

---

# Cours ASPD

## Principaux Algorithmes Répartis

### Telecom Nancy 2A IL

---

Dominique Méry  
Telecom Nancy  
Université de Lorraine

---

Année universitaire 2024-2025  
21 janvier 2026(1:43pm)

# Sommaire

---

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

# Section Courante

---

## ① Election

Problème de l'élection

]

## ② Election

Problème de l'élection

Election dans un graphe  
acyclique

## ③ Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## ④ Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et  
Lamport

## ① Election

Problème de l'élection

## ② Election

Problème de l'élection

Election dans un graphe acyclique

## ③ Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## ④ Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

# Election du leader

---

- Un algorithme d'élection dun leader satisfait les propriétés suivantes :
  - ▶ Chaque processus ou site du réseau exécute le même algorithme.
  - ▶ L'algorithme est décentralisé : le calcul peut être initialisé par un sous-ensemble arbitraire de processus
  - ▶ L'algorithme atteint une configuration terminale pour chaque calcul et dans toute configuration terminale accessible, il n'y a qu'un seul processus dans l'état de savoir qu'il est leader et tous les autres savent qu'ils ne sont pas leader c'est-à-dire perdus.
- Algorithmes considérés
  - ▶ Algorithme de LeLann et Chang et Roberts (anneau)
  - ▶ Algorithme IEEE 1394 (arborescence)

# Problème de l'élection d'un leader

---

- Les sites d'un réseau n'ont pas conscience des autres sites non connectés à eux.
- Chaque site a une connaissance locale
- Organiser des calculs dans un réseau nécessite de coordonner les calculs : besoin d'un leader.
- Calculs répartis par ondelettes

# Algorithme de LeLann

---

- Un ensemble de nœuds connectés en anneau
- Un sous-ensemble des nœuds est appelé l'ensemble des **candidats** à l'élection
- Chaque candidat maintient un ensemble de valeurs reçues des autres nœuds.
- Si un nœud reçoit le jeton d'un autre nœud et qu'il n'est pas candidat, il ne peut plus être candidat et passe le jeton à son voisin.
- Quand un nœud a reçu son jeton envoyé, il regarde s'il est le jeton de numéro le plus petit et il est élu.

# Algorithme de Chang et Roberts

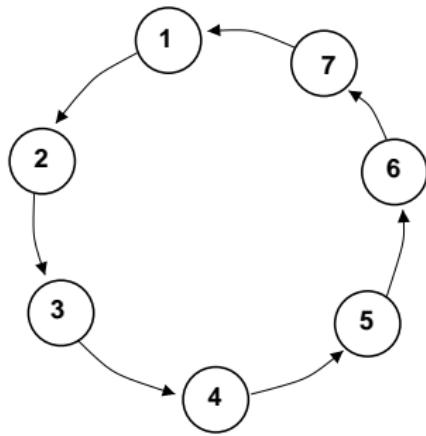
---

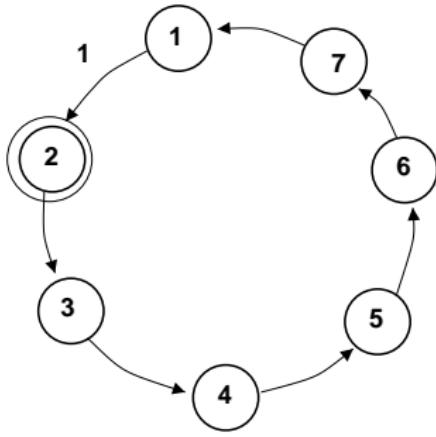
- Un ensemble de nœuds connectés en anneau
- Un sous-ensemble des nœuds est appelé l'ensemble des candidats à l'élection
- Si un nœud candidat reçoit un jeton plus petit que son jeton, il passe le jeton et se déclare élu, sinon il garde le jeton.
- Si un nœud reçoit le jeton d'un autre nœud et qu'il n'est pas candidat, il ne peut plus être candidat et passe le jeton à son voisin.
- Quand un nœud a reçu son jeton envoyé, il est élu.

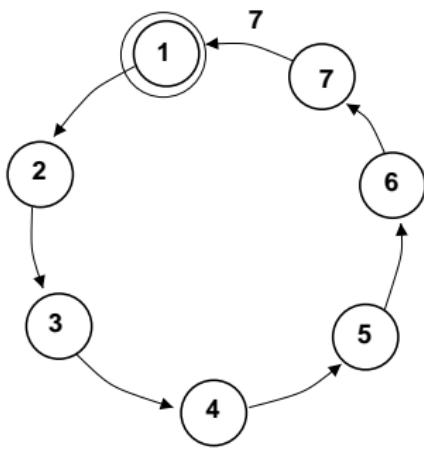
# Algorithme LRC Chang et Roberts

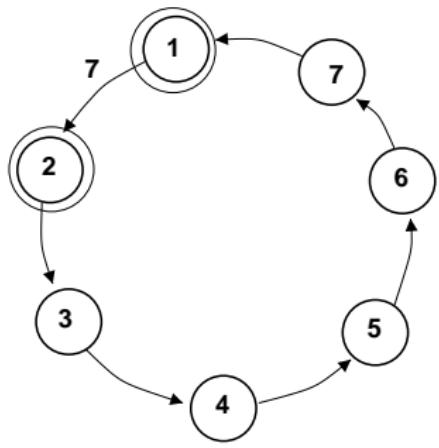
---

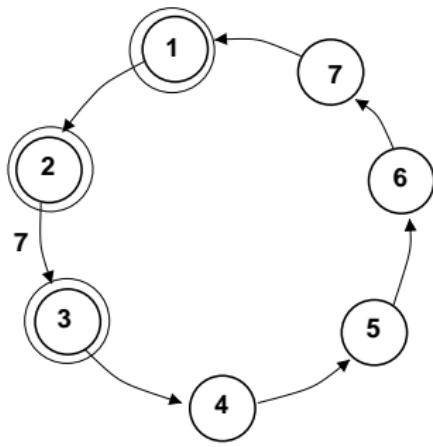
- Un réseau en anneau comporte des nœuds ayant un numéro différent.
- Un des nœuds est le maximum du réseau
- Le processus d'élection transmet un jeton avec une valeur mise à jour à chaque nœud.

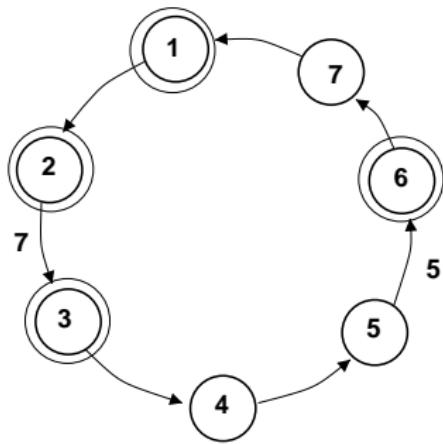


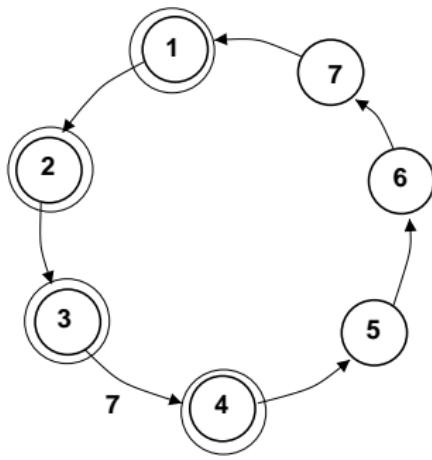


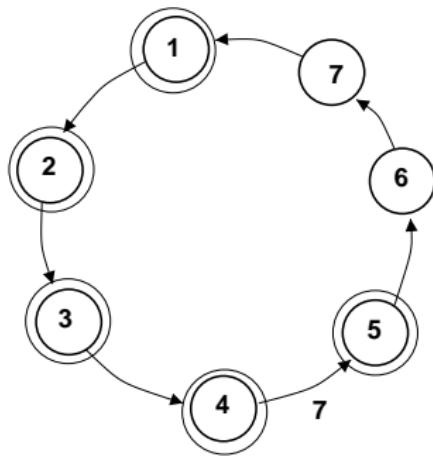












# Algorithme LCR

---

- Seul le nœud de numéro maximal sera élu leader
- Les nœuds de numéro minimal restent dans l'état inconnu

# Section Courante

---

## 1 Election

Problème de l'élection

]

## 2 Election

Problème de l'élection

Election dans un graphe  
acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et  
Lamport

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

# Election du leader

---

- Un algorithme d'élection dun leader satisfait les propriétés suivantes :
  - ▶ Chaque processus ou site du réseau exécute le même algorithme.
  - ▶ L'algorithme est décentralisé : le calcul peut être initialisé par un sous-ensemble arbitraire de processus
  - ▶ L'algorithme atteint une configuration terminale pour chaque calcul et dans toute configuration terminale accessible, il n'y a qu'un seul processus dans l'état de savoir qu'il est leader et tous les autres savent qu'ils ne sont pas leader c'est-à-dire perdus.
- Algorithmes considérés
  - ▶ Algorithme de LeLann et Chang et Roberts (anneau)
  - ▶ Algorithme IEEE 1394 (arborescence)

# Problème de l'élection d'un leader

---

- Les sites d'un réseau n'ont pas conscience des autres sites non connectés à eux.
- Chaque site a une connaissance locale
- Organiser des calculs dans un réseau nécessite de coordonner les calculs : besoin d'un leader.
- Calculs répartis par ondelettes

# Algorithme de LeLann

---

- Un ensemble de nœuds connectés en anneau
- Un sous-ensemble des nœuds est appelé l'ensemble des **candidats** à l'élection
- Chaque candidat maintient un ensemble de valeurs reçues des autres nœuds.
- Si un nœud reçoit le jeton d'un autre nœud et qu'il n'est pas candidat, il ne peut plus être candidat et passe le jeton à son voisin.
- Quand un nœud a reçu son jeton envoyé, il regarde s'il est le jeton de numéro le plus petit et il est élu.

# Algorithme de Chang et Roberts

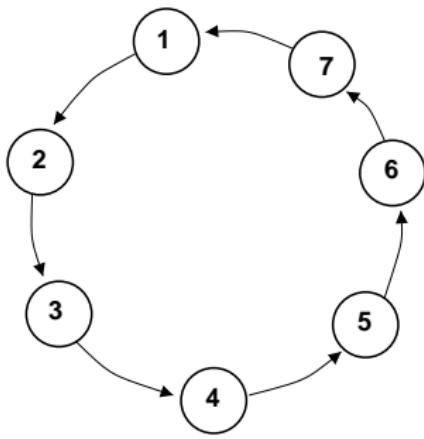
---

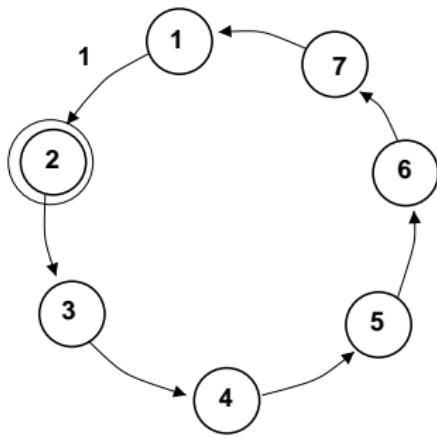
- Un ensemble de nœuds connectés en anneau
- Un sous-ensemble des nœuds est appelé l'ensemble des candidats à l'élection
- Si un nœud candidat reçoit un jeton plus petit que son jeton, il passe le jeton et se déclare élu, sinon il garde le jeton.
- Si un nœud reçoit le jeton d'un autre nœud et qu'il n'est pas candidat, il ne peut plus être candidat et passe le jeton à son voisin.
- Quand un nœud a reçu son jeton envoyé, il est élu.

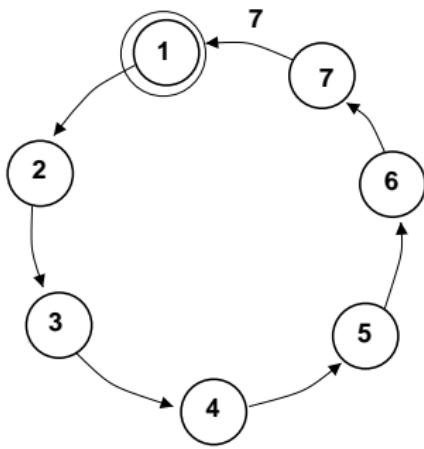
# Algorithme LRC Chang et Roberts

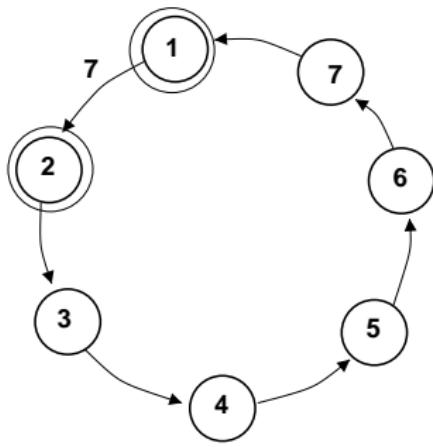
---

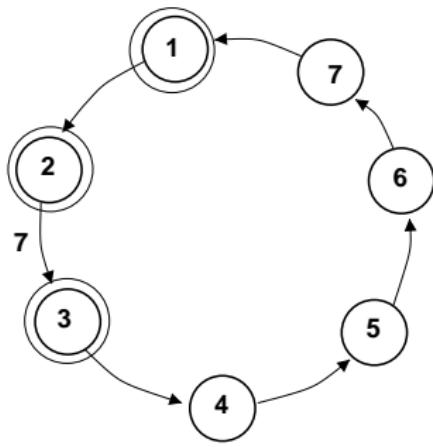
- Un réseau en anneau comporte des nœuds ayant un numéro différent.
- Un des nœuds est le maximum du réseau
- Le processus d'élection transmet un jeton avec une valeur mise à jour à chaque nœud.

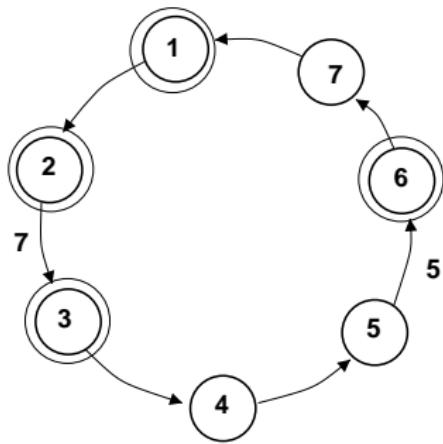


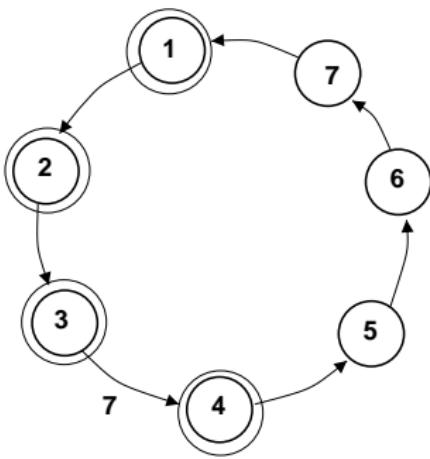


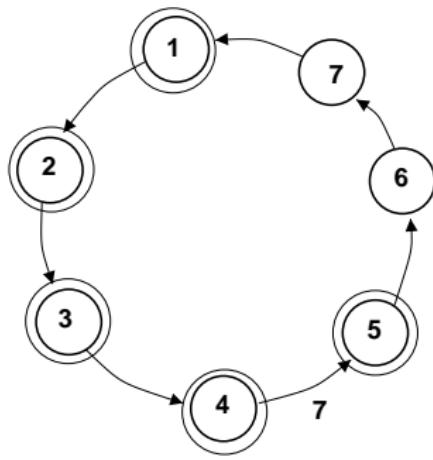












# Algorithme LCR

---

- Seul le nœud de numéro maximal sera élu leader
- Les nœuds de numéro minimal restent dans l'état inconnu

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

- standard international
- Intérêts commerciaux pour sa correction
- Sun, Apple, Philips, Microsoft, Sony, etc impliqués dans son développement
- Trois couches (physique, liaison, transaction)
- Le protocole d'étude est le Tree Identify Protocol
- Localisé dans la phase Bus Reset de la couche physique

# Le Problème (1)

---

- Le bus est utilisé pour acheminer des **signaux** audio et vidéo digitalisés.
- Il est "**hot-pluggable**"
- Unités et périphériques peuvent être **ajoutés ou retirés à tout instant**.
- De tels changements sont suivis d'un **bus reset**
- L'**élection du leader** a lieu après un reset dans le réseau.
- Un leader doit être choisi pour agir en tant que **manager du bus**

## Le Problème (2)

---

- Après un reset du bus tous les nœuds ont **même statut**.
- **Tout nœud** sait à quels nœuds il est directement connecté.
- Le réseau est **connexe**
- Le réseau est **acyclique**

## BASE

- IEEE. *IEEE Standard for a High Performance Serial Bus. Std 1394-1995.* 1995
- IEEE. *IEEE Standard for a High Performance Serial Bus (supplement). Std 1394a-2000.* 2000

### GENERAL

- N. Lynch. *Distributed Algorithms*. Morgan Kaufmann. 1996
- R. G. Gallager et al. *A Distributed Algorithm for Minimum Weight Spanning Trees*. IEEE Trans. on Prog. Lang. and Systems. 1983.

## MODEL CHECKING

- D.P.L. Simons et al. *Mechanical Verification of the IEEE 1394a Root Contention Protocol using Uppaal2* Springer International Journal of Software Tools for Technology Transfer. 2001
- H. Toetenel et al. *Parametric verification of the IEEE 1394a Root Contention Protocol using LPMC* Proceedings of the 7th International Conference on Real-time Computing Systems and Applications. IEEE Computer Society Press. 2000

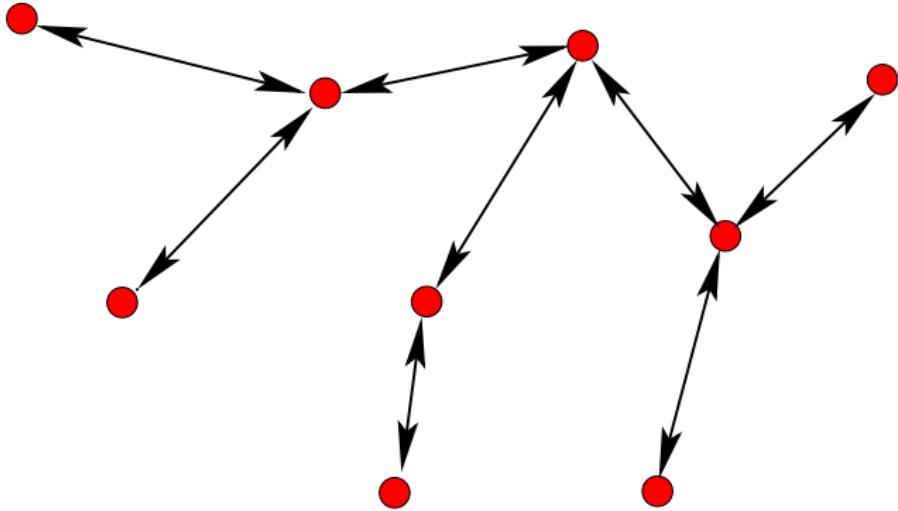
### THEOREM PROVING

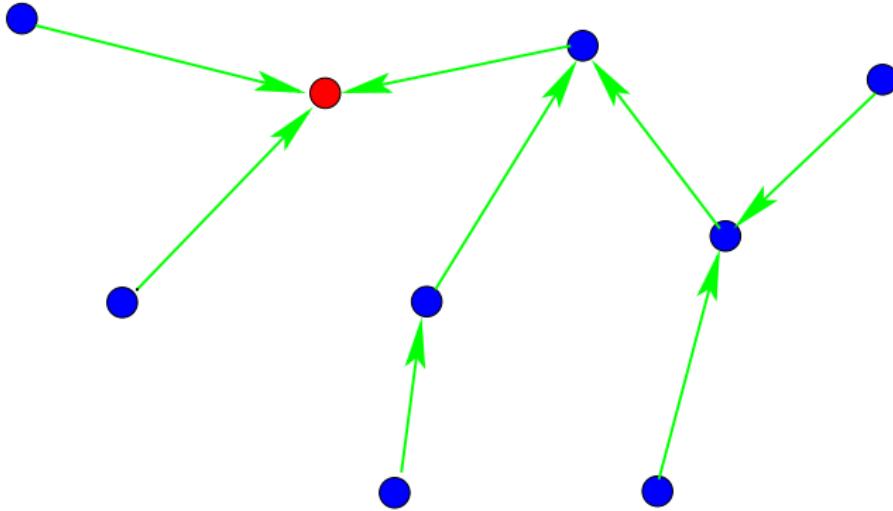
- M. Devillers et al. *Verification of the Leader Election : Formal Method Applied to IEEE 1394.* Formal Methods in System Design. 2000
- J.R. Abrial et al. *A Mechanically Proved and Incremental Development of IEEE 1394.* Formal Aspects of Computing, 2003.

