CertiKOS: A Breakthrough toward Hacker-Resistant Operating Systems

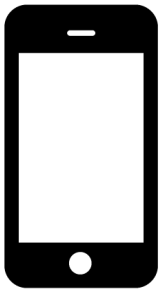
Zhong Shao
Yale University
January 25, 2018

Acknowledgement: Ronghui Gu, Newman Wu, Hao Chen, Jieung Kim, Jeremie Koenig, Vilhelm Sjoberg, Mengqi Liu, Lionel Rieg, Quentin Carbonneaux, Unsung Lee, Jiyong Shin, David Costanzo, Tahina Ramananandro, Hernan Vanzetto, Shu-Chun Weng, Zefeng Zeng, Zhencao Zhang, Liang Gu, Jan Hoffmann, Joshua Lockerman, and Bryan Ford. This research is supported in part by DARPA CRASH and HACMS programs and NSF SaTC and **Expeditions in Computing** programs.

Motivation

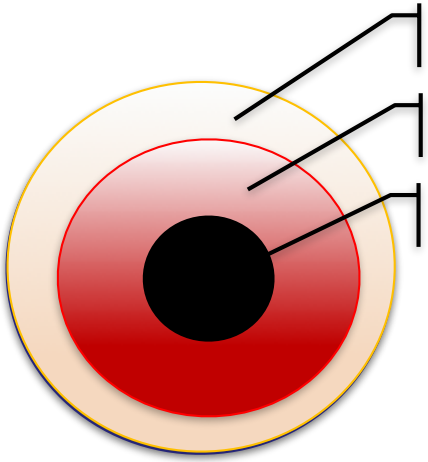
Desktop

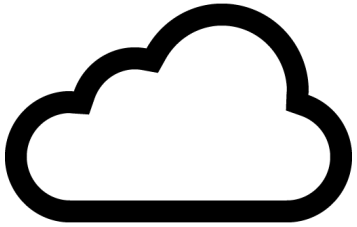
Transportation



Mobile

Health





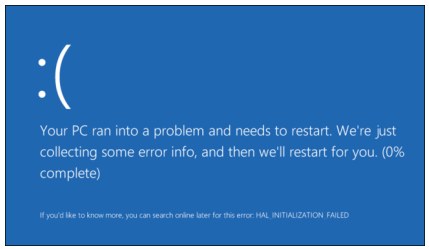
cloud

Aviation

Financial

Environment

Motivation



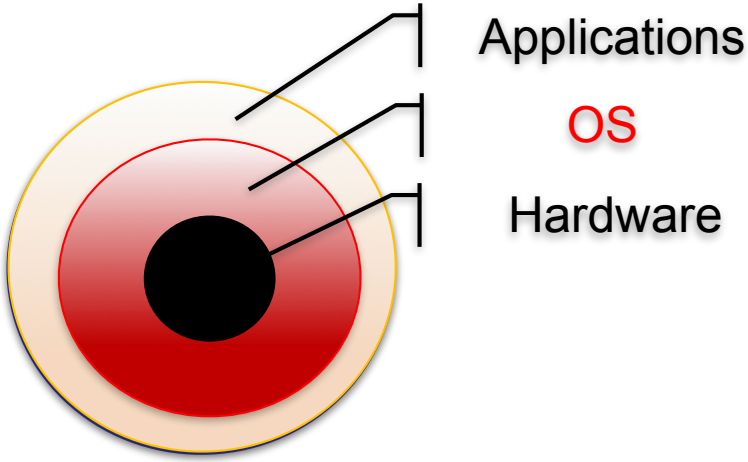
Crash

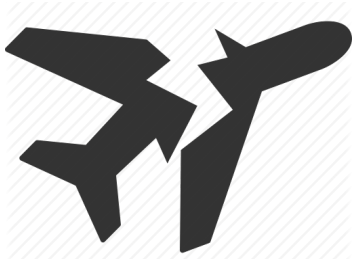
Accident

Mobile

Life

cloud





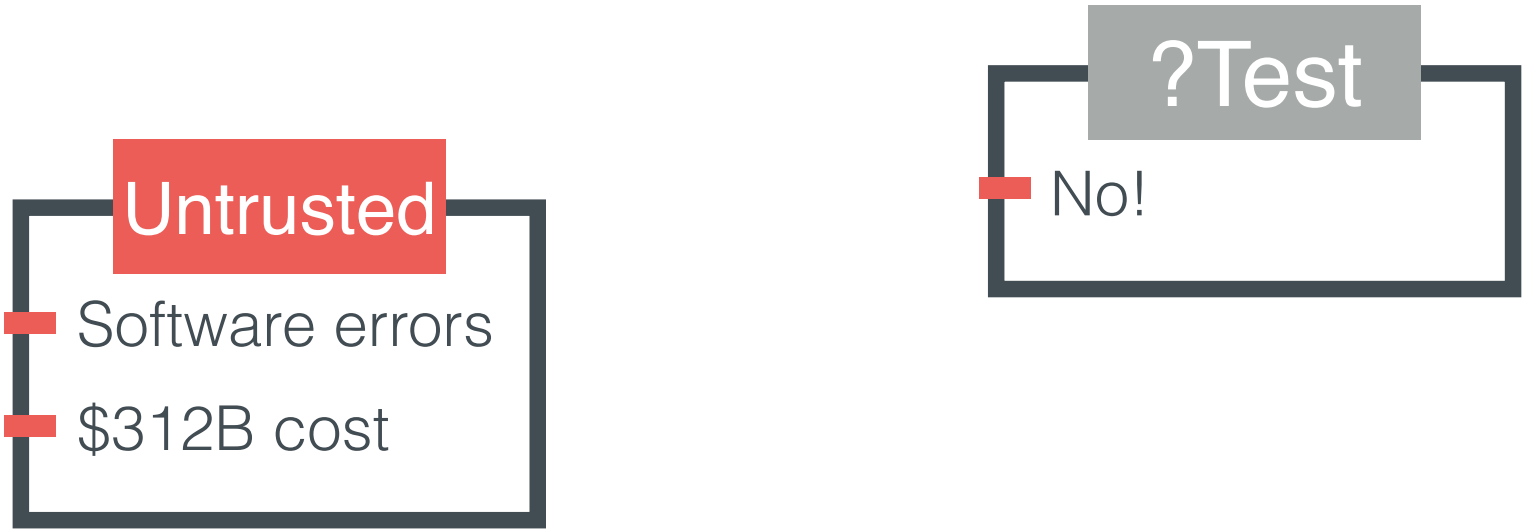
Loss

Financial

Environment

Motivation

System Software Runs Everywhere



Motivation

> “Program testing can be used to show the presence of bugs, but never to show their absence.”
>
> — Edsger Dijkstra

Motivation

> Complete formal verification is the **only** known way to guarantee that a system is free of programming errors.
>
> — seL4 [SOSP'09]

> Formal methods are the **only** reliable way to achieve security and privacy in computer systems.
>
> —NSF SFM Report[2016]

Motivation

Formal Verification

- mathematically prove
- program **meets** specification
- under **all** inputs
- under **all** execution
- rule out entire **classes** of attacks

“ mal verification is the **only** is ”

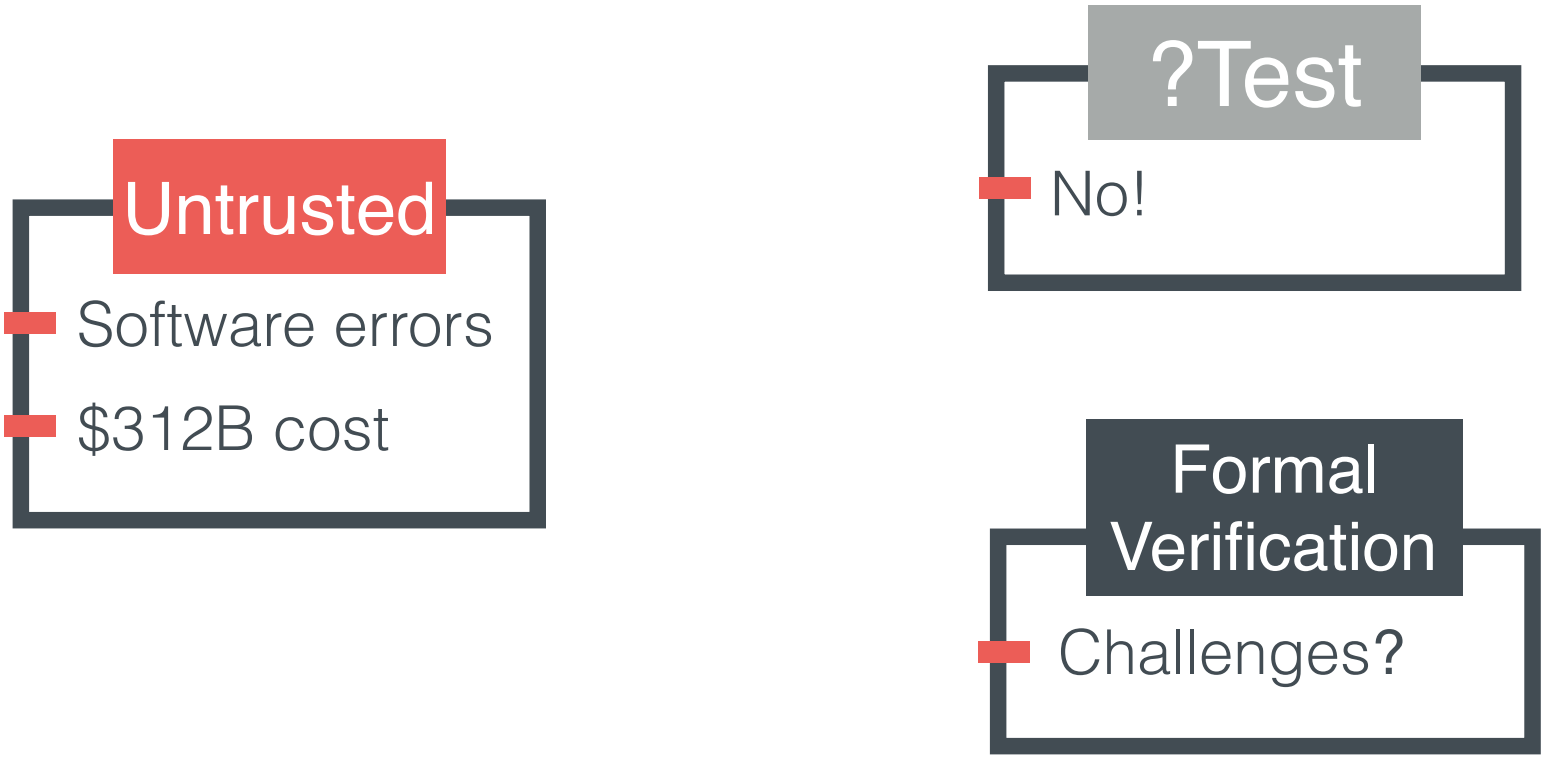
SP’09]

“ way to uter ”

—NSF SFM Report[2016]

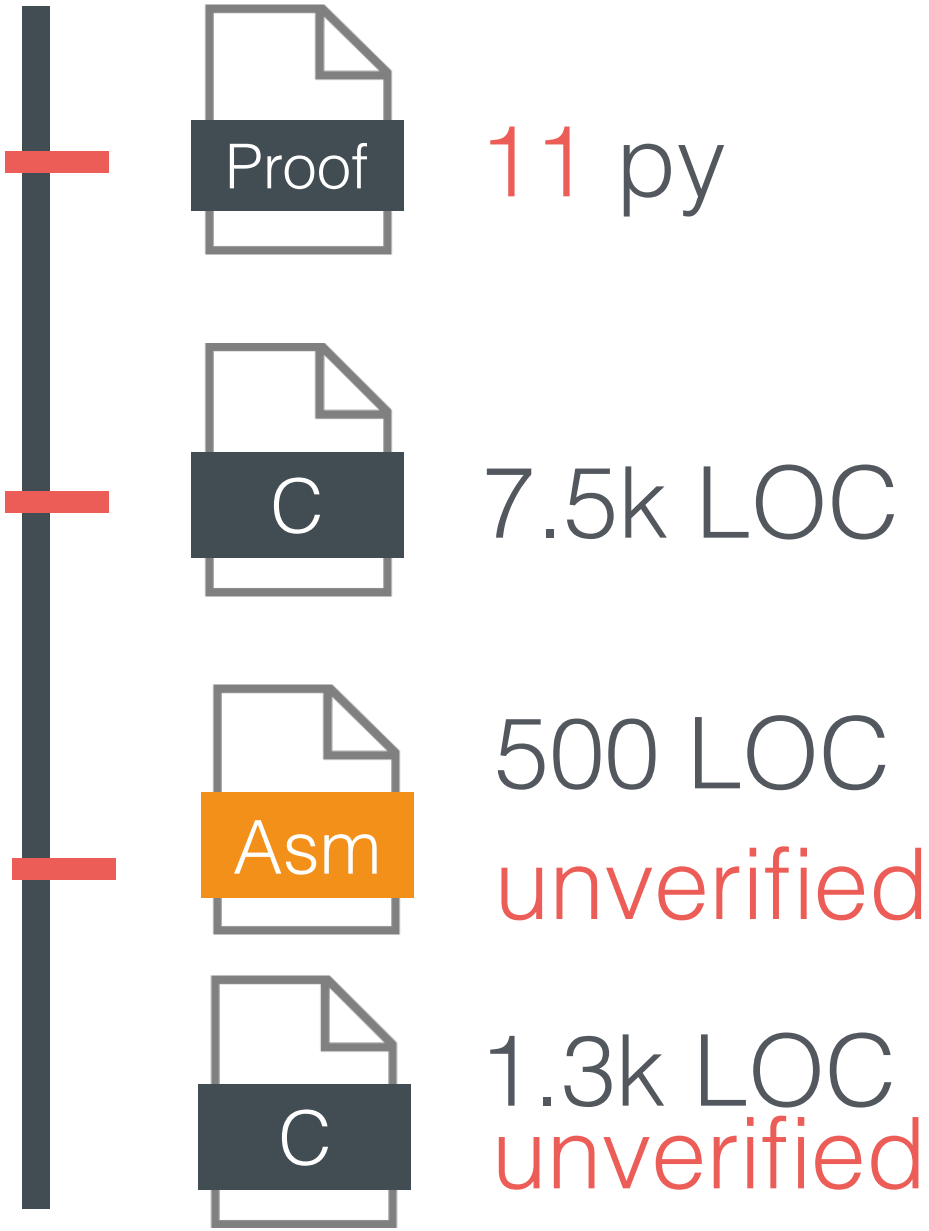
Motivation

System Software Runs Everywhere

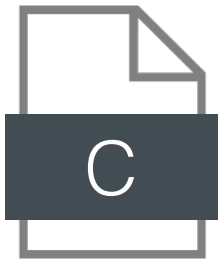


Challenges: huge proof efforts

seL4
[SOSP'09]

