The Model & Basic Computations

The Model

Broadcast

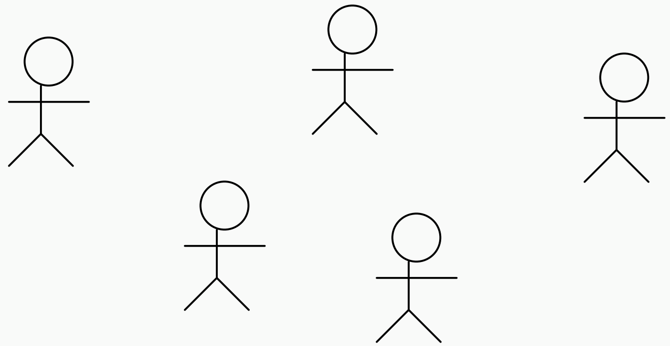
Spanning Tree Construction

Traversal

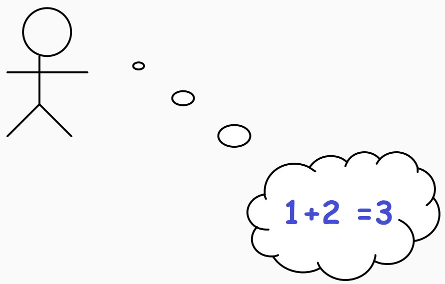
Wake-up

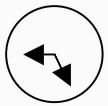
Distributed Environment

Multiplicity

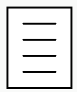


Autonomy





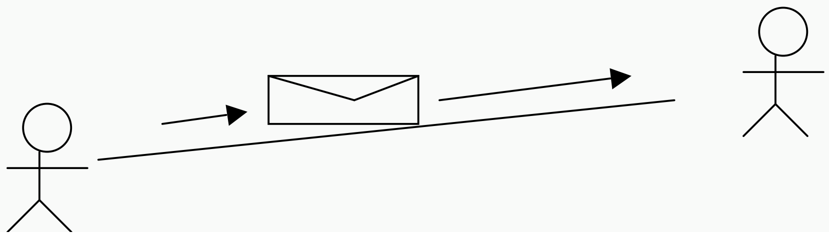
clock



memory

computing capabilities

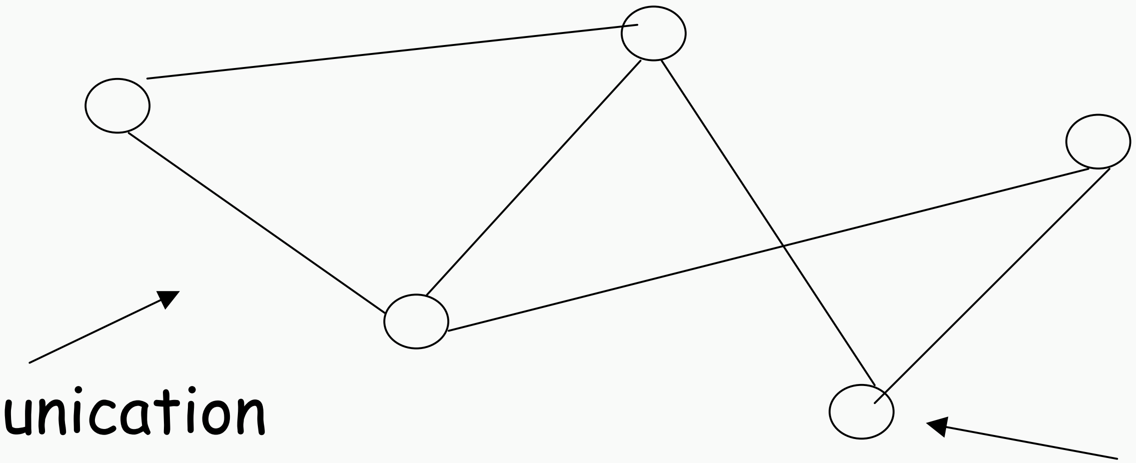
Interaction



typically by
exchange of messages

The Model

Collection of entities that communicate by exchanging messages

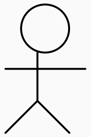


communication link

entity, node, site ...

(processor, process, object ....)

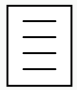
The Model

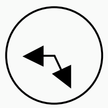
Entity: 

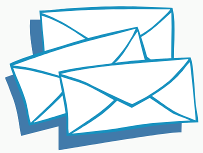
x

In memory: status(x)
value(x)

An entity can:

Access and change its
local memory 

Access and reset its local clock 

Send messages 

Perform local computations

Message

= finite sequence
of bits

Entity Behavior

S = finite set of states an entity can be in

(ex: status(x) $\in$ {waiting, sleeping, processing})

E = finite set of events that could occur

Possible events:

- receiving a message
- clock ring
- spontaneous impulse

The behavior of an entity is reactive: triggered by events

B(x) = ACTION of an entity x in response to an EVENT

Entity Behavior

State x Event $\longrightarrow$ Action

Action: sequence of activities, e.g.,

- computing
- sending message
- change state

the action is atomic

the activities cannot be interrupted

The behavior of an entity is

DETERMINISTIC
(non-ambiguous)

COMPLETE
($\forall$ (state, event) $\exists$ an action)

System Behavior

$$B = \{ B(x) : x \in E\}$$

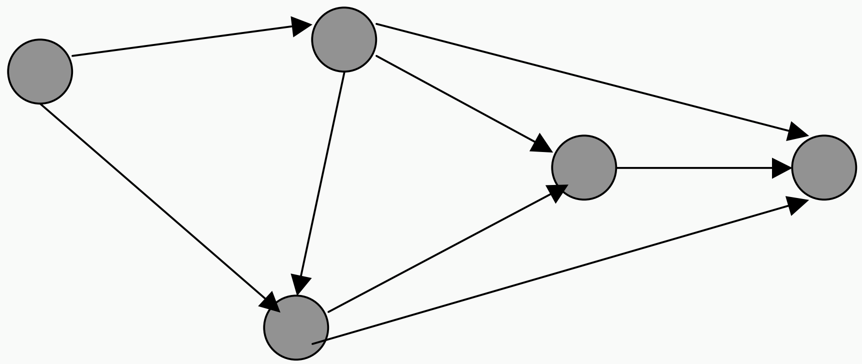
A system is SYMMETRIC (or homogeneous)
if all the entities have the same behavior

$$B(x) = B(y), \forall\, x,y \in E$$

Observation.

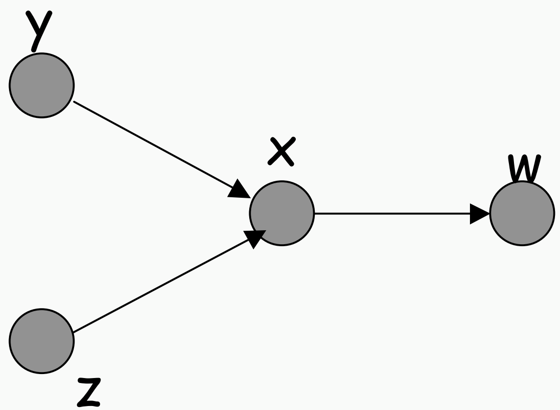
Every system can be made symmetric

Communication Network:



Communication

Point-to-point Model



$N_o(x)$ = out-neighbors of entity x
$N_i(x)$ = in-neighbors of entity x

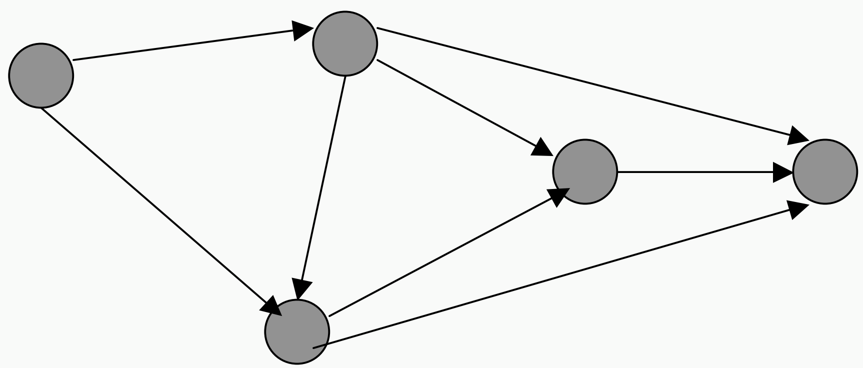
$$N(x) = N_o(x) \cup N_i(x)$$

Graph describing the
COMMUNICATION TOPOLOGY

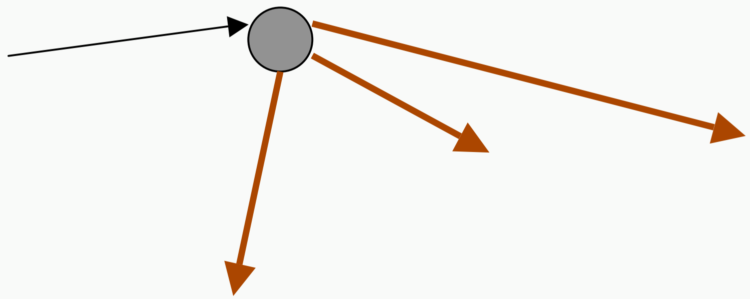
$G = (V, A)$

V: Entities

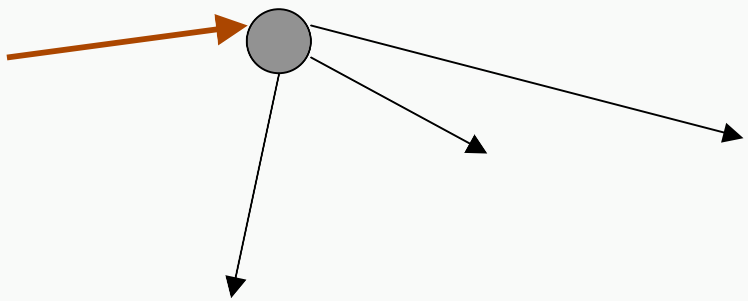
A: Arcs defined by N



An entity x can send a message only to its out-neighbors $N_o(x)$



and receive from the in-neighbours $N_i(x)$



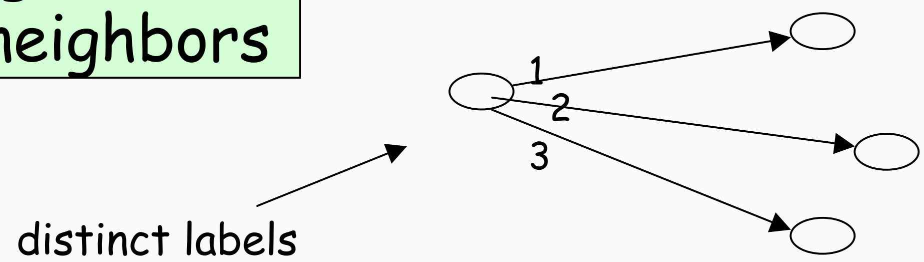
Axioms

Finite Transmission Delays

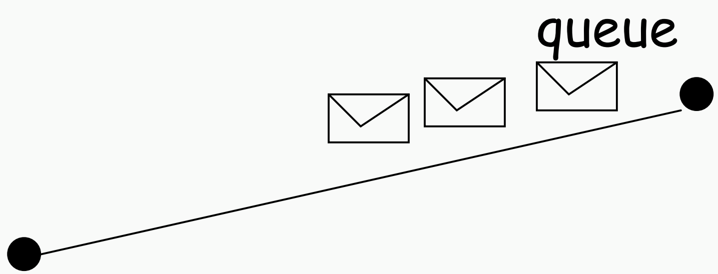
In absence of faults a message reaches its destination in finite time

Local orientation

Each entity distinguishes among his in/out neighbors



Restrictions of the model



Communication Restriction:

Message Ordering

In absence of failures, msgs transmitted along the same link arrive in the same order.

Restrictions of the model

Reliability Restrictions:

1. *Guaranteed delivery:*
   Any message that is sent will be received uncorrupted
2. *Partial Reliability:*
   There will be no failures during the computation
3. Total Reliability:
   No failures have occurred nor will occur

Restrictions of the model

Communication restriction:

Bidirectional Links

$\forall\ x, N_i(x) = N_o(x)$

Topological restriction:

The graph G is strongly connected

Restrictions of the model

Time restriction:

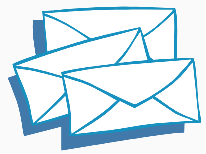
Bounded Communication Delay:
There exists a constant Δ such that, in absence of failures, the communication delay of any message on any link is at most Δ

Unitary Communication Delay: In absence of failures, the communication delay is always one unit of time

Synchronized clocks: All local clocks are incremented by one unit symultaneously and interval are constant

Complexity measures - Performance

1. Amount of communication



messages exchanged
bits exchanged

point of view
of SYSTEM

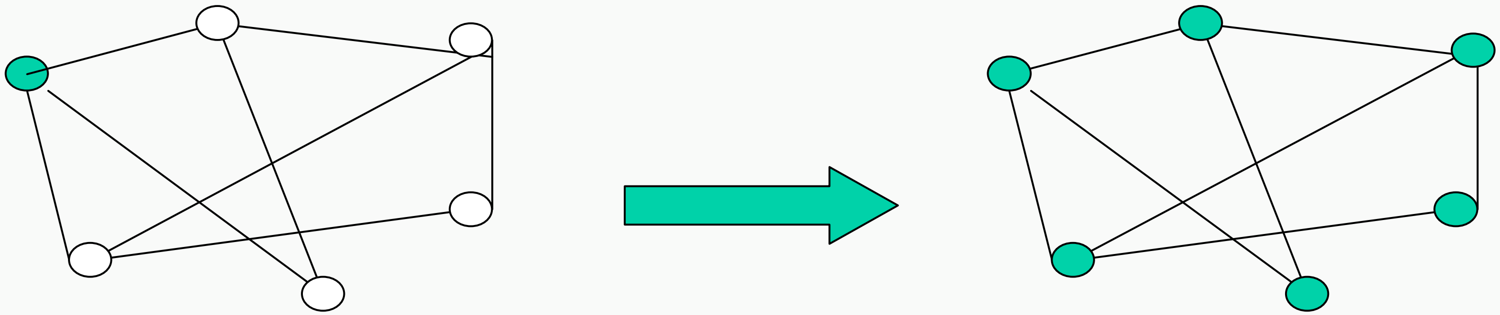
2. Time

Communication delays are in general unpredictable !!!

Ideal time:
1 unit of time to transmit 1 message

point of view
of USER

Example - Broadcast (by Flooding)

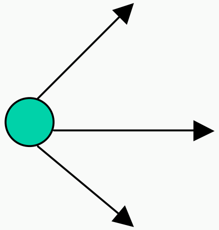


Assumptions:

G is connected
No failures
Bidirectional links
Unique Initiator

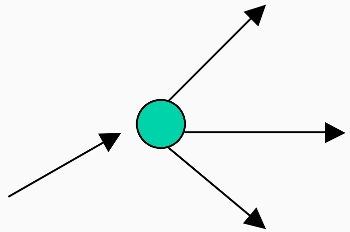
The idea: If an entity knows something, it sends the info to its neighbours

One entity is INITIATOR, the others are SLEEPING



```
INITIATOR
spontaneously
      send(I) to N(x)
```

```
SLEEPING
receiving(I)
      send(I) to N(x)
```



The idea: If an entity knows something, it sends it to its neighbours except the sender

```
INITIATOR
spontaneously
        send(I) to N(x)
```

```
SLEEPING
receiving(I)
      send(I) to N(x) - {sender}
```

not correct !

S = {initiator, sleeping, done}

Algorithm for node x:

```
INITIATOR
spontaneously
        send(I) to N(x)
        become(DONE)
```

```
SLEEPING
receiving(I)
        send(I) to N(x) - {sender}
        become(DONE)
```

Algorithm for node x:

```
INITIATOR
spontaneously
      send(I) to N(x)
      become(DONE)
```

```
SLEEPING
receiving(I)
      send(I) to N(x) - {sender}
      become(DONE)
```

```
DONE
```

Algorithm for node x:

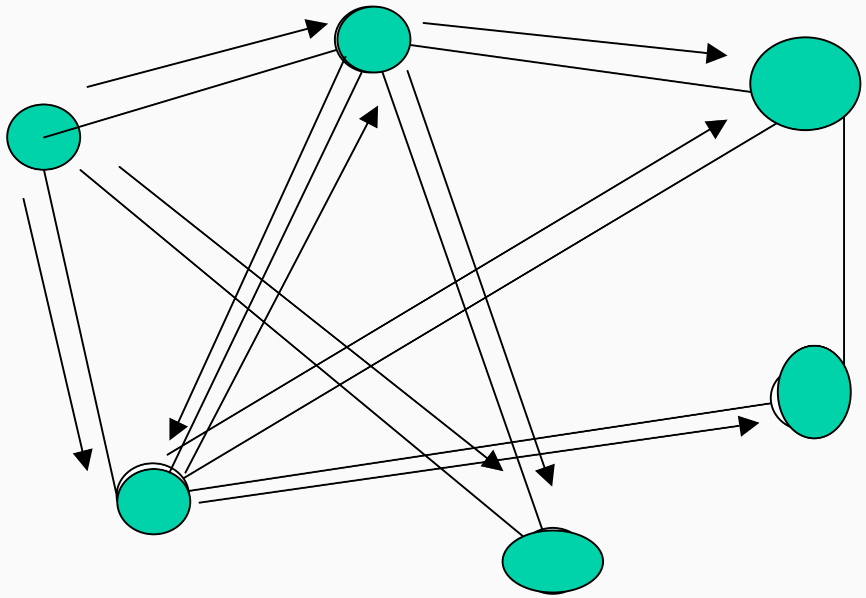
```
If INITIATOR
     spontaneously
          send(I) to N(x)
          become(DONE)


If SLEEPING
     receiving(I)
          send(I) to N(x) - {sender}
          become(DONE)



If DONE
         do-nothing
```

Example



Complexity - Worst Case

m = number of links

Messages: $\leq$ 2 on each link

$\longrightarrow \leq 2m$

$O(m)$

More precisely:

Let s be the initiator

$$|N(s)| + \sum_{x \neq s} (|N(x)|-1)$$

$$= \sum_{x} |N(x)| - \sum_{x \neq s} 1$$

$$= \mathbf{2m - (n-1)}$$

$$\sum_{x} |N(x)| = 2m$$

Complexity - Ideal Time

Time: (ideal time)

$$\text{Max}\{D(x,e)\} = \text{eccentricity}$$
$$\leq n-1$$

$$O(n)$$

Correctness

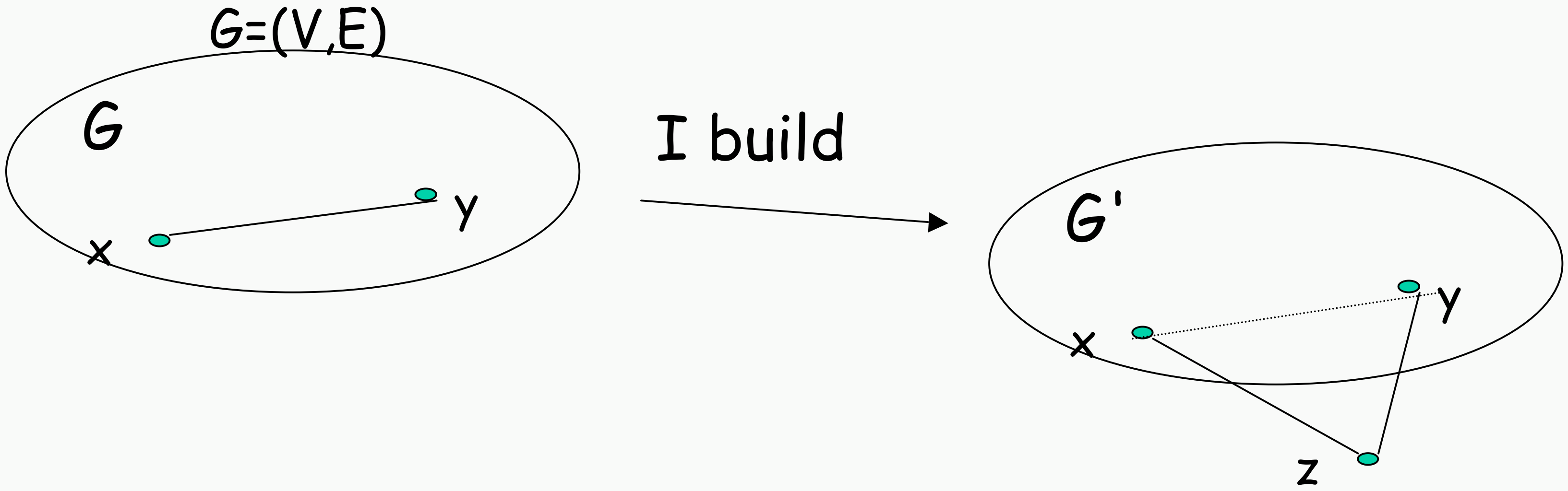
It follows from the fact that
G is connected

Lower Bounds for FLOODING

We want to prove that a lower bound on the number of messages is $\Omega(m)$

By contradiction:

Let e = (x,y) be a link where no messages are sent.



$G' = (V \cup \{z\}, E-e \cup \{(x,z),(y,z)\})$

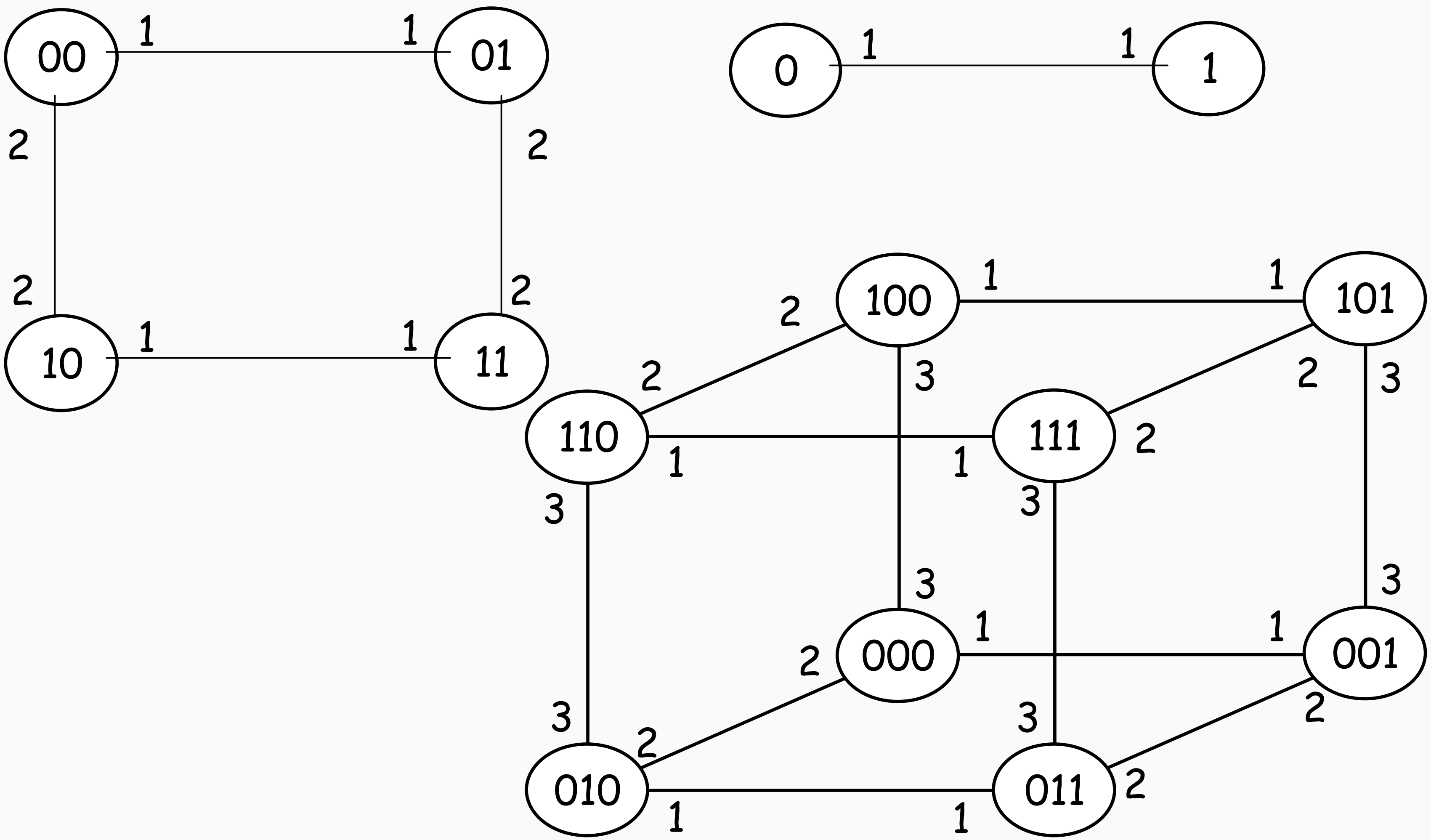
Execute the same algorithm on G'

In specific topologies flooding can be avoided and broadcast can be more efficient.

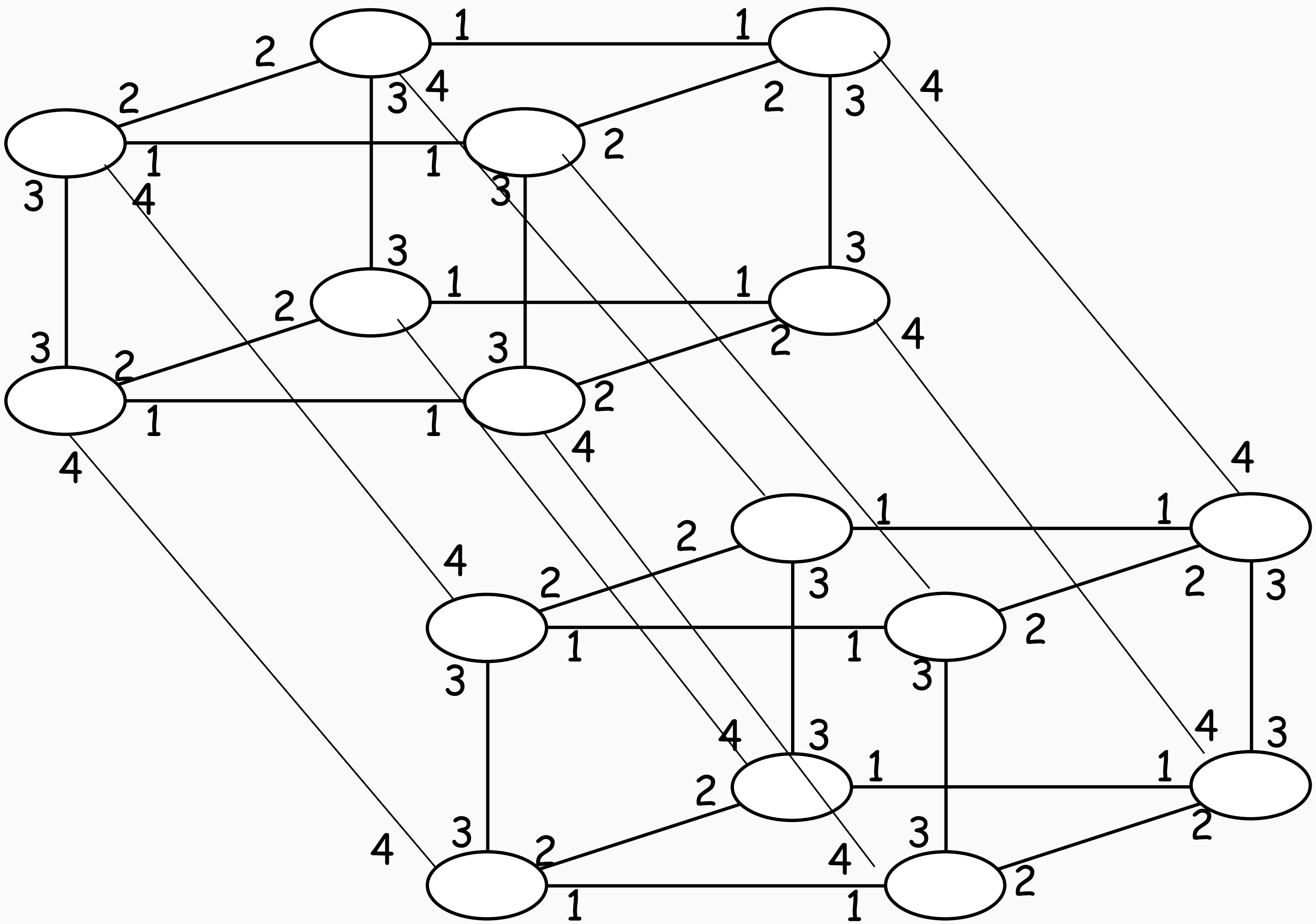
What is the complexity of flooding in a complete graph ?
How can it be done more efficiently ?

