
*National Science Foundation  Expedition in Computing*

# the science of deep specification

DeepSpec Summer School, July 2018, Lecture Video


# CertiKOS II

## Certified Kit Operating System

Zhong Shao

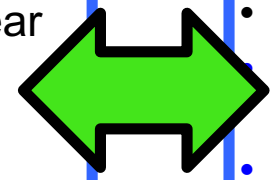
www.deepspec.org

# 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 Research: Traps & Pitfalls

- Too theoretical
  - isolated in its own niche, but real world systems have many other important issues (e.g., concurrency, persistence, fault-tolerance)
- Too practical (but messy)
  - Legacy COTS … difficult to apply new principled techniques
- Too narrow
  - One PL for one application domain (e.g., DSL)?
- Too general
  - No such thing as the "perfect" general-purpose language
- Too ambitious
  - New GPL: takes years to design & impl; need to build new libs; …
- Interoperability challenge? The tower of Babel
  - How to run on top of existing "system software stacks"

# PL Meets OS: A Marriage Made in Heaven?

- 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 compositional framework via linear logic + game semantics**

**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 (not just safety / partial correctness properties)

# Summary: Results So Far

- Applications
  - DeepSpec web server demo
  - Self-driving car (1/10th scale w. Intel NUC or NVIDIA Jetson TX2)
  - Quadcopter (w. Raspberry Pi3)

Penn

Yale

- Verification of user-level C programs using VST [JAR'18]
  - New verified crypto lib [CCS'17] and messaging system [OOPLSA'17]

Princeton

Verified Software Toolchain

- Compositional concurrent OS and abstract machines
  - CCAL & Concurrent CertiKOS [OSDI'16, APLAS'17, PLDI'18]
  - Preemption, priority scheduling, and real-time CertiKOS
  - CertiKOS w. modular device & networking support [PLDI'16, JAR'17]

Yale

CertiKOS

- User- & kernel-level semantic integration
  - Concurrent CompCert for VST and Verifiable C
  - CompCert extension for generating machine code w. flat memory model
  - Compositional & Concurrent & Stack-aware CompCert/CompCertX

Princeton

Yale

- Machine-level semantics
  - CertiKOS/CompCertX LAsm semantics for x86, ARM, and RISC-V
  - MIT Kami's formal semantics of RISC-V ISA

Yale

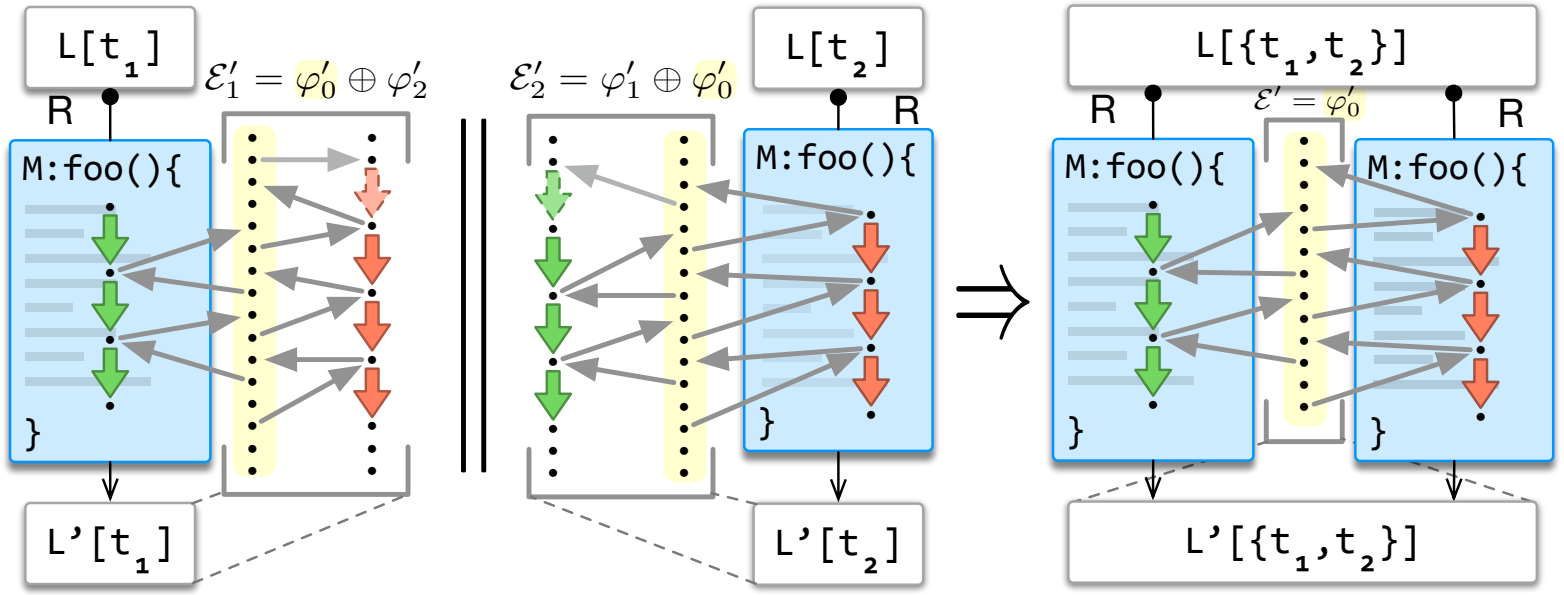

MIT

# CertiKOS: DeepSpec Challenges & Plans

- End-to-end assurance case for DeepSpec servers & smart cars
  - New semantic models for composing concurrent & networked objects (e.g., HW/SW/device components)
  - Preemption, priority scheduling, and resource-aware CertiKOS
  - Certified linking of concurrent layers with verified compilers
- Connecting CompCertX with new CertiKOS machine models
  - Compositional & concurrent & stack-aware CompCertX
- Connecting DeepSpec server and VST with CertiKOS
  - User- & kernel-level semantic integration
  - Connecting VST spec interface to CompCert w. Yale's & INRIA's
  - CompCertX for generating machine code w. flat memory model
- Connecting CertiKOS with machine-level semantics (RISC-V)

# *New Result:* Certified Concurrent Abstraction Layers (CCAL) [PLDI'18]

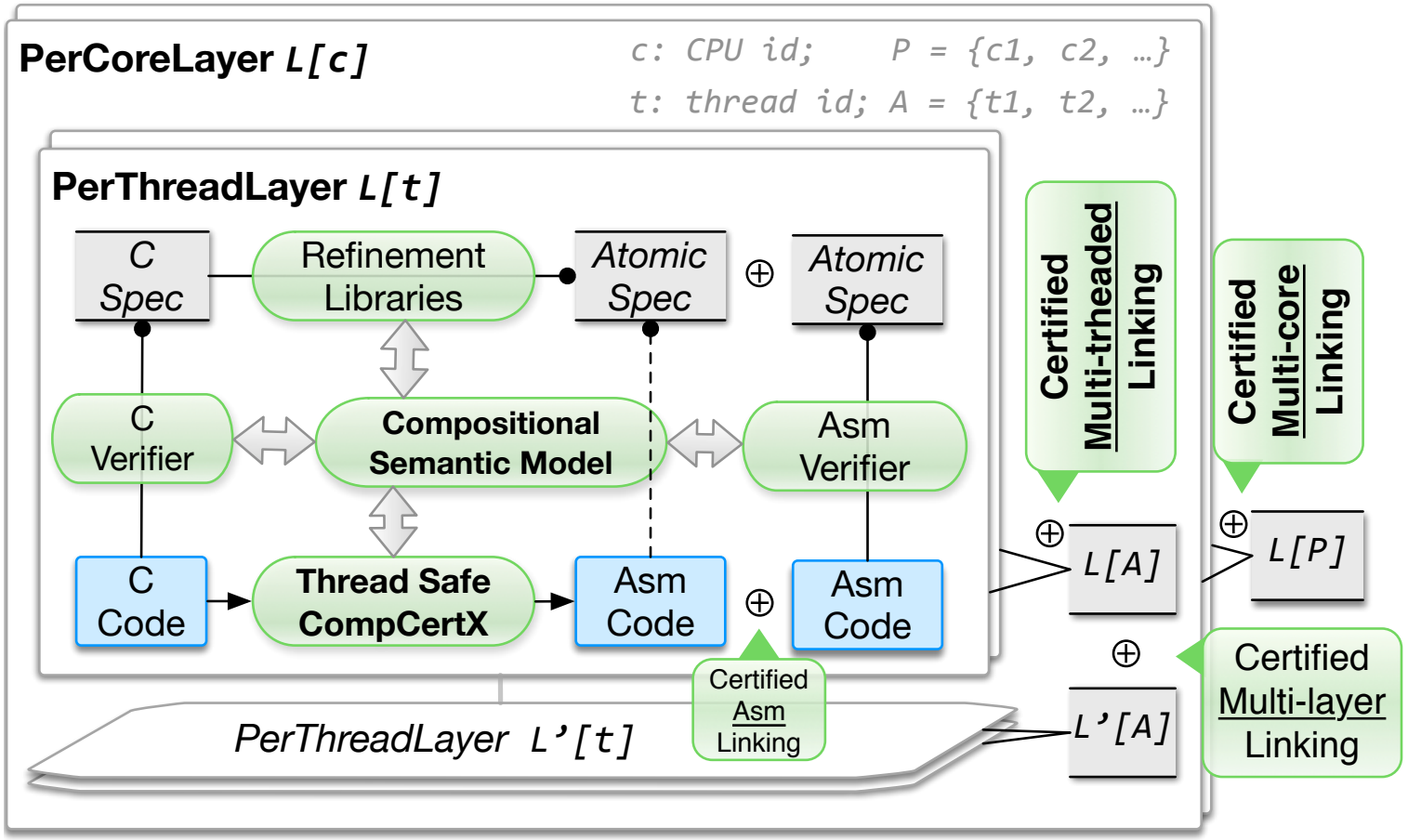
- **Key insight:** model each sequential/concurrent/networked object as a "game" (played against its ***environment context***)
- Multicore & multithreaded linking ←→ composing "games" ←→ composing "game semantic" strategies
- *Thread-local* (or CPU-local) *reasoning* for CCAL layers (against their *environment contexts*)



# *New Result:* Certified Concurrent Abstraction Layers (CCAL) [PLDI'18]

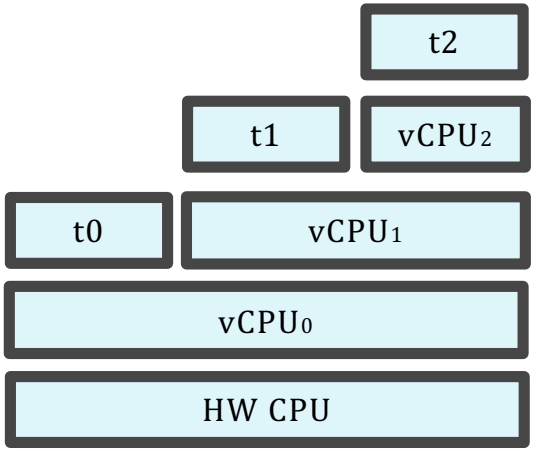
New programming toolkit w. certified multicore & multithreaded linking:

*Composition = parallel composition + hiding (abstraction)*

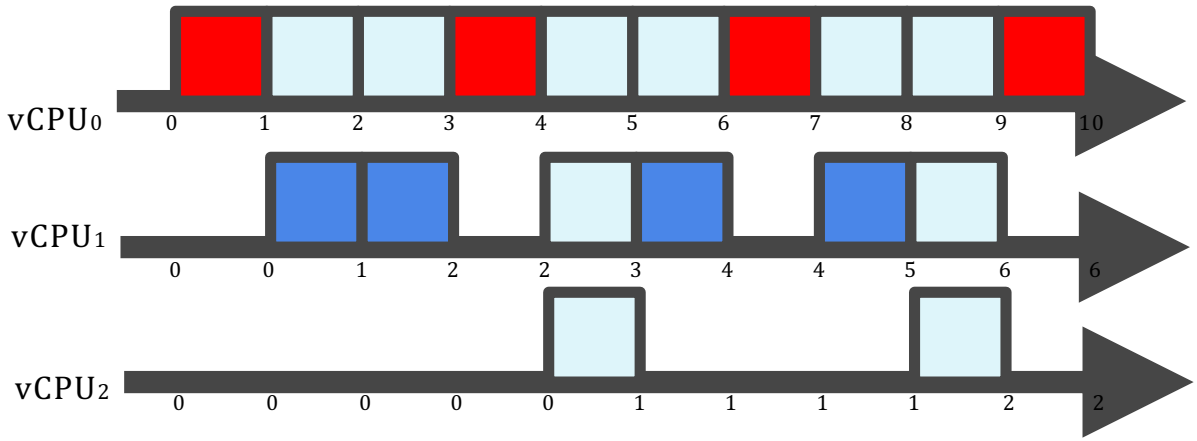


# *In Progress:* Real-Time CertiKOS

- CertiKOS now models timer interrupts & timing registers.
- CertiKOS is now preemptive with a hybrid scheduler
  - Priority scheduling for real-time periodic threads
  - round-robin for batch tasks
- New information-flow security proof for CertiKOS with temporal and spatial partitions
  - Virtual time: time available for a specific priority level
  - Virtual timeline: mapping from physical time to virtual time



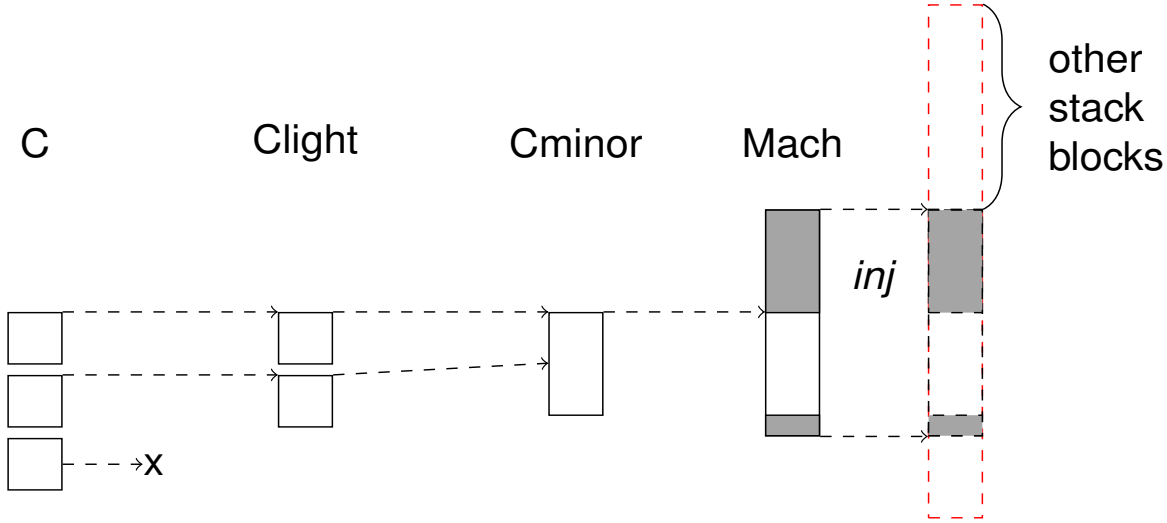

| priority | period | WCET |
|---|---|---|
| 0 | 3 | 1 |
| 1 | 5 | 2 |
| 2 | 9 | 1 |



# *In Progress:* Stack-Aware CompCert

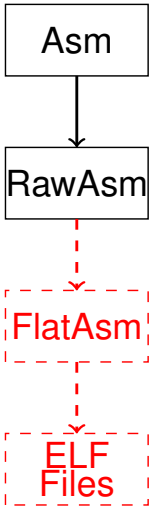
***Our proposal:*** enrich the block-based memory model with an abstract stack:

- instrument all CompCert IR semantics w. a stack requirement spec
- Maintain an injection from stack blocks to a continuous stack block
- Enforce access control for private / public regions of each stack block



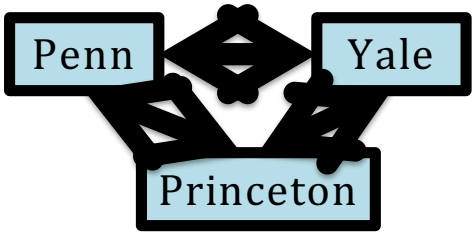

***Benefits:***

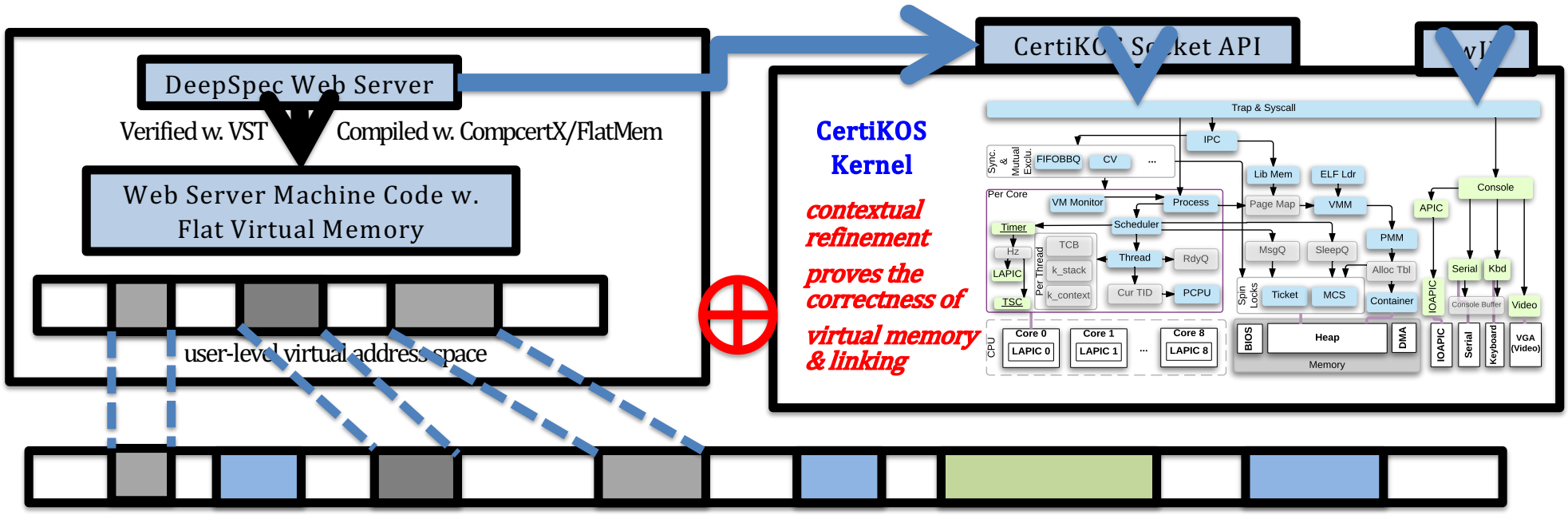
- Light-weight compositionality of heterogenous modules
- Merging the stack blocks into a finite contiguous space
- Verified translation from CompCert Asm into ELF files
  - Asm → RawAsm (merged stack blocks) → FlatAsm (flat mem) → machine code



# *In Progress:* Connecting VST / DeepSpec Web Server with CertiKOS

- Deepspec web server now runs on CertiKOS blackbox (see the demo)
- The web server will be verified using VST, while the socket API is specified as functions over CertiKOS abstract state
  - How to model the physical network?
  - What parts of CertiKOS should the web server be aware of?
  - How to make CertiKOS specs be compatible with VST specs?
- Integration of user-level flat virtual memory and CertiKOS kernel memory

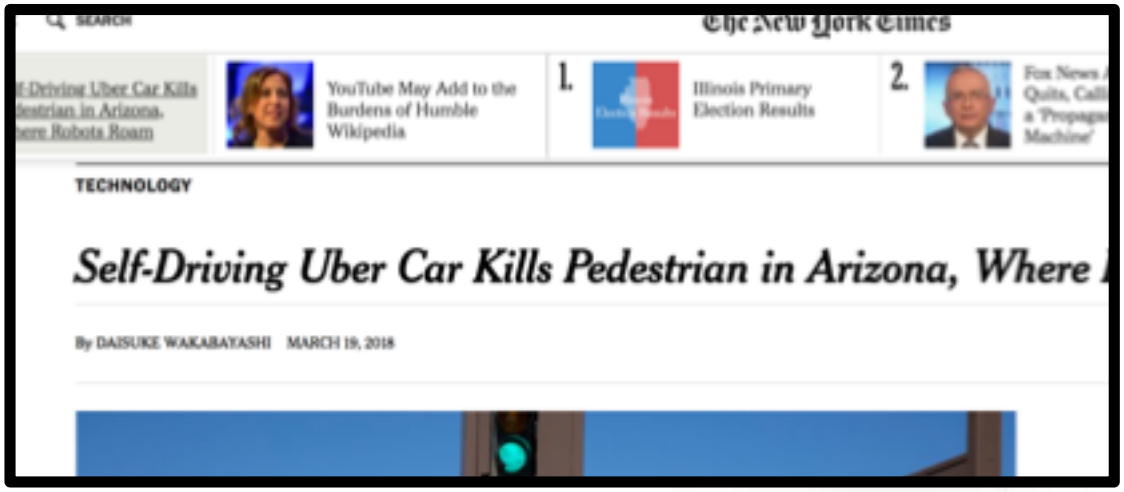


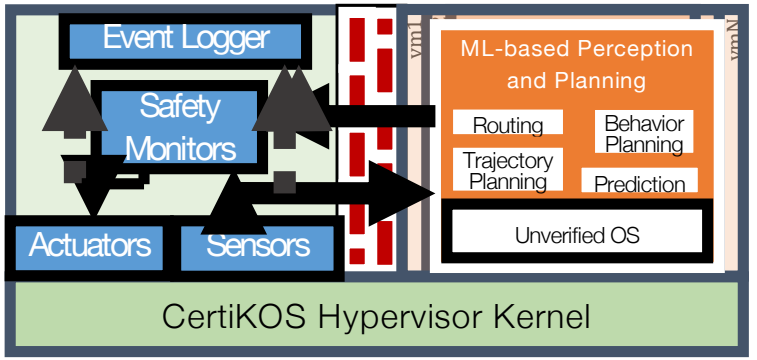
# *In Progress*: Certified Self-Driving Cars

**Goal:** End-to-end verification of safety & security properties of smart cars

Machine learning enables self-driving, but does not guarantee safe-driving

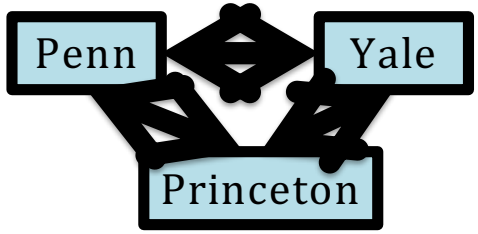



- Model-based safety monitors
  - e.g., safe distance, obstacle detection
  - Multi-agent verification (by rely-guarantee & CCAL)
- Verified event logger for post-analysis
  - E.g., accident investigation, maintenance
- CertiKOS-based security guarantee
  - Isolation for critical components
  - Real-time guarantee



# Highlights of Future Plans

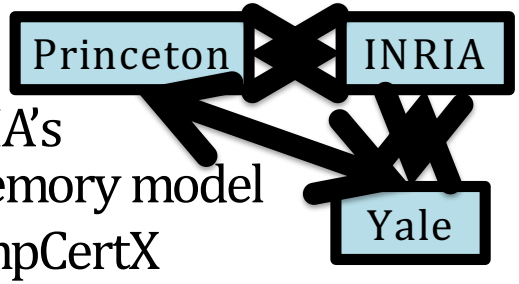
- Applications
  - Certified DeepSpec web server demo
  - Connecting DeepSpec server and VST with CertiKOS
  - Certified self-driving car & quadcopter



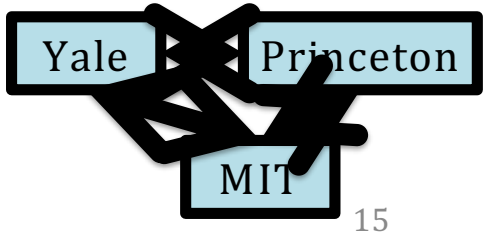

- Verification of user-level C programs using VST
  - Concurrent separation logic for VST


- Compositional concurrent OS and abstract machines
  - General compositional model for concurrent & networked abstract machines
  - Preemption, priority scheduling, and resource-aware CertiKOS
  - CertiKOS port to ARM / x86 / RISC-V & latest Coq/CompCert


- User- & kernel-level semantic integration
  - Connecting VST spec interface to CompCert w. Yale's & INRIA's
  - CompCert extension for generating machine code w. flat memory model
  - Compositional & Concurrent & Stack-aware CompCert/CompCertX




- Machine-level semantics
  - Bridging CertiKOS/CompCertX LAsm & RISC-V semantics
  - Provable resilience to side-channel attacks

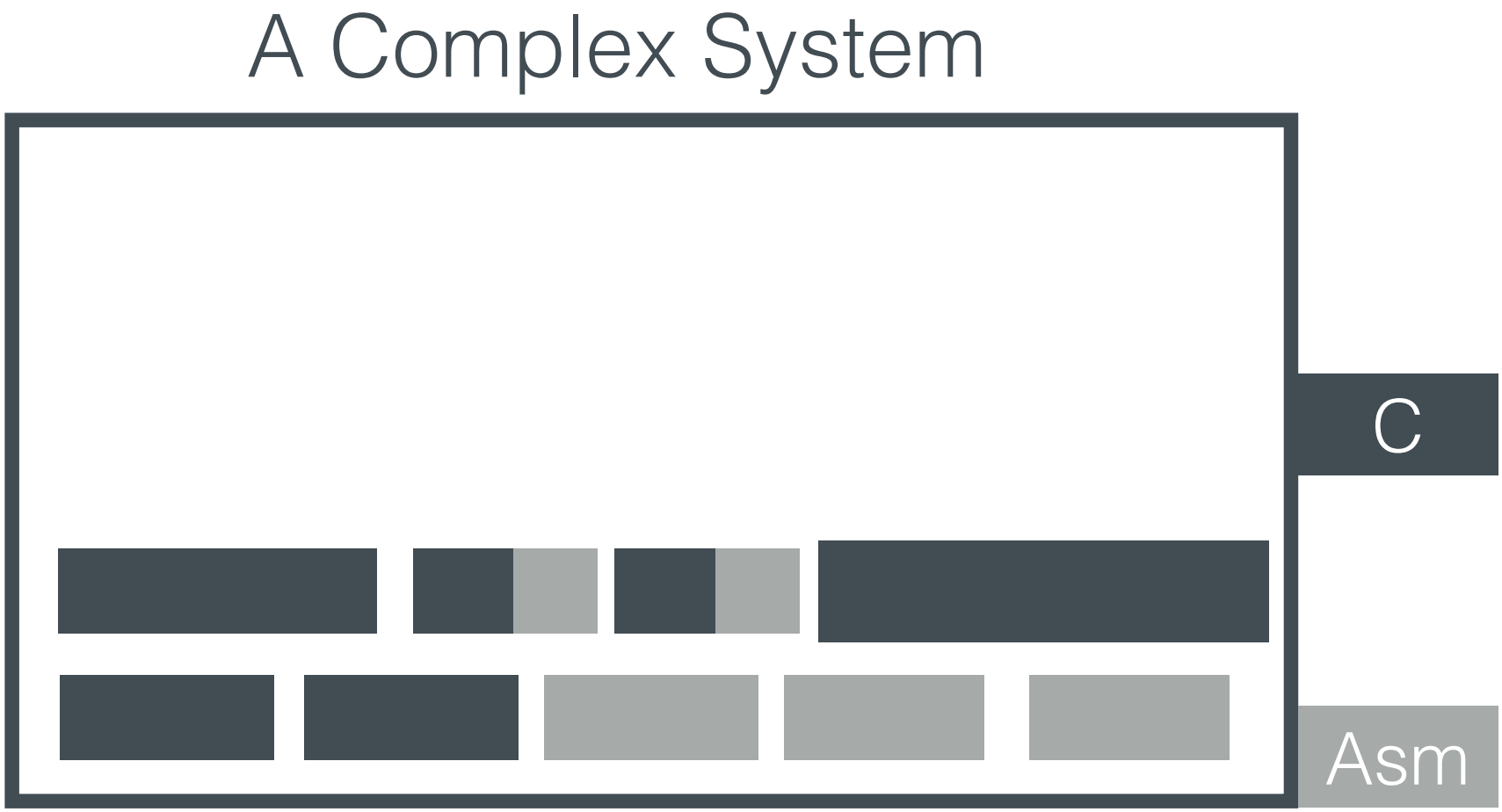


# Certified Concurrent Abstraction Layers

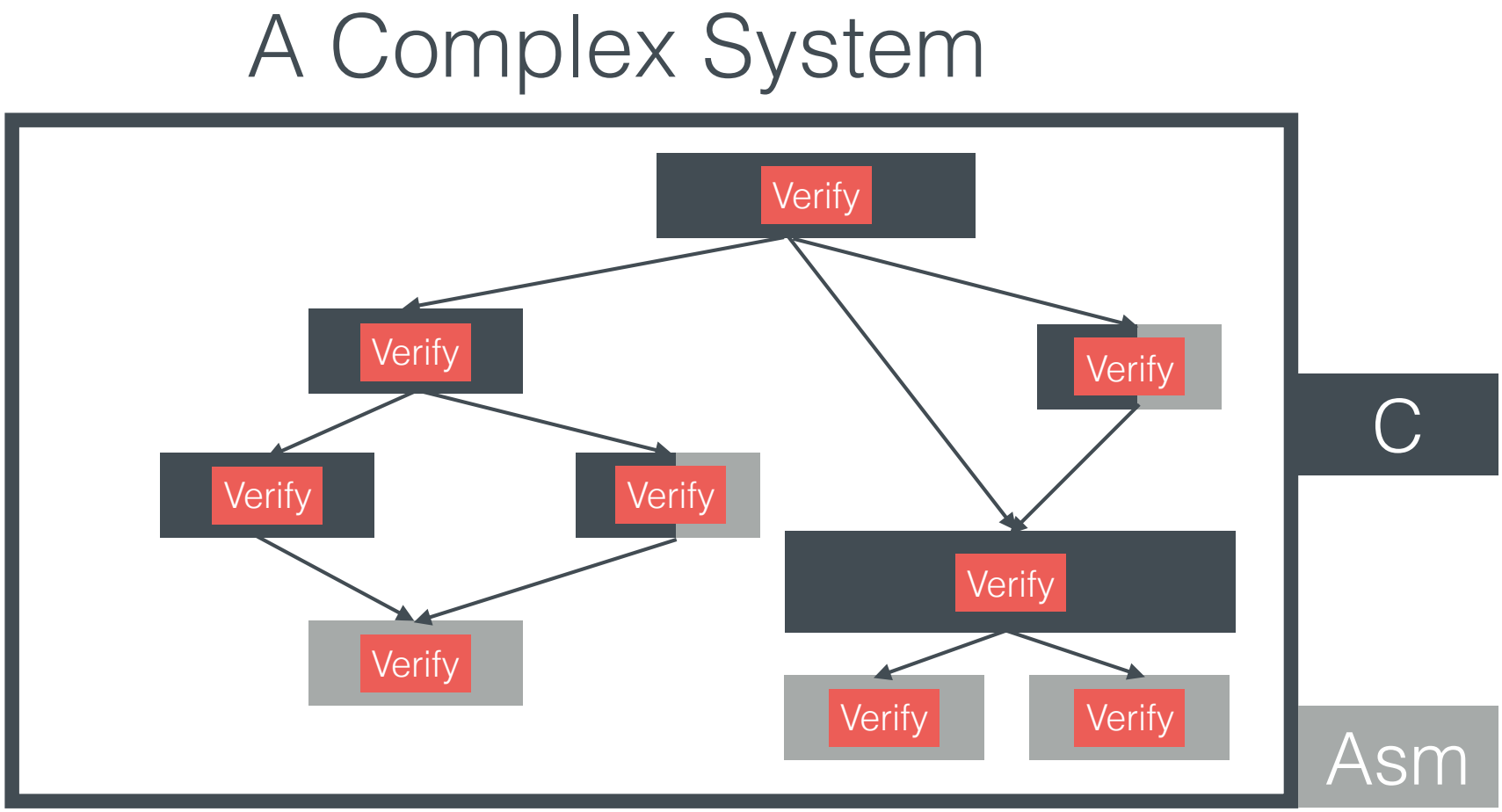
**Ronghui Gu**, Zhong Shao, Jieung Kim, Xiongnan (Newman) Wu, Jérémie Koenig, Vilhelm Sjöberg, Hao Chen, David Costanzo, Tahina Ramananandro

