

# CertiKOS: From Hacker-Resistant OS to Certified Heterogeneous Systems

Zhong Shao  
Yale University  
November 14, 2022

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

## The Future of CS: Heterogeneous Systems

- **Hardware Components**
  - Multicore CPU, MMU/IOMMU, Cache, FPGA, GPU, TPU, ASIC, Domain-Specific HW, ...
  - Cyber-physical systems, smart cars, robotics, IoTs, smart cities, blockchains, clouds, ...
- **Human and Social Components**
  - Passengers, pedestrians, insurance companies, lawyers, policy-makers, ...
  - Banks, consumers, currencies, smart contracts, crypto coins, DAO, federal reserve, ...
- **OS Components**
  - Processes, schedulers, containers, device drivers, virtual machines, file systems, sockets, databases, logs, atomic objects, transactions, network stacks, security protocols, ...
- **SW & PL Components**
  - Data structures; objects; transactions; modules; threads; interrupt handlers; exception; communication channels; concurrent & distributed objects; containers; micro-services; ...
  - Language specs, compilers, interpreters, JIT, binary translator; parsers, serializer / pretty printer, ...
- **Challenge: how to establish strong trust & accountability properties?**
  - Safety, security (isolation), resource efficiency, availability, accountability, extensibility, .....

## PL Research: The Charm

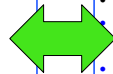
- Uncover the **essence** (i.e., **semantic abstraction**) of various (SW & HW) systems, or **systems of systems**
- Use these new theories to **bridge** different areas of specialty (e.g., via new lang., compilers, tools) and then **build** better & more secure/reliable systems

### Rich Logical Foundations:

- category theory
- universal algebra & co-algebra
- inductive vs. coinductive types
- classical vs. intuitionistic vs. linear logics
- monad vs. comonad
- equality (HTT) vs simulation
- Curry-Howard correspondence
- lambda & process calculi
- denotational vs operational semantics

### New PLs, DSLs, and Compilers:

- static vs dynamic typing vs. "no typing"
- explicit resource manager vs. GC
- effects vs. encapsulation
- OO vs. function programming
- *New multiparadigm languages:* Java Scala, Swift, Rust, Go, C++, C#
- **Concurrent** vs. **parallel** vs **distributed** programming
- proof assistant languages
- information flow control & security
- certified software & compilers



## PL Meets OS: An Ideal Marriage Yet to Happen?

- PL is to uncover the laws of abstraction in the cyber world
- PL is to use abstraction to reduce complexities
- PL depends on the underlying OS for sys lib.
- Many PL issues are easily resolved in OS
- OS is to build layers of abstraction (i.e., VMs) for the cyber world
- OS is full of complexities
- OS is to manage, multiplex, and virtualize resources
- OS really needs PL help to provide safety and security guarantee

## The CertiKOS / DeepSpec Project

**Killer-app:** high-assurance “heterogeneous” systems of systems!

**Conjecture:** today’s PLs fail because they ignored OS, and today’s OSes fail because they get little help from PLs

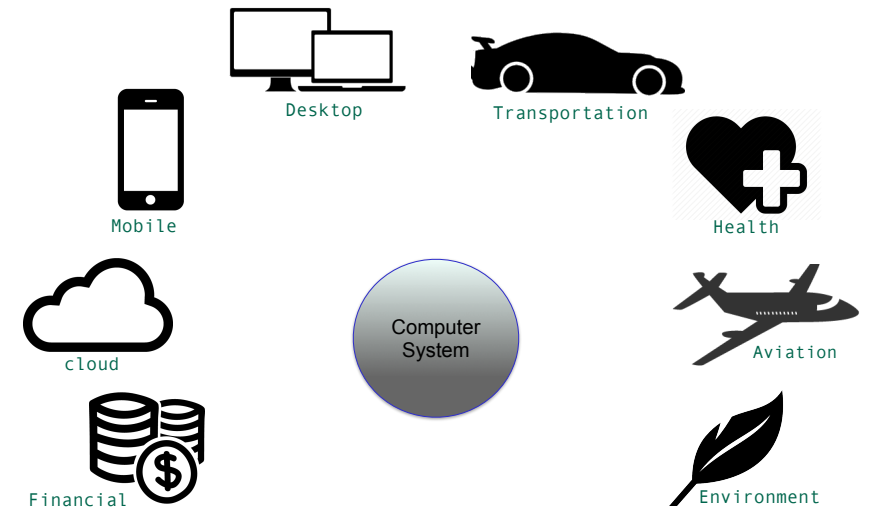
### New Insights:

- deepspec & certified abstraction layers;
- a unifying framework for composing heterogeneous components ( via game semantics + linear logic connectives)

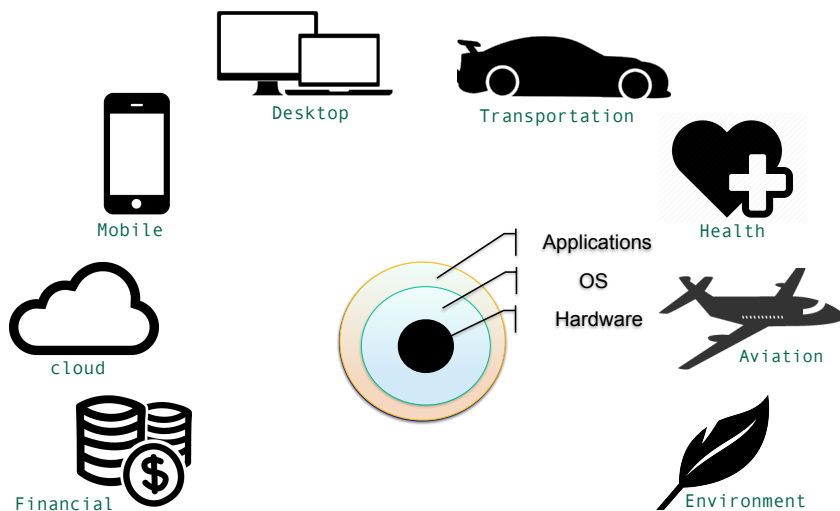
### Opportunities:

- New certified system software stacks (CertiKOS ++)
- New certifying programming languages (DeepSEA vs. C & Asm)
- New certified programming tools
- New certified modeling & arch. description lang. (DeepSEA)
- We verify all interesting properties (correctness, safety, security, availability, ...)

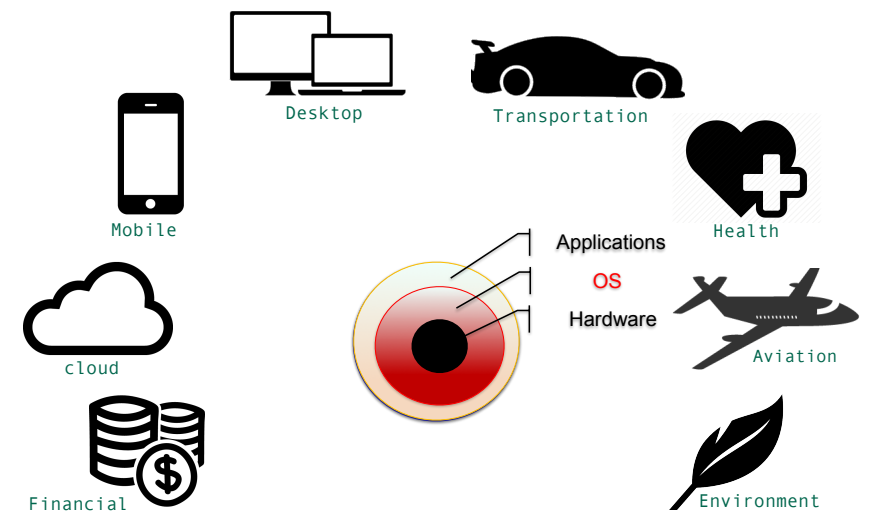
## Hacker-Resistant OS: Why?



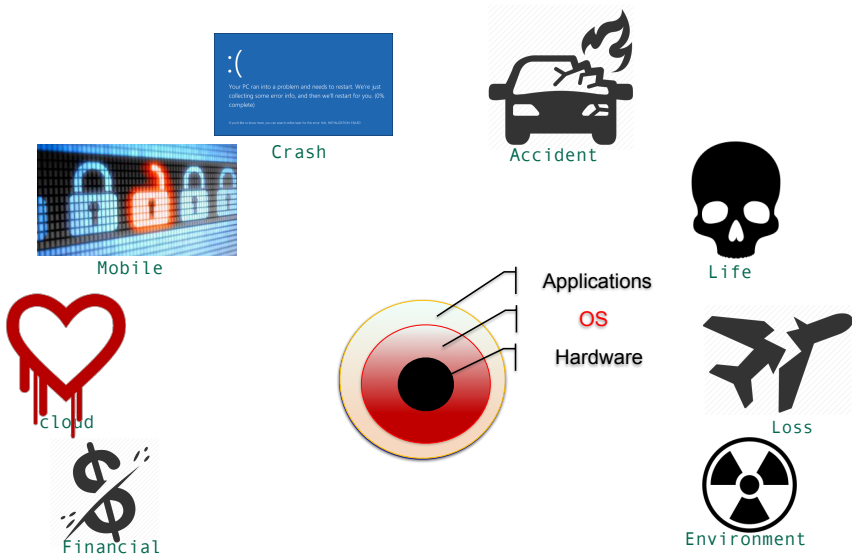
## Hacker-Resistant OS: Why?



## Hacker-Resistant OS: Why?



## Hacker-Resistant OS: Why?



## CertiKOS Problem Definition



- **What is a certified OS?**
  - an OS binary **implements** its specification?
  - what should its specification be like?
- **What properties do we want to prove?**
  - safety & partial correctness properties
  - total **functional correctness**
  - **security properties** (isolation, confidentiality, integrity, availability)
  - **resource usage properties** (stack overflow, real time properties)
  - race-freedom, **atomicity**, and linearizability
  - **liveness properties** (deadlock-freedom, starvation freedom)
- **How to cut down the cost of verification?**

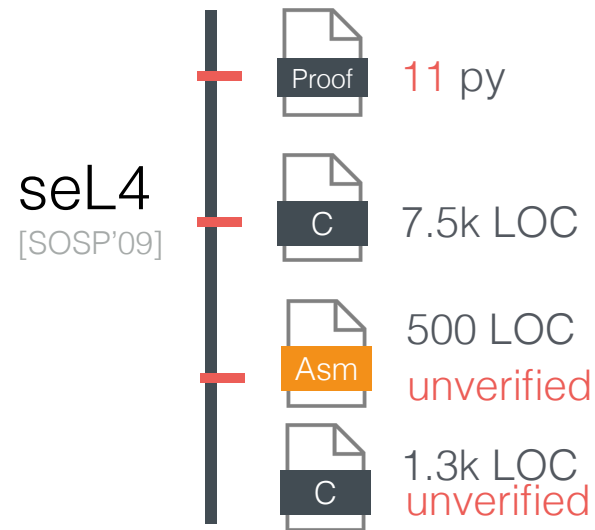
## Motivation

### Formal

“Verification is the **only** way to rule out entire **classes** of attacks”  
 —NSF SFM Report[2016]

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

## Challenges: huge proof efforts



Challenges: Compositionality



Abstraction Gap



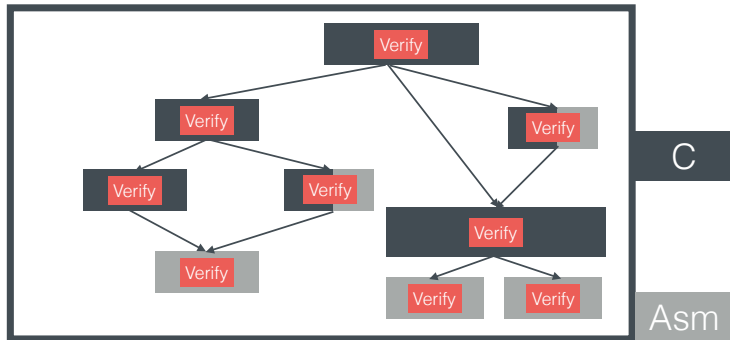
Challenges: Compositionality

A Complex System



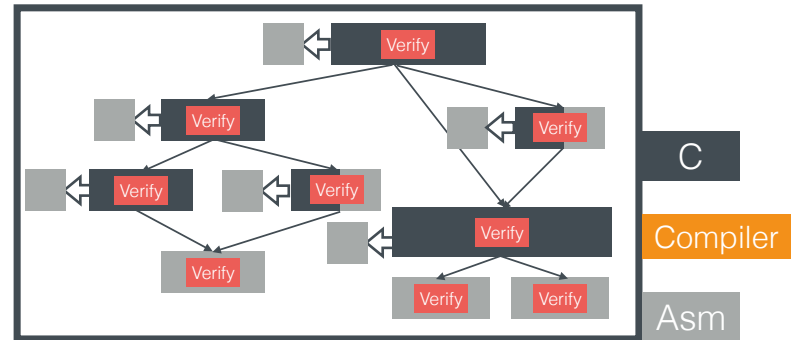
Challenges: Compositionality

A Complex System

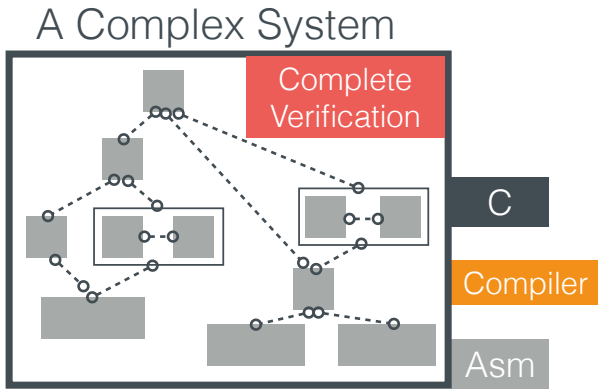


Challenges: Compositionality

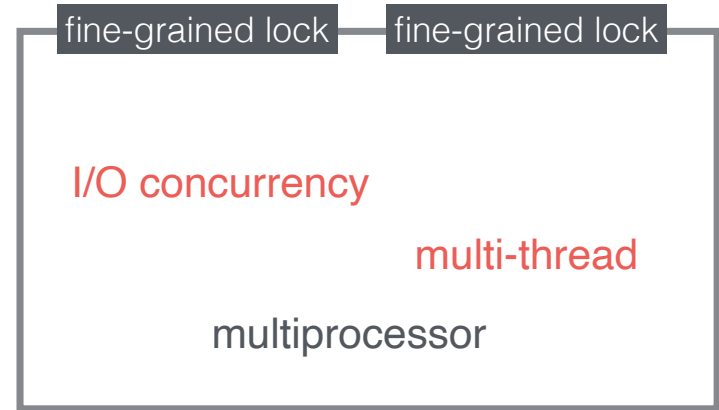
A Complex System



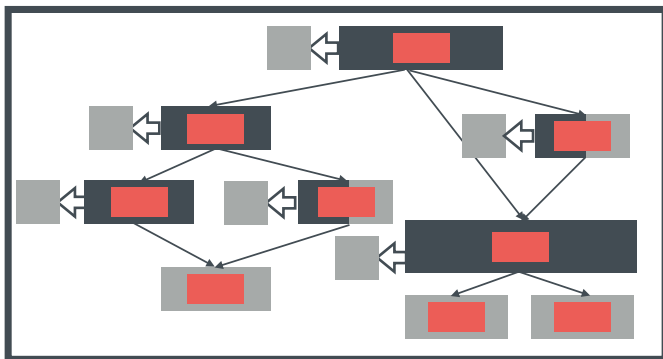
Challenges: Compositionality



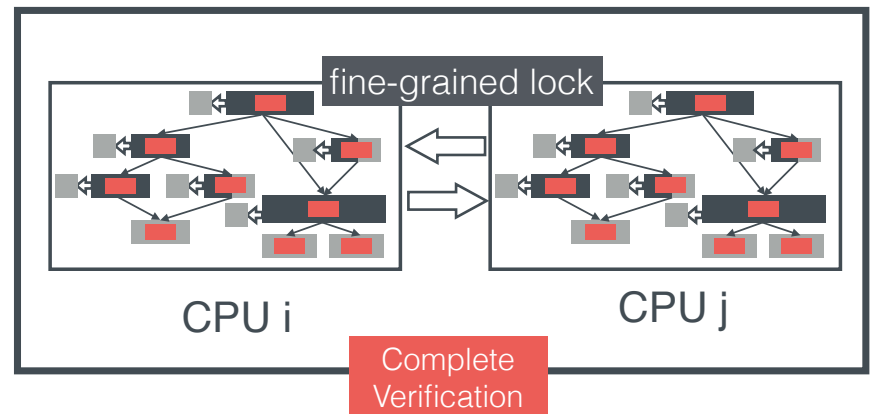
Challenges: Concurrency



Challenges: Concurrency



Challenges: Concurrency



Contribution

Certified Abstraction Layers

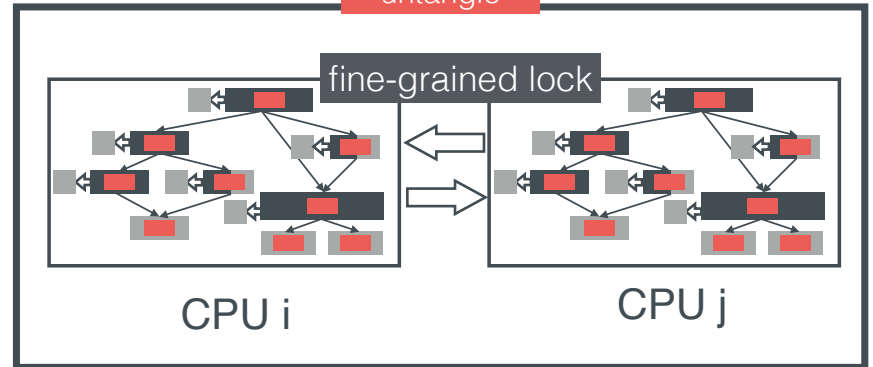
CertiKOS

aim to solve all these challenges

Contribution

Certified Abstraction Layers

untangle



Contribution

Certified Abstraction Layers

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

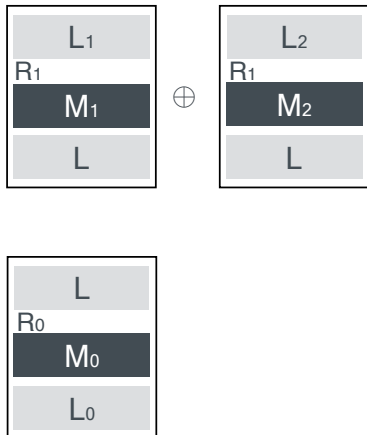
Contribution

Certified Abstraction Layers

- verify existing systems
- build **certified** heterogeneous systems

Contribution

### Certified Abstraction Layers



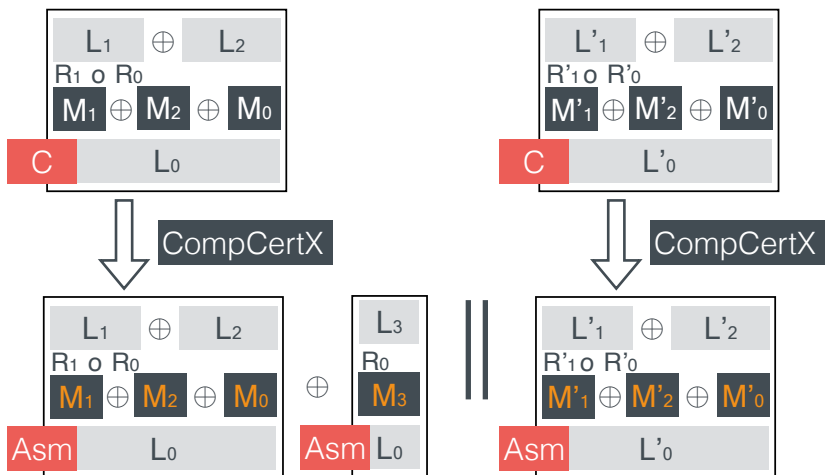
Contribution

### Certified Abstraction Layers



Contribution

### Certified Abstraction Layers



Contribution

### Certified Abstraction Layers

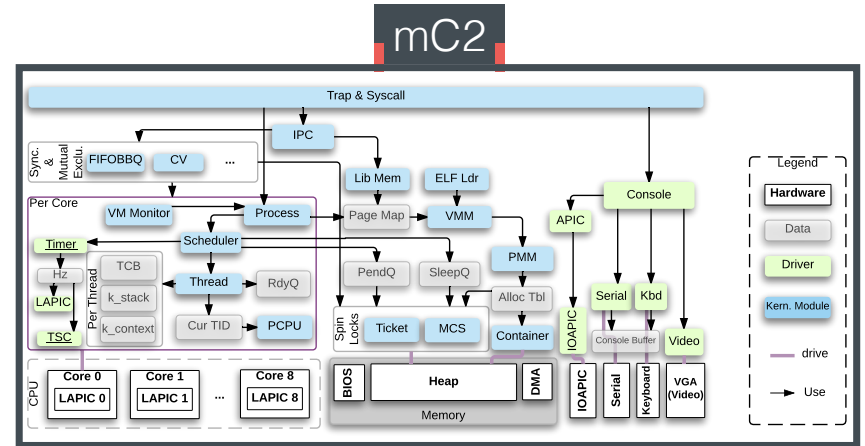
- **mCertikOS** [POPL'15]  
certified sequential OS kernels  
3k C&Asm, 1 py
- **Interrupt** [PLDI'16a] 0.5 py
- **Security** [PLDI'16b] 0.5 py
- **mC2** [OSDI'16] [PLDI'18]  
the **first** formally certified **concurrent**  
OS kernel with fine-grained locks  
6.5k C&Asm, 2 py

