The Model & Basic Computations

Chapter 1 and 2

The Model

Broadcast

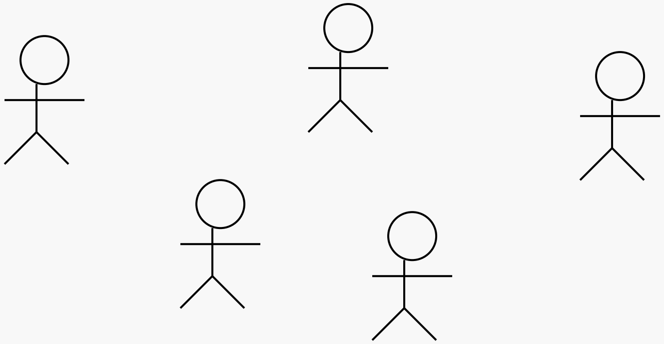
Spanning Tree Construction

Traversal

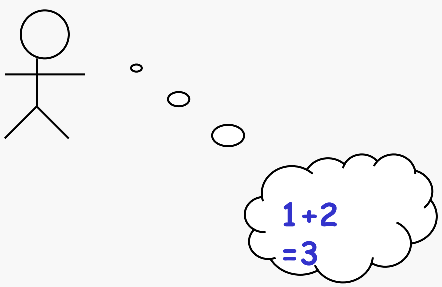
Wake-up

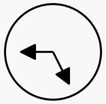
Distributed Environment

Multiplicity

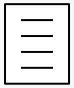


Autonomy





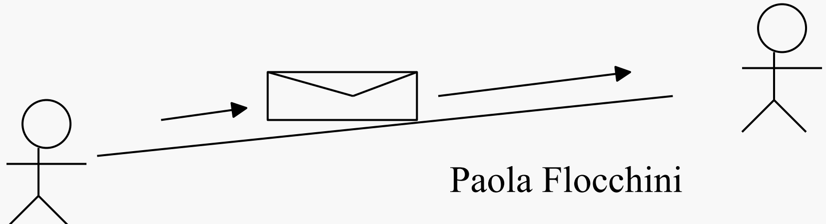
clock



memory

computing capabilities

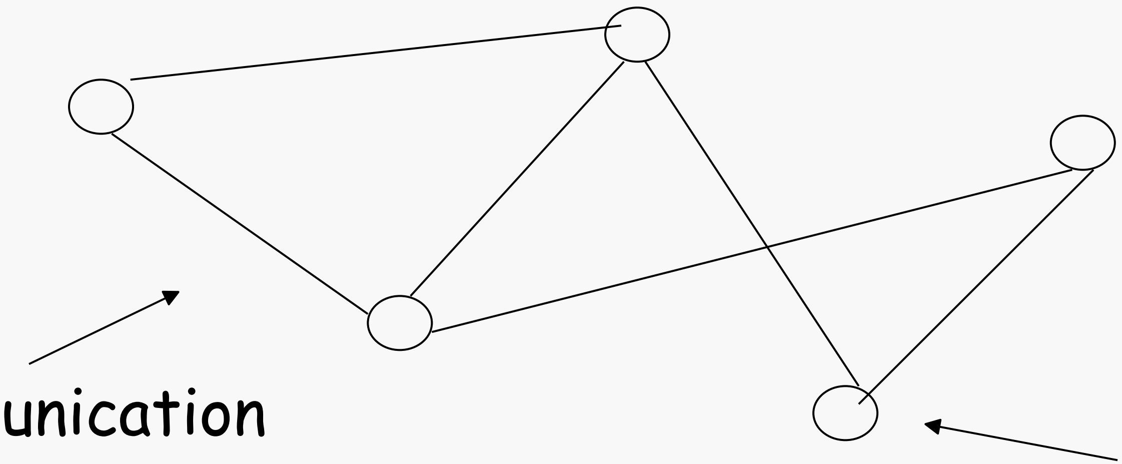
Interaction



typically by
exchange of messages

The Model

Collection of entities that communicate by exchanging messages

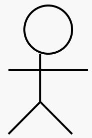


communication link

entity, node, site ...

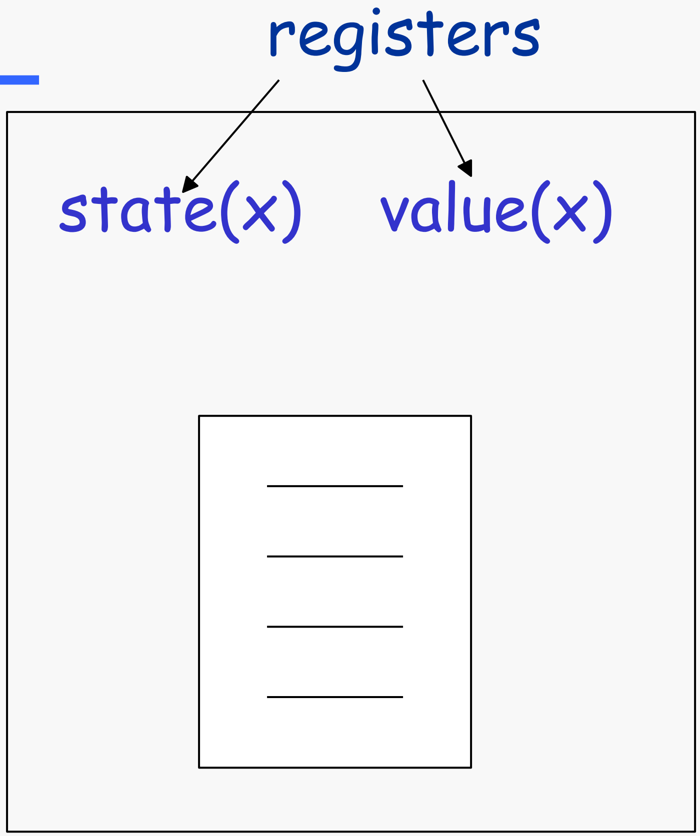
(processor, process, object ....)

message-passing



Entity x

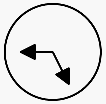
In memory:



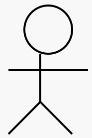
Possible operations:

- local storage and processing
- transmission of messages

- (re)setting the clock



- changing the value of the registers



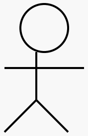
Entity x

state(x)

Finite set of possible system states
(ex: {idle, computing, waiting, processing ….})

Always defined

At any time an entity is in one of these states



External Events

The behavior of an entity is reactive:
triggered by events

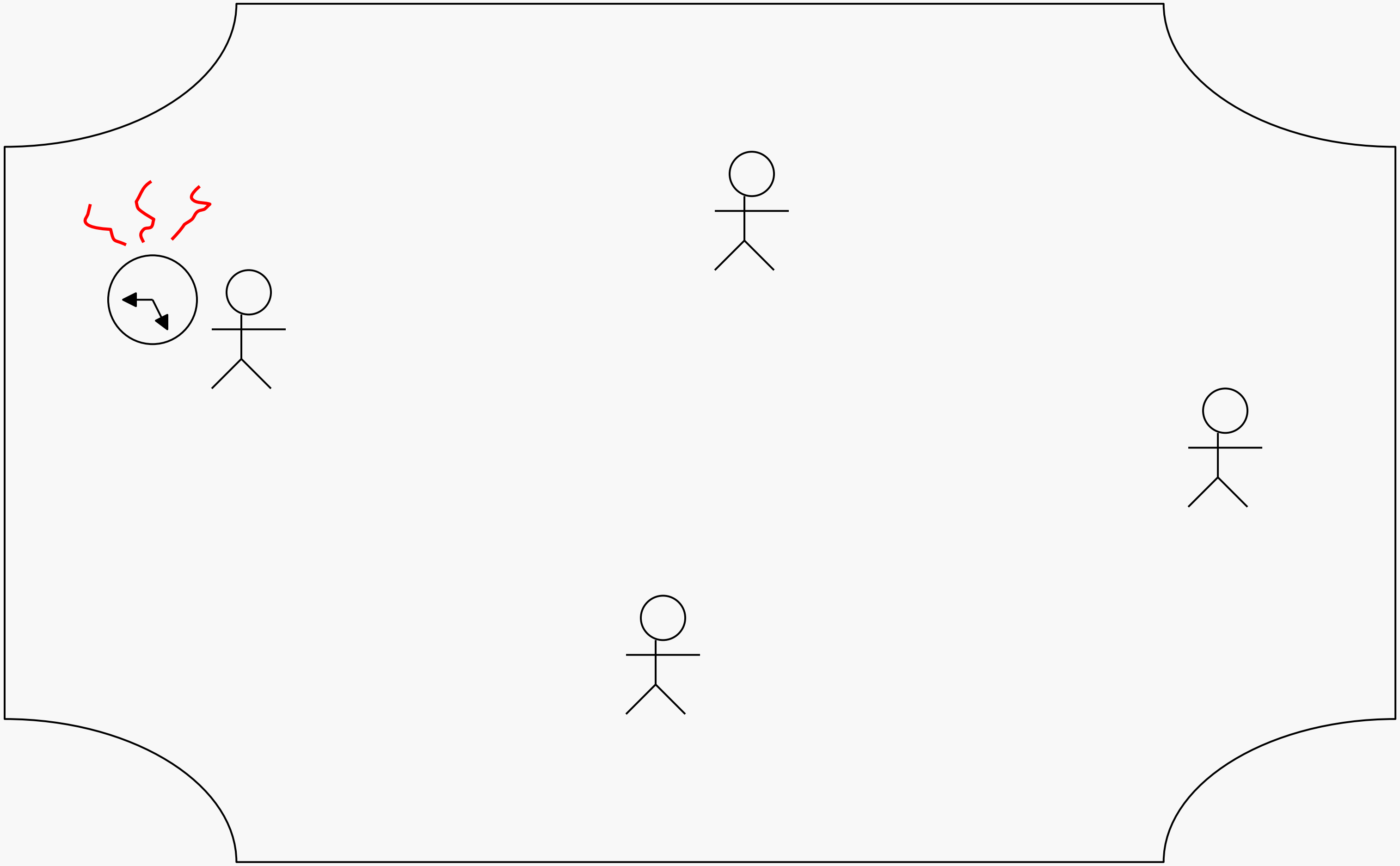
Possible events:

- clock tick
- receiving a message
- spontaneous impulse

Originated within the system

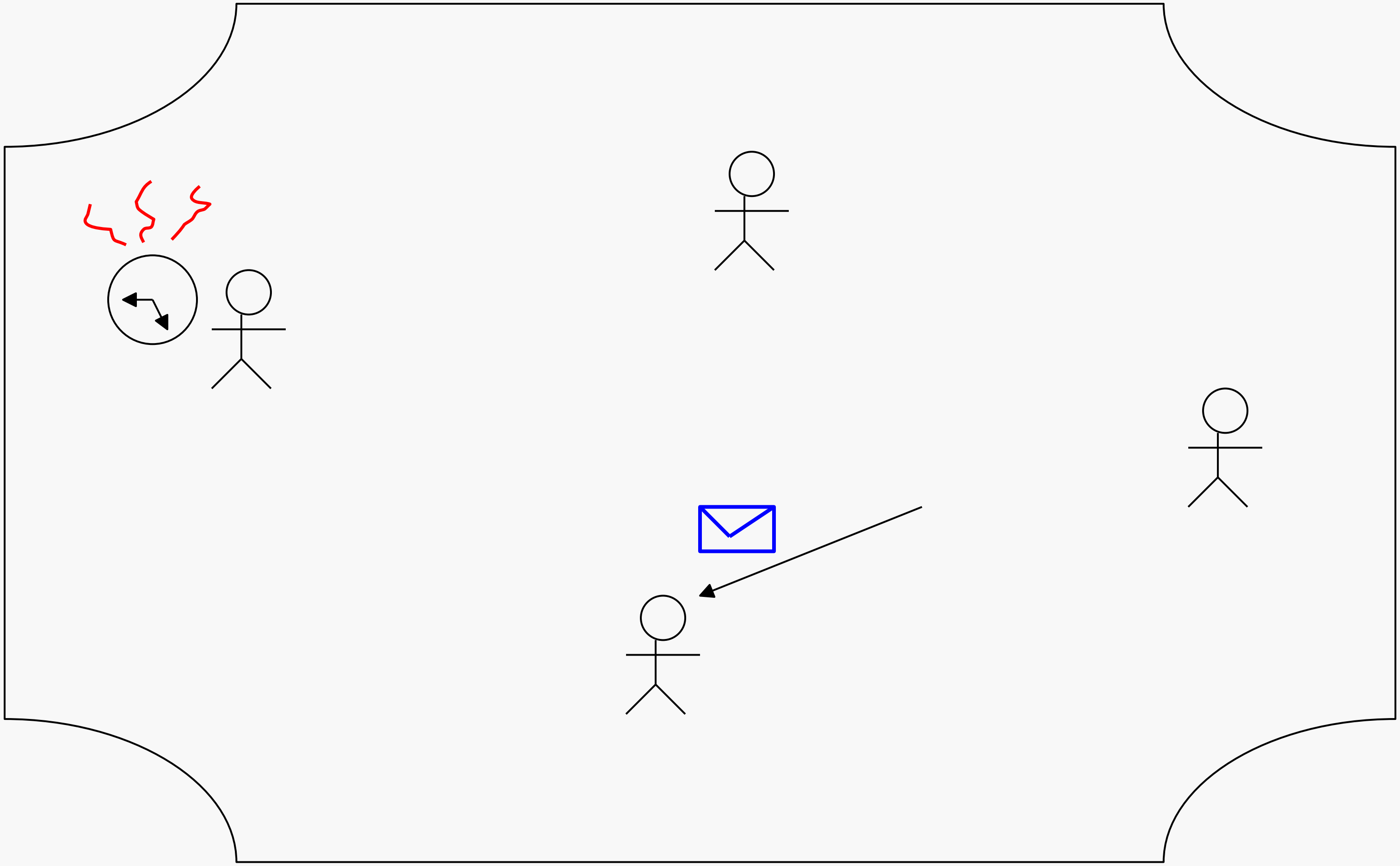
Triggered by something outside the system

