

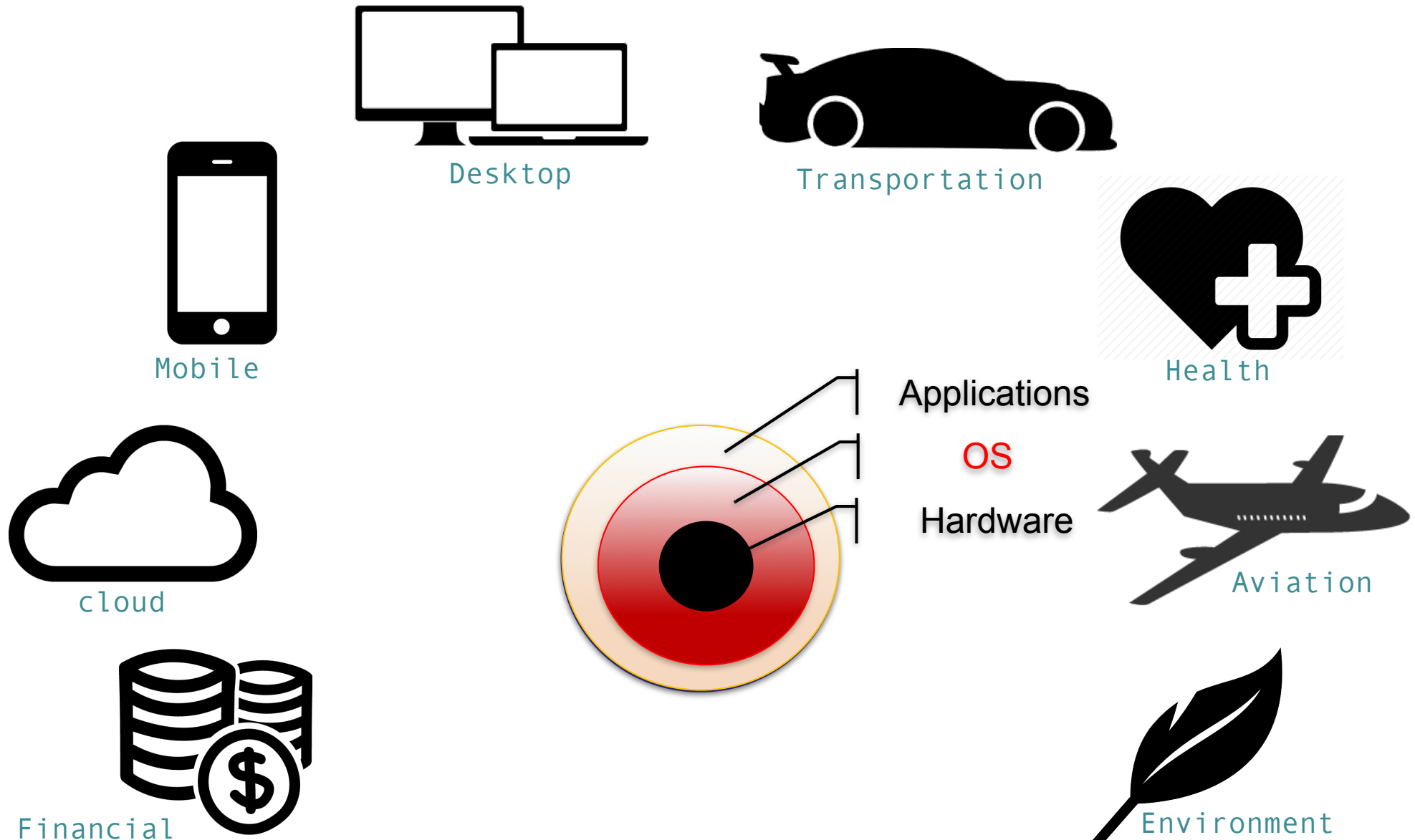
CertiKOS: A Breakthrough toward Hacker-Resistant Operating Systems



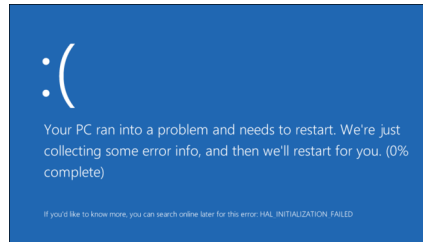
Zhong Shao
Yale University
January 25, 2018

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.

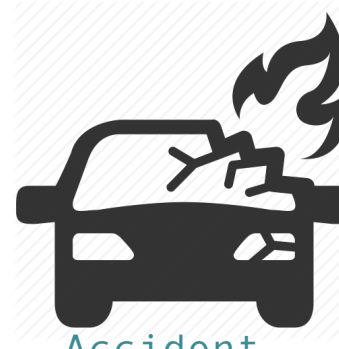
Motivation



Motivation



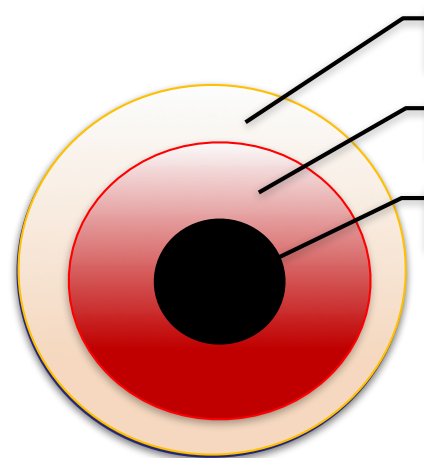
Crash



Accident



Mobile



Applications

OS

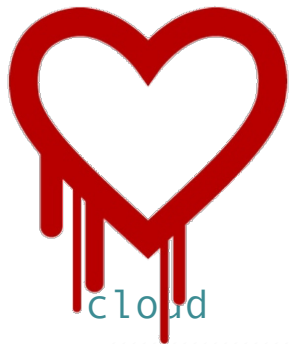
Hardware



Life



Loss



cloud



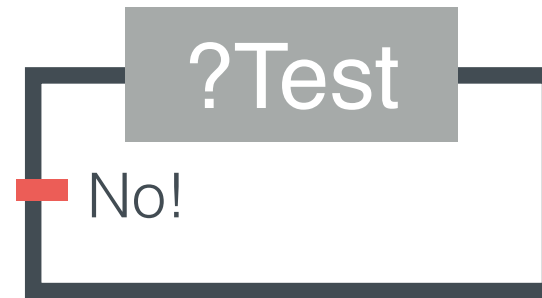
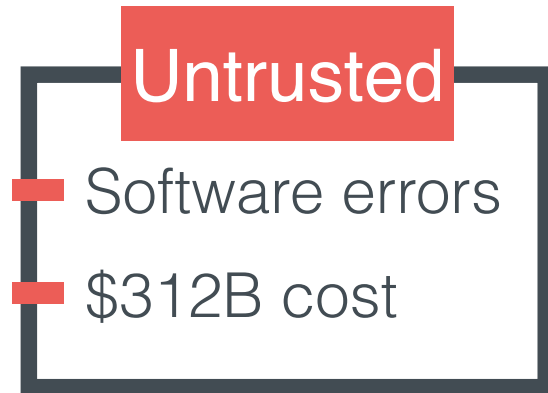
Financial



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

—NSF SFM Report[2016]

Motivation

System Software Runs Everywhere

Untrusted

- Software errors
- \$312B cost

?Test

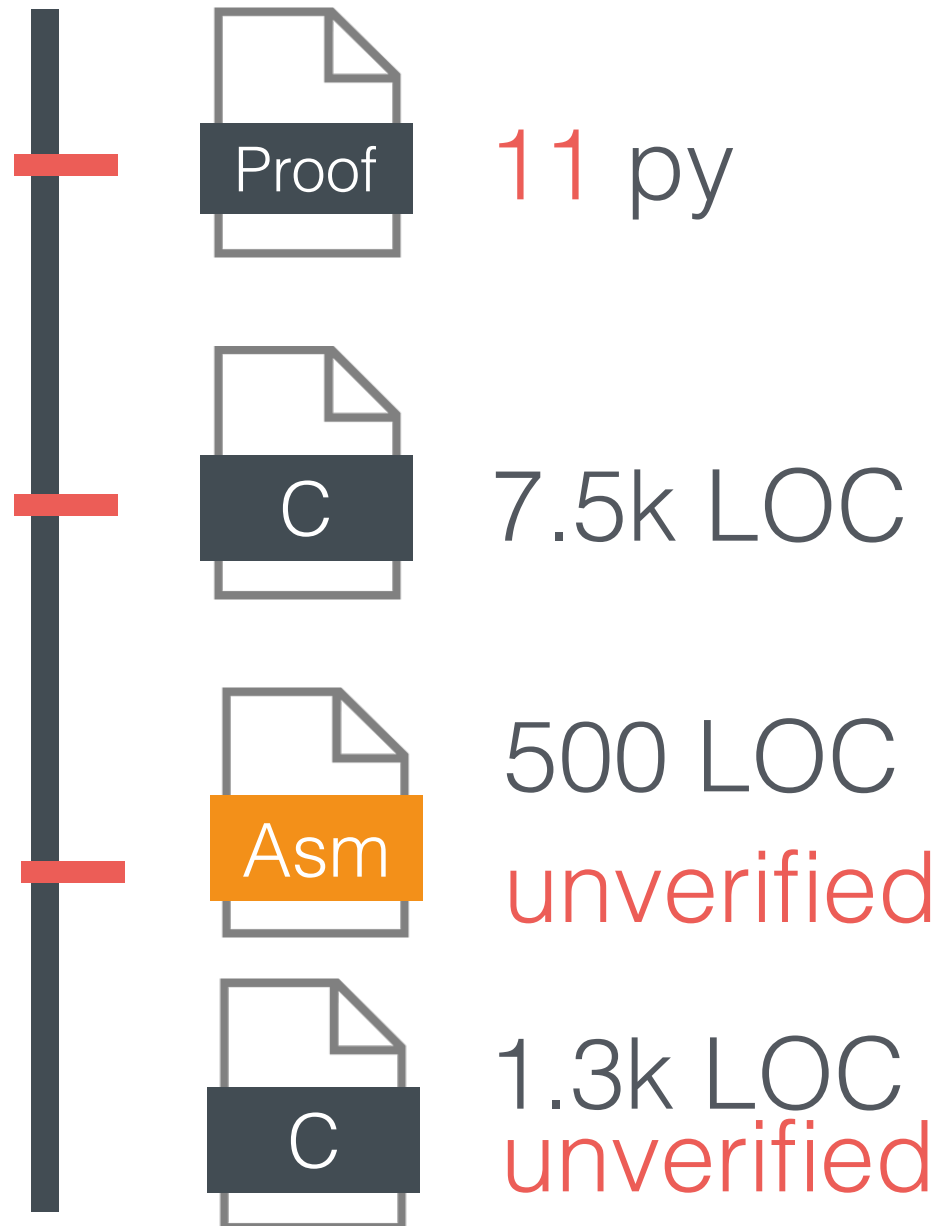
- No!

Formal Verification

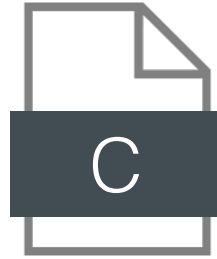
- Challenges?

Challenges: huge proof efforts

seL4
[SOSP'09]



