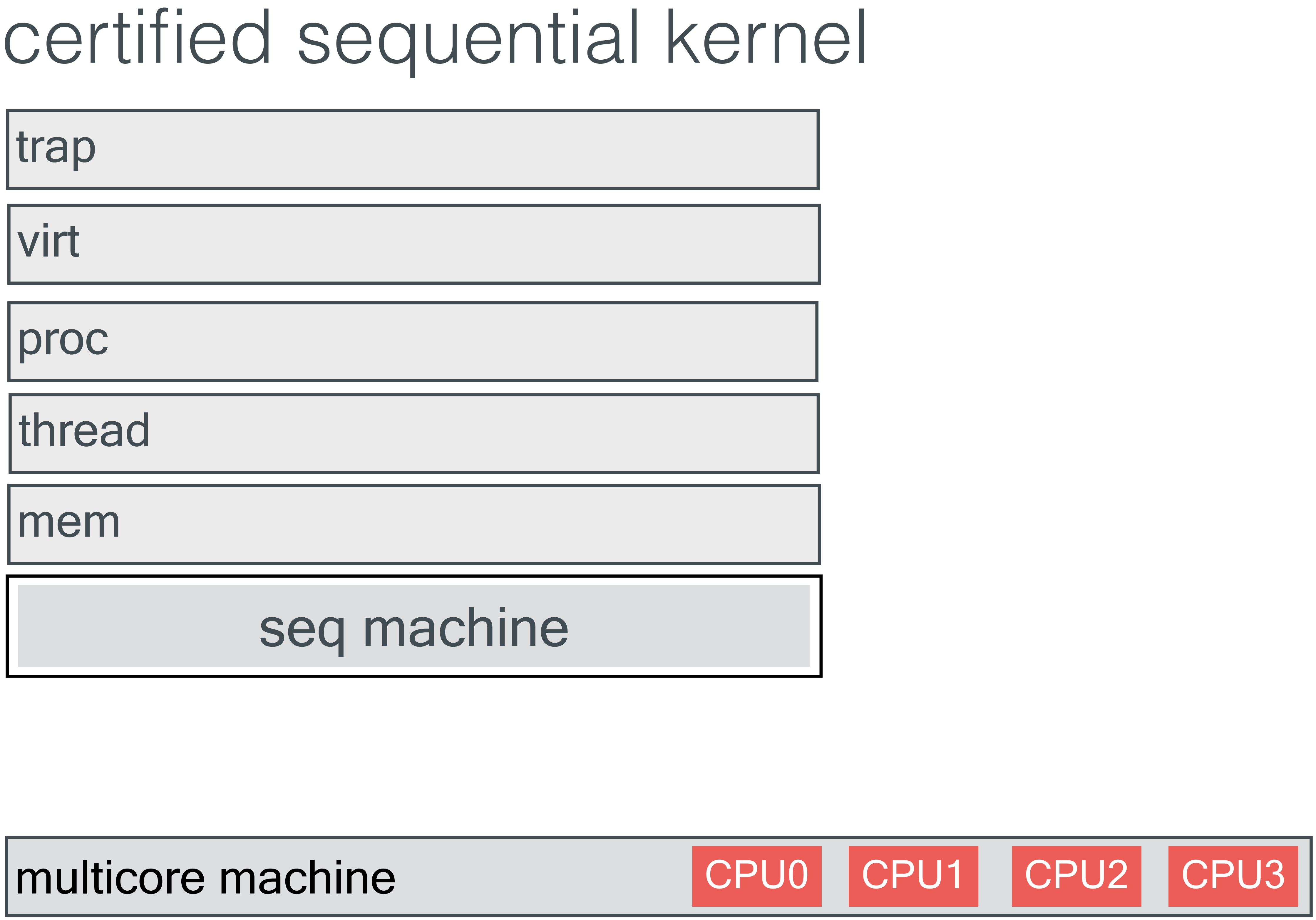
CS 428/528 Lecture 11: CCAL and mC2

Zhong Shao

Yale University

February 20, 2024

Concurrent Framework [OSDI’16, PLDI’18]



certified sequential kernel

trap

virt

proc

thread

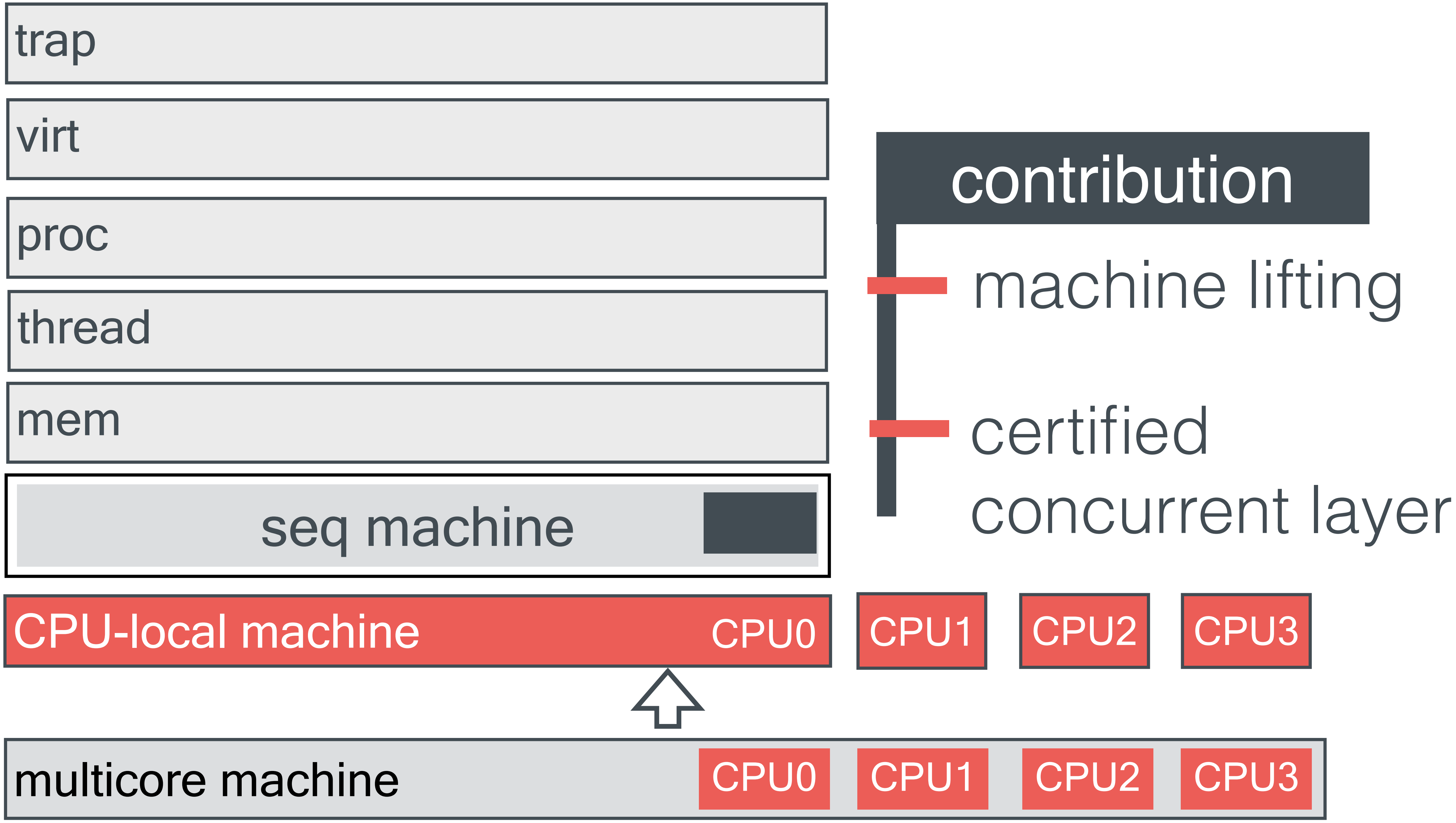
mem

seq machine

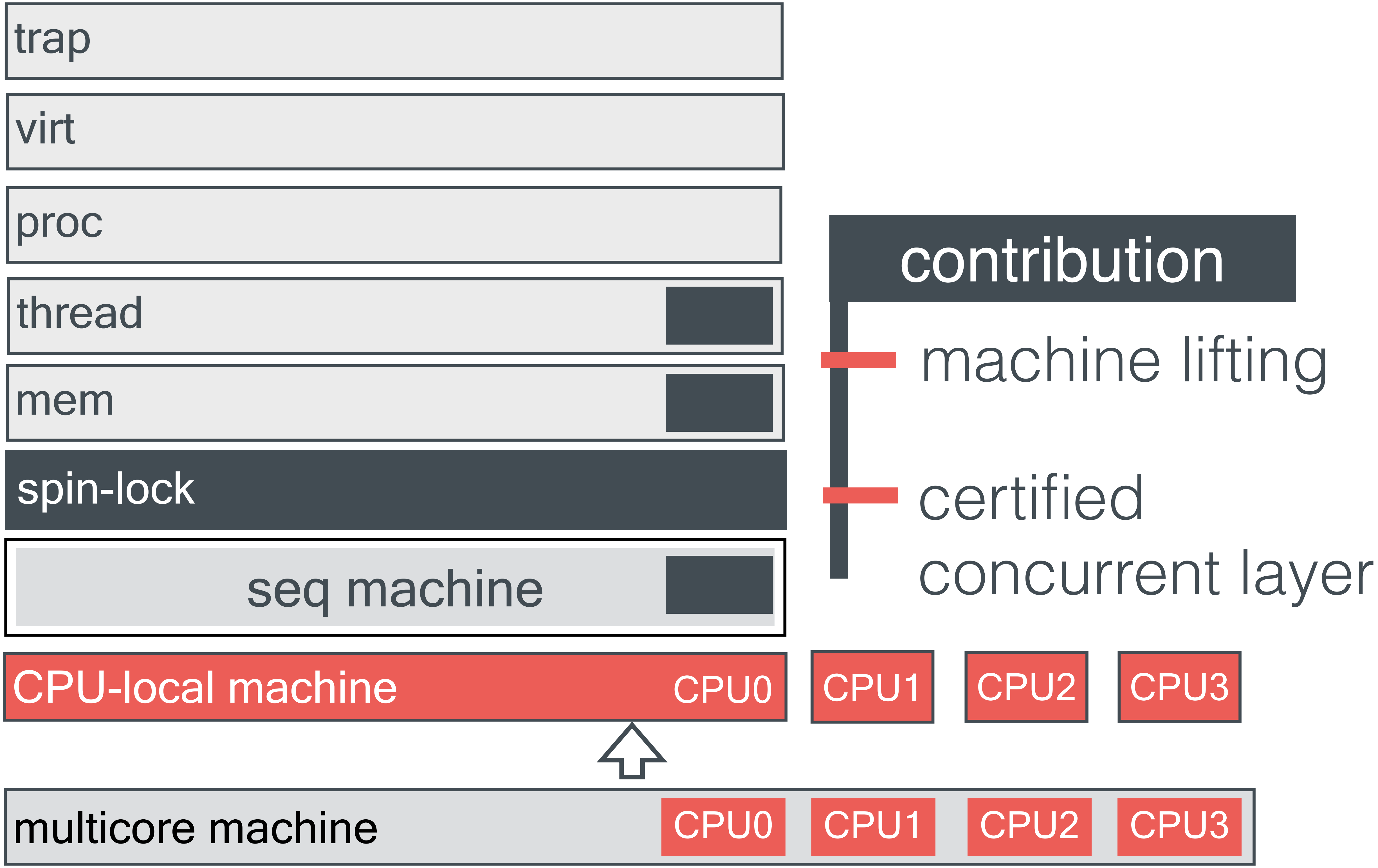
multicore machine

CPU0 CPU1 CPU2 CPU3

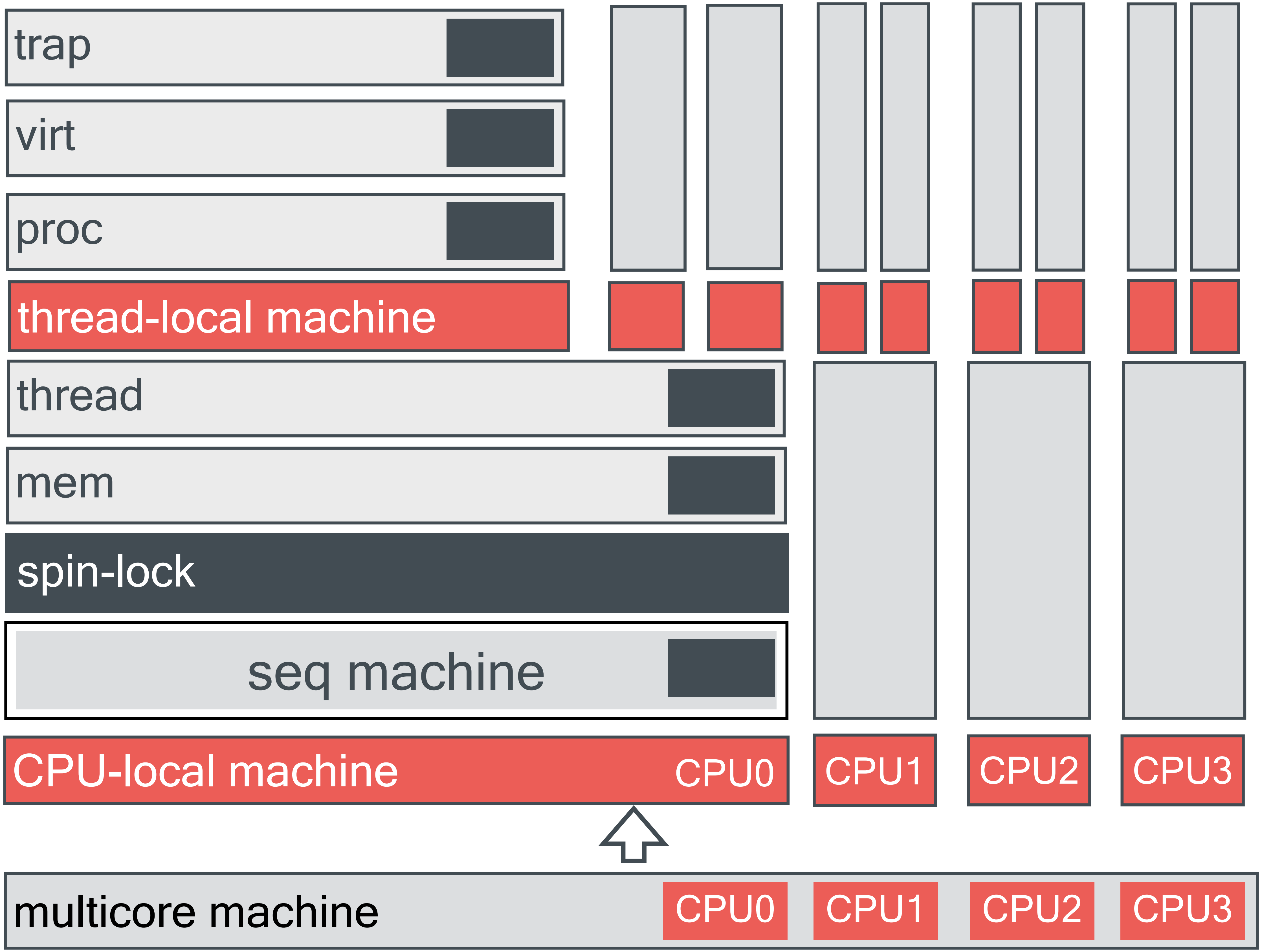
Concurrent Framework [OSDI'16, PLDI'18]



Concurrent Framework [OSDI’16, PLDI’18]

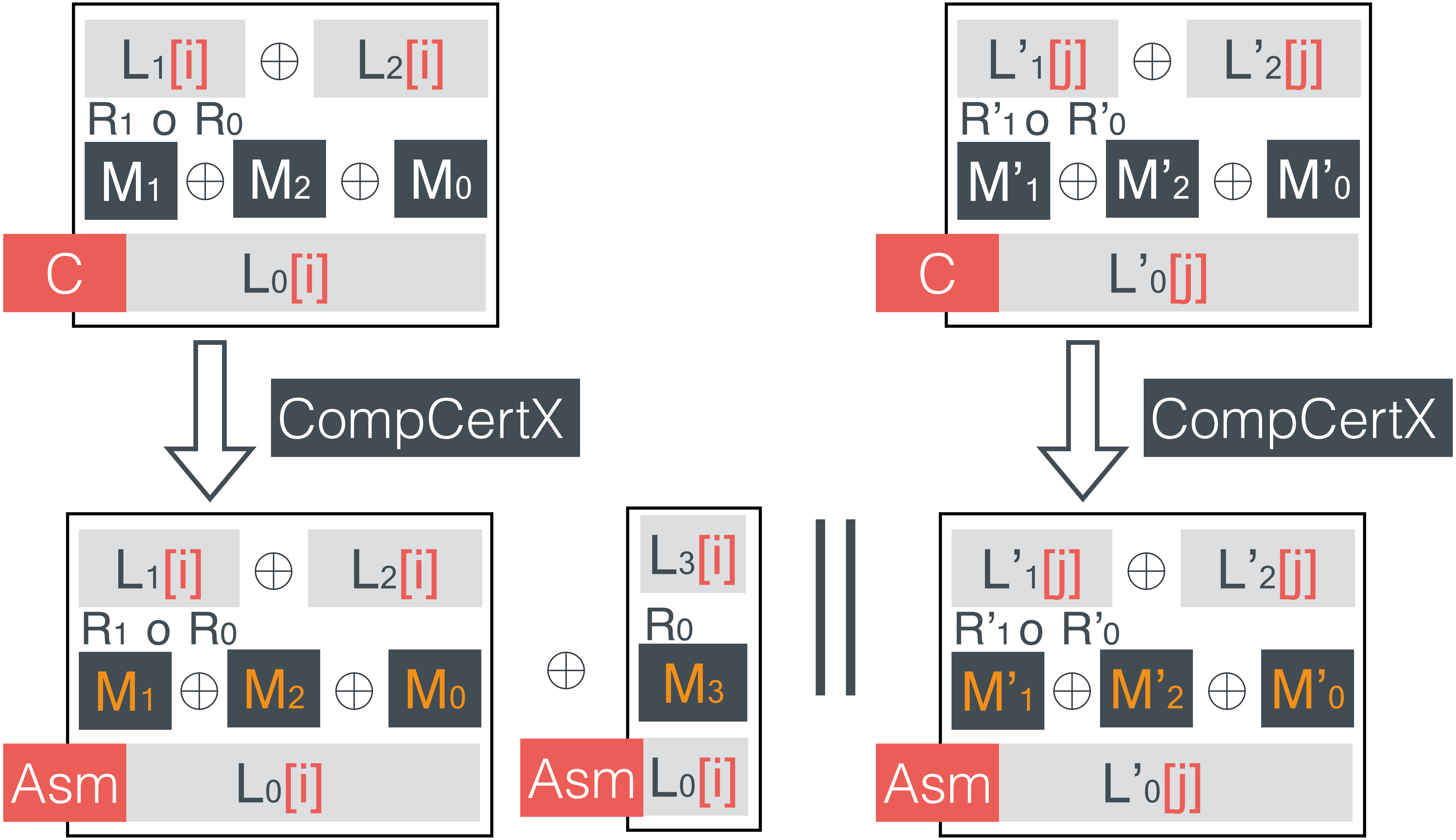


Concurrent Framework [OSDI’16, PLDI’18]



Contribution

Certified Concurrent Abstraction Layers



$L_1[i] \oplus L_2[i]$

$R_1 \circ R_0$

$M_1 \oplus M_2 \oplus M_0$

C $L_0[i]$

$L'_1[j] \oplus L'_2[j]$

$R'_1 \circ R'_0$

$M'_1 \oplus M'_2 \oplus M'_0$

C $L'_0[j]$

CompCertX

CompCertX

$L_1[i] \oplus L_2[i]$

$R_1 \circ R_0$

$M_1 \oplus M_2 \oplus M_0$

Asm $L_0[i]$

$\oplus$

$L_3[i]$

R_0

M_3

Asm $L_0[i]$

$||$

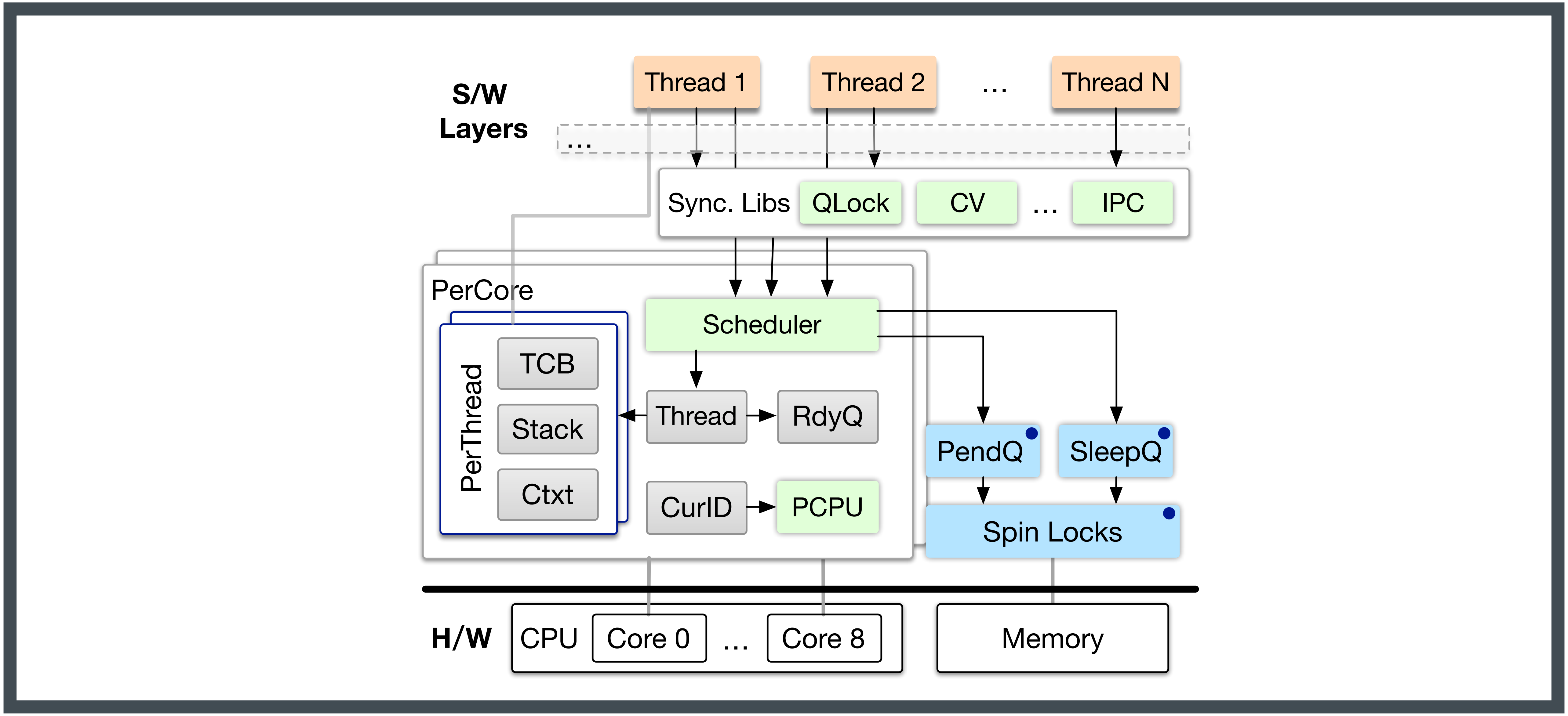
$L'_1[j] \oplus L'_2[j]$

$R'_1 \circ R'_0$

$M'_1 \oplus M'_2 \oplus M'_0$

Asm $L'_0[j]$

Contribution



Case Study

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

```
//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
//M2 module
int x = 0; //shared variable x
void update_x () {
  acq(); x += cpu_id();  rel();
}
//Methods provided by L2
extern void update_x();

//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }
```

C

Case Study

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

```
//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
//M2 module
int x = 0; //shared variable x
void update_x () {
  acq(); x += cpu_id();  rel();
}
//Methods provided by L2
extern void update_x();

//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }
```

C

Case Study

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

```
//Methods provided by L1
extern void acq();
extern void rel();
extern cpu_id();
//M2 module
int x = 0; //shared variable x
void update_x () {
  acq(); x += cpu_id();  rel();
}
//Methods provided by L2
extern void update_x();

//Client program P
//Thread running on CPU 1
void T1 () { update_x(); }
//Thread running on CPU 2
void T2 () { update_x(); }
```

C

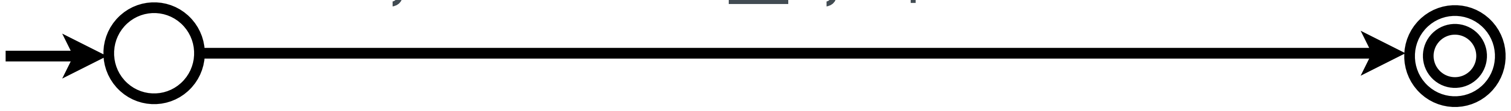
Strategies and Game Semantics

strategy $\psi_p[i]$

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

Strategies and Game Semantics

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

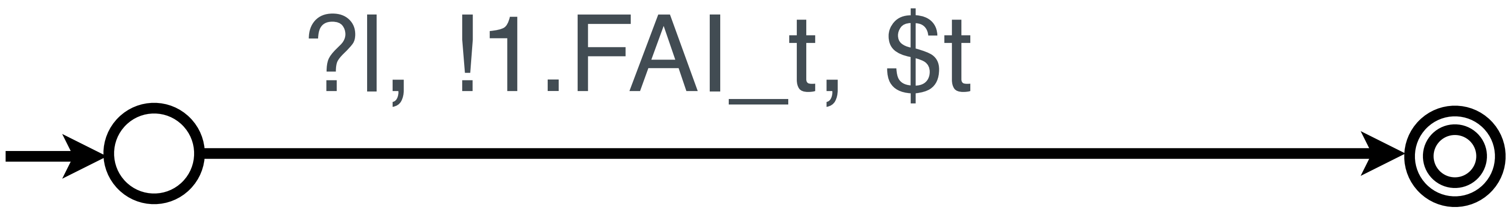


logical
log l

2.FAI_t	2.FAI_t

Strategies and Game Semantics

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

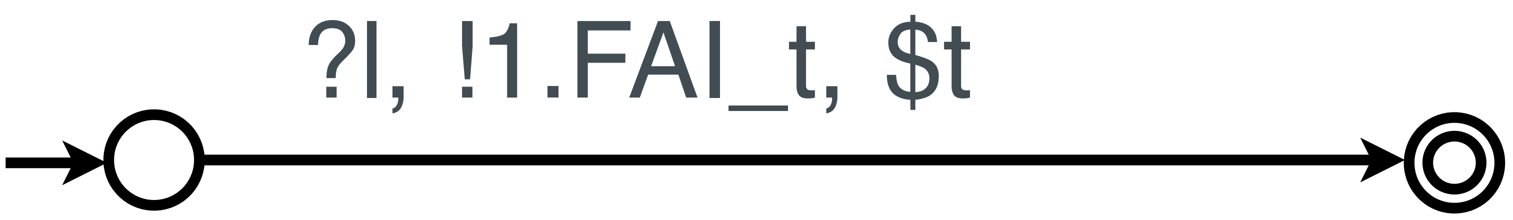


logical
log l

2.FAI_t	2.FAI_t	1.FAI_t
		$2

Strategies and Game Semantics

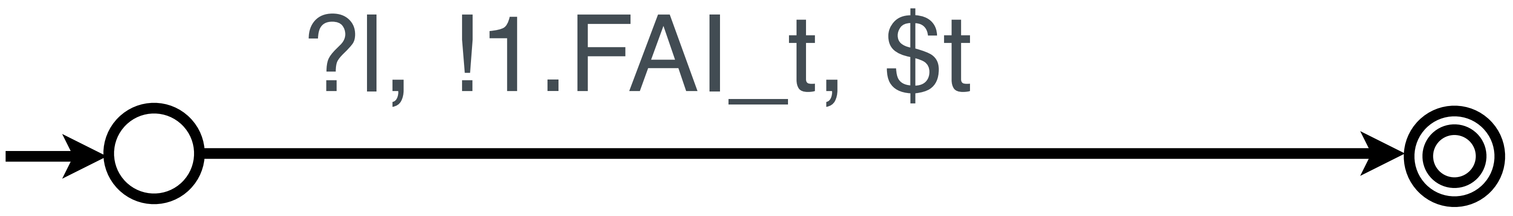
$$\psi_{\texttt{FAI\_t}}[1]$$



logical log l		
2.FAI_t	1.FAI_t	2.FAI_t

Strategies and Game Semantics

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



logical
log l

2.FAI_t	1.FAI_t	2.FAI_t	1.FAI_t

$3

Strategies and Game Semantics

$L_0[i]$

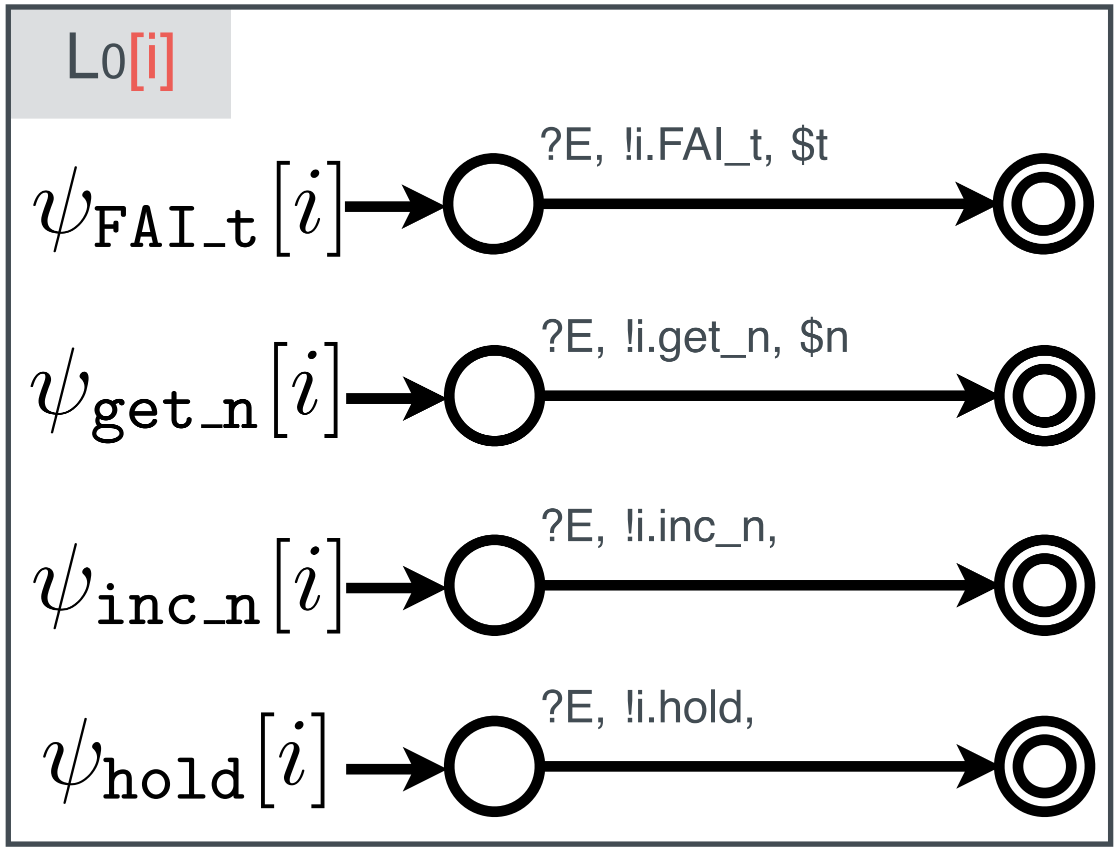
```
extern uint FAI_t();

extern uint get_n();

extern void inc_n();

extern void hold();
```

Strategies and Game Semantics


L0[i]
$\psi_{\texttt{FAI\_t}}[i]$
?E, !i.FAI_t, $t
$\psi_{\texttt{get\_n}}[i]$
?E, !i.get_n, $n
$\psi_{\texttt{inc\_n}}[i]$
?E, !i.inc_n,
$\psi_{\texttt{hold}}[i]$
?E, !i.hold,