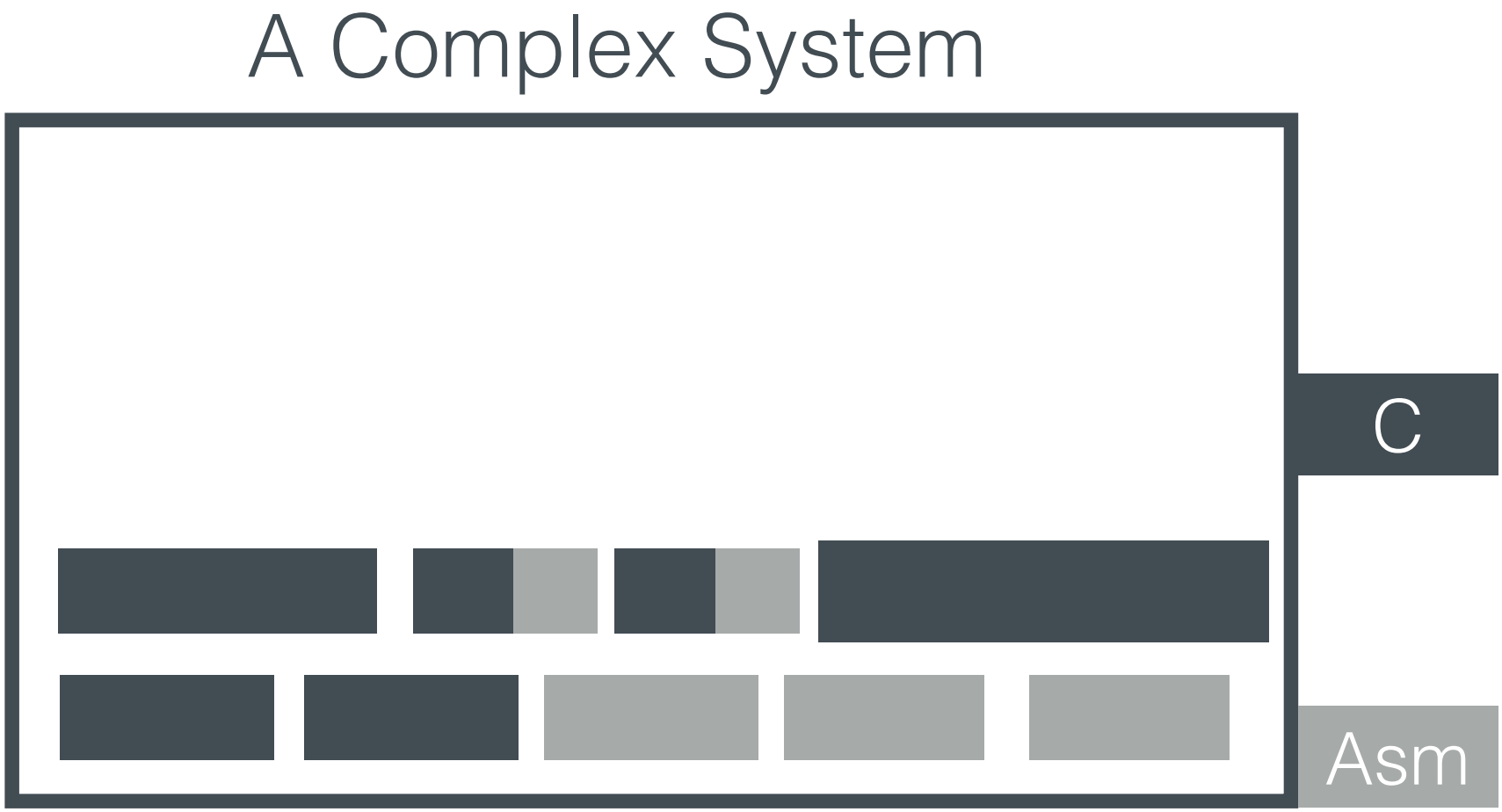


Challenges: Compositionality

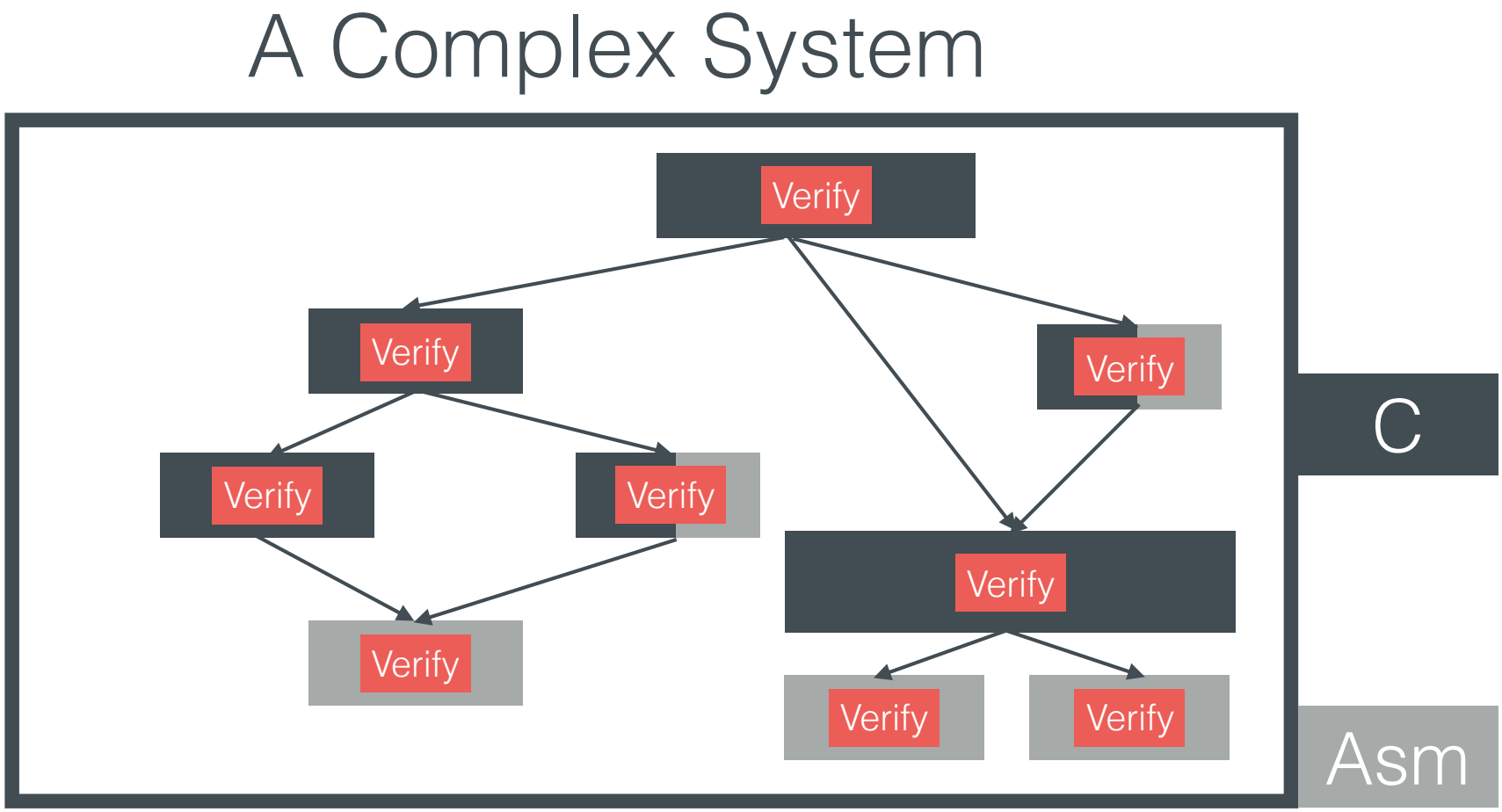


Abstraction Gap

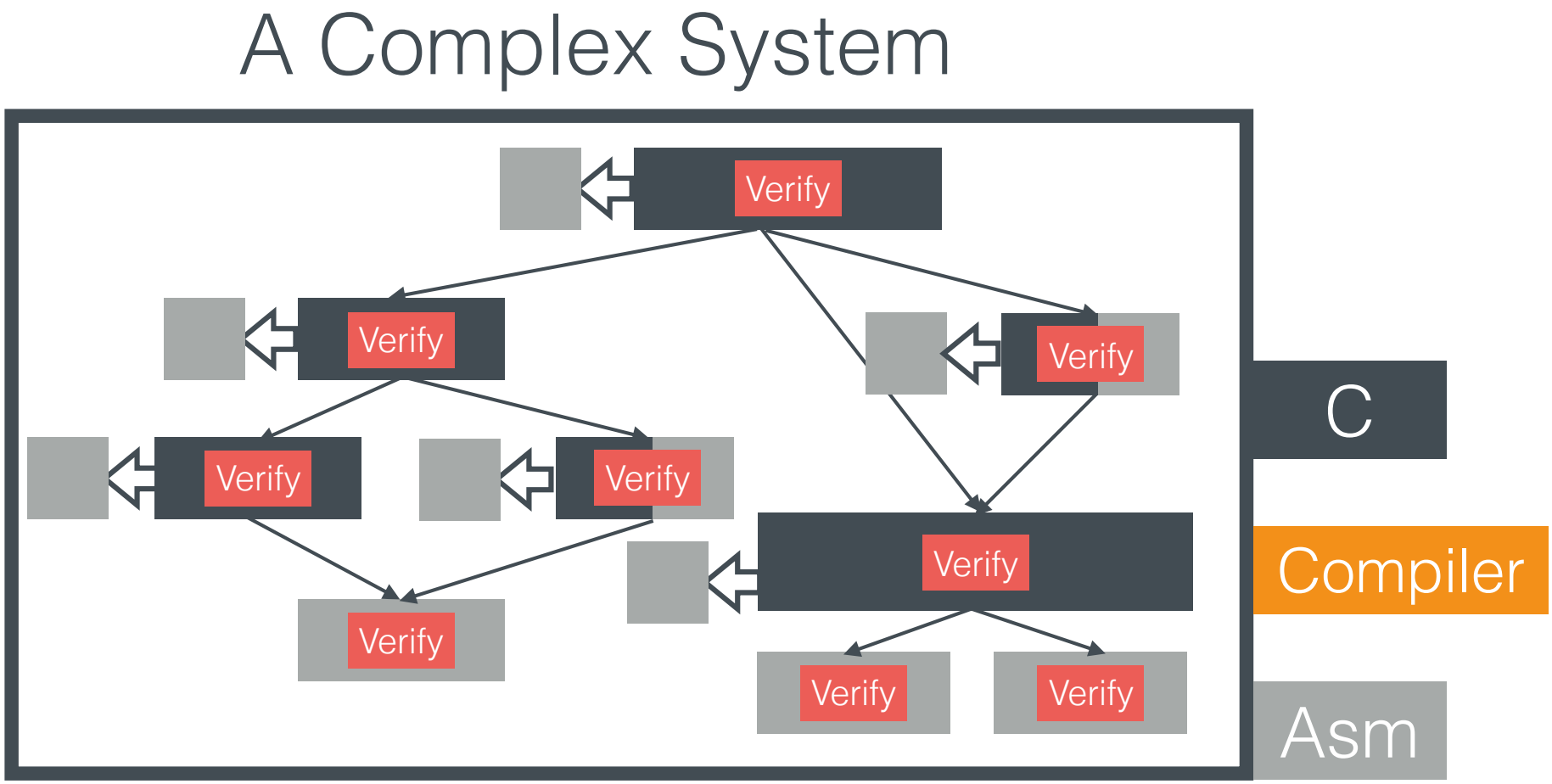
Challenges: Compositionality



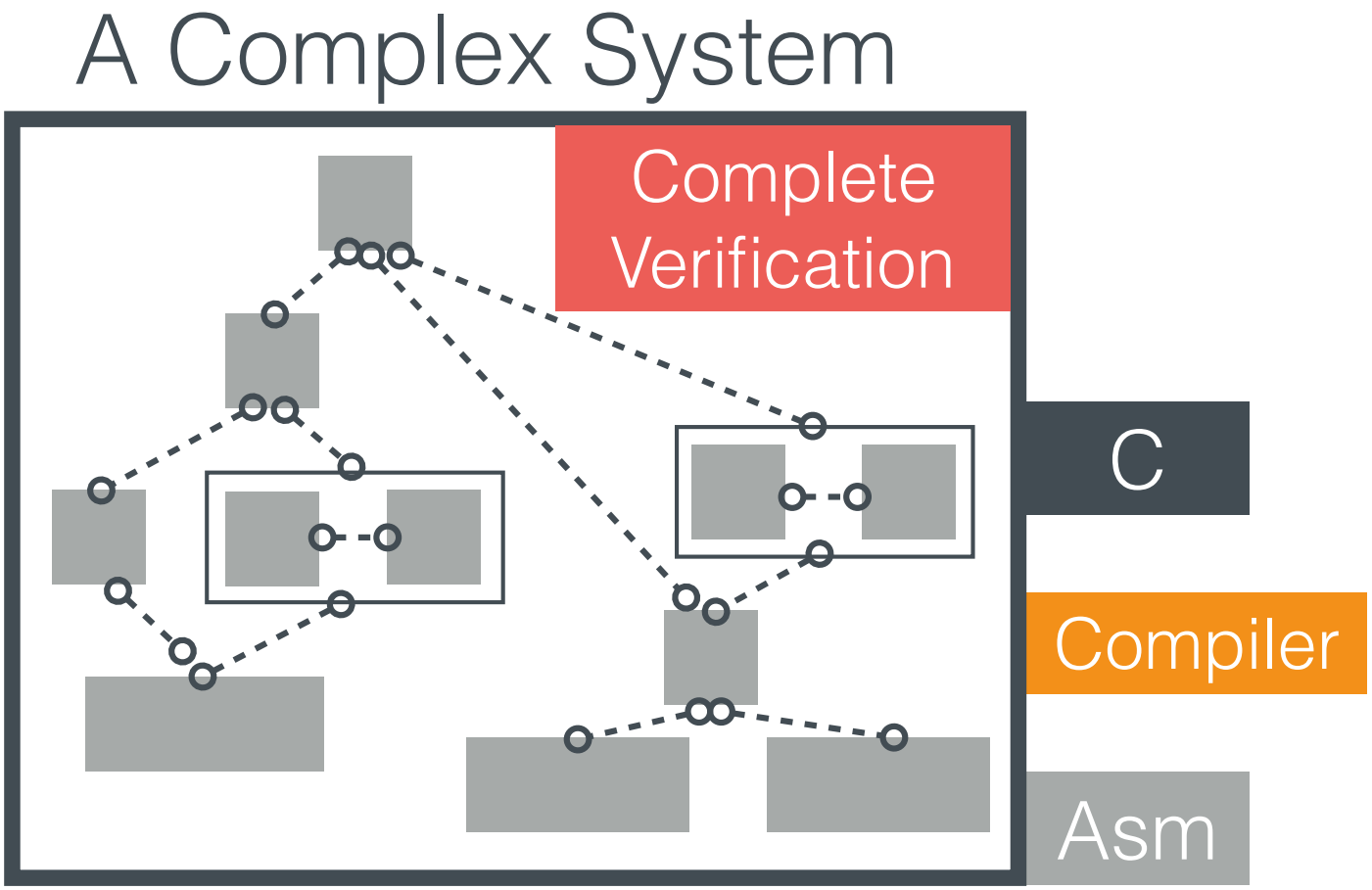
Challenges: Compositionality



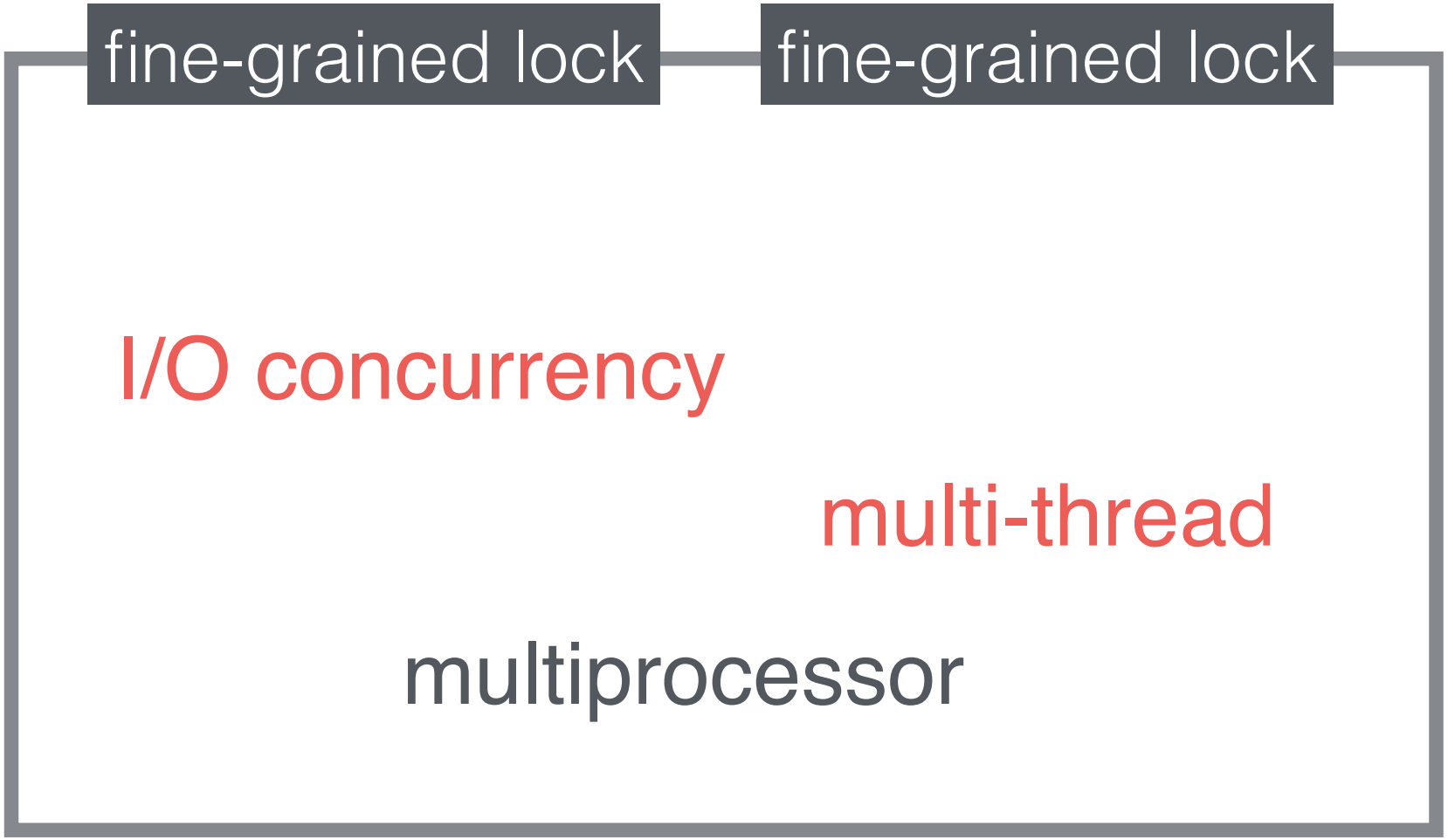
Challenges: Compositionality



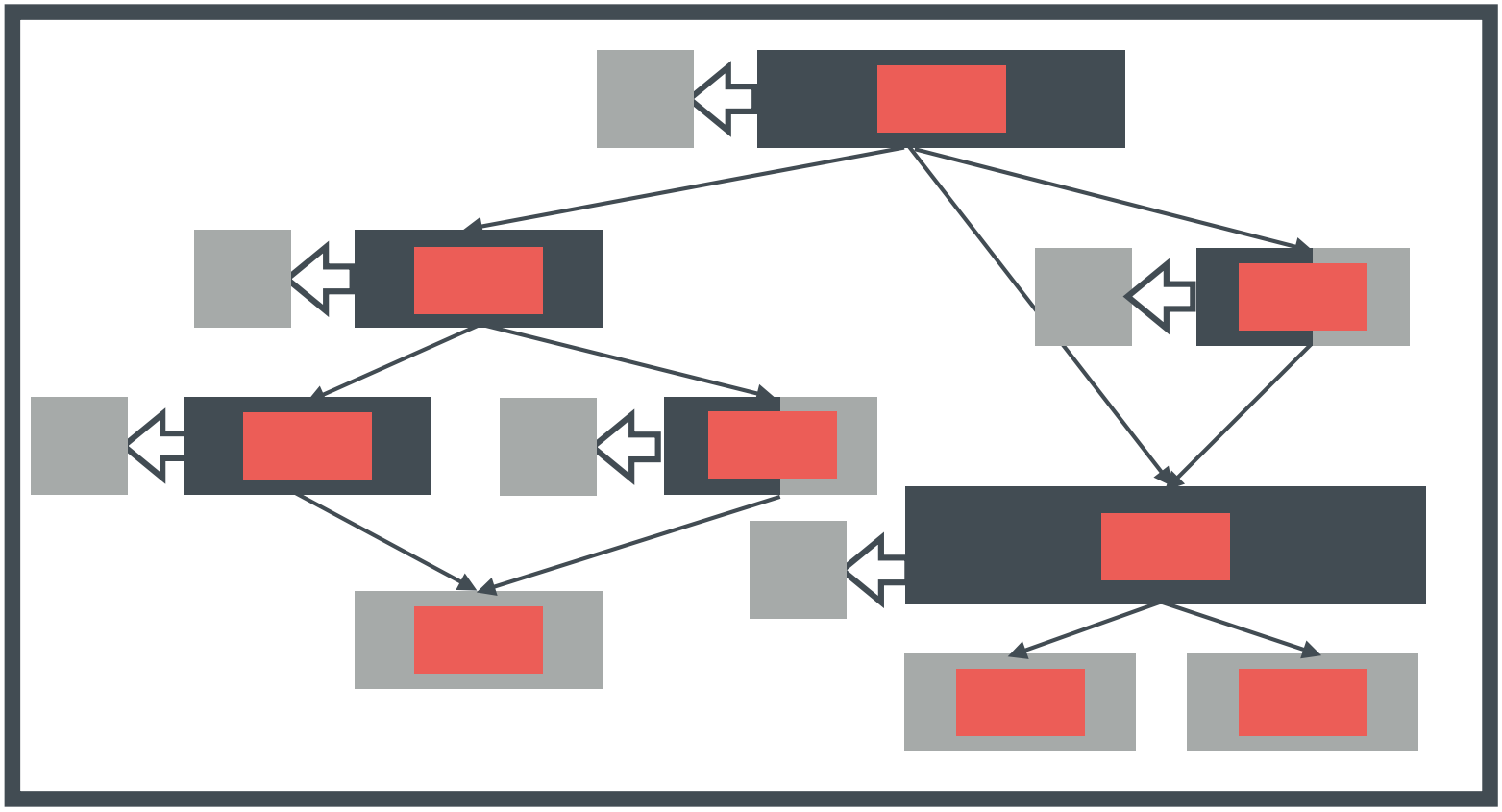
Challenges: Compositionality



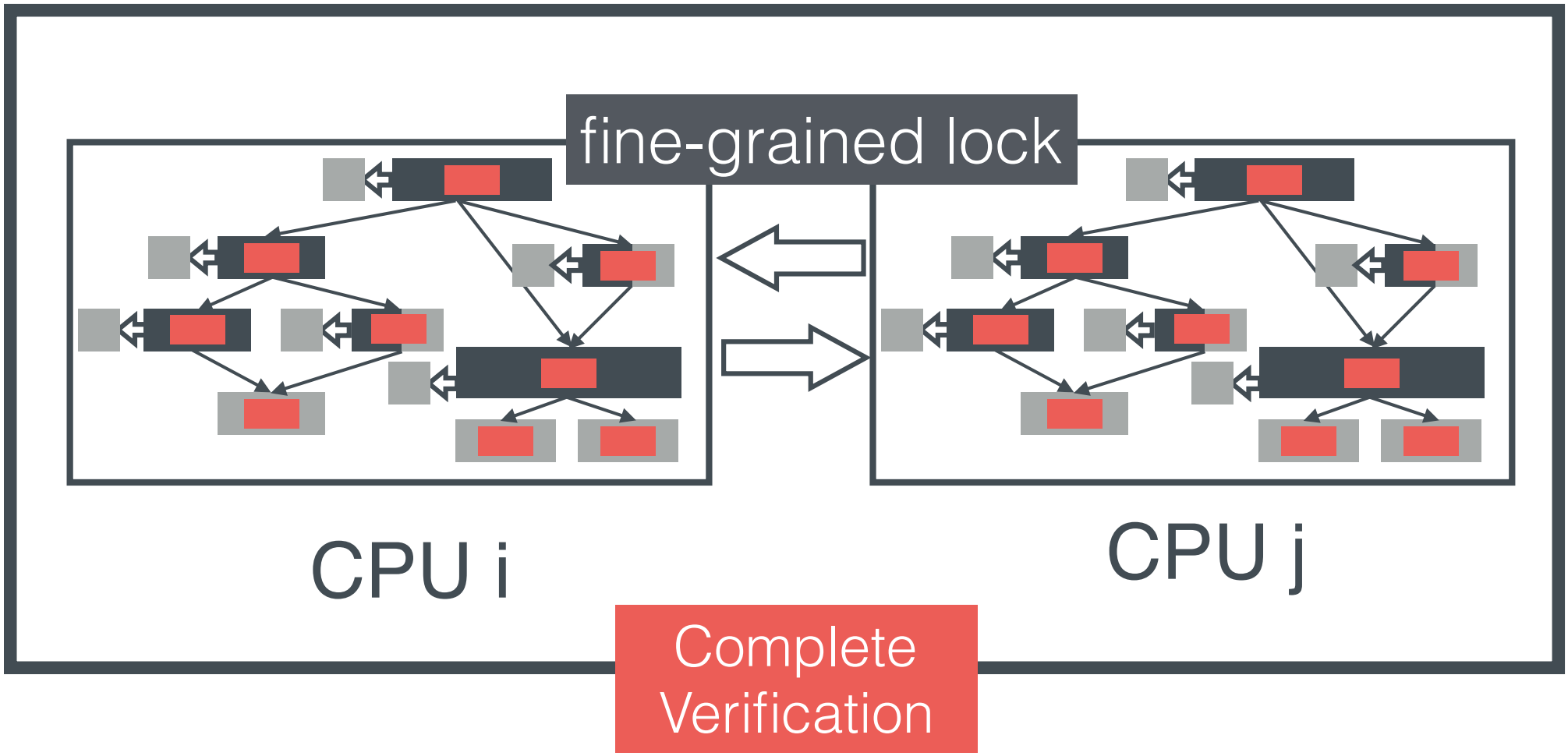
Challenges: Concurrency



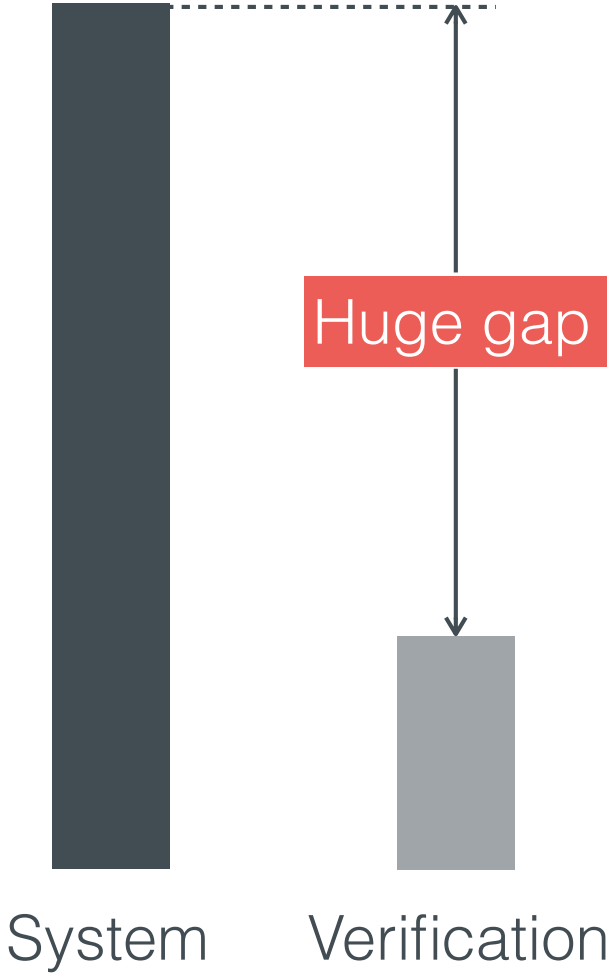
Challenges: Concurrency



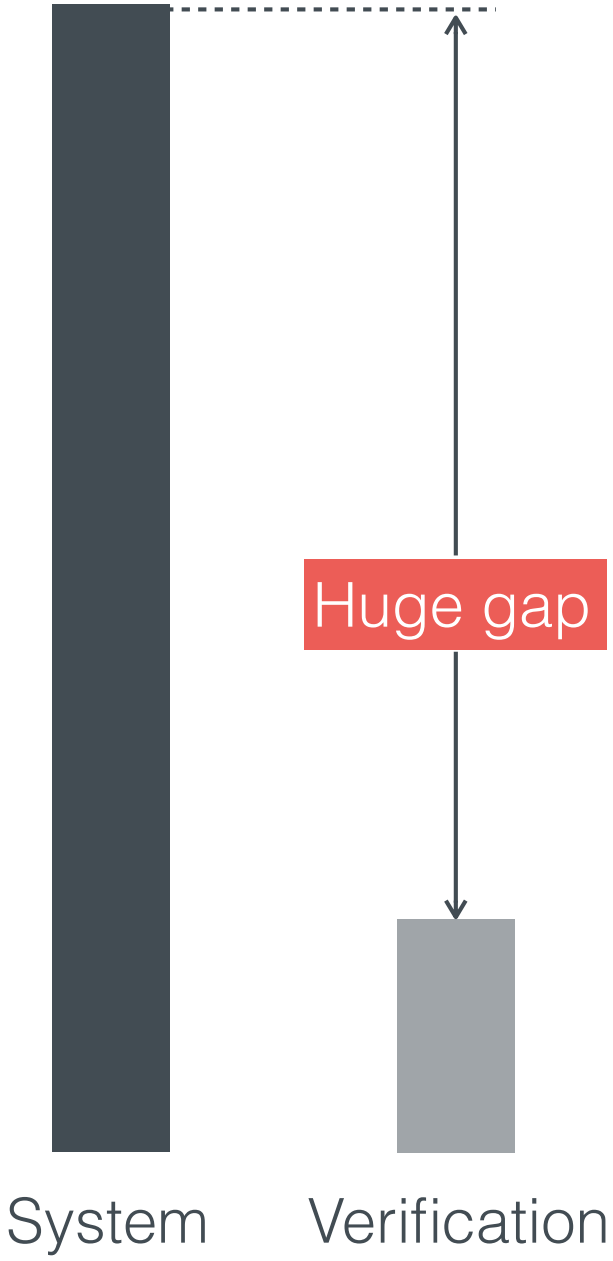
Challenges: Concurrency



Challenges: New Domain



Challenges: New Domain



Contribution

Certified Abstraction Layers

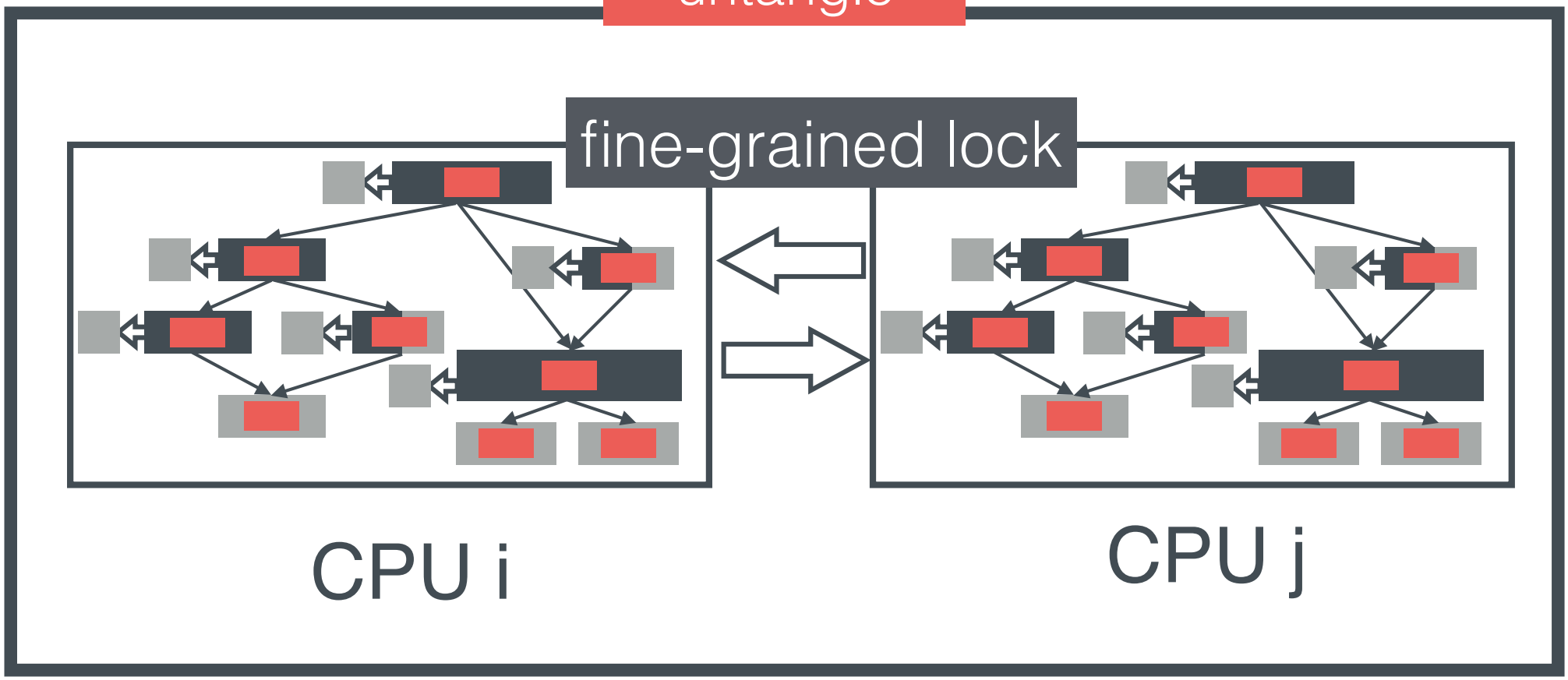
CertiKOS

aim to solve all these challenges

Contribution

Certified Abstraction Layers

untangle



Contribution

Certified Abstraction Layers

- verify existing systems
- build the next generation system software **designed** to be reliable and secure

