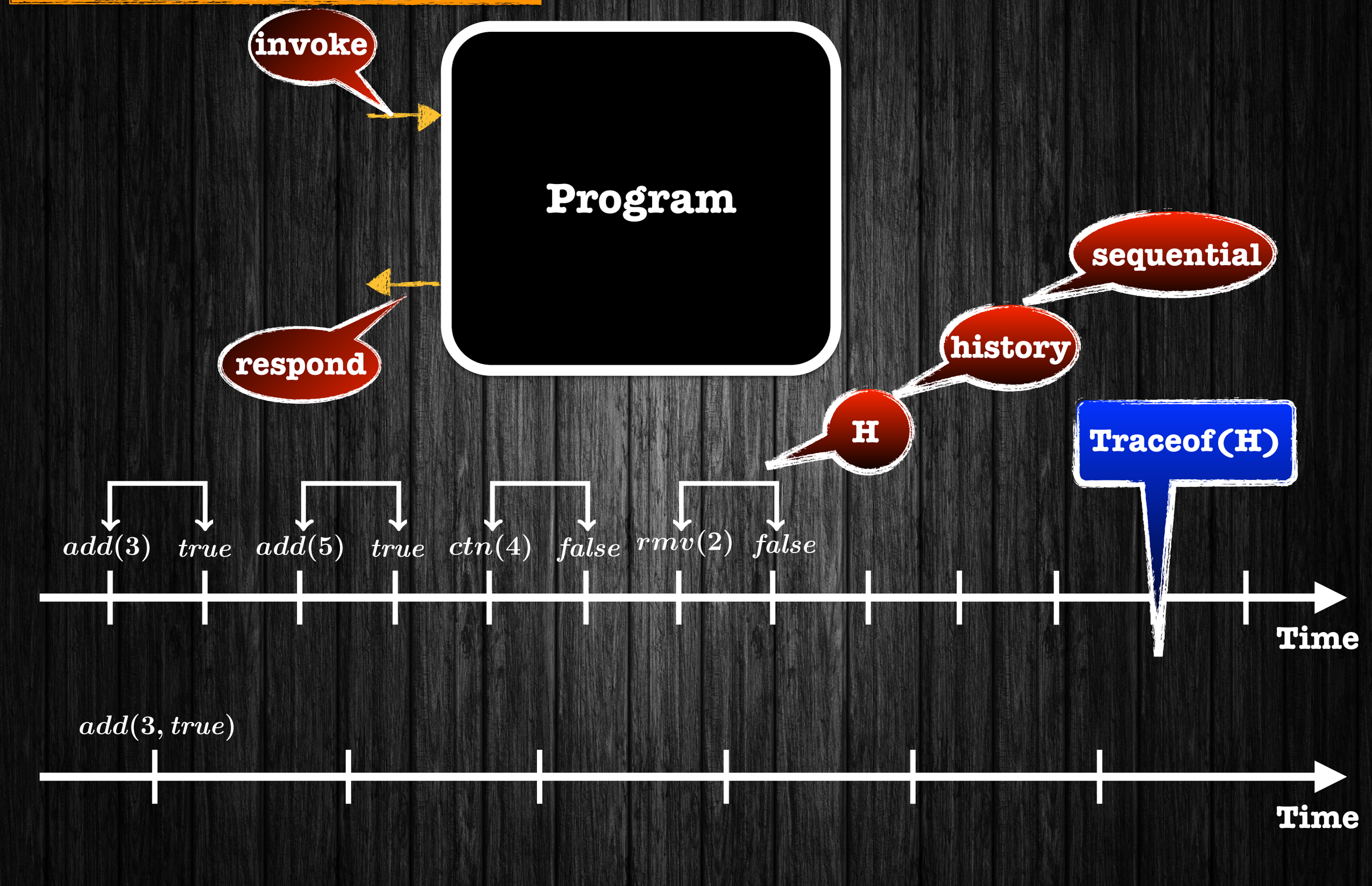
"merge each invocation with corresponding response"

$add(3, true)$

operation

Time

Sequential Histories



Sequential Histories

invoke

Program

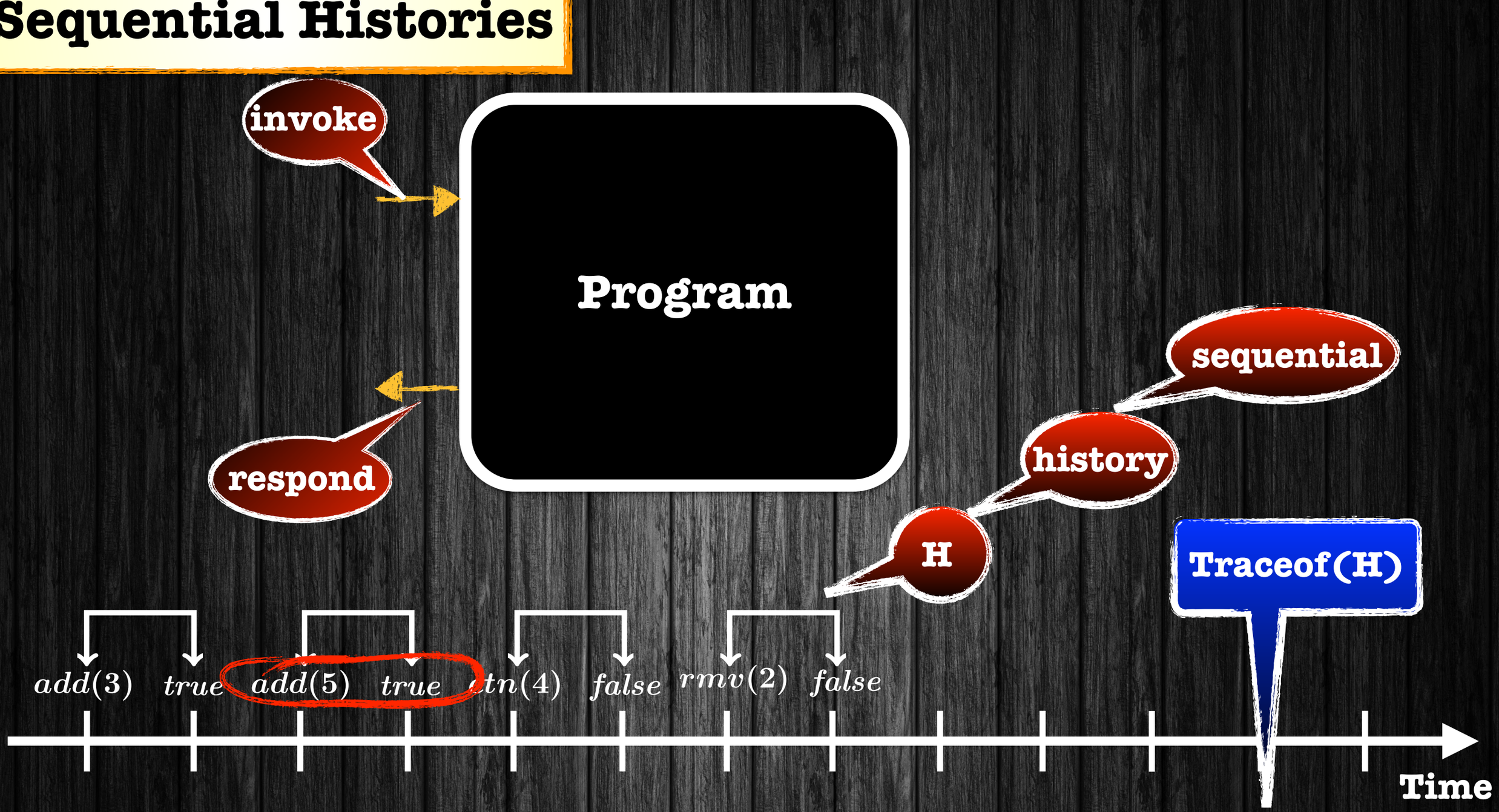
respond

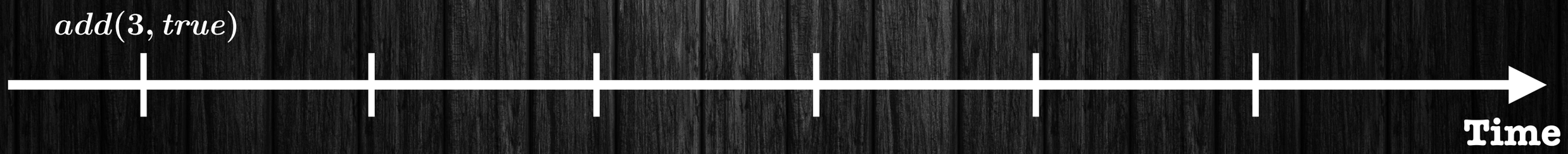
sequential

history

H

Traceof(H)





Sequential Histories

invoke

Program

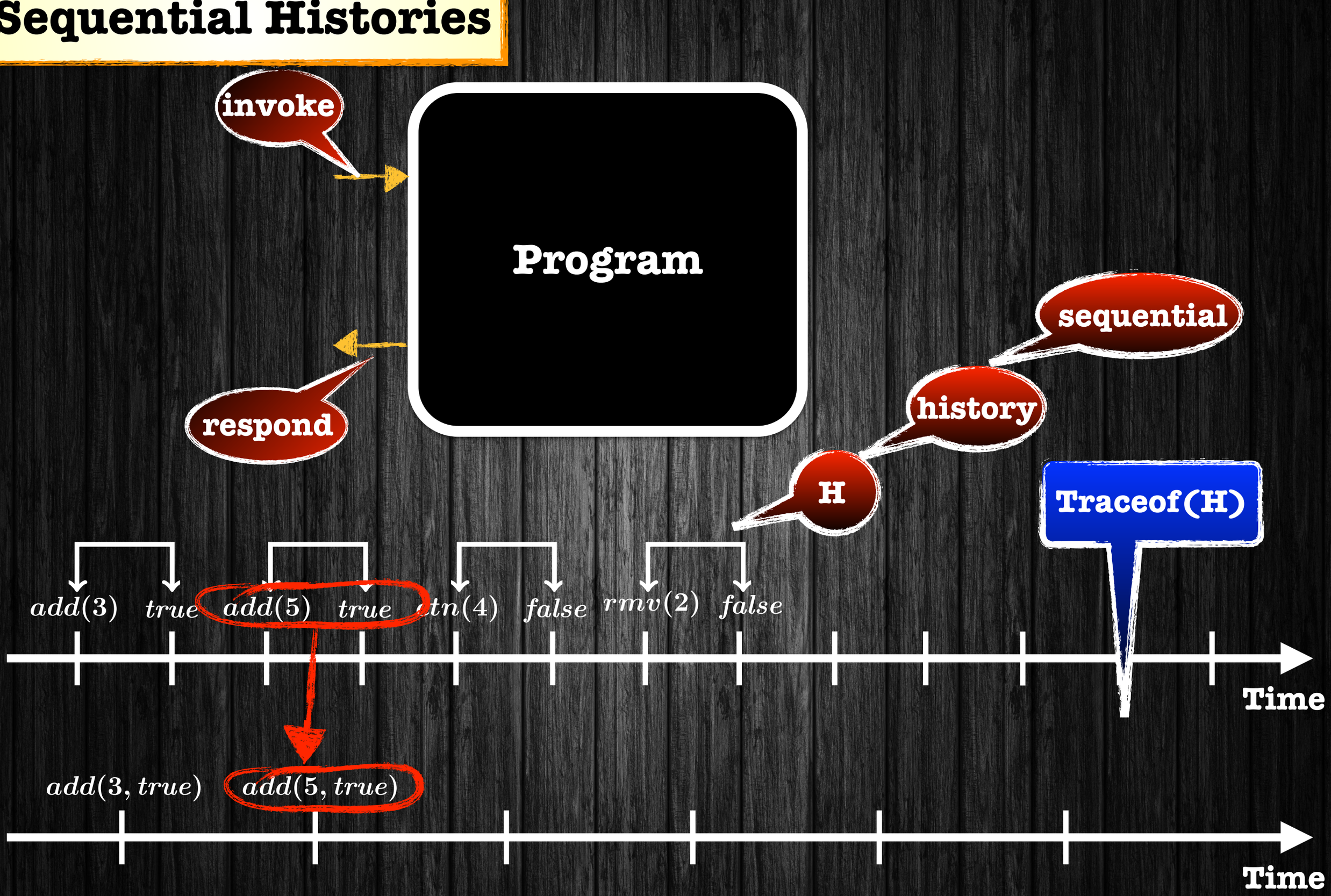
respond

sequential

history

H

Traceof(H)



Sequential Histories

invoke

respond

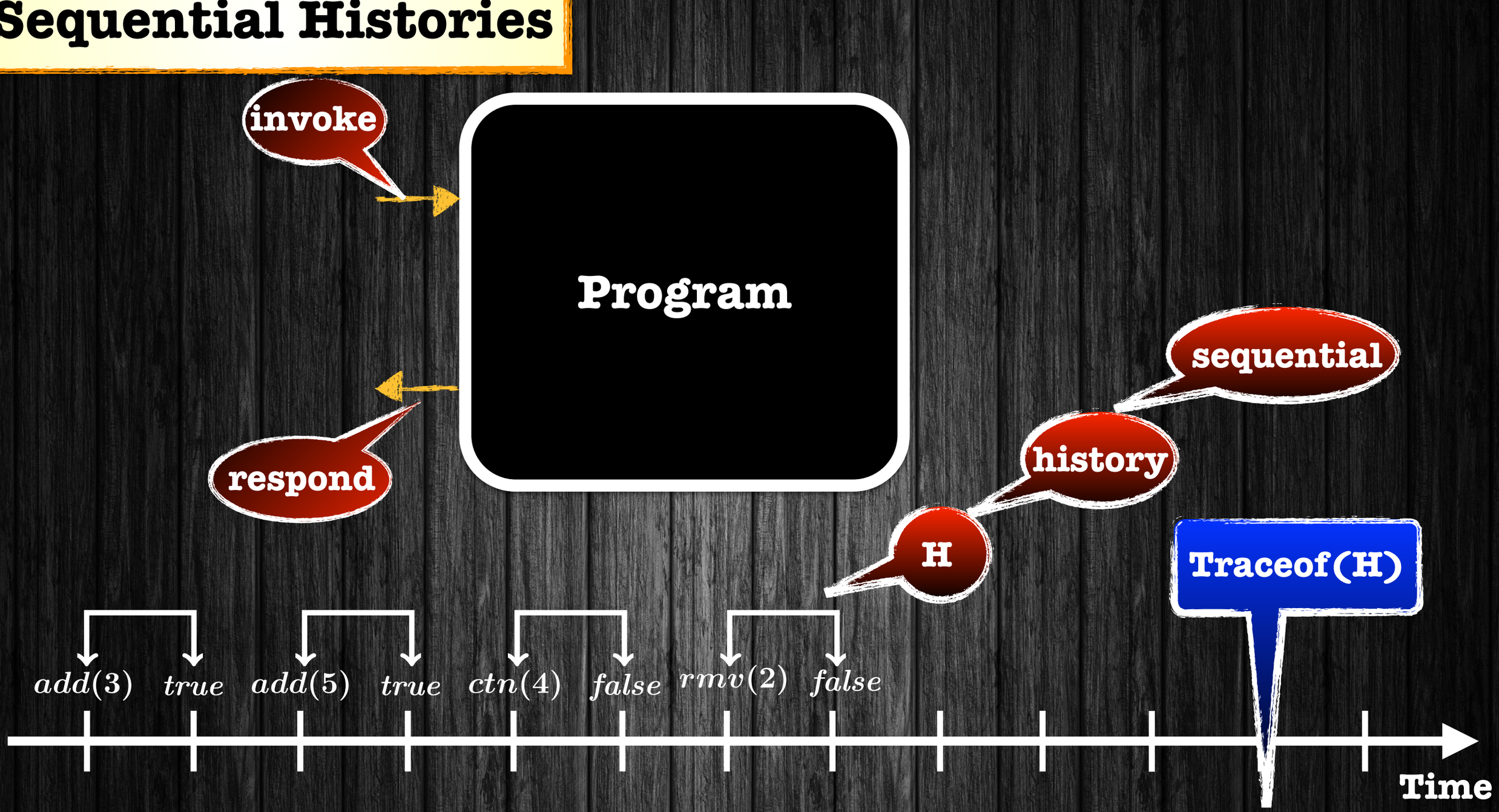
Program

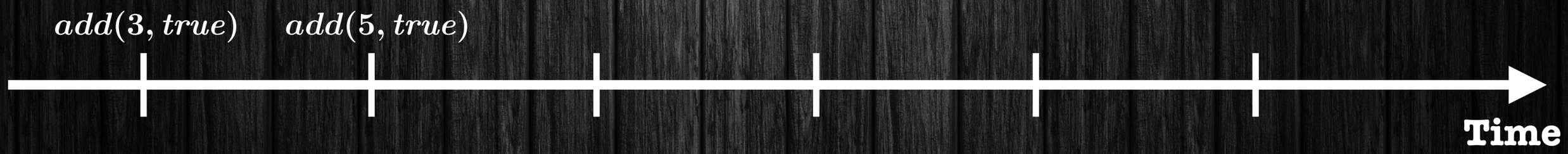
sequential

history

H

Traceof(H)





Sequential Histories

invoke

Program

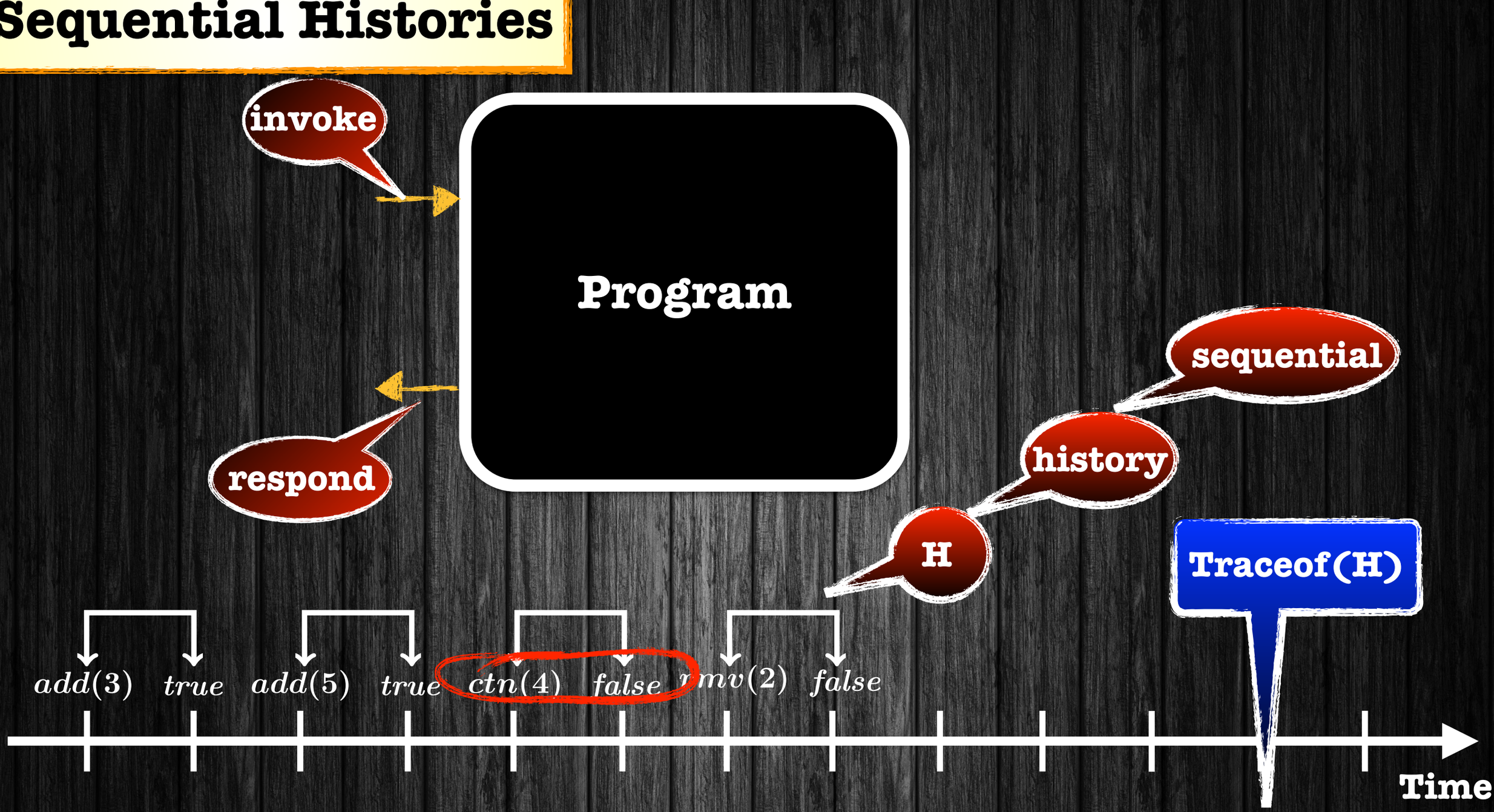
respond

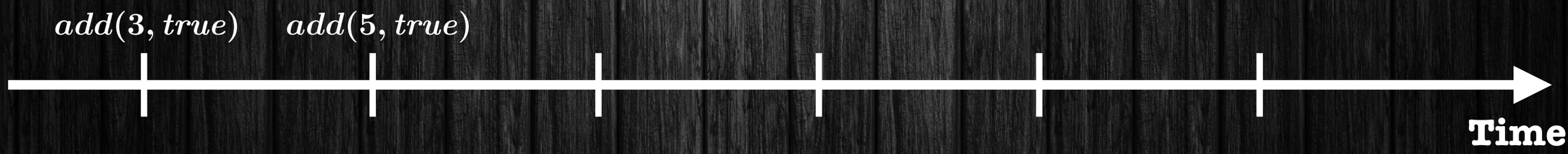
sequential

history

H

Traceof(H)