Challenges: Compositionality



Abstraction Gap



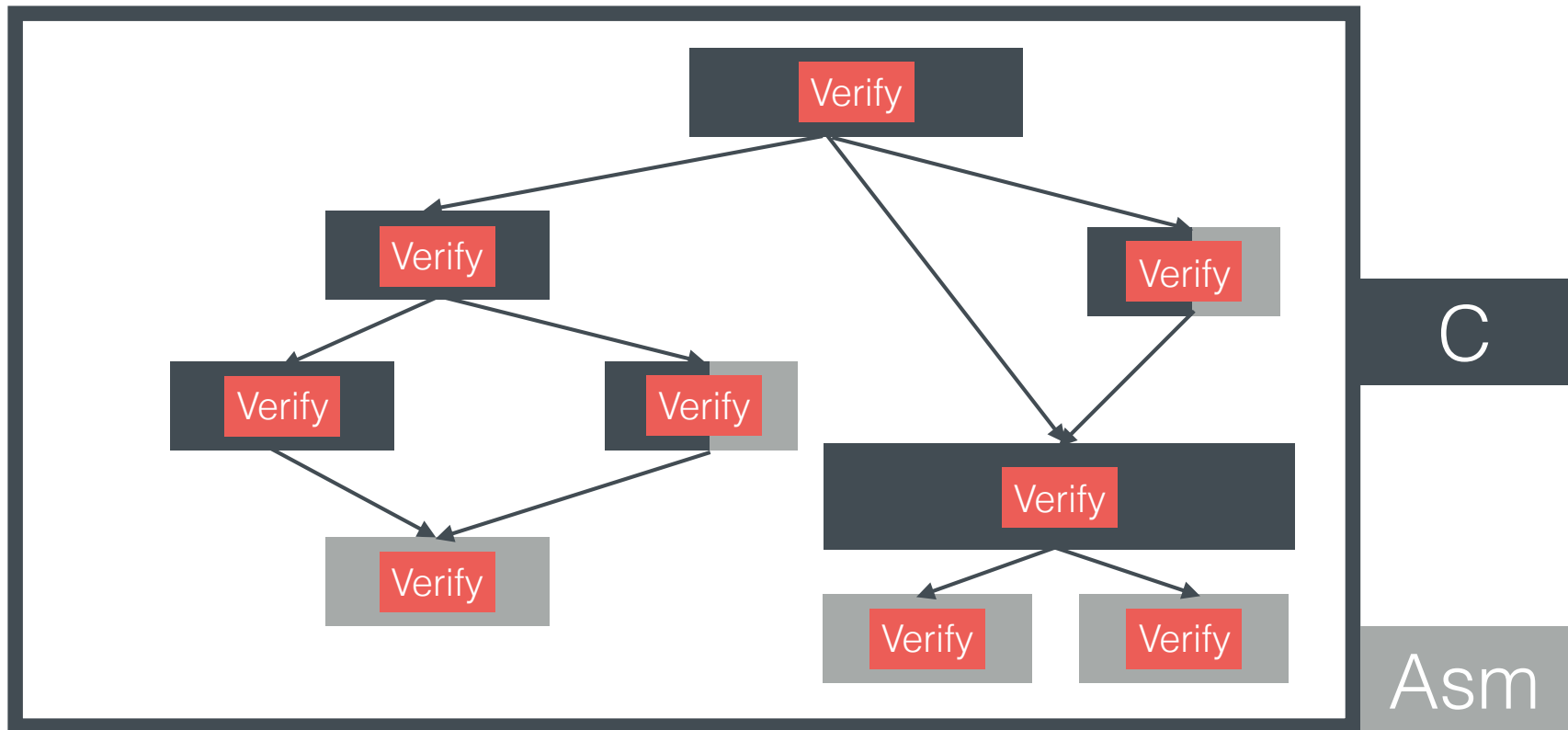
Challenges: Compositionality

A Complex System



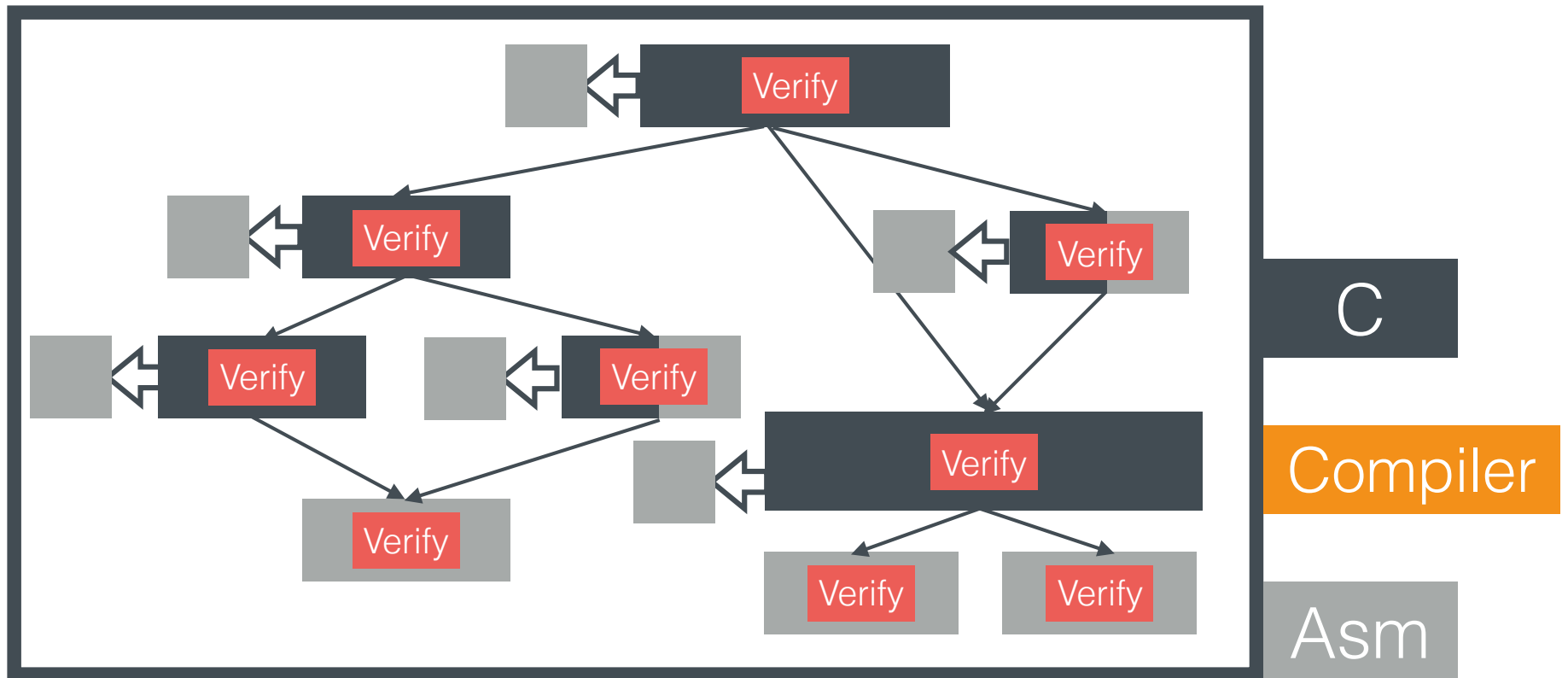
Challenges: Compositionality

A Complex System



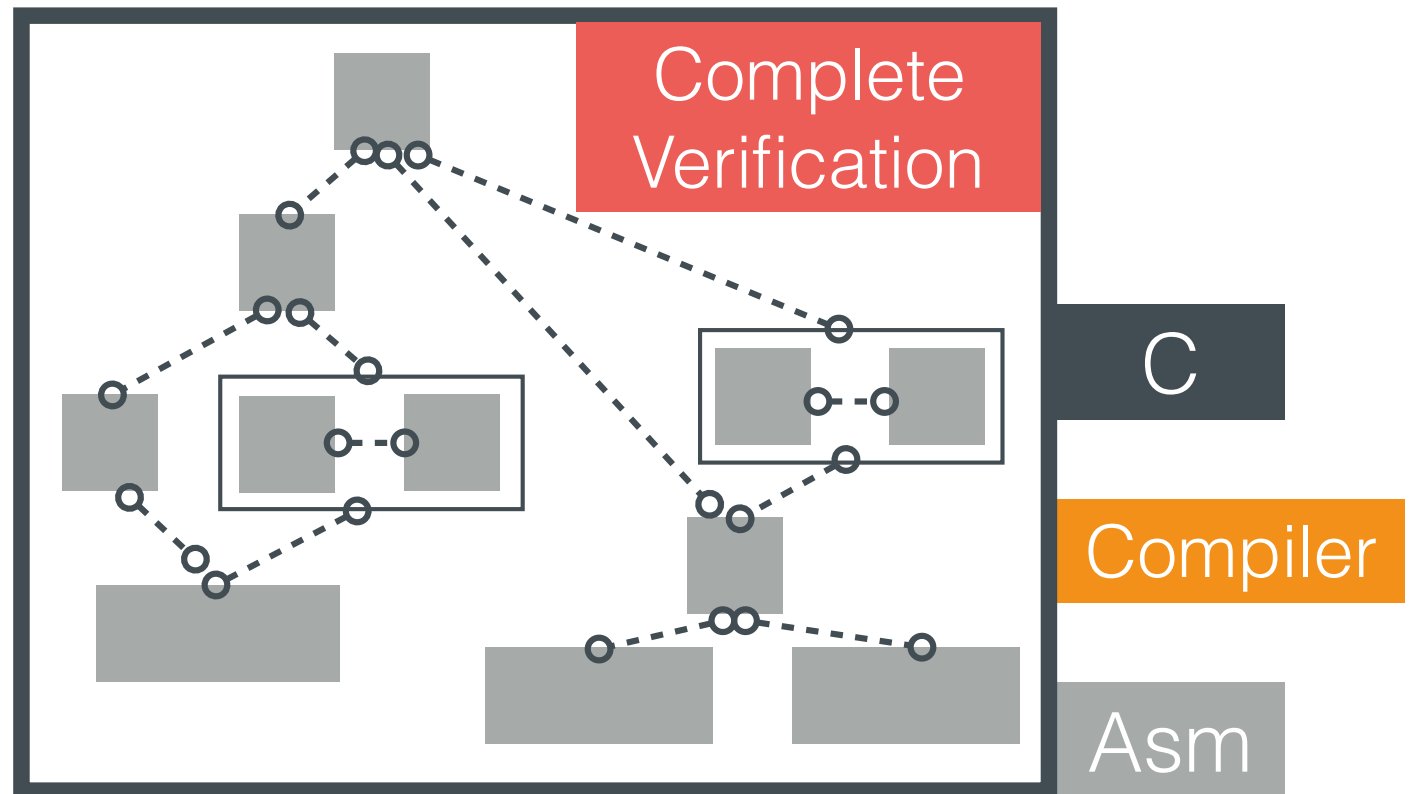
Challenges: Compositionality

A Complex System

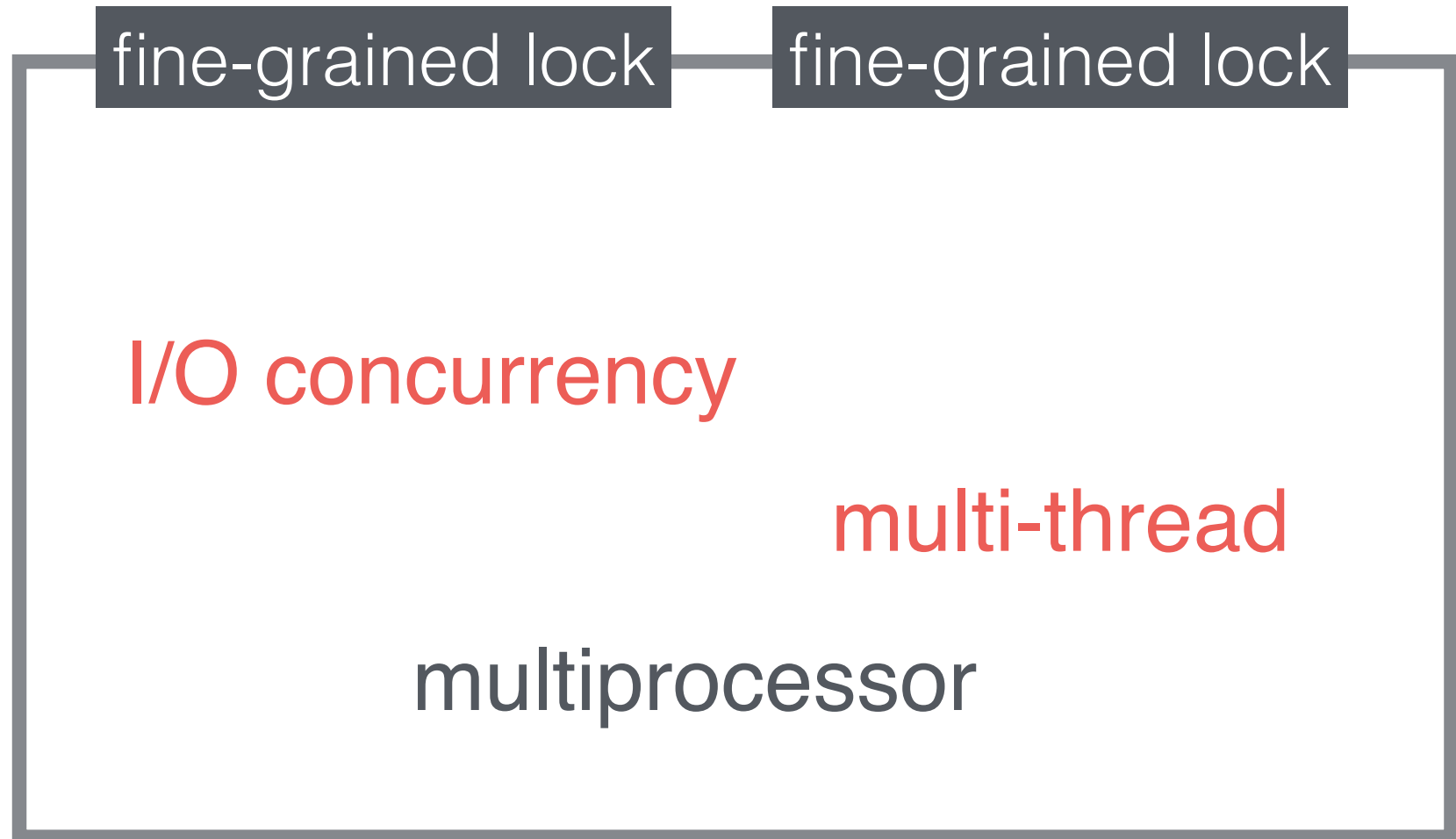


Challenges: Compositionality

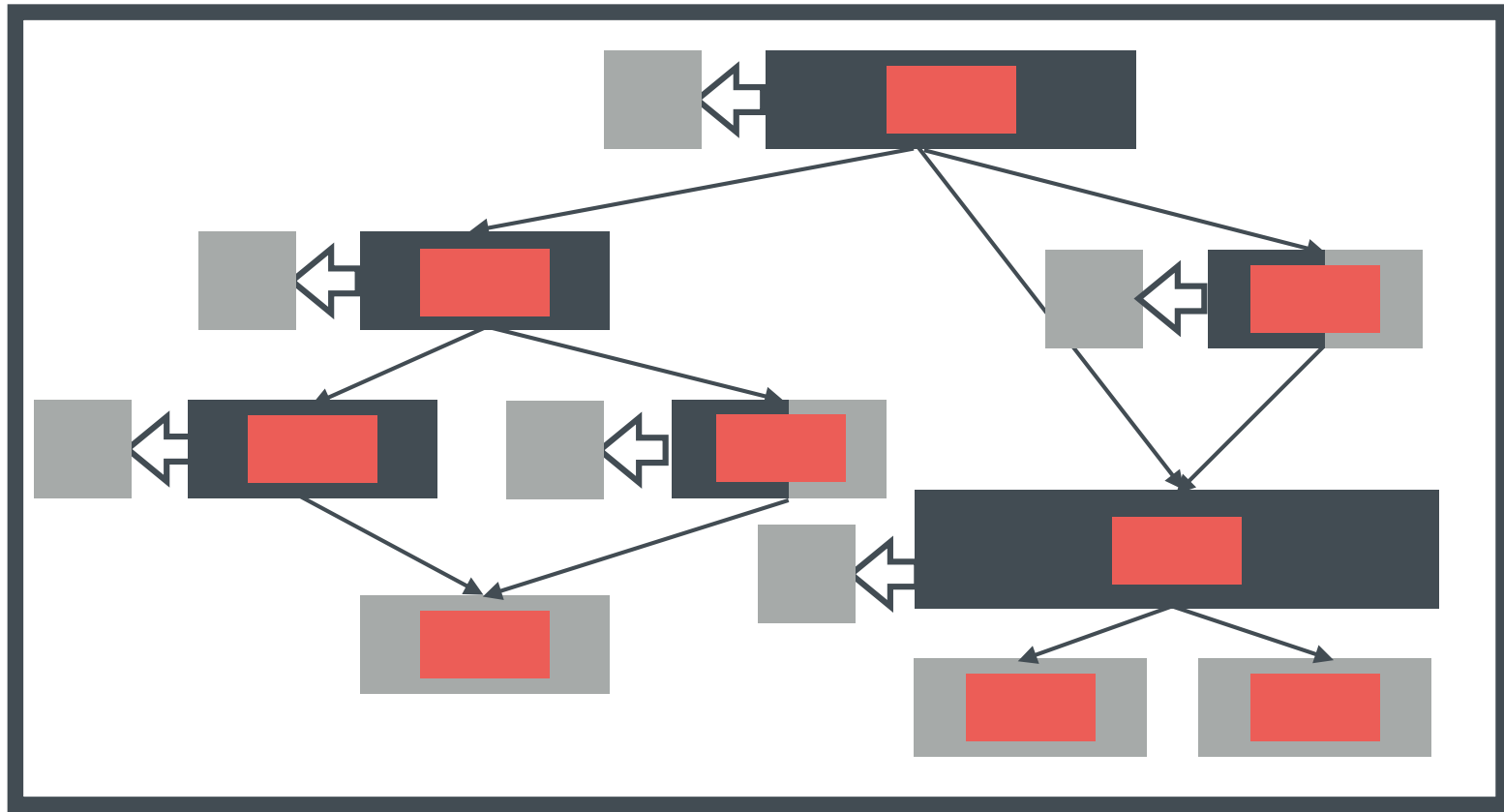
A Complex System



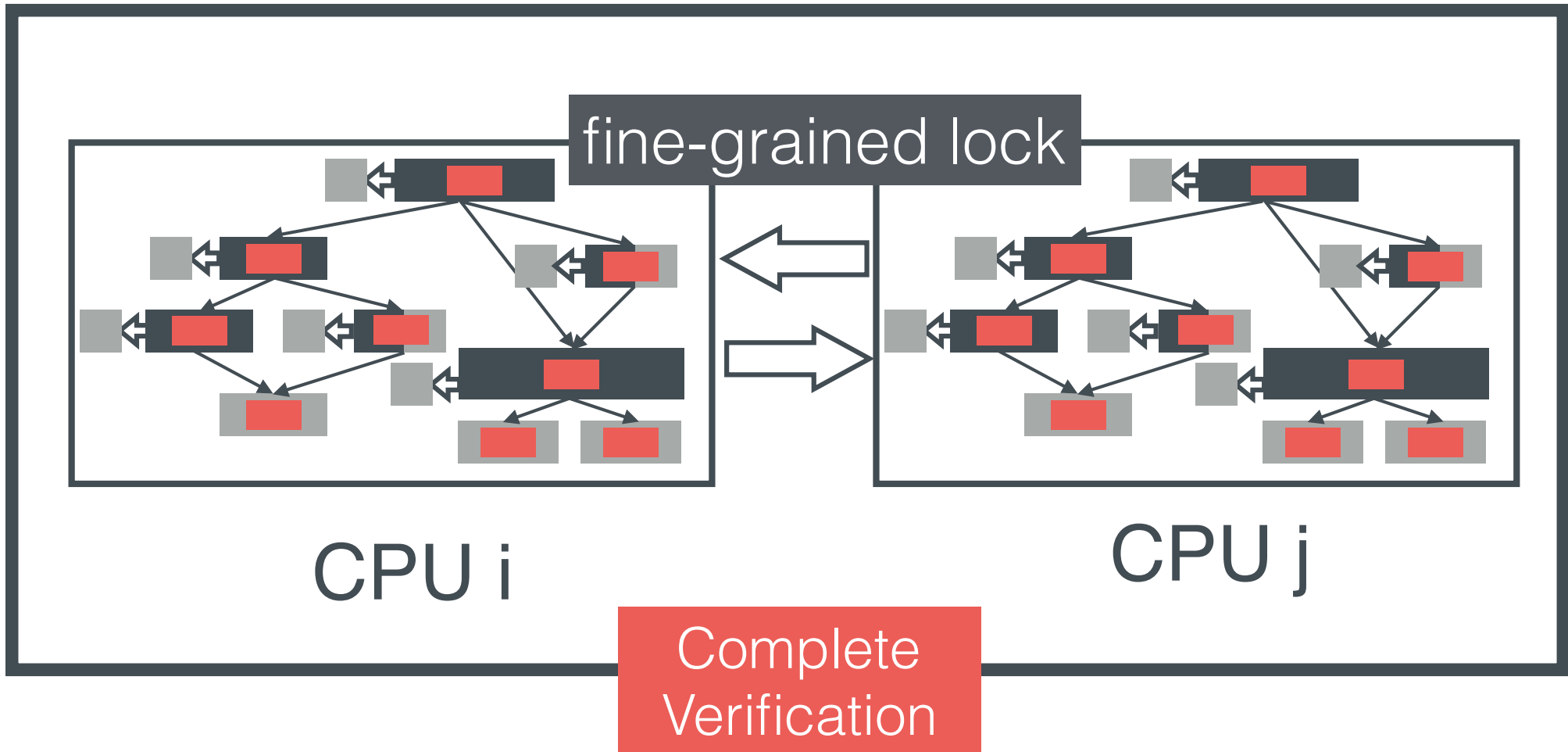
Challenges: Concurrency



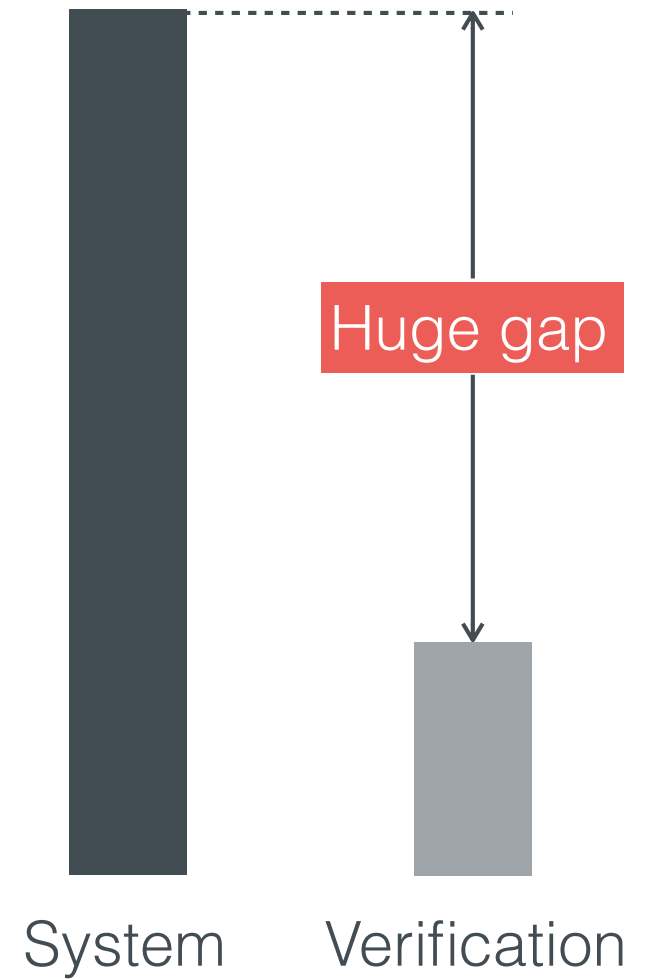
Challenges: Concurrency



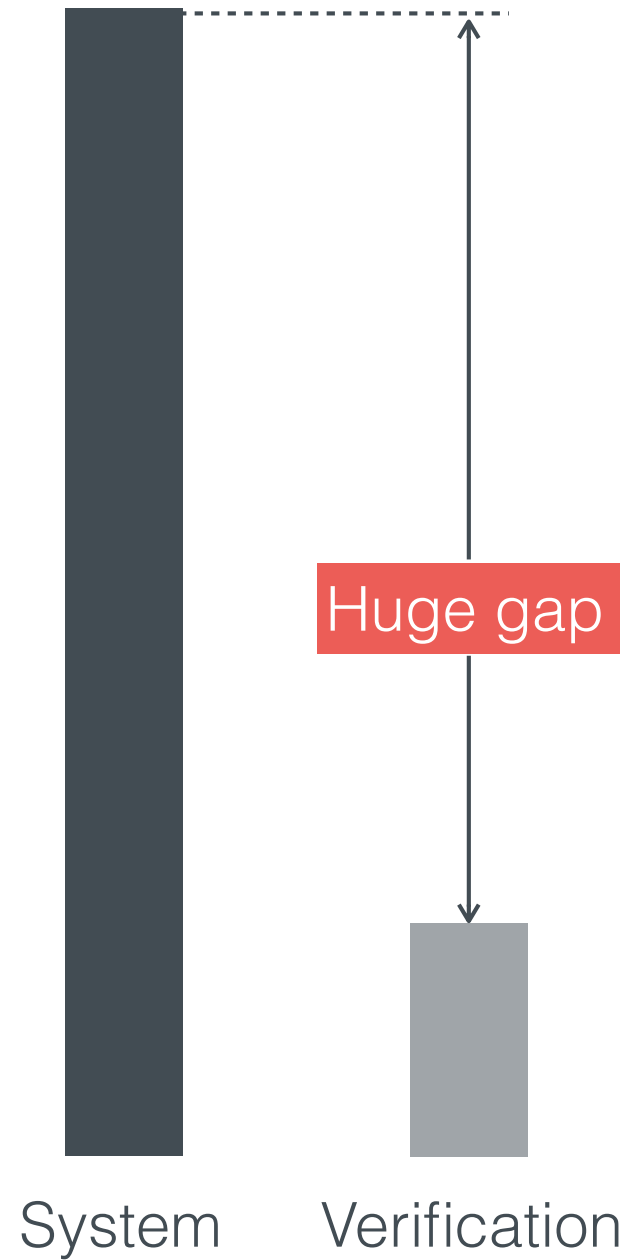
Challenges: Concurrency



Challenges: New Domain



Challenges: New Domain



Contribution

Certified Abstraction Layers

CertiKOS

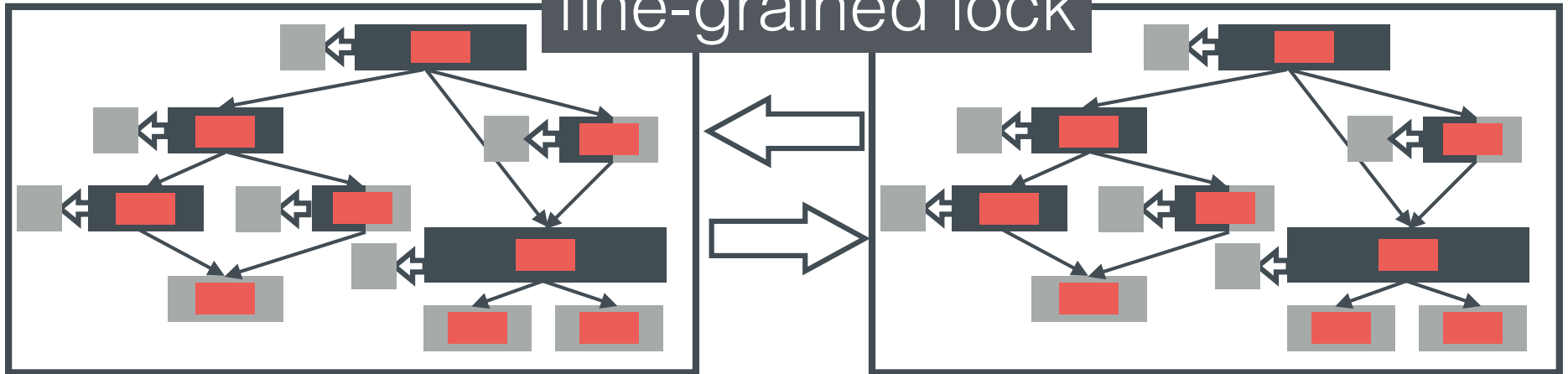
aim to solve all these challenges

Contribution

Certified Abstraction Layers

untangle

fine-grained lock



CPU i

CPU j

Contribution

Certified Abstraction Layers

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

Contribution

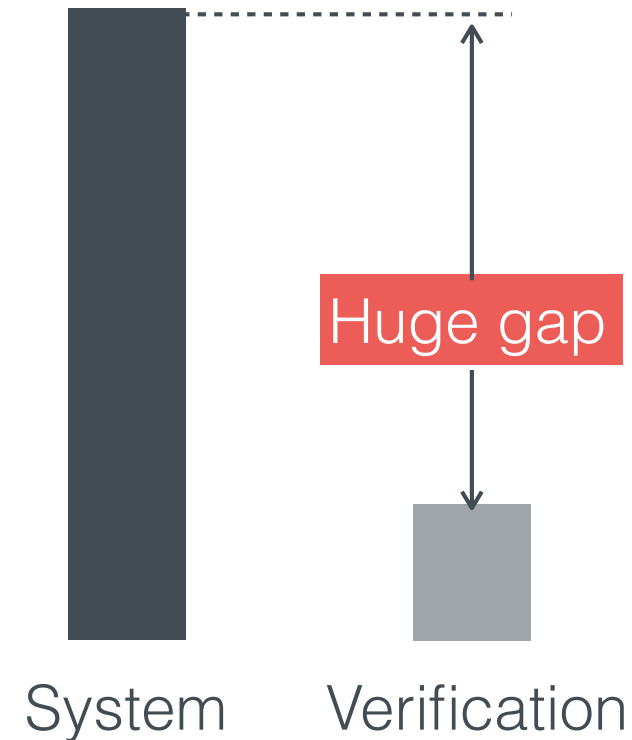
Certified Abstraction Layers

- verify existing systems
- build
certified system software

Contribution

Certified Abstraction Layers

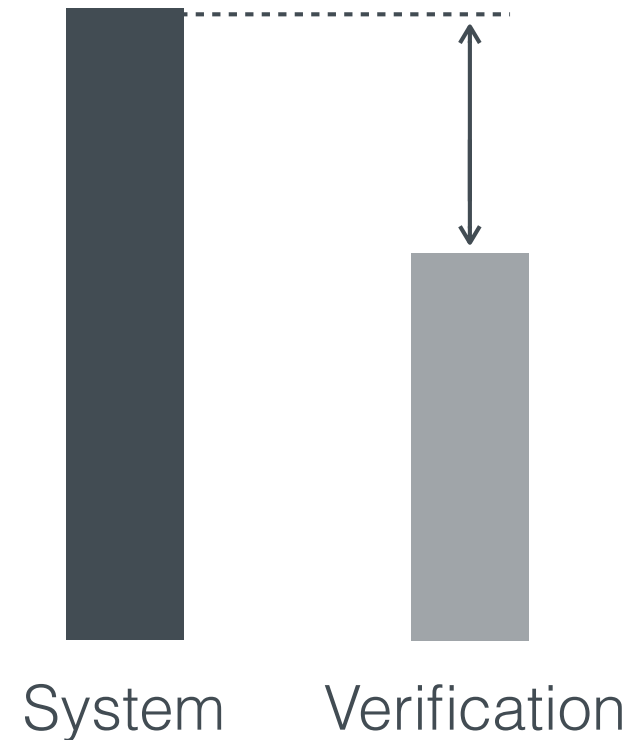
- verify existing systems
- build **certified** system software



Contribution

Certified Abstraction Layers

- verify existing systems
- build **certified** system software



Contribution

Certified Abstraction Layers

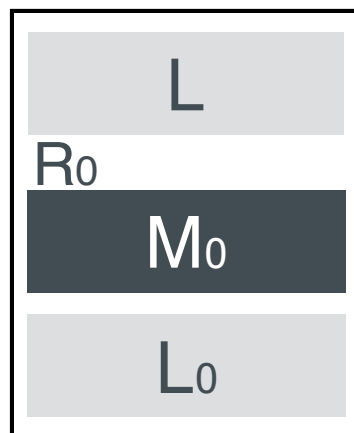
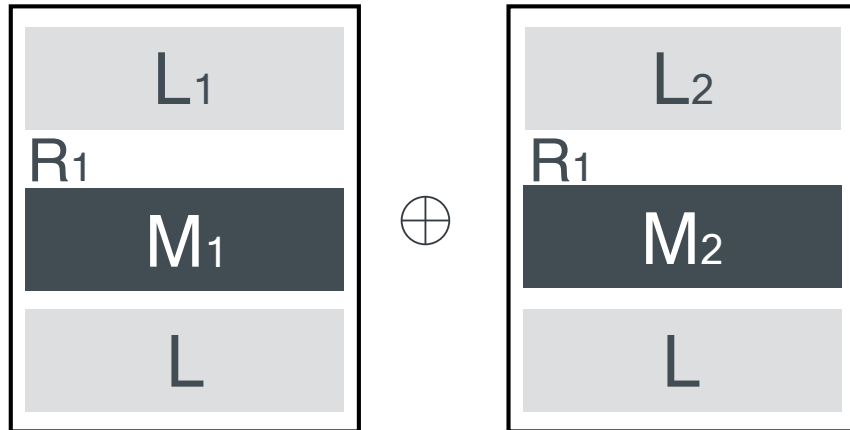
- verify existing systems
- build
certified system software



Certified System Software

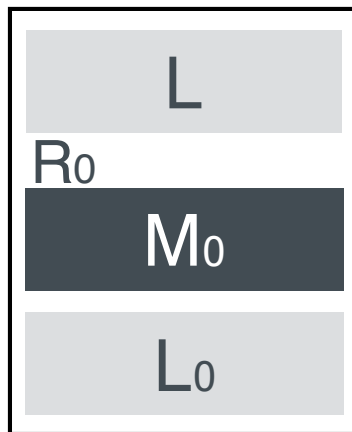
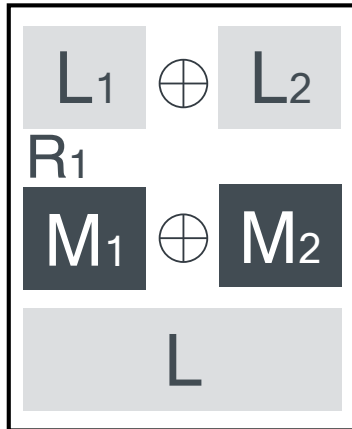
Contribution

Certified Abstraction Layers



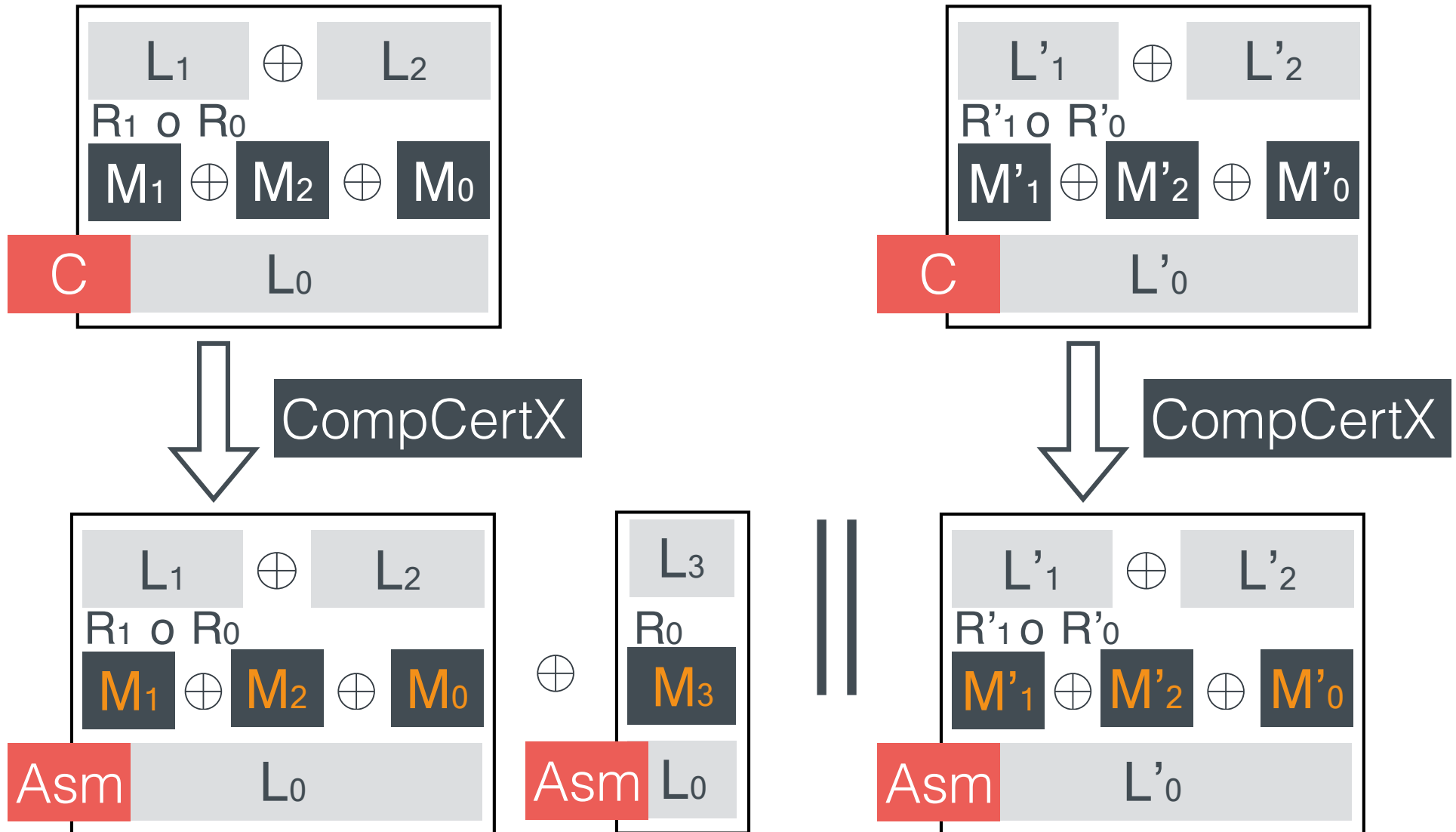
Contribution

Certified Abstraction Layers



Contribution

Certified Abstraction Layers



Contribution

Certified Abstraction Layers

- **mCertiKOS** [POPL'15]
certified sequential OS kernels
3k C&Asm, 1 py
- **Interrupt** [PLDI'16a] 0.5 py
- **Security** [PLDI'16b] 0.5 py
- **mC2** [OSDI'16] [CCAL 2017]
the first formally certified concurrent
OS kernel with fine-grained locks
6.5k C&Asm, 2 py

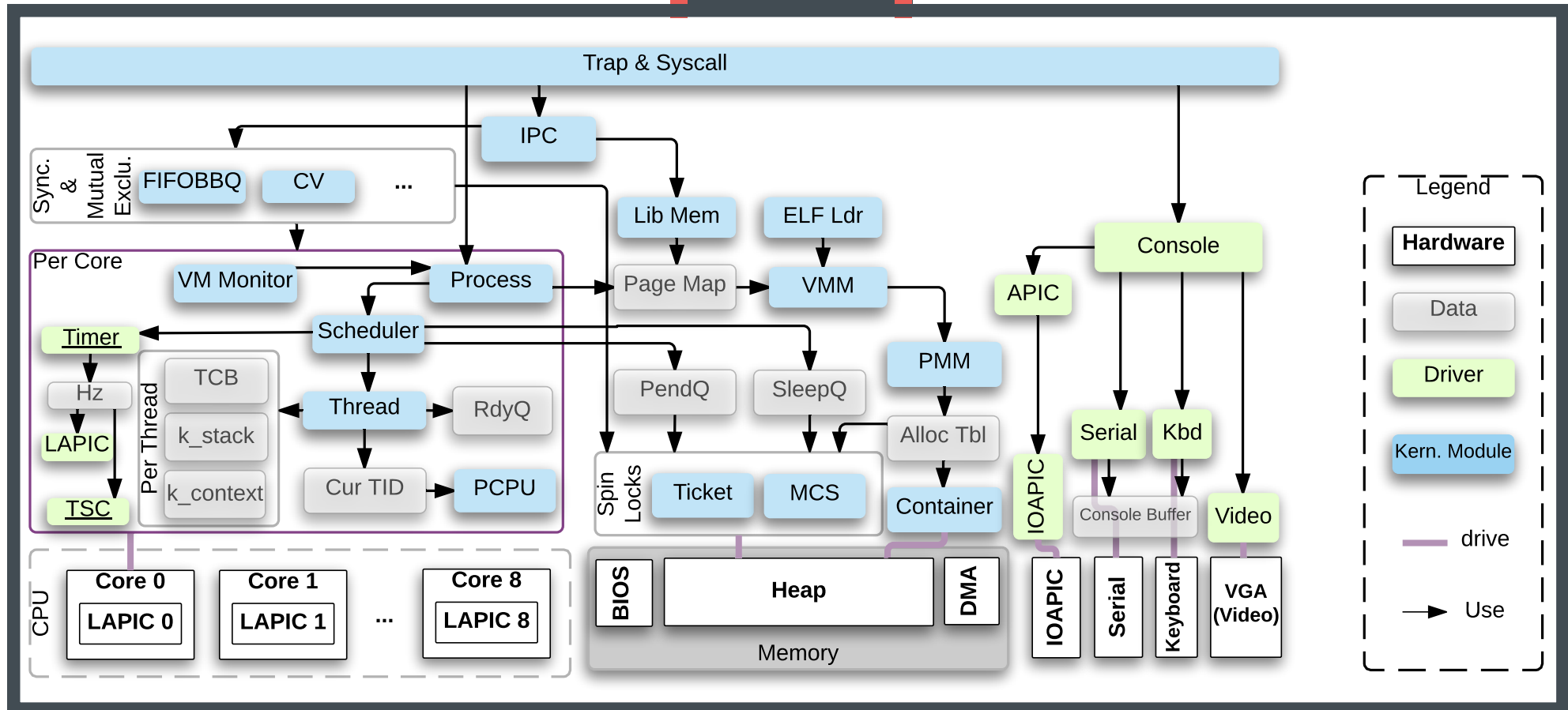
Contribution

Certified
System
Software

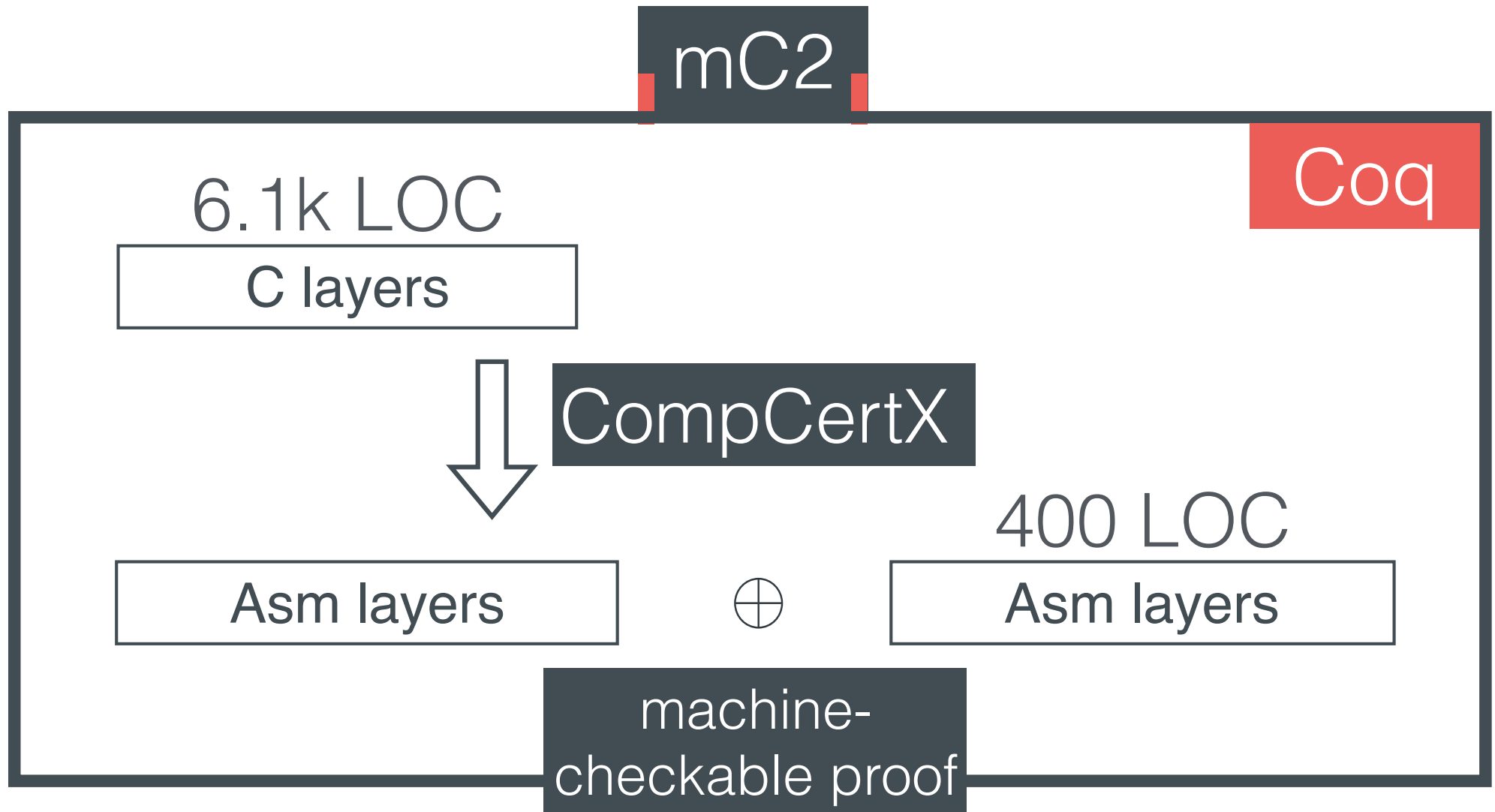
- functional correctness
- liveness
- no stack/integer/buffer overflow
- no race condition

Contribution

mC2



Contribution



Contribution

Proof Assistant

ACM Software
System Award

Coq

“

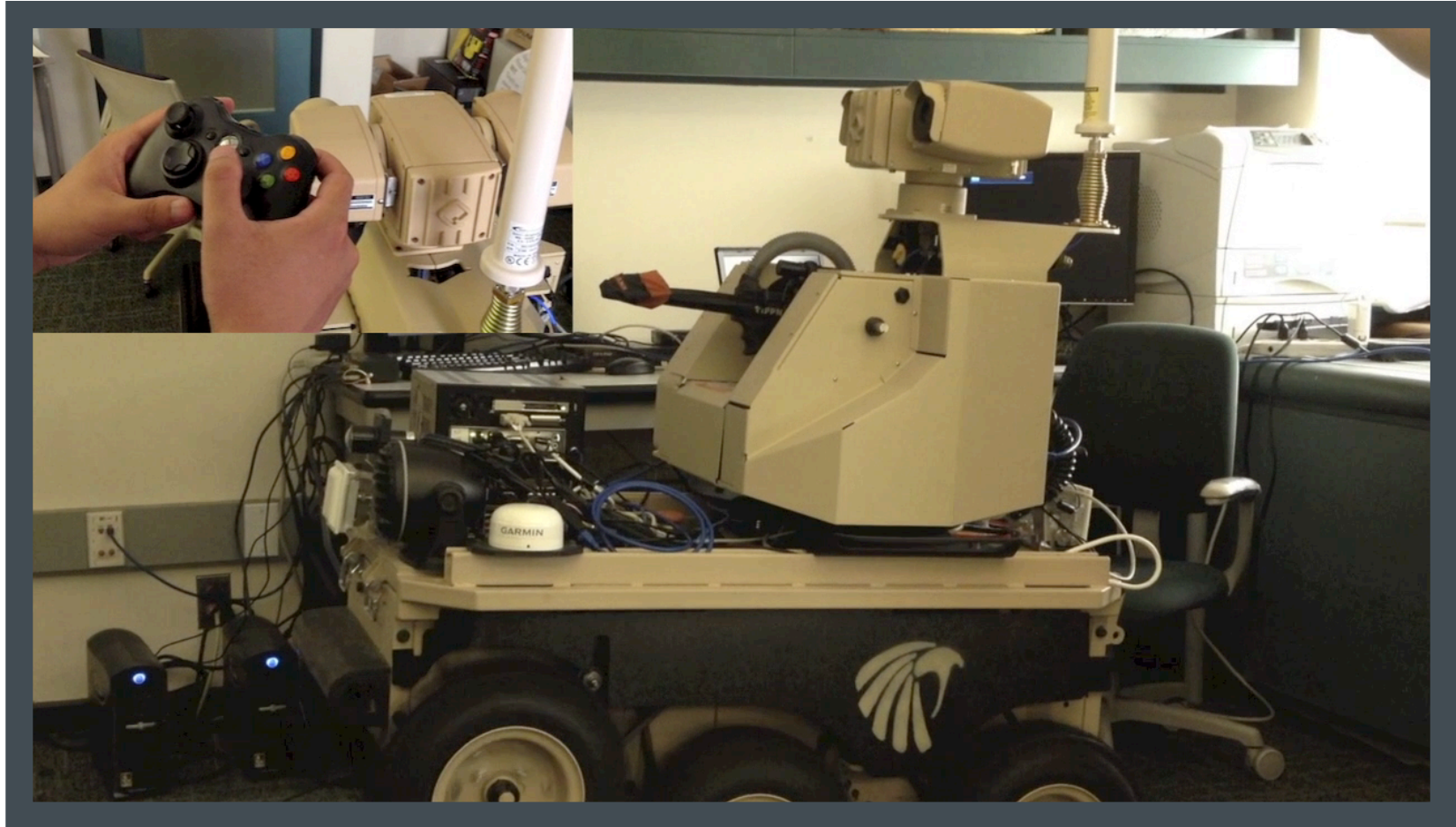
Some of the significant results that were accomplished using Coq are proofs for the **four color theorem**, the development of **CompCert** (a fully verified compiler for C), the development at Harvard of a verified version of Google's software fault isolation, and most recent, the fully specified and verified hypervisor OS kernel **CertiKOS**.

