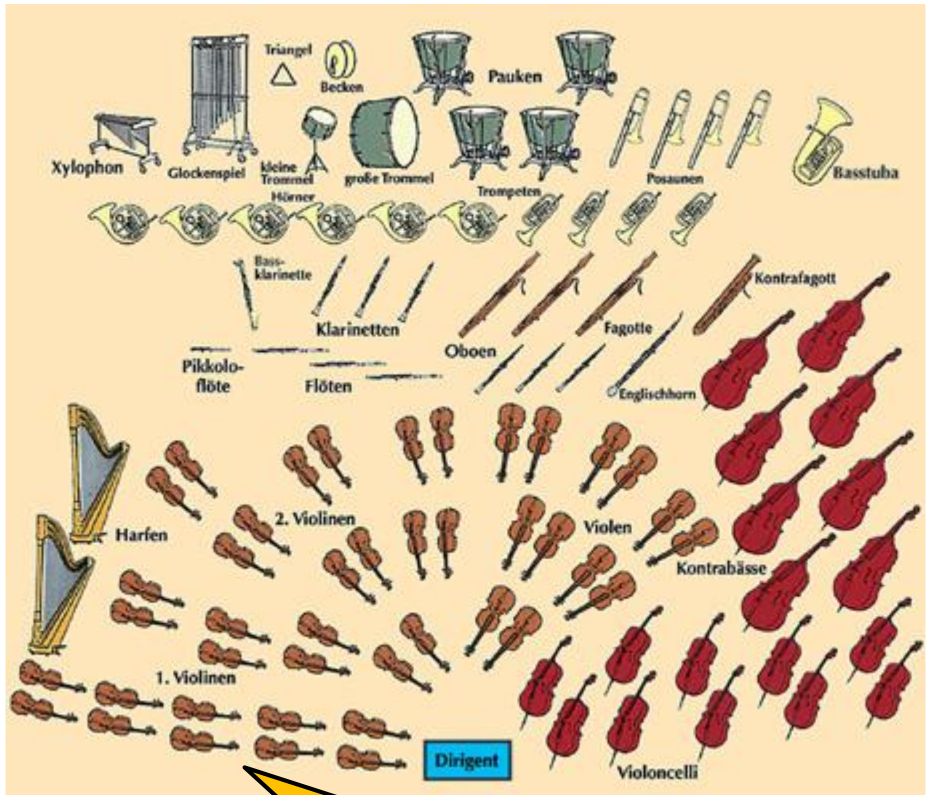
GIAN Course on Distributed Network Algorithms

Self-Stabilization

Self-Stabilization

We have seen: LOCAL algorithms can be run in asynchronous environments.
Today: LOCAL algorithms can also be made very robust, namely self-stabilizing!

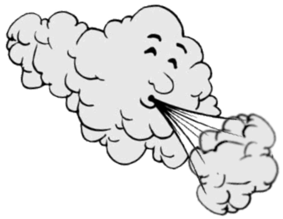
Example: A Fault-Tolerant Concert for the Mayor



Musicians are arranged in a graph.
Can see neighbors only!

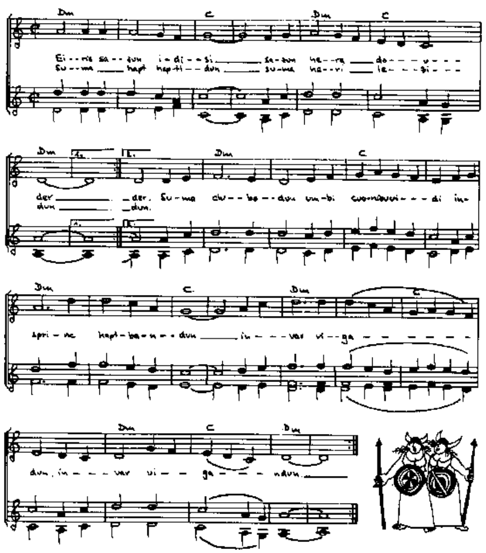
Example: A Fault-Tolerant Concert for the Mayor

Page 4



Page 22

Page 1



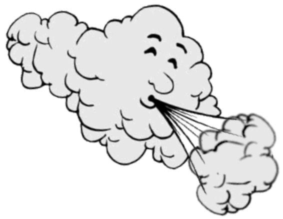
Setting:

- Play “happy birthday” again and again
- Wind changes pages
- Musicians can only observe immediate neigbors

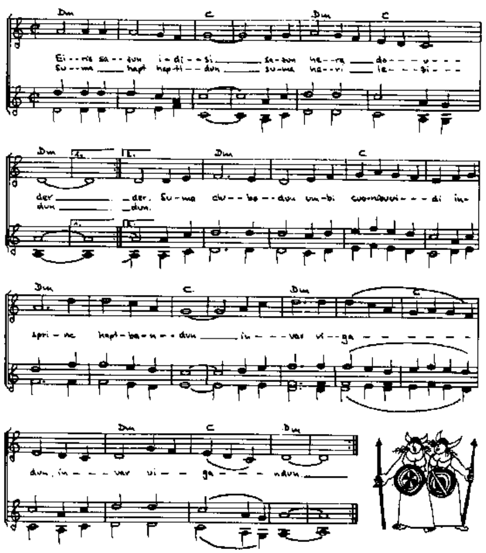
Goal:

- When wind stops, harmonize eventually!

Example: A Fault-Tolerant Concert for the Mayor



Page 4



Page 22

Page 1

Setting:

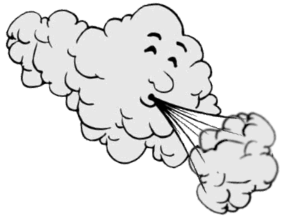
- Play “happy birthday” again and again
- Wind change
- Musicians ca

Algorithm to achieve this?

Goal:

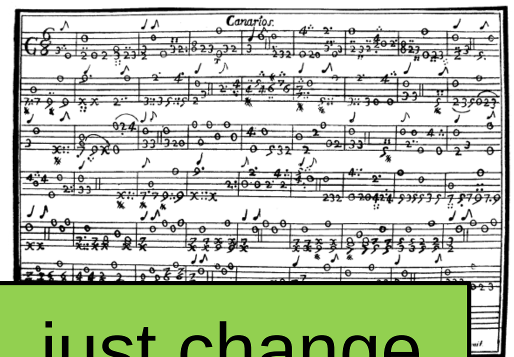
- When wind stops, harmonize eventually!

Example: A Fault-Tolerant Concert for the Mayor



Page 4

Page 22



Page 1

Idea 1: If out of sync, just change to the page of a nearby player!

Setting:

- Play “happy birthday” again and again
- Wind change
- Musicians ca

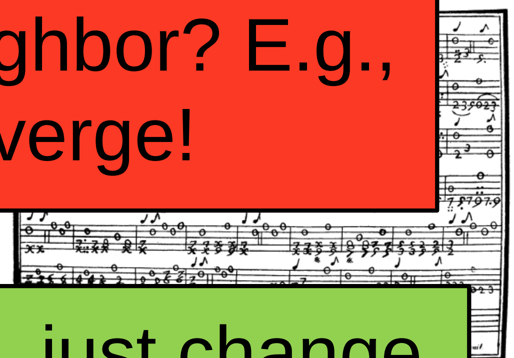
Algorithm to achieve this?

Goal:

- When wind stops, harmonize eventually!

Example: A Fault-Tolerant Concert for the Mayor

But what if the neighbour does the same with his neighbor? E.g., me?? May never converge!



Idea 1: If out of sync, just change to the page of a nearby player!

Page 1

Setting:

- Play "happy birthday" again and again
- Wind change
- Musicians ca

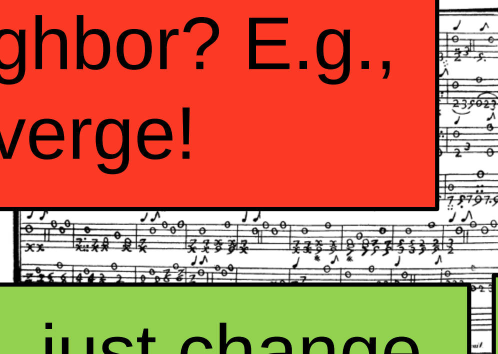
Algorithm to achieve this?

Goal:

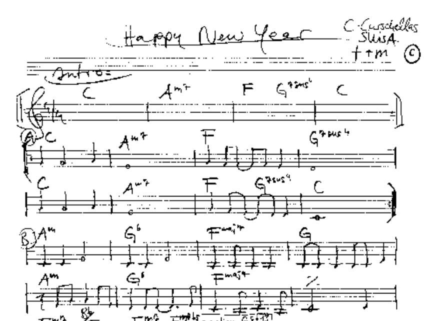
- When wind stops, harmonize eventually!

Example: A Fault-Tolerant Concert for the Mayor

But what if the neighbour does the same with his neighbor? E.g., me?? May never converge!



Page 1



Idea 1: If out of sync, just change to the page of a nearby player!

Idea 2: Go to start when asynchrony detected!

Setting:

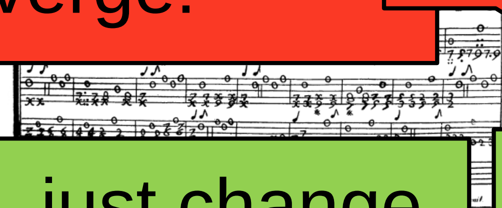
- Play “happy birthday” again and again
- Wind change
- Musicians ca

Algorithm to achieve this?

Goal:

- When wind stops, harmonize eventually!

Example: A Fault-Tolerant Concert for the Mayor



But what if the neighbour does the same with his neighbor? E.g. me?? May never converge!

But players further away detect it later and restart later! May never converge…

Idea 1: If out of sync, just change to the page of a nearby player!

Idea 2: Go to start when asynchrony detected!

Setting:

- Play "happy birthday" again and again
- Wind change
- Musicians ca

Algorithm to achieve this?

Goal:

- When wind stops, harmonize eventually!

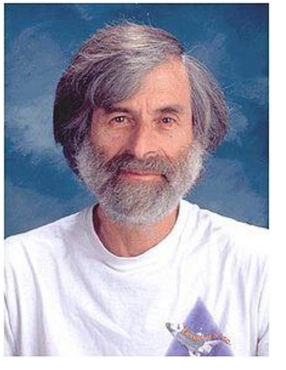
Self-Stabilization: A Powerful Concept in Fault-Tolerance

Self-Stabilization

Self-stabilizing algorithms pioneered by **Dijkstra** (1973): for example self-stabilizing mutual exclusion.

"I regard this as Dijkstra's most brilliant work. Self-stabilization is a very important concept in fault tolerance."

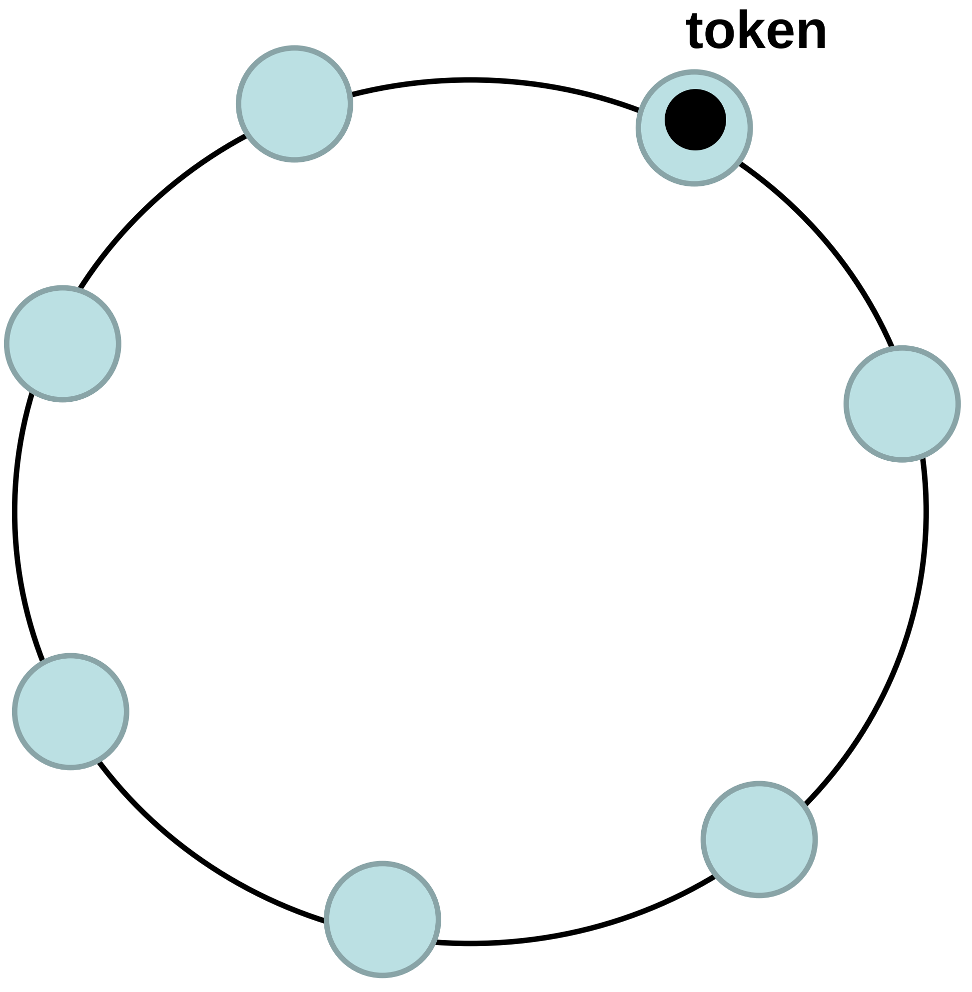
Leslie **Lamport** (PODC 1983)



Self-Stabilization: A Powerful Concept in Fault-Tolerance

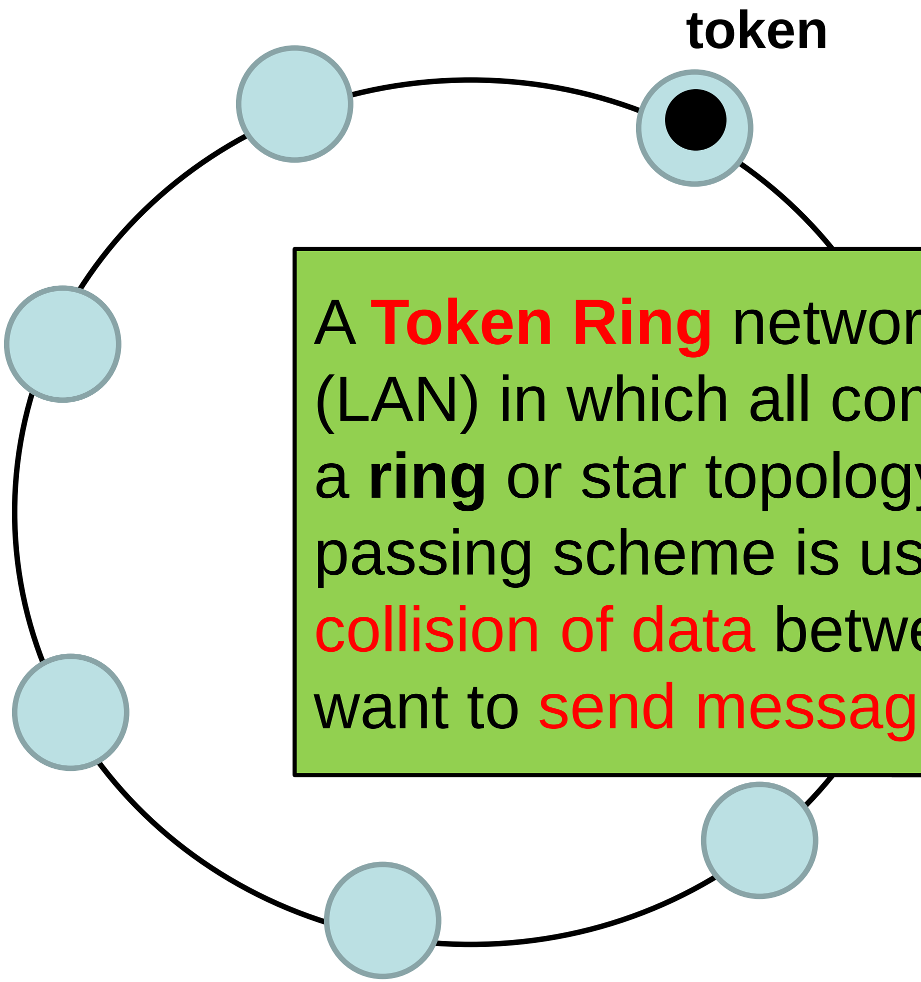
A distributed system is self-stabilizing if, starting from an arbitrary initial state, it is guaranteed to converge to a legitimate state. If the system is in a legitimate state, it is guaranteed to remain there, provided that no further faults happen. A state is legitimate if the state satisfies the specifications of the distributed system.

The Classic: Self-Stabilizing Token Ring



Heard of token-ring networks?

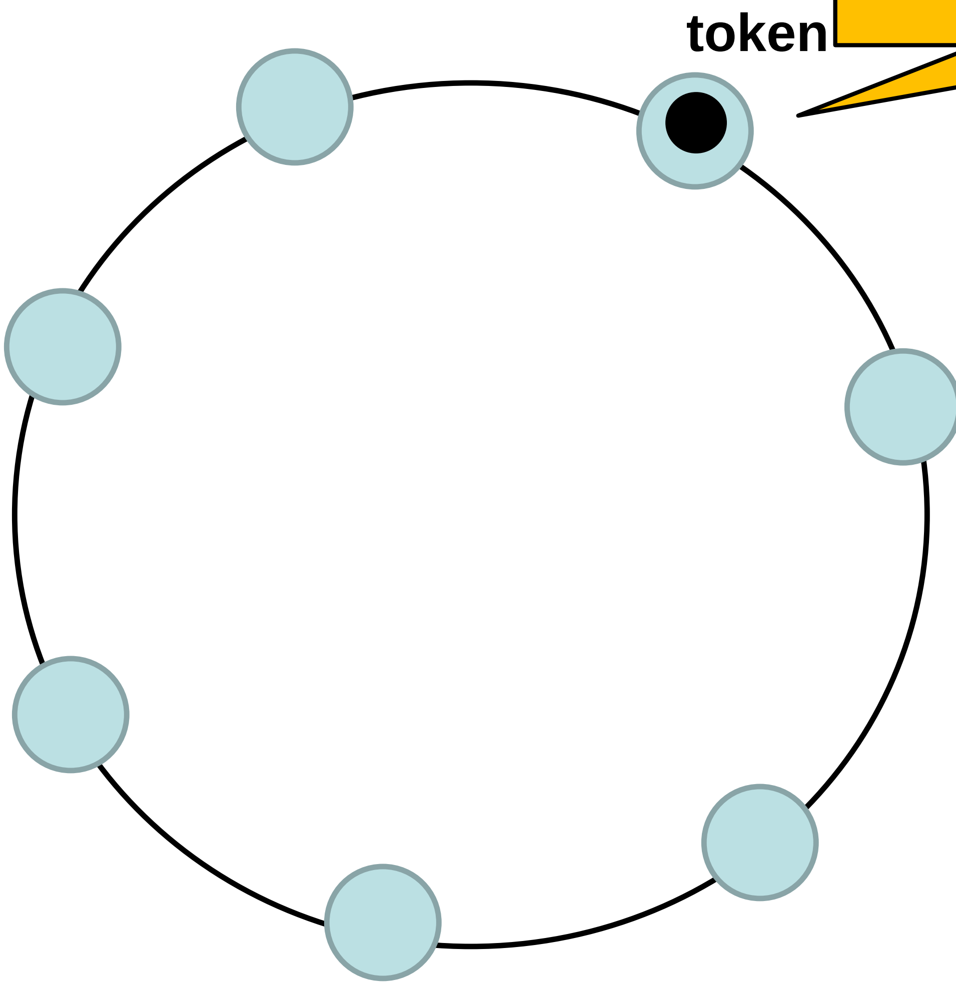
The Classic: Self-Stabilizing Token Ring



A **Token Ring** network is a local area network (LAN) in which all computers are connected in a **ring** or star topology and a bit- or **token**-passing scheme is used in order to prevent the collision of data between two computers that want to send messages at the same time.

