Algorithmique Distribuée

Problèmes et Algorithmes Fondamentaux

Shantanu Das

http://pageperso.lif.univ-mrs.fr/~shantanu.das/M1algodist/

Aix-Marseille Université

20/21 fevrier 2018

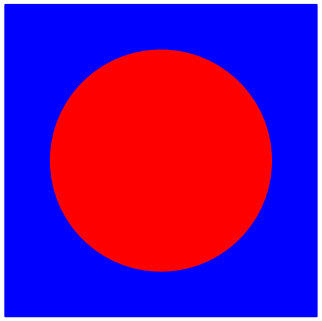
Problèmes et Algorithmes Fondamentaux

Deux types de systèmes :

- Système parallèle : séparer un gros problème en petits problèmes indépendants
- Système distribué : interaction et coopération de processus indépendants en vue de réaliser une tâche donnée

Dans ce cours, on ne s'intéressera qu'aux **systèmes distribuées**.

Algorithmique "classique"

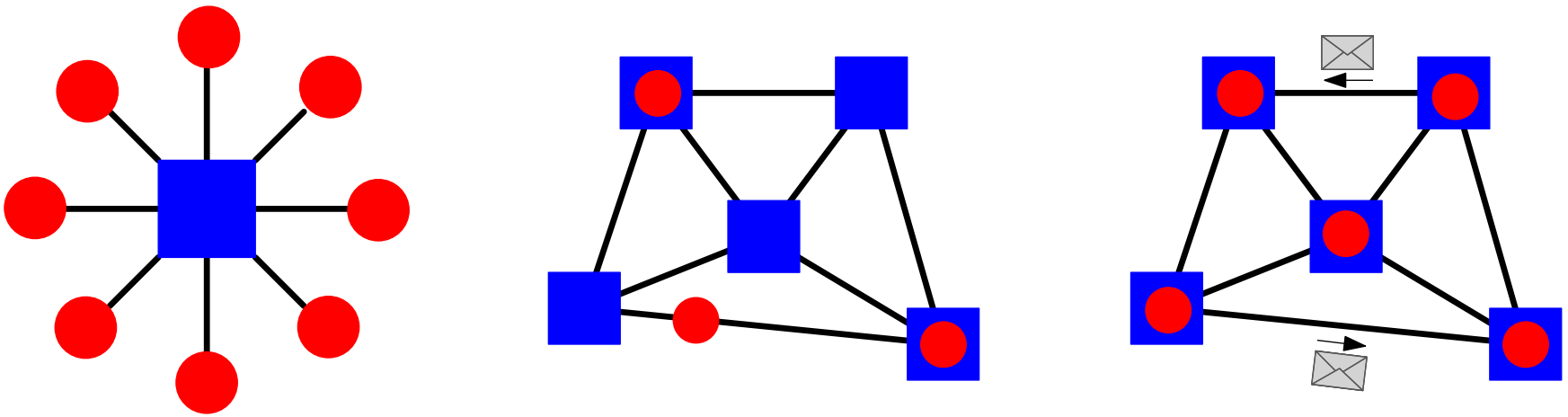


Processus

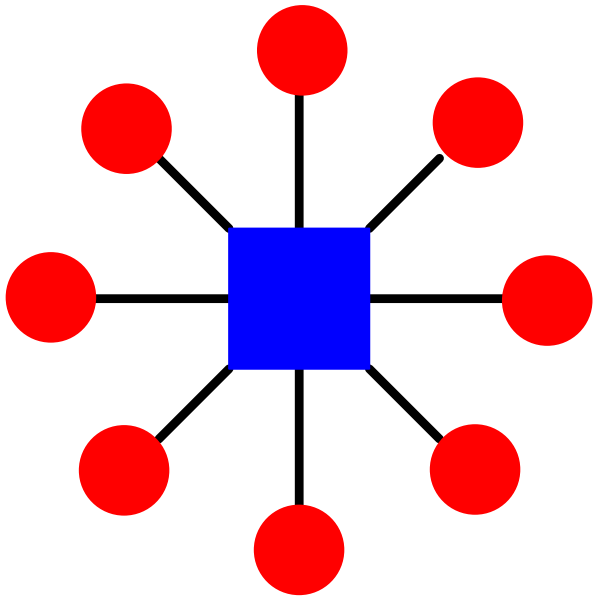
Donnée

- Un processus qui exécute un algorithme
- Une donnée
- Complexité : nombre d'opérations/instructions

Trois types de modèles pour l'algorithmique distribuée



Mémoire partagée

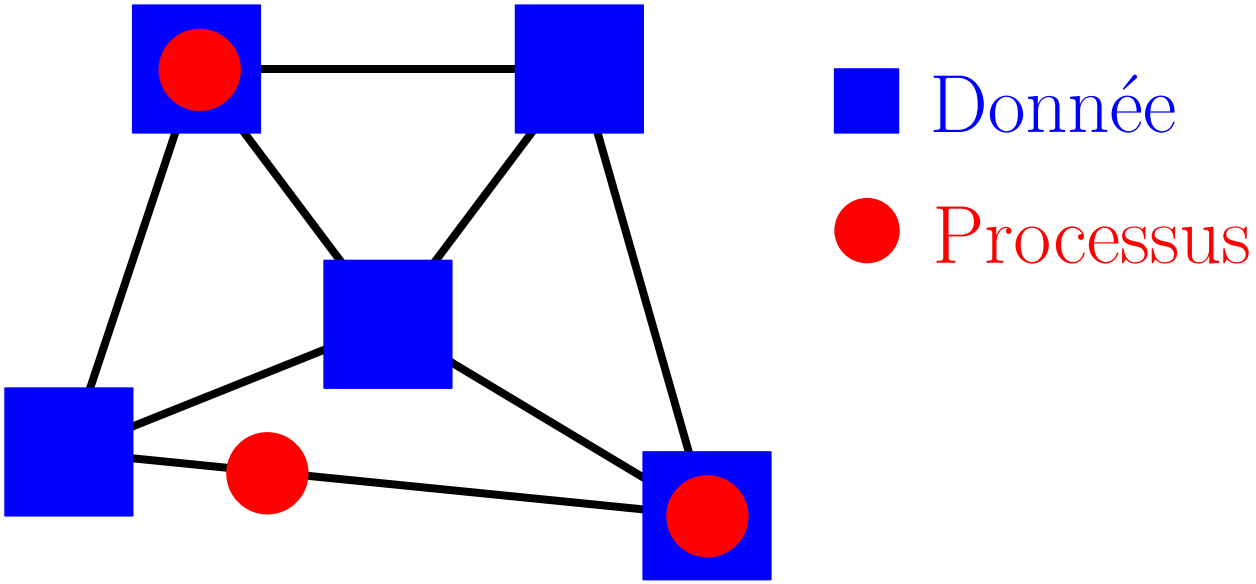


Processus

Donnée

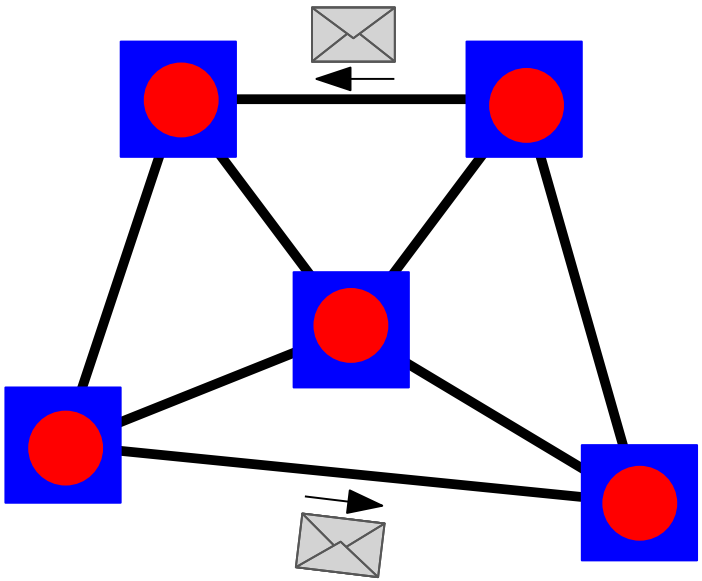
- une donnée
- n processus qui exécutent le même algorithme et peuvent lire/écrire dans la donnée
- complexité : nombre d'écriture/lecture

Agents mobiles



- n données
- k processus qui exécutent le même algorithme et peuvent se déplacer dans le réseau
- complexité : nombre de déplacements

Passage de message



Processus

Donnée

- n données
- n processus qui exécutent le même algorithme et peuvent envoyer des messages à leur voisins
- complexité : nombre de messages

Quelques propriétés pour le modèle de communication par passage de message

- Tout les processus exécutent le "même" algorithme. (l'état initial peut différer.)
- Chaque processus connait l'information locale (pas d'information globale). Ils doivent communiquer pour l'échange des informations.
- A chaque étape de l'algorithme, un nœud peut :
  - envoyer des messages aux voisins,
  - recevoir des messages de voisins, et
  - faire des calculs locaux.

Tâches Fondamentaux

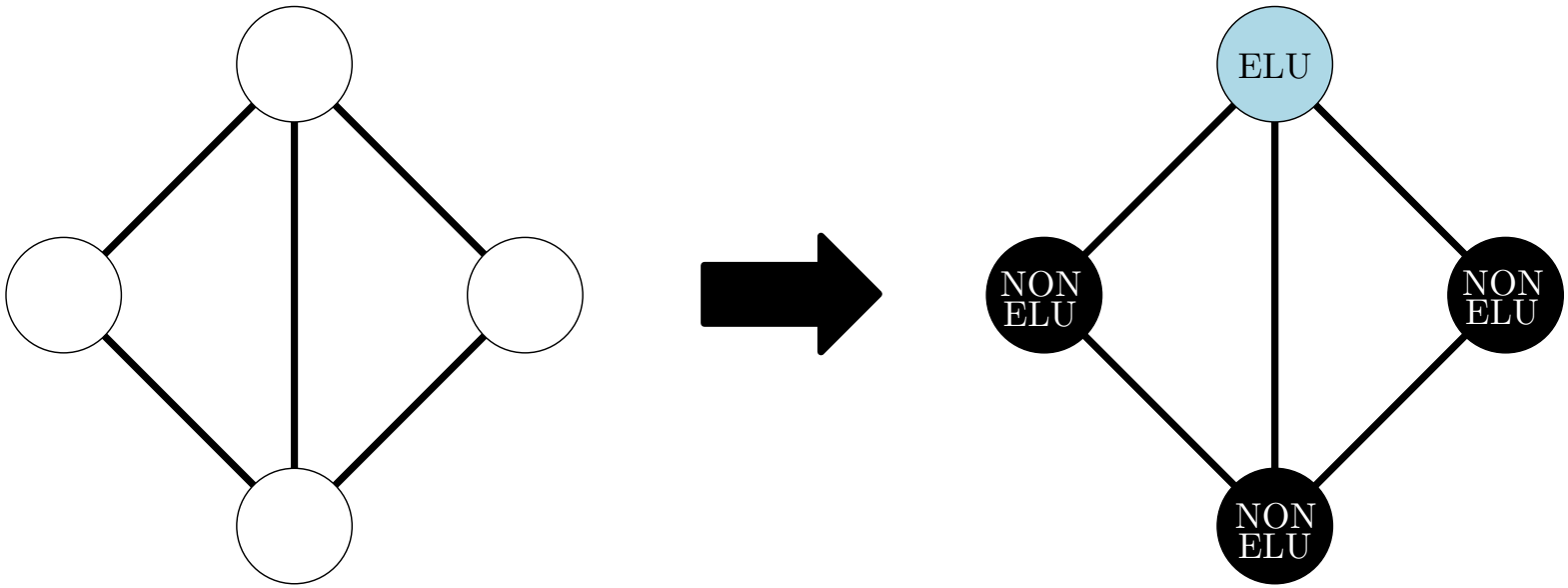
Les tâches distribuées se ramènent en général à une forme d'*accord distribué*.

Exemples (Définition informelle) :

- **Election** : un processus, et un seul, doit terminer dans l'état ELU (les autres sont non-ELU).
- **Consensus** : étant données des valeurs initiales pour chaque processus, ceux-ci doivent décider une, et une seule, de ces valeurs.

