Self-Stabilizing *K*-out-of-*L* Exclusion on Tree Networks

Stéphane Devismes, VERIMAG

Joint work with:

- *Ajoy K. Datta* (Univ. Of Nevada)
- *Florian Horn* (LIAFA)
- *Lawrence L. Larmore* (Univ. Of Nevada)

Roadmap

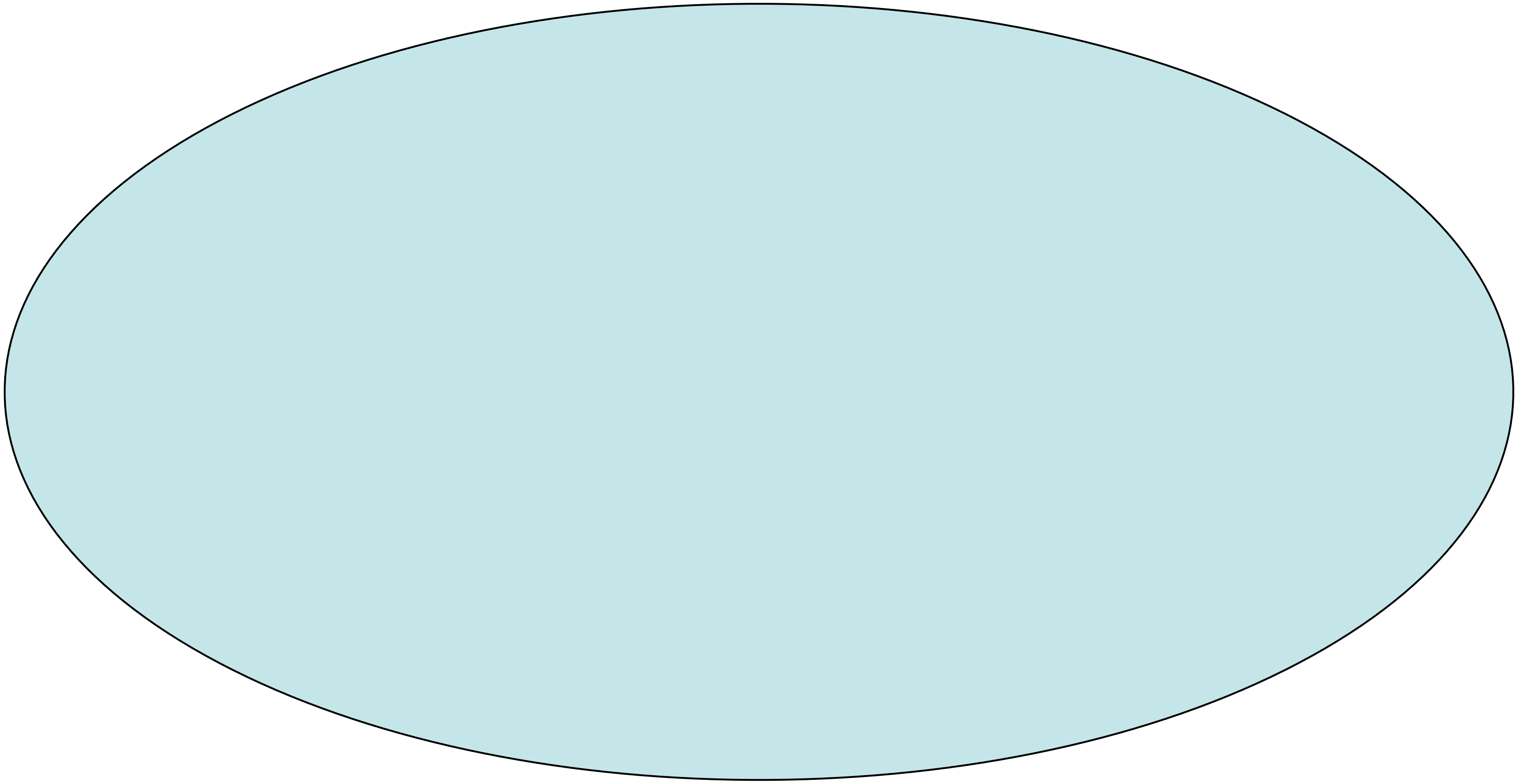
- Recall on Self-stabilization
- Definition of the problem
- The solution
- Conclusion and perspectives

Roadmap

- Recall on Self-stabilization
- Definition of the problem
- The solution
- Conclusion and perspectives

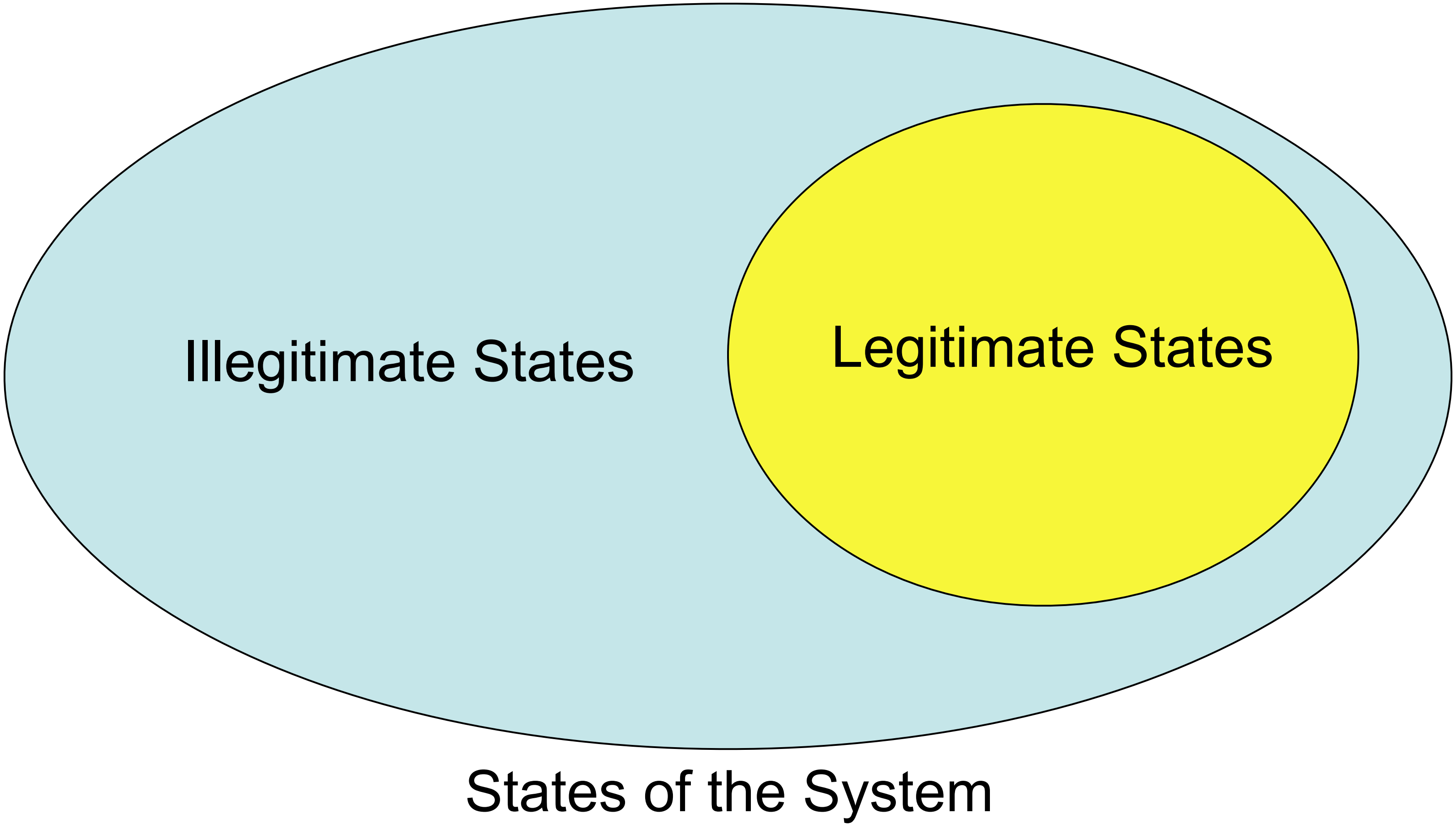
Self-Stabilization: Closure + Convergence