Yale University & Columbia University

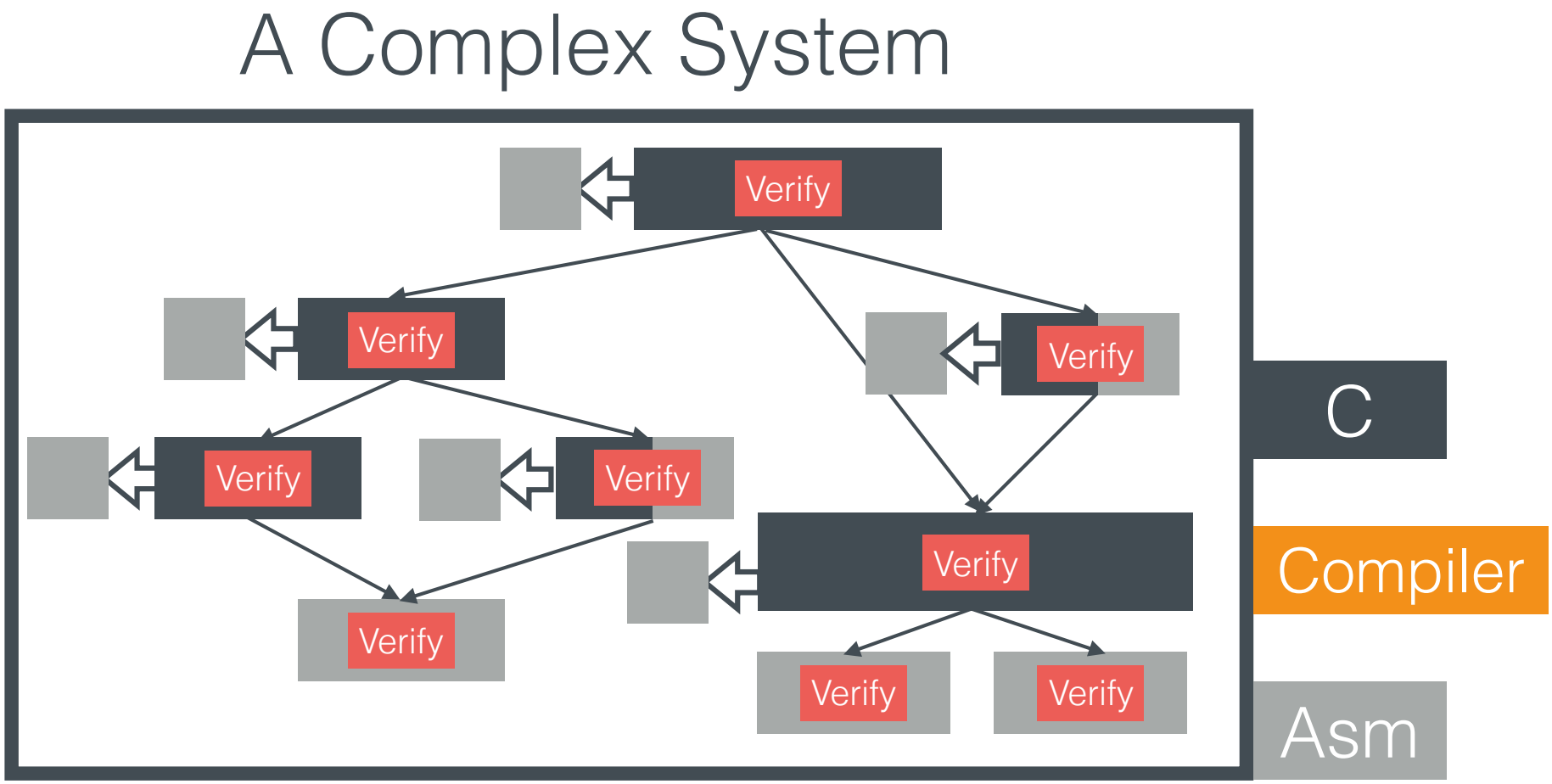
# Challenges: Compositionality



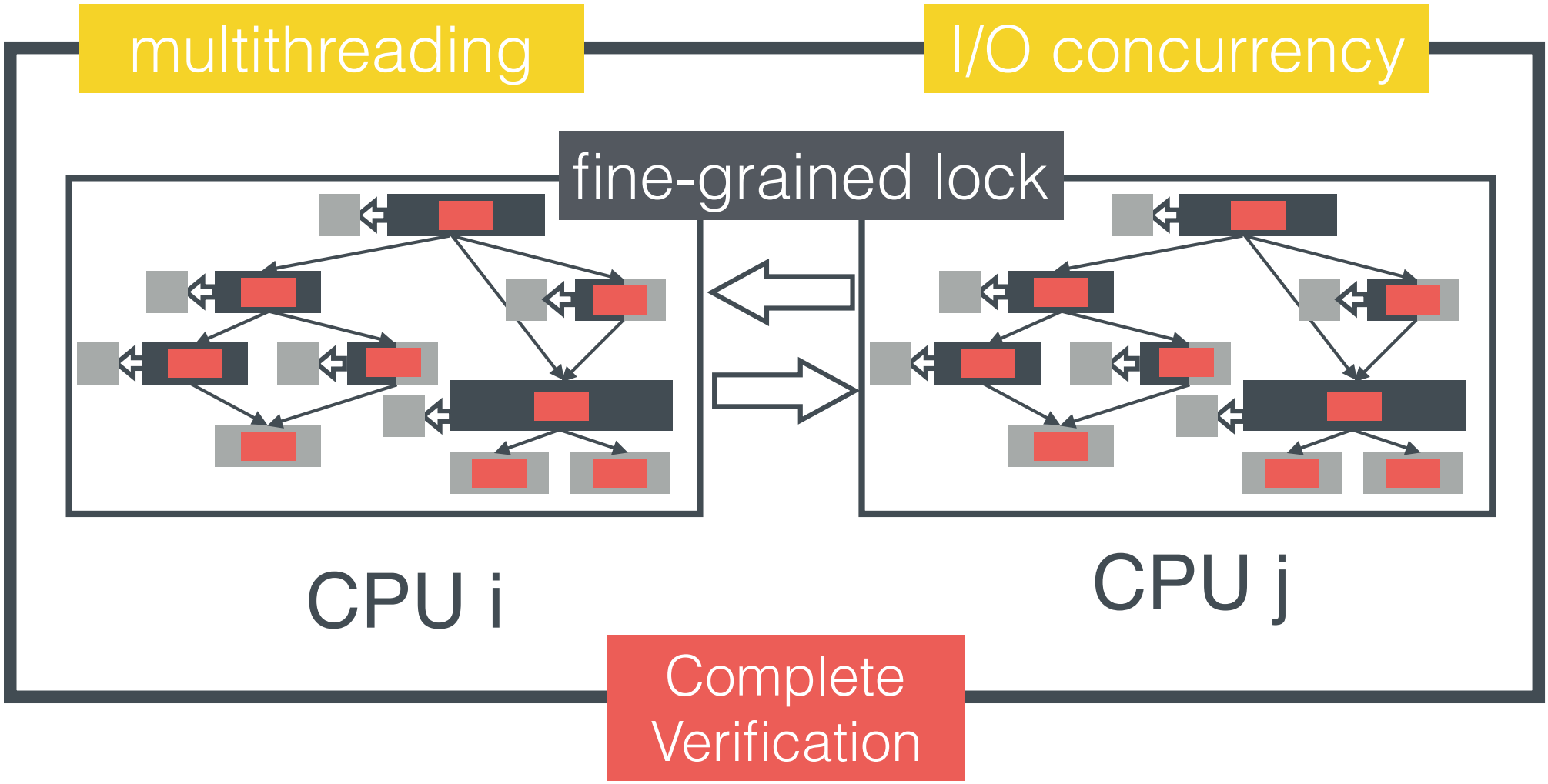
# Challenges: Compositionality



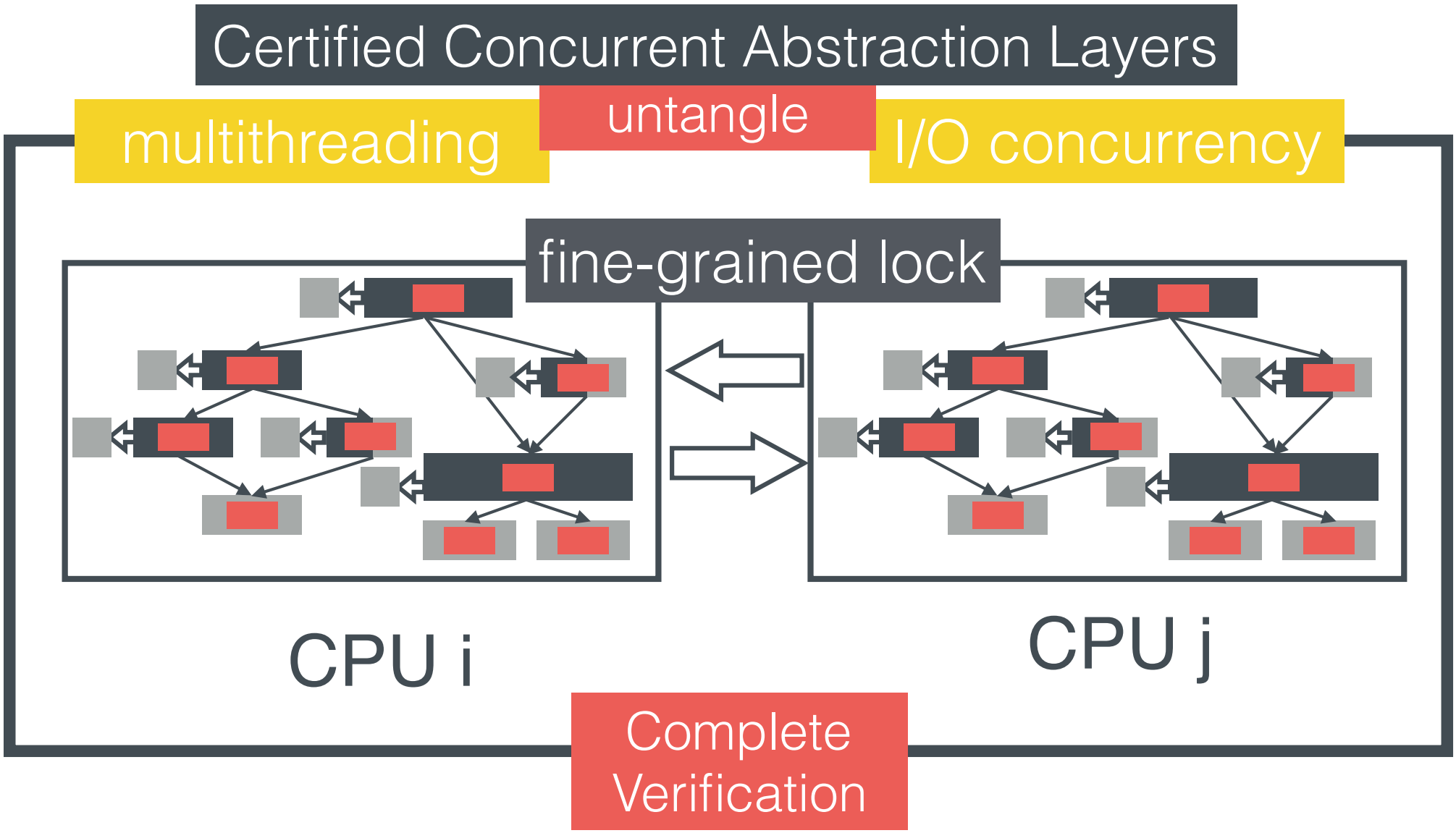
# Challenges: Compositionality



# Challenges: Concurrency

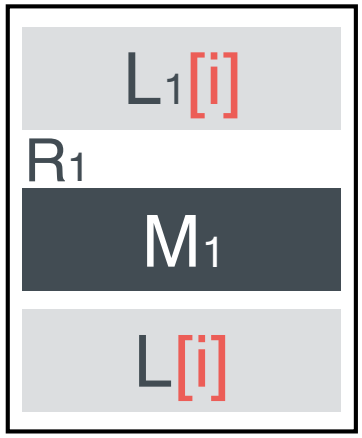


# Contribution



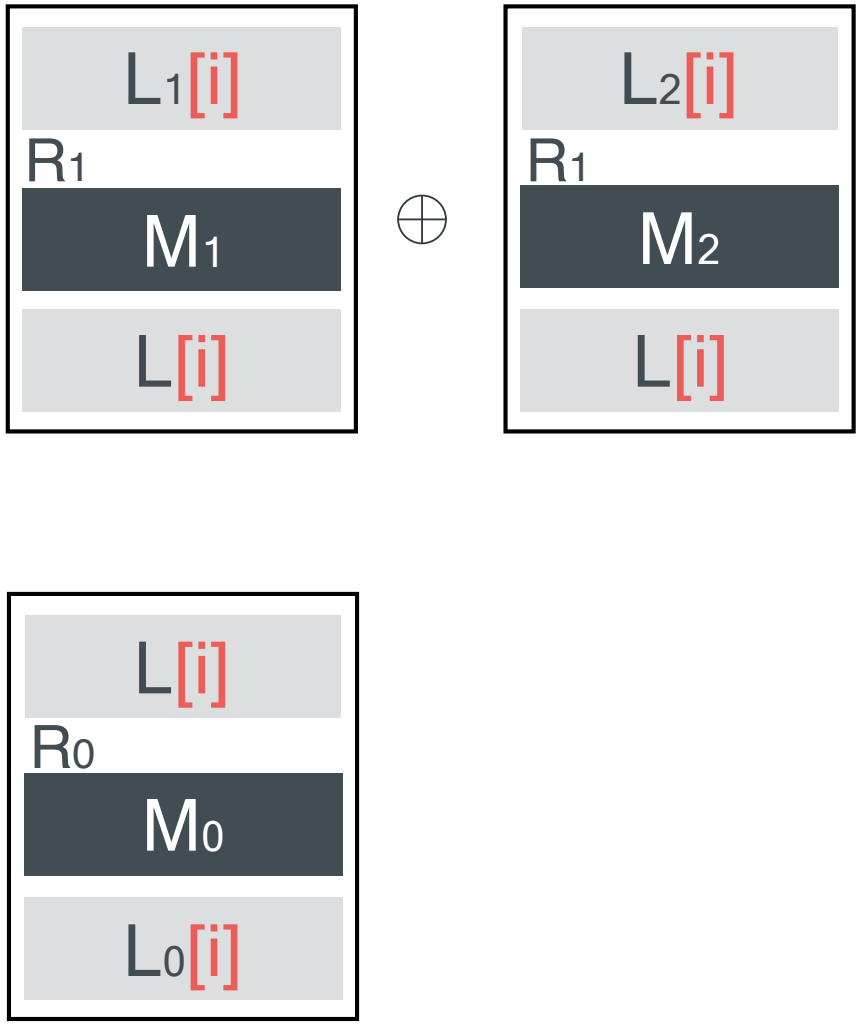
# Contribution

## Certified Concurrent Abstraction Layers



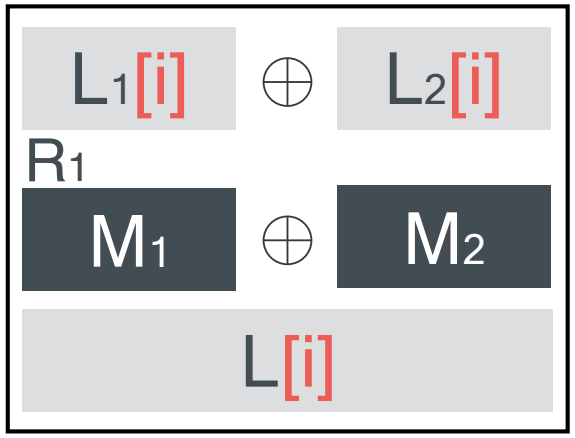
# Contribution

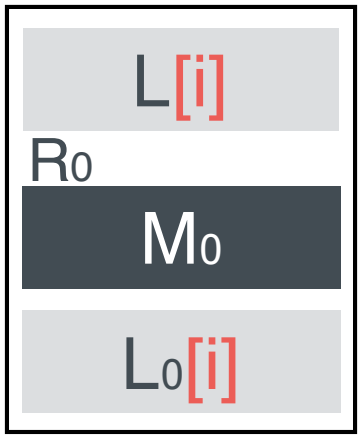
## Certified Concurrent Abstraction Layers



# Contribution

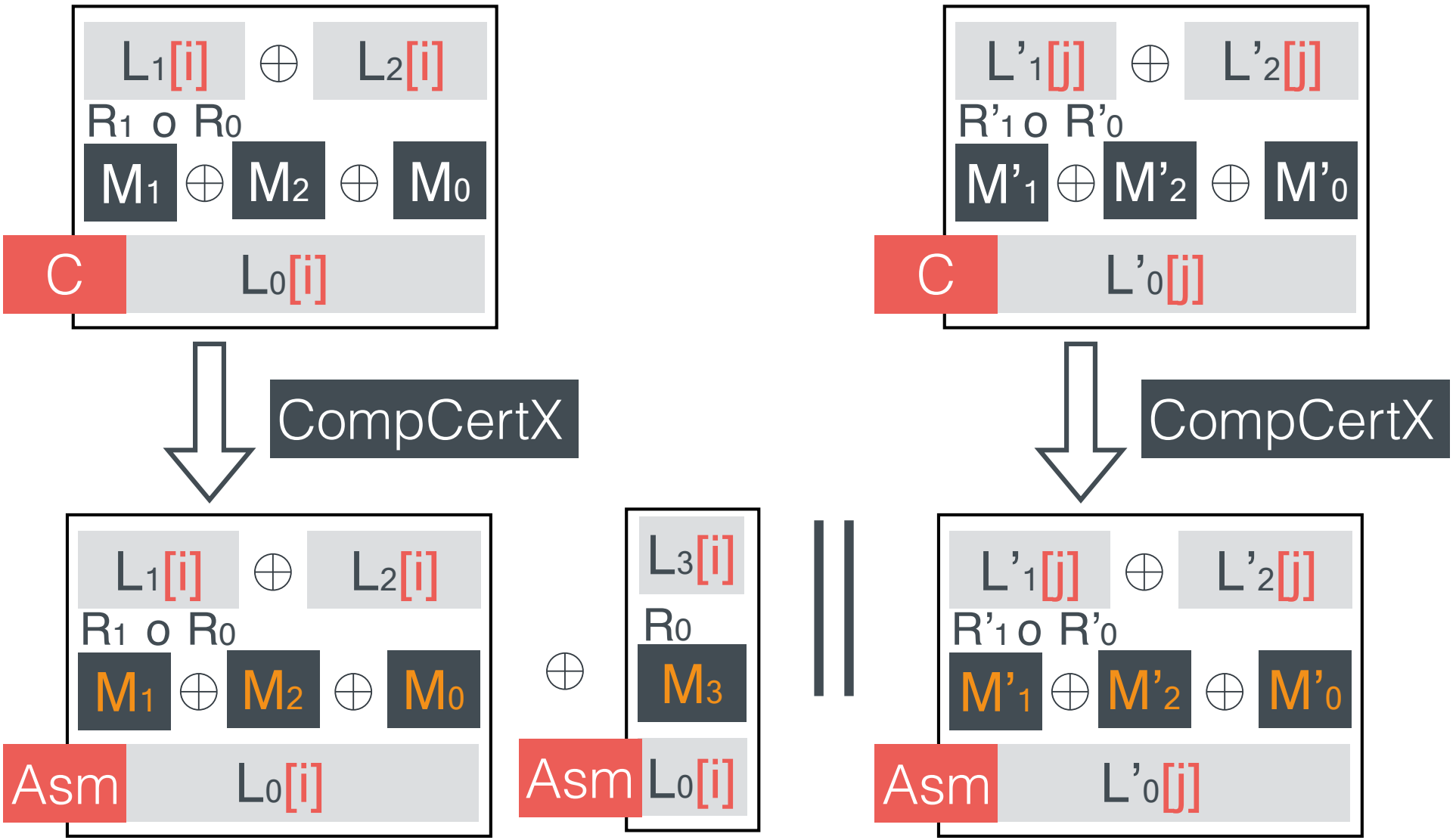
## Certified Concurrent Abstraction Layers



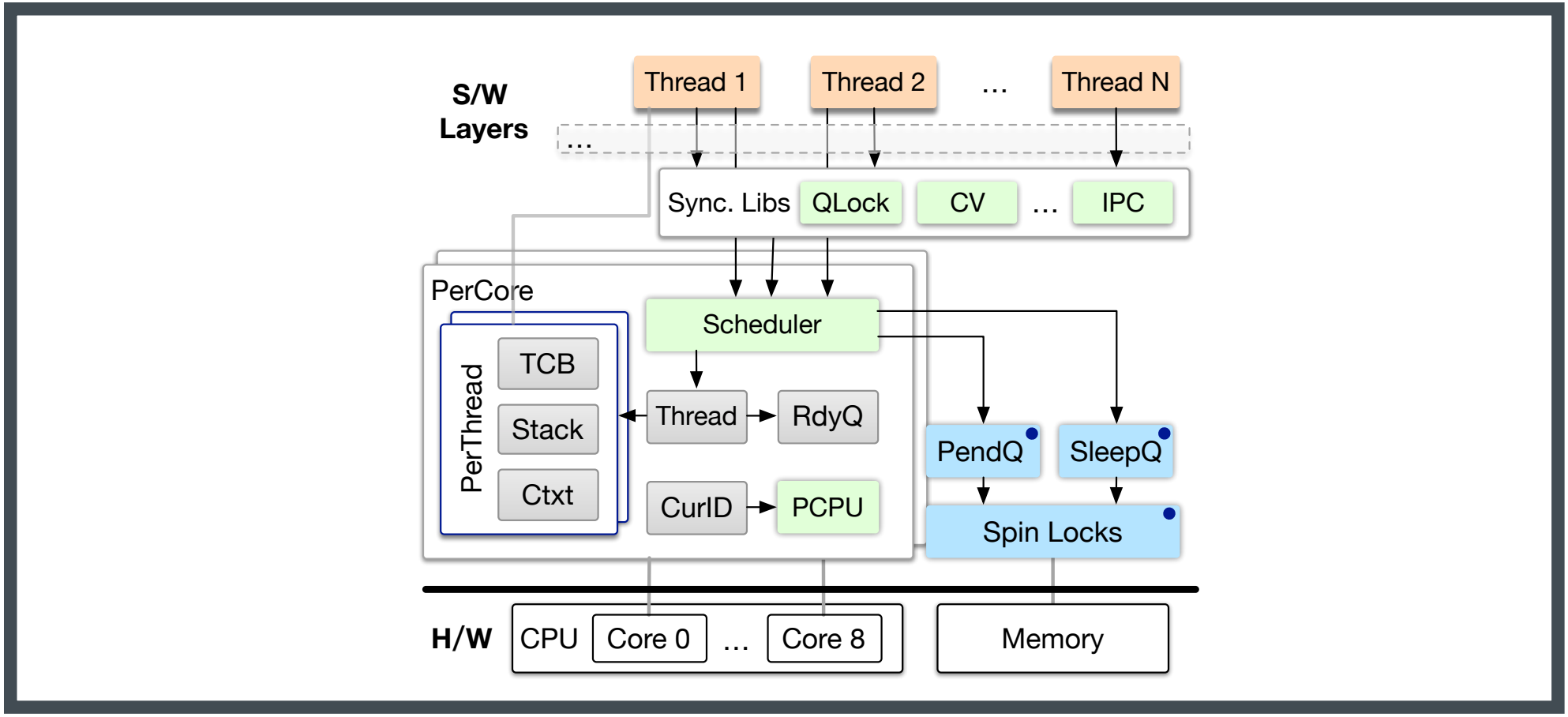



# Contribution

## 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

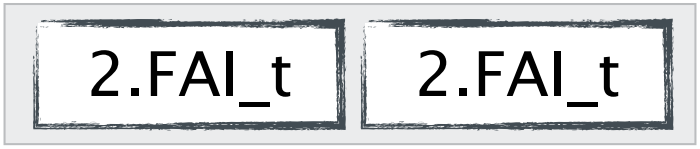
strategy $\psi_p[i]$

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

# Strategies and Game Semantics

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


logical
log l



# Strategies and Game Semantics

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


logical log l


$2

# Strategies and Game Semantics

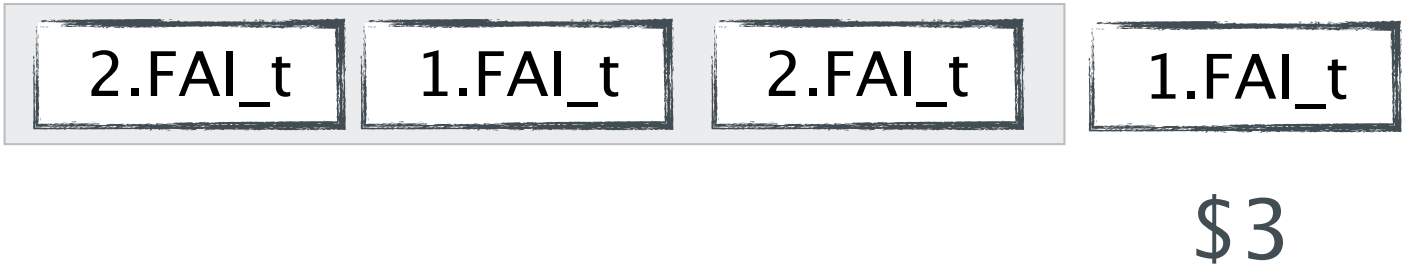
$$\psi_{\texttt{FAI\_t}}[1]$$


logical log l

# Strategies and Game Semantics

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


logical log l

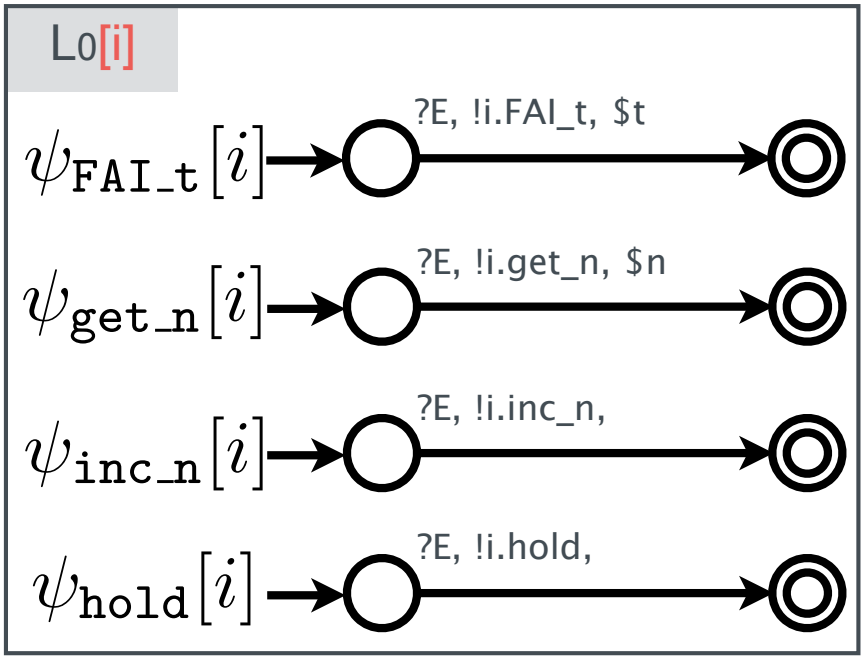


# Strategies and Game Semantics

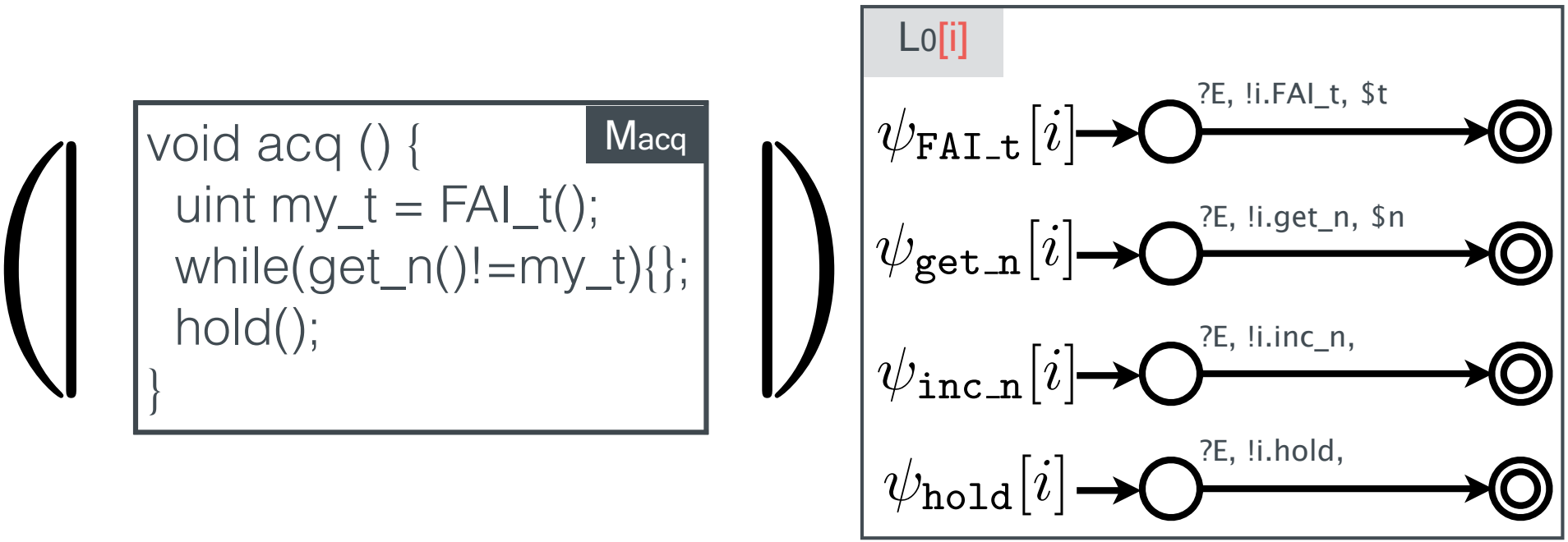
L0[i]

```
extern uint FAI_t();

extern uint get_n();

extern void inc_n();

extern void hold();
```