## Contribution

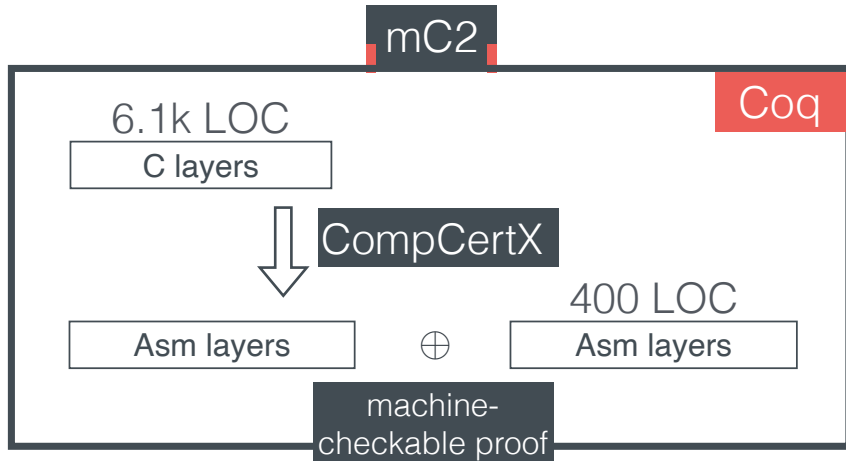
Certified System Software

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

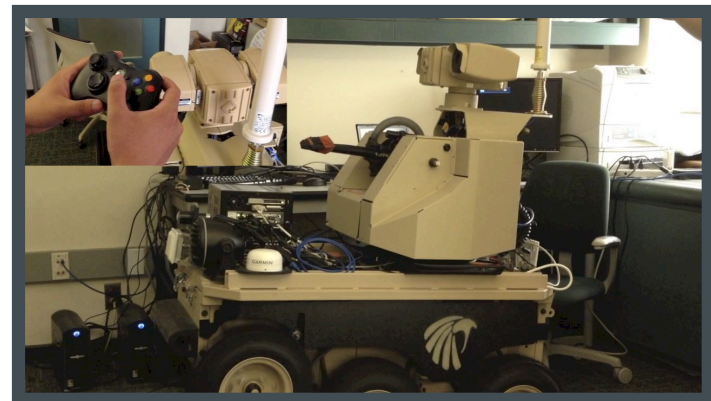
## Contribution



## Contribution



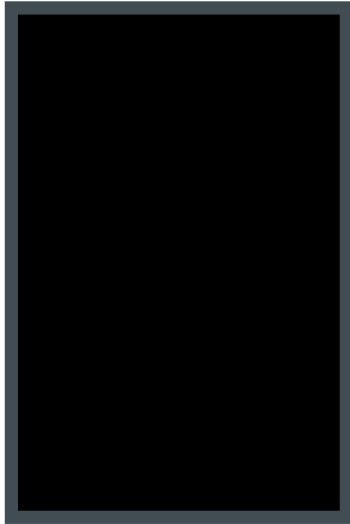
## Deployment



CertiKOS on Landshark, DARPA HACMS



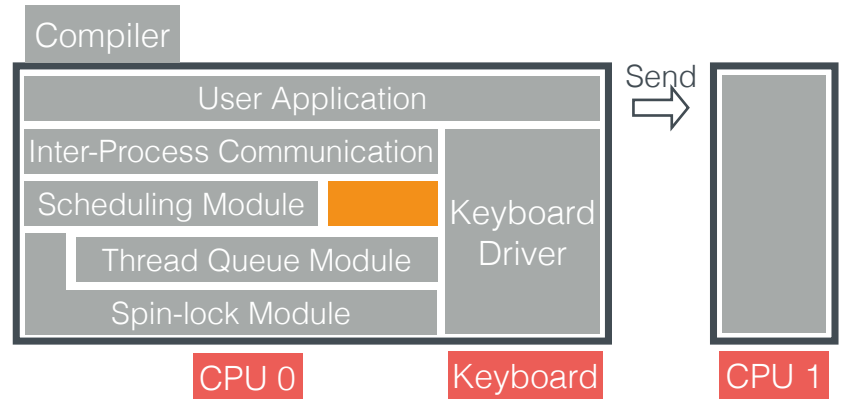
## Deployment



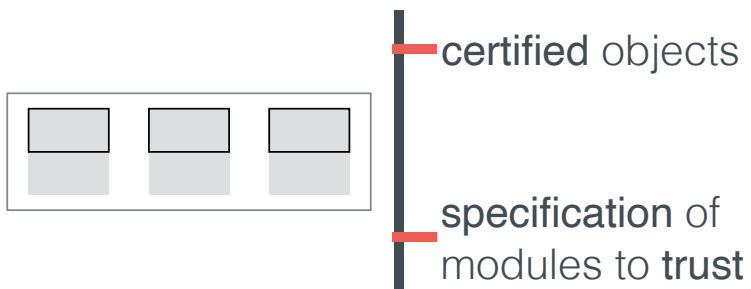
CertiKOS on  
Quadcopter

## Case Study

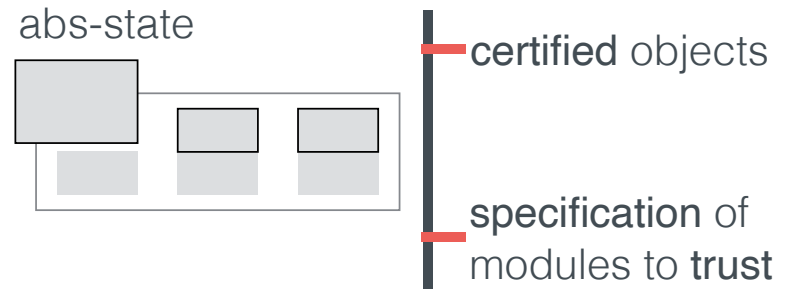
### Build a Certified System



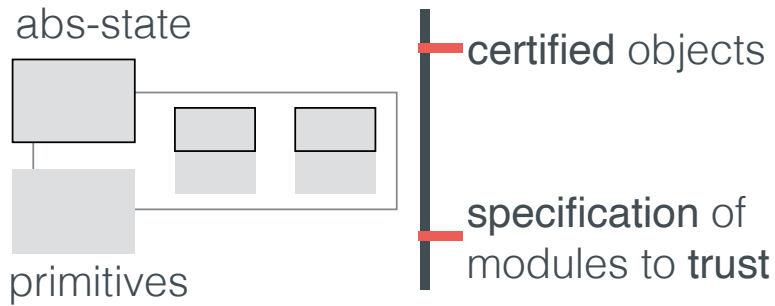
## Certified Sequential Layer [POPL'15]



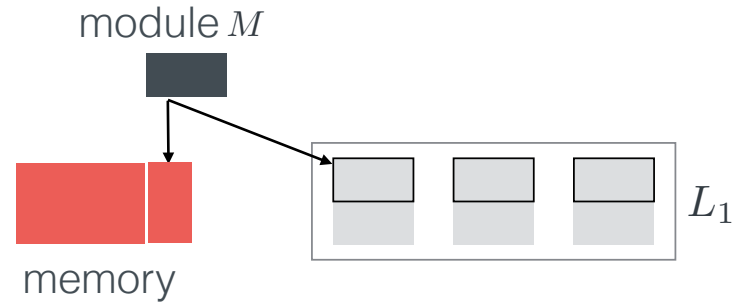
## Certified Sequential Layer [POPL'15]



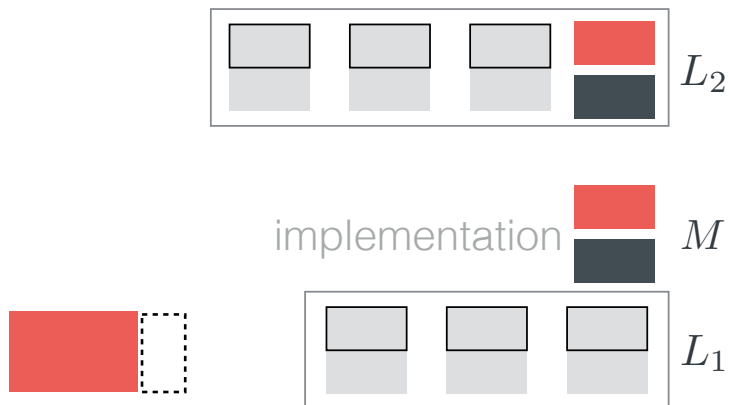
### Certified Sequential Layer [POPL'15]



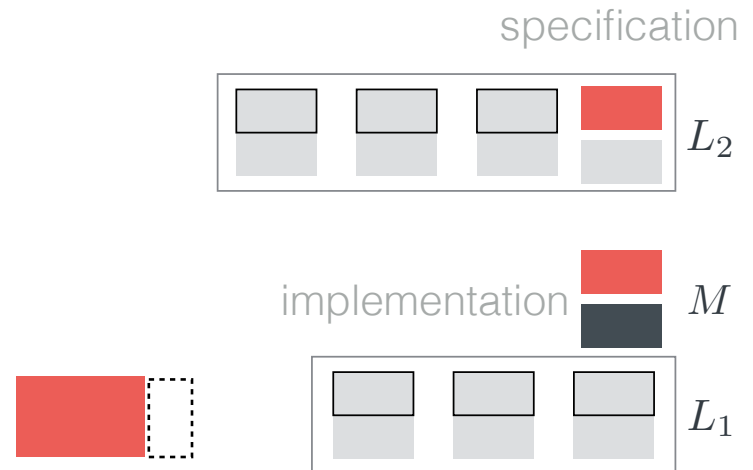
### Certified Sequential Layer



### Certified Sequential Layer

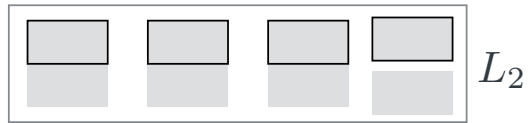


### Certified Sequential Layer

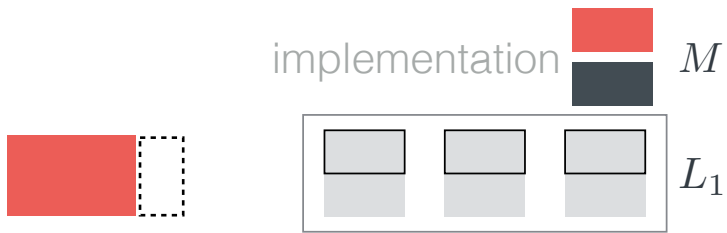


## Certified Sequential Layer

specification

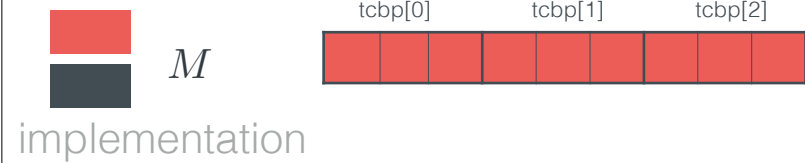


implementation  $M$



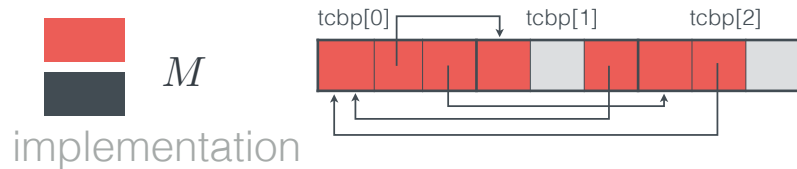
## Example: Thread Queue

<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; C</pre>
--	--



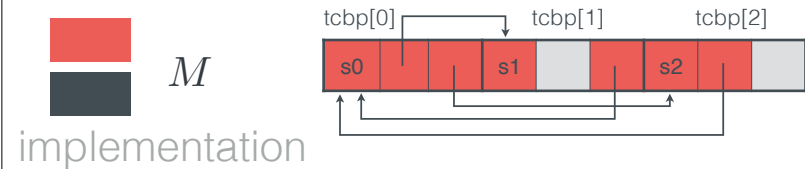
## Example: Thread Queue

<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; C</pre>
--	--



## 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; C</pre>
--	--



### Example: Thread Queue

```

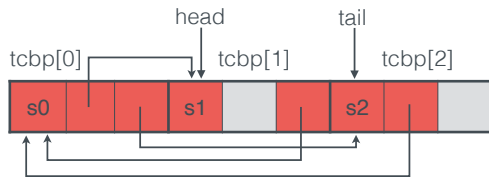
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];

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

tdq* td_queue;
    
```

 *M*  
implementation



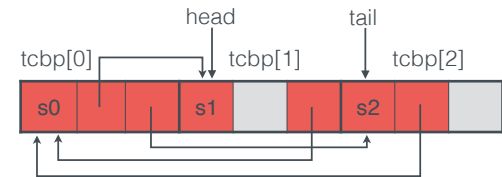
### Example: Thread Queue

```

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;
}
    
```

 *M*  
implementation



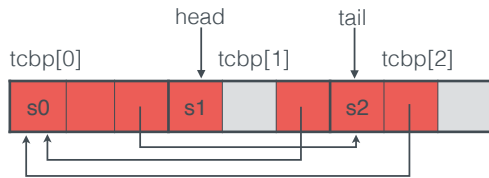
### Example: Thread Queue

```

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;
}
    
```

 *M*  
implementation



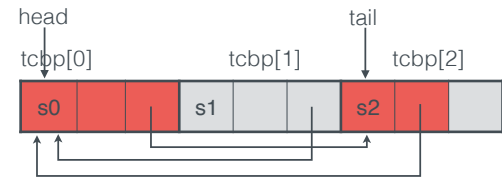
### Example: Thread Queue

```

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;
}
    
```

 *M*  
implementation



### Example: Thread Queue

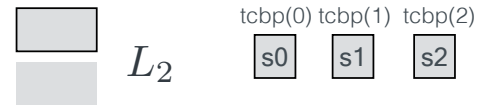
specification



Definition **tcbp** := ZMap.t state.  
Definition **td\_queue** := List Z. Coq

### Example: Thread Queue

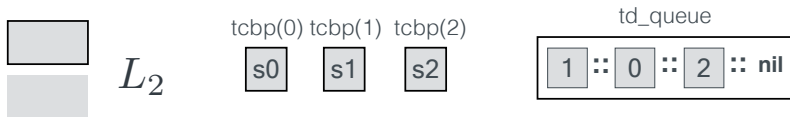
specification



Definition **tcbp** := ZMap.t state.  
Definition **td\_queue** := List Z. Coq

### Example: Thread Queue

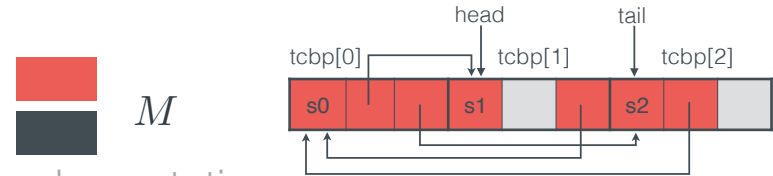
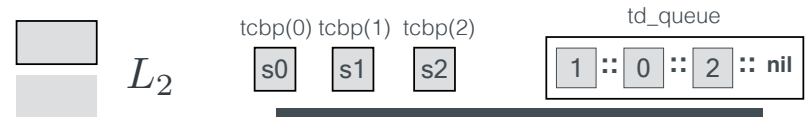
specification



