Self-stabilization

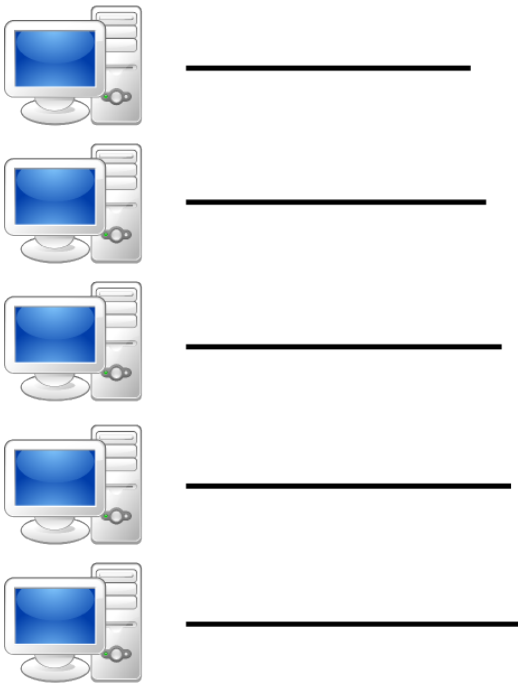
Peva BLANCHARD

30 novembre 2014

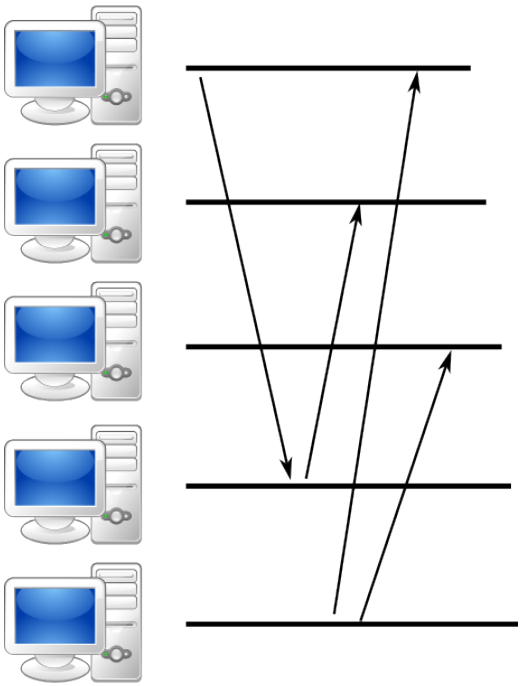
- 1. Basics
- 2. Dijkstra's Mutual Exclusion on Ring
- 3. Intringuing aspects of self-stabilization

- 1. Basics
- 2. Dijkstra's Mutual Exclusion on Ring
- 3. Intringuing aspects of self-stabilization