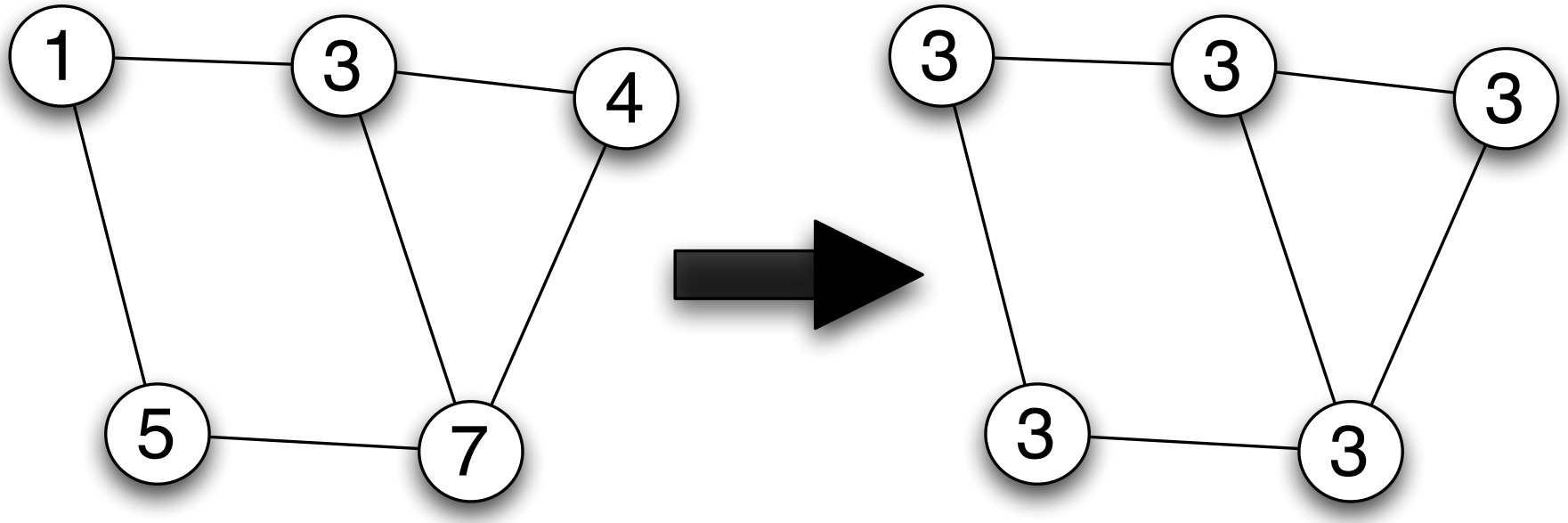
... et de nombreuses variations.

Problème d'Election



- Pourquoi est-il important ? (aider à la coordination)
- Pourquoi est-il difficile à résoudre ? (symétries)
- Comment effectuer l'élection ?
  (utiliser des identifiants uniques, UID)

Consensus dans un réseau



Tout le processus décide le même valeur.

Les Algorithmes

- Pour quel(s) modèle(s) de calcul distribué ?
- Avec quelle complexité ?
- Y-a-t-il toujours une solution ?

Les Algorithmes

- Pour quel(s) modèle(s) de calcul distribué ?
- Avec quelle complexité ?
- Y-a-t-il toujours une solution ?
  Non
  $\Longrightarrow$ importance de déterminer ce qui est possible/impossible en fonction des caractéristiques du système

Les Modèles Différents

1) Aspects Dynamiques :

- **synchrone :** tous les processus fonctionnent à la même vitesse
- **asynchrone :** les processus ont des vitesses différentes qui peuvent varier dans le temps
- **partiellement synchrone :** les processus fonctionnent à la même vitesse mais ne sont pas actifs à chaque ronde
- **types de pannes**
  - pannes définitives
  - pannes transitoires
  - pannes byzantines

Les Modèles Différents

2) Aspects Statiques

- information structurelle :
  - n° unique $\Longrightarrow$ identités
  - connexions orientées ou non
- connaissance structurelle :
  - famille de graphes (arbres, anneaux, planaires, ...) $\Longrightarrow$ domaine
  - borne sur la taille
  - borne sur le diamètre

Les Modèles Différents

3) Détection de la Terminaison

- **implicite (stabilisation) :** chaque processus ne sait pas quand il a décidé mais l'algorithme converge
- **faiblement locale :** chaque processus sait quand il a décidé et participe encore à l'algorithme une fois qu'il a décidé
- **locale :** chaque processus sait quand il a décidé et ne participe plus à l'algorithme une fois qu'il a décidé
- **globale :** un processus sait quand tous les processus ont décidés

NB Quand un processus termine en donnant une valeur, on dit qu'il *décide*.

Exemples des Problèmes

- **Wakeup** (Réveiller tous les processus)
- **Broadcast** (Diffusion d'information)
- **Gossip** (Tous les processus communiquent à tous les autres processus)
- Construction d'**arbre couvrant**
- **Election**

La Communication

- Communication point à point
- Chaque nœud peut choisir une (ou plusieurs) arête incidente pour envoyer un message.
- Le receveur sait de quelle arête le message est arrivé.
- Taille de messages limitée (généralement logarithmique en n)
- But : **Réduire la quantité de communication** (le communication coute plus que le calcul local !)

A Propos de la Complexité

Comment comparer les performances ?

- Complexité de la Communication
  - Complexité en "espace" : nombre de messages
  - Complexité en "espace" : nombre de bits transmis
- Complexité en "temps"
  - synchrone : nombre de rondes
  - asynchrone : parfois nombre de "rondes" (exécutions optimistes)

Rappel :

On es intéressés à la quantité de la communication plutôt que la quantité de calculs.

Problème de l'Election

Spécification

Etant donné un réseau de noeud initialement dans l'état Inconnu, l'algorithme termine avec les conditions suivantes :

- **Election :** Un noeud et un seul termine dans l'état ELU
- **Election avec terminaison explicite :** Tous les autres noeuds savent que l'algorithme est terminé (et qu'ils sont non-ELU)

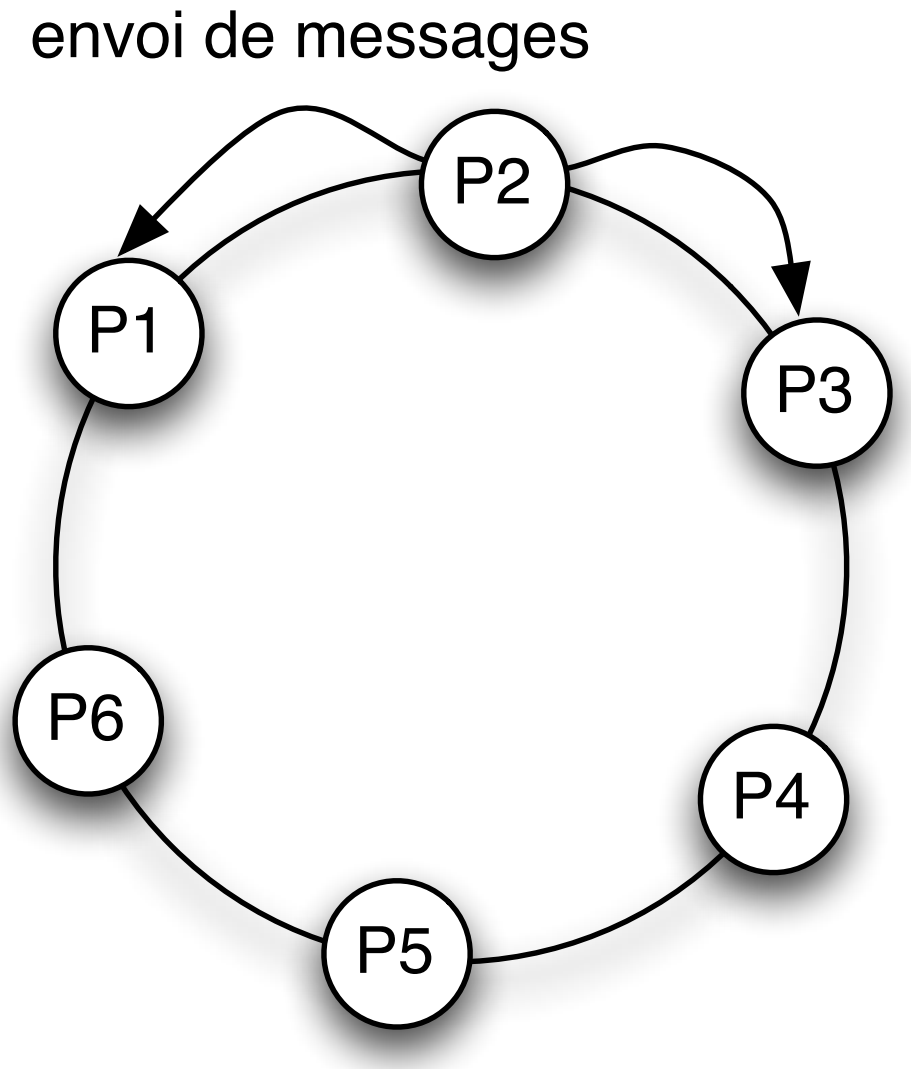
Systèmes Synchrones

Définition

Un système est synchrone si il existe une horloge globale partagée par tous les processus du système.

Envoi de message simultané, les messages sont reçus durant le même round.

Un Anneau de Processus



Processus Anonymes

Théorème

Dans un anneau de processus synchrones anonymes (états de départ identiques), il est impossible d'élire.

Preuve d'impossibilité

P_i : les agents ont tous le même état au début de la ronde i.

Preuve d'impossibilité

P_i : les agents ont tous le même état au début de la ronde i.

P_1 est vrai car les agents sont anonymes.

Preuve d'impossibilité

P_i : les agents ont tous le même état au début de la ronde i.

P_1 est vrai car les agents sont anonymes.

Supposons que P_i est vraie. Dans ce cas tous les agents vont effectuer les mêmes actions car il sont dans le même état. Il vont donc tous envoyer les mêmes messages à leur voisins de gauche et de droite.

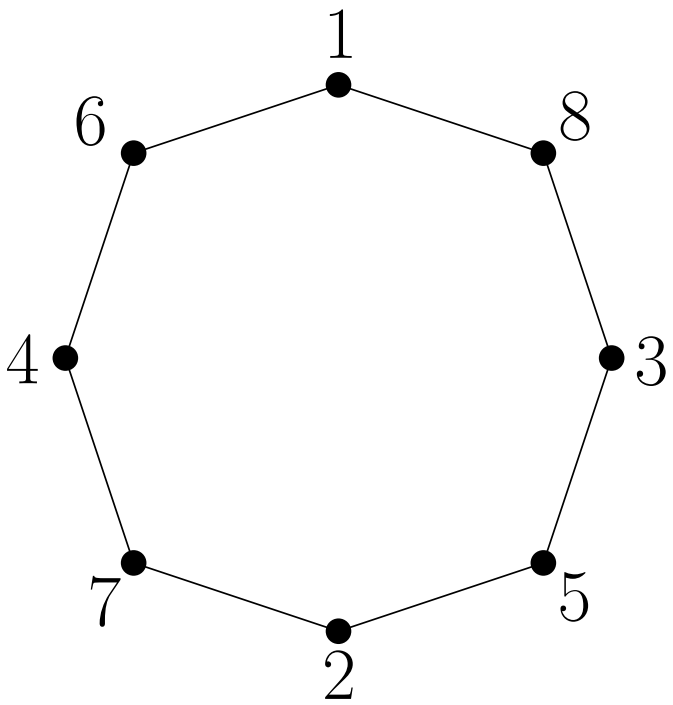
Par conséquent ils vont tous recevoir les mêmes messages des deux cotés et ils vont tous passer au même état. P_{i+1} est donc vraie.

Par récurrence, à chaque ronde, tous les processus sont dans le même état (symétrie des processus).

Supposons qu'un algorithme d'élection existe. Si l'un des processus est ELU alors tous les autres le sont aussi.

Processus Avec Identités

Avec identités uniques => Election est possible !



Processus Avec Identités

On suppose désormais que chaque processus possède un identifiant unique (UID).

Algorithme d'élection LCR

Pour un anneau unidirectionnel avec UID, sans connaissance de la taille.

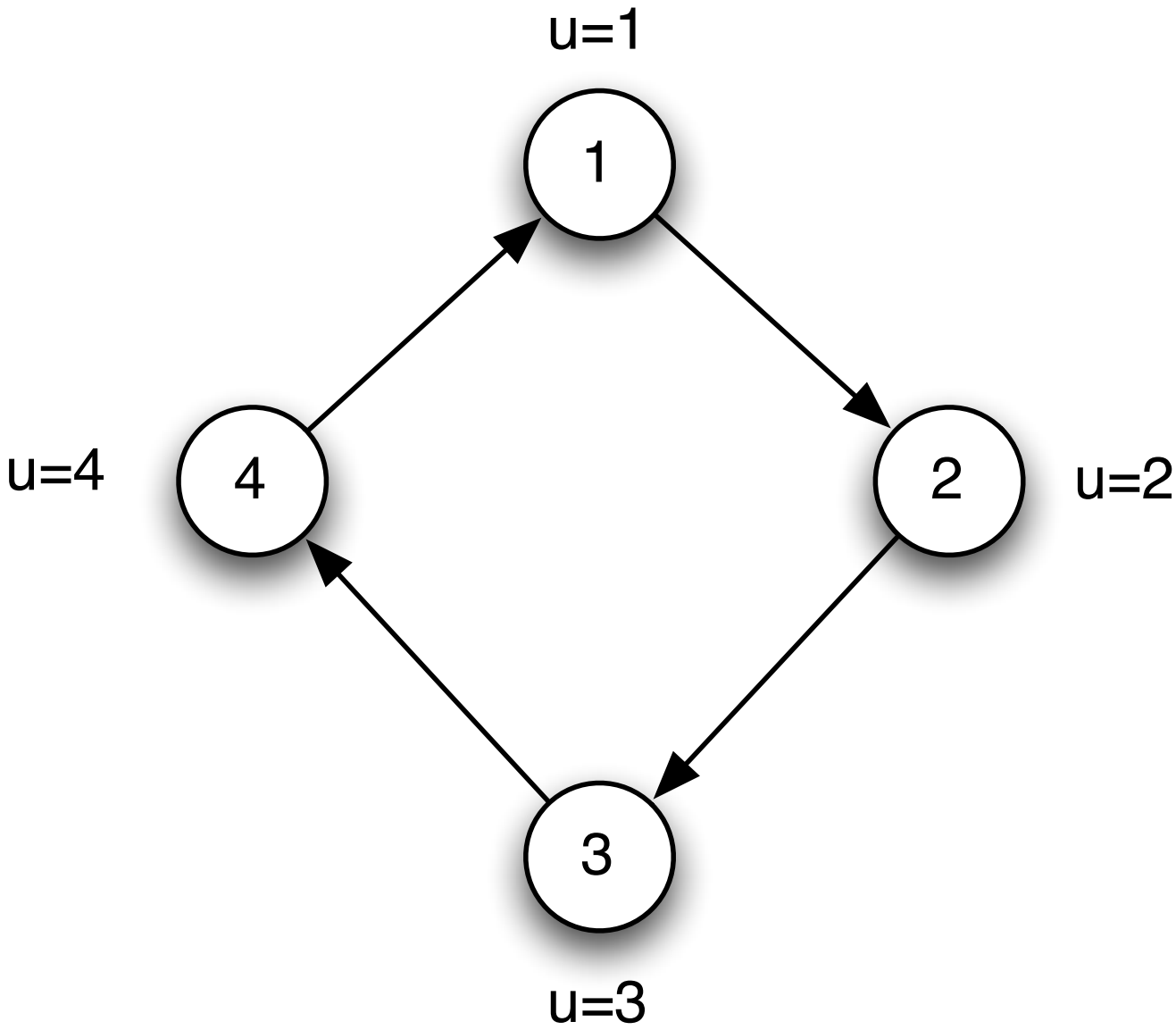
Unidirectionnel : envoi de message possible que sur un seul voisin (circuit)

