

# Systèmes et algorithmes répartis

## Modèle standard et principes algorithmiques

Philippe Quéinnec, Gérard Padiou

Département Informatique et Mathématiques Appliquées  
ENSEEIH

26 septembre 2017



# plan

## 1 Le modèle standard

- Approche événementielle
- Causalité
- Abstraction d'un calcul
- Prise de cliché

## 2 Description des algorithmes

- Description du comportement des processus
- Exemple : l'élection



# Plan

## 1 Le modèle standard

- Approche événementielle
- Causalité
- Abstraction d'un calcul
- Prise de cliché

## 2 Description des algorithmes

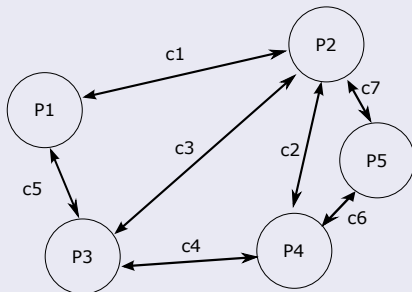
- Description du comportement des processus
- Exemple : l'élection



## Vision statique : Graphe de processus

### Description graphique

- Sommets  $\equiv$  processus / sites
- Arcs  $\equiv$  liaisons de communication / canaux



# Propriétés

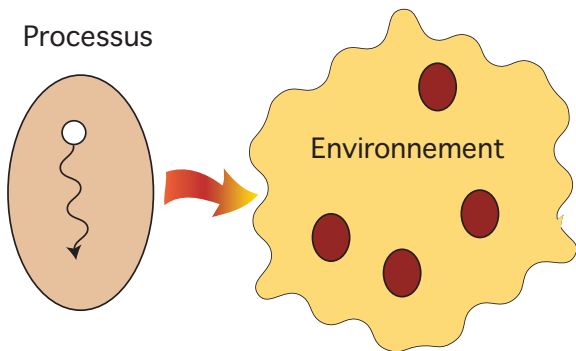
## Propriétés des processus / sites

- Un processus possède une identité unique
- Un processus possède un état rémanent
- Un processus exécute un code séquentiellement
- Un processus n'a qu'une connaissance partielle des autres
- Un processus peut communiquer avec un voisinage
- Pas de panne (par arrêt ou comportement byzantin)

## Propriétés du réseau

- Multiples paramètres : point à point ou diffusion, (a)synchrone, fiable, délais bornés, etc
- Messages : ni duplication ni erreur

## Connaissances d'un processus



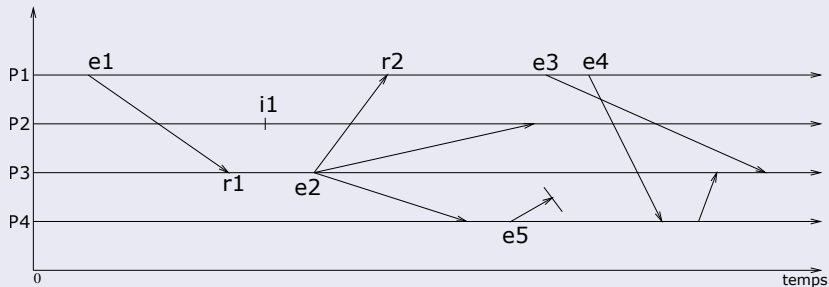
- Nombre de processus ?
- Voisinage de communication ?
- Structure du réseau : maillé, anneau, statique/dynamique, etc



## Vision dynamique : Chronogramme

### Représentation événementielle

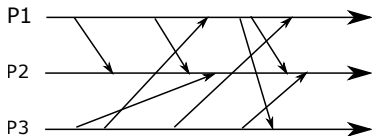
- 3 types d'événements : émission, réception, interne
- Mise en évidence de la causalité



# Système asynchrone vs synchrone

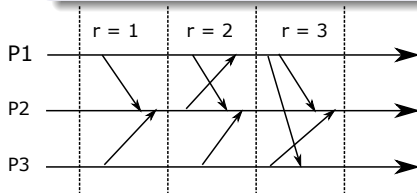
## Asynchrone

- Pas de temps externe
- Progression de chaque processus à son rythme
- Délai de transmission arbitraire