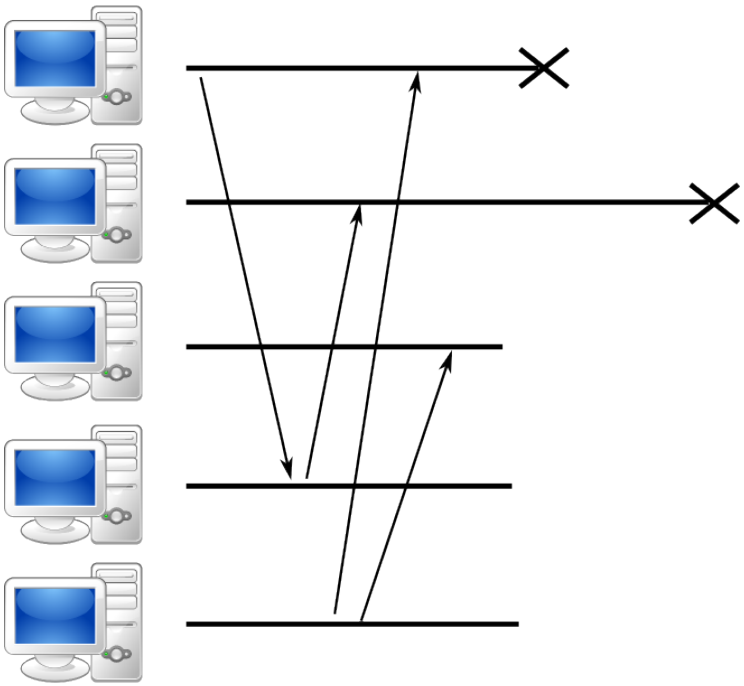
Usual fault-tolerance



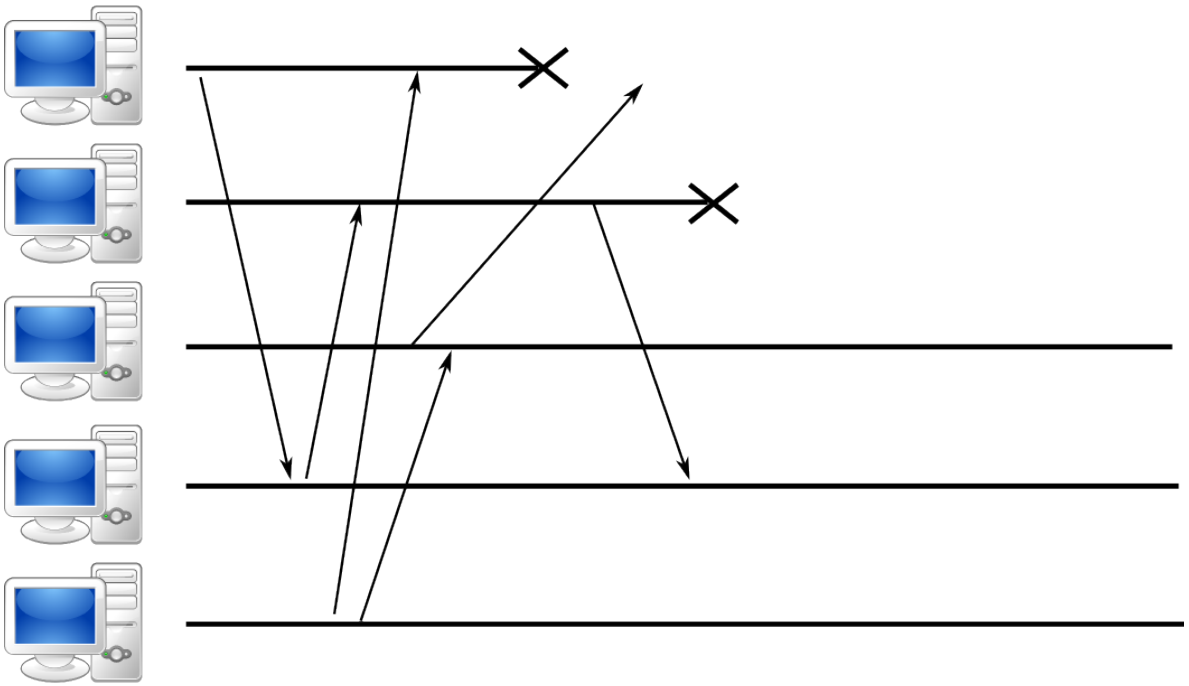
Usual fault-tolerance



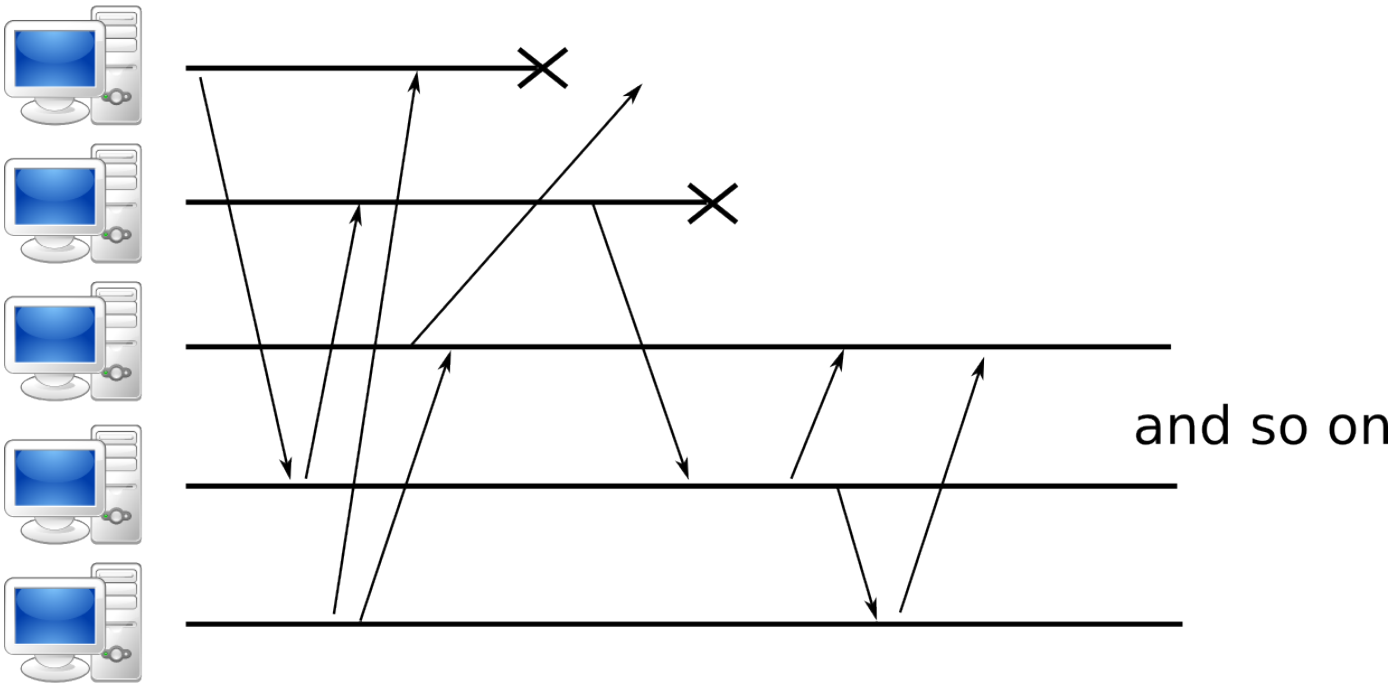
Usual fault-tolerance



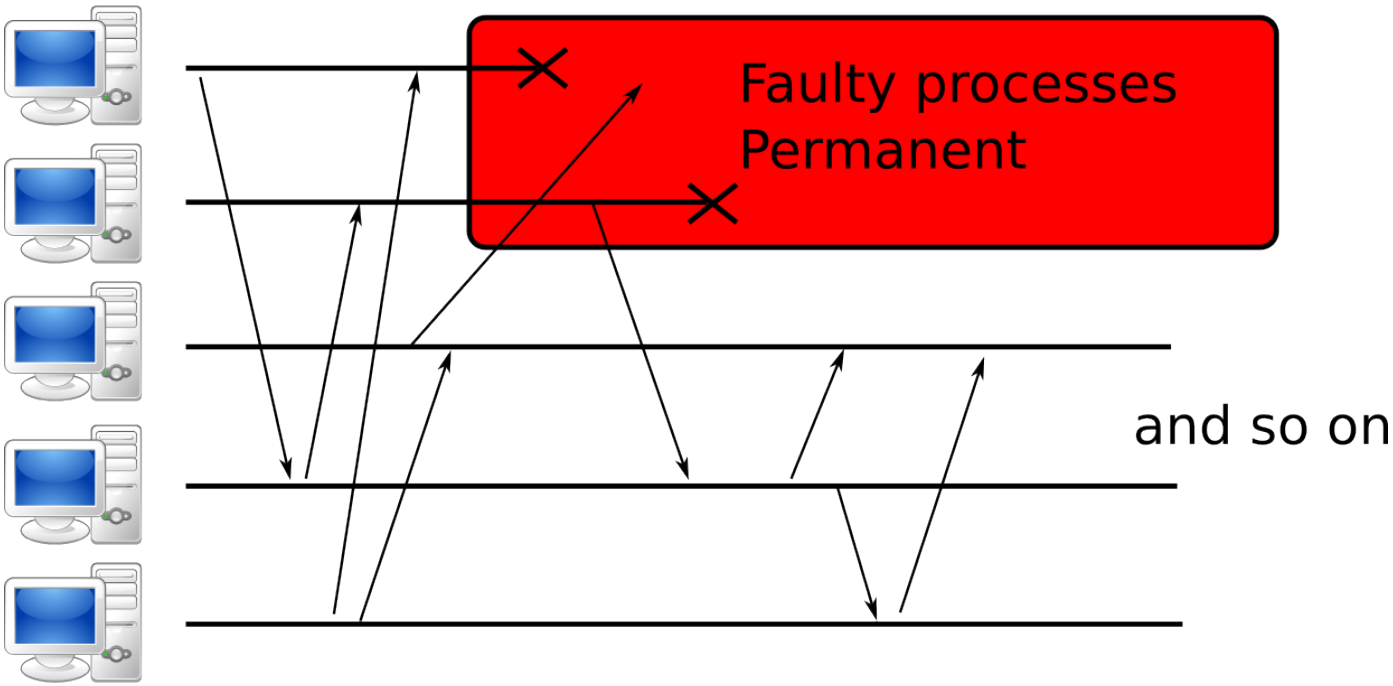
Usual fault-tolerance



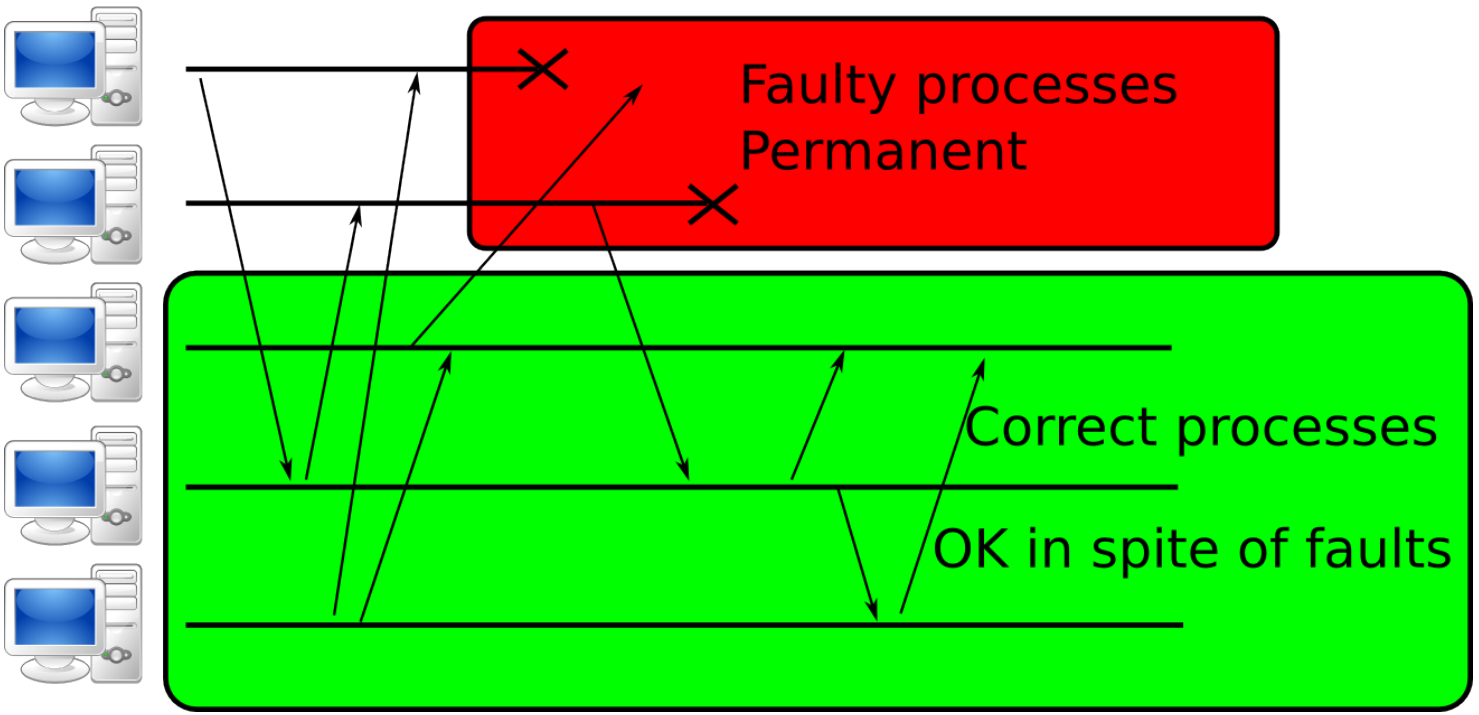
Usual fault-tolerance



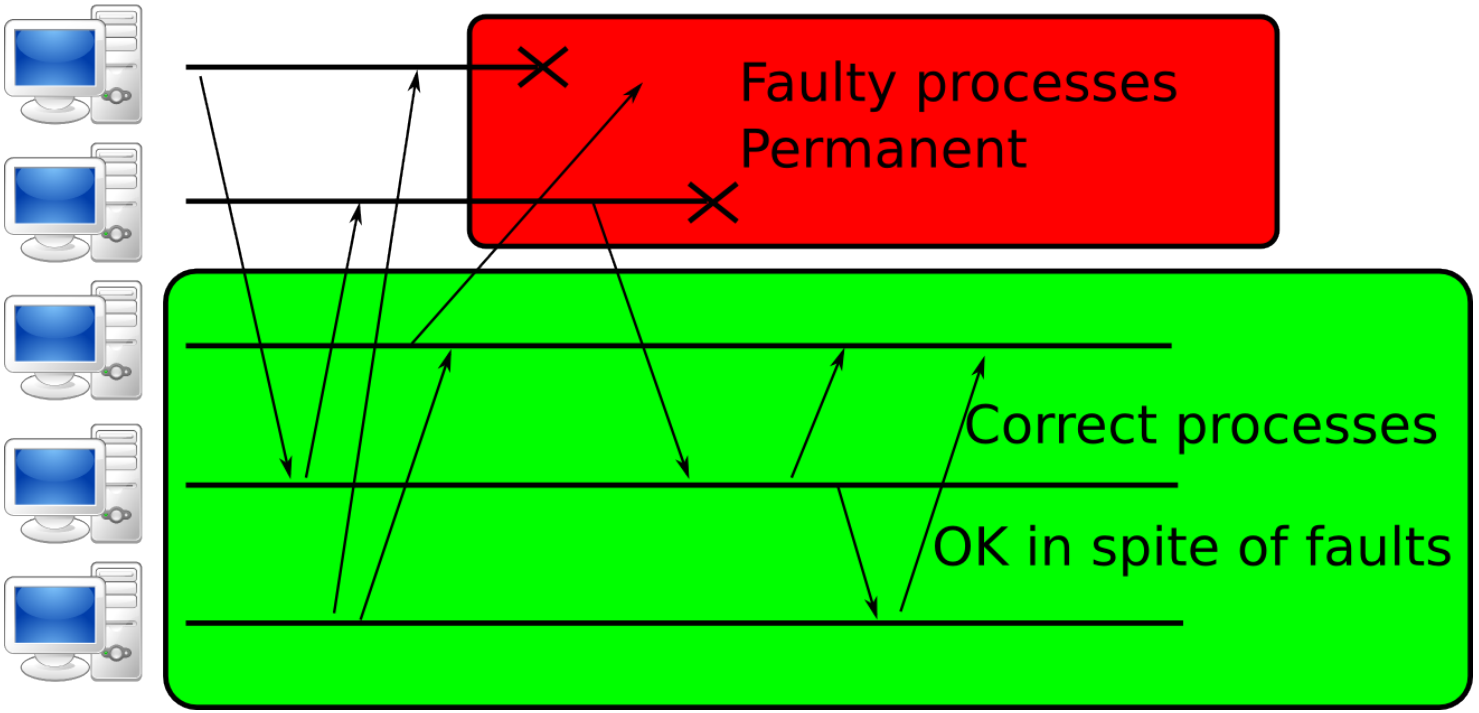
Usual fault-tolerance



Usual fault-tolerance

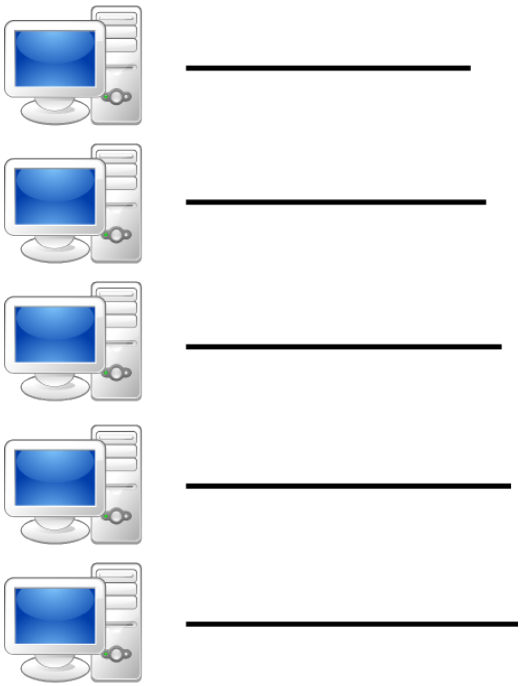


Usual fault-tolerance

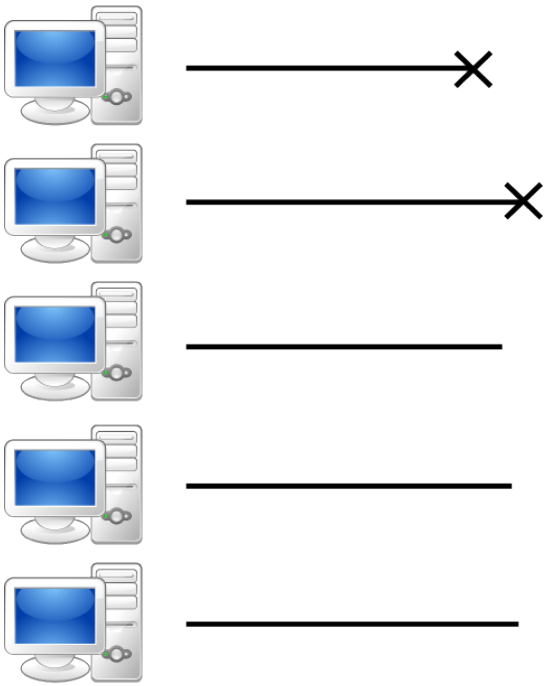


Usually : resiliency bounds $N > 2f+1, 3f + 1$, etc.