”

— ACM

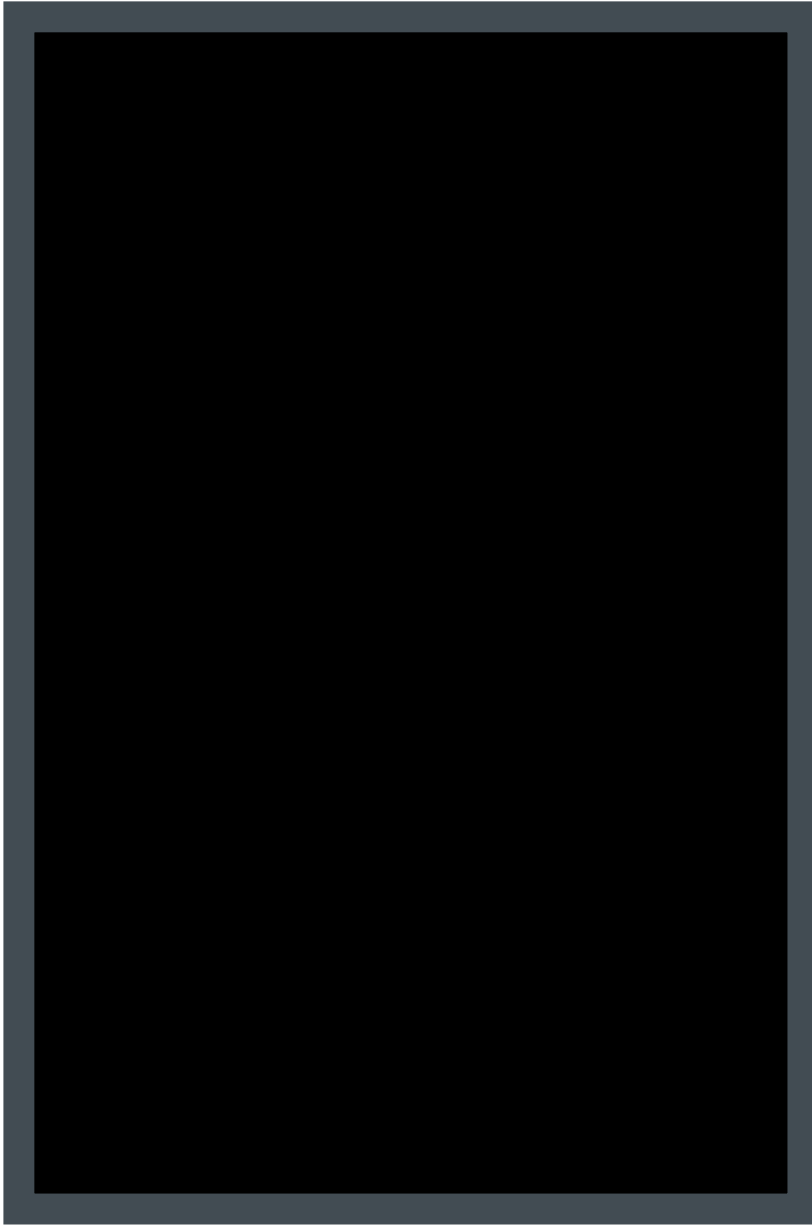
OC
yers

Deployment



CertiKOS on Landshark, DARPA HACMS

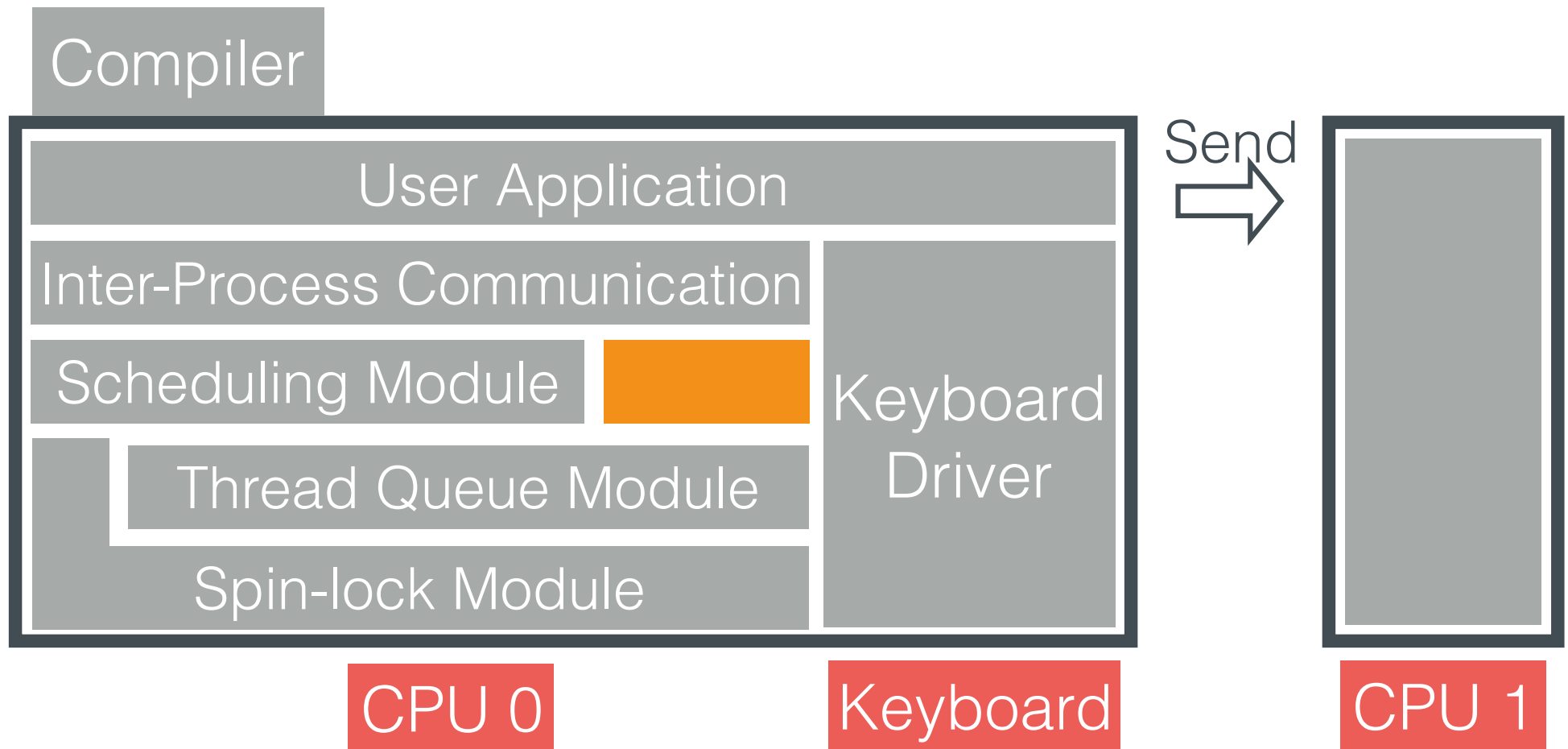
Deployment



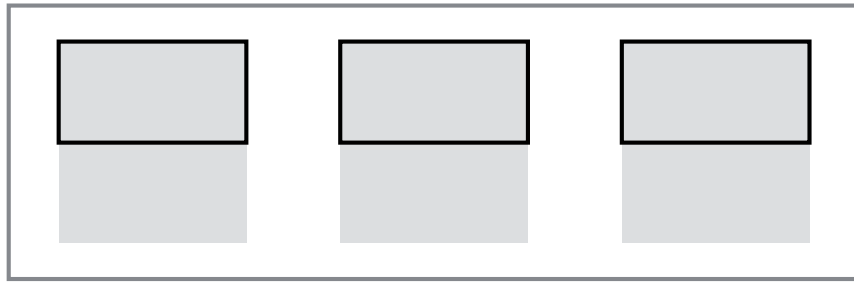
CertiKOS on
Quadcopter

Case Study

Build a Certified System



Certified Sequential Layer [POPL'15]

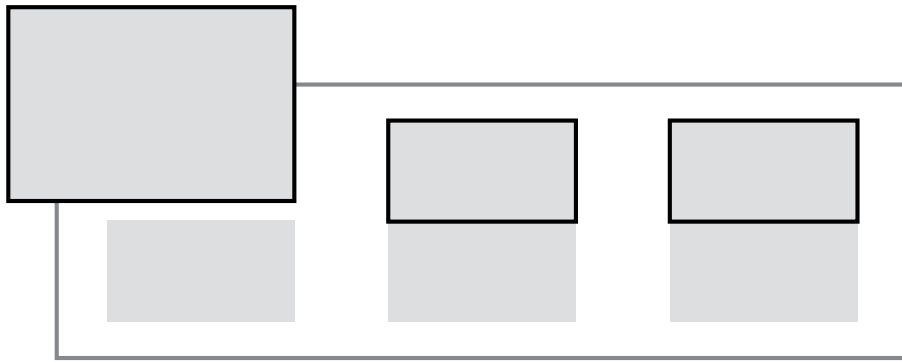


certified objects

specification of
modules to trust

Certified Sequential Layer [POPL'15]

abs-state

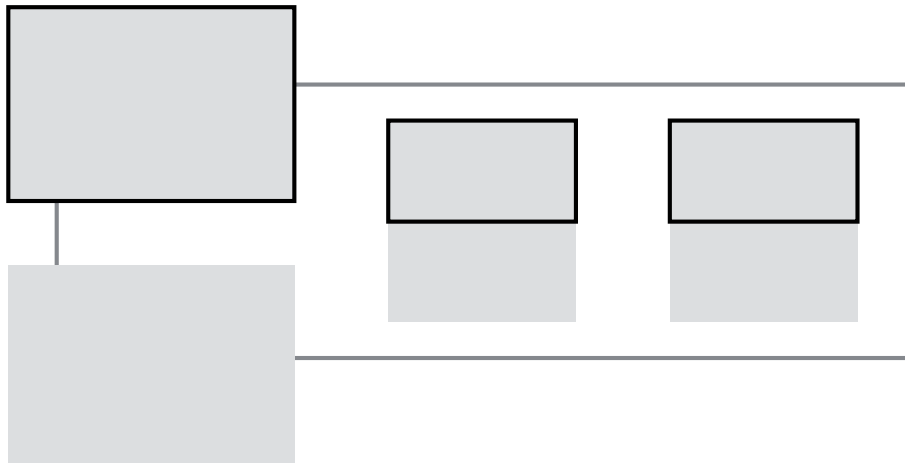


certified objects

specification of
modules to trust

Certified Sequential Layer [POPL'15]

abs-state

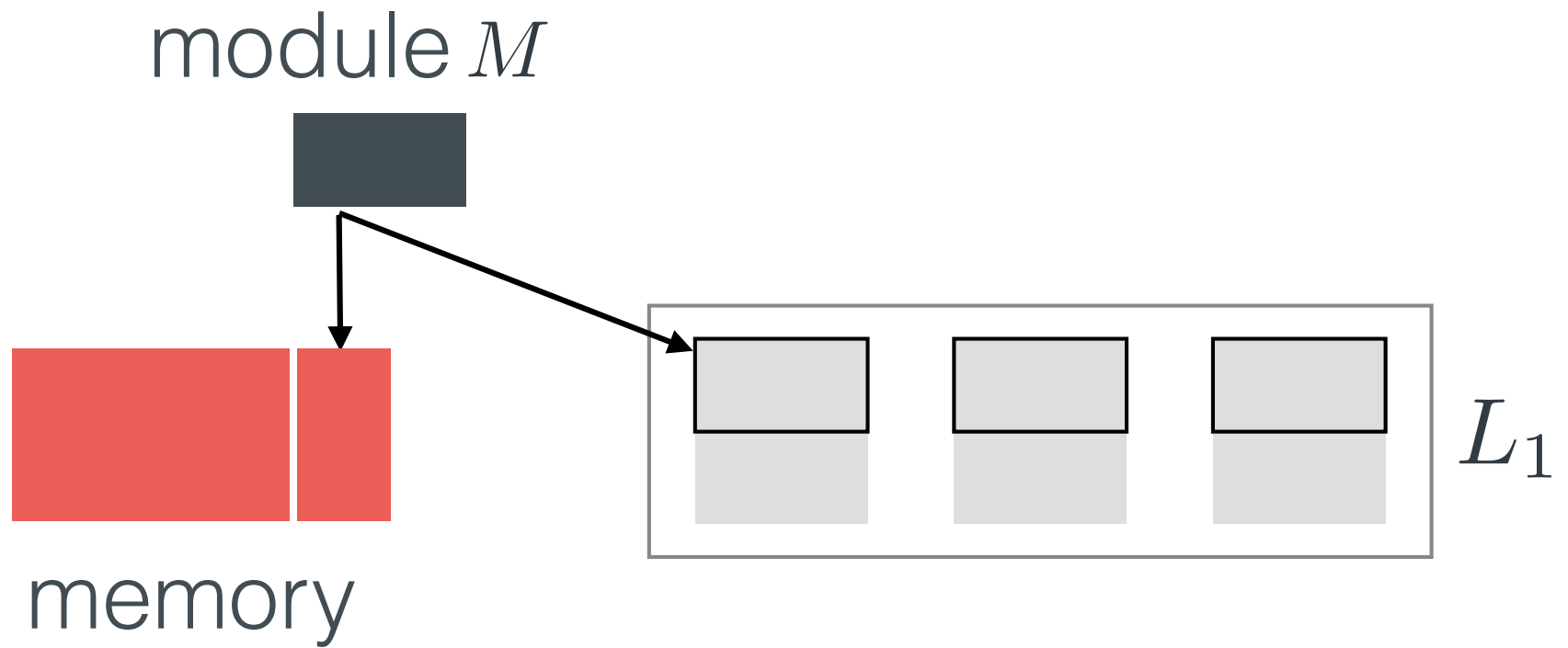


primitives

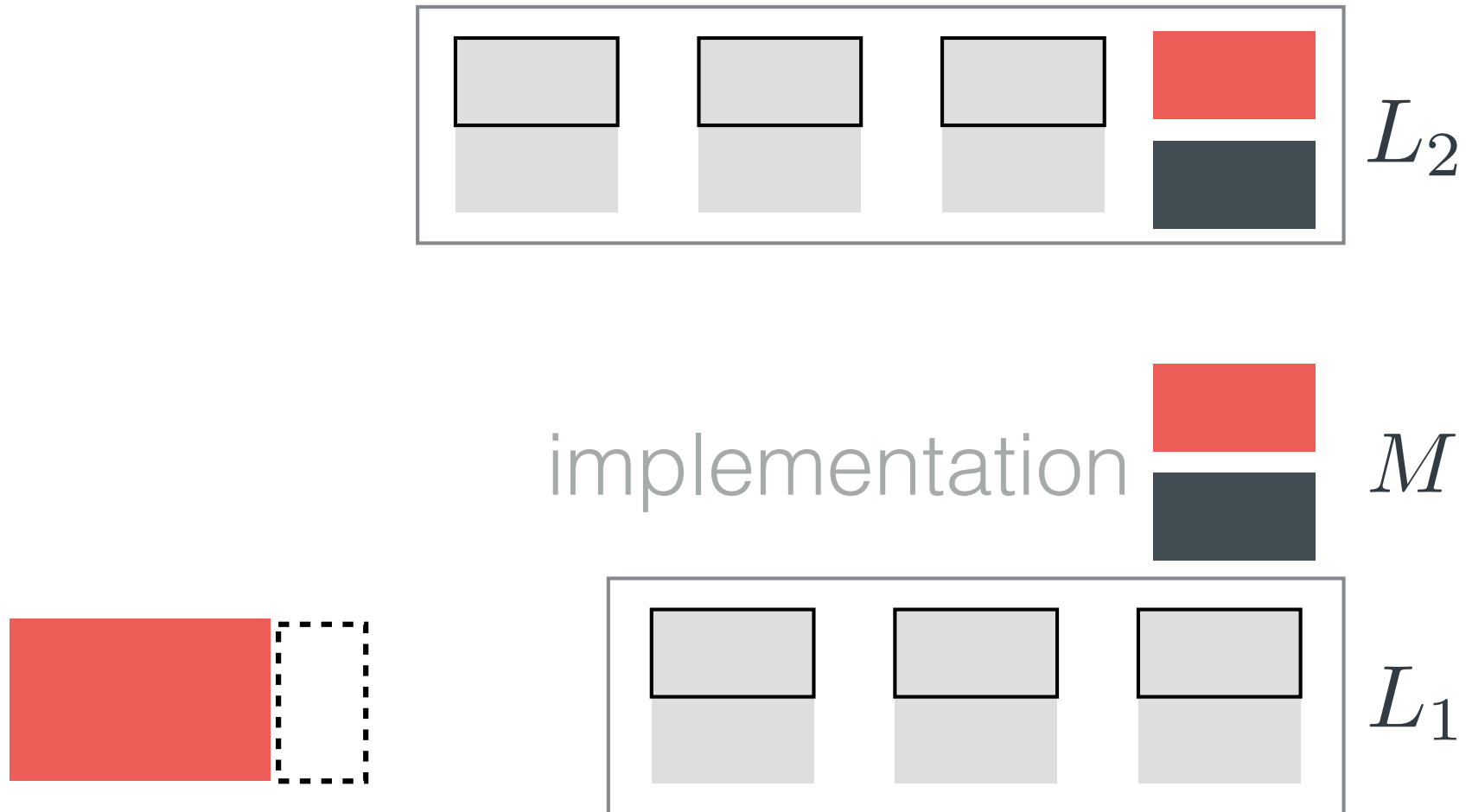
certified objects

specification of
modules to trust

Certified Sequential Layer

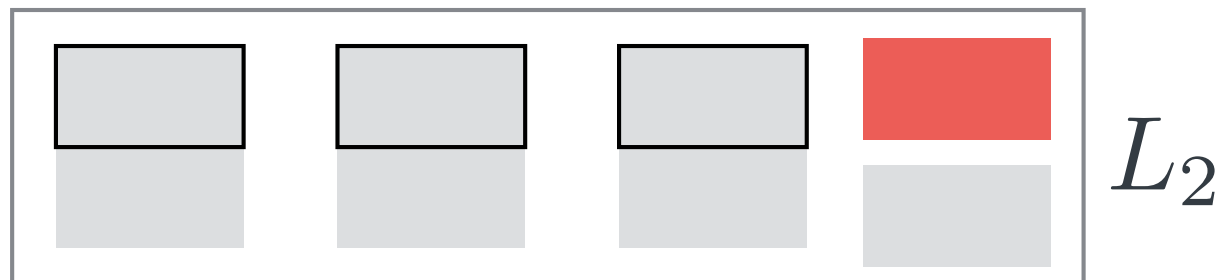


Certified Sequential Layer

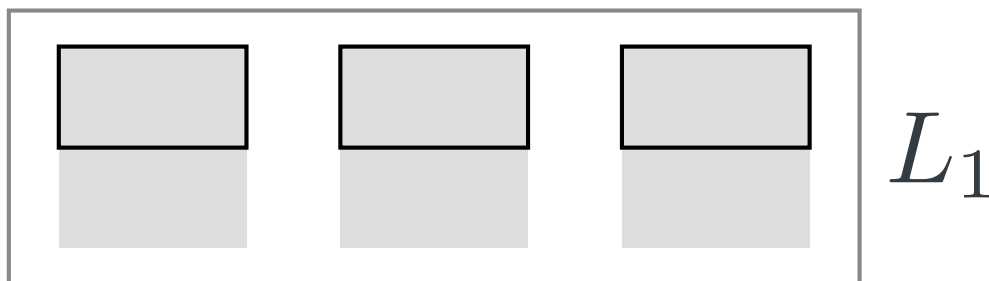


Certified Sequential Layer

specification

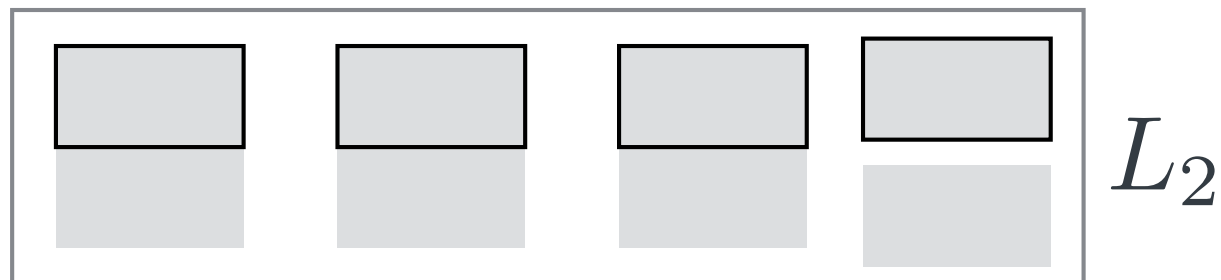


implementation

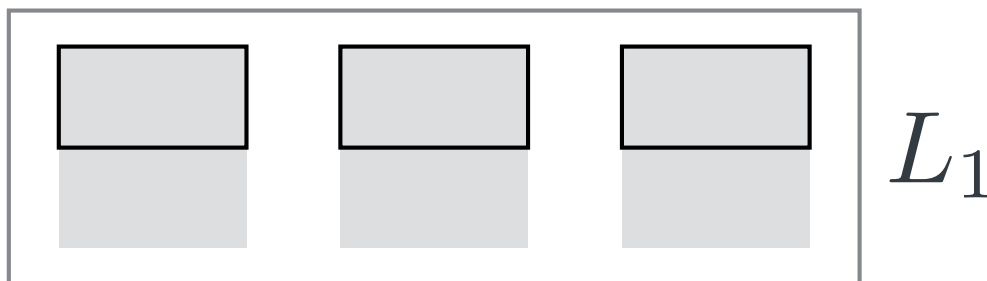


Certified Sequential Layer

specification



implementation



Example: Thread Queue

<pre>typedef struct tcb { state s; tcb *prev, *next; } tcb; tcb tcbp[1024];</pre>	<pre>typedef struct tdq { tcb *head, *tail; } tdq; tdq* td_queue;</pre>
---	---

C



M

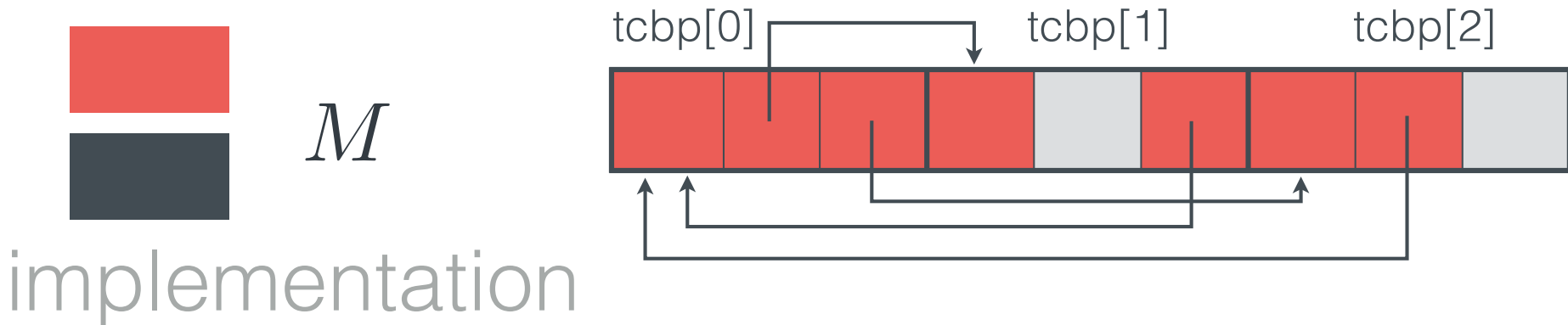


implementation

Example: Thread Queue

<pre>typedef struct tcb { state s; tcb *prev, *next; } tcb; tcb tcbp[1024];</pre>	<pre>typedef struct tdq { tcb *head, *tail; } tdq; tdq* td_queue;</pre>
---	---

C



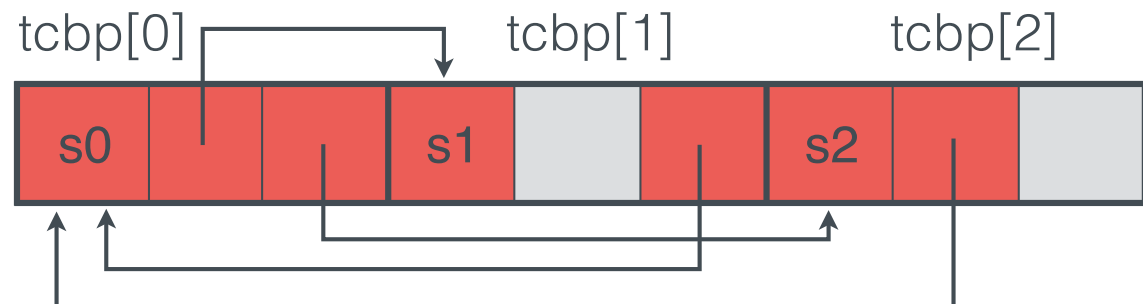
Example: Thread Queue

<pre>typedef struct tcb { state s; tcb *prev, *next; } tcb; tcb tcbp[1024];</pre>	<pre>typedef struct tdq { tcb *head, *tail; } tdq; tdq* td_queue;</pre>
---	---

C

M

implementation



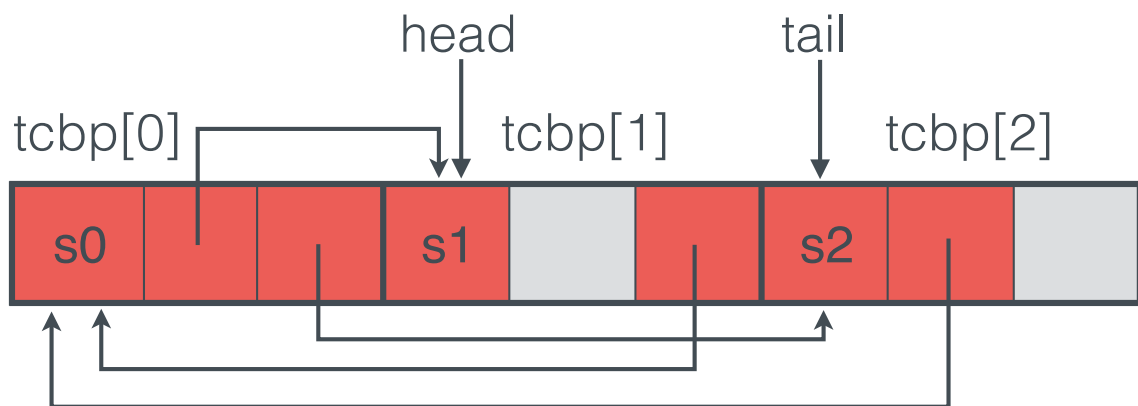
Example: Thread Queue

<pre>typedef struct tcb { state s; tcb *prev, *next; } tcb; tcb tcbp[1024];</pre>	<pre>typedef struct tdq { tcb *head, *tail; } tdq; tdq* td_queue;</pre>
---	---

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;  
    return i;  
}
```

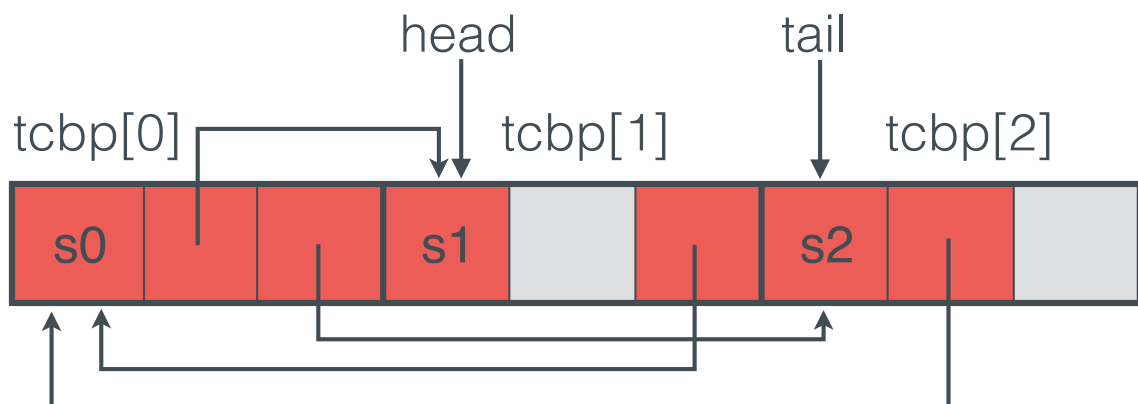
```
if (!next) {  
    q -> head = null;  
    q -> tail = null;  
} else {  
    next -> prev = null;  
    q -> head = next;  
}  
return i;
```

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;  
}
```

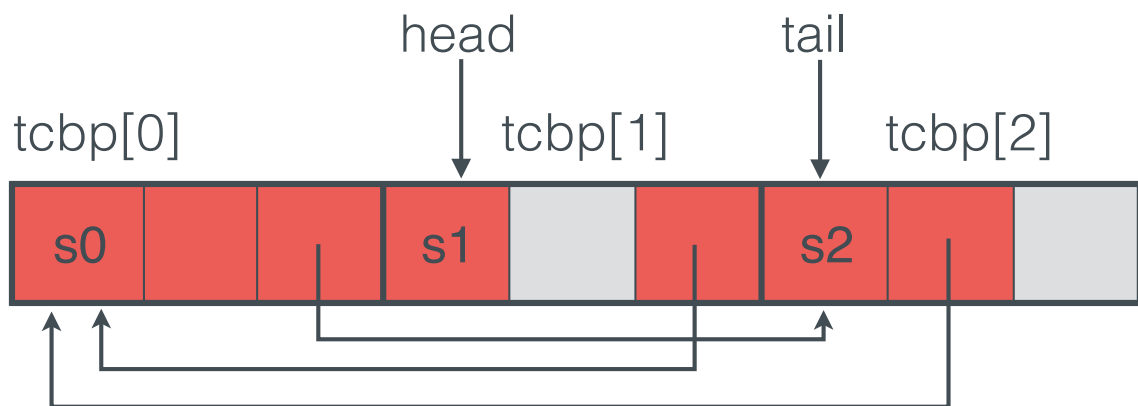
```
    if (!next) {  
        q -> head = null;  
        q -> tail = null;  
    } else {  
        next -> prev = null;  
        q -> head = next;  
    }  
    return i;  
}
```

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;  
    }
```

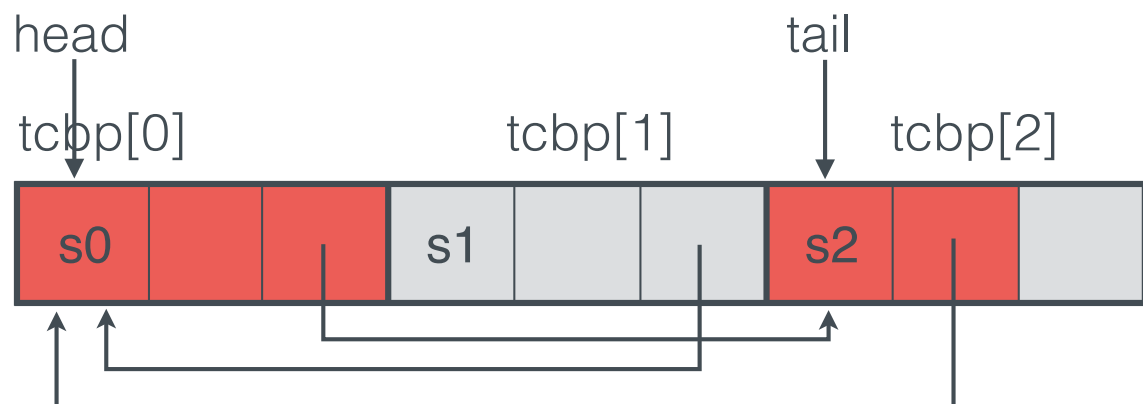
```
    if (!next) {  
        q -> head = null;  
        q -> tail = null;  
    } else {  
        next -> prev = null;  
        q -> head = next;  
    }  
    return i;  
}
```

C



M

implementation



Example: Thread Queue

specification



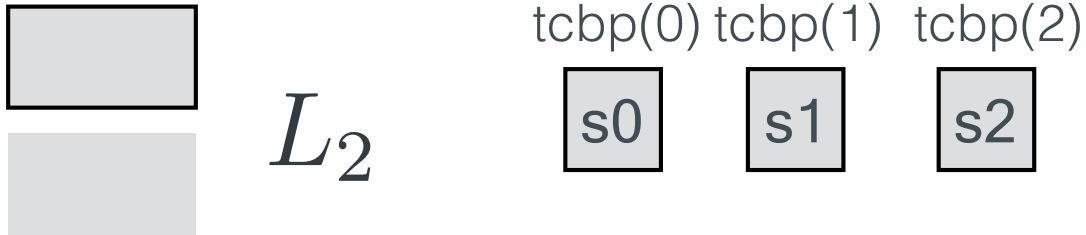
Definition **tcbp** := ZMap.t state.

Definition **td_queue** := List Z.

Coq

Example: Thread Queue

specification



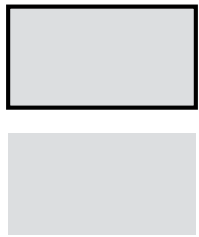
Definition **tcbp** := ZMap.t state.

Definition **td_queue** := List Z.

Coq

Example: Thread Queue

specification



L_2

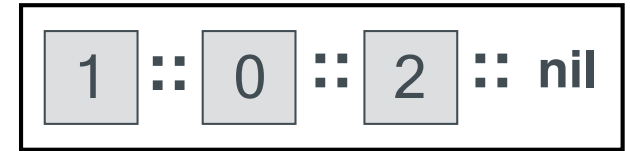
tcbp(0) tcbp(1) tcbp(2)

s0

s1

s2

td_queue



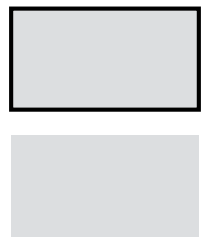
Definition **tcbp** := ZMap.t state.

Definition **td_queue** := List Z.

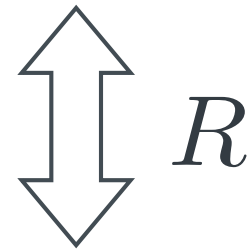
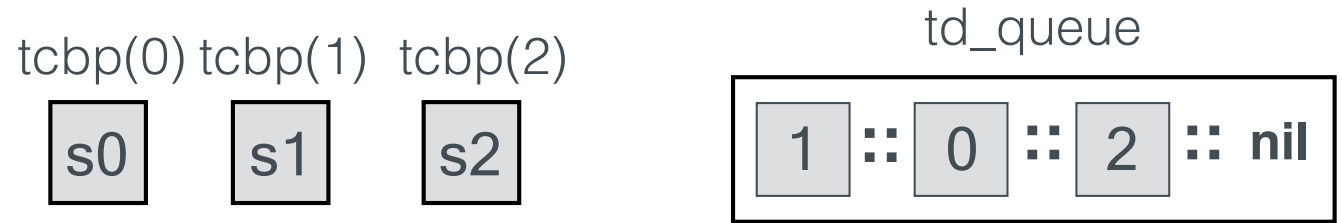
Coq

Example: Thread Queue

specification

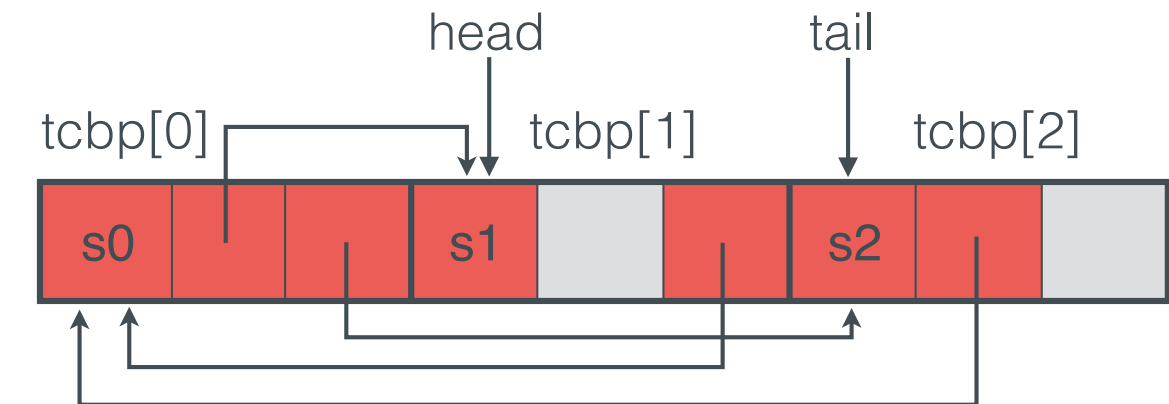


L_2



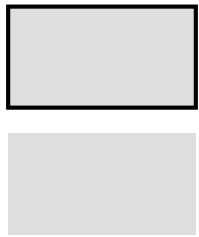
M

implementation



Example: Thread Queue

specification



L_2

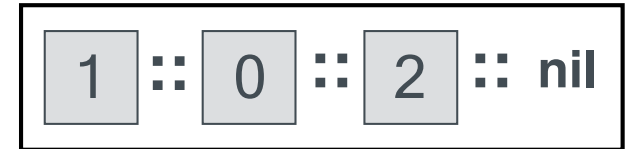
tcbp(0) tcbp(1) tcbp(2)

s0

s1

s2

td_queue

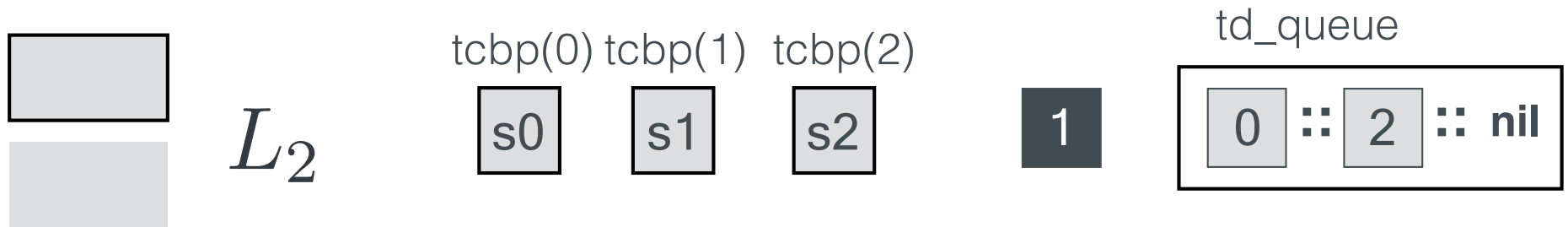


