Distributed Computing
PWr, WIiT, 2021

informatyka algorytmiczna

Prof. Mirosław Kutyłowski

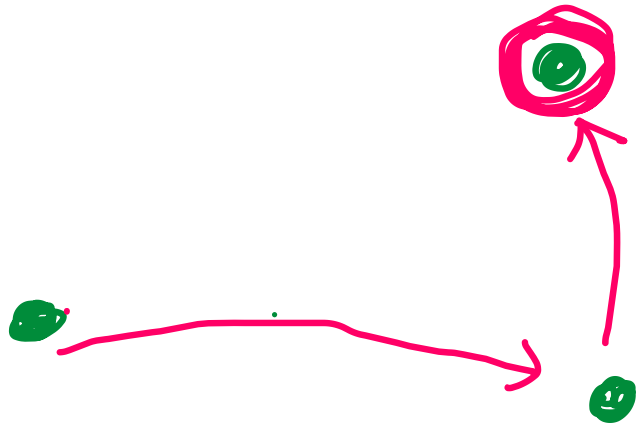
4: Self-stabilization

Problems

- programming errors
- algorithm faults
- failures
- Byzantine communication
- (temporal) subversion

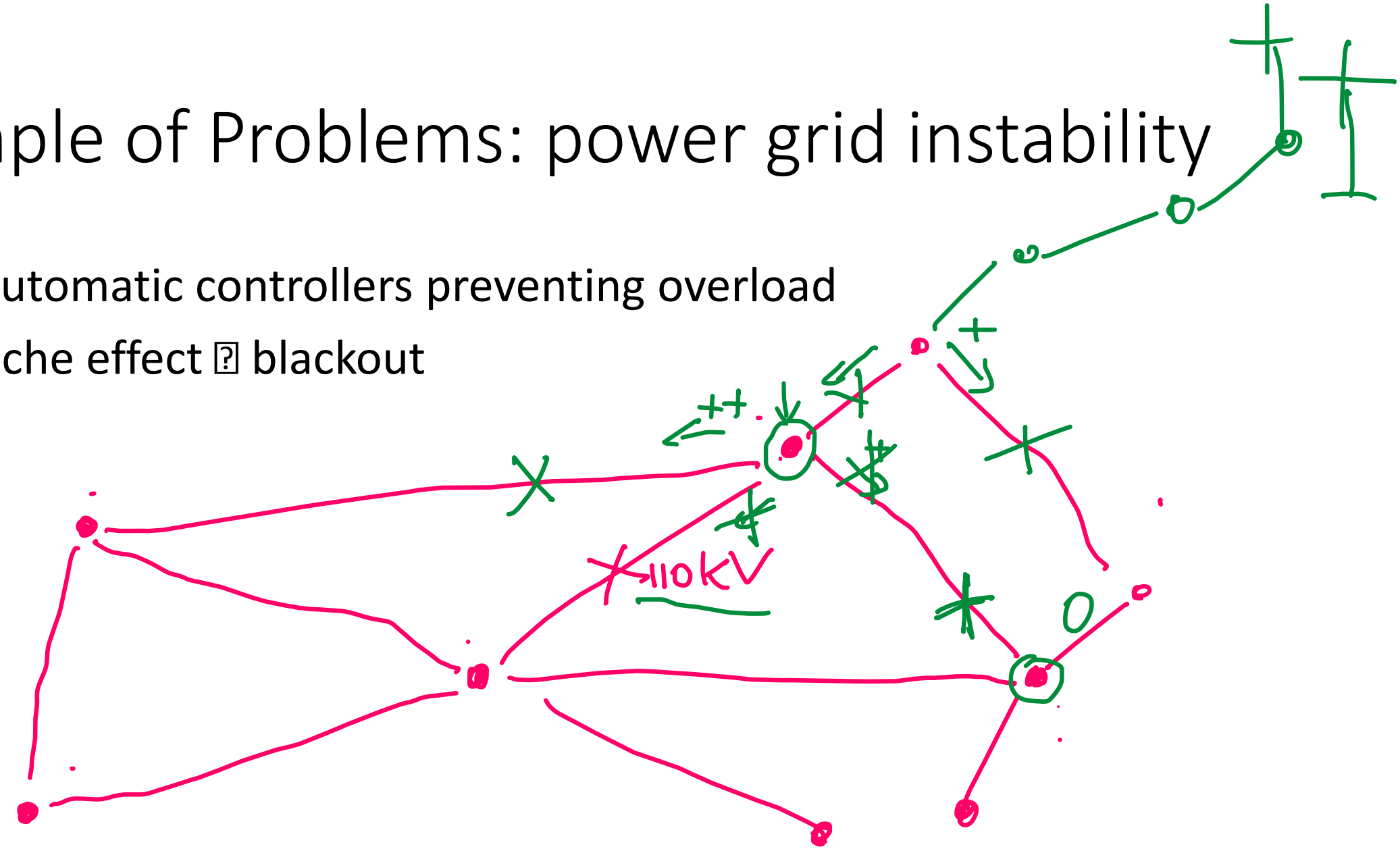
$p^3 = 0.00000001^3$

IoT



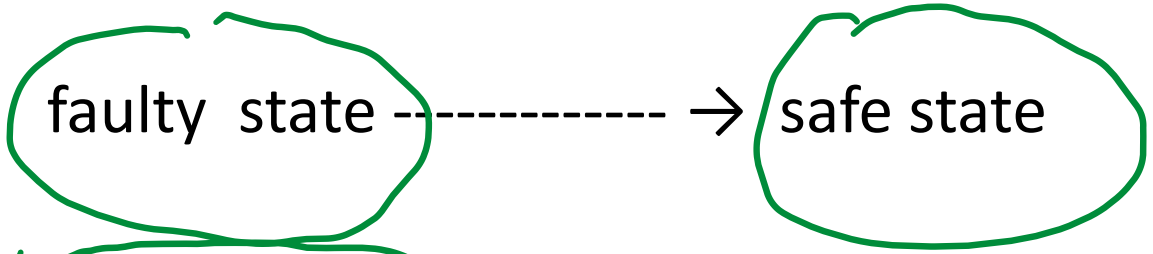
Example of Problems: power grid instability

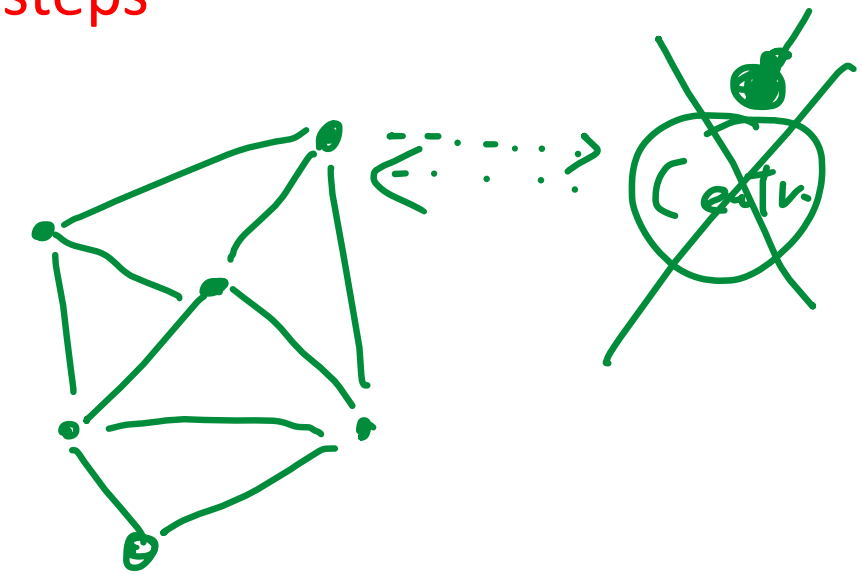
- local automatic controllers preventing overload
- avalanche effect ⍰ blackout



Self-stabilization idea

No matter what is the current state, if the network is working now properly, then it will reach a safe state after some number of steps

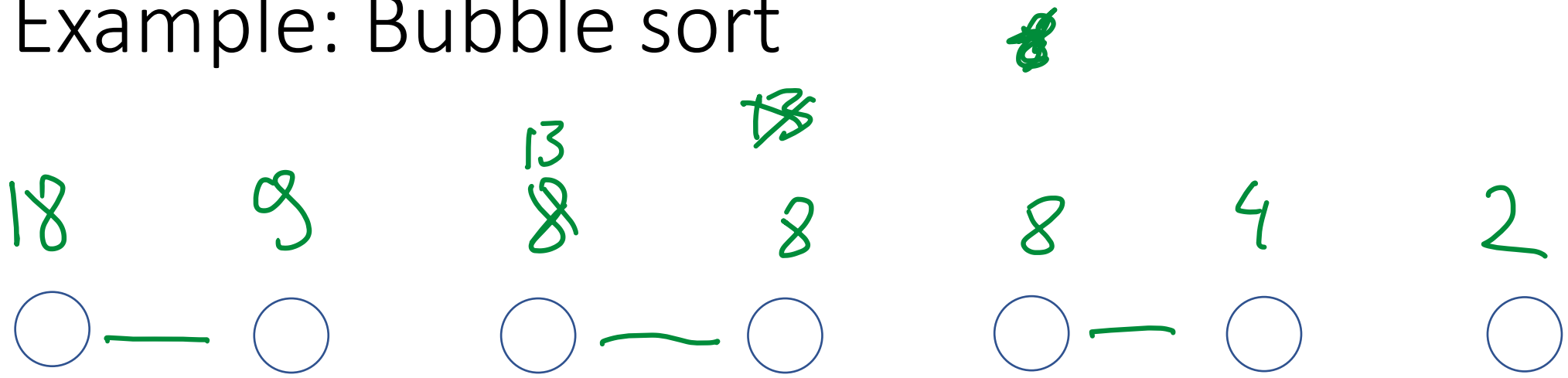




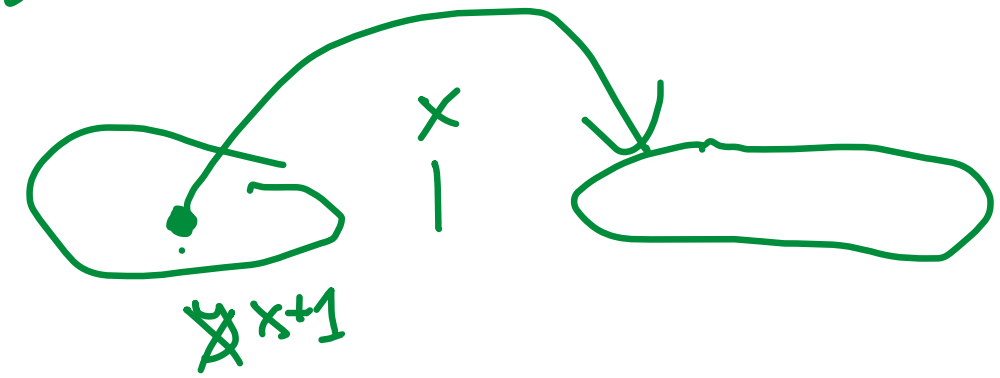
- automatically
- no global control

(It cannot be immediate, as the system is distributed and no instant information propagation)