LCR pour Le Lann, Chang et Roberts

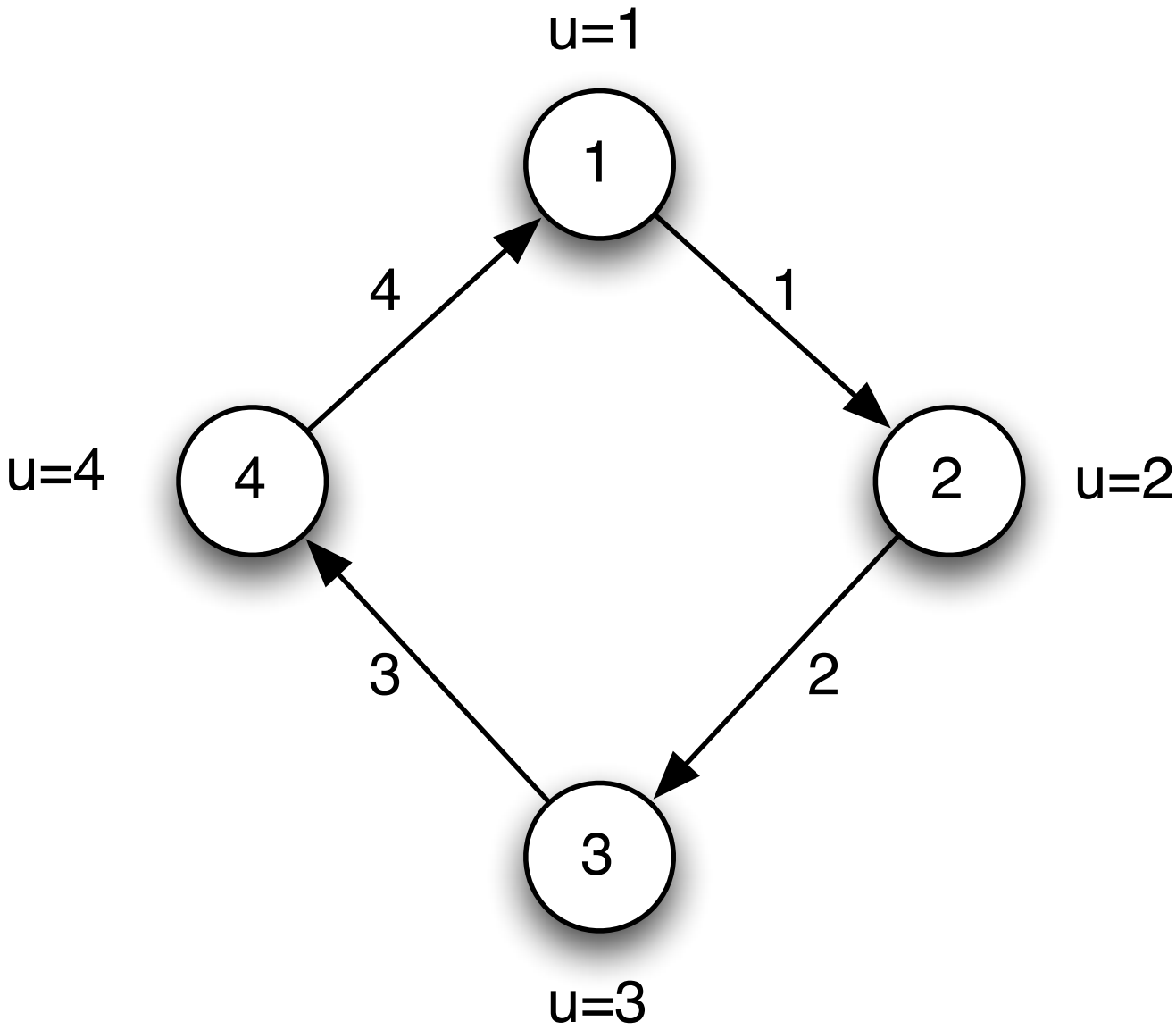
Algorithme LCR

```
etat = inconnu;
u = uid;
do {
   send(u);
   v = receive();
   if ( v > uid )
      u = v;
   if ( v < uid )
      u = null;
   if ( v == uid )
      etat = elu;
} while (etat == inconnu);
```