Sequential Histories

invoke

respond

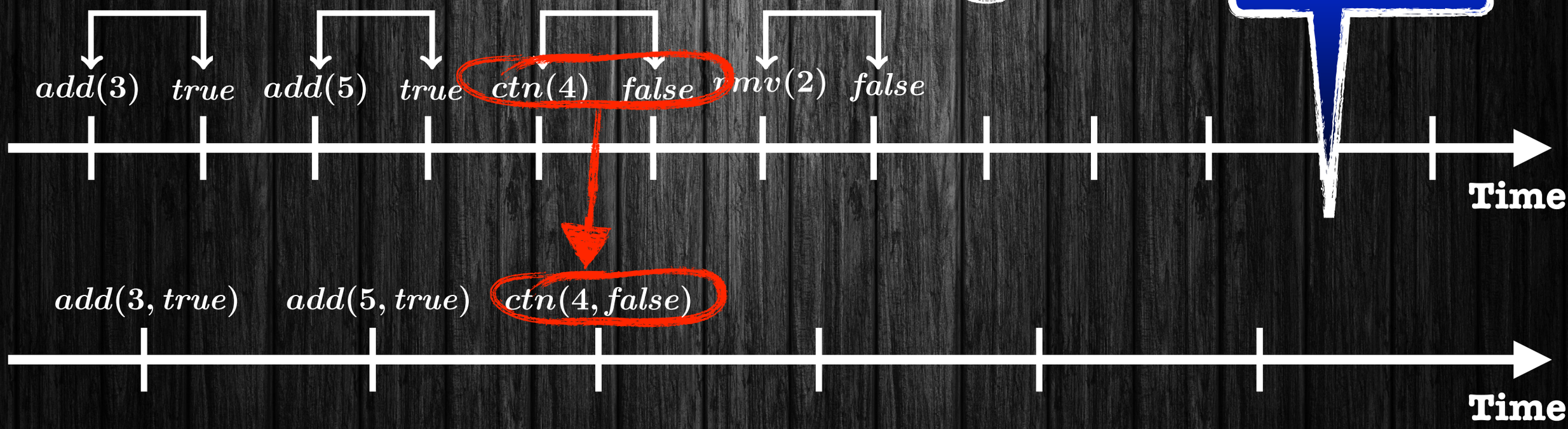
Program

sequential

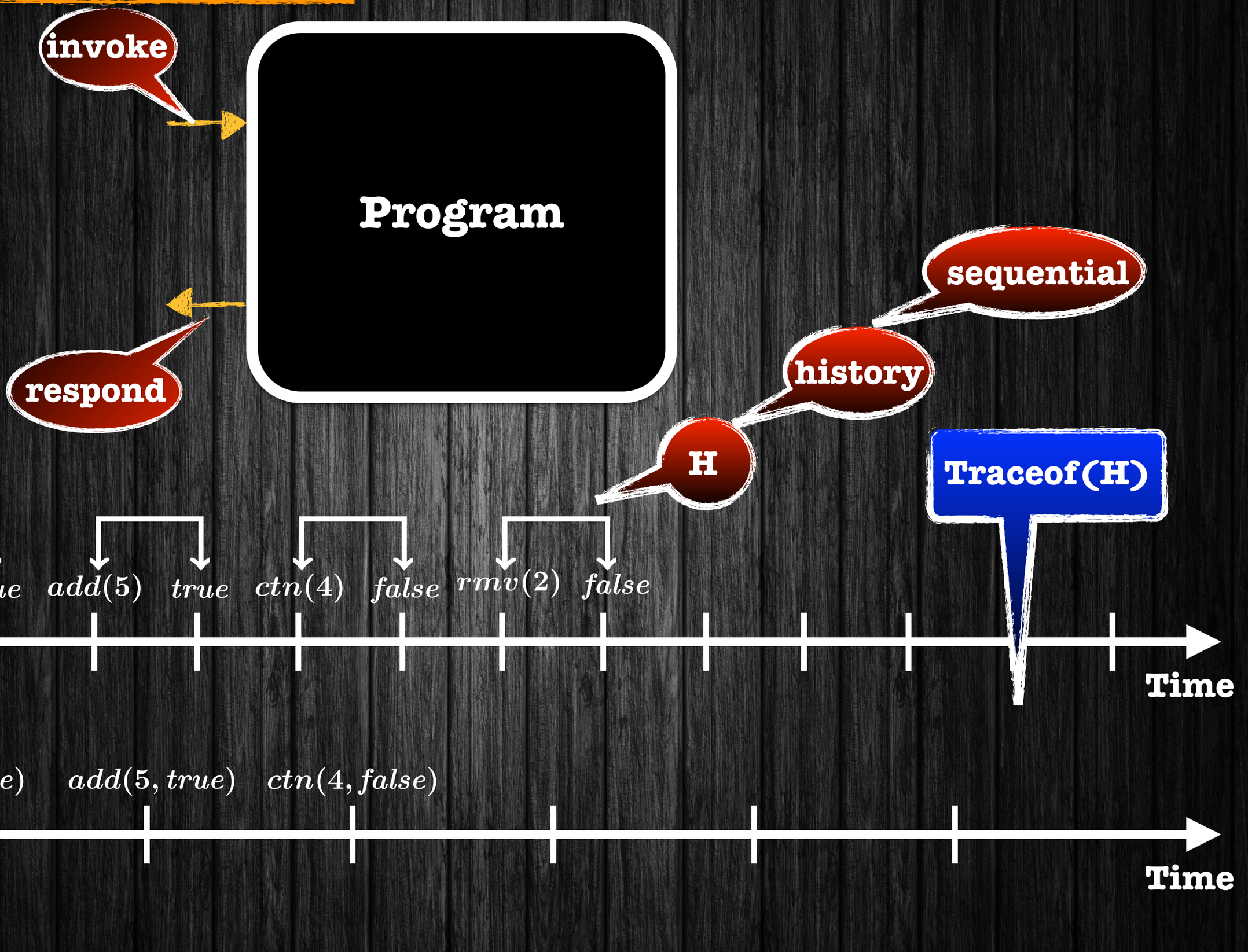
history

H

Traceof(H)



Sequential Histories



Sequential Histories

invoke

Program

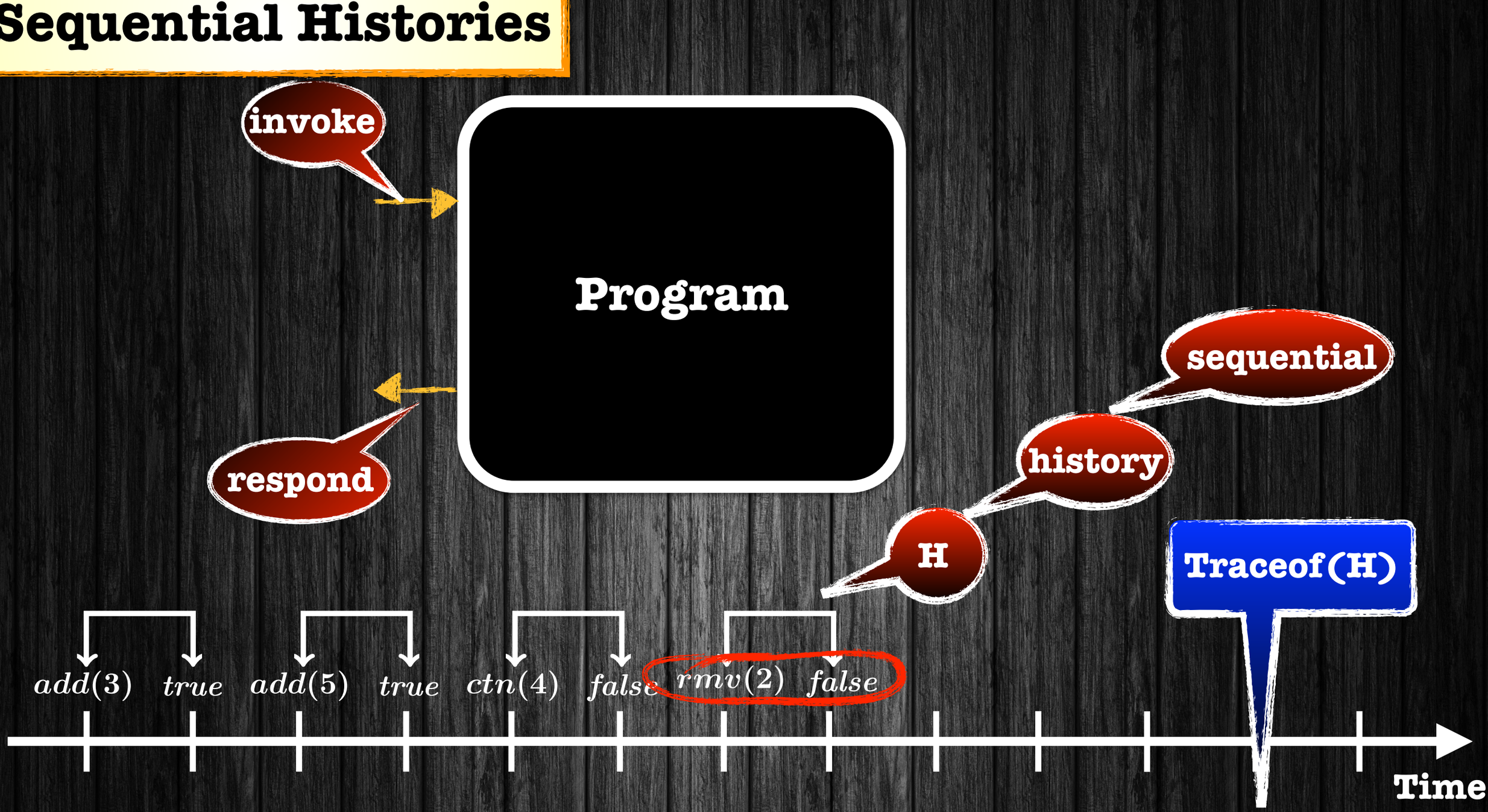
respond

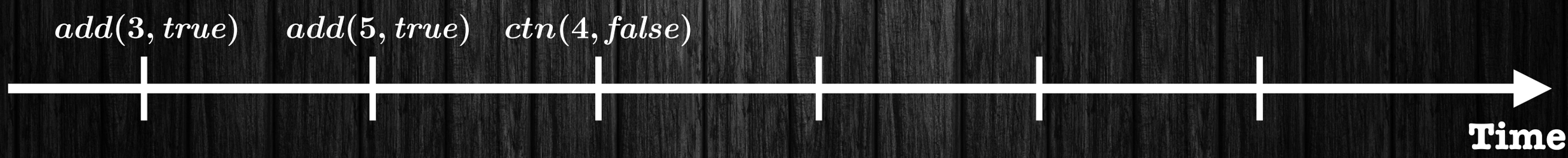
sequential

history

H

Traceof(H)





Sequential Histories

invoke

Program

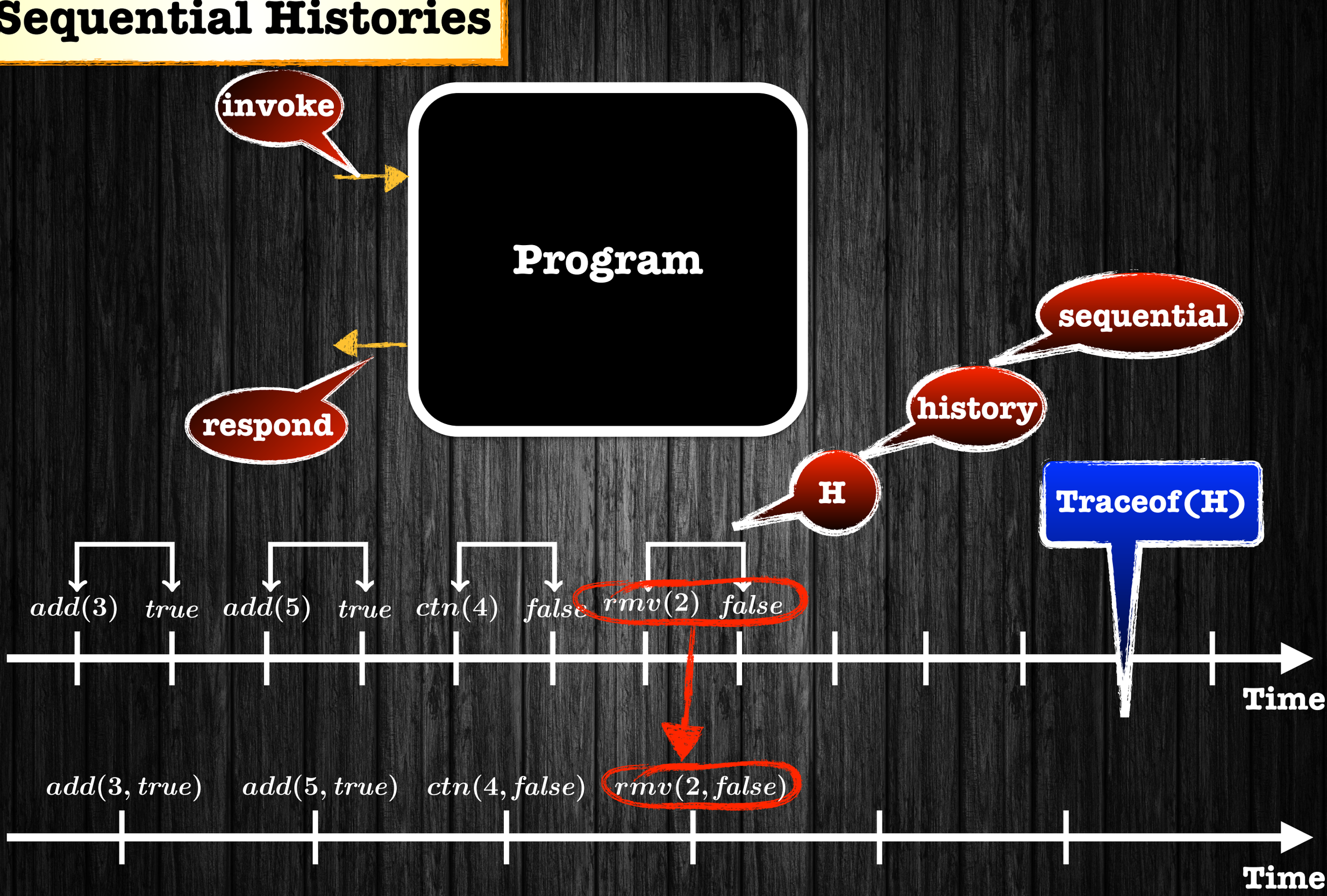
respond

sequential

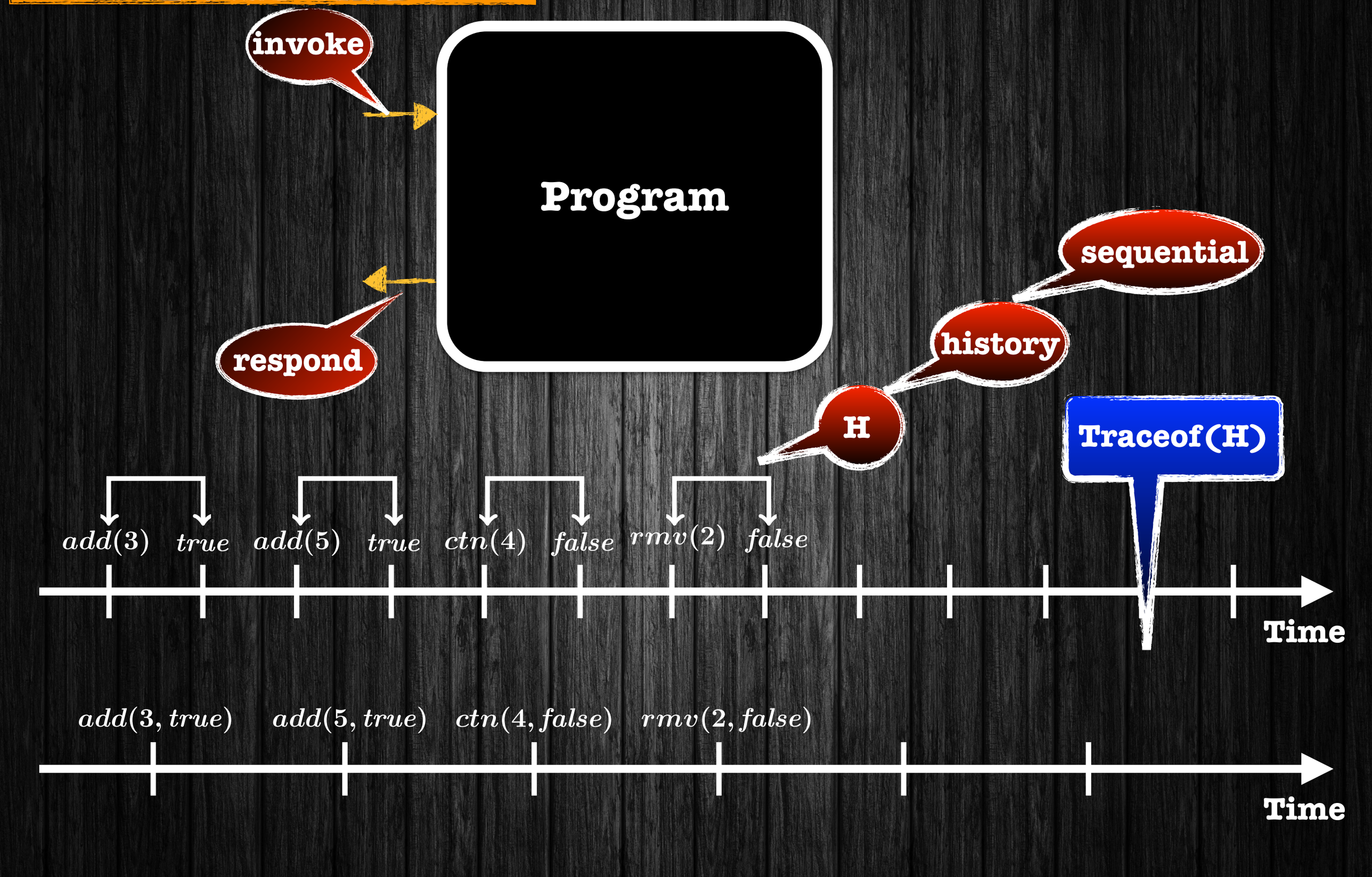
history

H

Traceof(H)



Sequential Histories



Sequential Histories

invoke

respond

Program

sequential

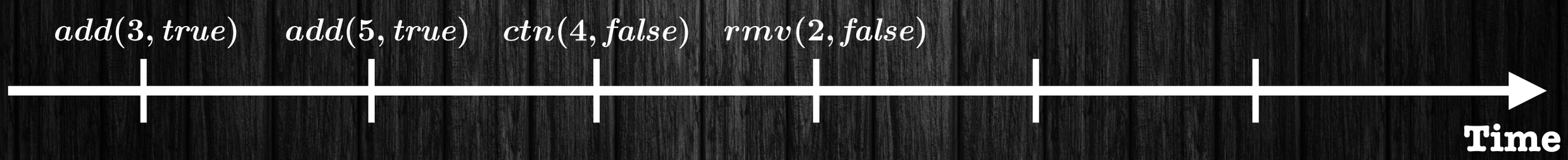
history

H

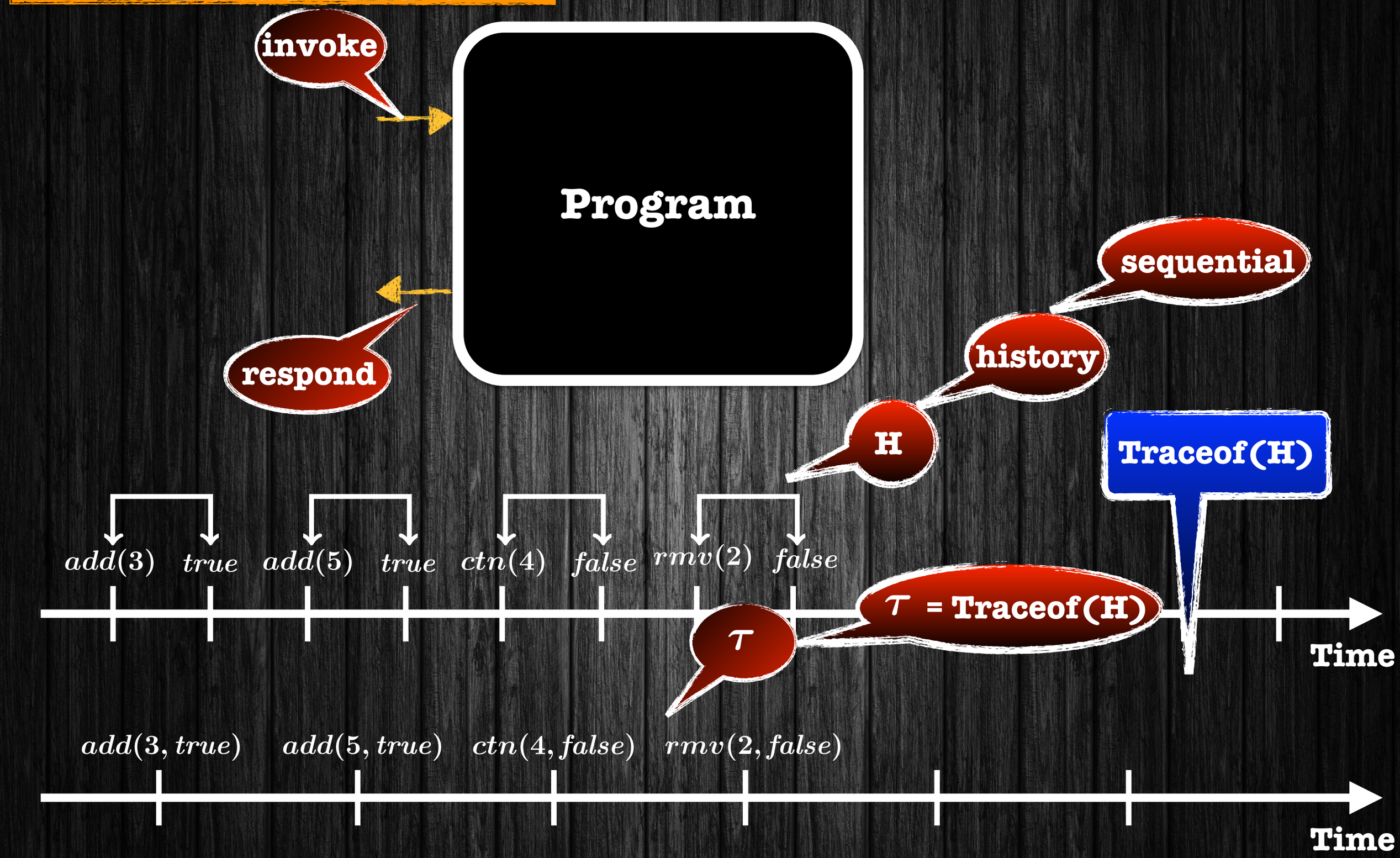
Traceof(H)



τ



Sequential Histories



Sequential Histories

invoke

respond

Program

sequential

history

H

Traceof(H)