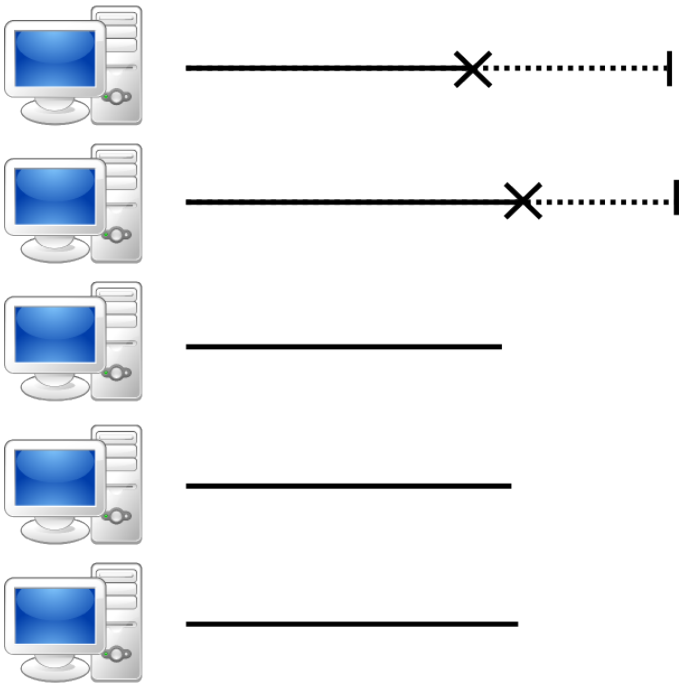
Transient fault



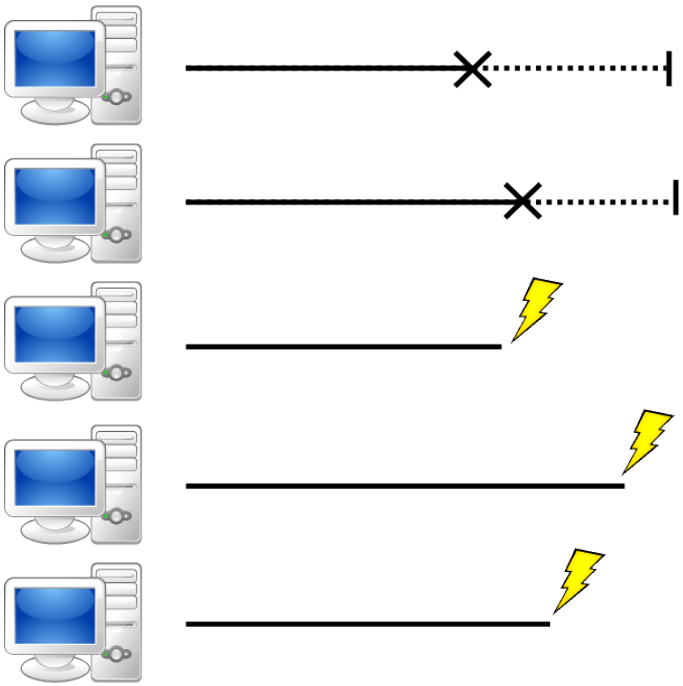
Transient fault



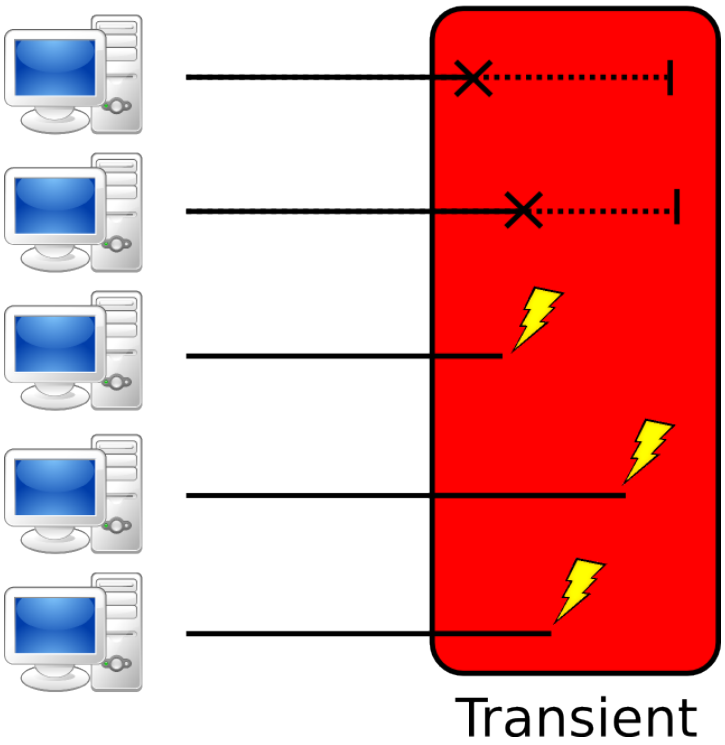
Transient fault



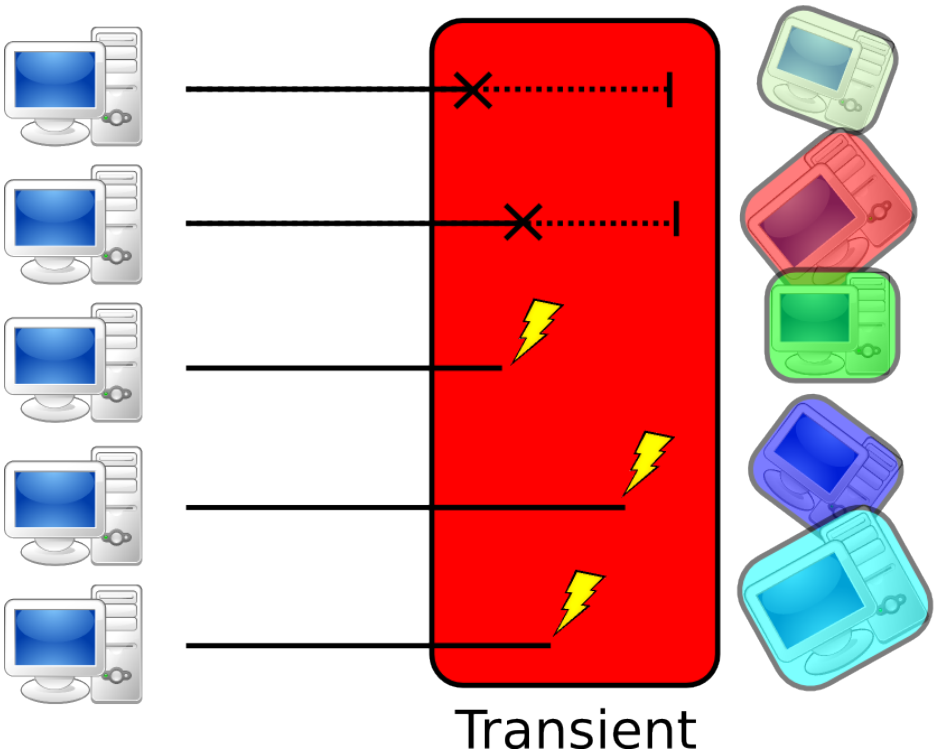
Transient fault



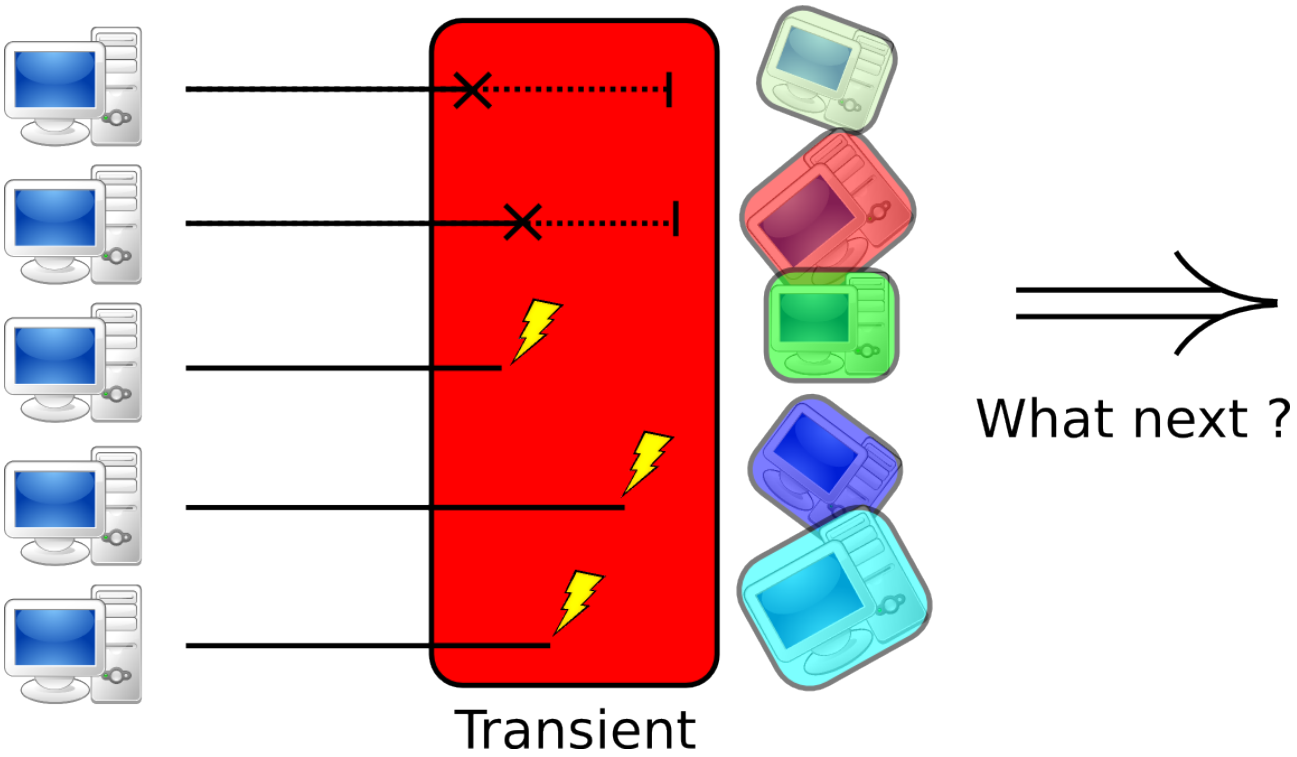
Transient fault



Transient fault

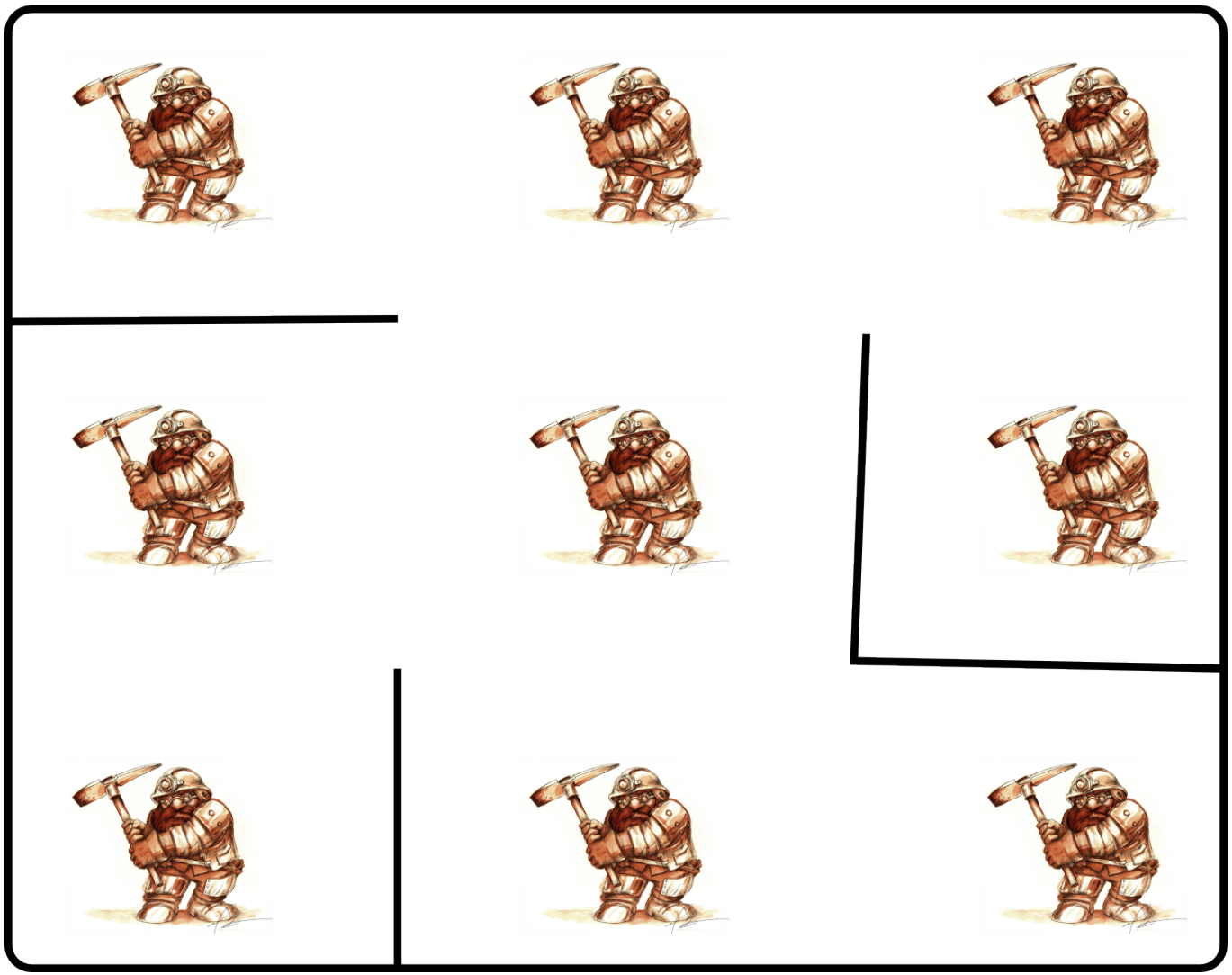


Transient fault

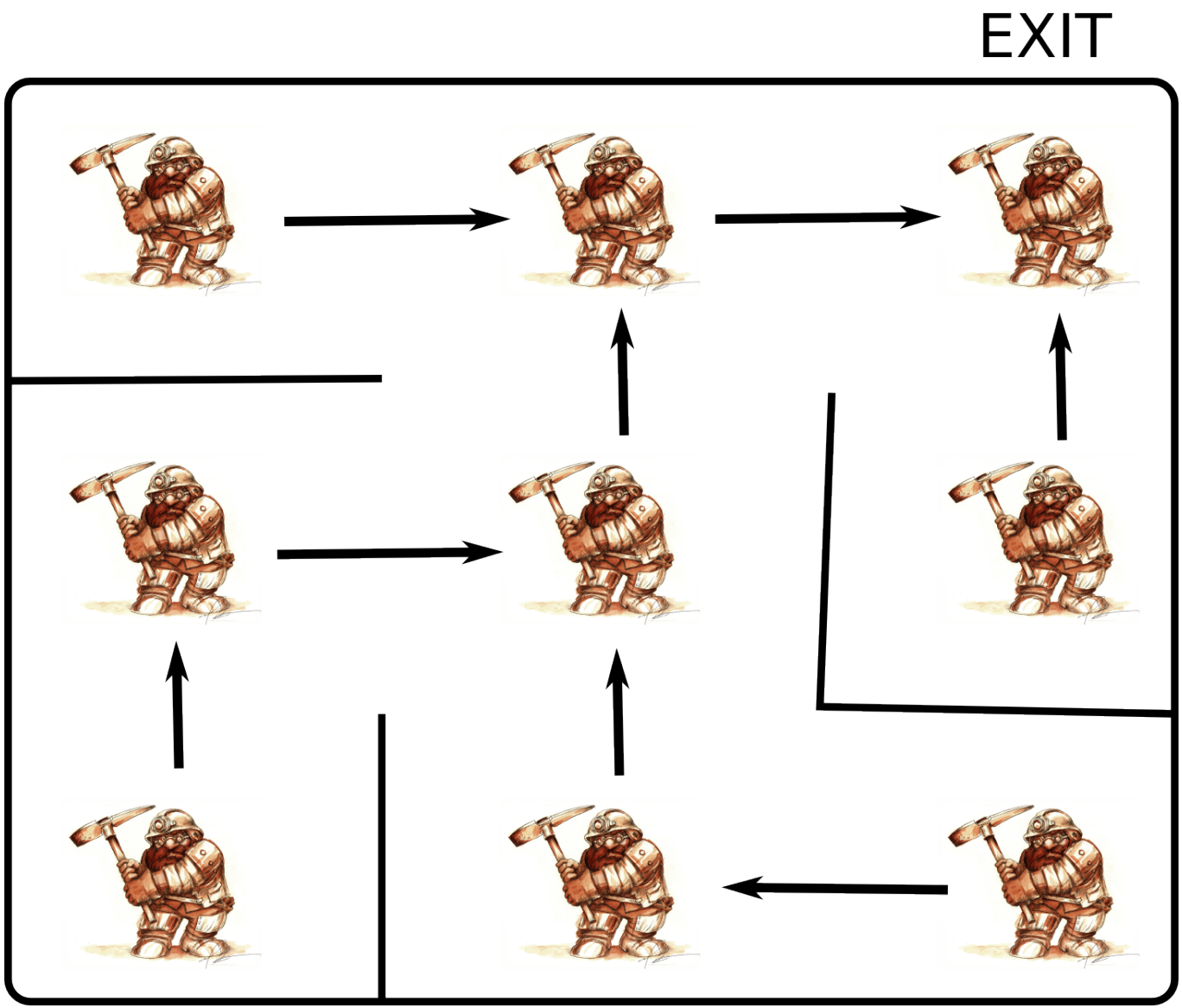


Example

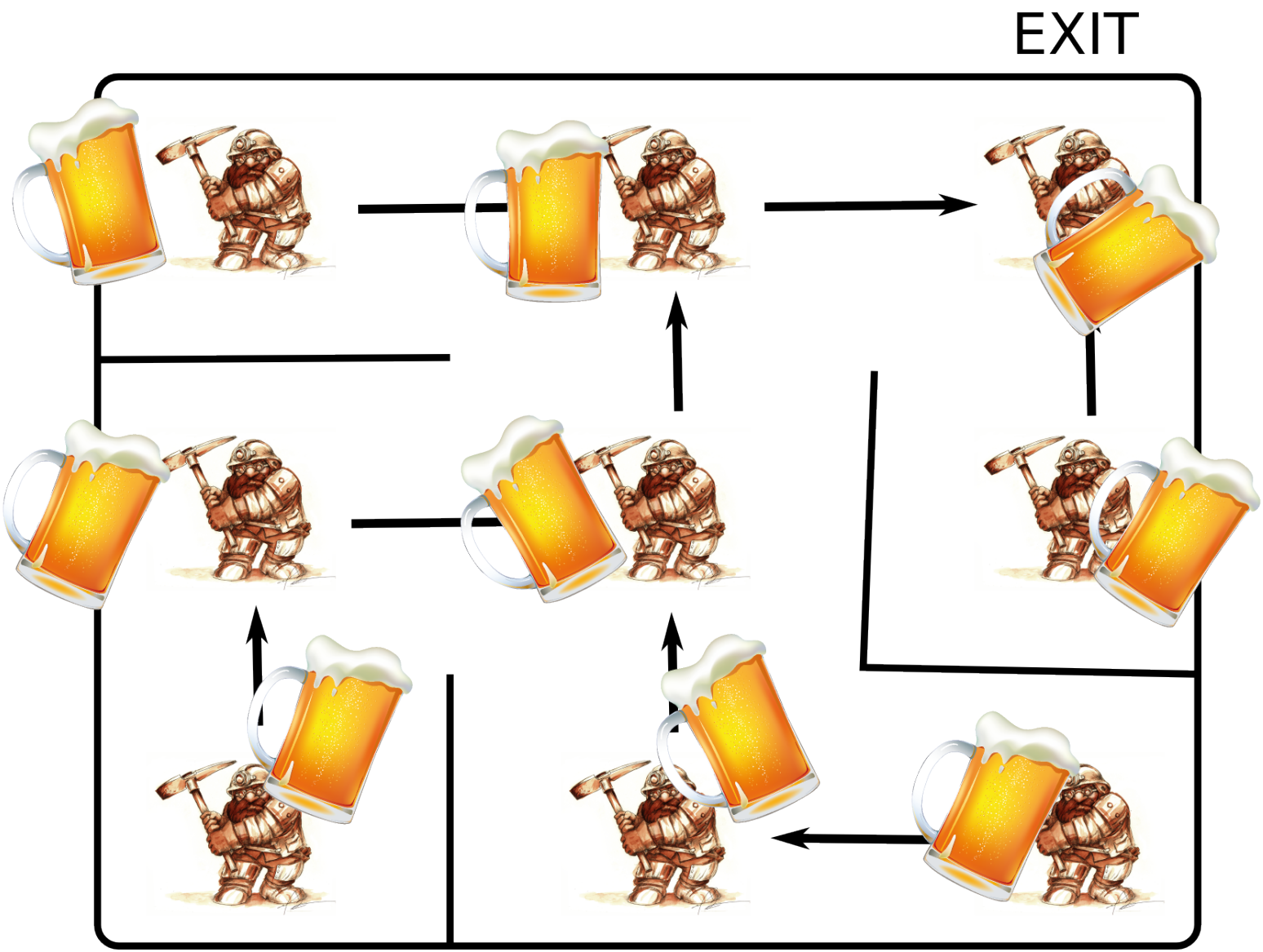
EXIT



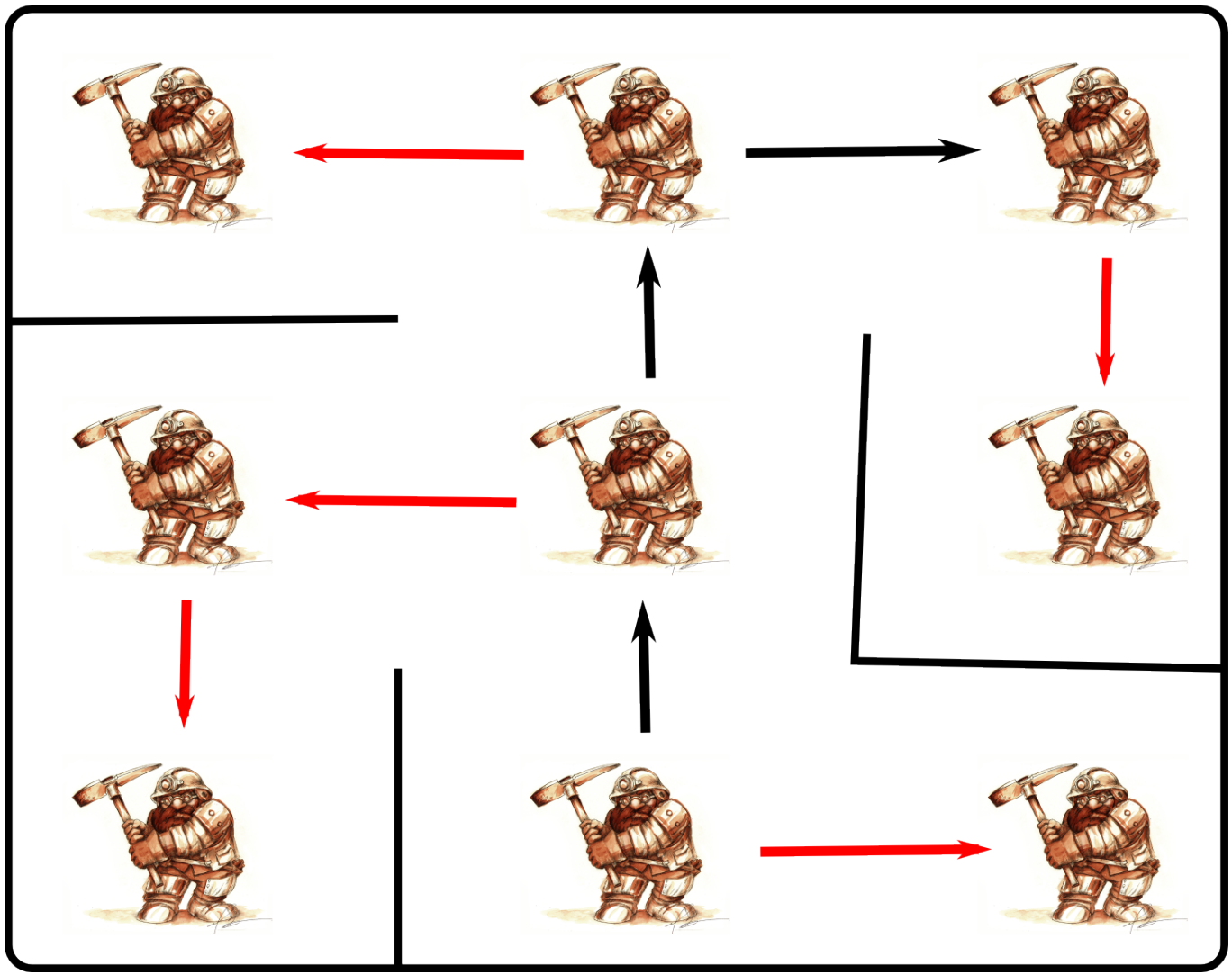
Example



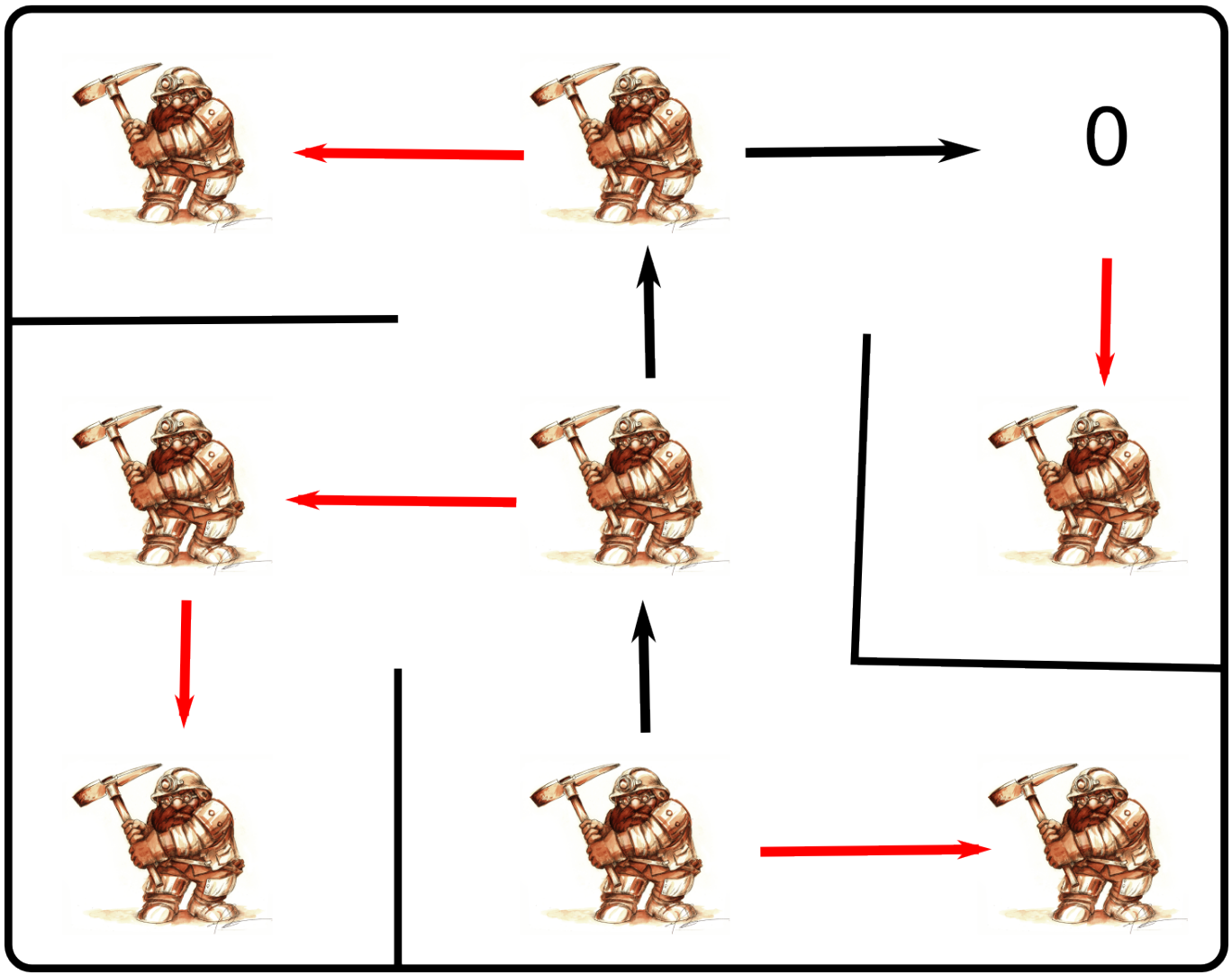
Example



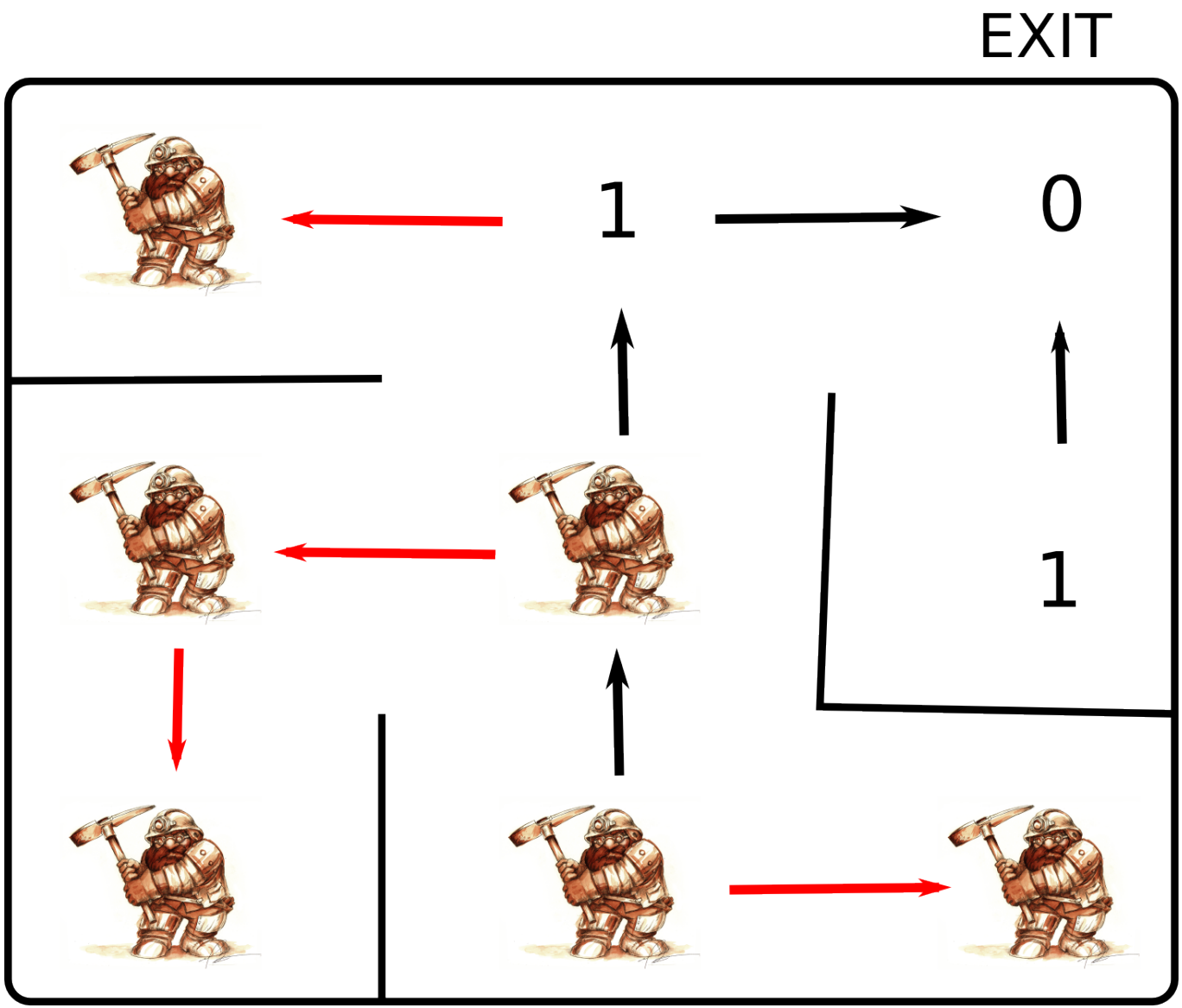
Example



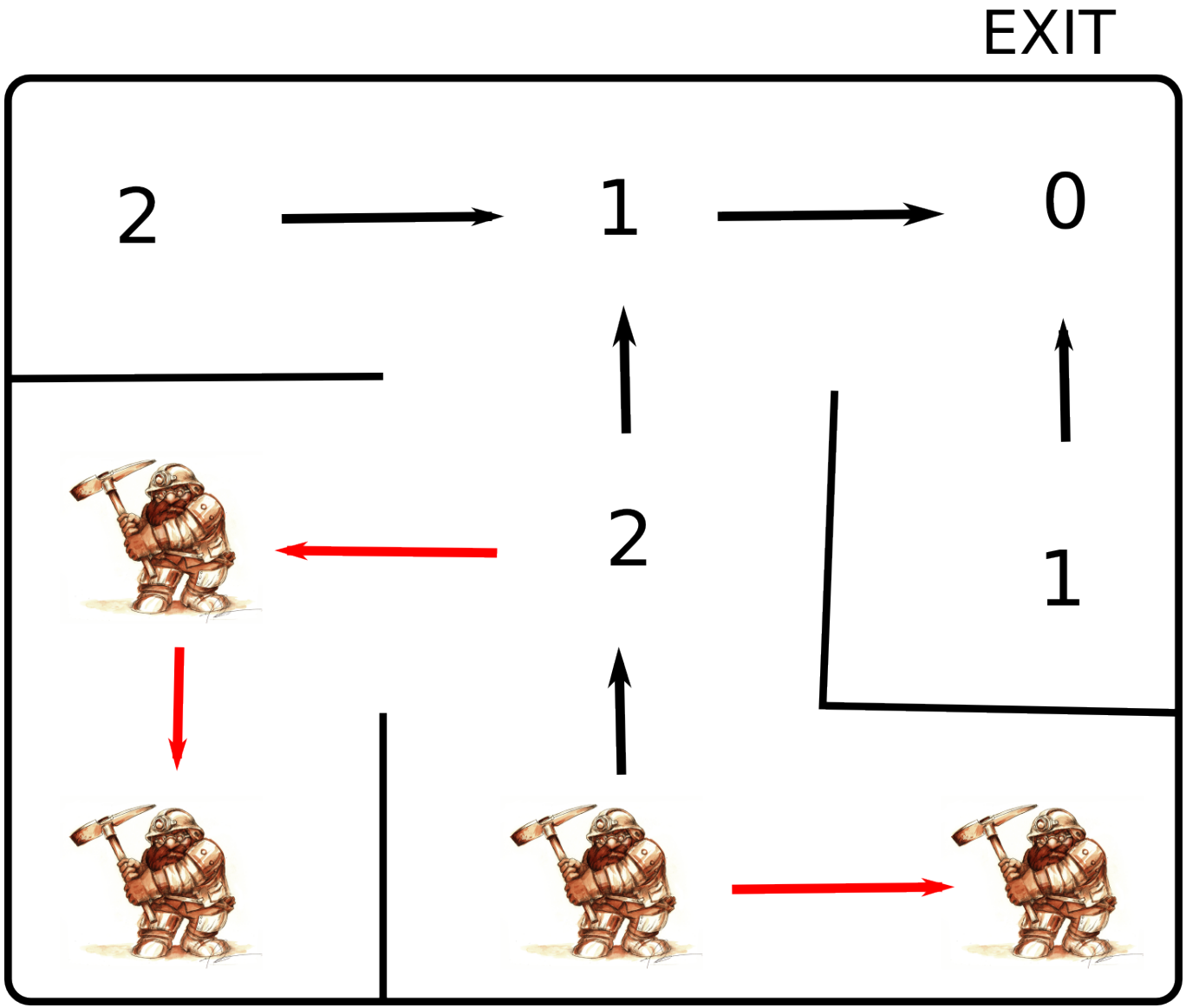
Example



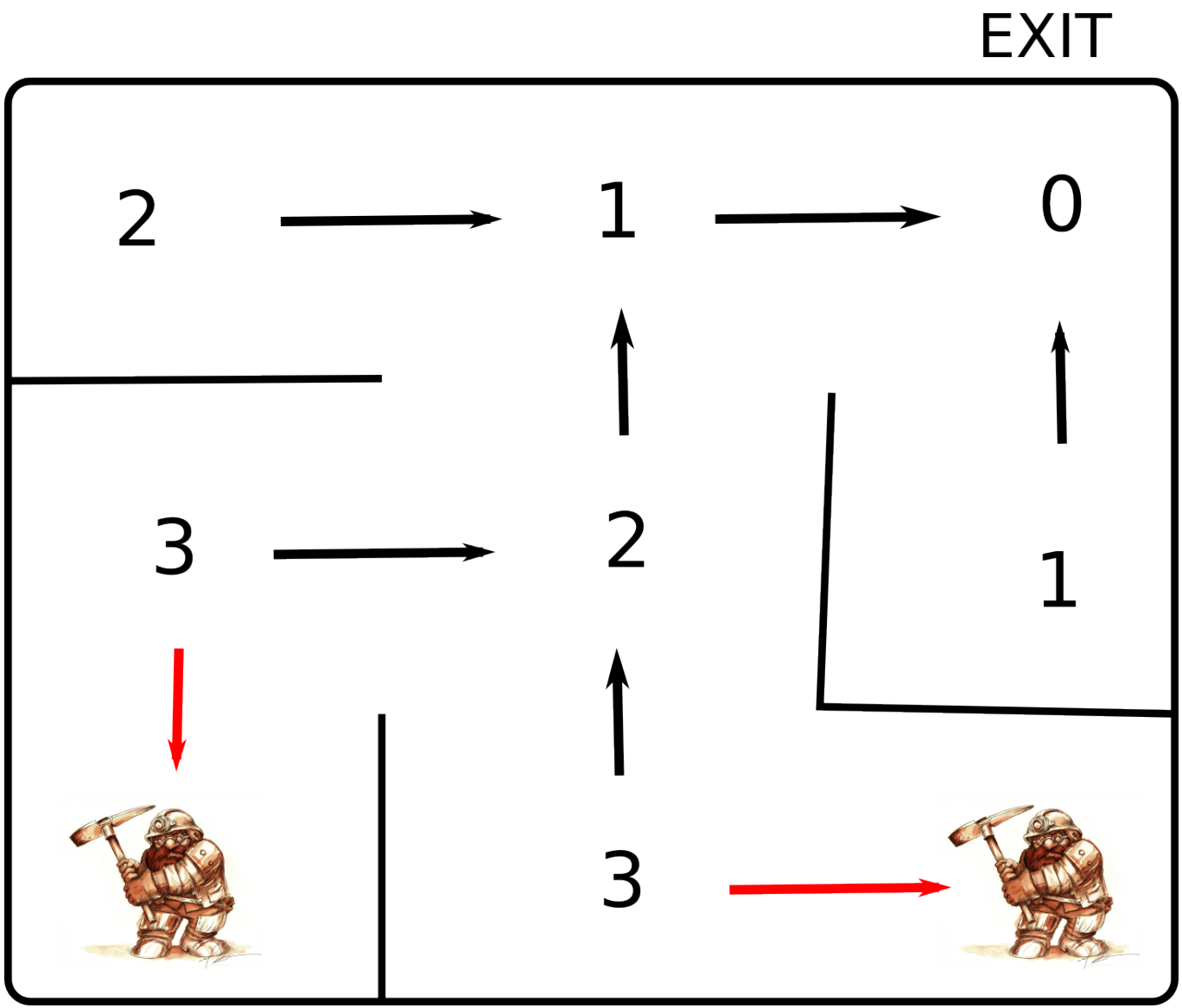
Example



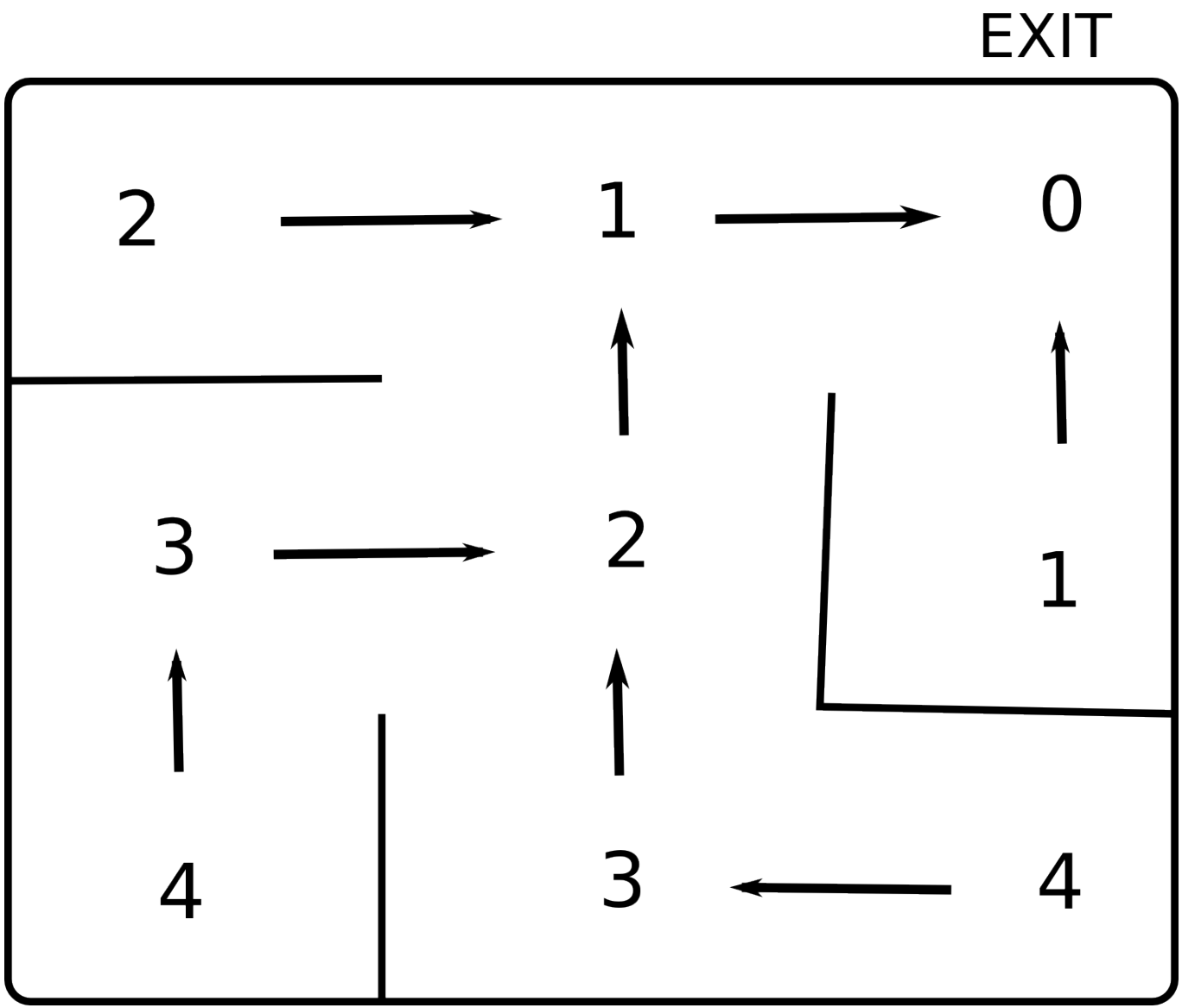
Example



Example



Example



Self-stabilization