Contribution

Certified Abstraction Layers

verify existing systems

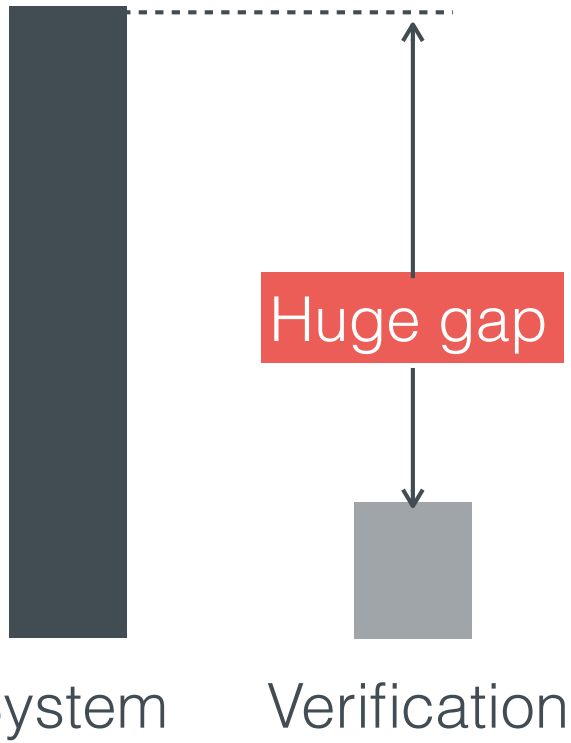
build
certified system software

Contribution

Certified Abstraction Layers

verify existing systems

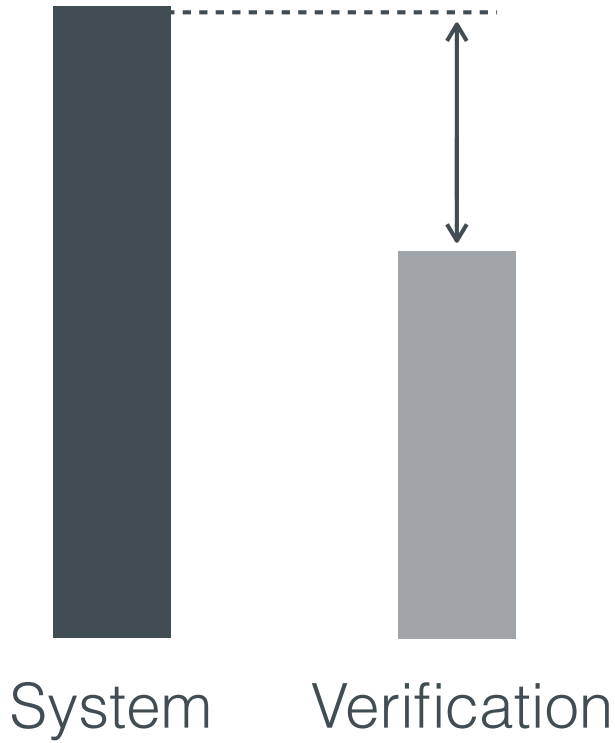
build
certified system software



Certified Abstraction Layers

verify existing systems

build
certified system software



Contribution

Certified Abstraction Layers

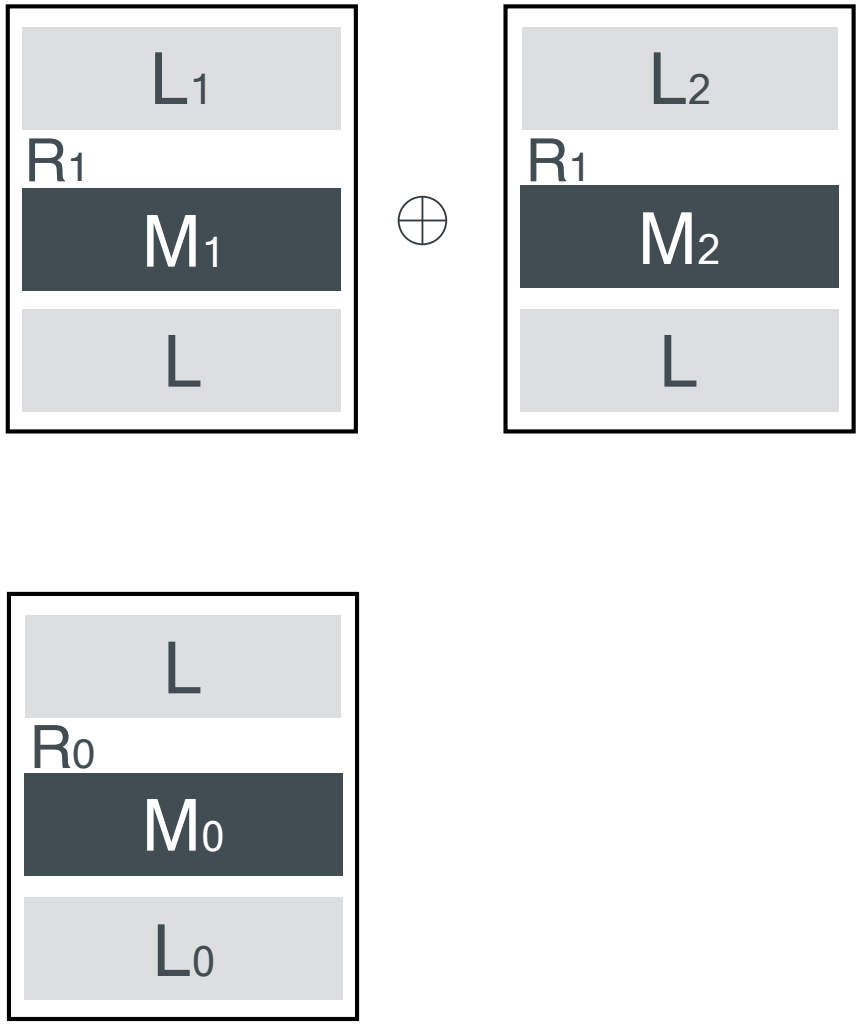
verify existing systems

build
certified system software

Certified System Software

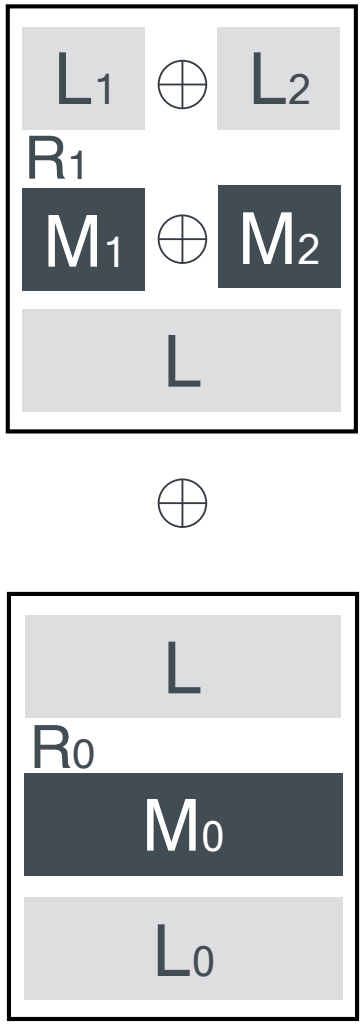
Contribution

Certified Abstraction Layers

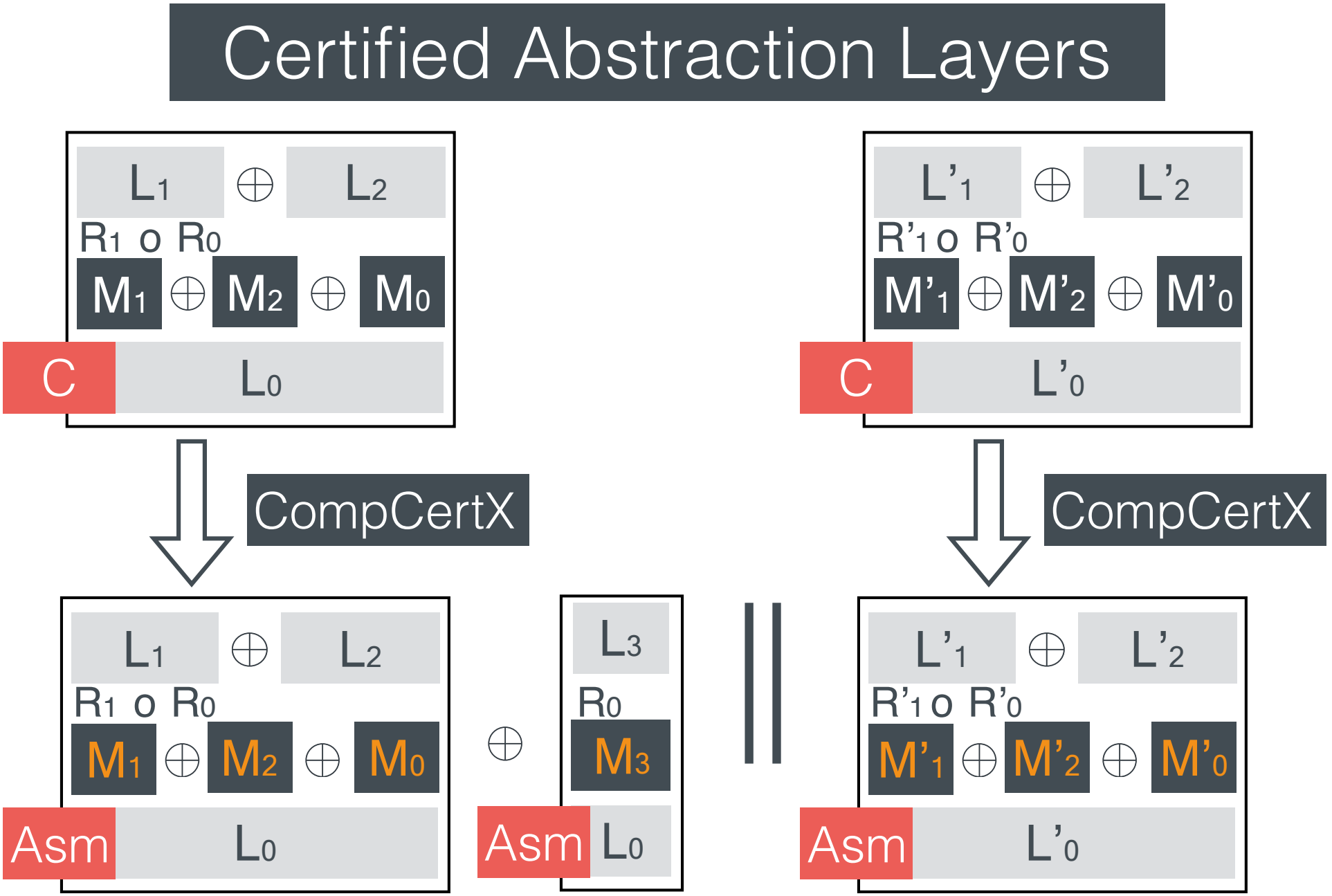


Contribution

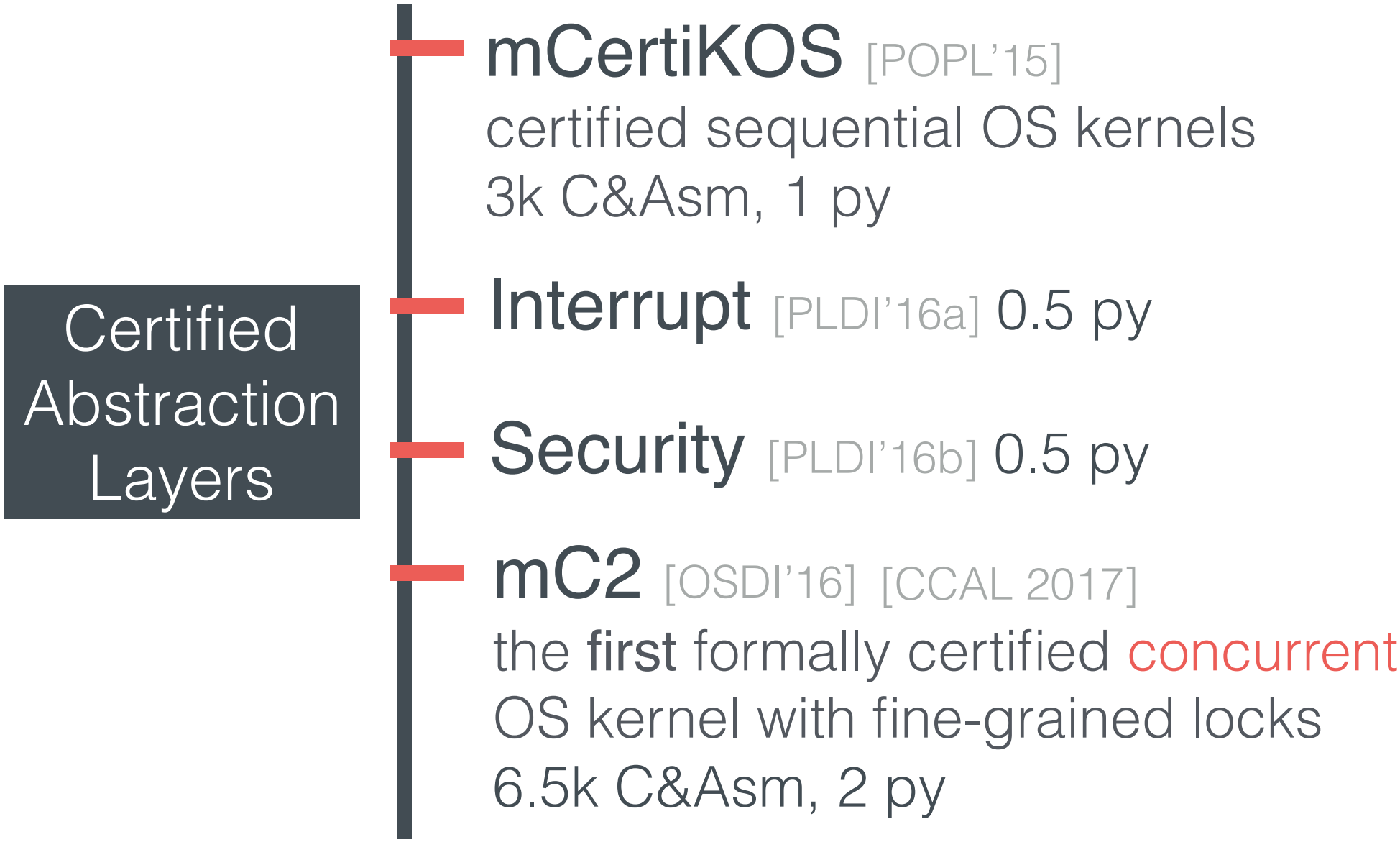
Certified Abstraction Layers



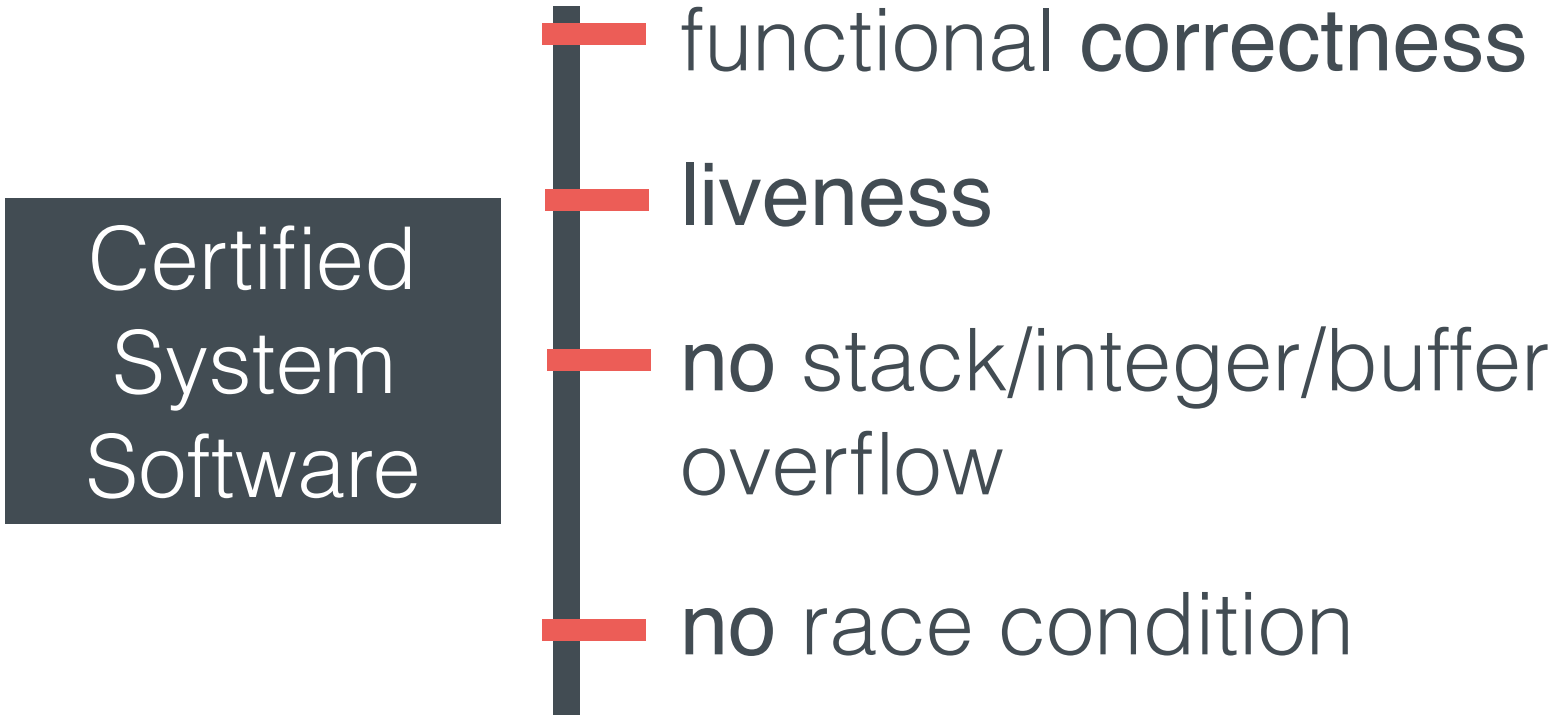
Contribution



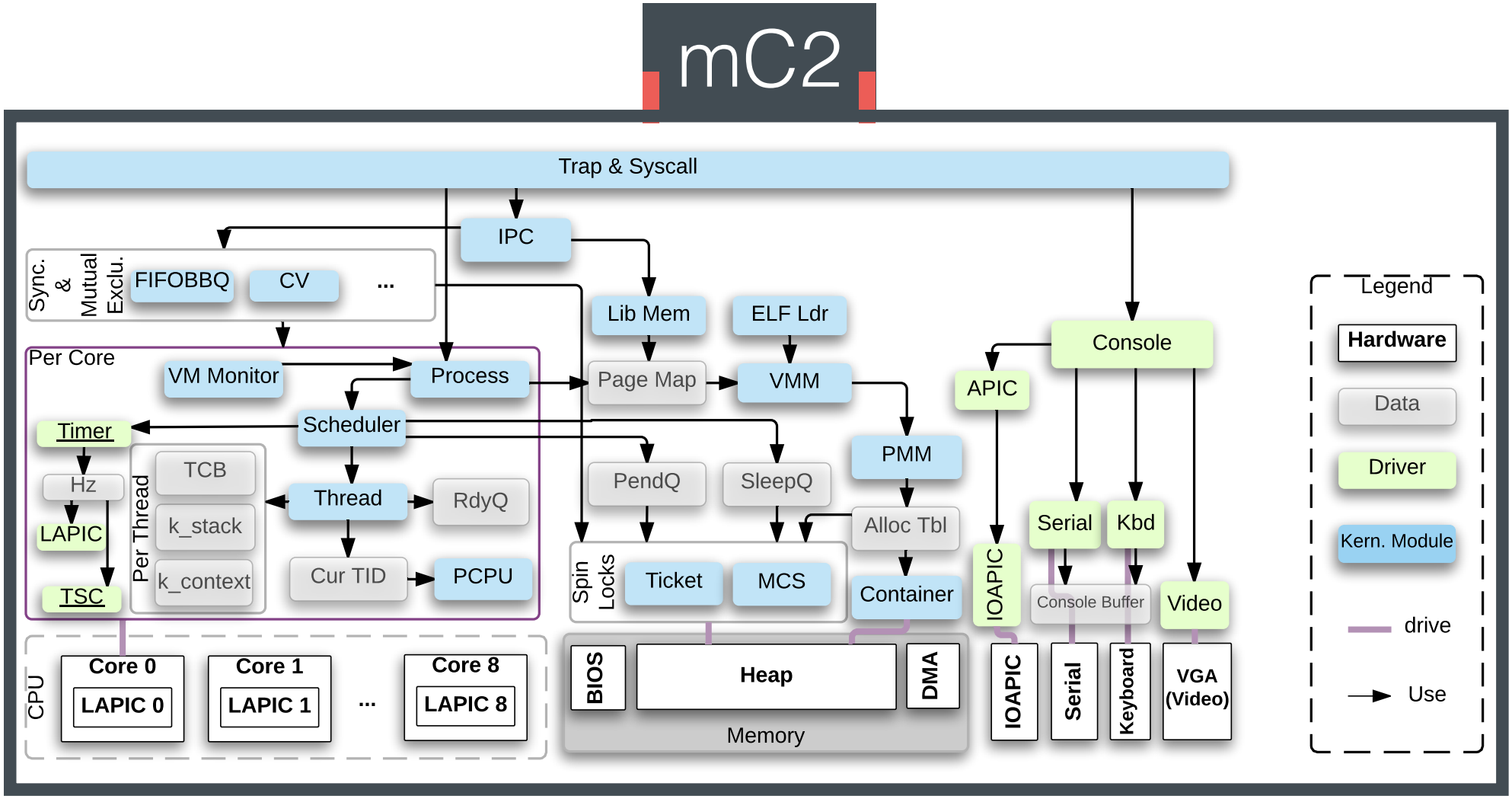
Contribution



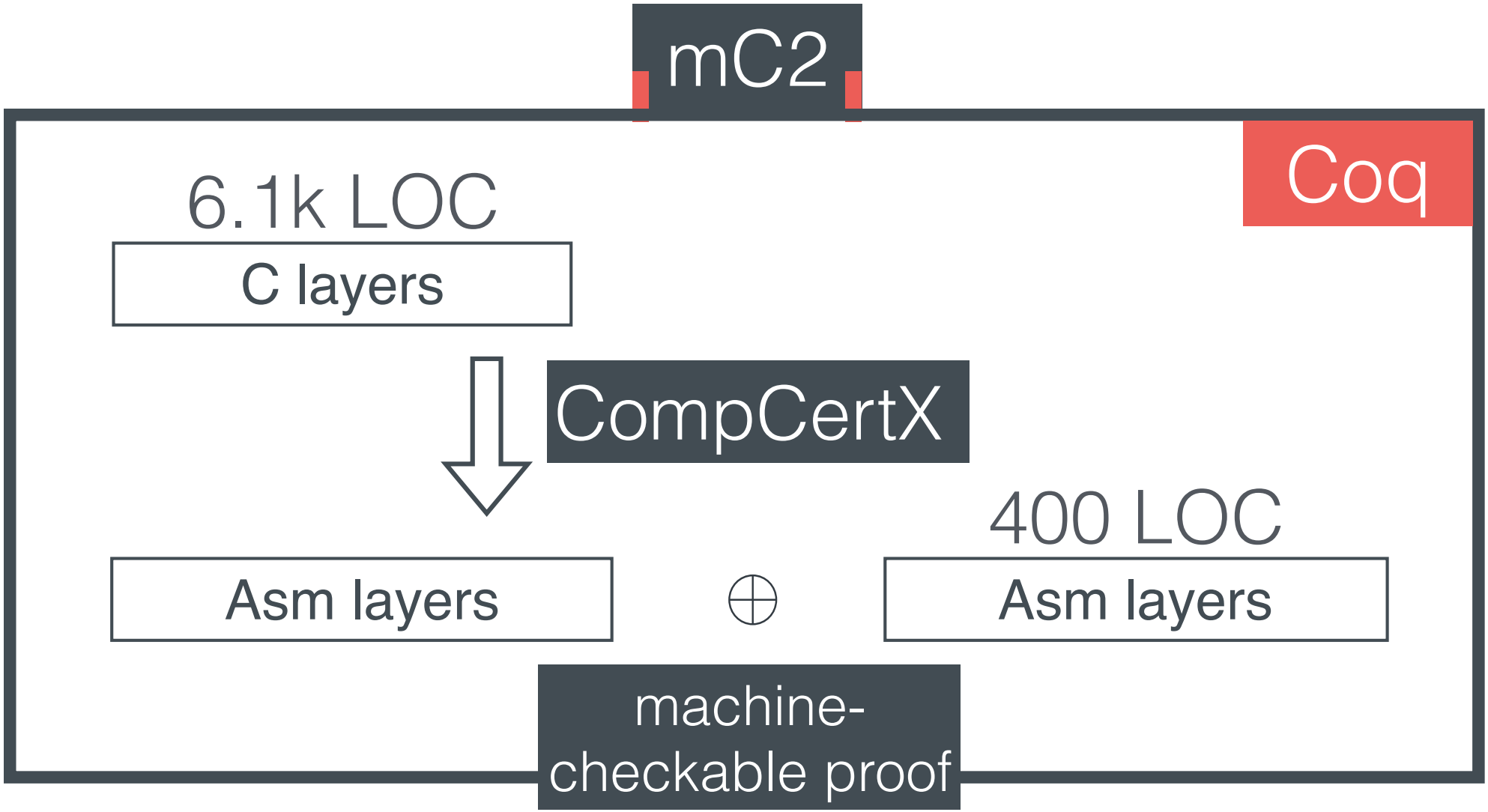
Contribution



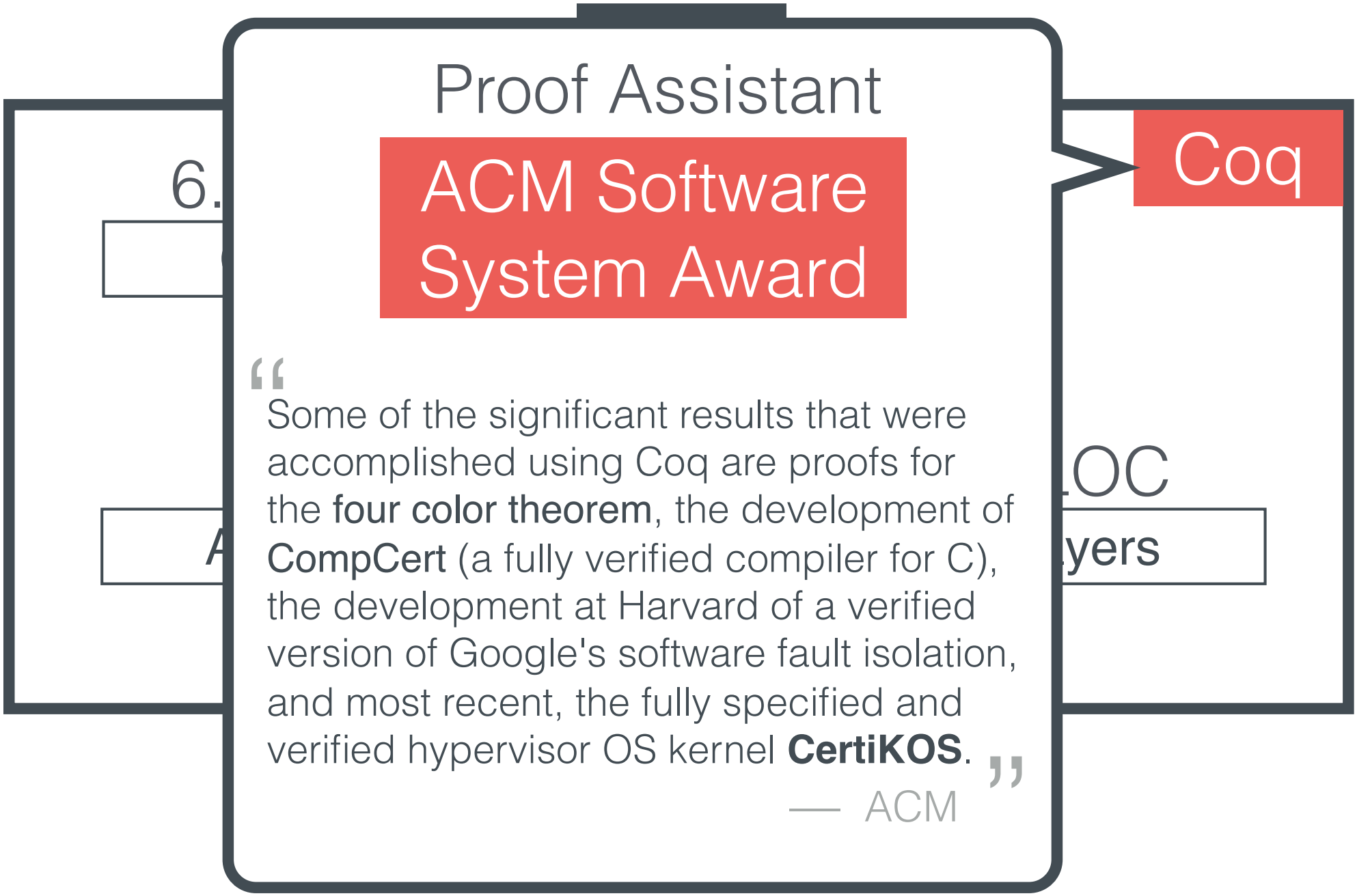
Contribution



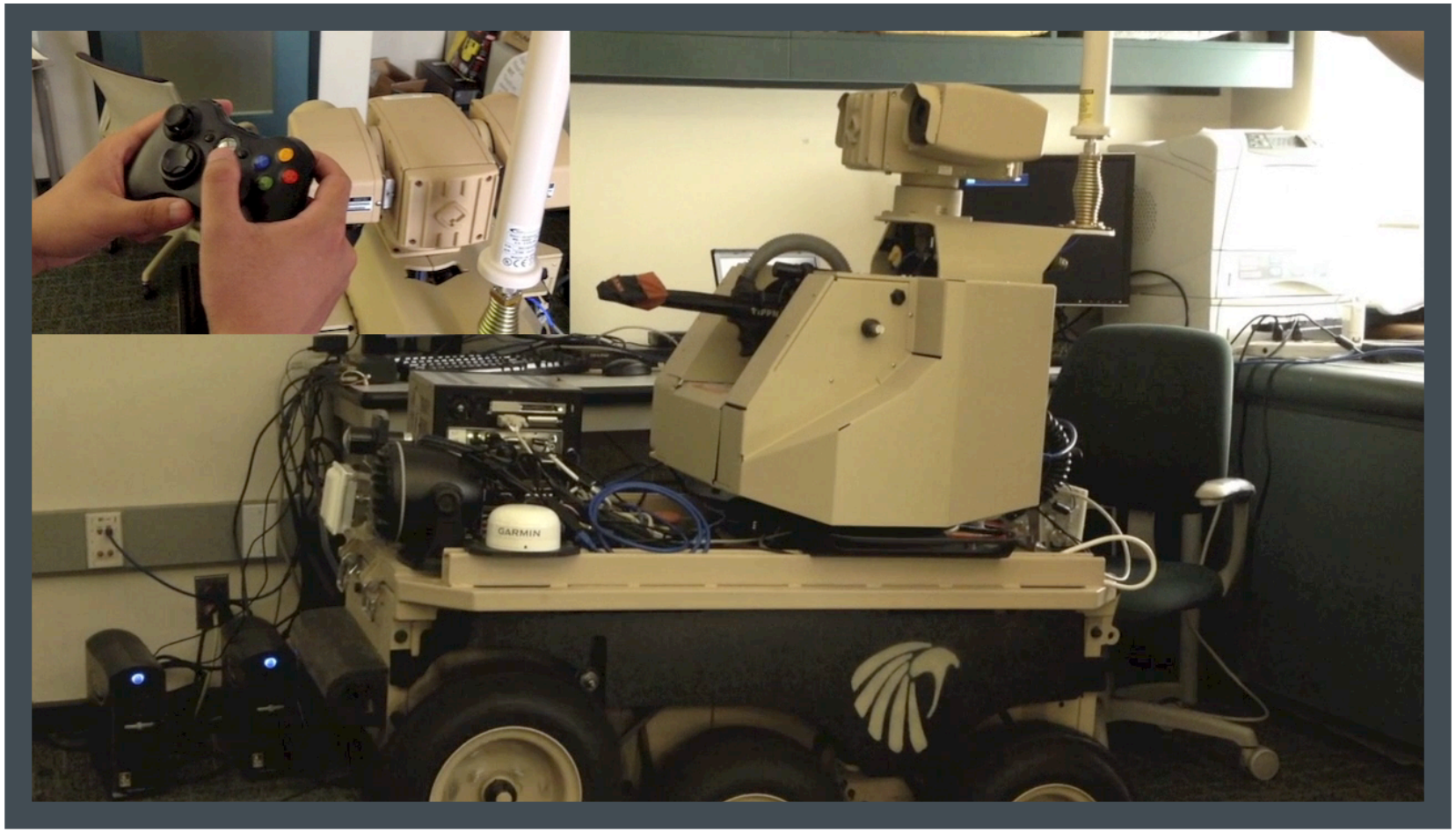
Contribution



Contribution

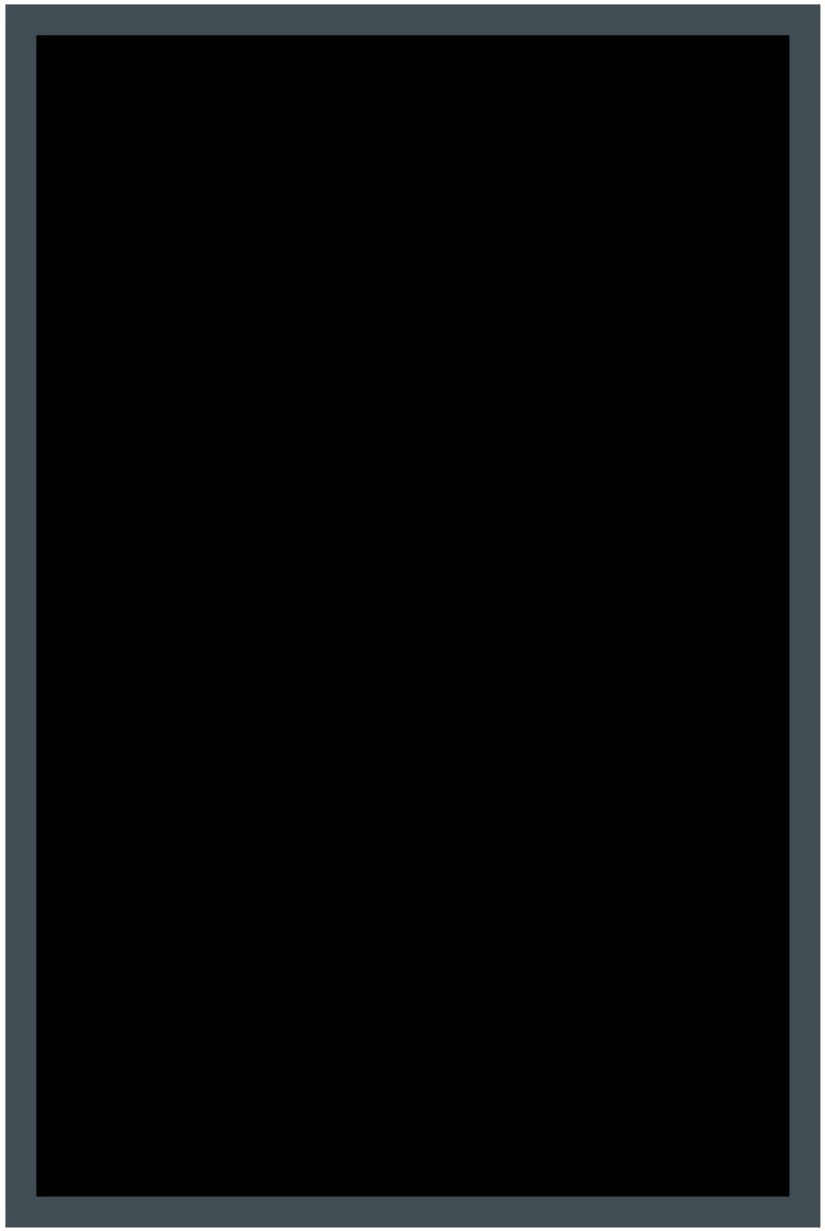


Deployment



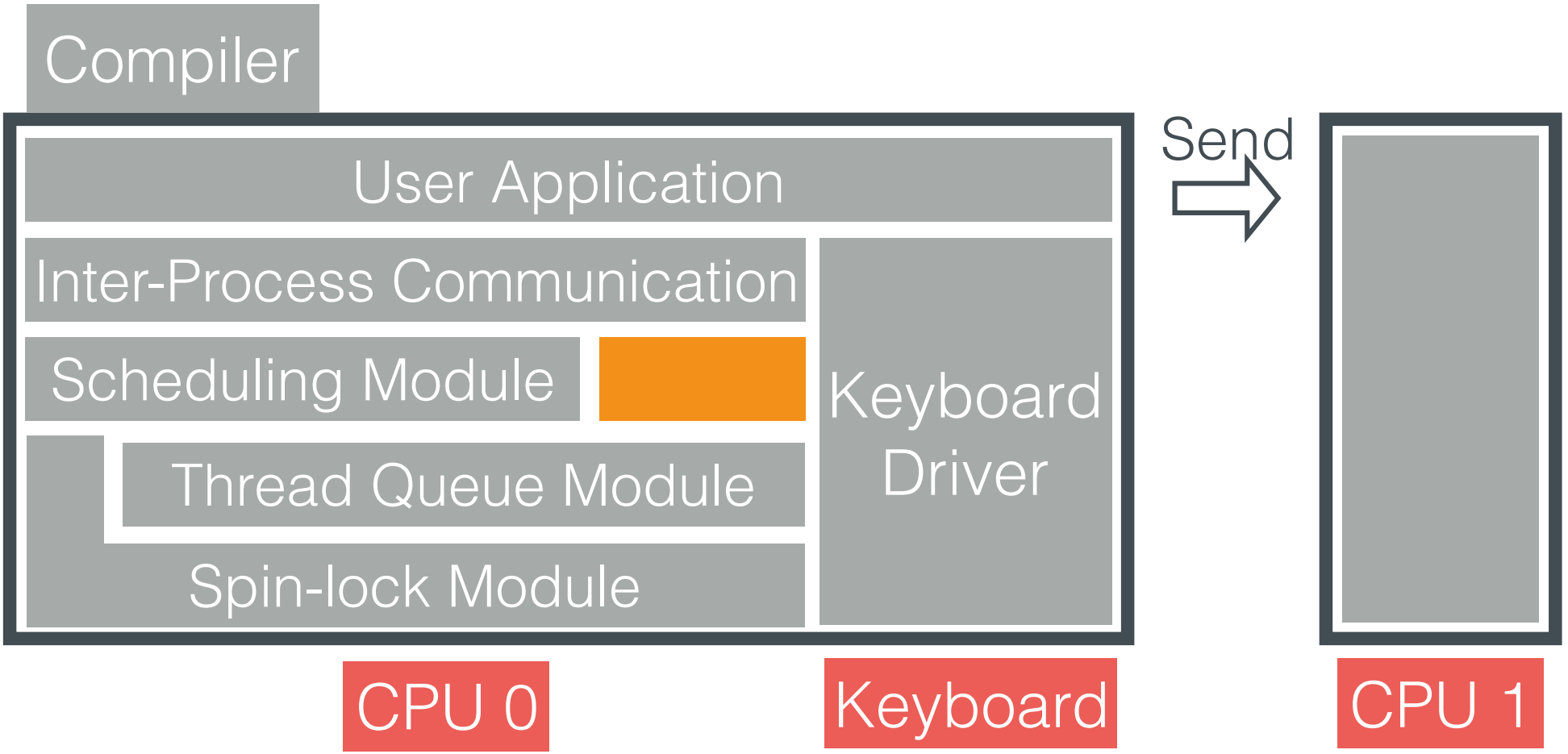
CertiKOS on **Landshark**, DARPA HACMS

Deployment

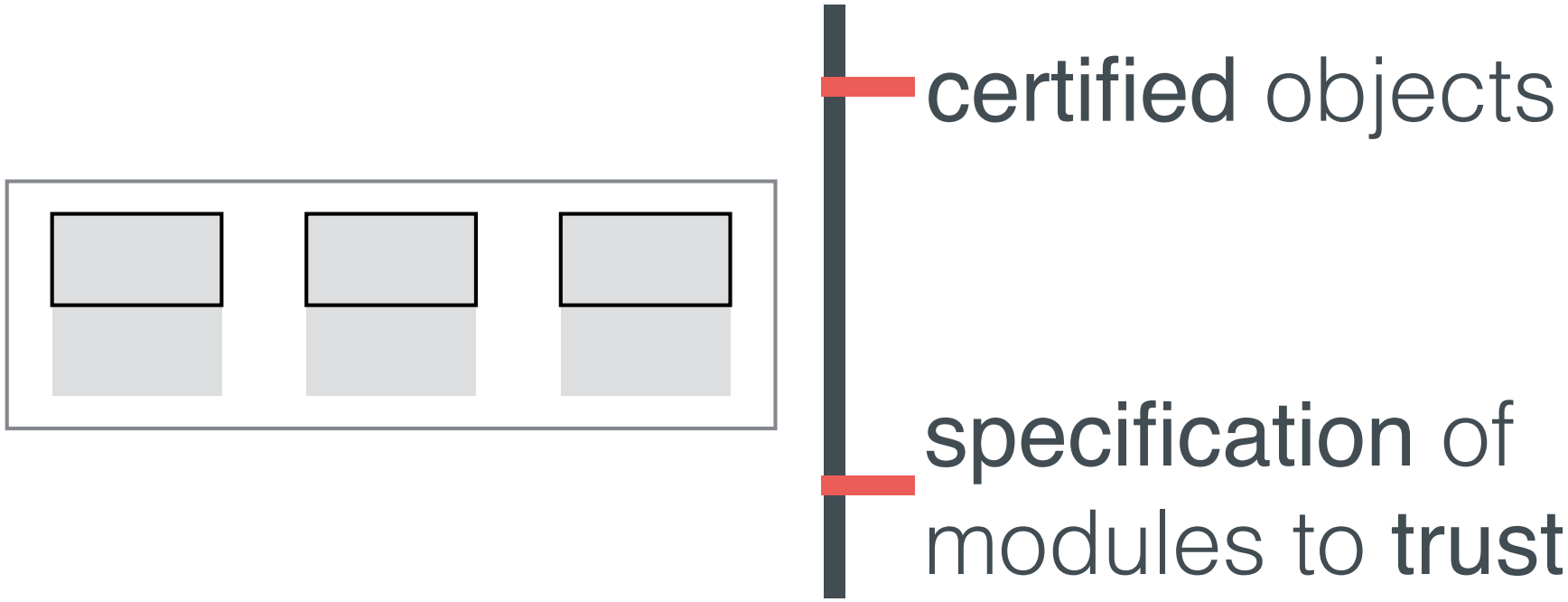


CertiKOS on
Quadcopter

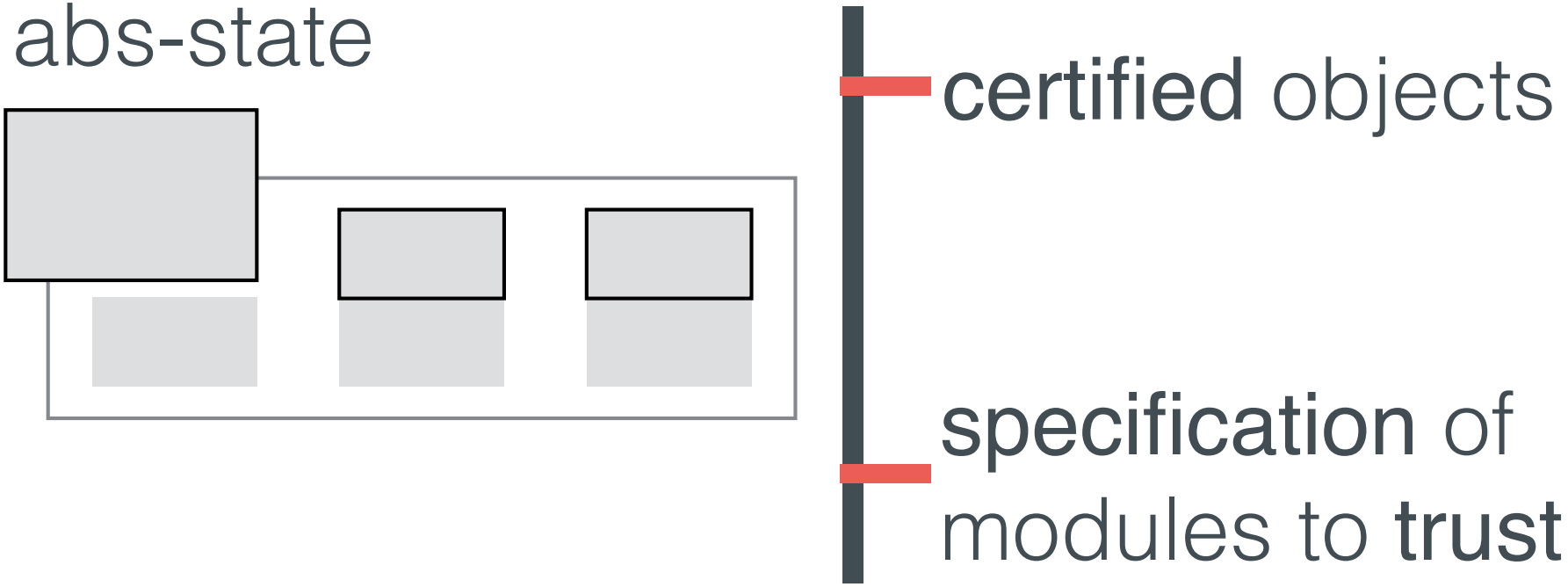
Build a Certified System



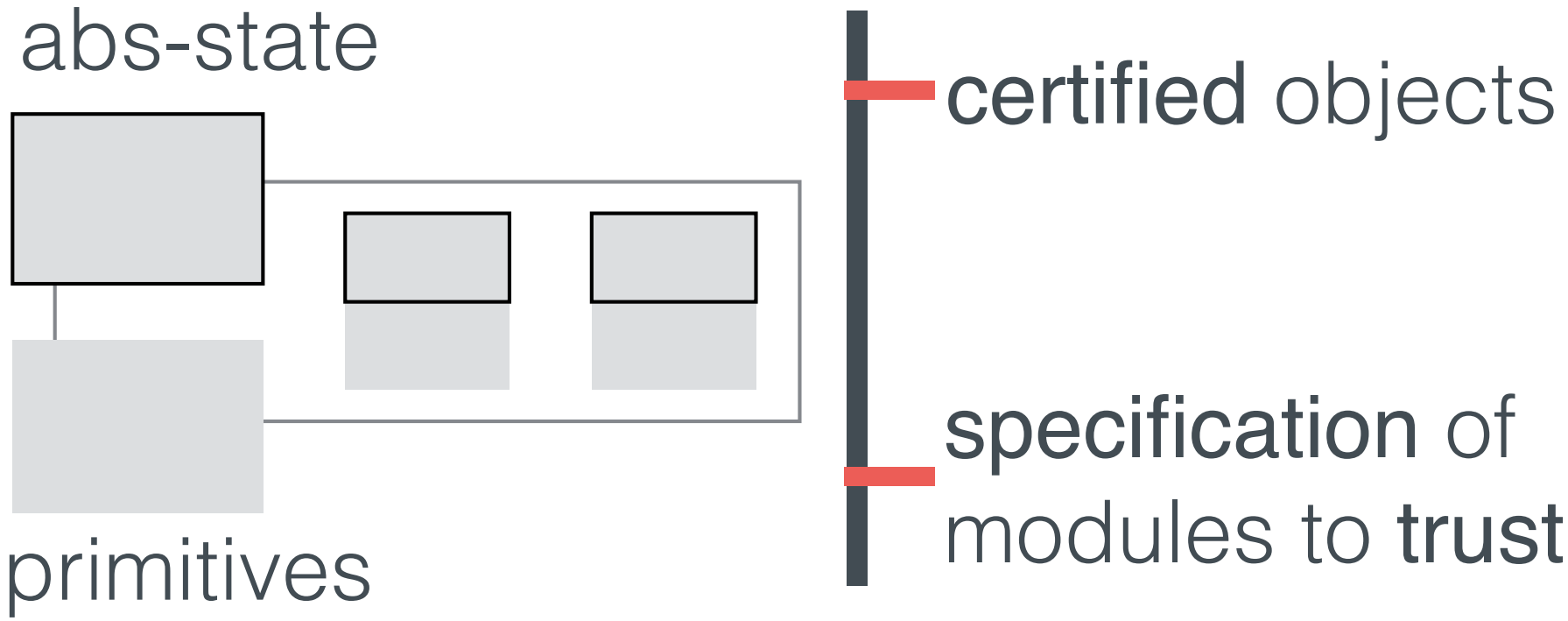
Certified Sequential Layer [POPL'15]



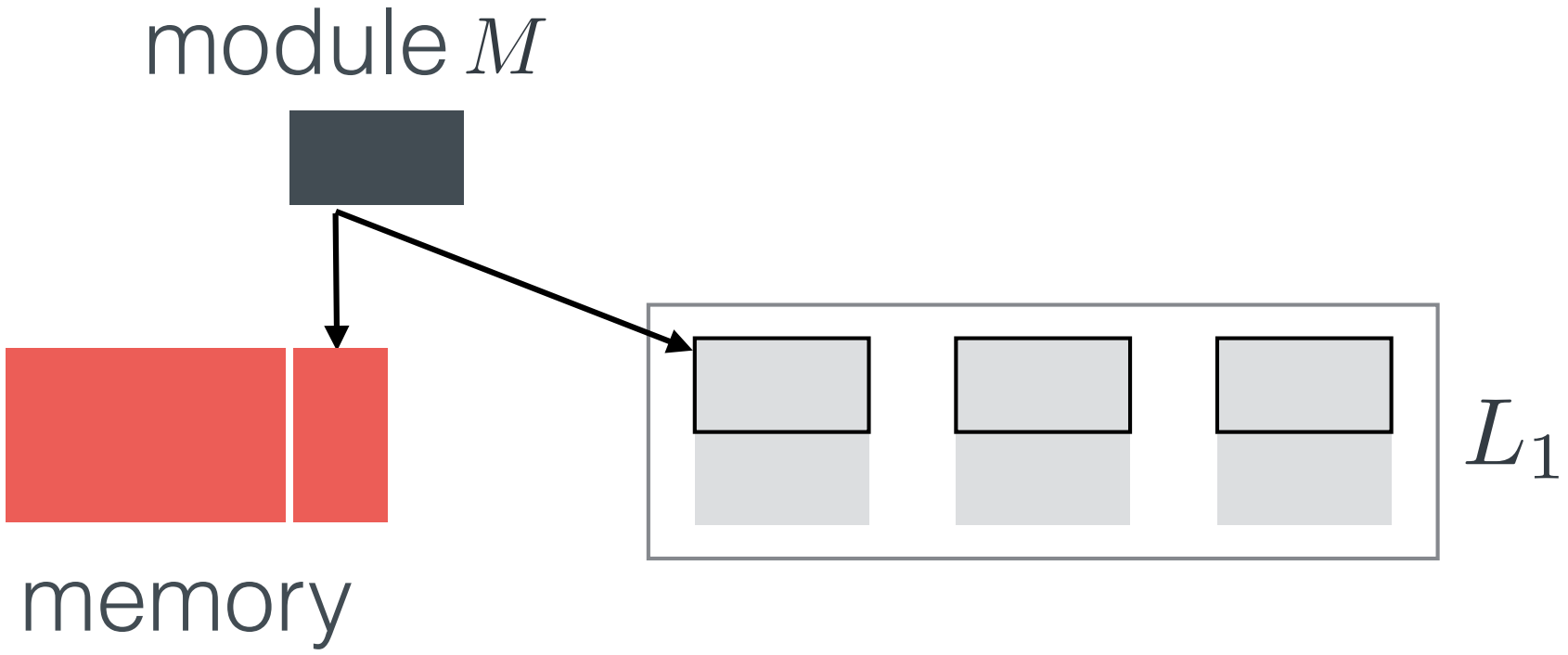
Certified Sequential Layer [POPL'15]



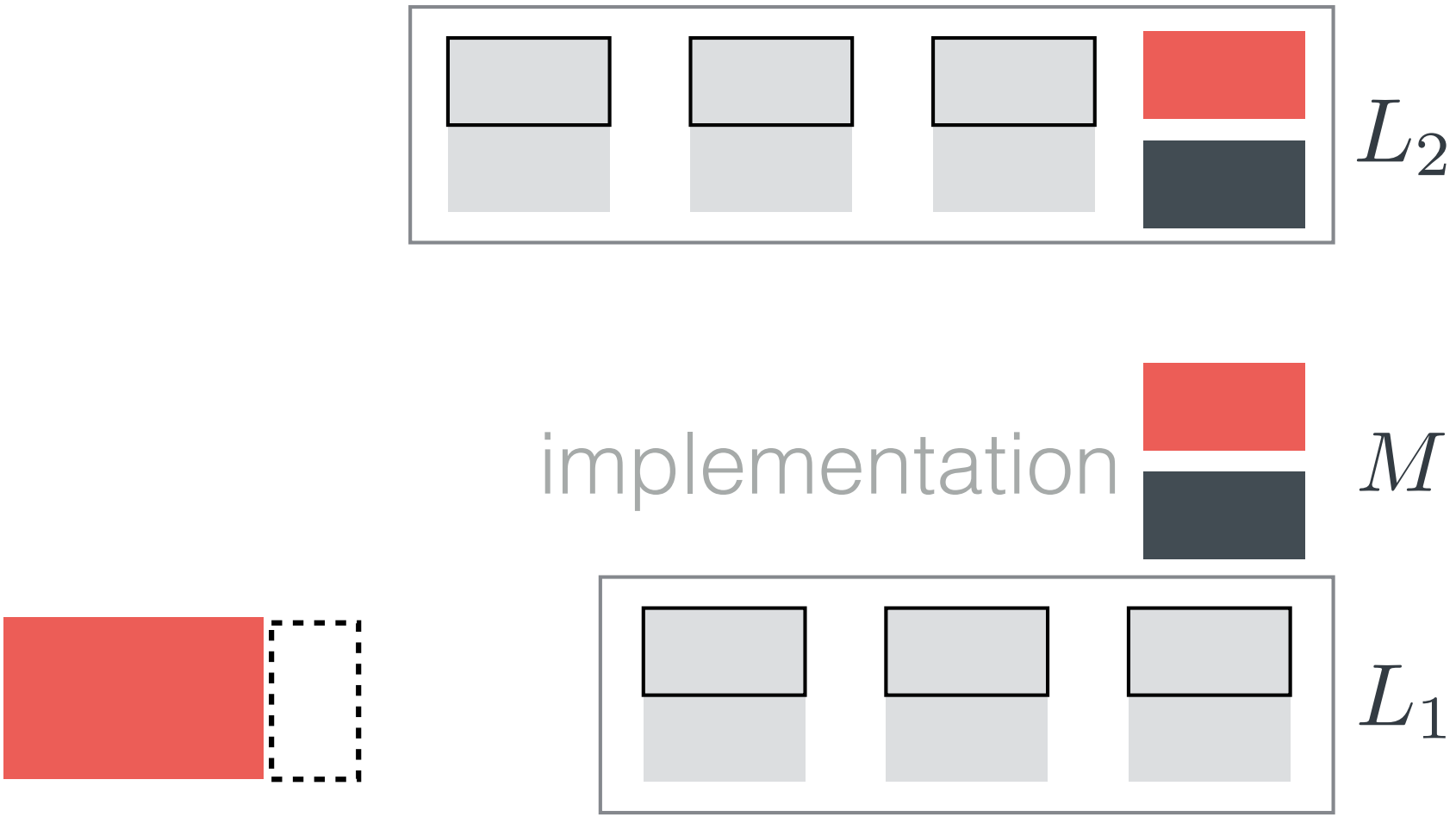
Certified Sequential Layer [POPL'15]



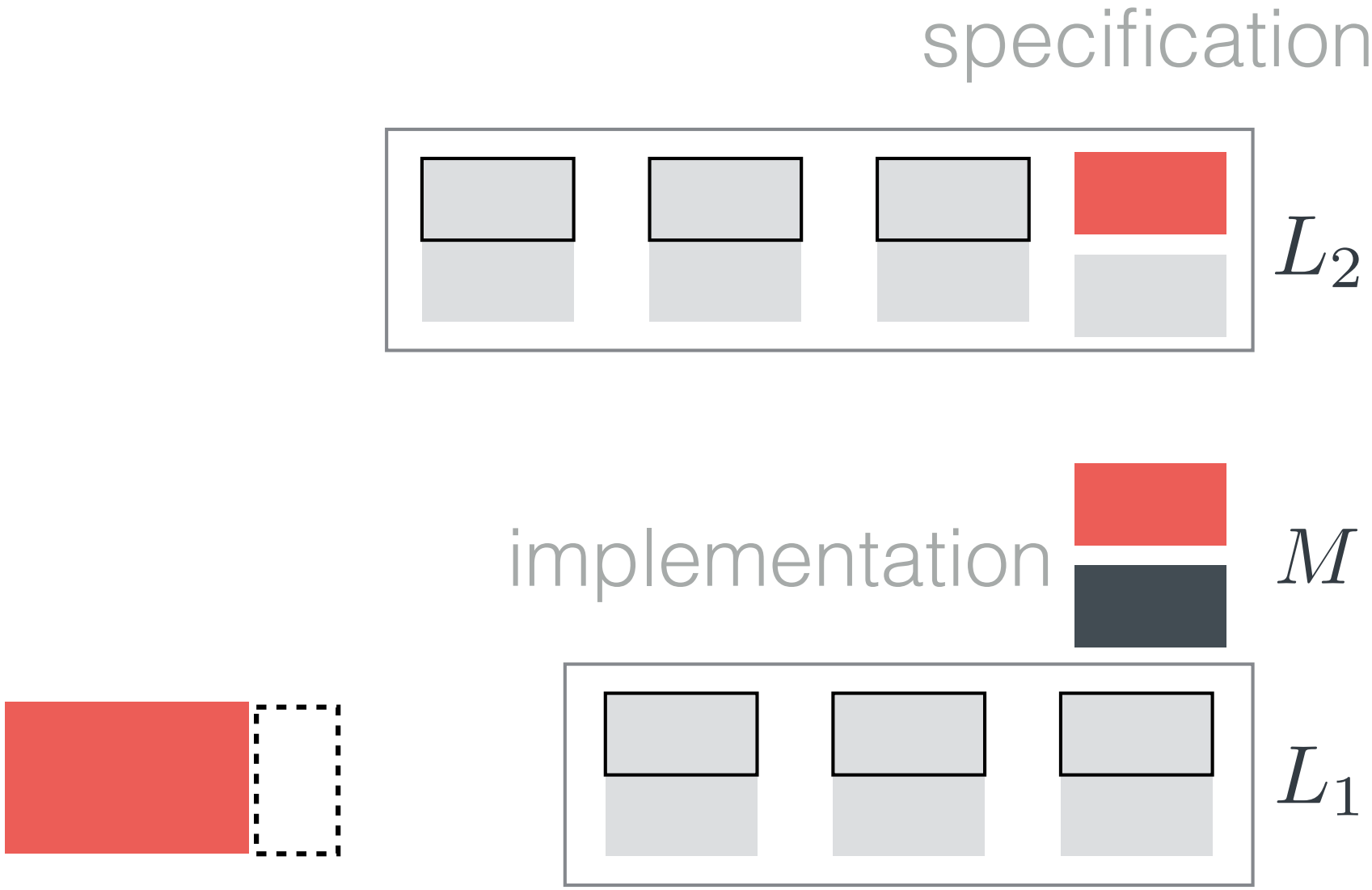
Certified Sequential Layer



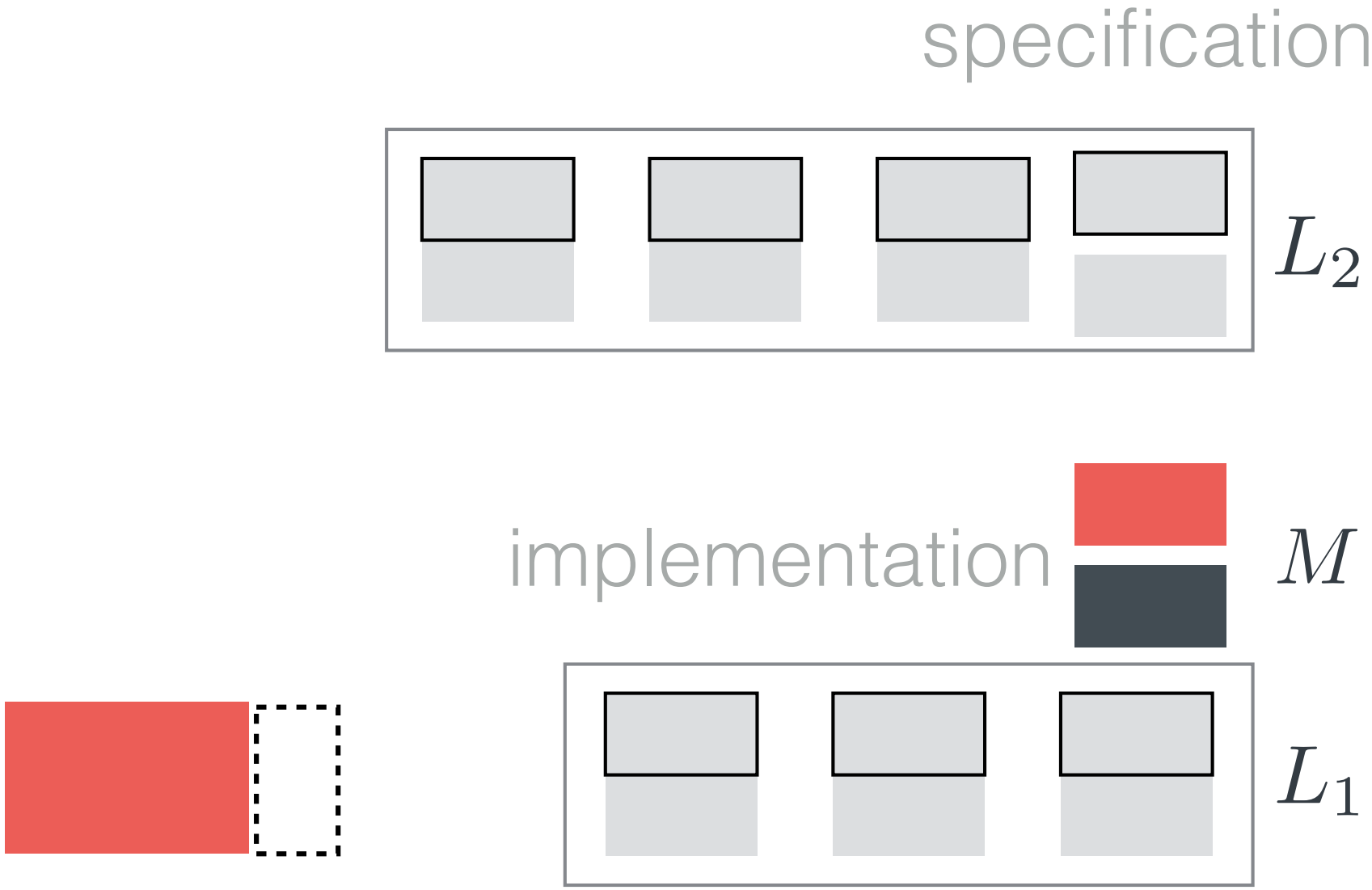
Certified Sequential Layer



Certified Sequential Layer



Certified Sequential Layer



Example: Thread Queue

```
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];
```

```
typedef struct tdq {
    tcb *head, *tail;
} tdq;

tdq* td_queue;
```

C

M

implementation

Example: Thread Queue

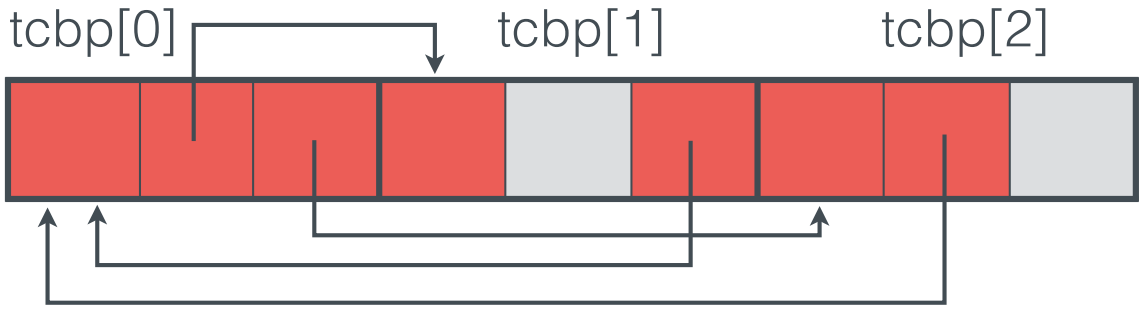
```
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];
```

```
typedef struct tdq {
    tcb *head, *tail;
} tdq;

tdq* td_queue;
```

C

M

implementation



Example: Thread Queue

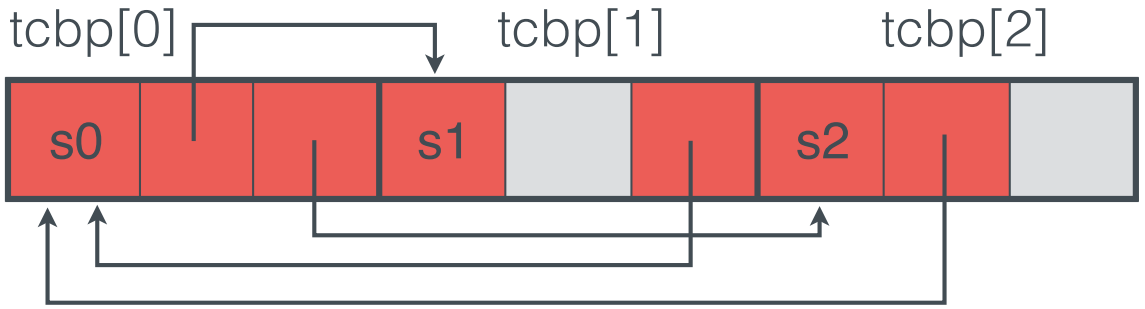
```
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];
```

```
typedef struct tdq {
    tcb *head, *tail;
} tdq;

tdq* td_queue;
```

C

M

implementation

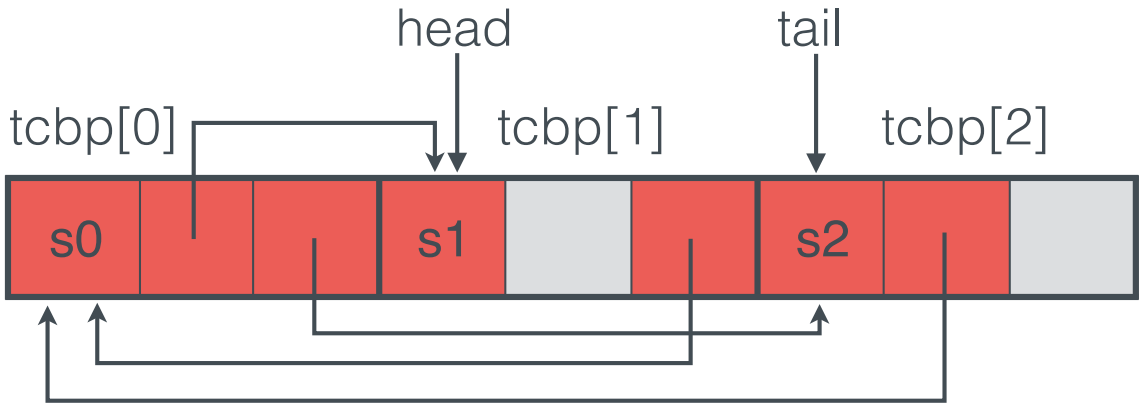


Example: Thread Queue

```
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];
```

```
typedef struct tdq {
    tcb *head, *tail;
} tdq;

tdq* td_queue;
```

C



M

implementation

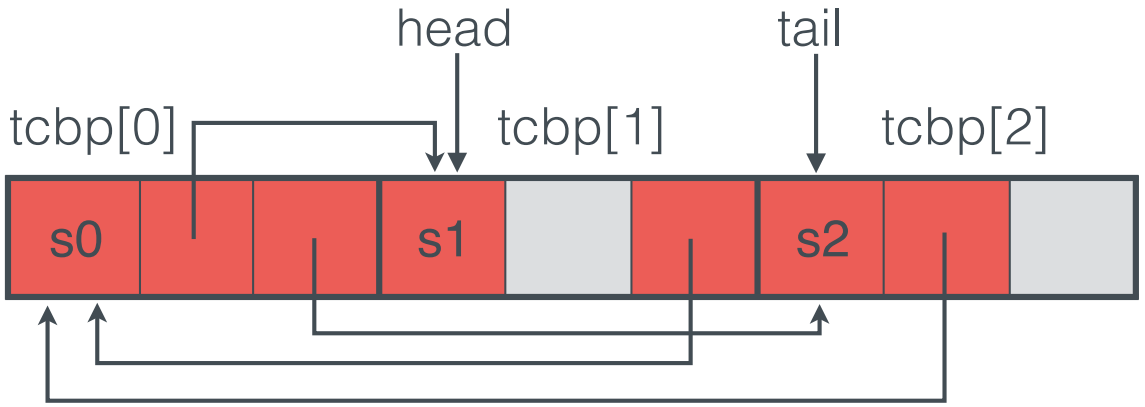
Example: Thread Queue

```
tcb* dequeue(tdq* q) {
    tcb *head, *next;
    tcb *i = null;
    if (!q) return i;
    head = q -> head;
    if (!head) return i;
    i = head;
    next = i -> next;
    if (!next) {
        q -> head = null;
        q -> tail = null;
    } else {
        next -> prev = null;
        q -> head = next;
    }
    return i;
}
```