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

τ

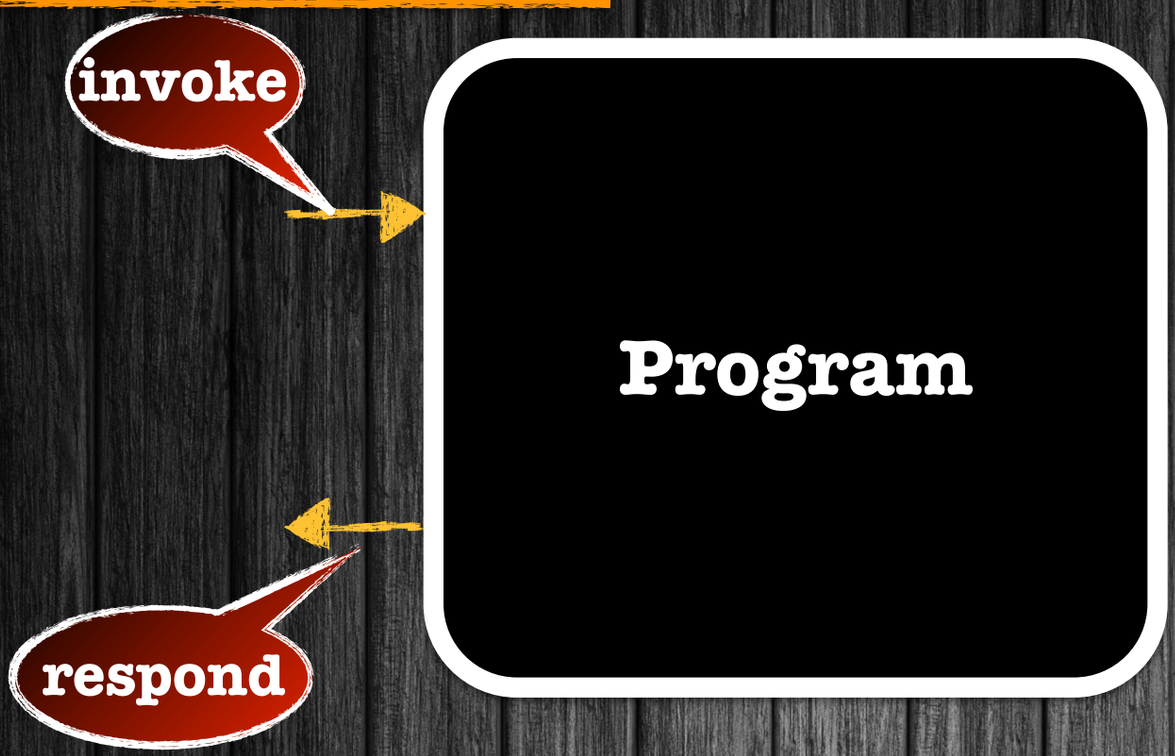
τ = Traceof(H)

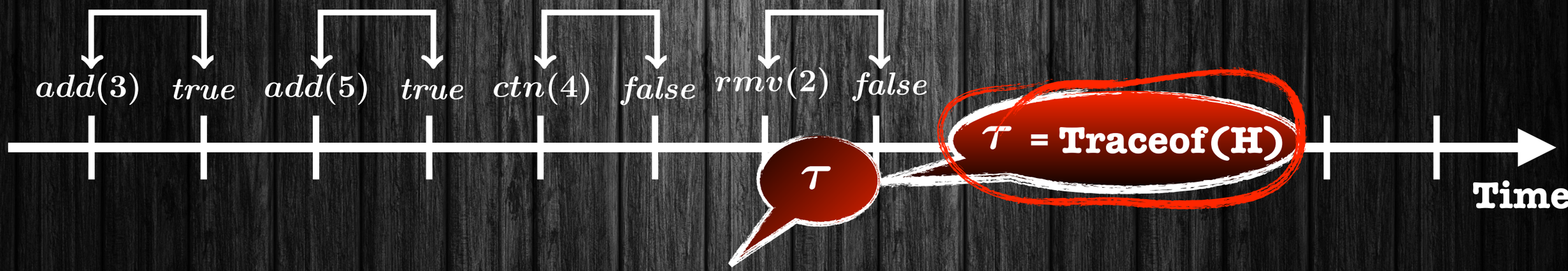
Time

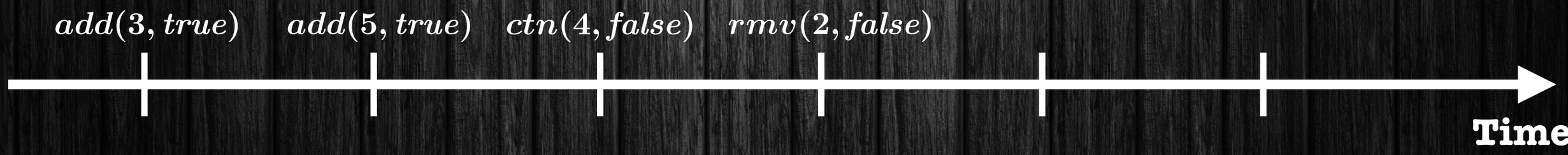
$add(3, true)$ $add(5, true)$ $ctn(4, false)$ $rmv(2, false)$

Time

Sequential Histories







Sequential Histories

A: abstract data type

invoke

Program

respond

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

τ = Traceof(H)

τ

Time

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

Time

Sequential Histories

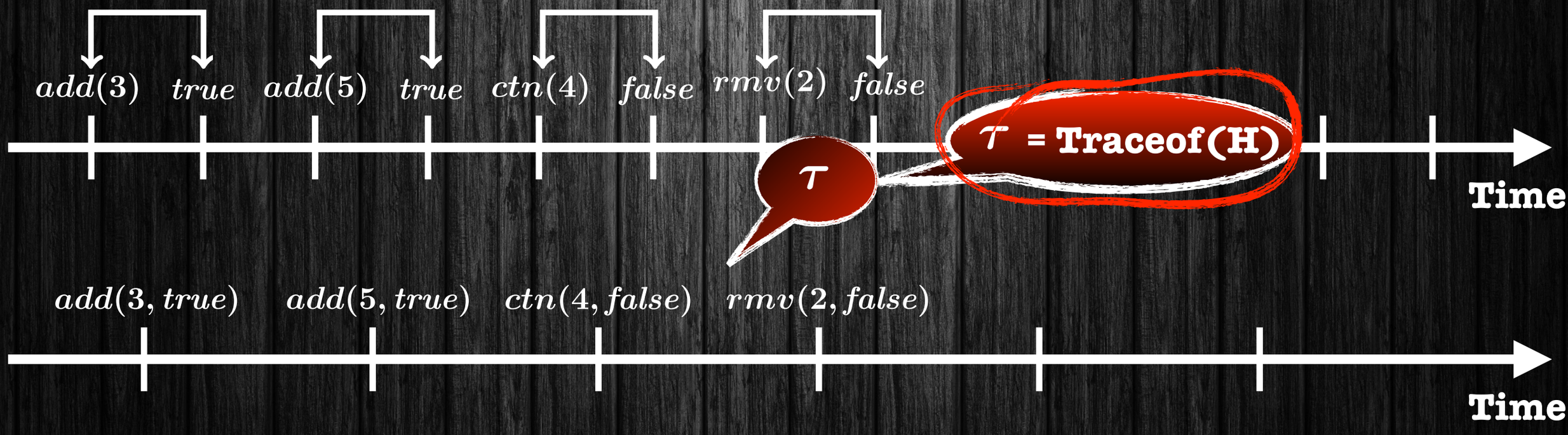
A: abstract data type

H ∈ Histories(A)

invoke

respond

Program

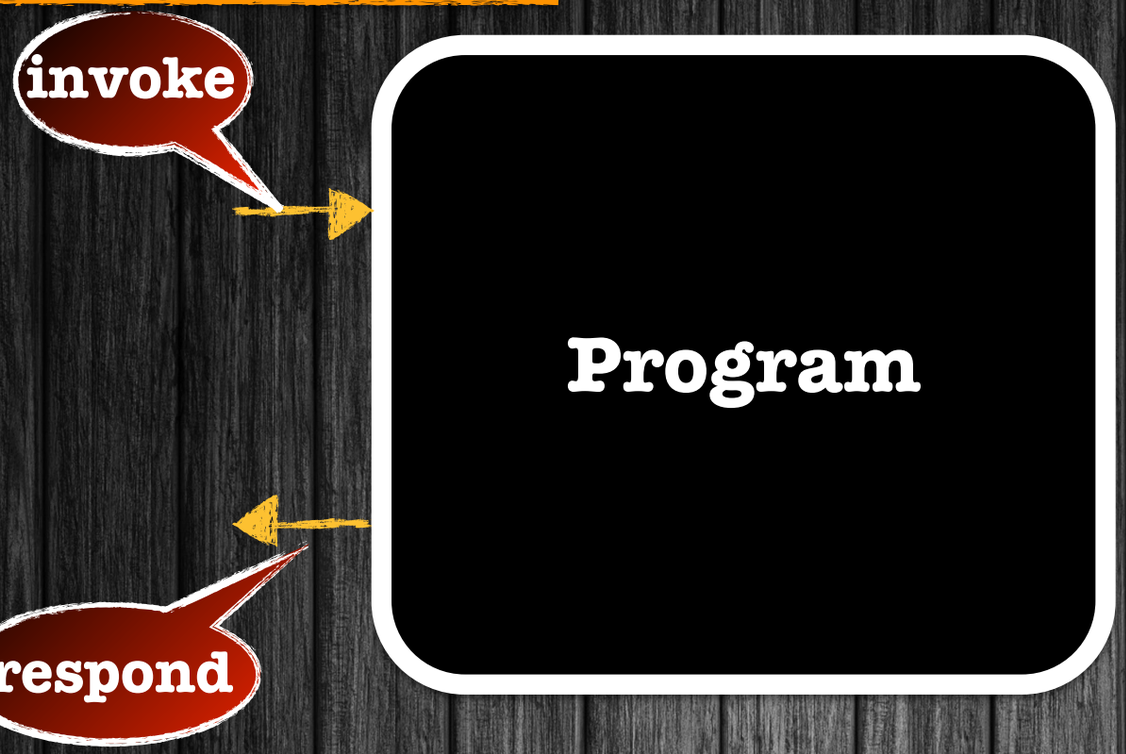


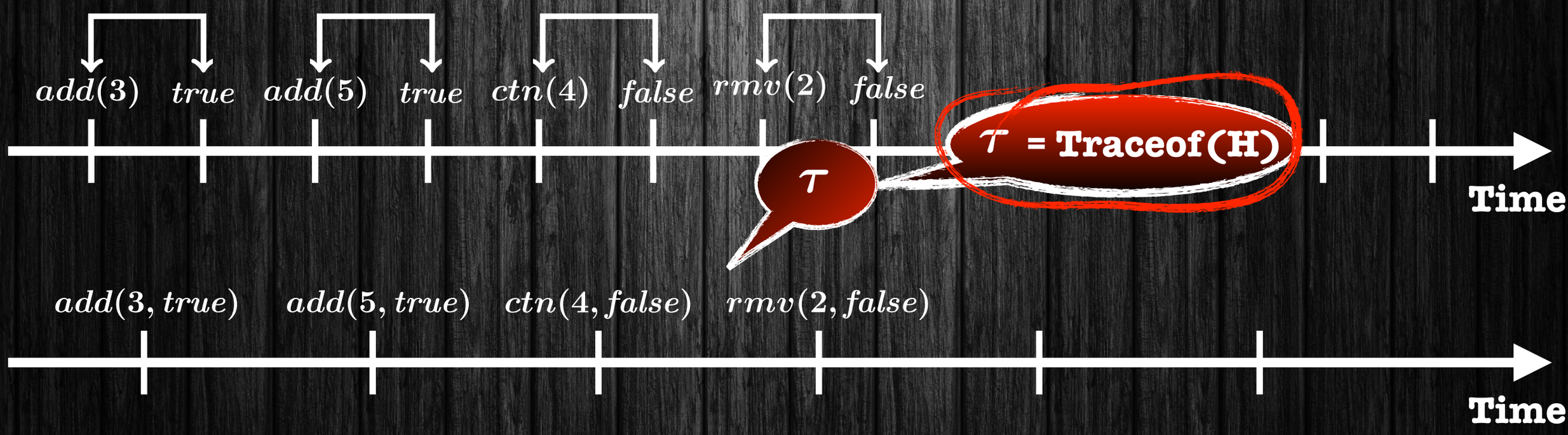
Sequential Histories

A: abstract data type

H ∈ Histories(A)

Traceof(H) ∈ Traces(A)





Sequential Histories

A: abstract data type

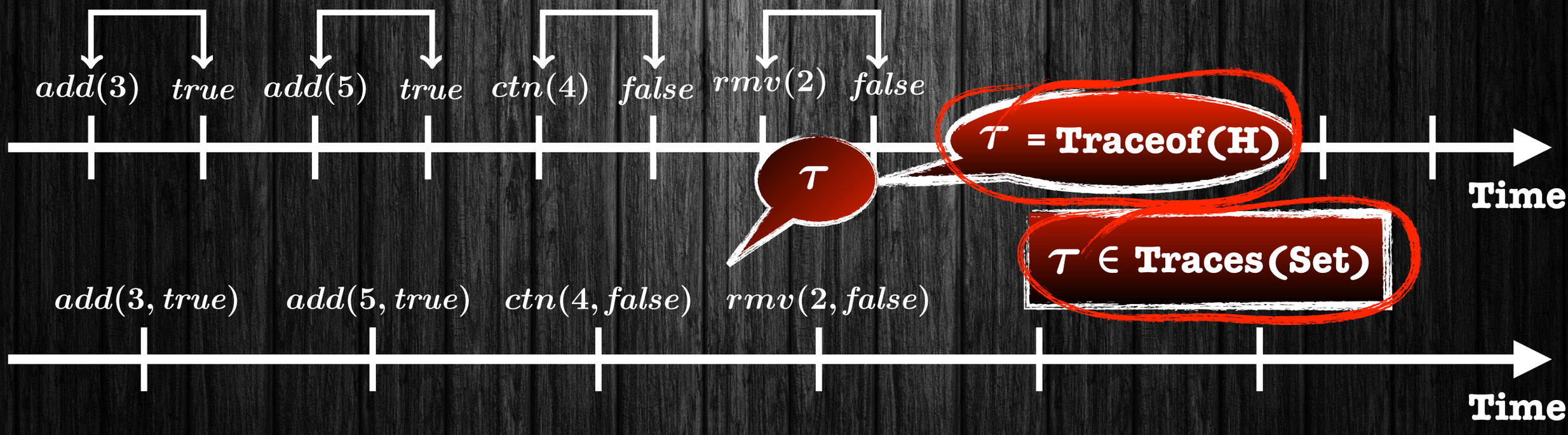
H ∈ Histories(A)

Traceof(H) ∈ Traces(A)

invoke

Program

respond



Sequential Histories

invoke

respond

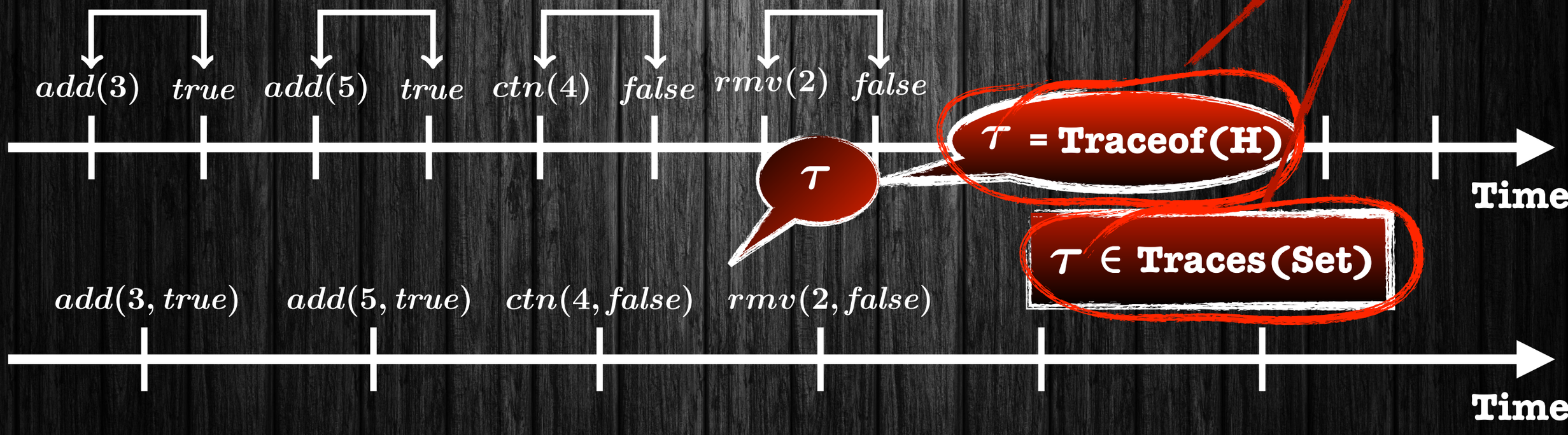
Program

A: abstract data type

H ∈ Histories(A)

Traceof(H) ∈ Traces(A)

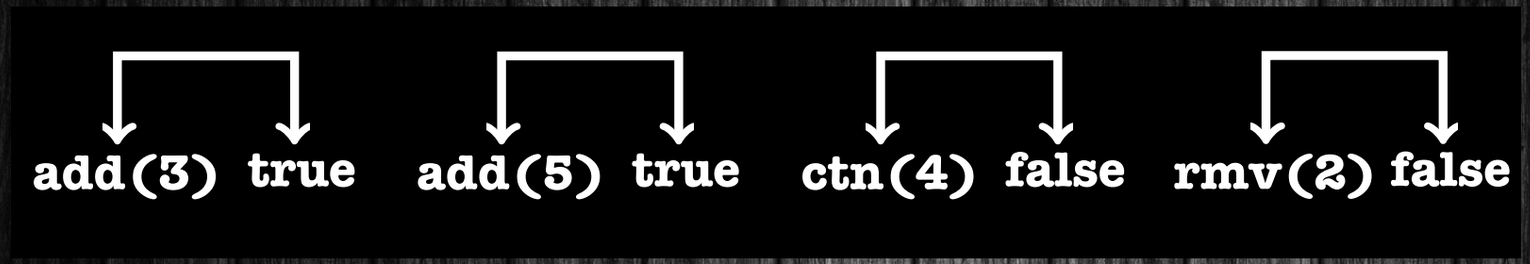
H ∈ Histories(Set)



Sequential Histories

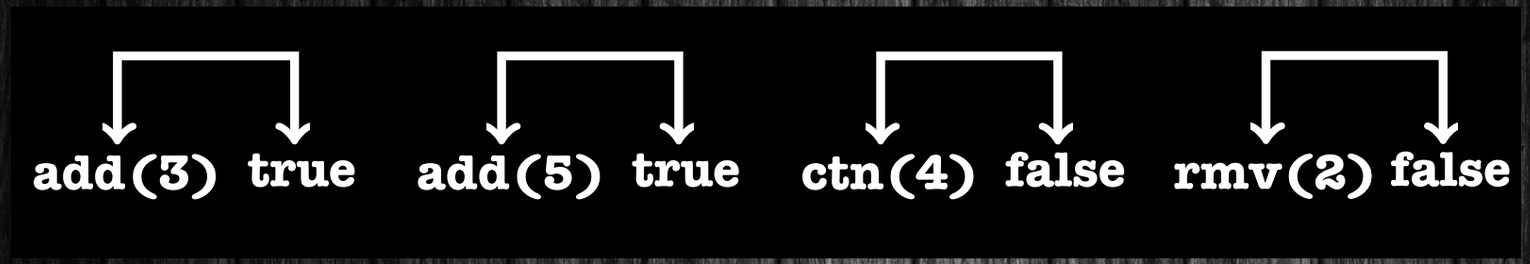
Sequential Histories

H_1



Sequential Histories

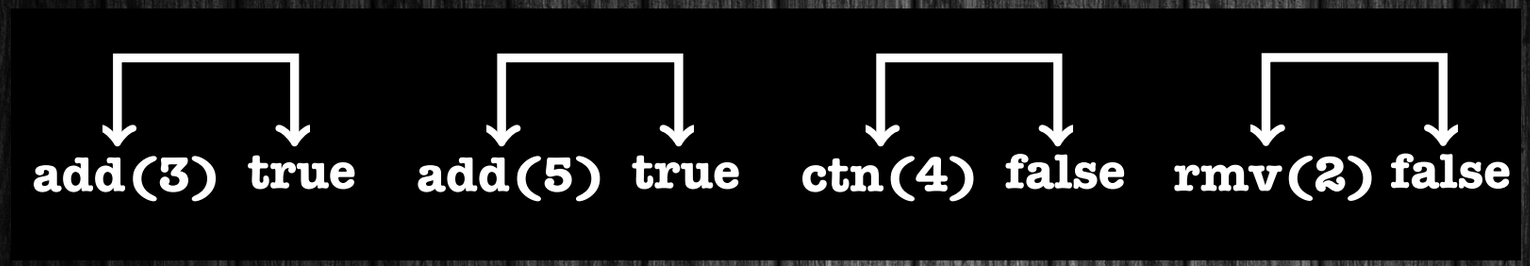
H_1



$H_1 \in$ Histories (Set)

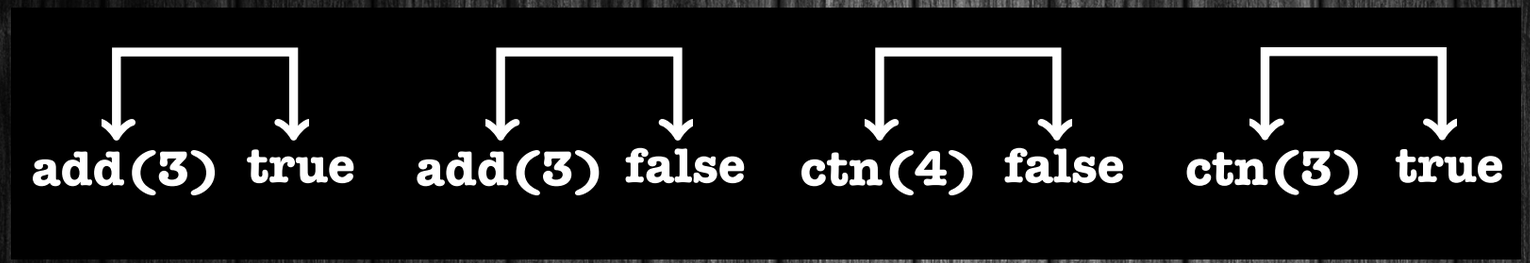
Sequential Histories

H_1



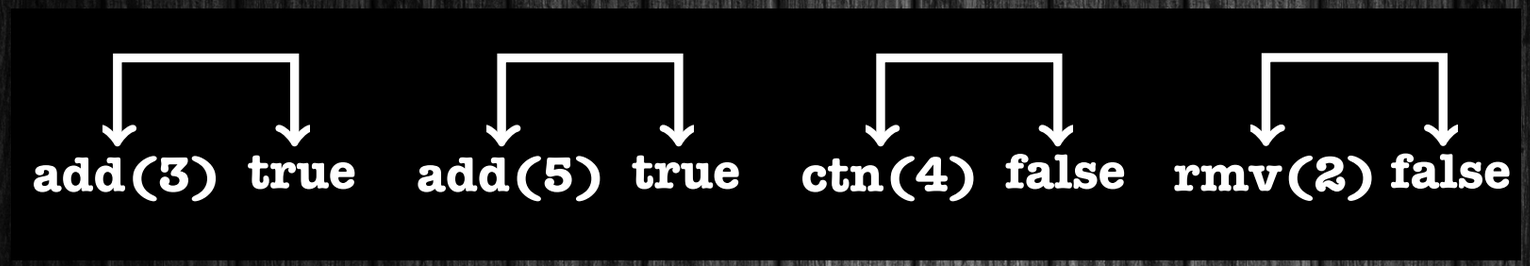
$H_1 \in$ Histories (Set)