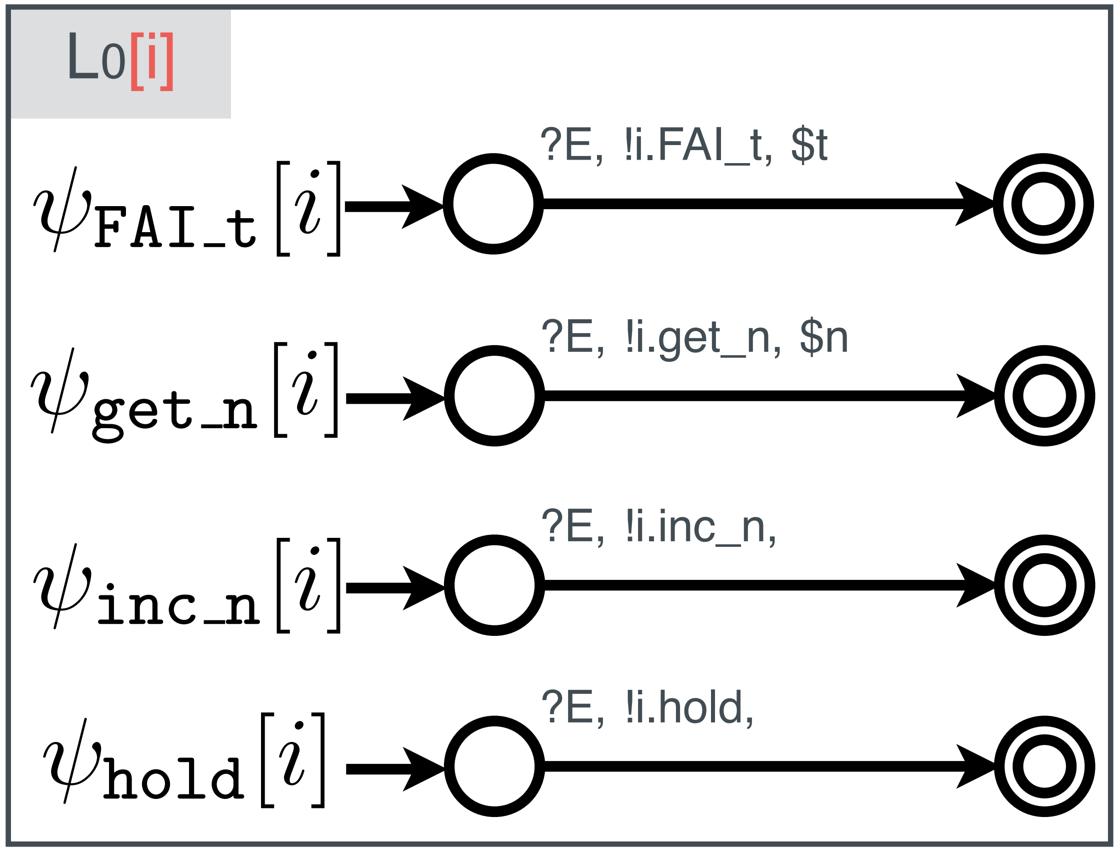
Strategies and Game Semantics

⦇

Macq

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

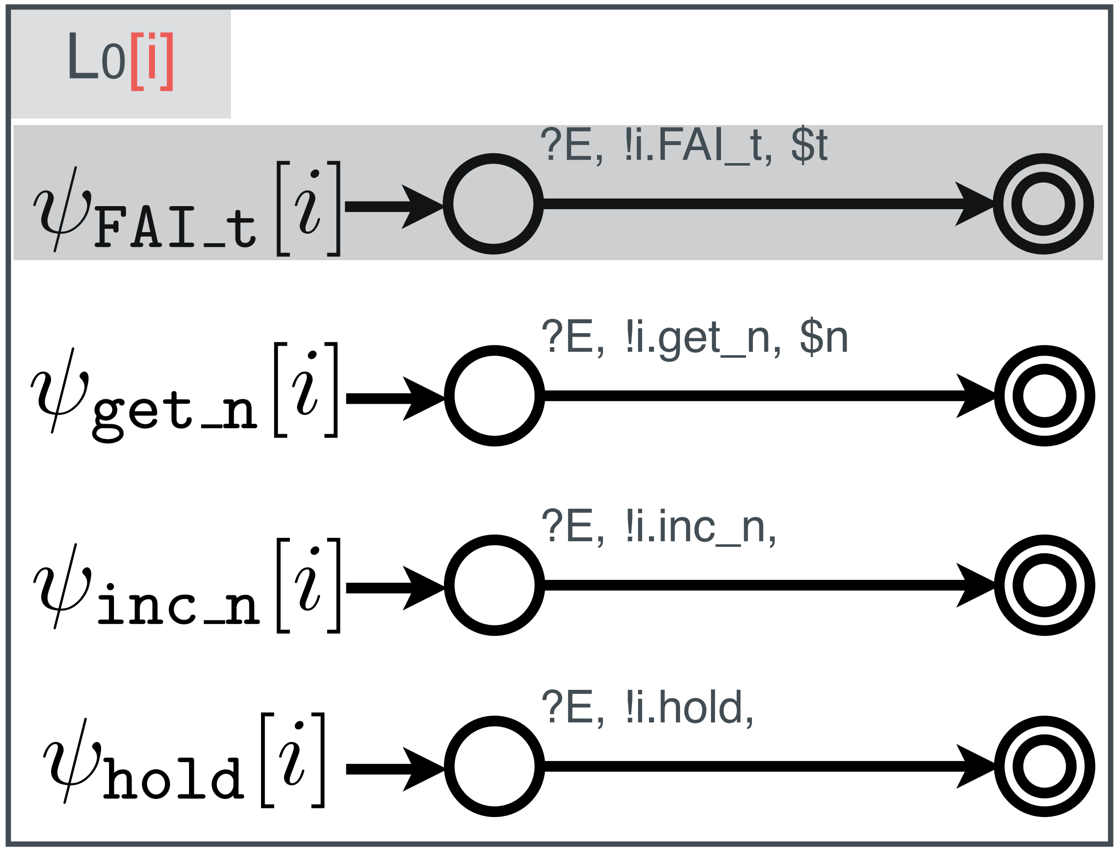
⦈


L0[i]
$\psi_{\texttt{FAI\_t}}[i]$
?E, !i.FAI_t, $t
$\psi_{\texttt{get\_n}}[i]$
?E, !i.get_n, $n
$\psi_{\texttt{inc\_n}}[i]$
?E, !i.inc_n,
$\psi_{\texttt{hold}}[i]$
?E, !i.hold,

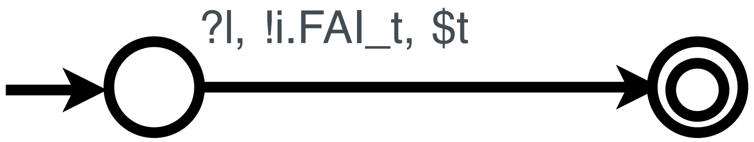
Strategies and Game Semantics

Macq

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



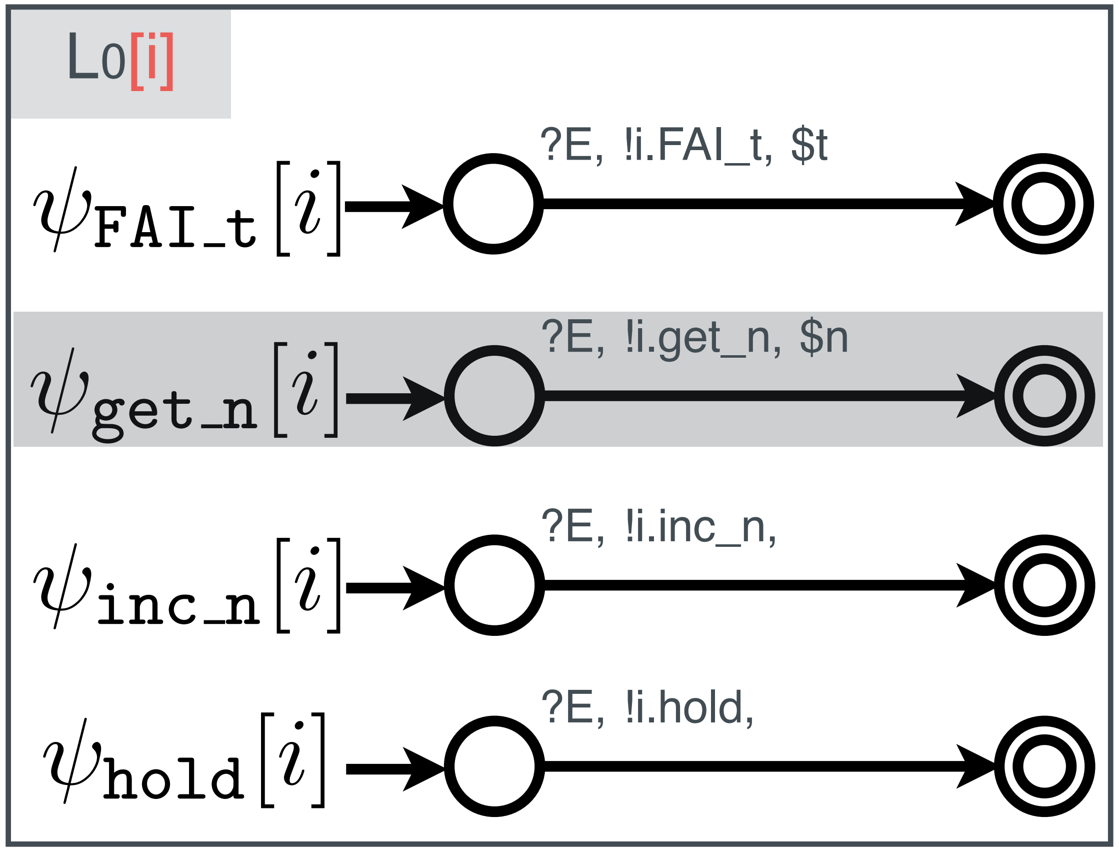
Lo[i]
$\psi_{\texttt{FAI\_t}}[i]$
?E, !i.FAI_t, $t
$\psi_{\texttt{get\_n}}[i]$
?E, !i.get_n, $n
$\psi_{\texttt{inc\_n}}[i]$
?E, !i.inc_n,
$\psi_{\texttt{hold}}[i]$
?E, !i.hold,

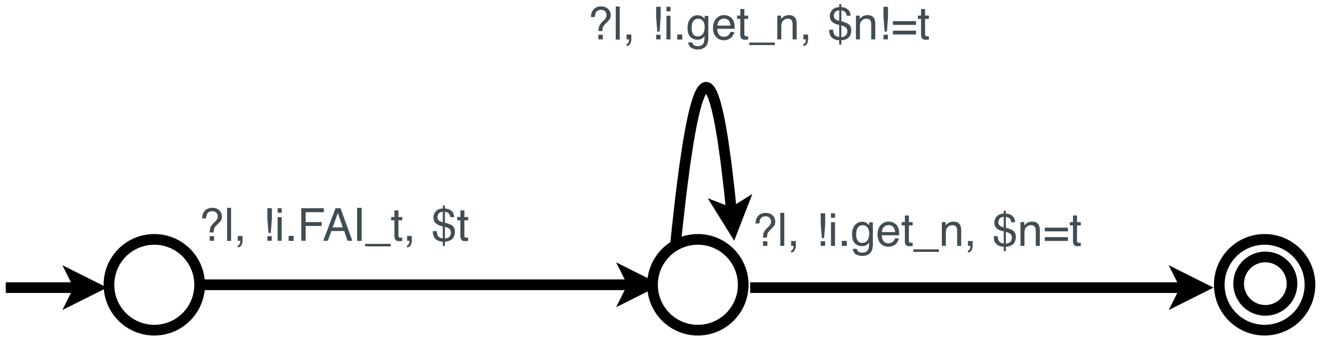


Strategies and Game Semantics

Macq

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

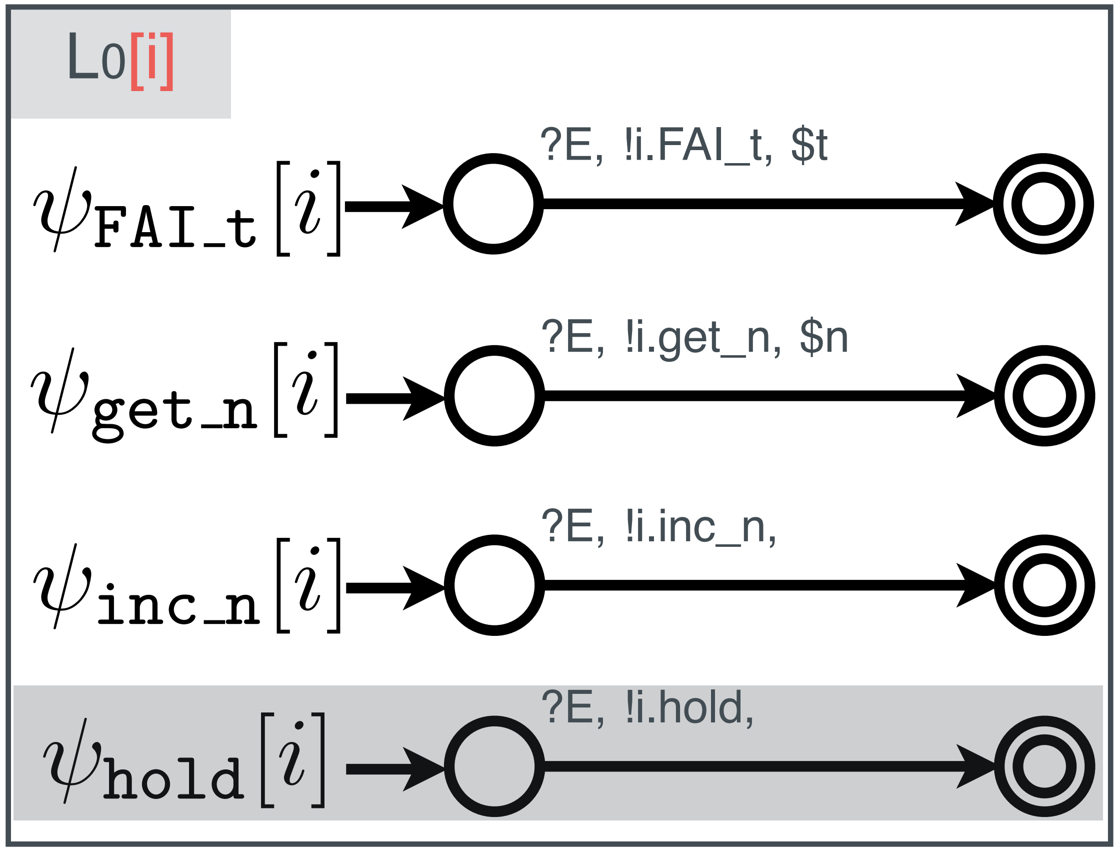

Lo[i]
$\psi_{\texttt{FAI\_t}}[i]$
?E, !i.FAI_t, $t
$\psi_{\texttt{get\_n}}[i]$
?E, !i.get_n, $n
$\psi_{\texttt{inc\_n}}[i]$
?E, !i.inc_n,
$\psi_{\texttt{hold}}[i]$
?E, !i.hold,

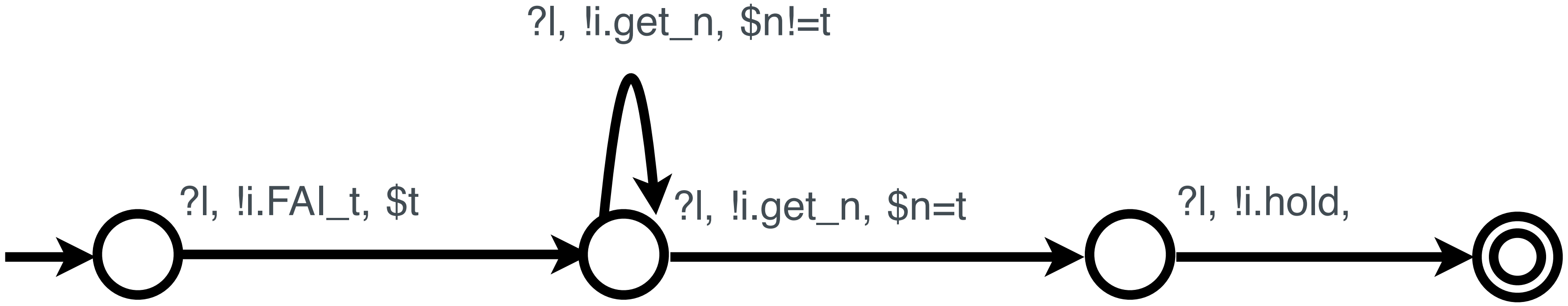


Strategies and Game Semantics

Macq

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


Lo[i]
$\psi_{\mathtt{FAI\_t}}[i]$
?E, !i.FAI_t, $t
$\psi_{\mathtt{get\_n}}[i]$
?E, !i.get_n, $n
$\psi_{\mathtt{inc\_n}}[i]$
?E, !i.inc_n,
$\psi_{\mathtt{hold}}[i]$
?E, !i.hold,



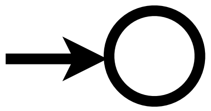
Strategies and Game Semantics

$$\text{strategy} \; \llparenthesis \, M_{acq} \, \rrparenthesis_{L_0[i]}$$

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

Strategies and Game Semantics

$$\left(\!\left| \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2} \right|\!\right)_{L0[1]}$$

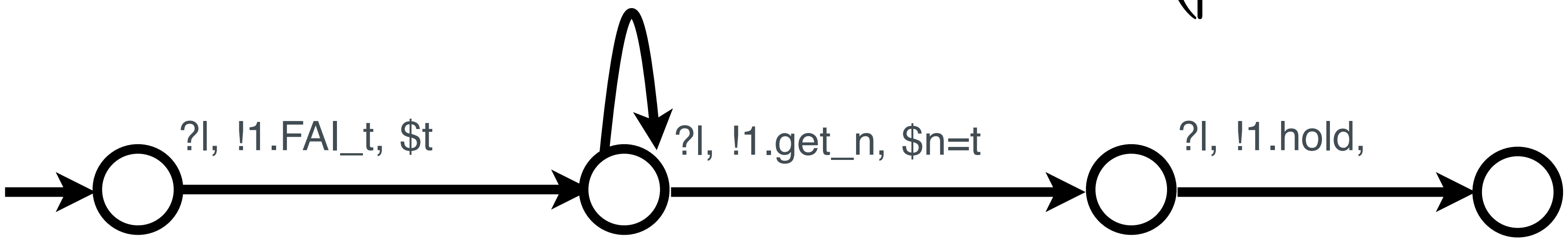


```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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



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

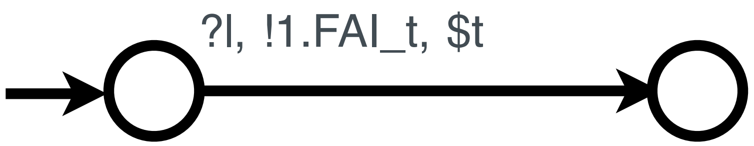
```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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

$$(\!| \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2} |\!)_{L0[1]}$$



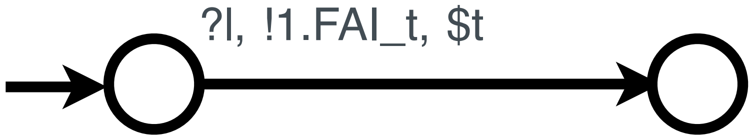
```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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

$$\llparenthesis \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2} \rrparenthesis_{L0[1]}$$



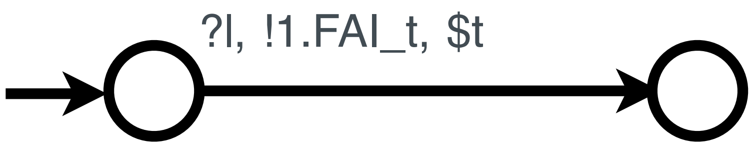
?I, !1.FAI_t, $t

logical
log l

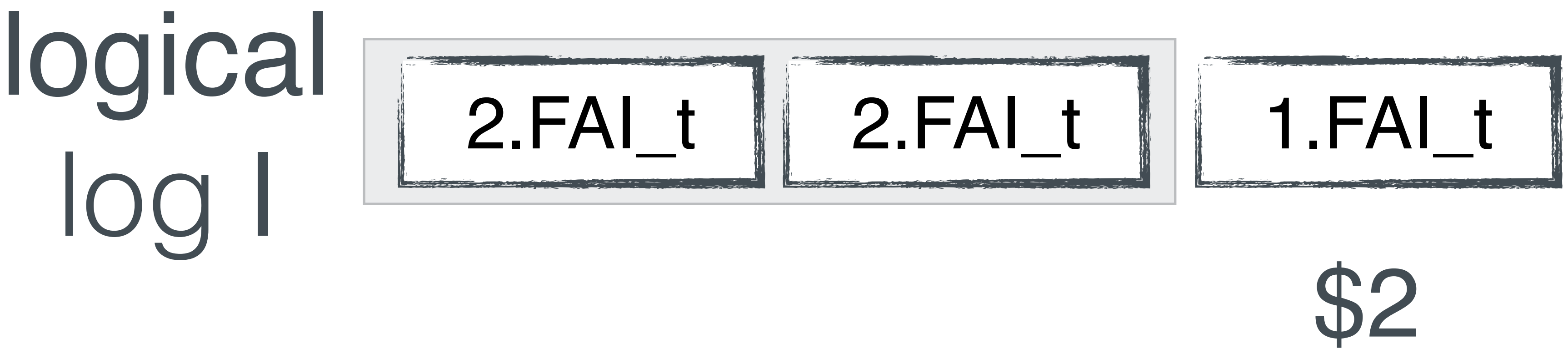
2.FAI_t	2.FAI_t

Strategies and Game Semantics

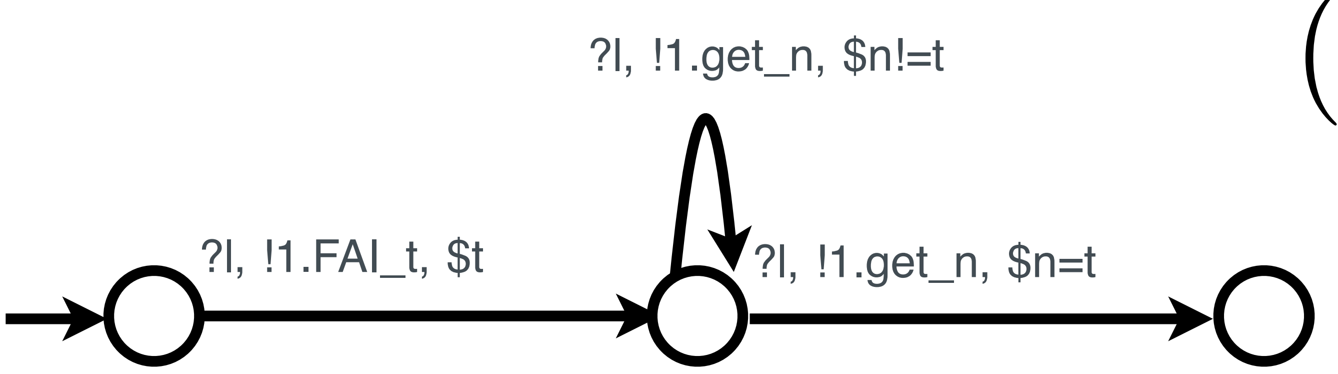
$$(\!| \, P \oplus M1 \oplus M2 \, |\!) \; L0[1]$$



?I, !1.FAI_t, $t



Strategies and Game Semantics



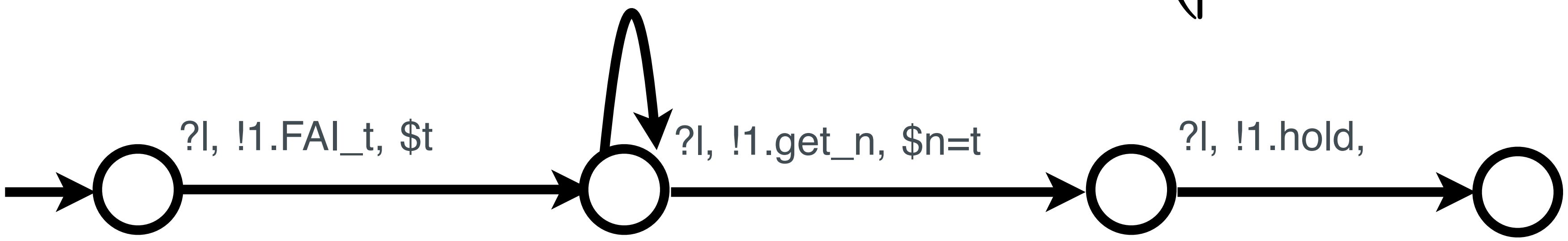
$(\!| P \oplus M1 \oplus M2 |\!)_{L0[1]}$

```
//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



$(\!|\ \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2}\ |\!)_{L0[1]}$

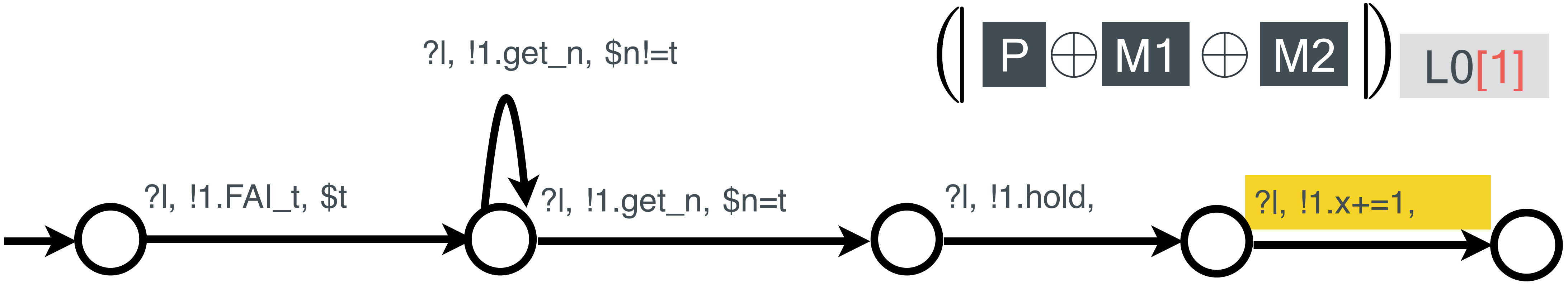
```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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

$(\!| P \oplus M1 \oplus M2 |\!)_{L0[1]}$

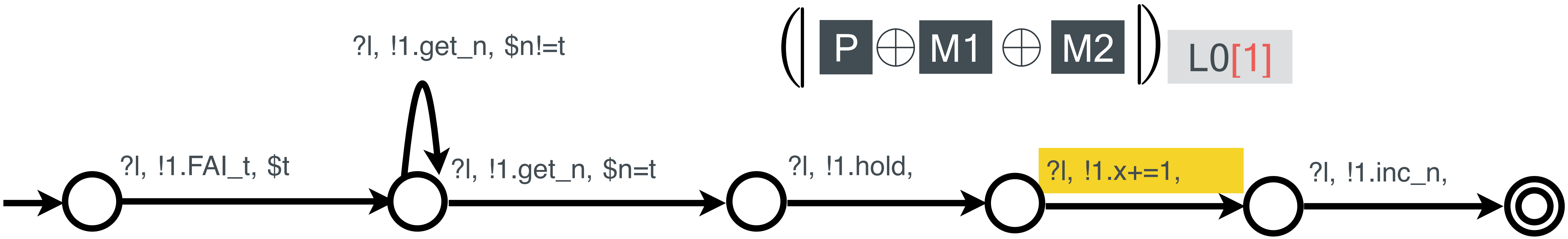


