Advanced Network Security

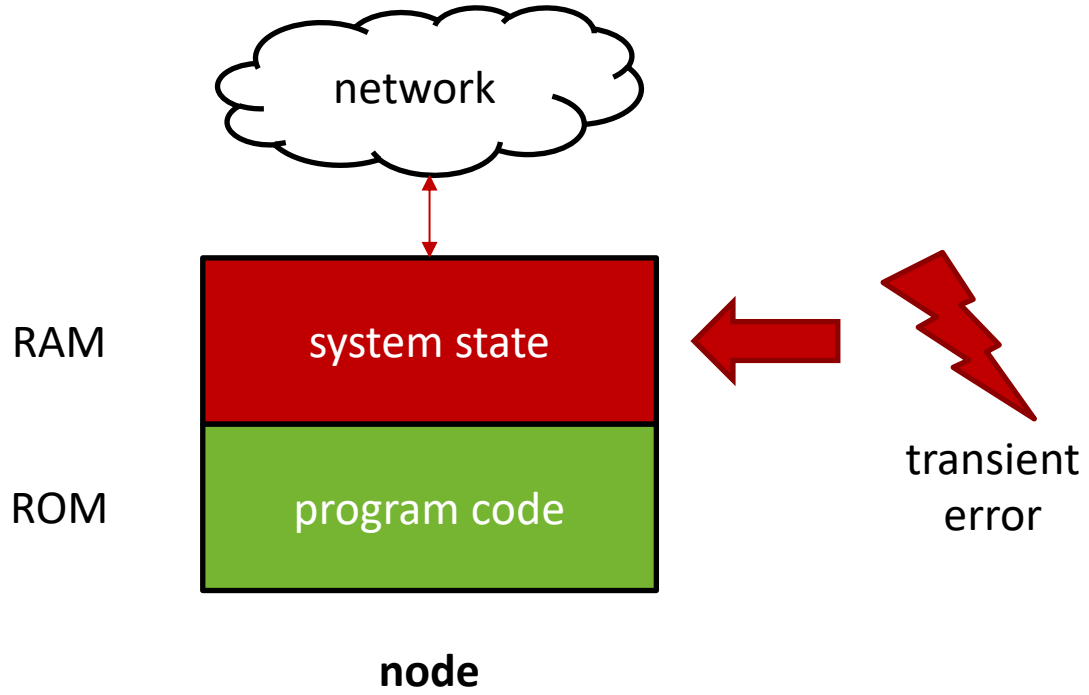
4. Self-stabilisation

Jaap-Henk Hoepman

Digital Security (DS)
Radboud University Nijmegen, the Netherlands
@xotoxot // ✉ *jhh@cs.ru.nl* // 🖱 *www.cs.ru.nl/~jhh*

Self-stabilisation: a different failure model

- **Instead of (permanent) processing failures we study *transient* memory failures**
  - State of a node stored in RAM, which can be changed arbitrarily
  - Program code stored in ROM, never changed
- **State of network can also change**
  - So study shared memory systems to simplify analysis
  - But self-stabilisation in message passing systems is also possible

