

Systemes Distribués

Alain Cournier & Stéphane Devismes

Info pratiques

- Contact
 - Alain.Cournier@u-picardie.fr
 - Stephane.Devismes@u-picardie.fr
- Support de cours sur Moodle :
 - <https://pedag.u-picardie.fr/moodle/upjv/course/view.php?id=8713>
- Emploi du temps : sur Celcat ?

But

- **Etude des algorithmes distribués :**
 - Spécification
 - Conception
 - Preuve de correction et complexité
 - Implantation
 - Dans le simulateur *Sinalgo*
 - Source (*plug-in Java*), Tutorial, supports et machine virtuelle **disponible sur moodle**
- **Problèmes considérés :** briques de bases pour les réseaux
 - élection, circulation de jeton, diffusion, *etc.*
- **Intérêt et inconvénient :** Indépendant de l'architecture matérielle

Contenu (prévisionnel)

- 30h, 8 séances ($7 \times 4h + 1 \times 2h$), alternance Cours-TD / TP
 1. CM (rappel) + début TD DFS
 2. Fin TD DFS + Présentation Simulateur + début TP DFS
 3. Cours-TD PIR + fin TP DFS + TP PIR
 4. Horloges de Lamport
 5. Tables de routage
 6. Prise d'instantané et détection de terminaison
 7. Impossibilité du consensus en présence de pannes franches
 8. Examen

Evaluation

- Contrôle continu : 2 notes (à définir)
- Note finale : moyenne des 2 notes

Introduction

Systemes distribués

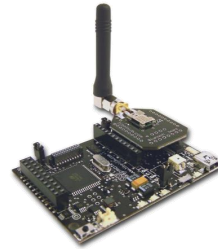
- « En informatique, un **systeme distribué** (aussi appelé système réparti) est un ensemble d'unités de traitement autonomes interconnectées entre-elles. »
 - Gérard Tel, *Introduction to distributed algorithms*

Systemes distribués

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

Systemes distribués

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



telephone-101.com

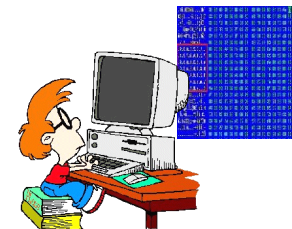
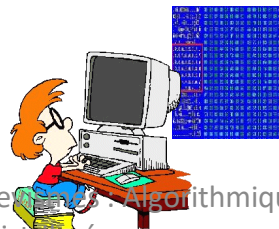
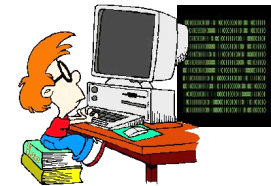


Systemes distribués

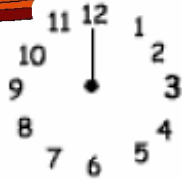
- **Autonome :**

- Chaque machine est pourvue de son propre contrôle

- Programmes locaux
- Mémoires locales



Systemes distribués



- **Autonome :**

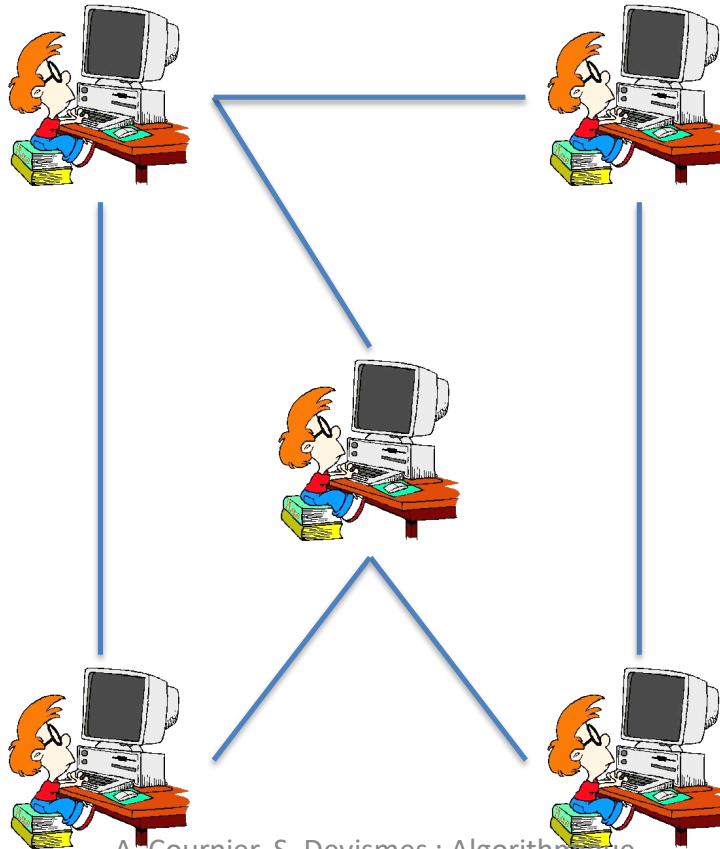
- ≠ algorithmes parallèles :
pas de synchronisation
(logicielle ou matérielle)