```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

```
//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



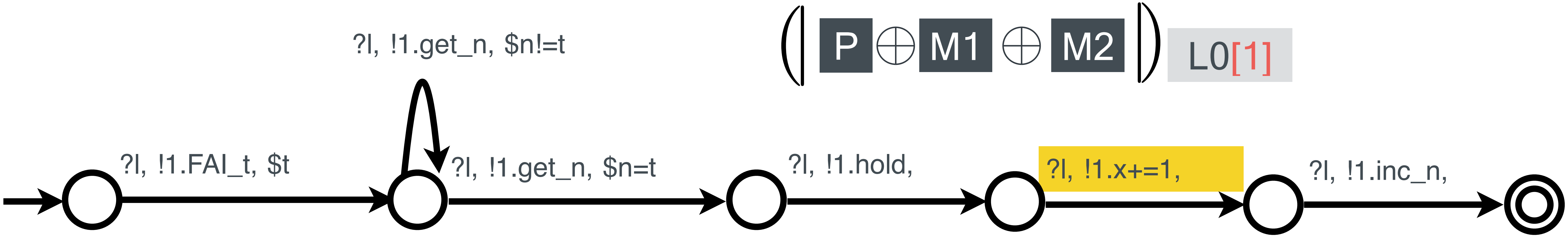
```
//Methods provided by L0
extern uint get_n();
extern void inc_n();
extern uint FAI_t();
extern void hold();
//M1 module
void acq () {
  uint my_t = FAI_t();
  while(get_n()!=my_t){};
  hold();
}
void rel () {   inc_n() ; }
```

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

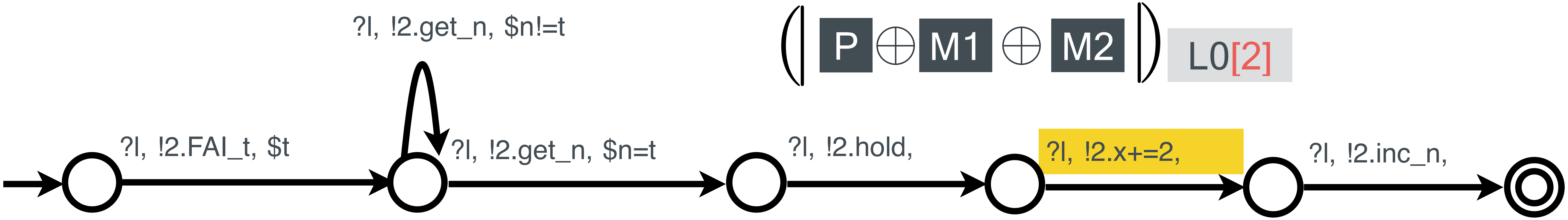
C

Strategies and Game Semantics

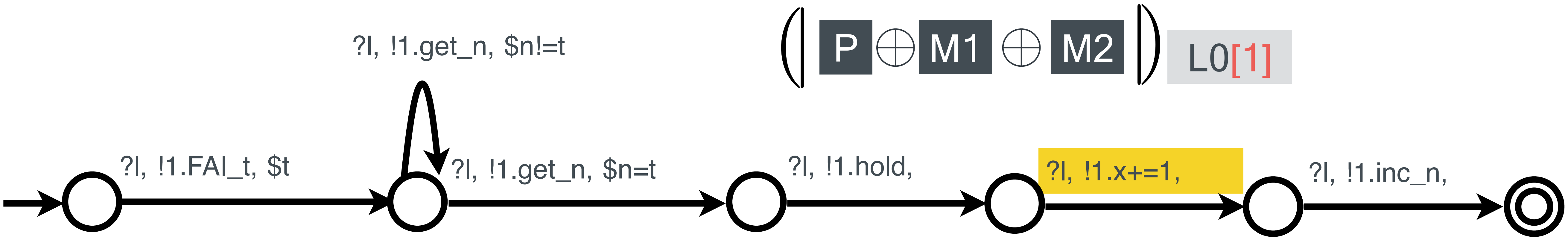
$(\!|\ \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2}\ |\!)$ L0[1]



$(\!|\ \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2}\ |\!)$ L0[2]

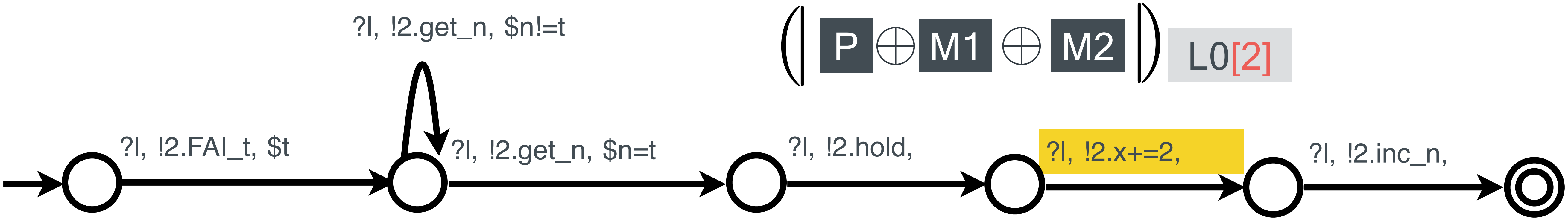


Strategies and Game Semantics



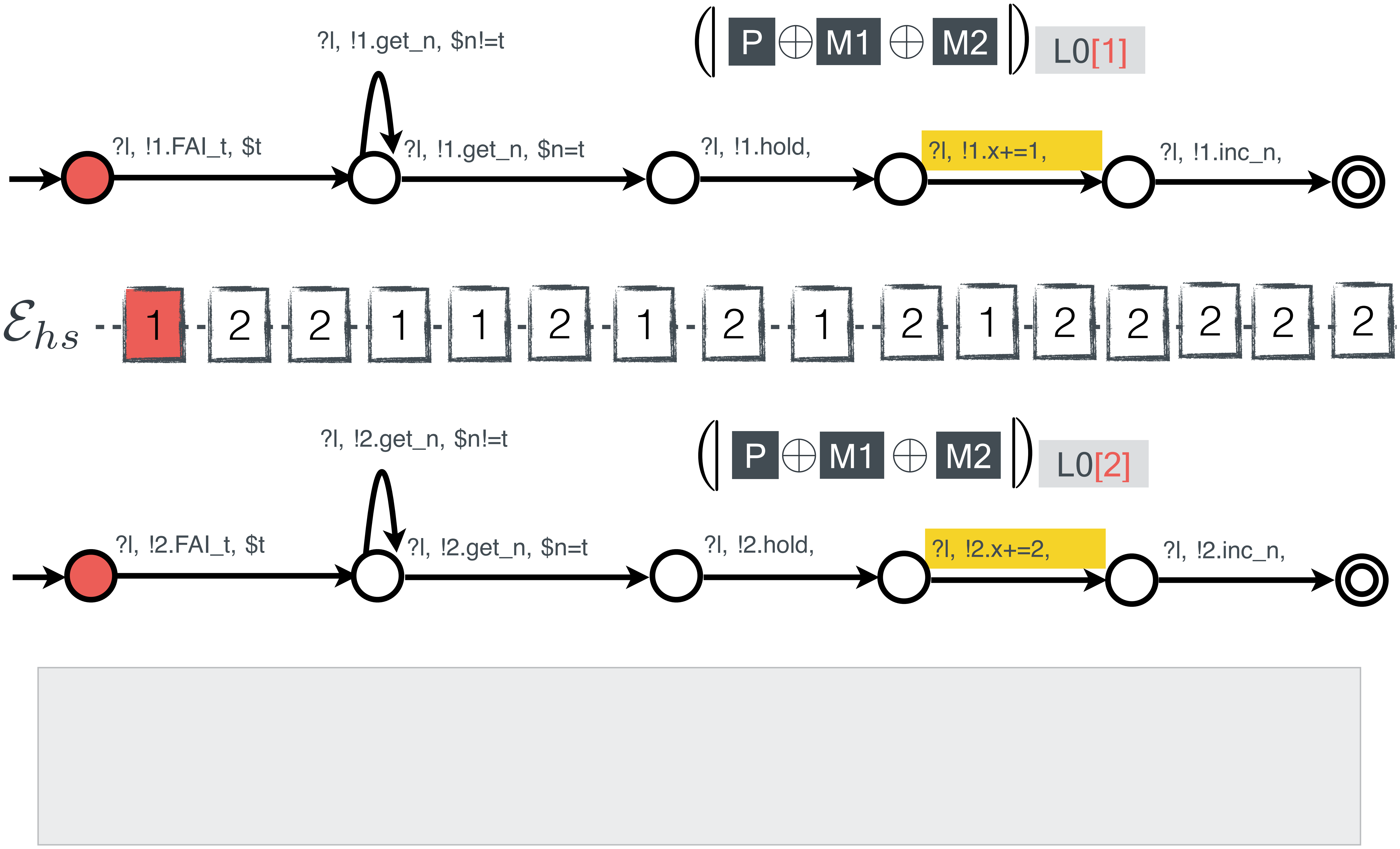
$(\!|\ P \oplus M1 \oplus M2\ |\!)_{L0[1]}$

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



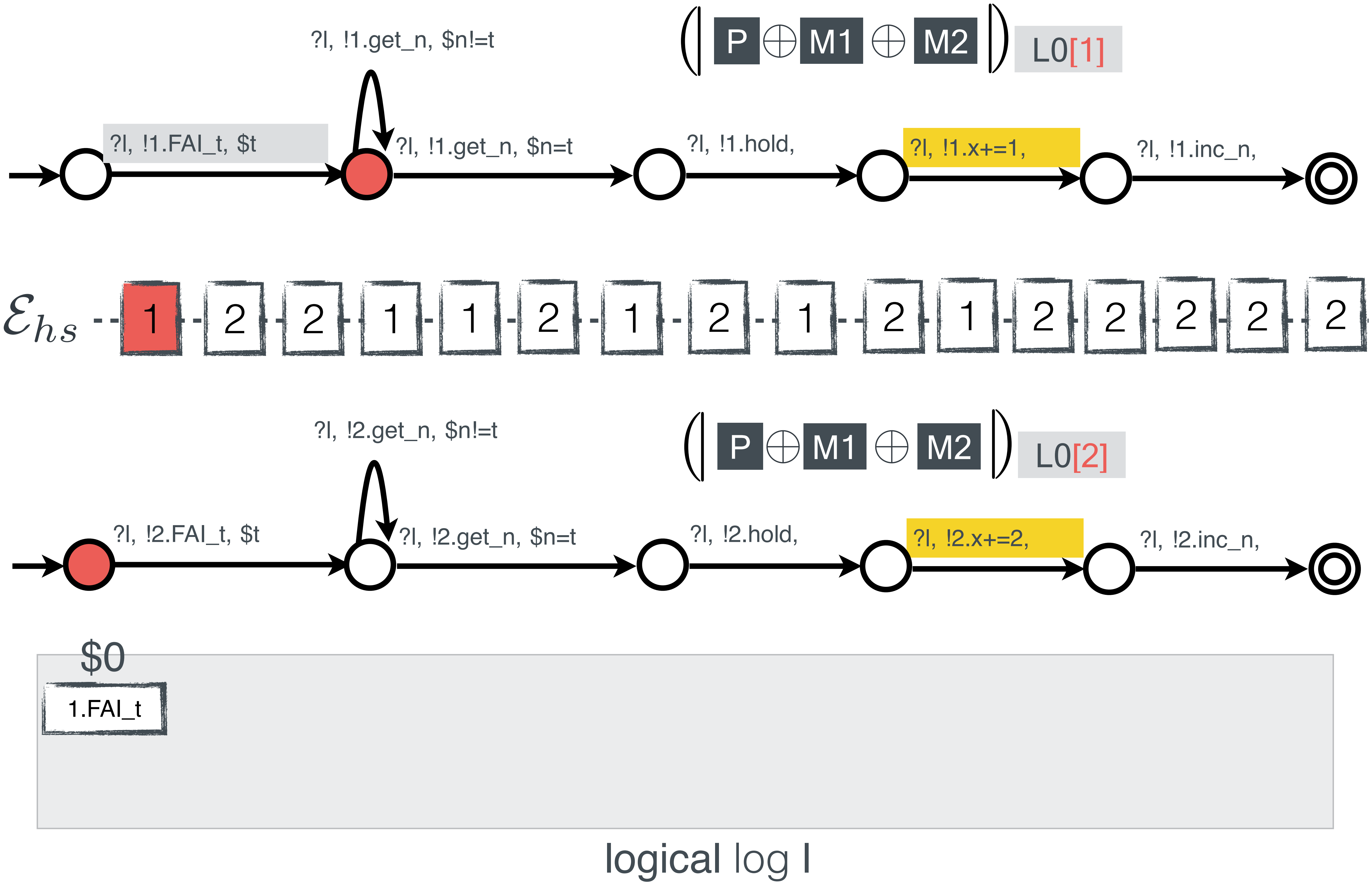
$(\!|\ P \oplus M1 \oplus M2\ |\!)_{L0[2]}$

Strategies and Game Semantics

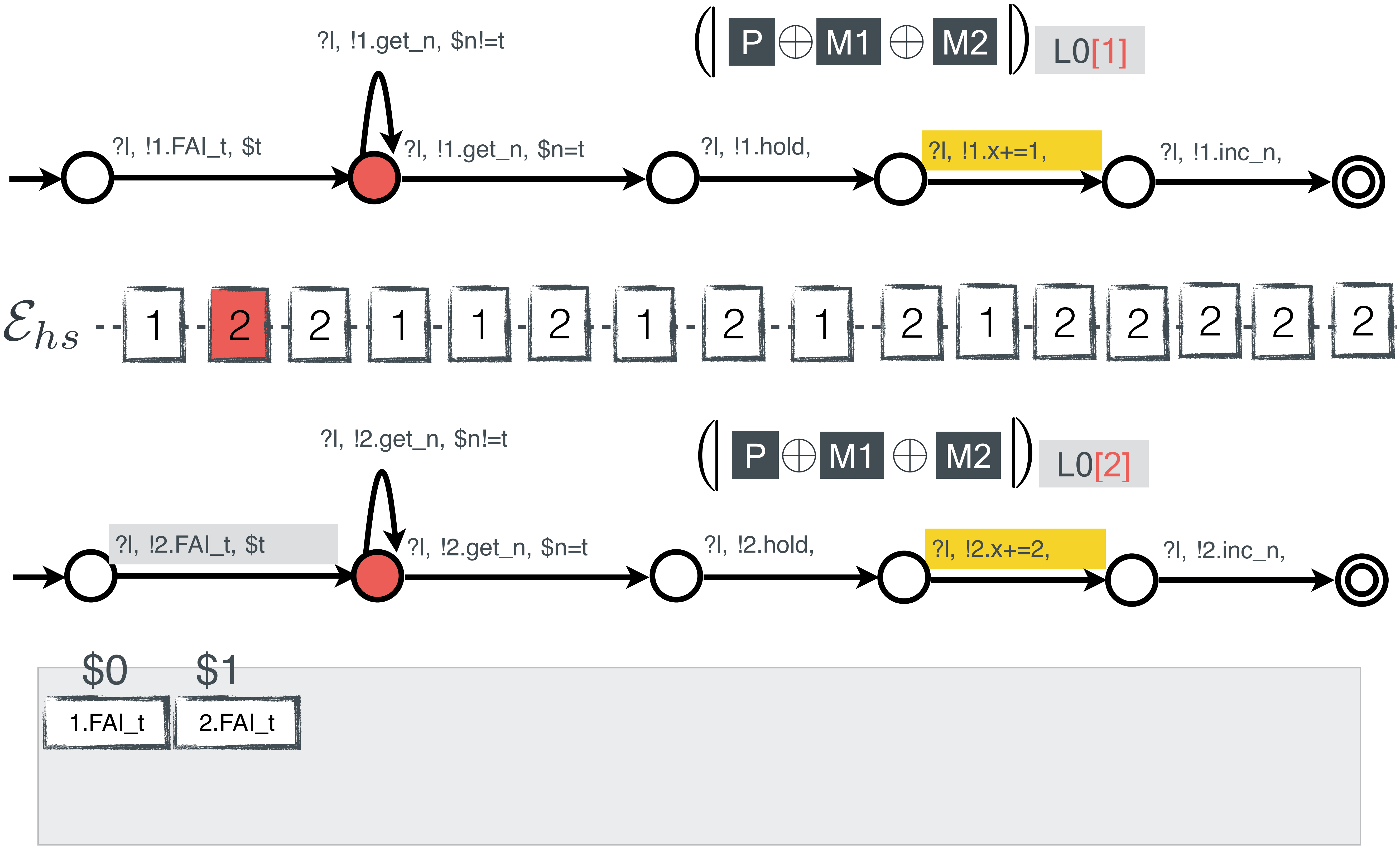


logical log l

Strategies and Game Semantics



Strategies and Game Semantics



(| P ⊕ M1 ⊕ M2 |) L0[1]

?l, !1.FAI_t, $t

?l, !1.get_n, $n!=t

?l, !1.get_n, $n=t

?l, !1.hold,

?l, !1.x+=1,

?l, !1.inc_n,

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

(| P ⊕ M1 ⊕ M2 |) L0[2]

?l, !2.FAI_t, $t

?l, !2.get_n, $n!=t

?l, !2.get_n, $n=t

?l, !2.hold,

?l, !2.x+=2,

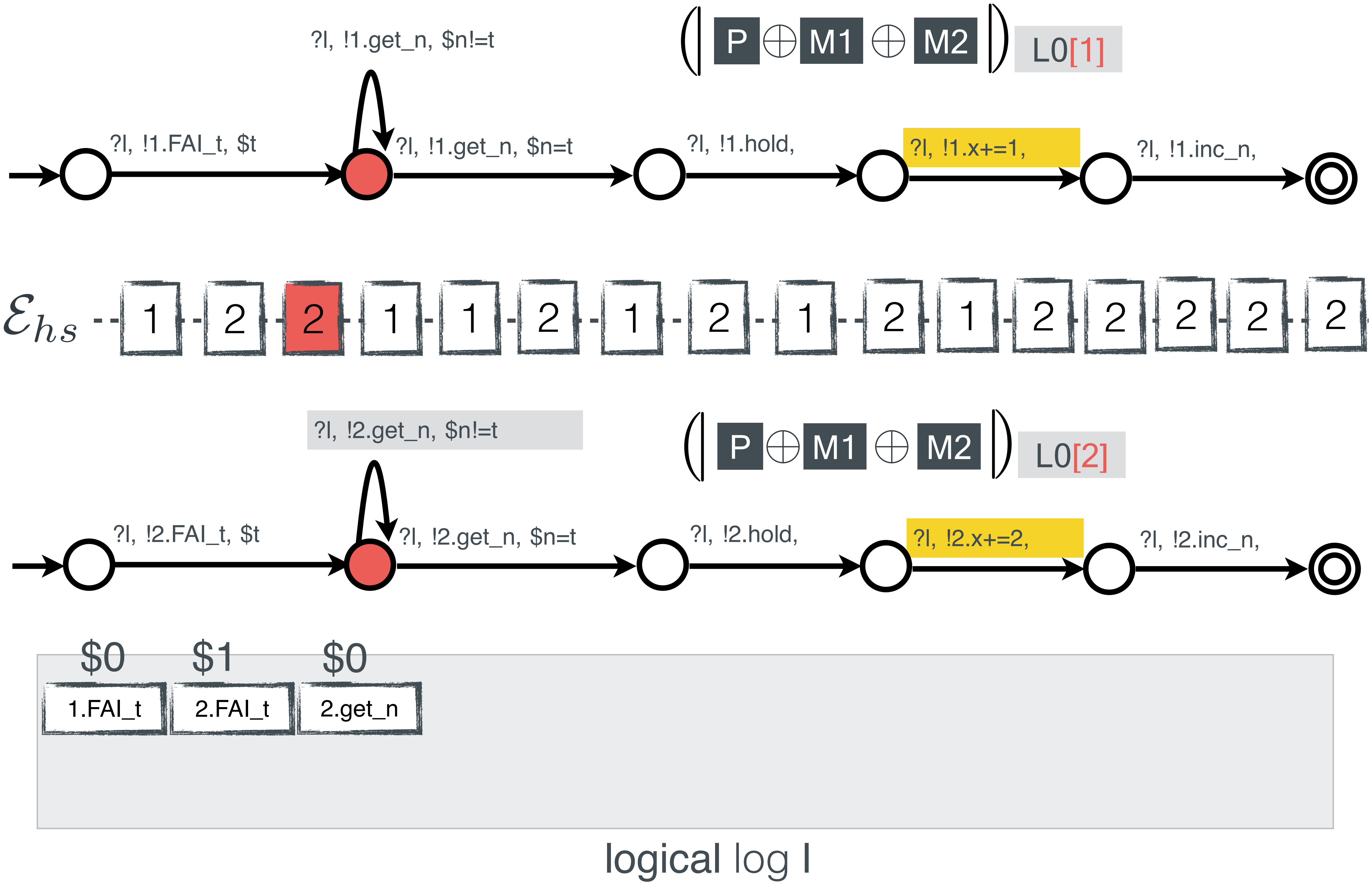
?l, !2.inc_n,

$0 1.FAI_t

$1 2.FAI_t

logical log l

Strategies and Game Semantics



(| P ⊕ M1 ⊕ M2 |) L0[1]

?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,

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

?l, !2.get_n, $n!=t

(| P ⊕ M1 ⊕ M2 |) L0[2]

?l, !2.FAI_t, $t

?l, !2.get_n, $n=t

?l, !2.hold,

?l, !2.x+=2,

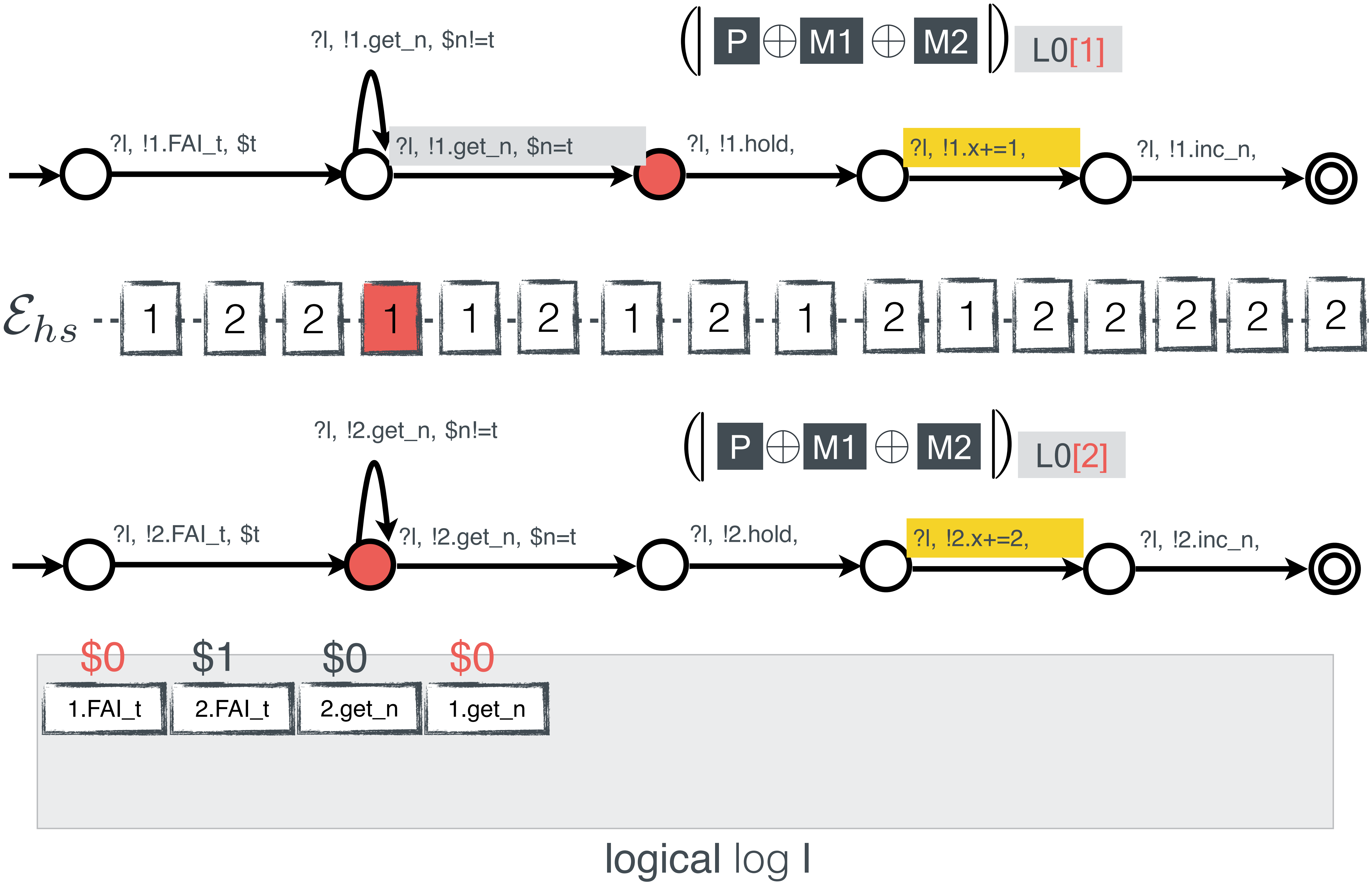
?l, !2.inc_n,

$0 $1 $0

1.FAI_t 2.FAI_t 2.get_n

logical log l

Strategies and Game Semantics

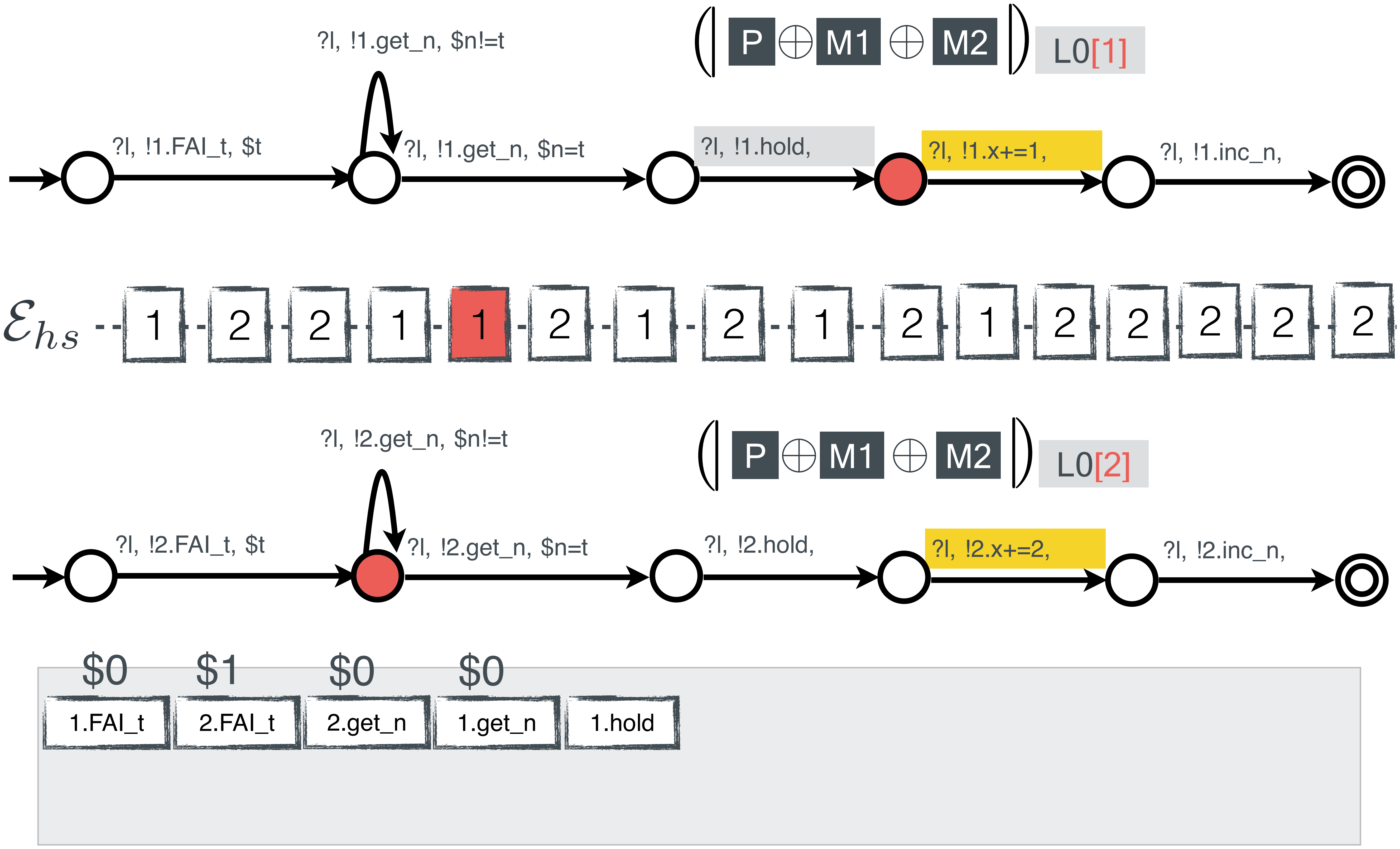


$\mathcal{E}_{hs}$ - - 1 - 2 - 2 - 1 - 1 - 2 - 1 - 2 - 1 - 2 - 1 - 2 - 2 - 2 - 2 - 2

$0	$1	$0	$0
1.FAI_t	2.FAI_t	2.get_n	1.get_n

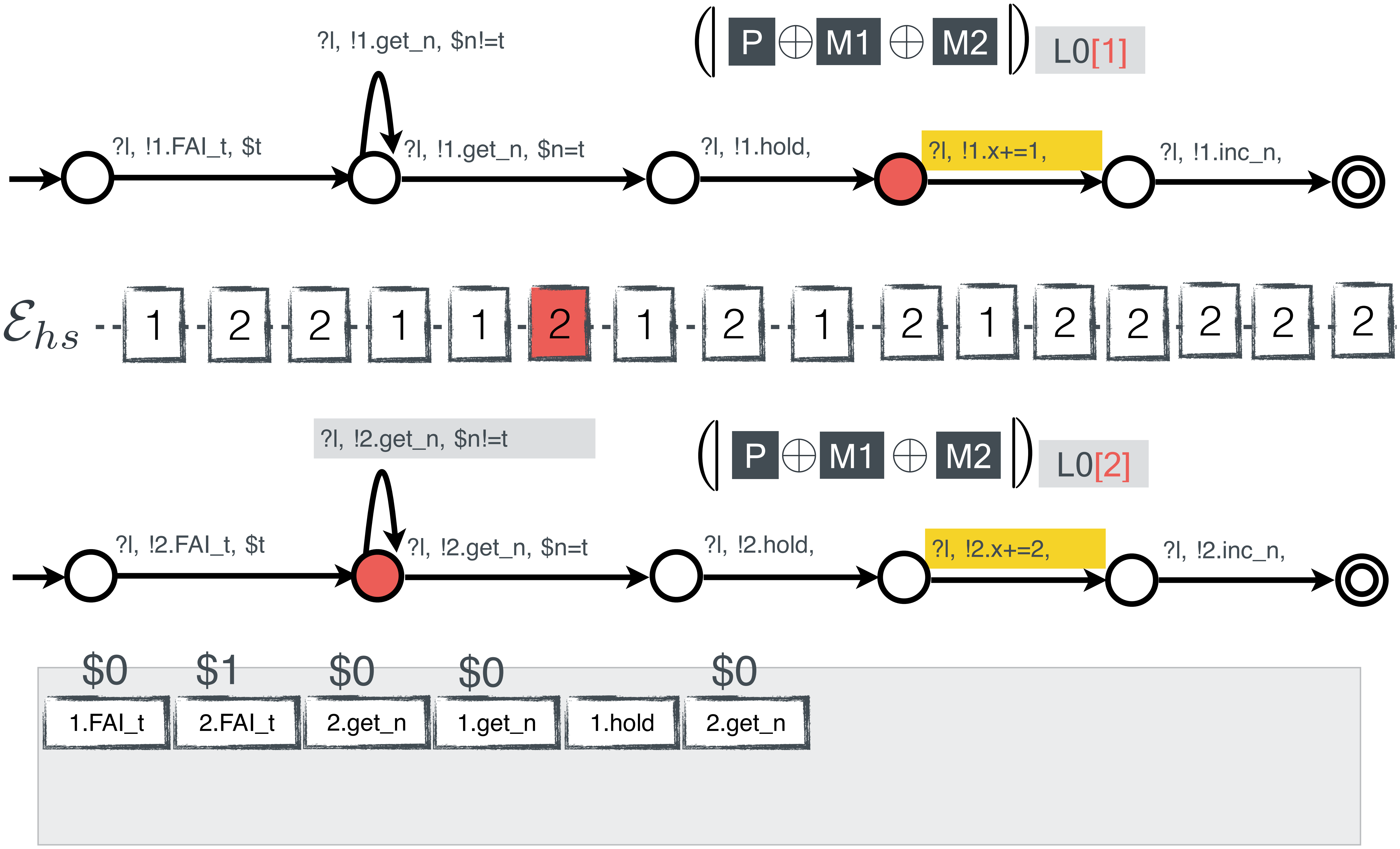
logical log l

Strategies and Game Semantics



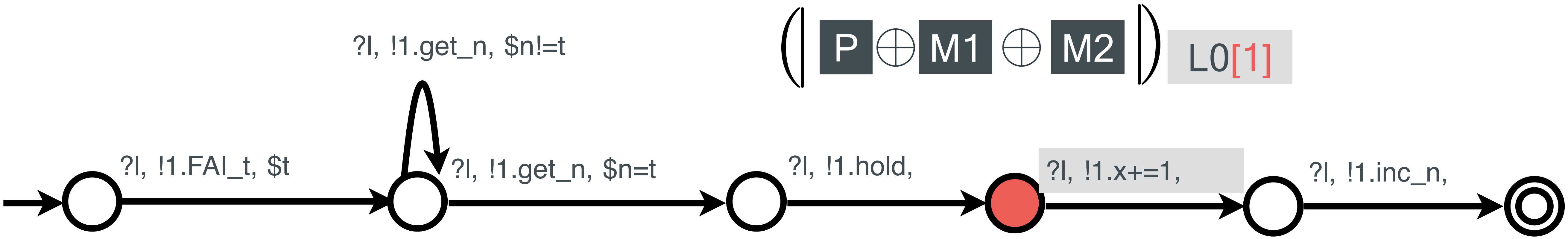
logical log l

Strategies and Game Semantics



logical log l

Strategies and Game Semantics



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

?l, !1.get_n, $n!=t

?l, !1.FAI_t, $t

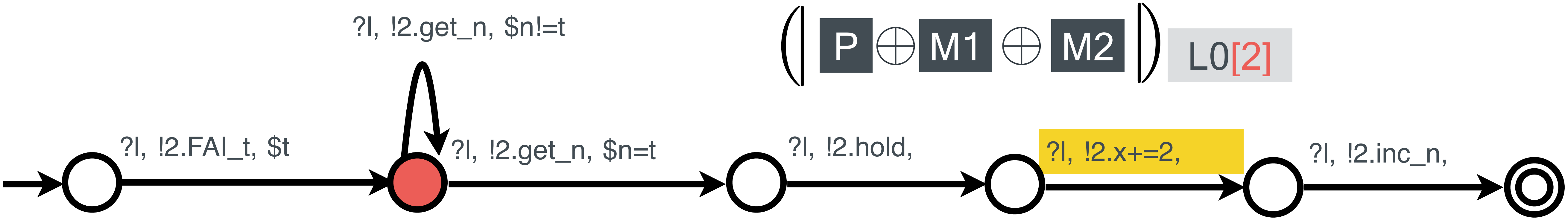
?l, !1.get_n, $n=t

?l, !1.hold,

?l, !1.x+=1,

?l, !1.inc_n,

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



$(\!| P \oplus M1 \oplus M2 |\!)$ L0[2]

?l, !2.get_n, $n!=t

?l, !2.FAI_t, $t

?l, !2.get_n, $n=t

?l, !2.hold,

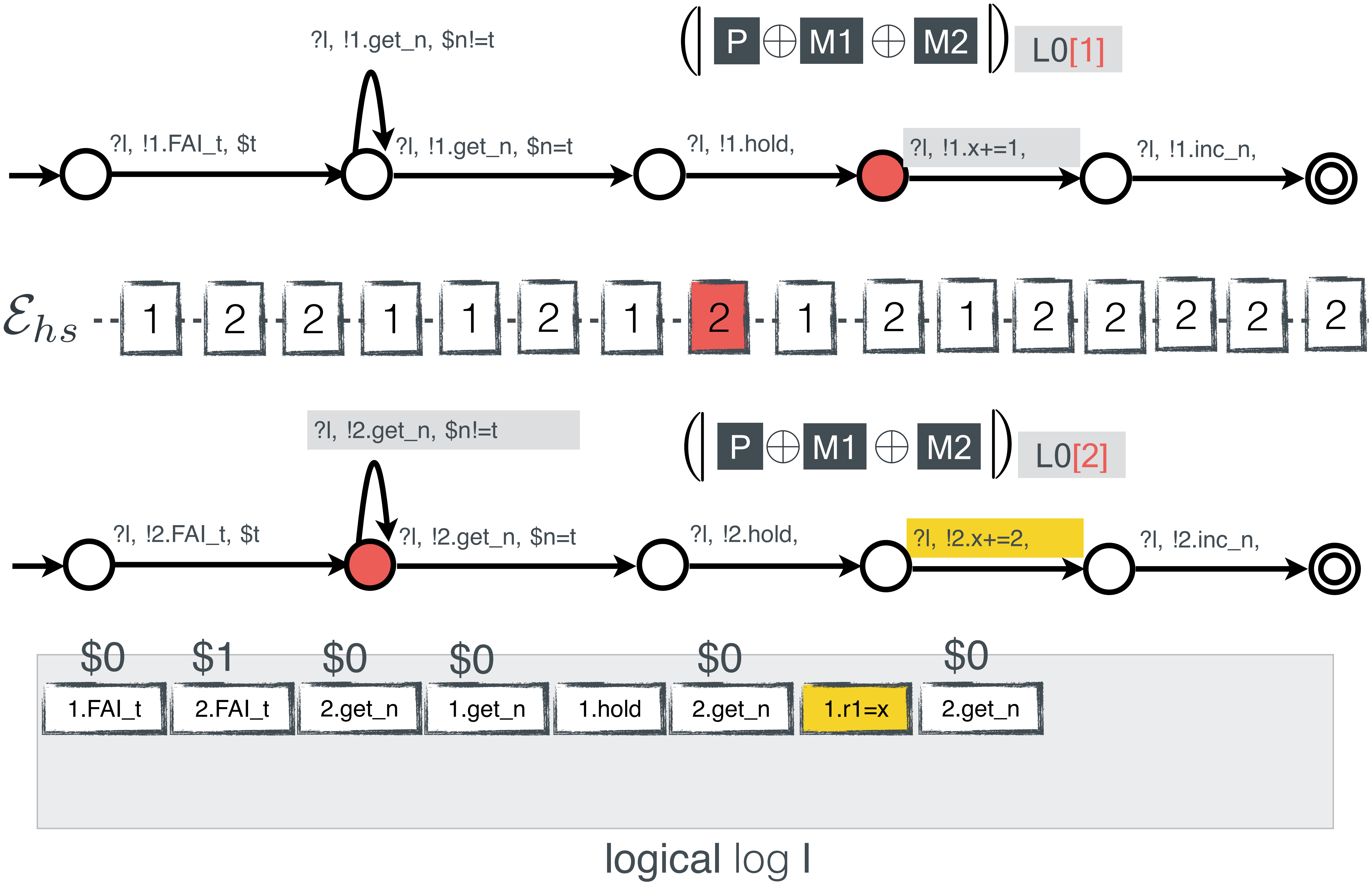
