Algorithmique Distribuée

Stéphane Devismes

Infos pratiques

- Contact : Stephane.Devismes@imag.fr
- Web : www-verimag.imag.fr/~devismes/WWW/enseignements.html
- Contact promo : Ricm5-reseau@googlegroups.com
- Salles de Cours/TD/TP, voir planning sur ADE : ade6-ujf-ro.grenet.fr/direct/

But

- **Algorithmes distribués** :
  - Spécification
  - Conception
  - Preuve de correction et complexité
  - Implantation
    - Dans le simulateur *Sinalgo*
    - Source *(plug-in Java)*, Tutorial et supports disponibles sur ma page Web
- **Problèmes considérés :** briques de bases pour les réseaux
  - élection, circulation de jeton, diffusion, *etc.*
- **Intérêt et inconvénient :** Indépendant de l'architecture matérielle

Contenu

- 8 semaines, 26 heures
- Alternance Cours-TD / TP
- Suite du cours « Algorithmique Distribuée » de RICM4
- Aspect « tolérance aux fautes »

Plan du cours

- Rappel « Algorithmique non tolérante aux fautes »
  - 2 heures
- Introduction à la tolérance aux fautes + bit alterné
  - 3 heures
- Consensus, impossibilité (FLP) et algorithmes
  - 10 heures
- Auto-stabilisation
  - 11 heures

Evaluation

- Examen
- 2h
- Fin décembre ?
- Annales disponibles sur ma page Web

Introduction

Algorithmes distribués

Algorithmes distribués

- Algorithmes pour des *systèmes distribués* ☺

- « En informatique, un **système distribué** (aussi appelé système réparti) est un ensemble d'unités de traitement autonomes interconnectées entre-elles. »
  – Gérard Tel, *Introduction to distributed algorithms*

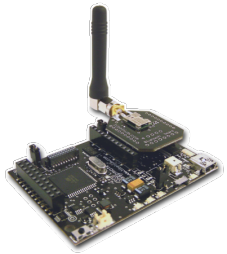
Algorithmes distribués

- **Système distribué ≃** modélisation des réseaux
  - Internet
  - Le réseau de l'Université
  - Le réseau téléphonique (filaire, cellulaire)
  - GPS
  - Réseau de capteurs (surveillance sismique)
  - Threads sur une machine
  - ...

Algorithmes distribués

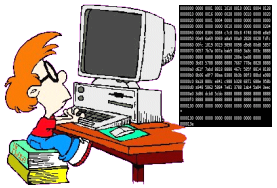
- **Unités de traitement** = processus = processeurs = nœuds

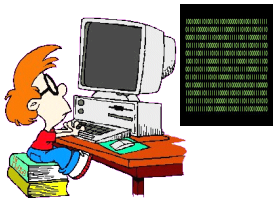
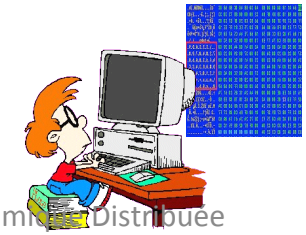
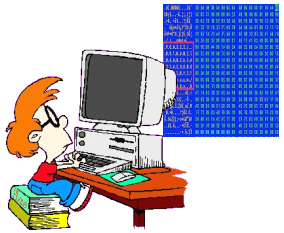




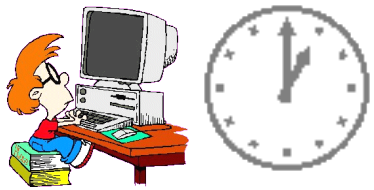

Algorithmes distribués

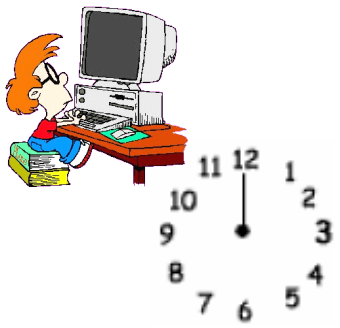
- **Autonome :**
  - Chaque machine est pourvue de son propre contrôle
    - Programmes locaux
    - Mémoires locales

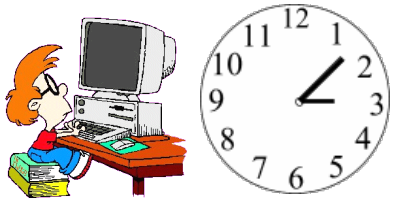


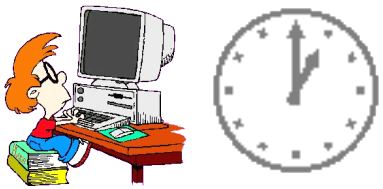




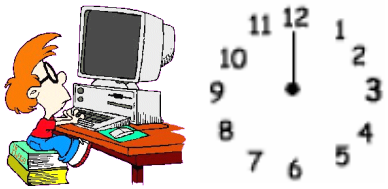
Algorithmes distribués







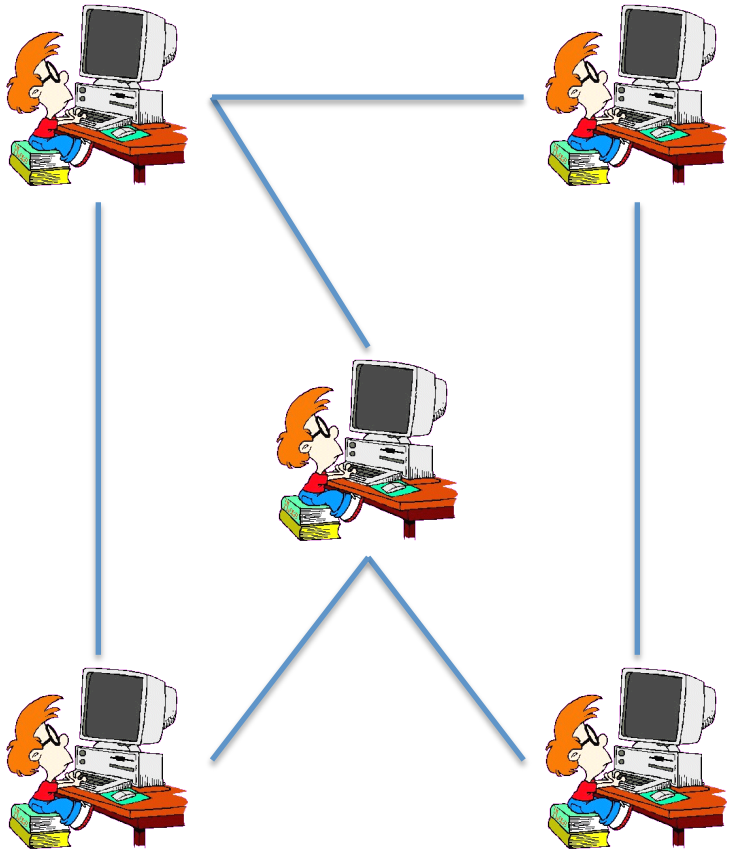




- **Autonome :**
  - ≠ algorithmes parallèles : pas de synchronisation (logicielle ou matérielle)
    - Asynchrone
    - Pas de temps global

Algorithmes distribués

- **Interconnectées :** échanges d'informations

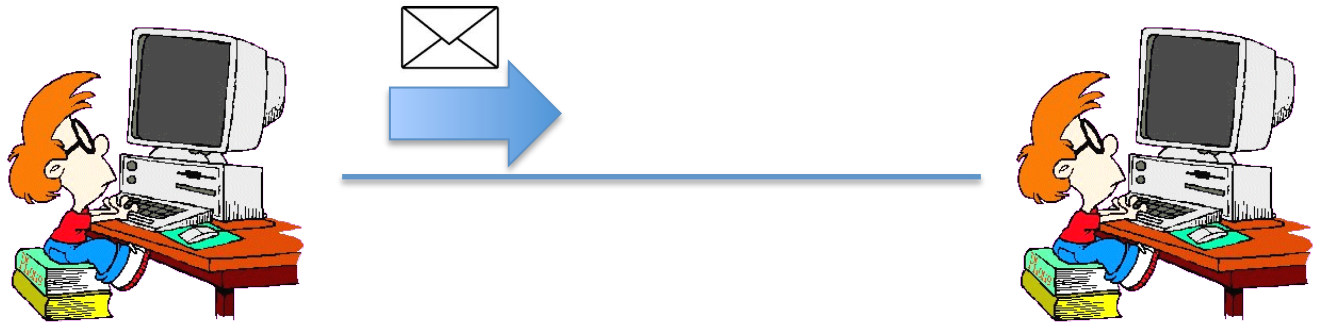


Algorithmes distribués

- **Interconnectées**

Algorithmes distribués

- **Interconnectées :** échanges d'informations
  - *via* messages



Algorithmes distribués

- **Interconnectées :** échanges d'informations
  - Couches de communication (Modèle OSI)

Utilisateur final

Application
Présentation
Session
Transport
Réseau
MAC
Physique

Deux fonctions :
- Envoyer(M,v)
- Recevoir(M,v)

Protocoles réseaux : Algorithmes distribués

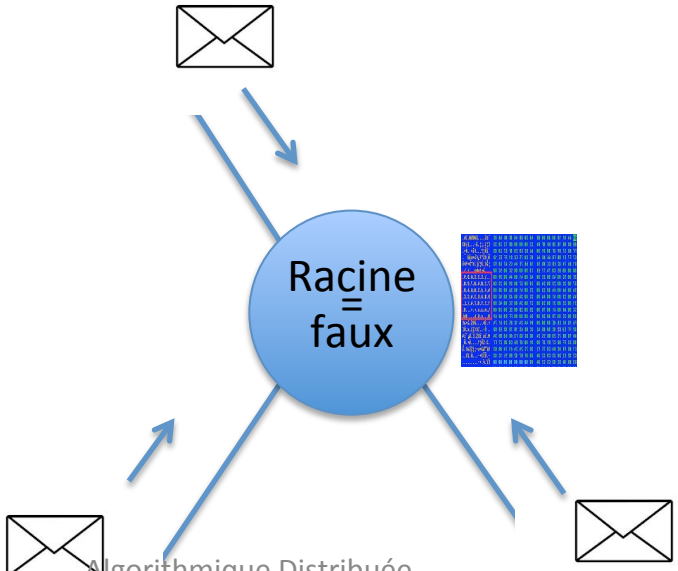
Envoi d'une trame de bits (message) point à point

Envoi d'un seul bit d'information point à point

Centralisé *vs.* Distribué

Centralisé *vs.* Distribué

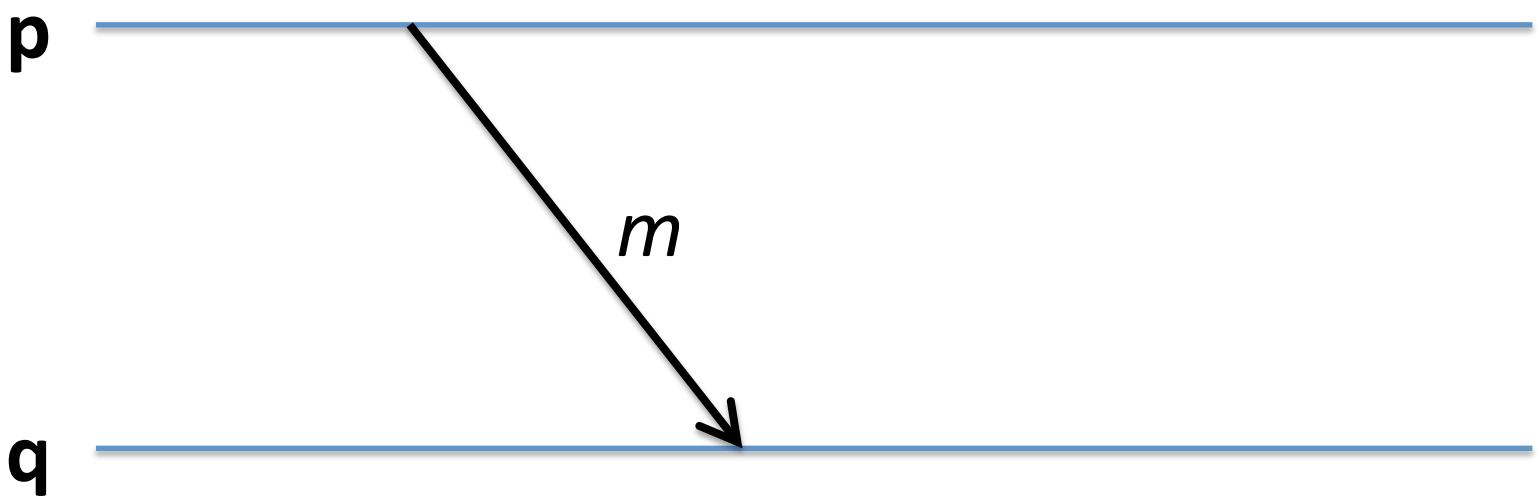
- **Absence de connaissance globale**
  - Pas d'accès à l'état global du système
  - Décisions basées la mémoire locale du nœud
    - Mise à jour en fonction des échanges d'informations
    - Problème de pérennité des informations (système asynchrone)



Centralisé *vs.* Distribué

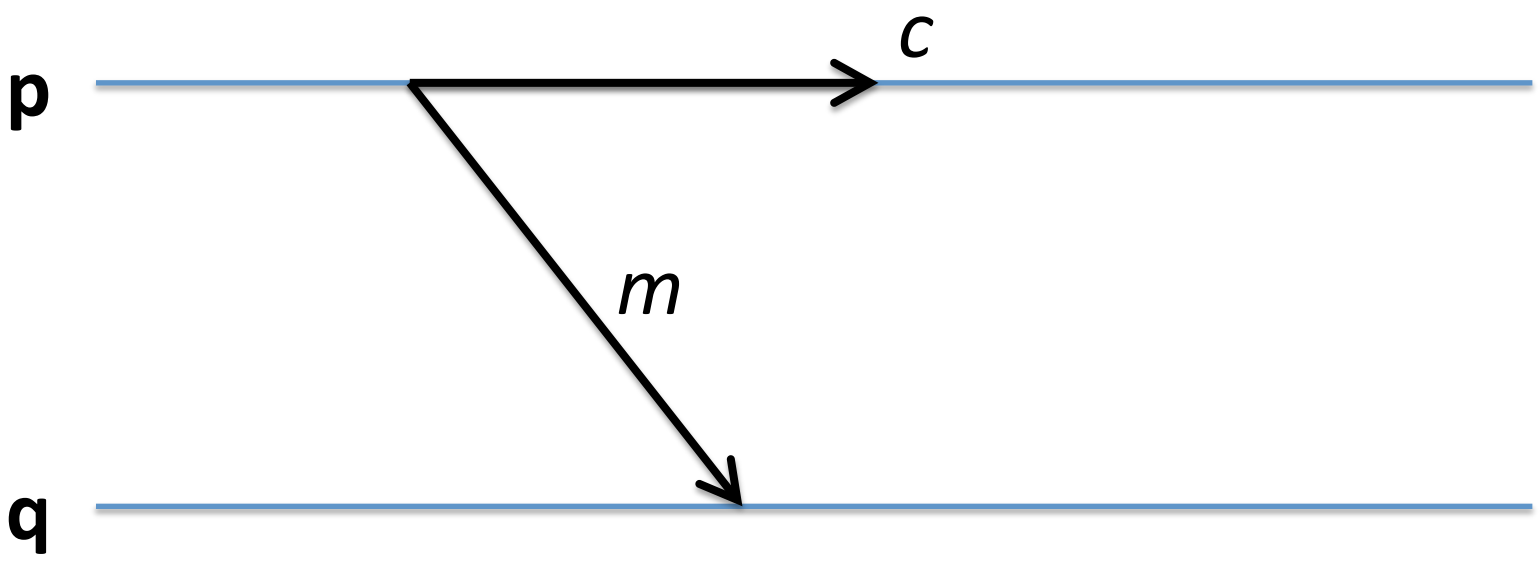
- Absence de temps global:
  - Temps d'exécution des nœuds ≠
  - Communications asynchrones
  - Horloges locales ≠ (valeurs initiales ≠ + dérive)
  - Conséquence : contrairement aux systèmes centralisés, l'ordre (observable par les processus) entre les actions des processus n'est que partiel : ***l'ordre de causalité***
    - (Lamport 1978)

Causalité : exemple



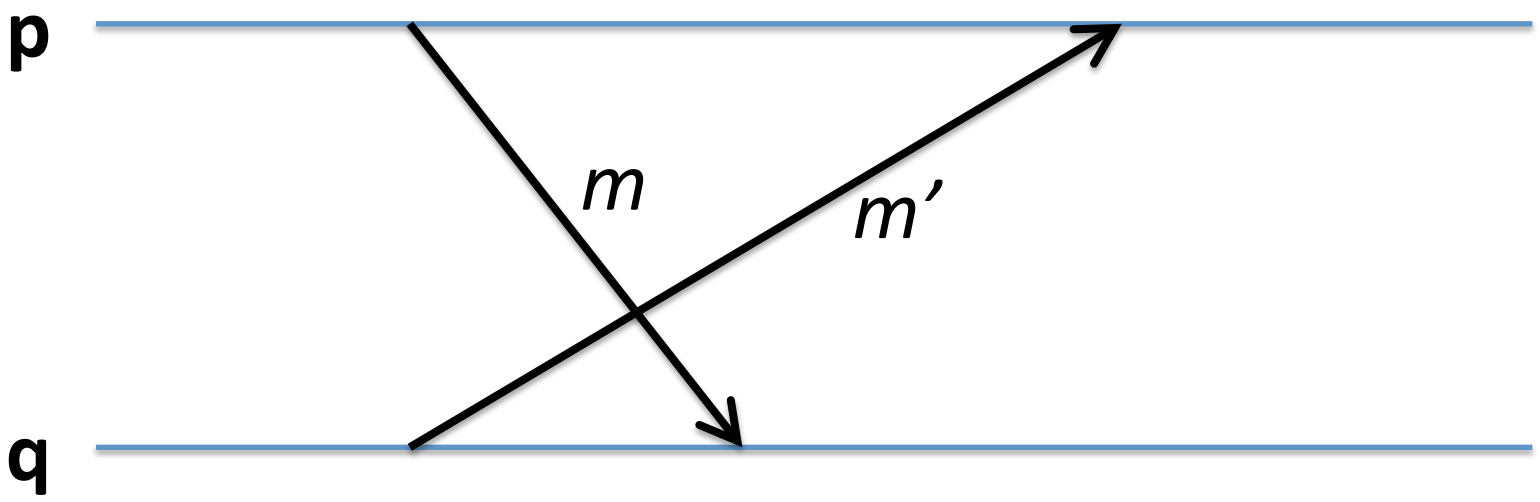
- L'événement « **p** envoie *m* » arrive *avant* l'événement « **q** reçoie *m* »

Causalité : exemple



- L'événement « **p** envoie *m* » arrive *avant* l'événement « **p** effectue le calcul interne *c* »

Causalité : exemple

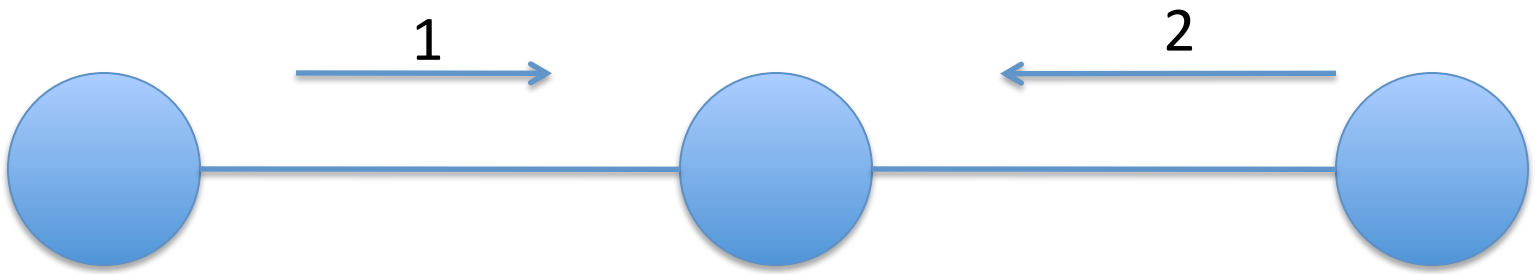


- Les événements « **p** envoie *m* » et « **q** envoie *m’* » sont *indépendants*

Centralisé *vs.* Distribué

- Non-déterminisme
  - Algorithme déterministe séquentiel : sorties fonction des entrées
  - Algorithme déterministe distribué : pour les mêmes entrées, plusieurs résultats peuvent être possible

Exemple



Sortie = premier message reçu – deuxième message reçu
Résultat ?

Caractéristiques d'un système distribué

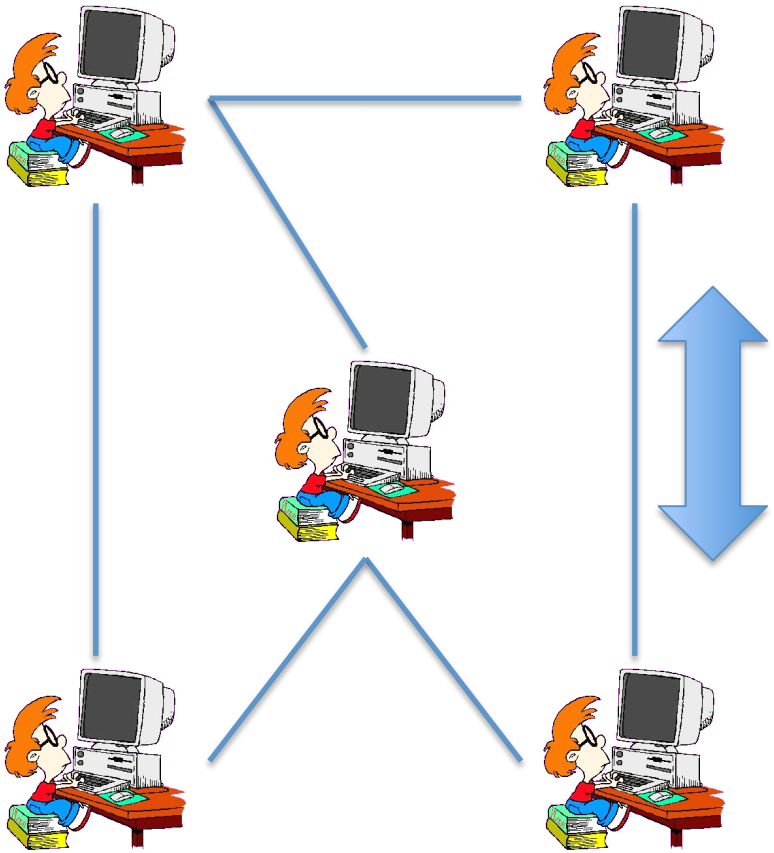
- Sa topologie
- Ses liens de communications
- Ses processus
- Le temps
- Les connaissances initiales des processus sur le système

Caractéristiques d'un système distribué : sa topologie

Caractéristiques

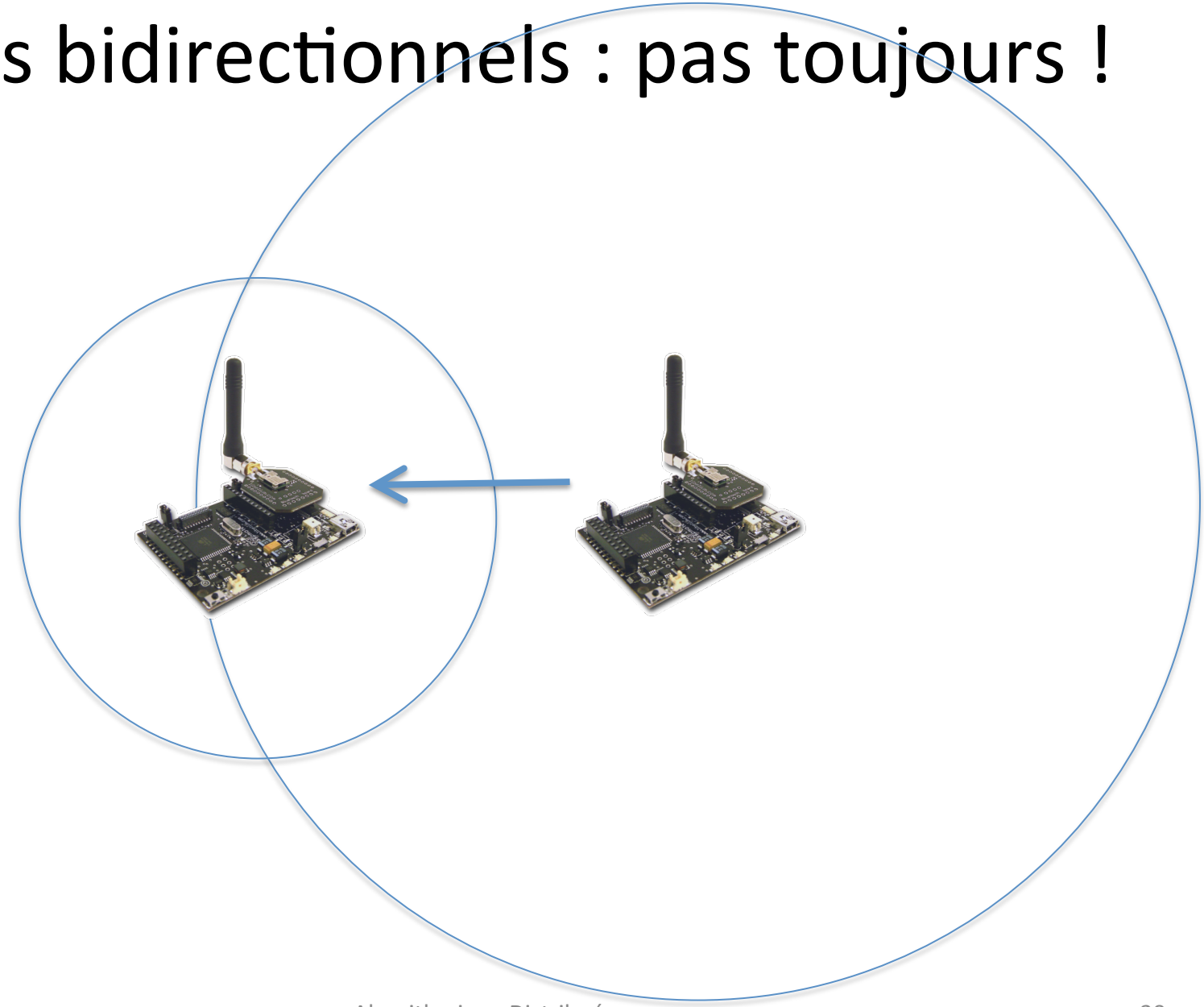
- **Topologie = réseaux d'interconnexion**
- **Topologies** ≈ Graphe (simple) G=(V,E)
  - 2 processus qui peuvent communiquer directement = *voisin*
  - Communications :
    - bidirectionnelles (*ex.*, réseaux filaires) ou
    - unidirectionnelles (*ex.*, réseaux à fibres optiques)
  - Généralement on supposera des communications bidirectionnelles et une topologie connexe

Les systèmes distribués

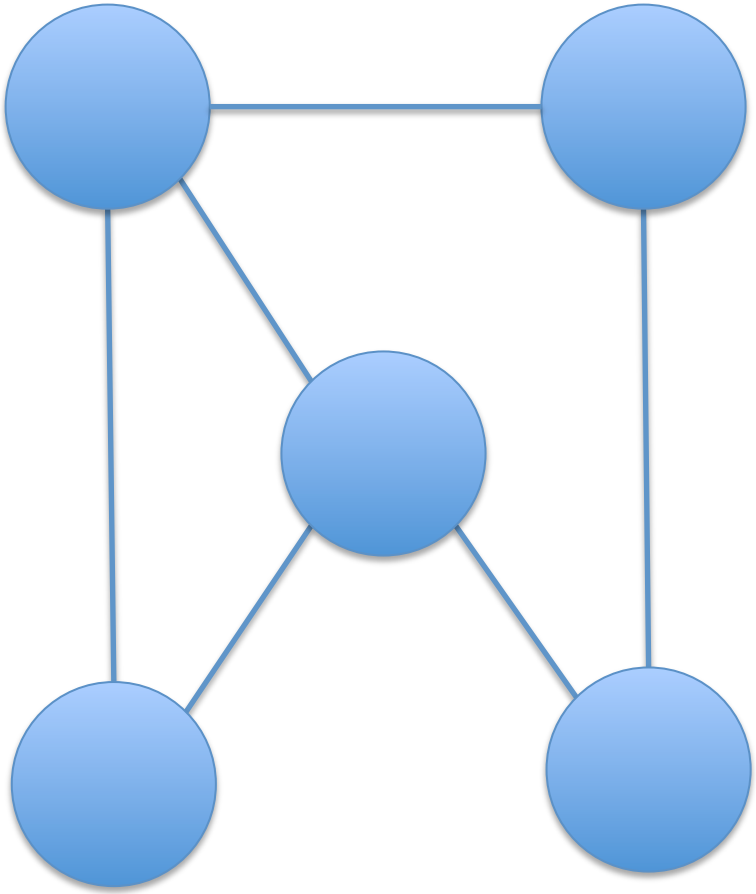


- Hypothèses
  - Liens bidirectionnels

Liens bidirectionnels : pas toujours !

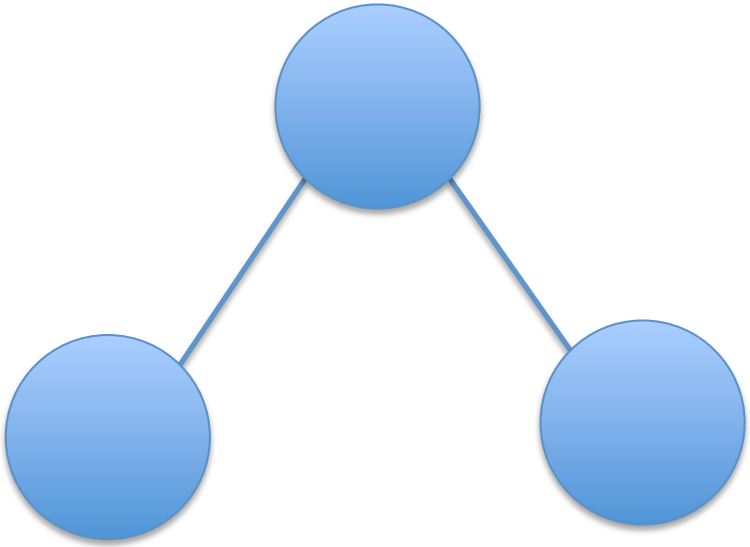


Rappel : Connexité



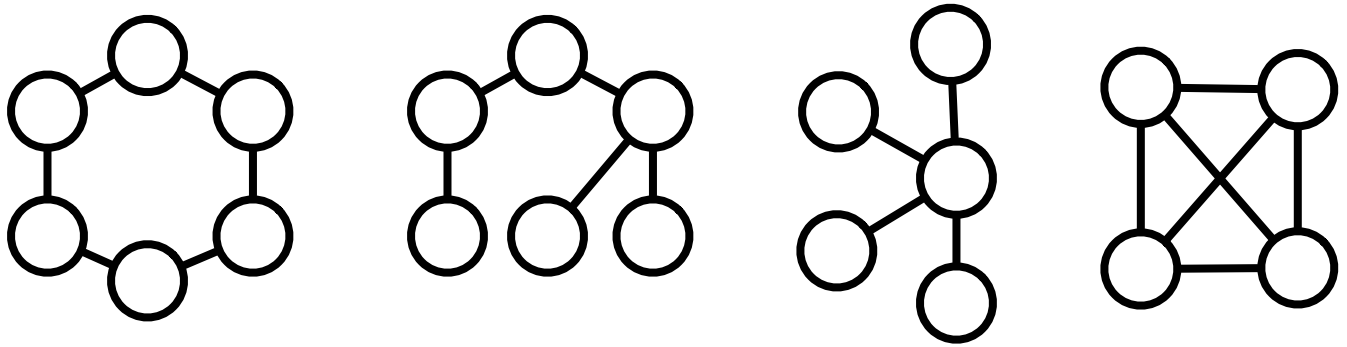
Connexe !

Rappel : Connexité



Pas connexe !

Exemples de topologies



Caractéristiques d'un système distribué : liens de communications

Communications

- Par message (modèle à passage de messages)
  - Fiabilité du canal
    - (fiable ou avec perte équitable ou taux de livraison ou dynamique)
    - Fiable = pas de perte, duplication, création
  - Ordre d'arrivée des messages
    - (FIFO ou non)
  - Capacité des canaux
    - (borné ou pas, borne connue ou pas)
  - Temps d'acheminement (toujours fini)
    - (synchrone, asynchrone, finalement synchrone)
    - (pour les deux derniers, borne connue ou pas)
  - Temps de traitement (supposé) négligeable

FIFO : Rappel

FIFO : Rappel

A

FIFO : Rappel

B A

FIFO : Rappel

C B A

FIFO : Rappel

C B

A

FIFO : Rappel

C

B

FIFO : Rappel

C

Temps d'acheminement

1. Un canal est *synchrone* s'il existe une constante *c* telle que pour tout temps *t*, si un message est émis au temps *t* alors il est reçu avant le temps *t* + *c* ou perdu.
2. Un canal est *finalement synchrone* s'il existe un temps *t* et une constante *c* telle que pour tout temps $t' \geq t$, si un message est émis au temps *t'* alors il est reçu avant le temps *t'* + *c* ou perdu.
3. Un canal est *asynchrone* s'il n'existe pas de temps *t* et de constante *c* telle que pour tout temps $t' \geq t$, si un message est émis au temps *t'* alors il est reçu avant le temps *t'* + *c* ou perdu.

Caractéristiques d'un système distribué : les processus

Caractéristiques

- **Processus**
  - *Initiateur* (démarrage « spontané ») ou *non-initiateur* (aussi appelé suiveur, démarrage suite à une communication avec un voisin)
  - Sujet ou pas aux pannes ? (et quel type ?)
  - Asynchrone (équitable), synchrone ou finalement synchrone
  - Mémoire locale de taille bornée ou non ?