Self-Stabilization: Closure + Convergence

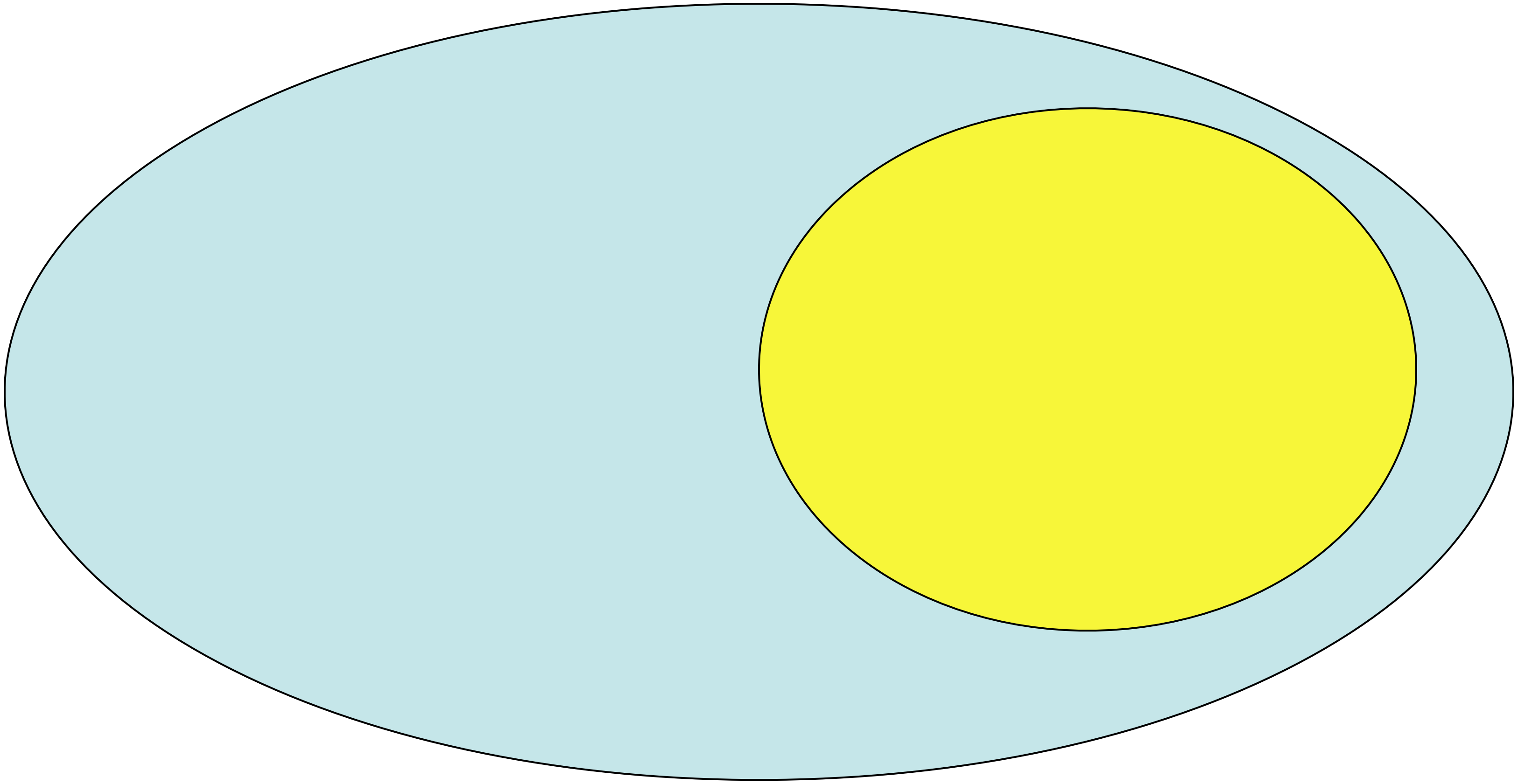


States of the System

Self-Stabilization: Closure + Convergence

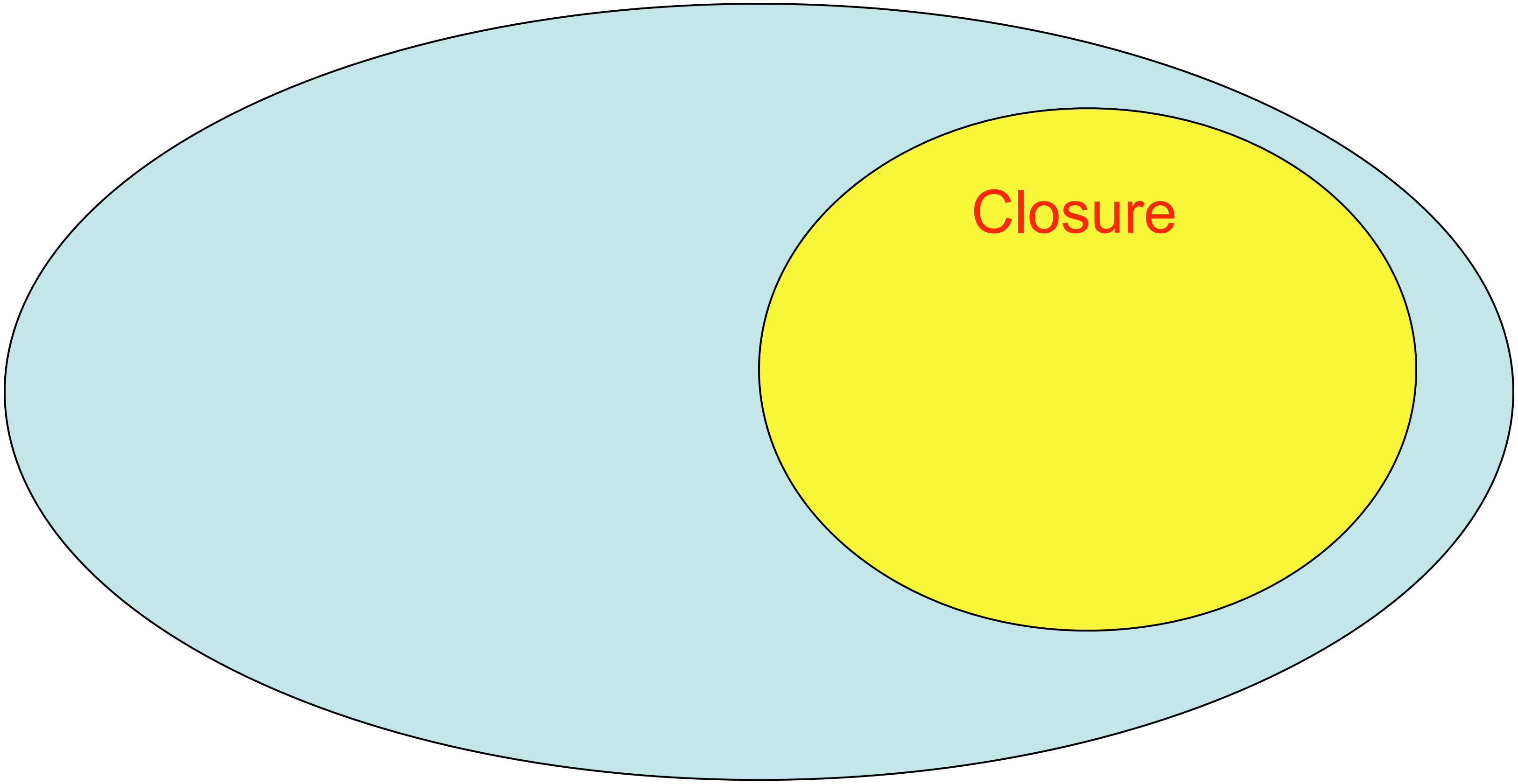


Self-Stabilization: Closure + Convergence



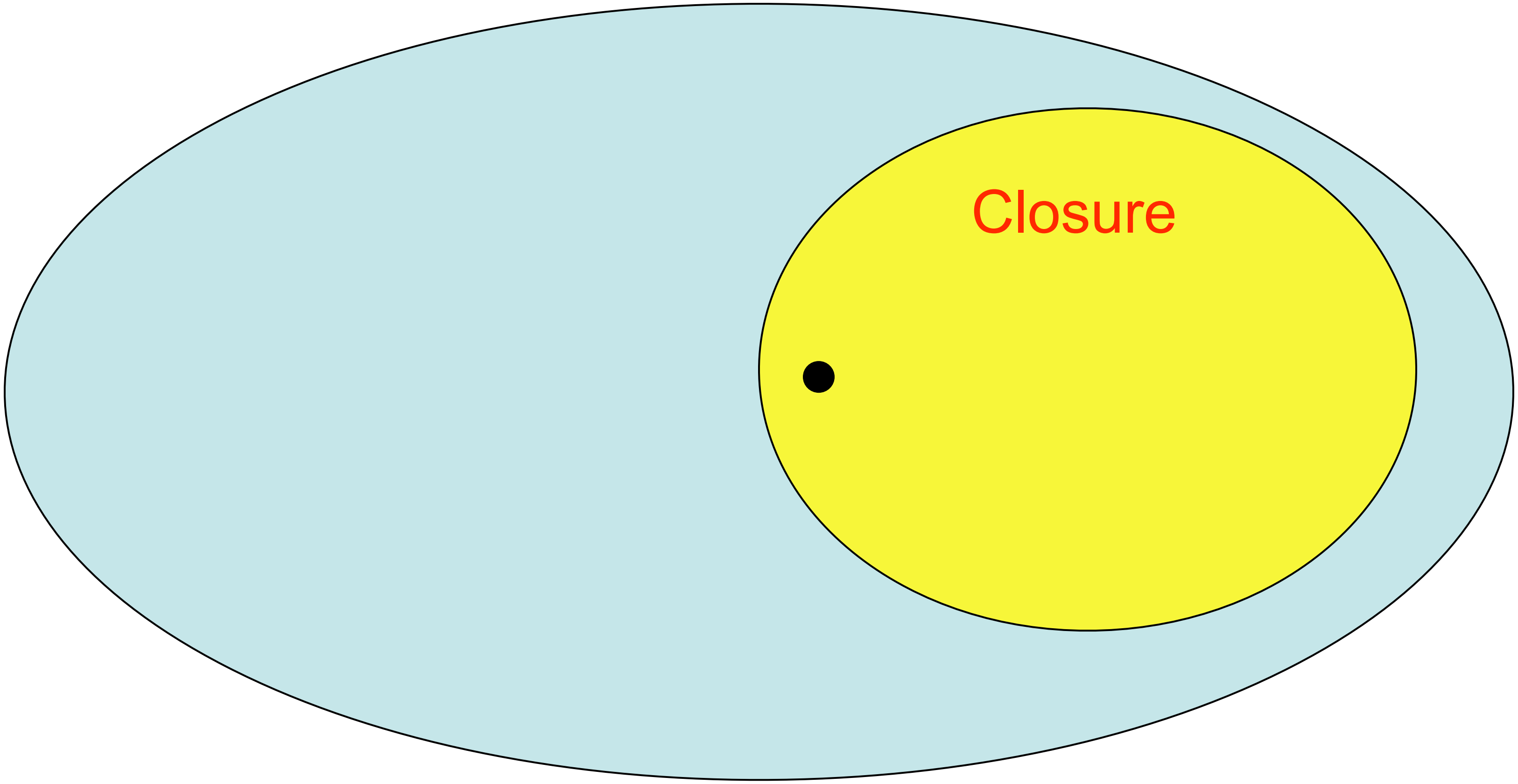
States of the System

Self-Stabilization: Closure + Convergence



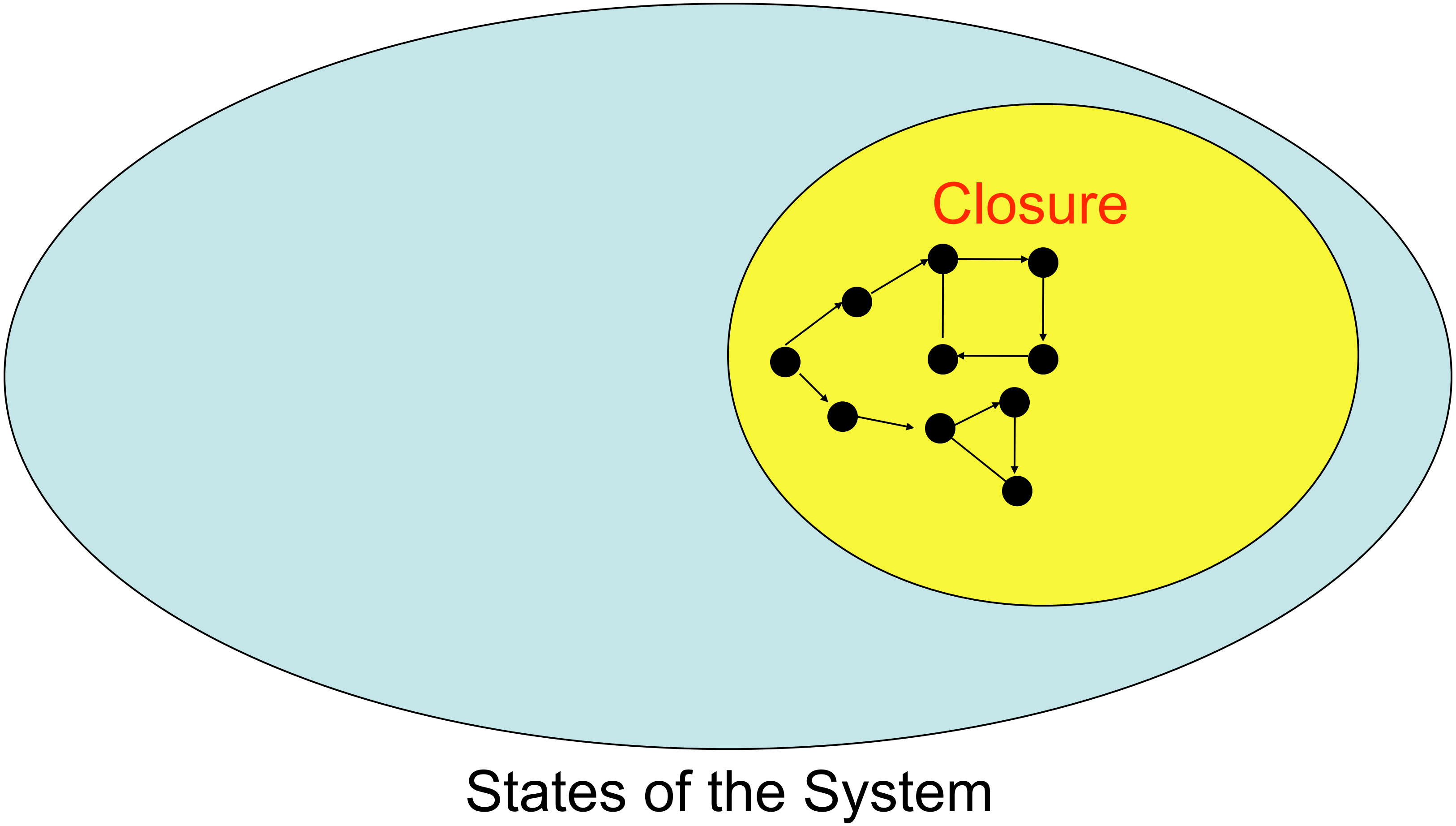
States of the System

Self-Stabilization: Closure + Convergence

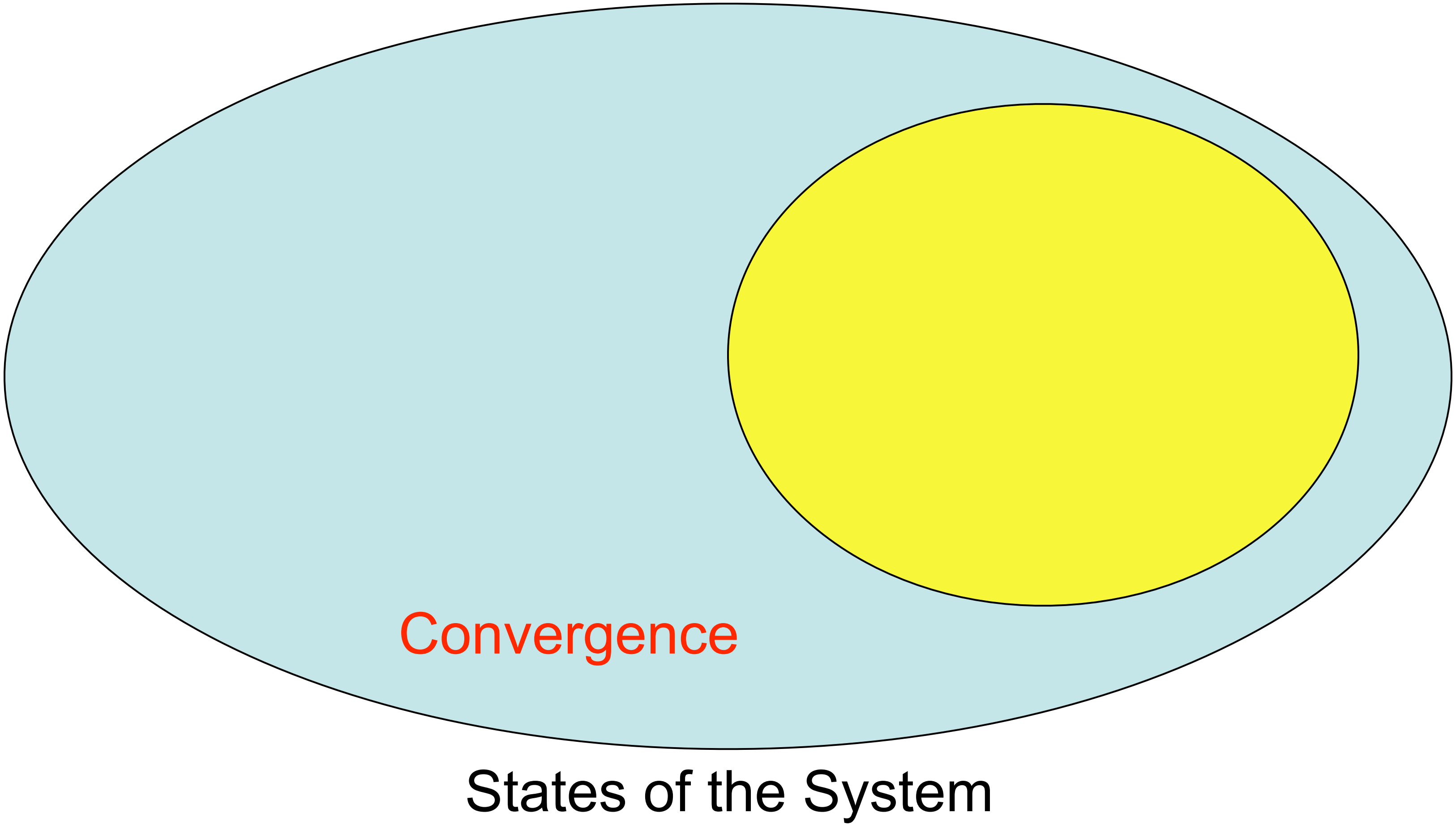


States of the System

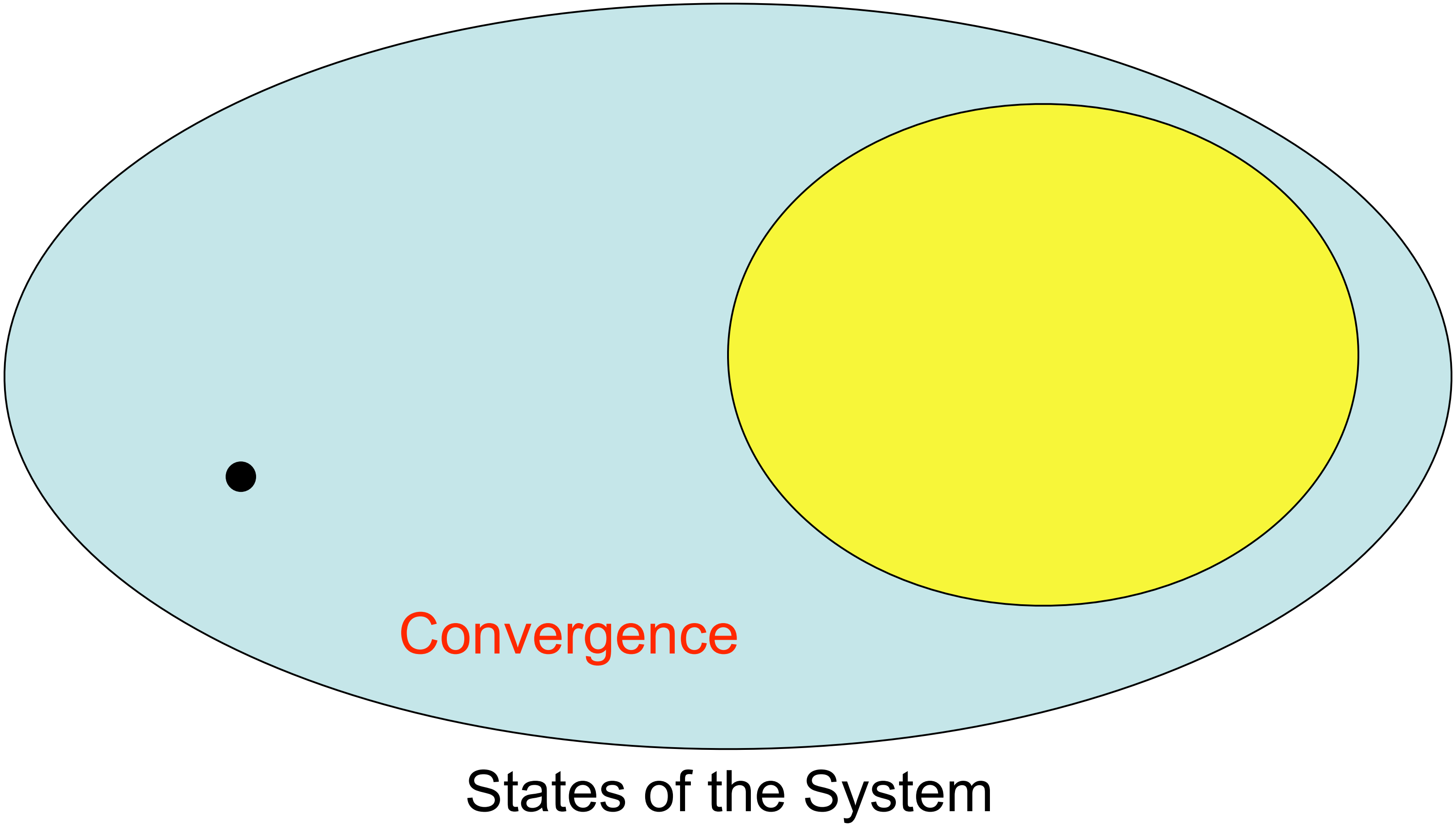
Self-Stabilization: Closure + Convergence



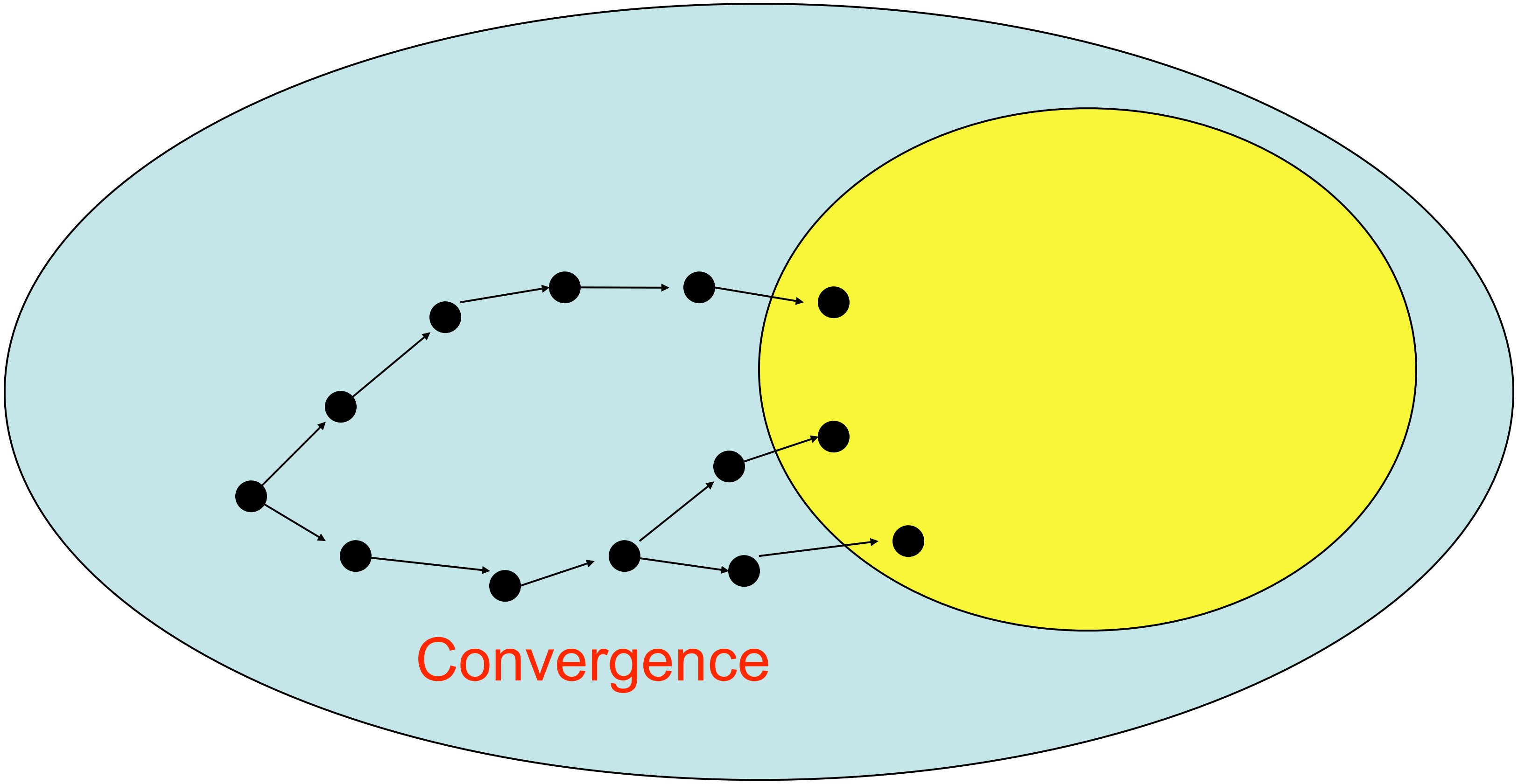
Self-Stabilization: Closure + Convergence