clock ring

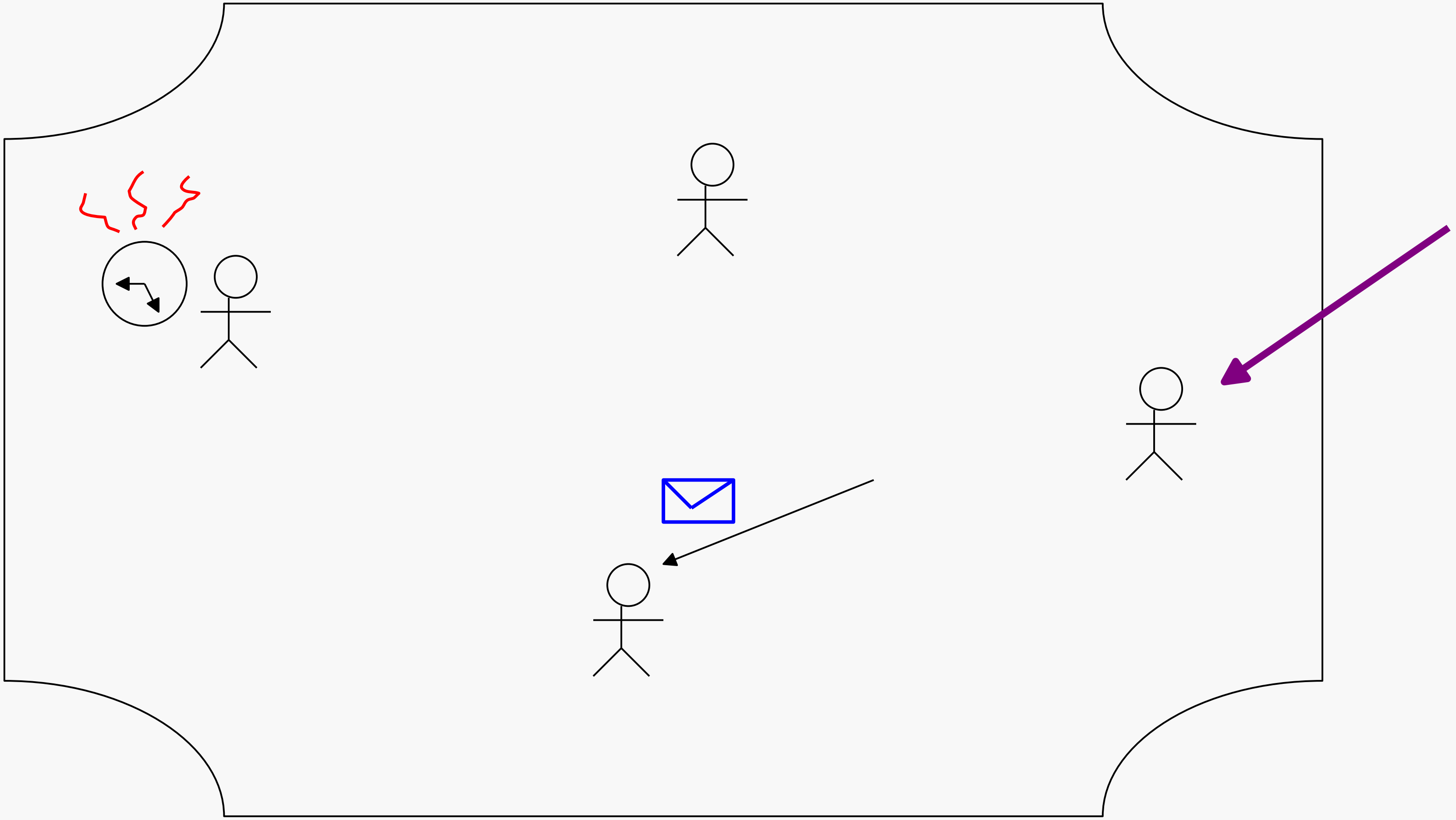


Paola Flocchini

message arrival



spontaneous impulse



Actions

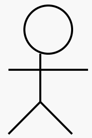
Action: sequence of activities, e.g.,

- computing
- sending message
- change state

an action is atomic
the activities cannot be interrupted

an action is terminating
the activities must terminate within finite time

the special action nil
the entity does not react to the event



Entity Behavior

Rule

State x Event ⟶ Action

Behavior B(x) = set of rules of entity x for all possible events and all possible states

The algorithm
The protocol

DETERMINISTIC
(state, event) --> only ONE action

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

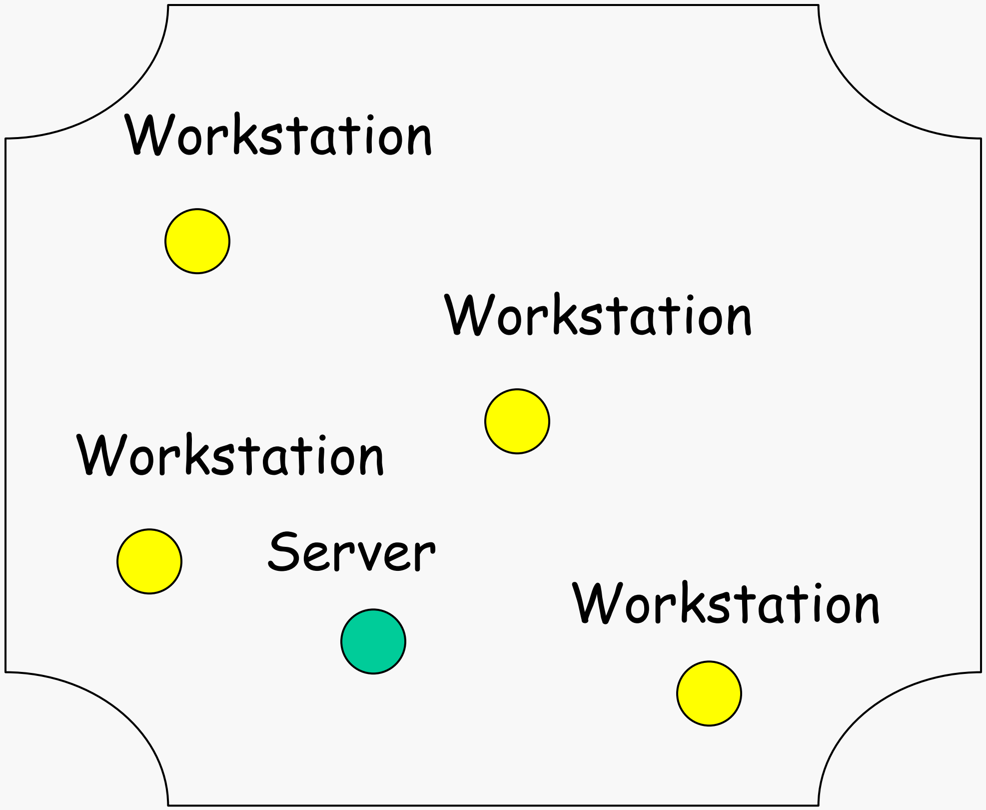
System Behavior

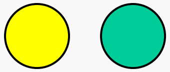
$$B = \{ B(x) : x \text{ in } E\}$$

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

$$B(x) = B(y), \text{ for all } x,y \text{ in } E$$

Property: Every system can be made symmetric





Very different rules, states ....

role =(workstation/server)

$s \times e$ →

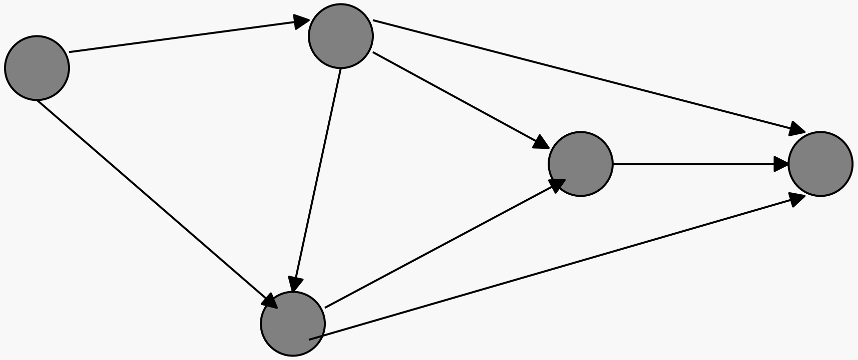
```
If role = workstation
      ActionW(s,e)
else
      ActionS(s,e)
```

Communication

Message: the unit of communication

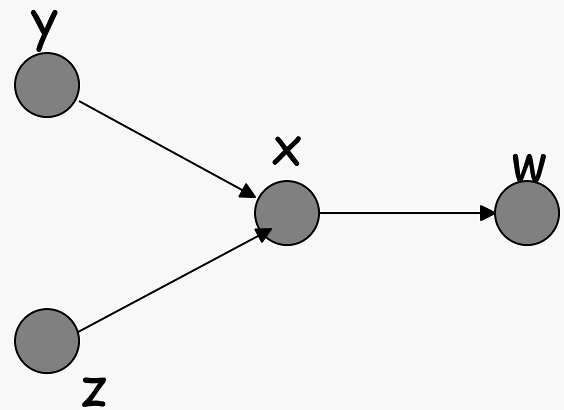
finite sequence of bits

Communication Network:



Communication

Point-to-point Model



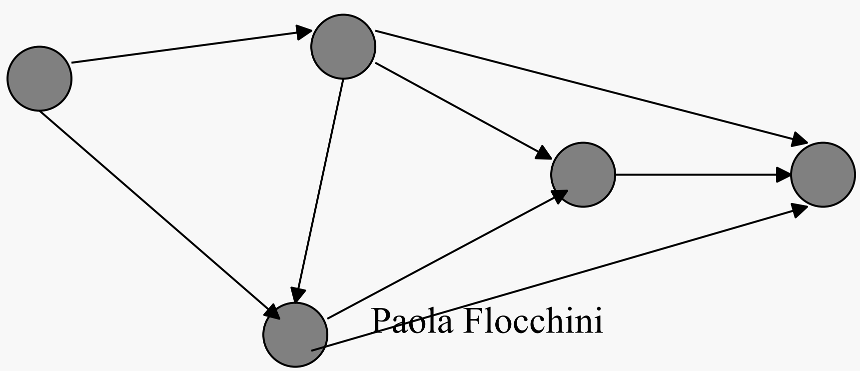
$N_o(x)$ = out-neighbors of entity x
$N_i(x)$ = in-neighbors of entity x

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

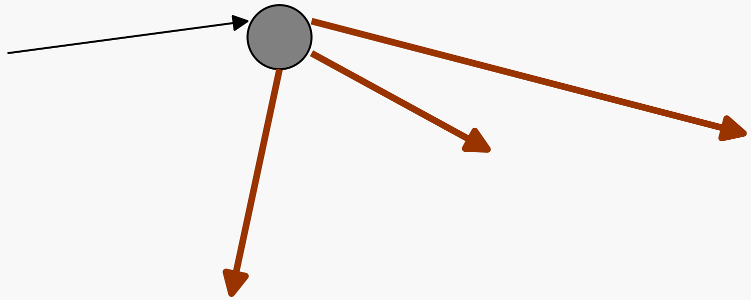
Graph describing the
COMMUNICATION TOPOLOGY