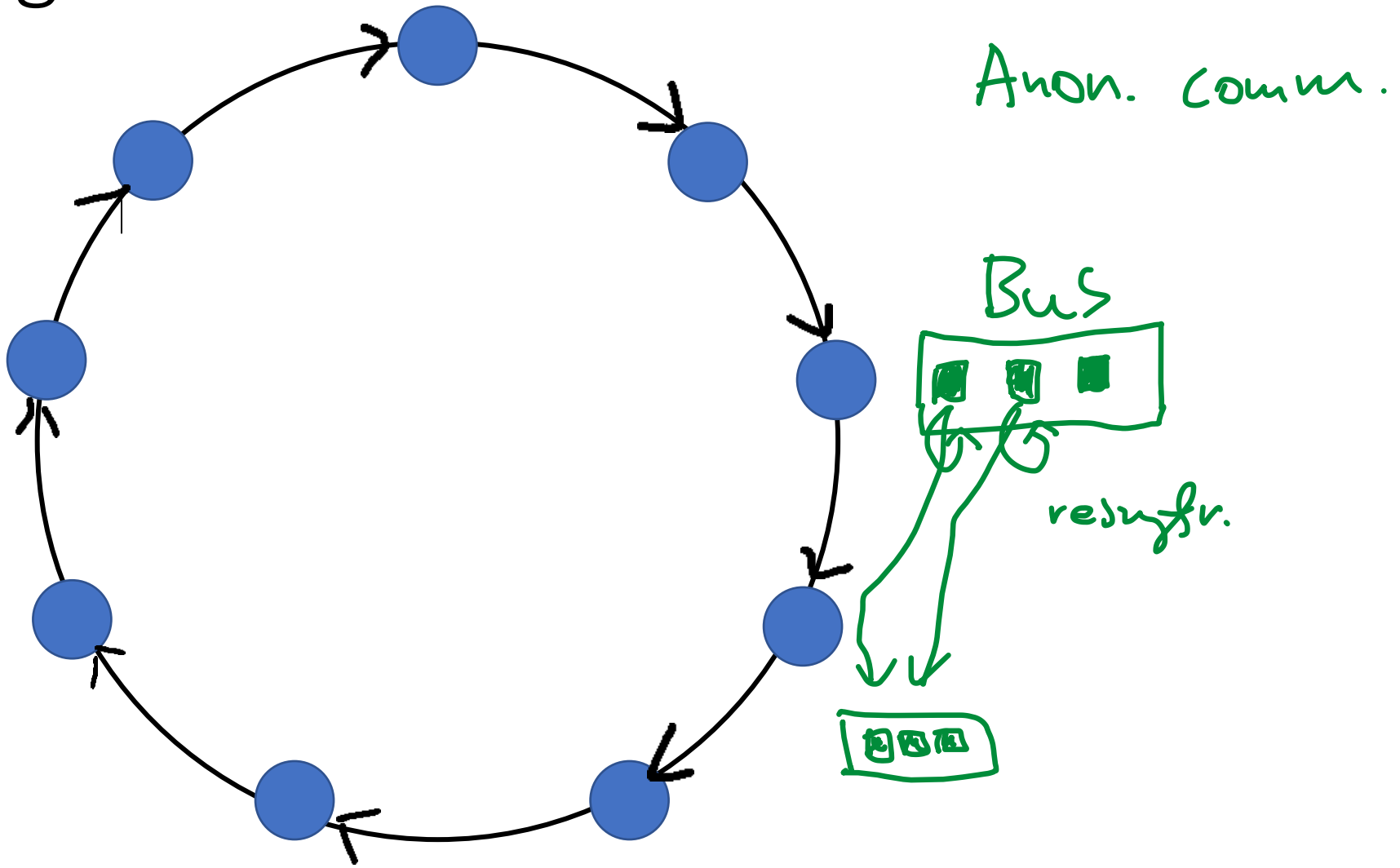
Example: Bubble sort



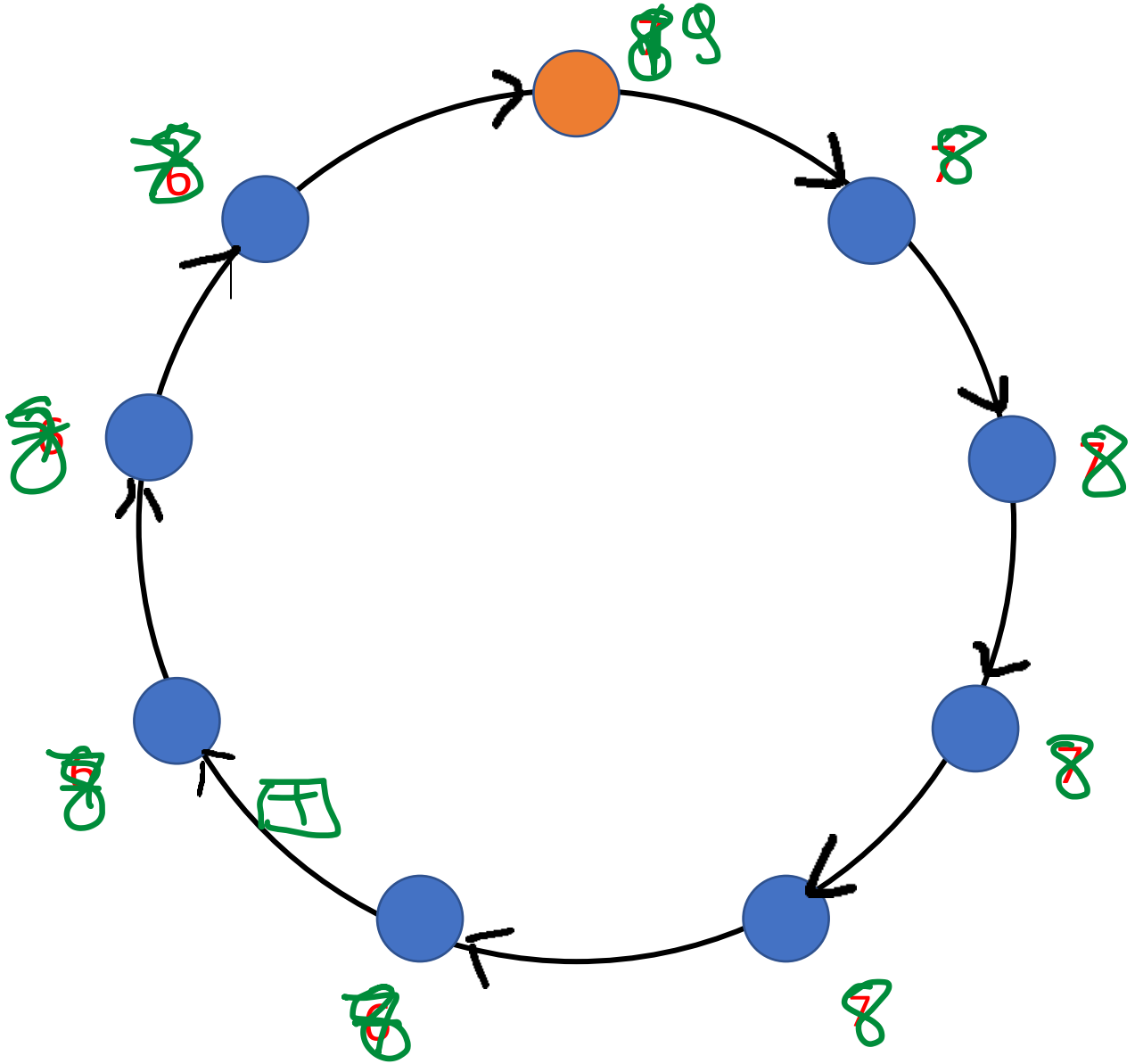
Quick sort



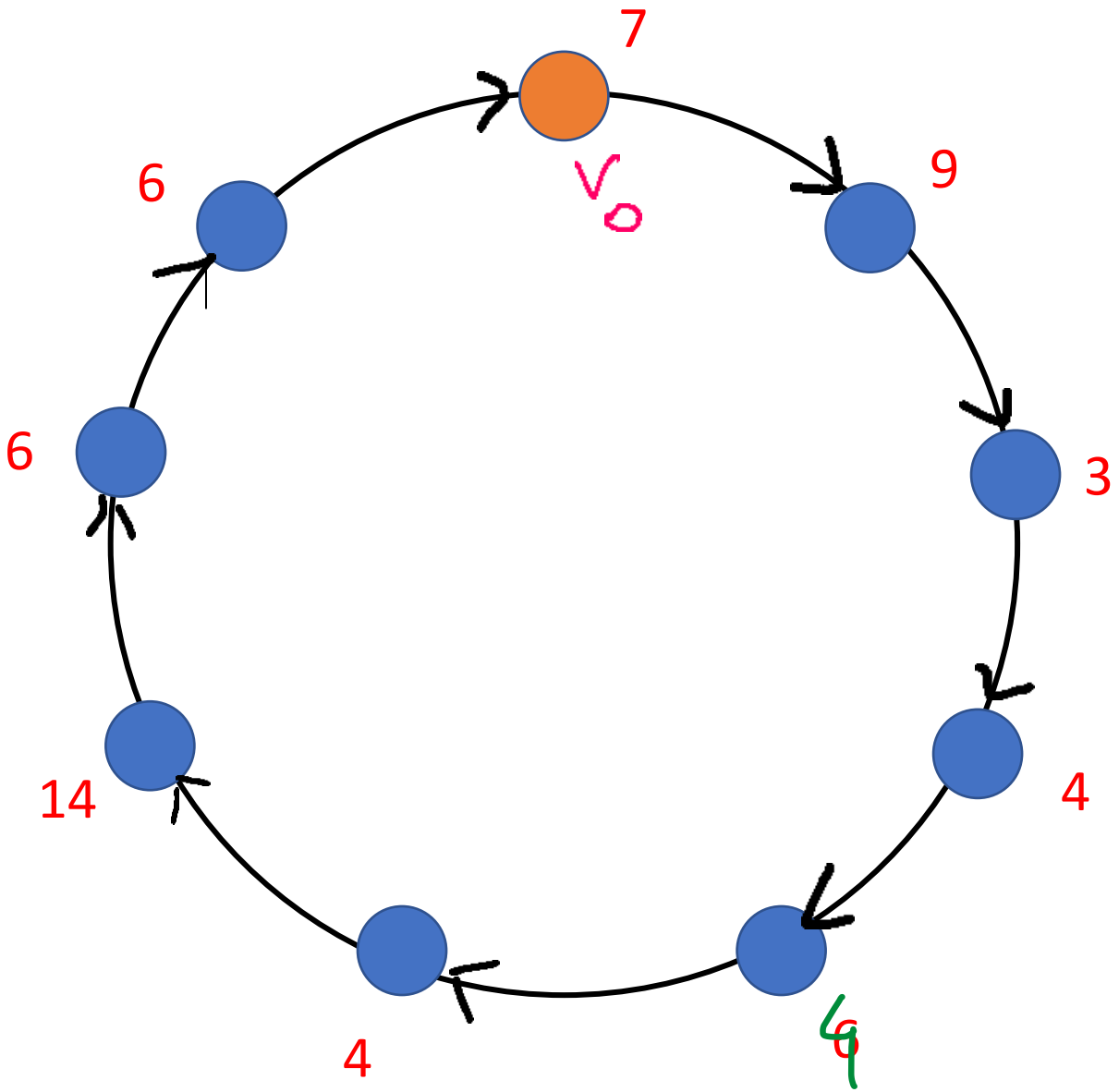
Token ring communication



Controlling token position



Unstable state – a mess..