## Synchrone

- Existence de pas de calcul (*round*) globaux
- Un message émis dans un pas est reçu au pas suivant / dans le même pas (selon le modèle)





## Relation de causalité (Lamport 1978)

### Ordre partiel entre événements $\prec$

- Les événements d'un processus sont totalement ordonnés :  
 $e$  et  $e'$  sur le même site, et  $e$  précède  $e'$ , alors  $e \prec e'$ .
- L'émission d'un message précède causalement sa réception :  
Si  $e = \text{émission}(m)$  et  $e' = \text{réception}(m)$ , alors  $e \prec e'$ .
- Transitivité :  $\forall e, e', e'' : e \prec e' \prec e'' \Rightarrow e \prec e''$
- La relation  $\prec$  est un **ordre partiel** :  $e \parallel e' \stackrel{\Delta}{=} e \not\prec e' \wedge e' \not\prec e$
- Indépendance du temps physique mais consistant avec :  
 $e \prec e' \Rightarrow e$  est survenu avant  $e'$  dans le temps absolu

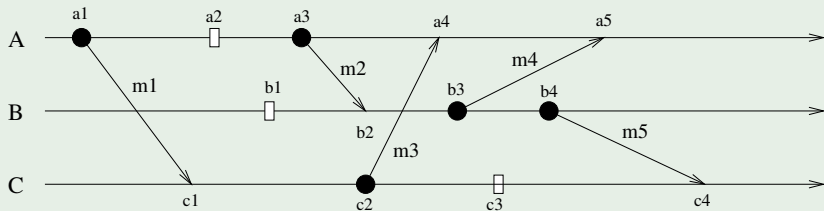
---

1. *Time, Clocks and the Ordering of Events in a Distributed System*, Leslie Lamport. Communications of the ACM, July 1978.



# Relation de causalité (Lamport 1978)

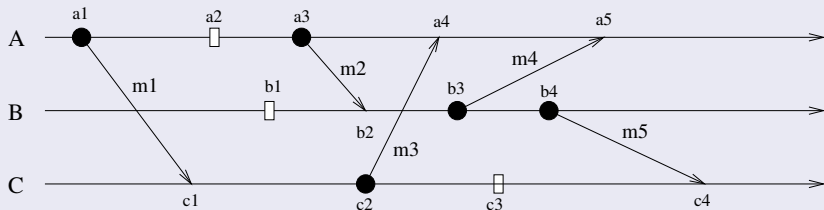
## Exemple



$$\begin{aligned} a_1 &\prec a_2 \prec a_3 \prec a_4 \prec \dots \\ a_1 &\prec c_1, c_2 \prec a_4, b_4 \prec c_4 \\ a_1 &\prec c_3 \text{ (car } a_1 \prec c_1 \prec c_2 \prec c_3) \\ a_2 &\prec c_4 \\ a_3 &\parallel c_2 \end{aligned}$$

# Abstraction d'un calcul réparti

## Exécutions causalement équivalentes



- Ensemble d'événements + relation causale  
→ ensemble d'exécutions réelles équivalentes

$$a_1; b_1; c_1; a_2; \dots \equiv a_1; a_2; c_1; b_1; \dots$$

$$a_1; c_1; a_2; \dots \not\equiv c_1; a_1; a_2; \dots \text{ car } a_1 \prec c_1$$

