

# **STABILIZATION & LOCALITY**

**Shay Kutten**  
**Technion, Israel**

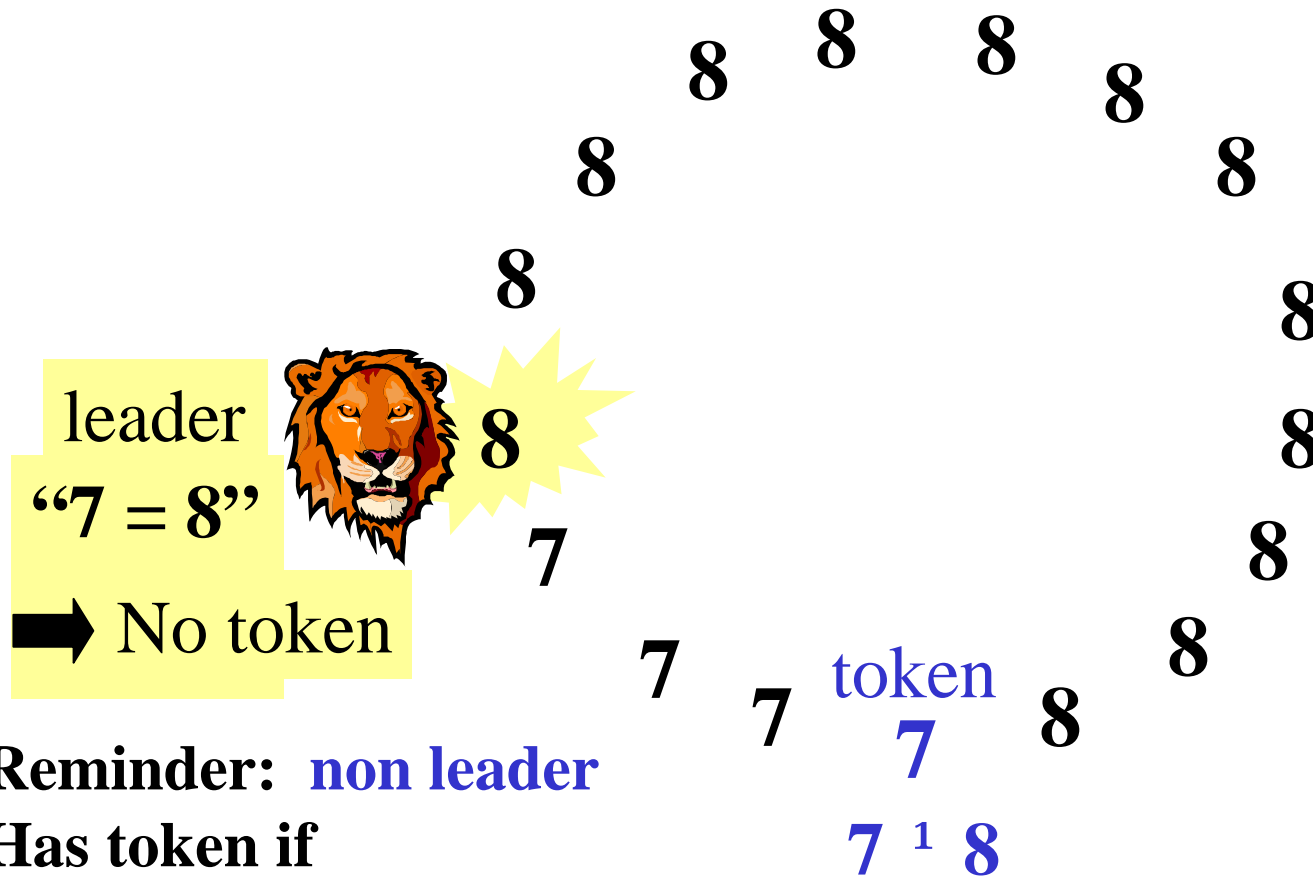
# Recall: Traditional methods are **global**

\* **Dijkstra- 1 fault-  $O(n)$  Time;  $f$  faults-  $O(fn)$  time**

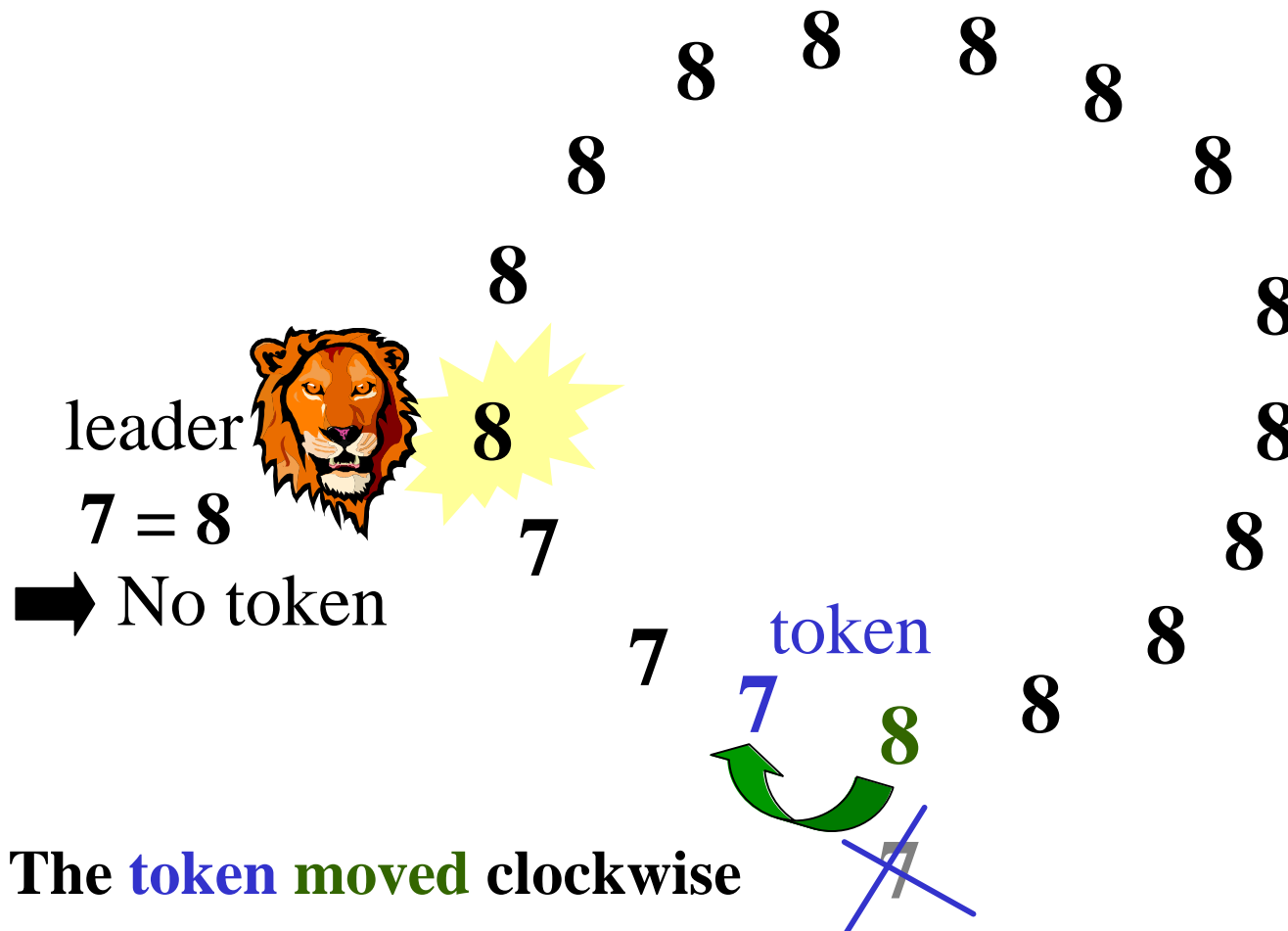
**(time = length of shortest causal chain)**

**(Lamport's "happened before")**

# Global effect example (Dijkstra's algorithm)

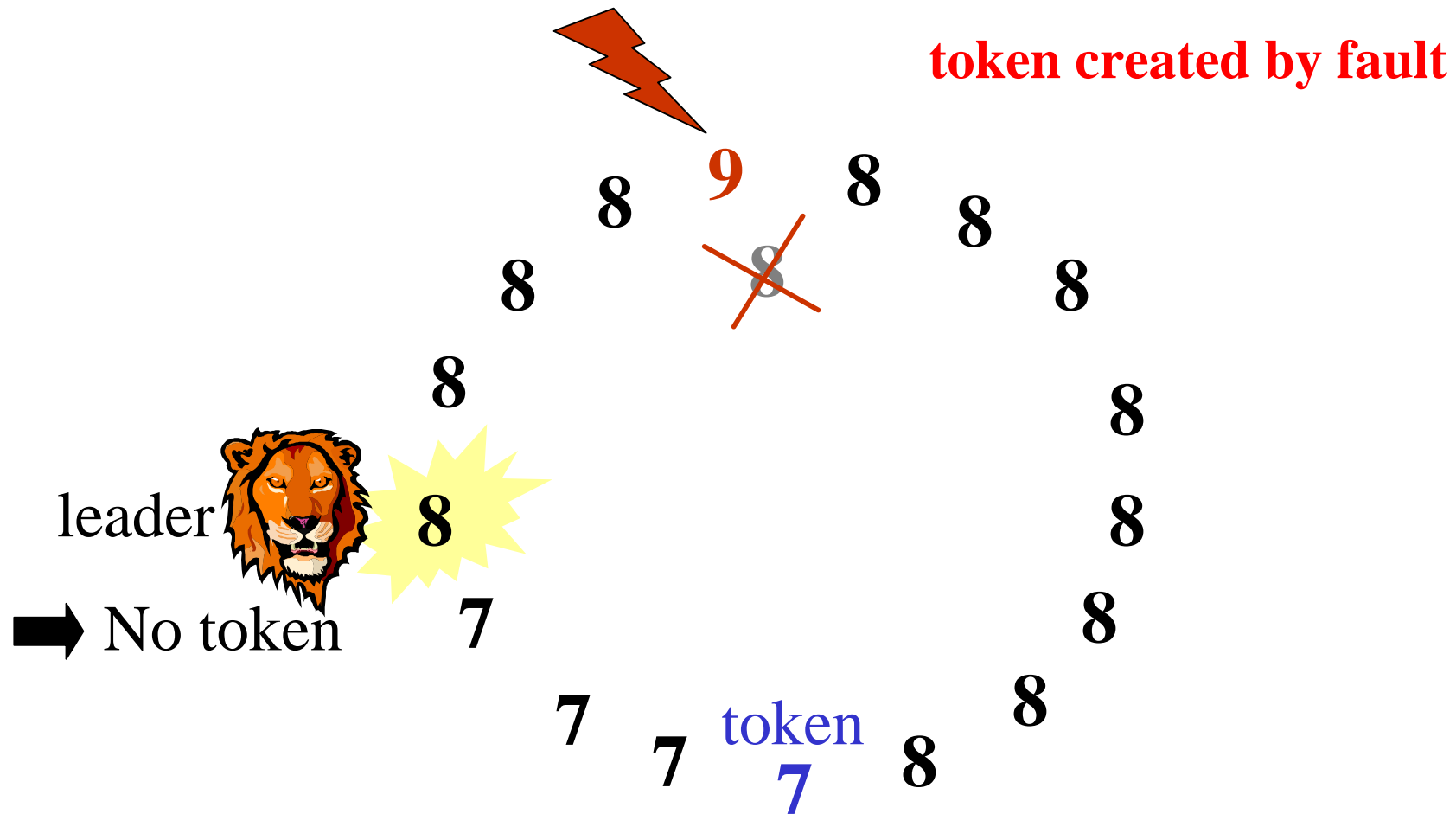


# Global effect example (1. No fault)





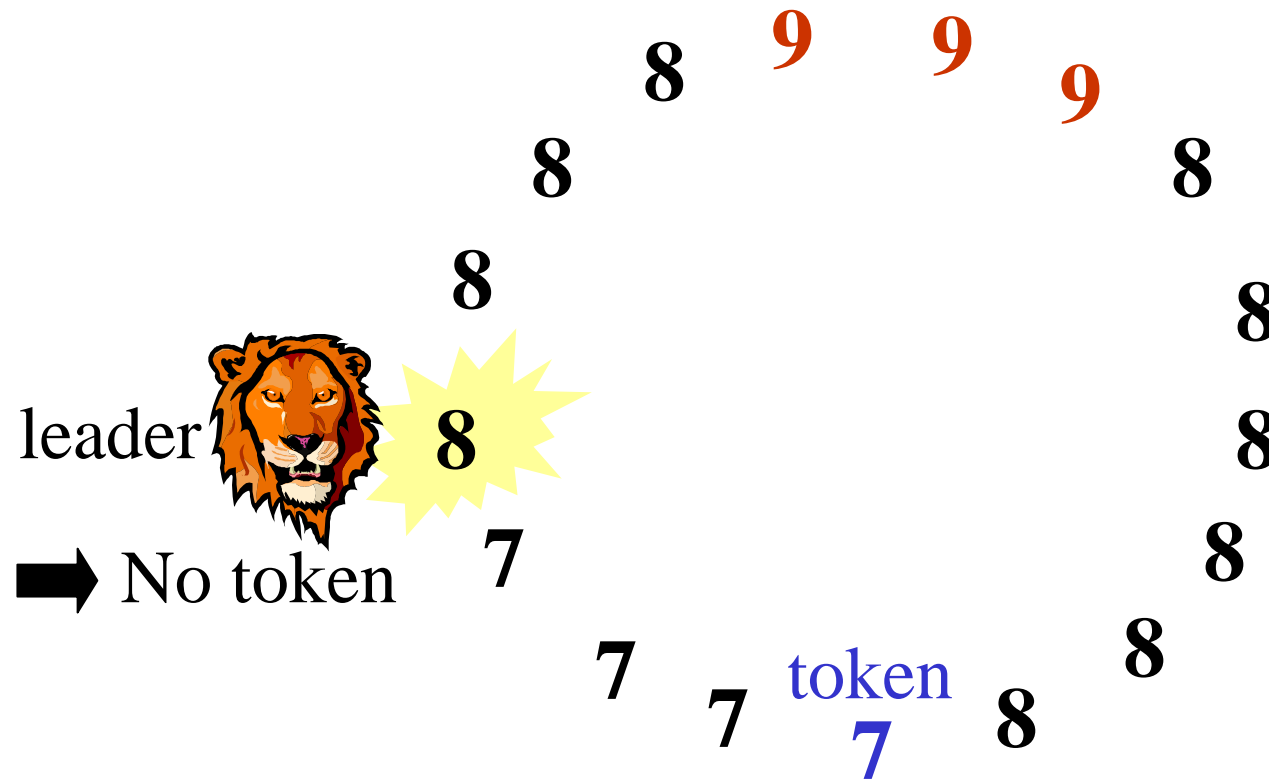
# Global effect example (2. Fault concurred)



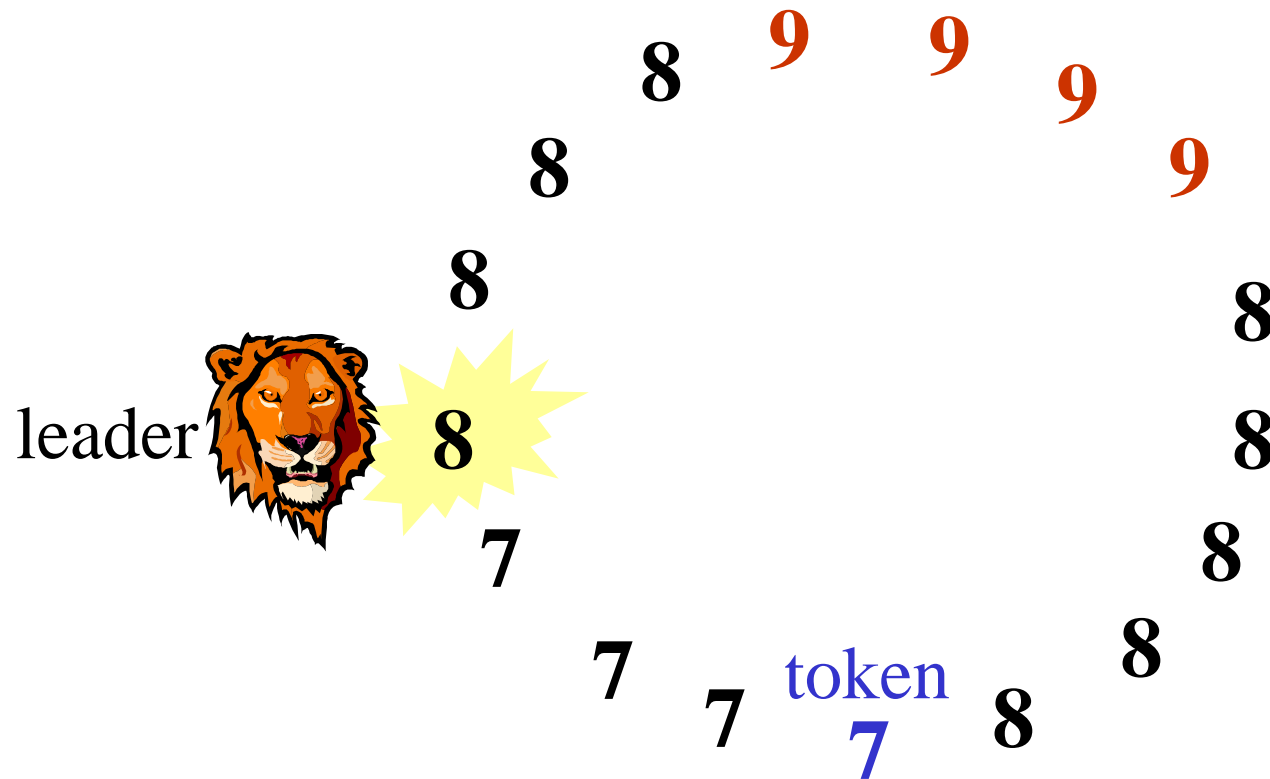


# Global effect example

The one fault expanding now  
creates the equivalent of 3 faults

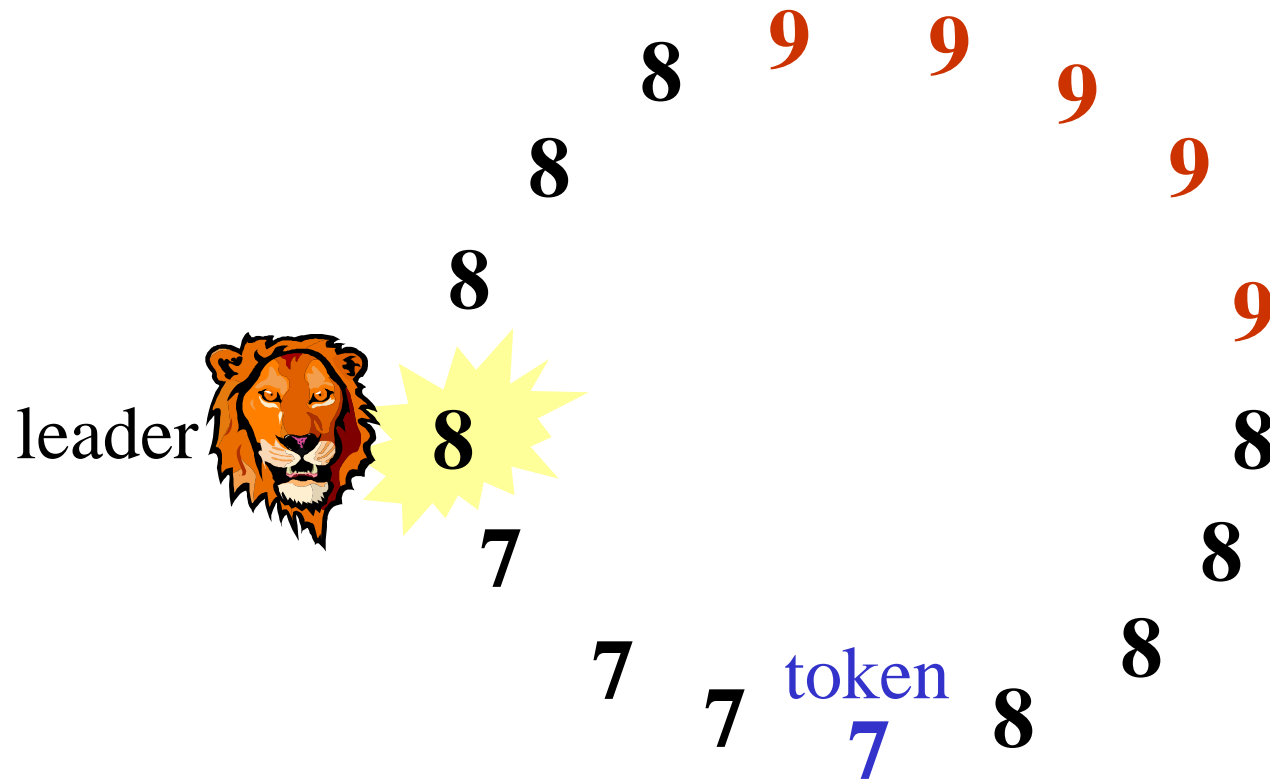


# Global effect example



# Global effect sample

*The effect of one fault may circle whole ring*



# Recall: Traditional methods are **global**

\* [Katz, Perry]- 1<sup>st</sup> general method, **1 (or even 0) faults-  $q(n)$  time**

- (1) self stab bcast freezes **all** nodes
- (2) **global** self-stab snapshot to a leaser
- (3) leader checks **global** state.
- (4) Leader initializes **every** node (if fault)
- (5) **bcast** (i.e. **broadcast**) unfreezes nodes

# Recall: Traditional methods are global

\* Global reset (general) methods, **1 fault-  $O(n)$  time**

[Afek, Kutten, Yung],

[Awerbuch, Patt-Shamir, Varghese],

[Awerbuch, K., Mansour, Patt-Shamir, Varghese]

(1) bcast freezes **all** nodes

(2) reset **every** node to a specific initial state

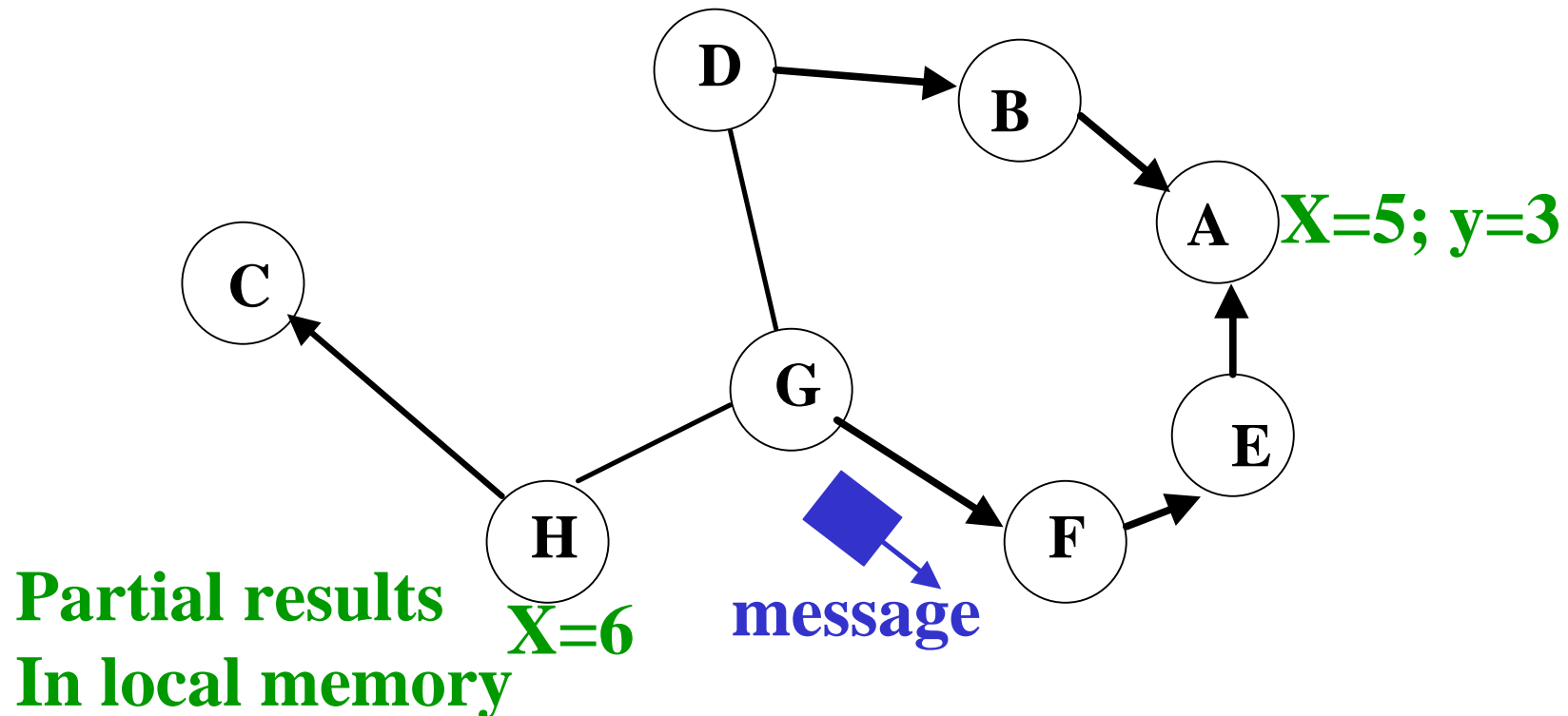
[Dolev, Herman] superstabilizing **global** reset:

(2') reset **every** node to a state “nearest” to

current one (still **1 fault-  $W(n)$  time**)

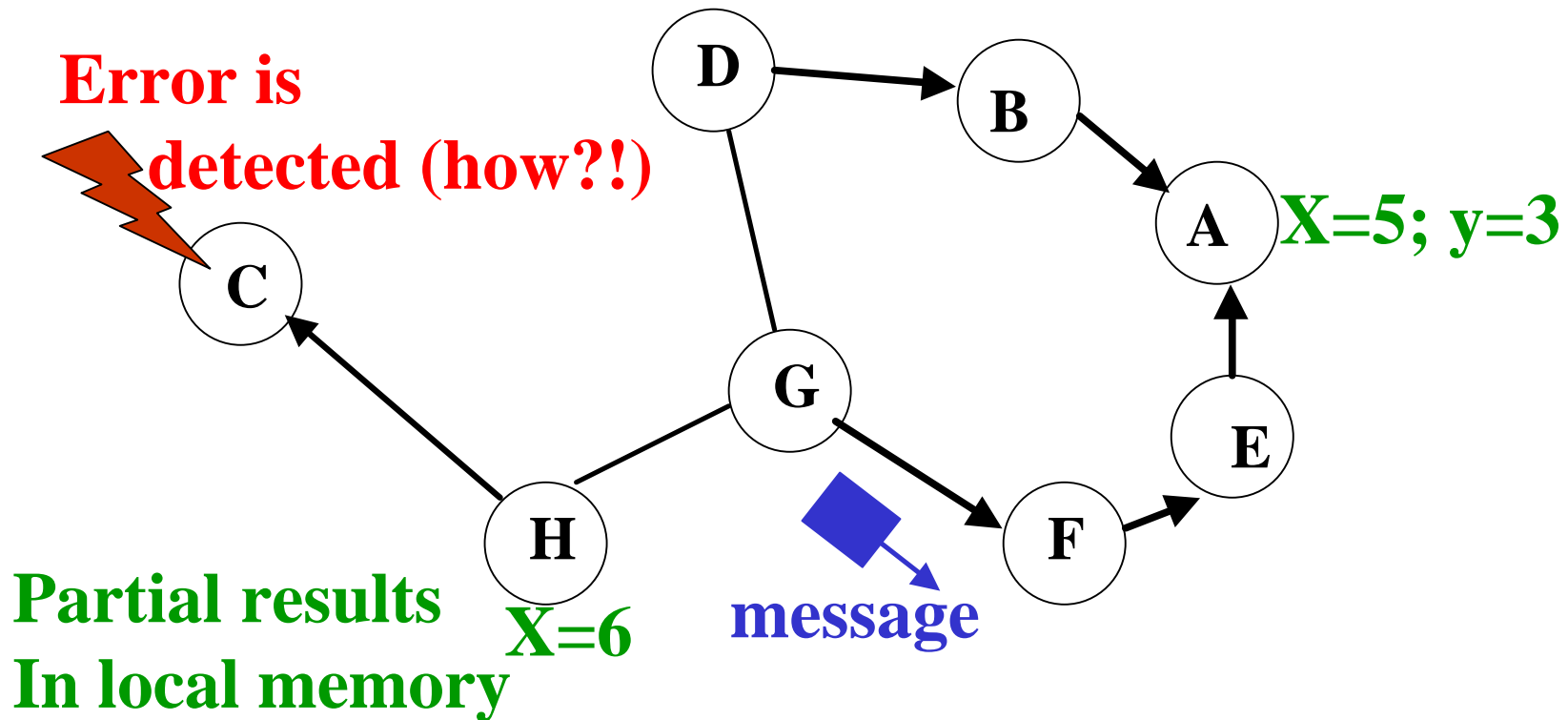
# Recall: Traditional methods are **global** (what is “reset”?)