Self stabilization

v_0 is the leader in the ring , *S(v)* is the state of the node *v*

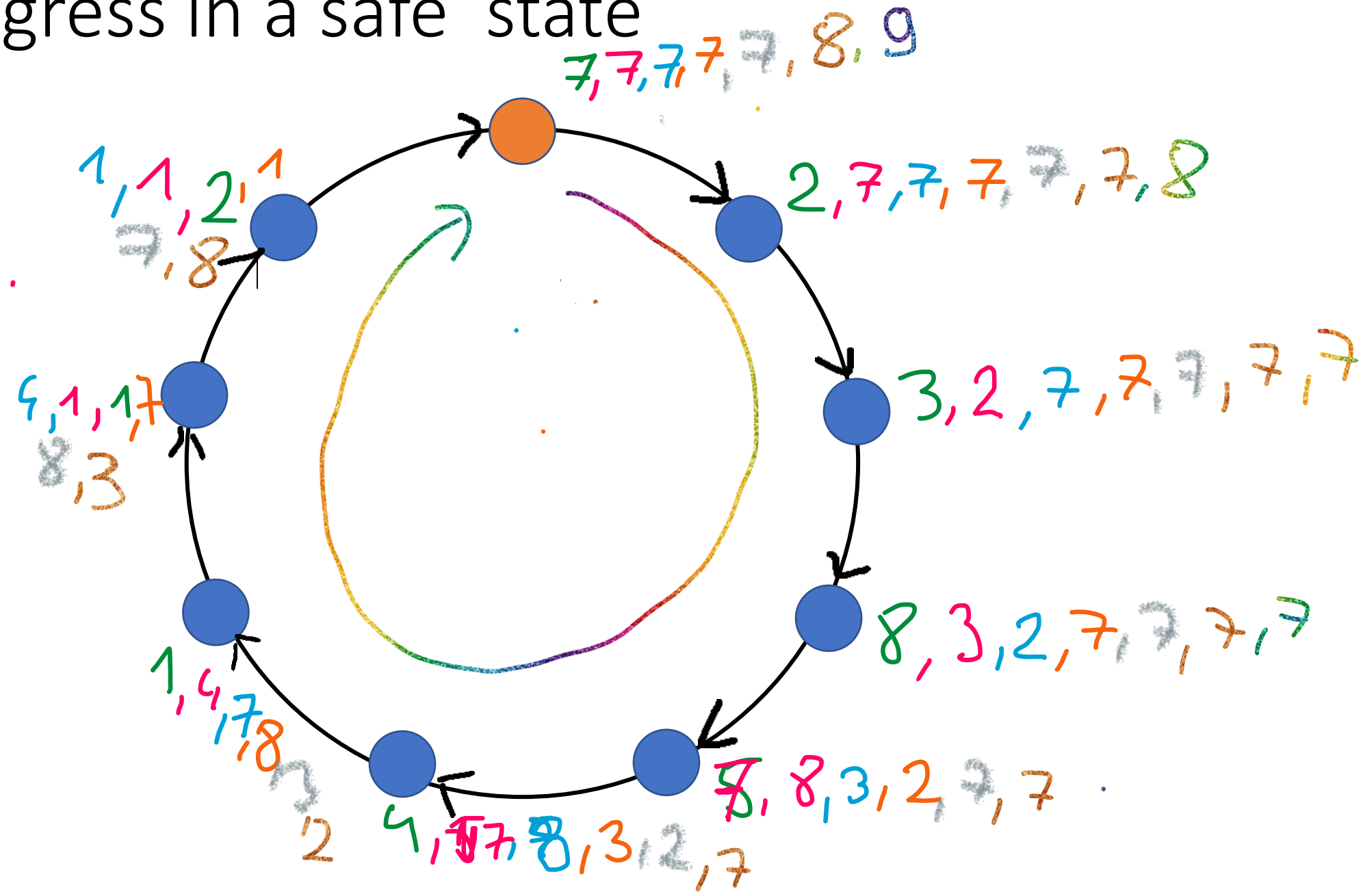
Algorithm 13.3 Self-stabilizing Token Ring

```
1: if v = v_0 then
2:     if S(v) = S(p) then
3:         S(v) := S(v) + 1 (mod n)
4:     end if
5: else
6:     S(v) := S(p)
7: end if
```

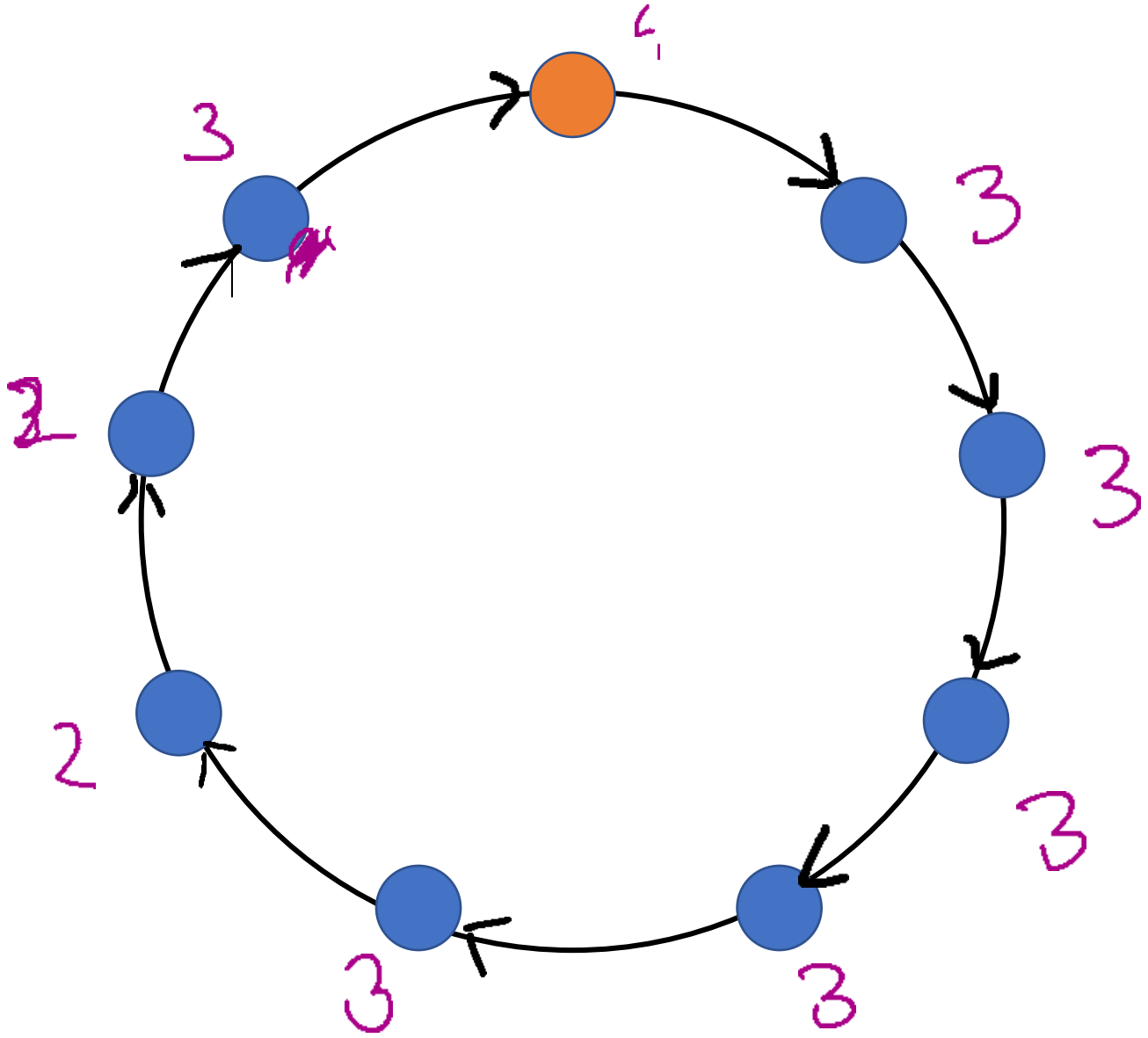
V – wierch.