?l, !2.x+=2,

?l, !2.inc_n,

$0	$1	$0	$0		$0	
1.FAI_t	2.FAI_t	2.get_n	1.get_n	1.hold	2.get_n	1.r1=x

logical log l

Strategies and Game Semantics



⦇ P ⊕ M1 ⊕ M2 ⦈ L0[1]

?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,

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

?l, !2.get_n, $n!=t

⦇ P ⊕ M1 ⊕ M2 ⦈ L0[2]

?l, !2.FAI_t, $t

?l, !2.get_n, $n=t

?l, !2.hold,

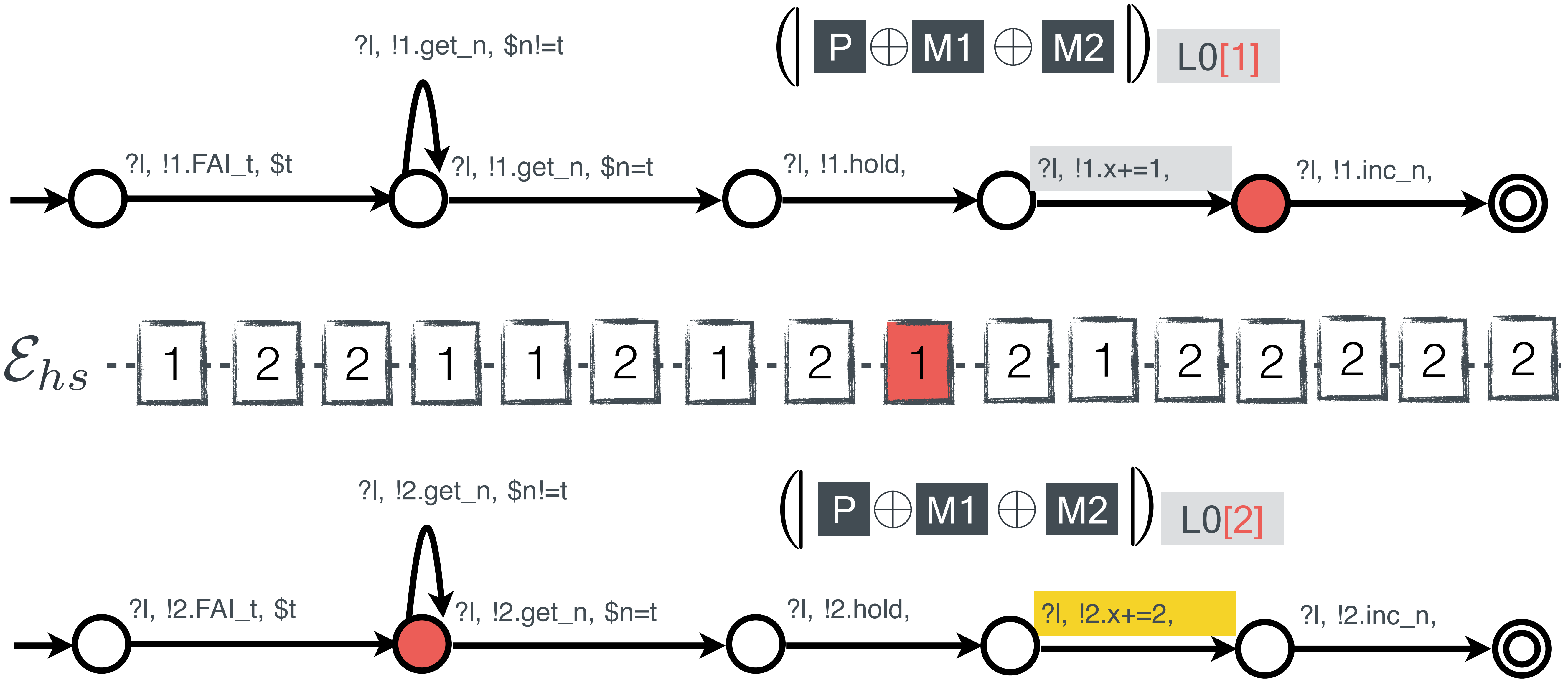
?l, !2.x+=2,

?l, !2.inc_n,

$0	$1	$0	$0		$0		$0
1.FAI_t	2.FAI_t	2.get_n	1.get_n	1.hold	2.get_n	1.r1=x	2.get_n

logical log l

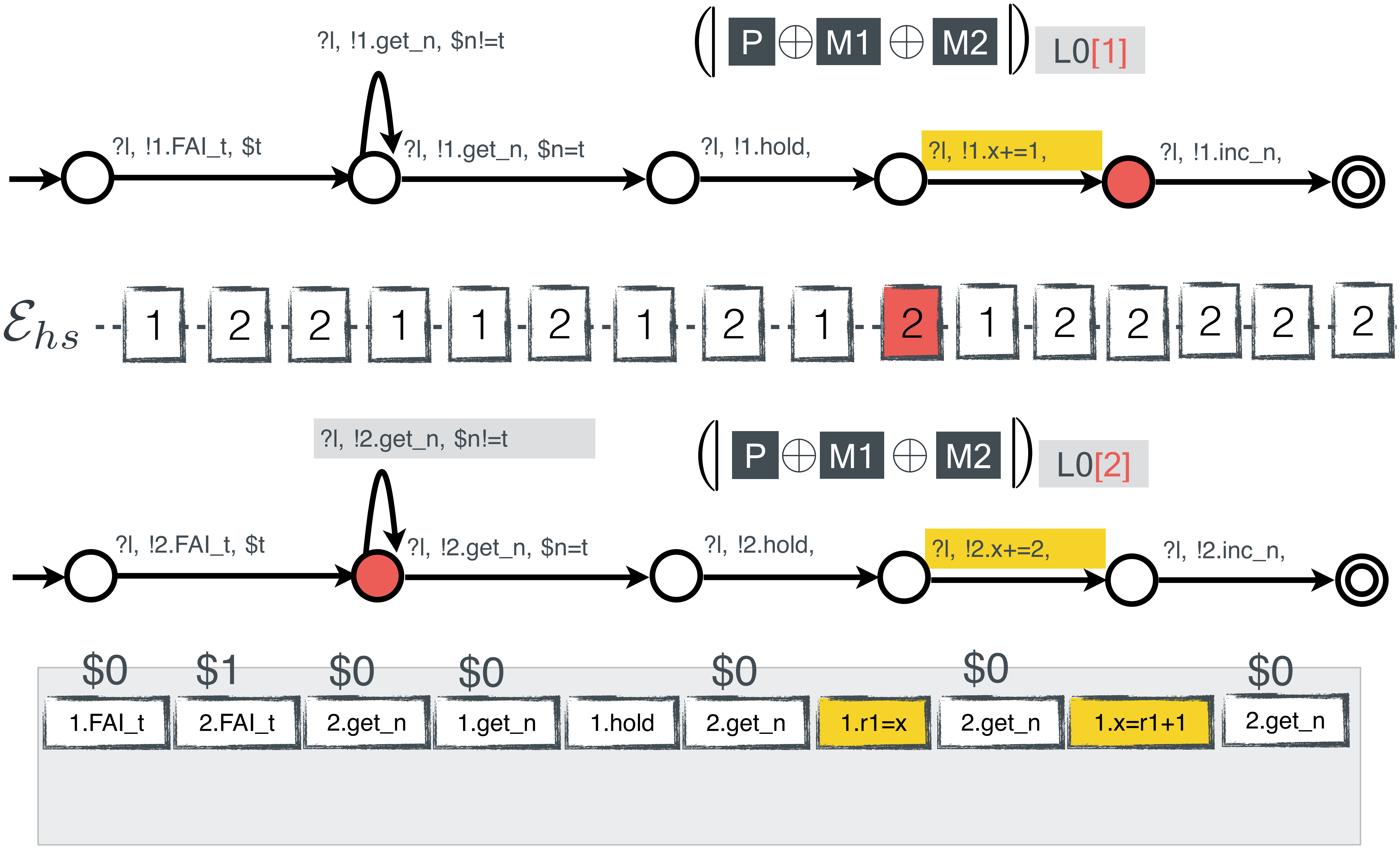
Strategies and Game Semantics



$0	$1	$0	$0		$0		$0	
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

logical log l

Strategies and Game Semantics



$\left(\!\left| \text{P} \oplus \text{M1} \oplus \text{M2} \right|\!\right)$ L0[1]

?I, !1.FAI_t, $t

?I, !1.get_n, $n!=t

?I, !1.get_n, $n=t

?I, !1.hold,

?I, !1.x+=1,

?I, !1.inc_n,

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

?I, !2.get_n, $n!=t

$\left(\!\left| \text{P} \oplus \text{M1} \oplus \text{M2} \right|\!\right)$ L0[2]

?I, !2.FAI_t, $t

?I, !2.get_n, $n=t

?I, !2.hold,

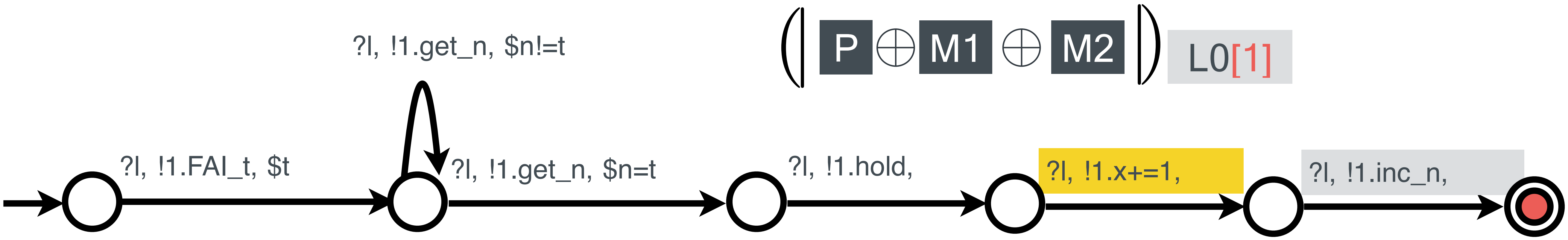
?I, !2.x+=2,

?I, !2.inc_n,

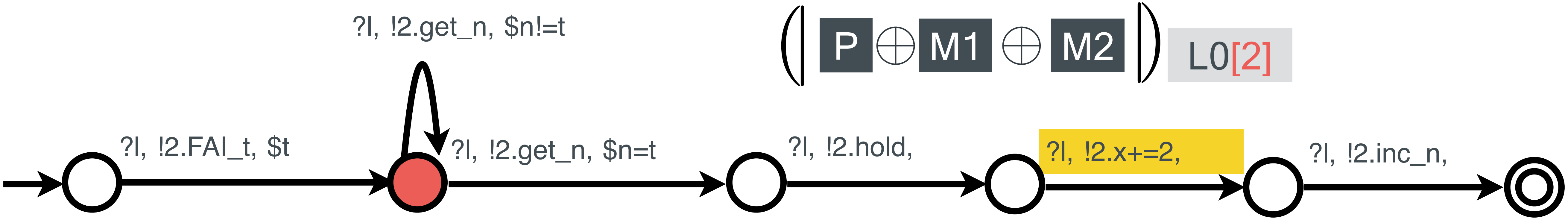
$0	$1	$0	$0		$0		$0		$0
1.FAI_t	2.FAI_t	2.get_n	1.get_n	1.hold	2.get_n	1.r1=x	2.get_n	1.x=r1+1	2.get_n

logical log l

Strategies and Game Semantics



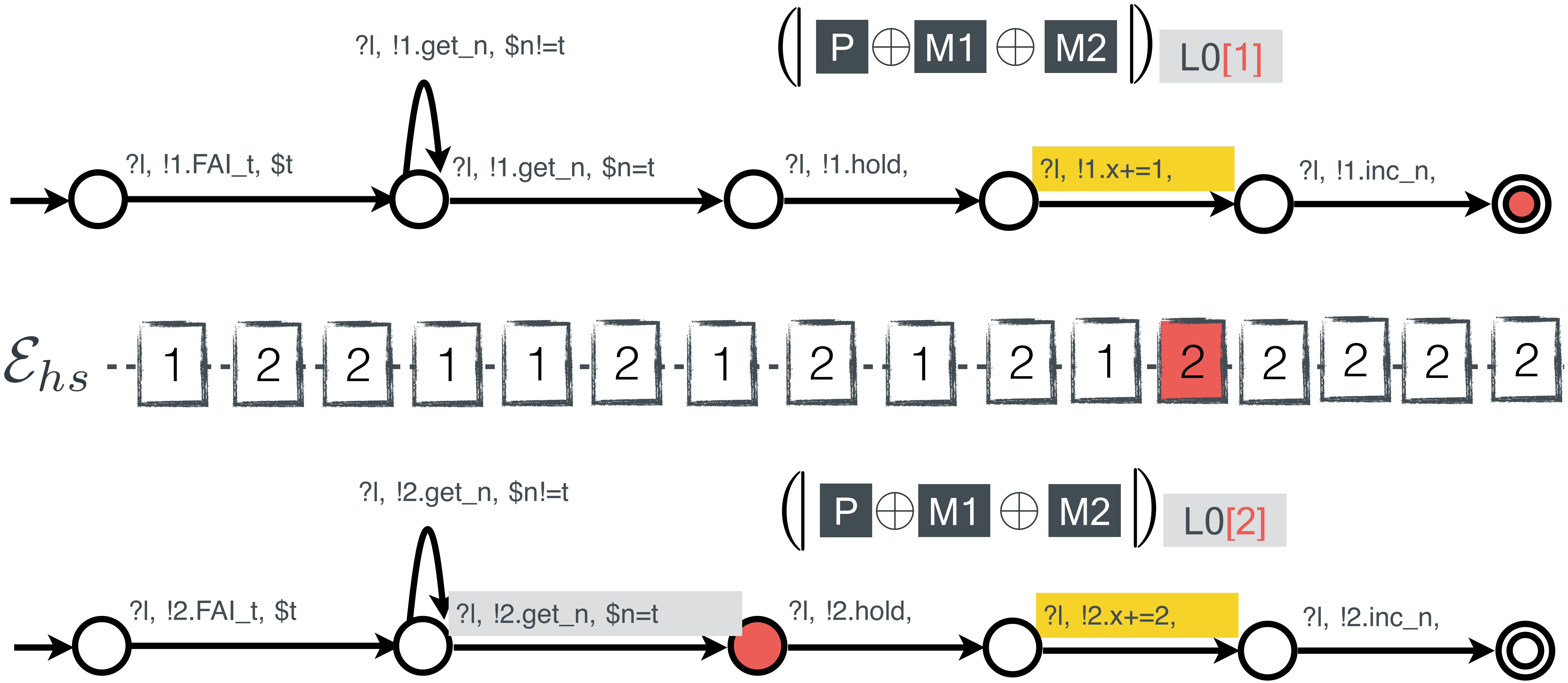
$\mathcal{E}_{hs}$															
1	2	2	1	1	2	1	2	1	2	1	2	2	2	2	2



$0	$1	$0	$0		$0		$0		$0
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									

logical log l

Strategies and Game Semantics



(| P ⊕ M1 ⊕ M2 |) L0[1]

?l, !1.FAI_t, $t

?l, !1.get_n, $n!=t

?l, !1.get_n, $n=t

?l, !1.hold,

?l, !1.x+=1,

?l, !1.inc_n,

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

(| P ⊕ M1 ⊕ M2 |) L0[2]

?l, !2.FAI_t, $t

?l, !2.get_n, $n!=t

?l, !2.get_n, $n=t

?l, !2.hold,

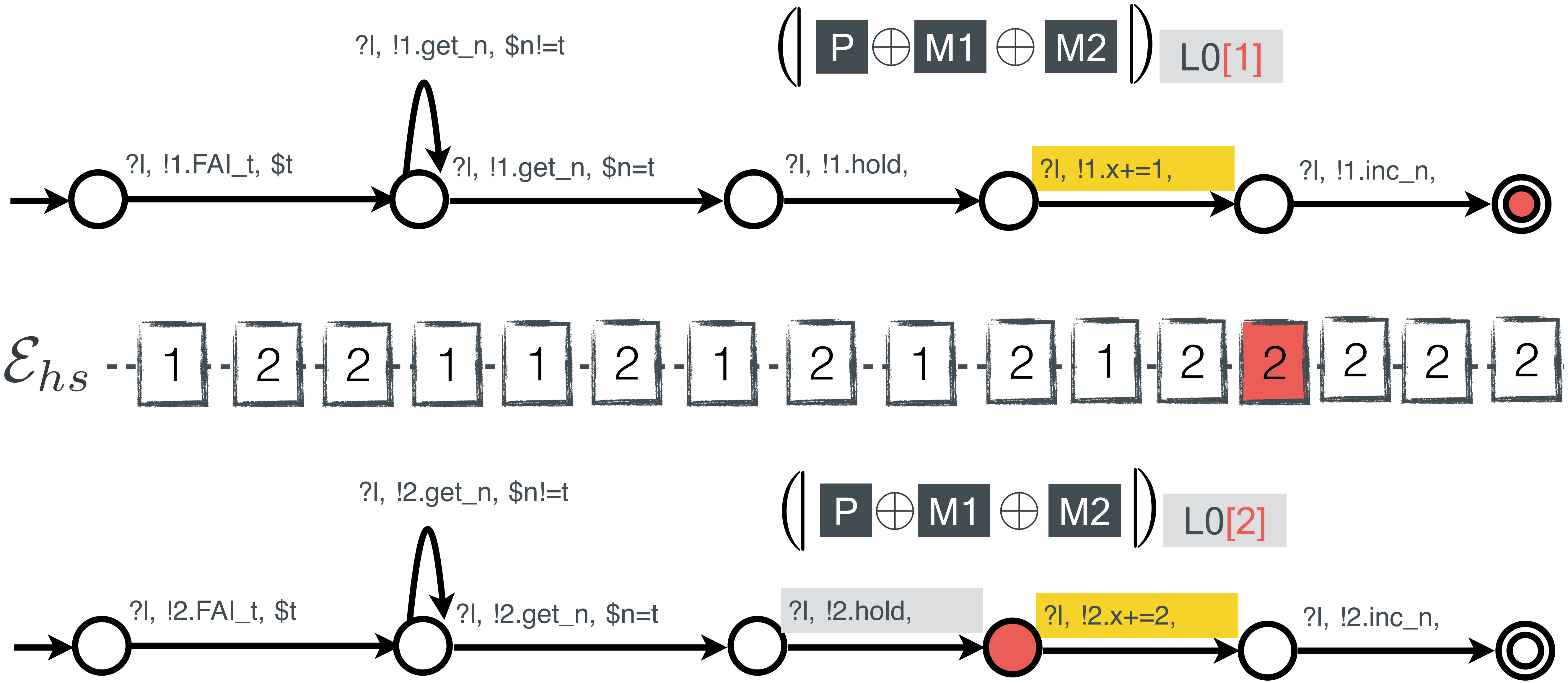
?l, !2.x+=2,

?l, !2.inc_n,

$0	$1	$0	$0		$0		$0		$0
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								
	$1								

logical log l

Strategies and Game Semantics



(| P ⊕ M1 ⊕ M2 |) L0[1]

?l, !1.FAI_t, $t

?l, !1.get_n, $n!=t

?l, !1.get_n, $n=t

?l, !1.hold,

?l, !1.x+=1,

?l, !1.inc_n,

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

(| P ⊕ M1 ⊕ M2 |) L0[2]

?l, !2.FAI_t, $t

?l, !2.get_n, $n!=t

?l, !2.get_n, $n=t

?l, !2.hold,

?l, !2.x+=2,

?l, !2.inc_n,

$0 1.FAI_t

$1 2.FAI_t

$0 2.get_n

$0 1.get_n

1.hold

$0 2.get_n

1.r1=x

$0 2.get_n

1.x=r1+1

$0 2.get_n

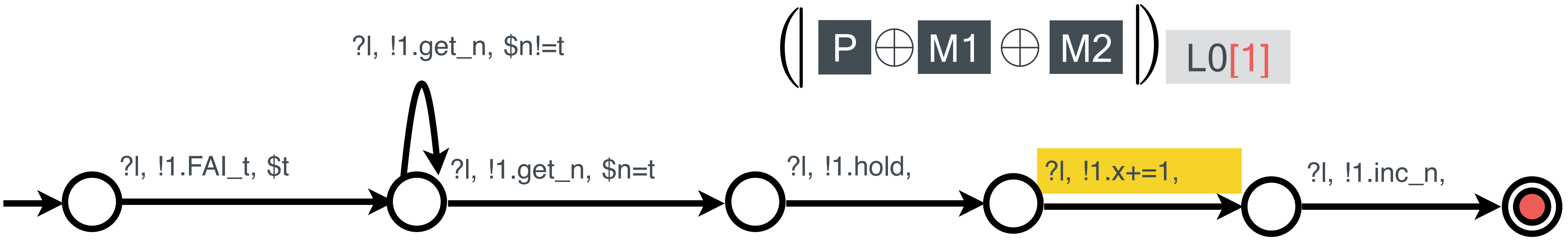
1.inc_n

2.get_n $1

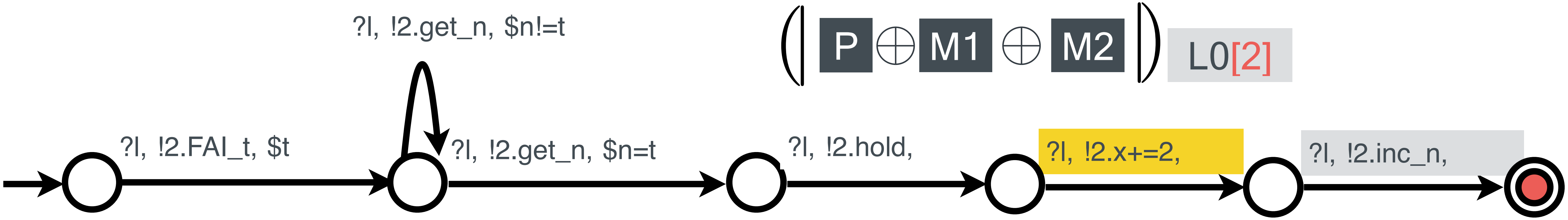
2.hold

logical log l

Strategies and Game Semantics



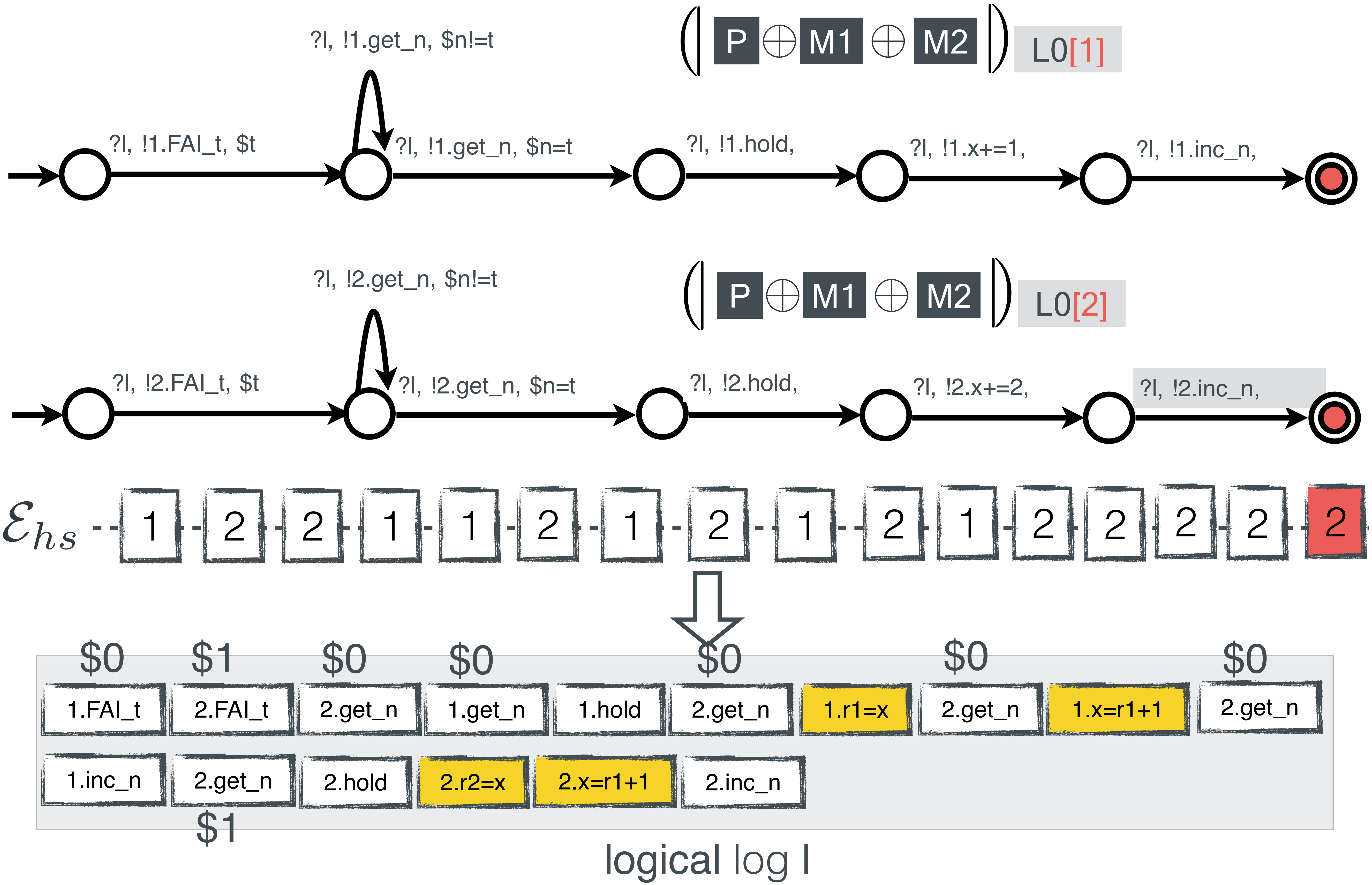
