



# Summary

---

- 1 Problème de l'élection
- 2 Election dans un graphe acyclique

# Section Courante

---

- ① Problème de l'élection
- ② Election dans un graphe  
acyclique

]

- 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

---

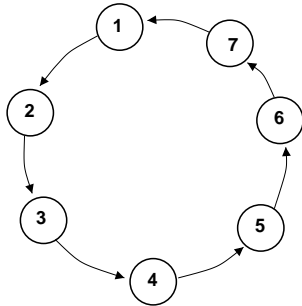
- Un ensemble de nœuds connectés en anneau
- Un sous-ensemble des nœuds est appelé l'ensemble des candidats à l'élection
- Si un nœud candidat reçoit un jeton plus petit que son jeton, il passe le jeton et se déclare perdu, sinon il garde le jeton.
- Si un nœud reçoit le jeton d'un autre nœud et qu'il 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.

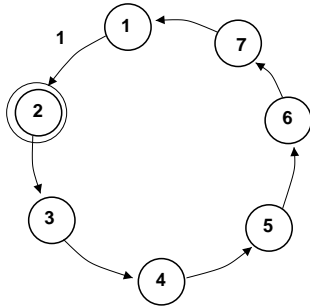
# Algorithme LRC Chang et Roberts

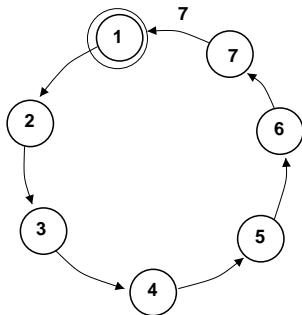
---

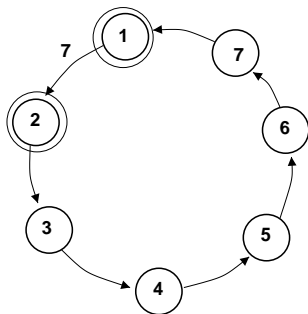
- Un réseau en anneau comporte des nœuds ayant un numéro différent.
- Un des nœuds est le maximum du réseau
- Le processus d'élection transmet un jeton avec une valeur mise à jour à chaque nœud.

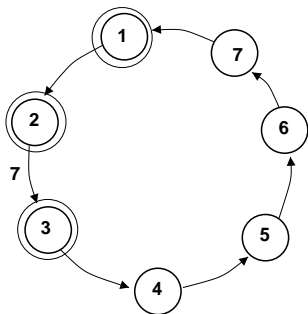


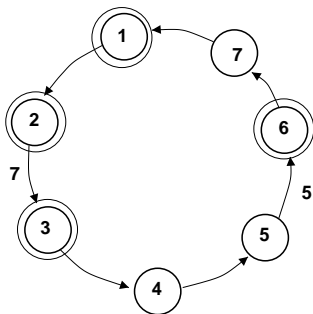


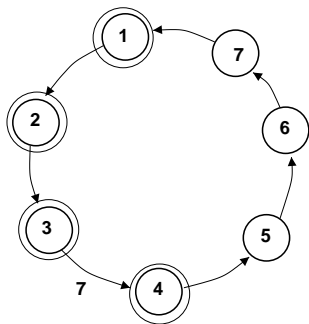


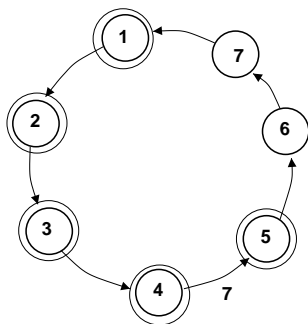














# Algorithme LCR

---

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

# Section Courante

---

- ① Problème de l'élection
- ② Election dans un graphe  
acyclique

]

## (FireWire)

---

- 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

## Références (1)

# 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

- Problèmes de l'élection du leader(18 mars 2025) (Dominique Méry)

- DDL: *Schema*

- 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



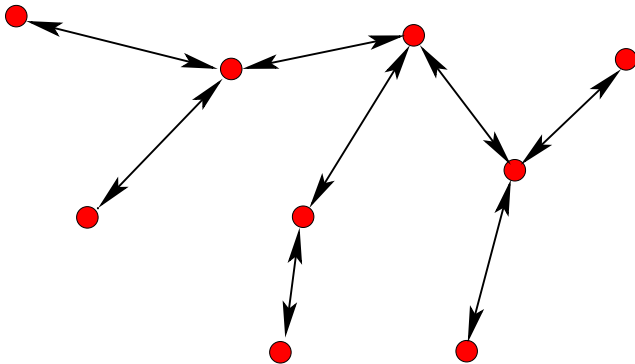
### THEOREM PROVING

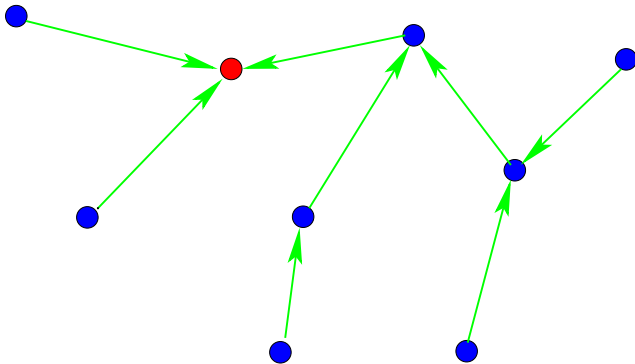
- M. Devillers et al. *Verification of the Leader Election : Formal Method Applied to IEEE 1394*. Formal Methods in System Design. 2000
- J.R. Abrial et al. *A Mechanically Proved and Incremental Development of IEEE 1394*. Formal Aspects of Computing, 2003.