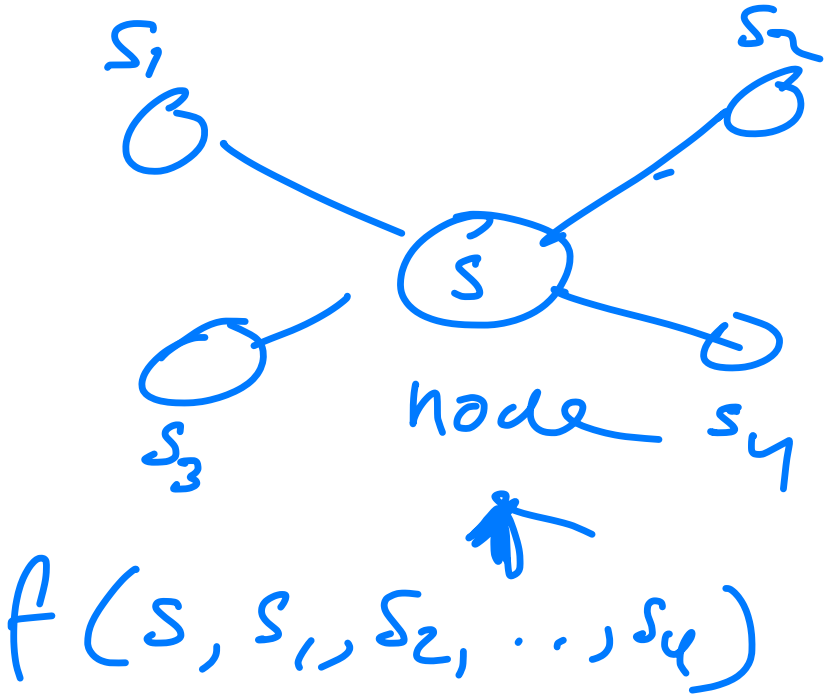


System model

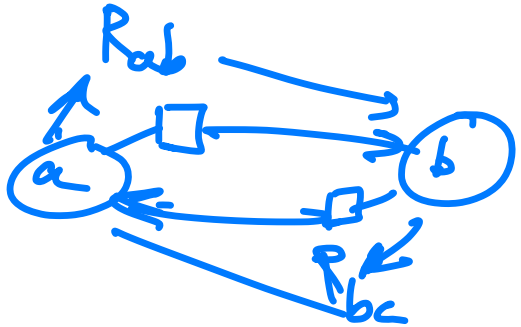
- n **nodes**
  - Uniform (all with the same state) or non-uniform
  - With or without known node identifiers (stored in ROM, i.e. cannot change)
- **Communicating through shared memory**
  - Modelled through graph $G = (V, E)$
  - **State reading** model: $(v, w) \in E$ means w can read *entire* state of v
  - **Link register** model: $(v, w) \in E$ means v writes a register R_{vw} read by w
- **Configuration $C \in \mathcal{C}$ consists of the Cartesian product of states of all nodes (and the registers on the edges).**

Link register vs state reading model

State reading



Link register

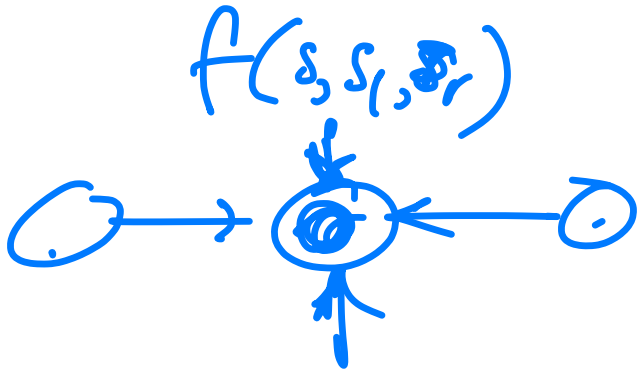


Synchronous system

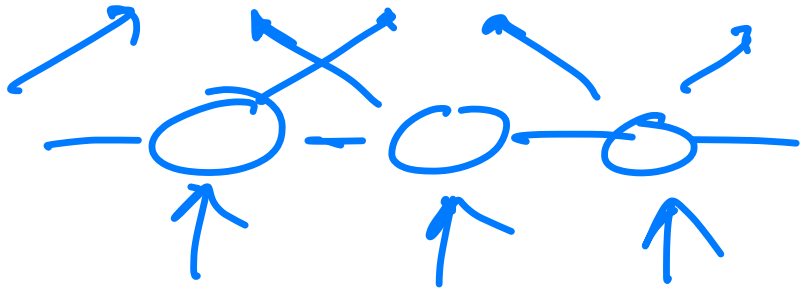
- **System $(\mathcal{C}, F)$ proceeds in rounds**
  - Program of a node i is a function $f_i \in F$ describing the resulting state (plus registers it writes) given the current state (and the registers it writes)
  - Uniform: $f_i = f_j$ for all i, j
  - Known identities: $f_i(C)$ may depend on i
- **Central daemon**
  - Scheduler fairly selects *one* node i to take a step: $C \xrightarrow{i} C'$ (i.e. $C' = f_i(C)$)
- **Distributed daemon**
  - Scheduler fairly selects *one or more* nodes I to take a step: first all nodes read their own state and the states/registers they can read, then compute the new state and then store the new state and the registers they write: $C \xrightarrow{I} C'$