**Before** the reset, the algorithm runs, messages are in Transit, partial results in nodes.

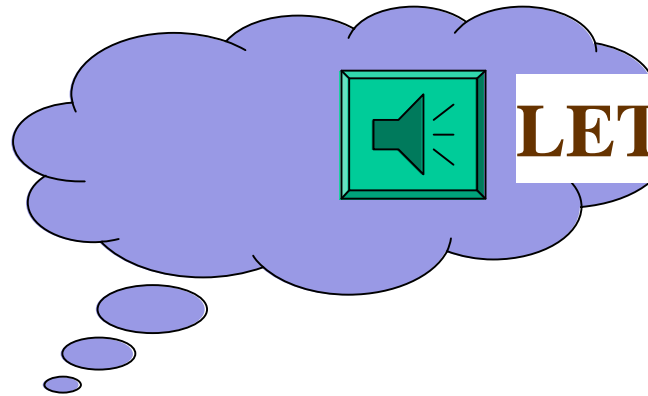


# Recall: Traditional methods are **global** (what is “reset”?)

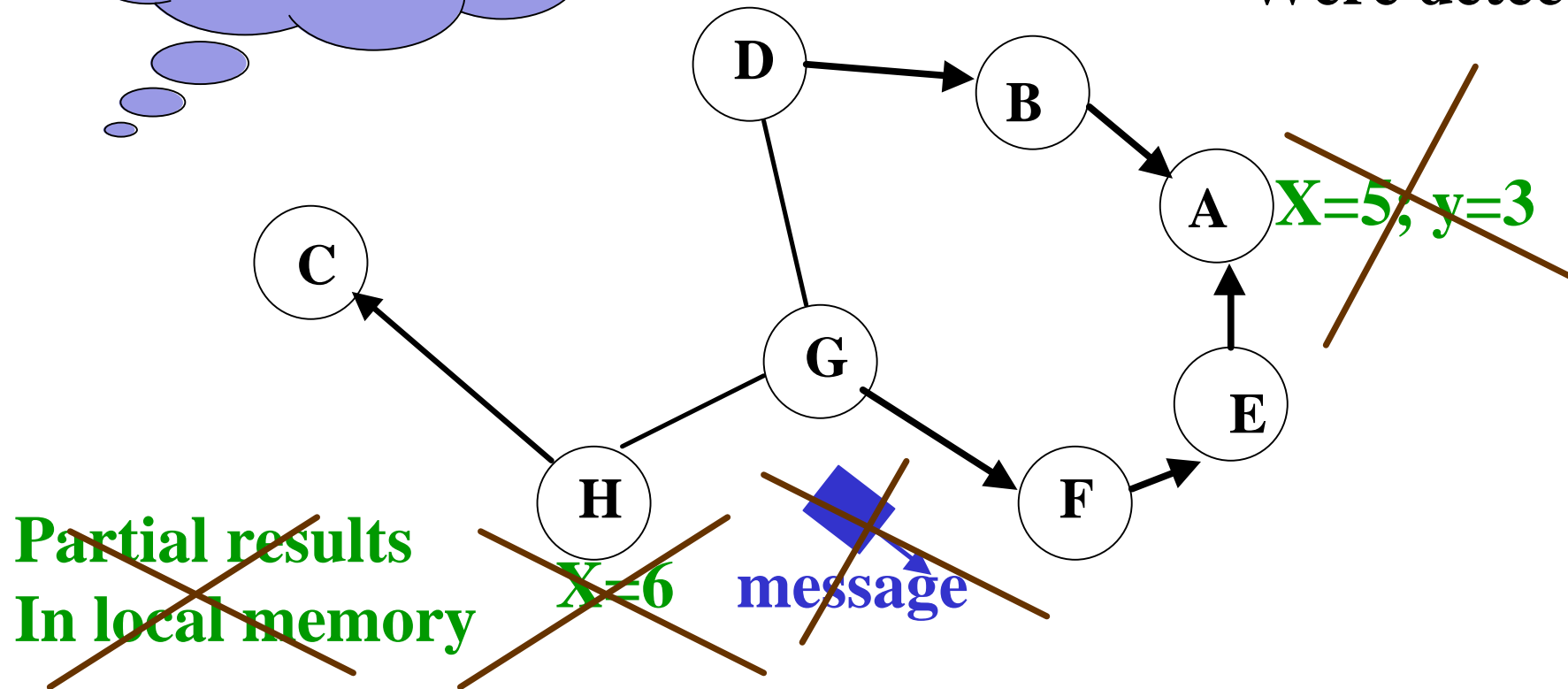
**Before the reset, the algorithm runs, messages are in Transit, partial results in nodes.**



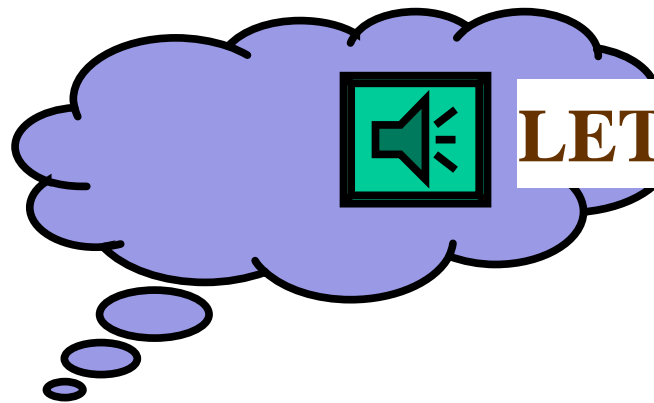
# Recall: Traditional methods are **global** (what is “reset”? cont.)



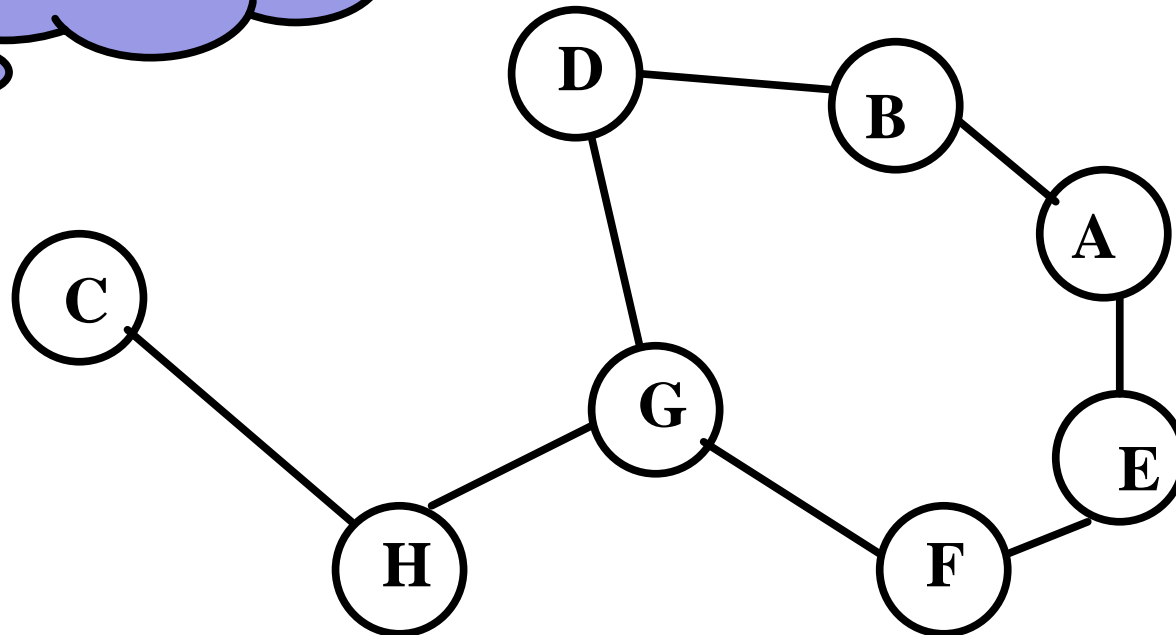
**LET THERE BE RESET** (since faults  
Were detected)



Recall: Traditional methods are global  
(what is “reset”)

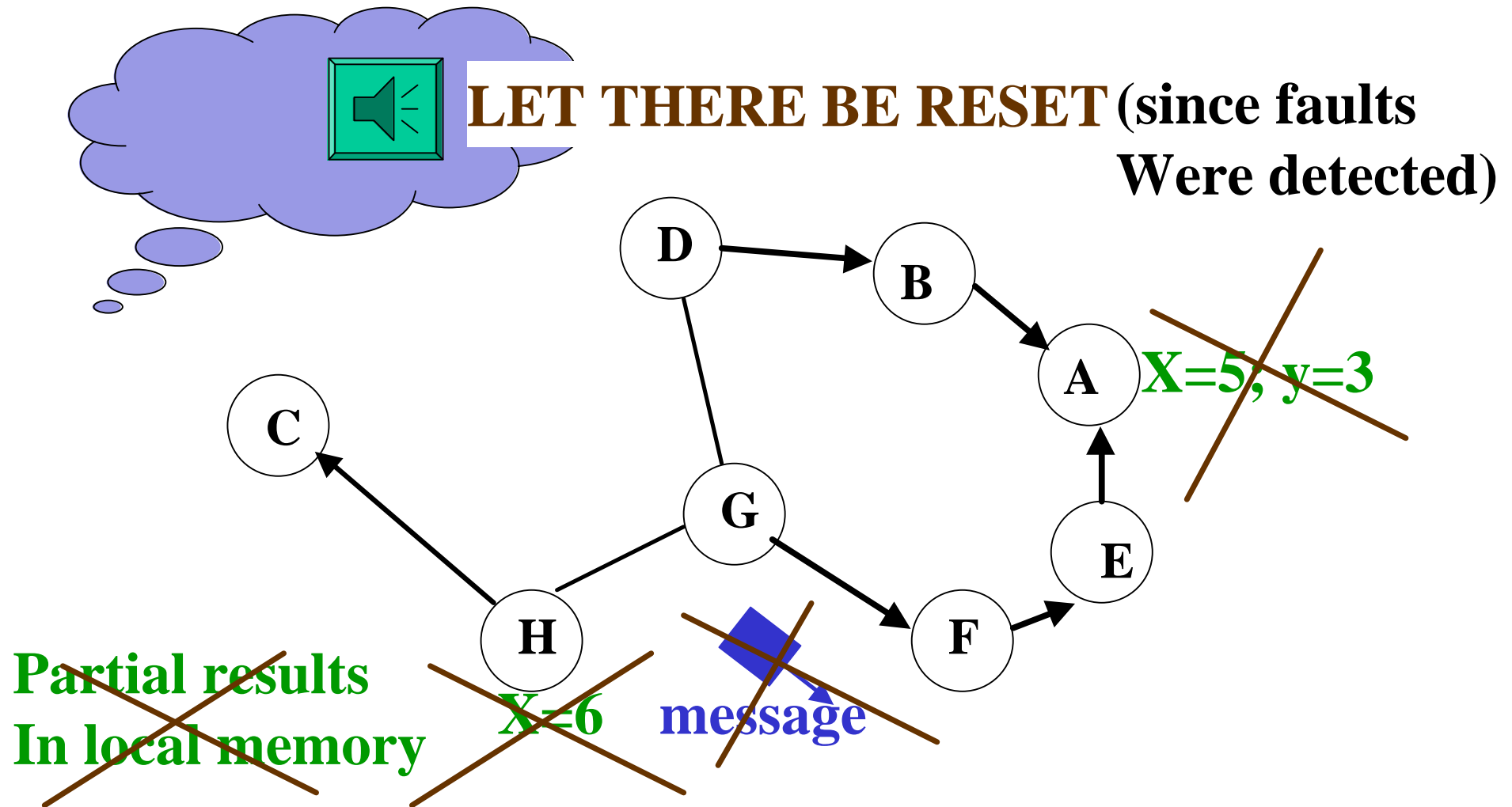


**LET THE ALGORITHM START**

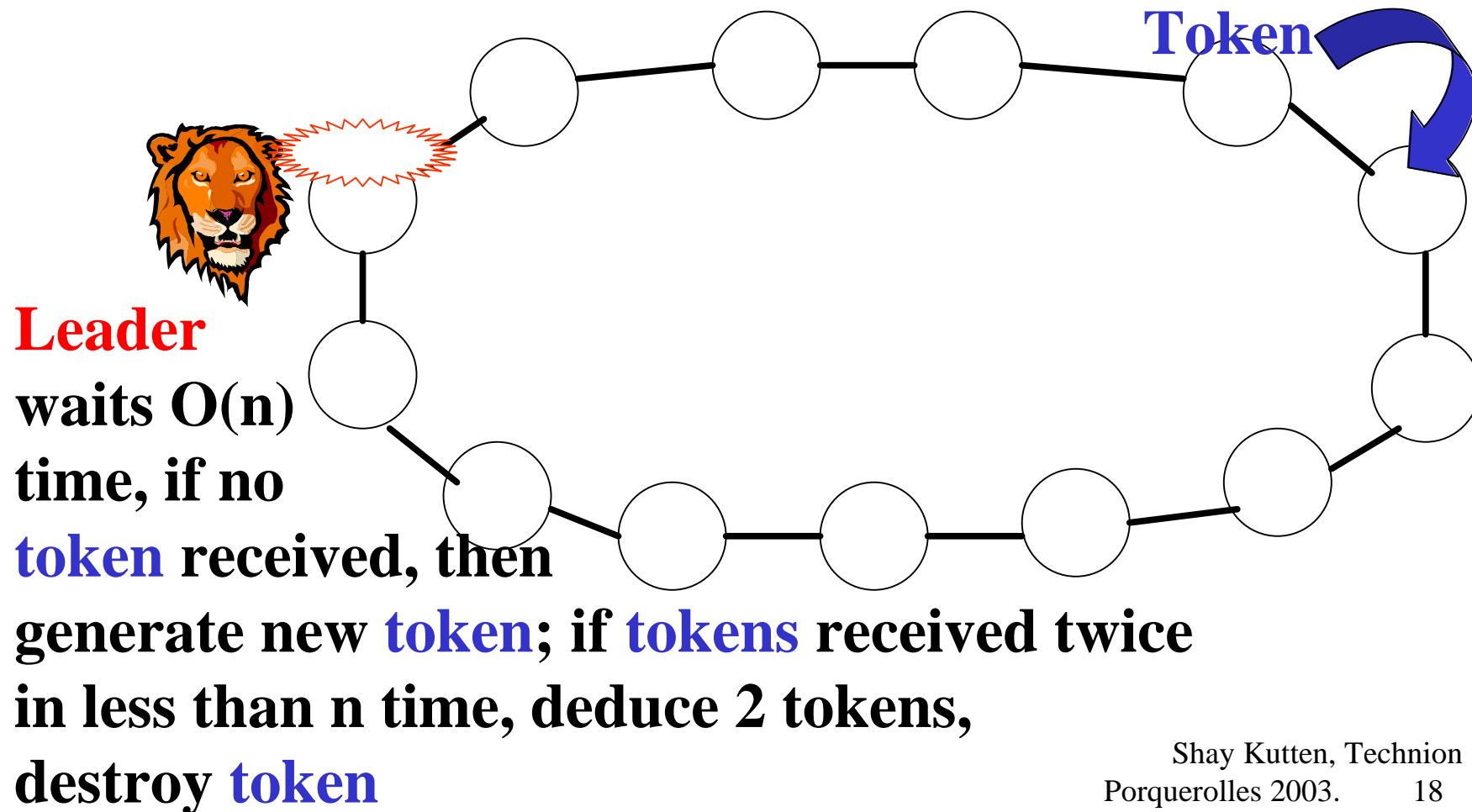


**The reset returned the algorithm to its initial (global) state**

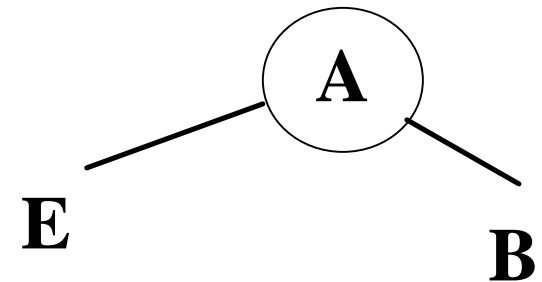
# How is reset performed? (I will show later if I have time...)



# Industrial solution to token self stab is global too

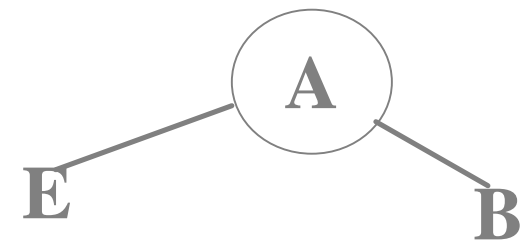
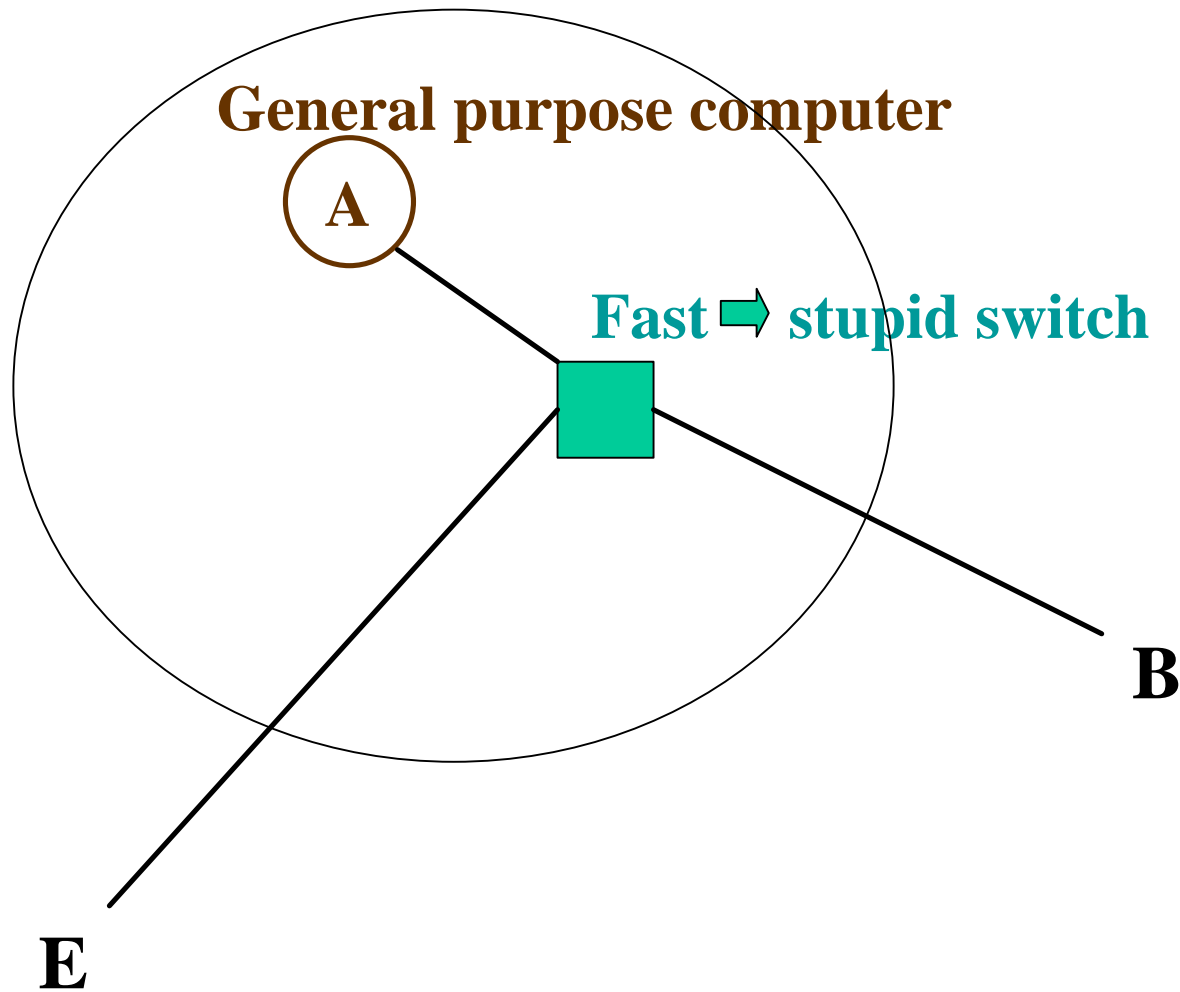


Another example of self stab in industry.  
First, consider inner structure of a node in a  
broadband network

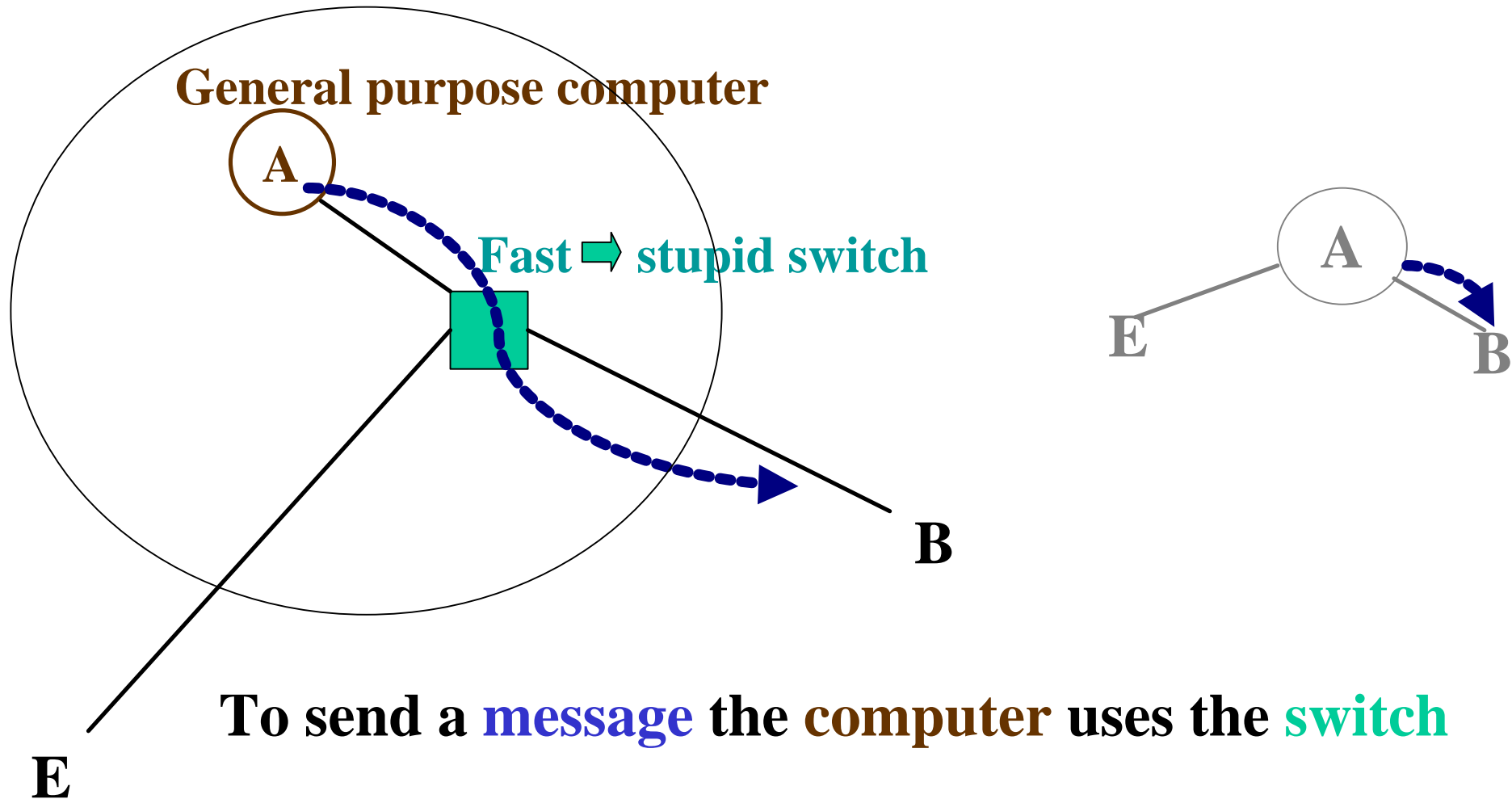


**Outer look at node A and its links to B and C**

# Self stab in industry. First, consider inner structure of a node in a broadband network

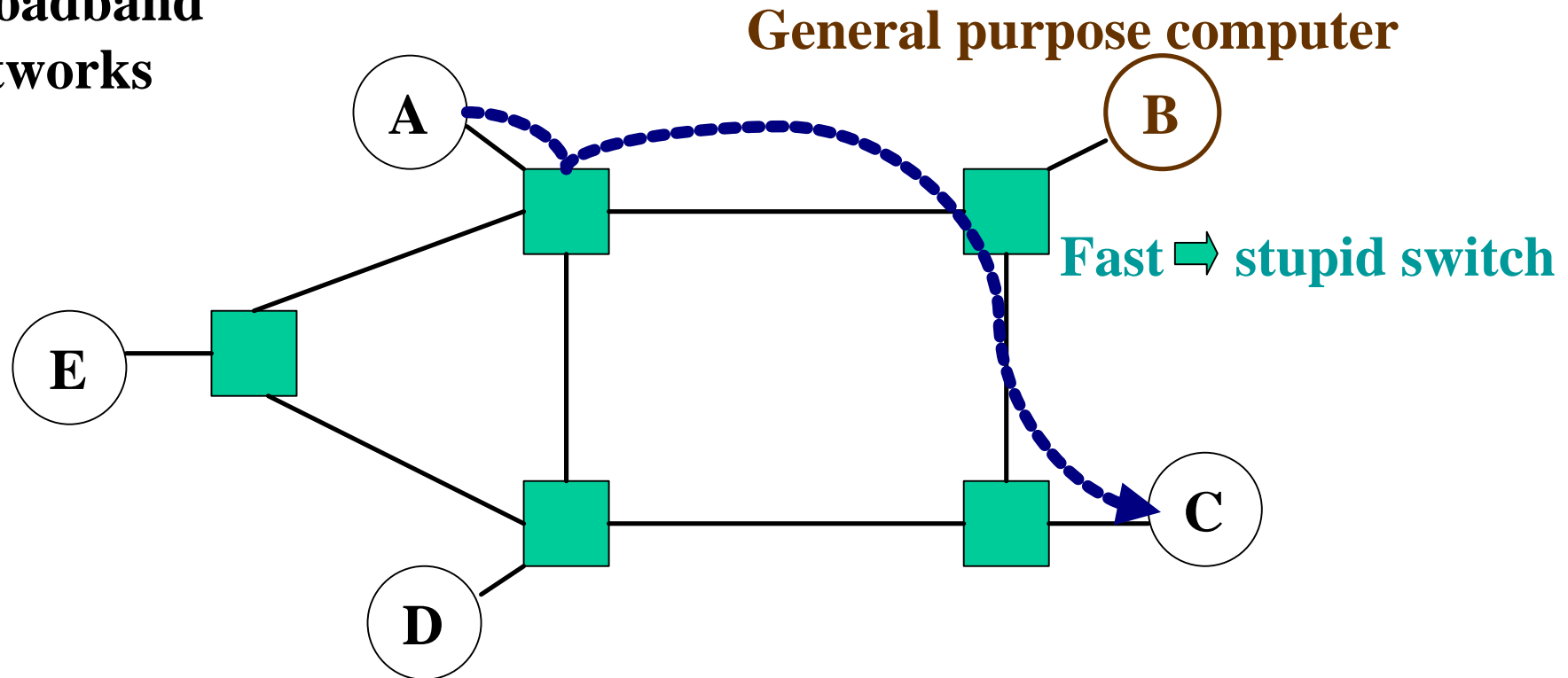


# Self stab in industry. First, consider inner structure of a node in a broadband network



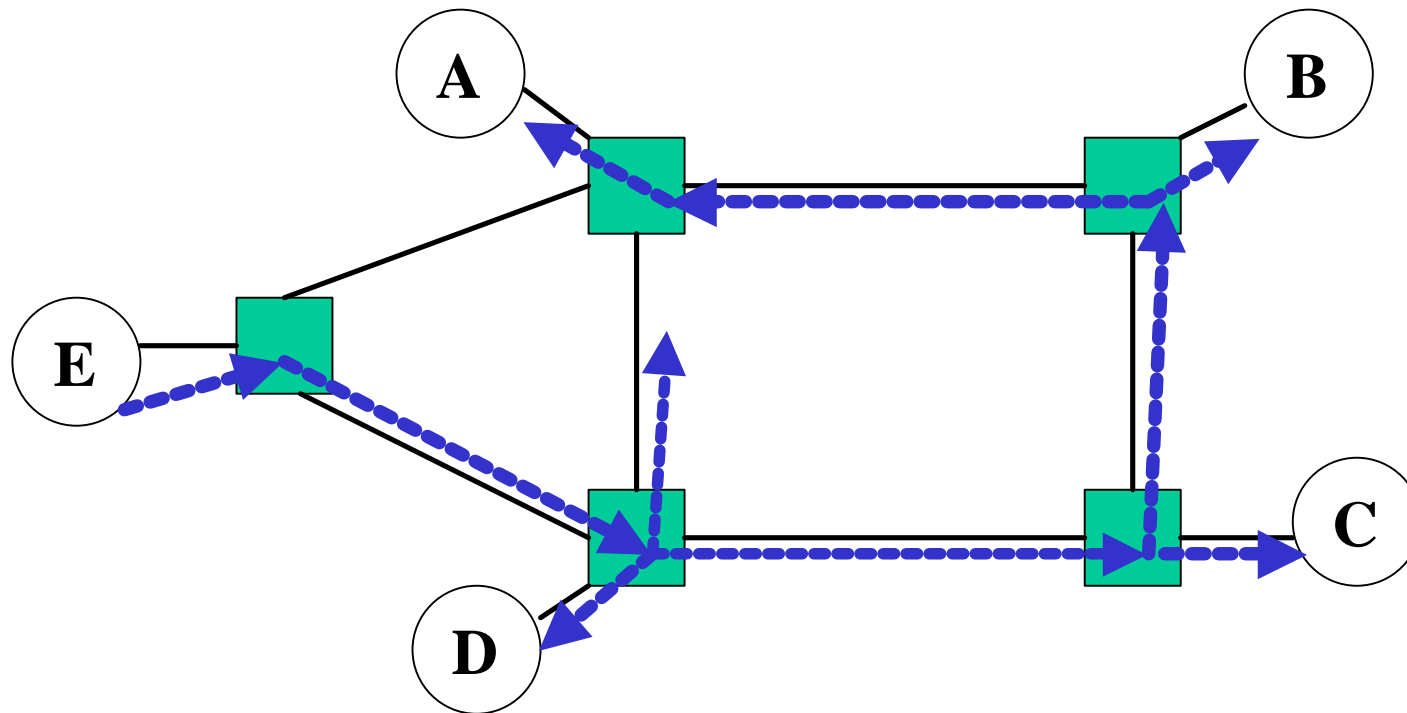
# (cont.) example of self stab in industry