- Le choix des événements fixe un niveau d'observation

# Passé / futur causal

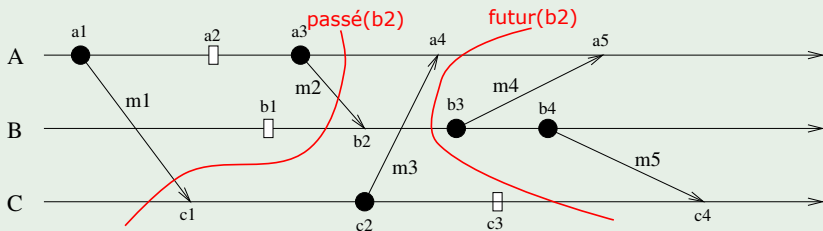
## Partition des événements

$$\text{passé}(e) \triangleq \{f \mid f \prec e\}$$

$$\text{futur}(e) \triangleq \{f \mid e \prec f\}$$

$$\text{concurrence}(e) \triangleq \{f \mid f \notin \text{passé}(e) \wedge f \notin \text{futur}(e)\}$$

## Exemple



## Coupure et coupure cohérente

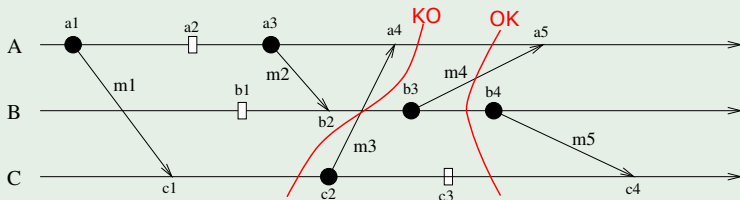
### Coupure

Une **coupure** est un ensemble d'événements qui forment des préfixes complets des histoires locales.

### Coupure cohérente

Une coupure  $C$  est **cohérente** si  $\forall e \in C : \forall e' : e' \prec e \Rightarrow e' \in C$

### Exemple

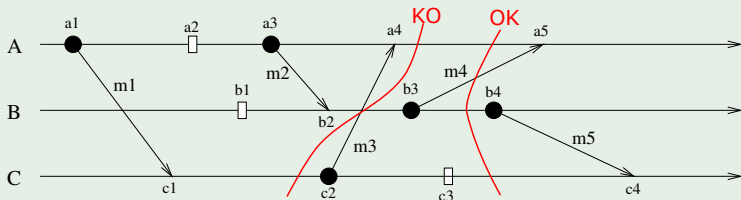


## Passé et coupure cohérente

Une coupure  $C$  est cohérente ssi  $C = \bigcup_{e \in C} (\text{passe}(e) \cup \{e\})$  :

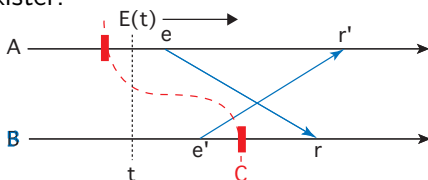
- Pas de « trou » sur un site
- Une réception n'est pas présente sans son émission

### Exemple



## Coupure cohérente et état global

Une coupure cohérente correspond à un état global qui **aurait pu** exister.

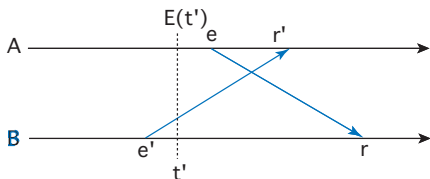


Réalité

La coupure est cohérente **mais...**

État effectif

L'état n'a pas existé à un instant global



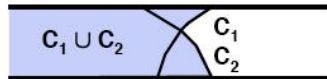
## Treillis des coupures (cohérentes)

### Treillis des coupures

L'ensemble des coupures forme un treillis pour l'inclusion : si  $C_1$  et  $C_2$  sont deux coupures, alors  $C_1 \cup C_2$  et  $C_1 \cap C_2$  sont des coupures

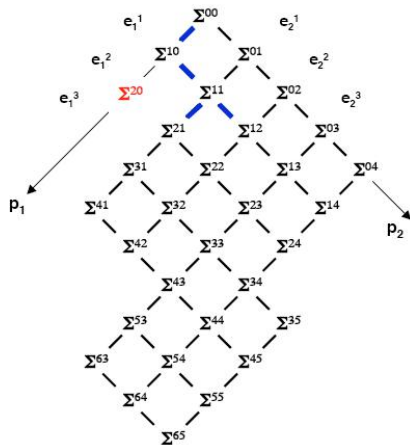
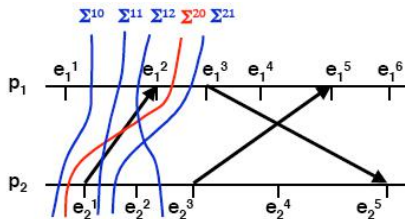
### Treillis des coupures cohérentes