```
Function dequeue (q) :=  
  match q with  
  | head :: q' => (q', Some head)  
  | nil => (nil, None)  
end.
```

Coq

Example: Thread Queue

specification



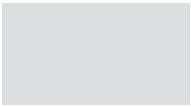
```
Function dequeue (q) :=  
  match q with  
  | head :: q' => (q', Some head)  
  | nil => (nil, None)  
end.
```

Coq

executable

Simulation Proof

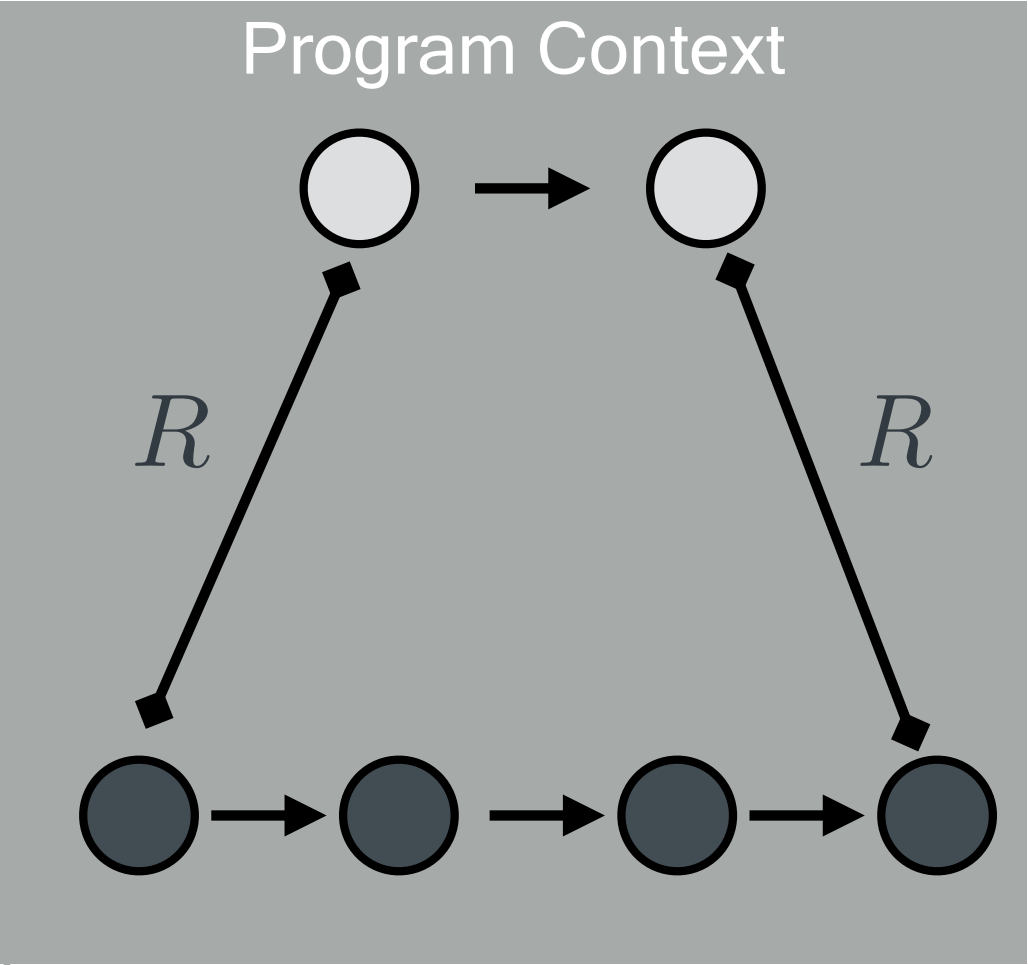
specification



L_2



M



Deep Specification



R



L_2

M

L_1

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

kernel

Trap

PM

TM

MM

seq machine

mCertiKOS

kernel

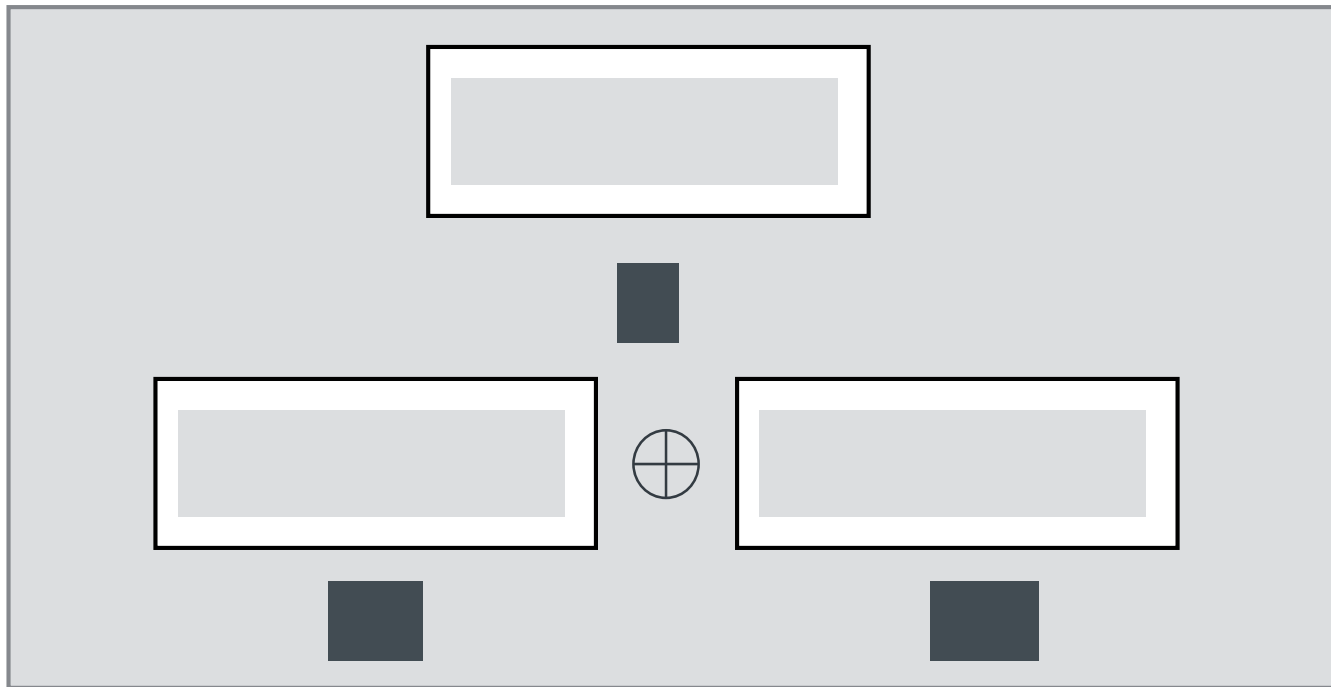
Trap

PM

TM

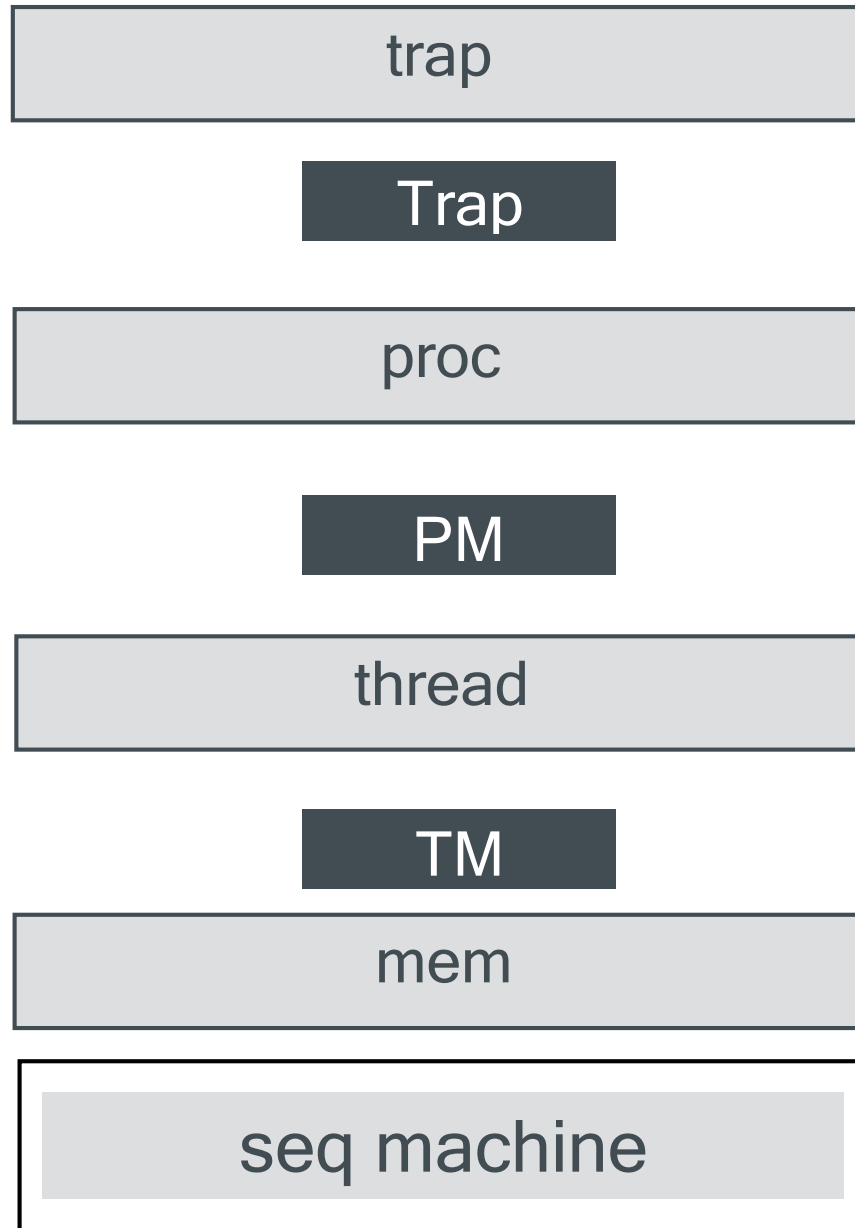
MM

memory management

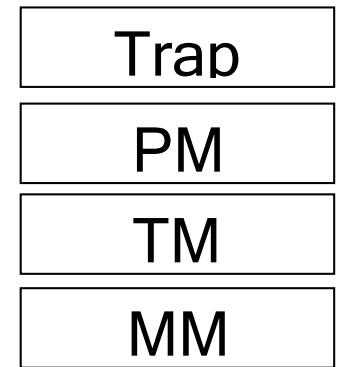


seq machine

mCertiKOS



kernel



mCertiKOS

certified sequential kernel

VM

Trap

PM

TM

MM

trap

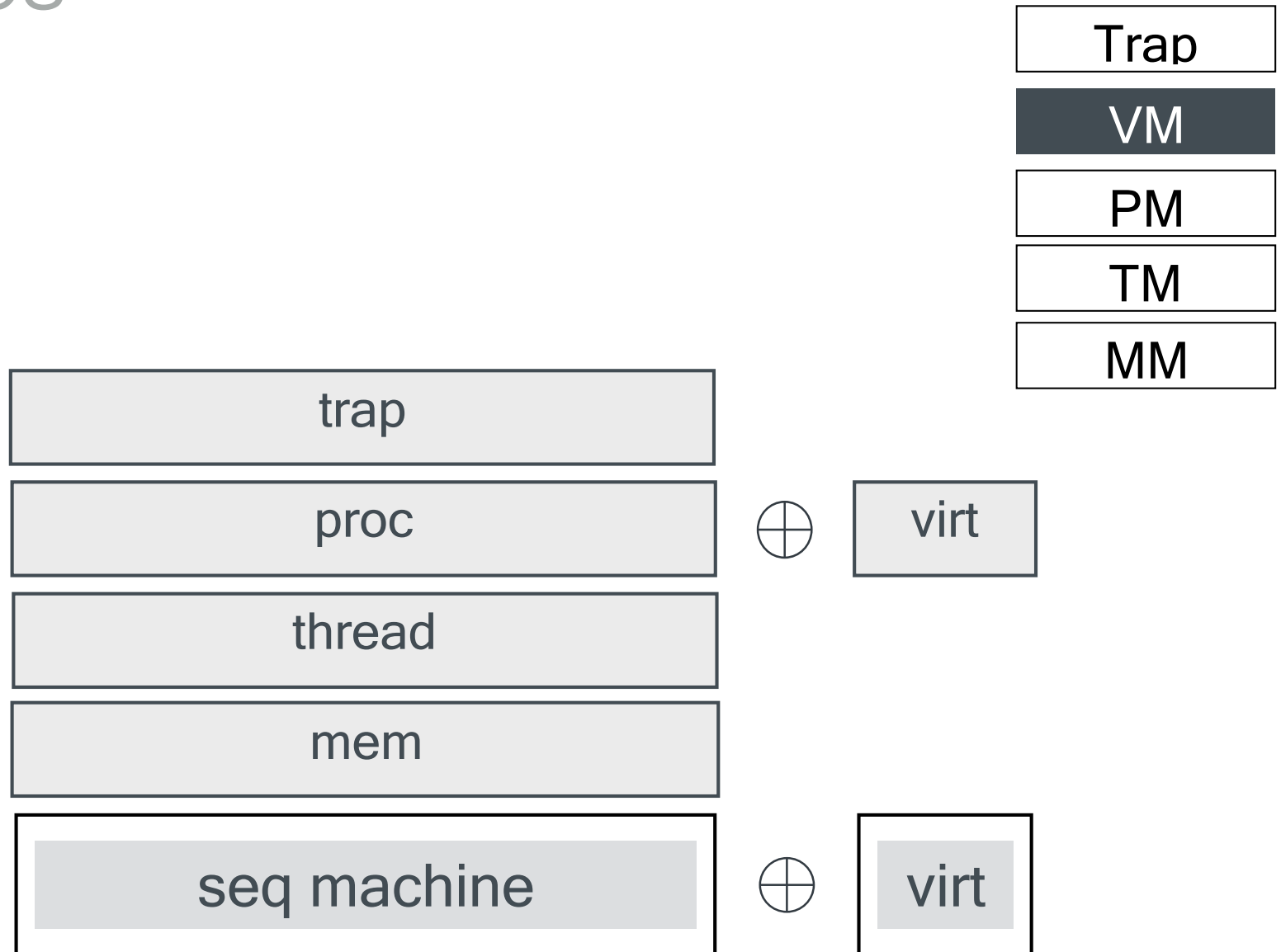
proc

thread

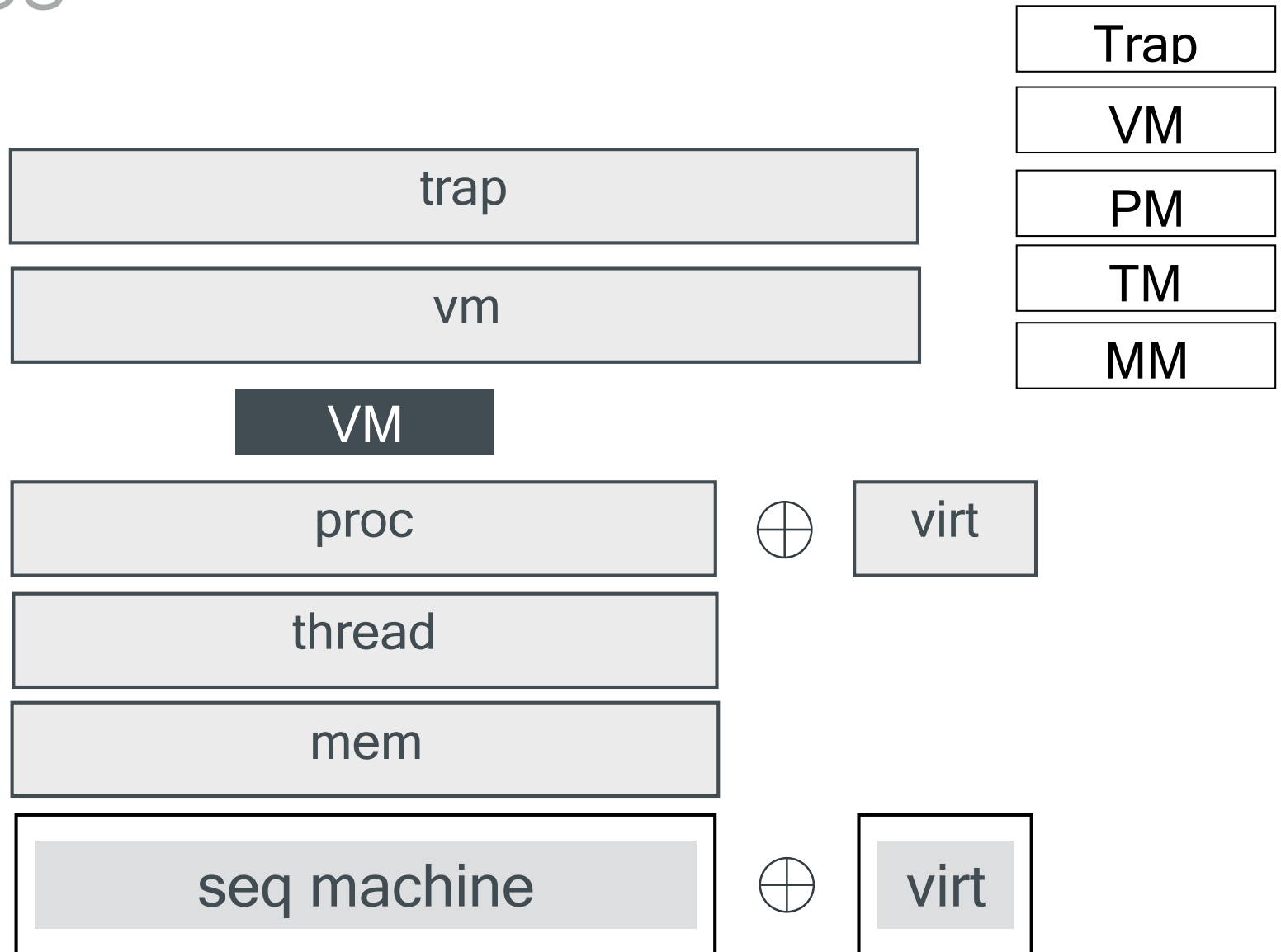
mem

seq machine

mCertiKOS

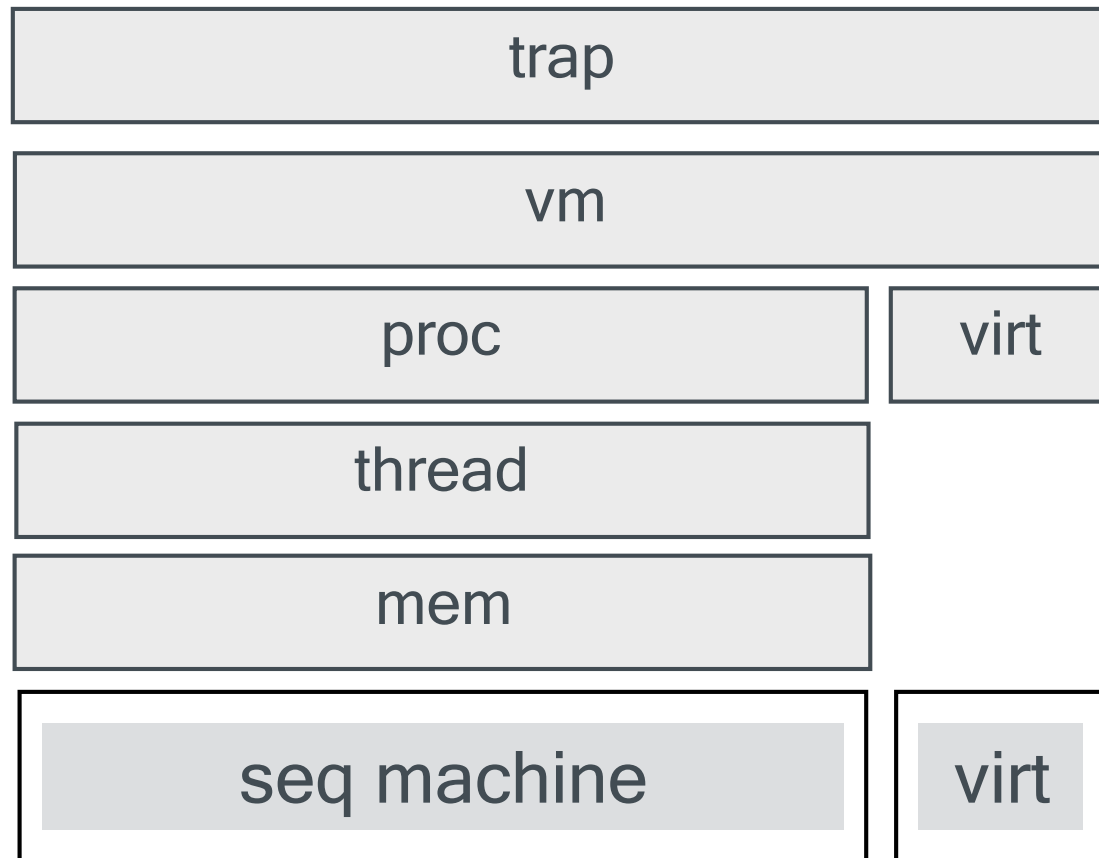


mCertiKOS



mCertiKOS

certified hypervisor



Trap

VM

PM

TM

MM

- mCertiKOS 3k LOC
[POPL'15] 1 person year
- Can boot Linux as a guest

TSysCall Layer

(pe, ikern, ihost, ipt, AT, PT, ptp, pbit, kctxp, Htcbp, Htqp, cid, chanp, uctxp, npt, hctx, vmst)

thread_wakeup/kill/sleep/yield	pt_read	get/set_uctx	palloc/free	cid_get
sys_chan_send/recv/wait/check	sys_yield	sys_get_exit_reason	sys_get_eip	
sys_check_shadow/pending_event	sys_proc_create	sys_set_seg	sys_inject	
sys_get_exit_io_width/port/rep/str/write/eip	sys_set_intcept_int	sys_npt_instr		
vmcbinit	pagefault_handler	sys_reg_get/set	sys_sync	sys_run
				vm_exit



TSysCall Layer

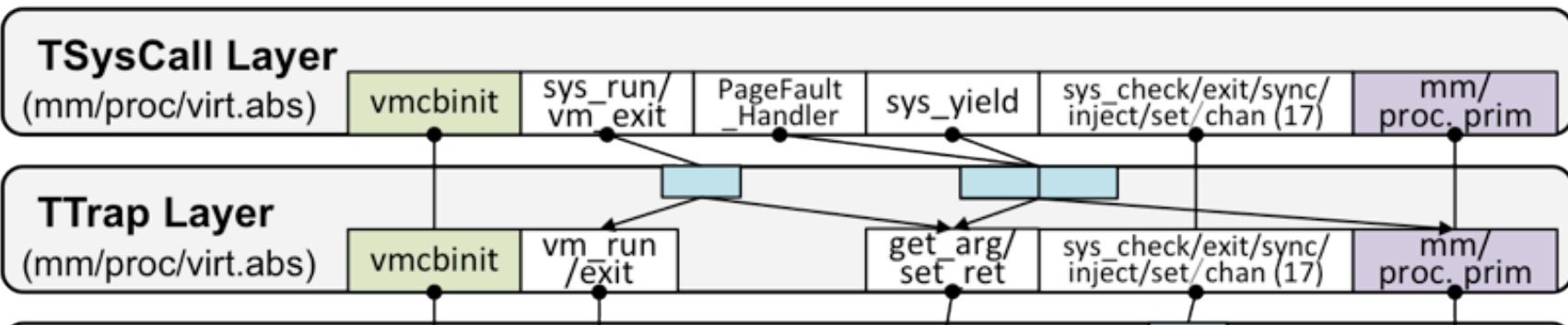
(mm/proc/virt.abs)

vmcbinit	sys_run/vm_exit	PageFault_Handler	sys_yield	sys_check/exit/sync/inject/set/chan (17)	mm/proc.prim
----------	-----------------	-------------------	-----------	--	--------------

TTrap Layer

(mm/proc/virt.abs)

vmcbinit	vm_run/exit	get_arg/set_ret	sys_check/exit/sync/inject/set/chan (17)	mm/proc.prim
----------	-------------	-----------------	--	--------------



Concurrent Framework [OSDI'16]

certified sequential kernel

trap

virt

proc

thread

mem

seq machine

