Parameterized verification of round-based shared-memory systems

Nicolas Waldburger [1]
Nathalie Bertrand [1], Nicolas Markey [1], Ocan Sankur [1]

[1]Univ Rennes, Inria, CNRS, IRISA, France

Round-based shared-memory algorithms

The distributed systems considered

- Parallel, identical processes communicating via shared memory
- Asynchrony: some processes might be faster than others
- Non-atomic reads/writes, no fault
- Round-based: each process has its own round number k, which only increases; each round has its own set of registers.

Round-based shared-memory algorithms

The distributed systems considered

- Parallel, identical processes communicating via shared memory
- Asynchrony: some processes might be faster than others
- Non-atomic reads/writes, no fault
- Round-based: each process has its own round number k, which only increases; each round has its own set of registers.

The binary consensus problem

Make all processes agree on a common value, each process having an initial preference p. Desired properties of consensus algorithms:

Validity : If a process decides value p, some process started with preference p.

Agreement : Two processes that decide decide of the same value.

Termination : All processes eventually decide of a value.

Consensus with shared memory is difficult: there is no wait-free consensus protocol with shared memory and two processes.

A motivating example: Aspnes' consensus algorithm

int $k := 0$, bool $p \in \{0,1\}$, $(\mathrm{rg}_b[r])_{b\in\{0,1\},r\in\mathbb{N}}$ all initialized to no;
while true **do**
  read from $\mathrm{rg}_0[k]$ and $\mathrm{rg}_1[k]$;
  if $\mathrm{rg}_0[k] =$ yes and $\mathrm{rg}_1[k] =$ no **then** $p := 0$;
  else if $\mathrm{rg}_0[k] =$ no and $\mathrm{rg}_1[k] =$ yes **then** $p := 1$;
  write yes to $\mathrm{rg}_p[k]$;
  if $k > 0$ **then**
    read from $\mathrm{rg}_{1-p}[k-1]$;
    if $\mathrm{rg}_{1-p}[k-1] =$ no **then** return p;
  $k := k+1$;

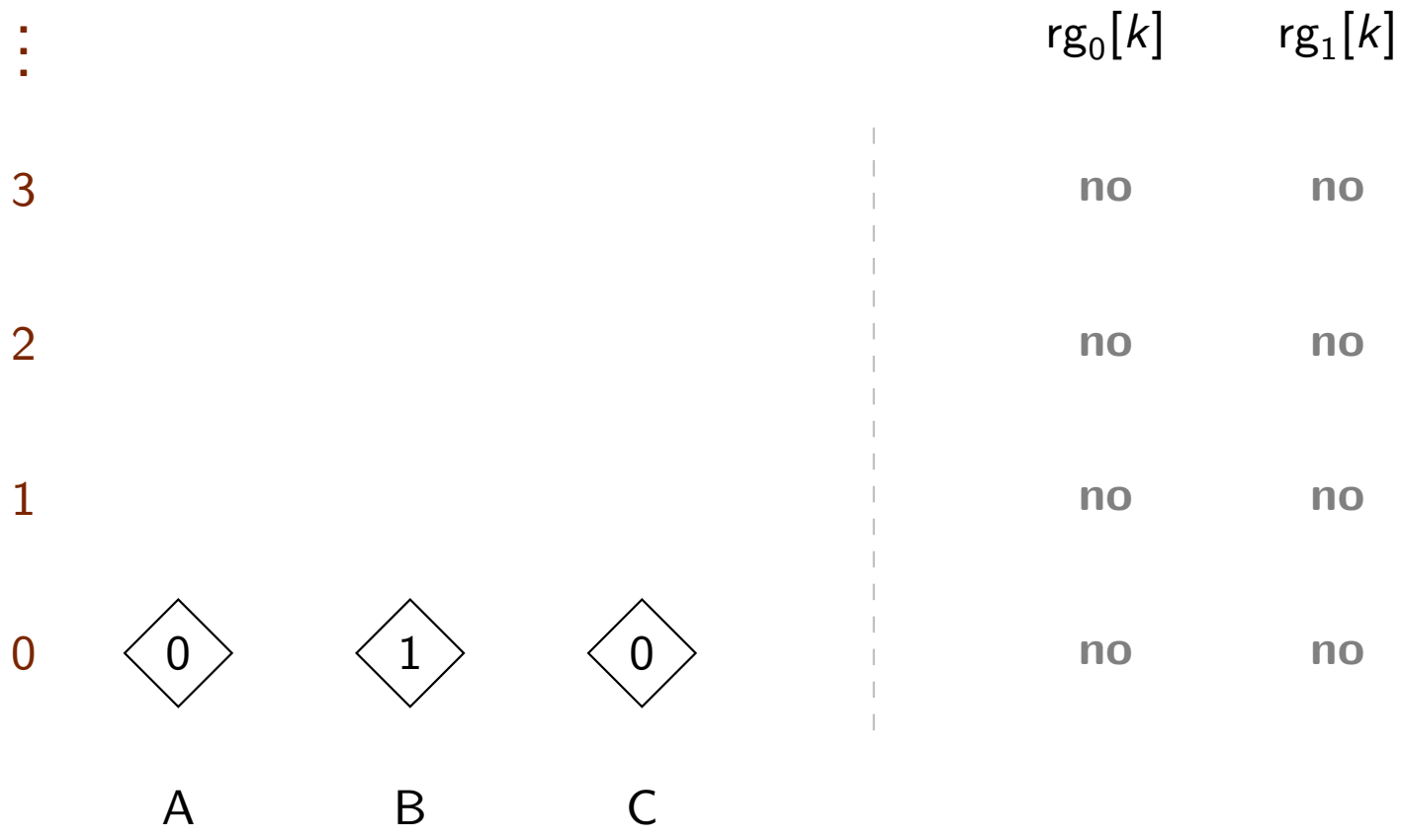
read from registers of rounds k and $k-1$

write to registers of round k

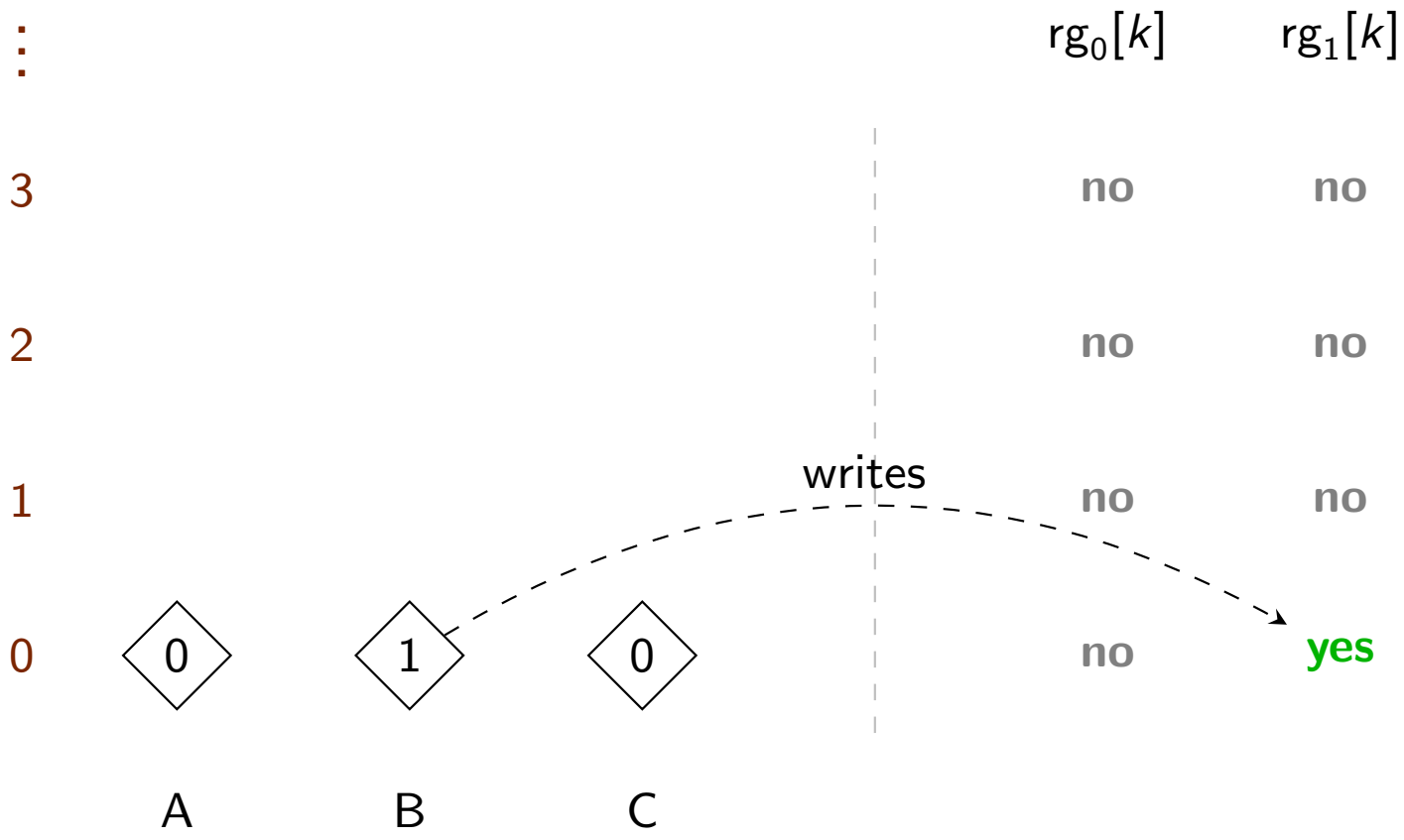
Algorithm 1: Aspnes' consensus algorithm[1].

[1] James Aspnes, Fast deterministic consensus in a noisy environment, *Journal of Algorithms*, 2002.

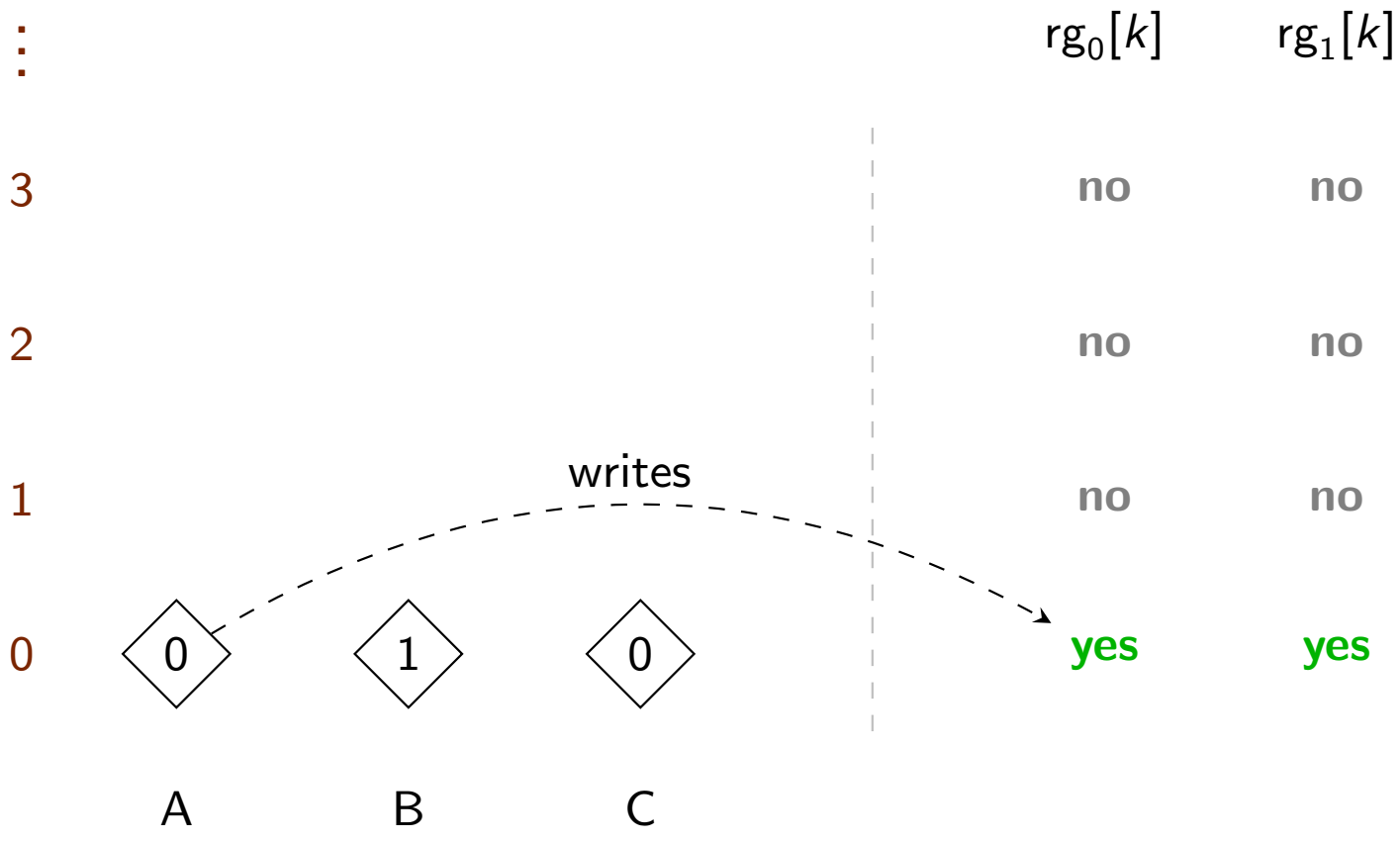
Aspnes' consensus algorithm illustrated



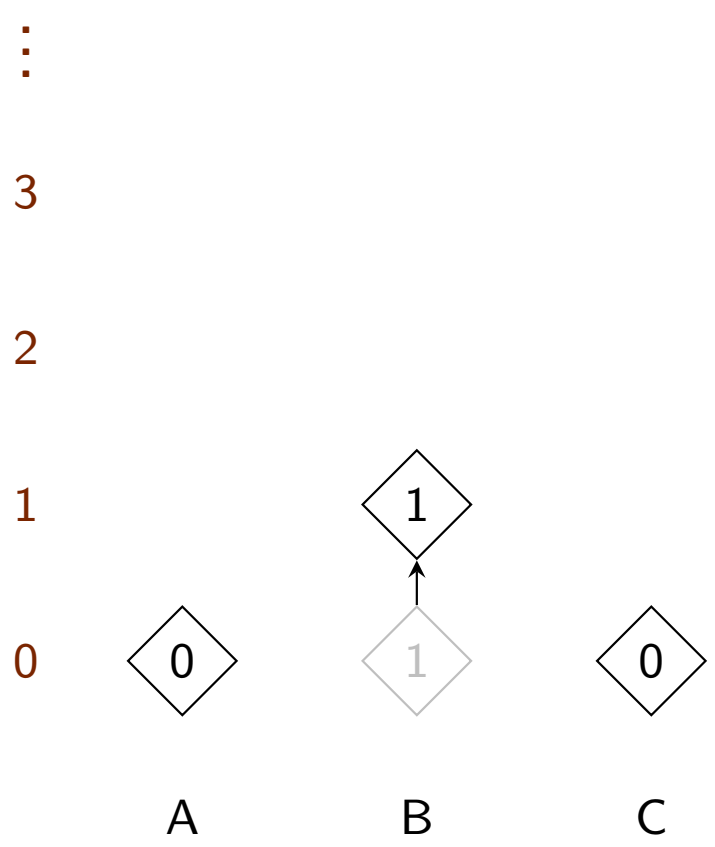
Aspnes' consensus algorithm illustrated



Aspnes' consensus algorithm illustrated

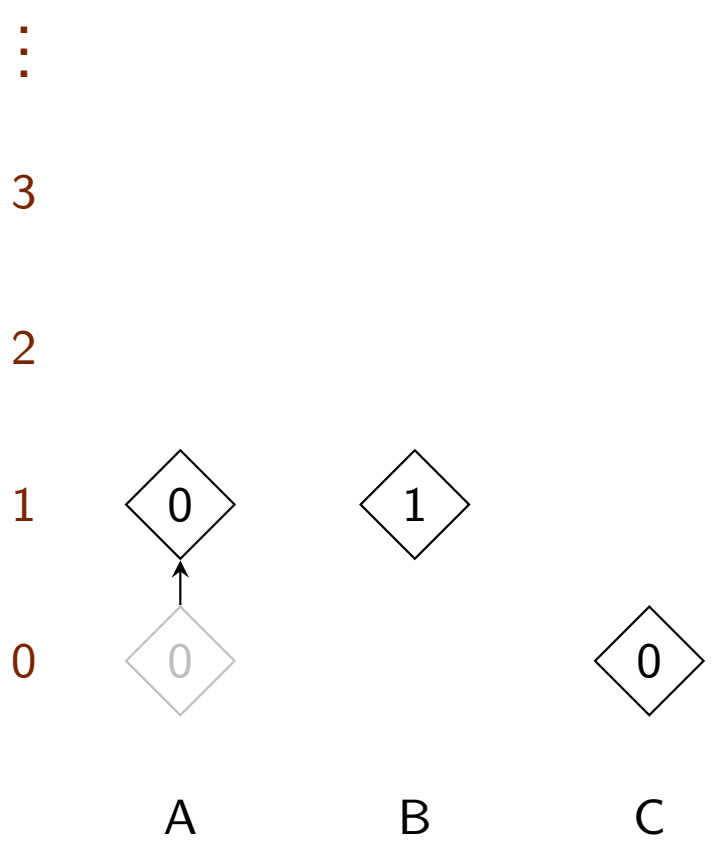


Aspnes' consensus algorithm illustrated



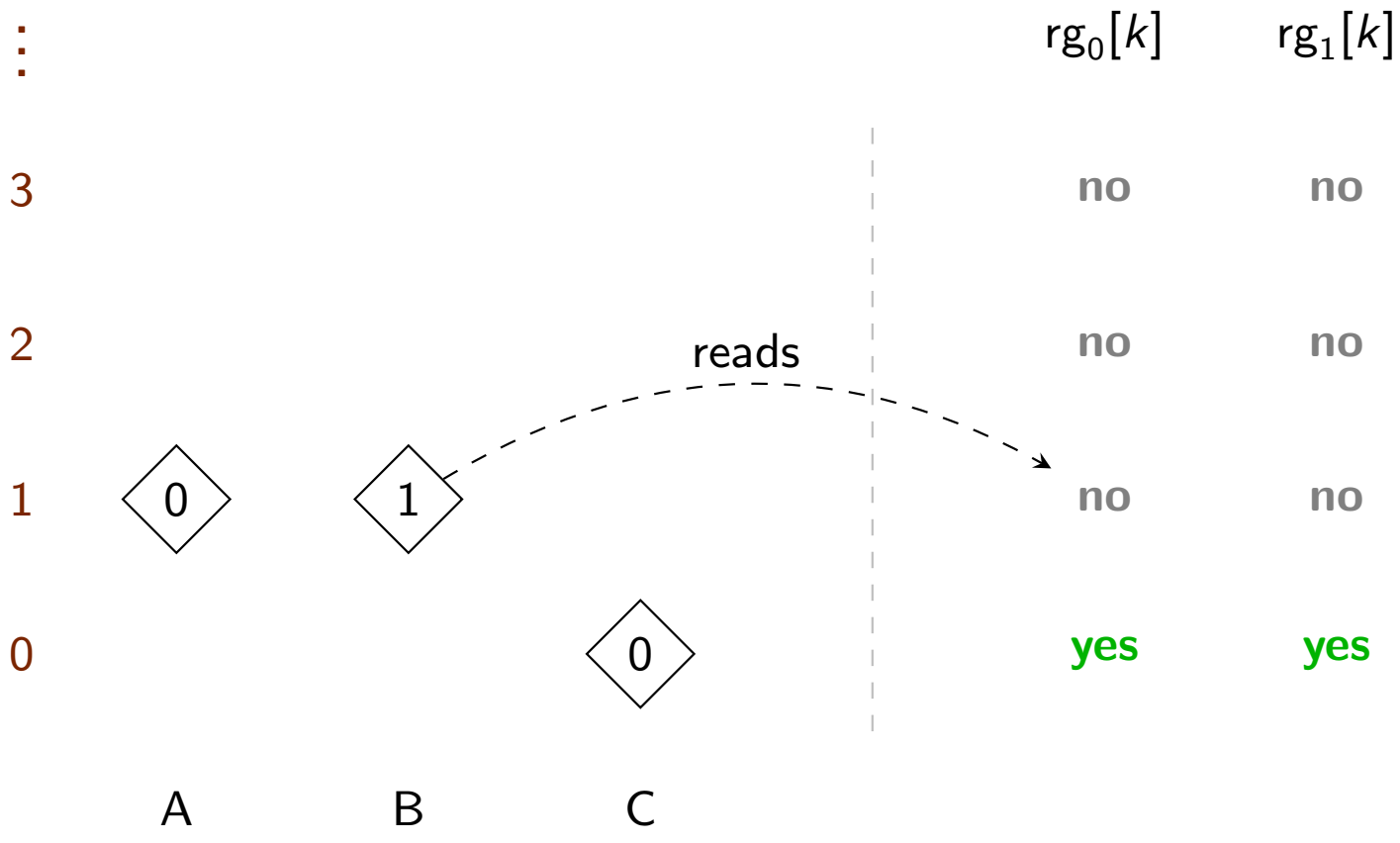
	$rg_0[k]$	$rg_1[k]$
⋮		
3	no	no
2	no	no
1	no	no
0	yes	yes