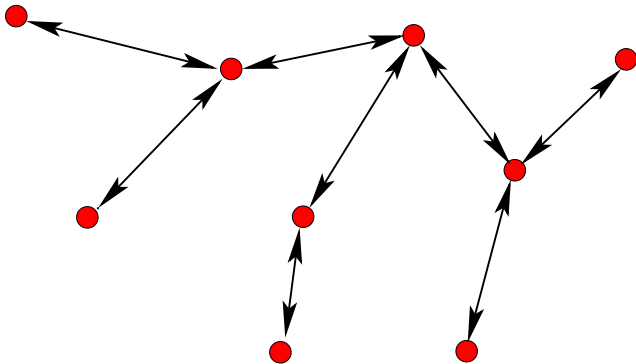
# Propriétés informelles abstraites du protocole

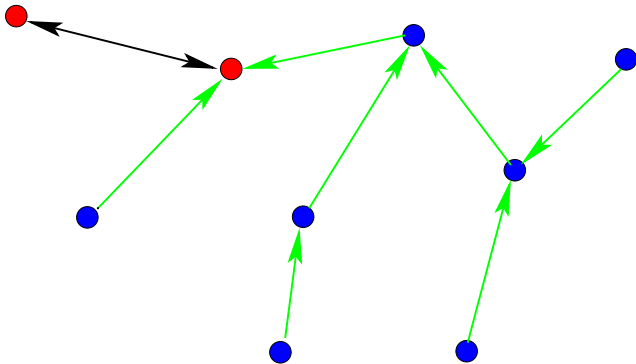
---

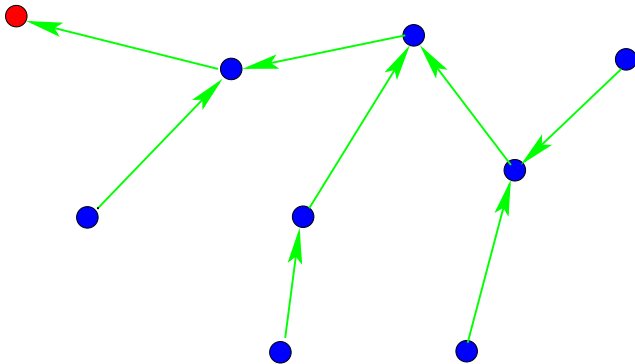
- Un réseau **connexe** et **acyclique** de nœuds.
- Nœuds reliés par des canaux **bidirectionnels**
- Election en un temps fini d'un **leader**.
- De manière **répartie** et **non-déterministe**.
- Un exemple d'animation du protocole.









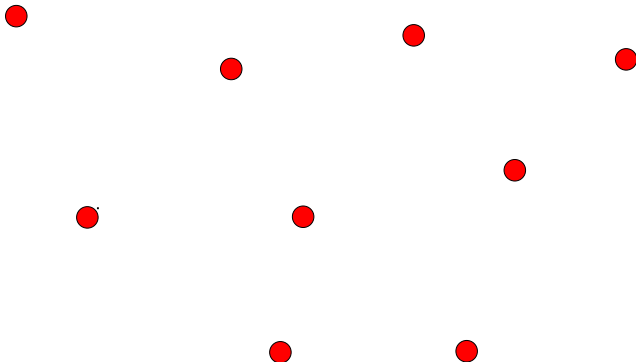


# Le processus de développement

---

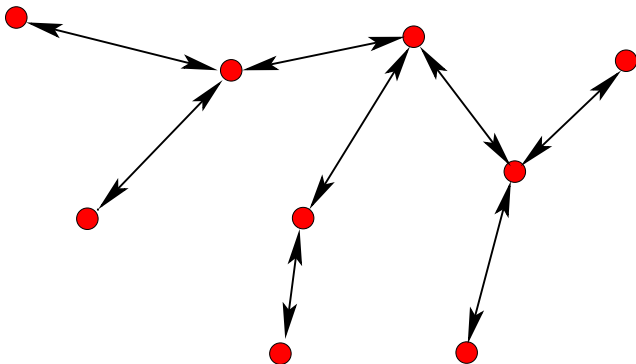
- Définition et propriétés du réseau dans un style formel
- Un modèle abstrait `one-shot` du protocole.
- Présentation d'une solution centralisée mais encore abstraite !
- Introduction du mécanisme de message passing entre les nœuds et des délais
- Modification de la structure de donnée pour répartir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





ND un ensemble de noeuds (au moins 2 noeuds)