multicore machine

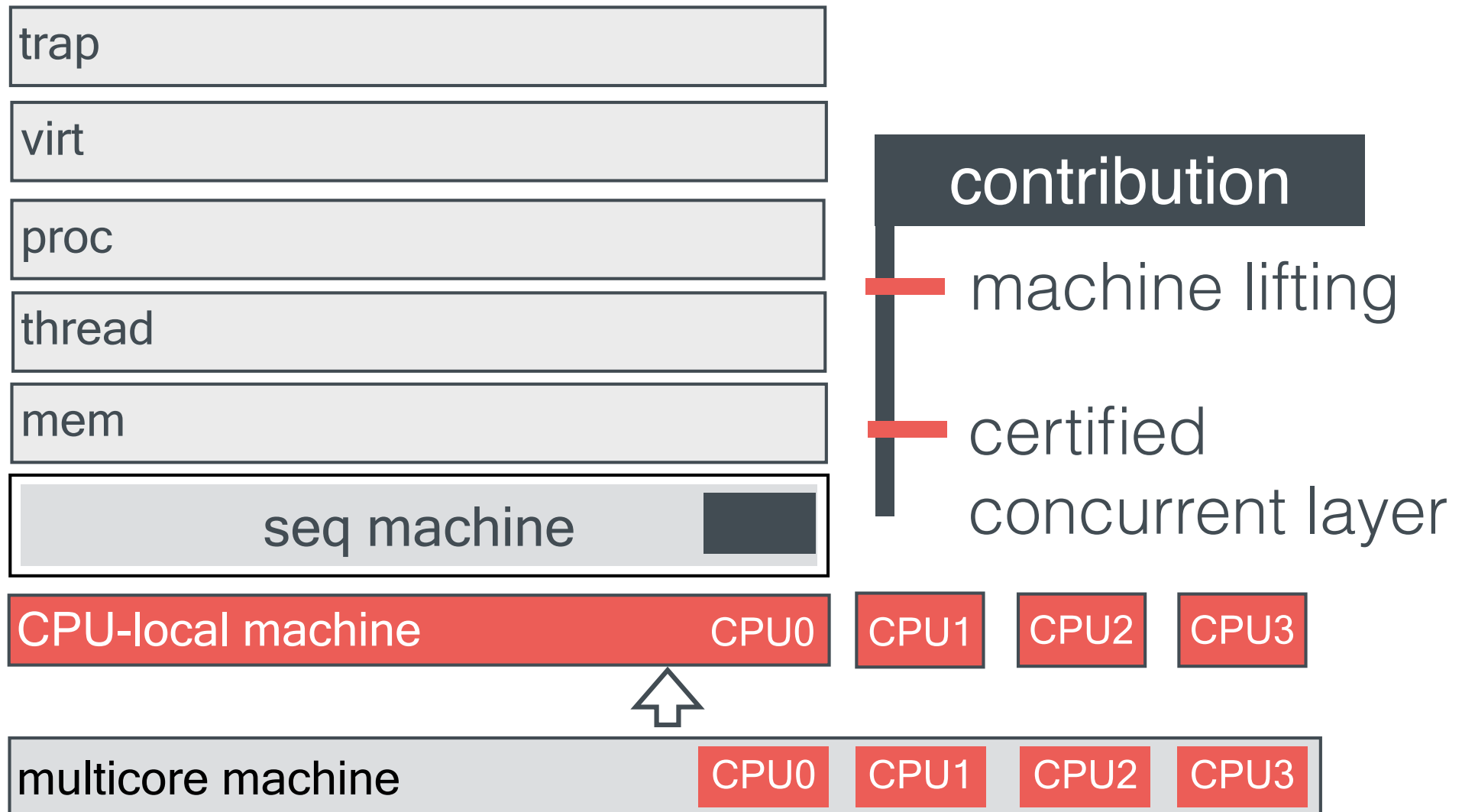
CPU0

CPU1

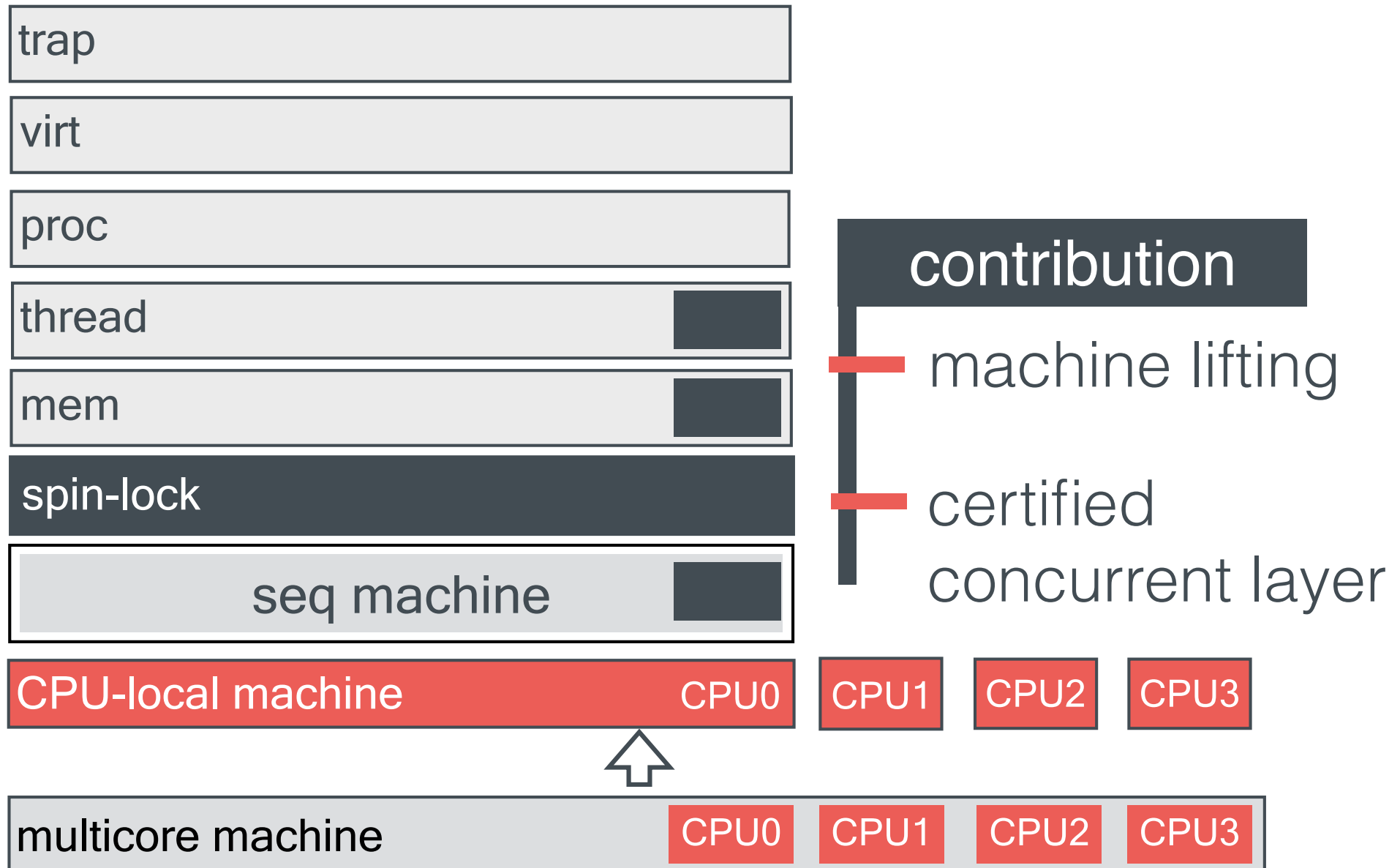
CPU2

CPU3

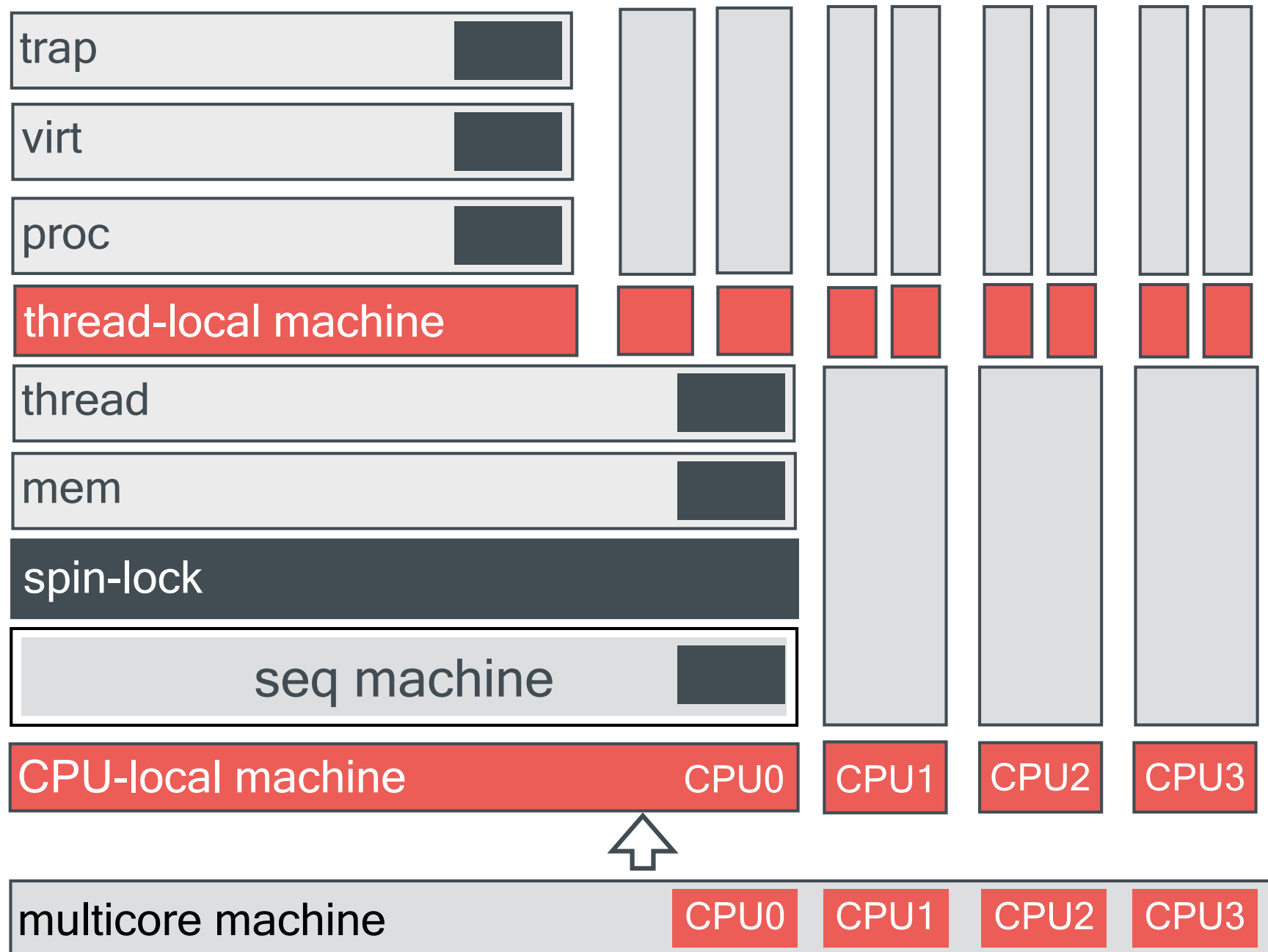
Concurrent Framework [OSDI'16]



Concurrent Framework [OSDI'16]

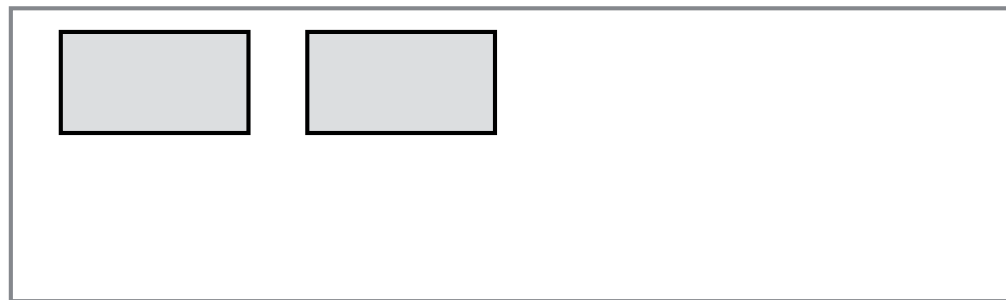


Concurrent Framework [OSDI'16]



Certified Concurrent Layers

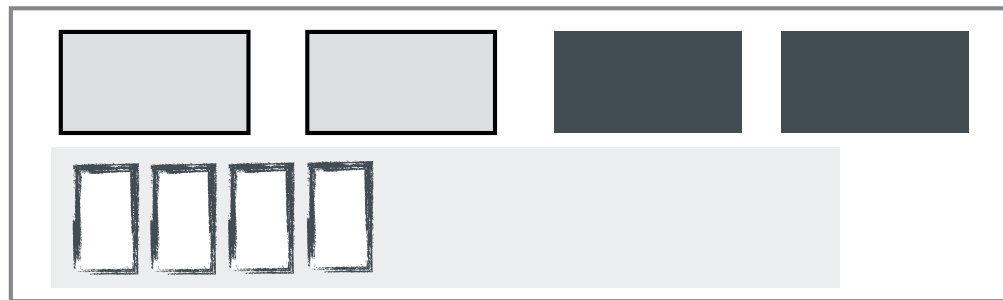
local certified objects



L_1

Certified Concurrent Layers

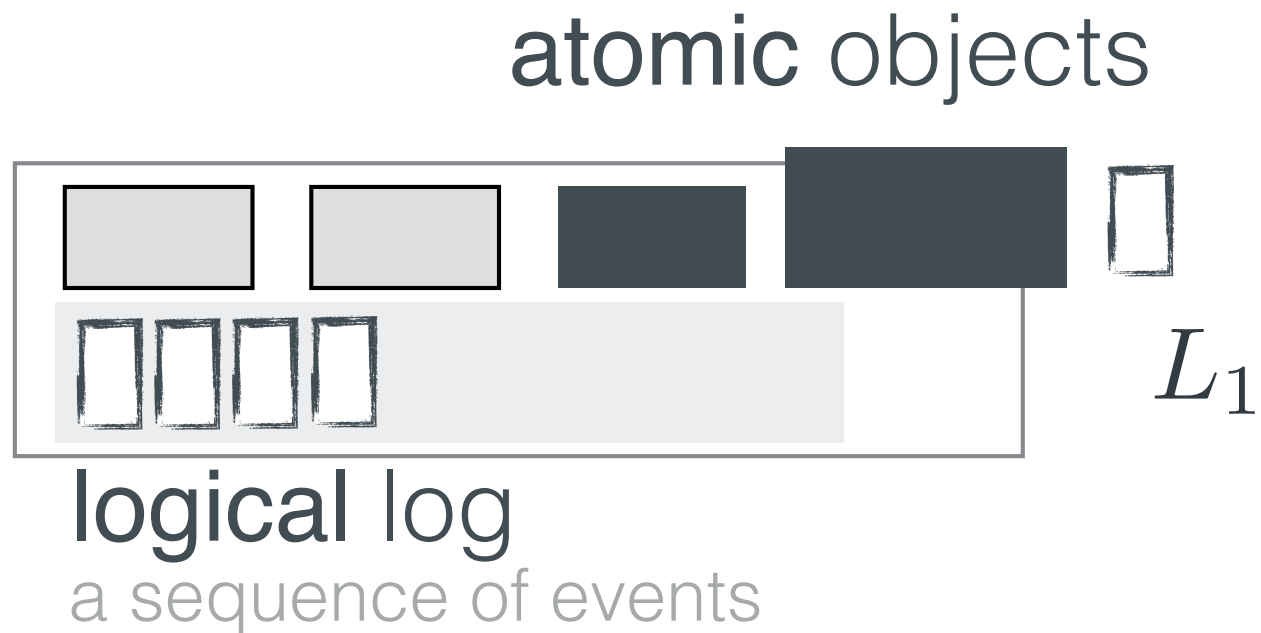
atomic objects



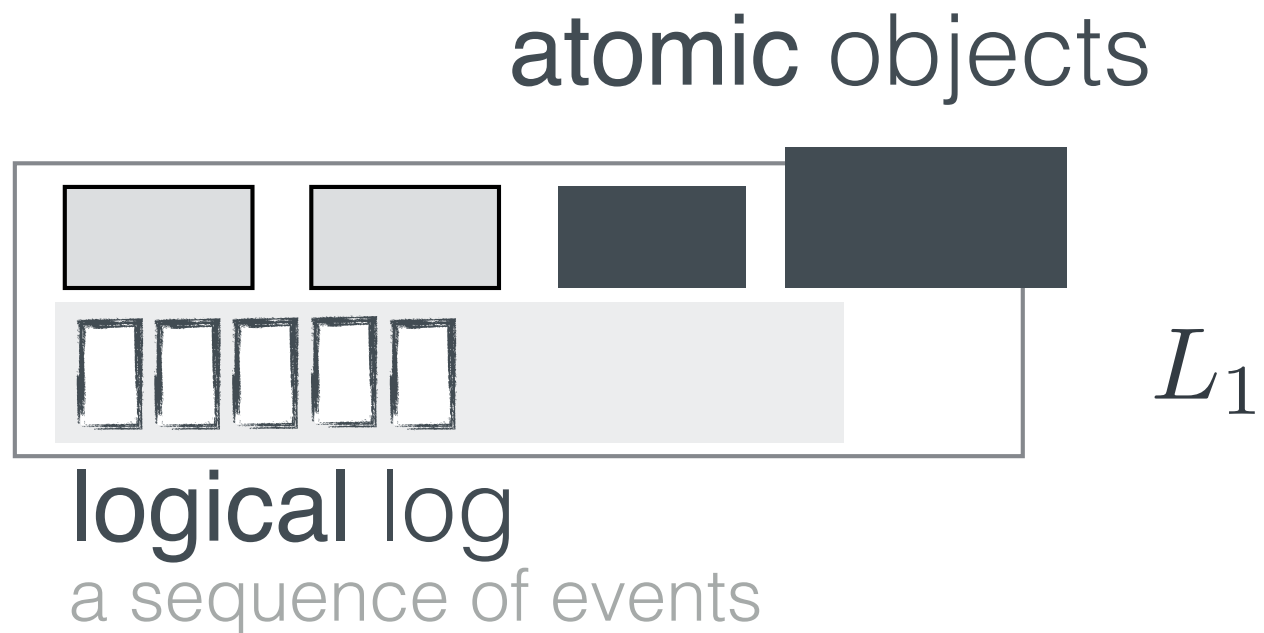
logical log

a sequence of events

Certified Concurrent Layers



Certified Concurrent Layers



Certified Concurrent Layers

to share



L_1

Certified Concurrent Layers



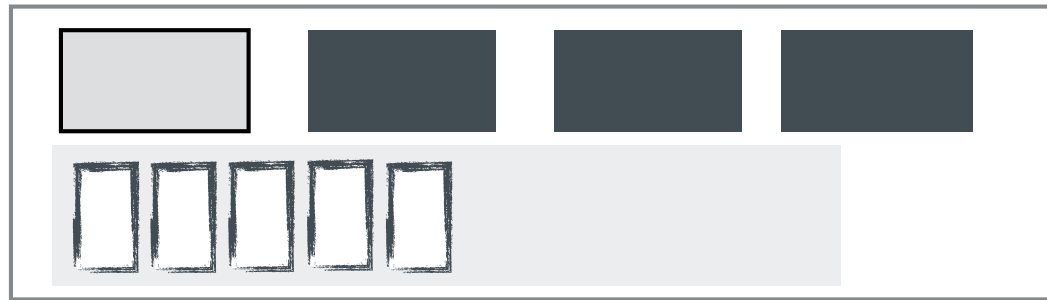
L_2

fine-grained locking



L_1

Concurrent Framework



CPU-local machine

CPU0

CPU1

CPU2

CPU3

machine lifting ↑

multicore machine

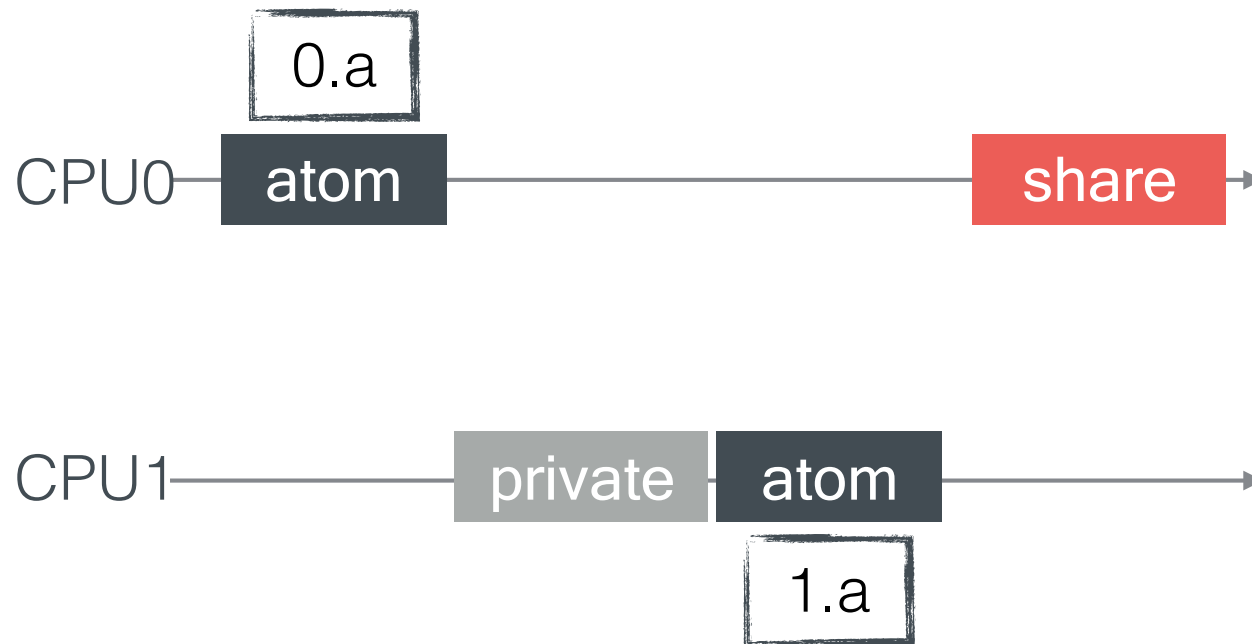
CPU0

CPU1

CPU2

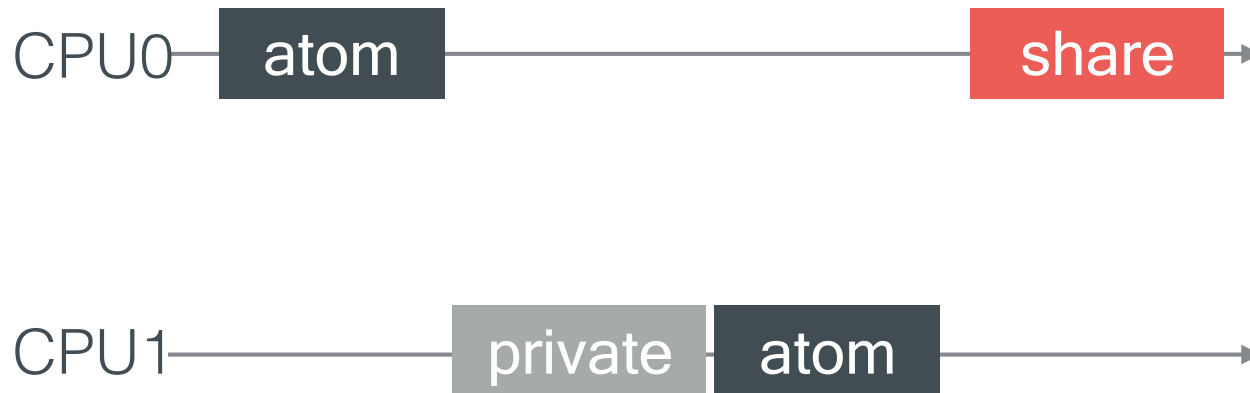
CPU3

step 0: raw x86 multicore model



step 0: raw x86 multicore model

non-determinism



logical log



multicore machine

CPU0

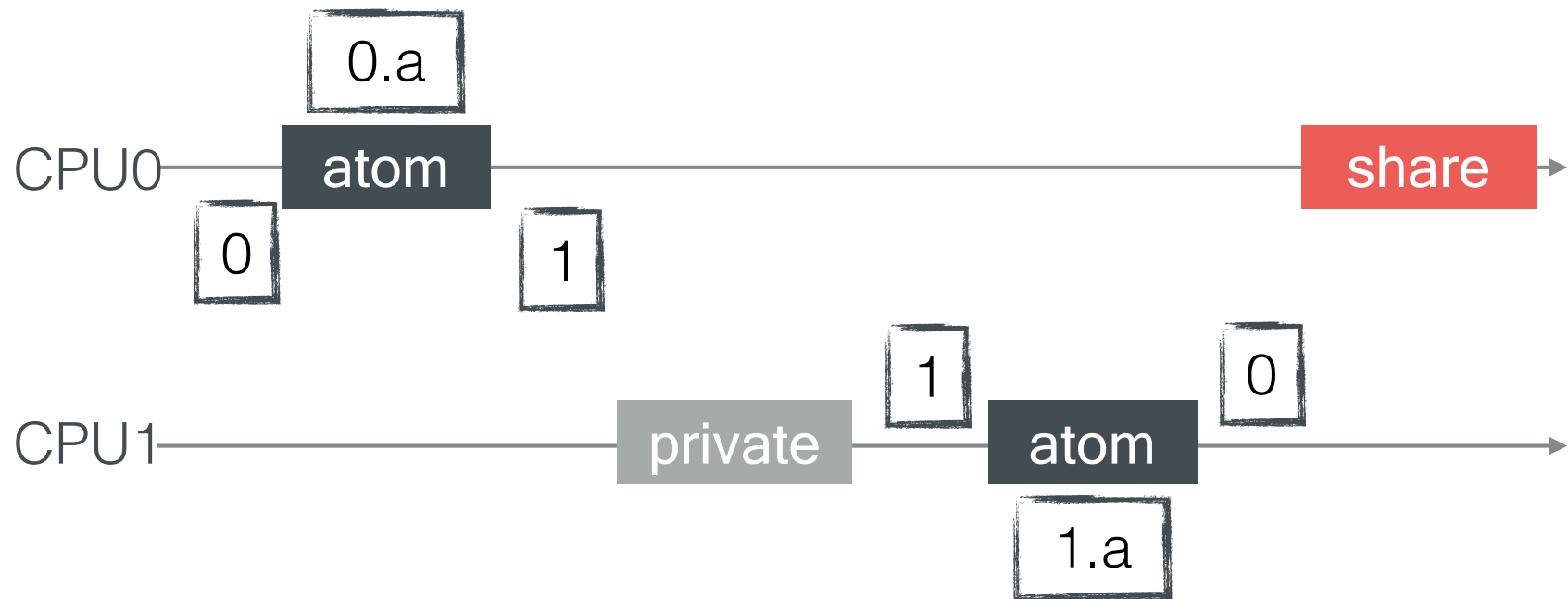
CPU1

CPU2

CPU3

step 0: raw x86 multicore model

non-determinism



multicore machine

CPU0

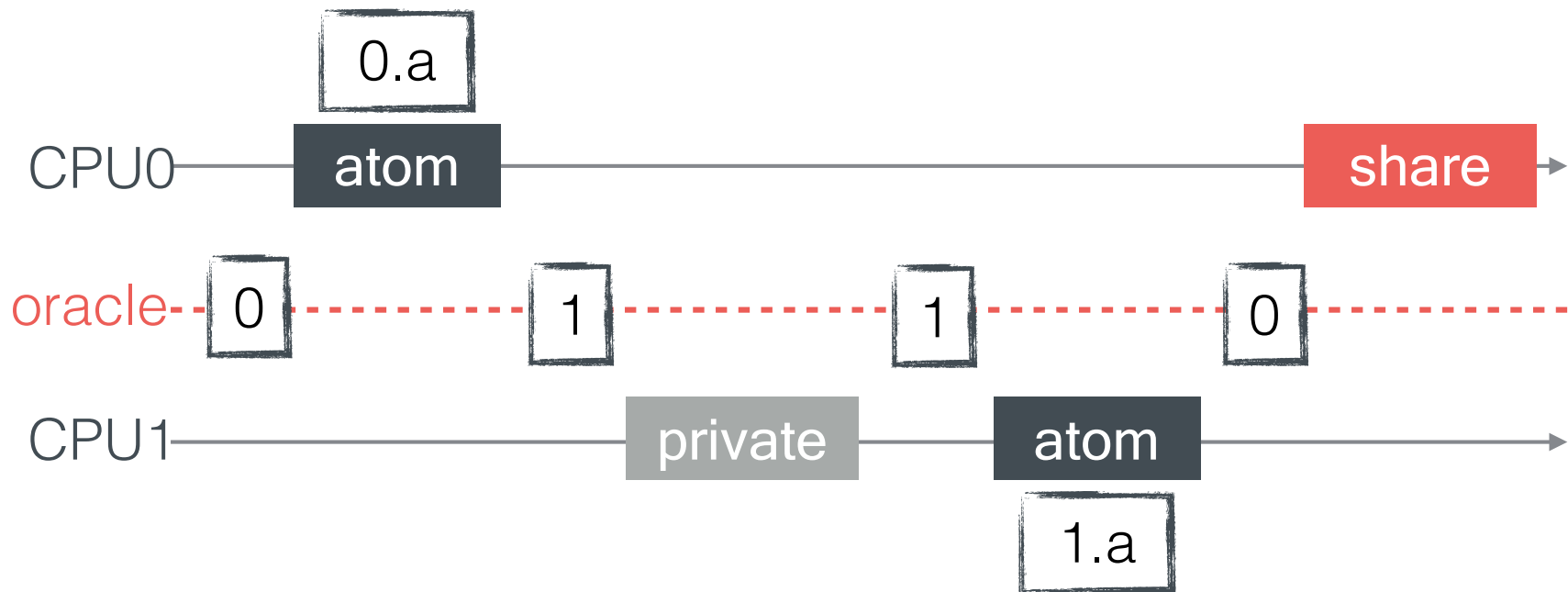
CPU1

CPU2

CPU3

step 0: raw x86 multicore model

non-determinism



multicore machine

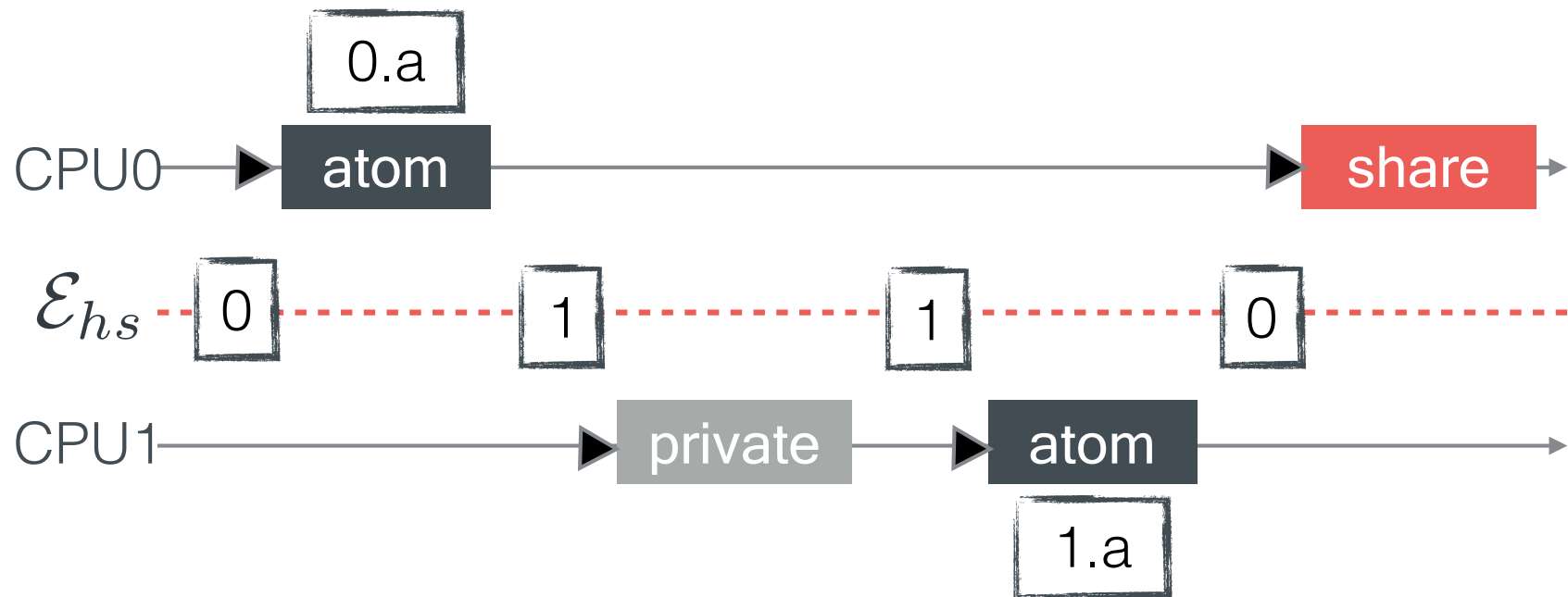
CPU0

CPU1

CPU2

CPU3

step 1: logical hardware scheduler



multicore machine

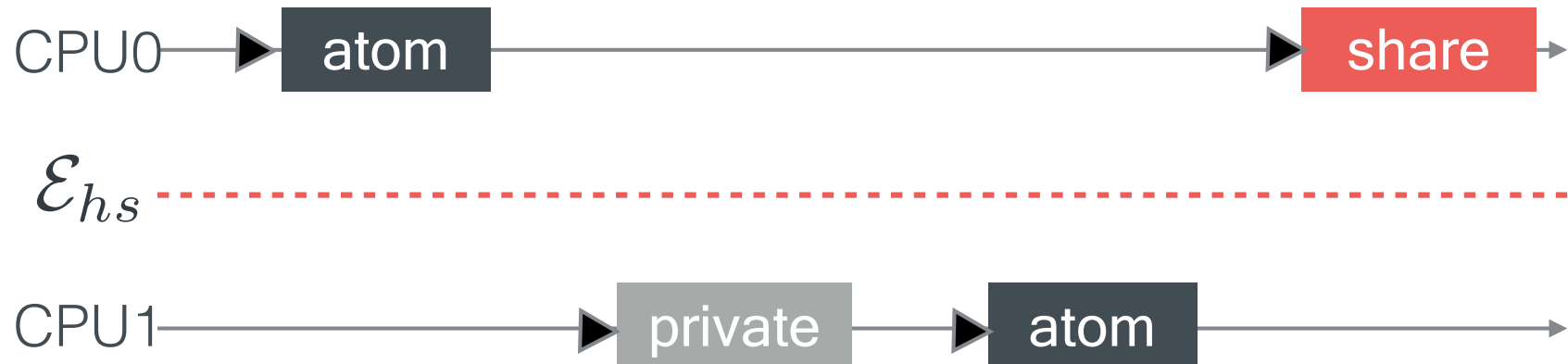
CPU0

CPU1

CPU2

CPU3

step 1: logical hardware scheduler



logical log



multicore machine

CPU0

CPU1

CPU2

CPU3

step 1: logical hardware scheduler

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

multicore machine

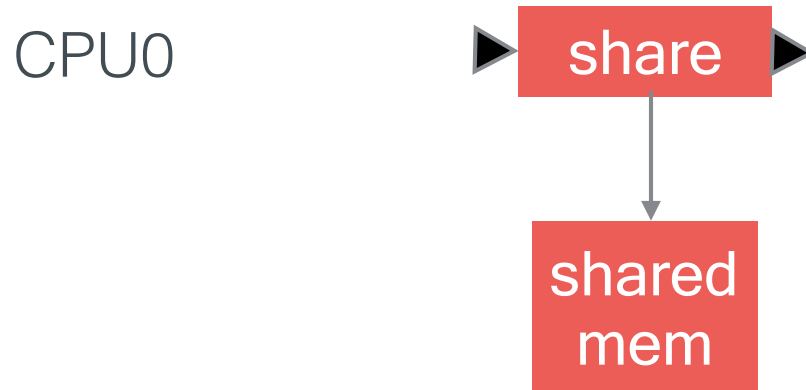
CPU0

CPU1

CPU2

CPU3

step 2: push/pull memory model



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

machine with hardware scheduler

multicore machine

CPU0

CPU1

CPU2

CPU3

step 2: push/pull memory model



logical
copy

shared
mem

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

machine with hardware scheduler

multicore machine

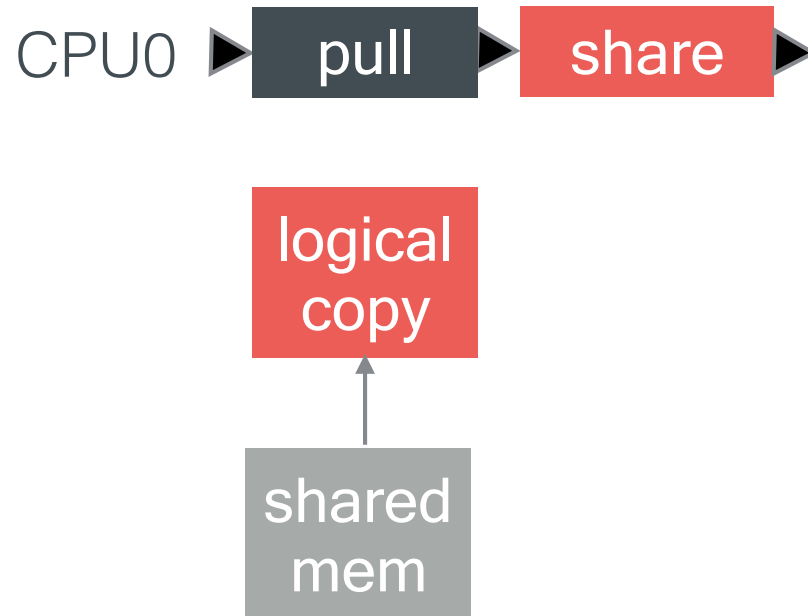