The Classic: Self-Stabilizing To

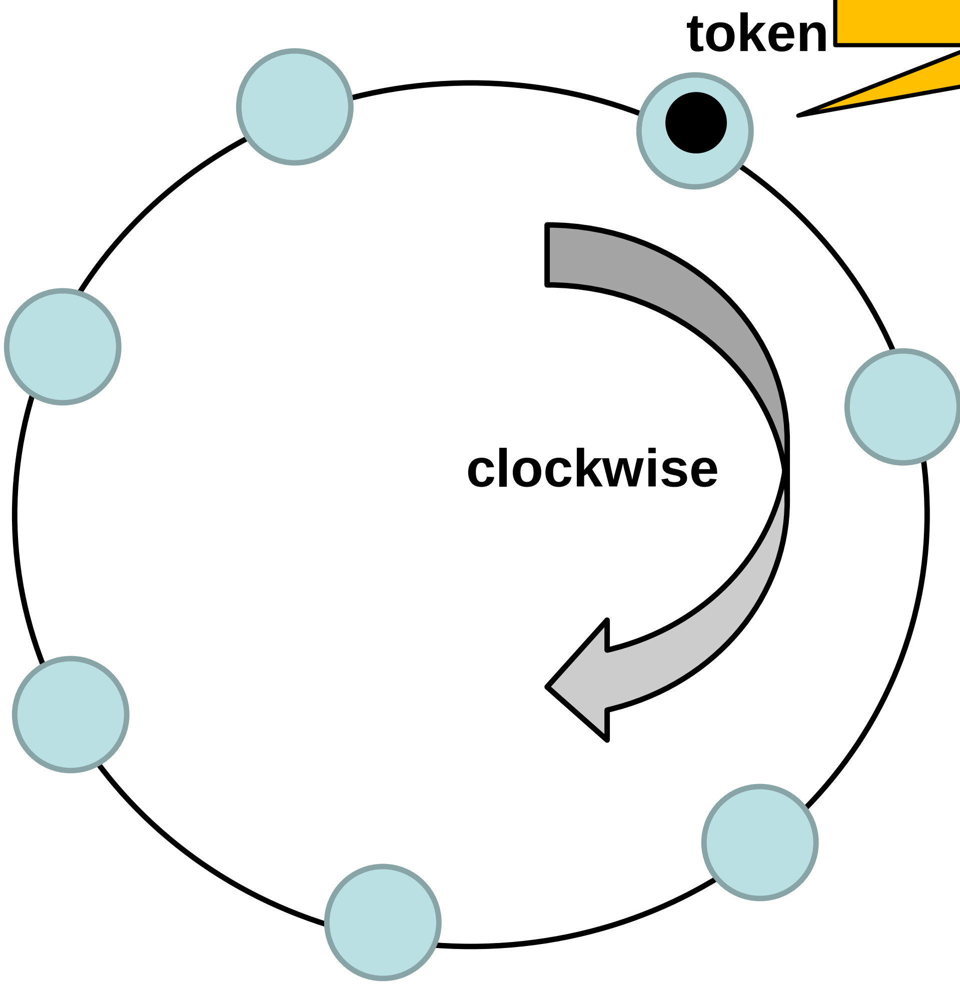
(Eventual) goal: A single token, circulating. E.g., mutual exclusion!



The Classic: Self-Stabilizing To

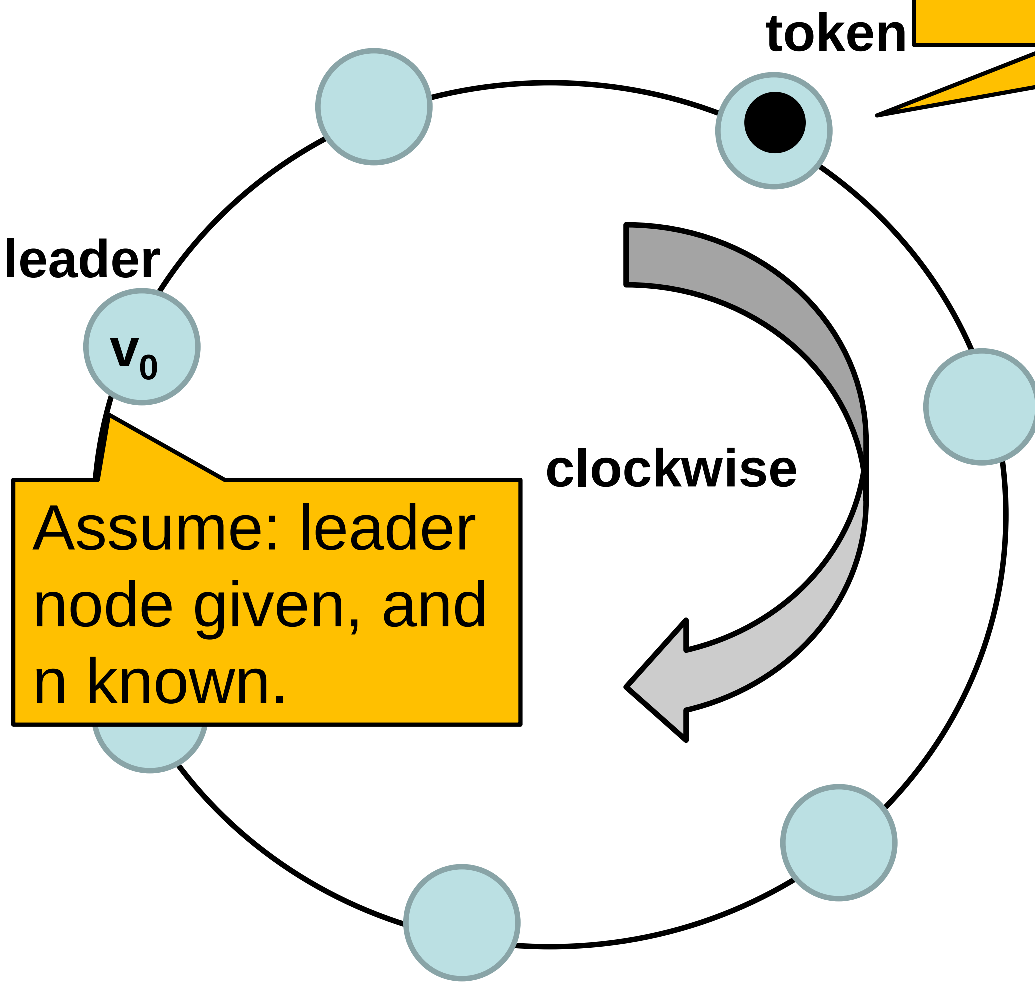
(Eventual) goal: A single token, circulating. E.g., mutual exclusion!

Assume: ring orientation is given.



The Classic: Self-Stabilizing To

(Eventual) goal: A single token, circulating. E.g., mutual exclusion!

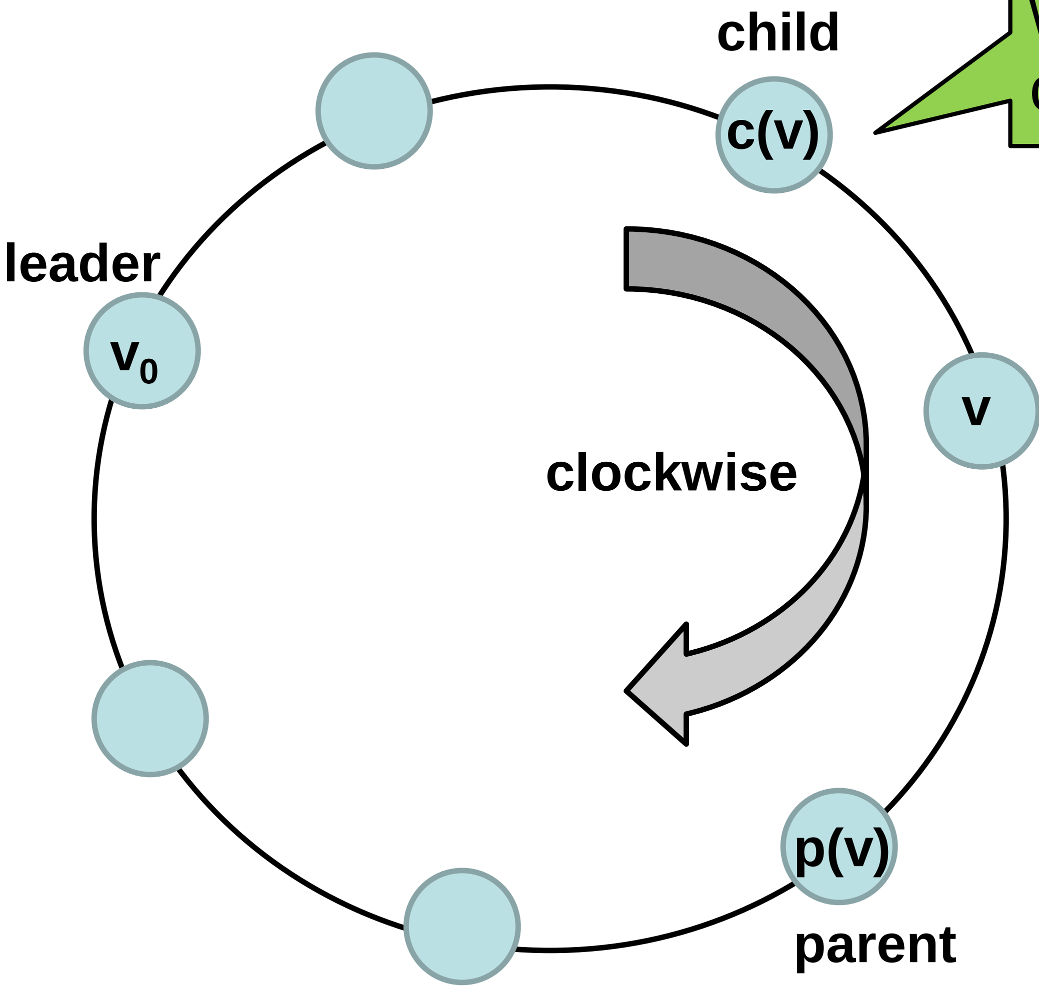


Assume: ring orientation is given.

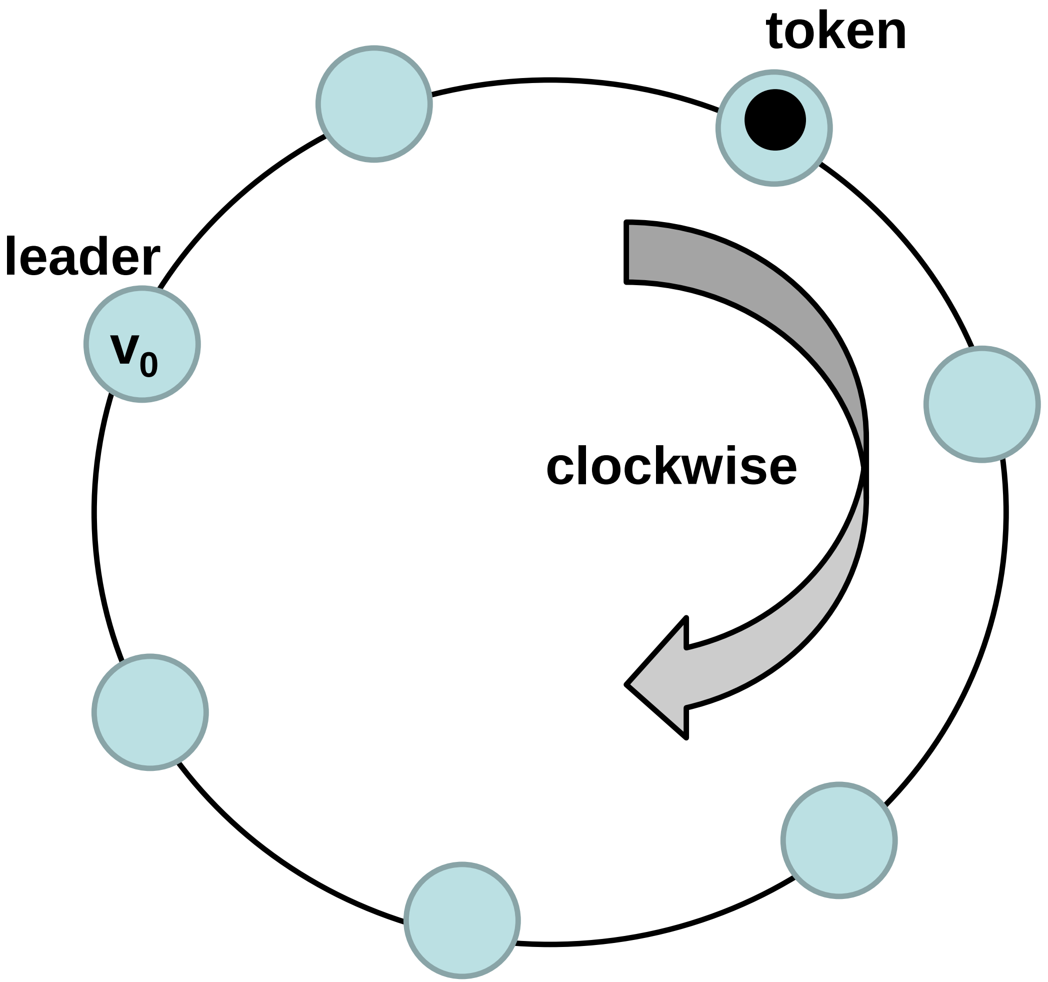
Assume: leader node given, and n known.

Child and Parent

Note, given orientation, we can use the notion of child and parent.

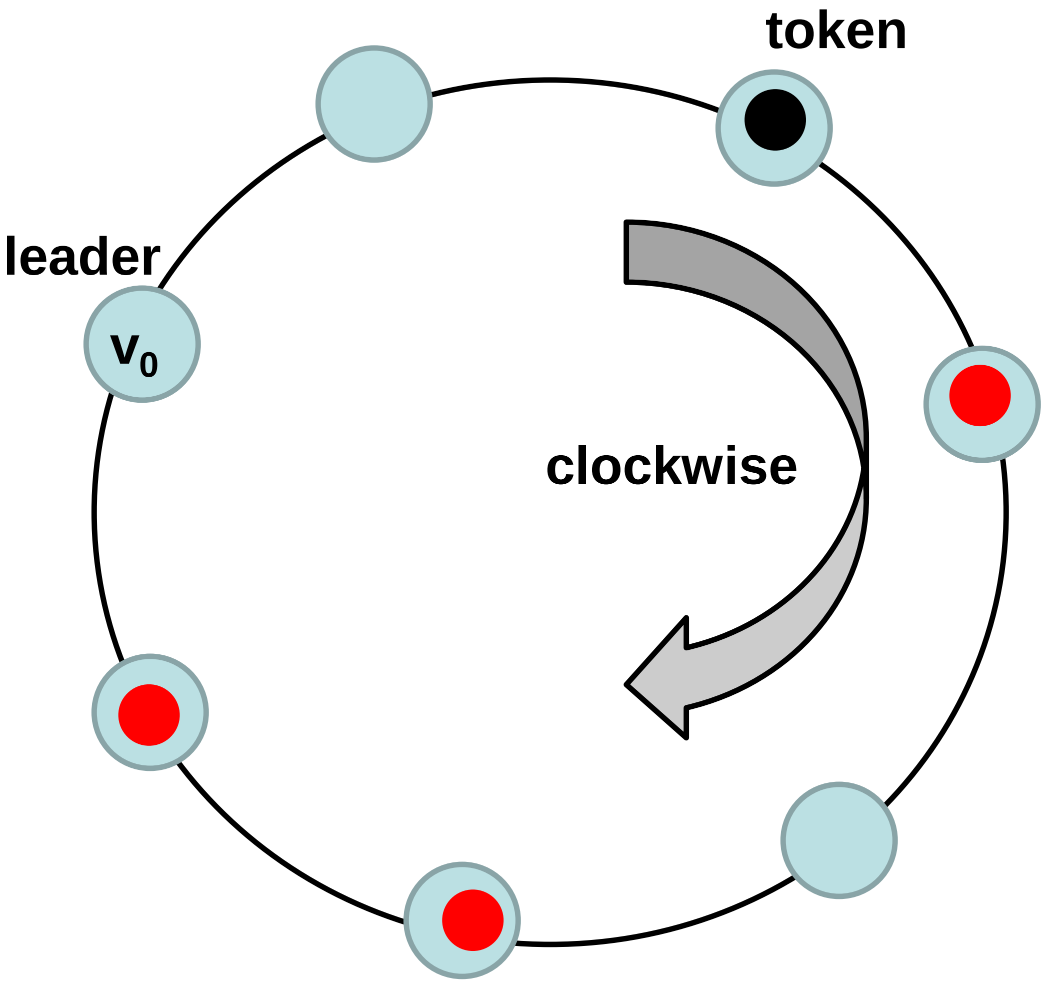


Adversary Model



Adversary may add and remove many tokens anytime!