$gr$  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  $ND$

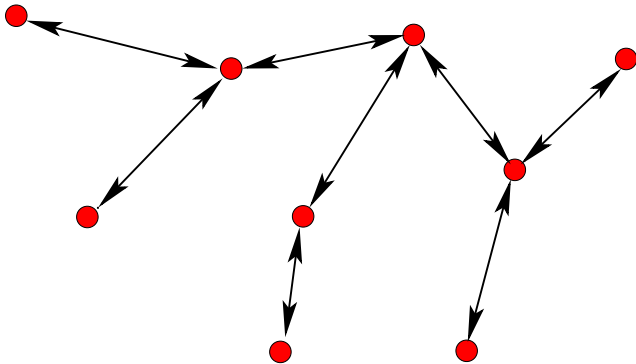
$$\text{dom}(gr) = ND$$

$gr$  est symétrique

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

$gr$  est non réflexif

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



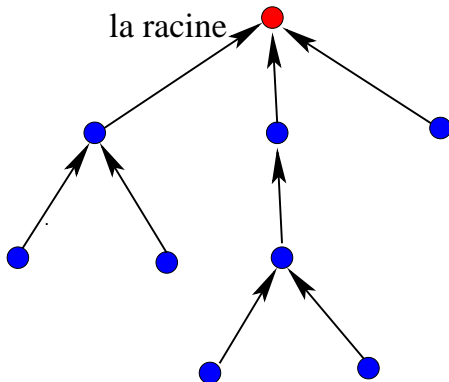
gr est connexe et acyclique



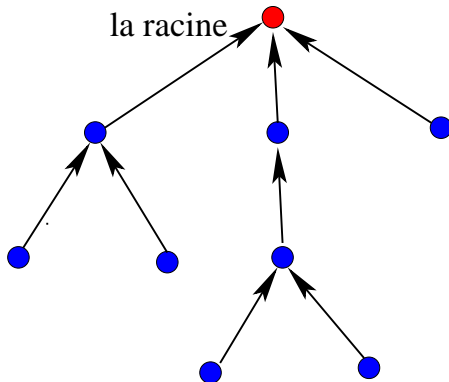
# Un détour par les Arbres

---

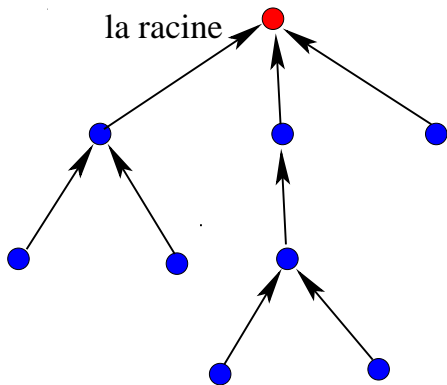
- Un arbre est un **graphe particulier**
- Un arbre a une **racine**
- Un arbre a une **fonction parentale**
- Un arbre est **acyclique**
- Un arbre est **connexe à partir de la racine**



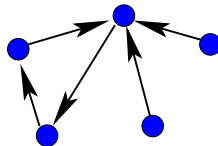
Un arbre  $t$  construit sur les noeuds



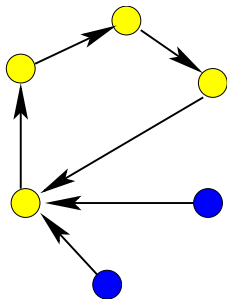
$t$  est une fonction définie sur ND sauf pour la racine!



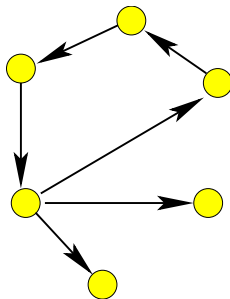
Eviter des cycles



Mauvais exemple!



Un cycle



Son image inverse

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

# Le prédicat $\text{tree}(r, t)$

Le prédicat  $\text{tree}(r, t)$

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

Le prédicat  $\text{tree}(r, t)$

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

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



Le prédicat  $\text{tree}(r, t)$

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

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

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

$t$  est acyclique : **formulations équivalentes**

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

On obtient une Règle d'Induction

$$\forall q \cdot$$

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

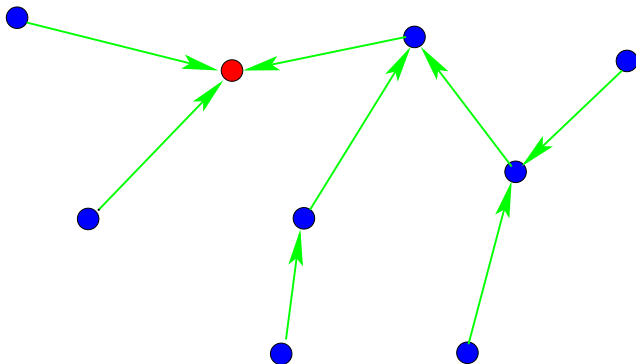
Le prédicat **tree** ( $r, t$ )