Caractéristiques d'un système distribué : le temps

Caractéristiques

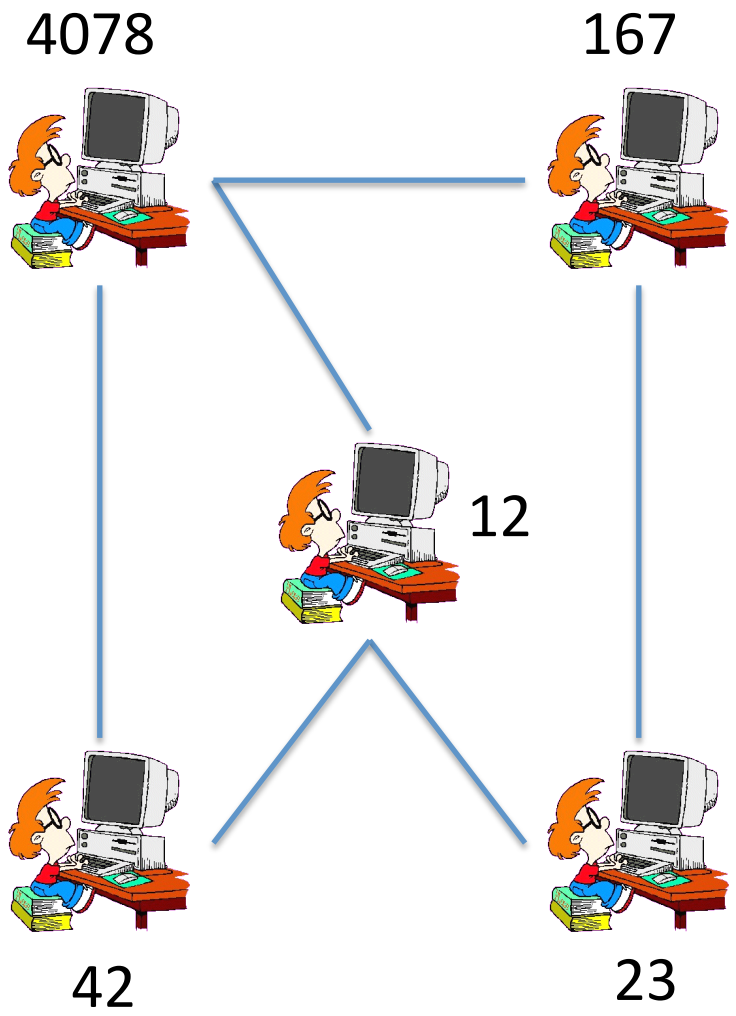
- **Temps** (global)
  - Discret : temps 0, 1, 2 …
  - Non accessible aux processus mais utilisé dans les preuves
  - Cependant, on utilisera parfois du temps local (*minuteurs*)

Caractéristiques d'un système distribué : connaissances initiales des processus

Caractéristiques

- **Connaissances initiales (entrées) des processus**
  - *Propriétés globales :*
    - Topologique :
      - Topologie, *e.g.*, je sais que je suis dans un arbre
      - Borne supérieure ou exacte sur
        - » Le nombre de processus (n)
        - » Le degré maximum (Δ)
        - » Le diamètre du réseau (D)
    - Distinction entre les processus
      - Anonyme
      - Identifié
      - Semi-anonyme
        - » Enraciné

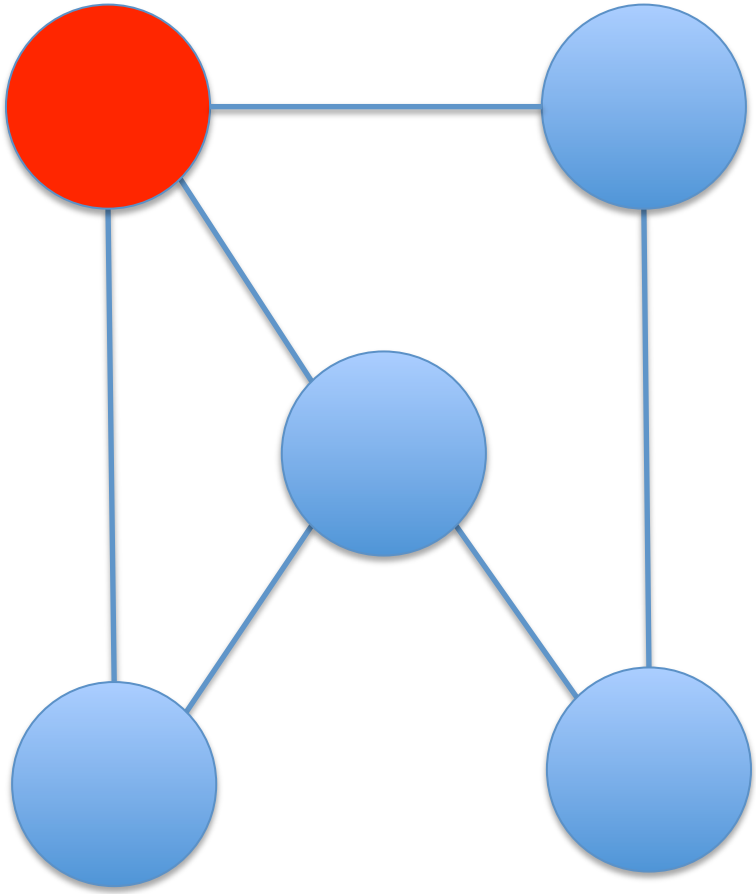
Réseaux identifiés : Rappel



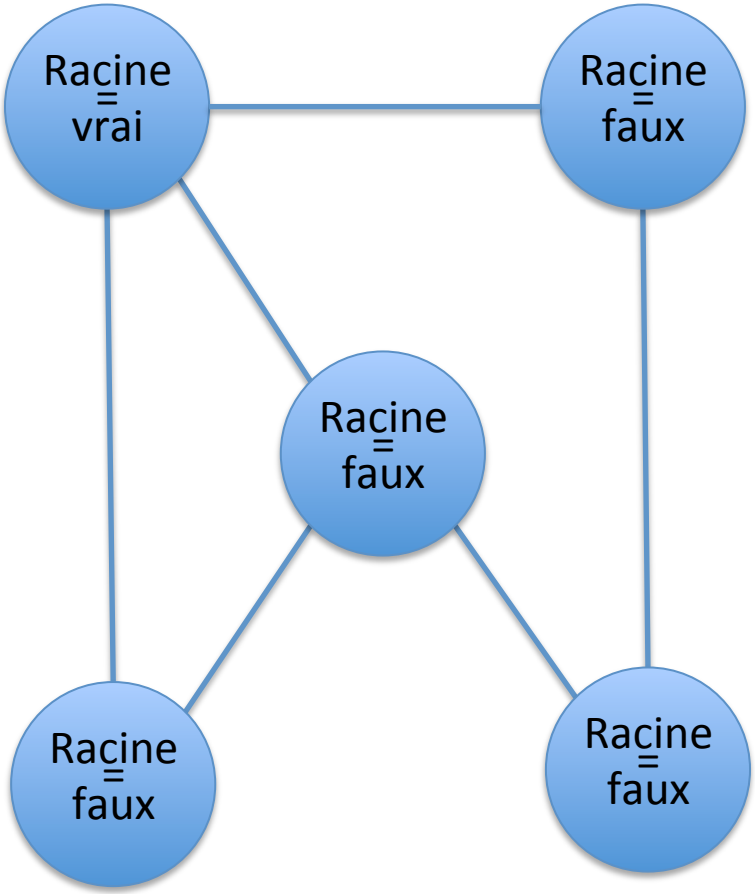
- Identité unique

  (*e.g.*, adresse IP)

Réseaux enracinés : Rappel



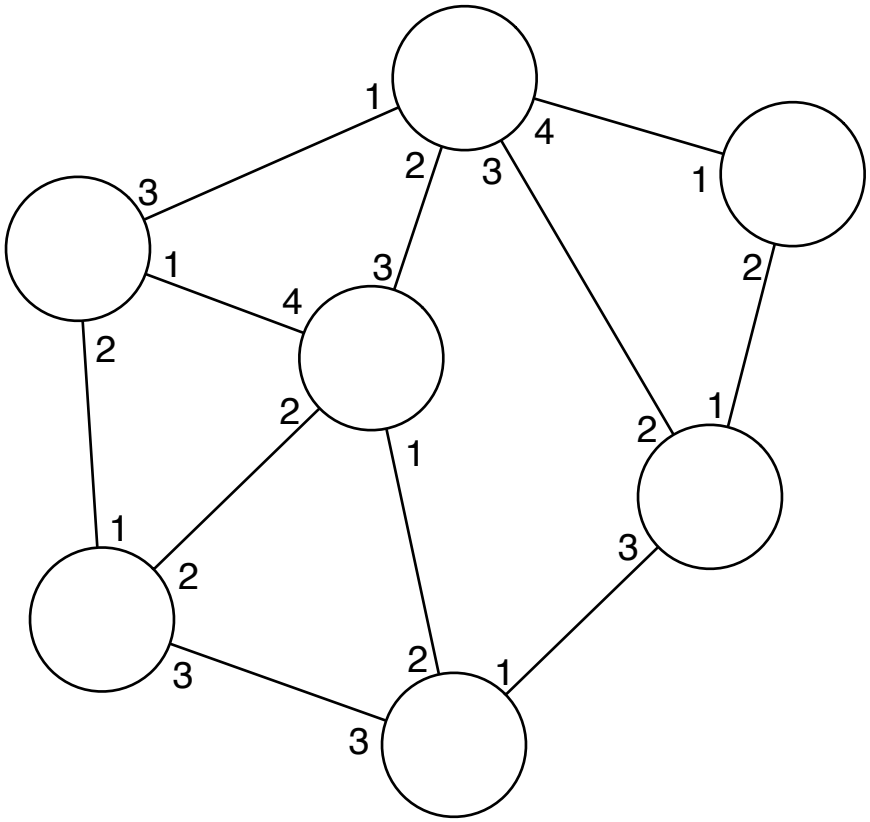
Réseaux enracinés : Rappel



Caractéristiques

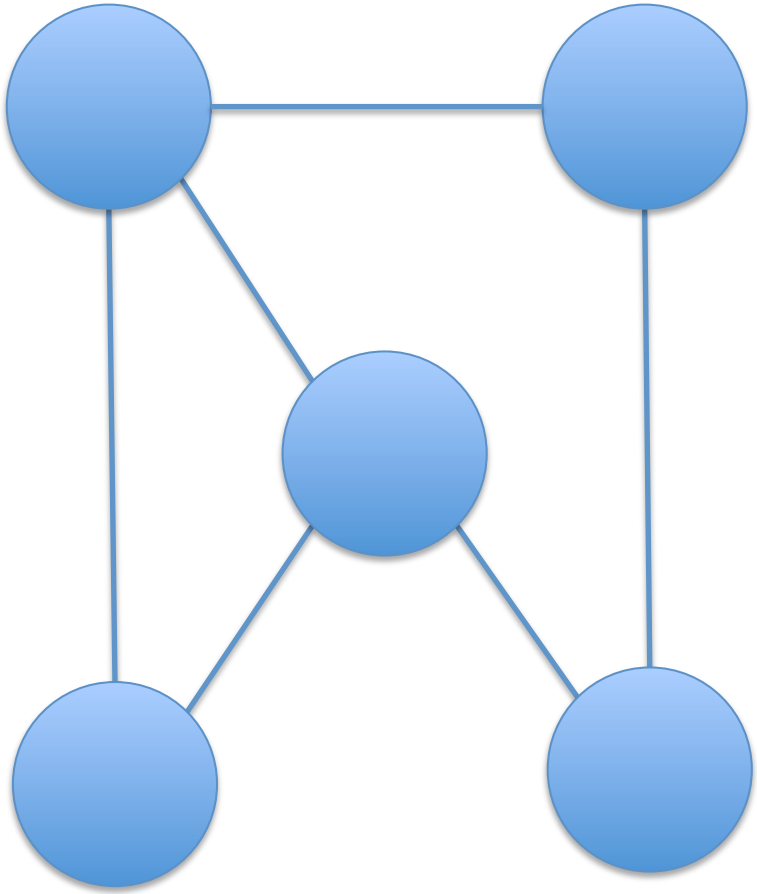
- **Connaissances initiales des processus**
  - *Propriétés locales :* connaissance sur le voisinage
    - Degré local ?
    - Identité des voisins ?
    - Numéro de canaux (étiquetage local) ?
    - Rien ? (*ex.* réseaux sans fils)

Numérotation des canaux

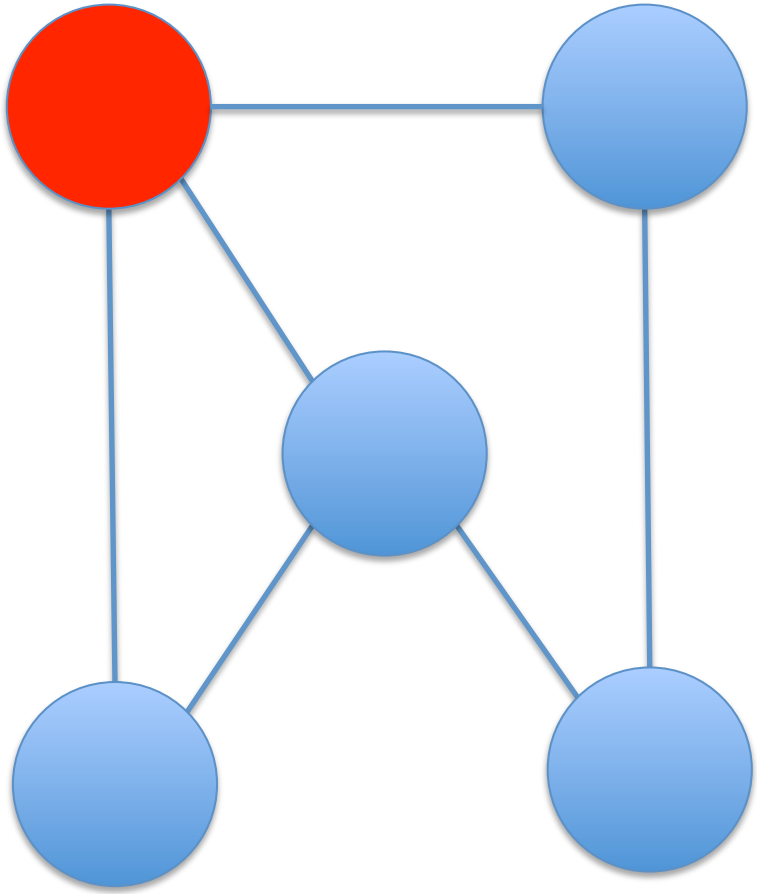


Exemple récapitulatif

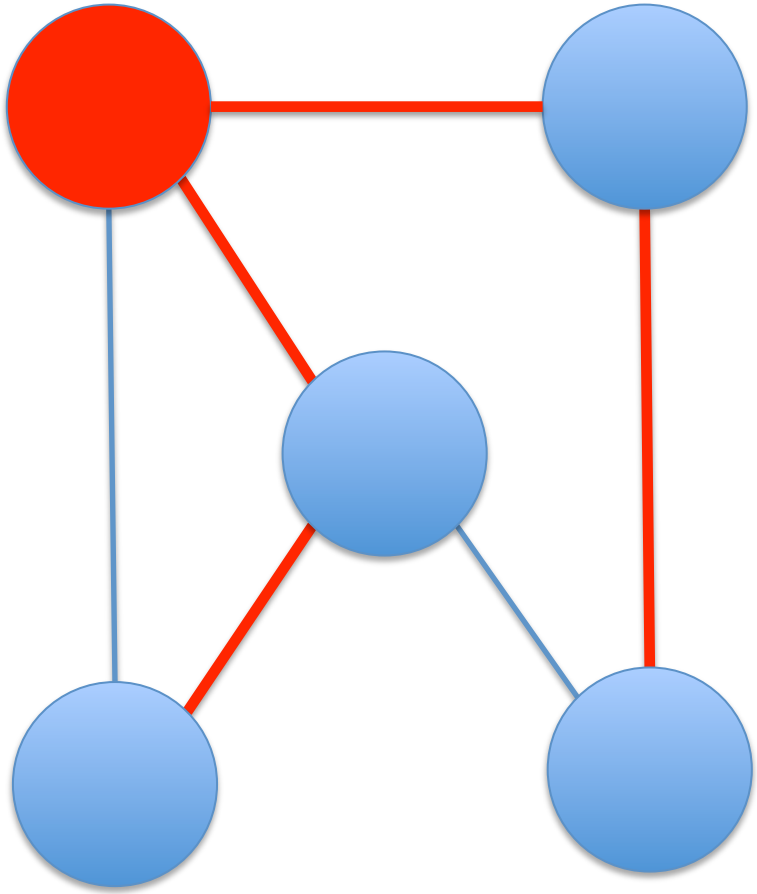
Algorithme Distribué
Exemple : Calcul d'un arbre couvrant



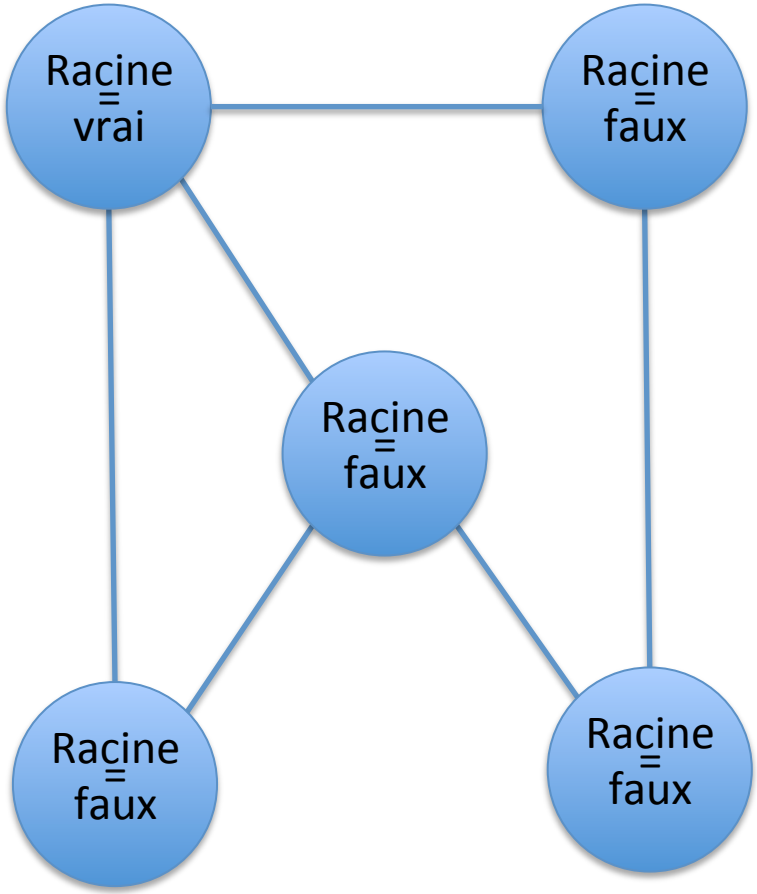
Algorithme Distribué
Exemple : Calcul d'un arbre couvrant



Algorithme Distribué
Exemple : Calcul d'un arbre couvrant

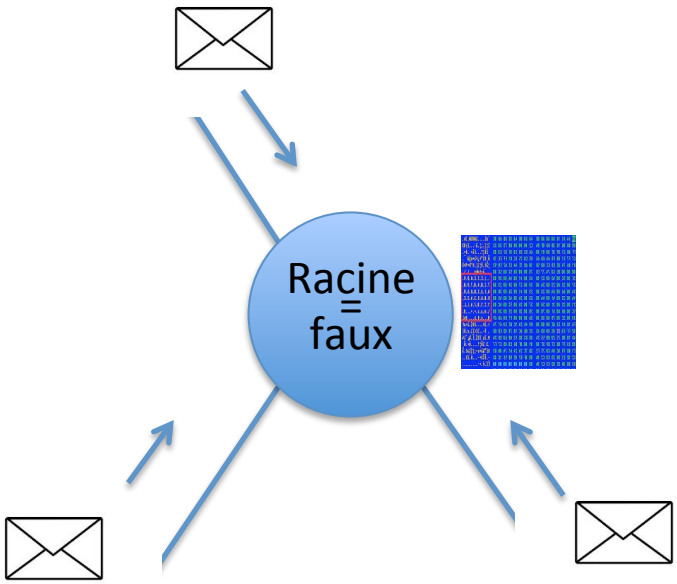


Algorithme Distribué
Exemple : Calcul d'un arbre couvrant



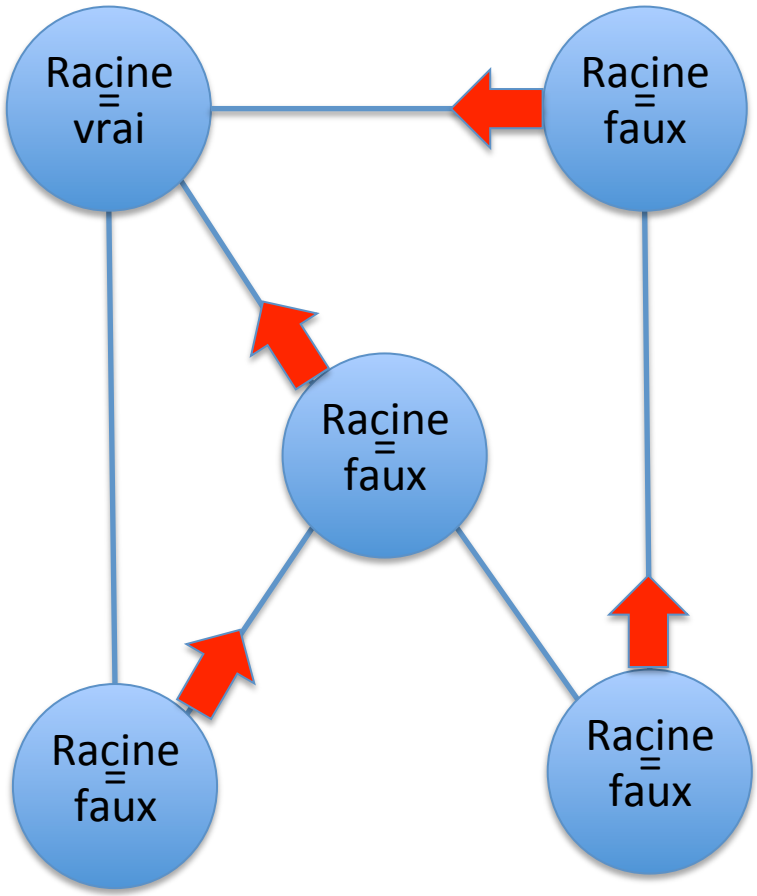
- Entrées réparties

Algorithme Distribué
Exemple : Calcul d'un arbre couvrant



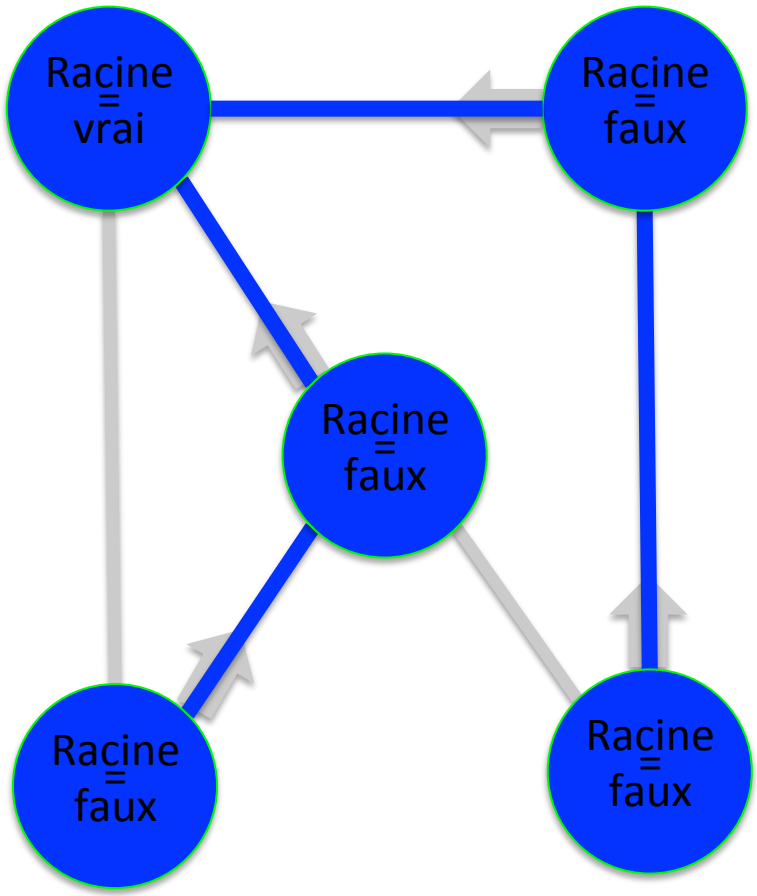
- Entrées réparties
- Calculs locaux
  - Mémoires locales
  - Programmes locals
  - Envoi de messages
  - **Décision locale**

Algorithme Distribué
Exemple : Calcul d'un arbre couvrant



- Entrées réparties
- Calculs locaux
  - Mémoires locales
  - Programmes locals
  - Envoi de messages
  - **Decision locale**
- Sorties réparties

Algorithme Distribué
Exemple : Calcul d'un arbre couvrant



- Entrées réparties
- Calculs locaux
  - Mémoires locales
  - Programmes locals
  - Envoi de messages
  - **Decision locale**
- Sorties réparties
- Tâche globale

Analyse de complexité

Complexité

- Pire des cas, cas moyen, meilleur des cas
- Exact ou asymptotique (notation de Landau)
- Calcul interne négligeable

Mesures de complexité

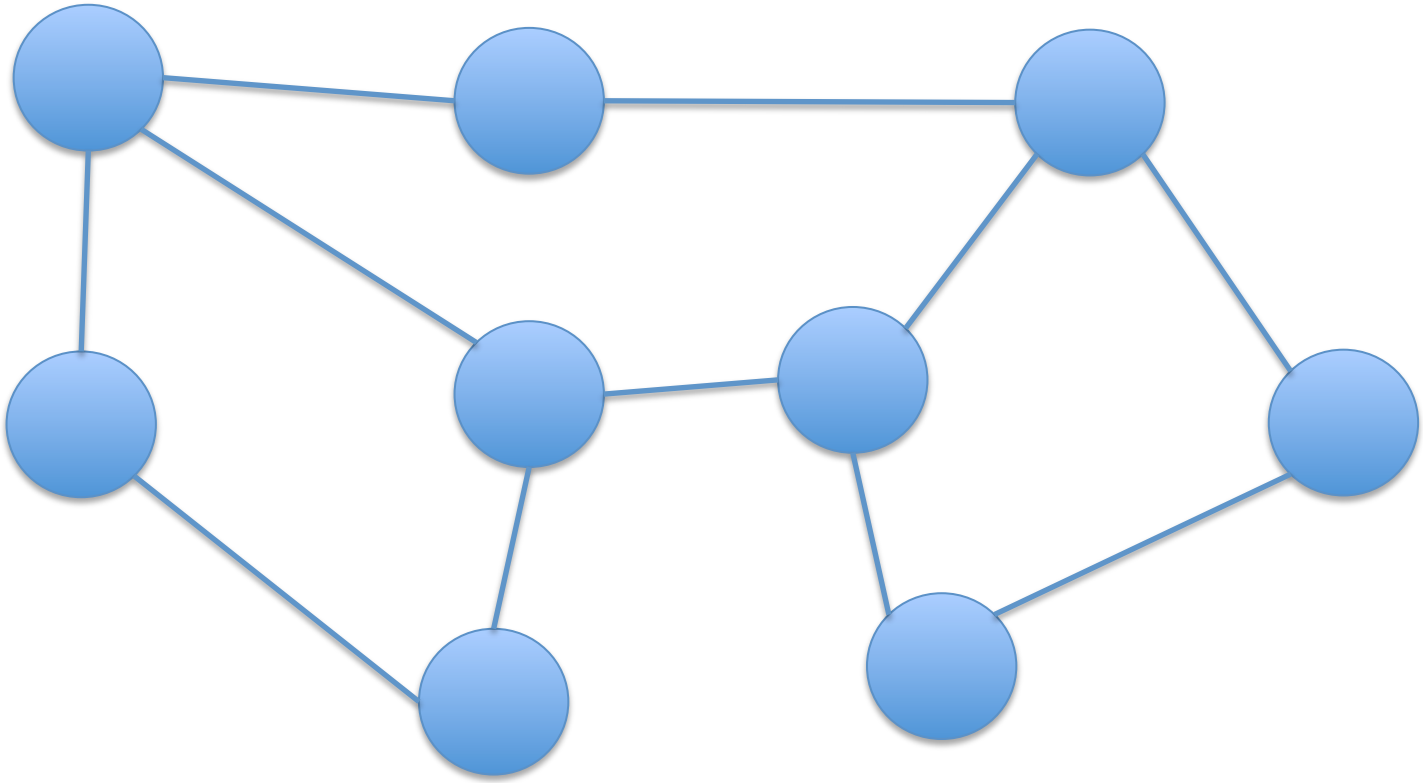
- Complexité en messages
- Complexité en volume
- Complexité en temps
  - Temps de traitement = 0, temps d'acheminement = 1
- Complexité en espace (bits ou états)

Problèmes classiques

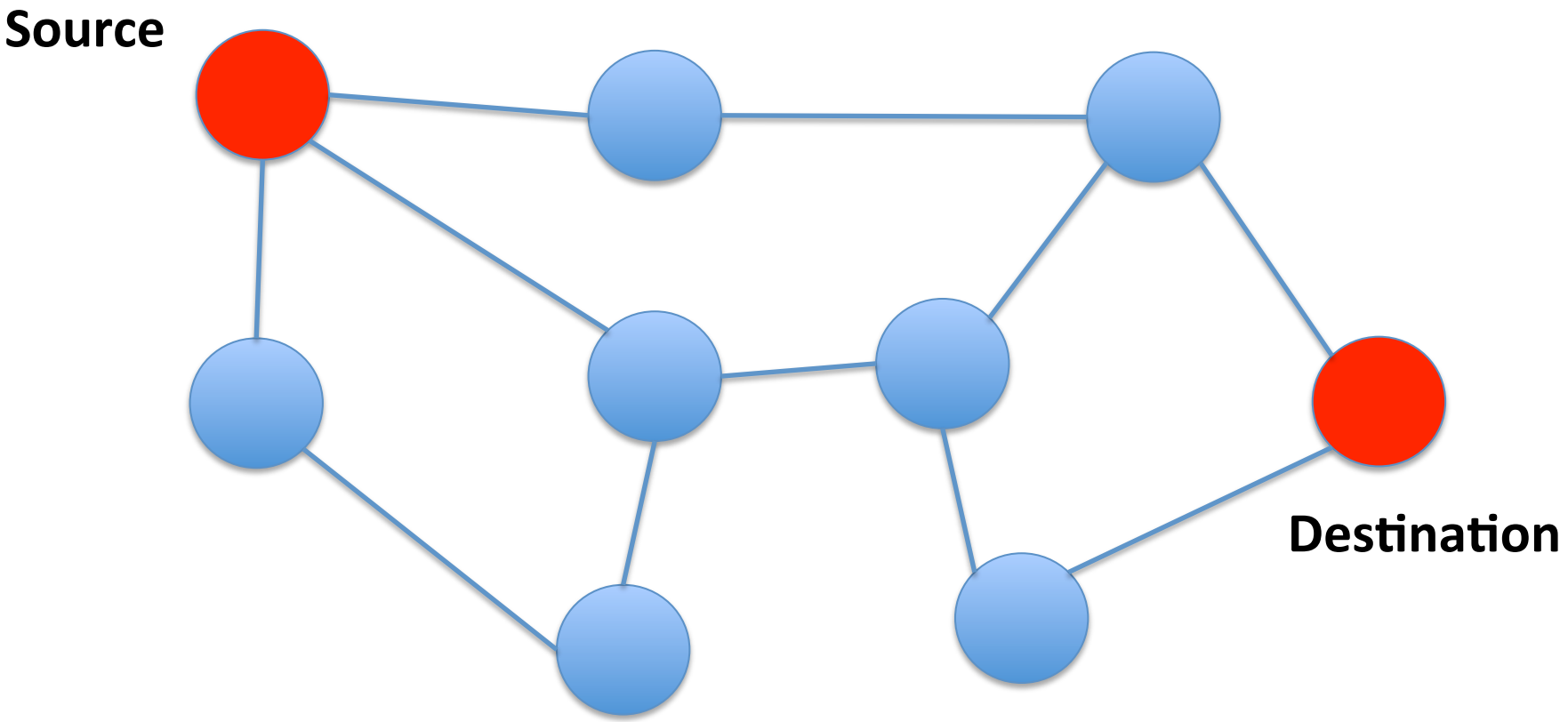
Problèmes classiques

- **Echange de donnée :** routage, diffusion, …
- **Accords :** consensus, élection, …
- **Auto-organisation :** arbre couvrant, clustering
- **Allocation de ressources :** exclusion mutuelle, diner des philosophes…