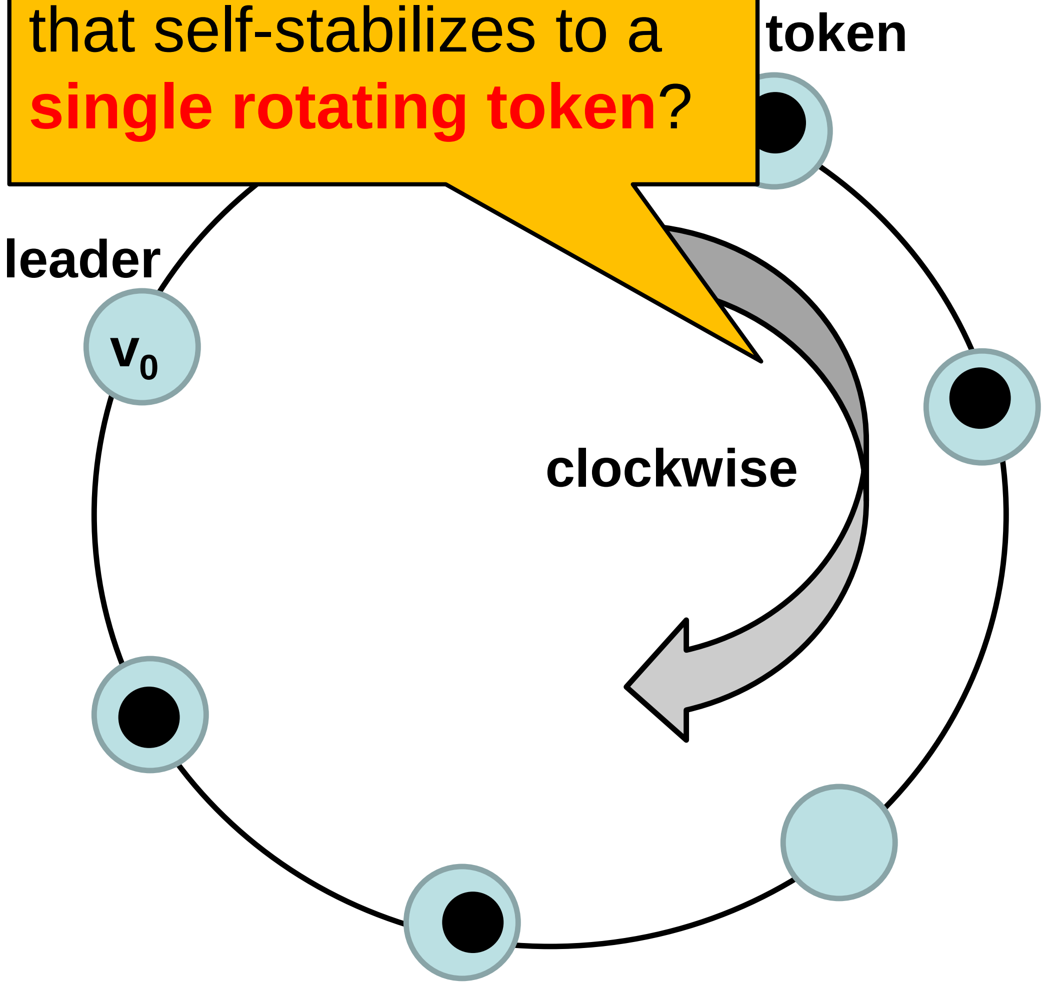
Adversary Model



Adversary may add and remove many tokens anytime!

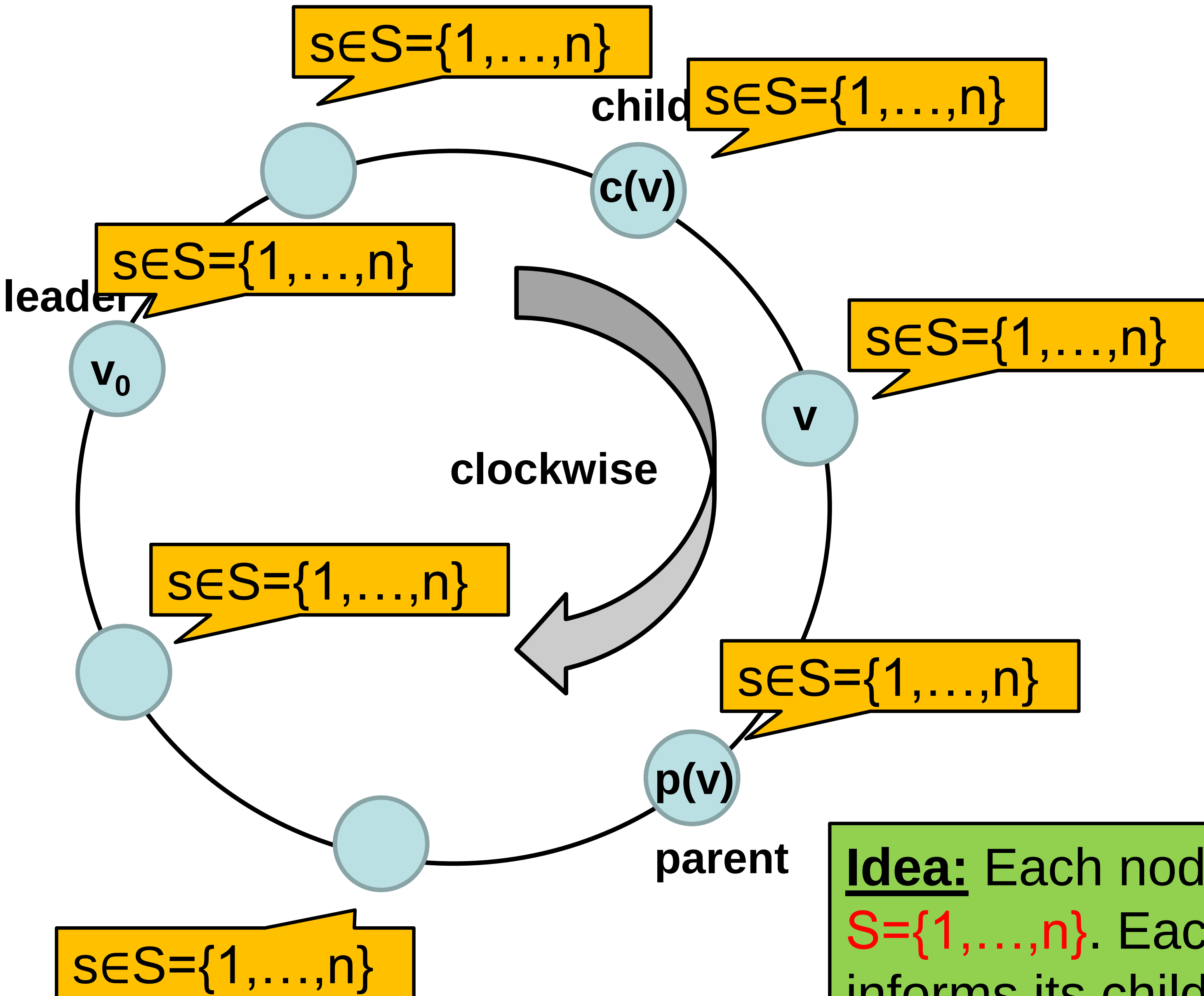
Possible initial configuration!

Distributed algorithm that self-stabilizes to a **single rotating token**?

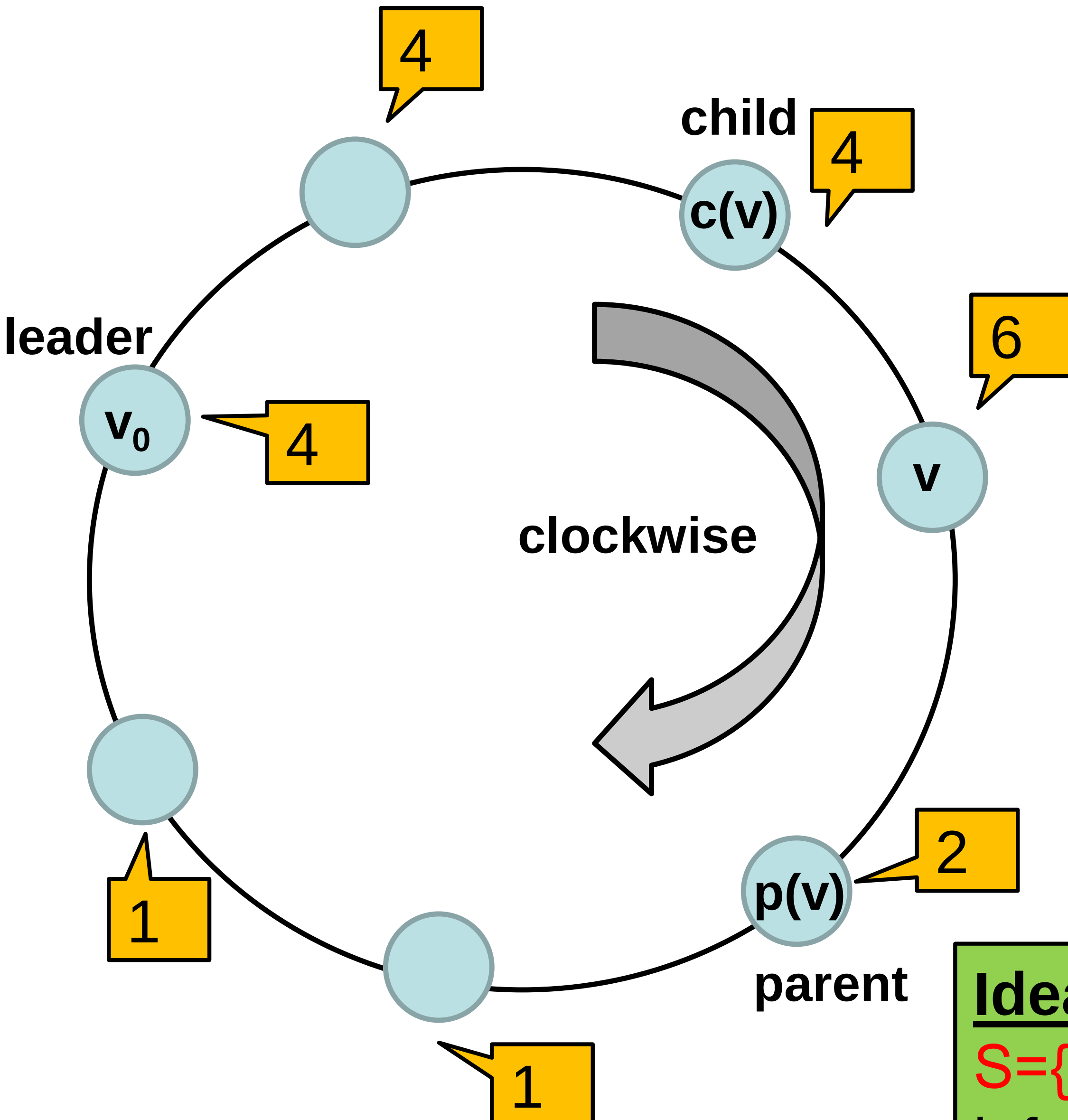


Adversary may add and remove many tokens anytime!

Possible initial configuration!

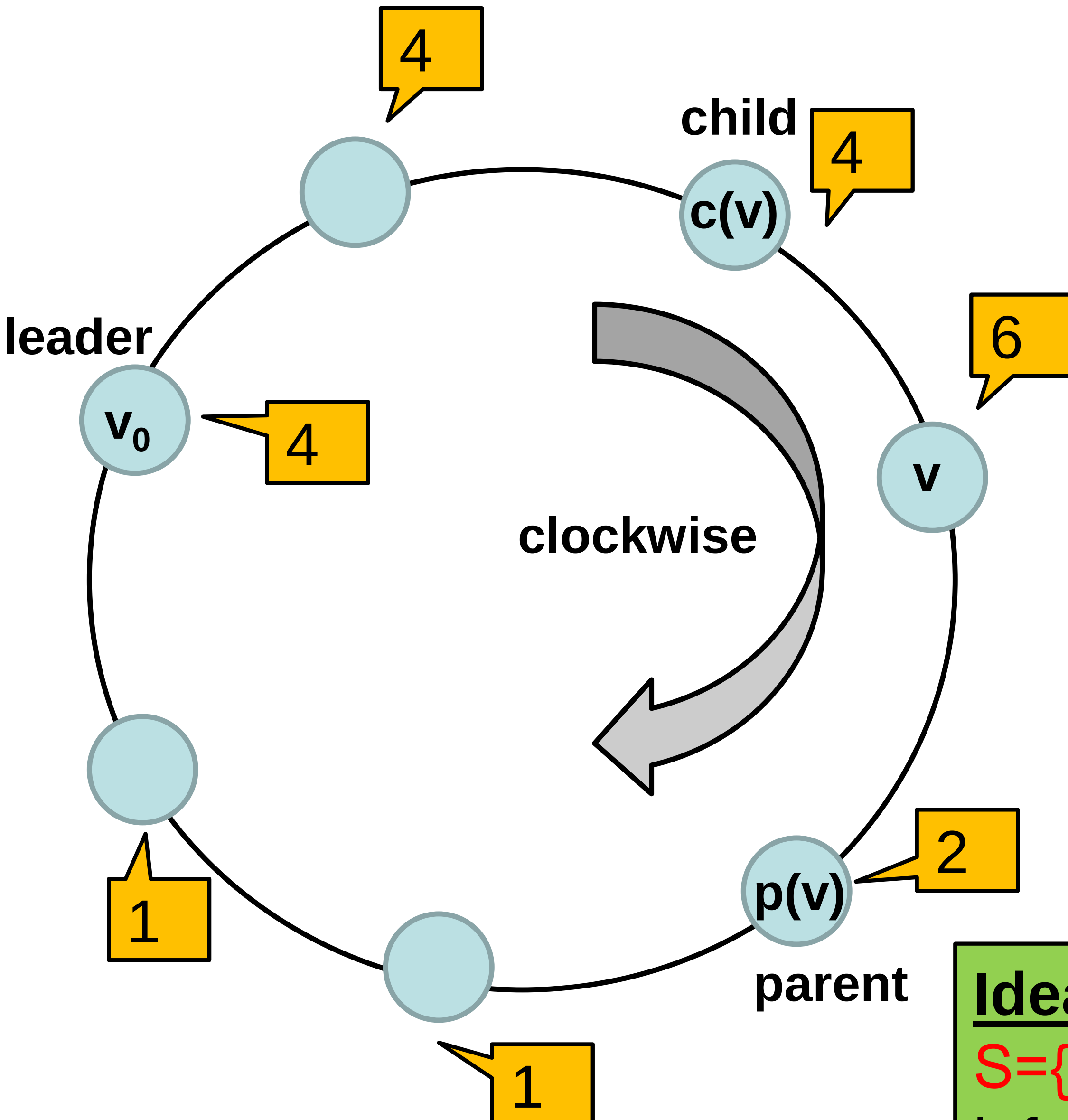


Idea: Each node is in a state S={1,…,n}. Each node informs its child continuously about its state.



Example

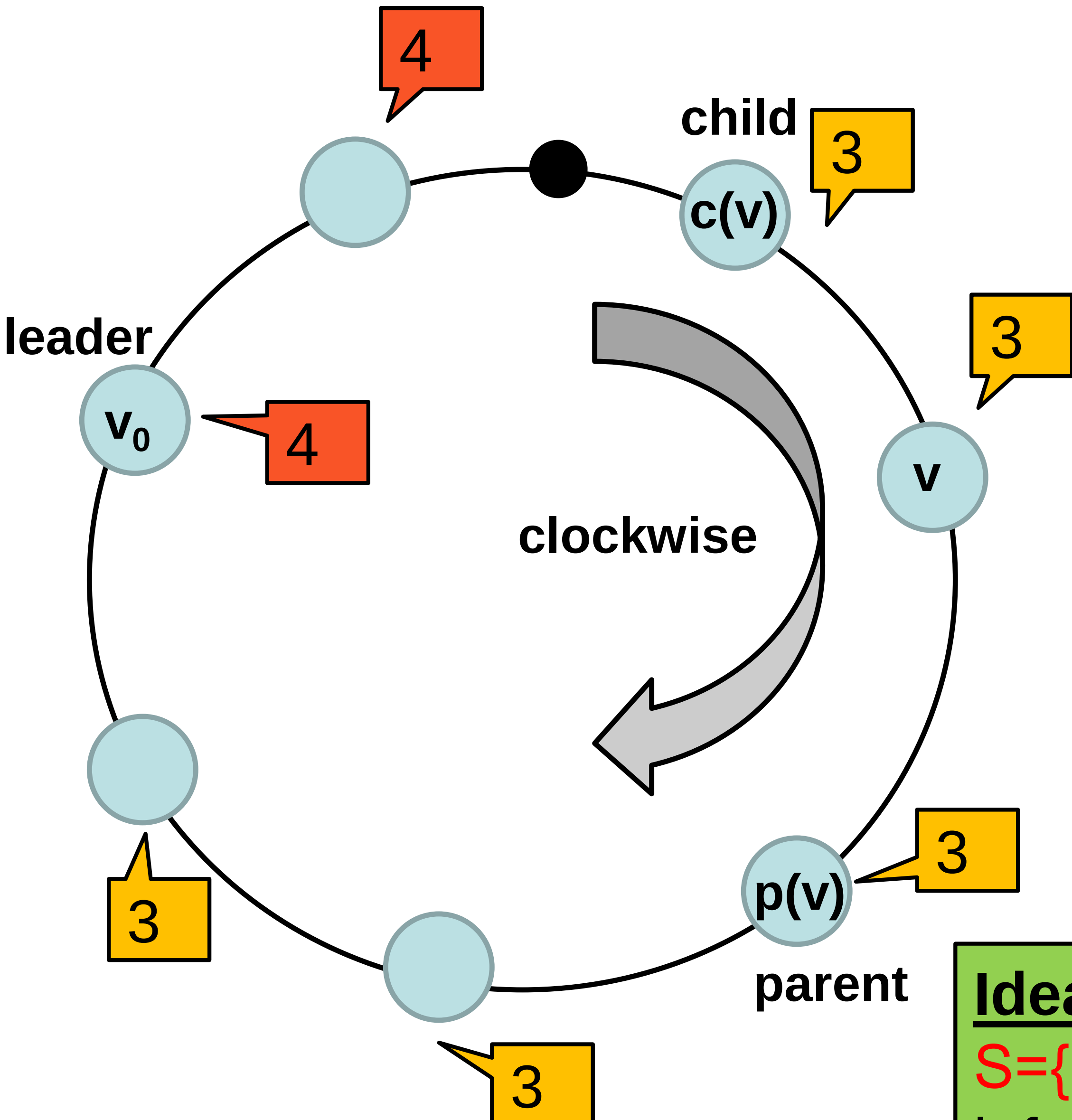
Idea: Each node is in a state S={1,…,n}. Each node informs its child continuously about its state.



Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!

Example

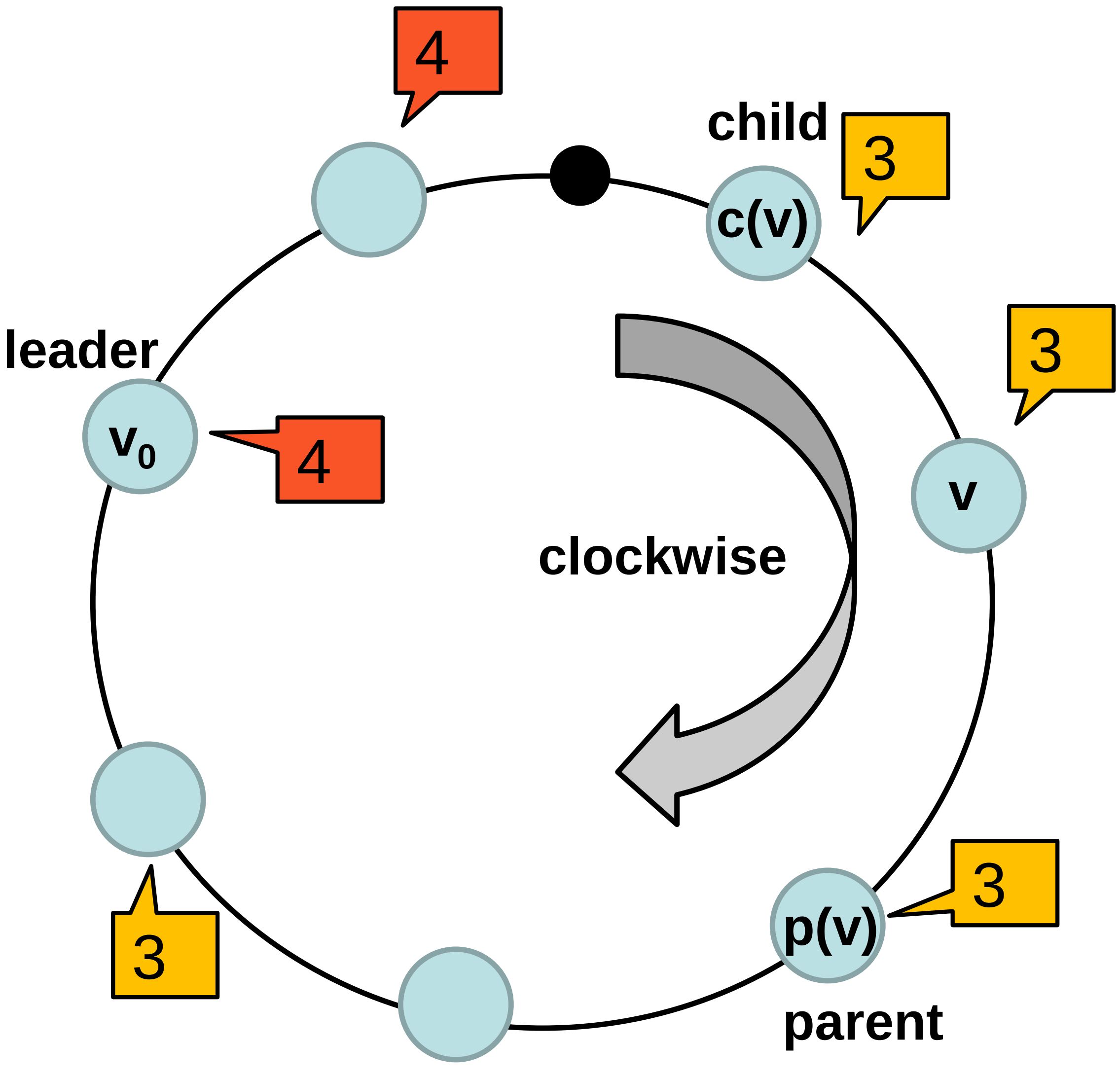
Idea: Each node is in a state S={1,…,n}. Each node informs its child continuously about its state.



Example

Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!

Idea: Each node is in a state S={1,…,n}. Each node informs its child continuously about its state.



Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!

Token Ring

If $v = v_0$ then

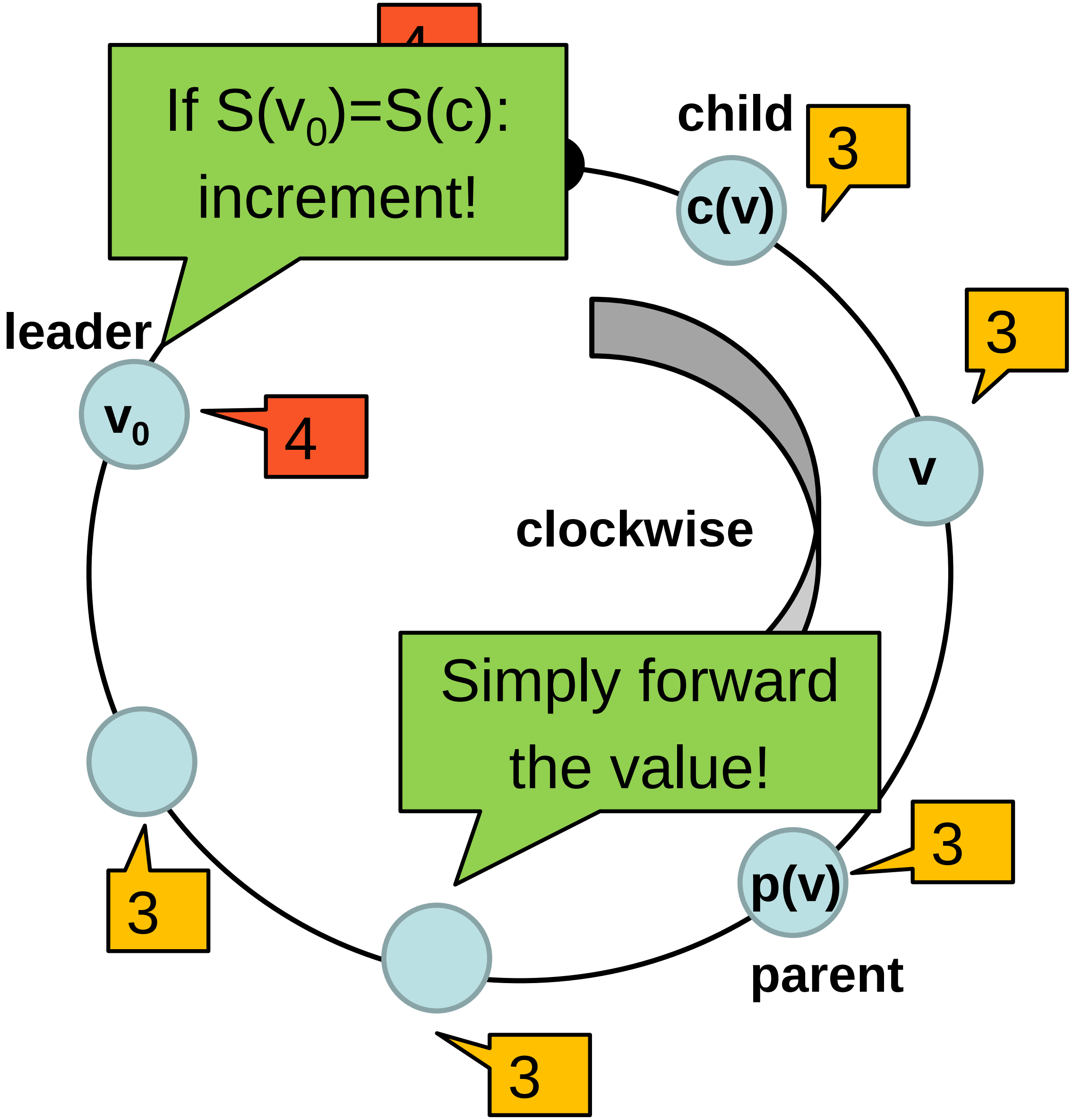
If S(v)=S(c) then

S(v) := S(v) +1 (mod n)

End If

Else S(v):=S(c)

Example

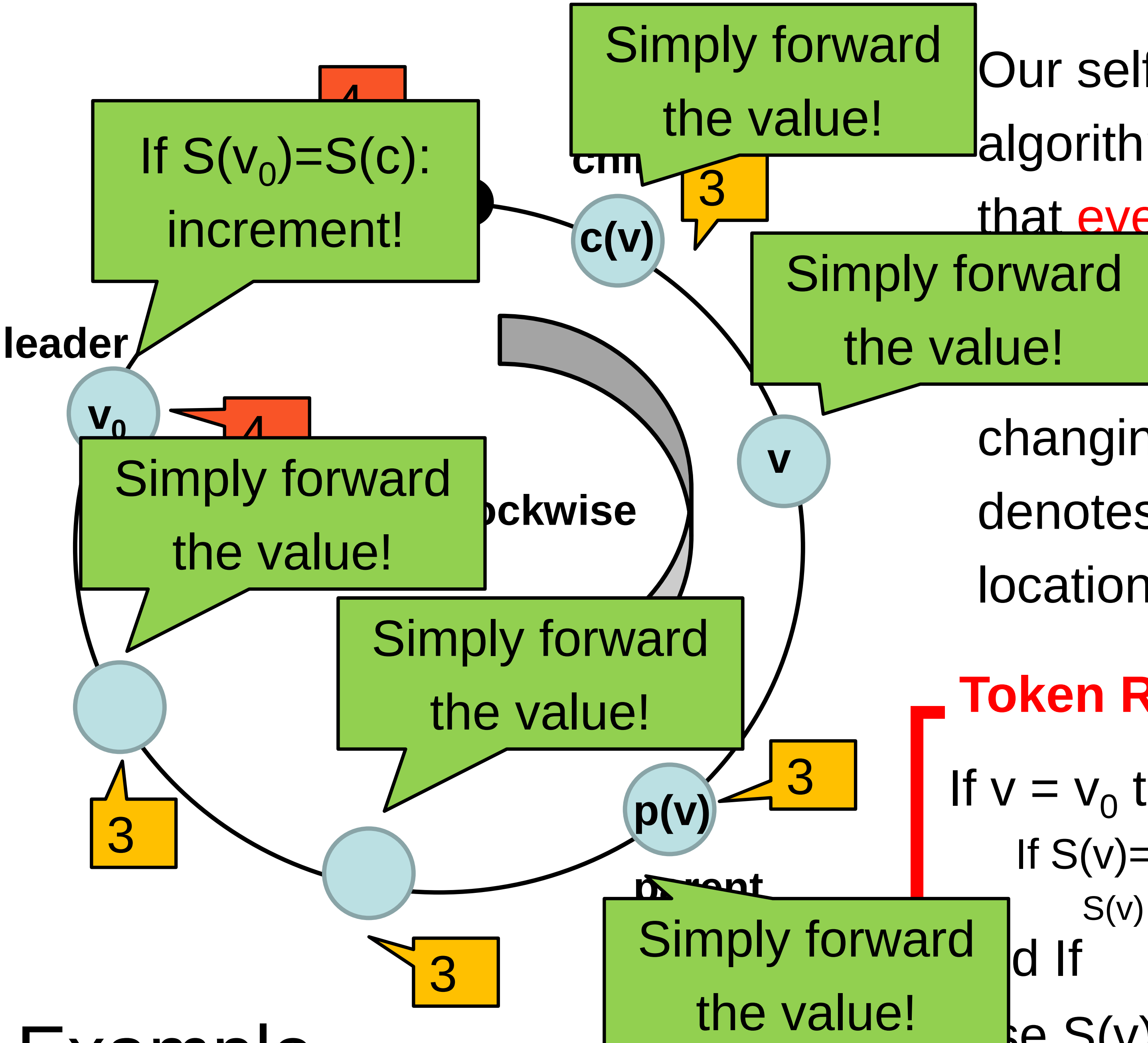


Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!

Token Ring

If $v = v_0$ then
  If $S(v)=S(c)$ then
    $S(v) := S(v) +1 \pmod n$
End If
Else $S(v):=S(c)$

Example



Our self-stabilizing algorithm will ensure that eventually there … two …s: the changing point denotes the token location!

Token Ring

If v = v_0 then

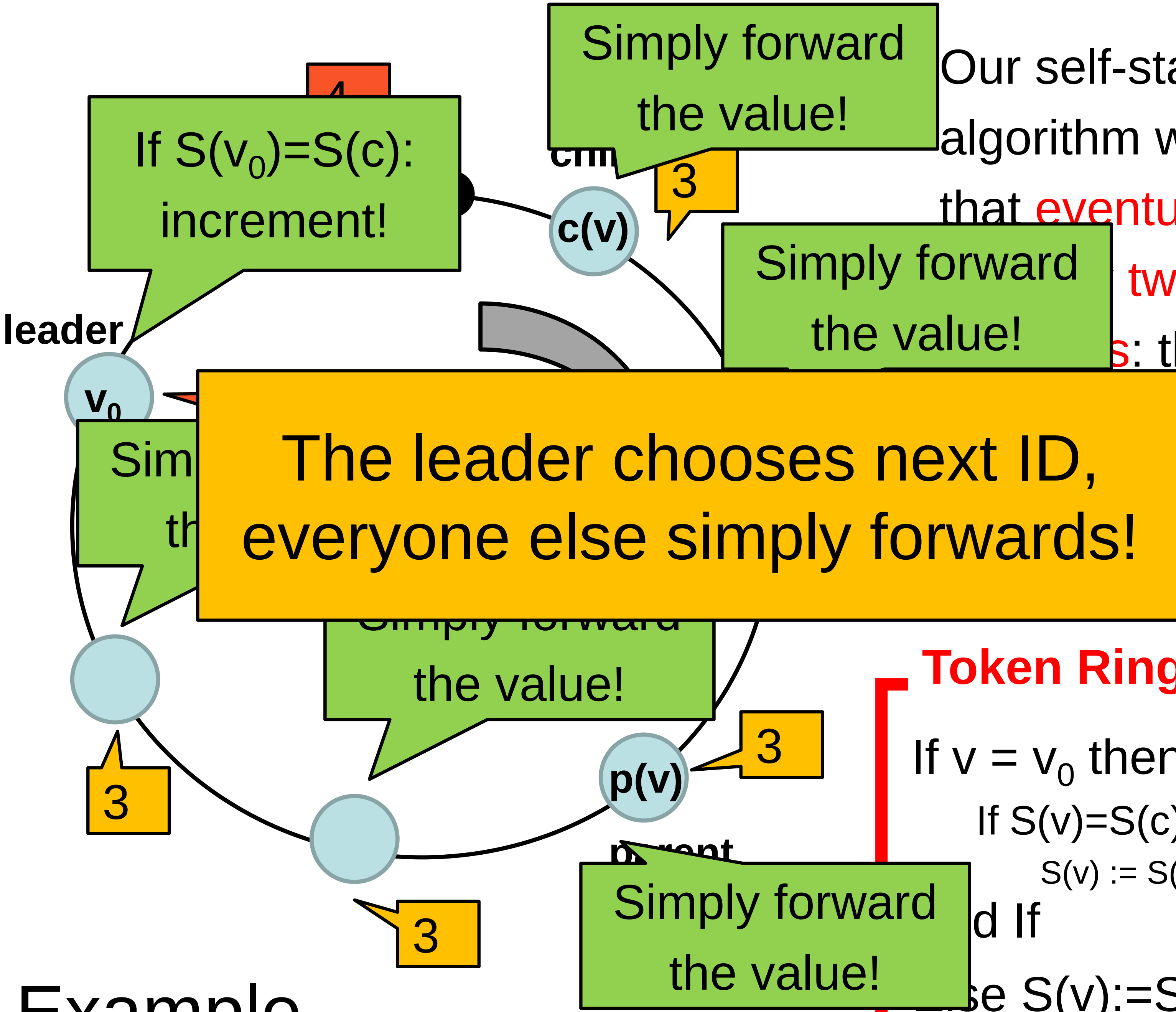
If S(v)=S(c) then

S(v) := S(v) +1 (mod n)

…d If

…se S(v):=S(c)

Example



Our self-stabilizing algorithm will ensure that eventually there two s: the oint e token

Token Ring

If $v = v_0$ then

If $S(v)=S(c)$ then

$S(v) := S(v) +1 \pmod n$

d If

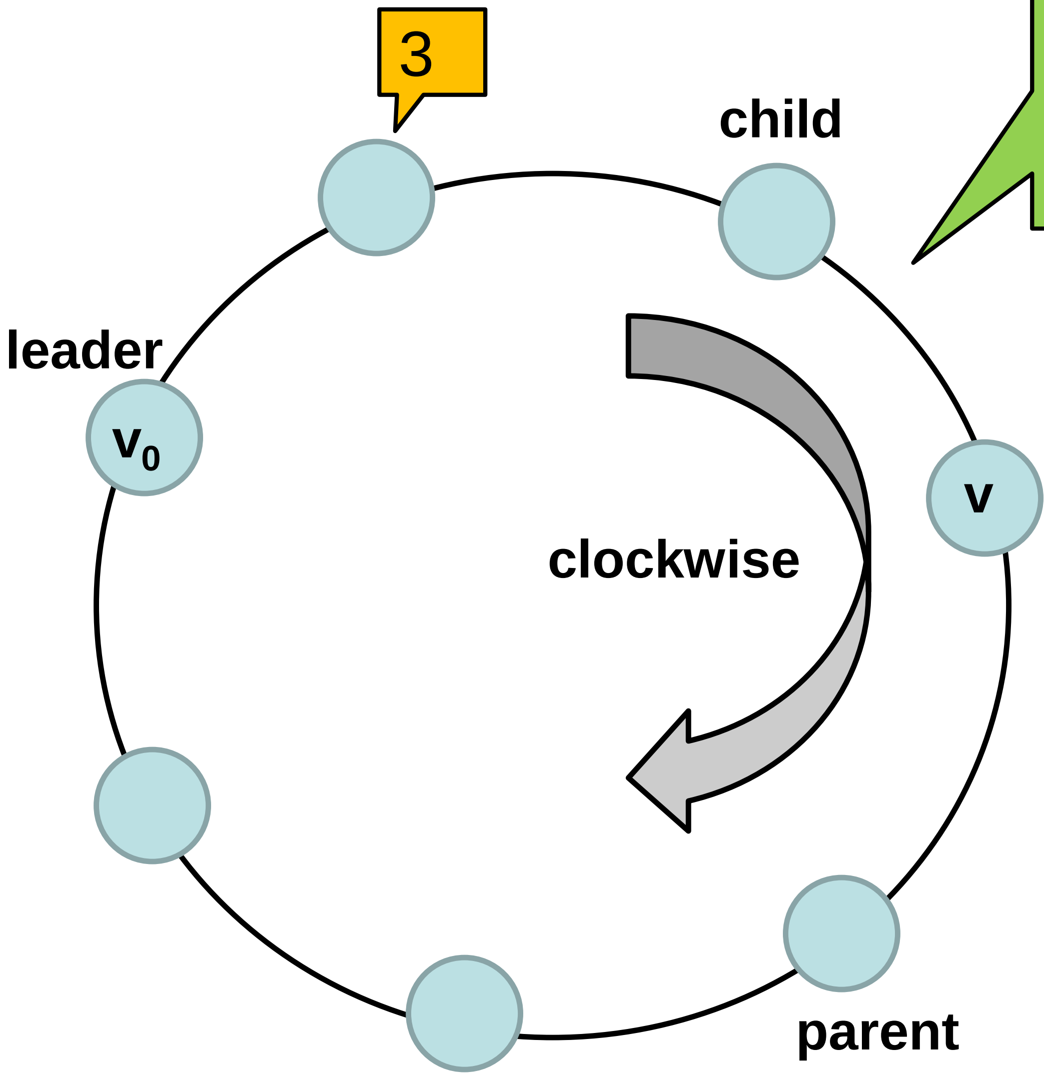
se $S(v):=S(c)$

Example

Token Ring

The algorithm stabilizes ... th...