- Asynchrone

- Pas de temps global

Systemes distribués

- **Interconnectées** : échanges d'informations



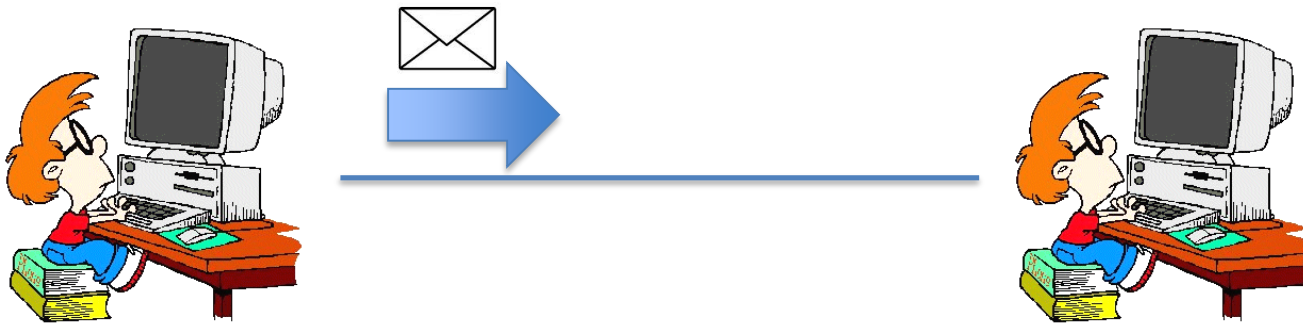
Systemes distribués

- **Interconnectées**



Systemes distribués

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

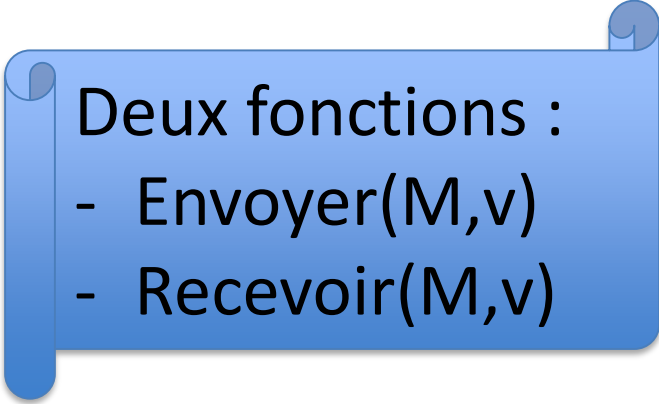


Systemes distribués

- **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

Algorithmes Distribués

Algorithmes pour des *systemes distribués* 😊

Systemes Centralisés *vs.* Systemes Distribués

Centralisé vs. Distribué

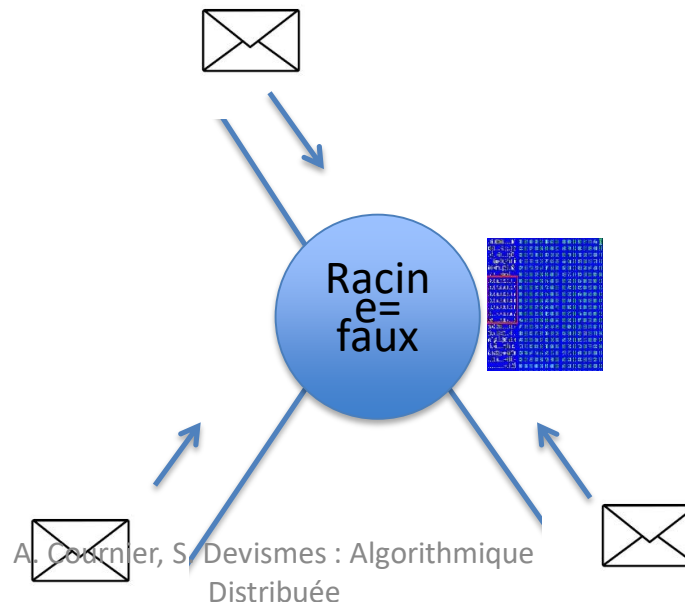
- **Absence de connaissance globale**

- 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

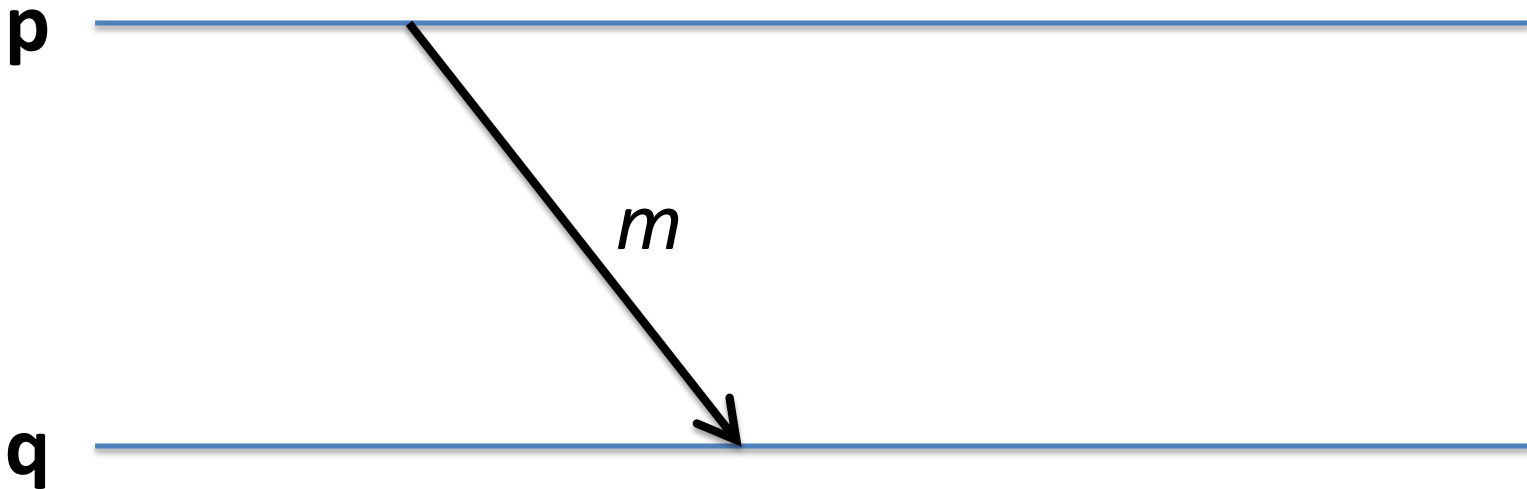
- Problème de pérennité des informations (système asynchrone)