What is the complexity of flooding in a tree ?
Can it be done more efficiently ?

Example: The labeled hypercube



Each link between two nodes is labeled by the dimension of the bit by which the nodes' name differ.



A hypercube of dimension k has $n = 2^k$ nodes

Each node has k links

➔ $m = n\,k/2 = O(n \log n)$

Simple Broadcast

1) The initiator sends the message to all its neighbours
2) A node receiving the message from link l, sends it only to links with label $l' < l$

Complexity: n-1 (OPTIMAL)

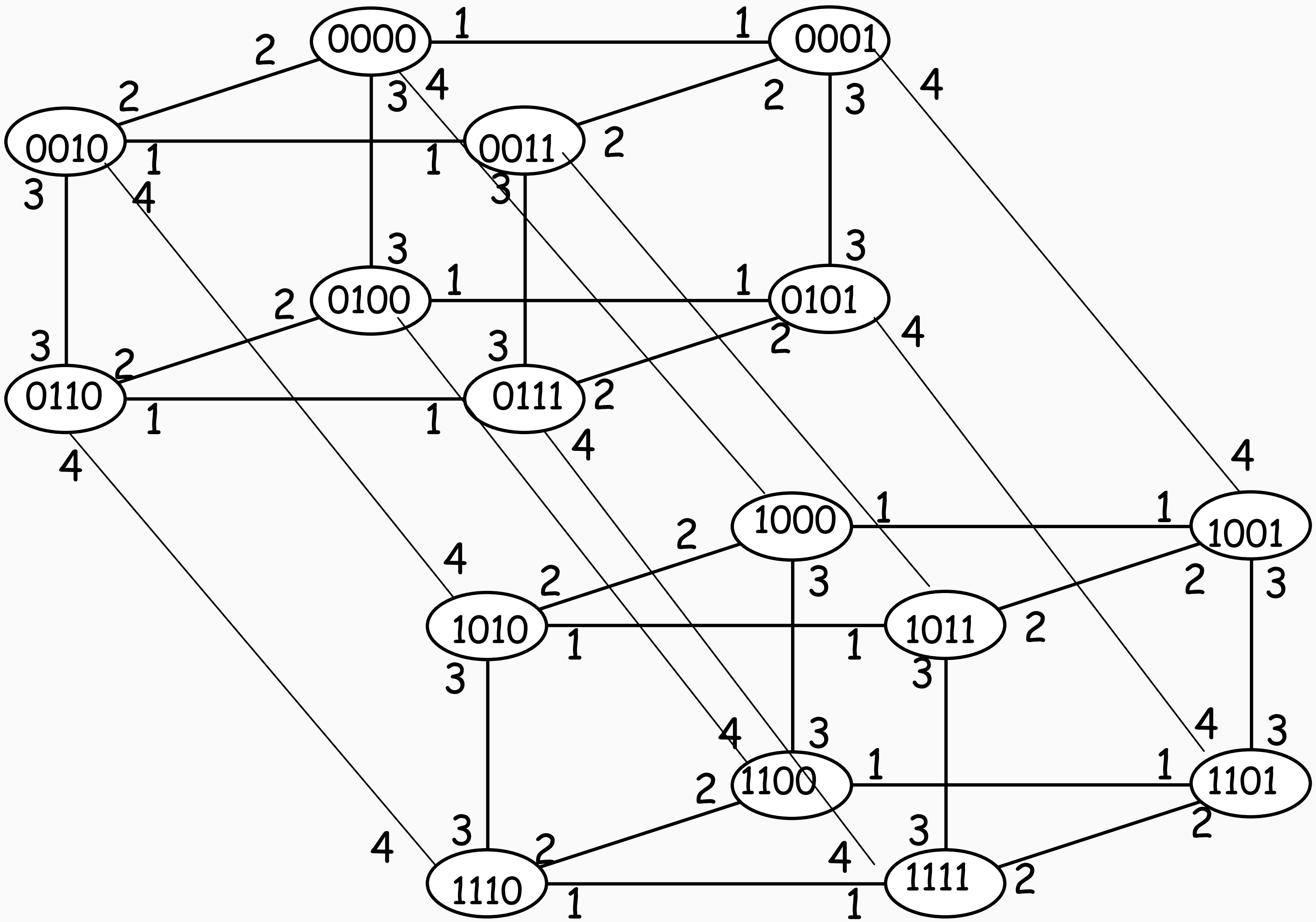
Because every entity receives the info only ONCE.

Correctness

Every node is touched

Based on the lemma:

For each pair of nodes x and y there exists a path of decreasing labels


0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

In Special Topologies

General Flooding: 2m - (n-1)

Ad-hoc algorithm in hypercube: (n-1)

Ad-hoc algorithm in complete network: (n-1)

In the tree Flooding is optimal: (n-1)

Observations:

1) Dense networks = more messages
(ex. in complete networks $m = n (n-1)$ ...)
2) It is optimum in acyclic graphs

Idea: to solve broadcast.

1. Build a spanning tree of G

2. Execute flooding

→

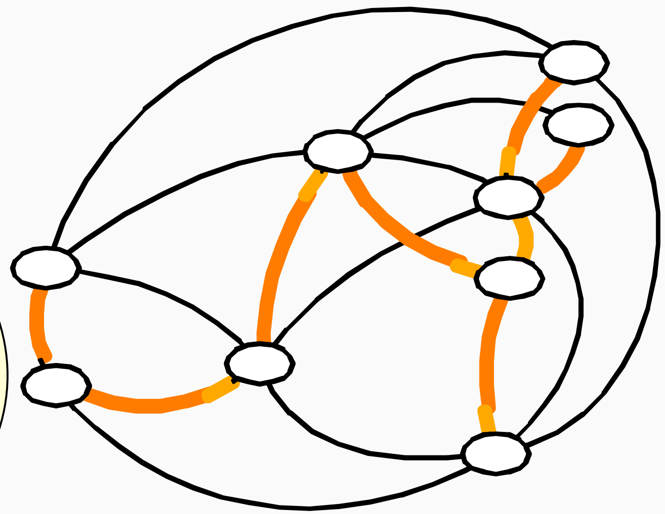
Spanning Tree construction Problem

Spanning Tree Construction

A spanning tree T of a graph G = (V,E) is an acyclic subgraph of G such that T=(V,E') and E' $\subset$ E.