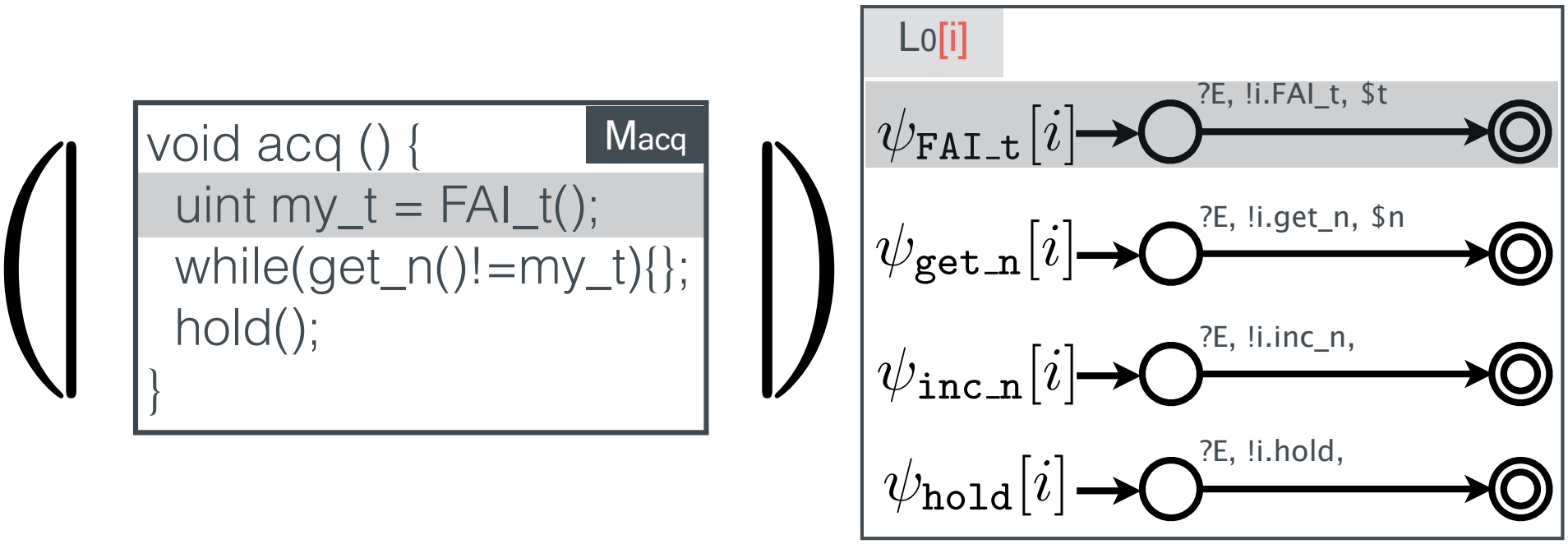
# Strategies and Game Semantics

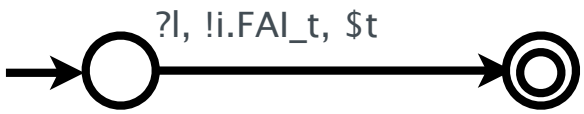


# Strategies and Game Semantics

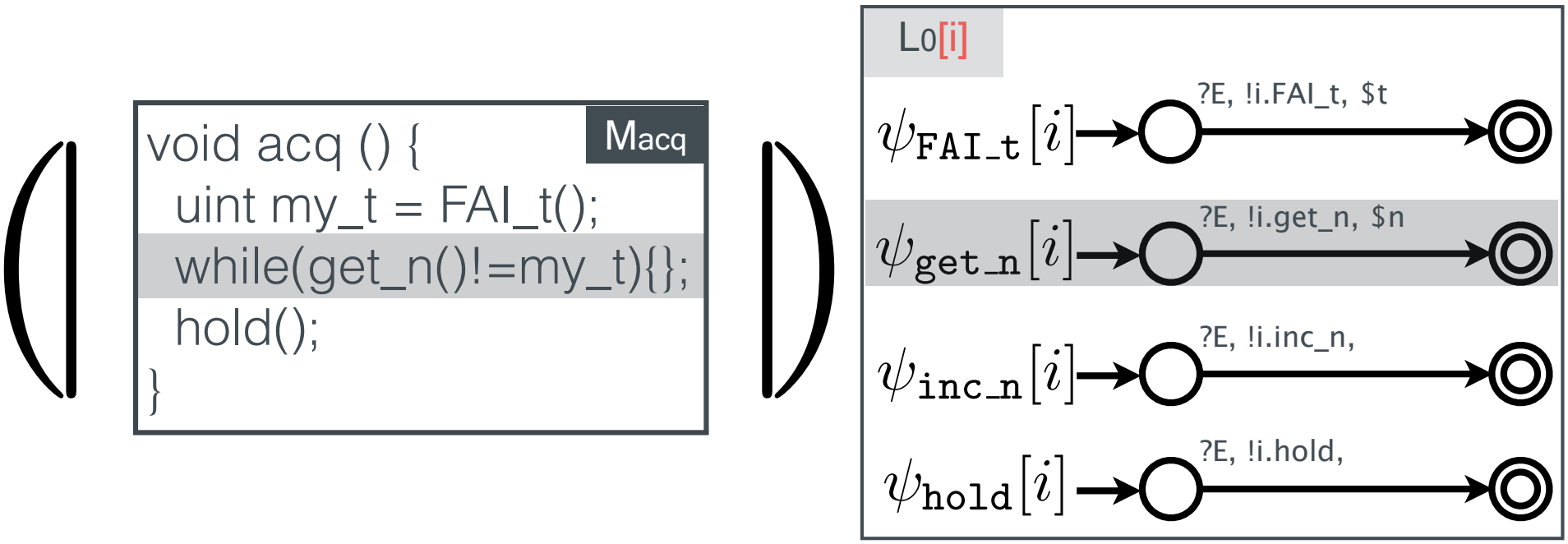


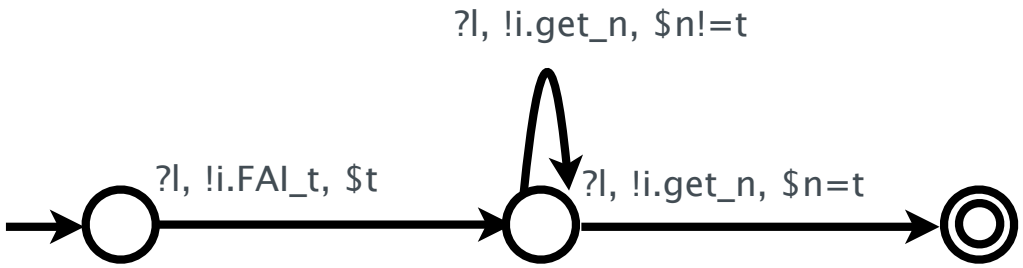
# Strategies and Game Semantics



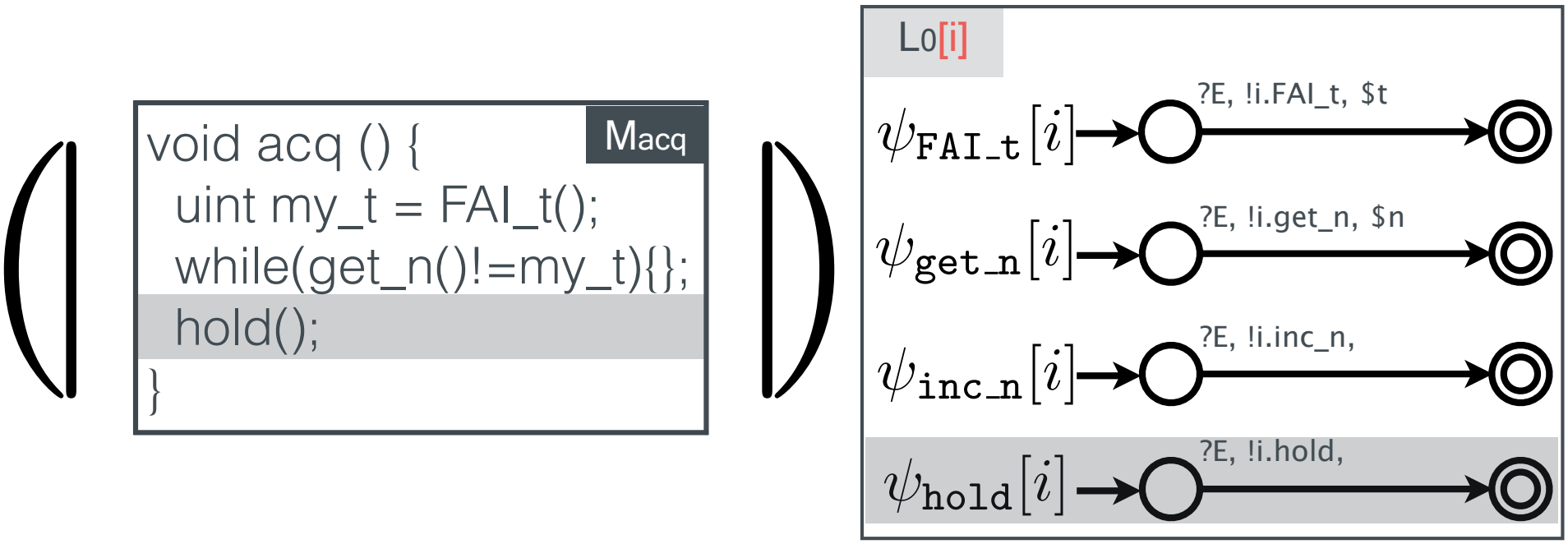



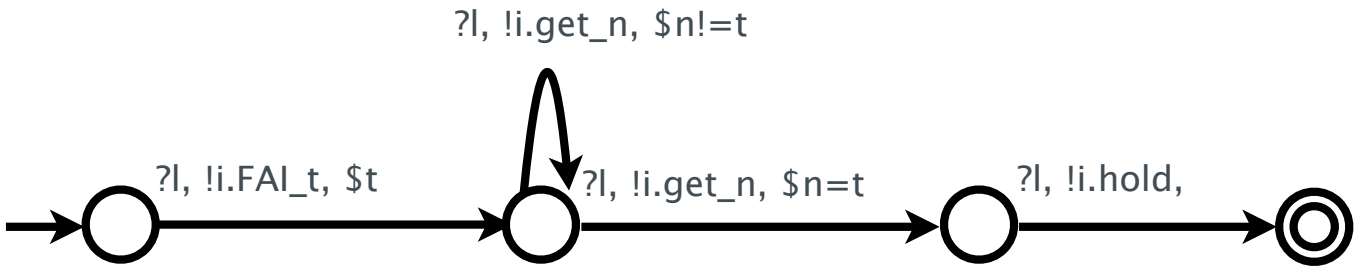
# Strategies and Game Semantics



# Strategies and Game Semantics



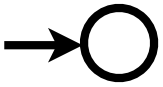



# Strategies and Game Semantics

strategy $(|$ Macq $|)$ $L_0[i]$

Given the current **log** l, how the module Macq running over $L_0[i]$ will generate **events** on behalf of CPU i at each step.

# Strategies and Game Semantics

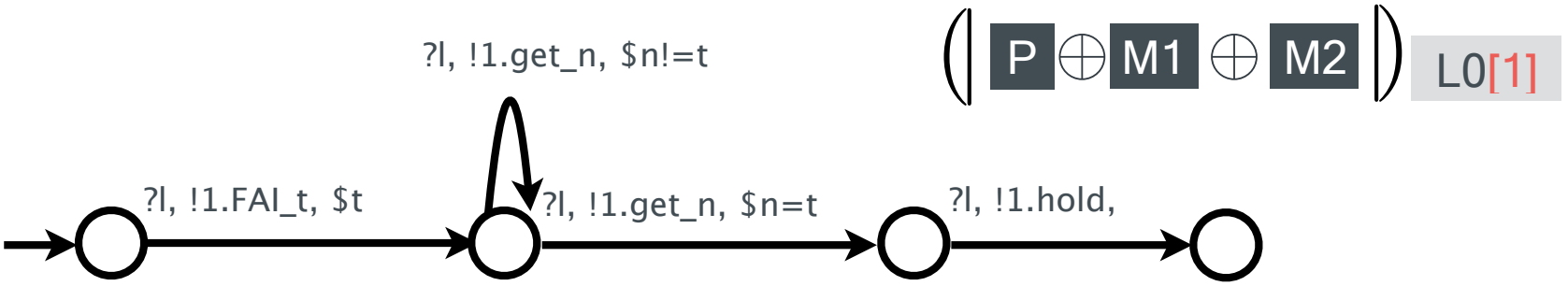



```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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




```
//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



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

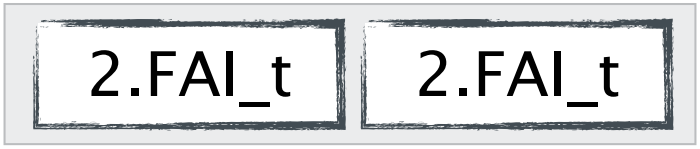
C

# Strategies and Game Semantics



?l, !1.FAI_t, $t

logical
log l



# Strategies and Game Semantics



?l, !1.FAI_t, $t

logical log l


$2

# Strategies and Game Semantics

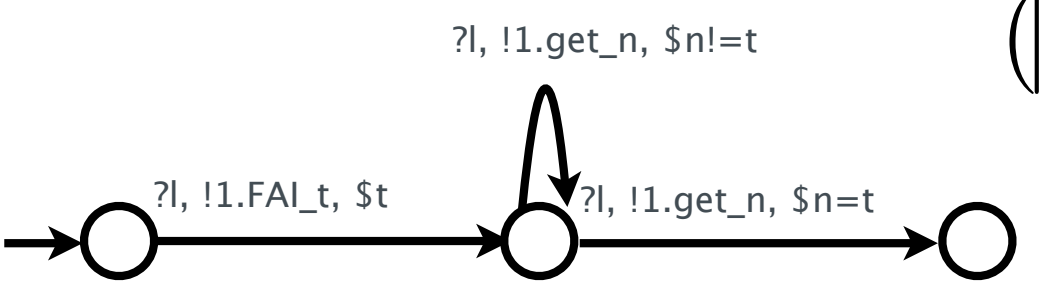


?l, !1.FAI_t, $t

logical log l


$3

# Strategies and Game Semantics



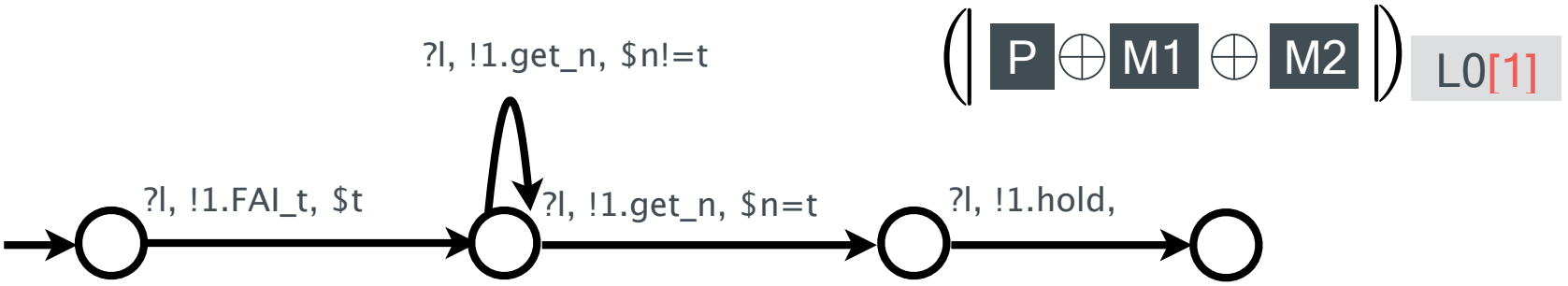


```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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

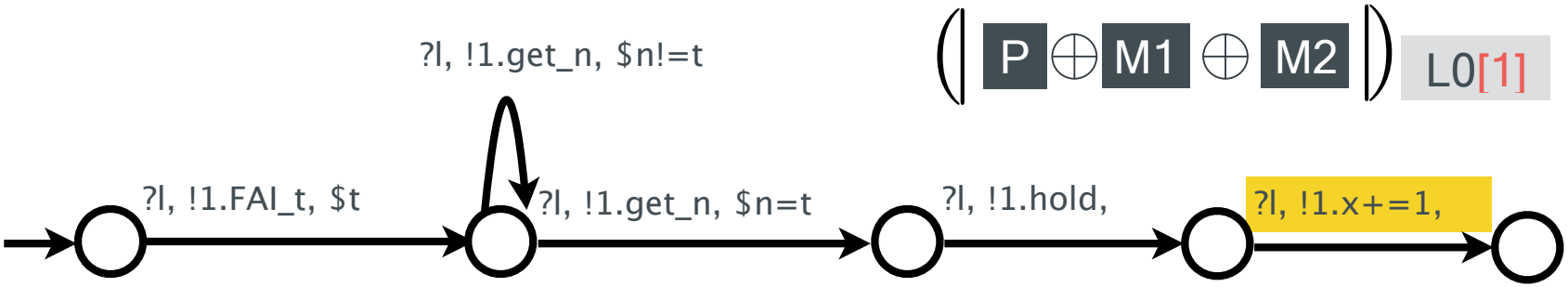



```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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

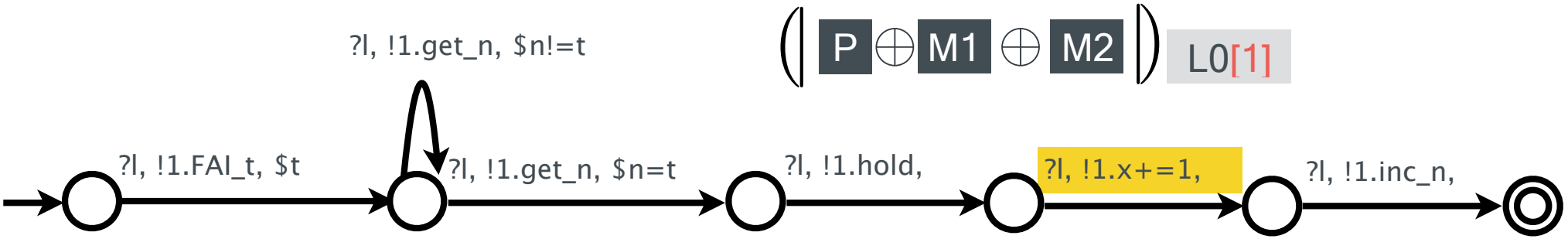



```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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

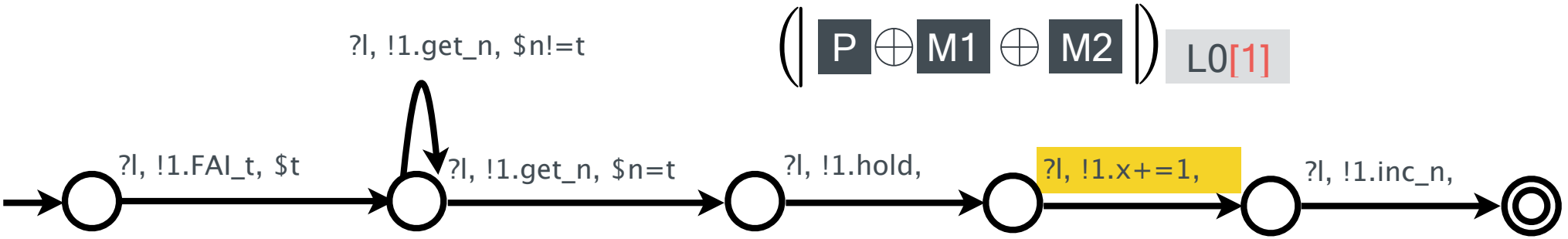


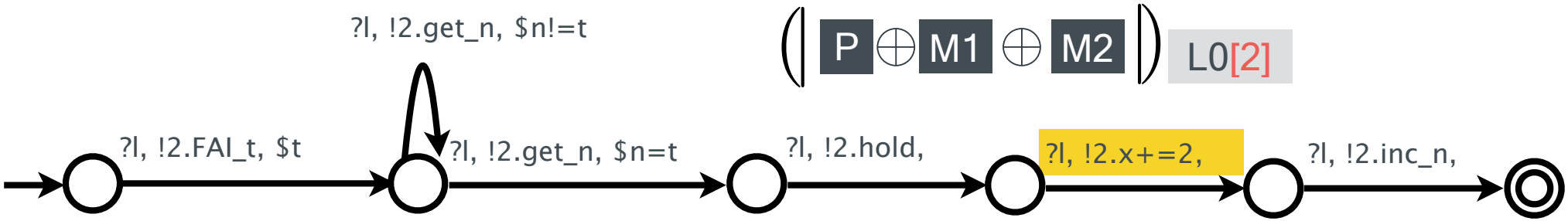