Centralisé vs. Distribué

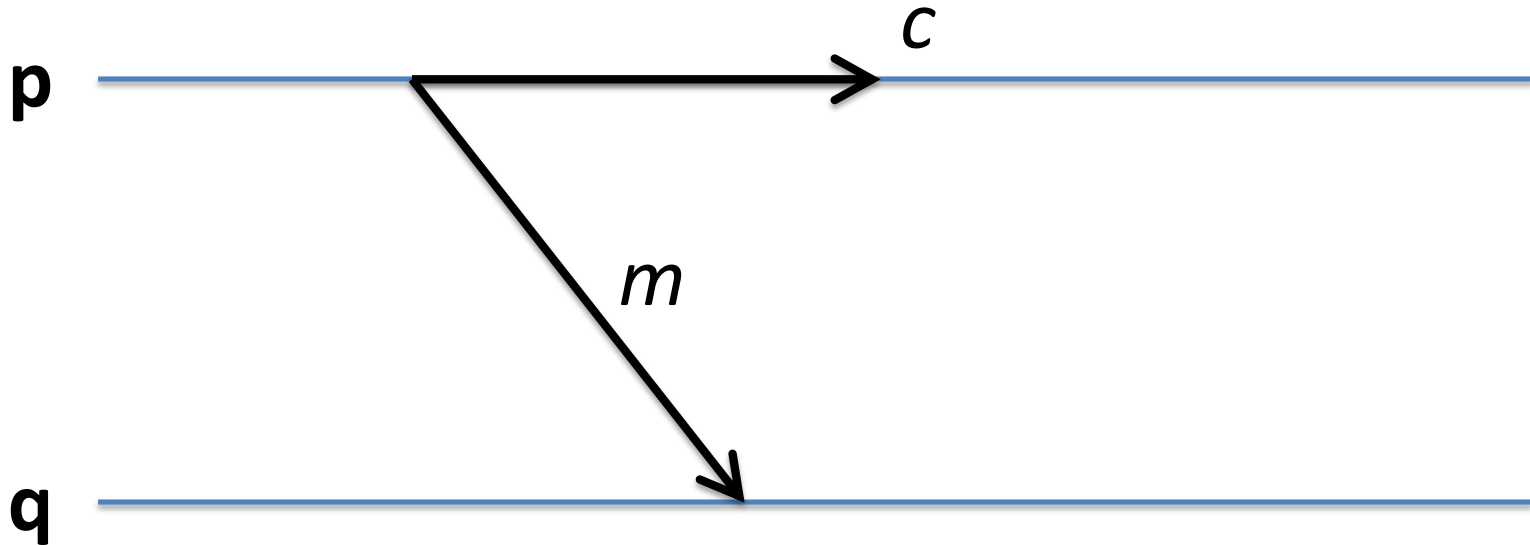
- **Absence de temps global :**
 - Temps d'exécution des nœuds \neq
 - Communications asynchrones
 - Horloges locales \neq (valeurs initiales \neq + dérive)
 - Conséquence : contrairement aux systèmes centralisés, l'ordre (observable par les processus) entre les actions des processus n'est que partiel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

Causalité : exemple



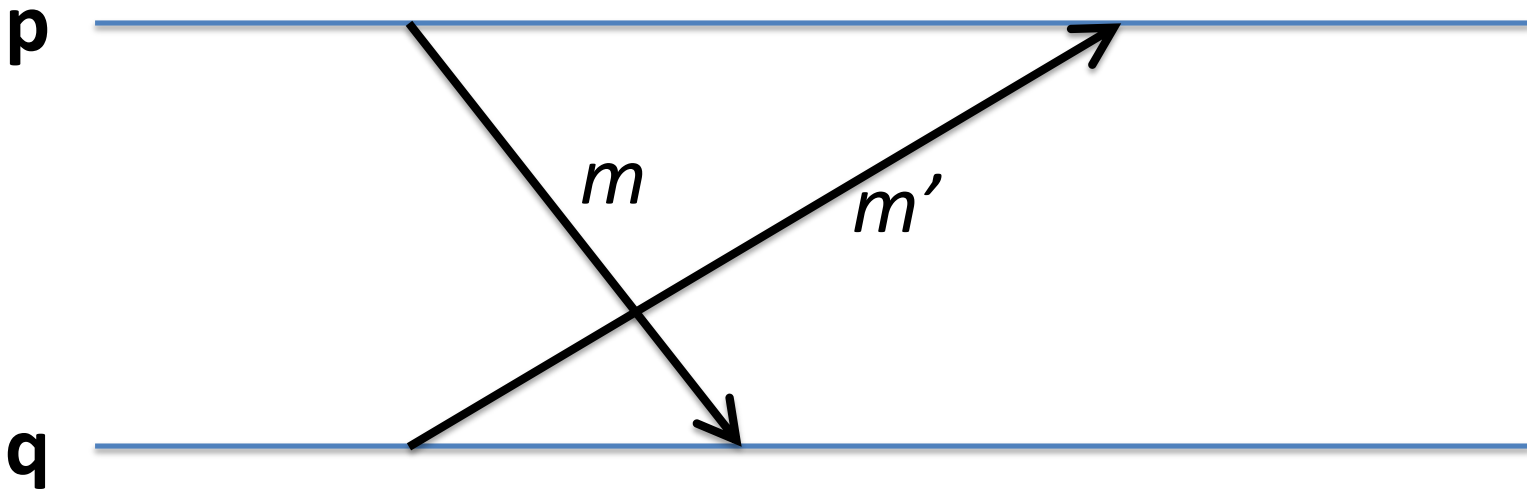
- L'événement « **p** envoie *m* » arrive *avant* l'événement « **q** reçoit *m* »