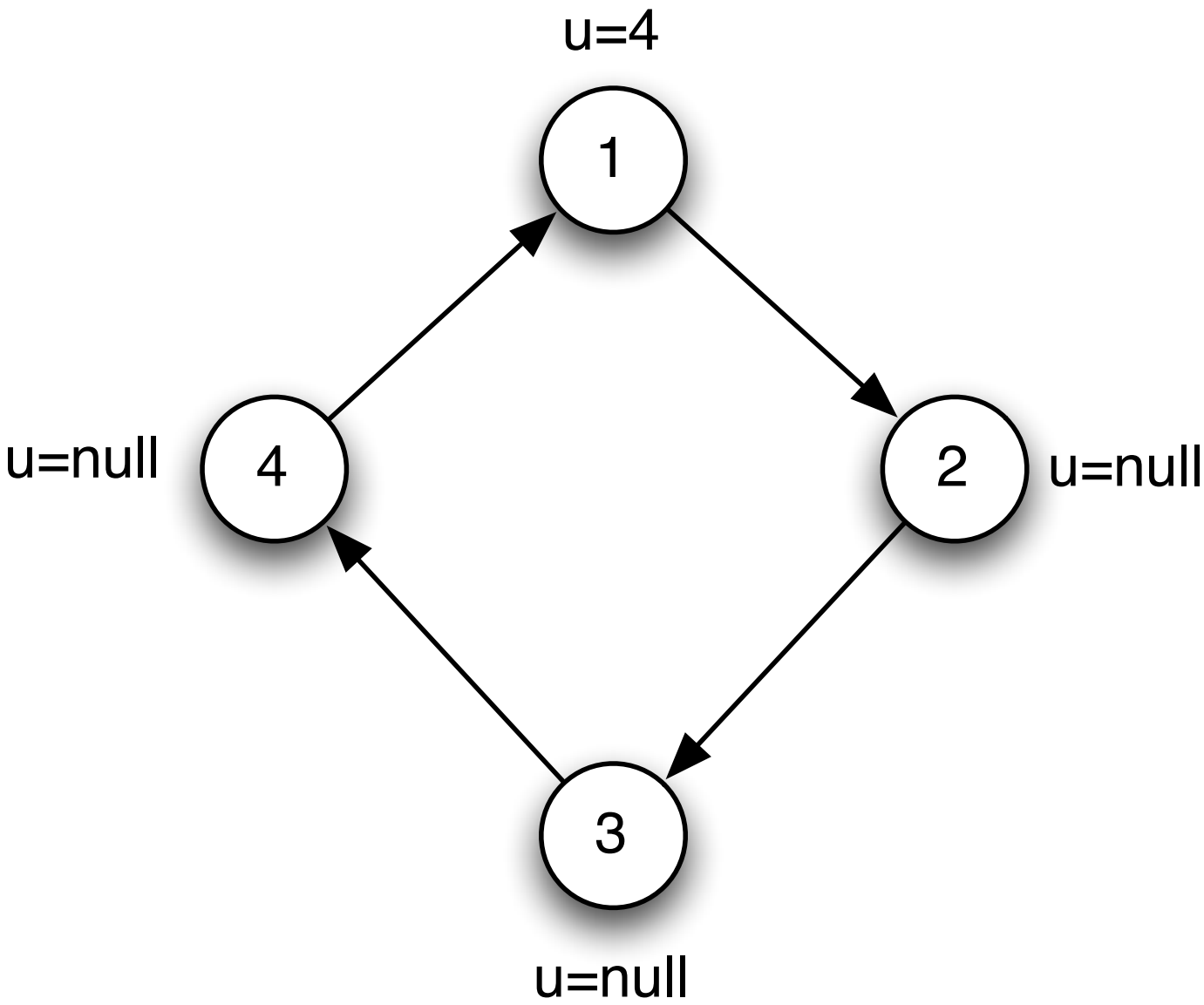
Execution de LCR



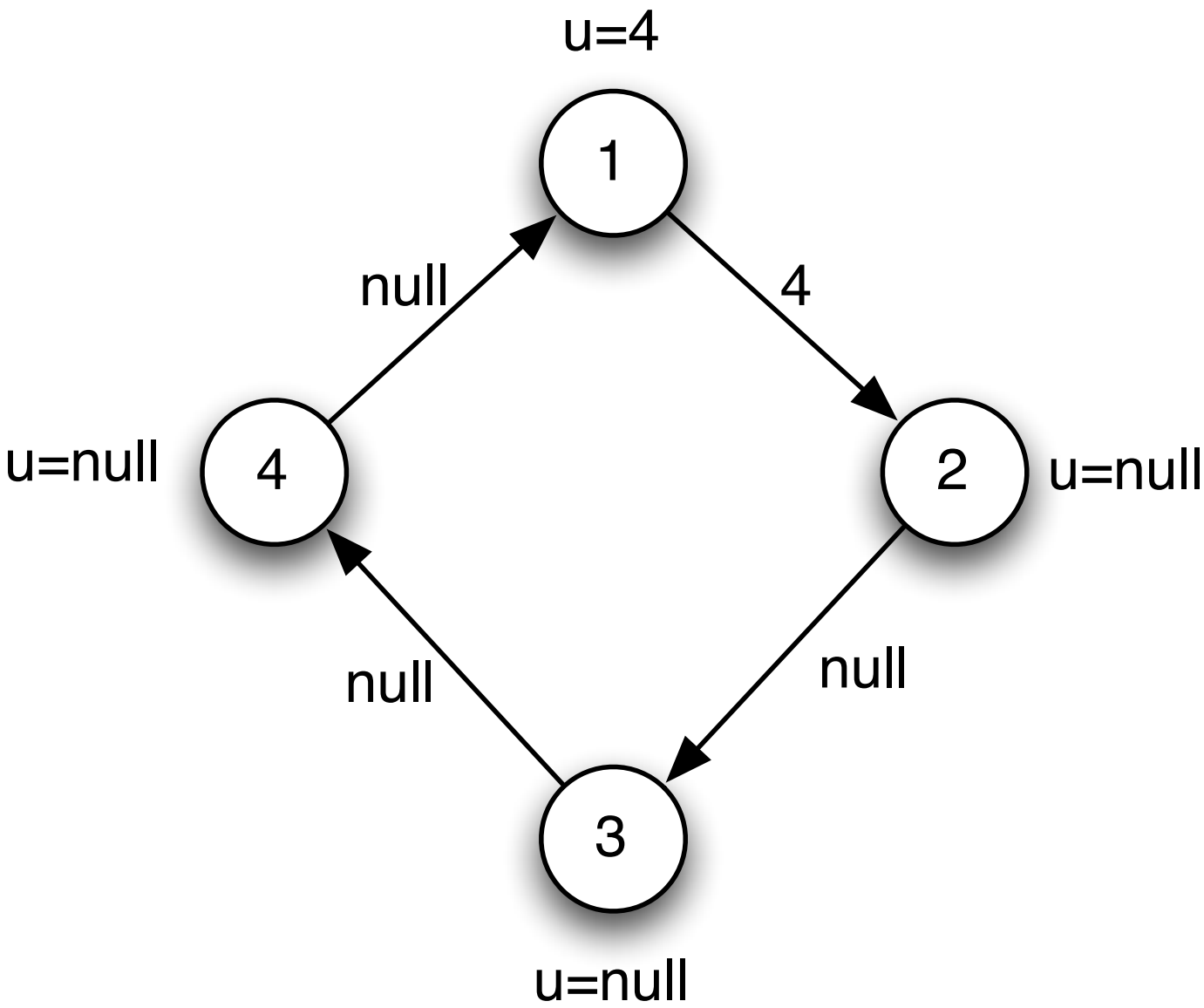
Execution de LCR



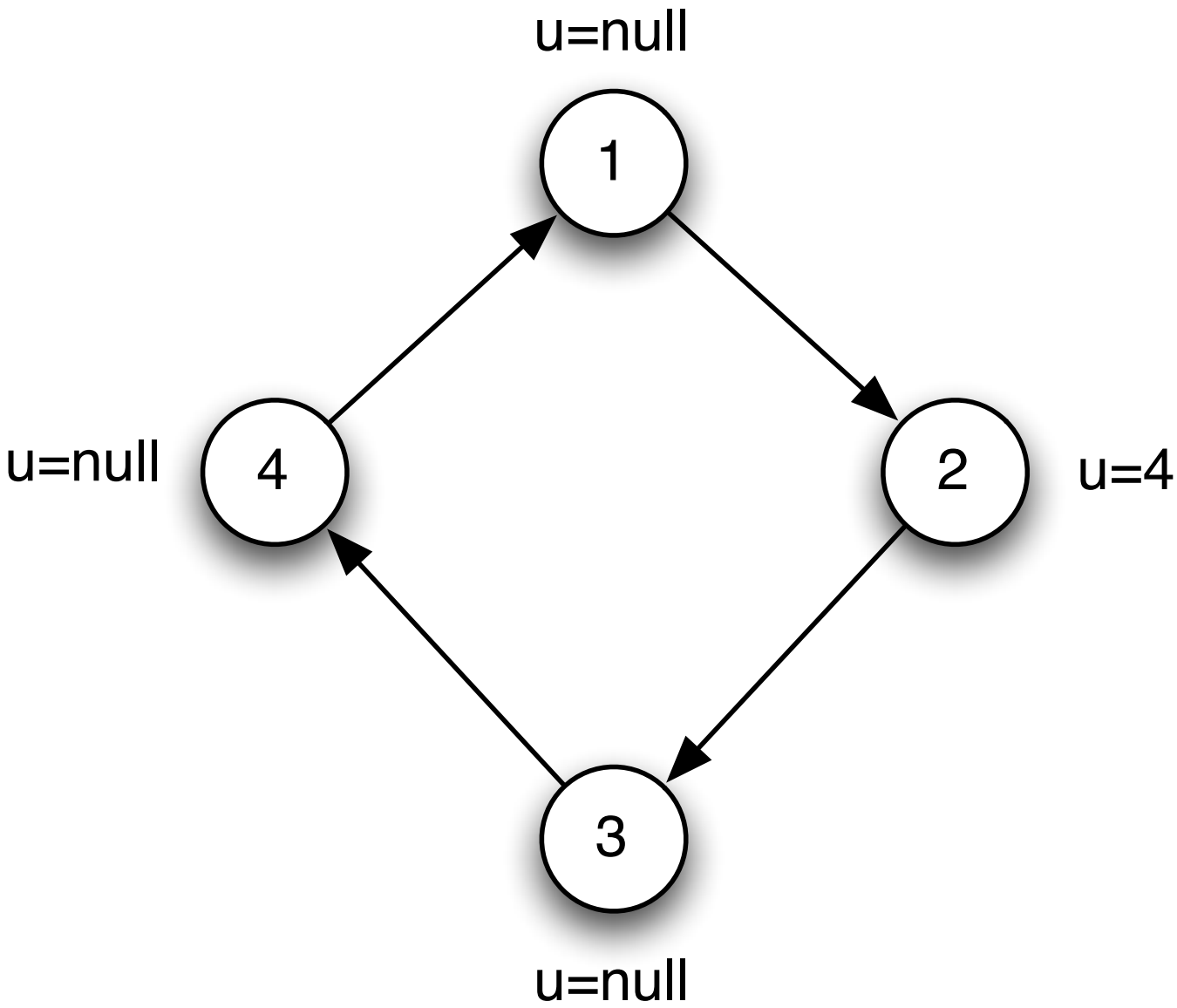
Execution de LCR



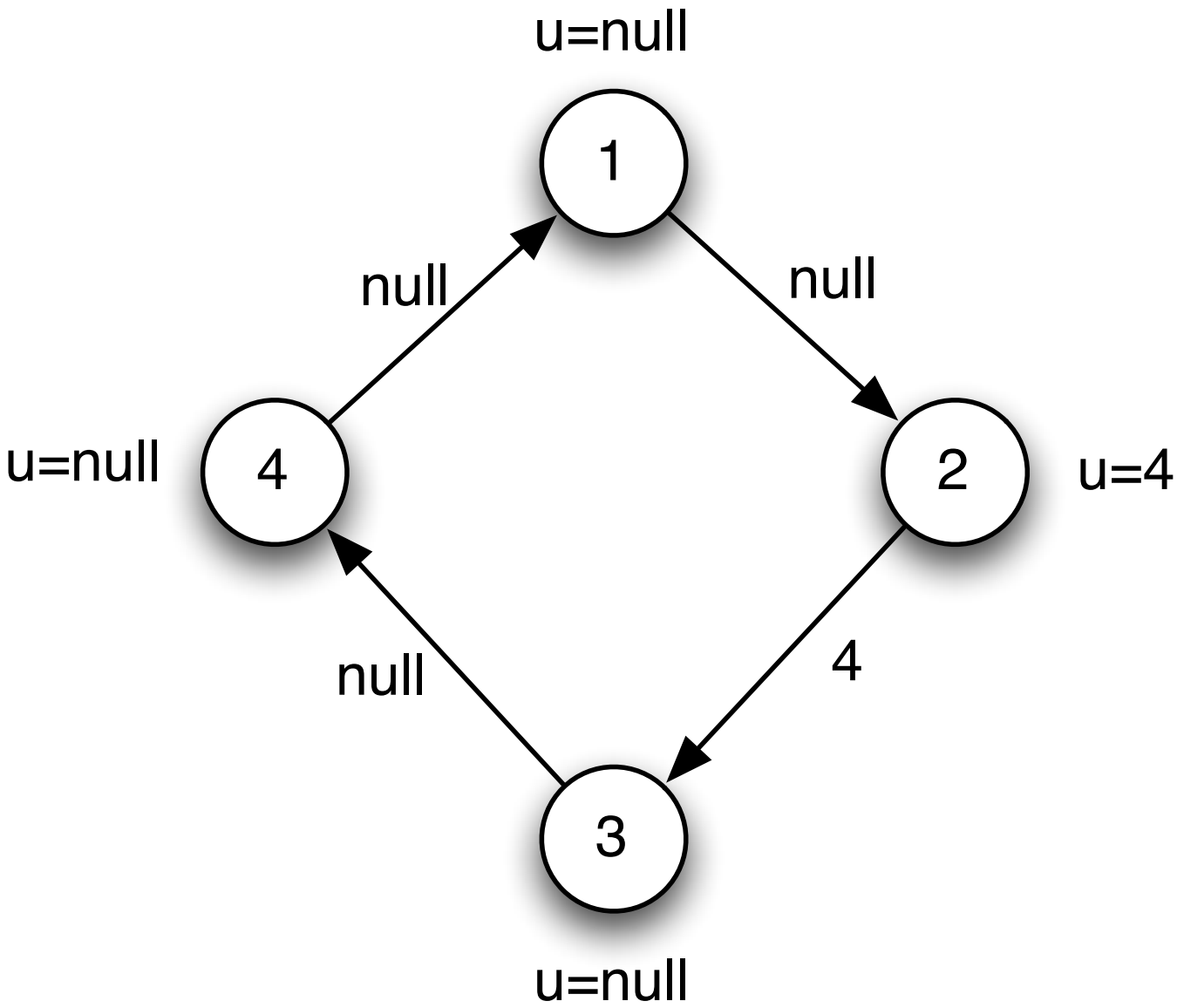
Execution de LCR



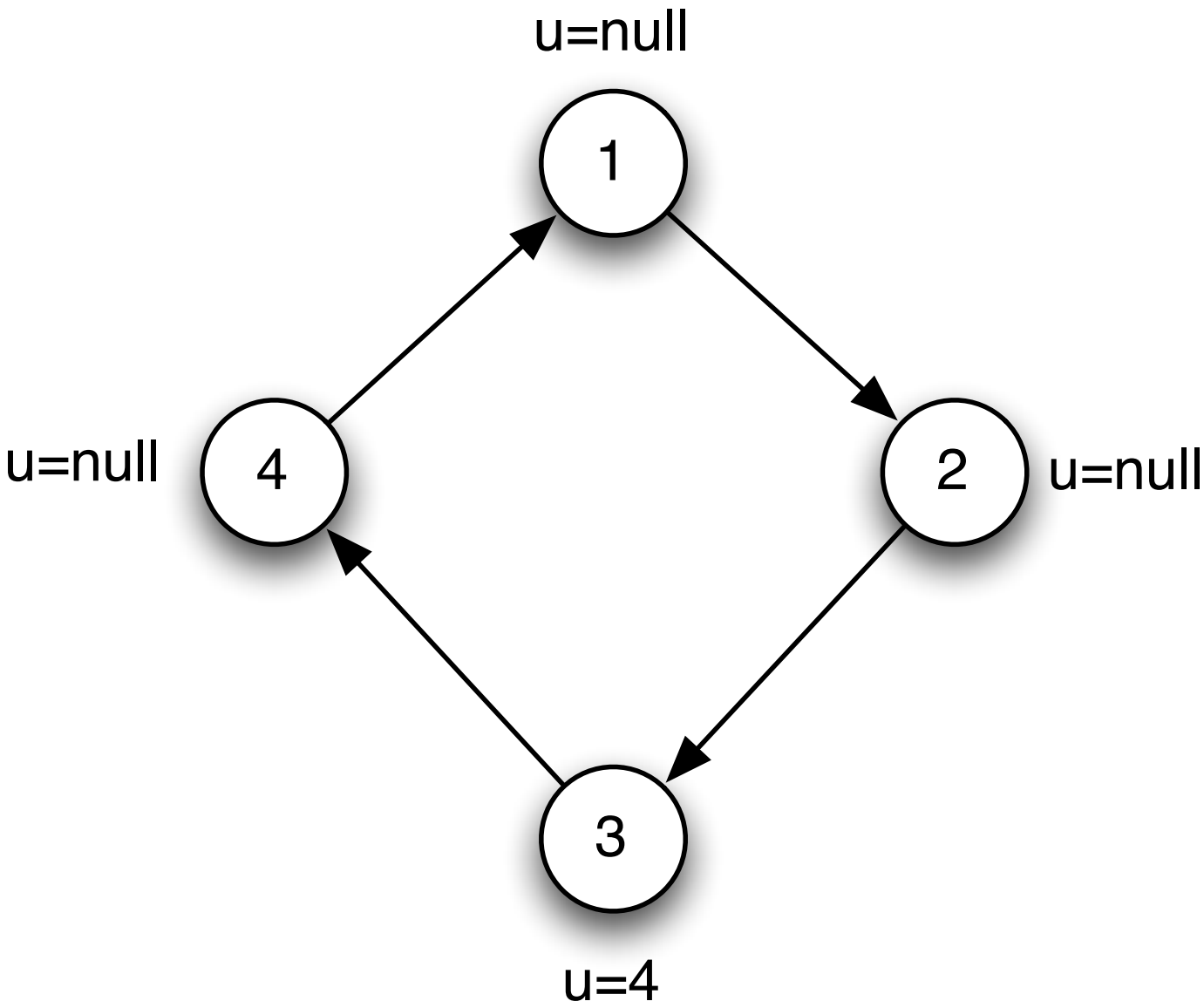
Execution de LCR



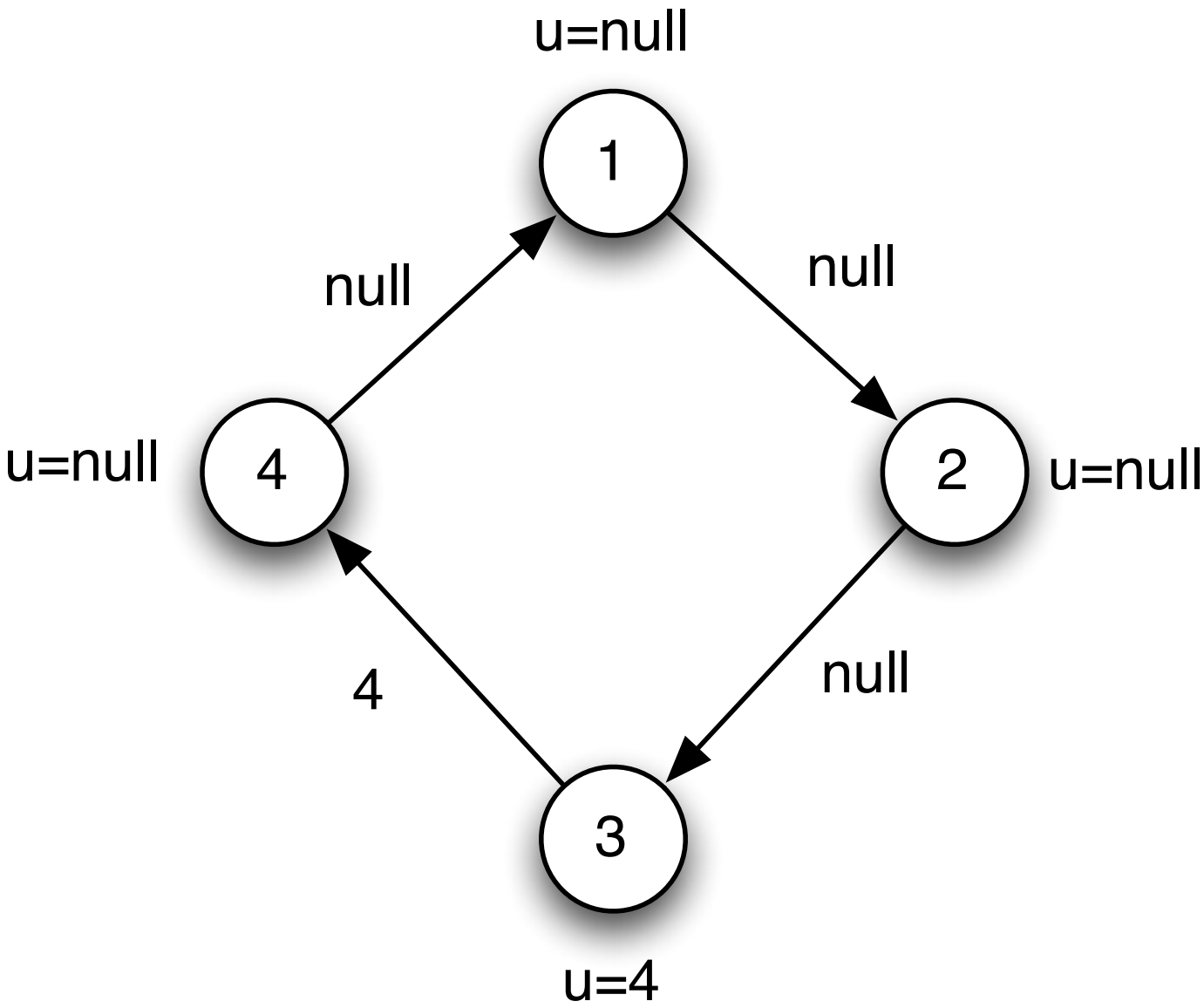
Execution de LCR



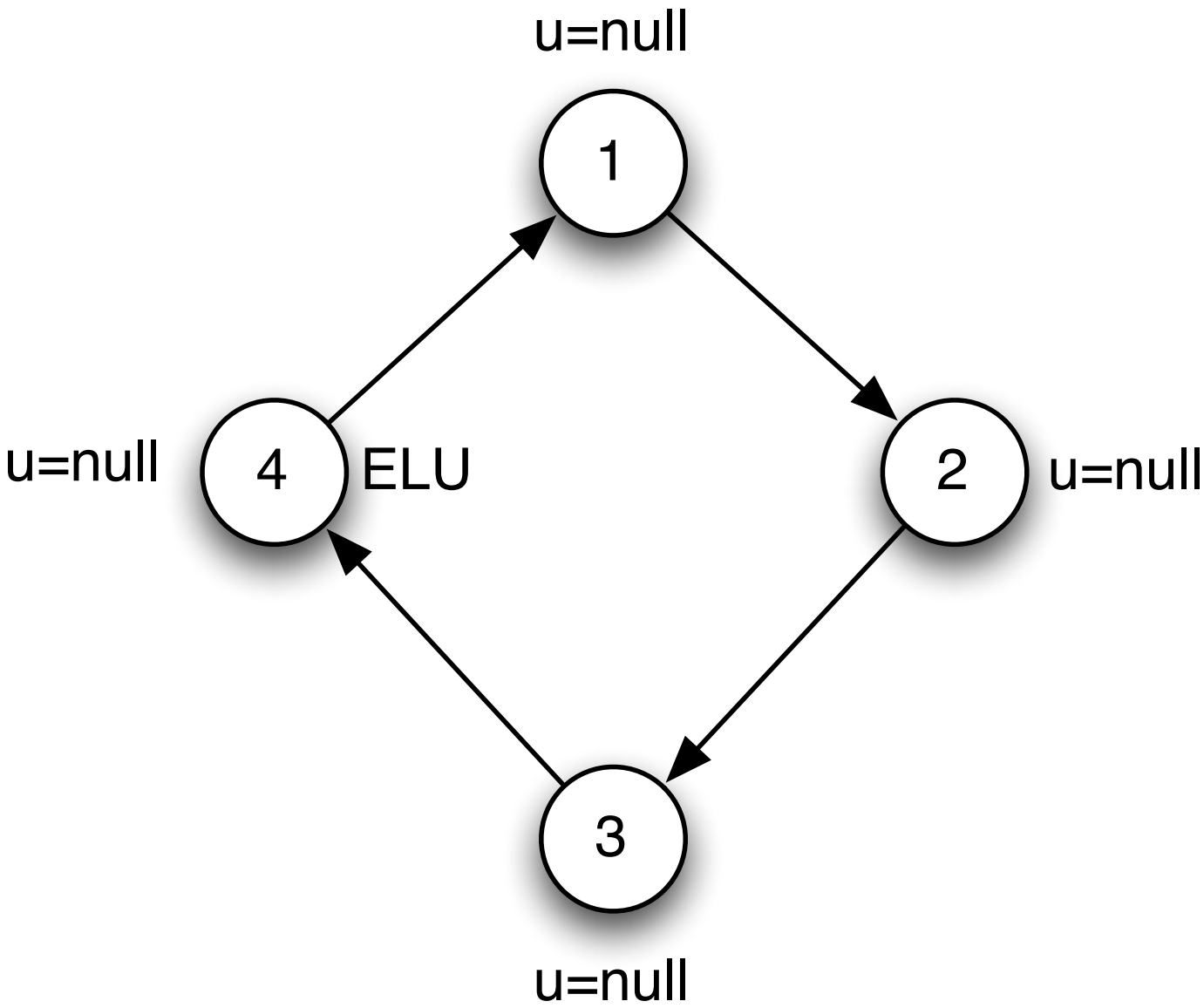
Execution de LCR



Execution de LCR



Execution de LCR



Preuve de Correction de LCR

Propriété

*Le processus d'*UID *maximal est dans l'état* ELU *à la n-ième ronde et c'est le seul.*

Preuve On note $i_{\max}$ l'indice du processus d'UID maximal.

- Pour toute ronde r avec $0 \leq r \leq n-1$, $u_{i_{\max}+r \mod n} = \text{UID}_{i_{\max}}$
- Après n rondes, $etat_{i_{\max}}$ =ELU,

Terminaison de LCR

Ainsi présenté, l'algorithme ne se termine pas pour les processus non-ELUs.
$\Longleftarrow$ ils sont bloqués dans une boucle infinie d'envoi et de réception de message vide.

- Chaque processus qui voit passer un UID plus grand que le sien peut se mettre dans l'etat non-ELU (mais ne sait pas mieux quand s'arrêter)
- Le processus ELU peut envoyer un message CestFini pour indiquer à chaque autre noeud la terminaison

Complexité de LCR

- sans détection de la terminaison :
  - en temps : n rondes
  - en espace : $O(n^2)$ messages
- avec détection de la terminaison locale :