```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

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

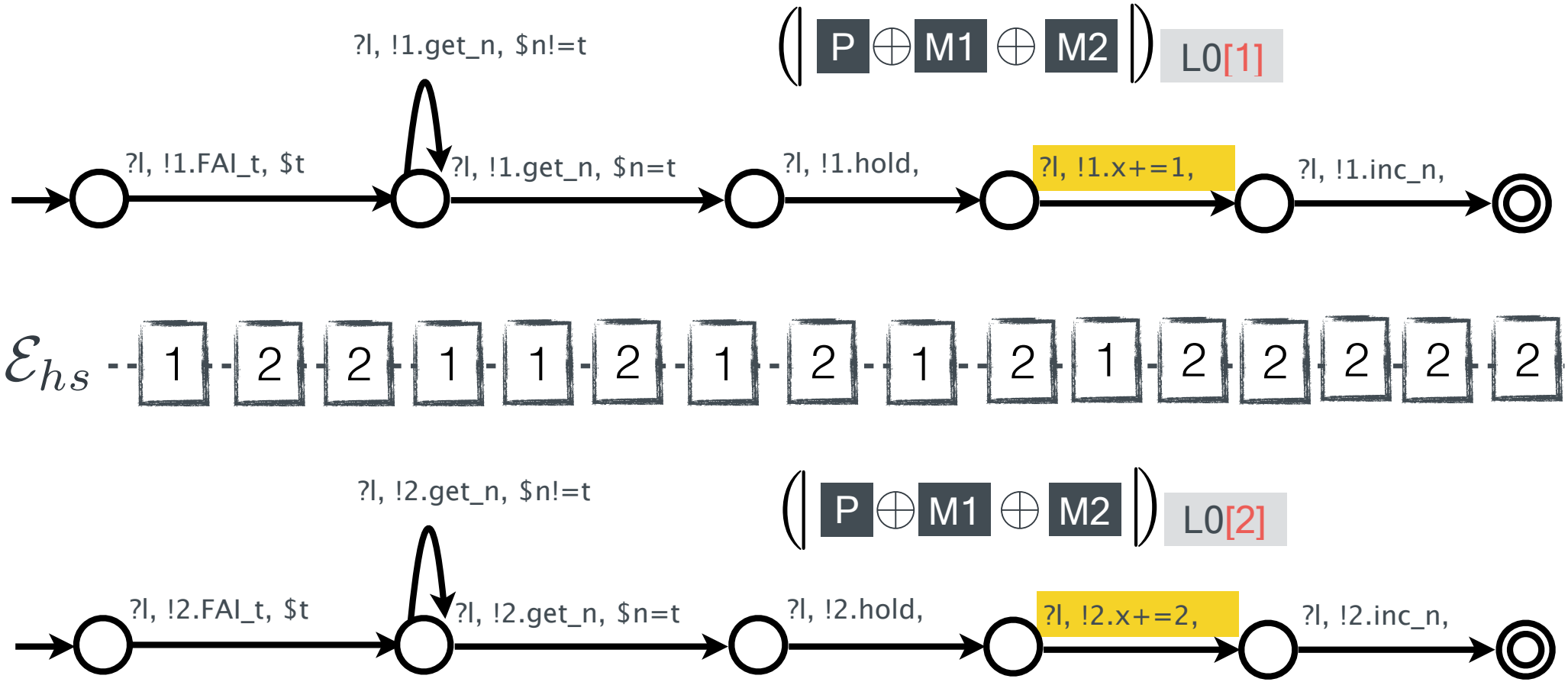
C

# Strategies and Game Semantics

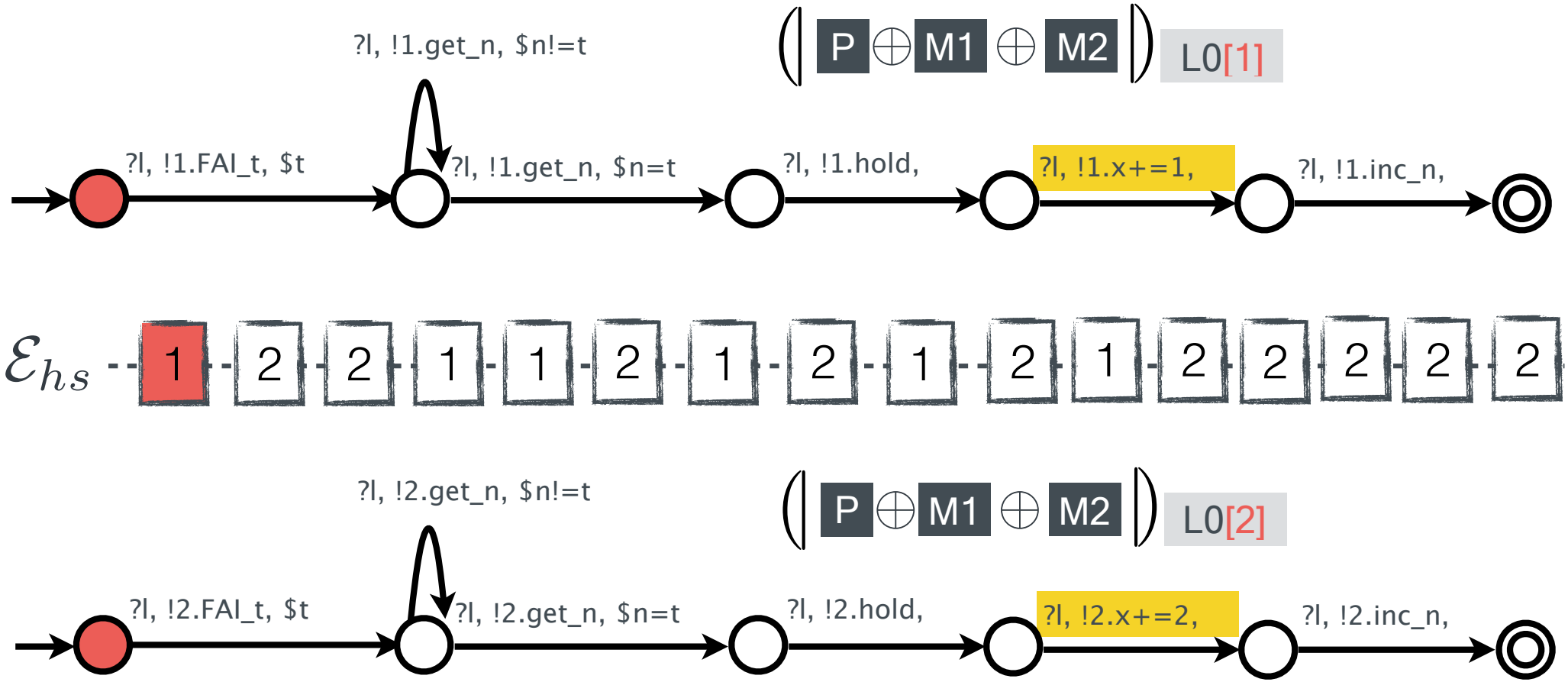


# Strategies and Game Semantics

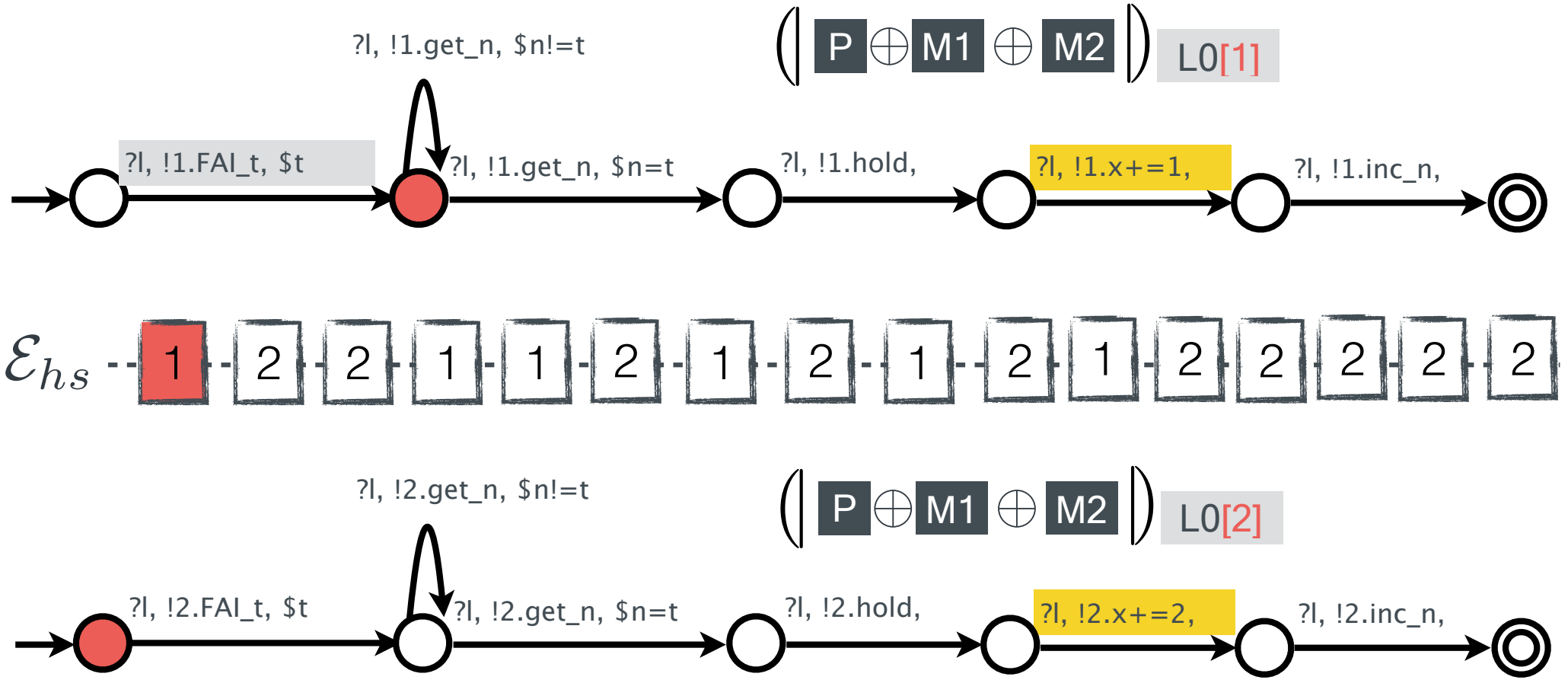


# Strategies and Game Semantics



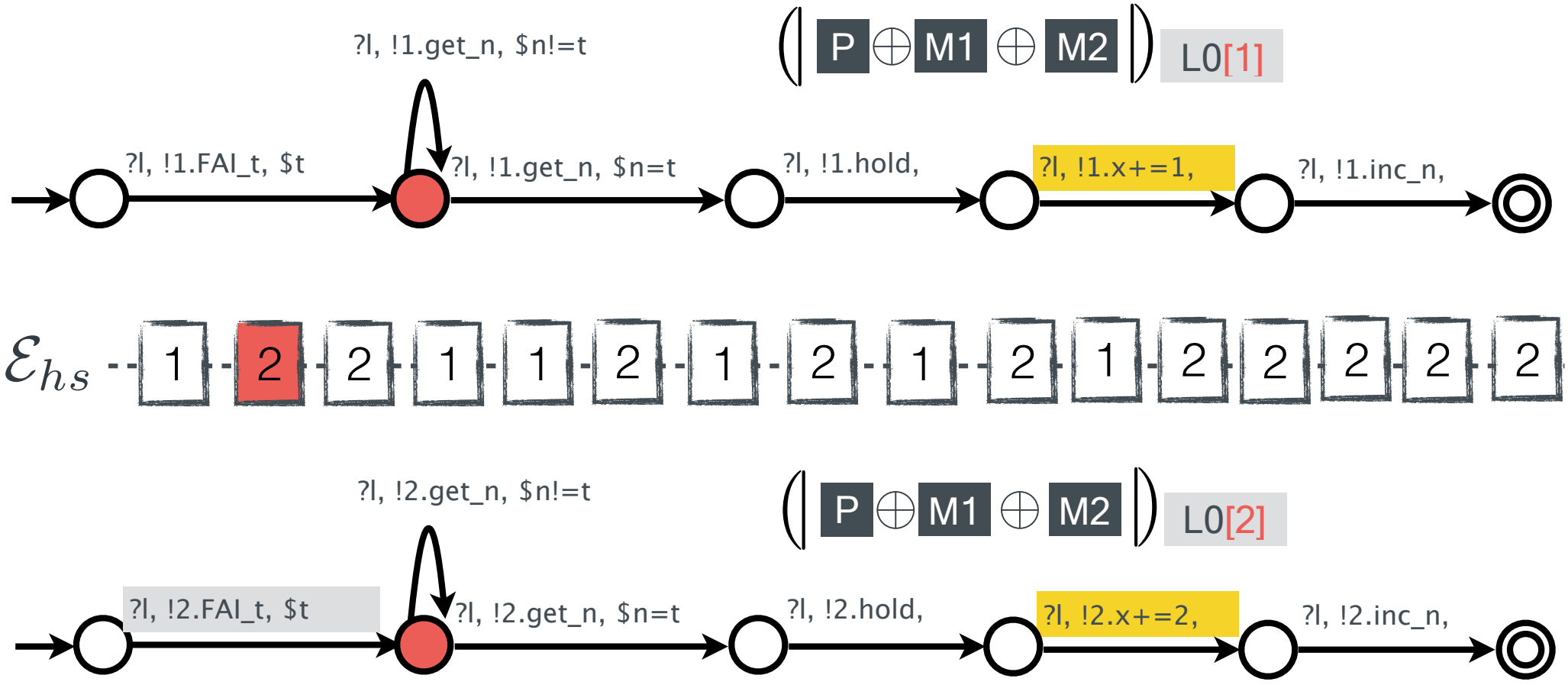

logical log l

# Strategies and Game Semantics



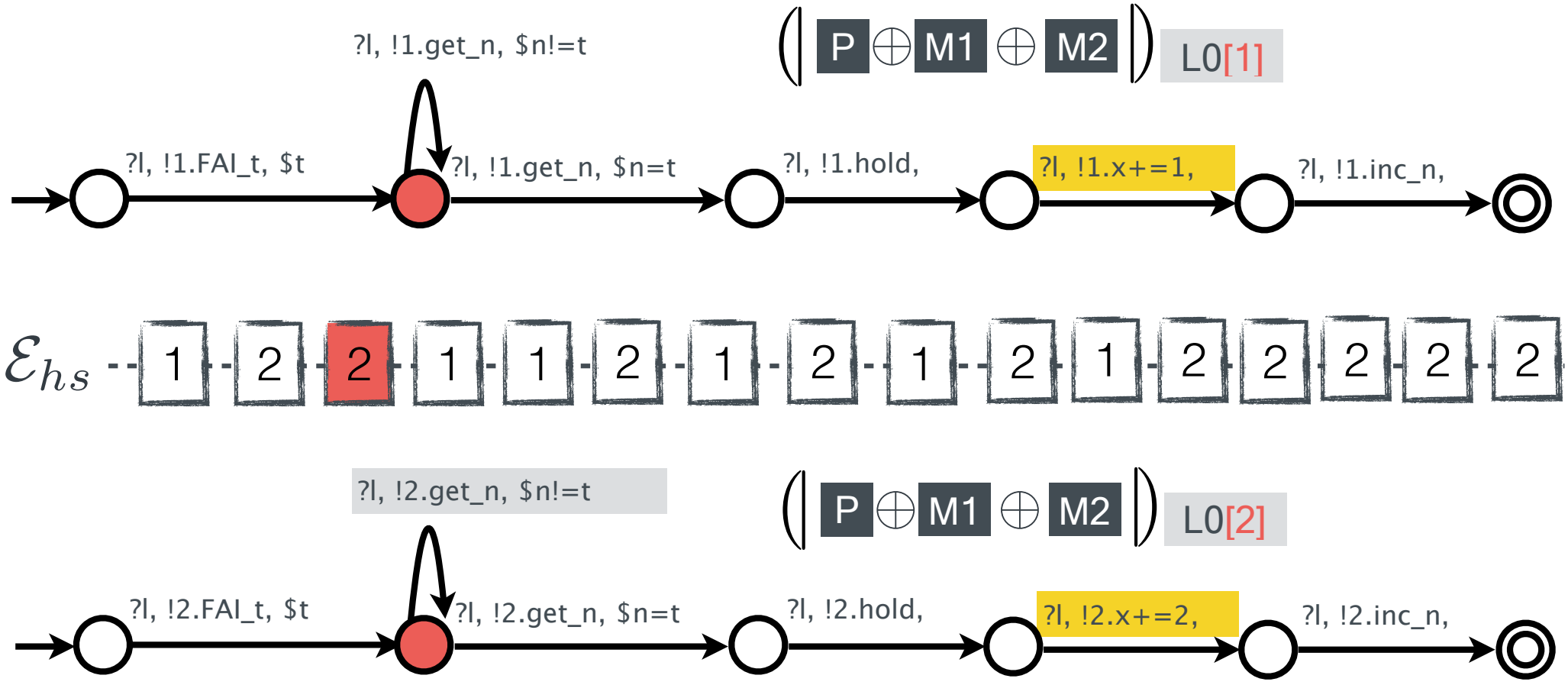

logical log l

# Strategies and Game Semantics



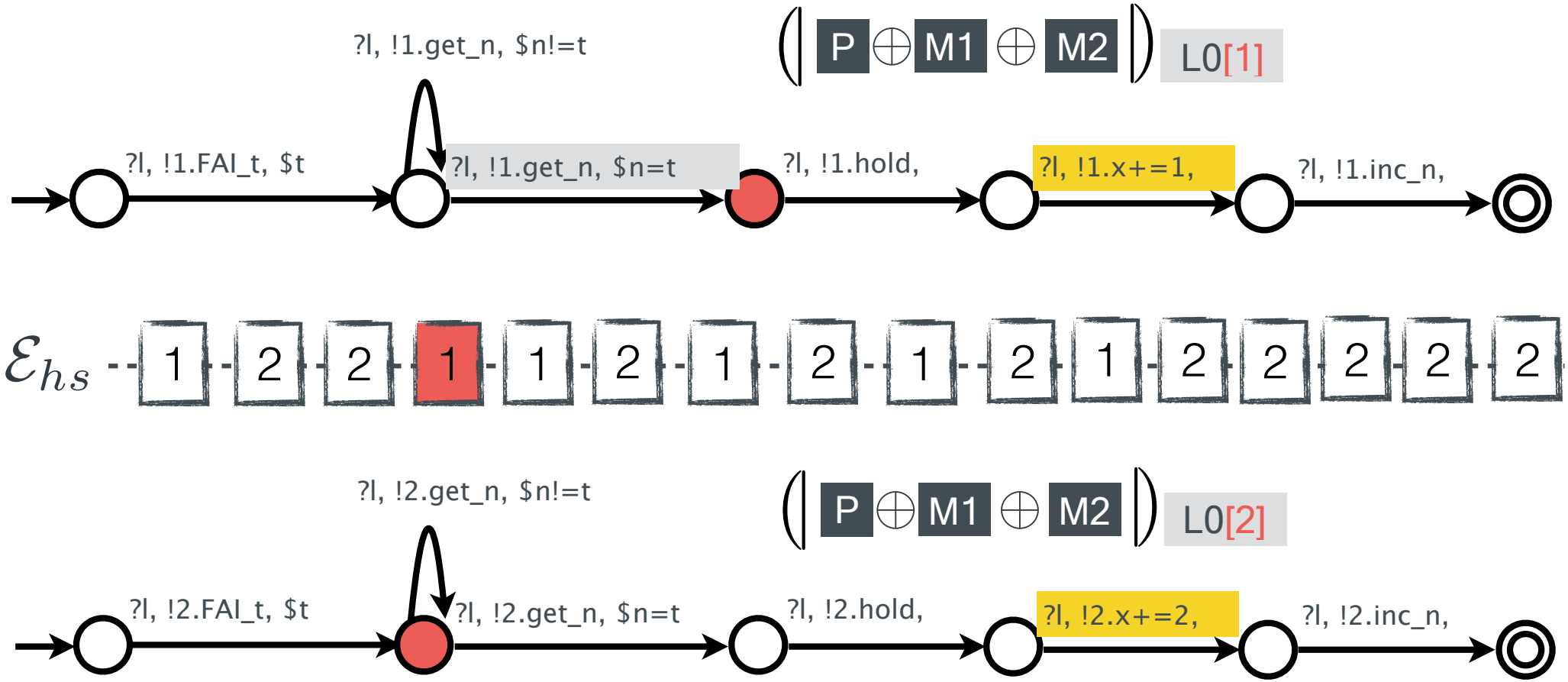

logical log l

# Strategies and Game Semantics



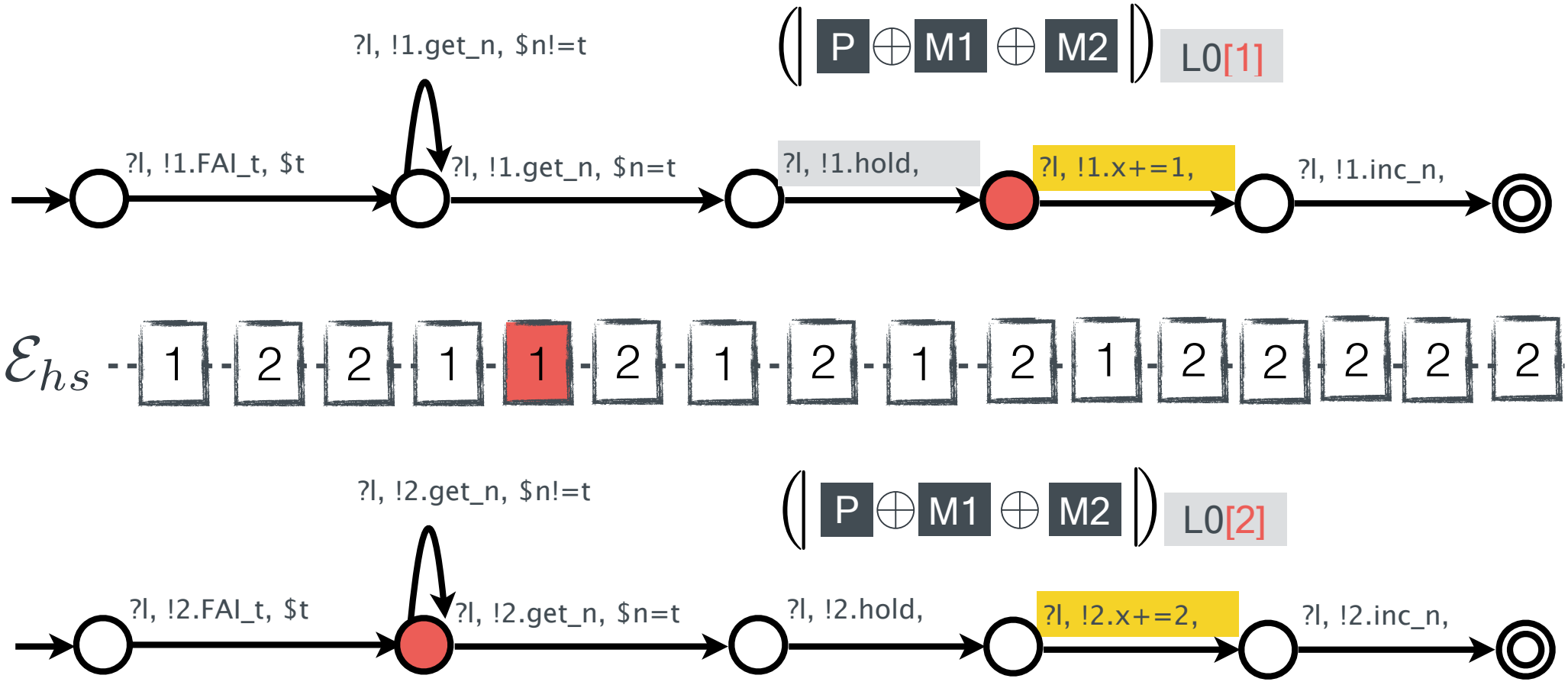

logical log I

# Strategies and Game Semantics



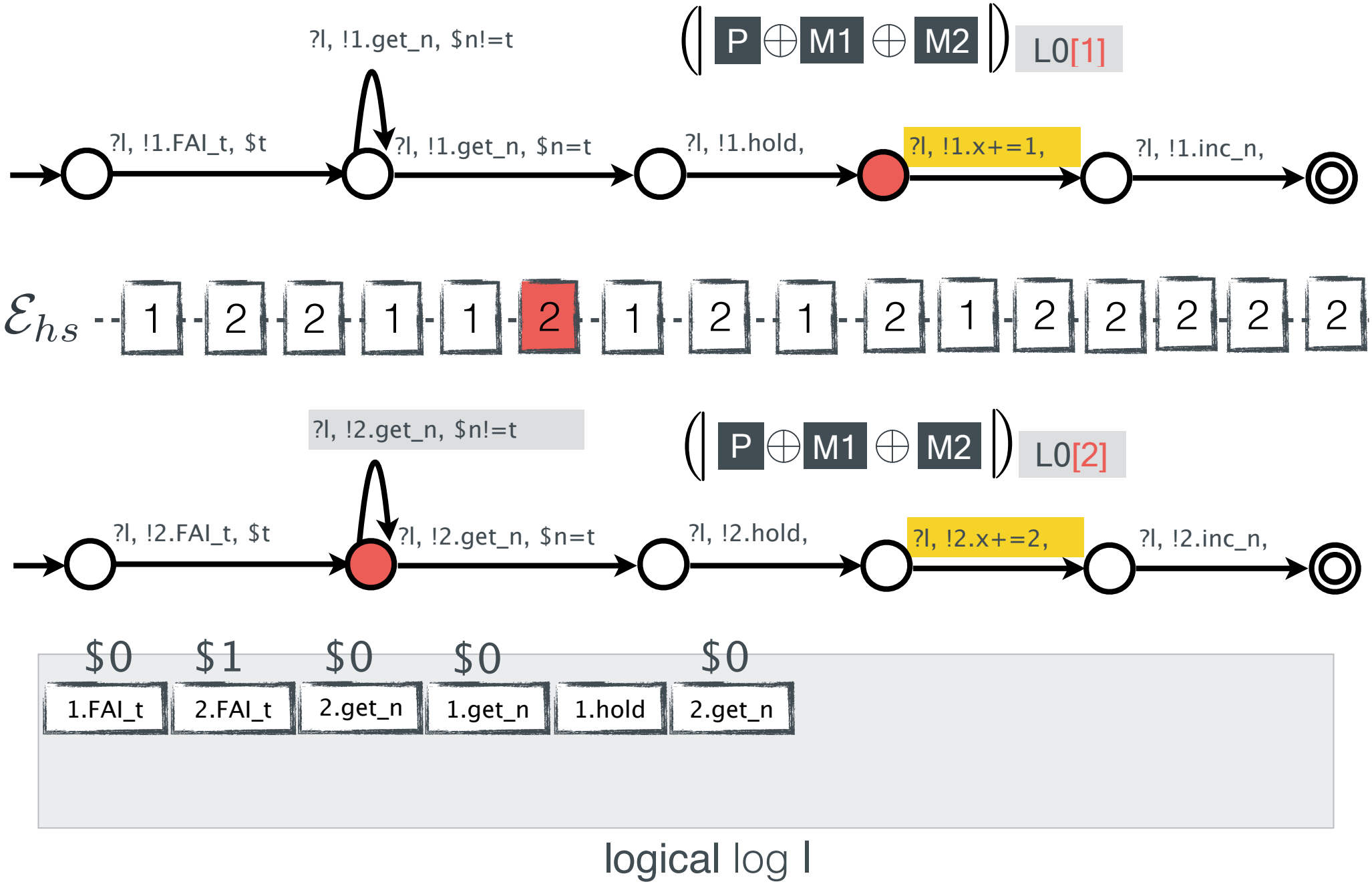

logical log l

# Strategies and Game Semantics

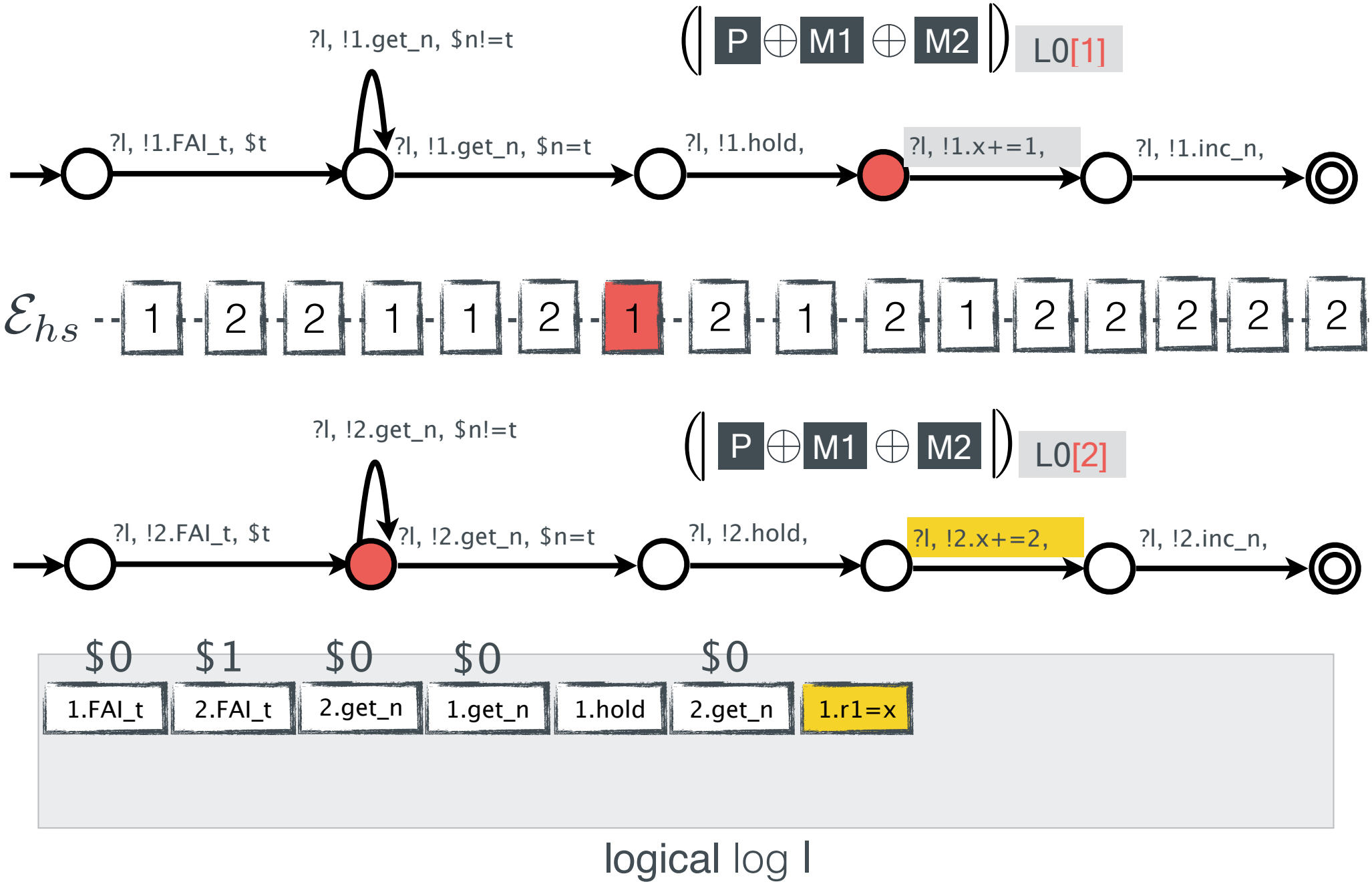



logical log l

# Strategies and Game Semantics

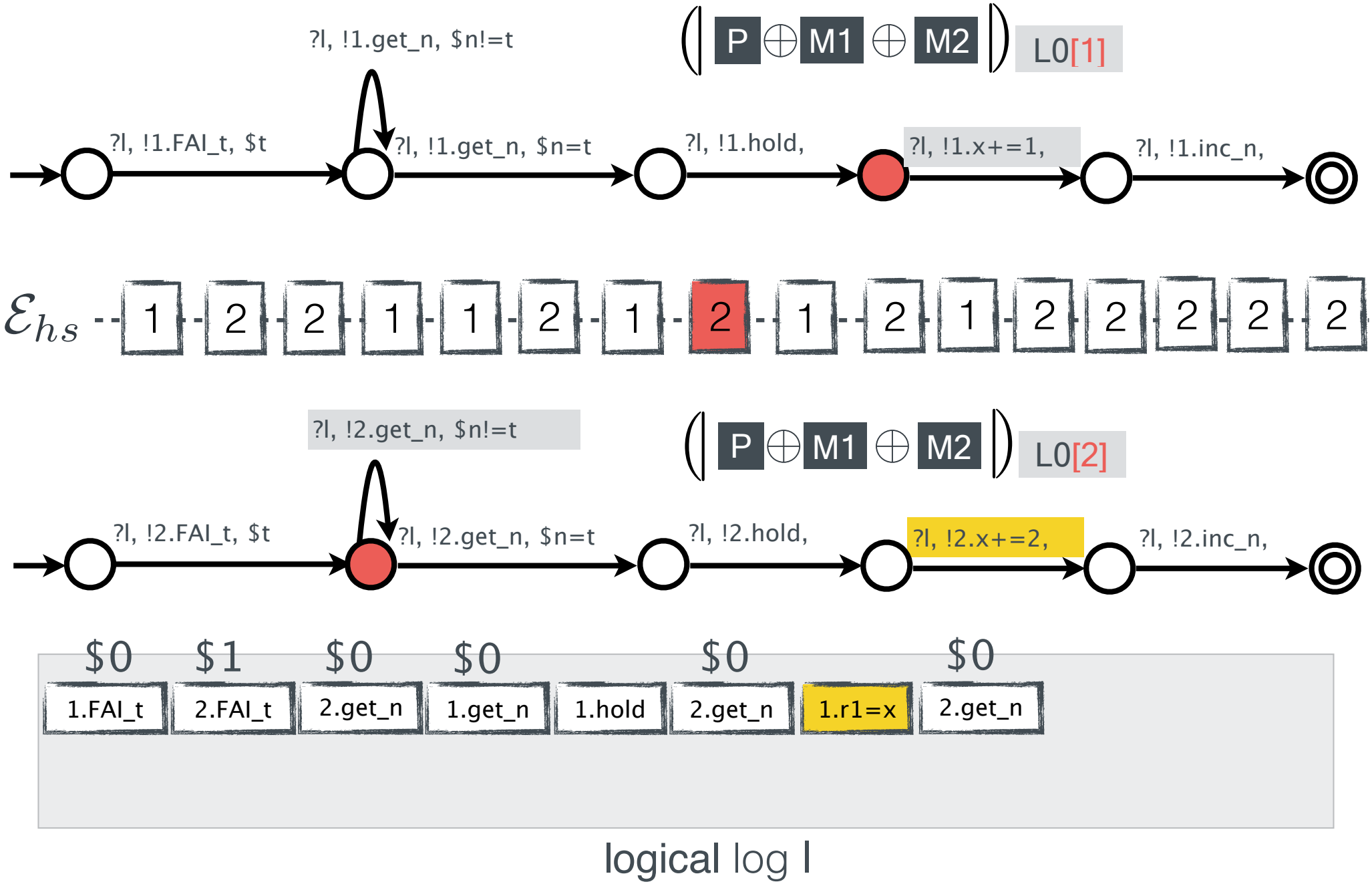


# Strategies and Game Semantics

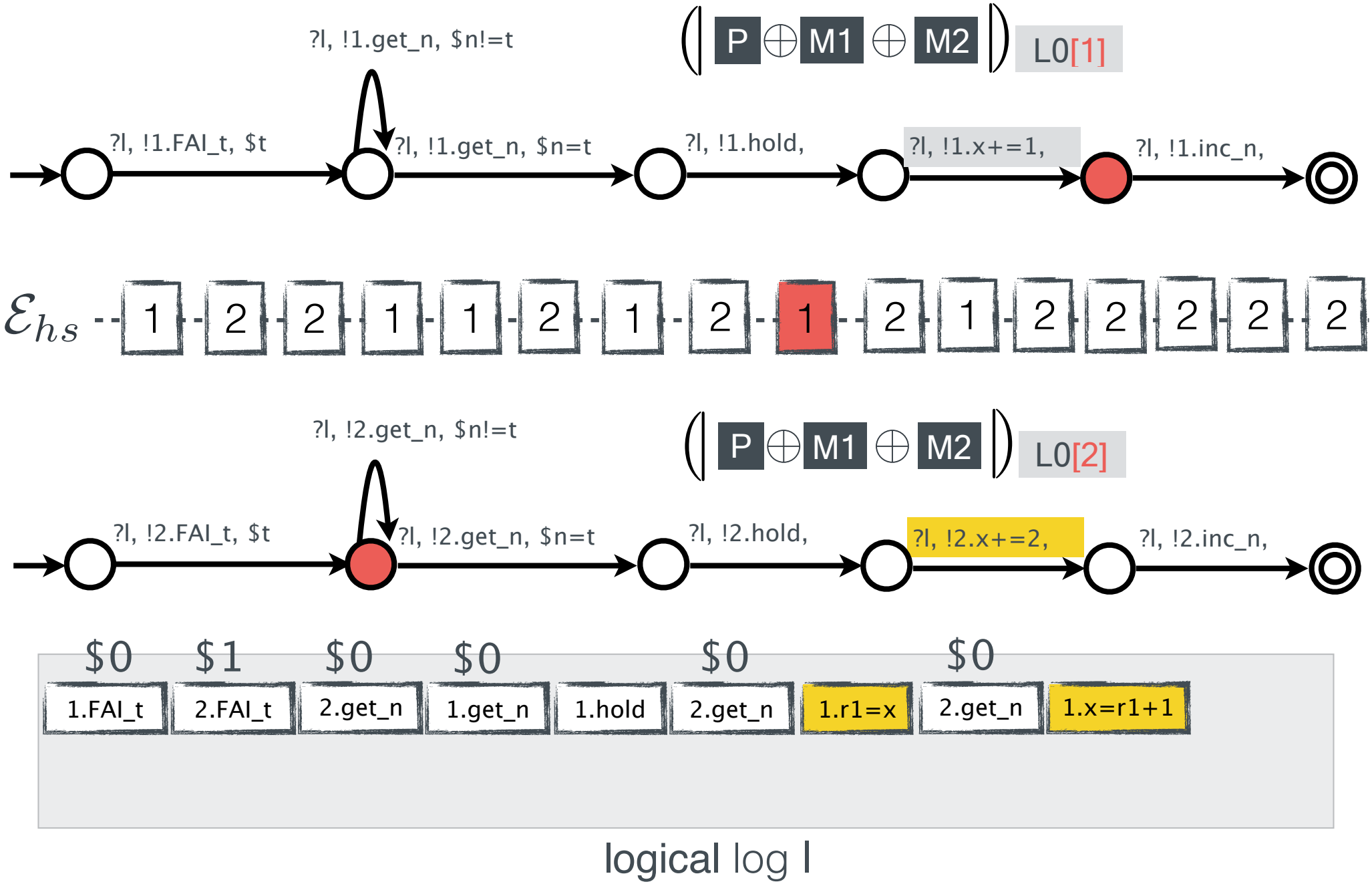



logical log l

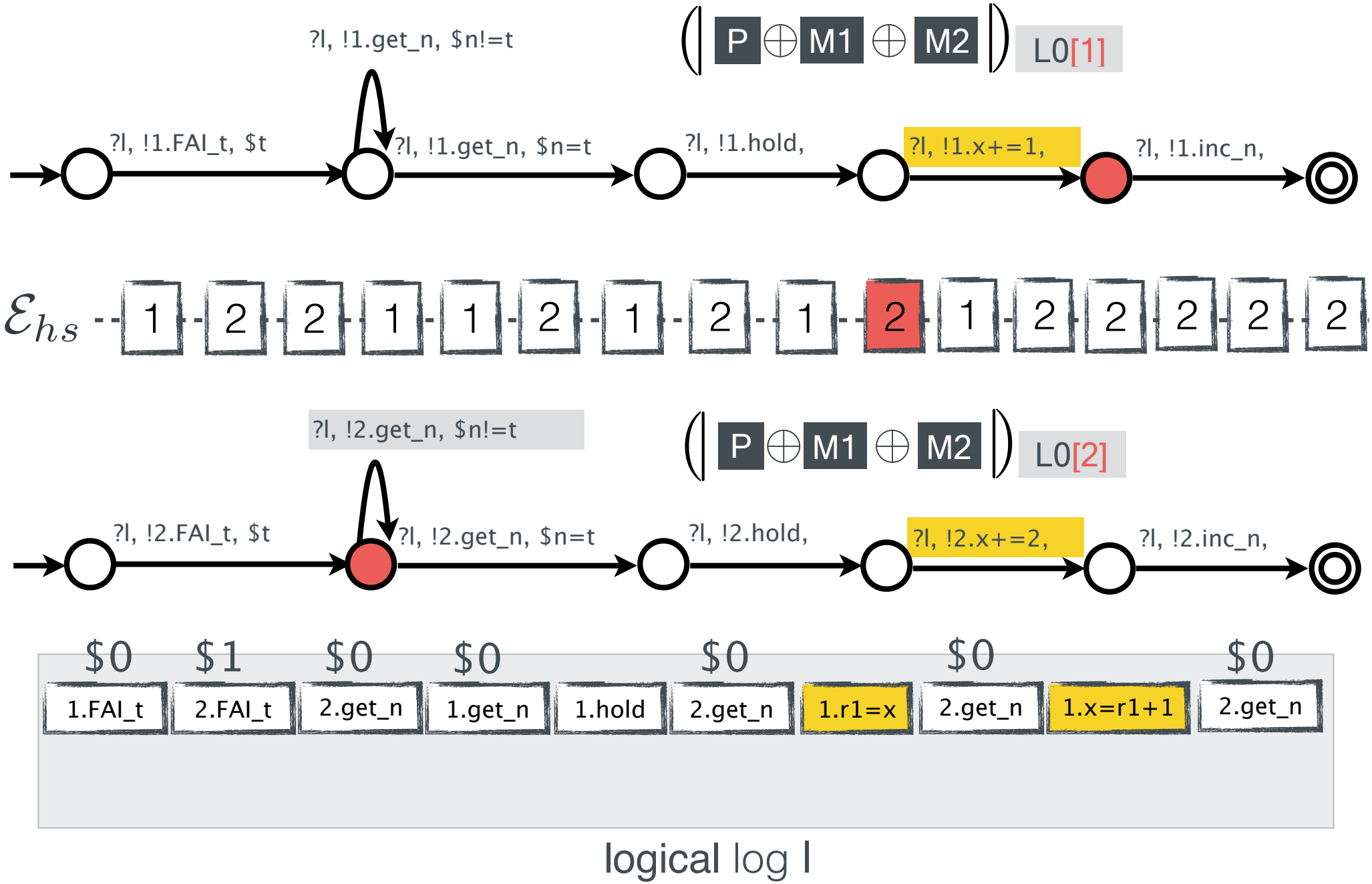
# Strategies and Game Semantics



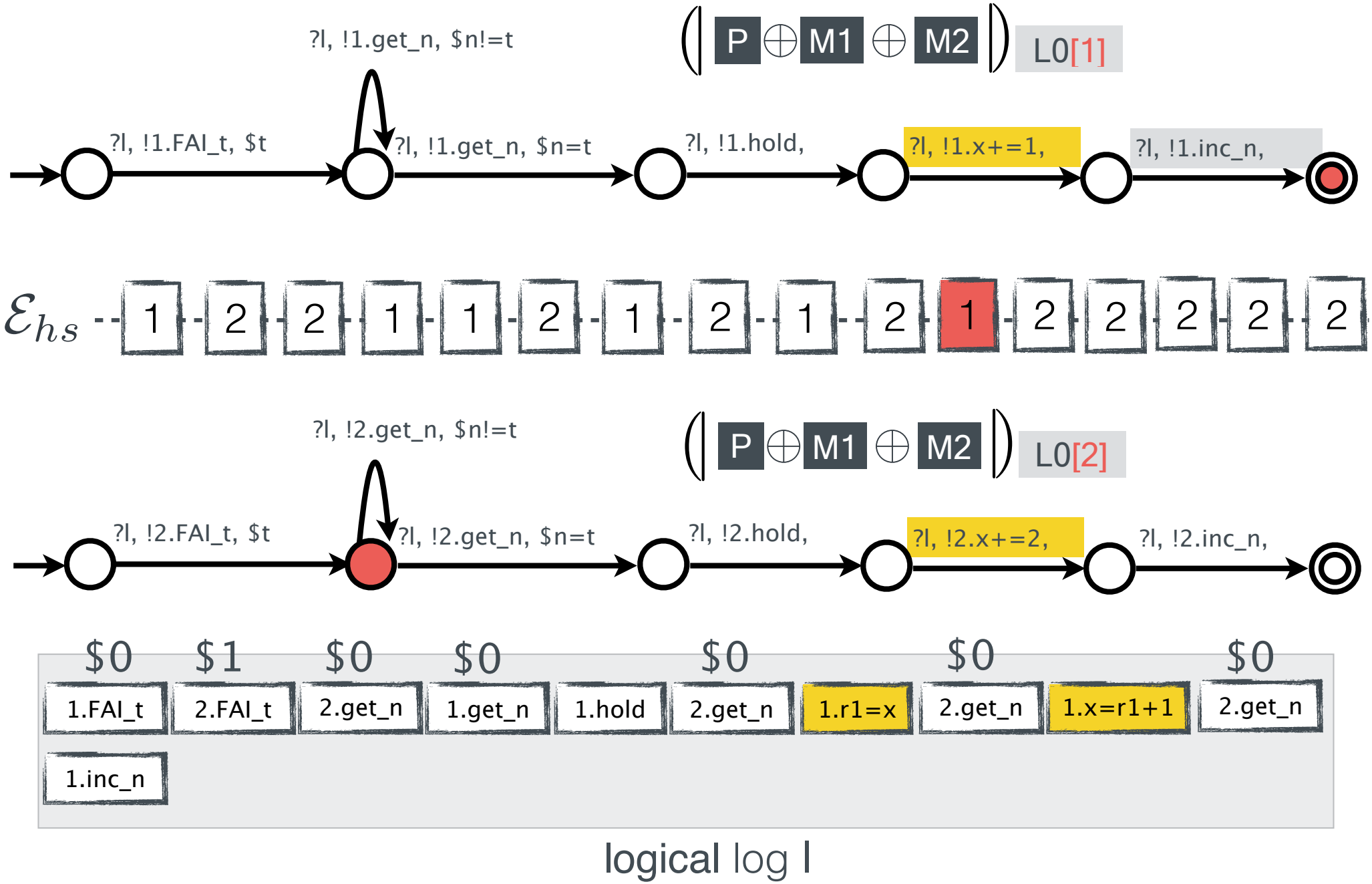
# Strategies and Game Semantics



# Strategies and Game Semantics

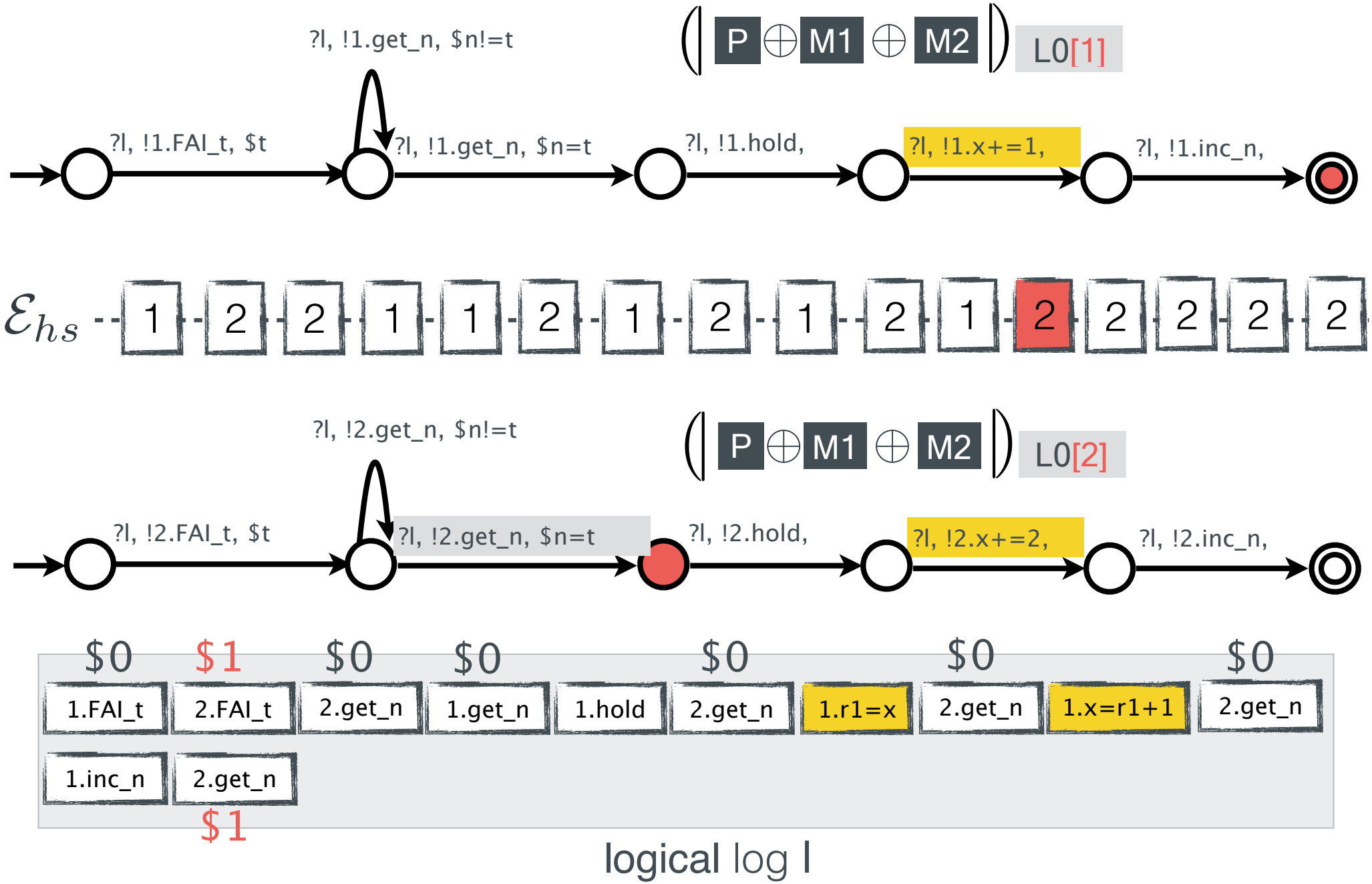