**Broadband  
networks**



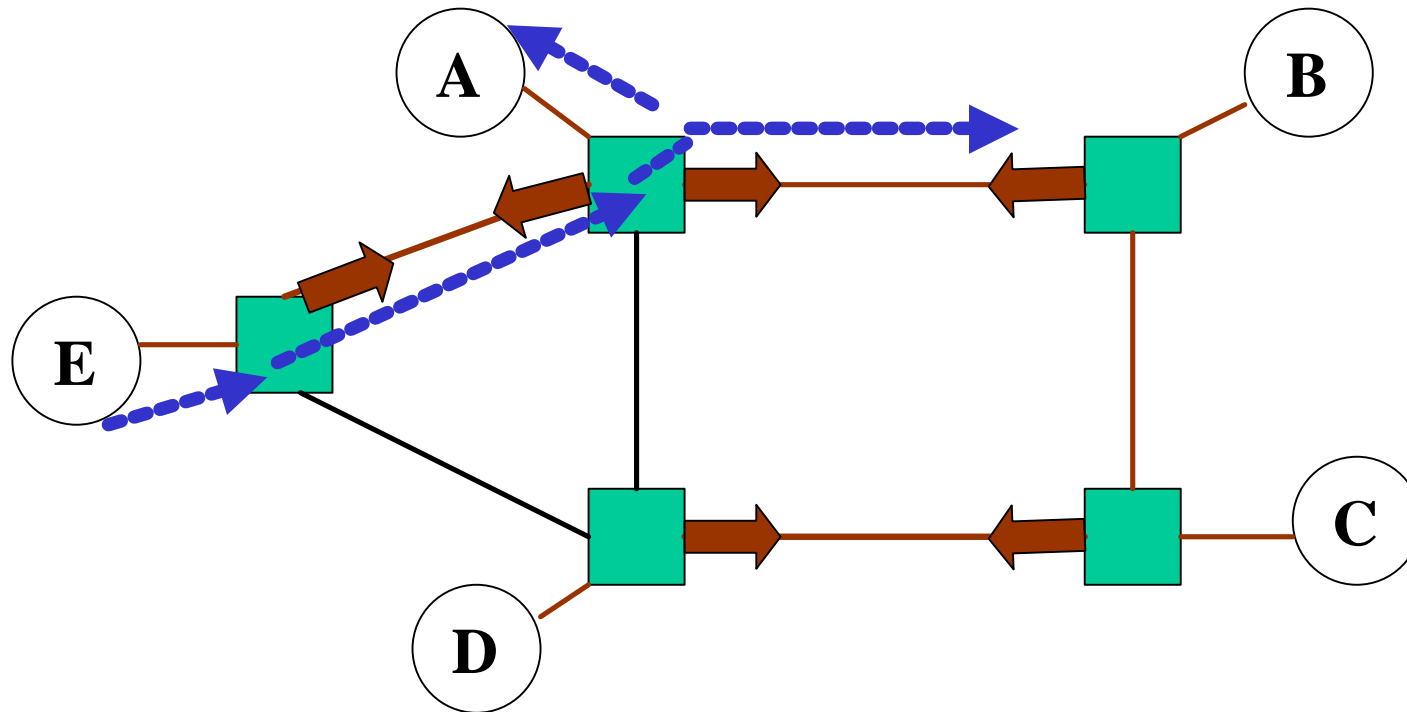
**Fast route** from A to C, passing only B's stupid switch, not B's general purpose computer. Possible since **route** is preset.

## (cont.) example of self stab in industry



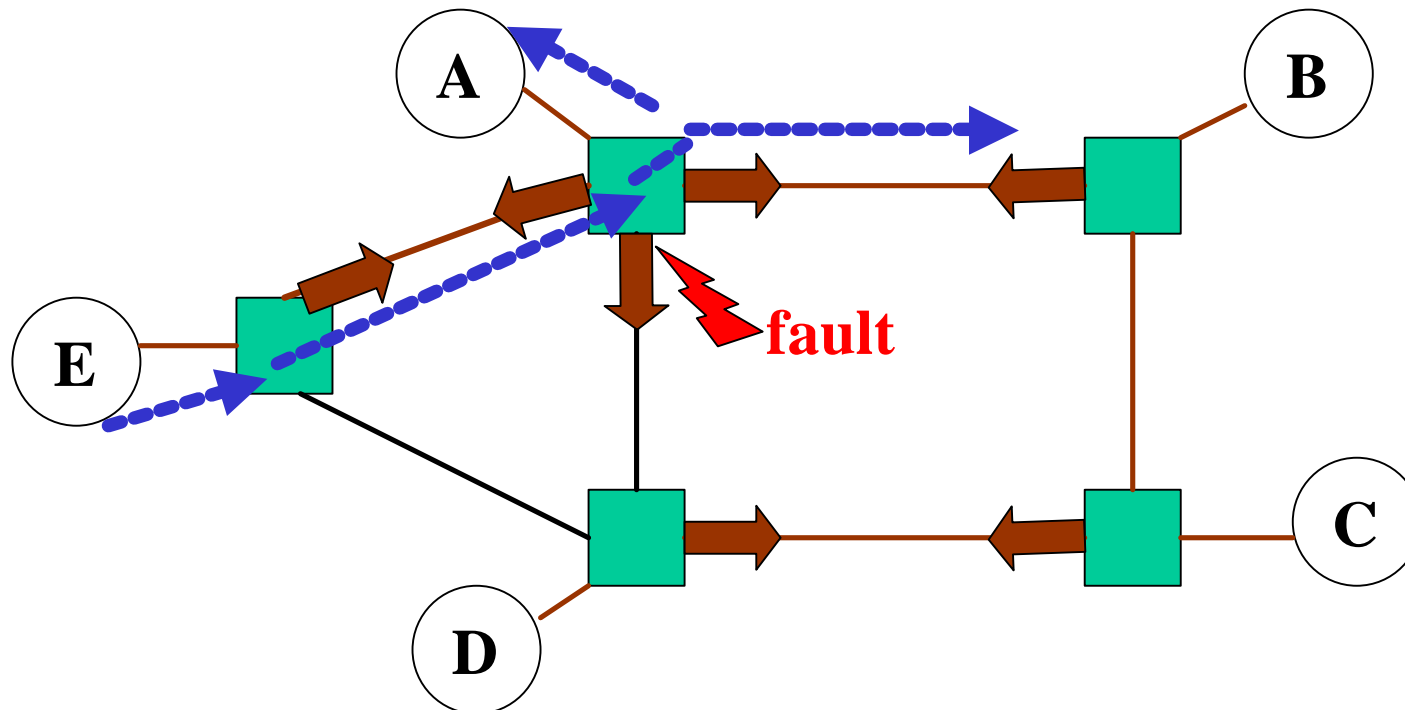
In **bcast**, how does A's **stupid** switch detect that it already received the bcast?? (no complex table consulting)

## (cont.) example of self stab in industry



(non- self stab) solution- stupid switch forwards only over **ports**  
Ports of links marked **tree**

## (cont.) example of self stab in industry

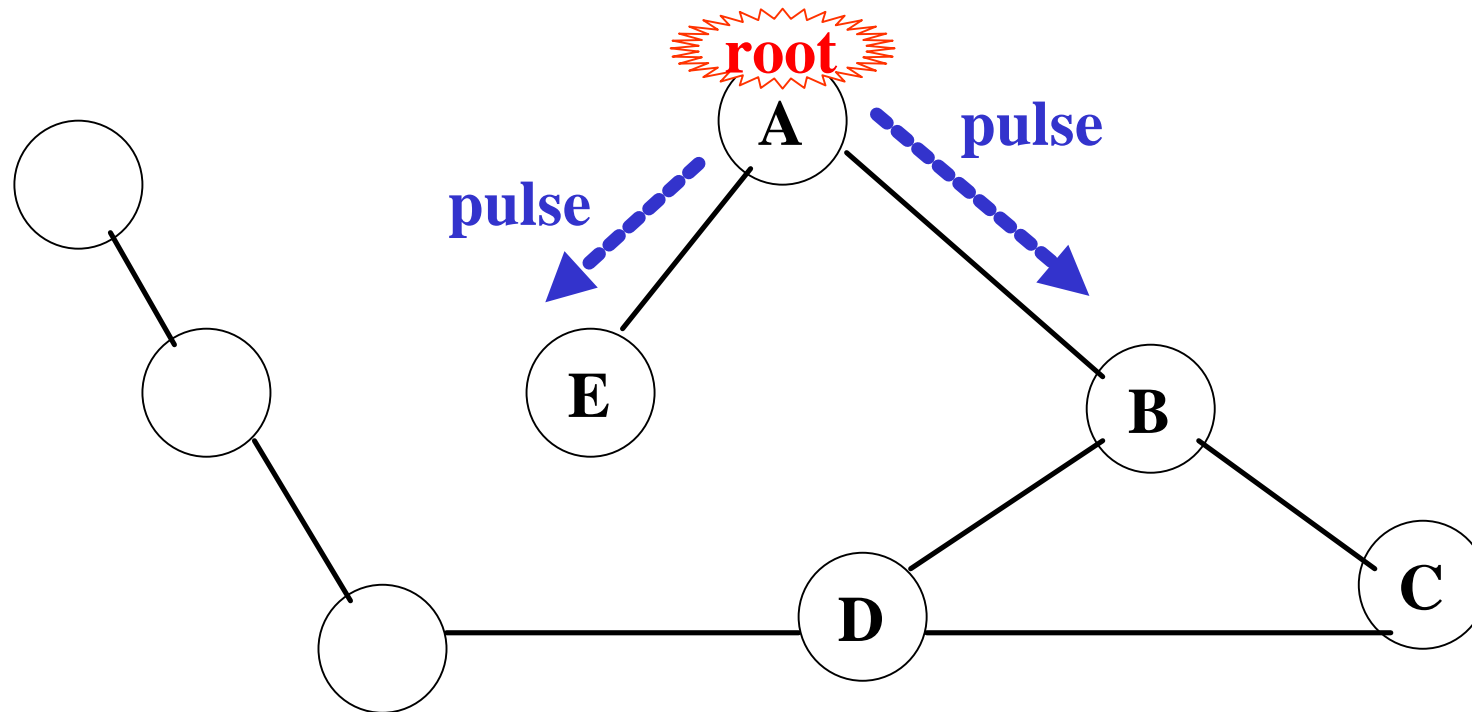


(non- self stab) solution- (stupid) switch forwards only over **ports**  
Ports of links marked **tree**.

Vulnerable to state **fault**: suppose the “**tree**” is really a cycle.

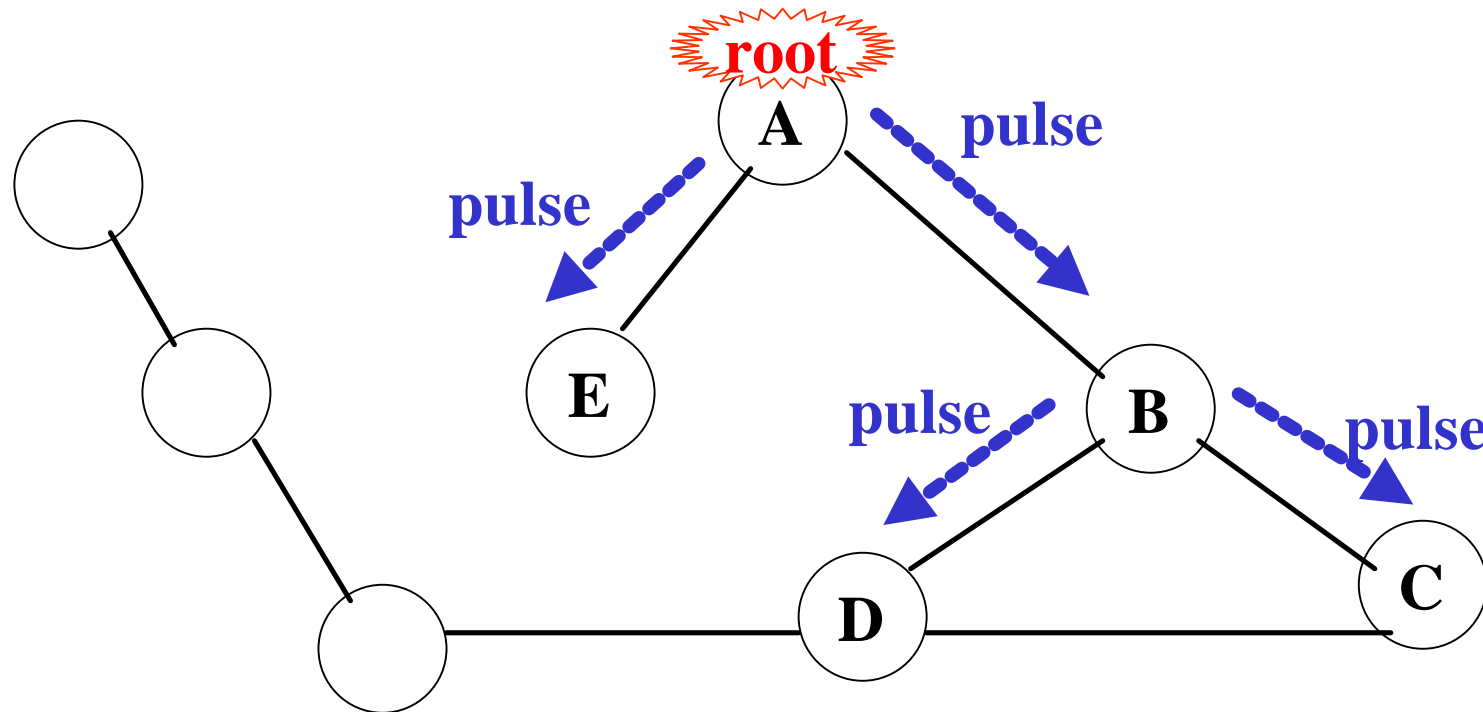
# Industrial solutions to the self stab problem

## (1) Digital's LAN bridges solution



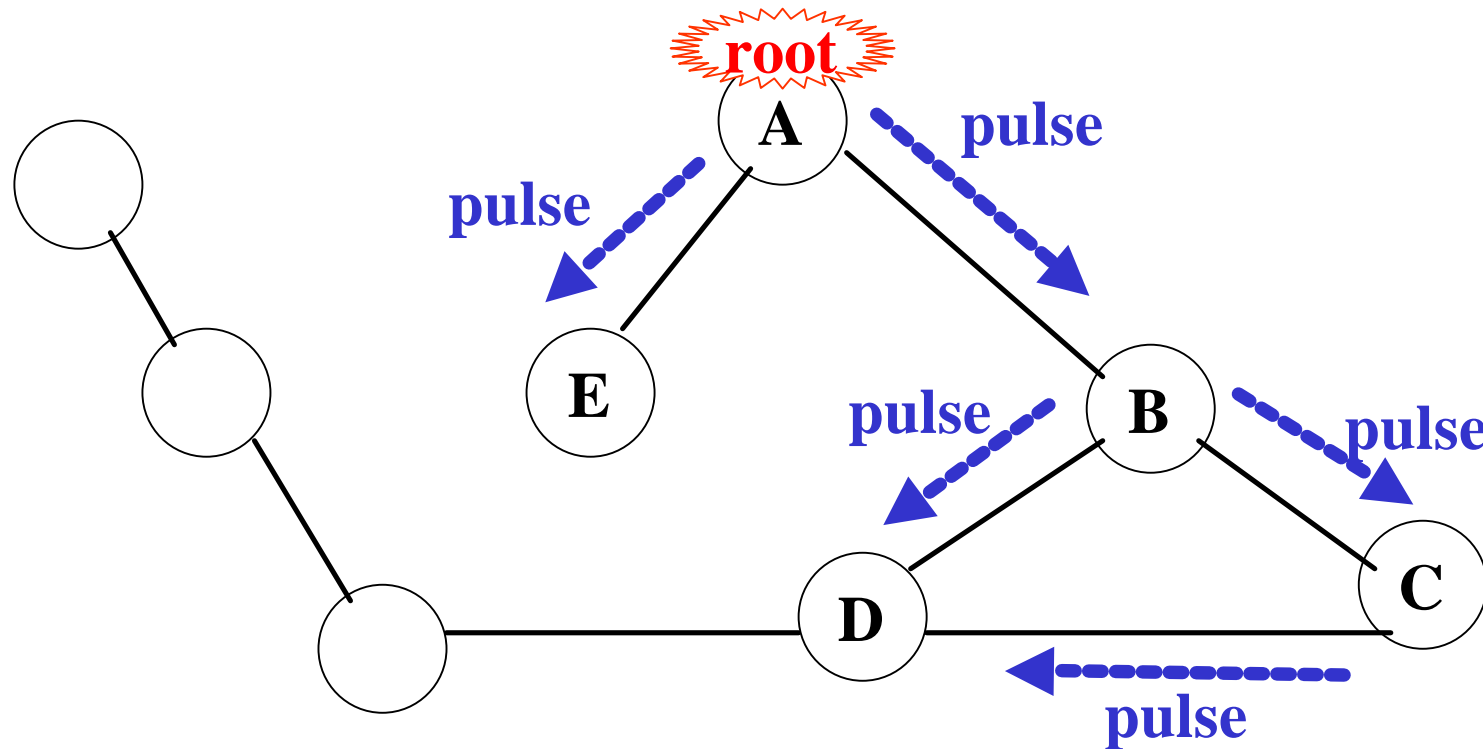
# Industrial solutions to the self stab problem

## (1) Digital's LAN Bridges Solution



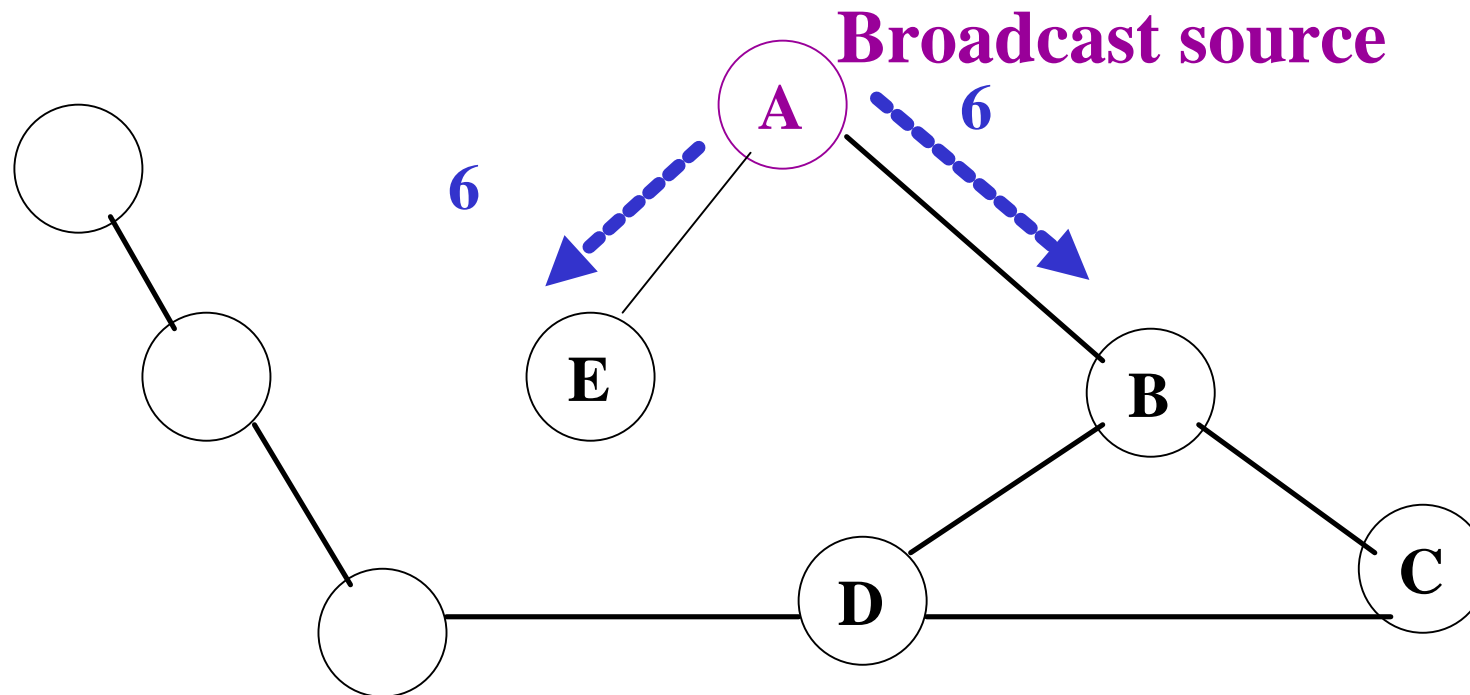
# Industrial solutions to the self stab problem