$G = (V, A)$

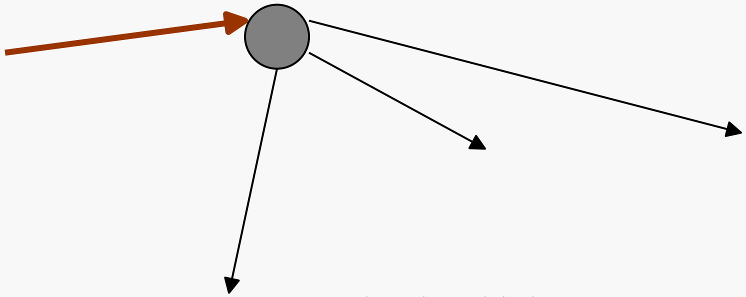
V: Entities
A: Arcs defined by N



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

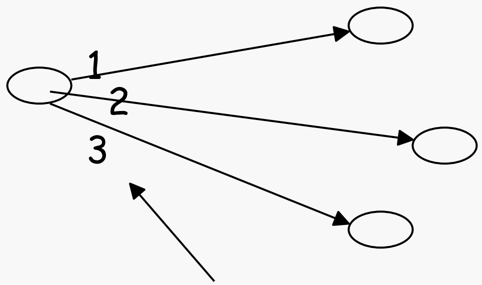
In absence of faults a message from x to its out-neighbour y reaches y in finite time

Local orientation

Each entity distinguishes among its neighbors

Local orientation: more precisely

Each entity distinguishes among its out-neighbors

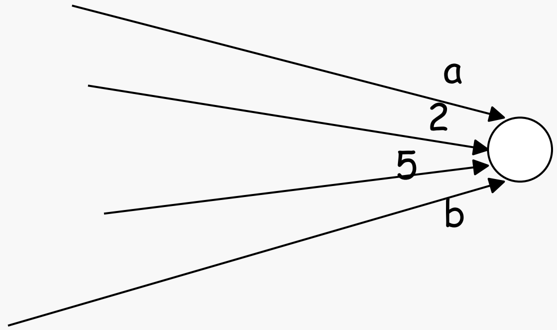


distinct labels
=
(out) port numbers

Send Message **to** 3

Local orientation: more precisely

Each entity distinguishes among its in-neighbors

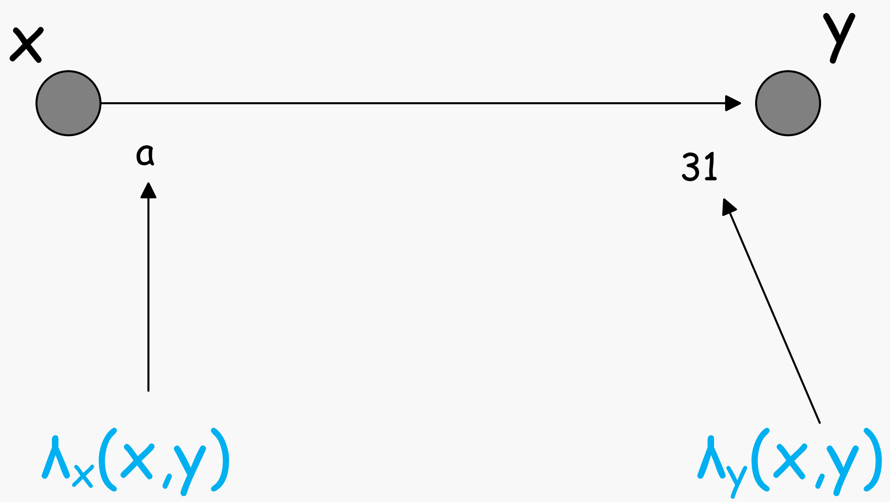


distinct labels
=
(in) port numbers

When a message arrives, the entity can detect from which port

Local orientation: more precisely

for an edge (x,y) there are two labels:

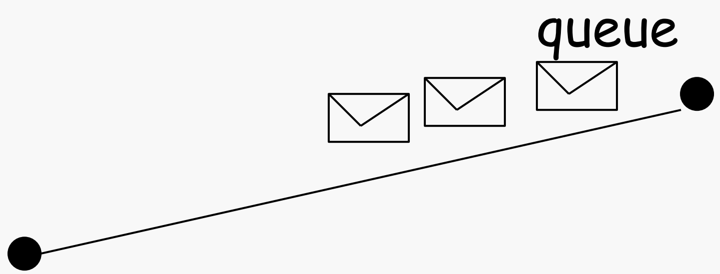


local labeling function

Topology = labeled graph (G, λ) $\lambda = \{ \lambda_x \text{ in } V \}$

Restrictions of the model: examples

Communication Restrictions



Message Ordering

(FIFO)

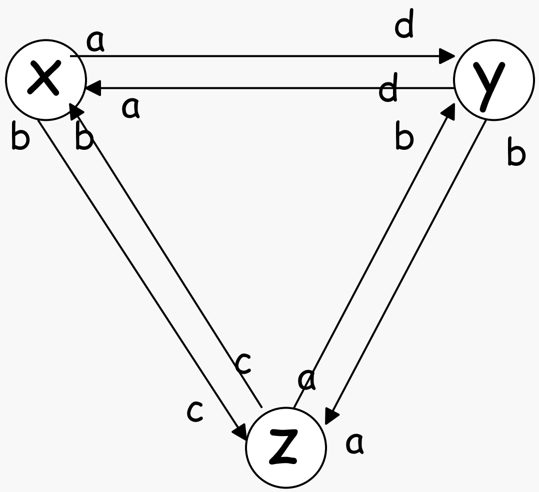
In absence of failures, msgs transmitted along the same link arrive in the same order.

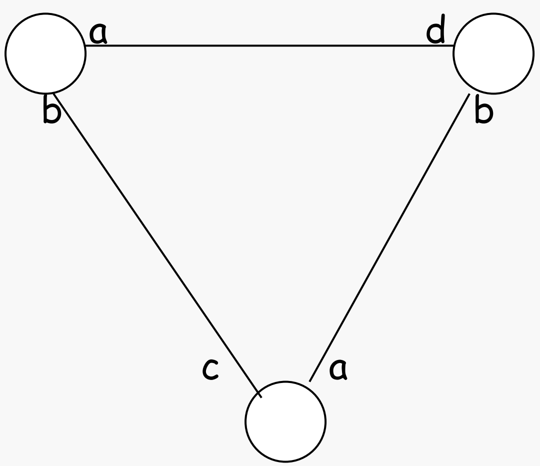
Restrictions of the model: examples

Communication Restrictions

Bidirectional Links

for all x, $N_i(x) = N_o(x) = N(x)$ and
for all y in N(x): $\lambda_x(x,y) = \lambda_x(y,x)$





Restrictions of the model: examples

Detection of Faults:

1. *Edge Failure Detection:*
   An entity can detect if a link failed or was recovered

2. *Entity Failure Detection:*
   An entity can detect if a neighbor failed

Restrictions of the model: examples

Reliability Restrictions:

1. *Guaranteed delivery:*
   Any message that is sent will be received uncorrupted

2. *Partial Reliability:*
   There will be no failures

3. Total Reliability:
   No failures have occurred nor will occur

Restrictions of the model: examples

Topological restriction:

The graph G is strongly connected

Knowledge Restrictions

Knowledge of number of nodes
Knowledge of number of links
Knowledge of diameter

Restrictions of the model: examples

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

Synchronized clocks:

All local clocks are incremented by one unit simultaneously and interval are constant

Complexity measures - Performance

1. Amount of communication activities

M number of messages exchanged
(finer granularity: number of bits)

point of view
of SYSTEM

Entity workload: **M**/|V|
Link workload: **M**/|E|

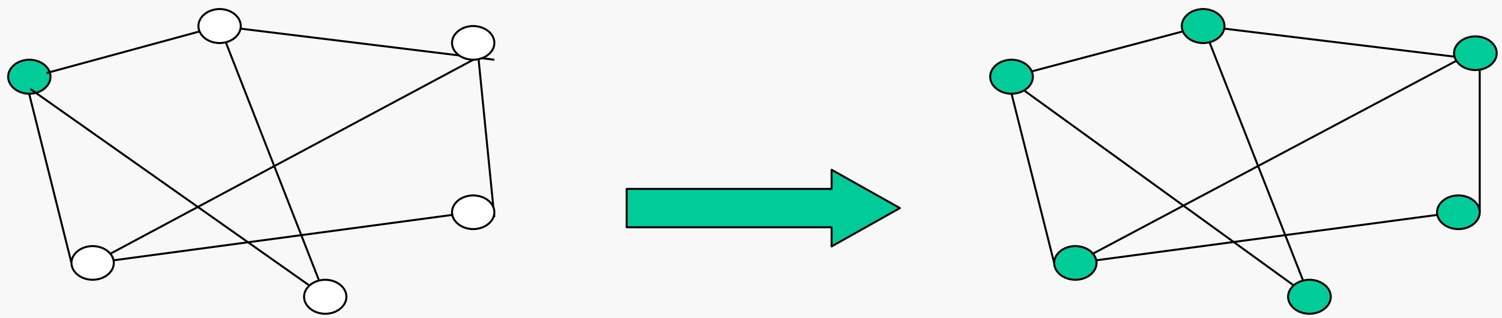
2. Time

point of view
of USER

Communication delays are in general unpredictable !!!

Ideal time:
1 unit of time to transmit 1 message

Example - Broadcast



Assumptions = Restrictions

Unique Initiator → By definition of problem

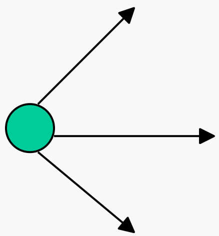
Total reliability
Bidirectional links → Simplifying assumptions
G is connected

→ Otherwise unsolvable

Algorithm FLOOD

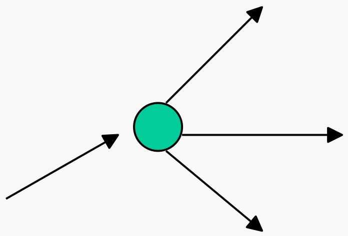
<u>The idea:</u> If an entity knows something, it sends the info to its neighbours

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}
```

It does not terminate

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 FLOOD

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 FLOOD

Algorithm for node x:

State

```
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
```

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
```

