# Algorithmique Répartie

# L'Observation

Université François Rabelais de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Master 1 Informatique

Option Systèmes d'Information et Aide à la Décision

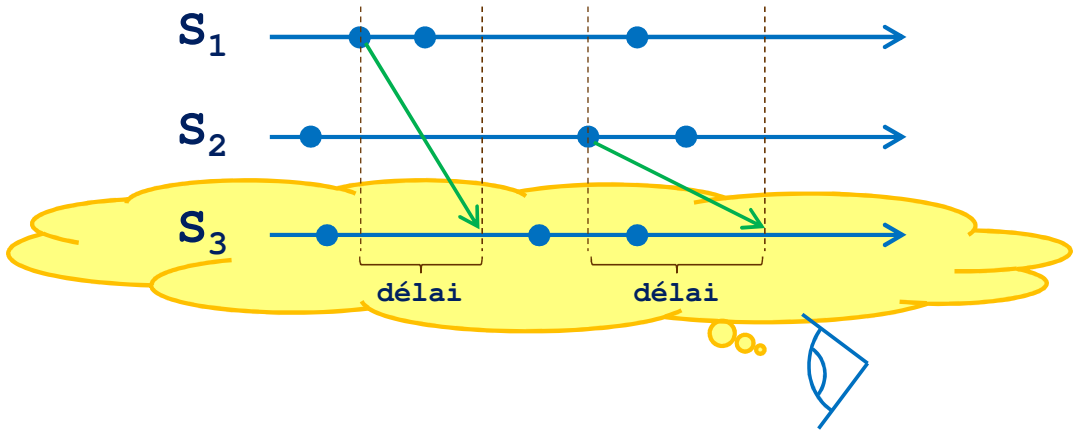
Mohamed Taghelit
taghelit@univ-tours.fr

# L'Observation

## Problématique

- Dans une application répartie, il est souvent important de savoir si une propriété est vraie ou pas :
  - la propriété s'exprime comme un prédicat de l'état global,
  - les différents sites ne partagent pas de mémoire commune et communiquent uniquement par échange de messages,

    à un instant réel donné, l'état global du système est donc déterminé par l'état local de chaque processus ainsi que par l'ensemble des messages en transit,
  - impossibilité d'observer plus d'un processus à la fois à cause des délais de propagation des messages,




A cause des délais de propagation des messages :

- un observateur extérieur au système a une vision décalée dans le temps de l'état local de tous les site,
- un observateur inclus dans le système n'a une vision exacte que de l'état du site sur lequel il se trouve.

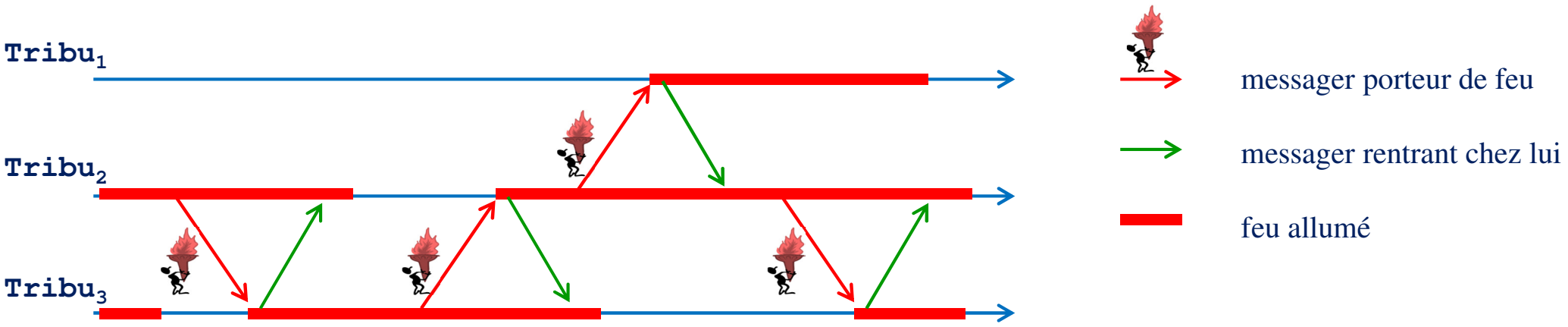
  - le décalage dans le temps n'est pas un problème si l'on est sûr que l'information collectée auprès de chacun des sites est cohérente.