## (1) Digital's LAN Bridges Solution



# Industrial solutions to the self stab problem

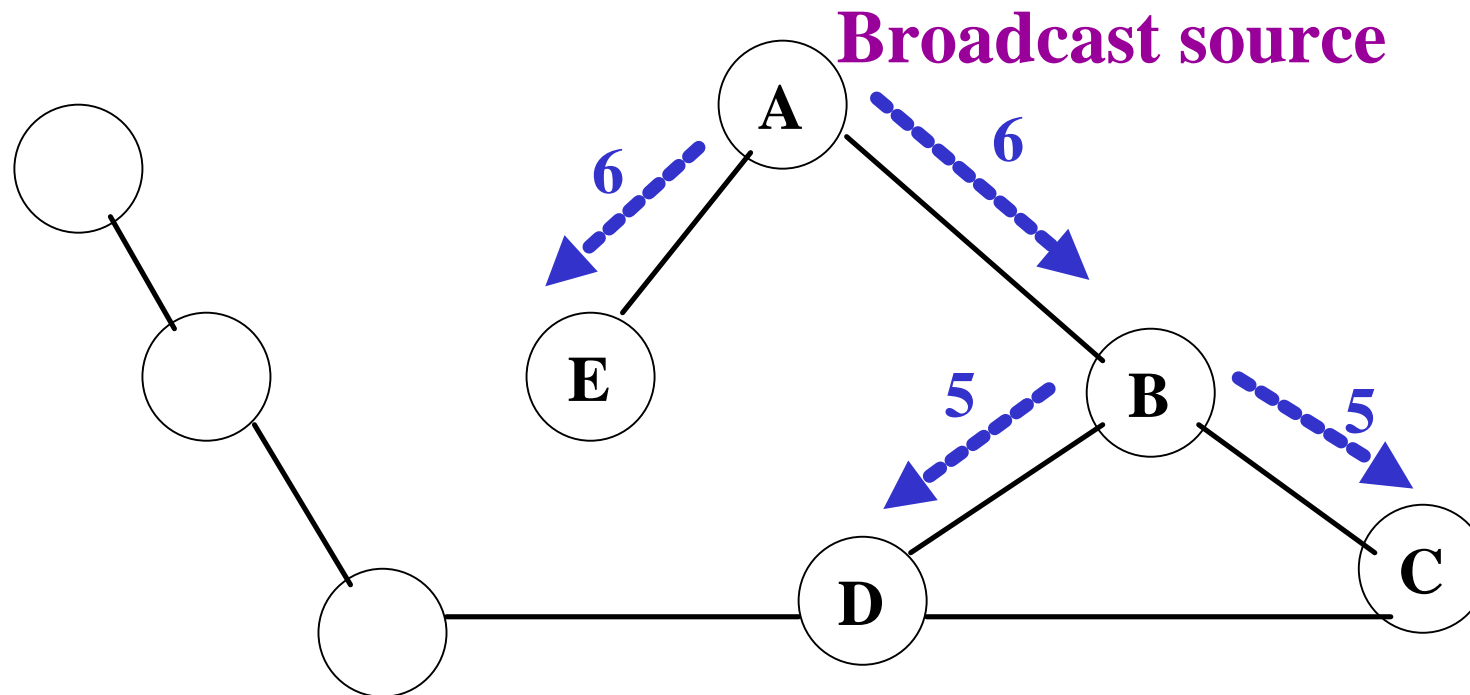
## (2) IBM's ATM Solution



**Hop counter decreased. When hop counter reaches 0, discard message.**

# Industrial Solutions to the self stab problem

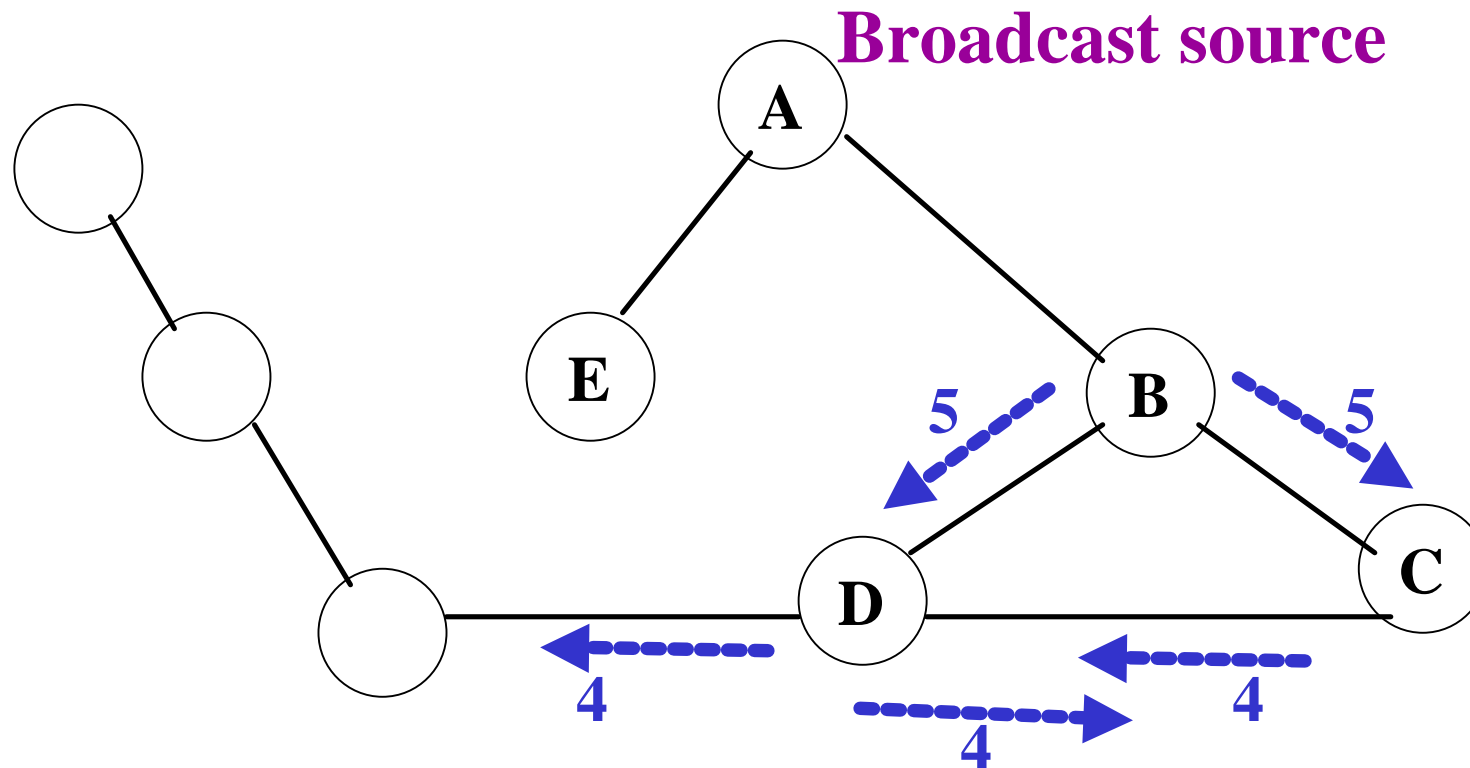
## (2) IBM's ATM Solution



**Hop counter decreased. When hop counter reaches 0, discard message.**

# Industrial Solutions to the self stab problem

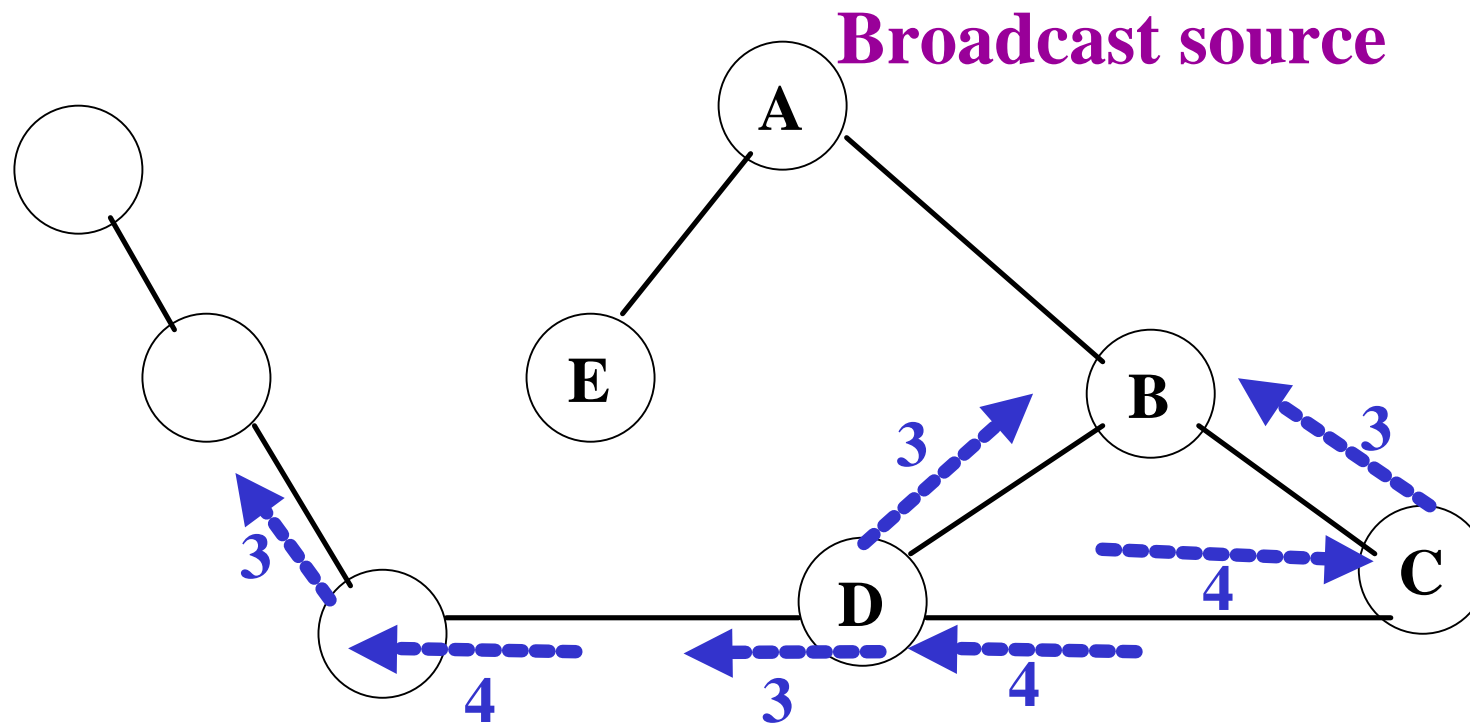
## (2) IBM's ATM Solution



**Hop counter decreased. When hop counter reaches 0, discard message.**

# Industrial Solutions to the self stab problem

## (2) IBM's ATM

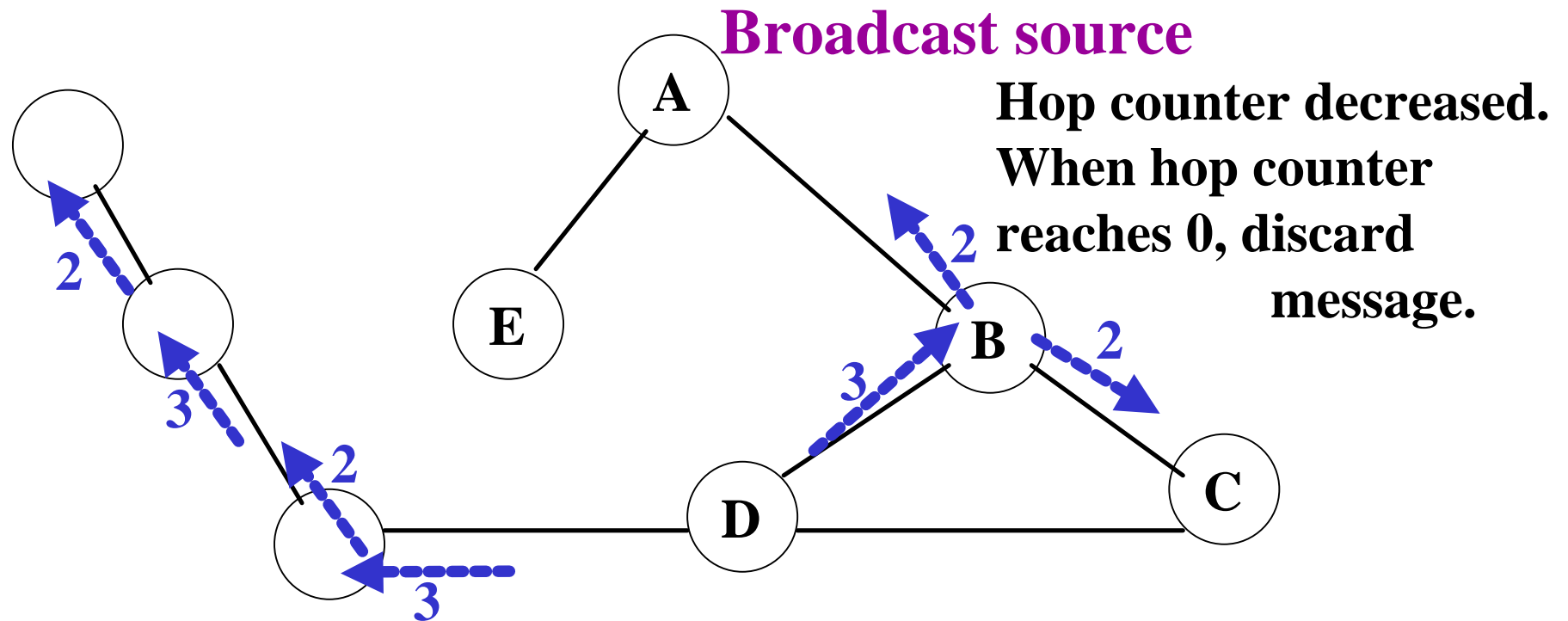


**Hop counter decreased. When hop counter reaches 0, discard message.**



# Industrial solutions to the self stab problem

## (3) Gnutella's P2P (no tree)



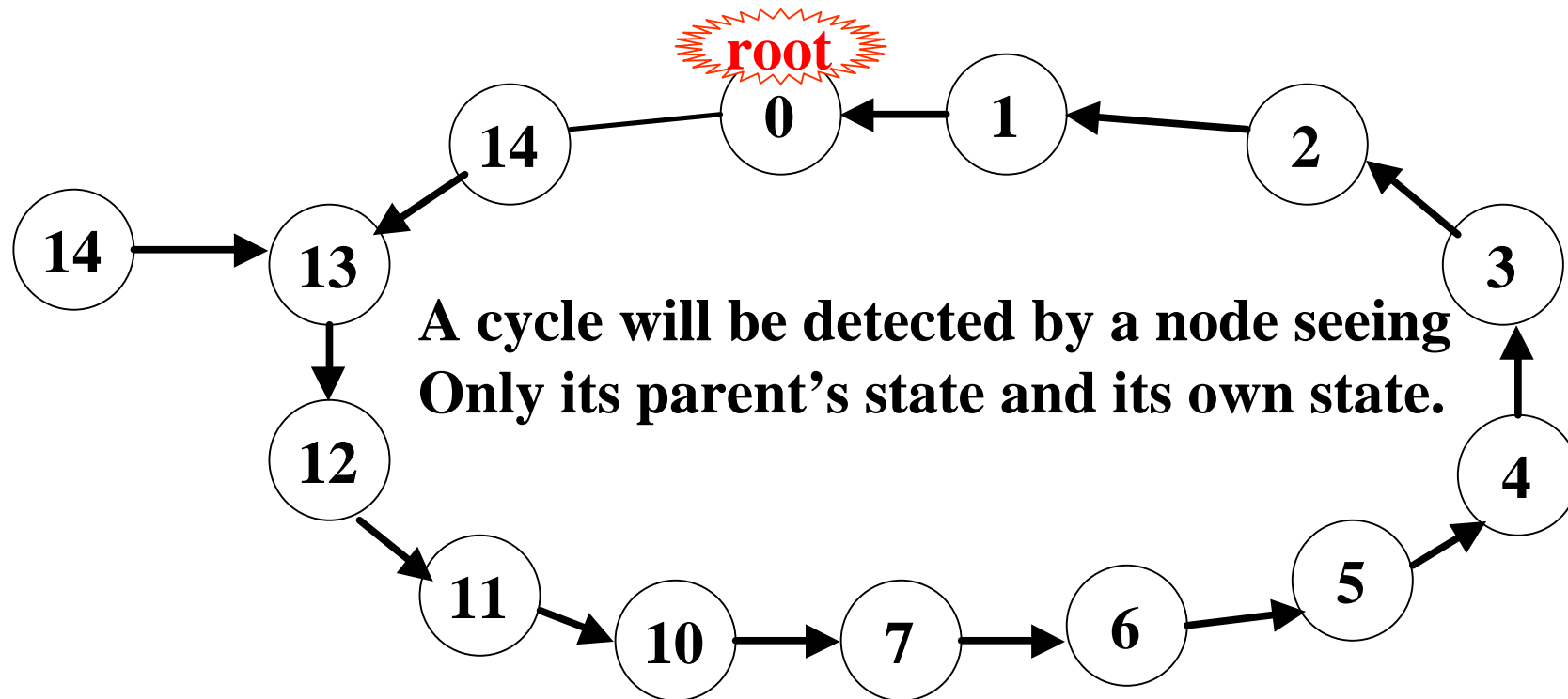
**Recently my student found that in one method they do not check the TTL (hop counter)- this crashed his system.**

# Scalability

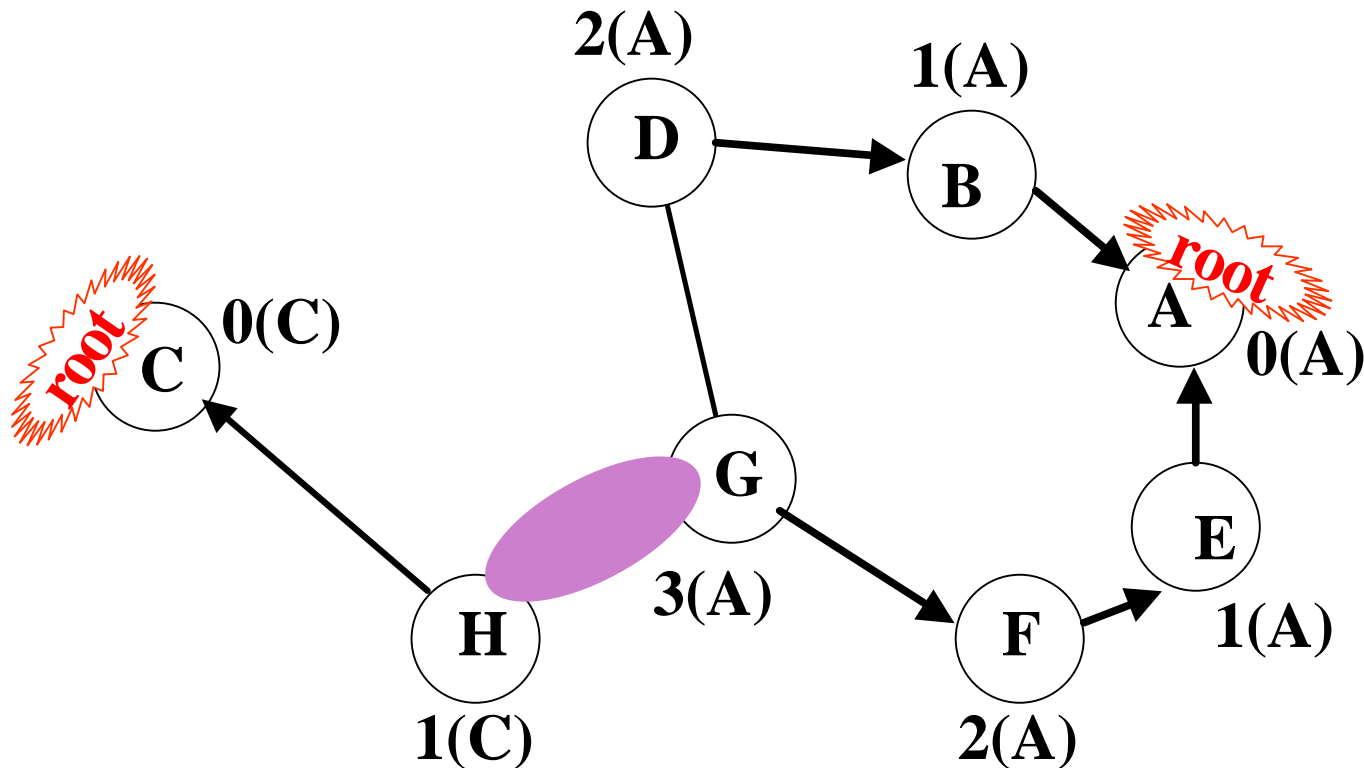
- **Industrial solutions are **global**.**  
**They do not scale well**
- **Conventional self stab protocols are **global**.**  
**They do not scale well**

# Local detection [Afek, Kutten, Yung]

(or local “checking” [Awerbuch, Patt-Shamir, Varghese])



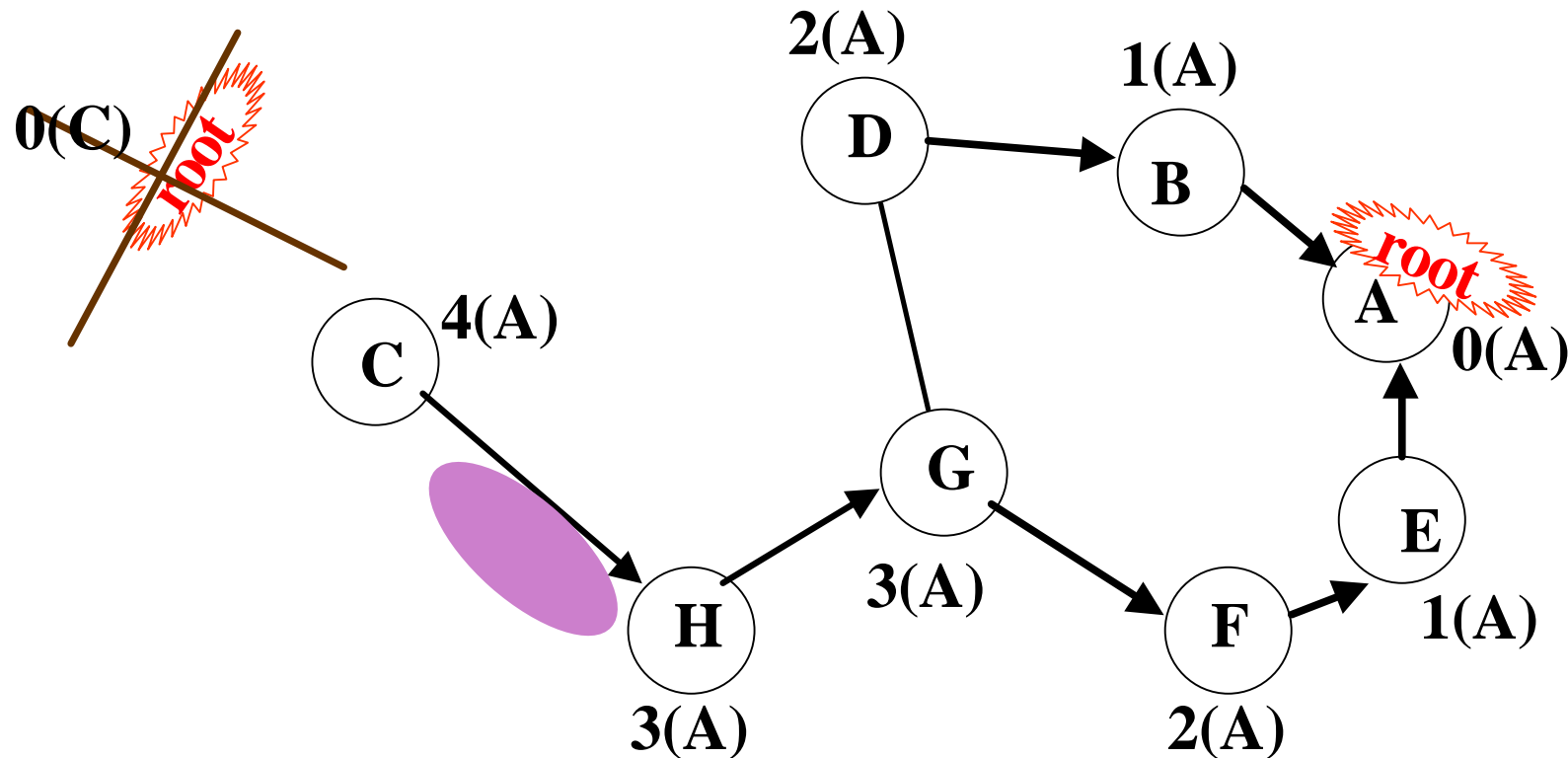
# Local checking of spanning tree, with root



(With unique node's ID) the existence of more than one tree is detected in the **border** between the tree.



By the way, a distributed algorithm:  
spanning tree



**Assume  $A < C$  then H abandons C and joins A's tree**

# Local checking of other predicates

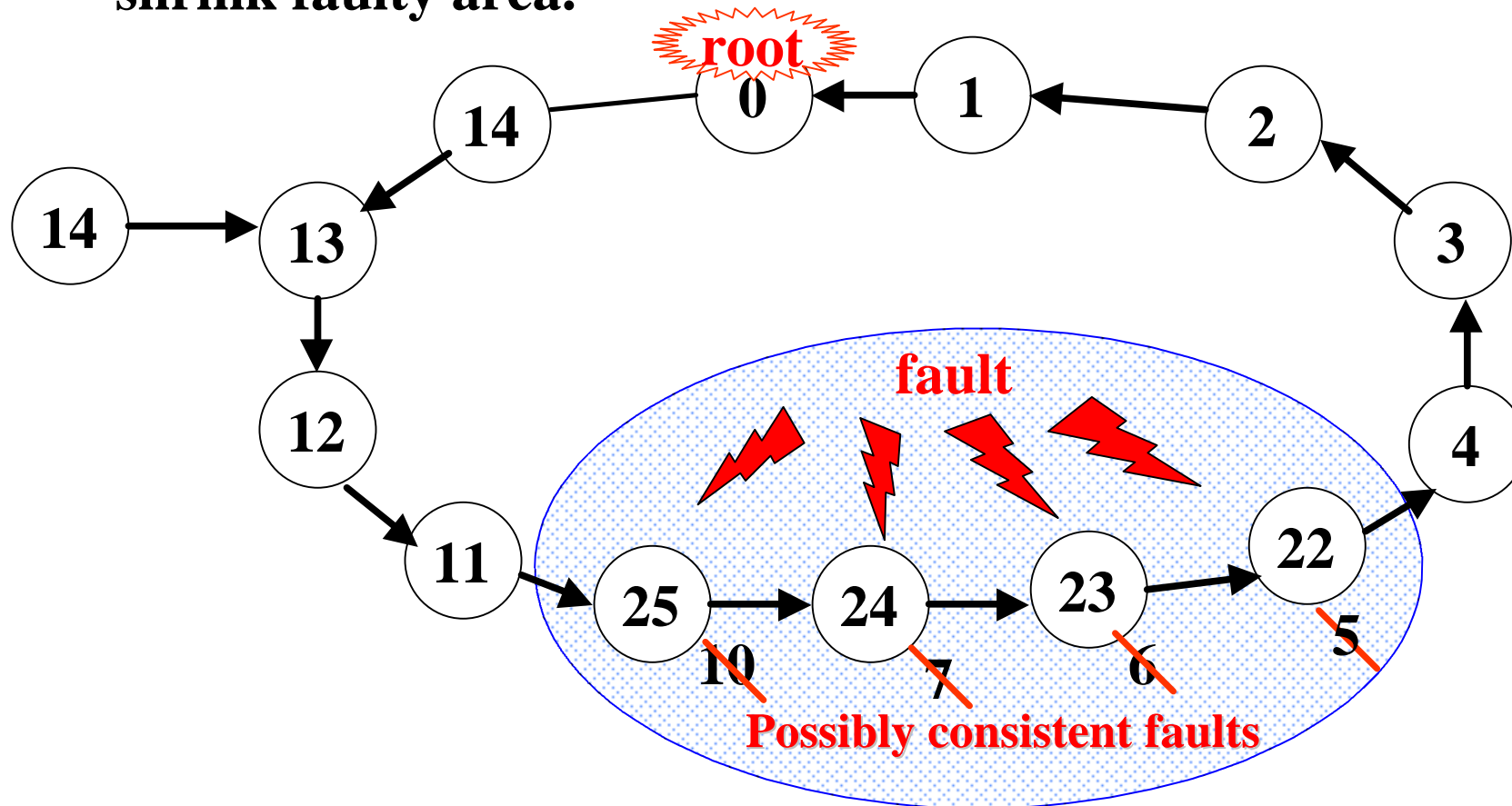
The bit complexity of tree local checking is  $q(\log n)$

- Any graph marking function is locally checkable
- $\Rightarrow$  Any algorithm global state is locally checkable
- For any bit complexity  $C$ ,  $\exists$  function with local checking bit complexity  $O(C)$
- Some complexities for interesting marking functions are known
- **Many other problems are open!**

# From local checking to local correction

**Goal:** (unknown)  $f$  faults  $\rightarrow O(f)$  time for correction

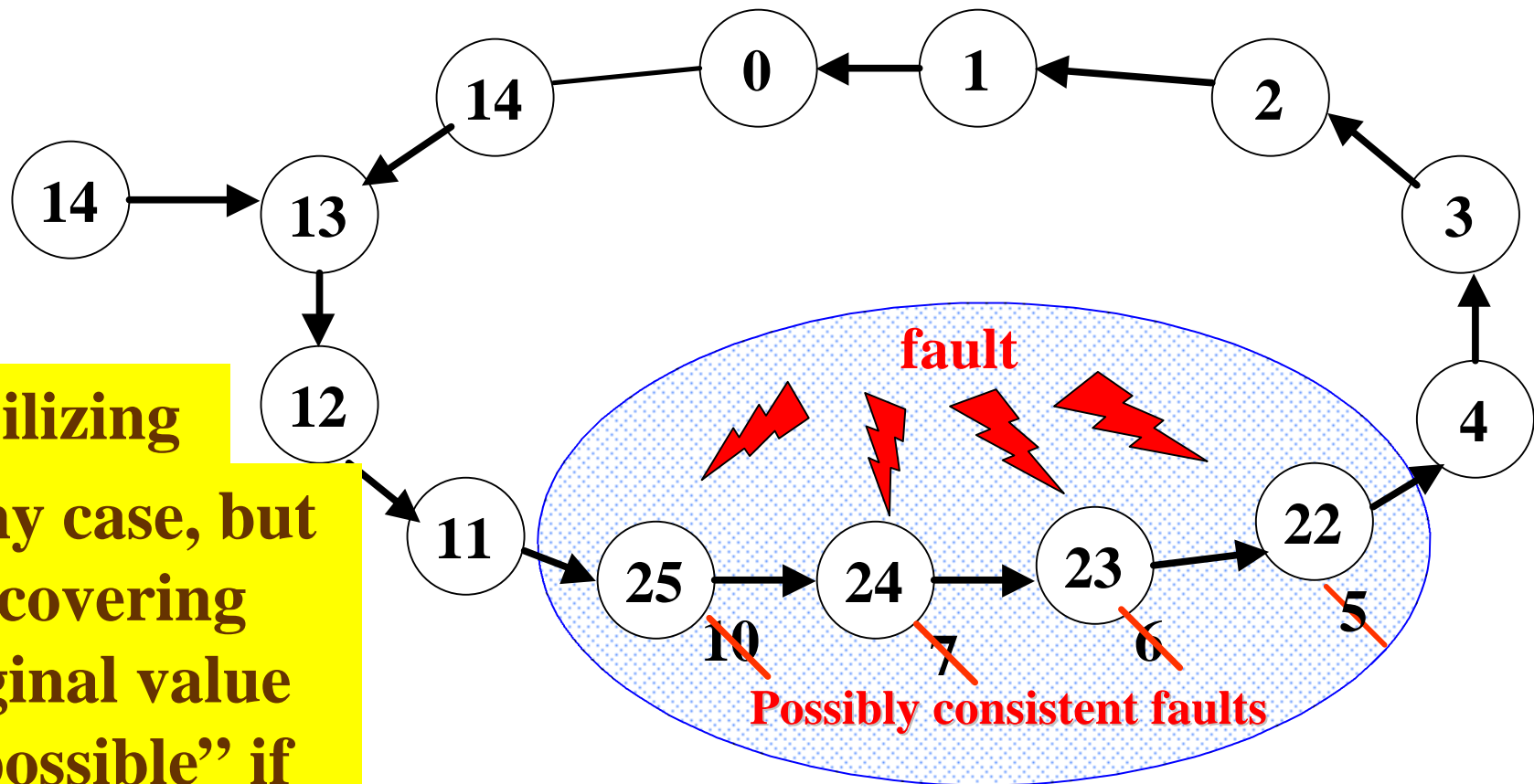
**Idea:** Diameter of faults is  $f$ . Prevent faults expansion, shrink faulty area.