L'ensemble des coupures cohérentes forme un treillis pour l'inclusion : si  $C_1$  et  $C_2$  sont deux coupures cohérentes, alors  $C_1 \cup C_2$  et  $C_1 \cap C_2$  sont des coupures cohérentes





## Treillis des coupures cohérentes



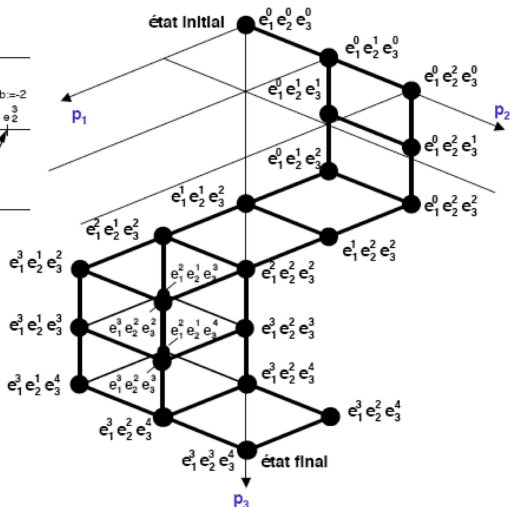
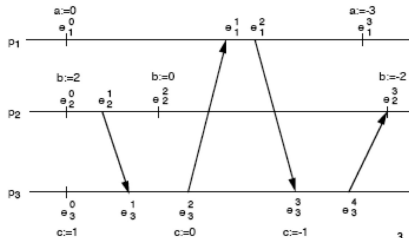
- Arc du treillis = occurrence d'un événement possible
- Une exécution = suite d'états globaux cohérents = chemin dans le treillis

Explosion du nombre d'exécutions causalement équivalentes

(dessins : cours S. Krakowiak)

# Treillis des coupures cohérentes

## Autre exemple



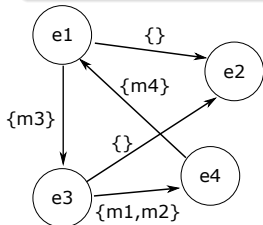
(dessin : cours S. Krakowiak)



## Prise de cliché (snapshot)

### Définition

**Objectif** : Capter, centraliser un état global **passé** des processus



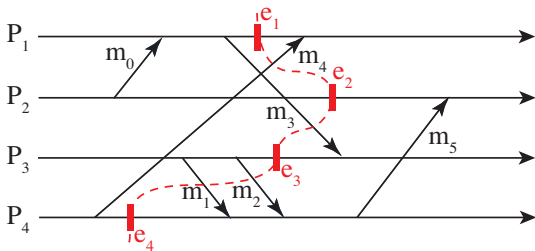
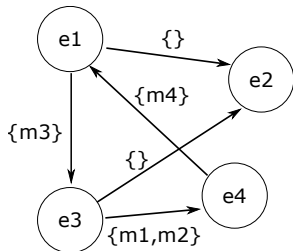
- Prise **instantanée** impossible
- Un site collecteur accumule
- Prise **cohérente** de clichés locaux
- Problème des messages en transit

### Cliché global instantané

Clichés locaux + Messages en transit  
 $\{e_1, e_2, e_3, e_4\} + \{m_1, m_2, m_3, m_4\}$

# Prise de cliché (snapshot)

Schéma temporel de la prise de cliché



## Algorithme de Chandy-Lamport (1985)

- Un système existant échange des messages ;
- On **superpose** des échanges de messages dédiés pour déclencher des actions locales de sauvegarde de l'état d'un site (= un cliché local + des messages reçus) ;
- Ces états sauvegardés sont **collectés** pour construire un cliché global.