$\mathcal{E}_{hs}$ 1 2 2 1 1 2 1 2 1 2 1 2 2 2 2 2



$0	$1	$0	$0		$0		$0		$0
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				
	$1								

logical log l

Strategies and Game Semantics



$(\!|$ P ⊕ M1 ⊕ M2 $|\!)$ L0[1]

?l, !1.FAI_t, $t

?l, !1.get_n, $n!=t

?l, !1.get_n, $n=t

?l, !1.hold,

?l, !1.x+=1,

?l, !1.inc_n,

$(\!|$ P ⊕ M1 ⊕ M2 $|\!)$ L0[2]

?l, !2.FAI_t, $t

?l, !2.get_n, $n!=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

$0 1.FAI_t

$1 2.FAI_t

$0 2.get_n

$0 1.get_n

1.hold

$0 2.get_n

1.r1=x

$0 2.get_n

1.x=r1+1

$0 2.get_n

1.inc_n

2.get_n $1

2.hold

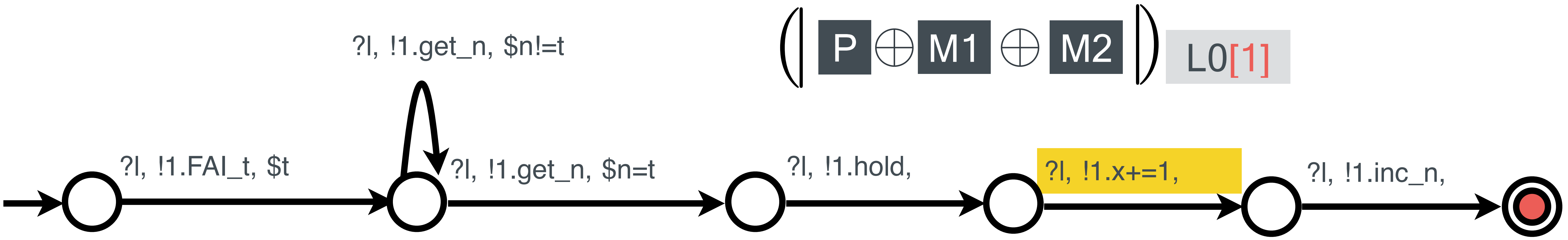
2.r2=x

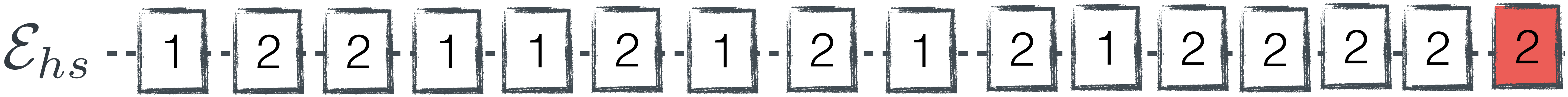
2.x=r1+1

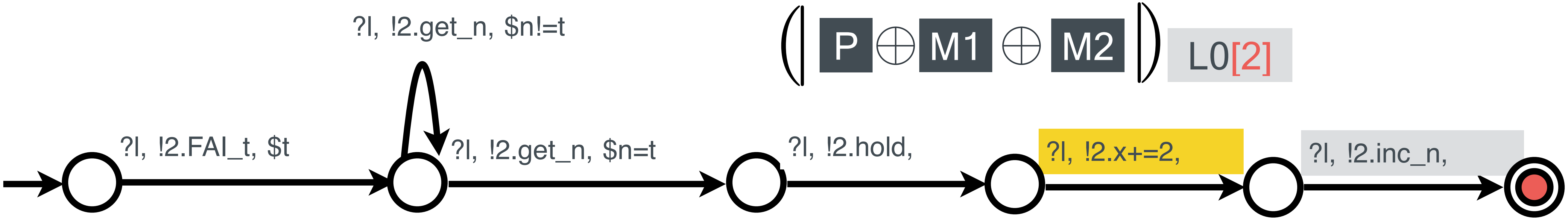
2.inc_n

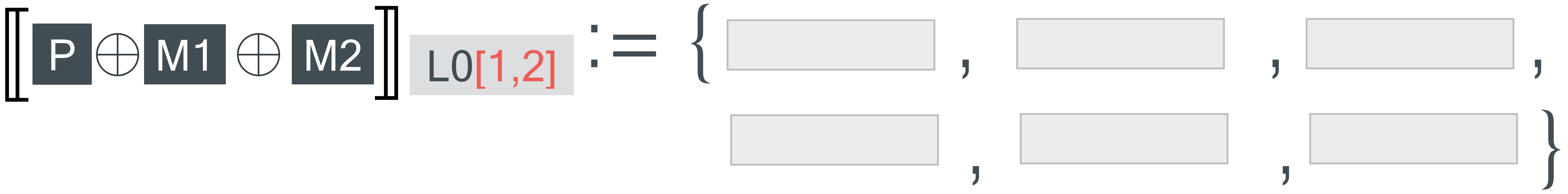
logical log l

Strategies and Game Semantics



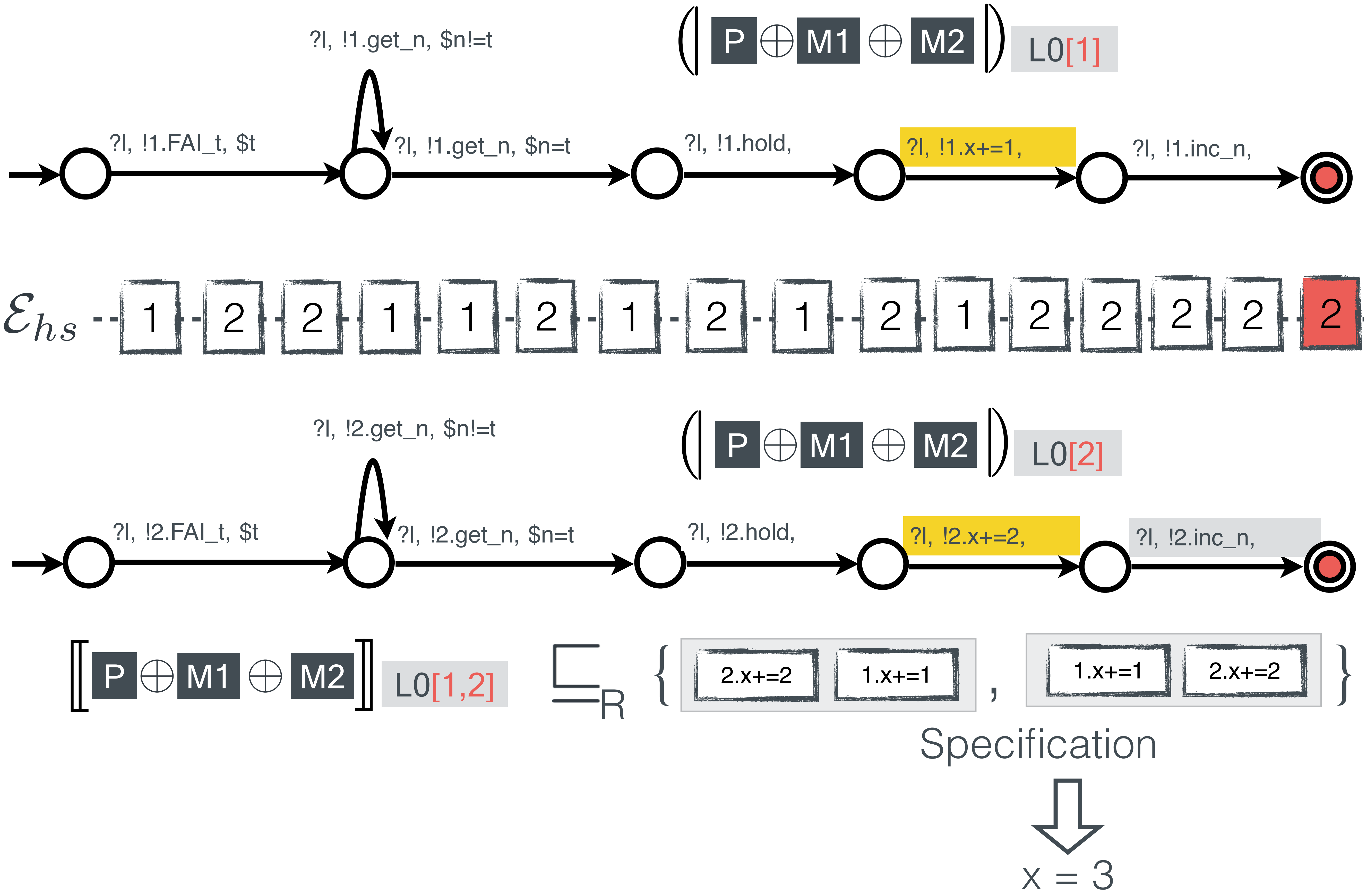






Set of **logical** logs

Strategies and Game Semantics



(| P ⊕ M1 ⊕ M2 |) L0[1]

?I, !1.get_n, $n!=t

?I, !1.FAI_t, $t

?I, !1.get_n, $n=t

?I, !1.hold,

?I, !1.x+=1,

?I, !1.inc_n,

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

(| P ⊕ M1 ⊕ M2 |) L0[2]

?I, !2.get_n, $n!=t

?I, !2.FAI_t, $t

?I, !2.get_n, $n=t

?I, !2.hold,

?I, !2.x+=2,

?I, !2.inc_n,

⟦P ⊕ M1 ⊕ M2⟧ L0[1,2] $\sqsubseteq_R$ { 2.x+=2 1.x+=1 , 1.x+=1 2.x+=2 }

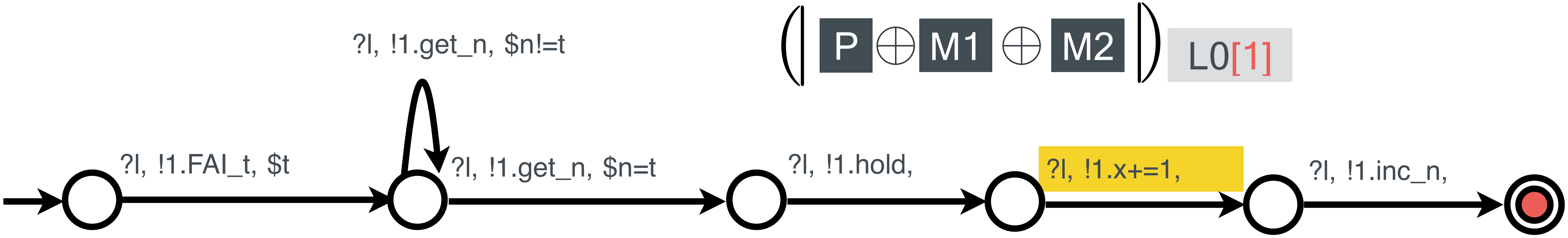
Specification

⇓

x = 3

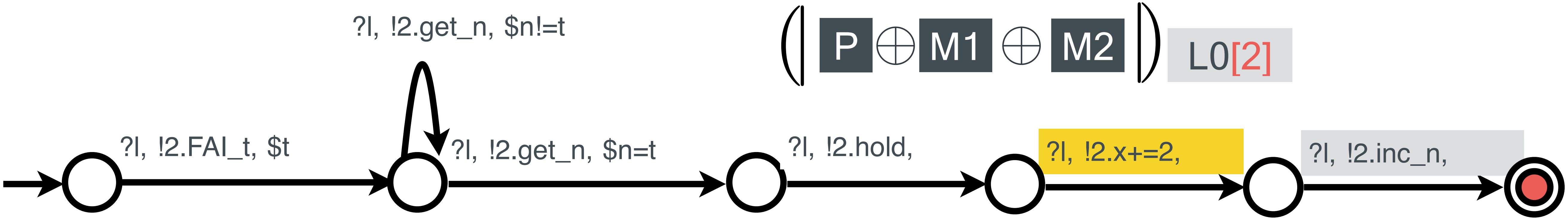
Strategies and Game Semantics

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



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

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



$[\![P \oplus M1 \oplus M2]\!]$ L0[1,2] $\sqsubseteq_R$ { 2.x+=2 1.x+=1 , 1.x+=1 2.x+=2 }

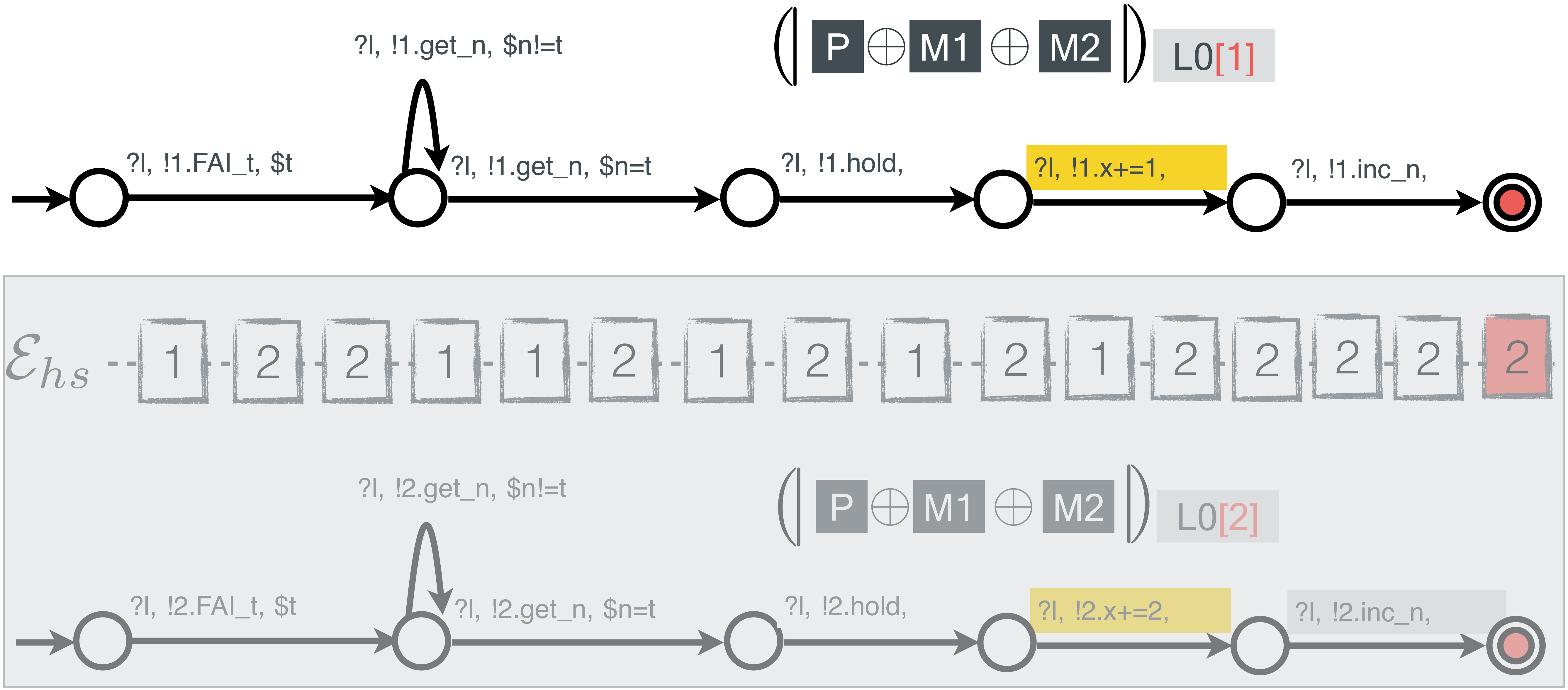
✗ 1.r1=x 2.r2=x 2.x=r1+1 1.x=r1+1

Specification

⇓

x = 3

Strategy Refinement



(| P ⊕ M1 ⊕ M2 |) L0[1]

?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,

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

(| P ⊕ M1 ⊕ M2 |) L0[2]

?l, !2.get_n, $n!=t

?l, !2.FAI_t, $t

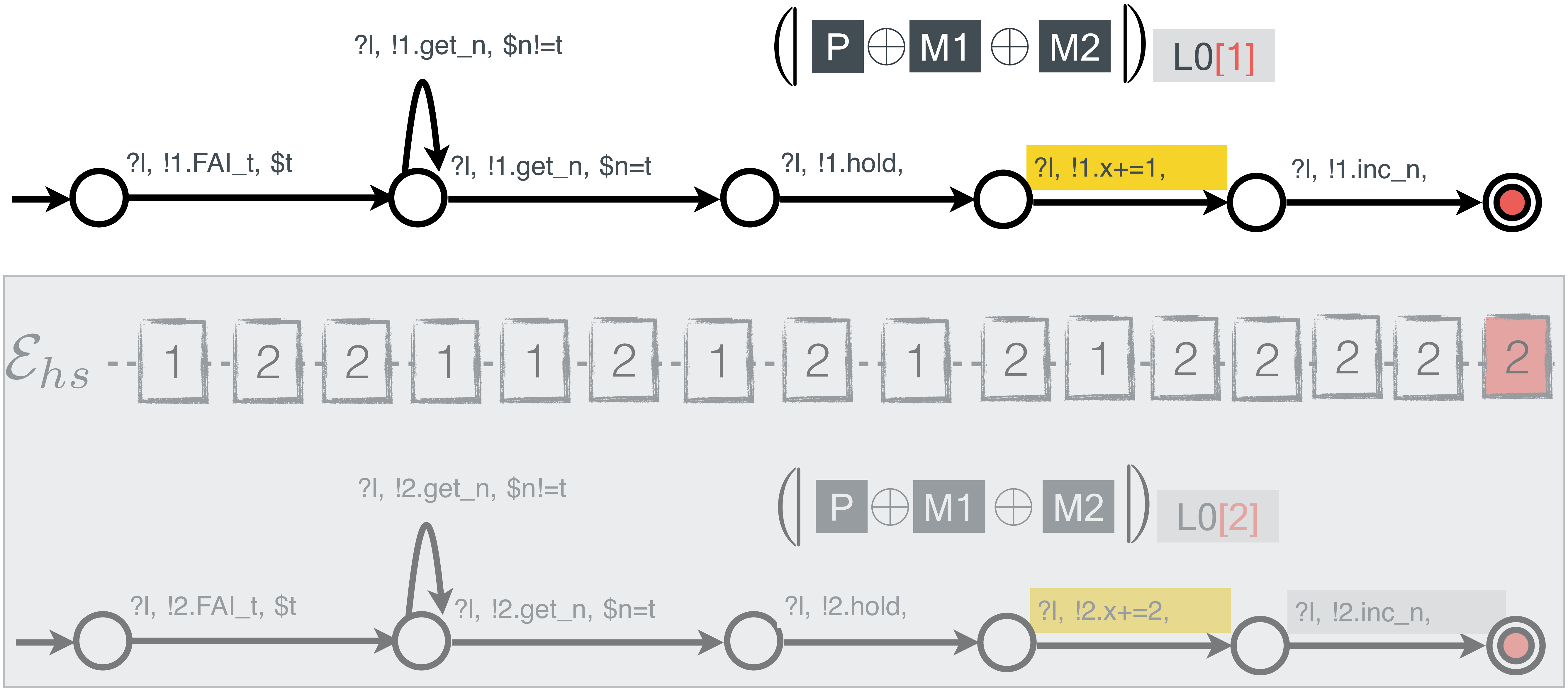
?l, !2.get_n, $n=t

?l, !2.hold,

?l, !2.x+=2,

?l, !2.inc_n,

Strategy Refinement



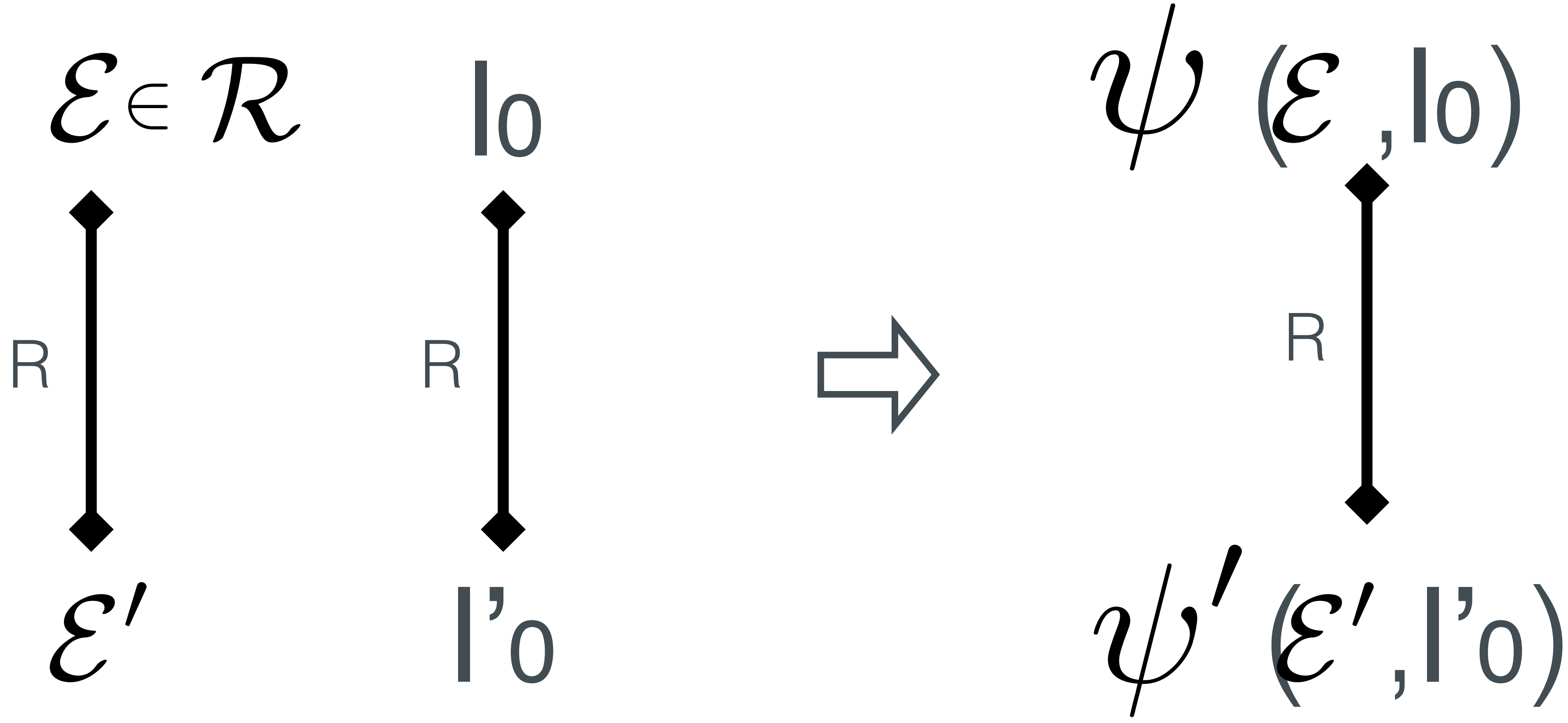
Environment Context $\mathcal{E} \in \mathcal{R}$

Strategy Refinement

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

Strategy Refinement

$$\boxed{\psi \;\leq_{\mathrm{R}}\; \psi'}$$

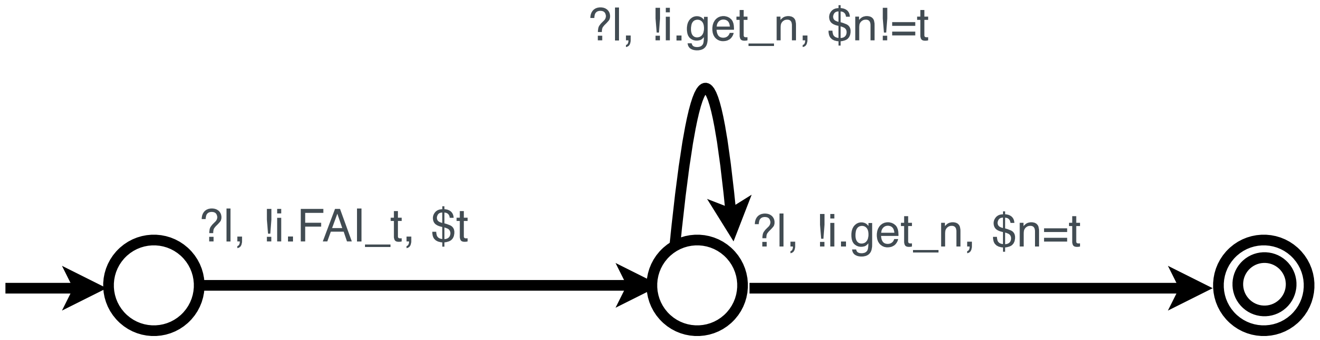

$\mathcal{E} \in \mathcal{R}$
R
$\mathcal{E}'$
l0
R
l’0
$\Rightarrow$
$\psi\,(\mathcal{E}, \mathrm{l}_0)$
R
$\psi'(\mathcal{E}', \mathrm{l'}_0)$

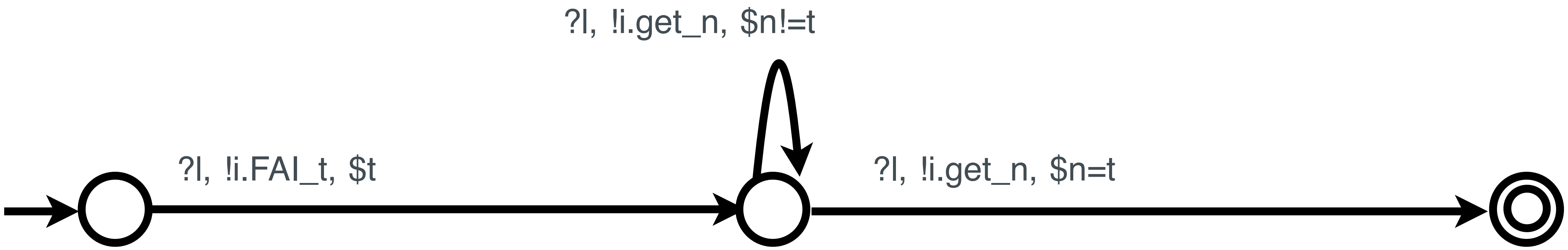
Strategy Refinement

$\llparenthesis$ Macq $\rrparenthesis$ L0[i]

$\leq_R$

ψ_{acq}[i]



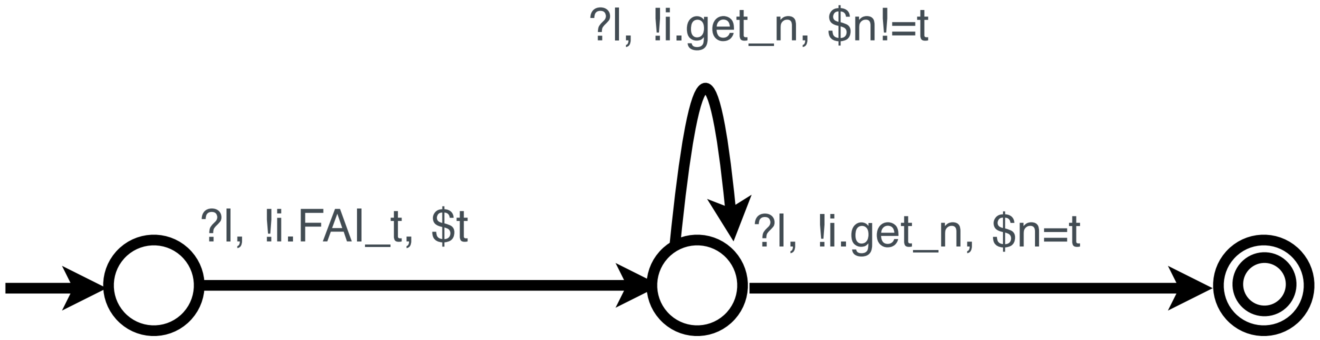


Strategy Refinement