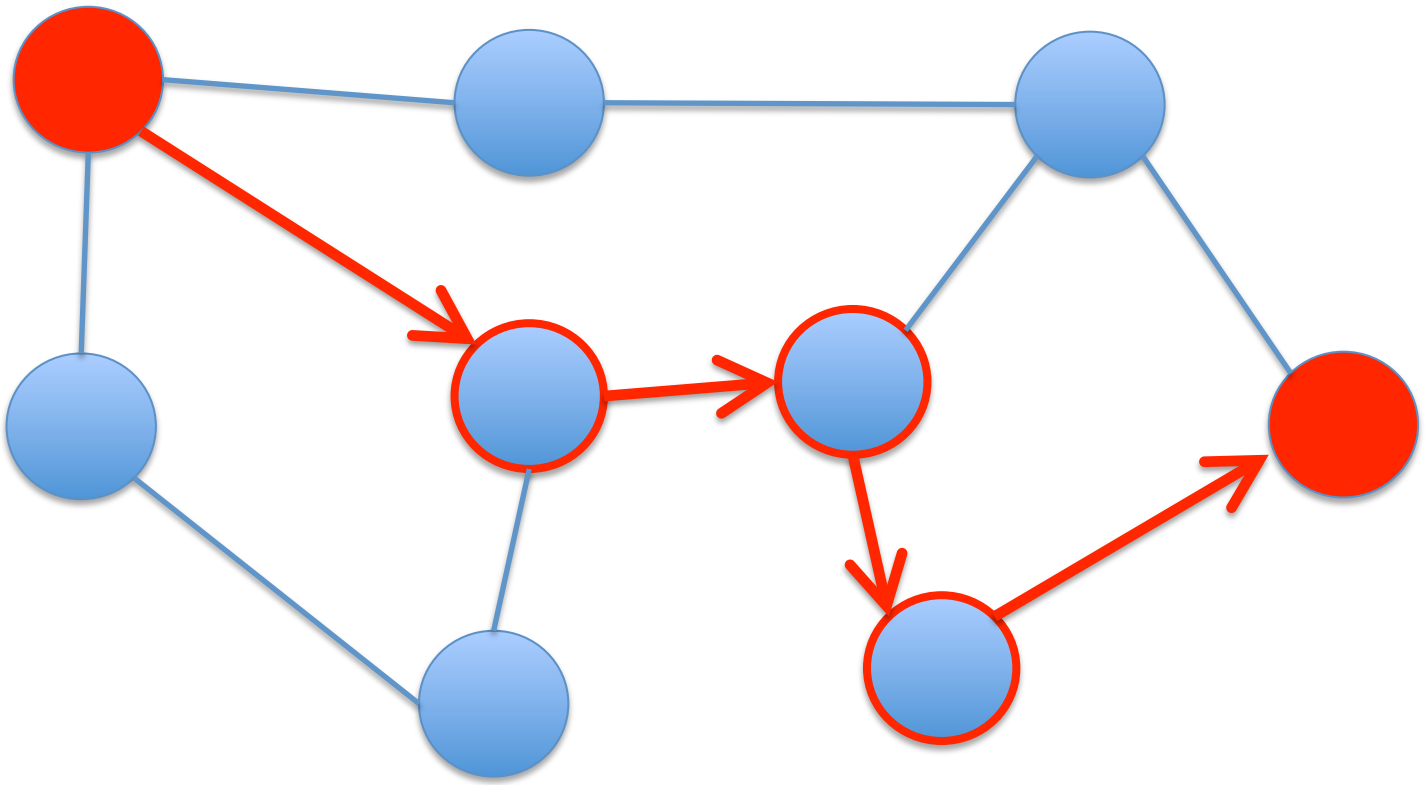
Echange de donnée : routage



Echange de donnée : routage



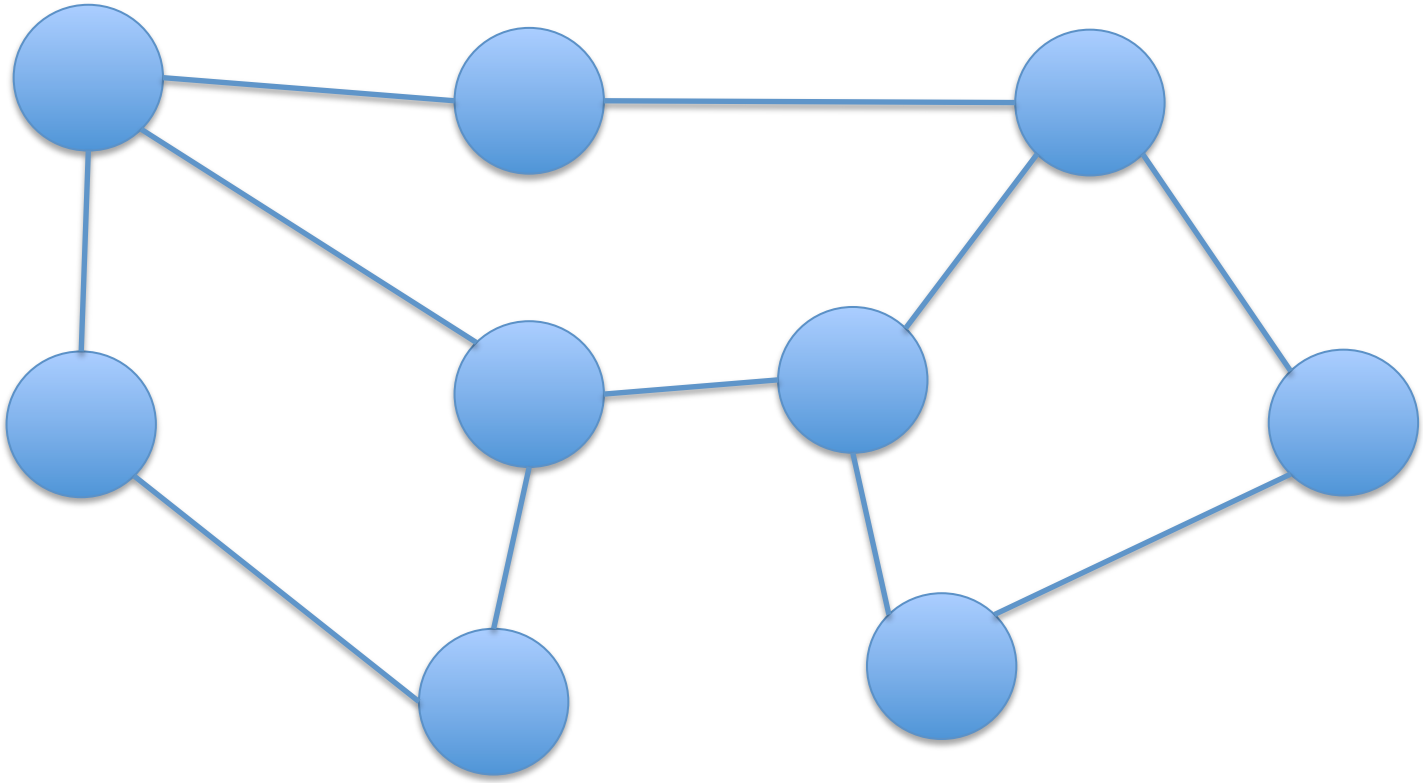
Echange de donnée : routage



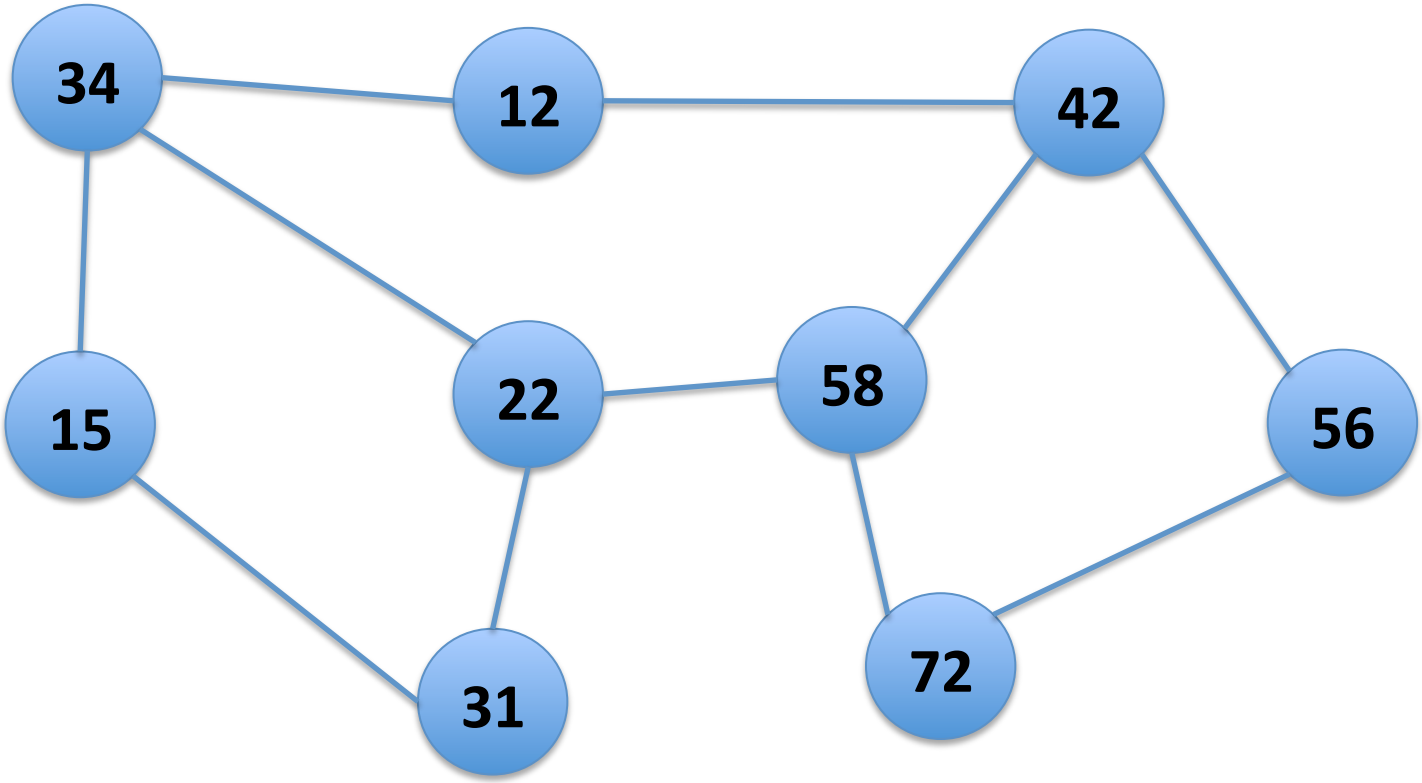
Accord : élection

Calculer un chef !

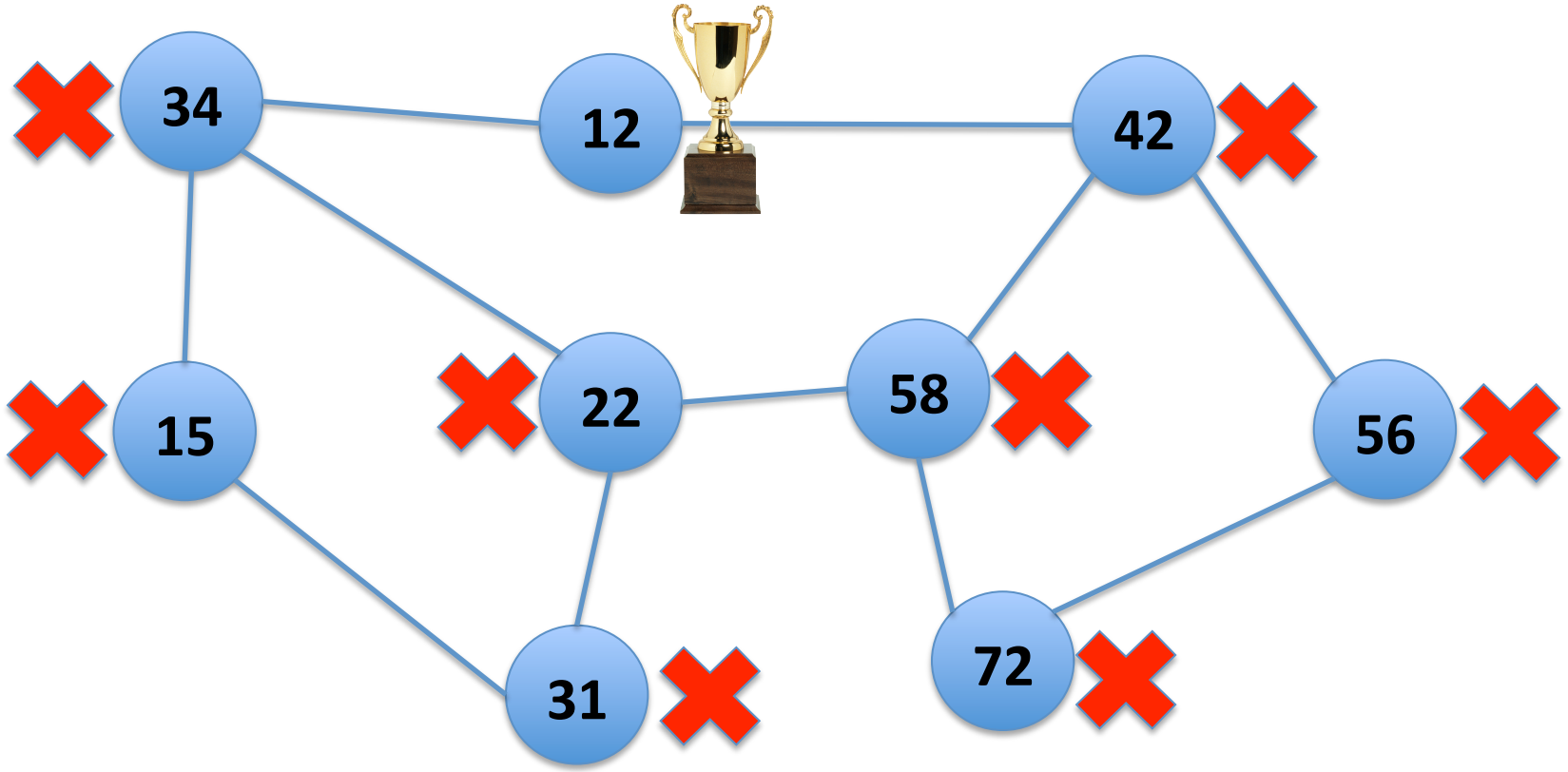
(avec publication ou non)



Accord : élection
Calculer un chef !



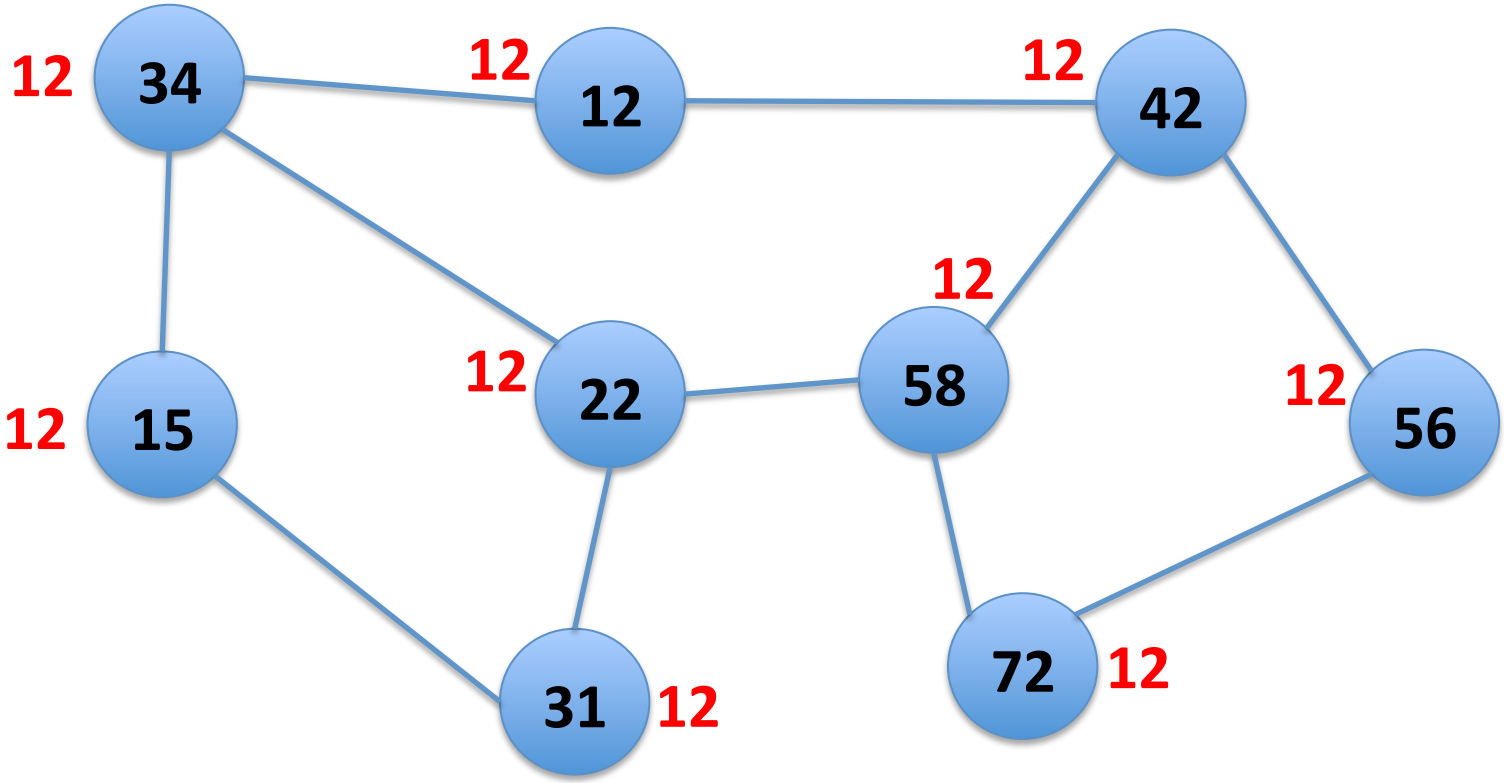
Accord : élection
Calculer un chef !



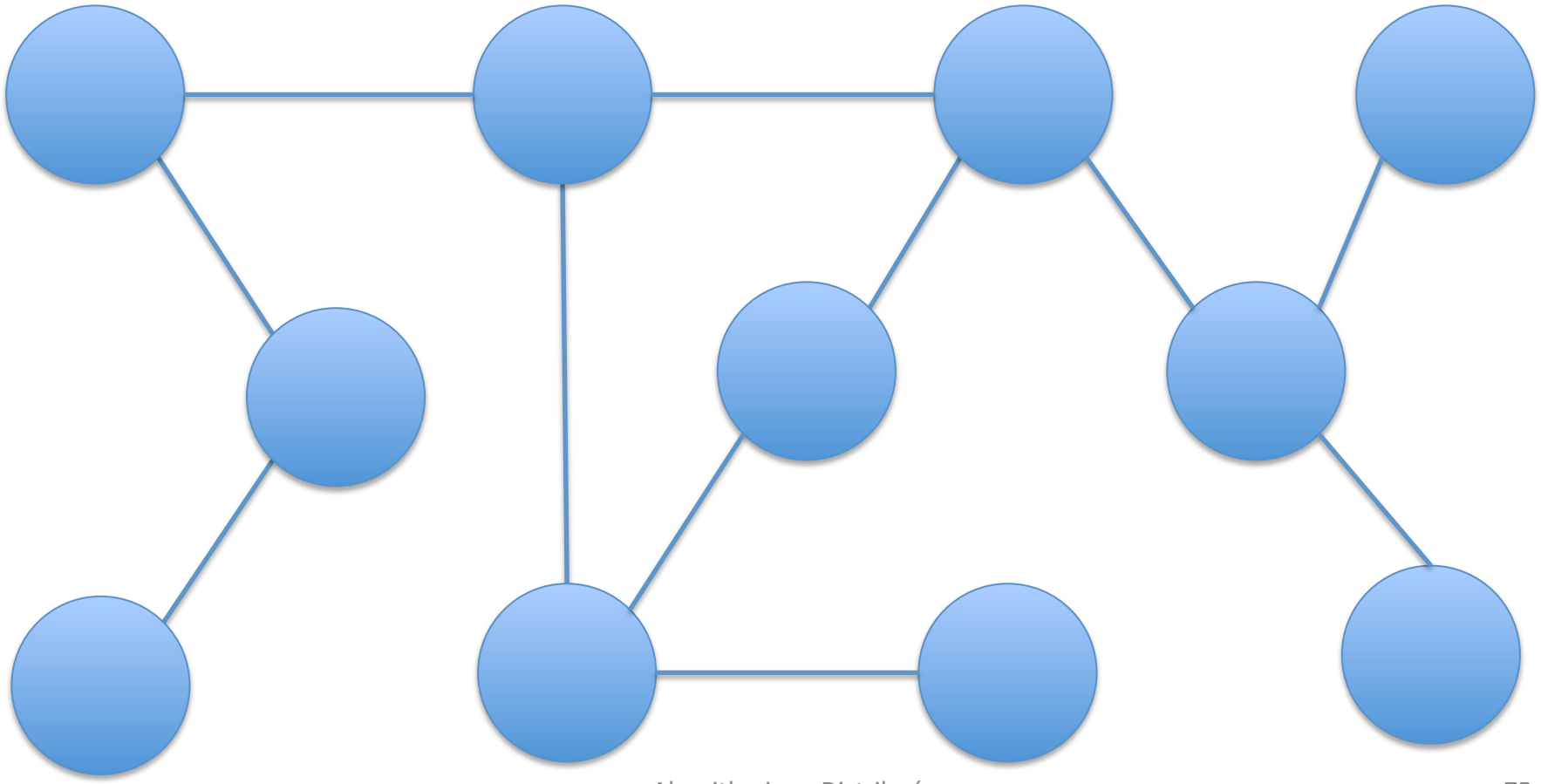
Accord : élection

Calculer un chef !

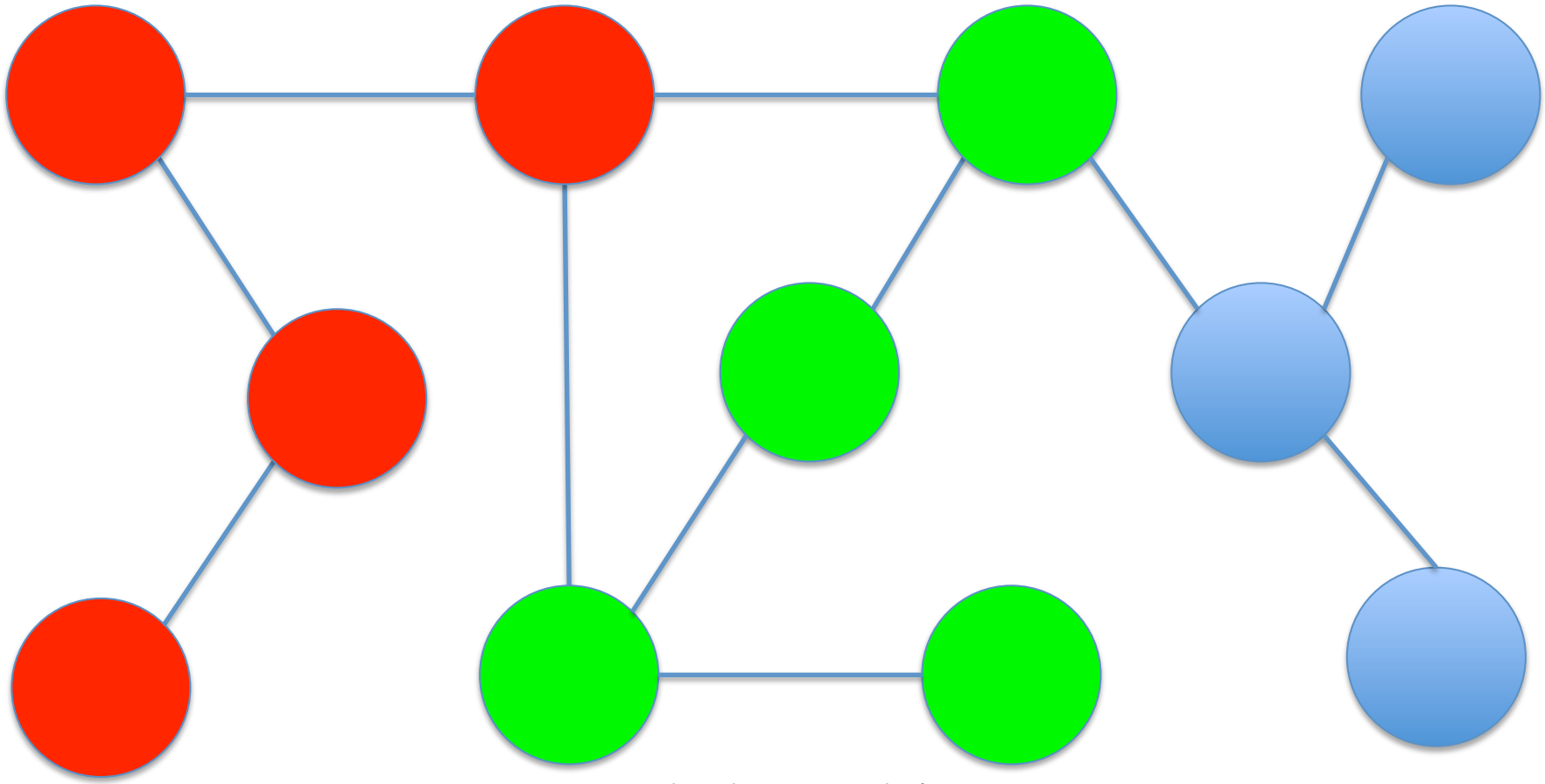
(avec publication)



