Le Modèle et les Opérations de Bases

Chapitres 1 et 2

Le Modèle

Broadcast (Diffusion)

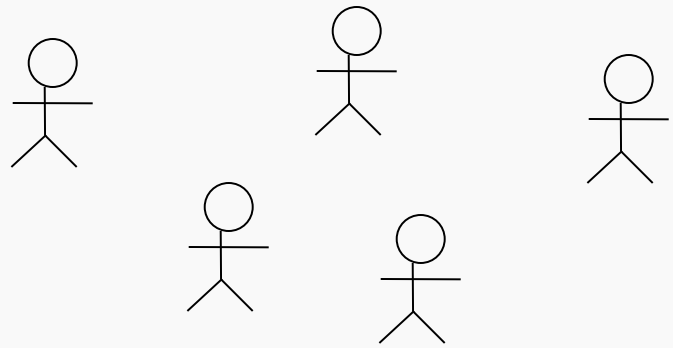
Construction d'un arbre couvrant

Traversal

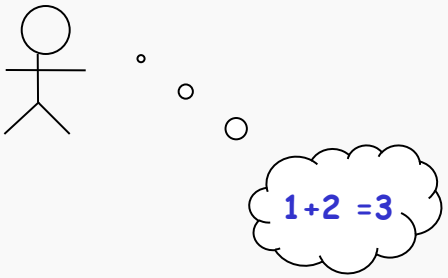
Wakeup

Systèmes répartis

Multiplicité



Autonomie

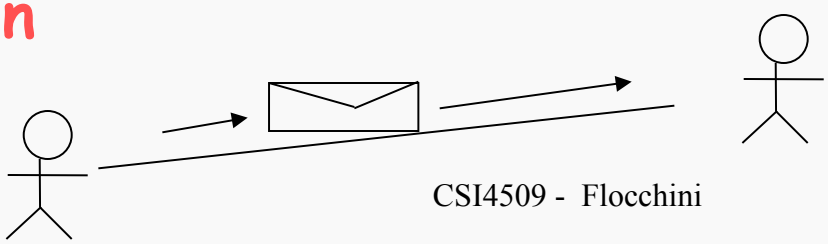


Horloge

Mémoire

Capacité de calcul

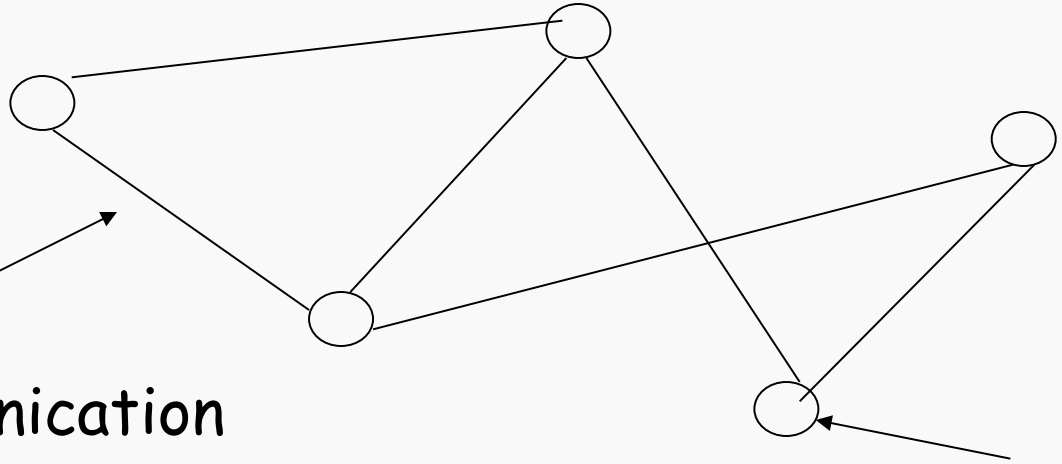
Interaction



Typiquement par échange de messages

Le Modèle

Systèmes répartis = Ensemble d'entités qui communiquent en échangeant des messages

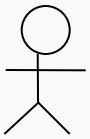


Lien de communication

entité, noeud, site ...

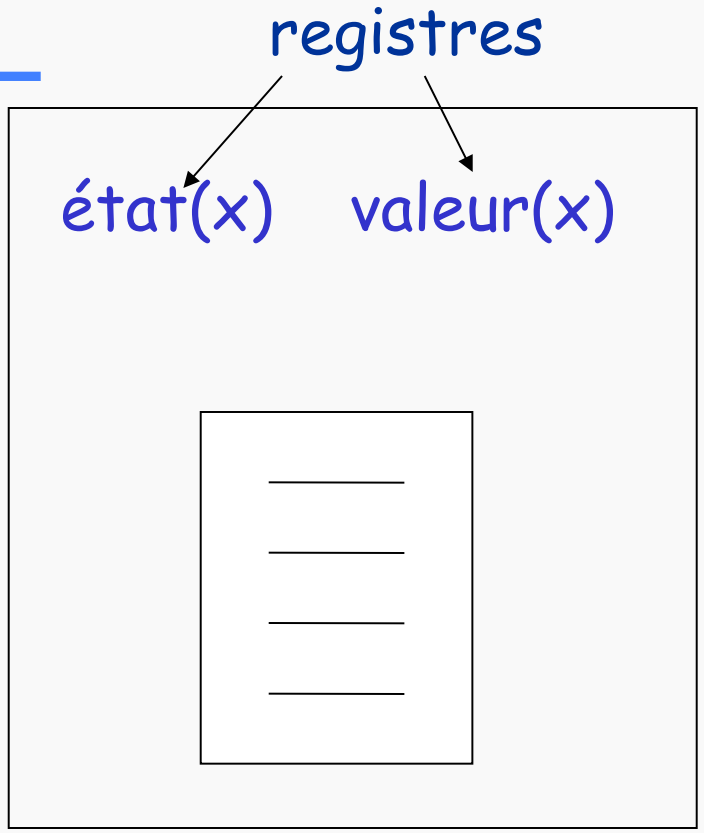
"message passing"

(processeur, processus, objet ....)

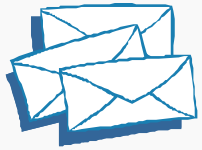


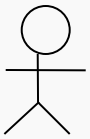
Entité x

En mémoire:



Opérations possibles

- calcul local et exécution
- transmission de messages 
- (ré) initialisation de l'horloge
- changement de la valeur des registres 



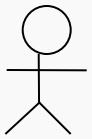
Entité x

etat(x)

Les états possibles constituent un ensemble fini
(ex: {idle, computing, waiting, processing ….})