Event

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
```

Action

Correctness

The Algorithm terminates in finite time

It follows from: G connected and total reliability

Termination

Local Termination: when DONE

Global Termination: when?

Message complexity

Worst Case

m = number of links

Worst for all possible initiators
and for all possible executions

Messages: 2 on each link

$$\sum_x |N(x)| = 2m$$

$$2m = O(m)$$

More precisely:

Let s be the initiator

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

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

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

Time Complexity - Ideal Time Worst Case

Worst for all possible initiators
and for all possible executions

Time: (ideal time) Unitary Transmission Delay & Synchronized Clocks

Distance between x and s

$r(s) = Max_x\{d(x,s)\}$ = eccentricity of s O(n)

$\leq$ Diameter(G) $\leq$ n-1

$Max_x\{r(x)\}$

Time and Events

External events:

spontaneously
receiving
when (clock)

Actions may generate events

send	generates	***receiving***
set-clock	generates	***when***

Generated events might not occur (in case of faults).
If they occur, they occur later.

In the case of *receiving* with some unpredictable delay.

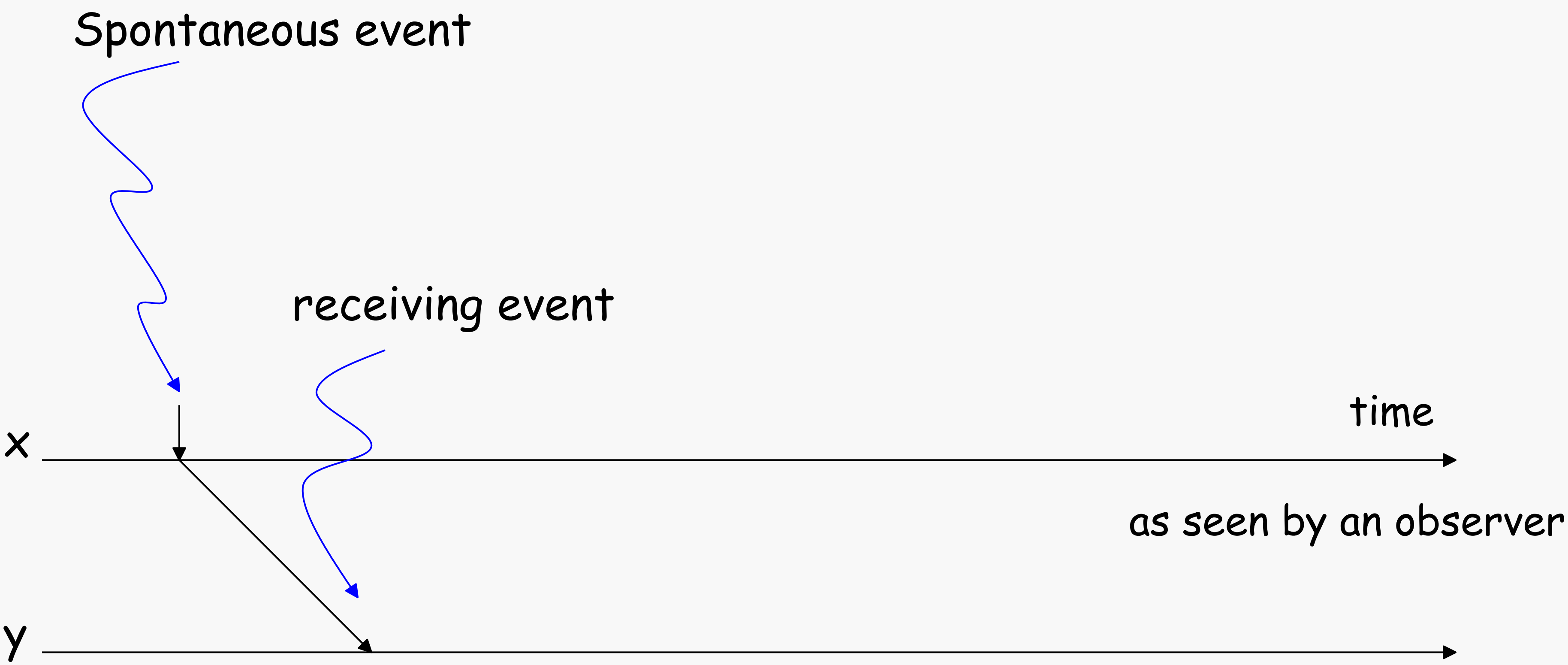
Different delays ---> different executions

Different executions could have different outcomes

(Spontaneous events are considered generated *before* execution starts: initial events)

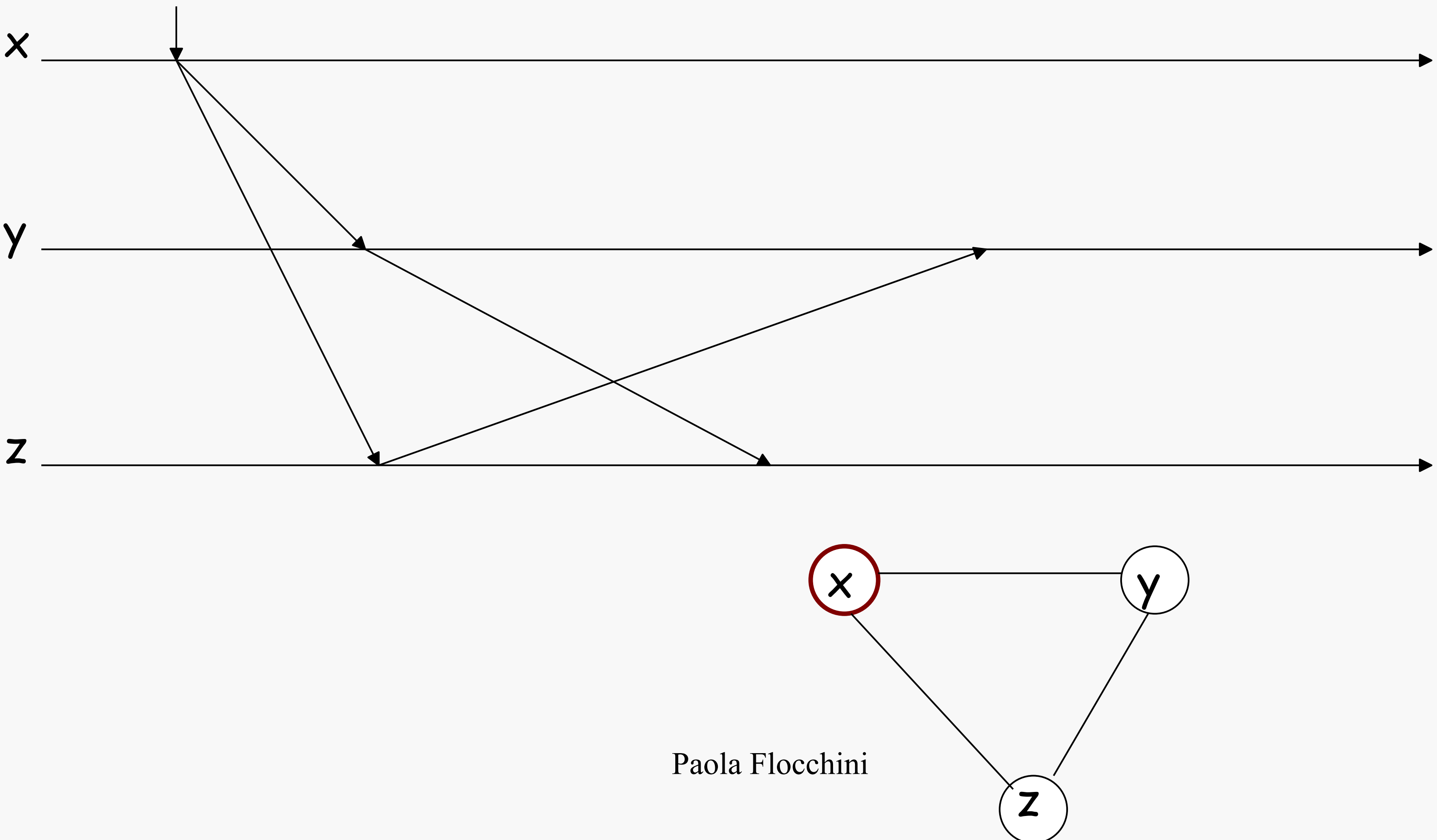
An executions is fully described by the sequence of events that have occurred

Time x Event diagram



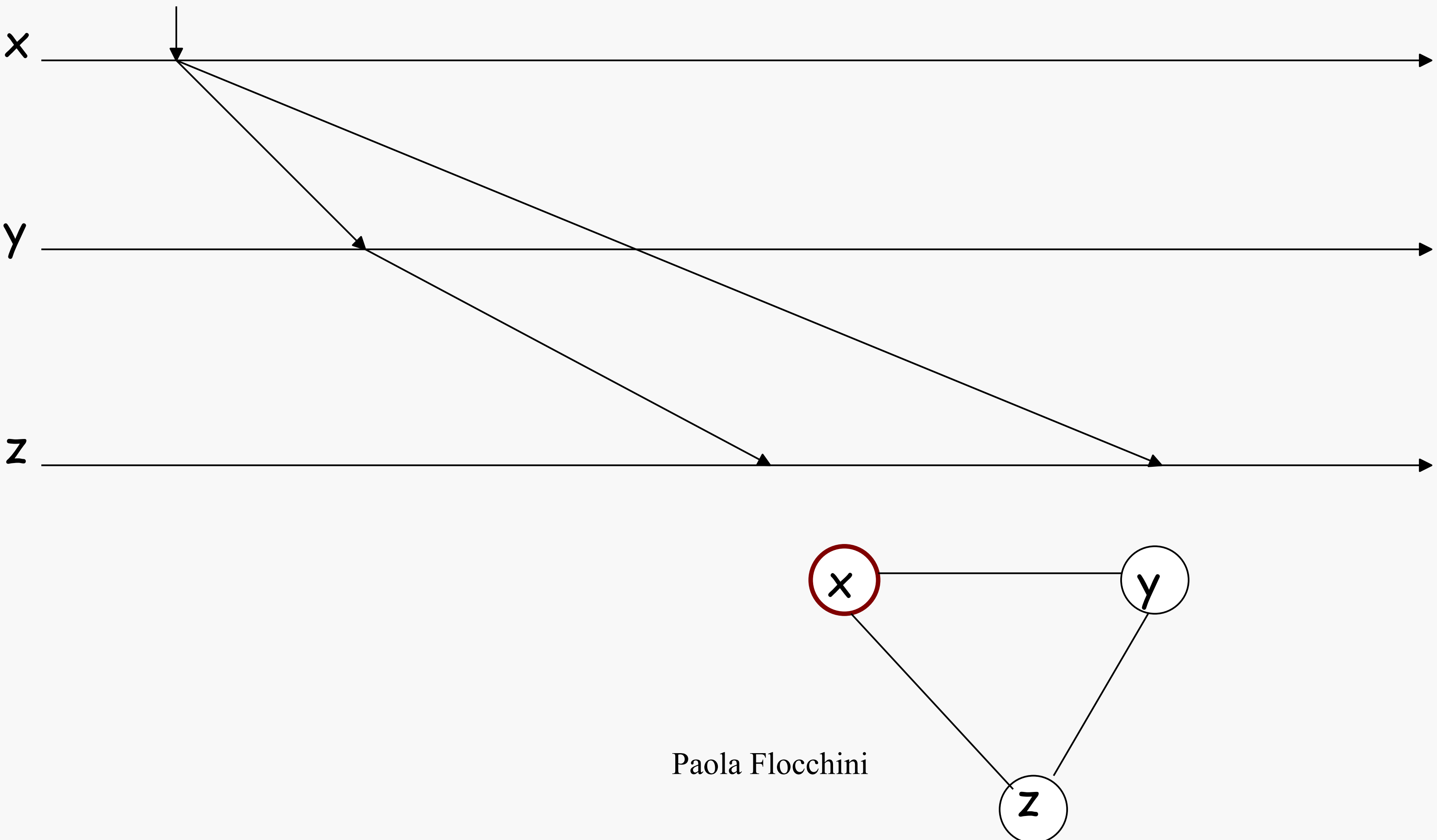