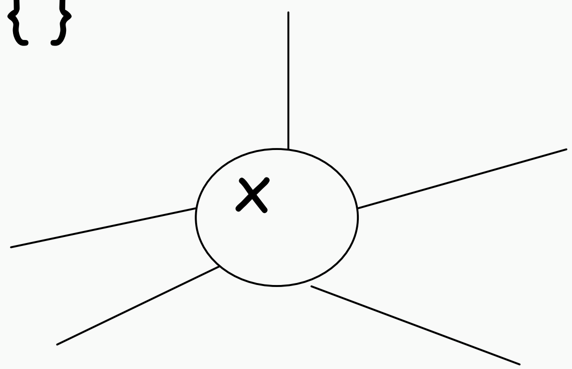
Assumptions:

bidirectional links
no failures
G connected
single initiator



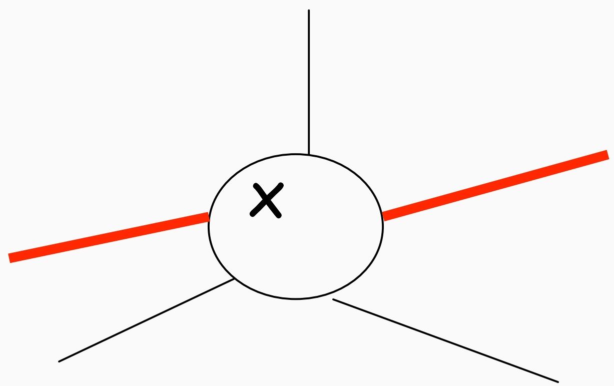
Protocol SHOUT

Initially: $\forall$ x, Tree-neighbors(x) = { }

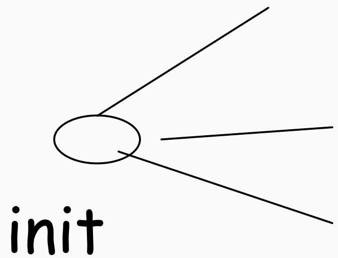


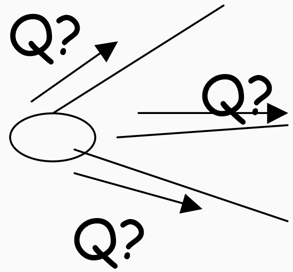
At the end:

$\forall$ x, Tree-neighbors(x) = {links that belong to the spanning tree }



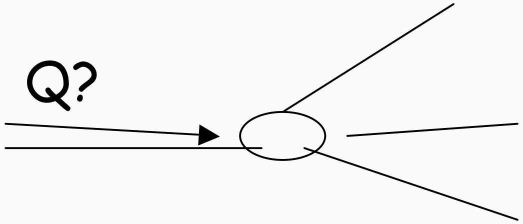
1.



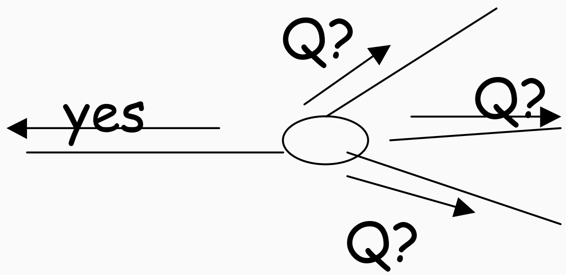


Q? = do you want to be
my neighbour
in the spanning tree ?

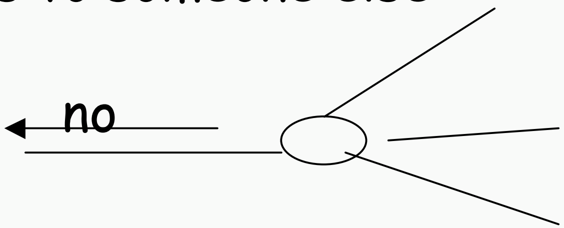
2.



If it is the first time:



If I have already answered
yes to someone else:



States S={INITIATOR, IDLE, ACTIVE, DONE}
Sinit = {INITIATOR, IDLE}
Sterm = {DONE}

INITIATOR
Spontaneusly

    root:= true
    Tree-neighbours := { }
    send(Q) to N(x)
    counter:=0
    become ACTIVE

IDLE
receiving(Q)

    root:= false
    parent := sender
    Tree-neighbours := {sender}
    send(yes) to sender
    counter := 1
    if counter = |N(x)| then
        become DONE
    else
        send(Q) to N(x) – {sender}
        Become ACTIVE

ACTIVE

receiving(Q)

send(no) to sender

receiving(yes)

Tree-neighbours:=

Tree-neighbours ∪ sender

counter := counter +1

if counter = |N(x)|

become DONE

receiving(no)

counter := counter +1

if counter = |N(x)|

become DONE

Termination and Correctness

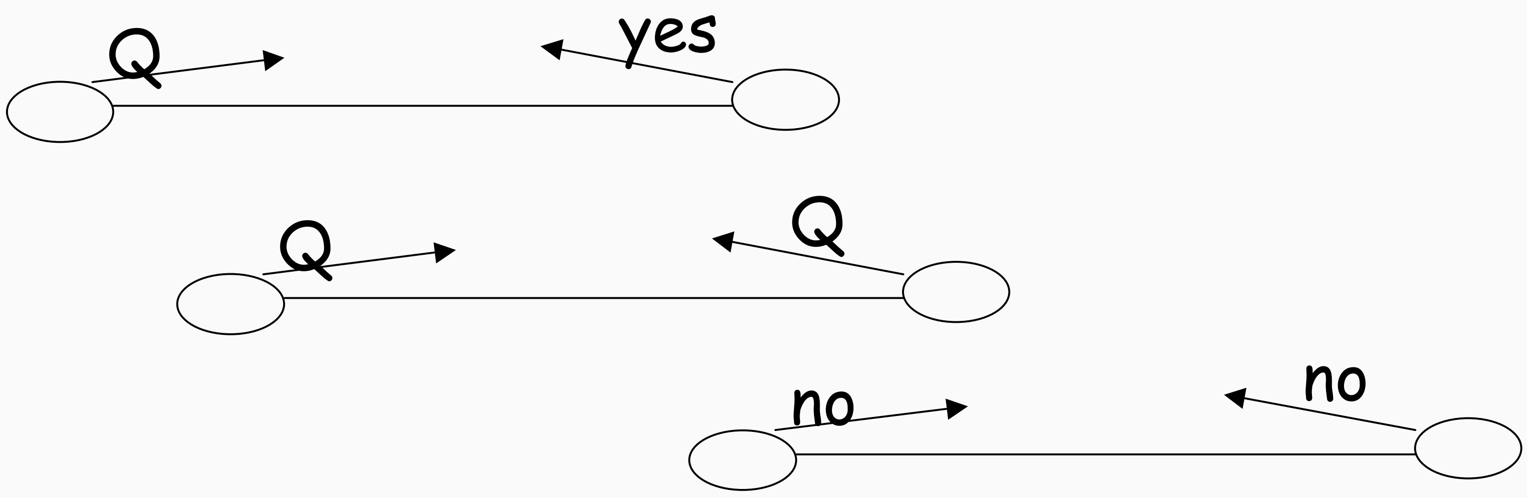
Notice: SHOUT = FLOOD +REPLY

If x is in Tree-neighbours of y, y is in Tree-neighbours of x
If x send YES to y, then x is in Tree-neighbour of y
and is connected to the initiator by a chain of YES
Every x (except the initiator) sends exactly one YES

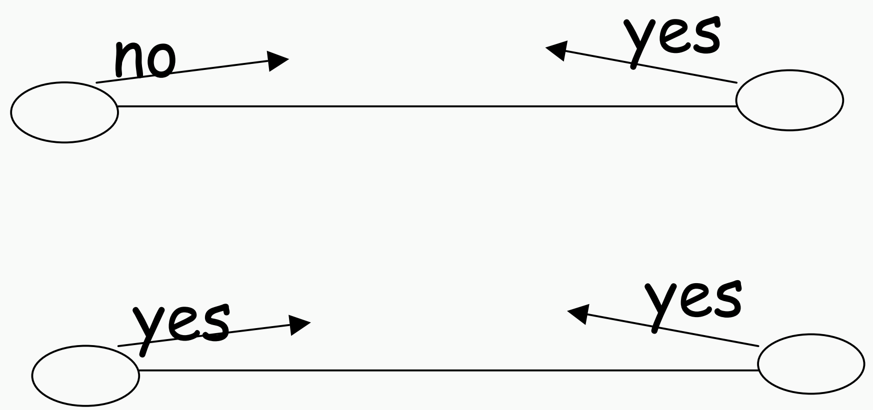
The spanning graph defined by the Tree-neighbour relation is connected and contains all the entities

Notice: local termination

Possible situations



Impossible situations



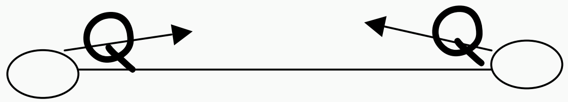
Complexity - worst case

Total n. of Q:

$\leq$ 2 on each link

TOT: $\leq 2m$

(n-1)



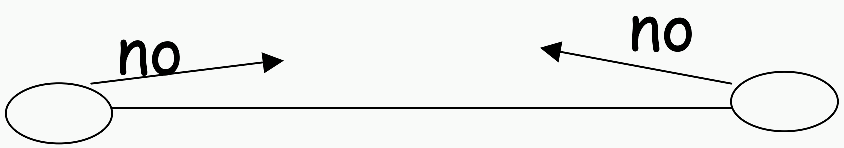
m -(n-1)

only one Q on the ST links

Exactly:

$$2(m-(n-1)) + (n-1)$$
$$= 2m - n + 1$$

Total n. of NO:



$\leq$ 2 on each link

TOT: $\leq 2m$

Exactly: $2(m - (n-1))$

as many as Q---Q

Total n. of YES:

$\leq$ 1 on each link of the ST

Exactly: $(n-1)$

$$2m - n + 1 + 2(m - (n-1)) + n-1$$
$$= 2m -n +1+2m -2n +2 +n - 1$$
$$= 4m -2n + 2$$

Messages(SHOUT) = 2 M(FLOOD)

$\Omega(m)$ is a lower bound also in this case

Spanning Tree Construction

Without "NO"