Definition **tcbp** := ZMap.t state.  
Definition **td\_queue** := List Z. Coq

### Example: Thread Queue

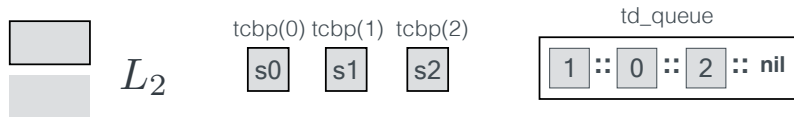
specification



implementation

## Example: Thread Queue

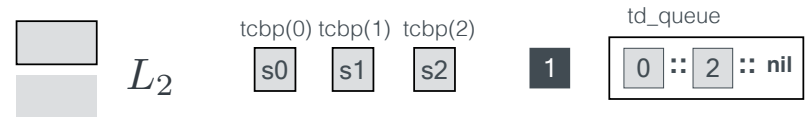
specification



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

## Example: Thread Queue

specification

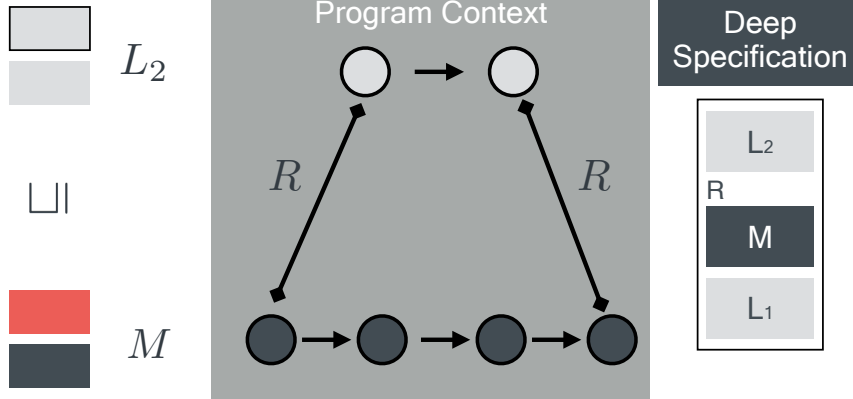


```
Function dequeue (q) :=
  match q with
  | head :: q' => (q', Some head)
  | nil => (nil, None)
end. Coq
```

executable

## Simulation Proof

specification

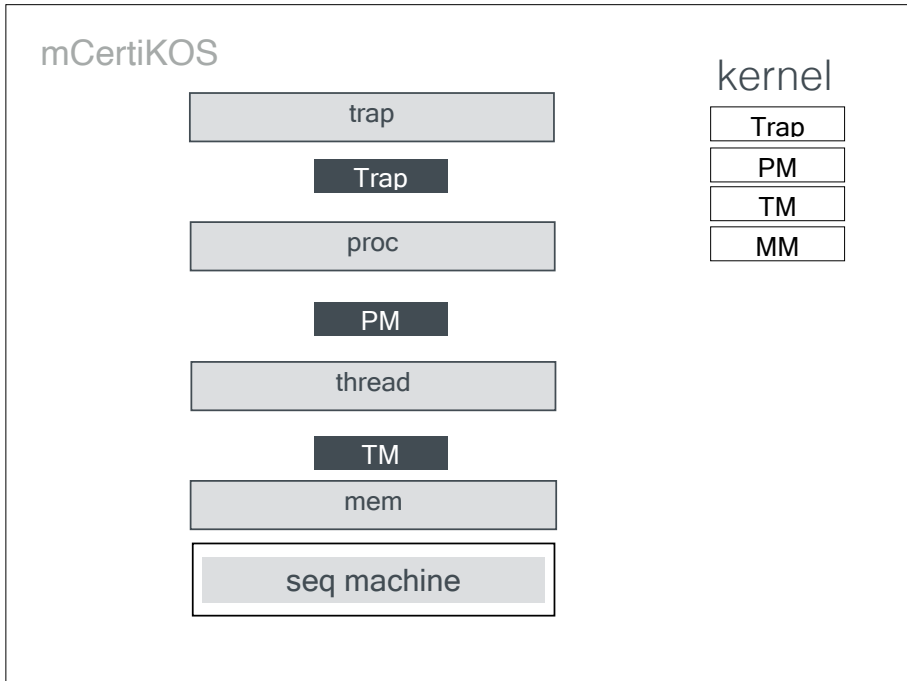
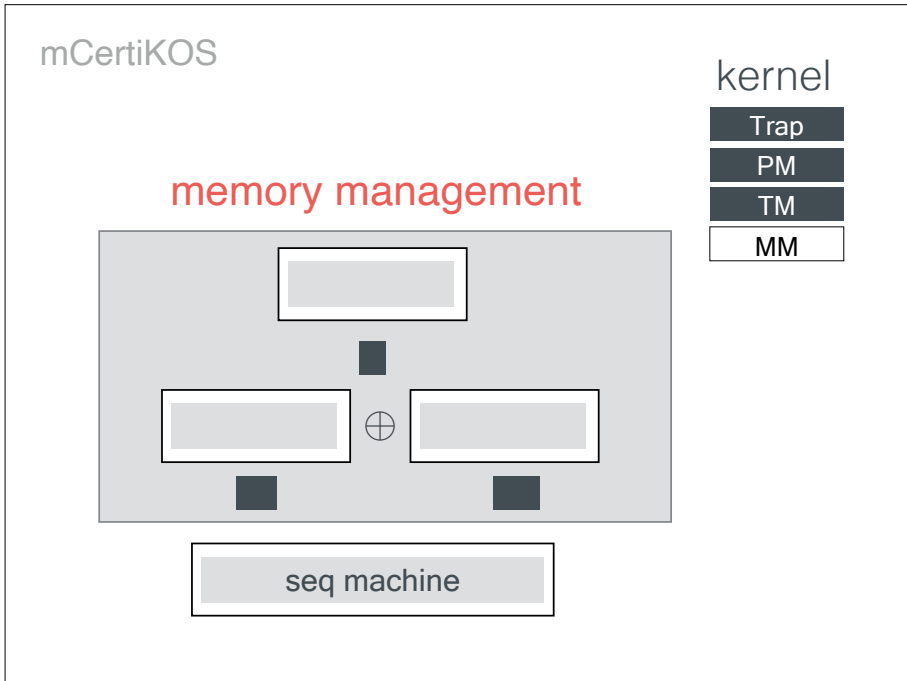
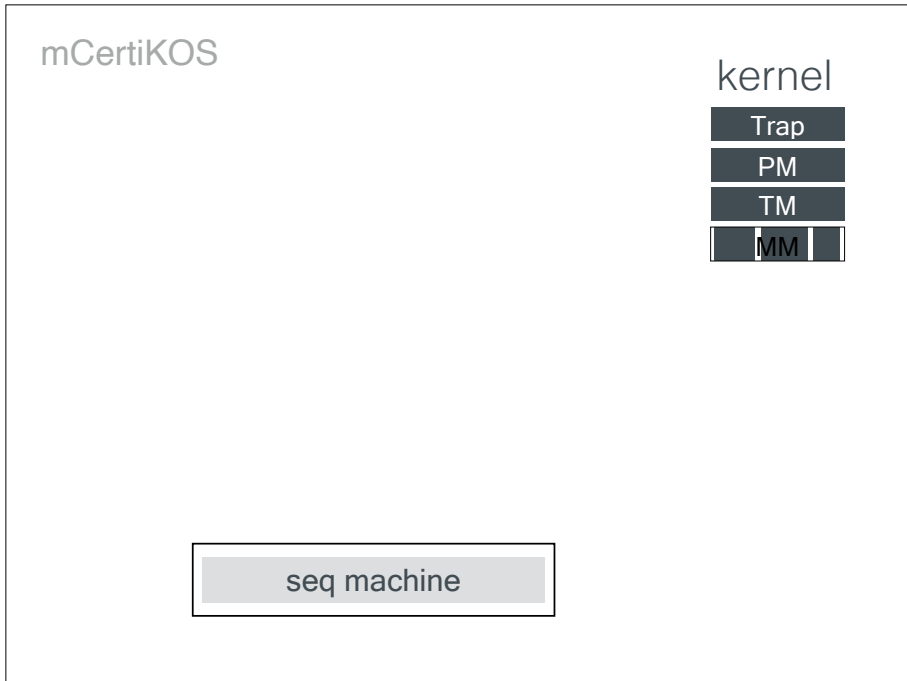


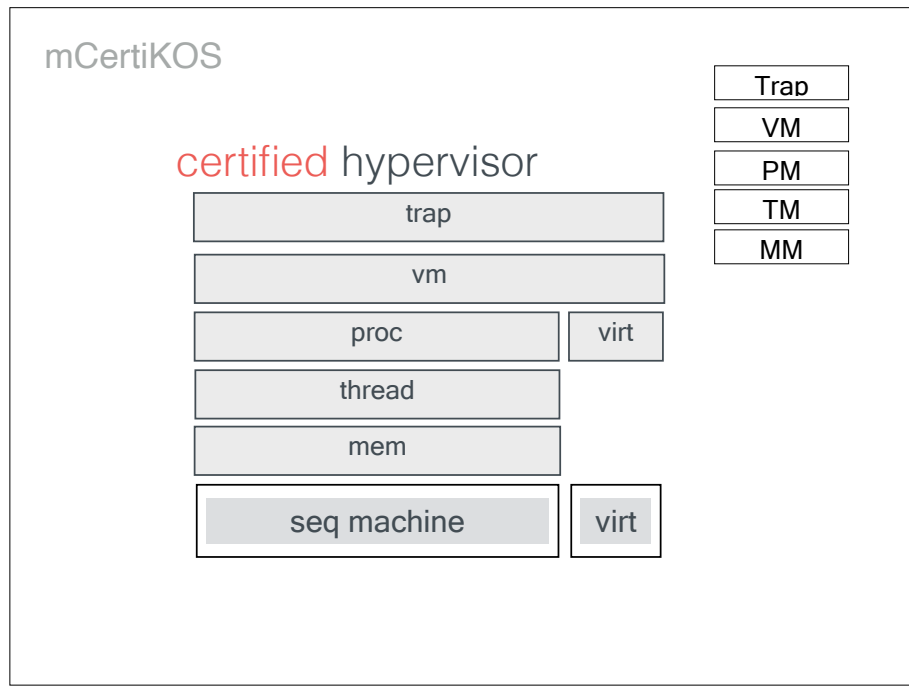
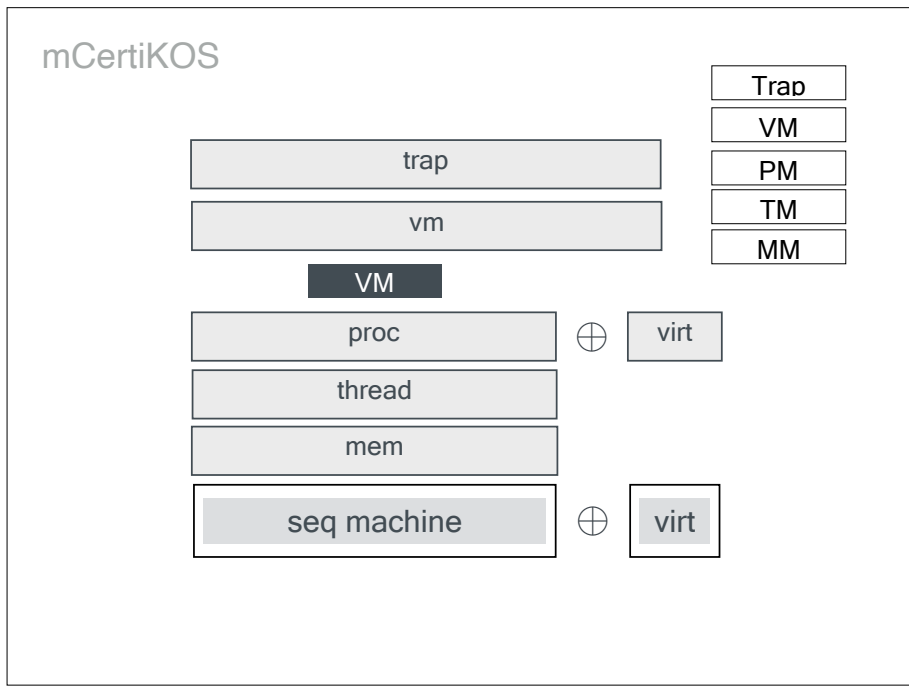
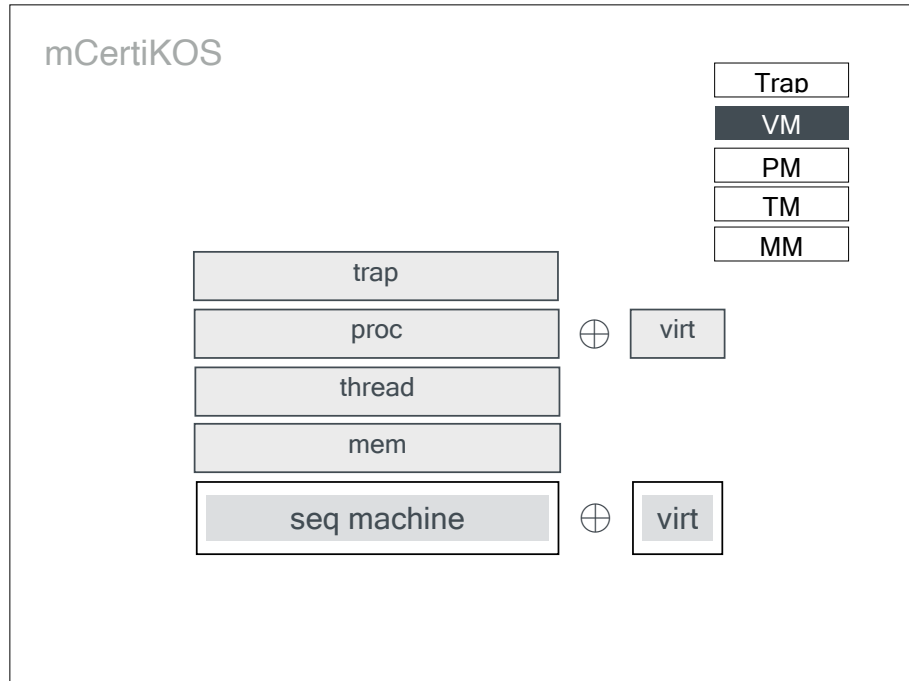
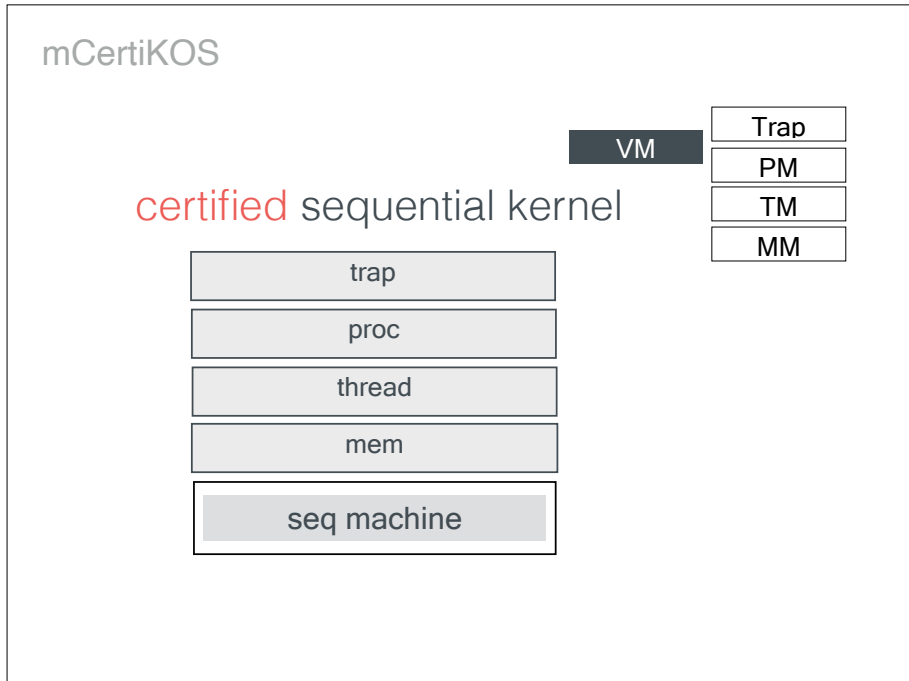
implementation

## 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** 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	malloc/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



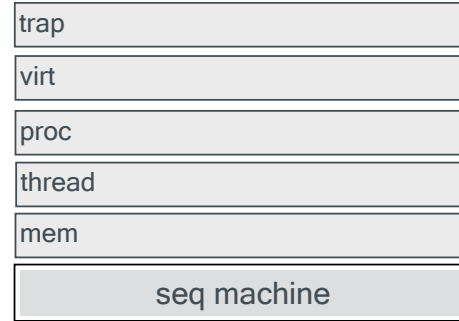
**TSysCall Layer**

(mm/proc/virt.abs) vmcbinit sys\_run/vm\_exit PageFault\_Handler sys\_yield sys\_check/exit/sync/inject/set\_chan(17) mm/proc\_prim

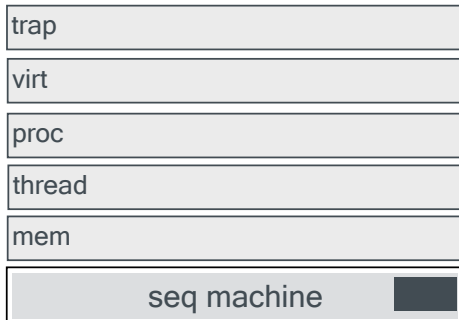
**TTrap Layer**  
 (mm/proc/virt.abs) vmcbinit vm\_run/vm\_exit get\_arg/set\_ret sys\_check/exit/sync/inject/set\_chan(17) mm/proc\_prim

Concurrent Framework [OSDI'16, PLDI'18]

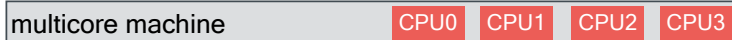
certified sequential kernel



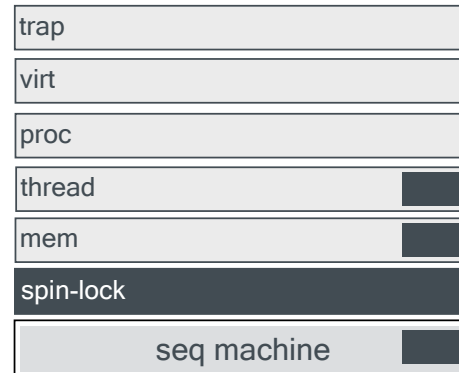
Concurrent Framework [OSDI'16, PLDI'18]



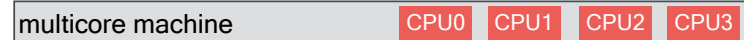
**contribution**  
 machine lifting  
 certified concurrent layer



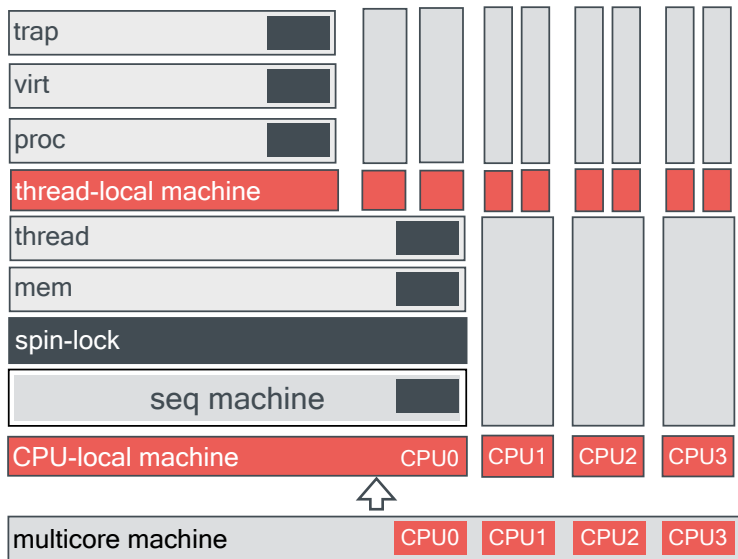
Concurrent Framework [OSDI'16, PLDI'18]



**contribution**  
 machine lifting  
 certified concurrent layer

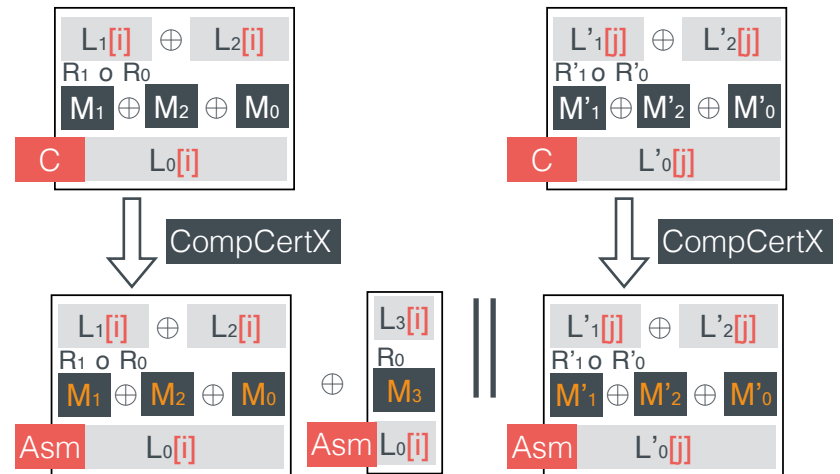


## Concurrent Framework [OSDI'16, PLDI'18]

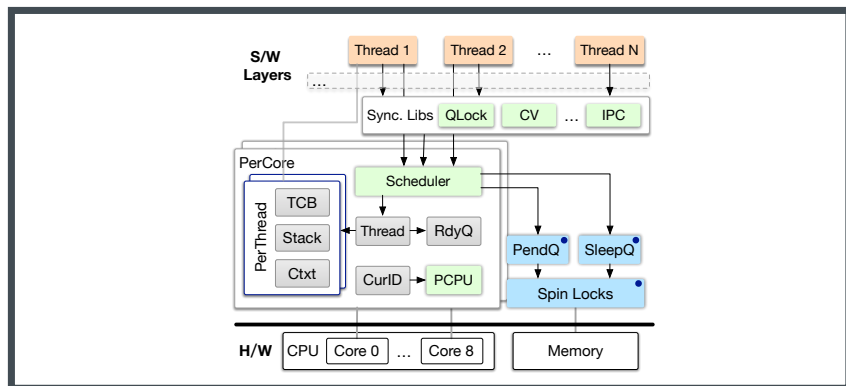


## Contribution

### Certified Concurrent Abstraction Layers



## Contribution



## Case Study

```

struct ticket_lock {
    volatile uint n, t;
};
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
    uint my_t = FAI_t();
    while(get_n()!=my_t){};
    hold();
}
void rel () {
    inc_n();
}

```

```

//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
//M2 module
int x = 0; //shared variable x
void update_x () {
    acq(); x += cpu_id(); rel();
}
//Methods provided by L2
extern void update_x();
//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }

```

C

## Case Study

```

struct ticket_lock {
  volatile uint n, t;
};
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {
  inc_n();
}
//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
//M2 module
int x = 0; //shared variable x
void update_x () {
  acq(); x += cpu_id(); rel();
}
//Methods provided by L2
extern void update_x();
//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }

```

C

## Case Study

```

struct ticket_lock {
  volatile uint n, t;
};
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {
  inc_n();
}
//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
//M2 module
int x = 0; //shared variable x
void update_x () {
  acq(); x += cpu_id(); rel();
}
//Methods provided by L2
extern void update_x();
//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }

```

C

## Strategies and Game Semantics

strategy  $\psi_p[i]$

How will the program  $p$  generate **events** on behalf of CPU  $i$  at each step regarding the given **logical log l** ?