P – jego poprzedni

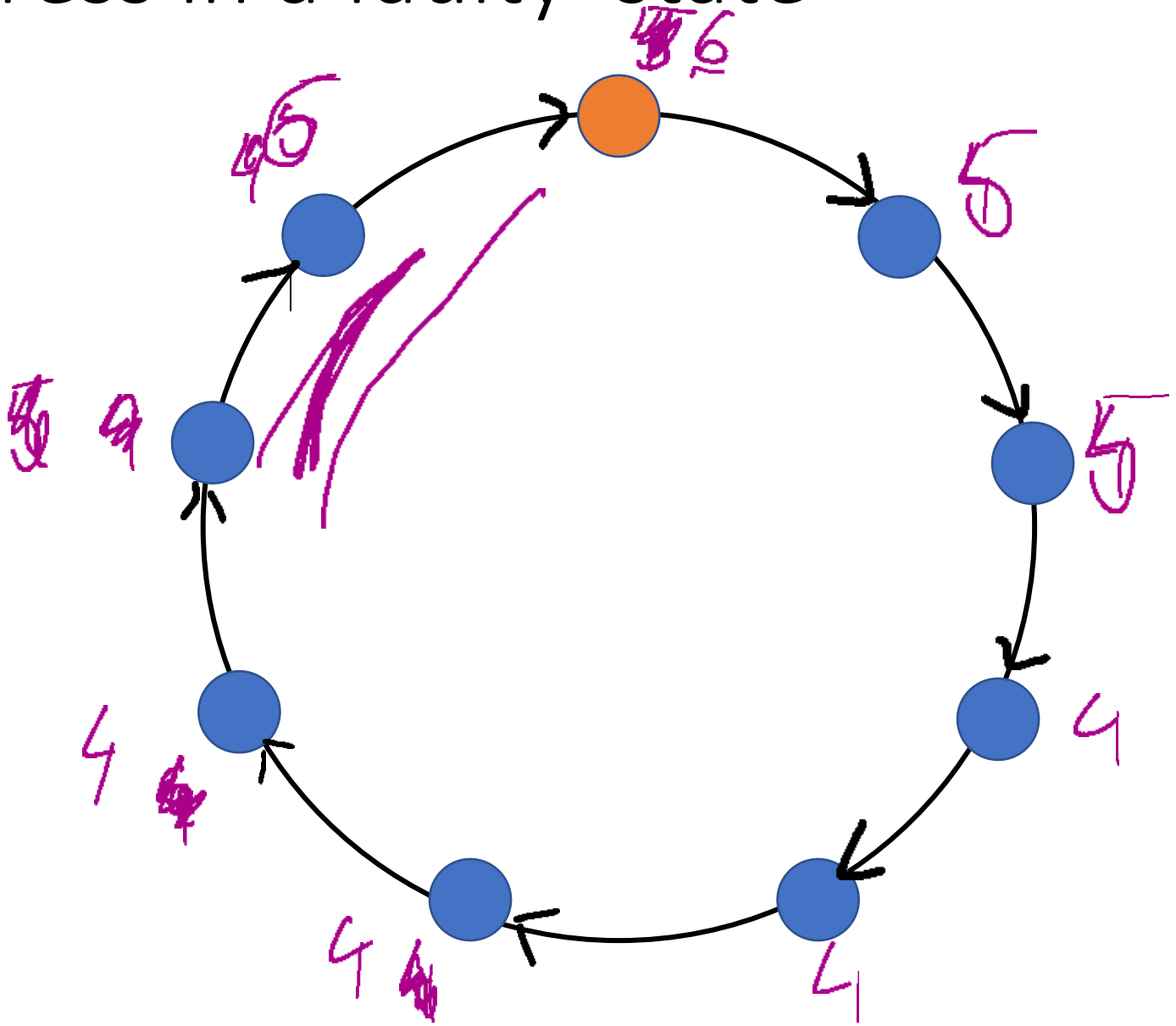
Progress in a safe state



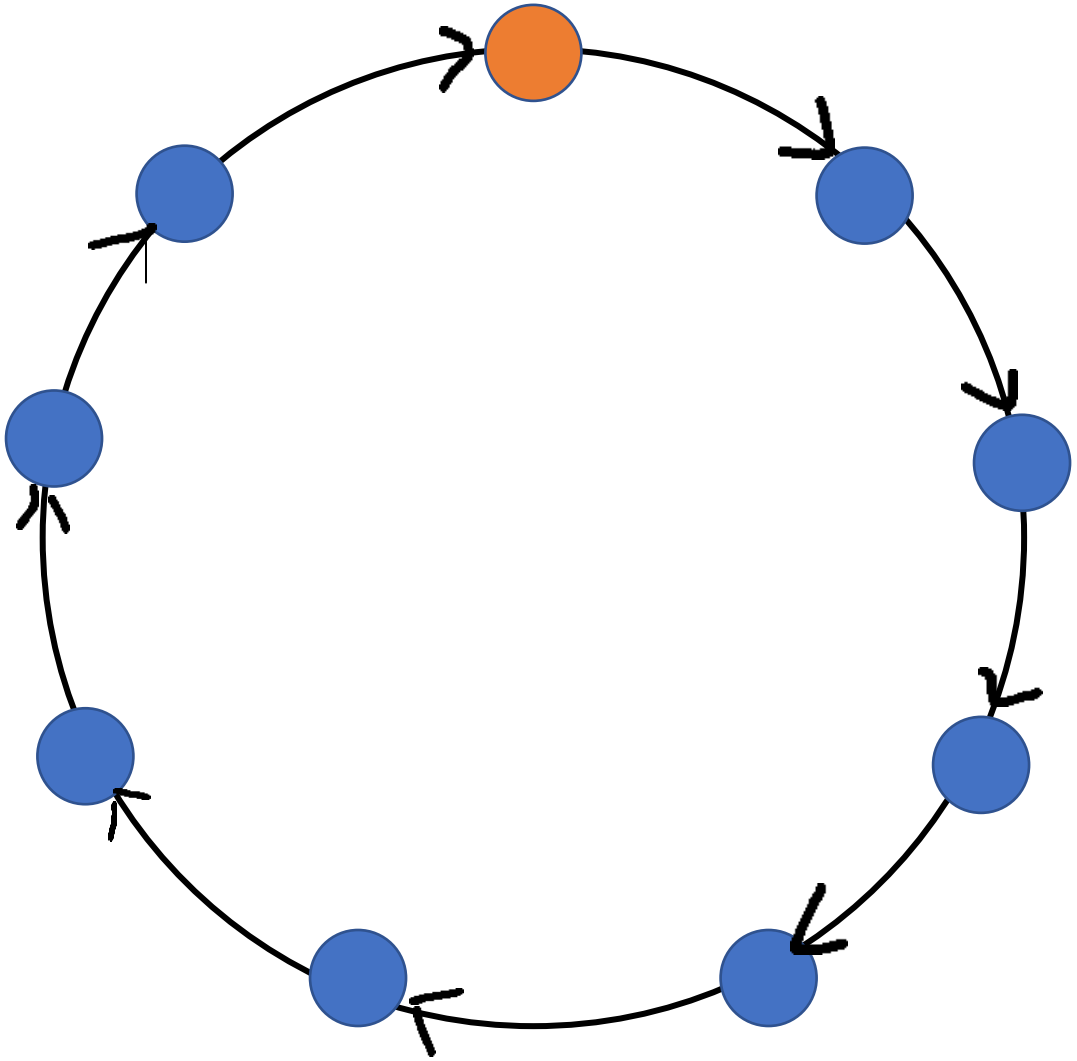
Progress in a faulty state



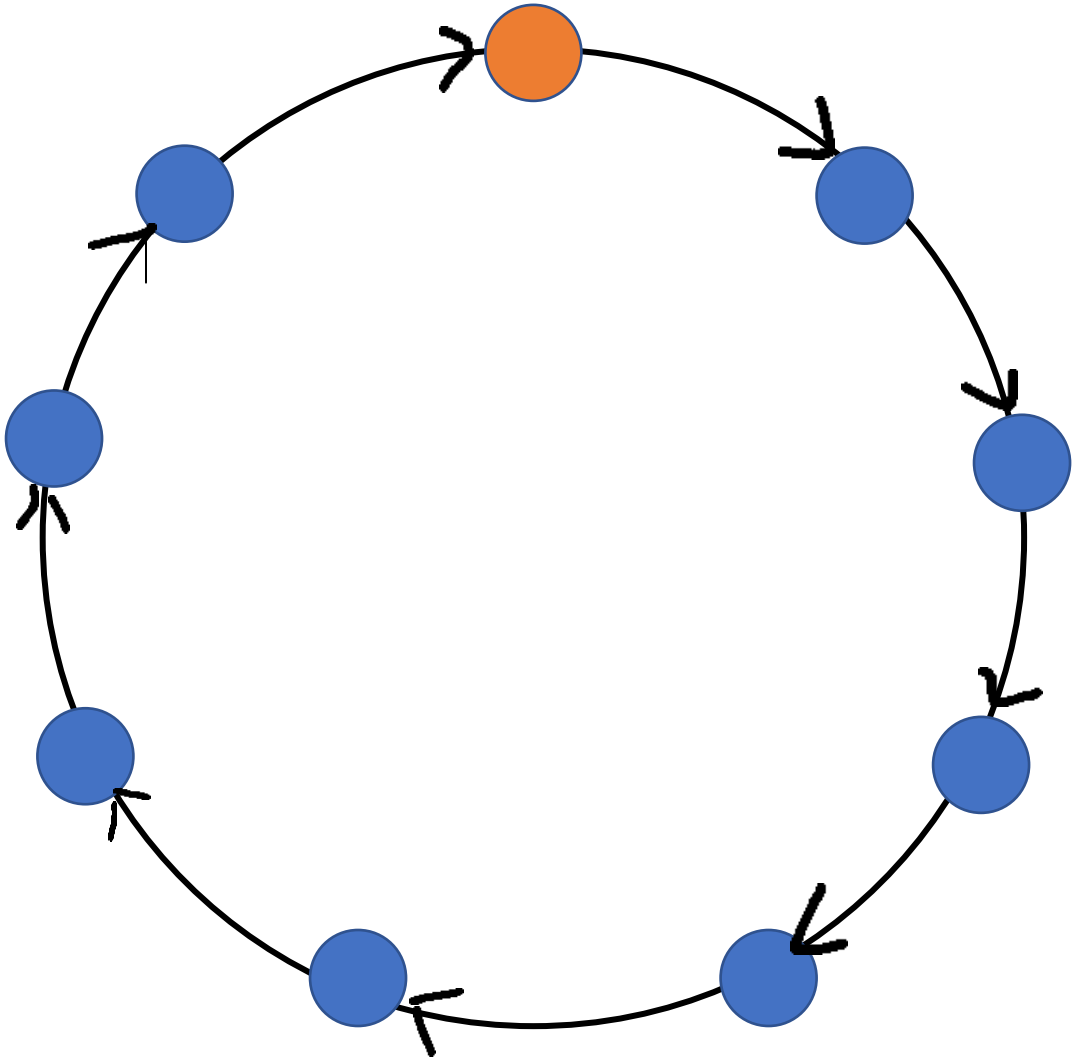
Progress in a faulty state



Progress in a faulty state



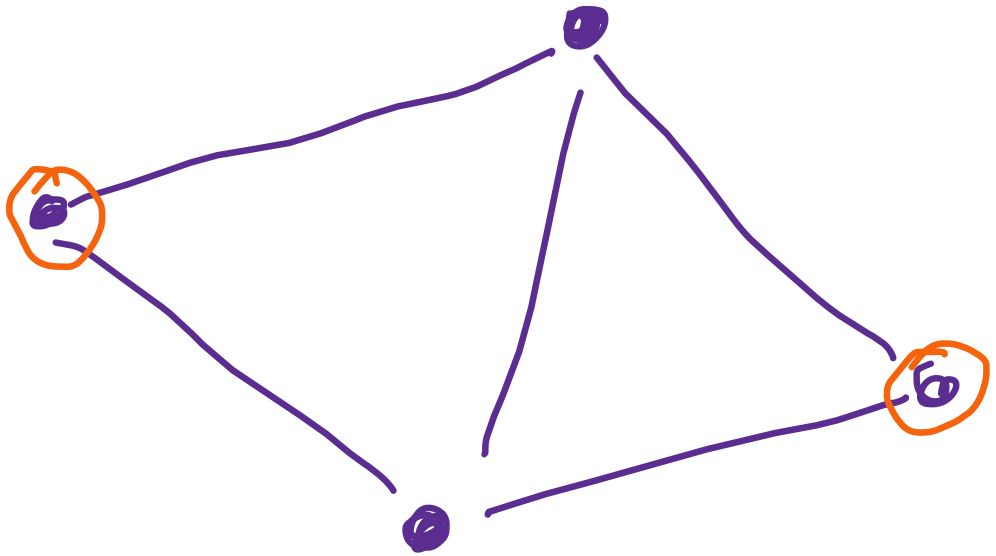
Progress in a faulty state



Maximal independent set (MIS)