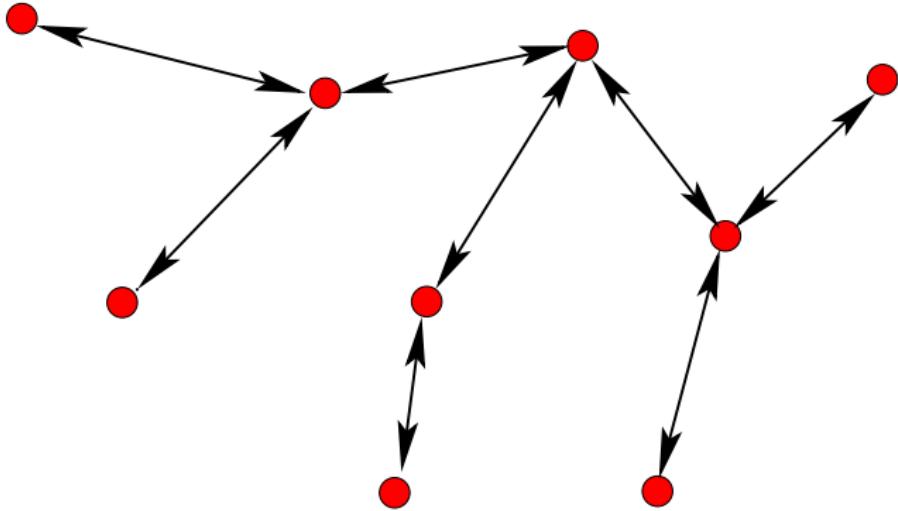
# Propriétés informelles abstraites du protocole

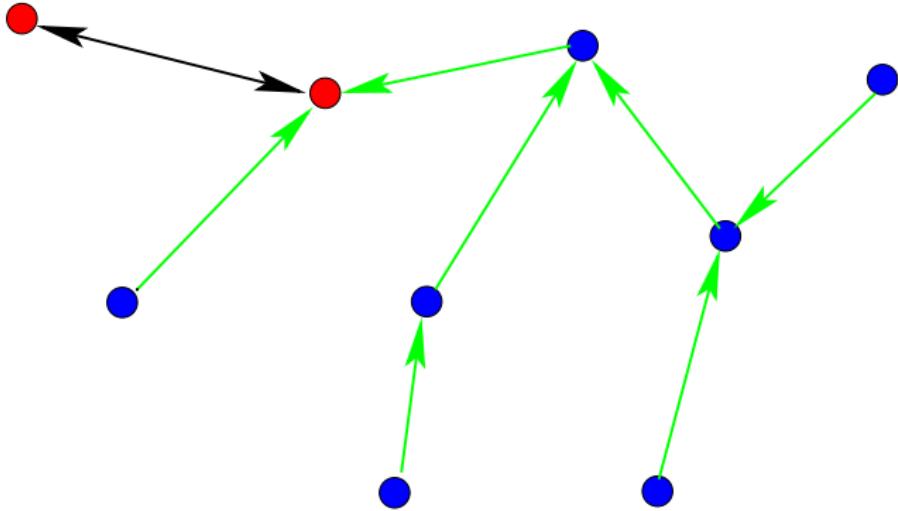
---

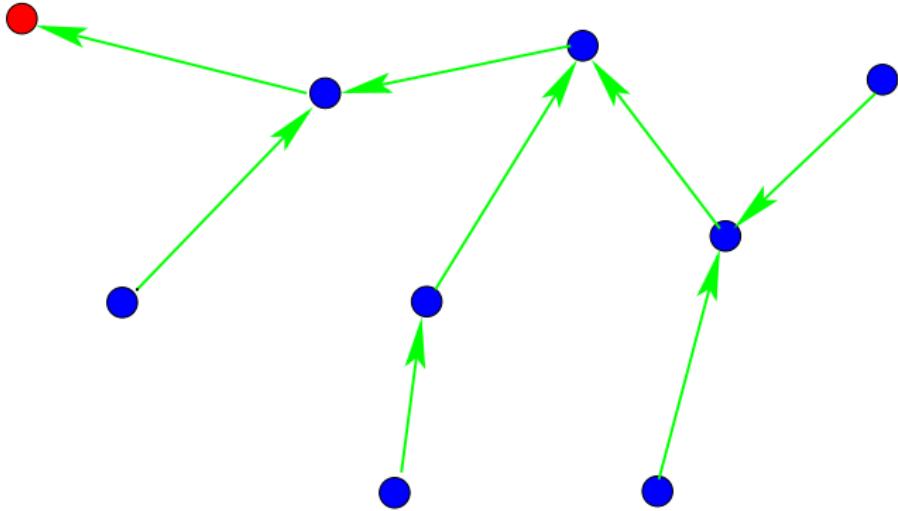
- Un réseau **connexe** et **acyclique** de nœuds.
- Nœuds reliés par des canaux **bidirectionnels**
- Election en un temps fini d'un **leader**.
- De manière **répartie** et **non-déterministe**.
- Un exemple d'animation du protocole.







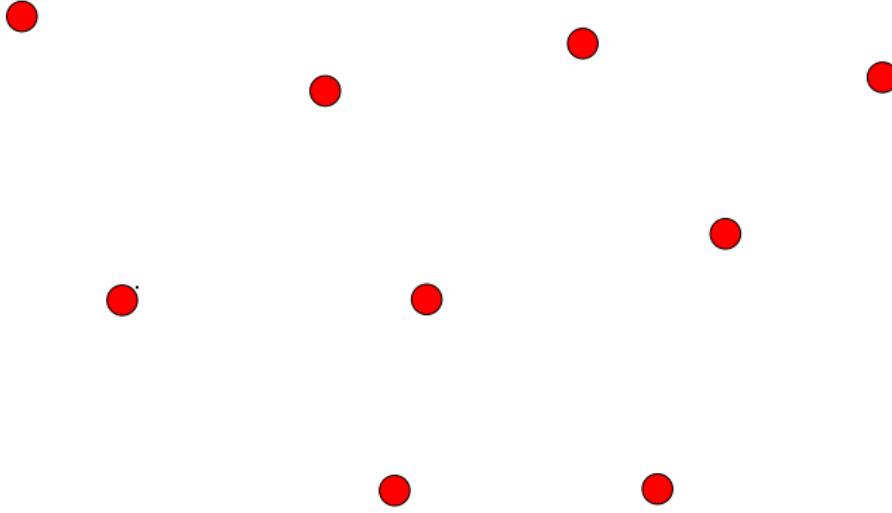




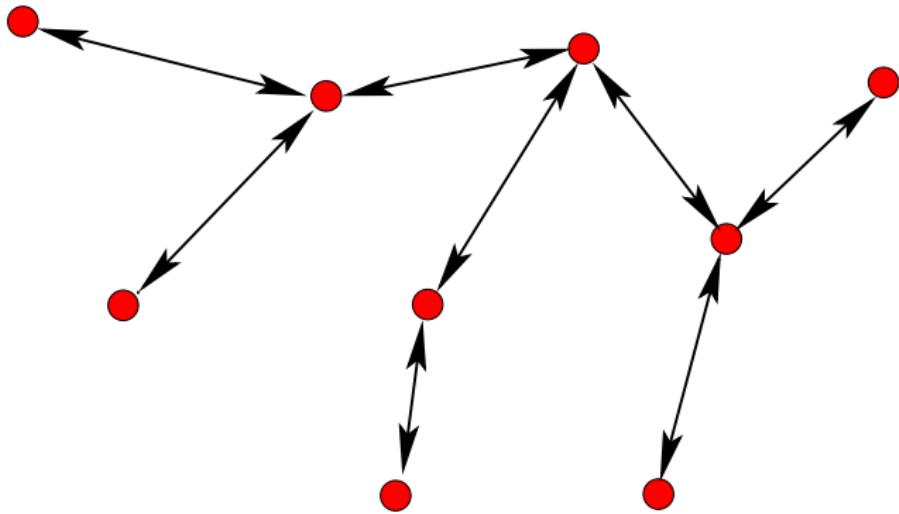
# Le processus de développement

---

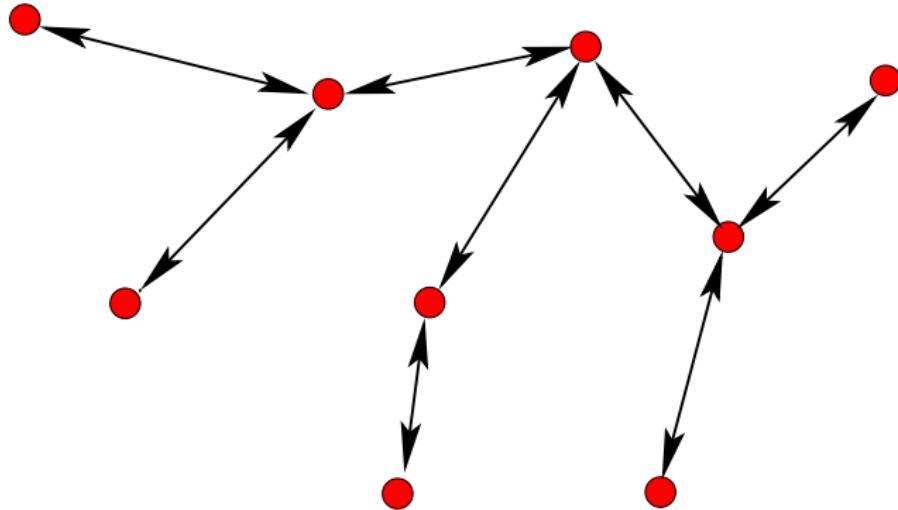
- Définition et propriétés du réseau dans un style formel
- Un modèle abstrait *one-shot* du protocole.
- Présentation d'une solution centralisée mais encore abstraite !
- Introduction du mécanisme de message passing entre les nœuds et des délais
- Modification de la structure de donnée pour répartir le protocole



ND un ensemble de noeuds (au moins 2 noeuds)



gr un graphe construit et défini sur ND



gr est un graphe symétrique et non réflexif

*gr* : un graphe construit sur  $ND$

$$gr \subseteq ND \times ND$$

*gr* : un graphe construit sur  $ND$

$$gr \subseteq ND \times ND$$

*gr* est défini sur  $ND$

$$\text{dom}(gr) = ND$$

$gr$  : un graphe construit sur  $ND$

$$gr \subseteq ND \times ND$$

$gr$  est défini sur  $ND$

$$\text{dom}(gr) = ND$$

$gr$  est symétrique

$$gr = gr^{-1}$$

*gr* : un graphe construit sur  $ND$

$$gr \subseteq ND \times ND$$

*gr* est défini sur  $ND$

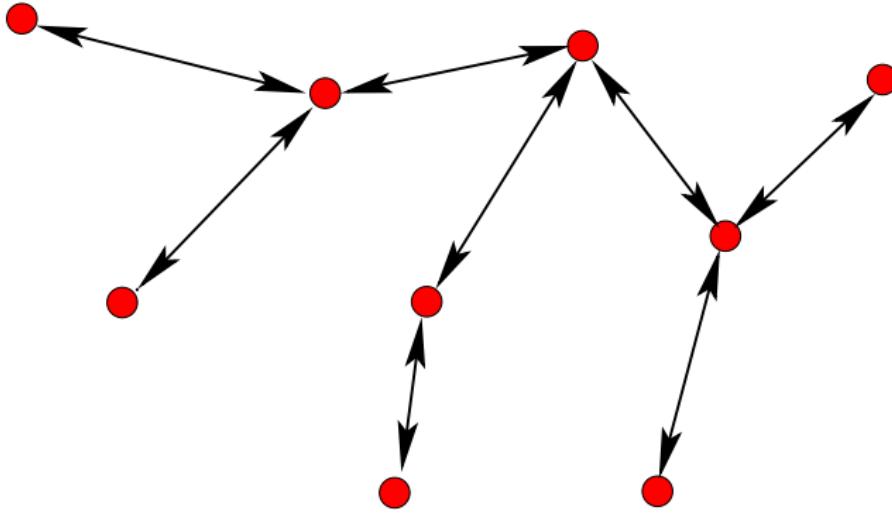
$$\text{dom}(gr) = ND$$

*gr* est symétrique

$$gr = gr^{-1}$$

*gr* est non réflexif

$$\text{id}(ND) \cap gr = \emptyset$$

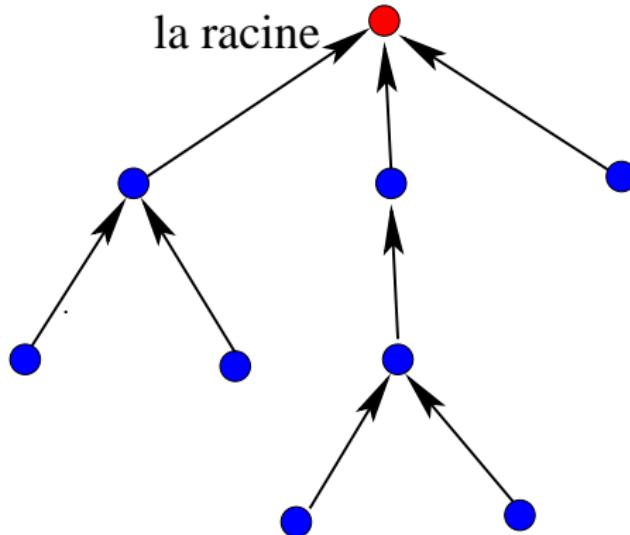


gr est connexe et acyclique

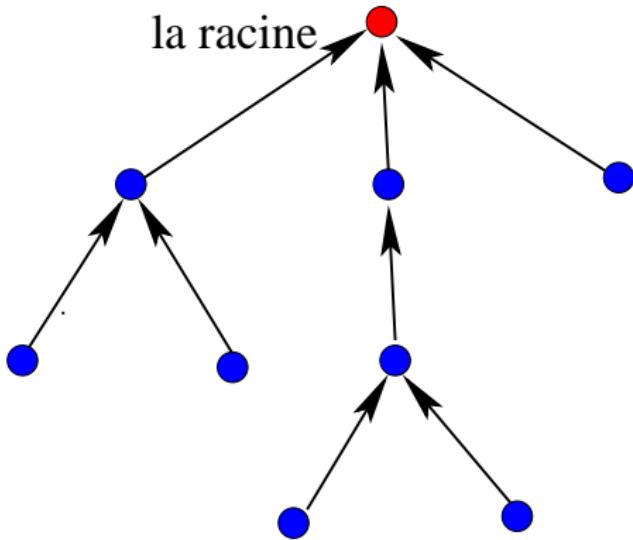
# Un détour par les Arbres

---

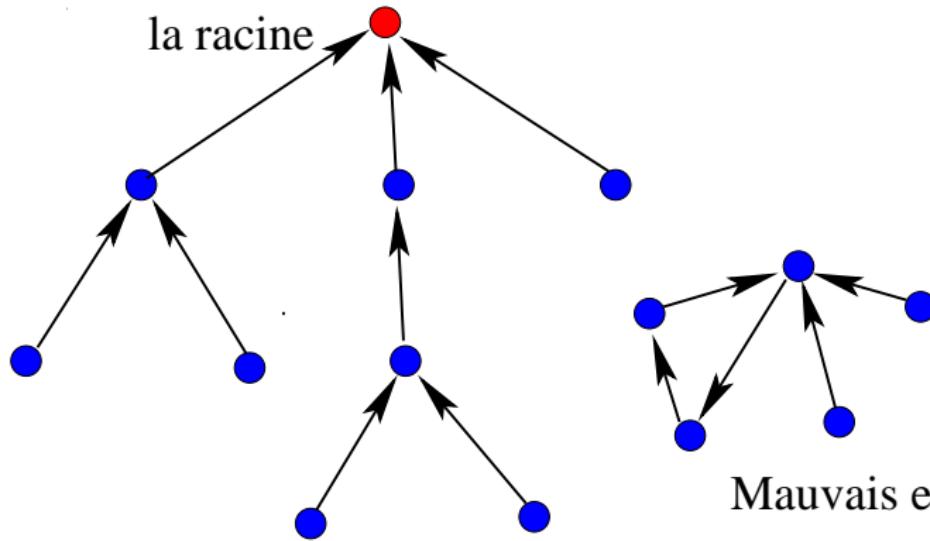
- Un arbre est un **graphe particulier**
- Un arbre a une **racine**
- Un arbre a une **fonction parentale**
- Un arbre est **acyclique**
- Un arbre est **connexe à partir de la racine**



### Un arbre t construit sur les noeuds

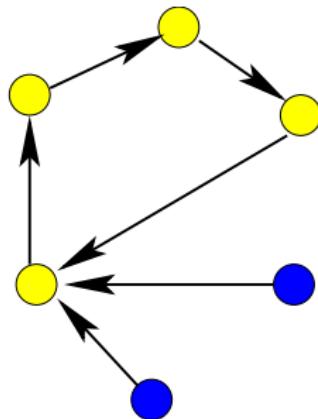


t est une fonction définie sur ND sauf pour la racine!