Ils sont toujours définis

A tout moments, une entité doit être
dans l'un ou l'autre des états



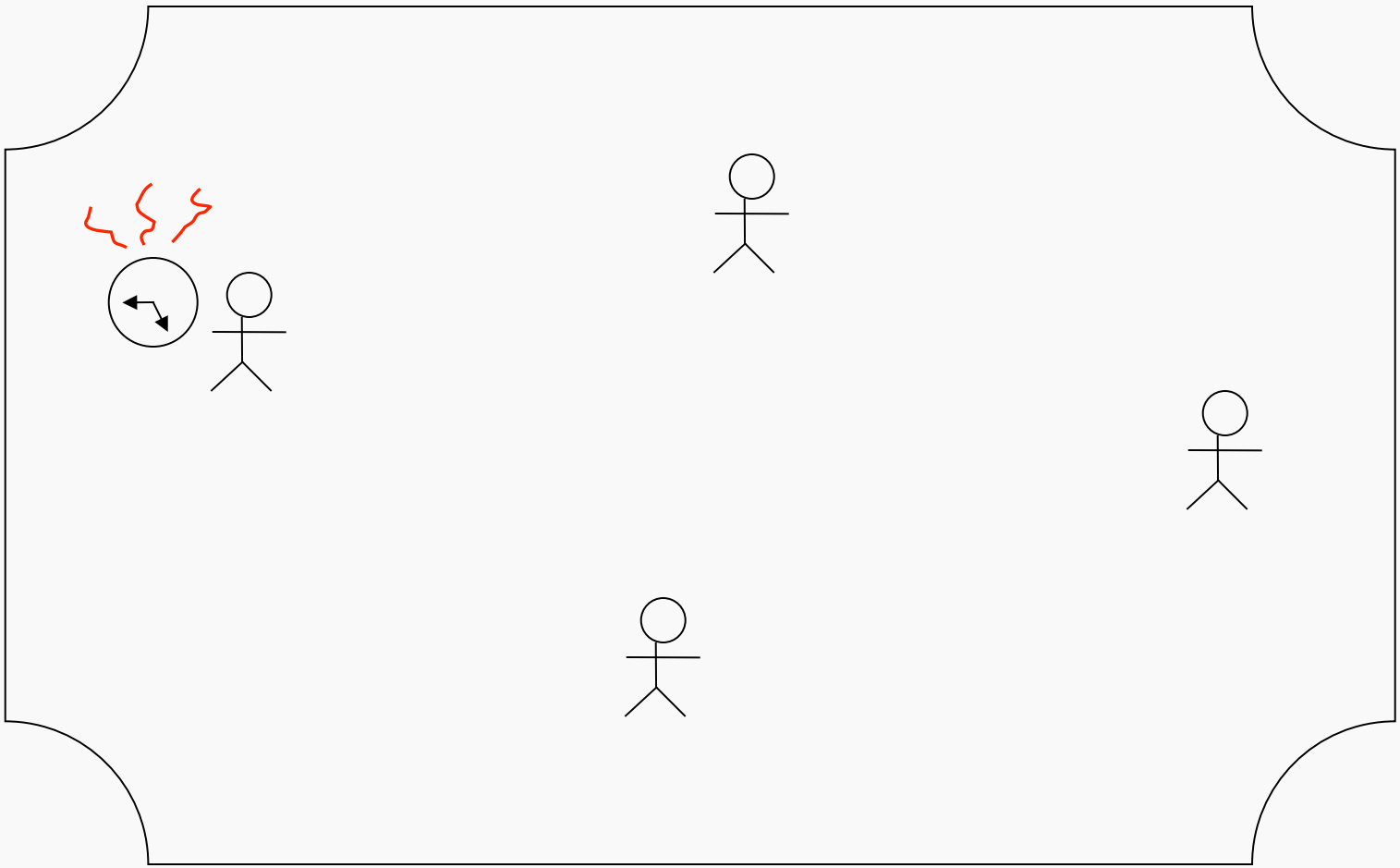
Évènement Externe

Le comportement d'une entité est réactif:
Activé par évènements

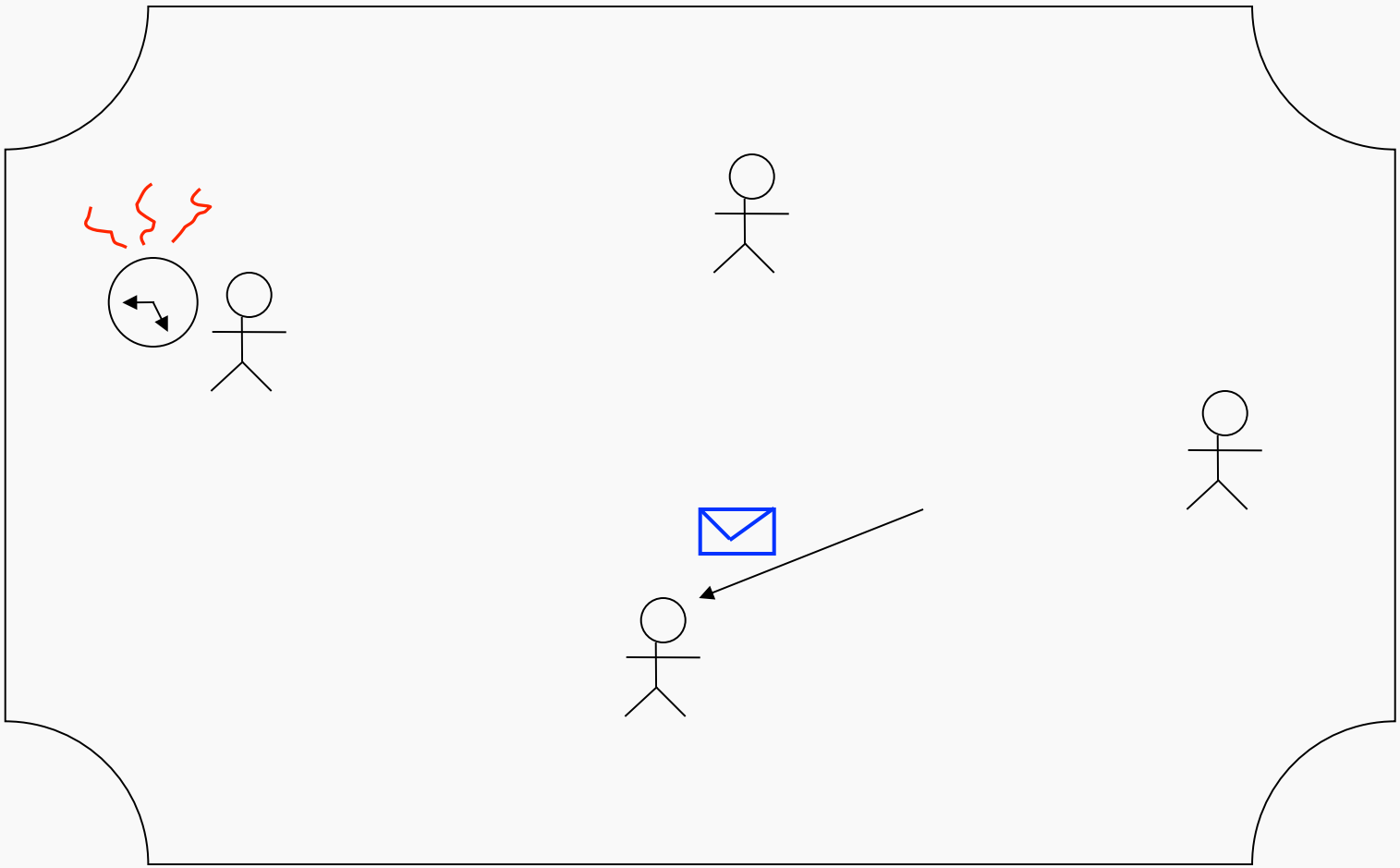
Évènements possibles:

Coup d'horloge
Réception d'un message
Déclanchement spontané

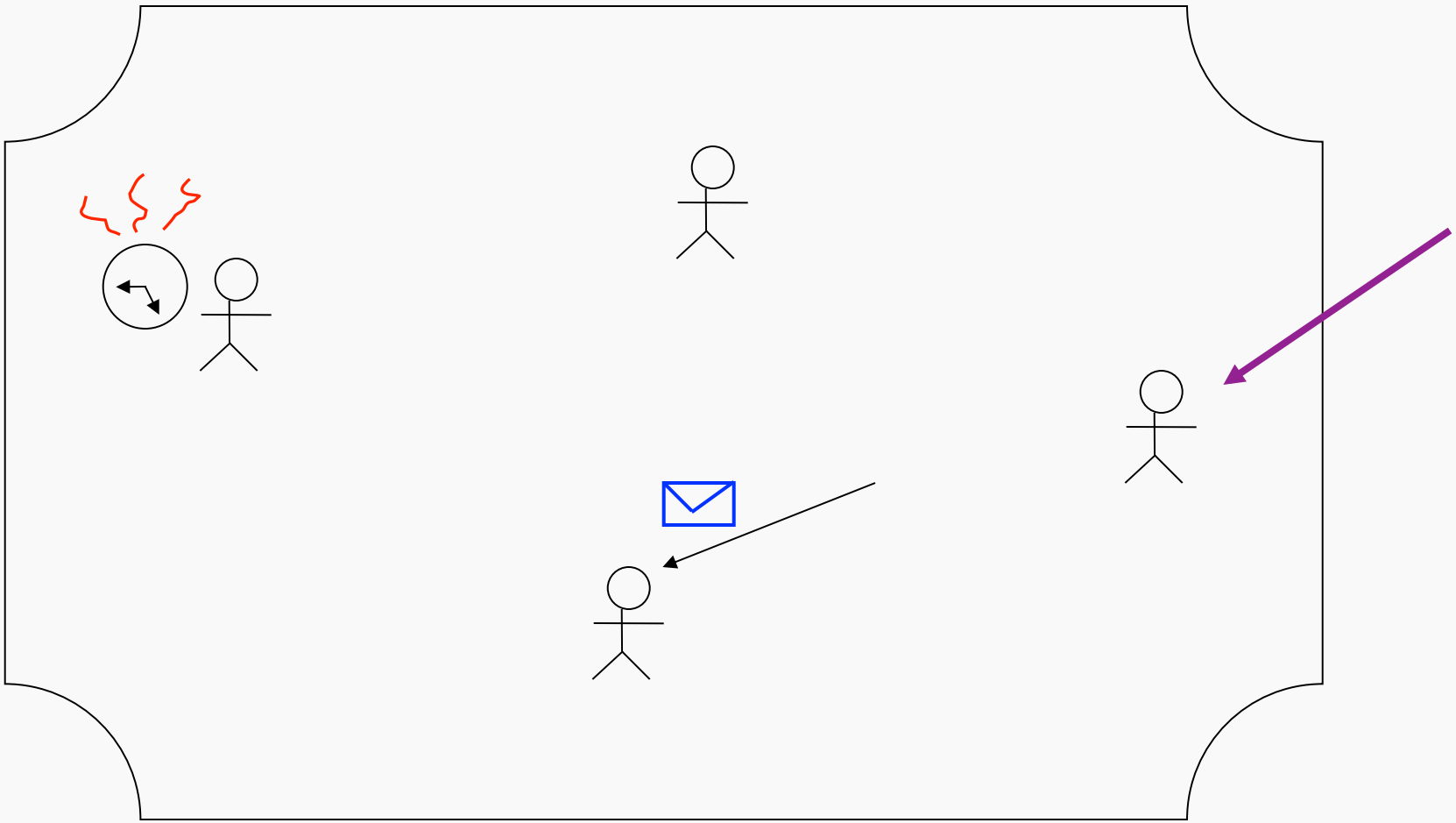
Coup d'horloge



Arrivée d'un message

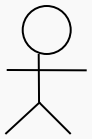


Déclanchement spontané



La réaction d'une entité dépend de son état et de l'évènement déclencheur

Etat x Évènement ⟶ Action



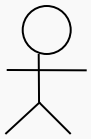
Actions

Action: sequence d'activités

- calculs
- envoi de messages
- changement d'état

Tout action est atomique
Les activités ne peuvent pas être interrompues

Tout action termine
Les activités doivent terminer en un temps fini



Comportement des Entités

Règles

État x Évènement ⟶ Action

Le comportement B(x) = à l'ensemble de règles de l'entité x pour tous les évènements et états possibles

L'algorithme
Le protocole

DÉTERMINISTE
(État, Évènement) --> une seule action

COMPLET
(∀ (État, Évènement) ∃ une action)

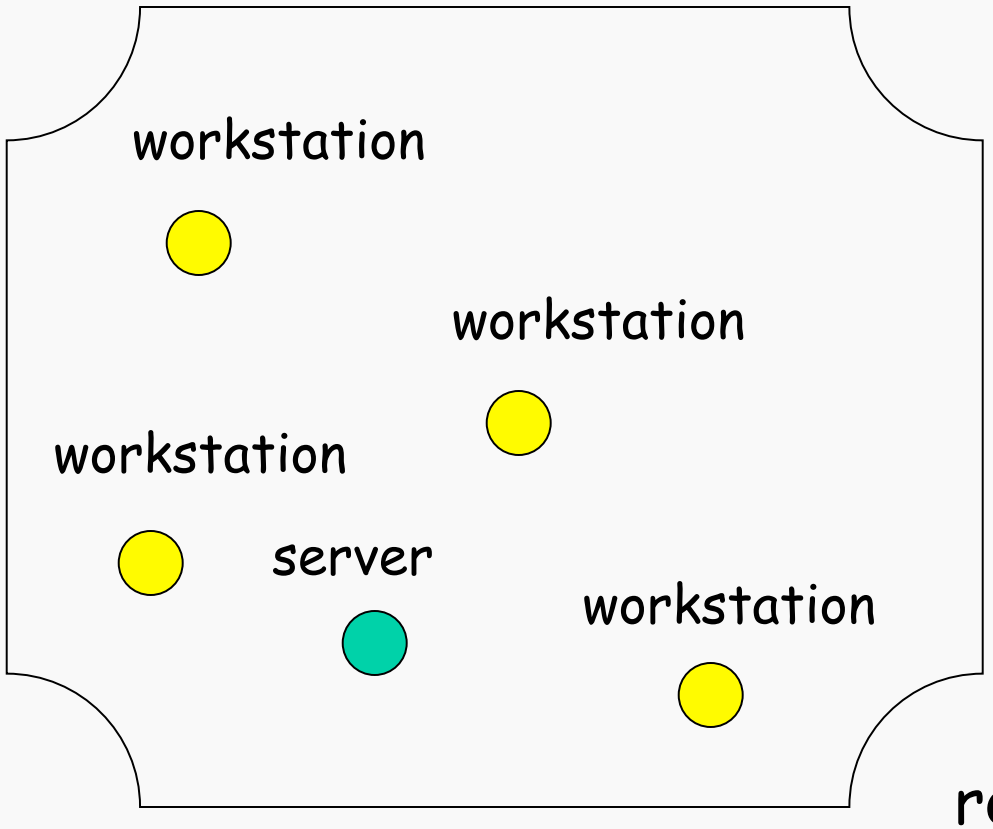
Comportement du Système

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

Un système est SYMÉTRIQUE (ou homogène)
si tous les entités ont le même comportement

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

Propriété: Tout système peut être transformé en système symétrique



États et règles très différentes

role =(workstation/server)

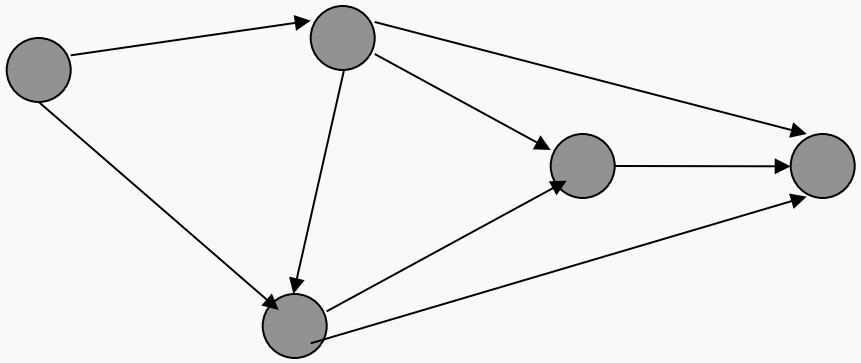
s x e ⟶

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

Communication

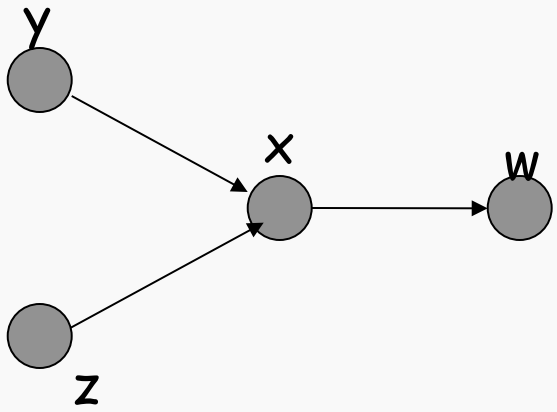
Message: Séquence finie de bits

Réseau de Communication:



Communication

Modèle point-à-point

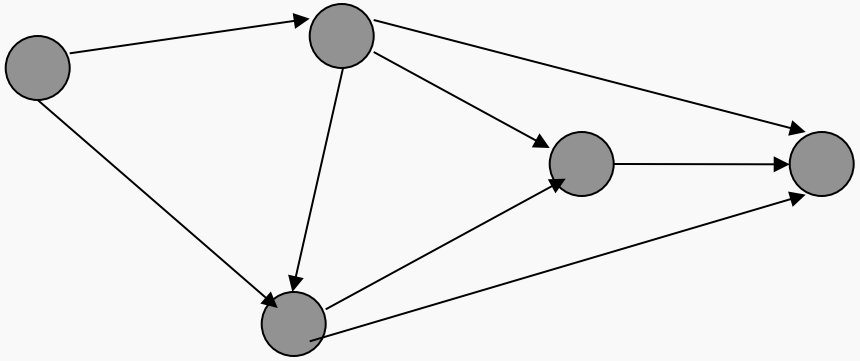


$N_o(x)$ = voisins sortant de l'entité x
$N_i(x)$ = voisins entrant de l'entité x
$N(x) = N_o(x) \cup N_i(x)$