H_2



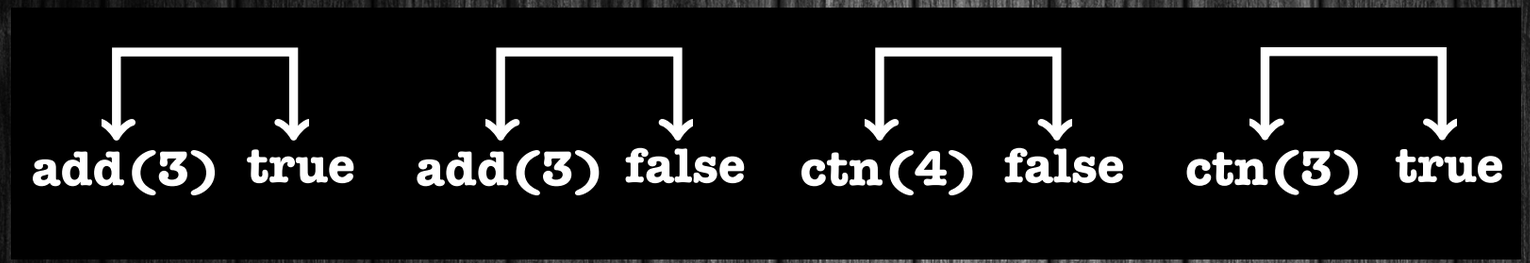
Sequential Histories

H_1



$H_1 \in$ Histories (Set)

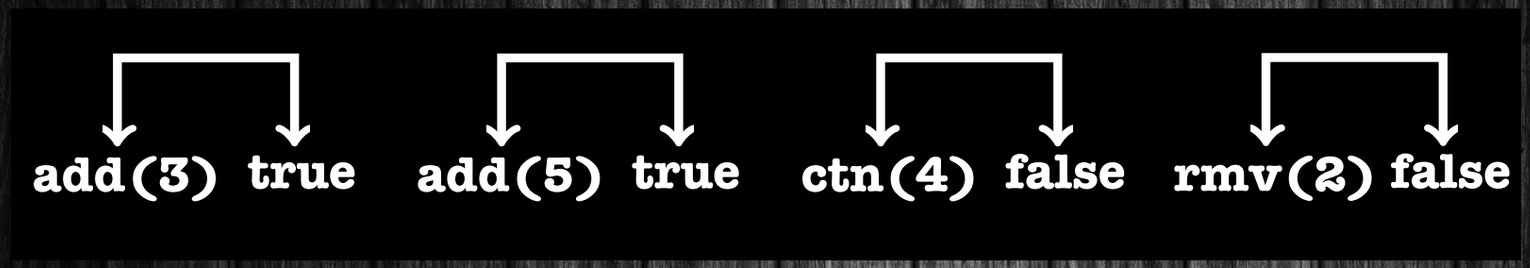
H_2



$H_2 \in$ Histories (Set)

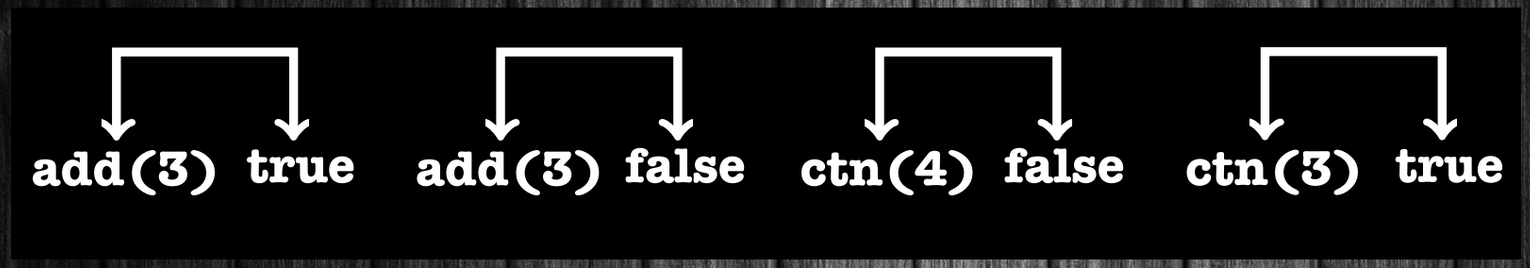
Sequential Histories

H_1



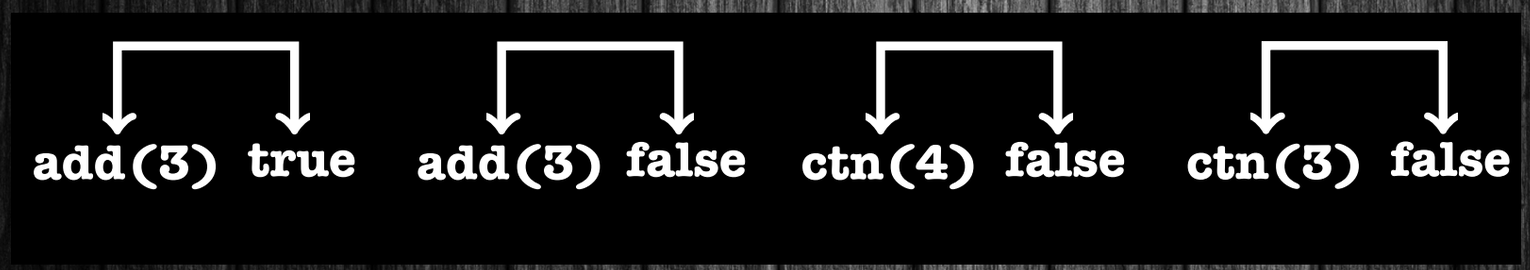
$H_1 \in$ Histories (Set)

H_2



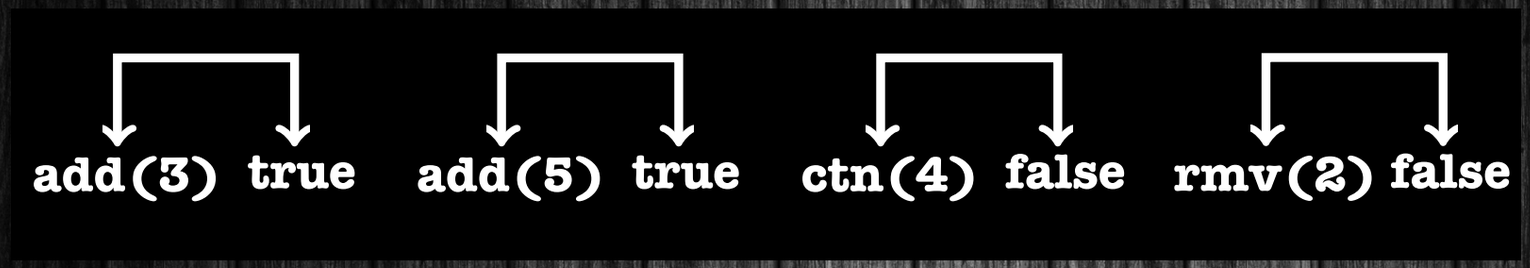
$H_2 \in$ Histories (Set)

H_3



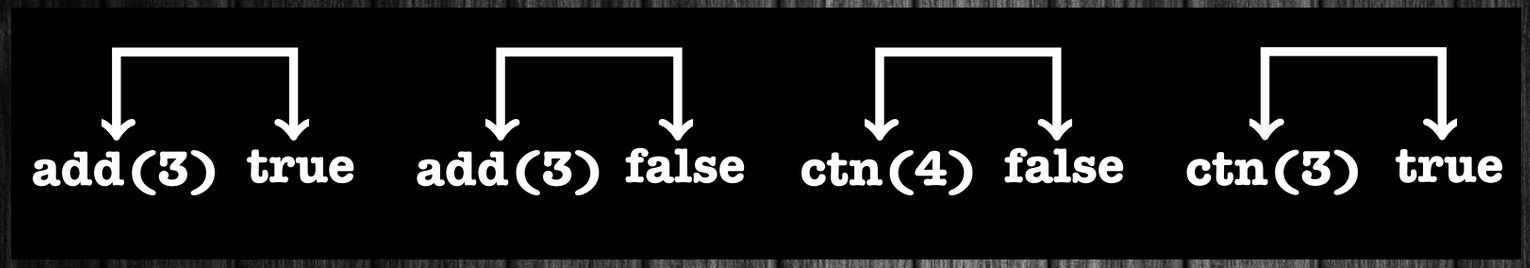
Sequential Histories

H_1



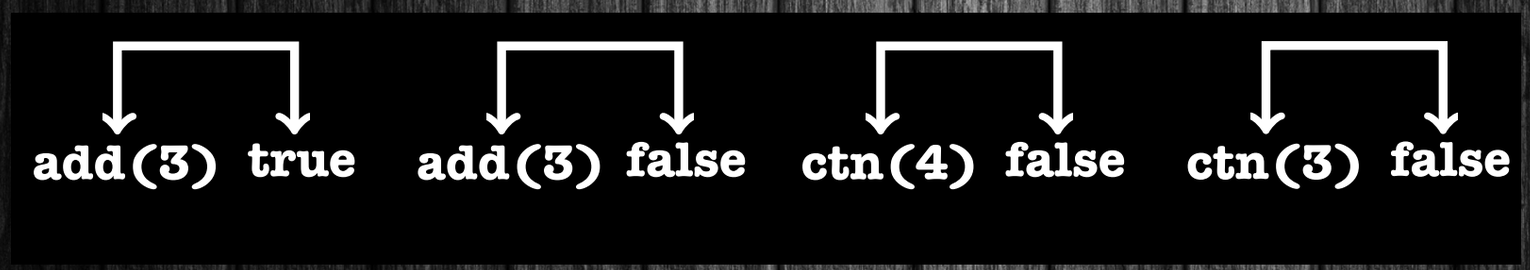
$H_1 \in$ Histories (Set)

H_2



$H_2 \in$ Histories (Set)

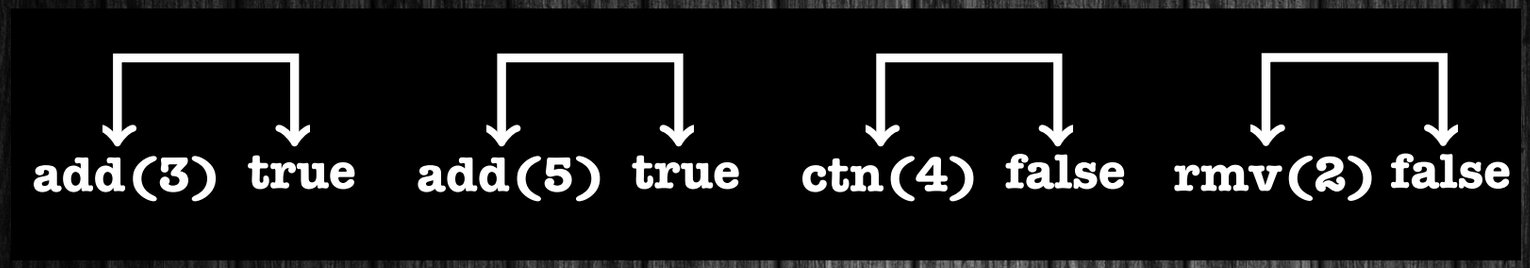
H_3



$H_3 \notin$ Histories (Set)

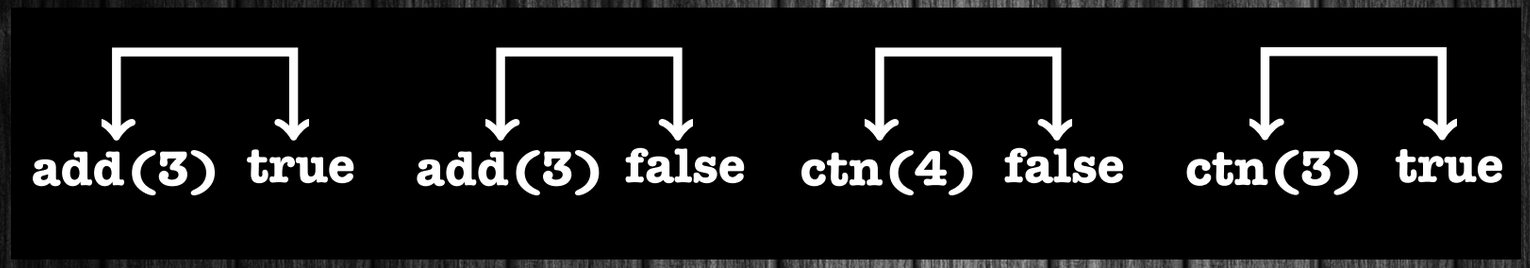
Sequential Histories

H_1



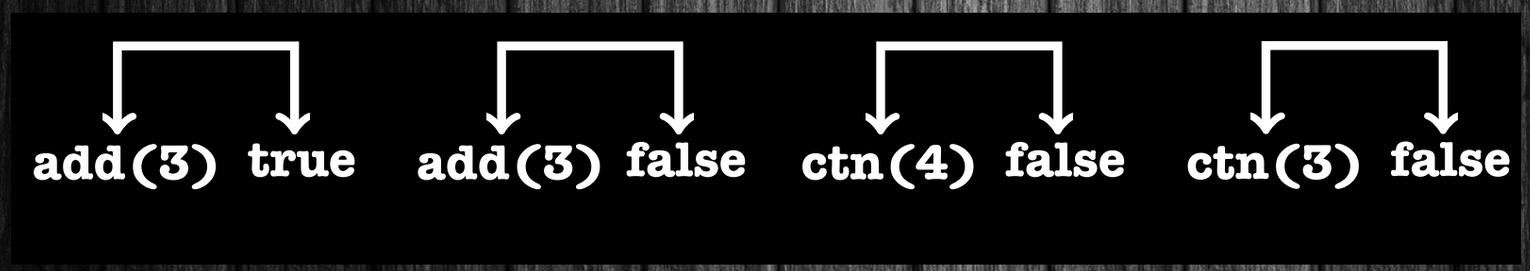
$H_1 \in$ Histories (Set)

H_2



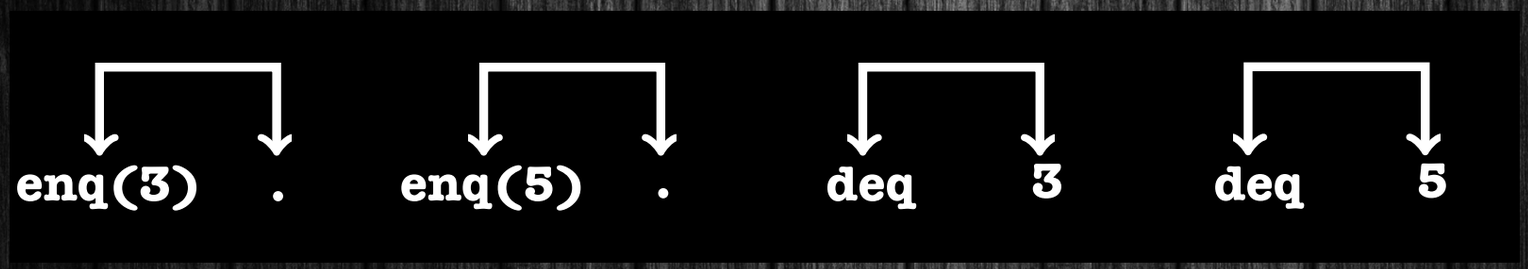
$H_2 \in$ Histories (Set)

H_3



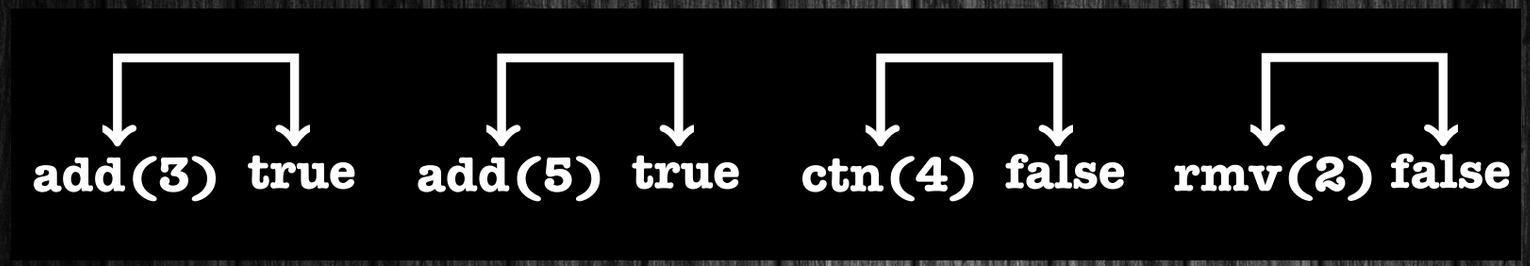
$H_3 \notin$ Histories (Set)

H_4



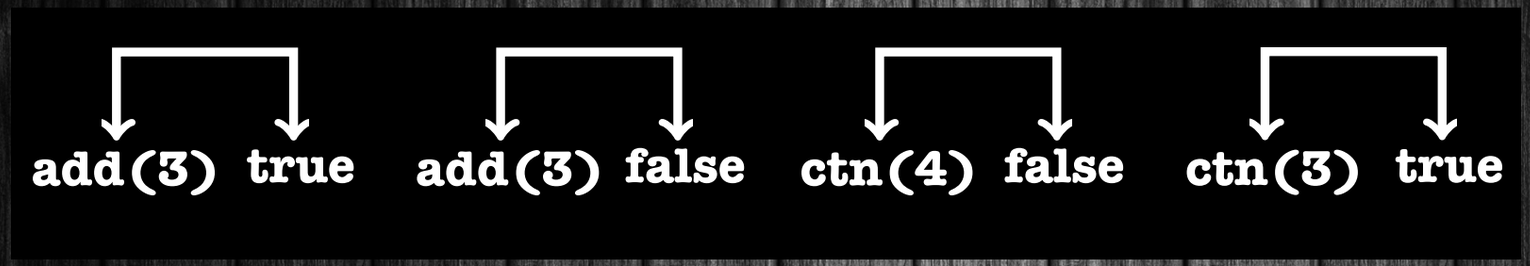
Sequential Histories

H_1



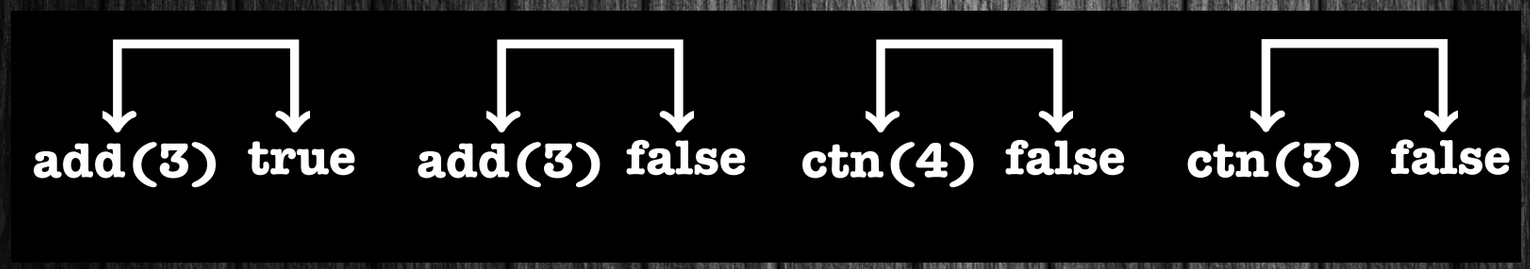
$H_1 \in$ Histories (Set)

H_2



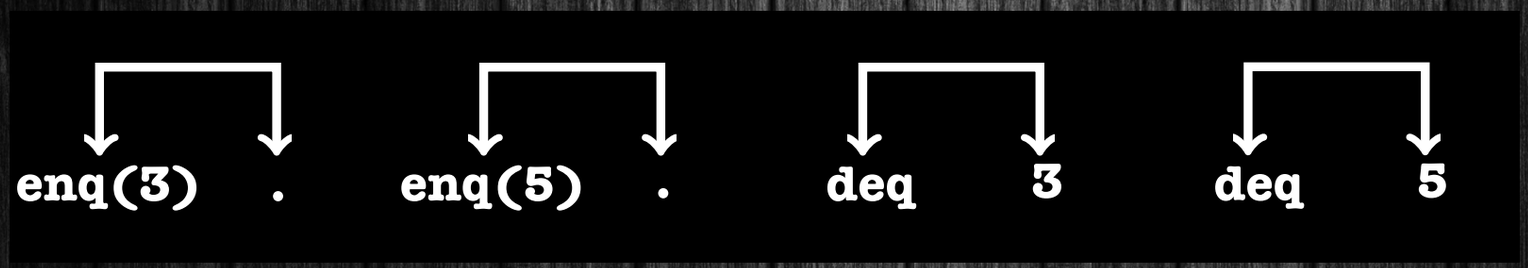
$H_2 \in$ Histories (Set)

H_3



$H_3 \notin$ Histories (Set)

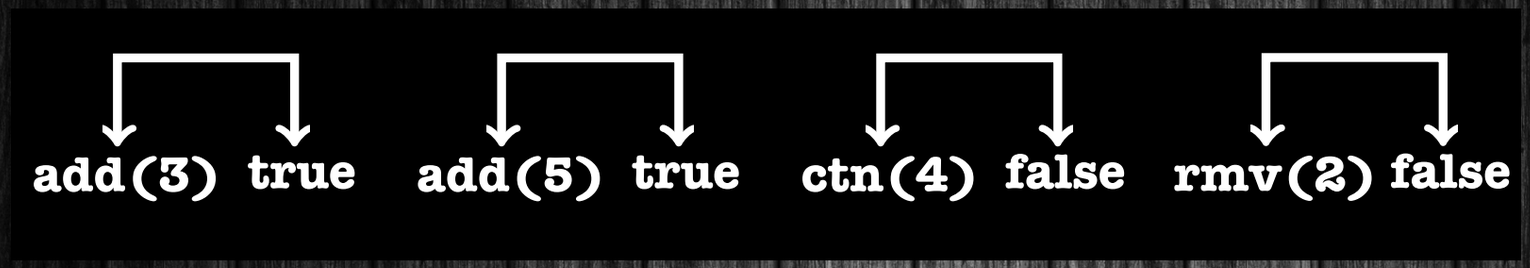
H_4



$H_4 \in$ Histories (Queue)