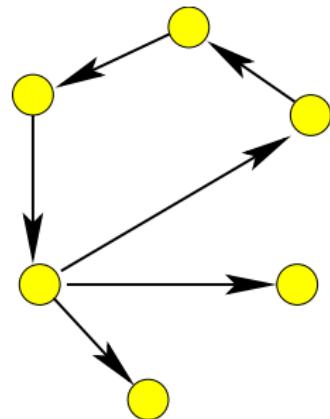


## Eviter des cycles

## Mauvais exemple!



Un cycle



Son image inverse

Les noeuds d'un cycle sont inclus  
dans leur image inverse

## Le prédictat **tree** ( $r, t$ )

Le prédictat **tree** ( $r, t$ )

$r$  est un élément de  $ND$      $r \in ND$

Le prédicat **tree**( $r, t$ )

*r* est un élément de  $ND$      $r \in ND$

*t* est une fonction  $t \in ND - \{r\} \rightarrow ND$

Le prédicat **tree** ( $r, t$ )

*r* est un élément de *ND*     $r \in ND$

*t* est une fonction

$$t \in ND - \{r\} \rightarrow ND$$

*t* est acyclique

$$\forall p . \left( \begin{array}{l} p \subseteq ND \wedge \\ p \subseteq t^{-1}[p] \\ \implies \\ p = \emptyset \end{array} \right)$$

$t$  est acyclique : formulations équivalentes

$$\forall p \cdot \left( \begin{array}{l} p \subseteq ND \wedge \\ p \subseteq t^{-1}[p] \\ \implies \\ p = \emptyset \end{array} \right) \Leftrightarrow \forall q \cdot \left( \begin{array}{l} q \subseteq ND \wedge \\ r \in q \wedge \\ t^{-1}[q] \subseteq q \\ \implies \\ ND \subseteq q \end{array} \right)$$

## On obtient une Règle d'Induction

$$\forall q \cdot$$
$$\left( \begin{array}{l} q \subseteq ND \wedge \\ r \in q \wedge \\ \forall x \cdot (x \in ND - \{r\} \wedge t(x) \in q \implies x \in q) \\ \implies \\ ND \subseteq q \end{array} \right)$$

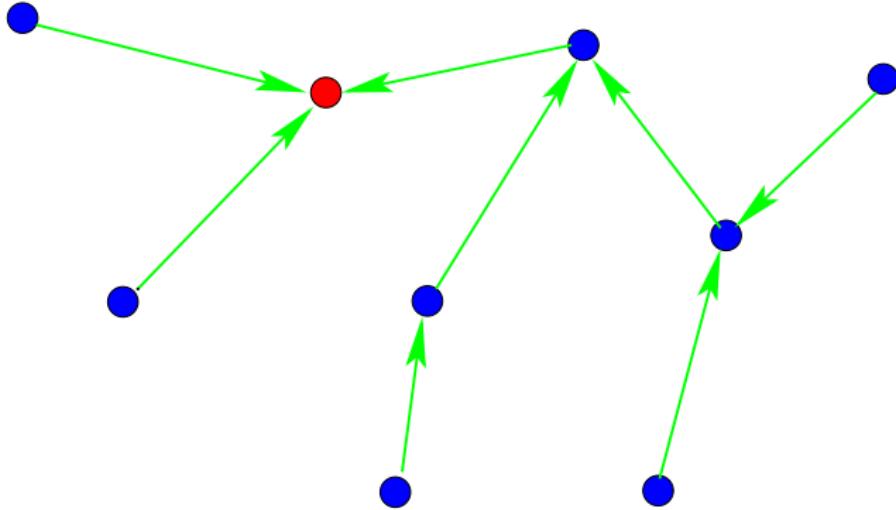
Le prédicat **tree** ( $r, t$ )

*r* est un élément de *ND*     $r \in ND$

**t est une fonction**       $t \in ND - \{r\} \rightarrow ND$

*t* est acyclique

$$\forall q \cdot \left( \begin{array}{l} q \subseteq ND \wedge \\ r \in q \wedge \\ t^{-1}[q] \subseteq q \\ \hline \Rightarrow \\ ND \subseteq q \end{array} \right)$$



Un arbre de recouvrement  $t$  de  $gr$

Le prédictat  $\text{spanning}(r, t, gr)$

$r, t$  est un arbre  $\text{tree}(r, t)$

$t$  est inclus dans  $gr$   $t \subseteq gr$

## Le graphe $gr$ est connexe et acyclique (1)

- Définir une relation  $fn$  liant un noeud aux possibles spanning trees de  $gr$  ayant ce noeud comme racine :

$$fn \subseteq ND \times (ND \rightarrow ND)$$

$$\forall(r, t) \cdot \left( \begin{array}{l} r \in ND \wedge \\ t \in ND \rightarrow ND \\ \implies \\ (r, t) \in fn \Leftrightarrow \text{spanning}(r, t, gr) \end{array} \right)$$

## Le graphe $gr$ est connexe et acyclique (2)

Totalité de la relation  $fn \implies$  Connectivité of  $gr$

Fonctionnalité d'une relation  $fn \implies$  Acyclicité de  $gr$

## Point sur les constantes $gr$ and $fn$

---

$$gr \subseteq ND \times ND$$

$$\text{dom}(gr) = ND$$

$$gr = gr^{-1}$$

$$\text{id}(ND) \cap gr = \emptyset$$

$$fn \in ND \rightarrow (ND \rightarrow ND)$$

$$\forall(r, t) \cdot \left( \begin{array}{l} r \in ND \wedge \\ t \in ND \rightarrow ND \\ \implies \\ t = fn(r) \Leftrightarrow \text{spanning}(r, t, gr) \end{array} \right)$$

- Variables  $rt$  et  $ts$

$rt \in ND$

$ts \in ND \leftrightarrow ND$

EVENT elect  $\hat{=}$

**BEGIN**

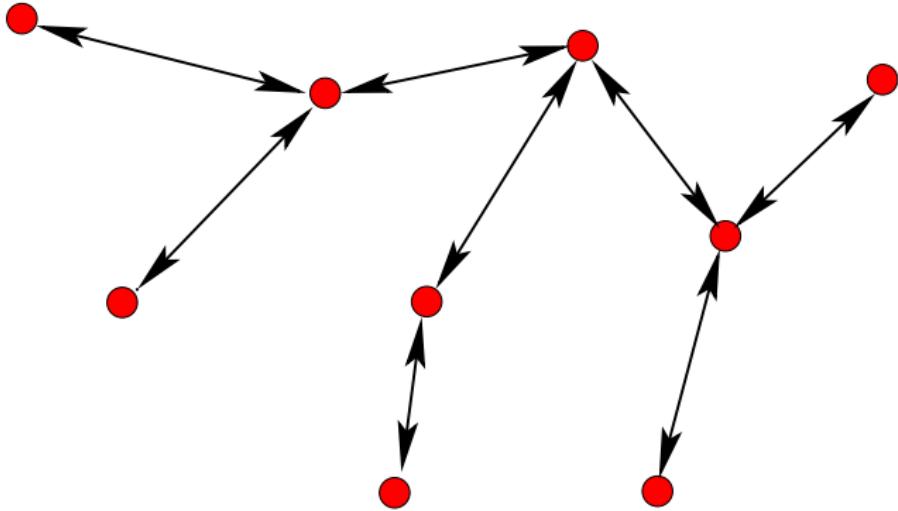
$rt, ts : \text{spanning}(rt, ts, gr)$

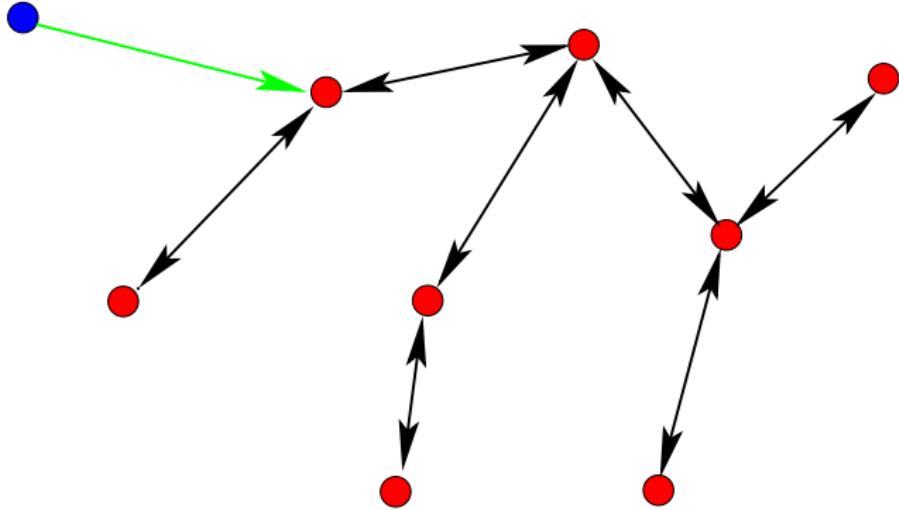
**END**

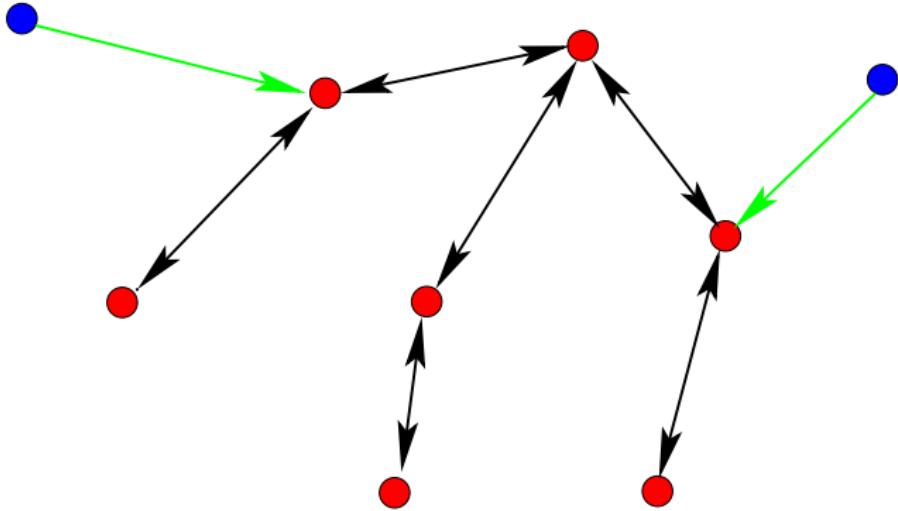
## Premier raffinement (1)

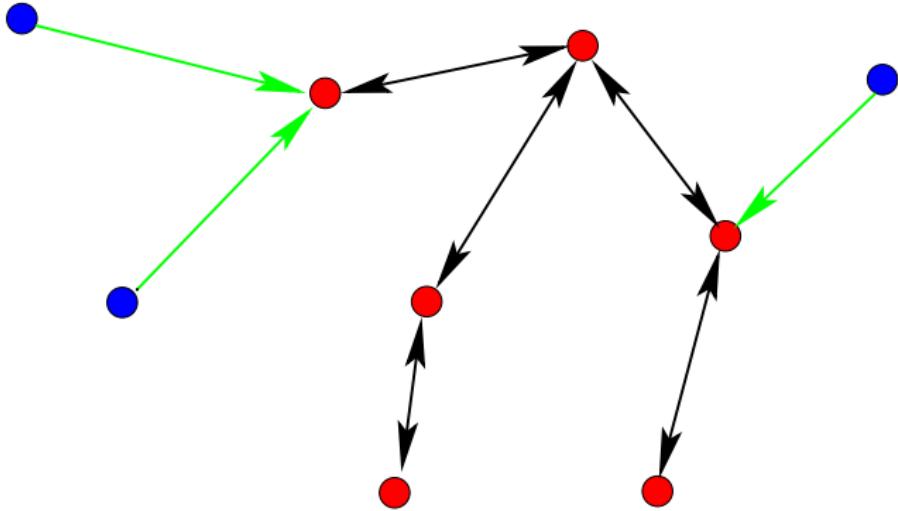
---

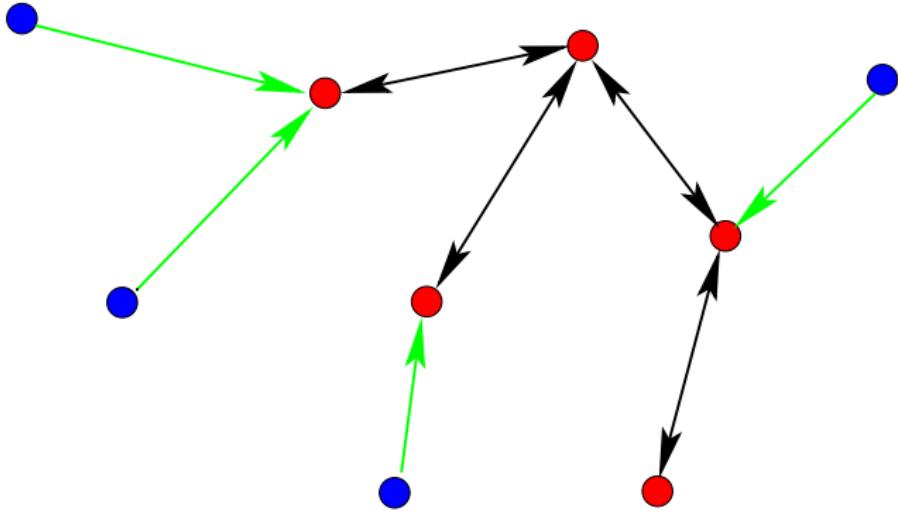
- Introduire une **nouvelle variable**,  $tr$ , correspondant à l'  
"arbre" en construction
- Introduire un **nouvel événement** : progression
- Définir l'**invariant**
- Retour à l'**animation** : Observez la construction de l'arbre

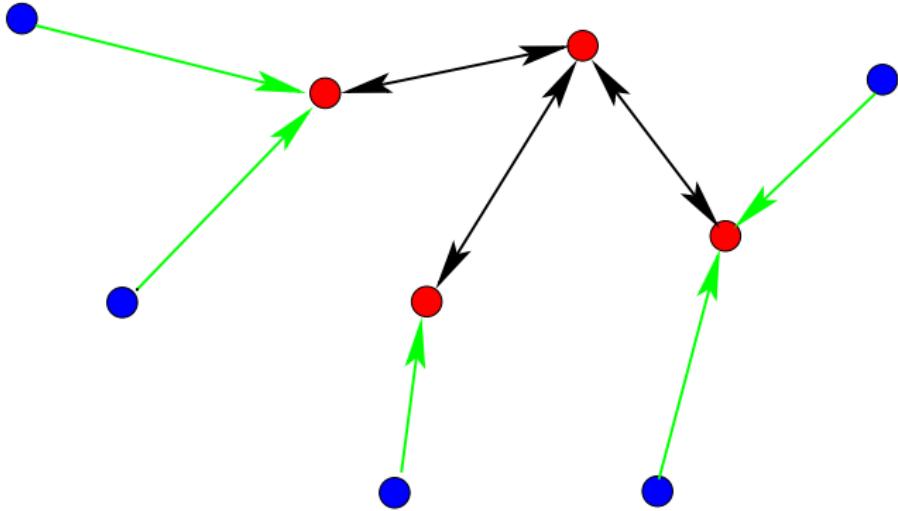


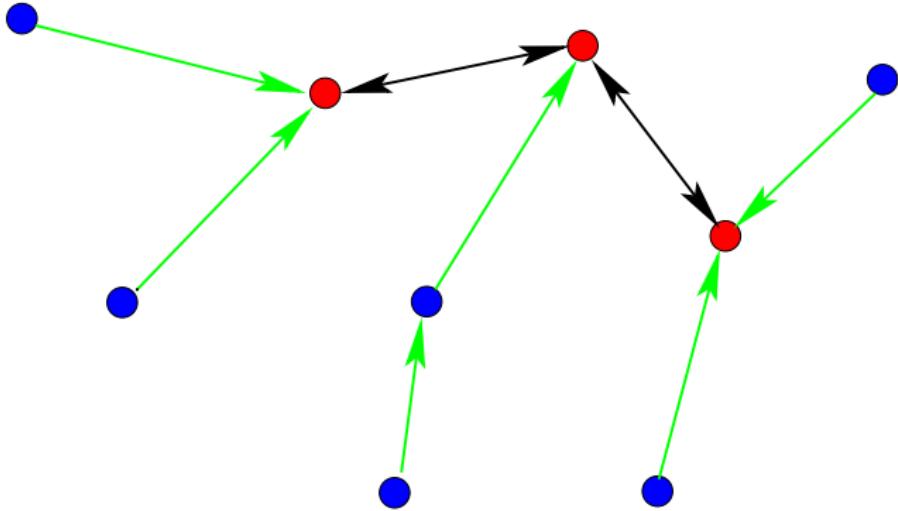


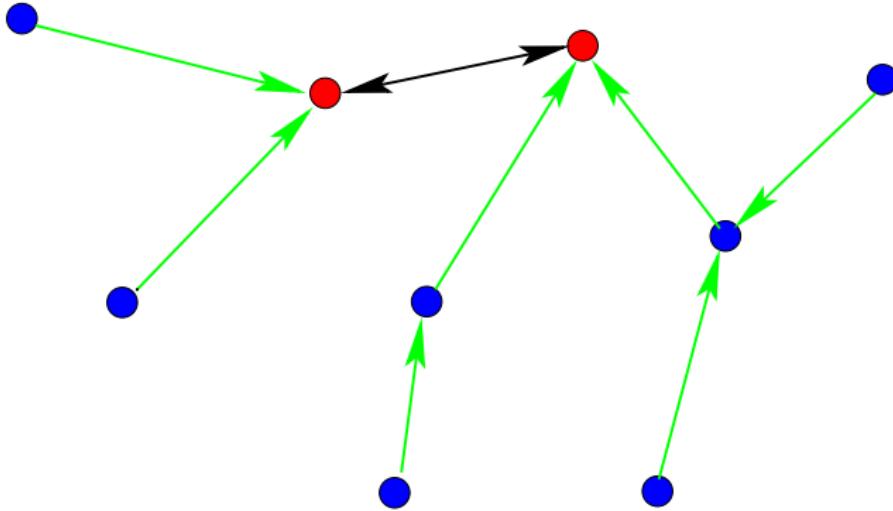


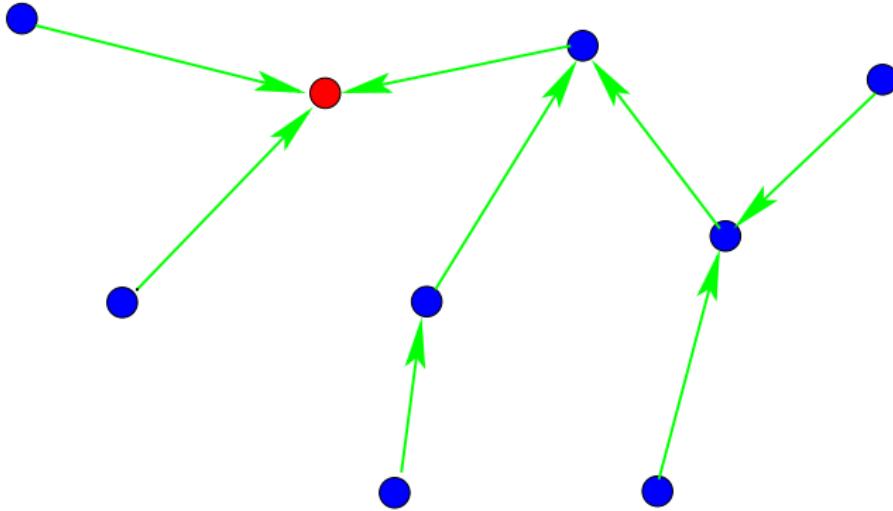












- Les flèches vertes correspondent à la fonction  $tr$
- Les nœuds bleus sont le domaine de  $tr$
- La fonction  $tr$  est une forêt sur les nœuds
- Les nœuds rouges sont les racines de ce arbres

## Le prédicat **invariant** (*tr*)

$tr \in ND \rightarrow ND$

## Le prédictat **invariant**(*tr*)

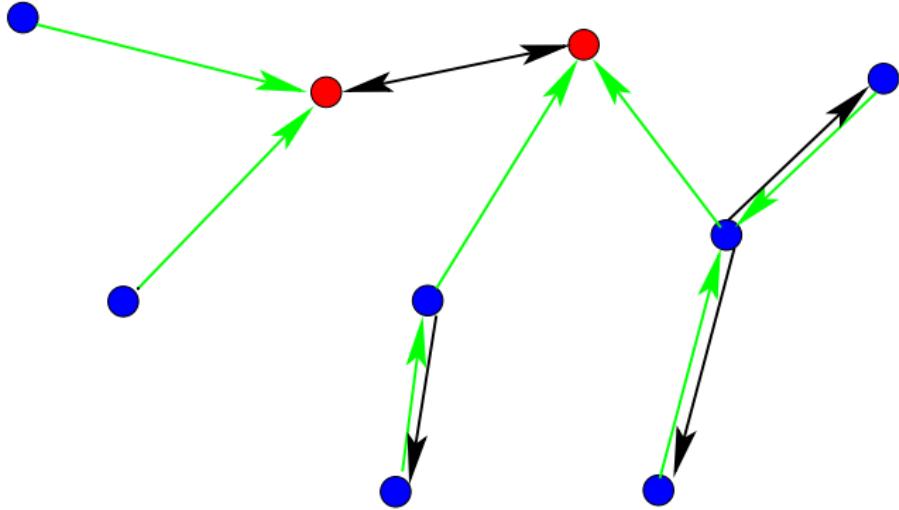
*tr* ∈  $ND \rightarrow ND$

$$\forall p \cdot \left( \begin{array}{l} p \subseteq ND \quad \wedge \\ \textcolor{red}{ND\text{-dom}\,(tr)} \subseteq p \quad \wedge \\ tr^{-1}[p] \subseteq p \\ \implies \\ ND \subseteq p \end{array} \right)$$