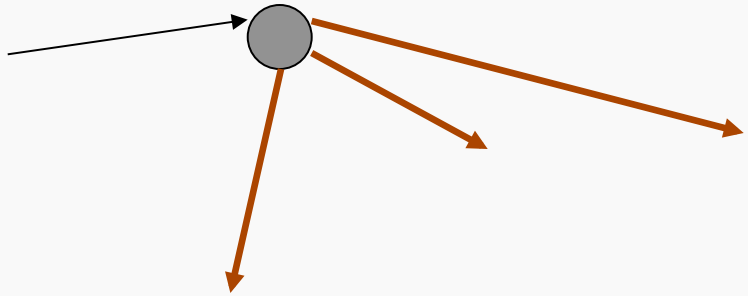
Graph décrivant la
TOPOLOGIE DE COMMUNICATION

$G = (V, A)$ V: Entités

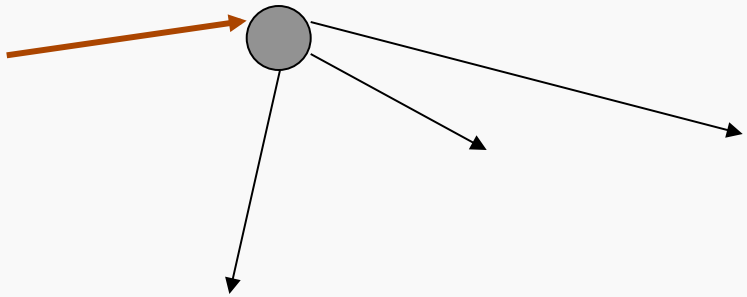
A: Arcs définies par N



Une entité x peut envoyer un message seulement à ses voisins sortants $N_o(x)$



Et en reçevoir seulement de ses voisins entrant $N_i(x)$



Axiomes

Délais de Transmission finis

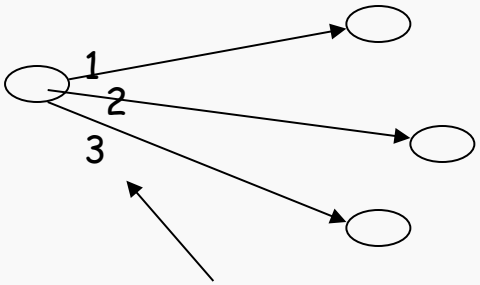
En l'absence d'erreurs, un message provenant de x et se dirigeant vers le voisin sortant y atteint y en un temps fini.

Orientation Locale

Chaque entité peut distinguer ses voisins les uns des autres

Orientation Locale: Précisions

Chaque entité peut distinguer ses voisins sortants les uns des autres

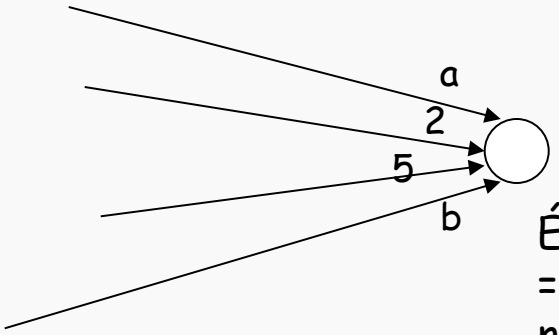


Étiquette distinctes
=
numéros de port sortant

Send Message **to** 3

Orientation Locale: Précisions

Chaque entité peut distinguer ses voisins entrant les uns des autres

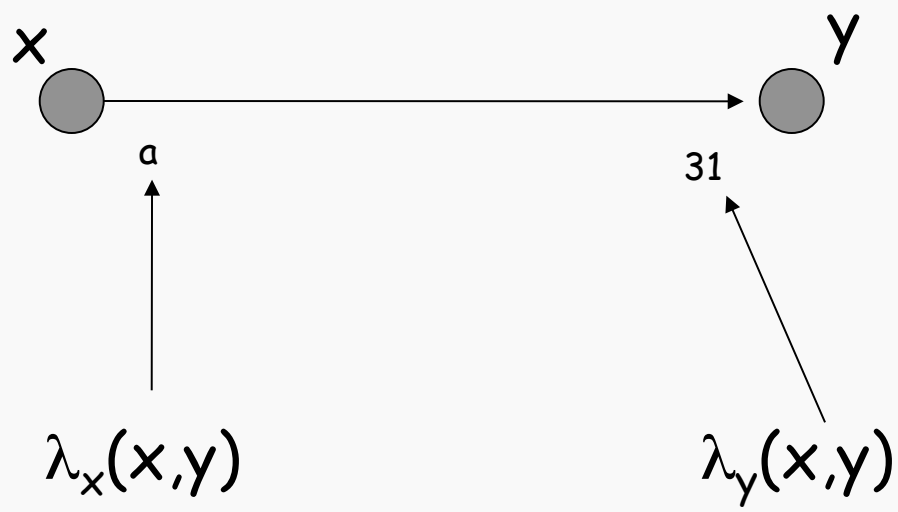


Étiquette distinctes
=
numéros de port entrant

Lorsqu'un message est reçu, l'entité peut détecter le port par lequel il a été acheminé.

Orientation Locale: Précisions

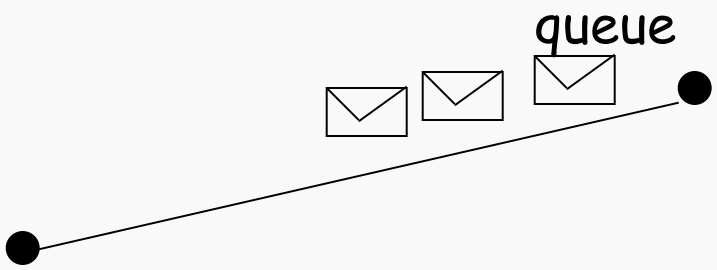
Pour un lien (x,y) il y a 2 étiquettes:



Topologie = graph étiqueté (G, λ)

Restrictions du modèle: examples

Restrictions de Communication



Messages Ordonnés

(FIFO)

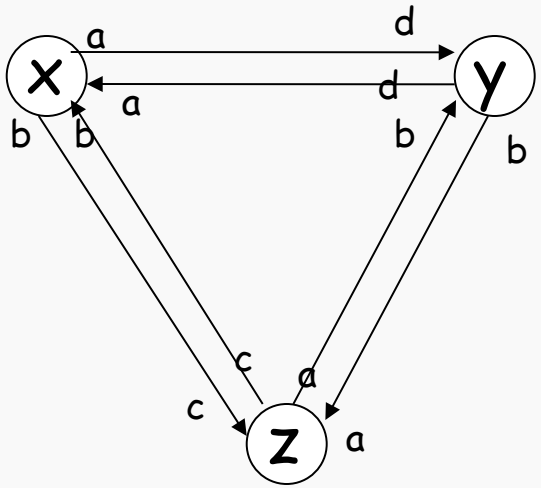
En l'absence d'erreurs, les messages transmis le long d'un même lien sont reçus dans le même ordre.

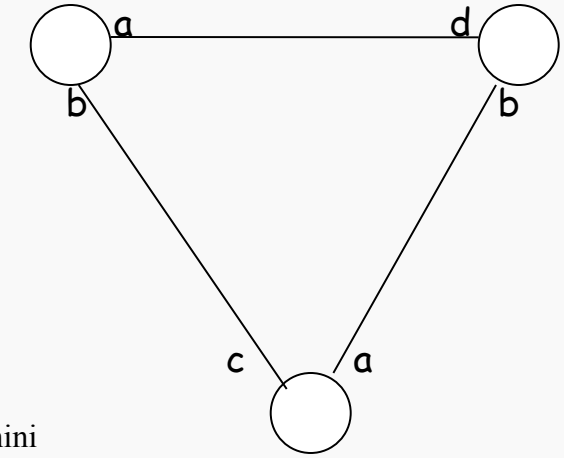
Restrictions du modèle: examples

Restrictions de Communication

Liens Bidirectionnels

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





Restrictions du modèle: examples

Restrictions de Fiabilité:

1. *Livraison Assurée:*
   Tout message envoyé sera reçu sans erreurs

2. *Fiabilité partielle:*
   Aucune erreur ne surviendra

3. Fiabilité Complète:
   Aucune erreur n'est survenue ni ne surviendra

Restrictions du modèle: examples

Restrictions Topologiques:

Le graph G est fortement connecté

.....

Restrictions sur les Connaissances

Conaissance du nombre de noeuds
Conaissance du nombre de liens
Conaissance du diamètre
....

Restrictions du modèle: examples

Restrictions Temporelles:

Délai de Communication limité:

Il existe une constante Δ tel qu'en l'absence d'erreurs, le délai de communication d'un message sera au plus Δ

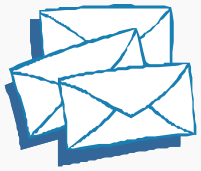
Horloges Synchronisées:

Tous les horloges locales sont incrémentées d'une unité simultanément et à intervales réguliers

.....

Mesure de Complexité - Performance

1. Complexité de Communication



Nombre de messages échangés
(pour plus grande précision: nombre de bits)

Point de vue du SYSTÈME

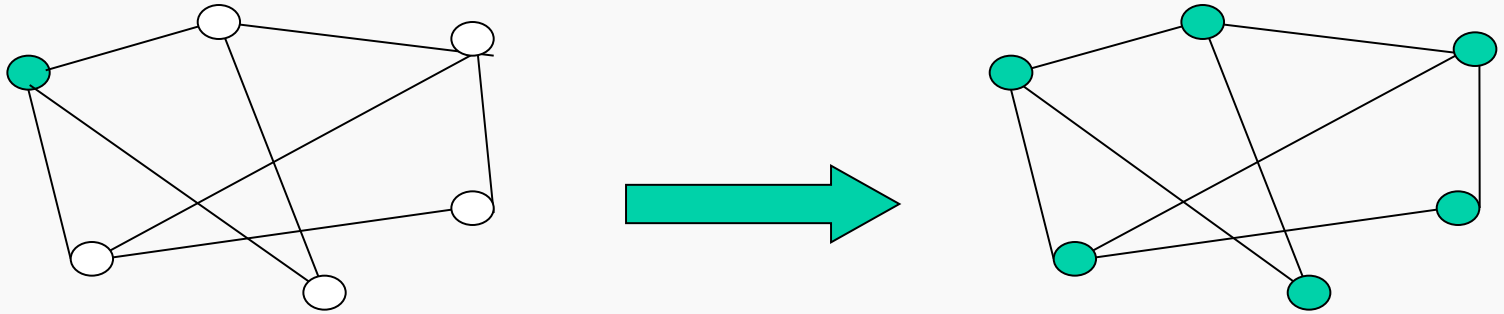
2. Temps

Point de vue de L'USAGER

Les délais de communications sont, en général, imprédictibles !!!

Temps Idéal:
La transmission d'un message a une durée d'une unité de temps

Example - Broadcast (Diffusion)



Suppositions = restrictions

Initiateur unique
Fiabilité complète
liens bidirectionnels
G est connecté

Par définition

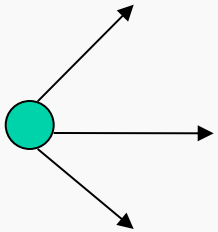
Simplification des suppositions

Nécessaire pour résoudre le problème

Algoritme FLOOD

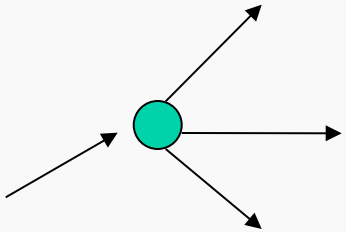
Le concept: Si une entité possède une information, elle l'envoie à ses voisins.