Independent: no two nodes in the set are the neighbors

Maximal: no nodes can be added without violating the independence

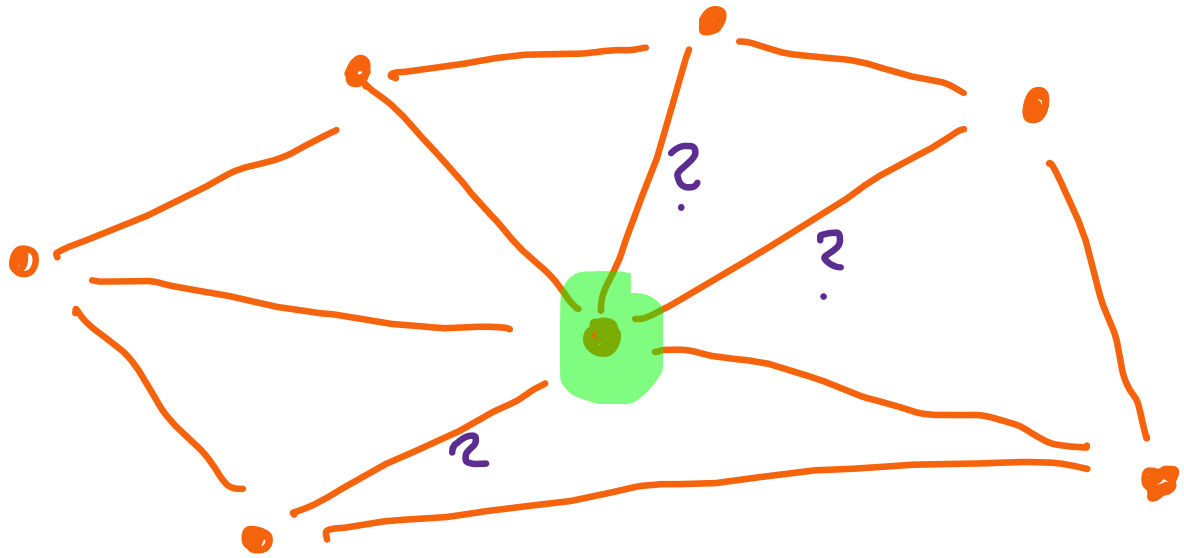


Algorithm 13.5 Self-stabilizing MIS

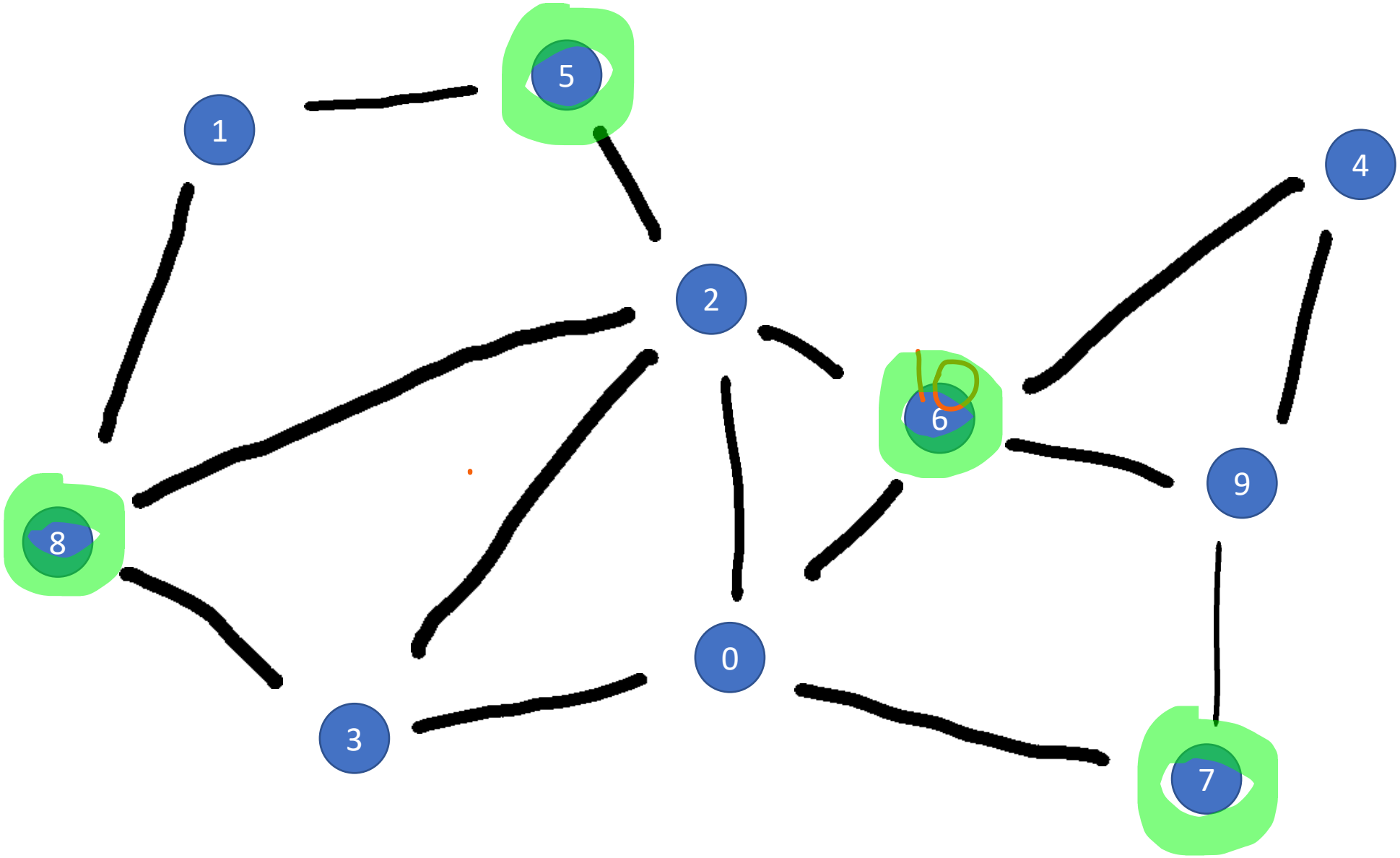
Require: Node IDs

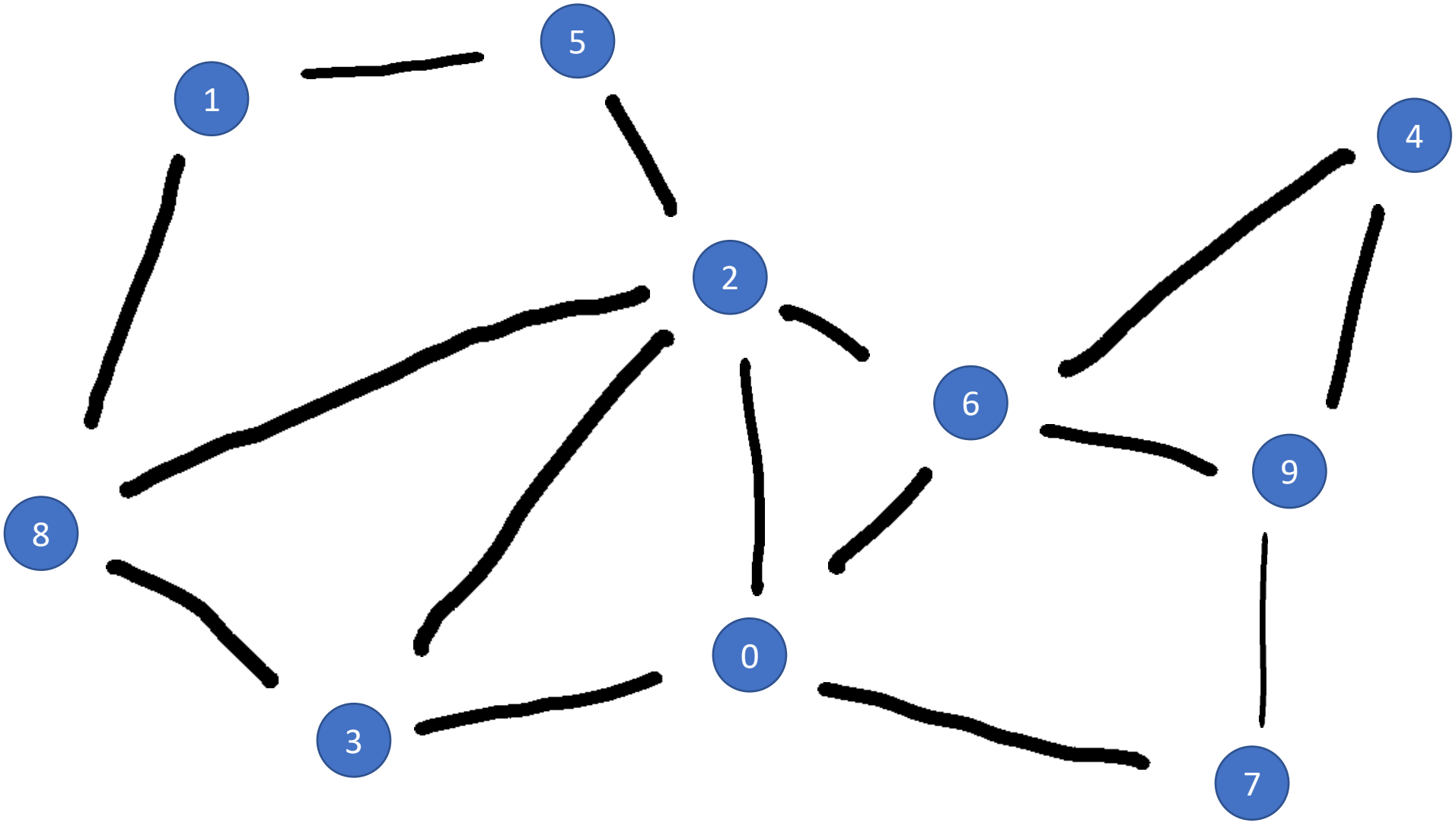
Every node v executes the following code:

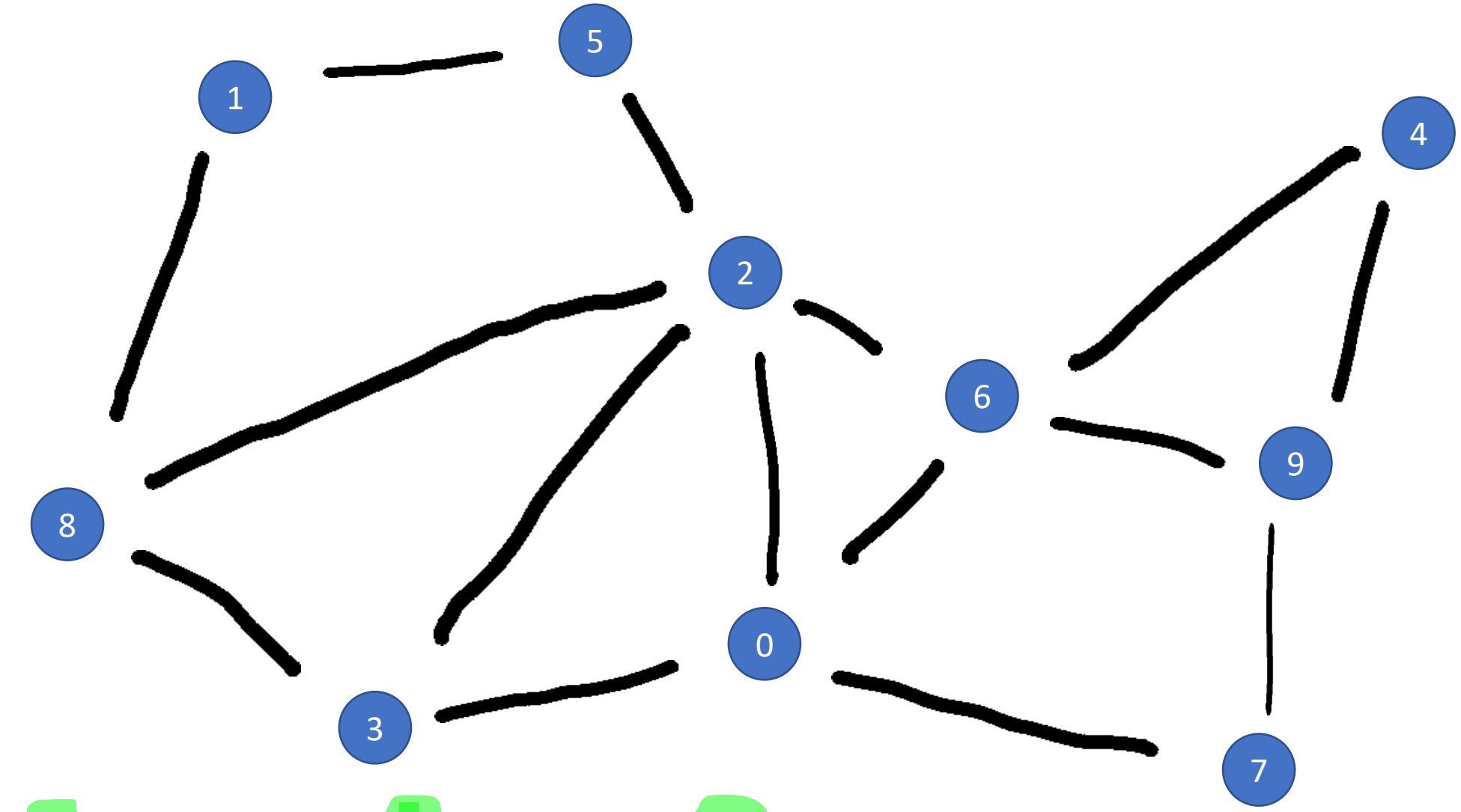
1: **do atomically**
2: Leave MIS if a neighbor with a larger ID is in the MIS
3: Join MIS if no neighbor with larger ID joins MIS
4: Send (node ID, MIS or not MIS) to all neighbors
5: **end do**