Aspnes' consensus algorithm illustrated

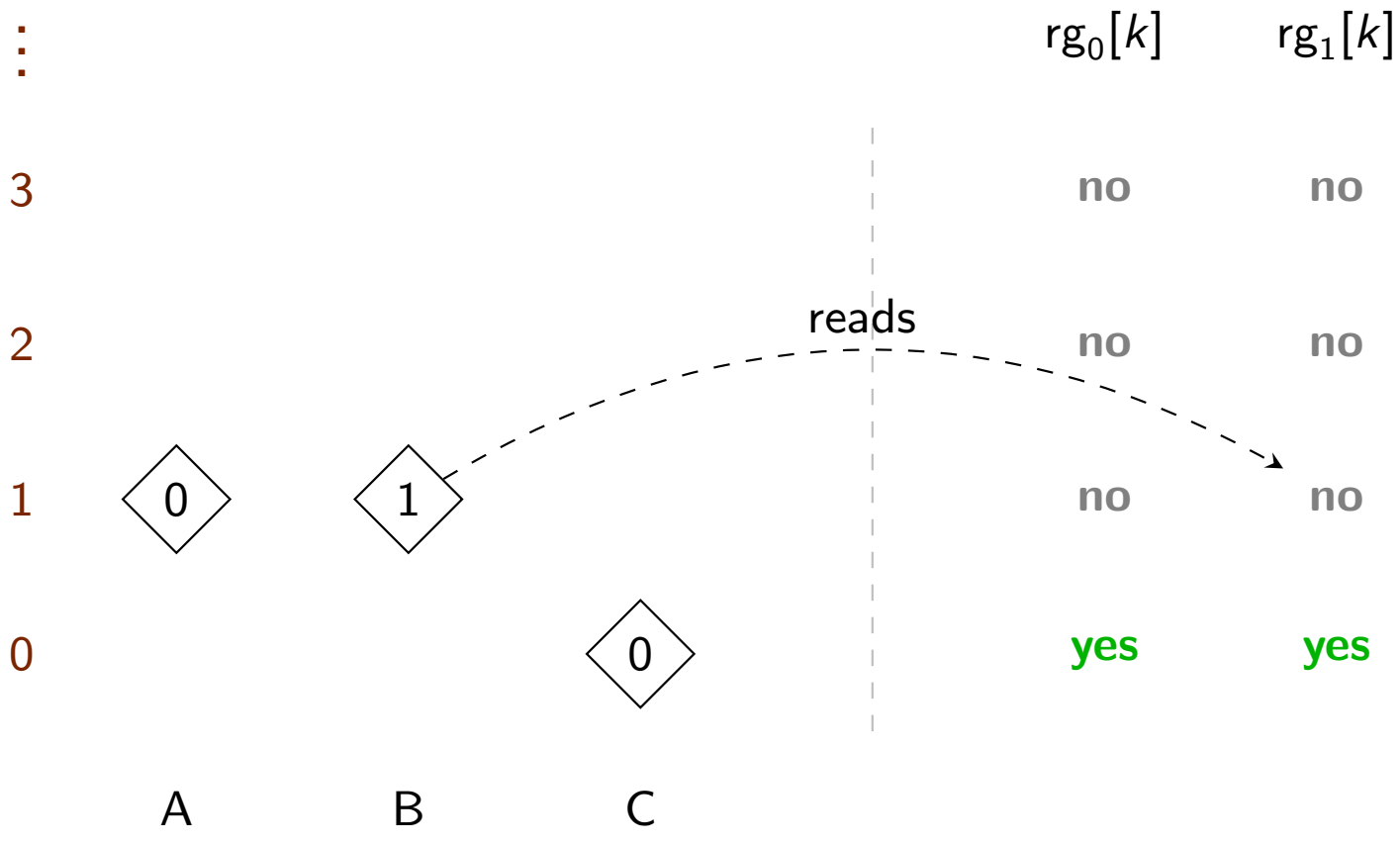


	$rg_0[k]$	$rg_1[k]$
⋮		
3	no	no
2	no	no
1	no	no
0	yes	yes

Aspnes' consensus algorithm illustrated

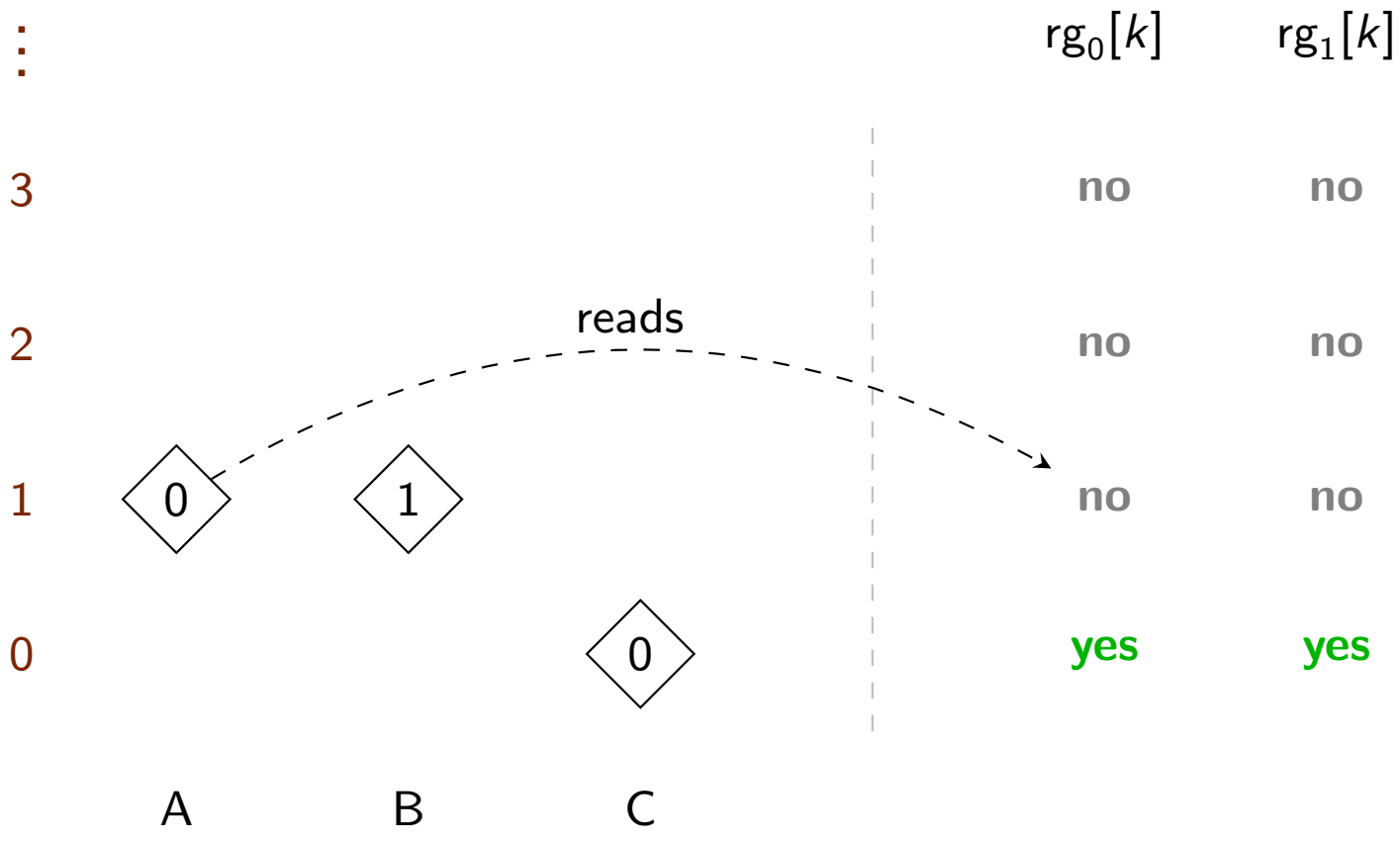


Aspnes' consensus algorithm illustrated

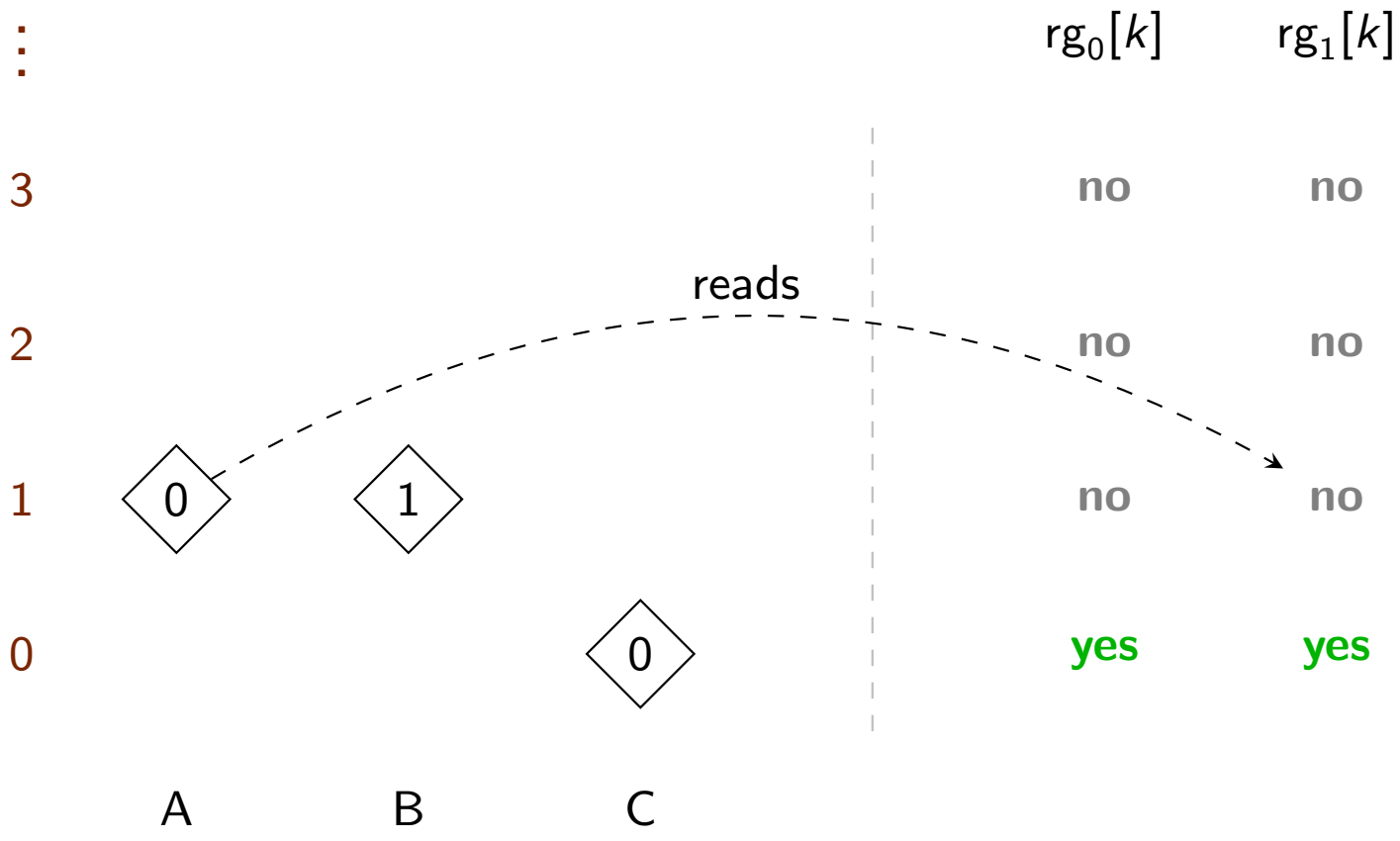


B wants to write its preference on $rg_1[k]$
Non-atomic: A may move before B writes