C

M

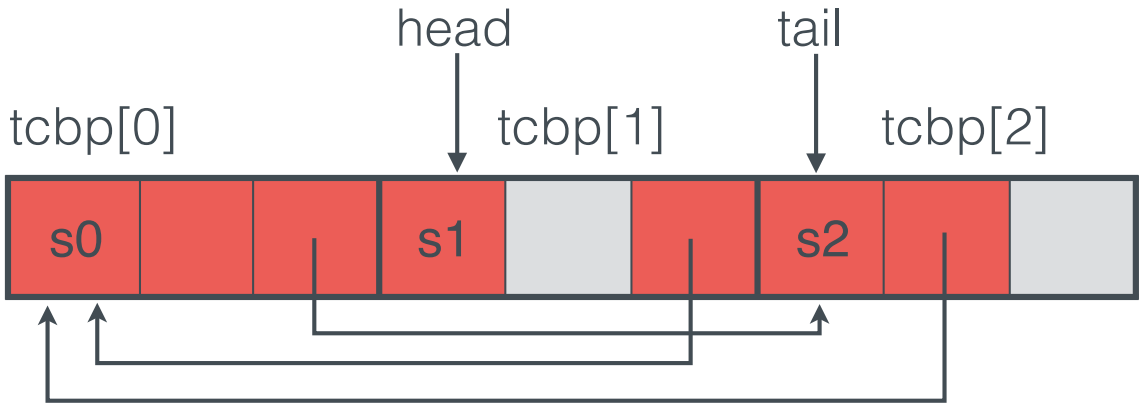
implementation



Example: Thread Queue

```
tcb* dequeue(tdq* q) {
    tcb *head, *next;
    tcb *i = null;
    if (!q) return i;
    head = q -> head;
    if (!head) return i;
    i = head;
    next = i -> next;
    if (!next) {
        q -> head = null;
        q -> tail = null;
    } else {
        next -> prev = null;
        q -> head = next;
    }
    return i;
}
```

C



M

implementation

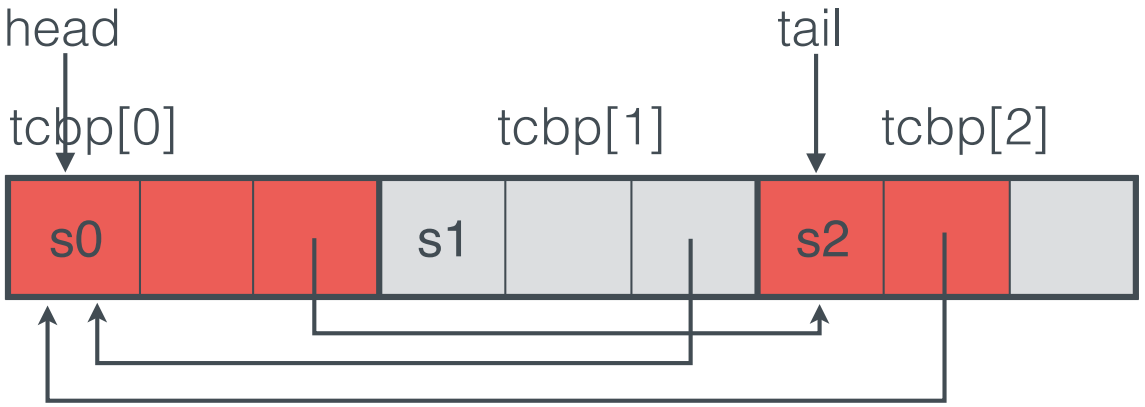
Example: Thread Queue

```
tcb* dequeue(tdq* q) {
    tcb *head, *next;
    tcb *i = null;
    if (!q) return i;
    head = q -> head;
    if (!head) return i;
    i = head;
    next = i -> next;
    if (!next) {
        q -> head = null;
        q -> tail = null;
    } else {
        next -> prev = null;
        q -> head = next;
    }
    return i;
}
```

C

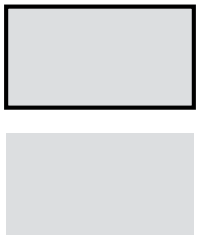
M

implementation



Example: Thread Queue

specification



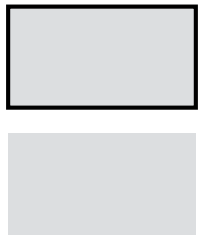
L_2

```
Definition tcbp := ZMap.t state.
Definition td_queue := List Z.
```

Coq

Example: Thread Queue

specification



L_2

tcbp(0) tcbp(1) tcbp(2)

s0 s1 s2

```
Definition tcbp := ZMap.t state.
Definition td_queue := List Z.
```

Coq

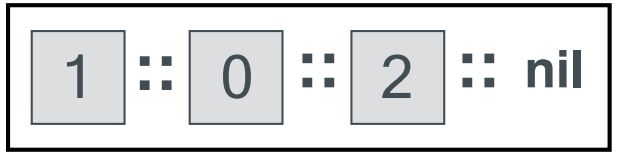
Example: Thread Queue

specification

L_2

tcbp(0) tcbp(1) tcbp(2)

td_queue

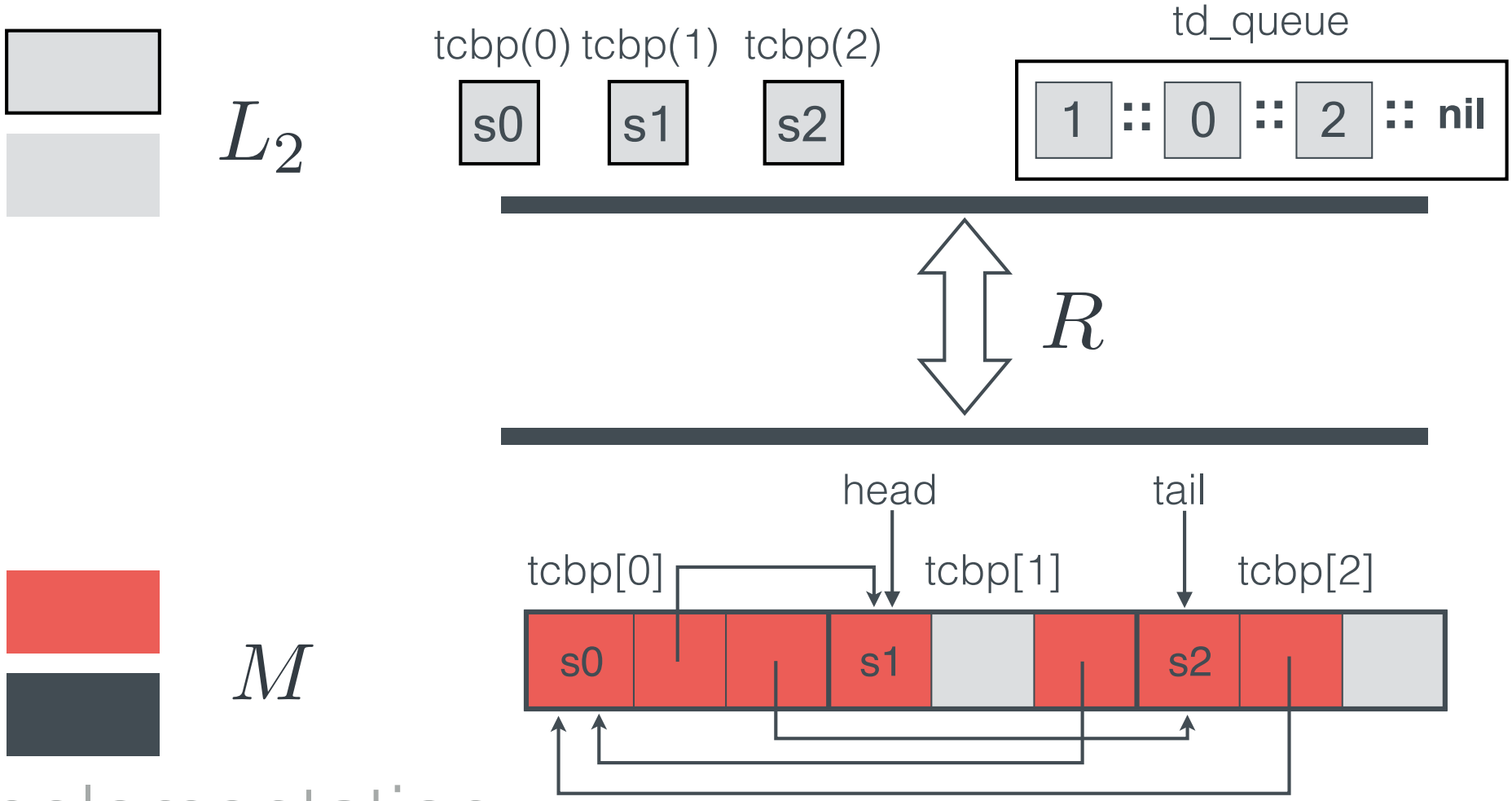



```
Definition tcbp := ZMap.t state.
Definition td_queue := List Z.
```

Coq

Example: Thread Queue

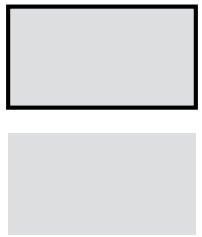
specification



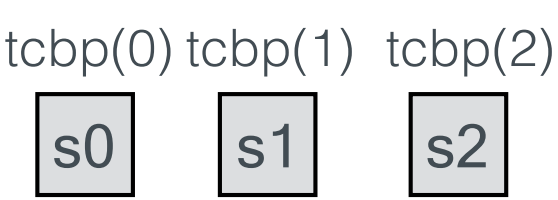
implementation

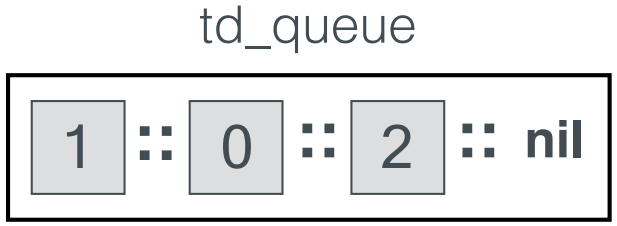
Example: Thread Queue

specification



L_2



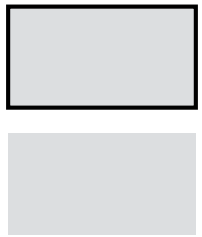



```
Function dequeue (q) :=
match q with
  | head :: q' => (q', Some head)
  | nil => (nil, None)
end.
```

Coq

Example: Thread Queue

specification



L_2

tcbp(0) tcbp(1) tcbp(2)

s0 s1 s2

1

td_queue

0 :: 2 :: nil

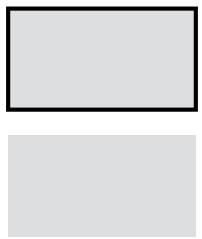
```
Function dequeue (q) :=
match q with
  | head :: q' => (q', Some head)
  | nil => (nil, None)
end.
```

Coq

executable

Simulation Proof

specification

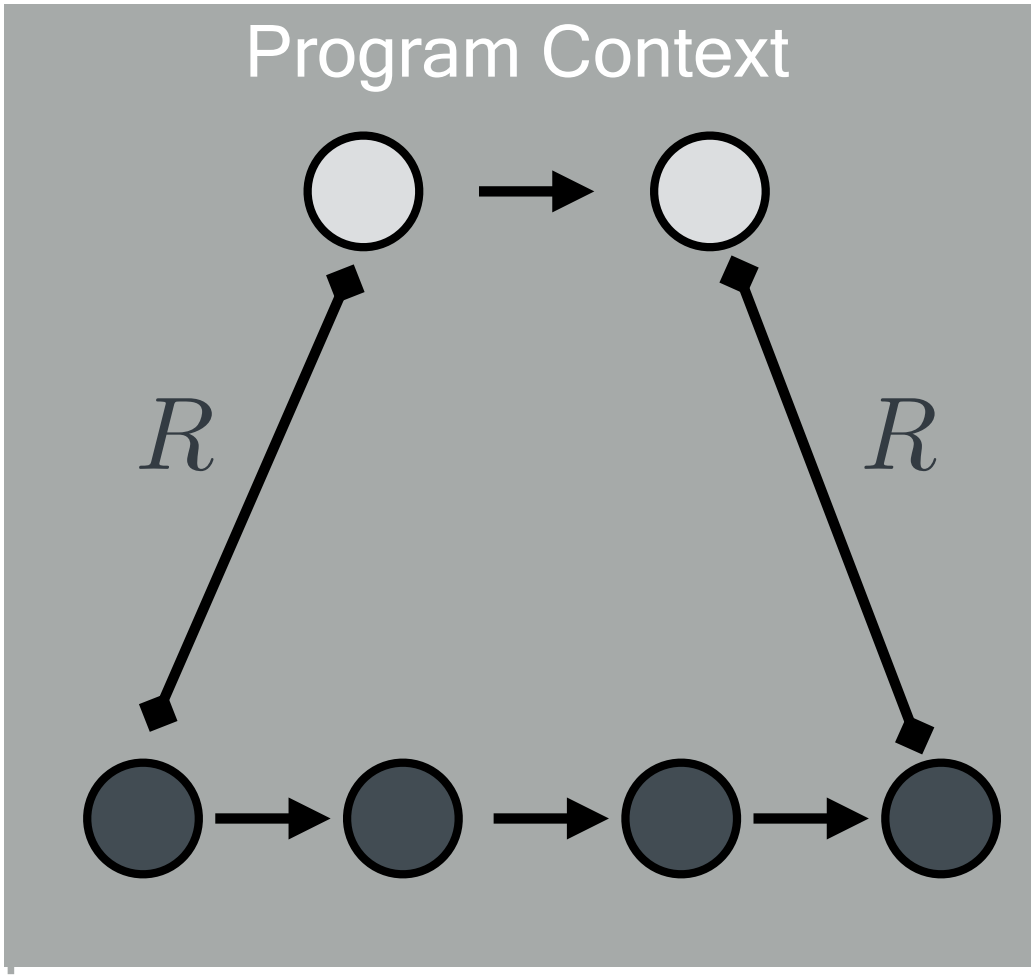


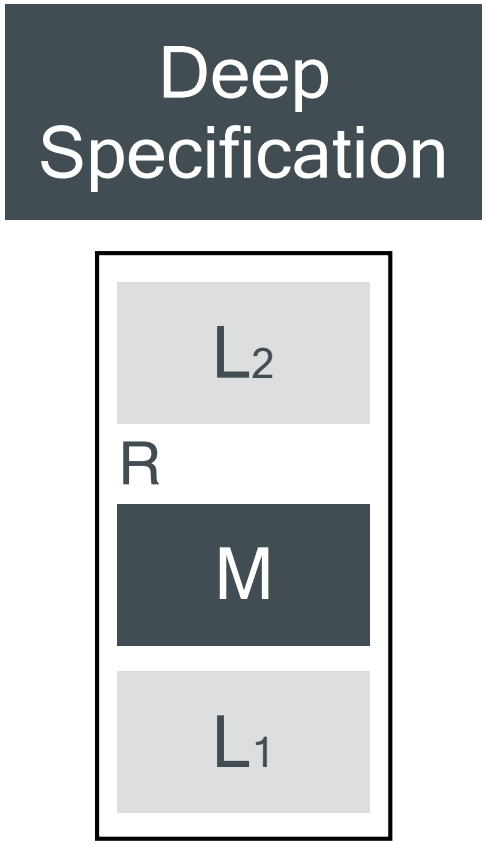
L_2

$\sqcup\!|$

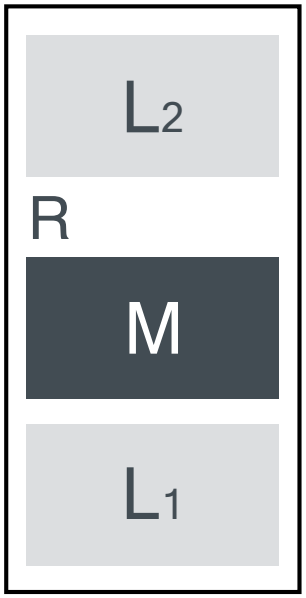
M

implementation





Deep Specification [POPL'15]



Deep spec L_2 captures **all** we need to know about **M** over L_1

Any property about **M** can be proved using L_2 alone

No need to look at **M** again

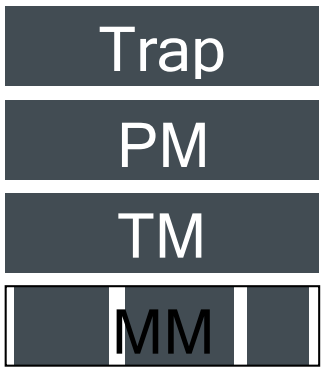
mCertiKOS

kernel

code

seq machine

mCertiKOS



seq machine

mCertiKOS

kernel

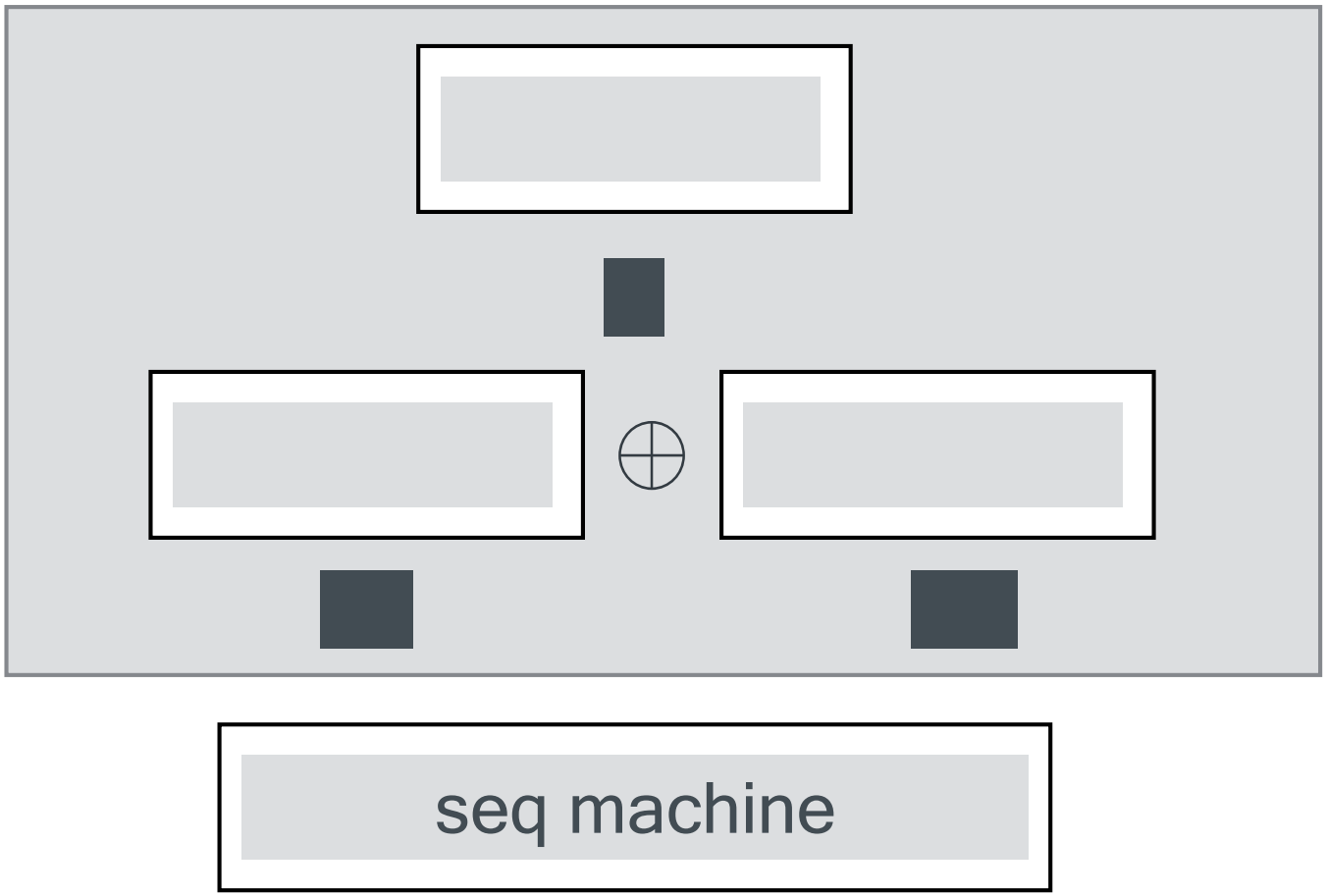
Trap

PM

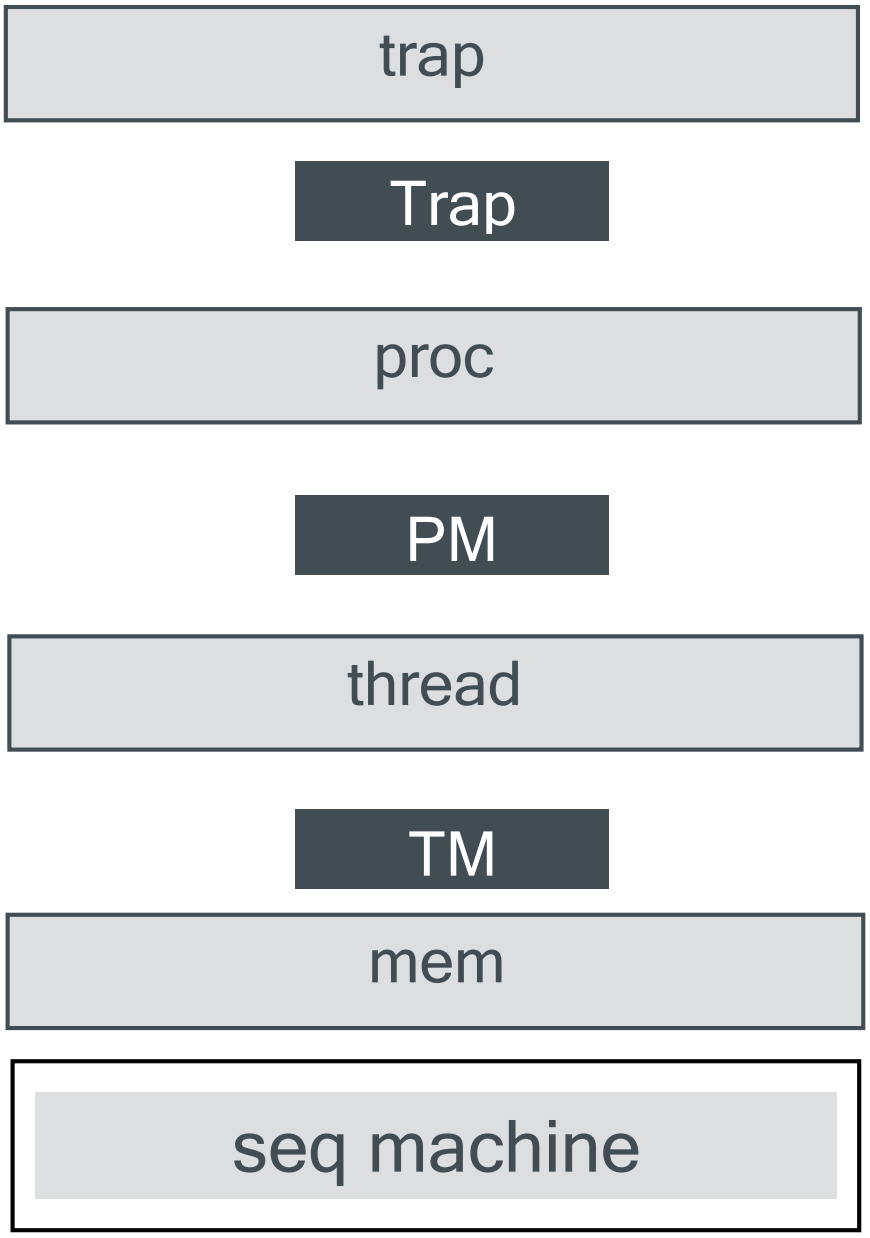
TM

MM

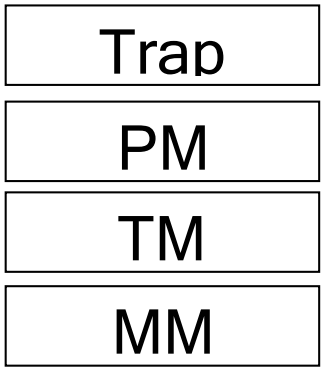
memory management



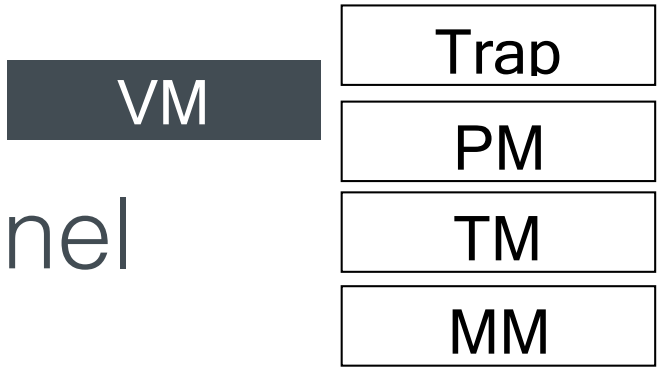
mCertiKOS



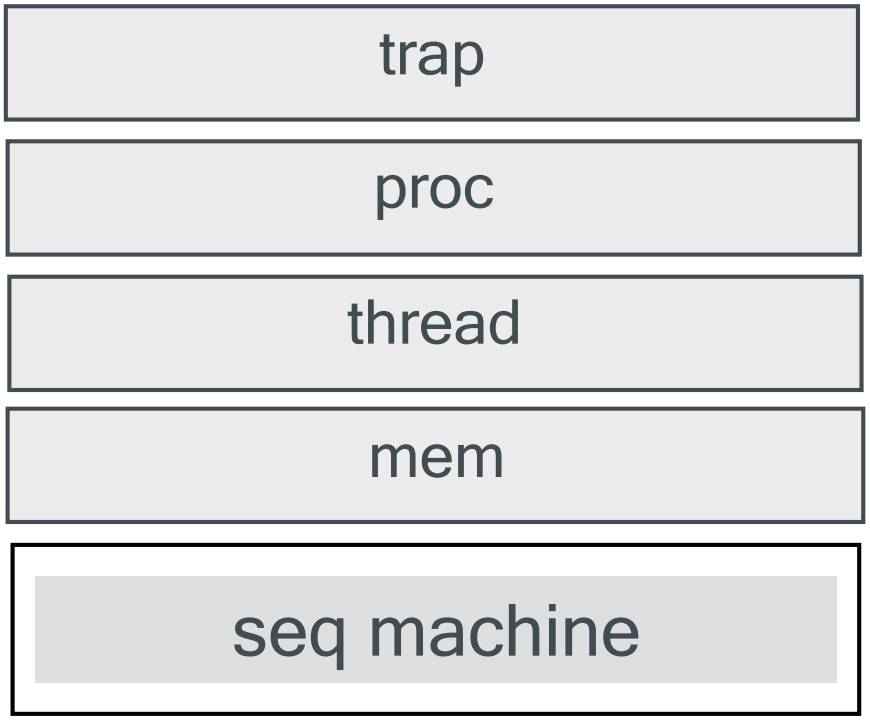
kernel



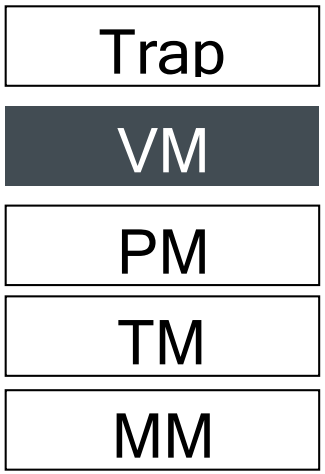
mCertiKOS

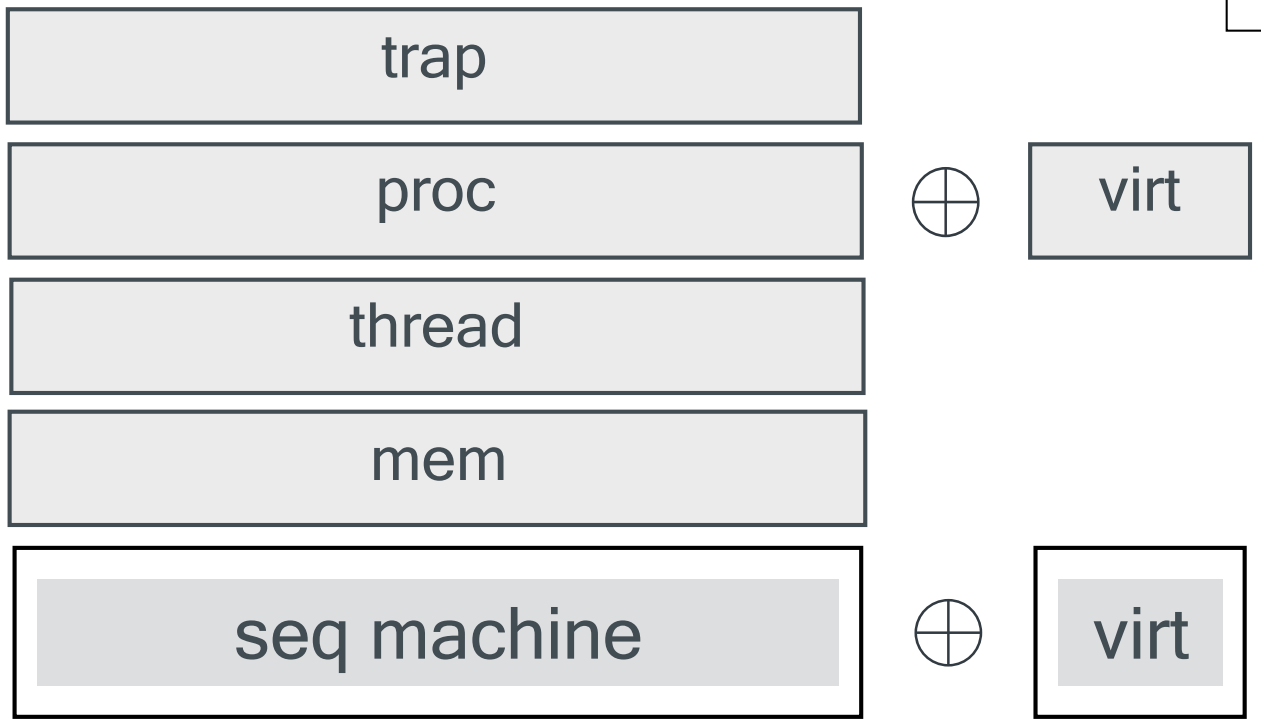


certified sequential kernel

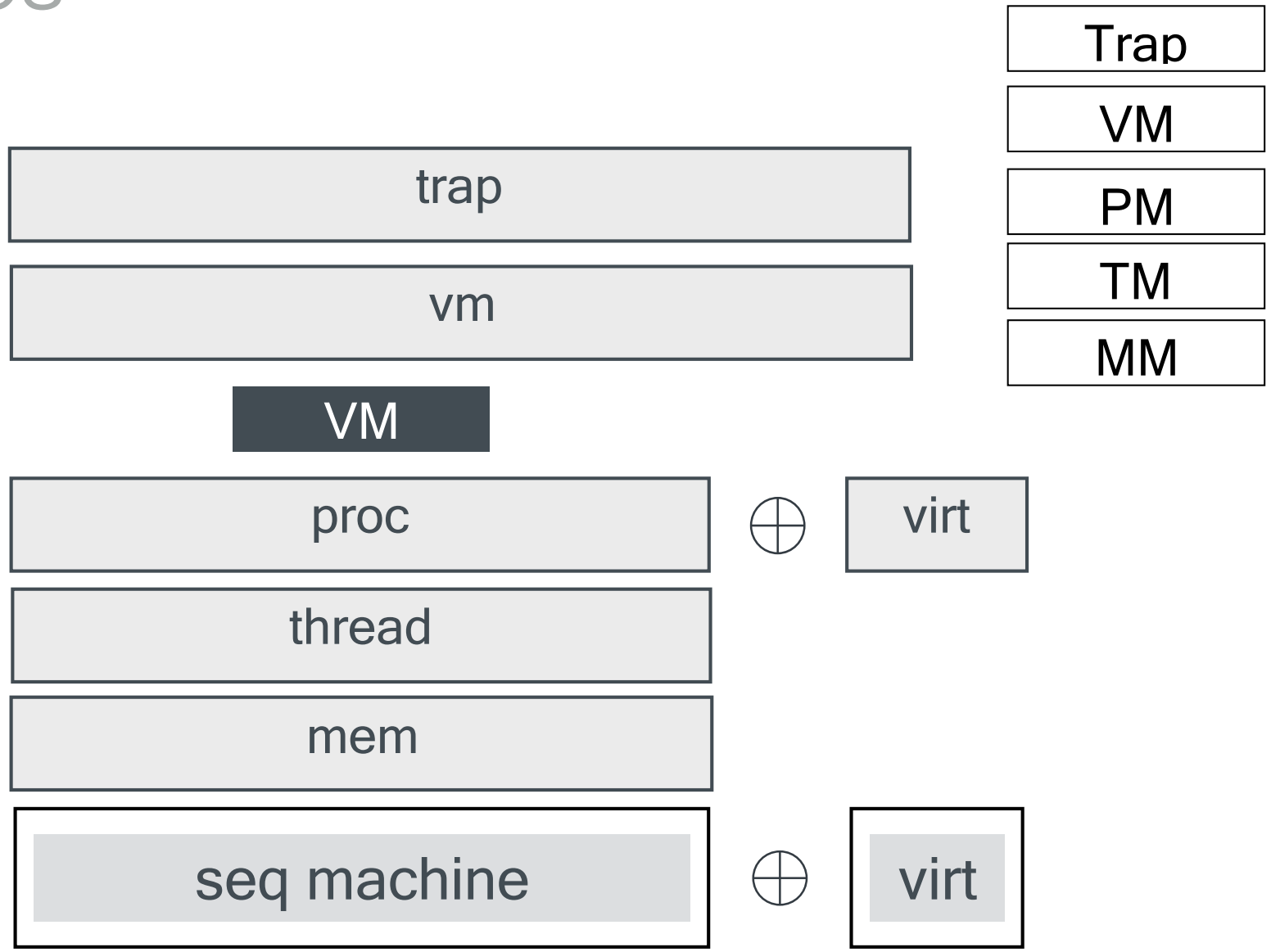


mCertiKOS

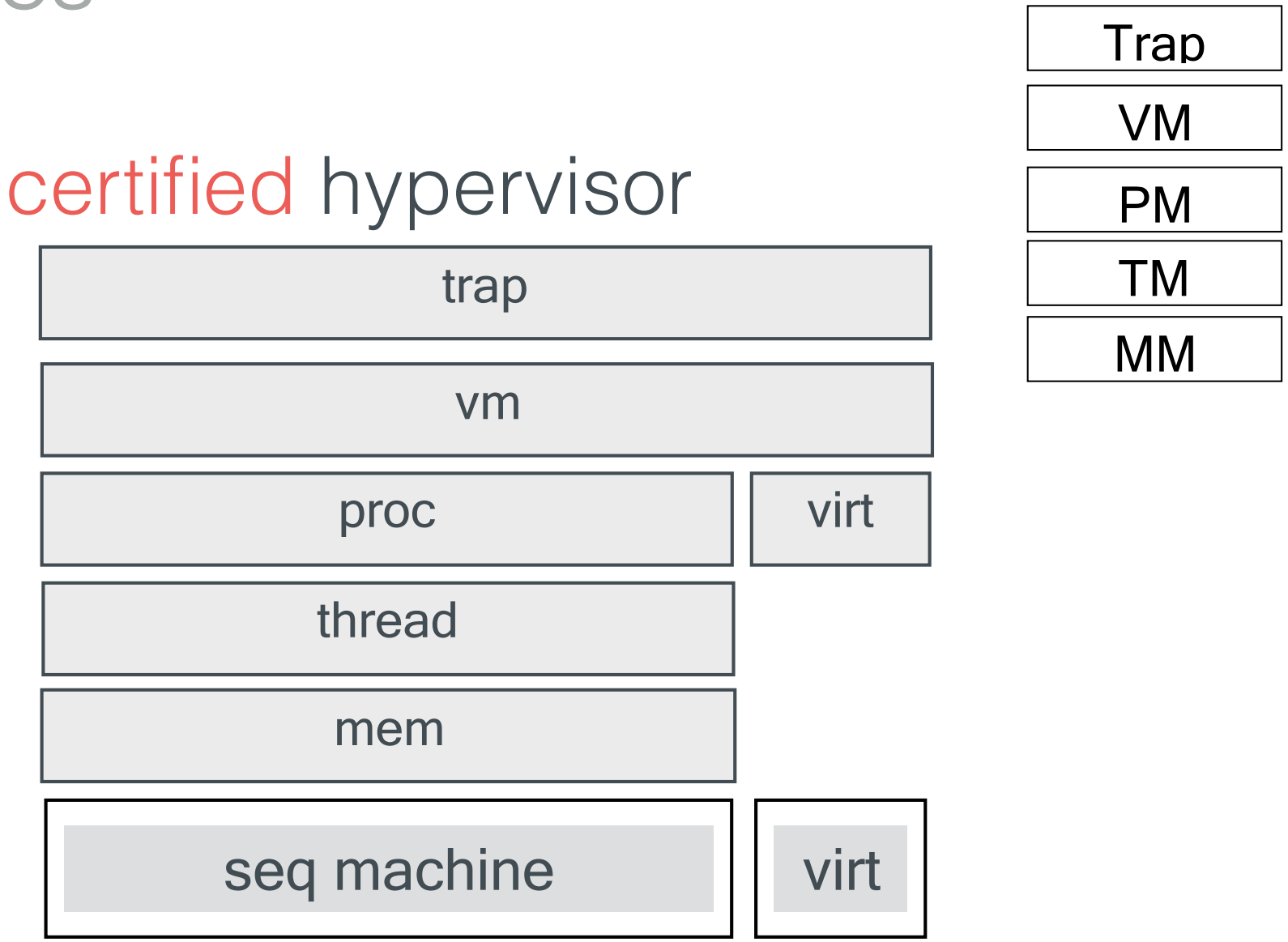




mCertiKOS


Trap
VM
PM
TM
MM
trap
vm
VM
proc
virt
thread
mem
seq machine
virt

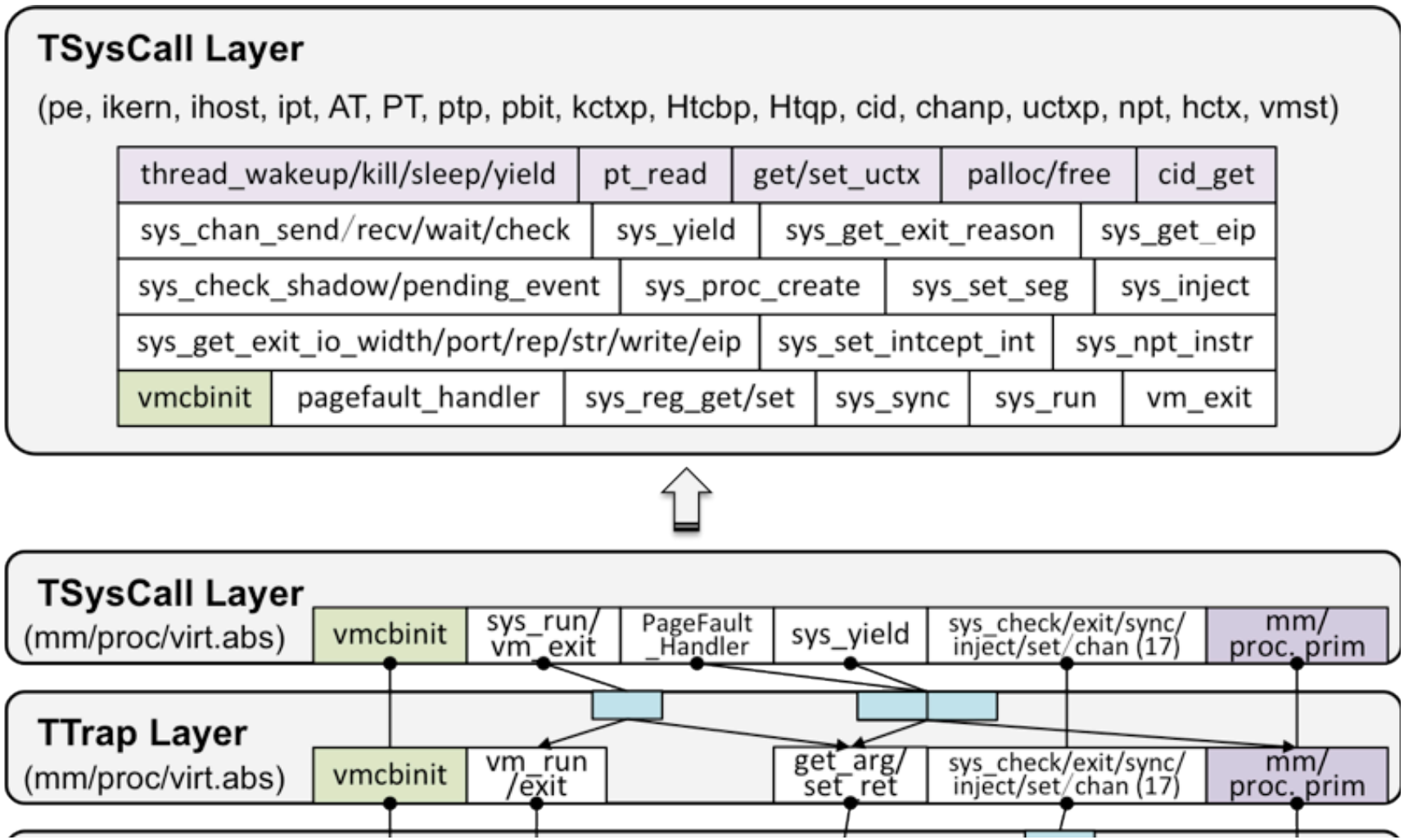
mCertiKOS


Trap
VM
PM
TM
MM
certified hypervisor
trap
vm
proc
virt
thread
mem
seq machine
virt