Central daemon vs disitributed daemon

Central daemon



Distributed daemon

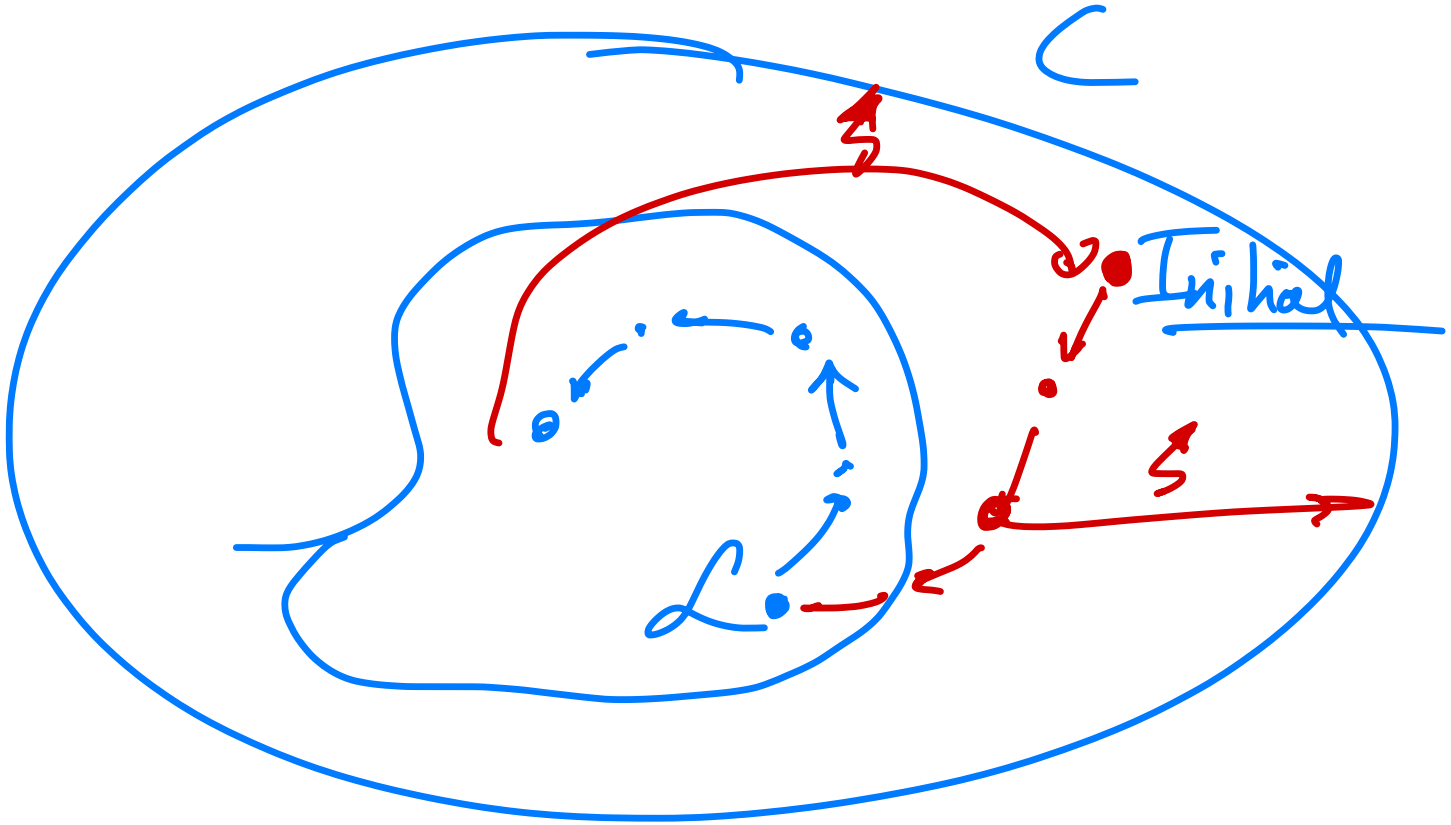


Self-stabilisation

- **Consider a system** $S = (\mathcal{C}, F)$
- **Define some set of legitimate configurations** $\mathcal{L} \subset \mathcal{C}$
  - Typically defined using a predicate
  - These are the good configurations that we want the system to be in
  - Note that a node may not be able to determine whether its *local* state is part of a *global* configuration that is legitimate!
- **System** S **is self-stabilising to** $\mathcal{L}$ **if**
  - **Convergence**: when we start the system in an arbitrary initial configuration $C \in \mathcal{C}$ it always reaches a legitimate configuration $C' \in \mathcal{L}$ within a finite number of steps (the convergence time)
  - **Closure**: Once the system is in a legitimate configuration it stays in a legitimate configuration after each system step

assuming no faults!