Causalité : exemple



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

Causalité : exemple



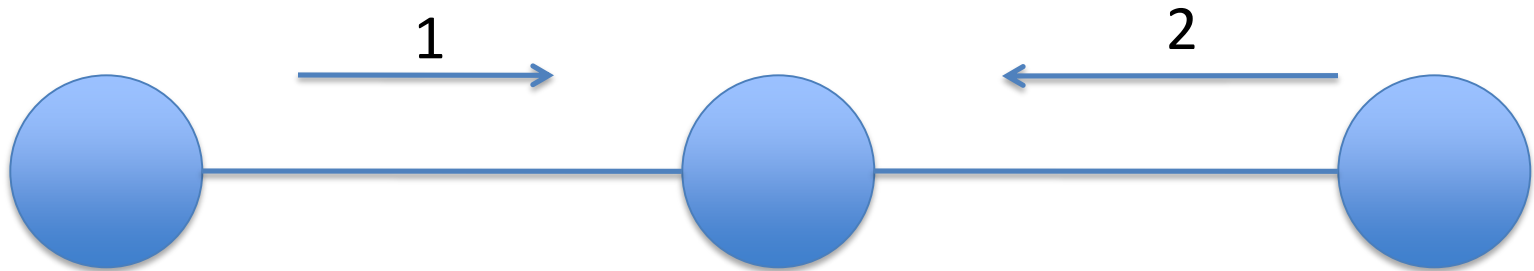
- Les événements « **p** envoie *m* » et « **q** envoie *m'* » sont *indépendants*

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

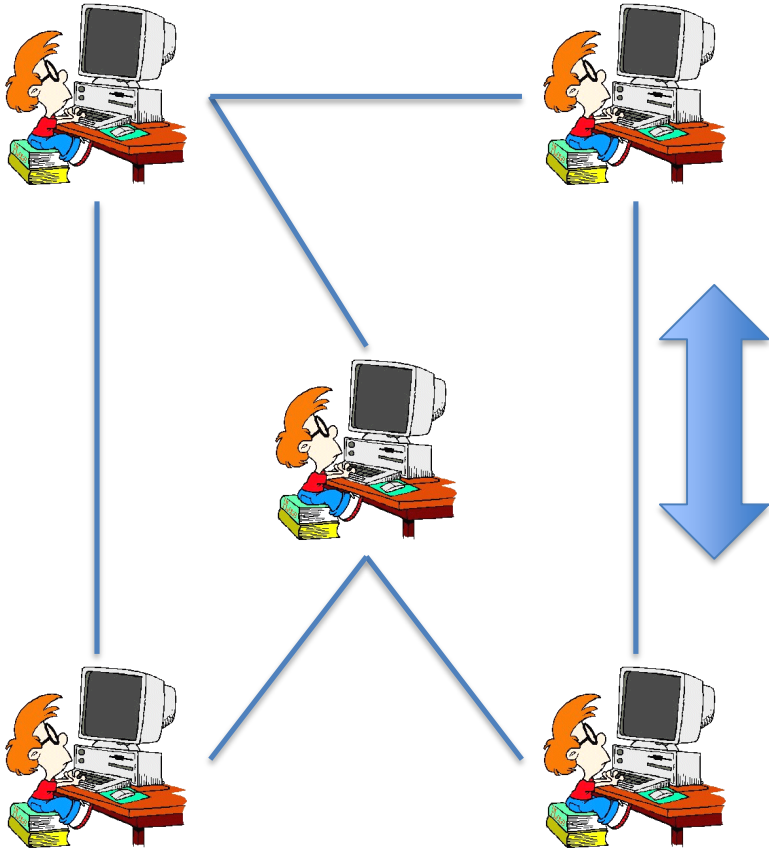
- Sa topologie
- Ses liens de communications
- Ses processus
- Le temps
- Les connaissances initiales des processus sur le système

Caractéristiques d'un système distribué : sa topologie

Caractéristiques

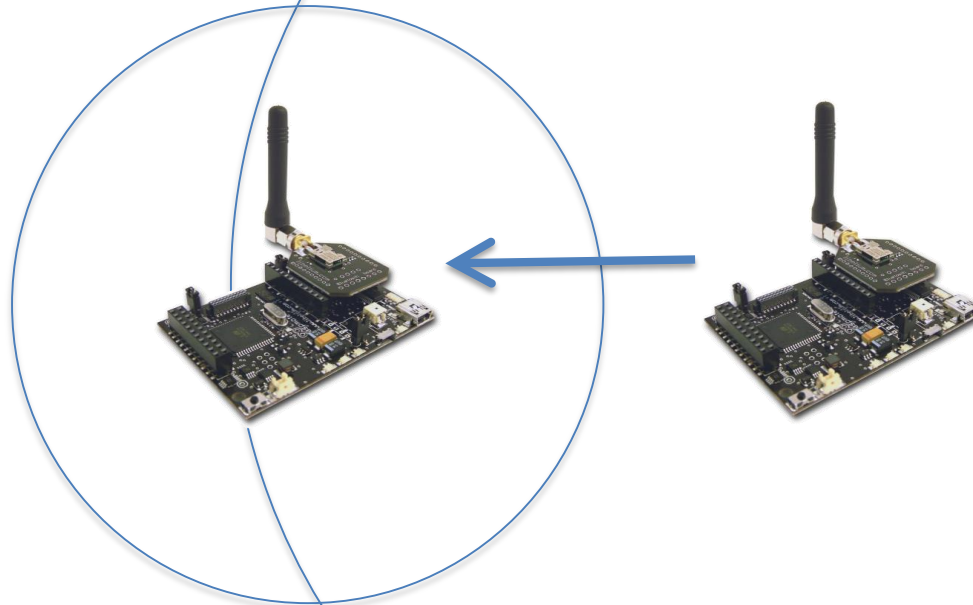
- **Topologie = réseaux d'interconnexion**
 - Communications :
 - bidirectionnelles (*ex.*, réseaux filaires) ou
 - unidirectionnelles (*ex.*, réseaux à fibres optiques)
 - Généralement on supposera des communications **bidirectionnelles** et une topologie **connexe**
 - **Topologie** \approx Graphe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