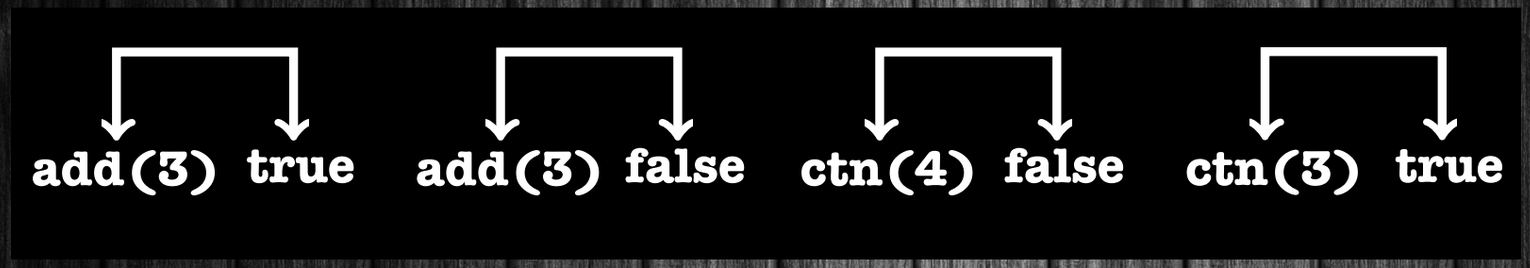
Sequential Histories

H_1



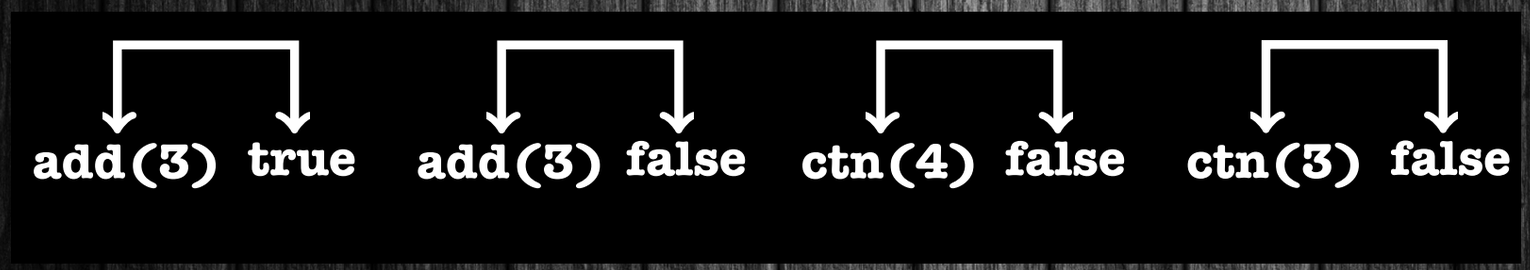
$H_1 \in$ Histories (Set)

H_2



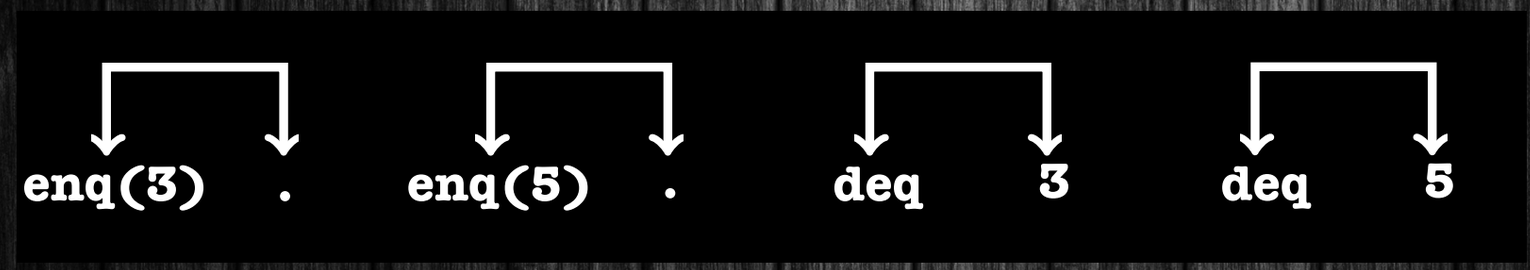
$H_2 \in$ Histories (Set)

H_3



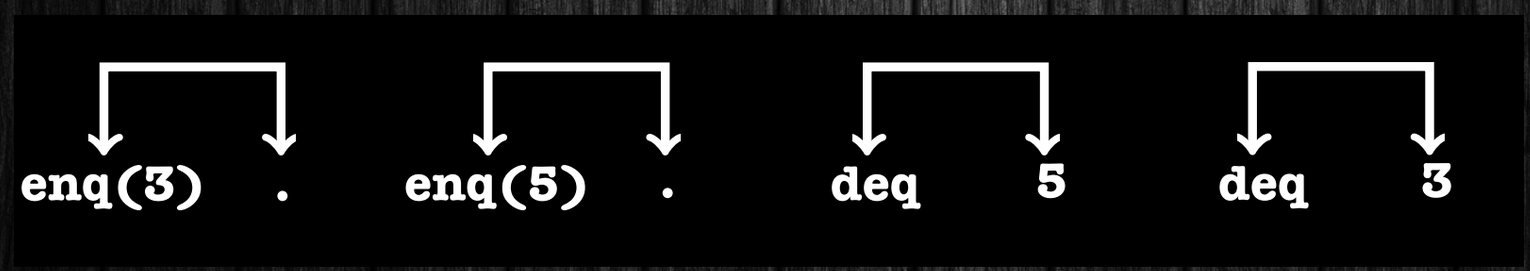
$H_3 \notin$ Histories (Set)

H_4



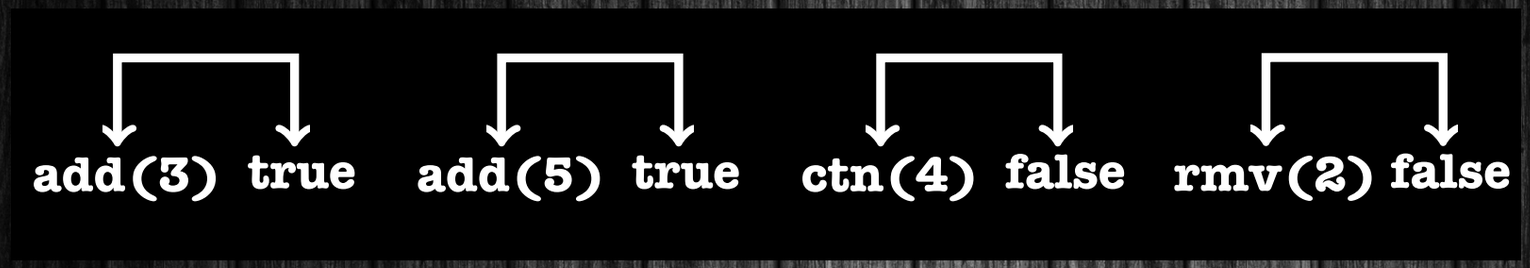
$H_4 \in$ Histories (Queue)

H_5



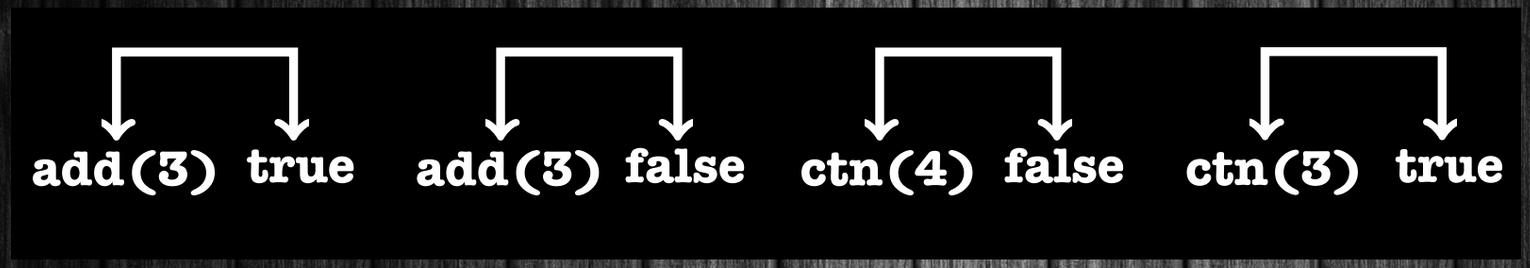
Sequential Histories

H_1



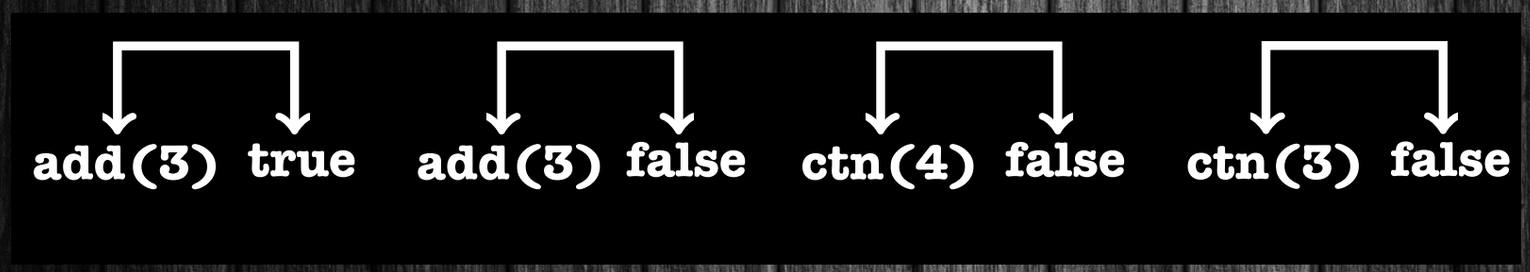
$H_1 \in$ Histories (Set)

H_2



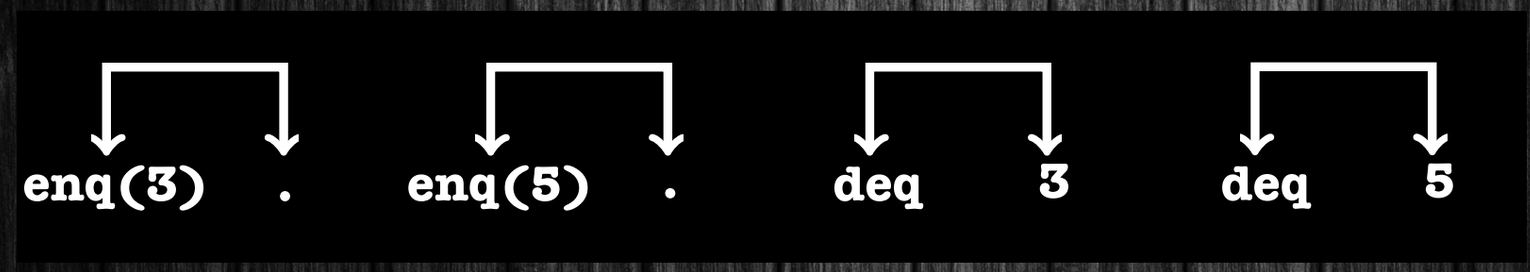
$H_2 \in$ Histories (Set)

H_3



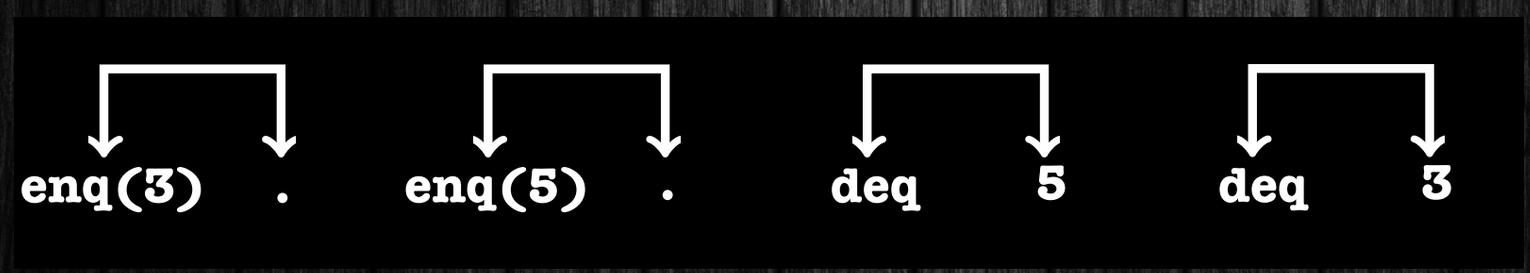
$H_3 \notin$ Histories (Set)

H_4



$H_4 \in$ Histories (Queue)

H_5



$H_5 \notin$ Histories (Queue)

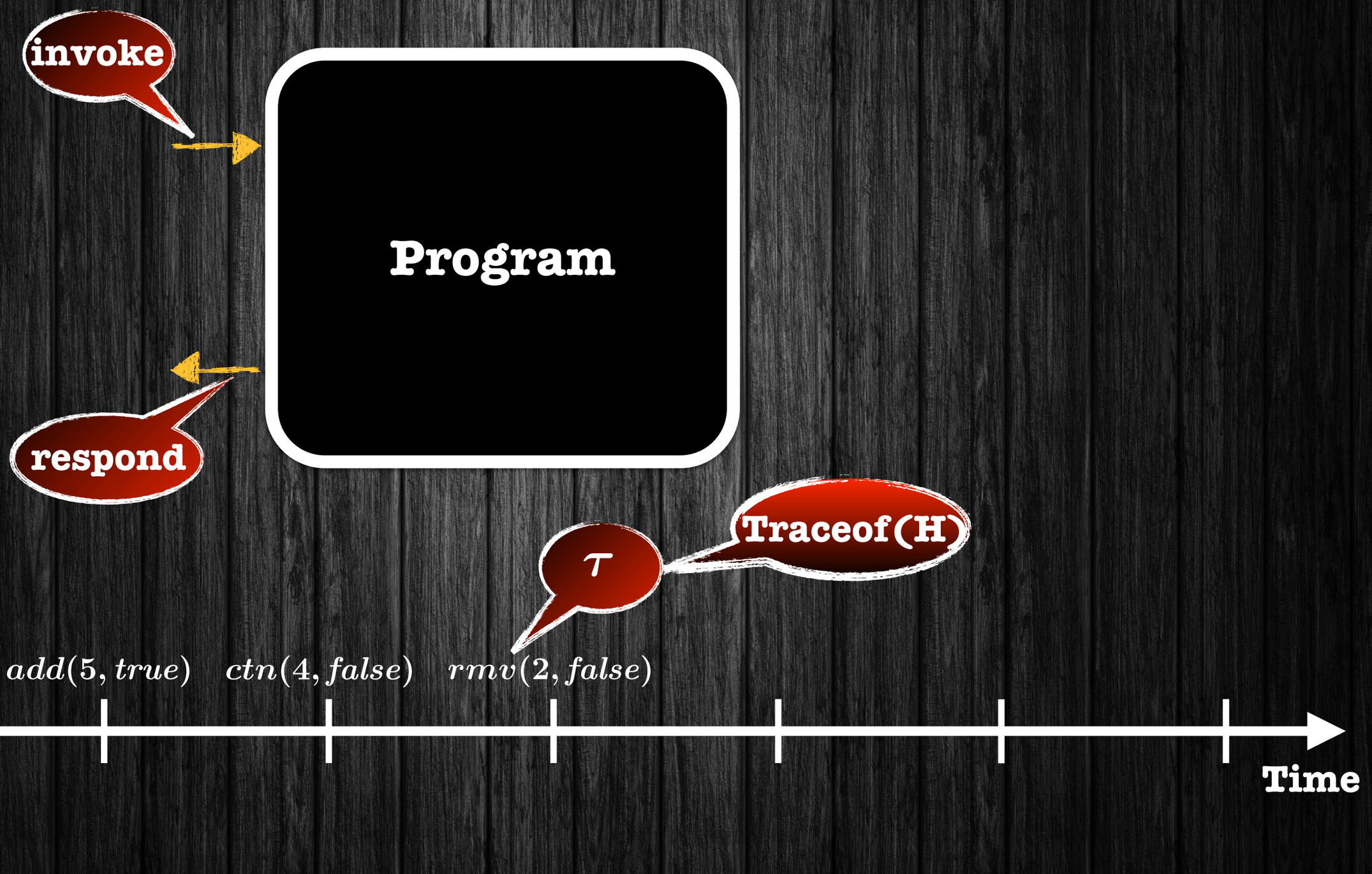
Sequential Programs

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

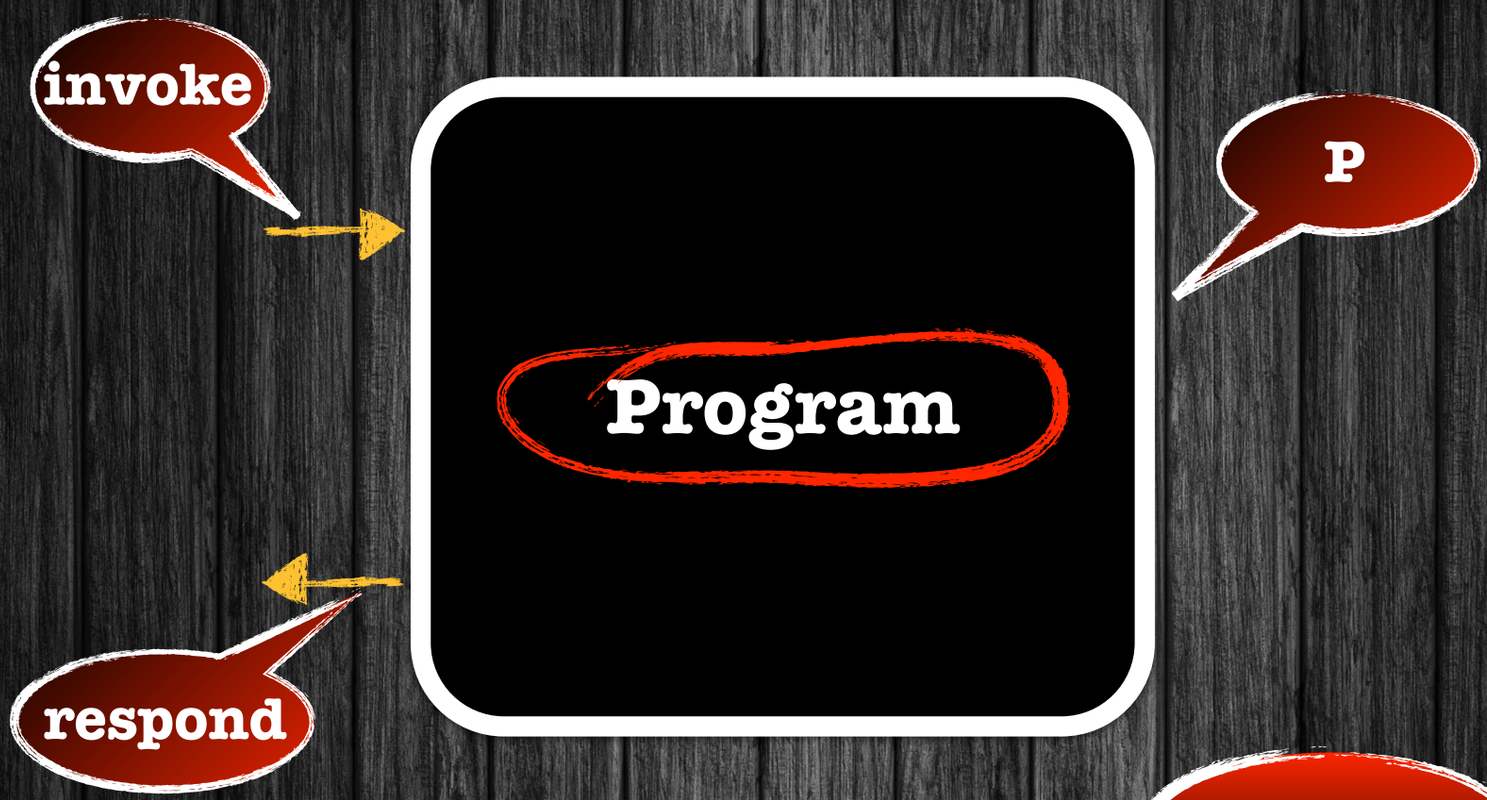
Sequential Programs

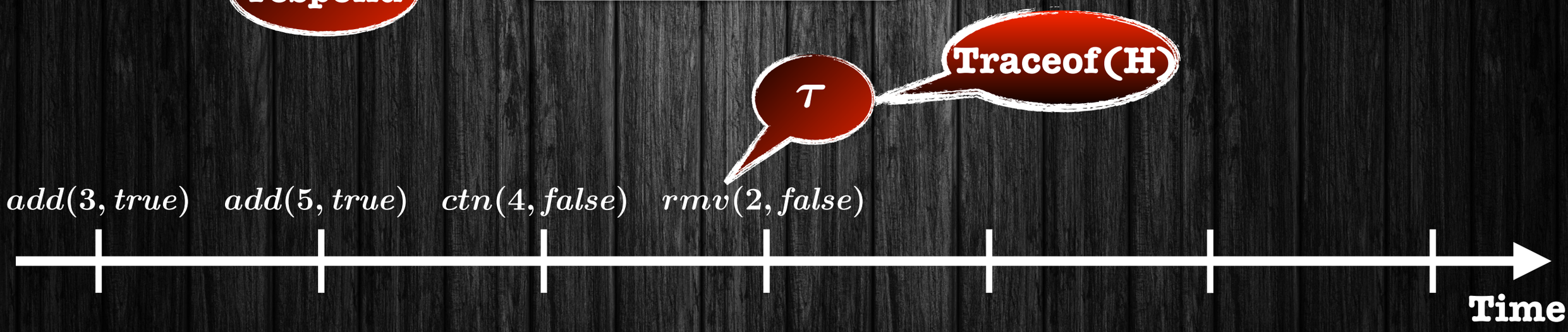
- Programs
- Sequential Histories
- Histories → Traces
- Example: implementing a set
- Program → Histories → Traces