# L'Observation

## Exemple du petit primitif (F. Mattern)

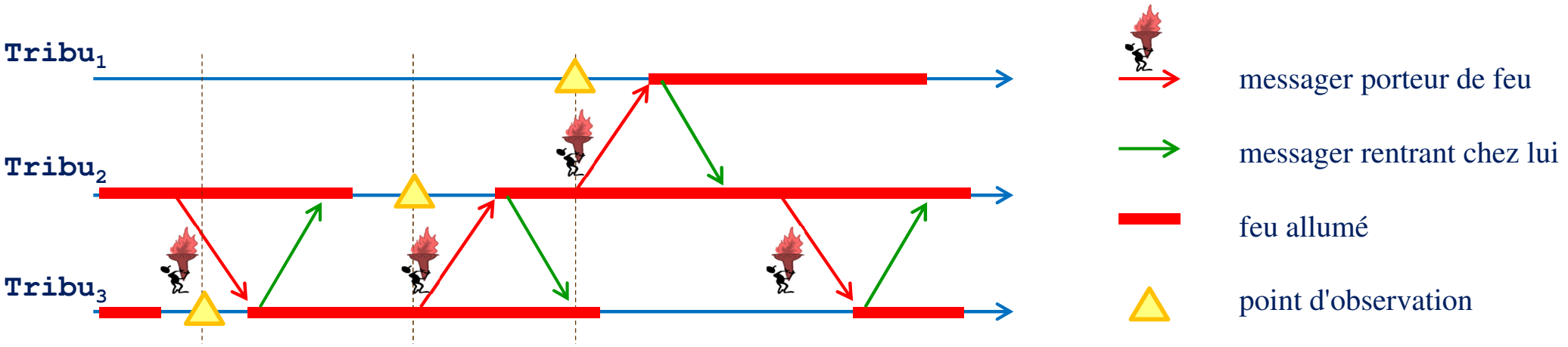
Les petits primitifs vivent dans des tribus dispersées géographiquement. Chaque tribu entretient un feu, mais parfois, lorsqu'un primitif manque de vigilance, le feu de sa tribu s'éteint. Malheureusement, les primitifs ne savent pas allumer un feu ! Donc, s'ils ne veulent pas attendre le prochain orage pour manger chaud à nouveau, il leur faut récupérer du feu auprès de leurs voisins. Comme ils n'ont pas la possibilité de savoir s'il y a un feu allumé ailleurs et qu'ils sont plus paresseux que frileux, ils attendent de recevoir la visite d'un messager. Ce dernier est systématiquement porteur d'une torche qui lui permet de rallumer le feu ci celui-ci est éteint.

Une exécution possible de ce système est représentée ci-dessous :



# L'Observation

Supposons qu'il existe un observateur qui visite les différentes tribus comme indiqué ci-dessous :




- à cause des délais nécessaires à ses voyages, il ne peut pas les voir (les tribus) simultanément,
- les conclusions de sa mission seront ici :

**le feu est définitivement éteint** (enfin, jusqu'au prochain orage).

En effet, chaque fois qu'il visite une tribu, il peut faire l'observation suivante :

- localement, le feu est éteint,
- de plus, aucun messager porteur d'une torche n'est en route.

- pourtant, si l'on observe le système, on voit qu'à tout instant, il y a au moins un feu allumé.
- sa conclusion est fausse car sa vision de l'état du système n'est pas cohérente avec la réalité.

# L'Observation

Il n'est pas possible d'observer simultanément tous les processus

- nécessité d'analyser les propriétés du système,
- produire des algorithmes qui garantissent à un observateur qu'à défaut de connaître l'état global, il peut obtenir une vision cohérente de l'état du système.