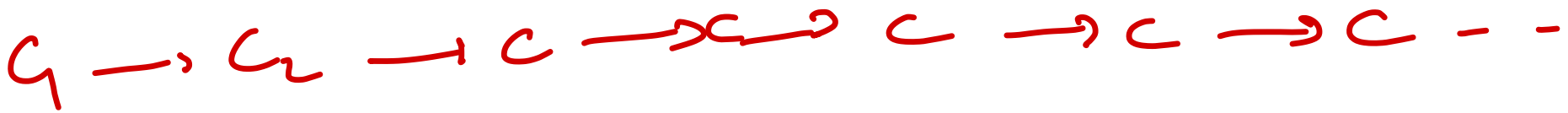
Self-stabilisation



Some questions

- **Why does a self-stabilising system recover from transient memory faults?**

- **Can a self-stabilising system terminate?**



Some toy problems (1)

- **Suppose the system consists of a single node…**

s arbitrary….

while $s \notin L$

do $s \leftarrow$ some $s \in L$

Some toy problems (2)

- Suppose the system is a complete graph, and the legitimate configurations are those where all nodes have the same state

node i

while $(s_1, s_2, \ldots, s_n) \notin L$

then $s_i \leftarrow$ default

↑ constant value

Some toy problems (3)

- **Let's add a progress condition and suppose the state of a node is a clock that needs to by in sync with all other nodes and increases in every round**

Proving self stabilisation

0): Define the legitimate states

Closure

- Show for every configuration $C \in \mathcal{L}$ and every step $\rightarrow$ from C to C' such that $C \rightarrow C'$, that $C' \in \mathcal{L}$

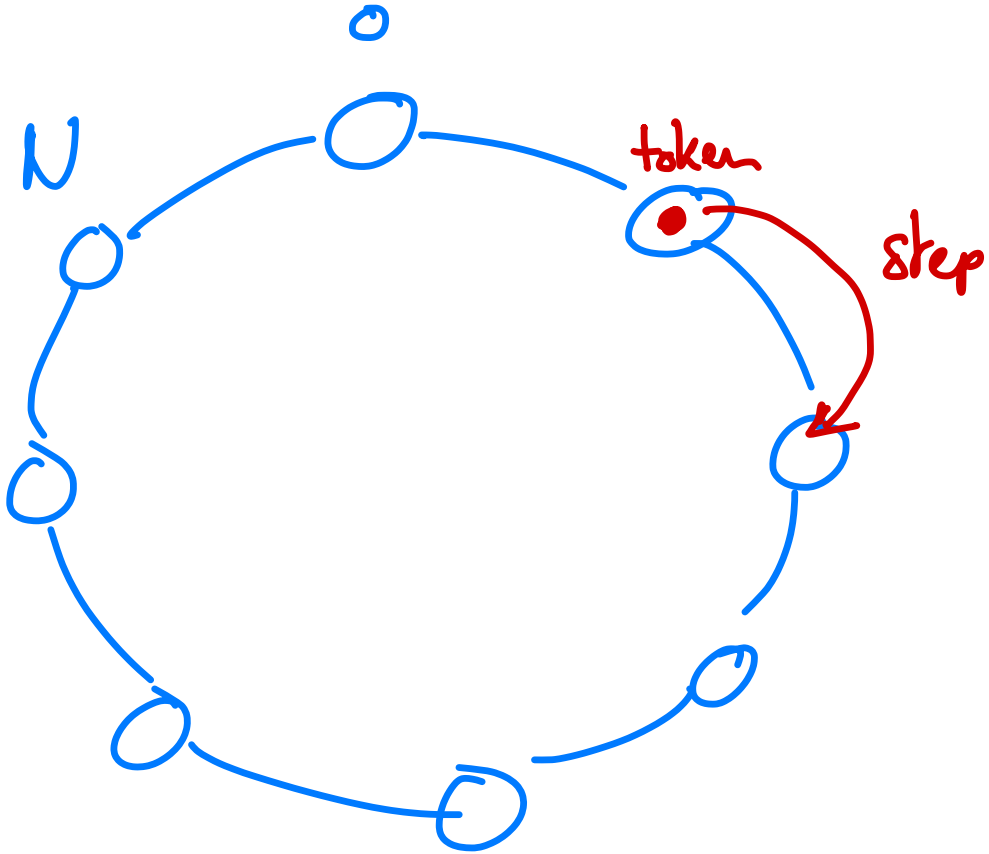
Convergence

- Define a bounding function $b: \mathcal{C} \rightarrow \mathbb{N}$ over all possible configurations such that
  - ★ *for all configurations* C, C' *and steps* $C \rightarrow C'$ *we have* $b(C) > b(C')$ *, and*
  - ★ $b(C) = 0$ *implies* $C \in \mathcal{L}$