# Strategies and Game Semantics

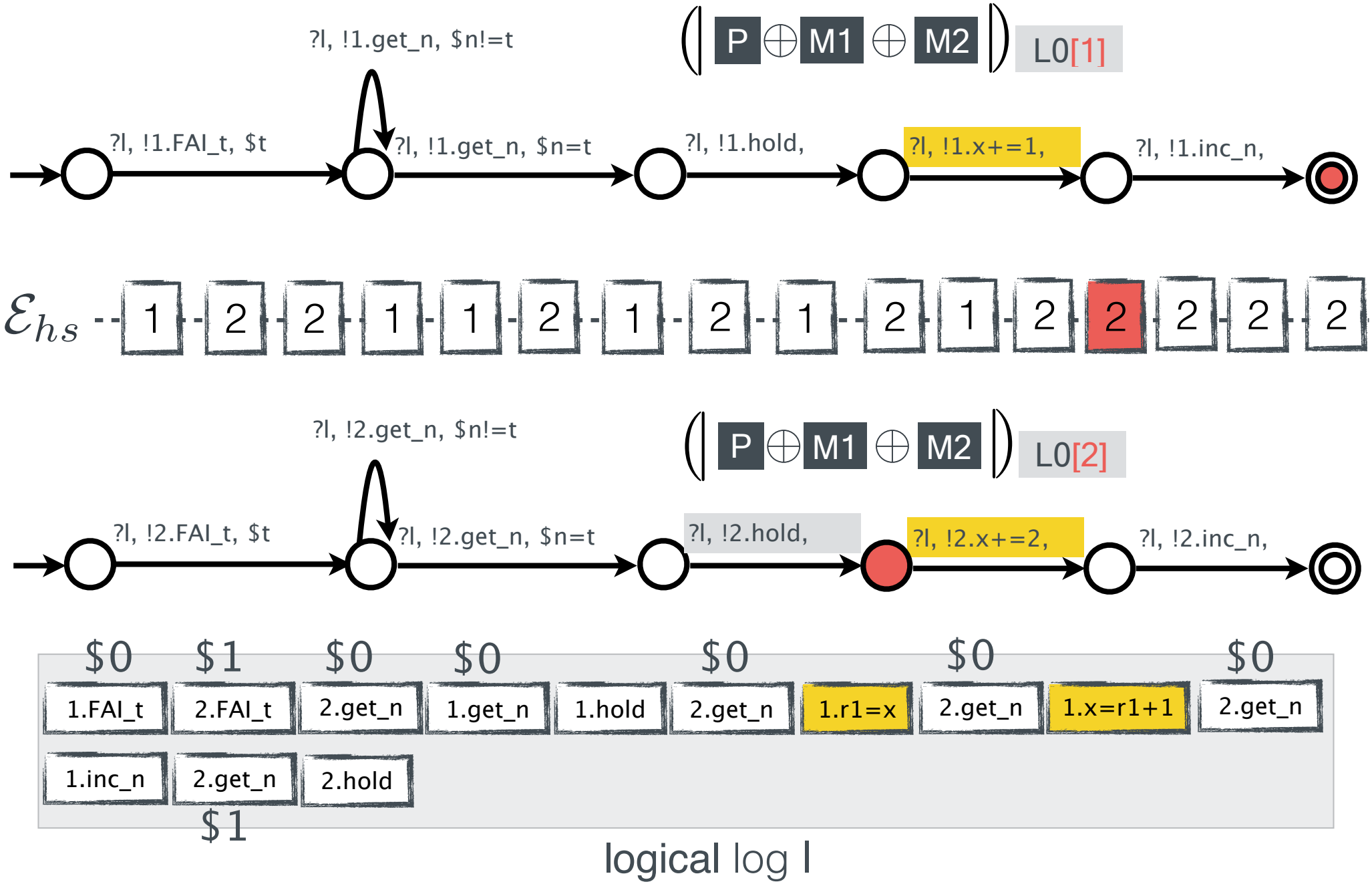



logical log l

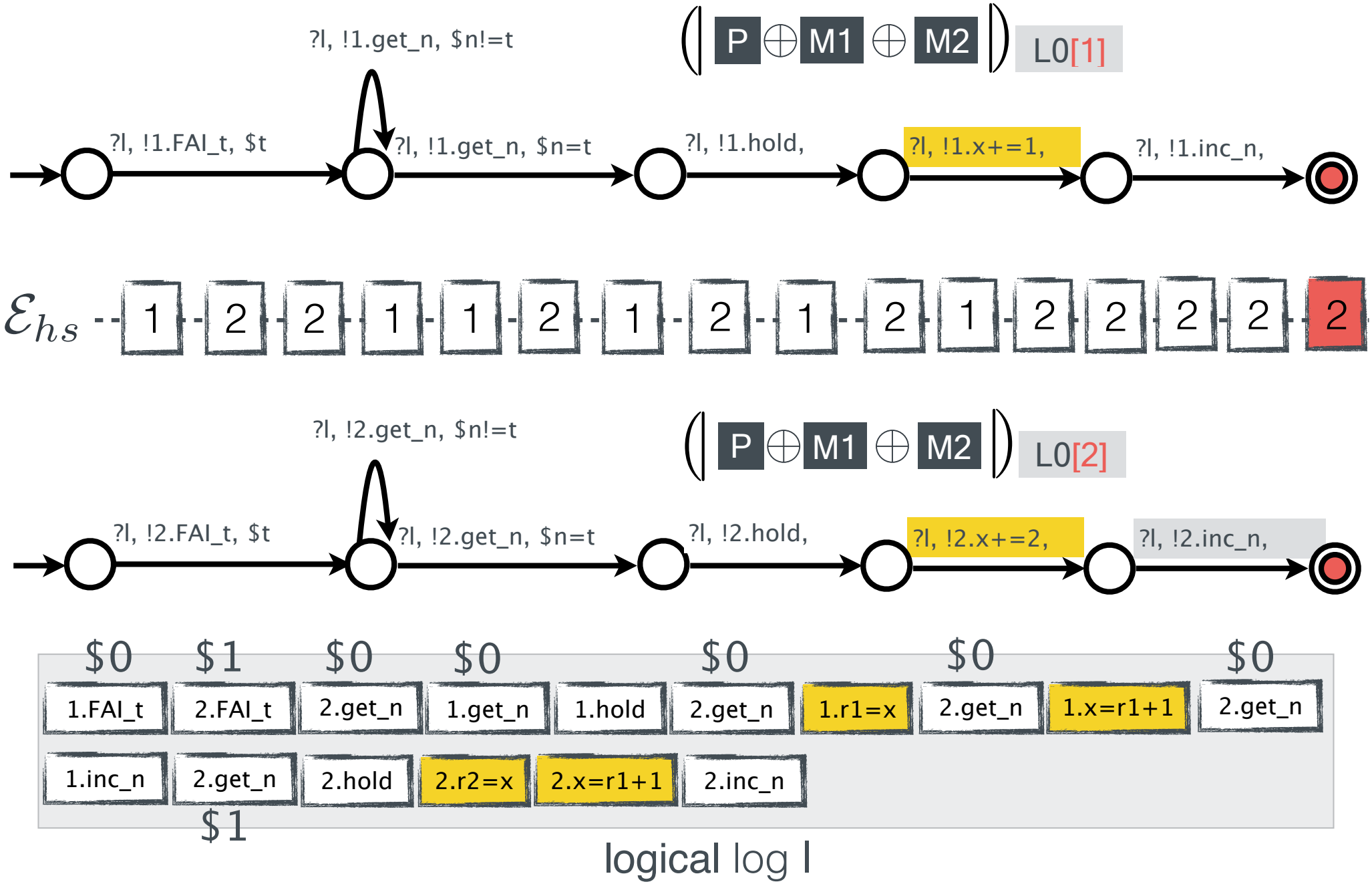
# Strategies and Game Semantics



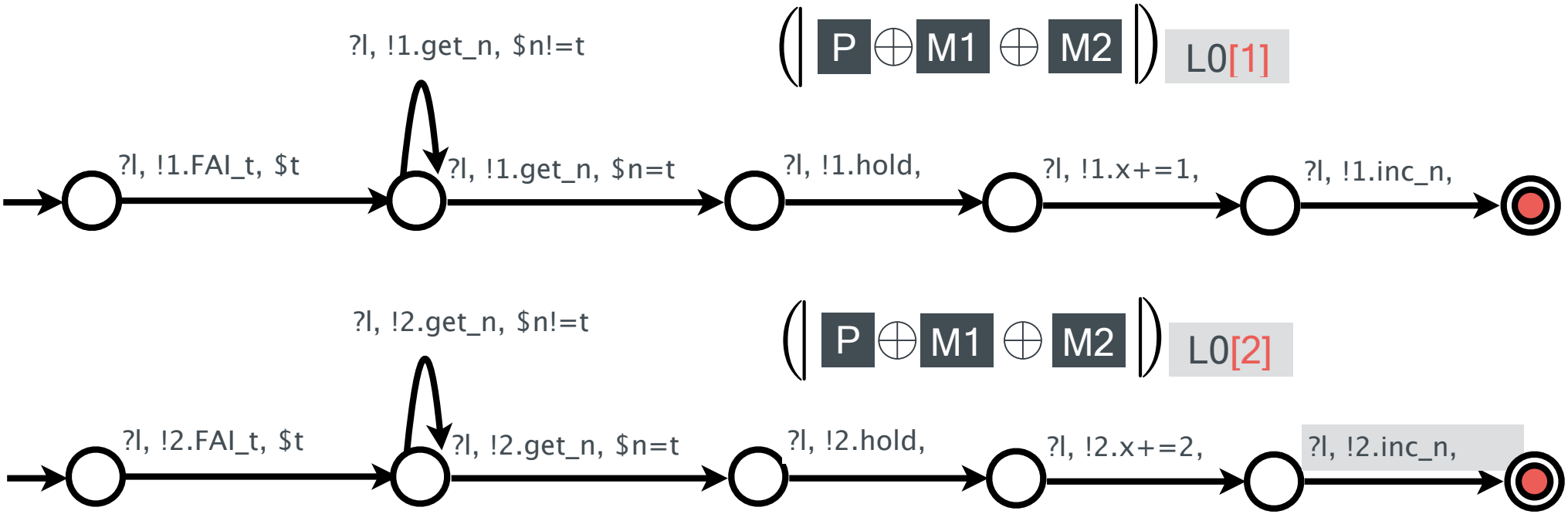
# Strategies and Game Semantics



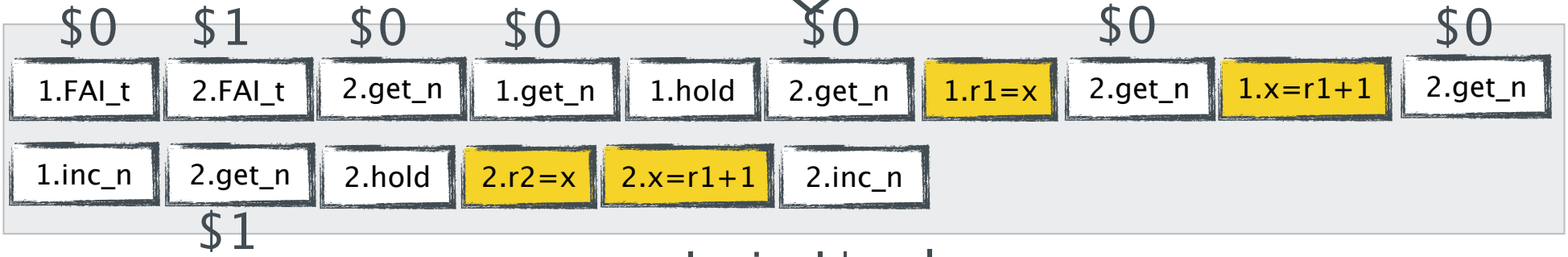
# Strategies and Game Semantics



# Strategies and Game Semantics

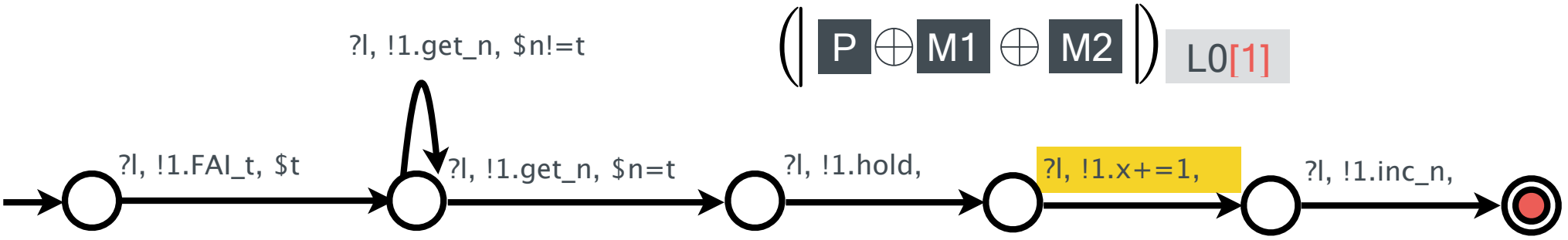




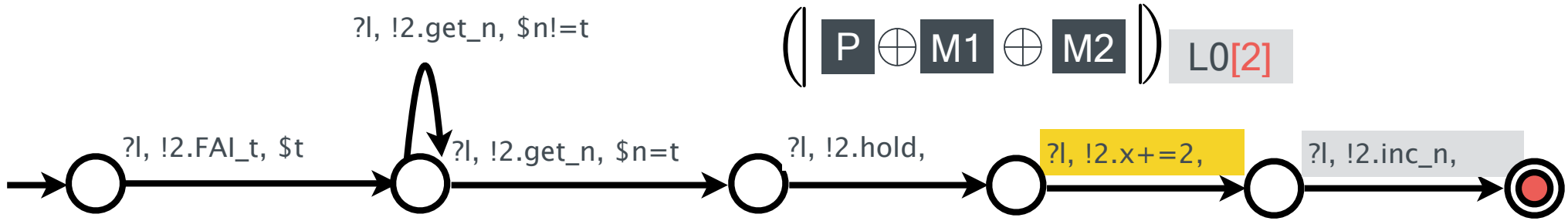



logical log l

# Strategies and Game Semantics



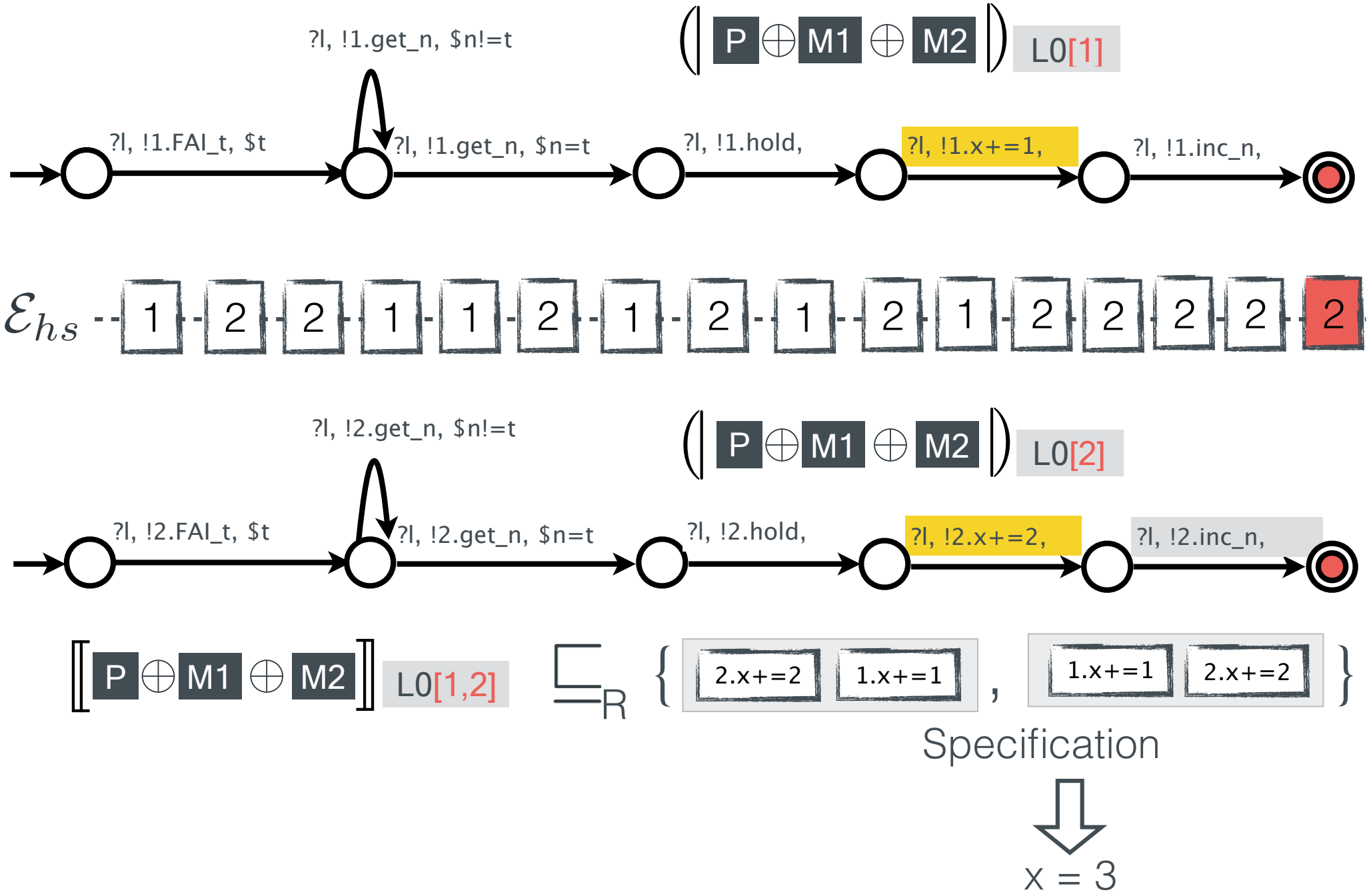




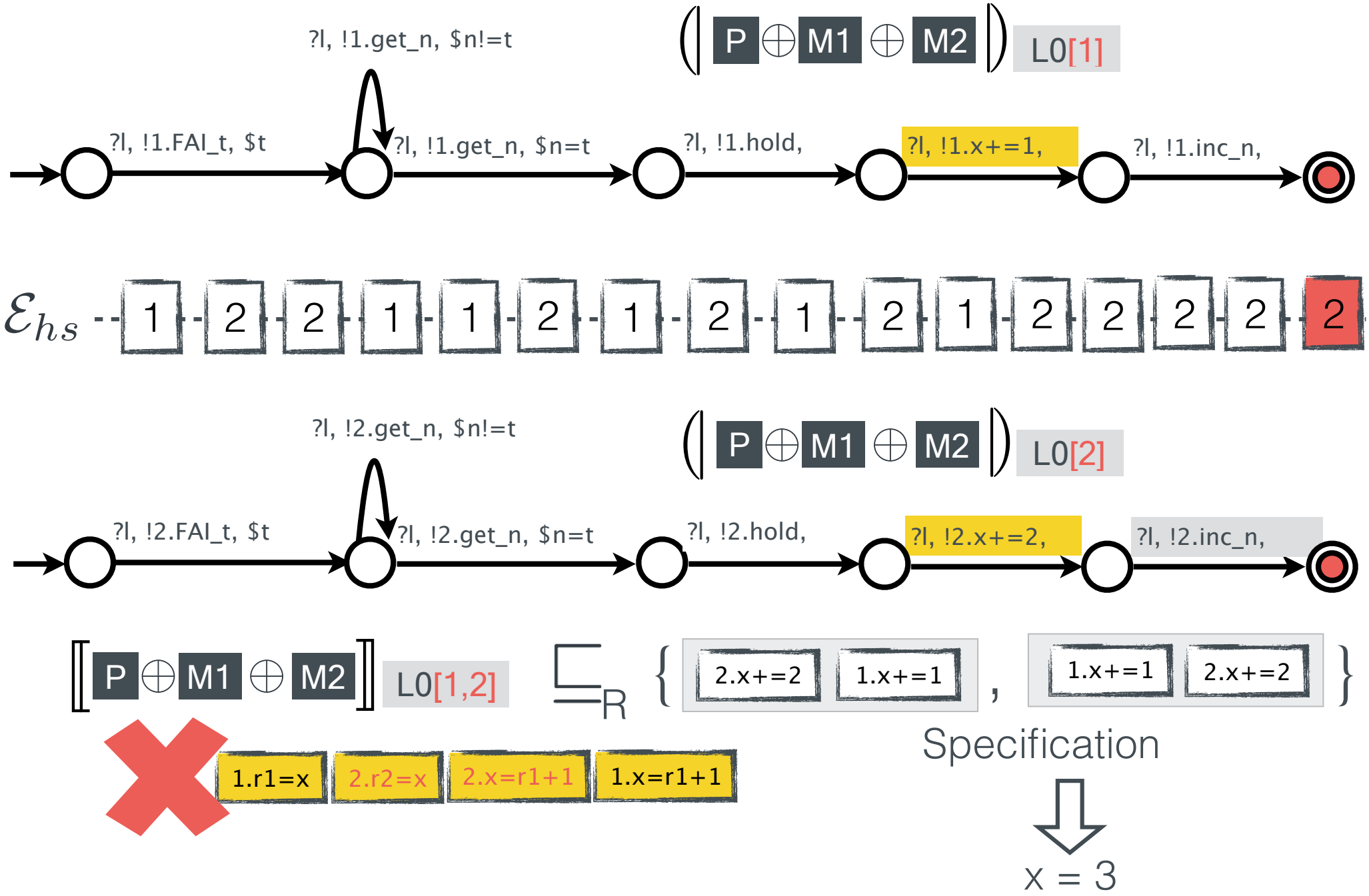

$[\![ P \oplus M1 \oplus M2 ]\!]$ L0[1,2] := { , , , , , }

Set of **logical** logs

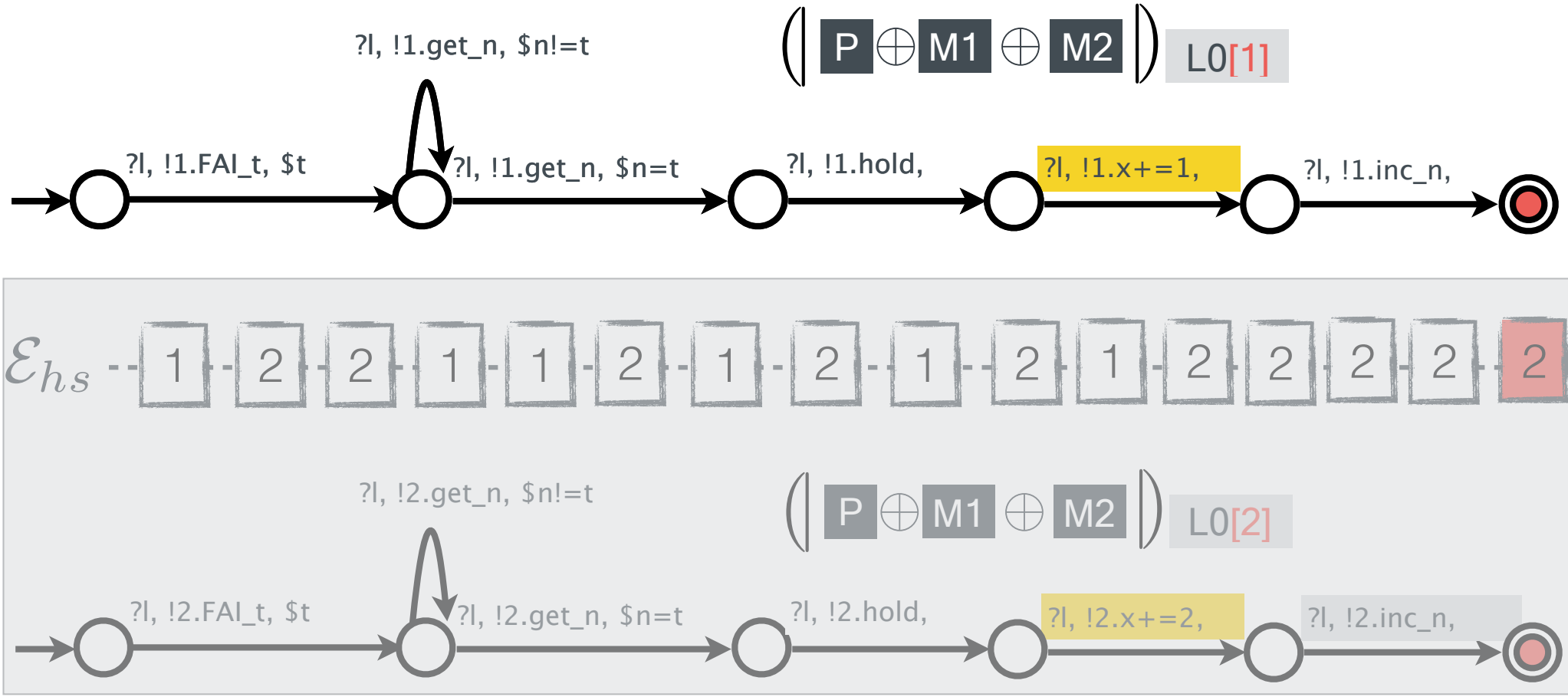
# Strategies and Game Semantics



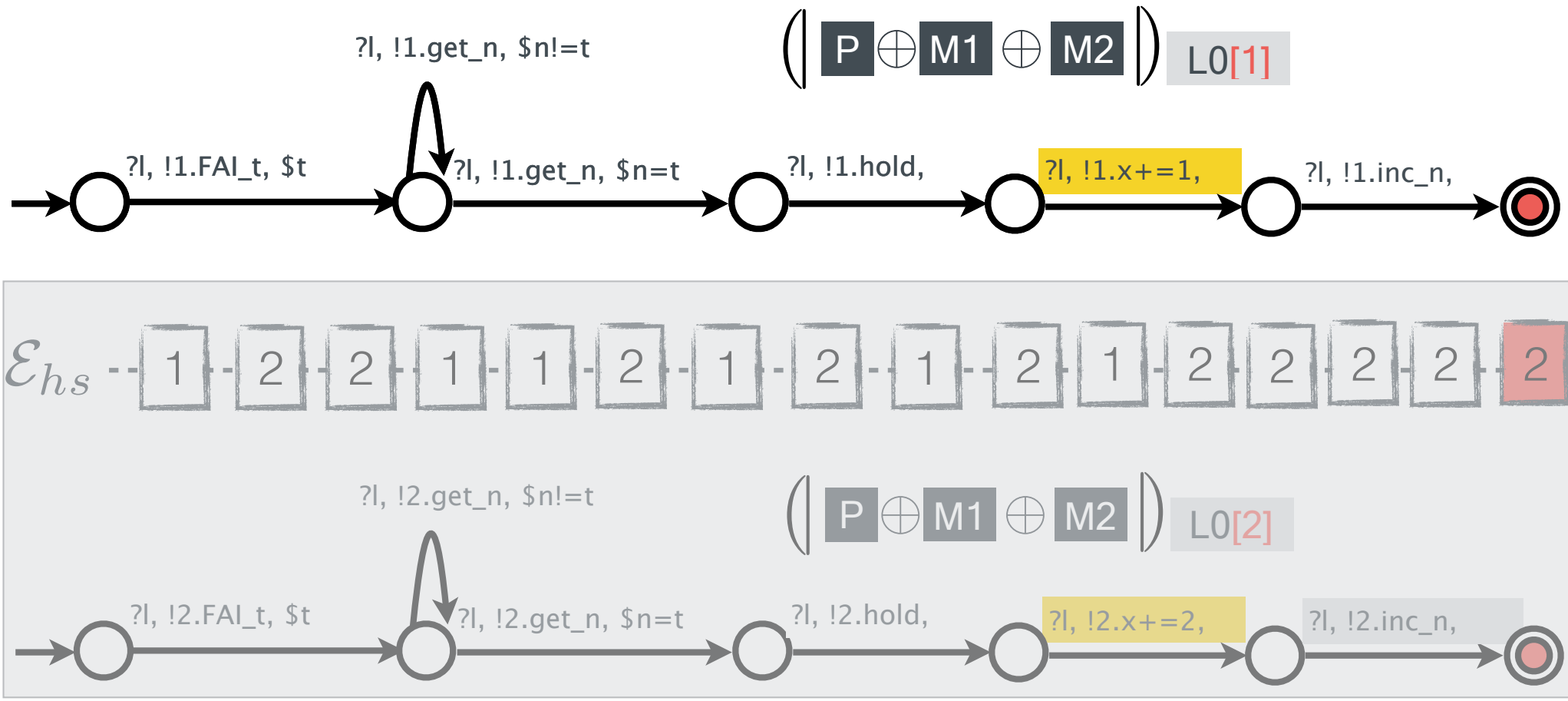
# Strategies and Game Semantics



# Strategy Refinement



# Strategy Refinement



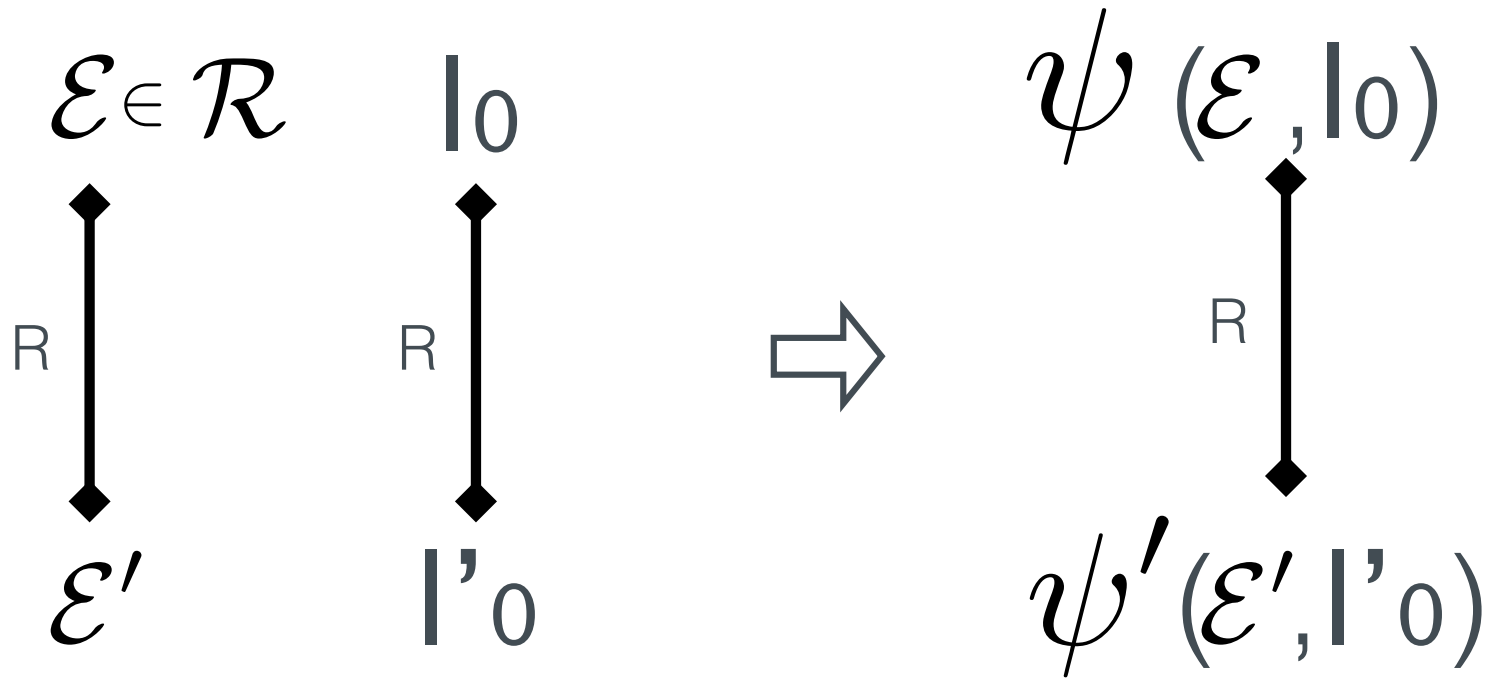

**Environment Context** $\mathcal{E} \in \mathcal{R}$

# Strategy Refinement

$$\psi\,(\varepsilon\,,\mathrm{I}_0)$$

# Strategy Refinement

$$\psi \leq_R \psi'$$

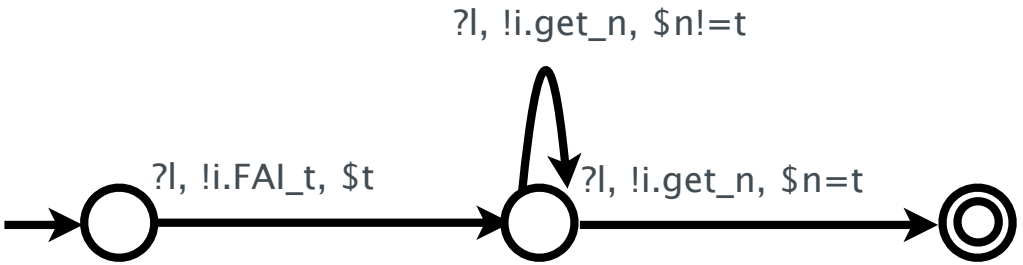


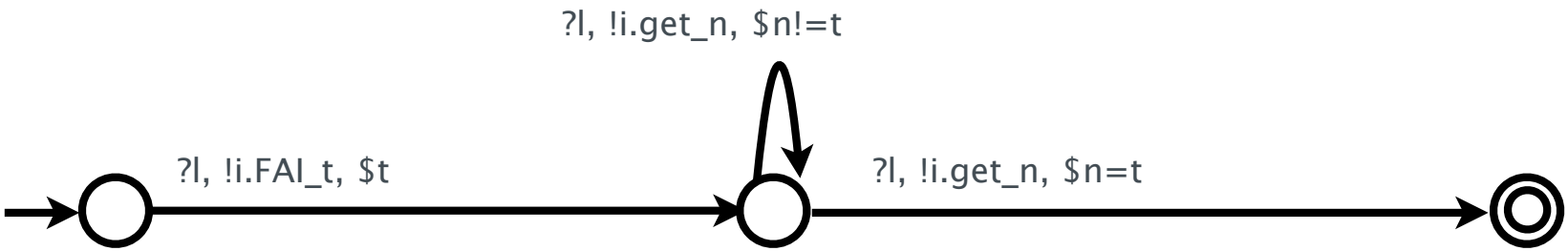
# Strategy Refinement

( Macq ) Lo[i]