mCertiKOS 3k LOC

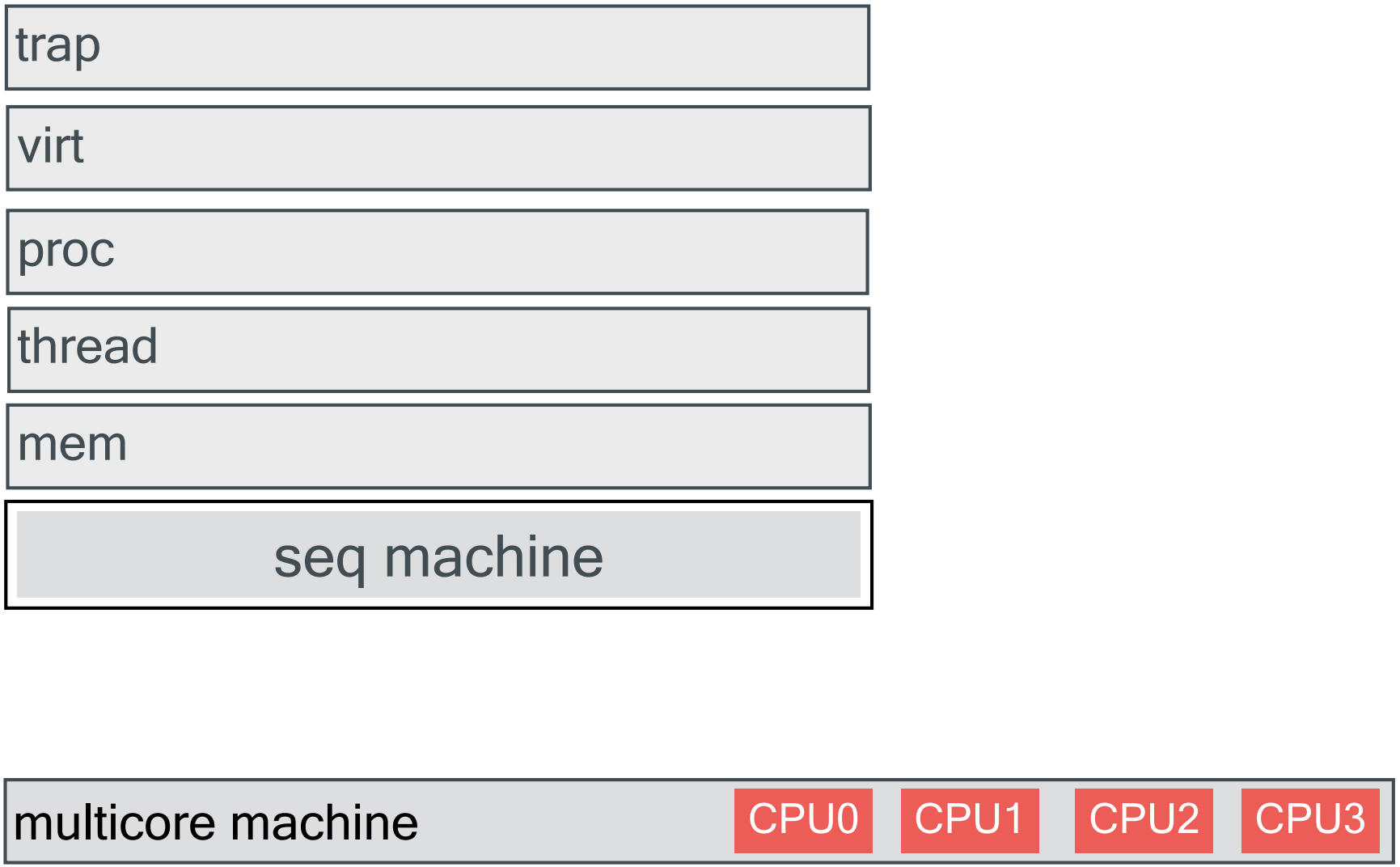
[POPL'15] 1 person year

Can **boot** Linux as a guest

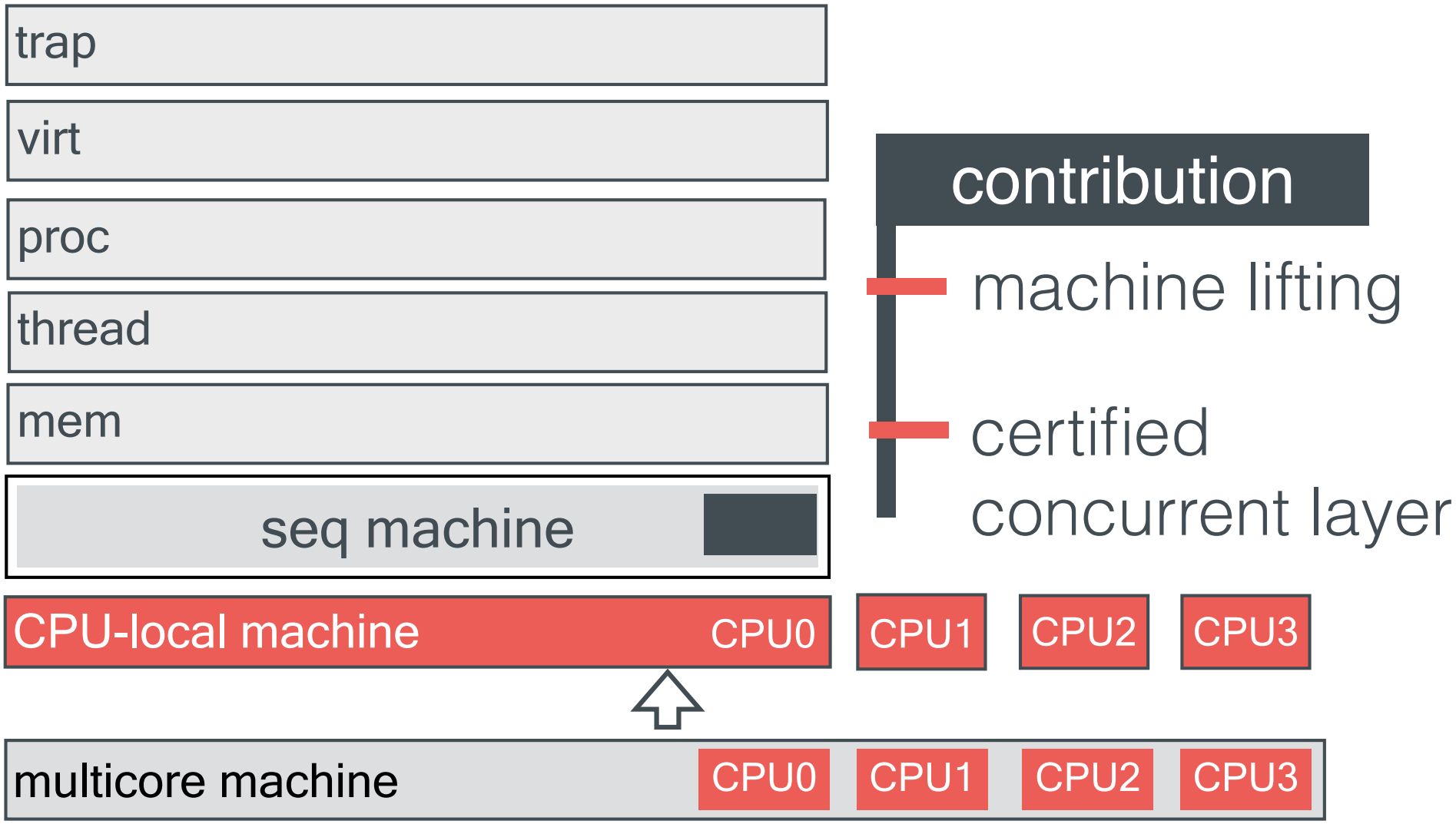


Concurrent Framework [OSDI'16]

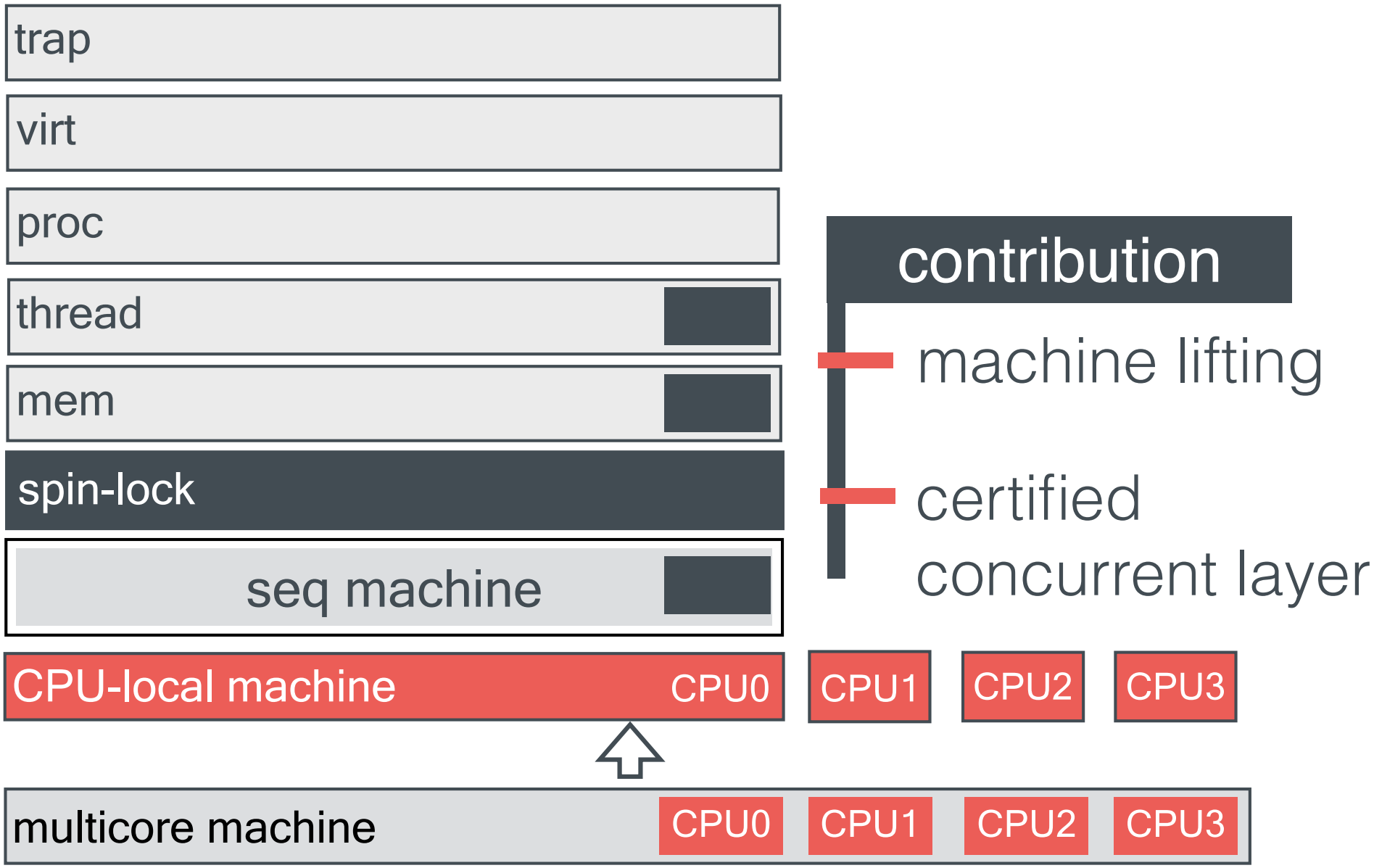
certified sequential kernel



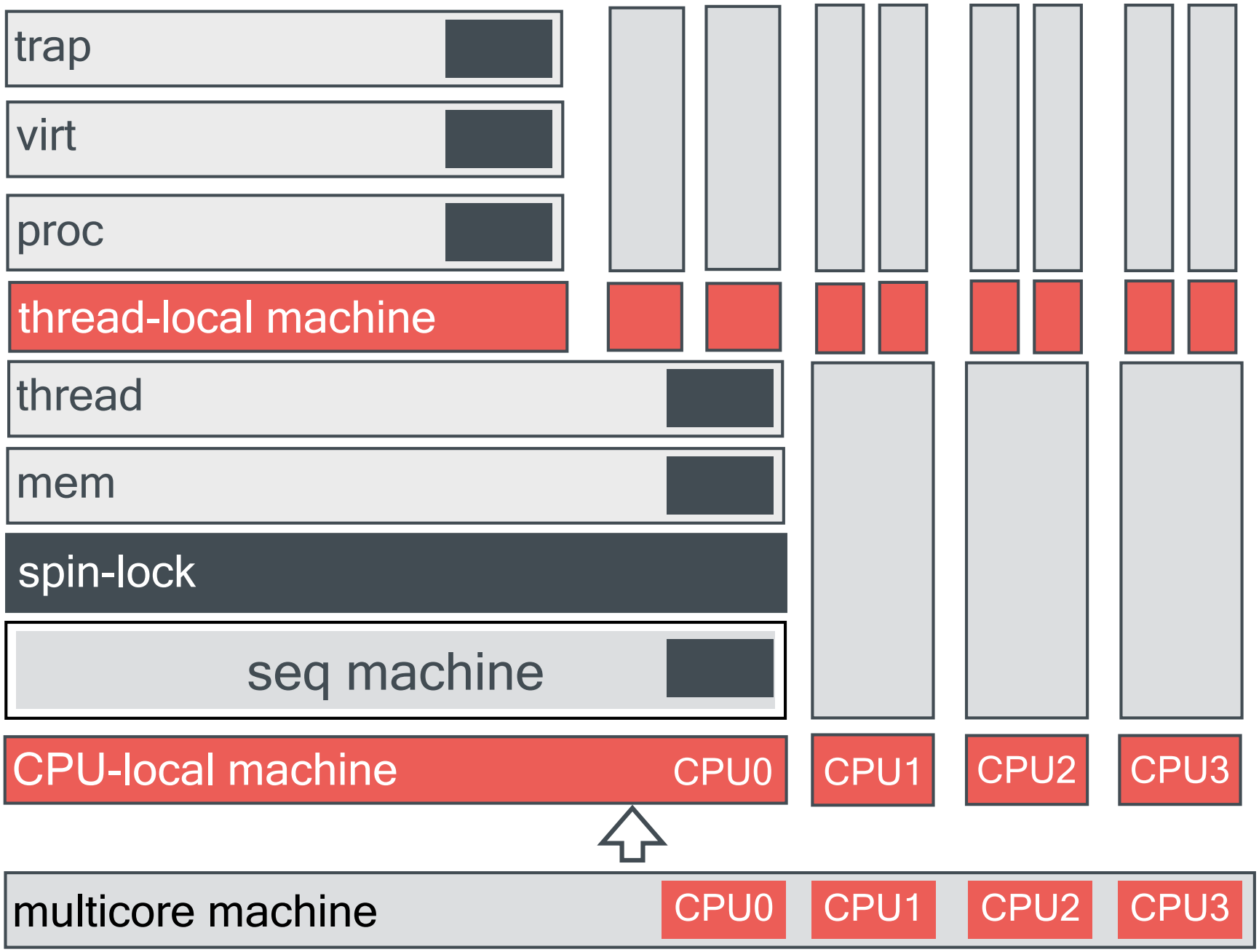
Concurrent Framework [OSDI'16]



Concurrent Framework [OSDI'16]

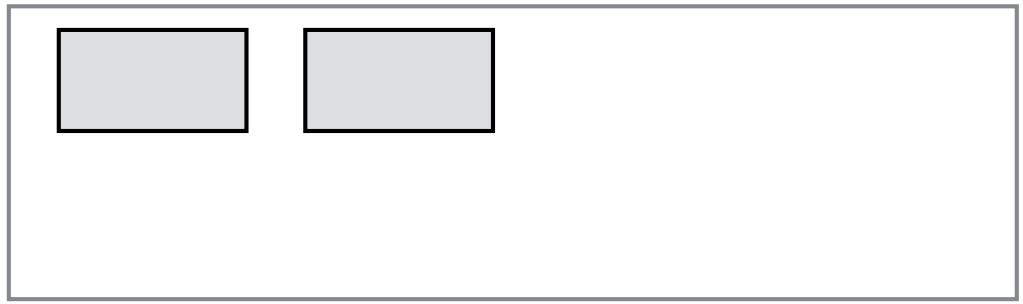


Concurrent Framework [OSDI'16]