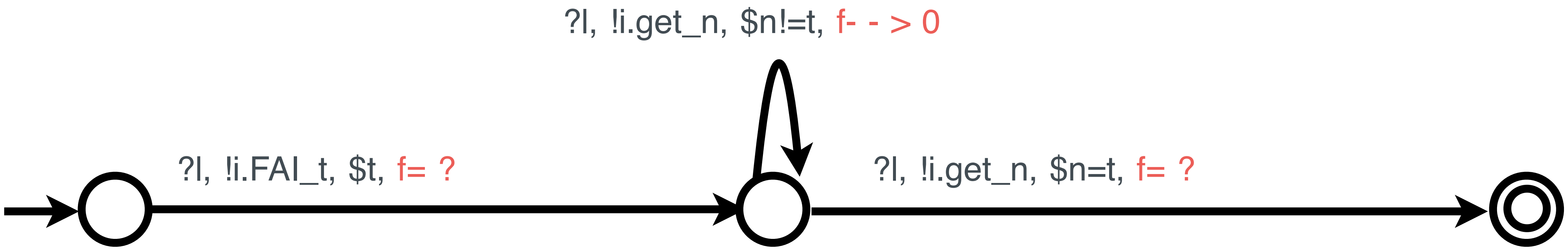
$$\llparenthesis \boxed{\mathrm{Macq}} \rrparenthesis_{\mathrm{L}_0[\mathrm{i}]}$$

$$\leq_R$$

$$\psi_{acq}[\mathrm{i}]$$



Add fuel (f) to prove liveness

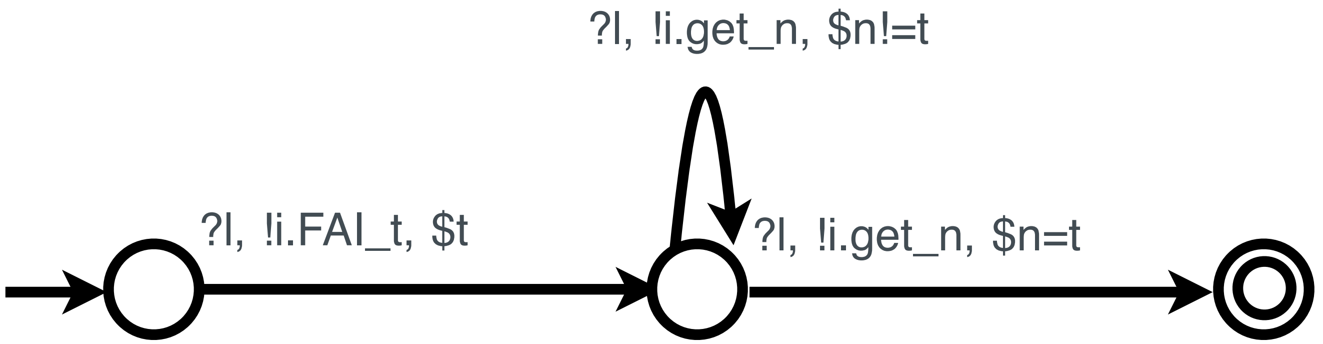


Strategy Refinement

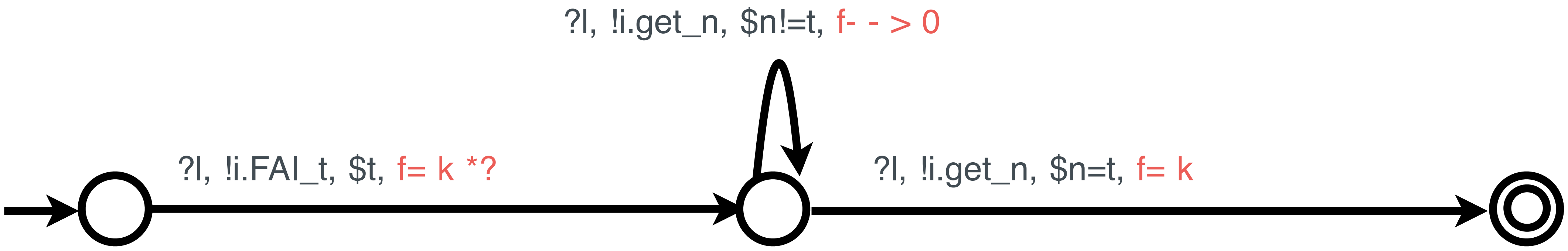
$\llparenthesis \text{Macq} \rrparenthesis$ L0[i]

$\leq_R$

ψ_{acq}[i]



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

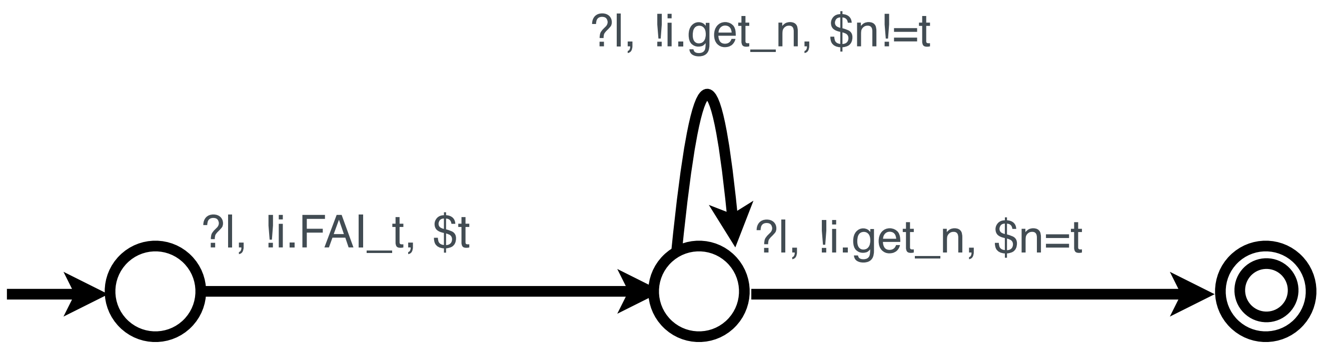


Strategy Refinement

$(\!|$ Macq $|\!)$ L0[i]

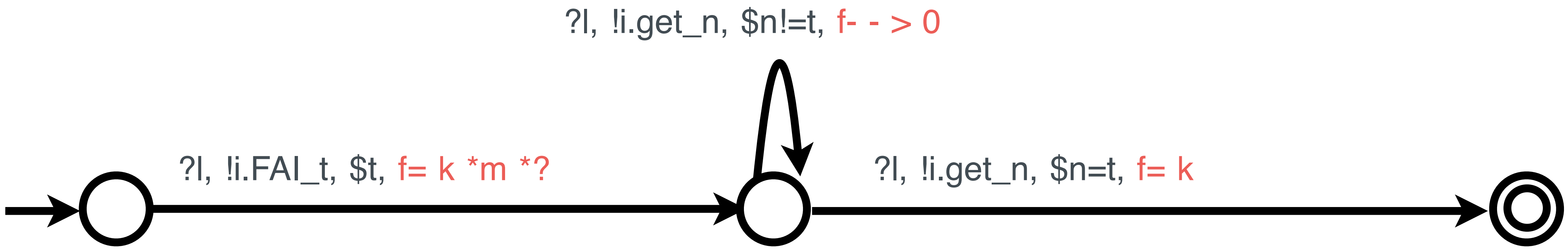
$\leq_{R}$

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

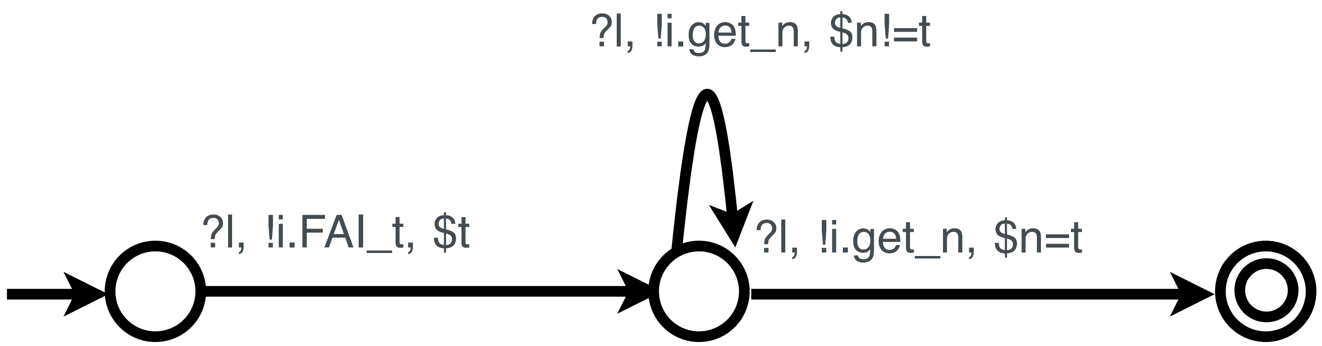


Strategy Refinement

$(\!|\, M_{acq} \,|\!)\, L_0[i]$

$\leq_R$

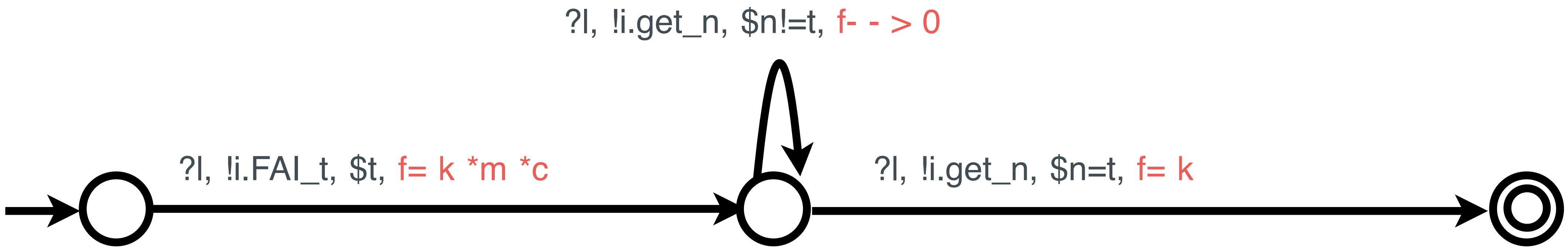
$\psi_{acq}[i]$



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

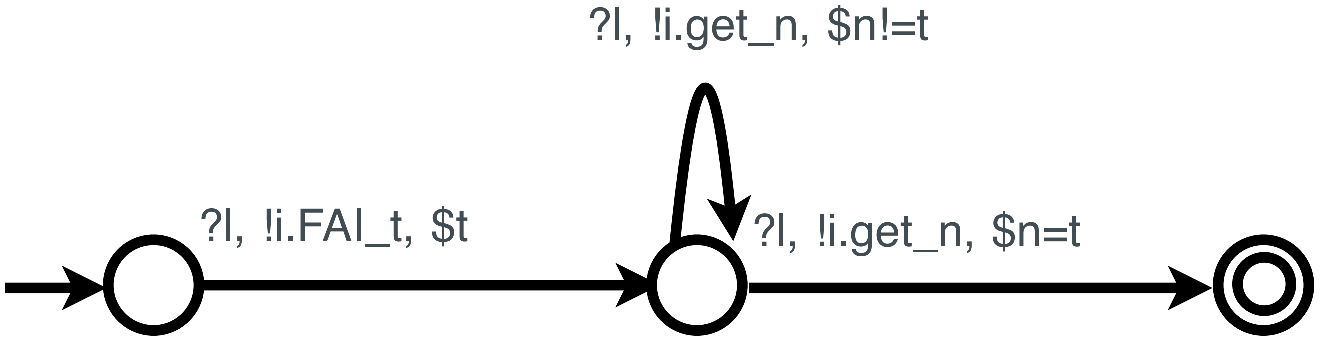
$\mathcal{R}_{hs}$: (fairness) each CPU will be rescheduled within m steps

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



Strategy Refinement

$$\llparenthesis \text{Macq} \rrparenthesis \text{L}_0[\text{i}] \leq_R \psi_{acq}[\text{i}]$$



?l, !i.get_n, $n!=t

?l, !i.FAI_t, $t

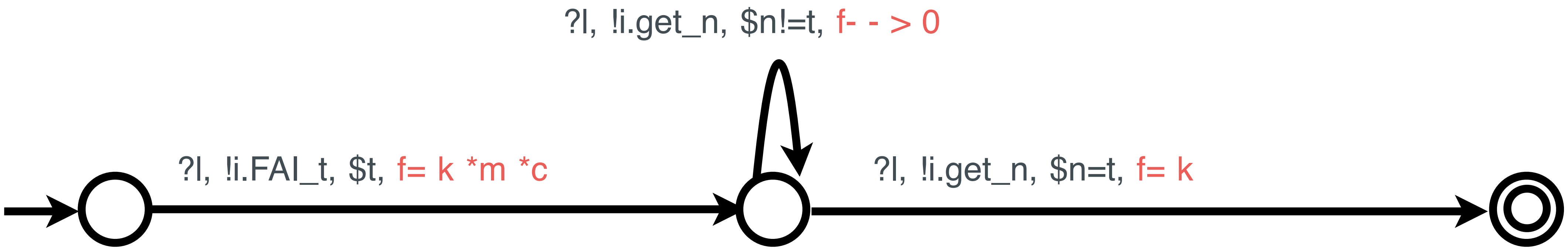
?l, !i.get_n, $n=t

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



?l, !i.get_n, $n!=t, f- - > 0

?l, !i.FAI_t, $t, f= k *m *c

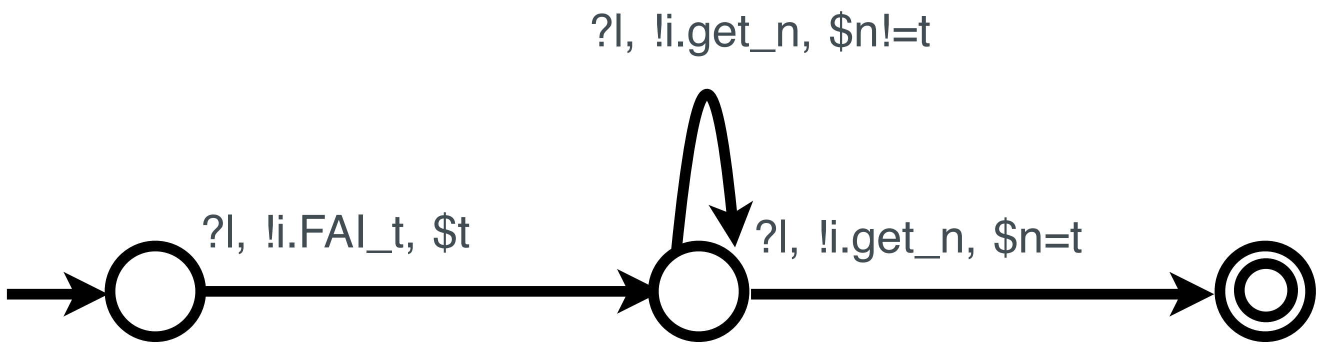
?l, !i.get_n, $n=t, f= k

Strategy Refinement

$$\left(\!\left| \boxed{\text{Macq}} \right|\!\right) \boxed{\text{L}_0[\text{i}]}$$

$$\leq_{\text{R}}$$

$$\psi_{acq}[\text{i}]$$

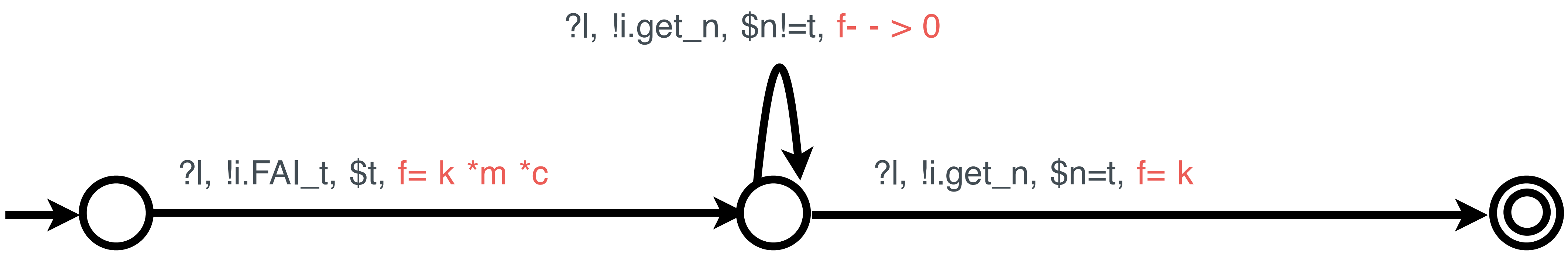


$\mathcal{R}_{j \neq i}$: will release lock within k steps

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

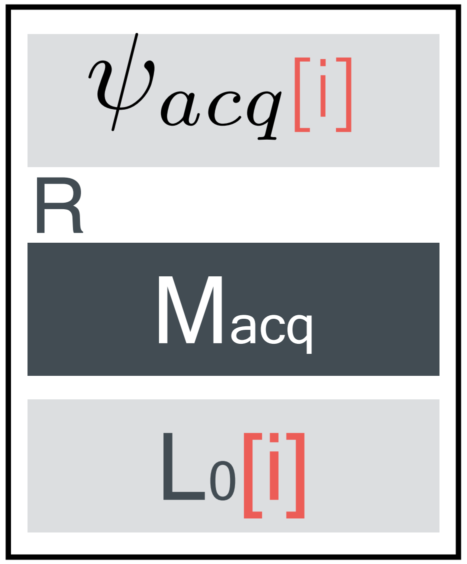
$\mathcal{R}_{cpu}$: #CPU = c < 2^{32}

mutual exclusion?



Certified Concurrent Abstraction Layer

$$(\!|\ \mathrm{M_{acq}}\ |\!)_{\mathrm{L_0[i]}} \leq_{\mathrm{R}} \psi_{acq}\mathrm{[i]}$$



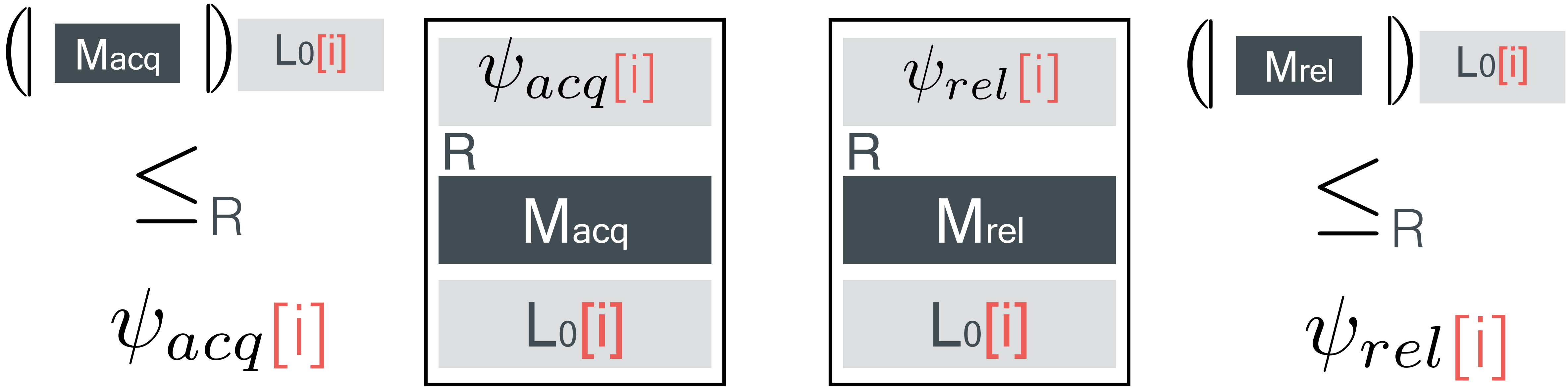
ψ_{acq}[i]

R

$\mathrm{M_{acq}}$

$\mathrm{L_0}$[i]

Certified Concurrent Abstraction Layer



$\llbracket M_{acq} \rrbracket_{L_0[i]} \leq_R \psi_{acq}[i]$

$\psi_{acq}[i]$

R

M_{acq}

$L_0[i]$

$\psi_{rel}[i]$

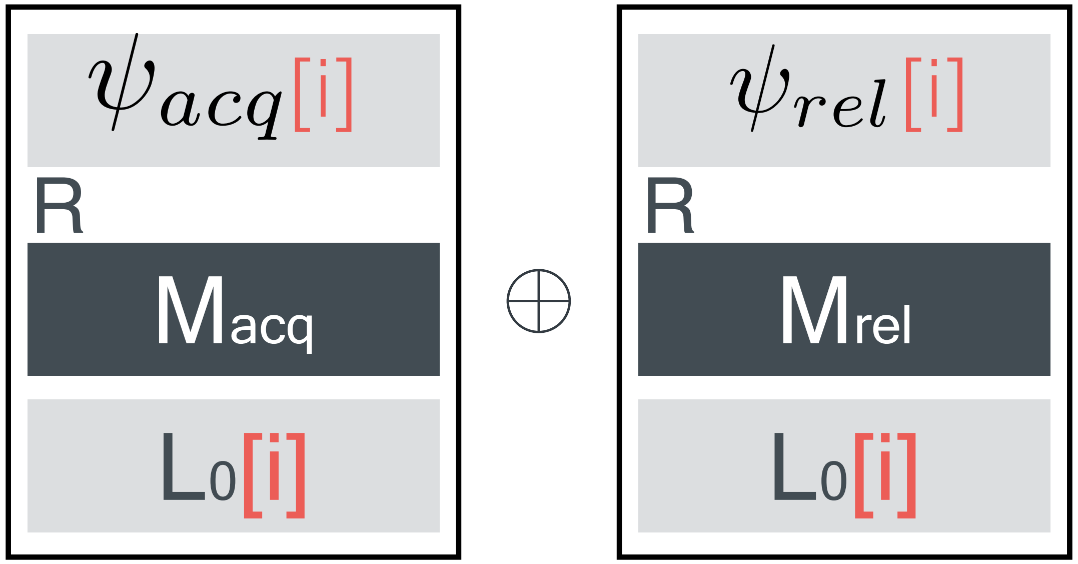
R

M_{rel}

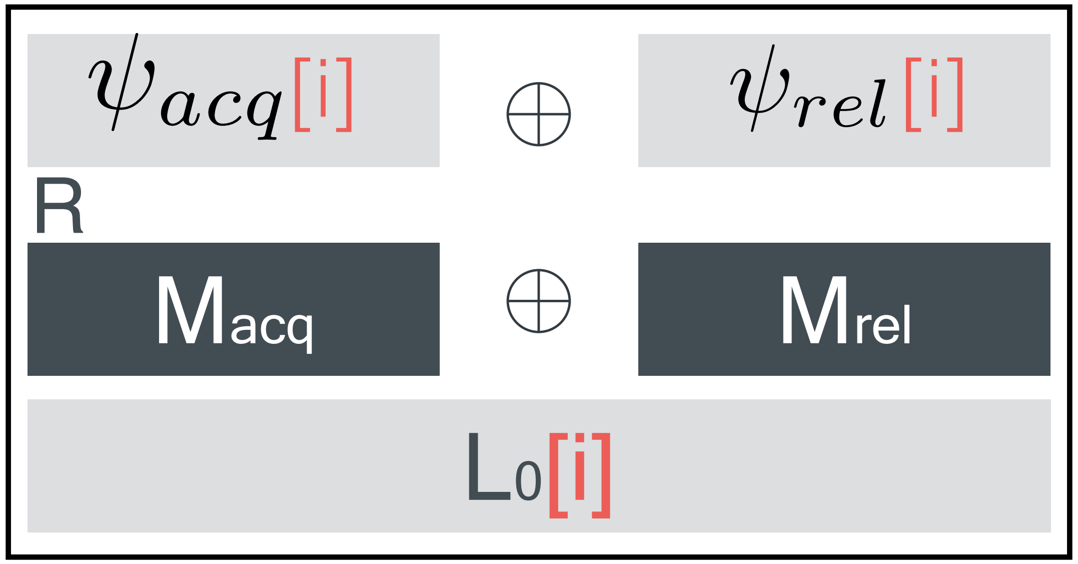
$L_0[i]$

$\llbracket M_{rel} \rrbracket_{L_0[i]} \leq_R \psi_{rel}[i]$

Horizontal Composition



Horizontal Composition



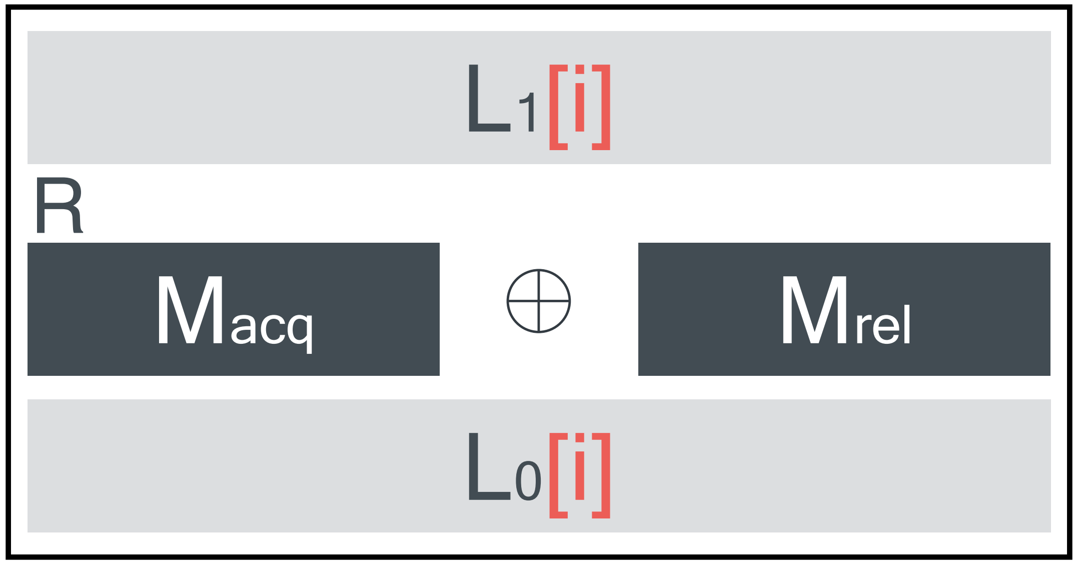
ψ_{acq}[i] $\oplus$ ψ_{rel}[i]

R

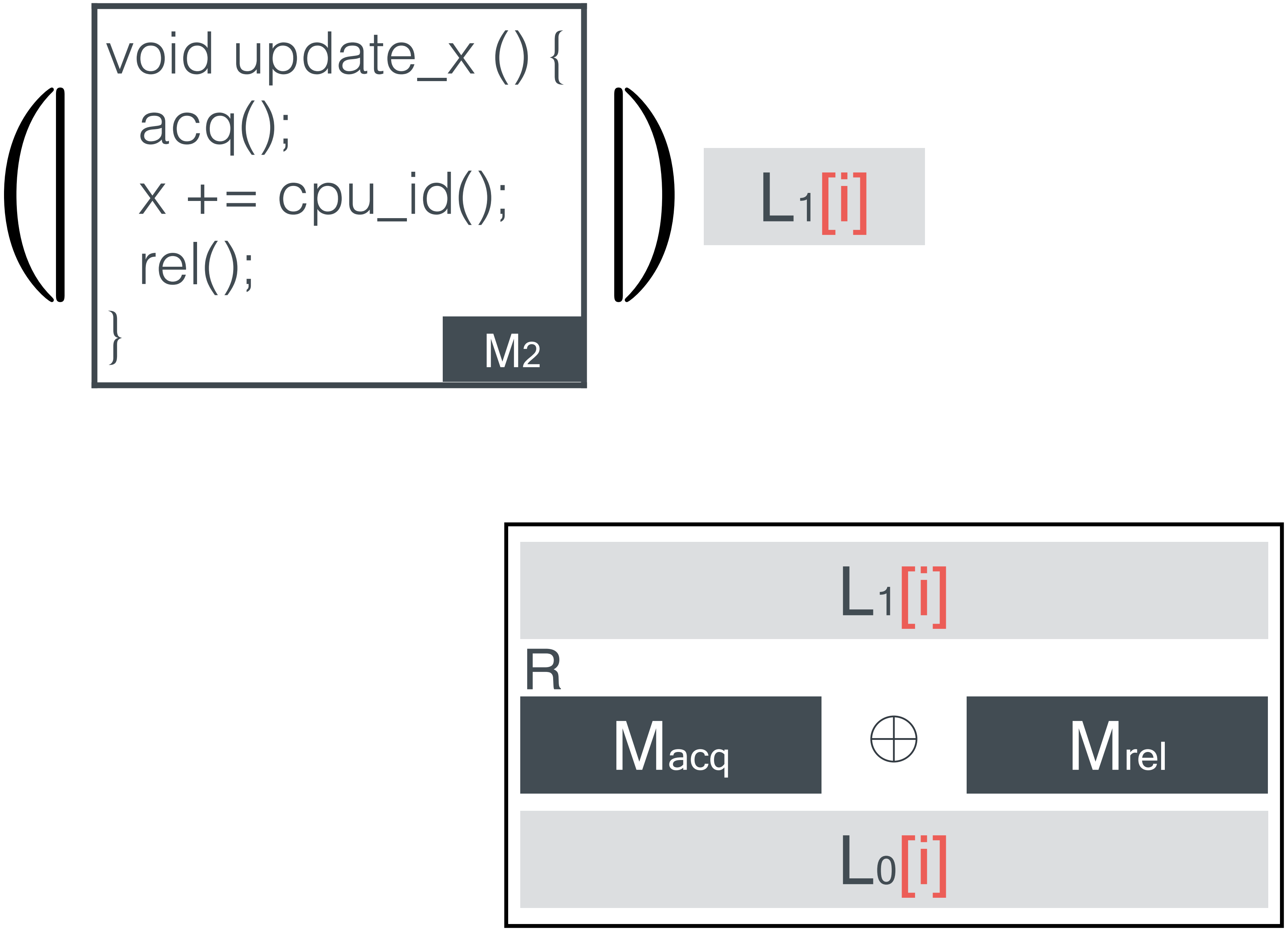
M_{acq} $\oplus$ M_{rel}

L_0[i]

Certified Concurrent Abstraction Layer

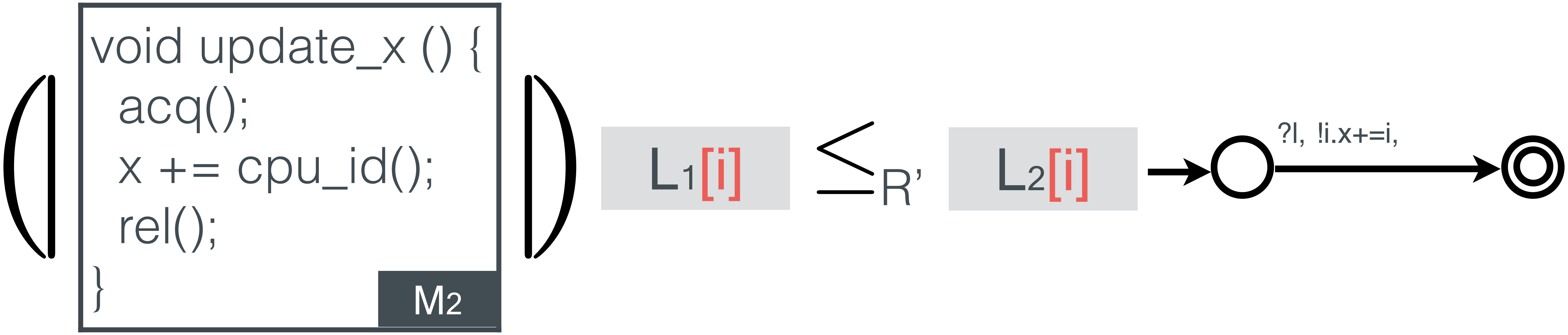


Certified Concurrent Abstraction Layer

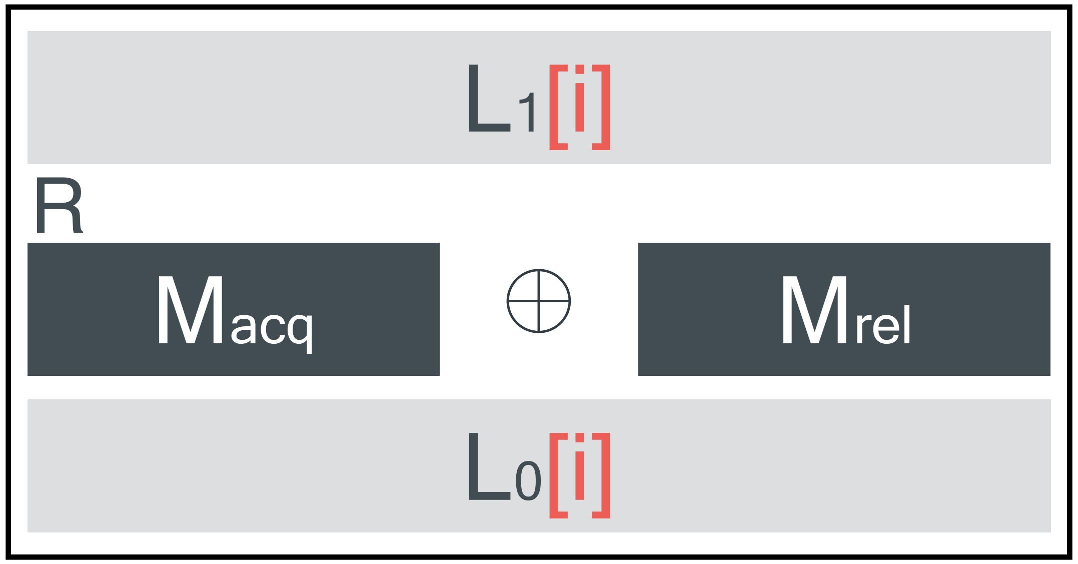