Example: Time x Event diagram of Flooding

One possible execution



Example: Time x Event diagram of Flooding

Another execution



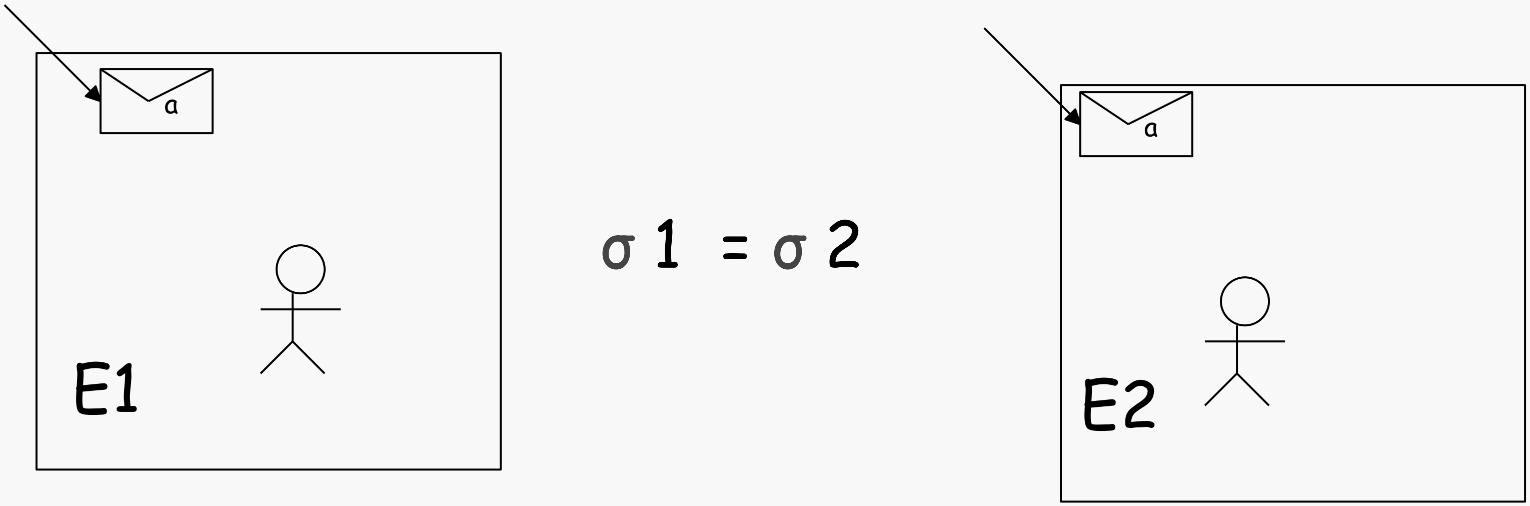
Important Facts

State x Event ---> Action

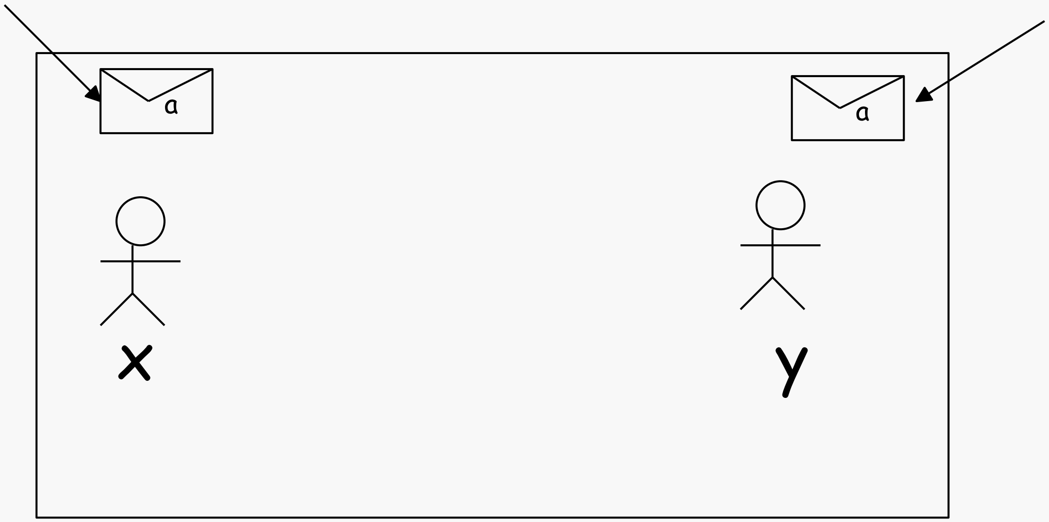
$\sigma(x,t)$ = internal state of entity x

content of memory (registers, clock, …) at time t

E1, E2: different environments



If the same event happens to x at time t in two different executions and if the internal states σ 1 and σ 2 of x in the two executions at that time are equal, then
the new internal state of x will be the same in both executions



$\sigma(x) = \sigma(y)$

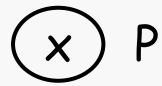
2) If the same event happens to x and y at time t in the same execution and if the internal states $\sigma(x)$ and $\sigma(y)$ are equal, then **the new internal states of x and y will be the same.**

Knowledge

P = fact; x = entity; S = set of entities.

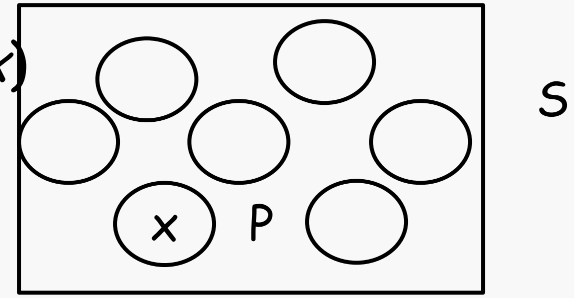
- **Local knowledge LK**

  P in $LK_t(x)$.



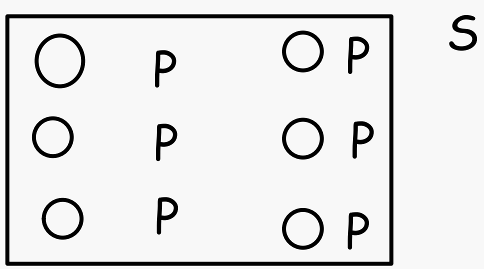
- **Implicit knowledge IK**

  P in $IK_t(S)$ if *exists x* in S: P in $LK_t(x)$



- **Explicit knowledge EK**

  P in $EK_t(S)$ if *for all x* in S: P in $LK_t(x)$

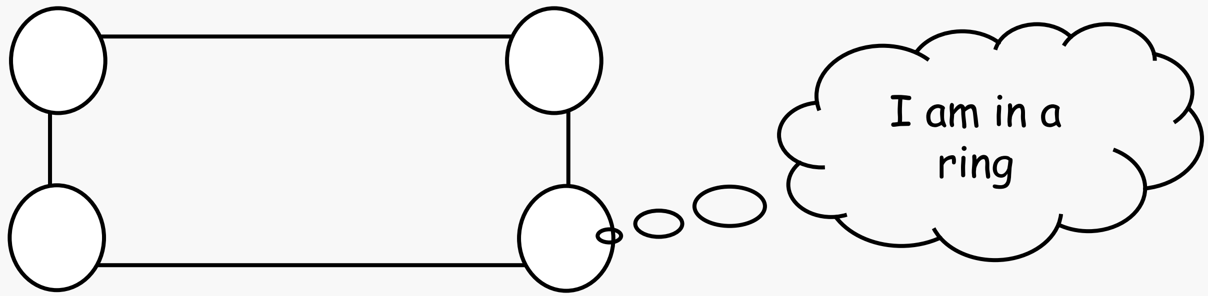


- **Common knowledge CK**

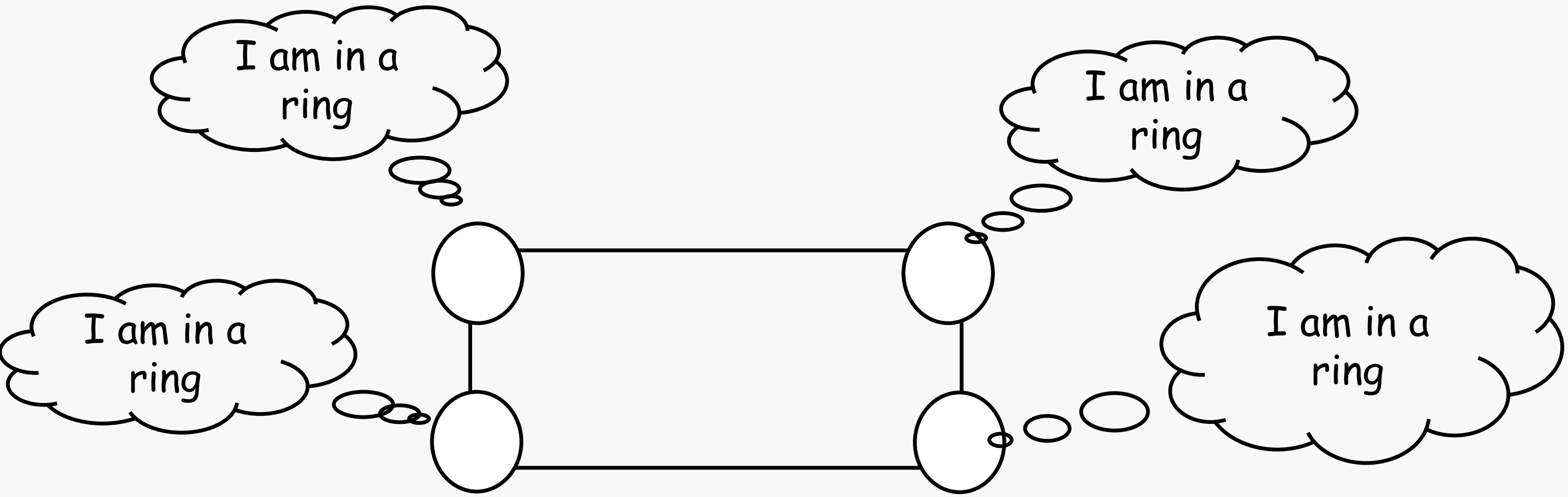
P in $CK_t(S)$ if
for all x in S, P in $LK_t(x)$ □
for all x in S (□x in S, P in $LK_t(x)$) /\ $LK_t(x)$ /\
for all x in S ((□x in S, P in LK_t (x)) /\ LK_t (x)) /\ LK_t (x) /\ ...

Examples

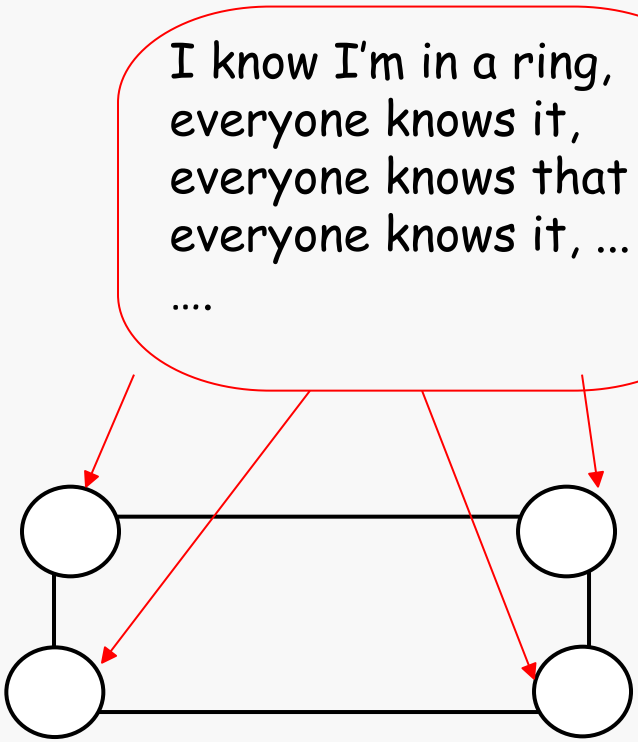
Implicit knowledge



Explicit knowledge



