CertiKOS: From Hacker-Resistant OS to Certified Heterogeneous Systems

Zhong Shao
Yale University
December 8, 2021

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.

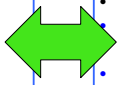
The Future of CS: Heterogeneous Systems

- 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, …)

CertiKOS Problem Definition

CERTIKOS

- 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

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

“…nal verification is the **only** …is …”

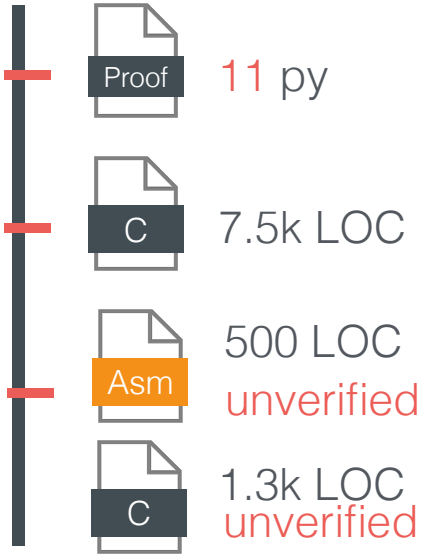
…SP’09]

“…way to …ter …”

—NSF SFM Report[2016]

Challenges: huge proof efforts

seL4
[SOSP’09]



Challenges: Compositionality

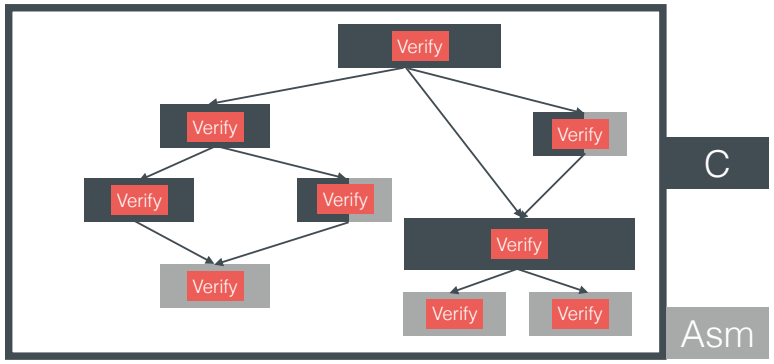
Abstraction Gap

Challenges: Compositionality

A Complex System

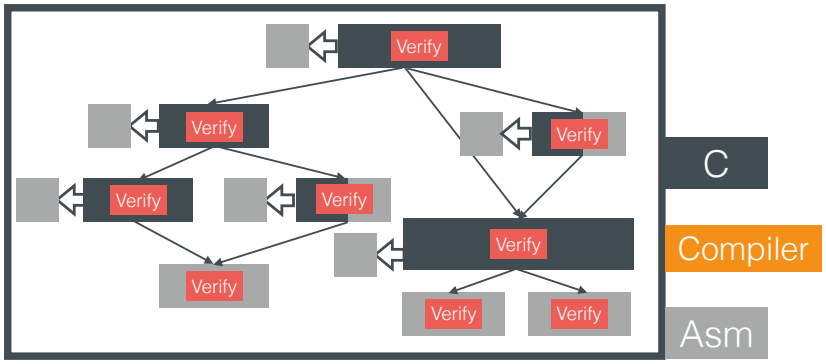
Challenges: Compositionality

A Complex System

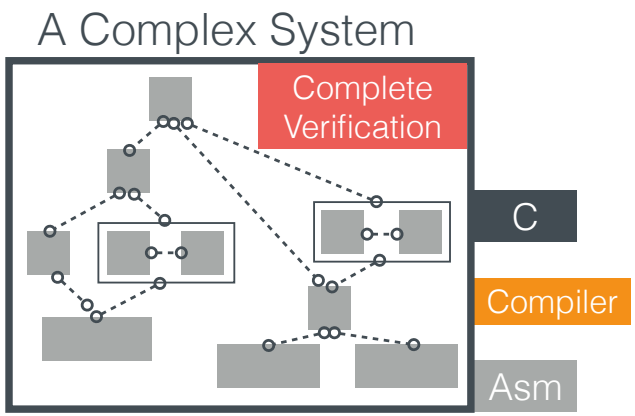


Challenges: Compositionality

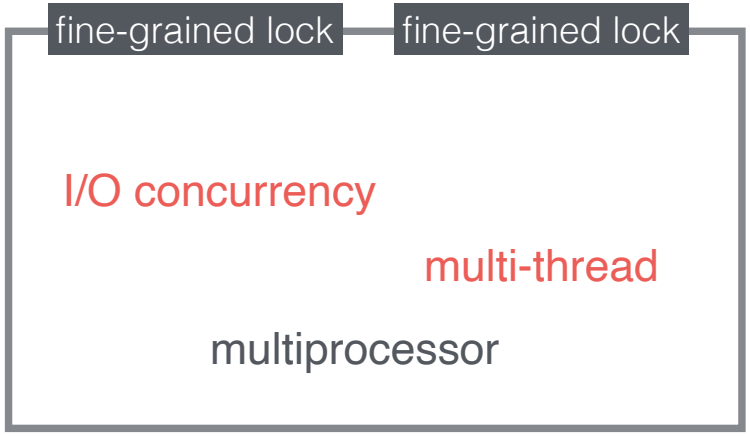
A Complex System



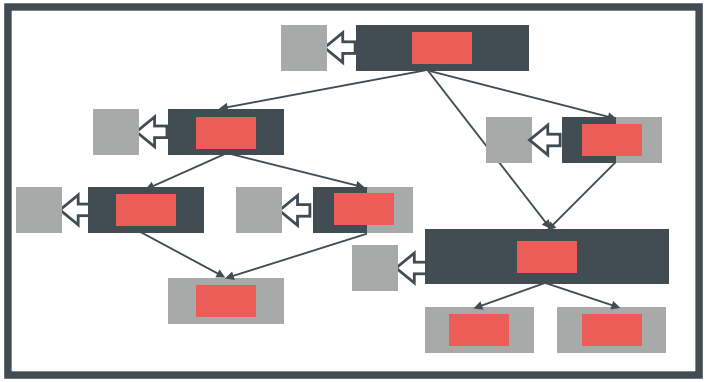
Challenges: Compositionality



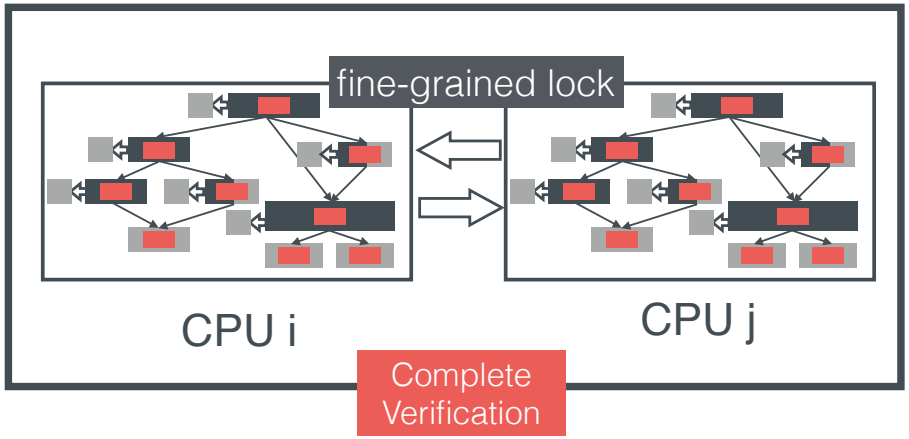
Challenges: Concurrency



Challenges: Concurrency



Challenges: Concurrency



Contribution

Certified Abstraction Layers

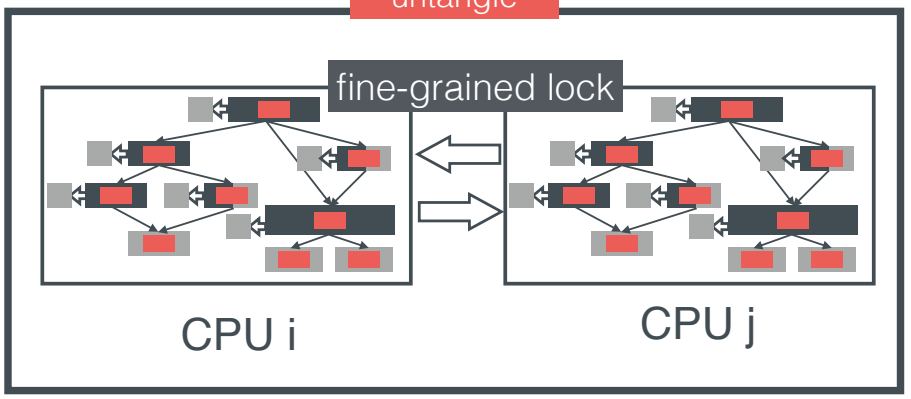
CertiKOS

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

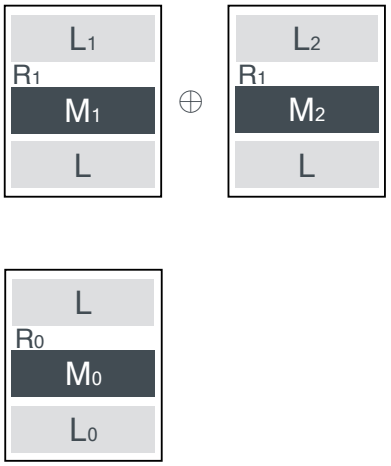
Contribution

Certified Abstraction Layers

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

Contribution

Certified Abstraction Layers

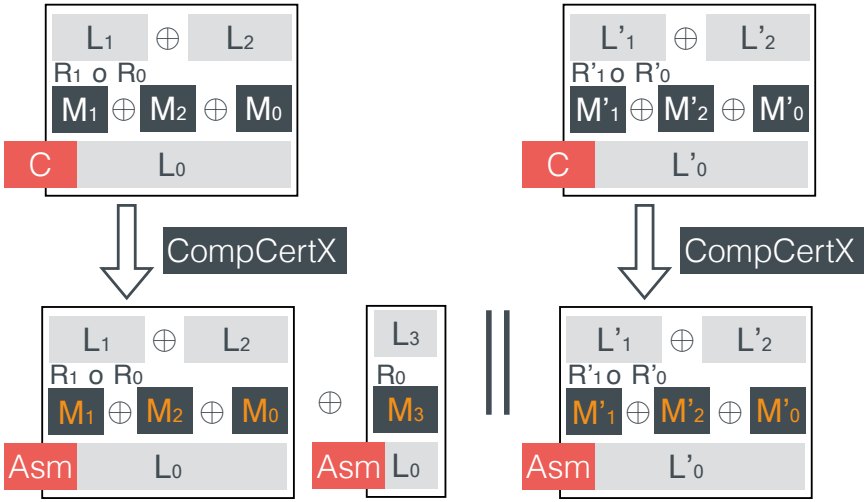


Contribution

Certified Abstraction Layers

Contribution

Certified Abstraction Layers



Contribution

Certified Abstraction Layers

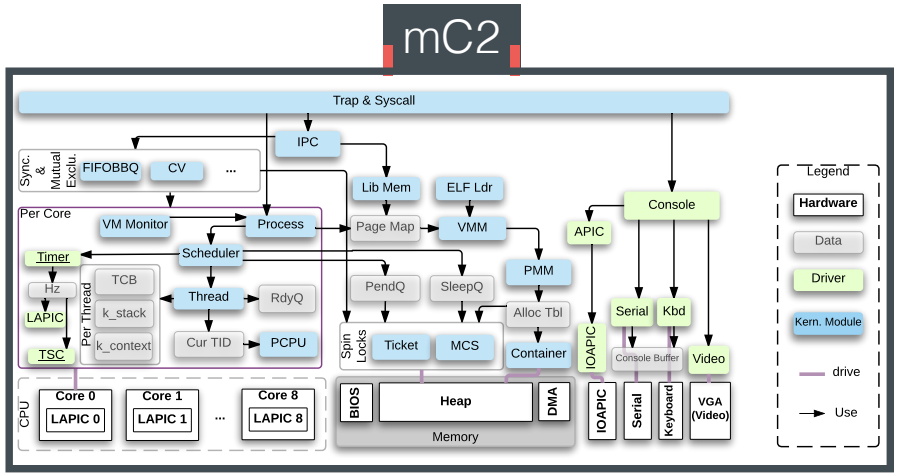
- **mCertiKOS** [POPL'15]
  certified sequential OS kernels
  3k C&Asm, 1 py
- **Interrupt** [PLDI'16a] 0.5 py
- **Security** [PLDI'16b] 0.5 py
- **mC2** [OSDI'16] [PLDI'18]
  the **first** formally certified concurrent OS kernel with fine-grained locks
  6.5k C&Asm, 2 py

Contribution

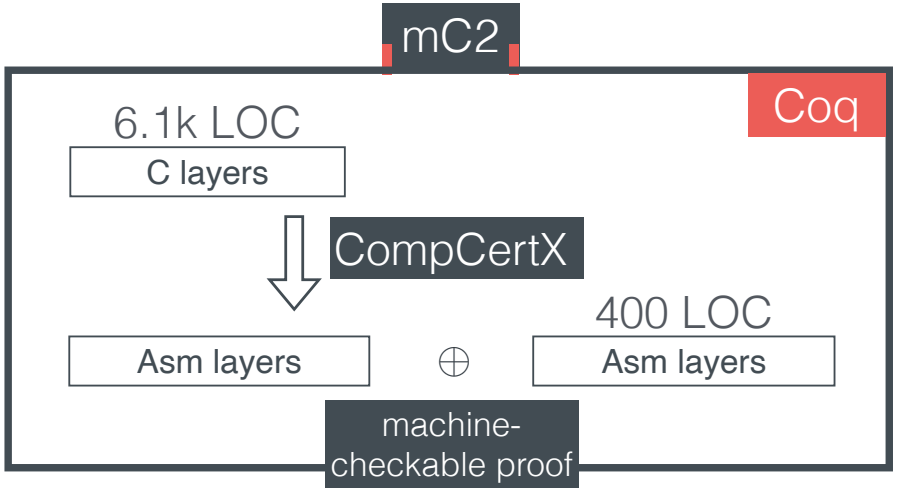
Certified System Software

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

Contribution

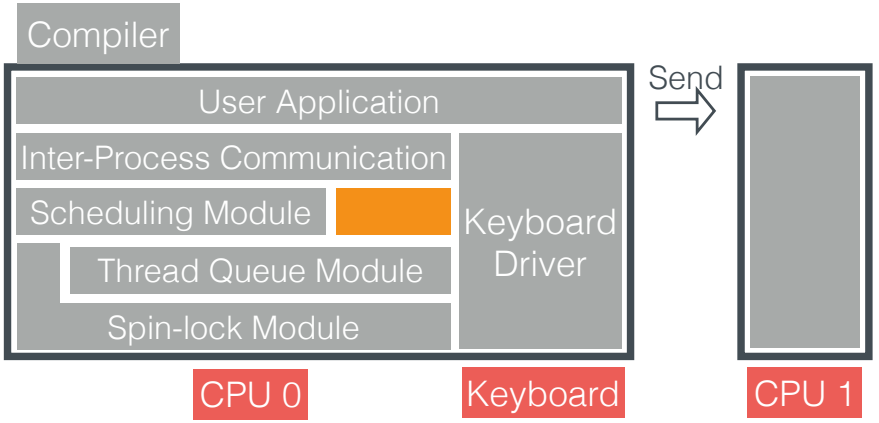


Contribution

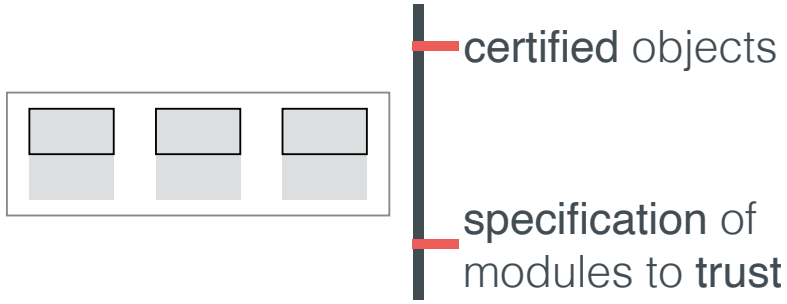


Case Study

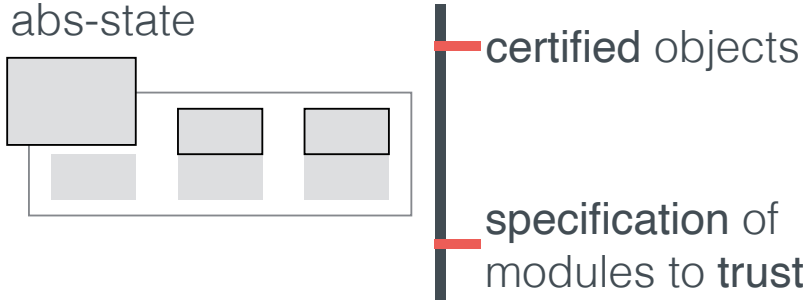
Build a Certified System



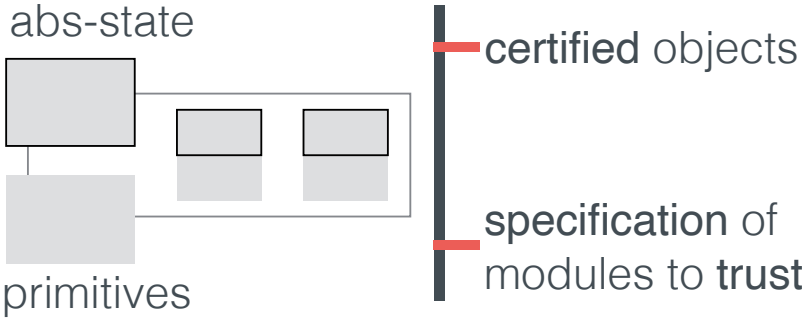
Certified Sequential Layer [POPL'15]