Une entité est INITIATOR, les autres sont SLEEPING



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

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



Le concept: Si une entité reçoit une information, elle l'envoie à ses voisins à l'exception de l'expéditeur du message

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

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

Il n'y a pas de terminaison

$S = \{$`initiator, sleeping, done`$\}$

Algorithme pour noeud x:

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

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

Algorithme FLOOD

Algorithme pour noeud x:

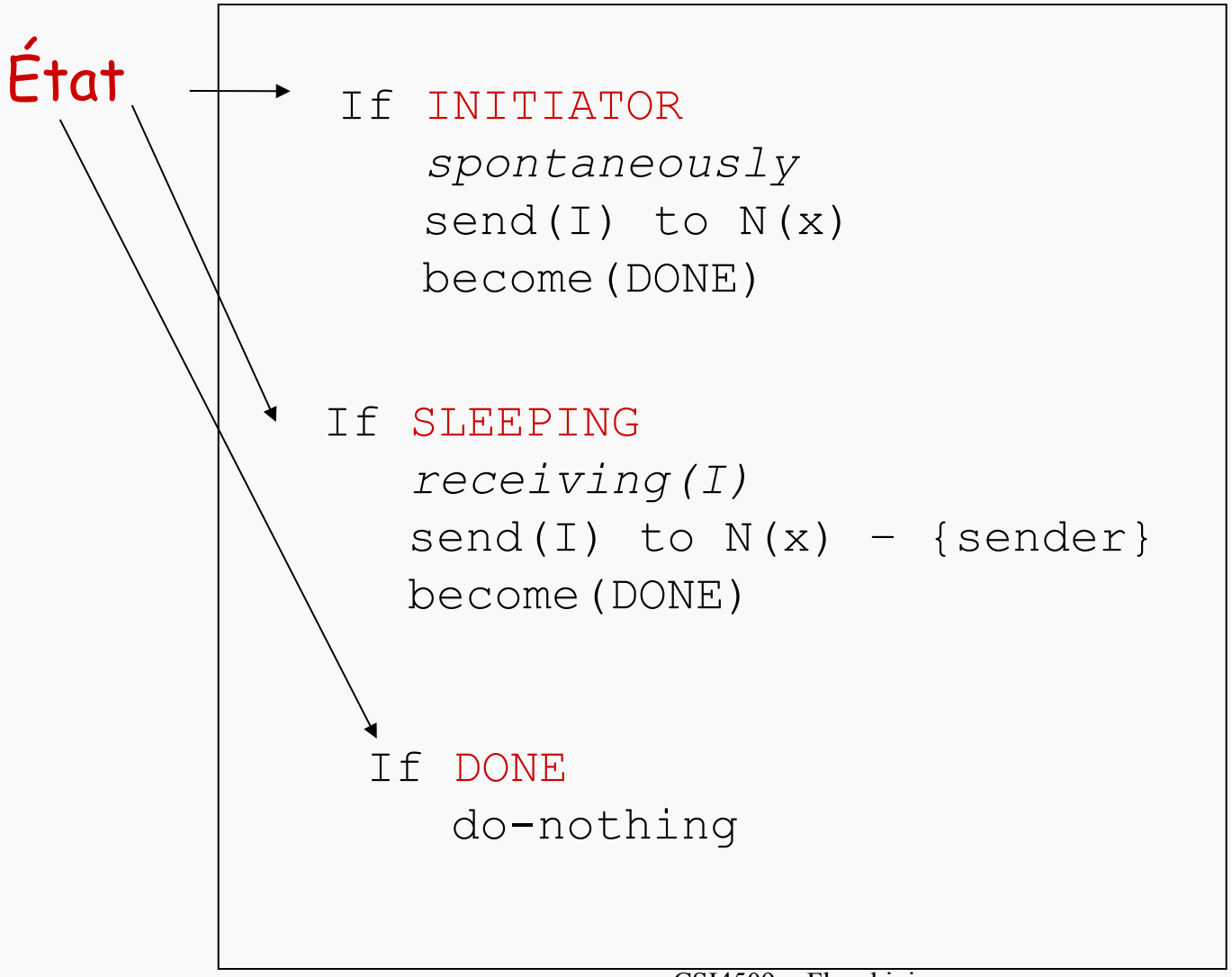
```
INITIATOR
spontaneously
    send(I) to N(x)
    become(DONE)
```

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

```
DONE
```

Algorithme FLOOD

Algorithme pour noeud x:



Algorithme pour noeud x:

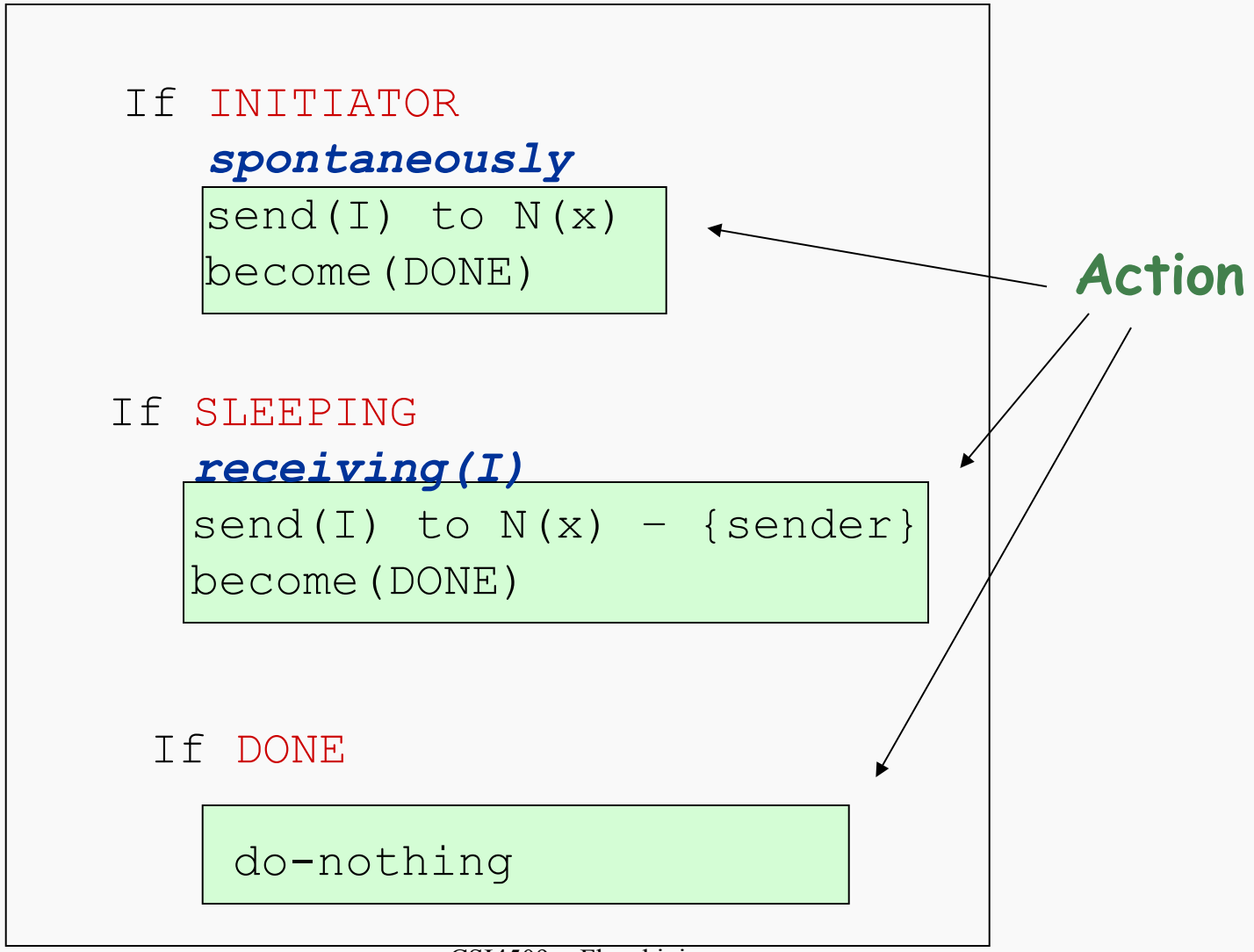
```
If INITIATOR
   spontaneously
   send(I) to N(x)
   become(DONE)

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

If DONE
   do-nothing
```

Évènement

Algorithme pour noeud x:



Example

Exactitude

L'Algorithme termine en un temps fini

parce que:
G est un graphe connexe et il y a fiabilité complète

Terminaison

Terminaison locale : lorsque `DONE`

Terminaison globale: ?

Complexité de Messages

Cas pire

m = nombre de liens

Cas pire pour tout initiateur
et exécution possibles

Messages: ≤ 2 sur chaque lien

⟶ ≤ 2m

O(m)

Plus précisément:

Soit S l'initiateur

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

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

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

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

Complexité temporelle – Temps idéal Cas Pire

Cas pire pour tout initiateurs
et exécutions possibles

Temps: (temps idéal)

$\max_x\{d(x,s)\}$ = eccentricité de s

≤ Diamètre(G) ≤ n-1

O(n)

Temps et Évènements

Évènements externes:

spontaneously
receiving
when (horloge)

Les actions peuvent causer des évènements

send génère ***receiving***
set-clock génère ***when***

Les évènements générés pourraient ne pas avoir lieu (en cas d'erreur).
S'ils ont lieu, ils ont lieu plus tard.

Dans le cas de réception avec délais imprédictibles.

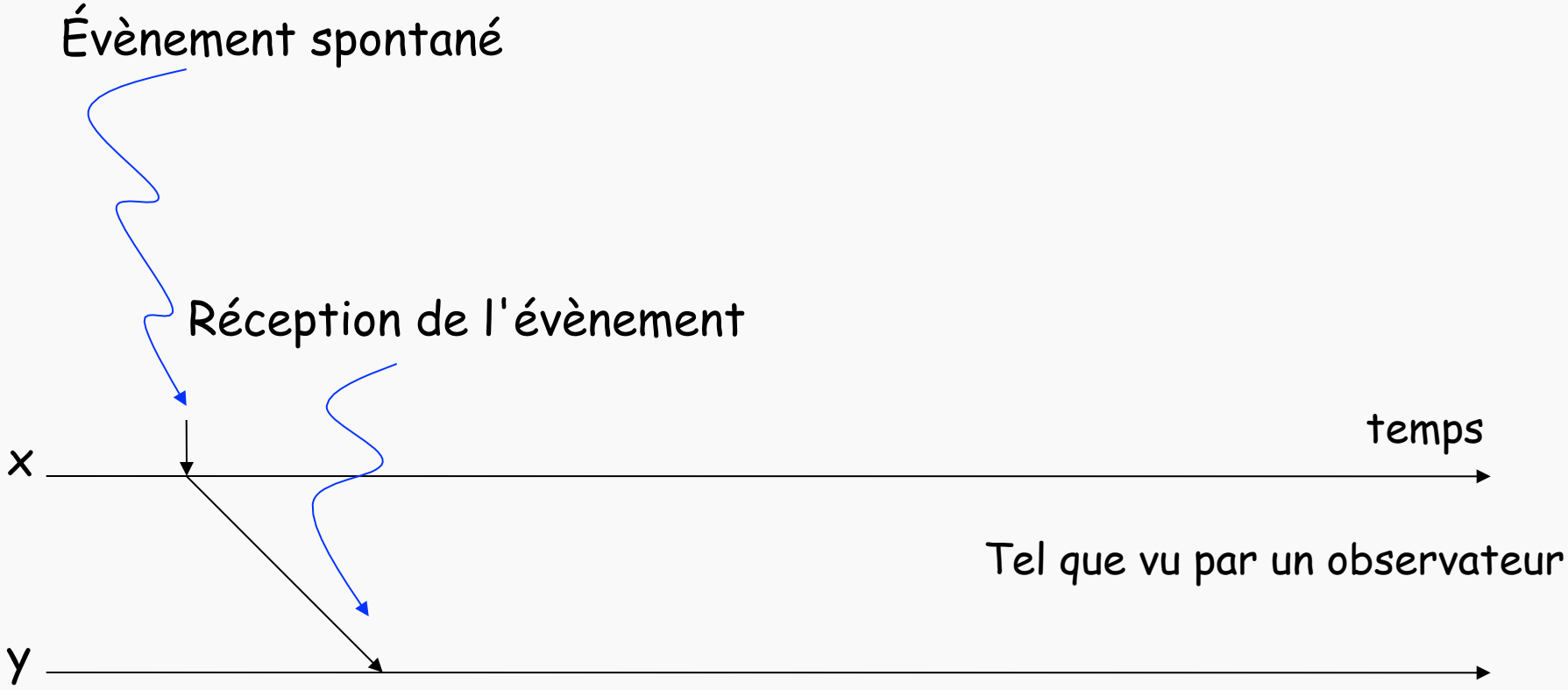
Différents Délais ---> différentes exécutions