---

1. *Distributed Snapshots : Determining Global States of Distributed Systems*,  
K. Mani Chandy and Leslie Lamport. ACM Transactions on Computer Systems, Feb. 1985



## Algorithme de Chandy-Lamport (1985)

### Idée

- Matérialiser la coupe par la circulation d'un **marqueur** visitant les sites
- Les messages émis **après** le passage du marqueur ne peuvent pas être dans la coupe
- Les messages émis par  $S_j$  **avant** le passage du marqueur sur  $S_j$ , et reçus par  $S_i$  **après** le passage du marqueur sur  $S_i$ , sont considérés comme les messages en transit de  $S_j$  vers  $S_i$

### Hypothèse

Canaux FIFO (pas de perte, pas de dépassement)



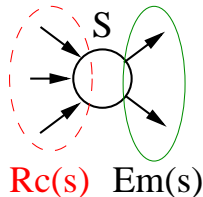
## Algorithme de Chandy-Lamport (1985)

### Hypothèses

- Réseau fortement connexe ( $\forall s, s' : \exists s \rightarrow^* s'$ )
- Canaux unidirectionnels et **fifo** :  
 $\forall s, Rc(s)$  : canaux en réception  
 $Em(s)$  : canaux en émission

### Principes de l'algorithme

- Utilisation de messages **marqueurs**
- Répartition de l'évaluation : chaque site  $s$  évalue :
  - **son** cliché local ;
  - les messages considérés en transit sur ses canaux en réception  $Rc(s)$



# Algorithme de Chandy-Lamport

## Comportement d'un site

Sur réception d'un **PREMIER marqueur** ou **spontanément** :

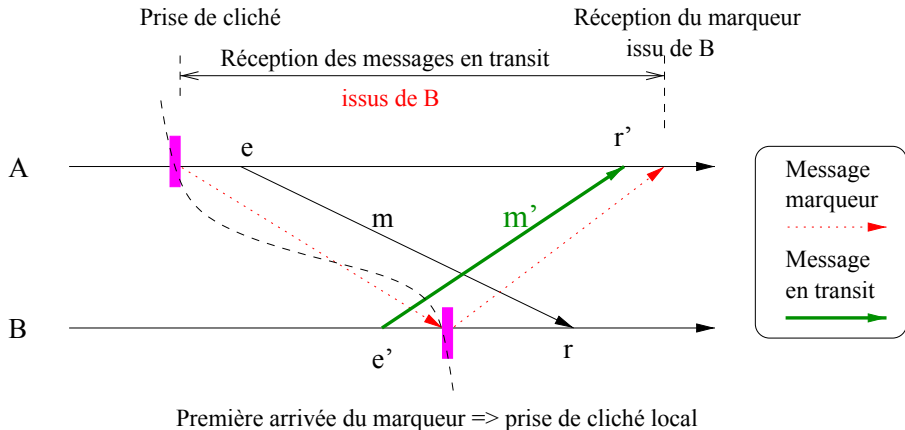
- 1 Prendre **son** cliché local  $L_s$  et émettre un **marqueur** sur chaque canal d'émission  $c \in Em(s)$
- 2 Enregistrer dans une liste  $enTransit[c]$  les messages reçus sur chaque canal de réception  $c \in Rc(s)$  jusqu'à la réception d'un **marqueur** sur ce canal
- 3 Lorsqu'un **marqueur** a été reçu sur **TOUS** les canaux de réception, communiquer au collecteur cet état partiel :

$$\langle L_s, \{enTransit[c] \mid c \in Rc(s)\} \rangle$$





## Prises de clichés locaux et marqueurs

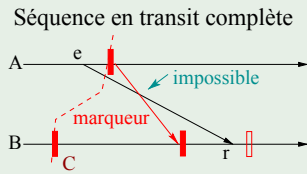
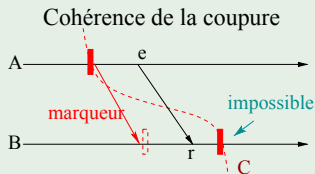


