Expressivity of Non-atomic Asychronous Networks

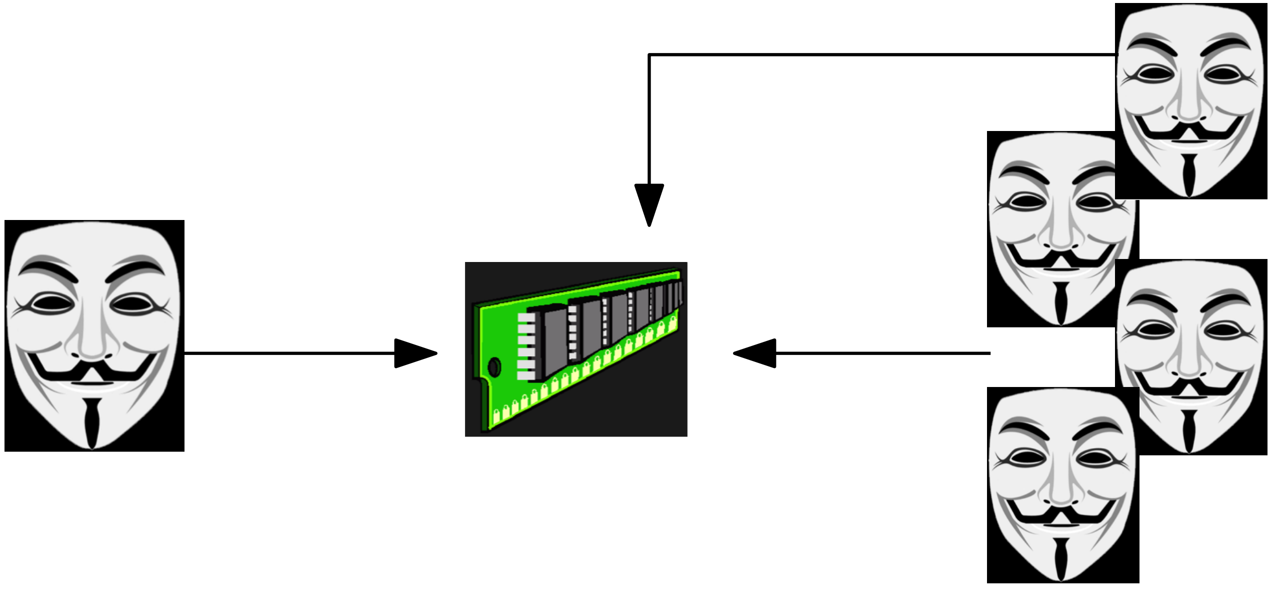
Pierre Ganty IMDEA Software Institute, Madrid

with Antoine Durand-Gasselin, Javier Esparza and Rupak Majumdar

processes have no identity

processes have no identity

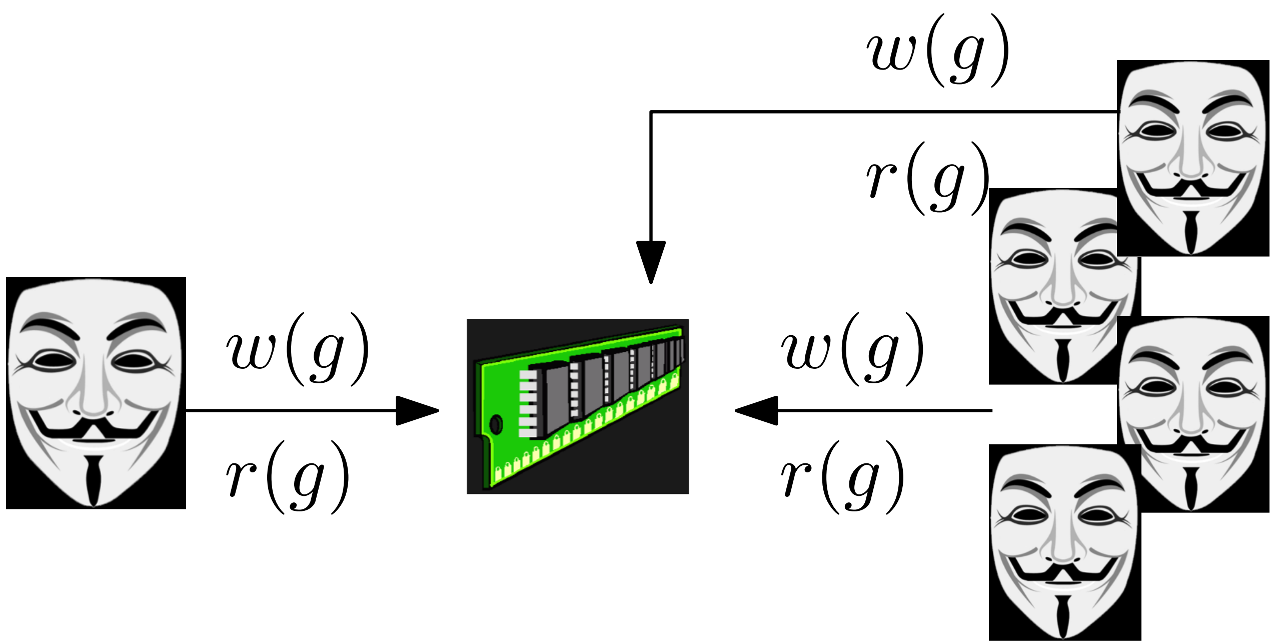
communication through a shared bounded-value register



processes have no identity

communication through a shared bounded-value register

register can be read from, written to but not **locked**

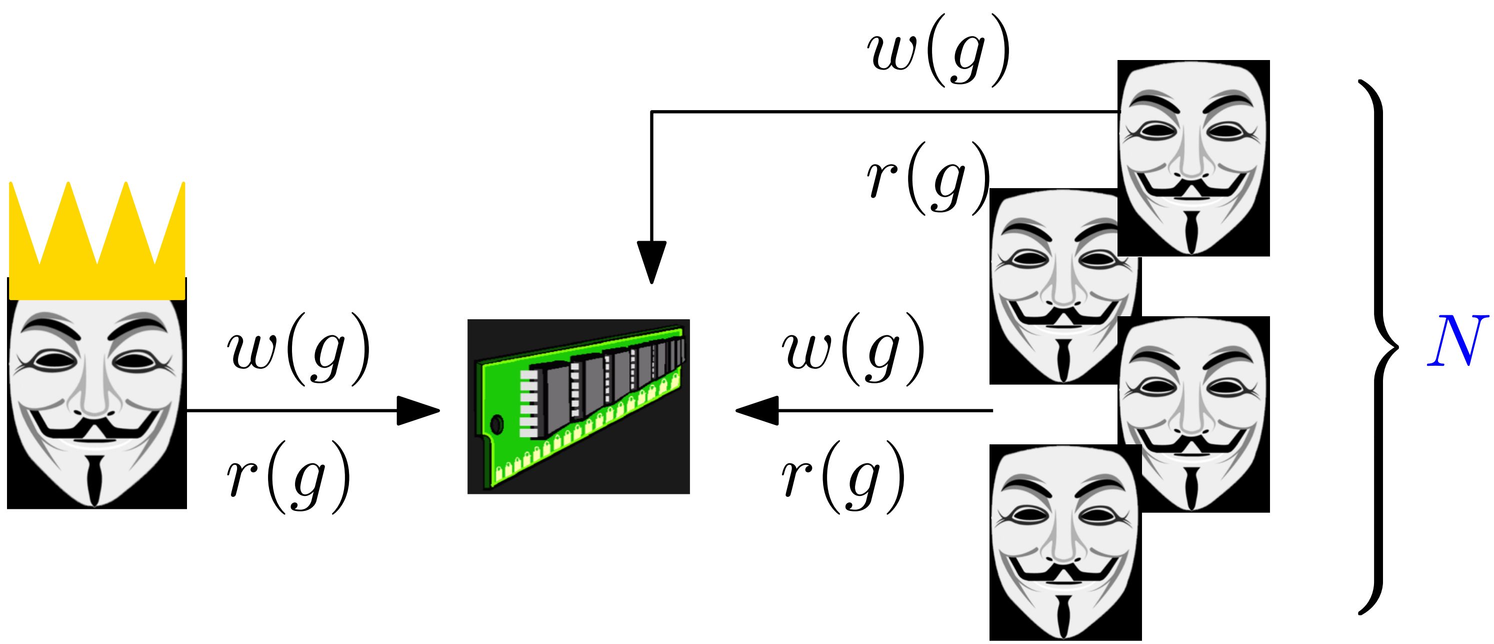


processes have no identity

communication through a shared bounded-value register

register can be read from, written to but not **locked**

1 leader process $\quad + \quad N$ contributors

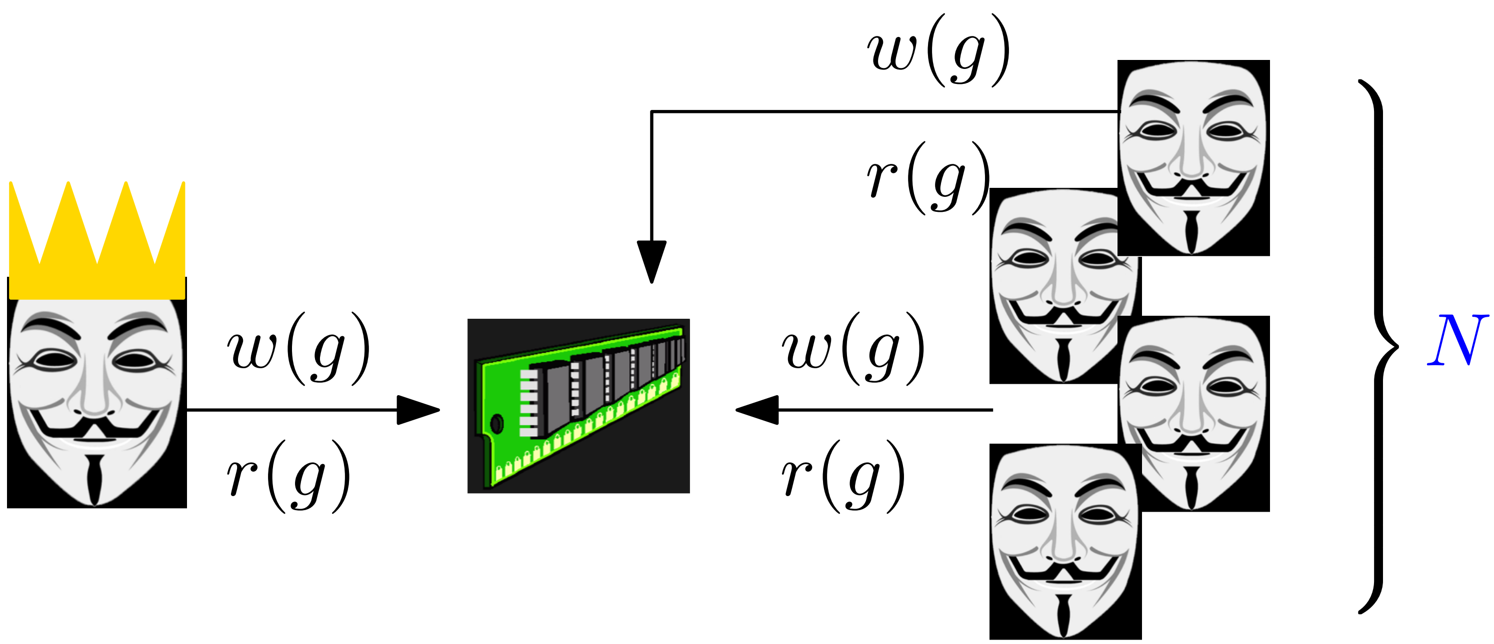


processes have no identity

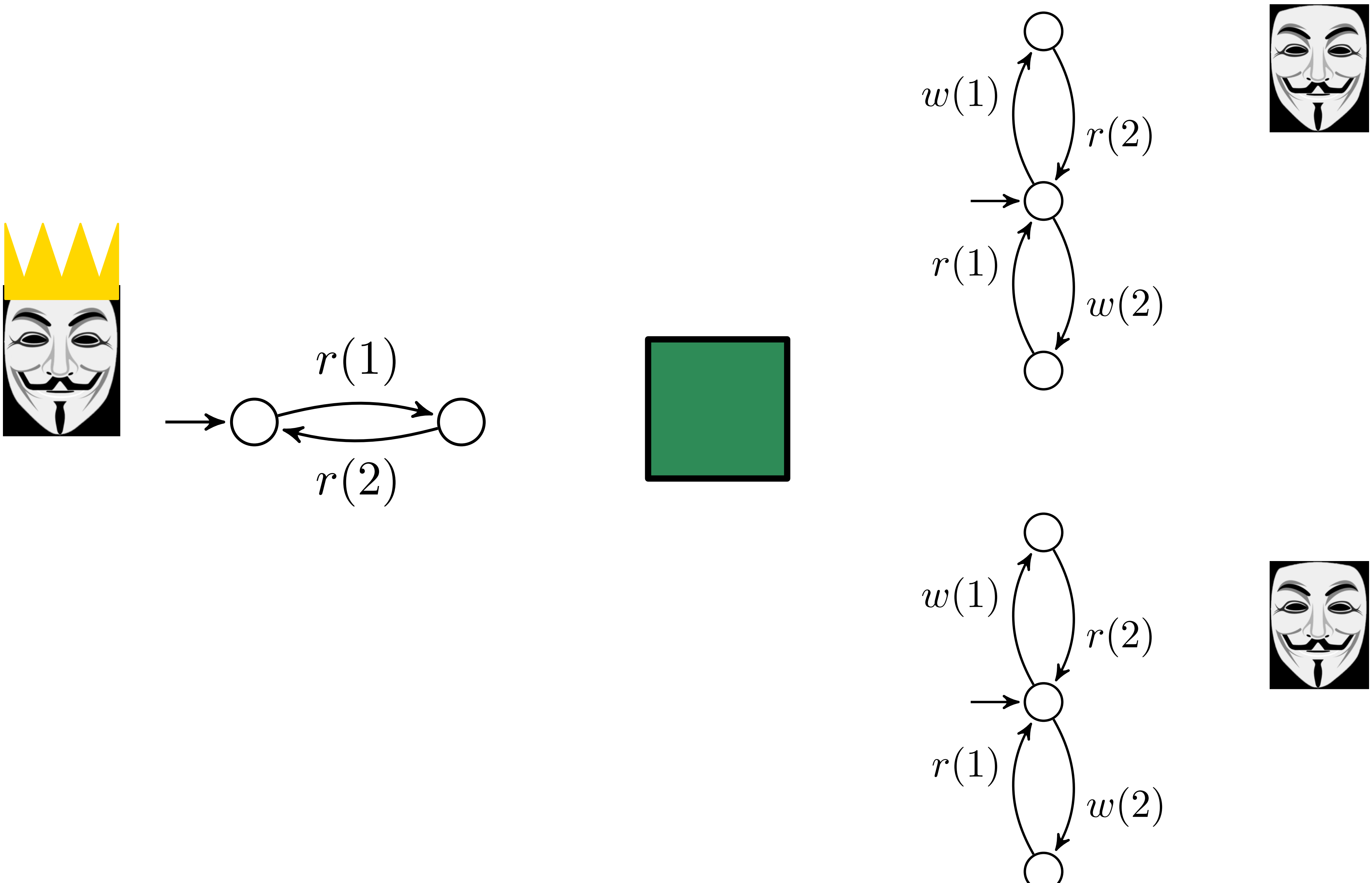
communication through a shared bounded-value register

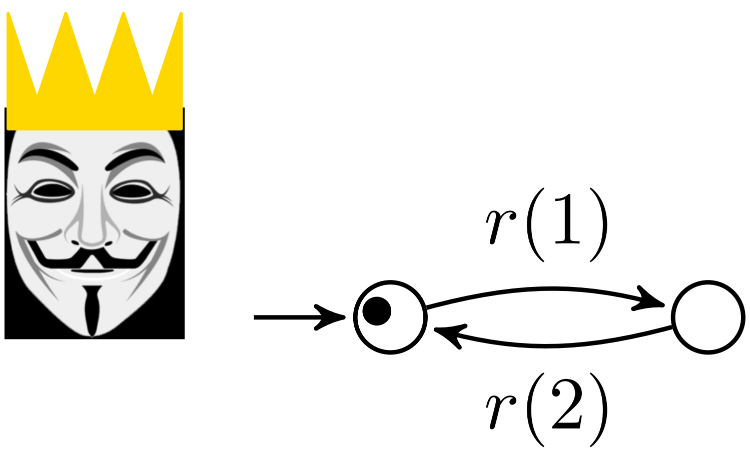
register can be read from, written to but not **locked**

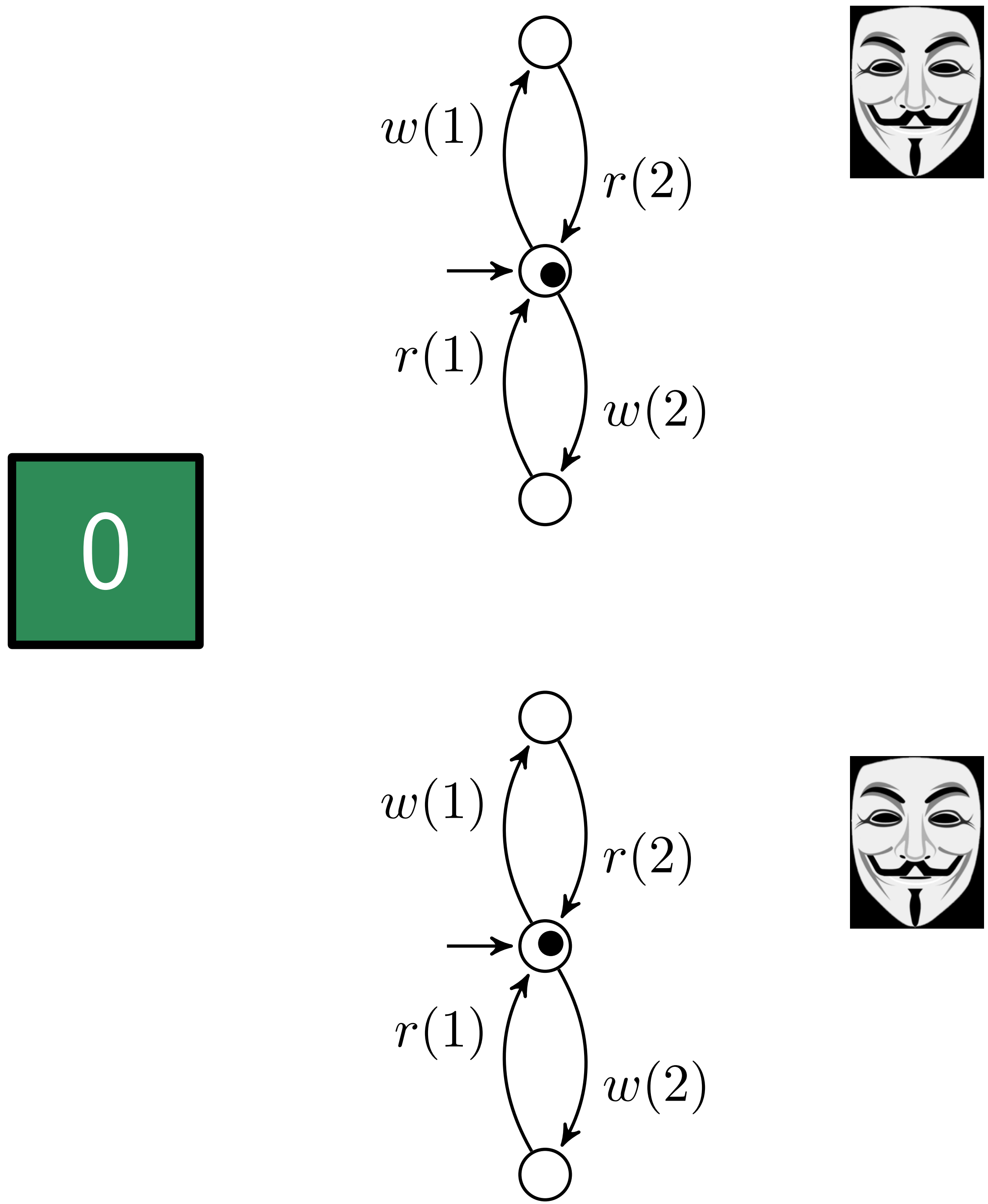
1 leader process $+$ N contributors



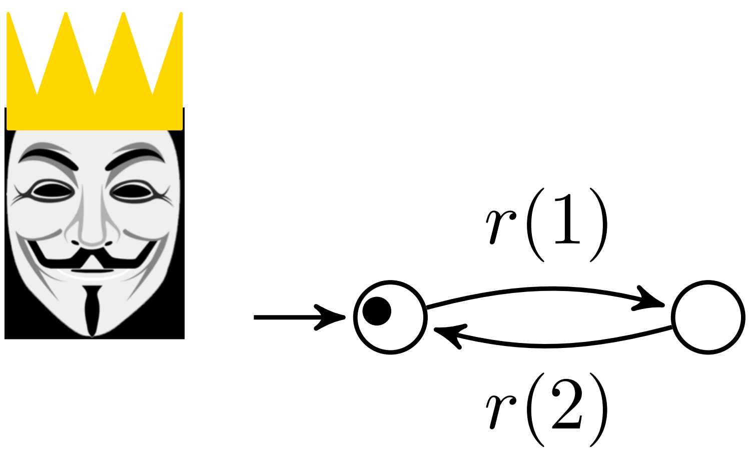
What can such network compute?