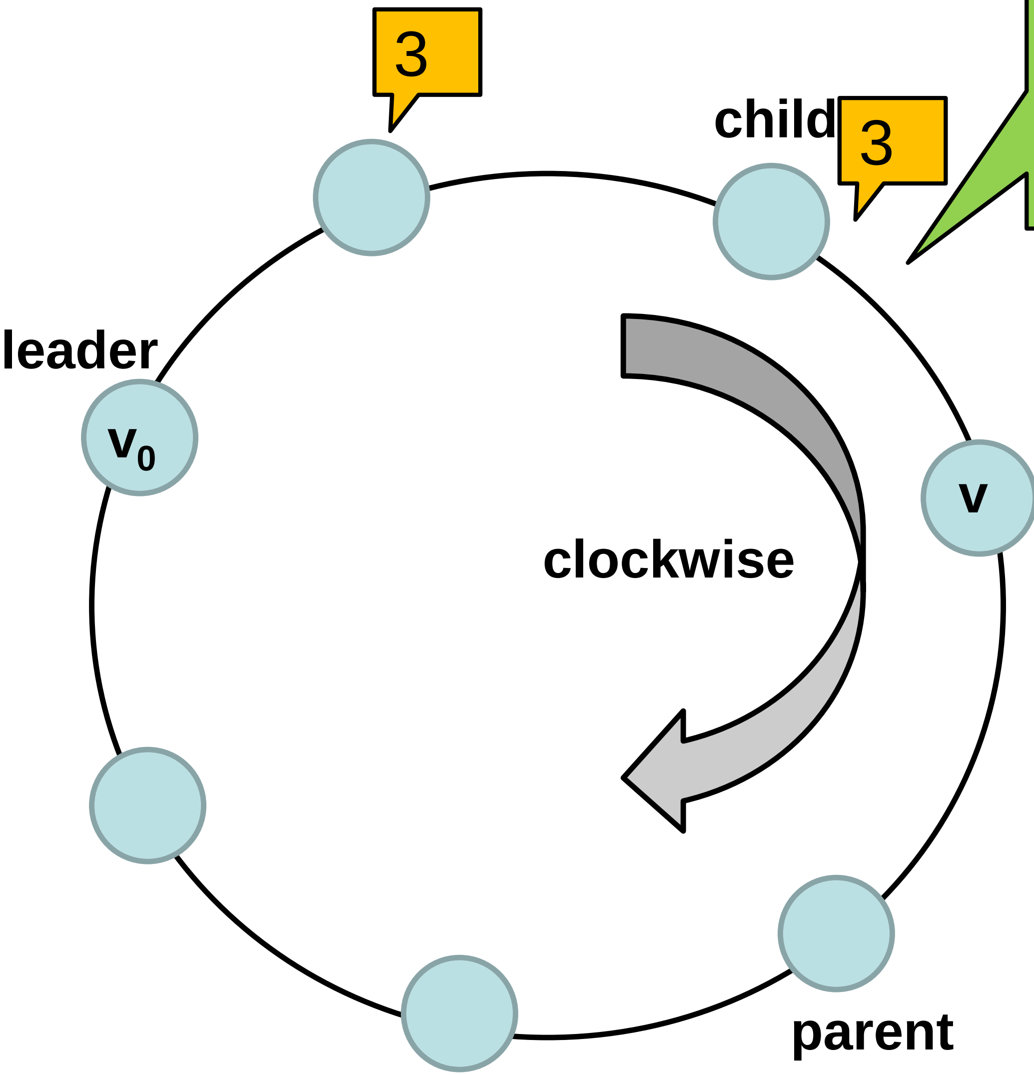
1. eventually, each node just copies from its child: same value throughout the ring.



Token Ring

The algorithm stabilizes

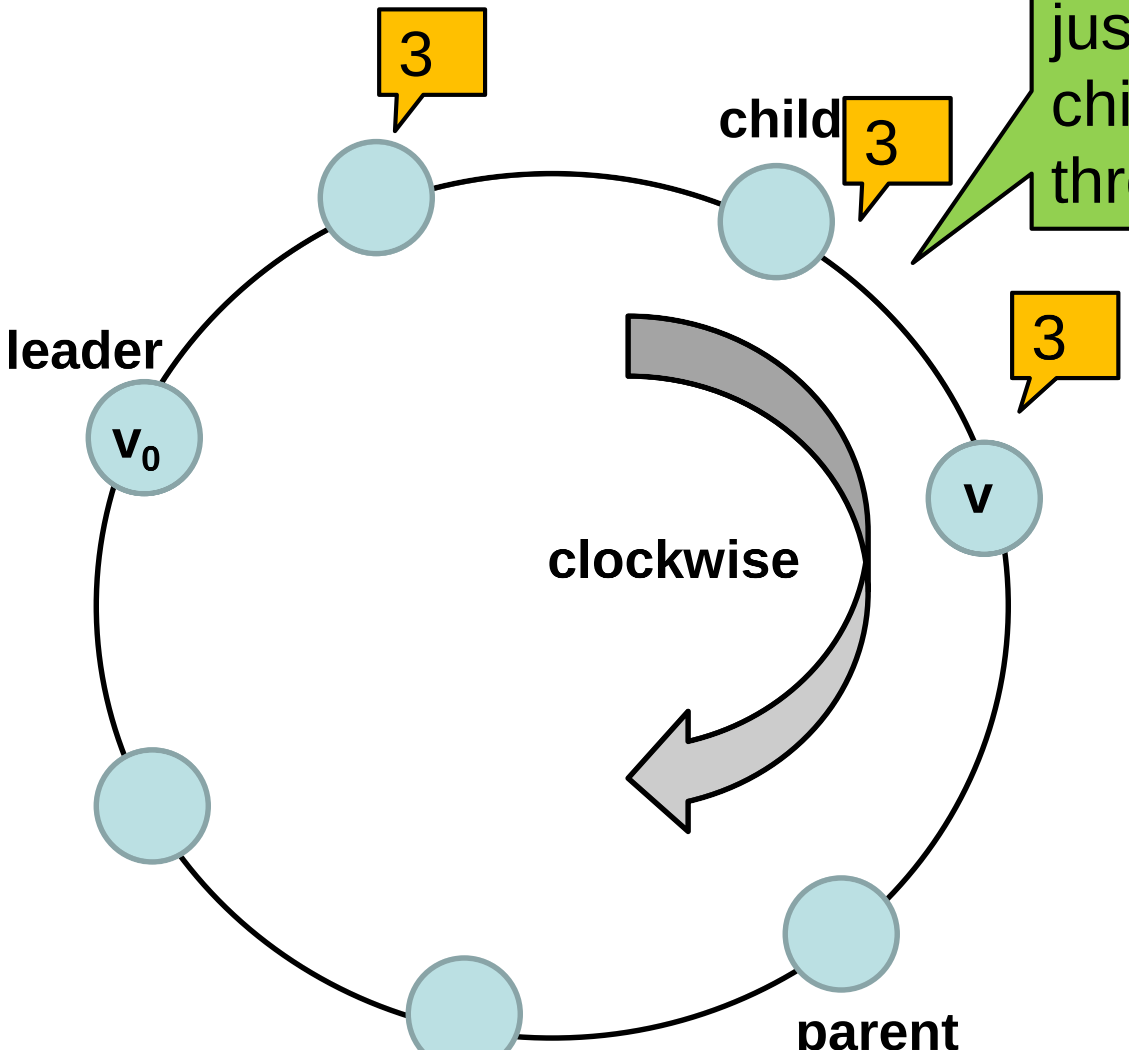
1. eventually, each node just copies from its child: same value throughout the ring.



Token Ring

The algorithm stabilizes ... th

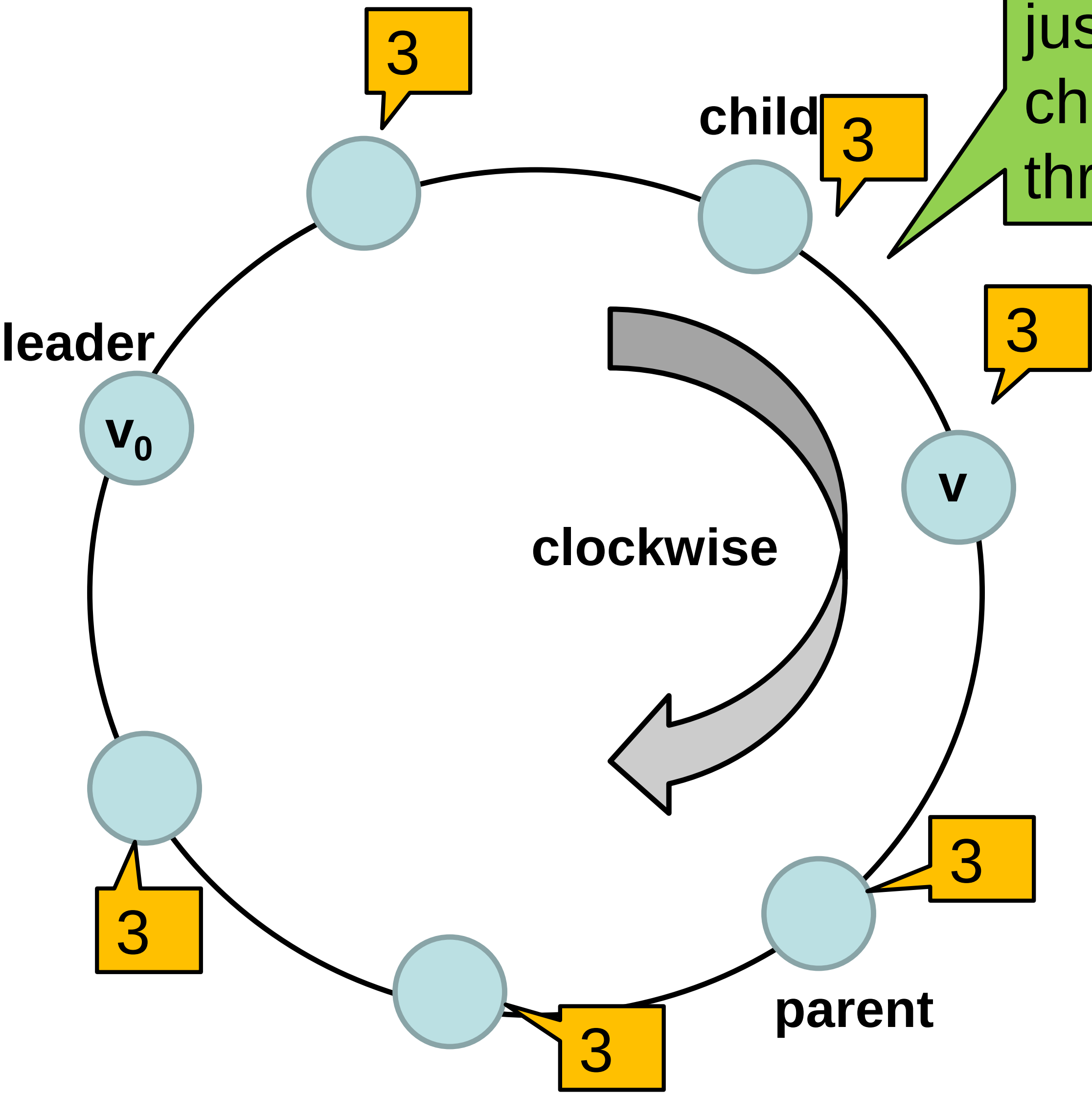
1. eventually, each node just copies from its child: same value throughout the ring.



Token Ring

The algorithm stabilizes

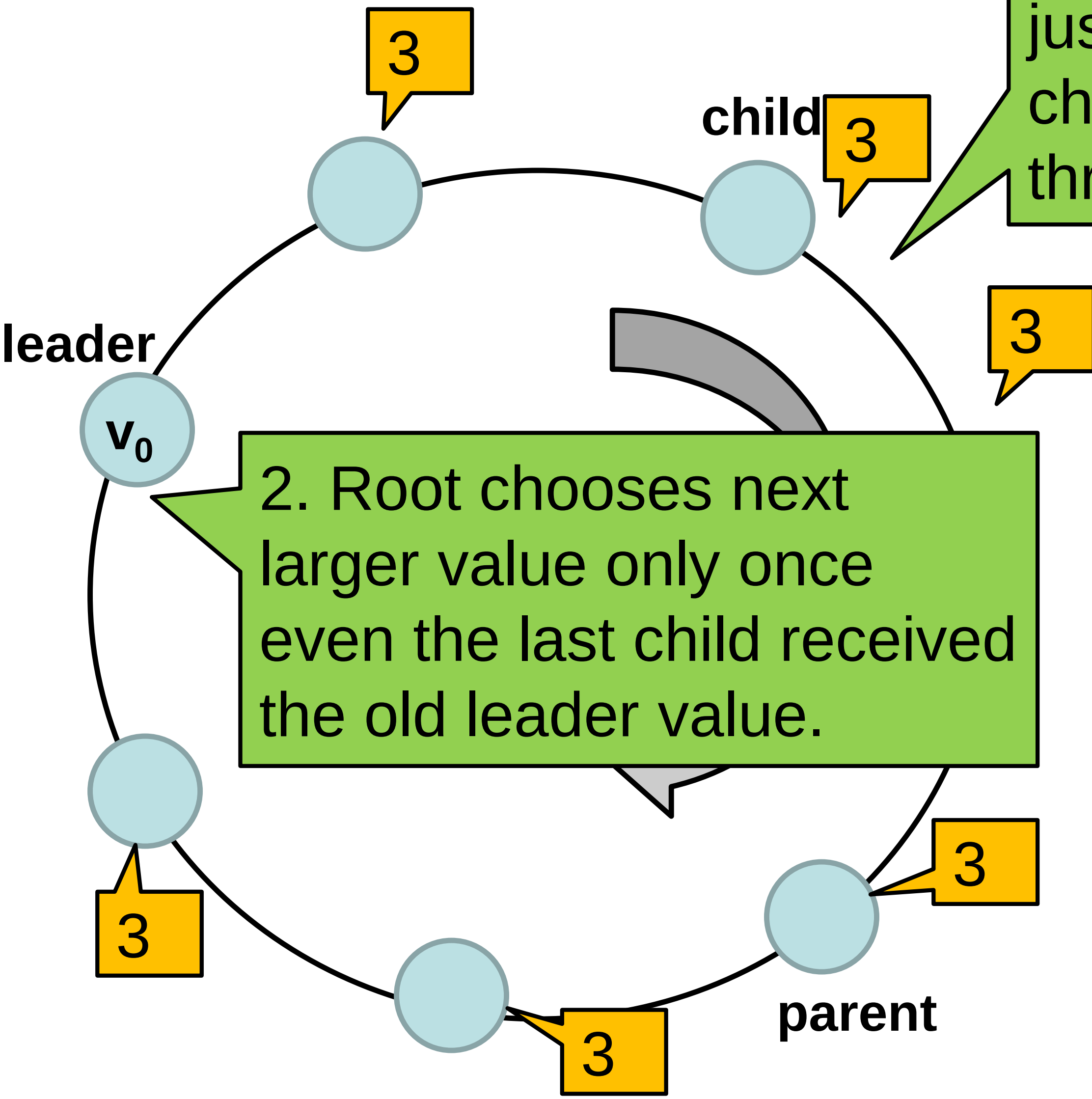
1. eventually, each node just copies from its child: same value throughout the ring.



Token Ring

The algorithm stabilizes

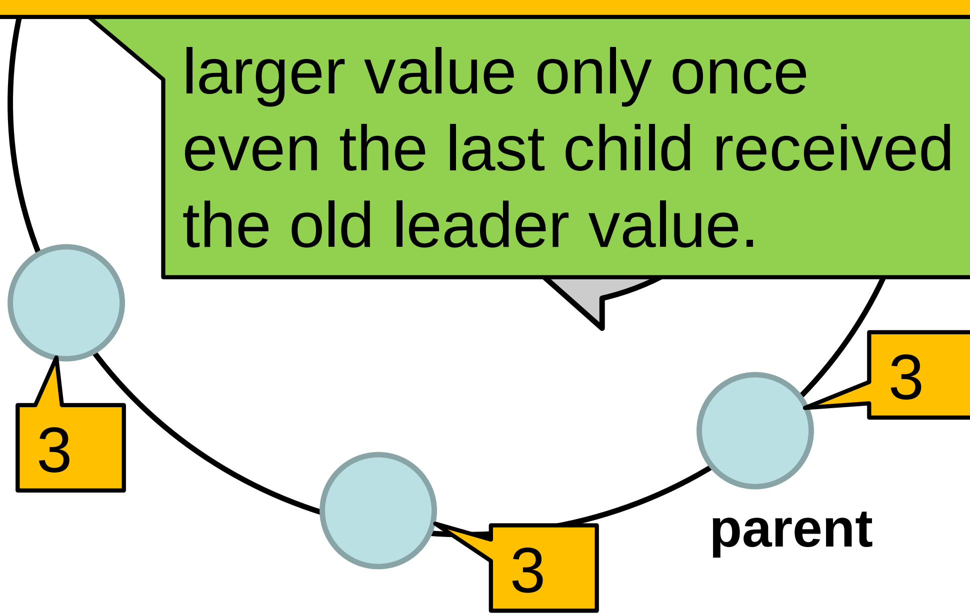
1. eventually, each node just copies from its child: same value throughout the ring.



