The Model & Basic Computations

Chapter 1 and 2

The Model

Broadcast

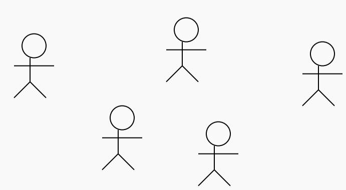
Spanning Tree Construction

Traversal

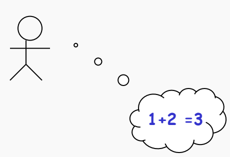
Wake-up

Distributed Environment

Multiplicity



Autonomy

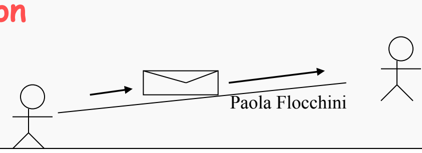


clock

memory

computing capabilities

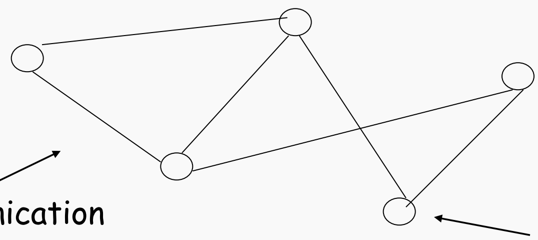
Interaction



typically by
exchange of messages

The Model

Collection of entities that communicate
by exchanging messages

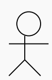


communication
link

entity, node, site ...

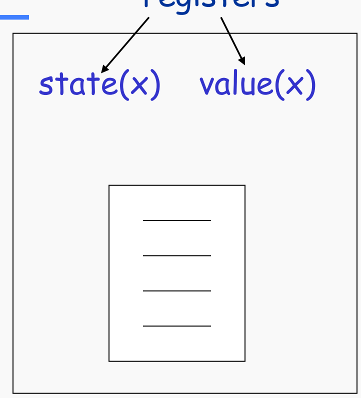
(processor, process,
object ….)

message-passing

 Entity x

In memory:

registers



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 ….})