Auto-organisation : *k*-Clustering

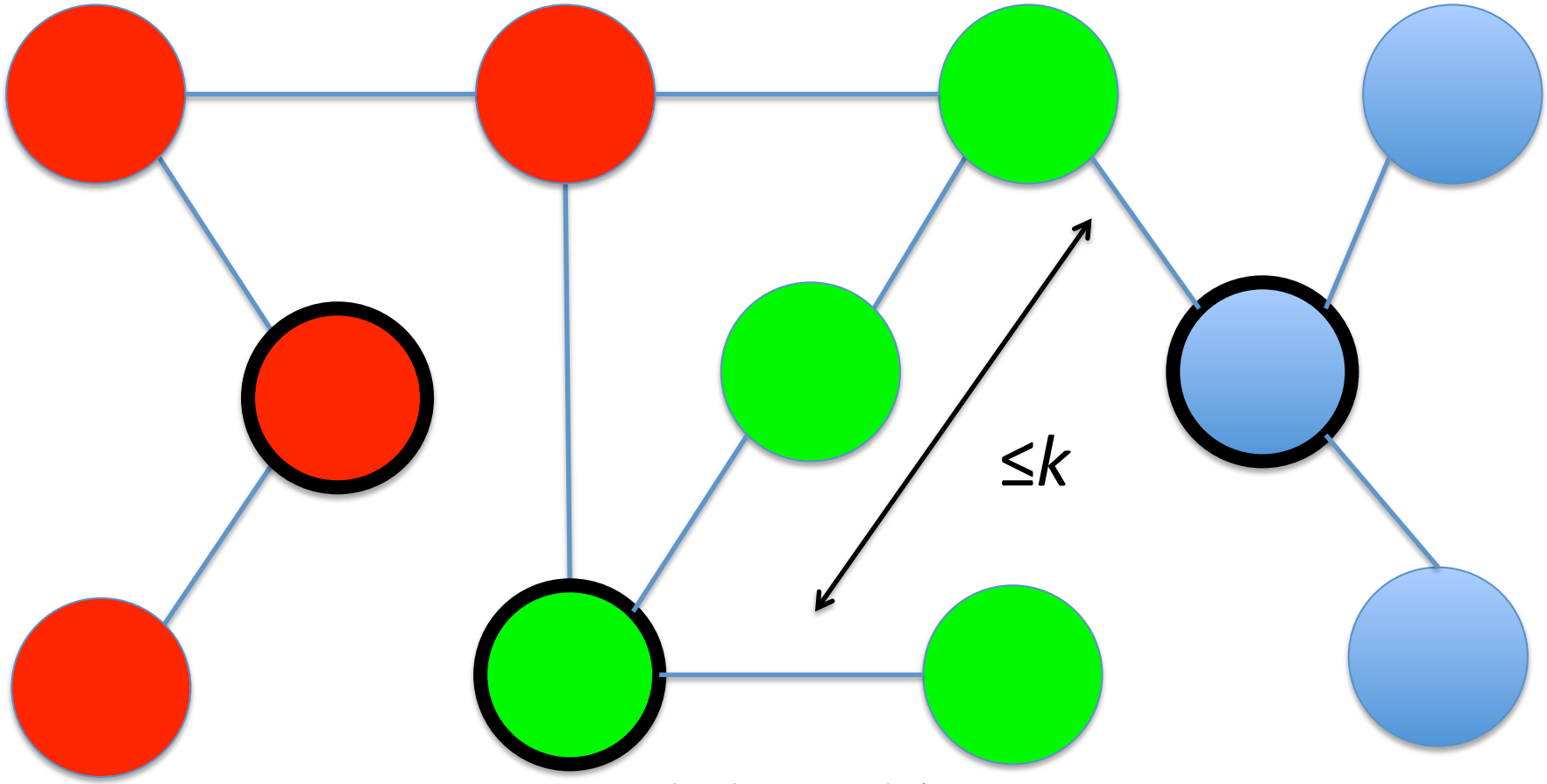


Auto-organisation : *k*-Clustering



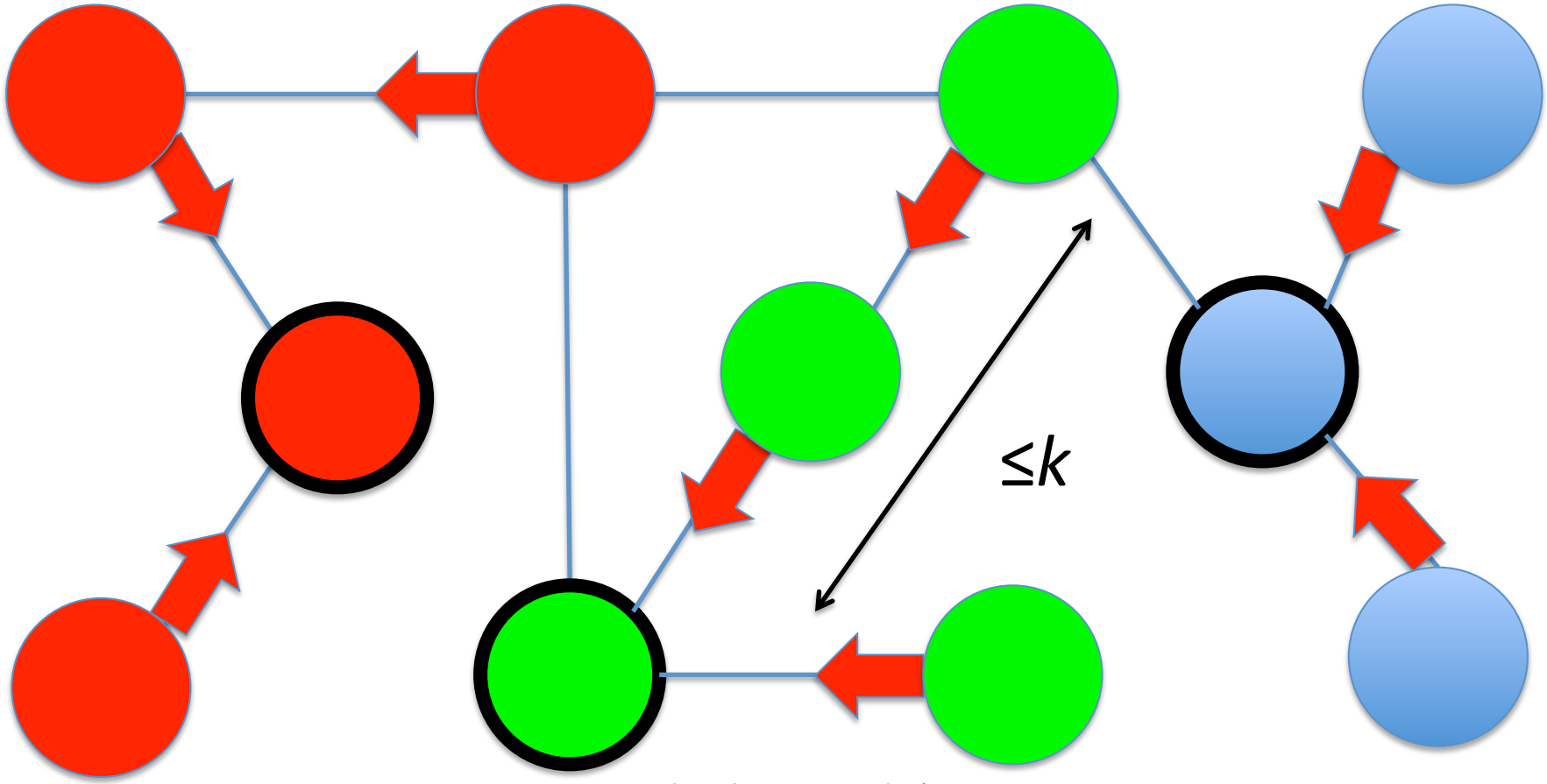
Auto-organisation : *k*-Clustering

- Ex. *k*=2

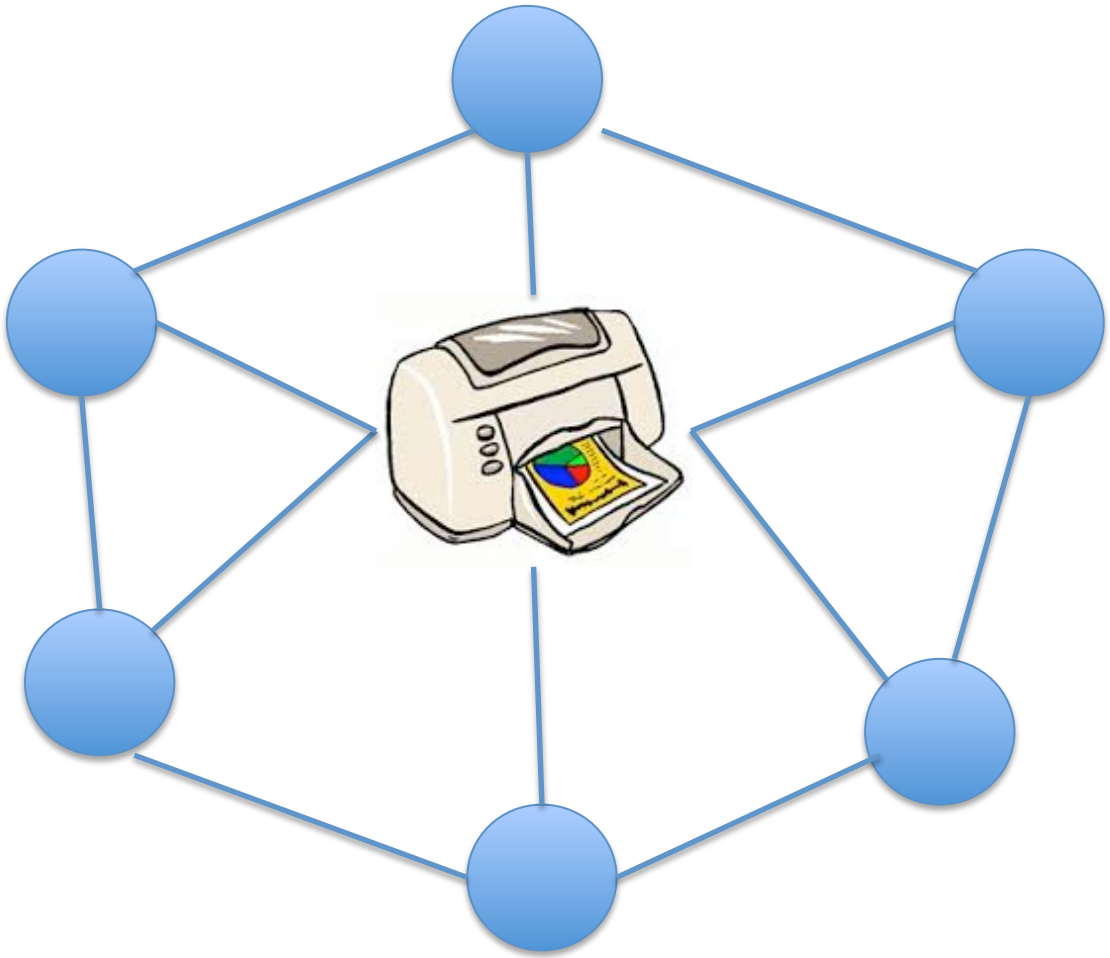


Auto-organisation : *k*-Clustering

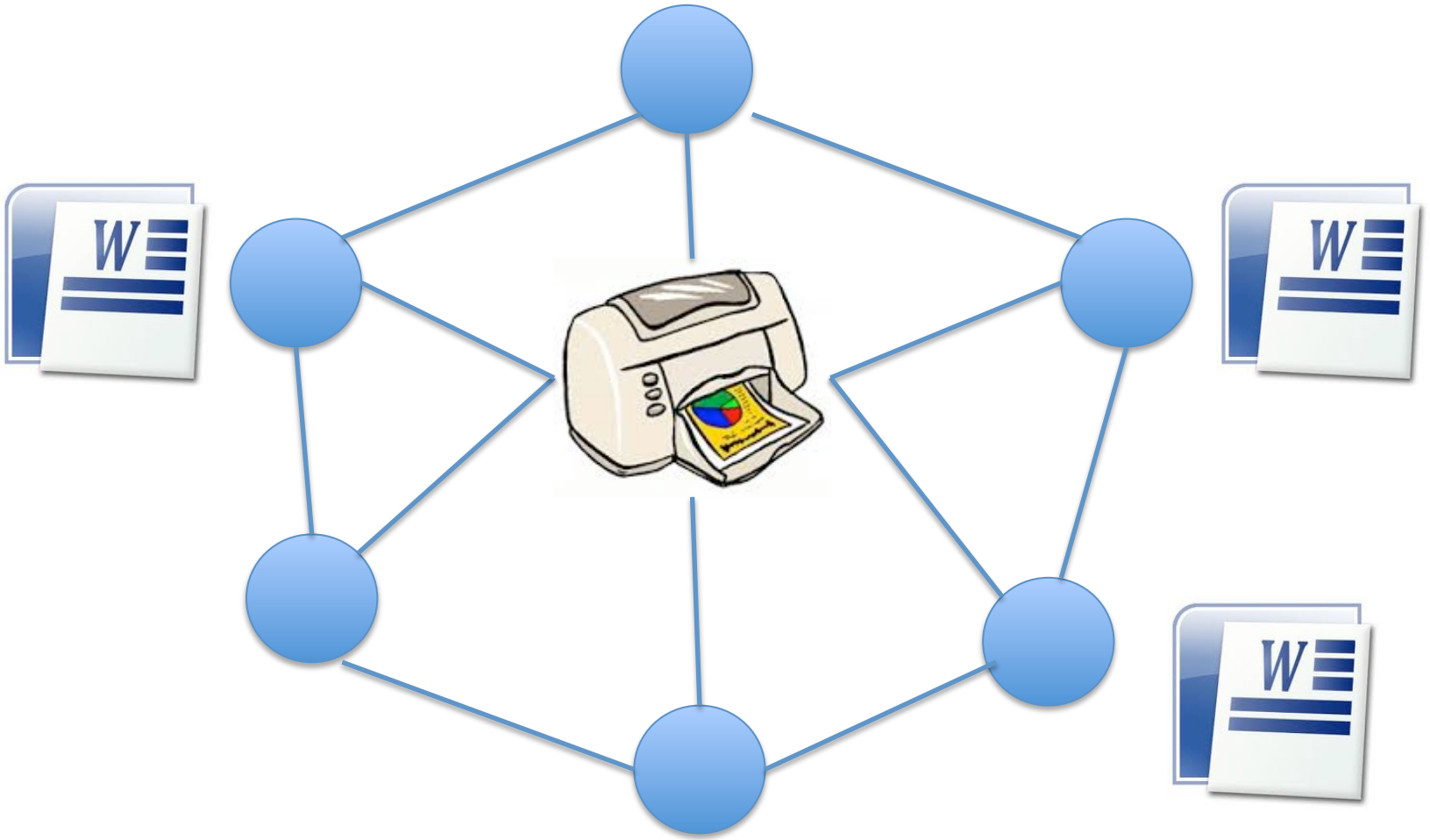
- Ex. k=2



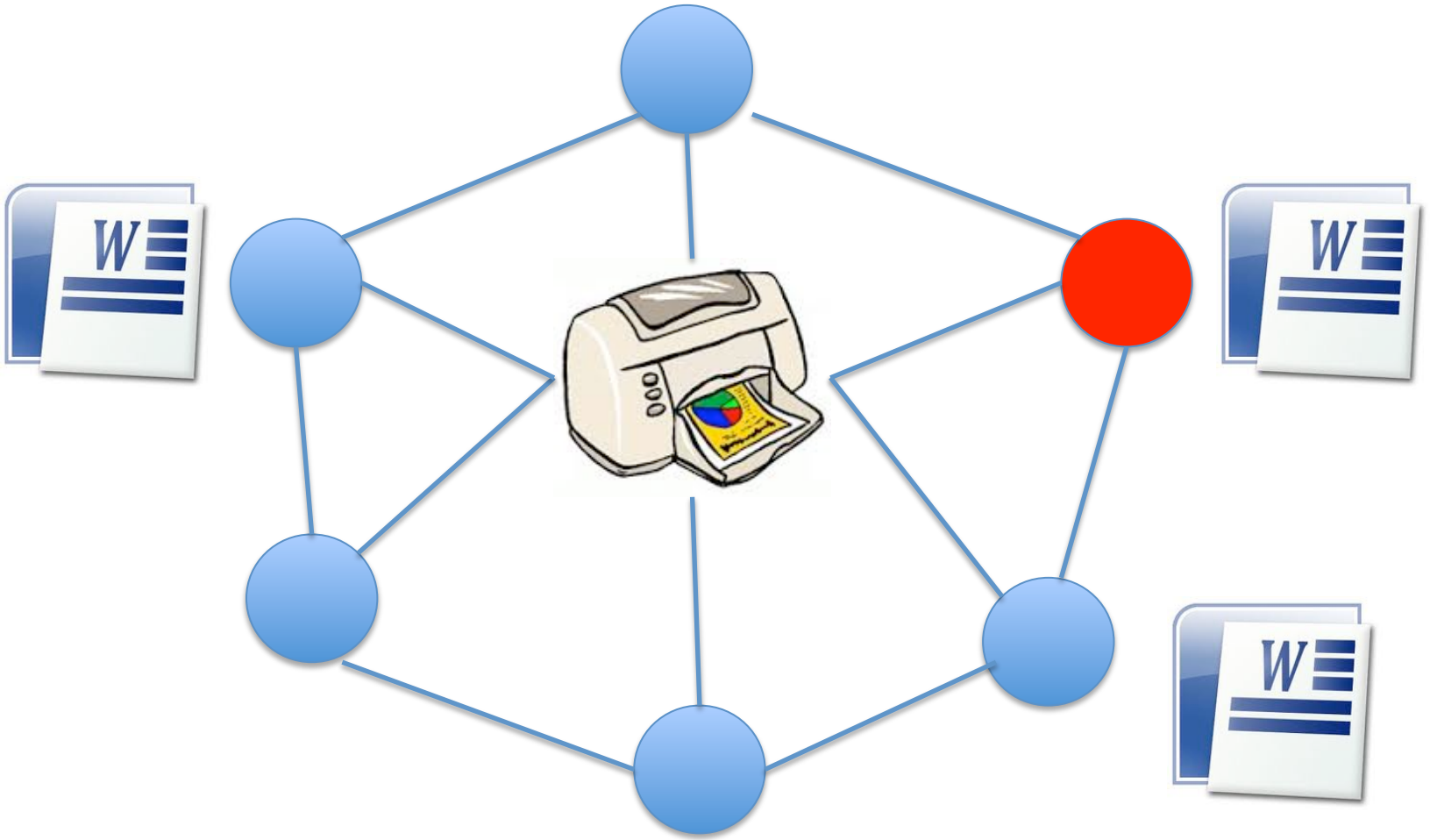
Allocation de ressources : exclusion mutuelle



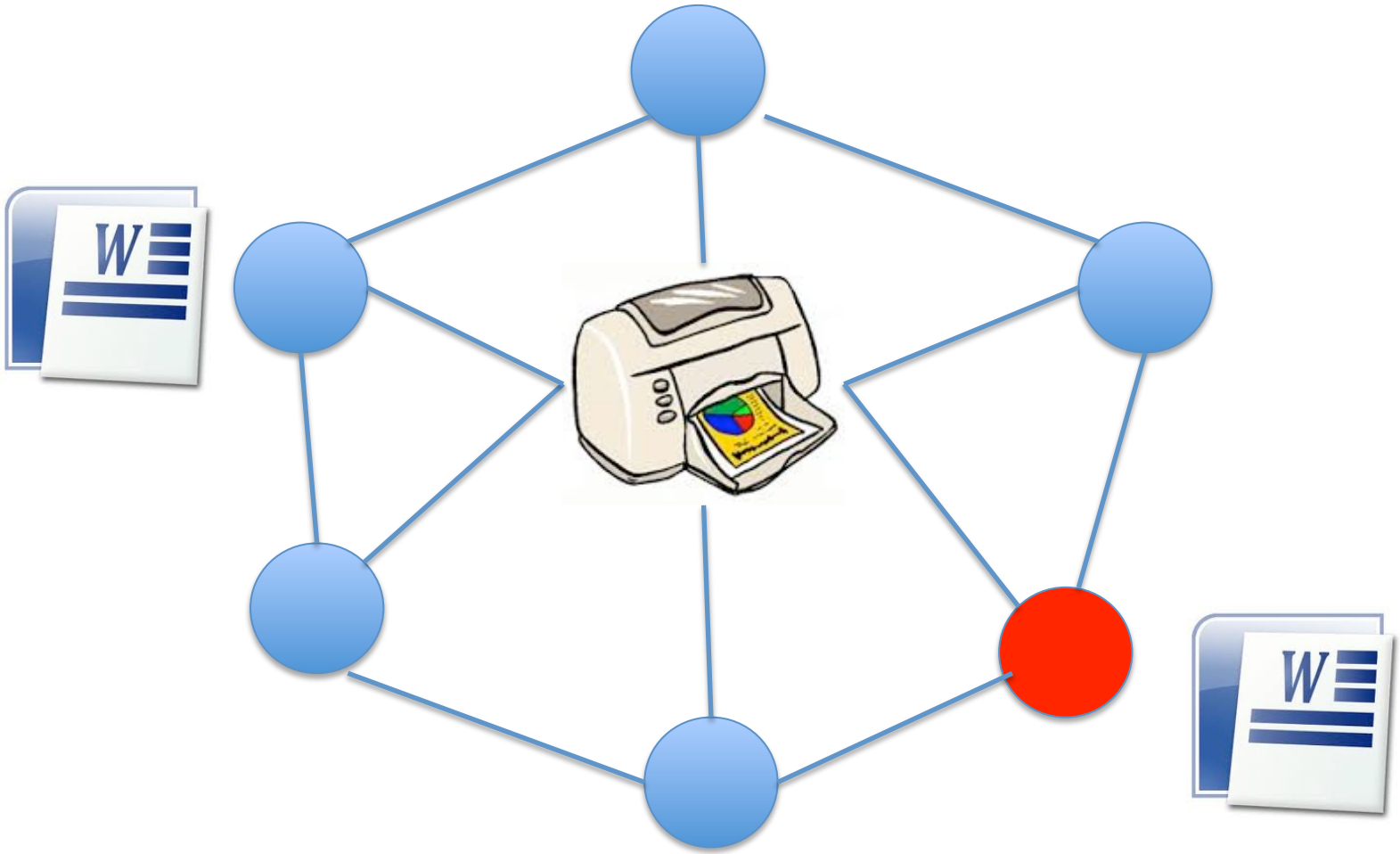
Allocation de ressources : exclusion mutuelle



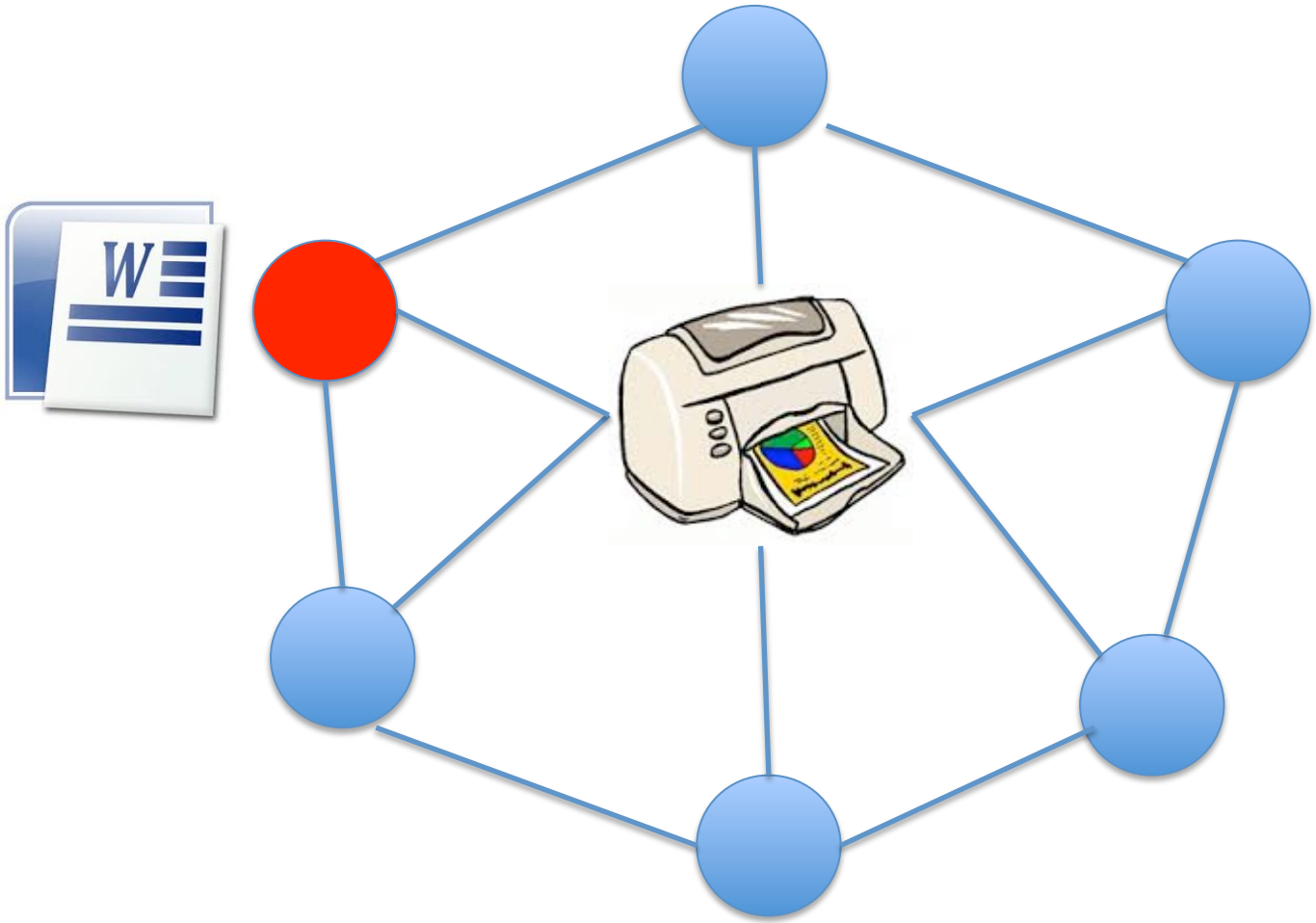
Allocation de ressources : exclusion mutuelle



Allocation de ressources : exclusion mutuelle



Allocation de ressources : exclusion mutuelle



Exemple trivial :
Exclusion mutuelle de Le Lann

Méthodologie

1. Spécifier le problème à résoudre
2. Fixer les hypothèses sur le système
3. Ecrire un algorithme
4. Prouver que l'algorithme vérifie la spécification sous les hypothèses fixées
5. Etudier la complexité
6. Implanter l'algorithme

Spécification

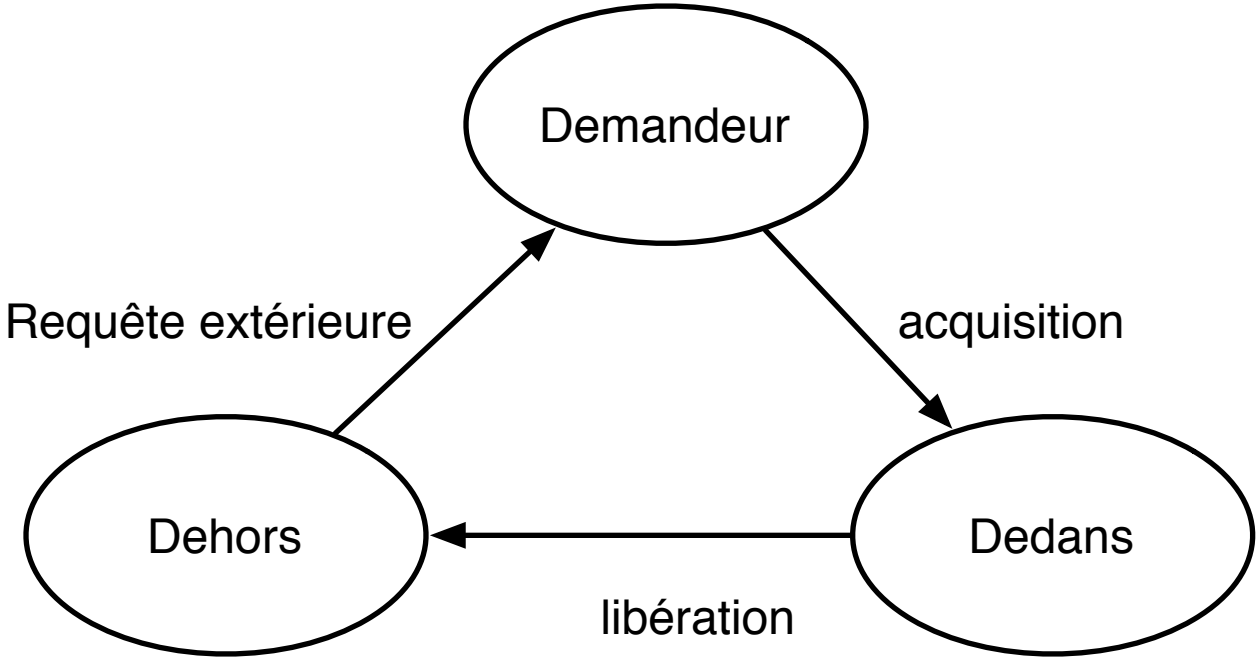
- Enoncé formel du problème que l'algorithme doit résoudre

- **Sûreté (Safety) + Vivacité (Liveness)**
  - **Sûreté :** l'ensemble des propriétés qui doivent être vérifiées à chaque instant de l'exécution de l'algorithme. « Rien de mal ne doit arriver. »

  - **Vivacité :** L'ensemble des propriétés qui doivent être vérifiées à certains instants de l'exécution de l'algorithme. « Quelque chose de bien finit par arriver. »

Spécification de l'exclusion mutuelle

- accès concurrentiel à une ressource partagée unique via une « section critique » du code
  - (*ex.* une imprimante).



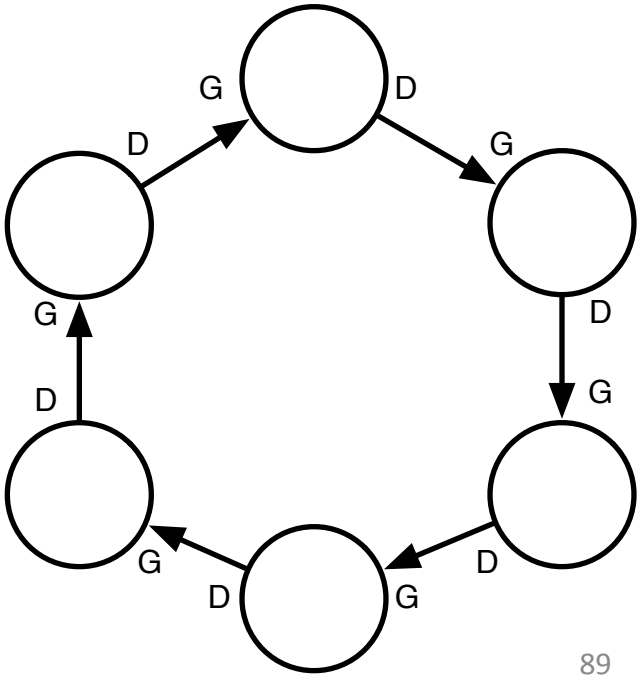
Spécification de l'exclusion mutuelle

- **Sûreté :** Au plus un processus est à la fois dans la section critique.

- **Vivacité :** Tout processus demandeur finit par entrer en section critique.

Hypothèses

- Processus et canaux asynchrones
- Pas de fautes
- Topologies : anneaux unidirectionnel avec orientation consistante
- Au moins deux processus
- Un seul initiateur
- Section critique : finie, mais non bornée



Syntaxe générale

- 2 primitives :
  - Envoyer <MessageType, liste de donnée …> à X
    - X est un **numéro de canal** ou une identité
  - Réception <MessageType, liste de donnée …> depuis X
    - X est un **numéro de canal** ou une identité
    - Vaut VRAI ou FAUX (réception non-bloquante)

Syntaxe générale

Algorithme 1 Algorithme $\mathcal{A}$ pour tout process p

```
1:  Instructions d'initialisation        /* pas de déclaration de variable ! */
2:  Tant que vrai faire
3:      Instructions
4:      Pour tout q ∈ V \ {p} faire        /* q est un numéro de canal ou une identité */
5:          Si réception ⟨TypeMess, liste variables ...⟩ depuis q alors
6:              Instructions
7:          Fin Si
8:      Fin Pour
9:      Instructions
10:     Si réception ⟨TypeMess, liste variables ...⟩ depuis X alors
11:         Instructions
12:     Fin Si
13:     Instructions
14: Fin Tant que
```

Algorithme 2 Exclusion mutuelle de Le Lann

```
1: Si Initiateur alors                    /* Code d'initialisation */
2:     Envoyer ⟨J⟩ à D
3: Fin Si
4: Tant que vrai faire
5:     Si réception ⟨J⟩ depuis G alors
6:         Si Etat = demandeur alors
7:             Etat ← dedans
8:             SC      /* Section critique */
9:             Etat ← dehors
10:        Fin Si
11:        Envoyer ⟨J⟩ à D
12:    Fin Si
13: Fin Tant que
```

Preuve de correction

(La preuve est triviale, mais voici une version très détaillée.)

Lemme 1 (Sûreté). *Jamais plus d'un processus n'est en section critique.*

Preuve.

- Une seule création : A l'initialisation, un seul jeton est créé car il n'y a qu'un seul initiateur.
- Pas de duplication : Chaque processus relaie un message « Jeton » à droite que s'il l'a reçu préalablement de la gauche.

Donc, le jeton créé initialement reste unique dans le système pendant toute l'exécution. Comme un processus ne peut exécuter la section critique que s'il détient le jeton, le lemme est vérifié. □

Preuve de correction

Lemme 2 (Vivacité). *Tout demandeur entre en section critique en temps fini.*

Preuve.

1. L'initialisation dure un temps fini. (algorithme)
2. Le temps d'exécution de la section critique est fini. (hypothèse)
3. Le temps d'acheminement des messages est fini. (hypothèse)
4. Le temps de traitement des messages est fini. (hypothèse)
5. Toute réception d'un jeton est suivie d'un envoi. (algorithme)
6. Le jeton se déplace toujours dans le même sens. (algorithme)

On a donc une circulation de jeton unidirectionnelle perpétuelle. Ainsi, tout demandeur finit par obtenir le jeton et ainsi finit par exécuter la section critique. □

Preuve de correction

D'après les deux lemmes précédents, nous avons :

Théorème 1. *L'algorithme 1 résout l'exclusion mutuelle dans un anneau unidirectionnel.*

Remarque 2. *Si on lève l'une des hypothèses, la preuve ne marche plus !*

Complexité

En nombre de messages et en temps d'exécution dans le meilleur et le pire des cas.

- Complexité pour un tour de jeton :
  - n
- Si un processus est demandeur en cours d'exécution, quel est le nombre de messages générés avant que le processus entre en section critique
  - (pire : $n - 1$, meilleur : 0)
- Temps de service : combien d'autres processus peuvent exécuter la section critique avant qu'un processus (demandeur) particulier ne le fasse
  - (pire : n − 1, meilleur : 0)
- Ratio nombre de messages / nombre de demandes
  - (pire : ∞ — aucune demande, meilleur : 1 — tous demandeurs)

Conclusion sur l'algorithme

- Le dernier résultat montre un inconvénient majeur de ce type de solution (proactive) : les échanges de messages continuent même s'il n'y a aucune demande.

- Pour régler ce problème, il existe des algorithmes dit « à permission » (réactive)

Deuxième Exemple : Circulation d'un jeton dans un réseau quelconque

Spécification

La *circulation de jeton*

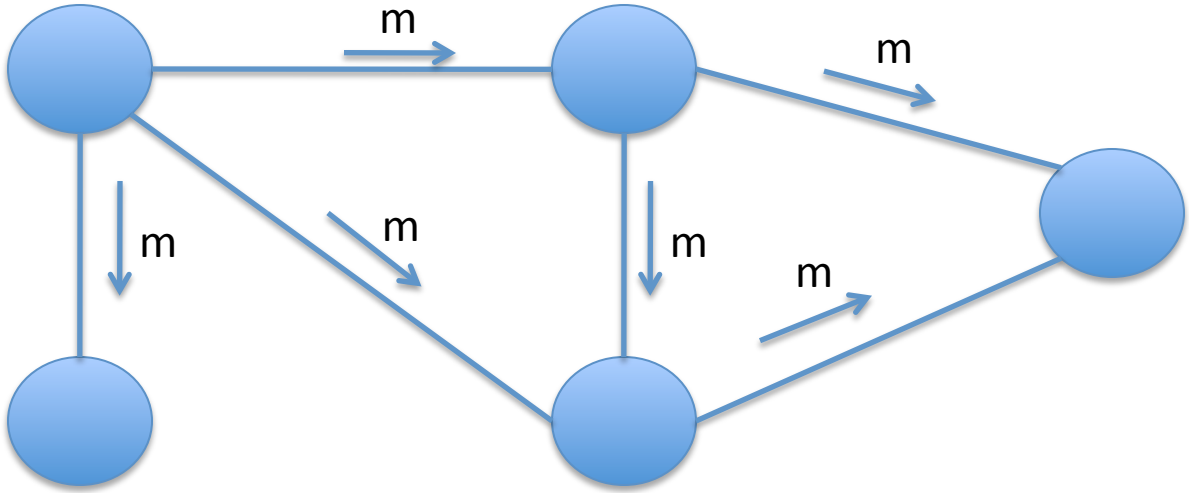
est un

algorithme à vague

Algorithme à vague

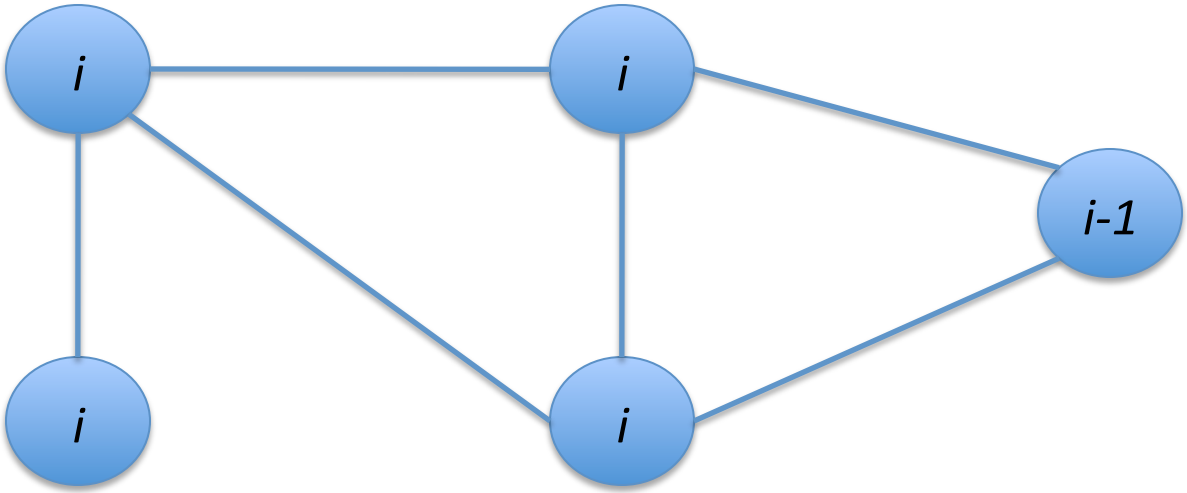
Introduction

- Dans un système distribué, on a (parfois) besoin de :
  - Diffuser des informations (à tous les processus)
    - (Broadcast)



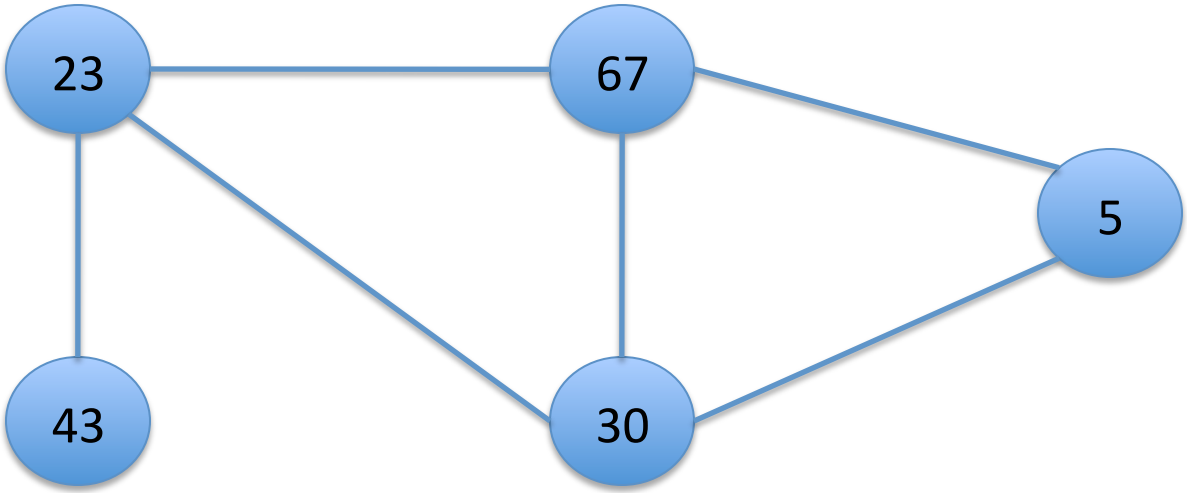
Introduction

- Dans un système distribué, on a (parfois) besoin de :
  - Synchroniser (globalement) les processus
    - *E.g.*, l'étape *i-1* est elle finie ?



Introduction

- Dans un systèmes distribué, on a (parfois) besoin de :
  - Calculer des fonctions globales
    - *E.g.*, quelle est la plus petite identité ?



Introduction

- Ces problèmes ont plusieurs points communs
- D'où, l'idée de trouver un algorithme général

- Les algorithmes à vague

Définition

- Un algorithme à vague vérifie les trois propriétés suivantes :
  - **Terminaison**
  - **Décision**
  - **Dépendance**

Définition

- **Terminaison** : Toutes ses exécutions (ou vagues) sont *finies*
- **Décision** : Chacune de ses exécutions contient au moins un évènement particulier appelé *décision*
- **Dépendance** : Chaque évènement de décision est *causalement précédé* (au sens de Lamport) par au moins un évènement sur chaque processus

Exemples

- Parcours
  - Largeur
  - Profondeur (à l'aide d'un jeton)

- Propagation d'Information avec Retour (PIR)

- Applications : snapshot, détection de terminaison, calcul d'infimum, *etc*.