Différentes exécutions peuvent avoir des résultats différents

(Les évènements spontanés sont considérés comme générés avant le début de l'exécution: évènements initiaux)

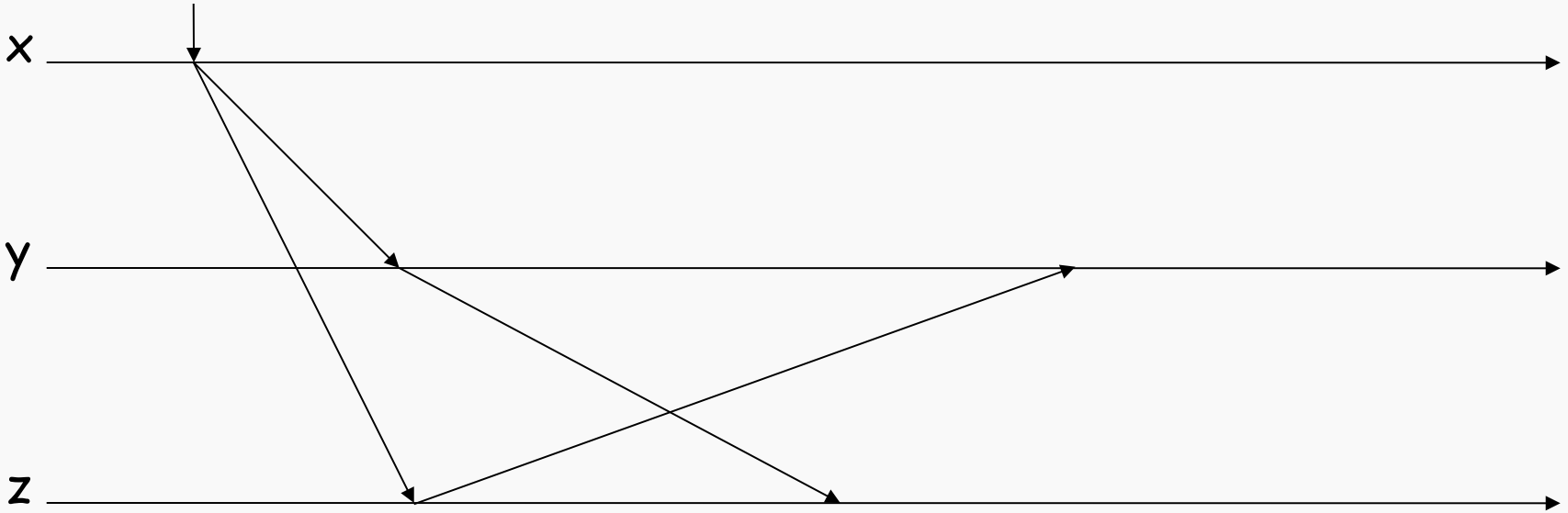
Une exécution peut être caractérisée par la séquence d'évènements qui ont eu lieu

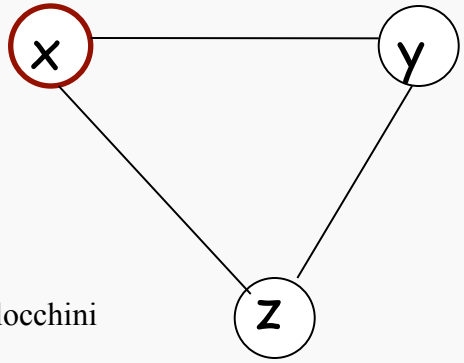
Diagramme temps x Évènement



Example: Diagramme Temps x Évènement de Flooding

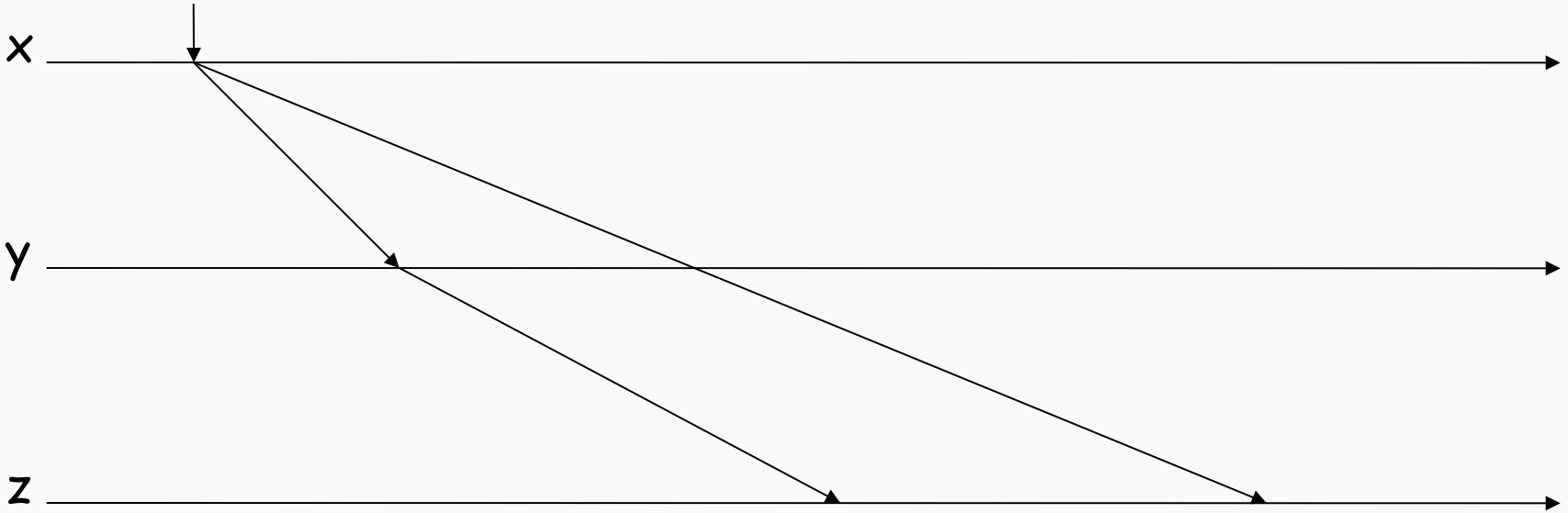
Une exécution possible

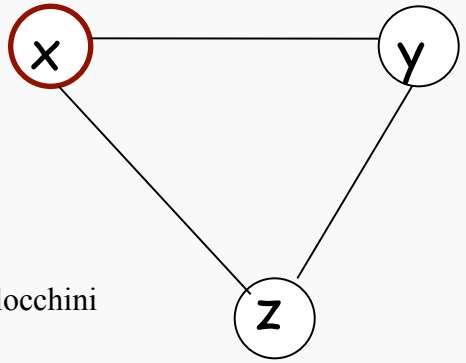




Example: Diagramme Temps x Évènement de Flooding

Une autre exécution



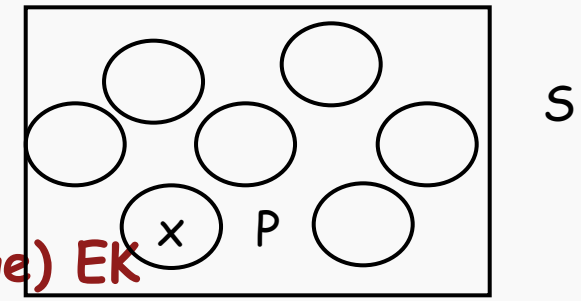


Connaissances

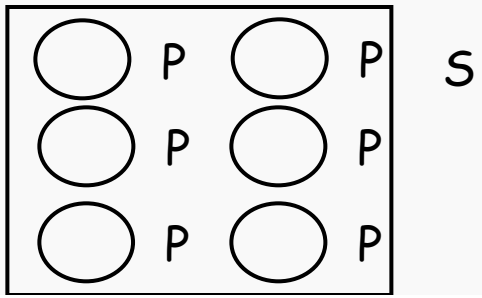
P = fait; x = entité; S = ensemble d'entités.

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

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



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



- **Connaissances communes (Common knowledge) CK**

$P \in CK(S)$ if

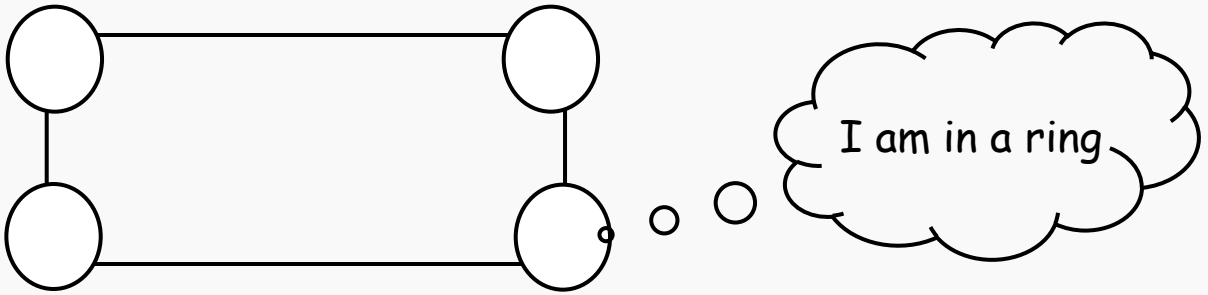
$\forall x \in S, P \in LK(x) \wedge$

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

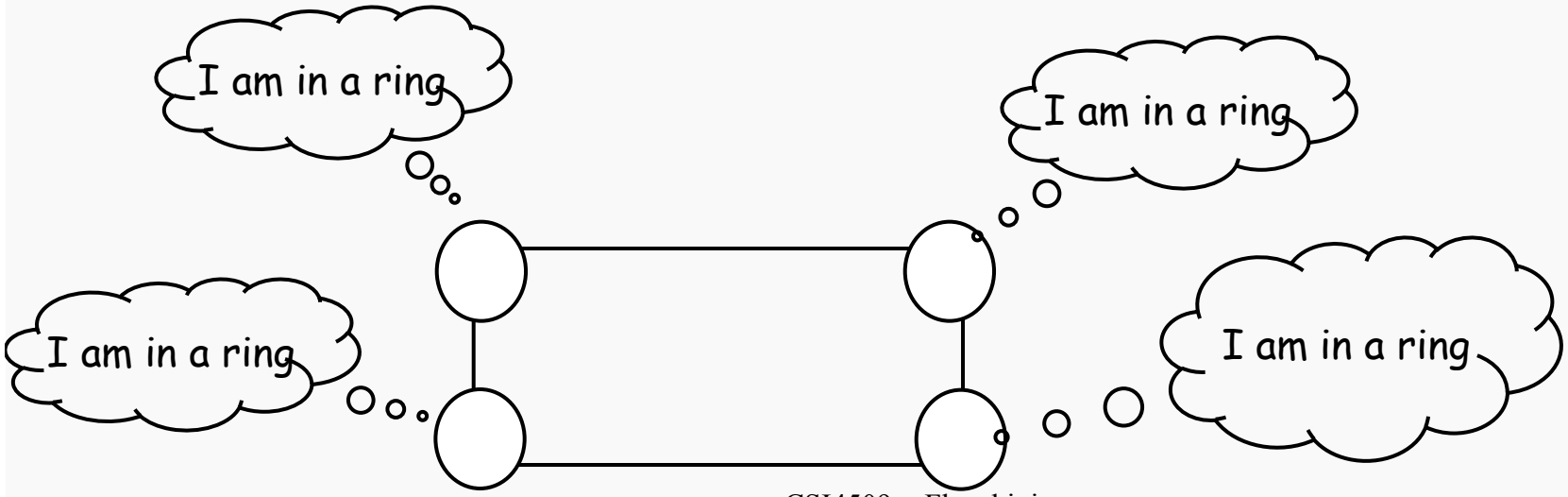
$\forall x \in S\ ((\forall x \in S, P \in LK(x)) \in LK(x)) \in LK(x) \wedge \ldots$