```
void update_x () {
  acq();
  x += cpu_id();
  rel();
}
```

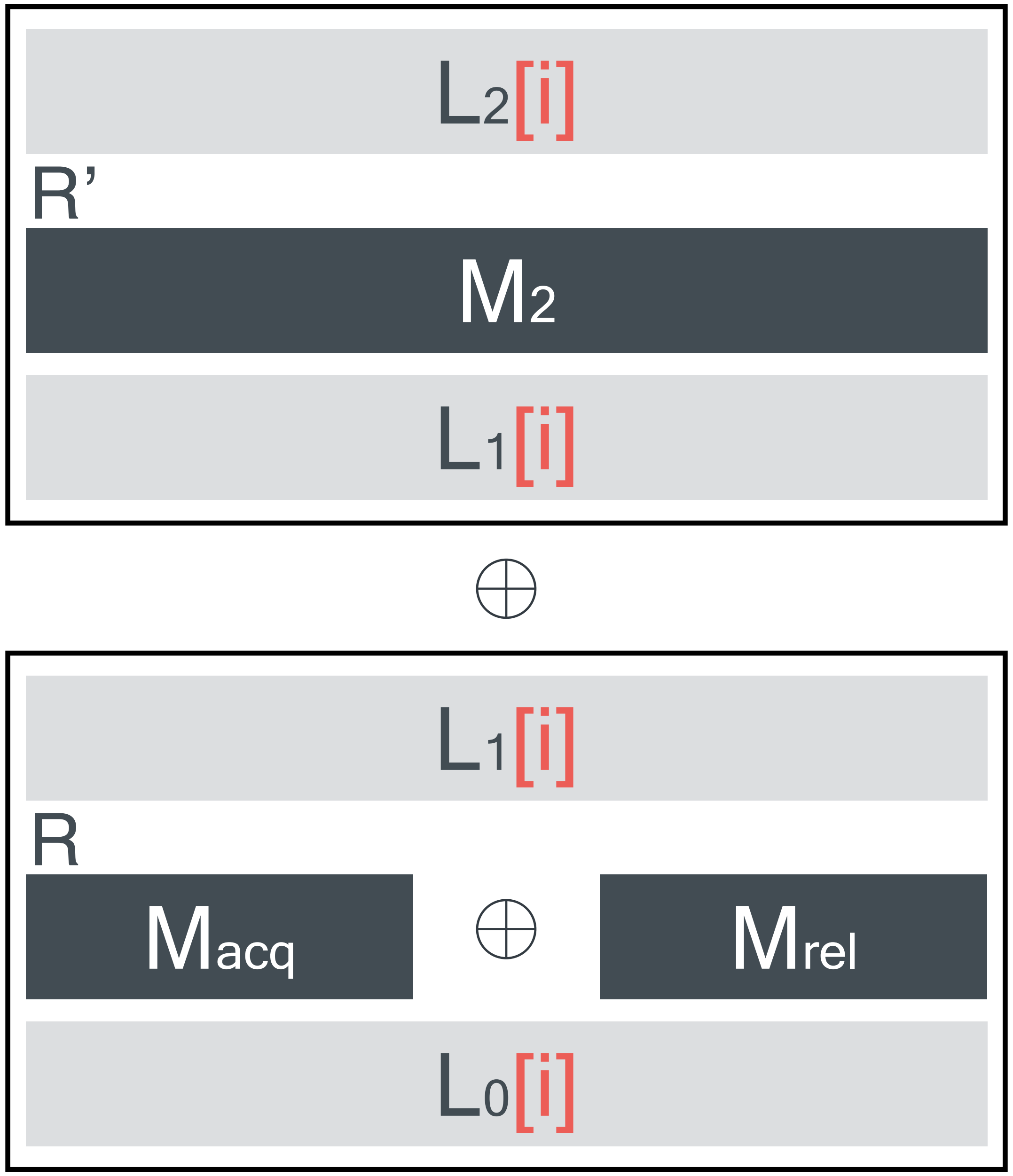
Certified Concurrent Abstraction Layer



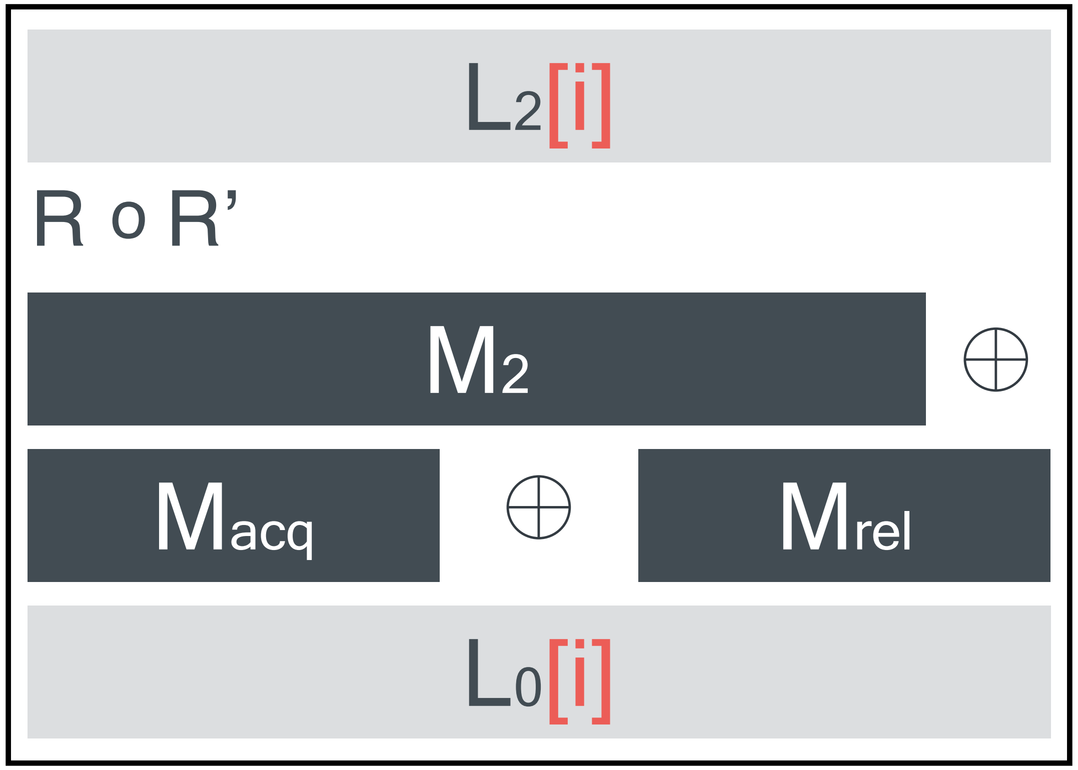
```
void update_x () {
  acq();
  x += cpu_id();
  rel();
}
```



Vertical Composition



Vertical Composition



$L_2[i]$

$R \circ R'$

M_2 $\oplus$

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

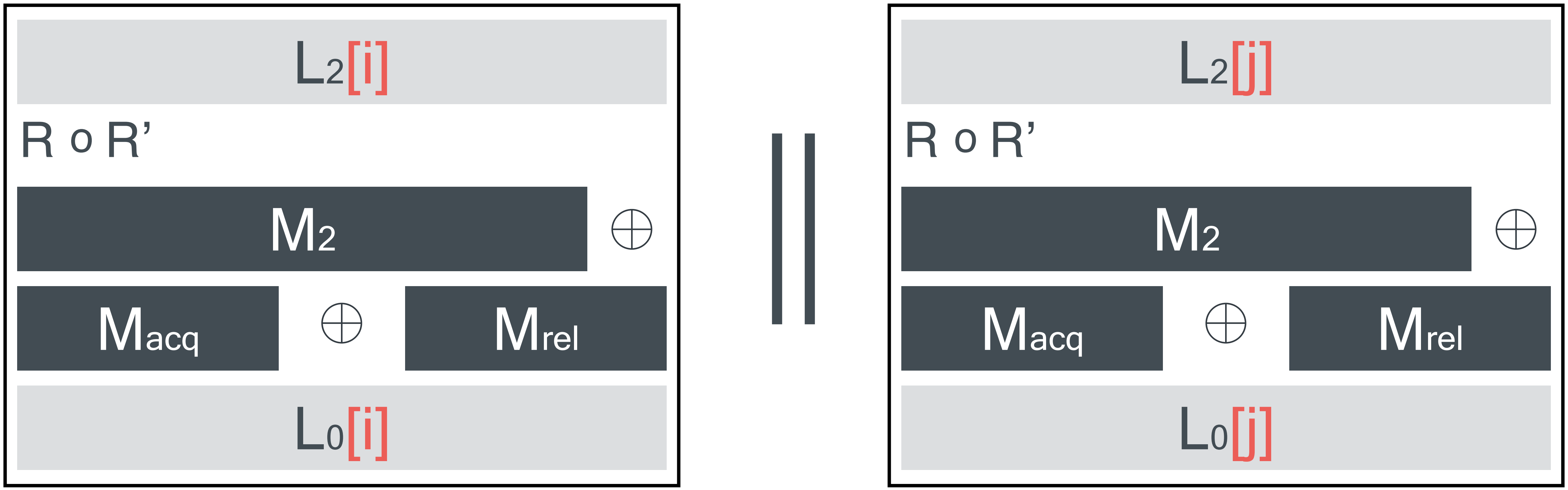
$L_0[i]$

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}



L_2[i]

R ∘ R'

M_2 ⊕

M_{acq} ⊕ M_{rel}

L_0[i]

||

L_2[j]

R ∘ R'

M_2 ⊕

M_{acq} ⊕ M_{rel}

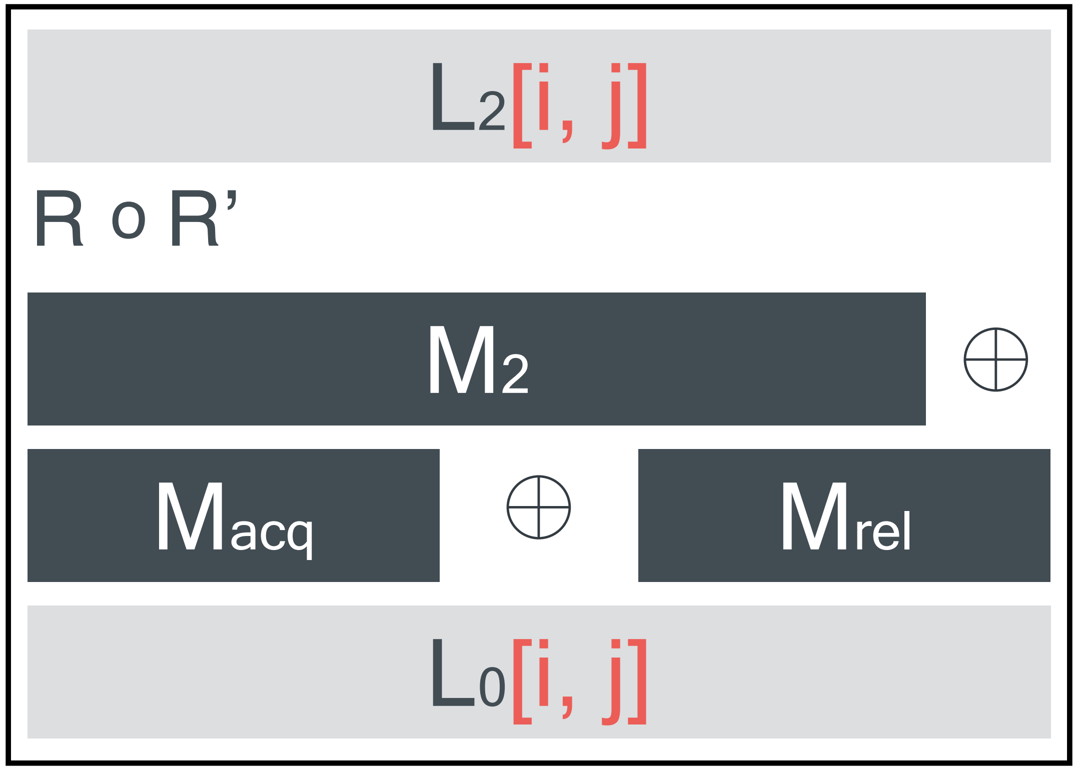
L_0[j]

Parallel Composition

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

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

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

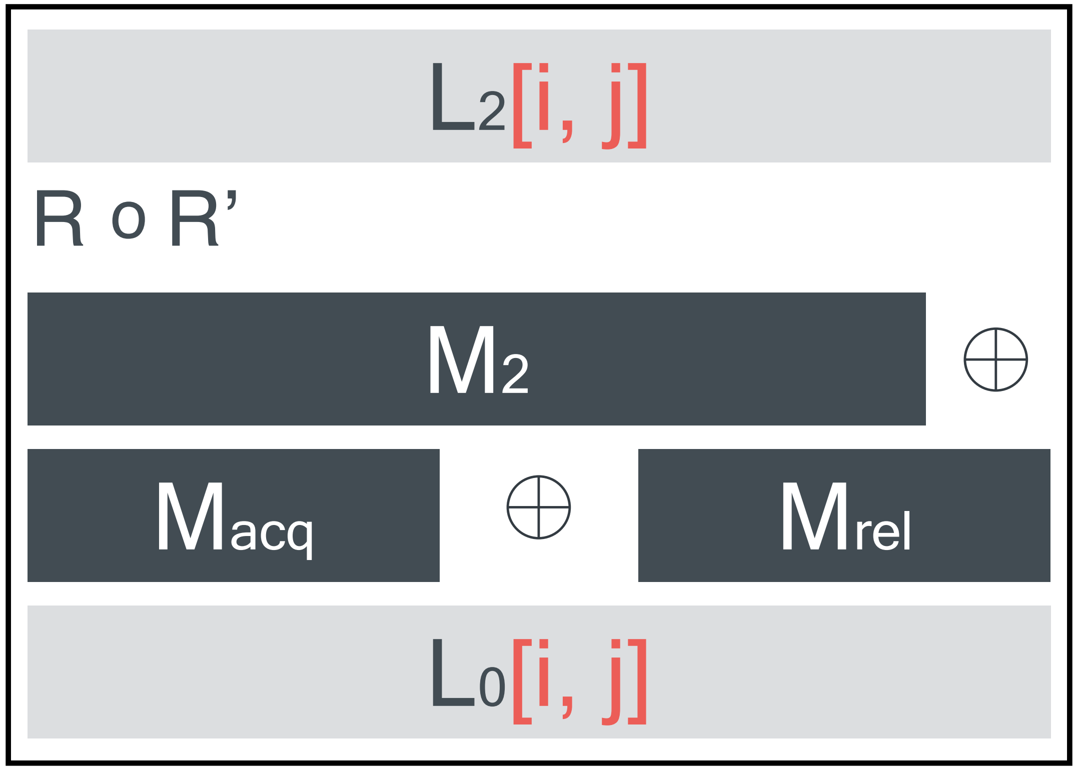


Parallel Composition

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

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

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



L_2[i, j]

R o R'

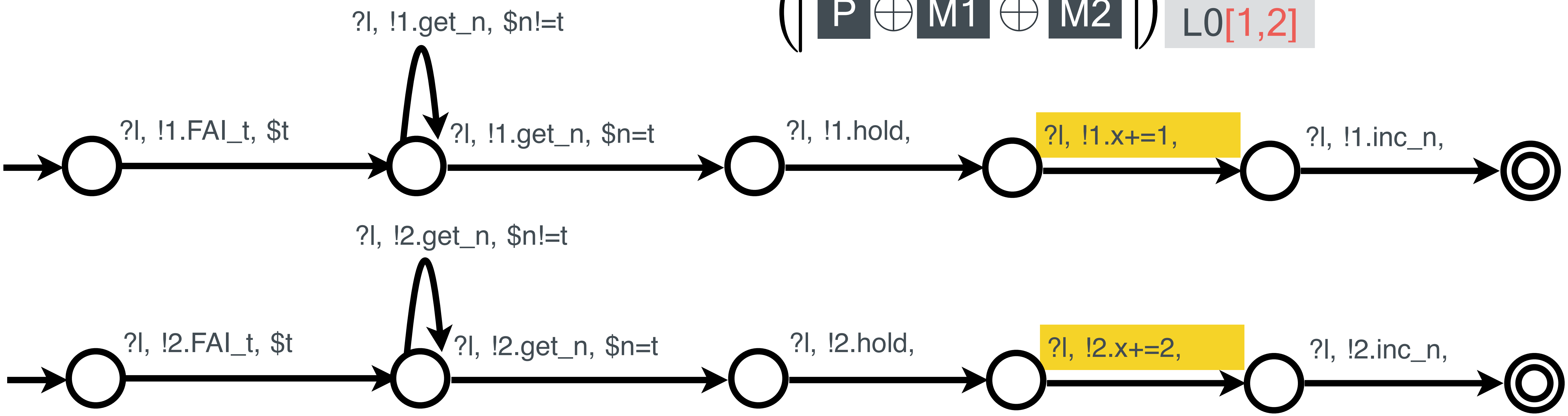
M_2 ⊕

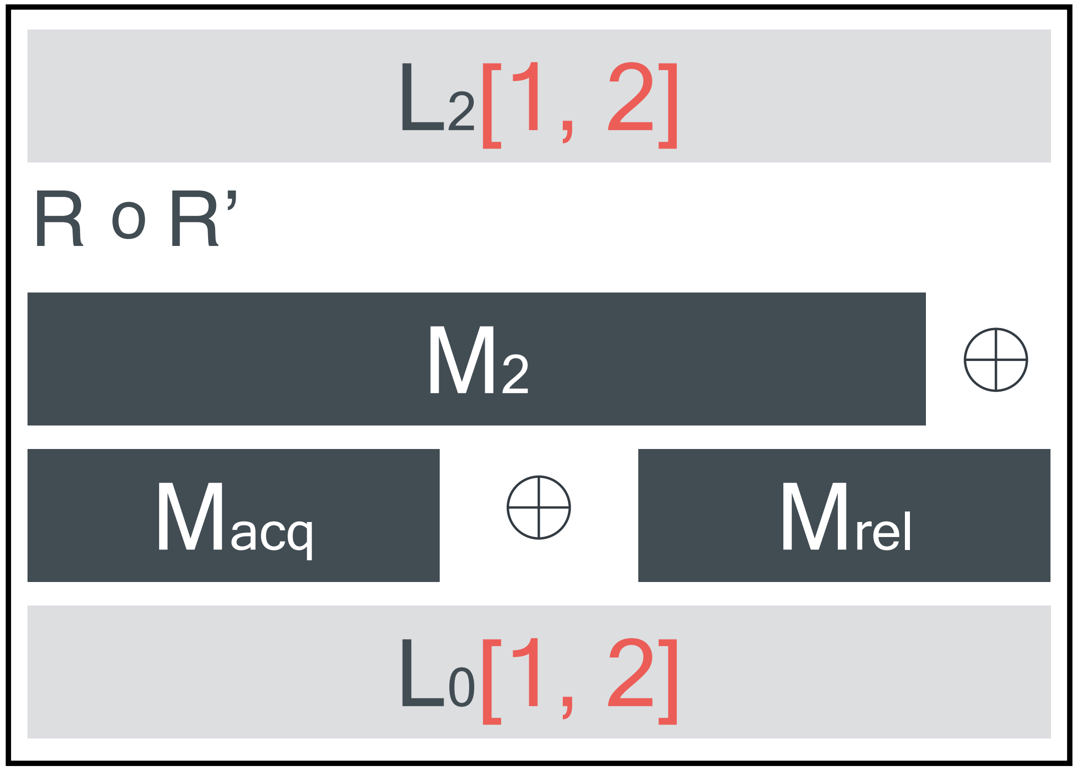
M_{acq} ⊕ M_{rel}

L_0[i, j]

Case Study

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





L2[1, 2]

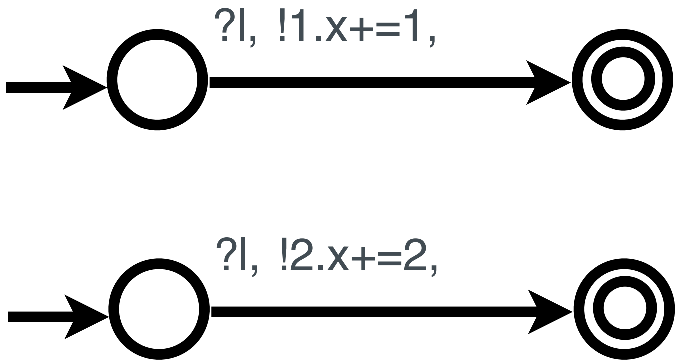
R o R'

M2 ⊕

Macq ⊕ Mrel

L0[1, 2]

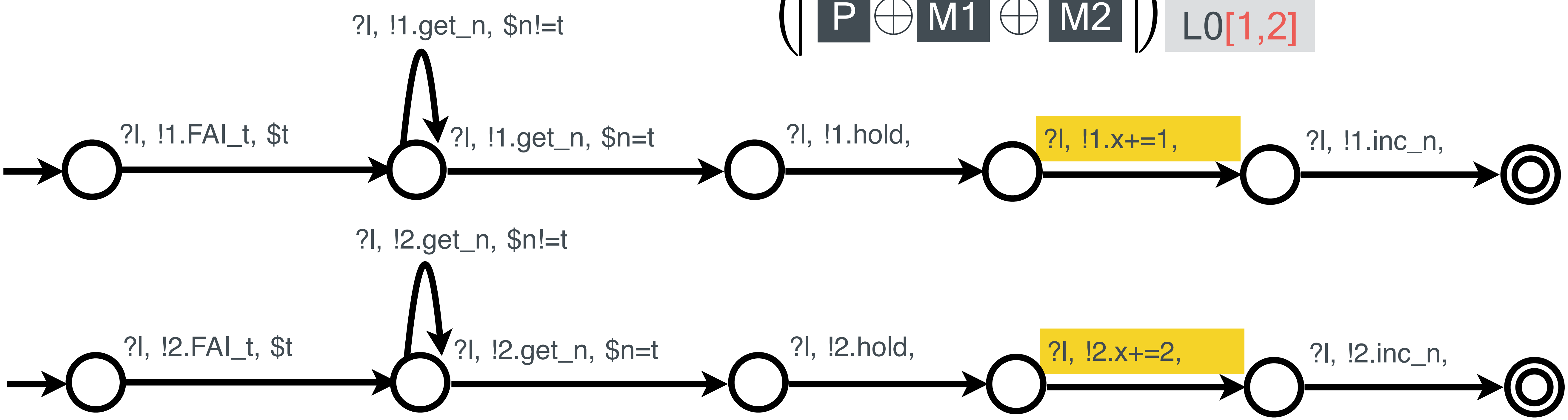
$\leq$ R o R'



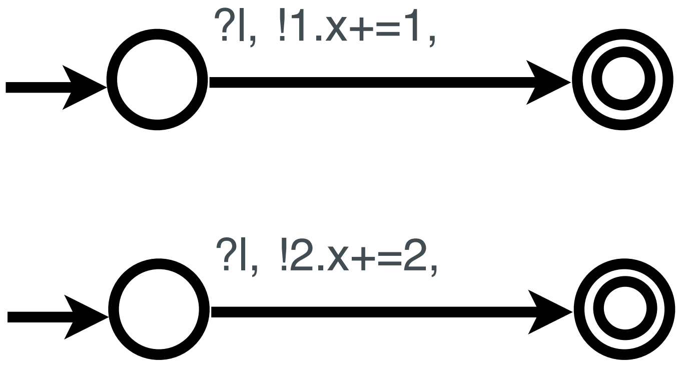
$(\!|\, P \,|\!)$ L2[1,2]

Case Study

$\llparenthesis\, \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2} \,\rrparenthesis$ L0[1,2]



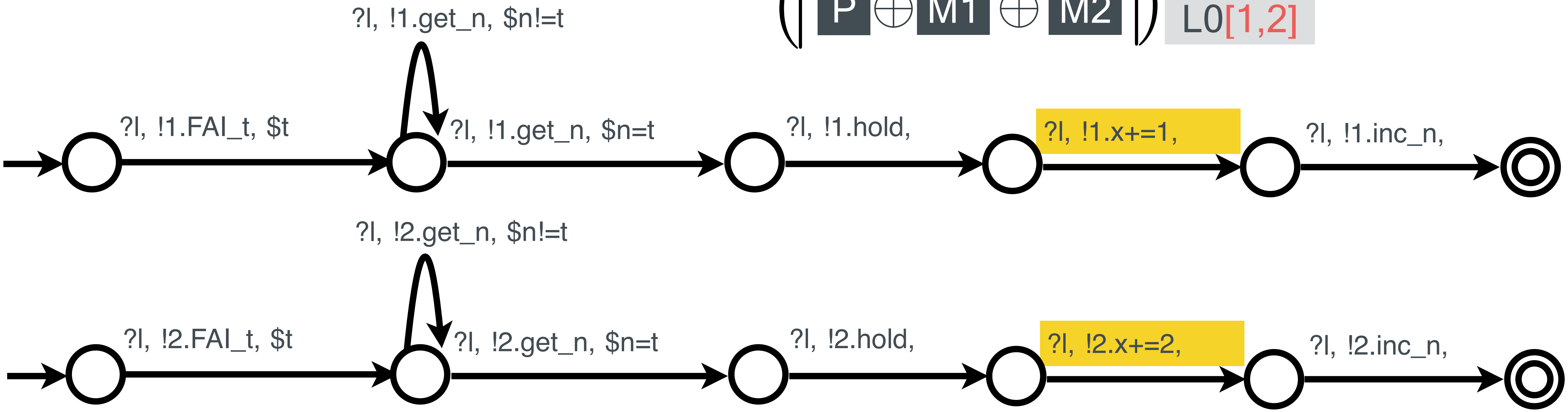
$\leq$ R o R'



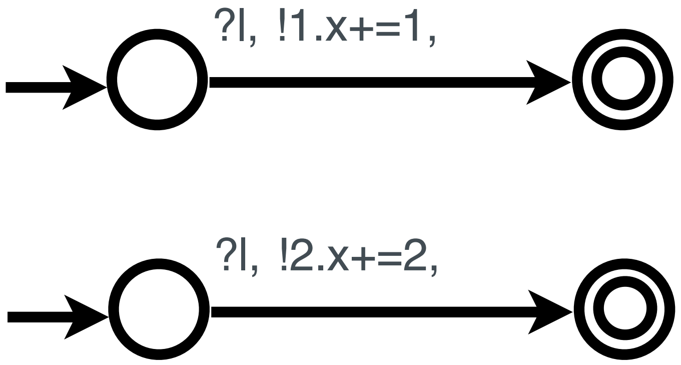
$\llparenthesis\, \boxed{P} \,\rrparenthesis$ L2[1,2]

Case Study

$\llbracket$ P $\oplus$ M1 $\oplus$ M2 $\rrbracket$ L0[1,2]



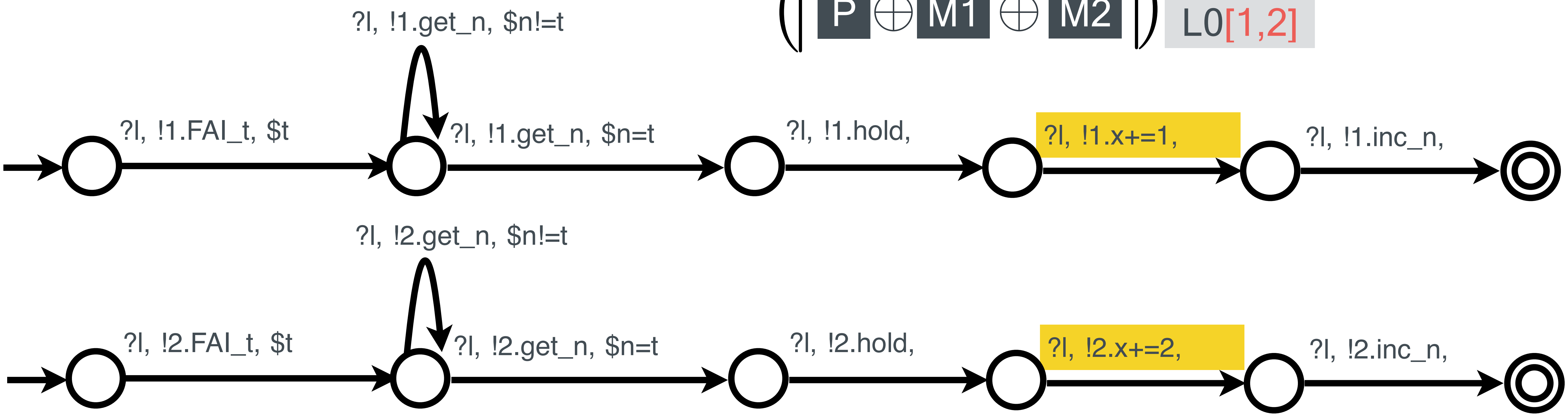
$\mathcal{E}_{hs}$ 1 2 2 1 1 2 1 2 1 2 1 2 2 2 2 2



$\llbracket$ P $\rrbracket$ L2[1,2]

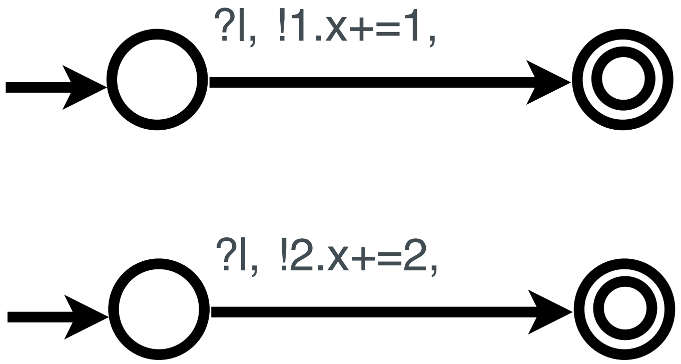
Case Study

$\left(\!\left|\, \boxed{P} \oplus \boxed{M1} \oplus \boxed{M2} \,\right|\!\right)$ L0[1,2]



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

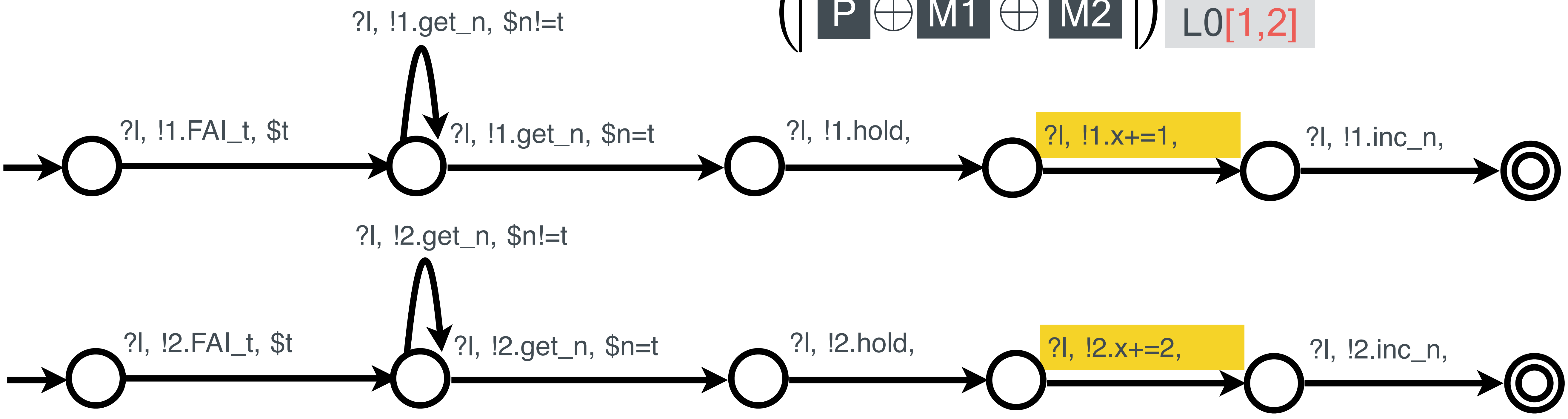
$\mathcal{E}'_{hs}$ - - 1 - 2 - - -



$\left(\!\left|\, \boxed{P} \,\right|\!\right)$ L2[1,2]

Case Study

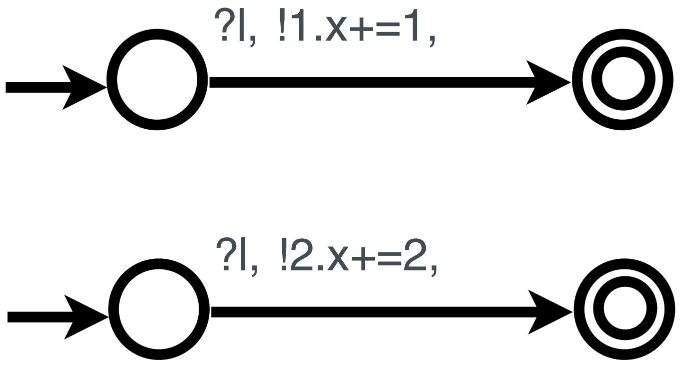
$(\!| \; P \oplus M1 \oplus M2 \; |\!)$ L0[1,2]



$\mathcal{E}_{hs}$ 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				

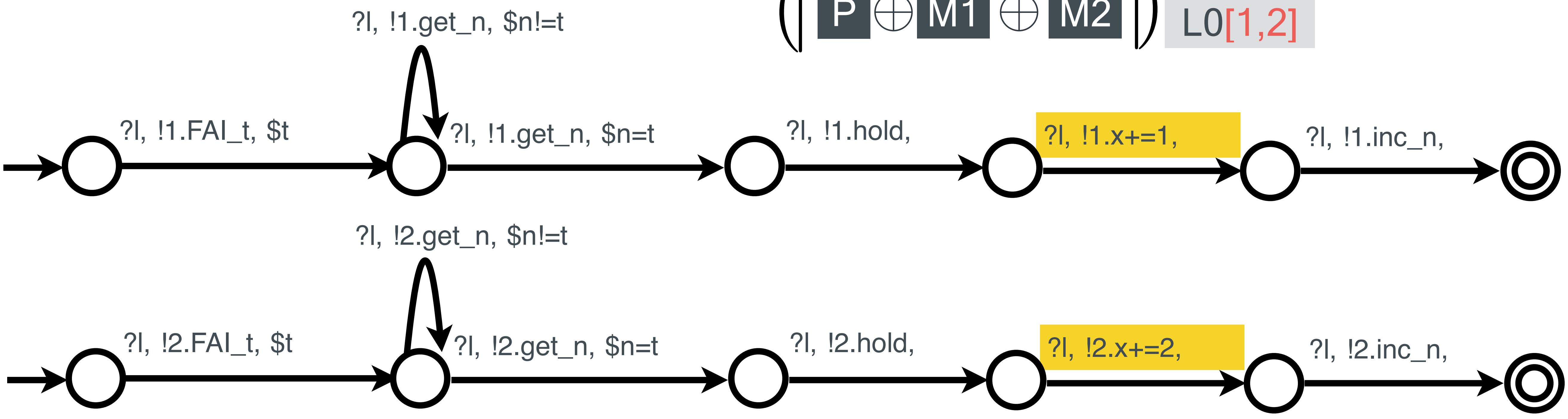
$\mathcal{E}'_{hs}$ 1 2



$(\!| \; P \; |\!)$ L2[1,2]

Case Study

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



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

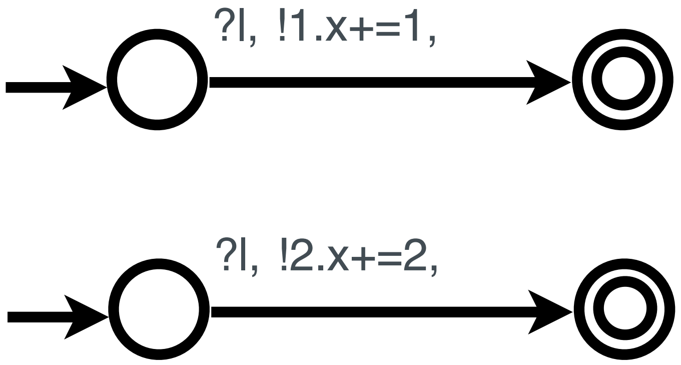
1.FAI_t | 2.FAI_t | 2.get_n | 1.get_n | 1.hold | 2.get_n | 1.r1=x | 2.get_n | 1.x=r1+1 | 2.get_n

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 ---



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

Soundness

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

Soundness

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

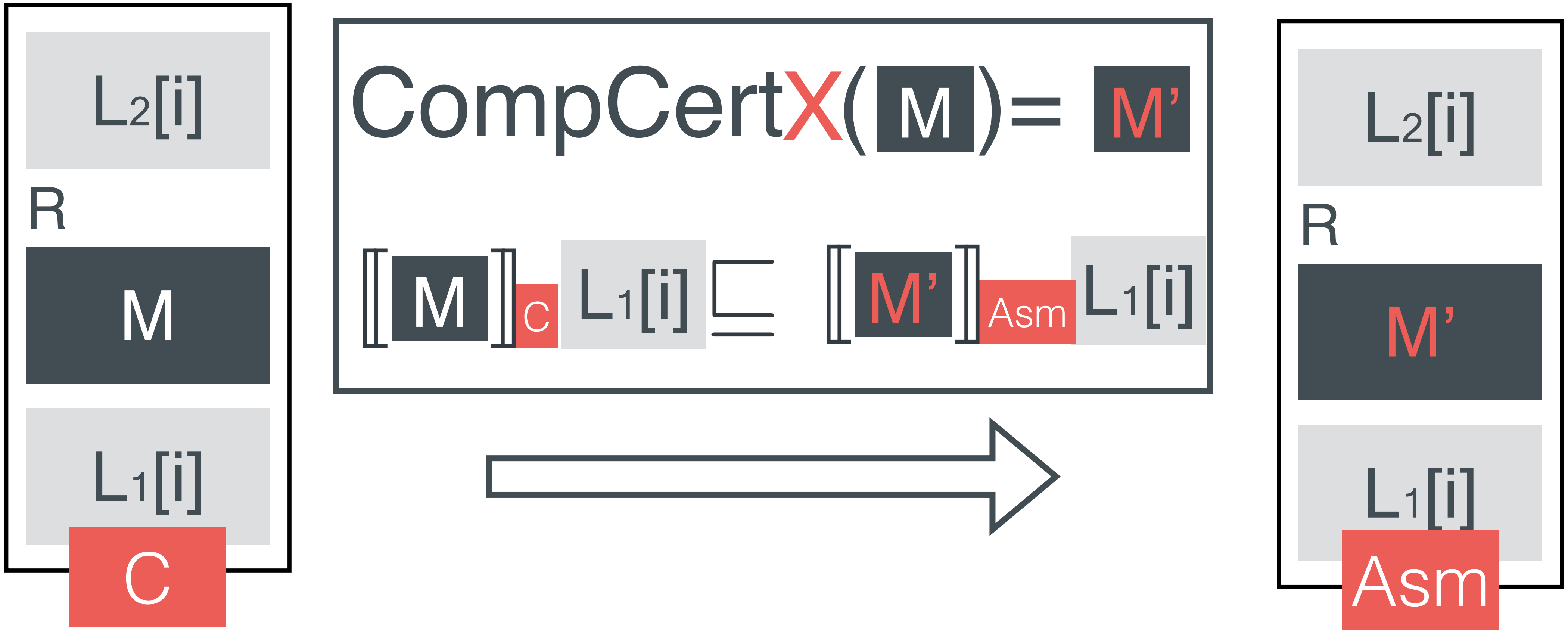
$\mathcal{R}_{cpu}$: #CPU = c < 2^{32}

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

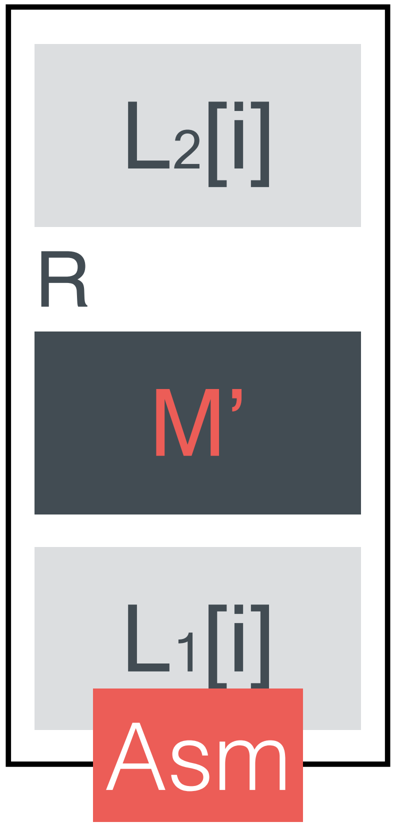
$$= \left\{ \boxed{\boxed{\text{2.x+=2}}\ \boxed{\text{1.x+=1}}} \, , \boxed{\boxed{\text{1.x+=1}}\ \boxed{\text{2.x+=2}}} \right\}$$

QED

CompCertX


$L_2[i]$
R
M
$L_1[i]$
C
CompCertX(M)= M’
$[\![M]\!]_C\ L_1[i] \sqsubseteq [\![M']\!]_{Asm}\ L_1[i]$
$L_2[i]$
R
M’
$L_1[i]$
Asm

Assembly Layers

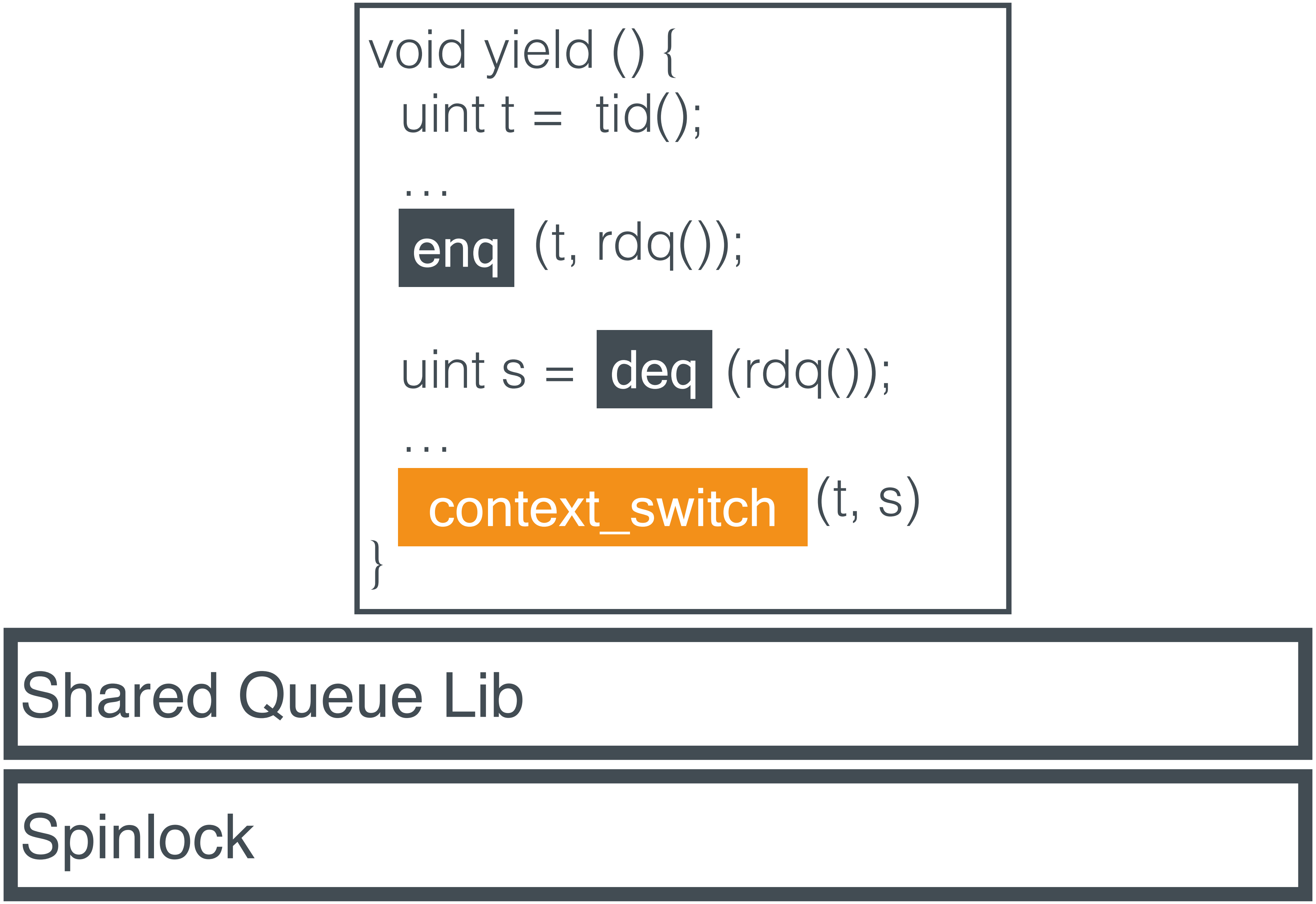


Horizontal Composition

Vertical Composition

Parellel Composition

Software Scheduler



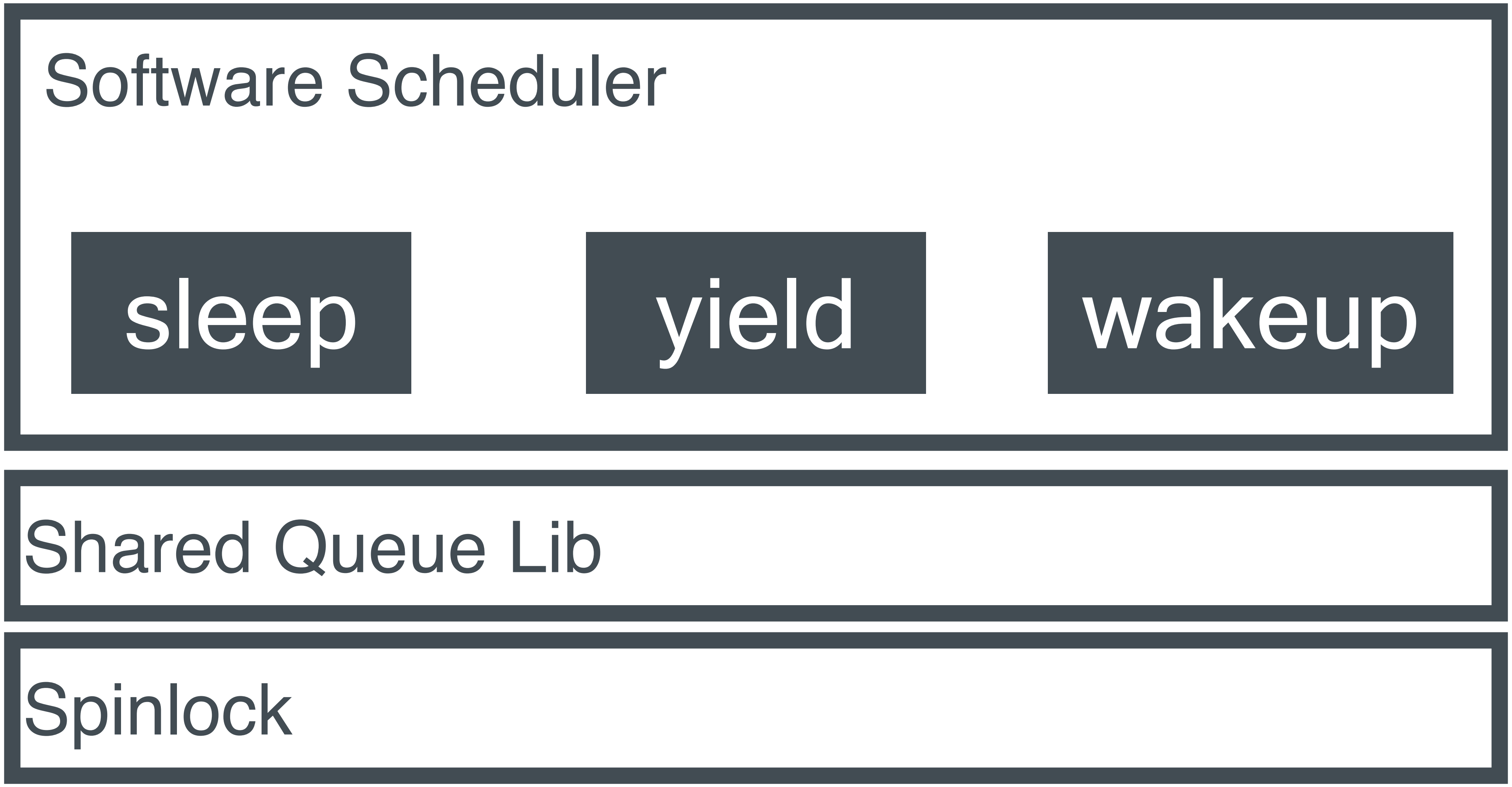
```
void yield () {
  uint t =  tid();
  . . .
  enq (t, rdq());