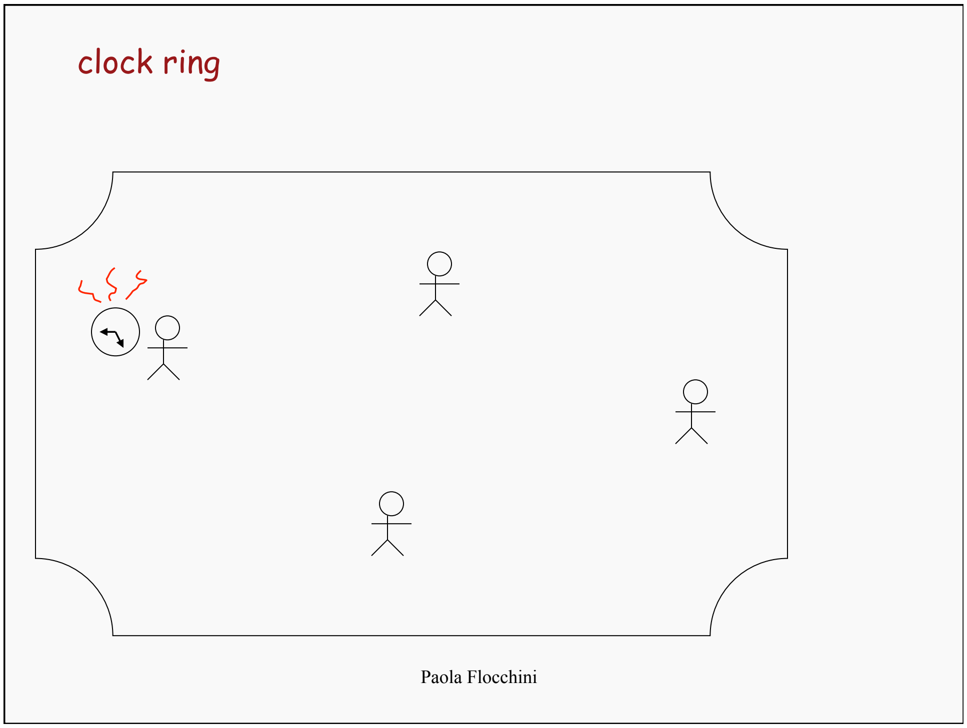
Always defined

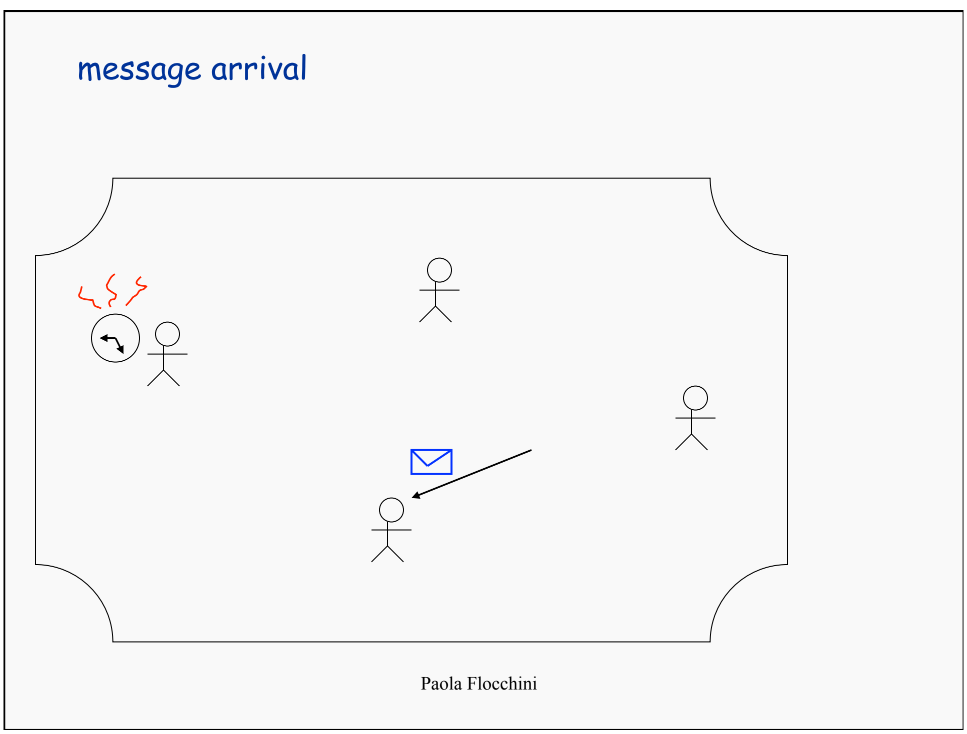
At any time an entity is in one of these states

External Events

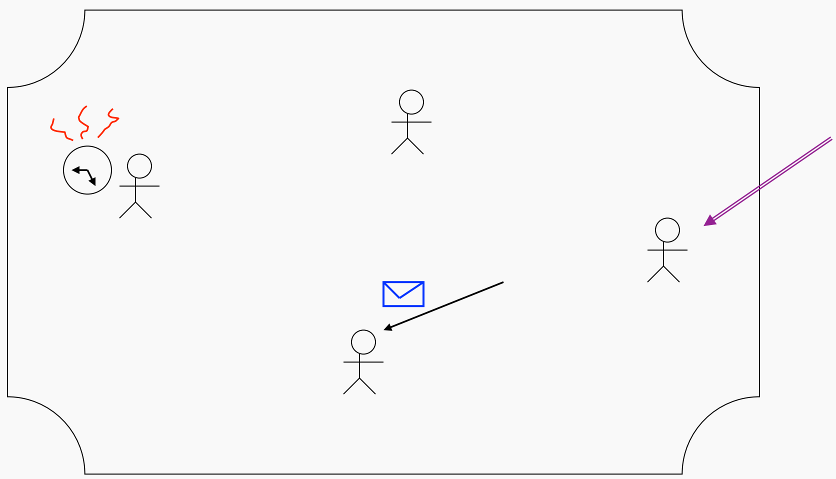
The behavior of an entity is reactive:
triggered by events

Possible events: clock tick
receiving a message
spontaneous impulse


clock ring
Paola Flocchini


message arrival
Paola Flocchini

spontaneous impulse



The reaction of an entity
depends on the event and on its state

State x Event ⟶ Action

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

Entity Behavior

Rule State x Event $\longrightarrow$ 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) $\exists$ an action)