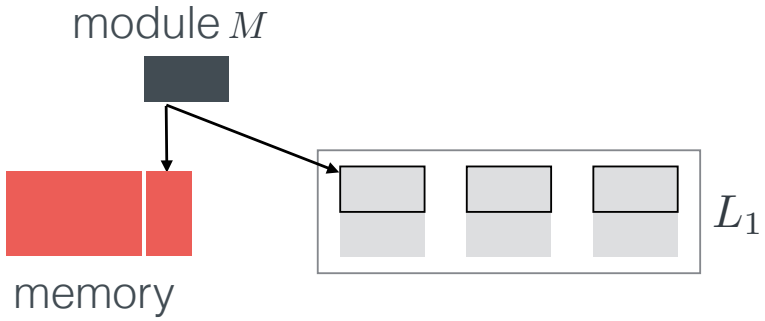
Certified Sequential Layer [POPL'15]



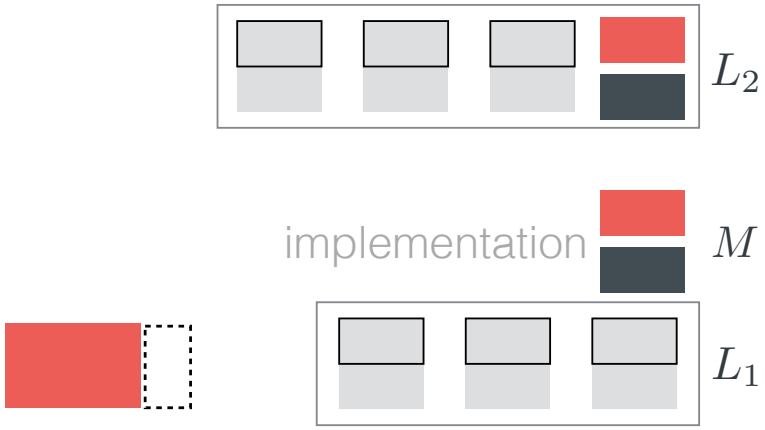
Certified Sequential Layer [POPL'15]



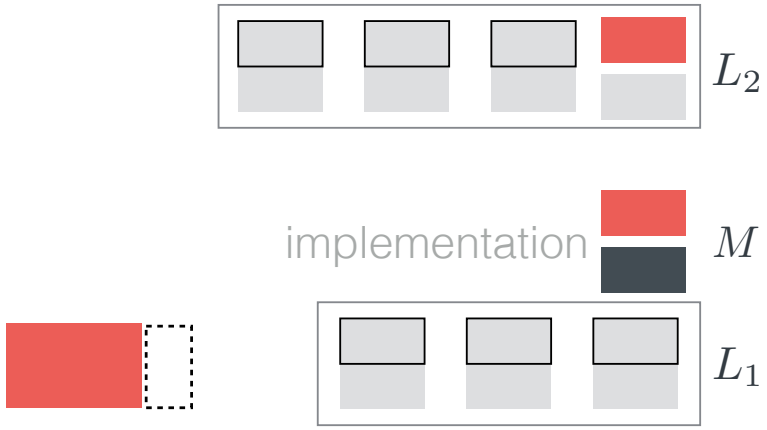
Certified Sequential Layer



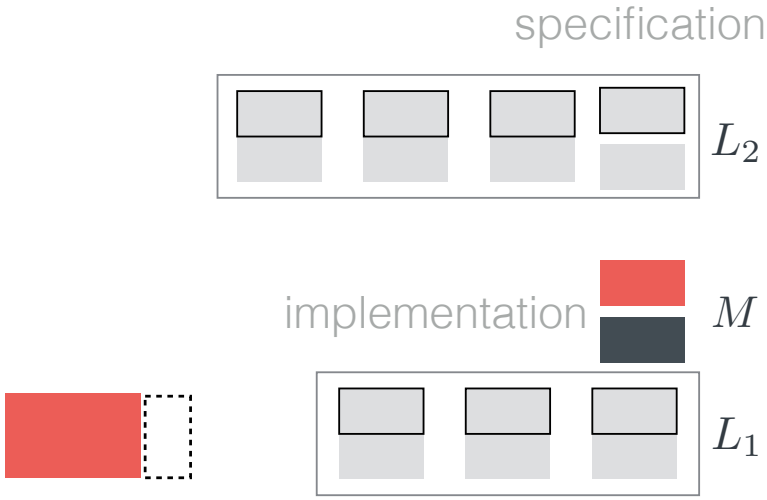
Certified Sequential Layer



Certified Sequential Layer



Certified Sequential Layer



Example: Thread Queue

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

tcb tcbp[1024];
```

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

tdq* td_queue;
```

C

Example: Thread Queue

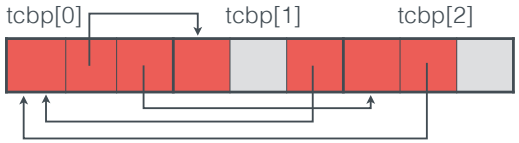
```
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];
```

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

tdq* td_queue;
```

C

M

implementation



Example: Thread Queue

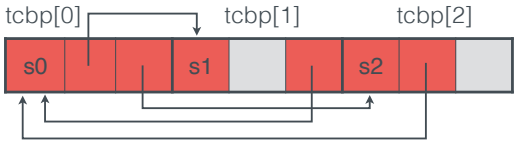
```
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];
```

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

tdq* td_queue;
```

C

M

implementation



Example: Thread Queue

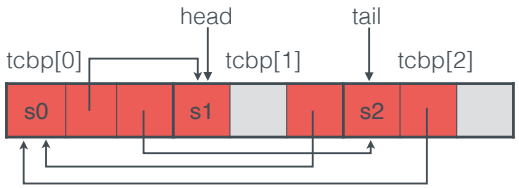
```
typedef struct tcb {
    state s;
    tcb *prev, *next;
} tcb;

tcb tcbp[1024];
```

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

tdq* td_queue;
```

C

M

implementation



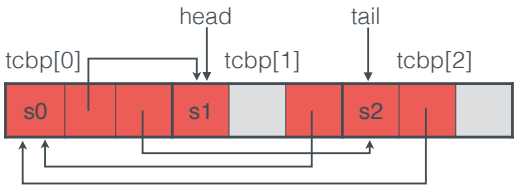
Example: Thread Queue

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

C

M

implementation

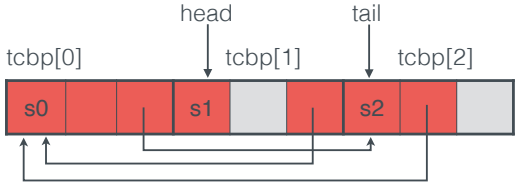


Example: Thread Queue

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

C

M



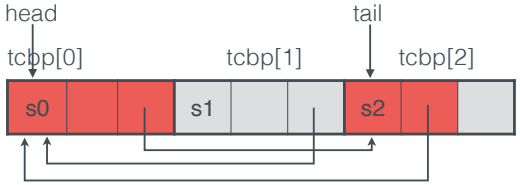
implementation

Example: Thread Queue

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

C

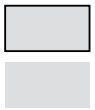
M



implementation

Example: Thread Queue

specification



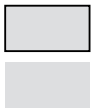
L_2

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

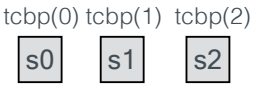
Coq

Example: Thread Queue

specification



L_2

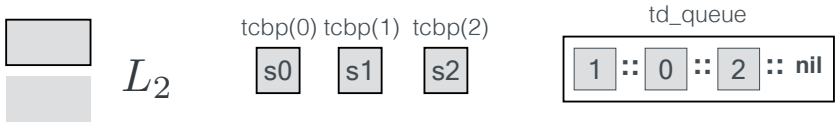


Definition **tcbp** := ZMap.t state.
Definition **td_queue** := List Z.

Coq

Example: Thread Queue

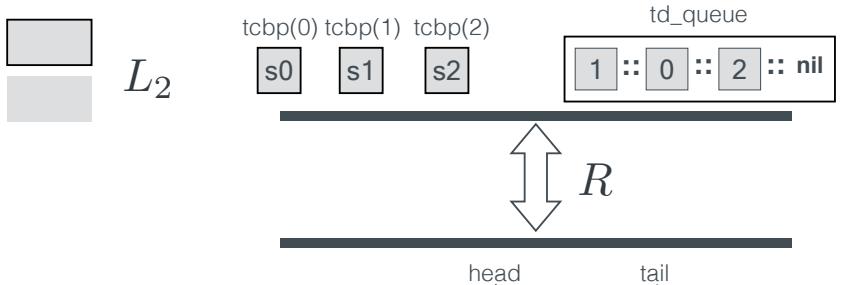
specification

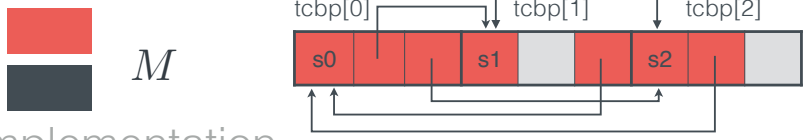



```
Definition tcbp := ZMap.t state.
Definition td_queue := List Z.
```
Coq

Example: Thread Queue

specification

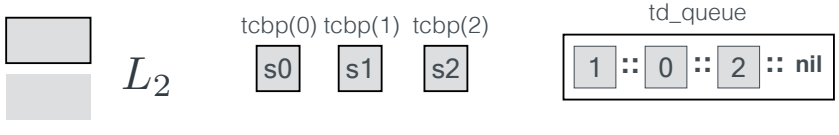




implementation

Example: Thread Queue

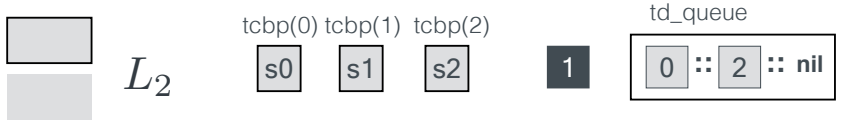
specification




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

Example: Thread Queue

specification

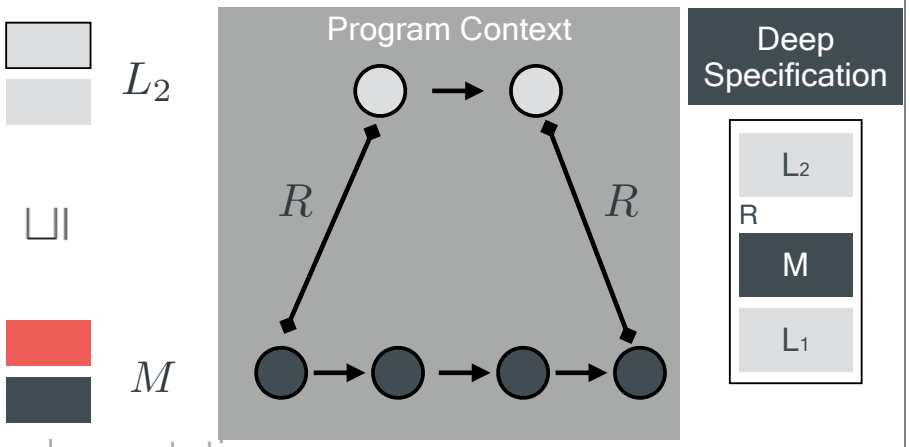