Les systèmes distribués

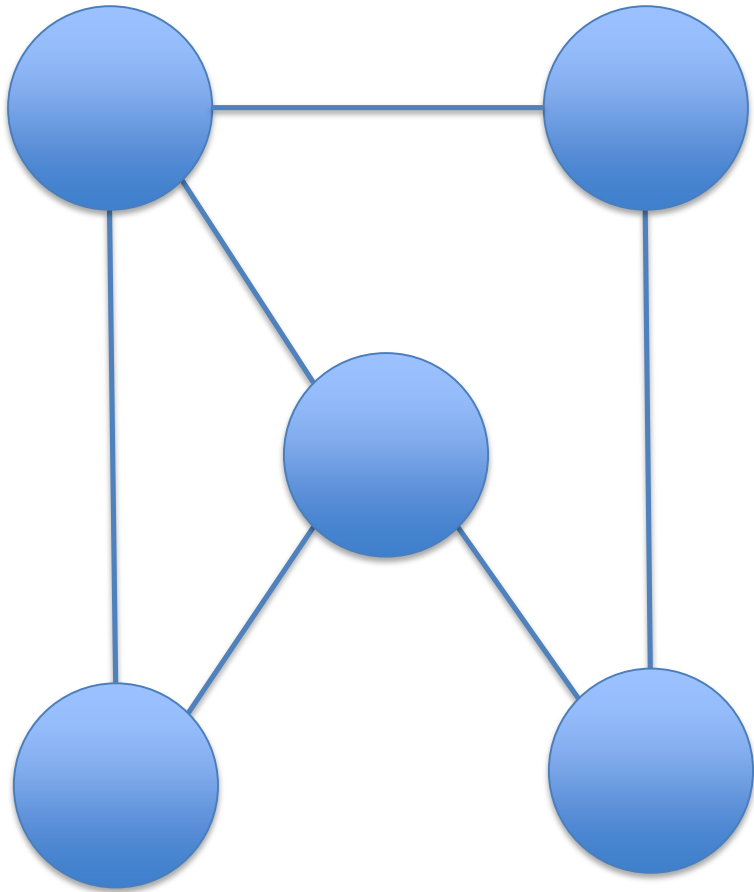


- Hypothèses
 - Liens bidirectionnels

Liens bidirectionnels : pas toujours !

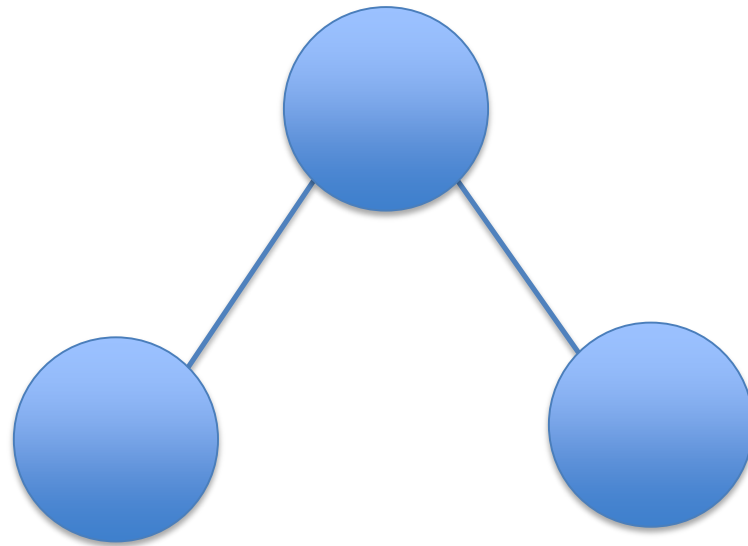


Rappel : Connexité



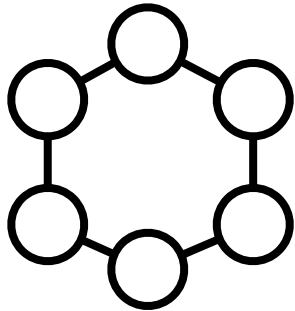
Connexe !

Rappel : Connexité

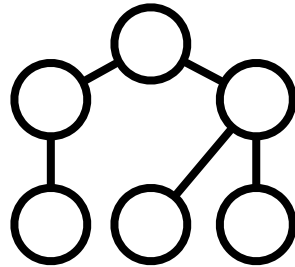


Pas connexe !