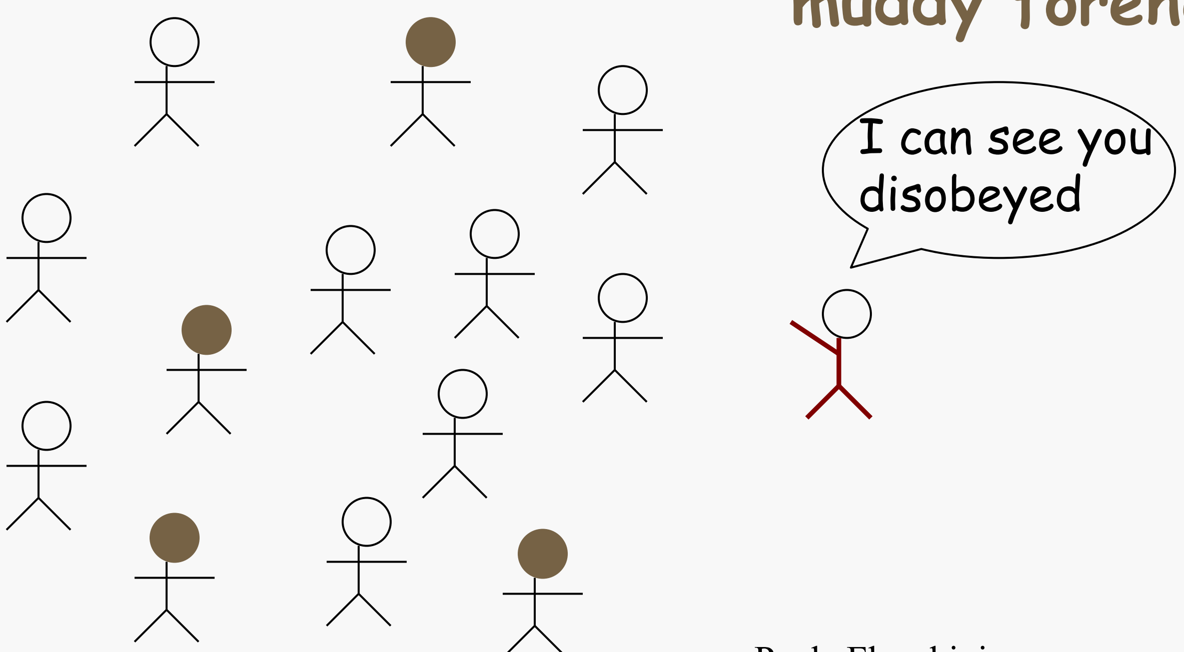
Common knowledge



How to reach common knowledge in FINITE TIME ?

P in *CK*(*S*) if
for all x in *S*, *P* in *LK*(*x*) ▯
for all x in *S* (▯*x* in *S*, *P* in *LK*(*x*)) /\ *LK*(*x*) /\
for all x in *S* ((▯*x* in *S*, *P* in *LK*(*x*)) /\ *LK*(*x*)) /\ *LK*(*x*) /\ ...

muddy forehead

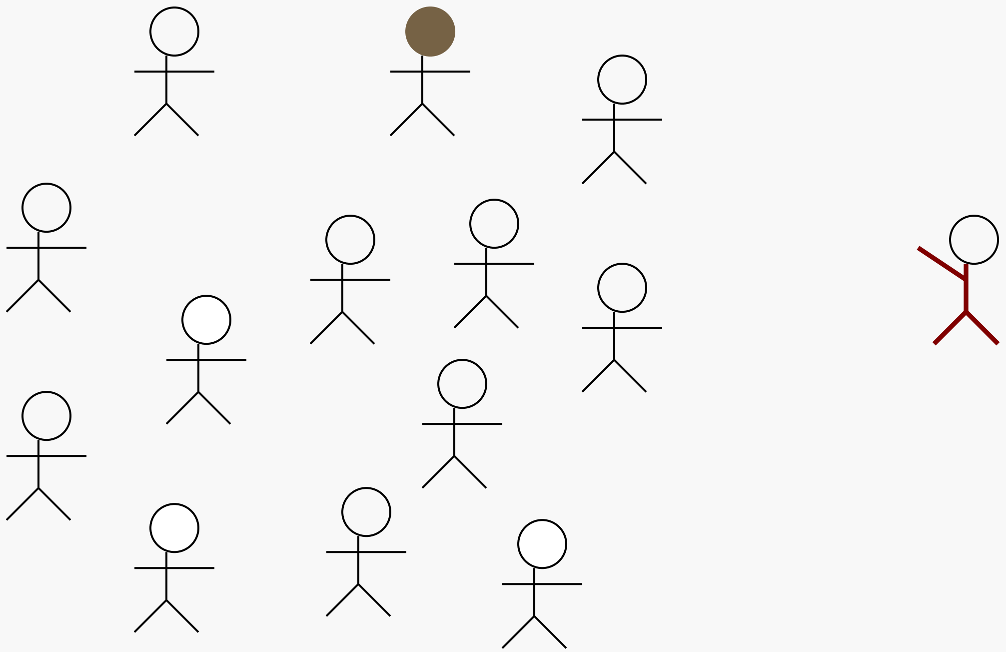


t=1

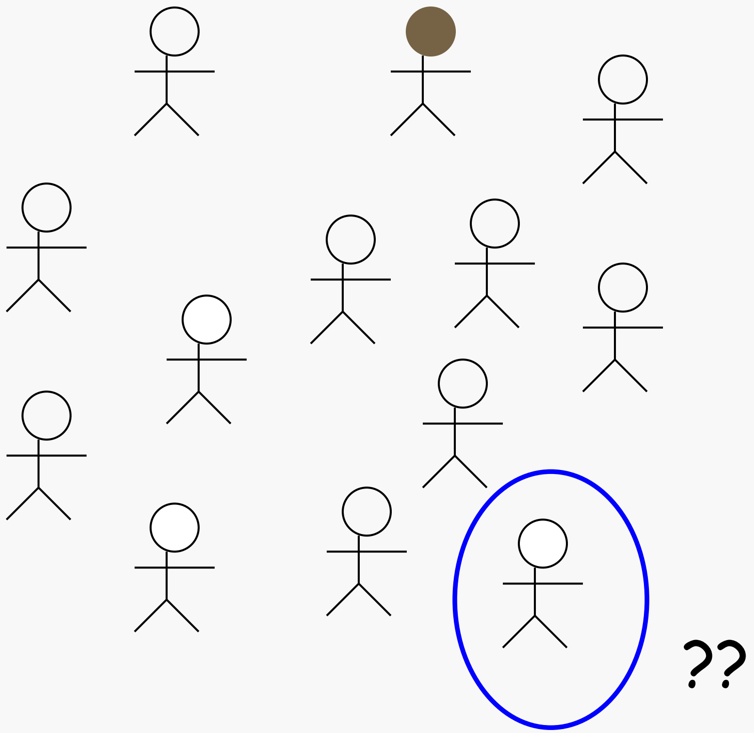
t=2

t=3

t=4



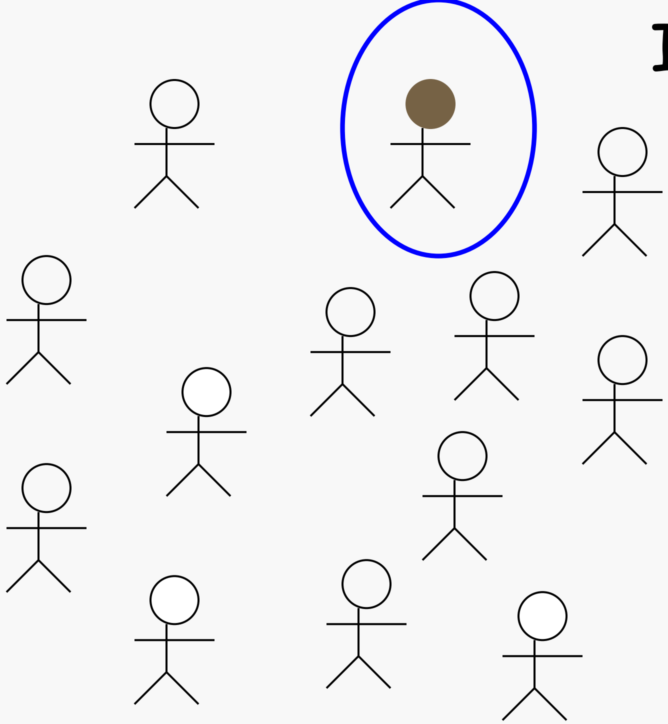
Paola Flocchini

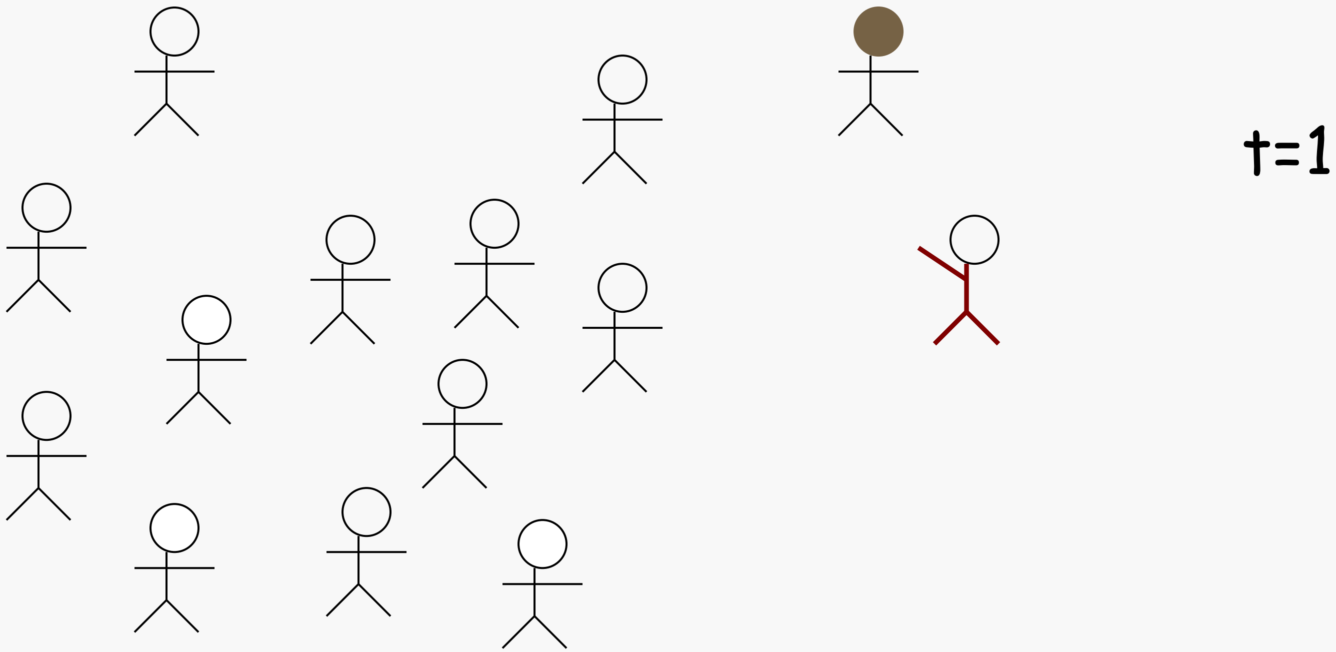


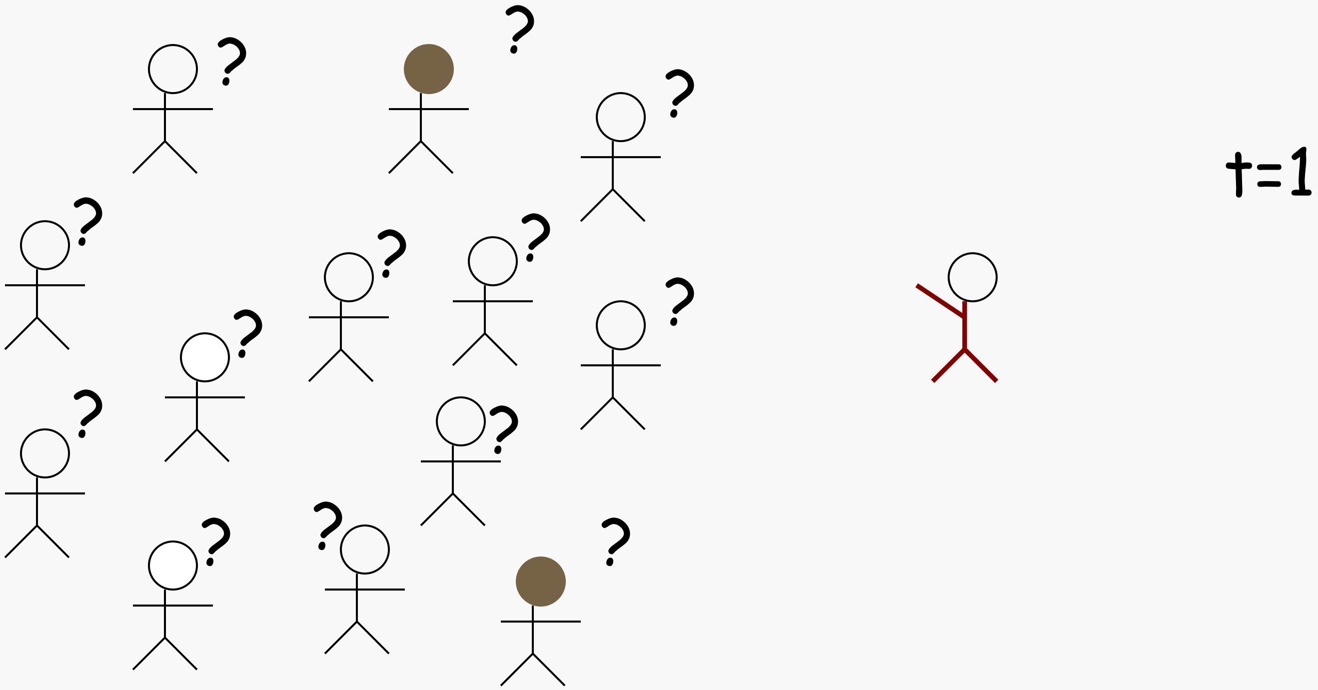
I see one dirty forehead

I see everybody else is clean

I must be dirty !!!!!

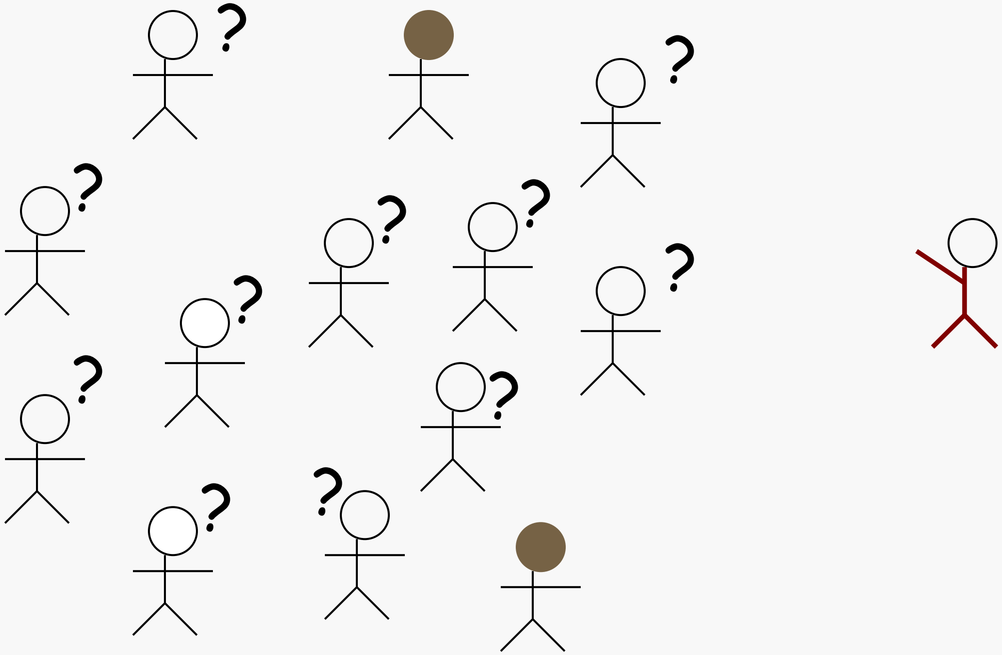



t=1

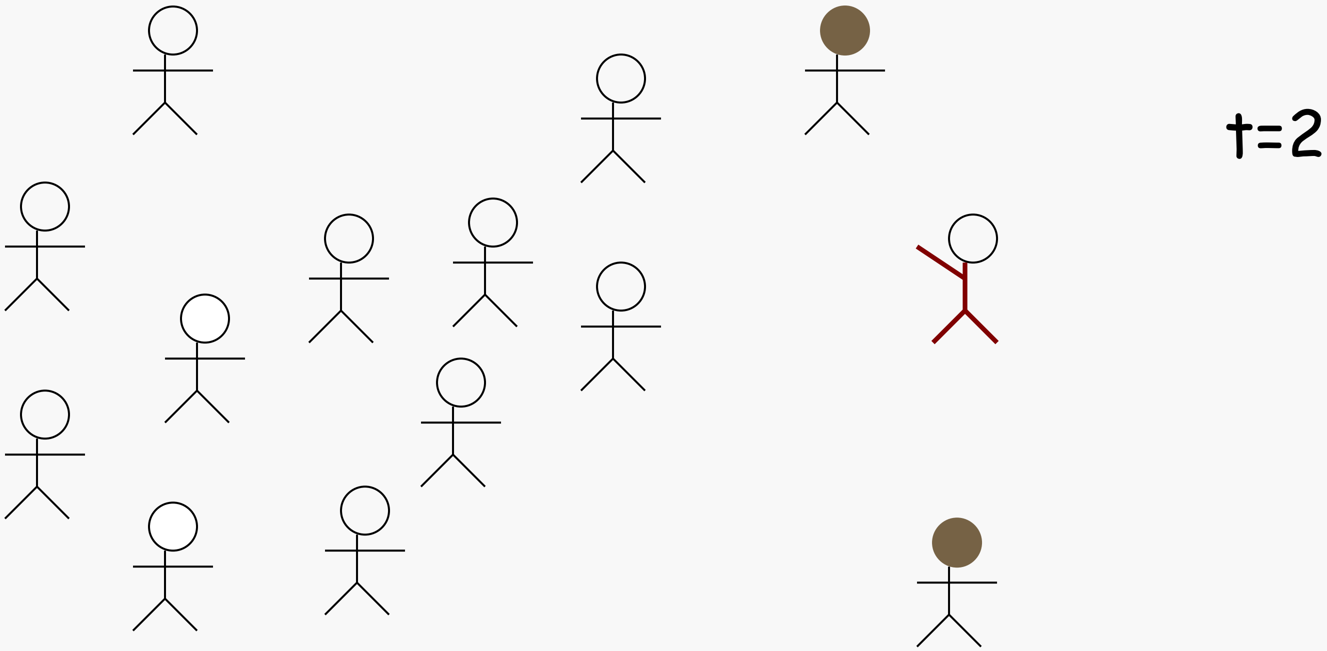

t=1

The other didn't go forward … I must be dirty too !!!!

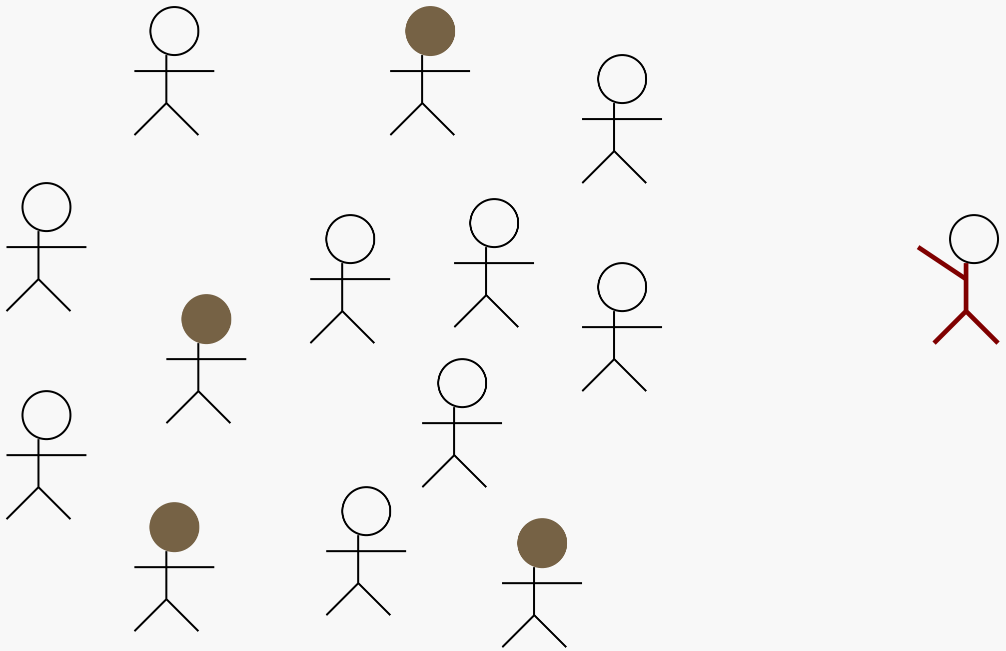
t=1



The other didn't go forward …
I must be dirty too !!!!


t=2

In general ....



If I see k dirty foreheads and they do not go forward all together at time k, I go forward at time k+1

Some types of knowledge

Topological knowledge
Graph type ("G is a ring"...), adjacency matrix of G ...

Metric knowledge
Number of nodes, diameter, eccentricity...

Sense of direction
Information on link labels
Information on node labels

As the available knowledge grows, the algorithm becomes less portable (rigid). Generic algorithms do not use any knowledge.

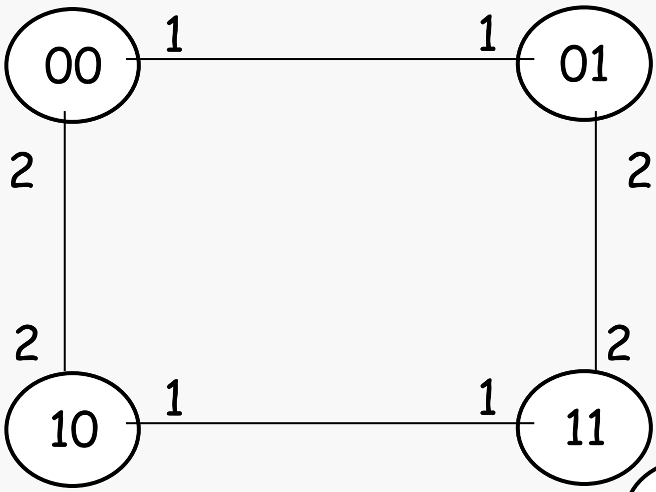
Example: impact of knowledge

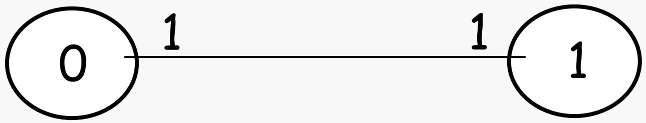
In specific topologies flooding can be avoided and broadcast can be much more efficient (if the topology is known).

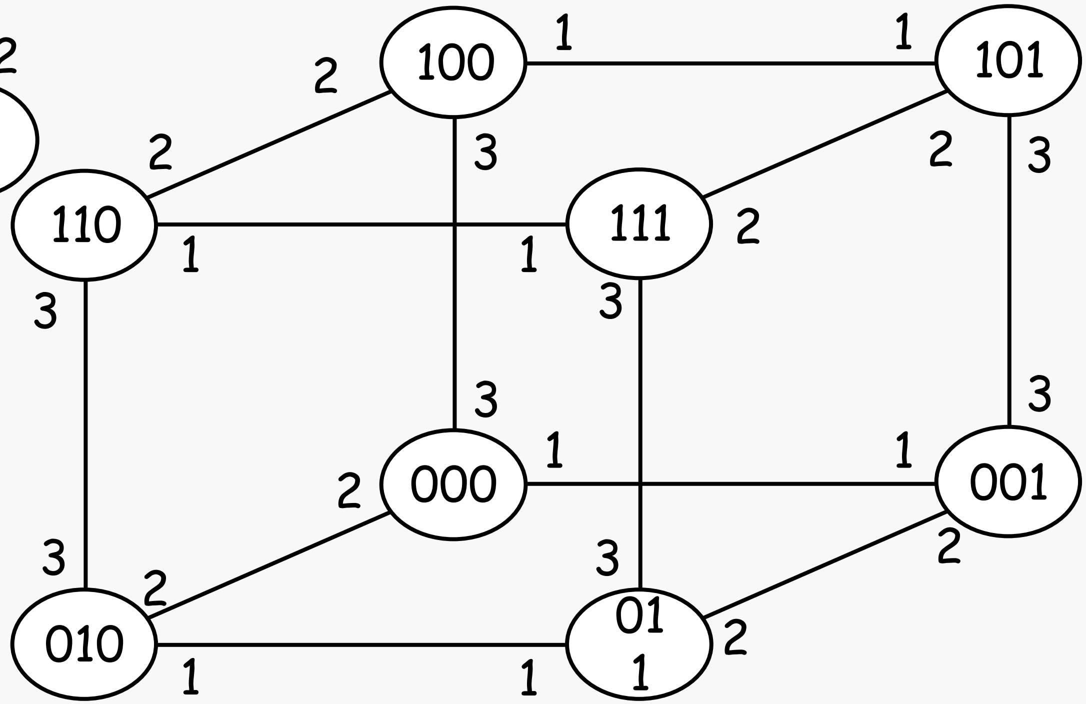