Instanciation

- Spécificité une (vague de) circulation de jeton
  - **Décision** (de terminaison)
    - Unique
    - Par l'initiateur
  - **Dépendance**
    - Circulation : séquentielle (ordre causal total)

Instanciation

- Une (vague de) circulation de jeton
  - **Sûreté** :
    - Il existe au plus un jeton dans le réseau
    - Au plus une décision est prise (*Décision*)
    - Si une décision est prise, alors tous les processus ont été visités par le jeton (*Dépendance*)
  - **Vivacité**
    - L'exécution termine (*Terminaison*)
    - L'initiateur finit par décider (*Décision*)

Remarque

- Il existe aussi des algorithmes qui exécutent une infinité de vagues
  - *E.g.*, *circulation de jeton perpétuelle* pour l'exclusion mutuelle

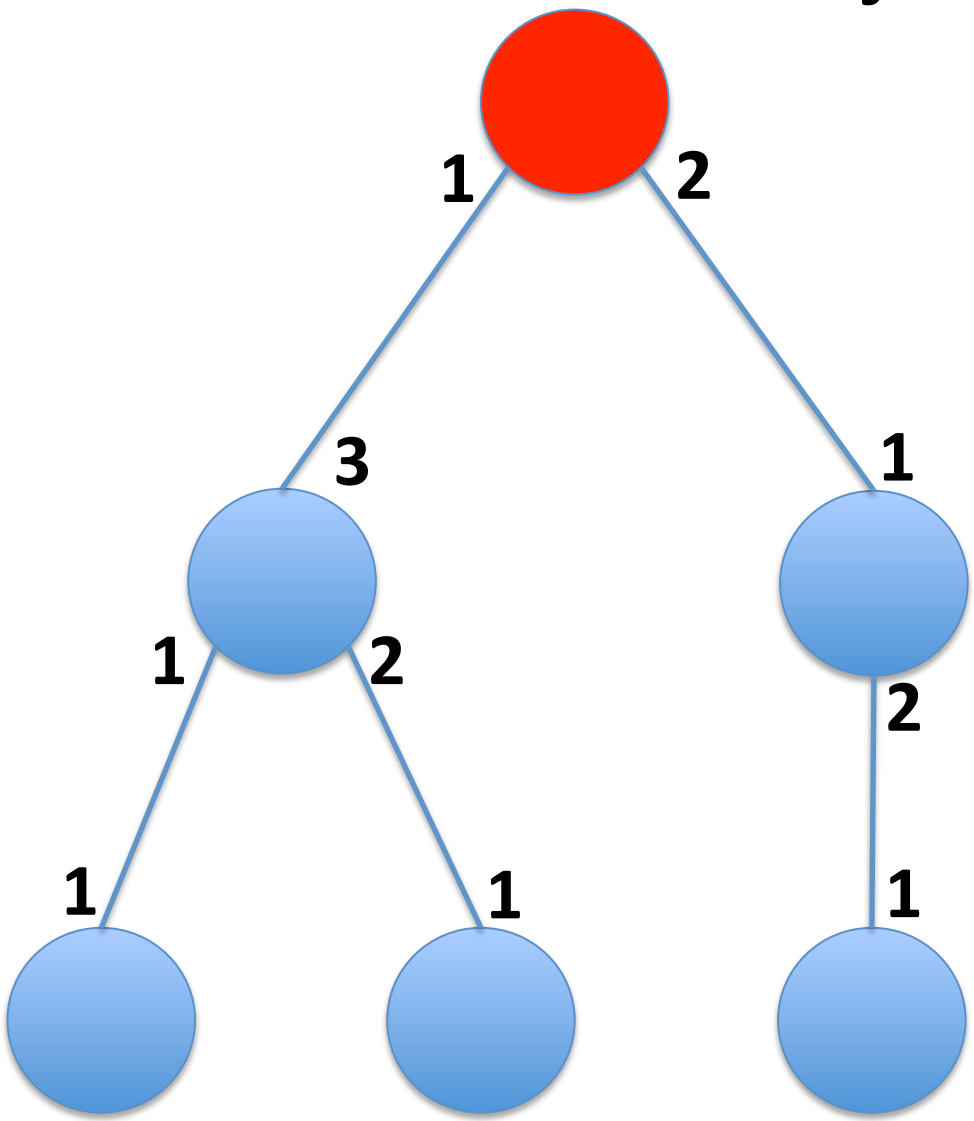
Hypothèses pour notre circulation de jeton

- Processus et canaux asynchrones
- Pas de fautes
- Canaux étiquetés de 1 à δ_p pour tout processus *p*
- Topologies : quelconque (connexe) d'au moins deux nœuds
- Mono-initiateur

Rappel

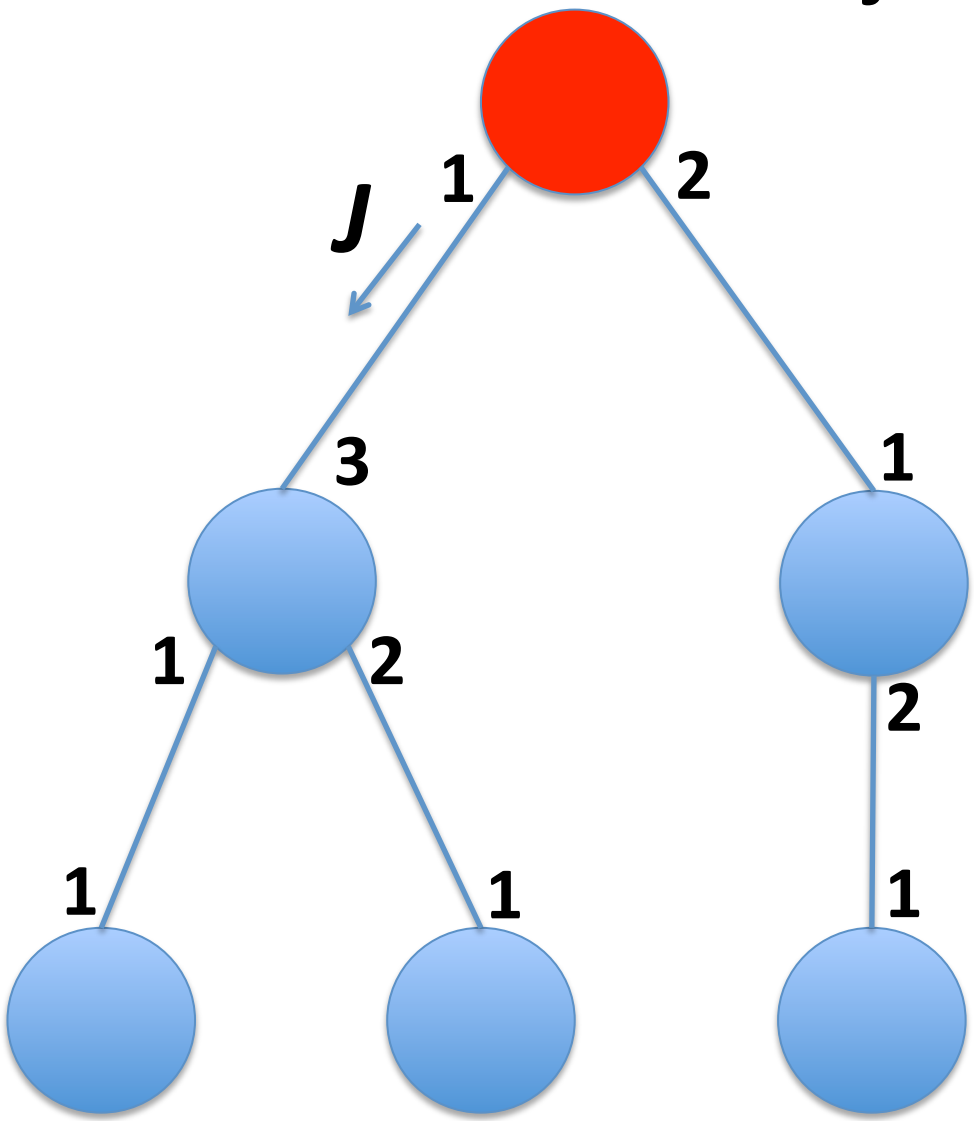
- **Cas plus simple :** circulation dans un réseau en arbre

Circulation d'un jeton dans un arbre



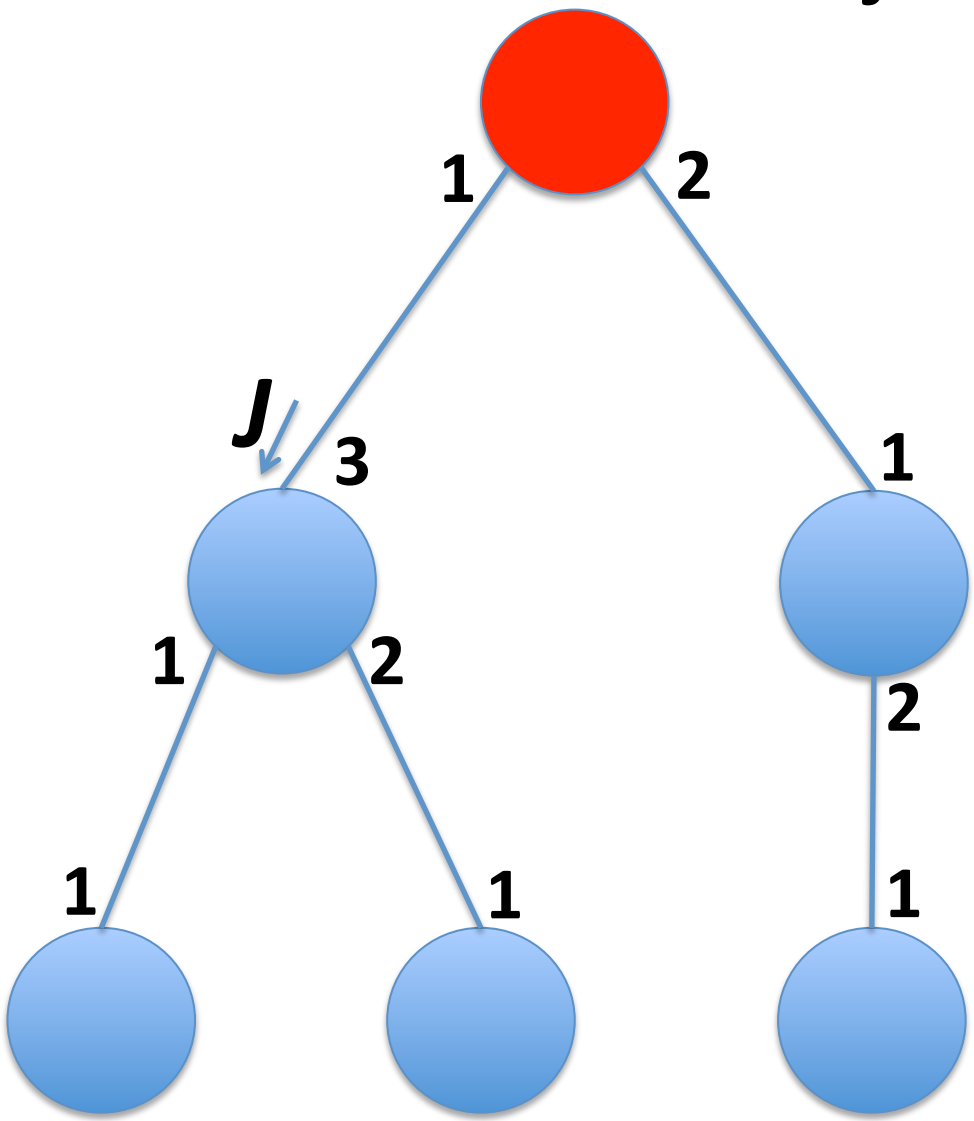
- Vu l'an dernier !

Circulation d'un jeton dans un arbre



- L'initiateur envoie le jeton ***J*** sur le canal **1**

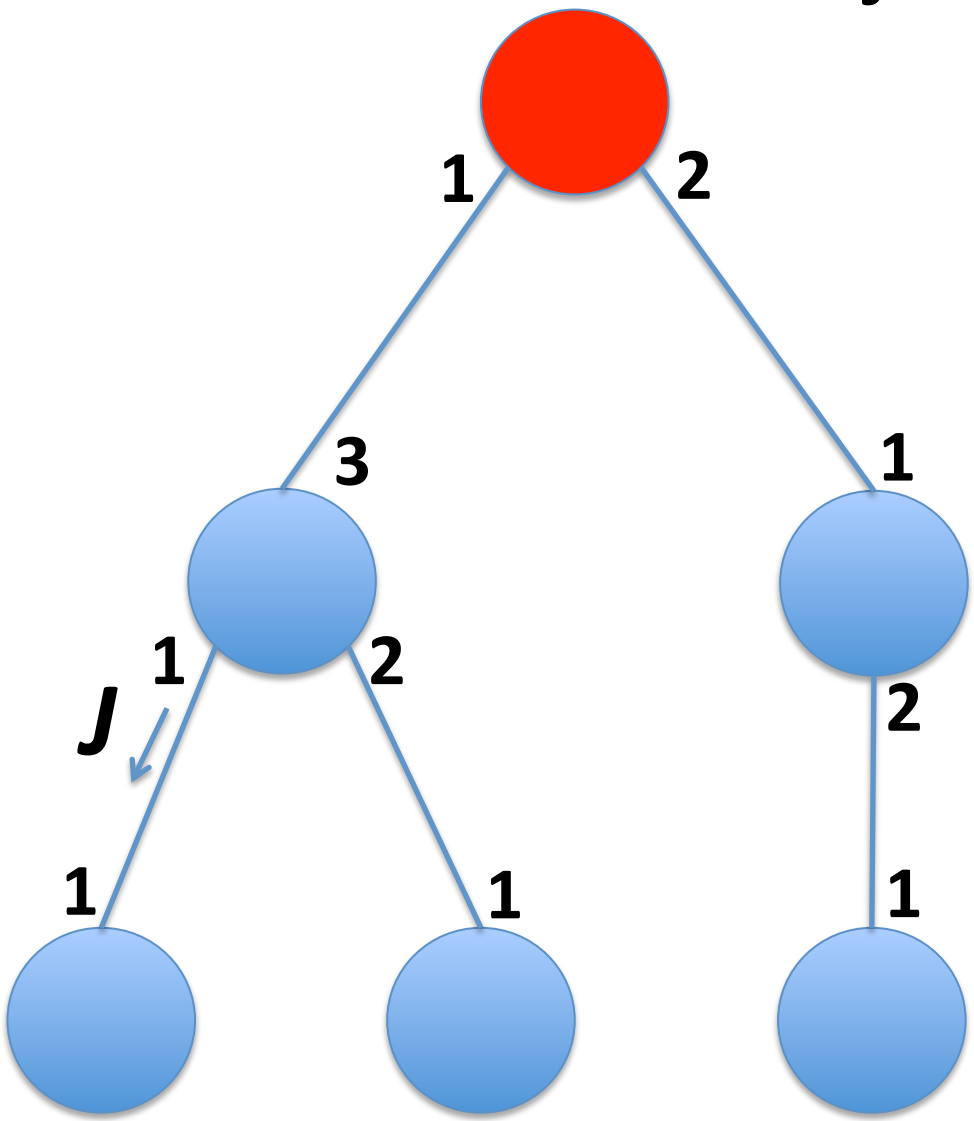
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 3**
- **(3 mod 3) + 1 = 1**

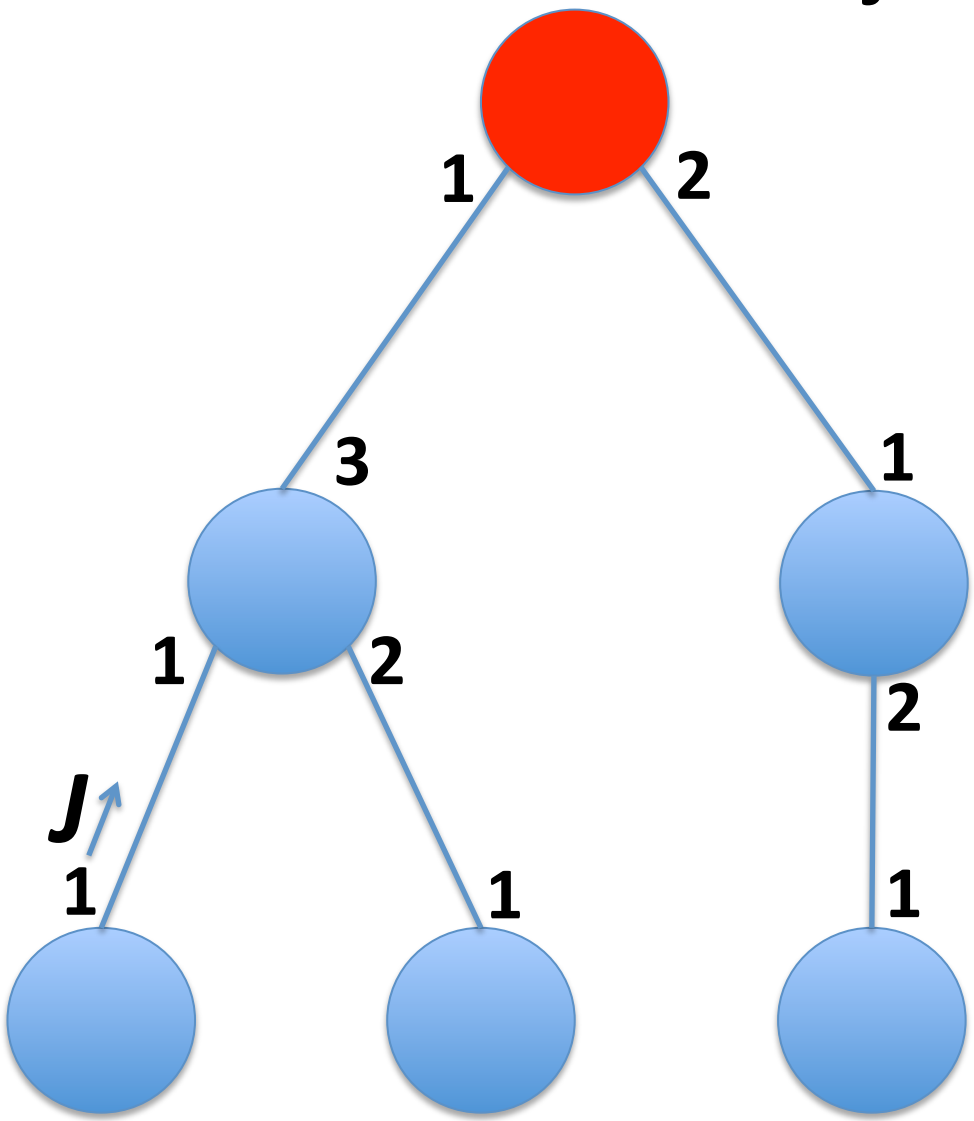
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 3**
- **(3 mod 3) + 1 = 1**

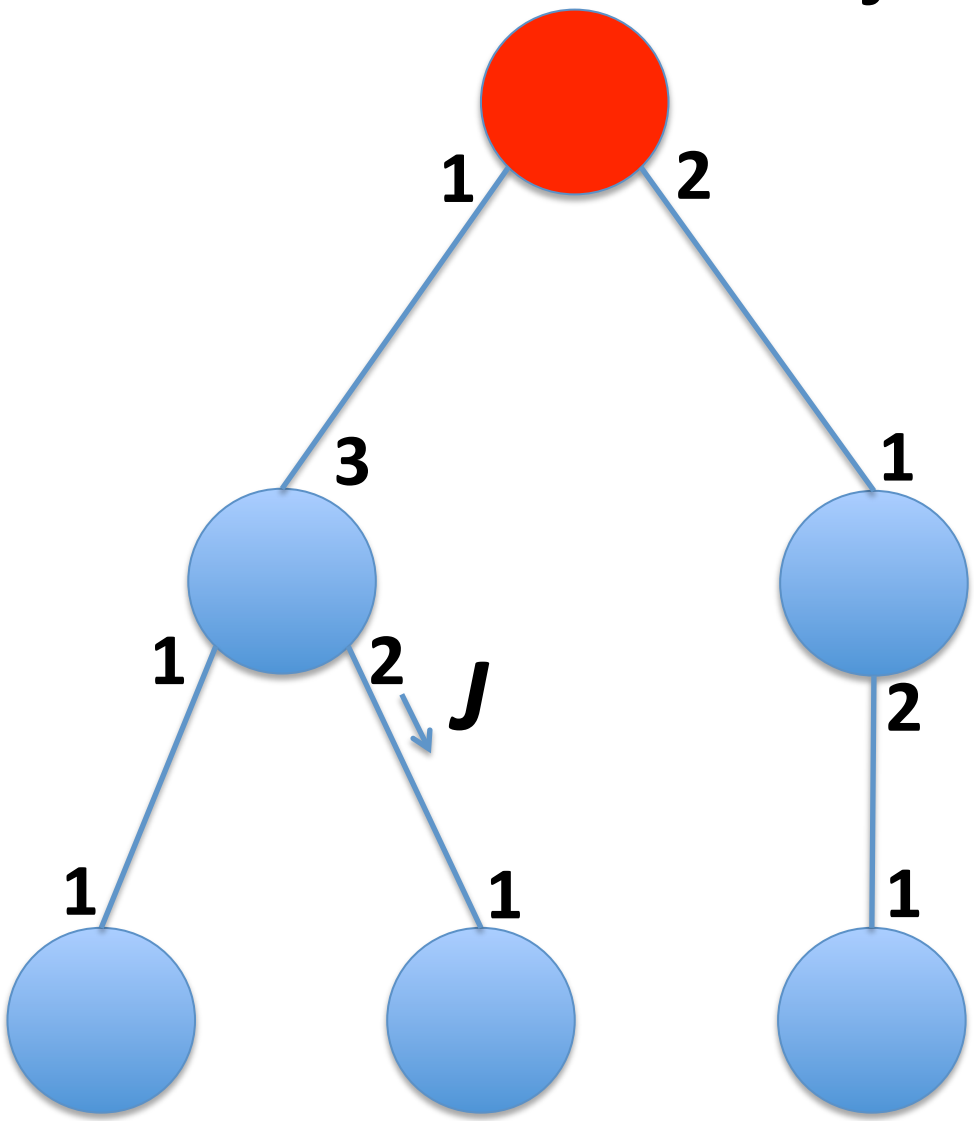
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 1**
- **(1 mod 1) + 1 = 1**

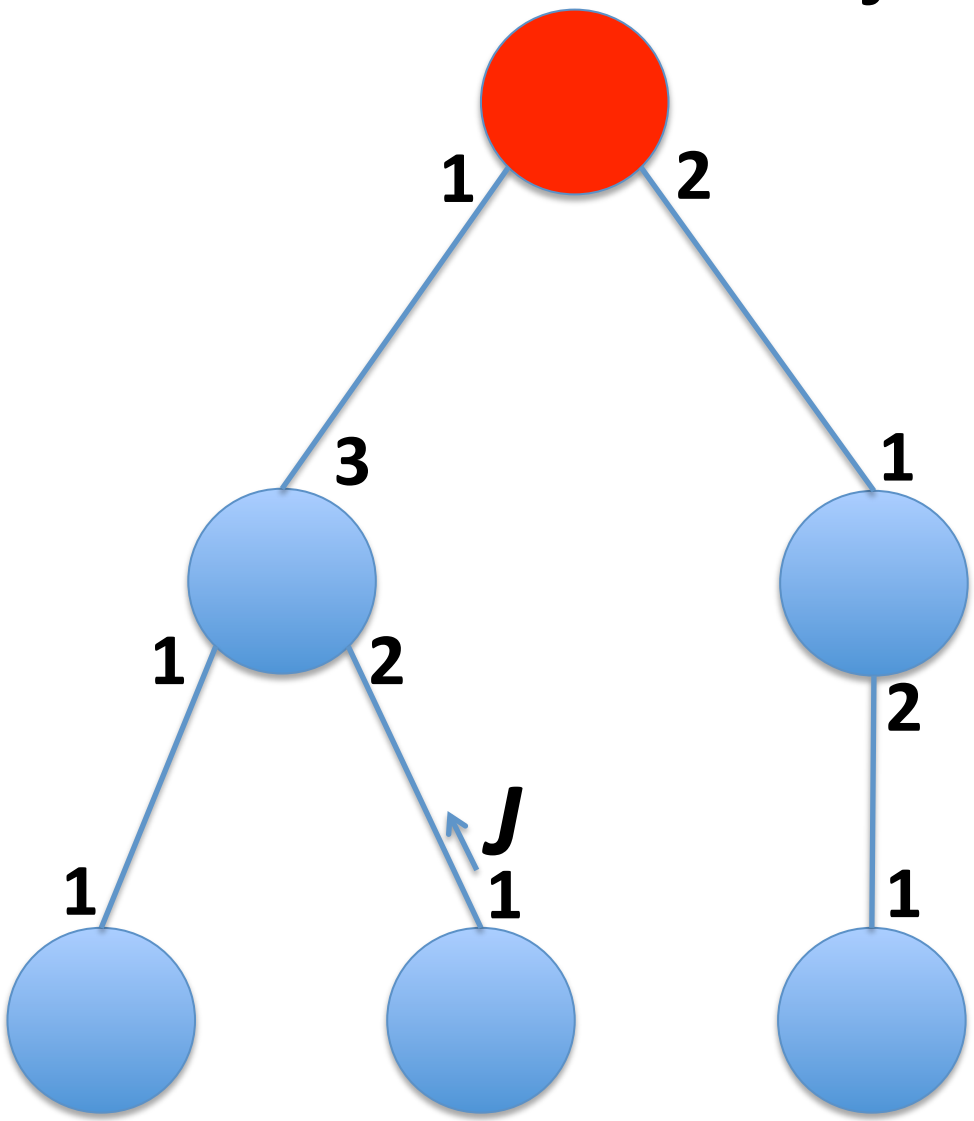
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 3**
- **(1 mod 3) + 1 = 2**

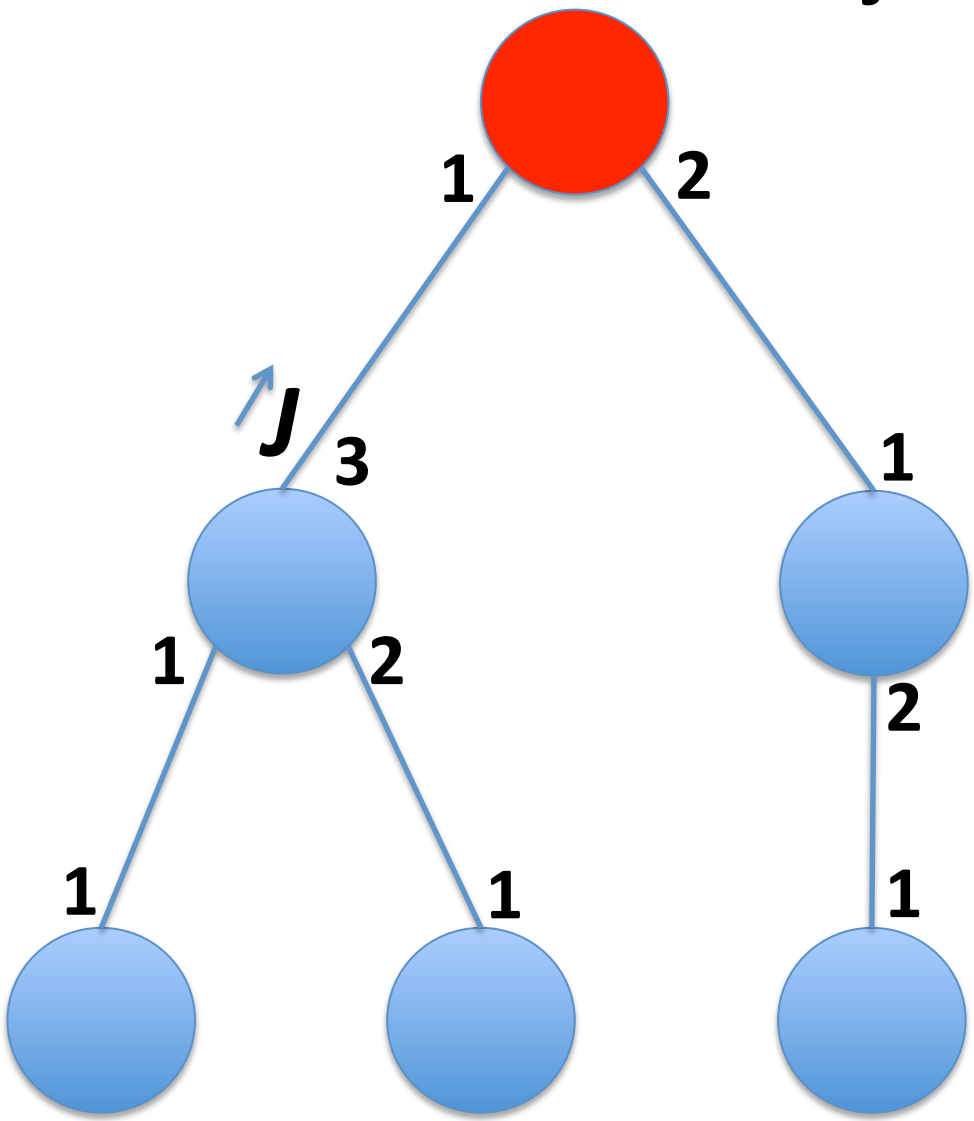
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 1**
- **(1 mod 1) + 1 = 1**

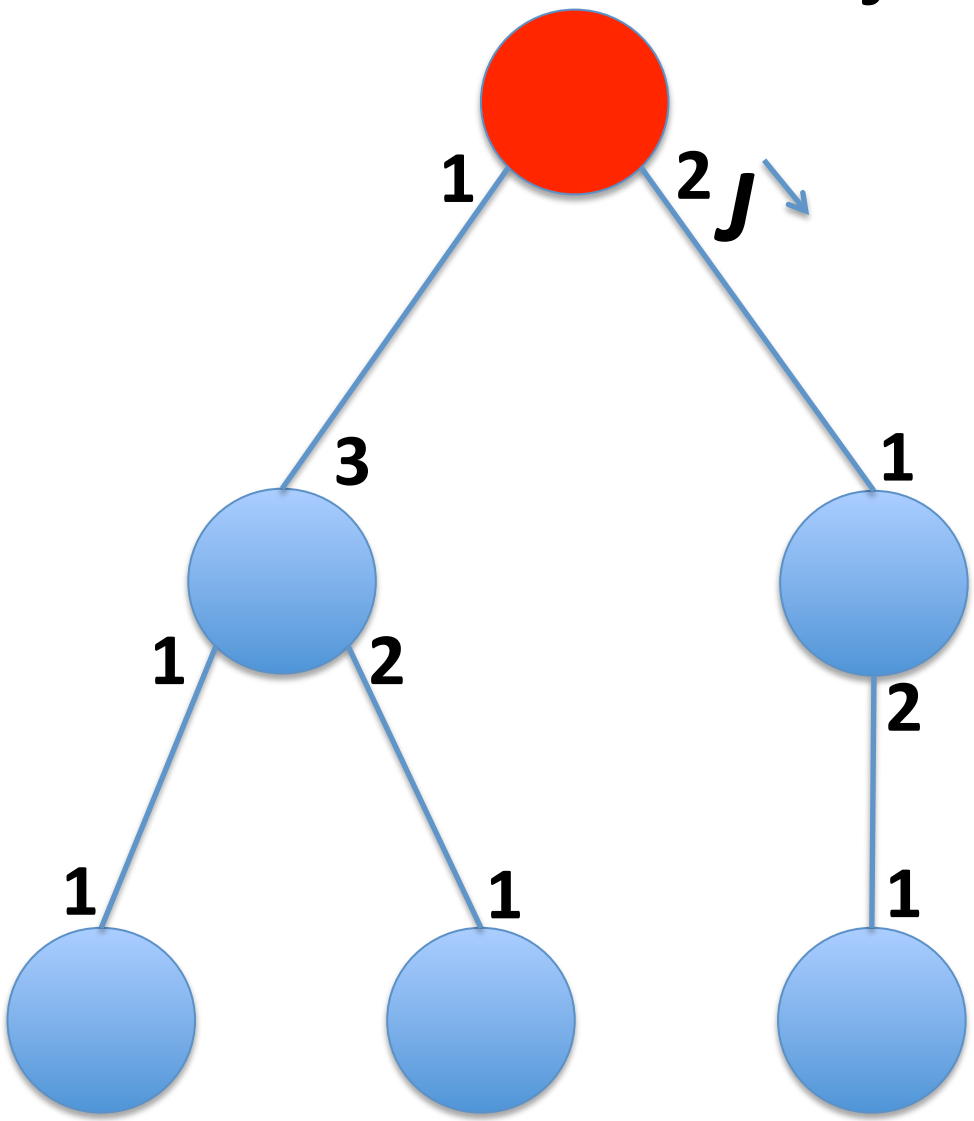
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 3**
- **(2 mod 3) + 1 = 3**

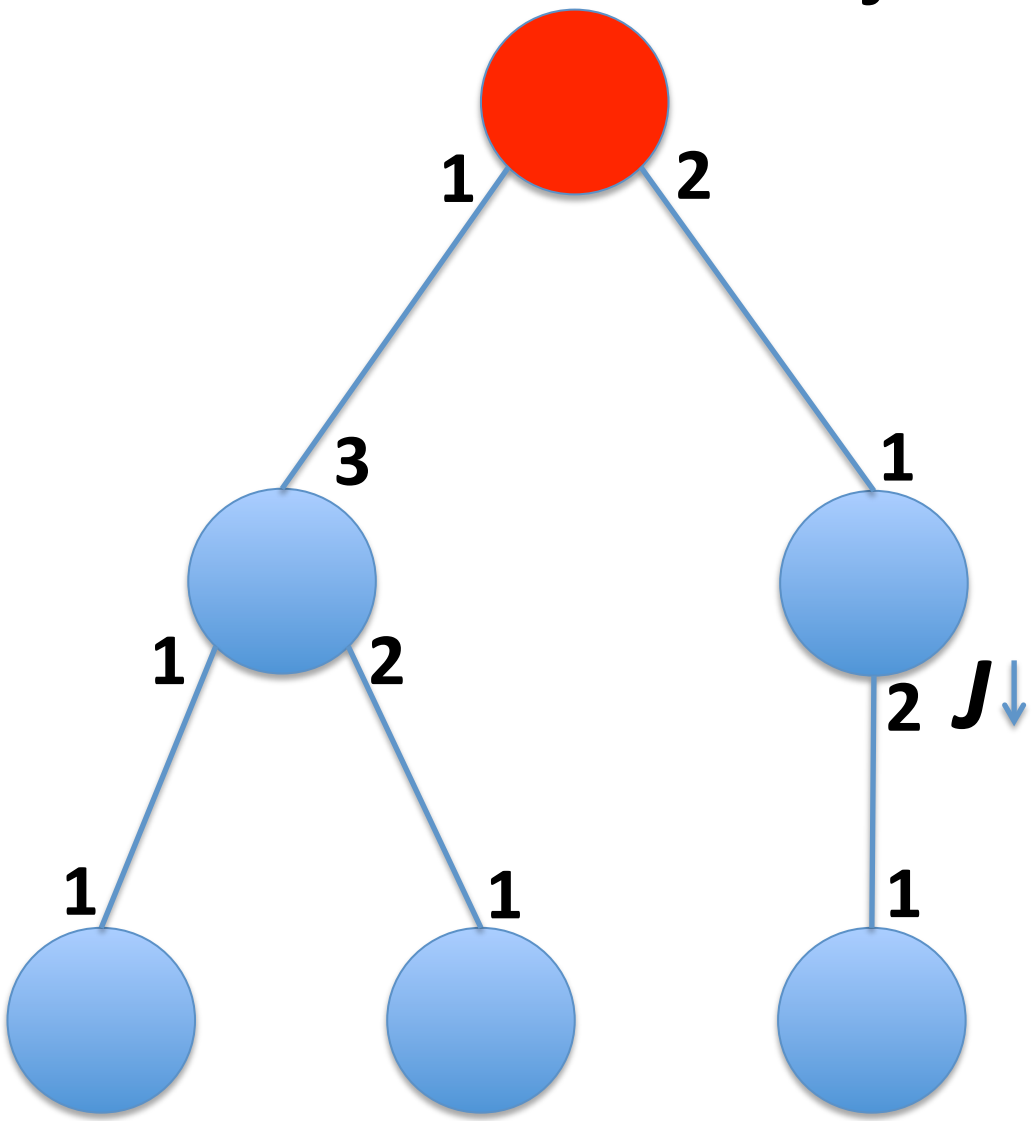
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 2**
- **(1 mod 2) + 1 = 2**