## Le prédictat **invariant** ( $tr$ )

$tr \in ND \rightarrow ND$

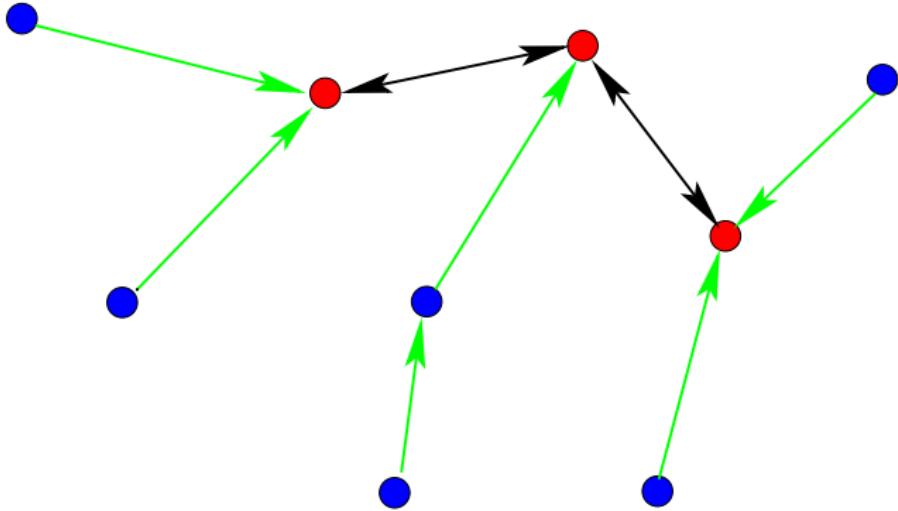
$$\text{dom } (tr) \lhd (tr \cup tr^{-1}) = \text{dom } (tr) \lhd gr$$

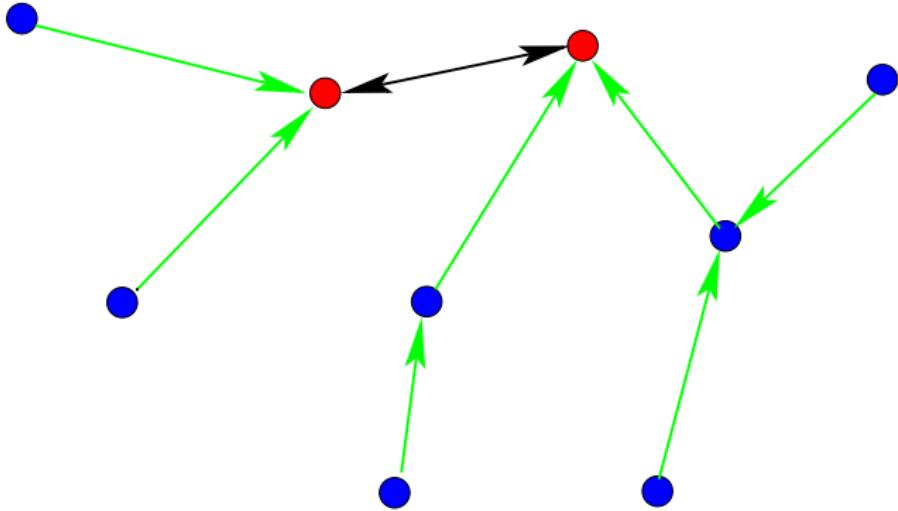


## Premier raffinement (2)

---

- Introduire le **nouvel événement** "progress"
- Raffiner l'événement abstrait "elect"
- Retour à l'**animation** : Observez la "garde" de progress



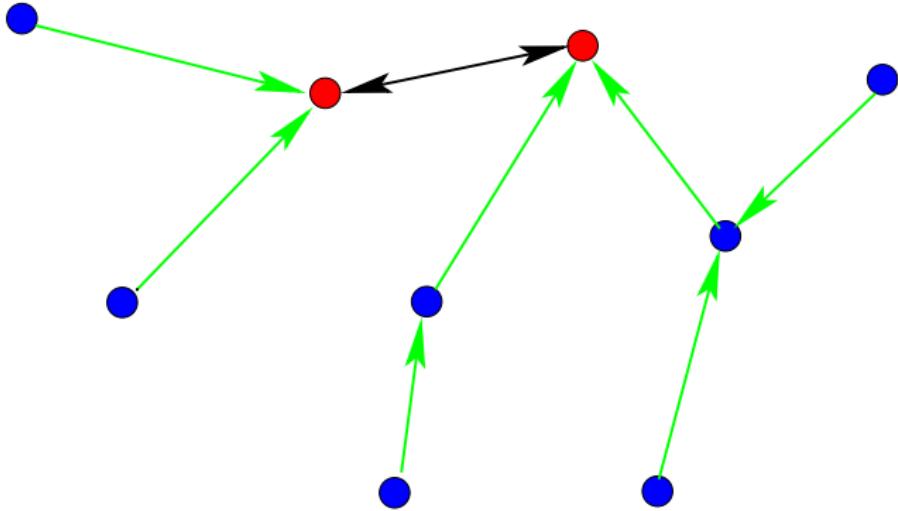


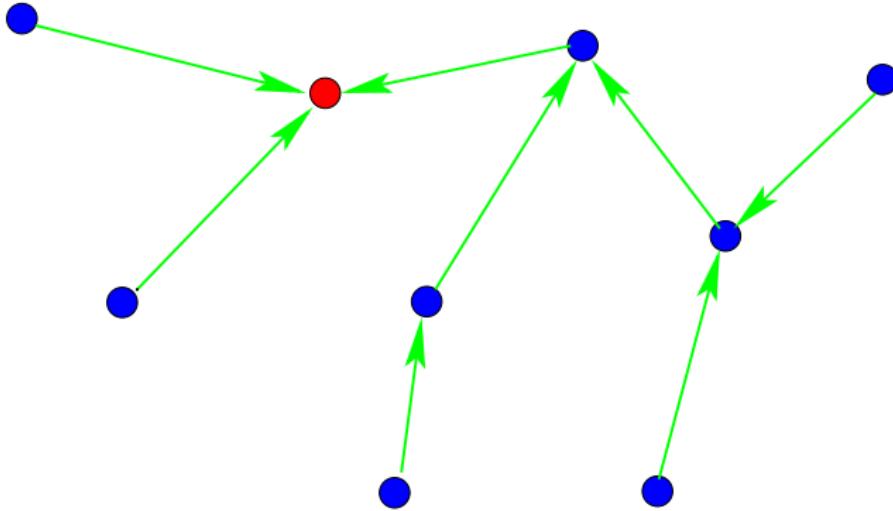
Quand un nœud rouge  $x$  est connecté à un autre nœud rouge  $y$  alors l'événement "progress" peut apparaître

```
EVENT progress  ≡  
ANY  $x, y$  WHERE  
     $x, y \in gr \wedge$   
     $x \notin \text{dom}(tr) \wedge$   
     $y \notin \text{dom}(tr) \wedge$   
     $gr[\{x\}] = tr^{-1}[\{x\}] \cup \{y\}$   
THEN  
     $tr := tr \cup \{x \mapsto y\}$   
END
```

Il faut prouver :

$$\begin{aligned} & \text{invariant}(tr) \quad \wedge \\ & x, y \in gr \quad \wedge \\ & x \notin tr \quad \wedge \\ & y \notin tr \quad \wedge \\ & gr[\{x\}] = tr^{-1}[\{x\}] \cup \{y\} \\ \implies & \text{invariant}(tr \cup \{x \mapsto y\}) \end{aligned}$$





Quand un **nœud rouge**  $x$  est **SEULEMENT**  
connecté aux **nœuds bleus** alors l'événement "elect"  
peut apparaître :

```
EVENT elect  ≡  
ANY  $x$  WHERE  
     $x \in ND \wedge$   
     $gr[\{x\}] = tr^{-1}[\{x\}]$   
THEN  
     $rt, ts := x, tr$   
END
```

EVENT elect  $\hat{=}$

**BEGIN**

$rt, ts : \text{spanning}(rt, ts, gr)$

**END**

EVENT elect  $\hat{=}$

**ANY**  $x$  **WHERE**

$x \in ND \wedge$

$gr[\{x\}] = tr^{-1}[\{x\}]$

**THEN**

$rt, ts := x, tr$

**END**

Il faut prouver :

$$\begin{aligned} & \text{invariant}(tr) \quad \wedge \\ & x \in ND \quad \wedge \\ & gr[\{x\}] = tr^{-1}[\{x\}] \\ & ts = tr \\ \implies & \text{spanning}(x, ts, gr) \end{aligned}$$

## Un lemme

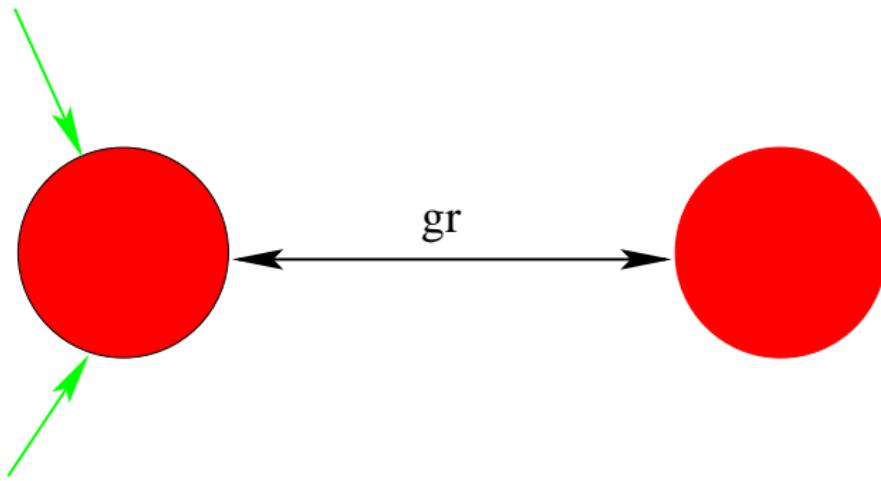
$$\begin{aligned} & \text{invariant}(tr) \quad \wedge \\ & x \in ND \quad \wedge \\ & gr[\{x\}] = tr^{-1}[\{x\}] \\ \implies & \\ & tr = fn(x) \end{aligned}$$

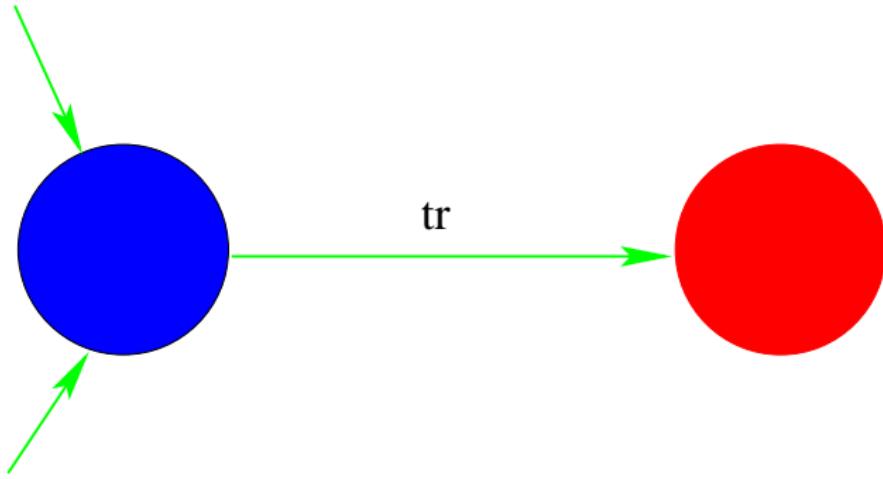
# Récapitulatif du premier raffinement

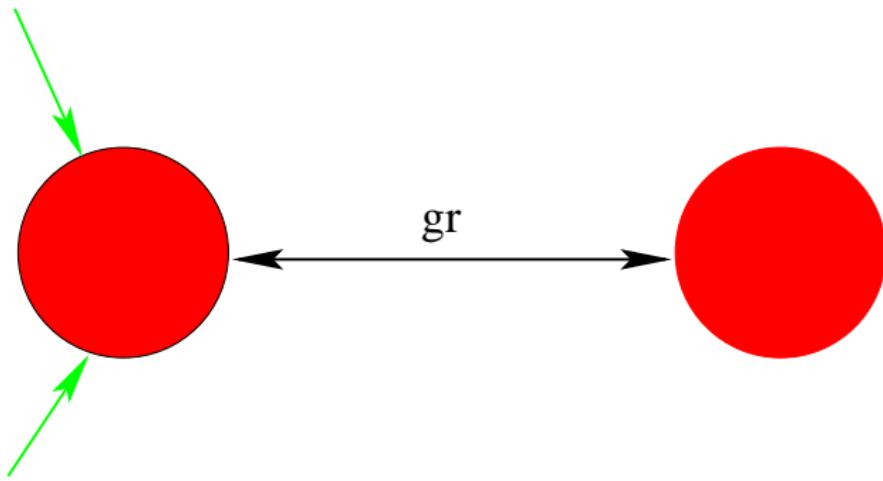
---

- 12 preuves
- Parmi lesquelles 5 interactives (une assez difficile !)

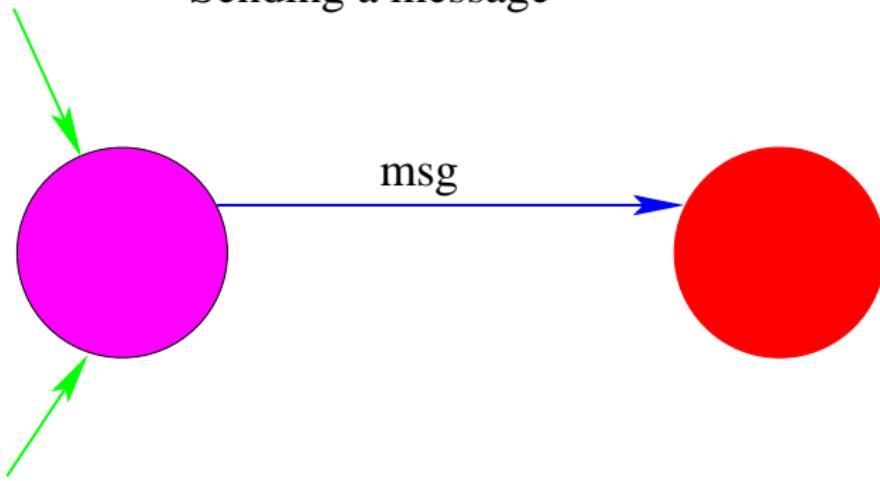
- Les nœuds **communiquent** avec leurs voisins
- On utilise des **messages**
- Les messages ont des **accusés de réception**
- Les accusés de réception sont **confirmés**
- Ensuite une animation **locale**





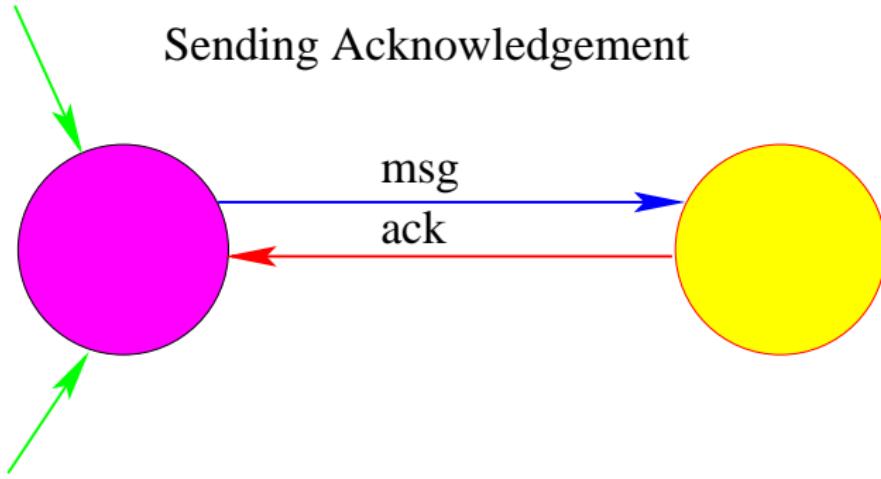


## Sending a message



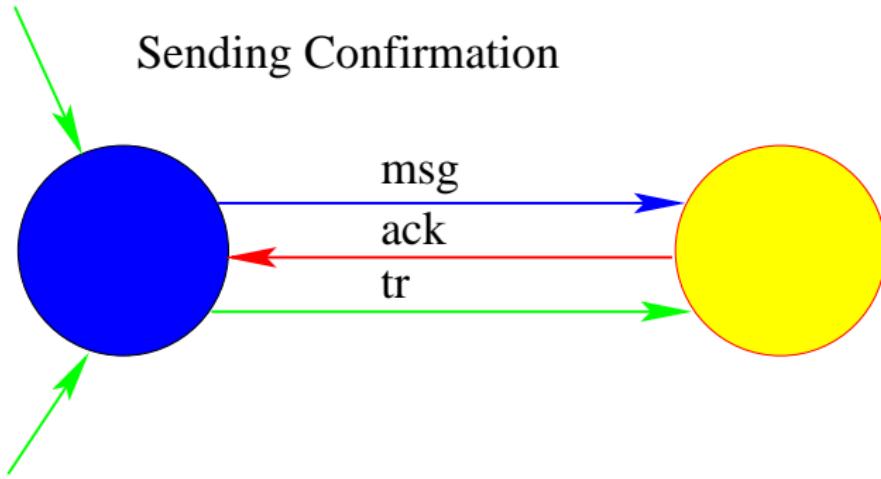
## Receiving a message

## Sending Acknowledgement

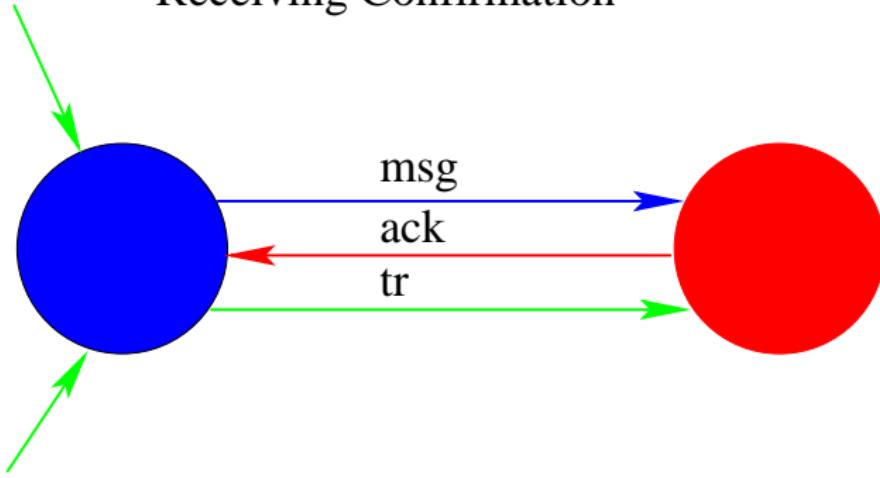


## Receiving Acknowledgement

## Sending Confirmation



## Receiving Confirmation



## Invariant (1)

$msg \in ND \rightarrow ND$

$ack \in ND \rightarrow ND$

$tr \subseteq ack \subseteq msg \subseteq gr$

Nœud  $x$  envoie un **message** au nœud  $y$

```
EVENT send_msg  ≡  
  ANY  $x, y$  WHERE  
     $x, y \in gr \wedge$   
     $gr[\{x\}] = tr^{-1}[\{x\}] \cup \{y\} \wedge$   
     $y, x \notin ack \wedge$   
     $x \notin \text{dom}(msg)$   
  THEN  
     $msg := msg \cup \{x \mapsto y\}$   
  END
```