## Strategies and Game Semantics

$\psi_{FAI\_t}[1]$

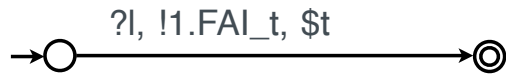


logical  
log l

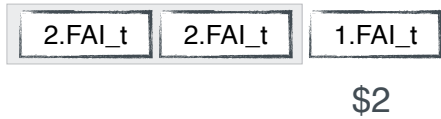
2.FAI\_t 2.FAI\_t

### Strategies and Game Semantics

$$\psi_{FAI\_t}[1]$$



logical  
log I

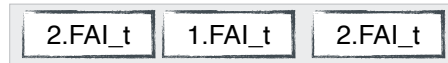


### Strategies and Game Semantics

$$\psi_{FAI\_t}[1]$$



logical  
log I

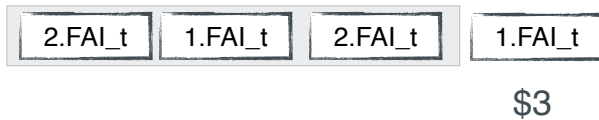


### Strategies and Game Semantics

$$\psi_{FAI\_t}[1]$$



logical  
log I



### Strategies and Game Semantics

```

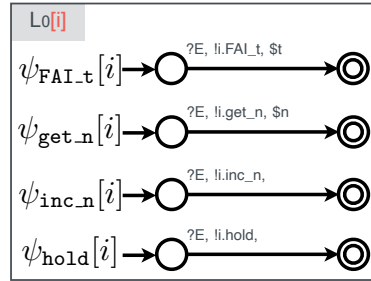
Lo[1]
extern uint FAI_t();

extern uint get_n();

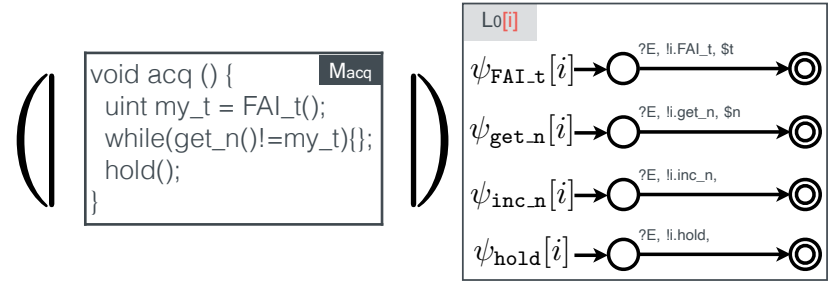
extern void inc_n();

extern void hold();
    
```

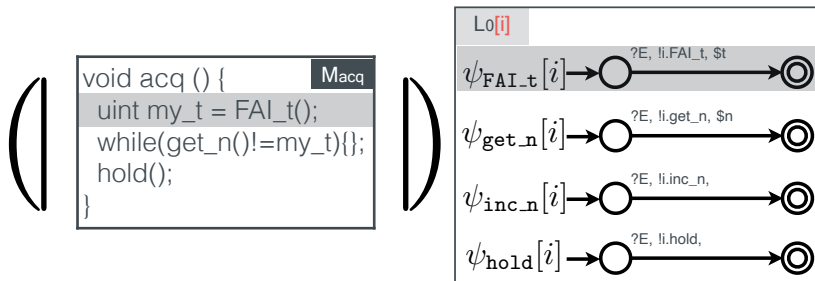
## Strategies and Game Semantics



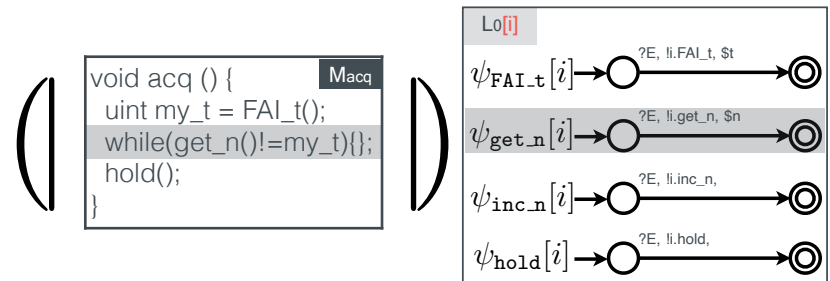
## Strategies and Game Semantics



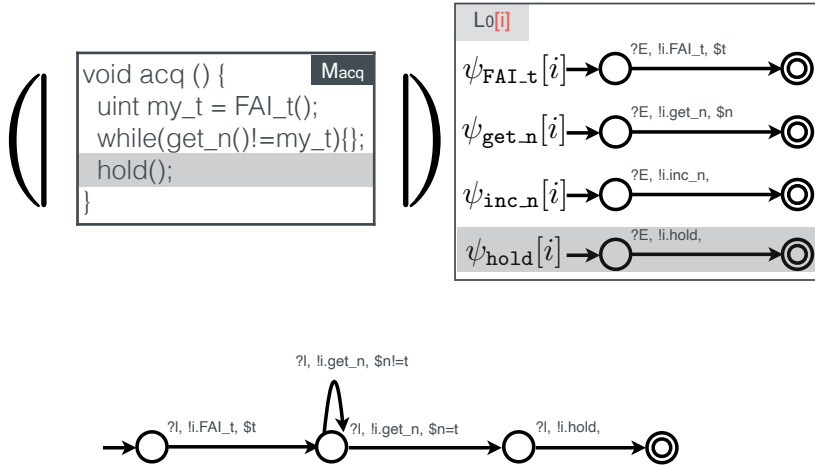
## Strategies and Game Semantics



## Strategies and Game Semantics



## Strategies and Game Semantics

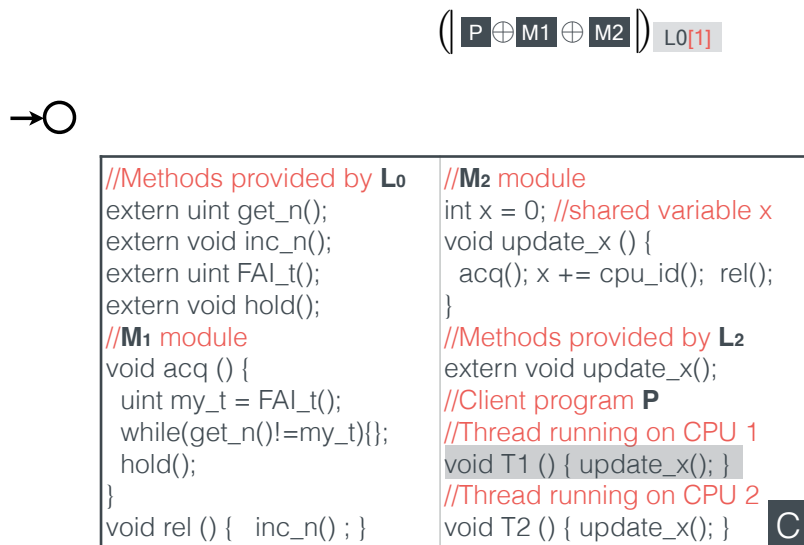


## Strategies and Game Semantics

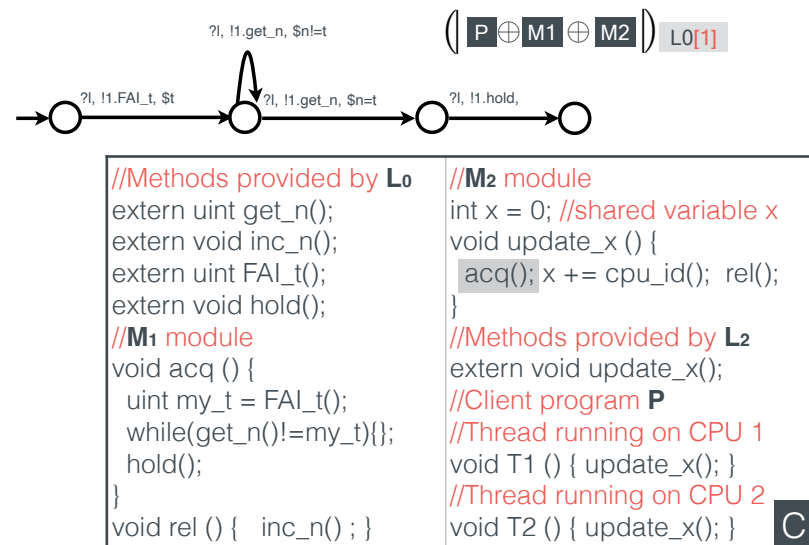
strategy ( **Macq** ) **Lo[i]**

Given the current **log l**, how the module **Macq** running over **Lo[i]** will generate **events** on behalf of CPU **i** at each step.

## Strategies and Game Semantics



## Strategies and Game Semantics



## Strategies and Game Semantics

$(P \oplus M1 \oplus M2) L0[1]$



<pre>//Methods provided by L0 extern uint get_n(); extern void inc_n(); extern uint FAI_t(); extern void hold(); //M1 module void acq () {   uint my_t = FAI_t();   while(get_n()!=my_t){};   hold(); } void rel () { inc_n(); }</pre>	<pre>//M2 module int x = 0; //shared variable x void update_x () {   acq(); x += cpu_id(); rel(); } //Methods provided by L2 extern void update_x(); //Client program P //Thread running on CPU 1 void T1 () { update_x(); } //Thread running on CPU 2 void T2 () { update_x(); }</pre>
--	---

**C**

## Strategies and Game Semantics

$(P \oplus M1 \oplus M2) L0[1]$



?l, !1.FAI\_t, \$t

logical  
log l

2.FAI\_t

2.FAI\_t

## Strategies and Game Semantics

$(P \oplus M1 \oplus M2) L0[1]$



?l, !1.FAI\_t, \$t

logical  
log l

2.FAI\_t

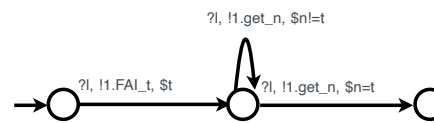
2.FAI\_t

1.FAI\_t

\$2

## Strategies and Game Semantics

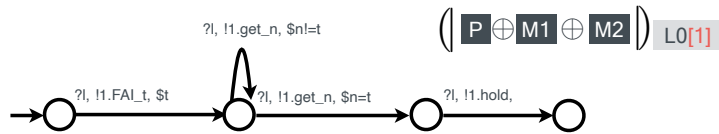
$(P \oplus M1 \oplus M2) L0[1]$



<pre>//Methods provided by L0 extern uint get_n(); extern void inc_n(); extern uint FAI_t(); extern void hold(); //M1 module void acq () {   uint my_t = FAI_t();   while(get_n()!=my_t){};   hold(); } void rel () { inc_n(); }</pre>	<pre>//M2 module int x = 0; //shared variable x void update_x () {   acq(); x += cpu_id(); rel(); } //Methods provided by L2 extern void update_x(); //Client program P //Thread running on CPU 1 void T1 () { update_x(); } //Thread running on CPU 2 void T2 () { update_x(); }</pre>
--	---

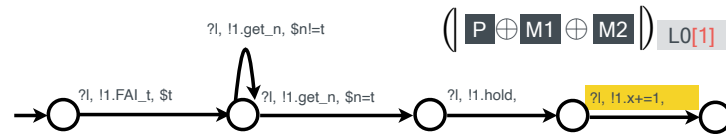
**C**

## Strategies and Game Semantics



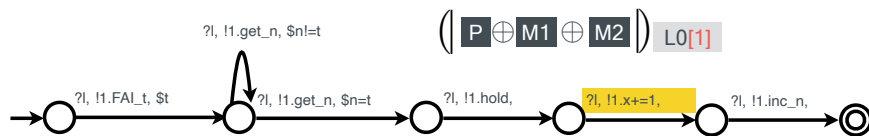
<pre>//Methods provided by L0 extern uint get_n(); extern void inc_n(); extern uint FAI_t(); extern void hold(); //M1 module void acq () {   uint my_t = FAI_t();   while(get_n()!=my_t){}   hold(); } void rel () { inc_n(); }</pre>	<pre>//M2 module int x = 0; //shared variable x void update_x () {   acq(); x += cpu_id(); rel(); } //Methods provided by L2 extern void update_x(); //Client program P //Thread running on CPU 1 void T1 () { update_x(); } //Thread running on CPU 2 void T2 () { update_x(); }</pre>
---	---

## Strategies and Game Semantics



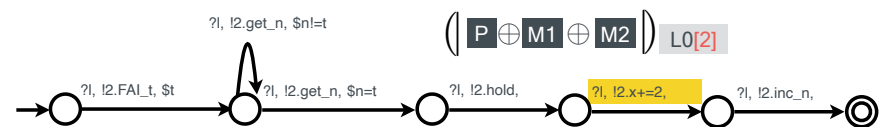
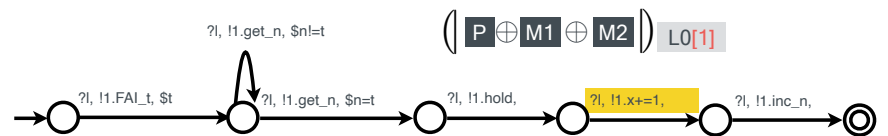
<pre>//Methods provided by L0 extern uint get_n(); extern void inc_n(); extern uint FAI_t(); extern void hold(); //M1 module void acq () {   uint my_t = FAI_t();   while(get_n()!=my_t){}   hold(); } void rel () { inc_n(); }</pre>	<pre>//M2 module int x = 0; //shared variable x void update_x () {   acq(); x += cpu_id(); rel(); } //Methods provided by L2 extern void update_x(); //Client program P //Thread running on CPU 1 void T1 () { update_x(); } //Thread running on CPU 2 void T2 () { update_x(); }</pre>
---	---

## Strategies and Game Semantics



<pre>//Methods provided by L0 extern uint get_n(); extern void inc_n(); extern uint FAI_t(); extern void hold(); //M1 module void acq () {   uint my_t = FAI_t();   while(get_n()!=my_t){}   hold(); } void rel () { inc_n(); }</pre>	<pre>//M2 module int x = 0; //shared variable x void update_x () {   acq(); x += cpu_id(); rel(); } //Methods provided by L2 extern void update_x(); //Client program P //Thread running on CPU 1 void T1 () { update_x(); } //Thread running on CPU 2 void T2 () { update_x(); }</pre>
---	---

## Strategies and Game Semantics

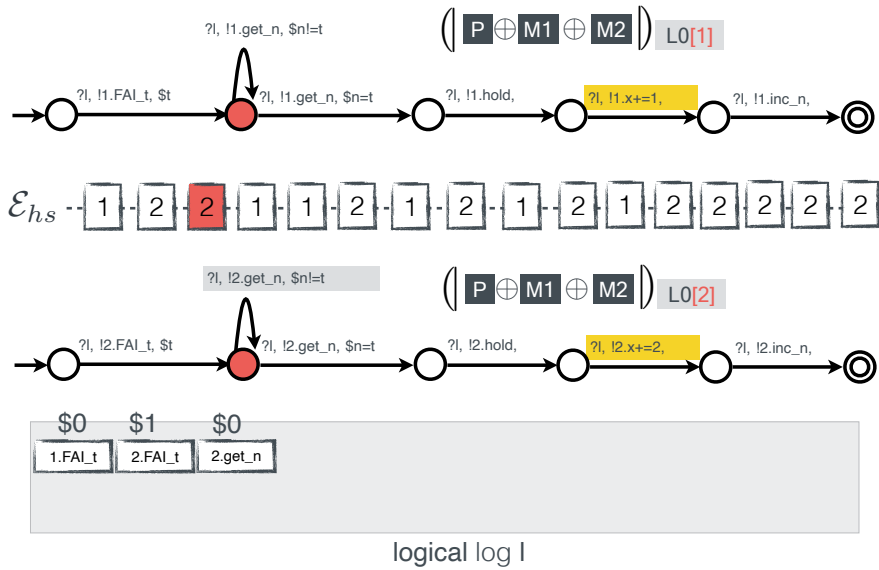


<pre>//Methods provided by L0 extern uint get_n(); extern void inc_n(); extern uint FAI_t(); extern void hold(); //M1 module void acq () {   uint my_t = FAI_t();   while(get_n()!=my_t){}   hold(); } void rel () { inc_n(); }</pre>	<pre>//M2 module int x = 0; //shared variable x void update_x () {   acq(); x += cpu_id(); rel(); } //Methods provided by L2 extern void update_x(); //Client program P //Thread running on CPU 1 void T1 () { update_x(); } //Thread running on CPU 2 void T2 () { update_x(); }</pre>
---	---