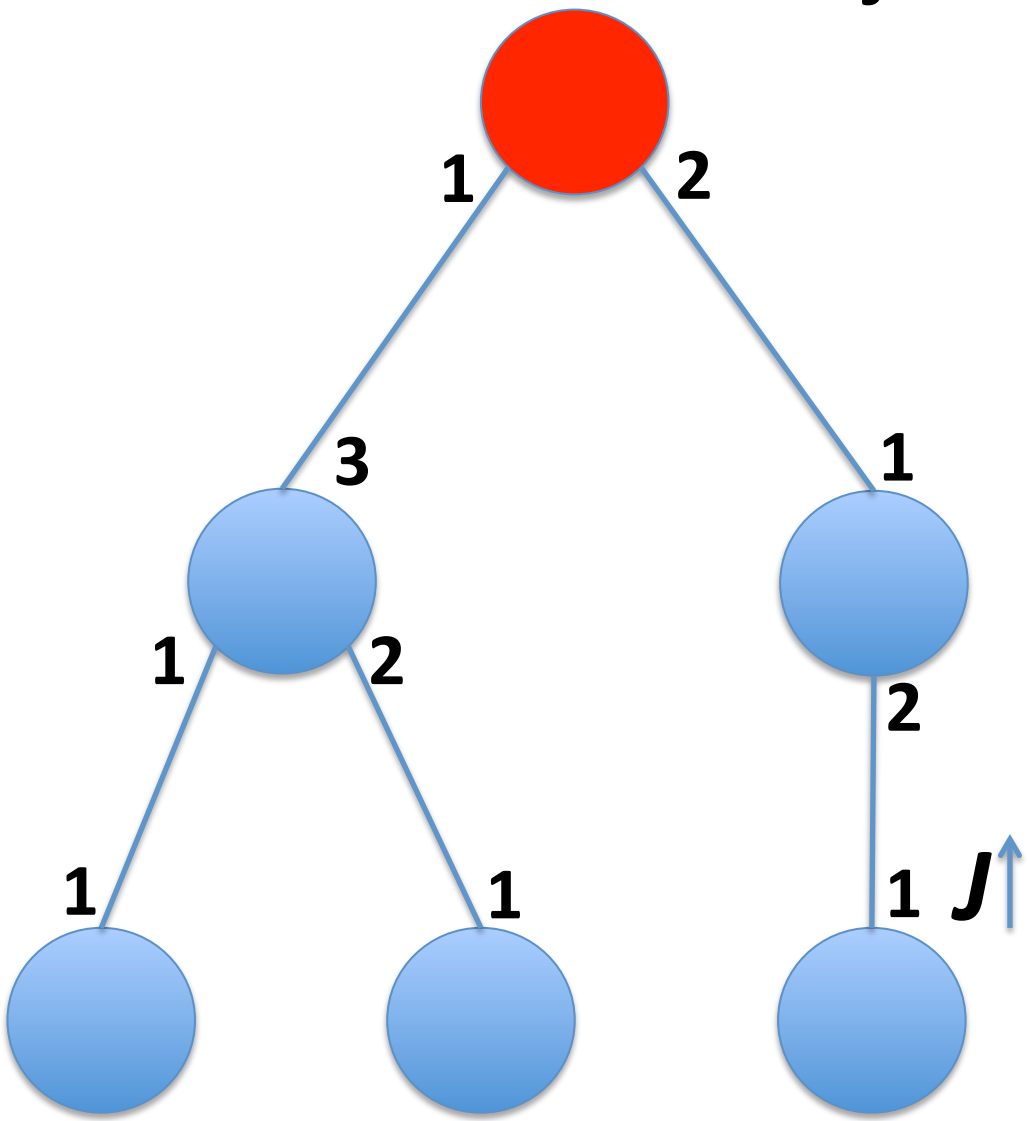
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 2**
- **(1 mod 2) + 1 = 2**

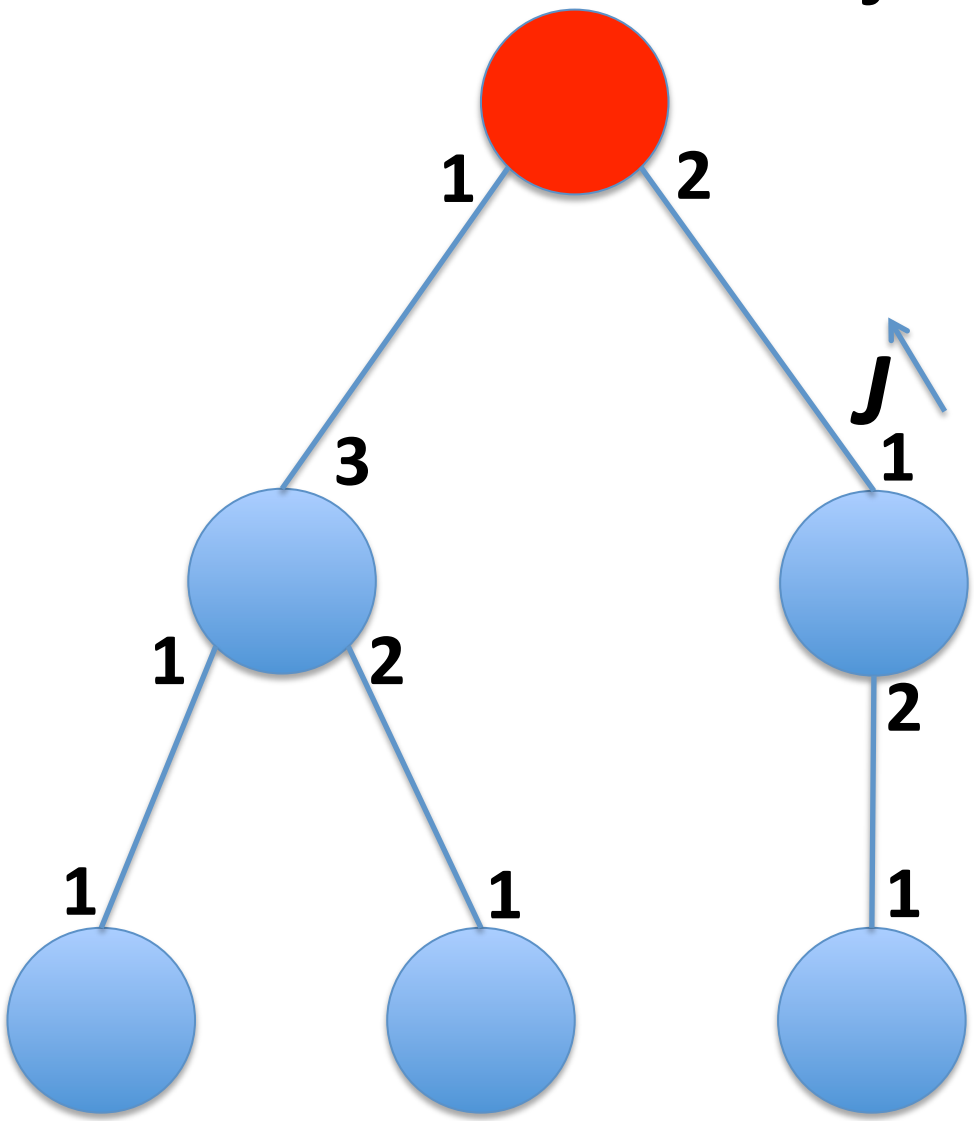
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 1**
- **(1 mod 1) + 1 = 1**

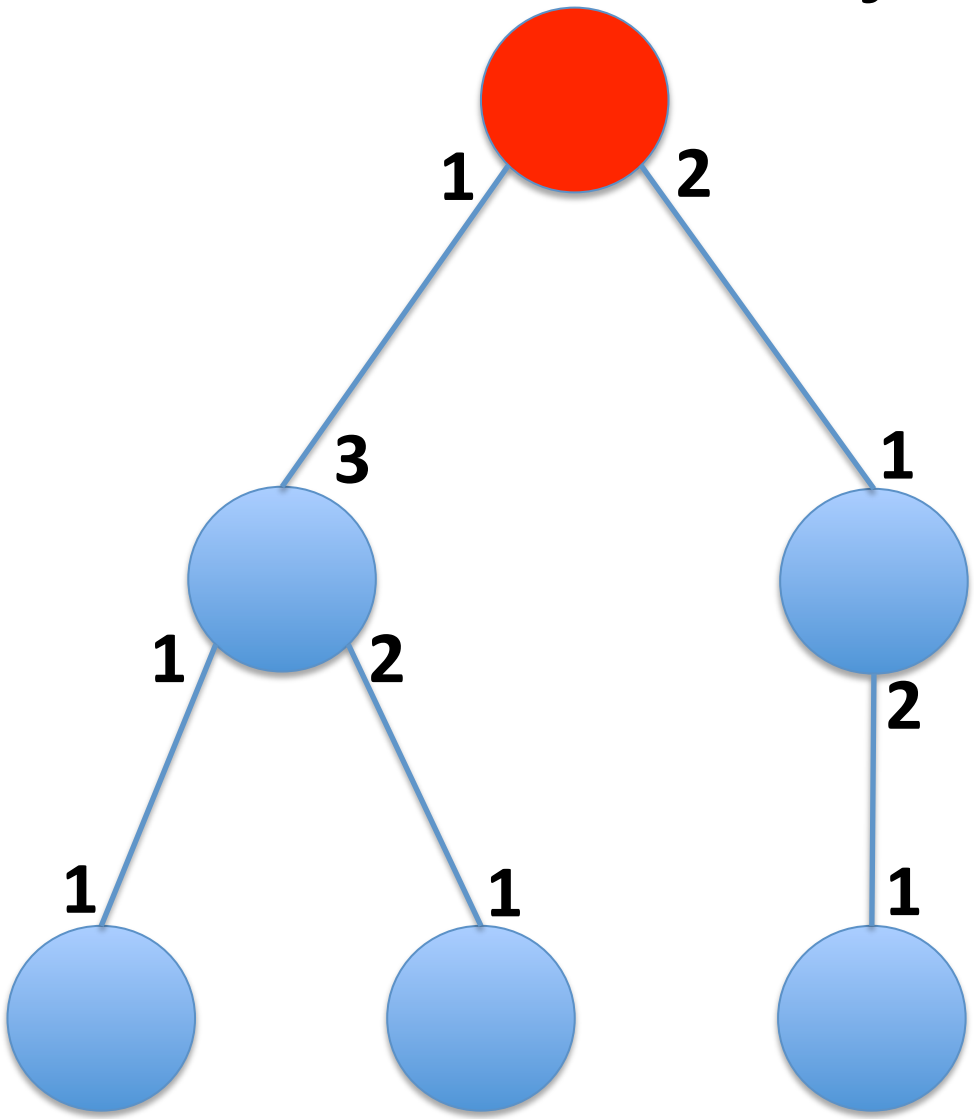
Circulation d'un jeton dans un arbre



- Sur réception du canal **i**, un non-initiateur renvoie le jeton sur le canal **(i mod δ)+1**

- **Ici δ = 2**
- **(2 mod 2) + 1 = 1**

Circulation d'un jeton dans un arbre



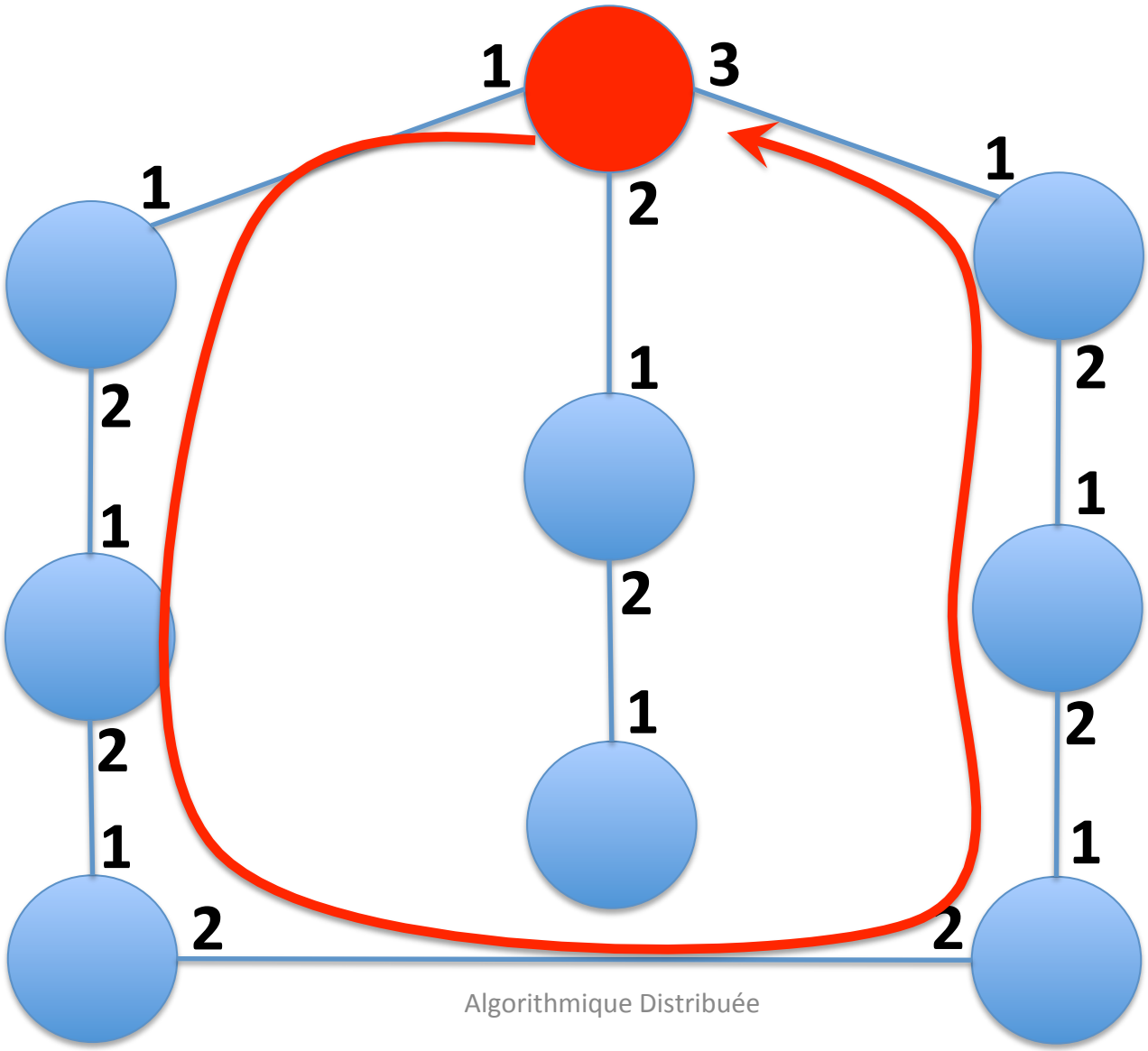
- Sur réception du canal **δ**, l'initiateur **décide** la terminaison
- **Ici δ = 2**

Circulation d'un jeton dans un réseau quelconque ?

- Est-ce que l'algorithme précédent fonctionne ?

NON !

Exemple

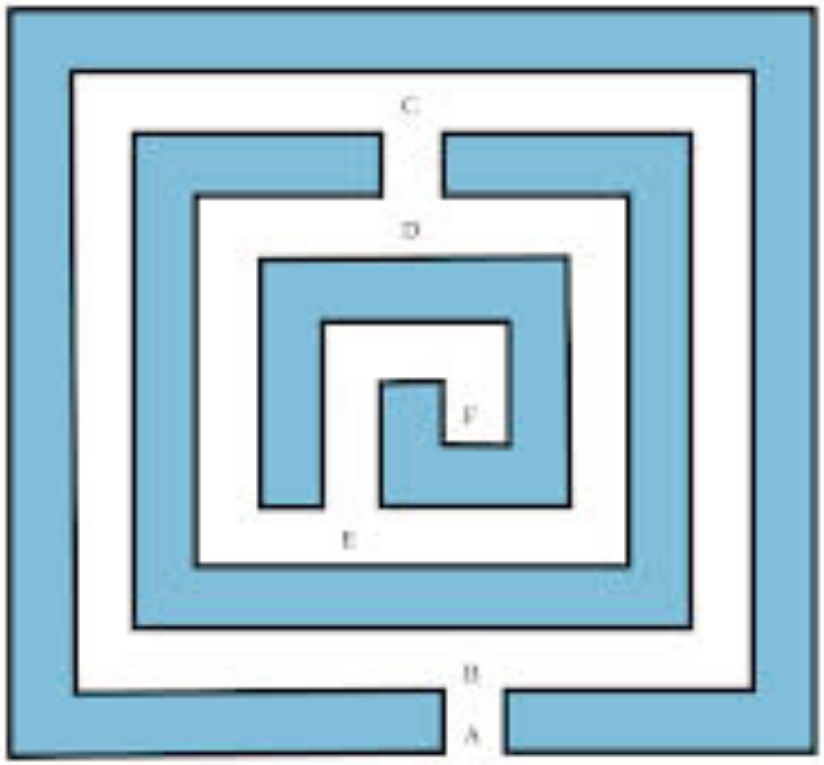


Solution

- Algorithme de Tarry (1885)

- Problème de Labyrinthe
- « Ne reprendre l'**allée initiale** qui a conduit à un **carrefour** pour la première fois que lorsqu'on ne peut pas faire autrement »

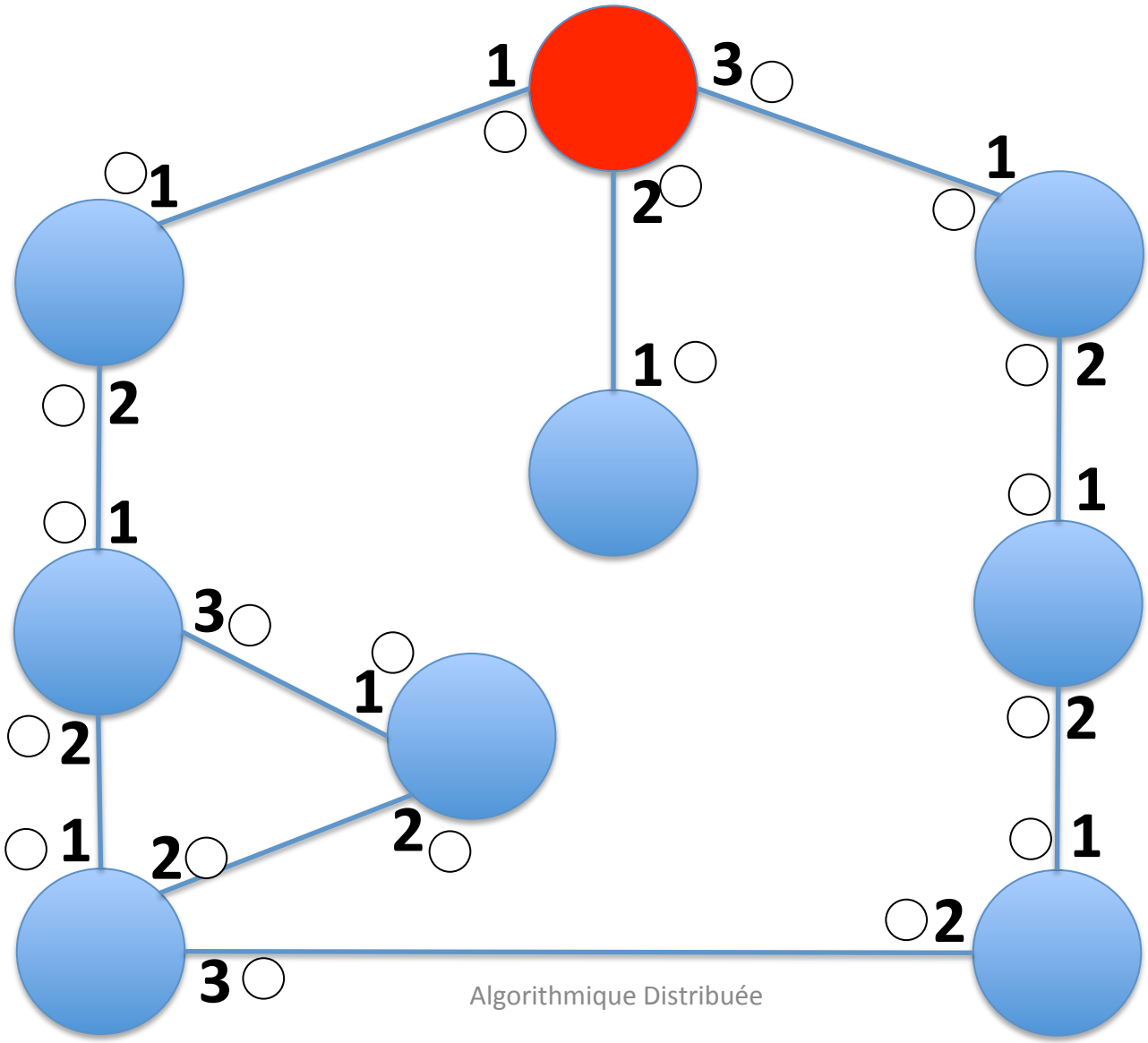
- Sommets = intersections
- Liens = allées entre les intersections des arêtes



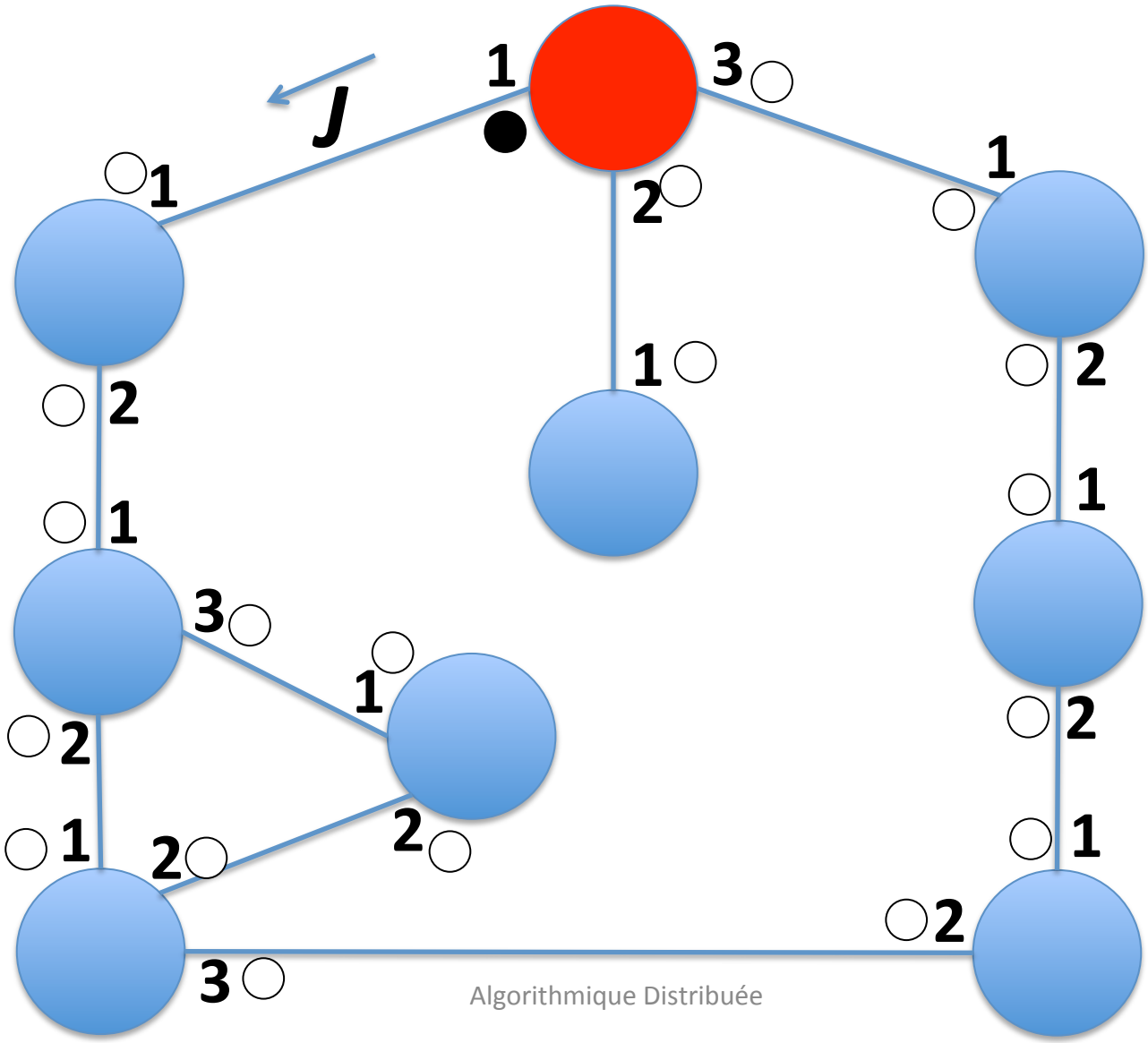
Variables

- Pour chaque processus
  - Un pointeur **Père** $\in \{T, \perp\} \cup \{1 \ldots \delta\}$ initialisé à
    - **T** pour l'initiateur
    - $\perp$ pour les suiveurs
  - Un tableau de Booléen **Visite[1..δ]**, initialement toutes les cases sont à faux.