Mutual exclusion on a ring (Dijkstra)

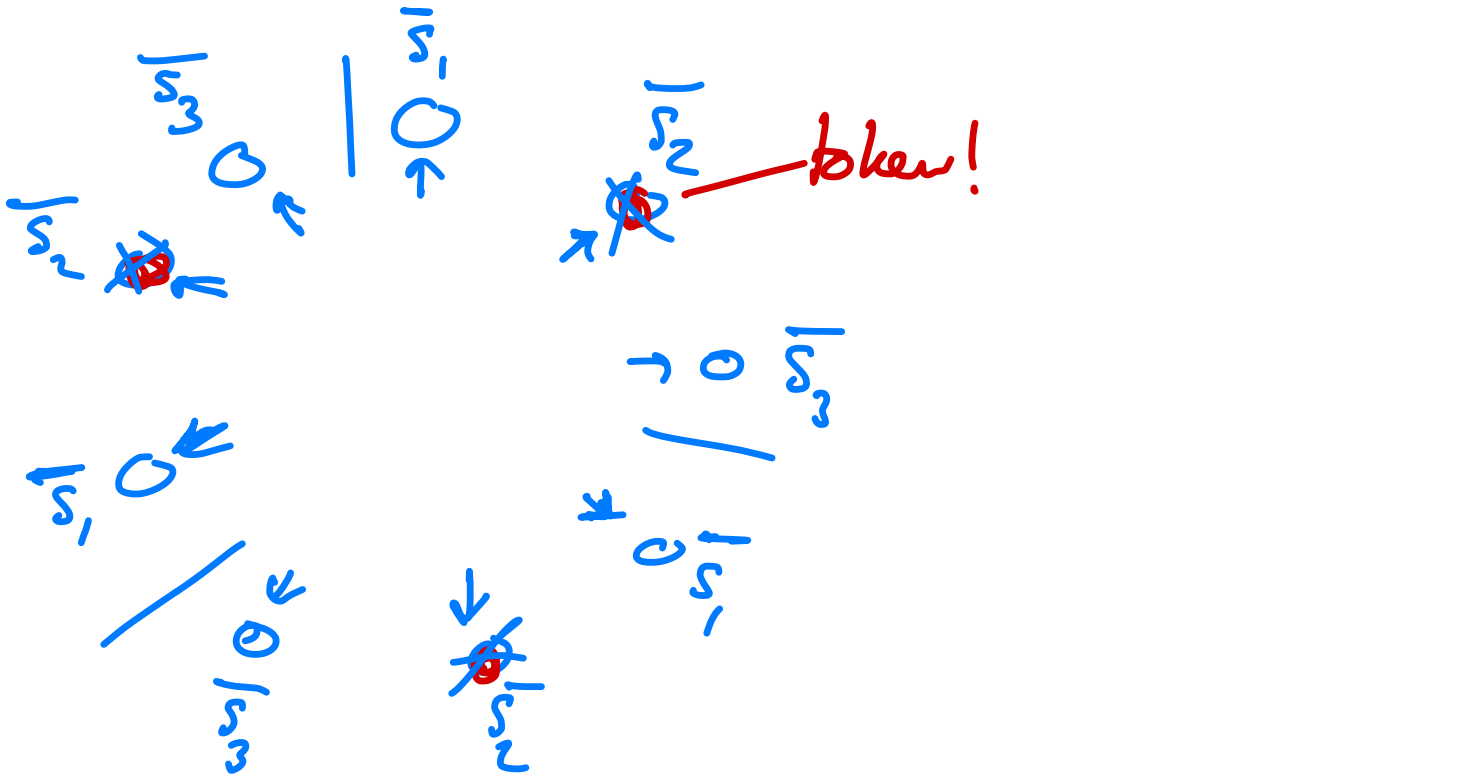
- $N + 1$ **nodes**
- **State reading**
- **Central daemon**