CPU0

CPU1

CPU2

CPU3

step 2: push/pull memory model



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

machine with hardware scheduler

multicore machine

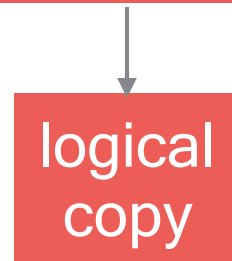
CPU0

CPU1

CPU2

CPU3

step 2: push/pull memory model



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

machine with hardware scheduler

multicore machine

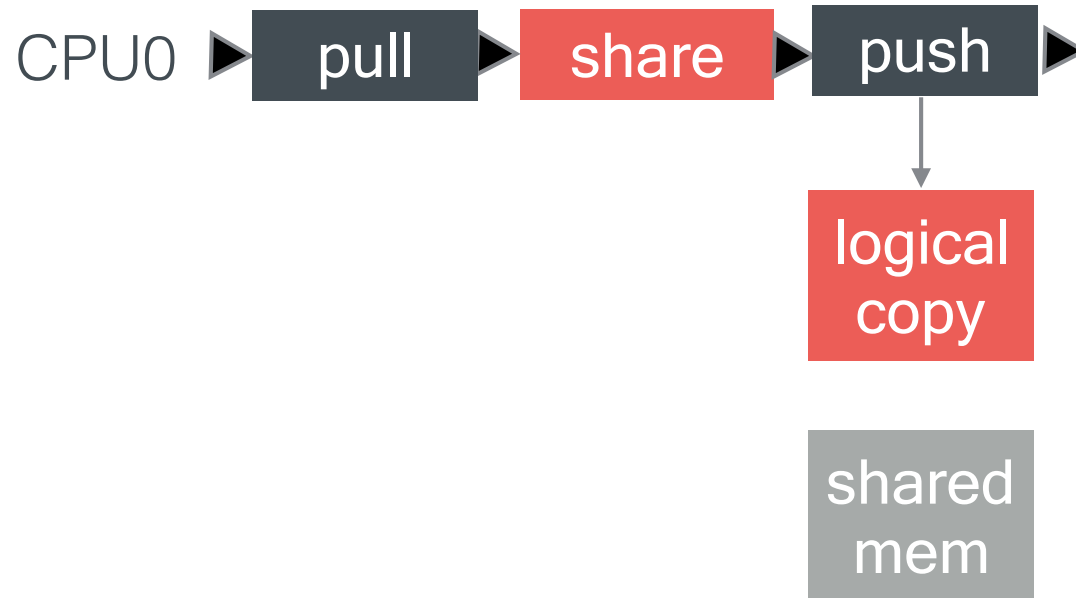
CPU0

CPU1

CPU2

CPU3

step 2: push/pull memory model



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

machine with hardware scheduler

multicore machine

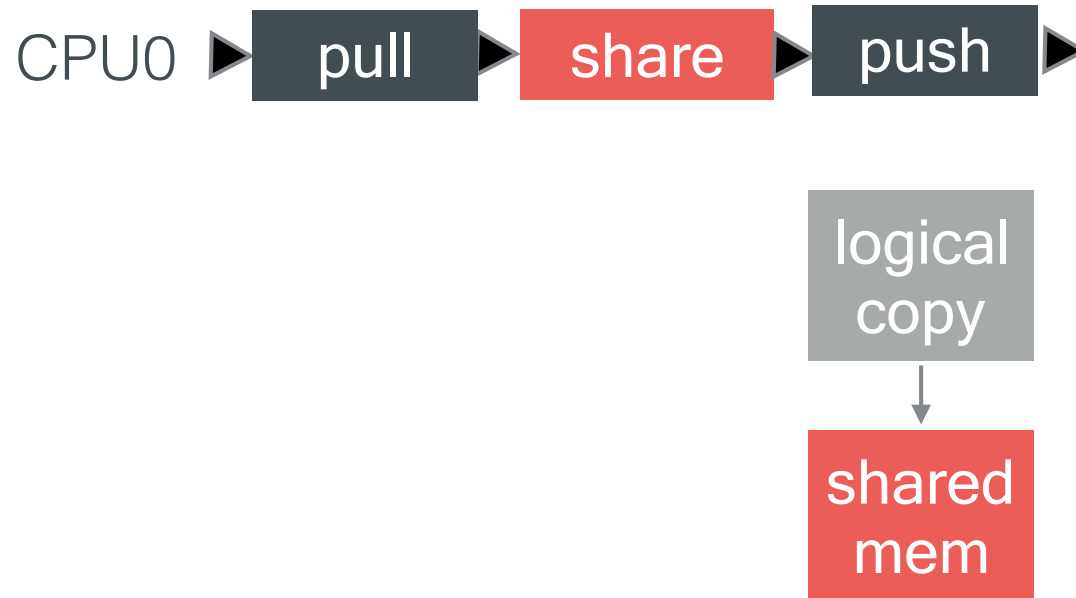
CPU0

CPU1

CPU2

CPU3

step 2: push/pull memory model



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

machine with hardware scheduler

multicore machine

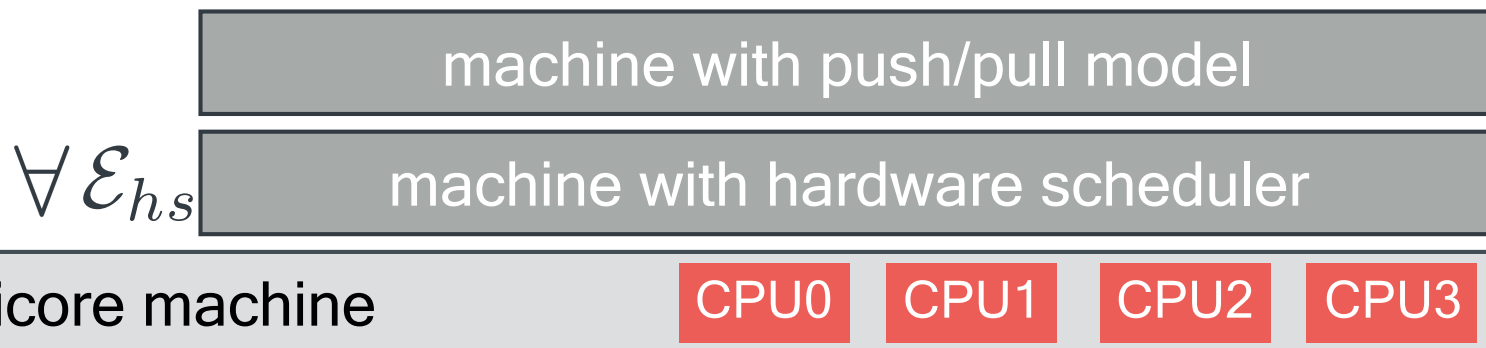
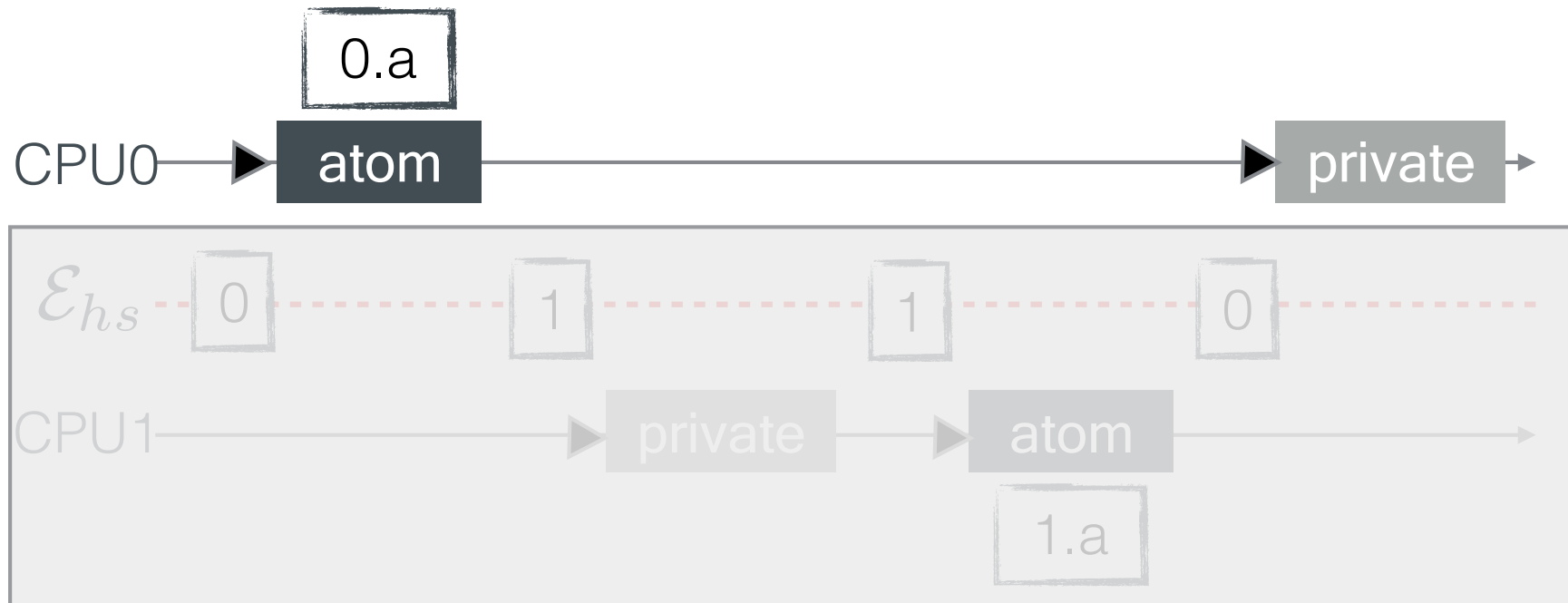
CPU0

CPU1

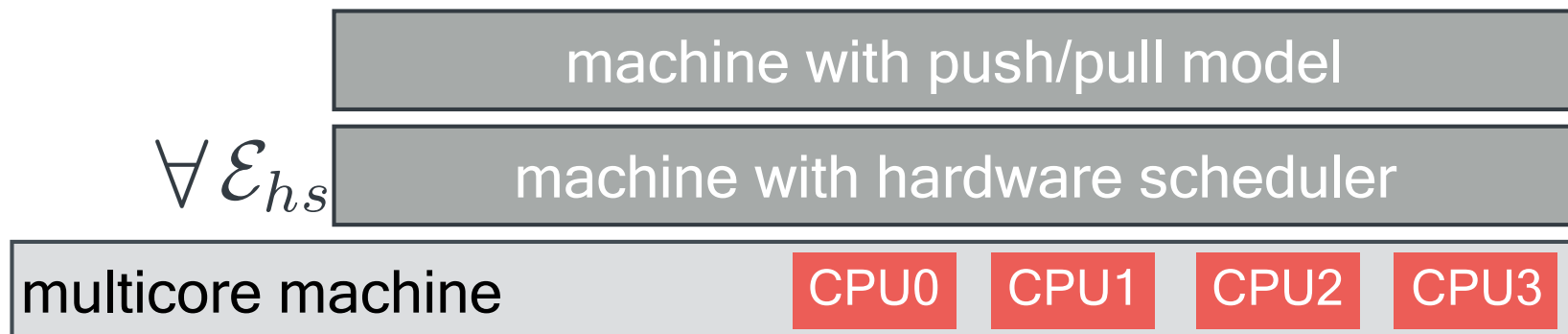
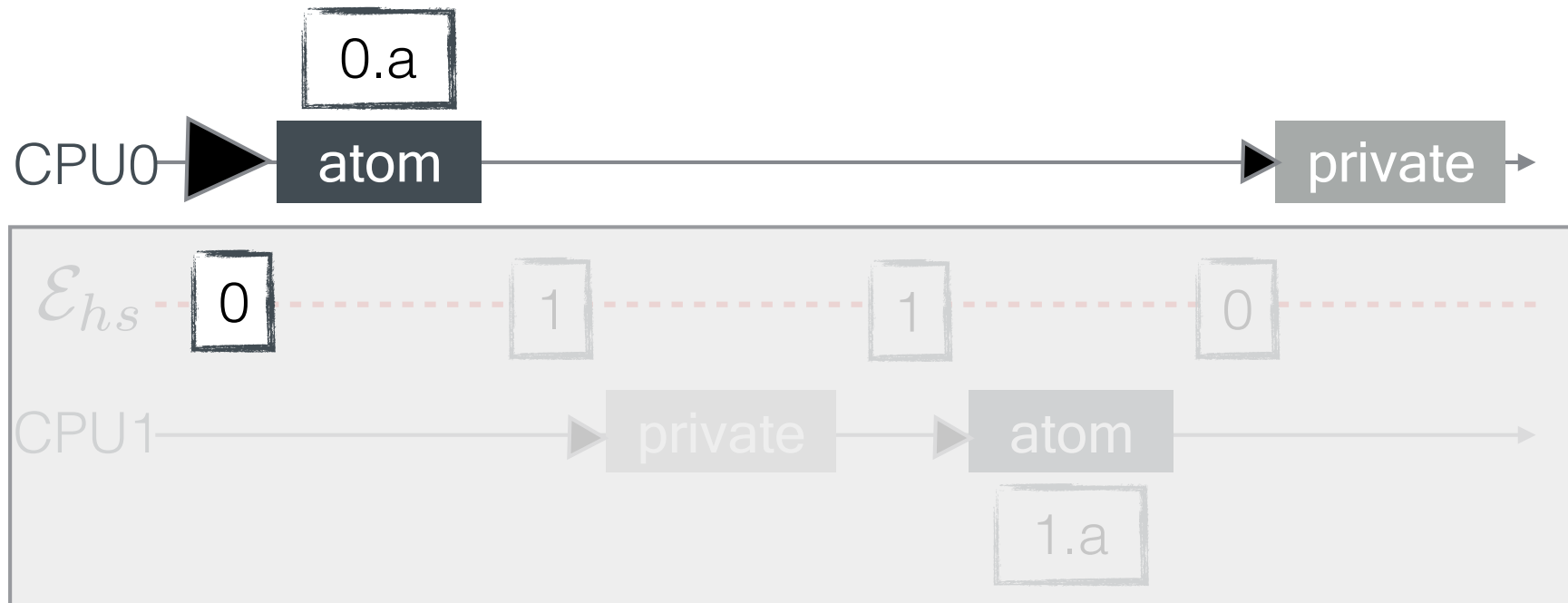
CPU2

CPU3

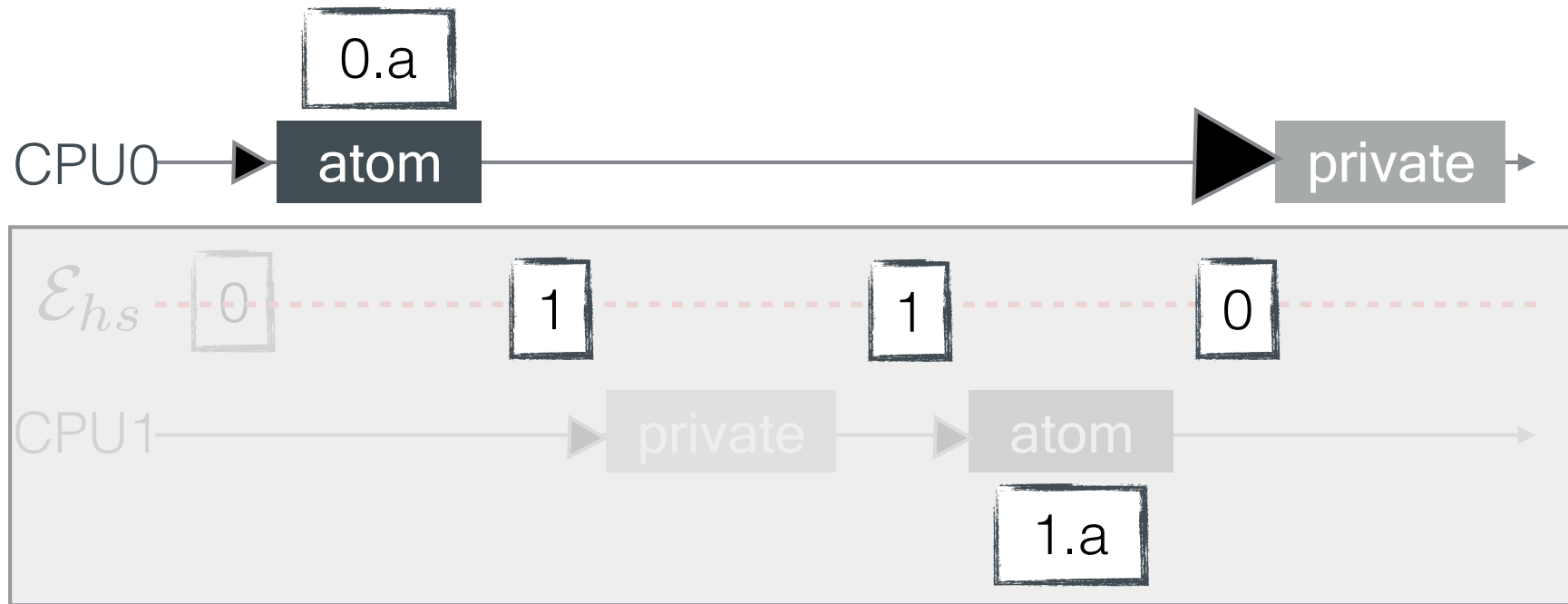
step 3: environment context model



step 3: environment context model



step 3: environment context model



machine with push/pull model

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

machine with hardware scheduler

multicore machine

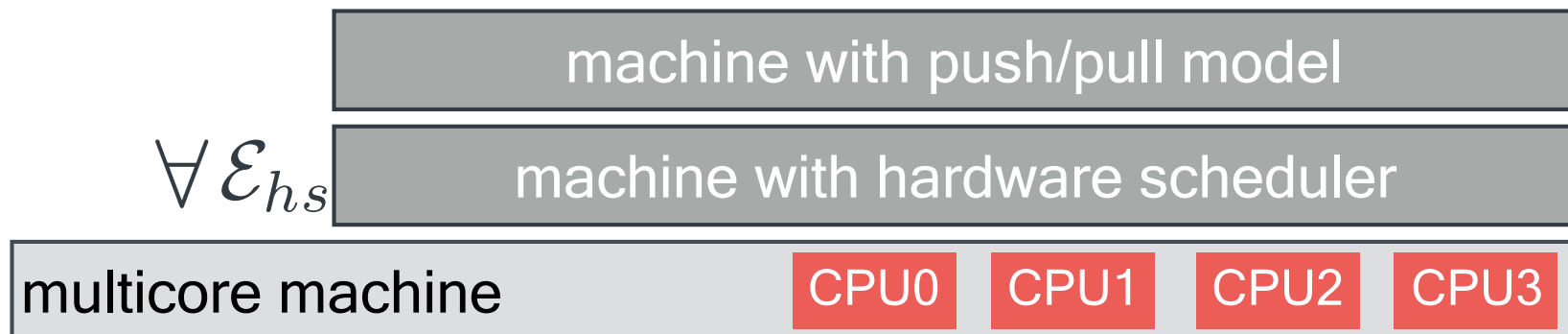
CPU0

CPU1

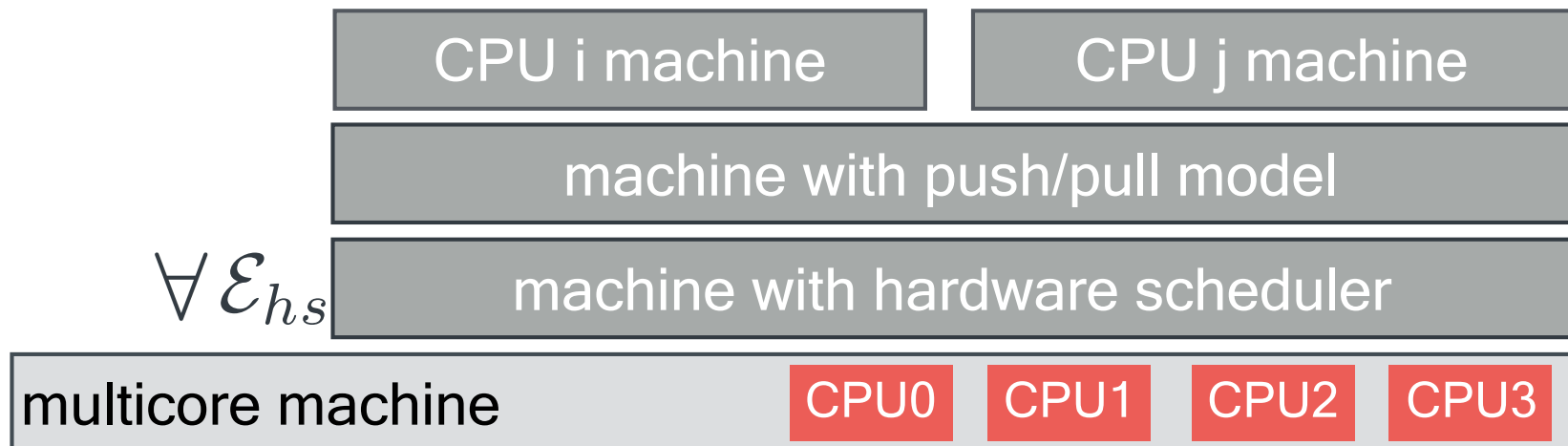
CPU2

CPU3

step 3: environment context model

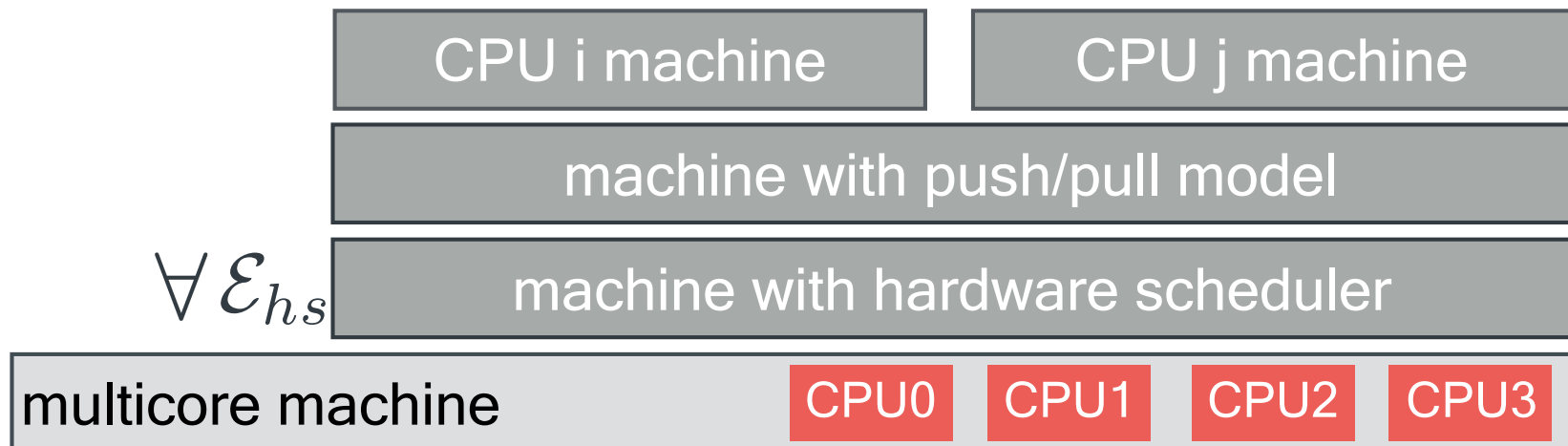


step 4: remove unnecessary interleaving



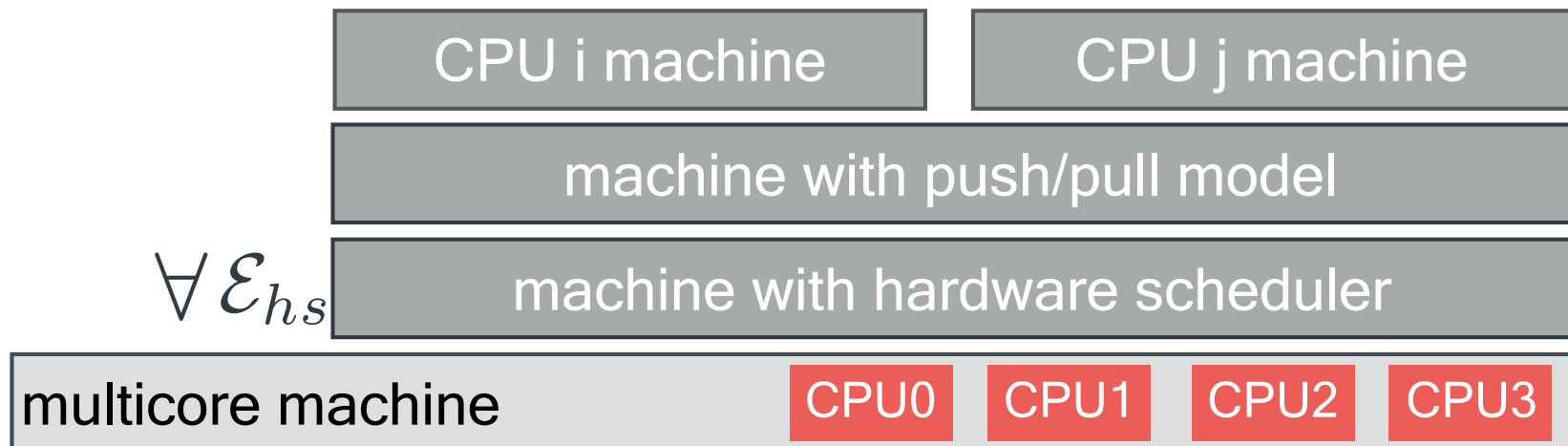
step 4: remove unnecessary interleaving

shuffle

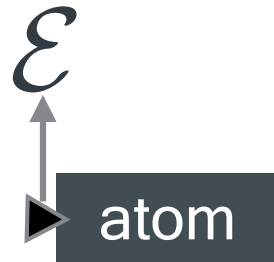


step 4: remove unnecessary interleaving

merge

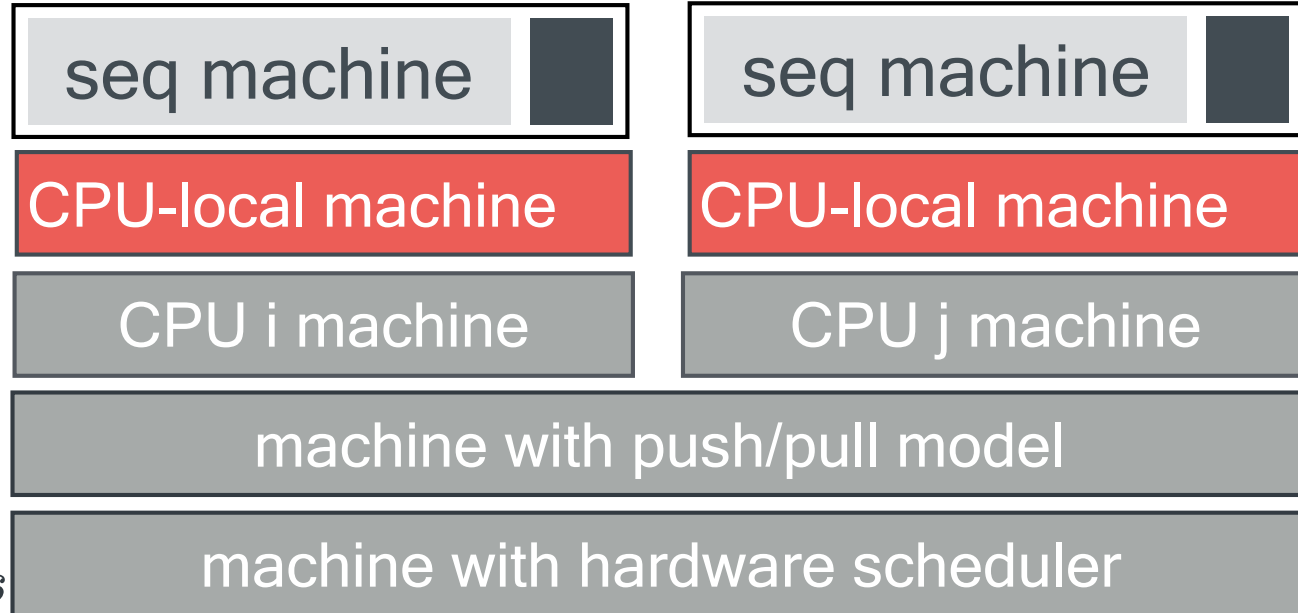


Machine Lifting



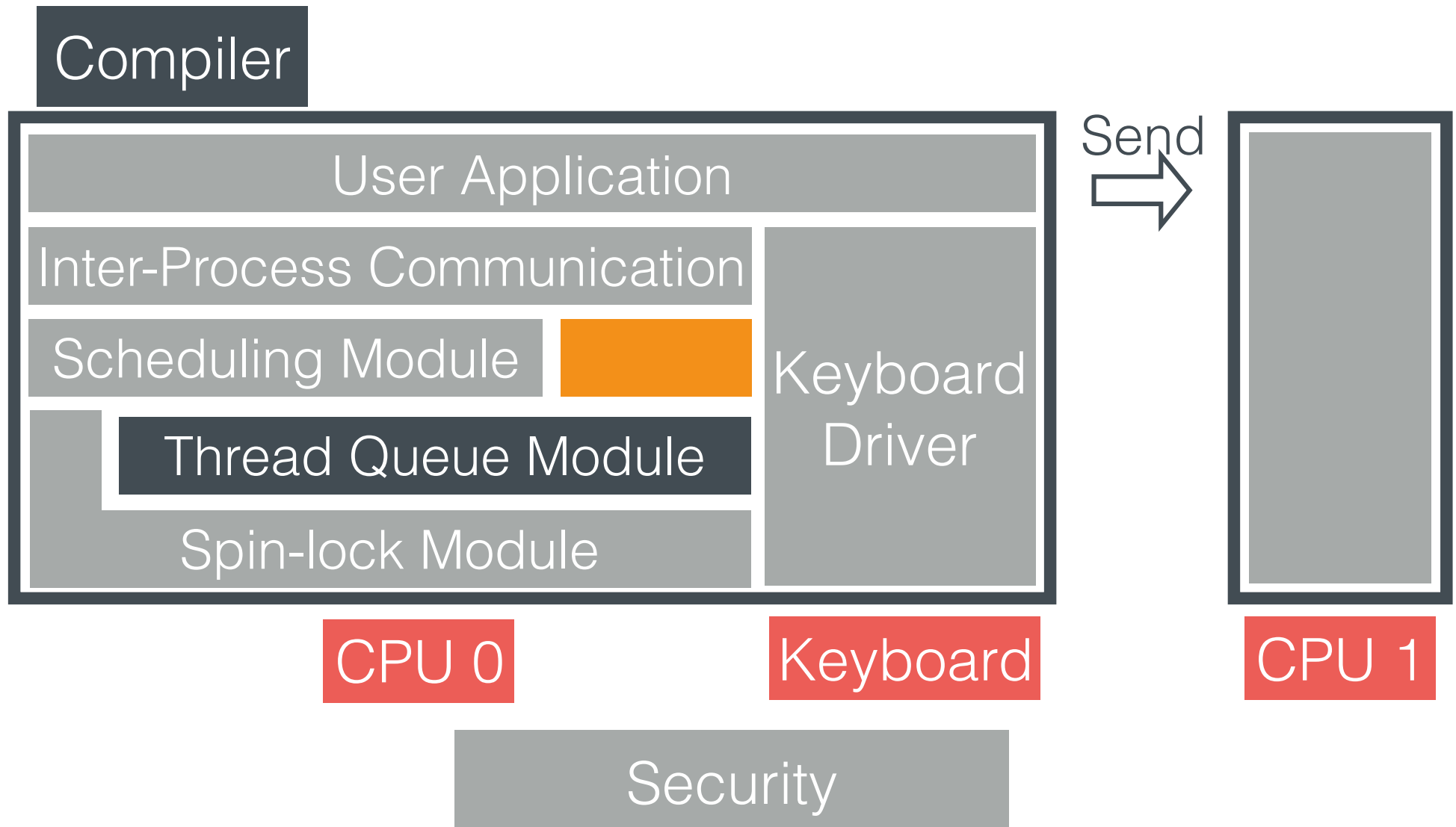
logical log

0	0.a	1	1	1.a	0
---	-----	---	---	-----	---



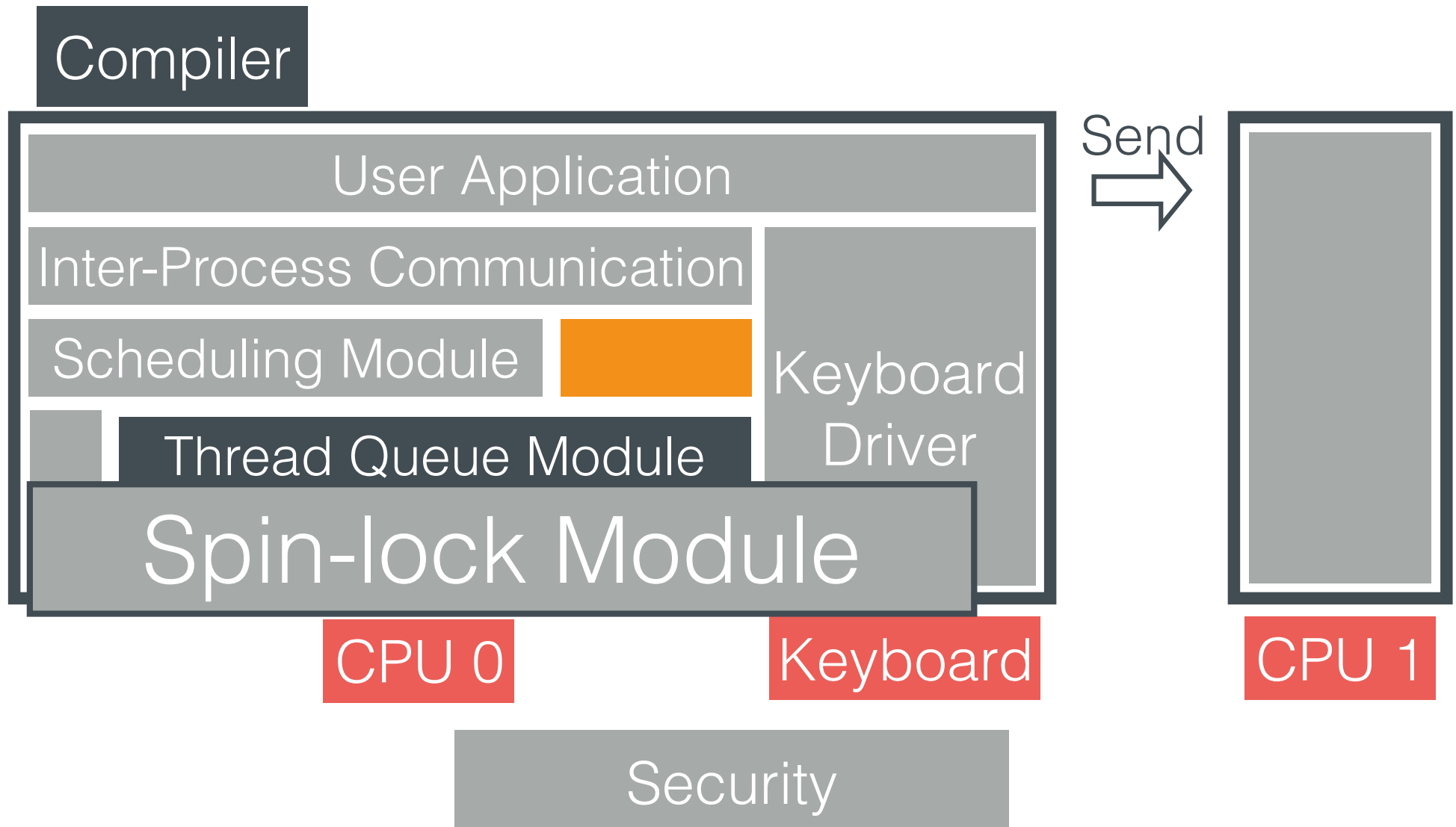
Case Study

Build a Certified System



Case Study

Build a Certified System



Acquire Lock Specification

safely
pull

logical
copy



Acquire Lock Specification

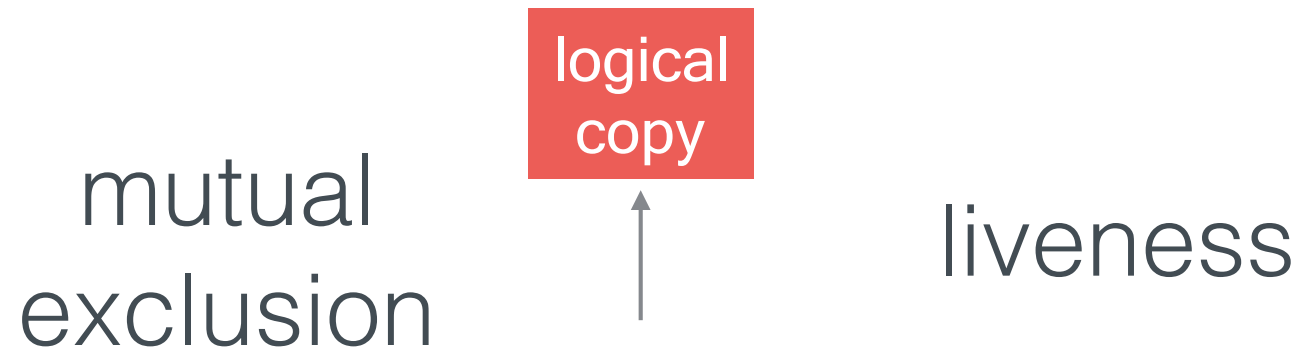
safely
pull

logical
copy



pull will
eventually return

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