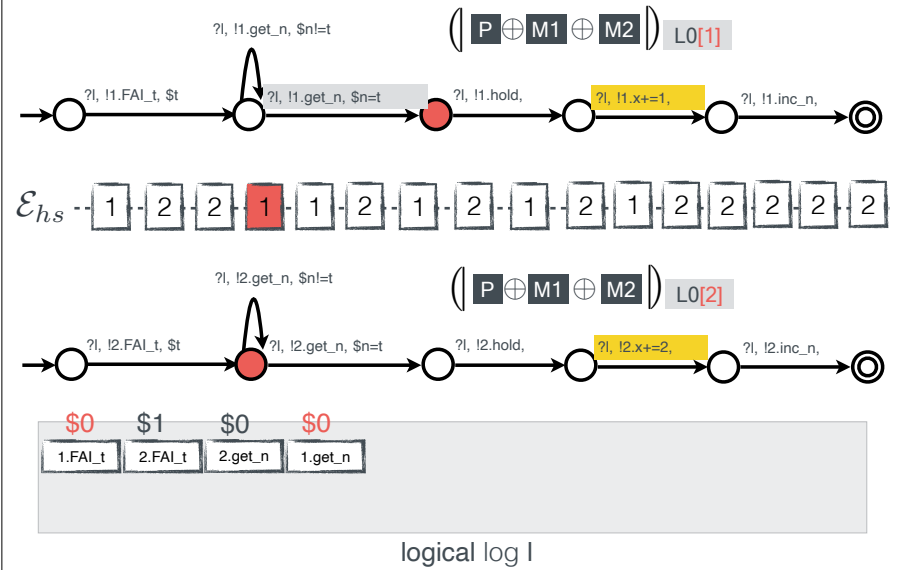




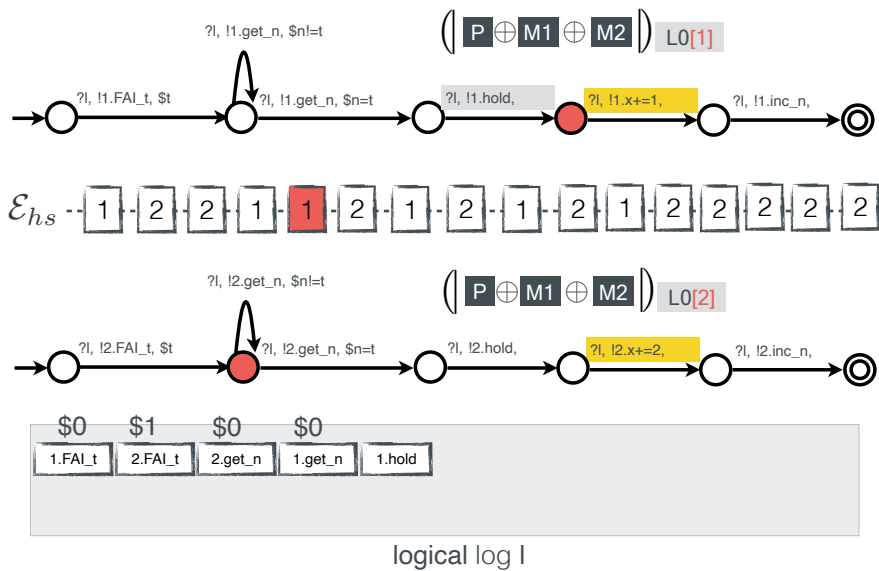
## Strategies and Game Semantics



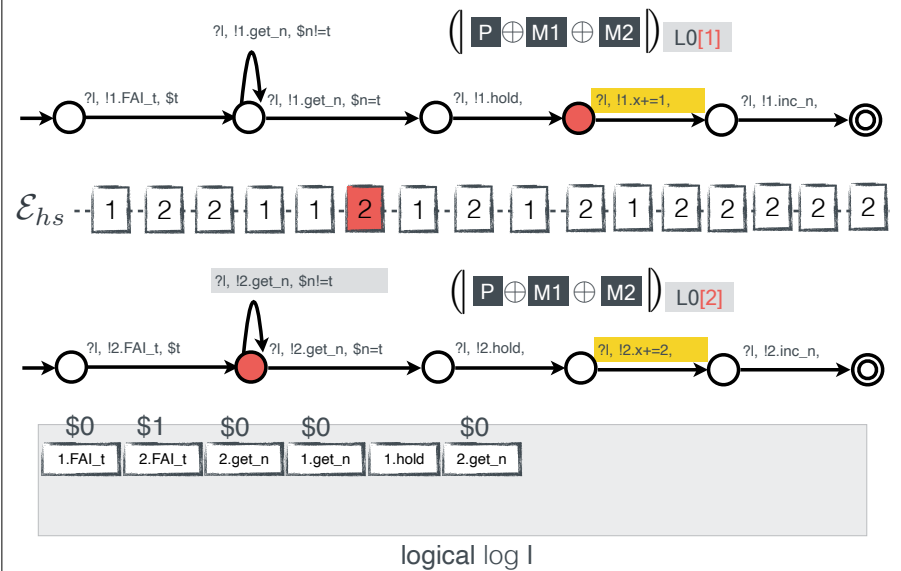
## Strategies and Game Semantics



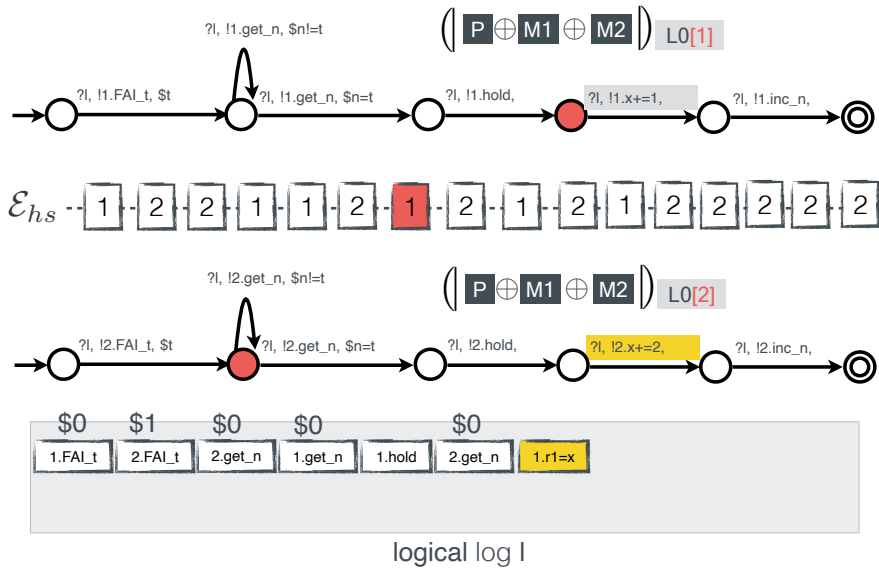
## Strategies and Game Semantics



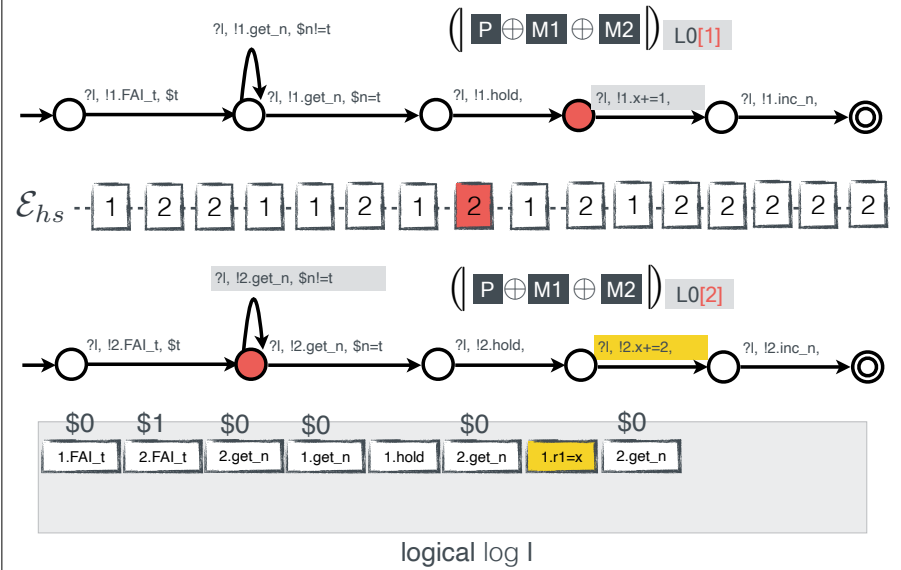
## Strategies and Game Semantics



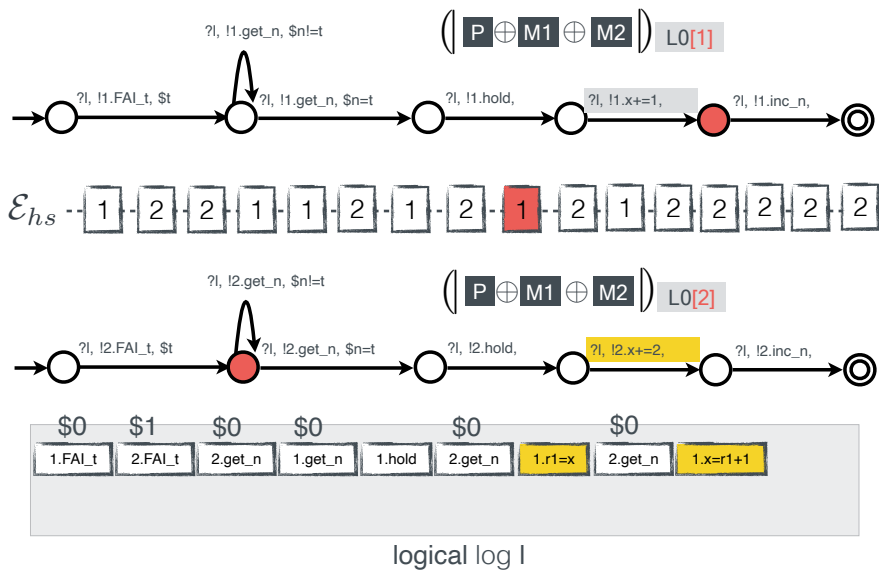
### Strategies and Game Semantics



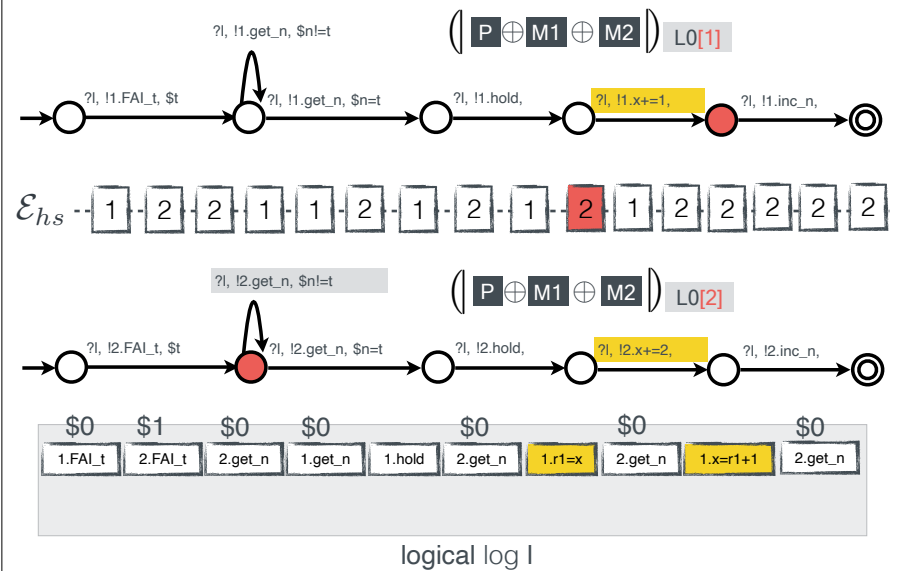
### Strategies and Game Semantics



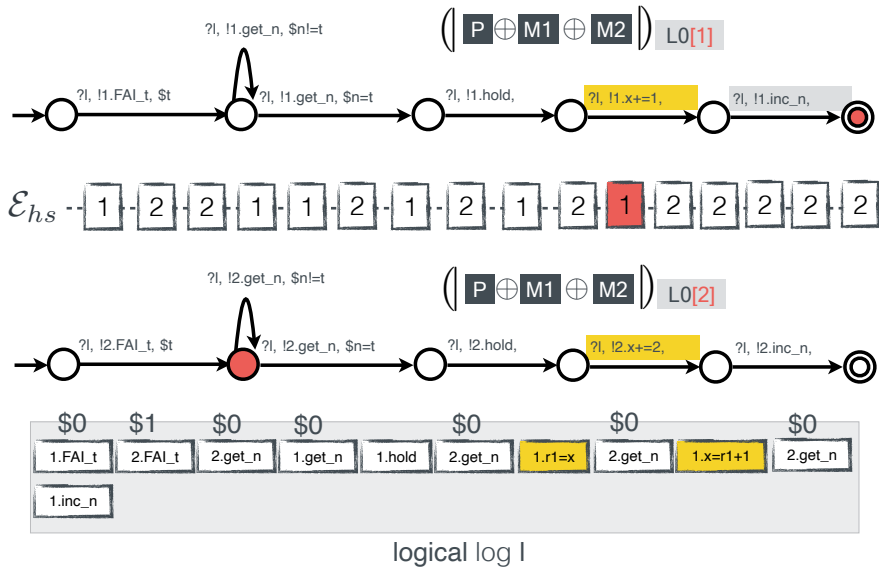
### Strategies and Game Semantics



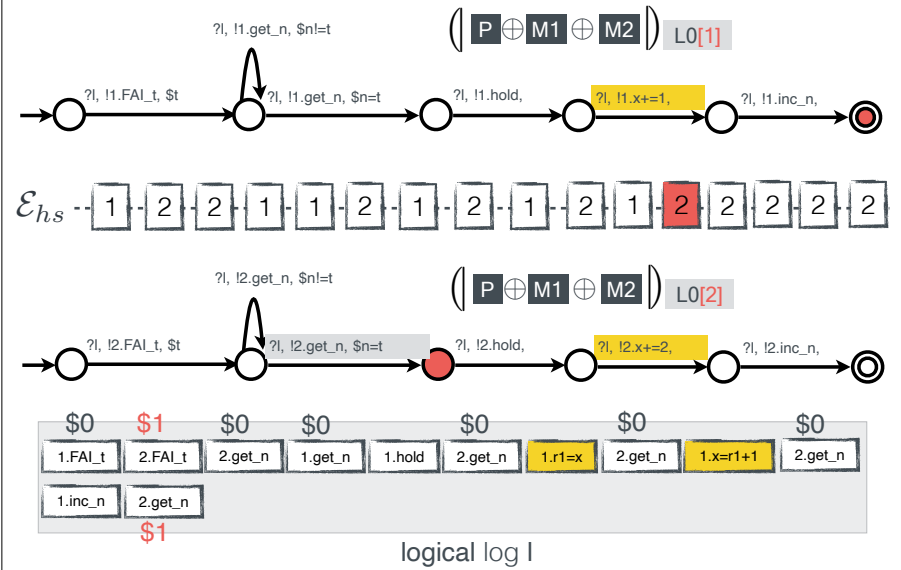
### Strategies and Game Semantics



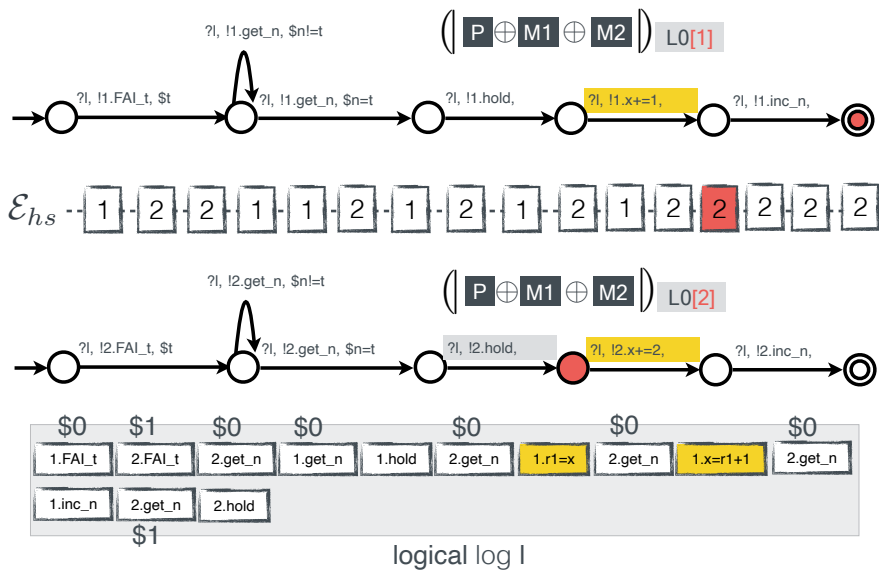
### Strategies and Game Semantics



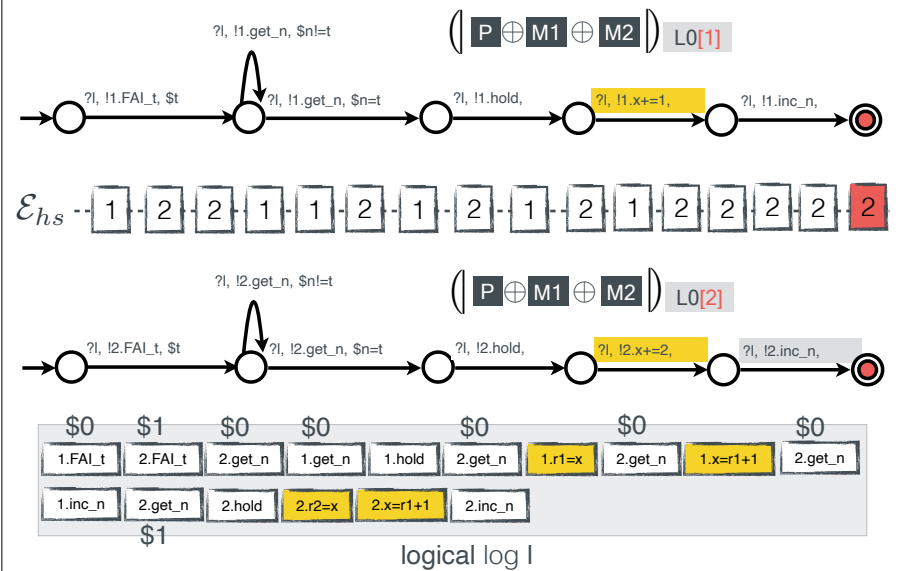
### Strategies and Game Semantics



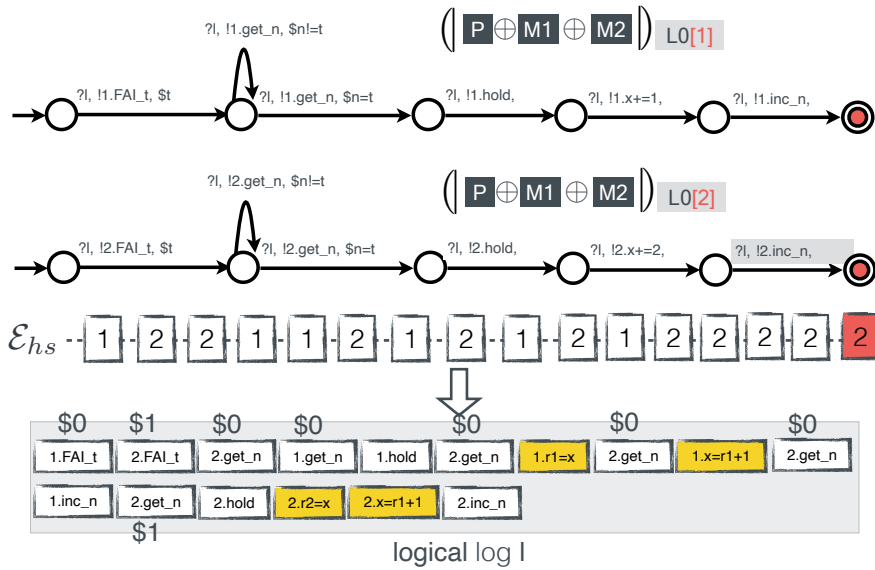
### Strategies and Game Semantics



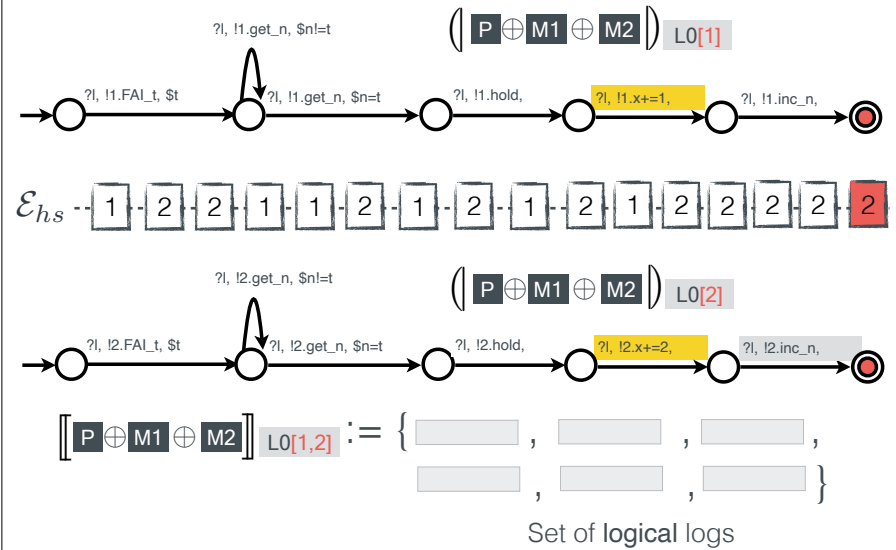
### Strategies and Game Semantics



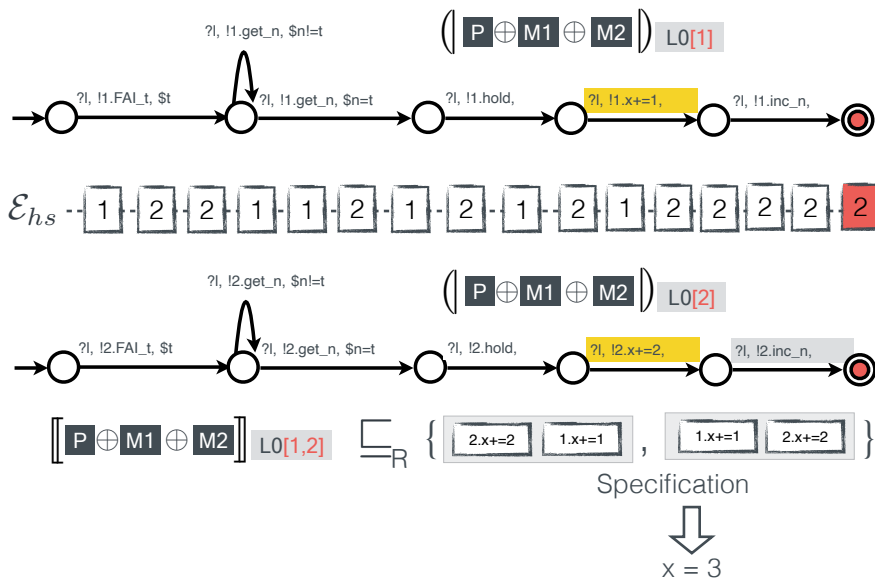
### Strategies and Game Semantics



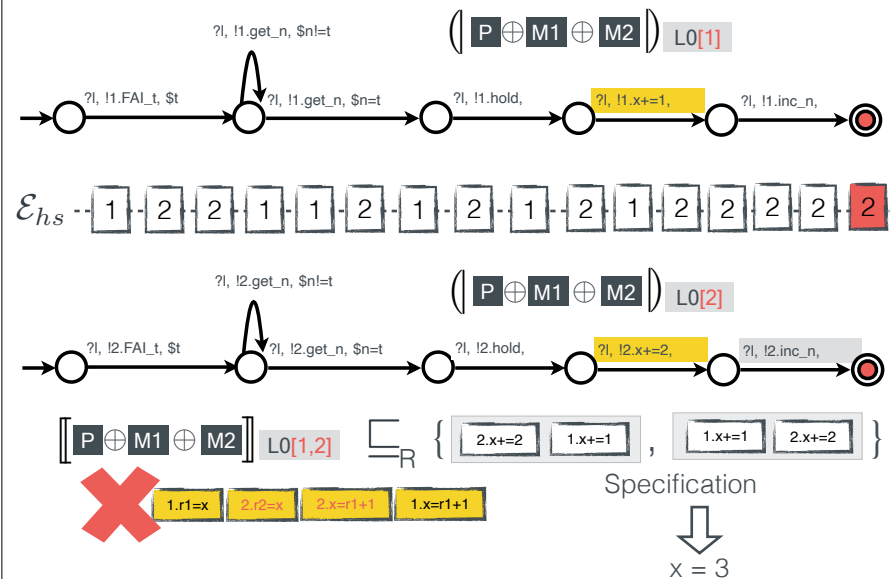
### Strategies and Game Semantics



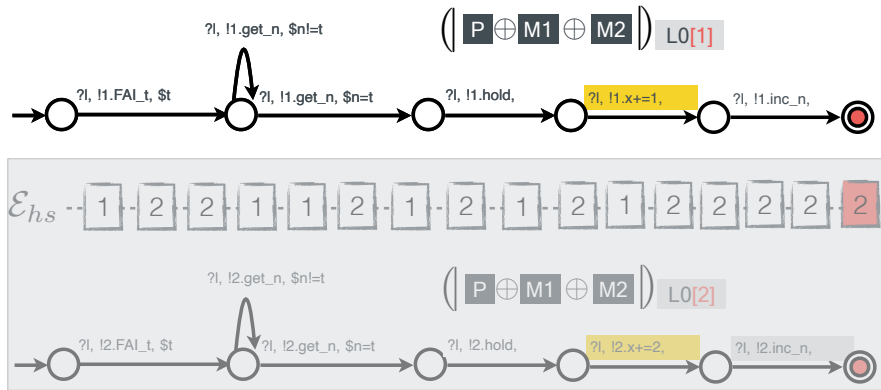
### Strategies and Game Semantics



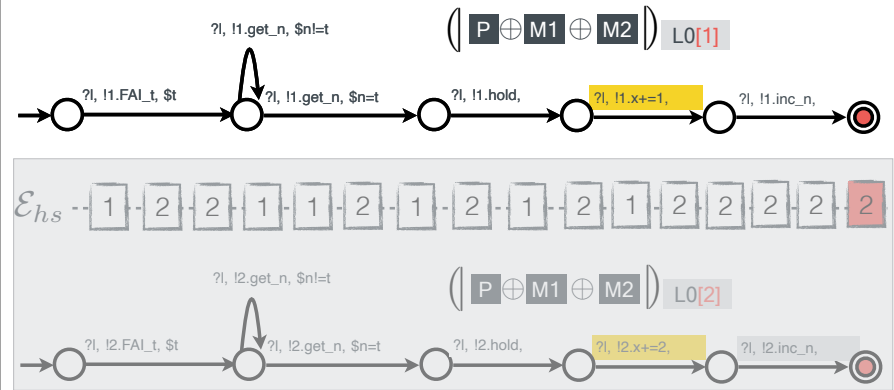
### Strategies and Game Semantics



### Strategy Refinement



### Strategy Refinement



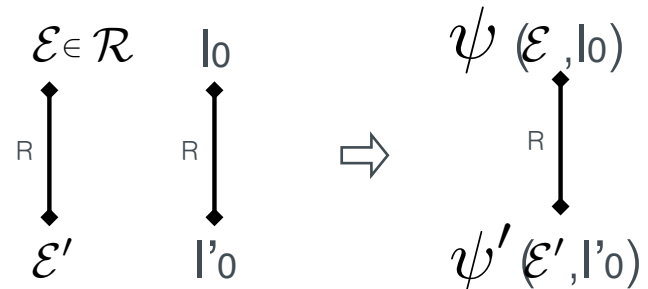