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

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

$t$  est acyclique

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



Un arbre de recouvrement  $t$  de  $gr$

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

$r, t$  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  $\triangleq$

**BEGIN**

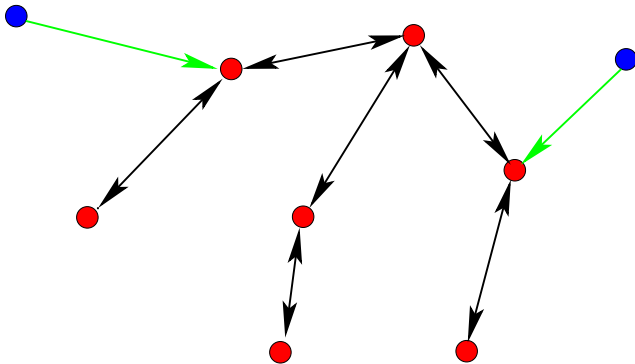
$rt, ts : \text{spanning}(rt, ts, gr)$

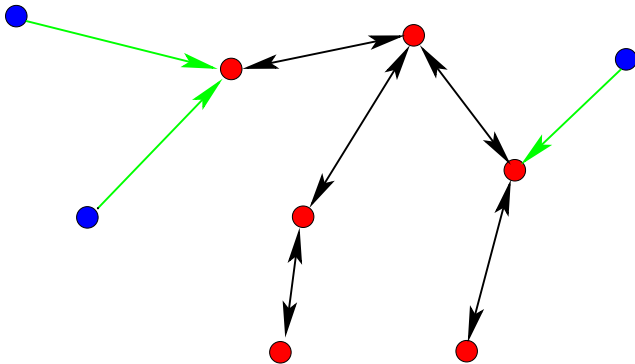
**END**

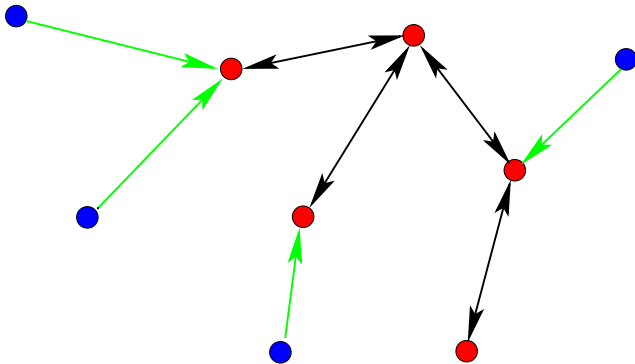




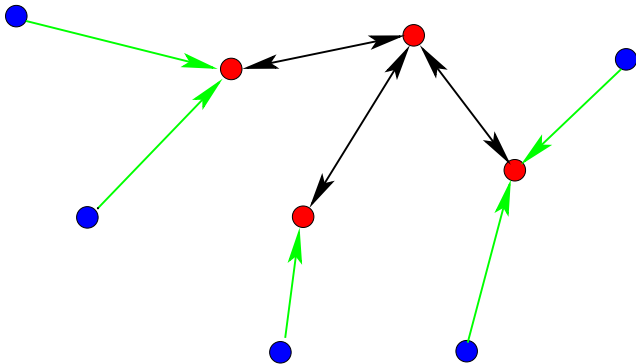


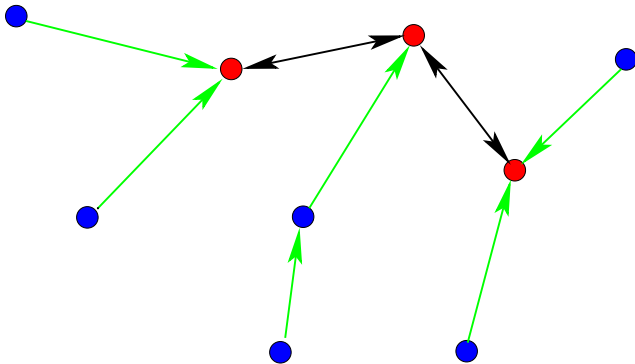




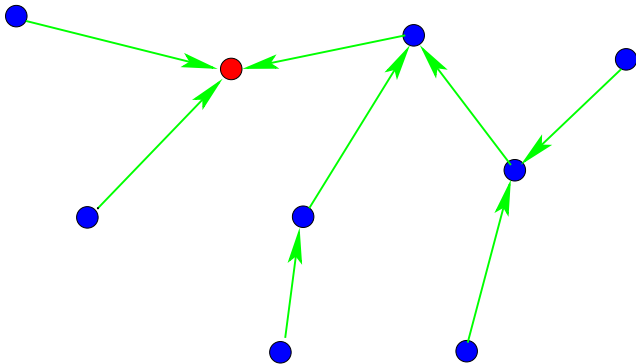












- Les **flèches vertes** correspondent à la fonction  $tr$
- Les **nœuds bleus** sont le **domaine** de  $tr$
- La fonction  $tr$  est une **forêt** sur les nœuds
- Les **nœuds rouges** sont les **racines** de ce arbres

Le prédicat **invariant** ( $tr$ )

$$tr \in ND \rightarrow ND$$

## Le prédicat **invariant** ( $tr$ )

$$tr \in ND \rightarrow ND$$

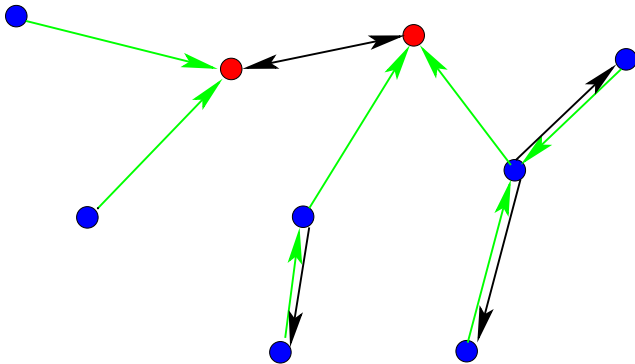
$$\forall p \cdot \left( \begin{array}{l} p \subseteq ND \quad \wedge \\ ND\text{-dom}(tr) \subseteq p \quad \wedge \\ tr^{-1}[p] \subseteq p \\ \implies \\ ND \subseteq p \end{array} \right)$$

Le prédicat **invariant** ( $tr$ )

$$tr \in ND \rightarrow ND$$

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



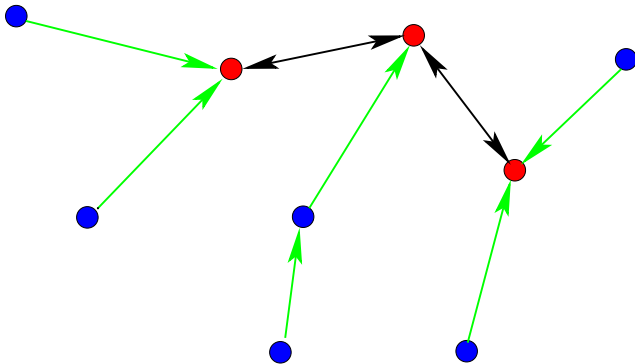


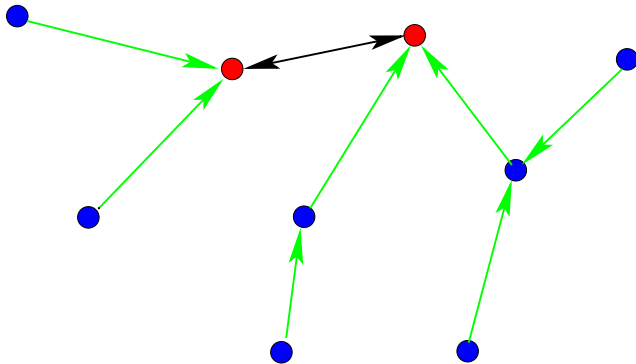
## Premier raffinement (2)

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

- Raffiner l'événement abstrait "elect"