Certified Concurrent Layers

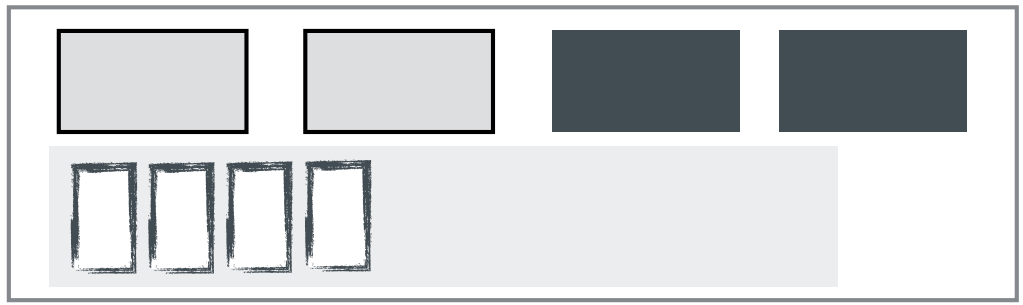
local certified objects



L_1

Certified Concurrent Layers

atomic objects

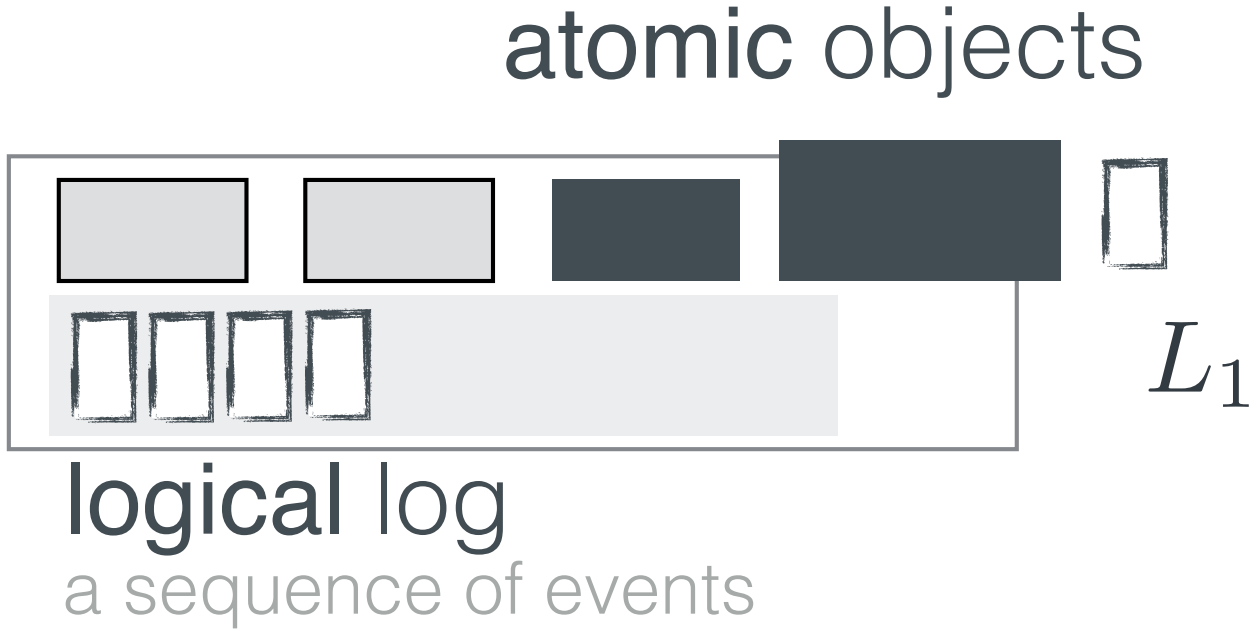


L_1

logical log

a sequence of events

Certified Concurrent Layers



Certified Concurrent Layers

Certified Concurrent Layers

to share 

L_1

Certified Concurrent Layers

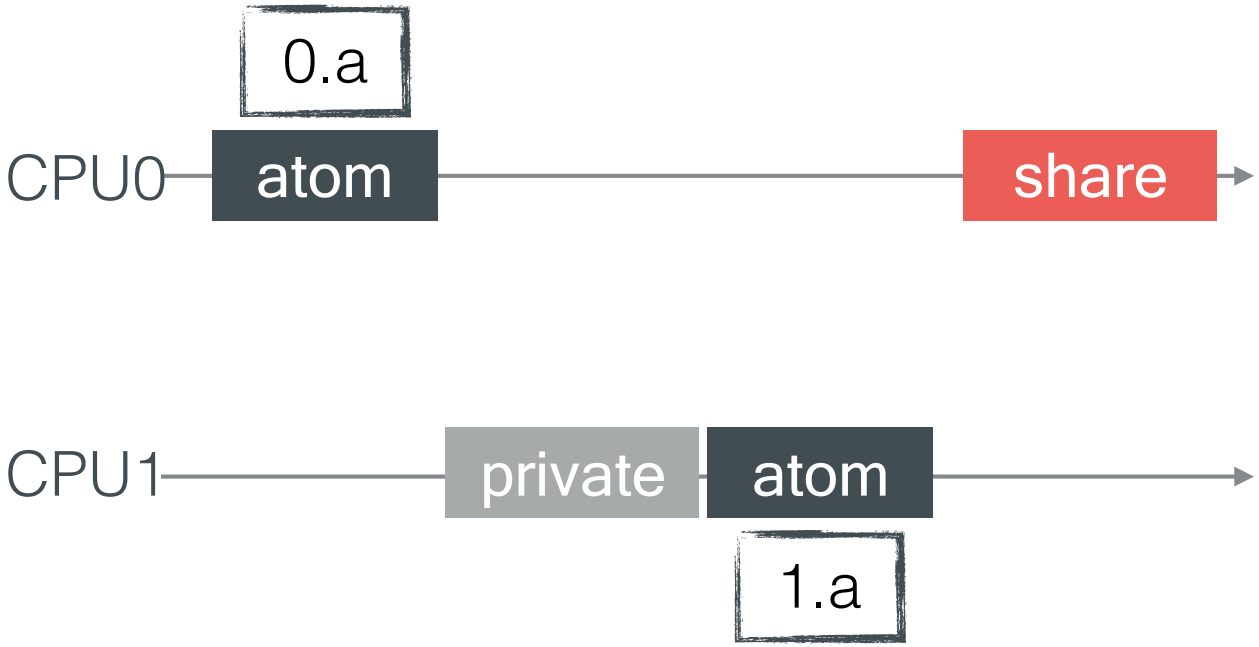
L_2

fine-grained locking

L_1

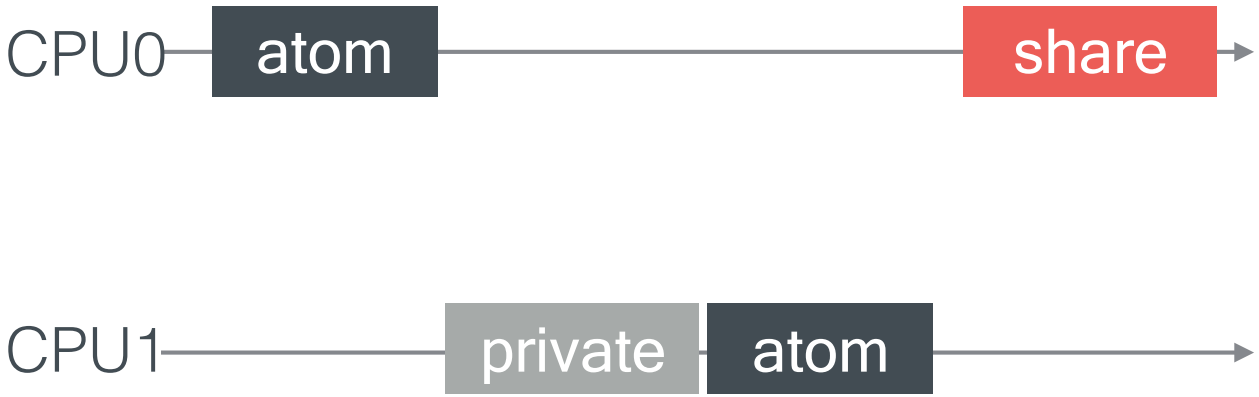
Concurrent Framework

step 0: raw x86 **multicore** model



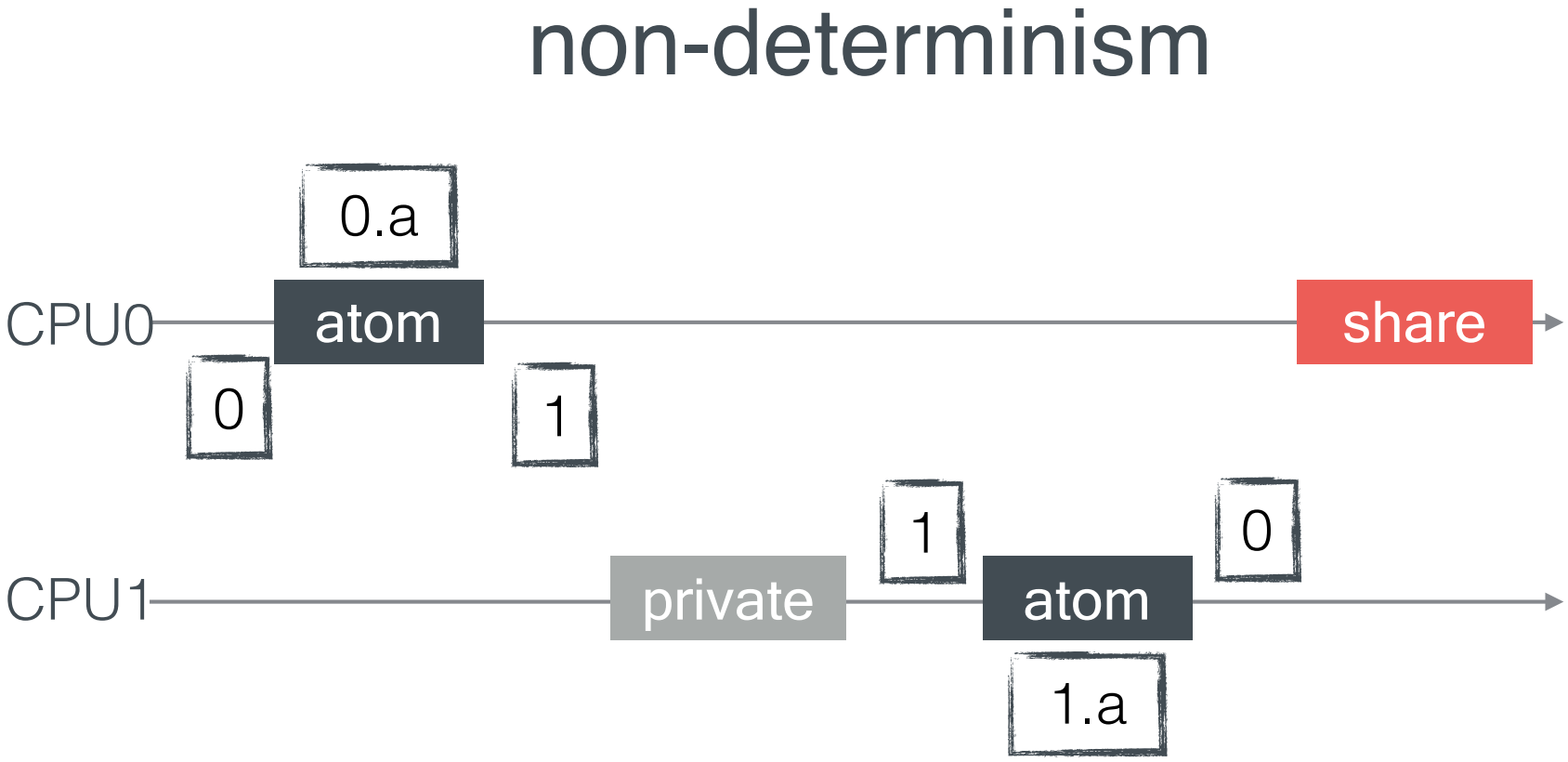
step 0: raw x86 **multicore** model

non-determinism

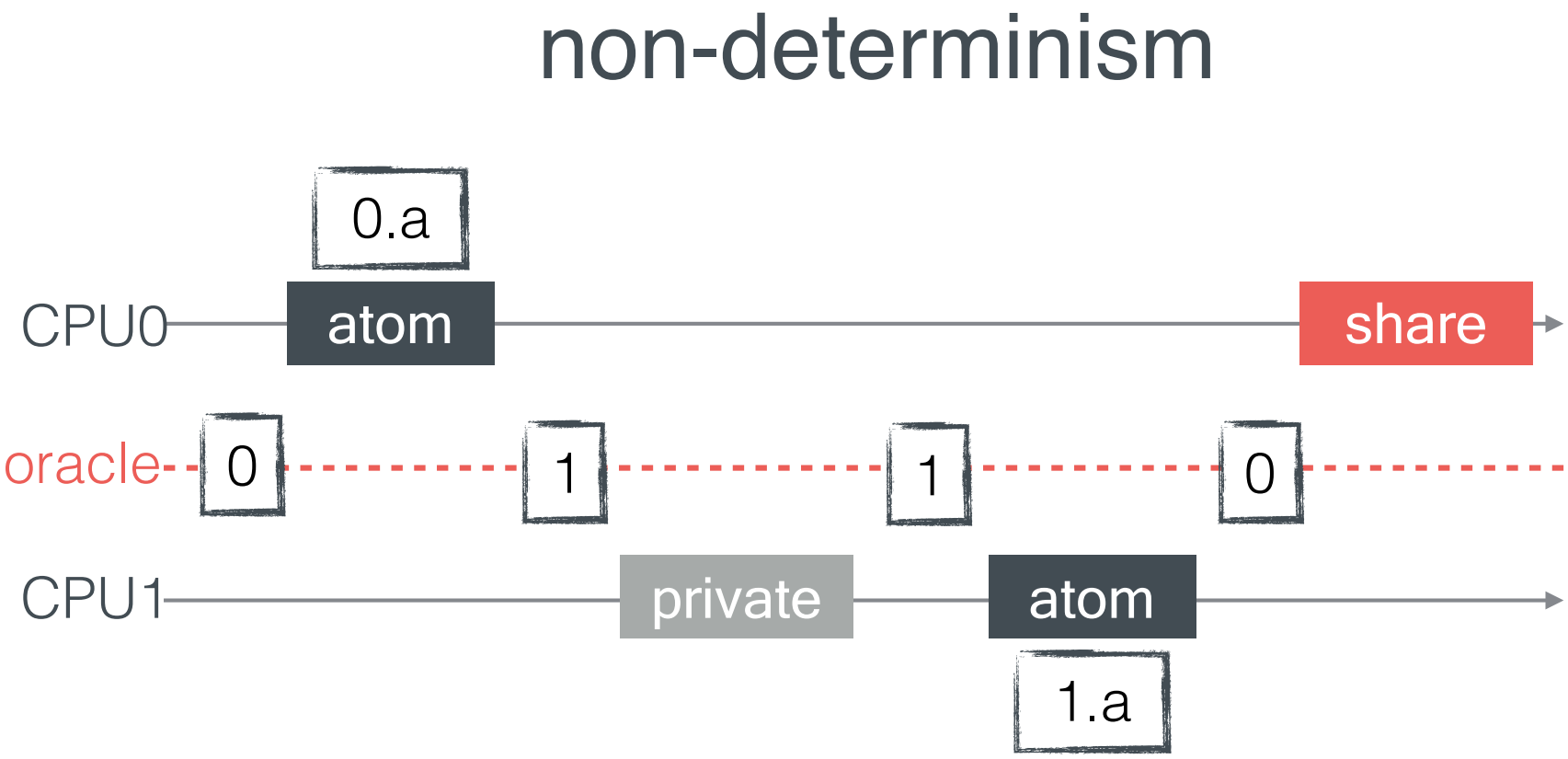


logical log

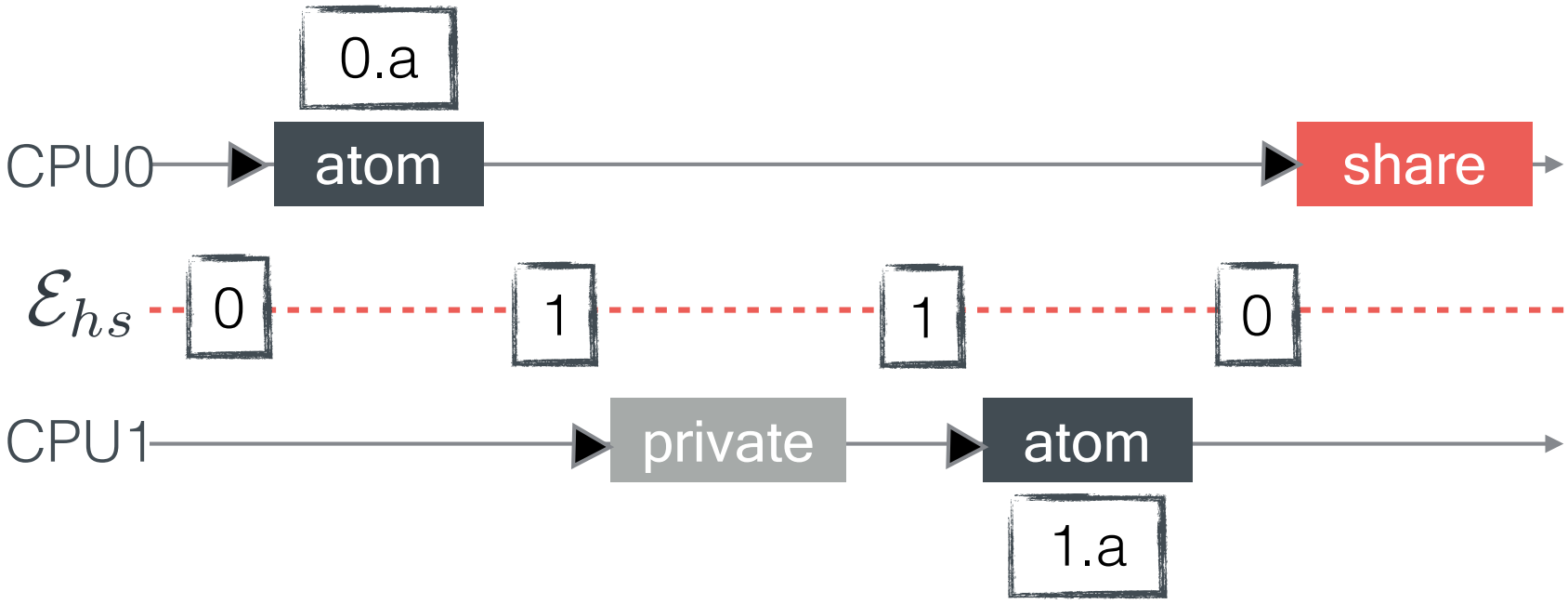
step 0: raw x86 **multicore** model



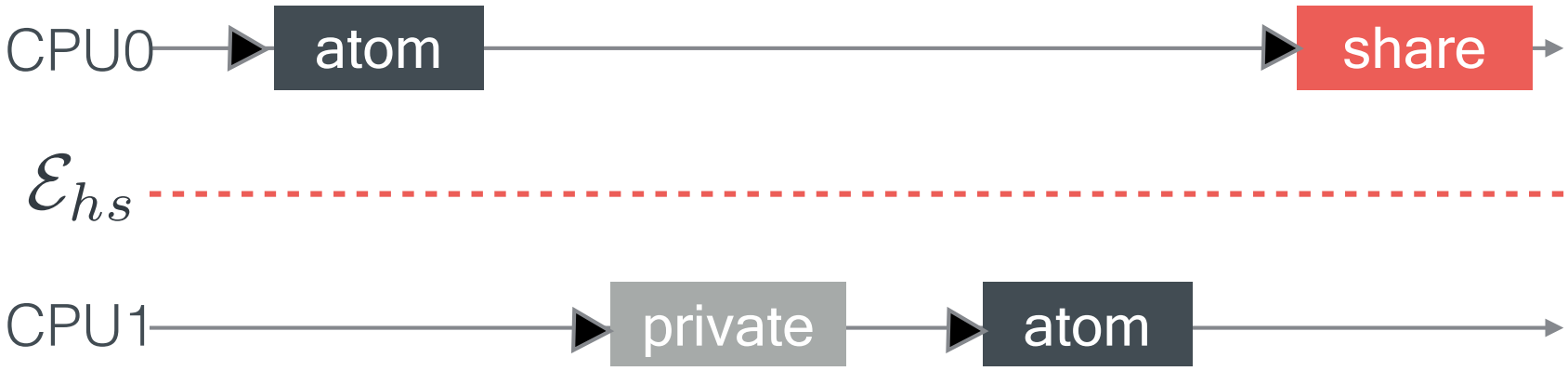
step 0: raw x86 **multicore** model



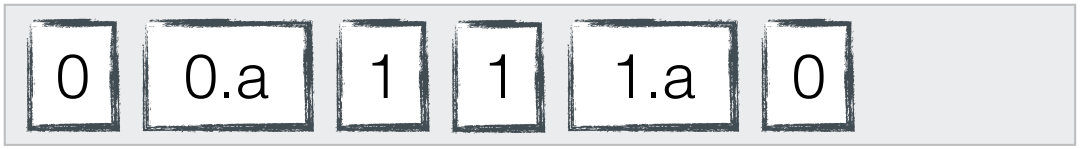
step 1: logical **hardware** scheduler



step 1: logical **hardware** scheduler



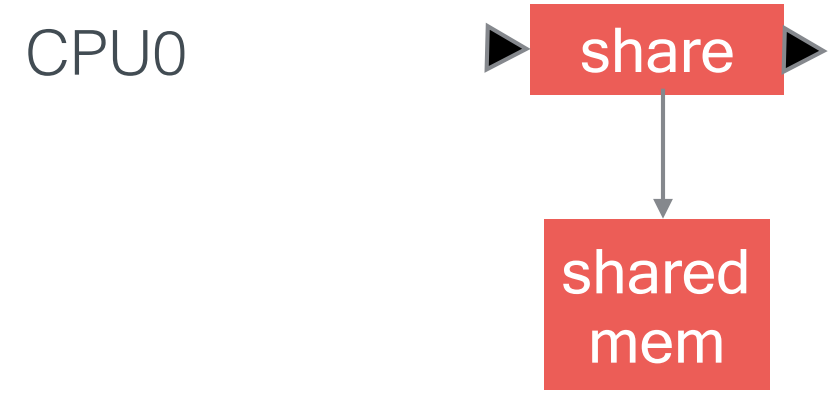
logical log

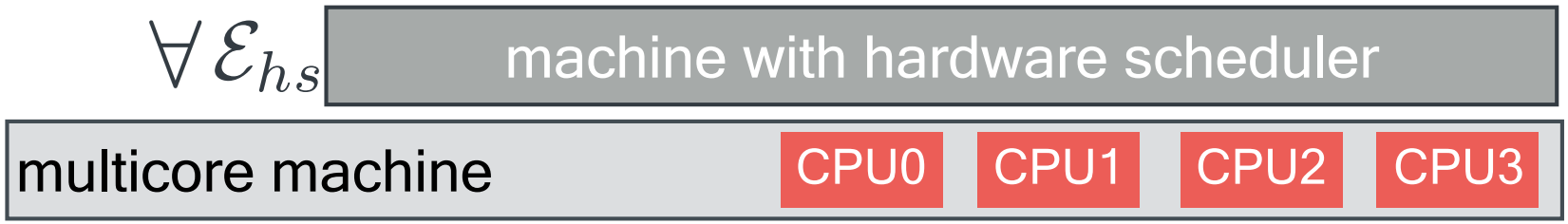


step 1: logical **hardware** scheduler

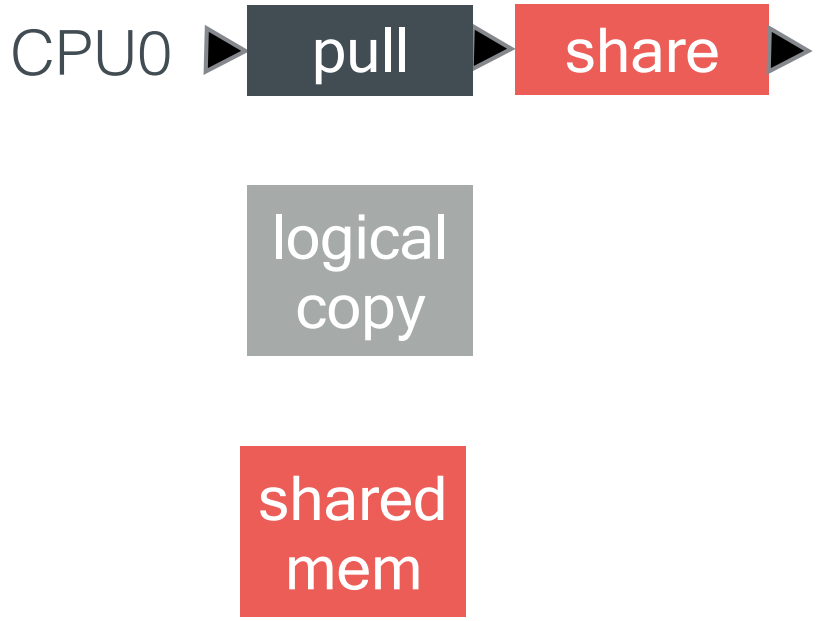
$$\forall\, \mathcal{E}_{hs}$$

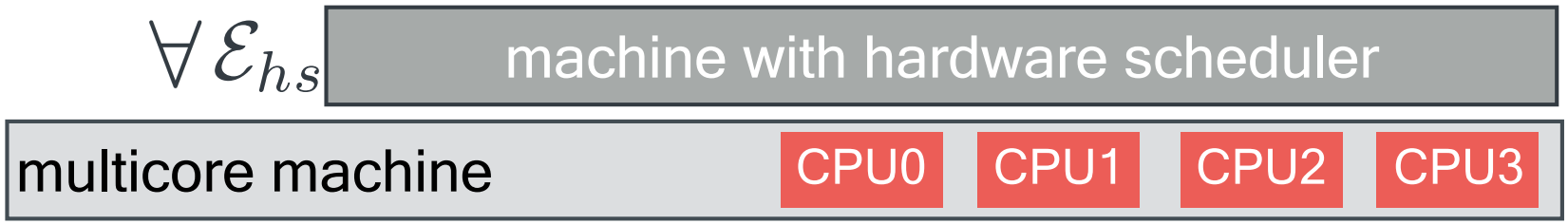
step 2: **push/pull** memory model



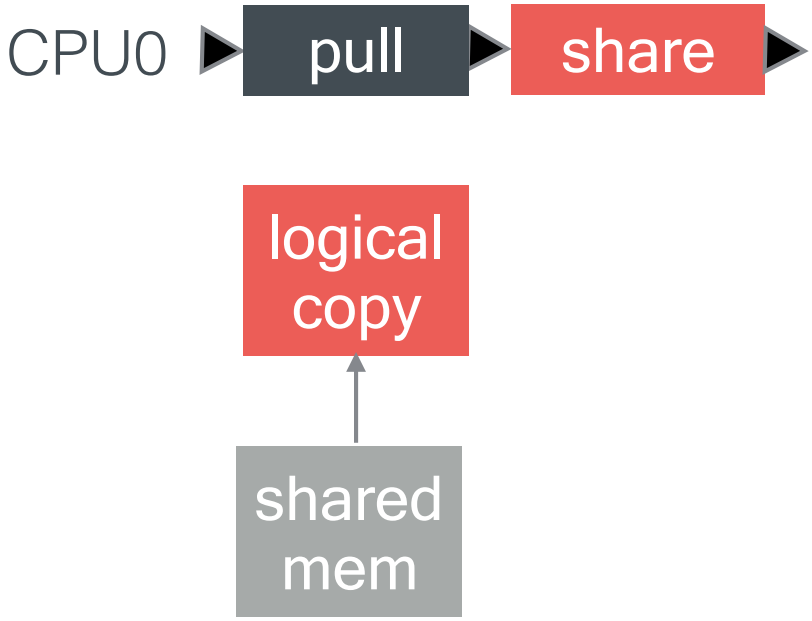


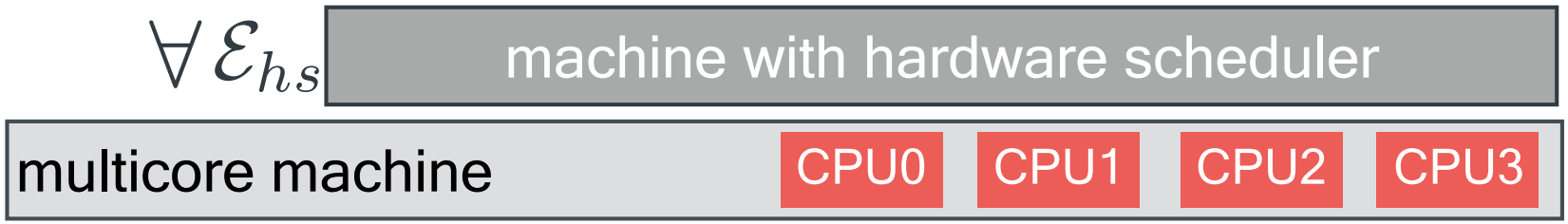
step 2: **push/pull** memory model



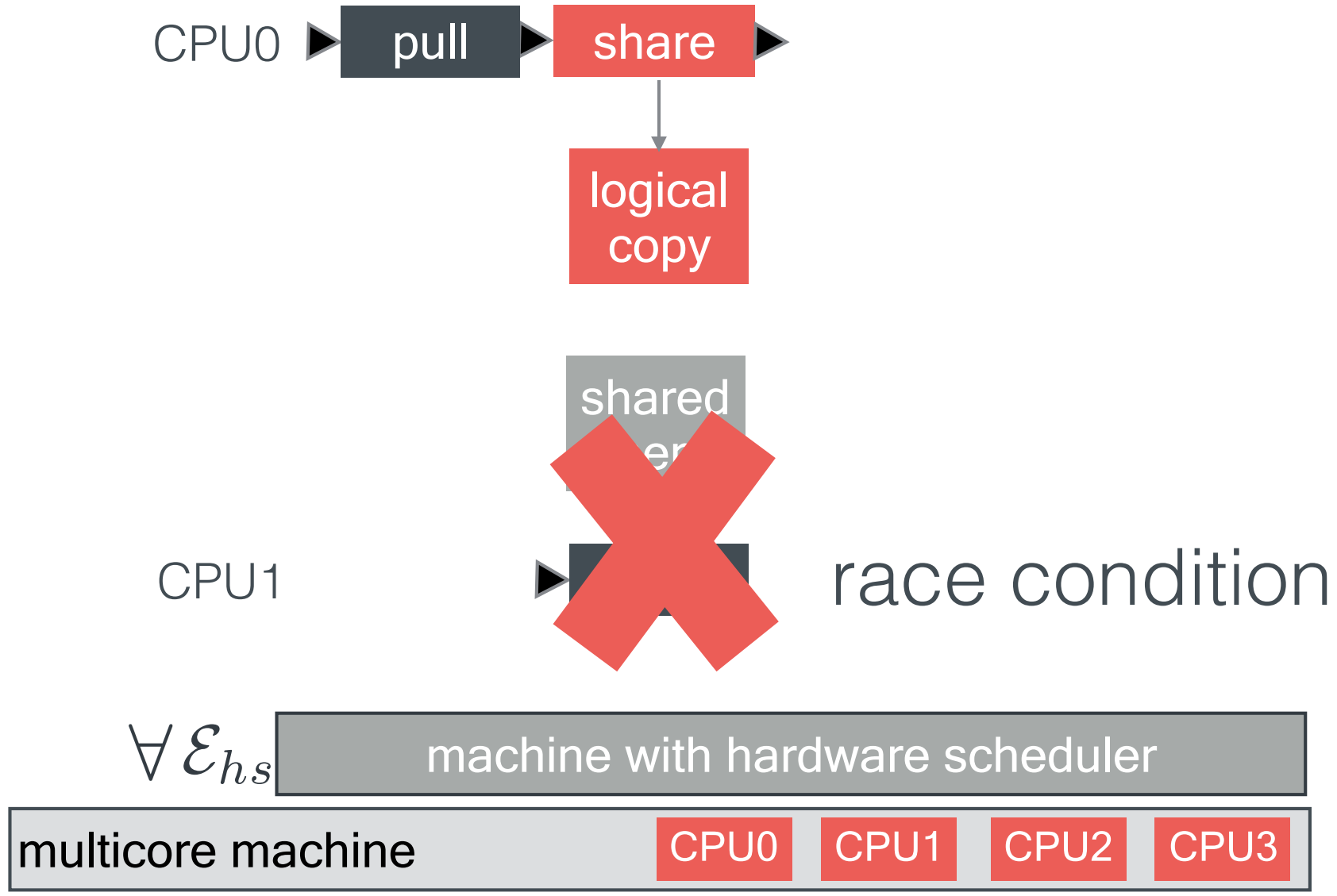


step 2: **push/pull** memory model

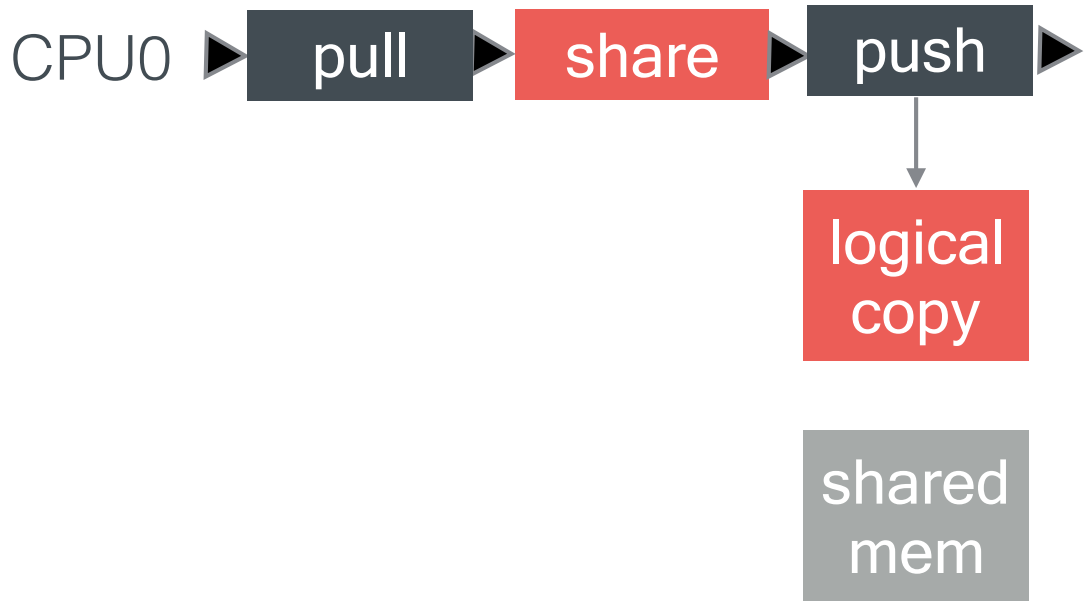


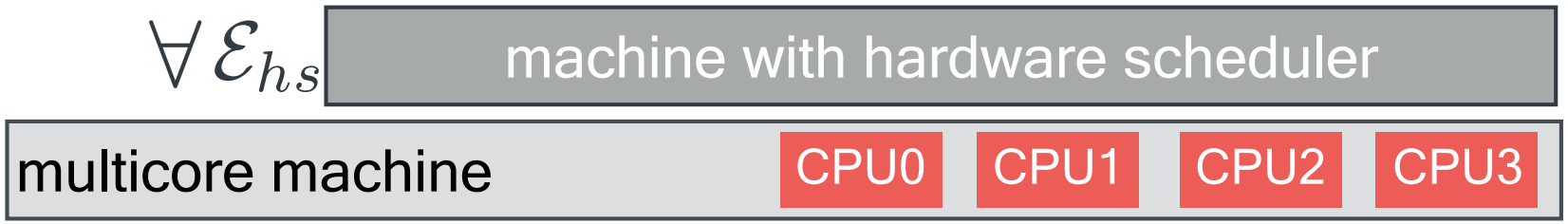


step 2: **push/pull** memory model

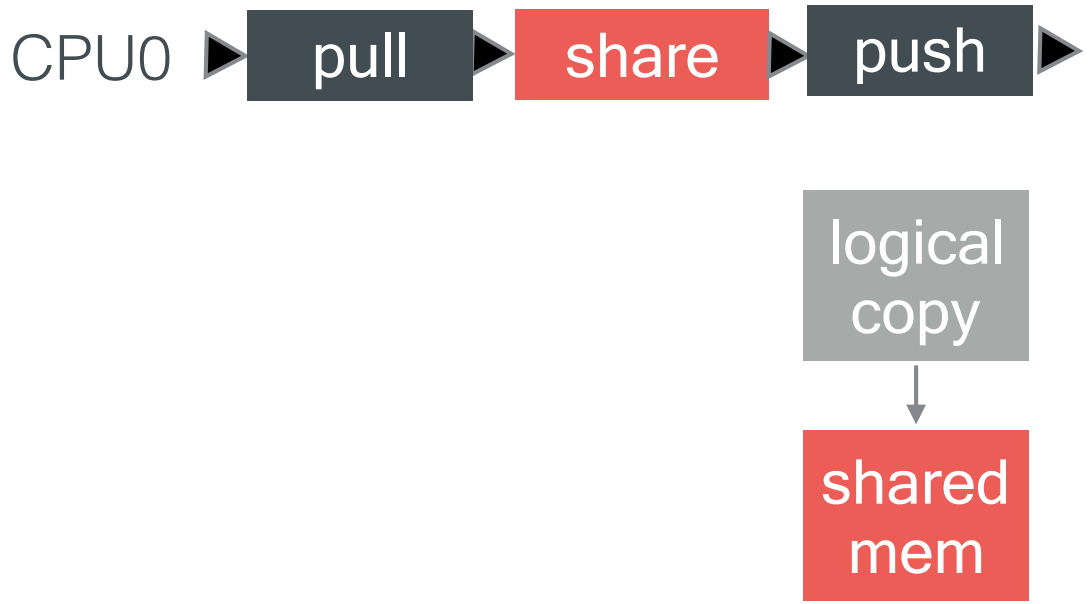


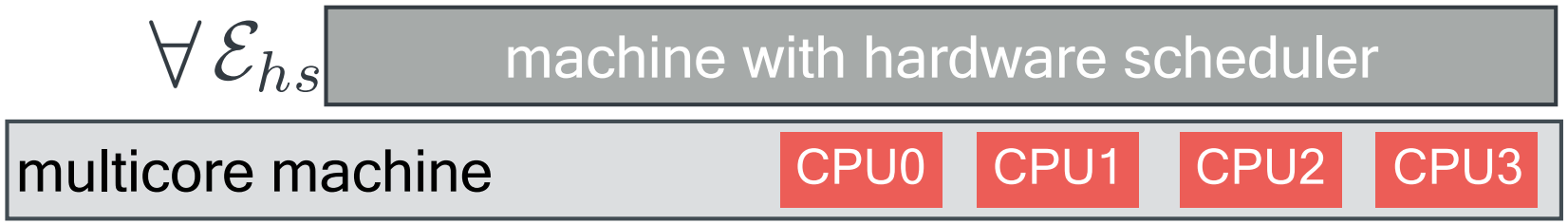
step 2: **push/pull** memory model



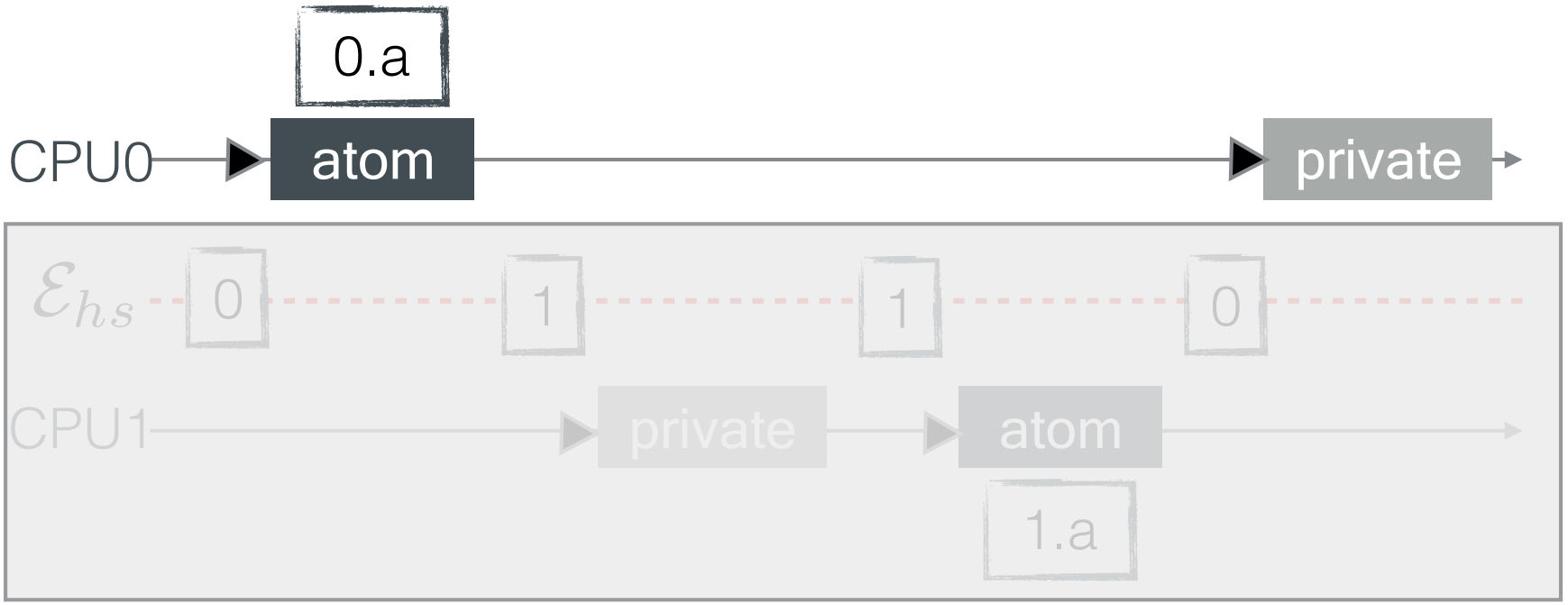


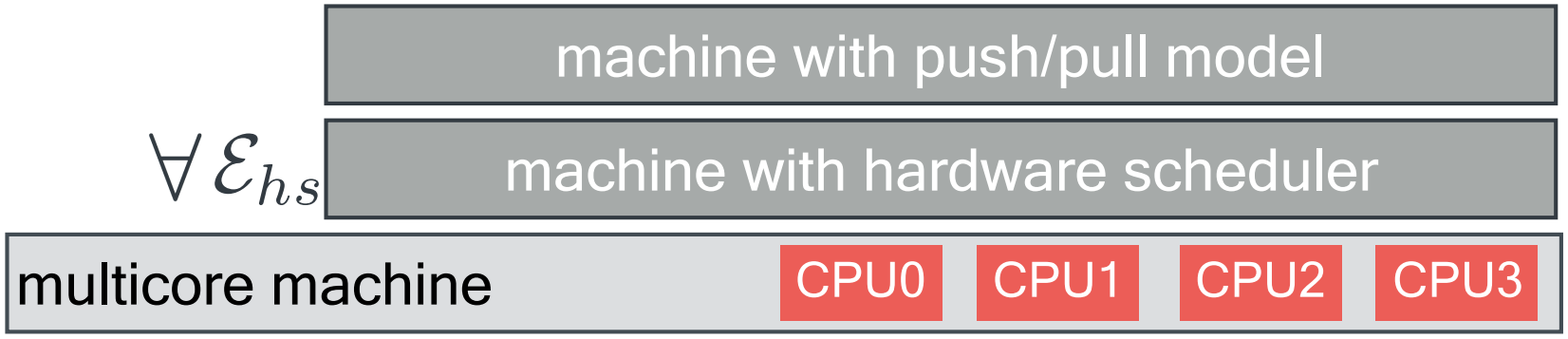
step 2: **push/pull** memory model



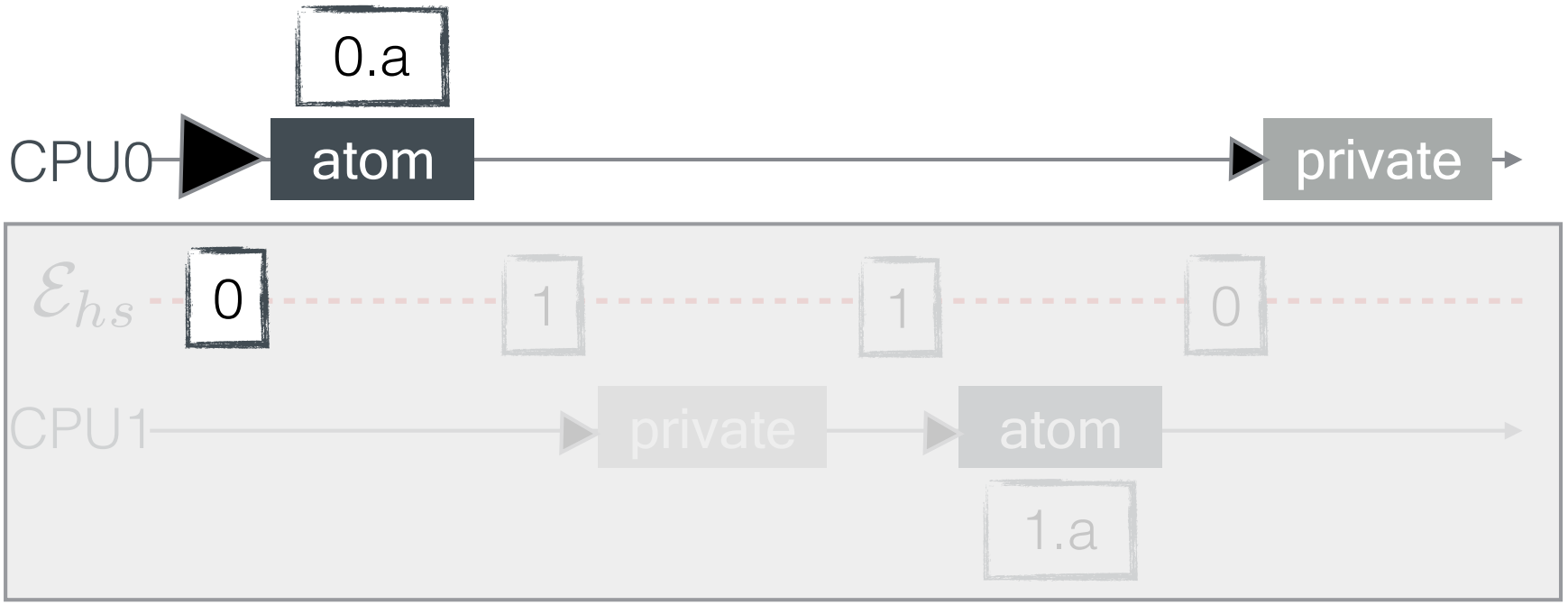


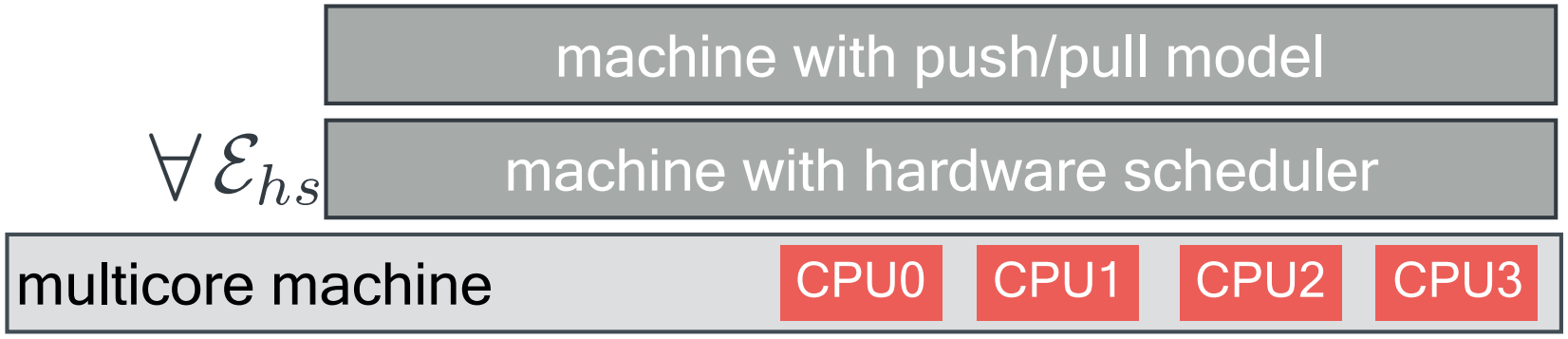
step 3: **environment context** model



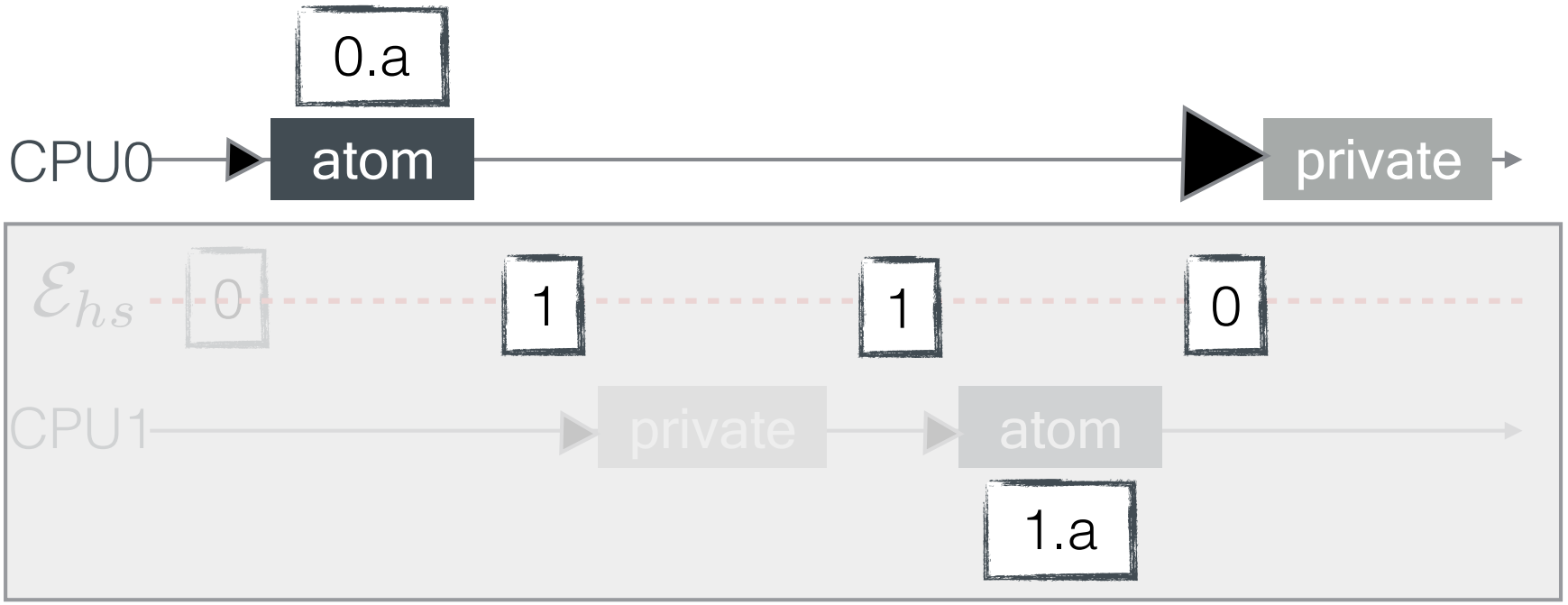


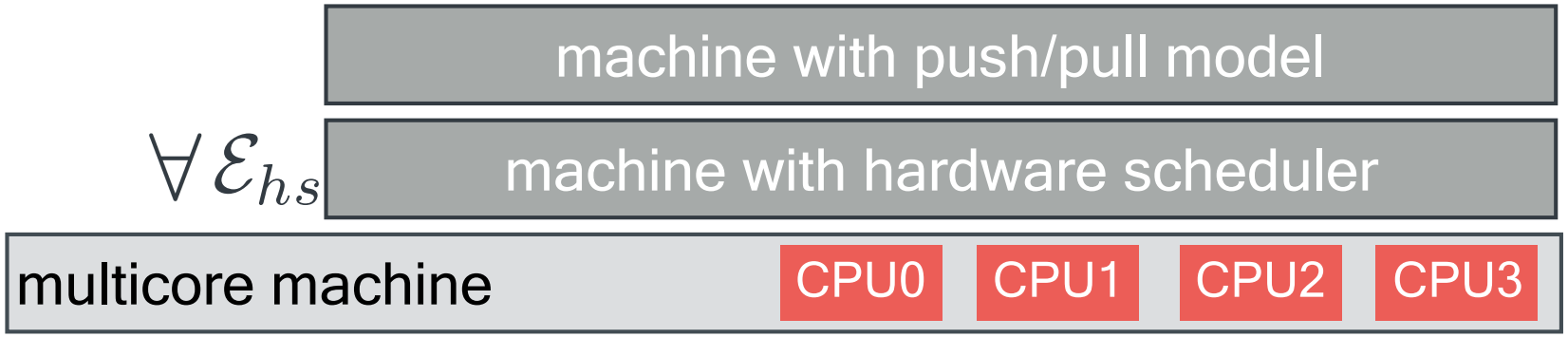
step 3: **environment context** model



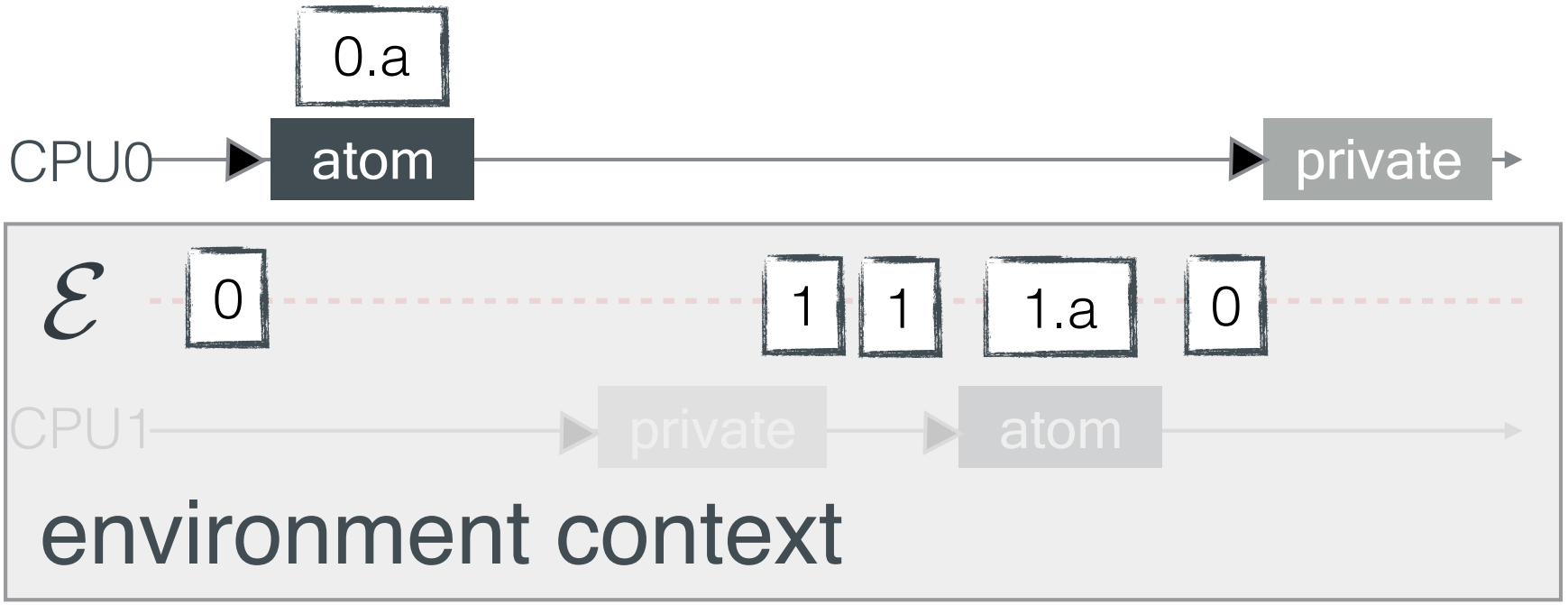


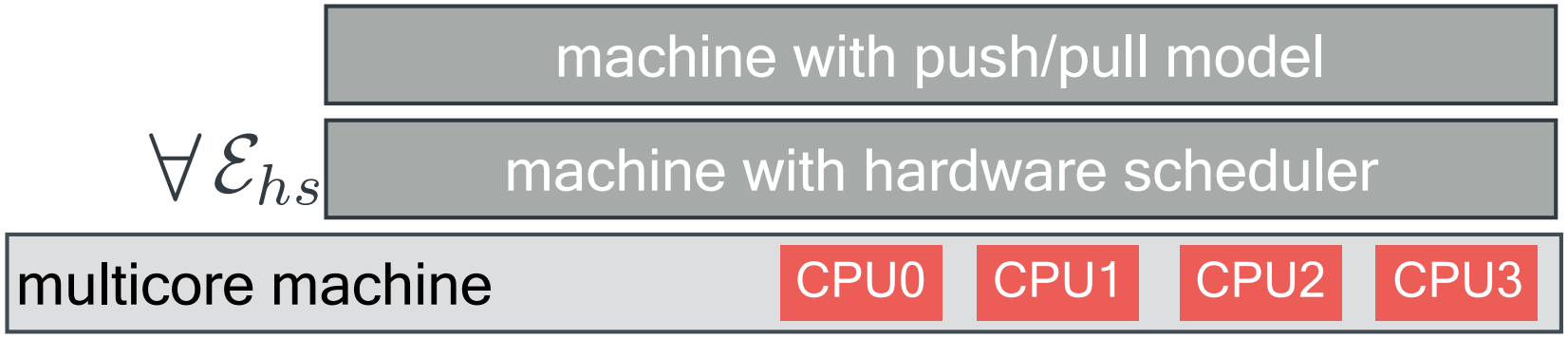
step 3: **environment context** model



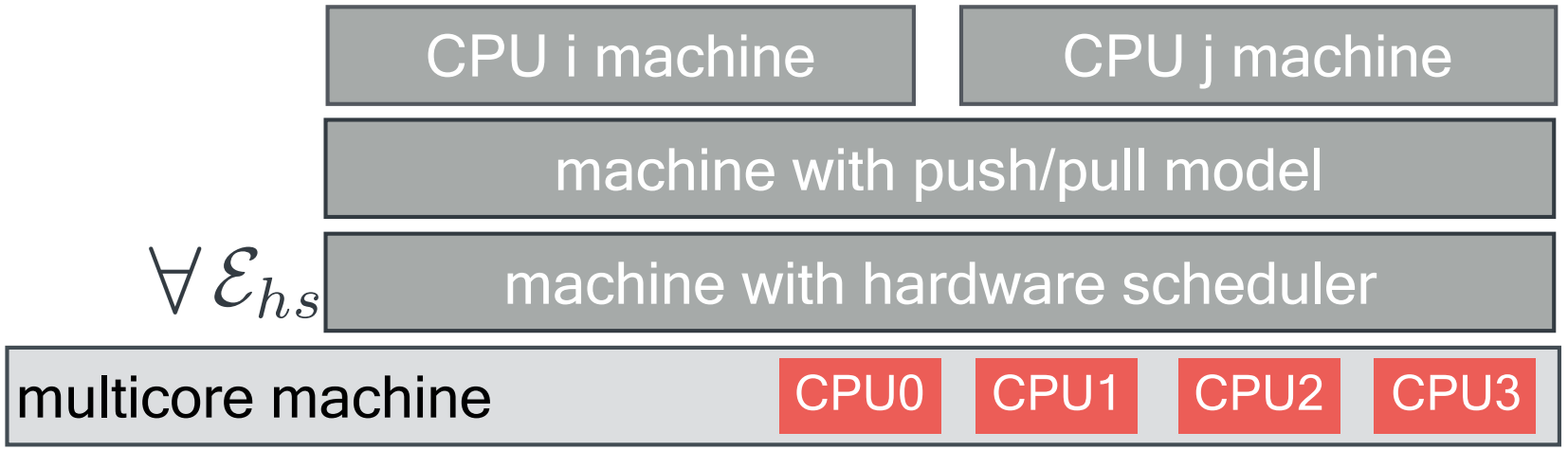


step 3: **environment context** model



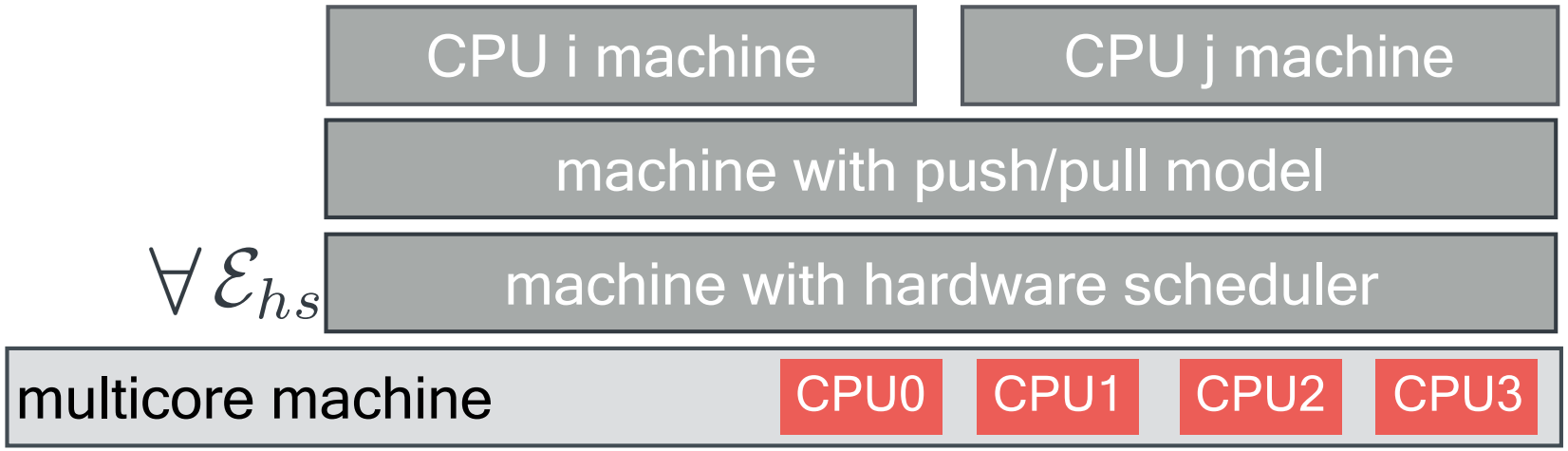


step 4: remove **unnecessary** interleaving



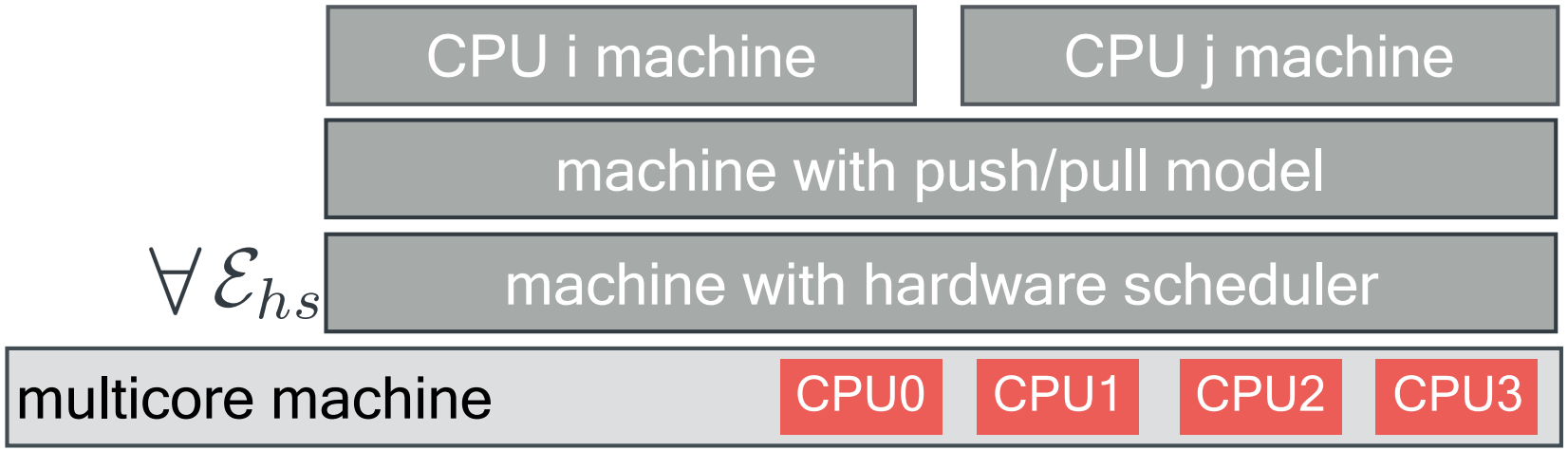
step 4: remove **unnecessary** interleaving