Intérêts à la construction d'un état global cohérent :

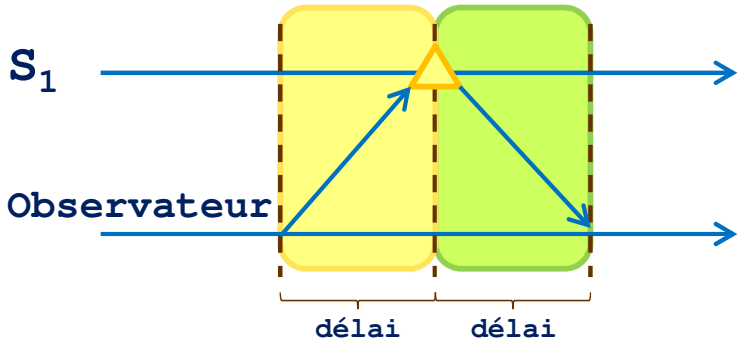
- le déverminage d'une application répartie : l'analyse après coup (off-line) des états d'une exécution erronée peut aider à comprendre pourquoi l'application ne se comporte pas comme prévu,
- la construction de points de reprise : si le système doit être redémarré après panne, on peut le reprendre à partir d'un état global cohérent calculé plutôt que l'état initial,
- la détection de propriétés stables : il s'agit de propriétés qui, lorsqu'elles deviennent vraies dans un état, restent vraies pour tous les états suivants de l'exécution. Ici, si une propriété stable devient vraie dans un état global cohérent, on veut pouvoir affirmer qu'elle reste vraie pour tous les états suivants.

# Observations Équivalentes

Au niveau de l'observation d'une application répartie, les délais de transmission des messages interviennent à deux niveaux :

- l'instant où le processus est observé (délai entre la demande d'observation et la réception de cette demande sur un site),

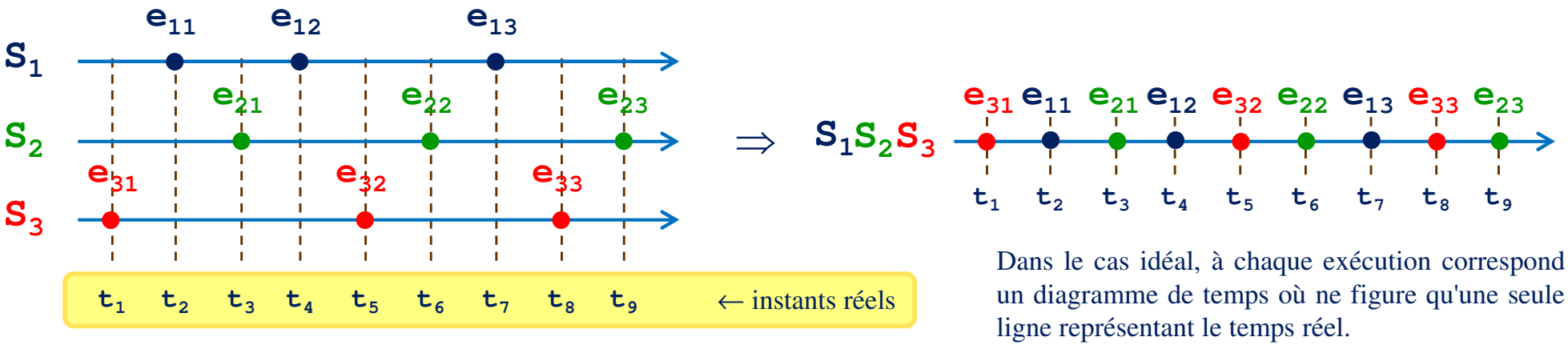
- l'instant où l'observateur reçoit l'information (délai entre l'observation et la réception de l'état local).



# Observations Équivalentes

Cas idéal :

- l'instant réel d'occurrence de chaque événement de l'application est connu, ce qui permet de dater les événements de manière exacte.