Examples

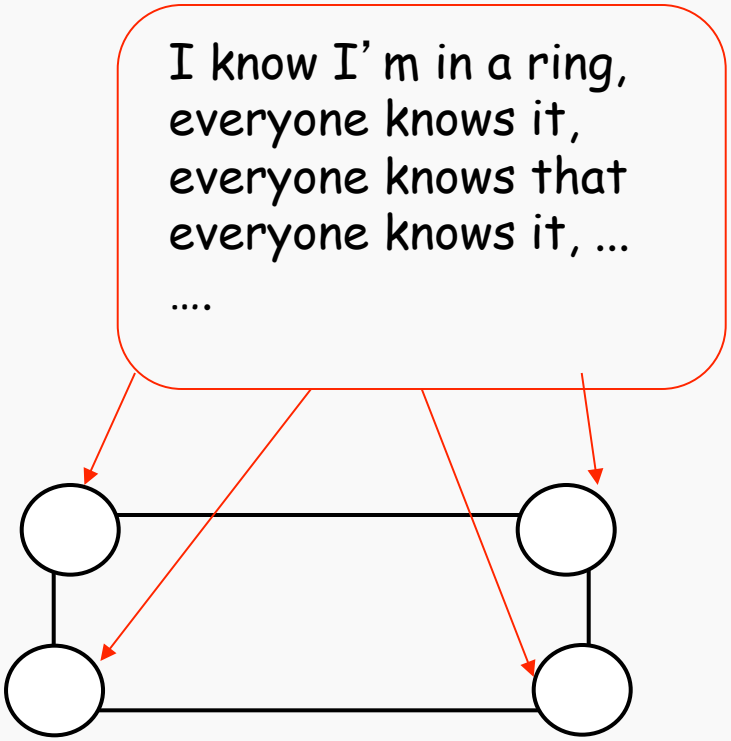
Connaissances implicites



Connaissances explicites

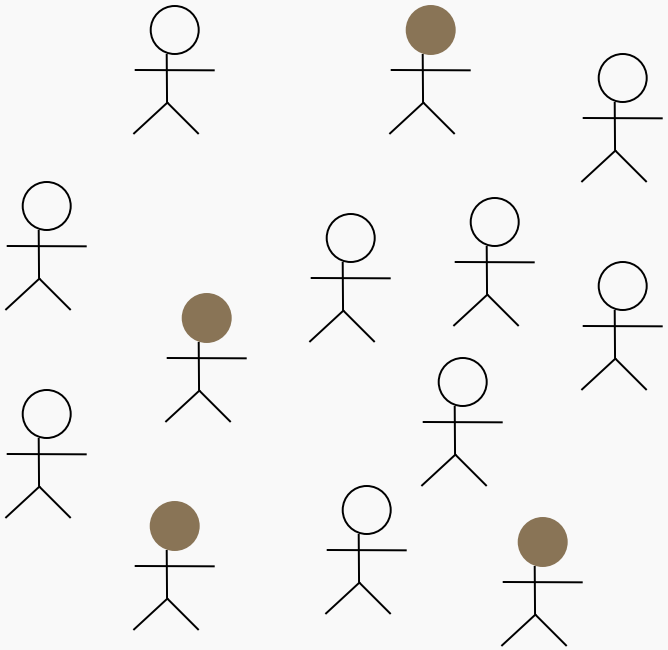


Connaissances communes

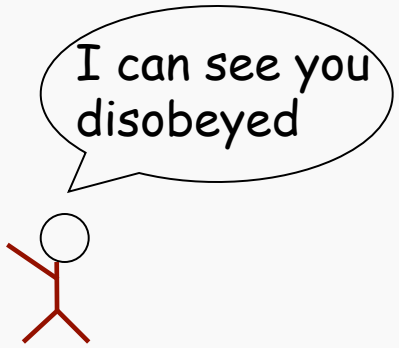


Comment atteindre le niveau de connaissance commune en un TEMPS FINI?

$P \in CK(S)$ 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

Quelques types de connaissances

Connaissance de la topologie
Type de graph ("G est un ring"...), matrice d'adjacence de G ...

Connaissances métriques
Nombre de noeuds, diamètre, eccentricité...

Sens d'orientation
Information par rapport aux étiquettes des liens
Information par rapport aux étiquettes des noeuds

Lorque la quantité de connaissance nécessaire grandit, l'algorithme deviens moins polyvalent. Les algorithmes génériques utilisent aucune connaissance.

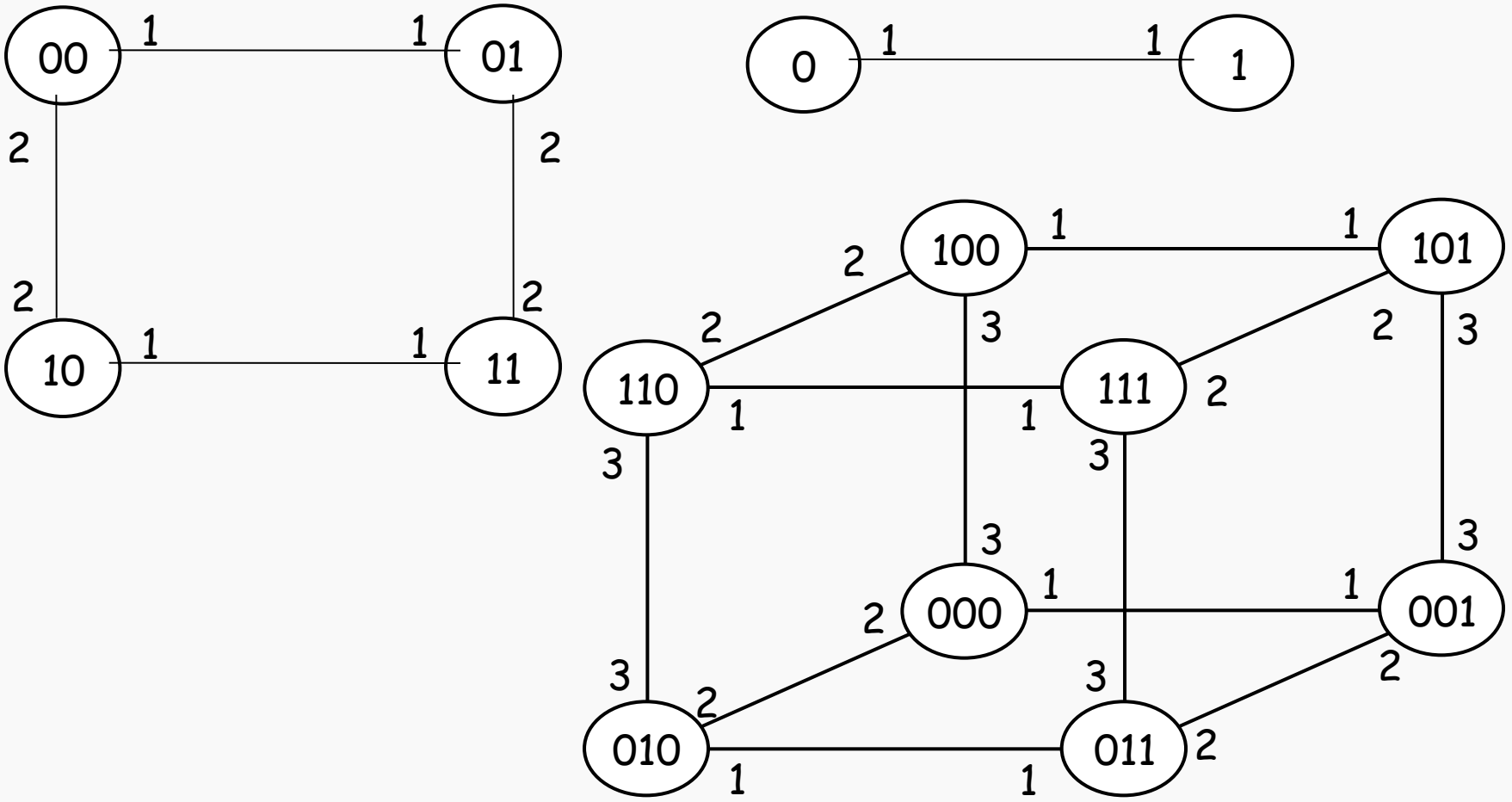
Example: l'effet de la connaissance

Dans certaines topologies, l'algorithme FLOOD peut être évité et broadcast peut être beaucoup plus efficace (si la topologie est connue).

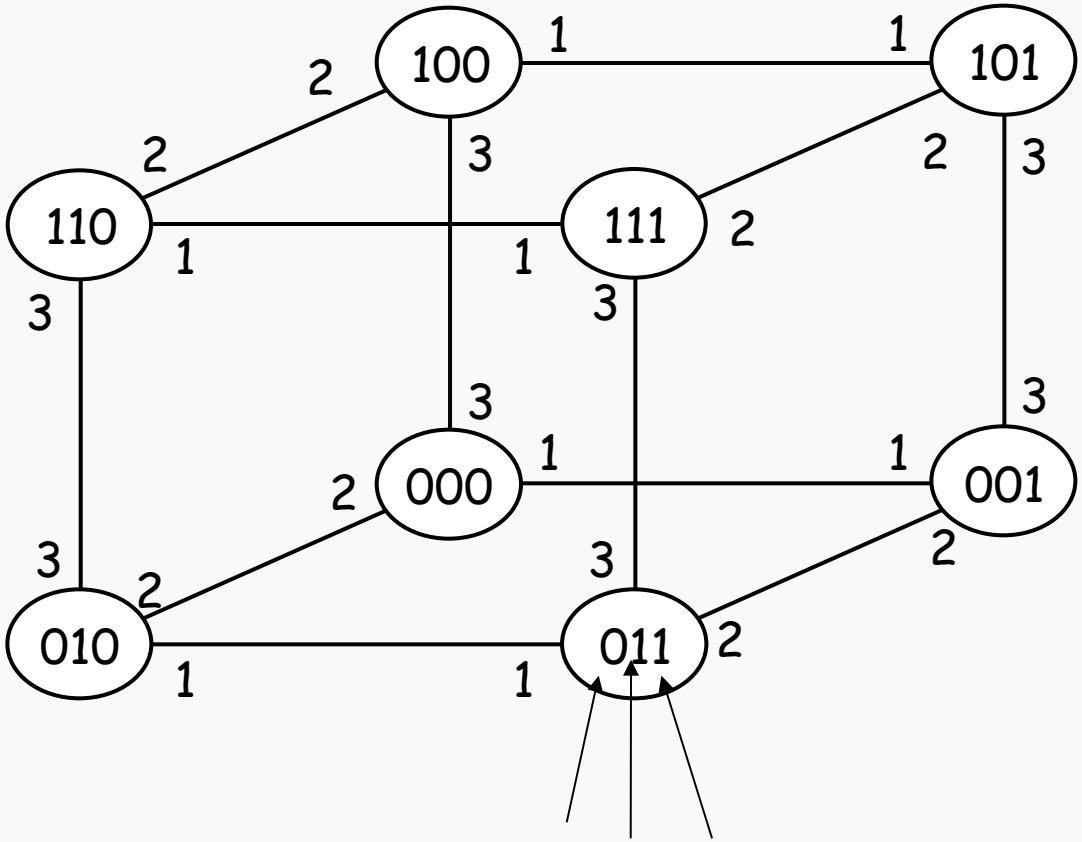
Quel est la complexité de FLOOD dans un graph complet ?
Comment peut-on faire mieux ?

Quel est la complexité de FLOOD dans un arbre ?
Peut-on faire mieux ?

Example: Labelled Hypercube



Chaque lien entre deux noeuds est étiqueté par la position du bit qui diffère entre les identificateurs des deux noeuds.