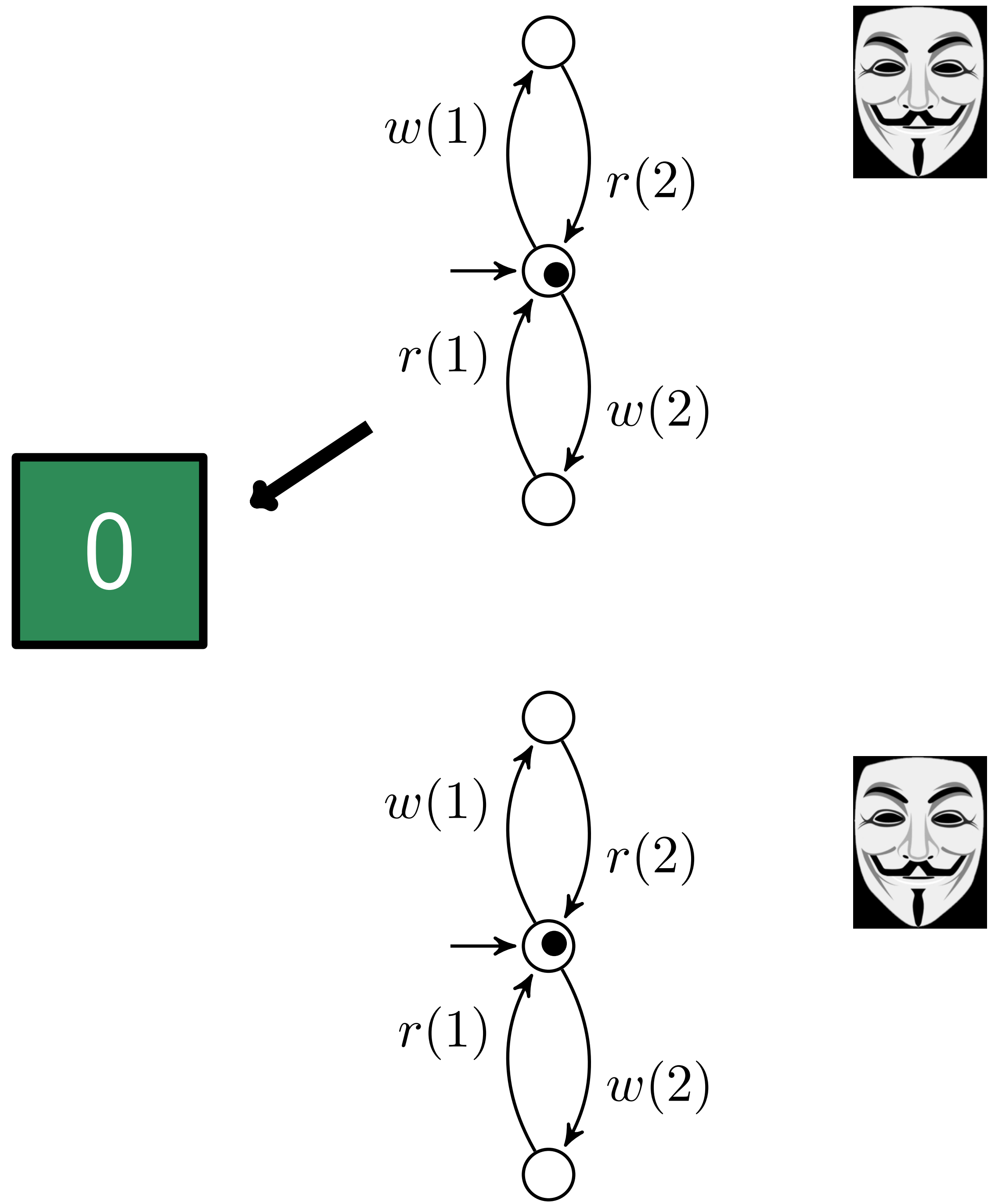

$r(1)$
$r(2)$
$w(1)$
$r(2)$
$r(1)$
$w(2)$
$w(1)$
$r(2)$
$r(1)$
$w(2)$



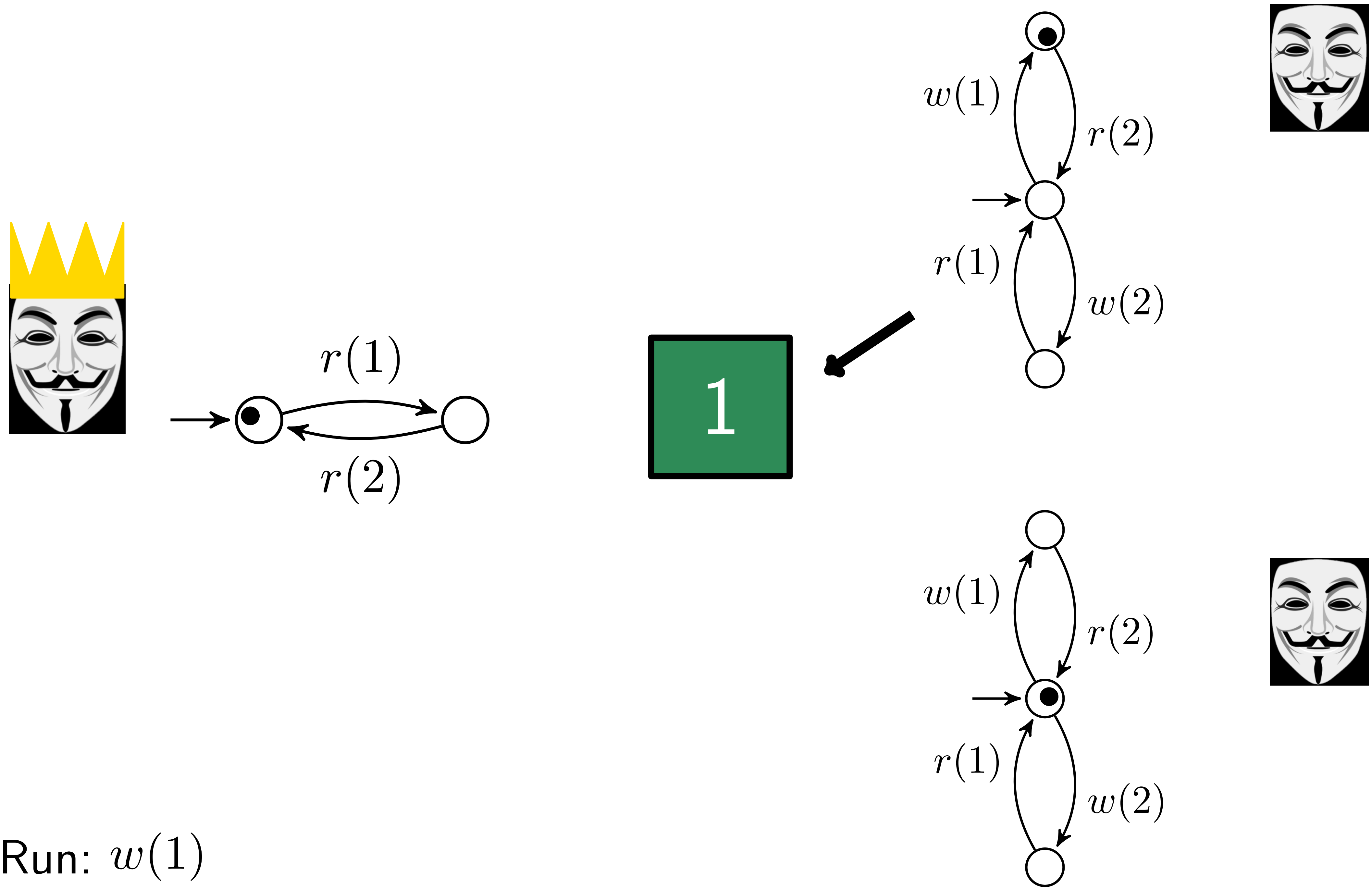


Run:

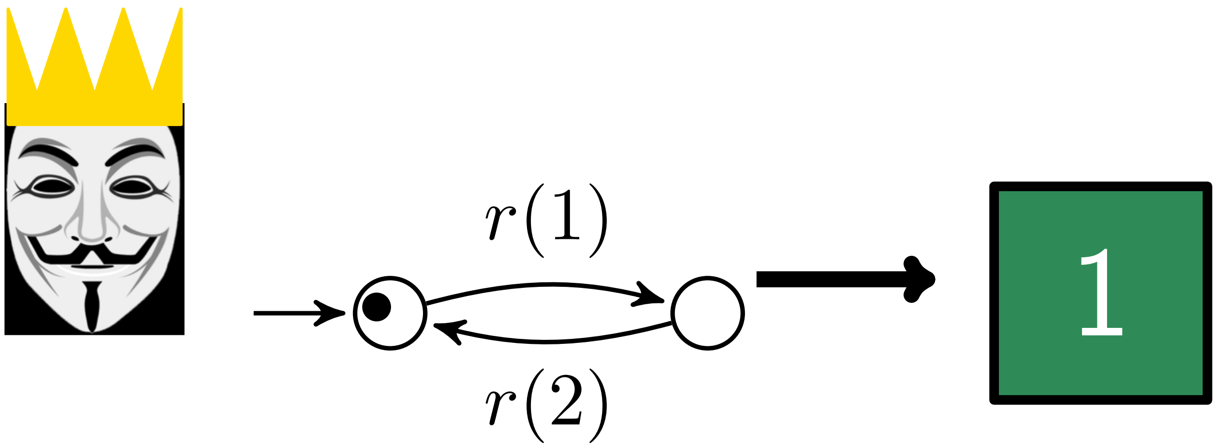


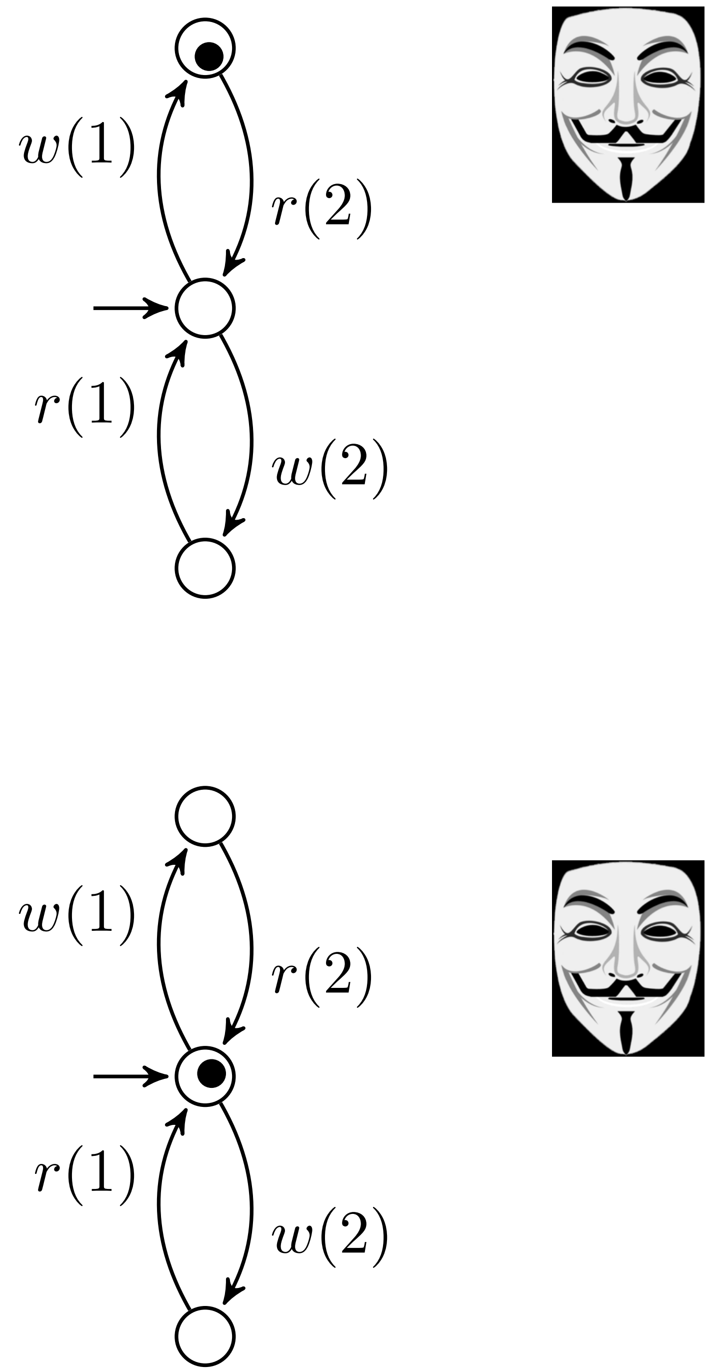


Run:

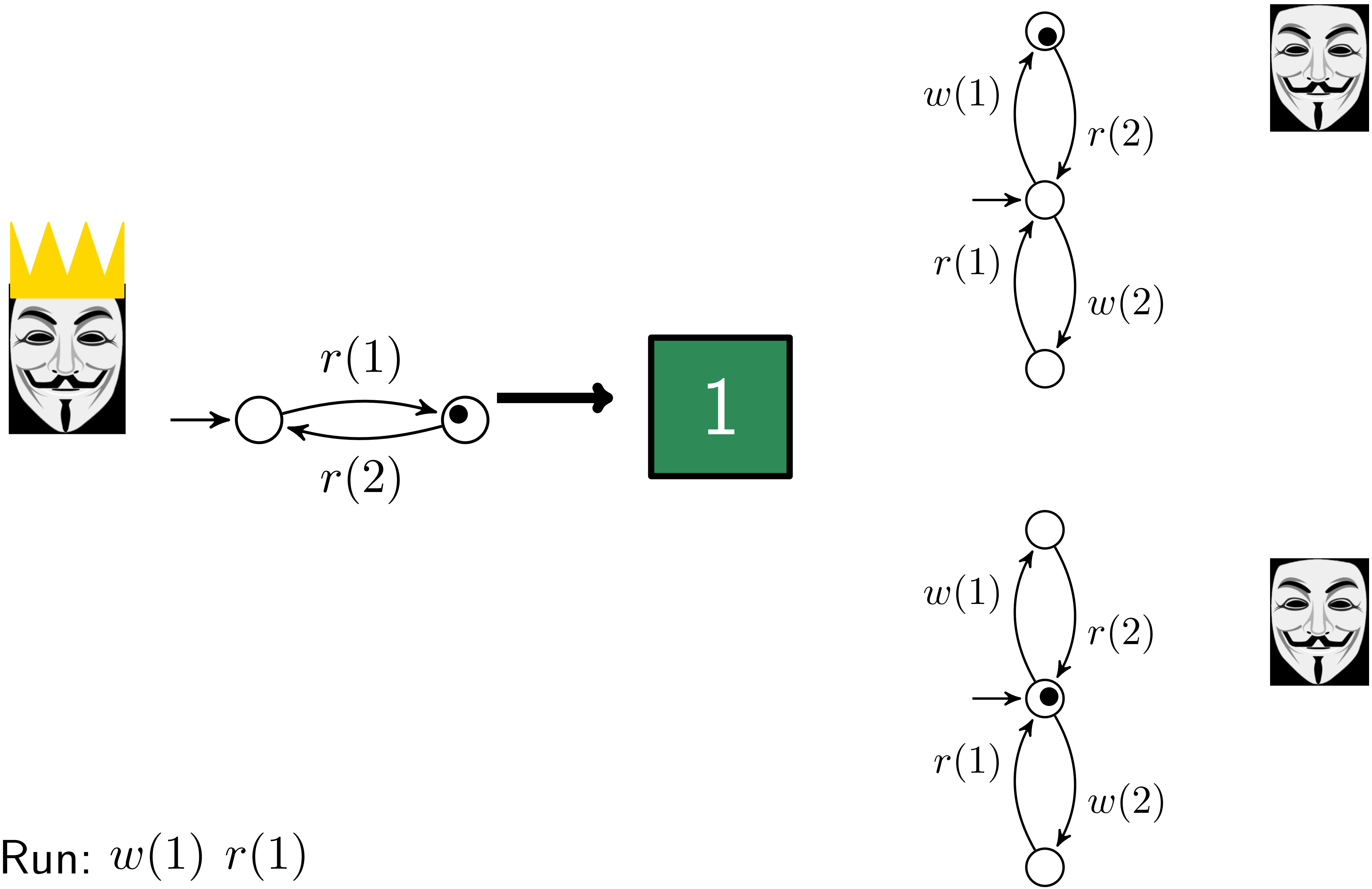


Run: $w(1)$

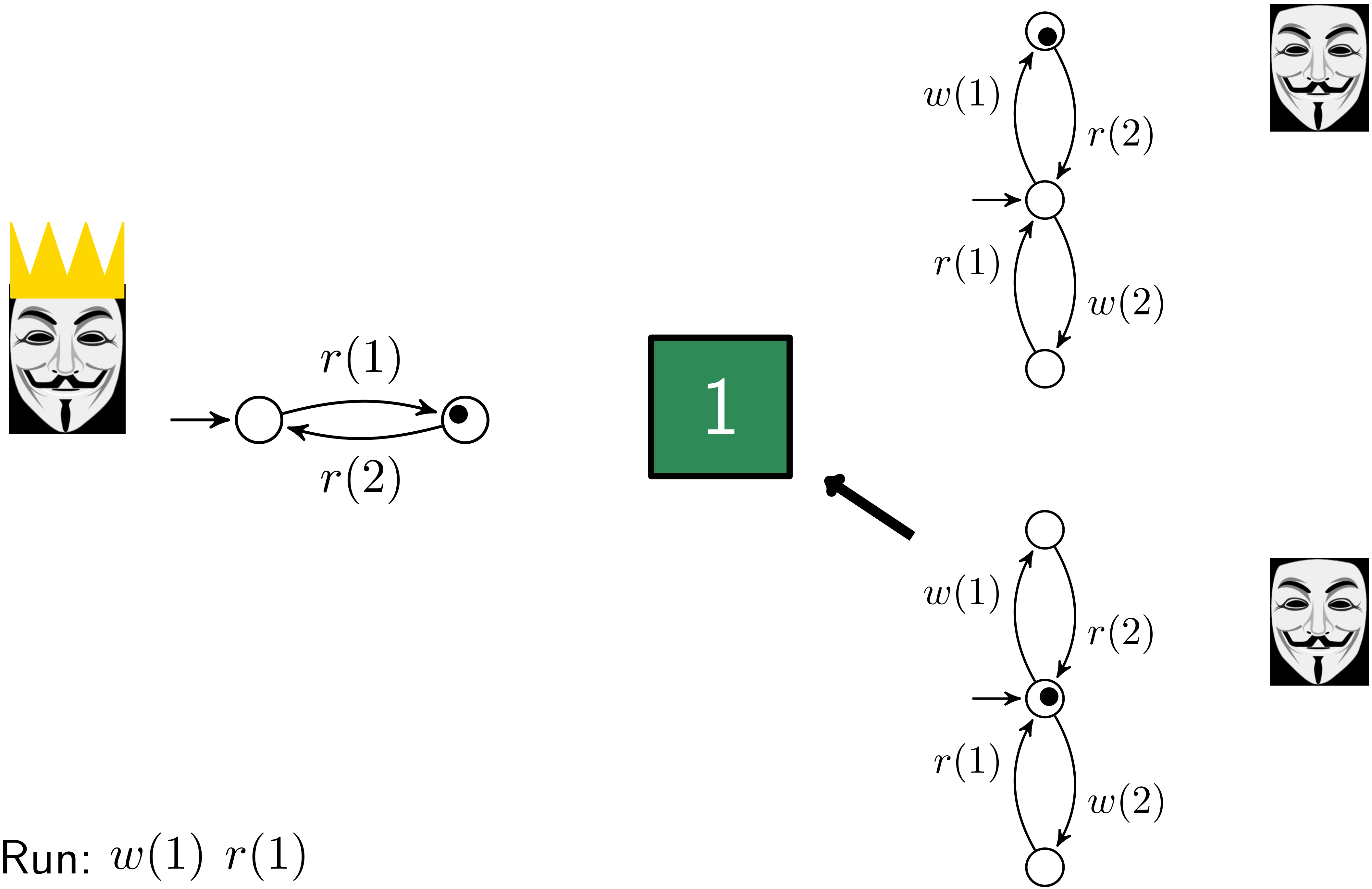




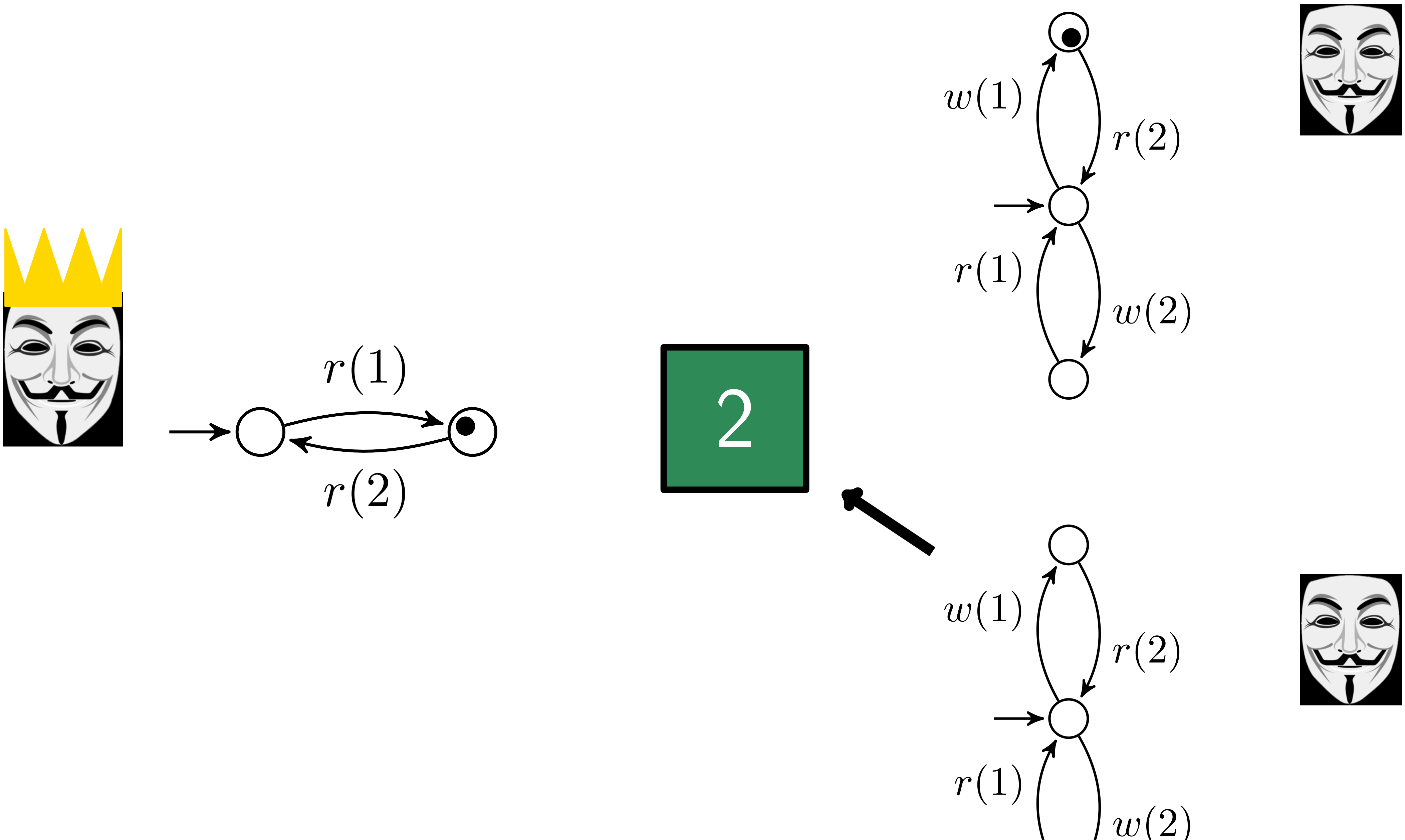
Run: $w(1)$



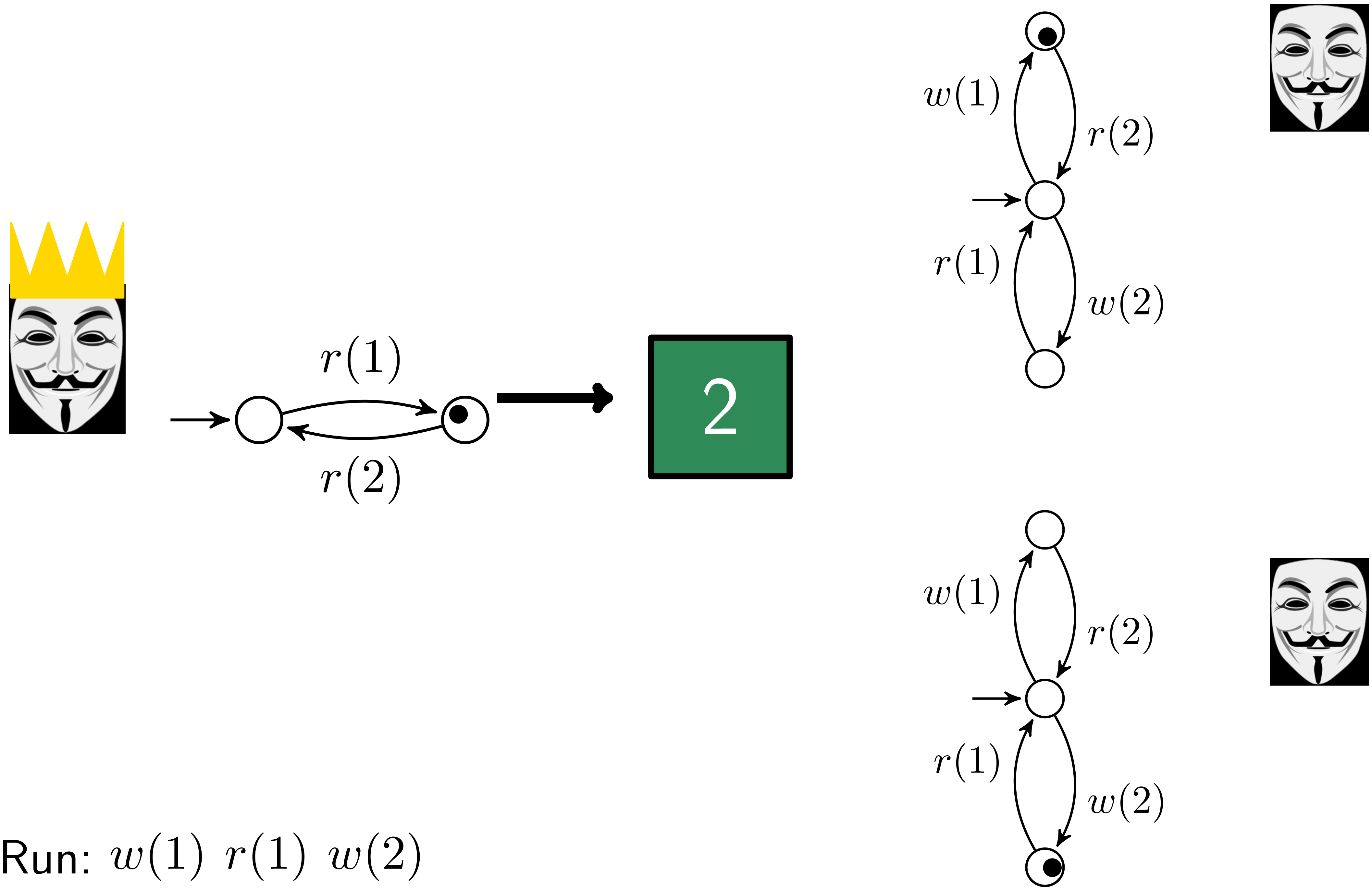
Run: $w(1)\ r(1)$

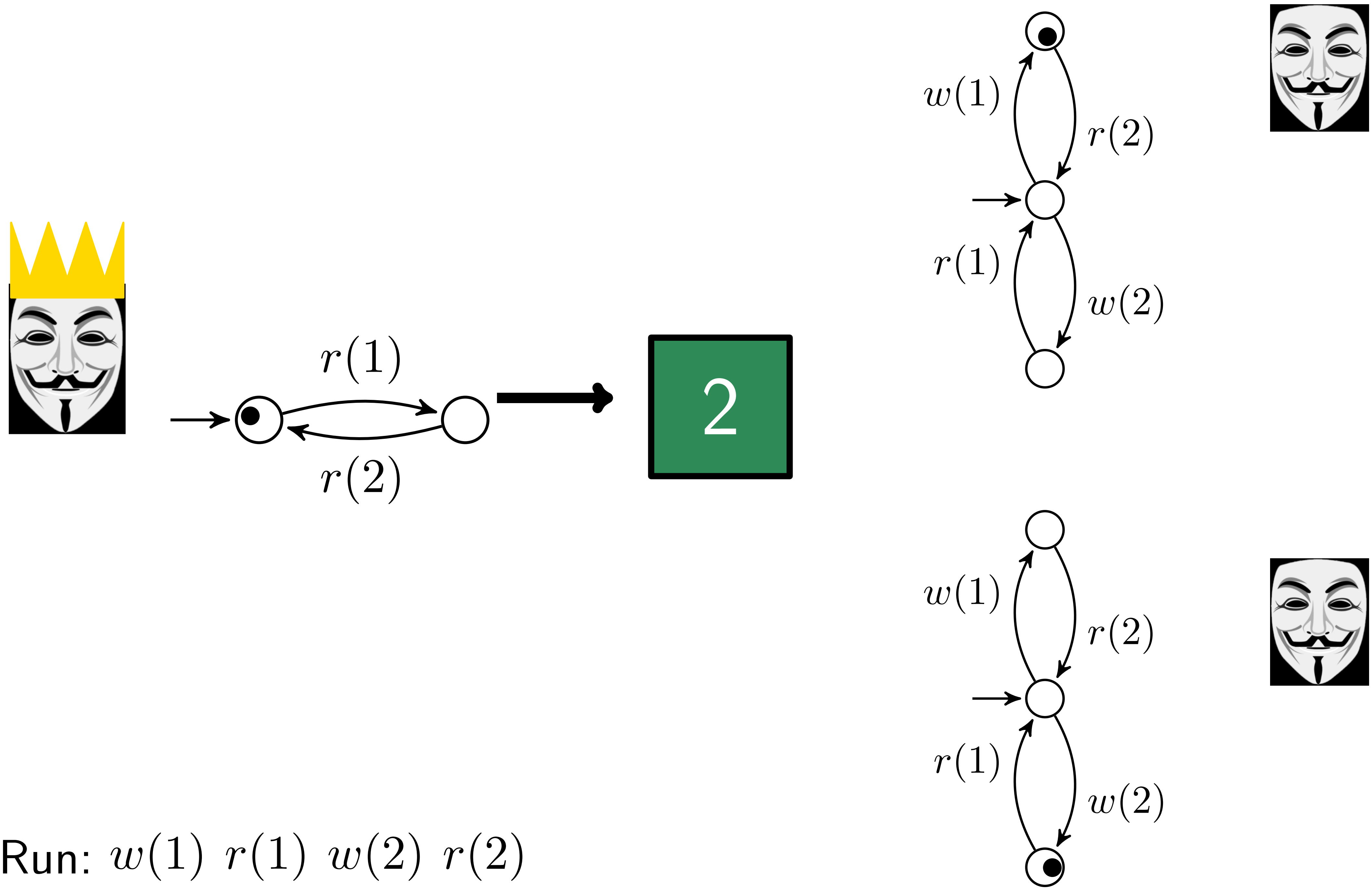


Run: $w(1)\ r(1)$

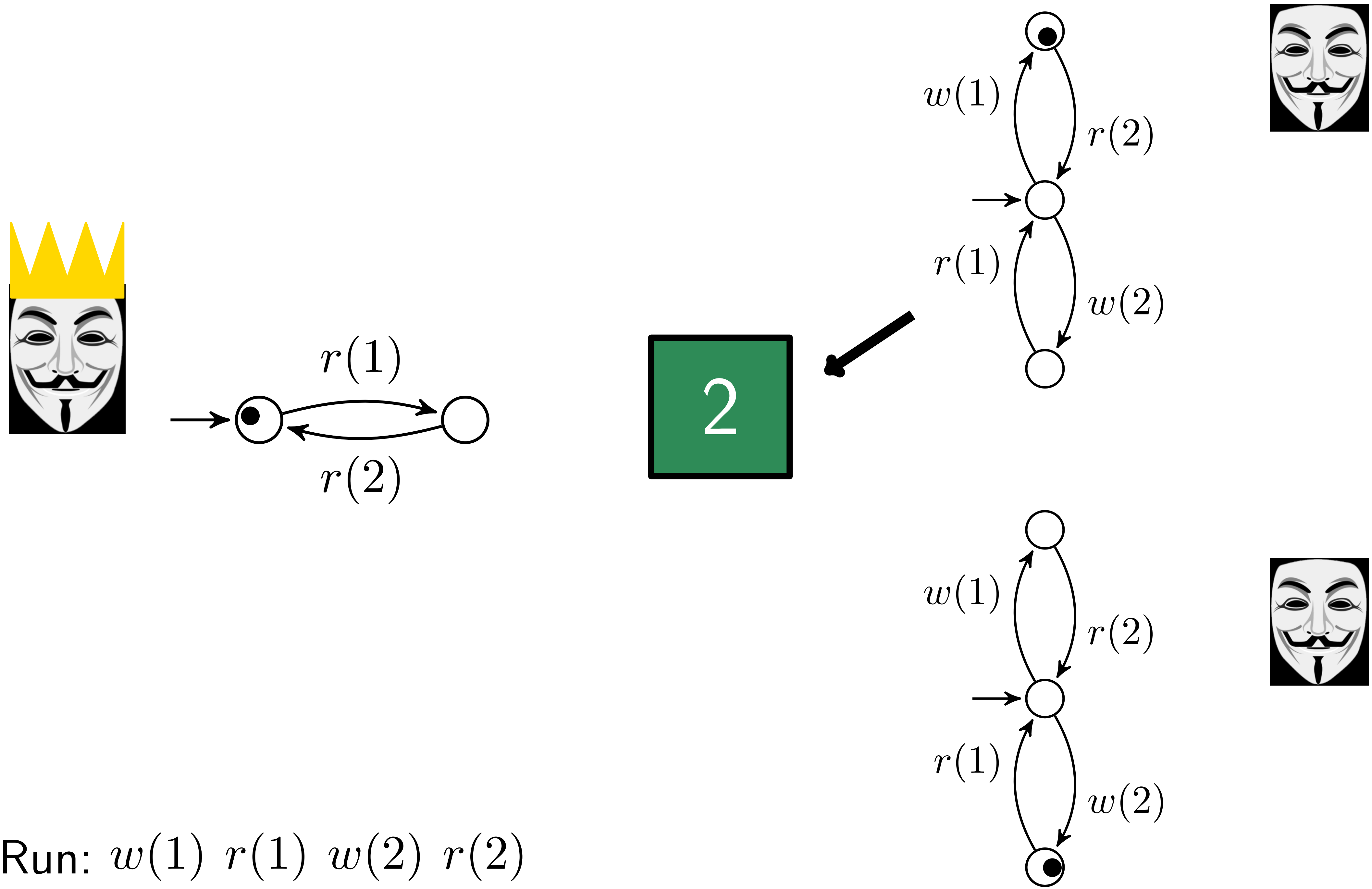


Run: $w(1)\ r(1)\ w(2)$

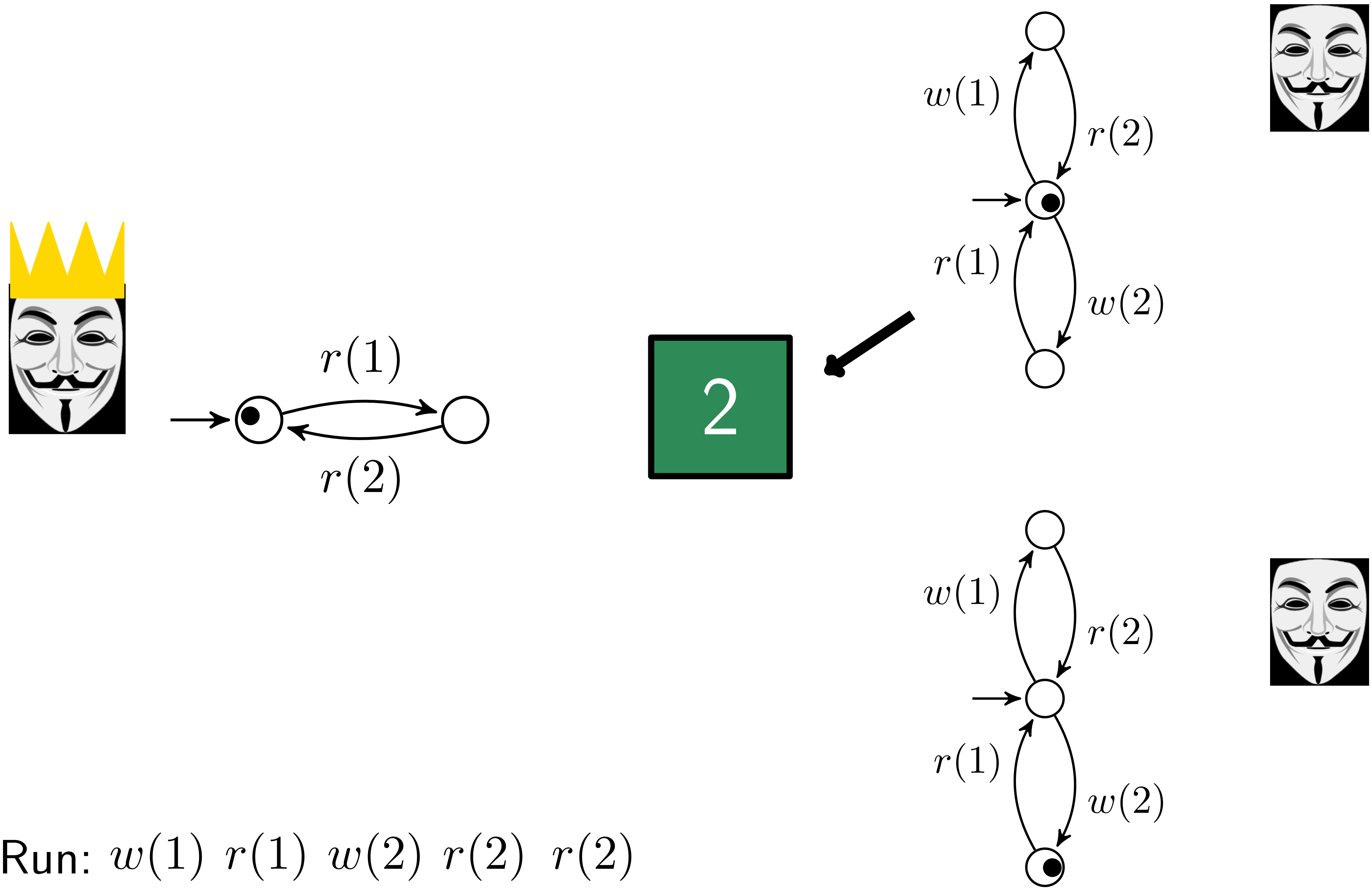

r(1)
r(2)
2
w(1)
r(2)
r(1)
w(2)
w(1)
r(2)
r(1)
w(2)
Run: w(1) r(1) w(2)