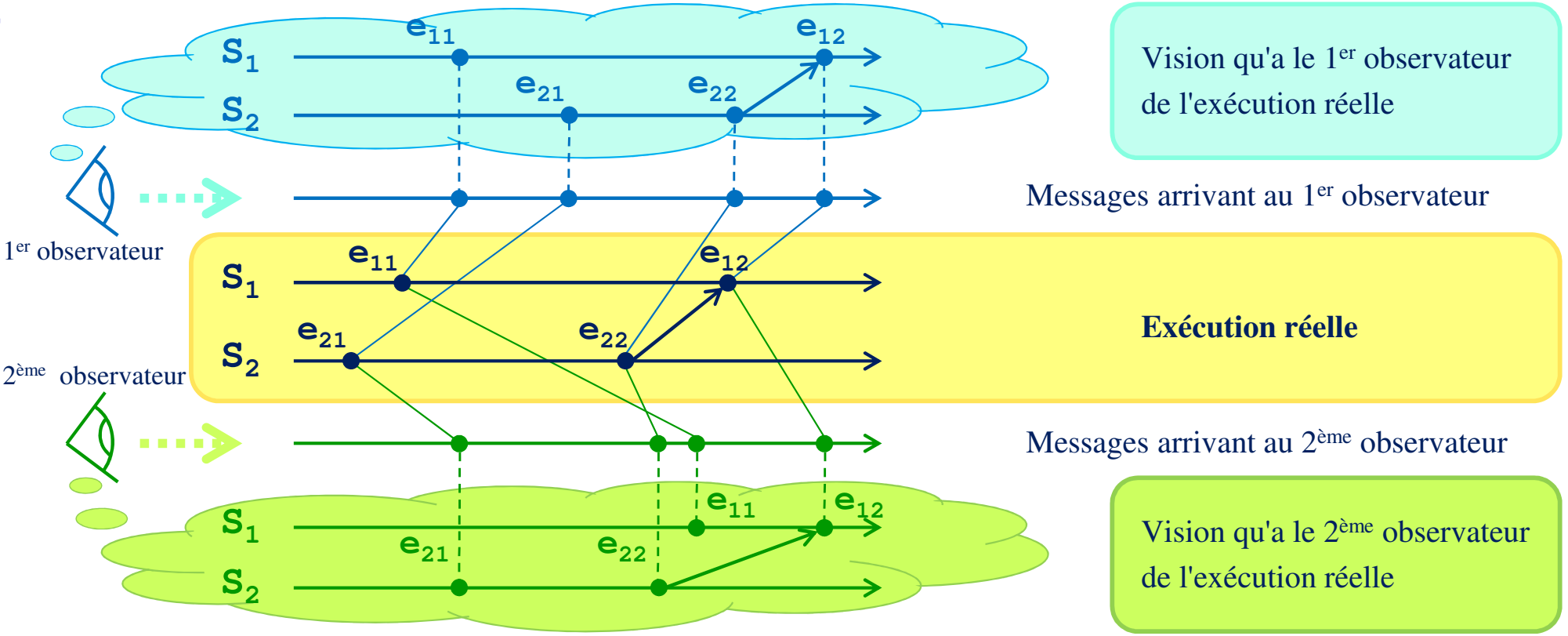

Dans le cas idéal, à chaque exécution correspond un diagramme de temps où ne figure qu'une seule ligne représentant le temps réel.

Cas réel:

- les dates des événements sont locales à chaque site,
- elles ne fournissent qu'une information partielle sur l'ordre des événements (à cause de la dérive des horloges),

  On peut classer les événements ayant lieu sur un même site, pas ceux ayant lieu sur des sites différents.
- ⇒ pour une même exécution, deux observateurs ayant observé localement les sites aux mêmes instants mais ayant rencontré des délais de transmission différents ne vont pas forcément reconstruire la même vision de l'exécution globale.

# Exécutions Équivalentes




- Aucune des deux visions ci-dessus ne préserve l'ordre réel d'occurrence des événements,
- Cependant, dans les deux cas, la relation de causalité est respectée par rapport à l'exécution réelle.