```
void acq_lock (uint i)
{
    uint64 t = FAI_ticket
    while (get_now(i) != t)
    { }

    pull(i);
}
```

FAI
ticket

C



Example: Ticket Lock

mutual exclusion + liveness

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

    while (▶get_now (
    { }

    ▶pull (i);
}
```



FAI
ticket

Example: Ticket Lock

mutual exclusion + liveness

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

    while (▶get_now (get
    { }
    { }

    ▶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) != 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

FAI
ticket

get
now

get
now

pull

Example: Ticket Lock

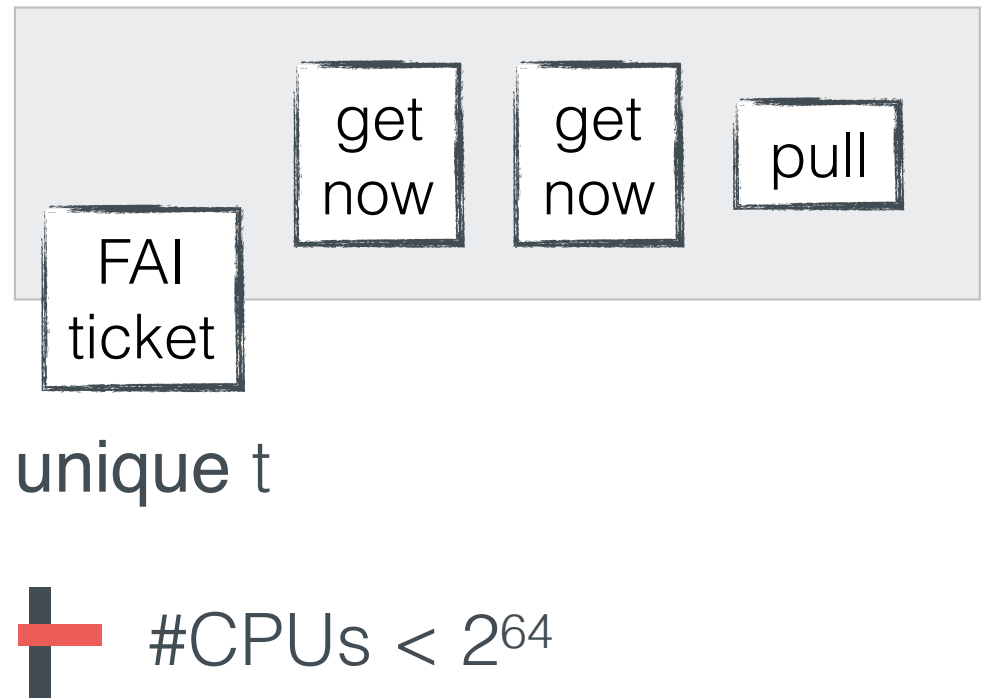
mutual exclusion + liveness

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

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

    pull (i);
}
```

C



Example: Ticket Lock

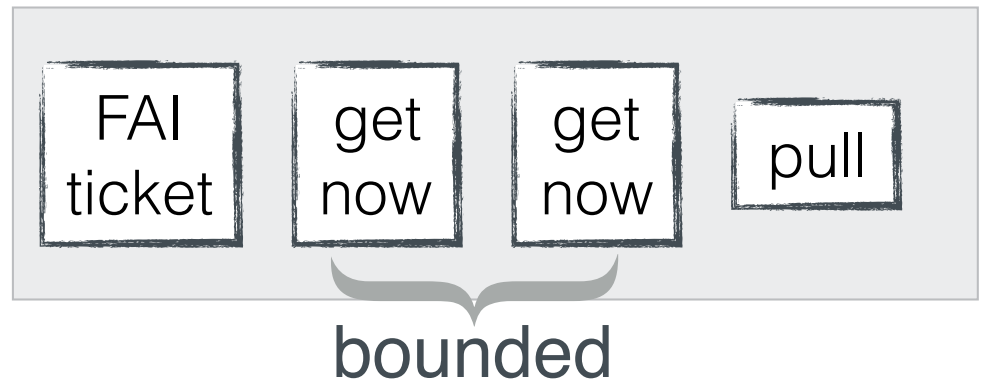
mutual exclusion + **liveness**

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

    while (▶get_now (i) != t)
    { }

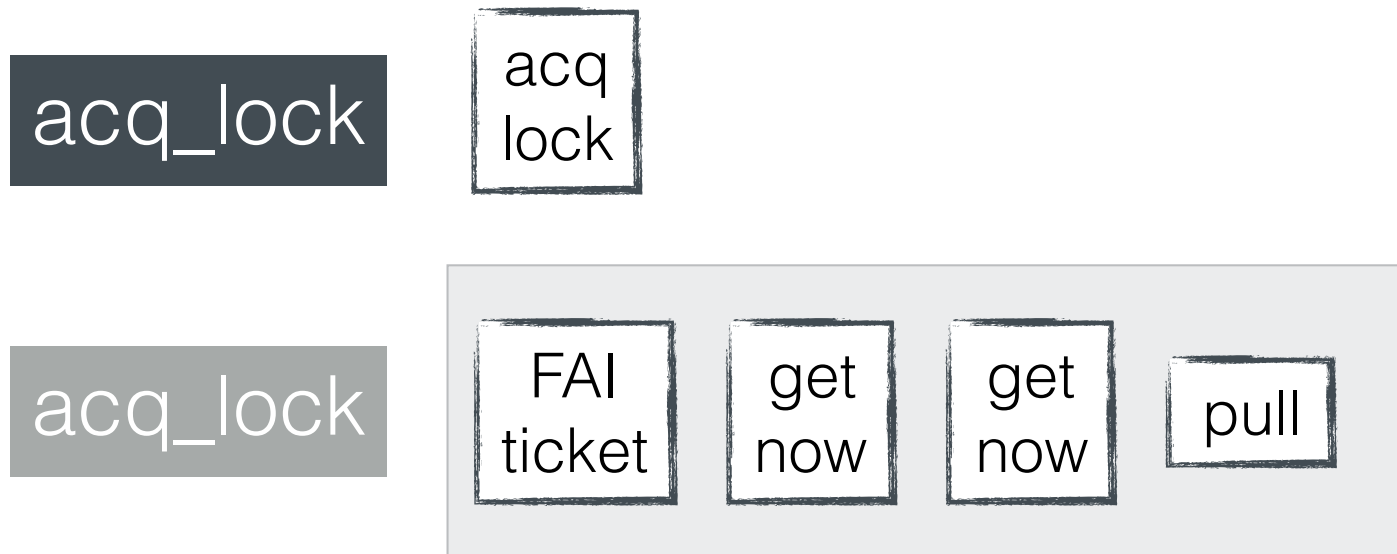
    ▶pull (i);
}
```