```
Function dequeue (q) :=
match q with
  | head :: q' => (q', Some head)
  | nil => (nil, None)
end.
```
Coq

executable

Simulation Proof

specification



implementation

Deep Specification [POPL'15]

- **Deep spec** L2 captures **all** we need to know about M over L1
- Any property about M can be proved using L2 alone
- No need to look at M again

mCertiKOS

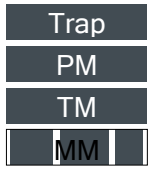
kernel

code

seq machine

mCertiKOS

kernel



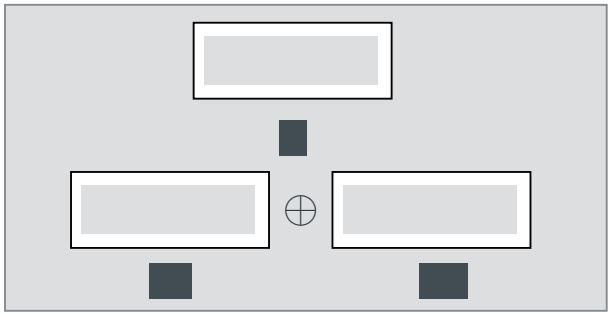
seq machine

mCertiKOS

kernel

Trap
PM
TM
MM

memory management



seq machine

mCertiKOS

kernel

Trap
PM
TM
MM

trap

Trap

proc

PM

thread

TM

mem

seq machine

mCertiKOS

Trap
VM
PM
TM
MM

certified sequential kernel

trap

proc

thread

mem

seq machine

mCertiKOS

Trap
VM
PM
TM
MM

trap

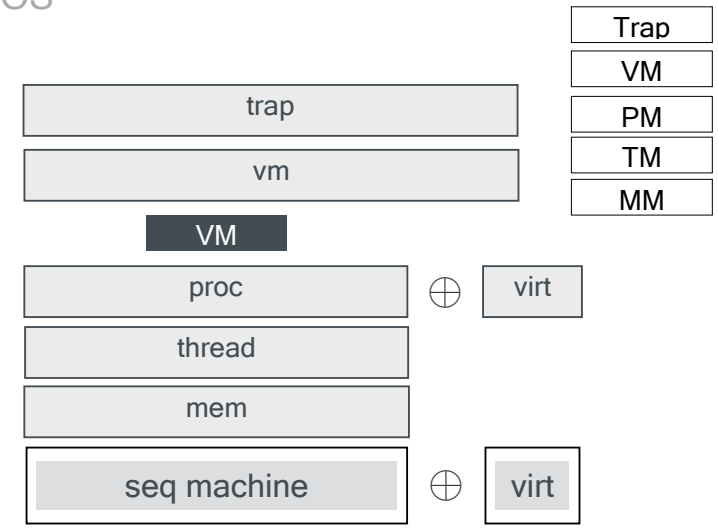
proc ⊕ virt

thread

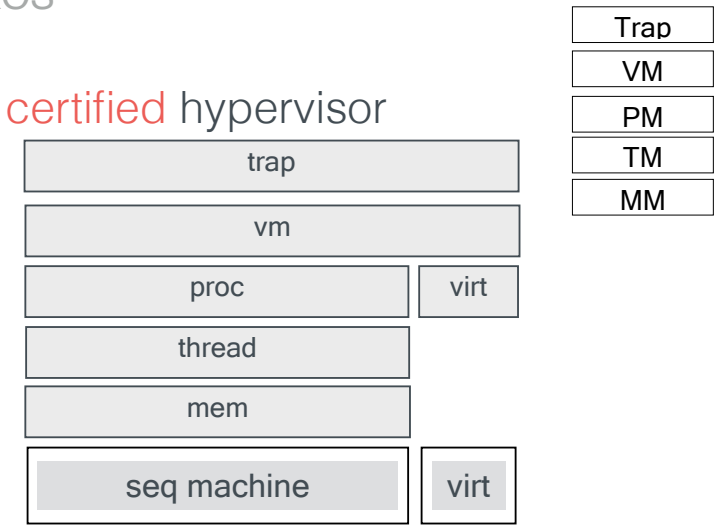
mem

seq machine ⊕ virt

mCertiKOS

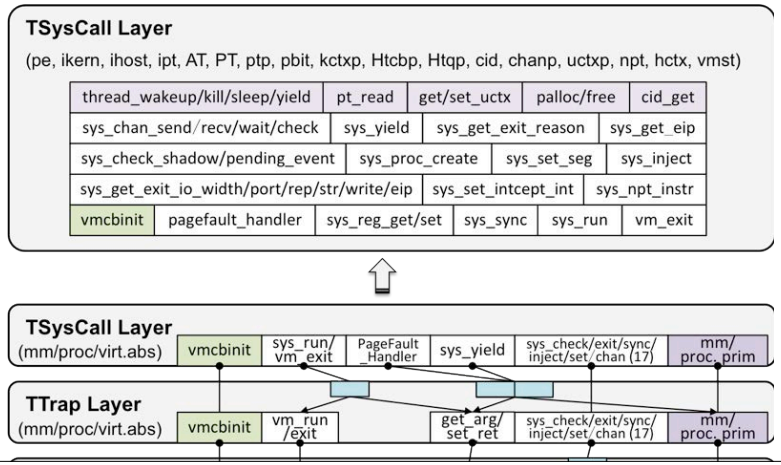


mCertiKOS



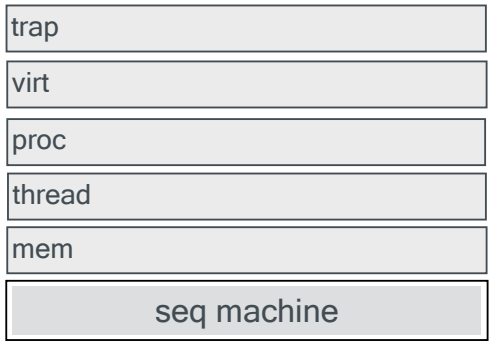
mCertiKOS 3k LOC
[POPL'15] 1 person year

Can **boot** Linux as a guest

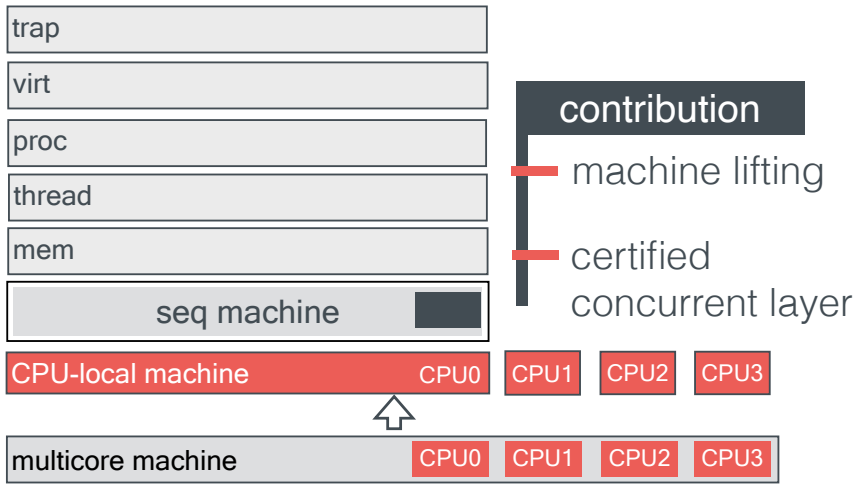


Concurrent Framework [OSDI'16, PLDI'18]

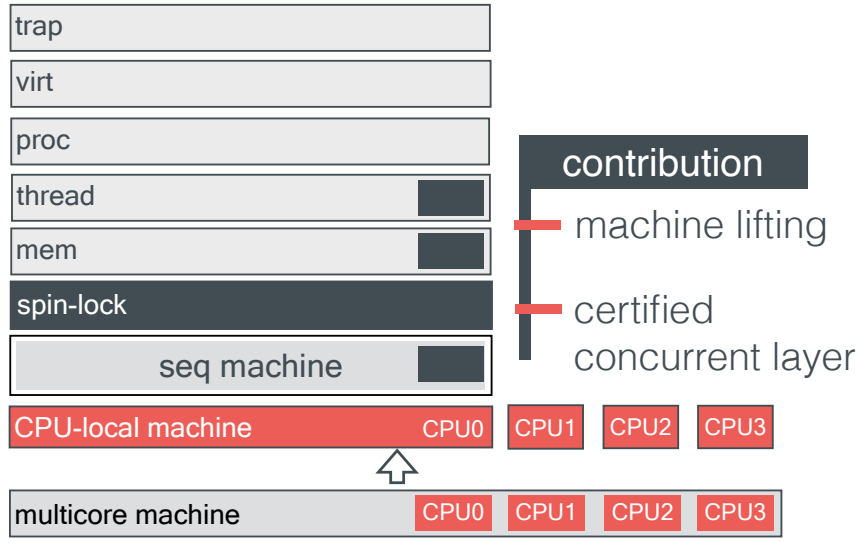
certified sequential kernel



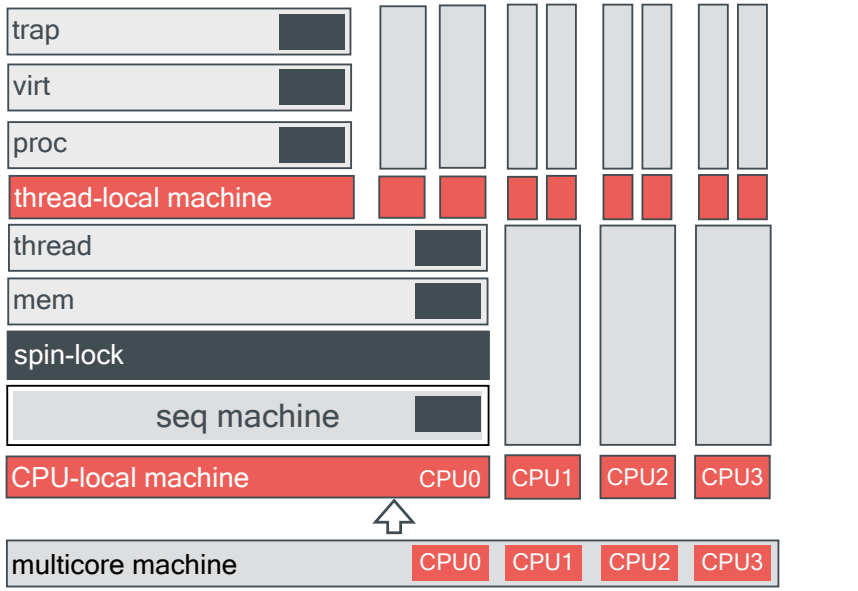
Concurrent Framework [OSDI'16, PLDI'18]



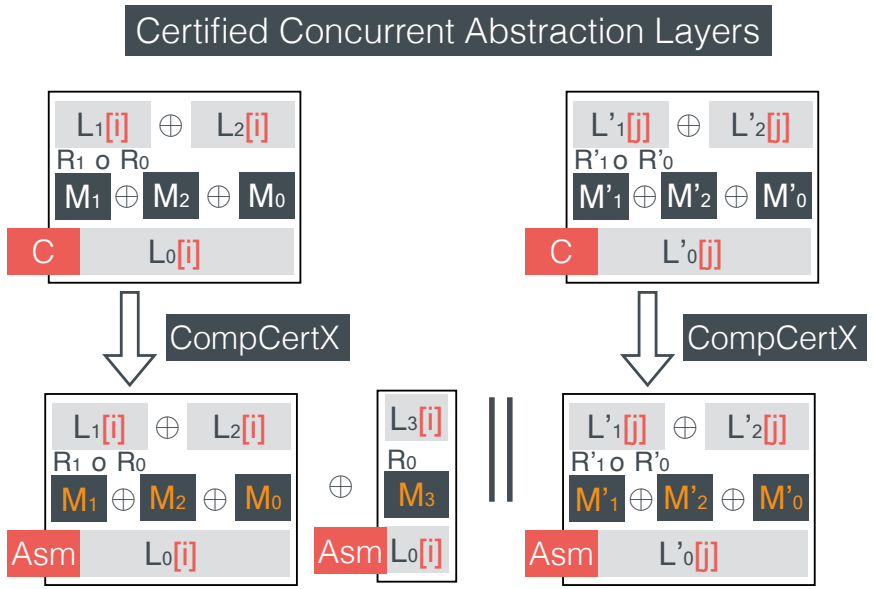
Concurrent Framework [OSDI'16, PLDI'18]



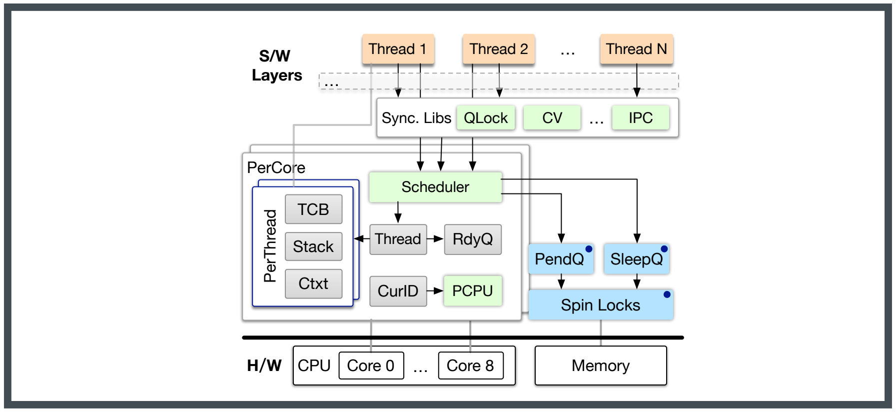
Concurrent Framework [OSDI'16, PLDI'18]



Contribution



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

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` ?

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

logical log `l`

2.FAI_t	2.FAI_t

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

logical log `l`

2.FAI_t	2.FAI_t	1.FAI_t

$2

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

logical log `l`

2.FAI_t	1.FAI_t	2.FAI_t

Strategies and Game Semantics

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

logical log `l`

$3

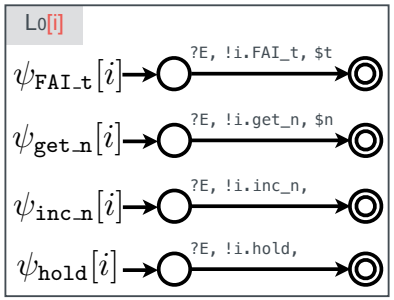
Strategies and Game Semantics

Lo[i]

```
extern uint FAI_t();

extern uint get_n();

extern void inc_n();

extern void hold();
```

Strategies and Game Semantics

Lo[i]

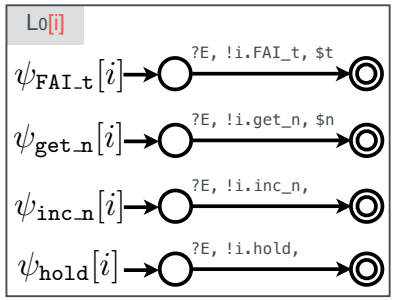


Strategies and Game Semantics

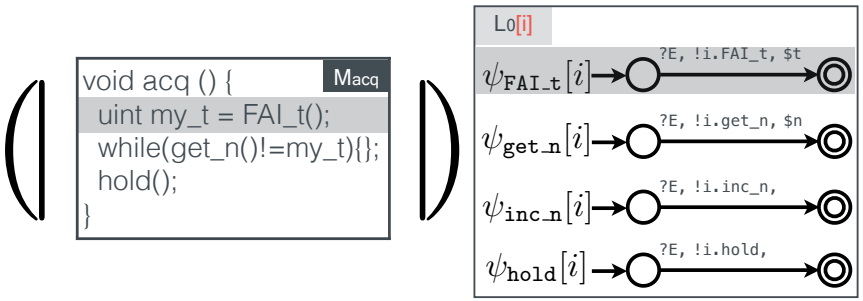
Macq