- Retour à l'**animation** : Observez la "garde" de progress



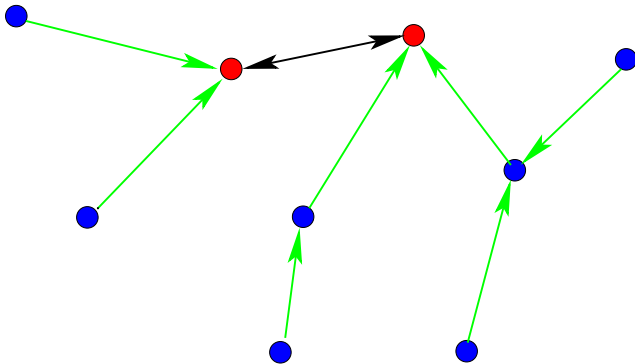


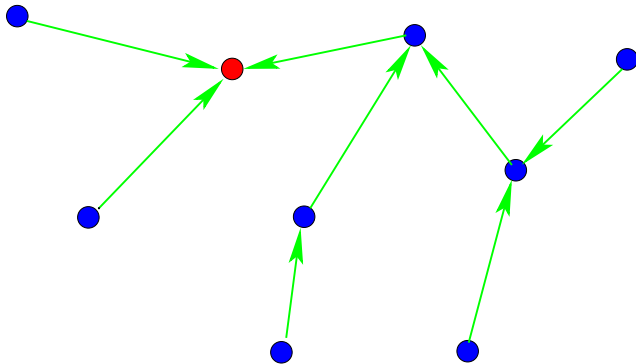
Quand un nœud rouge  $x$  est connecté à un autre nœud rouge  $y$  alors l'événement "progress" peut apparaître

EVENT progress  $\hat{=}$   
**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  $\hat{=}$   
  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 :

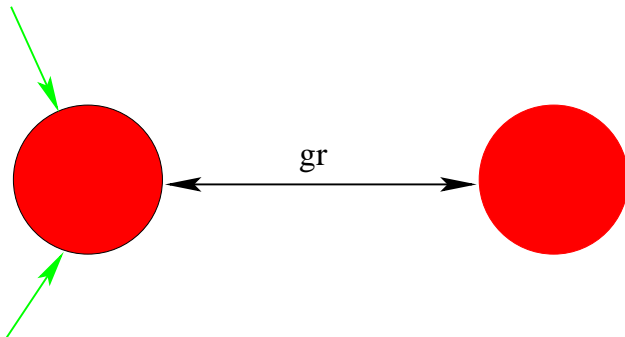
$$\begin{aligned} & \text{invariant}(tr) \quad \wedge \\ & x \in ND \quad \wedge \\ & gr[\{x\}] = tr^{-1}[\{x\}] \\ & ts = tr \\ \implies & \text{spanning}(x, ts, gr) \end{aligned}$$