States S={INITIATOR, IDLE, ACTIVE, DONE}
Sinit = {INITIATOR, IDLE}
Sterm = {DONE}

INITIATOR
Spontaneusly

```
root:= true
Tree-neighbours := { }
send(Q) to N(x)
counter:=0
become ACTIVE
```

IDLE
receiving(Q)

```
root:= false
parent := sender
Tree-neighbours := {sender}
send(yes) to sender
counter := 1
if counter = |N(x)| then
        become DONE
else
        send(Q) to N(x) – {sender}
        become ACTIVE
```

ACTIVE

receiving(Q) **(to be interpreted as NO)**

counter := counter +1
if counter = |N(x)|
    become DONE

receiving(yes)
Tree-neighbours:=
    Tree-neighbours $\cup$ {sender}
counter := counter +1
if counter = |N(x)|
    become DONE

Spanning Tree Construction

With Notification

States S={INITIATOR, IDLE, ACTIVE, DONE}
Sinit = {INITIATOR, IDLE}
Sterm = {DONE}

INITIATOR
Spontaneusly

```
root:= true
Tree-neighbours := { }
send(Q) to N(x)
counter:= 0
ack-counter:= 0
become ACTIVE
```

IDLE
receiving(Q)

```
root:= false
parent := sender
Tree-neighbours := {sender}
send(yes) to sender
counter := 1
ack-counter:= 0
if counter = |N(x)| then
        CHECK
else
        send(Q) to N(x) – {sender}
become ACTIVE
```

ACTIVE

receiving(Q)

counter := counter +1

if counter = |N(x)| and not root then

CHECK

receiving(yes)

Tree-neighbours:=

Tree-neighbours ∪ {sender}

counter := counter +1

if counter = |N(x)| and not root then

CHECK

ACTIVE (cont)

receiving(Ack)

ack-counter:= ack-counter +1

if counter = |N(x)| /* indicate tree-neighbors is done

if root then

if ack-counter = |Tree-neighbours|

send(Terminate) to Tree-neighbours

become DONE

else if ack-counter = |Tree-neighbours| - 1

send(Ack) to parent

receiving(Terminate)

send(Terminate) to Children

become DONE

CHECK

If I am a leaf

send(Ack) to parent

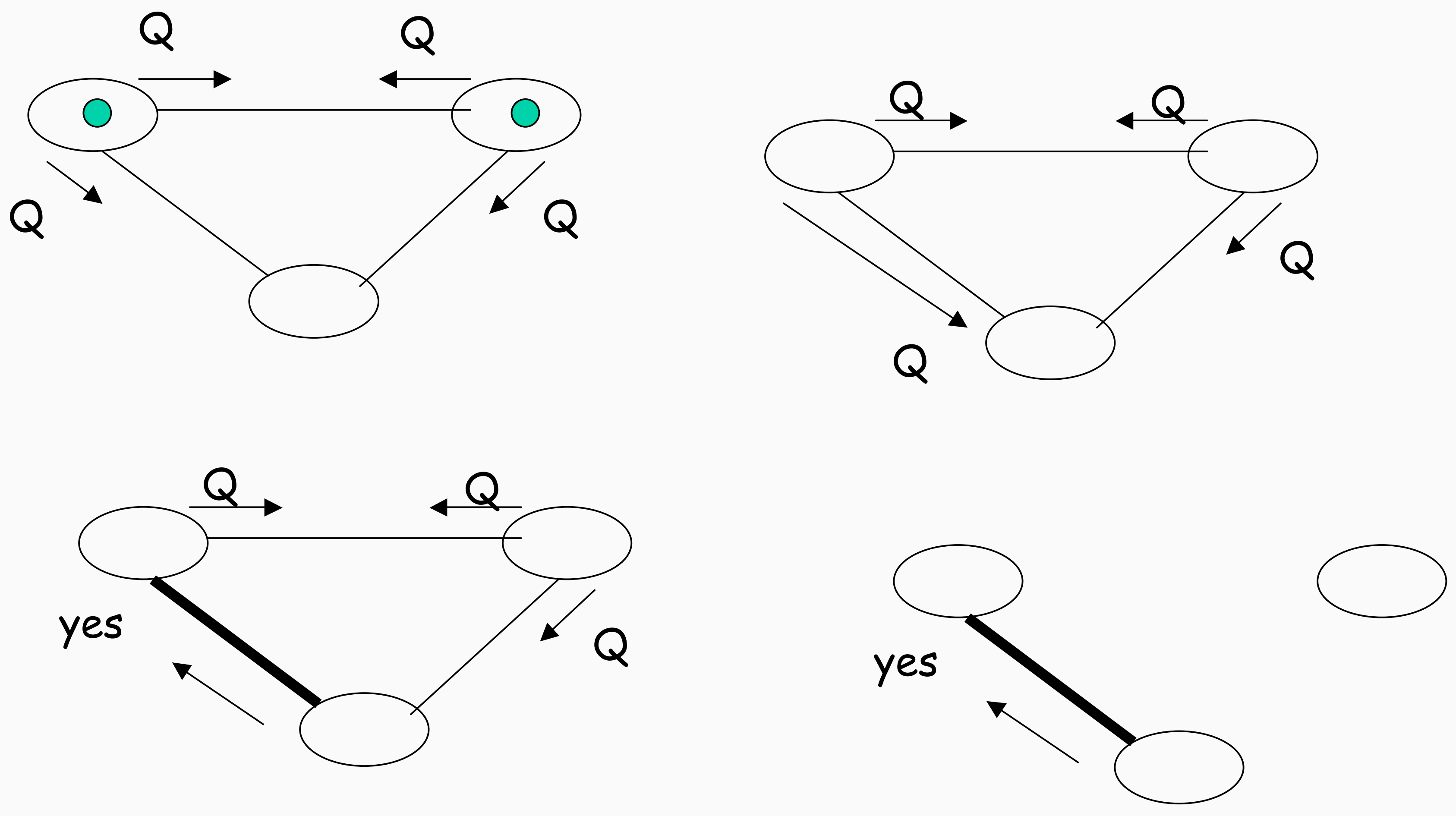
CHECK

Children:= Tree-neighbours – {parent}

if Children = emptyset then

send(Ack) to parent

What happens if there are multiple initiators ?



An election is needed to have a unique initiator.

NOTE: Election is impossible if the nodes do not have distinct IDs

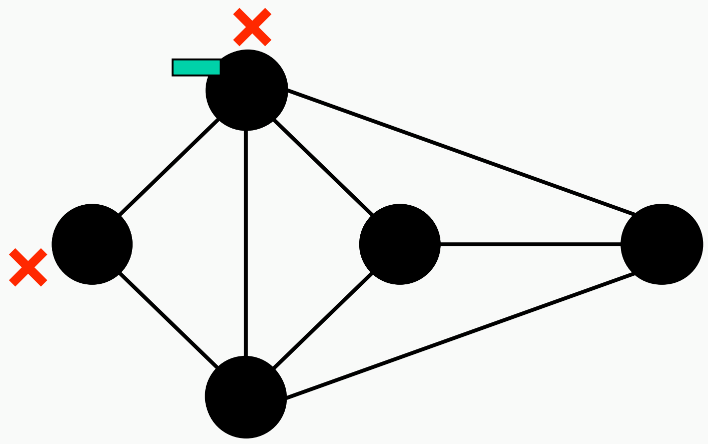
Or:
Another protocol has to be devised.