```
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
```

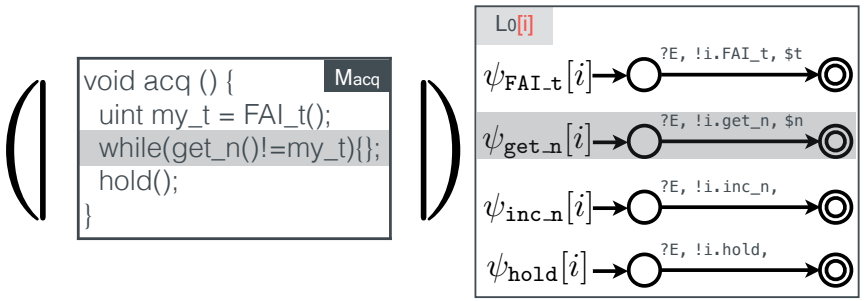
Lo[i]

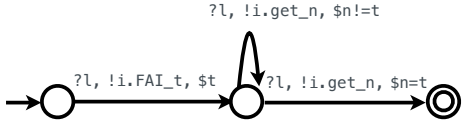


Strategies and Game Semantics

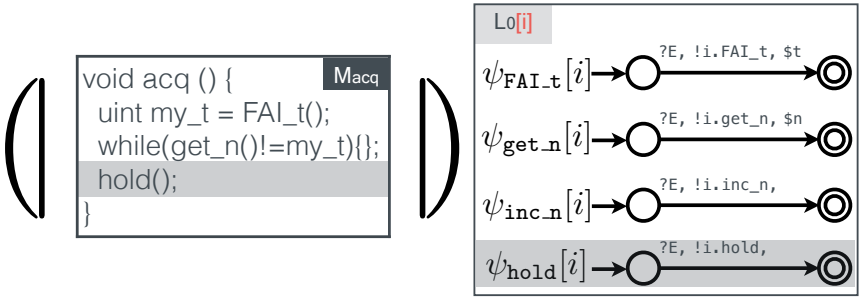


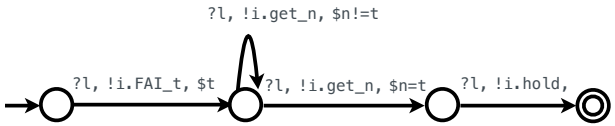
Strategies and Game Semantics





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

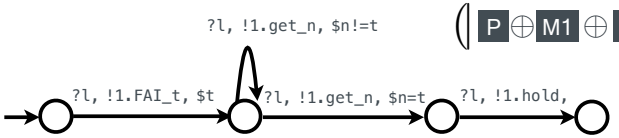
$(P \oplus M1 \oplus M2)\ L0[1]$

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

C

Strategies and Game Semantics

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




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

C

Strategies and Game Semantics

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


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

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

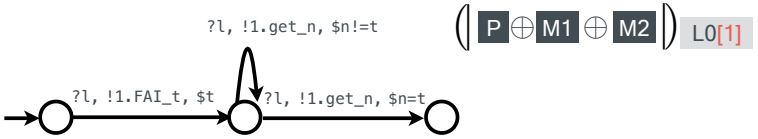
Strategies and Game Semantics

?l, !1.FAI_t, $t

logical log l

$2

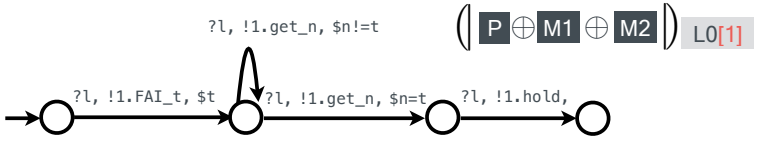
Strategies and Game Semantics




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

C

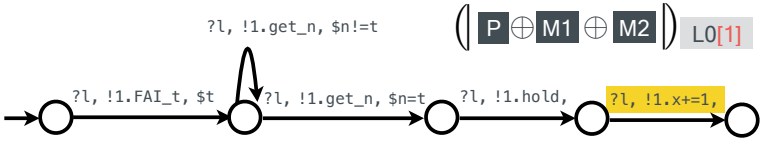
Strategies and Game Semantics




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

C

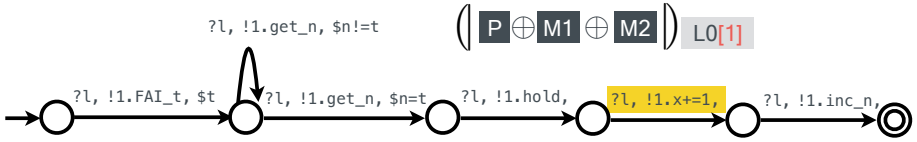
Strategies and Game Semantics




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

C

Strategies and Game Semantics

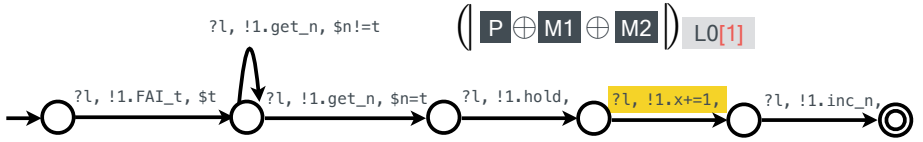


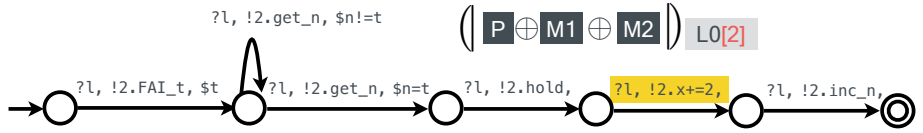