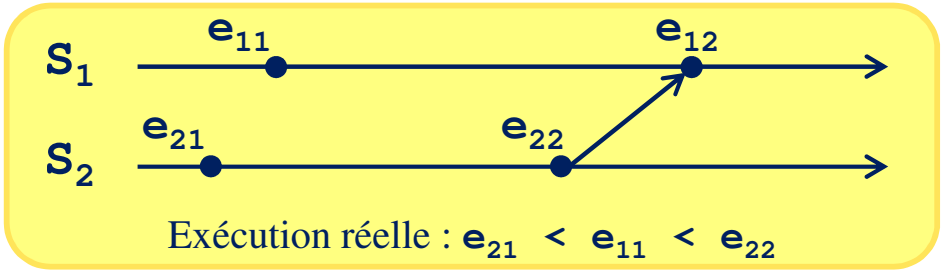
**Exécutions équivalentes** :

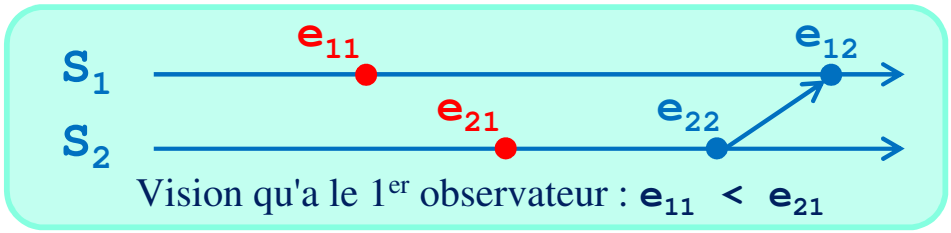
> Deux diagrammes de temps sont équivalents si l'on passe de l'un à l'autre en compressant ou en étirant les axes de temps et en respectant la contrainte qu'une flèche pointe toujours vers la droite (un message n'est pas reçu avant d'être envoyé).

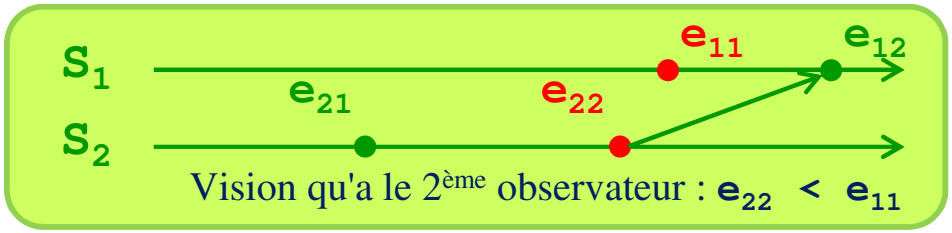
**Ici, on a trois exécutions équivalentes.**

# Exécutions Équivalentes

On notera que si les différentes représentations ci-dessous contiennent les mêmes ensembles d'événements et sont équivalentes, elles ne contiennent pas les mêmes configurations.




Il semble malgré tout raisonnable de considérer que la vision qu'a chaque observateur est une représentation valable de l'exécution réelle :

- lorsqu'on construira un état global, on n'imposera donc pas qu'il reflète une configuration qui s'est produite lors de l'exécution,
- on se contentera de rechercher des états qui reflètent des configurations possibles,

  $\Rightarrow$ des configurations qui existent dans au moins une représentation équivalente de l'exécution.

# Coupures, Coupure Cohérente, État Global

État d'une application répartie :

- ensemble des états locaux des sites (processus),
- ensemble des états des liaisons (ensemble des messages en transit).