- configuration C : local processors memory, register values, message buffer, etc.
- transition $C \rightarrow C'$: some processor takes a step.
- legitimate configurations $\mathcal{L}$: the good ones

Two key aspects

- *(stability).* For any configuration $C \in \mathcal{L}$, if C' is reachable from C then $C' \in \mathcal{L}$
- *(convergence).* For any configuration C, any execution from C reaches a configuration in $\mathcal{L}$

- 1. Basics
- 2. Dijkstra's Mutual Exclusion on Ring
- 3. Intringuing aspects of self-stabilization

Mutual Exclusion [Dijkstra]

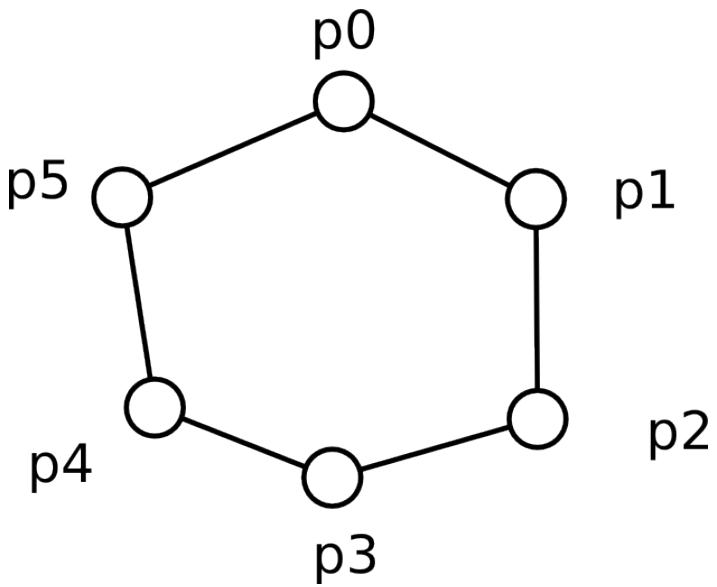
Problem statement

- Each processor may hold or not a token.
- There is a unique token in the system.
- Every processor gets infinitely often the token.

Mutual Exclusion [Dijkstra]

Settings

- Ring of processors $p_0, \ldots, p_{n-1}$.
- Distinguished processor p_0.