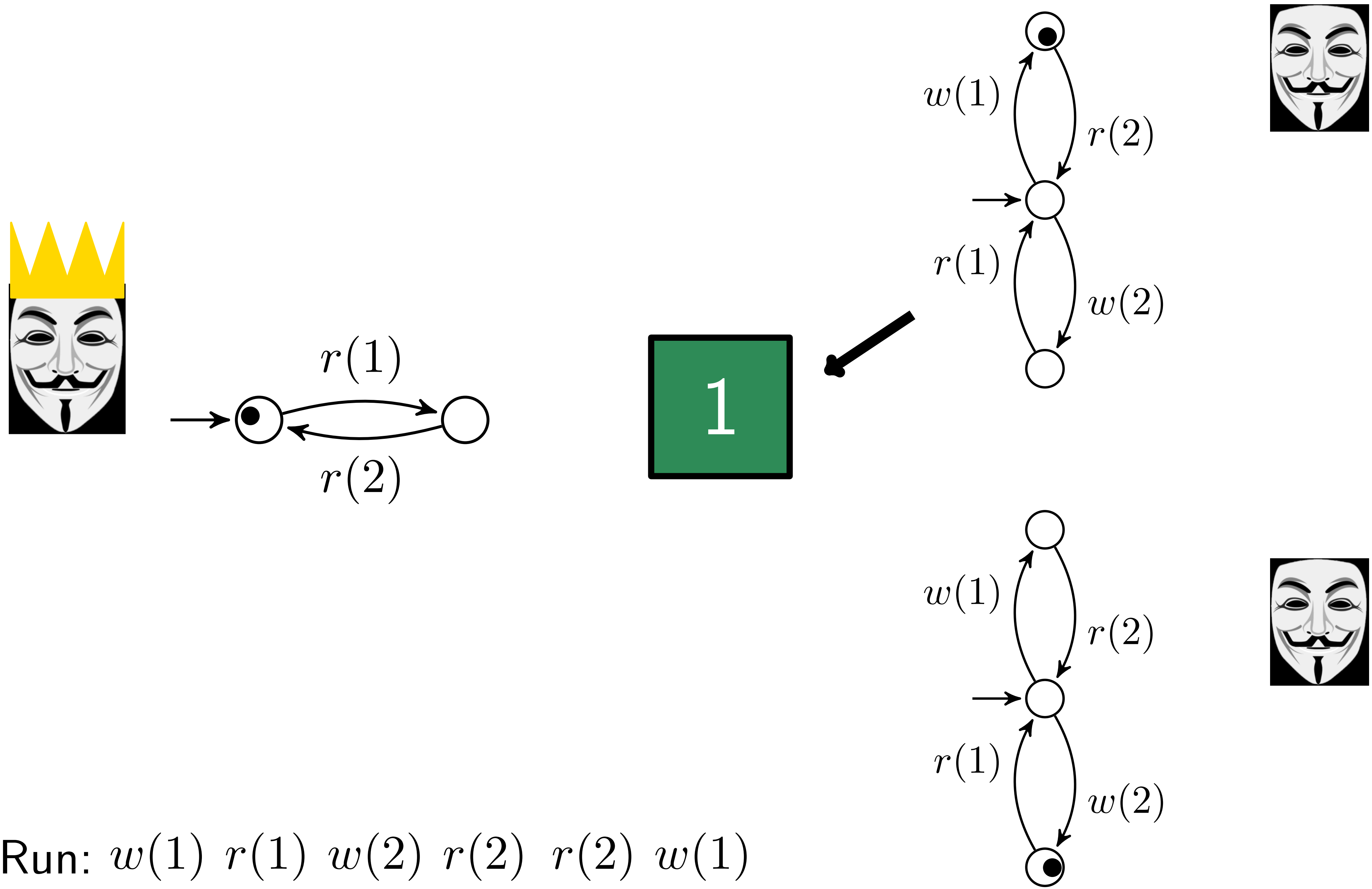
Run: $w(1)\ r(1)\ w(2)\ r(2)$



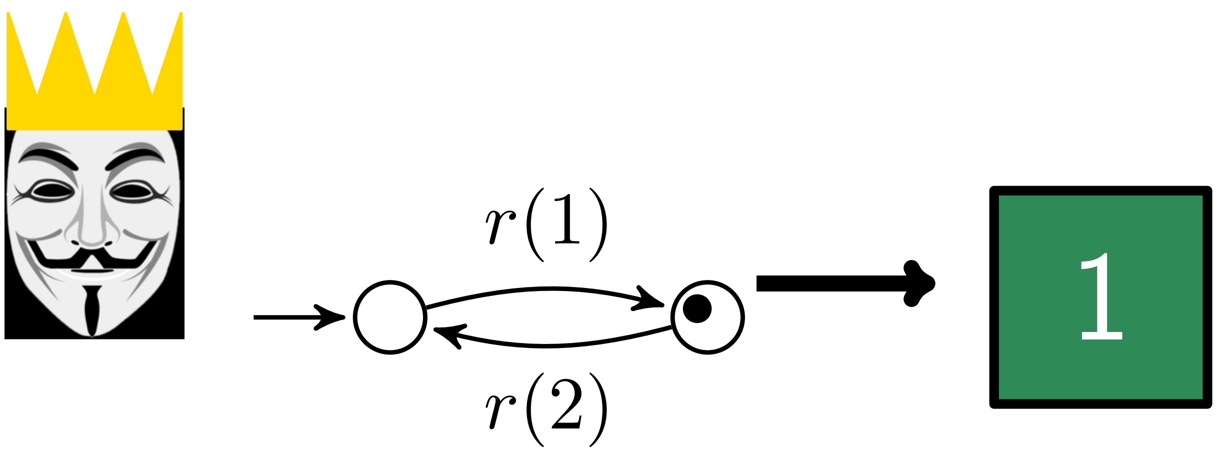
Run: $w(1)\ r(1)\ w(2)\ r(2)$

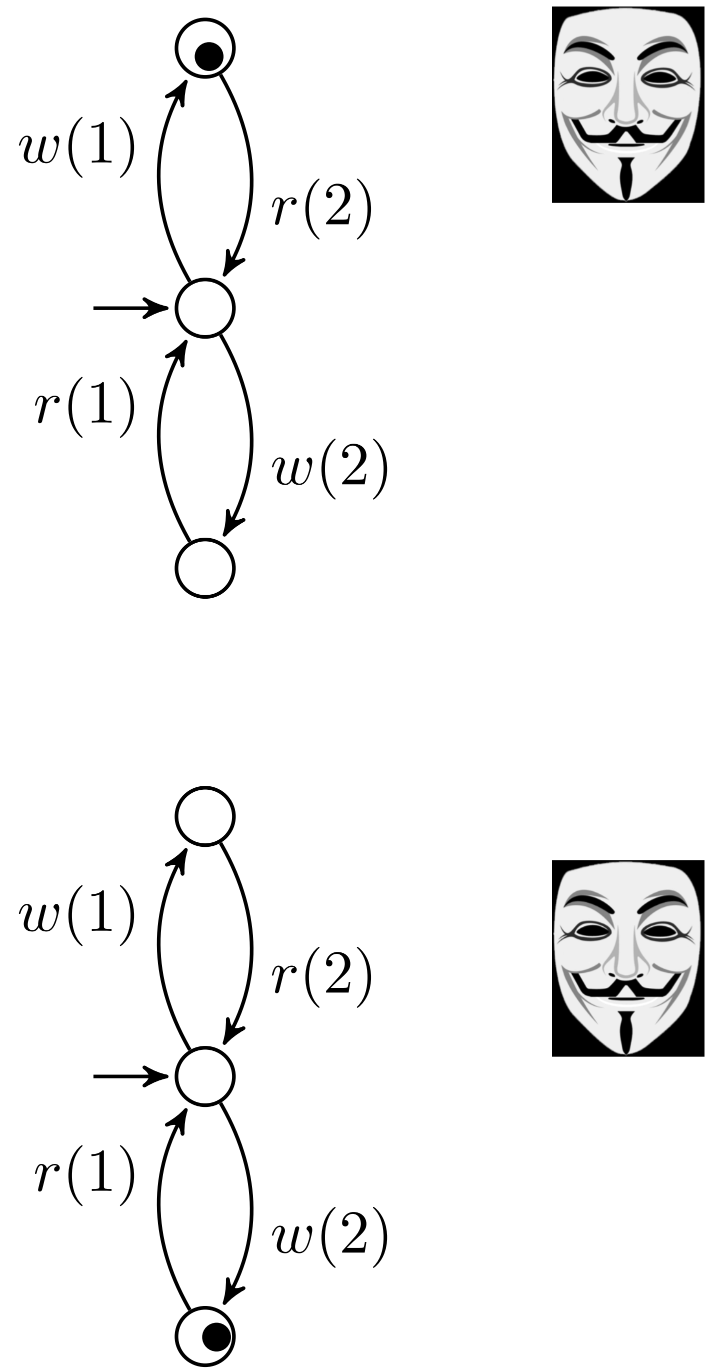


Run: $w(1)\ r(1)\ w(2)\ r(2)\ r(2)$

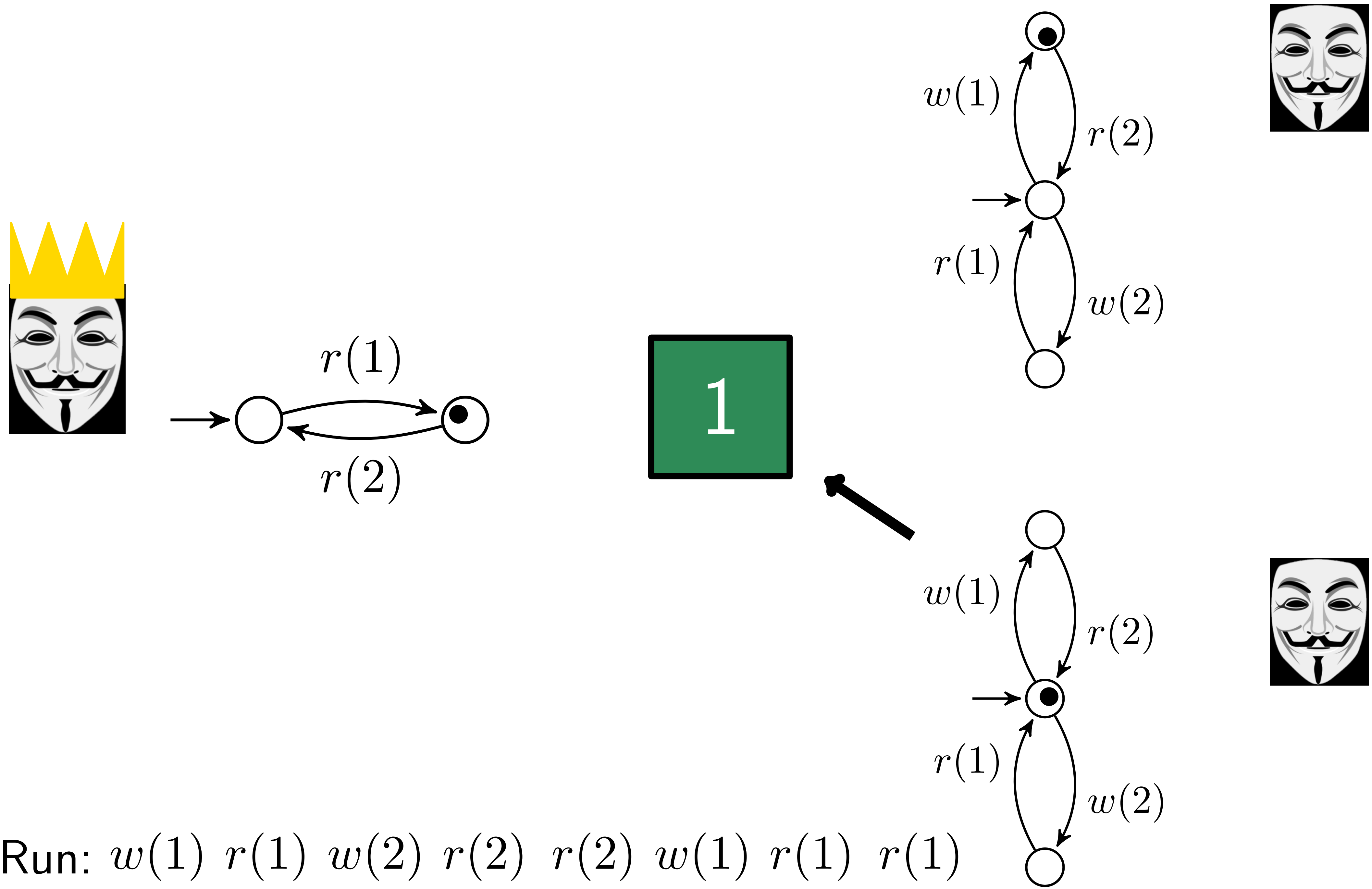


Run: $w(1)\ r(1)\ w(2)\ r(2)\ r(2)\ w(1)$





Run: $w(1)\ r(1)\ w(2)\ r(2)\ r(2)\ w(1)\ r(1)$



Run: $w(1)\ r(1)\ w(2)\ r(2)\ r(2)\ w(1)\ r(1)\ r(1)$

How much can non-atomic networks compute?

Verification reveals what non-atomic networks can(not) do

How much can non-atomic networks compute?

Verification reveals what non-atomic networks can(not) do

Hardness: X-hardness of safety checking implies non-atomic networks can solve (solving by reaching q) X-hard problems

How much can non-atomic networks compute?

Verification reveals what non-atomic networks can(not) do

Hardness: X-hardness of safety checking implies non-atomic networks can solve (solving by reaching q) X-hard problems

Membership: Safety checking in X implies non-atomic networks can not solve problems harder than X

How much can non-atomic networks compute?

Verification reveals what non-atomic networks can(not) do

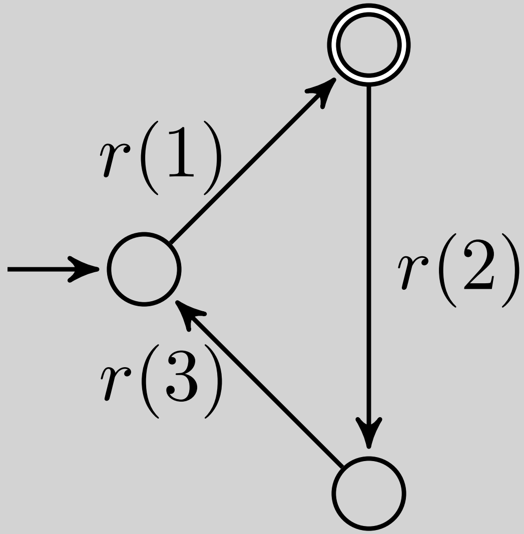
Hardness: X-hardness of safety checking implies non-atomic networks can solve (solving by reaching q) X-hard problems

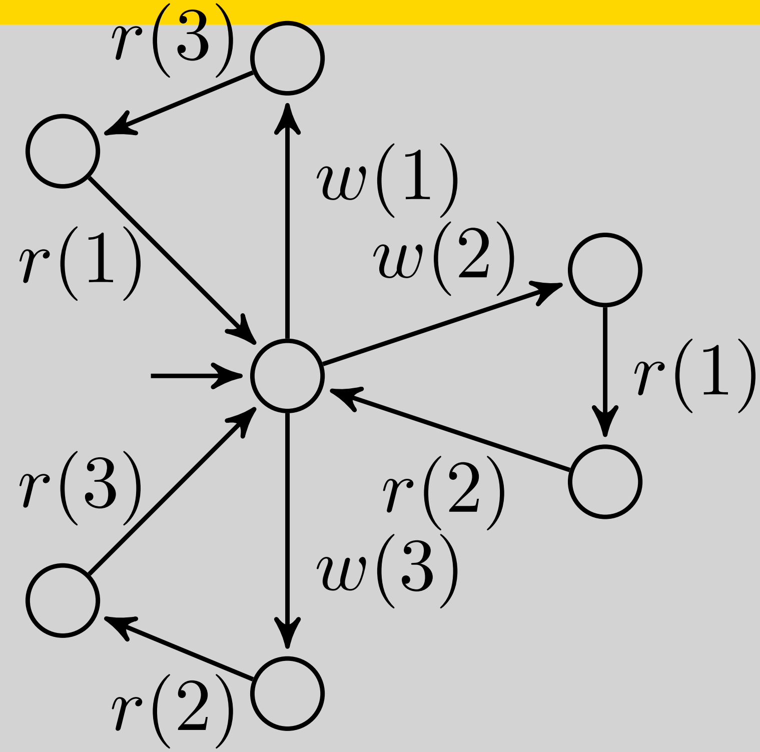
Membership: Safety checking in X implies non-atomic networks can not solve problems harder than X

Today solving means repeatedly reaching q

Parameterized model-checking

Given:



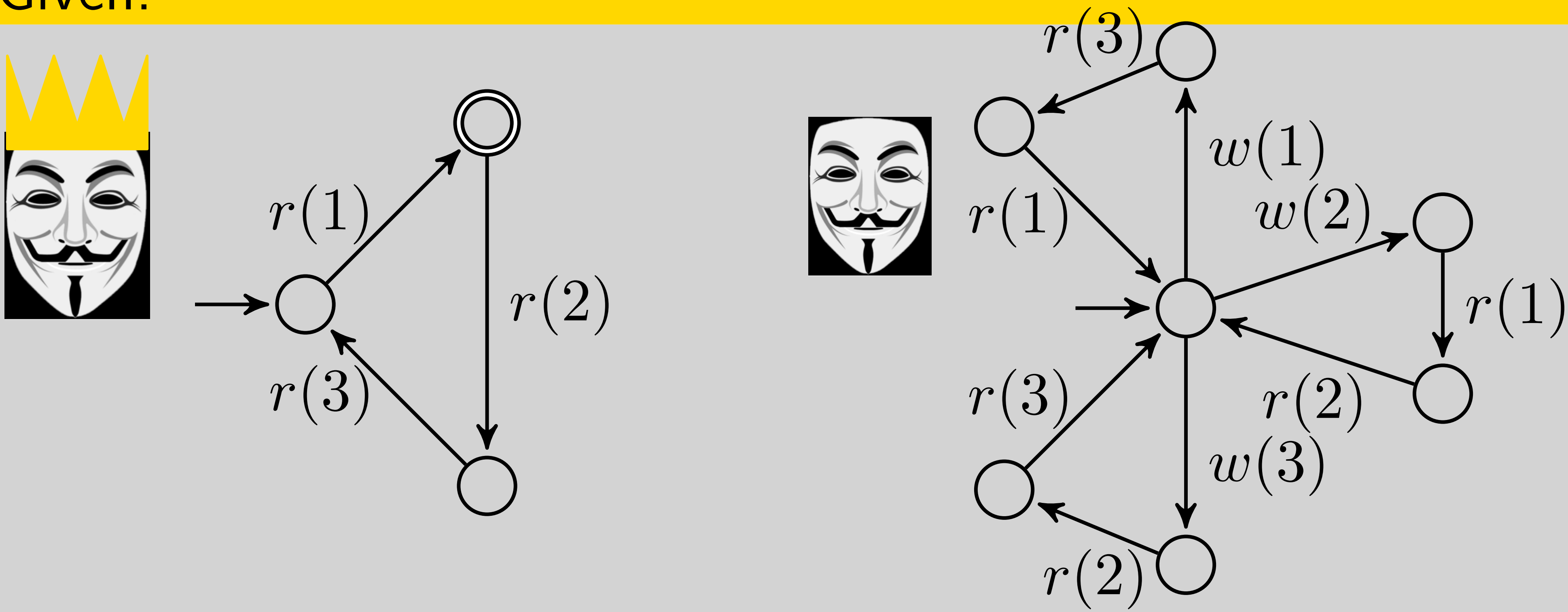


N copies

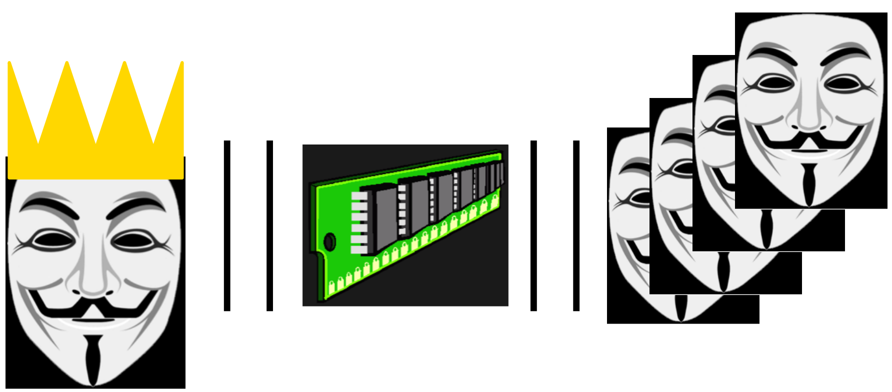
Is there N:   have an accepting run ?

Parameterized model-checking

Given:



N copies

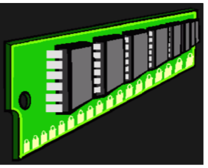
Is there N:  have an accepting run ?

Is there an infinite accepting run with finitely many processes?

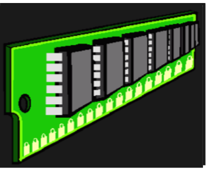
NP-complete

NP-complete

when all processes are given by finite state machine



$w(1)$ $r(1)$ $r(1)$ $w(2)$ $r(2)$ $w(1)$ $r(1)$ $r(1)$ $w(3)$ $r(3)$



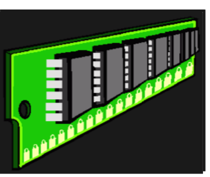
$r(1)$ $w(1)$

$w(1)$ $r(1)$ $r(1)$ $w(2)$ $r(2)$ $w(1)$ $r(1)$ $r(1)$ $w(3)$ $r(3)$

$r(1)$ $w(1)$

$r(1)$ $w(1)$

$r(1)$ $w(1)$

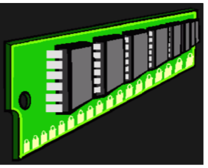
 $w(1)$ $r(1)$ $r(1)$ $w(2)$ $r(2)$ $w(1)$ $r(1)$ $r(1)$ $w(3)$ $r(3)$