Sets | Sequential Programs

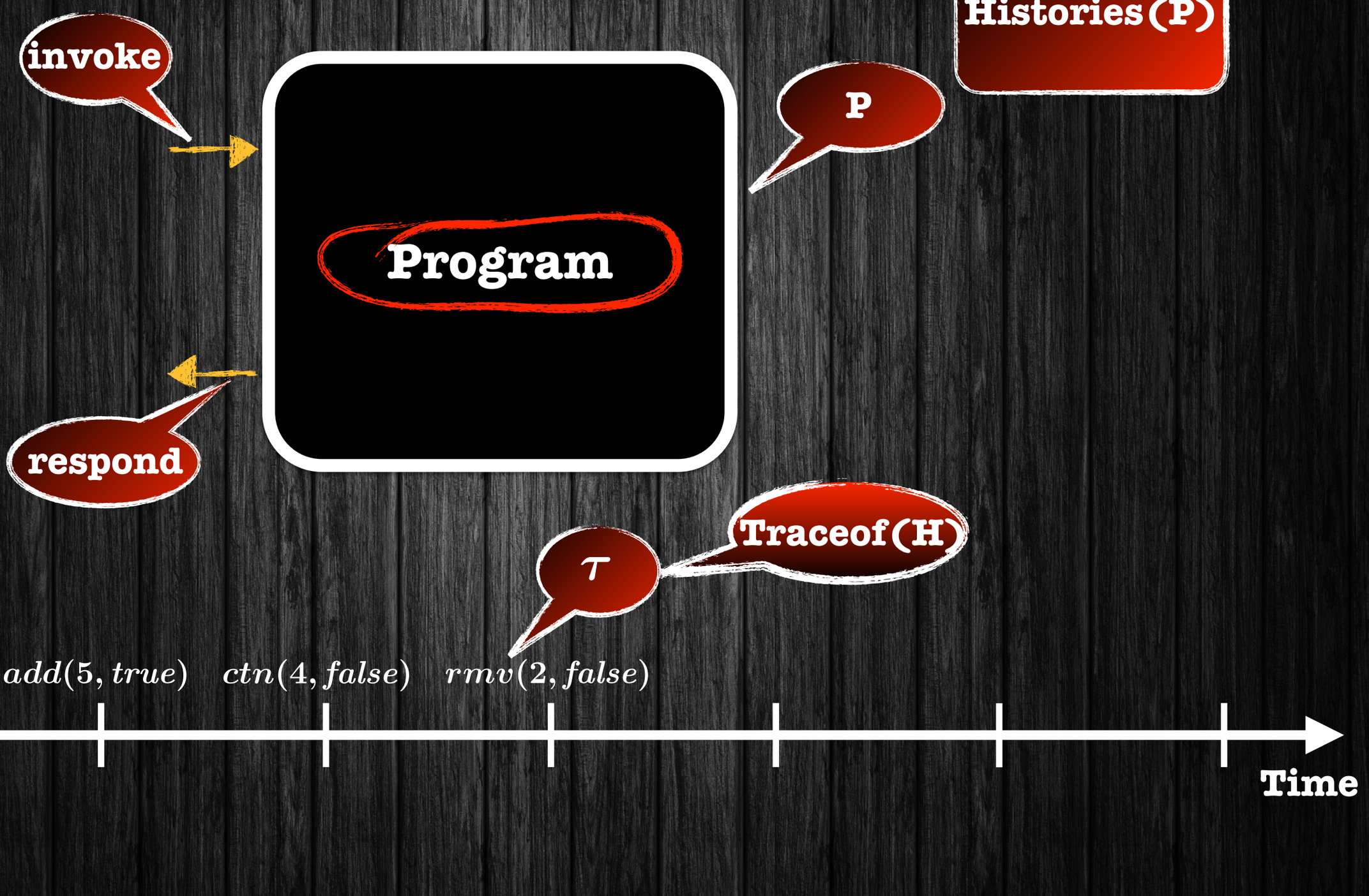


Sets | Sequential Programs





Sets | Sequential Programs



Sets | Sequential Programs

invoke

Program

respond

P

Histories(P)

Traces(P)

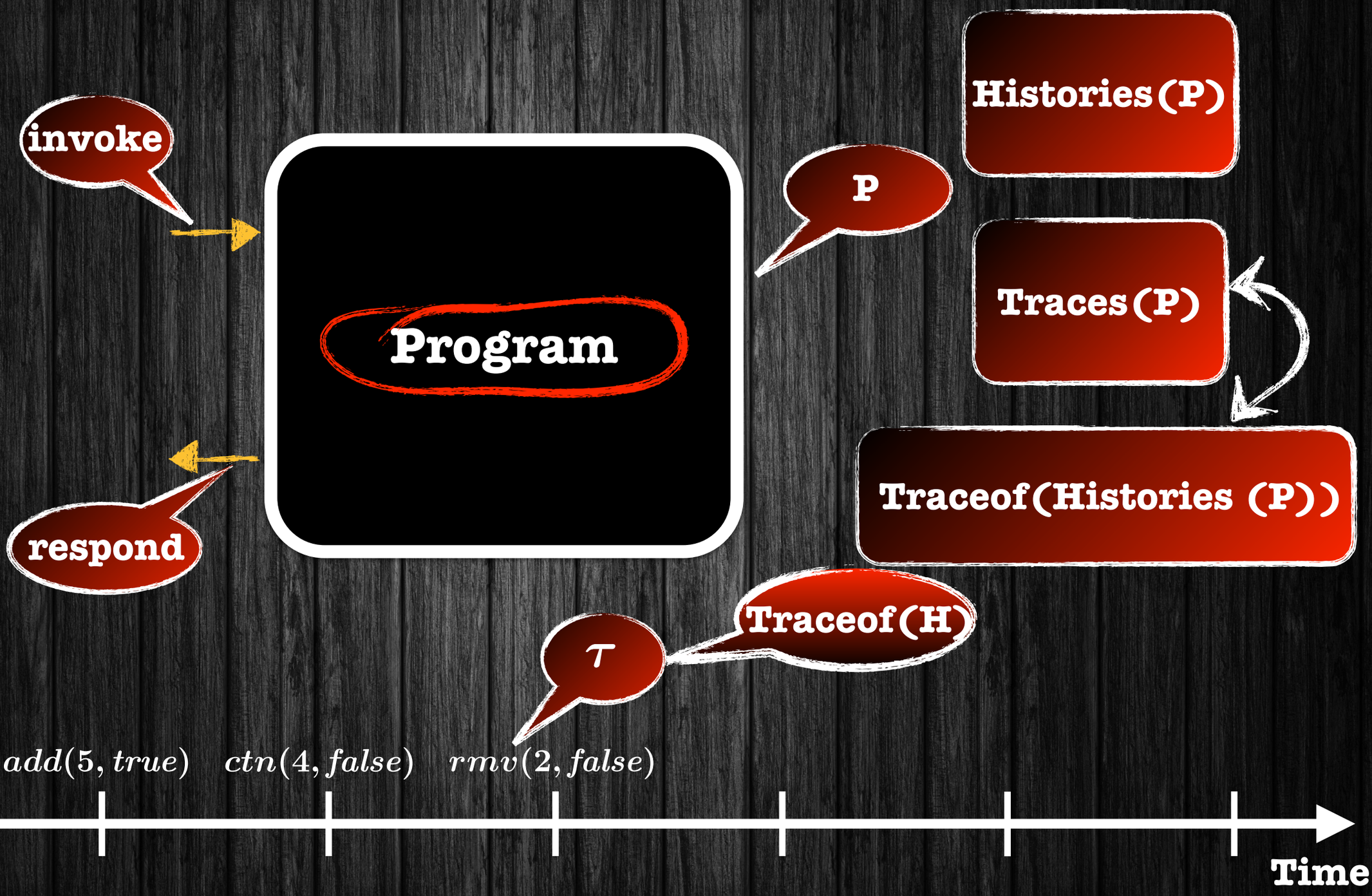
τ

Traceof(H)

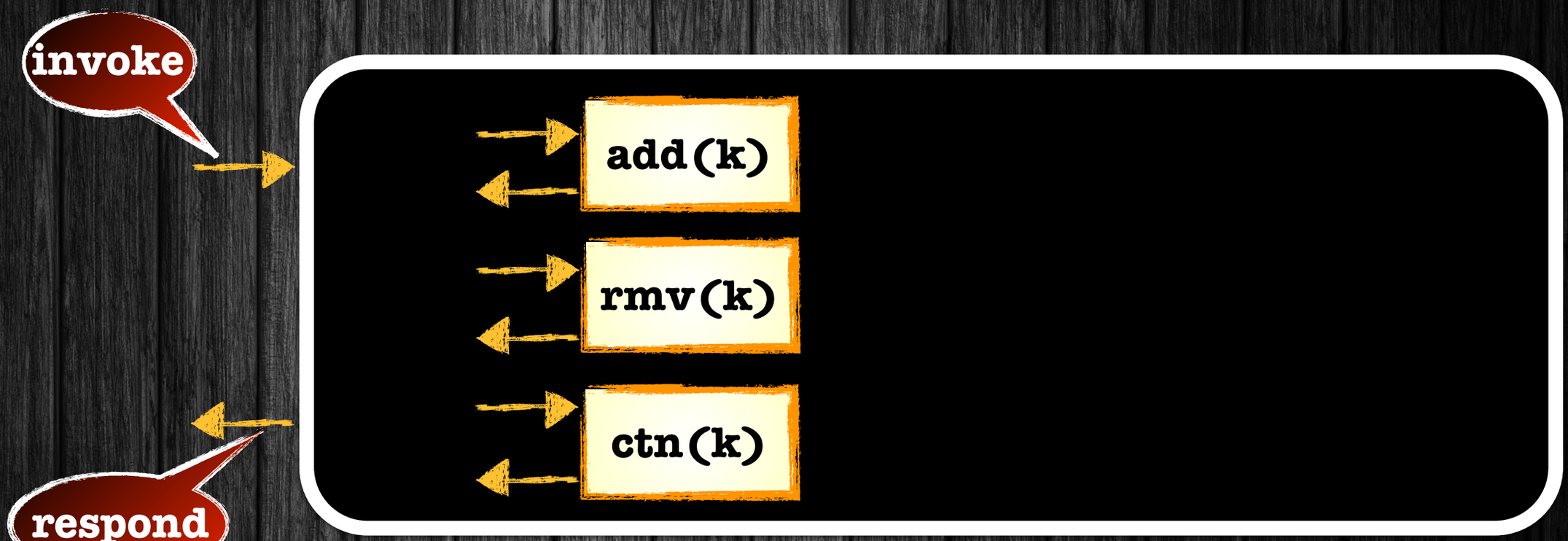
$add(3, true)$ $add(5, true)$ $ctn(4, false)$ $rmv(2, false)$