# From local checking to local correction

**Goal:** (unknown)  $f$  faults  $\Rightarrow O(f)$  time for correction

Following spirit of Lamport's "fast" mutual exclusion, and of "early termination".

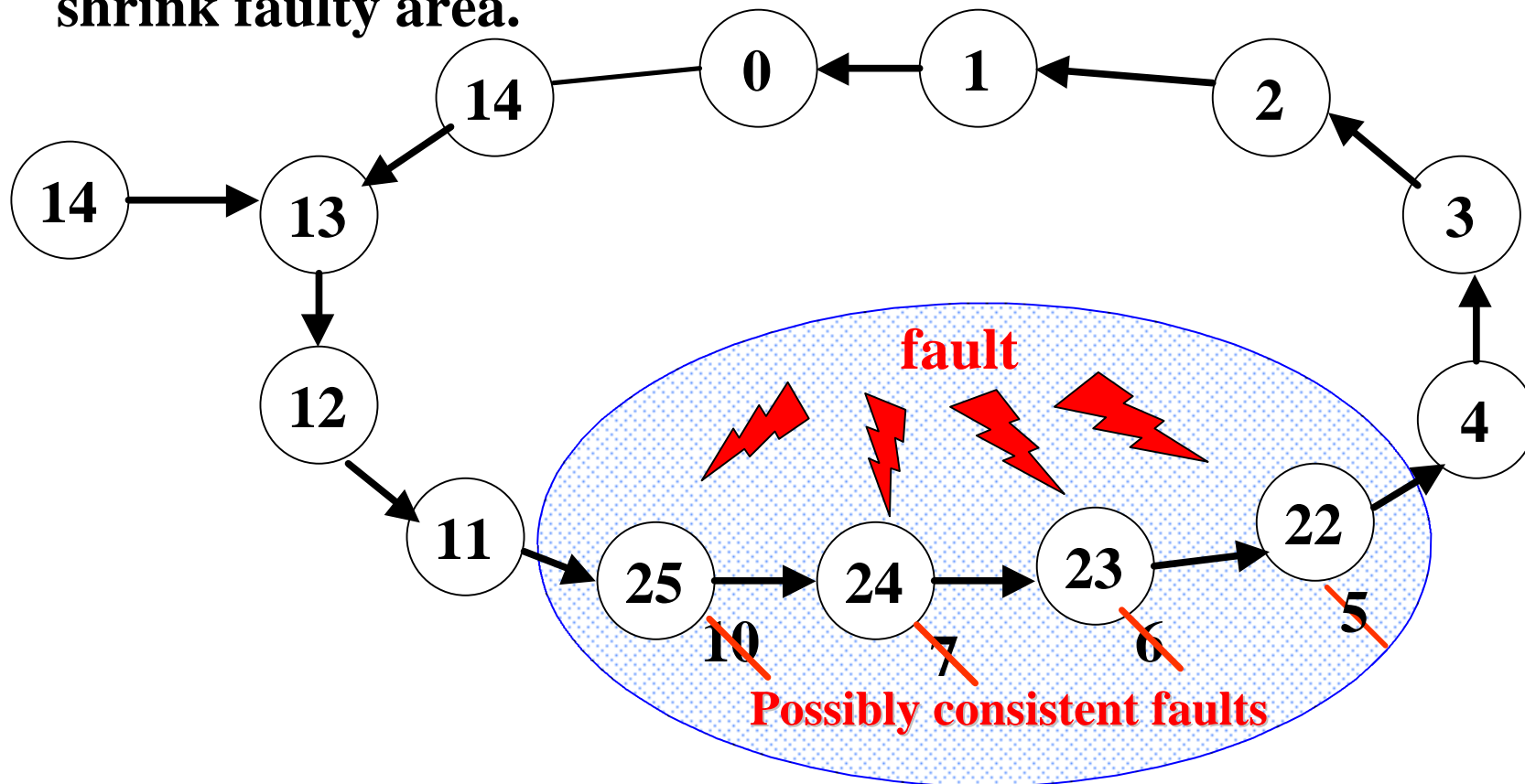


**Stabilizing in any case, but recovering original value impossible" if faulty majority)**

# From local checking to local correction

**Goal:** (unknown)  $f$  faults  $\Rightarrow O(f)$  time for correction

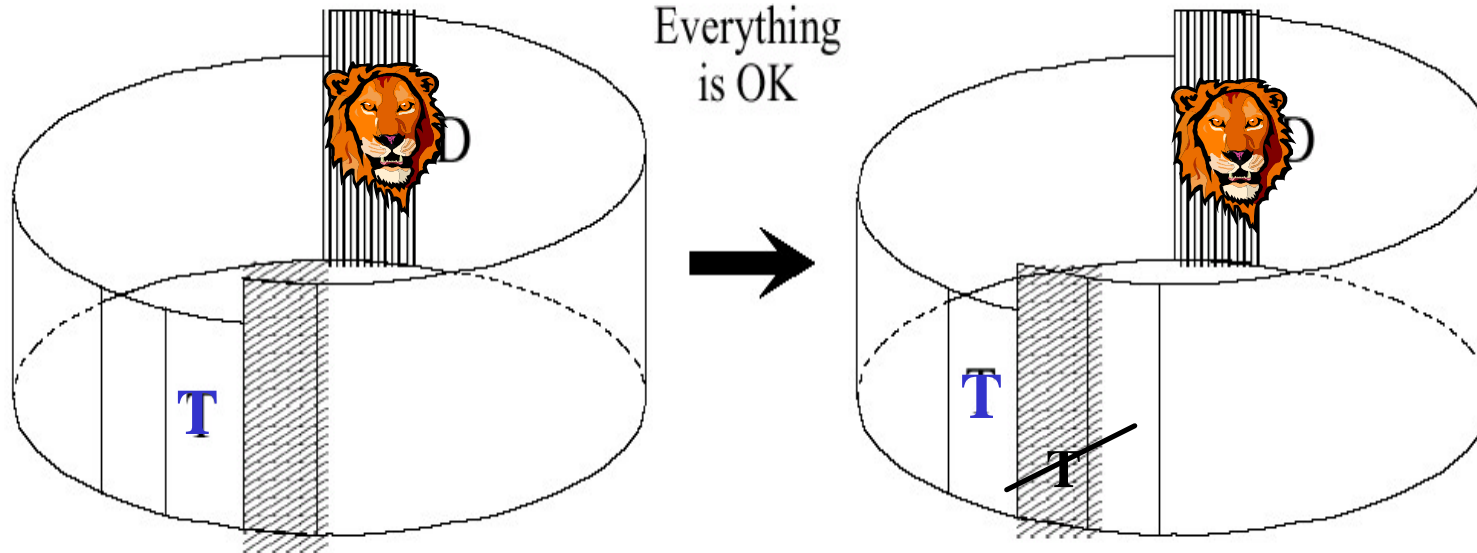
**Idea:** Diameter of faults is  $f$ . Prevent faults expansion, shrink faulty area.



# Another look at Dijkstra's algorithm

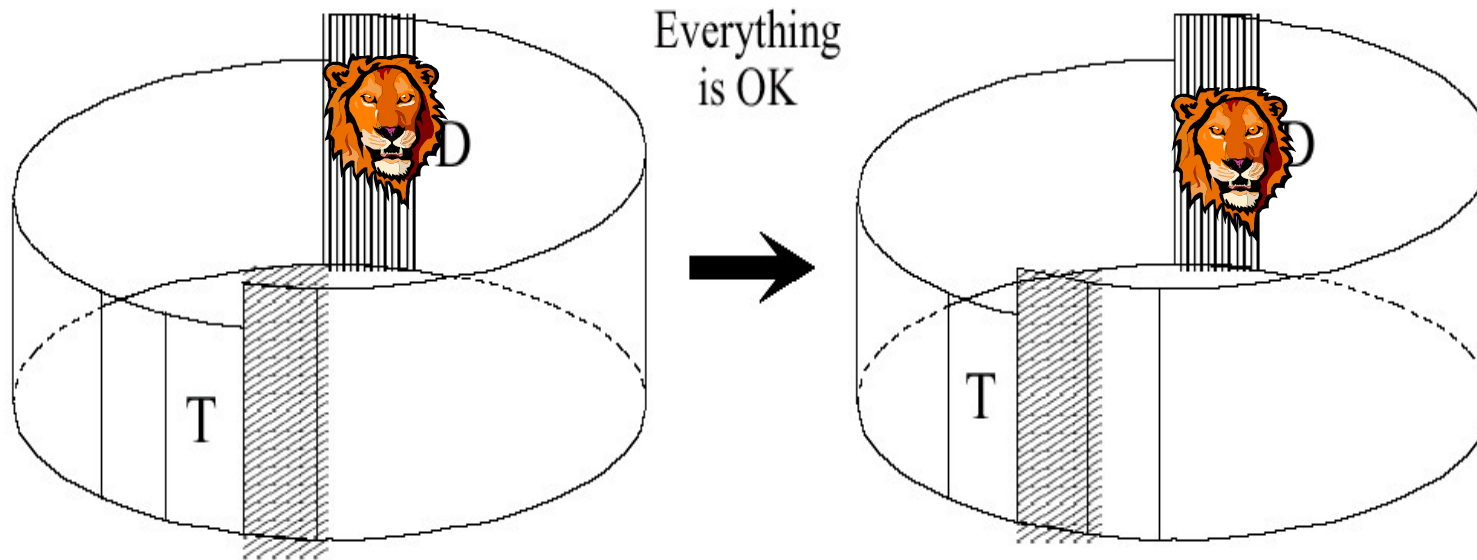
**A token in a non leader is a cut (or bump) in the wall**

**Special machine**

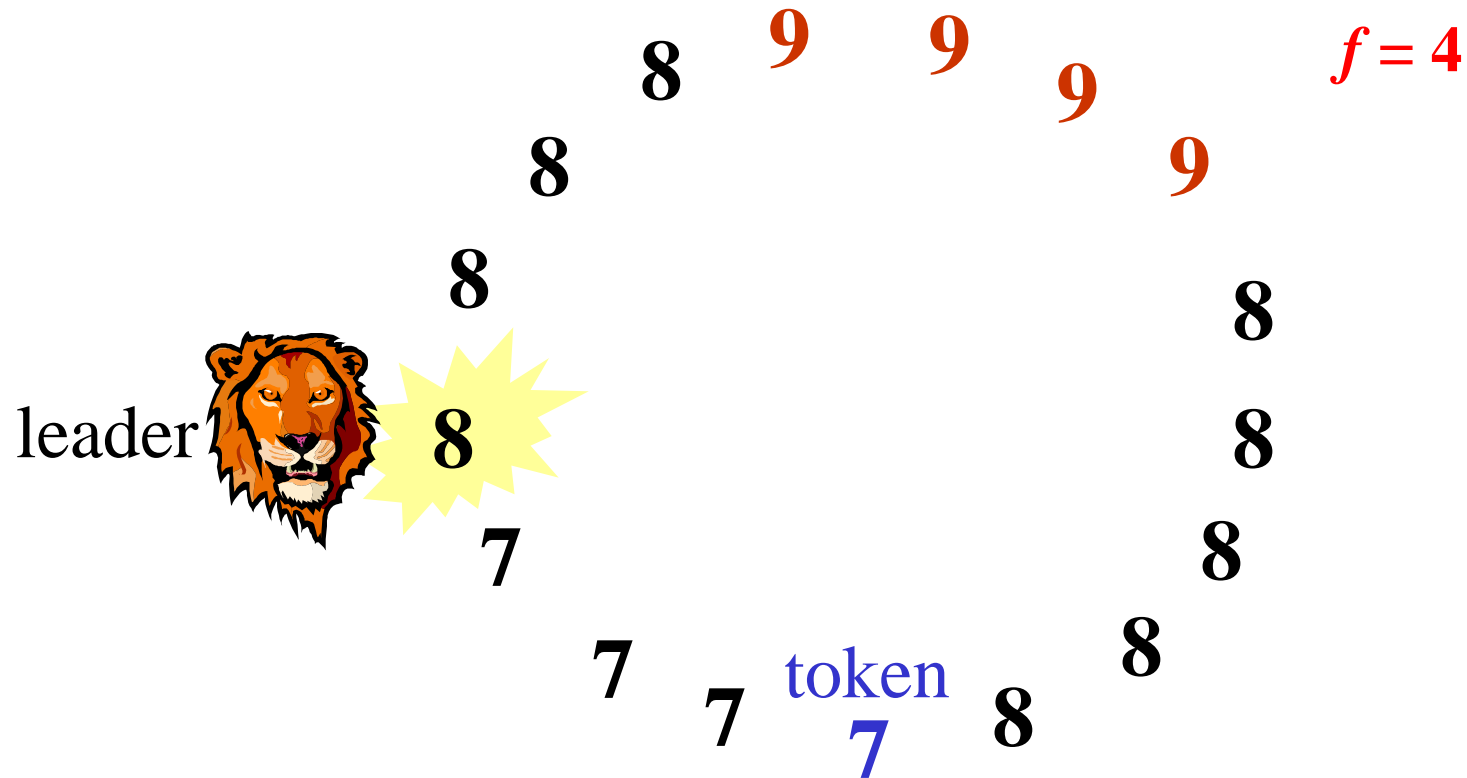


**Token move**

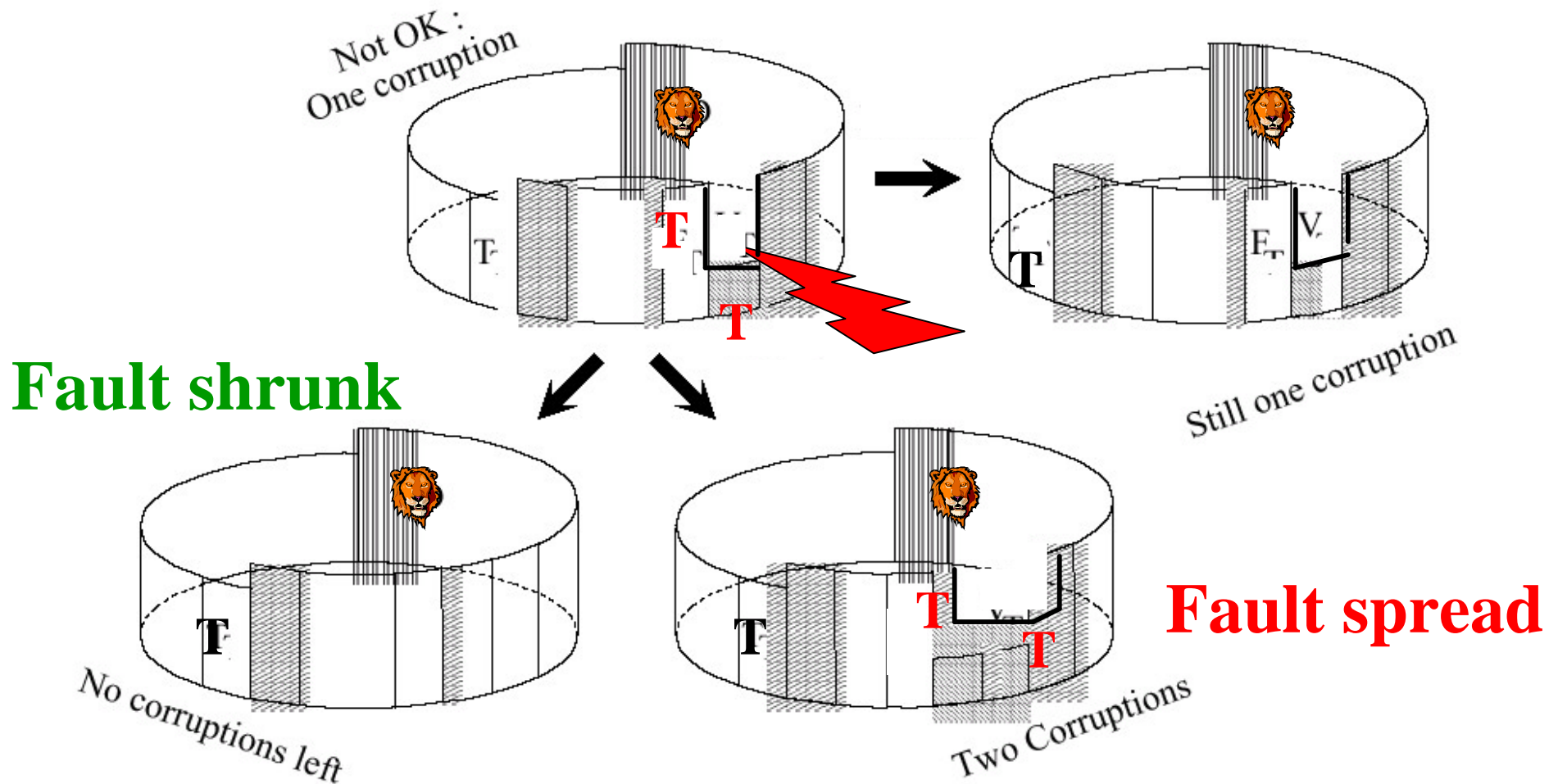
# Local correction example: need to prevent expansion of faulty area:



Recall:  $f$  faults create “gap” or “bump” of length  $f$ .



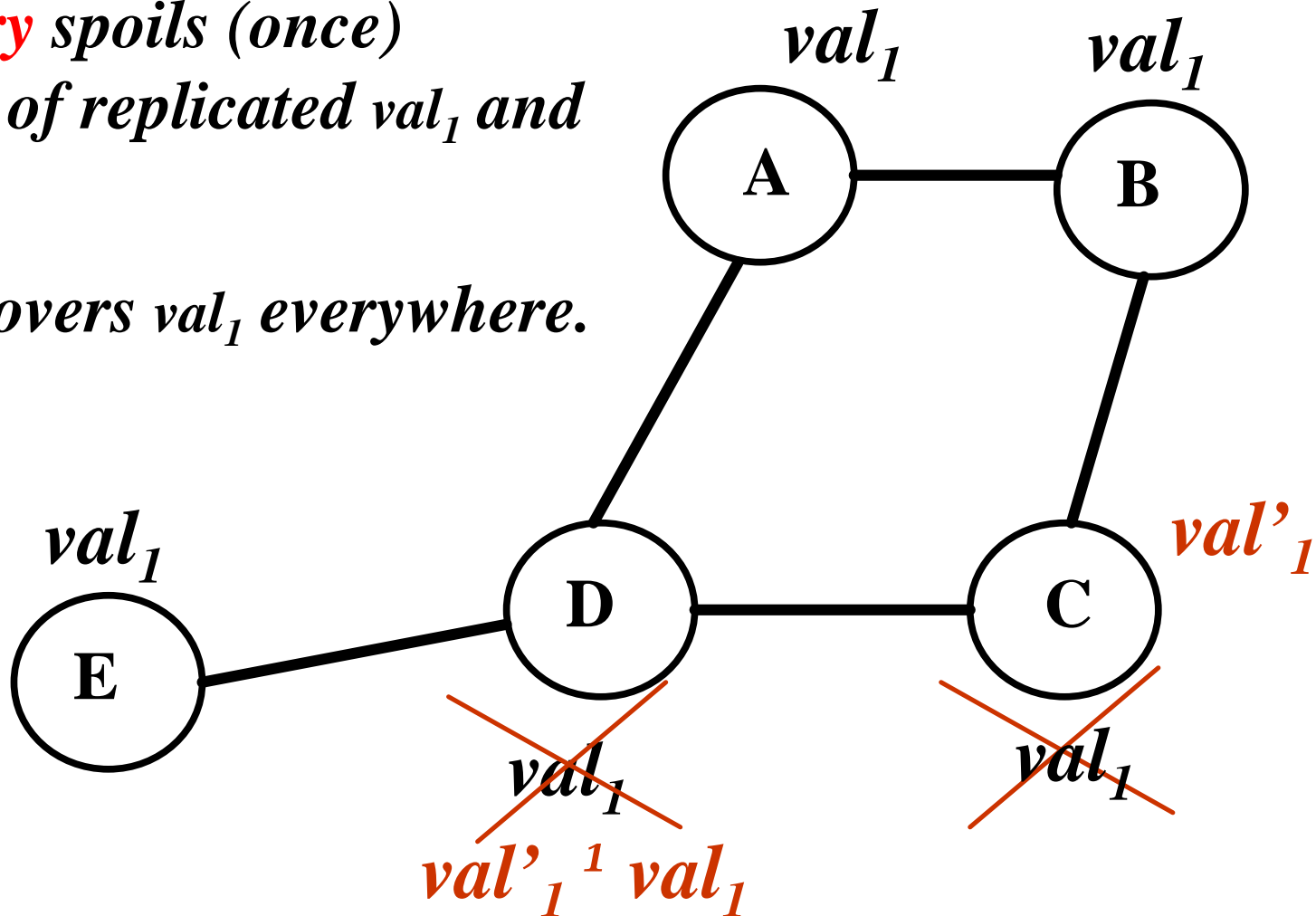
$f$  faults create “gap” or “bump” of length  $f$ .



# Stable Value Problem

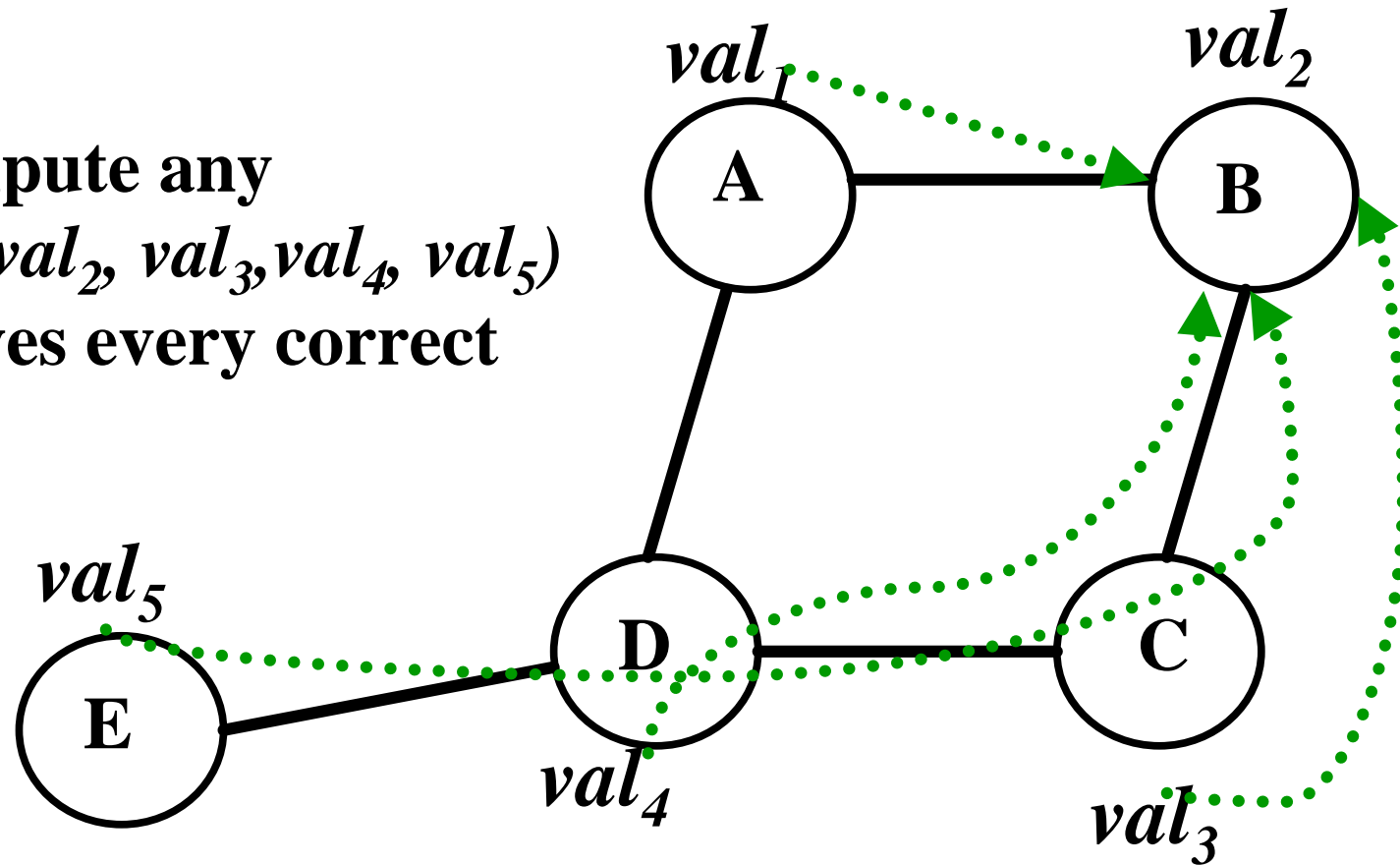
*Adversary spoils (once) minority of replicated  $val_1$  and of states.*

*Alg. Recovers  $val_1$  everywhere.*



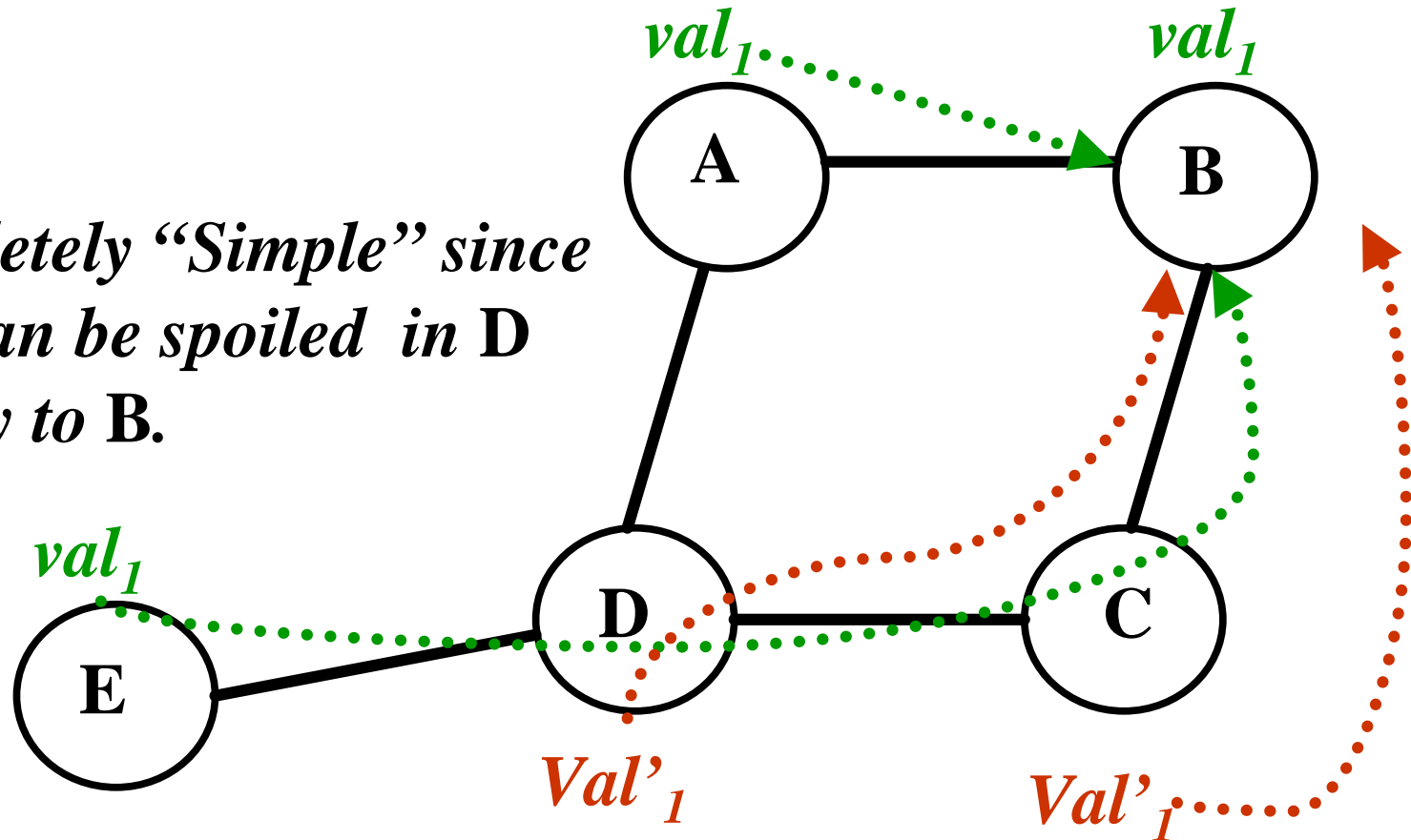
# Reducing a general problem to the Stable Value Problem