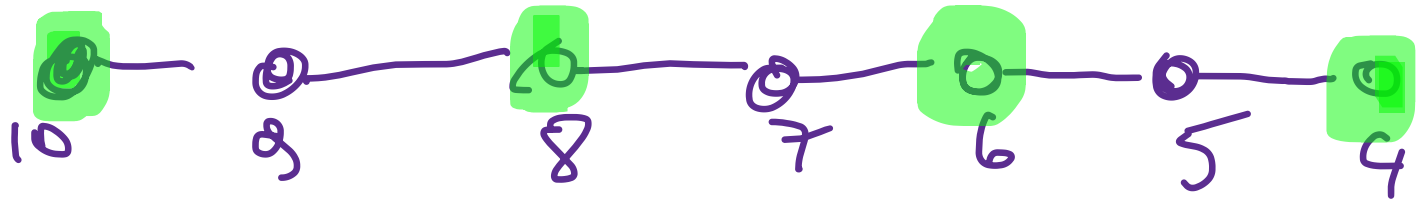


2 log n bitów w ID


5
1
4
2
6
9
8
0
3
7

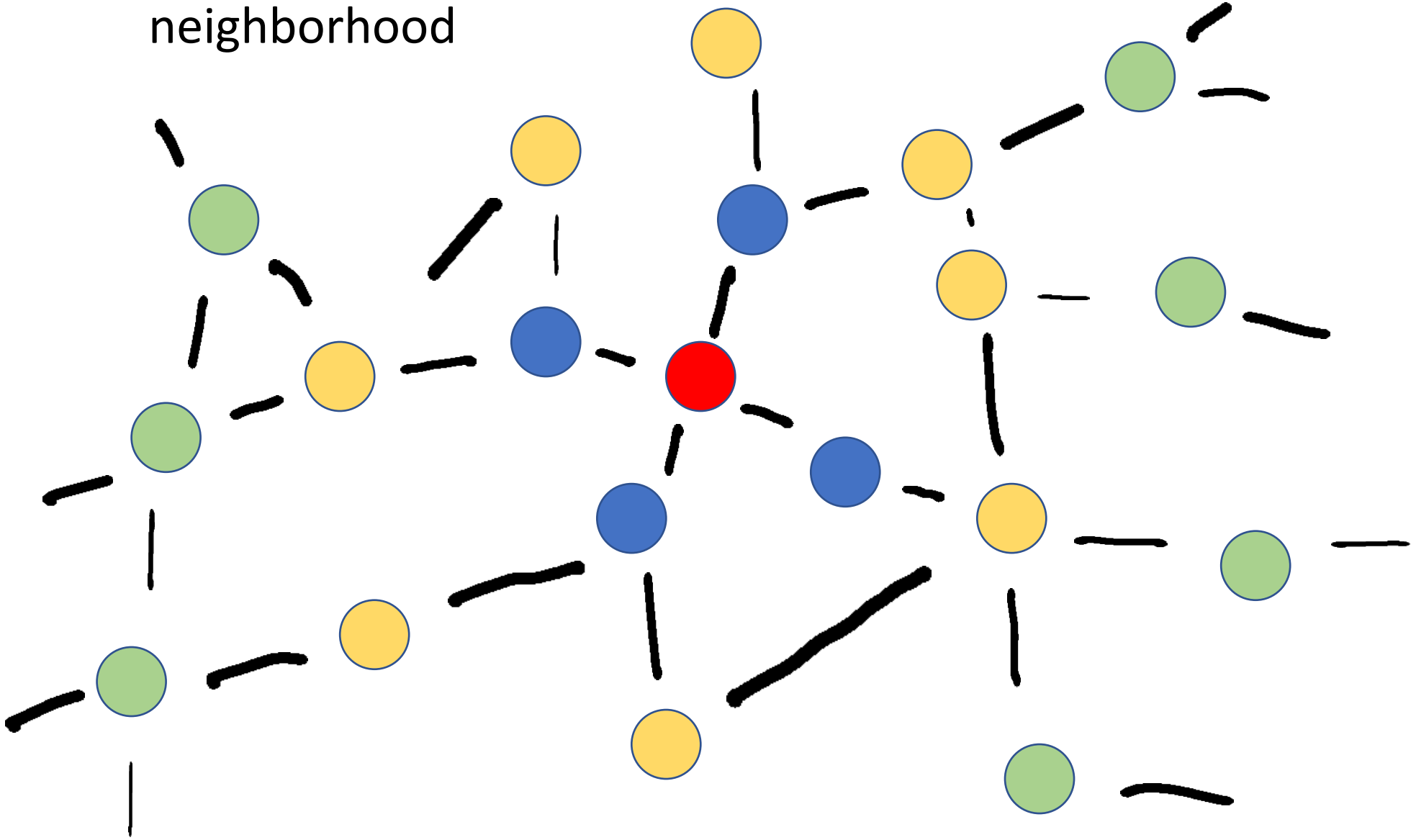

5
1
4
2
6
9
8
0
3
7

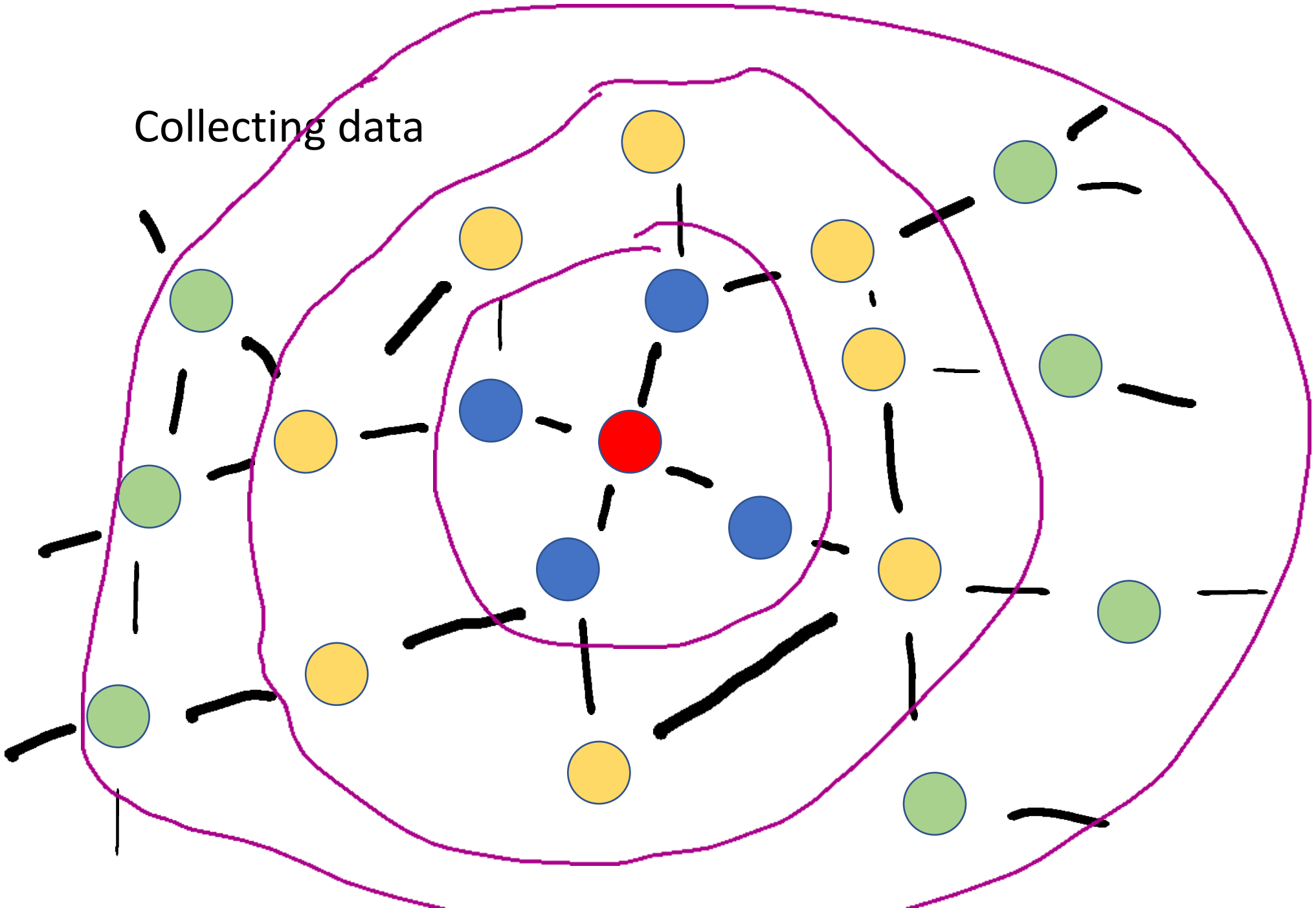

5
1
4
2
6
9
8
0
3
7


10
9
8
7
6
5
4

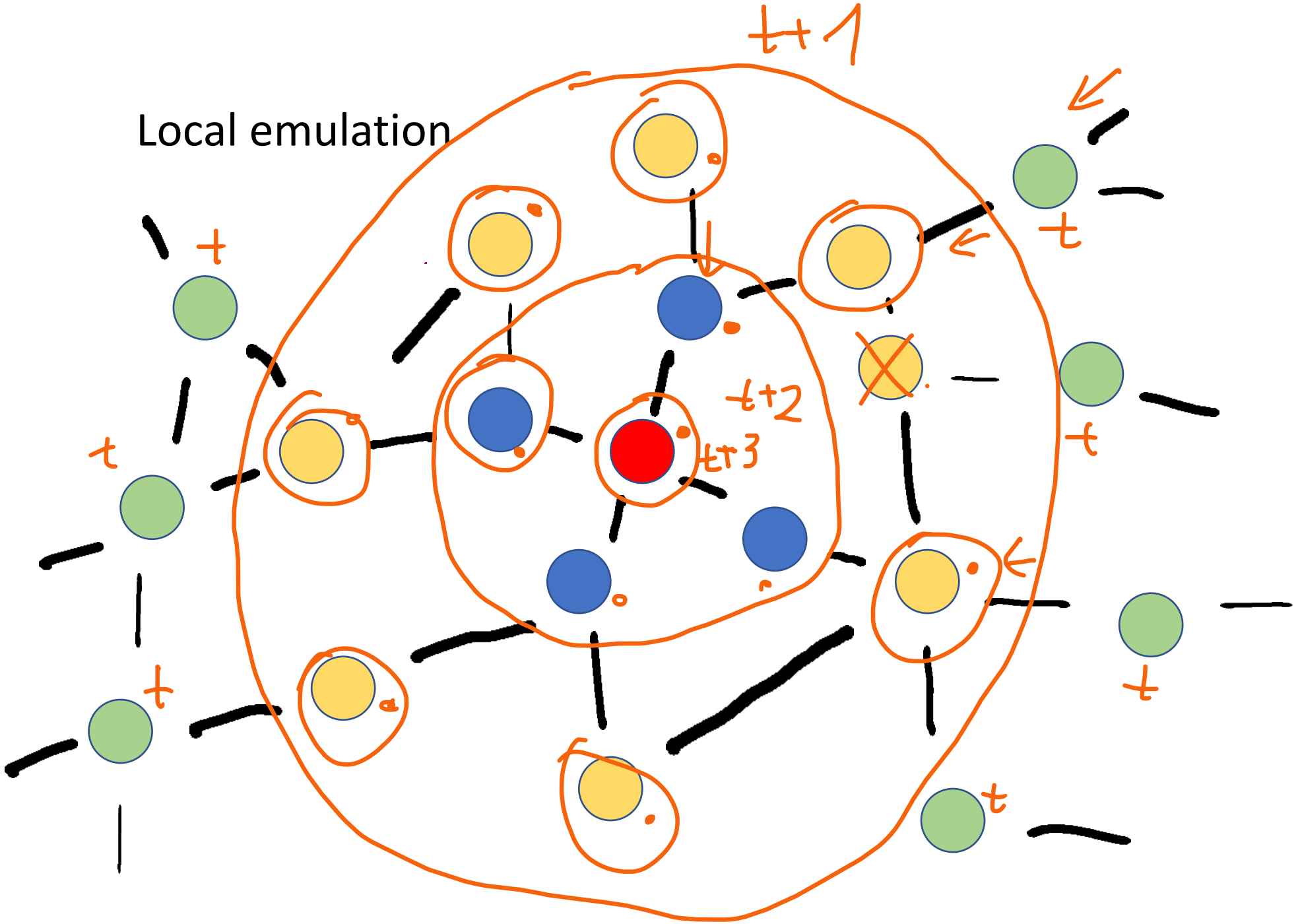
Transformation:
deterministic algorithm ⍰ self-stabilizing algorithm