$\leq_R$

$\psi_{acq}[i]$



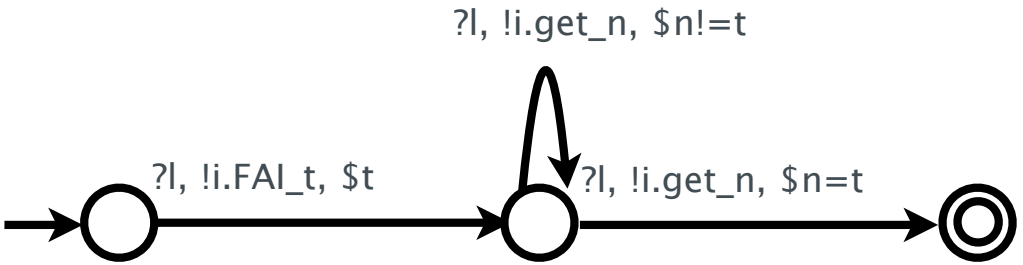



# Strategy Refinement

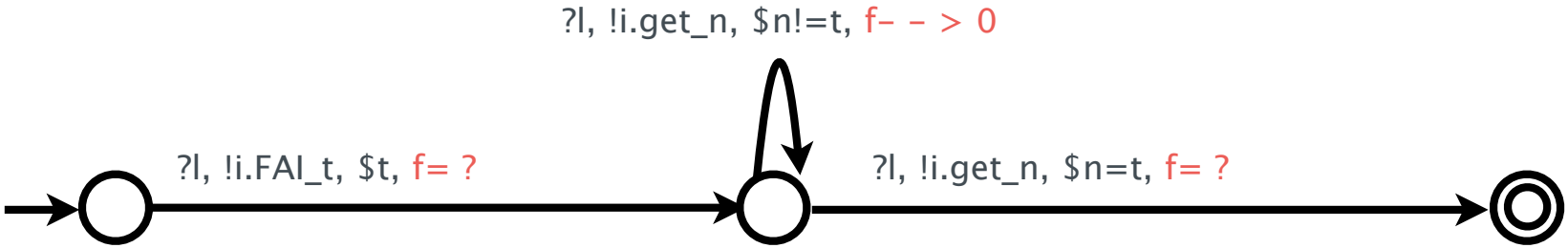
( Macq ) L0[i]

$\leq_R$

$\psi_{acq}$[i]



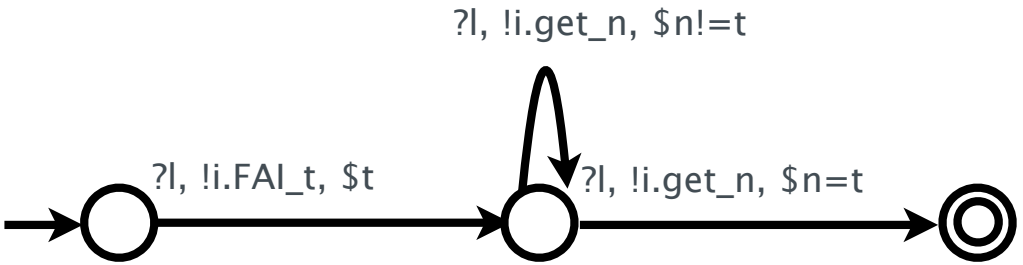

Add fuel (f) to prove liveness



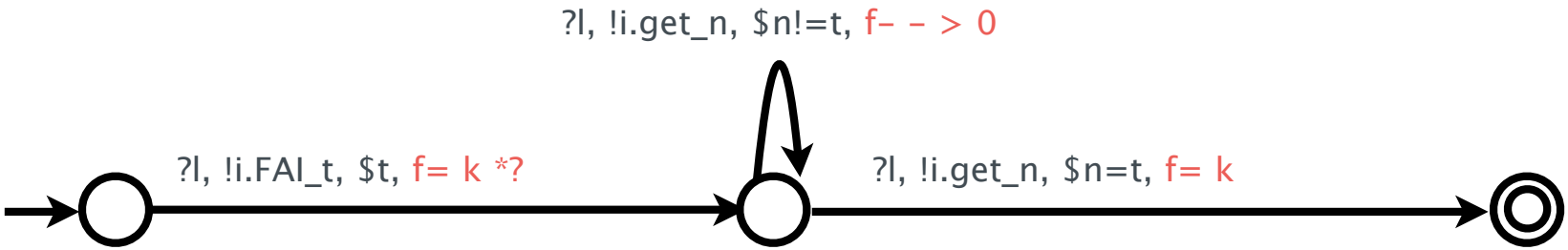
# Strategy Refinement


$\leq_R$

$\psi_{acq}[i]$




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

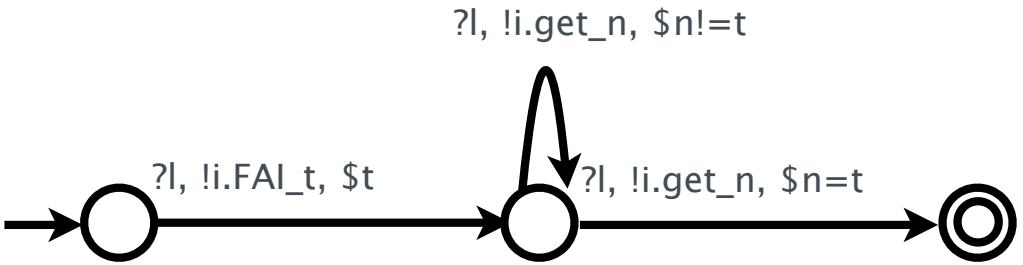


# Strategy Refinement

( Macq ) L0[i]

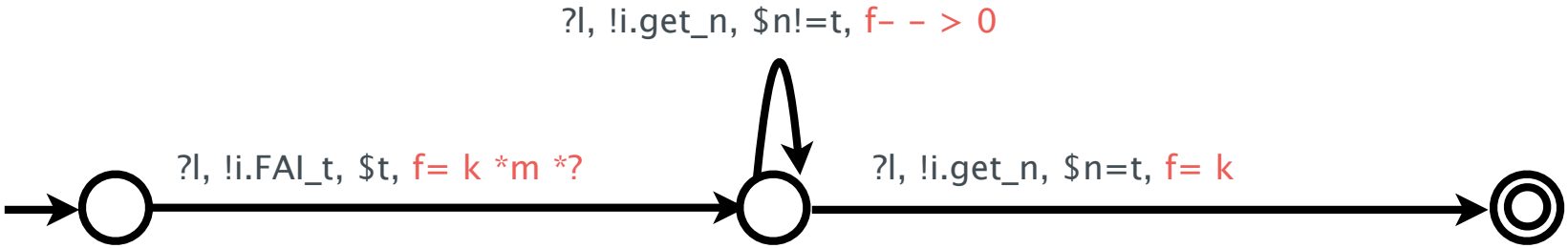
$\leq_R$

$\psi_{acq}[i]$




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

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

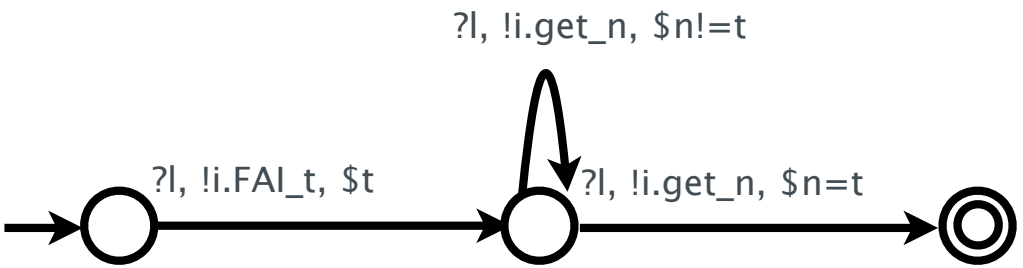


# Strategy Refinement

⟨ Macq ⟩ L0[i]

$\leq_R$

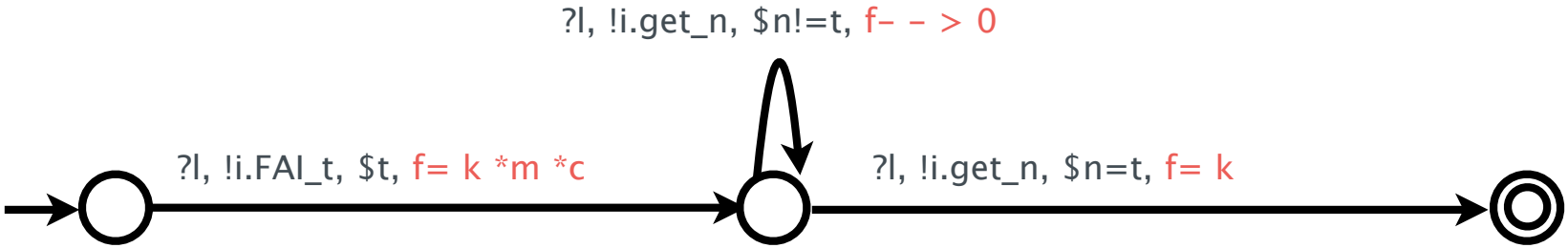
$\psi_{acq}$[i]




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

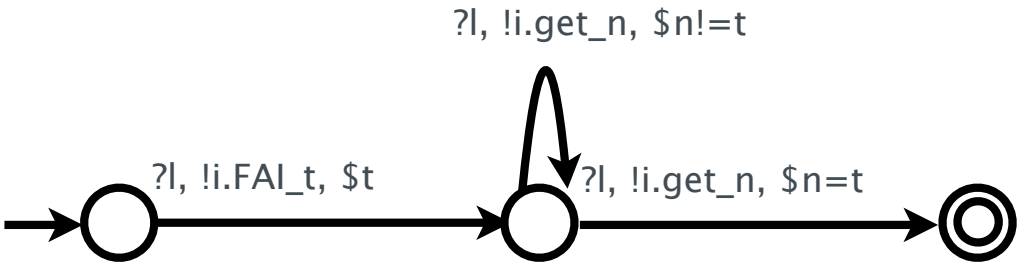


# Strategy Refinement

⟨ Macq ⟩ L0[i]

$\leq_R$

$\psi_{acq}$[i]

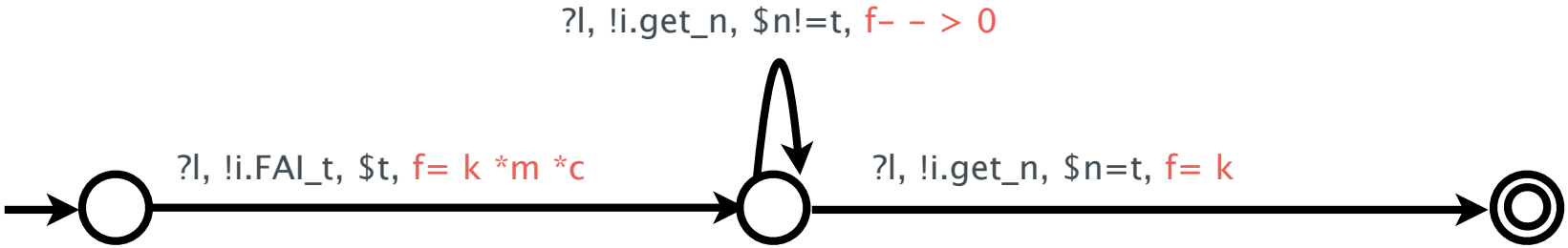



$\mathcal{R}_{j \neq i}$ : will release lock within k steps

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

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

mutual exclusion?