**B can compute any  $func(val_1, val_2, val_3, val_4, val_5)$  if it receives every correct  $val_i$ .**



“Simple” but *global* solution:  
consensus voting

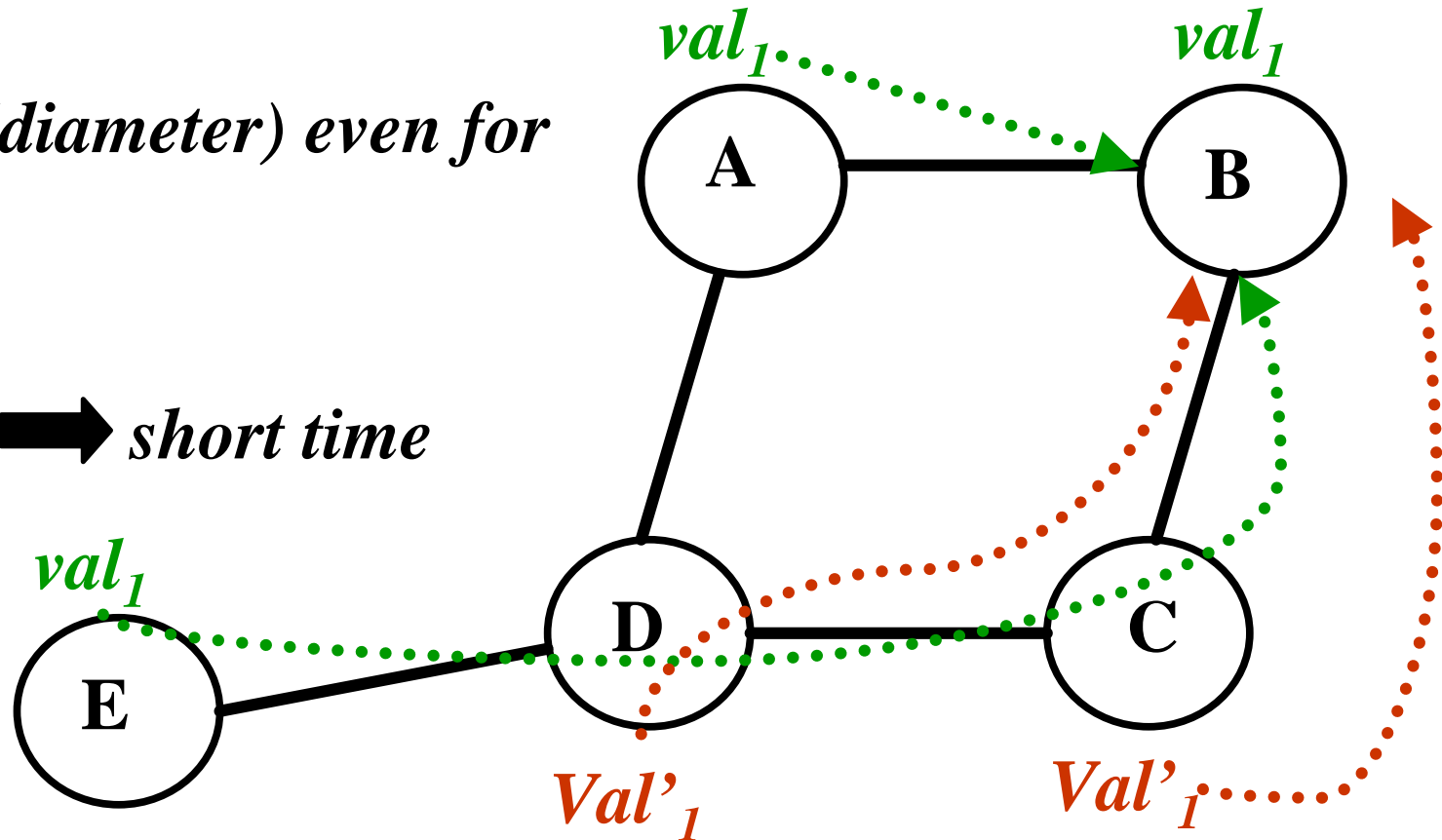
*Not completely “Simple” since  
E’s vote can be spoiled in D  
on the way to B.*



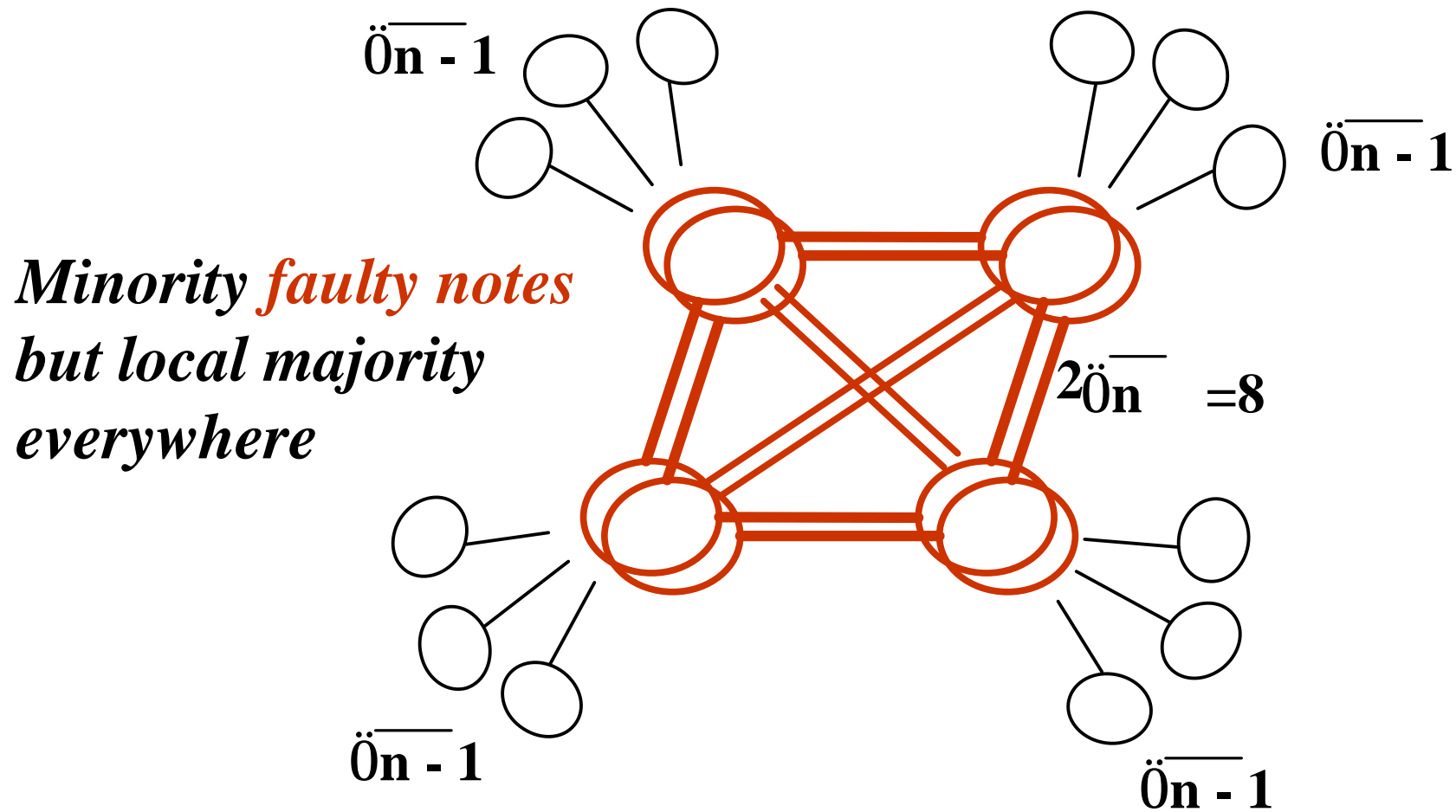
“Simple” but *global* solution:  
consensus voting

*Time =  $O(\text{diameter})$  even for  
**one fault.***

*Desired:*  
*few faults*  $\rightarrow$  *short time*

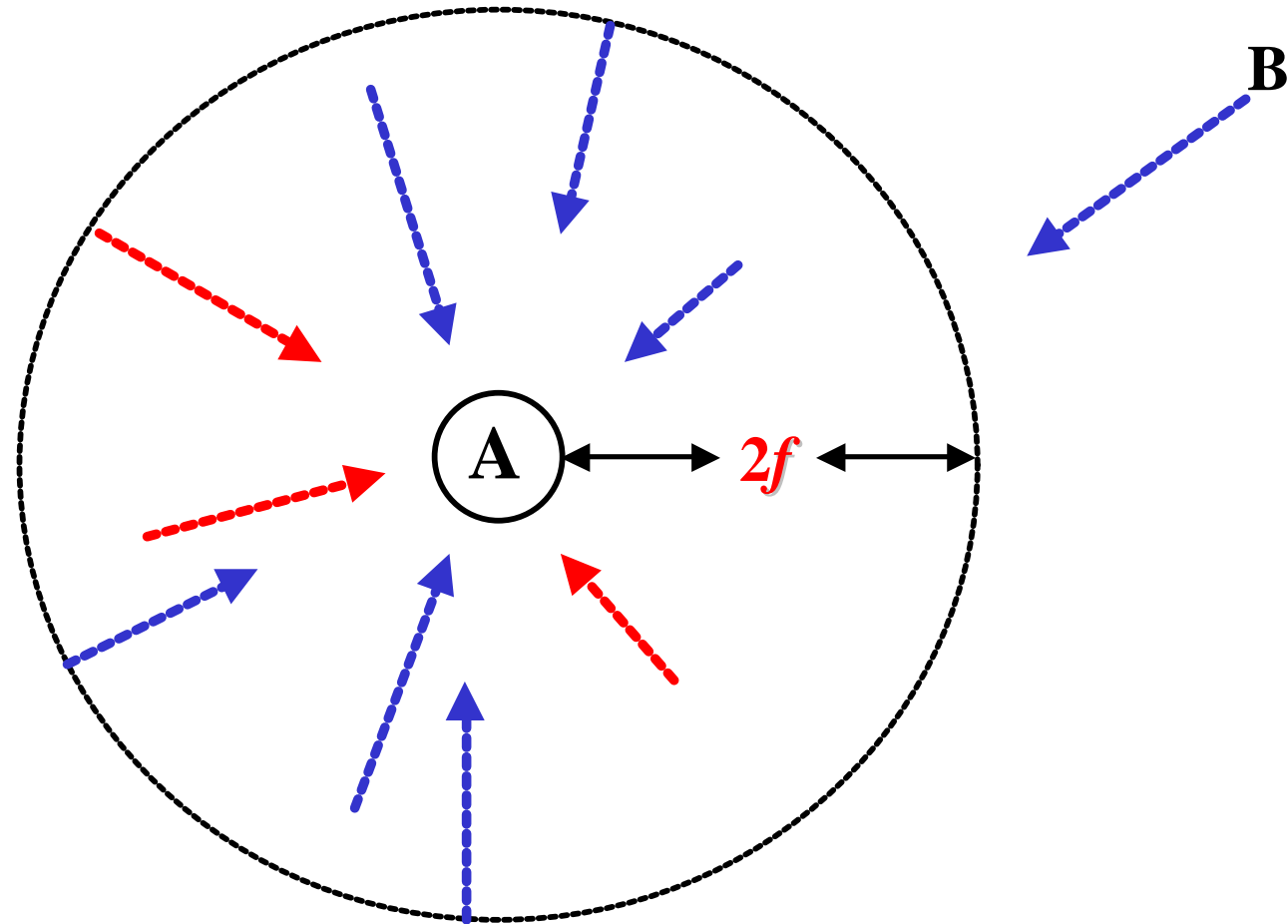


# Local voting does not solve



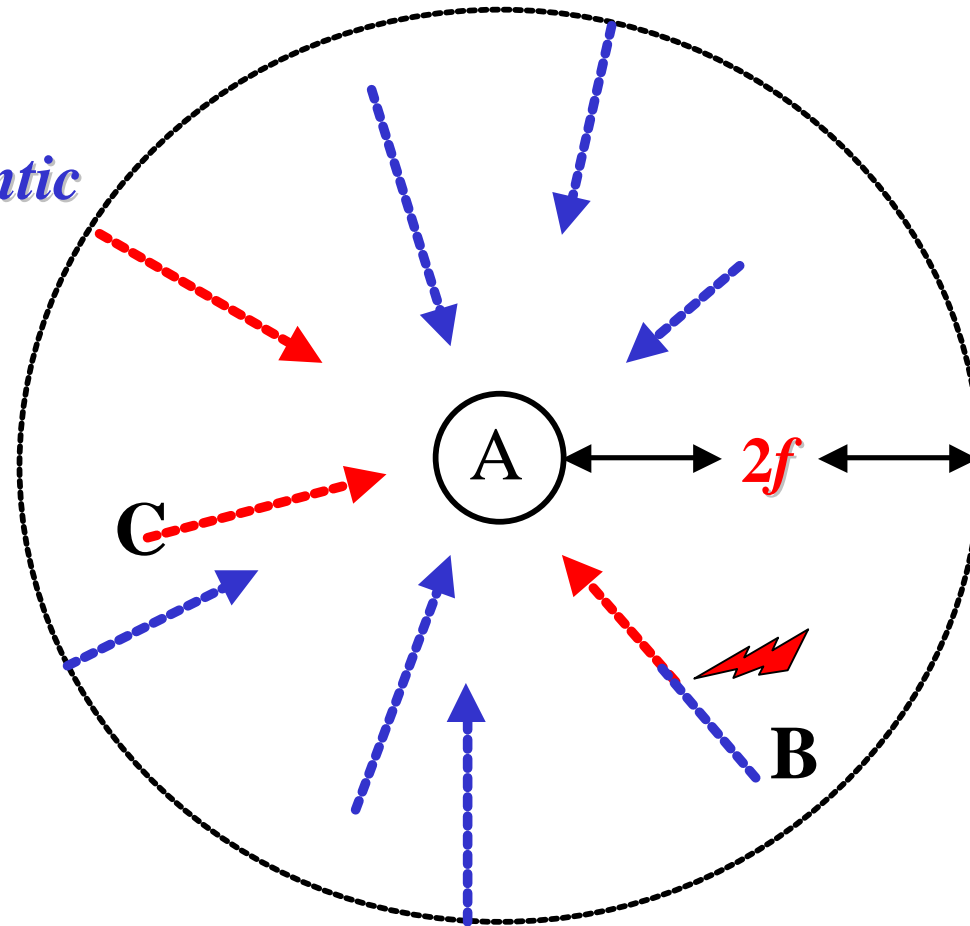
Idea 1: For (unknown)  $f$  faults get votes (values) from radius  $2f$

In  $O(f)$  time  
get  $>2f$  votes  
so majority is  
non faulty



Idea 1: For (unknown)  $f$  faults get votes (values) from radius  $2f$

In  $O(f)$  time  
get  $>2f$  *authentic*  
votes  
so majority is  
non faulty



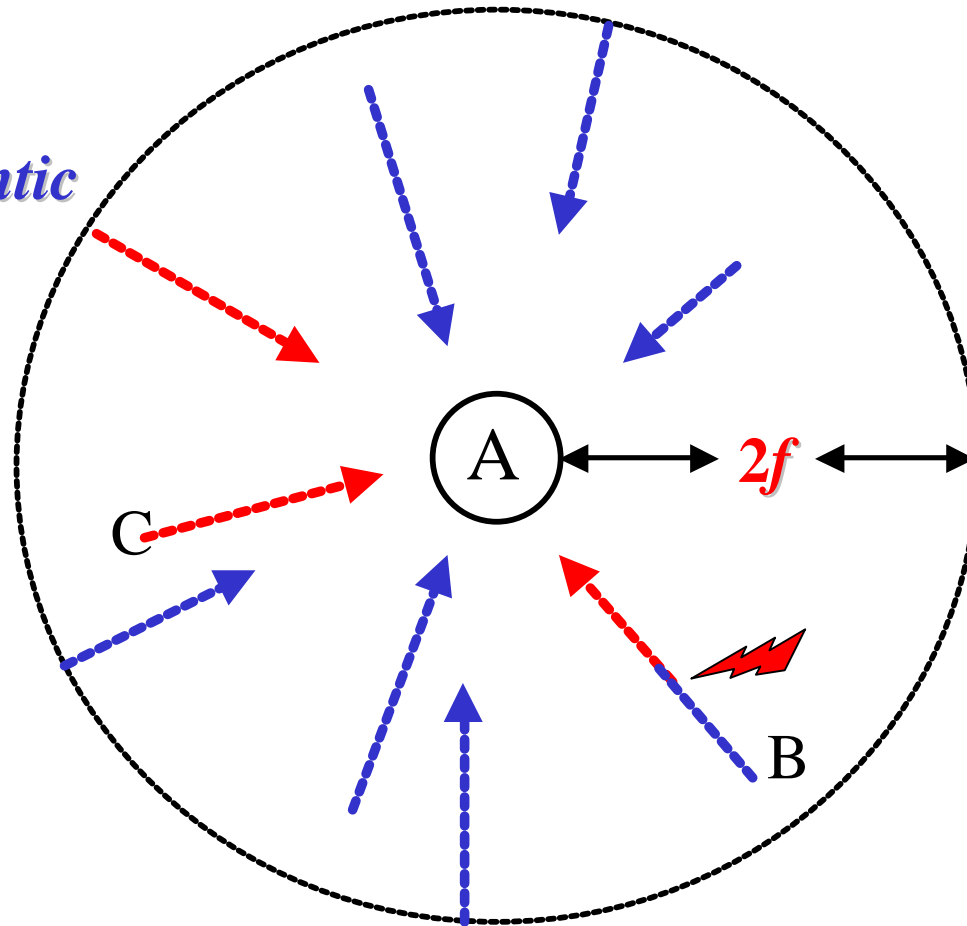
C's vote at A is  
authentic  
(though faulty)

B's vote at A is  
NOT authentic

# Idea 1: For (unknown) $f$ faults get votes (values) from radius $2f$

In  $O(f)$  time  
get  $>2f$  *authentic*  
votes  
so majority is  
non faulty

C's vote at A is  
authentic  
(though faulty)



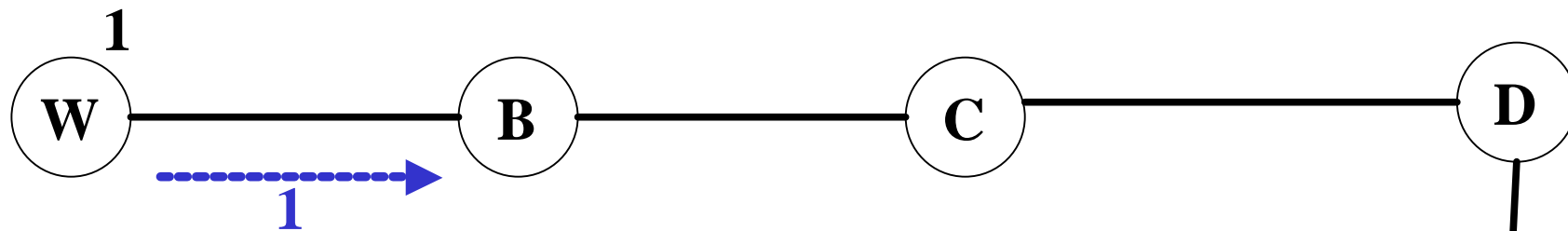
Also NOT  
authentic

A

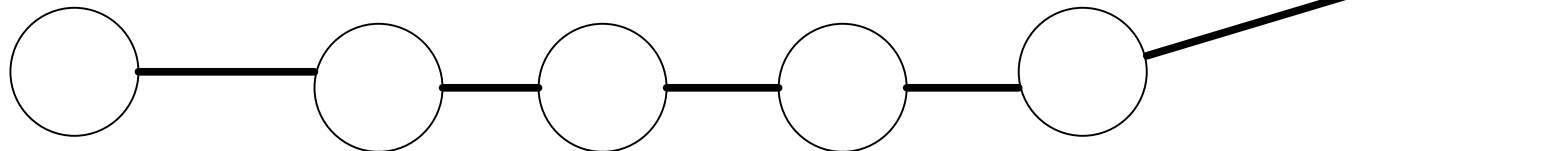
E

B's vote at A is  
NOT authentic

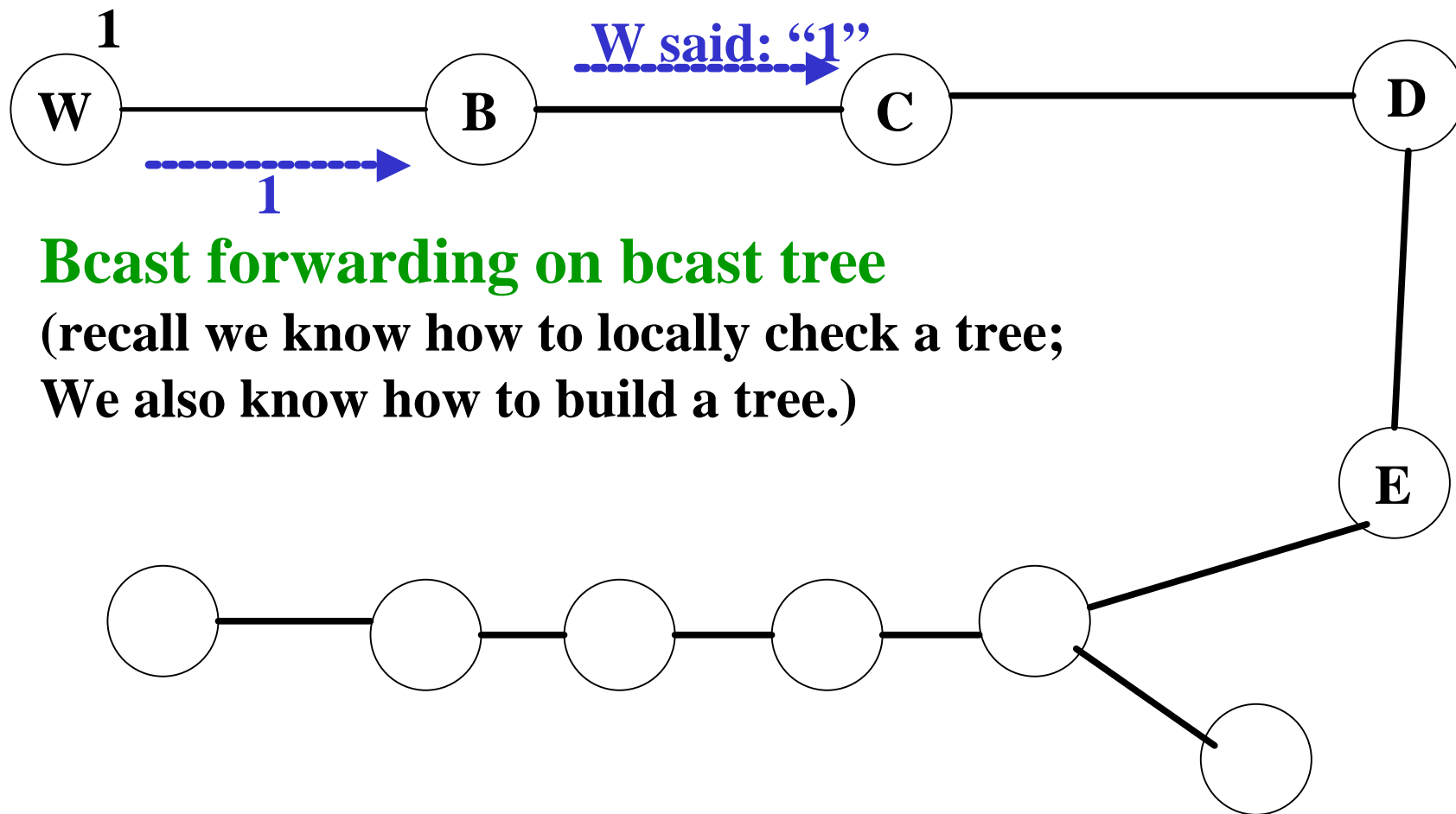
# Bcast (vote sending) spreading faults problem under state faults



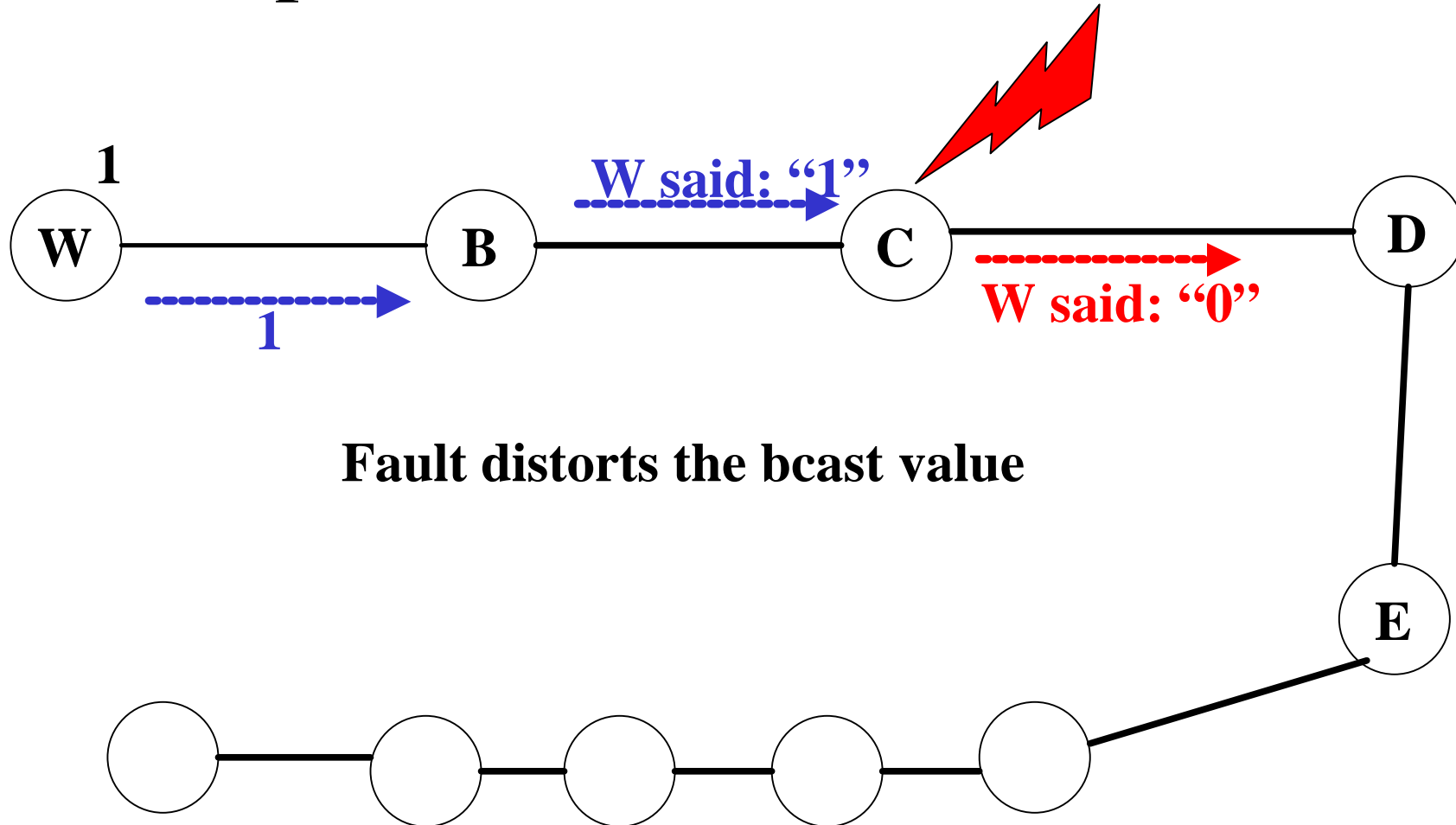
**W's vote is bcast over a self stabilized bcast spanning tree**