shuffle

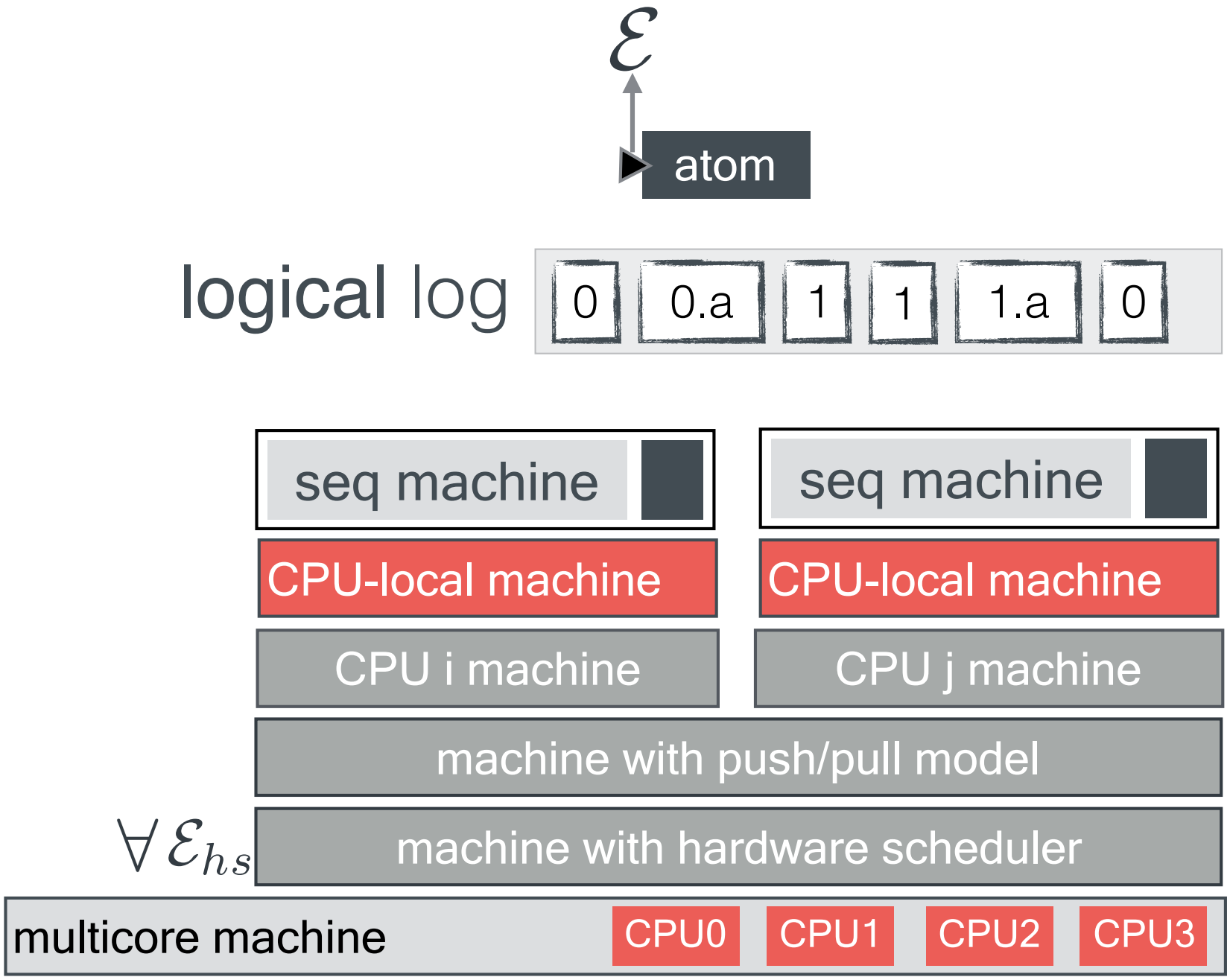


step 4: remove **unnecessary** interleaving

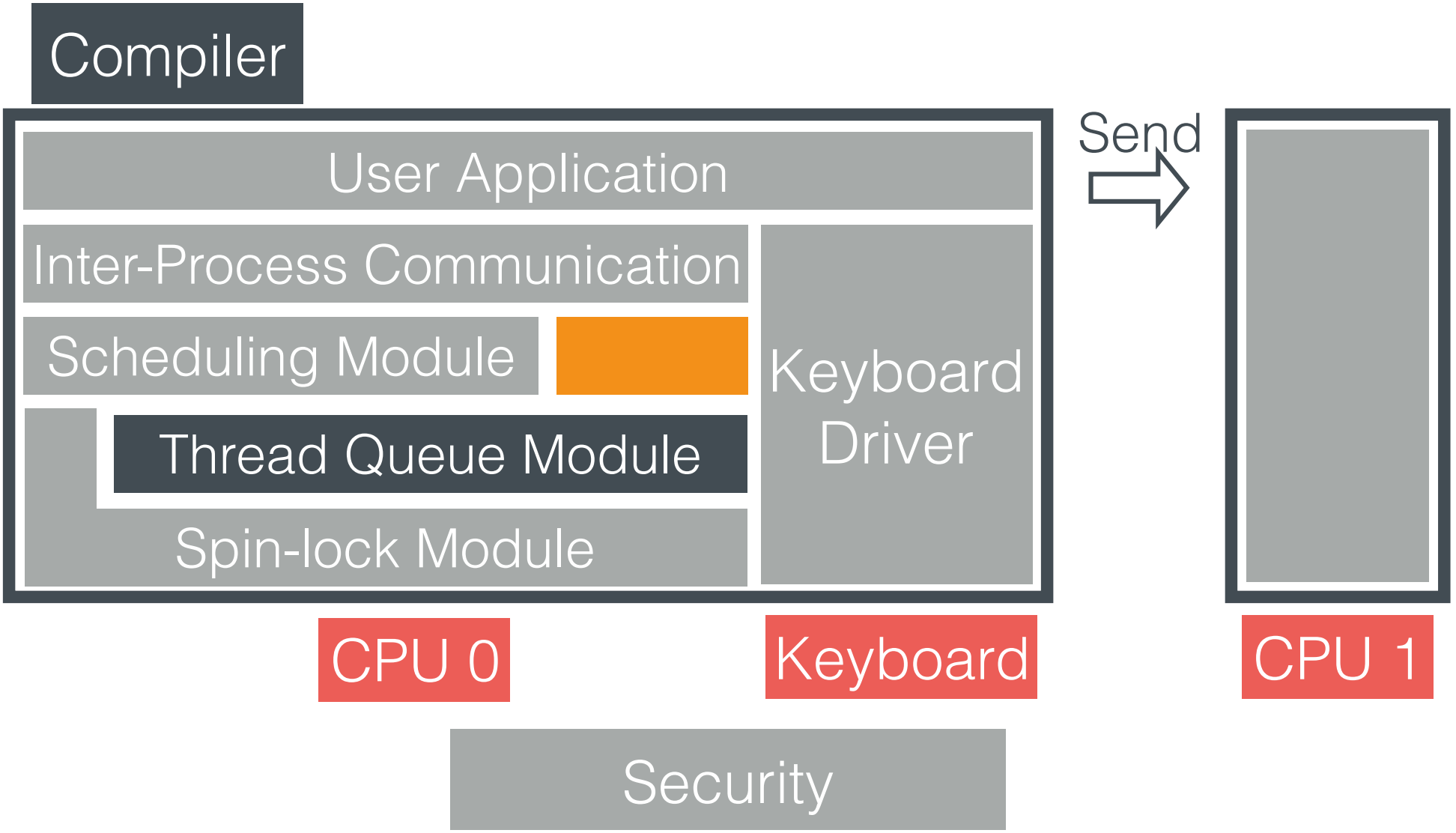
merge



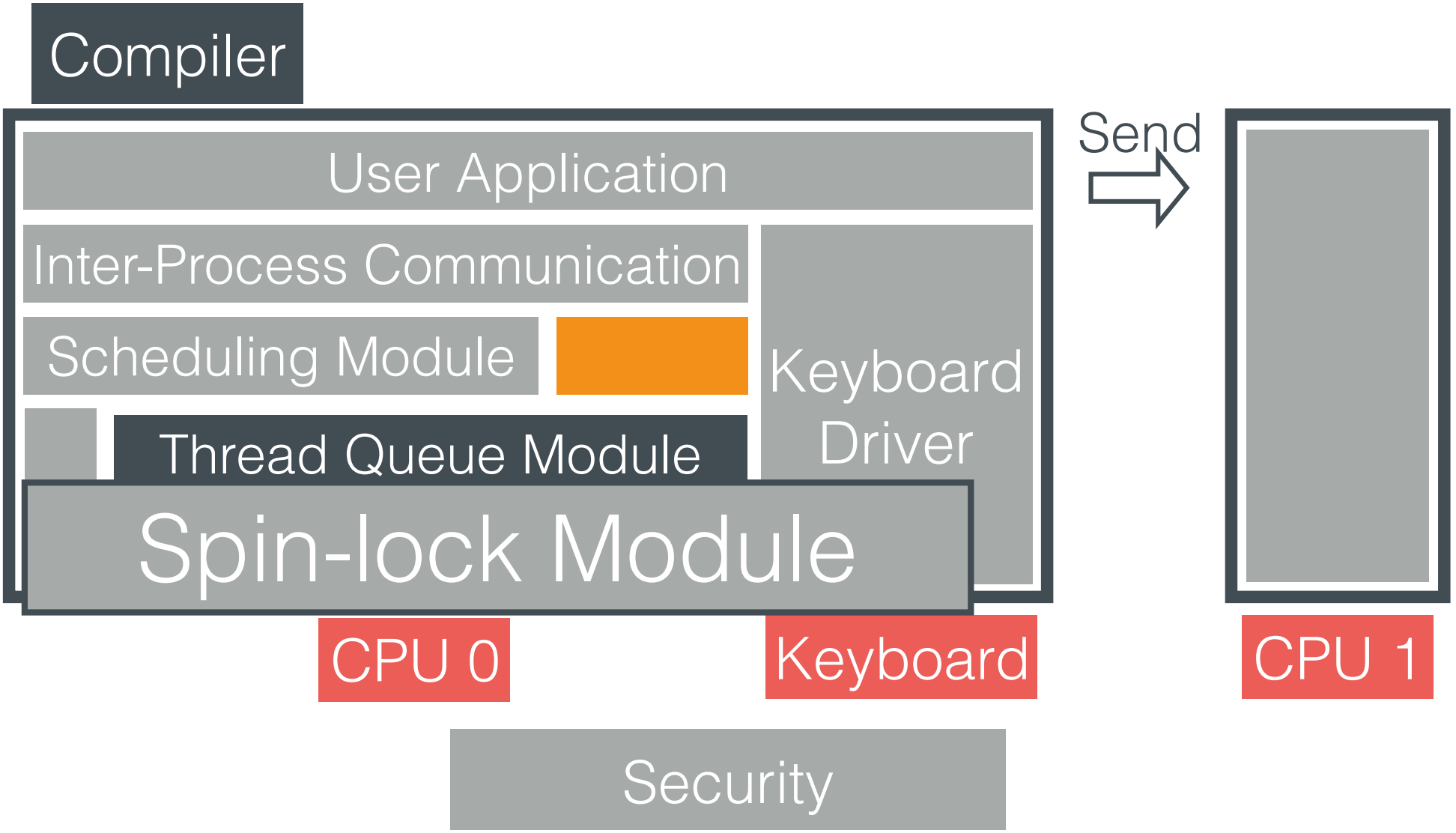
Machine Lifting



Build a Certified System

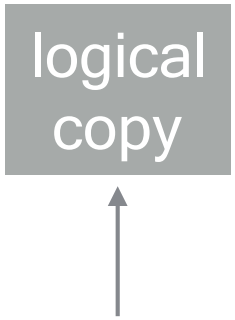


Build a Certified System

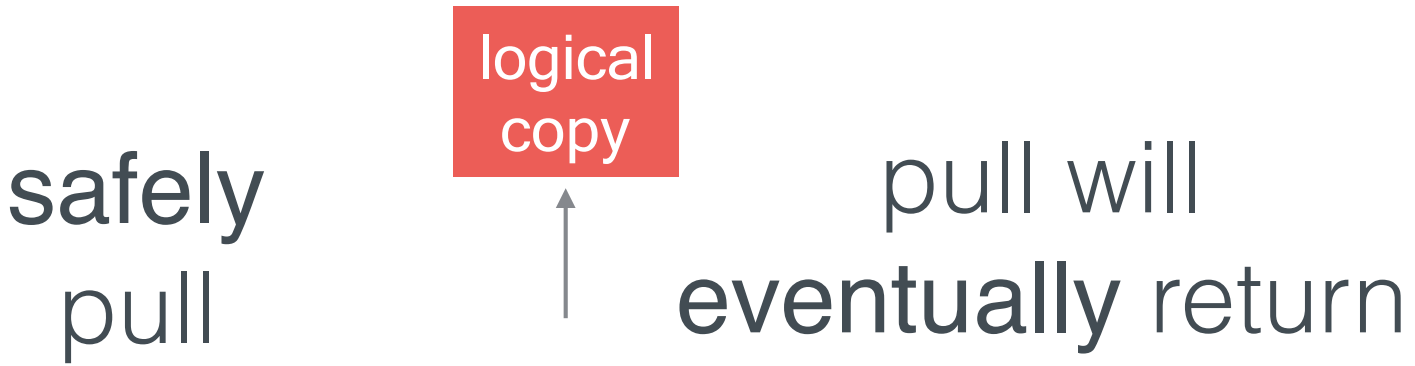


Acquire Lock Specification

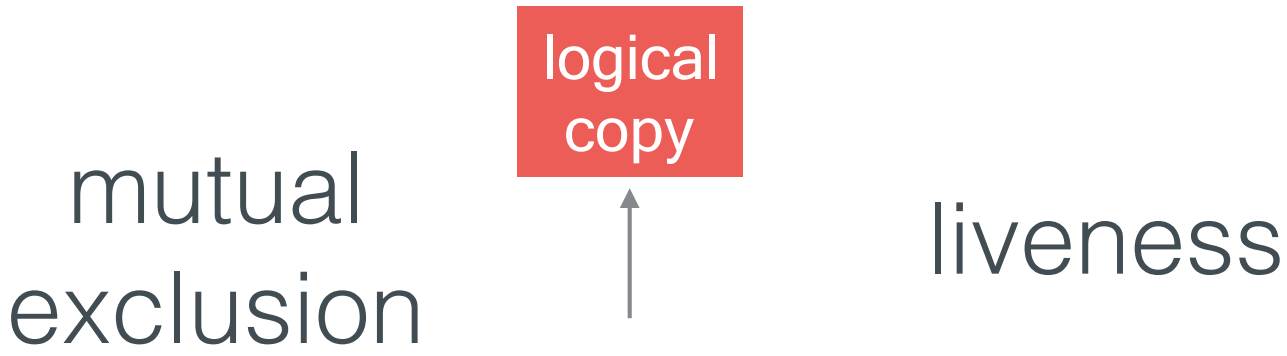
safely
pull



Acquire Lock Specification



Acquire Lock Specification



Example: Ticket Lock

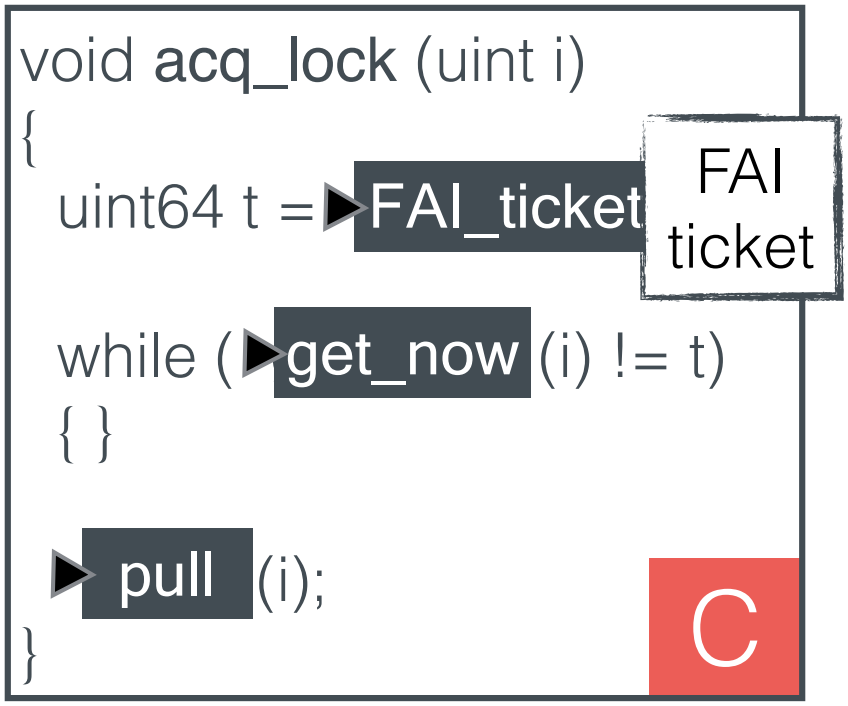
mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint64 t =   FAI_ticket  (i);

  while (  get_now  (i) != t)
  { }

     pull    (i);
}
```

C

Example: Ticket Lock

mutual exclusion + liveness



Example: Ticket Lock

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint64 t = FAI_ticket (i);

  while ( get_now (
  { }

  pull (i);
}
```

C

FAI ticket

get now

Example: Ticket Lock

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint64 t = FAI_ticket (i);

  while ( get_now (i
  { }

  pull (i);
}
```

C

FAI ticket

get now

get now

Example: Ticket Lock

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint64 t = FAI_ticket (i);

  while ( get_now (i) != t)
  { }

  pull (i)
}
```

pull

C

FAI ticket

get now

get now

Example: Ticket Lock

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint64 t = FAI_ticket (i);

  while (get_now (i) != t)
  { }

  pull (i);
}
```

C

Example: Ticket Lock

mutual exclusion + liveness

```
void acq_lock (uint i)
{
  uint64 t = FAI_ticket (i);

  while ( get_now (i) != t)
  { }

  pull (i);
}
```

C

unique t

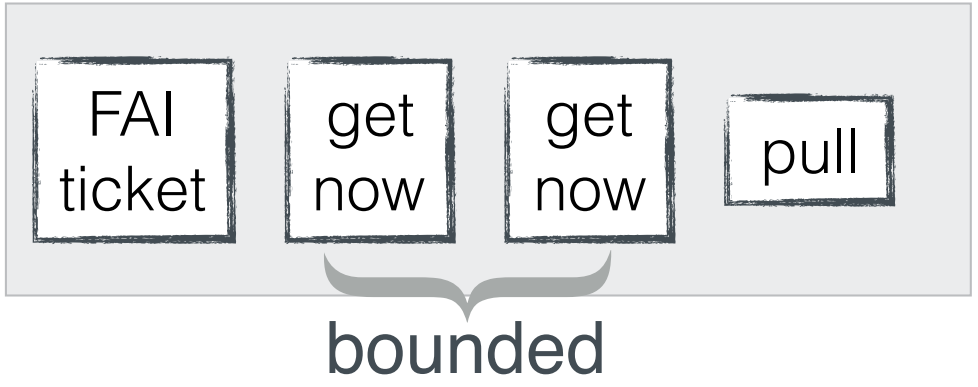
÷ #CPUs < 2^{64}

Example: Ticket Lock

mutual exclusion + **liveness**

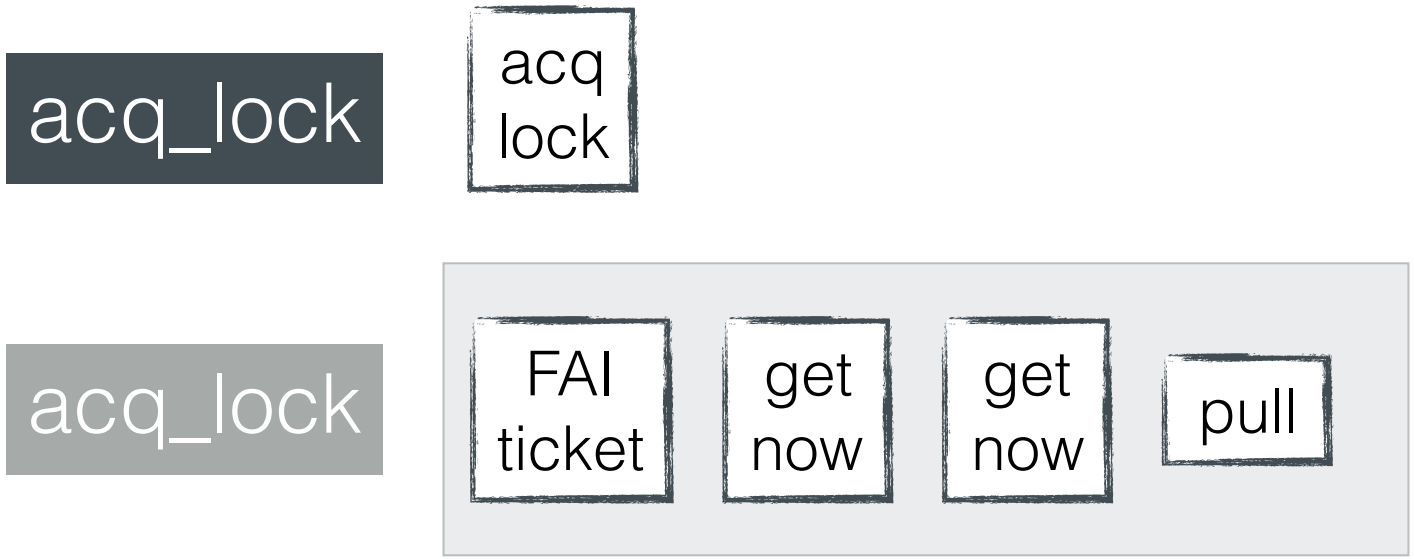
```c
void acq_lock (uint i)
{
  uint64 t = FAI_ticket (i);

  while (get_now (i) != t)
  { }

  pull (i);
}
```

C



- #CPUs is bounded
- a **fair** hardware scheduler
- lock holders will **release** lock

Example: Ticket Lock



Example: Ticket Lock

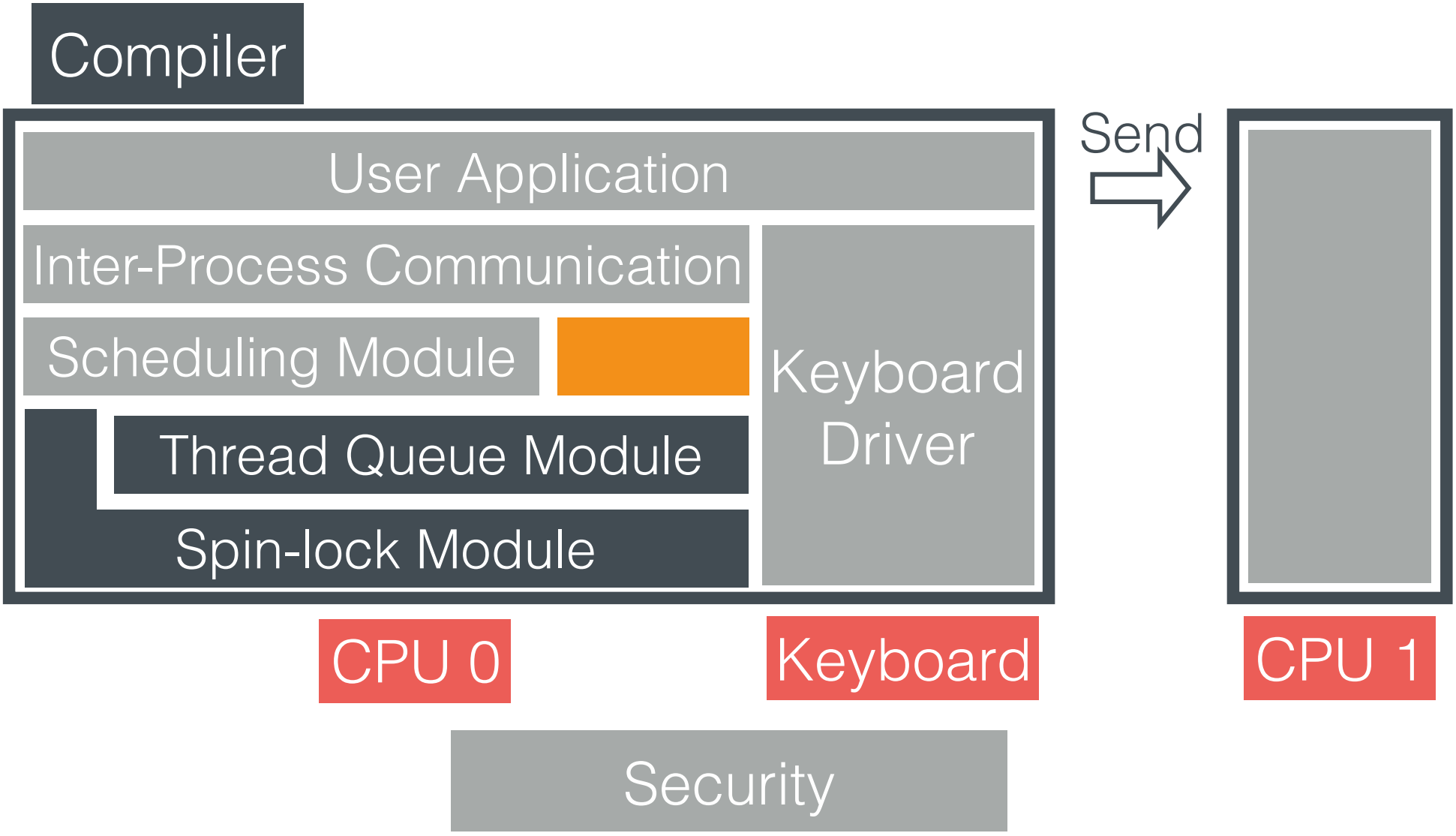
```
void acq_lock (uint i)
{
  uint64 t = FAI_ticket (i);

  while ( get_now (i) != t)
  { }

  pull (i);
}
```

C

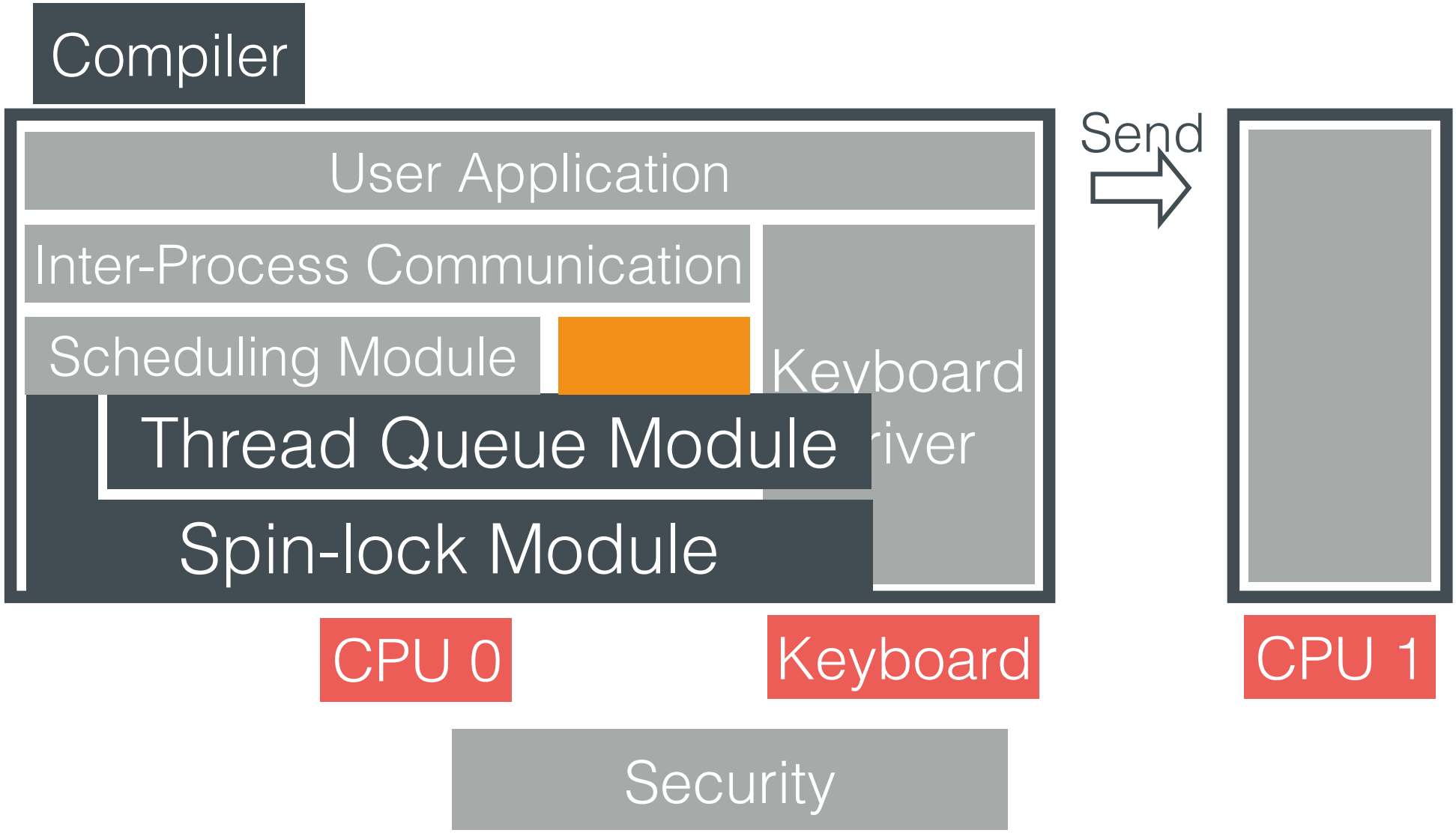
bug in the original implementation

mutual exclusion will be violated when there is an **integer overflow** for t

Build a Certified System



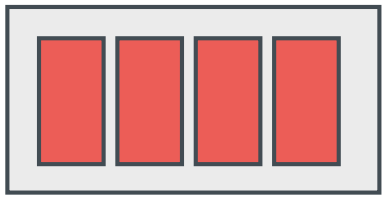
Build a Certified System



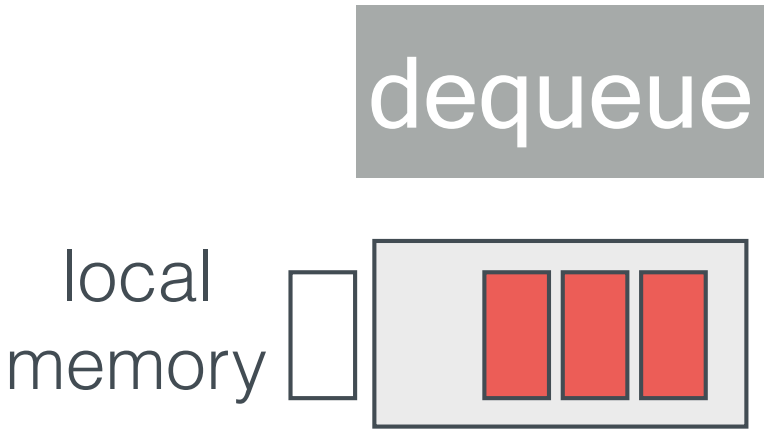
Example: Shared Thread Queue

dequeue

local
memory



Example: Shared Thread Queue



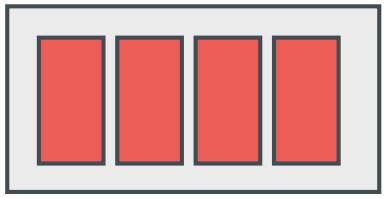
Example: Shared Thread Queue

acq lock

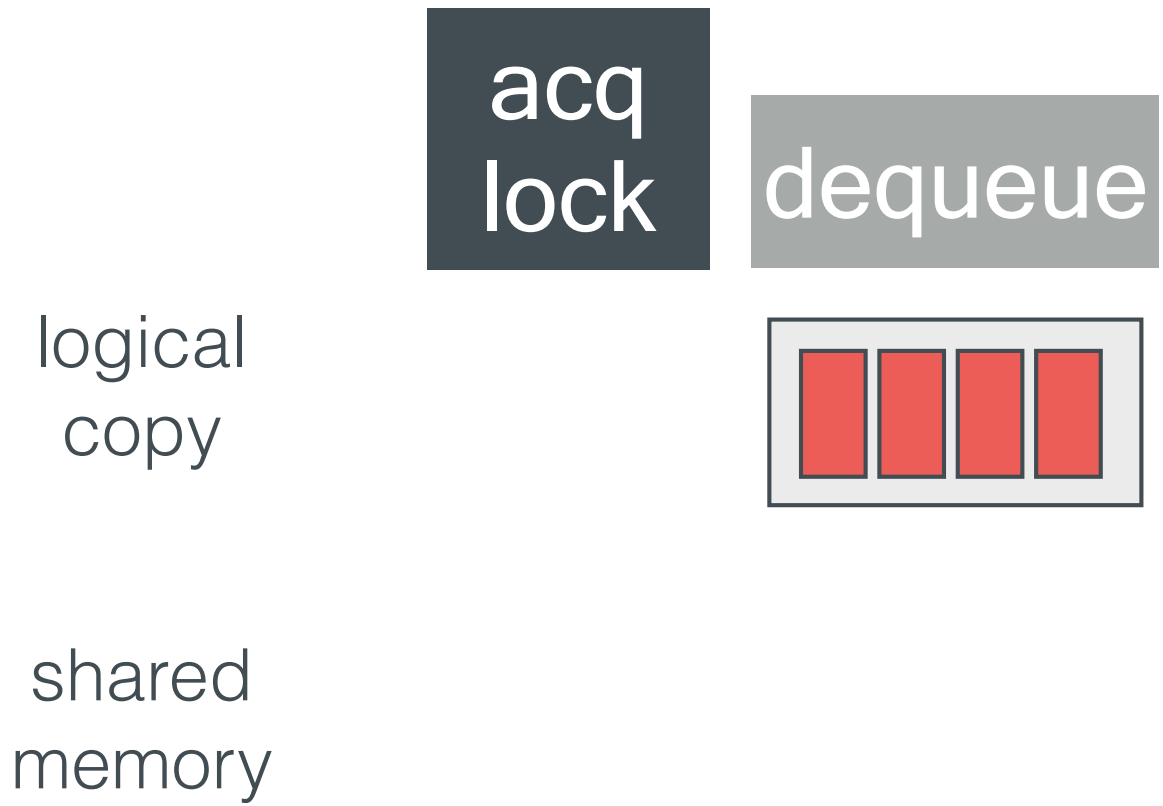
dequeue

logical copy

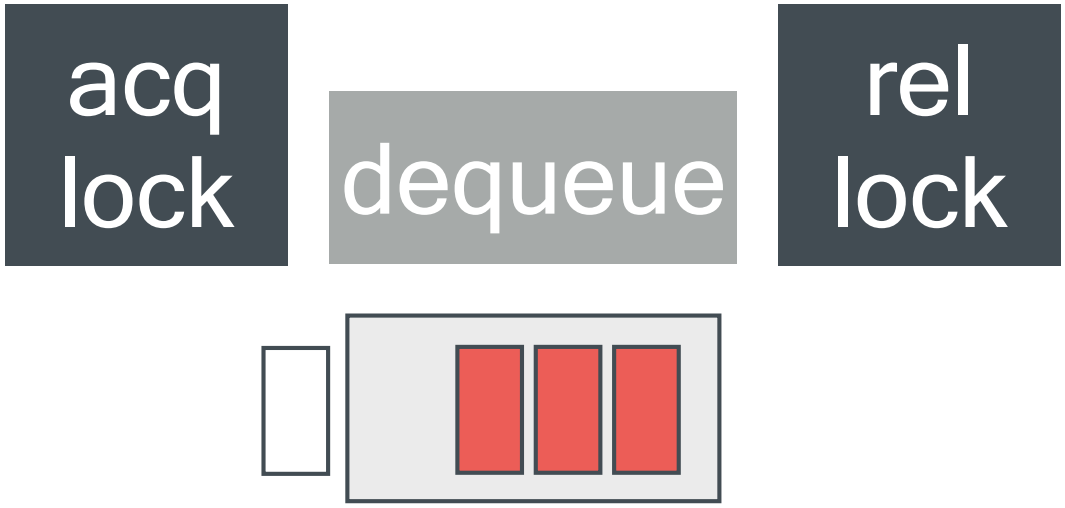
shared memory



Example: Shared Thread Queue



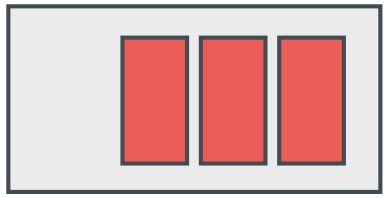
Example: Shared Thread Queue



Example: Shared Thread Queue

logical copy

shared memory

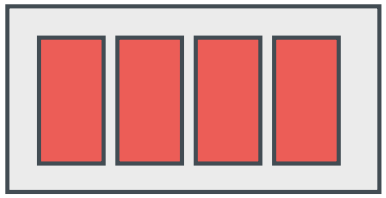


Example: Shared Thread Queue

dequeue

deq

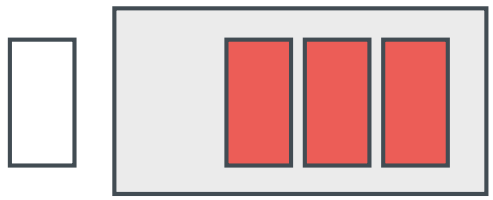
shared
memory



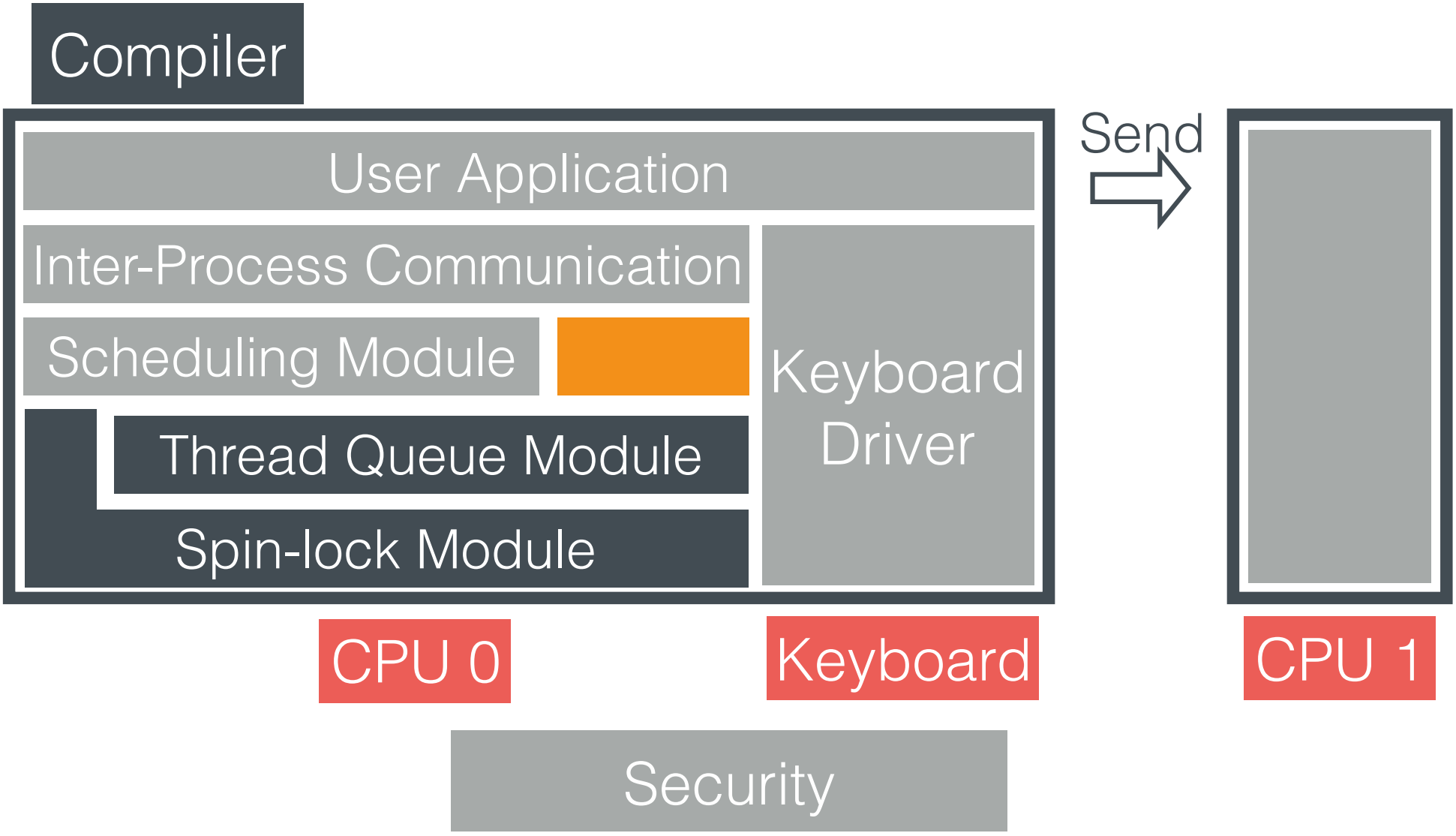
Example: Shared Thread Queue

dequeue

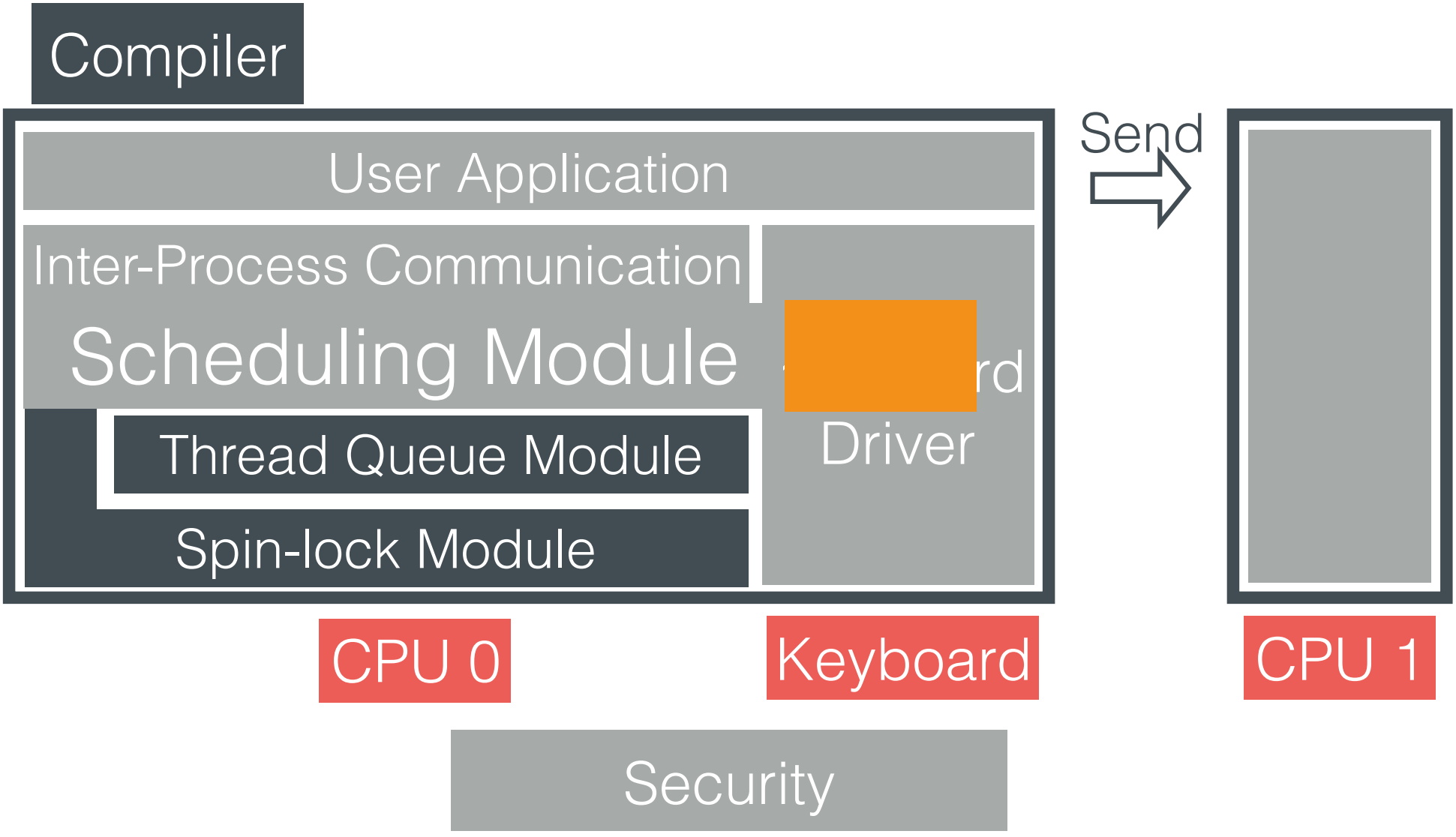
shared
memory



Build a Certified System



Build a Certified System



Thread-Local Machine