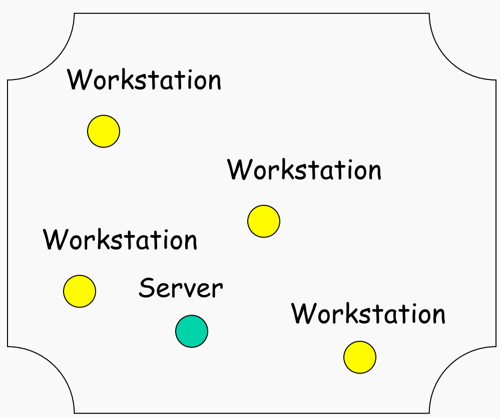
System Behavior

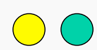
$$B = \{ B(x) : x \in E\}$$

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

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

Property: Every system can be made symmetric





Very different rules, states ....

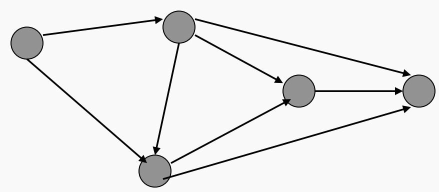
role =(workstation/server)

$s \times e \longrightarrow$
```
If role = workstation
       ActionW(s,e)
else
       ActionS(s,e)
```

Communication

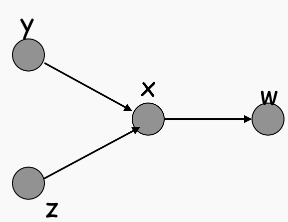
Message: finite sequence of bits

Communication Network:



Communication

Point-to-point Model

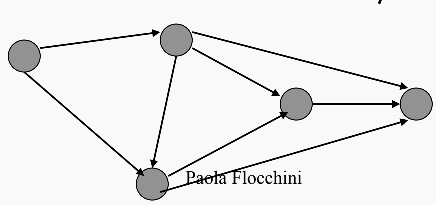


$N_o(x)$ = out-neighbors of entity x
$N_i(x)$ = in-neighbors of entity x
$N(x) = N_o(x) \cup N_i(x)$

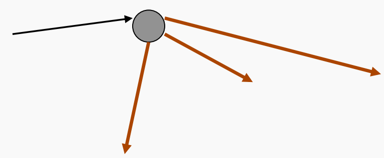
Graph describing the
COMMUNICATION TOPOLOGY

G = (V, A)

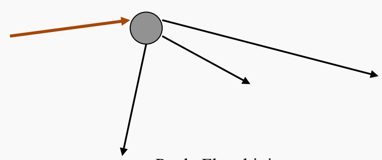
V: Entities
A: Arcs defined by N



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

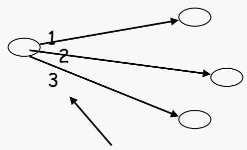
In absence of faults a message from x to its out-neighbour y reaches y in finite time

Local orientation

Each entity distinguishes among its neighbors

Local orientation: more precisely

Each entity distinguishes among its out-neighbors

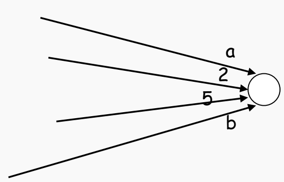


distinct labels
=
(out) port numbers

Send Message **to** 3

Local orientation: more precisely

Each entity distinguishes among its in-neighbors

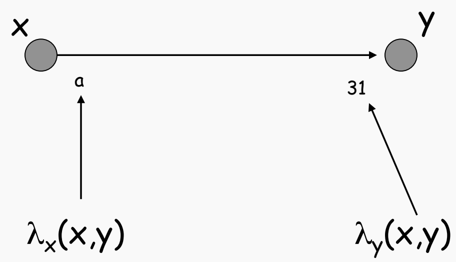


distinct labels
=
(in) port numbers

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

Local orientation: more precisely

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

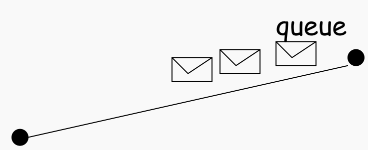


Topology = labeled graph (G, λ)

Paola Flocchini

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.