token ring



Impossibility of symmetric solution

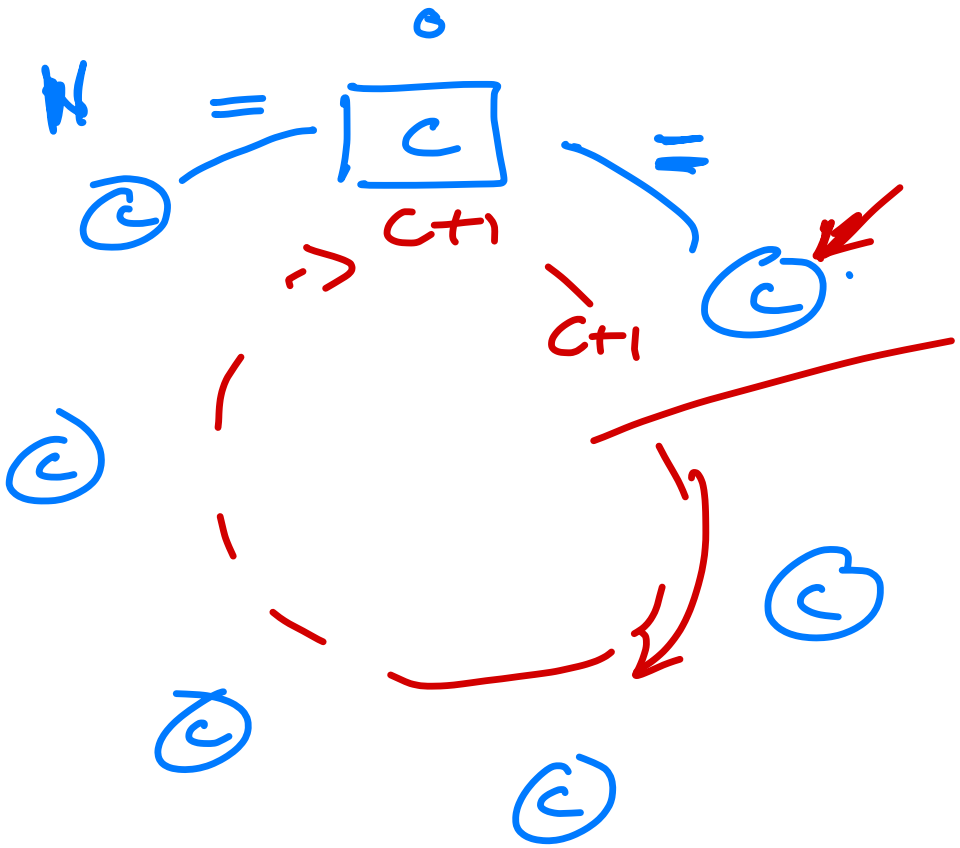
- For rings of non-prime size!



Mutual exclusion on a ring: construction

- $N+1$ **nodes** $0, \ldots, N$**, node** 0 **is special**
  - On an oriented, state reading ring: node $i+1 \; mod \; N$ reads state of node i
- **Each node has state** $x[i] \in \{0, \ldots, K-1\}$ **for** $K > N$
- **Protocol**
  - Node 0: if $x[0] = x[N]$ then $x[0] \leftarrow x[0]+1 \; mod \; K$
  - Node $i \neq 0$: if $x[i] \neq x[i-1]$ then $x[i] \leftarrow x[i-1]$
- **Privileged (i.e. has the token)**
  - Node 0 if $x[0] = x[N]$
  - Node $i \neq 0$ if $x[i] \neq x[i-1]$
  - I.e. node is privileged if it can take a step (which turns it unprivileged