## Vérifier la correction...

### Propriétés

- Sûreté
  - Coupure cohérente
  - Collecte complète des messages en transit
- Vivacité
  - Tout site finit par prendre un cliché local
  - Un marqueur finit par arriver sur chaque canal de réception

### Exemple



## État enregistré = état *possible*

### État enregistré

- $\Sigma_{enreg}$  = cliché enregistré
- $\Sigma_{init}$  = coupure cohérente contenant l'événement déclencheur du cliché
- $\Sigma_{final}$  = coupure cohérente dans lequel le protocole de prise de cliché est terminé

Alors  $\Sigma_{init} \prec \Sigma_{enreg} \prec \Sigma_{final}$

(il existe un chemin de  $\Sigma_{init}$  à  $\Sigma_{final}$  passant par  $\Sigma_{enreg}$  dans le treillis des coupures cohérentes)

Exemple : sur le treillis page 17, si  $\Sigma_{init} = \Sigma^{11}$  et  $\Sigma_{final} = \Sigma^{32}$ ,  $\Sigma_{enreg}$  peut être  $\Sigma^{11}$ ,  $\Sigma^{21}$ ,  $\Sigma^{12}$ ,  $\Sigma^{31}$ ,  $\Sigma^{22}$  ou  $\Sigma^{32}$ , et a pu ne pas être traversé dans la réalité.

## Utilisation du cliché : propriété stable

### Prédicat stable

Un prédicat  $P$  sur un état global  $E$  d'un système est stable ssi  
 $\forall E' : E \prec E' \wedge P(E) \Rightarrow P(E')$

(exemples : le calcul est terminé, il y a eu 10 messages reçus...)

### Vérification de $P$

Si  $P$  est un prédicat stable alors :

- $P(\Sigma_{enreg}) \Rightarrow P(\Sigma_{final})$  (et tout état ultérieur)
- $\neg P(\Sigma_{enreg}) \Rightarrow \neg P(\Sigma_{init})$  (et tout état antérieur)

## Utilisation du cliché : propriété possible/certaine

### Prédicat possible/certain

Pour un prédicat  $P$  :

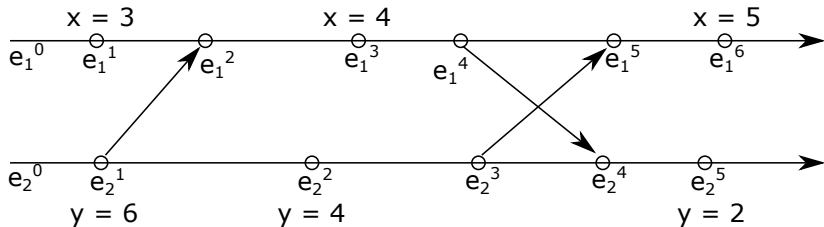
- $Pos(P)$  (possibly  $P$ ) : il existe une observation cohérente (= un chemin dans le treillis) qui passe par un état où  $P$  est vrai.
- $Def(P)$  (definitely  $P$ ) : toutes les observations cohérentes (= tous les chemins) passent par un état où  $P$  est vrai.

### Vérification

- $P(\Sigma_{enreg}) \Rightarrow Pos(P)$  mais pas l'inverse...
- $\neg Pos(P) \Rightarrow Def(\neg P)$  mais pas l'inverse...



## Exemple de vérification de propriétés

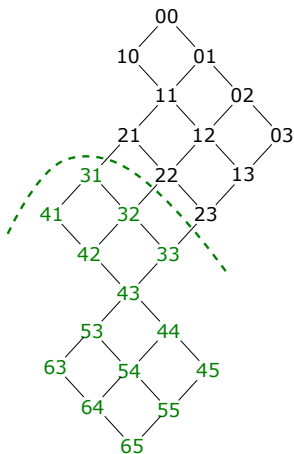


- $x \geq 4$ ? (en supposant  $x$  croissant  $\Rightarrow$  propriété stable)
- $Pos(x = y - 2)$ ?
- $Def(x = y)$ ?