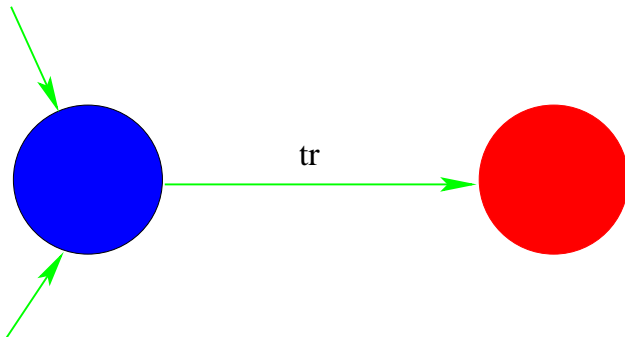
## Un lemme

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

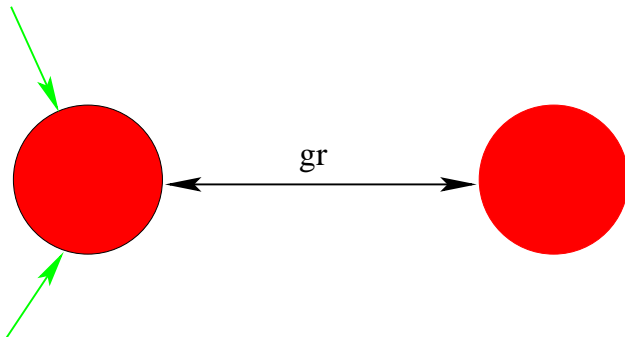




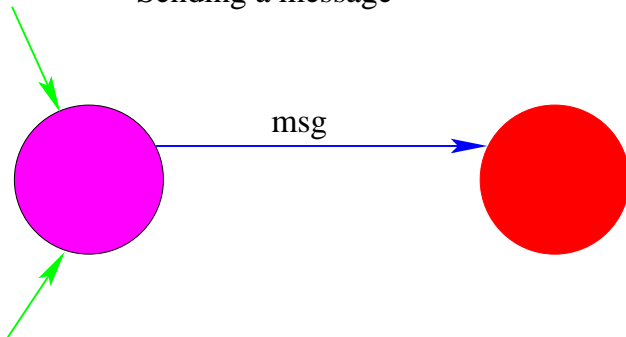






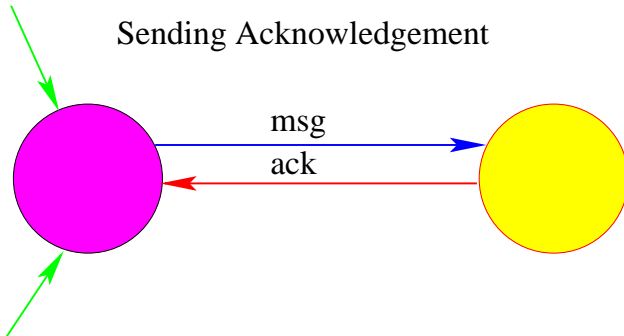


## Sending a message



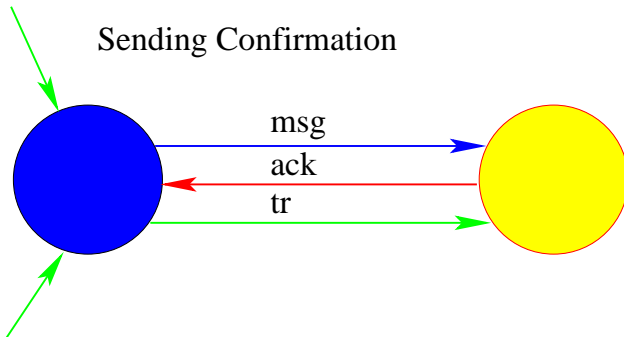
Receiving a message

Sending Acknowledgement

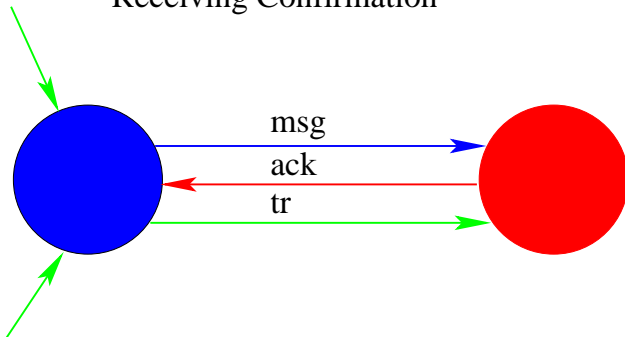


Receiving Acknowledgement

Sending Confirmation



## Receiving Confirmation



## Invariant (1)

$$msg \in ND \leftrightarrow ND$$

$$ack \in ND \leftrightarrow ND$$

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

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

EVENT send\_msg  $\hat{=}$

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



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

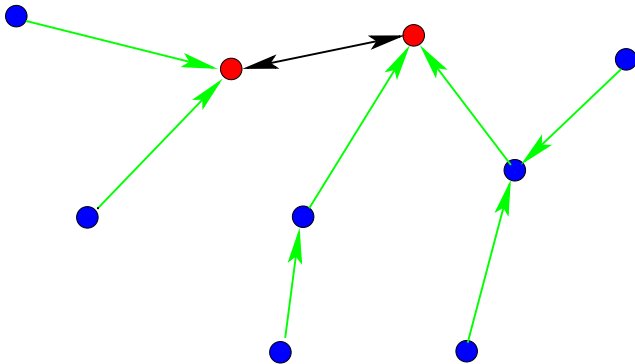
EVENT progress  $\hat{=}$   
**ANY**  $x, y$  **WHERE**  
 $x, y \in ack \wedge$   
 $x \notin \text{dom}(tr)$   
**THEN**  
 $tr := tr \cup \{x \mapsto y\}$   
**END**

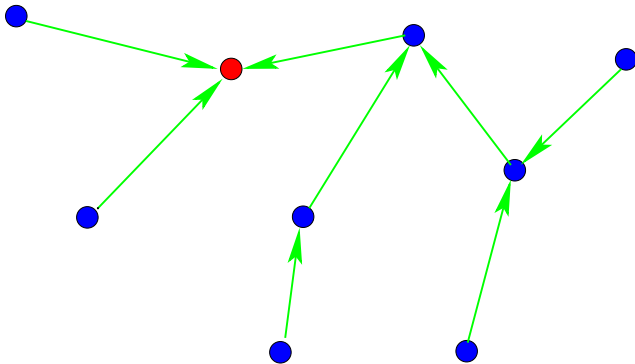
## Invariant (2)

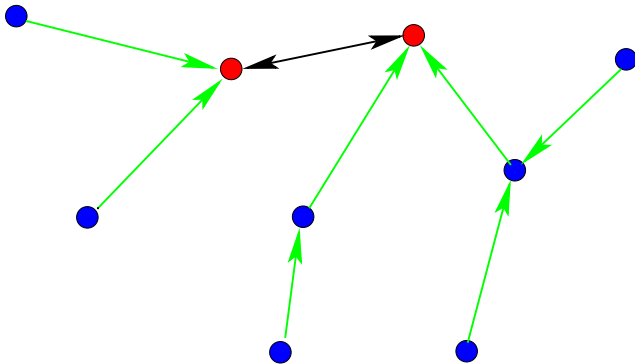
$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in msg-ack \\ \implies \\ x, y \in gr \quad \wedge \\ x \notin \text{dom}(tr) \quad \wedge \quad y \notin \text{dom}(tr) \quad \wedge \\ gr[\{x\}] = tr^{-1}[\{x\}] \cup \{y\} \end{array} \right)$$

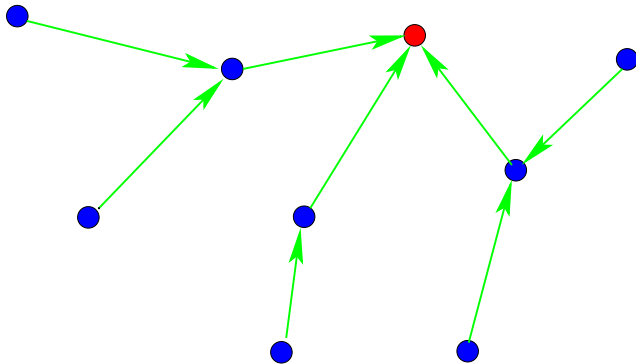
$$\forall (x, y) \cdot \left( \begin{array}{l} x, y \in ack \quad \wedge \\ x \notin \text{dom}(tr) \\ \implies \\ x, y \in gr \quad \wedge \\ y \notin \text{dom}(tr) \quad \wedge \\ gr[\{x\}] = tr^{-1}[\{x\}] \cup \{y\} \end{array} \right)$$

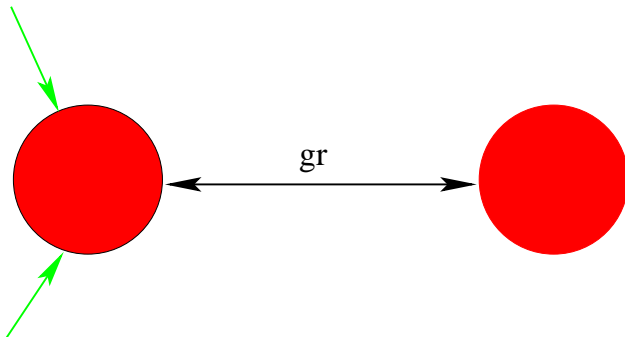
- Expliquer le **problème**
- Proposer une solution **partielle**.
- Vers un **meilleur** traitement
- Retour à l'animation **locale**.