État d'un site :

- reflète l'historique complet des communications de ce site,

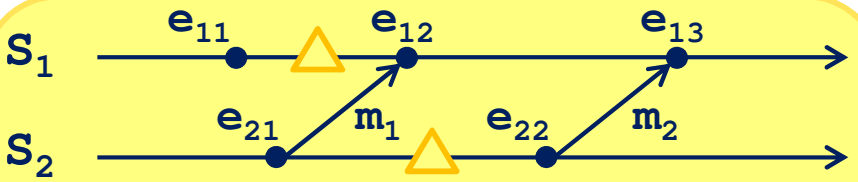
l'état local $c_i$ du site **i** inclut :

- s'il existe une liaison **(i, j)**, l'ensemble $sent_{ij}$ des messages envoyés par **i** à **j**,
- s'il existe une liaison **(j, i)**, l'ensemble $rcvd_{ji}$ des messages reçus par **i** en provenance de **j**.

Un algorithme de calcul d'état global doit reconstruire un état de l'application à partir des états locaux de chaque site :

- un site **i** enregistre son état local en prenant une photo de son état $c_i^*$ à cet instant, qu'on appellera état enregistré

- si l'état local est enregistré entre les événements $e_{ij}$ et $e_{i(j+1)}$, on appellera *pré-événements* les événements $e_{ik}$ tels que $k \leq j$ et *post-événements* les événements $e_{ik}$ tels que $k > j$.
- de même, on appellera *pré-messages* les messages émis dans un *pré-événement*, et *post-messages* les messages émis dans un *post-événement*.




$e_{11}$ et $e_{21}$ sont les pré-événements de $S_1$ et $S_2$ respectivement.

$\{e_{12}, e_{13}\}$ et $\{e_{22}\}$ sont les post-événements de $S_1$ et $S_2$ respectivement.

$m_1$ est un pré-message et $m_2$ est un post-message.

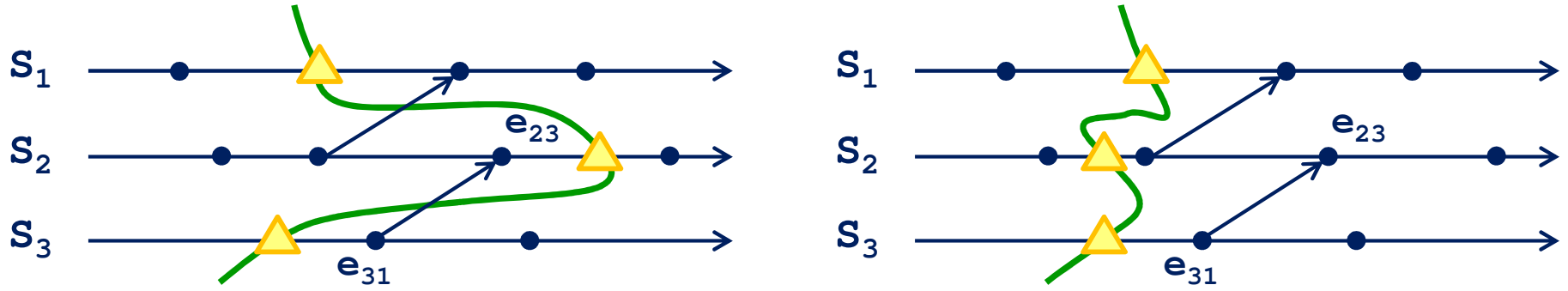
# Coupures, Coupure Cohérente, État Global

État global $S^*$:

- constitué des états locaux enregistrés par chaque site, noté $S^* = (c_1^*, \ldots, c_n^*)$
- l'état de la liaison $(i,j)$ pour l'état global $S^*$ est déterminé par $sent_{ij} \backslash rcvd_{ij}$

Coupure :

> Soit $E$ l'ensemble des événements de l'application. Une coupure de $E$ est un ensemble $C \subseteq E$ tel que :
>
> $\forall$ $i$ un site, $\forall$ $e_1, e_2$ deux événements de $i$, $e_1 \in C$ et $e_2 \rightarrow e_1 \Rightarrow e_2 \in C$
>
> où $\rightarrow$ dénote la relation de causalité.




Effectuer une coupure revient à prélever l'état local de chaque processus et séparer son passé de son futur.

Il existe une correspondance directe entre un état global et une coupure finie dans l'ensemble des événements de l'application répartie (on dira qu'un état global implique une coupure).