Time

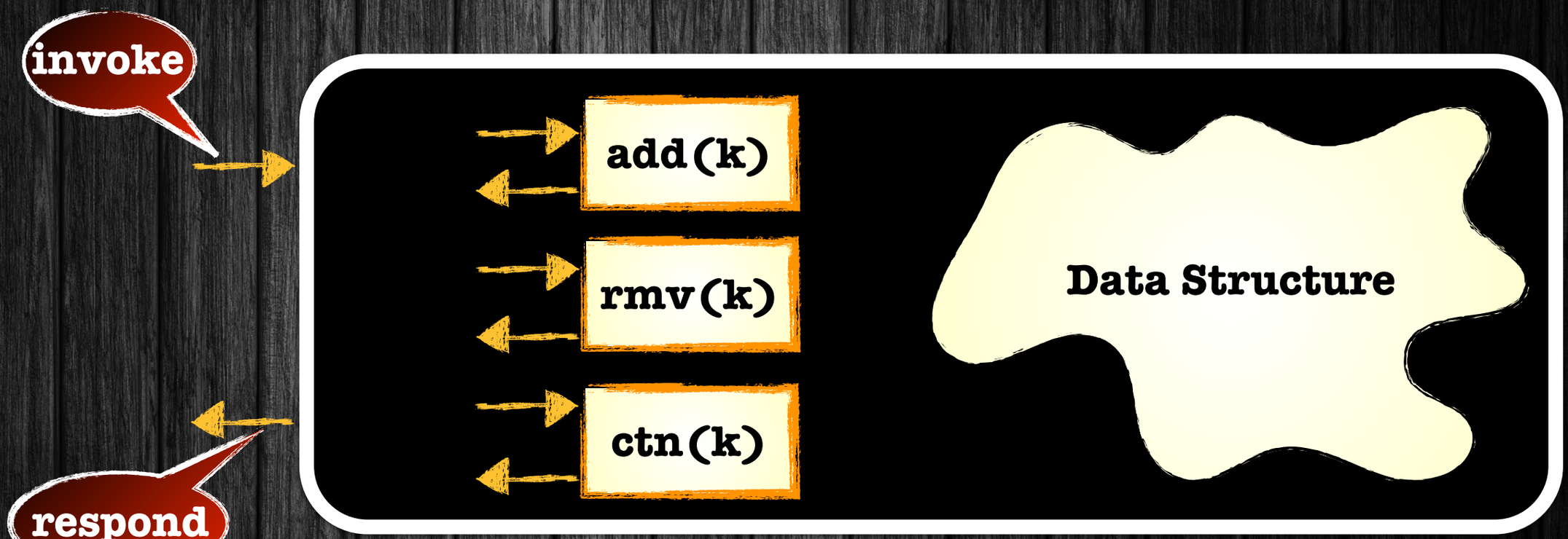
Sets | Sequential Programs



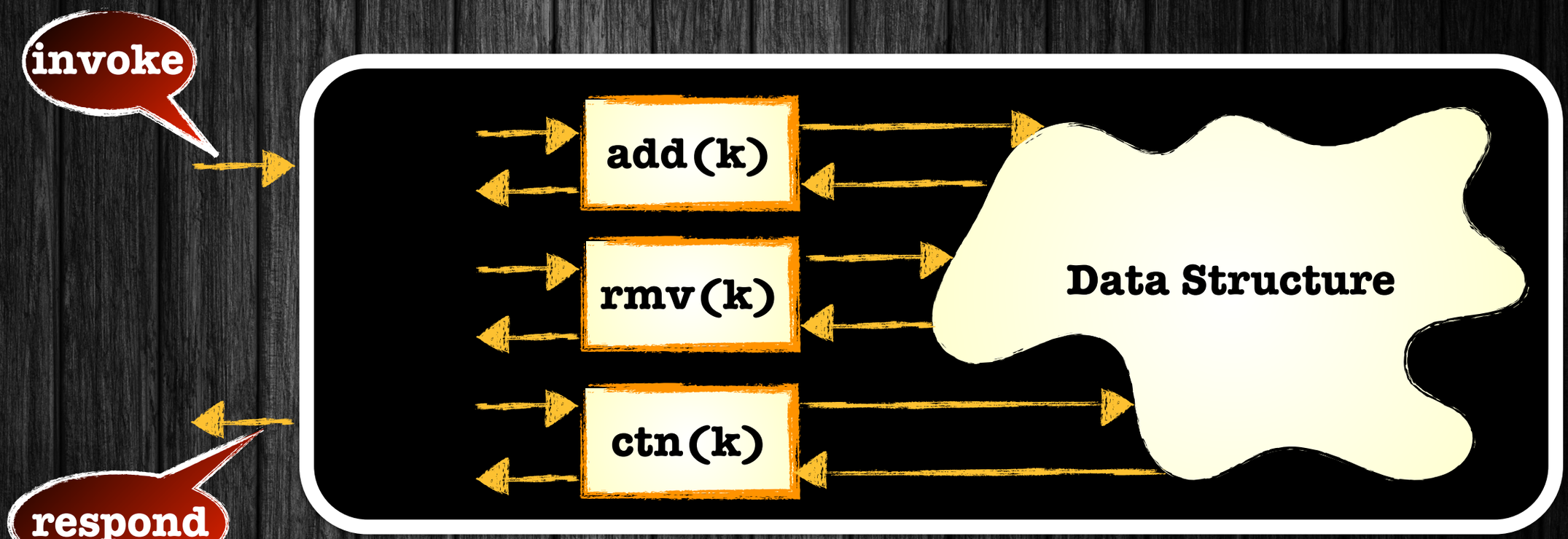
Sets | Sequential Programs



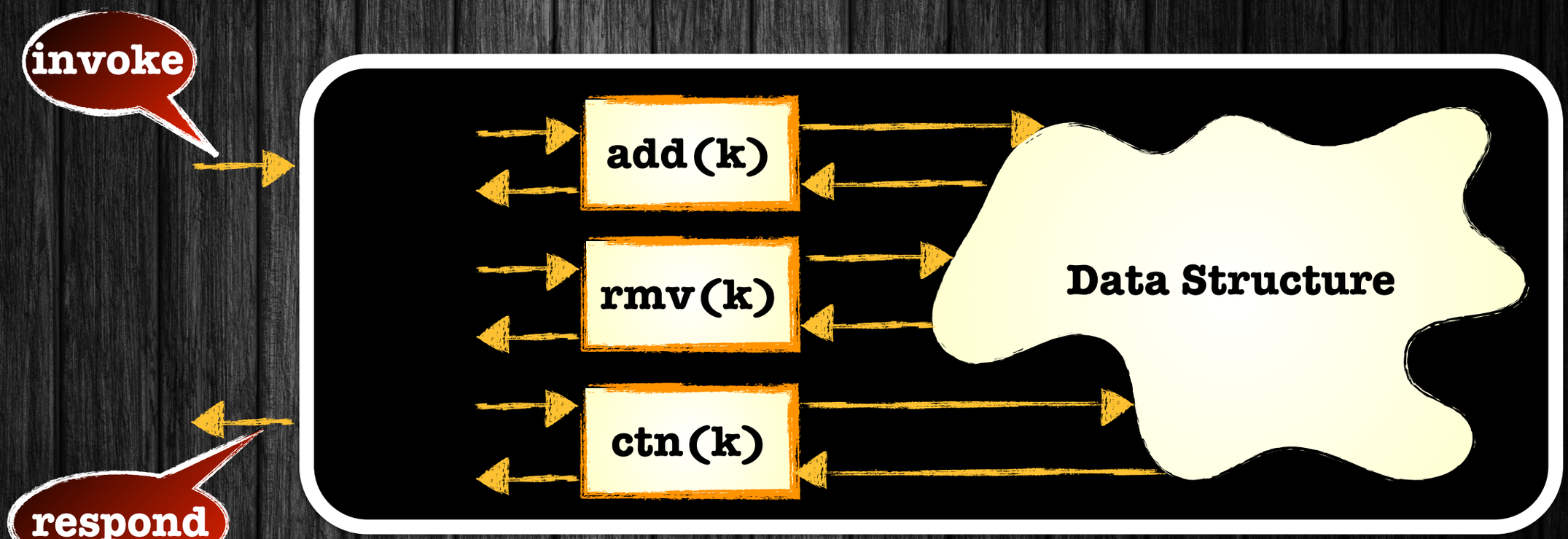
Sets | Sequential Programs



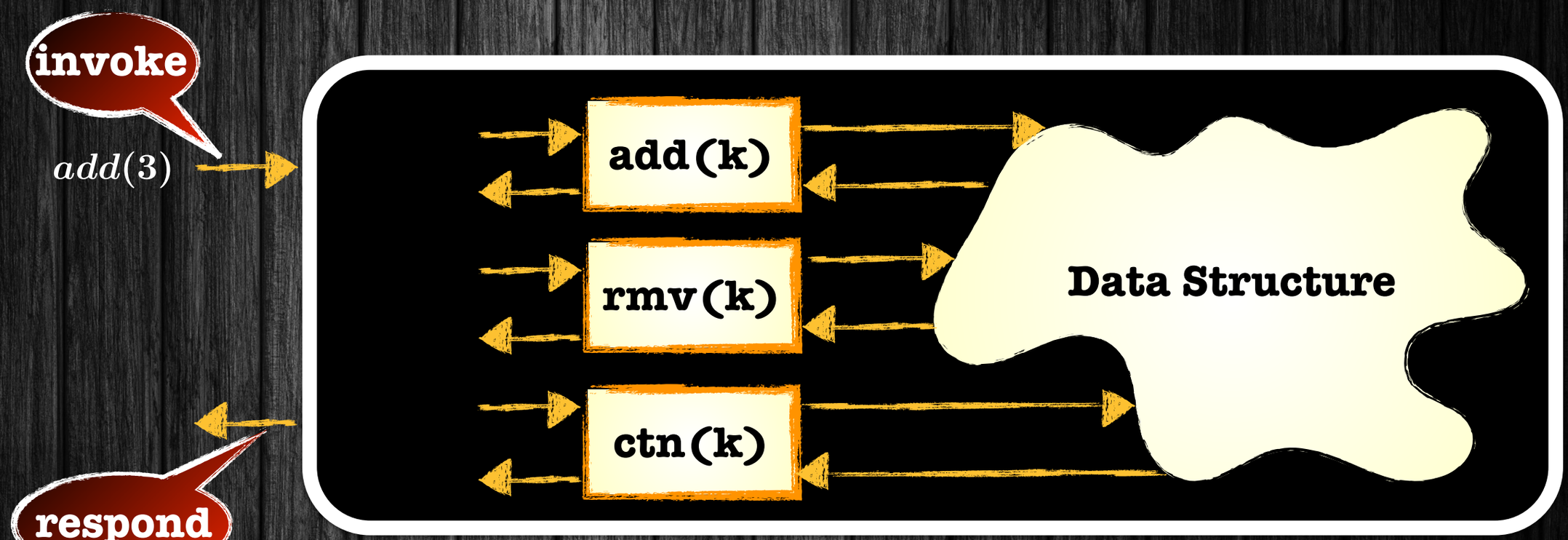
Sets | Sequential Programs



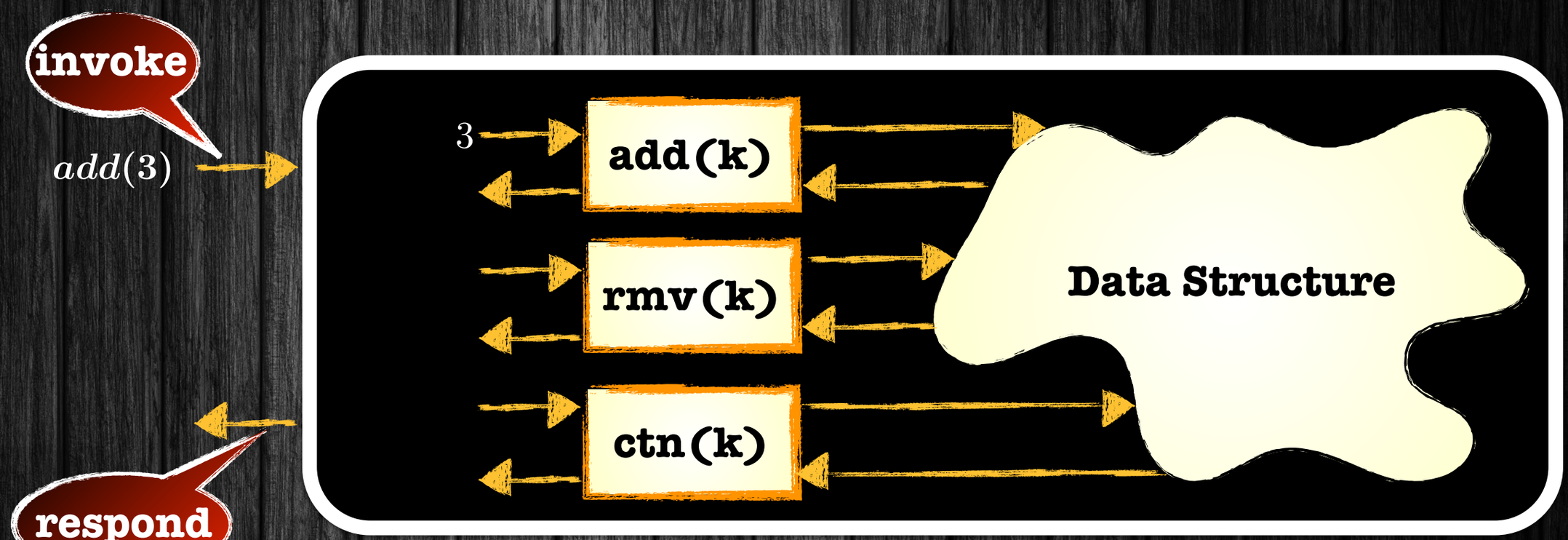
Sets | Sequential Programs



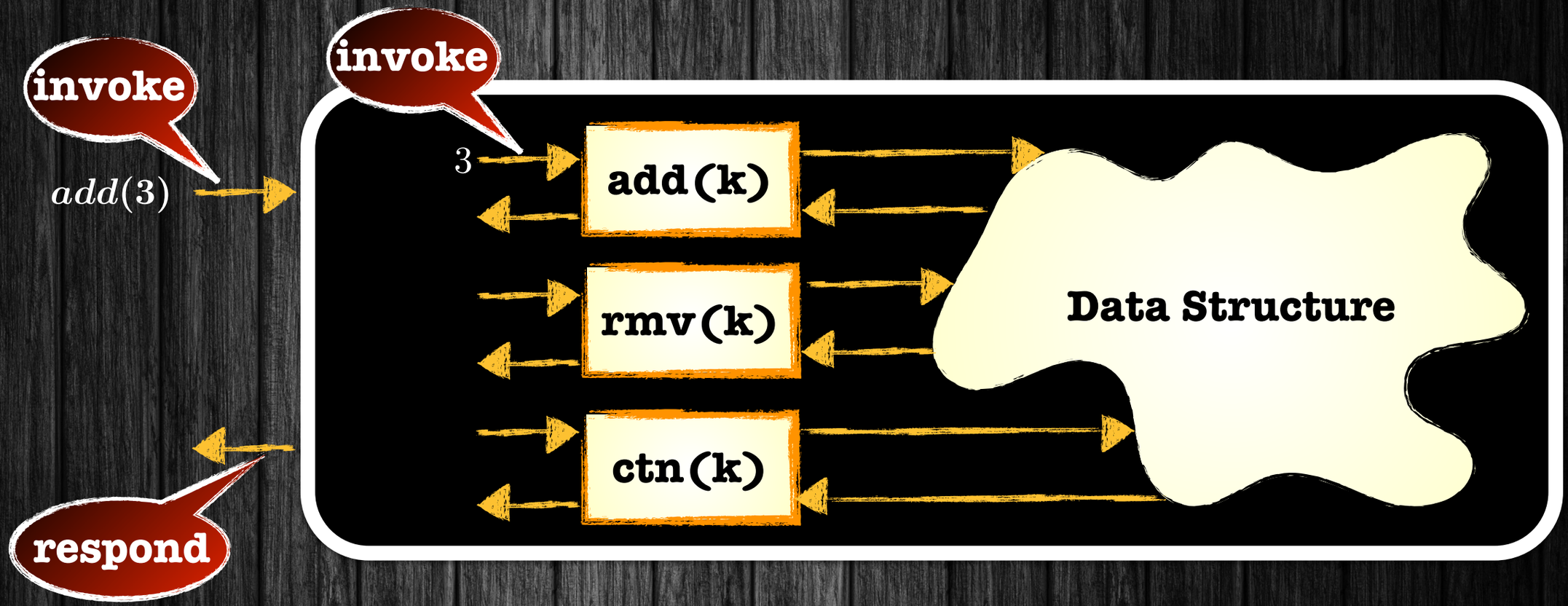
Sets | Sequential Programs



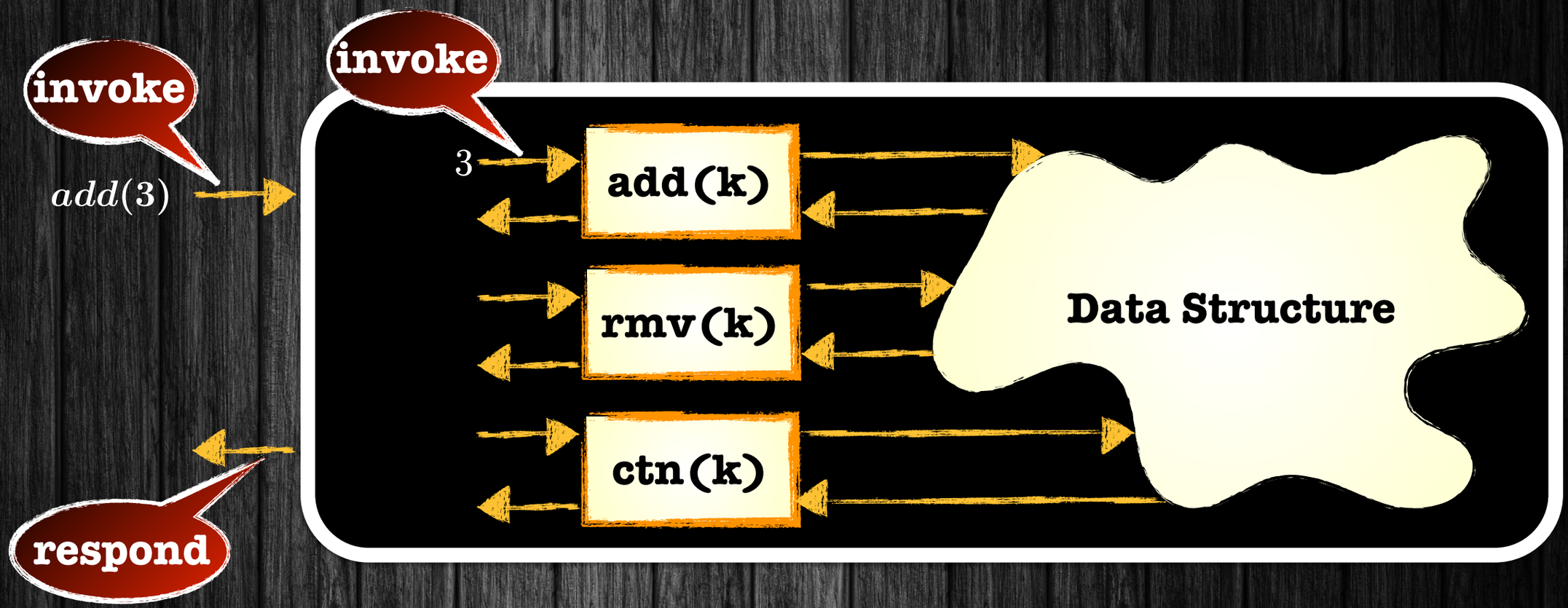
Sets | Sequential Programs



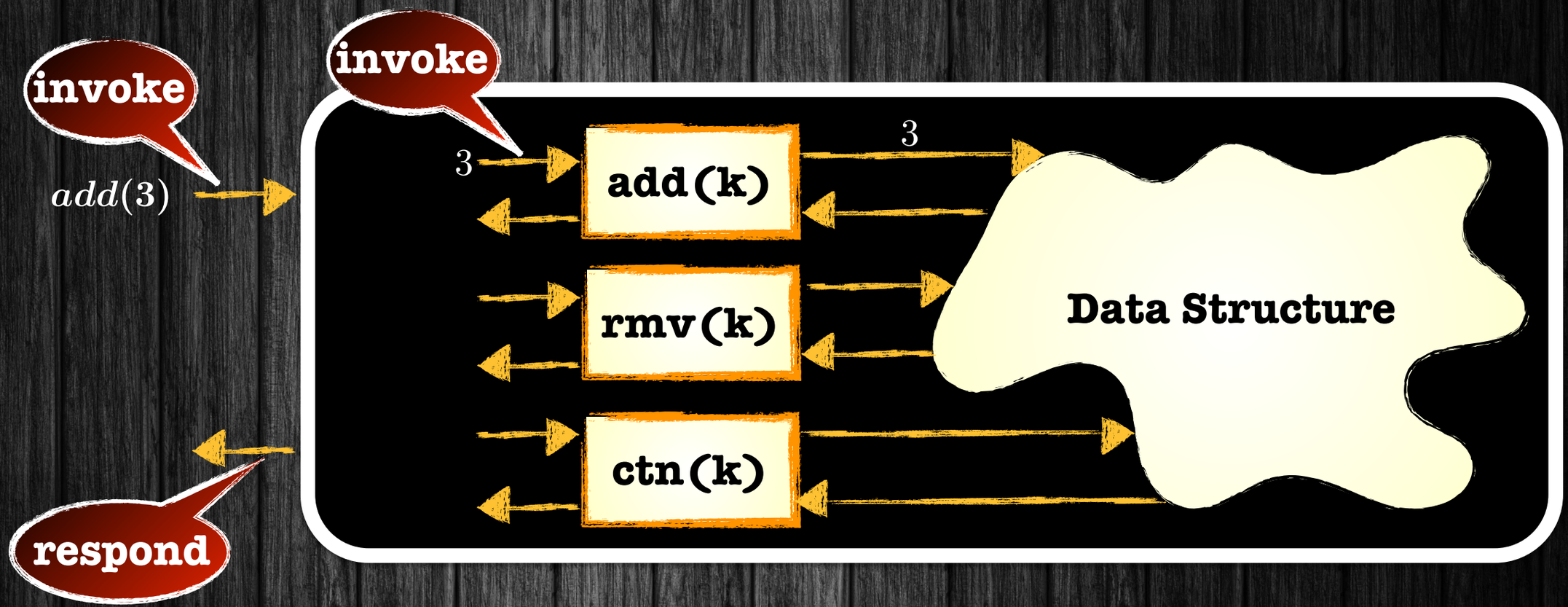
Sets | Sequential Programs



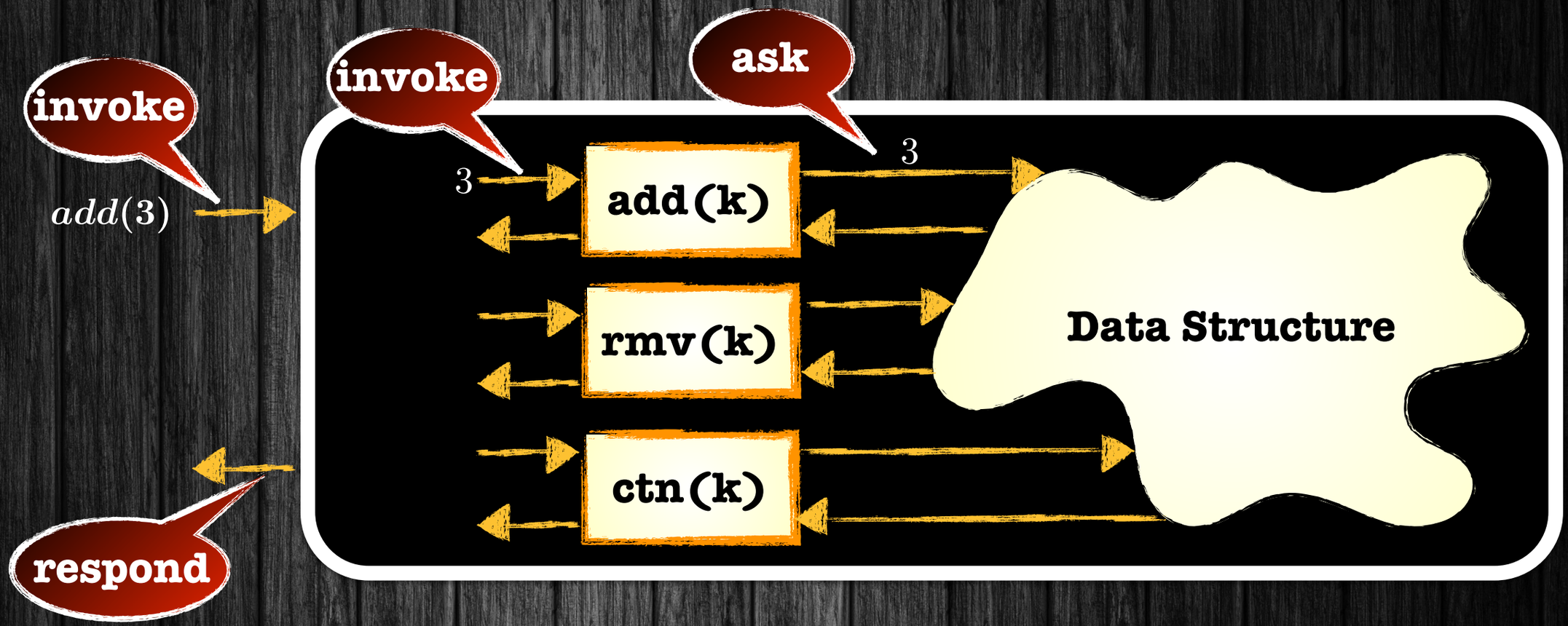
Sets | Sequential Programs



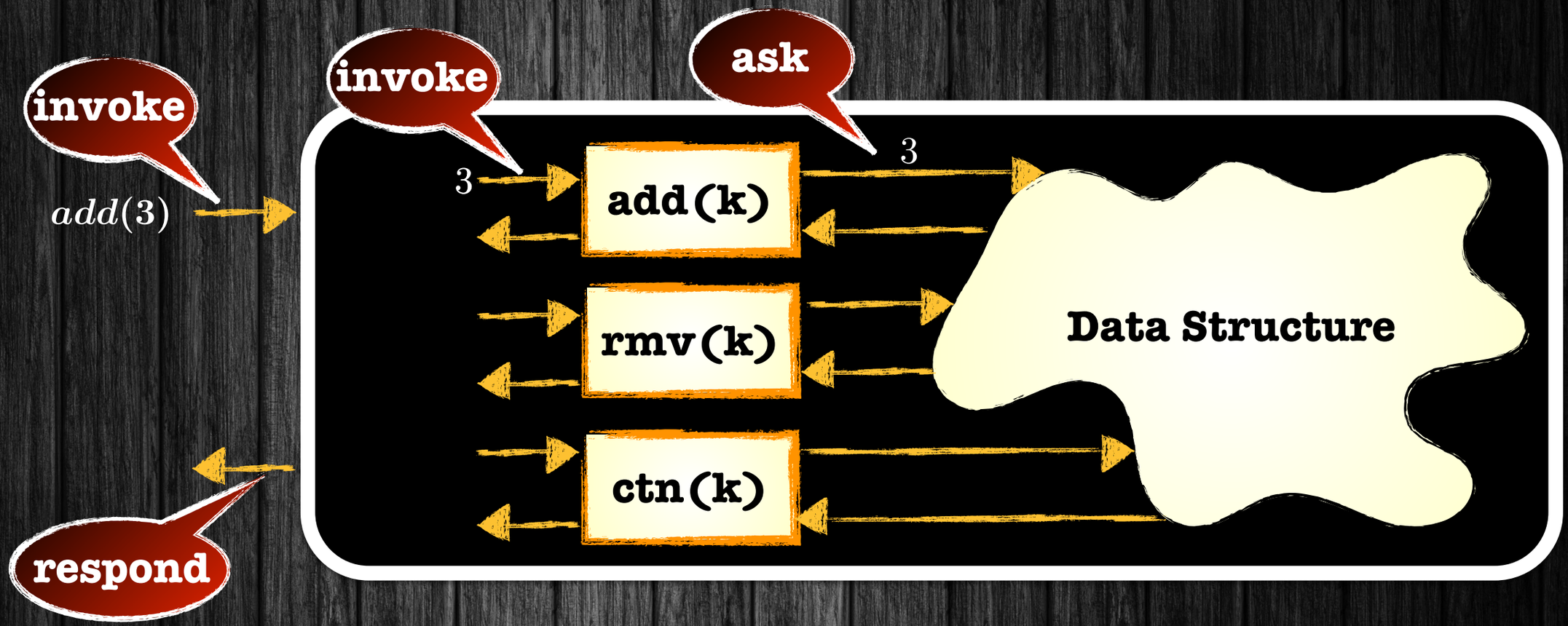
Sets | Sequential Programs