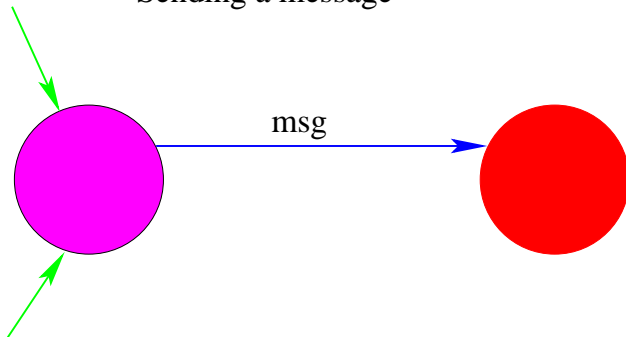






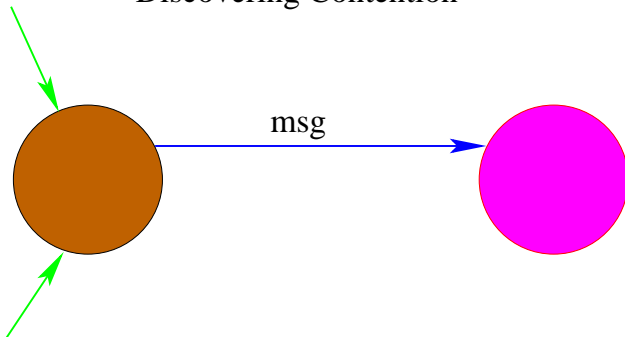


## Sending a message




The diagram illustrates a process sending a message to another process. On the left, a magenta circle represents the sender process. Two green arrows point towards it from the left, with the text "Sending another message" above them. A blue arrow labeled "msg" points from the sender process to a magenta circle on the right, which represents the receiver process. A second blue arrow labeled "msg" points from the receiver process back to the sender process, indicating a return message.

## Discovering Contention




# Discovering Contention



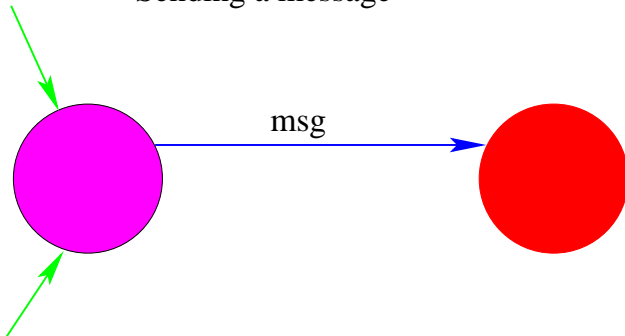
The diagram illustrates the concept of 'Discovering Contention'. It features two large orange circles. The circle on the left is the focus of two green arrows pointing towards it from the left, indicating incoming requests or contention. The circle on the right is isolated, with no arrows pointing towards it, representing a state without contention.

## Recovering from Contention

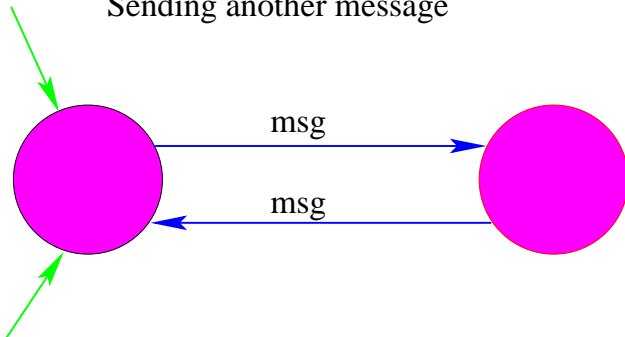


The diagram illustrates the recovery from contention. It features two red circles. The circle on the left is the focus of two green arrows pointing towards it from the left, indicating a recovery or resolution process. The circle on the right is isolated, representing a state where contention has been resolved or avoided.

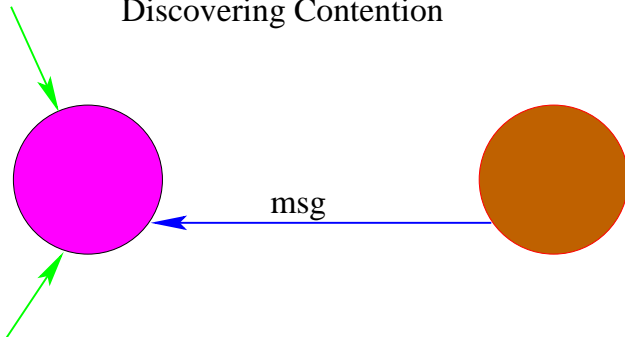
## Sending a message



Sending another message




## Discovering Contention






# Discovering Contention



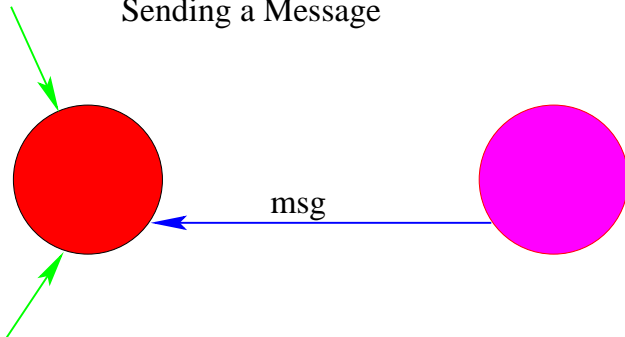
The diagram consists of two orange circles. The circle on the left has two green arrows pointing towards it from the top-left and bottom-left. The circle on the right has a red border.

## Recovering from Contention

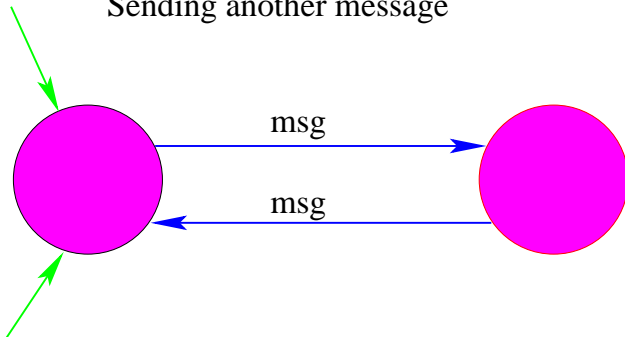


The diagram illustrates a recovery process. On the left, a red circle is the target of two green arrows pointing towards it from the left. On the right, there is another red circle, but it is not the target of any arrows.