Nœud  $y$  envoie un acknowledgement au nœud  $x$

```
EVENT send_ack  ≡  
  ANY x, y WHERE  
    x, y ∈ msg-ack ∧  
    y ∉ dom (msg)  
  THEN  
    ack := ack ∪ {x ↠ y}  
  END
```

Nœud  $x$  envoie une **confirmation** au nœud  $y$

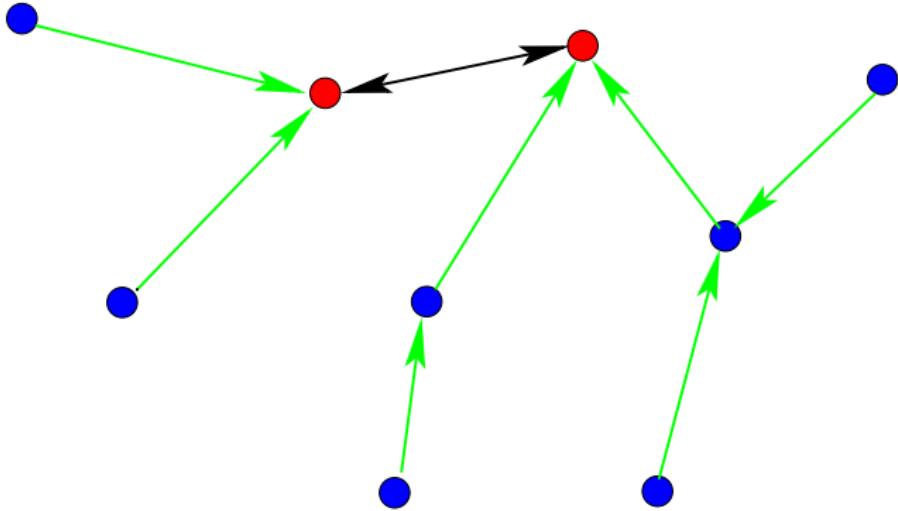
```
EVENT progress  $\hat{=}$ 
ANY  $x, y$  WHERE
     $x, y \in ack \wedge$ 
     $x \notin \text{dom}(tr)$ 
THEN
     $tr := tr \cup \{x \mapsto y\}$ 
END
```

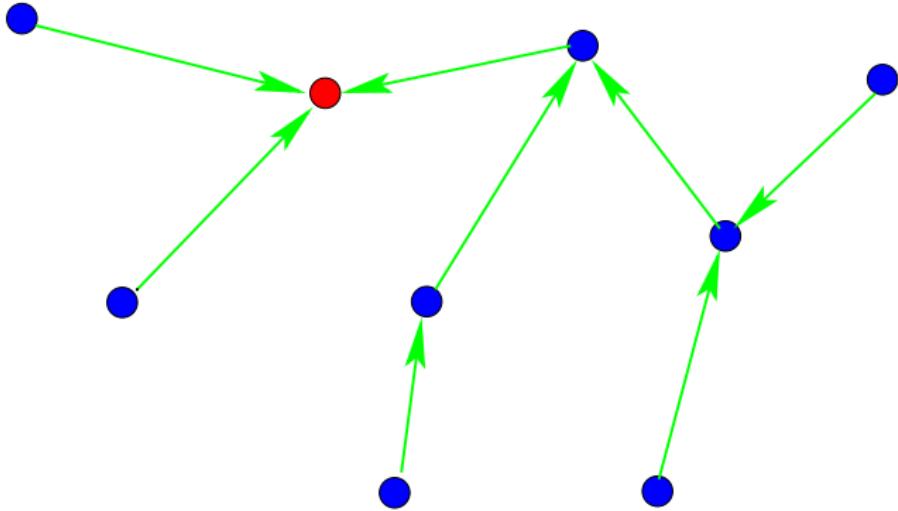
## Invariant (2)

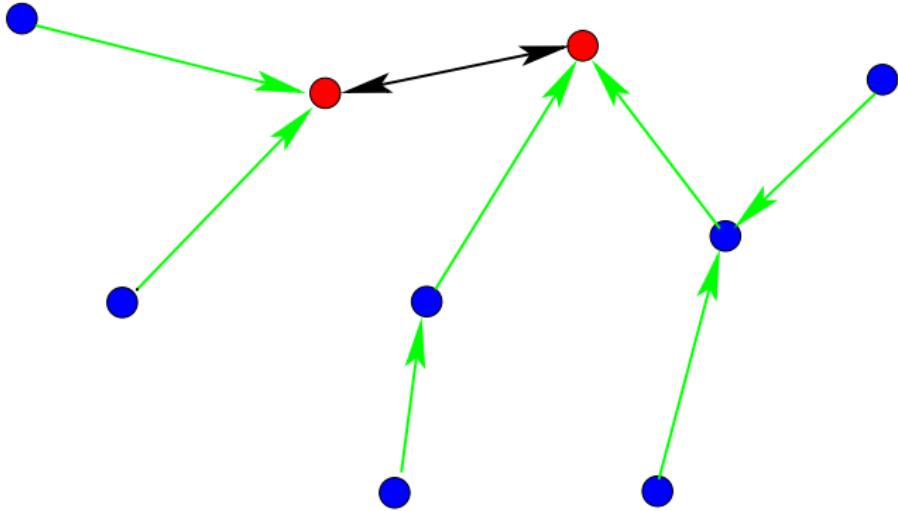
$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in msg-ack \\ \implies \\ x, y \in gr \quad \wedge \\ x \notin \text{dom}(tr) \wedge y \notin \text{dom}(tr) \wedge \\ gr[\{x\}] = tr^{-1}[\{x\}] \cup \{y\} \end{array} \right)$$

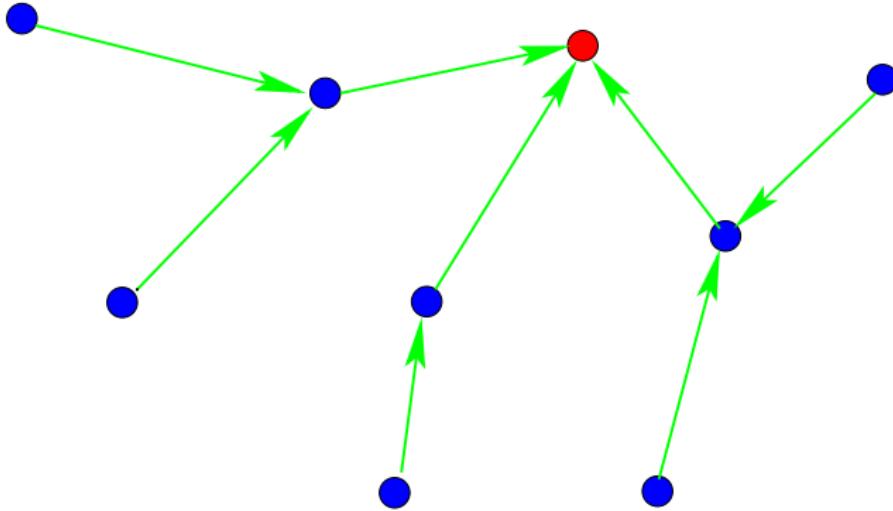
$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in ack \quad \wedge \\ x \notin \text{dom}(tr) \\ \implies \\ x, y \in gr \quad \wedge \\ y \notin \text{dom}(tr) \wedge \\ gr[\{x\}] = tr^{-1}[\{x\}] \cup \{y\} \end{array} \right)$$

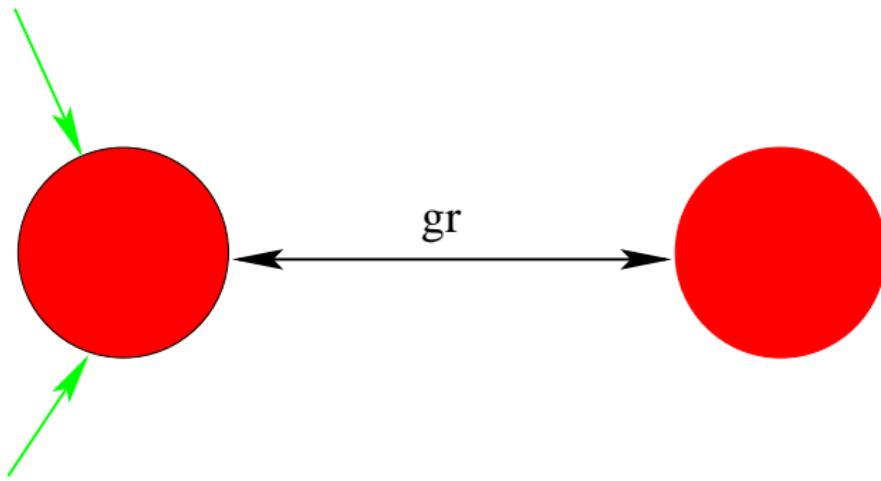
- Expliquer le **problème**
- Proposer une solution **partielle**.
- Vers un **meilleur** traitement
- Retour à l'animation **locale**.