Self-Stabilization: Closure + Convergence

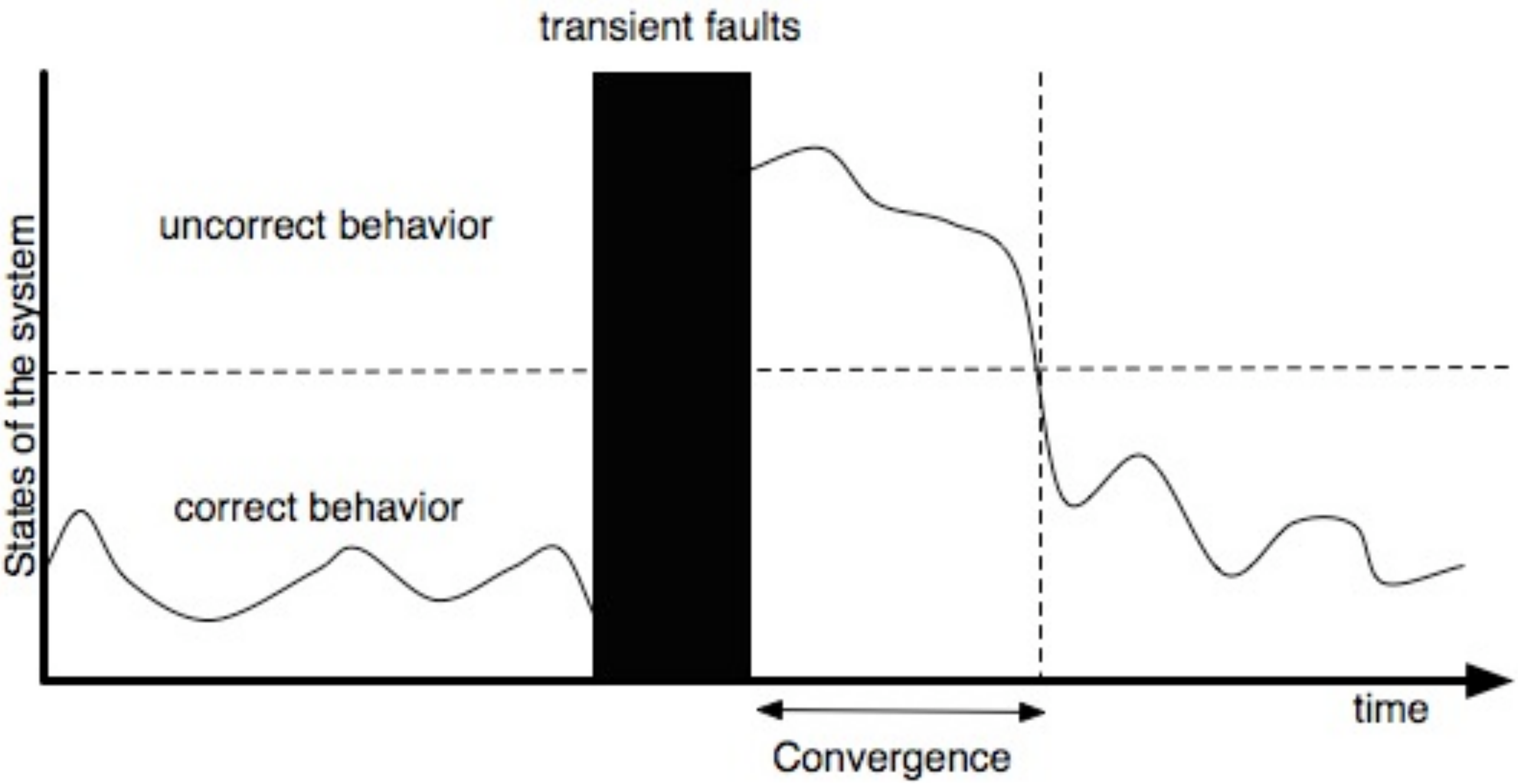


Self-Stabilization: Closure + Convergence



States of the System

Property: Tolerance to Transient Faults



Roadmap

- Recall on Self-stabilization
- Definition of the problem
- The solution
- Conclusion and Perspectives

K-out-of-*L* Exclusion [Raynal, 91]

- *L* resource units
- Requests from 1 to *K* resource units ($K \leq L$)

K-out-of-*L* Exclusion

- 3 property to ensure:
  - ***Safety***
  - ***Fairness***
  - ***Efficiency***

Safety

- At any time:
  - Each resource unit is used by **at most one** process
  - Each process uses **at most *K*** resource units
  - **At most *L*** resource units are available

Fairness

- Each request (of at most *K* units) is satisfied in finite time

(*i.e.* the process then uses the resource units it holds in a special section of code called *critical section*)

Efficiency

Efficiency

- « As many requests as possible must be satisfied simultaneously »

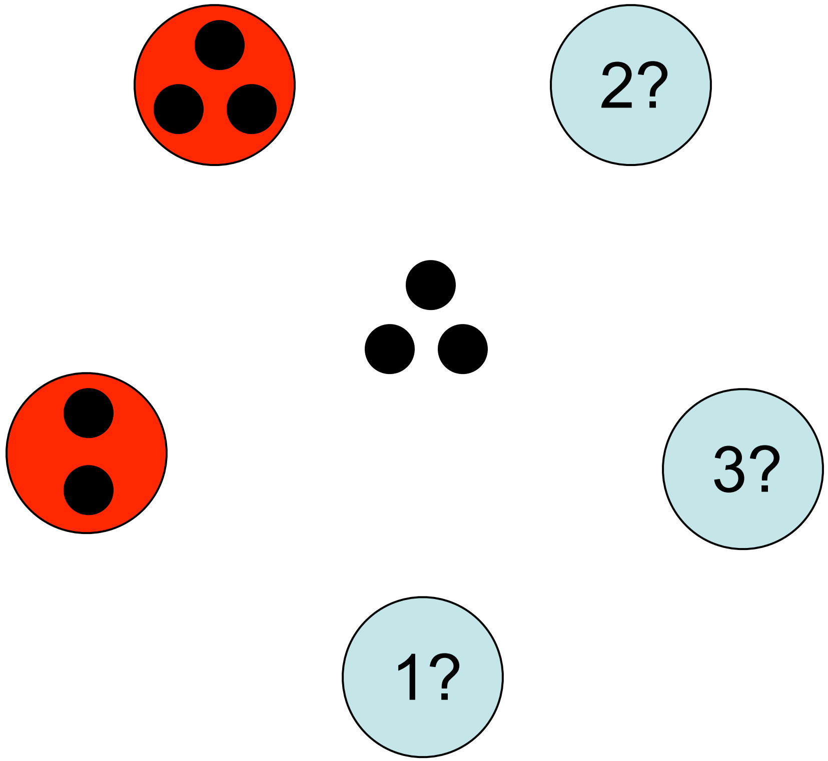
Efficiency

- « As many requests as possible must be satisfied simultaneously »

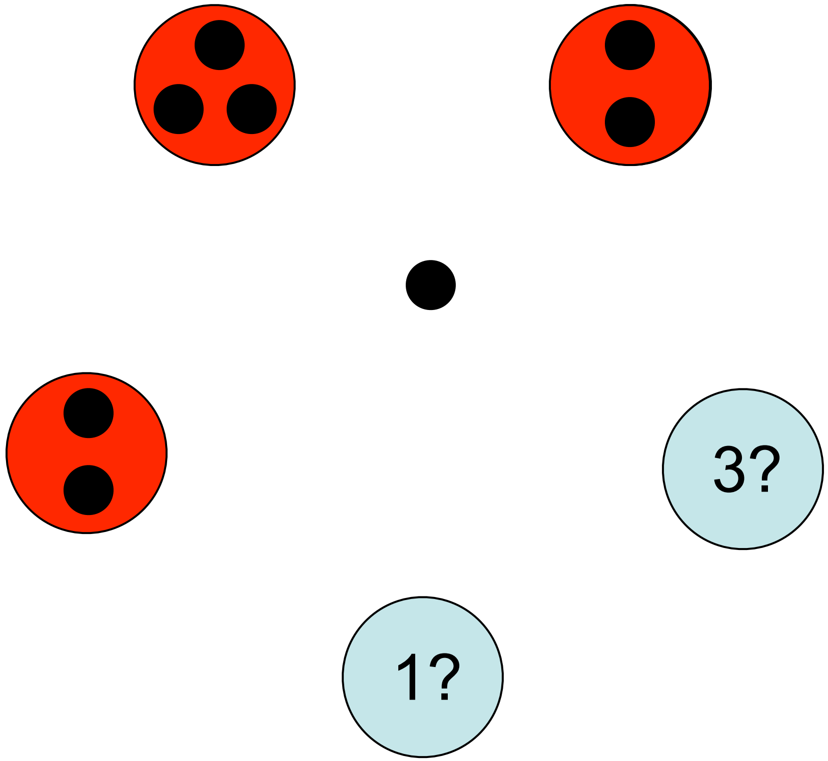
Efficiency

- « As many requests as possible must be satisfied simultaneously »
- More formally: (K,L)-Liveness

(K,L)-Liveness



(K,L)-Liveness



Waiting Time

« The maximum number of time, all other processes can enter in the critical section before some process *p*, starting from the moment requests the critical section »

Roadmap

- Recall on Self-stabilization
- Definition of the problem
- The solution
- Conclusion and perspectives

Model

Model

- Message-passing
- Bounded process memories

Model

- Message-passing
- Bounded process memories
- FIFO bidirectional links

Model

- Message-passing
- Bounded process memories
- FIFO bidirectional links
- Topology : oriented tree

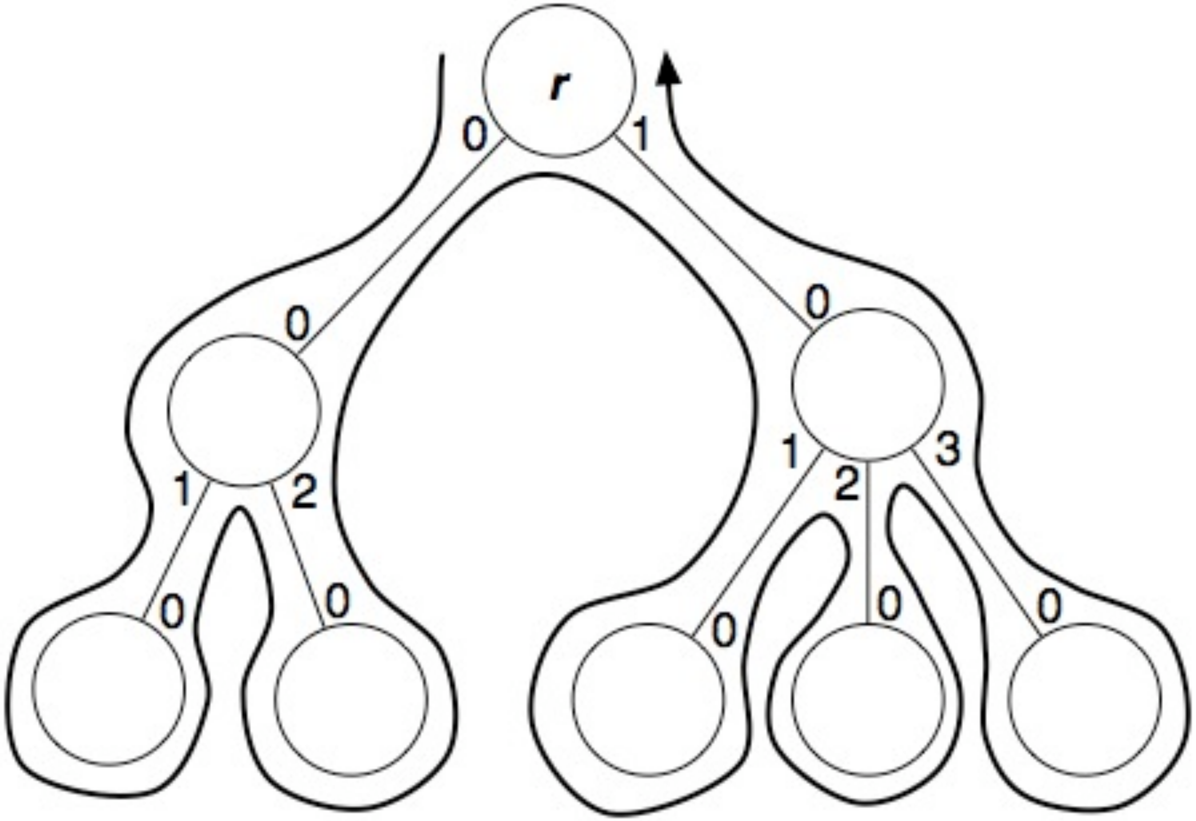
Model

- Message-passing
- Bounded process memories
- FIFO bidirectional links
- Topology : oriented tree
- Necessary condition [Gouda-Multari, 91]:
  - Bounded link capacity (*CMAX*)

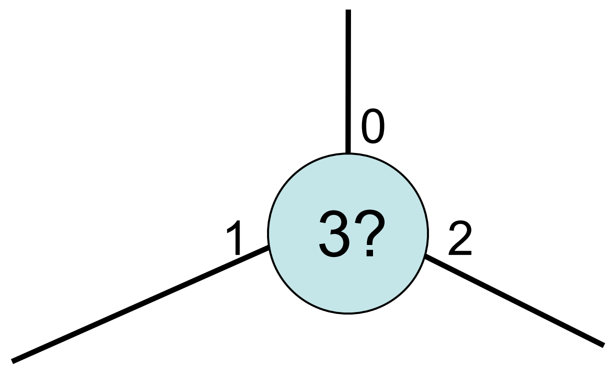
Approach

- Token-based (resource units = tokens)
- Modular:
  - Non Self-stabilizing *K*-out-of-*L* Exclusion
  - Self-stabilizing Controller

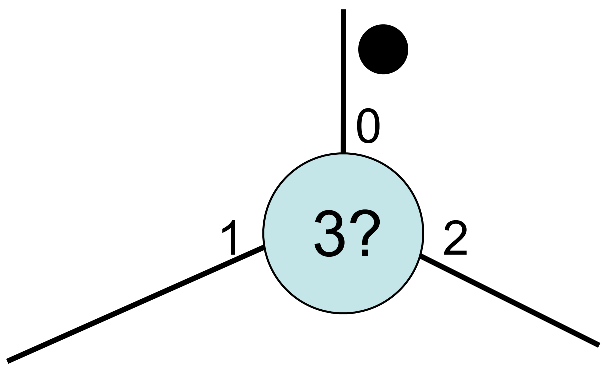
DFS circulation



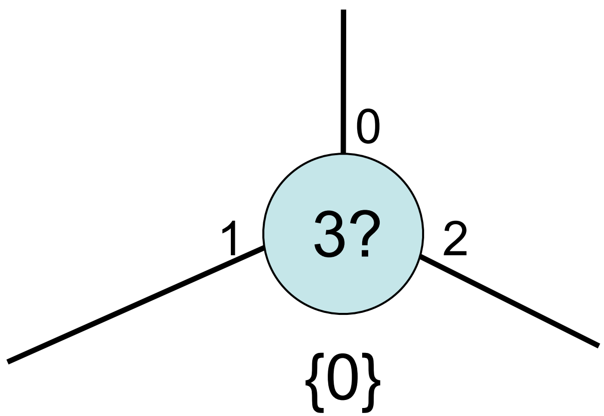
First idea:
K-out-of-*L* Exclusion = *L*-circulation



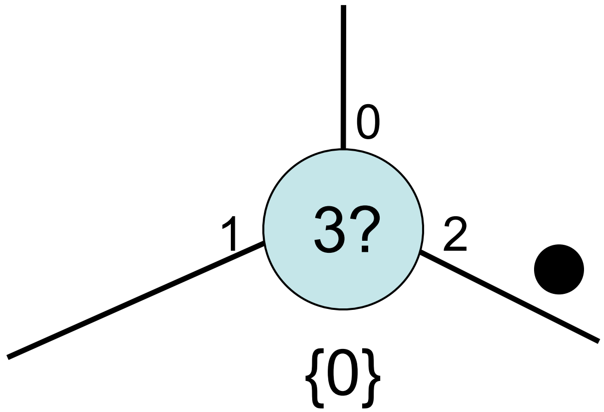
First idea:
K-out-of-*L* Exclusion = *L*-circulation