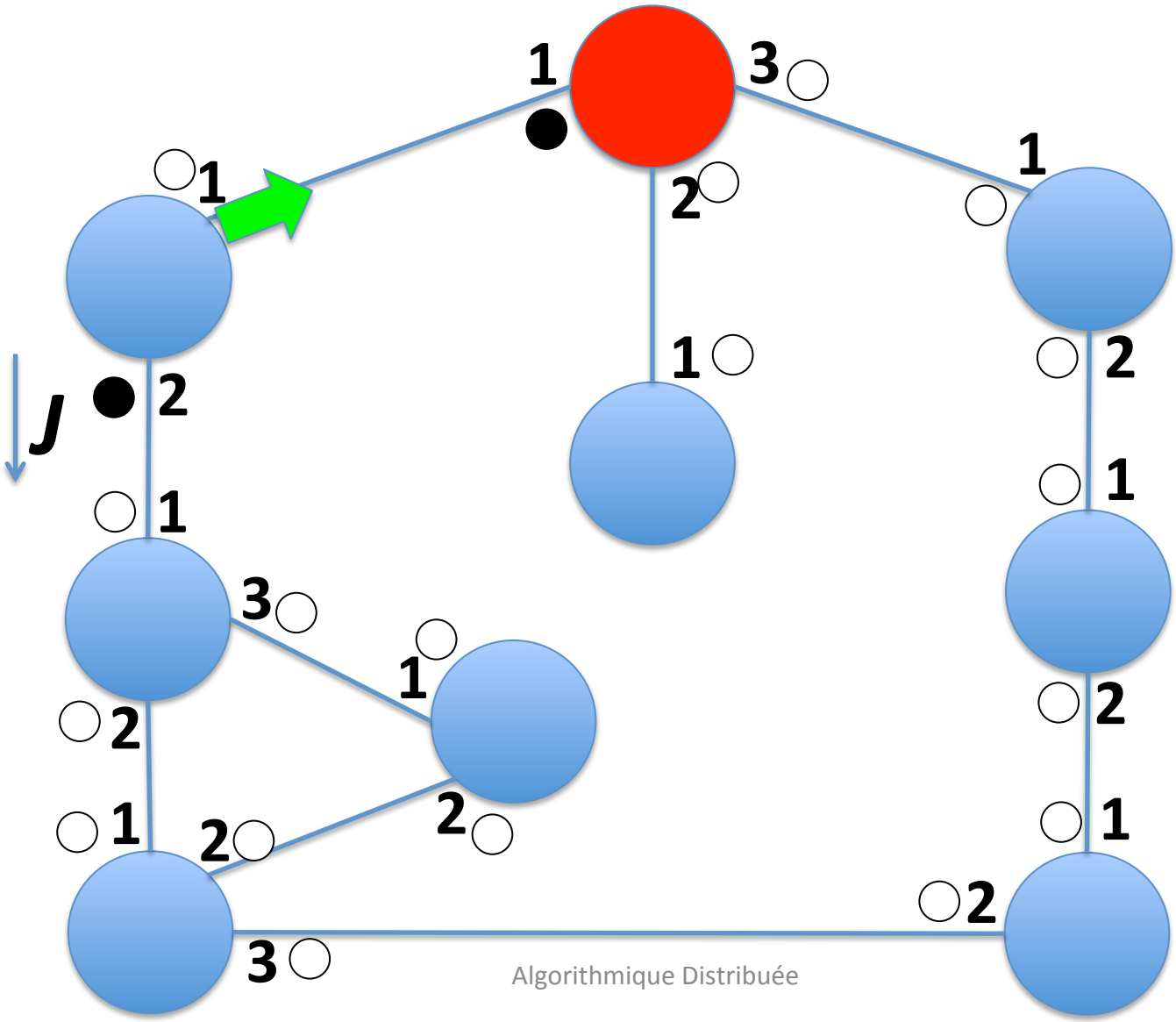
Exemple



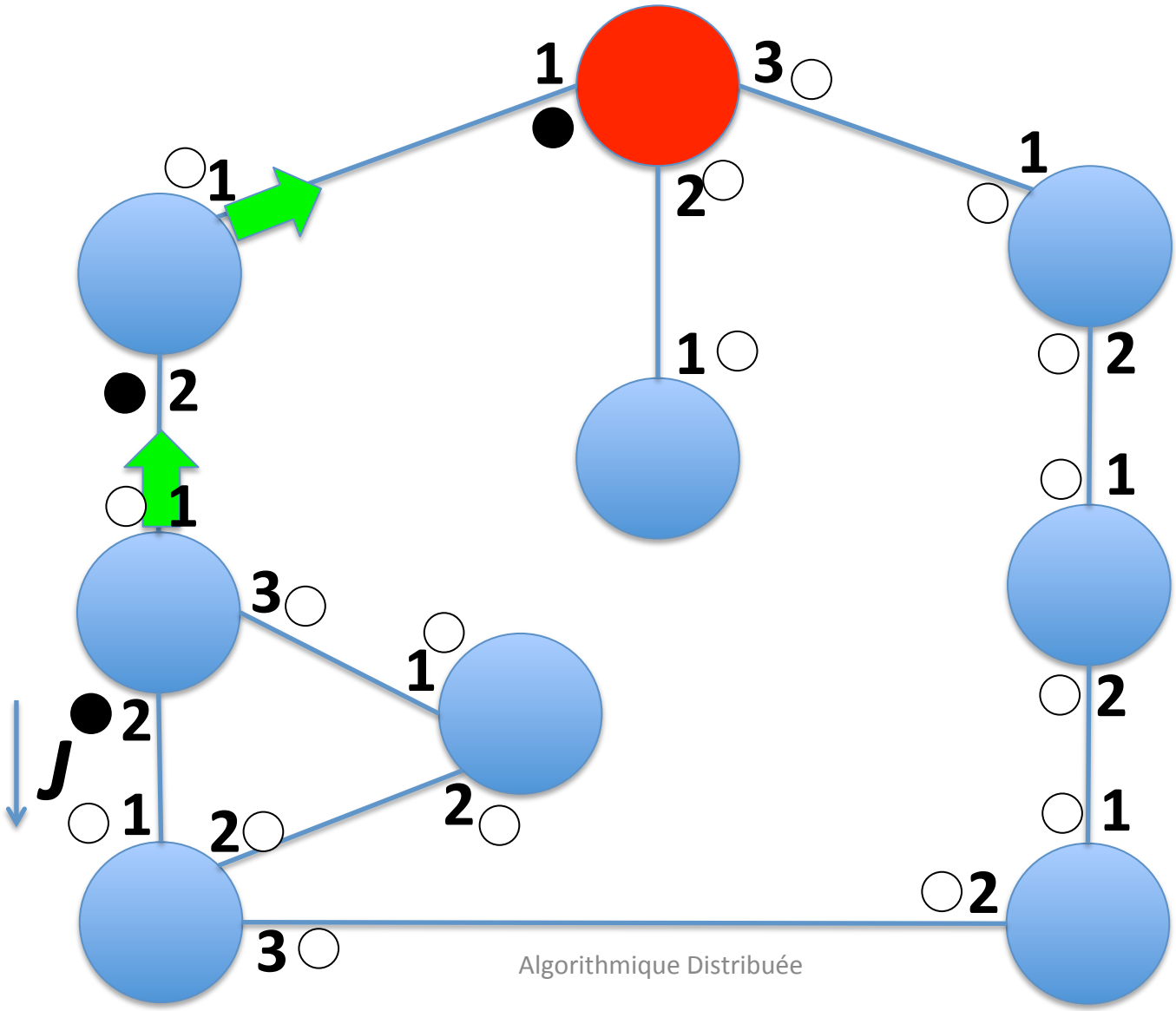
Exemple



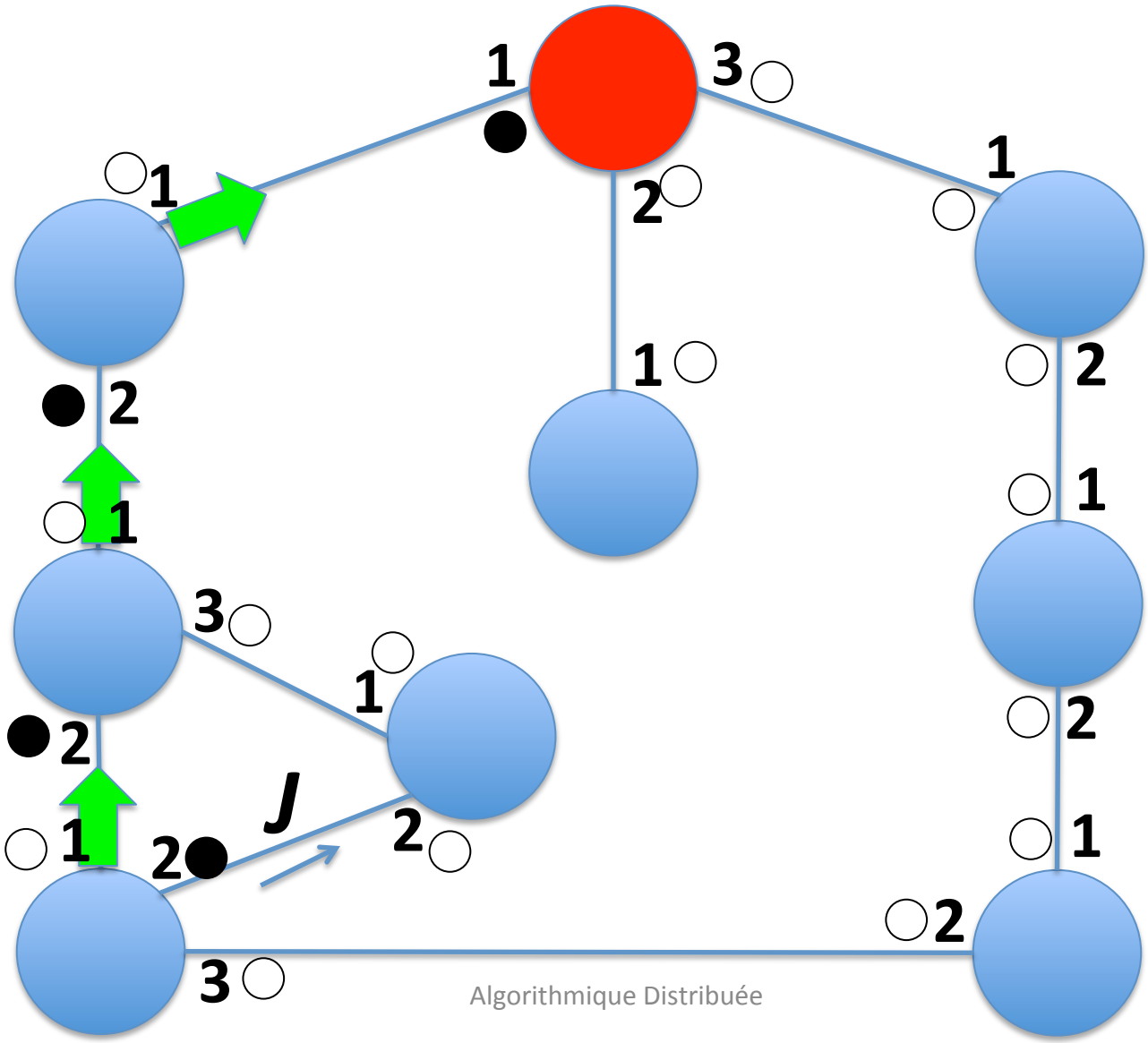
Exemple



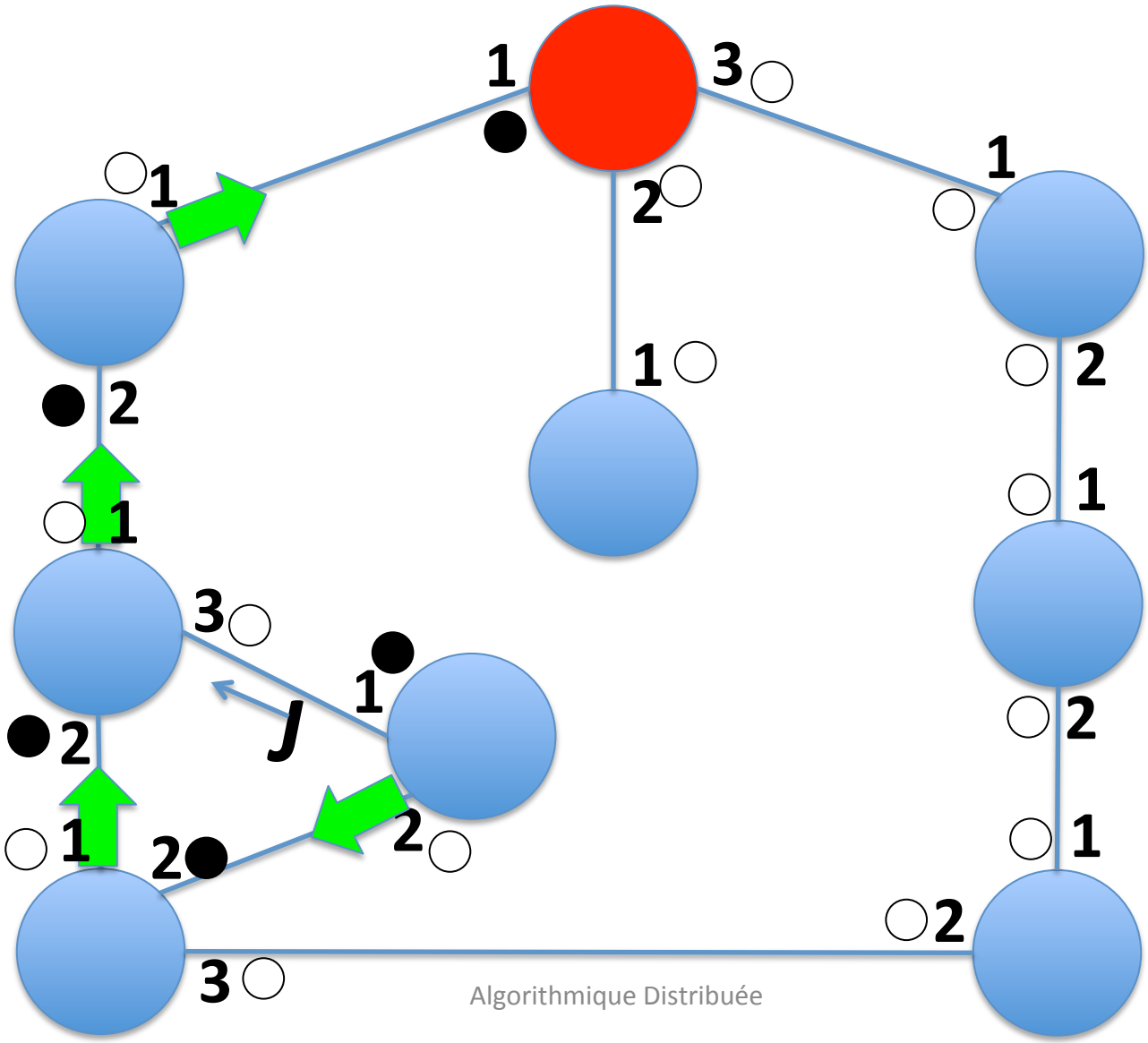
Exemple



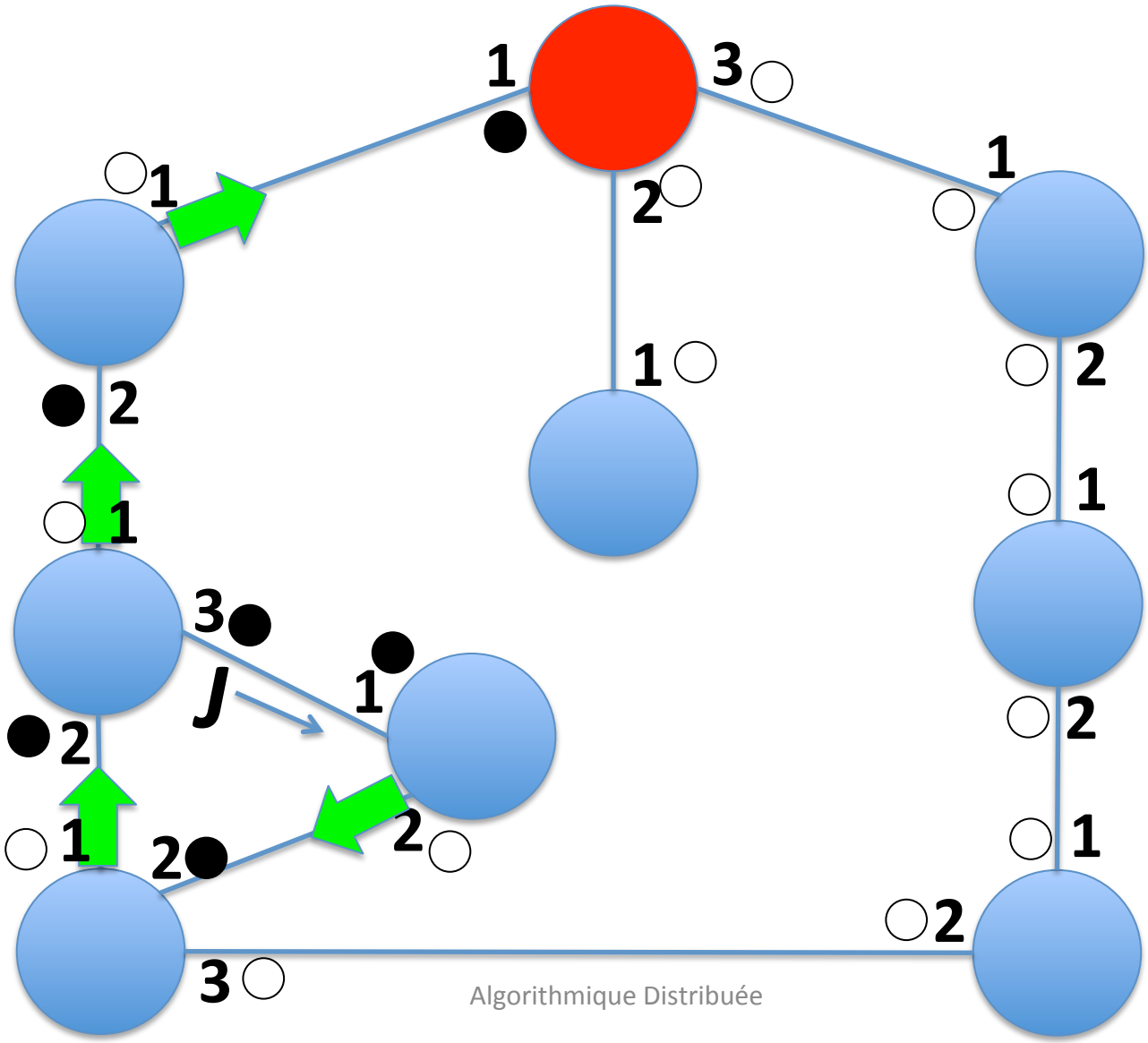
Exemple



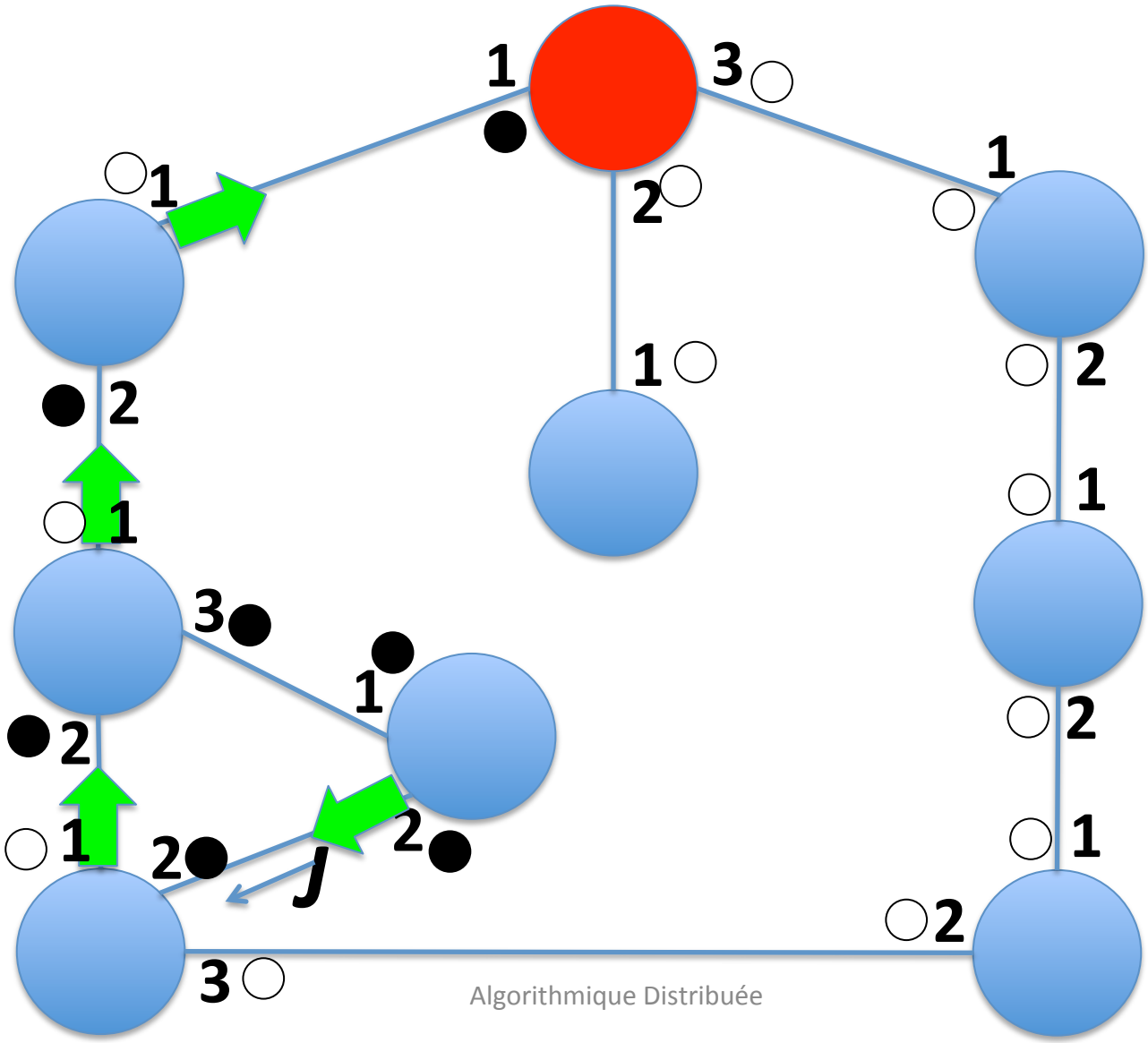
Exemple



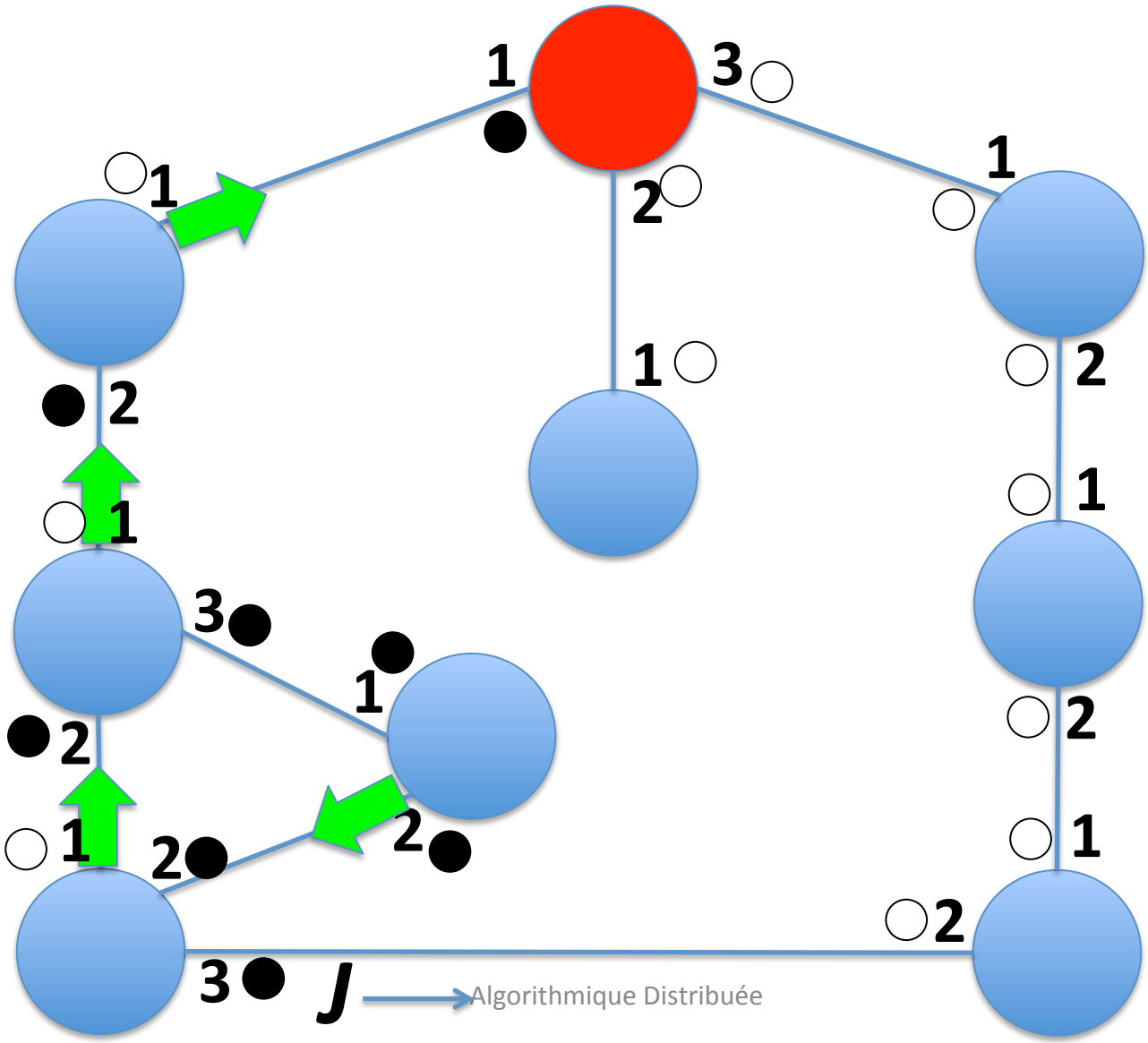
Exemple



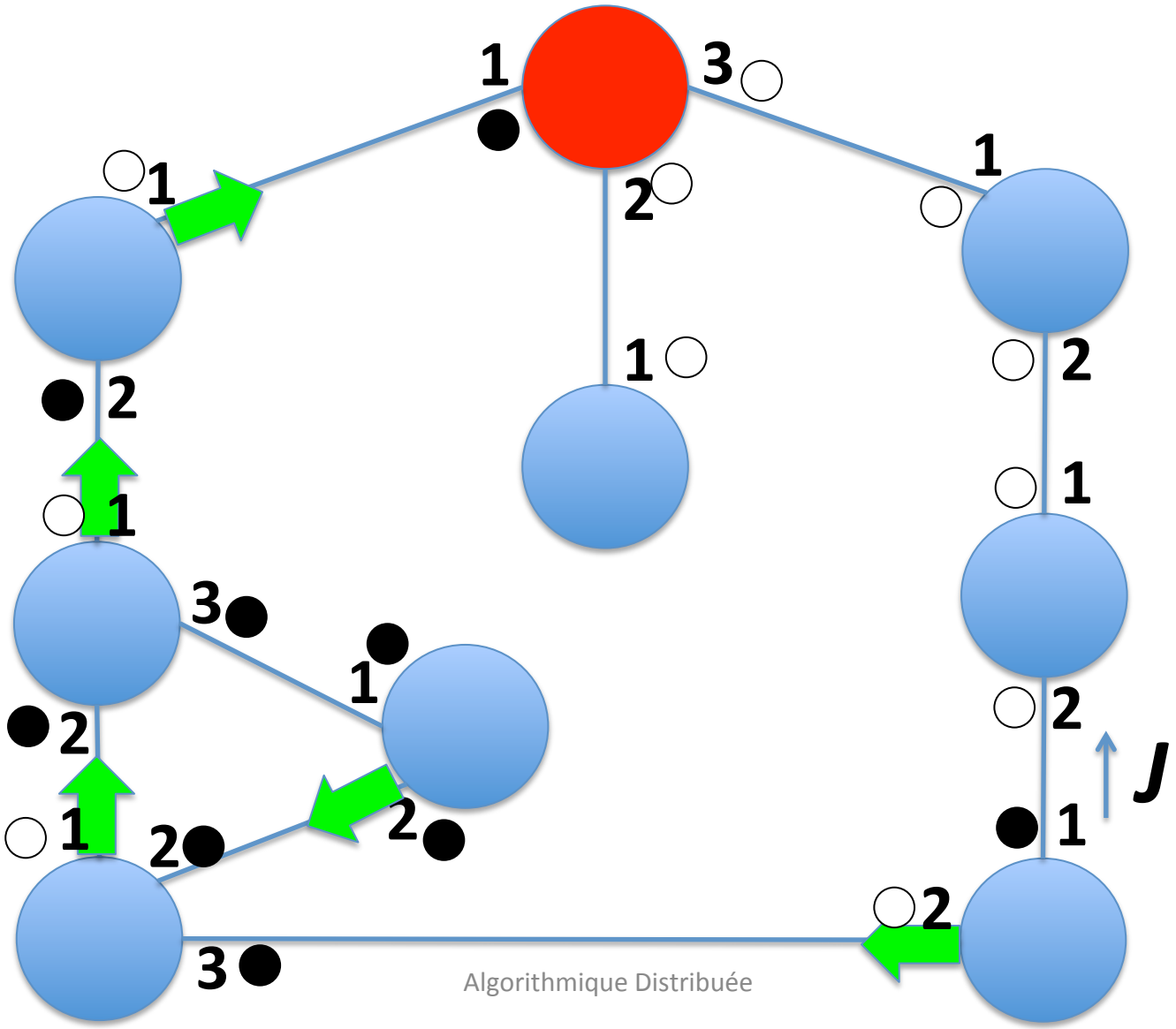
Exemple



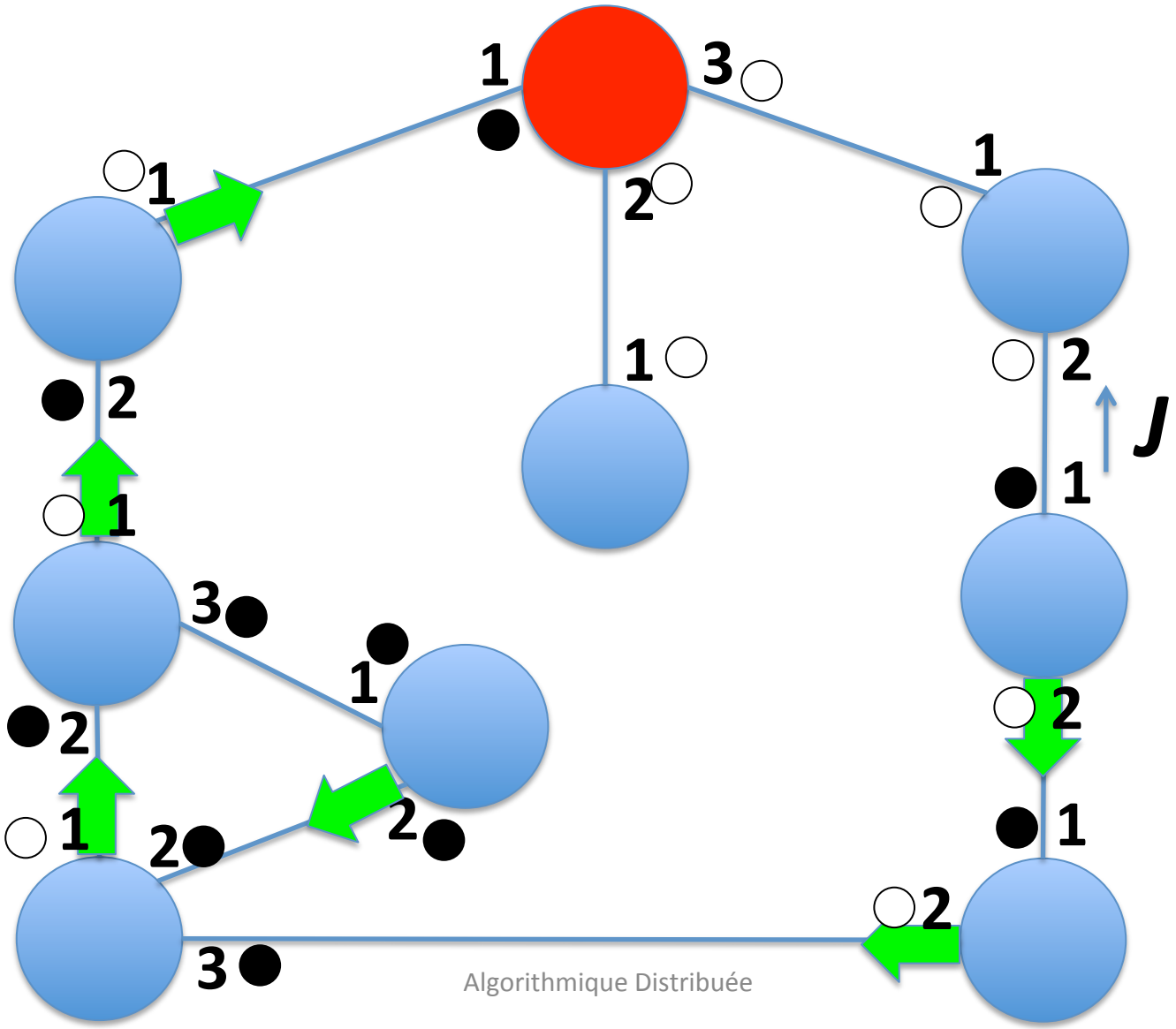
Exemple



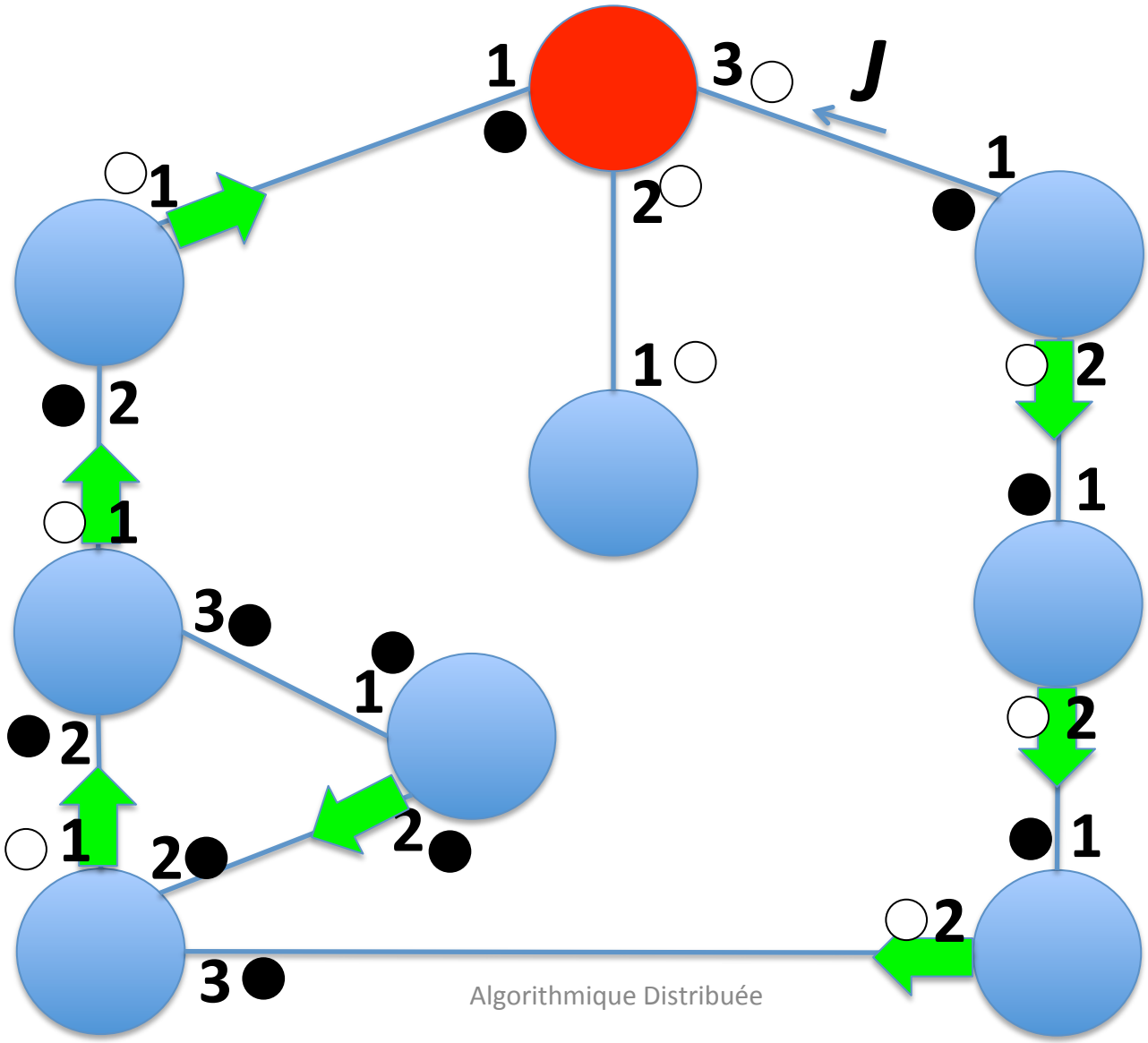
Exemple



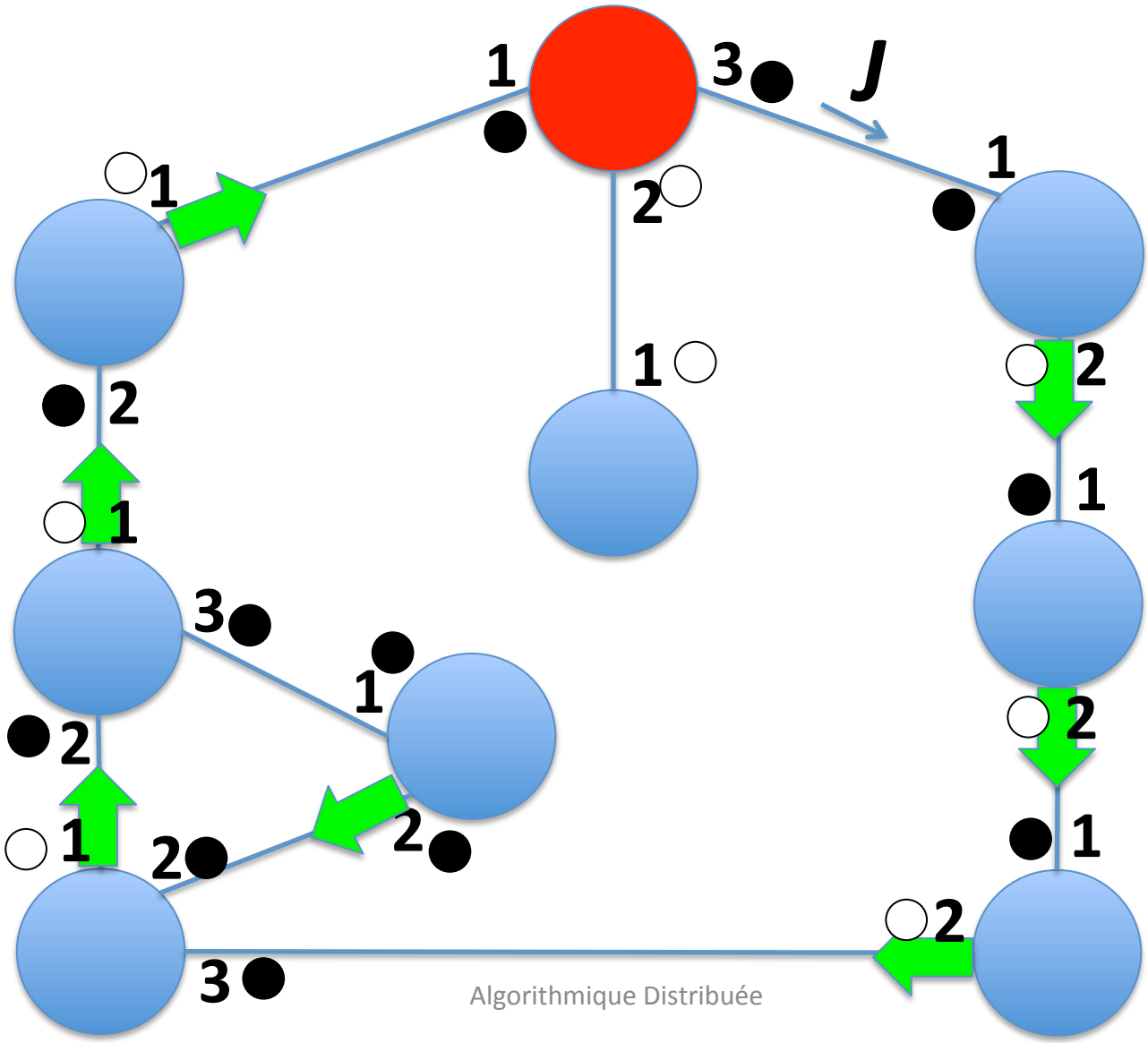
Exemple



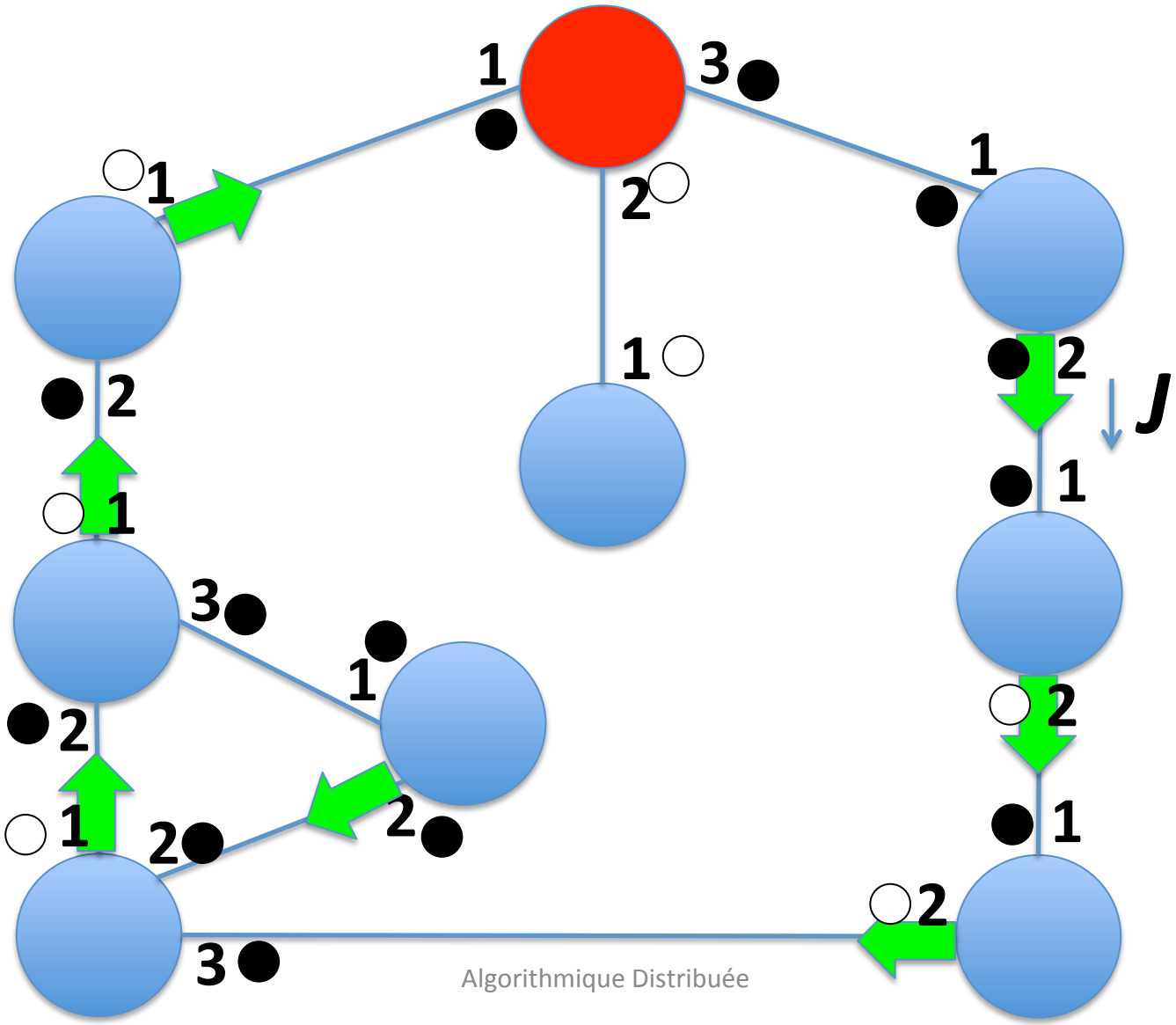
Exemple



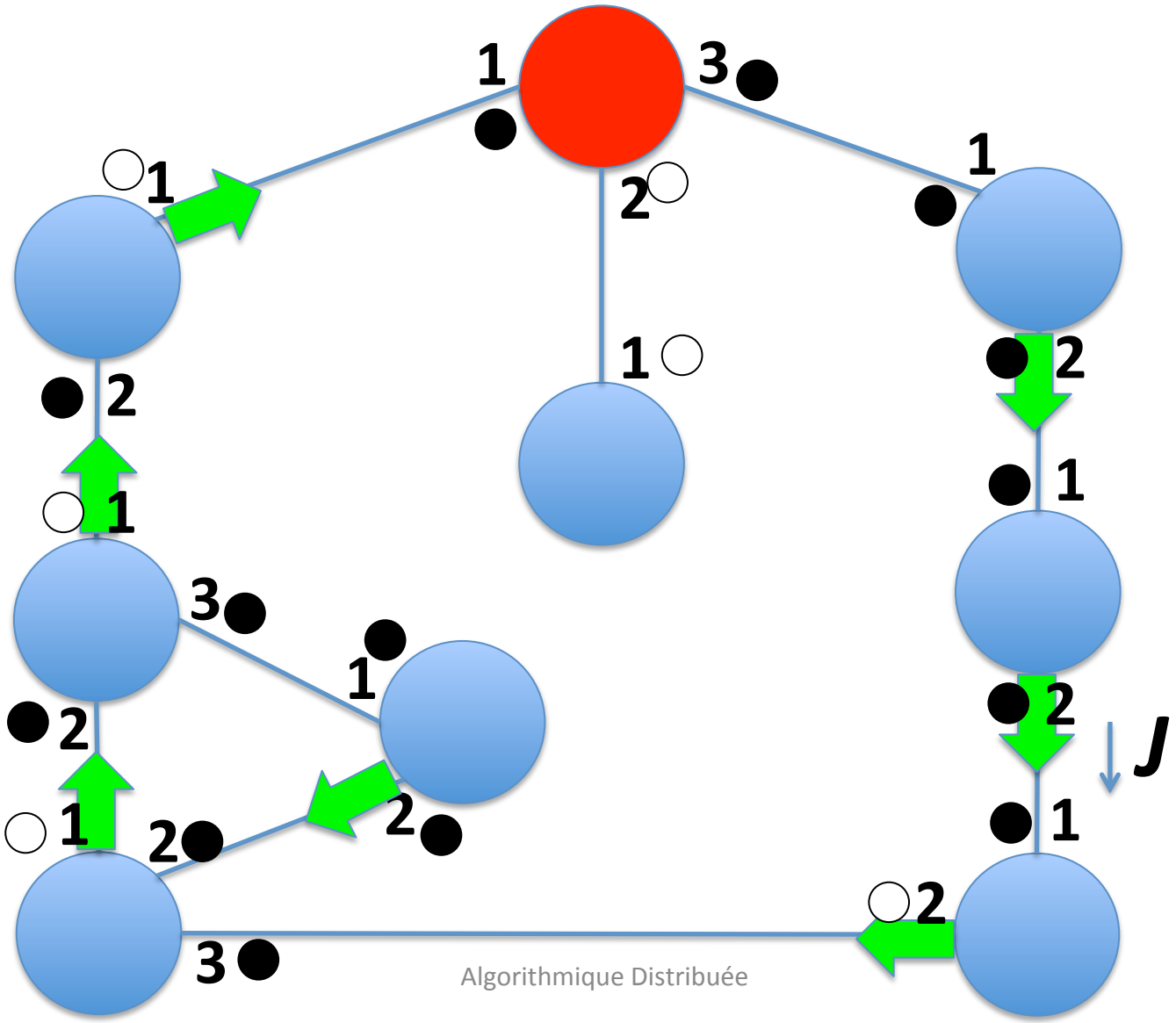
Exemple



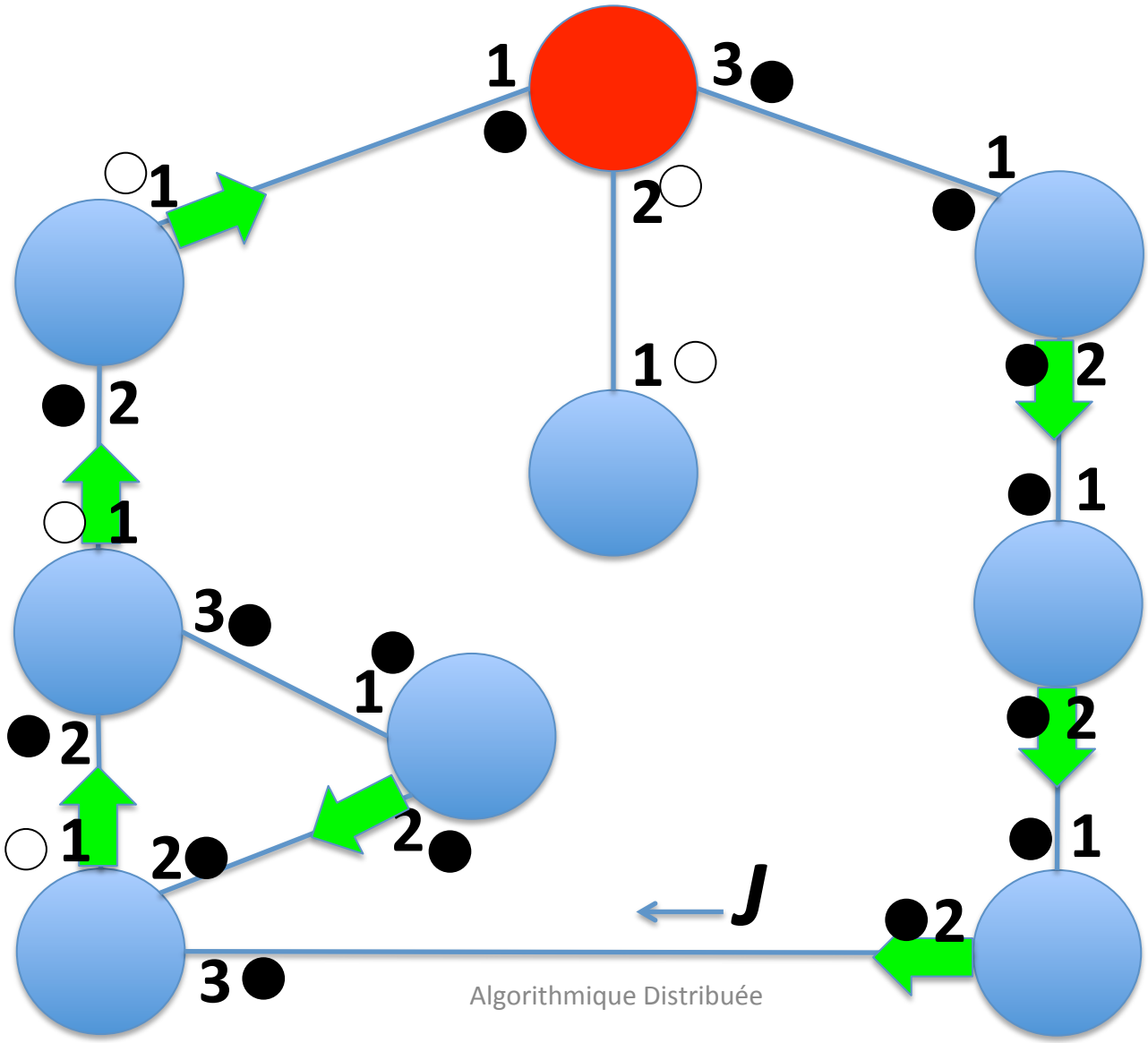
Exemple



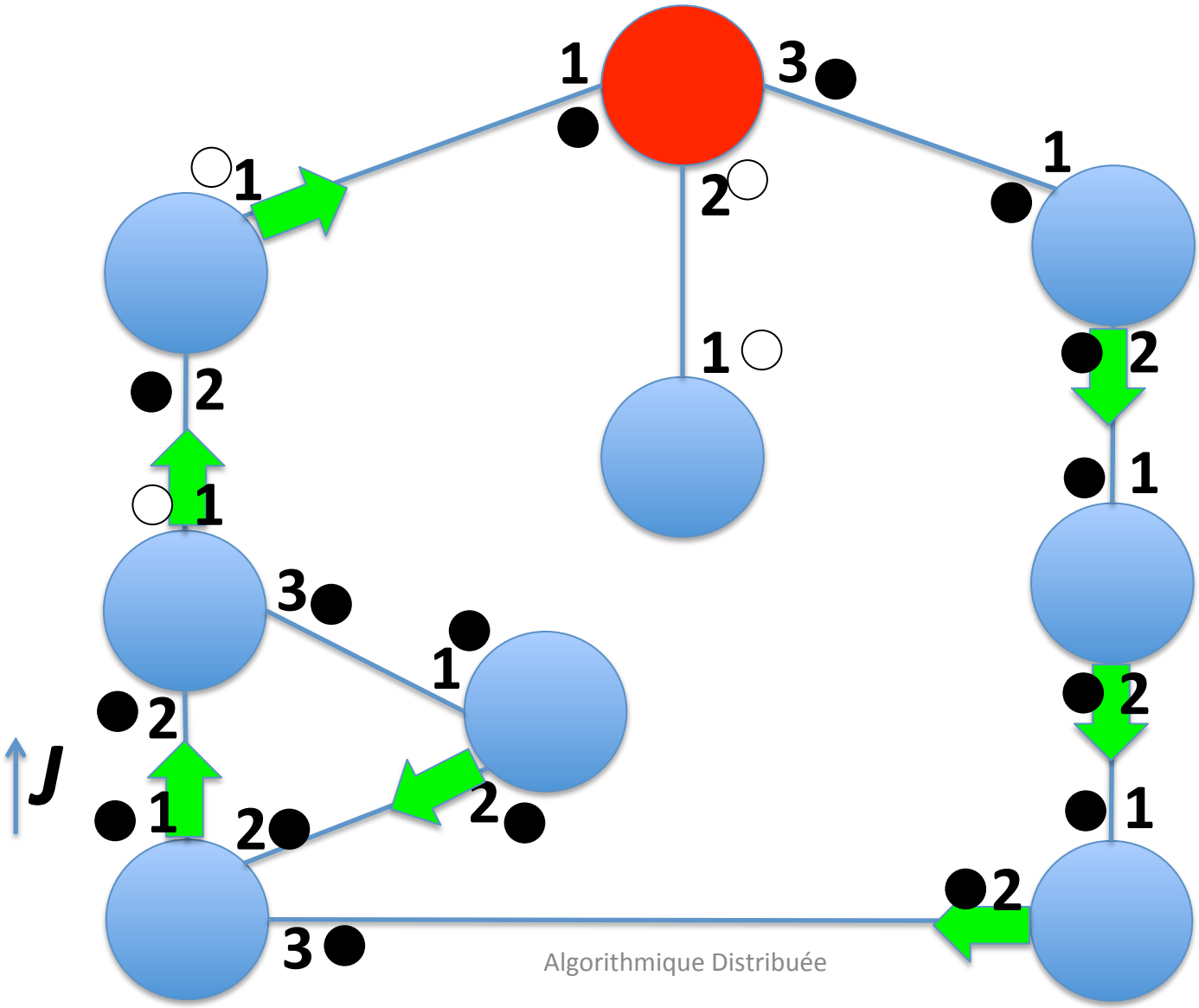
Exemple



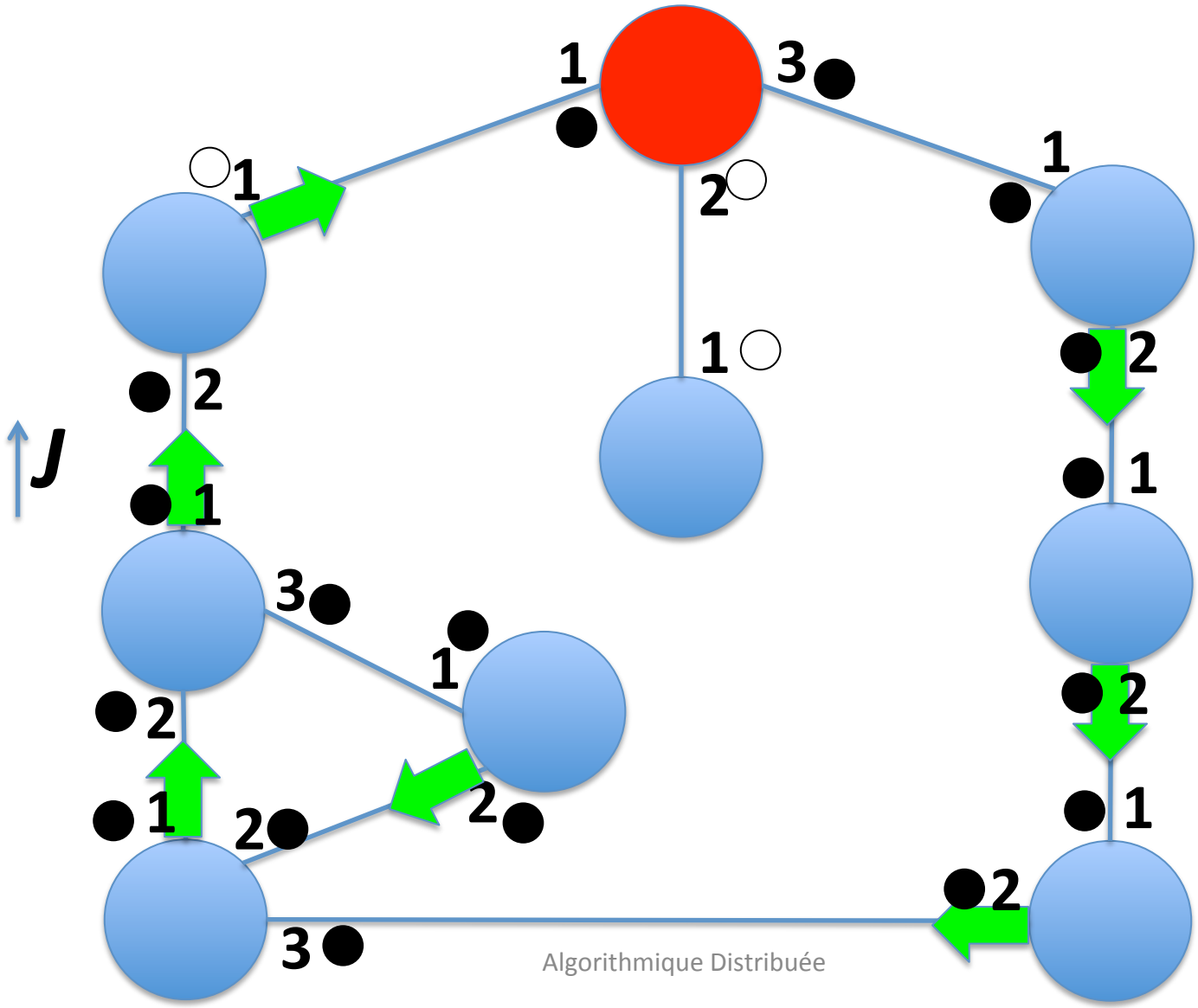
Exemple



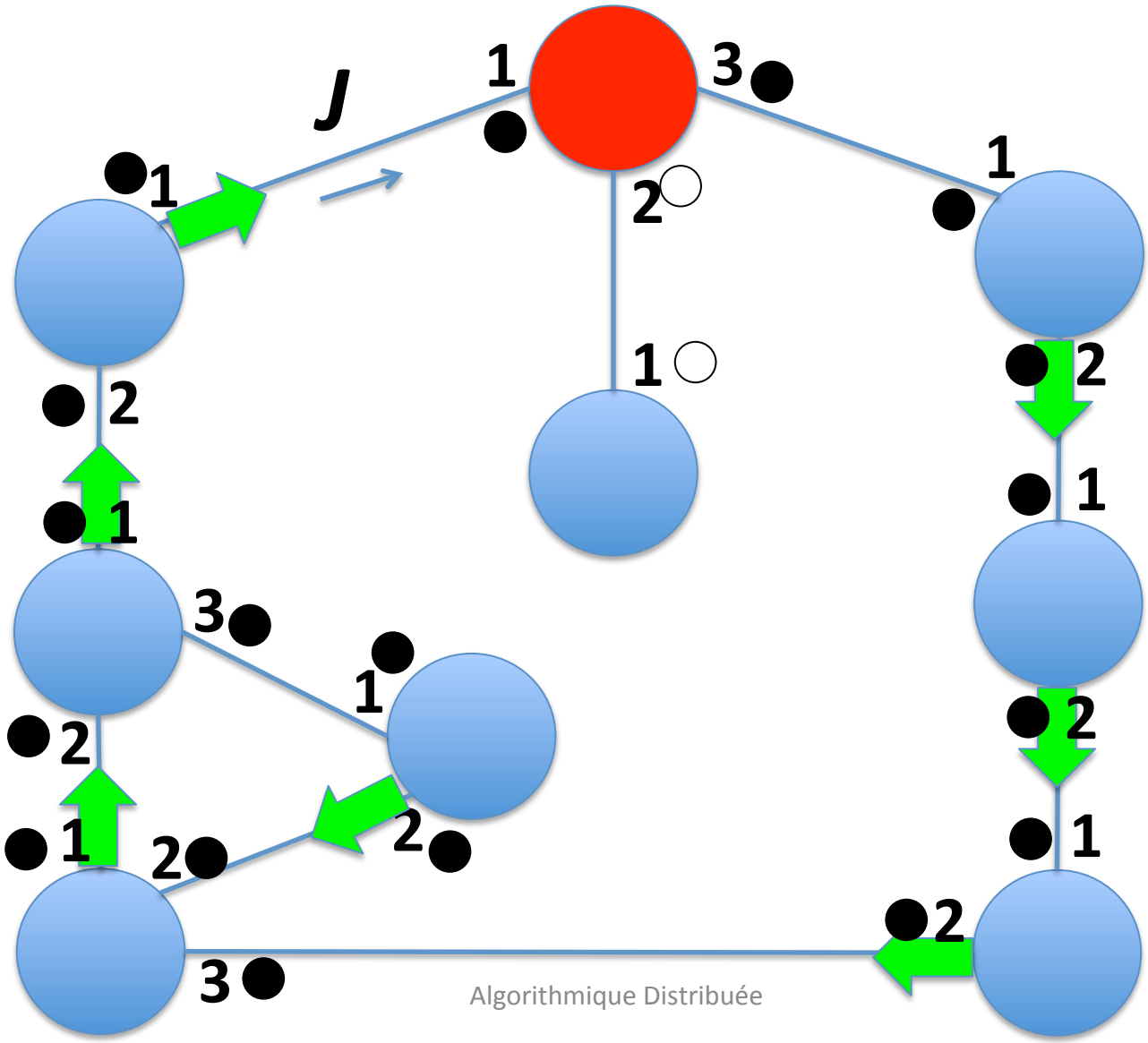
Exemple



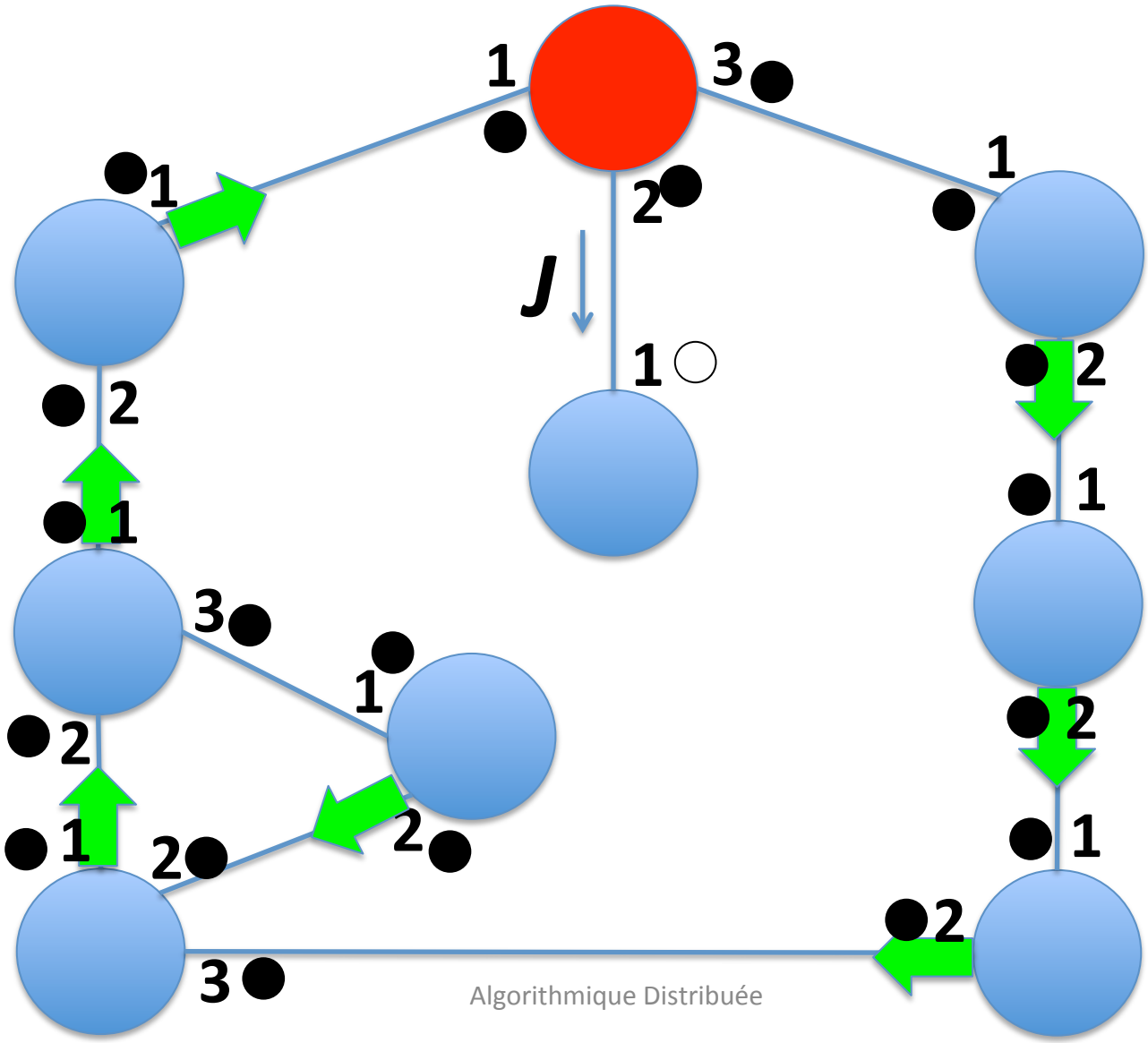
Exemple



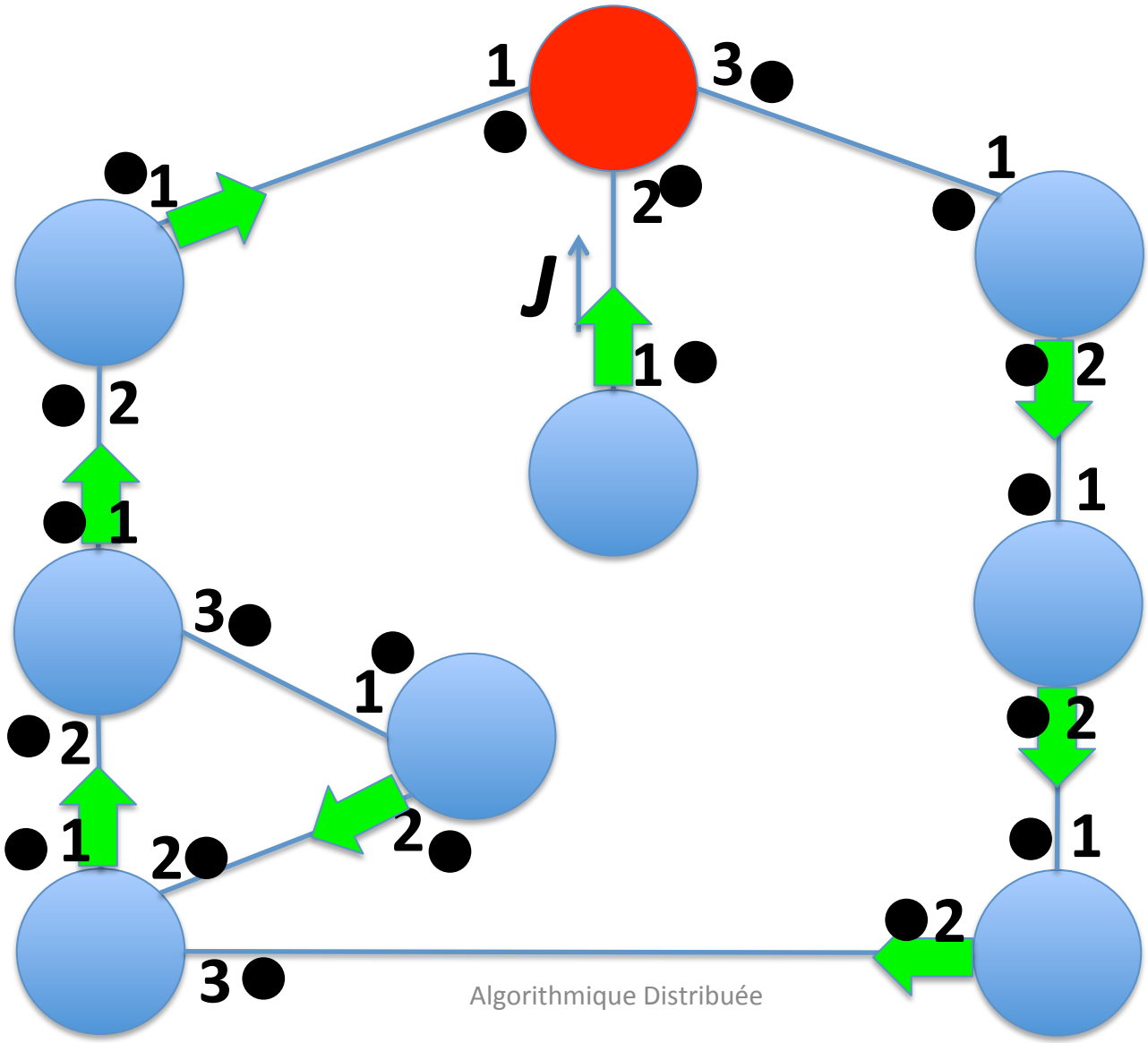
Exemple



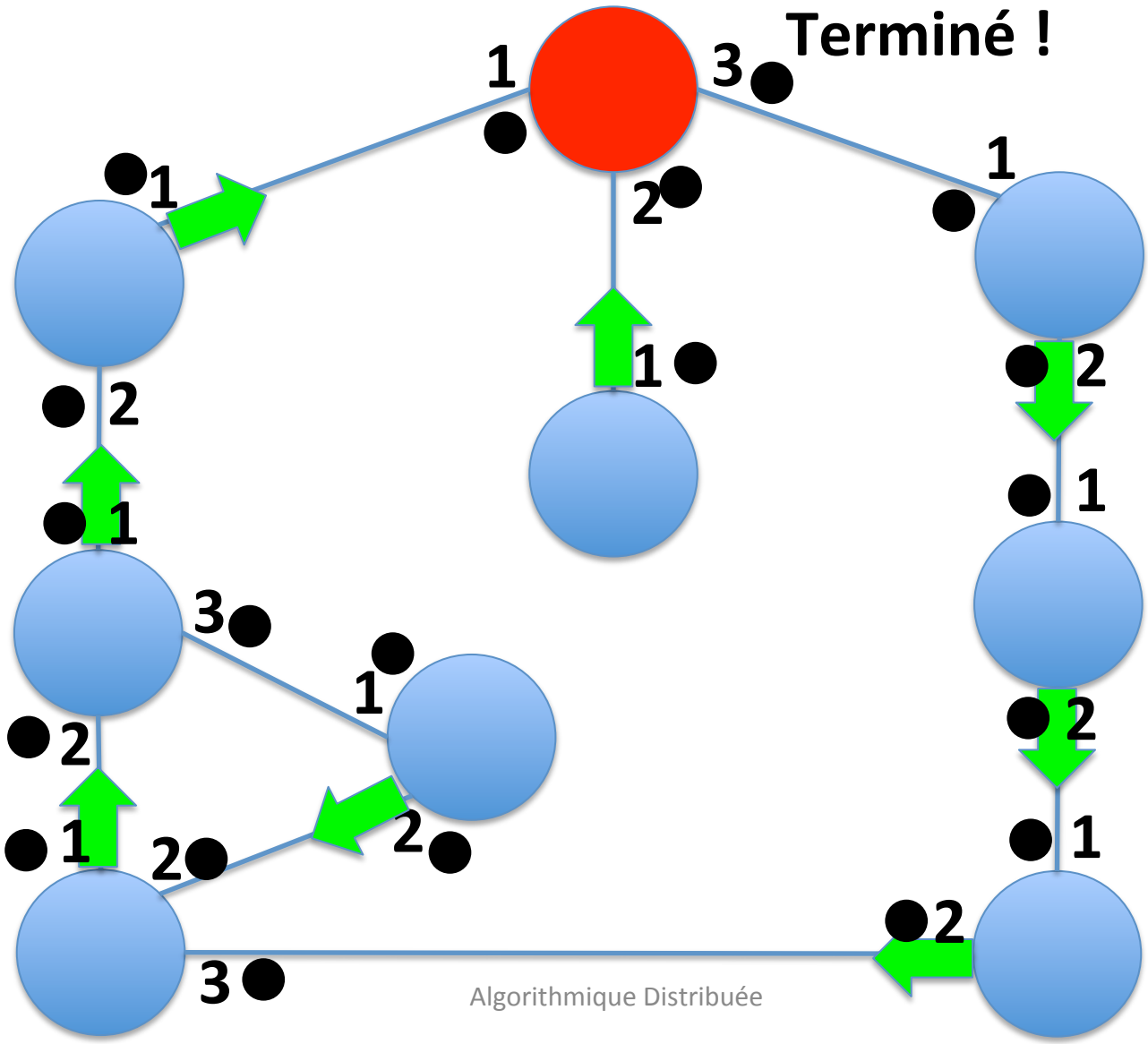
Exemple



Exemple



Exemple



Conclusion générale

- Depuis 40 ans
  - La plupart des problèmes d'algorithmiques réparties ont été résolus de manière **efficace**
  - En supposant des réseaux sans pannes …

Challenge actuel

- Les réseaux modernes sont **à grande-échelle** et fait de machines **hétérogènes** et produites en masses à **faible coût**, *e.g.*