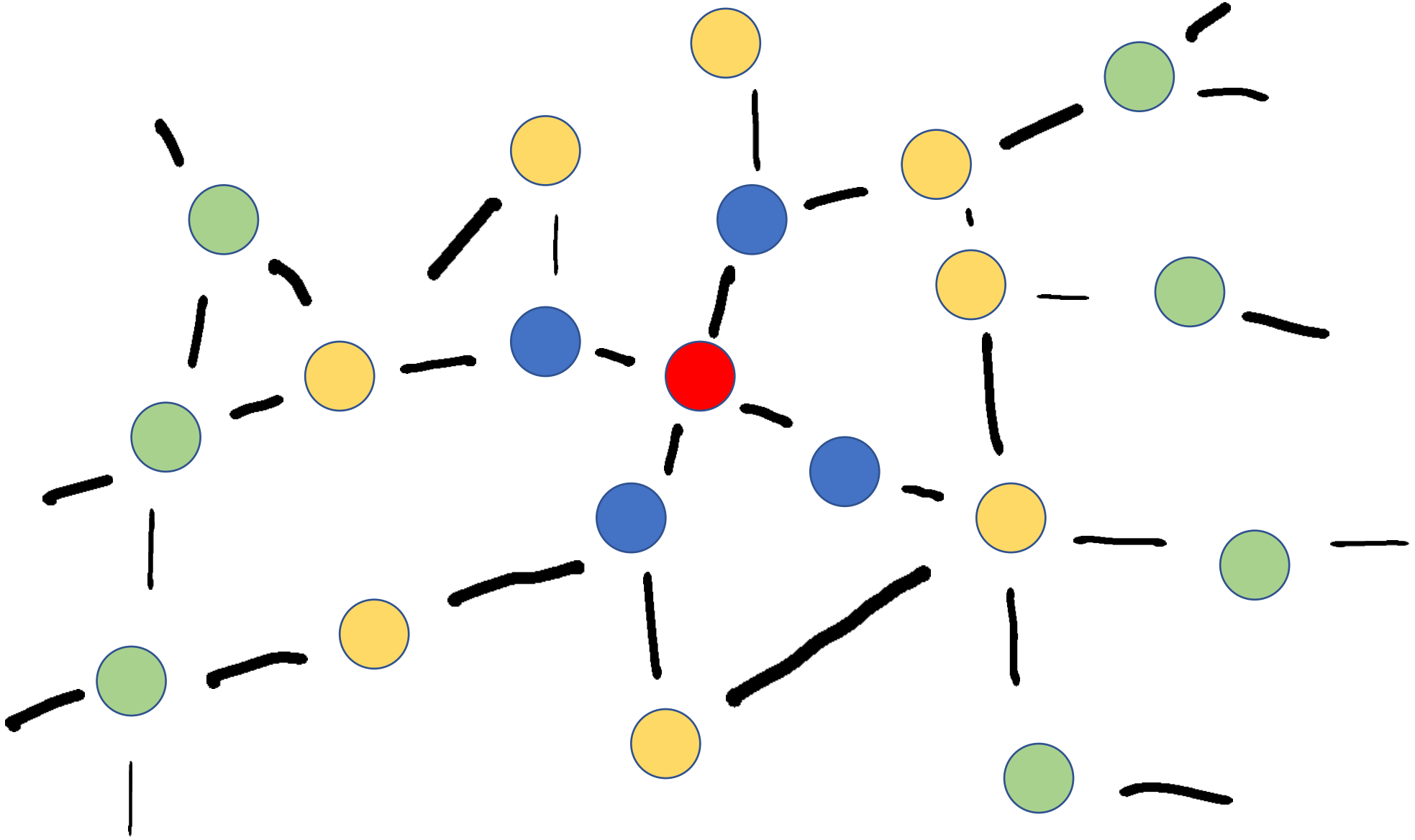
neighborhood




Collecting data

Local emulation





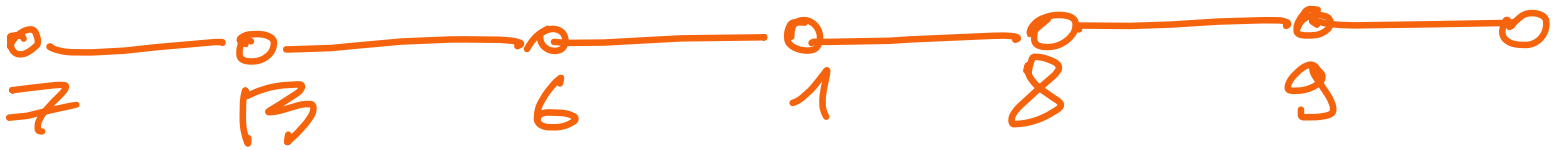
Transformation:
deterministic algorithm ⍰ self-stabilizing algorithm

- communication intensive
- many copies of the deterministic algorithm executed locally
- If deterministic stabilizes after T steps then the system self-heals in T steps after the error

Randomized case

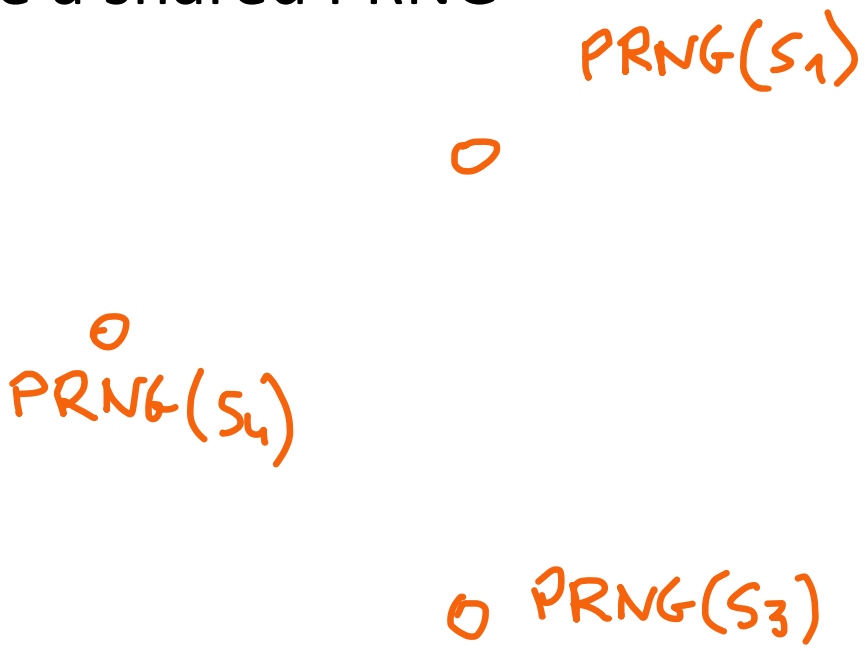
- faster
- robust

$\Rightarrow$ no emulation



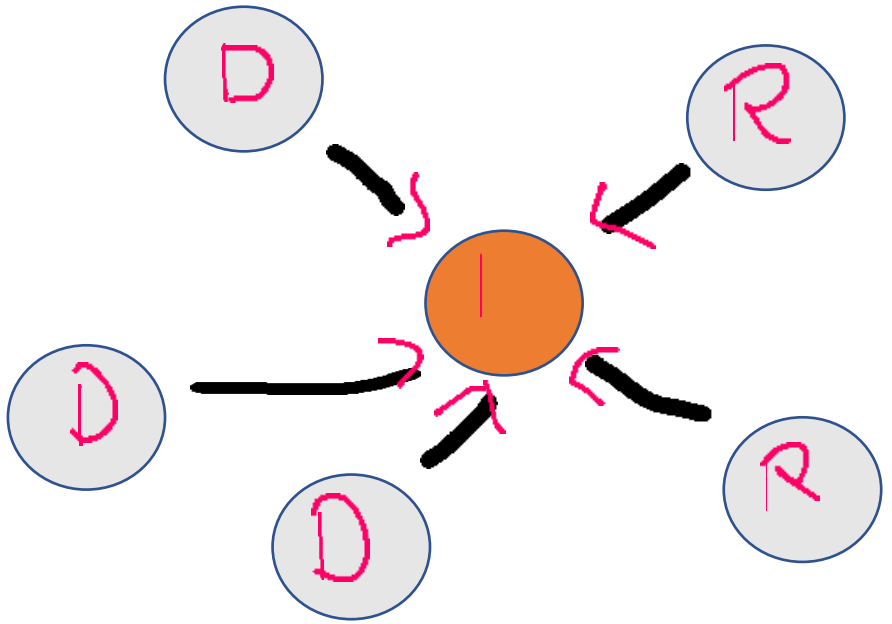
Randomized case

- One cannot emulate anymore
- Solution: use a shared PRNG



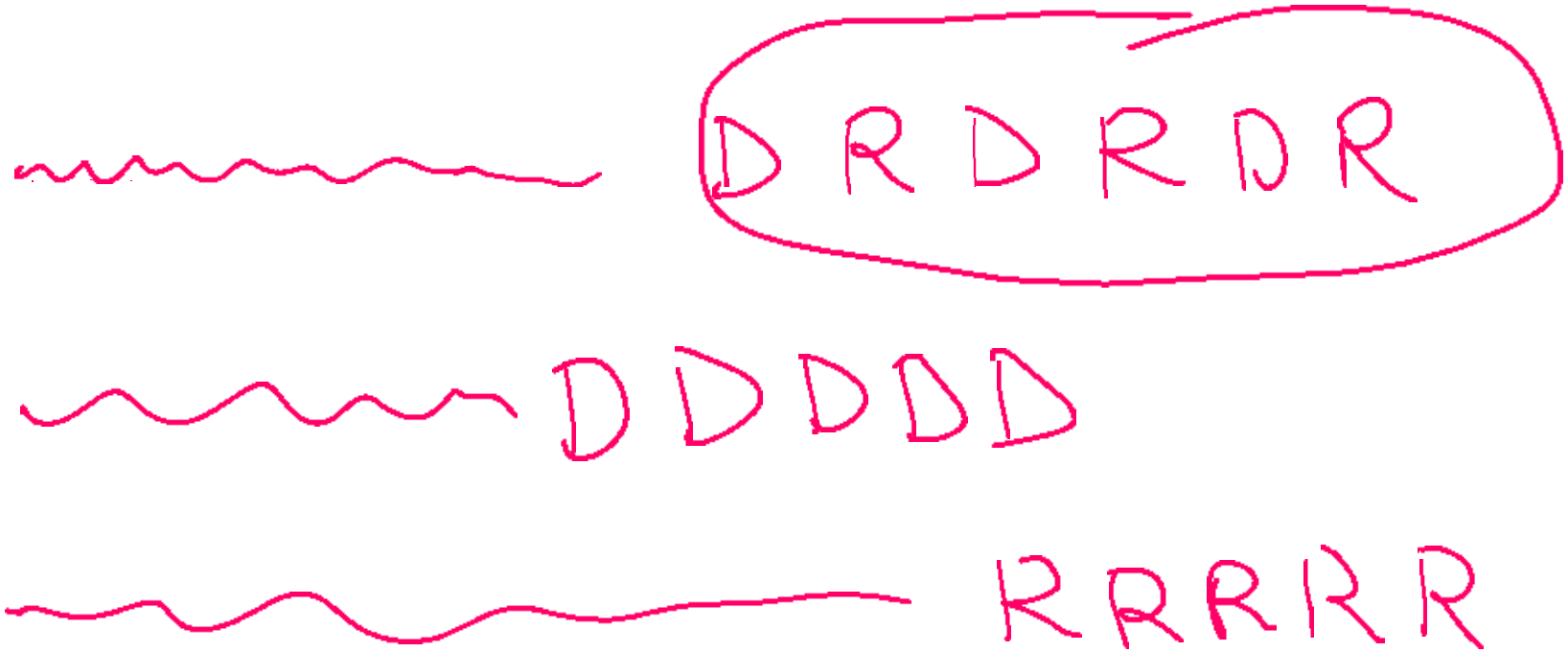
Advance example of self-stabilization (US elections)