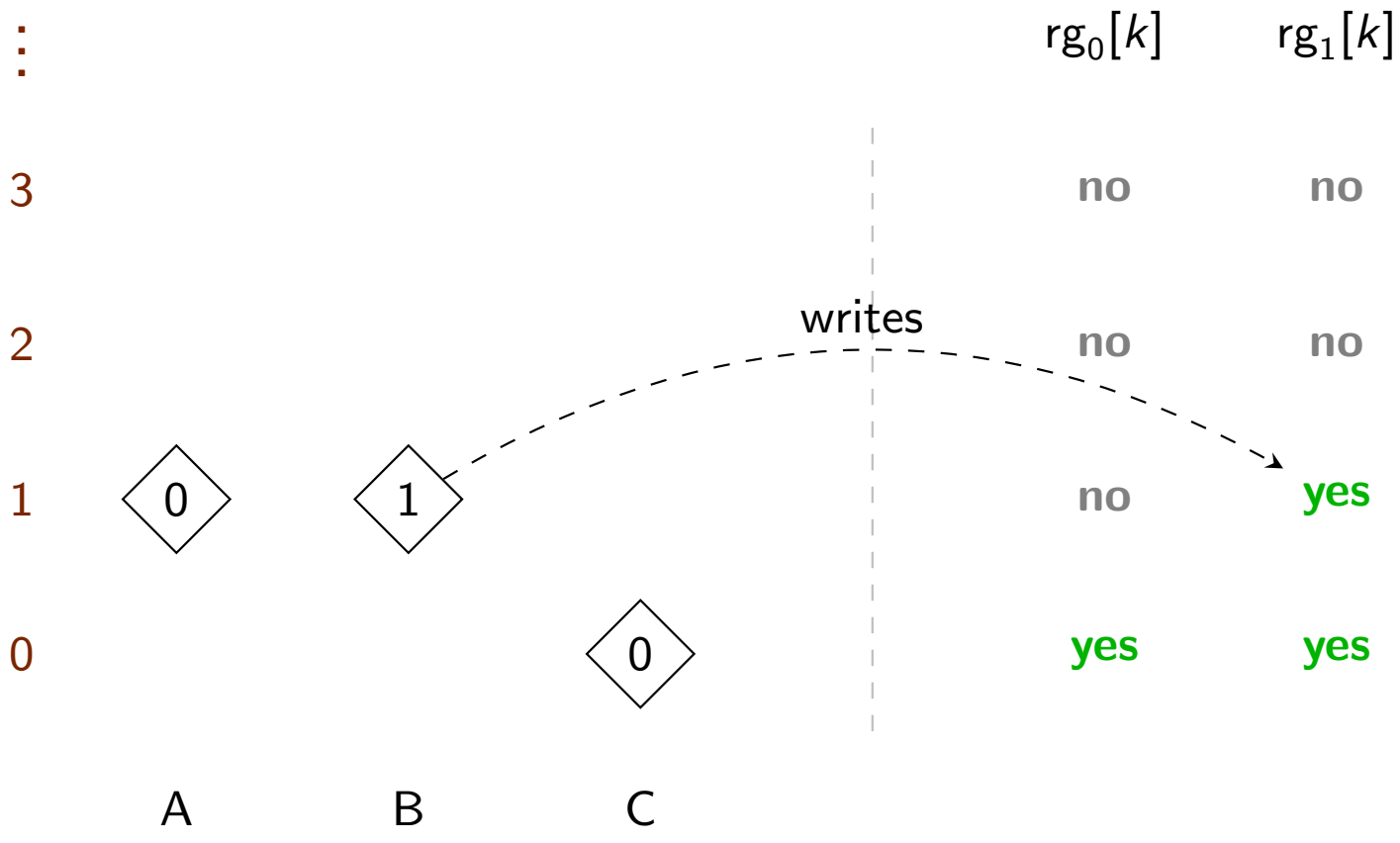
Aspnes' consensus algorithm illustrated



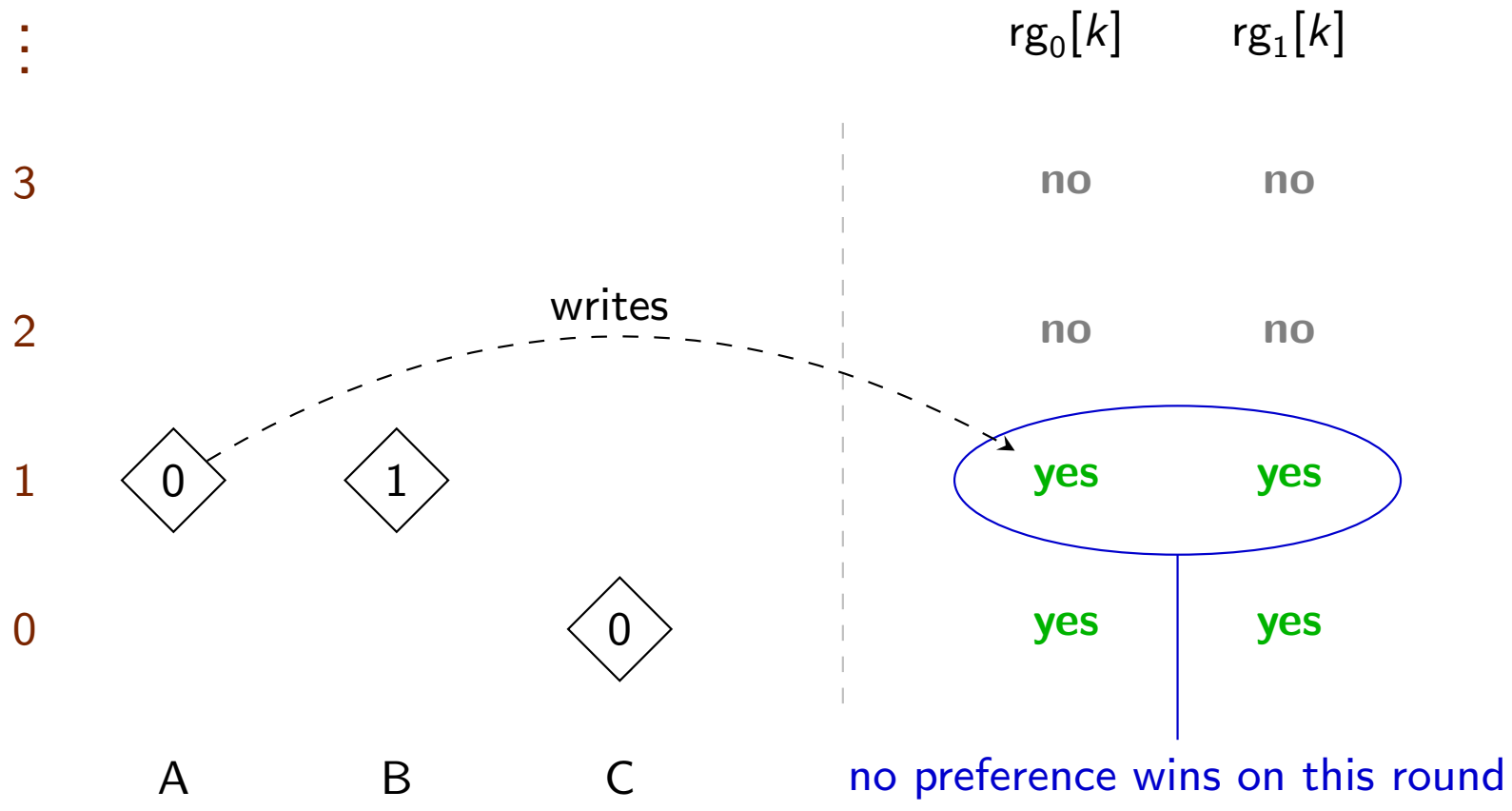
Aspnes' consensus algorithm illustrated



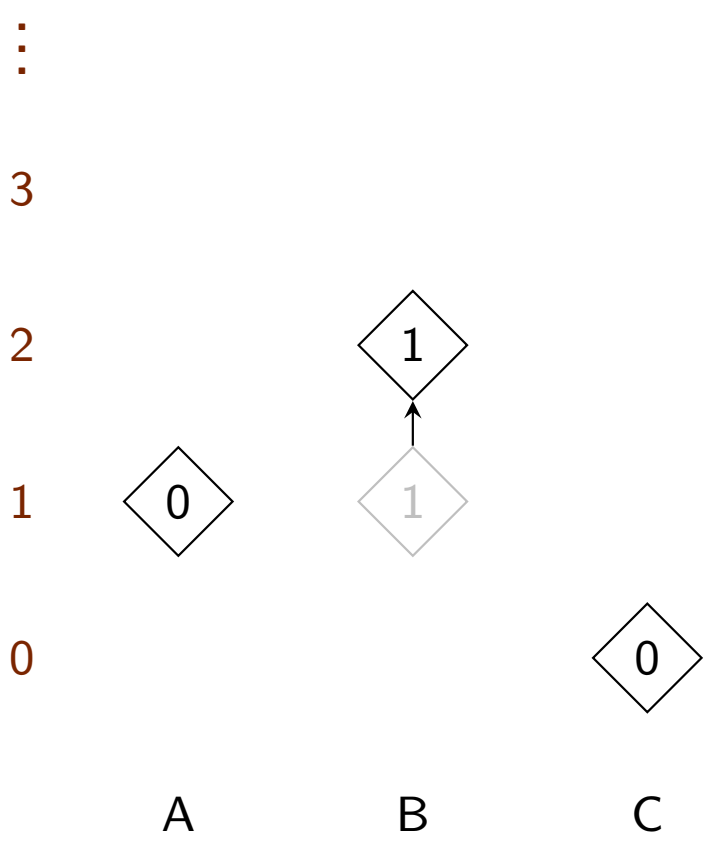
Aspnes' consensus algorithm illustrated



Aspnes' consensus algorithm illustrated

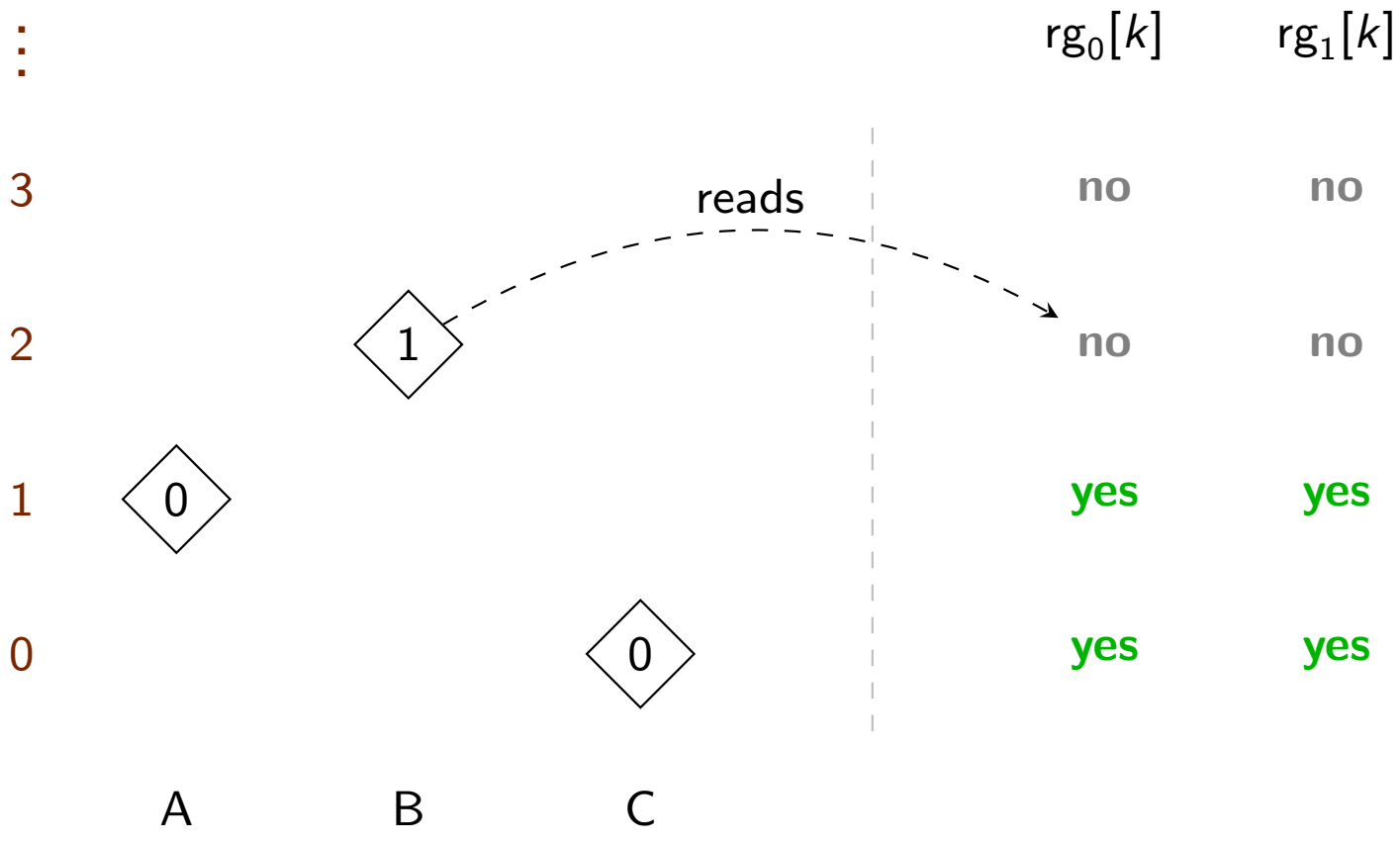


Aspnes' consensus algorithm illustrated

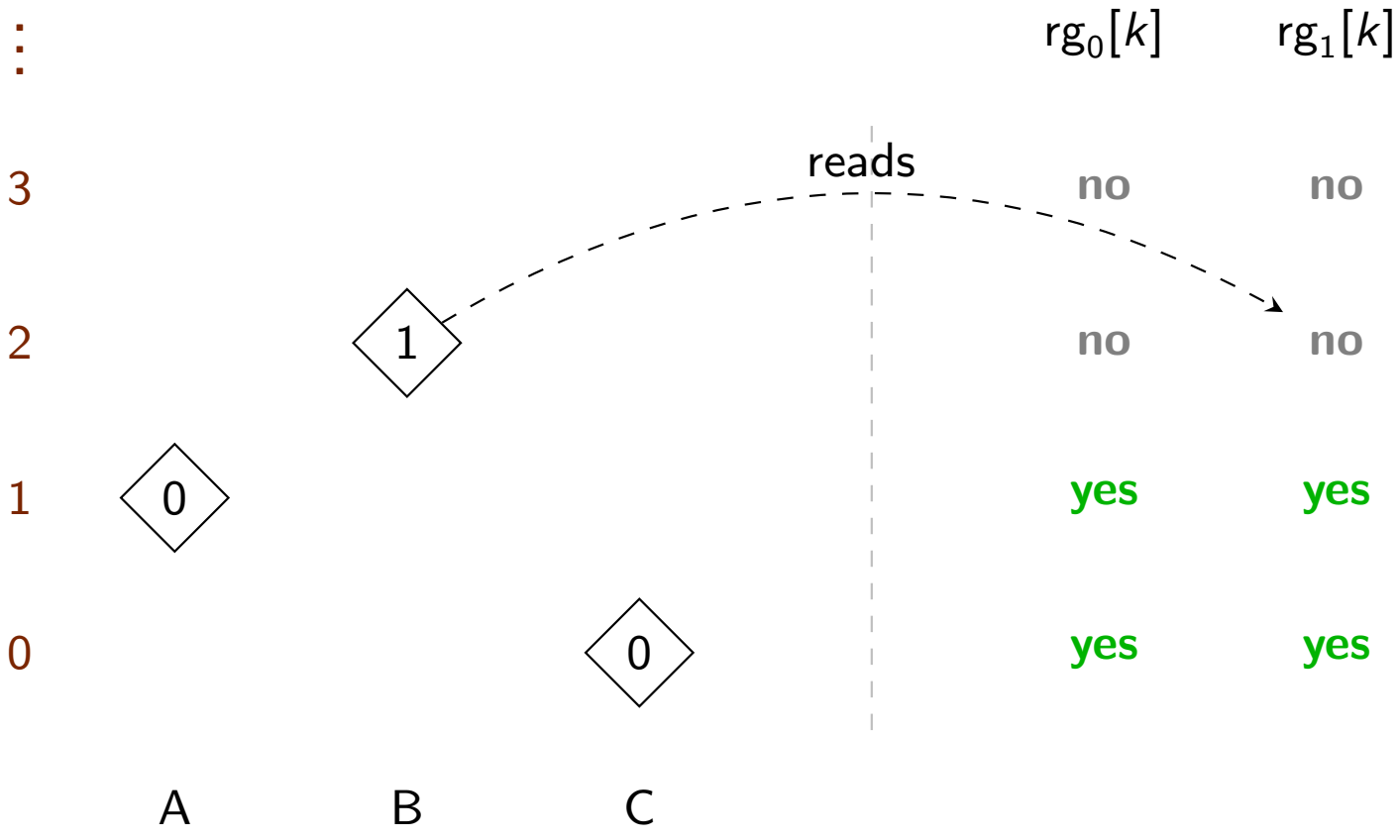


	$rg_0[k]$	$rg_1[k]$
3	no	no
2	no	no
1	yes	yes
0	yes	yes

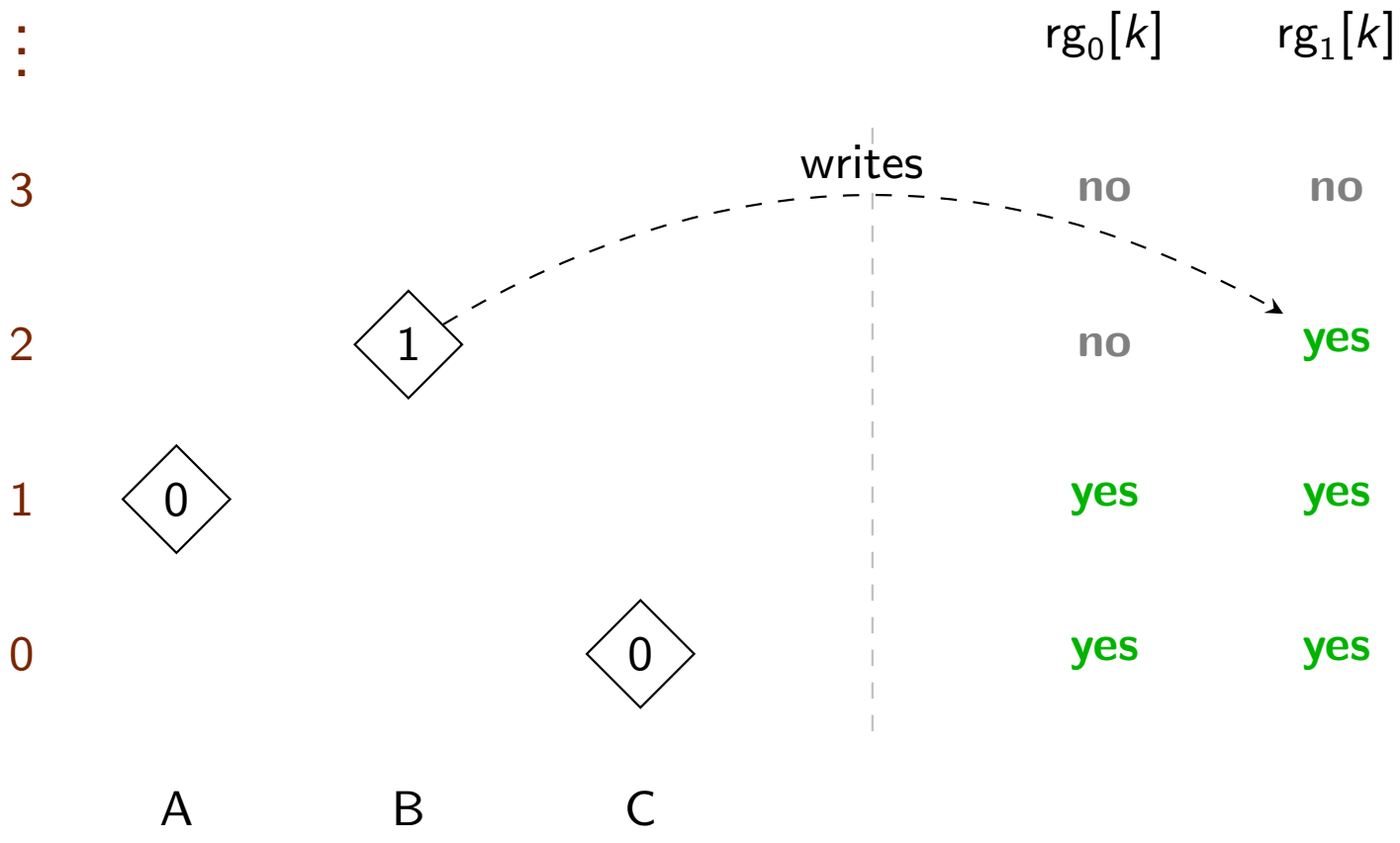
Aspnes' consensus algorithm illustrated



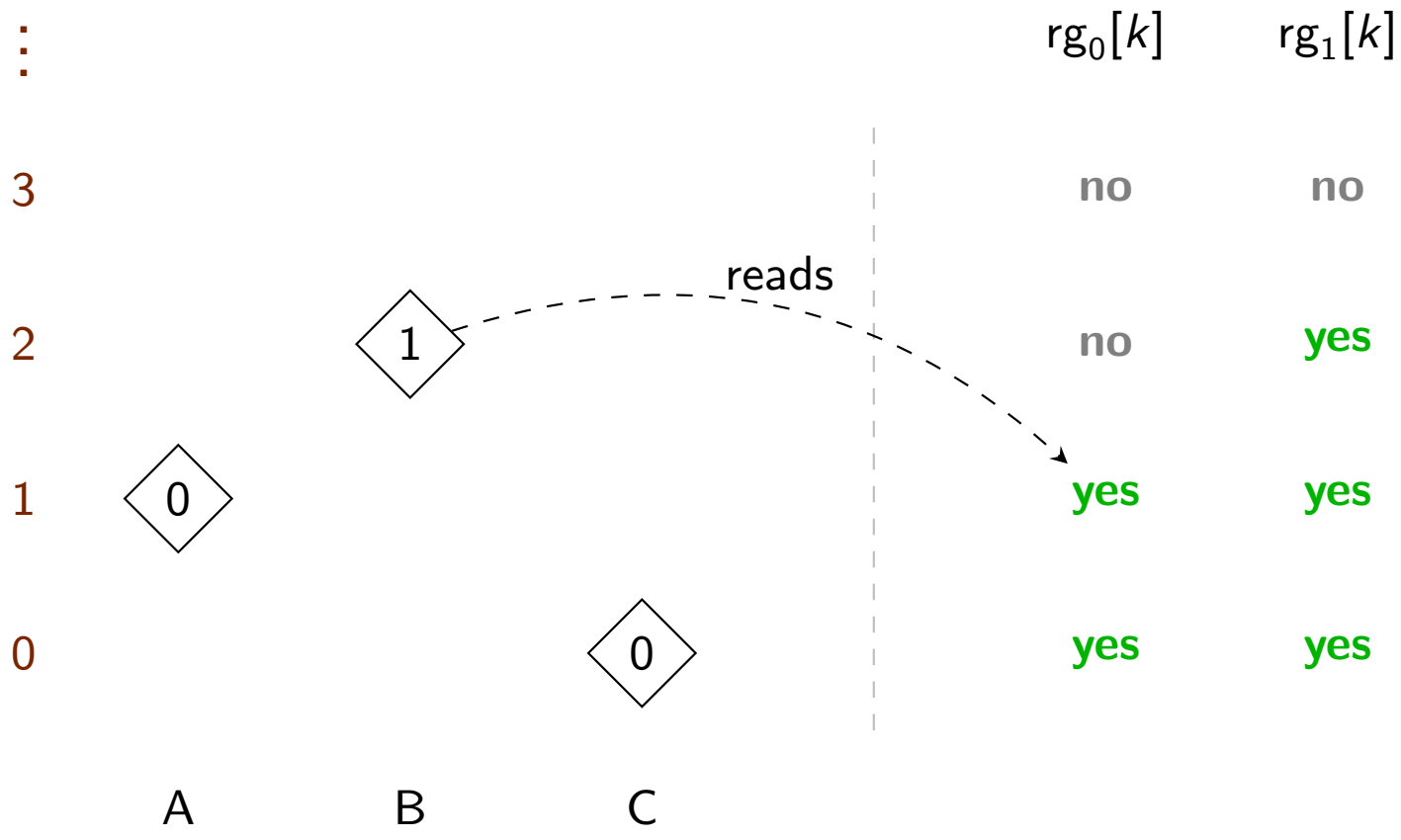
Aspnes' consensus algorithm illustrated



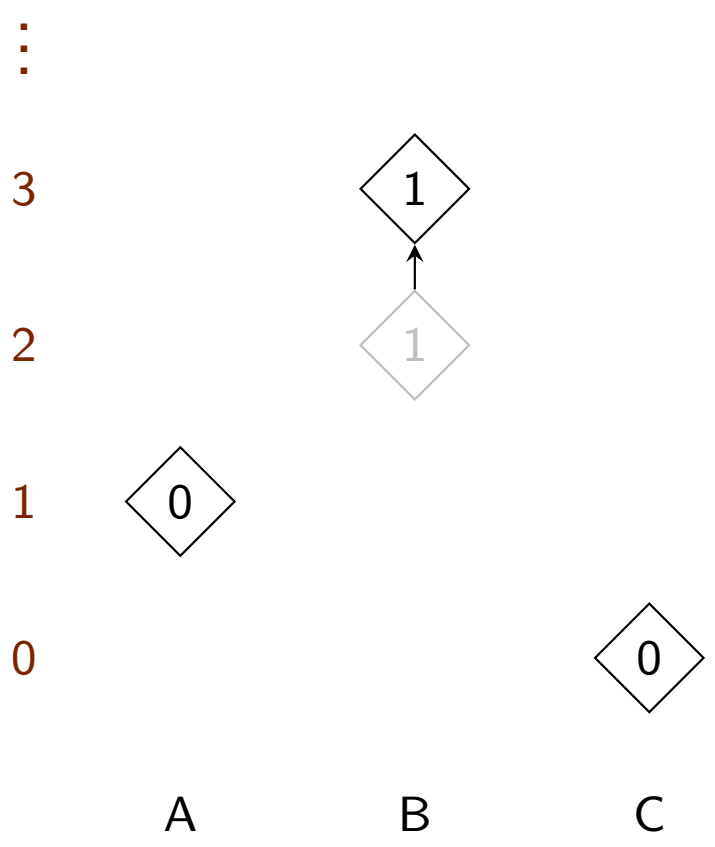
Aspnes' consensus algorithm illustrated



Aspnes' consensus algorithm illustrated

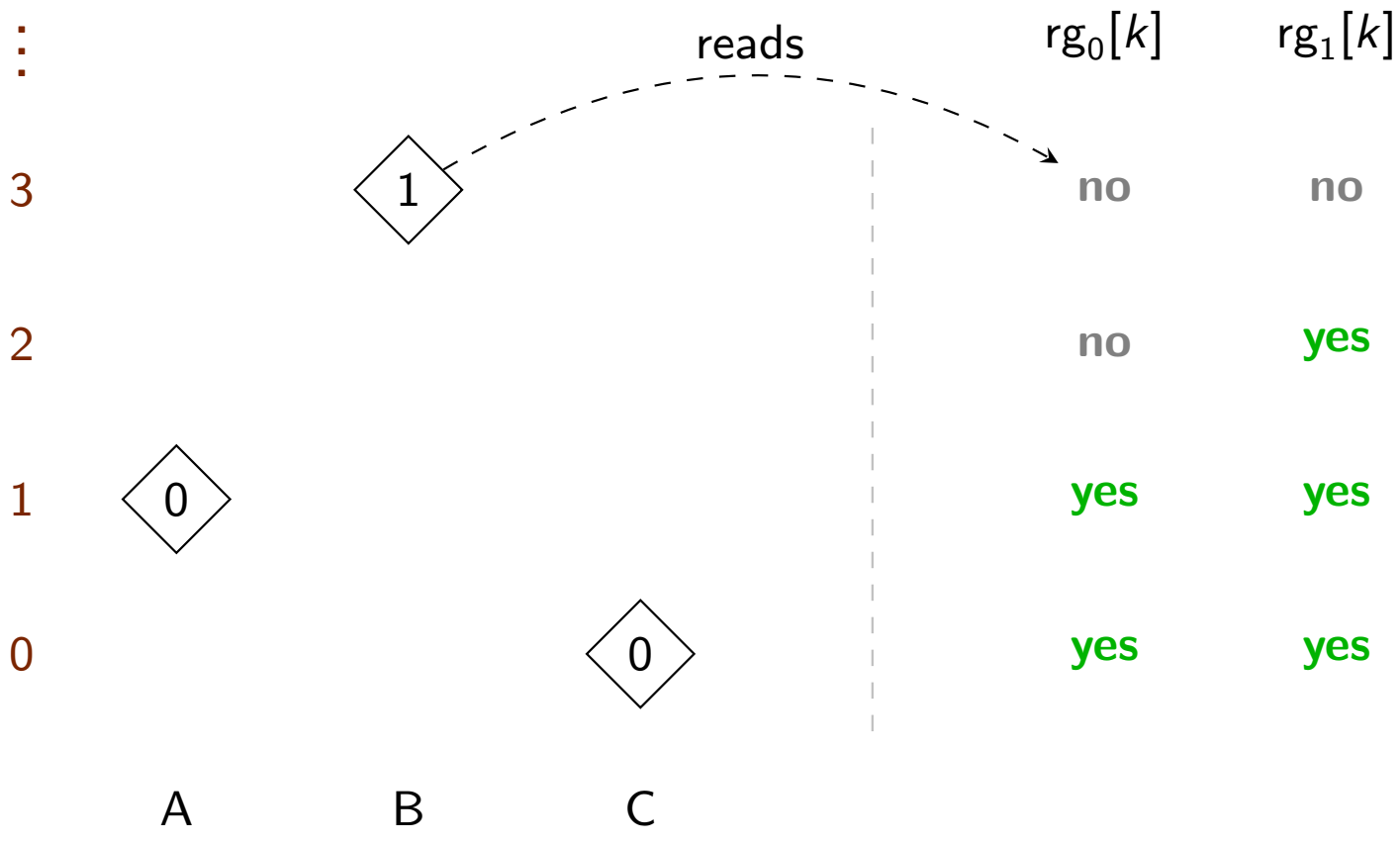


Aspnes' consensus algorithm illustrated

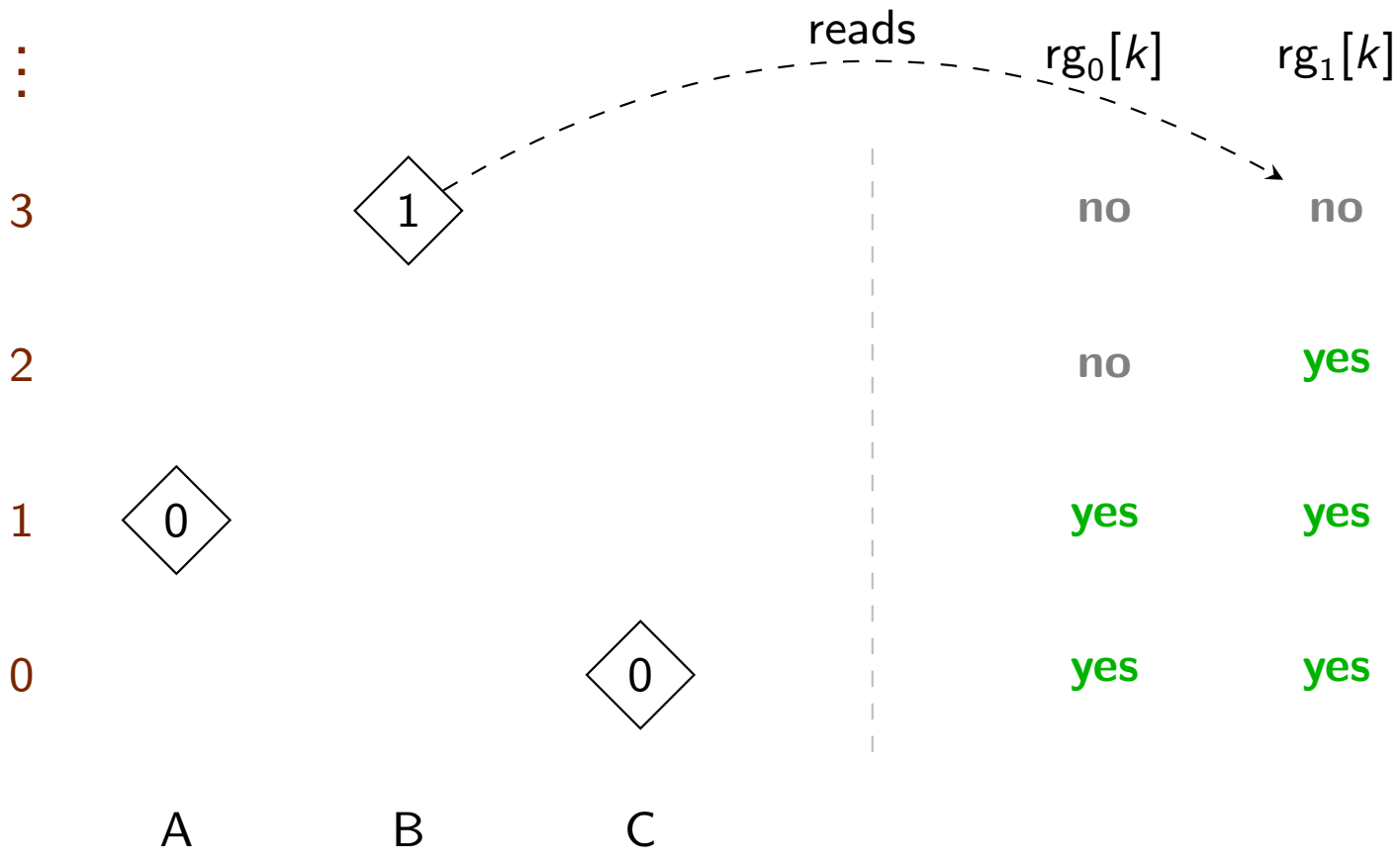


	$rg_0[k]$	$rg_1[k]$
⋮		
3	no	no
2	no	yes
1	yes	yes
0	yes	yes

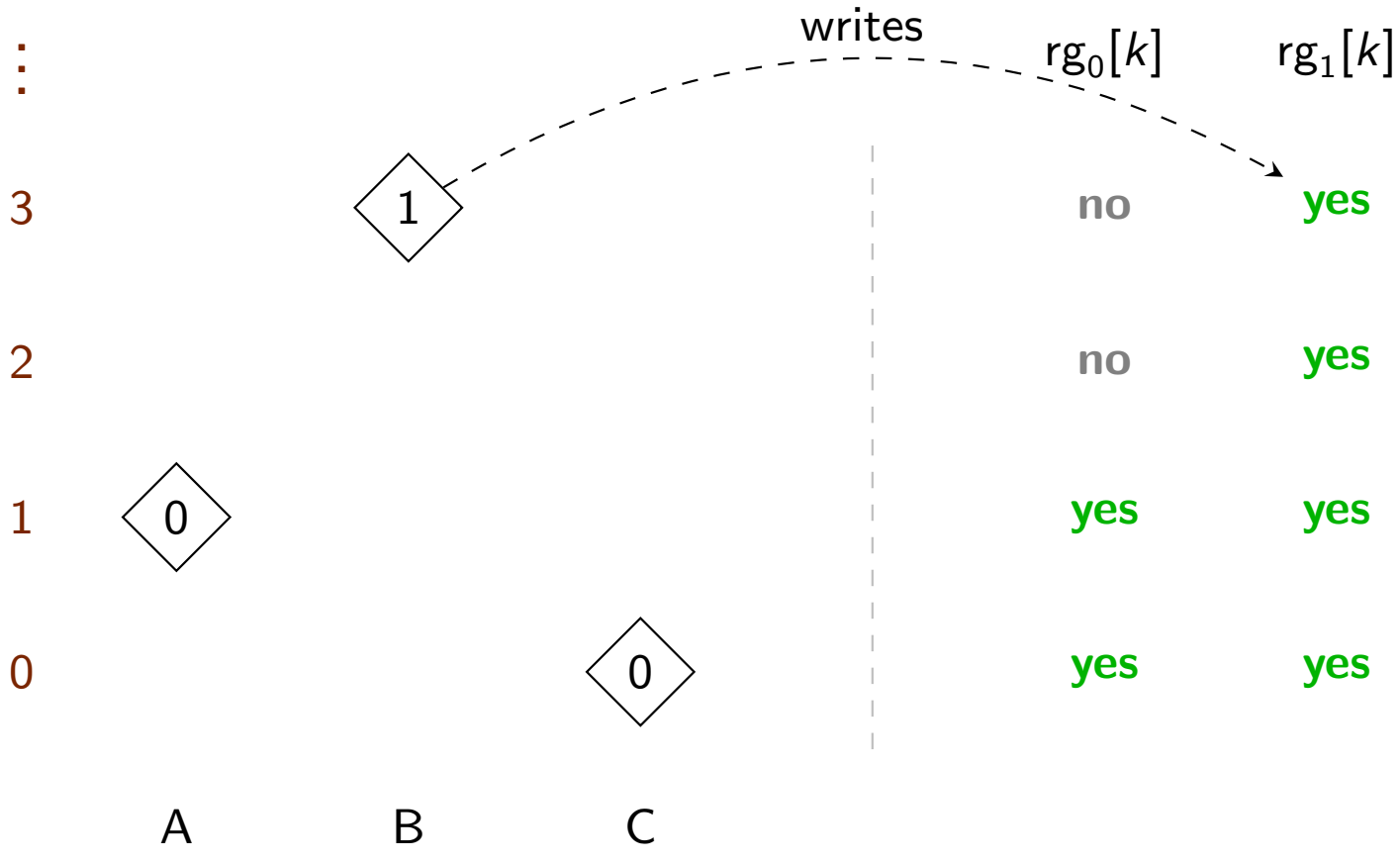
Aspnes' consensus algorithm illustrated



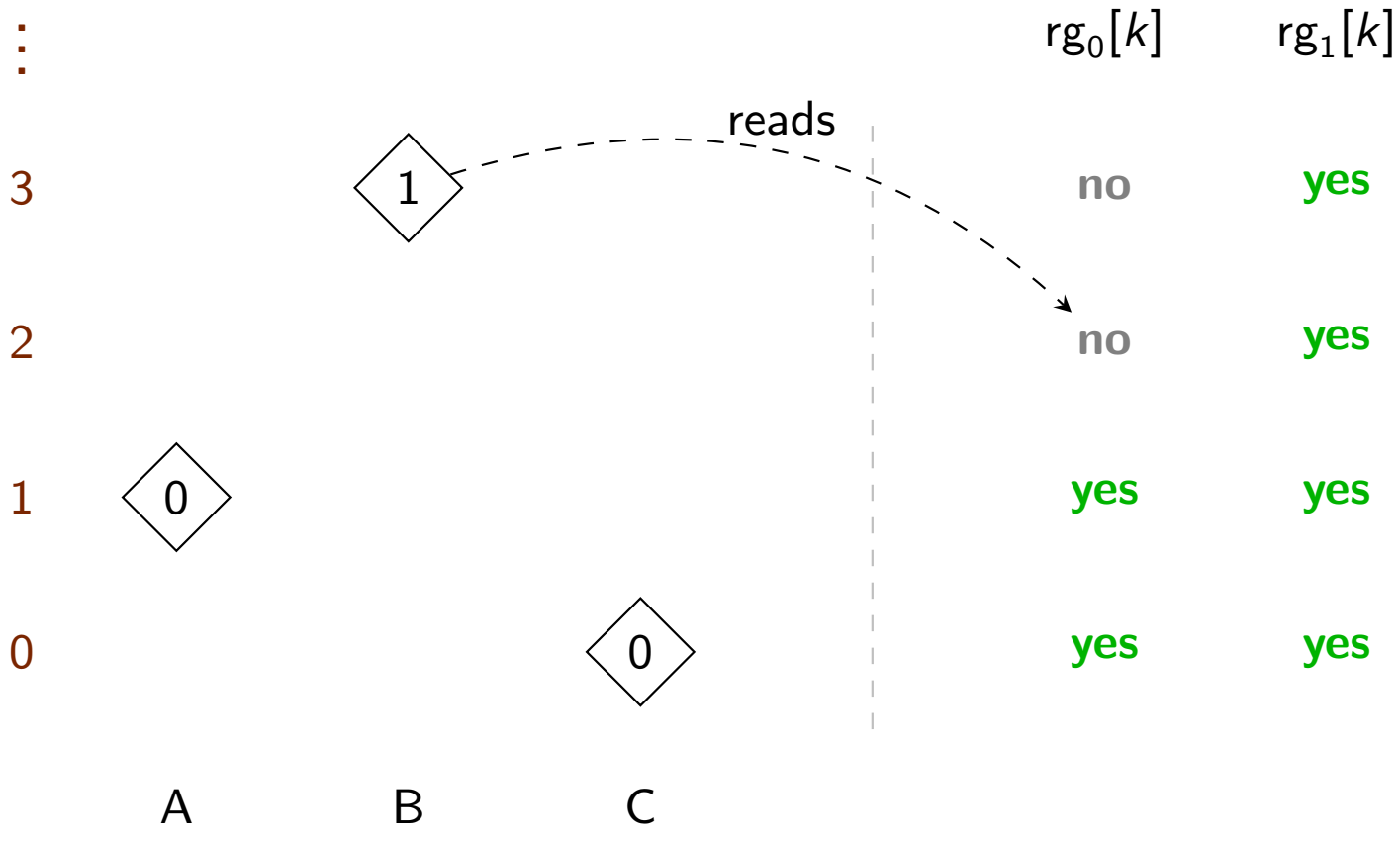