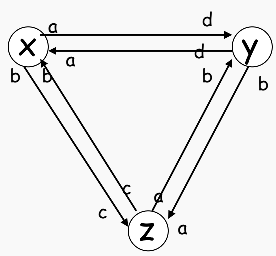
Paola Flocchini

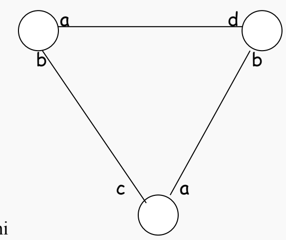
Restrictions of the model: examples

Communication Restrictions

Bidirectional Links

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





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 ....
.....

Paola Flocchini

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

.....

Paola Flocchini

Complexity measures - Performance

1. Amount of communication

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

point of view of SYSTEM

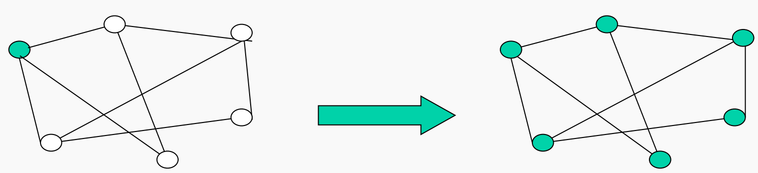
2. Time

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
G is connected

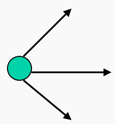
→ Simplifying assumptions

→ Otherwise unsolvable

Algorithm FLOOD

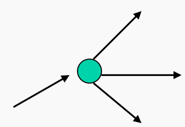
<u>The idea:</u> If an entity knows something, it sends the info to its neighbours

One entity is INITIATOR, the others are SLEEPING



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

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



Paola Flocchini

<u>The idea:</u> 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

Paola Flocchini

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
```

Paola Flocchini

Algorithm for node x:

Event