- Each processor can (atomically) read the states of its neighbours.
- Fair schedule : every process takes infinitely many steps.



Mutual Exclusion [Dijkstra]

Algorithm variables

- at each p_i : integer variable $x_i \in \{0, \ldots, n\}$ ($n+1$ distinct values !)
- token at p_0 : $x_0 = x_{n-1}$
- token at $p_i \neq p_0$: $x_i \neq x_{i-1}$

Mutual Exclusion [Dijkstra]

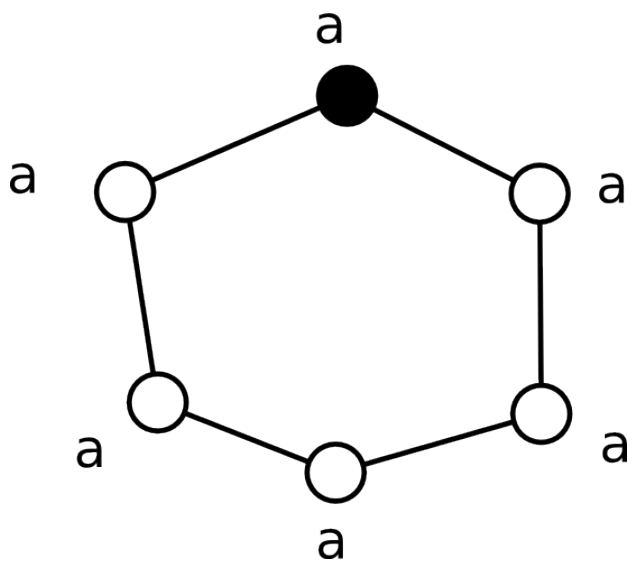
Algorithm

- at p_0 : if $x_0 = x_{n-1}$ then $x_0 \leftarrow x_0 + 1 \mod (n+1)$
- at p_i : if $x_i \neq x_{i-1}$ then $x_i \leftarrow x_{i-1} \mod (n+1)$

Mutual Exclusion [Dijkstra]

Legitimate configurations $\mathcal{L}$

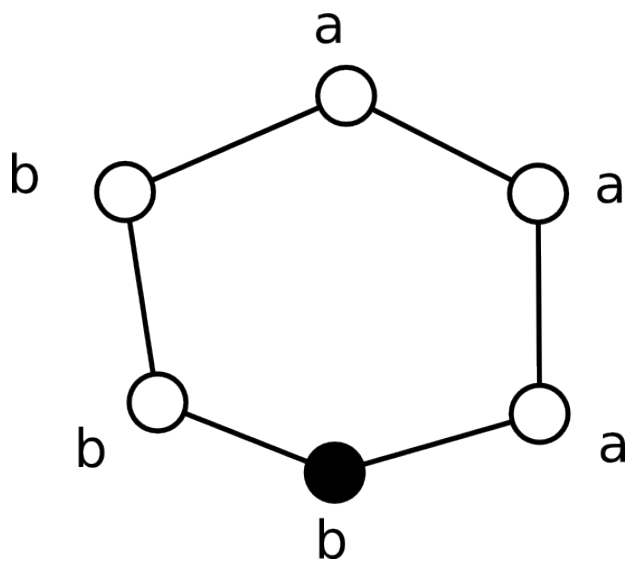
- all the same value $x_0 = \cdots = x_{n-1}$: unique token at p_0
- two values $x_0 = \cdots = x_{i-1} = a$, $x_i = \cdots = x_{n-1} = b$: unique token at p_i.