Environment Context  $\mathcal{E} \in \mathcal{R}$

### Strategy Refinement

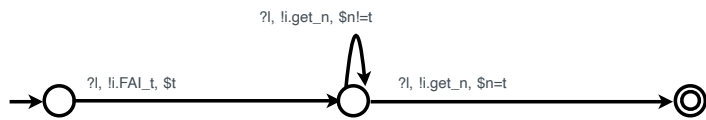
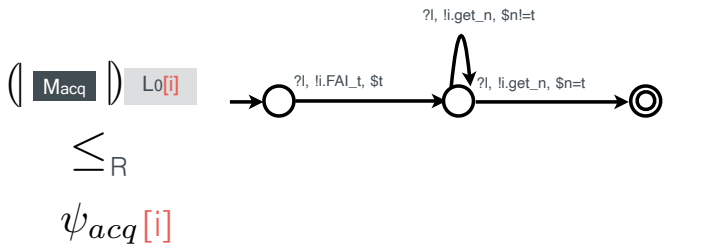
$$\psi(\mathcal{E}, l_0)$$

### Strategy Refinement

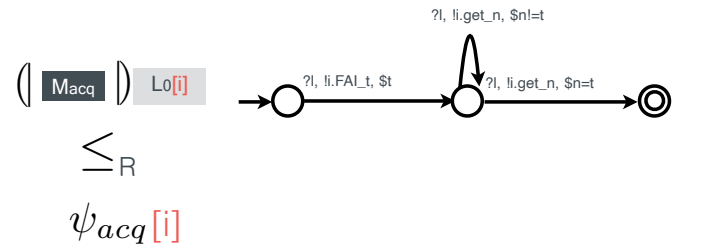
$$\psi \leq_R \psi'$$



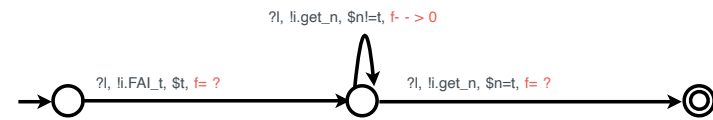
### Strategy Refinement



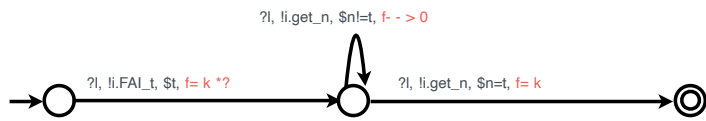
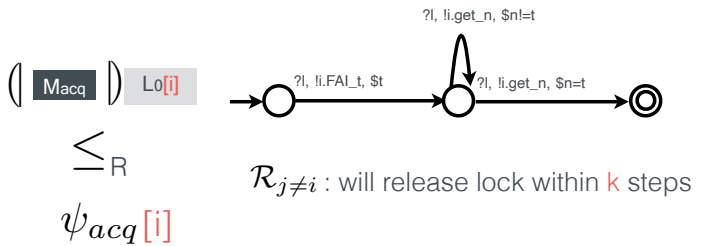
### Strategy Refinement



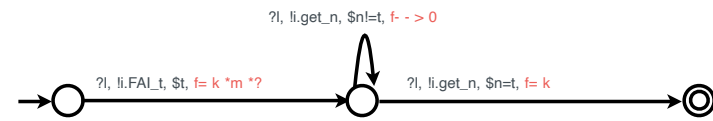
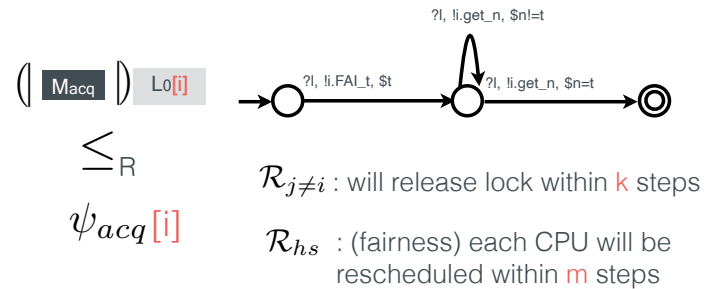
Add fuel (f) to prove liveness



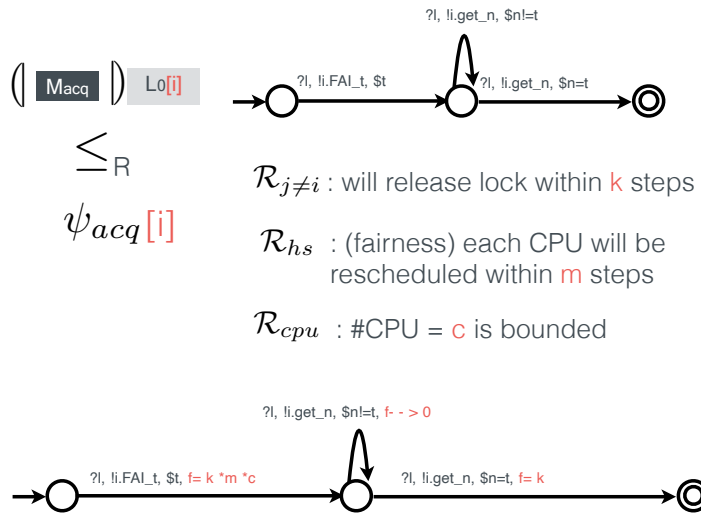
### Strategy Refinement



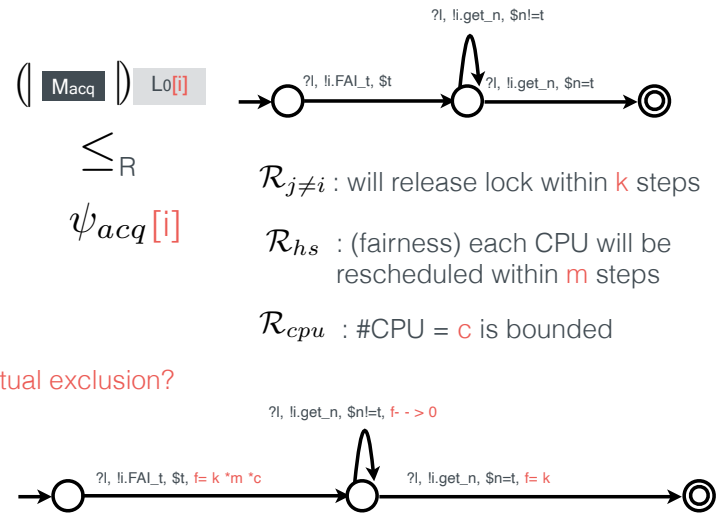
### Strategy Refinement



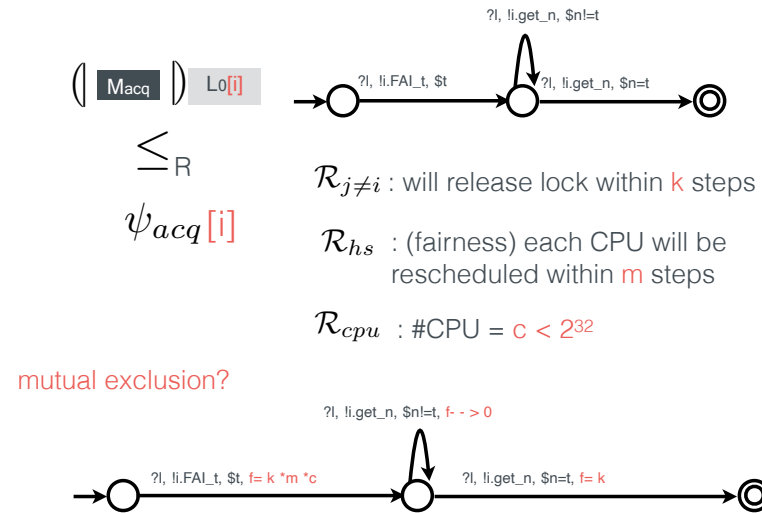
## Strategy Refinement



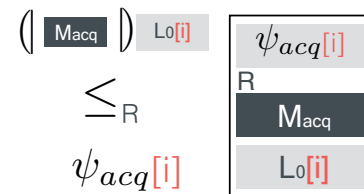
## Strategy Refinement



## Strategy Refinement

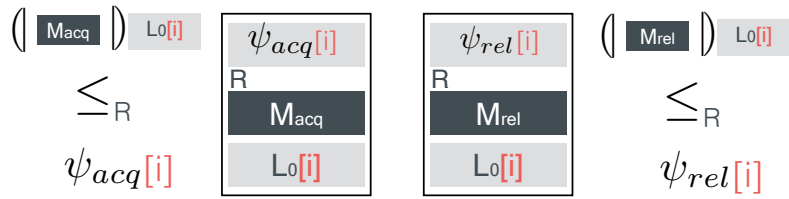


## Certified Concurrent Abstraction Layer

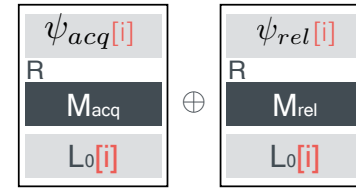




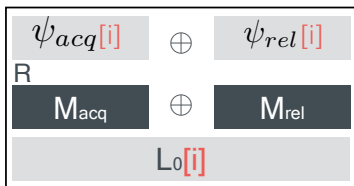
### Certified Concurrent Abstraction Layer



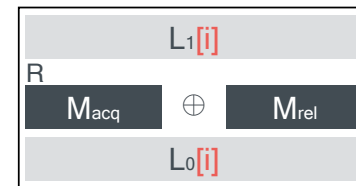
### Horizontal Composition



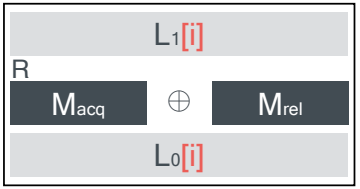
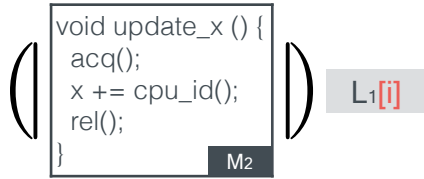
### Horizontal Composition



### Certified Concurrent Abstraction Layer



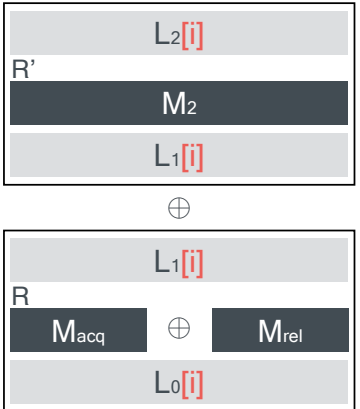
### Certified Concurrent Abstraction Layer