```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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(); }
```

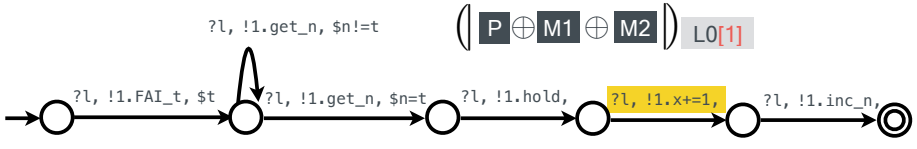
C

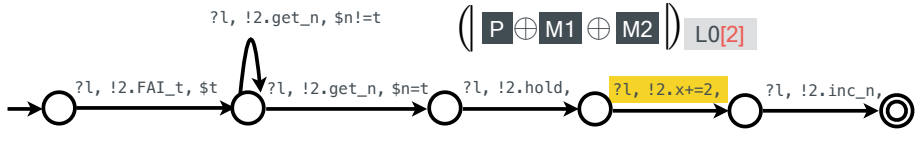
Strategies and Game Semantics



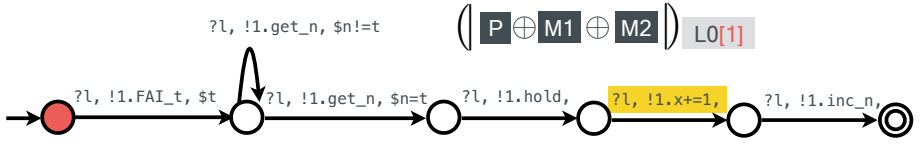


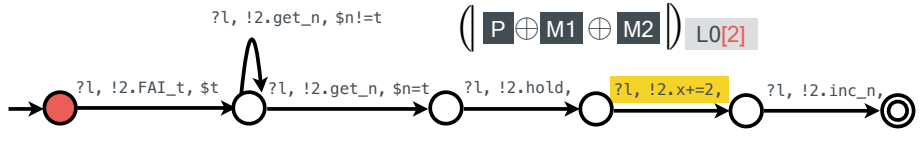
Strategies and Game Semantics





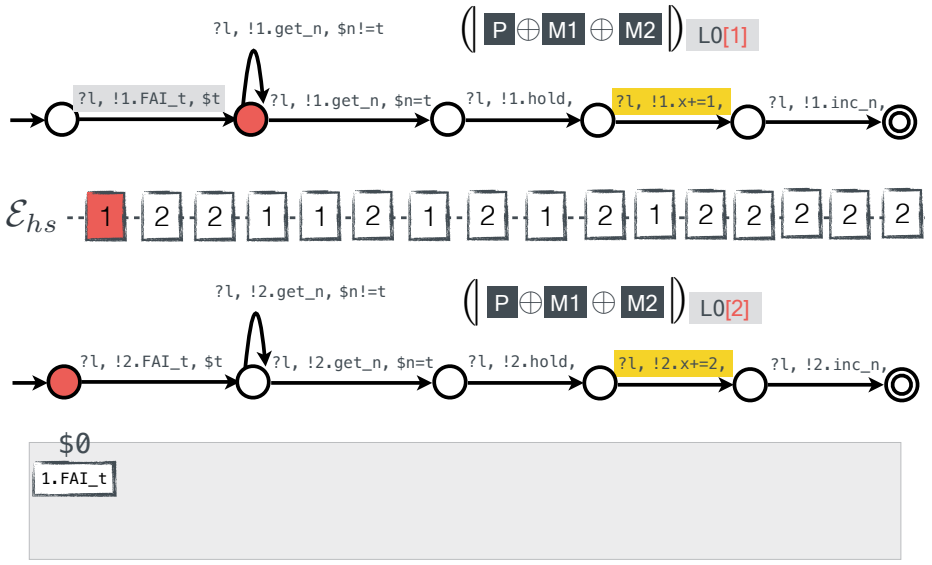
Strategies and Game Semantics





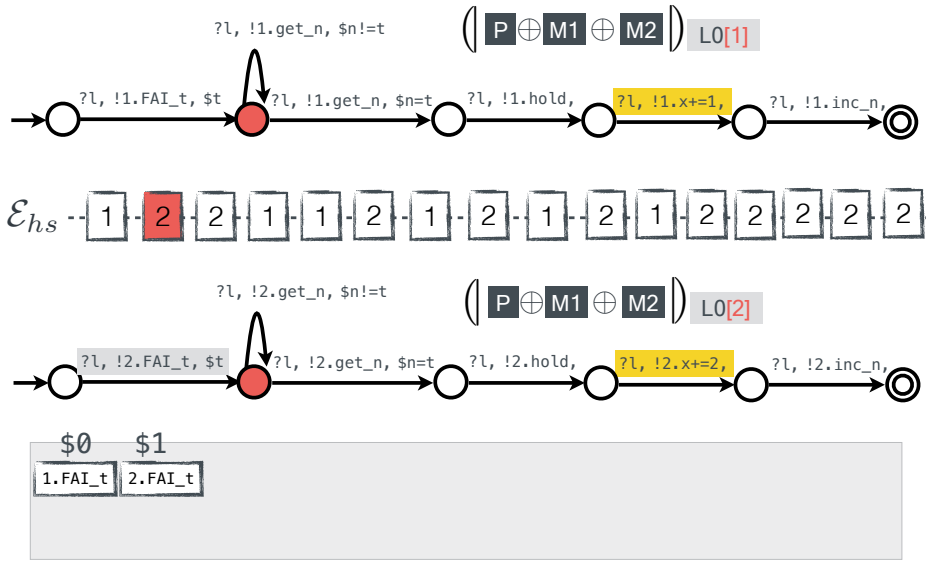
logical log l

Strategies and Game Semantics



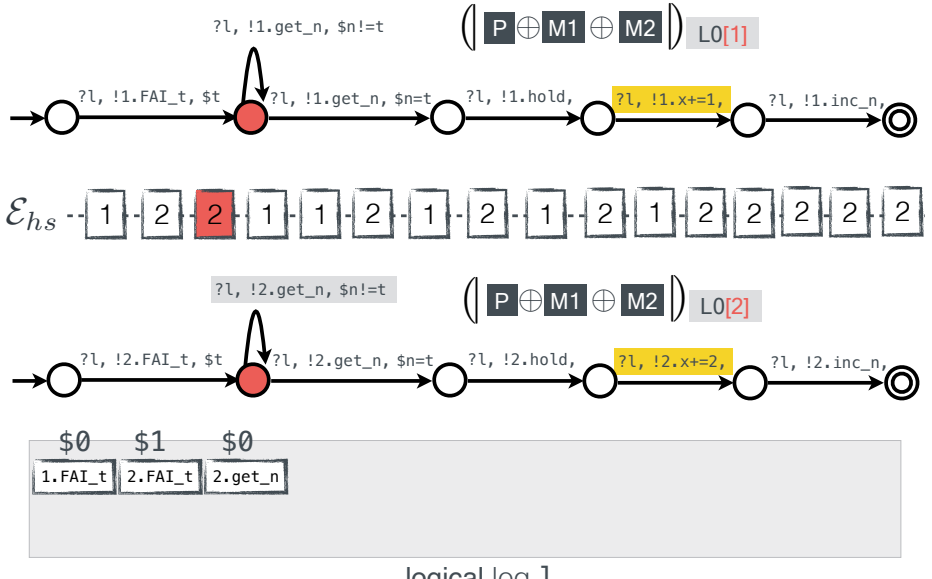
logical log `l`

Strategies and Game Semantics



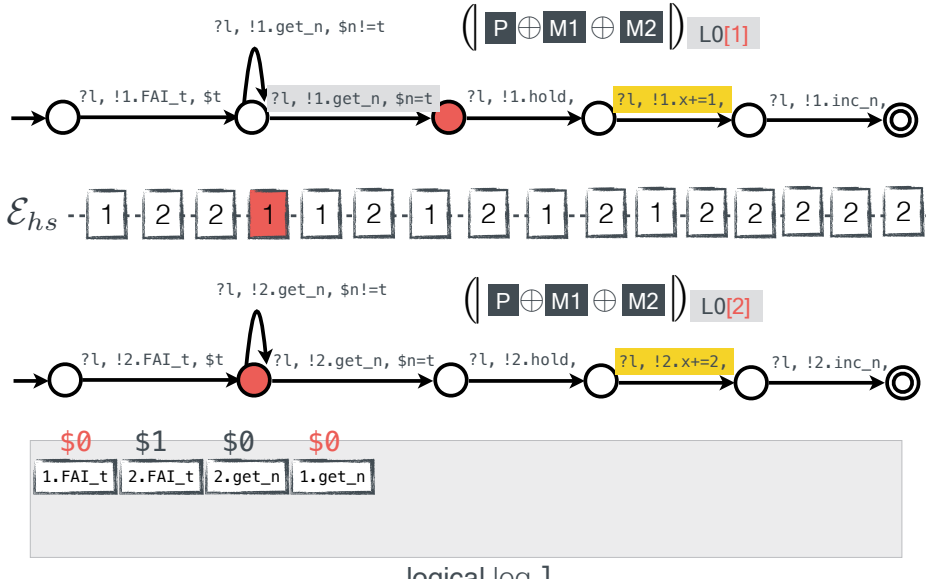
logical log `l`

Strategies and Game Semantics



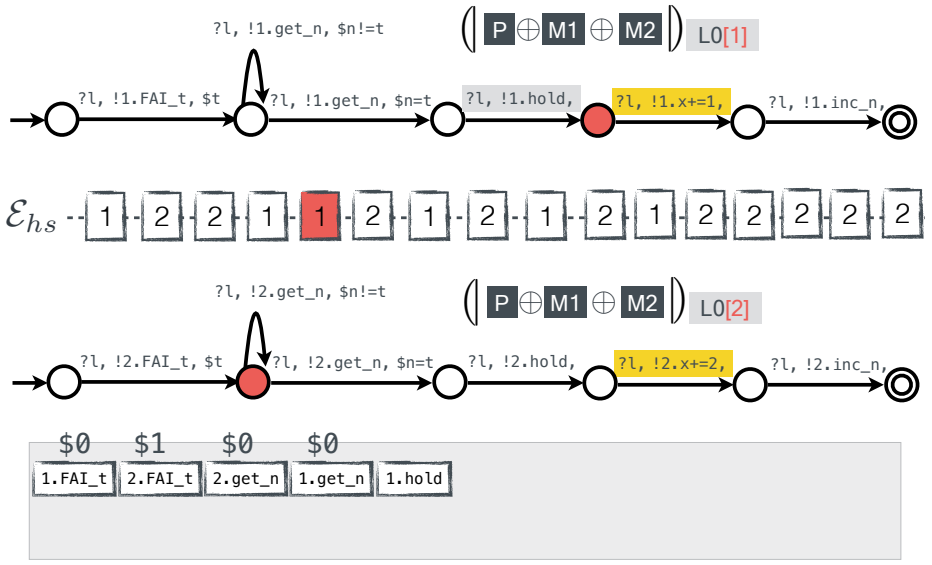
logical log `l`

Strategies and Game Semantics



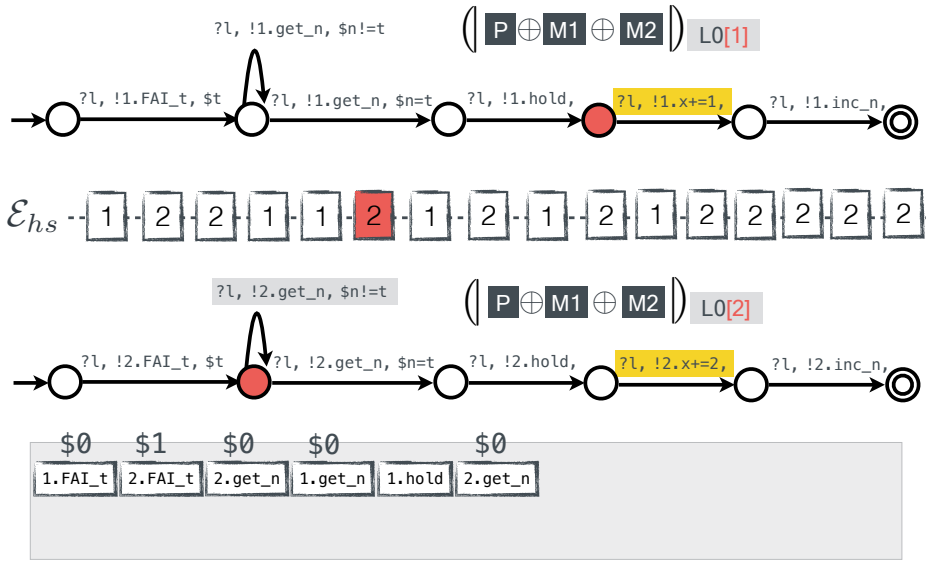
logical log `l`

Strategies and Game Semantics



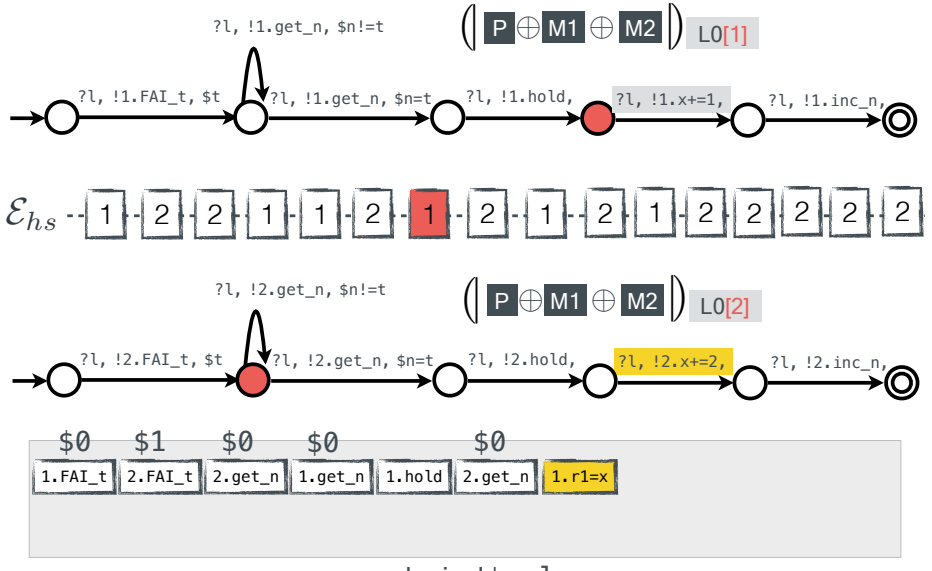
logical log `l`

Strategies and Game Semantics



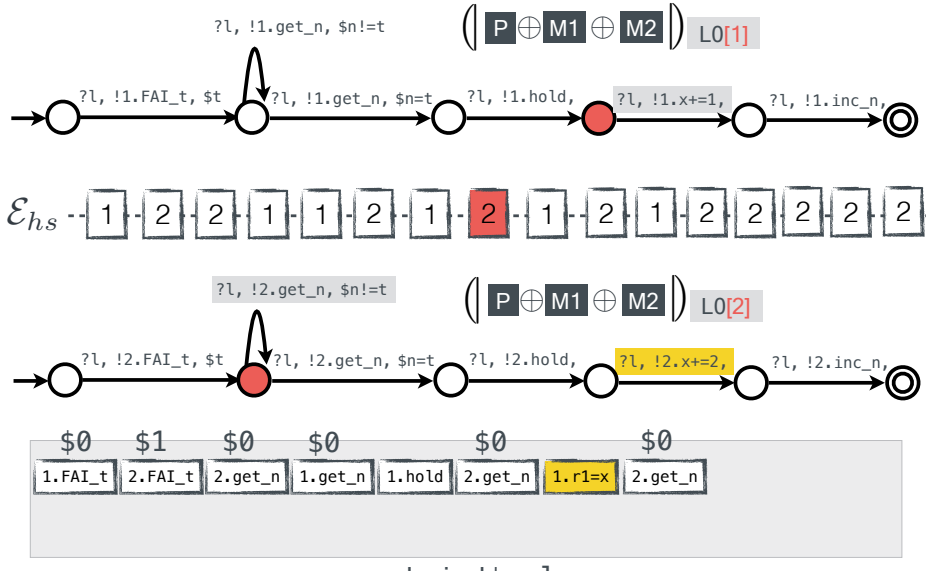
logical log `l`

Strategies and Game Semantics



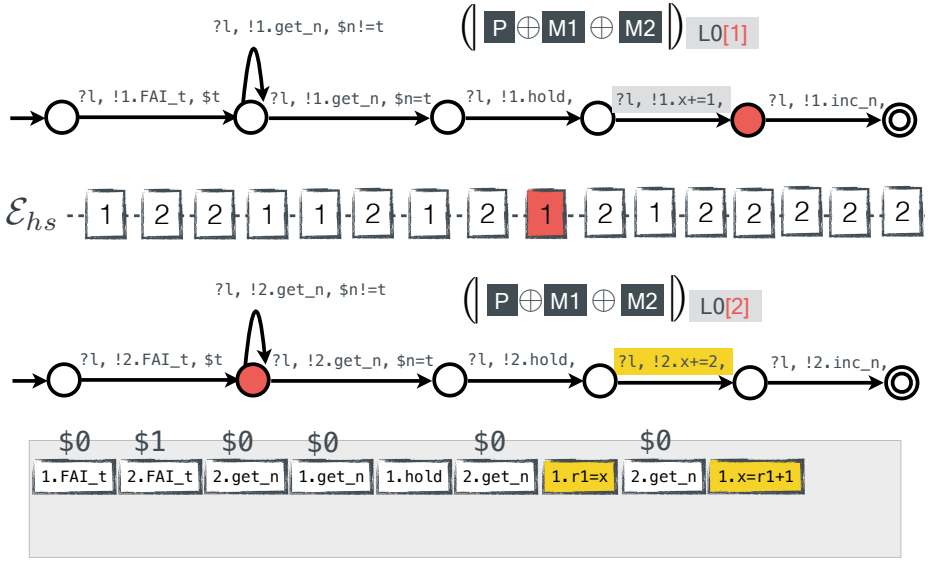
logical log `l`

Strategies and Game Semantics



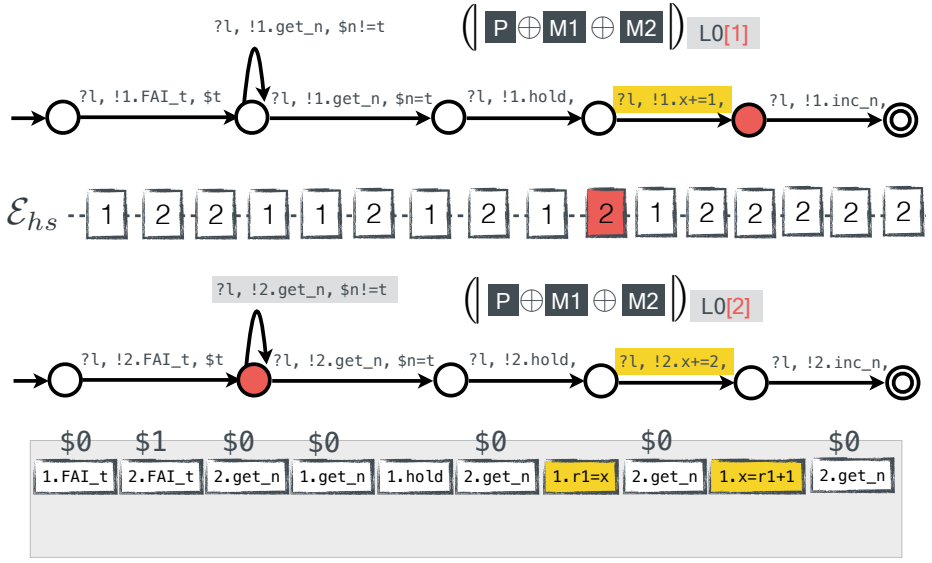
logical log `l`

Strategies and Game Semantics



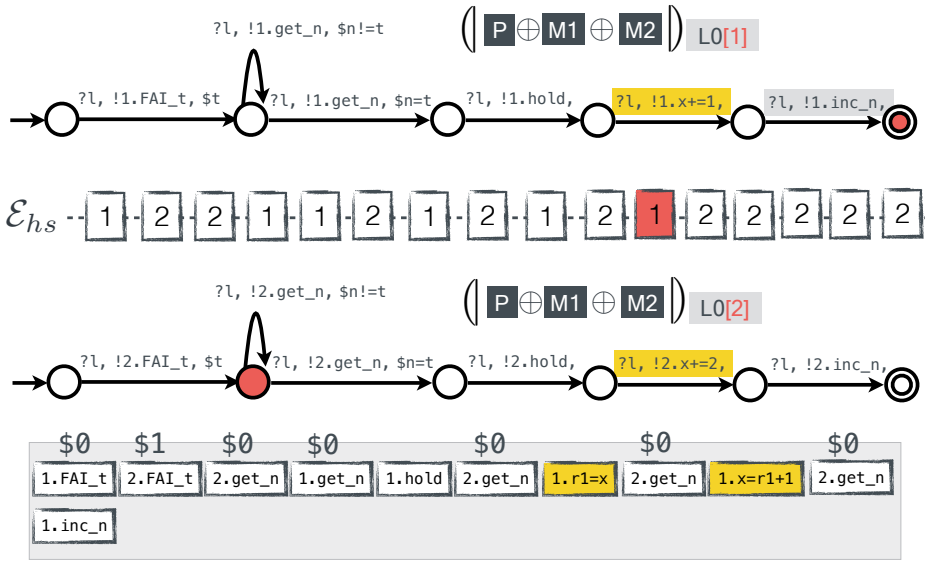
logical log l

Strategies and Game Semantics



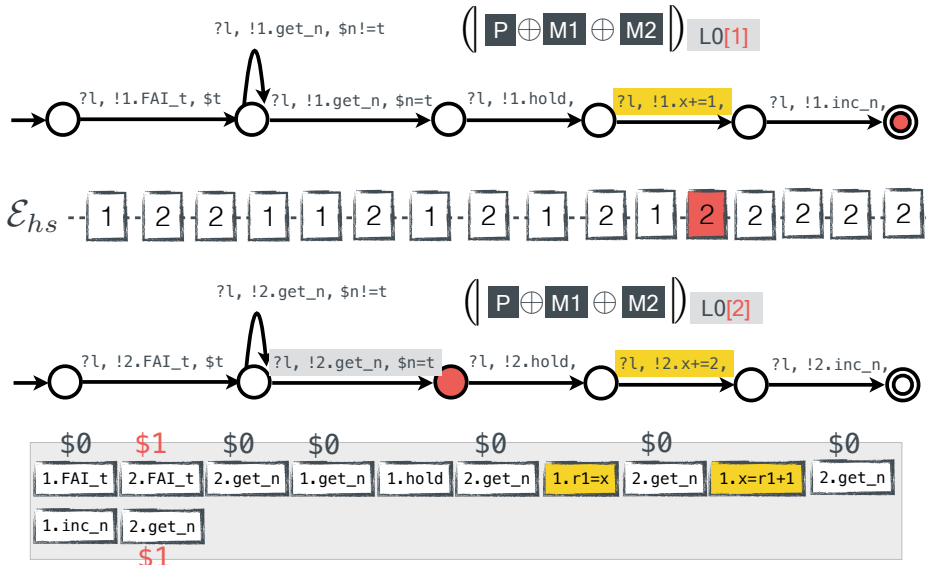
logical log l

Strategies and Game Semantics



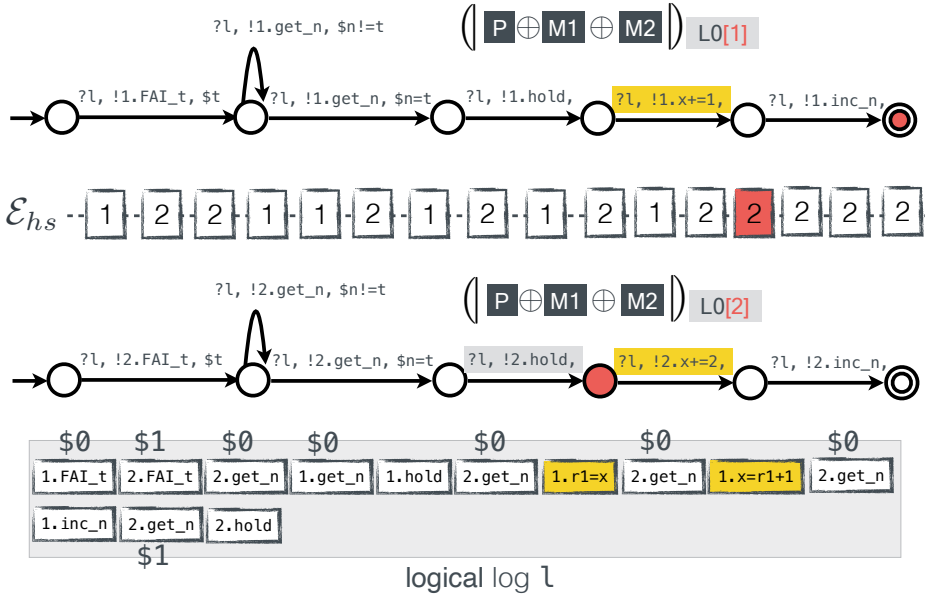
logical log l

Strategies and Game Semantics



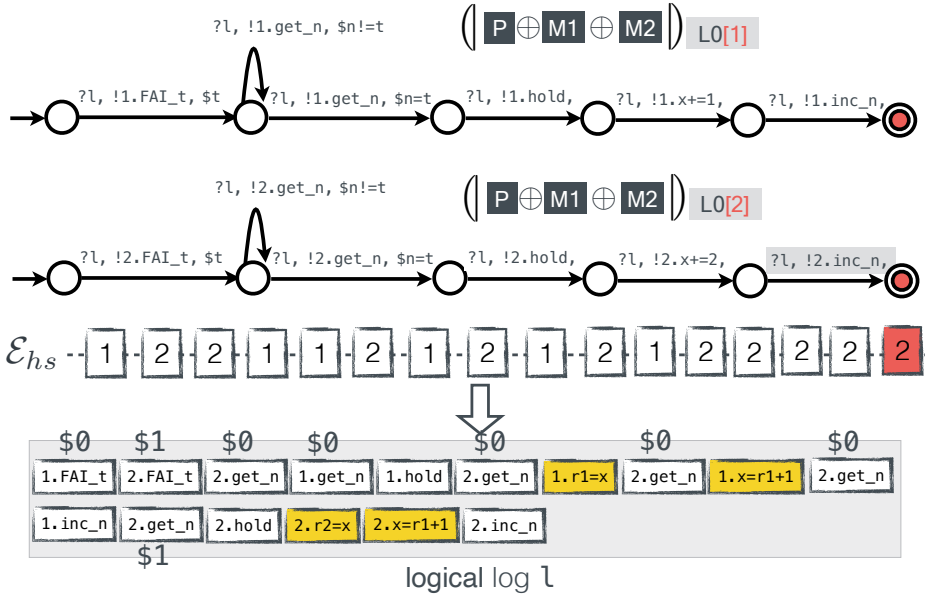
logical log l

Strategies and Game Semantics



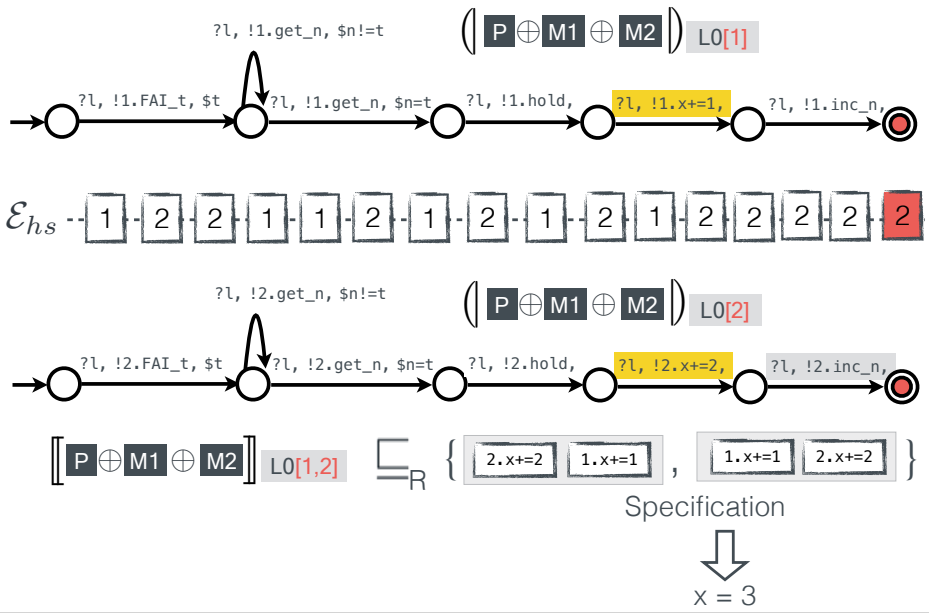
Strategies and Game Semantics

Strategies and Game Semantics

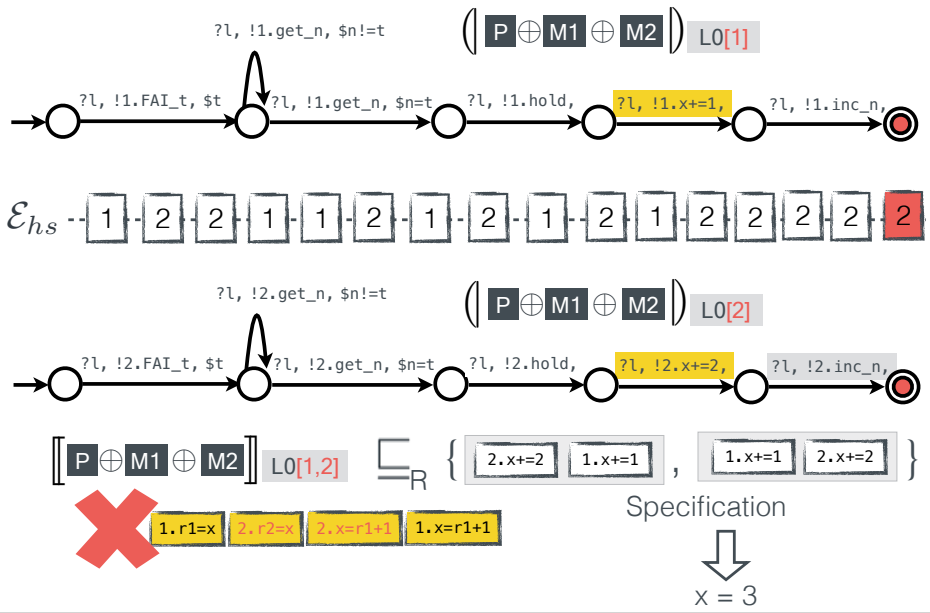


Strategies and Game Semantics

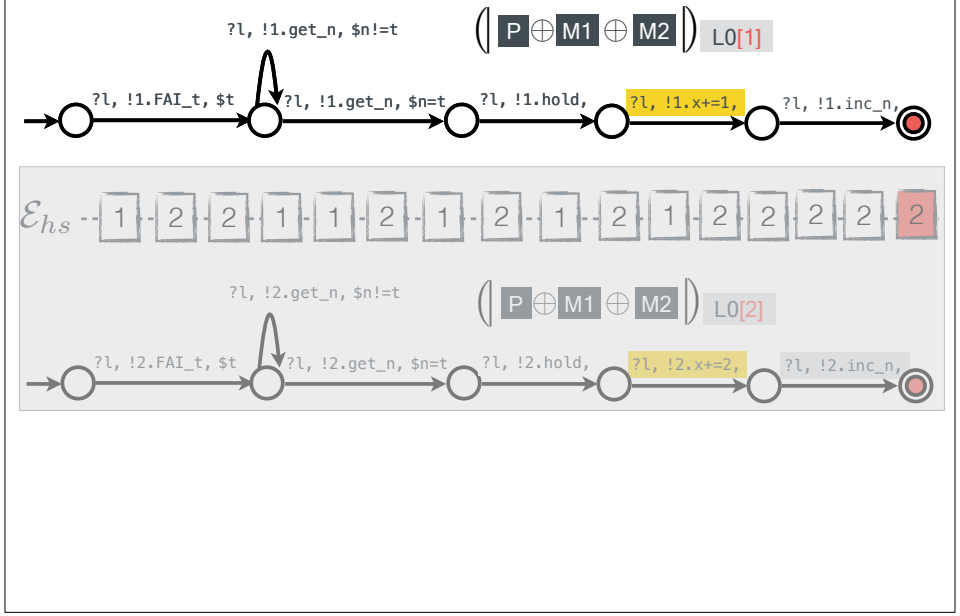
Strategies and Game Semantics



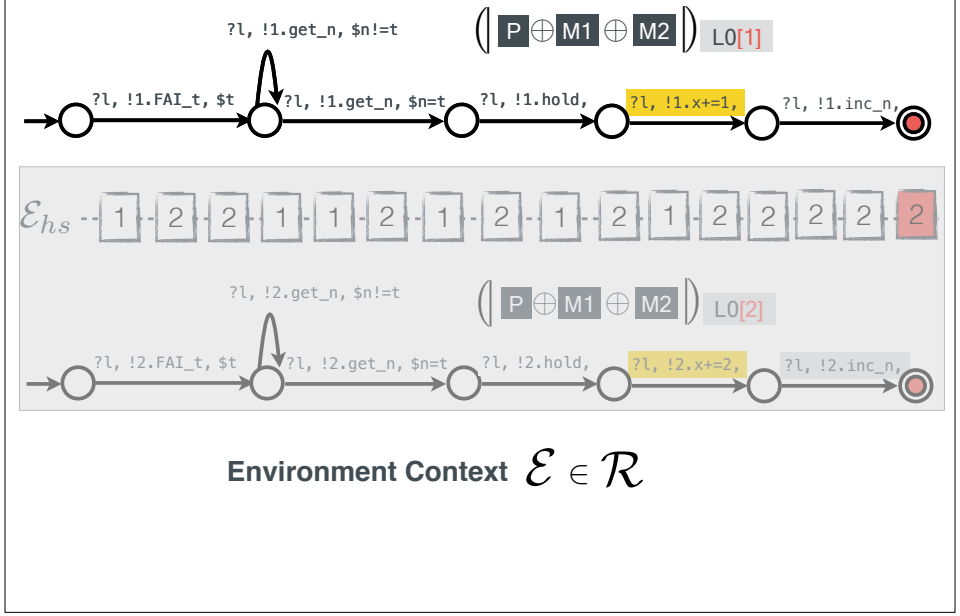
Strategies and Game Semantics



Strategy Refinement



Strategy Refinement



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

Strategy Refinement

$$\psi(\mathcal{E}, \mathtt{l0})$$

Strategy Refinement