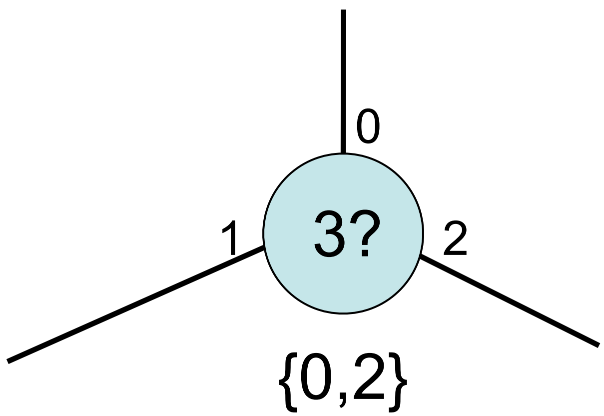
First idea:
K-out-of-*L* Exclusion = *L*-circulation



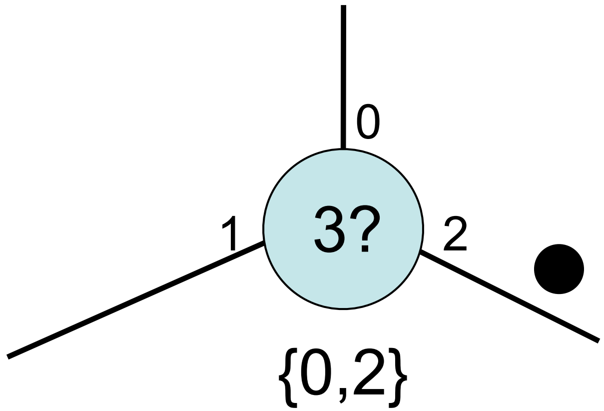
First idea:
K-out-of-*L* Exclusion = *L*-circulation



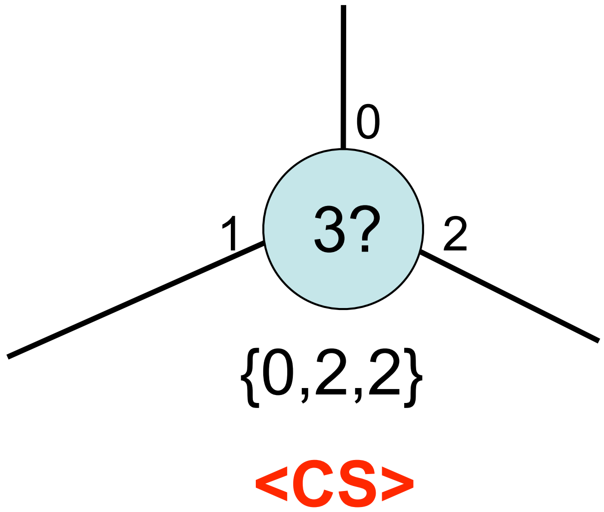
First idea:
K-out-of-*L* Exclusion = *L*-circulation



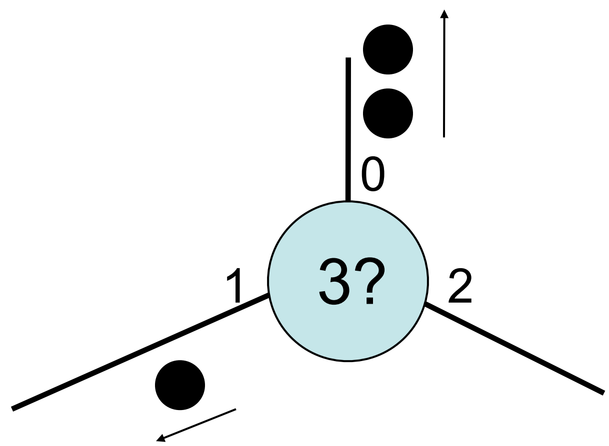
First idea:
K-out-of-*L* Exclusion = *L*-circulation



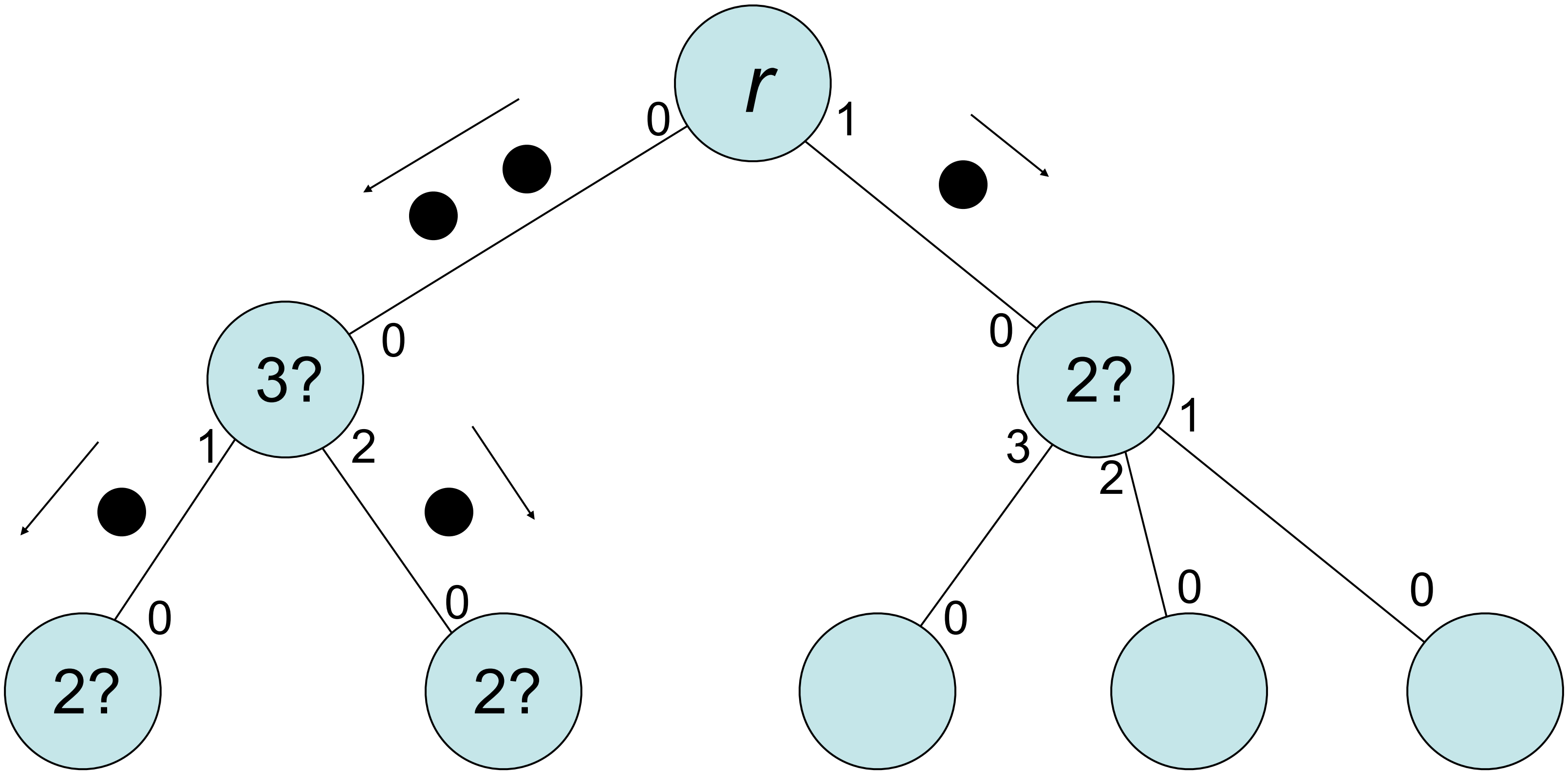
First idea:
K-out-of-*L* Exclusion = *L*-circulation



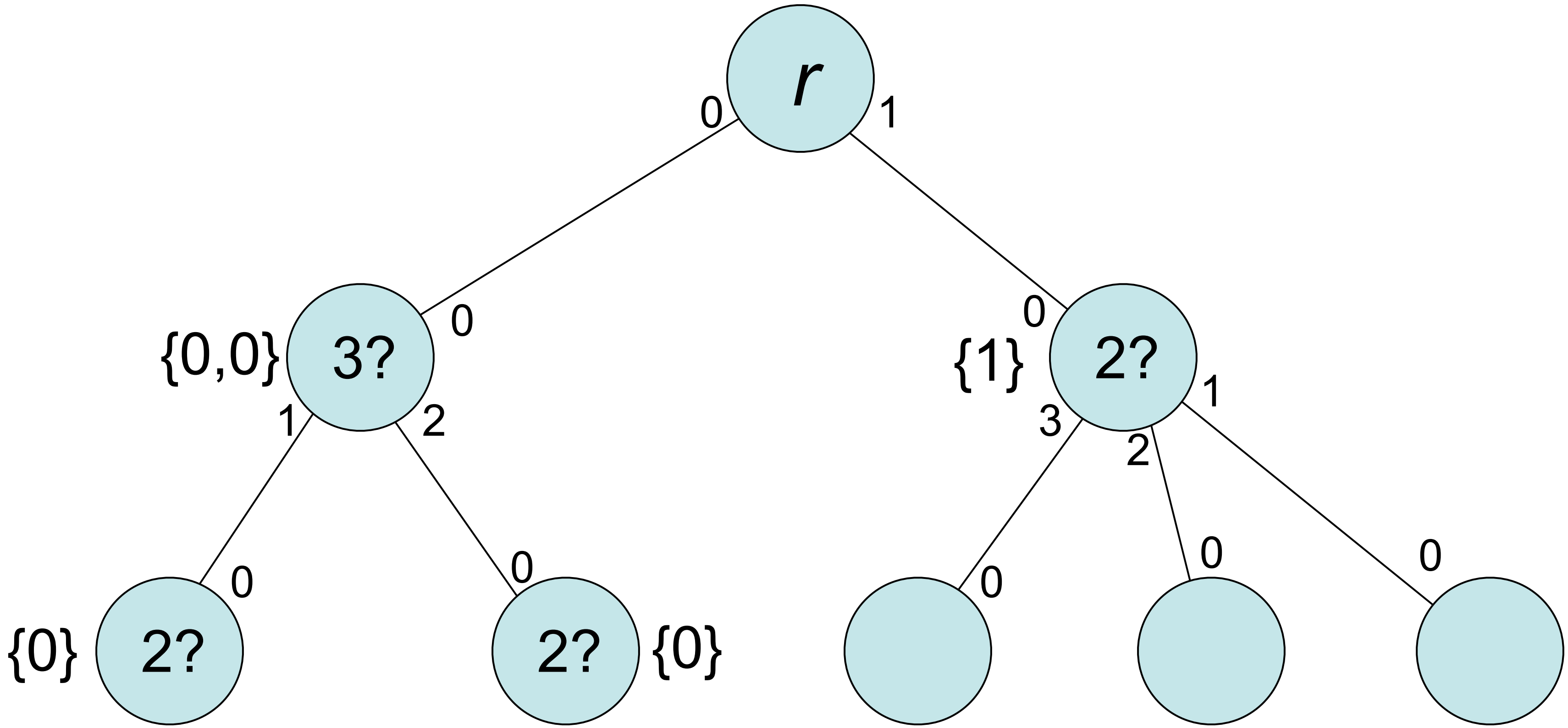
First idea:
K-out-of-*L* Exclusion = *L*-circulation



Deadlocks



Deadlocks

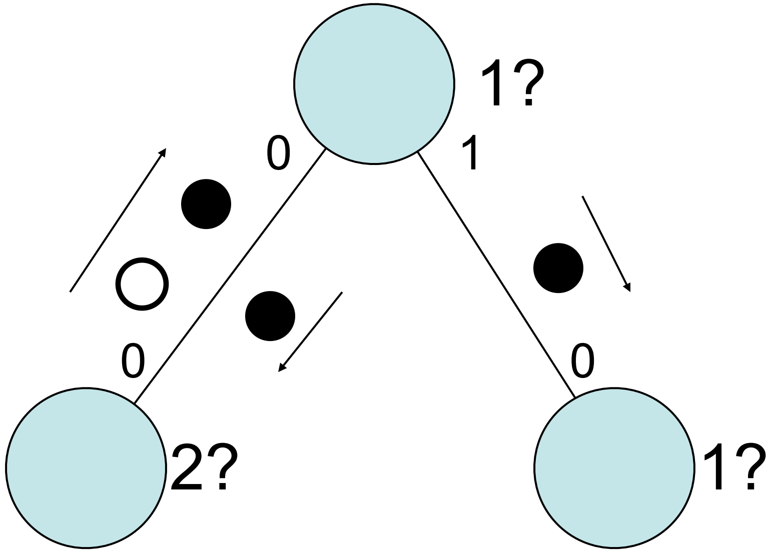


Solution

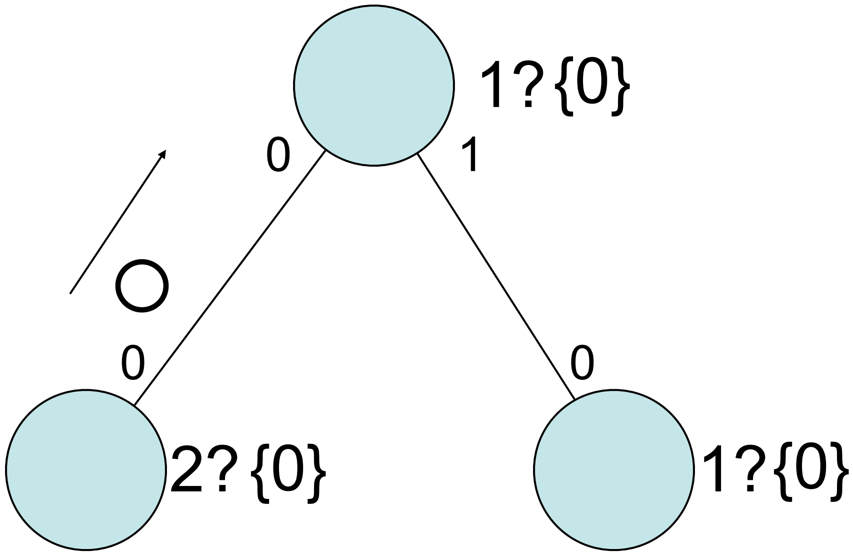
- Circulation of a special token :
  the *pusher*

- Upon receiving the *pusher*:
  a node releases all its resource tokens
  if its request is not satisfied