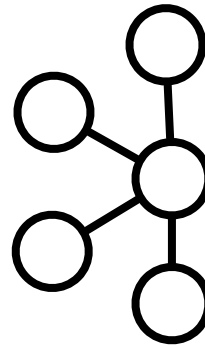
Exemples de topologies classiques



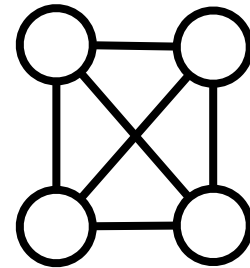
anneau



arbre

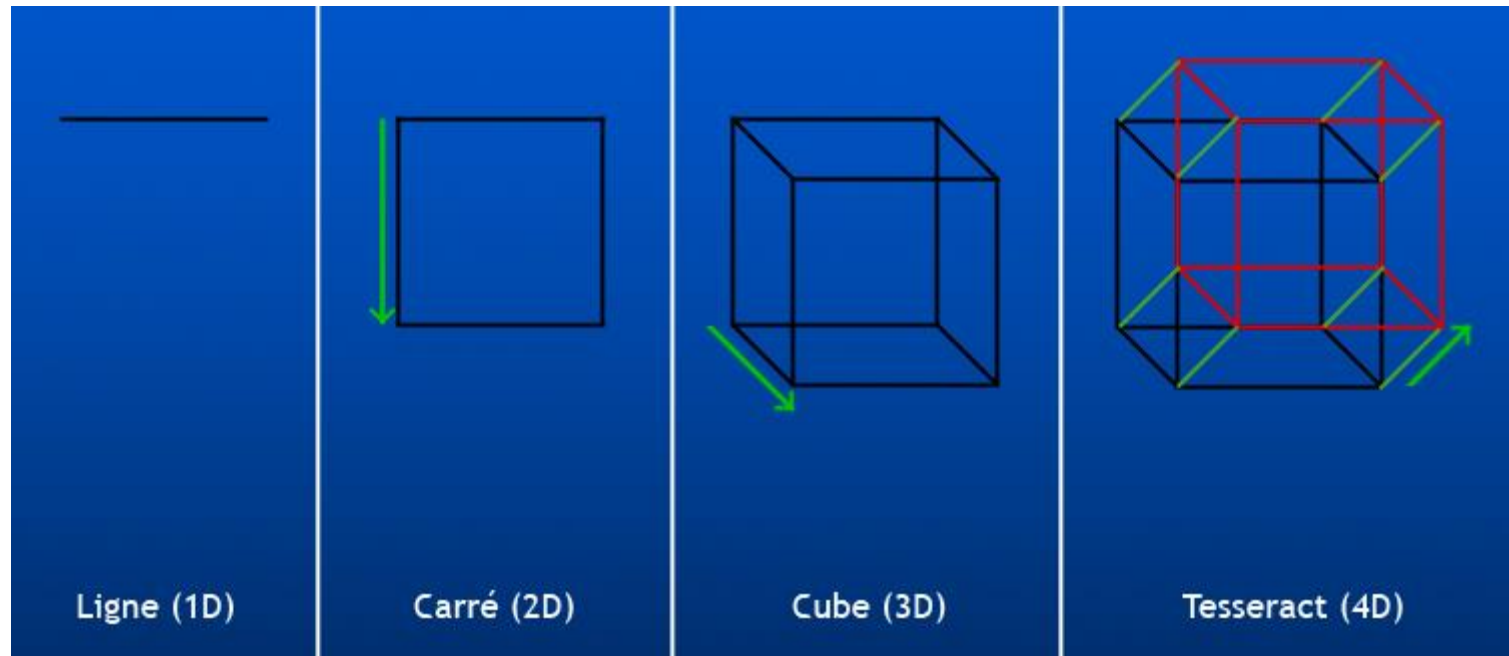


étoile



clique

Exemples de topologies classiques



Hypercubes (image source wikipédia)

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 quelconque)
 - Temps d'acheminement (toujours fini)
 - synchrone, **asynchrone**, ...
 - (pour le premier, borne connue ou pas)
 - Temps de traitement (supposé) négligeable
 - Capacité des canaux
 - bornée ou **pas**, borne connue ou pas