Copycat Lemma

$w(1)$ $r(2)$ $r(1)$ $r(3)$

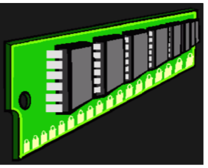
$r(1)$ $w(2)$ $r(1)$ $w(3)$

$r(1)$ $w(1)$

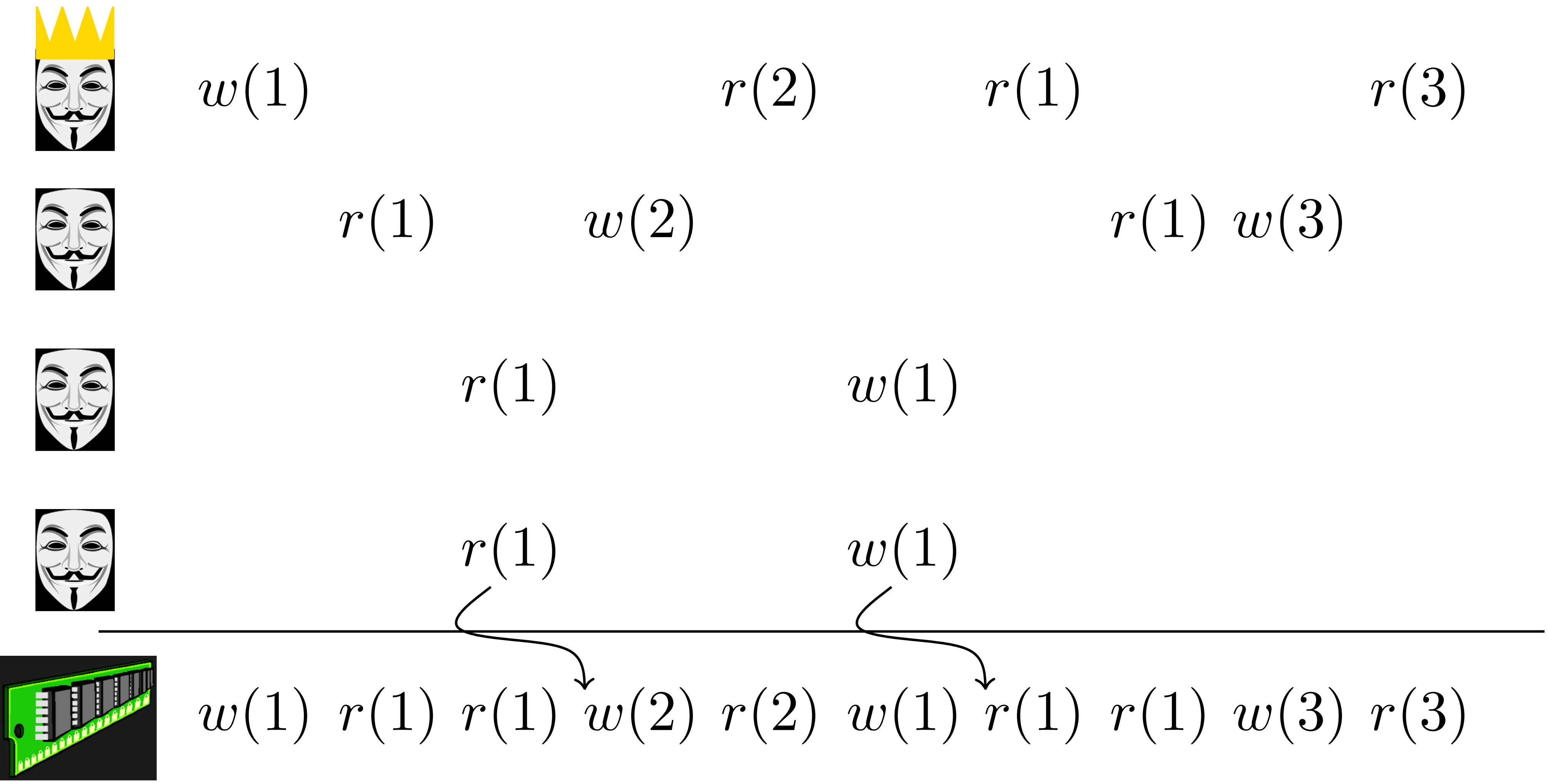


$w(1)$ $r(1)$ $r(1)$ $w(2)$ $r(2)$ $w(1)$ $r(1)$ $r(1)$ $w(3)$ $r(3)$

Copycat Lemma

	$w(1)$				$r(2)$		$r(1)$			$r(3)$
		$r(1)$		$w(2)$				$r(1)$	$w(3)$	
			$r(1)$			$w(1)$				
			$r(1)$			$w(1)$				
	$w(1)$	$r(1)$	$r(1)$	$w(2)$	$r(2)$	$w(1)$	$r(1)$	$r(1)$	$w(3)$	$r(3)$

Copycat Lemma



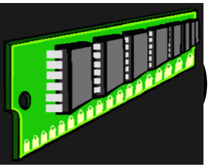
Copycat Lemma

 $w(1)$ $r(2)$ $r(1)$ $r(3)$

 $r(1)$ $w(2)$ $r(1)$ $w(3)$

 $r(1)$ $w(1)$

 $r(1)$ $w(1)$

 $r(1)$ $r(1)$ $r(1)$ $w(2)$ $r(2)$ $w(1)$ $w(1)$ $r(1)$ $r(1)$ $w(3)$ $r(3)$

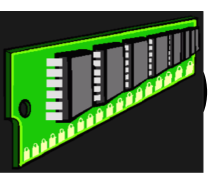
Copycat Lemma

$w(1)$ $r(2)$ $r(1)$ $r(3)$

$r(1)$ $w(2)$ $r(1)$ $w(3)$

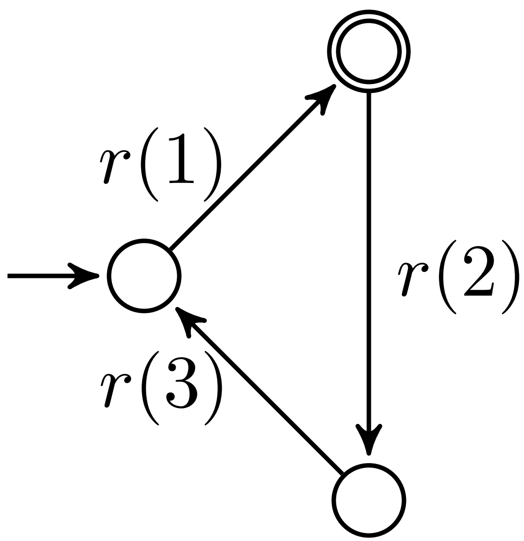
$r(1)$ $w(1)$

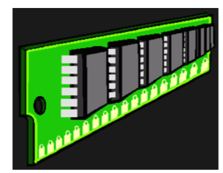
$r(1)$ $r(1)$ $r(1)$ $r(1)$ $w(1)$ $w(1)$ $w(1)$ $w(1)$

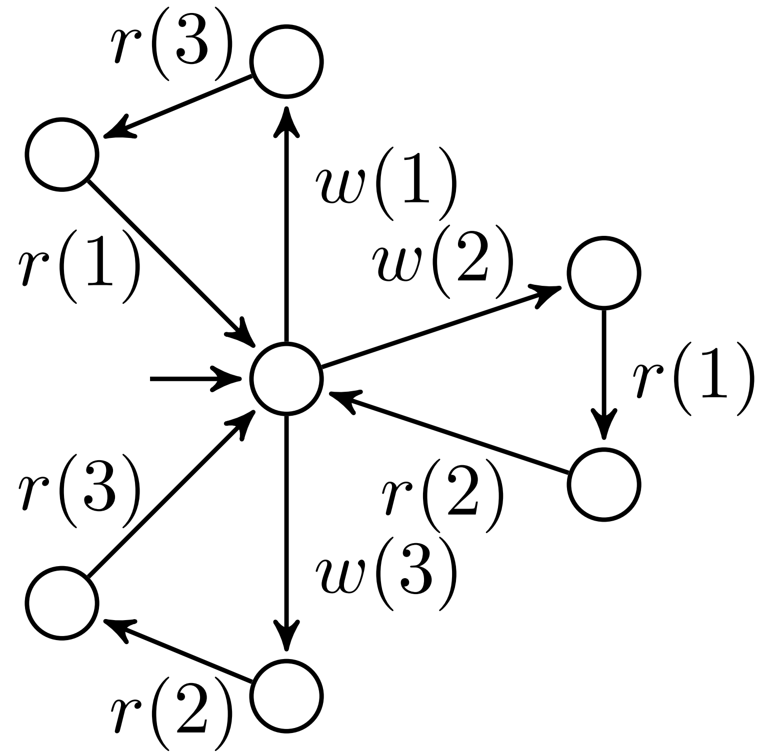


$r(1)$ $r(1)$ $r(1)$ $w(2)$ $r(2)$ $w(1)$ $w(1)$ $r(1)$ $r(1)$ $w(3)$ $r(3)$

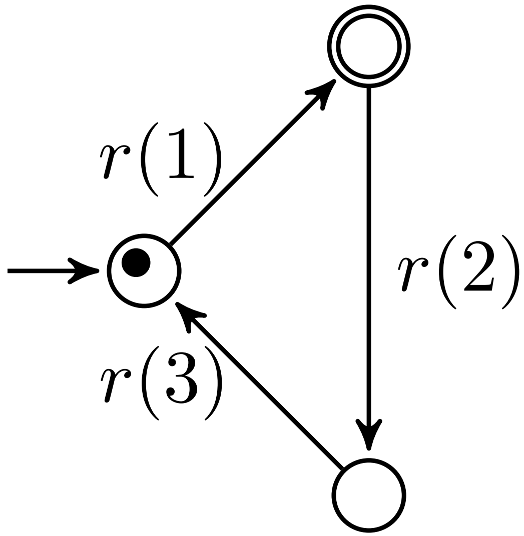
Finite abstraction

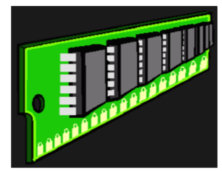


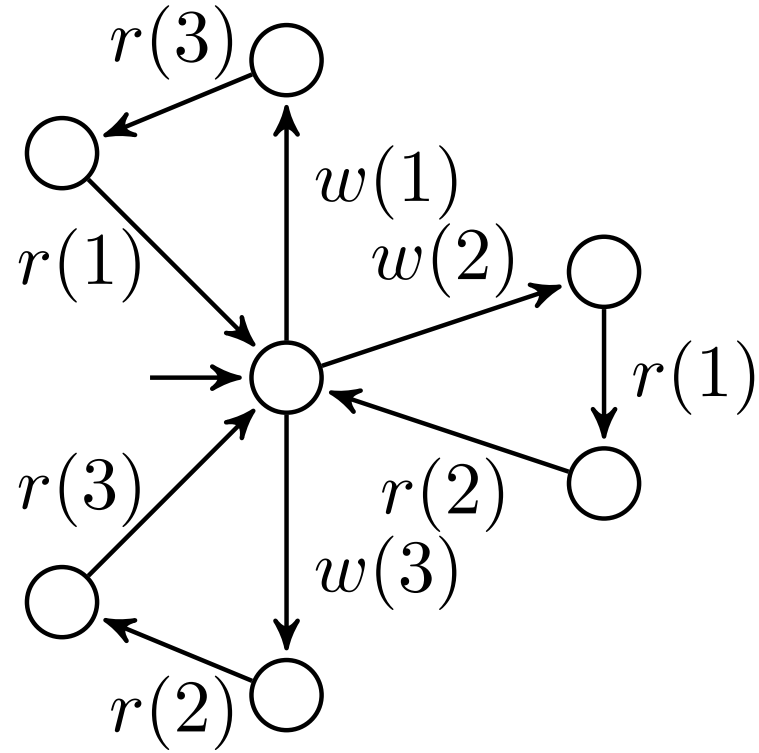




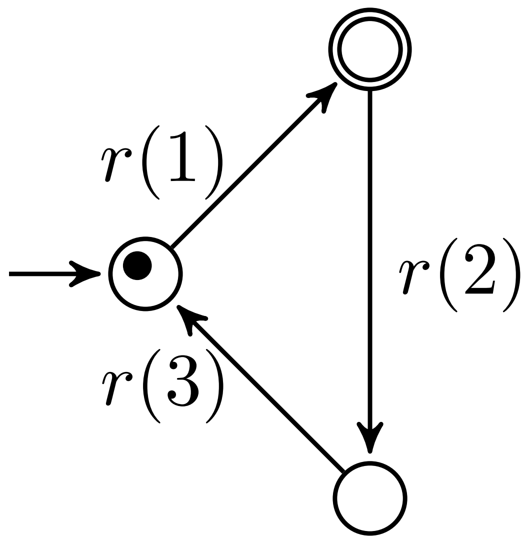
Finite abstraction

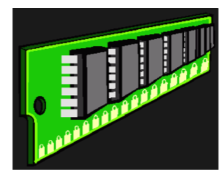


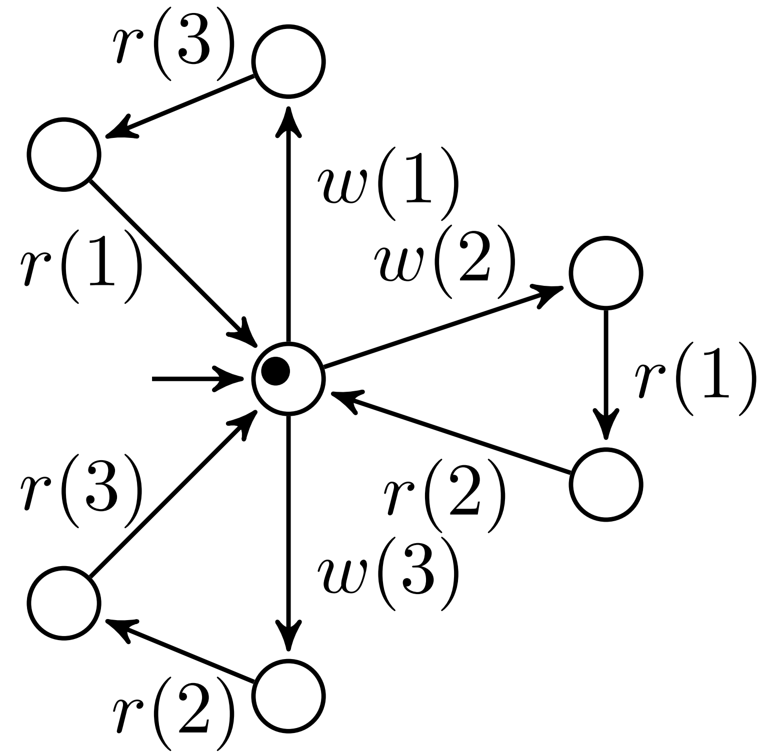




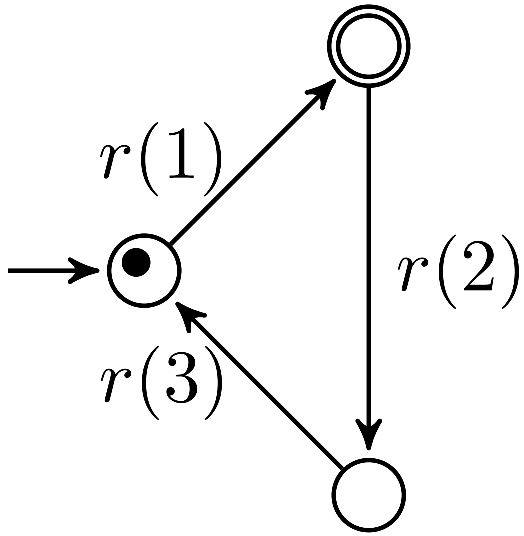
Finite abstraction

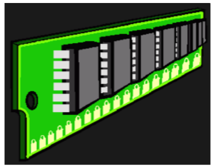


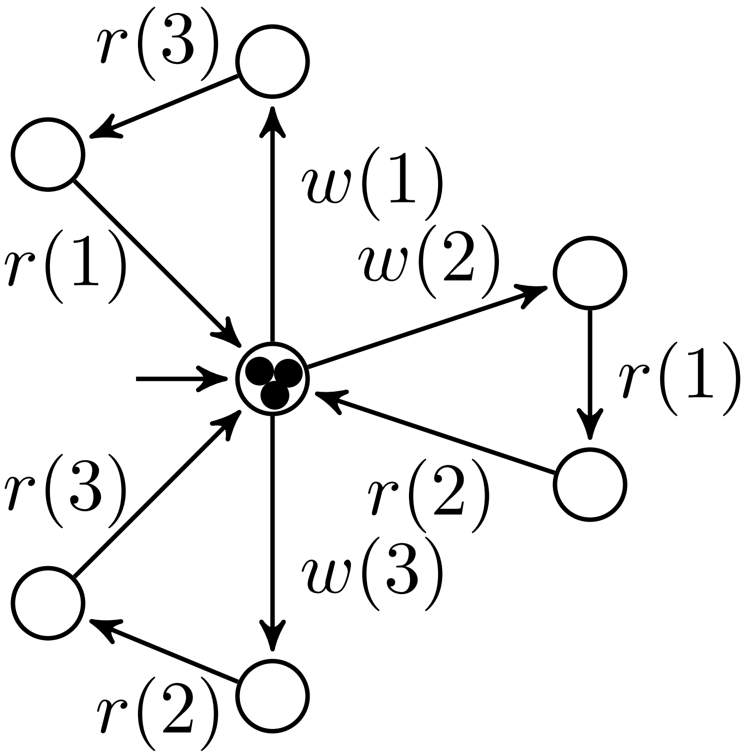




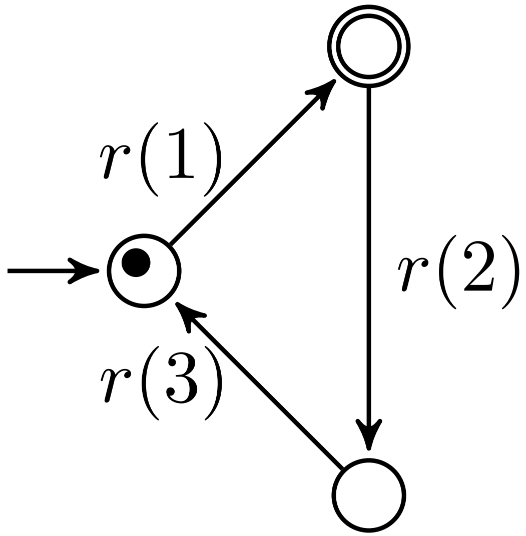
Finite abstraction

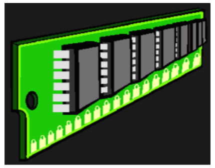


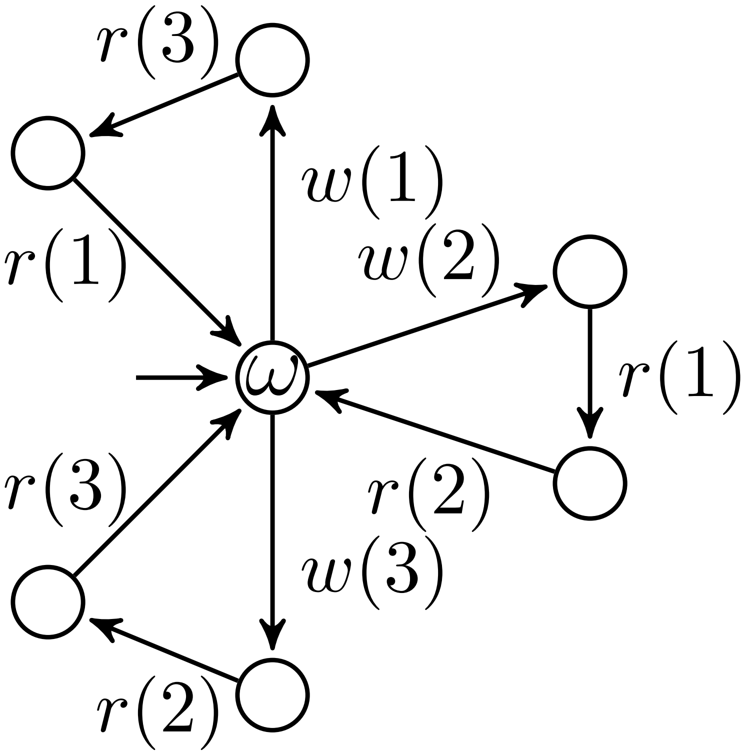




Finite abstraction

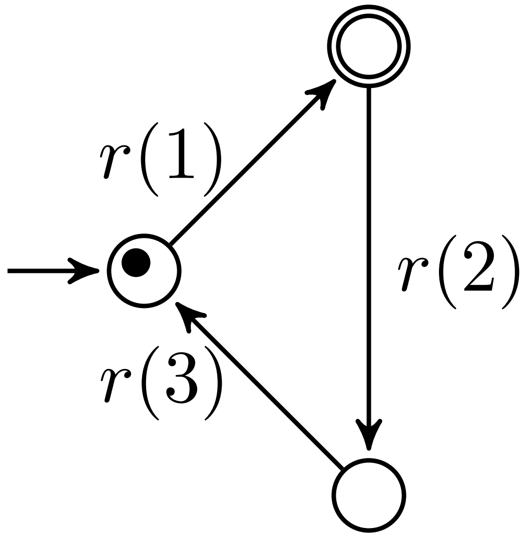


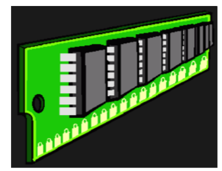


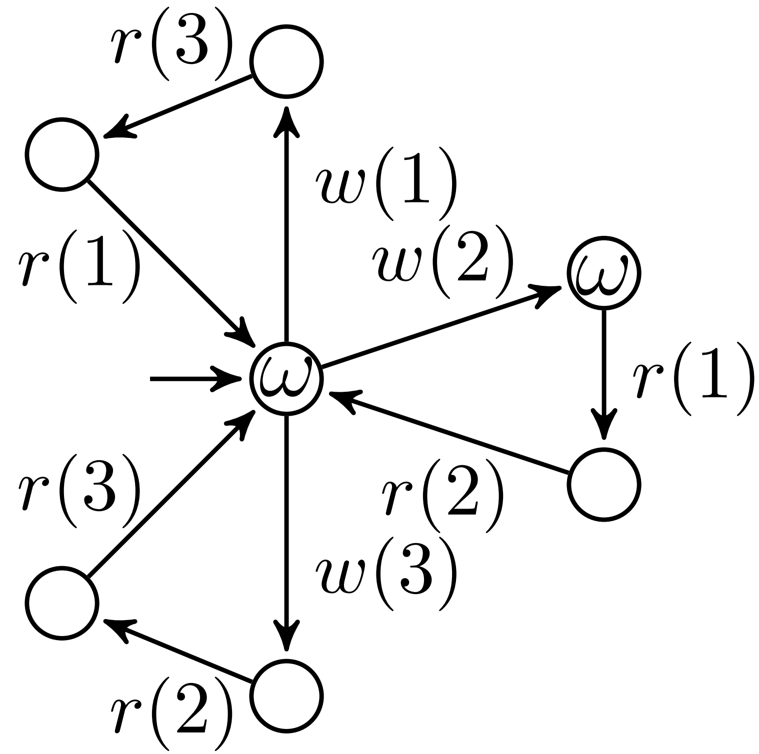


- Headcount of contributors not needed: $\bullet \to \omega$

Finite abstraction

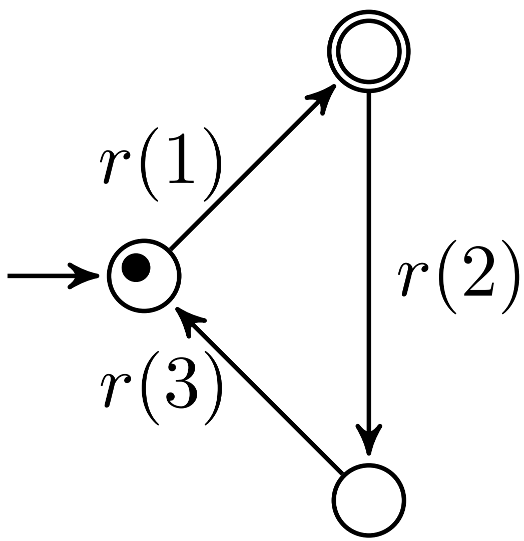


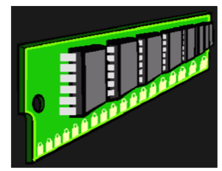


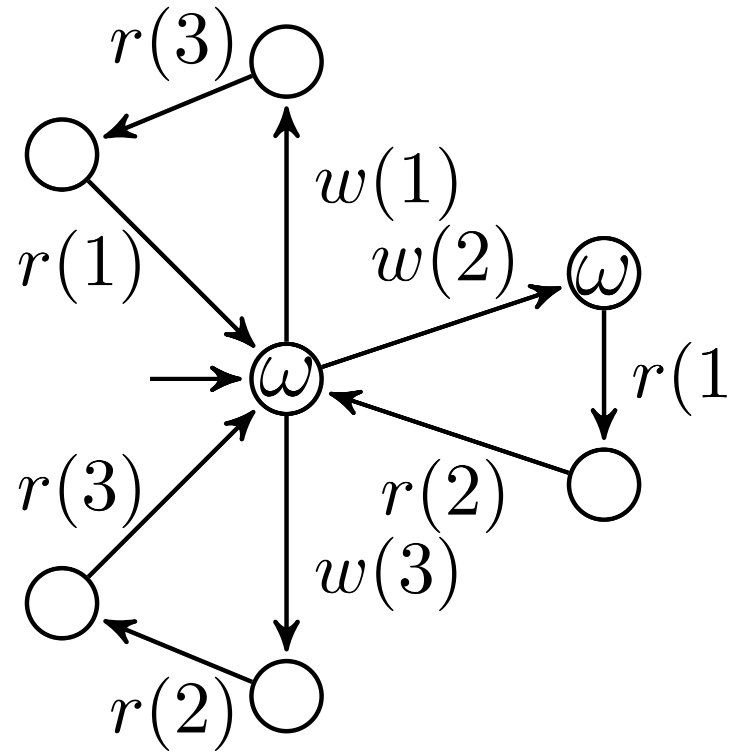


- Headcount of contributors not needed: $\bullet \to \omega$
- Never delete an ω (Copycat lemma)