Every evening:
-- a voter calls the friends
-- the friends give their recommendations
-- the voter changes preference according to majority of recommendations



- Is eventually everybody voting for the same party? No.
- Will each citizen eventually stay with the same party? No.
- Will citizens that stayed with the same party for some time, stay with that party forever? No.
- And if their friends also constantly root for the same party? No.
- Will this beast stabilize at all?!? Yes!

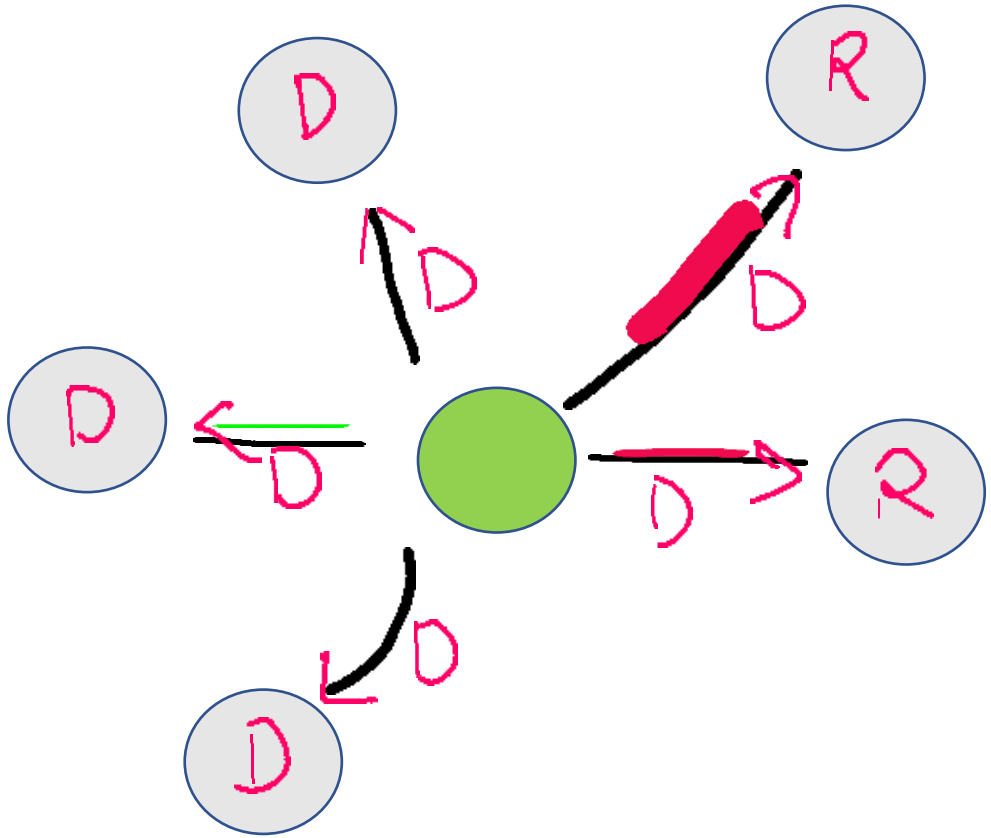
Theorem 13.7 (Dems & Reps). *Eventually every citizen is rooting for the same party every other day.*



Day t: supporter of Dems

Bad out-edges

Good out-edges



Day t:

b bad out-edges from step t

g good out-edges from step t

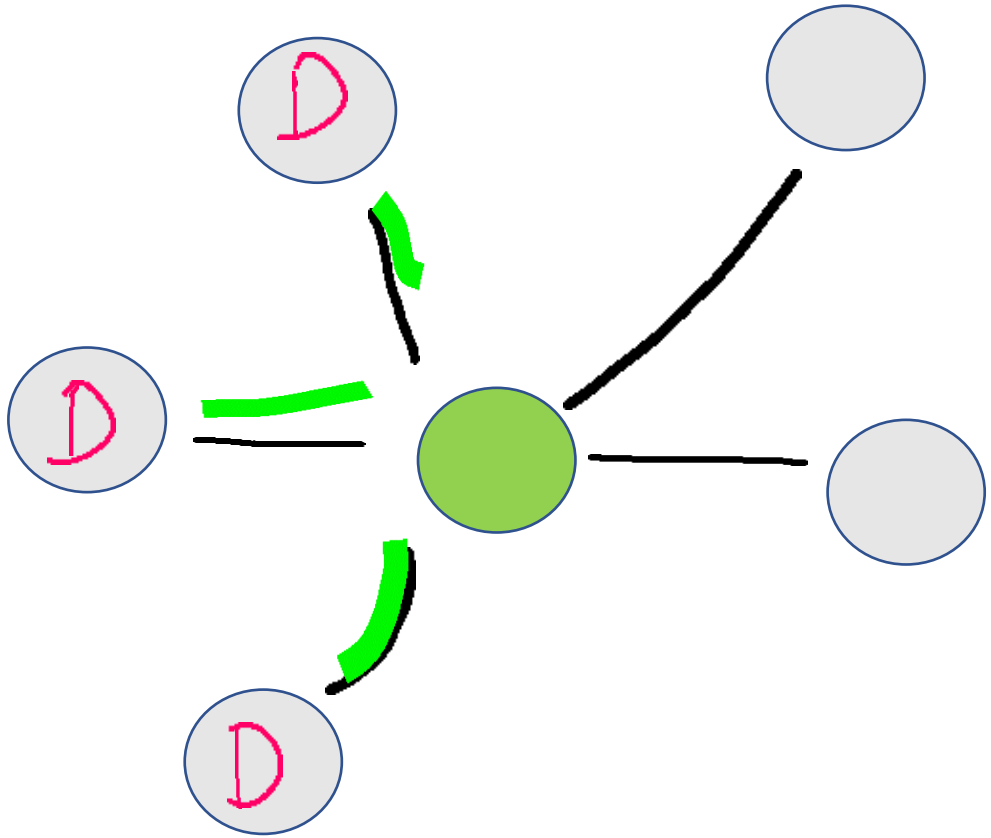
Day t+1:

b recommendations for Reps

g recommendations for Dems

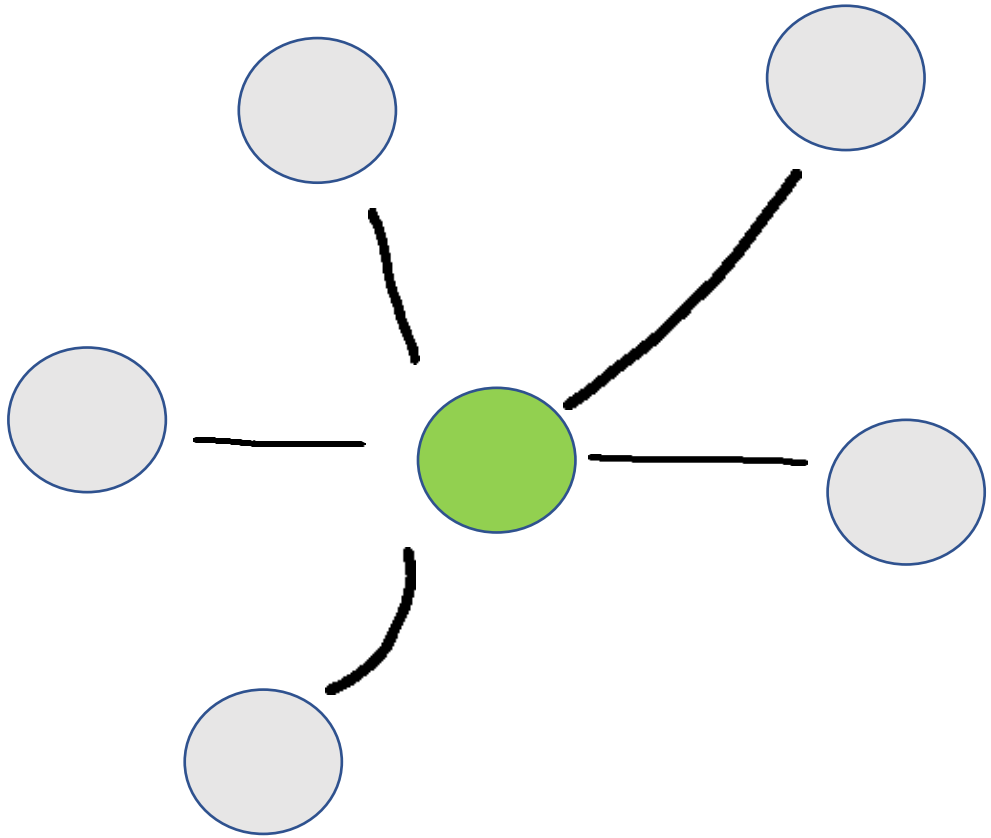
Case 1: g>b

voter roots Dems again



Case 1: b>g

voter roots Dems again

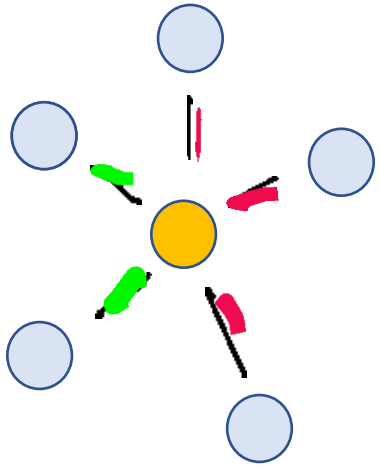


Case: g<b

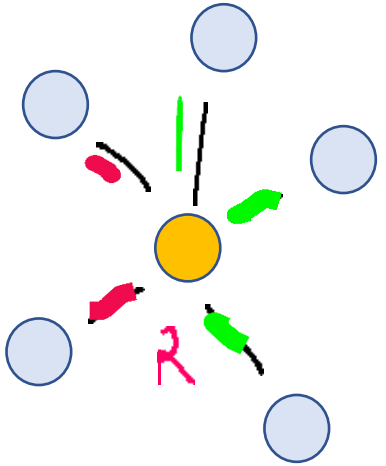
The voter roots REPs

bad in-edges at step t+1 = g
bad out-edges at step t = b

step t



step t+1



Total number of bad edges in the graph??

The total number of bad edges in the graph