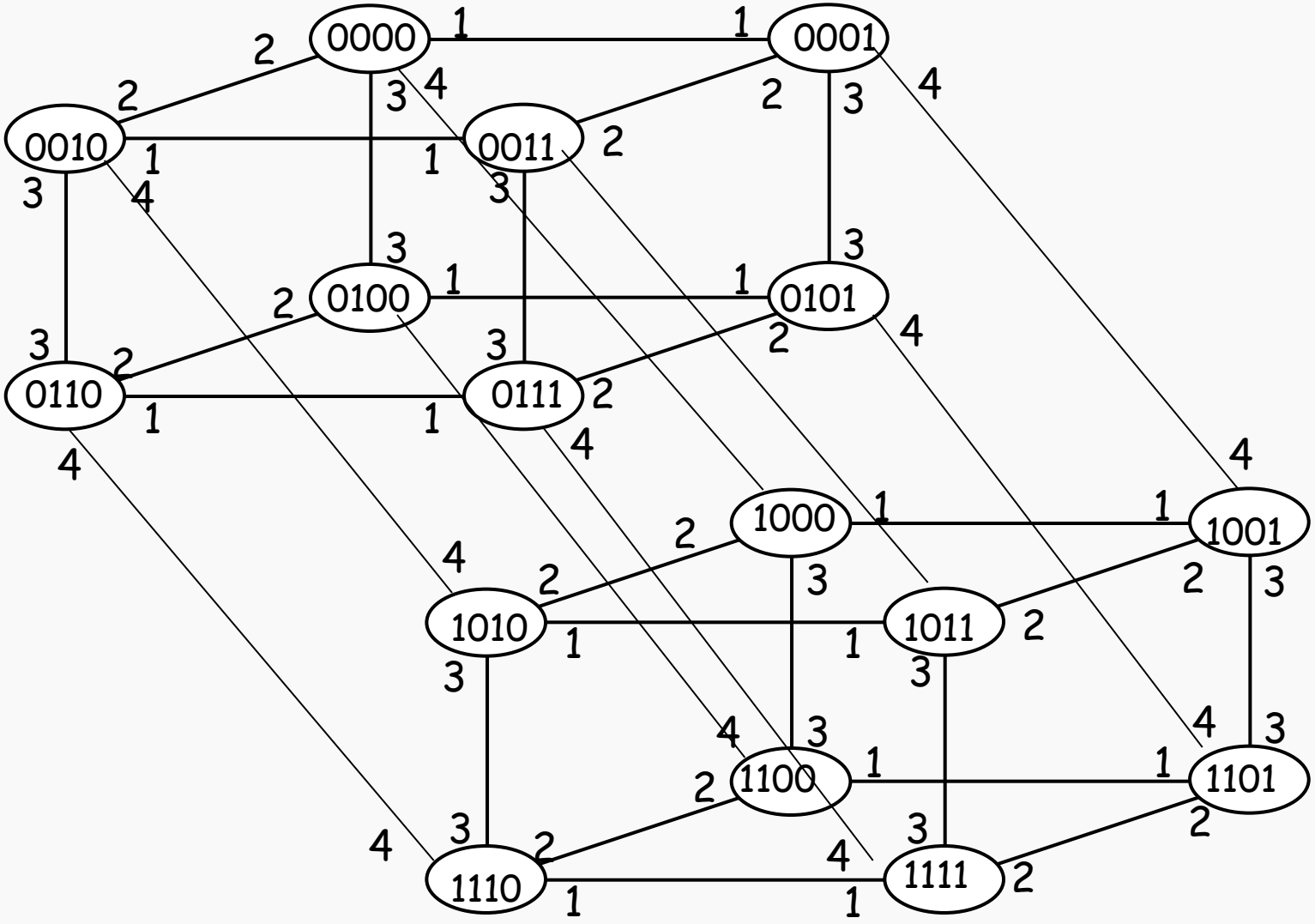


$X = x_k x_{k-1} \ldots x_1 x_0$

Nom de K-bit

$x_2 x_1 x_0$

premier bit


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

Un hypercube de dimension k a $n = 2^k$ noeuds

chaque noeud a k liens

➔ $m = n\,k/2 = O(n \log n)$

FLOOD aurait une complexité de $O(n \log n)$

HyperFlood – Broadcast efficace

1) L'initiateur envoie le message à tout ses voisins
2) Un noeud recevant le message du lien l,
   l'envoie seulement sur les liens l' < l

Exactitude

Tous les noeuds reçoivent le message

Découle du lemme suivant:

Pour chaque paire de noeuds x et y il existe un unique chemin passant par des liens de valeur décroissante

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

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

Exactitude

Tous les noeuds reçoivent le message

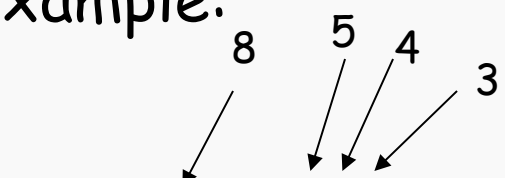
Découle du lemme suivant:

> Pour chaque paire de noeuds x et y il existe un unique chemin passant par des liens de valeur décroissante

$X = x_k, x_{k-1} \ldots x_1, x_0$ $\quad$ $Y = y_k, y_{k-1} \ldots y_1, y_0$

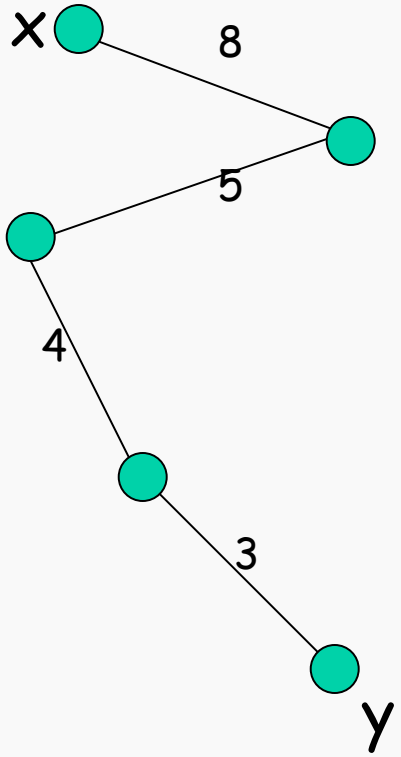
Considérez les positions où ils diffèrent en ordre décroissant ...

Example:



X= 100010100

Y= 110001000



x

100010100

110010100

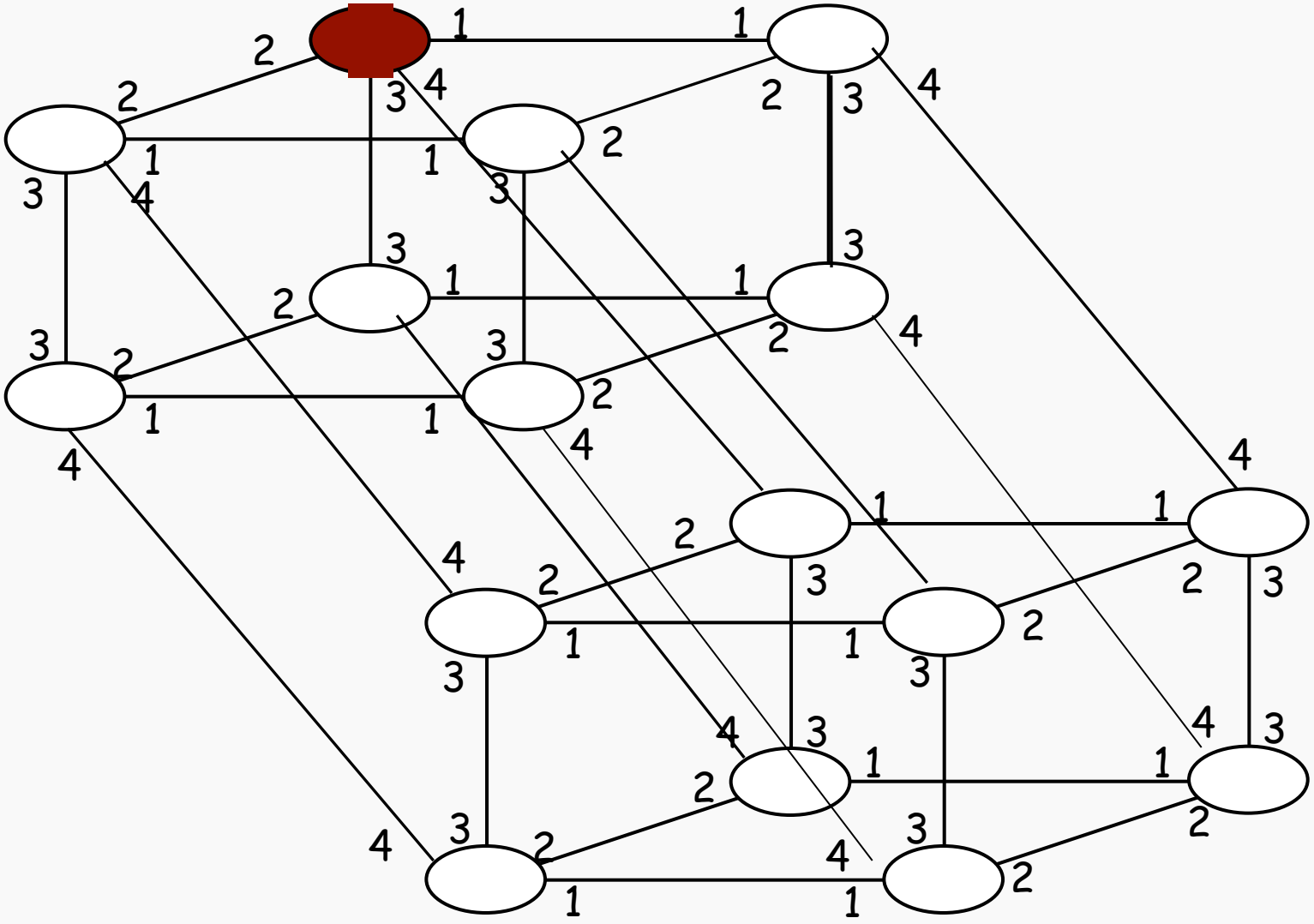
100000100

100001100

100001000 y

---> Les messages forment un arbre couvrant….

Et chaque noeud reçoit le message



Complexité: n-1 (OPTIMAL)

Parce que chaque entité reçoit
l'information une fois seulement

Pour des topologies particulières

Flooding générique: 2m - (n-1)

Algorithme adapté aux hypercube: (n-1)

Algorithme adapté aux graphs complet: (n-1)

Dans les arbres, flooding est optimal: (n-1)