What is the complexity of flooding in a complete graph ?
How can it be done more efficiently ?

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

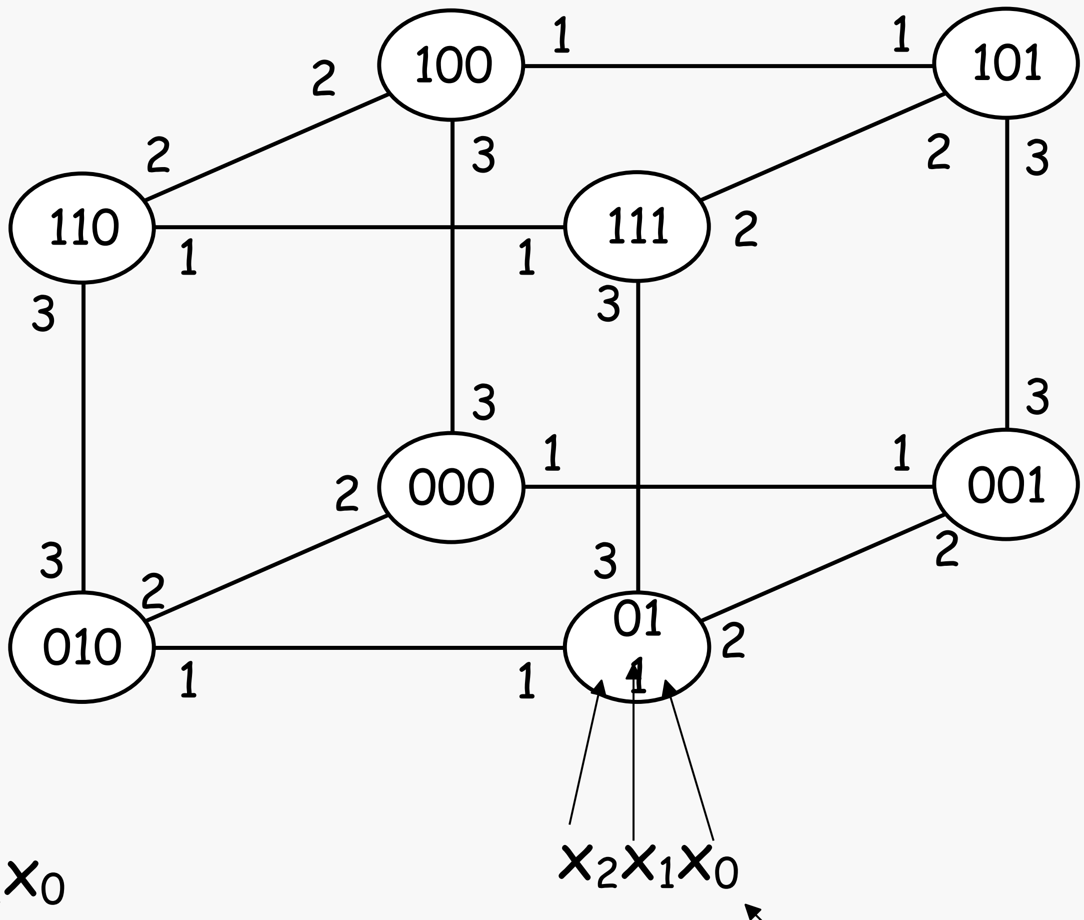
Example: The labeled hypercube





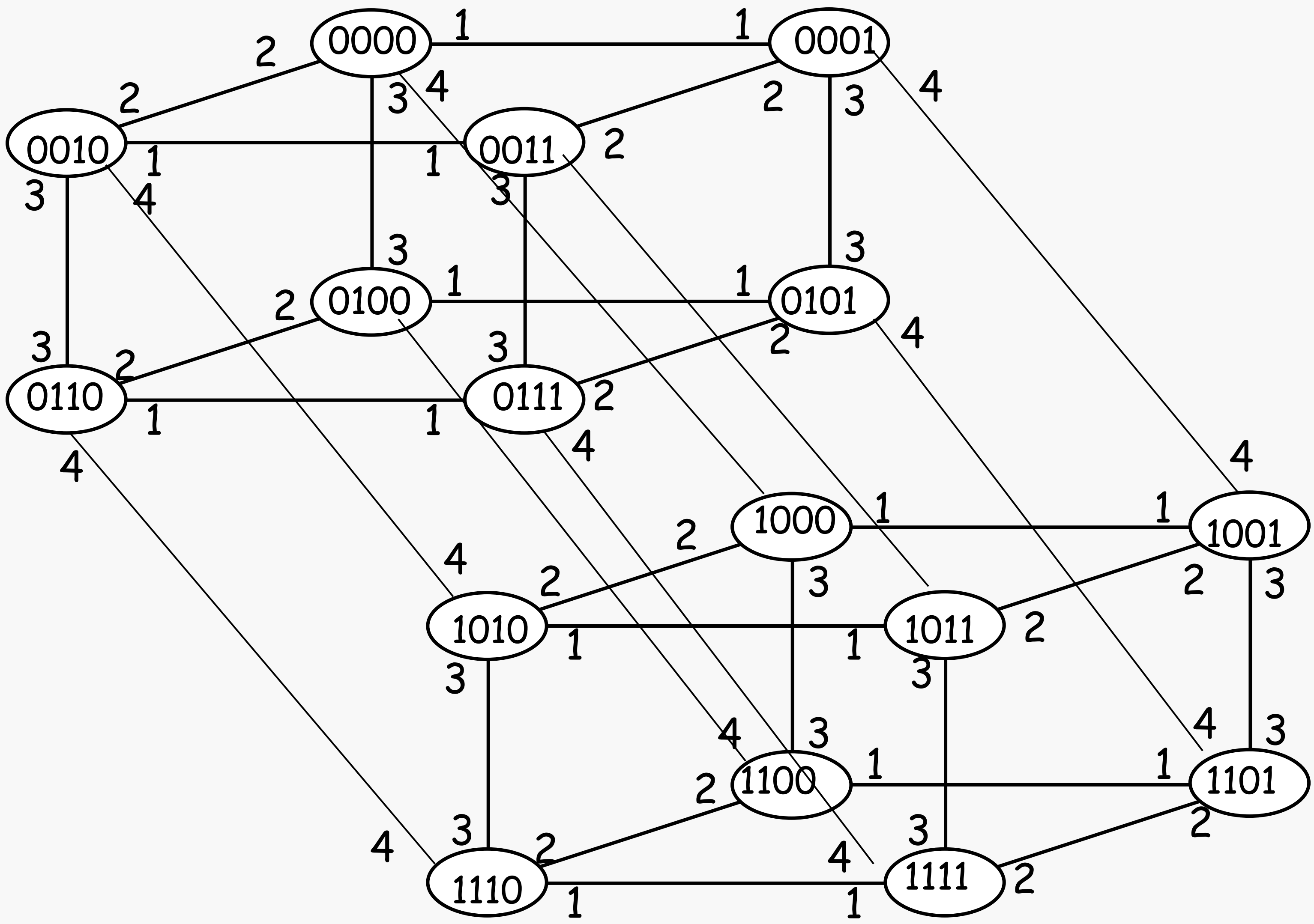


Each link between two nodes is labeled by the dimension of the bit by which the nodes' name differ.



$X = x_k x_{k-1} \ldots\ldots x_1 x_0$

K-bit name


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

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

Each node has k links

$m = n\,k/2$ (total number of links) $O(n \log n)$

Flooding would cost $O(n \log n)$

$$
\begin{aligned}
2\,m - (n - 1) &= n \log n - (n - 1) \\
&= n \log n /2 + 1 \\
&= O(n \log n)
\end{aligned}
$$

HyperFlood - Efficient Broadcast

1. The initiator sends the message to all its neighbors
2. A node receiving the message from link l, sends it only to links with label l’ < l

Correctness

Every node is touched

Based on the lemma:

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

$X = x_k, x_{k-1} \ldots\ldots x_1, x_0$

$Y = y_k, y_{k-1} \ldots\ldots y_1, y_0$

Correctness

Every node is touched

Based on the lemma:

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

$X = x_k, x_{k-1} \ldots\ldots x_1, x_0$

$y = y_k, y_{k-1} \ldots\ldots y_1, y_0$

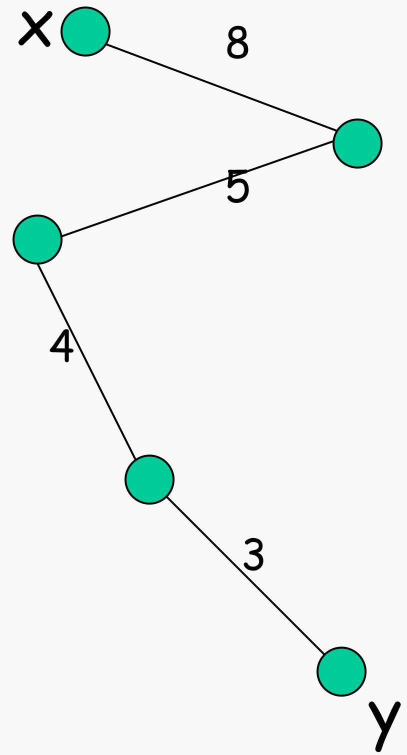
Consider positions where they differ in decreasing order …

Example:

8 5 4 3

X= 100010100

Y= 110001000



x

100010100

110010100

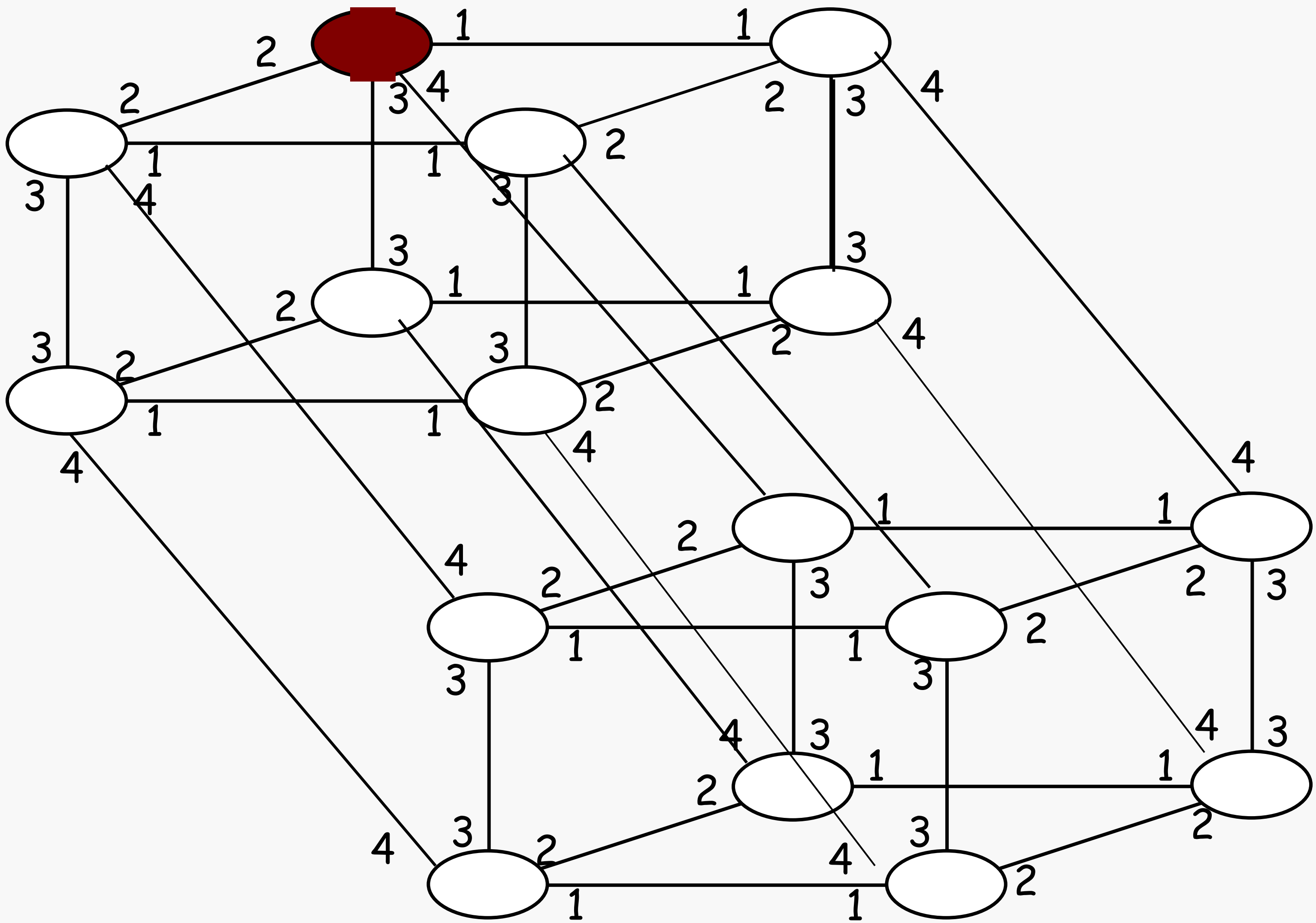
100000100

100001100

100001000 y

---> the messages create a spanning tree ....