FIFO : Rappel



FIFO : Rappel



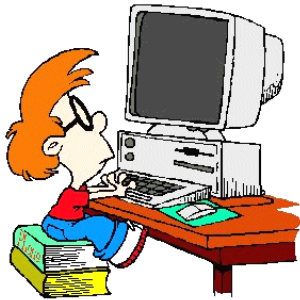
FIFO : Rappel



B A



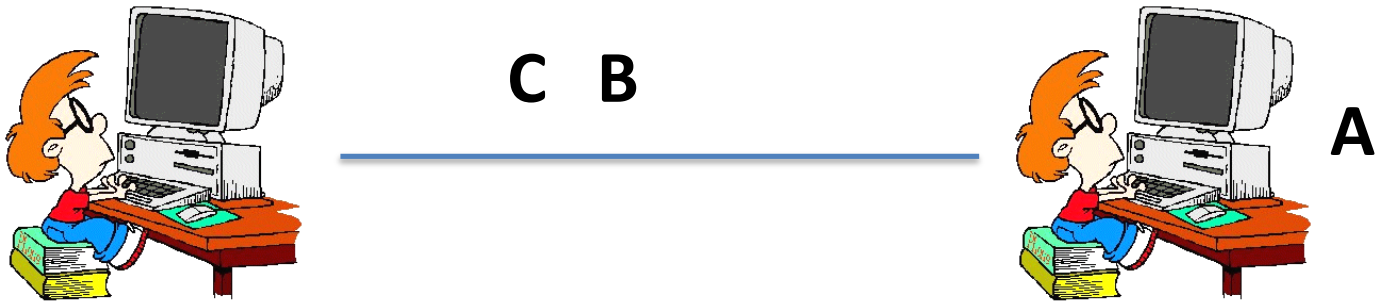
FIFO : Rappel



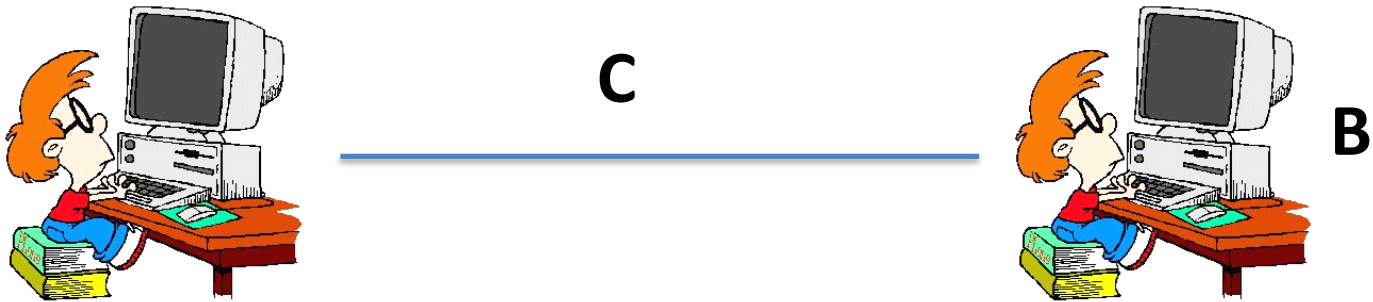
C B A



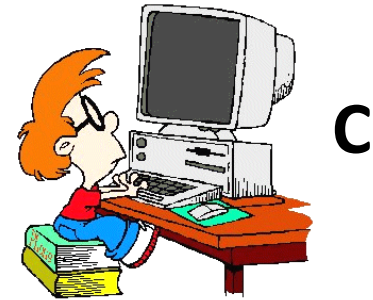
FIFO : Rappel



FIFO : Rappel



FIFO : Rappel



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 **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.
 - Un message peut mettre un temps arbitrairement long, mais fini, avant d'être reçu

Hypothèses

- Sauf avis contraire nous travaillerons :
 - Des liens bidirectionnels
 - Des canaux FIFO et asynchrones
 - Pas de perte de messages dans les canaux

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)

- Spontané : abstraction, appel depuis
 - la couche applicative (par ex., un utilisateur) ou
 - un autre algorithme (composition)

- Sujet ou **pas** aux pannes ? (et quel type ?)

- **Asynchrone** (équitable) ou synchrone

- Mémoire locale de taille bornée ou **non** ?

Caractéristiques d'un système distribué : le temps

Caractéristiques

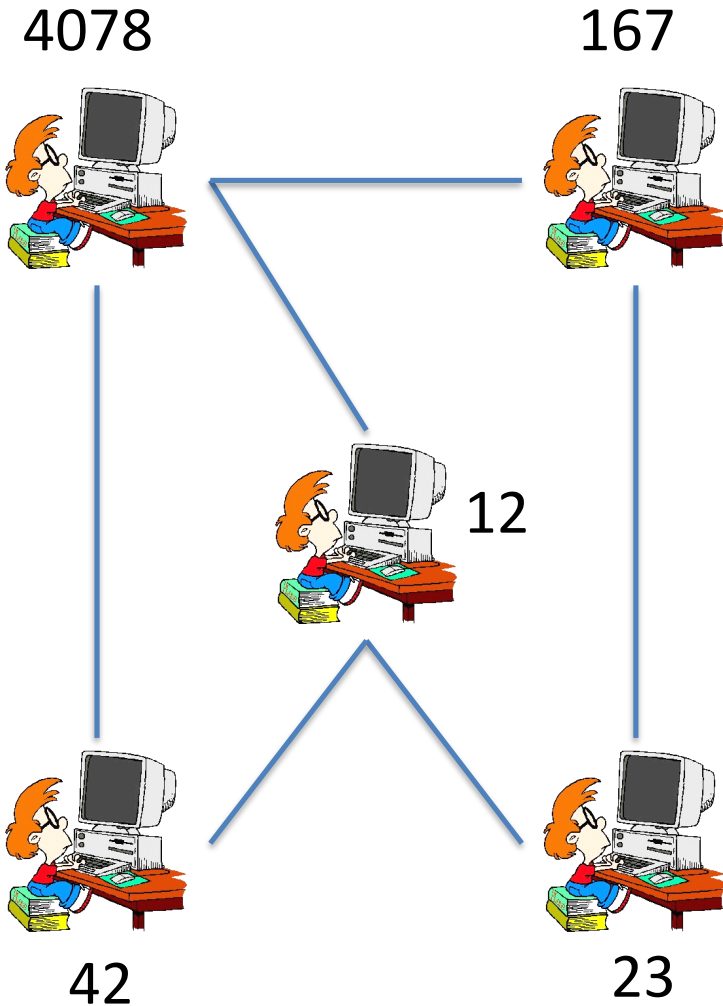
- **Temps (global)**
 - Discret : temps 0, 1, 2 ...
 - Non accessible aux processus mais utilisé dans les preuves
 - Cependant, on utilisera parfois du temps local (*minuteurs*)

Caractéristiques d'un système distribué : connaissances initiales des processus

Caractéristiques

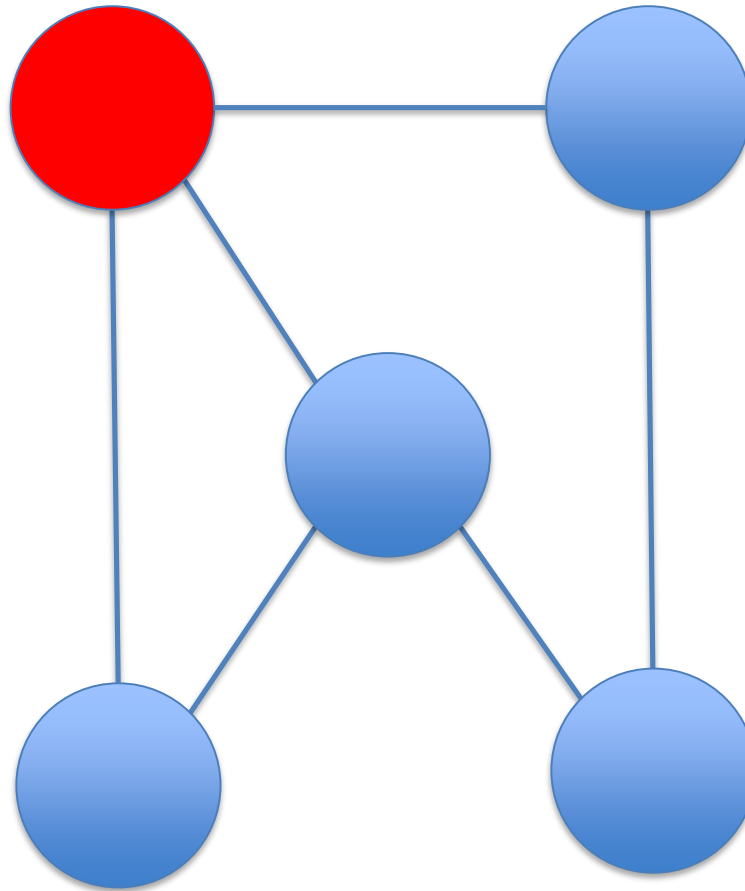
- **Connaissances initiales (entrées) des processus**
 - *Propriétés globales :*
 - Topologique :
 - Topologie, *e.g.*, je sais que je suis dans un arbre
 - Borne supérieure ou exacte sur
 - » Le nombre de processus (n)
 - » Le degré maximum (Δ)
 - » Le diamètre du réseau (D)
 - Distinction entre les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