C



- #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)
    { }

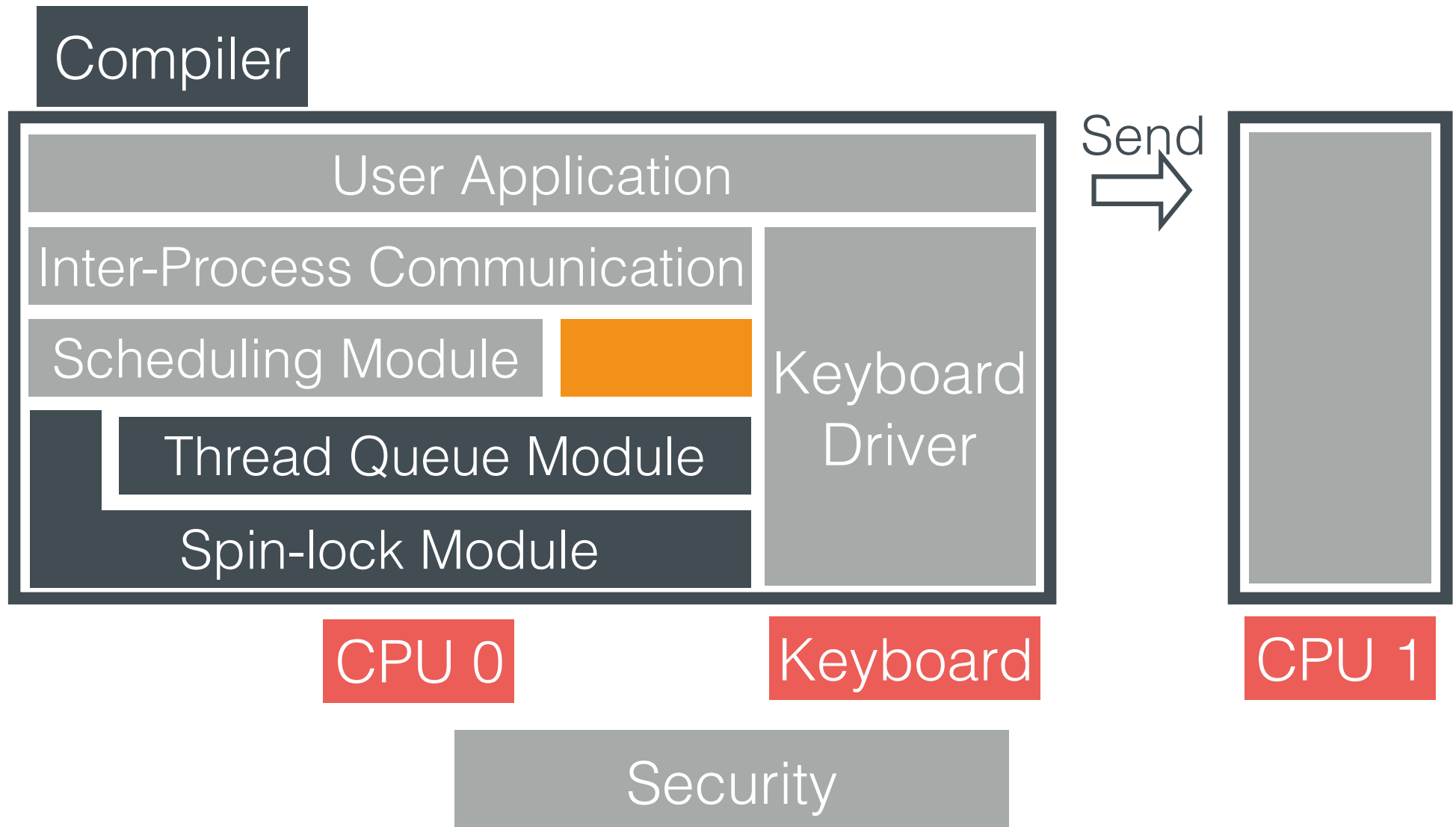
    pull (i);
}
```



- bug in the original implementation
- mutual exclusion will be violated when there is an integer overflow for t

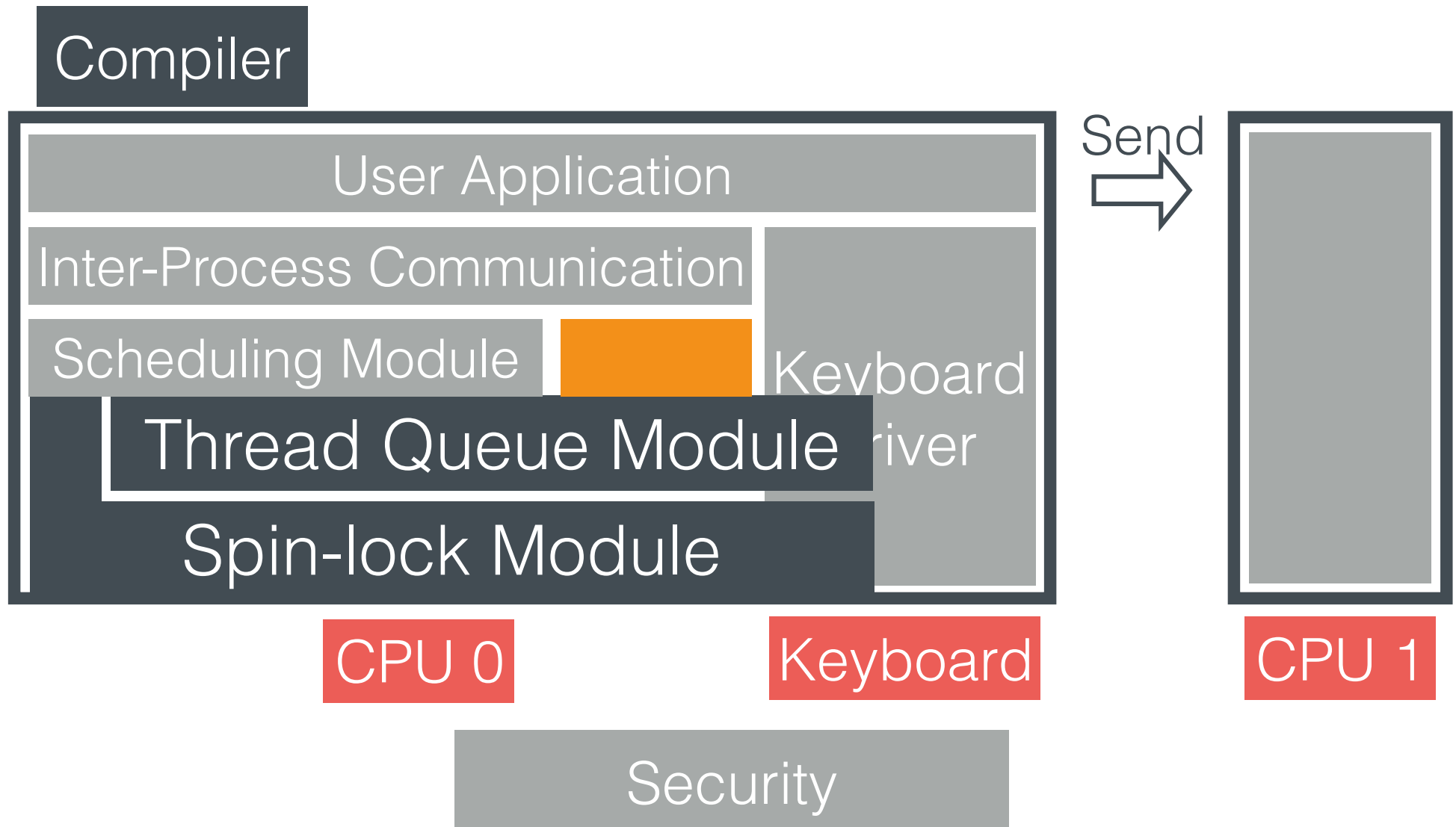
Case Study

Build a Certified System

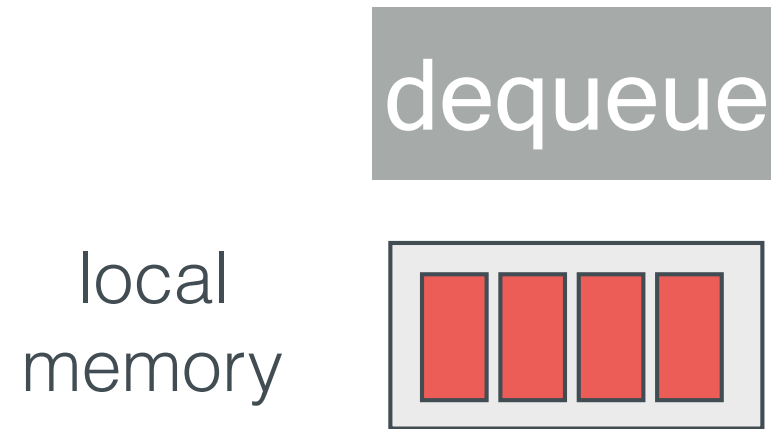


Case Study

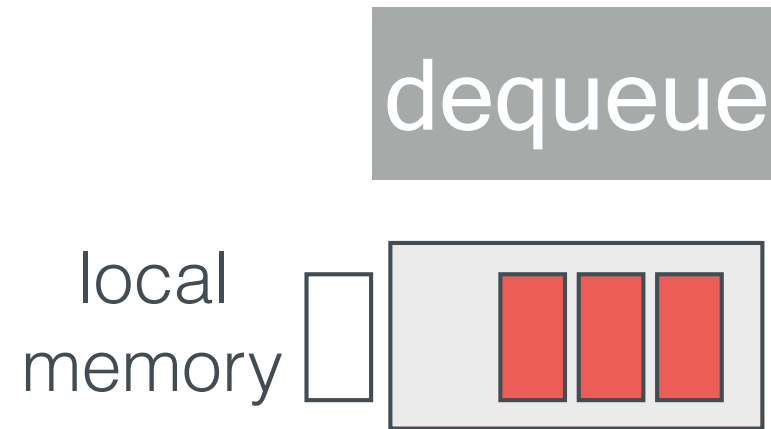
Build a Certified System



Example: Shared Thread Queue



Example: Shared Thread Queue



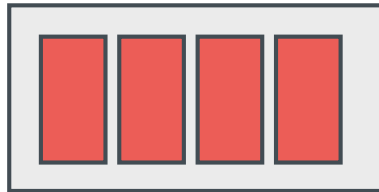
Example: Shared Thread Queue

acq
lock

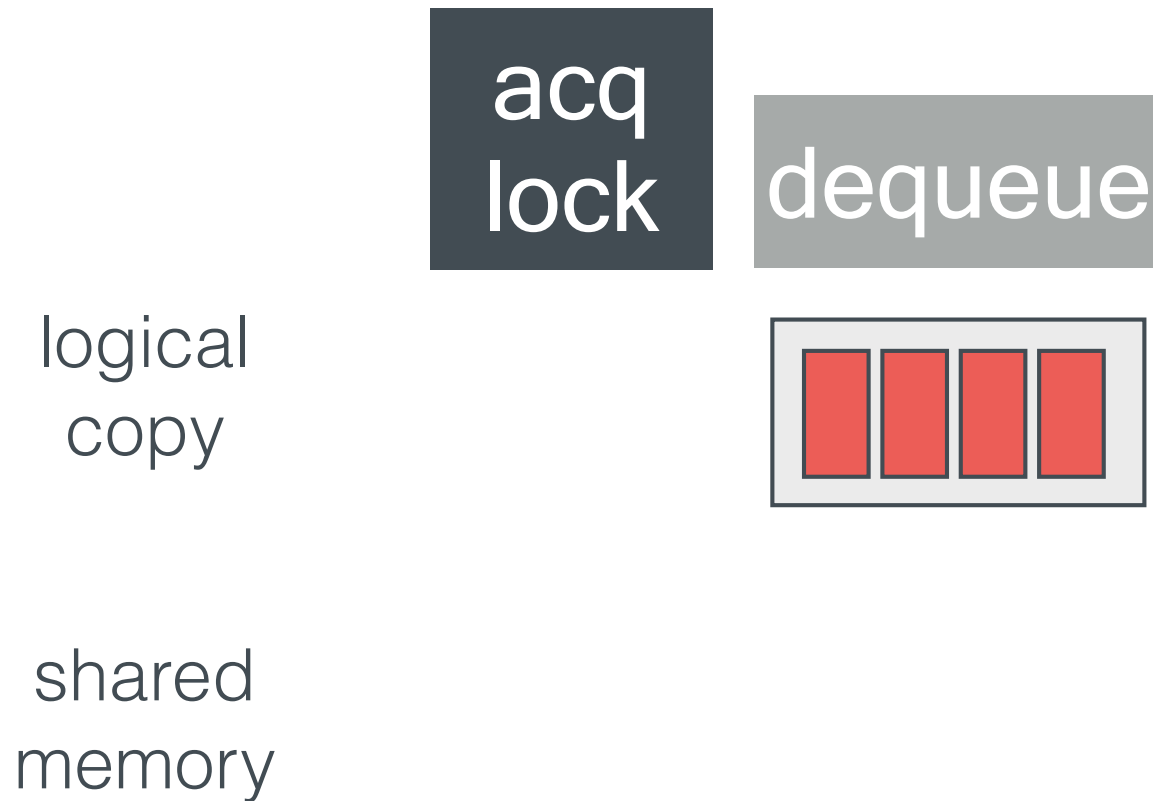
dequeue

logical
copy

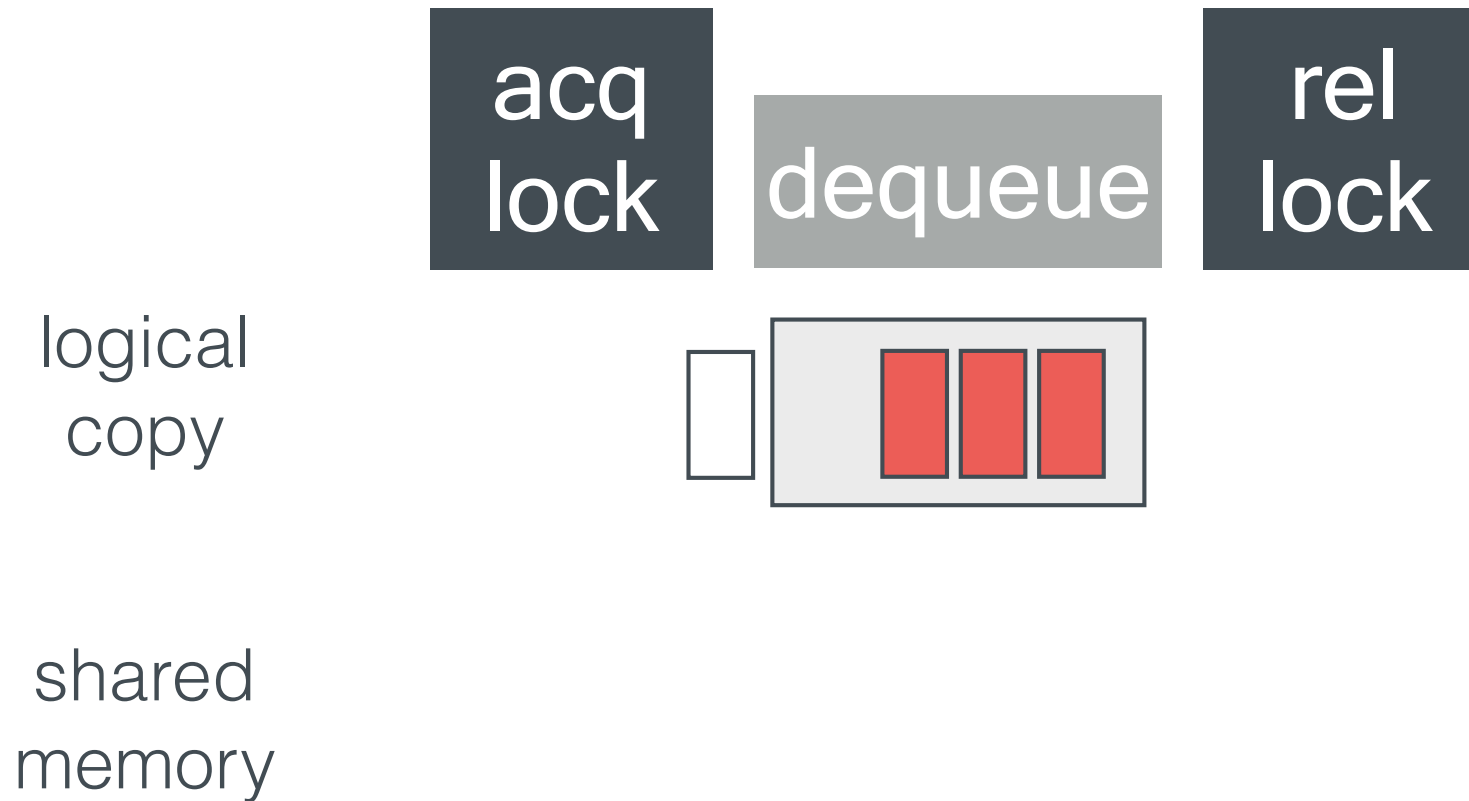
shared
memory



Example: Shared Thread Queue



Example: Shared Thread Queue

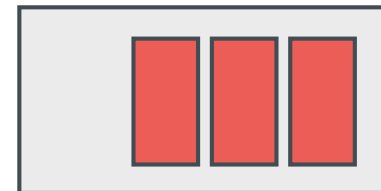


Example: Shared Thread Queue



logical
copy

shared
memory

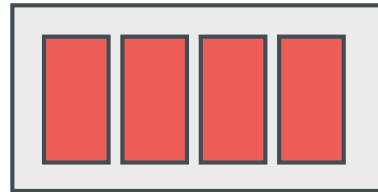


Example: Shared Thread Queue

dequeue

deq

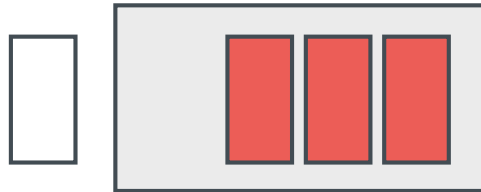
shared
memory



Example: Shared Thread Queue

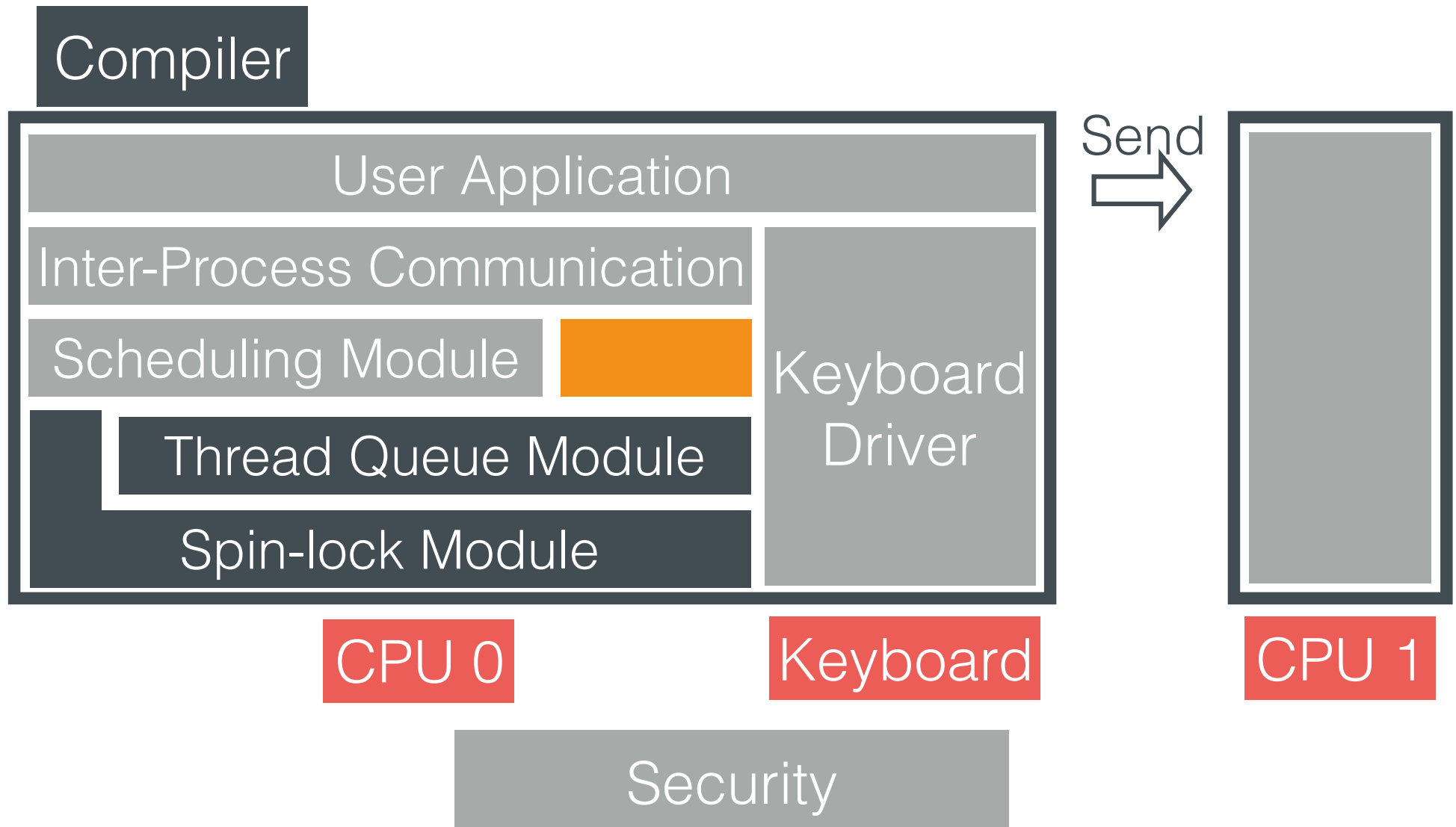
dequeue

shared
memory



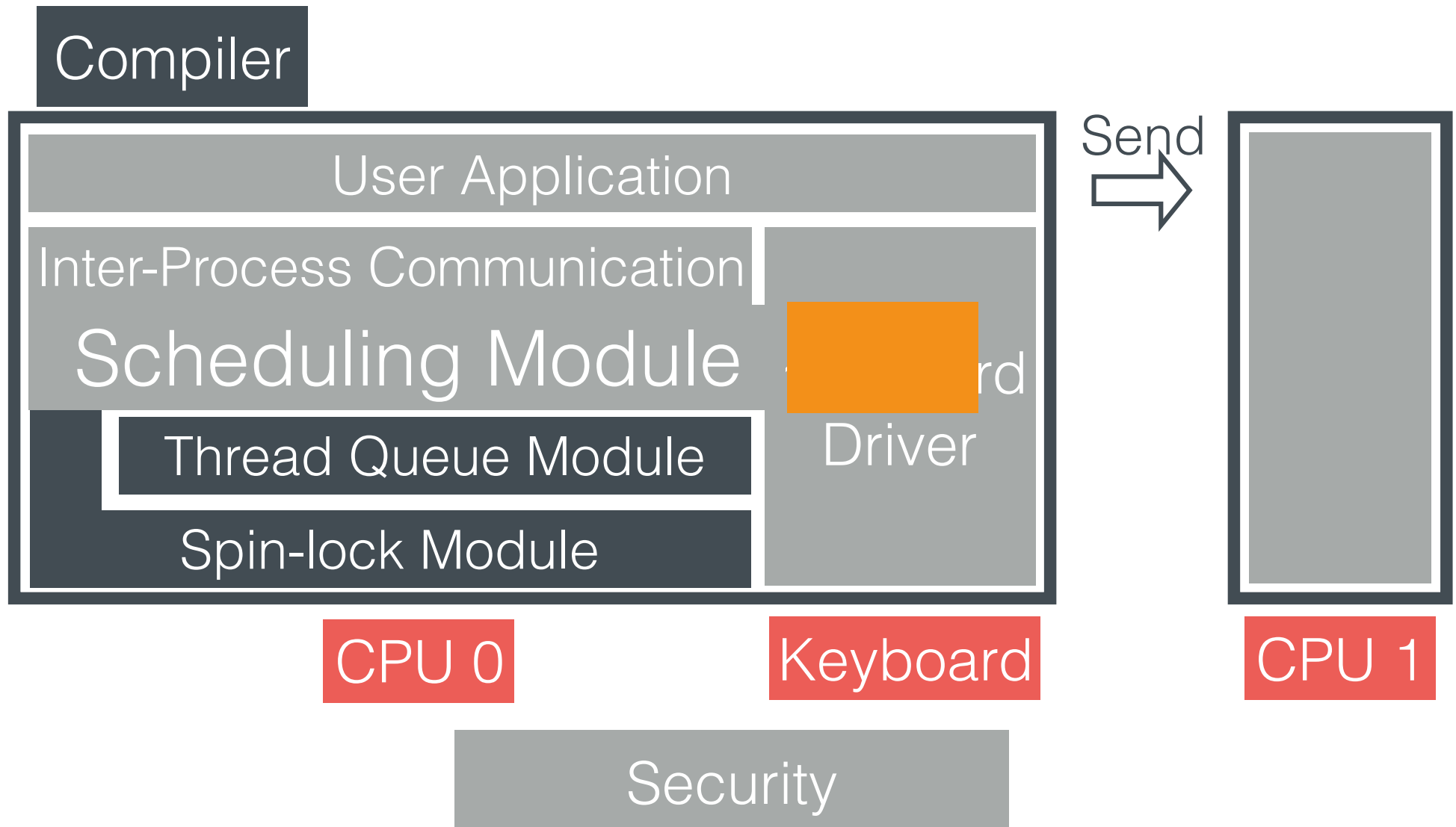
Case Study

Build a Certified System



Case Study

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]

thread-local machine

Software Scheduler

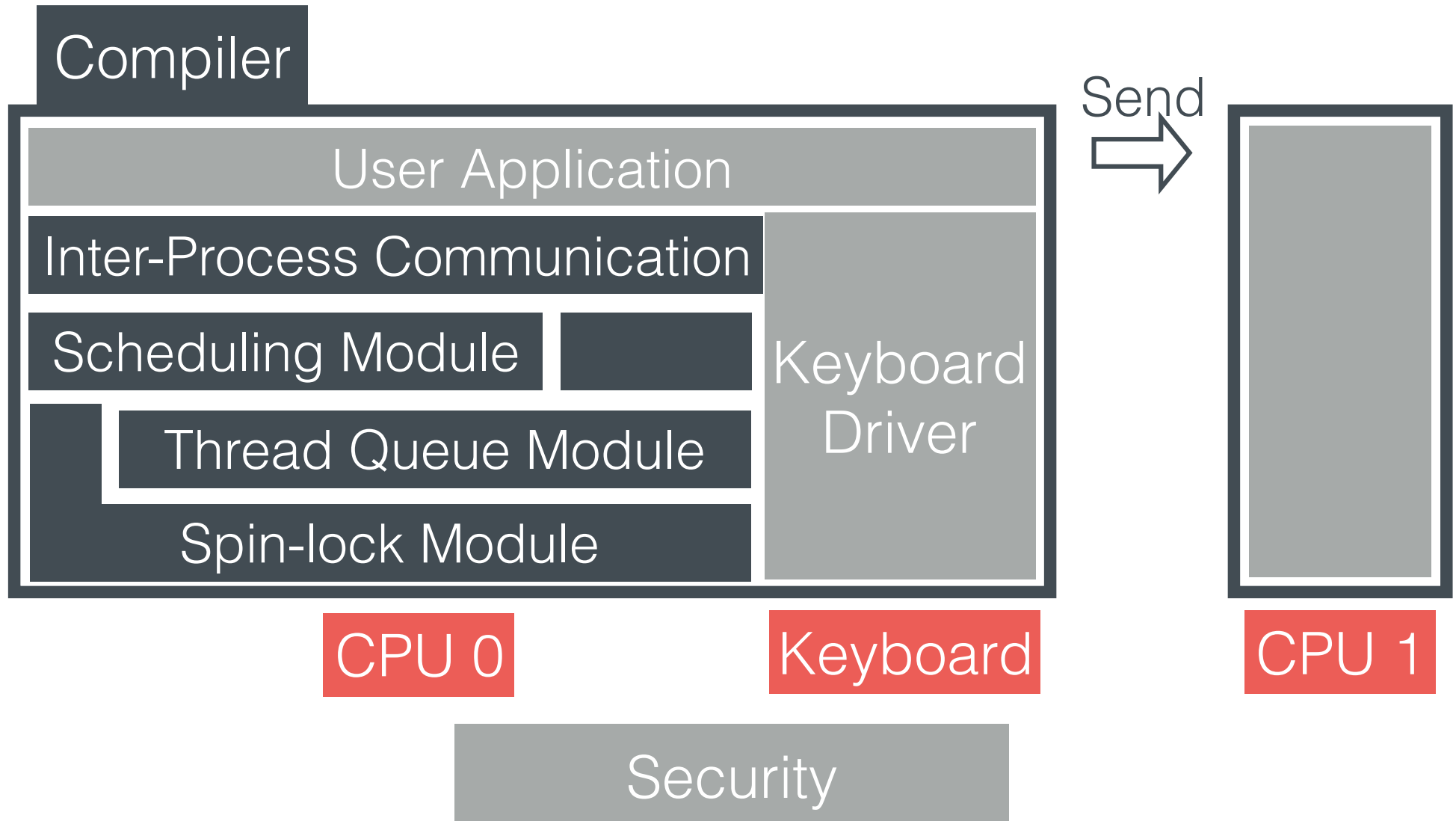
sleep

yield

wakeup

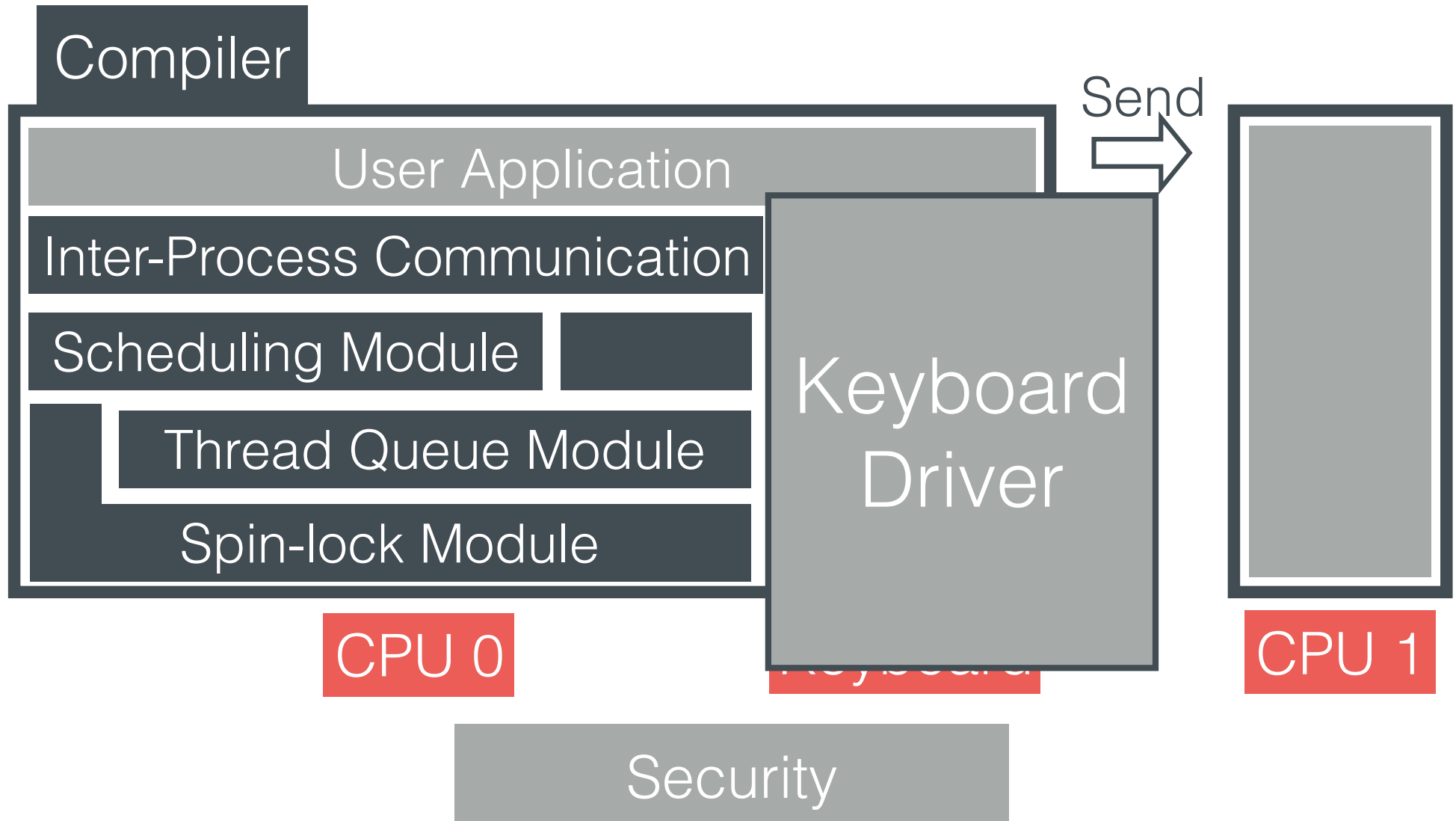
Case Study

Build a Certified System

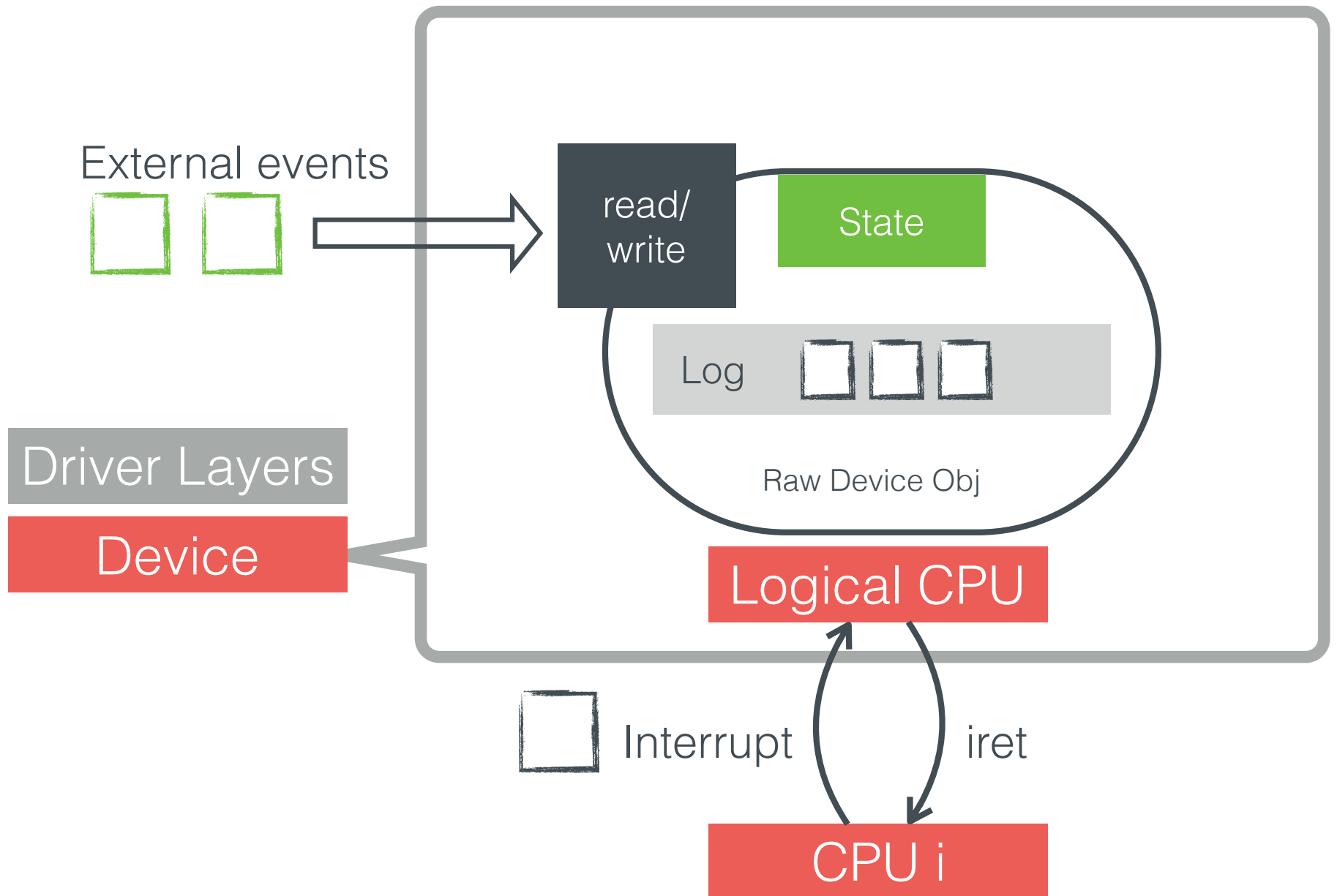


Case Study

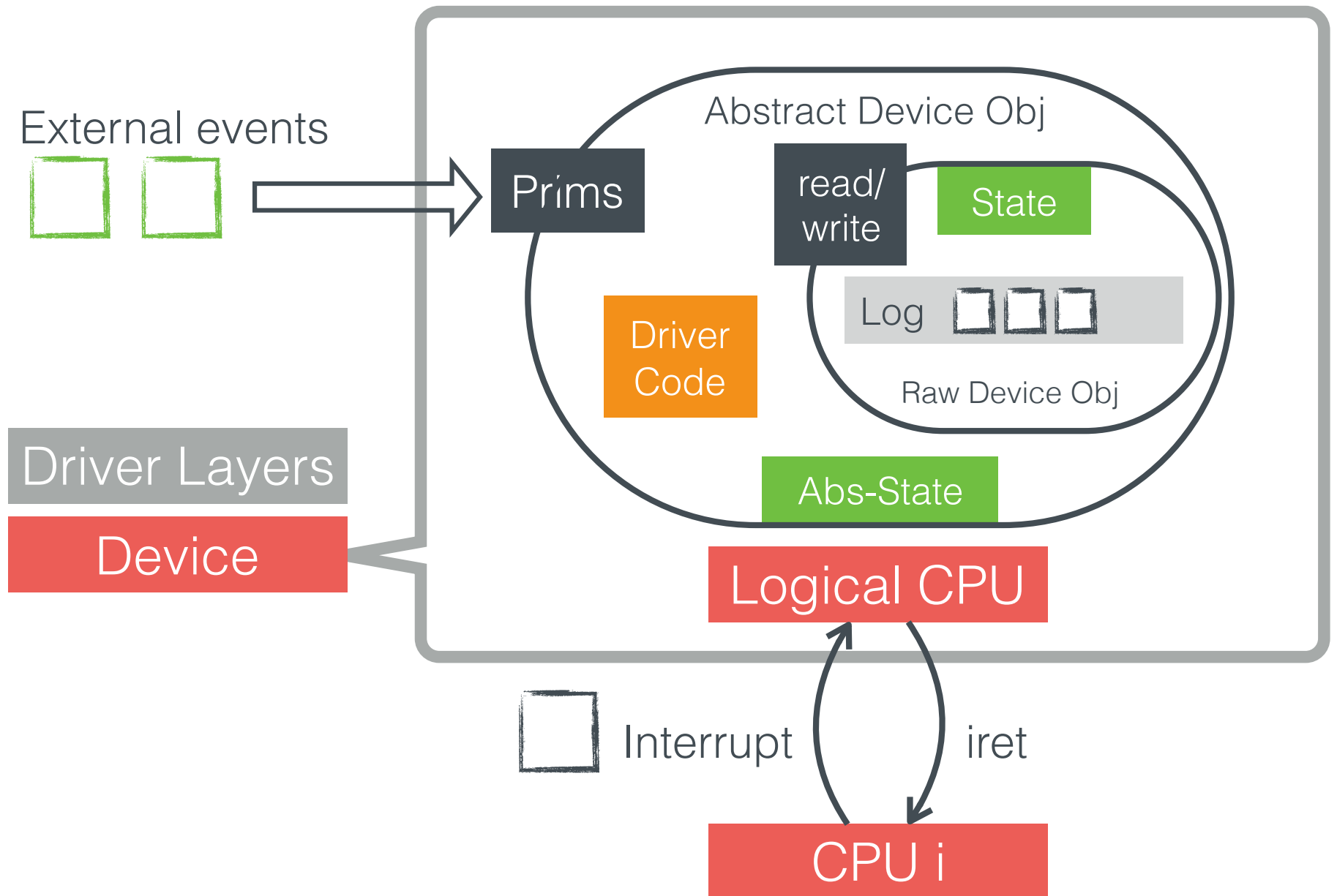
Build a Certified System



Device Driver [PLDI16'a]

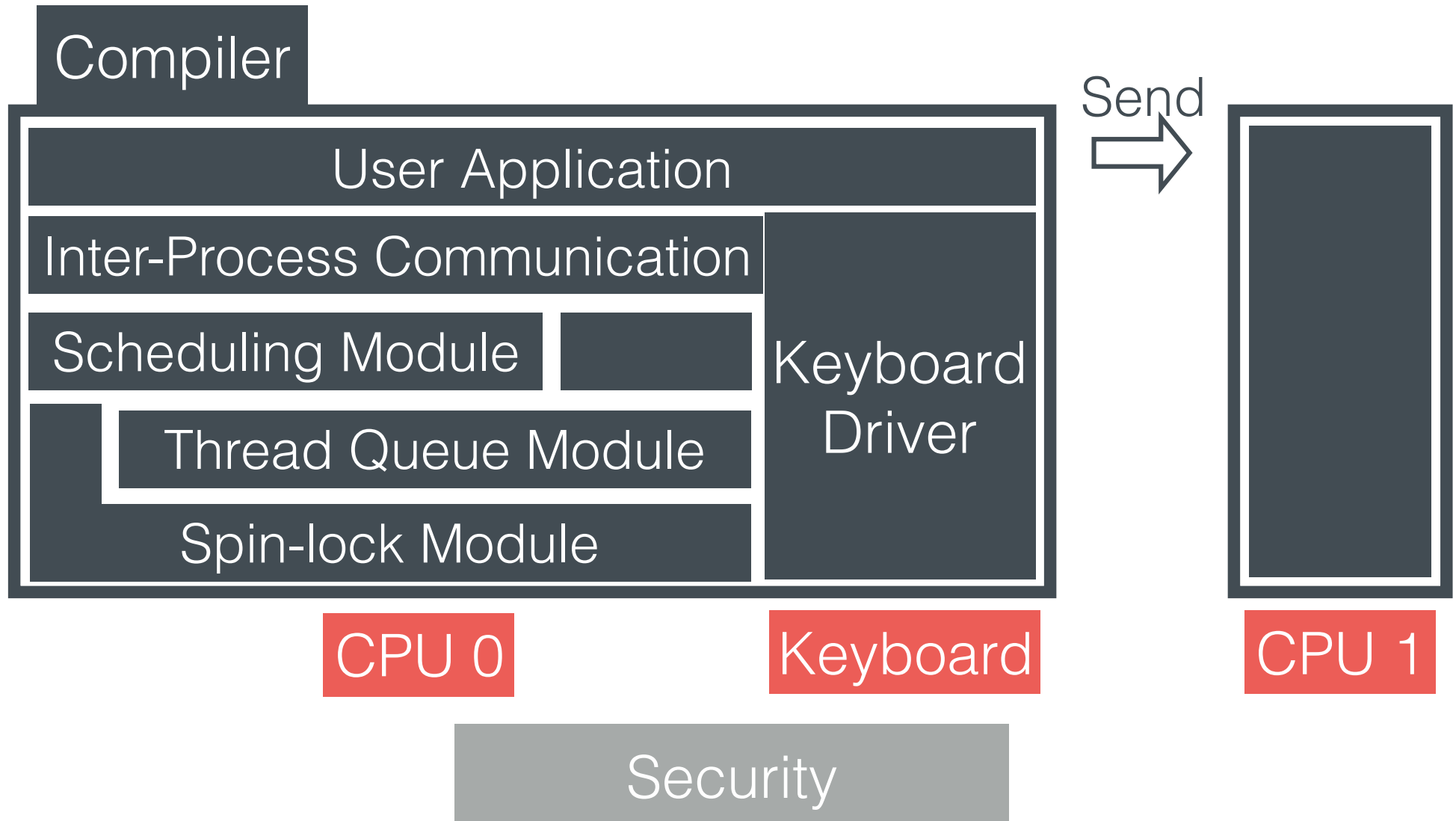


Device Driver [PLDI16'a]



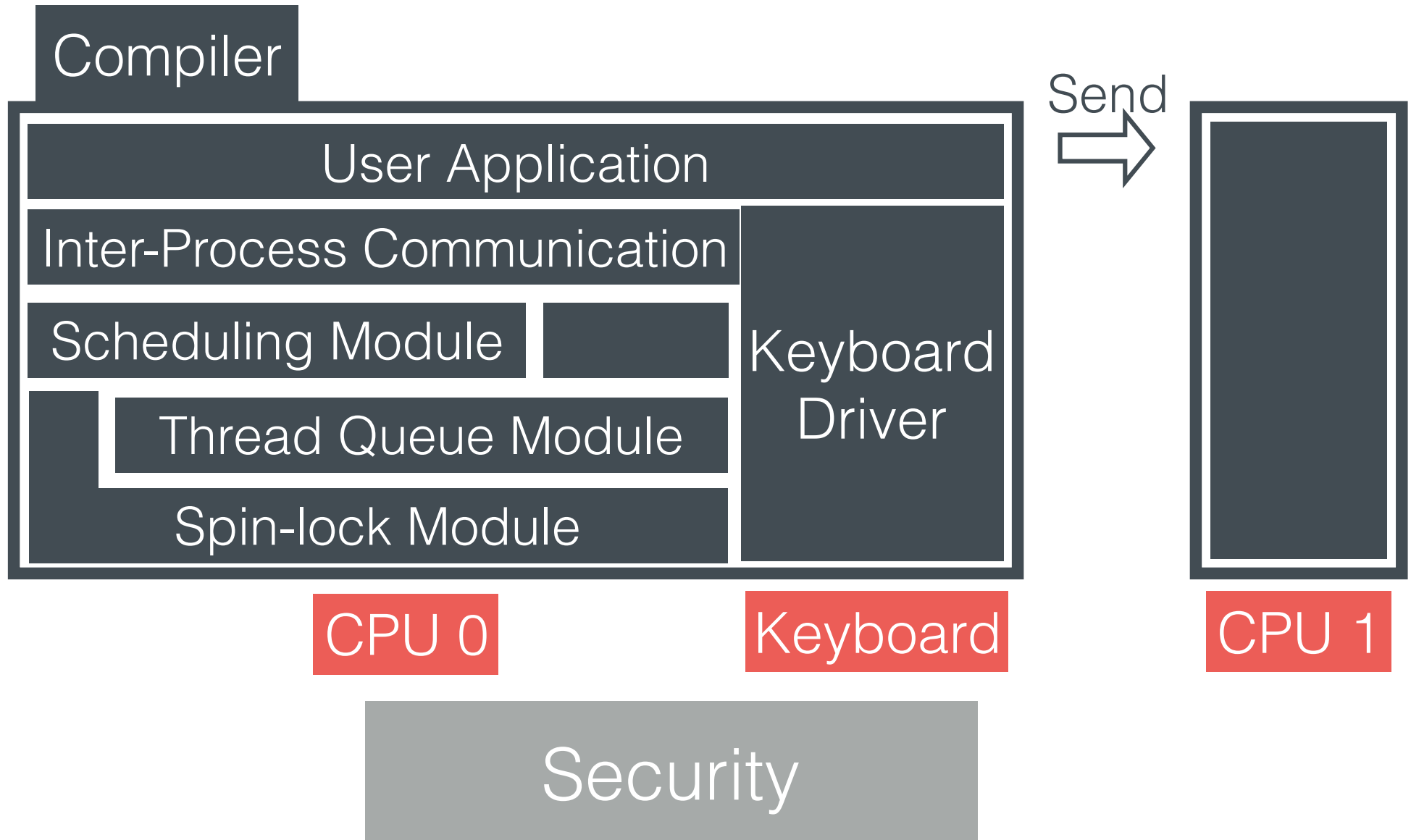
Case Study

Build a Certified System

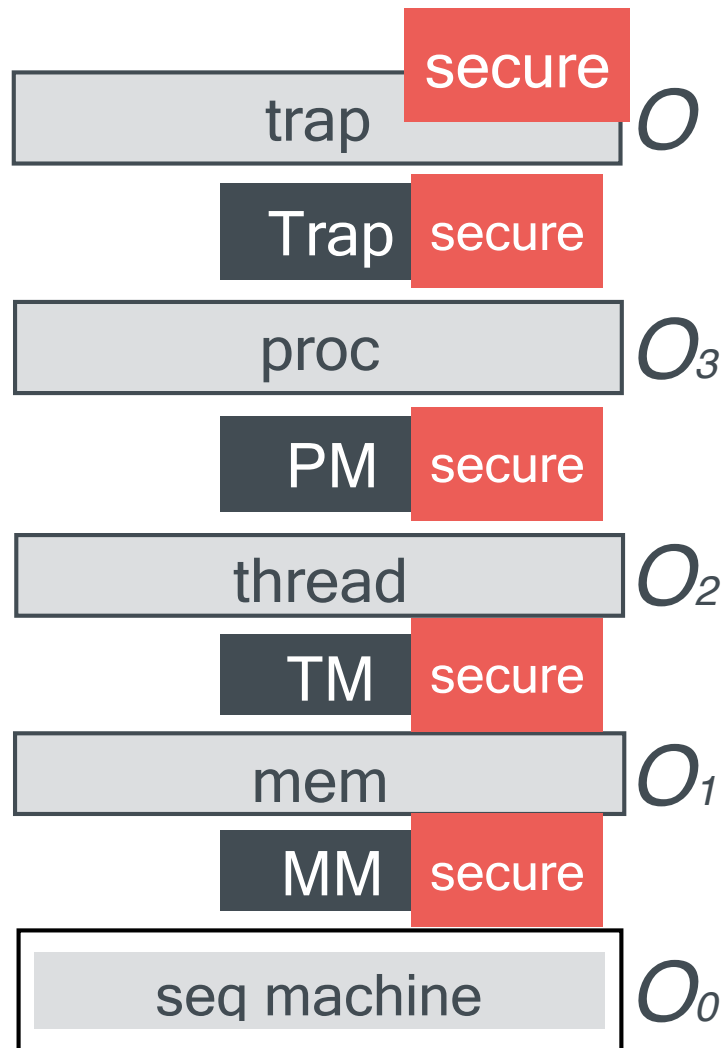


Case Study

Build a Certified System



End-to-End Security [PLDI16'b]

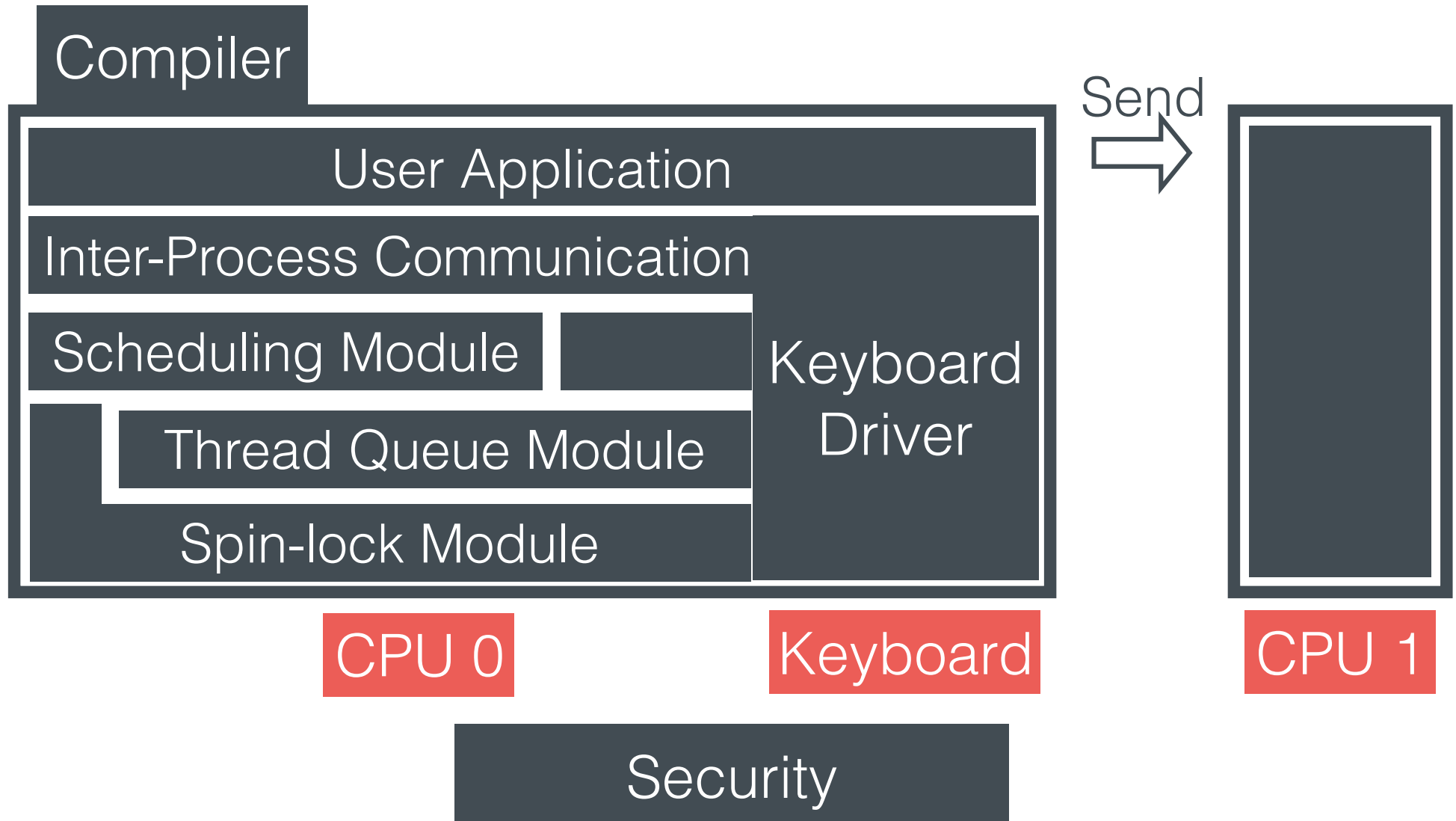


Observation function O

- specify and prove general security policies with declassification
 - security-preservation simulation
 - non-interference
- found **security-bugs**:
spawn, palloc,...

Case Study

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
(IoT, 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.