```
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
```

Paola Flocchini

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

Paola Flocchini

Example

Paola Flocchini

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

Worst for all possible initiators and for all possible executions

m = number of links

Messages: $\leq$ 2 on each link

→ $\leq 2m$

$O(m)$

More precisely:

Let s be the initiator

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

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

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

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

Time Complexity - Ideal Time   Worst Case

Worst for all possible initiators
and for all possible executions

Time: (ideal time)

$\mathrm{Max}_x\{d(x,s)\}$ = eccentricity of s

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

O(n)

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.

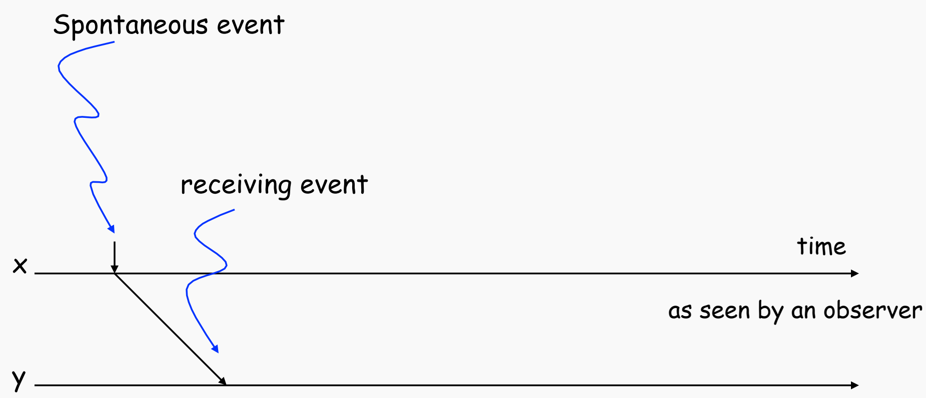
Different delays ---> different executions

Different executions could have different outcomes

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

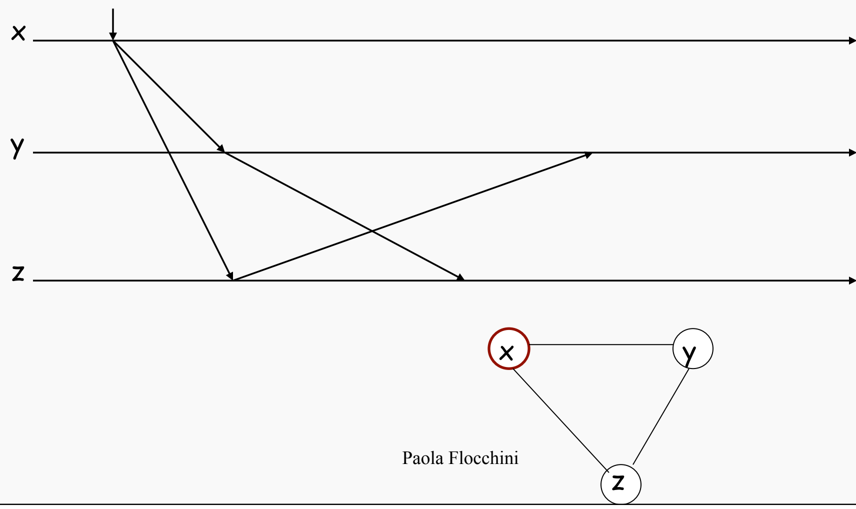
An executions is fully described by the sequence of events that have occurred

Time x Event diagram



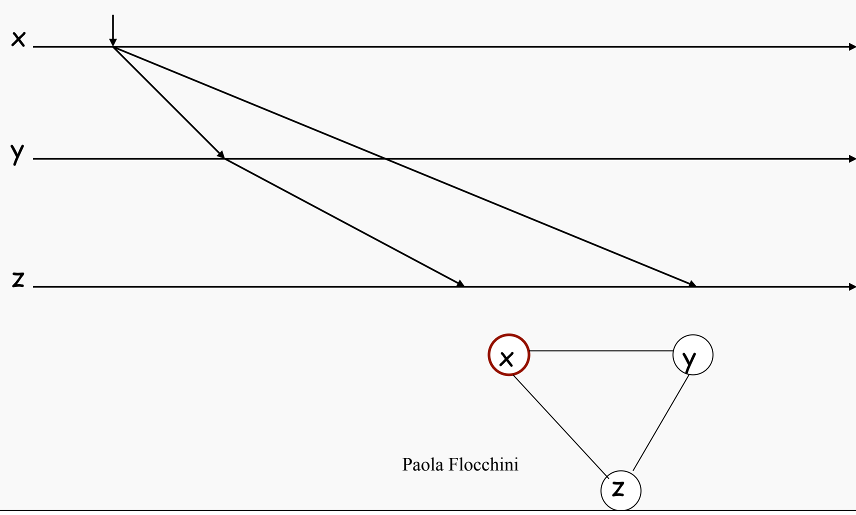
Example: Time x Event diagram of Flooding

One possible execution



Example: Time x Event diagram of Flooding

Another execution

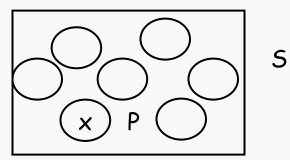


Knowledge

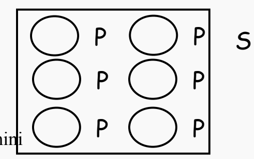