Mutual Exclusion [Dijkstra]

Legitimate configurations $\mathcal{L}$

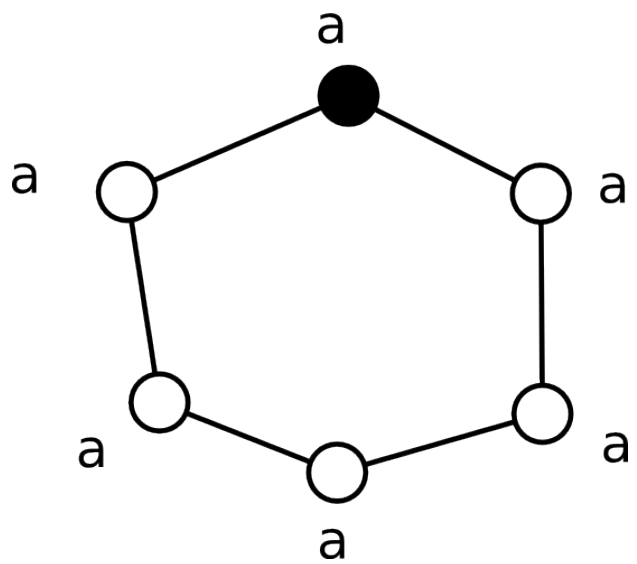
- all the same value $x_0 = \cdots = x_{n-1}$: unique token at p_0
- two values $x_0 = \cdots = x_{i-1} = a$, $x_i = \cdots = x_{n-1} = b$: unique token at p_i.



Mutual Exclusion [Dijkstra]

$\mathcal{L}$ is stable

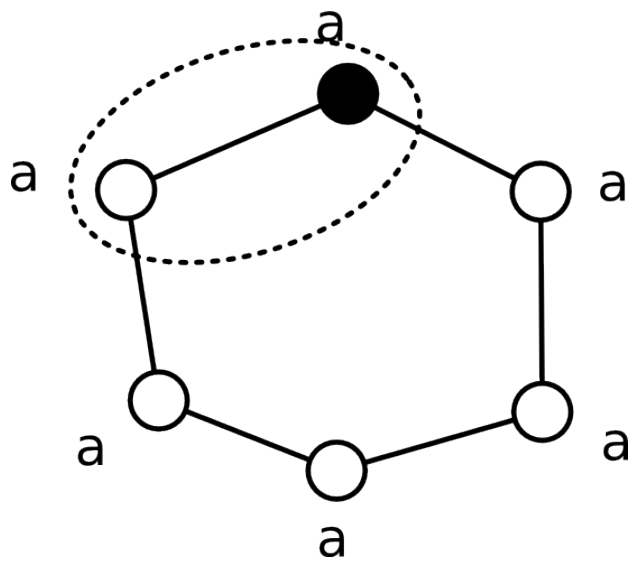
$$\forall C \in \mathcal{L},\ C \to C' \Rightarrow C' \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

$\mathcal{L}$ is stable

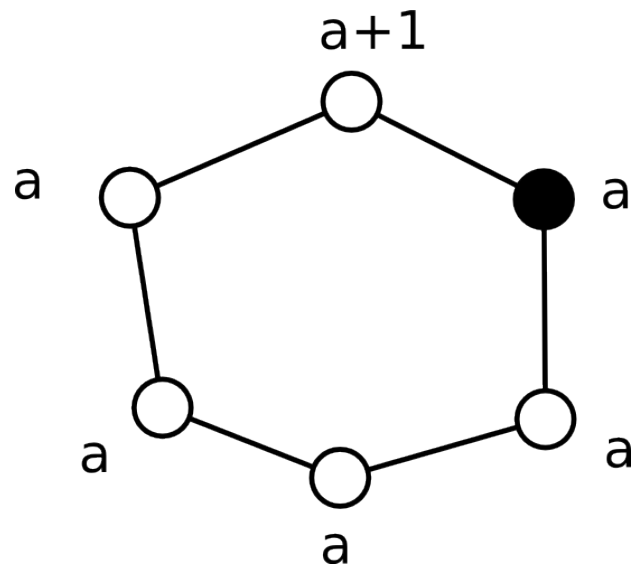
$$\forall C \in \mathcal{L},\ C \to C' \Rightarrow C' \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

$\mathcal{L}$ is stable

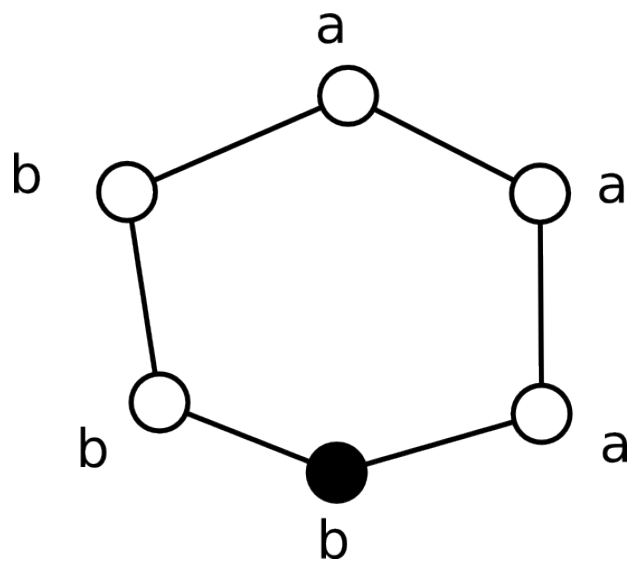
$$\forall C \in \mathcal{L},\ C \to C' \Rightarrow C' \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

$\mathcal{L}$ is stable

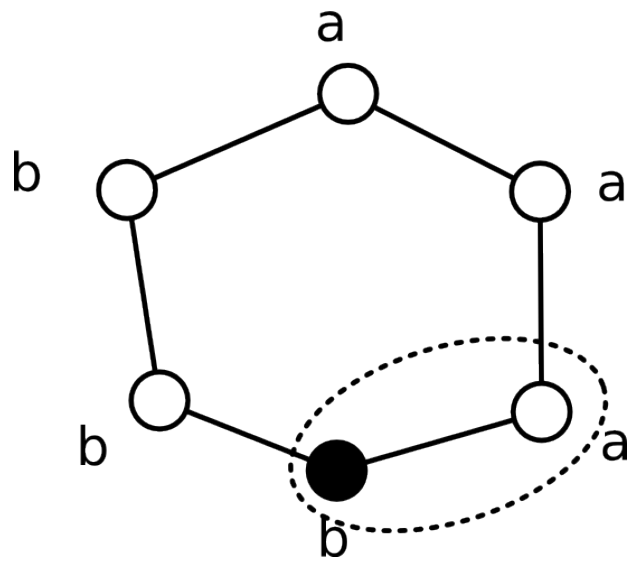
$$\forall C \in \mathcal{L},\ C \to C' \Rightarrow C' \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

$\mathcal{L}$ is stable

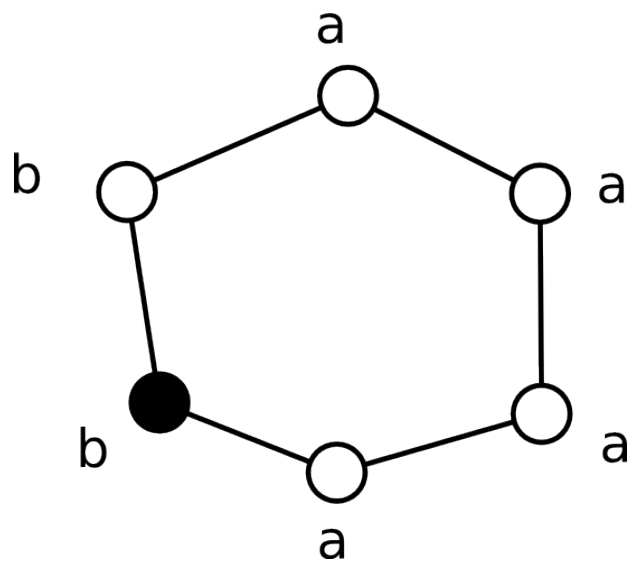
$$\forall C \in \mathcal{L},\ C \to C' \Rightarrow C' \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

$\mathcal{L}$ is stable

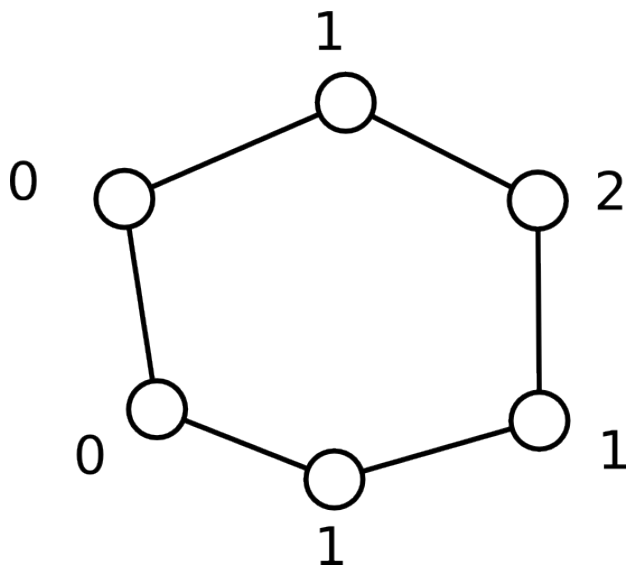
$$\forall C \in \mathcal{L},\ C \to C' \Rightarrow C' \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

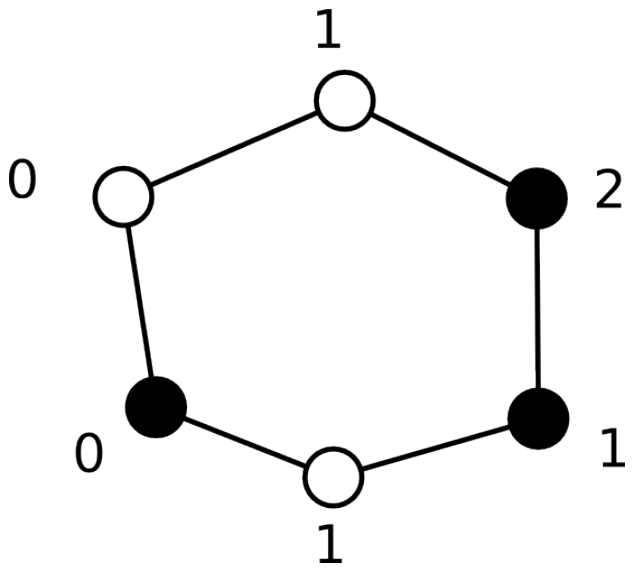
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

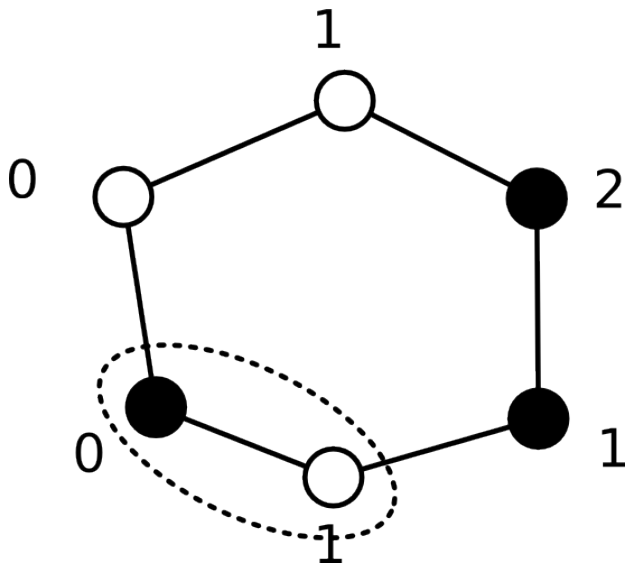
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

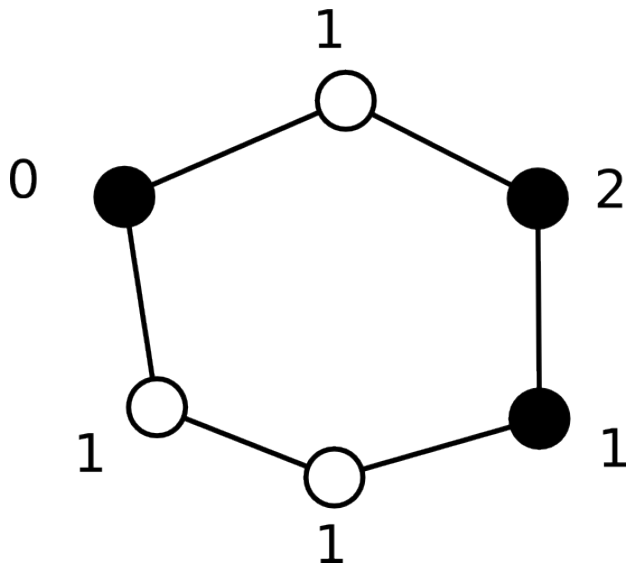
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

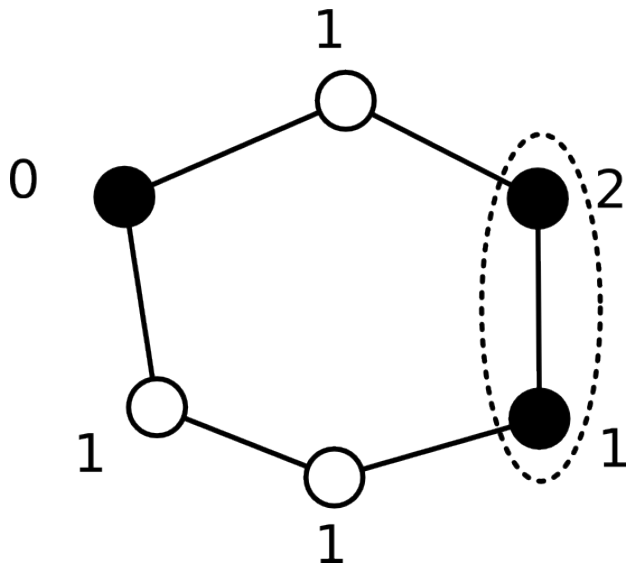
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

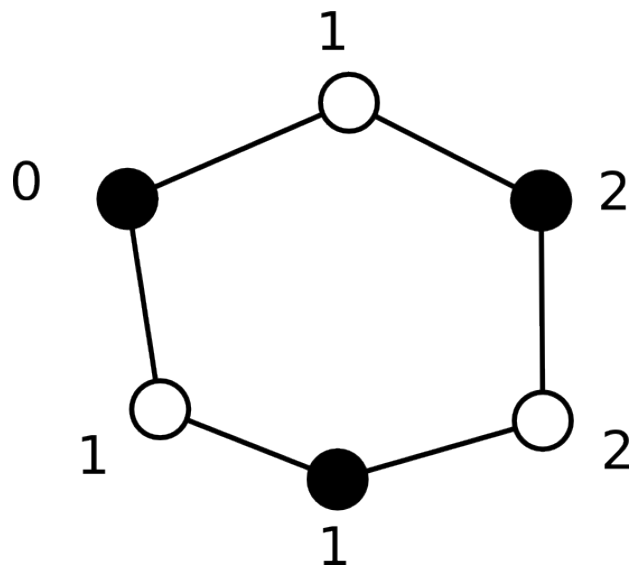
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