  - Internet
    - (10 milliard de machine connectée d'ici 2016)
    - Internet des objets
  - Réseaux sans fils
    - Communication radio : beaucoup de pertes de messages
    - Crash de machines à cause des batteries limitées

⇒ Forte probabilité de pannes

⇒ Intervention humain impossible

⇒ **Besoin d'algorithmes distribués tolérant les pannes**

Syntaxe : réception bloquante

Algorithme 3 Algorithme avec réception bloquante pour tout process p

1: Instructions d'initialisation
2: **Tant que** *vrai* **faire**
3: Instructions
4: **attendre de recevoir** X **messages de la forme** $\langle TypeMess,\ liste\ variables\ ...\rangle$ **depuis tous les processus de** S
5: Instructions
6: **Fin Tant que**

Syntaxe : réception bloquante

Algorithme 4 Algorithme avec réception bloquante pour tout process p

```
1:  Instructions d'initialisation
2:  Tant que vrai faire
3:      Instructions
4:      ListeMess ← ∅
5:      i ← 0
6:      Tant que i < X faire
7:          Pour tout q ∈ S faire
8:              Si réception M = ⟨TypeMess, liste variables ...⟩ depuis q alors
9:                  i++
10:                 ListeMess ← ListeMess ∪ {M}
11:             Fin Si
12:         Fin Pour
13:     Fin Tant que
14:     Instructions
15: Fin Tant que
```


Merci de votre attention !