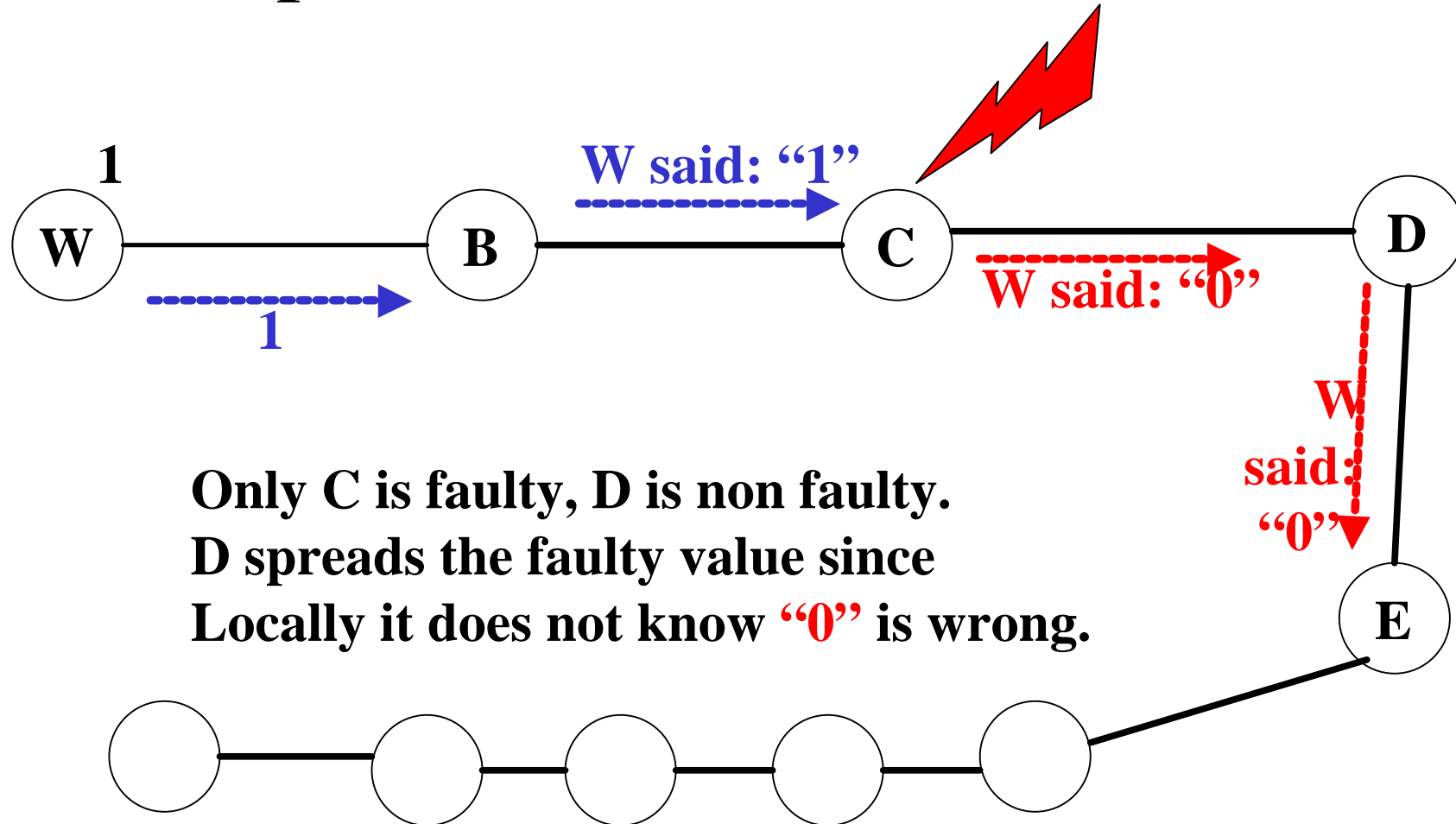
# Bcast (vote sending) spreading faults problem under state faults



# Bcast (vote sending) spreading faults problem under state faults

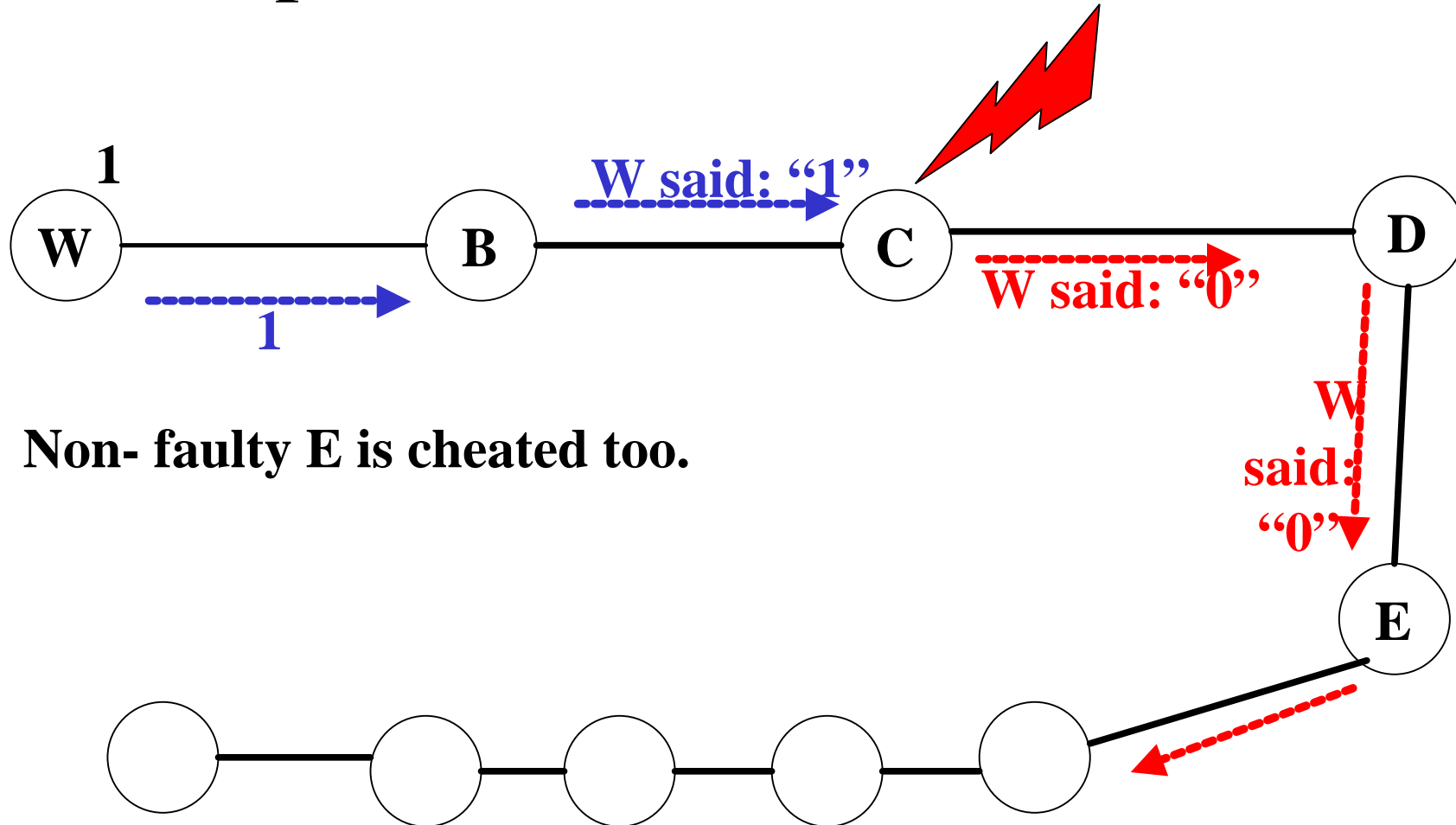


# Bcast (vote sending) spreading faults problem under state faults

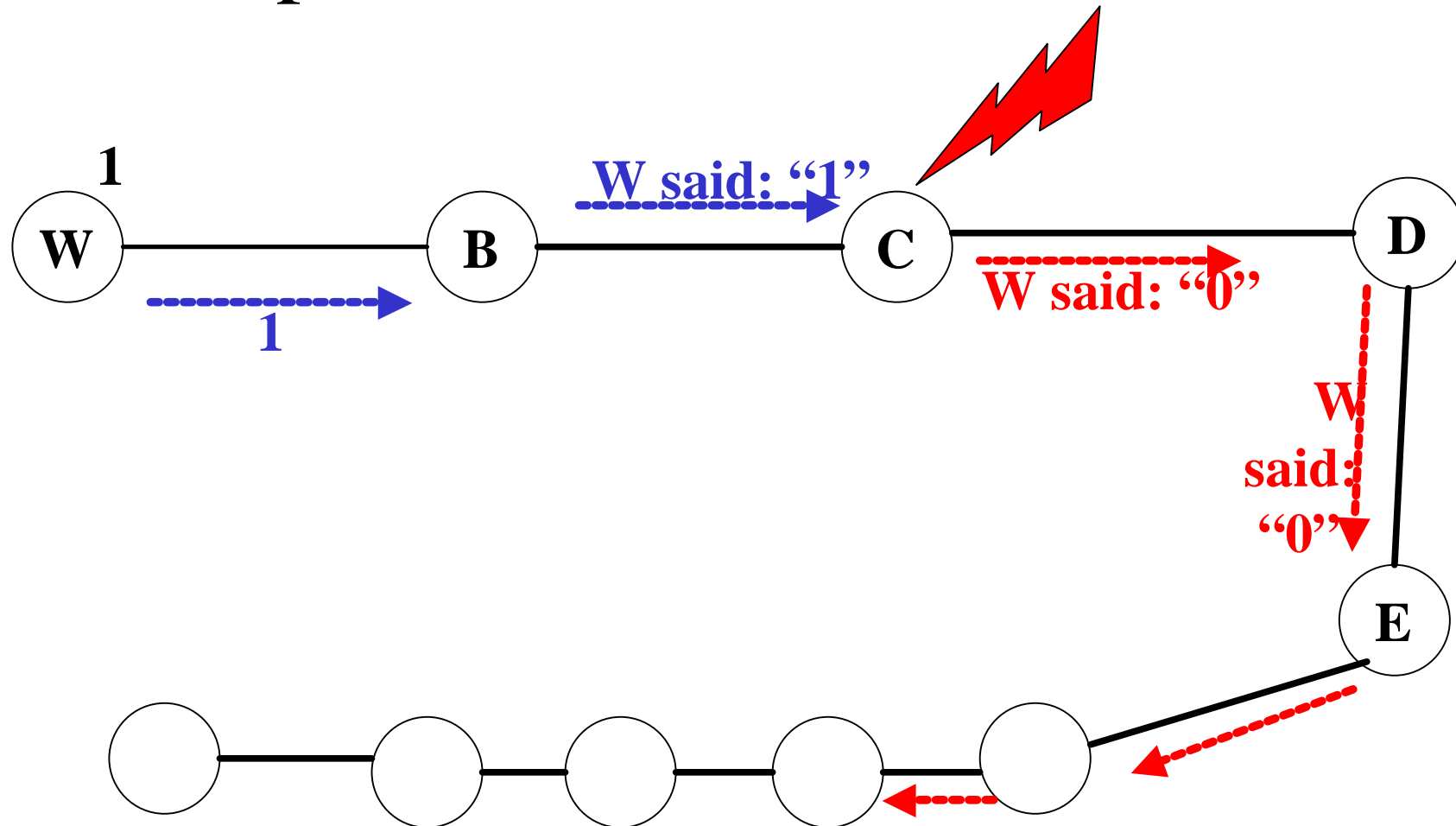


**Only C is faulty, D is non faulty.  
D spreads the faulty value since  
Locally it does not know "0" is wrong.**

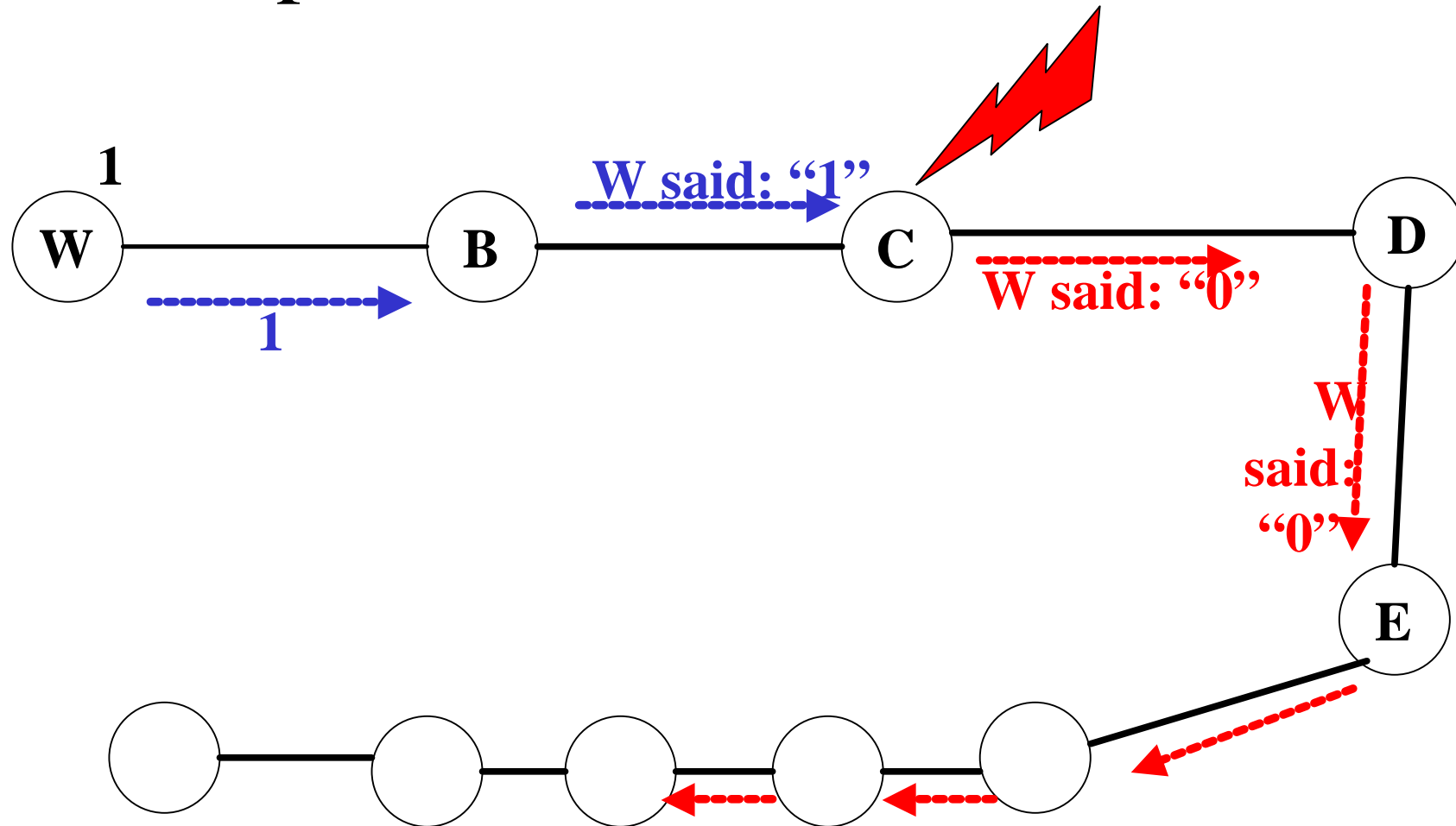
# Bcast (vote sending) spreading faults problem under state faults



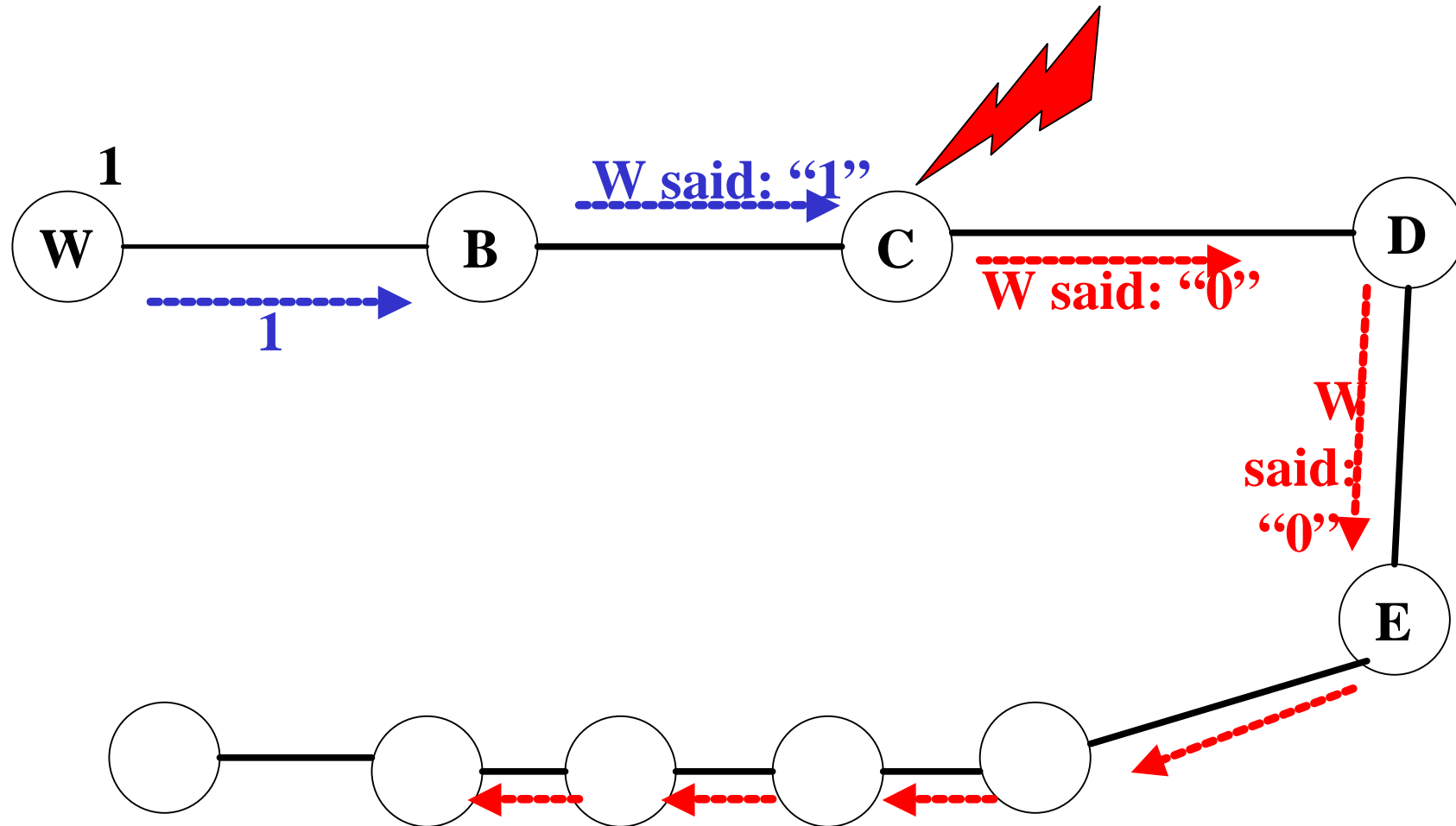
# Bcast (vote sending) spreading faults problem under state faults



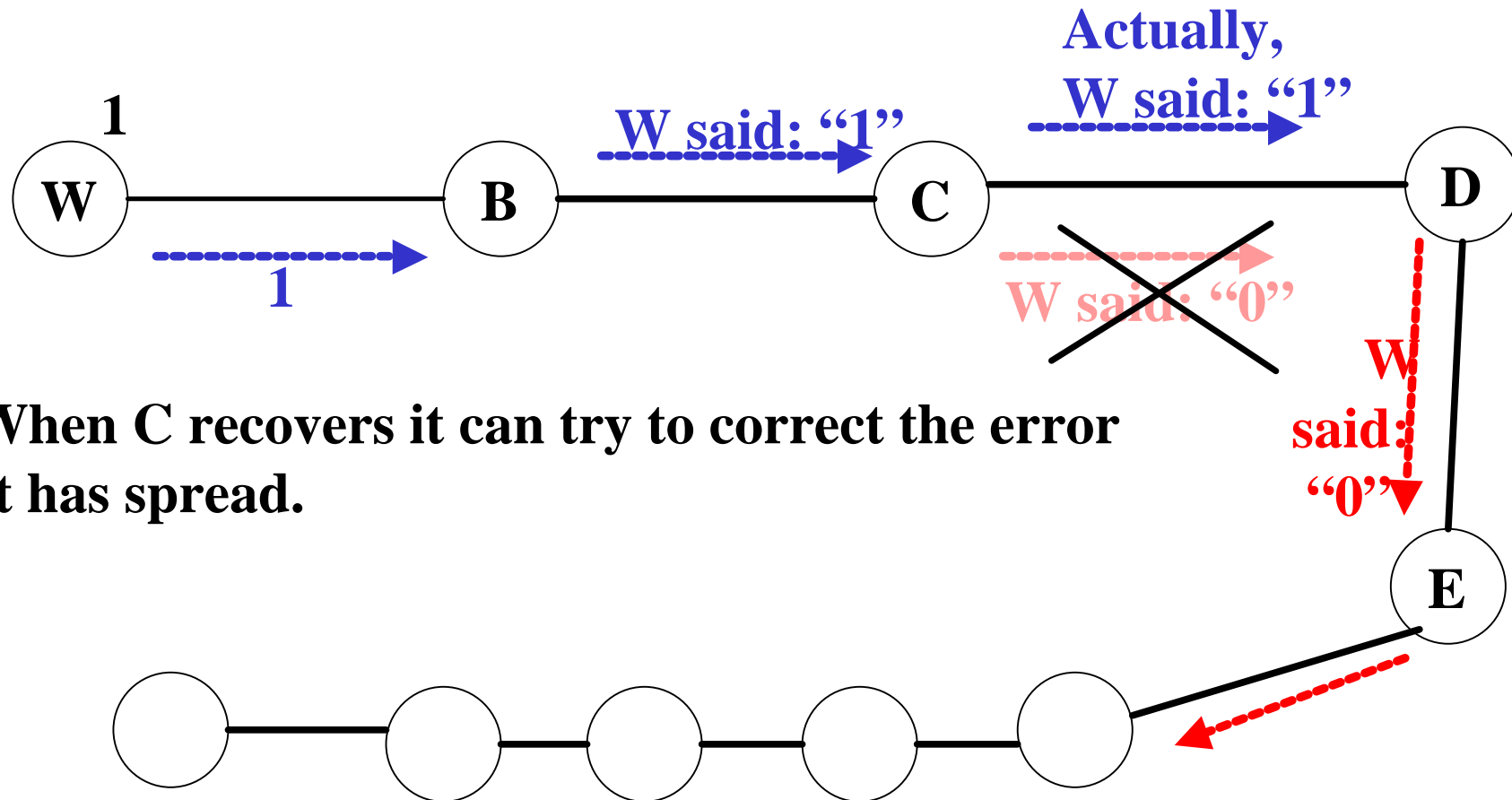
# Bcast (vote sending) spreading faults problem under state faults



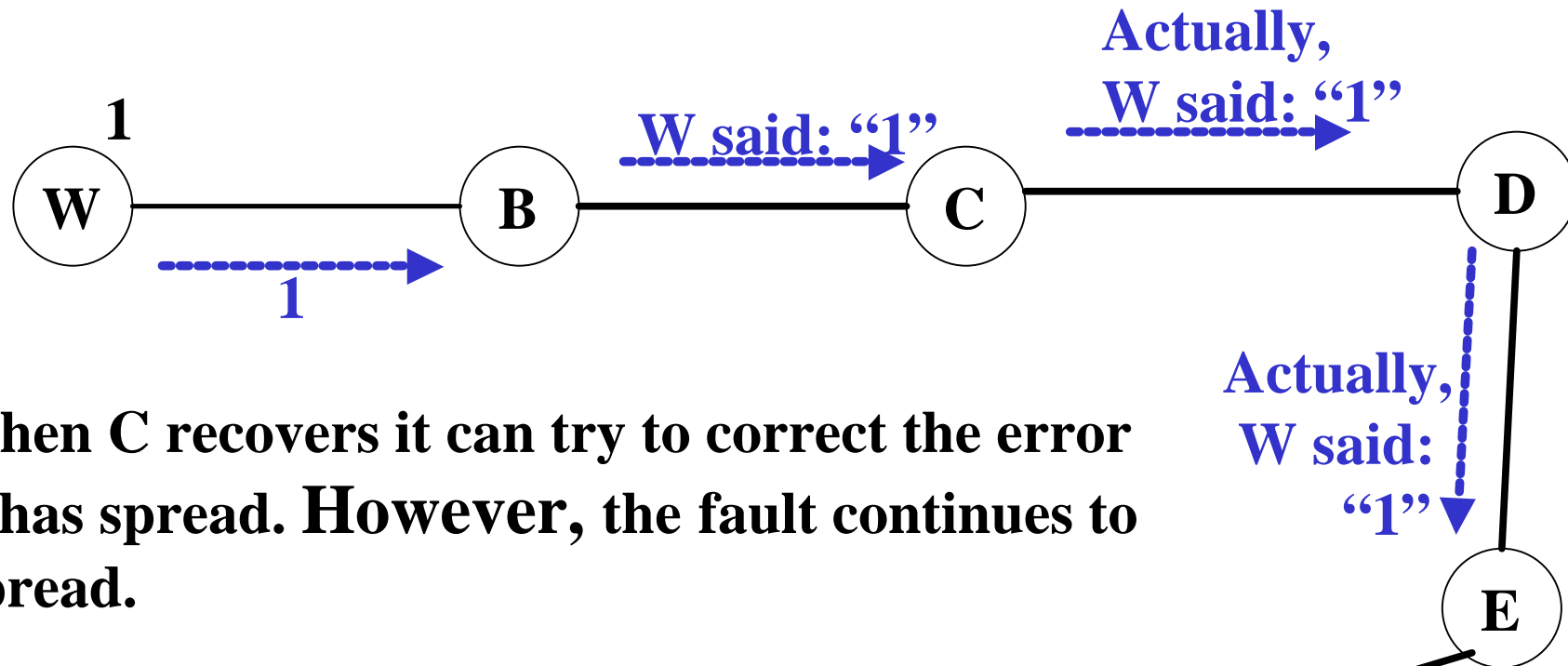
# Bcast (vote sending) spreading faults problem under state faults



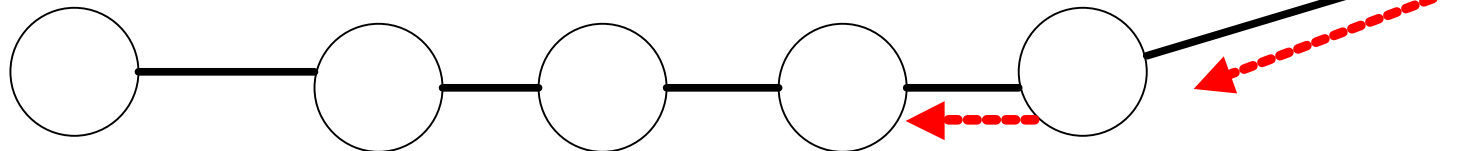
# Bcast (vote sending) spreading faults problem under state faults