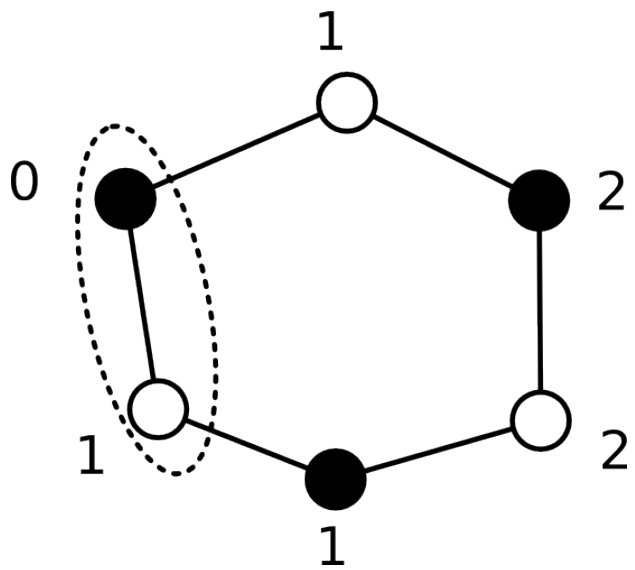
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

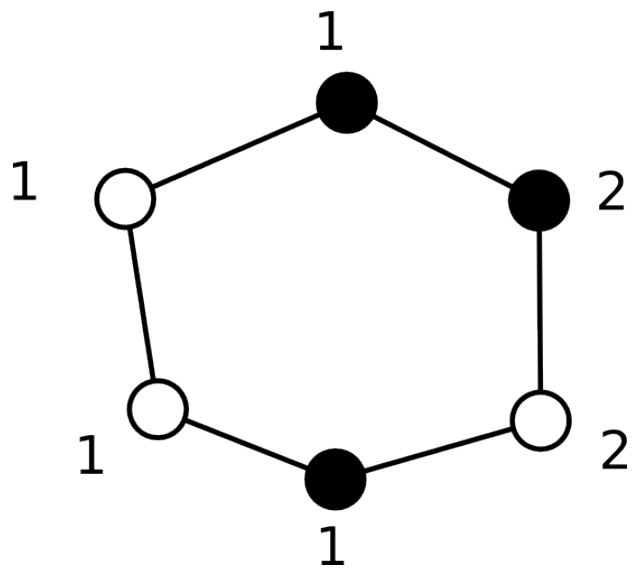
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

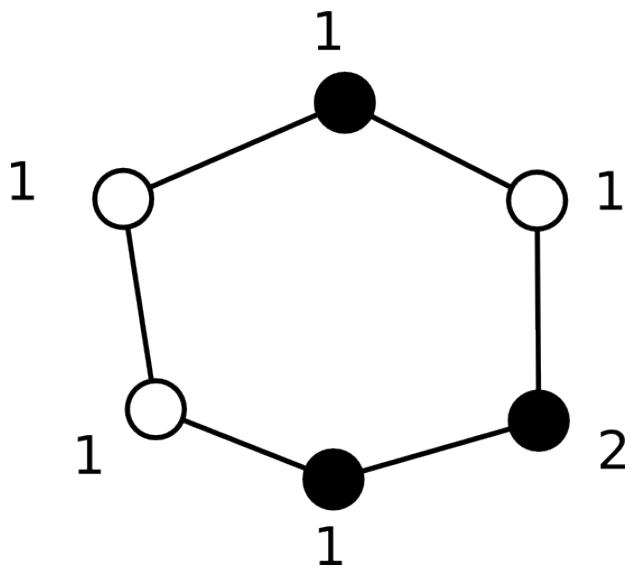
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

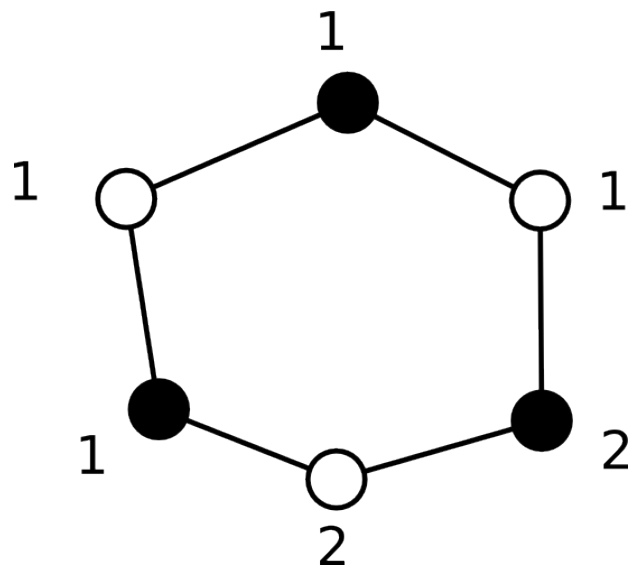
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

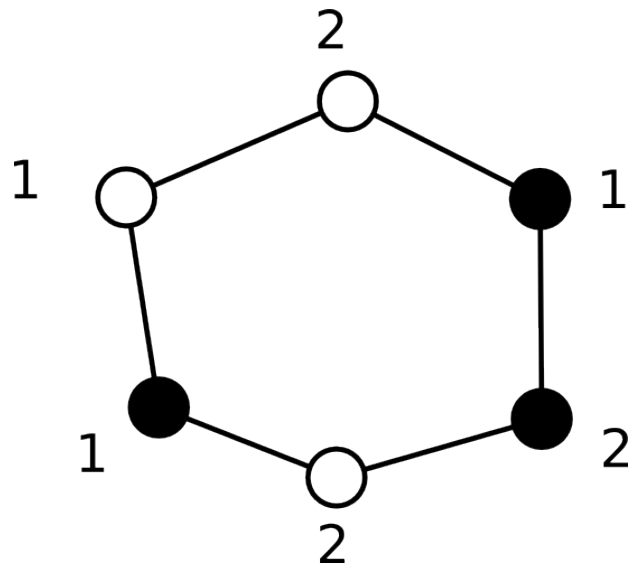
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

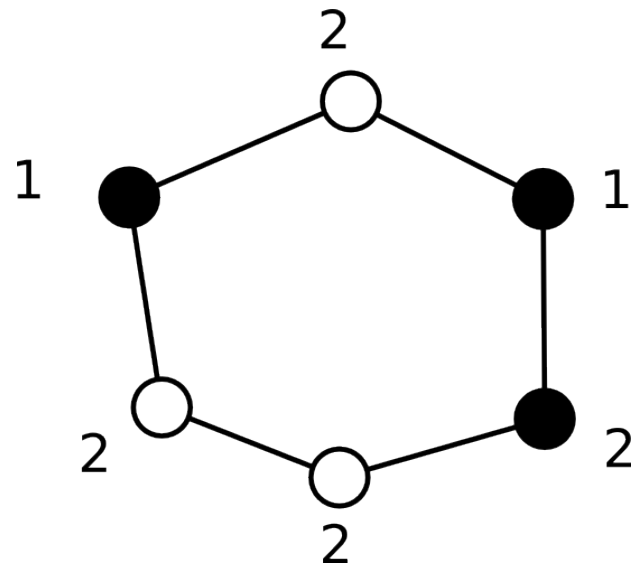
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

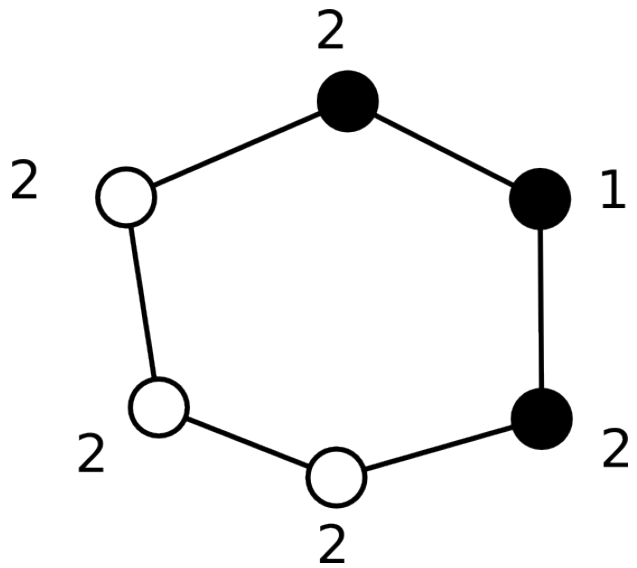
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

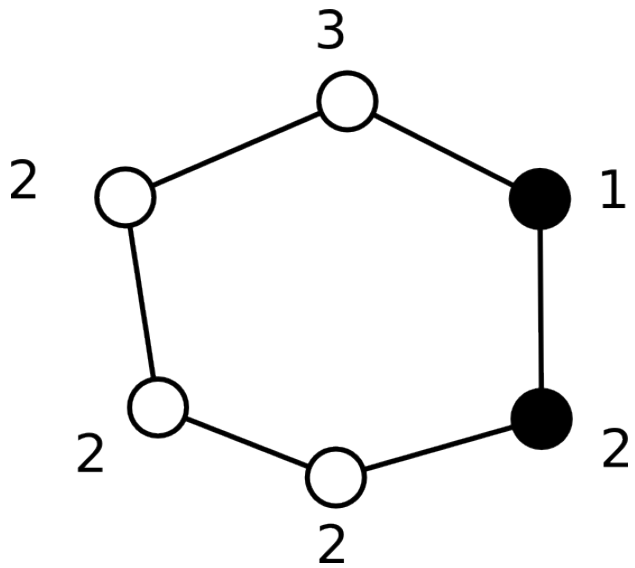
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

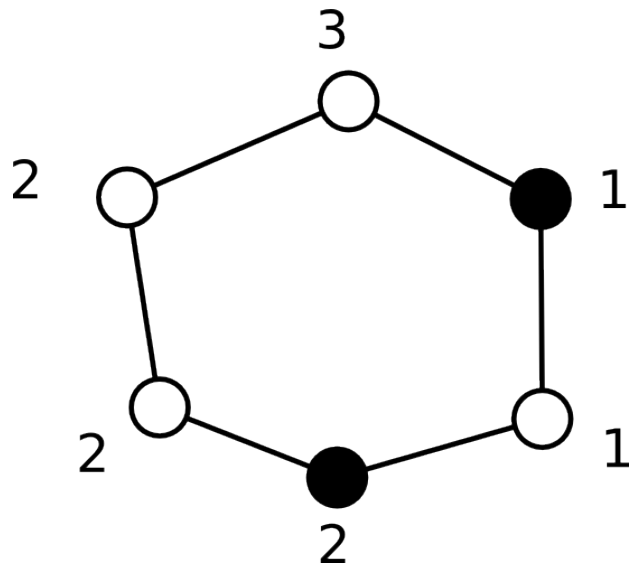
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

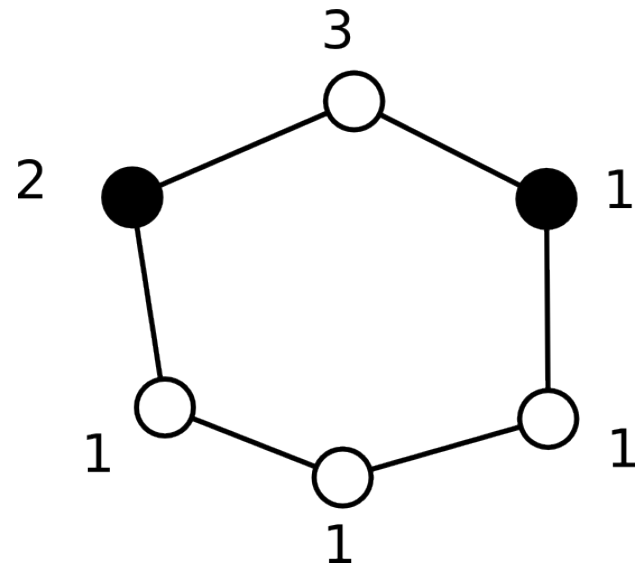
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

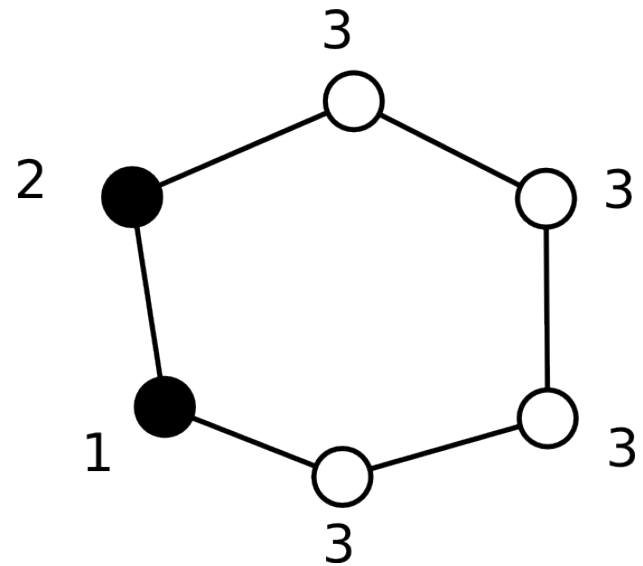
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

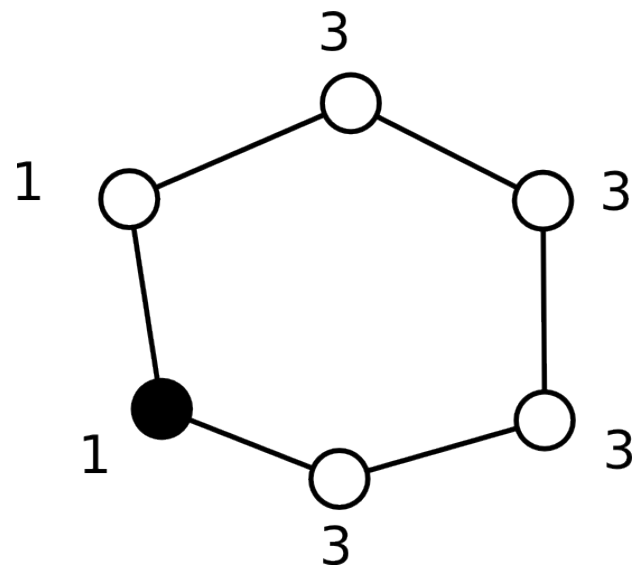
$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Convergence

$$\forall C, \forall \text{ fair execution } C \to C_1 \to \ldots, \exists i, C_i \in \mathcal{L}$$



Mutual Exclusion [Dijkstra]

Lemma

For any configuration C, there exists some $0 \leq z \leq n$, such that $z \neq x_i$ for all i.