Finite abstraction







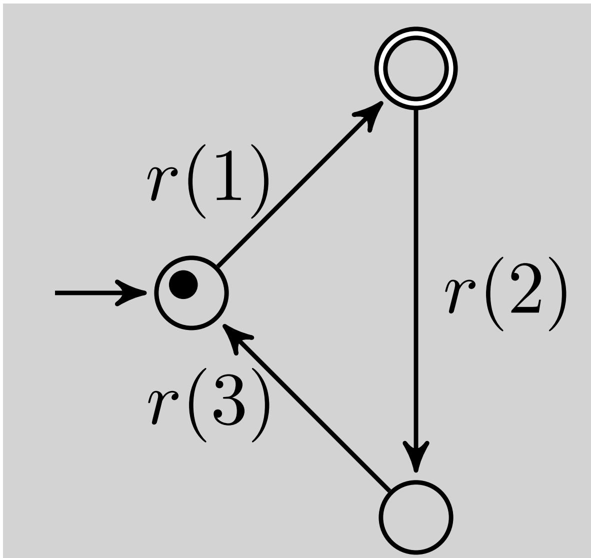
- Headcount of contributors not needed: $\bullet \to \omega$
- Never delete an ω (Copycat lemma)

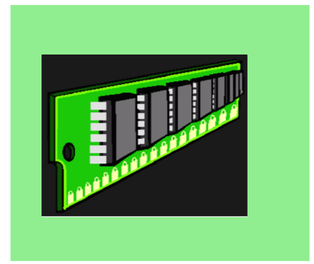
Build a graph αTS whose nodes reads as:

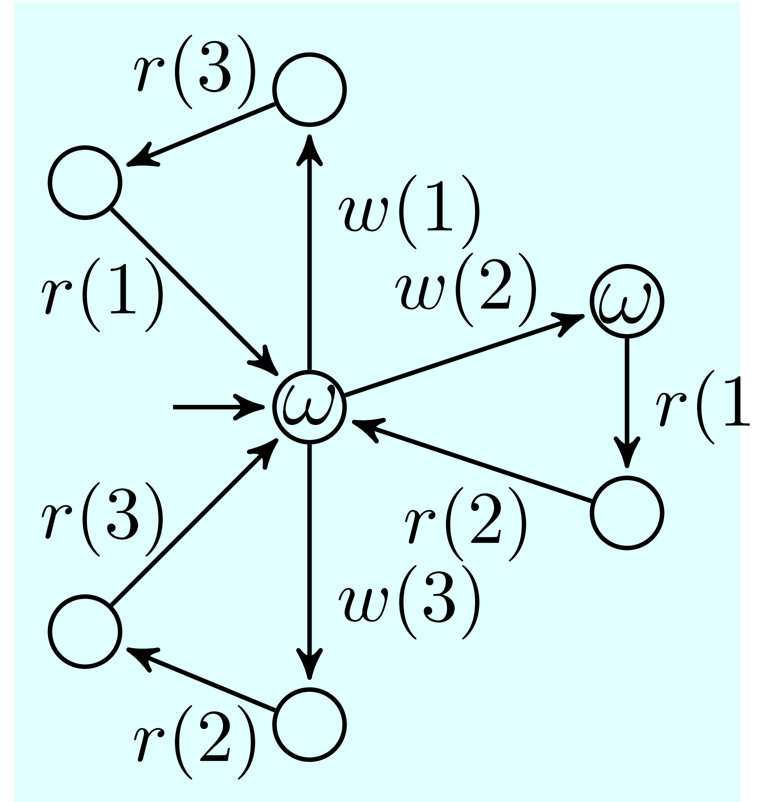
$\langle$ leader state , store value , { contribs states with ω }$\rangle$

edges are defined following leader or contributors moves

Finite abstraction







- Headcount of contributors not needed: $\bullet \to \omega$
- Never delete an ω (Copycat lemma)

Build a graph αTS whose nodes reads as:

$\langle$ leader state , store value , { contribs states with ω }$\rangle$

edges are defined following leader or contributors moves

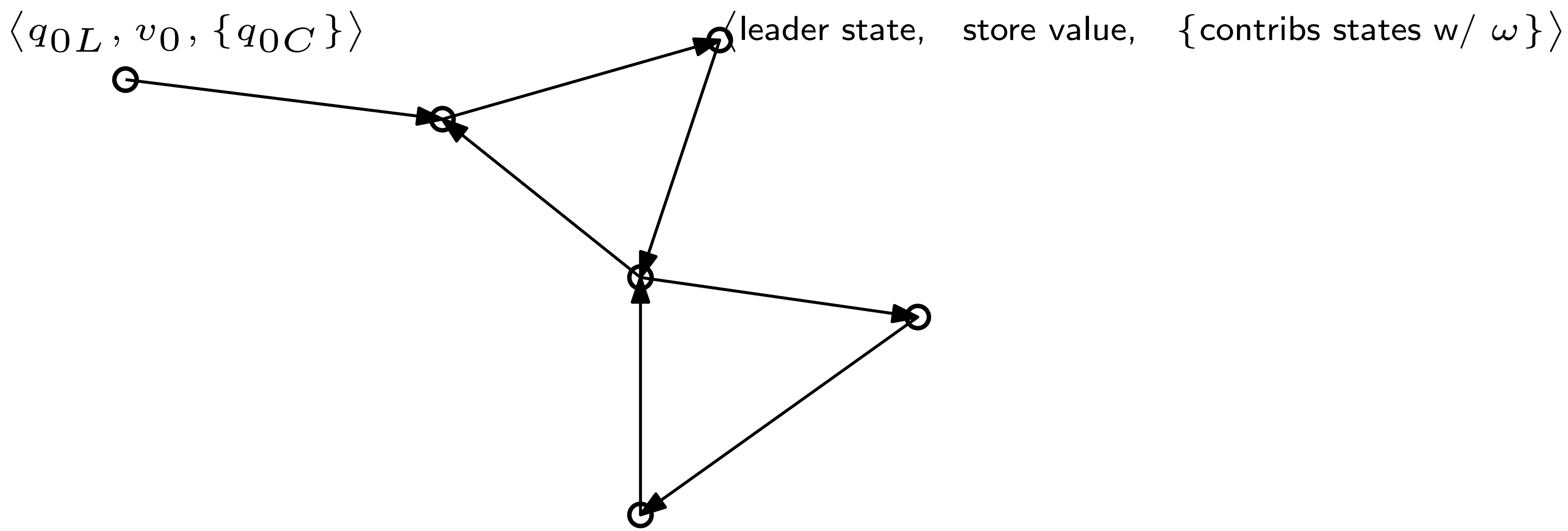
Decision procedure

Is there an infinite accepting run with finitely many processes?

Decision procedure

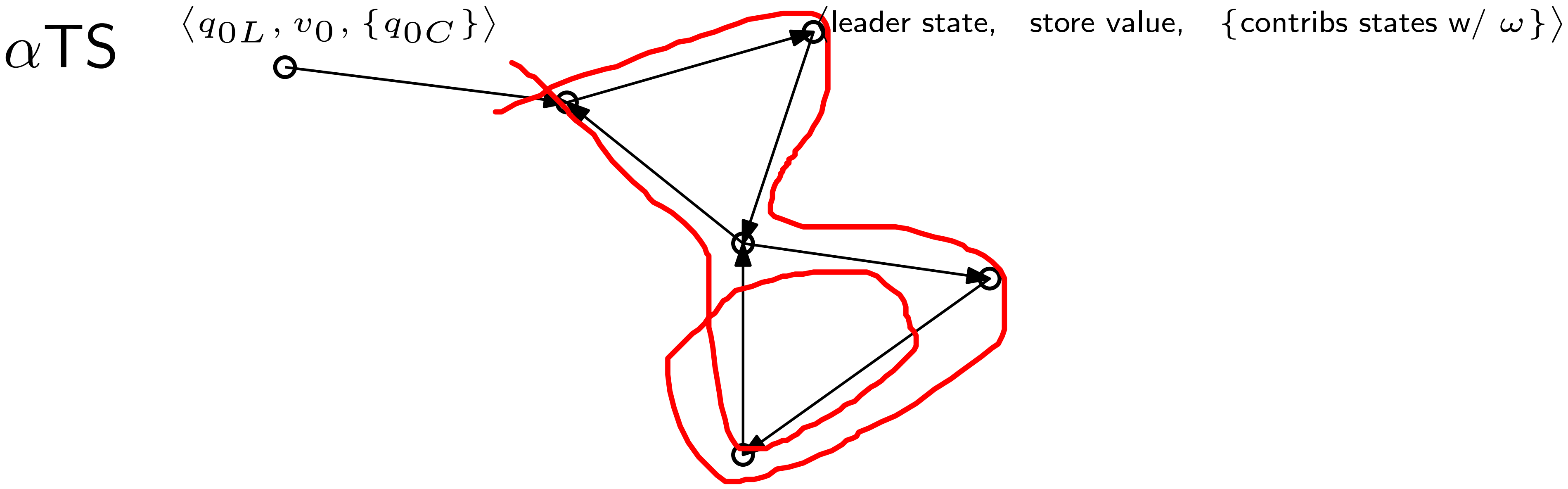
Is there an infinite accepting run with finitely many processes?

αTS



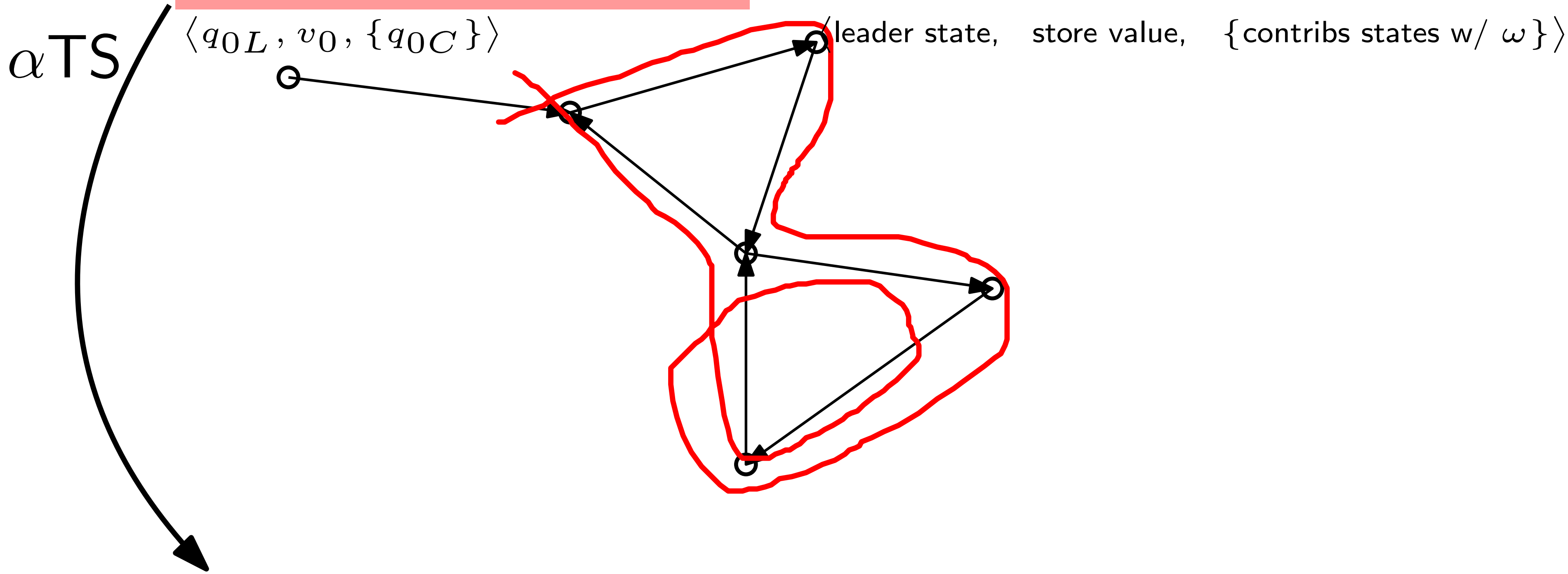
Decision procedure

Is there an infinite accepting run with finitely many processes?



Decision procedure

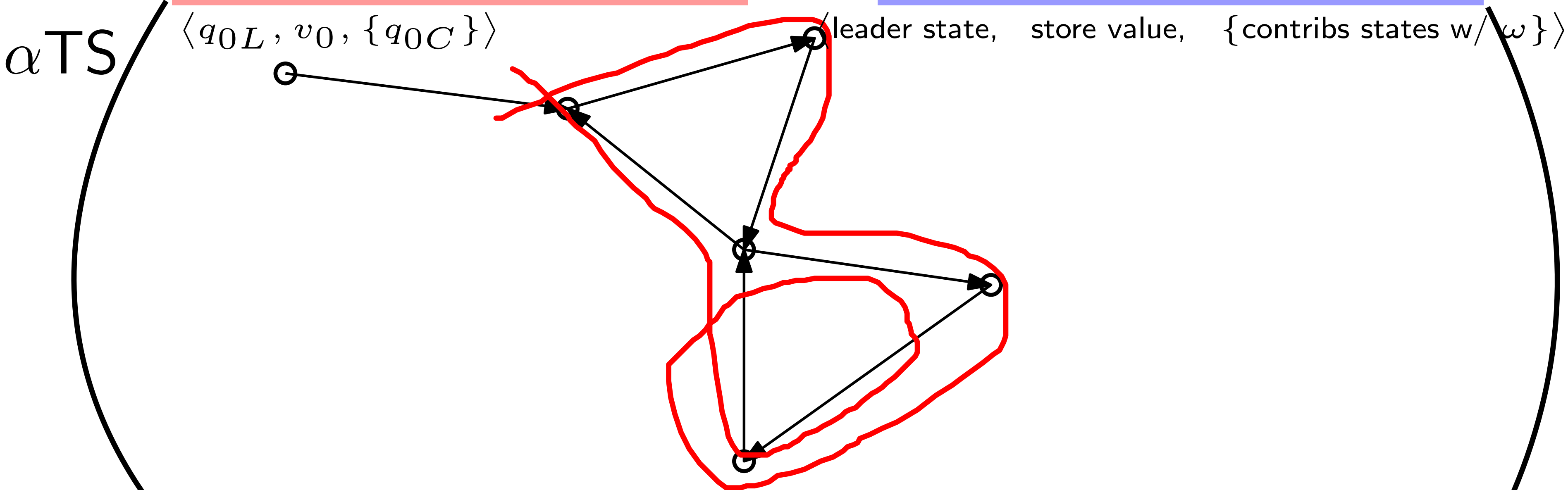
Is there an infinite accepting run with finitely many processes?



$\exists$ reachable cycle $\cap$ Buchi states $\neq \emptyset$

Decision procedure

Is there an infinite accepting run with finitely many processes?



$\exists$ reachable cycle $\cap$ Buchi states $\neq \emptyset$

each contributor involved makes a roundtrip

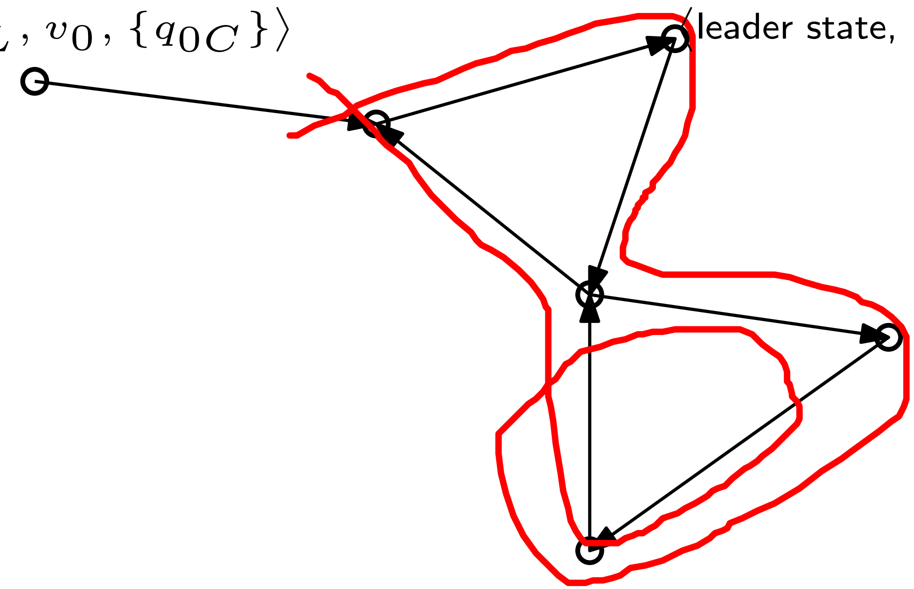
iff

the cycle has $\vec{0}$-weight

Decision procedure

Is there an infinite accepting run with finitely many processes?

αTS



Has αTS a reachable $\vec{0}$-weight cycle visiting Buchi states?

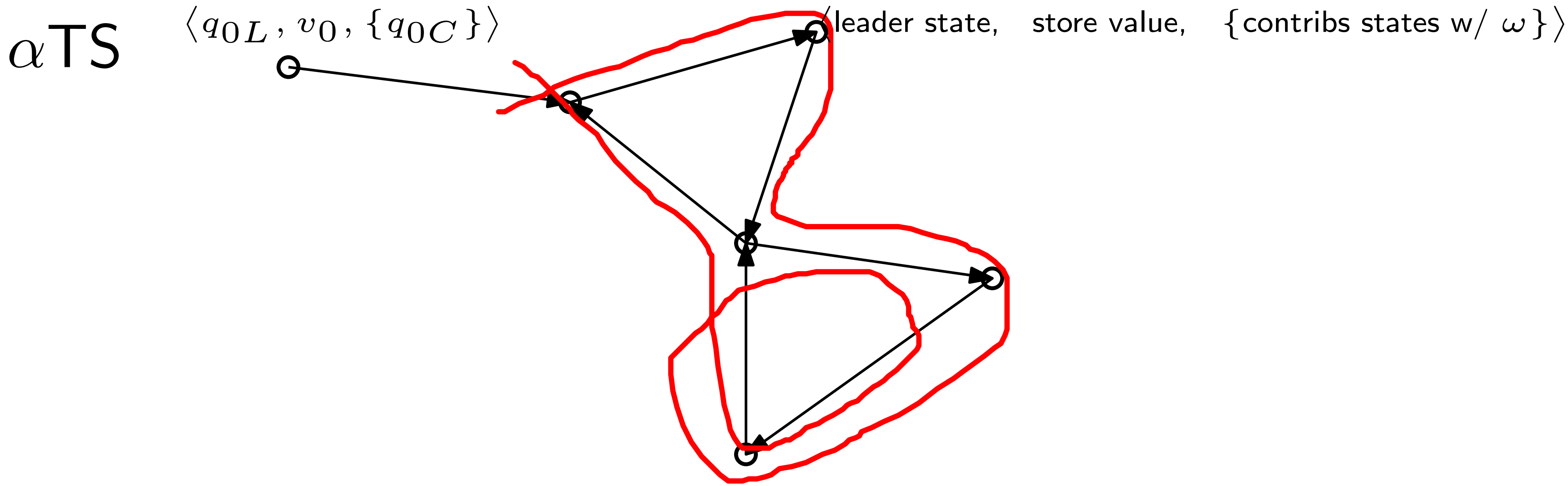
$\Downarrow$

Satisfibility of an existential Presburger formula, in NP

$$\Omega^{\circlearrowleft}(x_{t_1}, \ldots, x_{t_n}) \wedge \bigwedge_{q_c \in Q_c} \left(\sum_{tgt(t)=q_c} x_t = \sum_{src(t)=q_c} x_t \right)$$

Decision procedure

Is there an infinite accepting run with finitely many processes?