(d'après Lorenzo Alvisi)

# Exemple de vérification

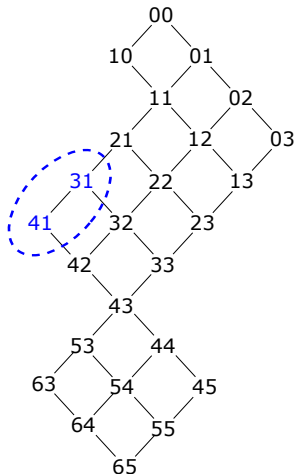
Propriété stable



- $x \geq 4$ ?  
(sous l'hypothèse  $x$  croissant)
- N'importe quel cliché  $\Sigma_{enreg}$  obtenu après  $\Sigma^{31}$  permet de le vérifier

# Exemple de vérification

## Possibilité

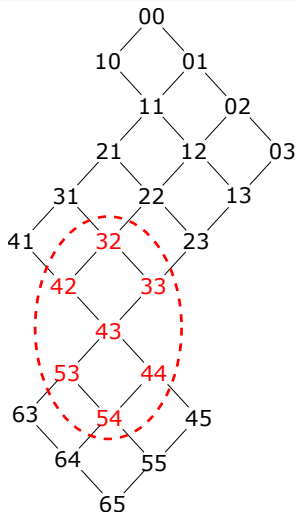


- $Pos(x = y - 2)$
- $x = y - 2$  est vrai dans les états cohérents  $\Sigma^{31}$  et  $\Sigma^{41}$
- Détecté uniquement si  $\Sigma_{enreg} \in \{\Sigma^{31}, \Sigma^{41}\}$
- $\Sigma_{enreg}$  pas nécessairement survenu dans la réalité



# Exemple de vérification

Certitude



- $Def(x = y)$  ?
- Vrai
- $Pos(x = y)$  pas détecté si on capture un état antérieur à  $\Sigma^{32}$  ou postérieur à  $\Sigma^{54}$
- La capture d'un état (p.e.  $\Sigma^{42}$ ) ne permet pas de conclure

# Utilisation du cliché : propriété possible/certaine

## Principe de la vérification

- Un processus moniteur  $M$  collecte tous les états locaux
- $M$  construit le treillis des coupures cohérentes (à partir d'un codage complet de la relation de causalité, cf chapitre suivant)
- Pour évaluer  $Pos(P)$  : parcourir le treillis depuis l'état initial, niveau par niveau, et s'arrêter au premier état où  $P$  est vrai. Aucun état  $\Rightarrow \neg Pos(P)$ .
- Pour évaluer  $Def(P)$  : parcourir le treillis depuis l'état initial, niveau par niveau, en ne développant que les états vérifiant  $\neg P$ . Si plus d'état, alors  $Def(P)$ ; si état final atteint (et  $\neg P$  dans cet état) alors  $\neg Def(P)$ .
- Explosion combinatoire : pour  $N$  sites ayant chacun au plus  $m$  états, possiblement  $m^N$  coupures cohérentes.



# Plan

## 1 Le modèle standard

- Approche événementielle
- Causalité
- Abstraction d'un calcul
- Prise de cliché

## 2 Description des algorithmes

- Description du comportement des processus
- Exemple : l'élection



# Principes algorithmiques

- Algorithmes symétriques
  - code répliqué,
  - données initiales propres : identité, voisinage de communication.
- Structurer les échanges de messages :
  - Utilisation de réseaux en anneau
  - Utilisation de la structure d'arbre
- Étudier des problèmes génériques :
  - Les services : datation, exclusion mutuelle, consensus, élection. . .
  - Les observations de propriétés stables : terminaison, interblocage
  - La tolérance aux fautes : réplication, atomicité