Example



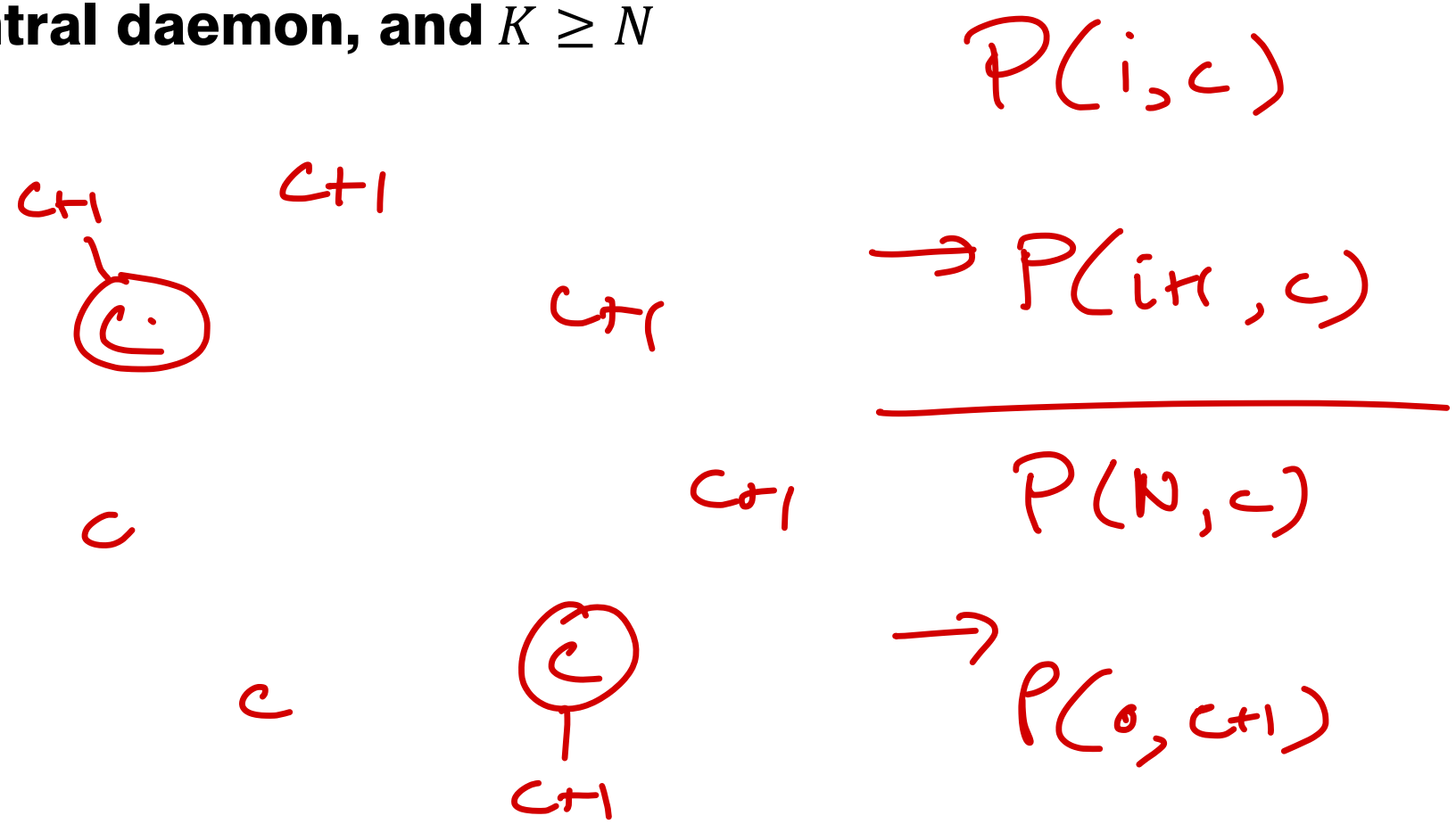
Legitimate states

Legitimate states $\mathcal{L}$

- Define $P(i, c)$ over a configuration as
  "for all $j, 0 \leq j \leq i$ we have $x[j] = c + 1 \; mod \; K$ and
  for all $j, i < j \leq N$ we have $x[j] = c$"
- All states where there is an i, $0 \leq i \leq N$ and a $c \in \{0, \ldots, K-1\}$ such that $P(i, c)$ holds are legitimate