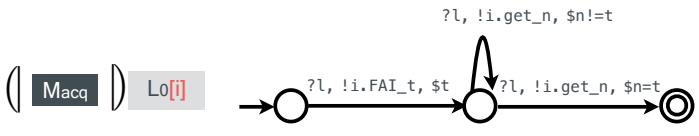
$$\psi \leq_R \psi'$$

$$\mathcal{E} \in \mathcal{R} \quad \mathtt{l0} \quad \Rightarrow \quad \psi(\mathcal{E}, \mathtt{l0})$$

R R R

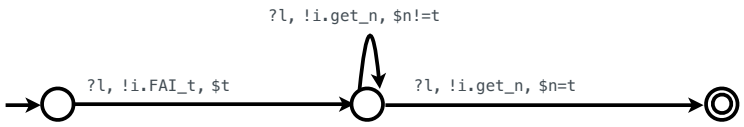
$$\mathcal{E}' \quad \mathtt{l'0} \quad \psi'(\mathcal{E}', \mathtt{l'0})$$

Strategy Refinement

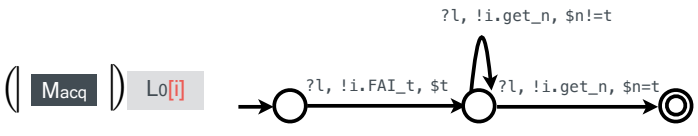


$\leq_R$

ψ_{acq}[i]



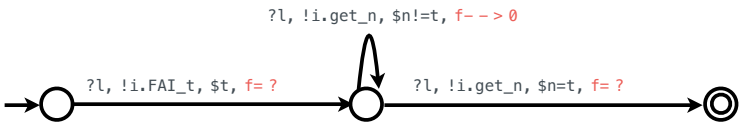
Strategy Refinement



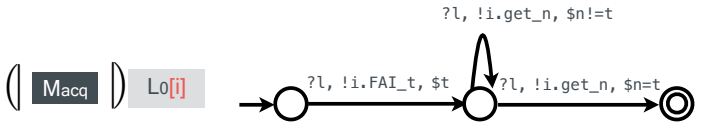
$\leq_R$

ψ_{acq}[i]

Add fuel (f) to prove liveness



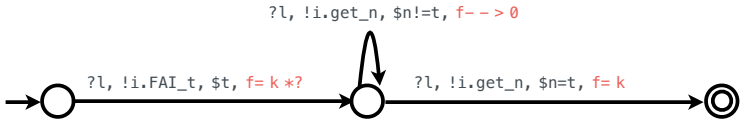
Strategy Refinement



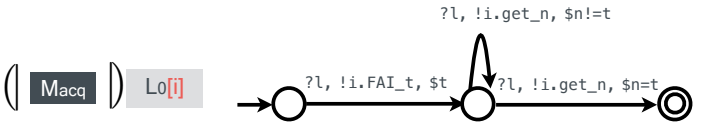
$\leq_R$

$\psi_{acq}[i]$

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



Strategy Refinement

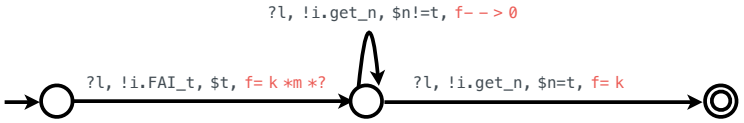


$\leq_R$

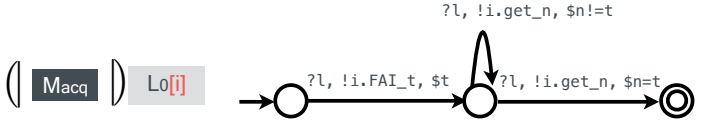
$\psi_{acq}[i]$

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

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



Strategy Refinement



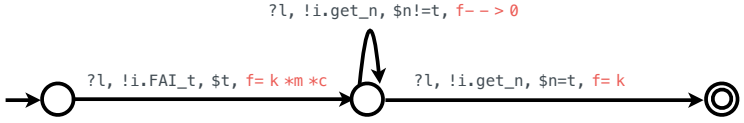
$\leq_R$

$\psi_{acq}[i]$

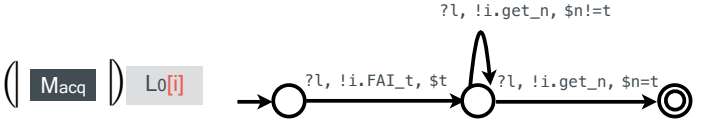
$\mathcal{R}_{j\neq i}$: will release lock within k steps

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

$\mathcal{R}_{cpu}$: #CPU = c is bounded



Strategy Refinement



$\leq_R$

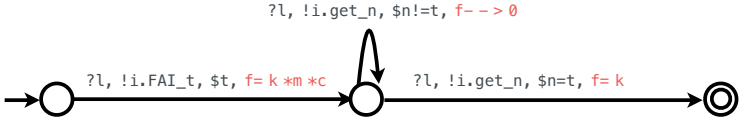
$\psi_{acq}[i]$

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

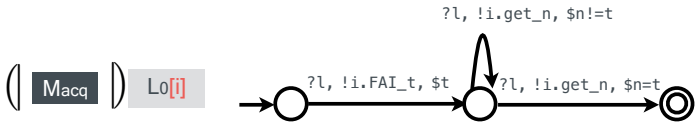
$\mathcal{R}_{hs}$: (fairness) each CPU will be rescheduled within m steps

$\mathcal{R}_{cpu}$: #CPU = c is bounded

mutual exclusion?



Strategy Refinement



$\langle$ Macq $\rangle$ L0[i]

$\leq_R$

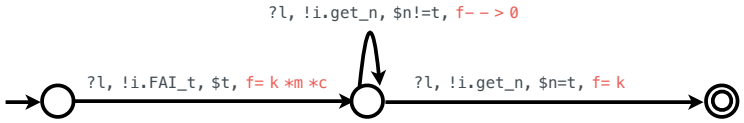
ψ_{acq}[i]

$\mathcal{R}_{j\neq i}$: 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}

mutual exclusion?



Certified Concurrent Abstraction Layer

$\langle$ Macq $\rangle$ L0[i]

$\leq_R$

ψ_{acq}[i]

ψ_{acq}[i]
R
Macq
L0[i]

Certified Concurrent Abstraction Layer

$\langle$ Macq $\rangle$ L0[i]

$\leq_R$

ψ_{acq}[i]

ψ_{acq}[i]
R
Macq
L0[i]

ψ_{rel}[i]
R
Mrel
L0[i]

$\langle$ Mrel $\rangle$ L0[i]

$\leq_R$

ψ_{rel}[i]