P = fact; x = entity; S = set of entities.

- **Local knowledge LK**
  $P \in LK(x)$.

- **Implicit knowledge IK**
  $P \in IK(S)$ if $\exists x \in S$: $P \in LK(x)$.



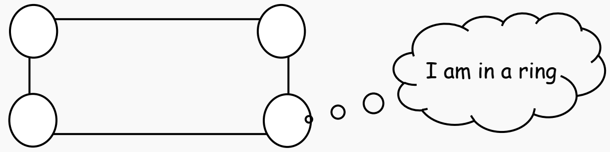
- **Explicit knowledge EK**
  $P \in EK(S)$ if $\forall x \in S$: $P \in LK(x)$.



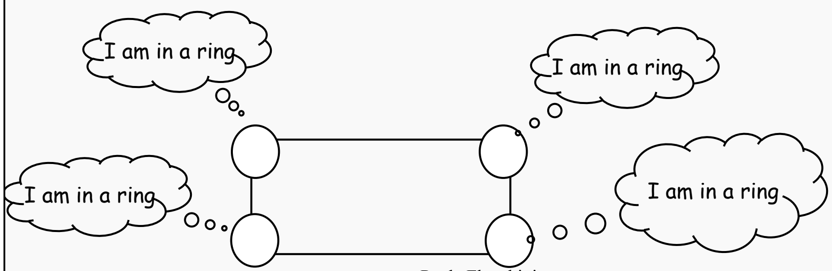
- **Common knowledge CK**
  $P \in CK(S)$ if
  $\forall x \in S, P \in LK(x) \wedge$
  $\forall x \in S\ (\forall x \in S, P \in LK(x)) \in LK(x) \wedge$
  $\forall x \in S\ ((\forall x \in S, P \in LK(x)) \in LK(x)) \in LK(x) \wedge \ldots$

Examples

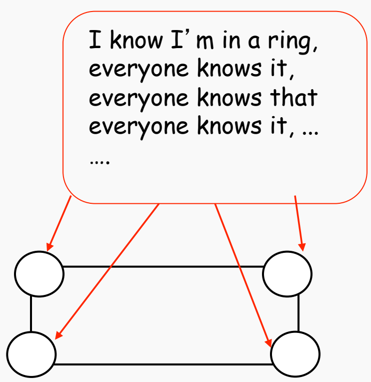
Implicit knowledge



Explicit knowledge



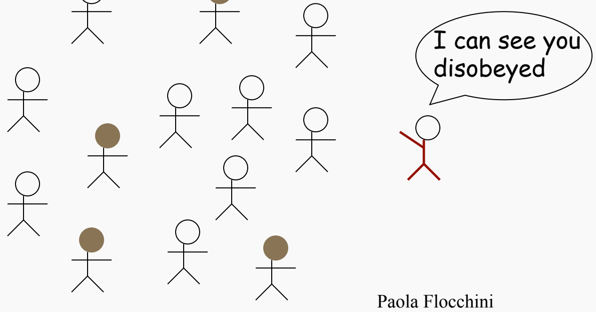
Common knowledge



How to reach common knowledge in FINITE TIME ?

$P \in CK(S)$ if
$\forall x \in S, P \in LK(x) \wedge$
$\forall x \in S\,(\forall x \in S, P \in LK(x)) \in LK(x) \wedge$
$\forall x \in S\,((\forall x \in S, P \in LK(x)) \in LK(x)) \in LK(x) \wedge \ldots$

muddy forehead

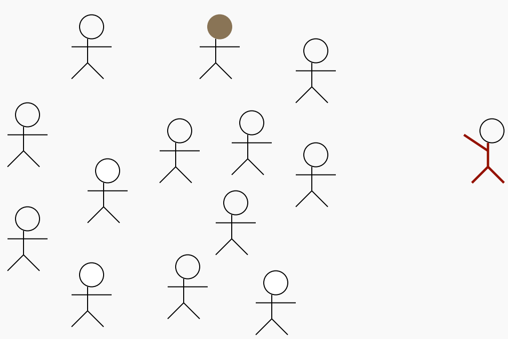


t=1

t=2

t=3

t=4



Some types of knowledge

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

Metric knowlege
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.

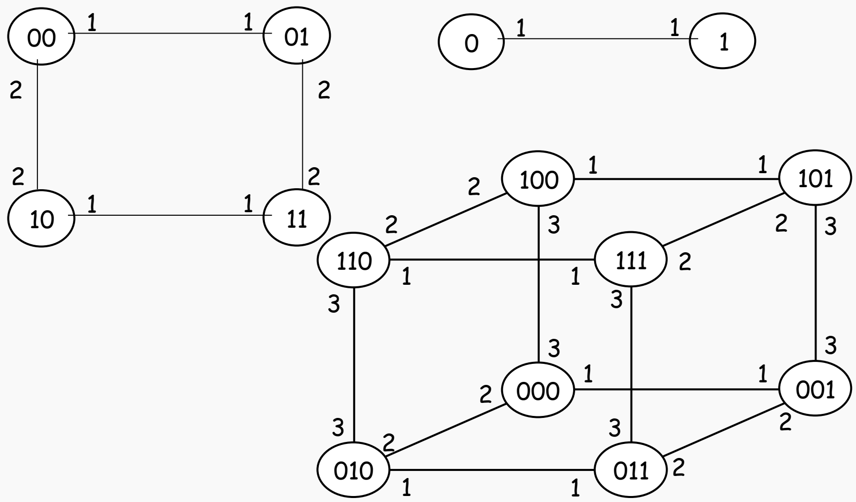
Example: impact of knowledge

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

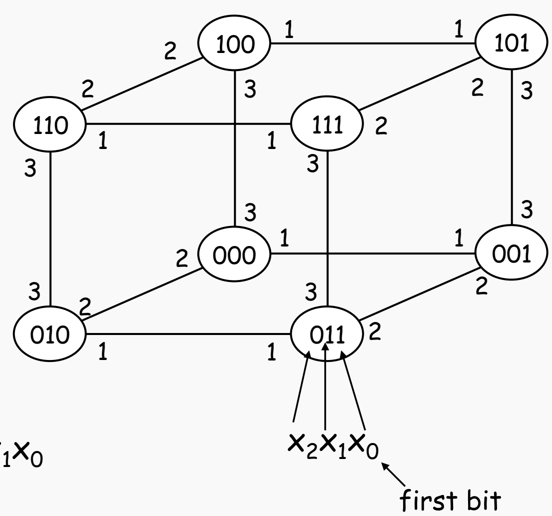
What is the complexity of flooding in a complete graph ?
How can it be done more efficiently ?

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

Example: The labeled hypercube