Aspnes' consensus algorithm illustrated



Aspnes' consensus algorithm illustrated

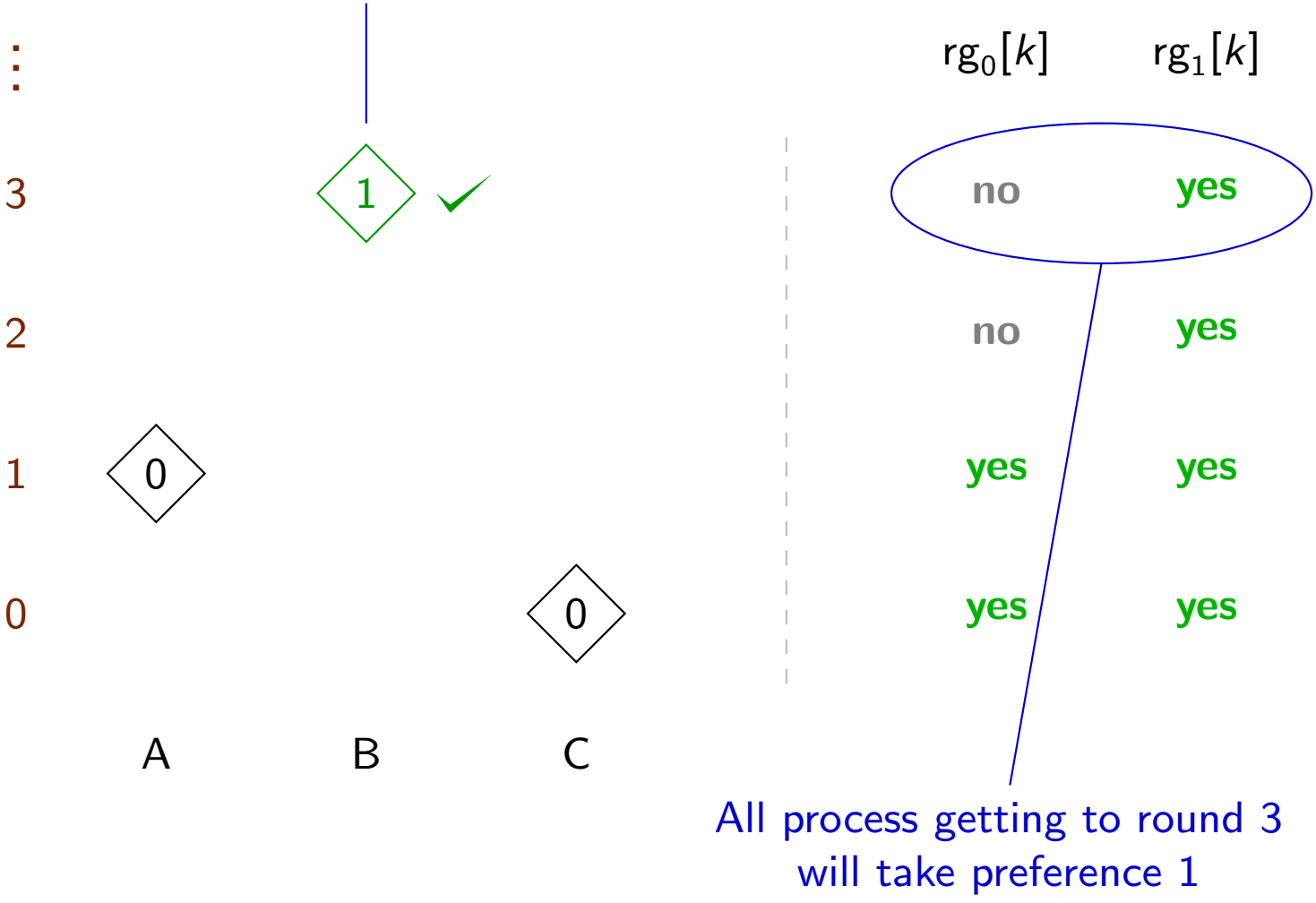


Aspnes' consensus algorithm illustrated



Aspnes' consensus algorithm illustrated

process B wins the race and decides, returns value 1

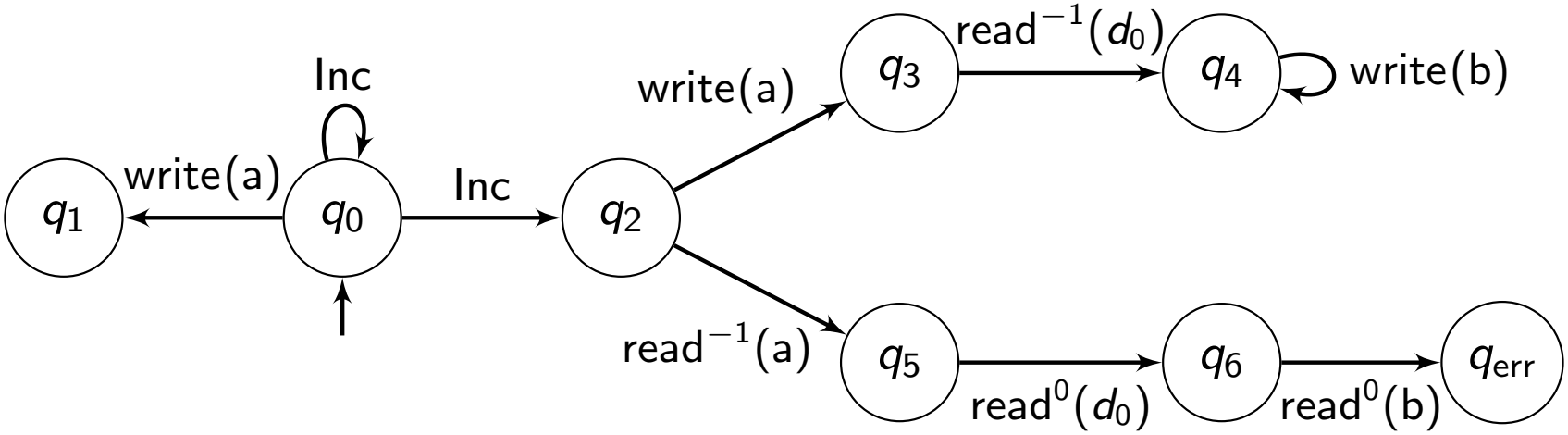


All process getting to round 3
will take preference 1

A model: round-based register protocols

Inspired by models for shared-memory systems without rounds[2][3].

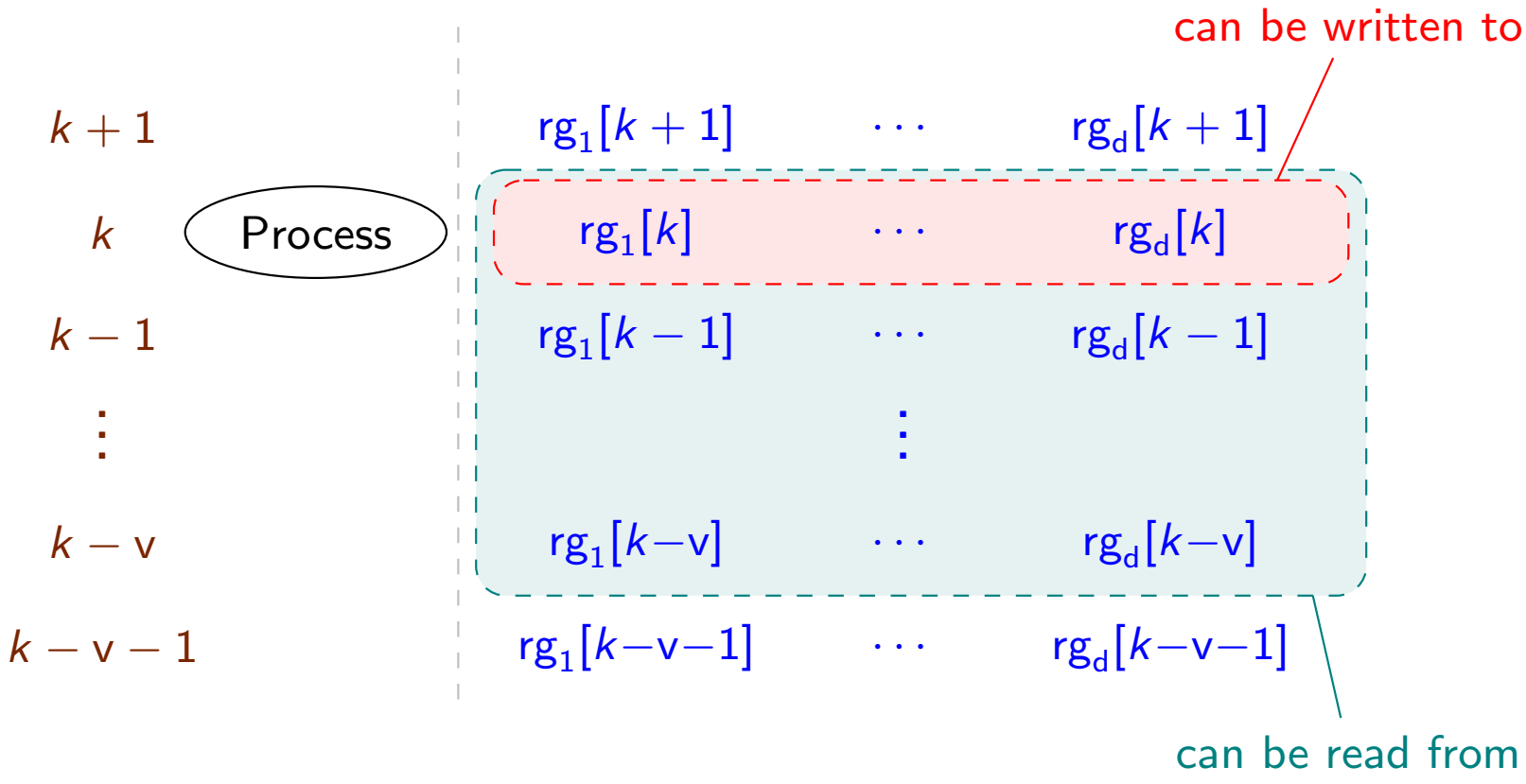
- One model for all processes: a finite automaton
- Each process has its own round number, which only increases
- A fixed number d of registers per round (the total number of registers is hence unbounded)



[2] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *CAV'13*

[3] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. *ICALP'16*

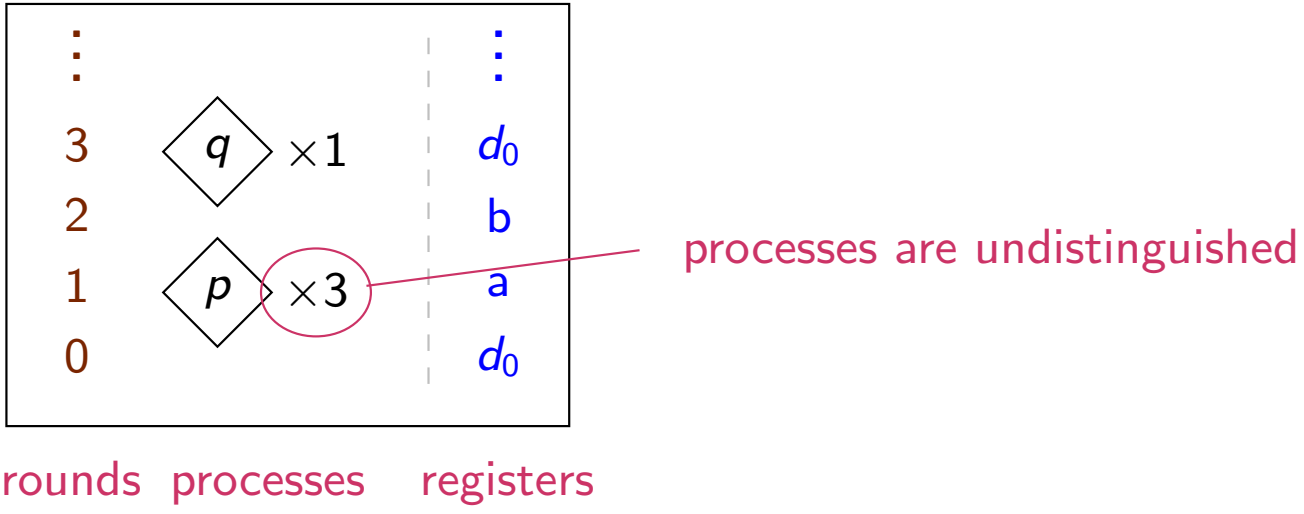
A limited visilibity range



v given in unary (in practical examples, $v = 1$)

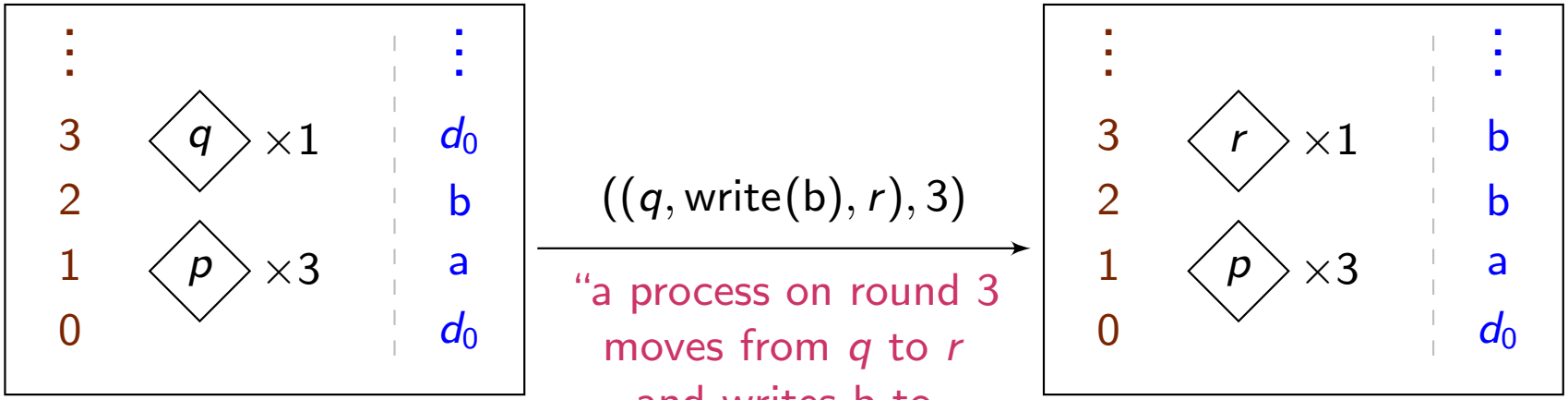
Semantics of the model

From now on, let d = 1: one register per round.