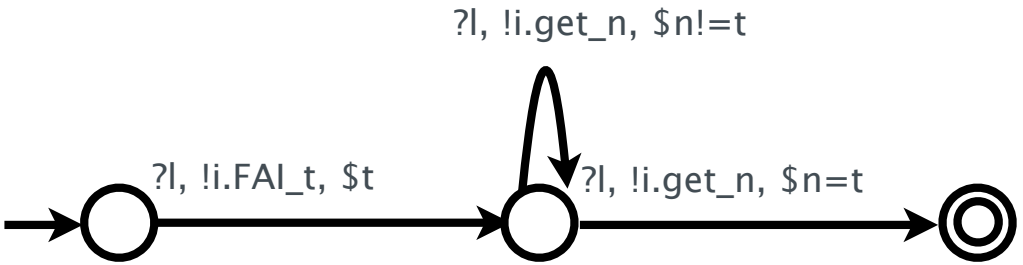


# Strategy Refinement

⟨ Macq ⟩ L0[i]

$\leq_R$

$\psi_{acq}$[i]

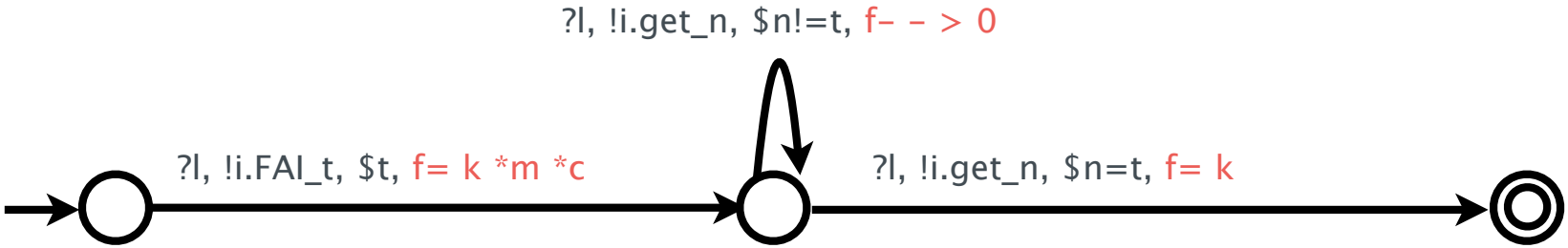



$\mathcal{R}_{j \neq i}$ : will release lock within k steps

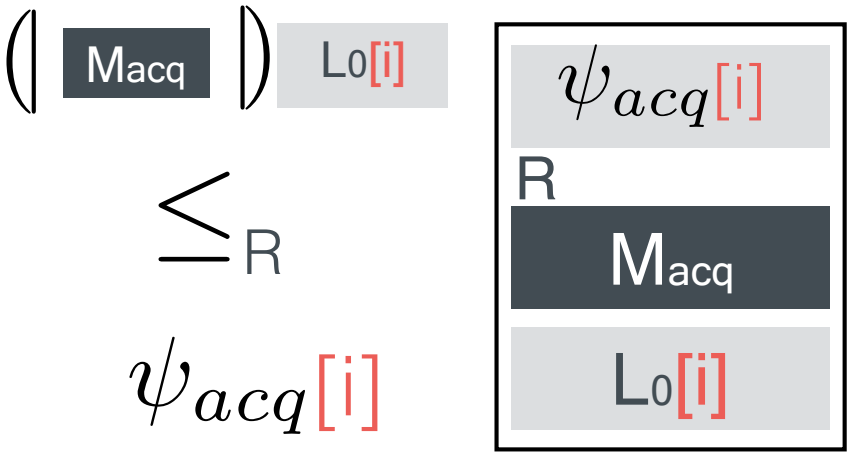
$\mathcal{R}_{hs}$ : (fairness) each CPU will be rescheduled within m steps

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

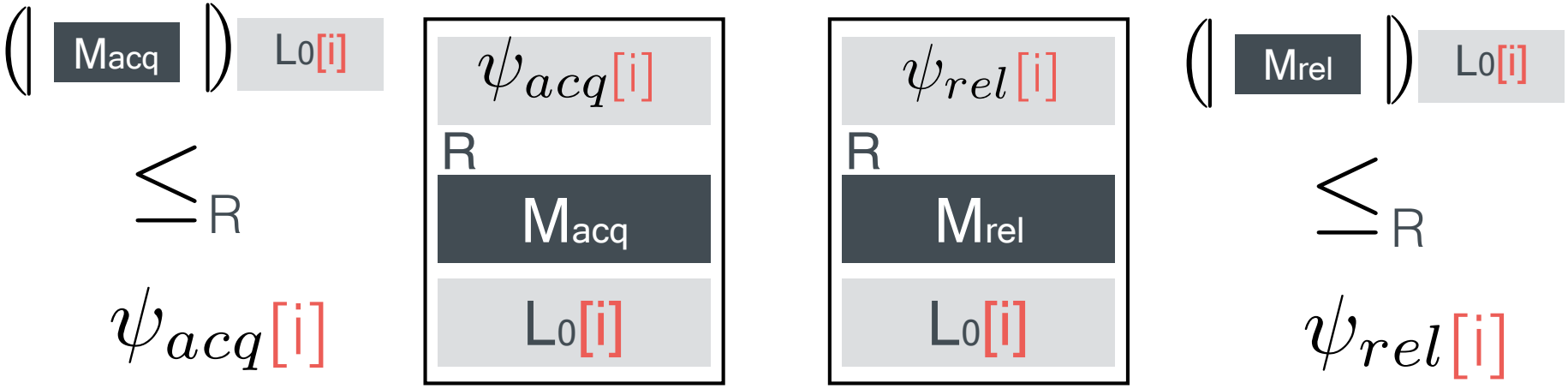
mutual exclusion?



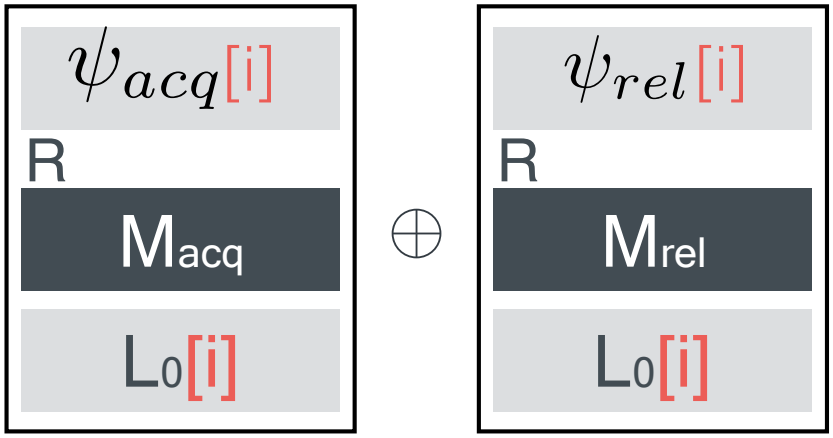
# Certified Concurrent Abstraction Layer



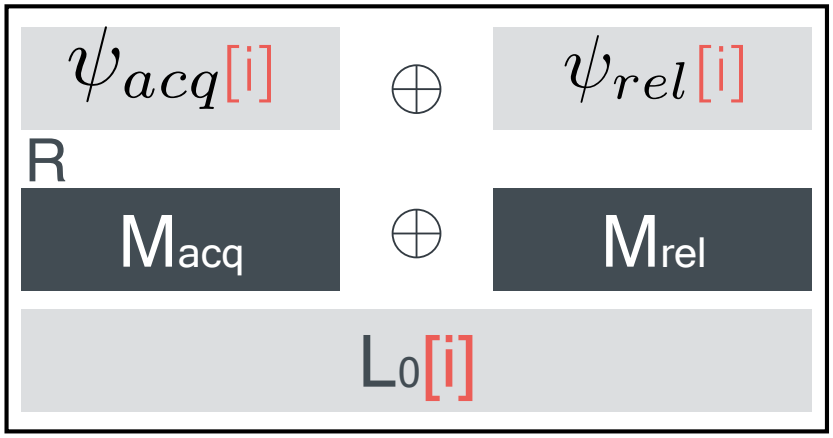
# Certified Concurrent Abstraction Layer



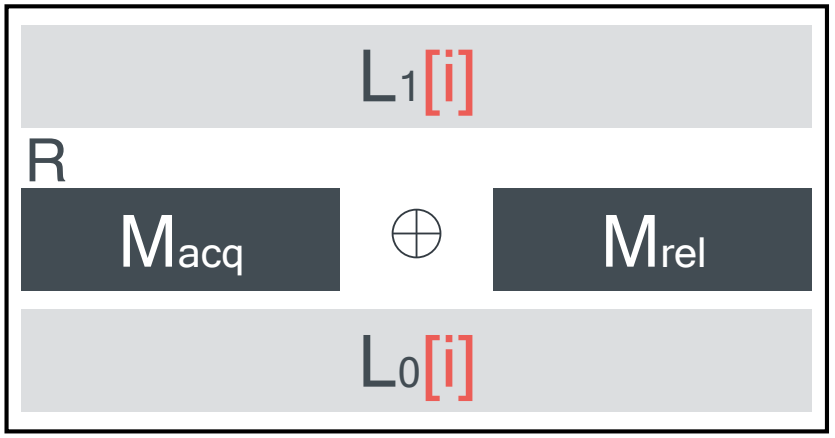
# Horizontal Composition



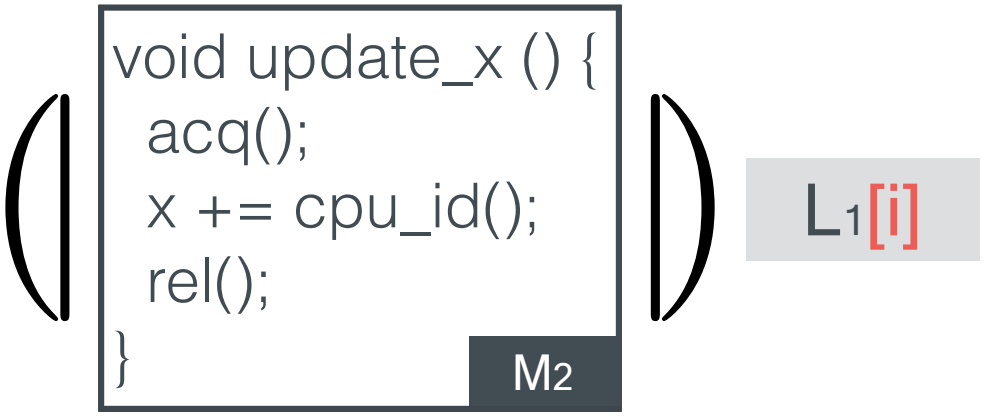
# Horizontal Composition

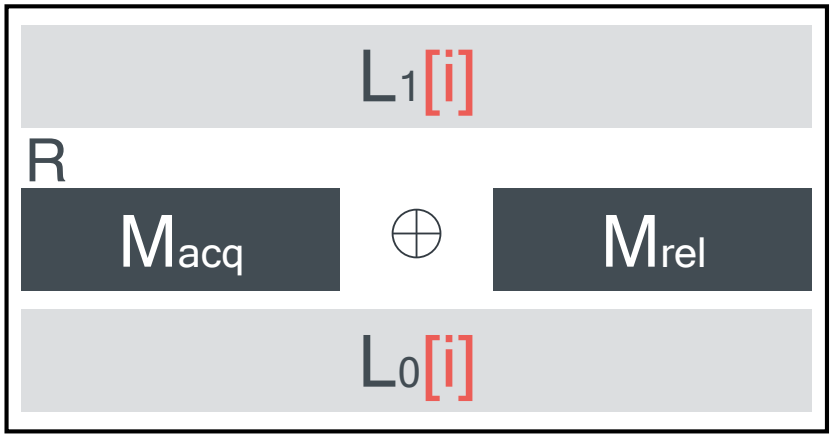


# Certified Concurrent Abstraction Layer

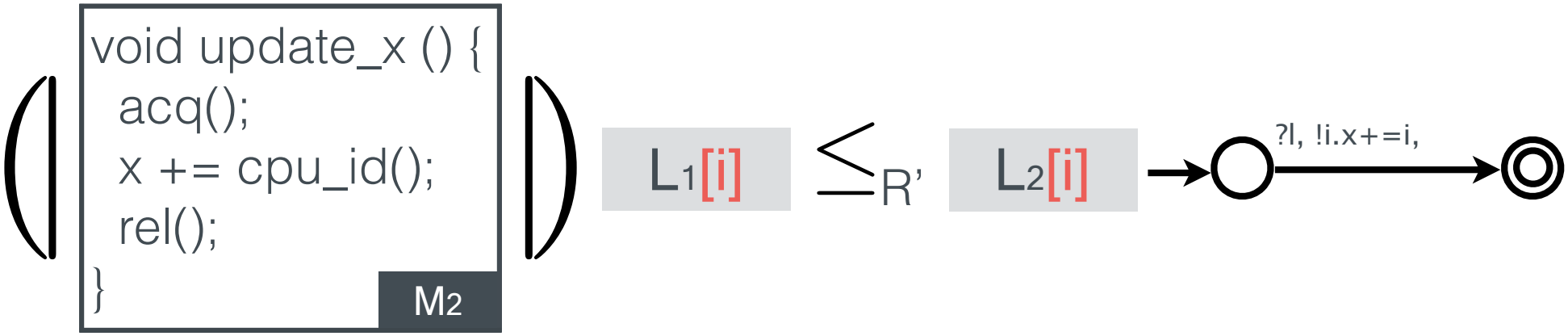


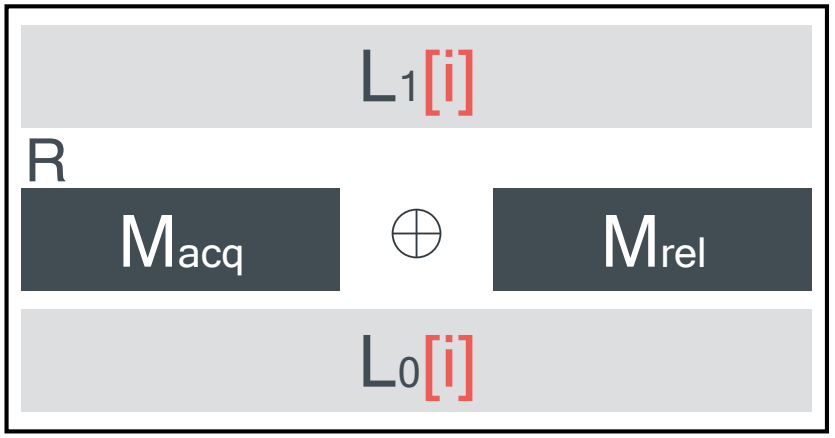
# Certified Concurrent Abstraction Layer



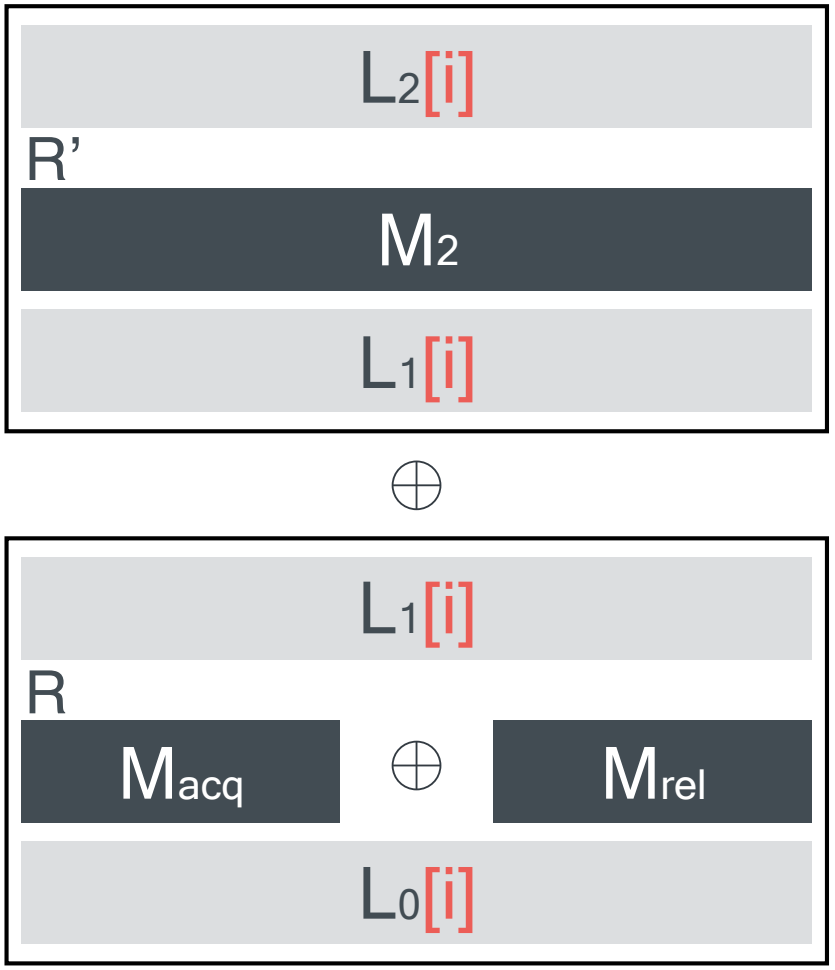



# Certified Concurrent Abstraction Layer

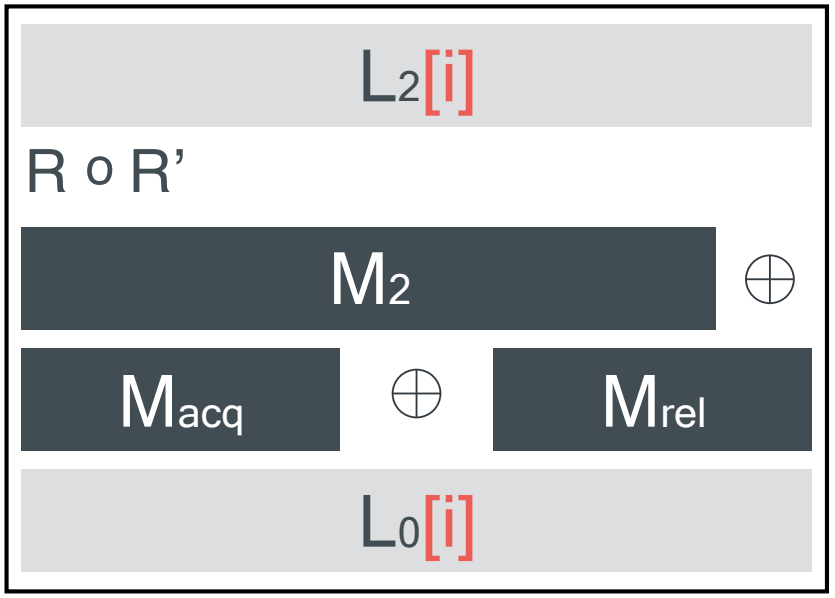


# Vertical Composition



# Vertical Composition

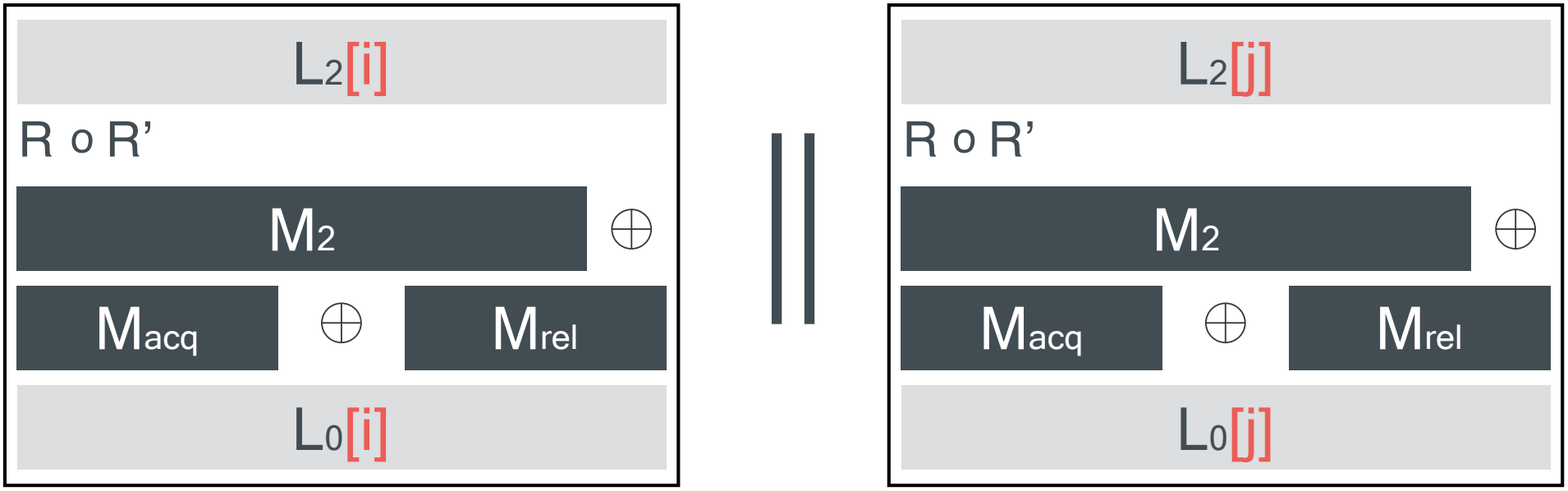


# Parallel Composition

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

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

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

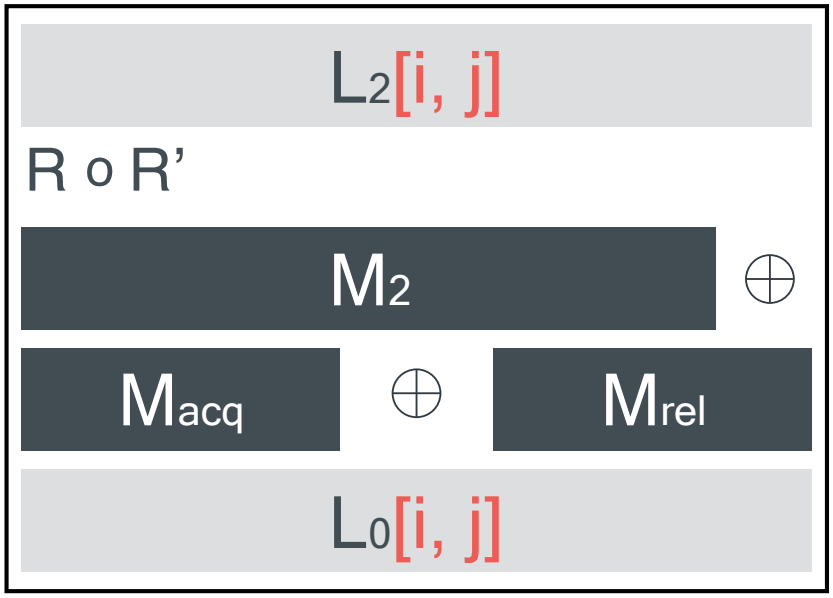


# Parallel Composition

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

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

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

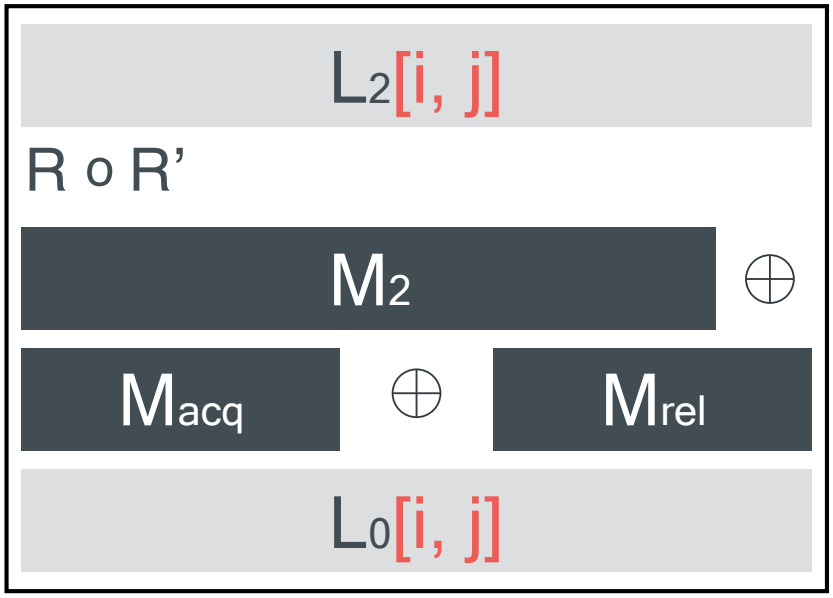


# Parallel Composition

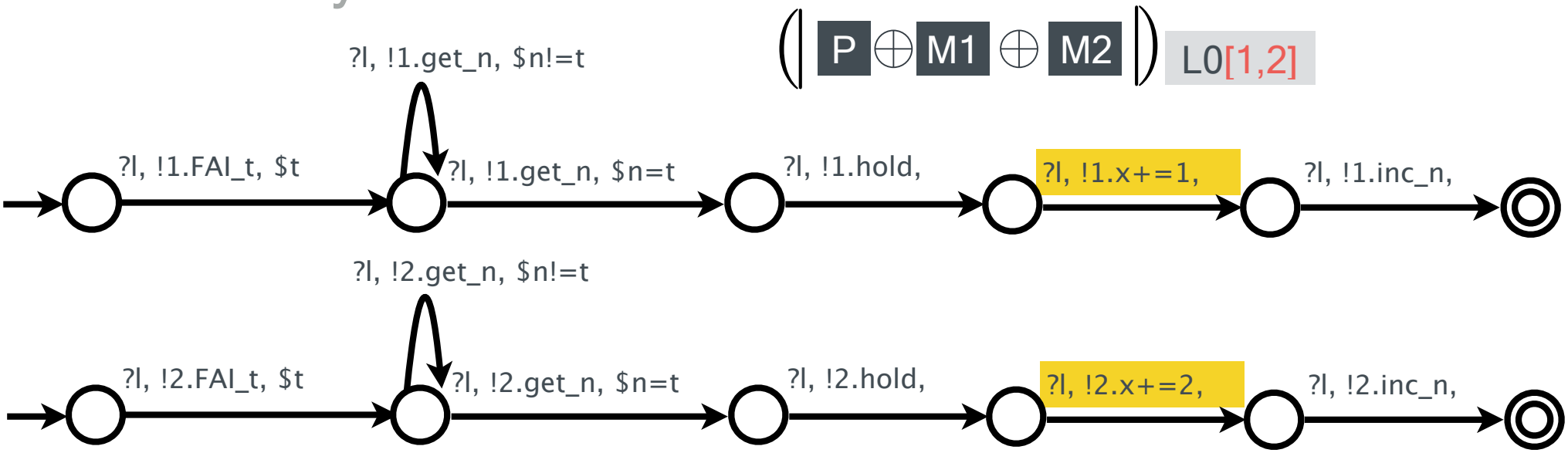
$\mathcal{R}_{hs}$ : (fairness) each CPU will be rescheduled within m steps