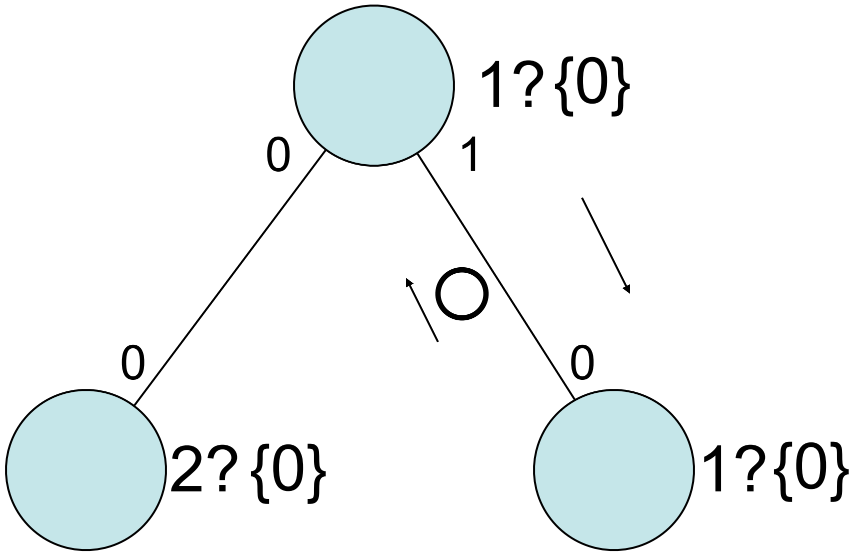
Livelock



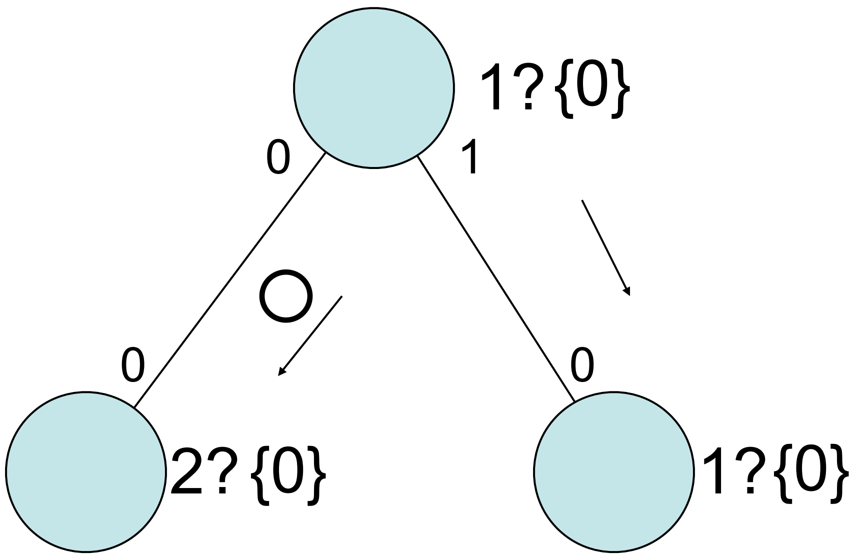
Livelock



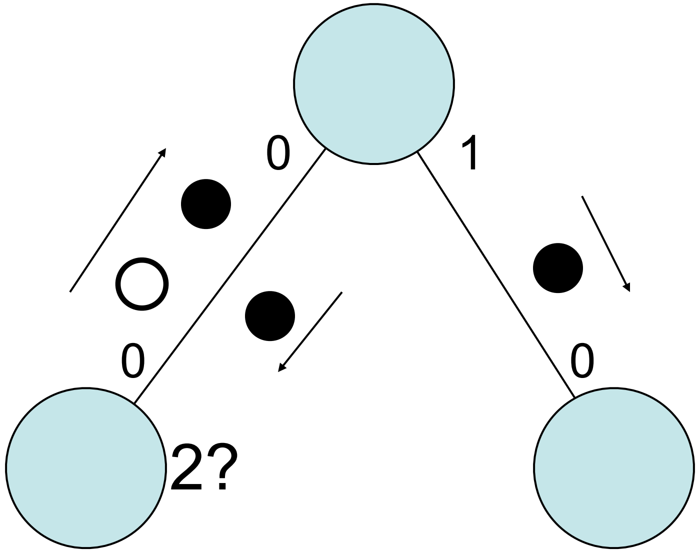
Livelock



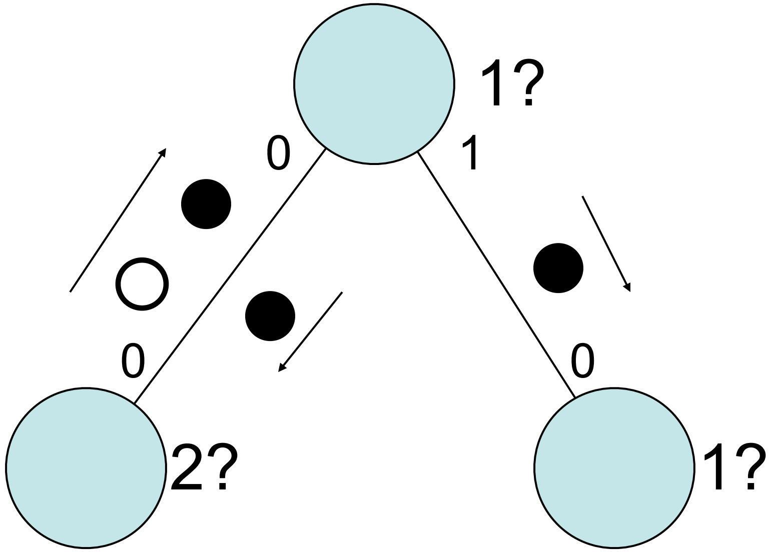
Livelock



Livelock



Livelock



Solution

- Circulation of a *Priority Token:*
  - A requesting process keeps the *Priority Token* while its request is unsatisfied
  - The *Priority Token* cancels the effect of the *Pusher Token*

Self-Stabilization: The Controller

Self-Stabilization: The Controller

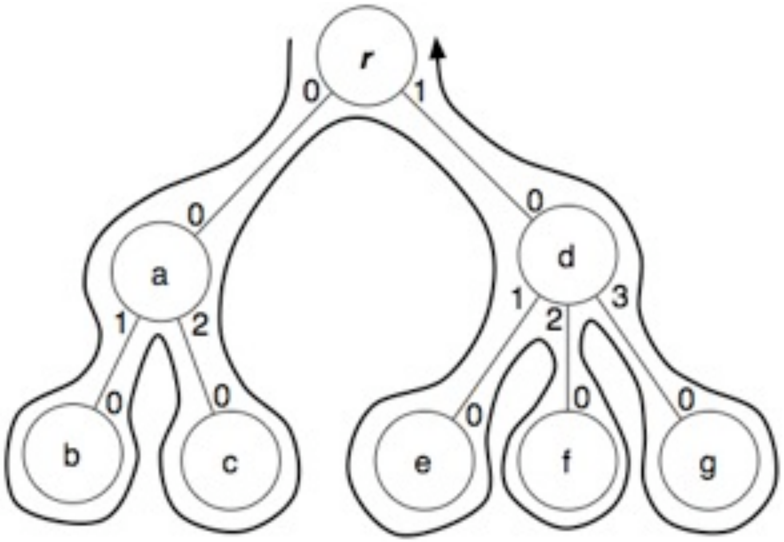
- After transient faults:
  - Some tokens may have disappeared
  - Some tokens may been duplicated

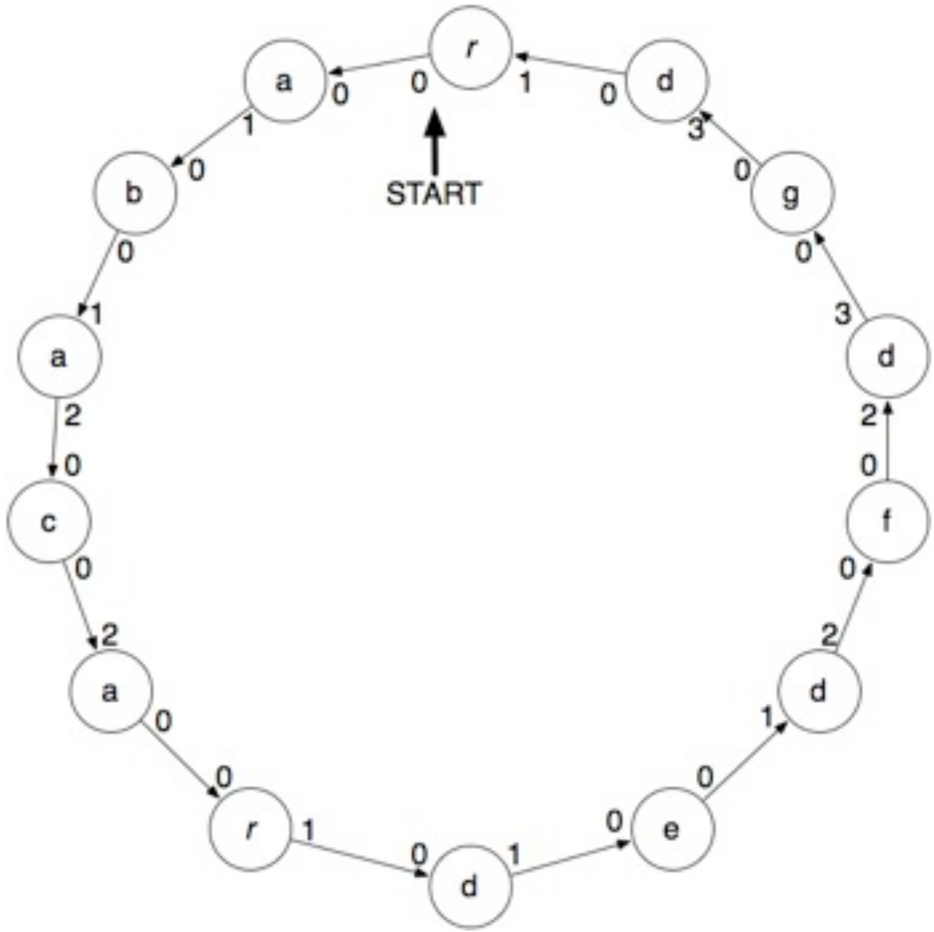
Self-Stabilization: The Controller

- After transient faults:
  - Some tokens may have disappeared
  - Some tokens may been duplicated

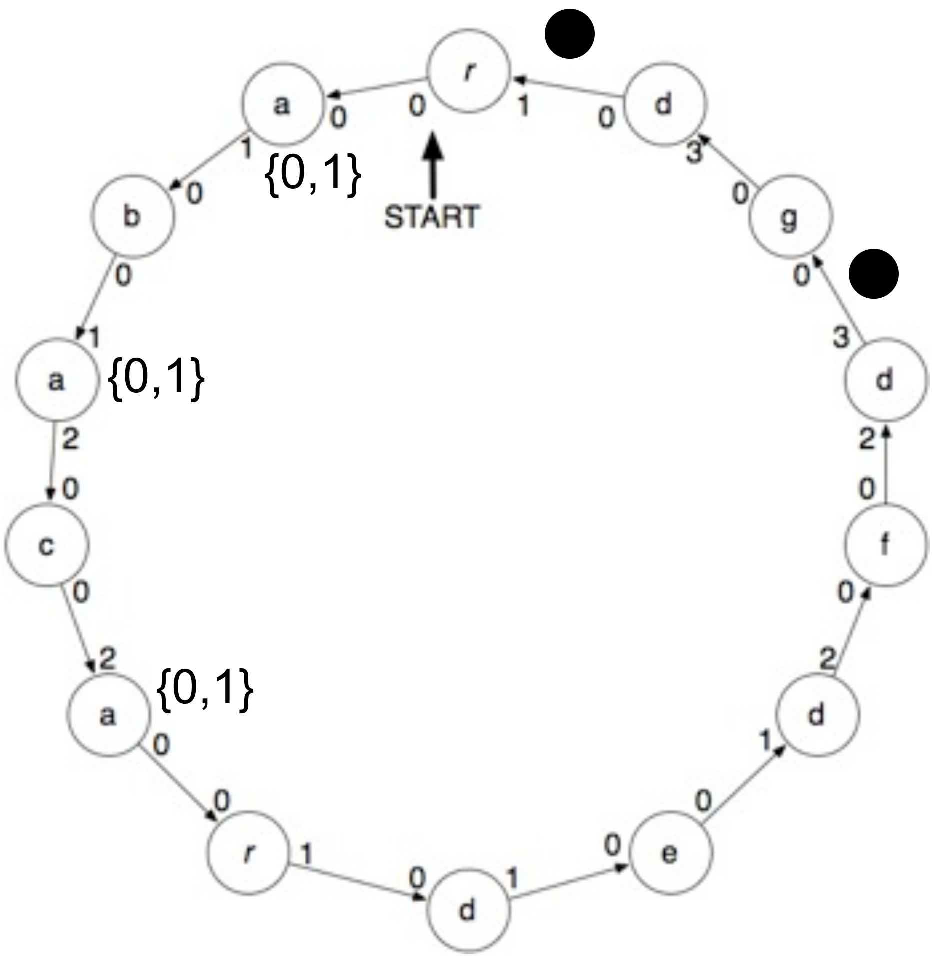
- Solution: a *Controller Token* counts and regulates the number of tokens