2. Root chooses next larger value only once even the last child received the old leader value.

Eventually:

- The leader will reach a state *s* that no other node had at time *t*0. (There are *n* nodes and *n* states.)
- Then one node after the other will learn the current state of the leader.
- The leader itself does not push the next value until the previous value travelled the entire ring!
- At most one node active at any time: Token passed implicitly with the switching state.



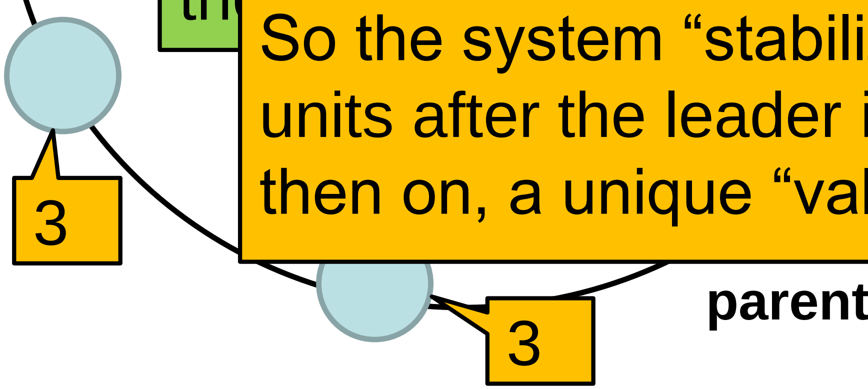
Eventually:

- The leader will reach a state *s* that no other node had at time *t*0. (There are *n* nodes and *n* states.)
- Then one node after the other will learn the current state of the leader.
- The leader itself does not push the next value until the previous value travelled the entire ring!
- At most one node active at any time: Token passed implicitly with the switching state.

e

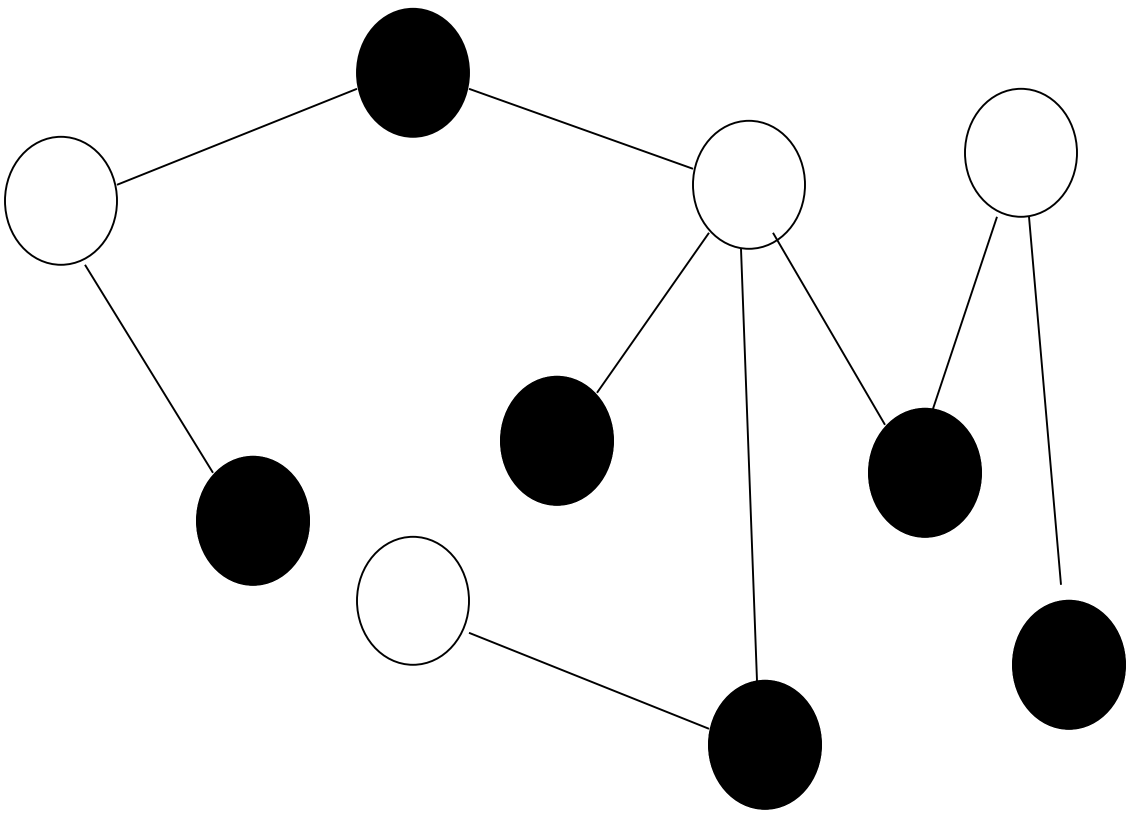
larger value only once
even the last child received
th

So the system “stabilizes” after at most *n* time units after the leader increased the value: from then on, a unique “value change” cycles the ring.



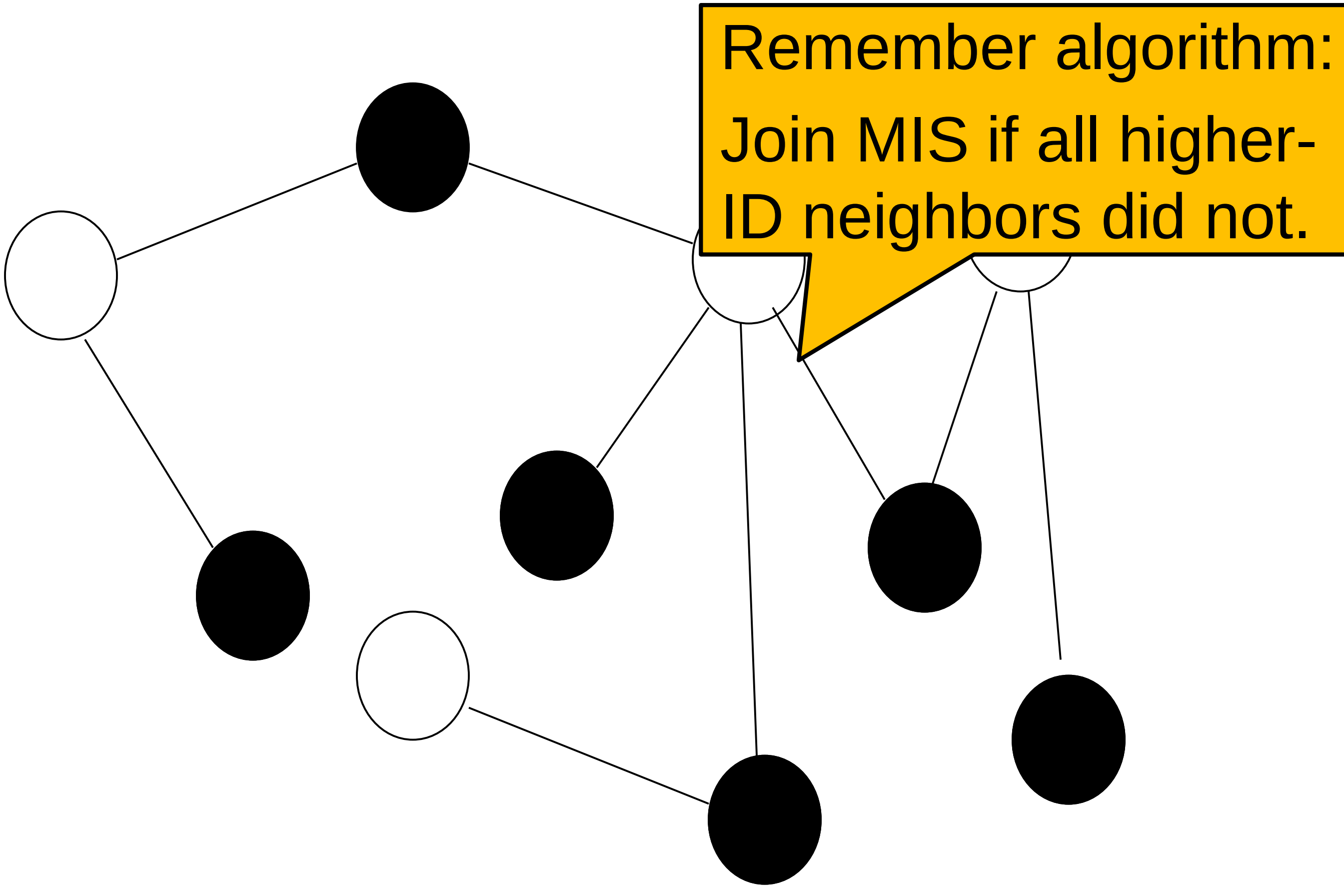
Self-Stabilizing Independent Sets

How to design self-stabilizing Maximal Independent Sets?



Self-Stabilizing Independent Sets

How to design self-stabilizing Maximal Independent Sets?



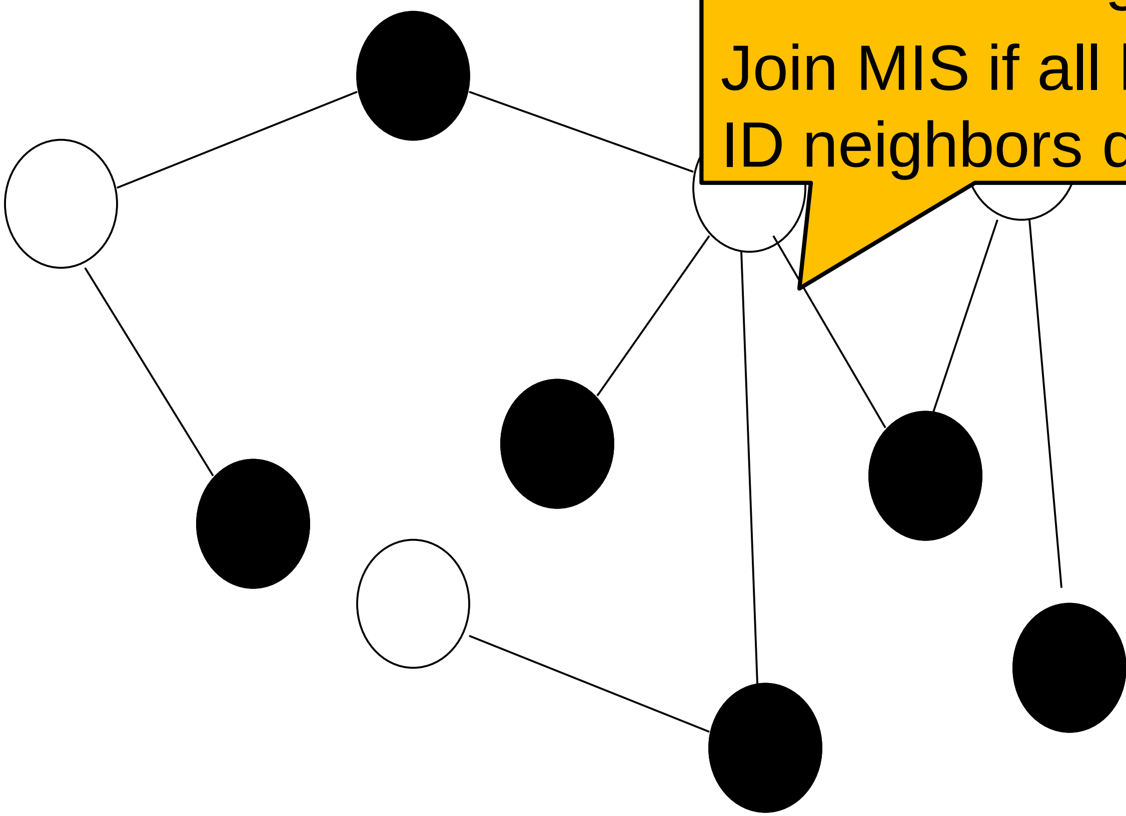
Self-Stabiliz

Idea: Make it self-stabilizing by executing this continuously!

How to design self-stabilizing Maximal … pendent Sets?

Remember algorithm:

Join MIS if all higher-ID neighbors did not.



Assume: node have unique IDs

Independent Sets

Every node *v* executes the following code:

1: do atomically (forever)

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

Assume: node have unique IDs

Independent Sets

Every node *v* executes the following code:

1: do atomically (forever)

2: Leave MIS if a neighbor with a larger ID is in the MIS

3: Join MIS if no neighbor with larger ID is in MIS

4: Send (node ID, MIS or not MIS) to all neighbors

5: end do

Why does it work? For same reason as before: eventually, highest-ID node will make decision, then its neighbors, then…

Self-Stabilizing Independent Sets

Assume: node have unique IDs

Independent Sets

Every node *v* executes the following code:

1

Can we make any LOCAL algorithm self-stabilizing?
E.g., coloring, matching, …?

2 … IS

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

Why does it work? For same reason as before: eventually, highest-ID node will make decision, then its neighbors, then…

Self-Stabilizing Independent Sets

Assume: node have unique IDs

Independent Sets

Yes! Automatic transformation.

Every node v executes the following code:

1

2 ... IS

Can we make any LOCAL algorithm self-stabilizing?
E.g., coloring, matching, …?

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

Why does it work? For same reason as before: eventually, highest-ID node will make decision, then its neighbors, then…

Transformation

Given:

Deterministic k-round LOCAL algorithm A.

Output:

k-round self-stab LOCAL algorithm, i.e.:

- if the adversary does not corrupt the system for k time units, the solution is stable
- if the adversary does not corrupt any node or message closer than distance k from a node u, node u will be stable (locality)

Idea: simulate all k-rounds in parallel! Annotate messages with round they belong to, and keep them in different tables for each round: local variables of the last k rounds.

Given:

Deterministic k-round LOCAL algorithm A.

A.k.a. *local checking*. Proof **by induction**: after t0, round 1 variables and messages will be correct, then round 2 variables and messages, then …

- if the adversary does not corrupt the system for k time units, the solution is stable
- if the adversary does not corrupt any node or message closer than distance k from a node u, node u will be stable (locality)

Idea: simulate all k-rounds in parallel! Annotate messages with round they belong to, and keep them in different tables for each round: local variables of the last k rounds.

Given:

Deterministic k-round LOCAL algorithm A.

A.k.a. *local checking*. Proof **by induction**: after t0, round 1 variables and messages will be correct, then round 2 variables and messages, then …

k-round self-stab LOCAL algorithm, i.e.:

- if the adversary does not corrupt the system for k time units, the solution is stable
- if the adversary does not corrupt any node or message closer

It is automatic: from Art to Craft!

Advanced Stabilization

Sometimes stabilization is not to a fixed state but to a cyclic state! E.g., token ring. Here comes another example!

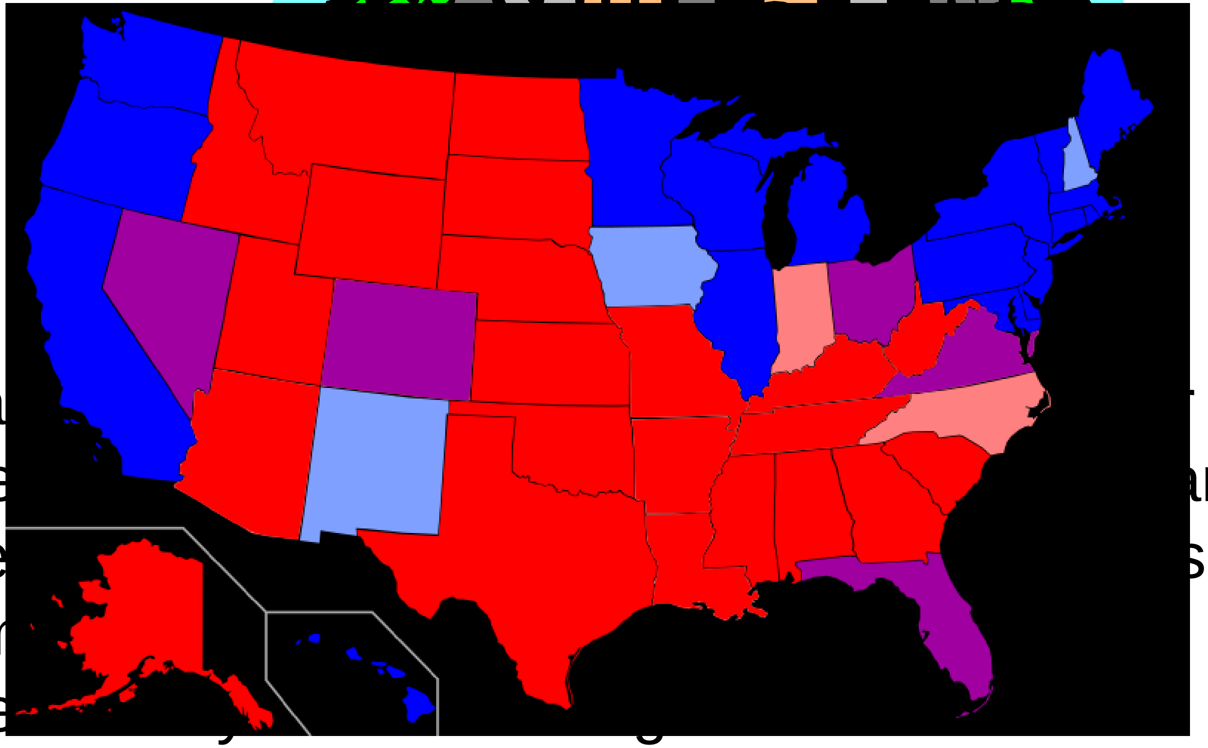
Advanced Stabilization

In a little town, each evening citizens call their friends to ask whether they vote for Democrats or Republicans.
Then they decide themselves for majority (assume odd number of friends).
Does this system «converge» or «stabilize»?

Advanced Stabilization

Democratic

Republicans



In a

friends to ask

whe

ans.