## Description des algorithmes

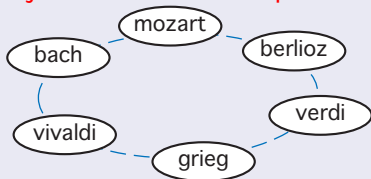
```
Process P(id : 0..N-1)
  <variables locales>
  on <condition-logique> :
    <Action>
  on reception message(arg) [ from P(j) ] :
    <Action>
  on <condition-logique> ^ reception message(arg) :
    <Action>
  on start : // initialement
    <Action>
  ...
```

- Action : modification des variables locales et/ou envoi(s) de message, ou terminaison (**terminate**)
- Envoi : **send** Msg(<args>) **to** <destinataire(s)>
- Choix d'un événement à traiter : non déterministe parmi ceux ayant la garde vraie et un message à consommer

## Exemple : l'élection

### Le problème de l'élection

Objectif : Élire un seul processus



- Un processus a une identité unique qu'il connaît
- Un processus ne connaît pas le nombre global de processus
- Un processus ne connaît pas l'identité des autres
- Communication sur un anneau logique

1. *An improved algorithm for decentralized extrema-finding in circular configurations of processes*, Ernest Chang and Rosemary Roberts. Communications of the ACM, May 1979.

## Solution correcte ou fausse ?

On suppose que les processus sont totalement ordonnés (ici par leur indice, en pratique, par leur adresse IP par exemple)

```
Process P(id : 0..N-1)
  //  $\ominus$  et  $\oplus$  : opérateurs modulo  $N$ 
  type Etat = {candidat, élu};
  Etat étatCourant  $\leftarrow$  candidat;
  on reception Candidat(proc) from P[id $\ominus$ 1] :
    if (proc < id) send Candidat(proc) to P[id $\oplus$ 1];
    else if (proc = id) étatCourant  $\leftarrow$  élu;
    else nop; // ignorer le message
  on (étatCourant = élu) :
    terminate;
```

Pourquoi cela ne marche-t-il pas ?



## Solution qui conduit à l'élection du plus petit

```
Process P(id : 0..N-1)
  type Etat = {candidat, élu};
  Etat étatCourant ← candidat;
  on start:
    send Candidat(id) to P[id⊕1]; // chacun candidate
  on reception Candidat(proc) from P[id⊖1]:
    if (proc < id) send Candidat(proc) to P[id⊕1];
    else if (proc = id) étatCourant ← élu;
    else nop; // ignorer le message
  on (étatCourant = élu) :
    terminate;
```

Pas parfait : un seul processus se termine





## Solution plus complète : tous les processus terminent

```
Process P(id :0..N-1) {
  type Etat = {candidat,élu,perdant};
  Etat étatCourant ← candidat;
  on start :
    send Candidat(id) to P[id⊕1]; // chacun candidate
  on reception Candidat(proc) from P[id⊖1]:
    if (proc < id) send Candidat(proc) to P[id⊕1];
    else if (proc = id) étatCourant ← élu;
    else nop; // ignorer le message
  on (étatCourant = élu) :
    send Elu(id) to P[id⊕1];
  on reception Elu(proc) from P[id⊖1]:
    if (proc ≠ id) then
      étatCourant ← perdant;
      send Elu(proc) to P[id⊕1];
  endif
  terminate
```

## Déclenchement spontané individuel

Pas nécessairement tous candidats au départ (mais tous éligibles)

```
Process P(id : 0..N-1)
```

```
  type Etat = {candidat, élu, perdant};
```

```
  Etat étatCourant ← candidat;
```

```
  on random() :
```

```
    send Candidat(id) to P[id⊕1];
```

```
  on reception Candidat(proc) from P[id⊖1]:
```

```
    if (proc < id) send Candidat(proc) to P[id⊕1];
```

```
    else if (proc = id) étatCourant ← élu;
```

```
    else if (proc > id) send Candidat(id) to P[id⊕1];
```

```
  :
```



## Conclusion

- Modélisation par des **événements** locaux
- Relation entre ces événements, en particulier la **causalité**
- Représentation avec des chronogrammes
- Notion d'**état global**, de **coupure**
- Calcul d'un état global