Example: Count the *Resource Tokens*

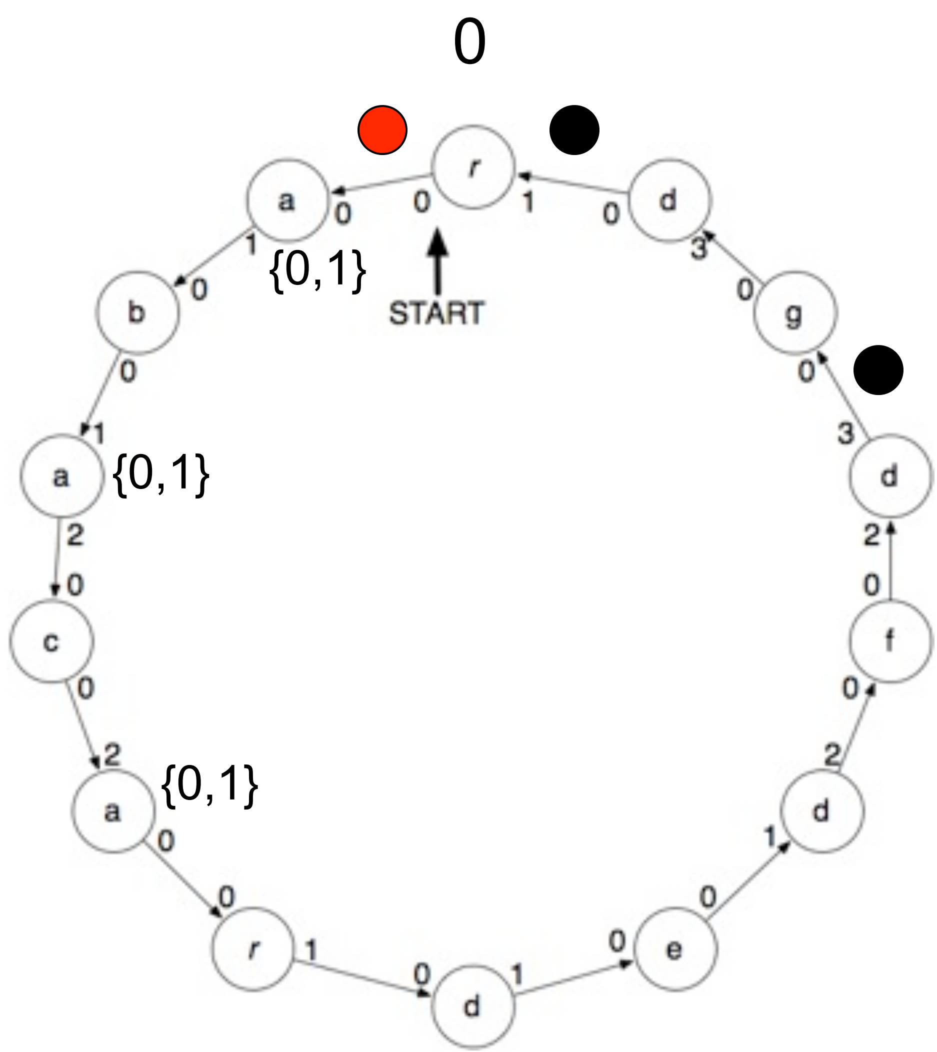




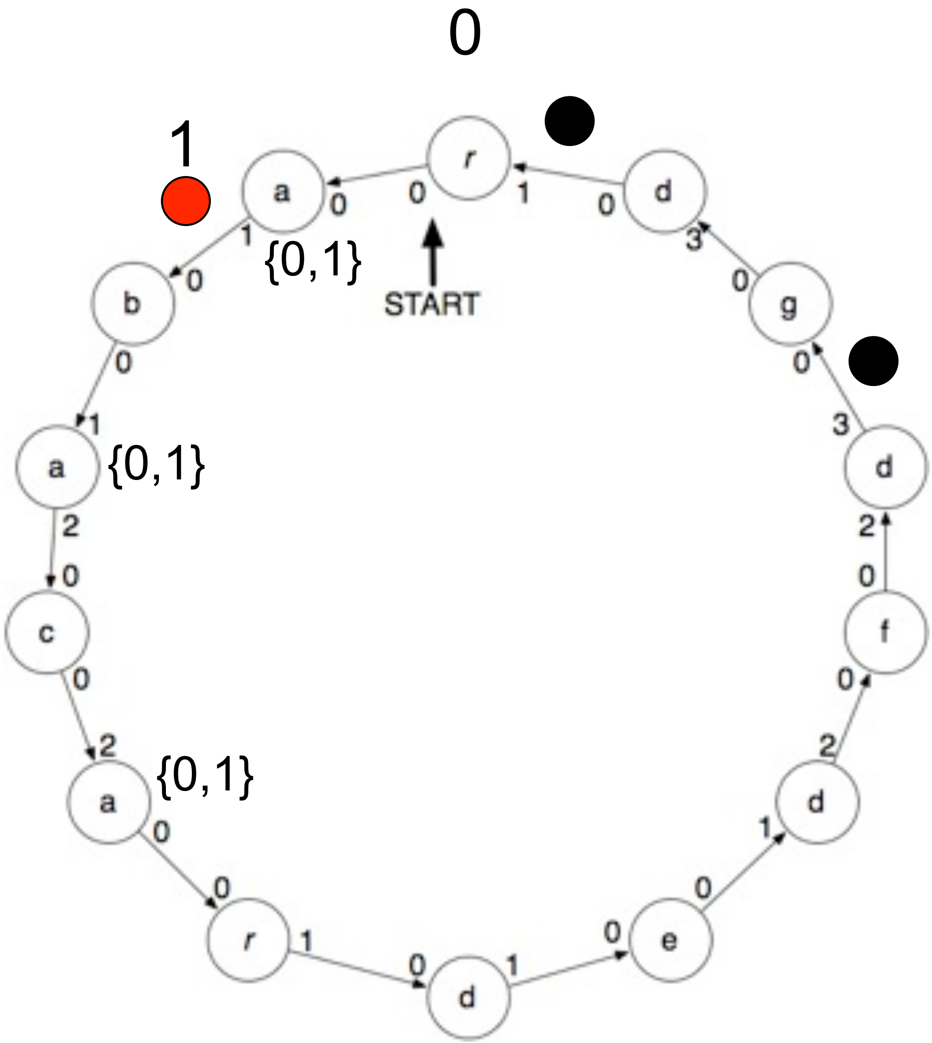
Example: Count the *Resource Tokens*



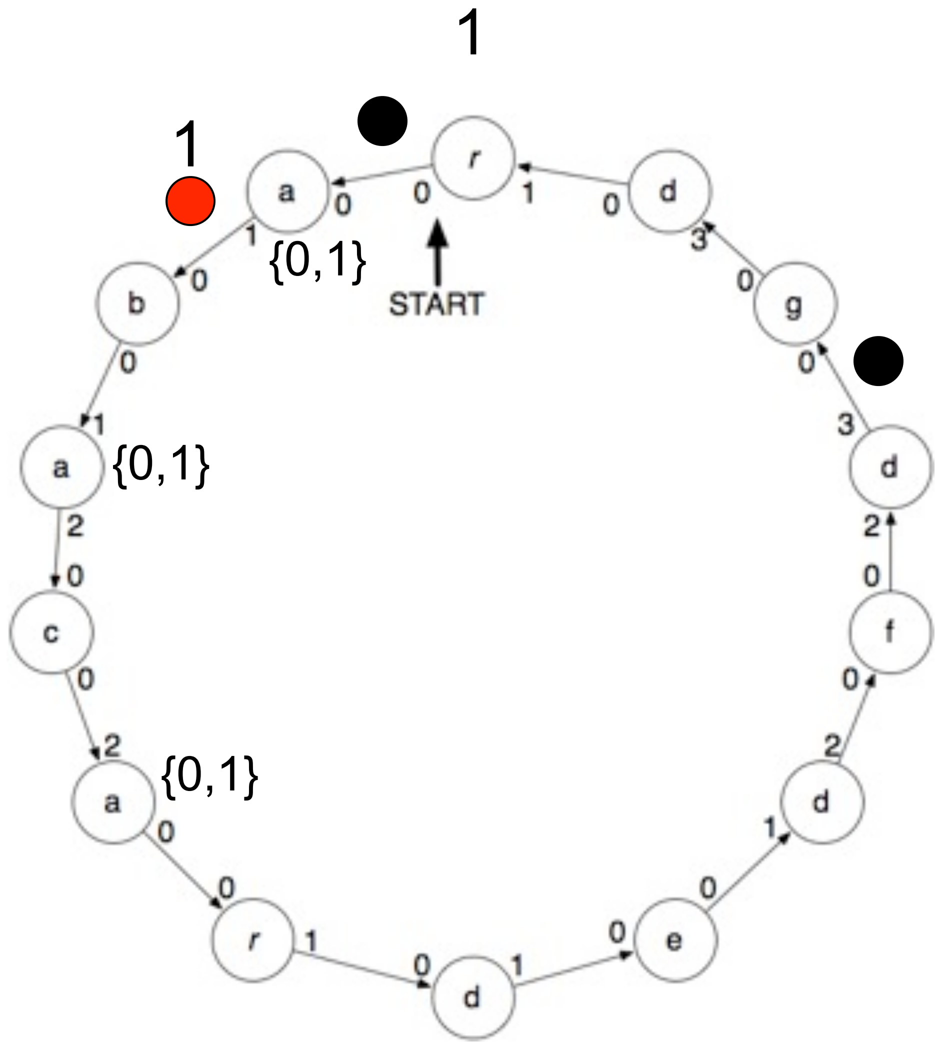
Example: Count the *Resource Tokens*



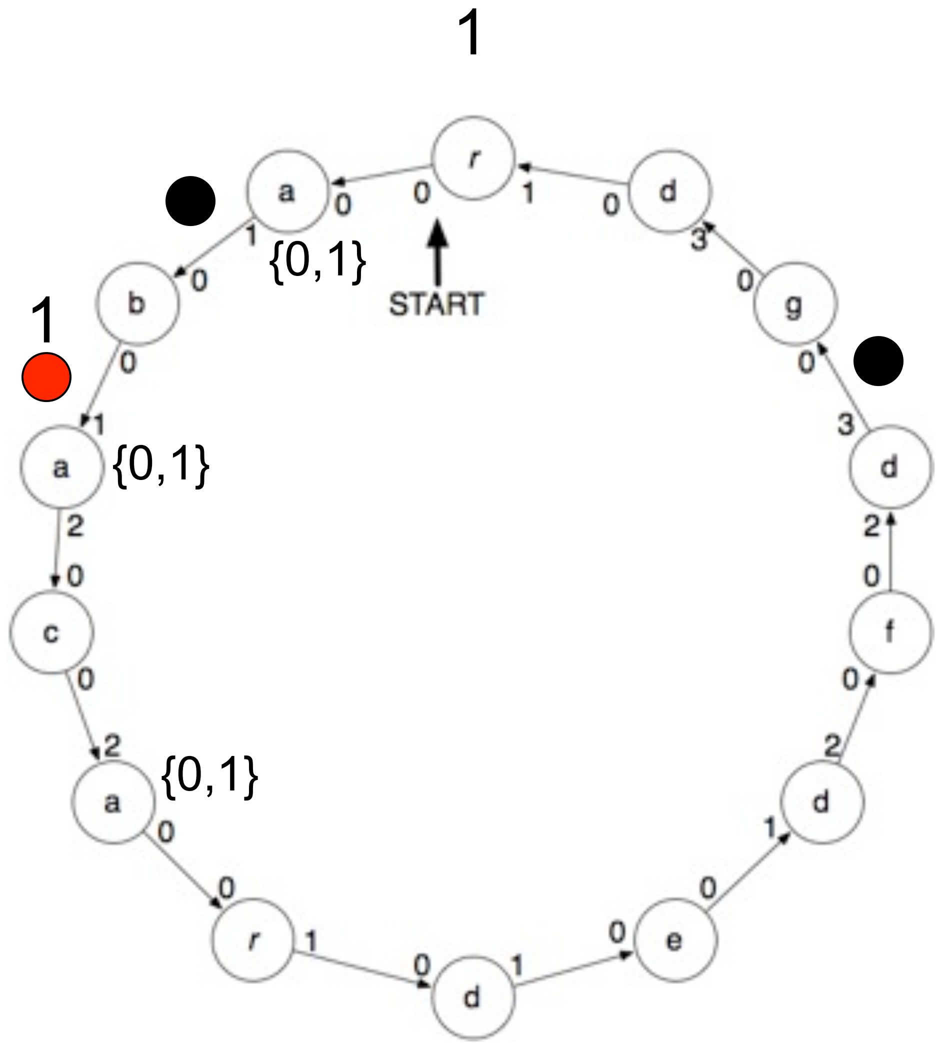
Example: Count the *Resource Tokens*



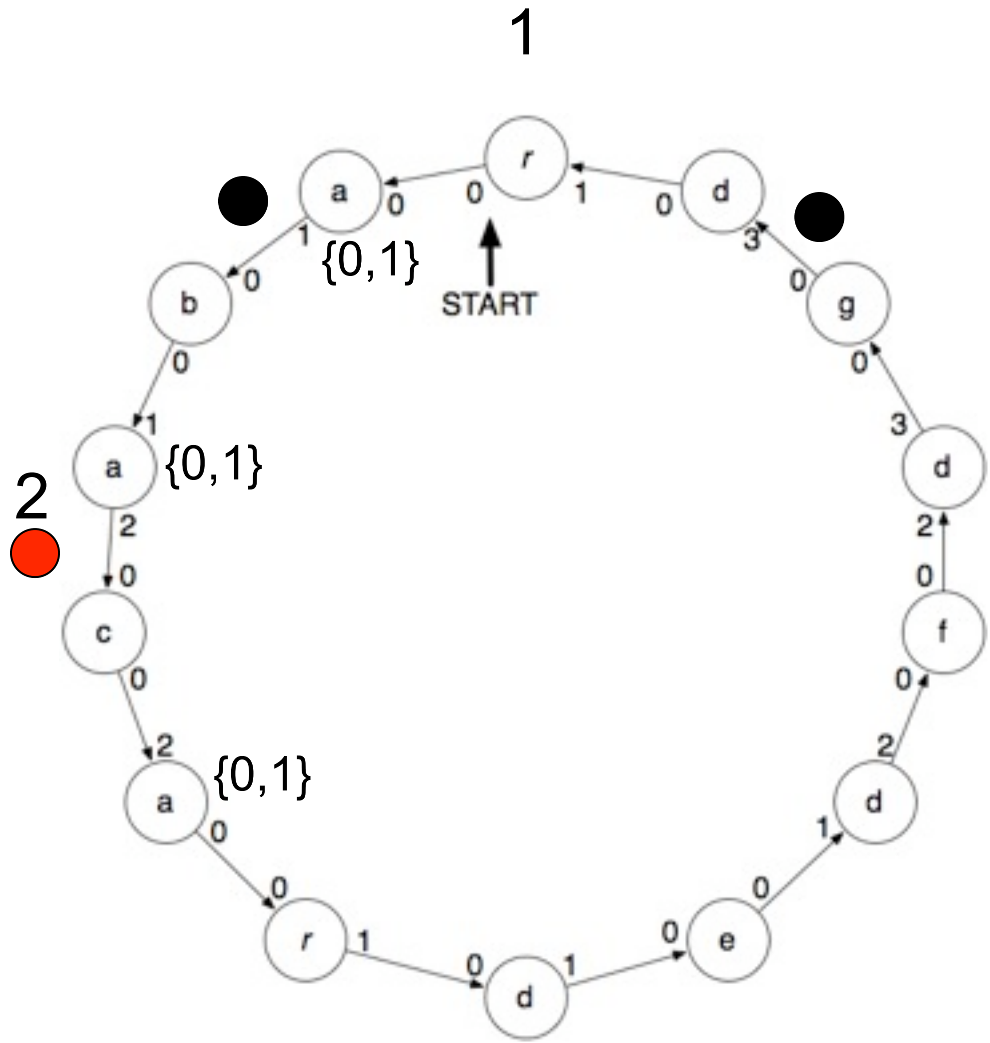
Example: Count the *Resource Tokens*



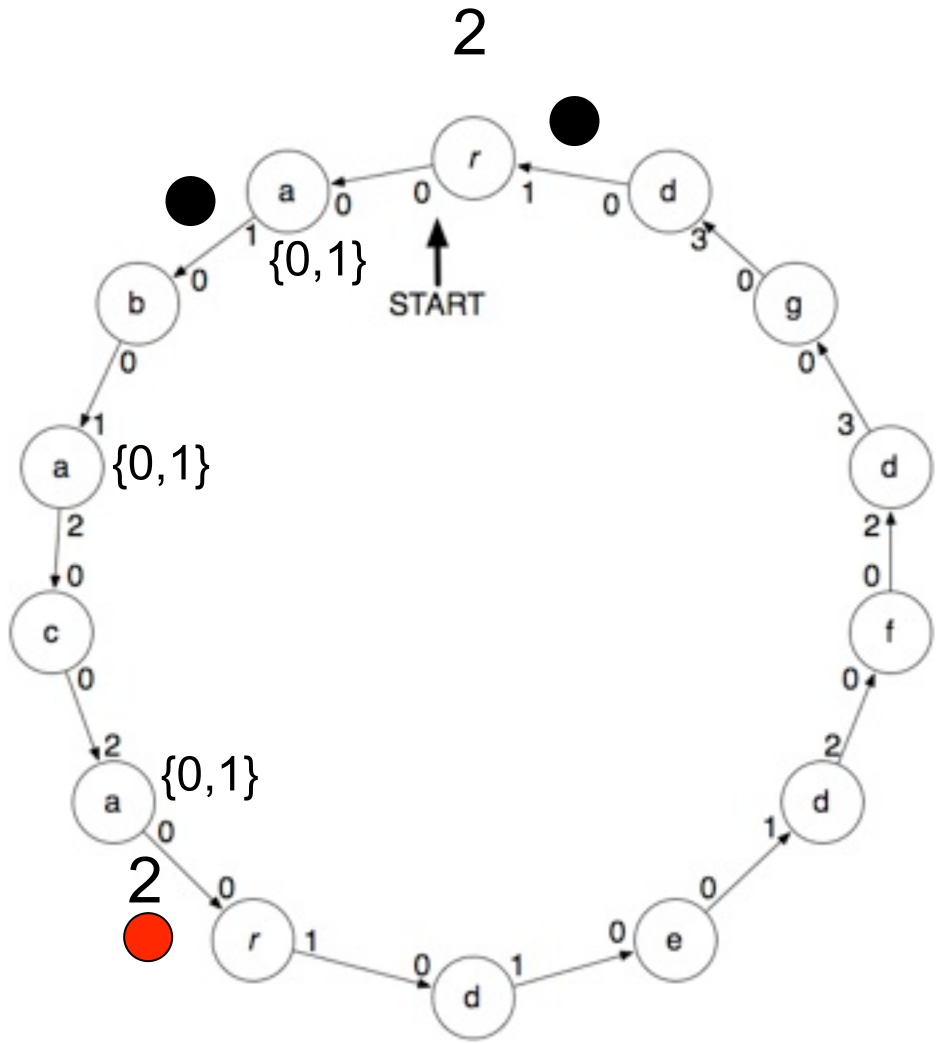
Example: Count the *Resource Tokens*



Example: Count the *Resource Tokens*



Example: Count the *Resource Tokens*



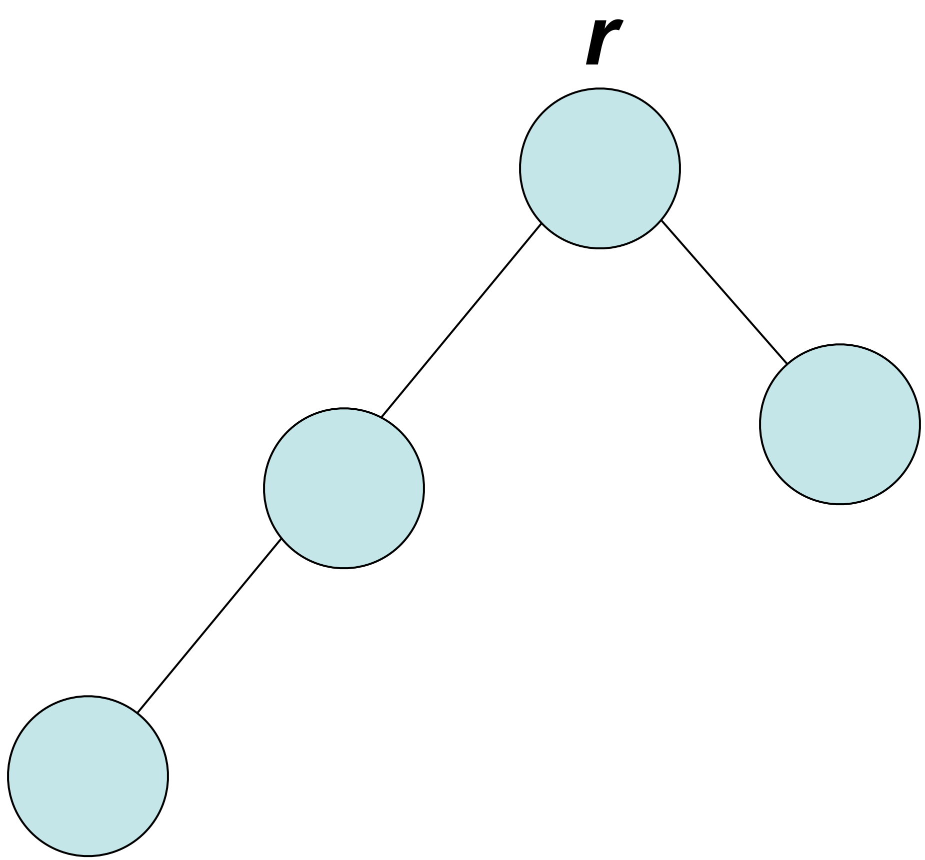
Example: Count the *Resource Tokens*

- At the end of the traversal:
  - The number of token is known
    - Too much tokens : RESET
    - Lake of tokens : Creation at the root

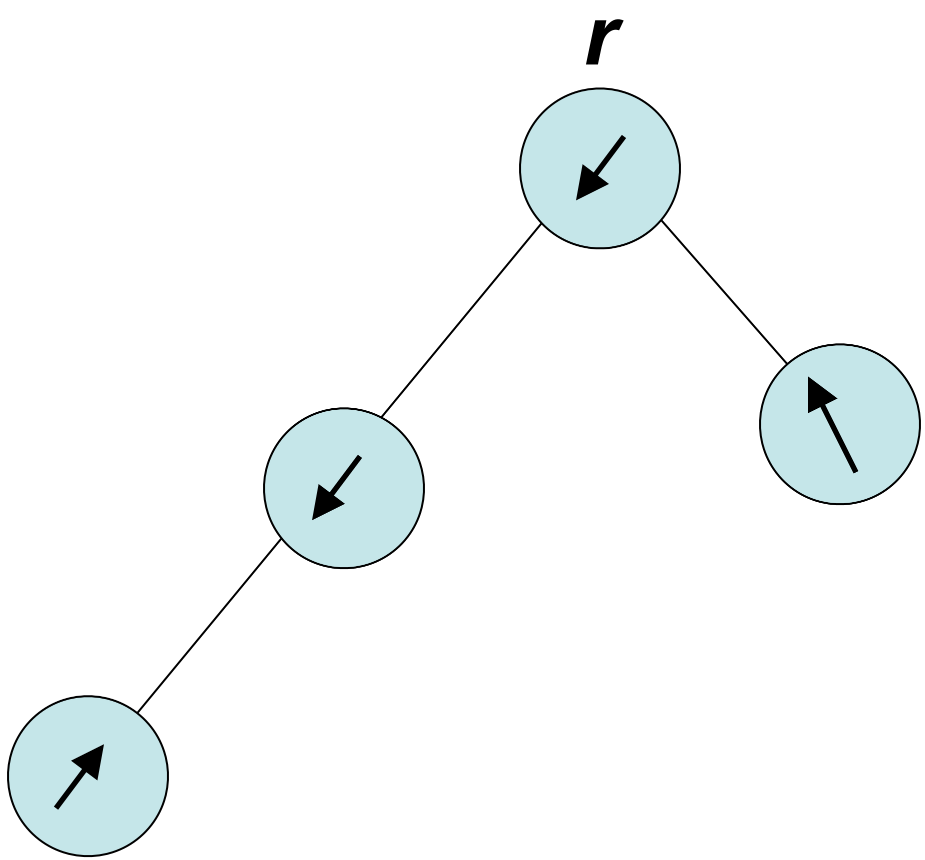
How to stabilize the *Controller* ?