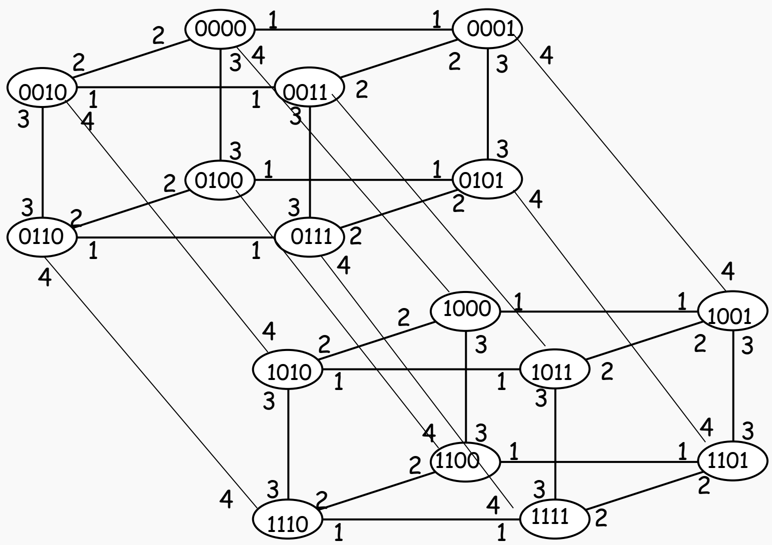


Each link between two nodes is labeled by the dimension of the bit by which the nodes' name differ.



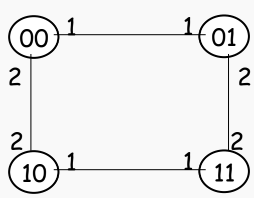
$X = x_k x_{k-1} \ldots\ldots x_1 x_0$

K-bit name

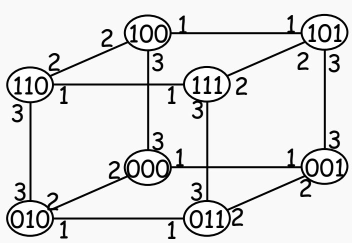


Paola Flocchini

n=2 dimensions=1



n=4 dimensions=2



n=8 dimensions=3

dimensions = log n

Paola Flocchini

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

Each node has d links

➔ $m = n\, d/2 =$ **O(n log n)**

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

HyperFlood - Efficient 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

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

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

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

Paola Flocchini

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 …

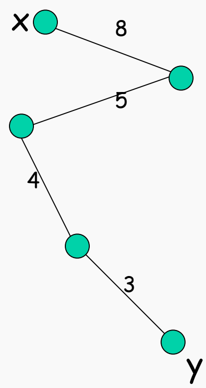
Paola Flocchini

Example:

8 5 4 3

X= 100010100

Y= 110001000



x

100010100

110010100

110000100

110001100

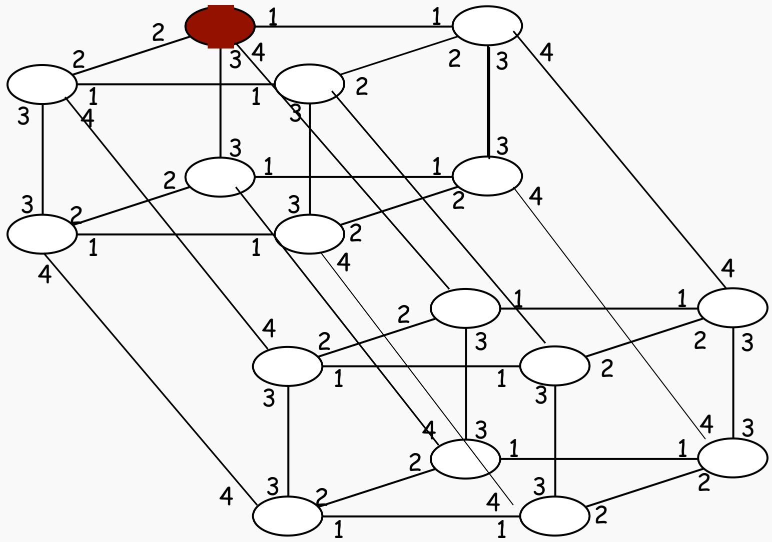
110001000 y

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

So every entity receives the info exactly ONCE.

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

and every node is touched



Complexity: **n-1** (OPTIMAL)

Because every entity receives the info only ONCE.

In Special Topologies

General Flooding: 2m - (n-1)

Algorithm specific for the hypercube: (n-1)

Algorithm specific for the complete network: (n-1)

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