### Certified Concurrent Abstraction Layer



### Vertical Composition



### Vertical Composition

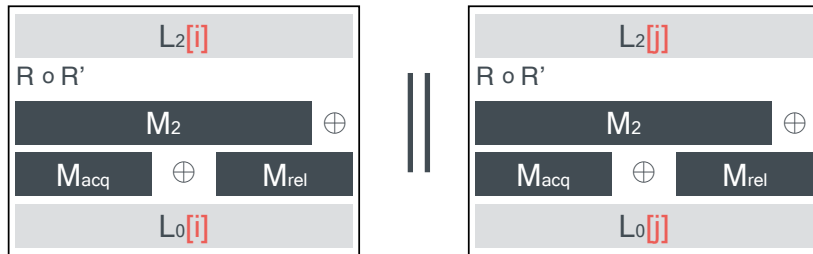


## Parallel Composition

$\mathcal{R}_{j \neq i}$  : will release lock within  $k$  steps

$\mathcal{R}_{hs}$  : (fairness) each CPU will be rescheduled within  $m$  steps

$\mathcal{R}_{cpu}$  : #CPU =  $c < 2^{32}$

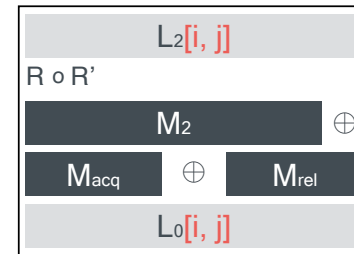


## Parallel Composition

$\mathcal{R}_{j \neq i}$  : will release lock within  $k$  steps

$\mathcal{R}_{hs}$  : (fairness) each CPU will be rescheduled within  $m$  steps

$\mathcal{R}_{cpu}$  : #CPU =  $c < 2^{32}$



## Parallel Composition

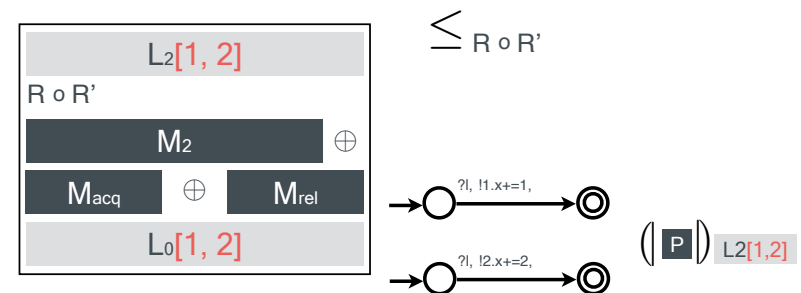
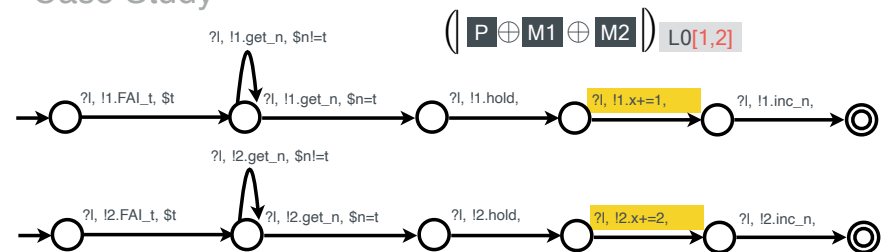
$\mathcal{R}_{hs}$  : (fairness) each CPU will be rescheduled within  $m$  steps

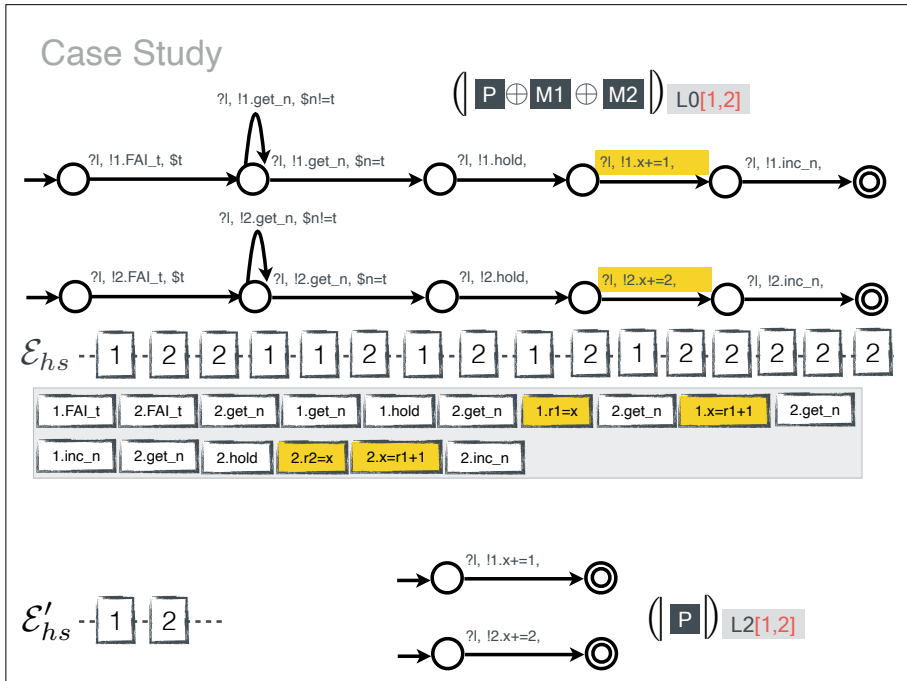
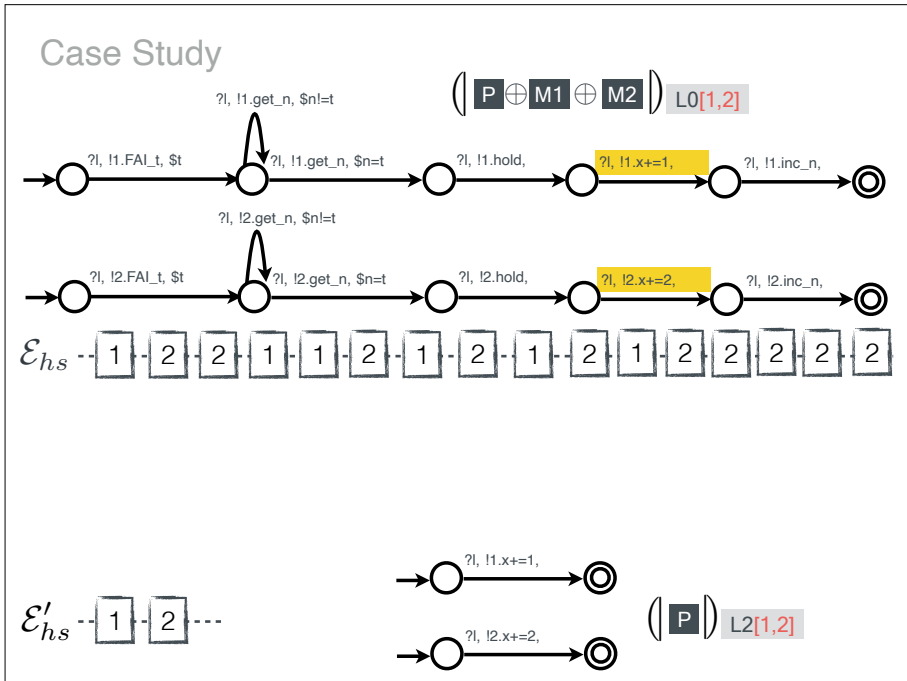
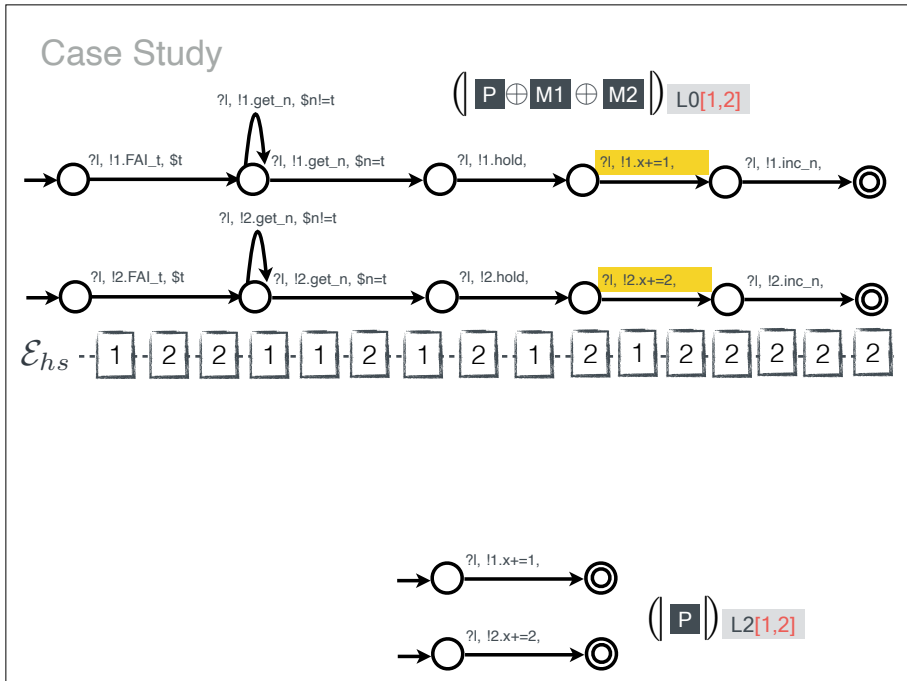
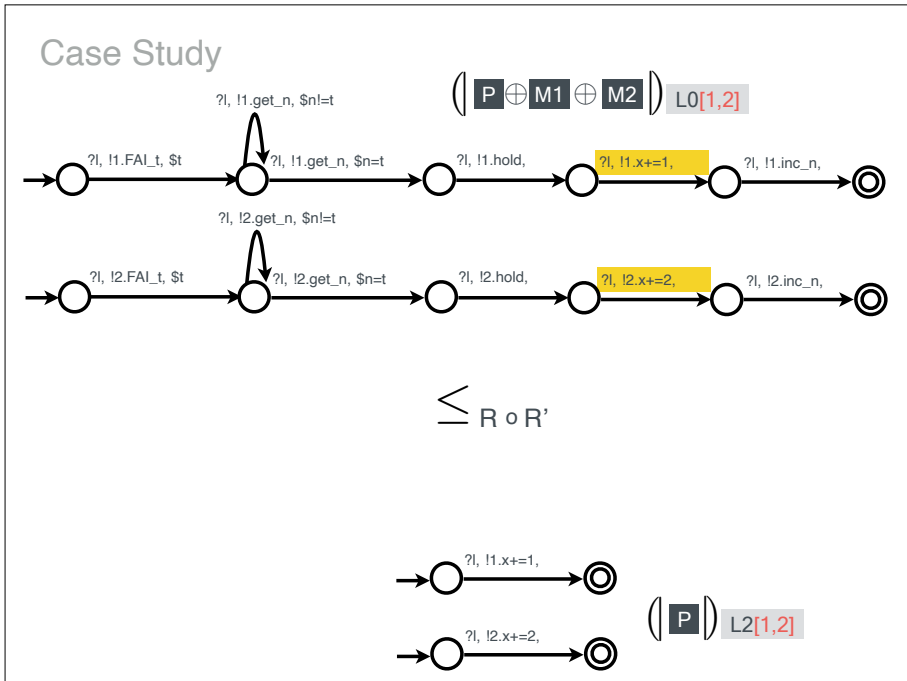
$\mathcal{R}_{cpu}$  : #CPU =  $c < 2^{32}$

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

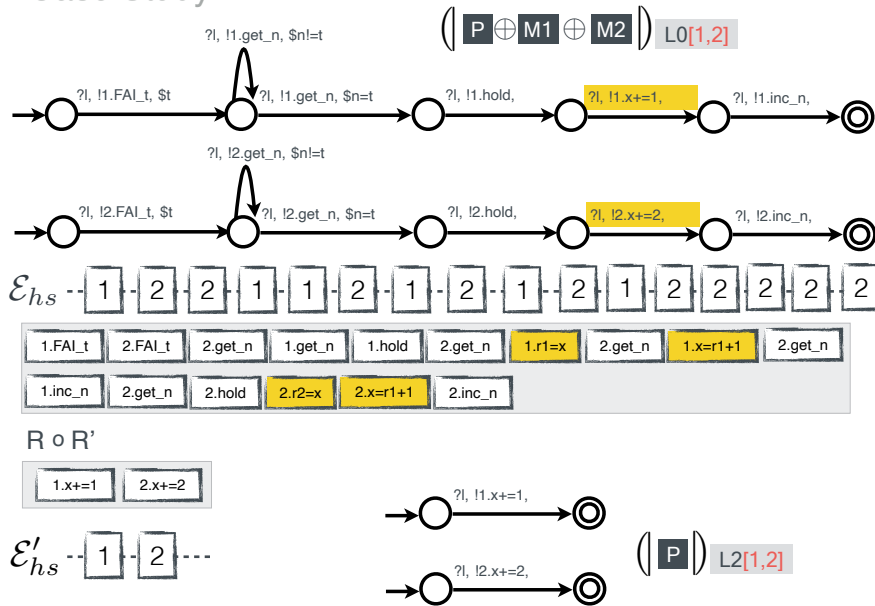


## Case Study





## Case Study



## Soundness

$$\llbracket P \oplus M1 \oplus M2 \rrbracket L0[1,2] \sqsubseteq_{R \circ R'} \llbracket P \rrbracket L2[1,2]$$

## Soundness

$\mathcal{R}_{hs}$  : (fairness) each CPU will be rescheduled within  $m$  steps

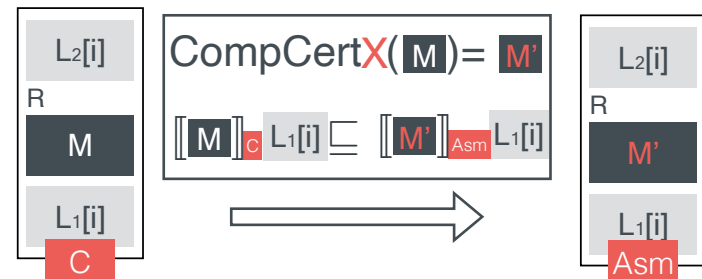
$\mathcal{R}_{cpu}$  : #CPU =  $c < 2^{32}$

$$\llbracket P \oplus M1 \oplus M2 \rrbracket L0[1,2] \sqsubseteq_{R \circ R'} \llbracket P \rrbracket L2[1,2]$$

$$= \{ \boxed{2.x+=2} \boxed{1.x+=1}, \boxed{1.x+=1} \boxed{2.x+=2} \}$$

QED

## CompCertX



## Assembly Layers



Horizontal Composition

Vertical Composition

Parallel Composition

## Software Scheduler

```
void yield () {
  uint t = tid();
  ...
  enq (t, rdq());
  uint s = deq (rdq());
  ...
  context_switch (t, s)
}
```

Shared Queue Lib

Spinlock

## Software Scheduler

Software Scheduler

sleep

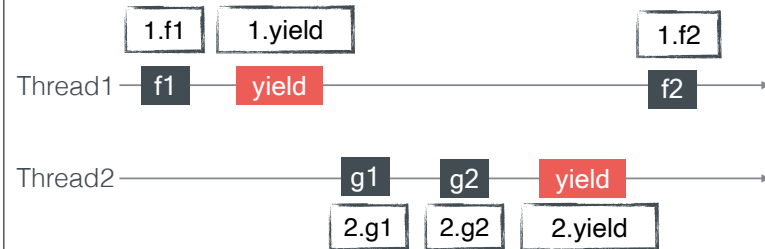
yield

wakeup

Shared Queue Lib

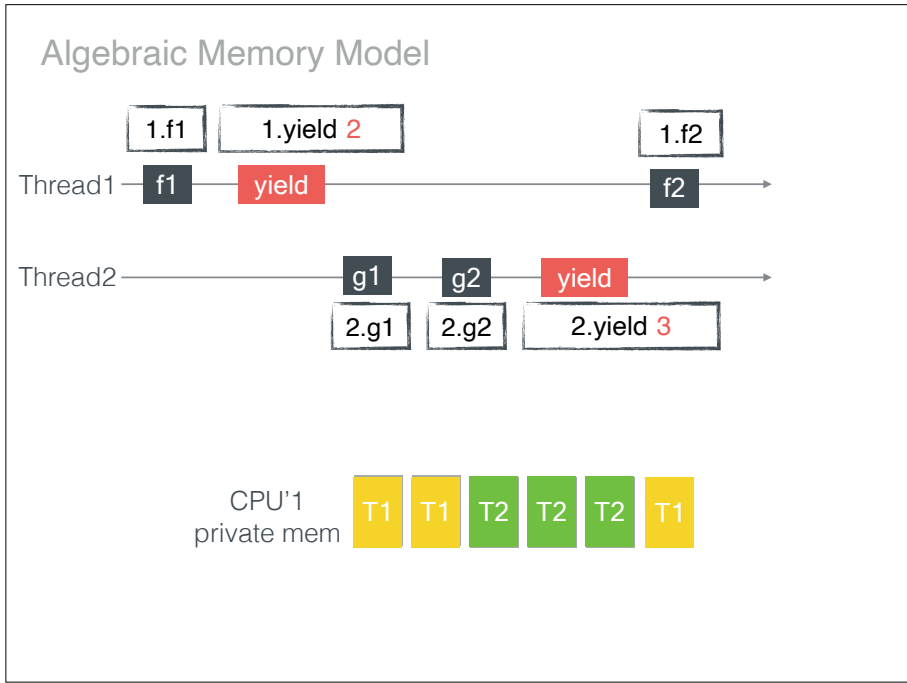
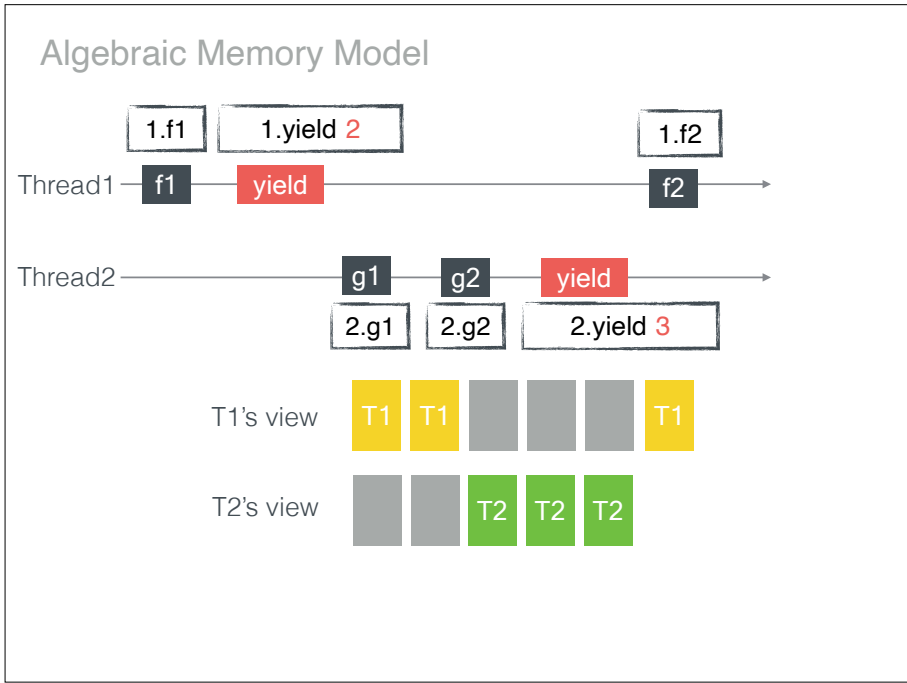
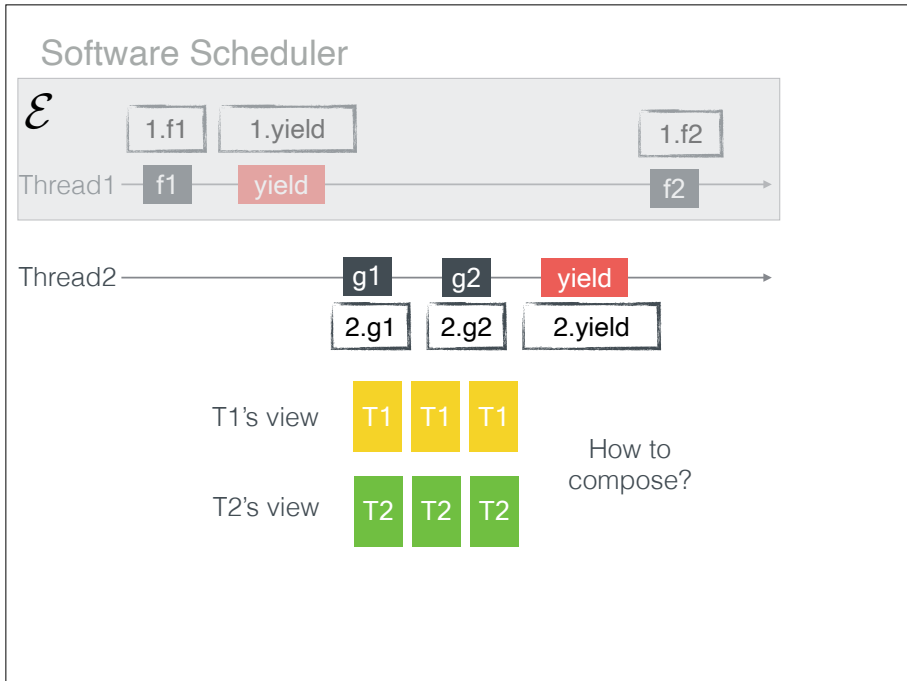
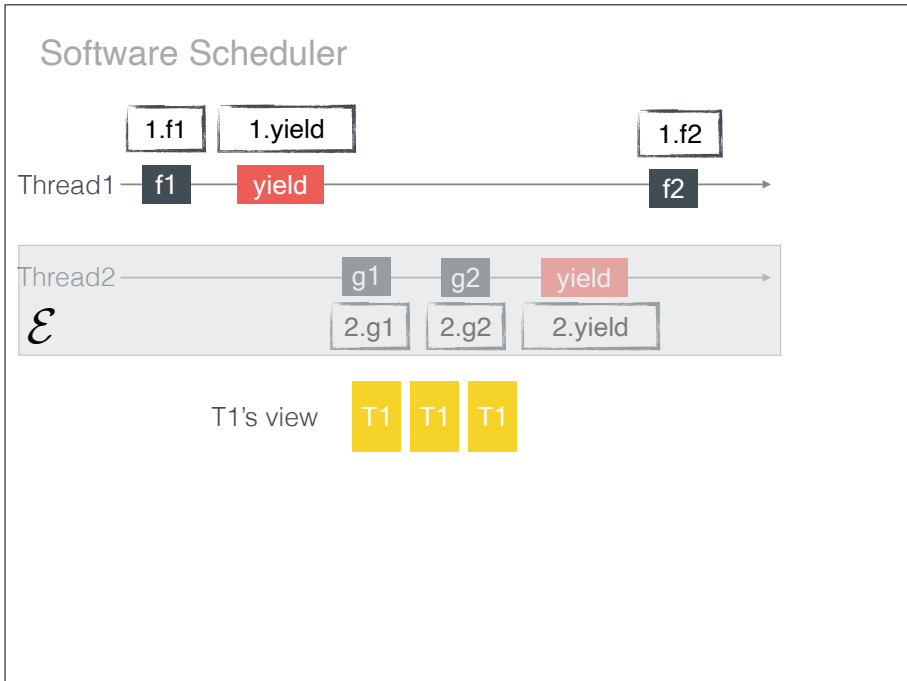
Spinlock