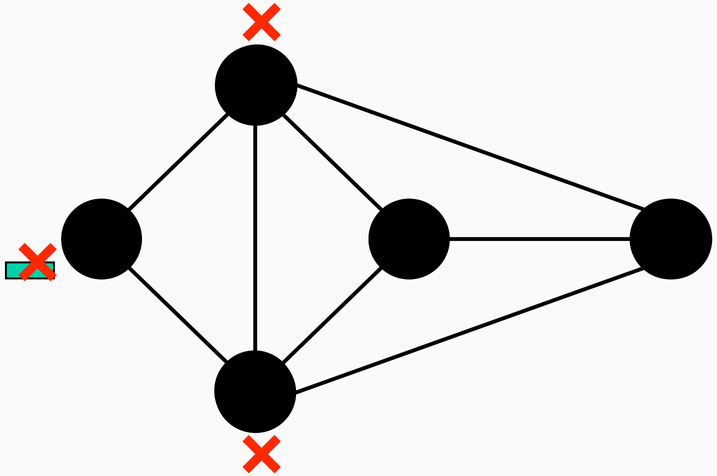
Traversal
Depth First Search

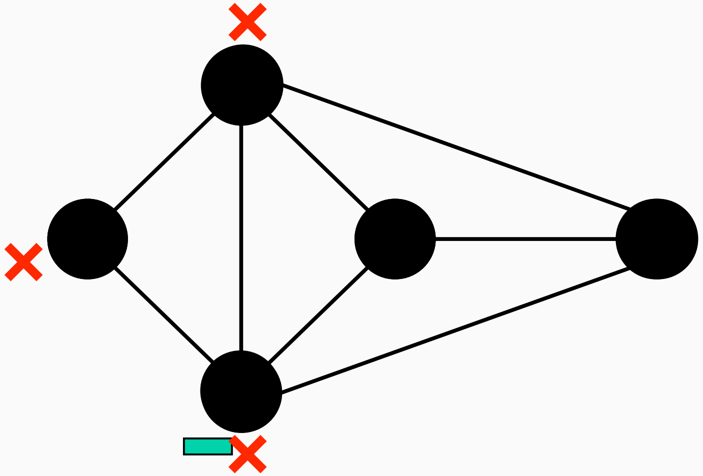
Assumptions

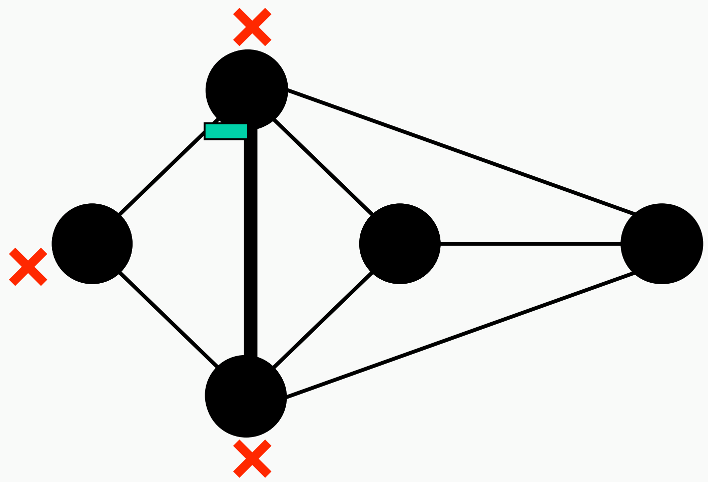
Single initiator
Bidirectional links
No faults