Important Facts

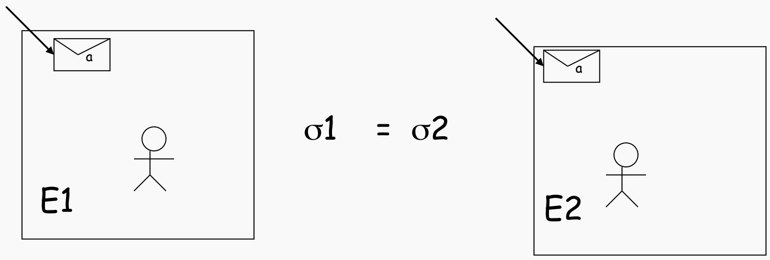
State x Event ---> Action

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

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

Paola Flocchini

E1, E2: different environments



$\sigma 1 = \sigma 2$

1) If the same event happens to x at time t in two different executions and if the internal states $\sigma 1$ and $\sigma 2$ 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.**

An example

Back to Broadcast …

Theorem: Under the set of assumptions:

unique Initiator
G is connected
no failures
bidirectional links

Every generic broadcast protocol requires, in the worst case, m messages.

Lower Bounds for Broadcast

Proof.

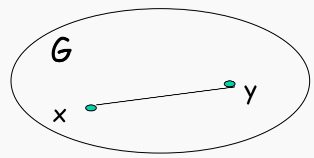
m(G) = n. of edges in G

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.

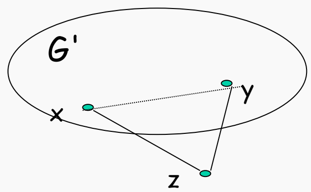
G=(V,E)



Construct a new graph G'

(remember: n is unknown)

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

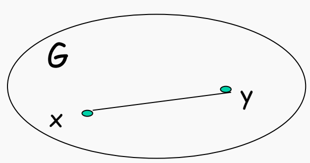


Execute the same algorithm on G' with the same time delays, same initial internal states for all nodes except for z which is `sleeping`

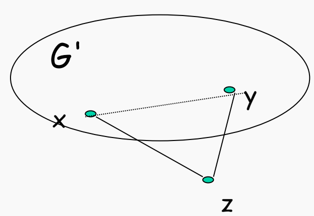
Paola Flocchini

Two executions in two environments

E1



E2



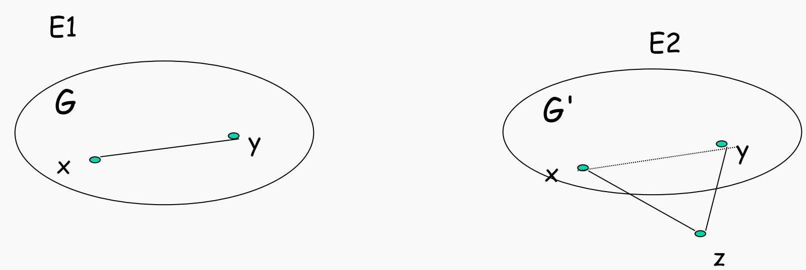
For all nodes, except z, the two executions are **identical**

x and y never send to each other in E1

--->

x and y never send to z in E2

Paola Flocchini



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