Proof of correctness: closure

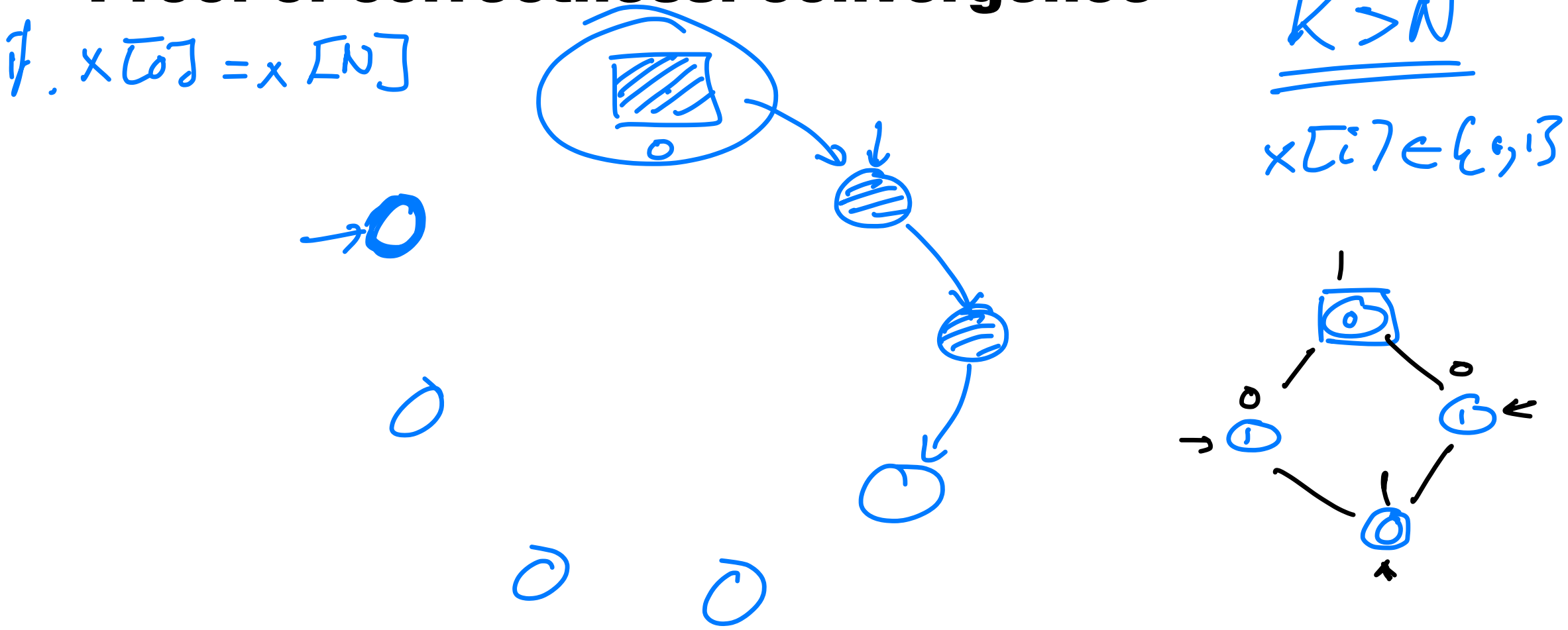
- **Assume a central daemon, and** $K \geq N$



Proof of correctness (1)

- **Note: always at least one node enabled, so there is no deadlock**
- **In a legitimate configuration exactly one node enabled/privileged**
- **Assume a central daemon, and $K \geq N$**
- **Closure**
  - Let the system be in a legitimate configuration for some choice of i, c, i.e. $P(i, c)$ = "for all $j, 0 \leq j \leq i$ we have $x[j] = c + 1 \; mod \; K$ and for all $j, i < j \leq N$ we have $x[j] = c$" holds.
  - Then only $k = i + 1 \; mod \; N$ is enabled
  - If $k \neq 0$, then after the step $x[k] = c + 1 \; mod \; K$ and hence $P(k, c)$ holds
  - If $k = 0$, then after the step $x[k] = c + 2 \; mod \; K$ while for all other i still $x[i] = c + 1 \; mod \; K$. Hence $P(0, c + 1 \; mod \; K)$ holds

Proof of correctness: convergence



Proof of correctness (2)

- **Convergence**
  - Initially colour all nodes white
  - Colour node 0 blue the first time it takes a step; after that it stays blue forever
  - Nodes colour blue when they copy a blue value from their counterclockwise neighbour (and then stay blue forever)
  - Let h be the number of times node 0 takes a step while node N is still white
  - Then $h \leq N$: after the first step of 0, there are at most $N-1$ white nodes that can provide N with at most $N-1$ new white states
  - W.l.o.g. let $x[0]$ initially be $K-1$; after the first step $x[0]$ becomes 0; so after i-th step, $x[0] = i-1$
  - Starting at 0, we observe that all blue nodes form a decreasing chain of values
  - Now let 0 be about to take the $h+1$ th step (i.e. N is blue). Then before that step $x[0] = h-1$. As $h \leq N \leq K$ we see that $x[0]$ did not wrap around.
  - All nodes are blue at this point, and $x[N] = x[0]$.
  - As now all nodes are blue, and have decreasing values, we must have $x[i] = h-1$ for all nodes