The

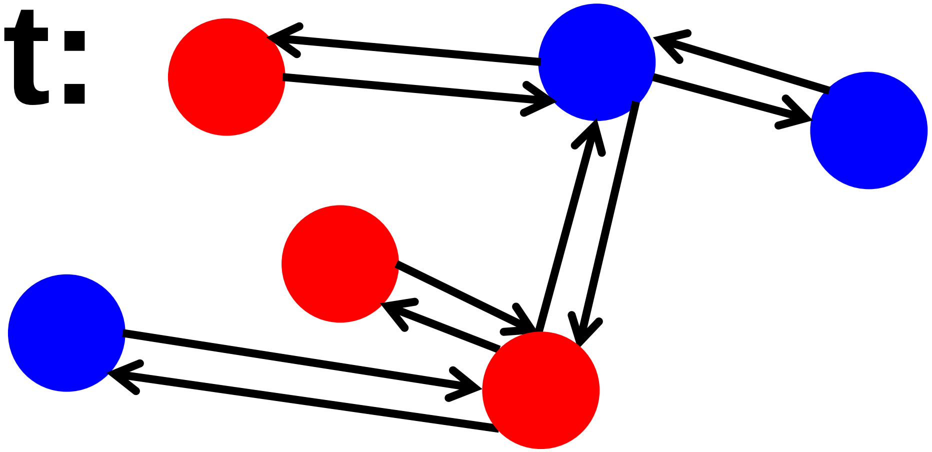
sume odd

num

Doe

Example

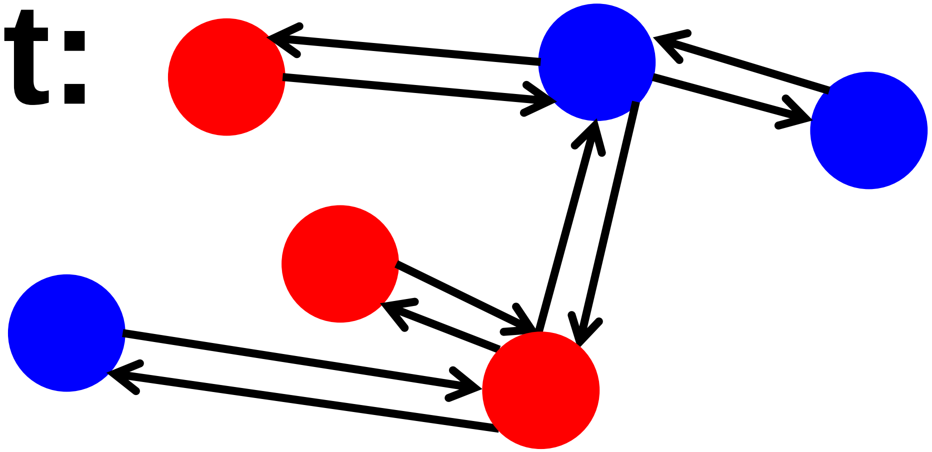
t:



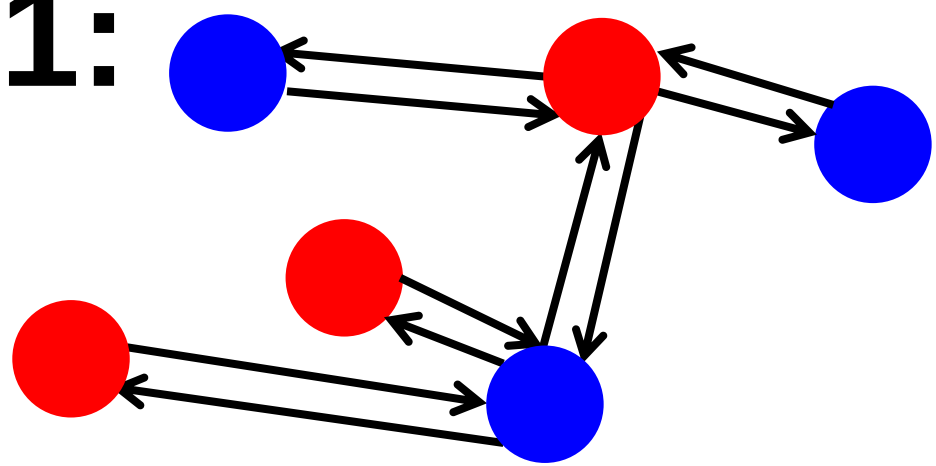
t+1:

Example

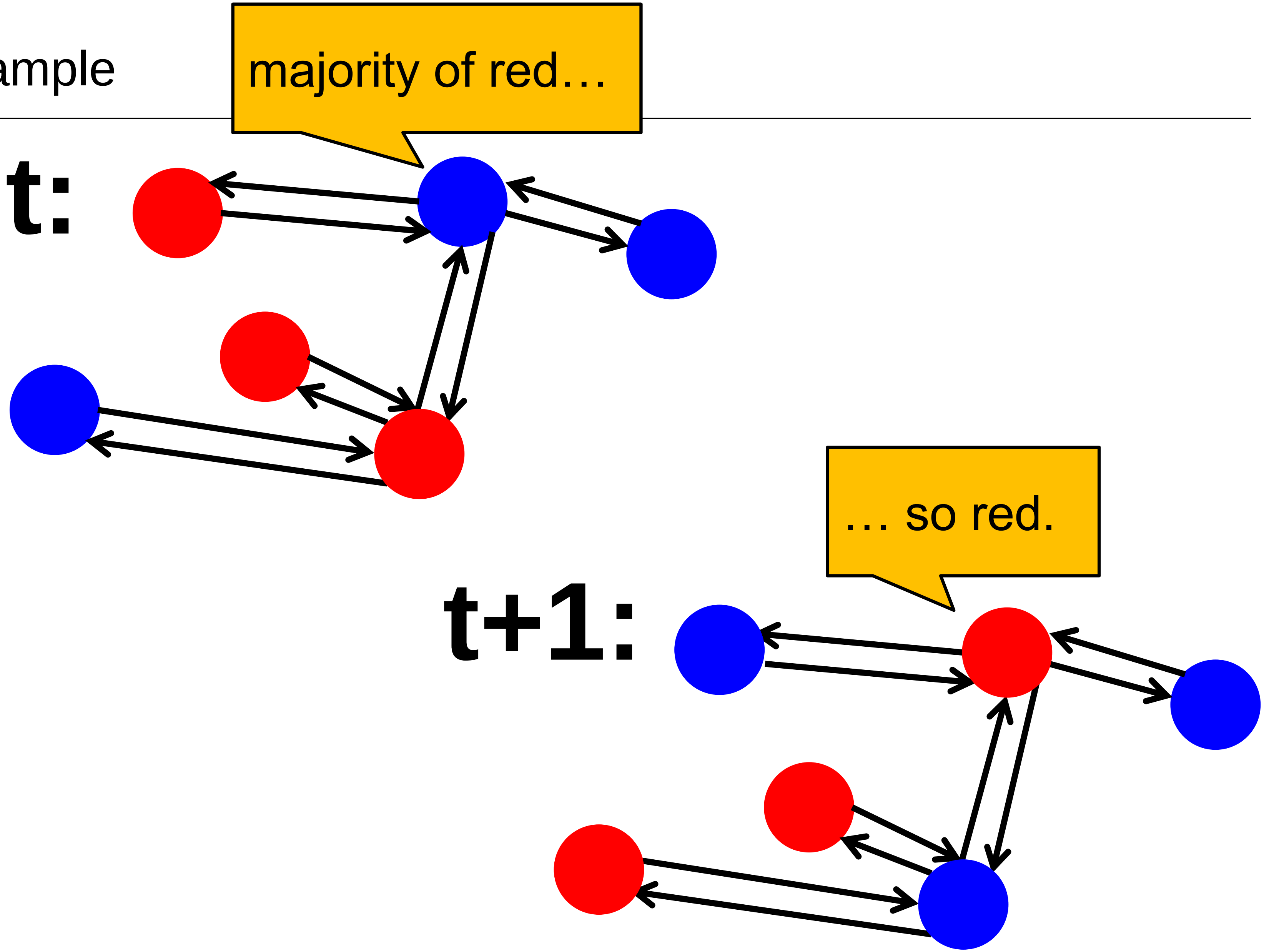
t:



t+1:



Example



What do you think?

- Does eventually everybody vote for the same party?

- Will each citizen eventually stay with the same party?

- Will citizens who stayed with the same party for some time, stay with that party forever?

- And if their friends also constantly root for the same party?

- Will this beast stabilize at all? ☺

What do you think?

- Does eventually everybody vote for the same party?
- Will each citizen eventually stay with the same party?
- Will citizens who stayed with the same party for some time, stay with that party forever?
- And if their friends also constantly root for the same party?
- Will this beast stabilize at a

No, no, no! ☺

Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

At least one can show this:
Some kind of convergence...

Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Why?

Democrats / Republicans

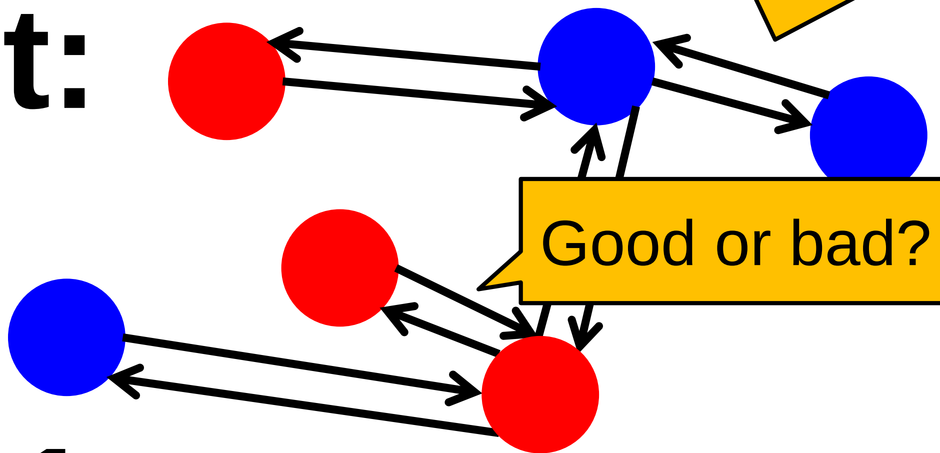
Eventually each citizen will vote for the same party every other day.

Good or bad?

Why?

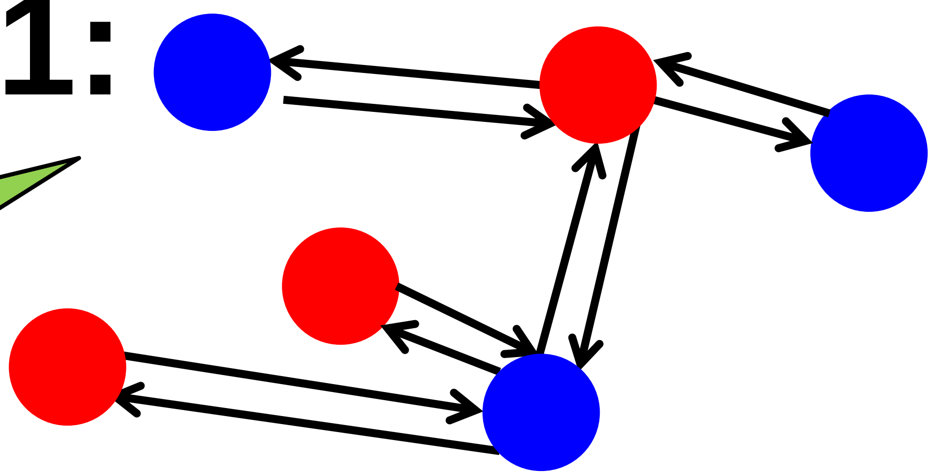
Represent friendship as bidirected edges.

t:



Good or bad?

t+1:



Define bad edge: points to node which does not follow the advisor's opinion on next day!

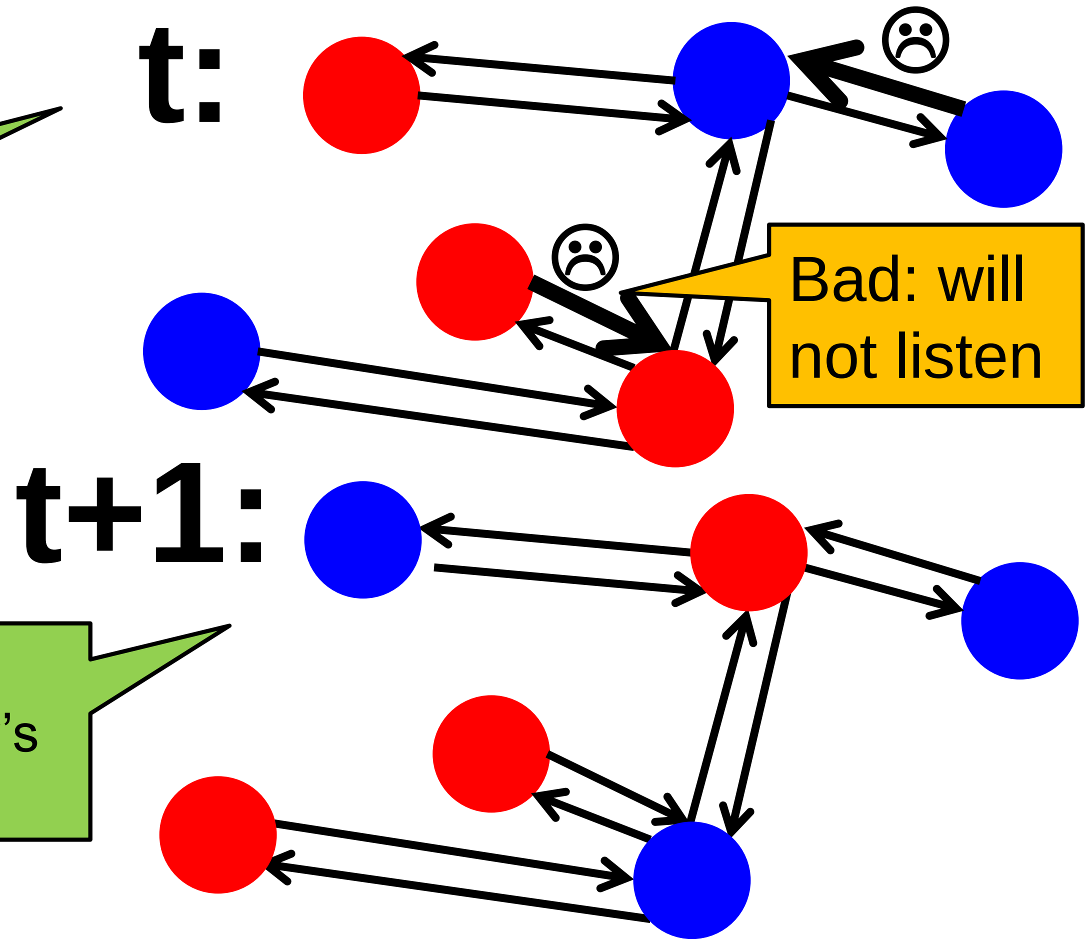
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Bad: will not listen

Why?

Represent friendship as bidirected edges.

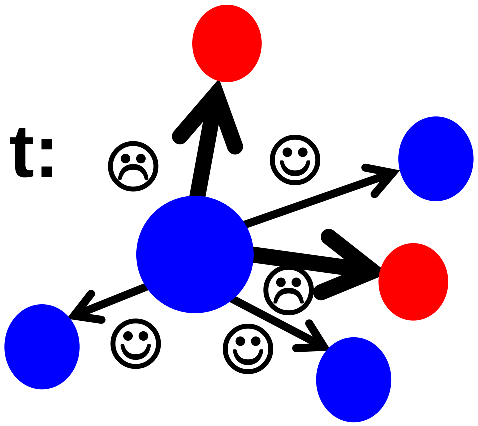


Define bad edge: points to node which does not follow the advisor’s opinion on next day!

Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (**Democrat**) with g good and b bad out-edges on a **day t** (= will be c resp. not c at t+1)
- Degree of citizen c is hence g+b.

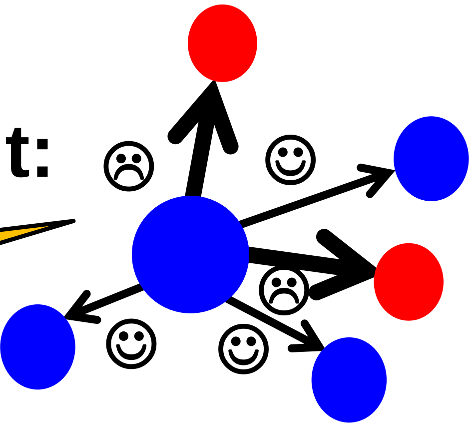


Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (**Democrat**) with g good and b bad out-edges on a **day t** (= will be c resp. not c at t+1)
- Degree of citizen c is hence g+b.

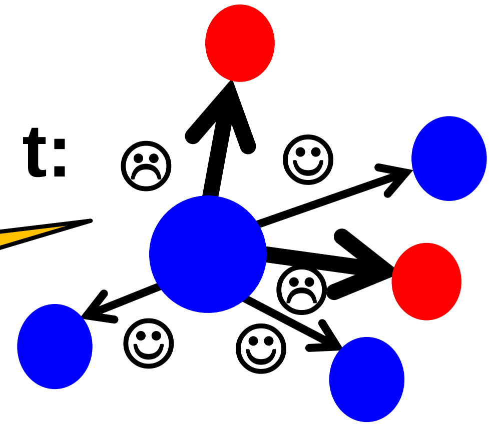
What happens in round t+1? How many neighbors root for which party?



Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (**Democrat**) with g good and b bad out-edges on a **day t** (= will be c resp. not c at t+1)
- Degree of citizen c is hence g+b.



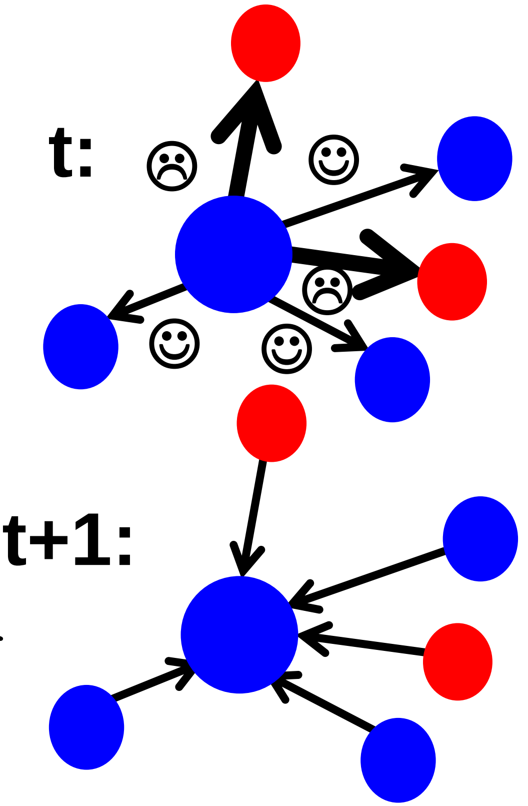
What happens in round t+1? How many neighbors root for which party?

Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (**Democrat**) with g good and b bad out-edges on a **day t** (= will be c resp. not c at t+1)
- Degree of citizen c is hence g+b.

On day *t*+1, *g* friends of *c* root for the Democrats, and *b* friends root for the Republicans. And in evening of *t*+1, *c* will receives *g* recommendations for Democrats, and *b* for Republicans.