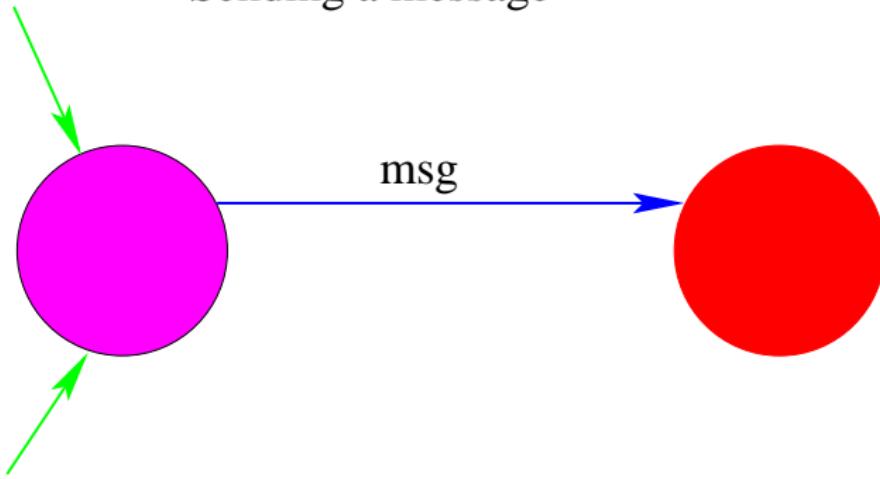




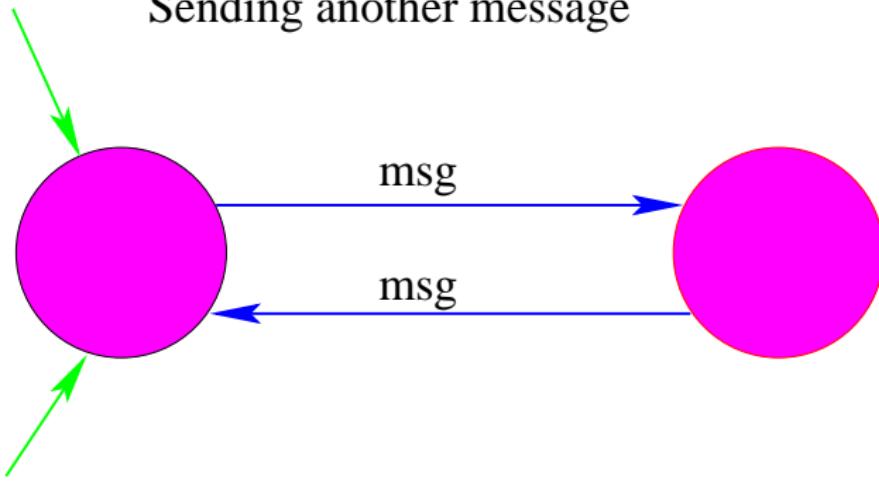




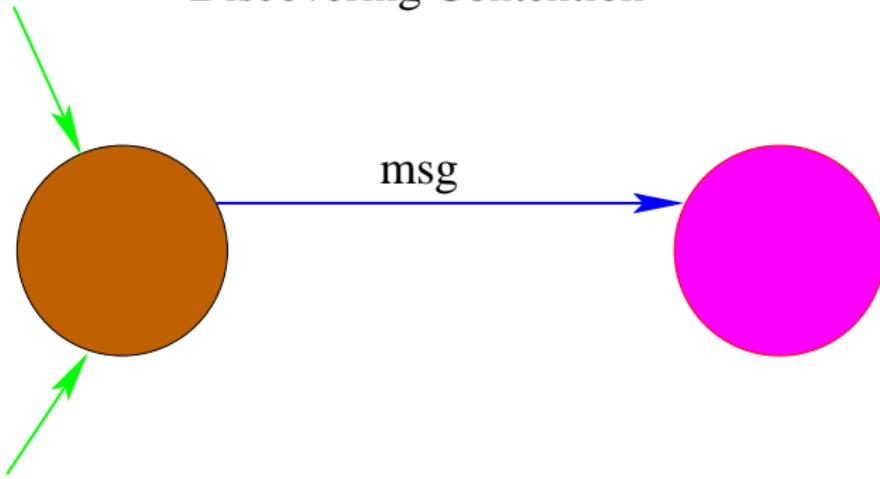
## Sending a message



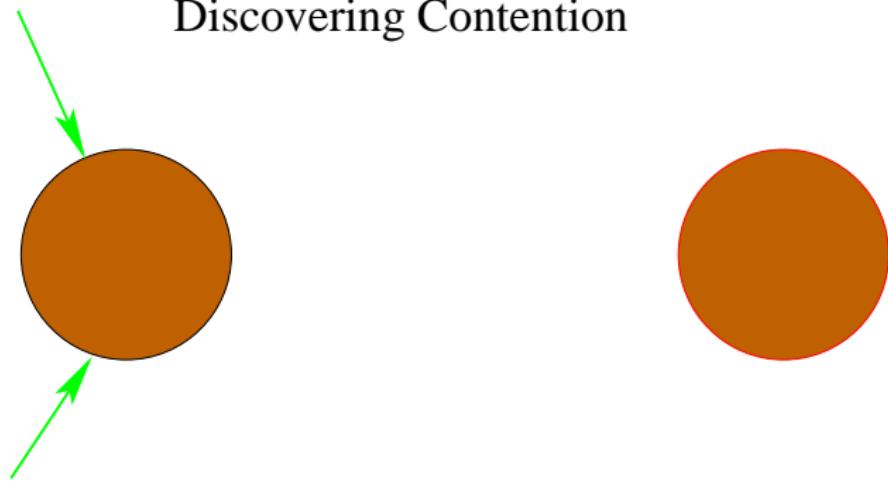
## Sending another message



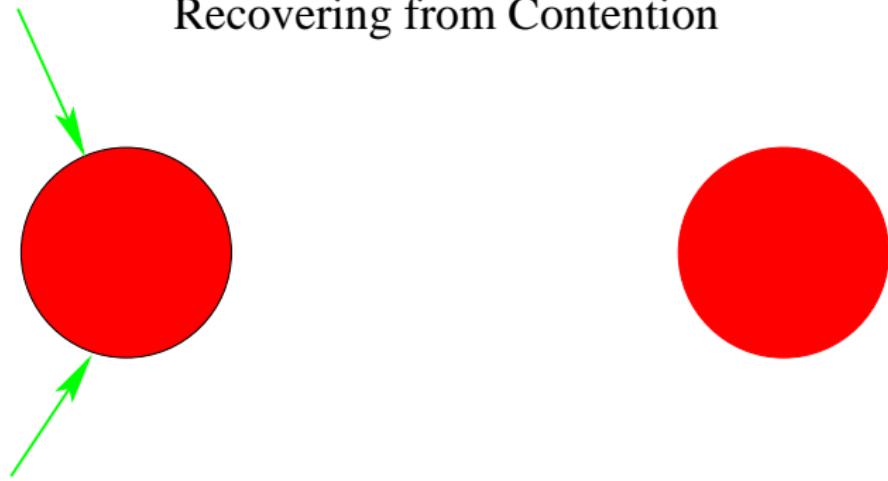
## Discovering Contention



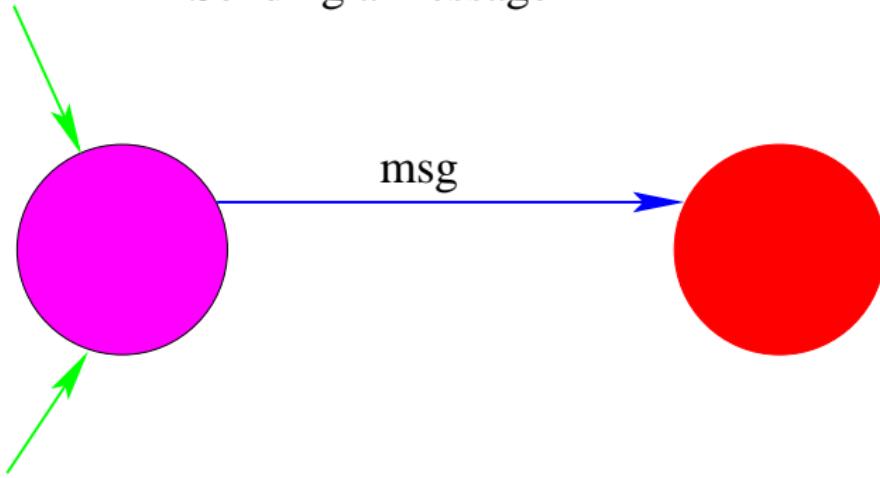
# Discovering Contention



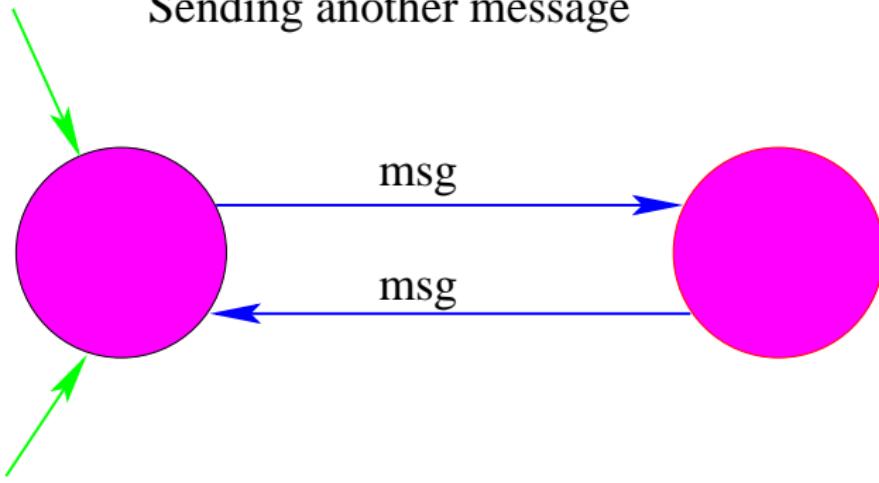
## Recovering from Contention



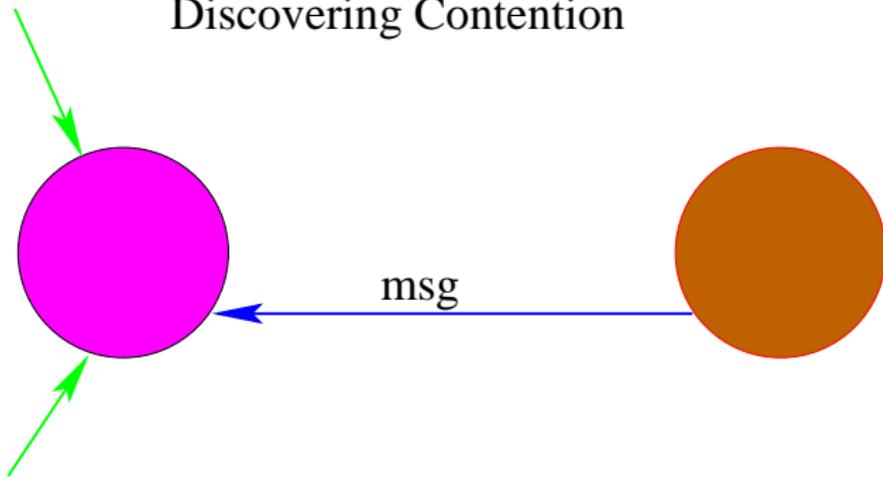
## Sending a message



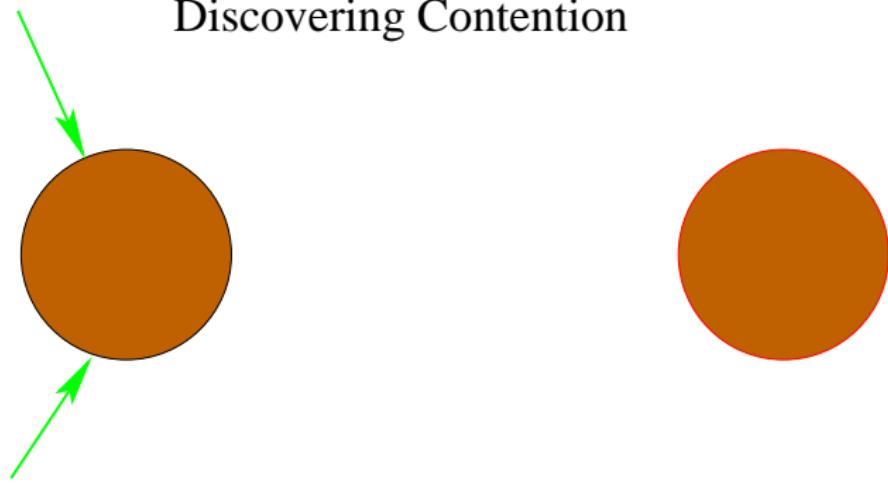
## Sending another message



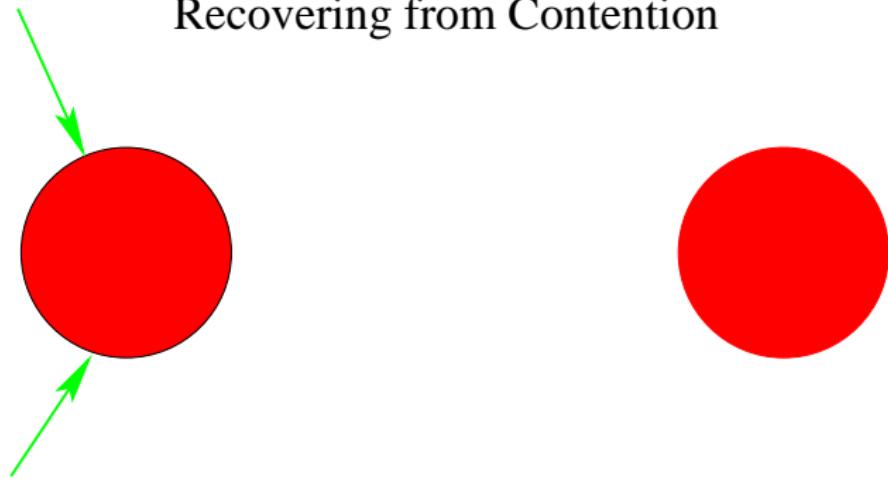
# Discovering Contention



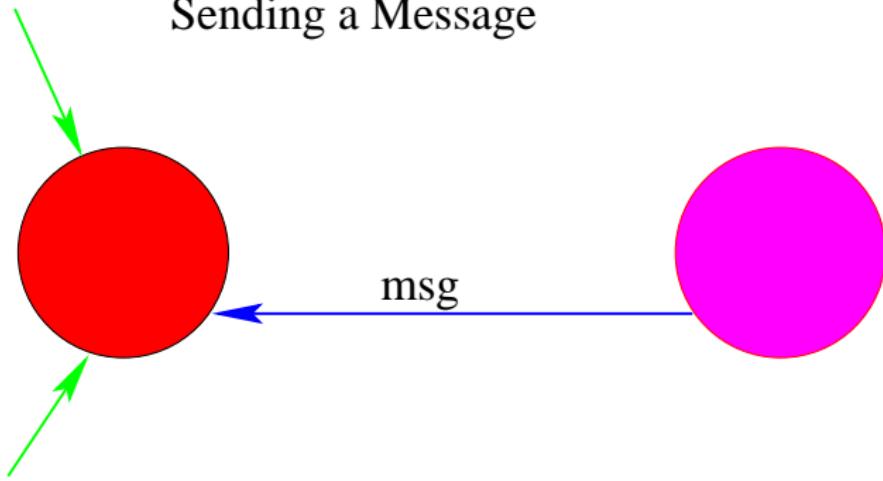
## Discovering Contention



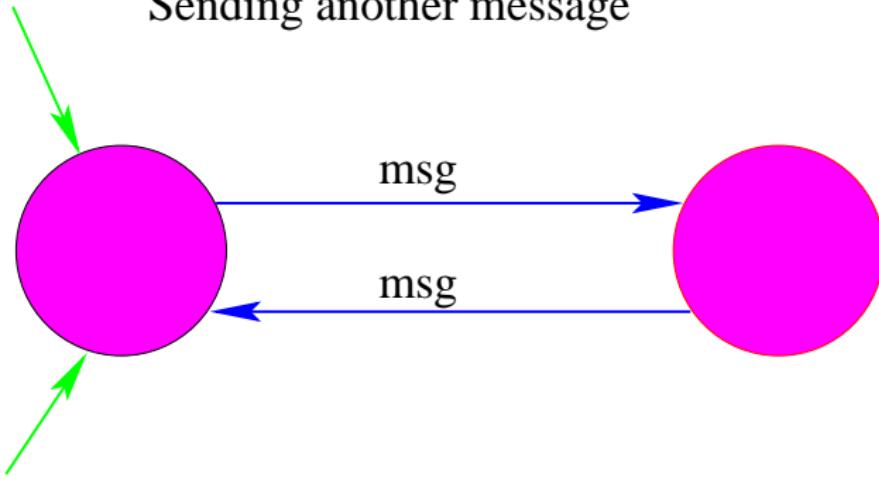
## Recovering from Contention



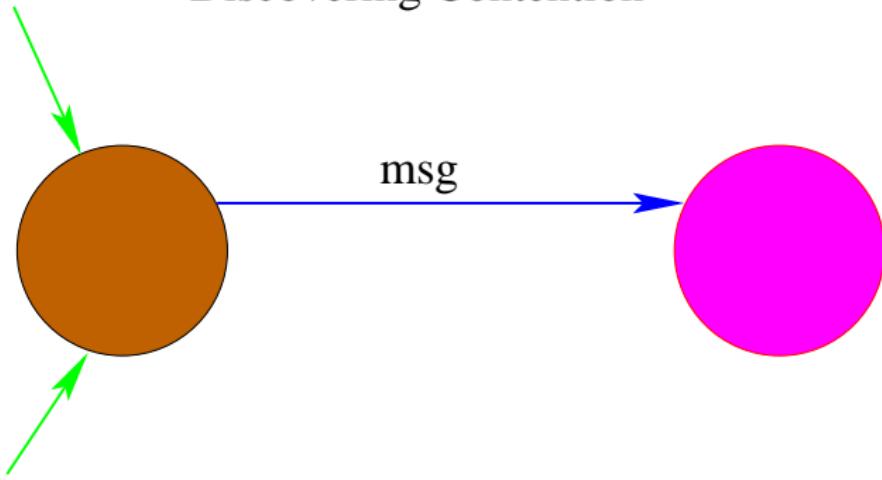
## Sending a Message



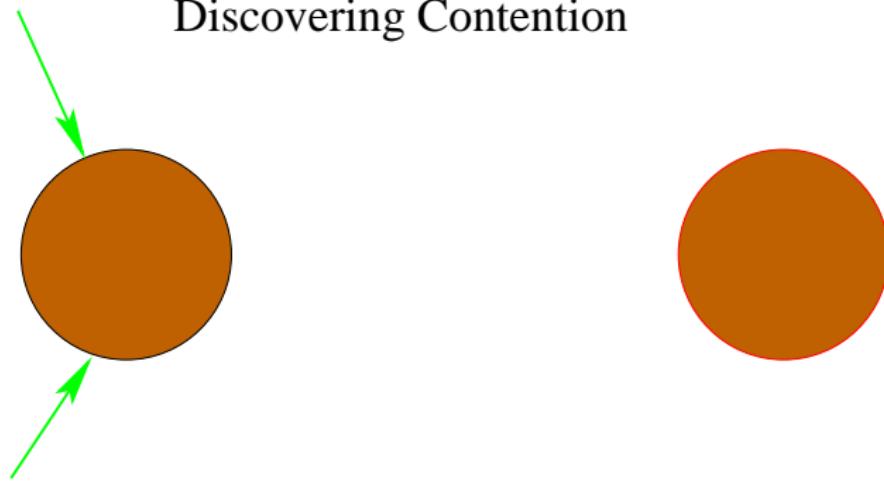
## Sending another message



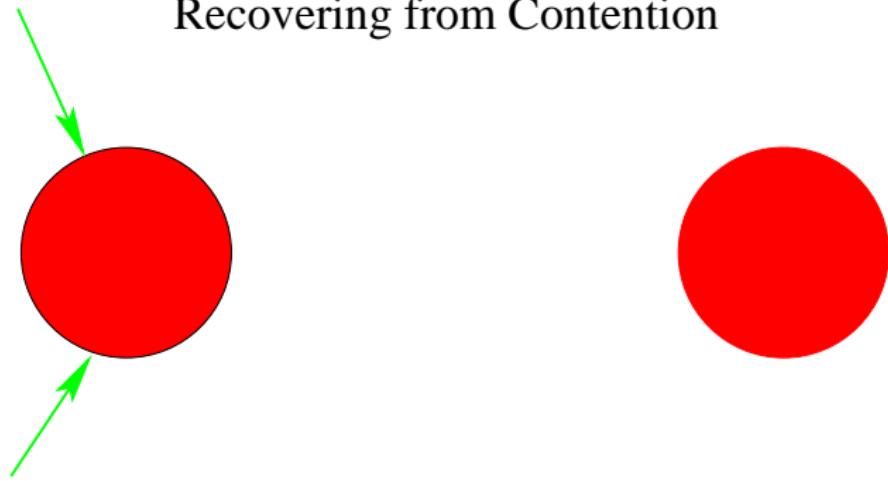
## Discovering Contention



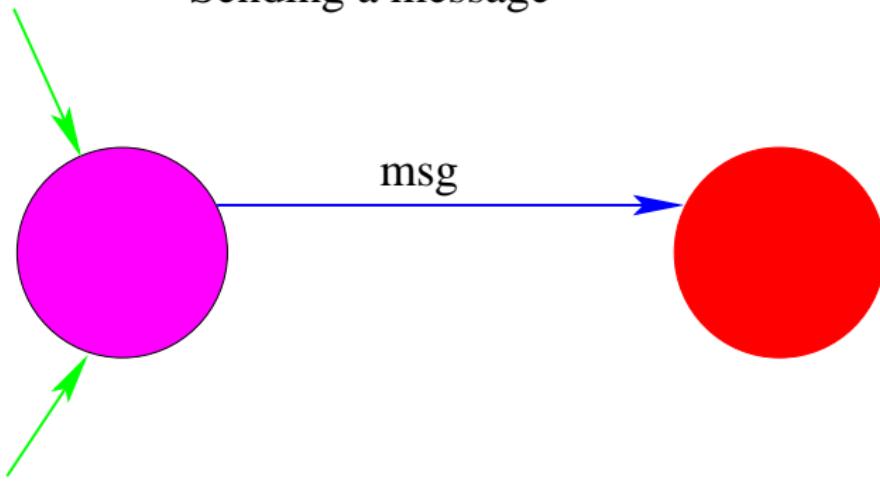
# Discovering Contention



## Recovering from Contention

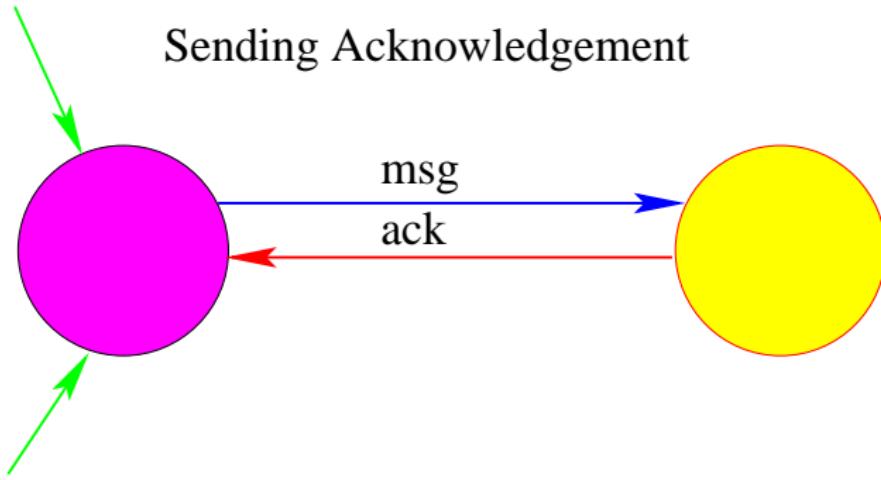


## Sending a message



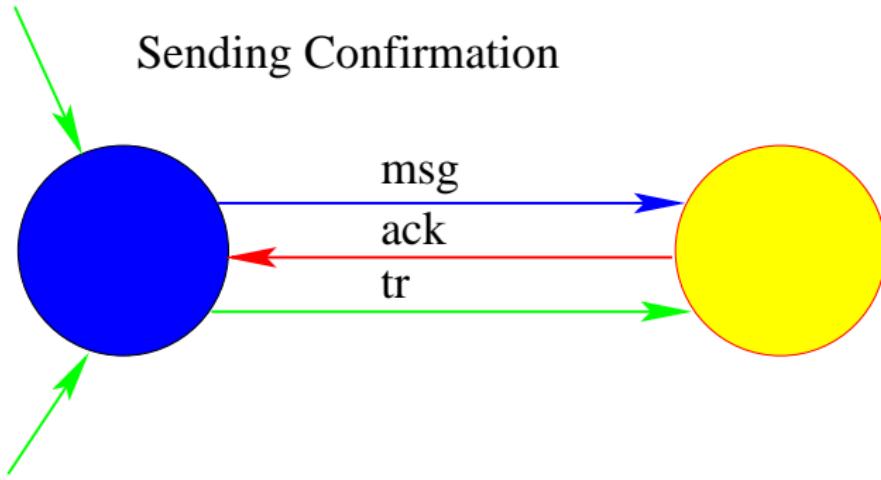
## Receiving a message

## Sending Acknowledgement

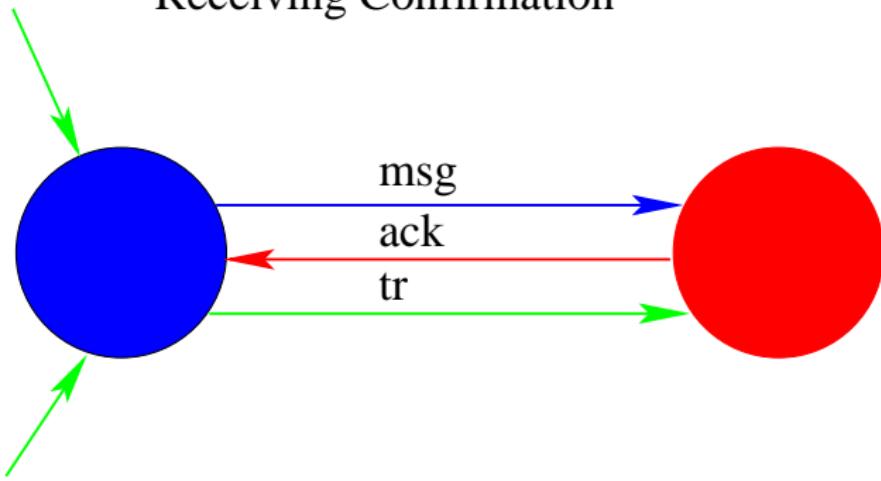


## Receiving Acknowledgement

## Sending Confirmation



## Receiving Confirmation



## Découverte de la contention (1)

---

- Nœud  $y$  découvre la contention avec  $x$  car :
  - Il a envoyé un message à  $x$
  - Il n'a pas encore reçu une réponse du nœud  $x$
  - Il reçoit à la place un message de  $x$

## Découverte de la contention (2)

---

- $x$  découvre aussi la contention avec  $y$
- Hypothèse : Le temps entre deux découvertes  
EST SUPPOSÉ BORNÉ  
PAR  $\tau$  ms
- Le temps  $\tau$  est le temps maximum de transmission  
entre 2 noeuds connectés

## Une solution partielle

---

- Chaque noeud attend  $\tau$  ms après sa découverte
- Après cela, chaque noeud sait que l'autre a aussi découvert la contention
- Chaque noeud essaie à nouveau immédiatement
- PROBLEME : Cela peut continuer indéfiniment

## Une meilleure solution (1)

---

- Tout nœud attend  $\tau$  ms après sa propre découverte
- Chaque nœud choisit avec équiprobabilité :
  - soit d'attendre un délai court
  - soit d'attendre un délai long
- Chaque nœud essaie à nouveau

EVENT send\_ack  $\hat{=}$   
**ANY**  $x, y$  **WHERE**  
 $x, y \in msg-ack \wedge$   
 $y \notin \text{dom}(msg)$   
**THEN**  
 $ack := ack \cup \{x \mapsto y\}$   
**END**

EVENT contention  $\hat{=}$   
**ANY**  $x, y$  **WHERE**  
 $x, y \in msg-ack \wedge$   
 $y \in \text{dom}(msg)$   
**THEN**  
 $cnt := cnt \cup \{x \mapsto y\}$   
**END**

## Invariant (3)

$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in msg\text{-}ack \wedge \\ y \in \text{dom}(msg) \\ \implies \\ y, x \in msg\text{-}ack \end{array} \right)$$

$$\forall (x, y, z) \cdot \left( \begin{array}{l} x, y \in msg \quad \wedge \\ z \in gr[\{x\}] \quad \wedge \\ z \neq y \\ \implies \\ z, x \in tr \end{array} \right)$$

## Invariant (4)

$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in msg\text{-}ack \quad \wedge \\ y \notin \text{dom}(msg) \\ \implies \\ x, y \notin cnt \end{array} \right)$$

$$ack \cap ack^{-1} = \emptyset$$

$$ack \cap cnt = \emptyset$$

# Résolution de la contention (simuler le délai $\tau$ )

```
EVENT solve_contention  ≡  
  ANY  $x, y$  WHERE  
     $x, y \in cnt \cup cnt^{-1}$   
  THEN  
     $msg := msg - cnt \quad \|$   
     $cnt := \emptyset$   
  END
```

# Récapitulatif du Second Raffinement

---

- 63 preuves
- Parmi lesquelles 31 interactives

- Etablir le **cadre mathématique**
- Résoudre le problème mathématique en un coup.
- Résoudre le même problème progressivement.
- Inclure les communications par des messages.
- Localiser les structures de donnée
- Méthodologie fondée sur le raffinement et la preuve
- Autres développements : algorithmique répartie, algorithmique séquentielle, systèmes, SoCs, ...