- Implemented as a Self-Stabilizing DFTC using the **Varghese Counter Flushing**

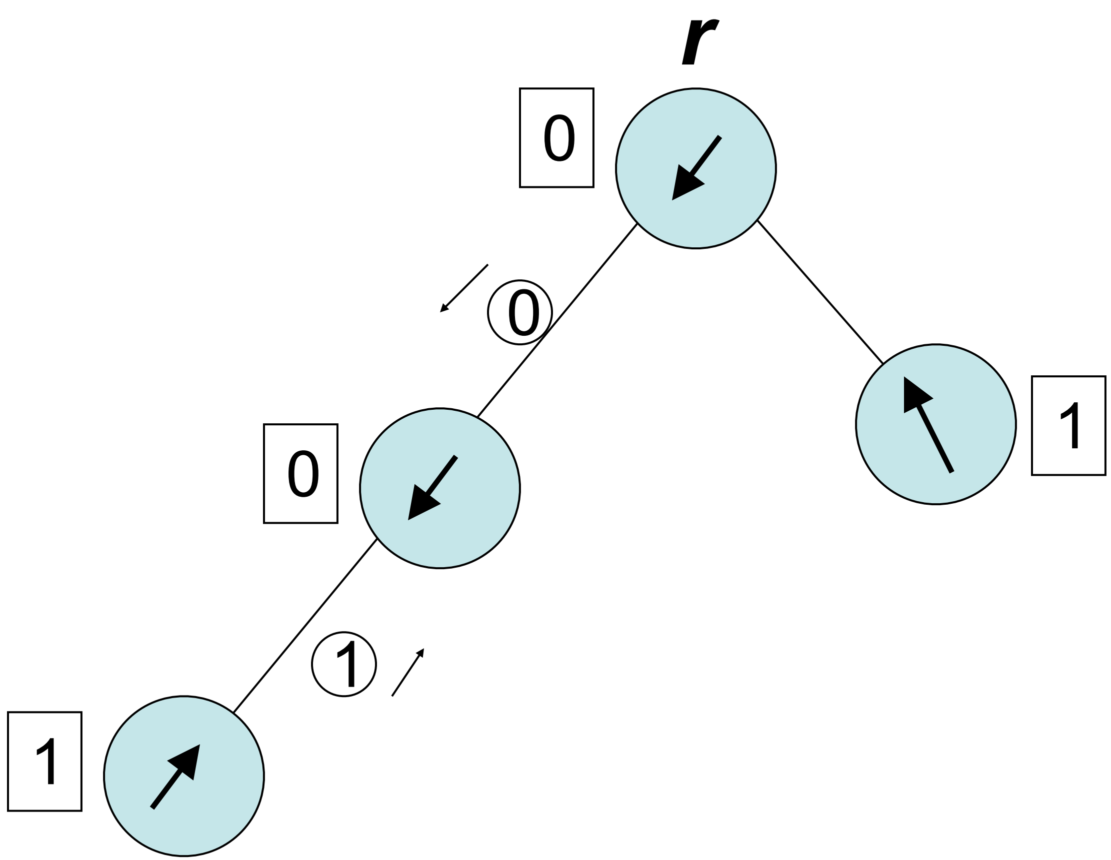
Self-Stabilizing DFTC



Self-Stabilizing DFTC



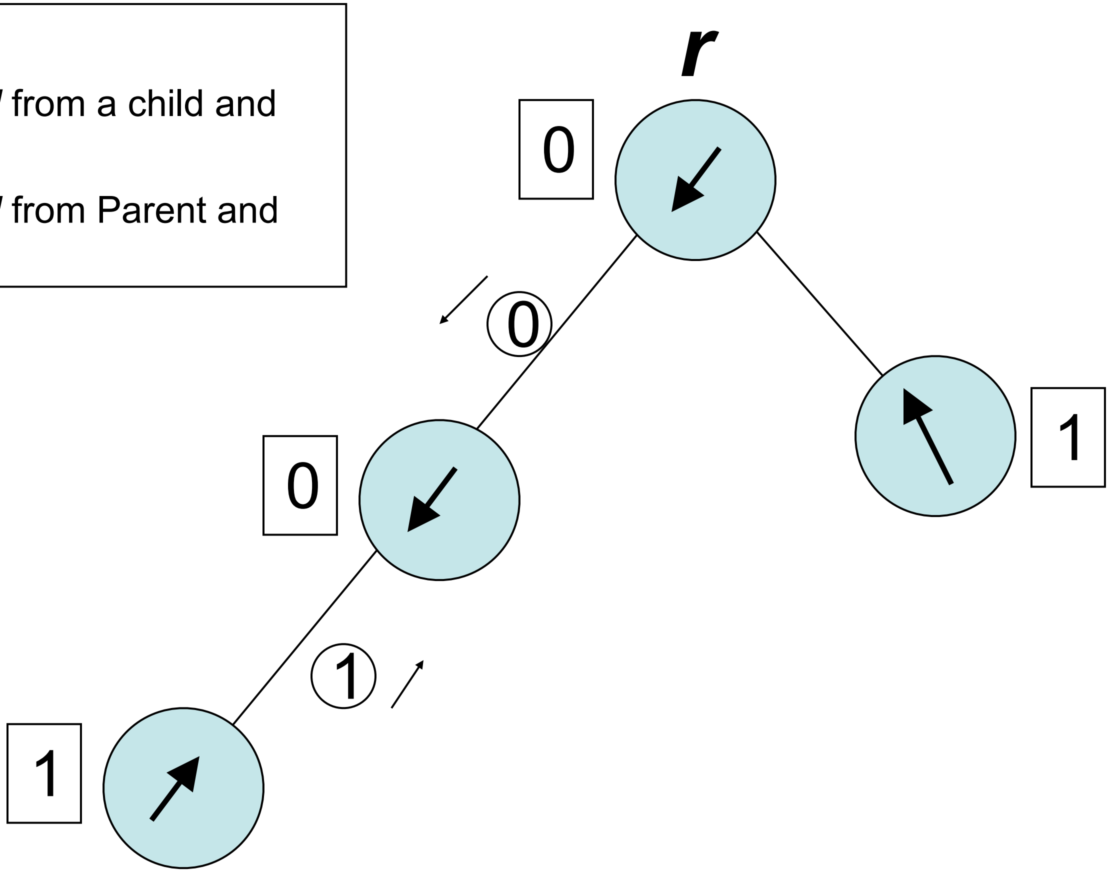
Self-Stabilizing DFTC



Self-Stabilizing DFTC

TokenHolder:

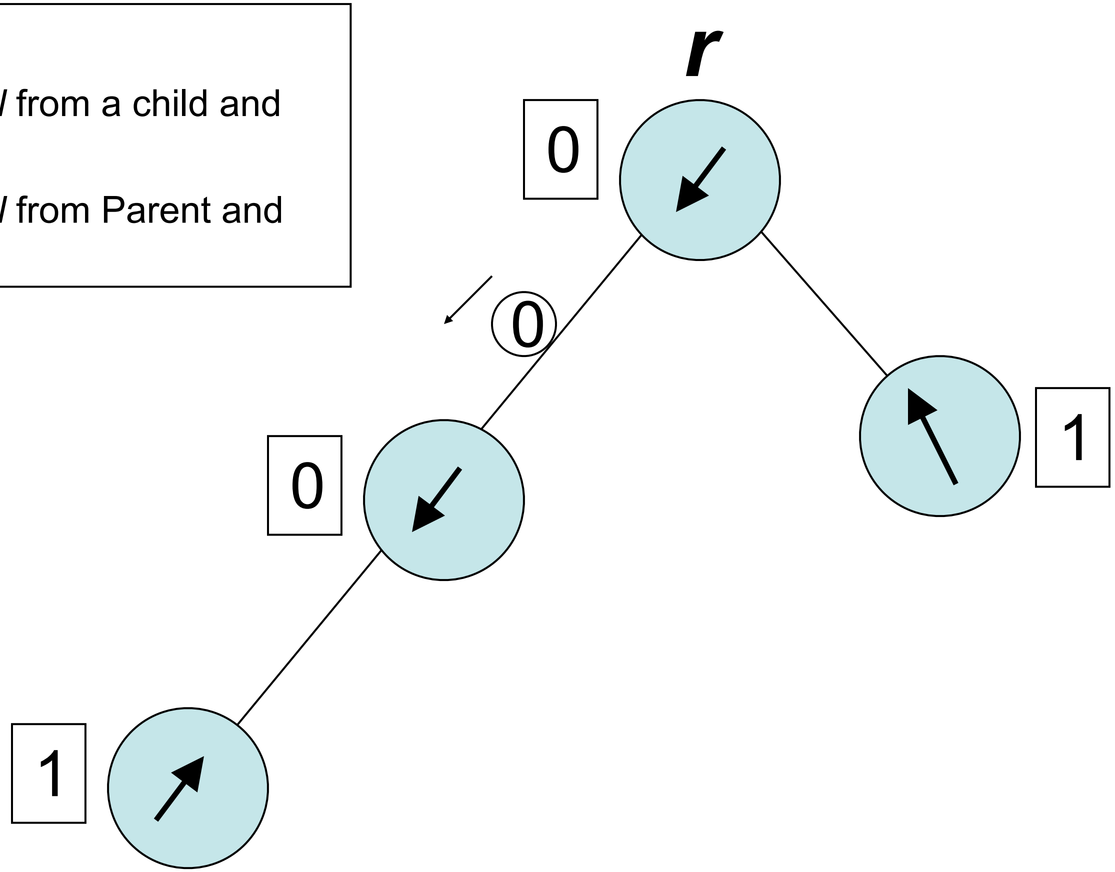
- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Self-Stabilizing DFTC

TokenHolder:

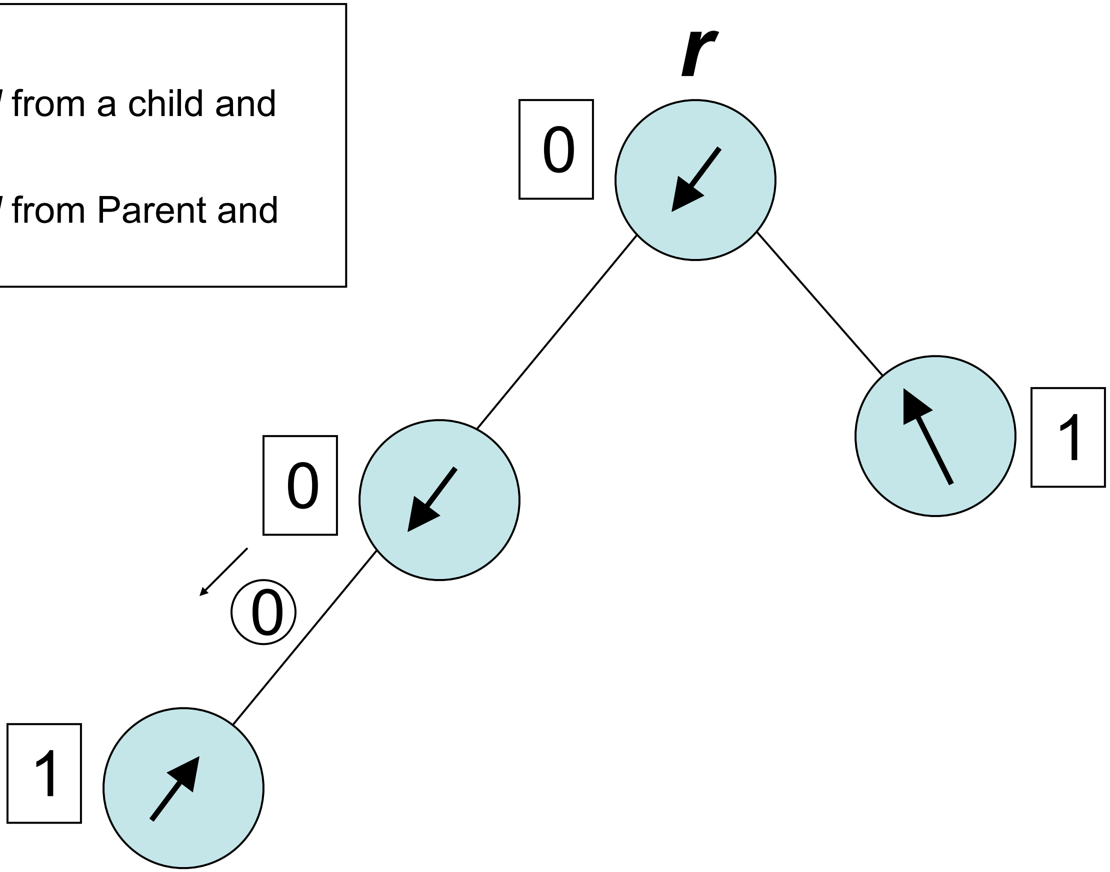
- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Self-Stabilizing DFTC

TokenHolder:

- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*

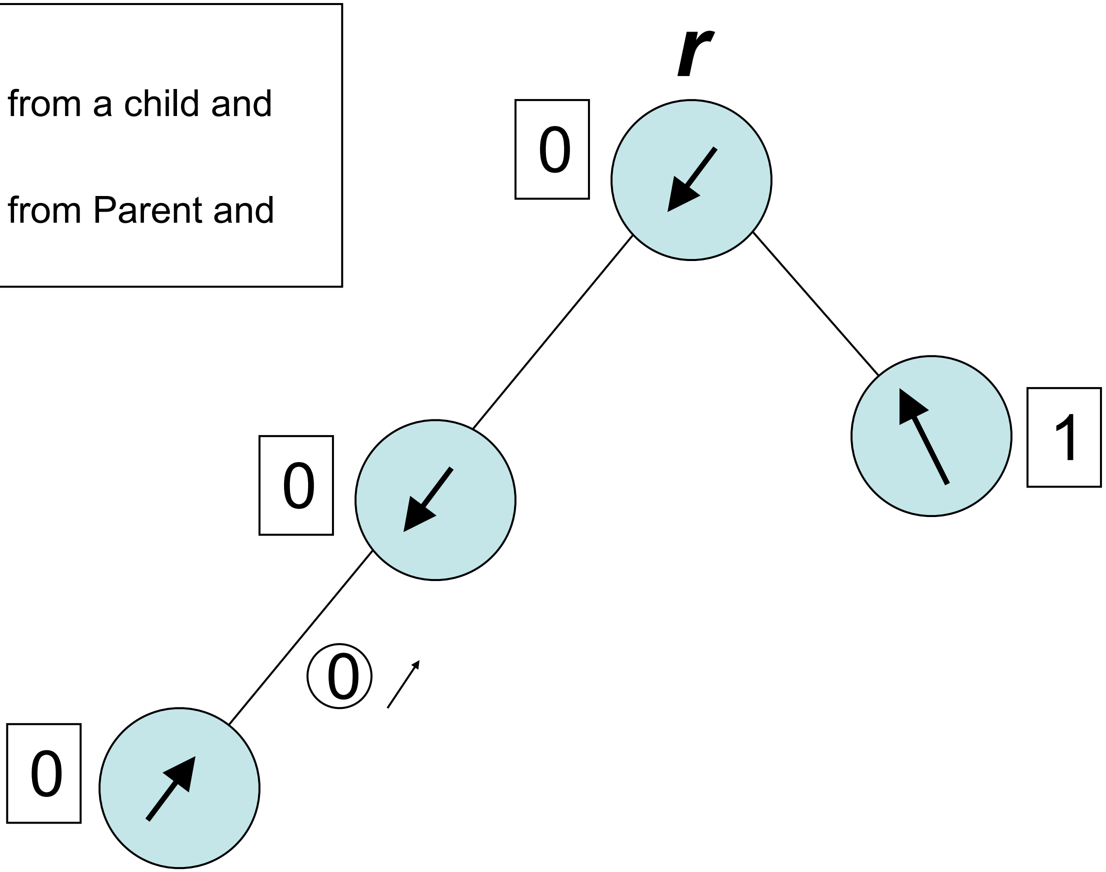


Self-Stabilizing DFTC

TokenHolder:

• Receive *MesVal* from a child and *MyVal* = *MesVal*

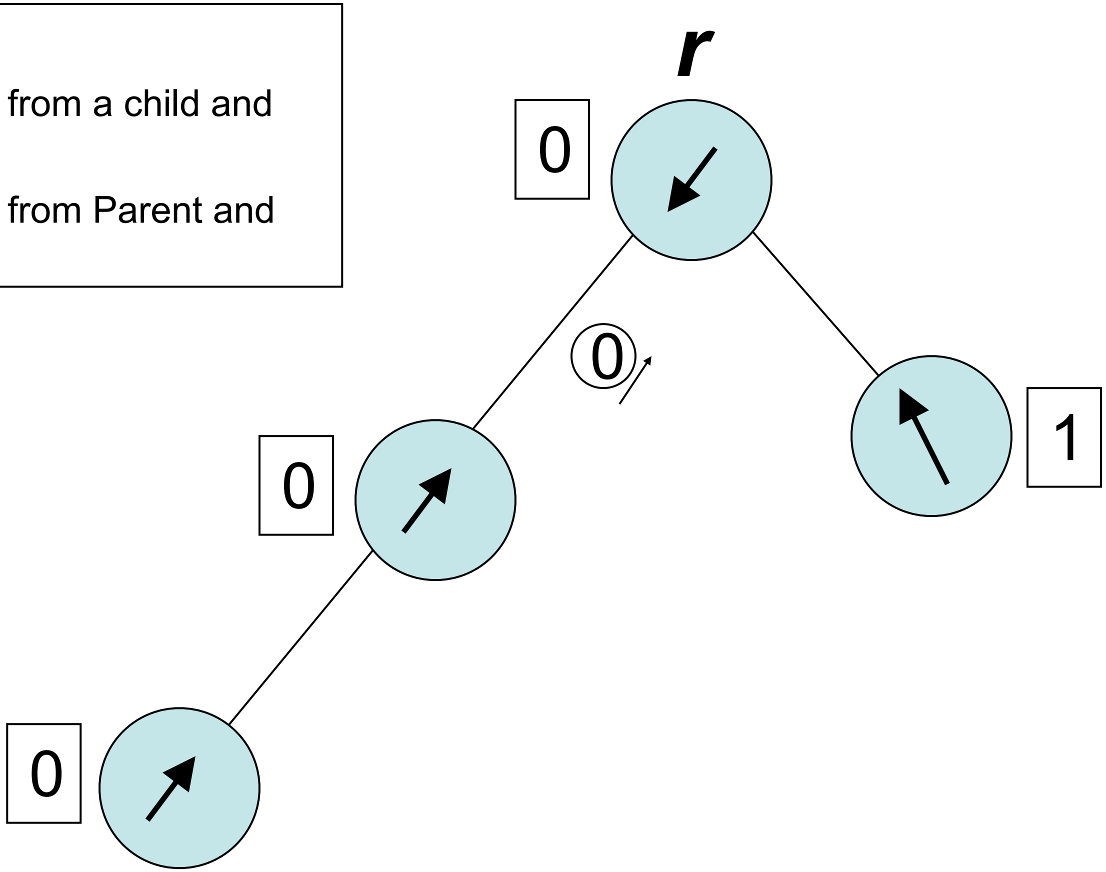
• Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Self-Stabilizing DFTC

TokenHolder:

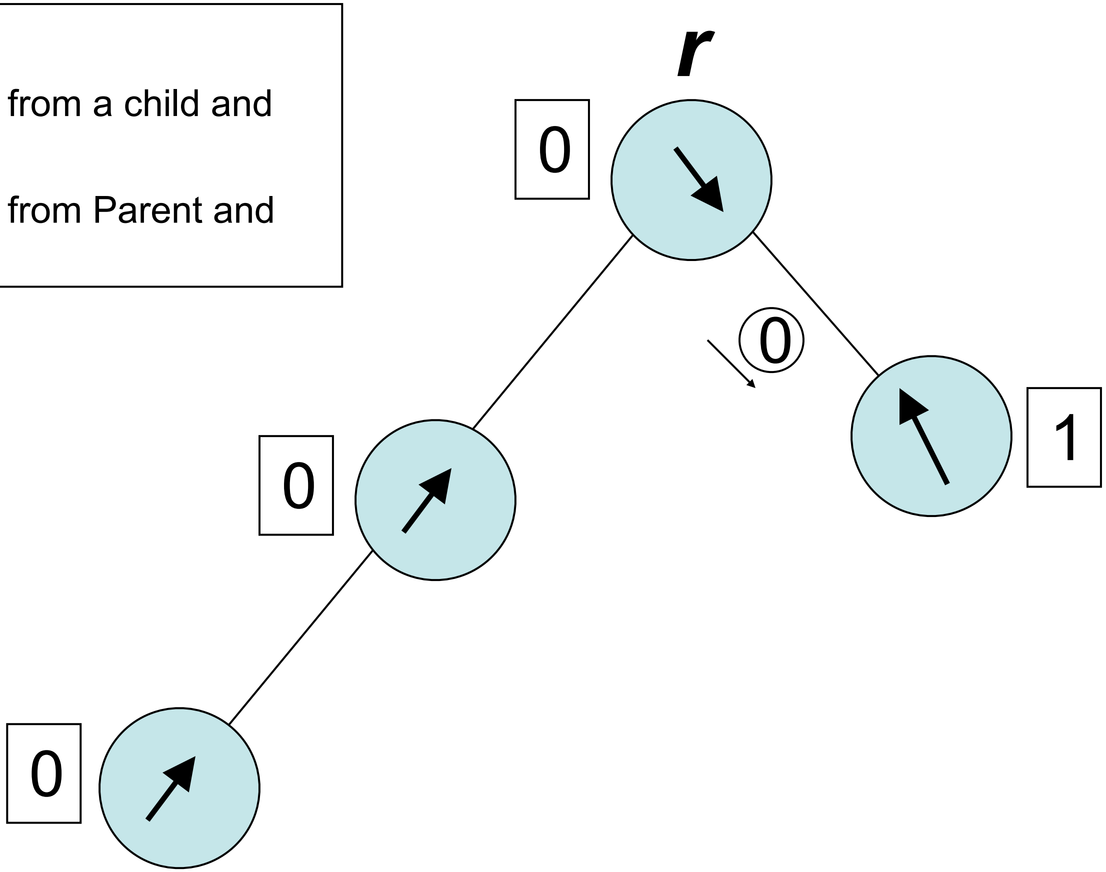
- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Self-Stabilizing DFTC

TokenHolder:

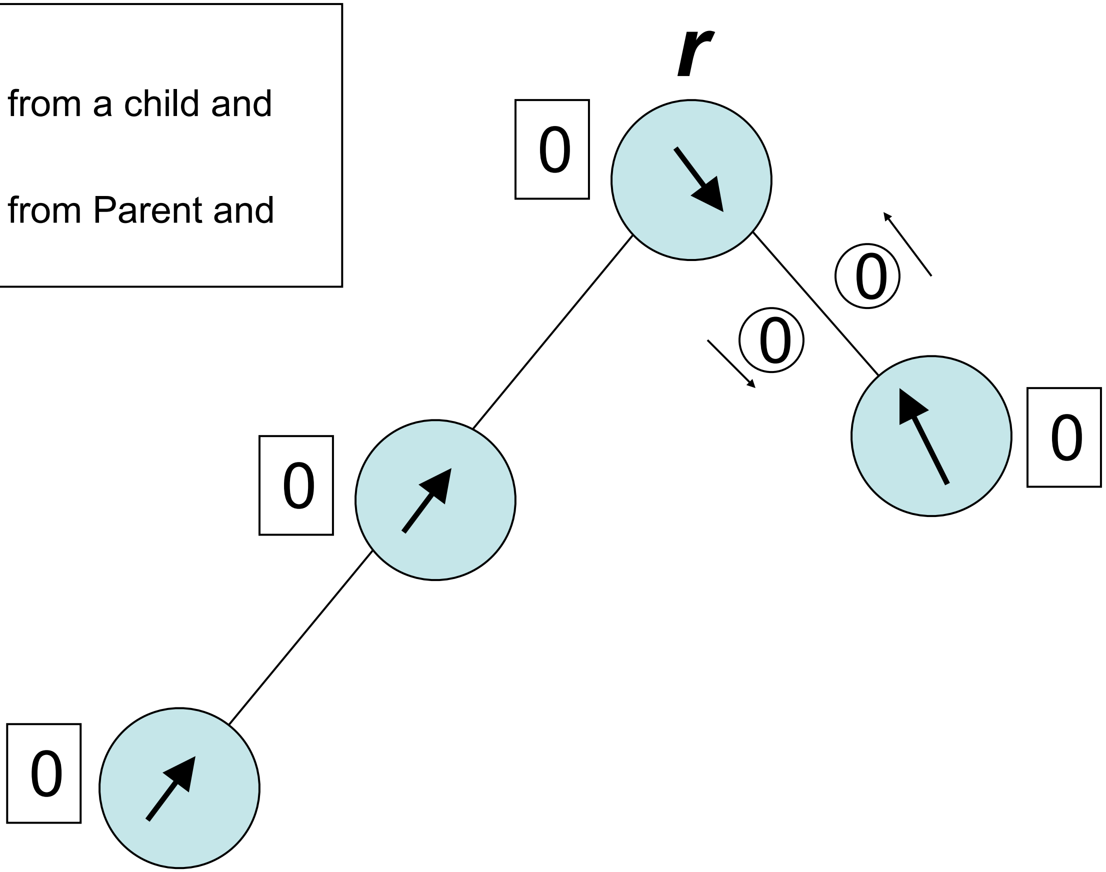
- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Self-Stabilizing DFTC

TokenHolder:

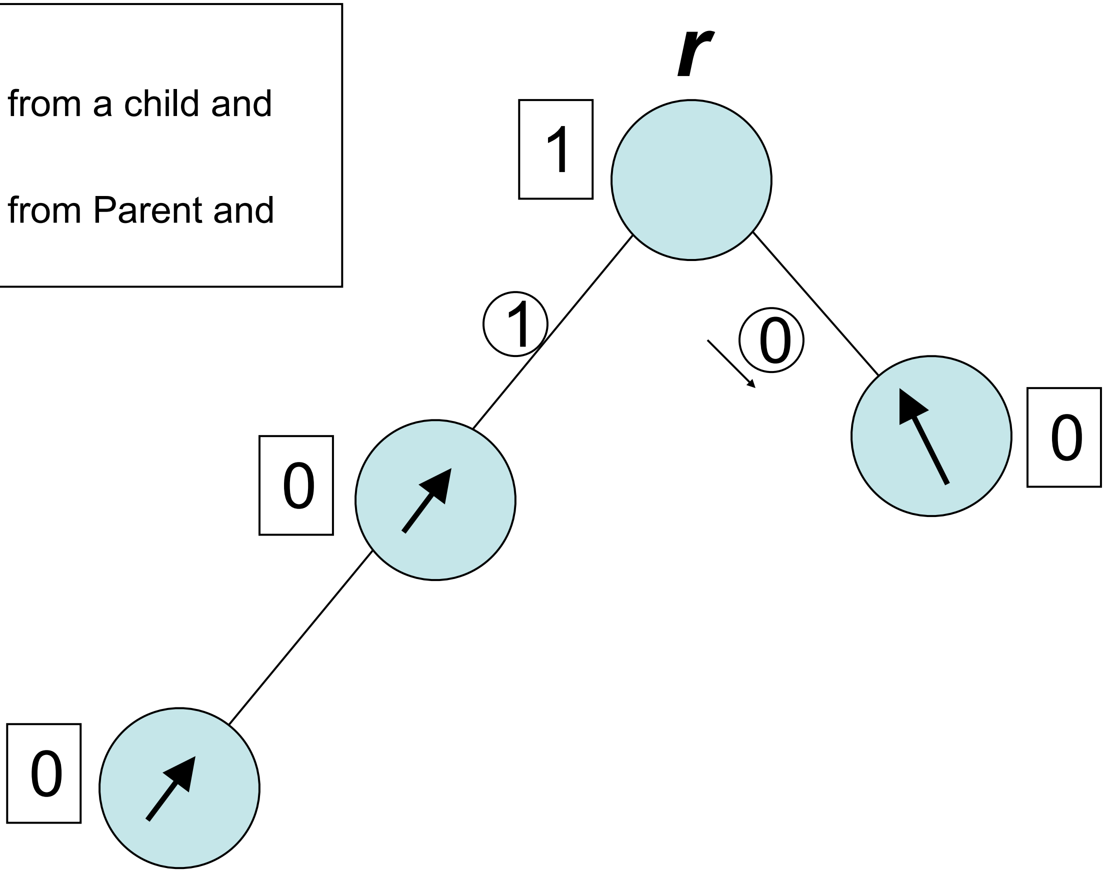
- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Self-Stabilizing DFTC

TokenHolder:

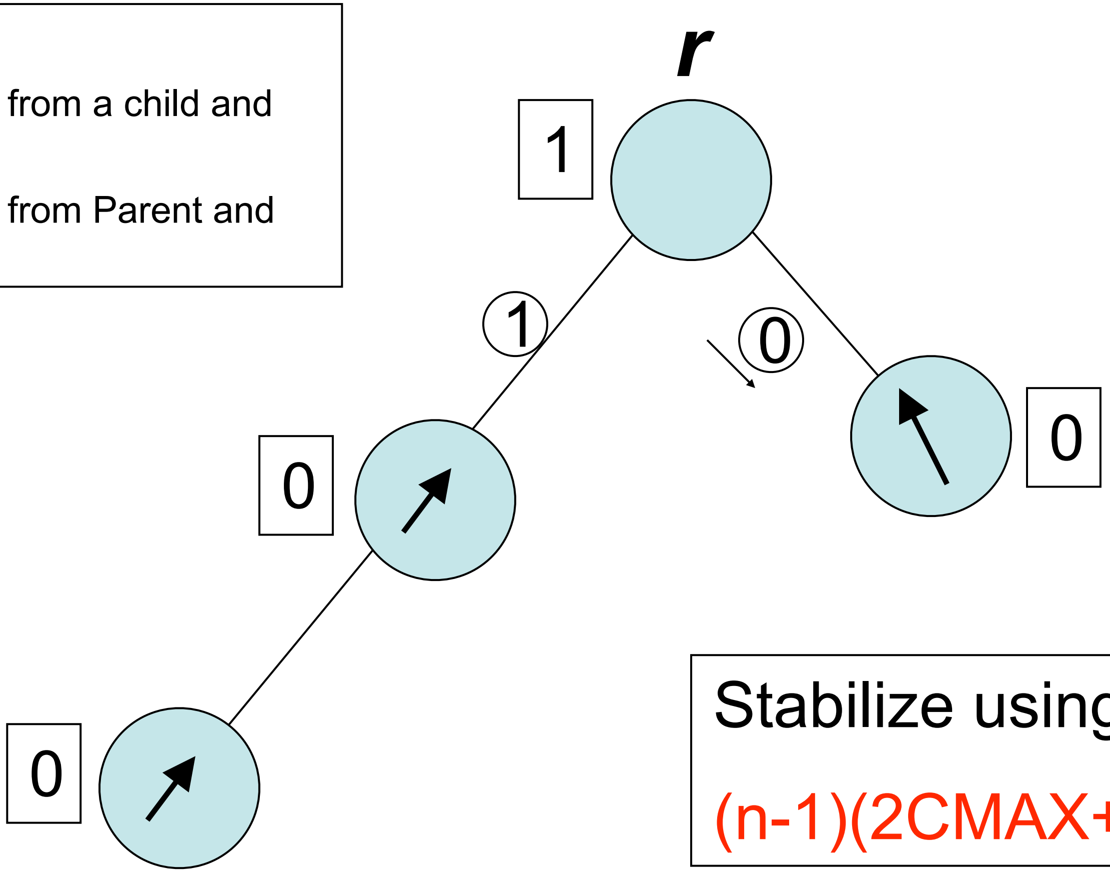
- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Self-Stabilizing DFTC

TokenHolder:

- Receive *MesVal* from a child and *MyVal* = *MesVal*
- Receive *MesVal* from Parent and *MyVal* ≠ *MesVal*



Stabilize using

(n-1)(2CMAX+1)+1 values

Roadmap

- Recall on Self-stabilization
- Definition of the problem
- The solution
- Conclusion and perspectives

Conclusion

Conclusion

- Waiting Time: $\boldsymbol{L^*(2n-3)^2}$

Conclusion

- Waiting Time: $L^*(2n-3)^2$
- Oriented Tree -> Arbitrary Rooted Network (Huang-Chen BFS Tree)

Conclusion

- Waiting Time: $\boldsymbol{L^*(2n-3)^2}$
- Oriented Tree -> Arbitrary Rooted Network

  (Huang-Chen BFS Tree)
- Bounded/Unbounded Link Capacity

  (Katz & Perry)

Perspectives

Perspectives

- Compute the convergence time

Perspectives

- Compute the convergence time
- Enhance the waiting time ?

Perspectives

- Compute the convergence time
- Enhance the waiting time ?
- Reactive solution ?

Perspectives

- Compute the convergence time
- Enhance the waiting time ?
- Reactive solution ?
- Fault-Containment ?

Thank you!

(K,L)-Liveness

- Let ***V*** be the set of processes
- Let ***I*** be a subset of processes that execute their critical section forever (in particular they hold some resource units forever)
- Let α be the number of resource units held by ***I***
- Let ***R*** be the subset of ***V - I*** such that any process in ***R*** is a requestor
- Let r_{max} by the maximal request of a process in ***R***
- If $R \neq \emptyset$ and $r_{max} \leq L - \alpha$ then at least one member of ***R*** eventually satisfies its request