```
void yield ()
{
  uint t =  tid();
  …
  enq (t, rdq());

  uint s =  deq (rdq());
  …
  context_switch (t, s)
}
```

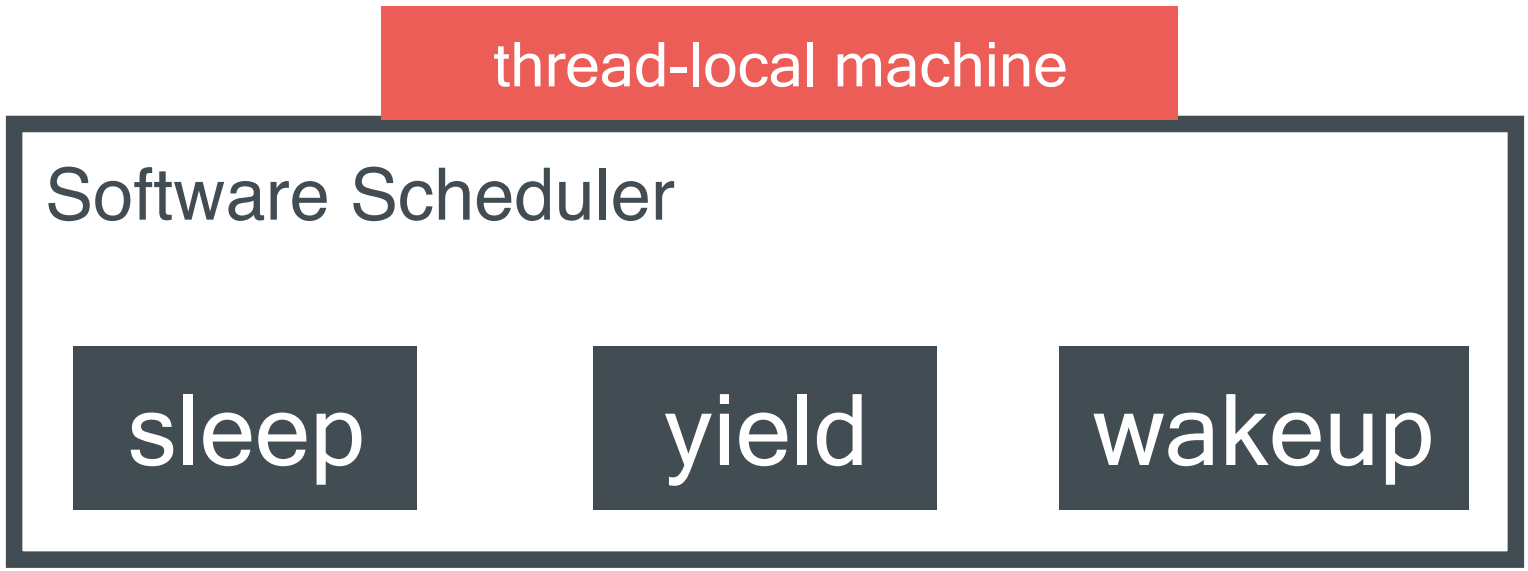
Thread-Local Machine

IPC

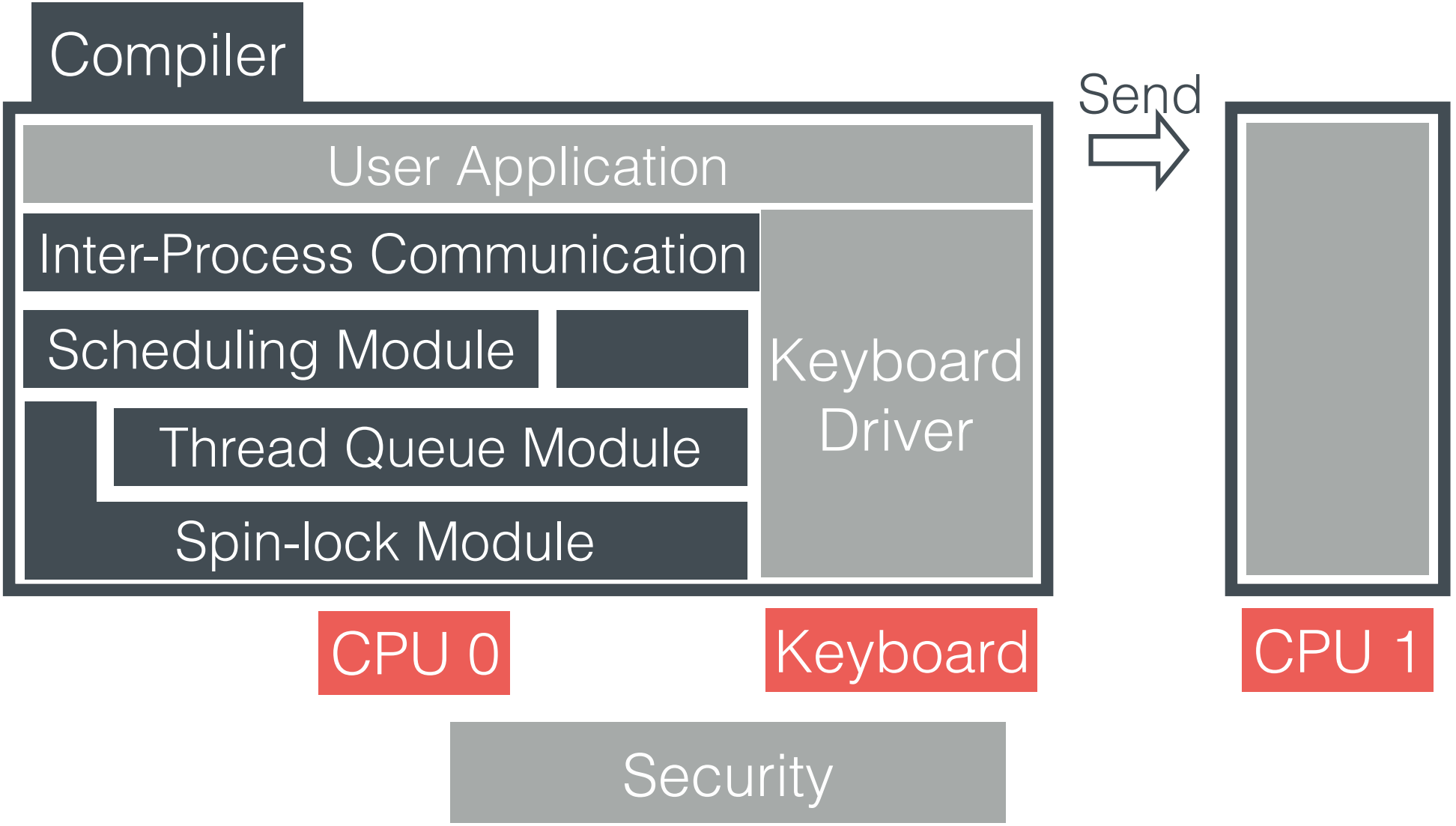
CV

Found hard **bugs** in the popular OS textbook

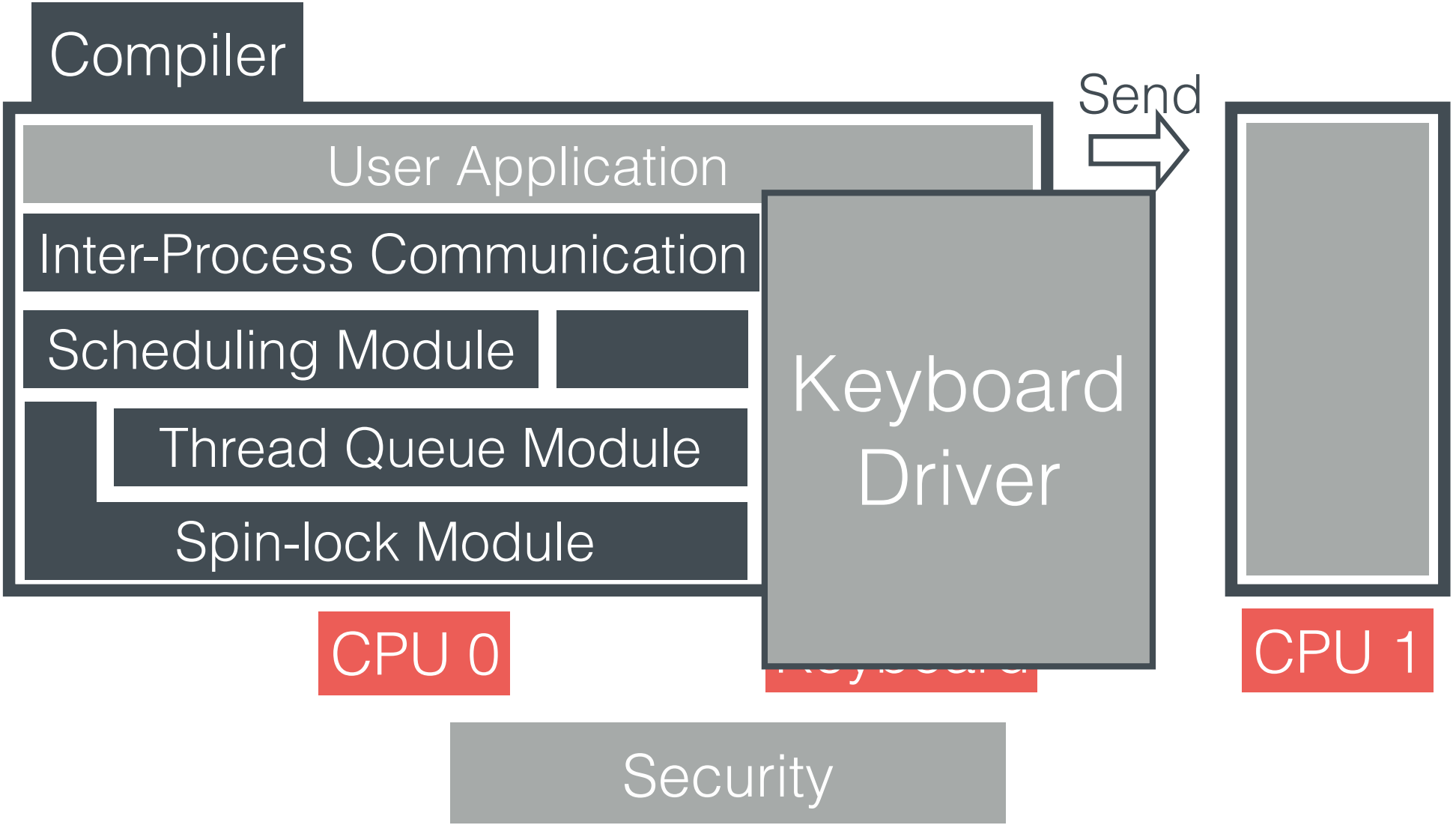
[Operating Systems Principles and Practice 2011]



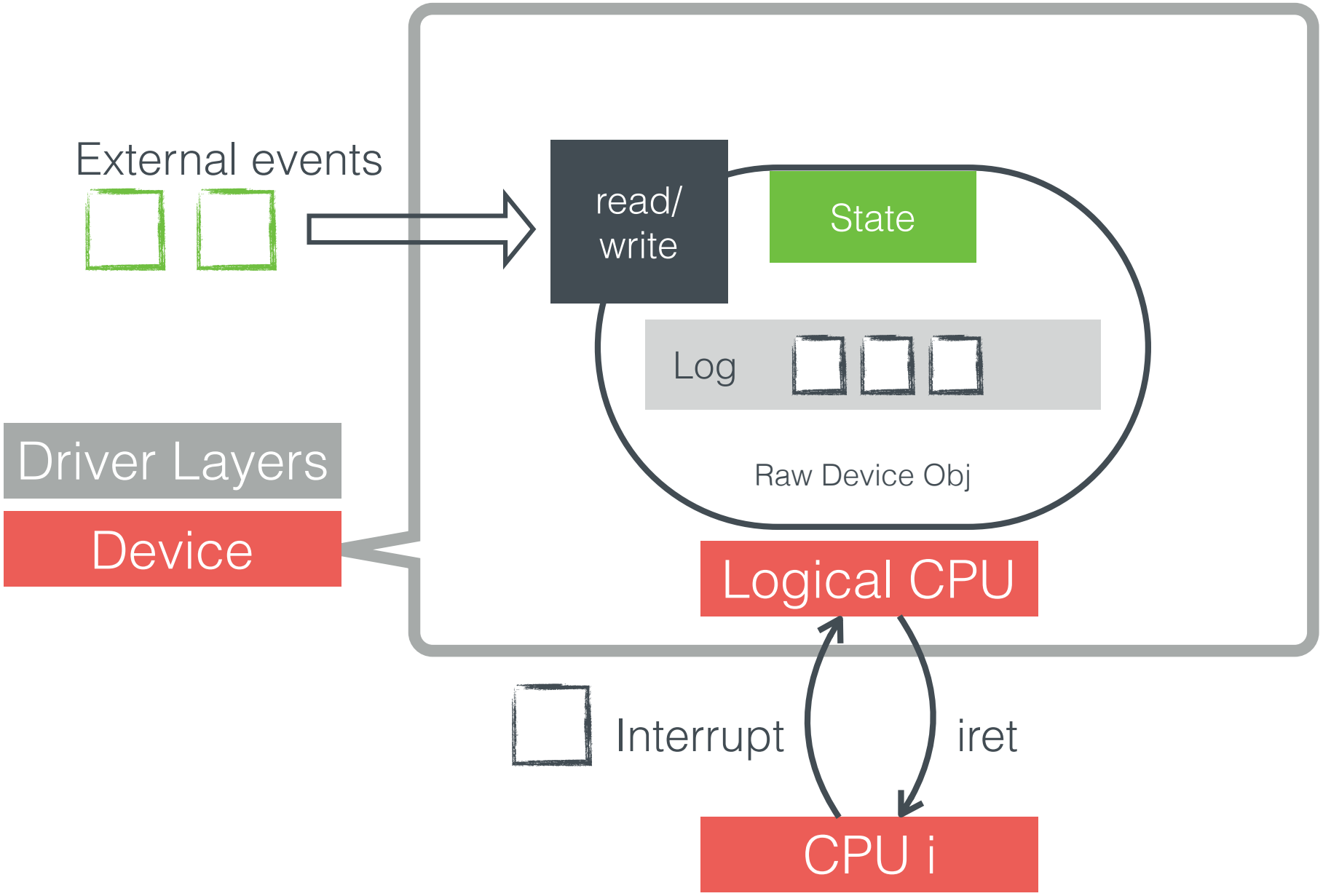
Build a Certified System



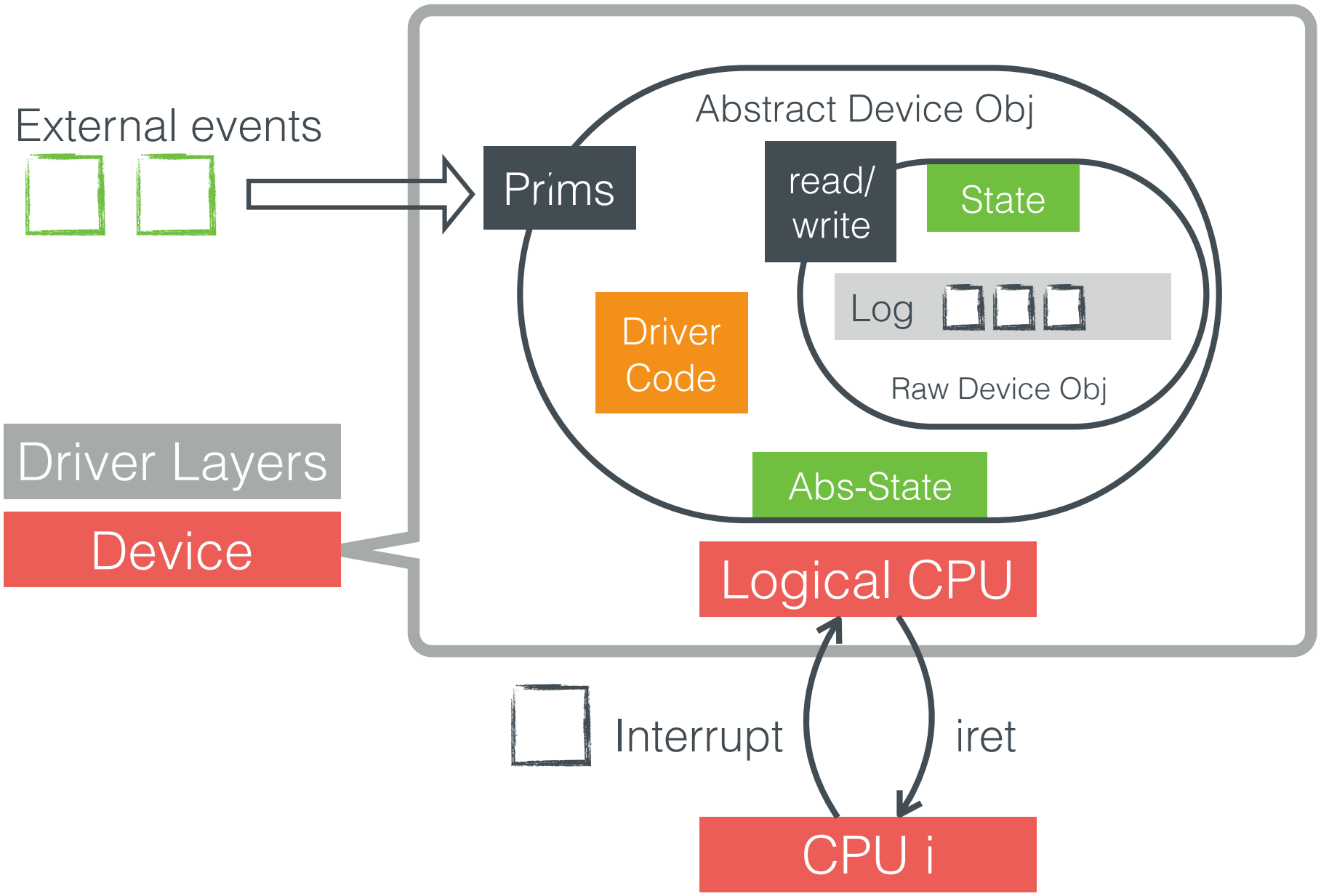
Build a Certified System



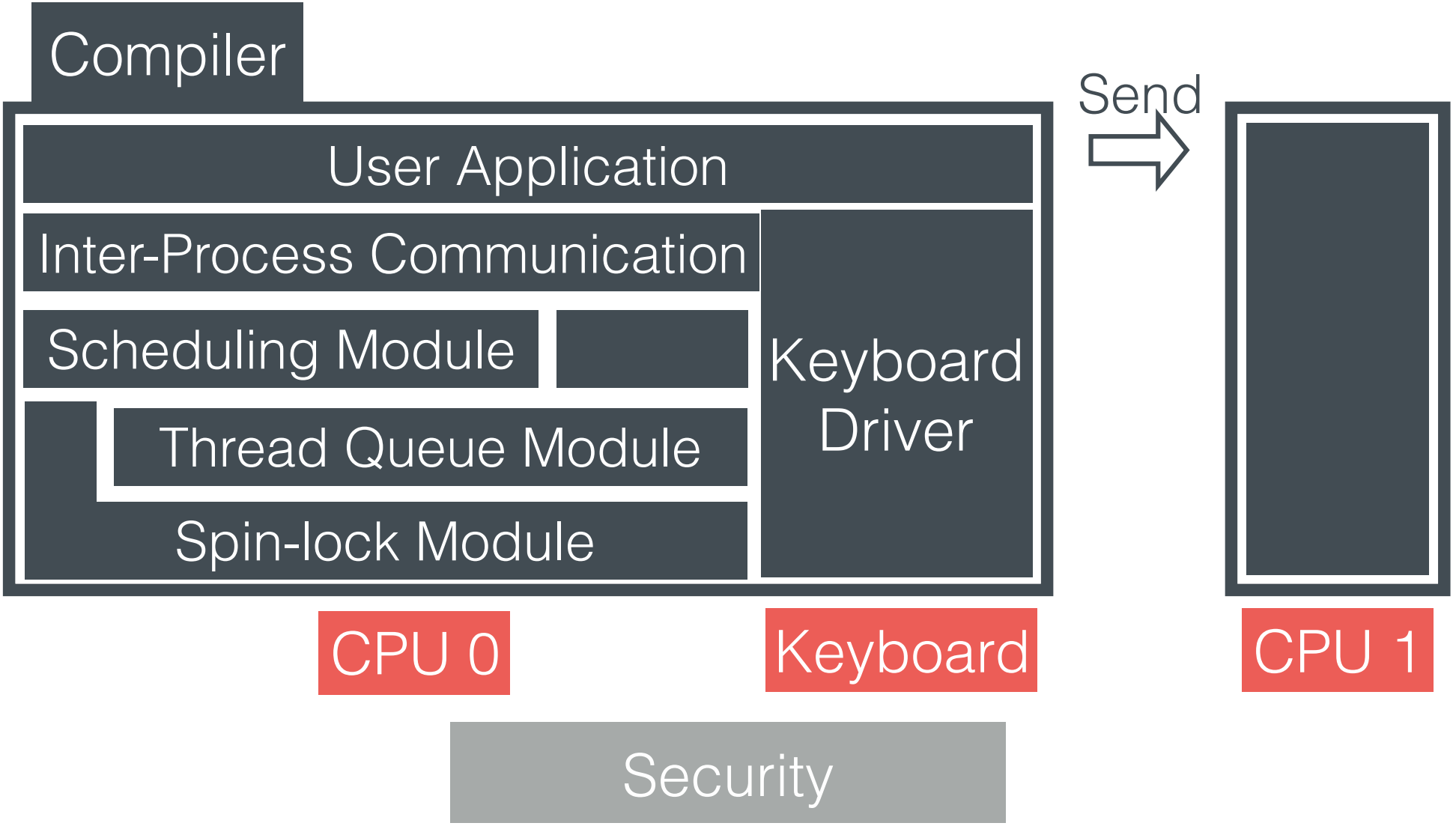
Device Driver [PLDI16'a]



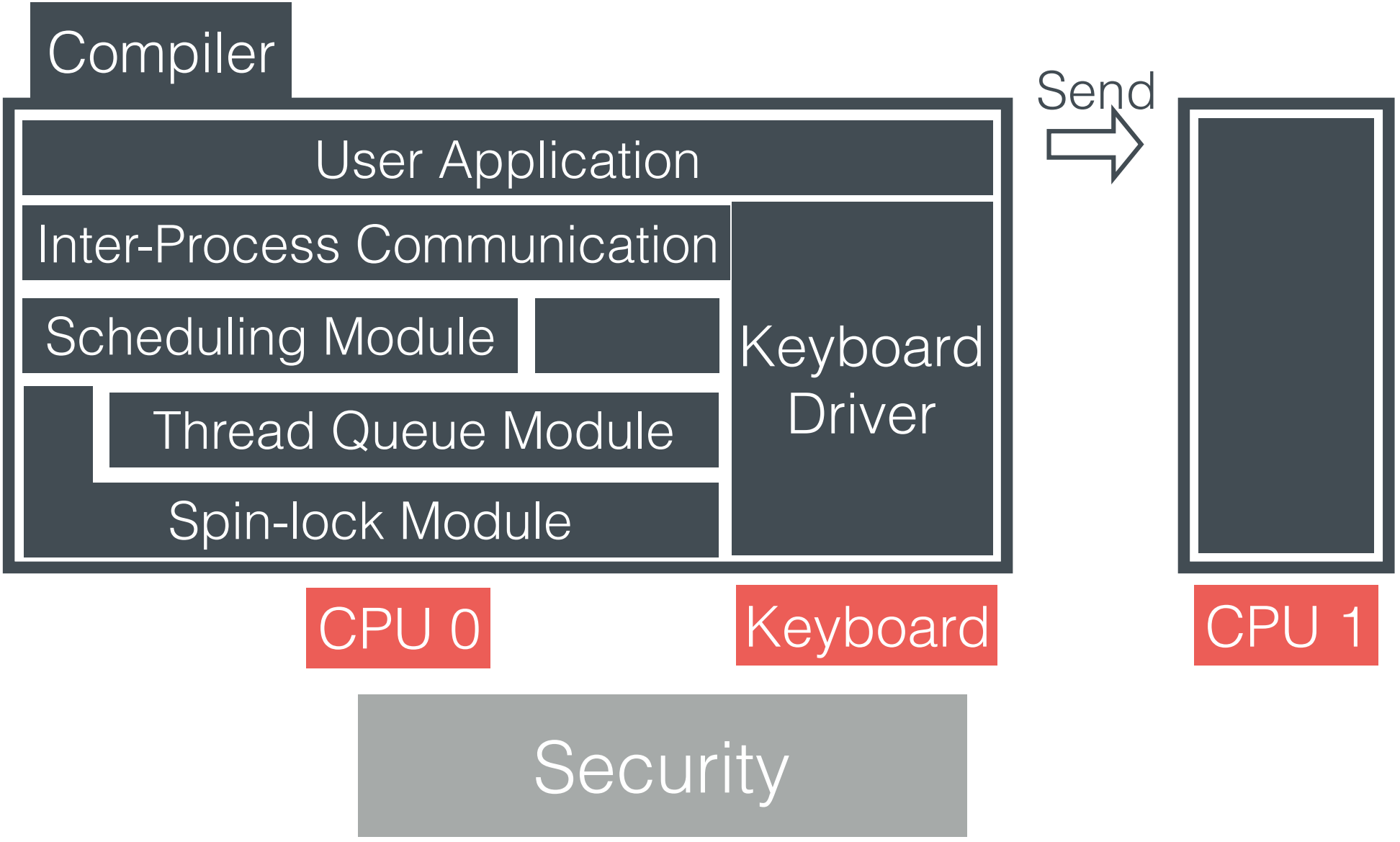
Device Driver [PLDI16'a]



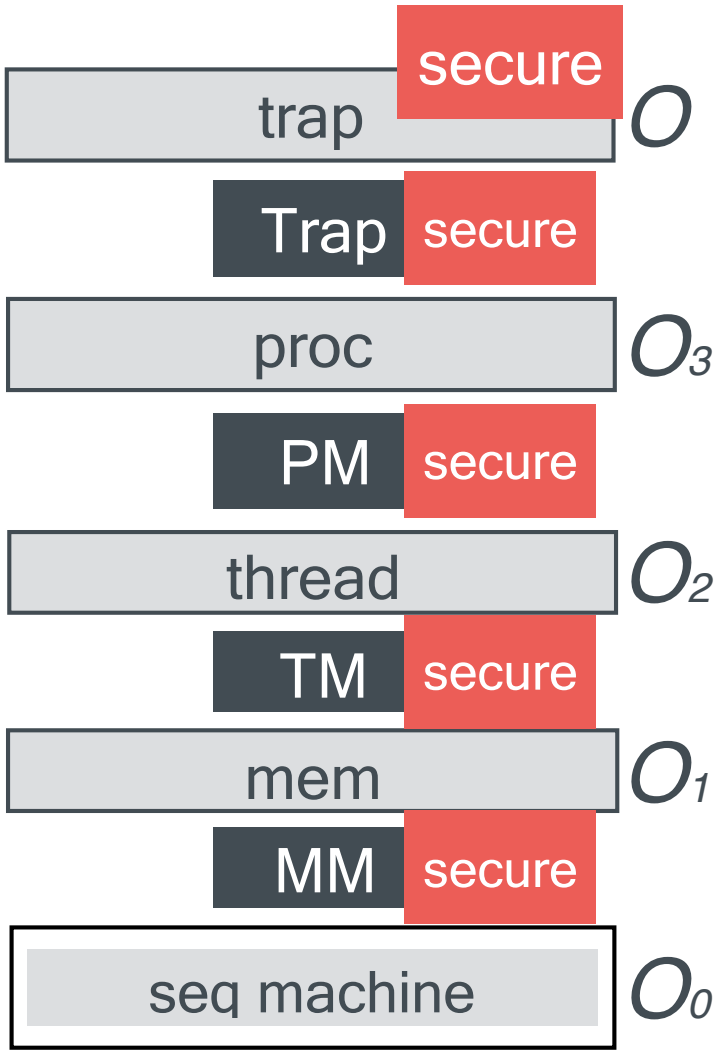
Build a Certified System



Build a Certified System



End-to-End Security [PLDI16'b]

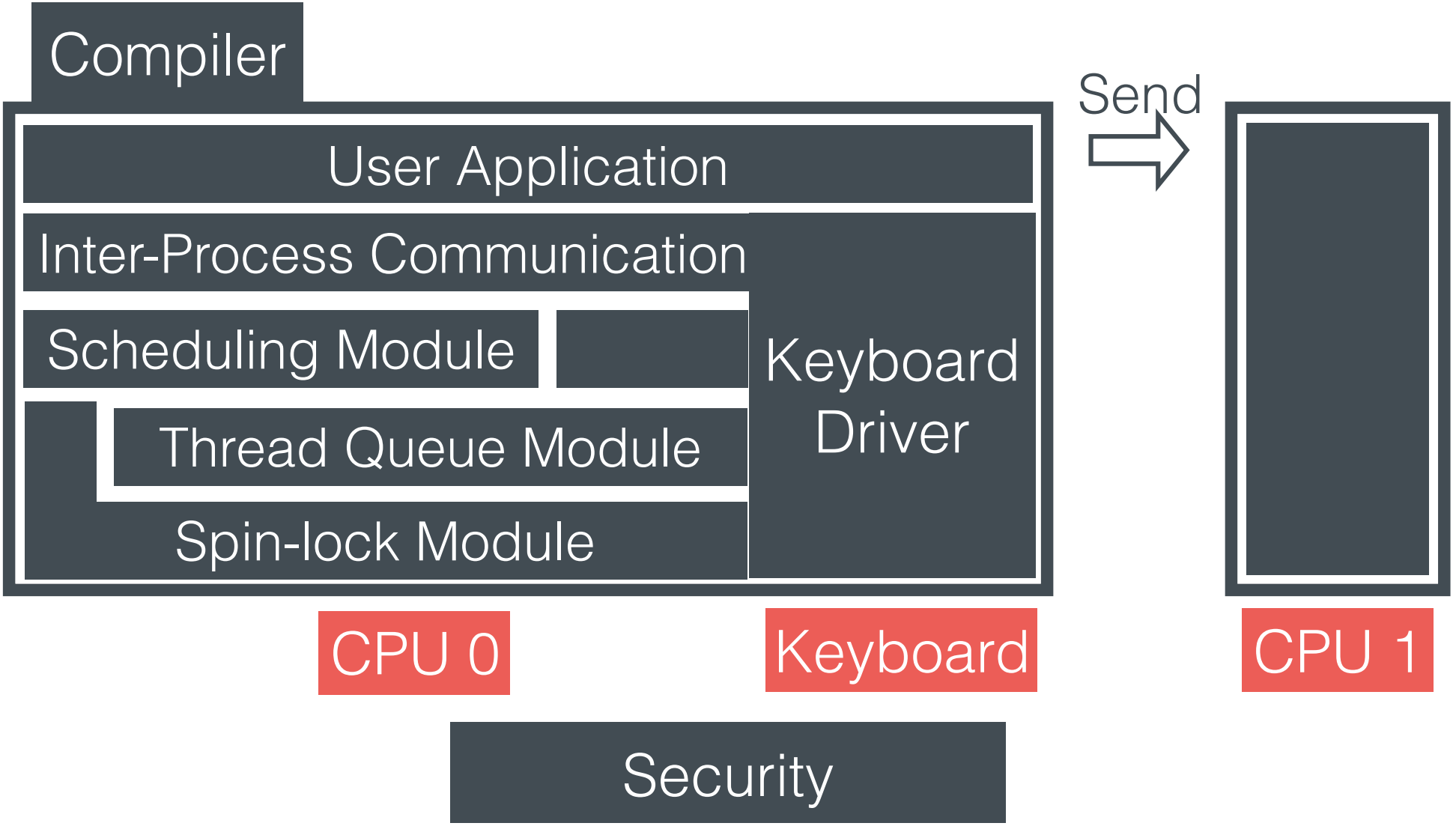


Observation function O

- **specify and prove** general security policies with **declassification**
- **security-preservation** simulation
- **non-interference**

found **security-bugs**: spawn, palloc,…

Build a Certified System



Summary: Certified OS

CertiKOS is the first fully certified OS kernel that is done economically (< 3 person years), proves more properties, runs on concurrent HW, and is truly extensible

Still very high barriers of entry:

(1) OS kernel development is very difficult

(2) Formal specifications and proofs are hard to build

(3) Need intimate programming language expertise to succeed

- These are three completely different communities
- Most people can only do one out of the above three.
- The Yale team has been working on all three for >15 years

Summary: OS Landscape (Nov 2017)

Desktop: Linux, macOS, Windows, ChromeOS, freeBSD, …
Hypervisor/Cloud: Linux KVM & Docker, VMWare, Xen, …
Mobile: Android (Linux), iOS, …
Embedded: Embedded Linux, VxWorks, QNX, LynxOS, …

- All of them are bloated, old, and contain many bugs
- Urgently need new OSes for emerging platforms & apps
  (IoTs, Drones, Self-Driving Cars, Cloud, NetworkOS, Blockchains, …)

OS evolution has reached an inflection point:

Need a certified OS that provides security, extensibility, performance, and can work across multiple platforms.