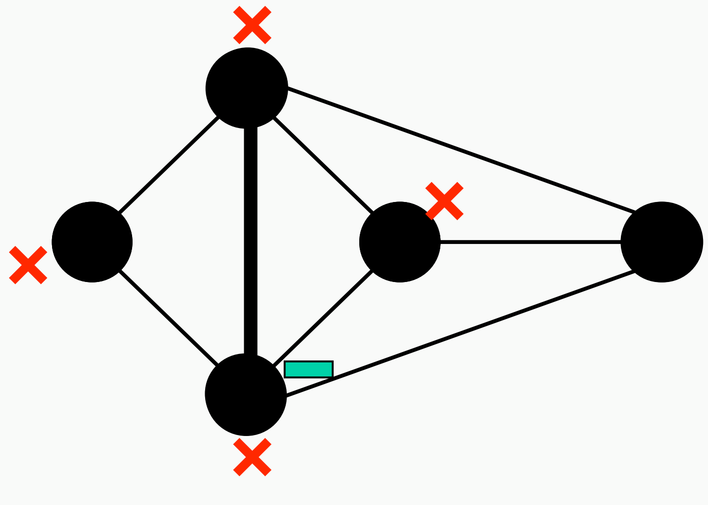
S = {INITIATOR, SLEEPING, ACTIVE, DONE}

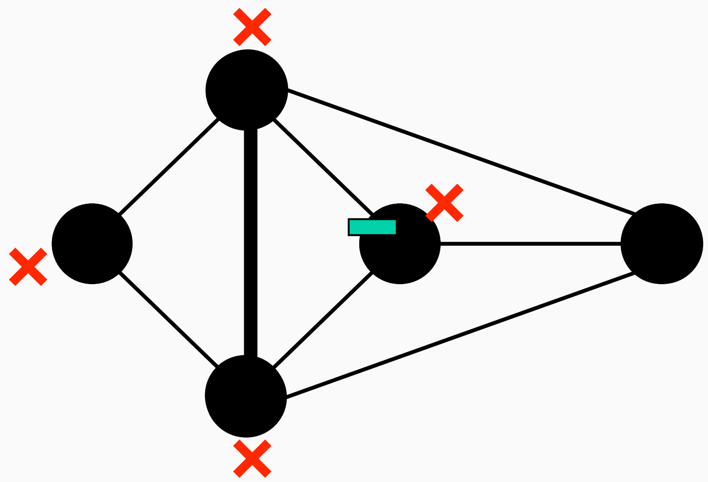


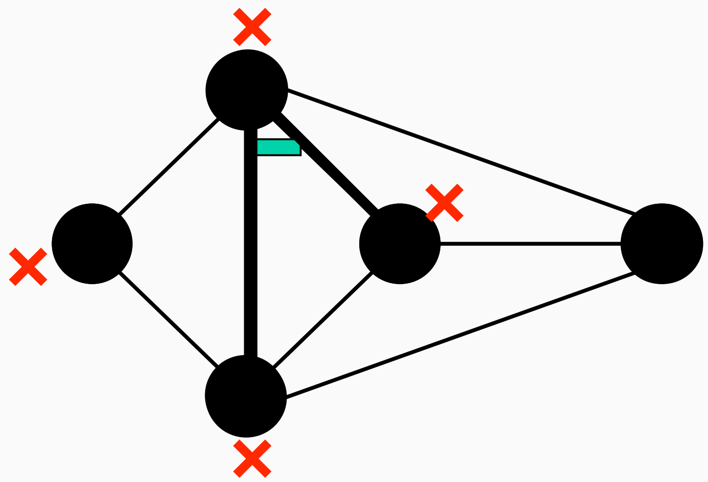


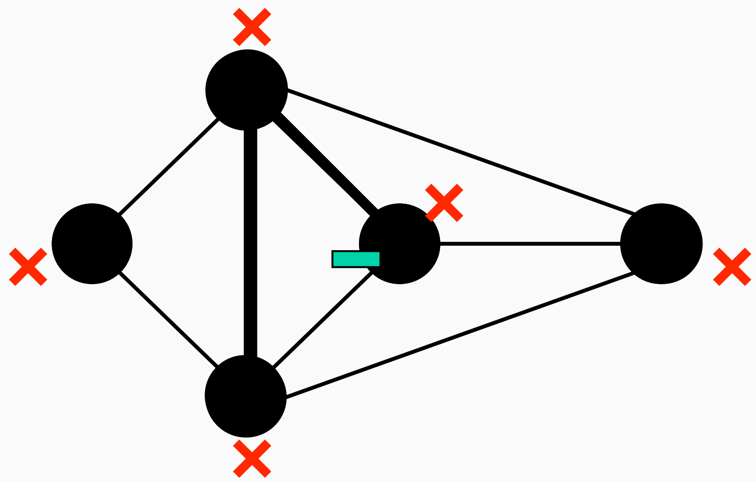


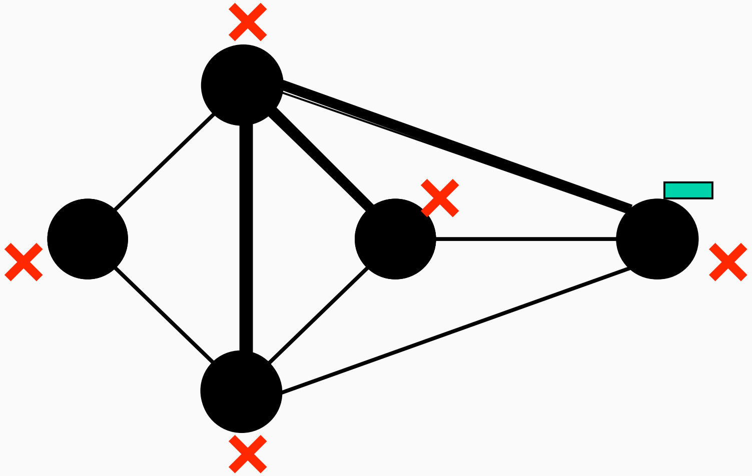


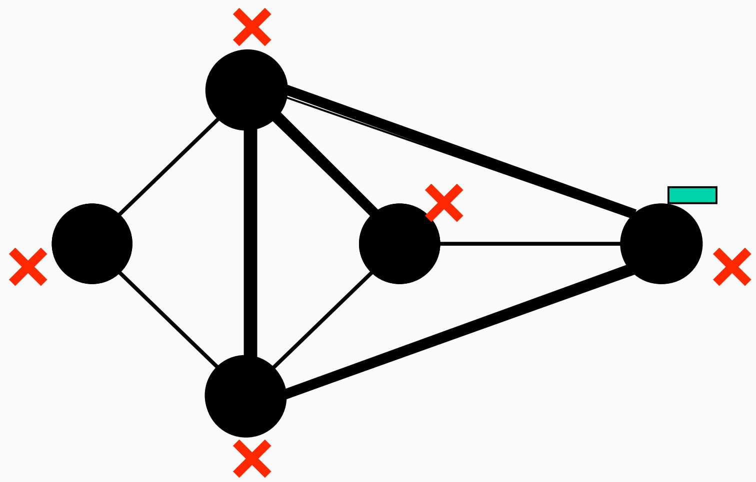


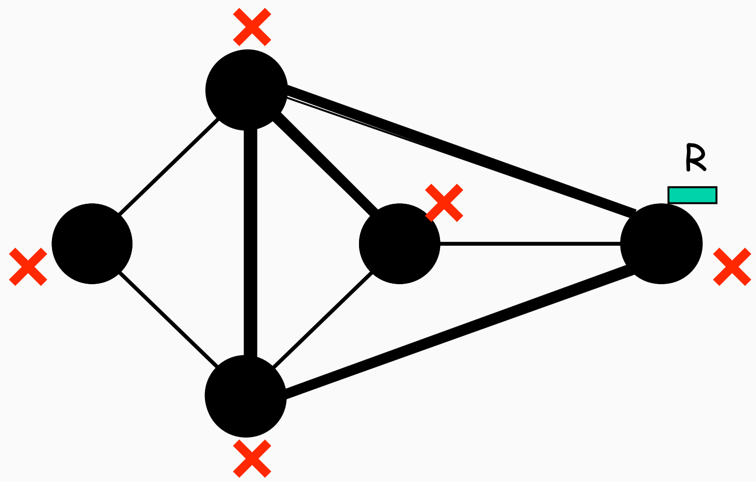


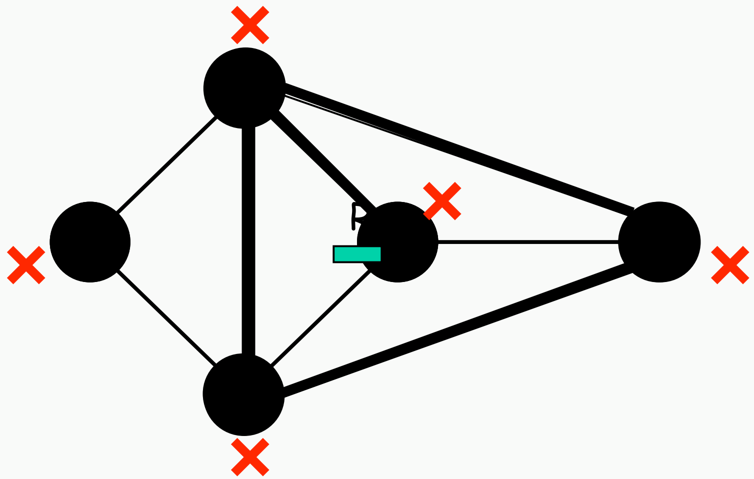









R



One version

1) When first visited, remember who sent,
forward the token to one of the unvisited neighbours
wait for its reply

2) When neighbour receives,
if already visited, it will return the token saying it is a back edge
otherwise, will forward it (sequentially) to all its unvisited neighbour before returning it

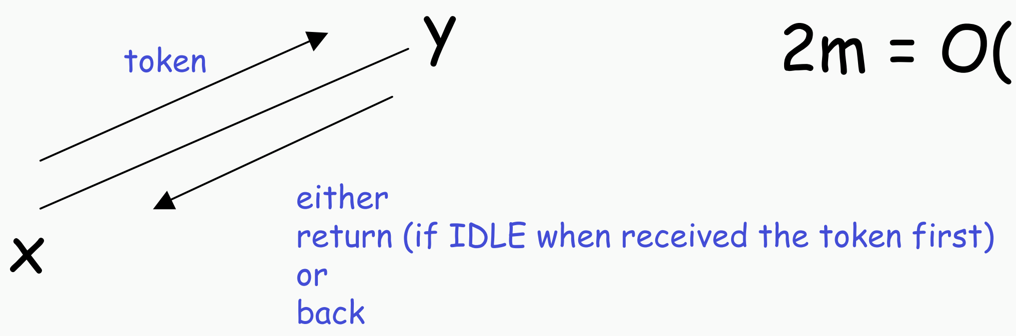
3) If there are no more unvisited neighbours, return the token (reply) to the node from which it first received the token

4) Upon reception of reply, forward the token to another unvisited neighbour

Complexity

Message Complexity:

Type of messages: `token, back, return`



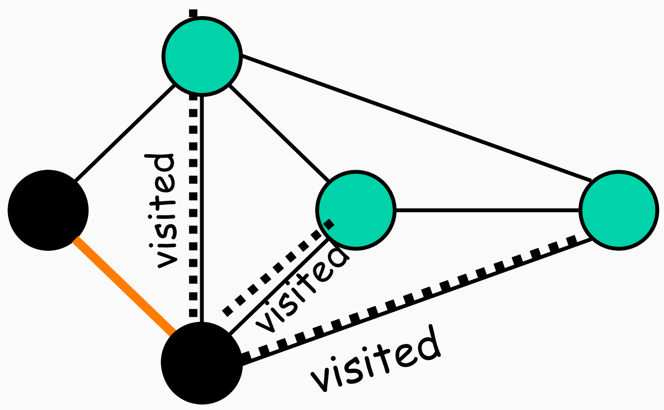
$2m = O(m)$

Time Complexity:
(ideal time)

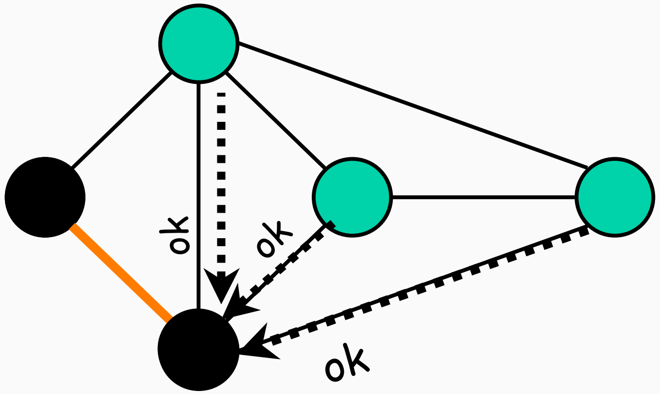
$2m = O(m)$

$\Omega(m)$ is also a lower bound

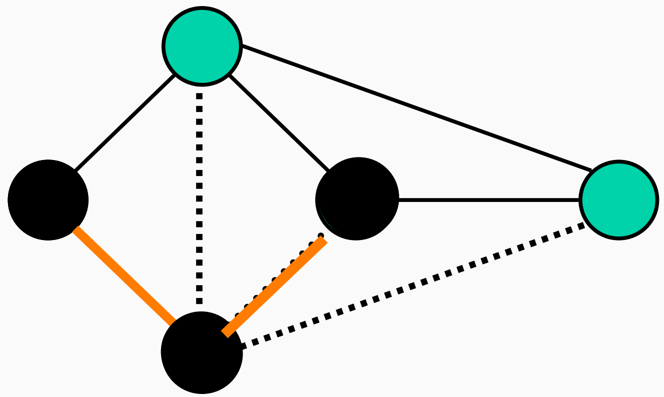
Improving Time



Improving Time



Improving Time



Complexity Message

Messages: **Token**, **Return**, **Visited**, **Ack (ok)**

Each entity (except init): receives 1 **Token**, sends 1 **Return**:

2(n-1)

Each entity(except initiator):

1 **Visited** to all neighbours except 1

Let s be the initiator

$$|N(s)| + \sum_{x \neq s} (|N(x)|-1)$$

$$= 2m - (n-1)$$

(same for **Ack**)

TOT: 4m

Complexity   Time (ideal time)

Token and Return are sent sequentially: $2(n-1)$

Visited and Ack are done in parallel: $2n$

TOT: $4n - 2$

Summarizing:

DF Traversal

	Messages	Ideal Time
VERSION 1:	2m	2m
VERSION 2:	4m	4n -2

Observations about Traversals

Termination ...

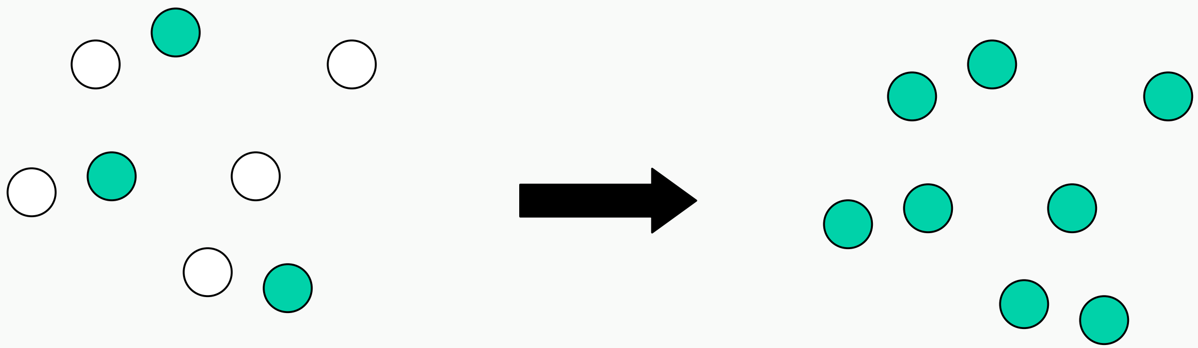
Application:

Access permission problems: Mutual Exclusion

Any Traversal does a Broadcast (not very efficient)
The reverse is not true.

Computations with Multiple initiator: WAKE-UP

Definition



FLOOD solves the problem.

General FLOOD algorithm: $O(m)$

More precisely: $2m - n + k^*$

WHY ?

n. of initiators

1 init = broadcast = $2m - n + 1$

All init = $2m$

Computations with Multiple initiator: WAKE-UP

In special topologies ?

TREE

Flood is optimal $\qquad n + k^* - 2$

COMPLETE GRAPH

$\Omega(n^2)$

HYPERCUBE

$\Omega(n \log n)$