and every node is touched



Complexity: n-1 (OPTIMAL)

Because every entity receives the info only ONCE.

Ideal time complexity: k

Because every entity's eccentricity is k.

Complete Graphs

General Flooding: $2m - (n-1) = O(n^2)$

Observation

Since everybody is connected to everybody, the initiator needs to send the information only ONCE.

Message Complexity: $(n-1)$

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)

Lower Bounds for Broadcast

Back to General Broadcast

Theorem: Under the set of assumptions:

unique Initiator
G is connected
no failures
bidirectional links

Theorem
Every generic broadcast protocol requires, in the worst case, $\Omega(m)$ messages.

Lower Bounds for Broadcast

Proof.

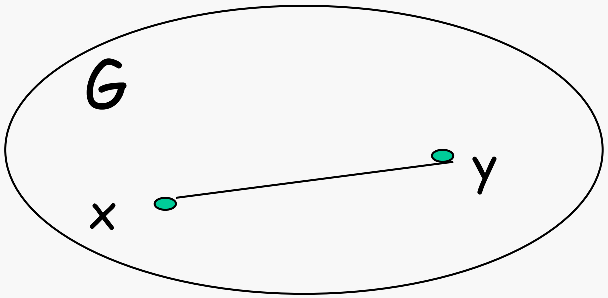
m(G) = n. of edges in G

By contradiction.
Let *A* be an algorithm that broadcasts exchanging less than *m(G)* messages (in all executions, and for any graph G) under those assumption.

Then there is at least a link in G where no messages are sent.

Let e = (x,y) be such a link.

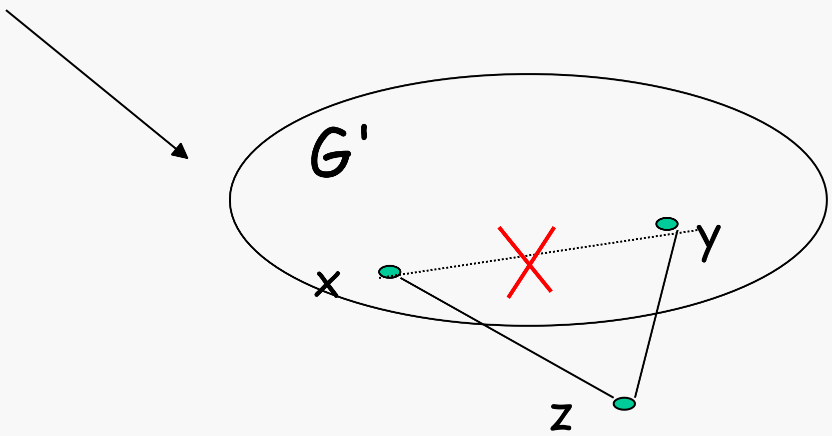
G=(V,E)



Construct a new graph G′

(remember: n is unknown)

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



Execute the same algorithm on G′ with the same time delays, same initial internal states for all nodes except for z which is `sleeping`

Two executions in two environments

For all nodes, except z, the two executions are **identical**

x and y never send to each other in E1

--->

x and y never send to z in E2

Within finite time the protocol terminates

but in E2 node z will never be reached.

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