  - en temps : $2n$ rondes
  - en espace : $O(n^2)$ messages

Pire des cas quand les UIDs sont en ordre décroissant.

Complexité de LCR

- sans détection de la terminaison :
  - en temps : n rondes
  - en espace : $O(n^2)$ messages
- avec détection de la terminaison locale :
  - en temps : $2n$ rondes
  - en espace : $O(n^2)$ messages

Pire des cas quand les UIDs sont en ordre décroissant.

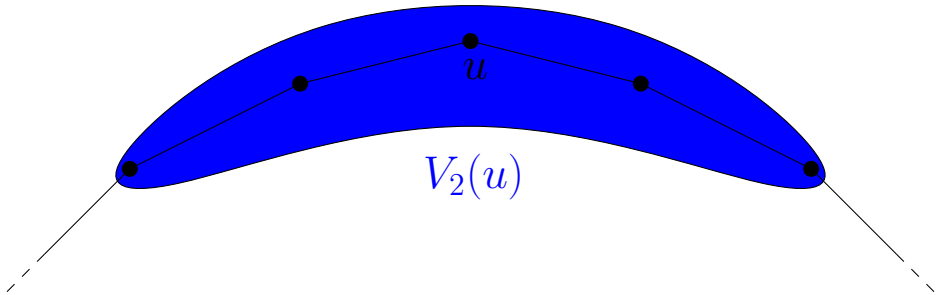
Existe-il un algorithme plus efficace ?

Élection en Phases : $O(n \log n)$ messages

Un algorithme plus efficace existe.

On considère l'anneau bidirectionnel.

On définit le voisinage de taille d d'un processus noté $V_d(u)$ comme étant tous les processus à distance $\leq d$ de u (d processus à droites et d à gauche)



Élection en Phases : principe

Principe

A la phase k, le processus se compare à ses voisins actifs à distance 2^k : $V_{2^k}(u)$. S'il a le plus grand identifiant, il reste actif. Sinon il devient passif et ne compte plus pour les comparaisons suivantes

Phase 0 : on se compare aux voisins à distance 1
Phase 1 : on se compare aux voisins à distance 2
Phase 2 : on se compare aux voisins à distance 4
Phase 3 : on se compare aux voisins à distance 8

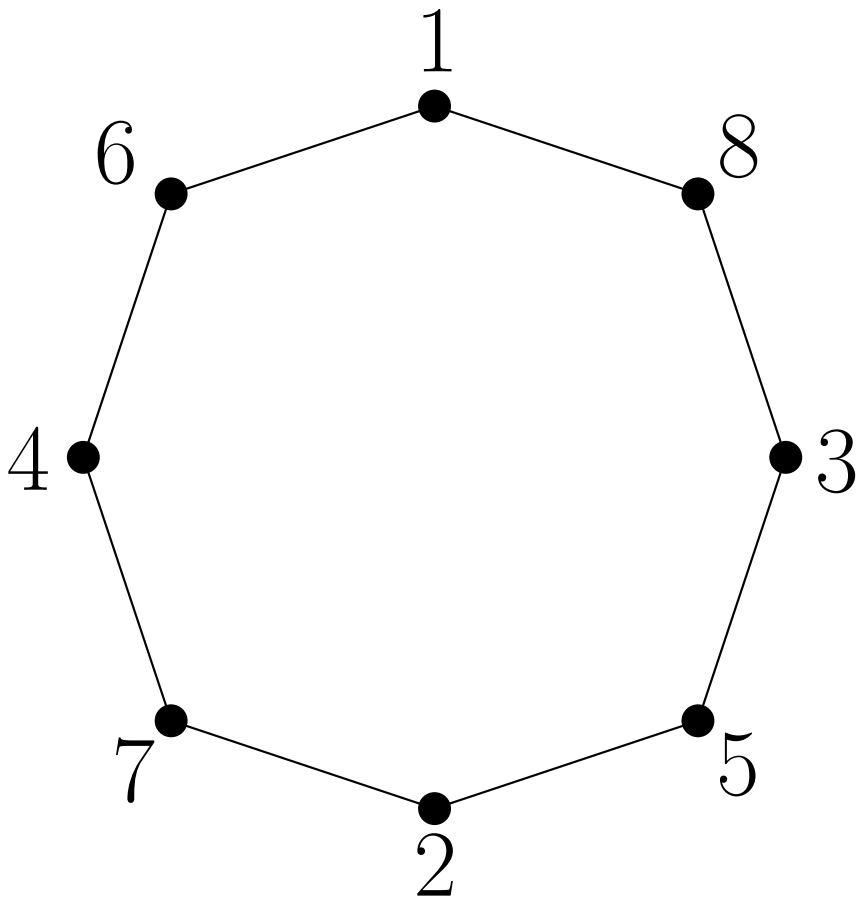
Élection en Phases : code

To initiate an election (phase 0) :
send(ELECTION$< my_id, 0, 0 >$) to left and right ;