# Coupures, Coupure Cohérente, État Global

États globaux pour lesquels :

- $\mathbf{rcvd}_{ij}^{*} \not\subset \mathbf{sent}_{ij}^{*}$ (autrement dit, certains messages ont été reçus alors qu'ils n'ont pas été envoyés),
- ne correspondent à aucun état de l'application que l'on pourrait observer si on obtenait tous les états locaux en un point unique du temps.

État global réalisable :

> Un état global $\mathbf{S}^{*}$ est réalisable si, pour tout couple $\mathbf{(i,j)}$ de voisins, on a $\mathbf{rcvd}_{ij}^{*} \subseteq \mathbf{sent}_{ij}^{*}$.

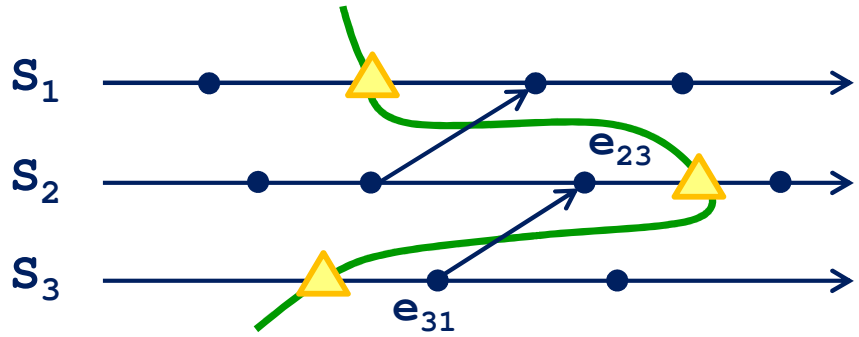
Coupure cohérente :

> Soit $\mathbf{E}$ l'ensemble des événements de l'application. Une coupure cohérente de $\mathbf{E}$ est un ensemble $\mathbf{C} \subseteq \mathbf{E}$ tel que :
>
> $$\forall\ \mathbf{e}_1, \mathbf{e}_2 \text{ deux événements de } \mathbf{E},\ \mathbf{e}_1 \in \mathbf{C} \text{ et } \mathbf{e}_2 \rightarrow \mathbf{e}_1 \Rightarrow \mathbf{e}_2 \in \mathbf{C}$$
>
> où $\rightarrow$ dénote la relation de causalité.

Il existe une correspondance directe entre un état global réalisable et une coupure finie cohérente dans l'ensemble des événements de l'application répartie.

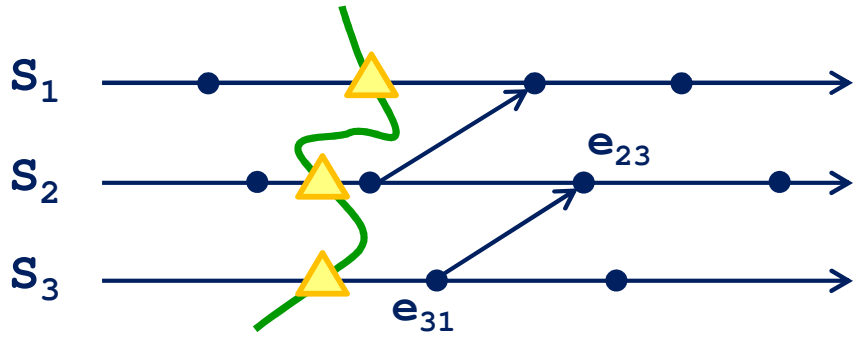
# Coupures, Coupure Cohérente, État Global

Exemples de coupures :




**Coupure non cohérente**

$e_{31}$ qui précède causalement $e_{23}$ n'appartient pas à la coupure alors que $e_{23}$ lui appartient




**Coupure cohérente**

État global calculé

- l'idéal serait qu'il corresponde à une configuration de l'application qui se produit effectivement au cours de l'exécution
- on se contentera en fait de calculer des états qui se produisent dans au moins une exécution équivalente de l'exécution réelle.