Has αTS a reachable $\vec{0}$-weight cycle visiting Buchi states ?

$\Downarrow$

Satisfibility of an existential Presburger formula, in NP

$$\Omega^{\circlearrowleft}(x_{t_1}, \ldots, x_{t_n}) \wedge \bigwedge_{q_c \in Q_c} \left(\sum_{tgt(t)=q_c} x_t = \sum_{src(t)=q_c} x_t \right)$$

Parikh Image for finite state automaton

[Seidl et al., ICALP'04]

Wake up!

Wake up!

I did a mistake

Wake up!

I did a mistake

This is not a polynomial!

Wake up!

I did a mistake

This is not a polynomial!

There are exponentially many nodes

$\langle$ leader state, store value, $\{$ contribs states with ω $\}\rangle$

Wake up!

I did a mistake

This is not a polynomial!

There are exponentially many nodes

$\langle$ leader state , store value , $\{$ contribs states with ω $\}\rangle$

Obs: along every path of αTS, the states with ω can only grow

Step 1: compute the subgraph for a guessed growing sequence
Step 2: compute the formula and check for SAT

The NP membership in perspective

The **non-parameterized** variant is PSPACE-hard [Kozen,FOCS'77]

Arbitrarily many processes yields a **noisy** channel and the problem gets easier

The results show that **noise** can **not** be cancelled out

The NP membership in perspective

The **non-parameterized** variant is PSPACE-hard [Kozen,FOCS'77]

Arbitrarily many processes yields a **noisy** channel and the problem gets easier

The results show that **noise** can **not** be cancelled out

What if we give processes some auxiliary memory?

Processes + stacks

Processes + stacks

- To the leader only:

  remains in NP (proof generalization)

Processes + stacks

- To the leader only:

  remains in NP (proof generalization)

- To both the leader and contributor:

  in NEXPTIME by reduction to the previous case

Processes + stacks

- To the leader only:

  remains in NP (proof generalization)

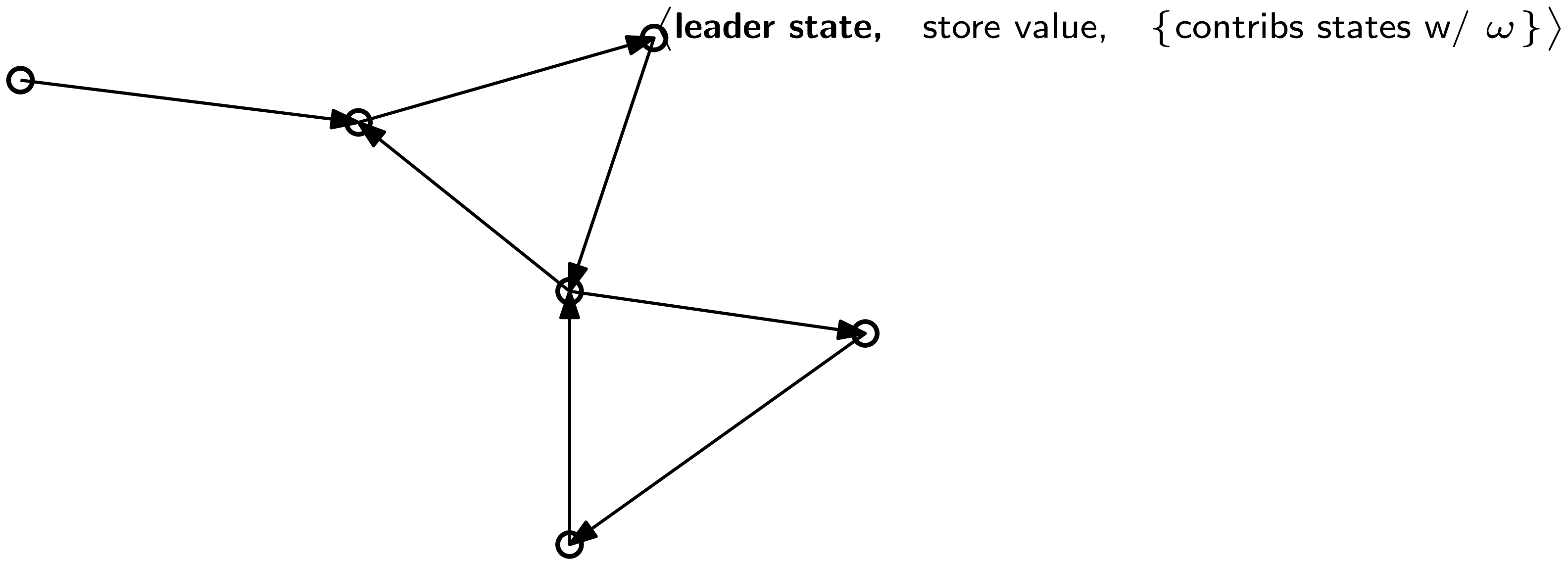
- To both the leader and contributor:

  in NEXPTIME by reduction to the previous case

Compare with the undecidability result for **fixed** number of processes **all** of them with stack

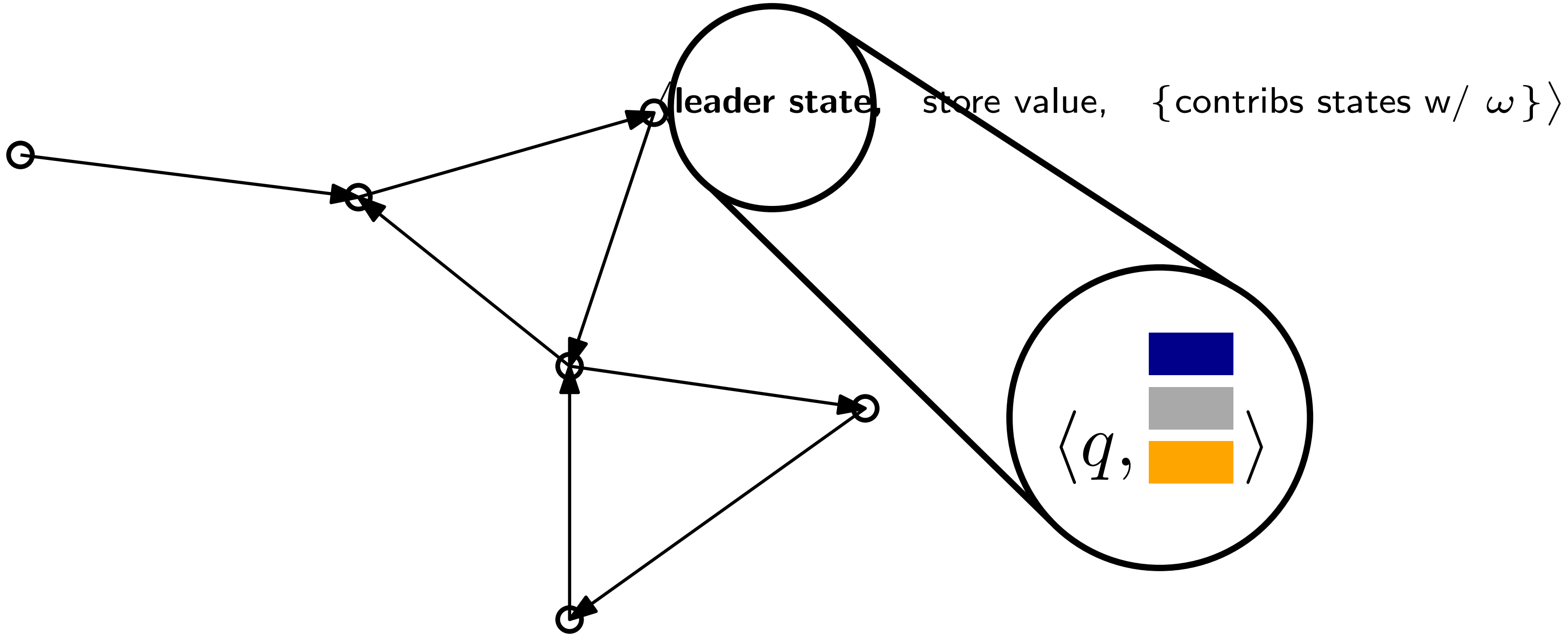
Only the leader has a stack

Is there an infinite accepting run with finitely many processes?



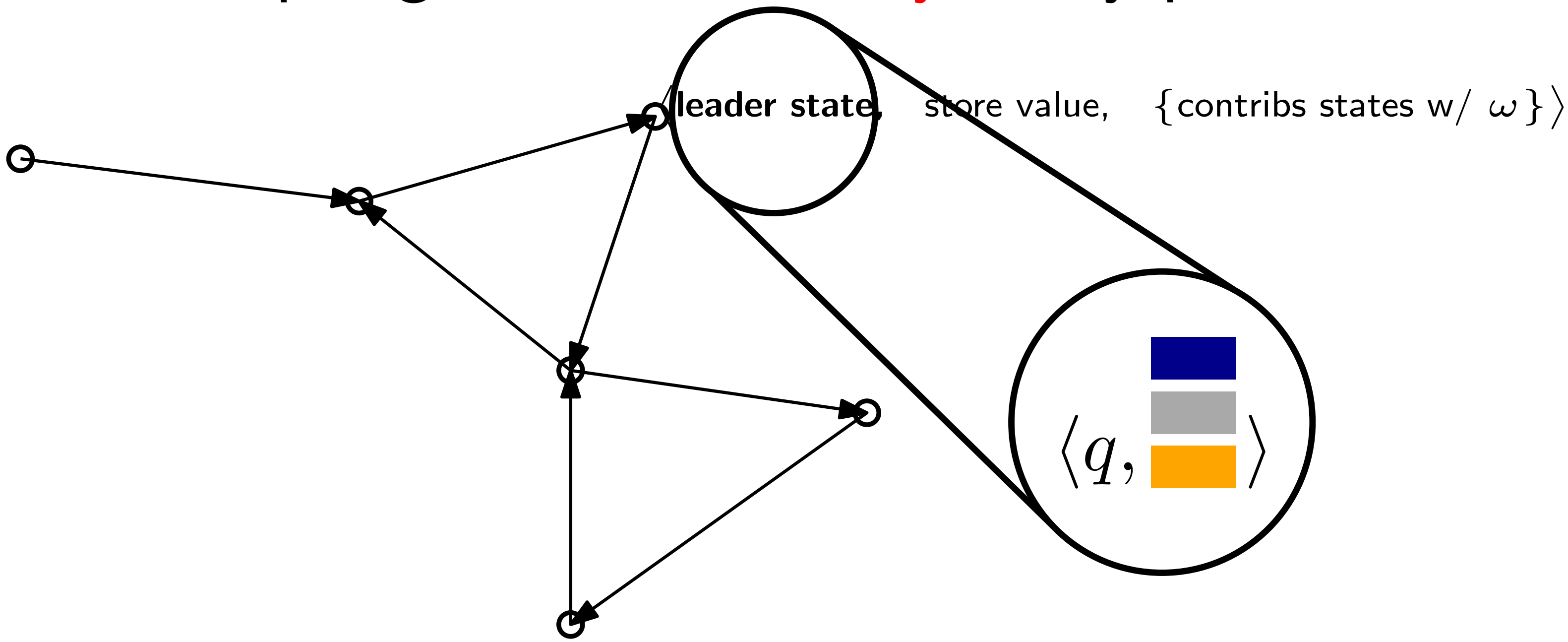
Only the leader has a stack

Is there an infinite accepting run with finitely many processes?



Only the leader has a stack

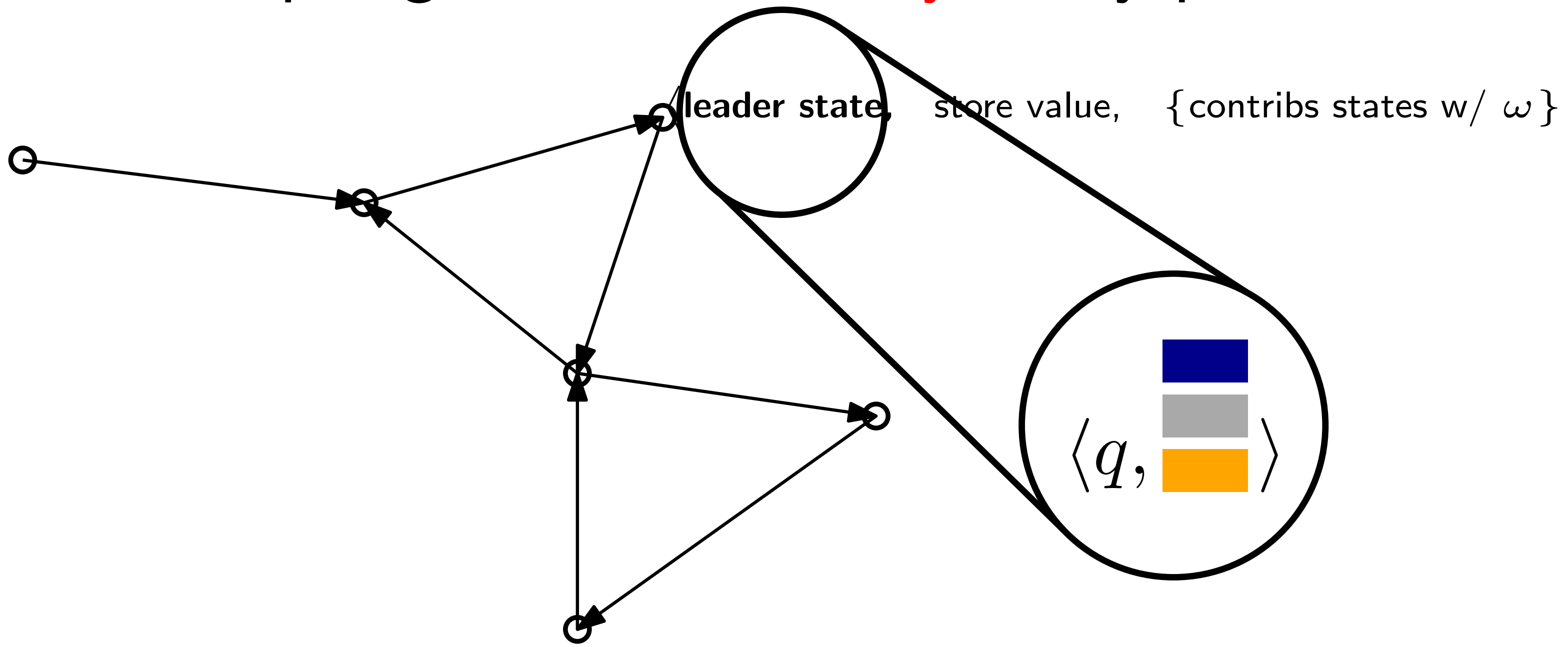
Is there an infinite accepting run with finitely many processes?



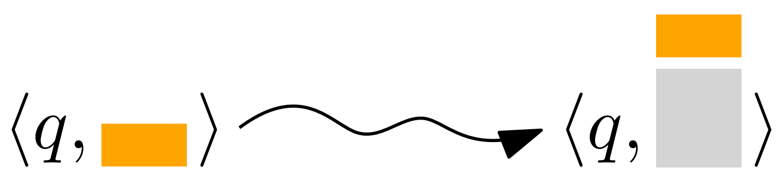
Has αTS a reachable $\vec{0}$-weight cycle visiting Buchi states ?

Only the leader has a stack

Is there an infinite accepting run with finitely many processes?

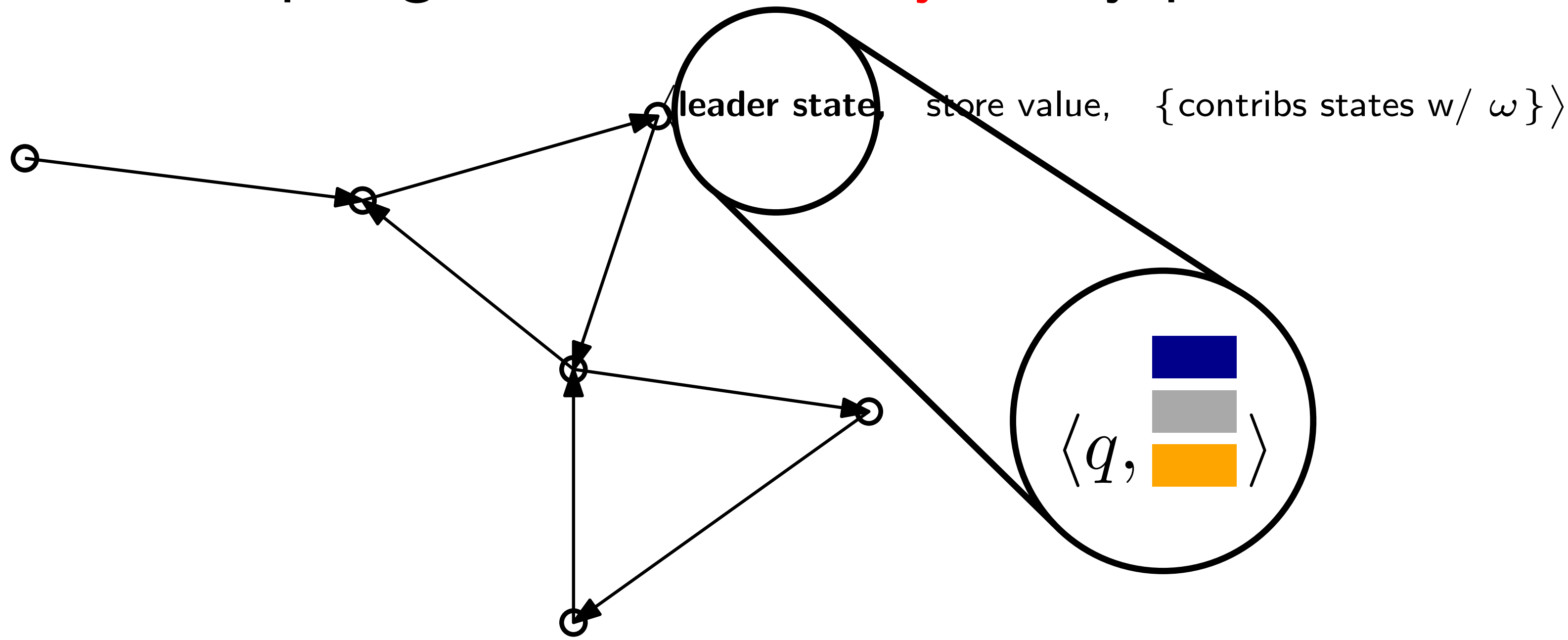


Has αTS a reachable $\vec{0}$-weight cycle visiting Buchi states?

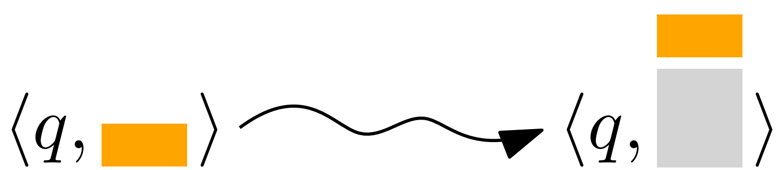


Only the leader has a stack

Is there an infinite accepting run with finitely many processes?



Has αTS a reachable $\vec{0}$-weight cycle visiting Buchi states ?



$$\Omega^{\circlearrowleft}(x_{t_1}, \dots, x_{t_n}) \wedge \bigwedge_{q_c \in Q_c} \left(\sum_{tgt(t)=q_c} x_t = \sum_{src(t)=q_c} x_t \right)$$

Parikh Image for context-free grammars

[Seidl et al., CADE'05][Esparza, Fun. Informatica'97]

Everybody has a stack

Is there an infinite accepting run with finitely many processes?

Everybody has a stack

Is there an infinite accepting run with finitely many processes?

NEXPTIME

by reduction to the “only the leader has a stack” case

Everybody has a stack

Is there an infinite accepting run with finitely many processes?

NEXPTIME

by reduction to the “only the leader has a stack” case

Compare with the undecidability result for **fixed** number of processes **all** of them with stack

Everybody has a stack

Is there an infinite accepting run with finitely many processes?