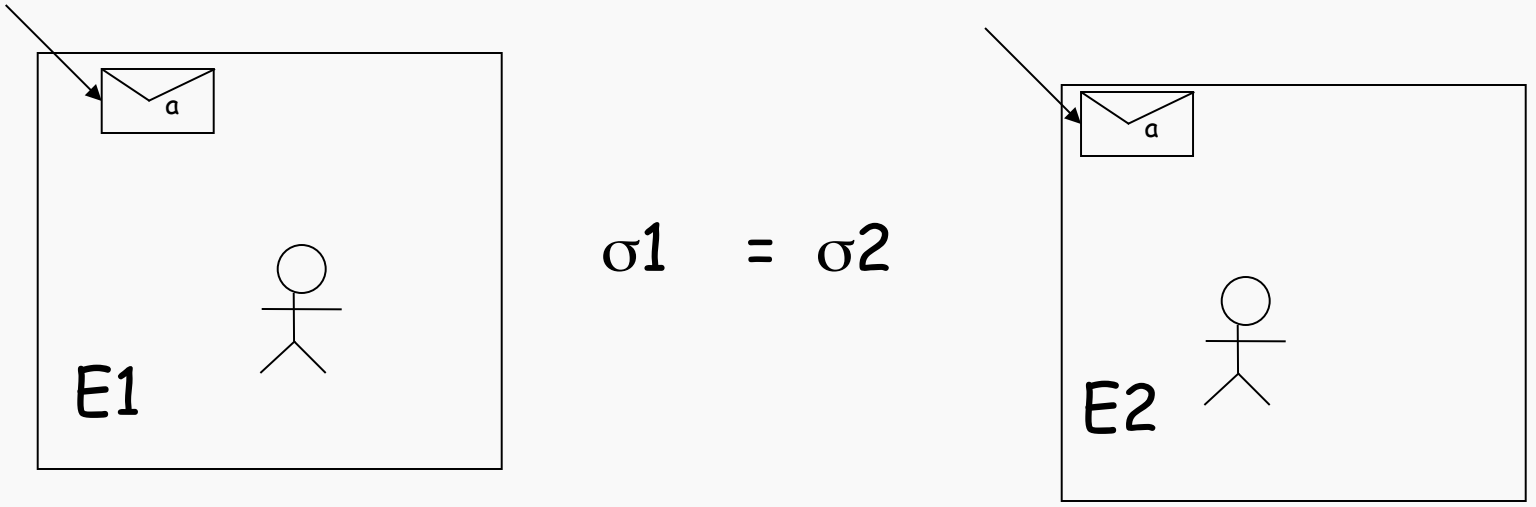
Faits Importants

État x Évènement ---> Action

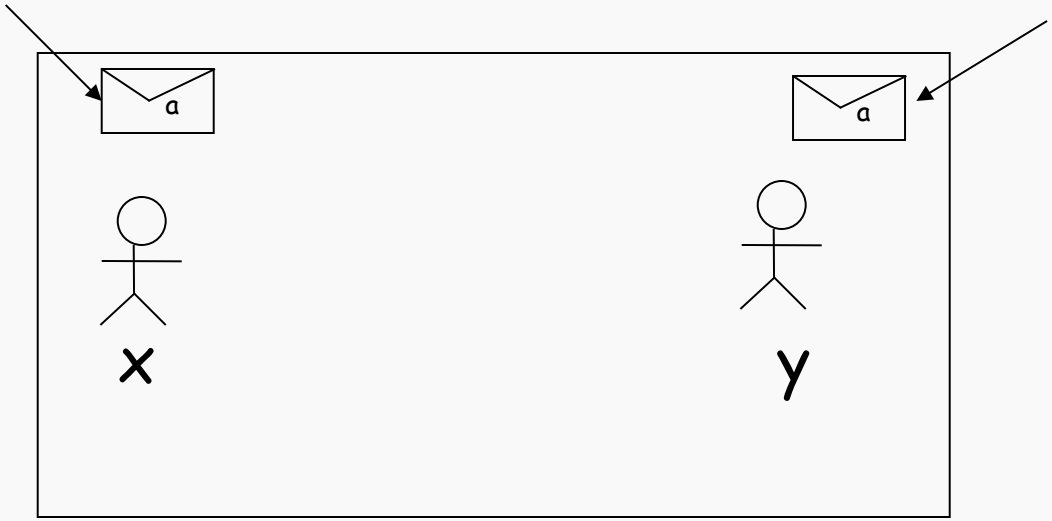
$\sigma(x,t)$ = état interne d'un entité x

Contenu de la mémoire (registres, horloge, …) au temps t

E1, E2: environments différents



1) Si le même évènement a lieux à x au temps t dans deux différentes exécutions et si les états internes $\sigma 1$ et $\sigma 2$ de x sont identiques dans les deux exécutions au temps t, alors **le nouvel état interne de x sera le même dans les deux exécutions.**



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

2) Si le même évènement a lieu à x et y au temps t lors de la même exécution, et si les état internes $\sigma(x)$ and $\sigma(y)$ sont égaux, alors **les nouveaux états internes de x et y seront égaux.**

Un example

De retour à Broadcast …

Théorème: considérant l'ensemble d'hypothèses suivant:

Initiateur unique
G est connexe
Absence d'erreurs
Liens bidirectionels

Tout algorithme Broadcast générique requiert, dans le pire cas, m messages.

Limite inférieure pour Broadcast

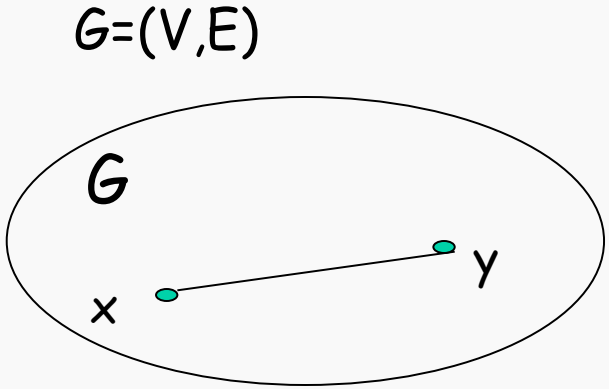
Preuve.

m(G) = nombre de liens dans G

Par contradiction.
Soit A un algorithmee qui exécute un Broadcast en échangeant moins de m(G) messages (peut importe l'exécution et le graph choisi) sous les conditions spécifiés.

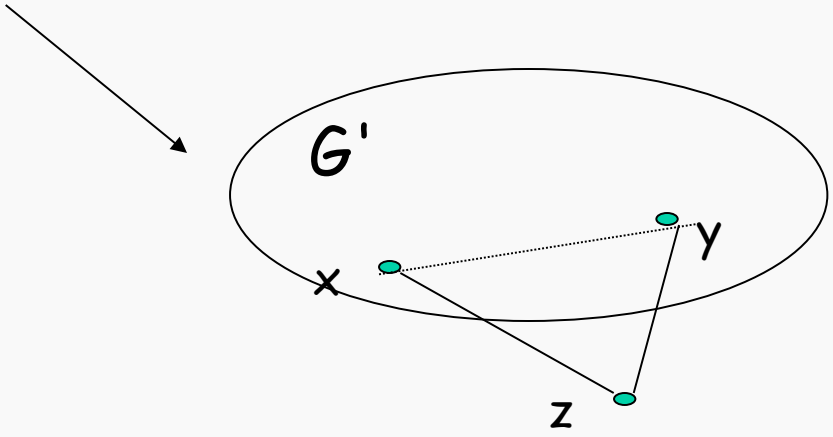
Alors il y a au moins un lien dans G qu'aucun message n'a parcouru

Soit e = (x,y) un tel lien



Construisons un nouveau graph G'

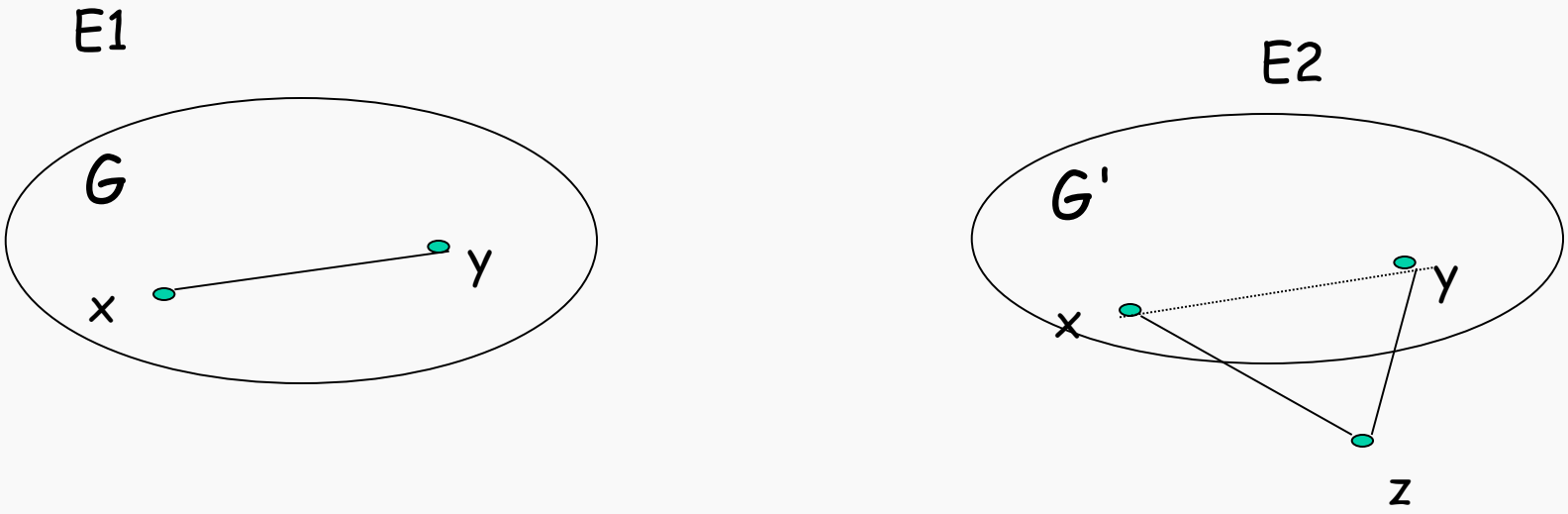
(rappel: n est inconnu)



G' =(V∪ {z}, E-e ∪ {(x,z),(y,z)}

Exécutons le même algoritme sur G' avec les même délais et mêmes états initiaux pour tous les noeuds sauf z, qui est `sleeping`

Deux exécutions dans deux environnements

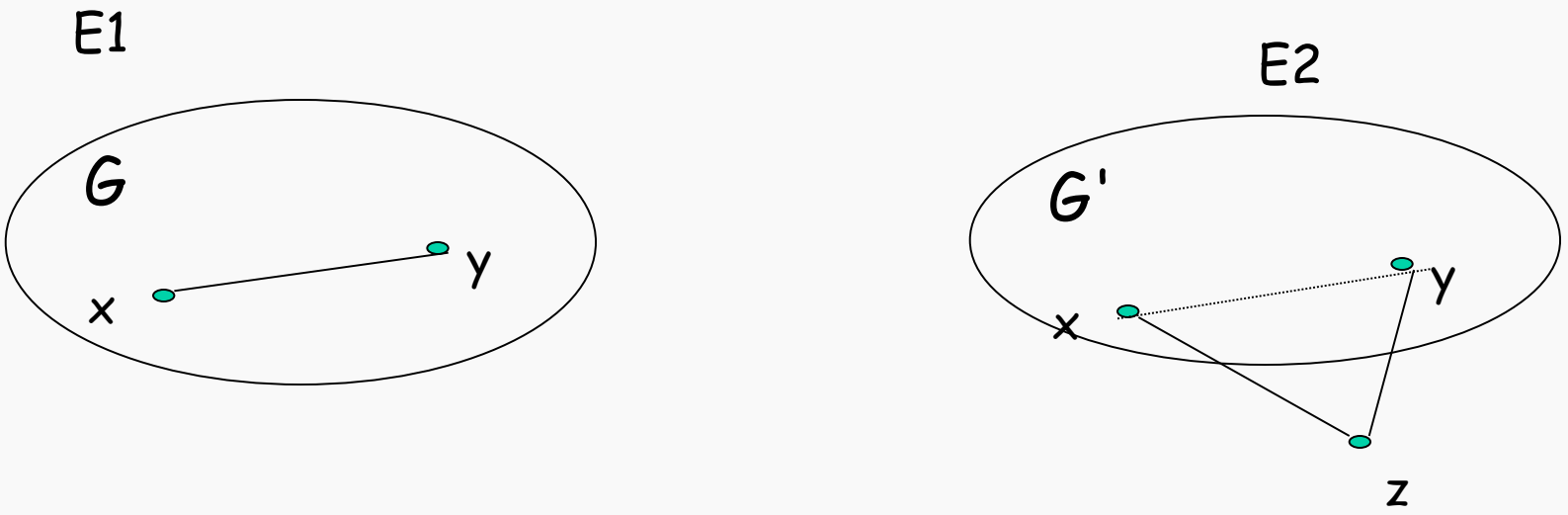


Pour tous les noeuds à l'exception de z, les deux exécutions sont **identiques**

x et y ne s'envoient pas de message en E1

--->

x et y n'envoient pas de message à z en E2



L'Algorithme termine en temps fini

Mais dans E2, le noeud z n'est jamais atteint.

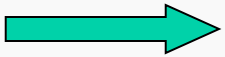
Observations:

1) Réseau dense = plus de messages
(ex. réseau complet = n (n-1) …)
2) Optimal lorsque le graphe est acyclique

Idée: pour résoudre broadcast:

1. construire un arbre couvrant de G

2. Exécuter flooding



Construction d'un arbre couvrant