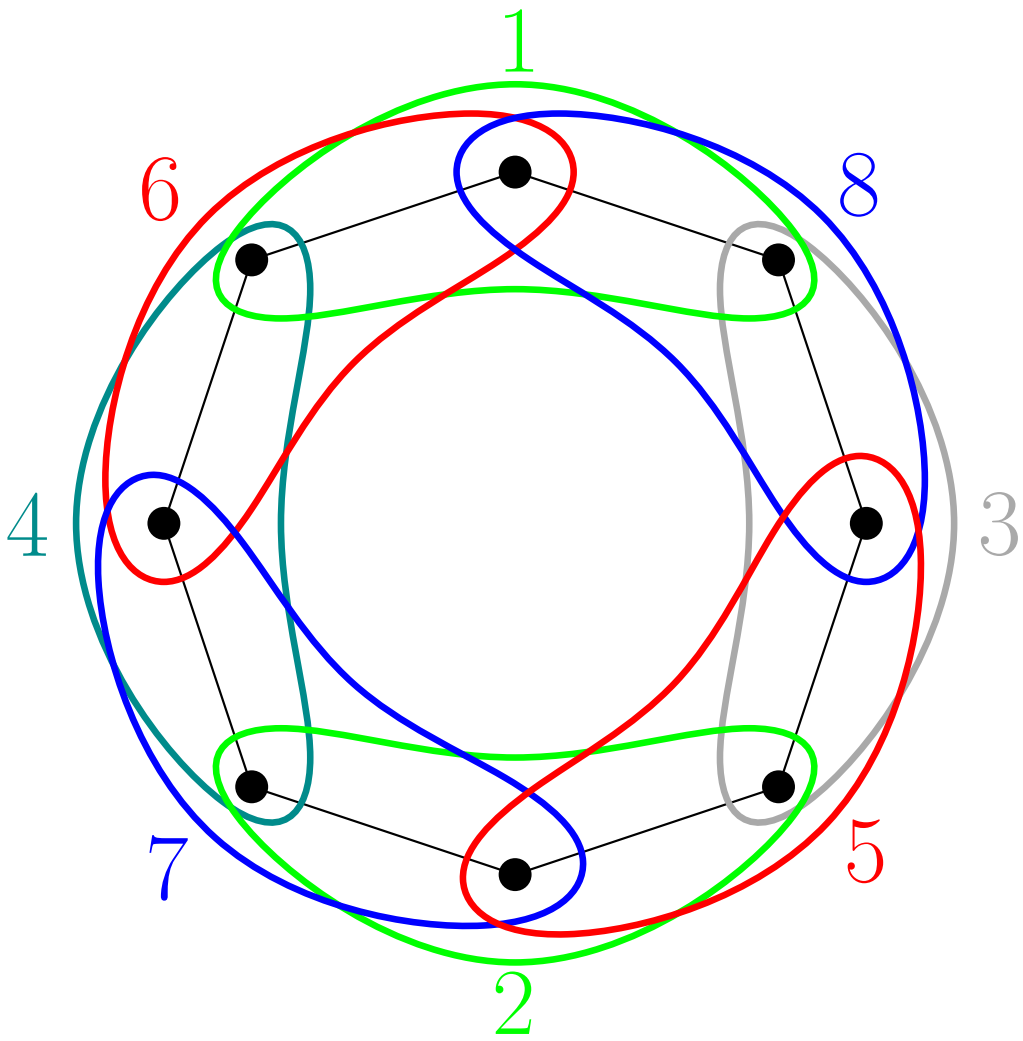
Upon receiving a message ELECTION$< j, k, d >$ **from left (right) :**
if $((j > my\_id) \wedge (d \leq 2^k))$ then
    send(ELECTION$< j, k, d + 1 >$) to right (left) ;
if $((j > my\_id) \wedge (d = 2^k))$ then
    send(REPLY$< j, k >$) to left (right) ;
if $(my\_id = j)$ then announce itself as leader ;

Upon receiving a message REPLY$< j, k >$ **from left (right) :**
if $(my\_id \neq j)$ then
    send(REPLY$< j, k >$) to right (left) ;
else
    if (already received REPLY$< j, k >$)
        send(ELECTION$< j, k + 1, 1 >$) to left and right ;