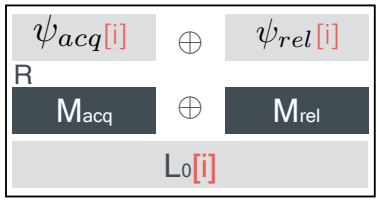
Horizontal Composition

ψ_{acq}[i]
R
Macq
L0[i]

$\oplus$

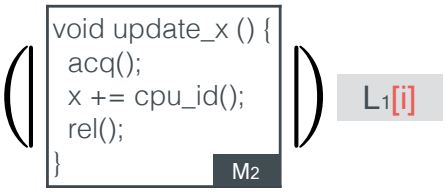
ψ_{rel}[i]
R
Mrel
L0[i]

Horizontal Composition



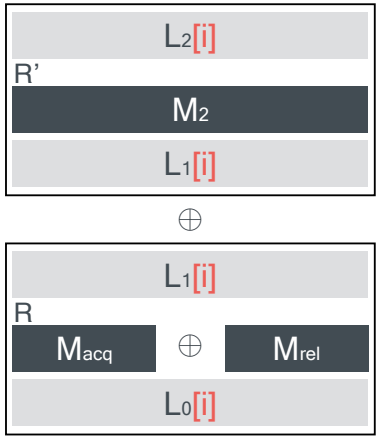
Certified Concurrent Abstraction Layer

Certified Concurrent Abstraction Layer

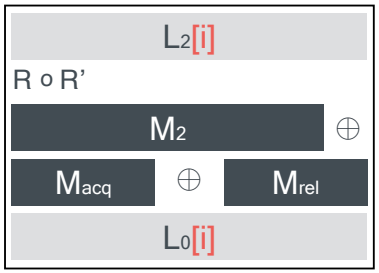


Certified Concurrent Abstraction Layer

Vertical Composition



Vertical Composition

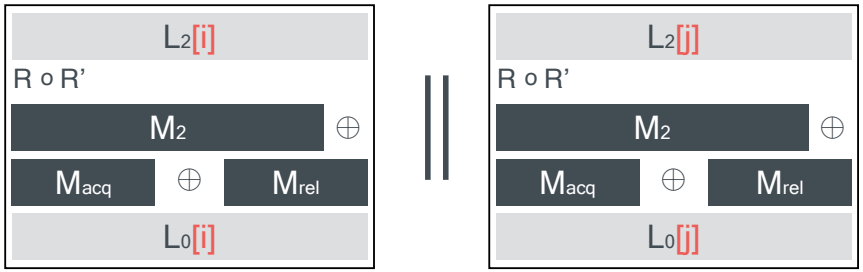


Parallel Composition

$\mathcal{R}_{j \neq i}$: 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

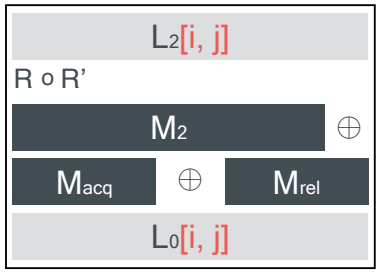
$\mathcal{R}_{cpu}$: #CPU = c < 2^{32}

Parallel Composition

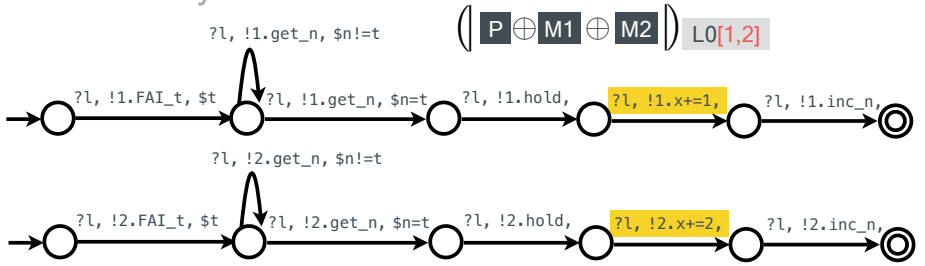
$\mathcal{R}_{hs}$: (fairness) each CPU will be rescheduled within m steps

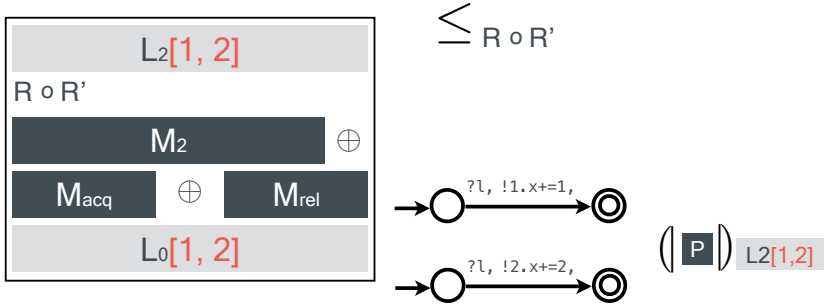
$\Rightarrow$ $\mathcal{E}_{hs}$

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

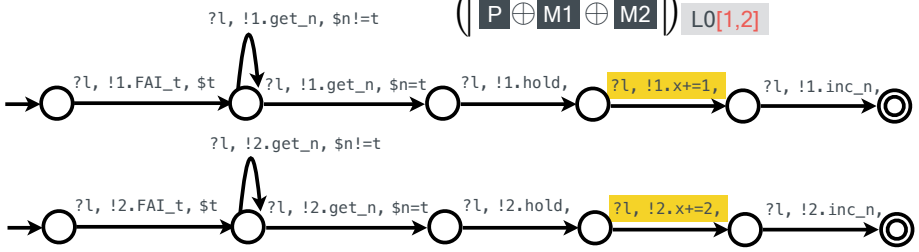


Case Study

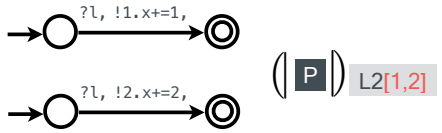




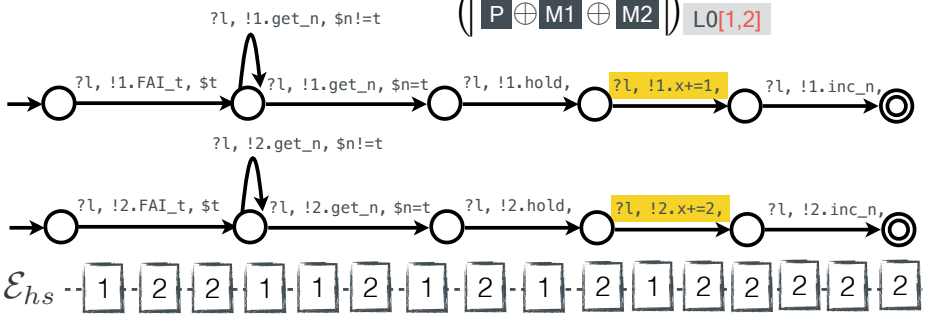
Case Study

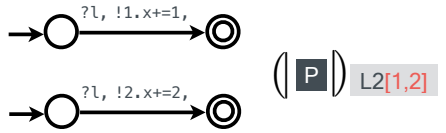


≤ R o R'

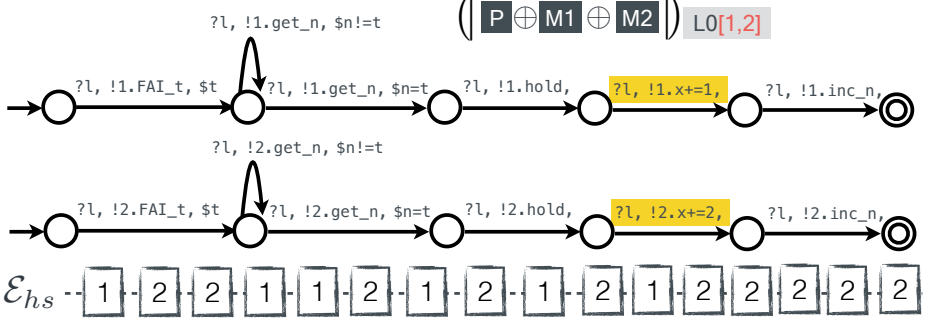


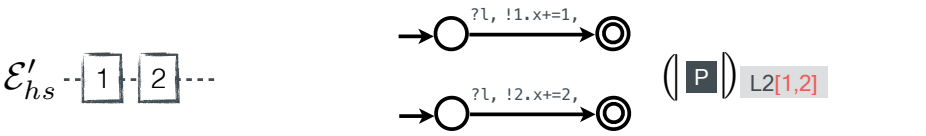
Case Study



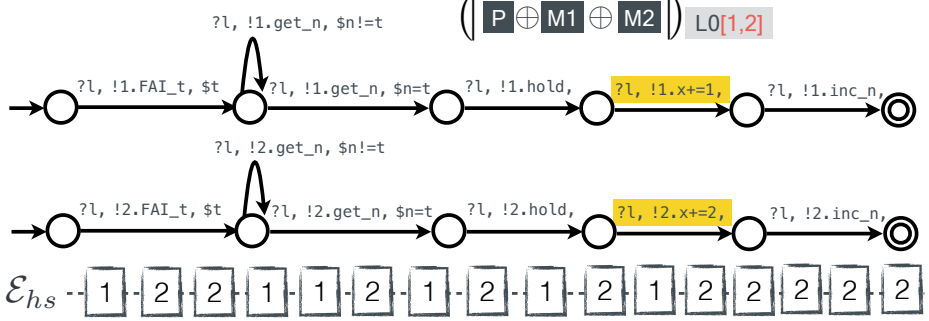


Case Study

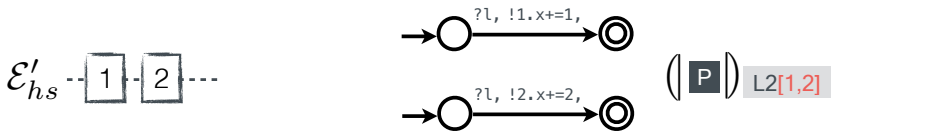




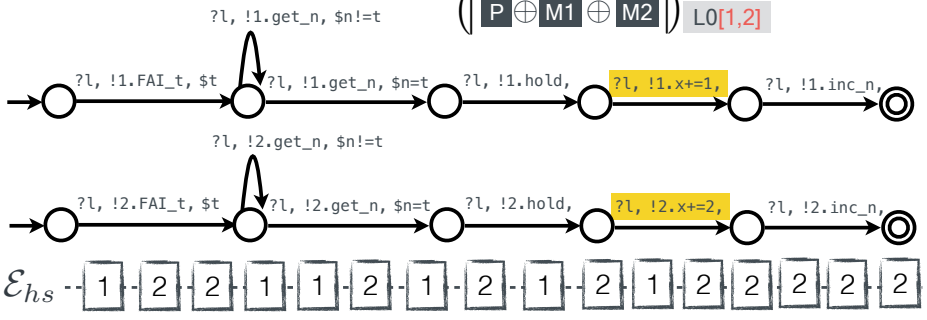
Case Study



1.FAI_t 2.FAI_t 2.get_n 1.get_n 1.hold 2.get_n 1.r1=x 2.get_n 1.x=r1+1 2.get_n
1.inc_n 2.get_n 2.hold 2.r2=x 2.x=r1+1 2.inc_n



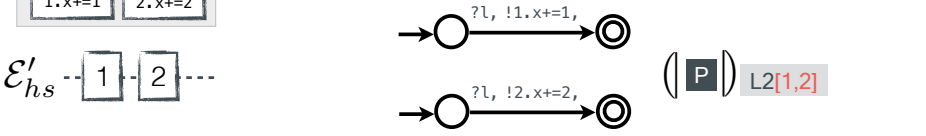
Case Study



1.FAI_t 2.FAI_t 2.get_n 1.get_n 1.hold 2.get_n 1.r1=x 2.get_n 1.x=r1+1 2.get_n
1.inc_n 2.get_n 2.hold 2.r2=x 2.x=r1+1 2.inc_n

R o R'

1.x+=1 2.x+=2



Soundness

$[\![\text{P} \oplus \text{M1} \oplus \text{M2}]\!]$ L0[1,2] $\sqsubseteq_{\text{R o R'}}$ $[\![\text{P}]\!]$ L2[1,2]

Soundness

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

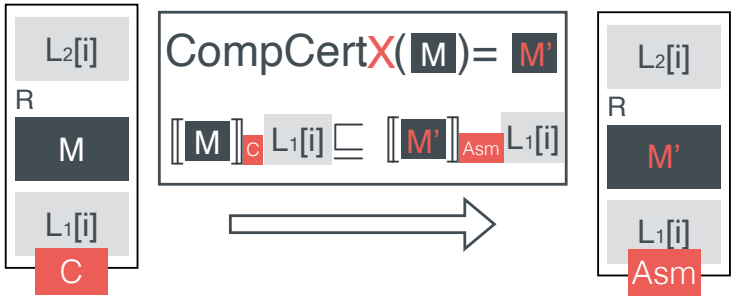
$\mathcal{R}_{cpu}$: #CPU = c < 2^{32}

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

= { 2.x+=2 1.x+=1 , 1.x+=1 2.x+=2 }

QED

CompCertX



Assembly Layers