# Algorithmes de Calcul d'État Global

## Propriétés d'un algorithme de calcul d'état global

- doit garantir la cohérence de l'état global obtenu,
- doit éviter d'interférer avec l'application répartie en cours

  Le système peut se modifier pendant l'observation, mais l'observation ne doit en aucun cas altérer le système.

## Distinction des messages

- "messages de contrôle" : messages de l'algorithme de calcul d'état global,
- "messages de base" : messages de l'application répartie en cours (celle que l'on observe).

## Conditions que doit remplir un algorithme de calcul d'état global

(Coordination des prises de vues locales pour garantir que l'état global résultant est réalisable)

- un enregistrement de l'état local doit être déclenché sur chaque site, } état global
- il n'y a pas de post-message reçu dans un pré-événement.

} état global réalisable

# Algorithme de Chandy et Lamport

Principes de l'algorithme de Chandy et Lamport

- s'exécute en concurrence avec l'application répartie qu'il observe,
- les processus s'informent mutuellement de la construction d'un état global en s'envoyant des messages de contrôle (marqueurs) **`mkr`** sur chaque liaison,
- chaque site envoie un marqueur exactement une fois à chacun de ses voisins, au moment où il enregistre son état local,
- lorsqu'un site reçoit un marqueur, s'il n'a pas encore enregistré son état local, il le fait et renvoie un marqueur à chacun de ses voisins.

Variables du site **`i`** :

- **`voisins`**$_{\texttt{i}}$ : table des voisins du site **`i`**
- **`enregistré`**$_{\texttt{i}}$ : booléen, indique si l'état local a été enregistré
- **`site`** : variable locale, un des sites voisins.

# Algorithme de Chandy et Lamport

```
Initialiser(){
    enregistrer l'état local;
    enregistré_i = vrai;
    Pour tout site appartenant à voisins_i Faire
    |   envoyer_à(site, mkr);
    Finpour
}
```

```
sur_réception_de(j, mkr){
    Si  enregistré_i = faux) Alors
    |   enregistrer l'état local;
    |   enregistré_i = vrai;
    |   Pour tout site appartenant à voisins_i Faire
    |   |       envoyer_à(site, mkr);
    |   Finpour
    Fsi
}
```

# Algorithme de Chandy et Lamport

## Propriétés de l'algorithme de Chandy et Lamport

On se place comme d'habitude dans la situation où le réseau est connexe et les communications sont fiables.

L'algorithme de Chandy et Lamport ne fonctionne que sous l'hypothèse que les liaisons sont FIFO.

Propriété 1 :

> Si au moins un site initialise l'algorithme, alors tous les sites enregistrent leur état local au bout d'un temps fini.

Il est à noté que l'algorithme doit être initialisé par au moins un site, mais qu'il fonctionne aussi correctement dans le cas où un nombre quelconque (non nul) de sites exécutent l'initialisation.

Propriété 2 :

> L'algorithme de Chandy et Lamport calcule un état global réalisable au bout d'un temps fini dès lors qu'il a été initialisé par au moins un site.

## Complexité en nombre de messages de l'algorithme

- elle est de **2m**, où **m** est …

# Références


Cours d'Algorithmique Répartie, "État Global, Observation", Claude Dutheillet, Laboratoire Lip6, Université Paris 6.

F. Mattern "Virtual time and global states of distributed systems", In : Parallel and Distributed Algorithms, M. Cosnard et al. eds, North-Holland, 1989, pp 215-226

M. Raynal "Synchronisation et état global dans les systèmes répartis", Collection Direction des Études et des Recherches d'EDF n° 79. Hermès. 1992 ISSN 0399-4198

R. Schwarz, F. Mattern

Detecting Causal Relationships in Distributed Computations, in search of the holy grail

Rapport SFB 124-15/92, Décembre 1992

G. Tel "Introduction to Distributed Algorithms" Second Edition Cambridge University Press. ISBN 0-521-79483-8. 2000