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

Shared Queue Lib

Spinlock

Software Scheduler



Software Scheduler

sleep

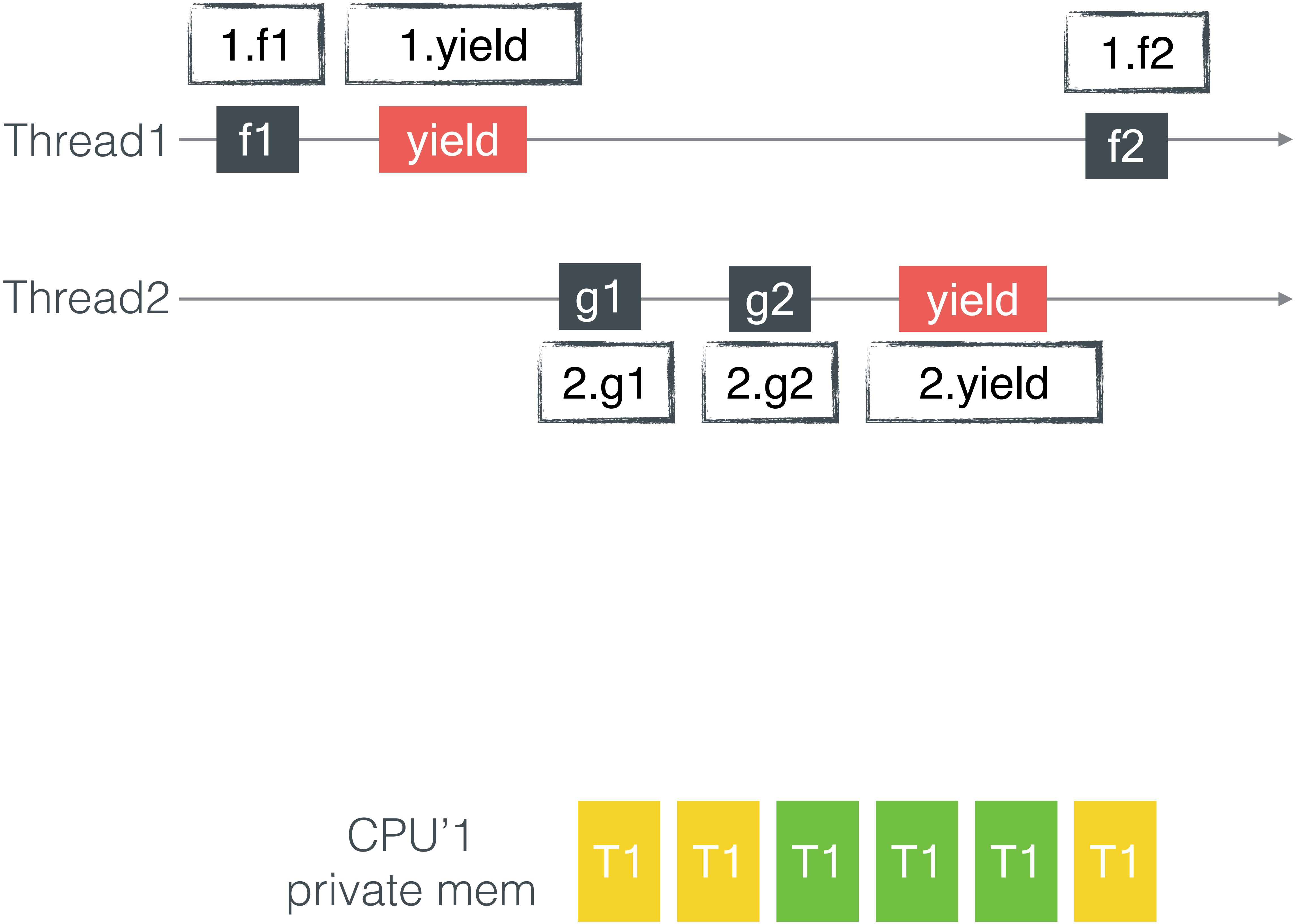
yield

wakeup

Shared Queue Lib

Spinlock

Software Scheduler



1.f1

1.yield

1.f2

Thread1

f1

yield

f2

Thread2

g1

g2

yield

2.g1

2.g2

2.yield

CPU’1
private mem

T1

T1

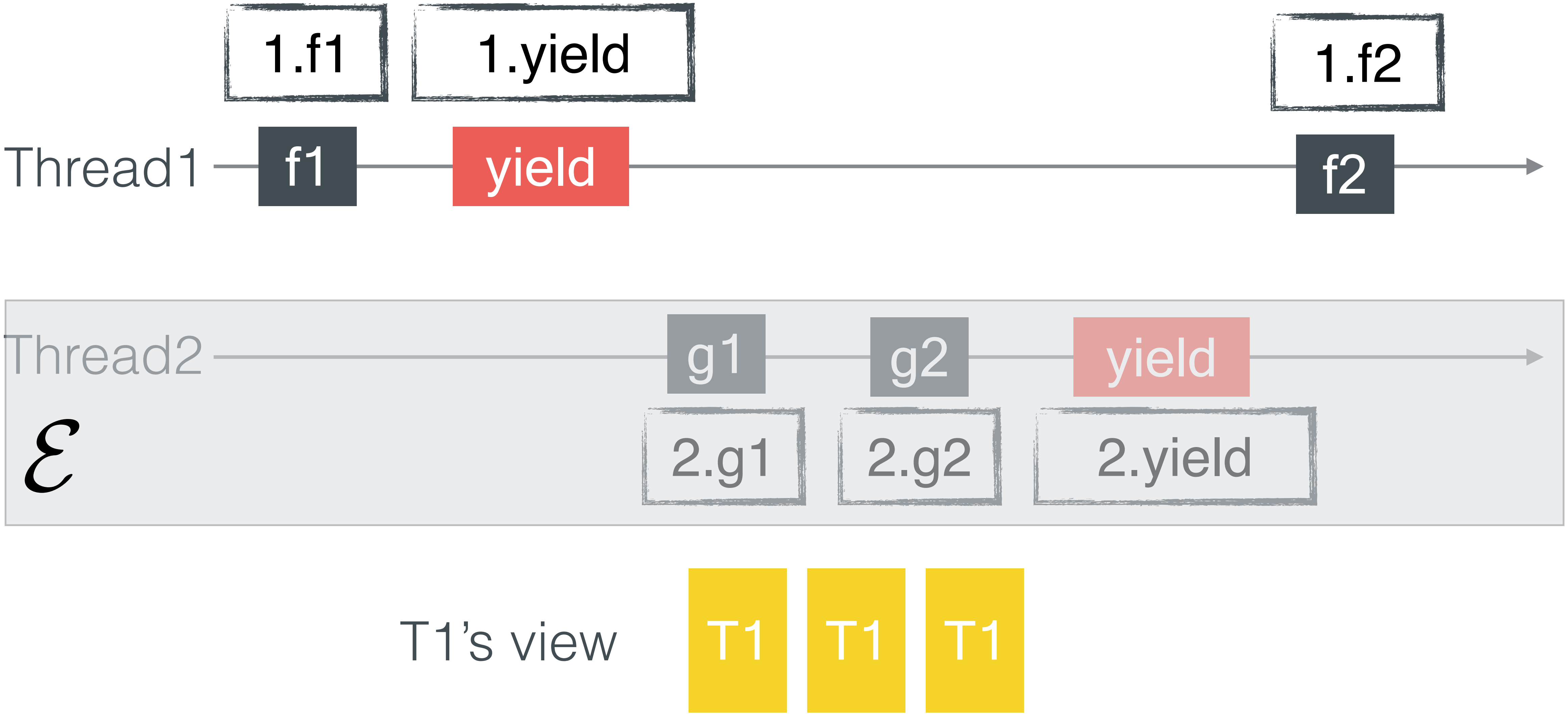
T1

T1

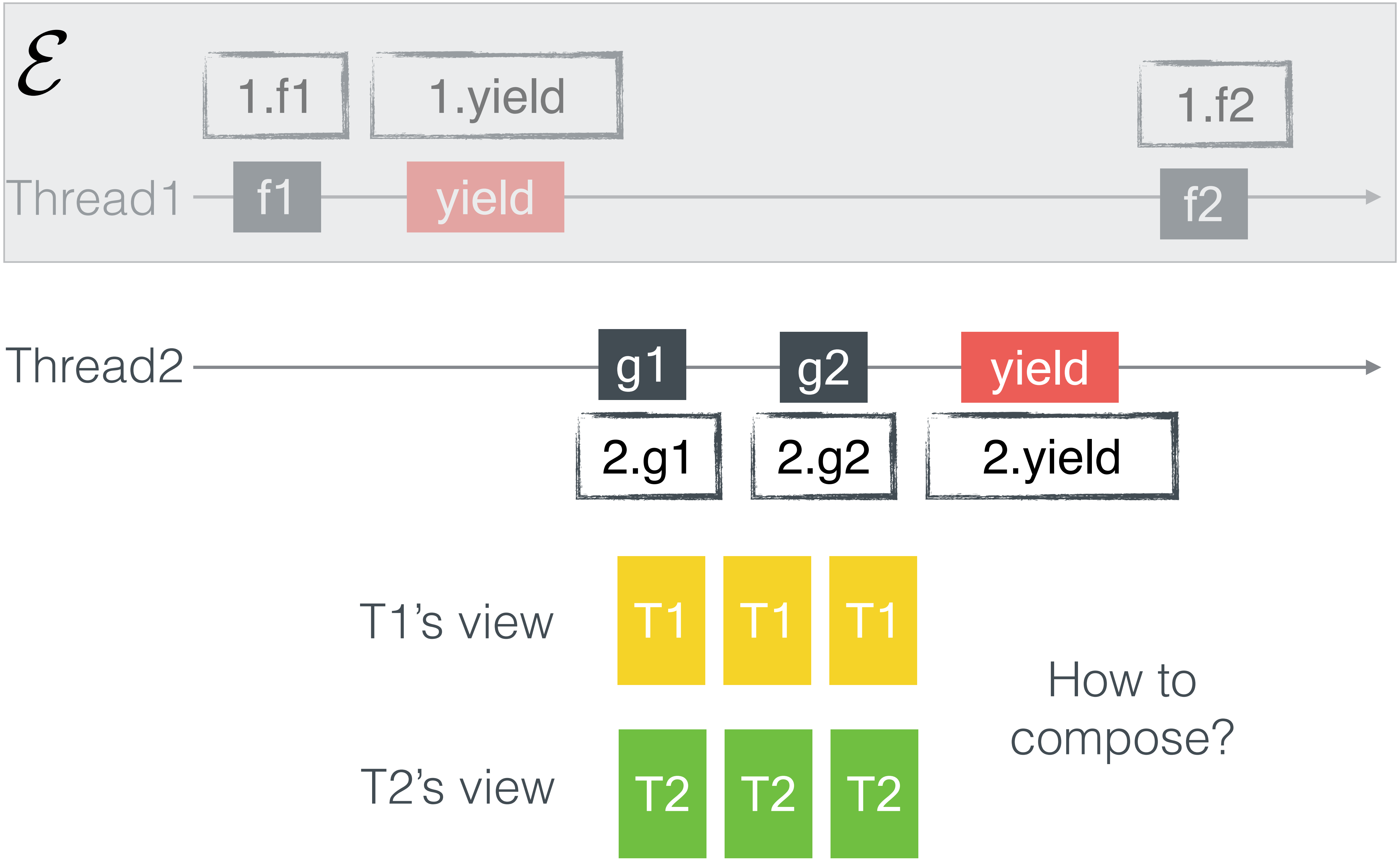
T1

T1

Software Scheduler



Software Scheduler



$\mathcal{E}$

1.f1

1.yield

1.f2

Thread1

f1

yield

f2

Thread2

g1

g2

yield

2.g1

2.g2

2.yield

T1's view

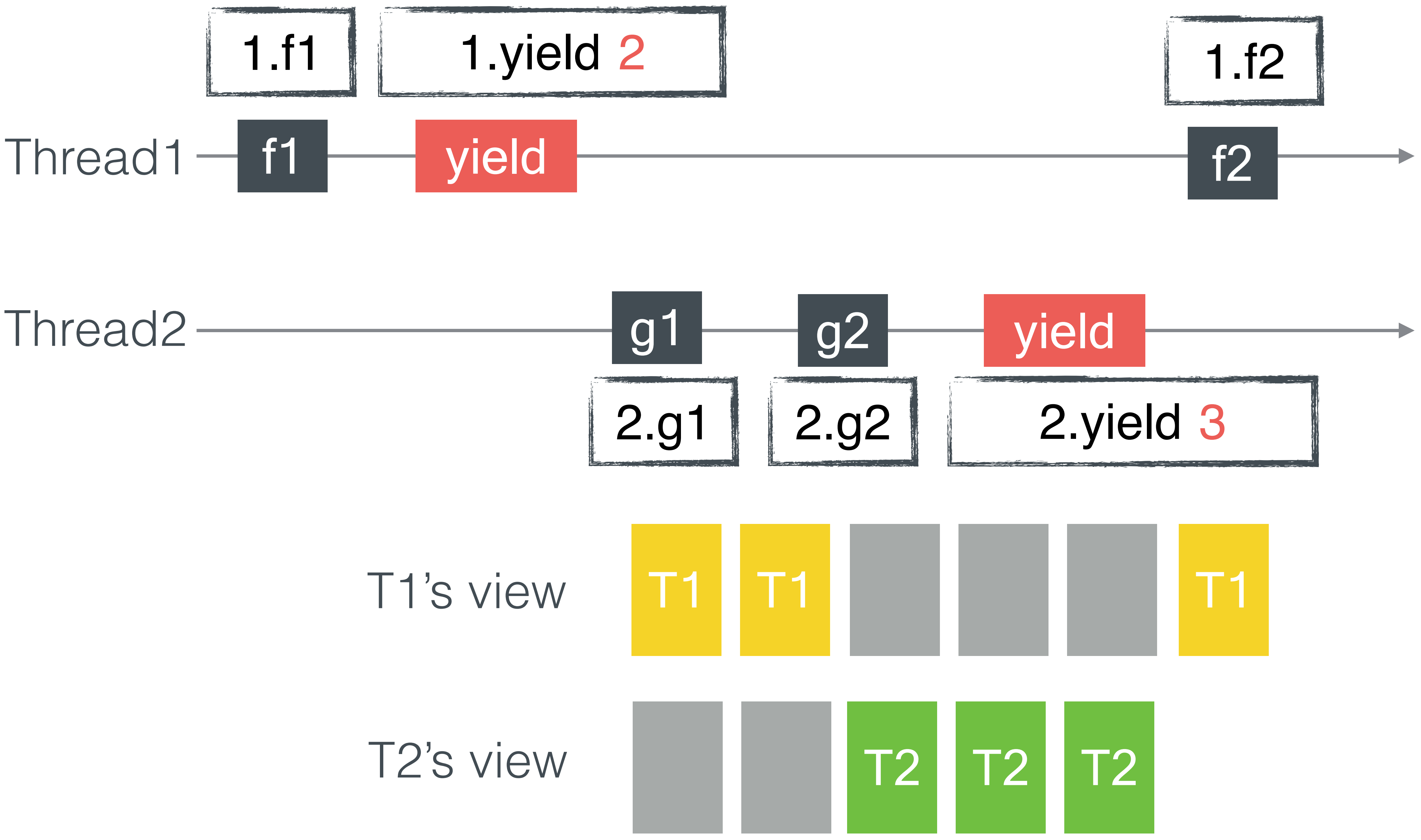
T1 T1 T1

T2's view

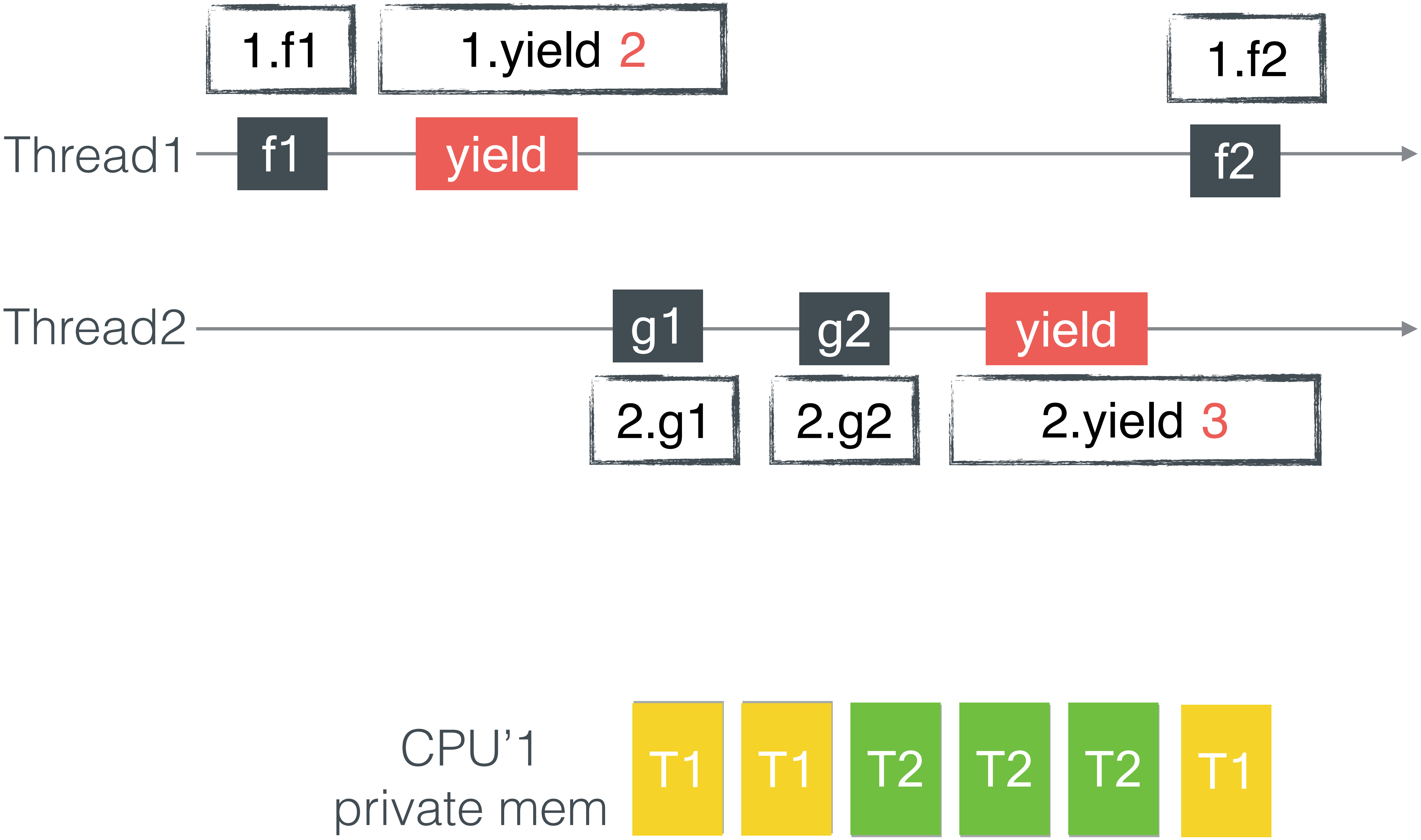
T2 T2 T2

How to compose?

Algebraic Memory Model



Algebraic Memory Model



CompCertX

CompCertX

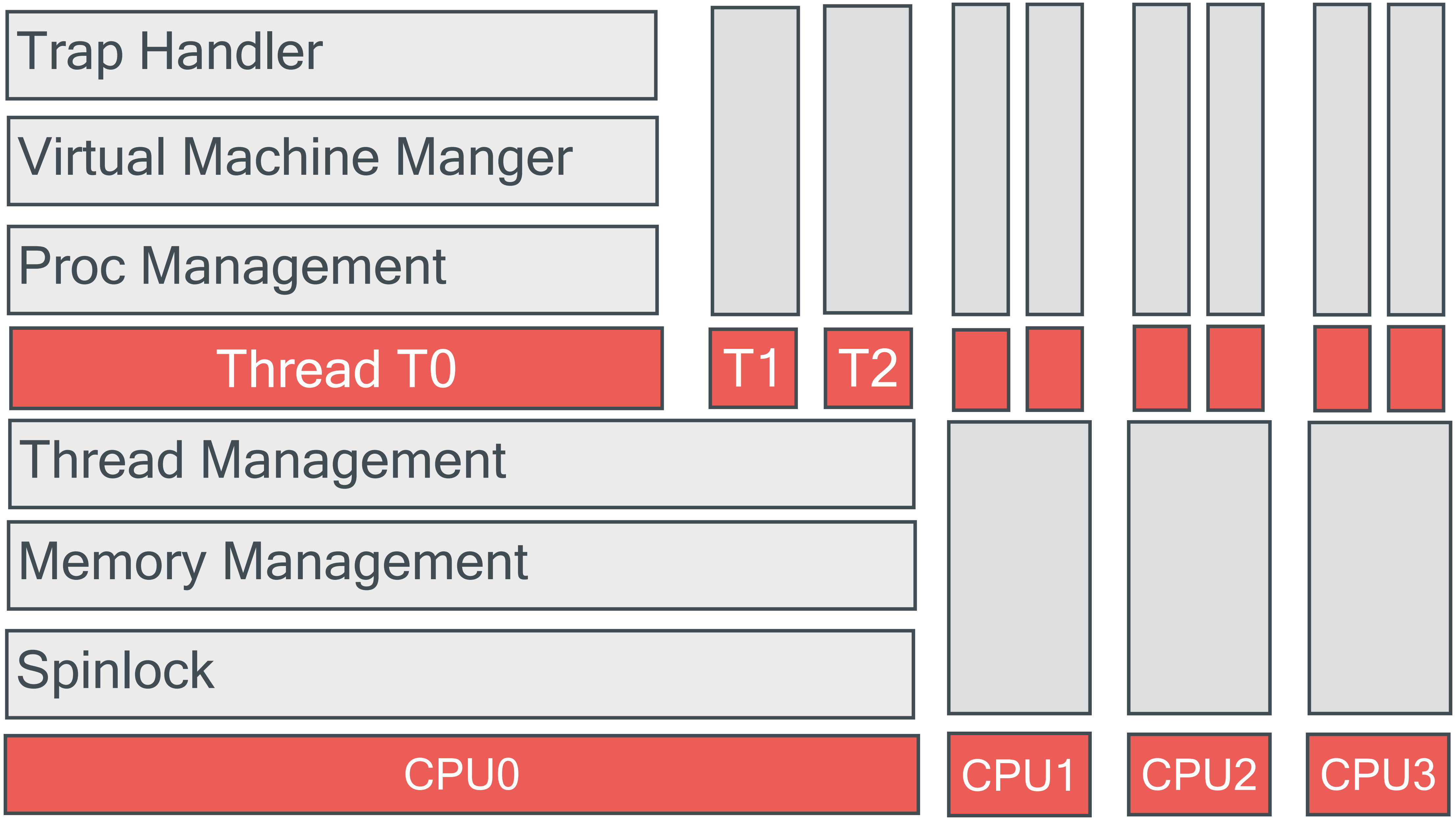
+

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

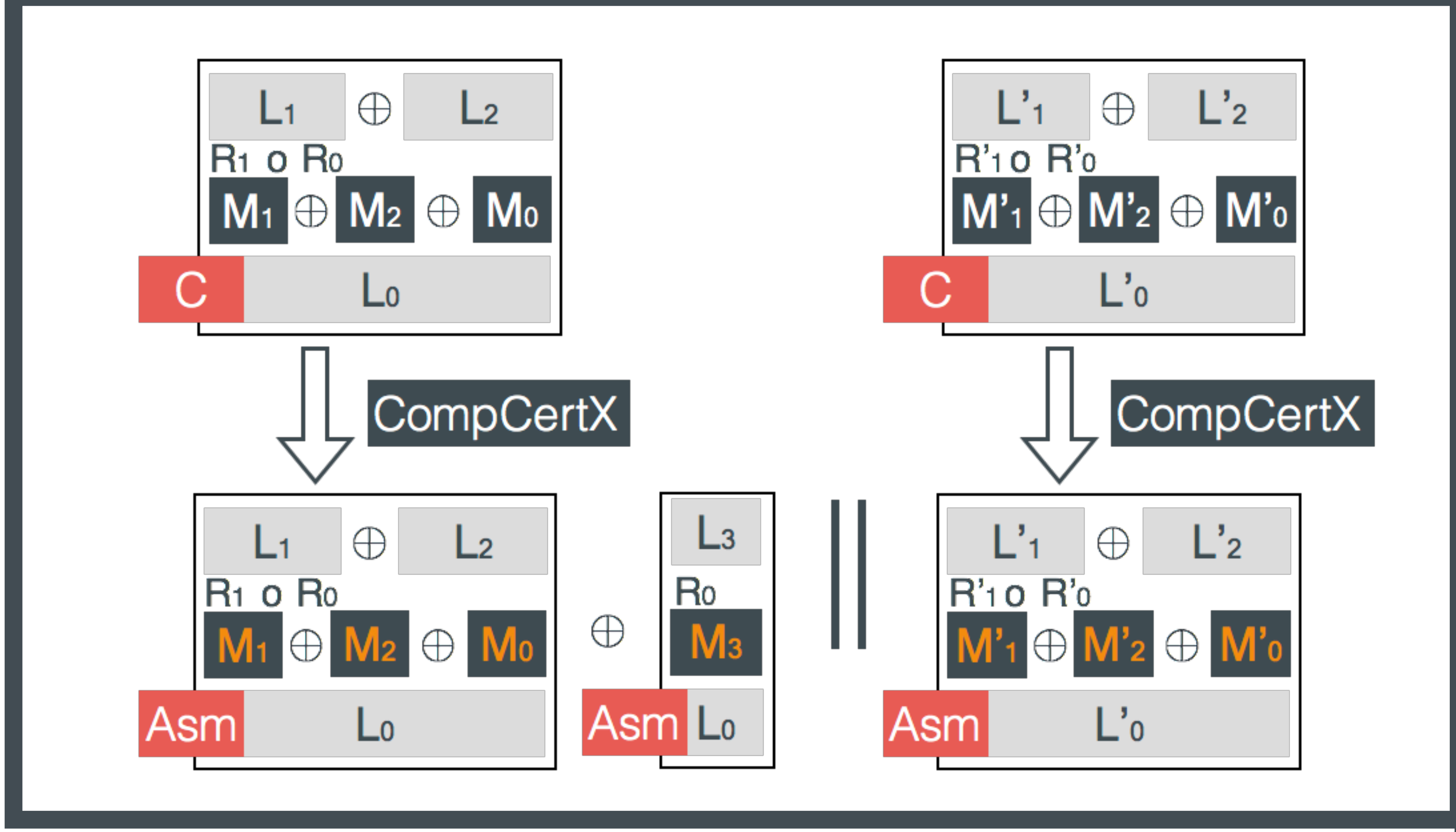
Layer	Refinement proof	Code verification	Source code	Proof linking
PHBThread	HBThreadGen	--	--	HBThreadGenLink

Layers for per-CPU

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

Contribution Summary

Certified Concurrent Abstraction Layers

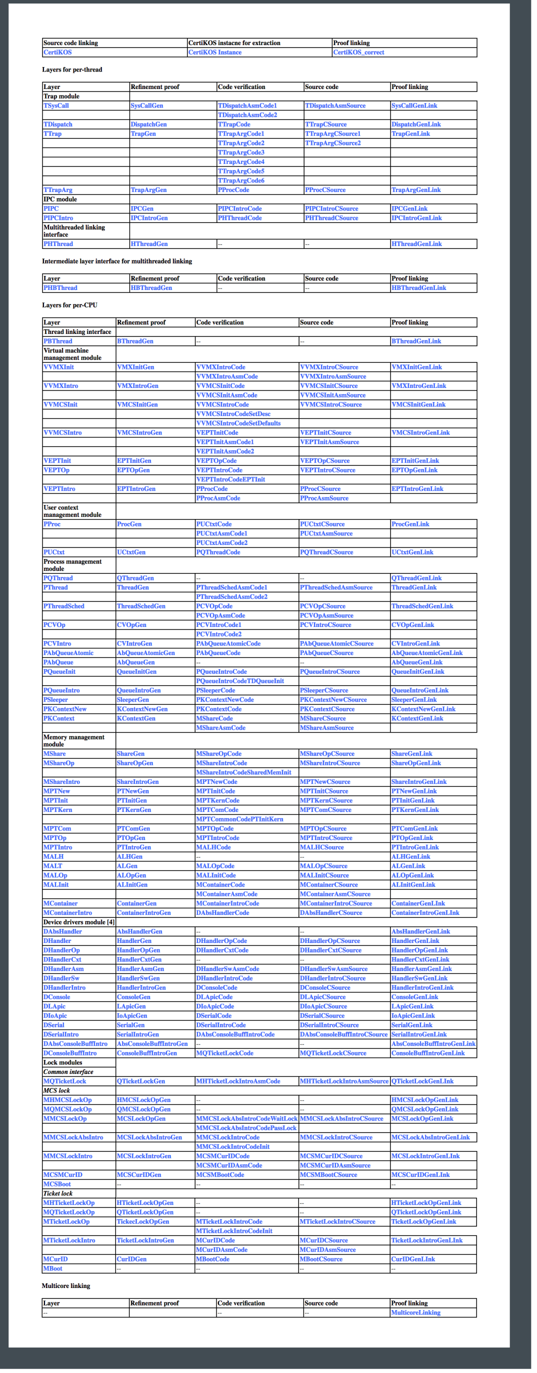


Strategy Refinement

$$\psi \leq_R \psi'$$

Thread-safe CompCert

CertiKOS



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 management module				
VVMXInit	VMXInitGen	VVMXIntroCode	VVMXIntroCSource	VMXInitGenLink
		VVMXIntroAsmCode	VVMXIntroAsmSource	
VVMXIntro	VMXIntroGen	VVMCSInitCode	VVMCSInitCSource	VMXIntroGenLink
		VVMCSInitAsmCode	VVMCSInitAsmSource	
VVMCSInit	VMCSInitGen	VVMCSIntroCode	VVMCSIntroCSource	VMCSInitGenLink
		VVMCSIntroCodeSetDesc		
		VVMCSIntroCodeSetDefaults		
VVMCSIntro	VMCSIntroGen	VEPTInitCode	VEPTInitCSource	VMCSIntroGenLink
		VEPTInitAsmCode1	VEPTInitAsmSource	
		VEPTInitAsmCode2		
VEPTInit	EPTInitGen	VEPTOpCode	VEPTOpCSource	EPTInitGenLink
VEPTOp	EPTOpGen	VEPTIntroCode	VEPTIntroCSource	EPTOpGenLink
		VEPTIntroCodeEPTInit		
VEPTIntro	EPTIntroGen	PProcCode	PProcCSource	EPTIntroGenLink
		PProcAsmCode	PProcAsmSource	
User context management module				
PProc	ProcGen	PUCtxtCode	PUCtxtCSource	ProcGenLink
		PUCtxtAsmCode1	PUCtxtAsmSource	
		PUCtxtAsmCode2		
PUCtxt	UCtxtGen	PQThreadCode	PQThreadCSource	UCtxtGenLink
Process management module				
PQThread	QThreadGen	--	--	QThreadGenLink
PThread	ThreadGen	PThreadSchedAsmCode1	PThreadSchedAsmSource	ThreadGenLink
		PThreadSchedAsmCode2		
PThreadSched	ThreadSchedGen	PCVOpCode	PCVOpCSource	ThreadSchedGenLink
		PCVOpAsmCode	PCVOpAsmSource	
PCVOp	CVOpGen	PCVIntroCode1	PCVIntroCSource	CVOpGenLink
		PCVIntroCode2		
PCVIntro	CVIntroGen	PAbQueueAtomicCode	PAbQueueAtomicCSource	CVIntroGenLink
PAbQueueAtomic	AbQueueAtomicGen	PAbQueueCode	PAbQueueCSource	AbQueueAtomicGenLink
PAbQueue	AbQueueGen	--	--	AbQueueGenLink
PQueueInit	QueueInitGen	PQueueIntroCode	PQueueIntroCSource	QueueInitGenLink
		PQueueIntroCodeTDQueueInit		
PQueueIntro	QueueIntroGen	PSleeperCode	PSleeperCSource	QueueIntroGenLink
PSleeper	SleeperGen	PKContextNewCode	PKContextNewCSource	SleeperGenLink
PKContextNew	KContextNewGen	PKContextCode	PKContextCSource	KContextNewGenLink
PKContext	KContextGen	MShareCode	MShareCSource	KContextGenLink
		MShareAsmCode	MShareAsmSource	
Memory management module				
MShare	ShareGen	MShareOpCode	MShareOpCSource	ShareGenLink
MShareOp	ShareOpGen	MShareIntroCode	MShareIntroCSource	ShareOpGenLink
		MShareIntroCodeSharedMemInit		
MShareIntro	ShareIntroGen	MPTNewCode	MPTNewCSource	ShareIntroGenLink
MPTNew	PTNewGen	MPTInitCode	MPTInitCSource	PTNewGenLink
MPTInit	PTInitGen	MPTKernCode	MPTKernCSource	PTInitGenLink
MPTKern	PTKernGen	MPTComCode	MPTComCSource	PTKernGenLink
		MPTCommonCodePTInitKern		
MPTCom	PTComGen	MPTOpCode	MPTOpCSource	PTComGenLink
MPTOp	PTOpGen	MPTIntroCode	MPTIntroCSource	PTOpGenLink
MPTIntro	PTIntroGen	MALHCode	MALHCSource	PTIntroGenLink
MALH	ALHGen	--	--	ALHGenLink
MALT	ALGen	MALOpCode	MALOpCSource	ALGenLink
MALOp	ALOpGen	MALInitCode	MALInitCSource	ALOpGenLink
MALInit	ALInitGen	MContainerCode	MContainerCSource	ALInitGenLink
		MContainerAsmCode	MContainerAsmCSource	
MContainer	ContainerGen	MContainerIntroCode	MContainerIntroCSource	ContainerGenLInk
MContainerIntro	ContainerIntroGen	DAbsHandlerCode	DAbsHandlerCSource	ContainerIntroGenLInk
Device drivers module [4]				
DAbsHandler	AbsHandlerGen	--	--	AbsHandlerGenLink
DHandler	HandlerGen	DHandlerOpCode	DHandlerOpCSource	HandlerGenLink
DHandlerOp	HandlerOpGen	DHandlerCxtCode	DHandlerCxtCSource	HandlerOpGenLink
DHandlerCxt	HandlerCxtGen	--	--	HandlerCxtGenLink
DHandlerAsm	HandlerAsmGen	DHandlerSwAsmCode	DHandlerSwAsmSource	HandlerAsmGenLink
DHandlerSw	HandlerSwGen	DHandlerIntroCode	DHandlerIntroCSource	HandlerSwGenLink
DHandlerIntro	HandlerIntroGen	DConsoleCode	DConsoleCSource	HandlerIntroGenLink
DConsole	ConsoleGen	DLApicCode	DLApicCSource	ConsoleGenLink
DLApic	LApicGen	DIoApicCode	DIoApicCSource	LApicGenLink
DIoApic	IoApicGen	DSerialCode	DSerialCSource	IoApicGenLink
DSerial	SerialGen	DSerialIntroCode	DSerialIntroCSource	SerialGenLink
DSerialIntro	SerialIntroGen	DAbsConsoleBuffIntroCode	DAbsConsoleBuffIntroCSource	SerialIntroGenLink
DAbsConsoleBuffIntro	AbsConsoleBuffIntroGen	--	--	AbsConsoleBuffIntroGenLink
DConsoleBuffIntro	ConsoleBuffIntroGen	MQTicketLockCode	MQTicketLockCSource	ConsoleBuffIntroGenLink
Lock modules				
Common interface				
MQTicketLock	QTicketLockGen	MHTicketLockIntroAsmCode	MHTicketLockIntroAsmSource	QTicketLockGenLInk
MCS lock				
MHMCSLockOp	HMCSLockOpGen	--	--	HMCSLockOpGenLink
MQMCSLockOp	QMCSLockOpGen	--	--	QMCSLockOpGenLink
MMCSLockOp	MCSLockOpGen	MMCSLockAbsIntroCodeWaitLock	MMCSLockAbsIntroCSource	MCSLockOpGenLink
		MMCSLockAbsIntroCodePassLock		
MMCSLockAbsIntro	MCSLockAbsIntroGen	MMCSLockIntroCode	MMCSLockIntroCSource	MCSLockAbsIntroGenLink
		MMCSLockIntroCodeInit		
MMCSLockIntro	MCSLockIntroGen	MCSMCurIDCode	MCSMCurIDCSource	MCSLockIntroGenLInk
		MCSMCurIDAsmCode	MCSMCurIDAsmSource	
MCSMCurID	MCSCurIDGen	MCSMBootCode	MCSMBootCSource	MCSCurIDGenLInk
MCSBoot	--	--	--	--
Ticket lock				
MHTicketLockOp	HTicketLockOpGen	--	--	HTicketLockOpGenLink
MQTicketLockOp	QTicketLockOpGen	--	--	QTicketLockOpGenLink
MTicketLockOp	TickecLockOpGen	MTicketLockIntroCode	MTicketLockIntroCSource	TicketLockOpGenLink
		MTicketLockIntroCodeInit		
MTicketLockIntro	TicketLockIntroGen	MCurIDCode	MCurIDCSource	TicketLockIntroGenLInk
		MCurIDAsmCode	MCurIDAsmSource	
MCurID	CurIDGen	MBootCode	MBootCSource	CurIDGenLInk
MBoot	--	--	--	--

Multicore linking

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