Horizontal Composition

Vertical Composition

Parellel Composition

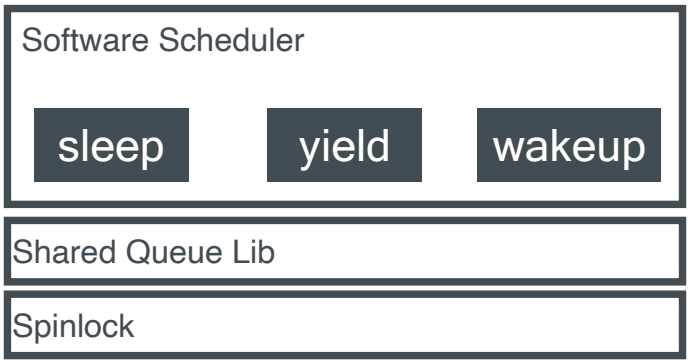
Software Scheduler

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

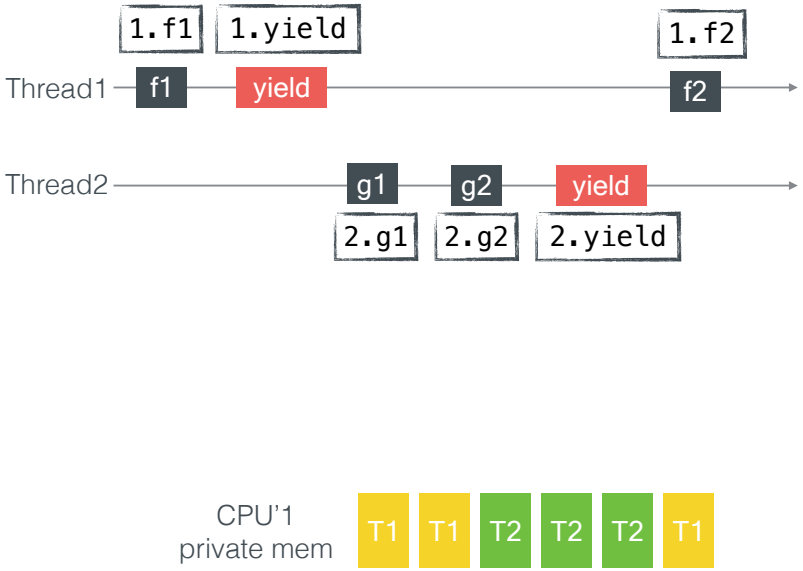
Shared Queue Lib

Spinlock

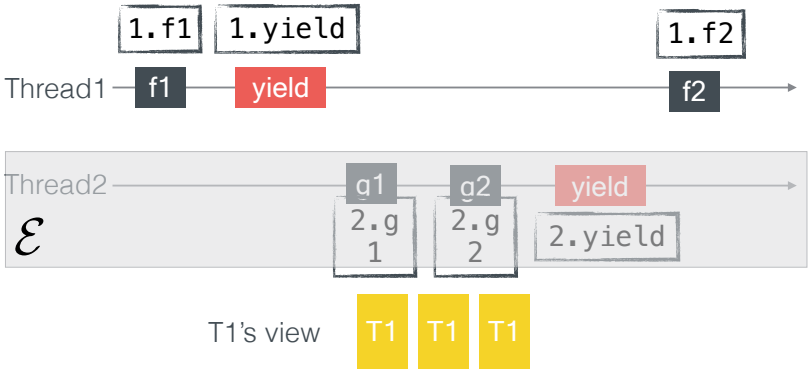
Software Scheduler



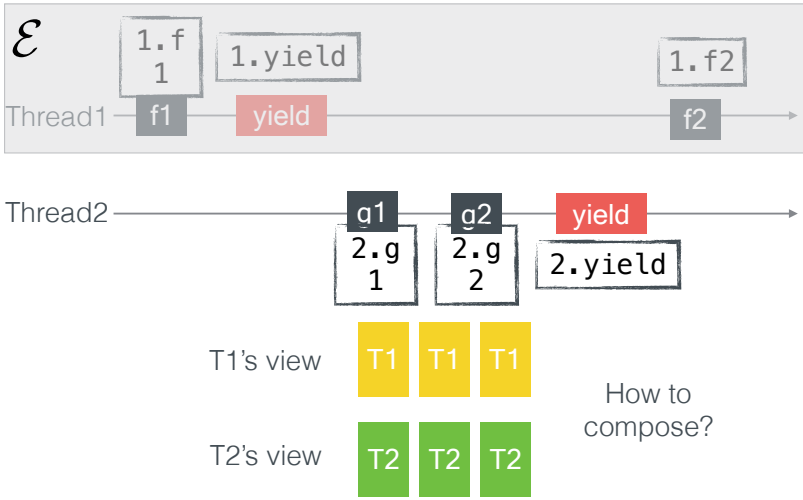
Software Scheduler



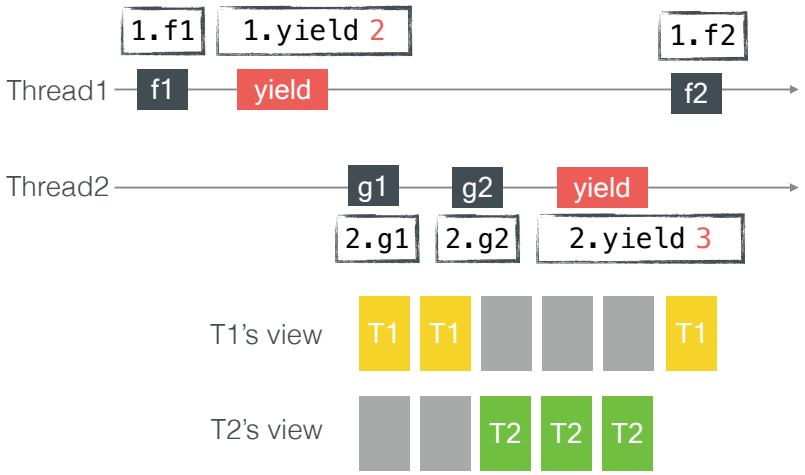
Software Scheduler



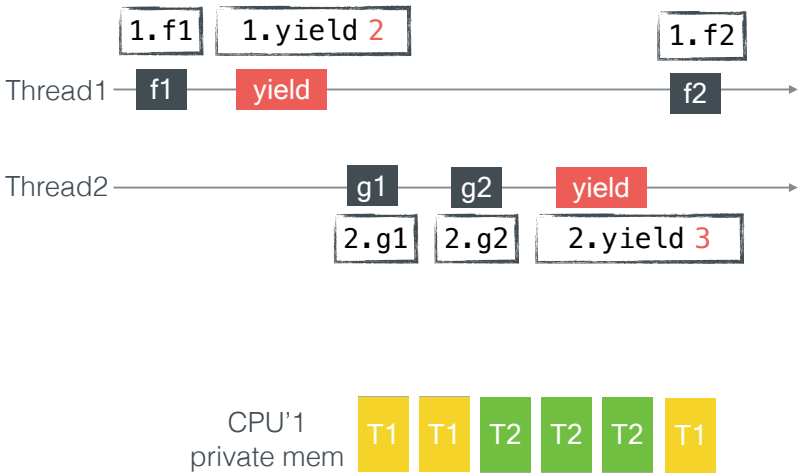
Software Scheduler



Algebraic Memory Model



Algebraic Memory Model



CompCertX

CompCertX

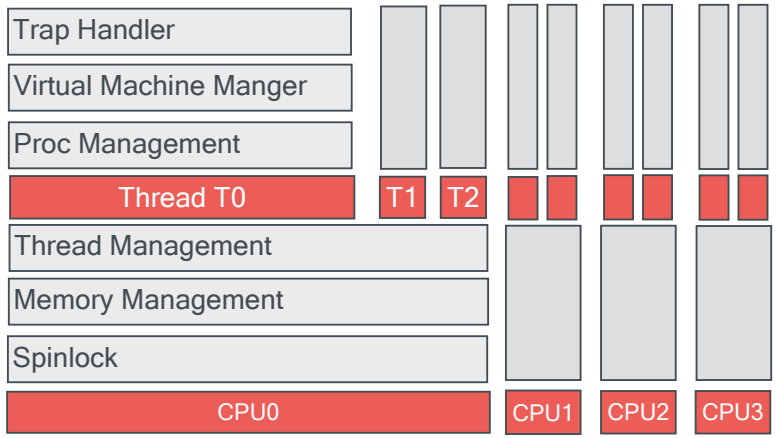
\+

Algebraic
Memory Model

=

Thread-safe
Verified Compiler

Verification of a Concurrent OS Kernel



Layer	Refinement proof	Code verification	Source code	Proof linking
--		--	--	MulticoreLinking

Source code linking	CertiKOS instacne for extraction	Proof linking
CertiKOS	CertiKOS Instance	CertiKOS_correct

Layers for per-thread

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

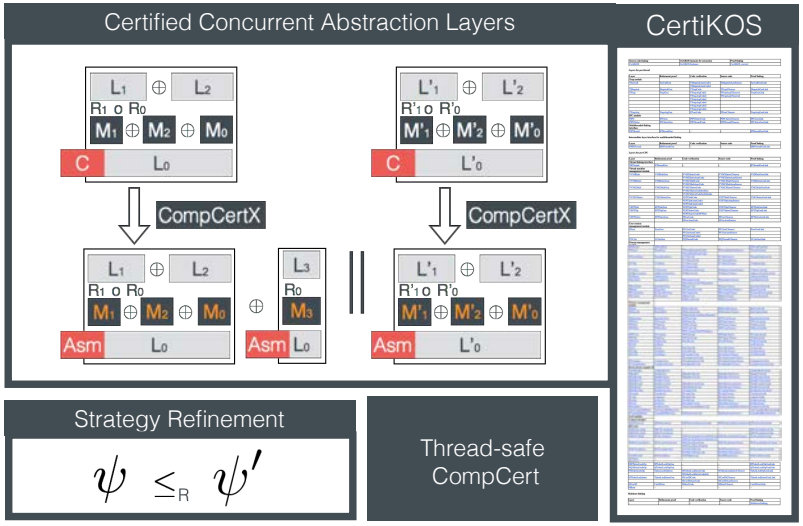
Intermediate layer interface for multithreaded linking

Layer	Refinement proof	Code verification	Source code	Proof linking
PHBThread	HBThreadGen	--	--	HBThreadGenLink

Layers for per-CPU

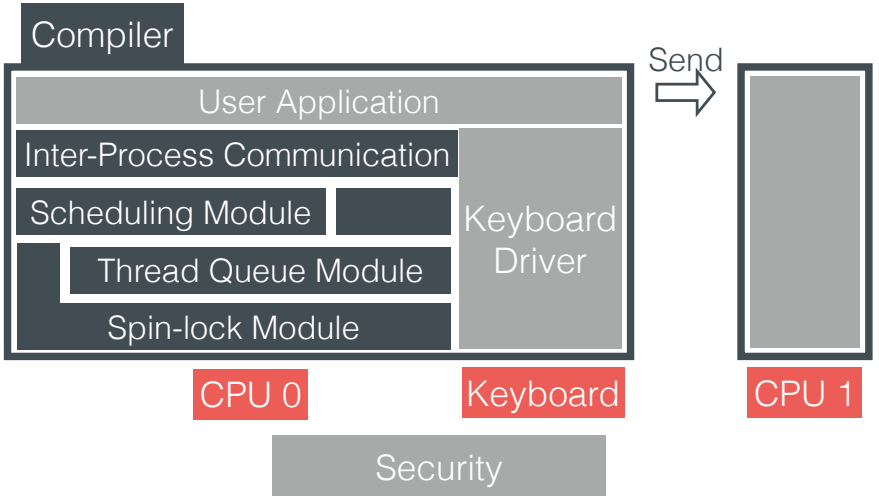
Layer	Refinement proof	Code verification	Source code	Proof linking
Thread linking interface				
PBThread	BThreadGen	--	--	BThreadGenLink

Contribution Summary



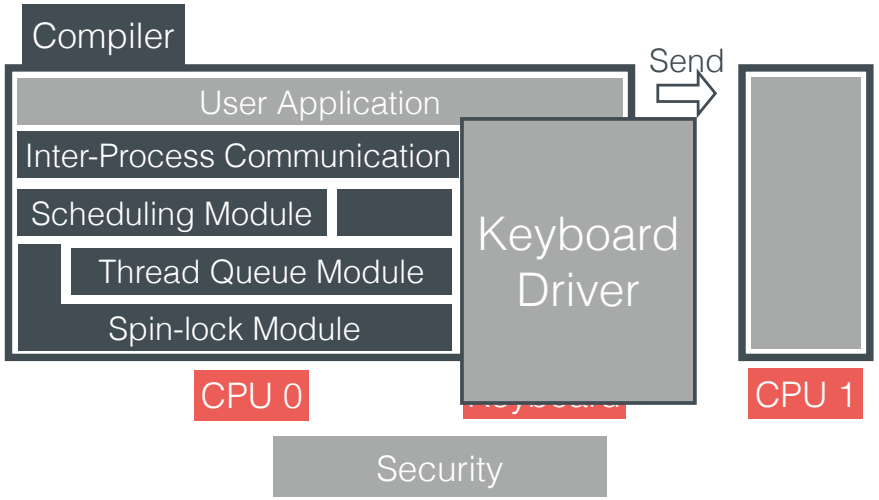
Case Study

Build a Certified System

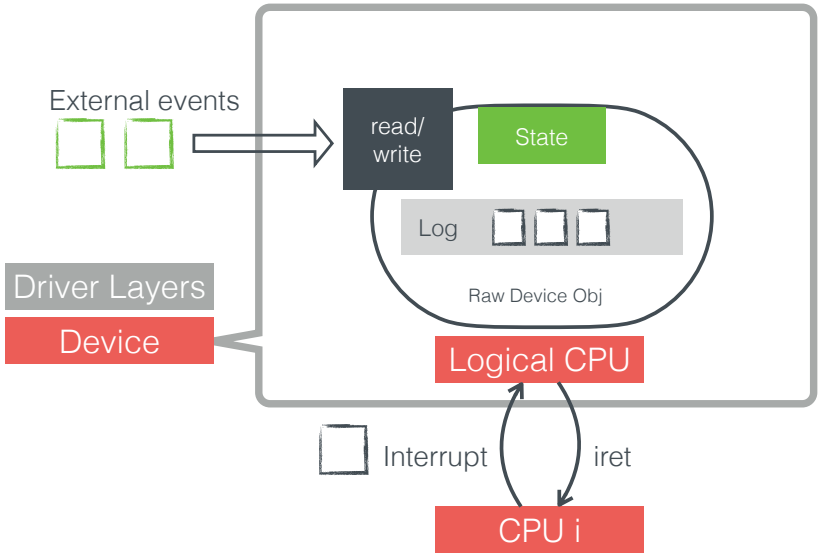


Case Study

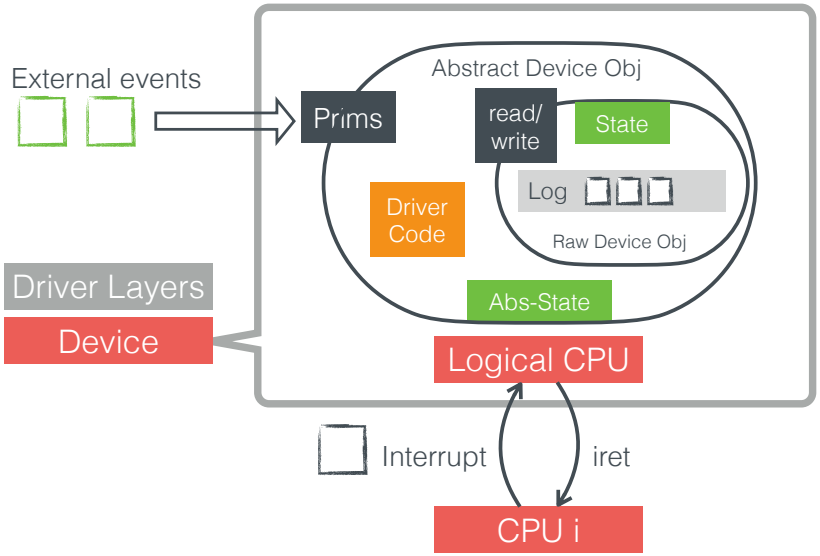
Build a Certified System



Device Driver [PLDI16'a]

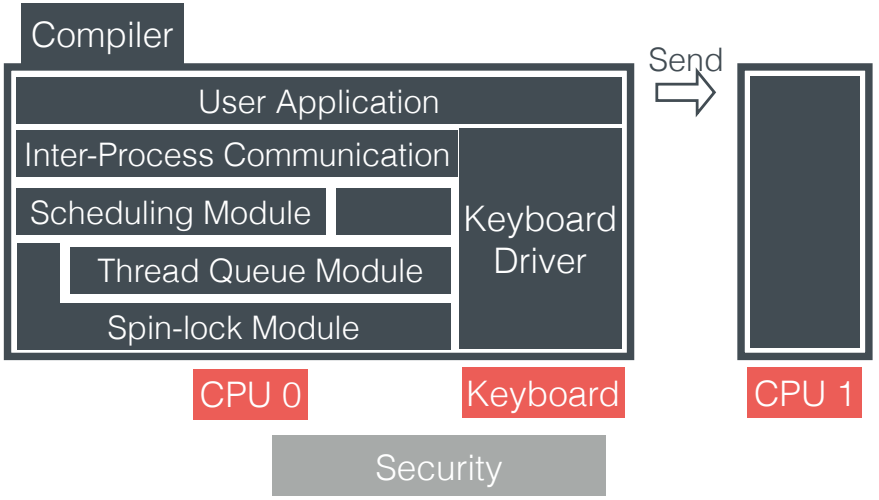


Device Driver [PLDI16'a]



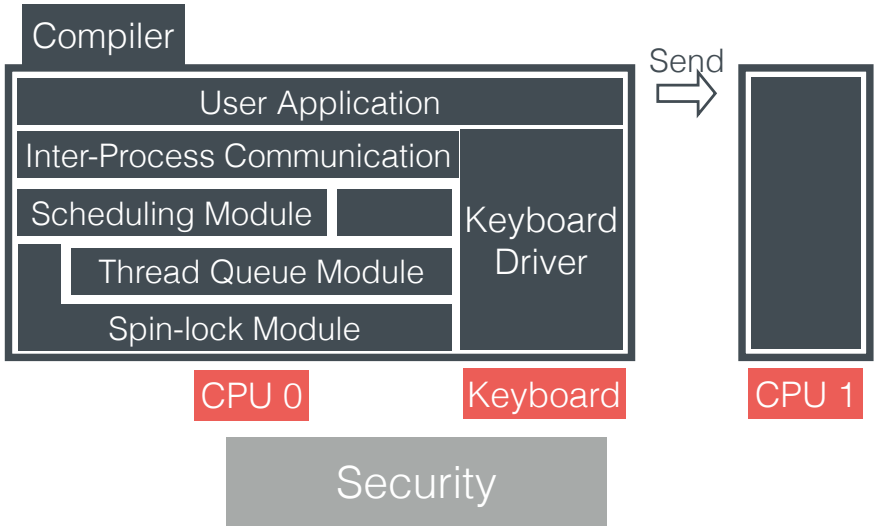
Case Study

Build a Certified System

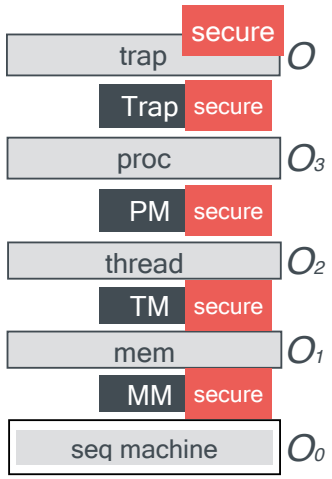


Case Study

Build a Certified System



End-to-End Security [PLDI16'b]



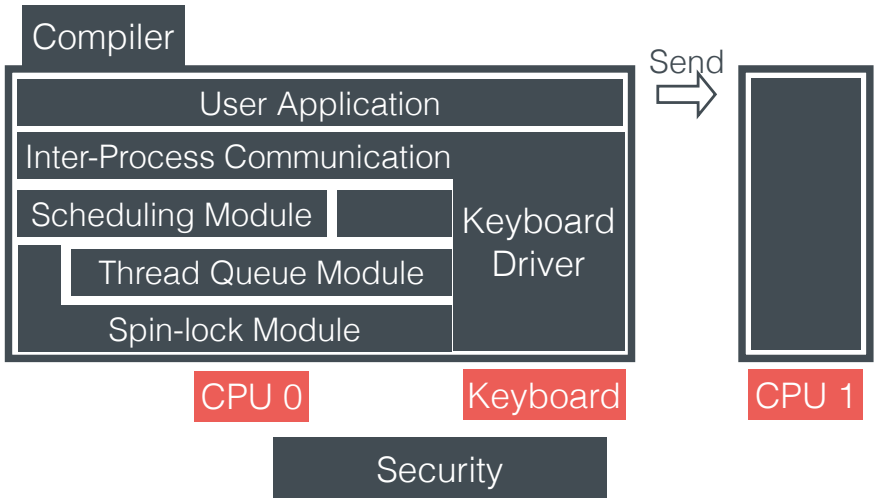
Observation function O

- **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: 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, …)