## The predicate **invariant** (*tr*)

$tr \in ND \rightarrow ND$

## The predicate **invariant** ( $tr$ )

$tr \in ND \rightarrow ND$

$$\text{dom } (tr) \lhd (tr \cup tr^{-1}) = \text{dom } (tr) \lhd gr$$

## Invariant (3)

$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in msg \quad \wedge \\ y, x \in msg \\ \implies y, x \notin ack \end{array} \right)$$

$$\forall (x, y, z) \cdot \left( \begin{array}{l} x, y \in msg \quad \wedge \\ z \in gr[\{x\}] \quad \wedge \\ z \neq y \\ \implies z, x \in tr \end{array} \right)$$

## Invariant (4)

$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in cnt \\ \implies \\ x, y \in msg\text{-}ack \quad \wedge \\ y \in \text{dom}(msg) \end{array} \right)$$

$$ack \cap ack^{-1} = \emptyset$$

$$ack \cap cnt = \emptyset$$

## Third Refinement : Localization

---

- The representation of the **graph**  $gr$  is **modified**
- The representation of the **tree**  $tr$  is **modified**
- Other data structures are **localized**

## Localization (1)

---

The graph  $gr$  and the tree  $tr$  are now **localized**

$$nb \in ND \rightarrow \mathbb{P}(ND)$$

$$\forall x \cdot (x \in ND \implies nb(x) = gr[\{x\}])$$

$$sn \in ND \rightarrow \mathbb{P}(ND)$$

$$\forall x \cdot (x \in ND \implies sn(x) \subseteq tr^{-1}[\{x\}])$$

## Localization (2)

---

$$\textcolor{red}{bm} \subseteq ND$$

$$bm = \text{dom}(msg)$$

$$\textcolor{red}{bt} \subseteq ND$$

$$bt = \text{dom}(tr)$$

$$\textcolor{red}{ba} \in ND \rightarrow \mathbb{P}(ND)$$

$$\forall x \cdot (x \in ND \implies ba(x) = ack^{-1}[\{x\}])$$

- Node  $x$  is elected the leader

```
EVENT elect  ≡  
ANY x WHERE  
   $x \in ND \wedge$   
   $nb(x) = sn(x)$   
THEN  
   $rt := x$   
END
```

- Node  $x$  sends a message to node  $y$  ( $y$  is unique)

```
EVENT send_msg  ≡  
  ANY  $x, y$  WHERE  
     $x \in ND - bm \wedge$   
     $y \in ND - ba(x) \wedge$   
     $nb(x) = sn(x) \cup \{y\}$   
  THEN  
     $msg := msg \cup \{x \mapsto y\} \quad ||$   
     $bm := bm \cup \{x\}$   
  END
```

- Node  $y$  sends an acknowledgement to node  $x$

EVENT send\_ack  $\hat{=}$

**ANY**  $x, y$  **WHERE**

$x, y \in msg \wedge$

$x \notin ba(y) \wedge$

$y \notin bm$

**THEN**

$ack := ack \cup \{x \mapsto y\} \quad ||$

$ba(y) := ba(y) \cup \{x\}$

**END**

- Node  $x$  sends a confirmation to node  $y$

EVENT progress  $\hat{=}$

**ANY**  $x, y$  **WHERE**

$x, y \in ack \wedge$

$x \notin bt$

**THEN**

$tr := tr \cup \{x \mapsto y\} \quad ||$

$bt := bt \cup \{x\}$

**END**

- Node  $y$  receives confirmation from node  $x$

```
EVENT rcv_cnf  ≡  
  ANY x, y WHERE  
    x, y ∈ tr ∧  
    x ∉ sn(y)  
  THEN  
    sn(y) := sn(y) ∪ {x}  
  END
```

EVENT contention  $\hat{=}$

**ANY**  $x, y$  **WHERE**

$x, y \in cnt \cup cnt^{-1} \wedge$

$x \notin ba(y) \wedge$

$y \in bm$

**THEN**

$cnt := cnt \cup \{x \mapsto y\}$

**END**

EVENT solve\_contention  $\triangleq$   
**ANY**  $x, y$  **WHERE**  
 $x, y \in cnt \cup cnt^{-1}$   
**THEN**  
 $msg := msg - cnt \quad \|$   
 $bm := bm - \text{dom}(cnt) \quad \|$   
 $cnt := \emptyset$   
**END**

# Section Courante

---

## ① Election

Problème de l'élection

]

## ② Election

Problème de l'élection

Election dans un graphe  
acyclique

## ③ Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## ④ Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et  
Lamport

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

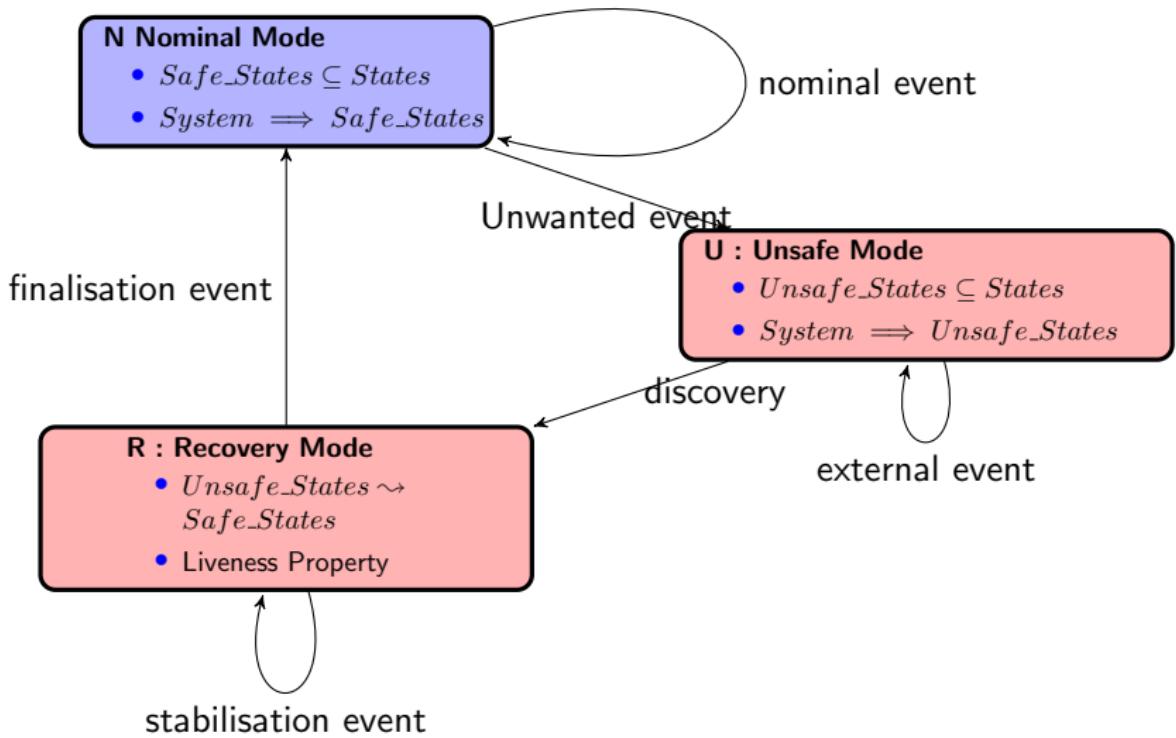
## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

## Auto-stabilisation dans les systèmes répartis



- Les systèmes peuvent être victimes de fautes transitoires
- Un certain nombre d'algorithmes permettent d'anticiper de tels problèmes.
- Les algorithmes auto-stabilisants constituent une alternative à la perte de l'état consistant d'un système donné à la suite d'une défaillance.
- Différents phénomènes peuvent apparaître lors de l'exécution d'un algorithme : en particulier peuvent se produire des changements dans le réseau (ajout ou disparition d'un sommet ou d'un lien) ou bien encore des altérations de messages ou de mémoires.
- Un algorithme est dit auto-stabilisant s'il se termine correctement en dépit de l'apparition de ces phénomènes.
- Un algorithme auto-stabilisant ne nécessite pas d'initialisation particulière.

- Les systèmes répartis sont exposés à des risques de défaillances transitoires qui peuvent placer un système donné dans un état ou une configuration arbitraire.
- Cette configuration arbitraire peut être une configuration ne satisfaisant plus l'invariant du système.
- Dans ce cas, la question est de concevoir un algorithme réparti permettant de faire converger le système global d'une configuration arbitraire vers une configuration dite stable et satisfaisant l'invariant.

## ① Election

Problème de l'élection

## ② Election

Problème de l'élection

Election dans un graphe acyclique

## ③ Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## ④ Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

# Modélisation d'un système auto-stabilisant

---

- Soit un système réparti  $\mathcal{S}$  modélisé par un système de transition  $(States, \rightarrow)$  où  $States$  est un ensemble d'états ou de configurations et où  $\rightarrow$  modélise la relation de transition simulant l'activité du système.
- Une exécution ou un calcul de  $\mathcal{S}$  est une suite maximale  $(\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_i, \dots)$  engendrée par  $(States, \rightarrow)$ .
- Tout préfixe d'une telle suite est un calcul puisqu'il n'y a pas d'états initiaux déterminés.
- Tout préfixe d'exécution est donc un calcul ou une exécution de  $\mathcal{S}$  (pas de dépendance d'un état initial)

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

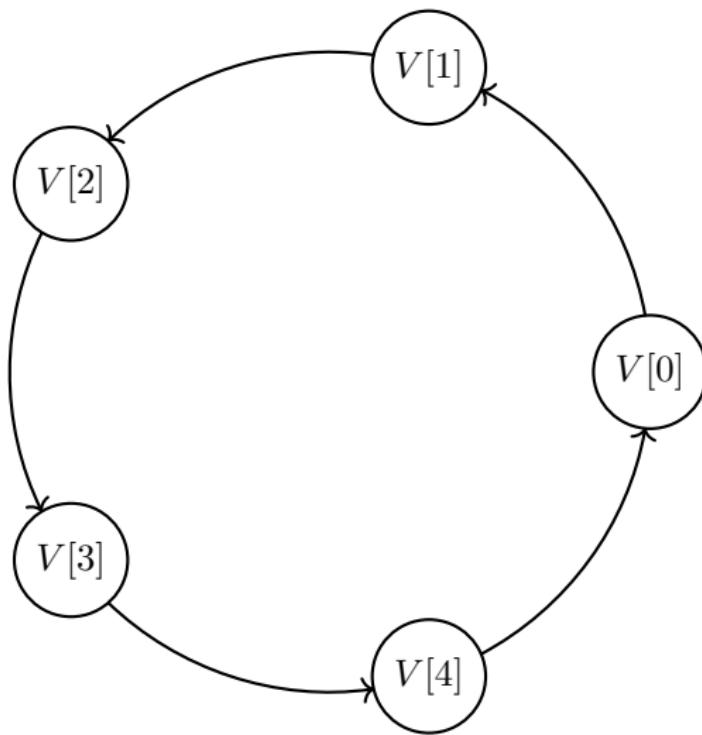
Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

## Réseau en anneau

---



## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

**Algorithme de Dijkstra**

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

# Problème de l'exclusion mutuelle

## Description

- Dans toute configuration, au plus un processus a le privilège.
  - Chaque processus a le privilège infiniment souvent.
- 
- Dans un réseau avec une topologie en anneau, des processus ont accès à une ressource.
  - Le privilège est accordé à un processus à la fois.
  - Le problème est que le privilège est géré par un jeton qui peut être perdu et donc être dans une configuration ne satisfaisant plus la propriété.

# Algorithme de Dijkstra

DOMAINE  $\triangleq 0..N$

IMAGE  $\triangleq 0..M$

NToZERO  $\triangleq$

$\wedge \quad V[0] = V[N]$

$\wedge \quad V' = [V \text{ EXCEPT } ![0] = (V[0]+1)\%(M+1)]$

OTHERS(I)  $\triangleq$

$\wedge \quad I \in 0..N$

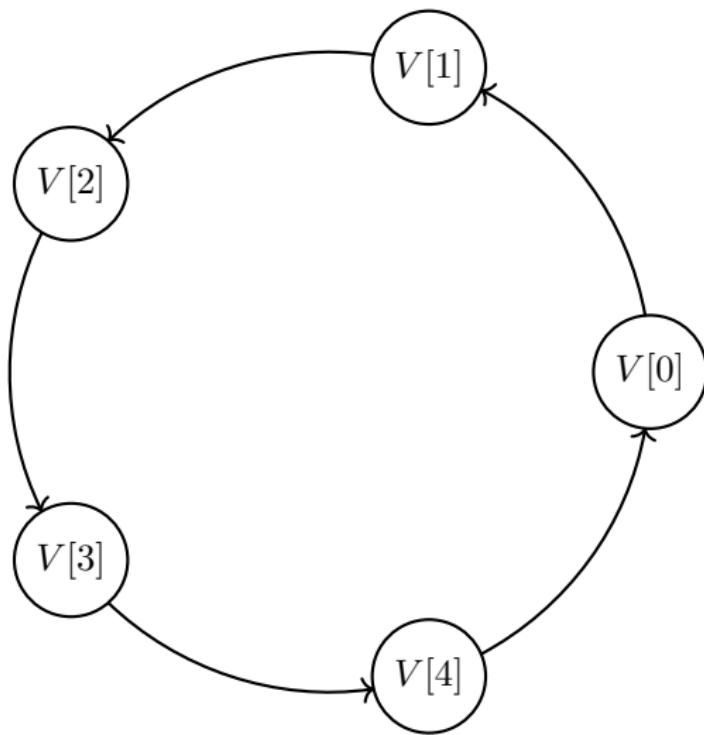
$\wedge \quad I \neq N$

$\wedge \quad V[I+1] \neq V[I]$

$\wedge \quad V' = [V \text{ EXCEPT } ![I+1] = V[I]]$

## Réseau en anneau

---



# Propriétés de stabilisation

---

---

$$\begin{aligned} \text{Init} &= V = [i \in 0..N \rightarrow (\text{IF } i \neq N \text{ THEN } i \text{ ELSE } 0)] \\ \text{Next} &= \text{NToZero} \vee (\exists I \in 0..N-1: \text{Others}(I)) \end{aligned}$$
$$\begin{aligned} \text{Prop1} &= (V[0] = V[N]) \\ &\Rightarrow (\forall i \in 1..N-1: V[i+1] = V[i]) \end{aligned}$$
$$\begin{aligned} \text{Prop2} &= \\ &\vee (\exists i, j \in 0..N-1: i \neq j \\ &\quad \wedge V[i+1] \neq V[i] \wedge V[j+1] \neq V[j]) \\ &\vee (V[0] = V[N] \wedge \\ &\quad (\exists i \in 0..N-1: V[i+1] \neq V[i])) \end{aligned}$$

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

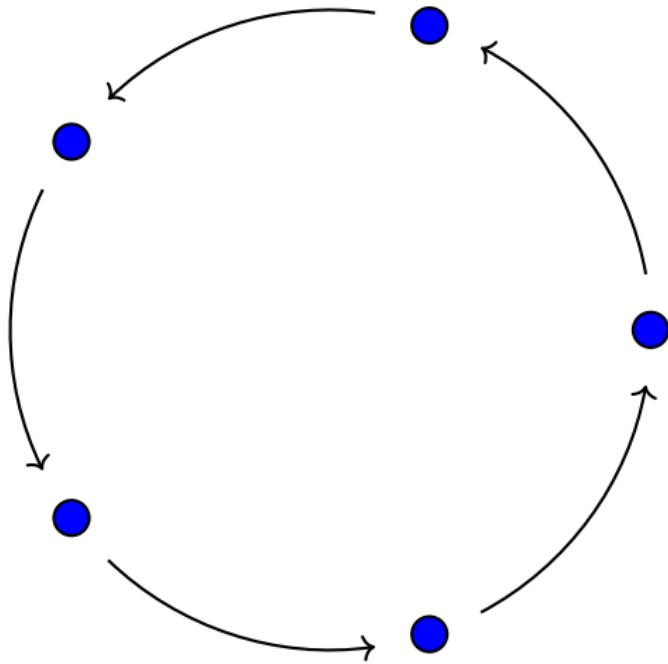
# Coloriage d'un anneau

---

Le problème du coloriage d'un anneau concerne l'existence d'une configuration dans laquelle trois nœuds consécutifs n'ont pas la même couleur et pose la question de définir un algorithme réparti permettant de l'atteindre à partir de toute configuration initiale de l'anneau.

- Modéliser un processus de coloration d'un anneau à partir d'une situation quelconque initiale pour atteindre une situation stable caractérisée par le fait que deux nœuds voisins n'ont pas la même couleur.
- La règle de changement de couleur : Si un nœud a la même couleur que l'un de ses voisins, il choisit une autre couleur que celle de ses voisins.
- La règle est applicable tant que deux nœuds ont la même couleur.
- La règle est non-applicable dès que les nœuds adjacents ont tous une couleur différente : le cas d'un anneau à un nœud est exclus en considérant qu'il y a au moins deux nœuds dans l'anneau.

## Exemple d'anneau



**CONTEXT**  $ring$

**SETS**

$N$

**CONSTANTS**

$n$

**AXIOMS**

$axm1 : n \in N \rightarrow N$

$axm2 : \forall s \cdot s \subseteq n[s] \Rightarrow N \subseteq s$

$axm3 : \text{finite}(N)$

**END**

**MACHINE** *one-shot*

**SEES**  $ring, color$

**VARIABLES**

$col$

**INVARIANTS**

$inv1 : col \in N \rightarrow color$

**EVENTS**

**EVENT INITIALISATION**

$act1 : col : \in N \rightarrow color$

**EVENT stable**

**ANY**

$f$

**WHERE**

$grd1 : f \in N \rightarrow color$

$grd2 : \forall x \cdot f(x) \notin f[(n \cup n^{-1})[\{x\}]]$

**THEN**

$act1 : col := f$

**END**

Le modèle définit la solution à trouver par un événement abstrait exprimant le lien entre la configuration initiale et une configuration stable. Les règles données explique comment le calcul est effectué par les différents processus du réseau. Chaque règle suppose que leur application est possible quand trois noeuds consécutifs vérifient une condition donnée.

Chaque règle constitue un élément de calcul réparti et cet élément est appliqué sur la *boule* centrée sur un nœud donné et tient compte des couleurs des nœuds périphériques.

**MACHINE** *rule* **REFINES** *one-shot*

**SEES** *ring, color*

**VARIABLES** *col, c*

**INVARIANTS**

*inv1 :*  $c \in N \rightarrow color$

**THEOREMS**

*thm1 :*

$$\left( \begin{array}{l} \forall x \cdot c(x) \neq c(n(x)) \\ \vee \\ \exists x, clr \cdot c(x) = c(n(x)) \wedge clr \neq c(x) \wedge clr \neq c(n(x)) \end{array} \right)$$

## EVENTS

### EVENT INITIALISATION

$$act1 : col, c : \mid \left( \begin{array}{l} col' \in N \rightarrow color \\ \wedge \\ c' \in N \rightarrow color \\ \wedge \\ col' = c' \end{array} \right)$$

EVENT stable REFINES stable

### WHEN

$$grd1 : \forall x \cdot c(x) \neq c(n(x))$$

### WITNESSES

$$f : f = c$$

### THEN

$$act1 : col := c$$

### END

EVENT rule

**STATUS convergent**

**ANY**

$x, clr$

**WHERE**

$grd1 : c(x) = c(n(x)) \vee c(x) = c(n^{-1}(x))$

$grd2 : clr \neq c(n(x))$

$grd3 : clr \neq c(n^{-1}(x))$

**THEN**

$act1 : c(x) := clr$

**END**

La terminaison de ce processus dépend de la condition appelée variant et qui doit décroître par application des règles.