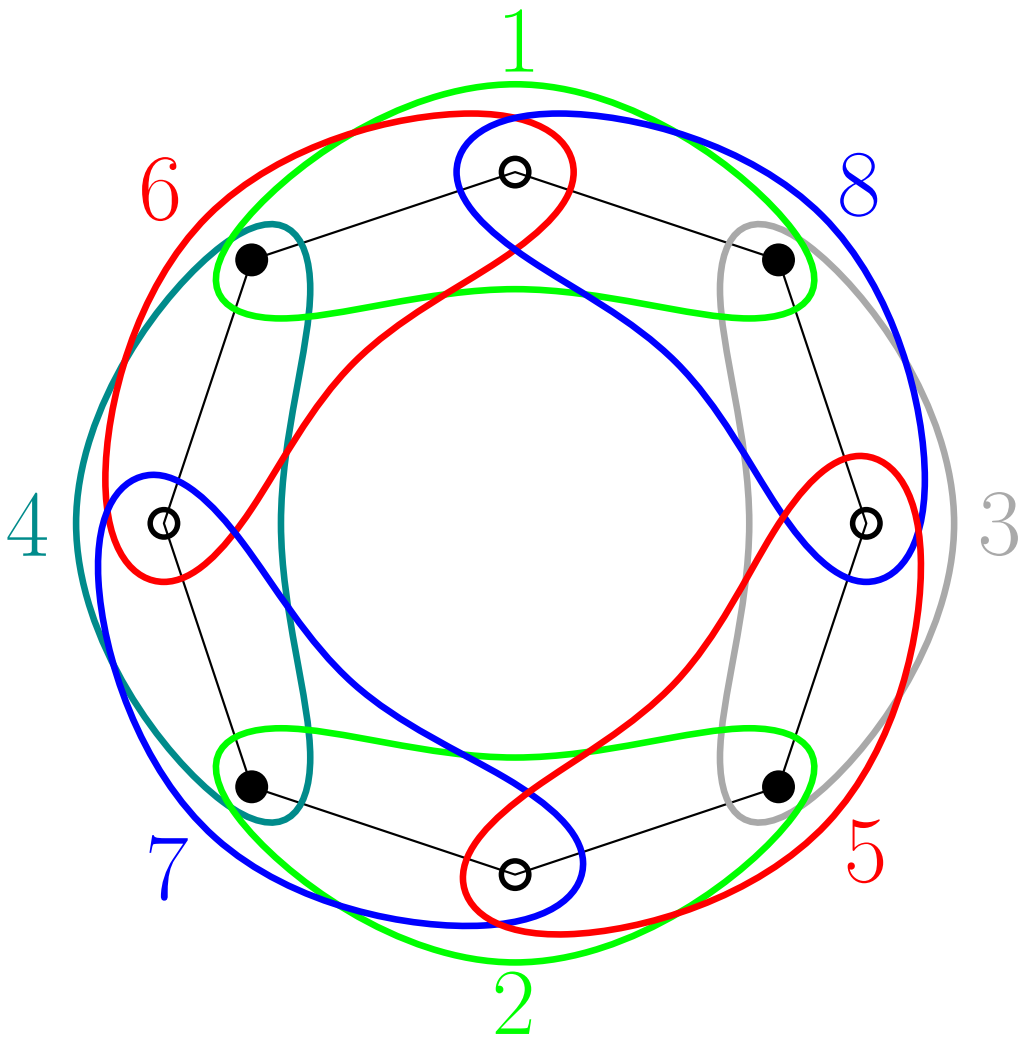
Exemple d'exécution



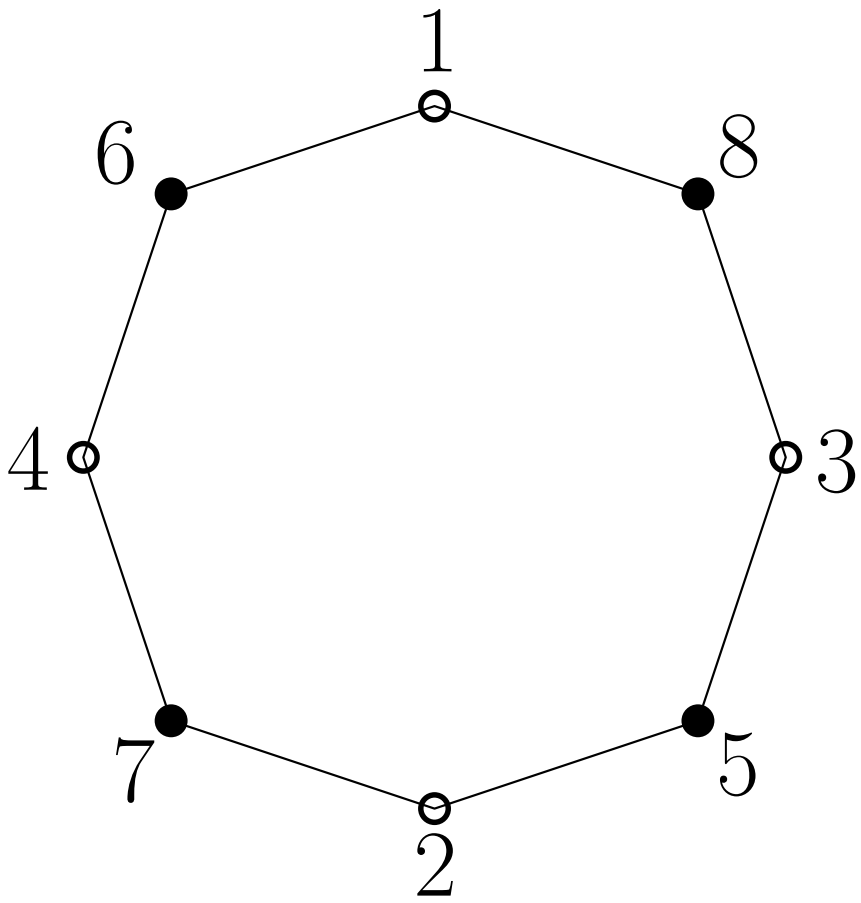
Exemple d'exécution



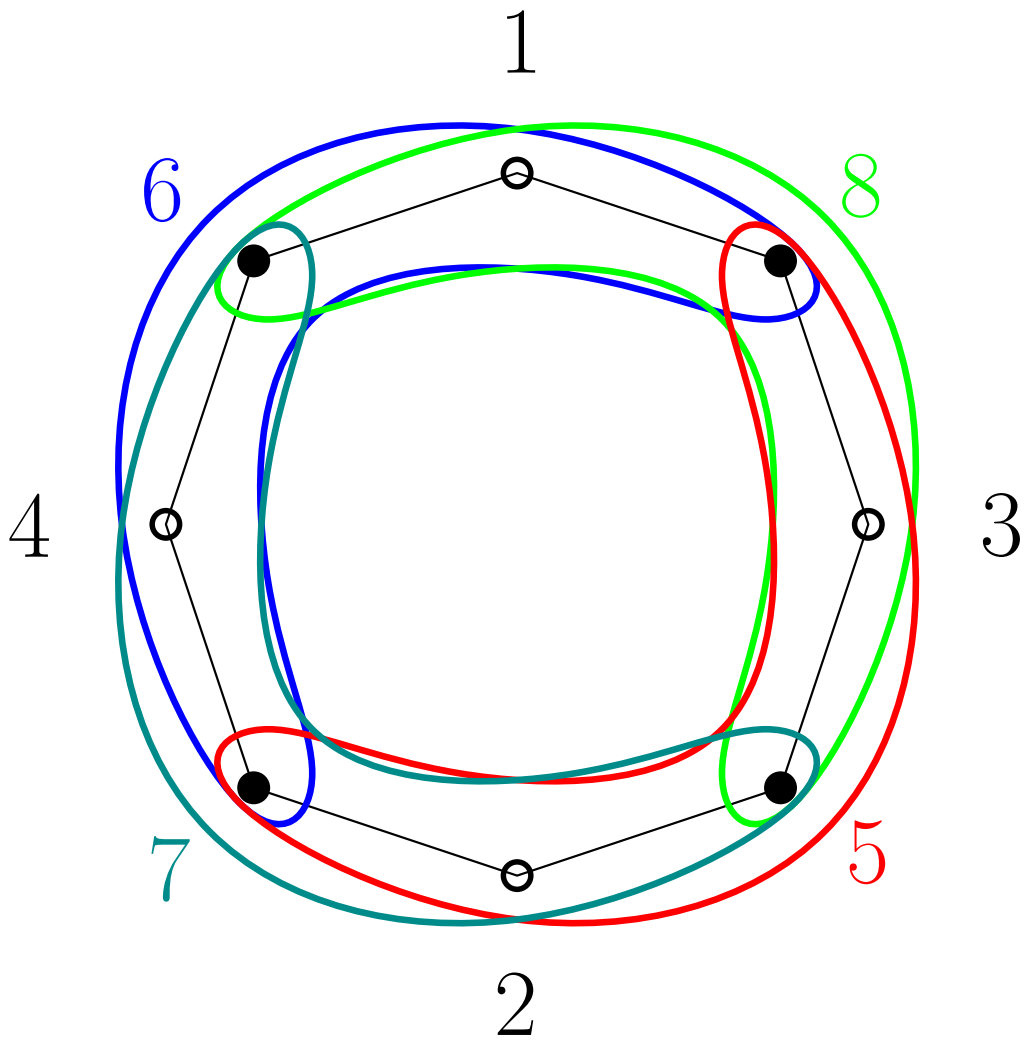
Exemple d'exécution



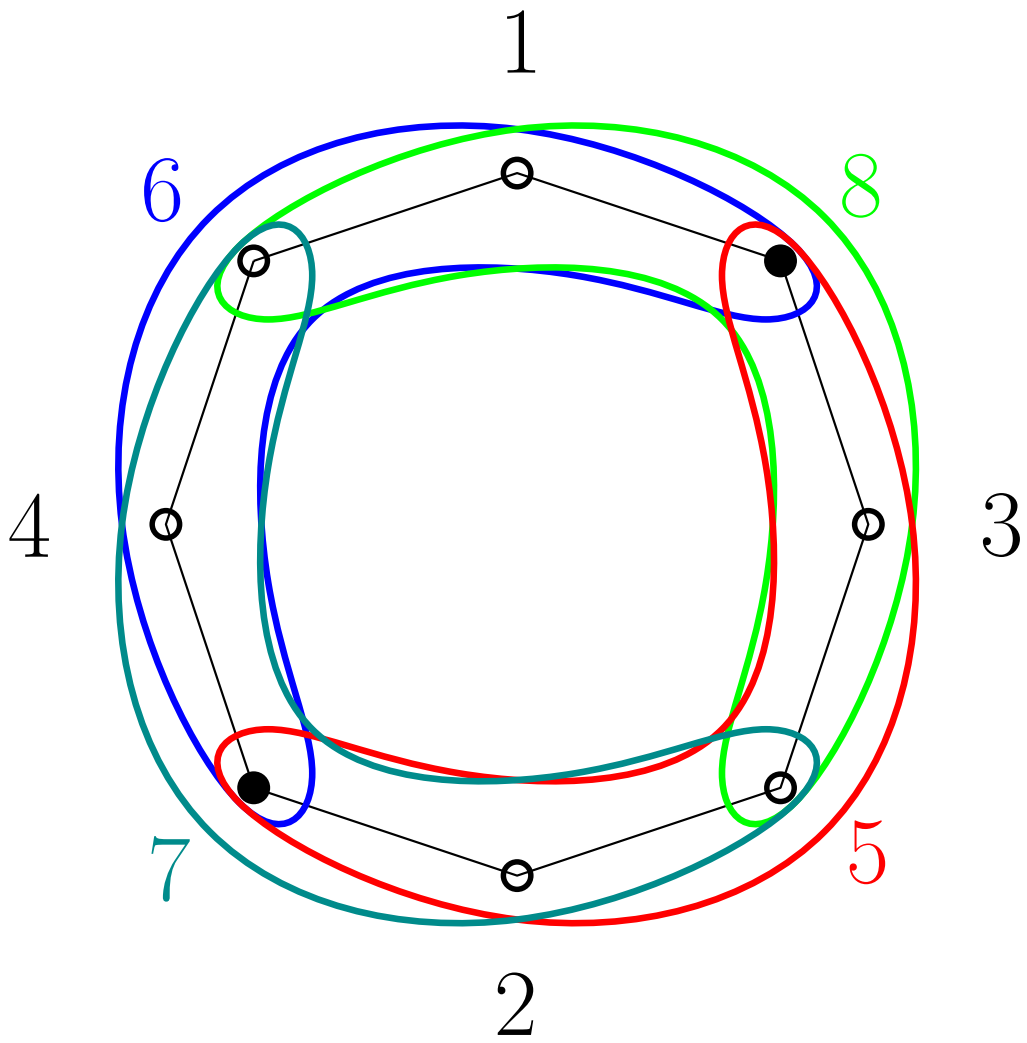
Exemple d'exécution



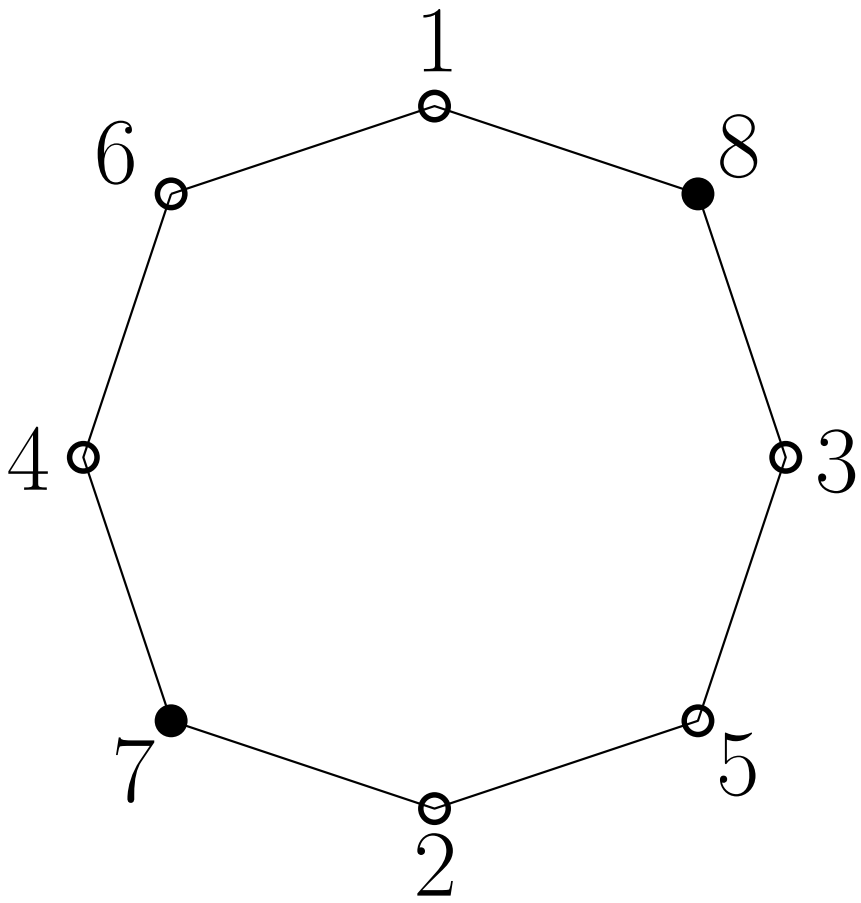
Exemple d'exécution



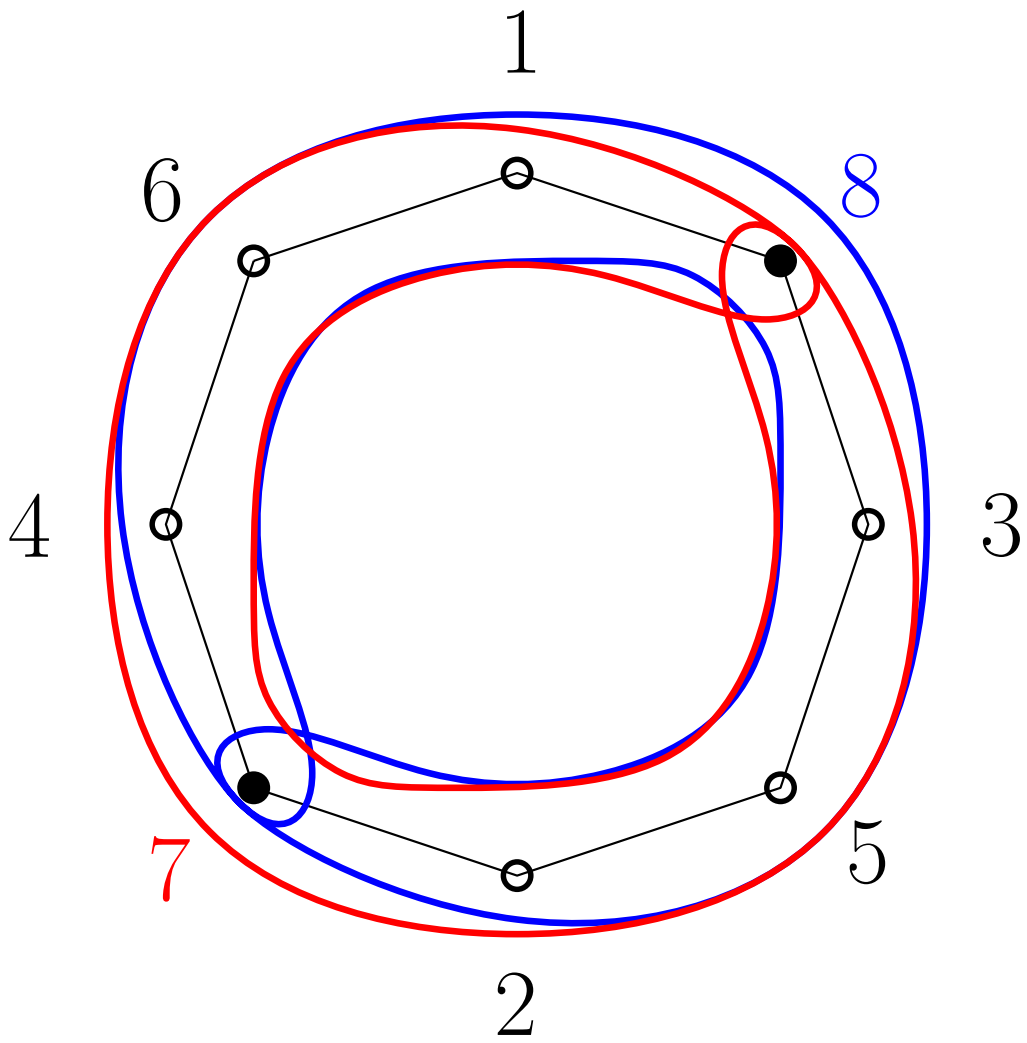
Exemple d'exécution



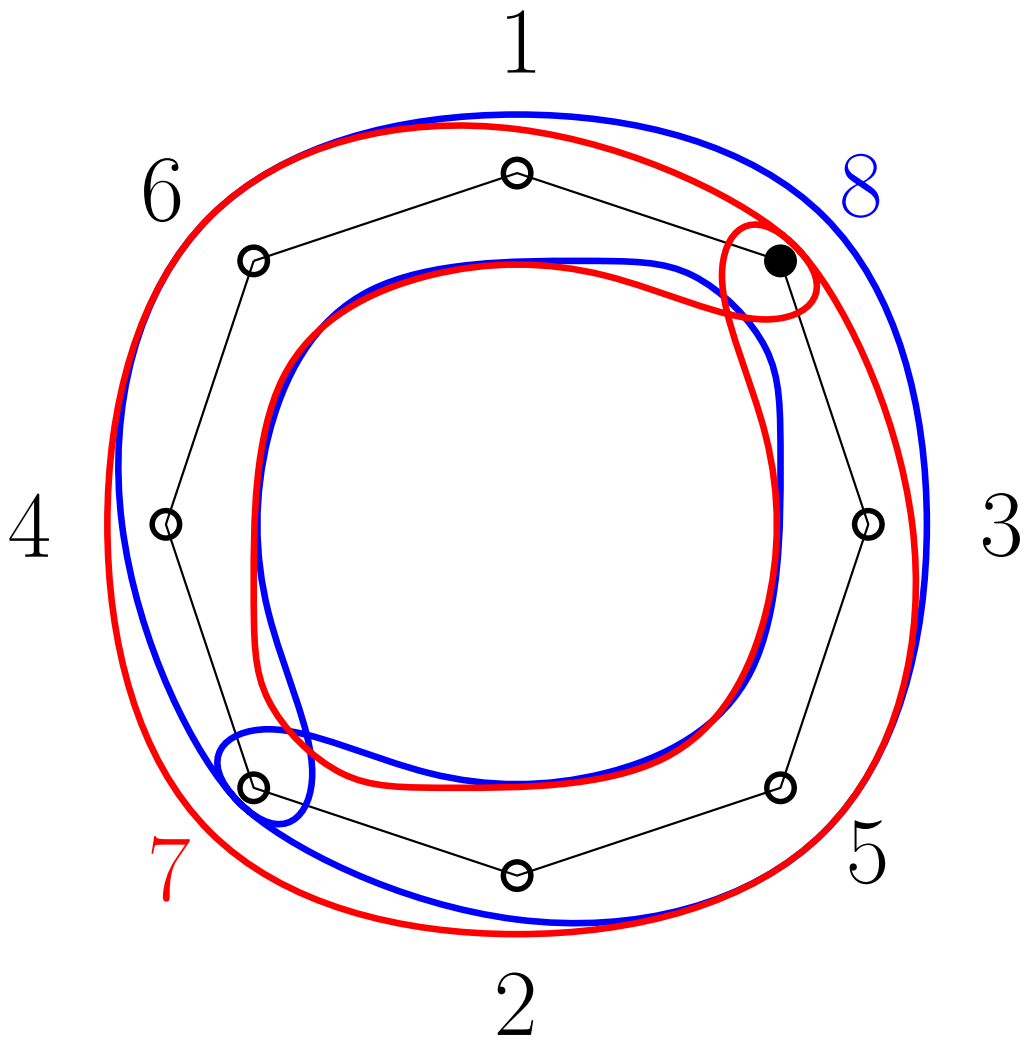
Exemple d'exécution



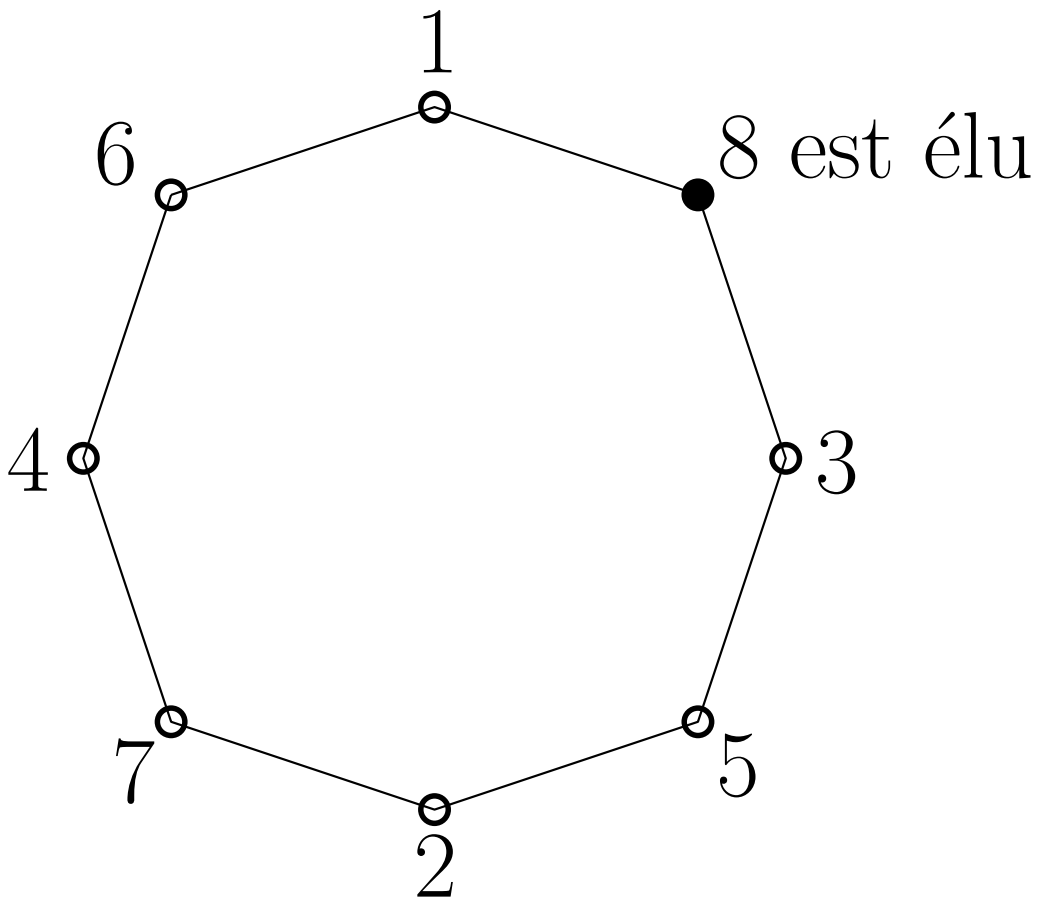
Exemple d'exécution



Exemple d'exécution



Exemple d'exécution



Élection en Phases : analyse

Lemme

A la phase k au plus 4.2^k messages sont envoyés pour un candidat actif.