## Software Scheduler



CPU'1  
private mem

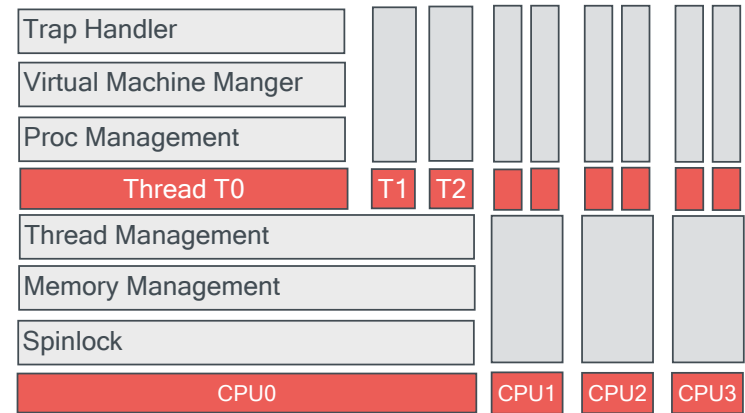




CompCertX

$$\text{CompCertX} + \text{Algebraic Memory Model} = \text{Thread-safe Verified Compiler}$$

## Verification of a Concurrent OS Kernel



Layer	Refinement proof	Code verification	Source code	Proof linking
--		--	--	<a href="#">MulticoreLinking</a>

Source code linking	CertKOS instance for extraction	Proof linking
<a href="#">CertKOS</a>	<a href="#">CertKOS Instance</a>	<a href="#">CertKOS_correct</a>

### Layers for per-thread

Layer	Refinement proof	Code verification	Source code	Proof linking
<b>Trap module</b>				
<a href="#">TSysCall</a>	<a href="#">SysCallGen</a>	<a href="#">TDispatchAsmCode1</a>	<a href="#">TDispatchAsmSource</a>	<a href="#">SysCallGenLink</a>
		<a href="#">TDispatchAsmCode2</a>		
<a href="#">TDispatch</a>	<a href="#">DispatchGen</a>	<a href="#">TTrapCode</a>	<a href="#">TTrapCSource</a>	<a href="#">DispatchGenLink</a>
<a href="#">TTrap</a>	<a href="#">TrapGen</a>	<a href="#">TTrapArgCode1</a>	<a href="#">TTrapArgCSource1</a>	<a href="#">TrapGenLink</a>
		<a href="#">TTrapArgCode2</a>	<a href="#">TTrapArgCSource2</a>	
		<a href="#">TTrapArgCode3</a>		
		<a href="#">TTrapArgCode4</a>		
		<a href="#">TTrapArgCode5</a>		
		<a href="#">TTrapArgCode6</a>		
<a href="#">TTrapArg</a>	<a href="#">TrapArgGen</a>	<a href="#">PProcCode</a>	<a href="#">PProcSource</a>	<a href="#">TrapArgGenLink</a>
<b>IPC module</b>				
<a href="#">PIPC</a>	<a href="#">IPCGen</a>	<a href="#">PIPCIntroCode</a>	<a href="#">PIPCIntroCSource</a>	<a href="#">IPCGenLink</a>
<a href="#">PIPCIntro</a>	<a href="#">IPCIntroGen</a>	<a href="#">PHThreadCode</a>	<a href="#">PHThreadCSource</a>	<a href="#">IPCIntroGenLink</a>
<b>Multithreaded linking interface</b>				
<a href="#">PHThread</a>	<a href="#">HThreadGen</a>	--	--	<a href="#">HThreadGenLink</a>

### Intermediate layer interface for multithreaded linking

Layer	Refinement proof	Code verification	Source code	Proof linking
<a href="#">PHBThread</a>	<a href="#">HBThreadGen</a>	--	--	<a href="#">HBThreadGenLink</a>

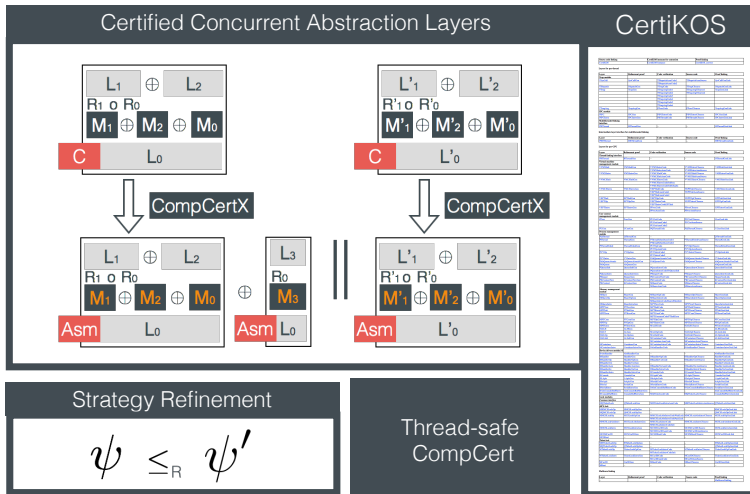
### Layers for per-CPU

Layer	Refinement proof	Code verification	Source code	Proof linking
<b>Thread linking interface</b>				
<a href="#">PBThread</a>	<a href="#">BThreadGen</a>	--	--	<a href="#">BThreadGenLink</a>



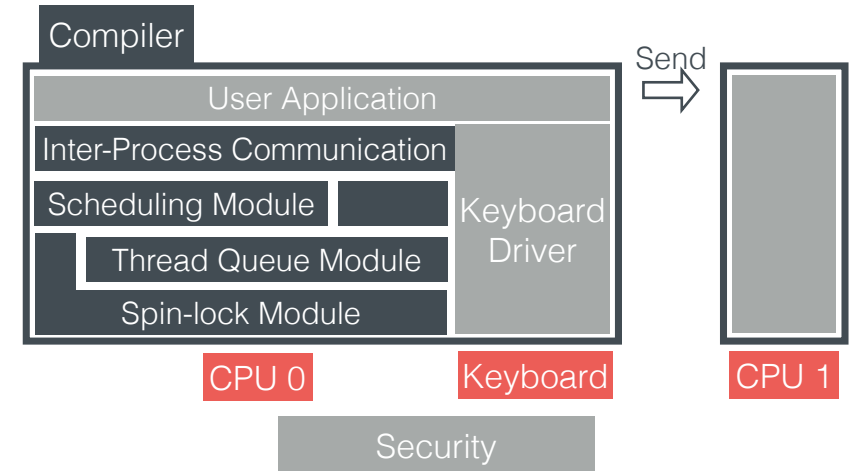


## Contribution Summary



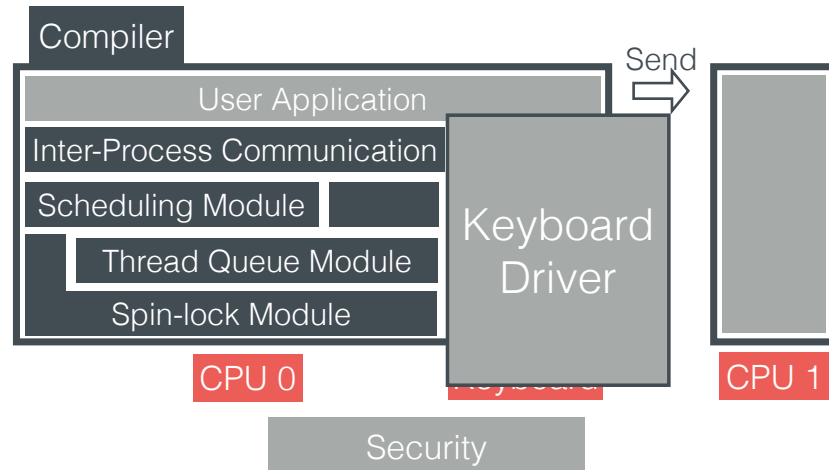
## Case Study

### Build a Certified System

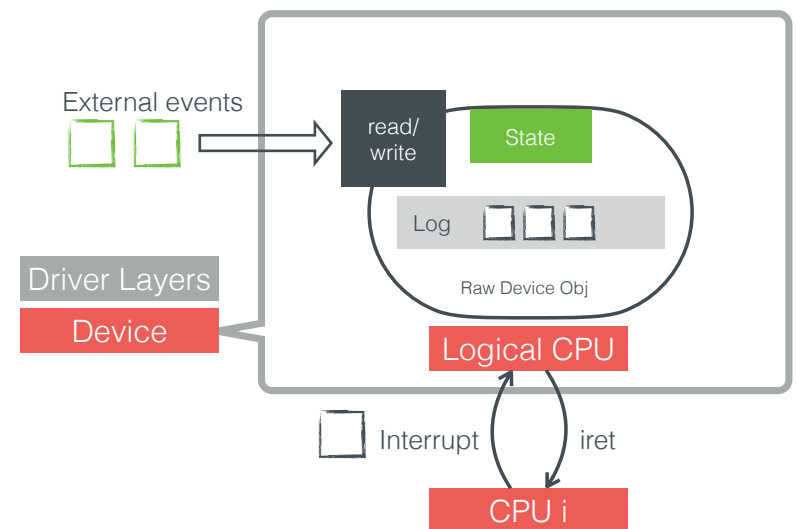


## Case Study

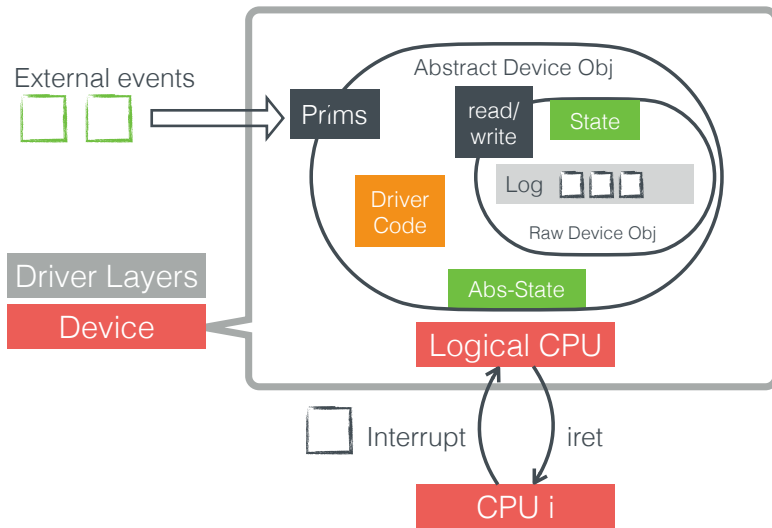
### Build a Certified System



## Device Driver [PLDI16'a]

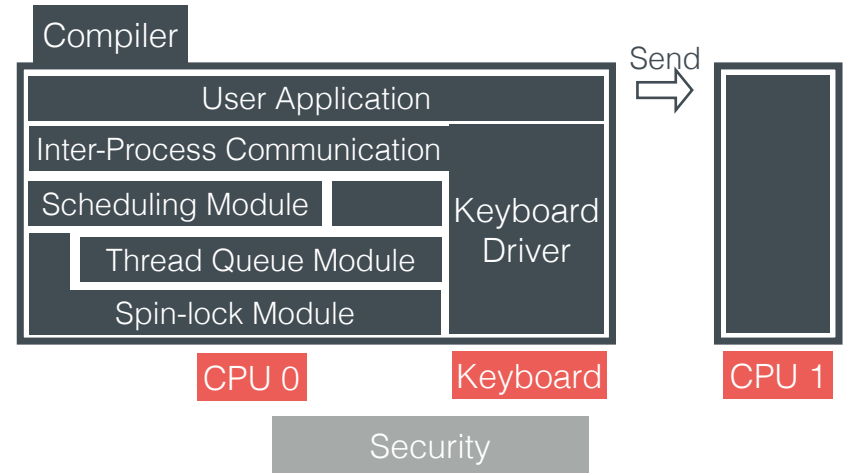


## Device Driver [PLDI16'a]



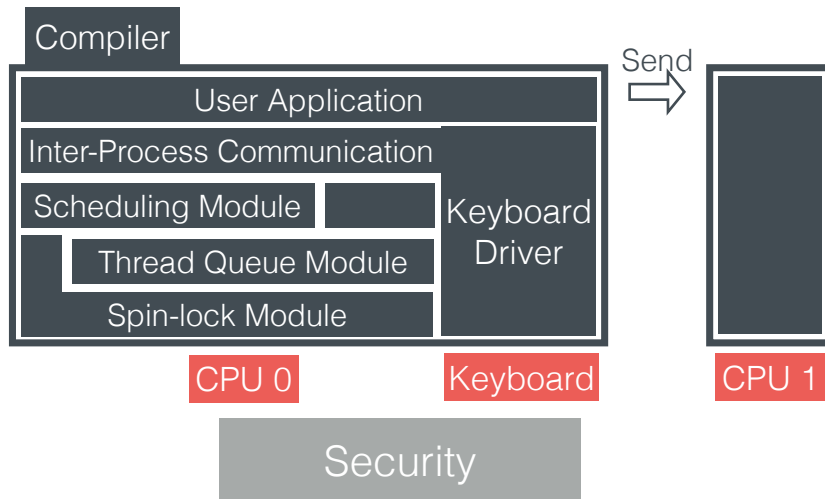
## Case Study

### Build a Certified System

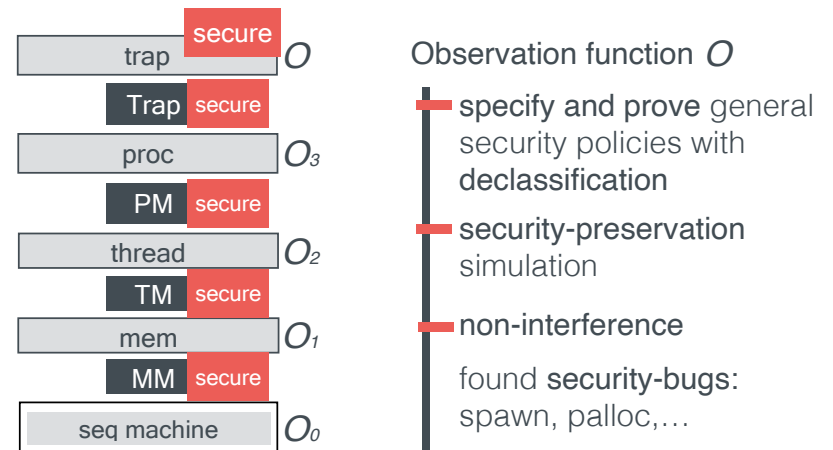


## Case Study

### Build a Certified System

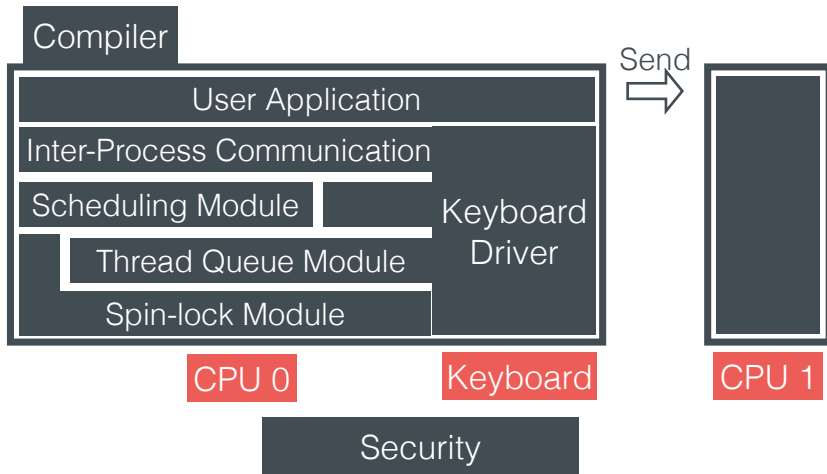


## End-to-End Security [PLDI16'b]



## Case Study

### Build a Certified System



## Summary: The CertiKOS / DeepSpec Project

**Killer-app:** high-assurance “heterogeneous” systems of systems!

**Conjecture:** today’s PLs fail because they ignored OS, and today’s OSES fail because they get little help from PLs

### New Insights:

- deepspec & certified abstraction layers;
- a unifying framework for composing heterogeneous components (via game semantics + linear logic connectives)

### Opportunities:

- New certified system software stacks (CertiKOS ++)
- New certifying programming languages (DeepSEA vs. C & Asm)
- New certified programming tools
- New certified modeling & arch. description lang. (DeepSEA)
- We verify all interesting properties (correctness, safety, security, availability, ...)