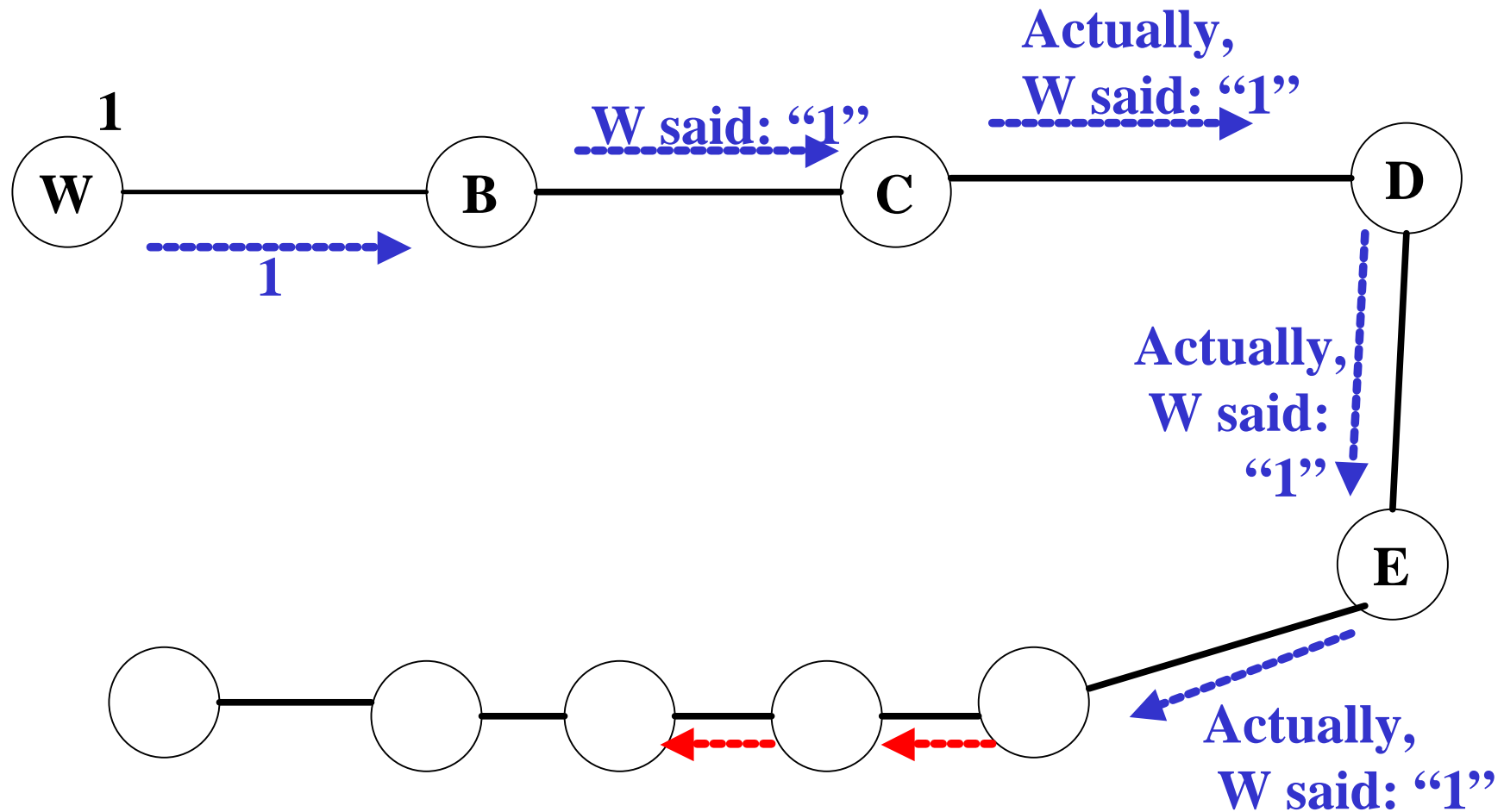
# Bcast (vote sending) spreading faults problem under state faults



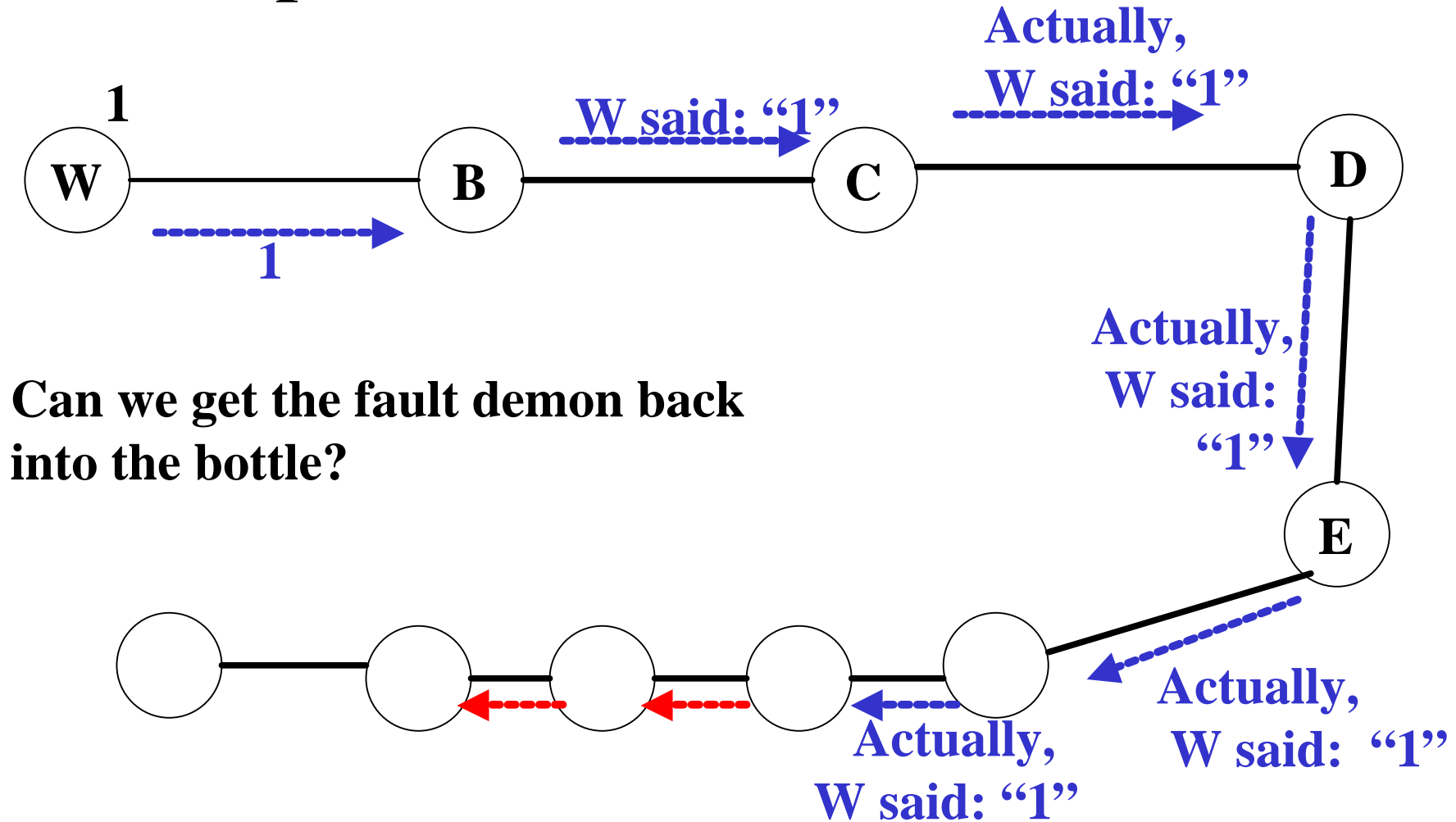
**When C recovers it can try to correct the error It has spread. However, the fault continues to Spread.**



# Bcast (vote sending) spreading faults problem under state faults

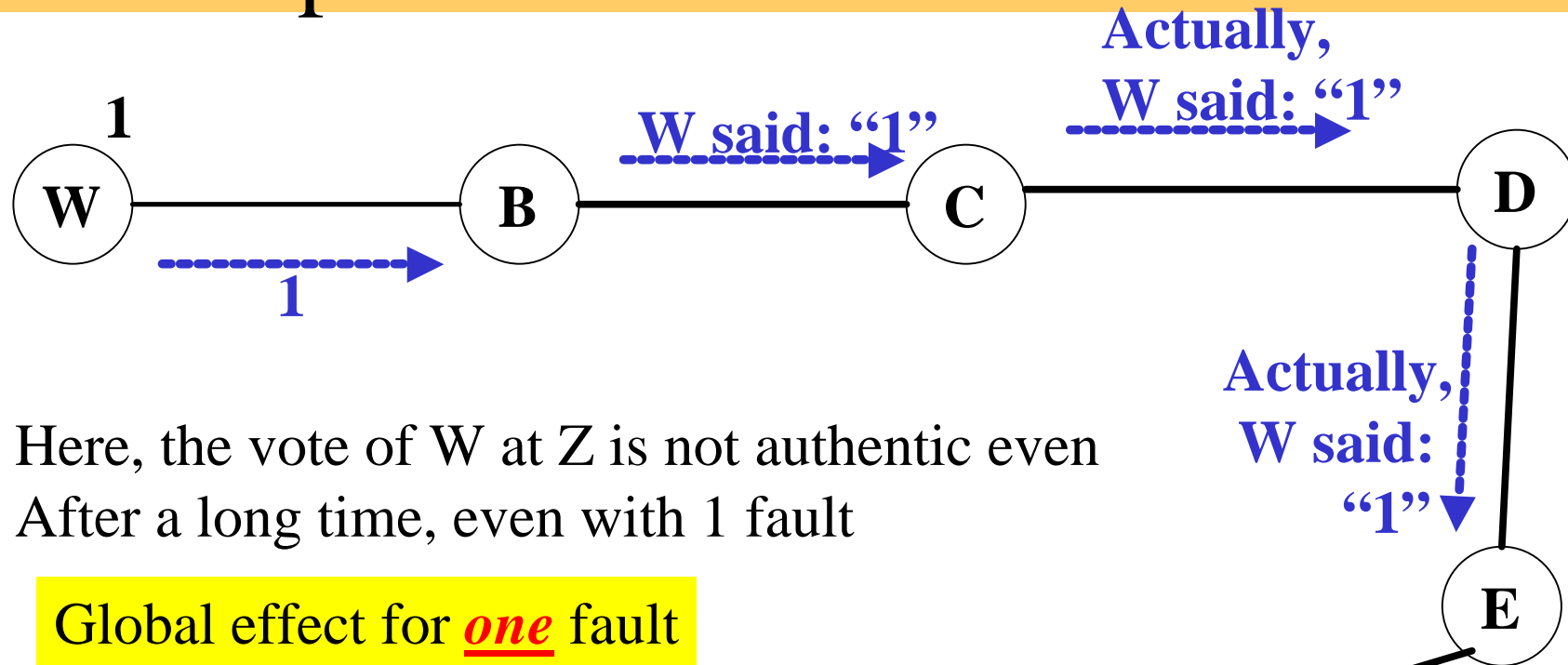


# Bcast (vote sending) spreading faults problem under state faults



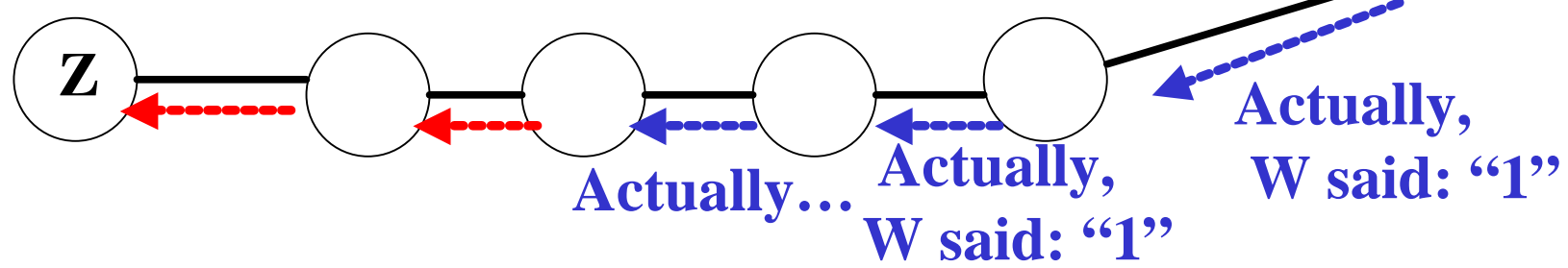


# Bcast (vote sending) spreading faults problem under state faults

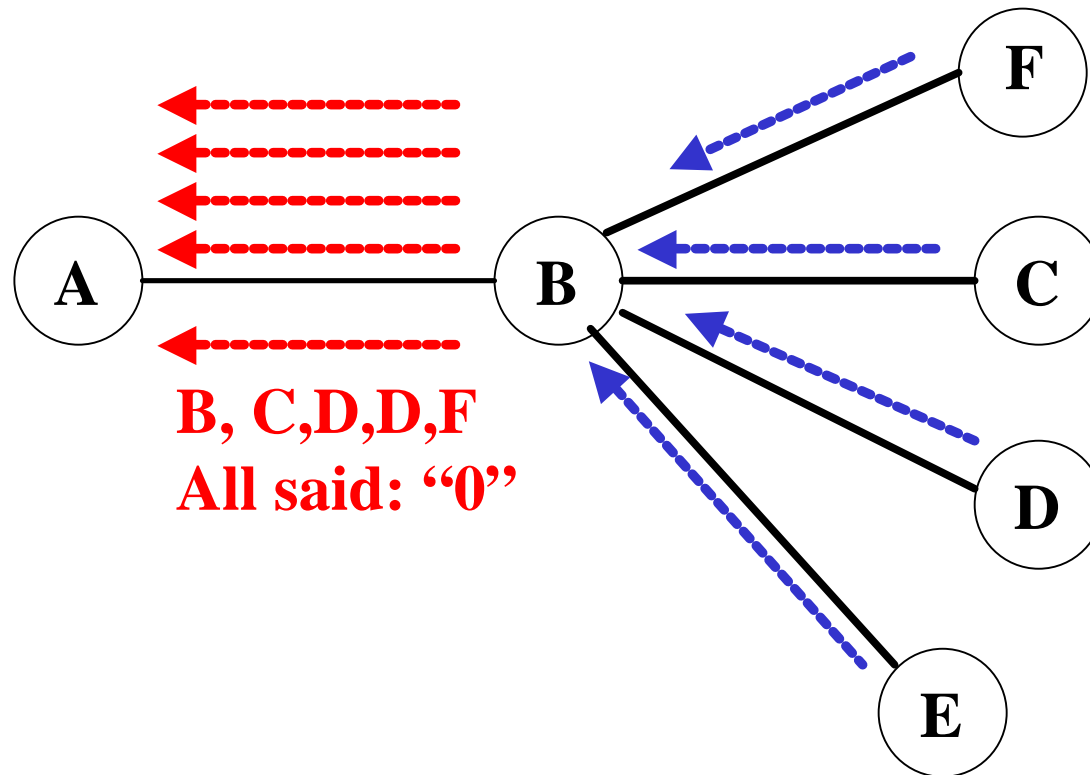


Here, the vote of W at Z is not authentic even  
After a long time, even with 1 fault

Global effect for one fault



# Another bcast spreading fault problem

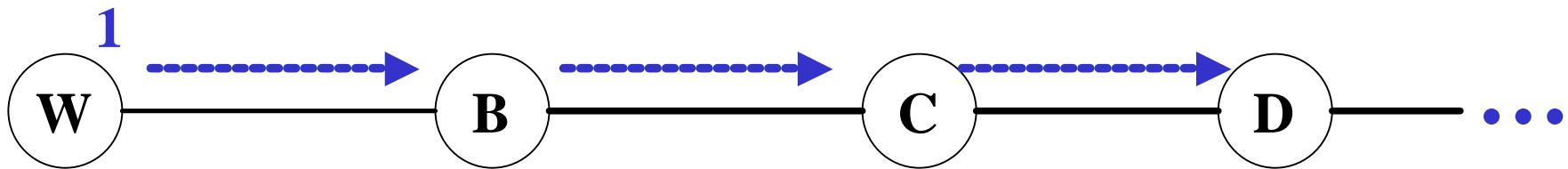


One faulty node can make the cotes of a majority non-authentic

# Sample technique: solving the bcast authenticity in $O(f)$ time

**Regulated Bcast [Kutten, Patt-Shamir]**

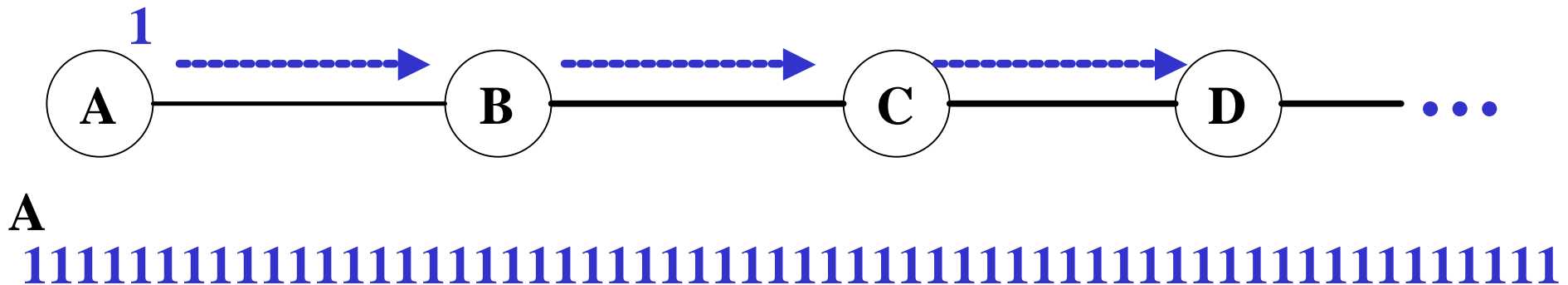
**Power Supply [Afek, Bremler]**



# Sample technique: solving the bcast authenticity in $O(f)$ time

Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]



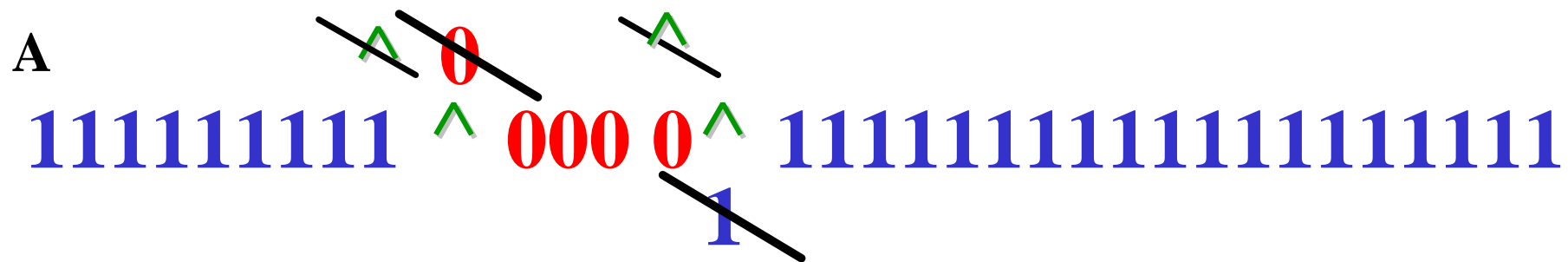




# Sample technique: solving the bcast authenticity in $O(f)$ time

Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]

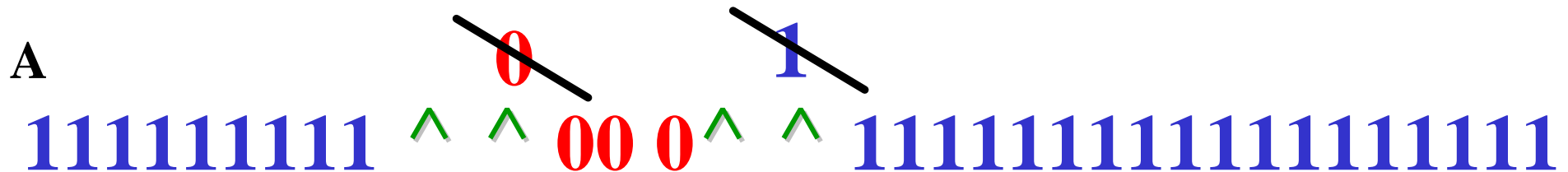


After **two** time units, the  $\wedge$  spreads twice as fast as the bcast (“0”s and “1”s).

# Sample technique: solving the bcast authenticity in $O(f)$ time

Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]



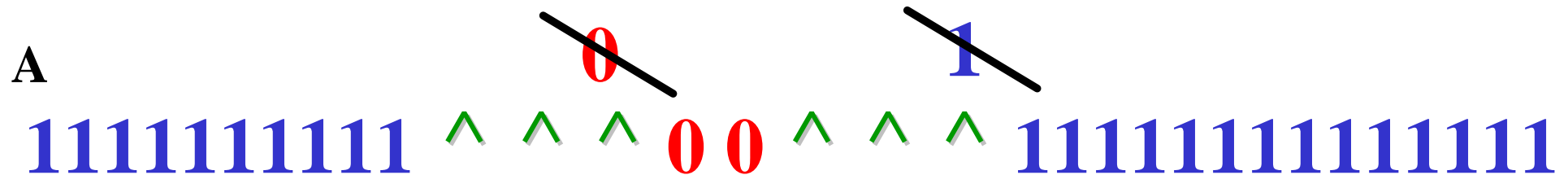
After **Three** time units, the <sup>^</sup> spreads twice as fast as the bcast (“0”s and “1”s).



# Sample technique: solving the bcast authenticity in $O(f)$ time

Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]

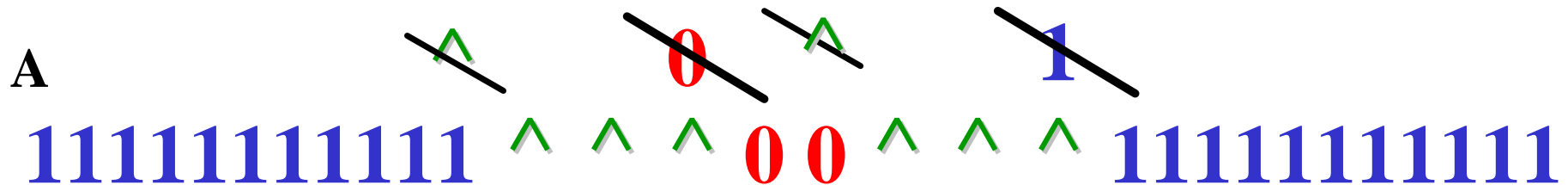


After **Five** time units, the ^ spreads twice as fast as the bcast (“0”s and “1”s).

# Sample technique: solving the bcast authenticity in $O(f)$ time

Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]

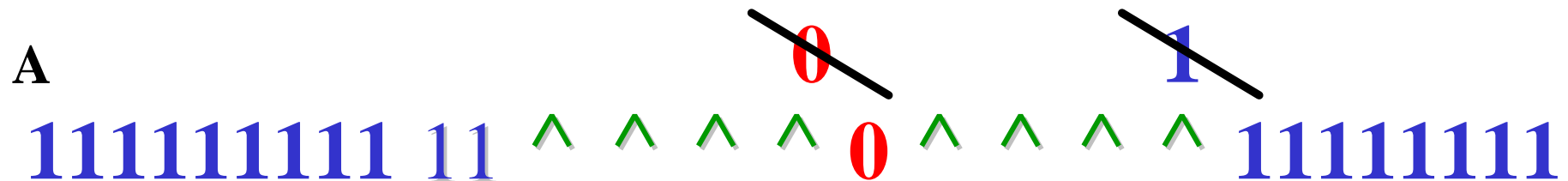


After **Six** time units, the  $\wedge$  spreads twice as fast as the bcast (“0”s and “1”s).

# Sample technique: solving the bcast authenticity in $O(f)$ time

Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]



After **SEVEN** time units, the ^ spreads twice as fast as the bcast (“0”s and “1”s).

# Sample technique: solving the bcast authenticity in $O(f)$ time

Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]



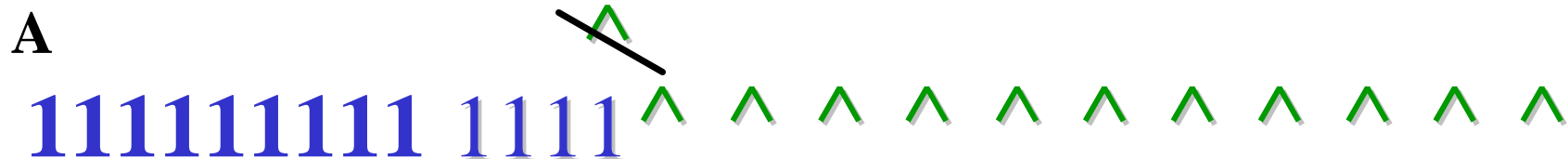
After **eight** time units, the  $\wedge$  spreads twice as fast as the bcast (“0”s and “1”s).



# Sample technique: solving the bcast authenticity in $O(f)$ time

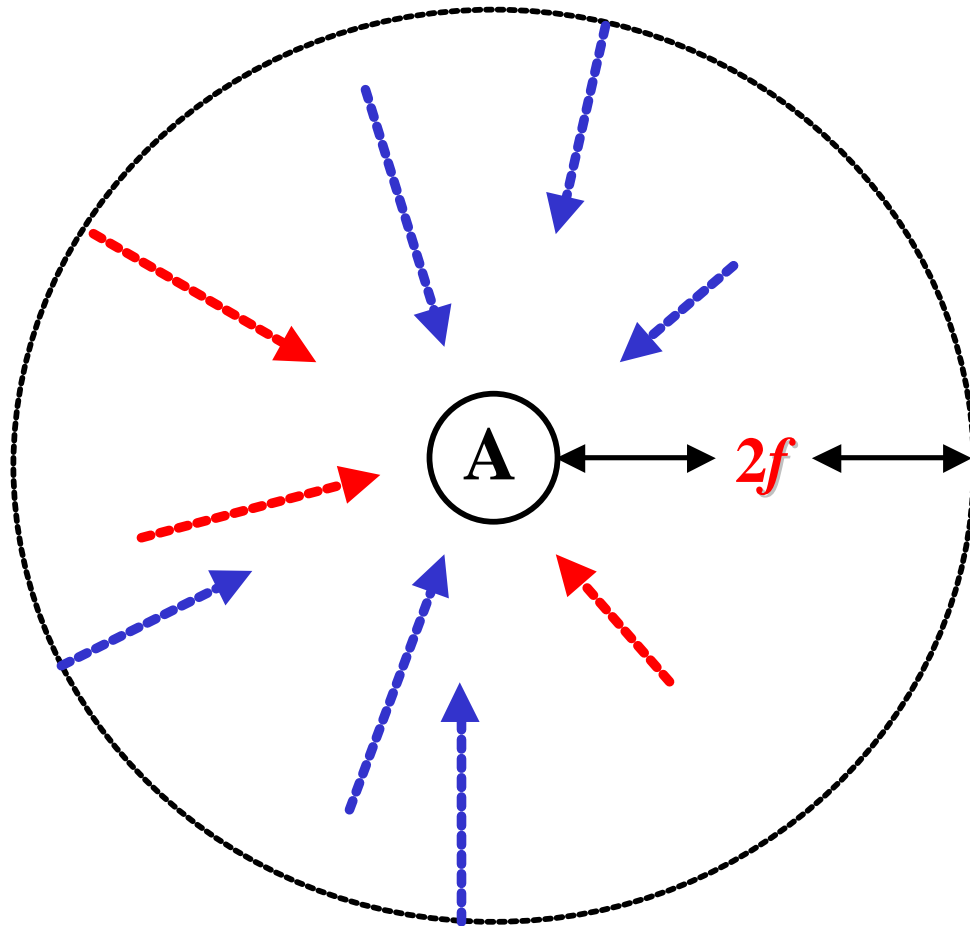
Regulated Bcast [Kutten, Patt-Shamir]

Power Supply [Afek, Bremler]



After 10 time units, the ^ replaces the un-authentic vote, while the authentic vote “1” continues to proceed at half speed.

# Recall the Stable Value Problem



(1) In  $O(f)$  time all non authentic values disappear in A.

**Impossible**  
after  $O(f)$

(2) In another  $O(f)$  time A knows the authentic votes of  $2f + 1$ .

(3) Majority of these votes are not faulty.

**A lot of recent related work.  
A very partial bibliography:**

**[Kutten, Peleg], [KP1]**

**[Ghosh, Gupta, Herman, Pamaraju]**

**[Afek, Dolev]**

**[Chlamtac,Pinter], [Dolev, Herman], [Naor, Stockmeyer]**

**[Arora, Zhang]**

**Beauquier, Genolini, Cournier, Datta, Petit, Viliain, Xin He**

**Error Confinement, Time Adaptive, Fault Local, Mending,**

**Fault Containment, Snap Stabilization, Local Stabilization**

**Many open problems!**