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

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



# Case Study




$(\ P \oplus M1 \oplus M2\ )$ $L0[1,2]$

$L_2[1, 2]$

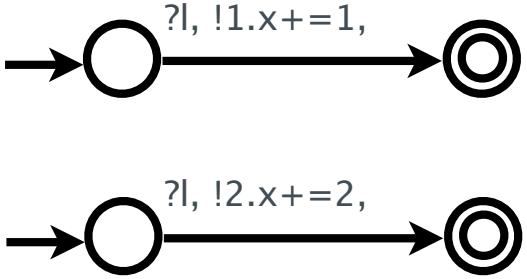
$R \circ R'$

$M_2 \oplus$

$M_{acq} \oplus M_{rel}$

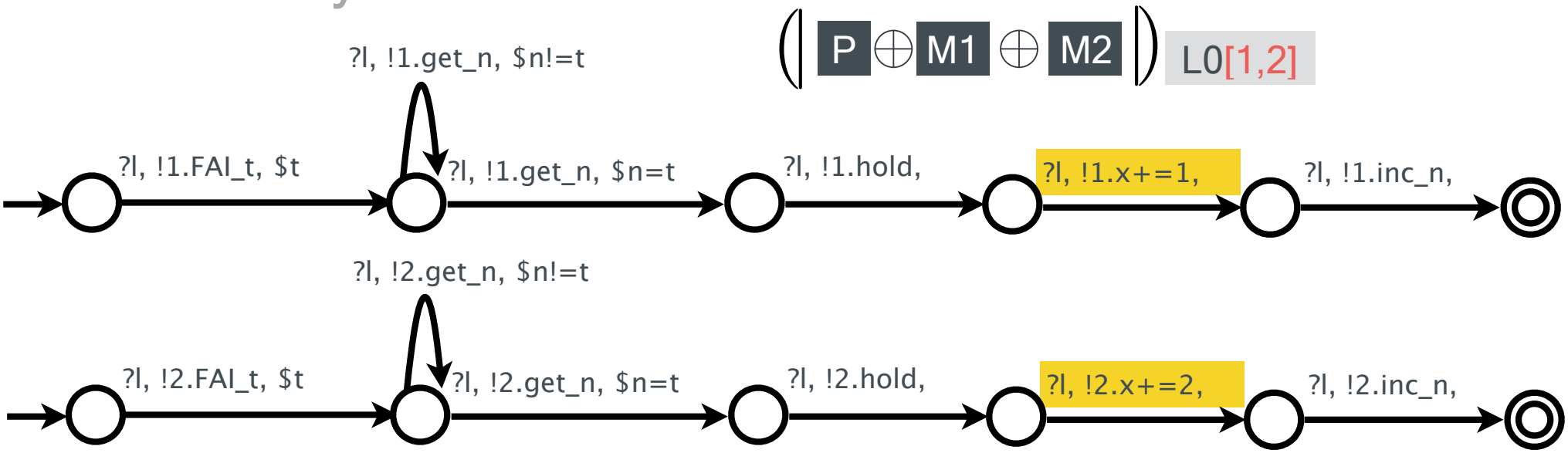
$L_0[1, 2]$

$\leq R \circ R'$

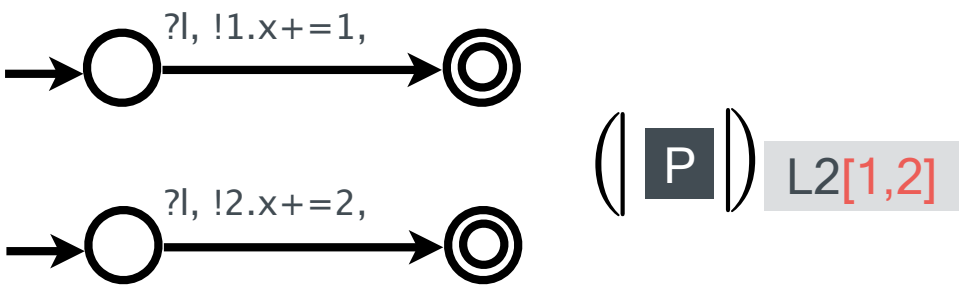



$(\ P\ )$ $L2[1,2]$

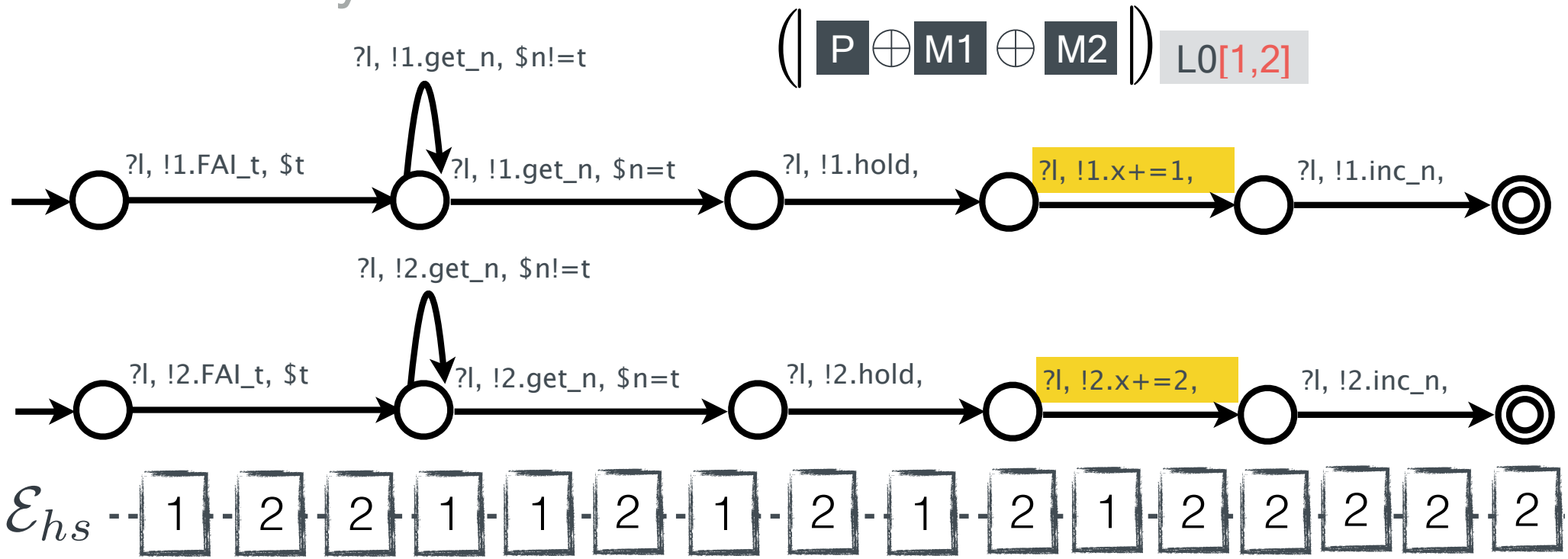
# Case Study

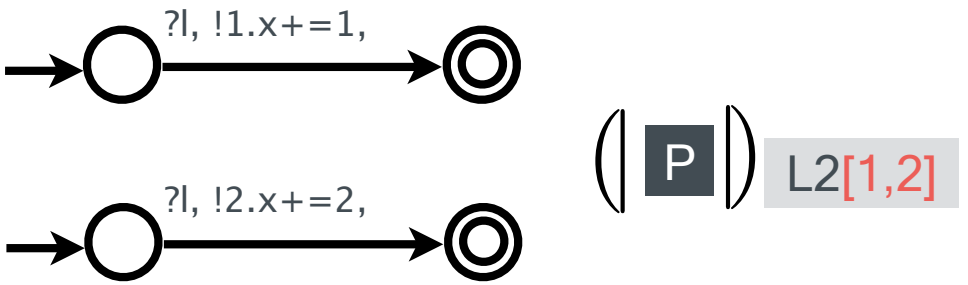



$\leq$ R o R'

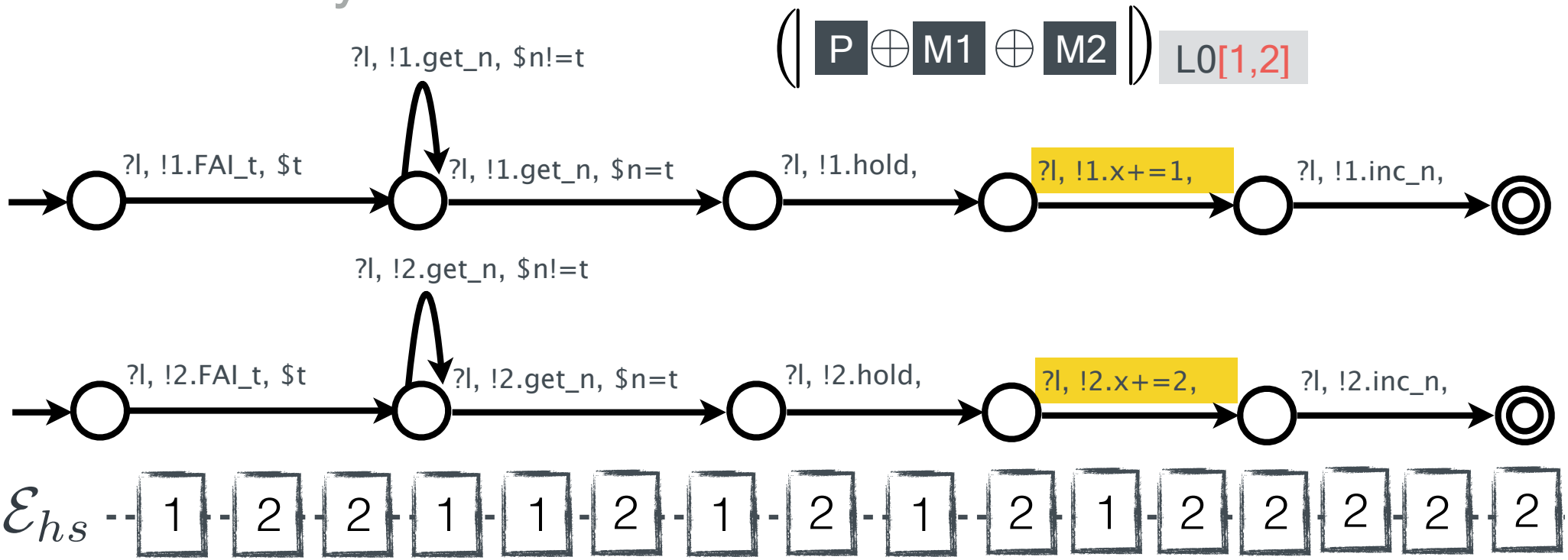


# Case Study

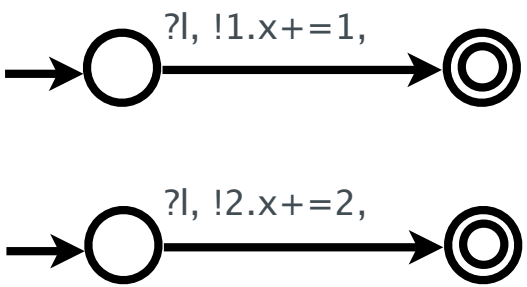

P ⊕ M1 ⊕ M2
L0[1,2]
?l, !1.get_n, $n!=t
?l, !1.FAI_t, $t
?l, !1.get_n, $n=t
?l, !1.hold,
?l, !1.x+=1,
?l, !1.inc_n,
?l, !2.get_n, $n!=t
?l, !2.FAI_t, $t
?l, !2.get_n, $n=t
?l, !2.hold,
?l, !2.x+=2,
?l, !2.inc_n,
$\mathcal{E}_{hs}$
1 2 2 1 1 2 1 2 1 2 1 2 2 2 2 2


?l, !1.x+=1,
?l, !2.x+=2,
P
L2[1,2]

# Case Study

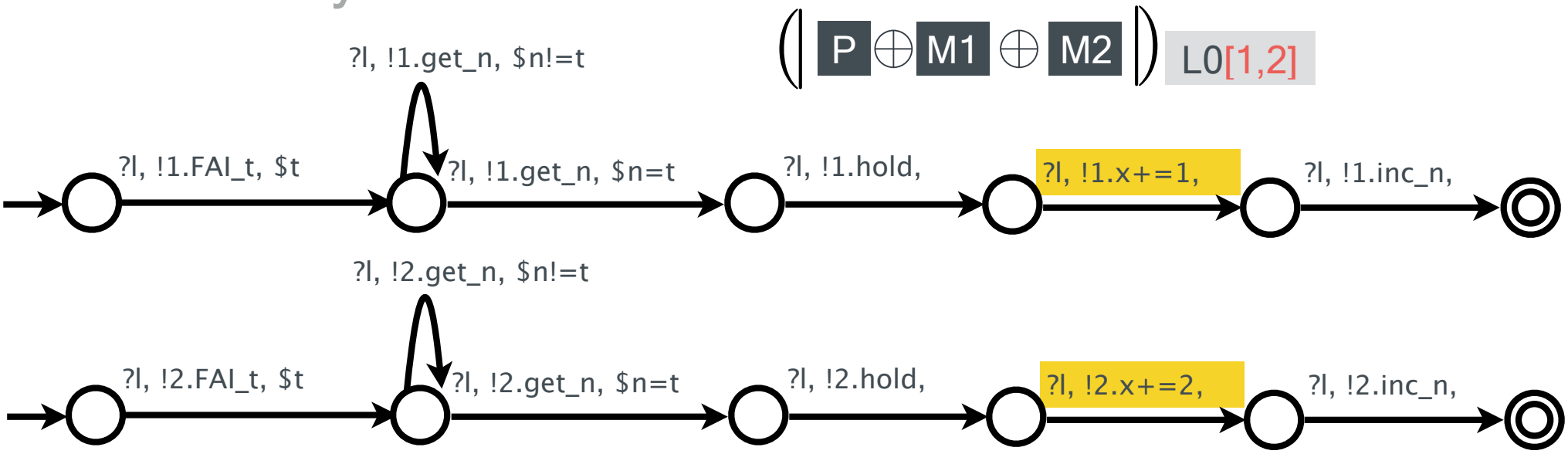

?l, !1.get_n, $n!=t
P ⊕ M1 ⊕ M2
L0[1,2]
?l, !1.FAI_t, $t
?l, !1.get_n, $n=t
?l, !1.hold,
?l, !1.x+=1,
?l, !1.inc_n,
?l, !2.get_n, $n!=t
?l, !2.FAI_t, $t
?l, !2.get_n, $n=t
?l, !2.hold,
?l, !2.x+=2,
?l, !2.inc_n,
$\mathcal{E}_{hs}$
1 2 2 1 1 2 1 2 1 2 1 2 2 2 2 2


$\mathcal{E}'_{hs}$
1 2


?l, !1.x+=1,
?l, !2.x+=2,


P
L2[1,2]

# Case Study



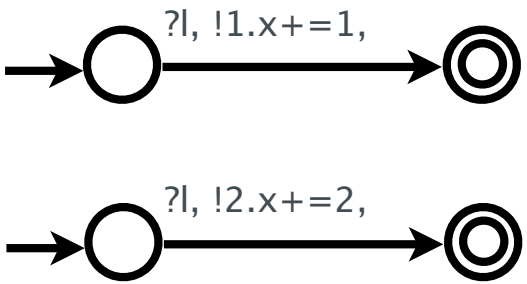




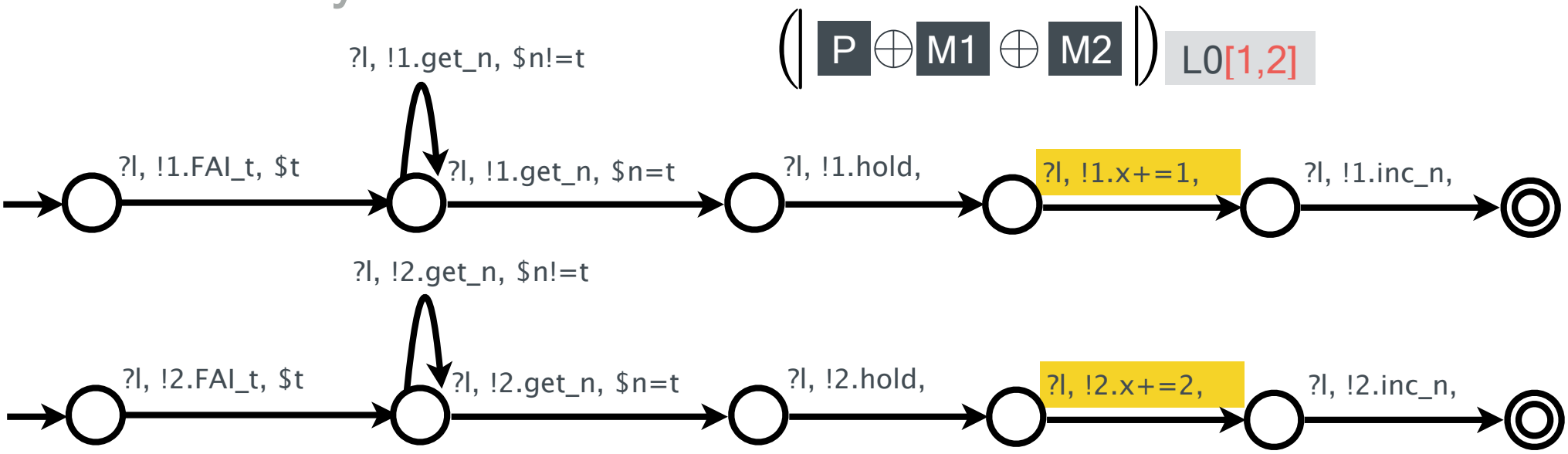
$\mathcal{E}_{hs}$ -- 1 - 2 - 2 - 1 - 1 - 2 - 1 - 2 - 1 - 2 - 1 - 2 - 2 - 2 - 2 - 2

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 | | | | |

$\mathcal{E}'_{hs}$ -- 1 - 2 ---




# Case Study







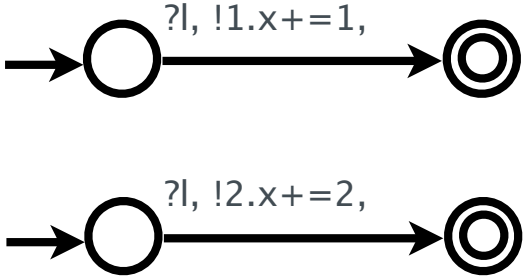
$\mathcal{E}_{hs}$ 1 2 2 1 1 2 1 2 1 2 1 2 2 2 2 2

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

$\mathcal{E}'_{hs}$ 1 2

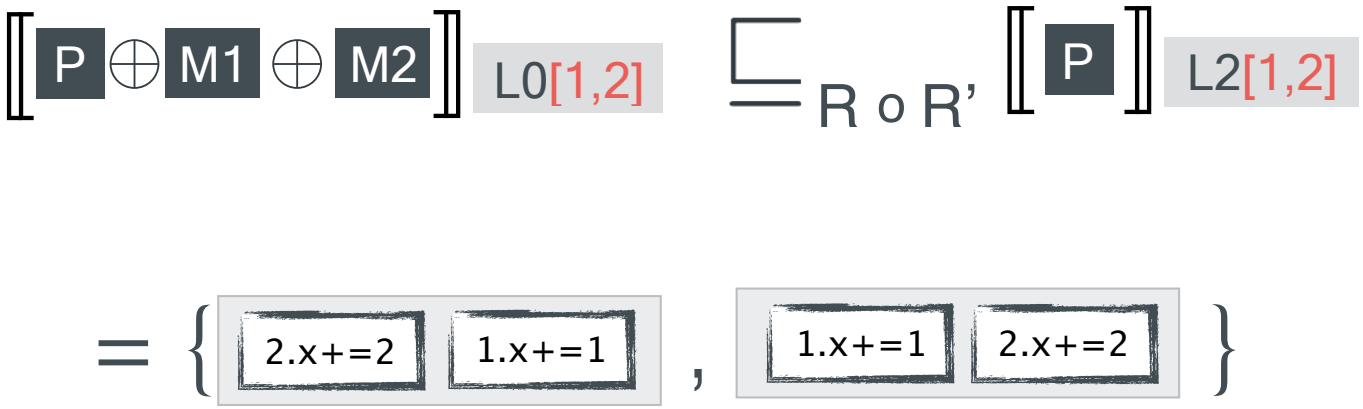



# Soundness

# Soundness

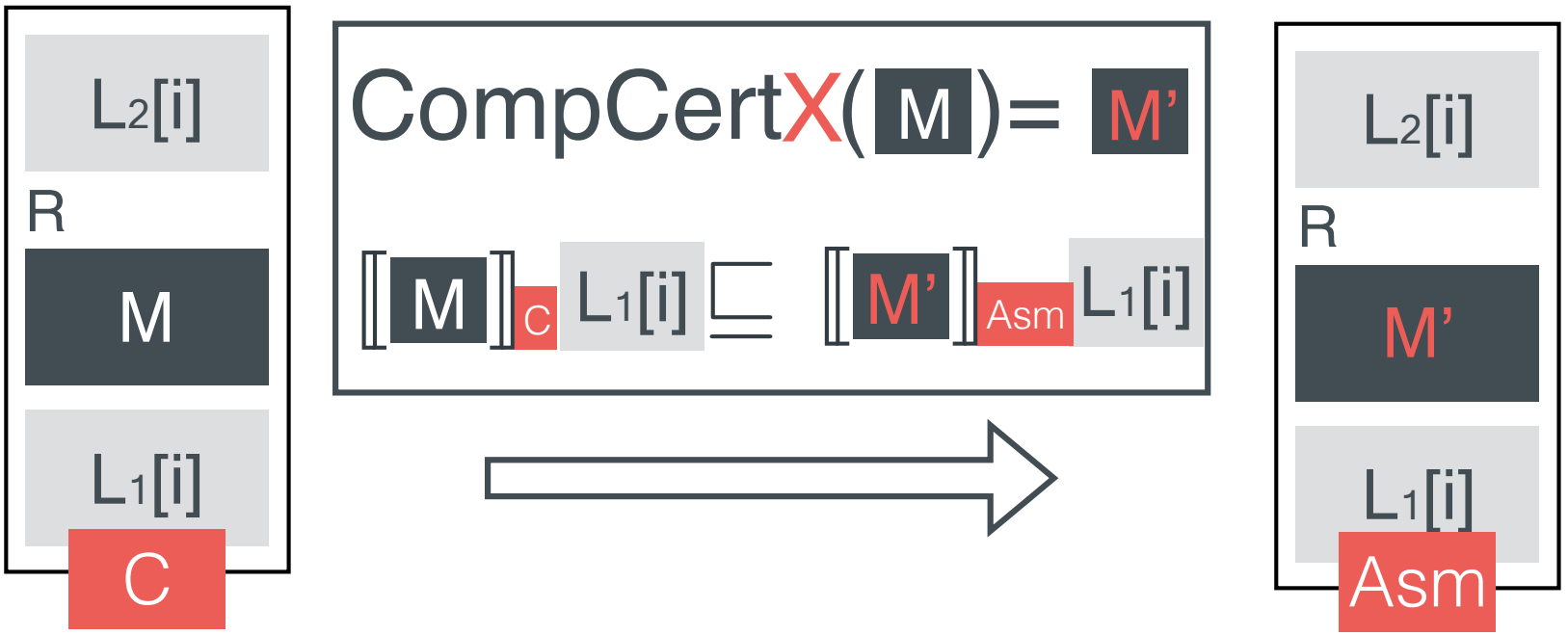
$\mathcal{R}_{hs}$ : (fairness) each CPU will be rescheduled within m steps

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




QED

# CompCertX



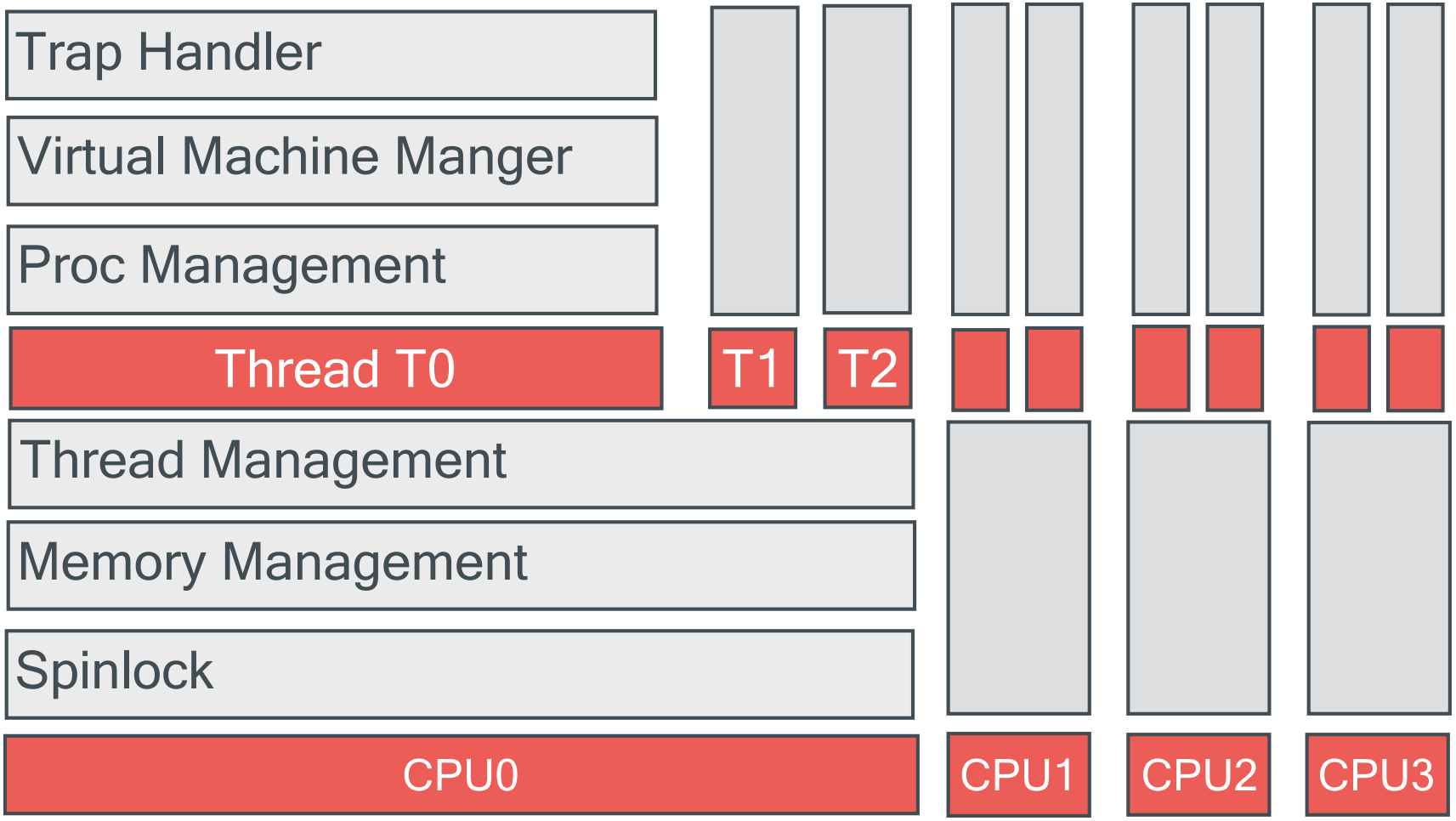
# Assembly Layers


Horizontal Composition

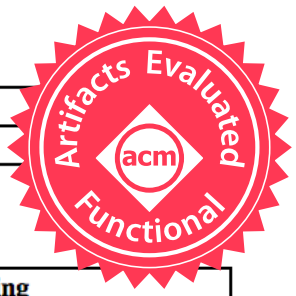
Vertical Composition

Parellel Composition

# 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**

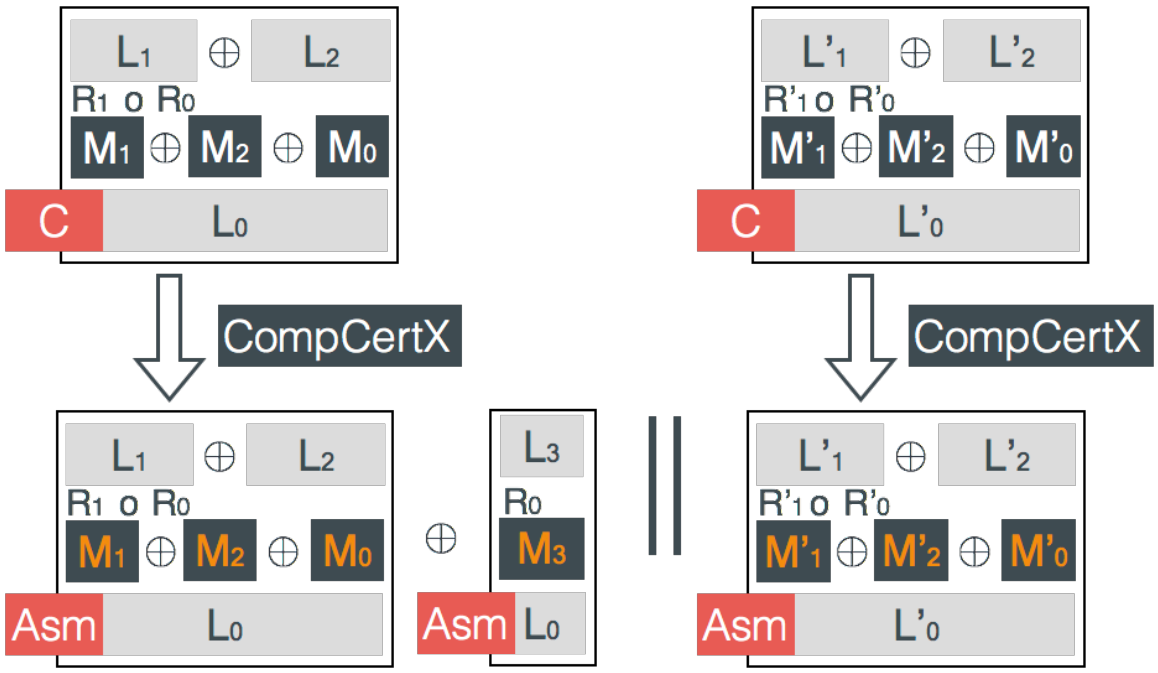
| Layer | Refinement proof | Code verification | Source code | Proof linking |
|---|---|---|---|---|
| PHBThread | HBThreadGen | -- | -- | HBThreadGenLink |

**Layers for per-CPU**

| Layer | Refinement proof | Code verification | Source code | Proof linking |
|---|---|---|---|---|
| **Thread linking interface** | | | | |
| PBThread | BThreadGen | -- | -- | BThreadGenLink |
| **Virtual machine** | | | | |

# Contribution Summary

## Certified Concurrent Abstraction Layers




## Strategy Refinement

$$\psi \leq_R \psi'$$

## Thread-safe CompCert

## CertiKOS