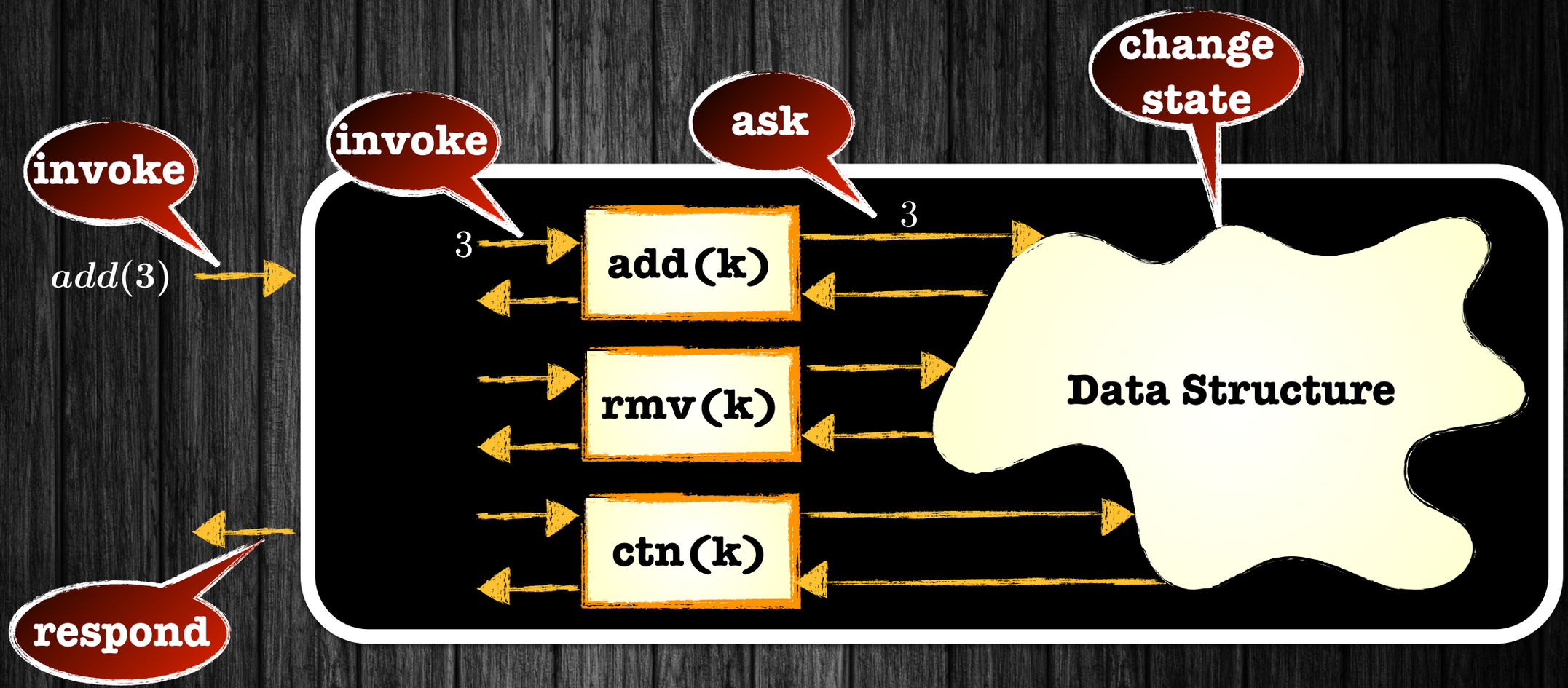
Sets | Sequential Programs



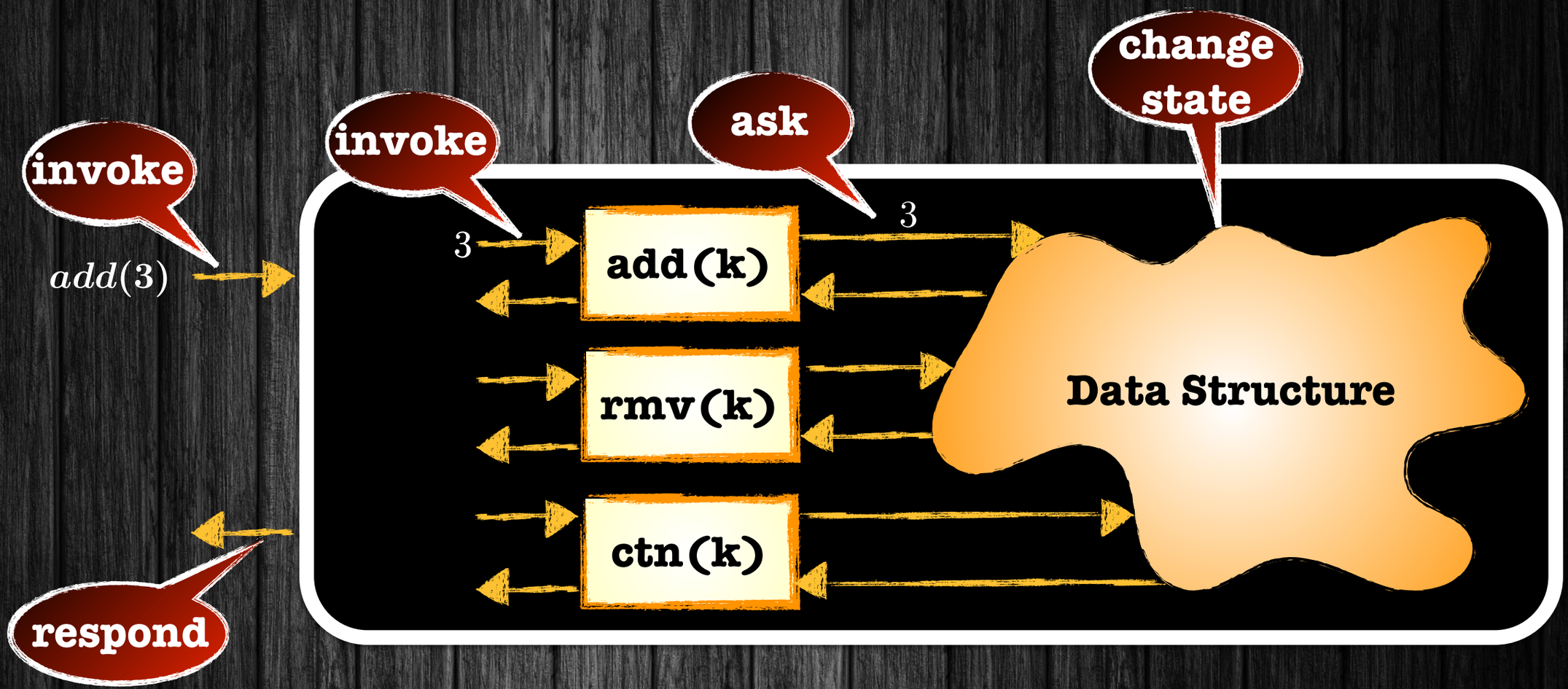
Sets | Sequential Programs



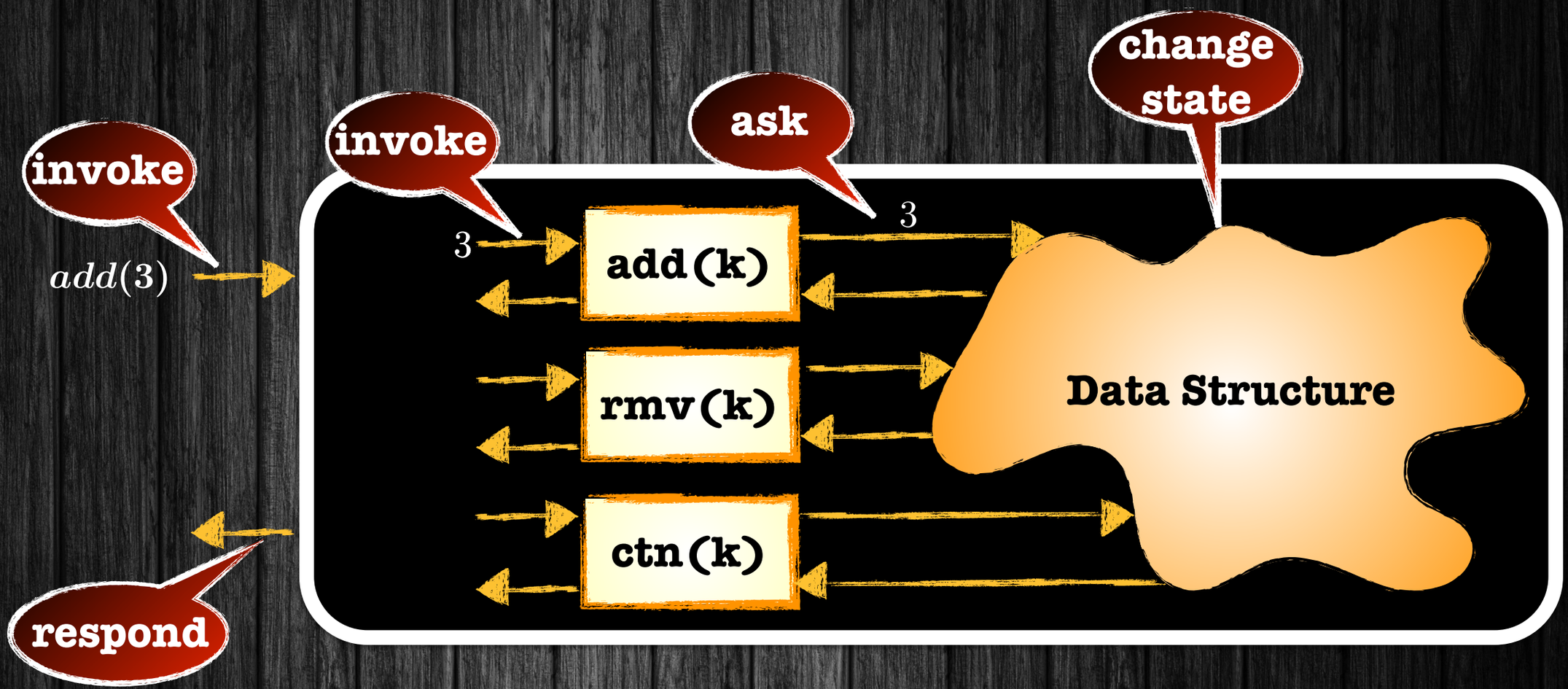
Sets | Sequential Programs



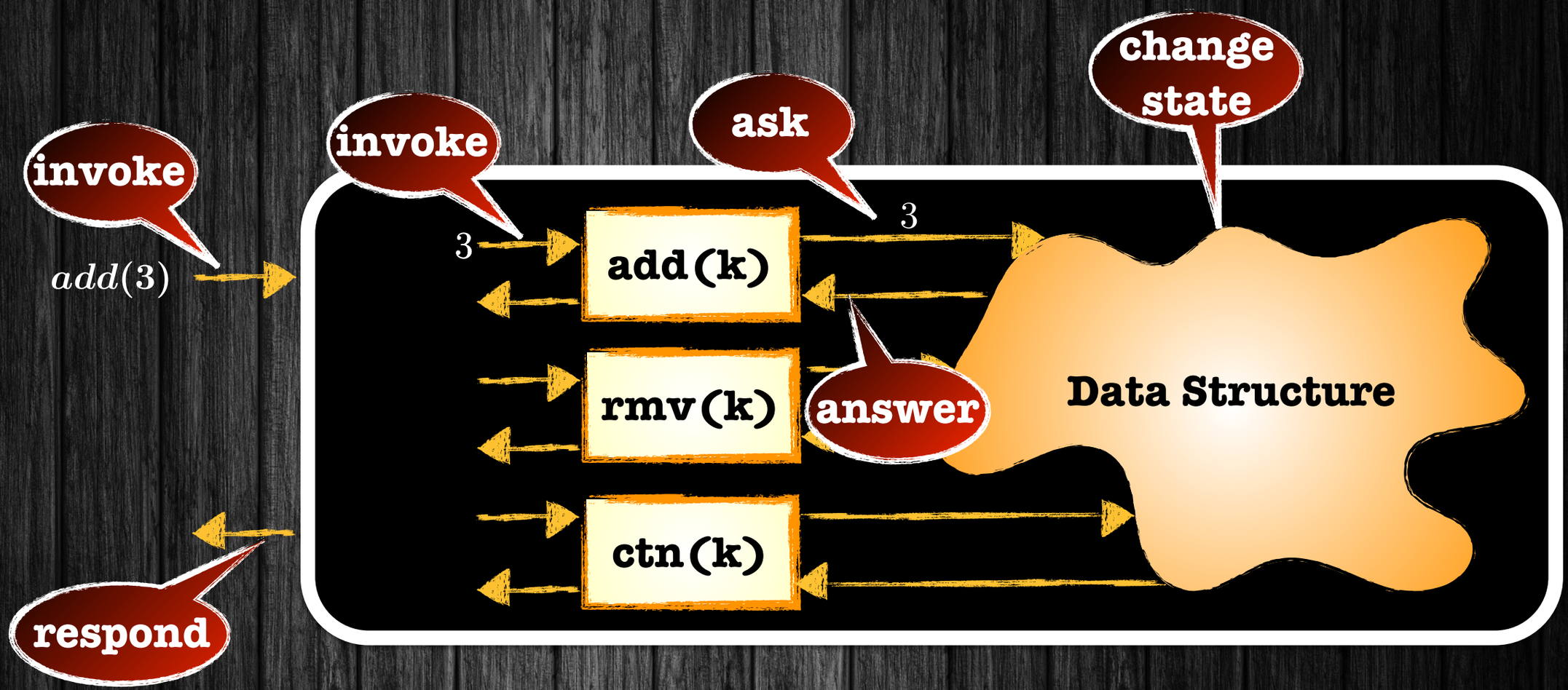
Sets | Sequential Programs



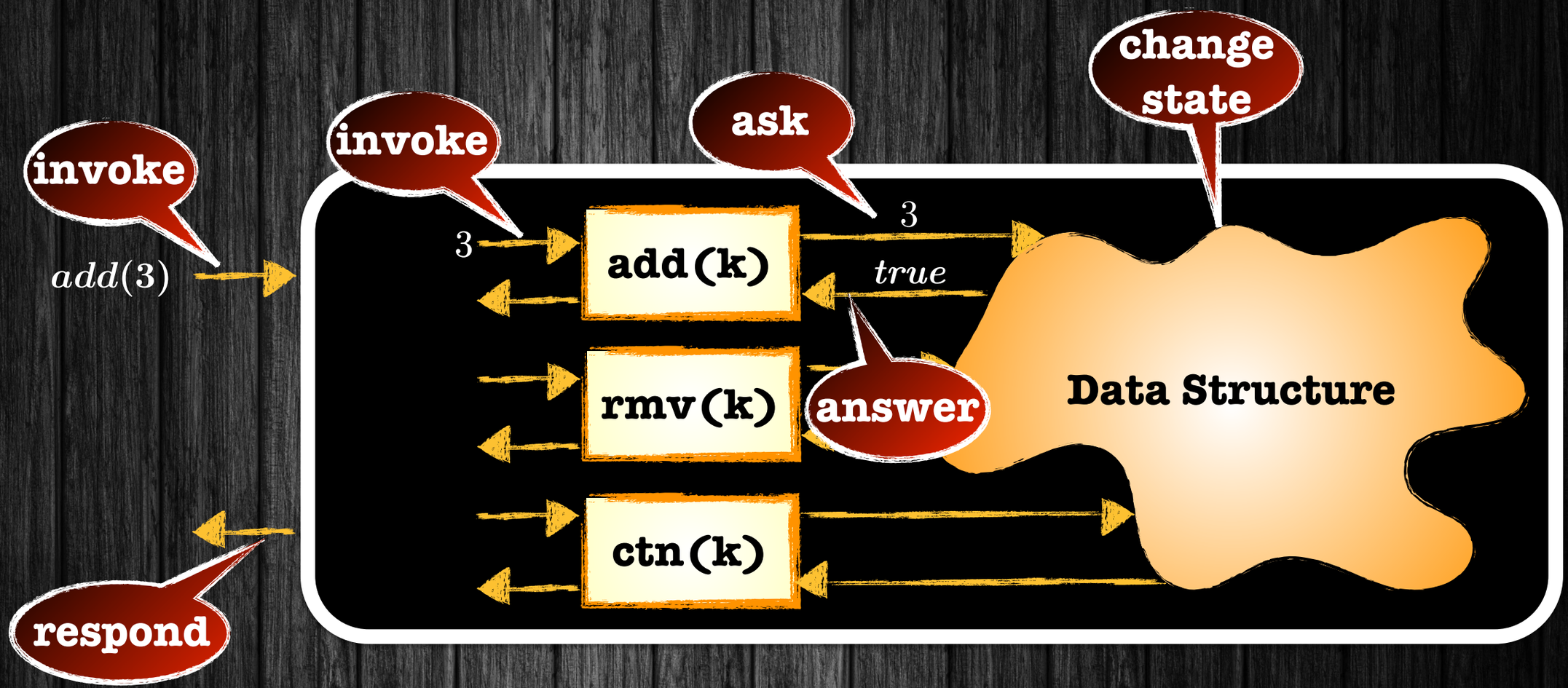
Sets | Sequential Programs



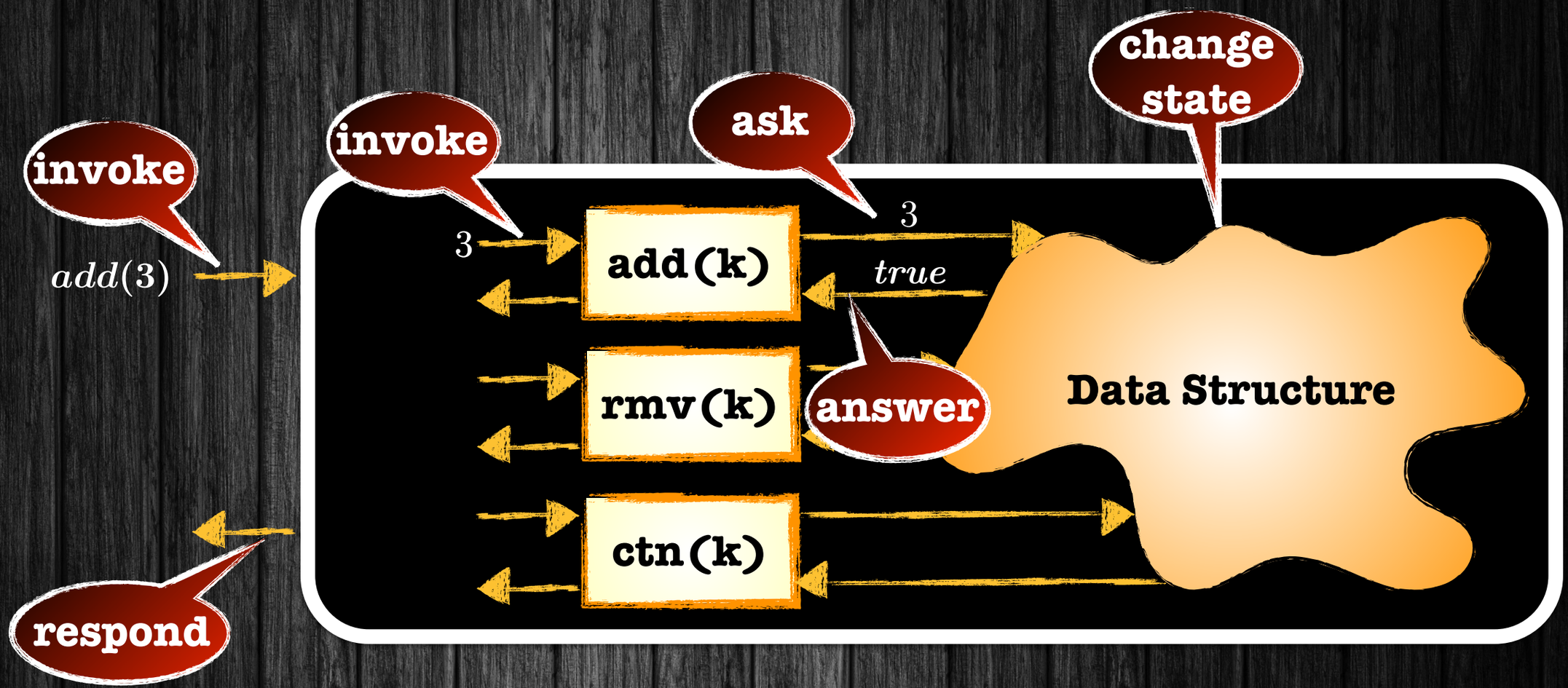
Sets | Sequential Programs



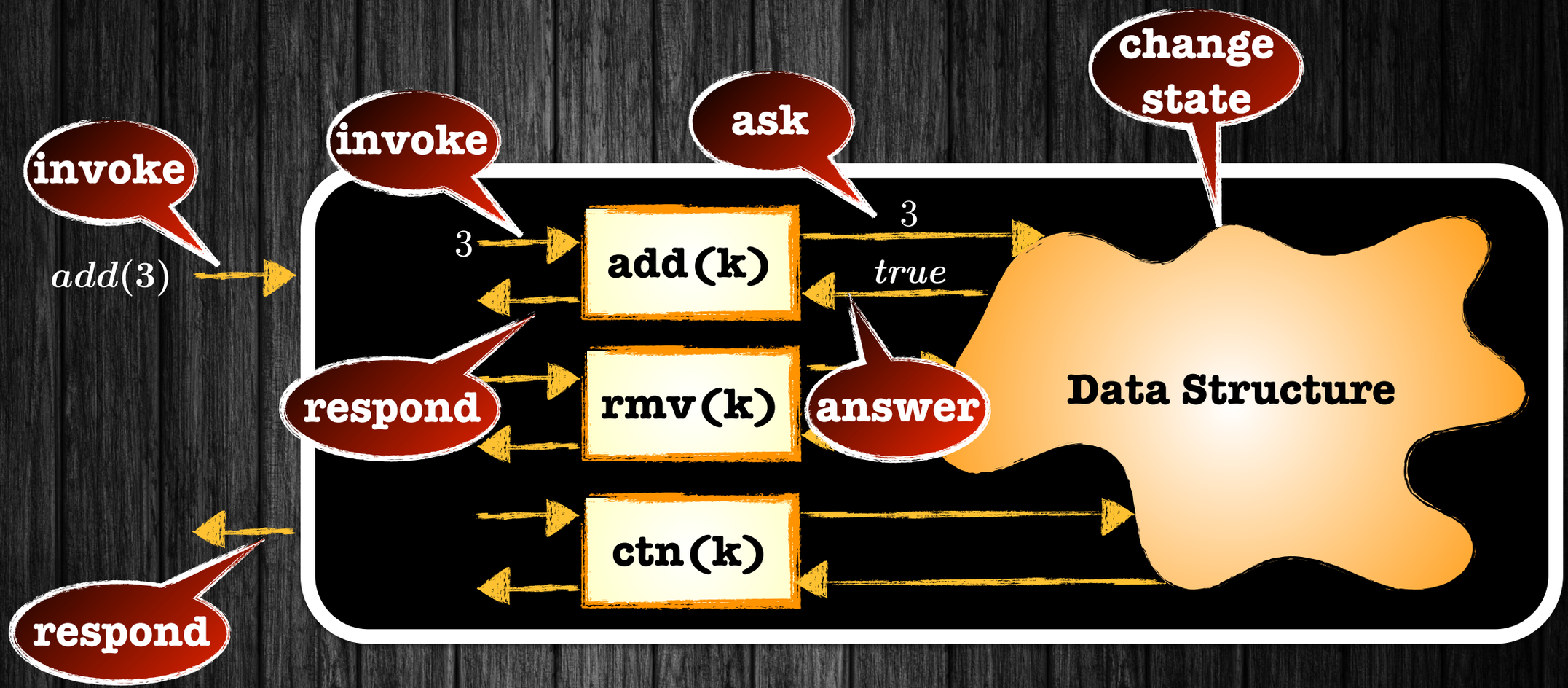
Sets | Sequential Programs



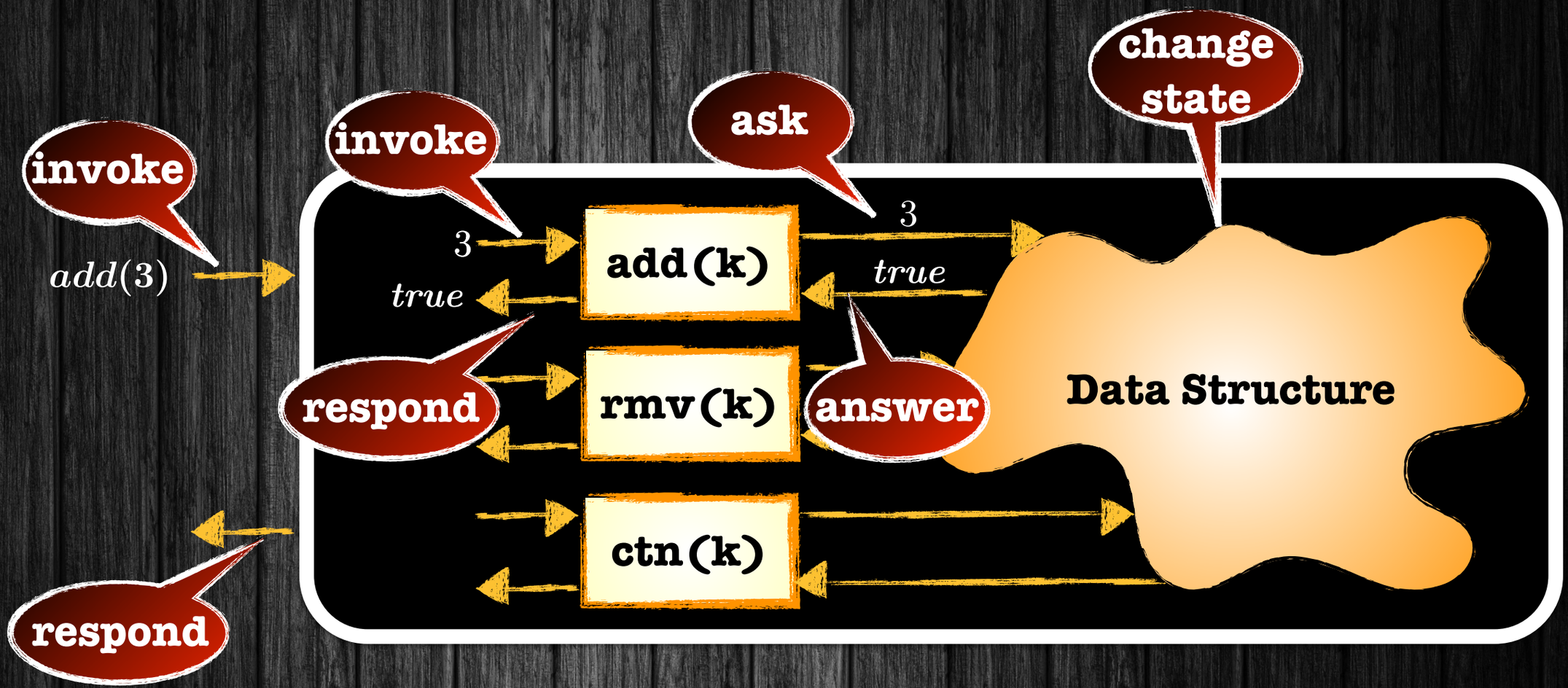
Sets | Sequential Programs



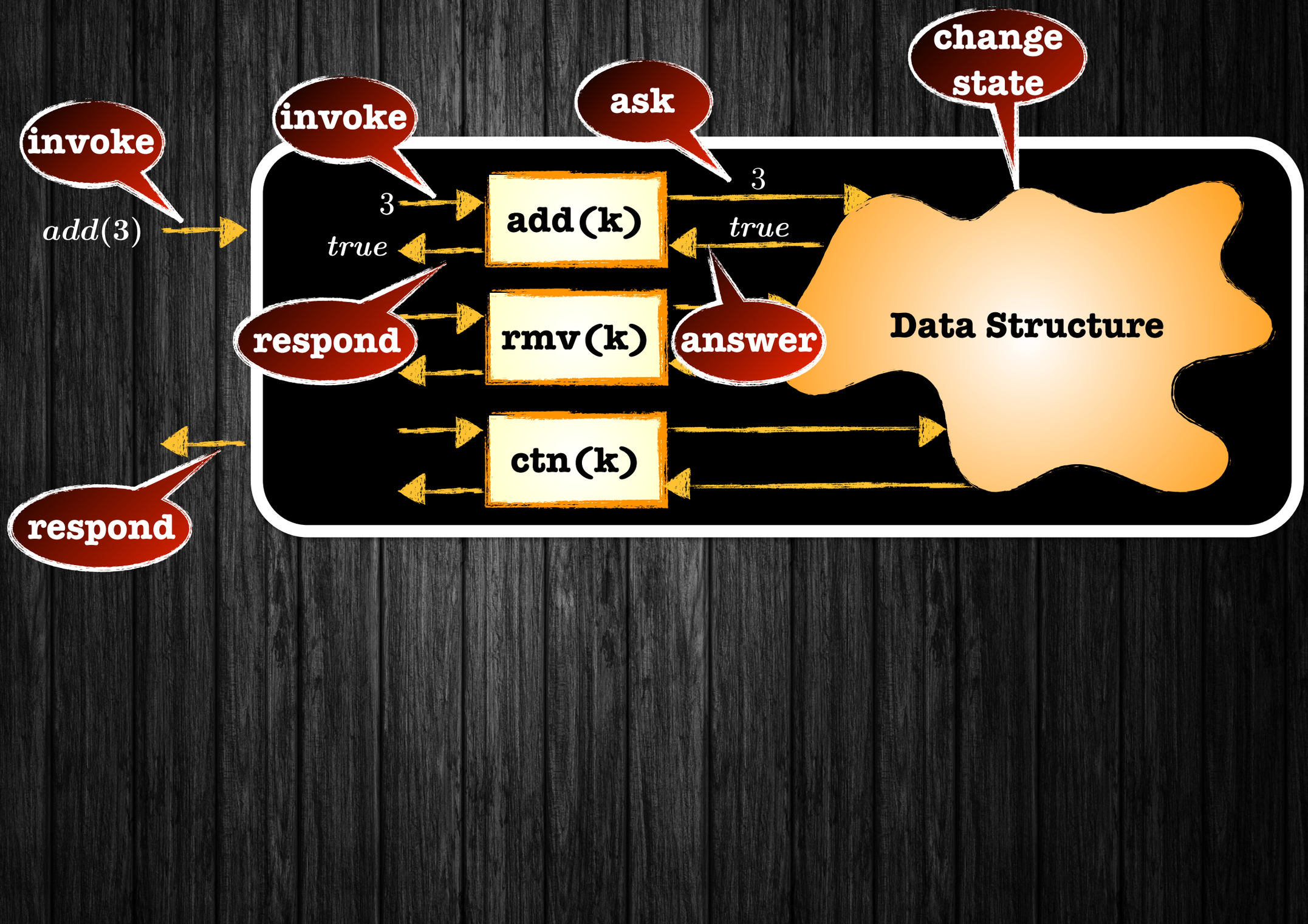
Sets | Sequential Programs



Sets | Sequential Programs



Sets | Sequential Programs



Sets | Sequential Programs