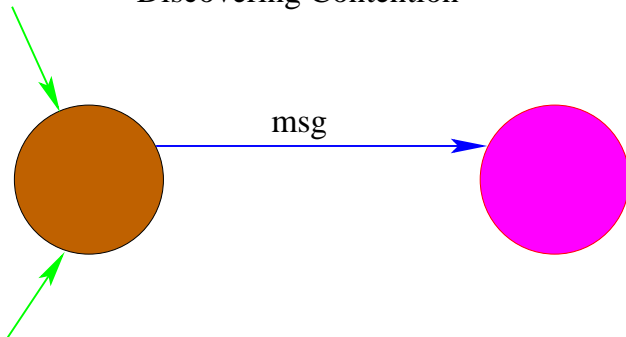
## Sending a Message




Sending another message



## Discovering Contention




# Discovering Contention



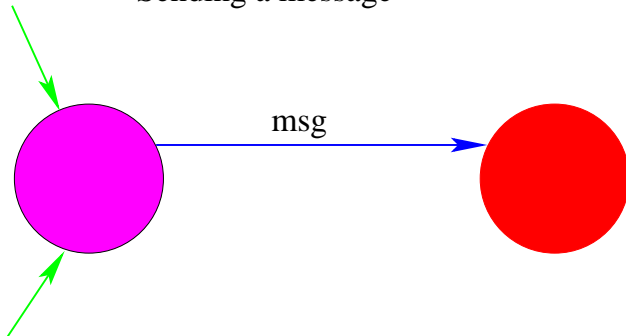
The diagram consists of two orange circles. The left circle has two green arrows pointing towards it from the top-left and bottom-left. The right circle has a red border.

## Recovering from Contention



The diagram illustrates the recovery from contention. It features two red circles. The left circle is the focus, with two green arrows pointing towards it from the left, indicating incoming requests or data. The right circle is isolated, representing a state where contention has been resolved or avoided.

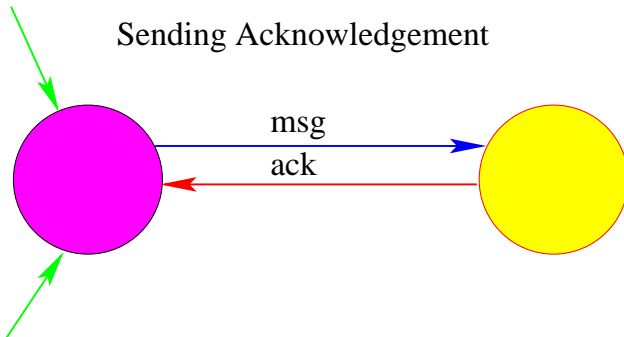
## Sending a message





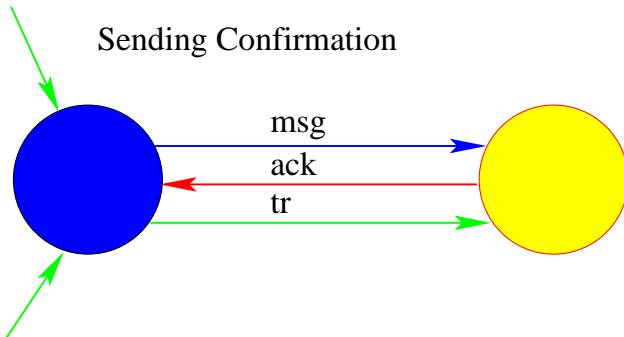
Receiving a message

Sending Acknowledgement

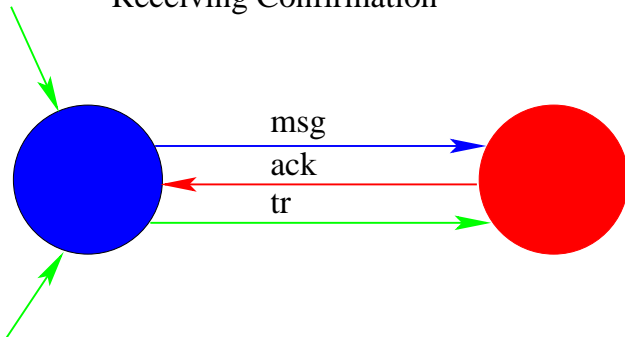


Receiving Acknowledgement

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)

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

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

## Invariant (4)

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

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

$$ack \cap cnt = \emptyset$$

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

```
EVENT solve_contention  $\hat{=}$   
  ANY  $x, y$  WHERE  
     $x, y \in cnt \cup cnt^{-1}$   
  THEN  
     $msg := msg - cnt$  ||  
     $cnt := \emptyset$   
  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** ( $tr$ )

$$tr \in ND \rightarrow ND$$

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

### 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)

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

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

$$ack \cap cnt = \emptyset$$



# 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)

---

$$bm \subseteq ND$$

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

$$bt \subseteq ND$$

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

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

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

- Node  $x$  is elected the leader

EVENT elect  $\hat{=}$   
**ANY**  $x$  **WHERE**  
 $x \in ND \wedge$   
 $nb(x) = sn(x)$   
**THEN**  
 $rt := x$   
**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  $\hat{=}$   
  ANY  $x, y$  WHERE  
     $x, y \in tr \wedge$   
     $x \notin sn(y)$   
  THEN  
     $sn(y) := sn(y) \cup \{x\}$   
  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  $\hat{=}$ 
  ANY  $x, y$  WHERE
     $x, y \in cnt \cup cnt^{-1}$ 
  THEN
     $msg := msg - cnt$  ||
     $bm := bm - \text{dom}(cnt)$  ||
     $cnt := \emptyset$ 
  END

```