Lemme

Pour chaque $k \geq 1$, le nombre de processeurs actifs est au $\left\lfloor \frac{n}{2^{k-1}+1} \right\rfloor$.

Preuve : La distance minimum entre deux processus actifs à la fin de la phase $k-1$ est $2^{k-1}+1$.

Élection en Phases : analyse

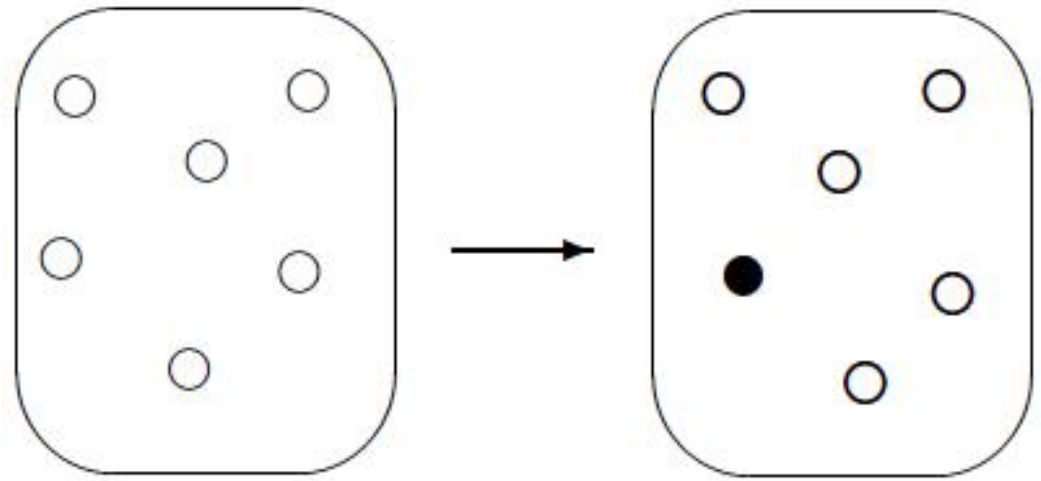
- Le nombre total de phases avant élection est $\lceil \log n \rceil + 1$.
- Le nombre total de messages est :

$$\underbrace{4n.2^0}_{\text{phase 0}} + \underbrace{\sum_{k=1}^{\lceil \log n \rceil + 1} \left\lfloor \frac{n}{2^{k-1}+1} \right\rfloor (4.2^k)}_{\text{Phase 1 à } \lceil \log n \rceil + 1} = \sum_{k=1}^{\lceil \log n \rceil + 1} O(n)$$

$$= O(n \log n)$$

Election dans un graphe Quelconque

Un graphe arbitraire G (avec identités uniques pour chaque processus)



Comment faire l'élection (sans connaissance du G) ?

Dans un Réseau Quelconque

Comment élire dans un graphe quelconque avec UIDs et connaissance d'une borne Δ sur le diamètre ?

$\Longrightarrow$**Algorithme d'Inondation** Chaque noeud commence par diffuser son UID, puis le maximum des UIDs reçus. L'algorithme dure Δ rondes.

- Si l'UID maximal est celui du processus alors il se met dans l'état ELU
- Sinon, le processus se met dans l'état non-ELU

Correction de l'Algorithme d'Inondation

La correction repose sur l'invariant suivant :
Après r rondes, si $d(i, i_{\max}) \leq r$ alors

$$\max\{\text{UID reçu en } i\} = \text{UID}_{i_{\max}}.$$

Terminaison de l'Algorithme d'Inondation

Tous les processus ont terminé au bout de Δ rondes. (et sont soit dans l'état ELU, soit dans l'état non-ELU)

Complexité de l'Algorithme d'Inondation

- En temps : Δ rondes
- En nombre du messages : $\Delta \times$ nb d'arêtes du G

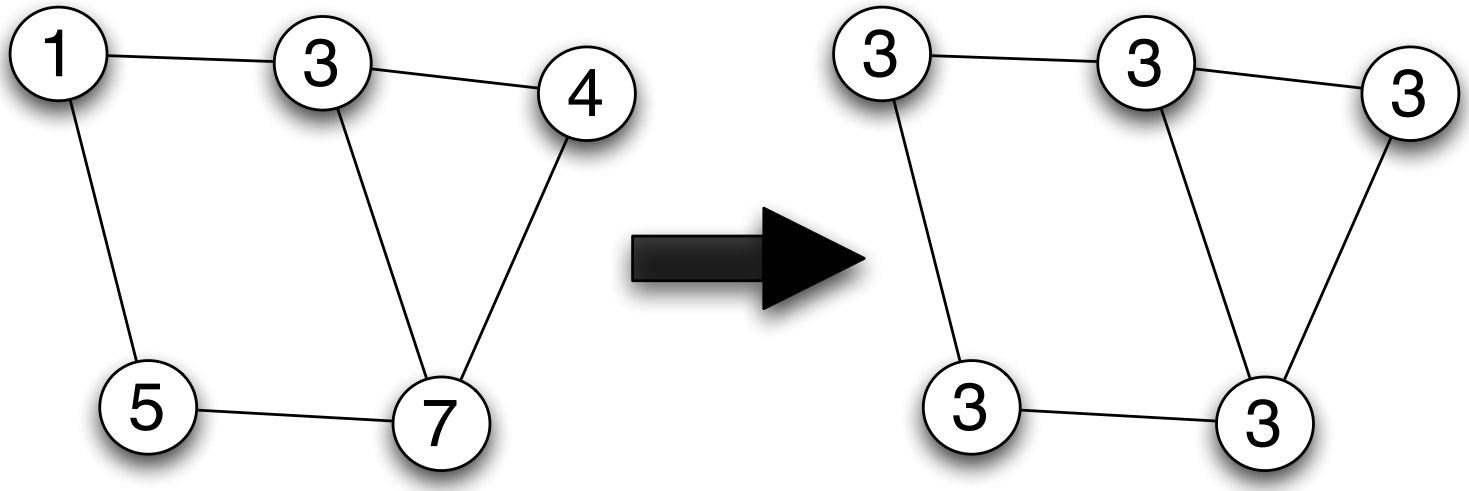
Pour Conclure Provisoirement

- Il nous a fallu la connaissance d'une **borne sur le diamètre** pour élire dans un réseau quelconque
- Avec LCR, il est possible d'élire dans un anneau orienté sans connaissance du taille
- Est-il possible d'élire dans un réseau synchrone quelconque avec UID, sans autre connaissance ou information structurelles ?

Tolerance aux pannes

Algorithmique distribuée
pour
les systèmes avec défaillances

Problème du consensus



Problème du Consensus

Spécification

Etant donnés des processus avec une valeur initiale quelconque,

- **Accord** Tous les processus qui décident une valeur décident la même valeur.
- **Validité** Si tous les processus ont w comme valeur initiale, alors c'est la valeur décidée.
- **Terminaison** Tous les processus (non défaillants) décident.

Consensus avec Défaillances

S'il n y a pas de défaillances :
Il *suffit* d'échanger les valeurs initiales et d'en choisir une par une méthode déterministe (maximum par exemple).

Quand il y a défaillances ...

- de liens
- de processus
- byzantines

L'histoire des deux généraux

Deux armées rouges surplombent une armée bleue de part et d'autre d'une vallée. Les forces en présence sont telles que si une seule armée rouge attaque, elle perdra la bataille. Par contre, si les deux attaquent simultanément, alors c'est l'armée bleue qui perd.

Les armées rouges communiquent en envoyant des messagers qui doivent traverser la vallée et peuvent donc être capturés par l'armée bleue.

Comment les généraux rouges peuvent-ils se mettre d'accord pour lancer éventuellement l'attaque ou renoncer ?

Il n'y a pas de solution (déterministe)

Les deux généraux de manière plus formelle

Le problème

On considère deux processus A et B.
Le processus A a un registre d'entrée I_A égal à 0 (pas d'attaque) ou 1 (attaque).
Les deux processus A et B ont chacun un registre de sortie O_A et O_B initialisé à $\perp$.
Le but des deux processus est d'écrire la valeur I_A dans leur registre de sortie (O_A ou O_B). Il n'ont le droit qu'à une écriture (une fois décidé il est impossible de changer d'avis).
Le système est **synchrone** mais des messages peuvent se perdre.

Théorème

Le problème des deux généraux n'a aucune solution déterministe.

Preuve d'mpossibilité I

Preuve :

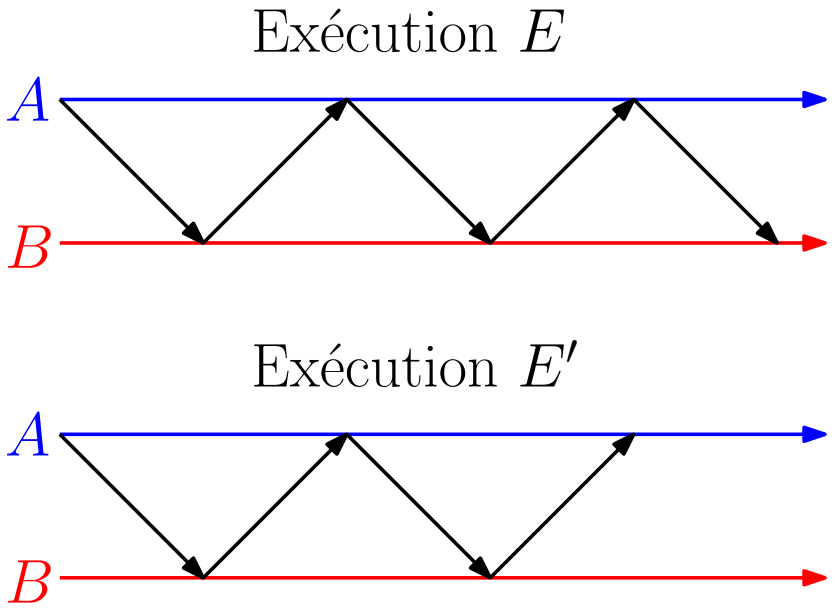
On peut observer que dans une exécution ou les généraux décident 1 (cas $I_A = 1$), au moins un message doit être reçu. En effet, dans le cas contraire, B ne peut pas distinguer le cas où $I_A = 1$ de celui où $I_A = 0$. Le lien doit donc laisser passer au moins un message avant de tomber en panne.

Supposons par contradiction qu'il existe un algorithme résolvant le problème. Parmi toutes les exécutions dans lesquelles les deux processus décident 1, prenons l'exécution E la plus plus courte (en nombre de messages reçus) qui est de longueur $k \geq 1$.

Preuve d'mpossibilité II

Sans perte de généralité, on peut supposer que le dernier (k-ième) message reçu est de A vers B. On note t le numéro de ronde pour lequel A et B ont décidé 1.

On considère maintenant une execution E' qui est identique à E sauf que le k-ième message est perdu.



Preuve d'mpossibilité III

Le général A ne peut différencier E de E'. Puisque A attaque dans E il doit aussi attaquer dans E'. La valeur O_A est donc fixée à la ronde t dans E'.

Puisque A attaque dans E', B doit aussi attaquer dans E'.

Si $k - 1 > 0$ alors l'exécution E' est plus courte que E et contient au moins un message. Cela contredit le fait que E est l'exécution la plus courte qui décide 1.

Si $k - 1 = 0$ alors il n'y a pas de message reçu dans E' et malgré cela A et B décident 1. Contradiction avec notre remarque initiale.

Autres Modèles de fautes

Question

1. Si on ne peut perdre qu'un seul message à la fois,
ou bien
2. Si un seul des deux processus peut perdre ses messages,

Y a t-il un solution pour consensus ?