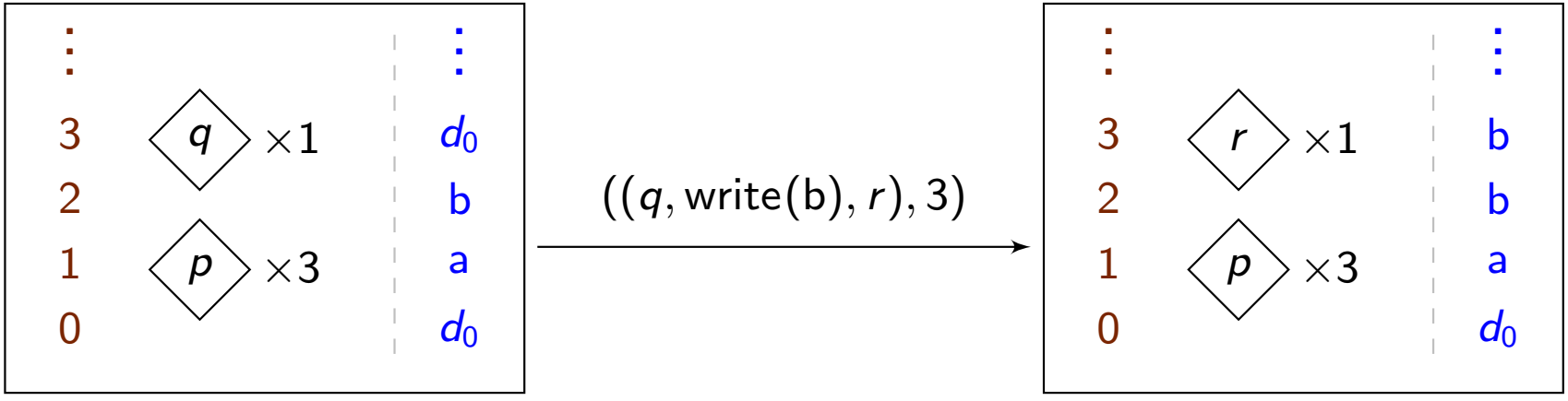
Semantics of the model

From now on, let d = 1: one register per round.

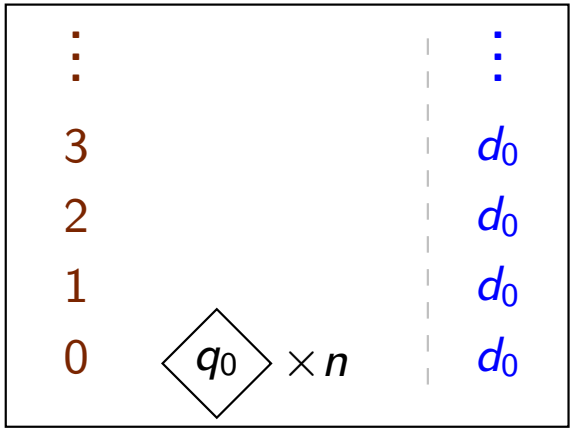


Semantics of the model

From now on, let d = 1: one register per round.



Initial configuration of size n:



The safety problem

The (parameterized) safety problem

Is it true that, for all numbers of processes n and all executions from the initial configuration of size n, an error state q_{err} is avoided?

Dual problem: look for an execution *covering* the error.

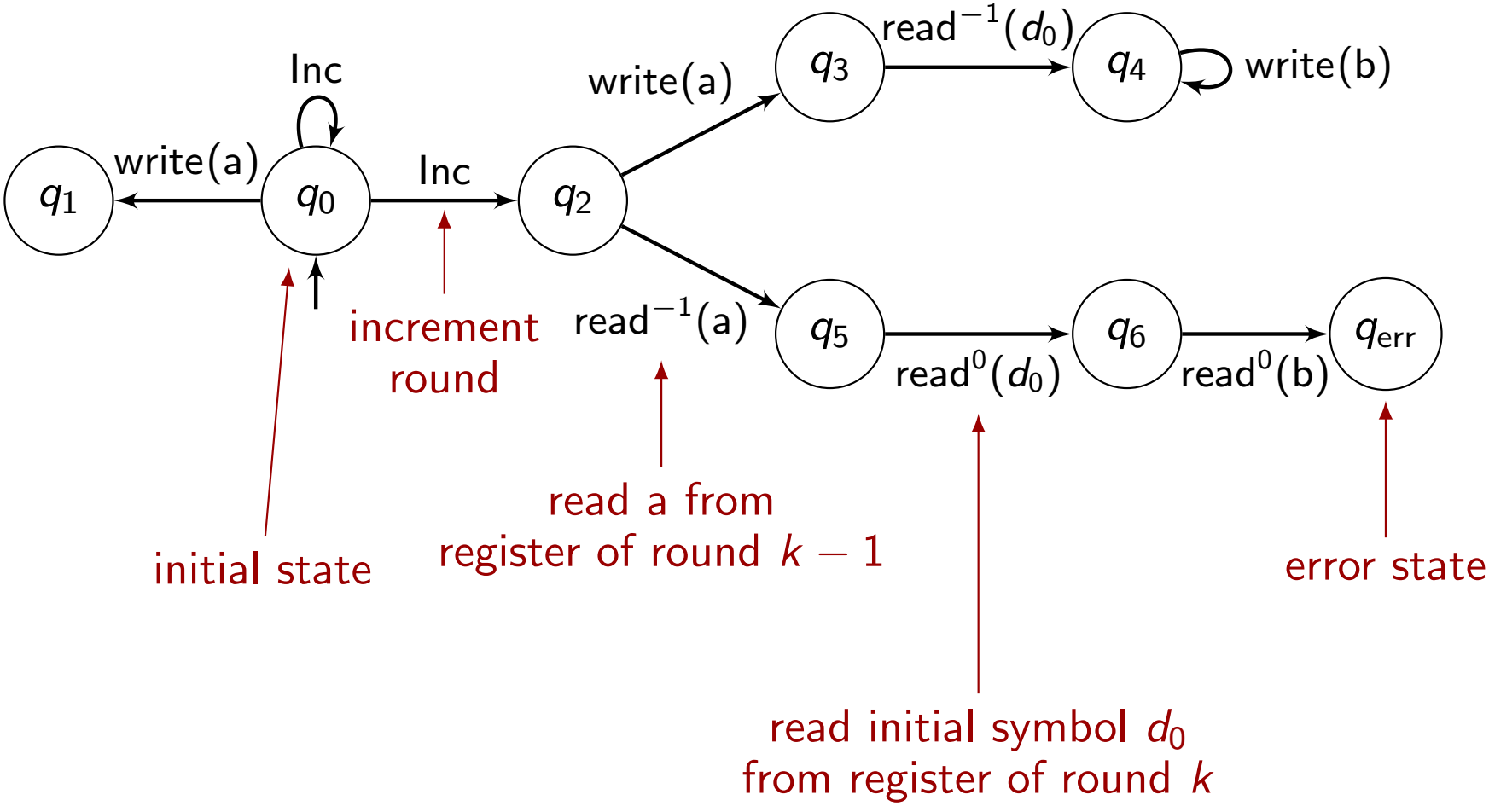
The safety problem

The (parameterized) safety problem

Is it true that, for all numbers of processes n and all executions from the initial configuration of size n, an error state q_{err} is avoided?

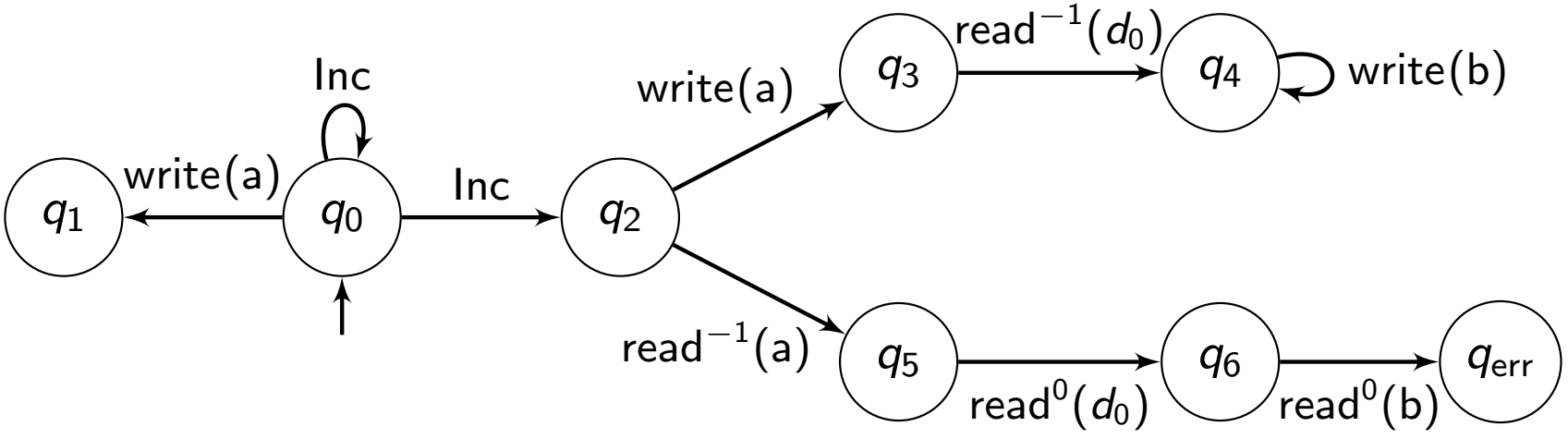
Dual problem: look for an execution *covering* the error.
If the error state cannot be covered, the system is safe.
Both Agreement and Validity of Aspnes' consensus algorithm can be encoded as safety properties.

A small example



v = 1 (processes can read one round back)

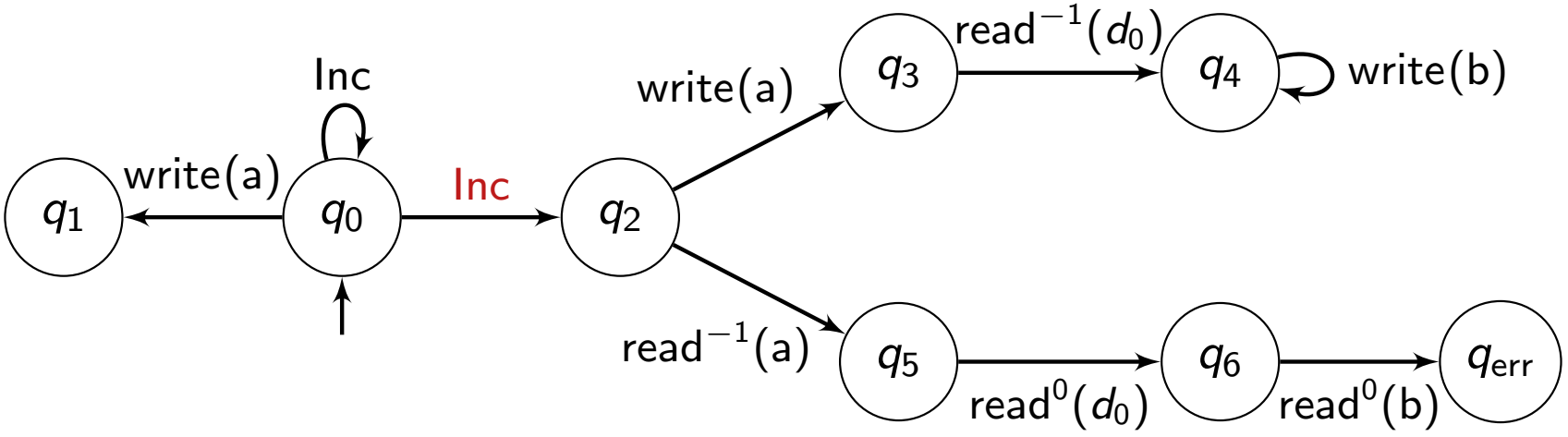
A small example



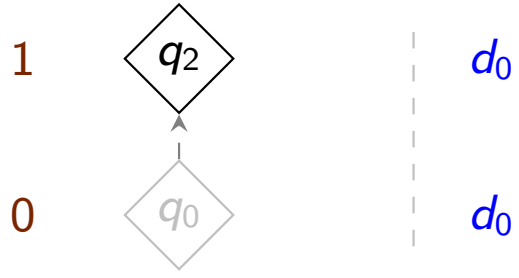
State q_4 can be covered from the initial configuration with one process:

1		d_0
0	q_0	d_0

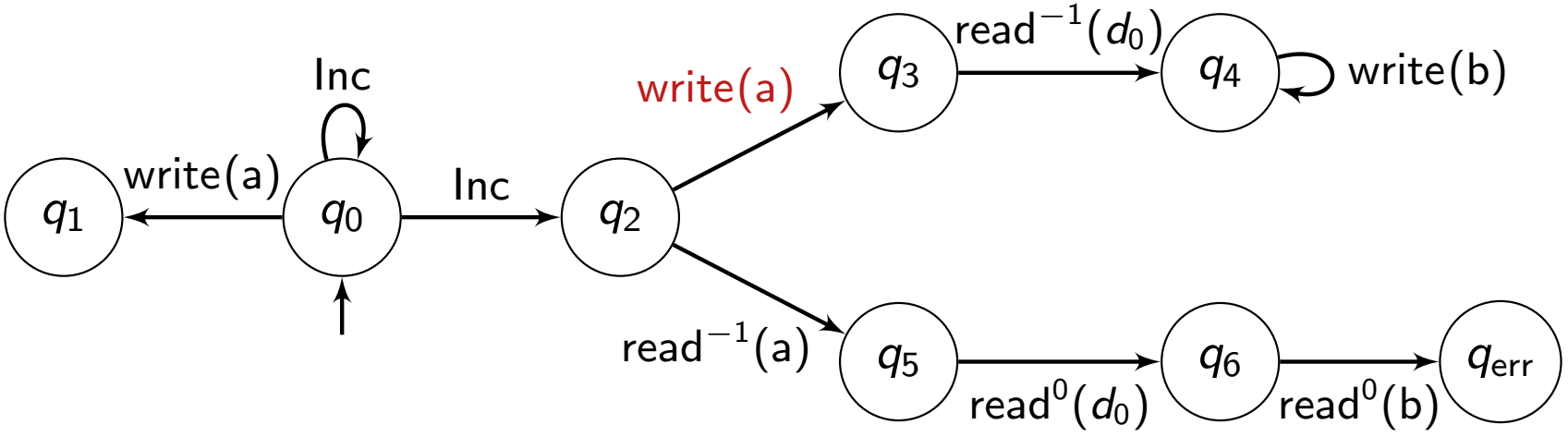
A small example



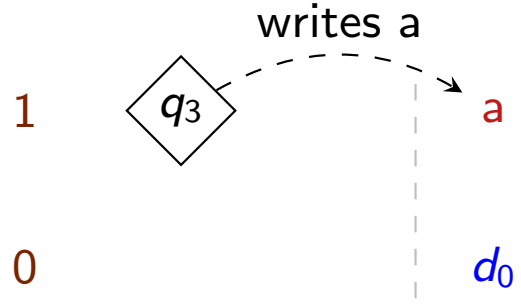
State q_4 can be covered from the initial configuration with one process:



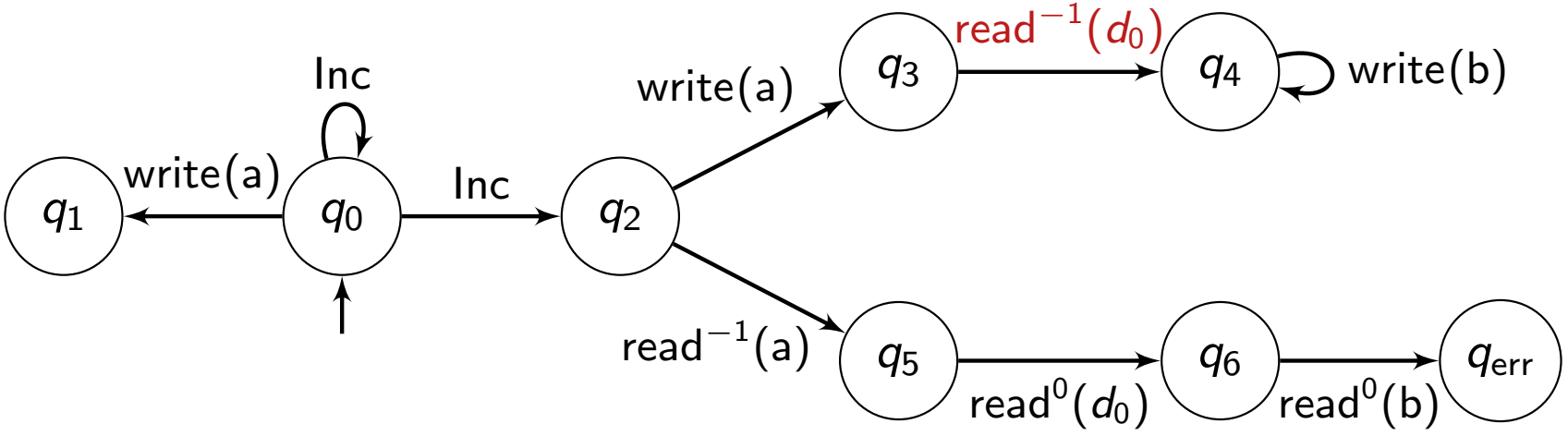
A small example



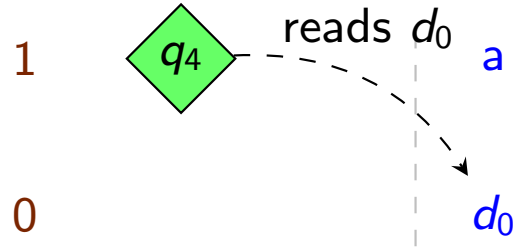
State q_4 can be covered from the initial configuration with one process:



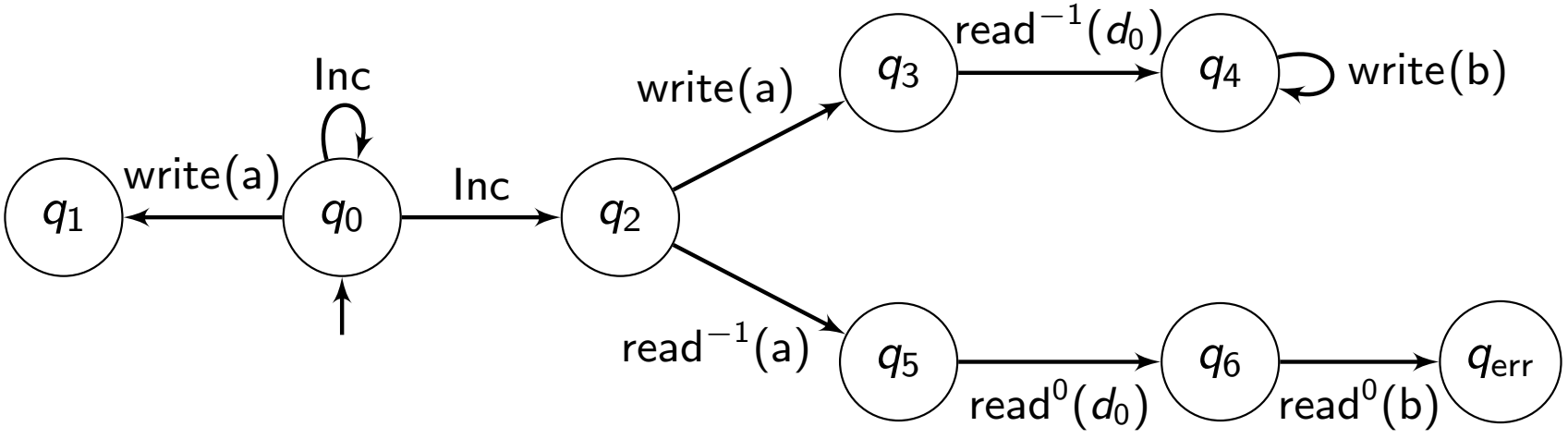
A small example



State q_4 can be covered from the initial configuration with one process:

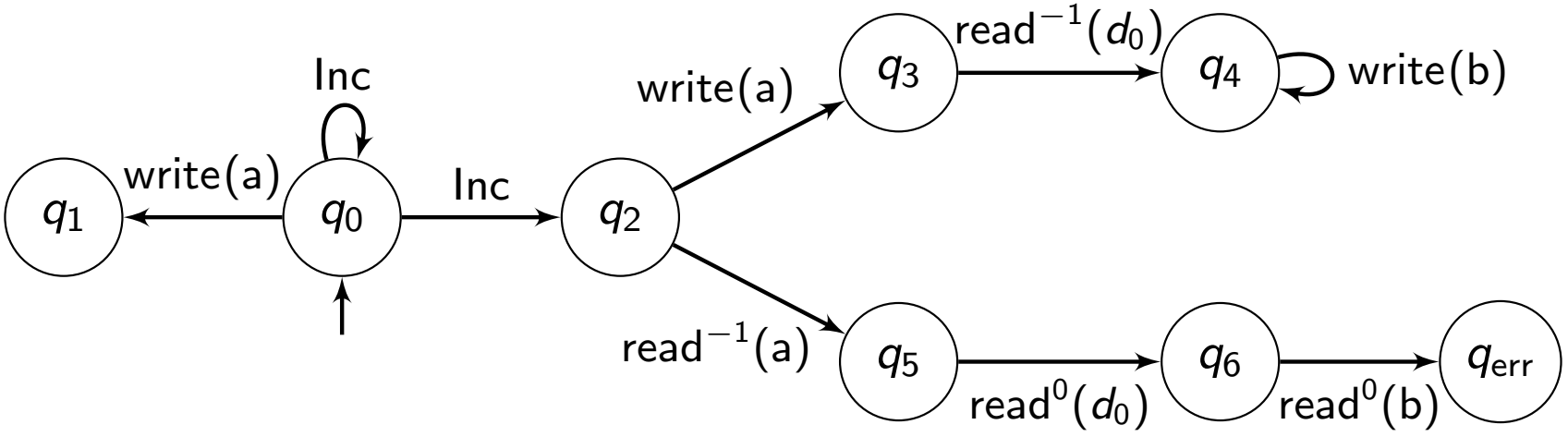


A small example



State q_6 can be covered from the initial configuration with two processes:

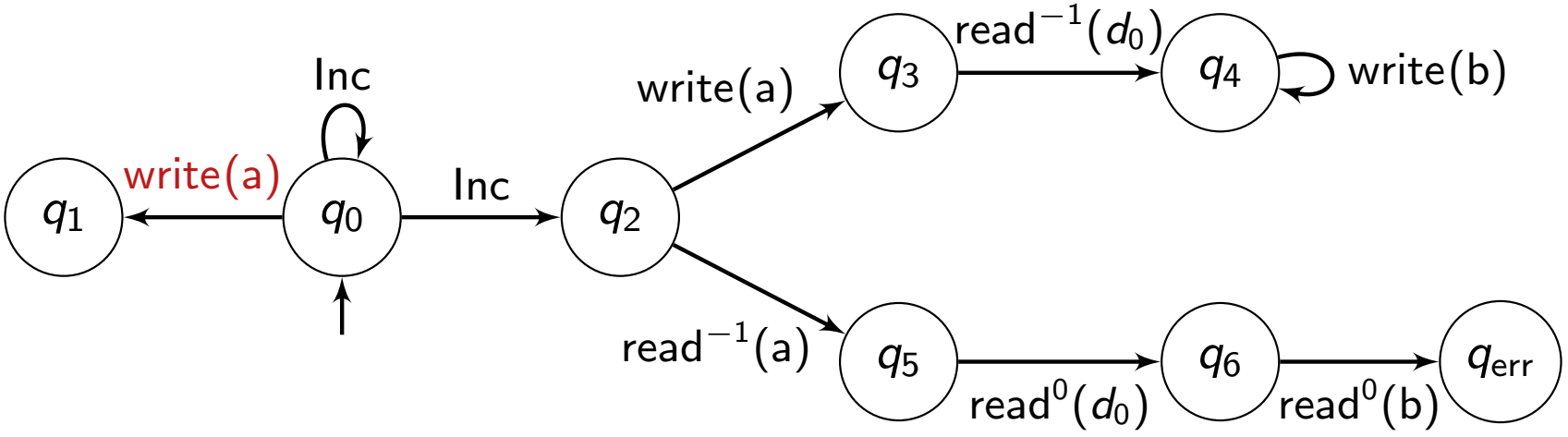
A small example



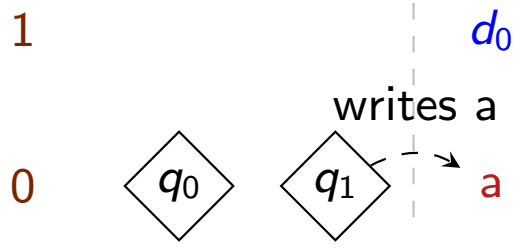
State q_6 can be covered from the initial configuration with two processes:

1		d_0
0	q_0 $\times 2$	d_0

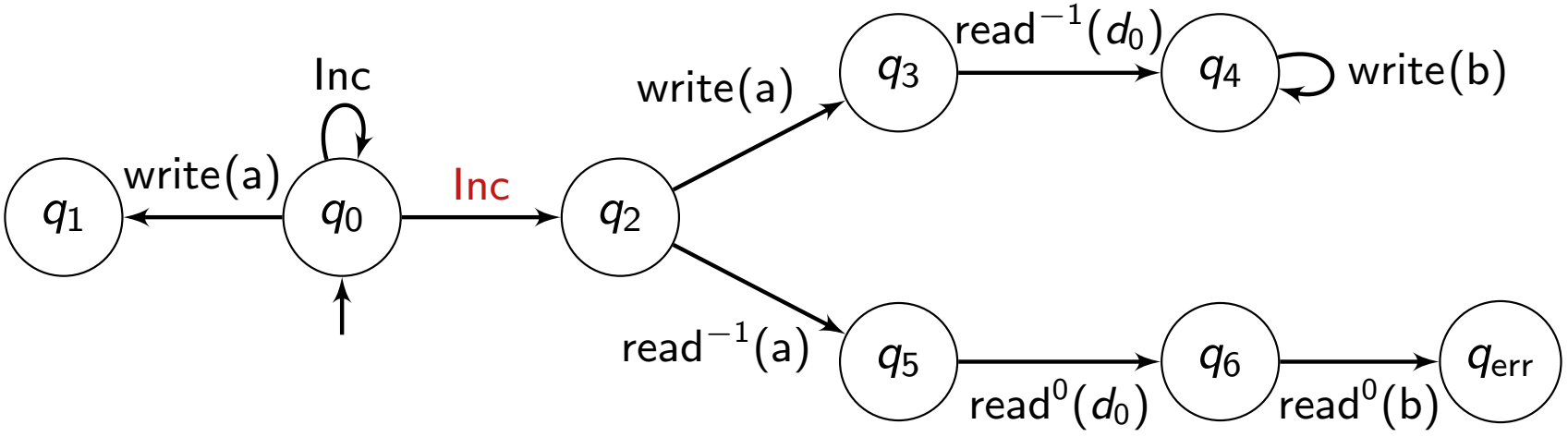
A small example



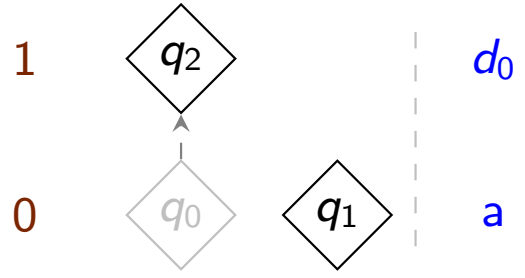
State q_6 can be covered from the initial configuration with two processes:



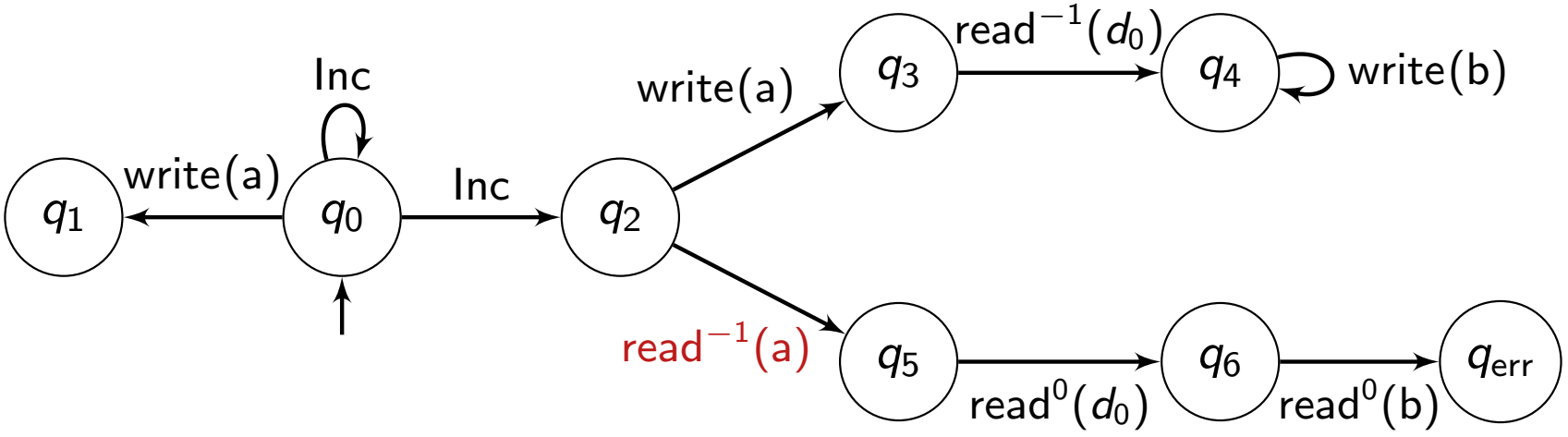
A small example



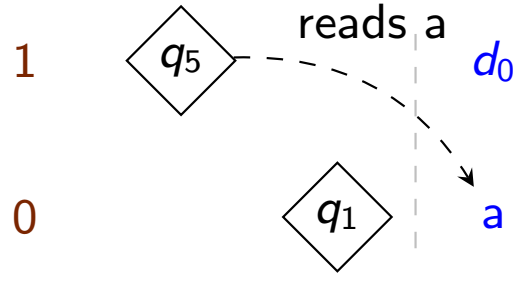
State q_6 can be covered from the initial configuration with two processes:



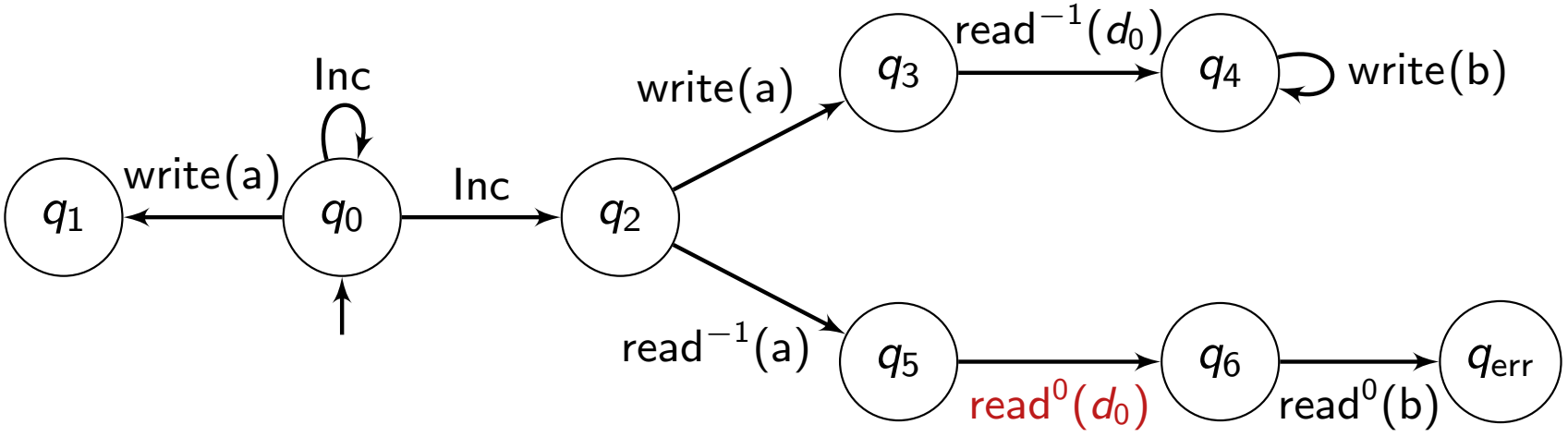
A small example



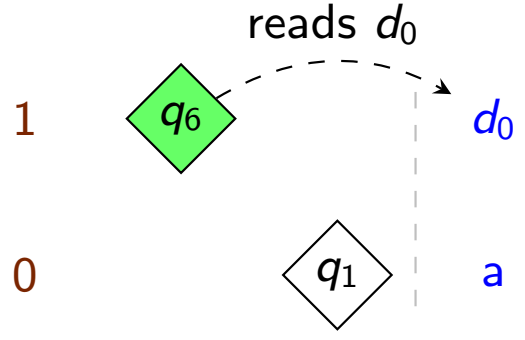
State q_6 can be covered from the initial configuration with two processes:



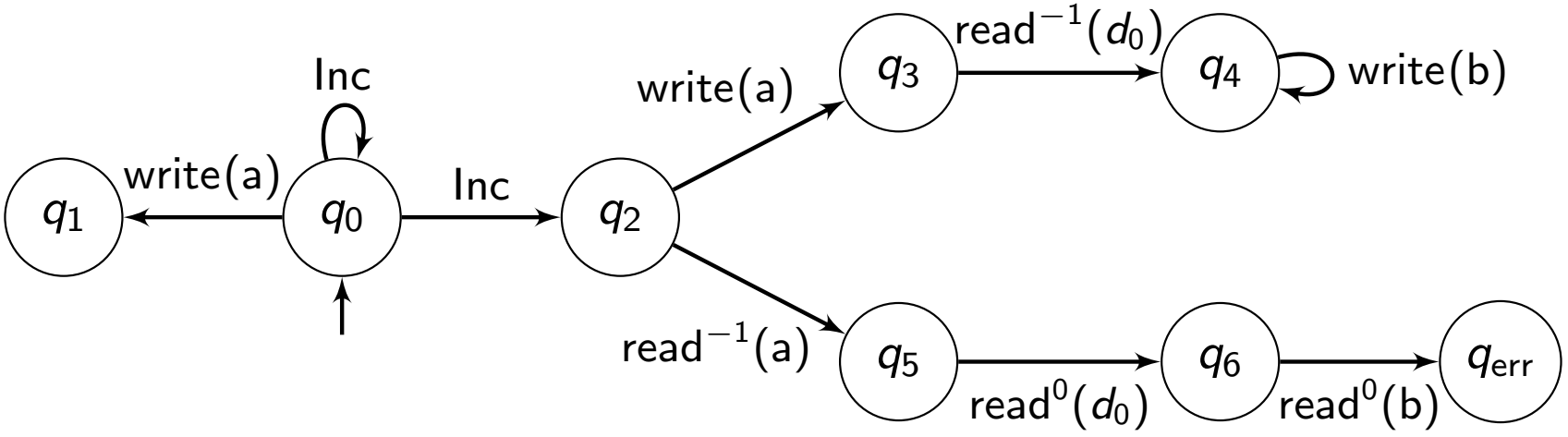
A small example



State q_6 can be covered from the initial configuration with two processes:

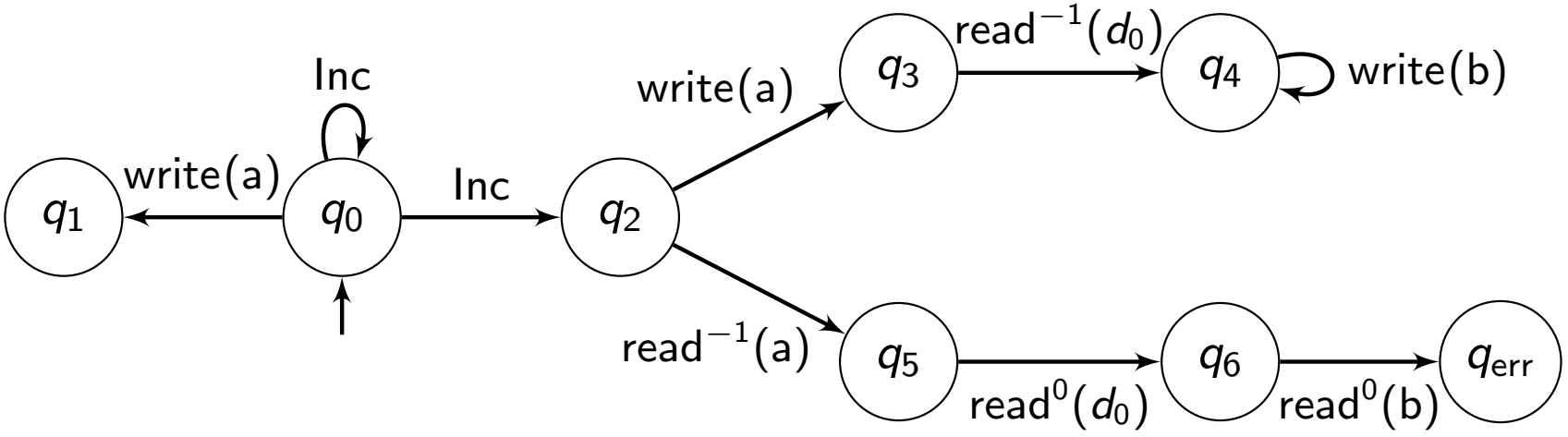


A small example



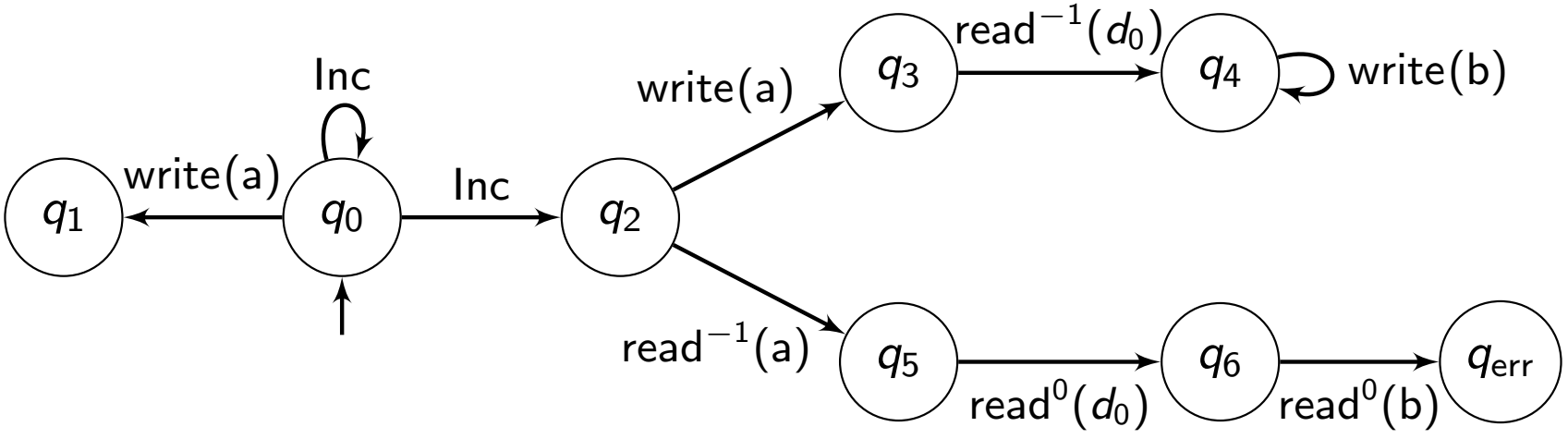
Claim: the system is safe.

A small example



Claim: the system is safe.
Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. This is not possible:

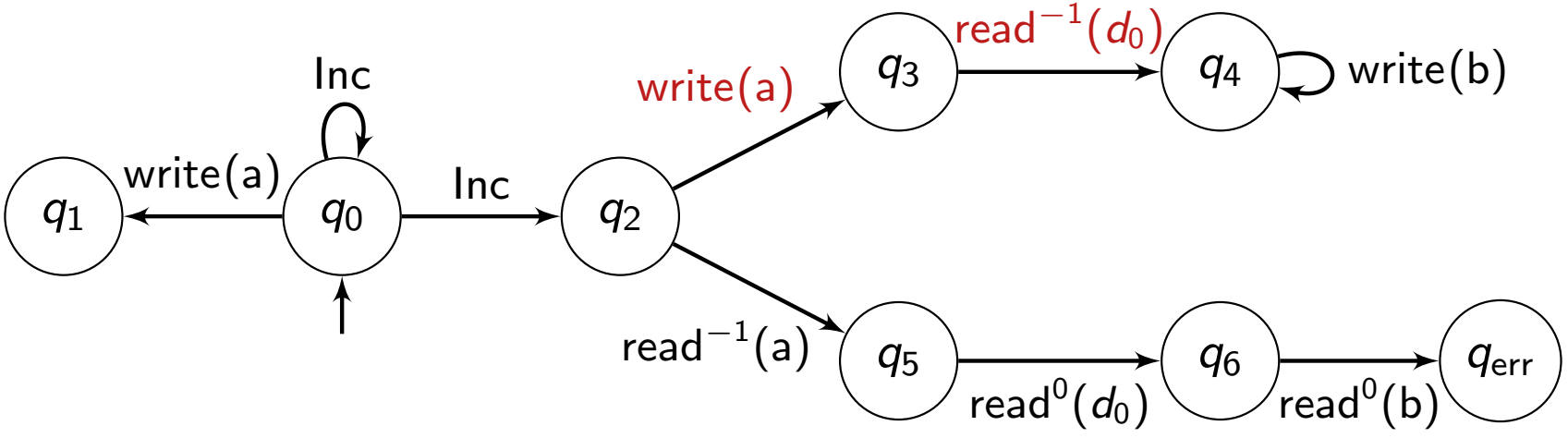
A small example



Claim: the system is safe.
Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. This is not possible:

(q_4, k) and (q_6, k) are incompatible

A small example

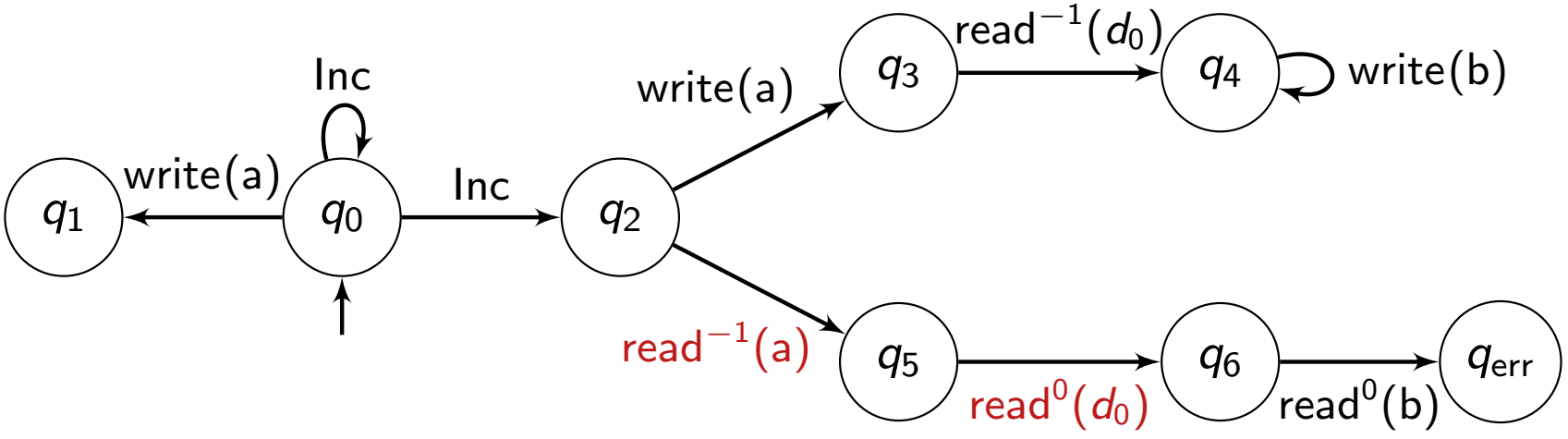


Claim: the system is safe.
Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. This is not possible:

(q_4, k) and (q_6, k) are incompatible

- To cover (q_4, k), one must write to $\text{rg}[k]$ while $\text{rg}[k-1]$ still has value d_0;

A small example



Claim: the system is safe.
Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. This is not possible:

(q_4, k) and (q_6, k) are incompatible

- To cover (q_4, k), one must write to rg$[k]$ while rg$[k-1]$ still has value d_0;
- To cover (q_6, k), one must write to rg$[k-1]$ while rg$[k]$ still has value d_0.

This is the only source of incompatibility!

Theorem

Parameterized safety in round-based register protocols is PSPACE-complete.[4]

[4] Nathalie Bertrand, Nicolas Markey, Ocan Sankur, Nicolas Waldburger. Parameterized safety verification of round-based shared-memory systems. *ICALP'22*

Lower bounds

Exponential lower bounds

In order to reach an error state, one might need at least:

- An exponential number of processes,
- spreading across an exponential number of rounds at the same time.

Lower bounds

Exponential lower bounds

In order to reach an error state, one might need at least:

- An exponential number of processes,
- spreading across an exponential number of rounds at the same time.

Theorem

The safety problem is PSPACE-hard.