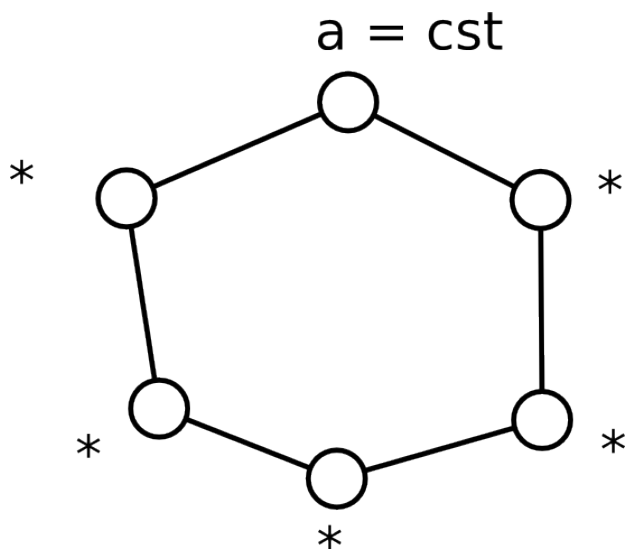
- n distinct processors
- $n+1$ possible values in $\{0, \ldots, n\}$

Mutual Exclusion [Dijkstra]

Lemma

For any configuration C, in any execution from C, the processor p_0 changes the value of x_0 infinitely often.

By contradiction

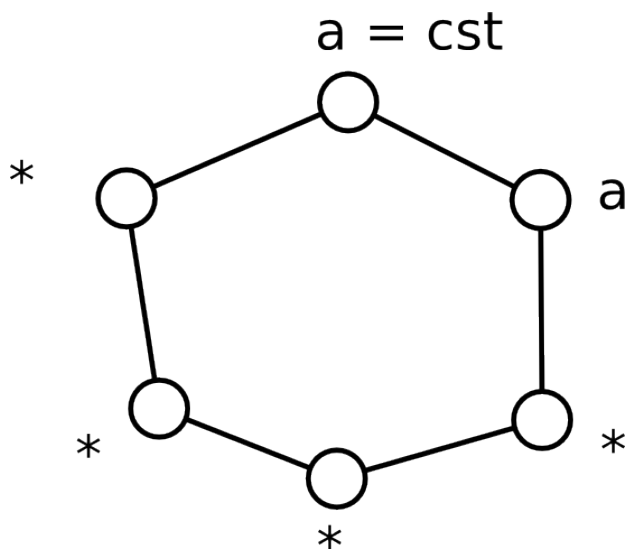


Mutual Exclusion [Dijkstra]

Lemma

For any configuration C, in any execution from C, the processor p_0 changes the value of x_0 infinitely often.

By contradiction

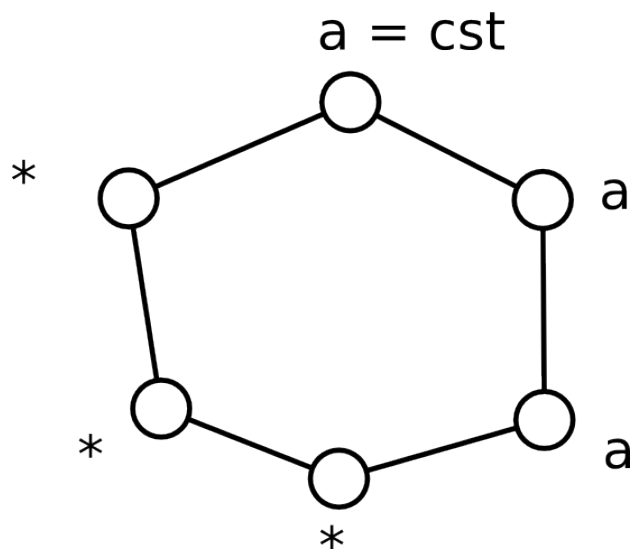


Mutual Exclusion [Dijkstra]

Lemma

For any configuration C, in any execution from C, the processor p_0 changes the value of x_0 infinitely often.

By contradiction

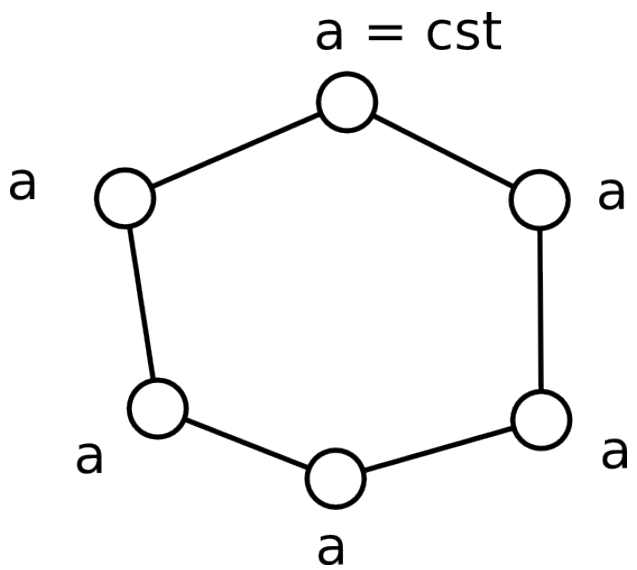


Mutual Exclusion [Dijkstra]

Lemma

For any configuration C, in any execution from C, the processor p_0 changes the value of x_0 infinitely often.

By contradiction

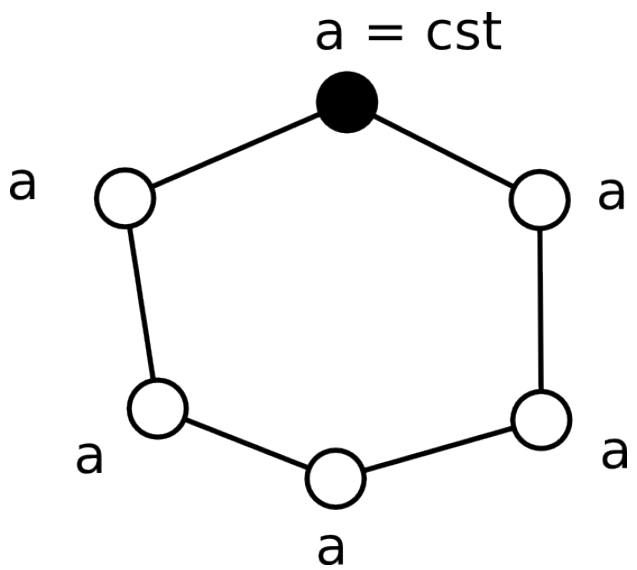


Mutual Exclusion [Dijkstra]

Lemma

For any configuration C, in any execution from C, the processor p_0 changes the value of x_0 infinitely often.

By contradiction

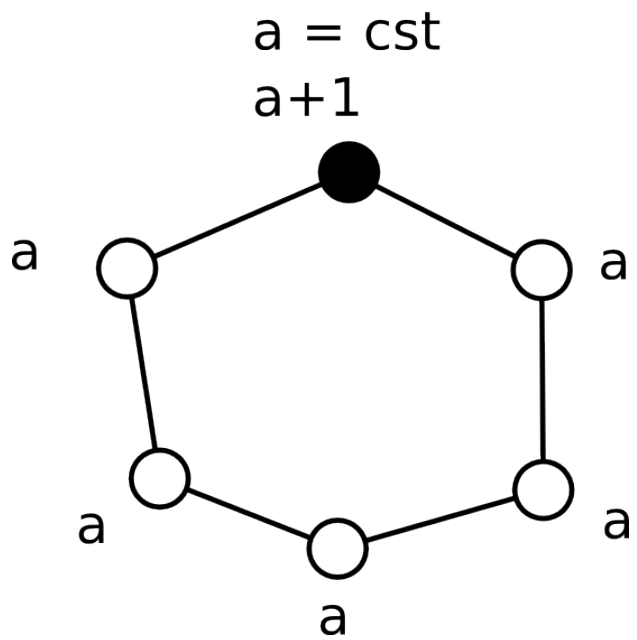


Mutual Exclusion [Dijkstra]

Lemma

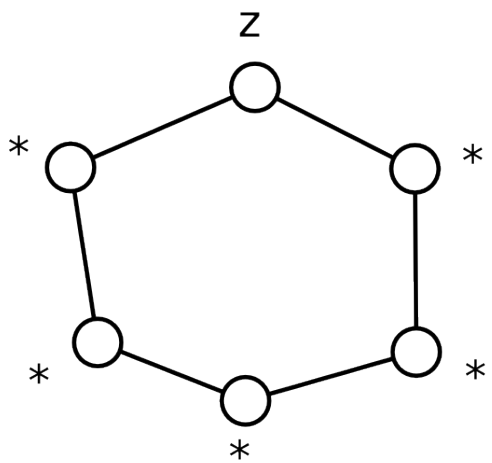
For any configuration C, in any execution from C, the processor p_0 changes the value of x_0 infinitely often.

By contradiction



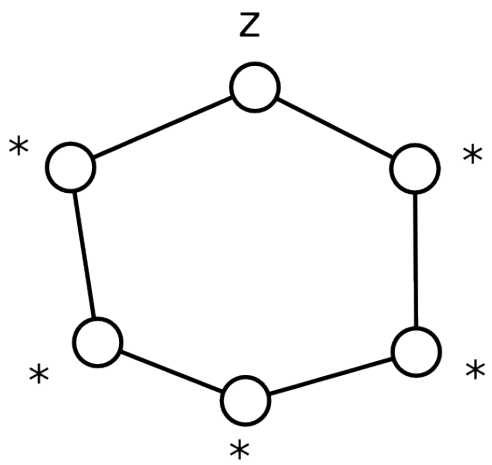
Mutual Exclusion [Dijkstra]

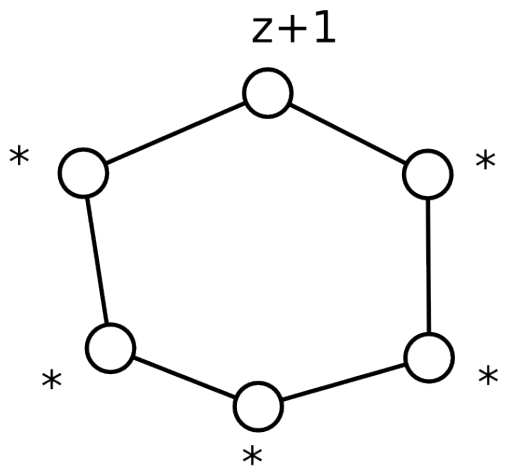
Final argument : at some point, a new value z is introduced



Mutual Exclusion [Dijkstra]

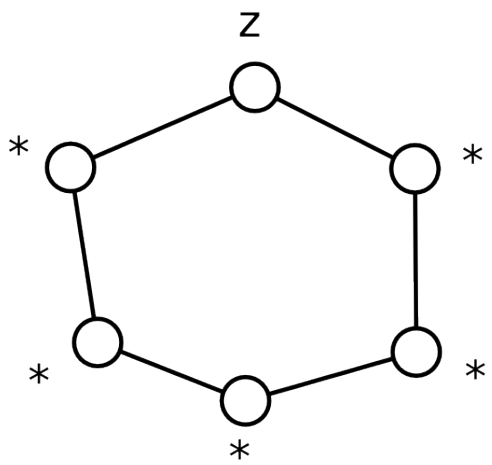
Final argument : at some point, a new value z is introduced

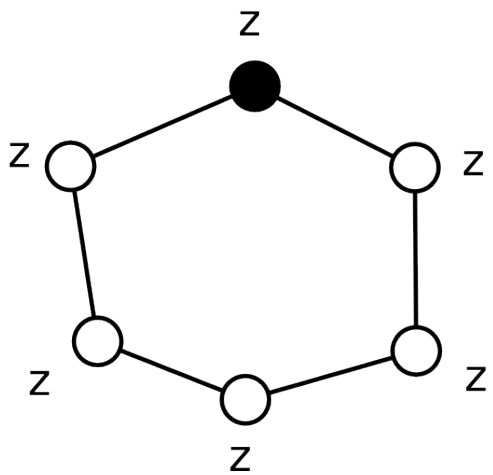


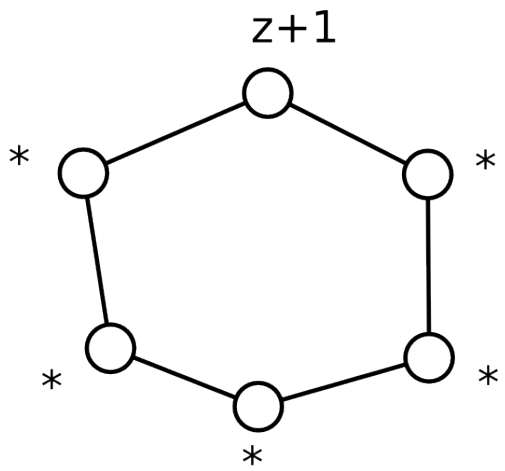


Mutual Exclusion [Dijkstra]

Final argument : at some point, a new value z is introduced







- 1. Basics
- 2. Dijkstra's Mutual Exclusion on Ring
- 3. Intringuing aspects of self-stabilization

A. Transient fault only corrupt local data, **not** the actual program.

- Program code stored in a safe place, reload if necessary
- If the program is corrupted, then one cannot reason about the processes actions : byzantine behaviour
  $\Rightarrow$ resiliency bounds are required

B. During the convergence period, the algorithm may violate the safety property.

- The algorithm may violate safety **because of** transient faults.
- Classic algorithm : ensures safety as long as no transient fault occurs ; if it occurs, the system may permanently violate safety, liveness, etc.
- Self-stabilizing algorithm : also ensures safety as long as no transient fault occurs ; if it occurs, the system eventually recovers.

C. Processes don’t know when the system has stabilized.

- A self-stabilizing algorithm is not intended to be started in an arbitrary configuration.
- As long as transient fault do not occur, processes knowledge is accurate.

Summary

- Transient faults : focus on suffix after last fault.
- Arbitrary starting configuration
- Self-stabilization : eventually behaves correctly
- Orthogonal to usual fault-tolerance (crash, byzantine, etc.)

References

- Dijkstra, Edsger W. (1974), *Self-stabilizing systems in spite of distributed control*, Communications of the ACM 17 (11)= 643-644
- Dolev, Shlomi (2000), *Self-Stabilization*, MIT Press

(Soon on wandida.)