Democrats / Republicans

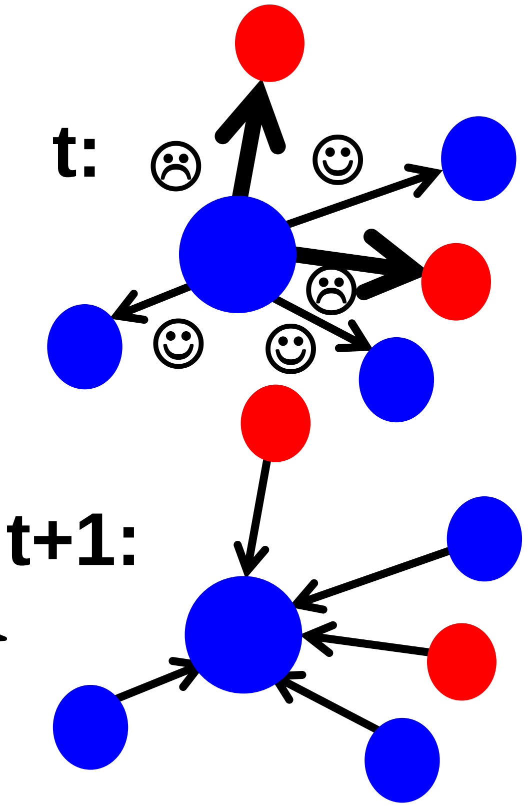
Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (**Democrat**) with g good and b bad out-edges on a **day t** (= will be c

What happens in round t+2?

Degree of citizen ... g+b.

On day *t*+1, *g* friends of *c* root for the Democrats, and *b* friends root for the Republicans. And in evening of *t*+1, *c* will receives *g* recommendations for Democrats, and *b* for Republicans.



Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

If g>b at t, then v will vote in time t+2 as in time t again, otherwise the opposite!

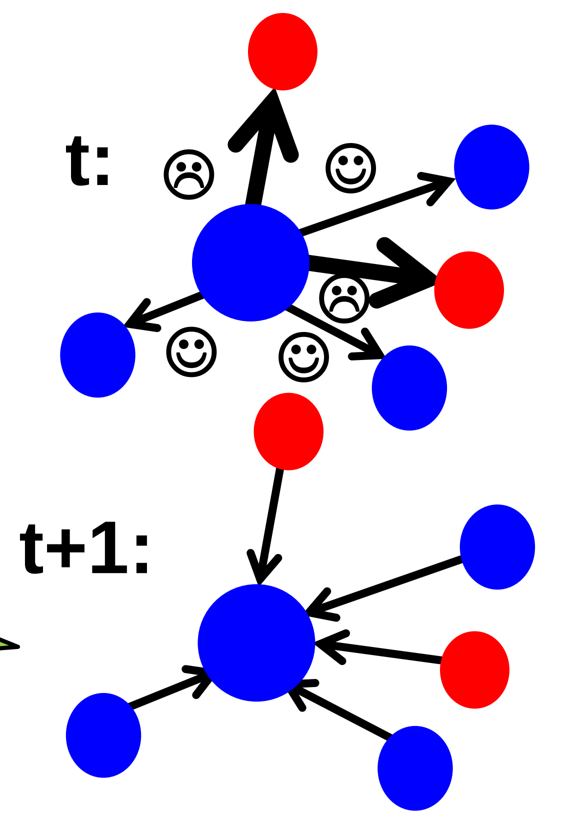
- good

and b bad out-edges on a **day t** (= will be c

What happens in round t+2?

Degree of citizen ... g+b.

On day *t*+1, *g* friends of *c* root for the Democrats, and *b* friends root for the Republicans. And in evening of *t*+1, *c* will receives *g* recommendations for Democrats, and *b* for Republicans.



Democrat

Eventually each

every other day

In other words, if it was g<b, the number of bad edges incident will reduce at time t+2: v changes the party!

If g>b at t, then v will vote in time t+2 as in time t again, otherwise the opposite!

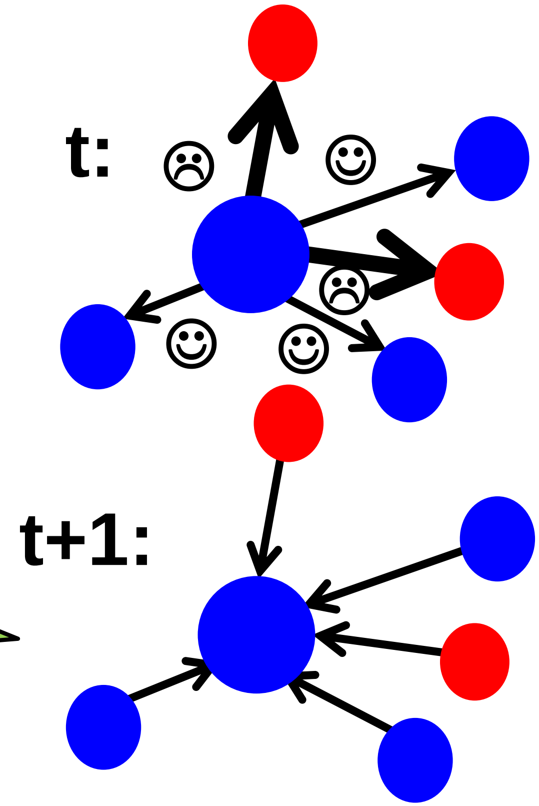
- good

and b bad out-edges on a **day t** (= will be c

What happens in round t+2?

Degree of citizen g+b.

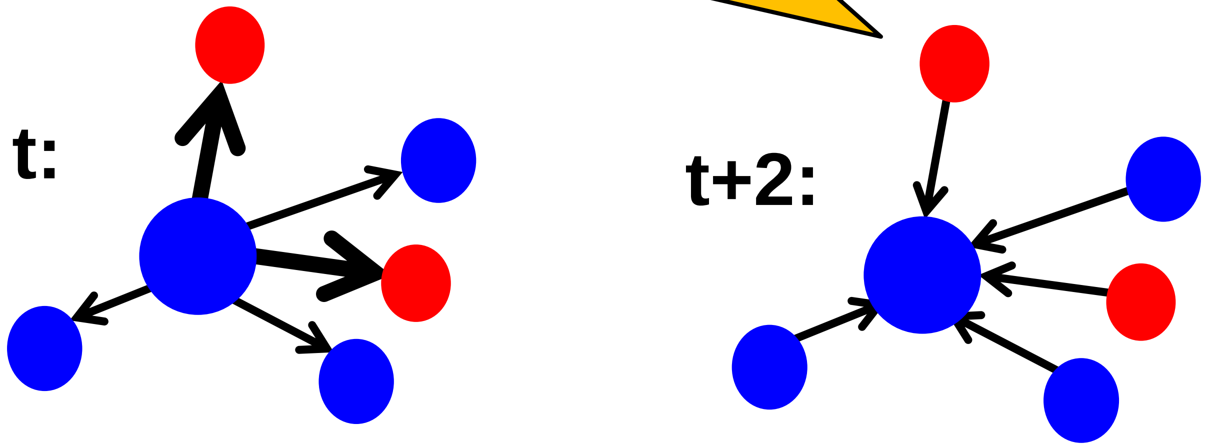
On day *t*+1, *g* friends of *c* root for the Democrats, and *b* friends root for the Republicans. And in evening of *t*+1, *c* will receives *g* recommendations for Democrats, and *b* for Republicans.



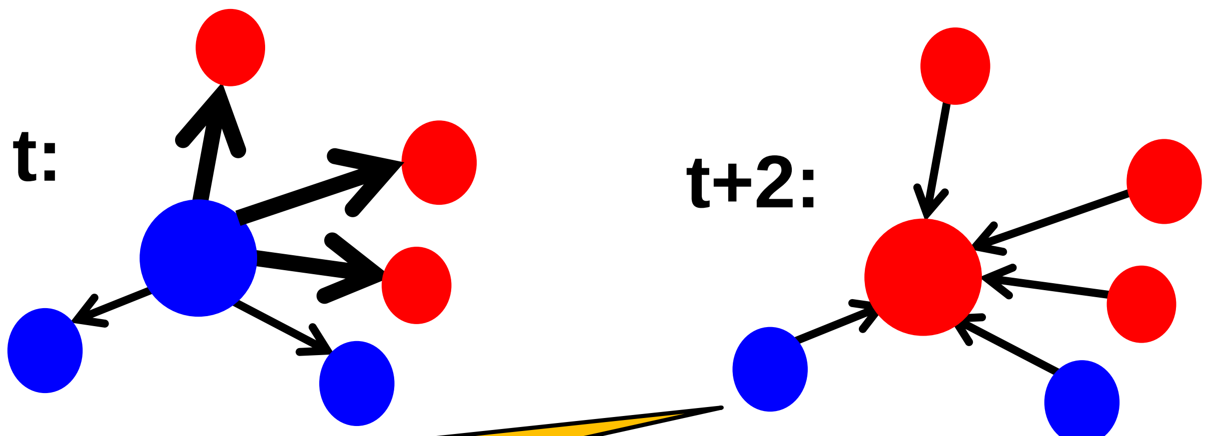
Example:

The number of bad edges stays the same!

g>b: stay in same party



b>g: change to opposite party

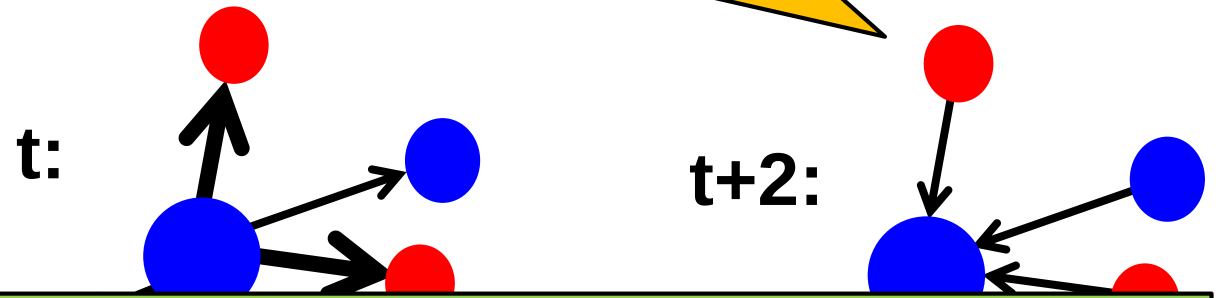


The number of bad edges decreases!

Example:

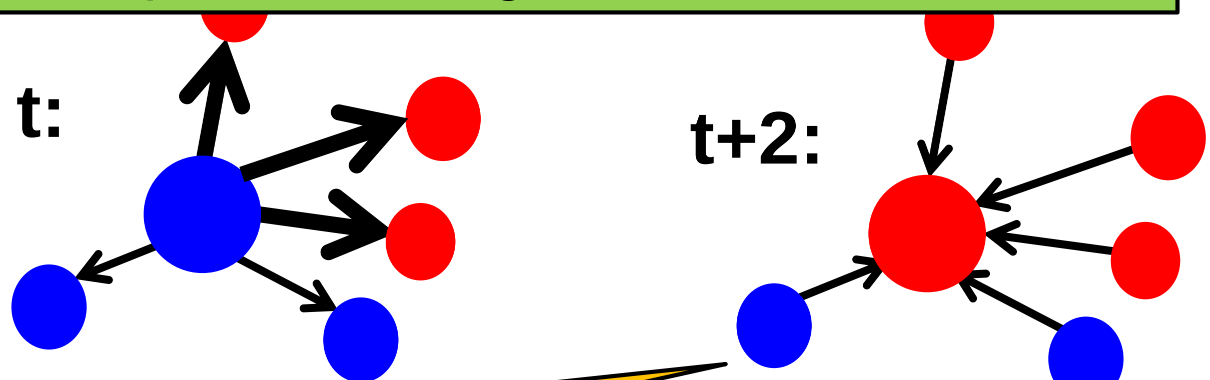
The number of bad edges stays the same!

g>b: stay in same party



The number of bad edges does never increase. Thus, it will at some point converge to a certain value. From then on, user will vote for the same party every second day. A complex “convergence”!

b>g: change to opposite party



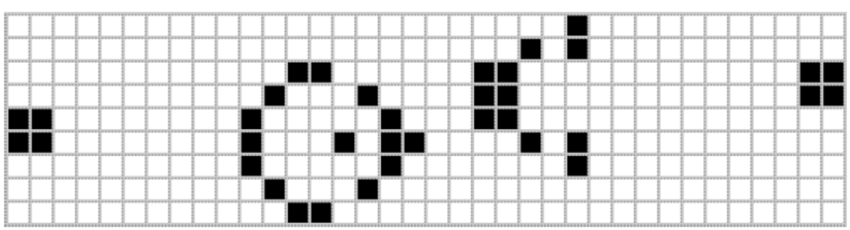
The number of bad edges decreases!

Related to Conway's Game of Life

Dynamic behavior and "convergence" not well understood.

- Turing-complete game: LIFE
- 2d cell grid, each cell *dead* or *alive*
- Every cell interacts with its eight neighbors:
  - Any live cell with fewer than two live neighbors dies (loneliness).
  - Any live cell with more than three live neighbors dies, as if by overcrowding.
  - Any live cell with >2 live neighbors lives on to the next generation.

Can model complex behavior: gun + glider:



End of Lecture

Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

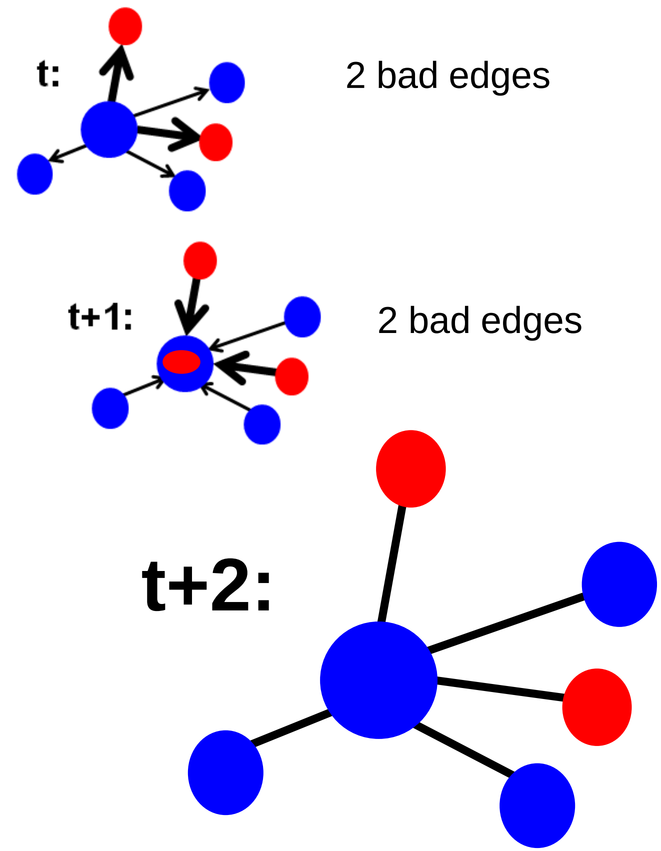
Continued…: **if g>b**

- At day t+1, (blue) citizen c:
    - g>b neighbors blue, b red

- So citizen c will be blue (still/again) at t+2

- So b (red) neighbors pointing to c are bad at t+1 (from neighbor's perspective), since c will be blue at t+2.

Bad out-edges of c at time t will be bad edges to c at time t+1! Total number of bad edges remains the same. (No matter what color of c is at time t+1.)



Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

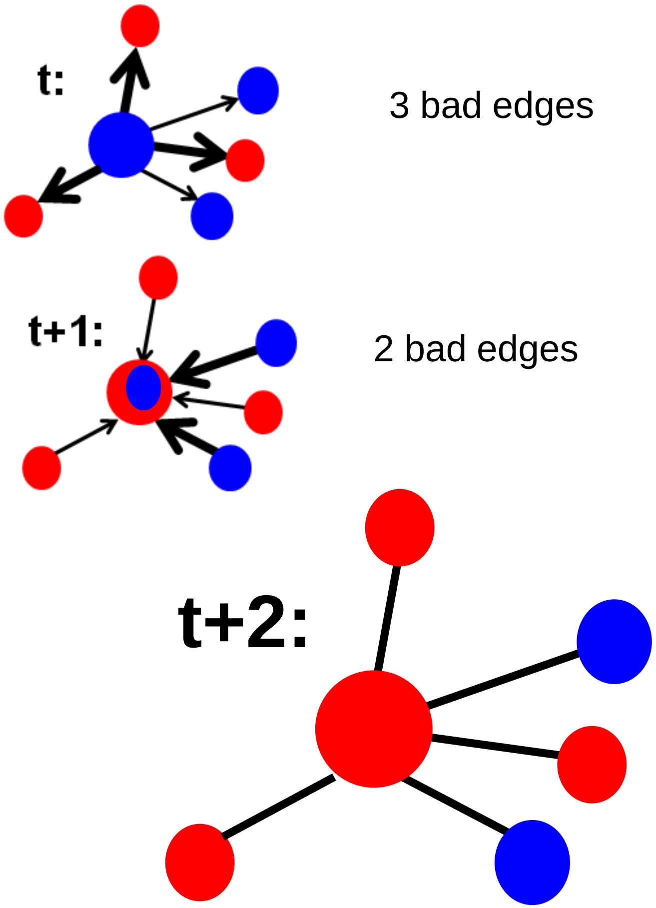
Continued…: **if b>g**

- At day t+1, (blue) citizen c:
  - b>g neighbors blue, g red

- So citizen c will be red at t+2

- So g<b (blue) neighbors pointing to c are bad at t+1 (from neighbor's perspective), since c will be red at t+2 .

Bad out-edges of c at time t will be good edges to c at time t+1! Total number of bad edges decreases. (No matter what color of c is at time t+1.)



Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Continued:

- In both cases, the number of bad edges does not increase.
- In fact, it decreases if any node switches the party.
- Since the number of bad edges cannot be negative, the system will stabilize for a certain number of bad edges.
- Once number of bad edges stabilized, each node either stabilizes to a party or switches back and forth between times t and t+2.

QED

Discussion

- How to do it for randomized algorithms?
  - Do not know k, the number of rounds!
  - But can just simulate more rounds, no problem.
  - Careful about adversary: should not compromise randomness of choices (e.g., have nodes produce random bits until it's what he wanted)
  - Problem: can also not just stick to given random choices once and forever! Adversary may have corrupted the variables before.

- Some additional memory overhead, but usually bearable.
  - Memory overhead depends on k, the number of rounds, which is low.

- Good for mobile environments: if k-neighborhood does not change, nothing changes