By reduction from Quantified Boolean Formula.

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

Ingredients of the algorithm

- Copycat property (thanks to non-atomicity)

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

Ingredients of the algorithm

- Copycat property (thanks to non-atomicity)
- Exploit locality of reads and writes with respect to the round

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

Ingredients of the algorithm

- Copycat property (thanks to non-atomicity)
- Exploit locality of reads and writes with respect to the round
- Rely on a sliding window along the rounds

An abstraction for safety

Two observations

- Whenever an execution puts one process on (q, k), one can put arbitrarily many processes on (q, k) (increasing the total number of processes).
- Whenever a symbol $x \neq d_0$ has be written to some register $\mathrm{rg}_k[\alpha]$, it can be written to this register again.

An abstraction for safety

Two observations

- Whenever an execution puts one process on (q, k), one can put arbitrarily many processes on (q, k) (increasing the total number of processes).
- Whenever a symbol $x \neq d_0$ has be written to some register $\mathrm{rg}_k[\alpha]$, it can be written to this register again.

An abstraction for safety

In an abstract configuration, one remembers only:

- which pairs (state, round) are populated by at least one process,
- which registers still have symbol d_0 and which symbols can be written to which registers.

An abstraction for safety

Two observations

- Whenever an execution puts one process on (q, k), one can put arbitrarily many processes on (q, k) (increasing the total number of processes).
- Whenever a symbol $x \neq d_0$ has be written to some register $\mathrm{rg}_k[\alpha]$, it can be written to this register again.

An abstraction for safety

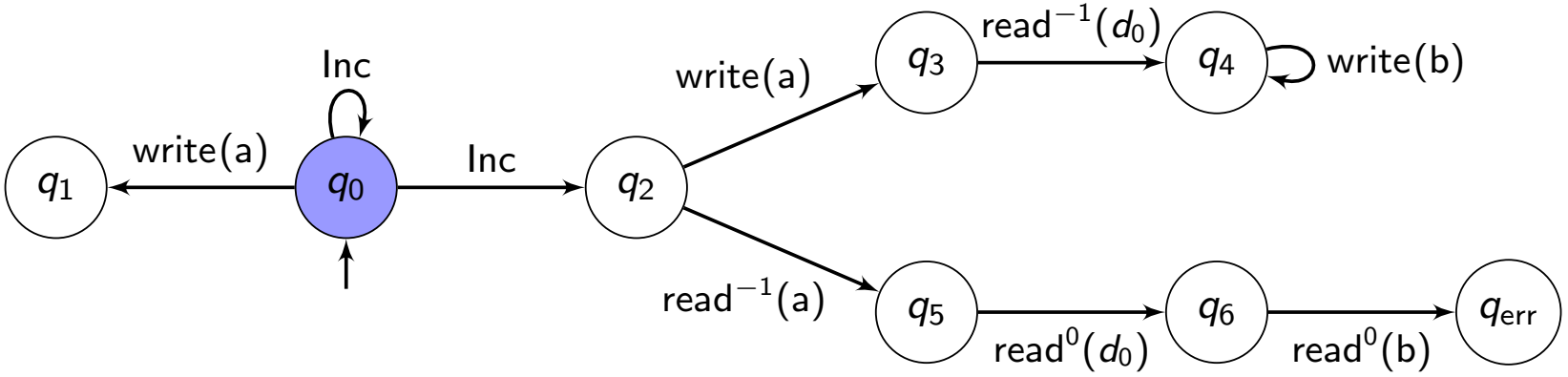
In an abstract configuration, one remembers only:

- which pairs (state, round) are populated by at least one process,
- which registers still have symbol d_0 and which symbols can be written to which registers.

Soundness and completeness of the abstraction

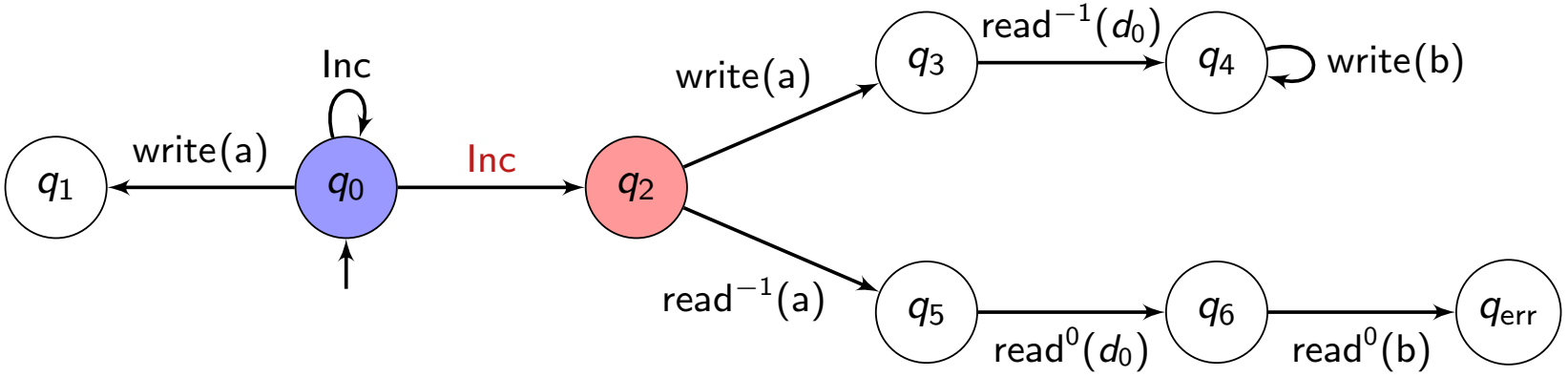
For all q and k, (q, k) is coverable in the abstract semantics if and only if it is coverable in the original semantics.

Back to the example



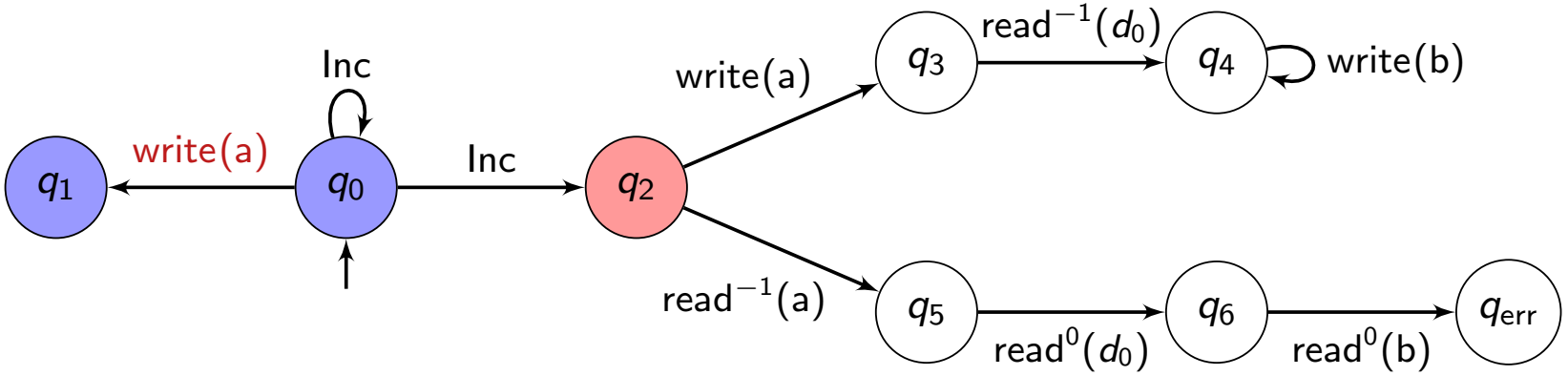
Locations covered : $\{(q_0, 0)\}$
Symbols : $\emptyset$

Back to the example



Locations covered : $\{(q_0, 0), (q_2, 1)\}$
Symbols : $\emptyset$

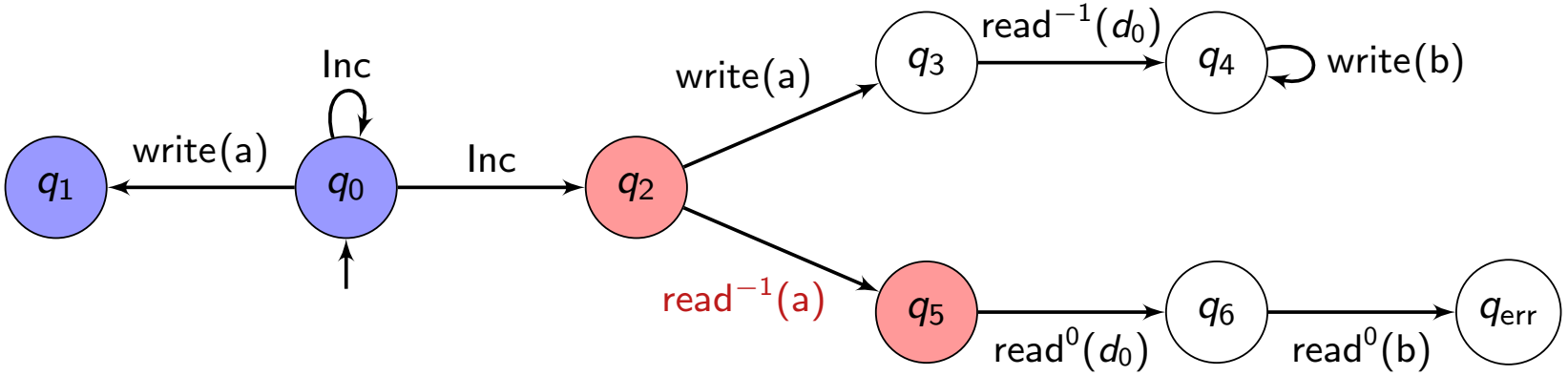
Back to the example



Locations covered : $\{(q_0, 0), (q_2, 1), (q_1, 0)\}$

Symbols : $\text{rg}[0] : \{\text{a}\}$

Back to the example

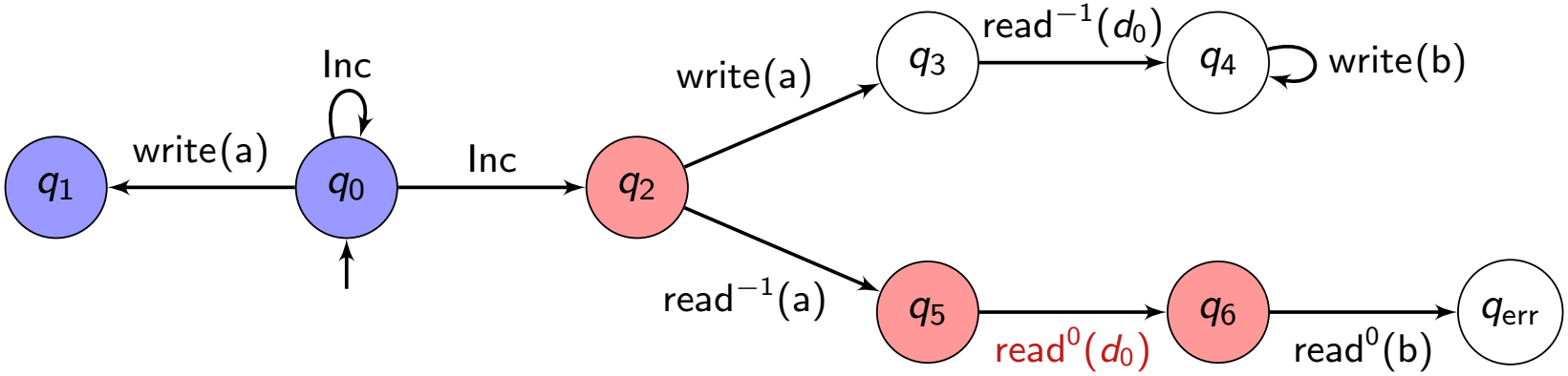


Possible to read a from rg[0]

Locations covered : $\{(q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1)\}$

Symbols : rg[0] : {a}

Back to the example

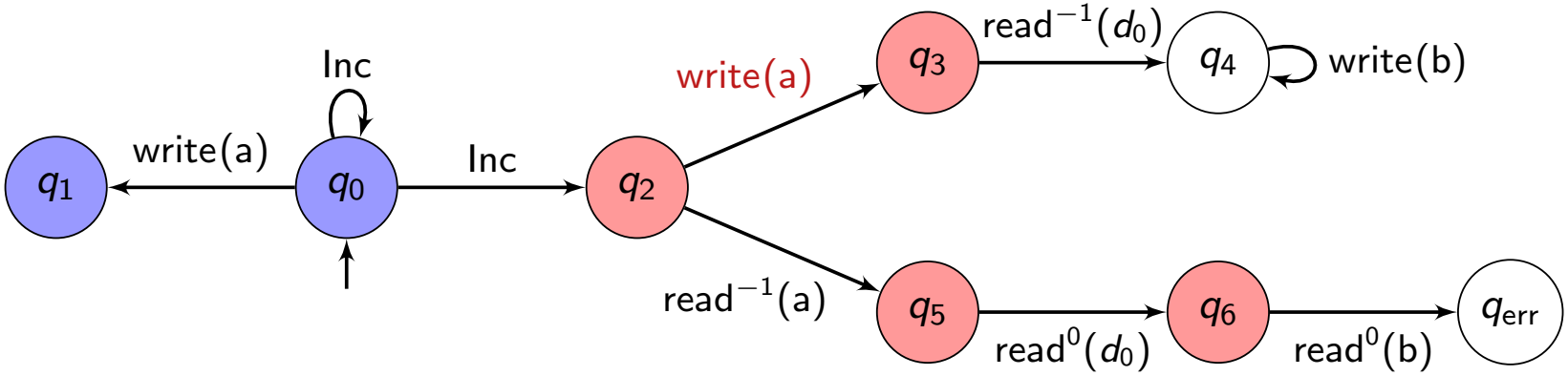


rg[1] does not appear below: it still has value d_0

Locations covered : $\{(q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1), (q_6, 1)\}$

Symbols : rg[0] : {a}

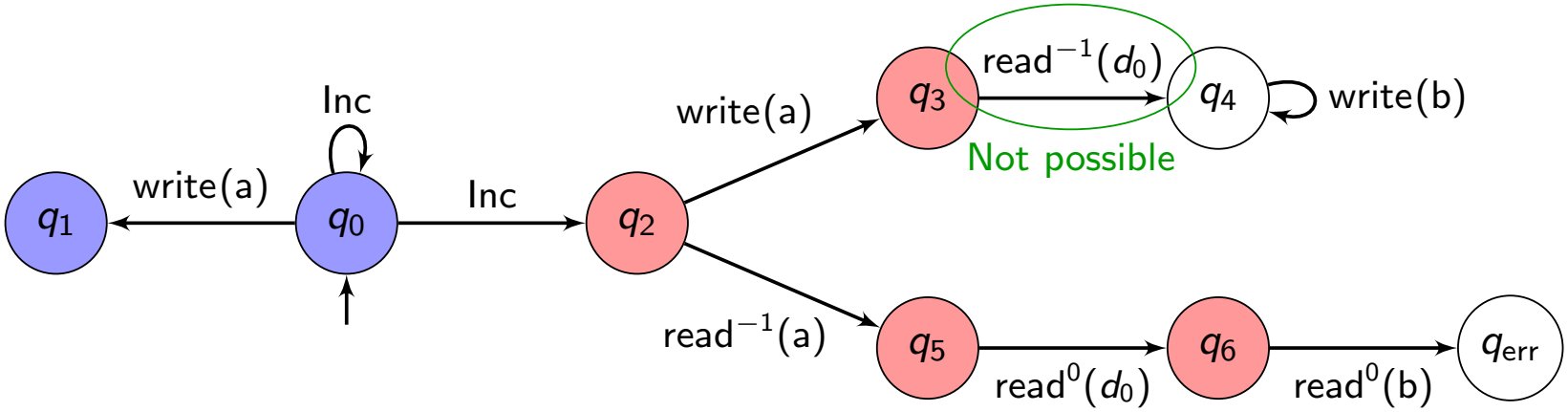
Back to the example



Locations covered : $\{(q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1), (q_6, 1), (q_3, 1)\}$

Symbols : $\text{rg}[0] : \{\text{a}\}, \text{rg}[1] : \{\text{a}\}$

Back to the example



Locations covered : $\{(q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1), (q_6, 1), (q_3, 1)\}$

Symbols : rg[0] : {a}, rg[1] : {a}

rg[0] no longer has value d_0

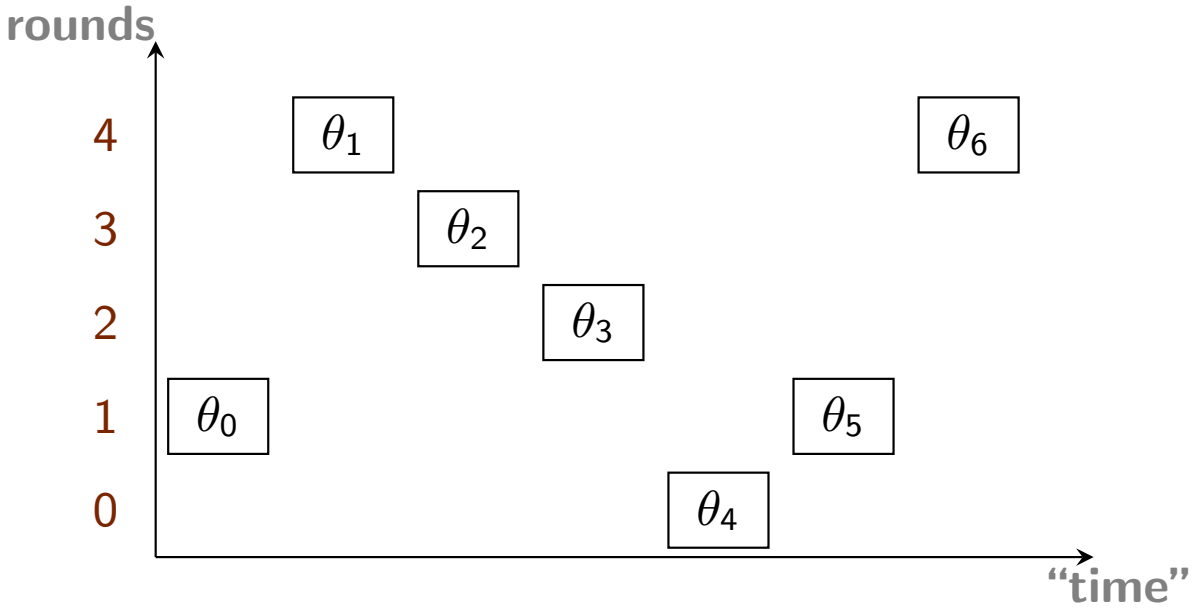
A visual display for executions

Execution: $\sigma_0 \xrightarrow{\theta_0} \sigma_1 \xrightarrow{\theta_1} \sigma_2 \xrightarrow{\theta_2} \sigma_3 \xrightarrow{\theta_3} \sigma_4 \xrightarrow{\theta_4} \sigma_5 \xrightarrow{\theta_5} \sigma_6 \xrightarrow{\theta_6} \sigma_7$

⇓

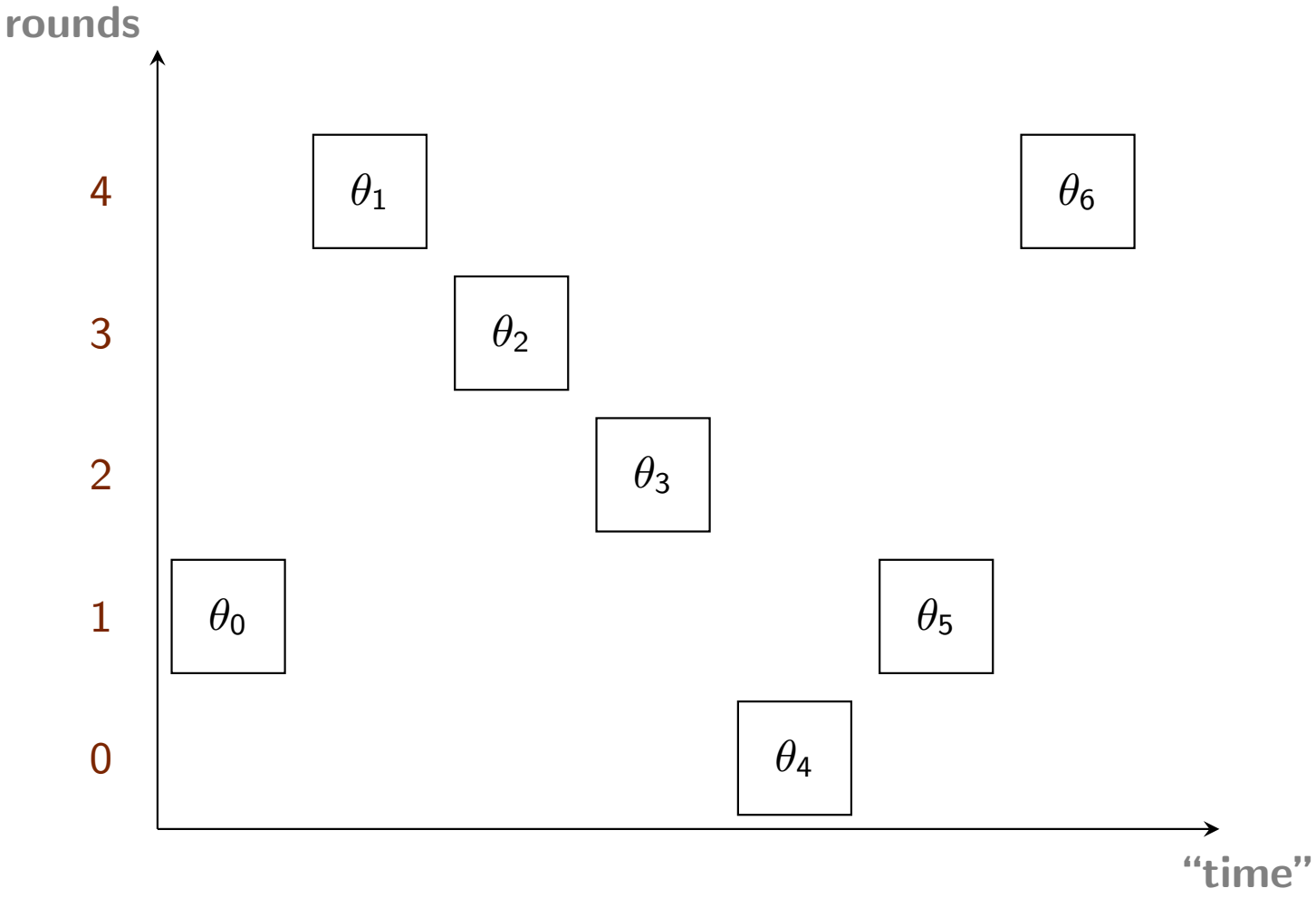
moves:	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
rounds:	1	4	3	2	0	1	4