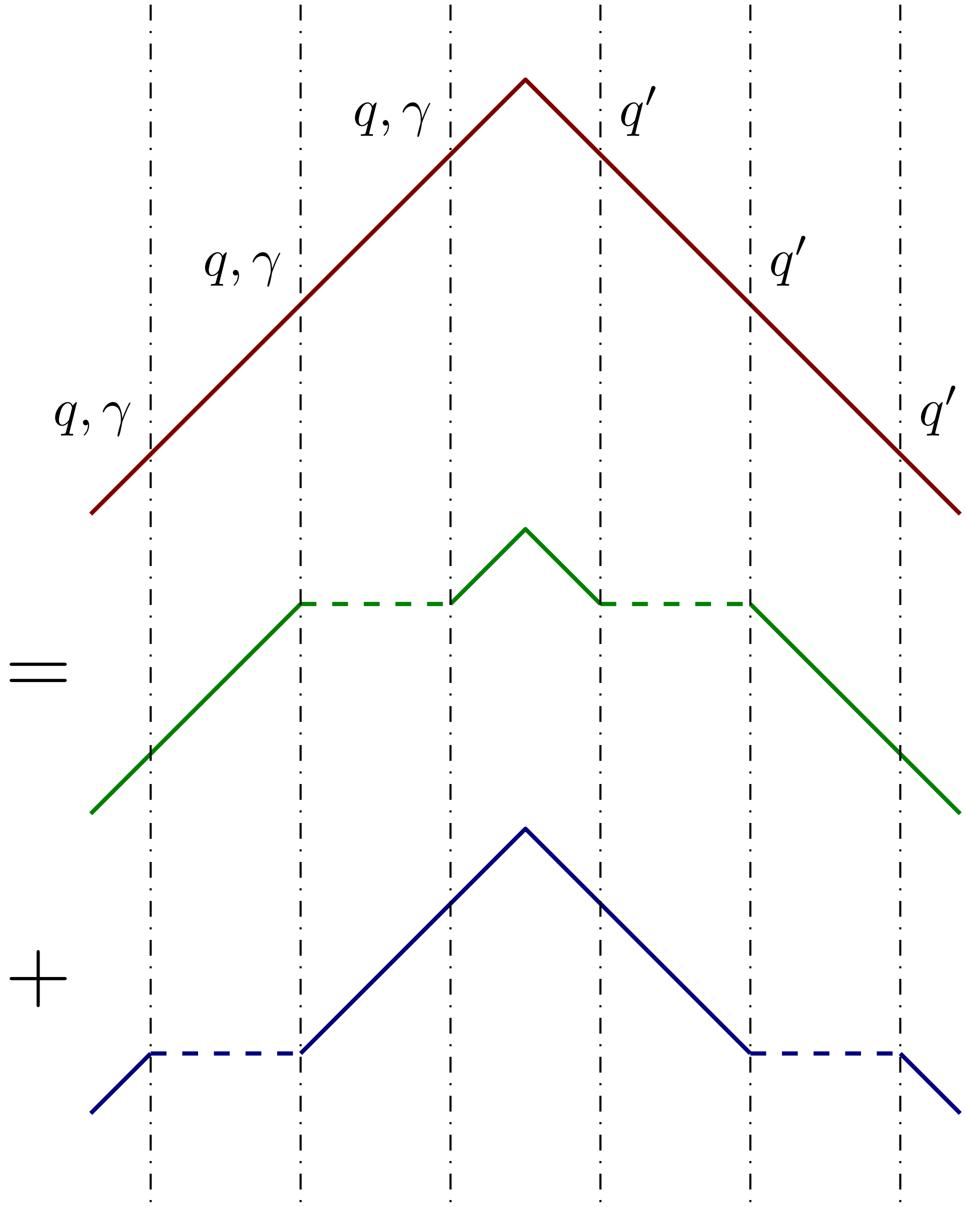
NEXPTIME

by reduction to the “only the leader has a stack” case

Compare with the undecidability result for **fixed** number of processes **all** of them with stack

Proof of correctness is not simple or easy

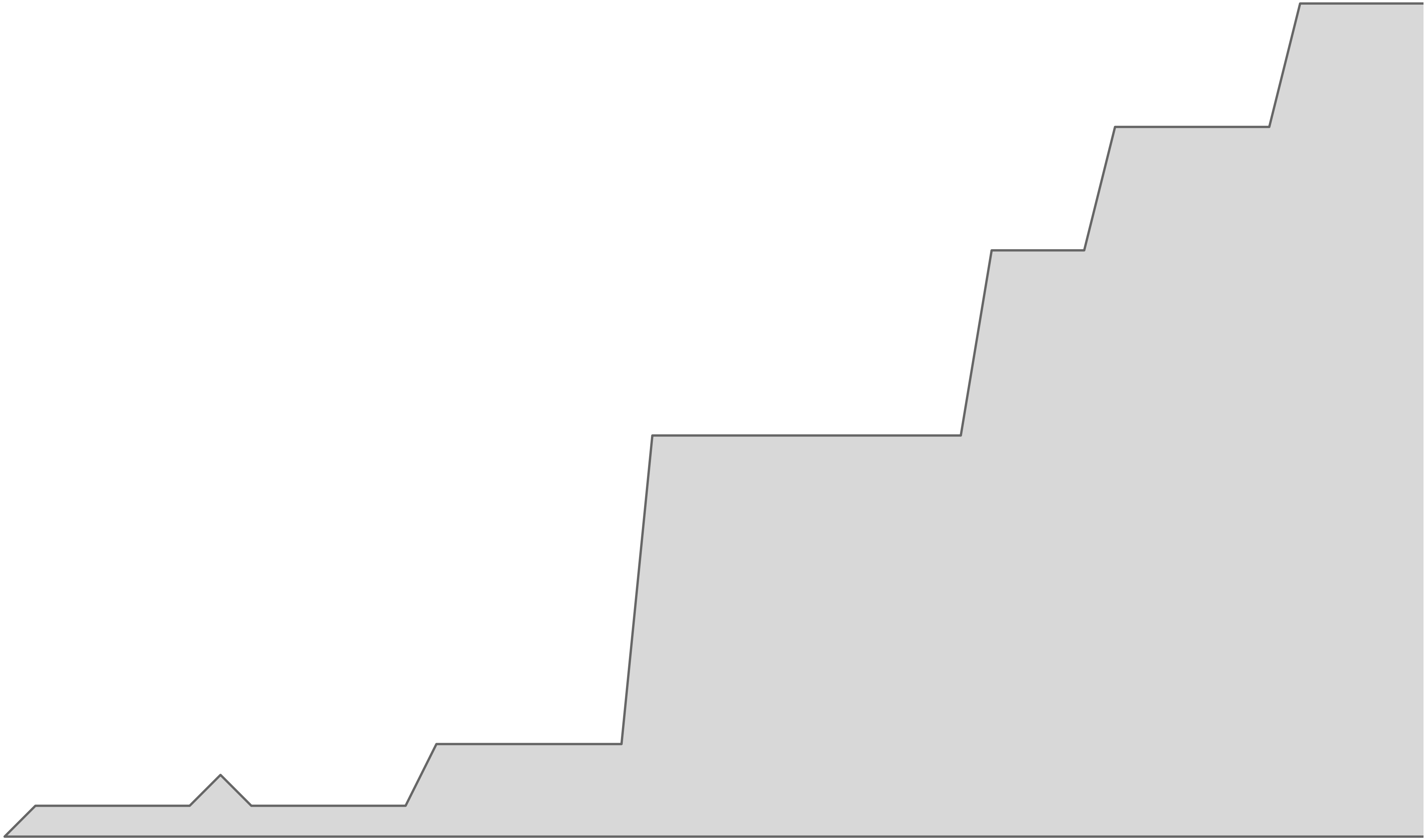
Replace a contributor run by **more** runs using "less memory"



Less memory = retrieve shallow stack frames only

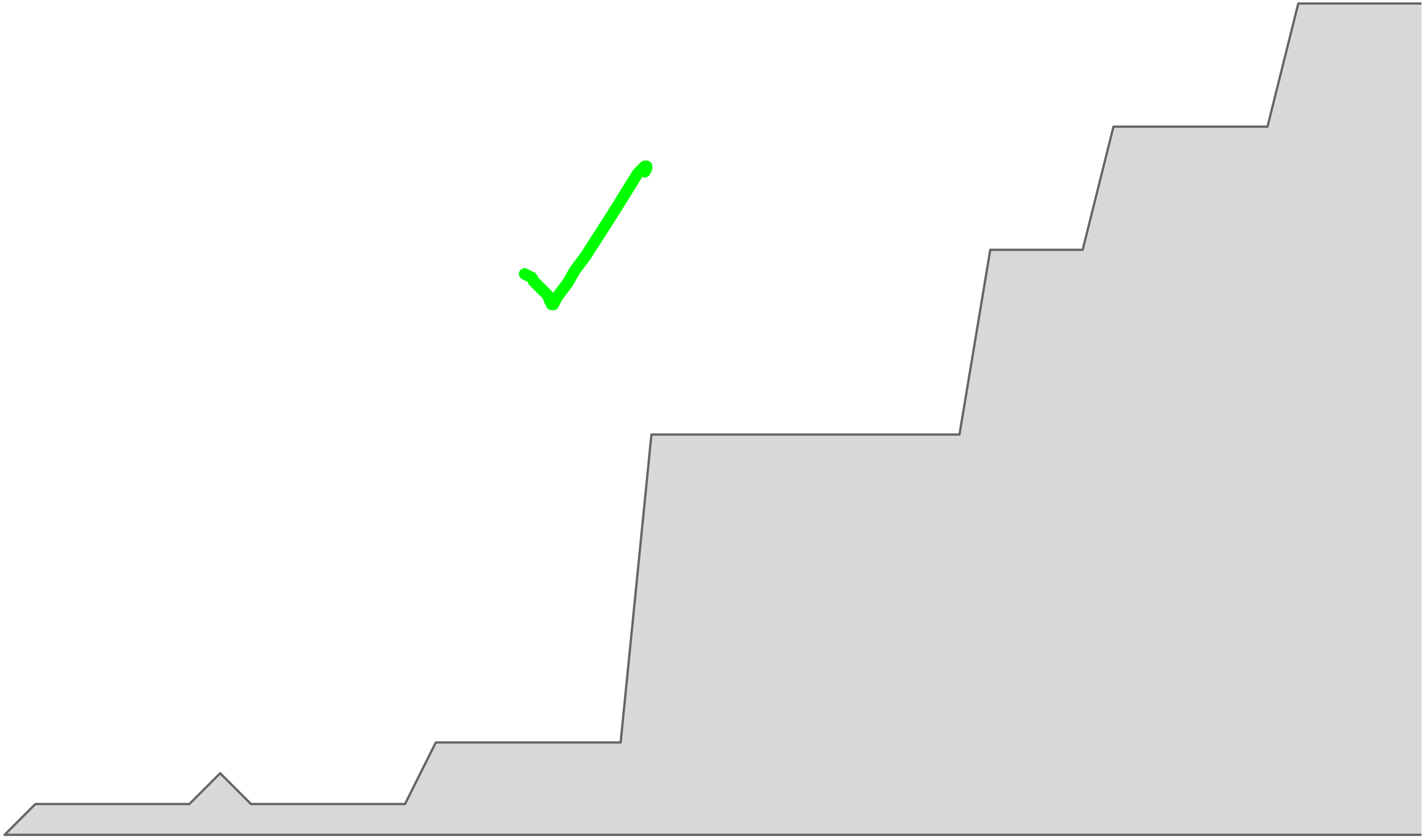
Less memory = retrieve shallow stack frames only

- Examples



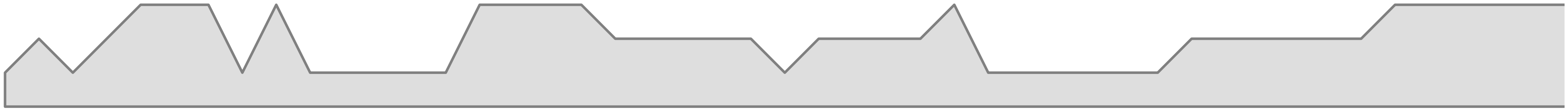
Less memory = retrieve shallow stack frames only

- Examples



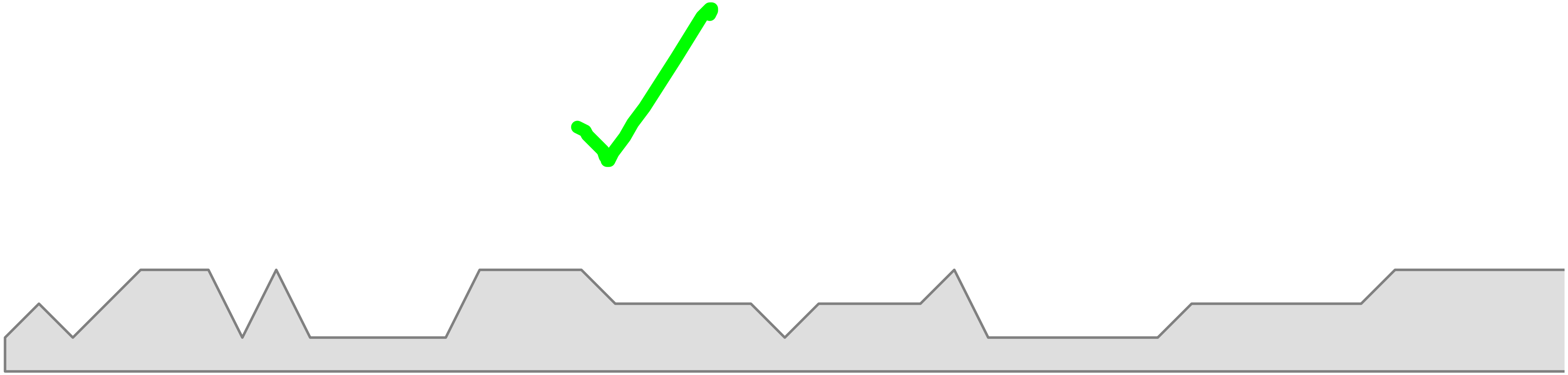
Less memory = retrieve shallow stack frames only

- Examples



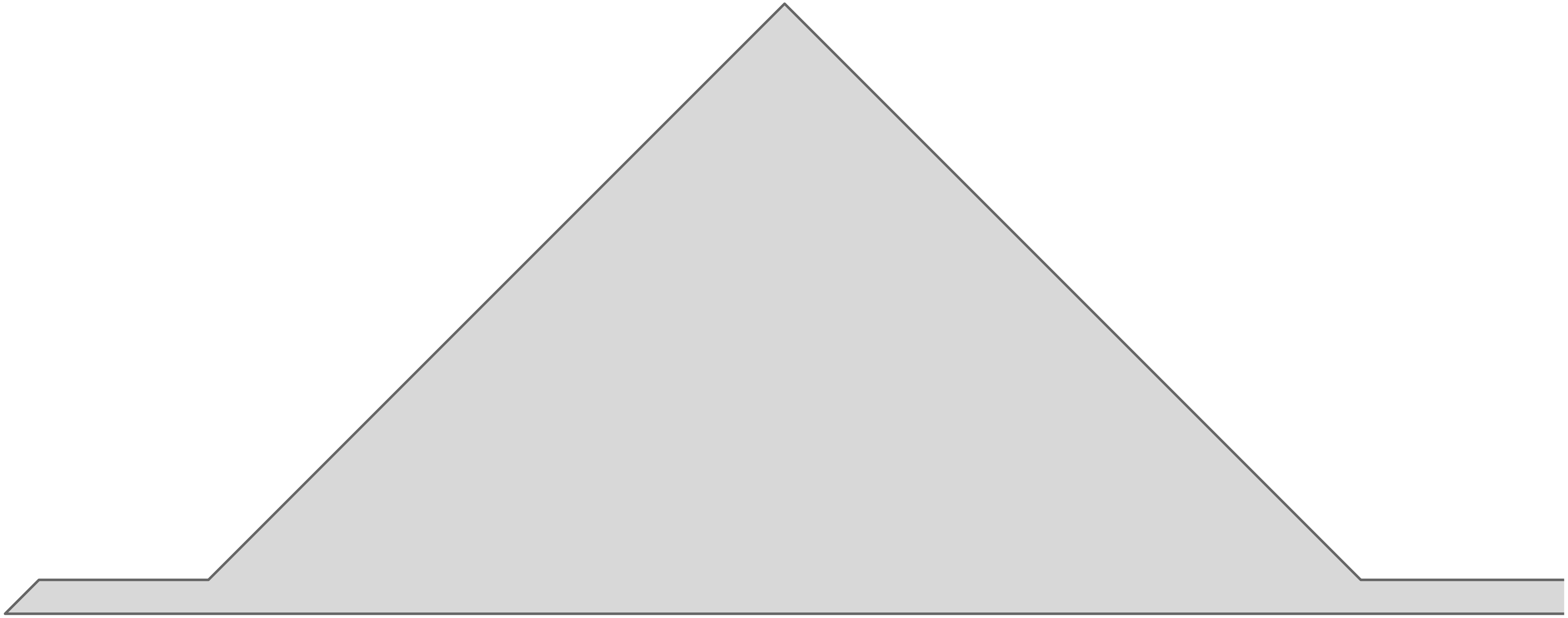
Less memory = retrieve shallow stack frames only

- Examples



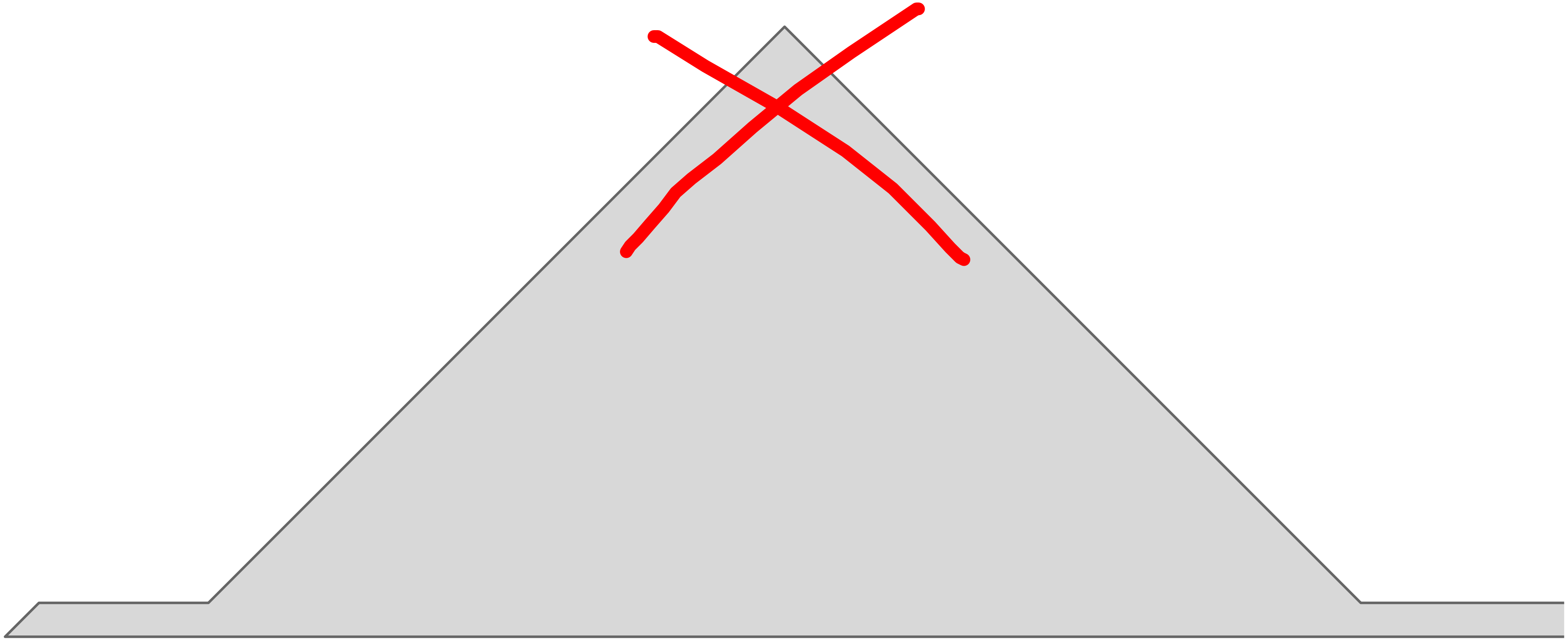
Less memory = retrieve shallow stack frames only

- Examples



Less memory = retrieve shallow stack frames only

- Examples



Less memory = retrieve shallow stack frames only

- having each contrib keeping track of its topmost $O(n^3)$ frames suffice

- Memory is now bounded and hard-encoded into the states

- Fall back onto the “only the leader has a stack” case

Conclusions

Safety checking (CAV'13)

	FSM	PDM
FSM	coNP-c	coNP-c
PDM	coNP-c	PSPACE-c

Liveness checking (CAV'15)

	FSM	PDM
FSM	NP-c	NP-c
PDM		PSPACE-hard ? NEXPTIME

Liveness is as easy as safety

- Contrast: broadcast protocols

Parameterized verification is easier than non-parameterized

- Contrast: n finite state machines

Adding memory leads to harder, but not undecidable, problems

- Contrast: fixed pushdown systems

Conclusions

Safety checking (CAV'13)

	FSM	PDM
FSM	coNP-c	coNP-c
PDM	coNP-c	PSPACE-c

Liveness checking (CAV'15)

	FSM	PDM
FSM	NP-c	NP-c
PDM		PSPACE-hard ? NEXPTIME

Liveness is as easy as safety

- Contrast: broadcast protocols

Parameterized verification is easier than non-parameterized

- Contrast: n finite state machines

Adding memory leads to harder, but not undecidable, problems

- Contrast: fixed pushdown systems