**VARIANT**  $\text{card}(\{x|c(x) = c(n(x))\})$

Les graphes VISIDIA sont symétriques ; le graphe  $g$  est défini à partir du graphe  $n$  et la règle *rule* est raffinée en *visidia-rule*

**CONTEXT** *ringgr*

**EXTENDS** *ring1*

**CONSTANTS**

$g$

**AXIOMS**

$axm1 : g = n \cup n^{-1}$

**END**

## EVENT INITIALISATION

$act1 : col, c : |(col' \in N \rightarrow color \wedge c' \in N \rightarrow color \wedge col' = c')$

EVENT stable REFINES stable

**WHEN**

$grd1 : \forall x \cdot c(x) \notin c[g[\{x\}]]$

**THEN**

$act1 : col := c$

**END**

EVENT rule

**STATUS convergent**

REFINES rule

**ANY**

$x, clr$

**WHERE**

$grd1 : c(x) \in c[g[\{x\}]]$

$grd2 : clr \notin c[g[\{x\}]]$

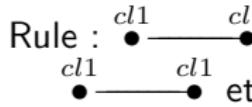
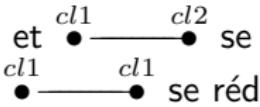
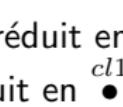
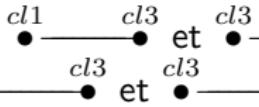
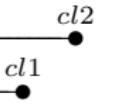
**THEN**

$act1 : c(x) := clr$

**END**

# Dérivation d'un programme VISIDIA

---

Rule :  et  se réduit en  et  se réduit en 

# Sommaire sur l'autostabilisation

---

- Définition d'algorithmes indépendants des états initiaux
- Algorithmes très complexes à vérifier en particulier *la terminaison*
- Version autostabilisante de beaucoup d'algorithmes

# Terminaison de l'algorithme

---

- stabilité : un seul processus a la main ou un seul  $i \in 0..N \wedge v[i] = v[i+1]$ .
- propriété 1 : il y a au moins un processus avec une garde franchissable :
  - ▶ tous les processus de 1 à N ont la même valeur (hypothèse)
  - ▶ nécessairement 0 et N ont la même valeur et 0 a la main.
- propriété 2 : toute configuration légale satisfait la propriété de clôture :
  - ▶ les processus de 0 à i-1 ont la même valeur
  - ▶ les processus de i à N ont la même valeur
  - ▶  $v[i] \neq v[i-1]$
- propriété 3 : à partir de toute configuration illégale, l'anneau atteint une configuration légale (le nombre de processus ayant la main ne peut pas croître et au mieux il peut stagner)
  - ▶  $0 \dots i-1$  : non franchissable
  - ▶ chaque processus maintient le nombre de processus franchissables : on construit à partir de p, une suite  $w_1 \dots w_N$  différent deux à deux
  - ▶ cette suite ne peut pas rester indéfiniment stable pr propagation de la valeur et respect de la constnace.

# Conclusion

---

- Algorithmes répartis : problème de l'expression locale de la globalité
- Algorithmes très complexes à vérifier
- Prise en compte de nombreux aspects de l'environnement comme les fautes, les erreurs, ...

# Section Courante

---

## ① Election

Problème de l'élection

]

## ② Election

Problème de l'élection

Election dans un graphe  
acyclique

## ③ Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## ④ Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et  
Lamport

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

- un programme réparti est constitué d'un ensemble de processus s'exécutant en parallèle et communiquant seulement par envoi de messages via des canaux de communication.
- Il n'y a pas d'horloge globale commune pour ces processus
- Les temps de transfert des messages ne sont pas bornés : sous hypothèse de communication fiable, ces temps sont finis. n a
- **Conséquence** : pas d'observation *simultanée* globale possible de l'état
- **Problème** : déterminer un état global consistant
- Le **snapshot** est une copie ou une photographie instantanée de l'état d'un système.

# Exemples

---

- Calcul de la topologie d'un réseau
- Compter le nombre de processus dans un système réparti
- Déetecter la terminaison
- Déetecter le blocage
- Déetecter la perte de coordination

# Reconstituer l'état global

---

- Les algorithmes de snapshots permettent de donner des clichés répartis de l'état global
- Comment rassembler ces *clichés répartis* ou *snapshots* afin de donner une vue consistante et globale à chaque processus du système ?
- Proposer des algorithmes rassemblant les *fragments* en un état global.

# Applications de ce problème

---

- *détection du blocage* : tout processus bloqué momentanément souhaite connaitre l'état global du système, afin de savoir s'il a atteint une configuration de blocage.
- *détection de la terminaison* : dans le cas de calculs répartis, les processus exécutent des actions au cours de phases de calculs mais le passage à une phase suivante peut exiger que le processus en cours obtienne la connaissance de l'état des autres processus : nécessité de connaître l'état global.
- *détection de la conservation de jetons dans un réseau*
- *réinitialisation du réseau* : un système réparti peut entrer dans une configuration nécessitant de relancer ou de réinitialiser l'état global courant et cette relance peut être faite à partir des snapshots conservés qui vont donc constituer des points de reprise (*checkpoints*)

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

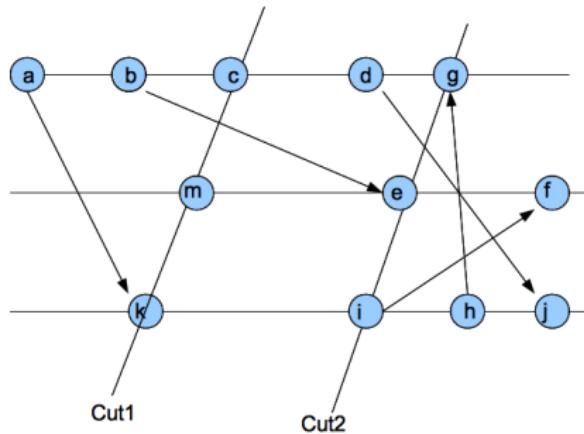
# Modélisation d'un système réparti

---

- Un système réparti est modélisé comme un ensemble fini de processus séquentiels
- Chaque processus  $i$  est caractérisé par un état local noté  $LS_i$
- Chaque canal de communication  $C_{ij}$  a un état noté  $SC_{ij}$ .
- Chaque processus séquentiel peut réaliser trois types d'actions :
  - ▶ Une action interne modifiant l'état local
  - ▶ Une envoi sur un canal
  - ▶ Une réception sur un canal

# Coupures

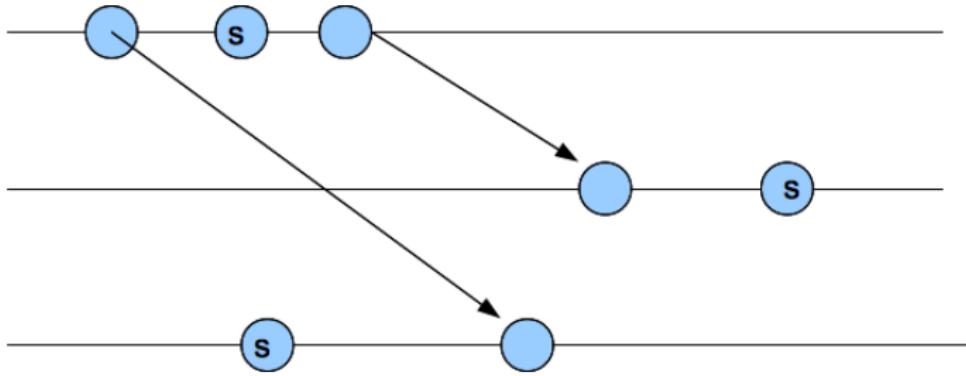
---



- $Cut1 = \{a, b, c, m, k\}$
- $Cut2 = \{a, b, c, d, g, m, e, k, i\}$

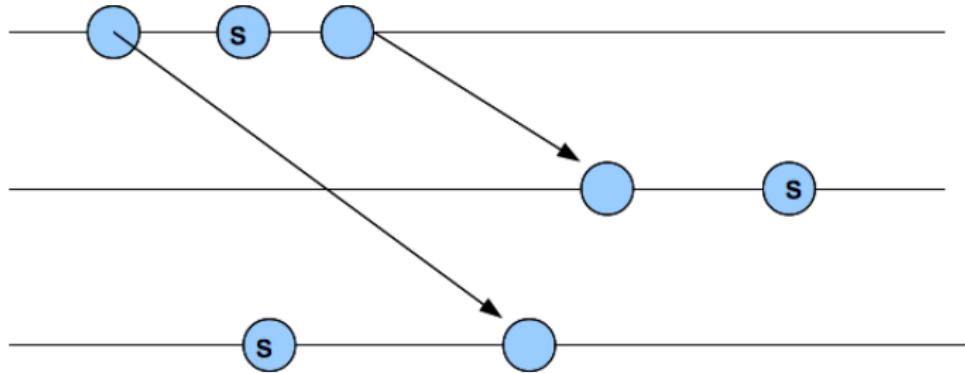
- Une coupure  $C$  est un ensemble d'événements
- Une coupure consistante  $C$  est une coupure satisfaisant la propriété suivante :  $\forall a, b. a \in C \wedge b < a \Rightarrow b \in C$
- $Cut1$  est consistante mais  $Cut2$  n'est pas consistante ( $h < g$  mais  $h \notin Cut2$ ).

## Cas pathologique



## Cas pathologique

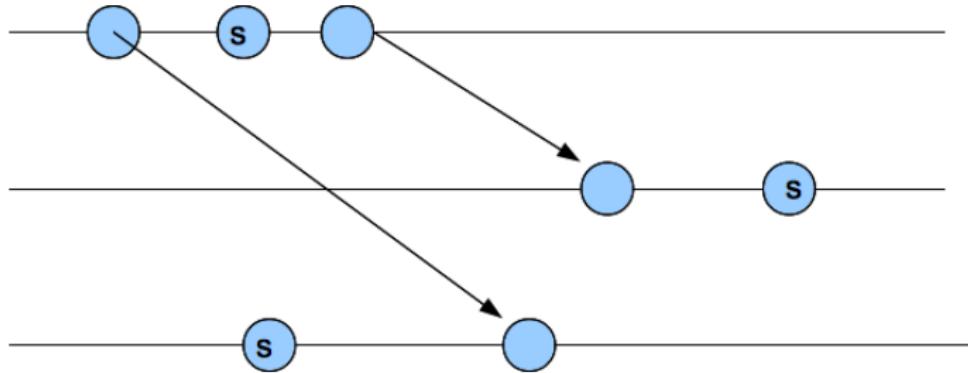
---



- Un message est reçu parce qu'il a été envoyé

## Cas pathologique

---



- Un message est reçu parce qu'il a été envoyé
- *Pour tout canal de communication, tout message reçu a été envoyé*

# Des coupures aux snapshots

---

- Deux coupures peuvent être équivalentes dans la mesure où l'échange de deux actions indépendantes c'est-à-dire causalement indépendantes.
- Pour une coupure donnée, on peut définir l'ensemble des événements les plus récents : ces événements constituent une frontière du calcul en cours.
- Une coupure induit donc un snapshot : une photographie du système et de son état global.
- Toute coupure consistante induit un snapshot consistant et le snapshot courant est l'ensemble des états locaux suivants les événements récemment enregistrés.
- Si  $C1$  et  $C2$  sont deux coupures telles que  $C1 \subseteq C2$ , alors  $S2$  est plus récent que  $S1$

- Une exécution (ou un calcul)  $C$  est un ordre total des événements
- Une exécution  $C$  est *consistante*, si elle est construite en satisfaisant la condition suivante :

Pour toute action  $a$  et  $b$  de  $C$ , si  $a < b$ , alors  $a$  precedes  $b$ .

- Deux exécutions sont équivalentes modulo la relation d'indépendance : deux actions  $a$  et  $b$  sont indépendantes si ni  $a$  precedes  $b$  et ni  $b$  precedes  $a$
- Les algorithmes vont construire des snapshots consistants mais vont faire des choix selon les techniques utilisées.

## 1 Election

Problème de l'élection

## 2 Election

Problème de l'élection

Election dans un graphe acyclique

## 3 Auto-stabilisation

Généralités

Définition

Réseaux en anneau

Algorithme de Dijkstra

Algorithme de coloriage

## 4 Calcul de l'état global

Introduction du problème

Etat global consistant

Algorithme de Chandy et Lamport

# Hypothèses de l'algorithme

---

- $P$  est un ensemble de processus connectés via un ensemble de canaux  $C$
- Pas d'horloge commune ni de mémoire partagée
- Toute processus peut enregistrer son état local, envoyer des messages ou recevoir des messages
- Les canaux sont
  - ▶ unidirectionnels
  - ▶ gérés selon la politique FIFO
  - ▶ de capacité infinie
  - ▶ sans erreur
  - ▶ équitables
- *Construire un snapshot consistant*

# Principe de cet algorithme

---

- On se donne deux ensembles :  $P$  est l'ensemble des processus et  $M$  est l'ensemble des messages
- On définit un message particulier appelé marqueur et noté  $\bowtie$
- Un ensemble de processus  $P$  connectés par des canaux caractérisés par deux variables
  - ▶  $sent \in P \times P \rightarrow (Nodes \rightarrow M)$
  - ▶  $rec \in P \times P \rightarrow (Nodes \rightarrow M)$
  - ▶  $dom(sent) = dom(rec)$
- On se donne un initiateur du calcul  $i \in P$ .
- On se donne une variable  $s$  qui contient tous les processus au début de l'algorithme
- Deux actions sont possibles :
  - ▶  $CL1$  : le processus  $i$  réalise les actions suivantes en un pas de calcul, si  $i \in s$  : soustraire  $i$  de  $s$  ; enregistrer l'état local ; envoyer le marqueur  $\bowtie$  à tous ses canaux.
  - ▶  $CL2$  : le processus  $j \in P$  reçoit le marqueur  $\bowtie$  sur un canal incident et réalise les actions suivantes : soustraire  $j$  de  $s$  ; enregistrer l'état local ; envoyer le marqueur  $\bowtie$  à tous ses canaux.

# Algorithme de Chandy et Lamport CL

## Initialisation de CL

Initialisation du processus de calcul par un processus  $p$  tel que  $taken_p = false$ .

**BEGIN**

**record(state( $p$ ))**

$taken_p := true$

**send  $m$  to  $Neighbors(p)$**

**For each port  $i$  of  $p$ , record messages arriving on  $i$**

**END**

# Algorithme de Chandy et Lamport CL

## Initialisation de CL

Initialisation du processus de calcul par un processus  $p$  tel que  $taken_p = false$ .

**BEGIN**

**record(state( $p$ ))**

$taken_p := true$

**send  $m$  to  $Neighbors(p)$**

**For each port  $i$  of  $p$ , record messages arriving on  $i$**

**END**

- Applicable par n'importe quel nœud qui souhaite commencer

# Algorithme de Chandy et Lamport CL

## Initialisation de CL

Initialisation du processus de calcul par un processus  $p$  tel que  $taken_p = false$ .

**BEGIN**

**record(state( $p$ ))**

$taken_p := true$

**send  $m$  to  $Neighbors(p)$**

**For each port  $i$  of  $p$ , record messages arriving on  $i$**

**END**

- Applicable par n'importe quel nœud qui souhaite commencer
- Action effectuée localement

# Algorithme de Chandy et Lamport

## R-CL

Une action est activée si un marqueur m arrive sur un port j de p.

**BEGIN**

**receive**  $m$  **on**  $j$  **of**  $p$

**mark**( $port\ j$ )

**IF**  $not\ taken_p$  **THEN**

$taken_p := true$

**record**(state( $p$ ))

**send**  $m$  **to**  $Neighbors(p)$

**For each port**  $i \neq j$  **of**  $p$ , **record messages arriving on i via**  $M_{p,i}$

**ELSE**

**Stop recording messages incoming from port j of p**

**record**( $M_{p,j}$ )

**FI**

**IF**  $p$  **has received a marker via all incoming channels** **THEN**

$local\_snapshot_p := true$

**FI**

**END**

# Propriétés de l'algorithme

---

- Pourquoi l'algorithme termine :
  - ▶ les canaux sont supposés fiables
  - ▶ les messages sont transmis FIFO
  - ▶  $s$  est vide à la terminaison
  - ▶ Quand  $s$  est vide, tous les processus ont reçu le marqueur sur leurs canaux incidents.

# Applications de l'algorithme

---

- Calculer le nombre de jetons dans un réseau
- Détermination de point de reprise de calcul

- On considère un réseau de type anneau de 1 à 3 dans ce sens

- On considère un réseau de type anneau de 1 à 3 dans ce sens
- CL1 : le processus 1 envoie le jeton à 2 ; le processus devient local ( $n(1) = 0$ ) ; il positionne les variables associées aux canaux :  $sent(1, 2) = 1$  et  $rec(3, 1) = 0$ ) ; il envoie le marqueur sur le canal (1, 2).

- On considère un réseau de type anneau de 1 à 3 dans ce sens
- CL1 : le processus 1 envoie le jeton à 2 ; le processus devient local ( $n(1) = 0$ ) ; il positionne les variables associées aux canaux :  $sent(1, 2) = 1$  et  $rec(3, 1) = 0$  ; il envoie le marqueur sur le canal (1, 2).
- E : le processus i reçoit le jeton et le renvoie à son voisin.

- On considère un réseau de type anneau de 1 à 3 dans ce sens
- CL1 : le processus 1 envoie le jeton à 2 ; le processus devient local ( $n(1) = 0$ ) ; il positionne les variables associées aux canaux :  $sent(1, 2) = 1$  et  $rec(3, 1) = 0$  ; il envoie le marqueur sur le canal (1, 2).
- E : le processus i reçoit le jeton et le renvoie à son voisin.
- CL2 : le processus i reçoit le marqueur ; le processus i devient local ( $n(i) = 0$ ) ; il positionne  $rec(i-1, i) = 1$  et  $sent(i, i+1) = 1$  ; il envoie le marqueur sur le canal (i, i+1).

- On considère un réseau de type anneau de 1 à 3 dans ce sens
- CL1 : le processus 1 envoie le jeton à 2 ; le processus devient local ( $n(1) = 0$ ) ; il positionne les variables associées aux canaux :  $sent(1, 2) = 1$  et  $rec(3, 1) = 0$  ; il envoie le marqueur sur le canal (1, 2).
- E : le processus i reçoit le jeton et le renvoie à son voisin.
- CL2 : le processus i reçoit le marqueur ; le processus i devient local ( $n(i) = 0$ ) ; il positionne  $rec(i-1, i) = 1$  et  $sent(i, i+1) = 1$  ; il envoie le marqueur sur le canal (i, i+1).
- L'algorithme termine et le nombre de jetons enregistrés est :  
$$n_1 + n_2 + n_3 + sent(1, 2) - rec(1, 2) + sent(2, 3) - rec(2, 3) + sent(3, 1) - rec(3, 1)$$

# Algorithme de Lay et Yang

---

# Algorithme de Lay et Yang

---

- LY1 : L'initialisateur enregistre son état et quand il doit envoyer un message à un autre processus, il envoie le message  $(m,r)$  où  $r$  désigne une couleur.

# Algorithme de Lay et Yang

---

- LY1 : L'initialisateur enregistre son état et quand il doit envoyer un message à un autre processus, il envoie le message  $(m,r)$  où  $r$  désigne une couleur.B
- LY2 : Quand un processus reçoit un message  $(m,r)$ , il enregistre son état, si ce n'est pas déjà fait, et accepte le message  $m$ .

roa

# Algorithme de Lay et Yang

---

- LY1 : L'initialisateur enregistre son état et quand il doit envoyer un message à un autre processus, il envoie le message  $(m,r)$  où  $r$  désigne une couleur.B
- LY2 : Quand un processus reçoit un message  $(m,r)$ , il enregistre son état, si ce n'est pas déjà fait, et accepte le message  $m$ .

roa

- L'accessibilité de l'état global n'est pas garanti