⇓



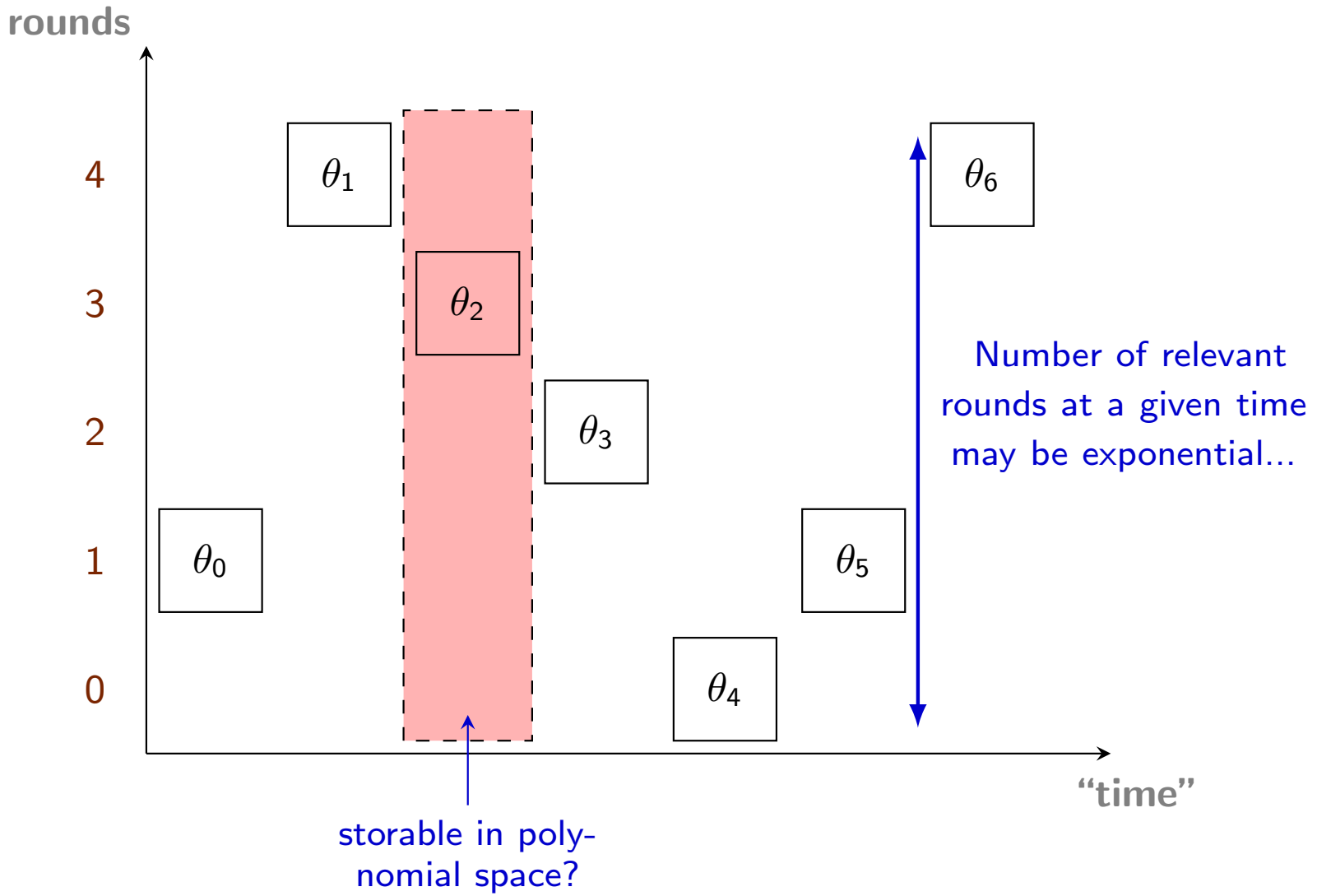
The sliding window

Here v = 1: processes at round k can read from rounds k and $k-1$



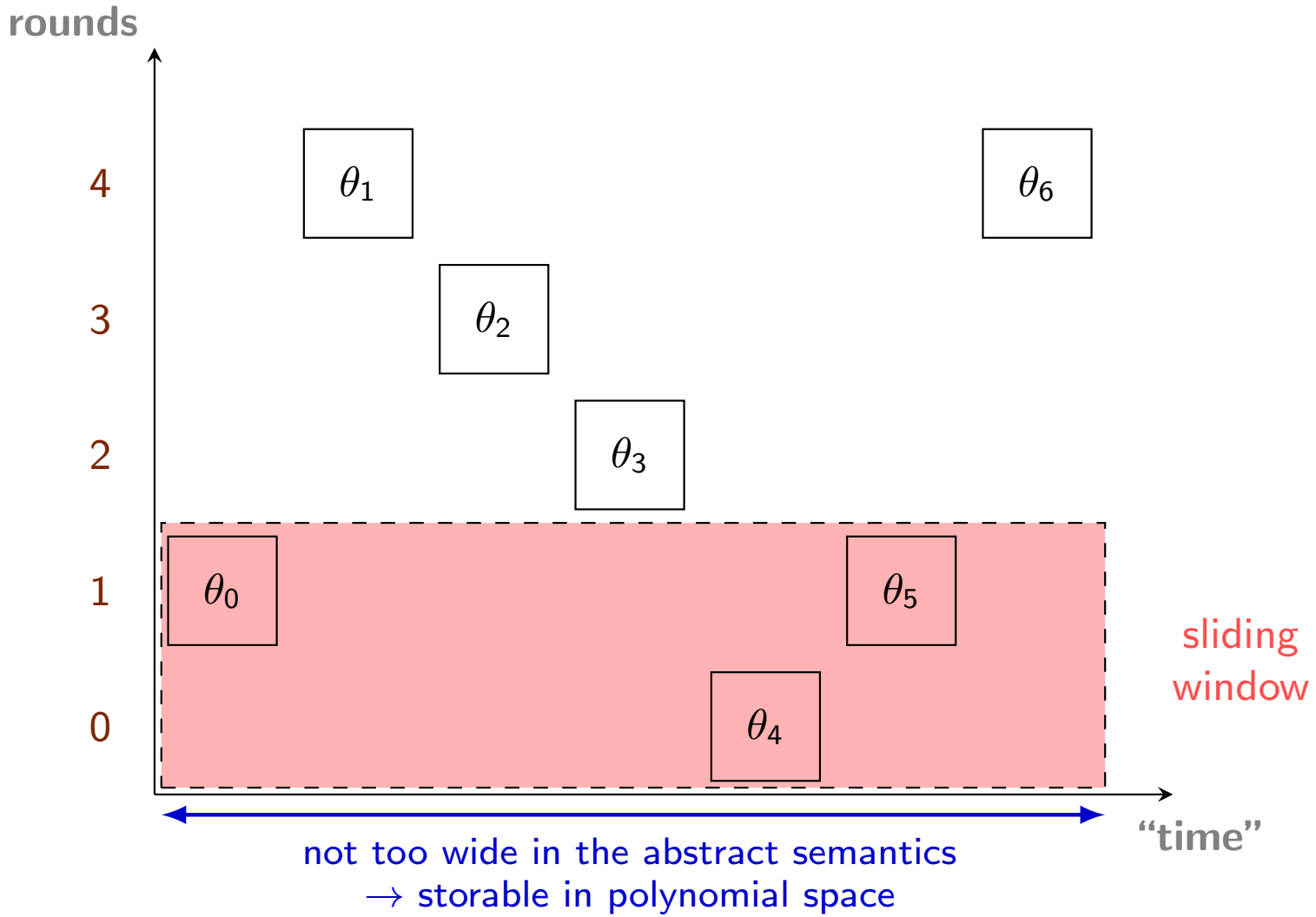
The sliding window

Intuitive idea of proceeding move by move is not working:



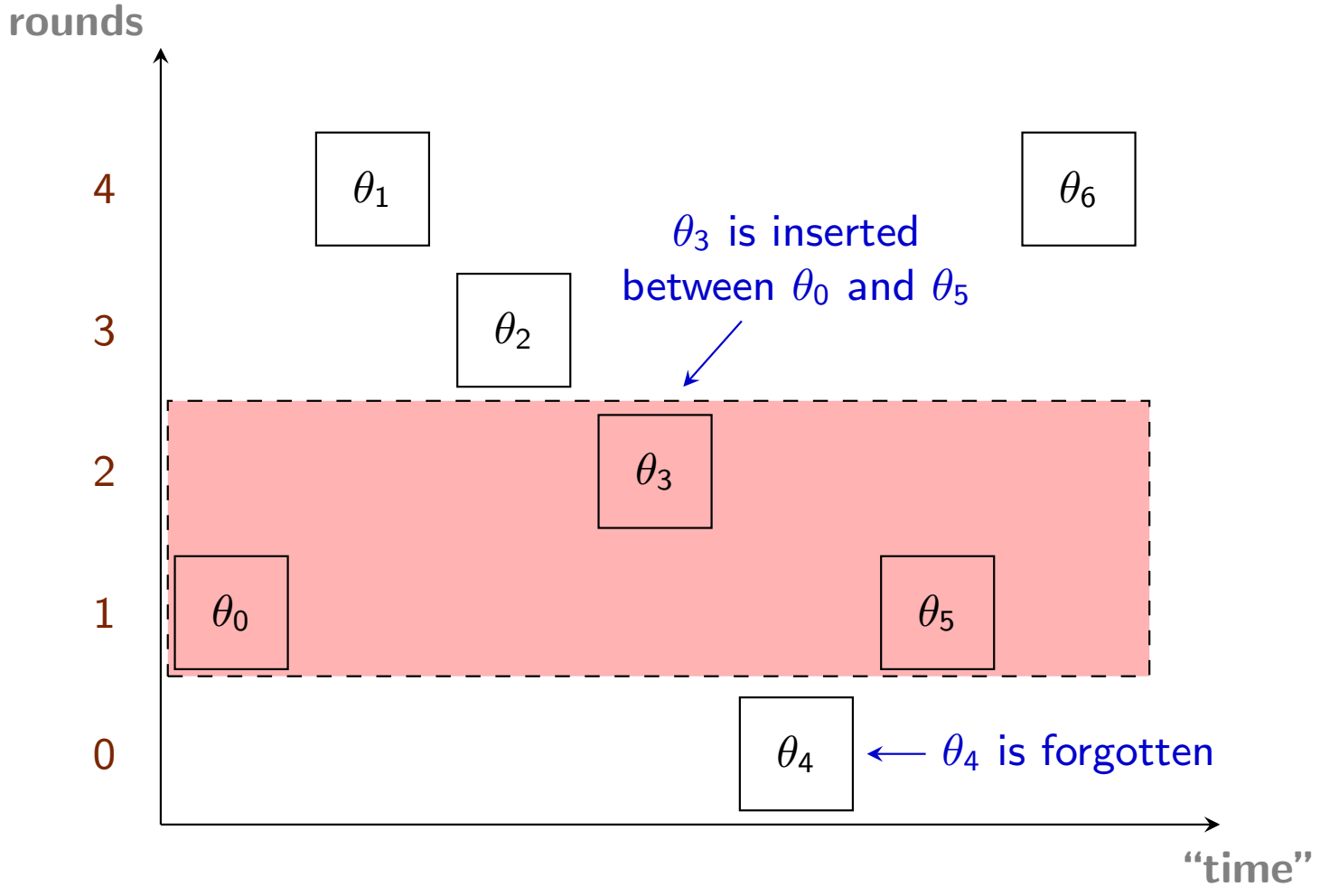
The sliding window

Instead: sliding window along the rounds non-deterministically guessing the execution



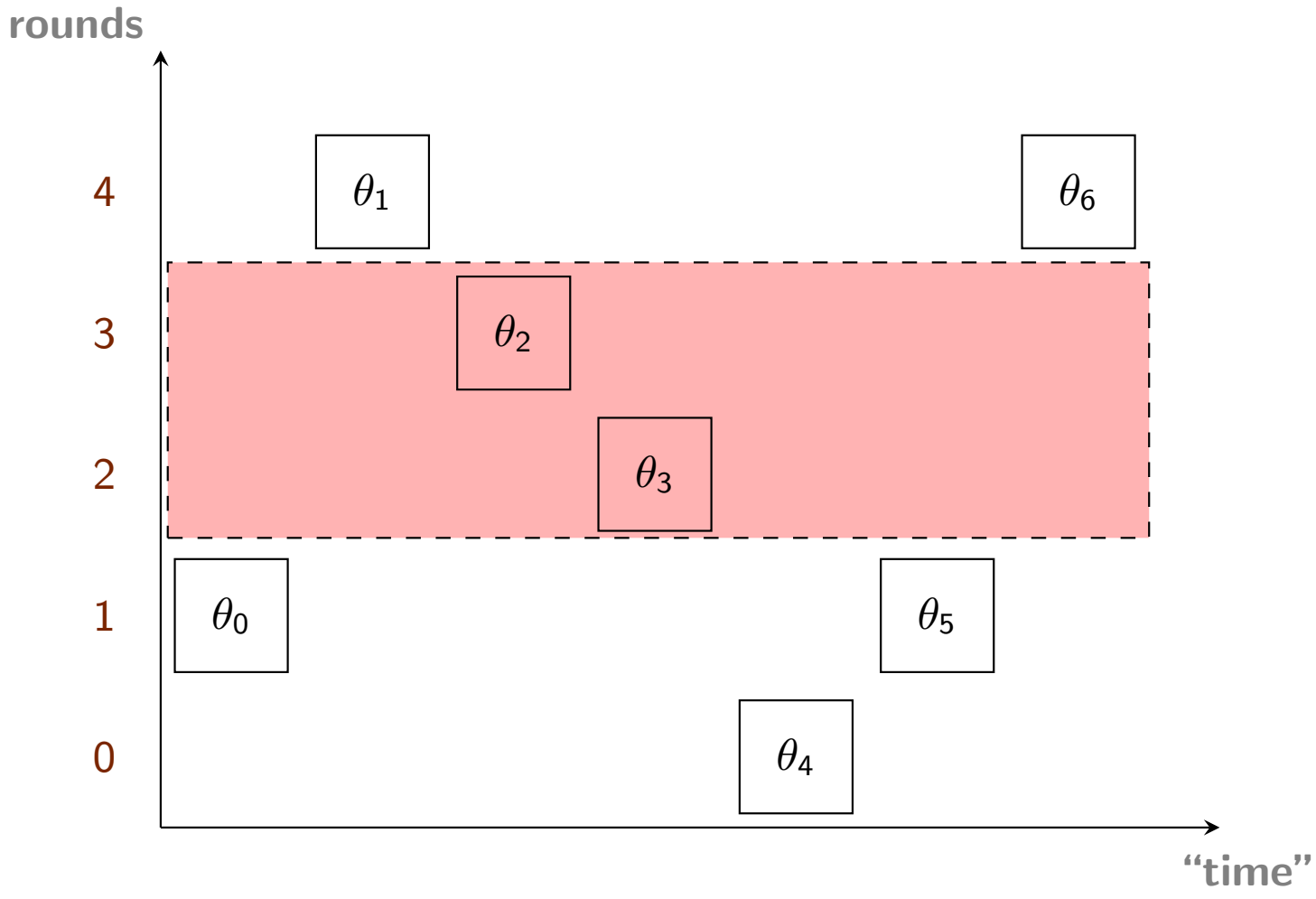
The sliding window

Checking that a move is valid only depends on what happens locally: on moves of rounds 1 and 2 in the case of θ_3.



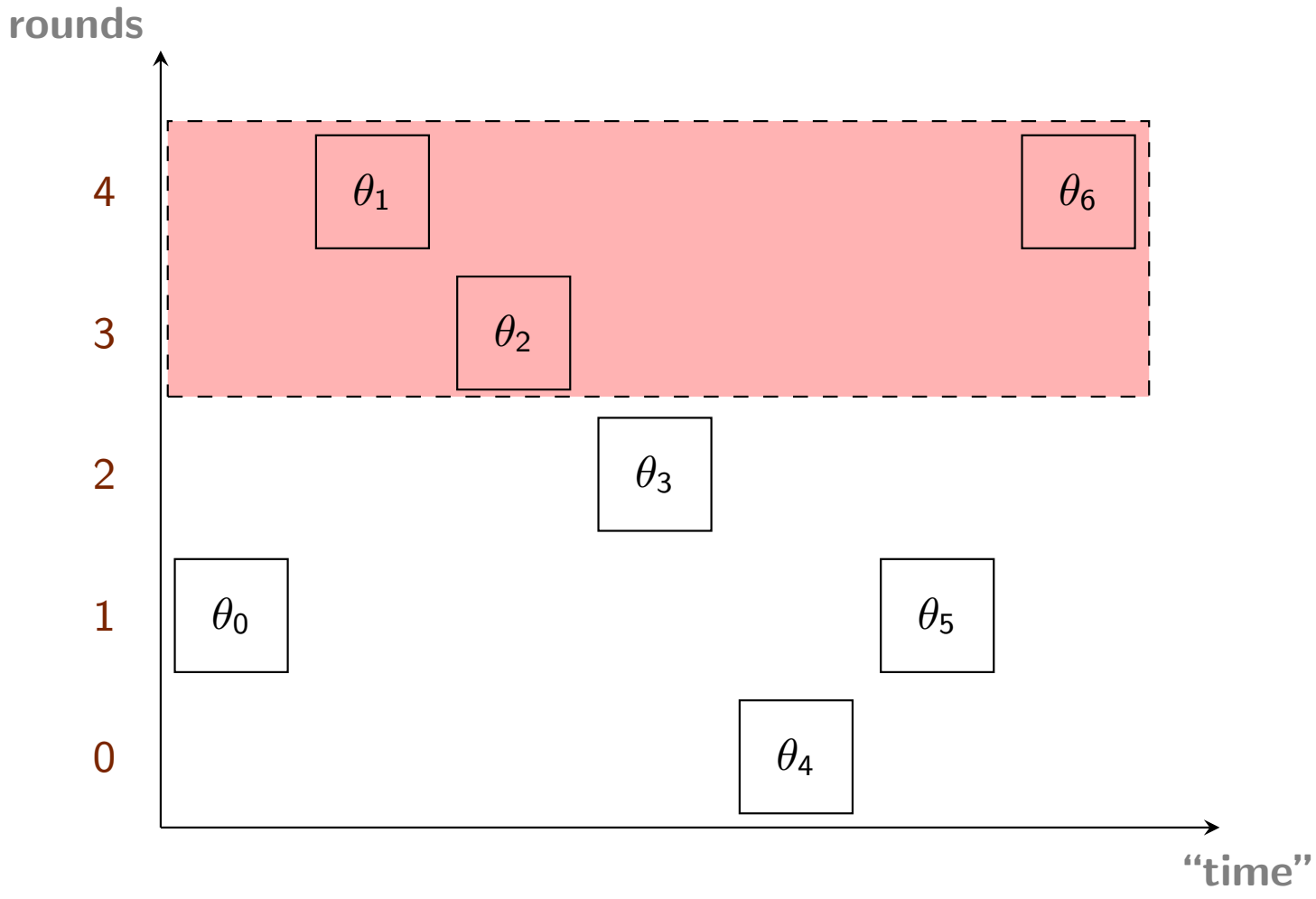
The sliding window

And so on...



The sliding window

And so on...



The polynomial-space algorithm

Outline of the algorithm

At round k:

- non-deterministically insert moves of round $k+1$,
- check that inserted moves are valid (possible with local information),
- forget all about round $k-$v and move to next round.

The polynomial-space algorithm

Outline of the algorithm

At round k:

- non-deterministically insert moves of round $k+1$,
- check that inserted moves are valid (possible with local information),
- forget all about round $k-\mathrm{v}$ and move to next round.

The algorithm stops and returns that the system is not safe if a local configuration reached contains q_{err}.
After an exponential number of steps, the information has looped and the algorithm stops.

Exponential upper bounds

From the algorithm, we derive exponential upper bounds matching the exponential lower bounds:

Exponential upper bound on "cutoff"

There exists an exponential upper bound on the number of processes needed to reach the error state.

Exponential upper bound on the number of rounds

There exists an exponential upper bound on the number of rounds needed to reach the error state.

Ongoing and future work

Summary

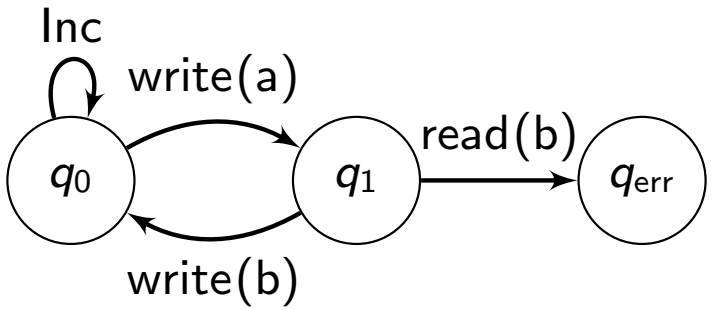
- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithms
- The verification problem of parameterized safety is PSPACE-complete
- The poly-space algorithm relies on a sliding window along the rounds

Future work

- Generalisation to parameterized TARGET ("Is it possible, with enough processes, to make all processes converge towards a given state?") and other similar problems
- Parameterized problems for infinite executions, for exemple INEVITABILITY: do all infinite executions encounter a given state?
- Study almost-sure reachability in round-based register protocols with stochastic schedulers (termination of Aspnes' algorithm)
- Explore its links with classical notions of fairness

Classical notions of fairness are not satisfying here

Classical notions of fairness are not satisfying



- If fairness = "any move that is available infinitely often is taken infinitely often": a fair execution with two processss can stay on q_0 by taking each $((q_0, \text{Inc}, q_0), k)$ twice, never reaching q_{err}.
- If fairness = "any transition that is available infinitely often is taken infinitely often (regardless of round)": a fair execution with two processes can have one of them go to q_1 and back to q_0 on every round, while the other process stays on q_0 forever: transition from q_1 to q_{err} is never enabled.

Fairness and Aspnes' consensus algorithm

Termination of Aspnes' consensus algorithm requires that "a process wins the race", i.e., that processes are not too close in terms of speed.
The two notions of fairness above cannot guarantee this. Neither does: "for all m, any process eventually acts m times in a row while all other processes are idle".
A stochastic scheduler allows to guarantee Aspnes' termination. It could also be proven with some "ad hoc" fairness property.