Réseaux identifiés : Rappel

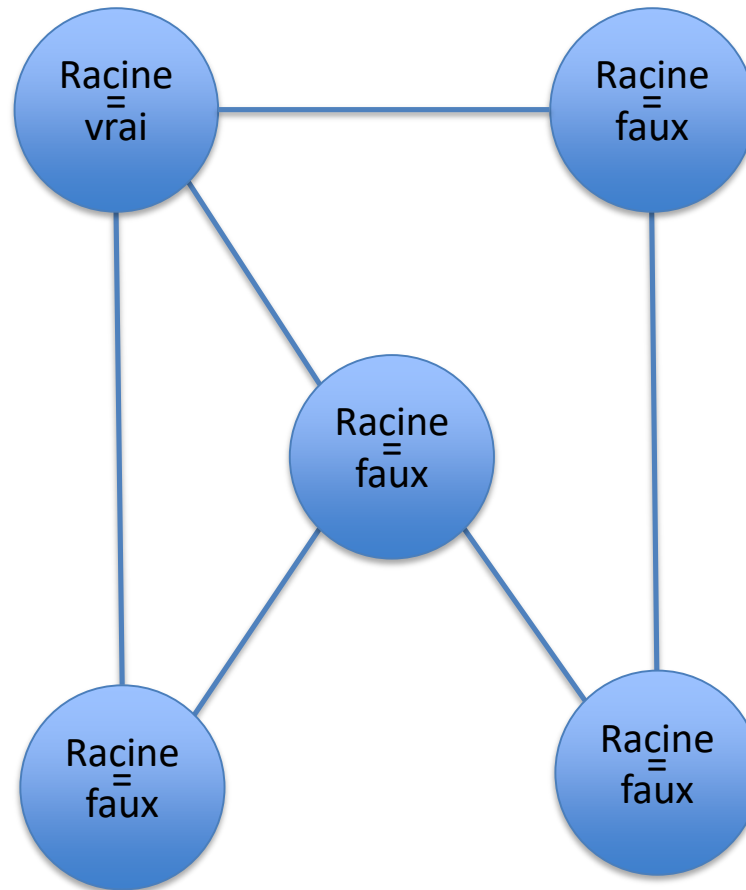


- Identité unique
(*e.g.*, adresse IP)

Réseaux enracinés : Rappel



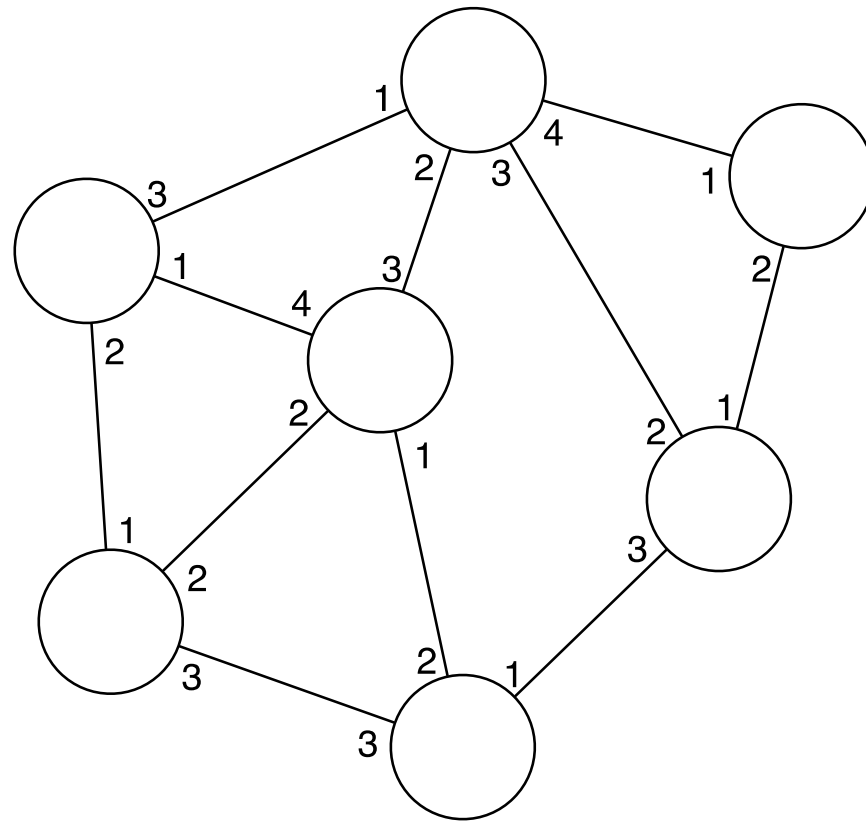
Réseaux enracinés : Rappel



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)

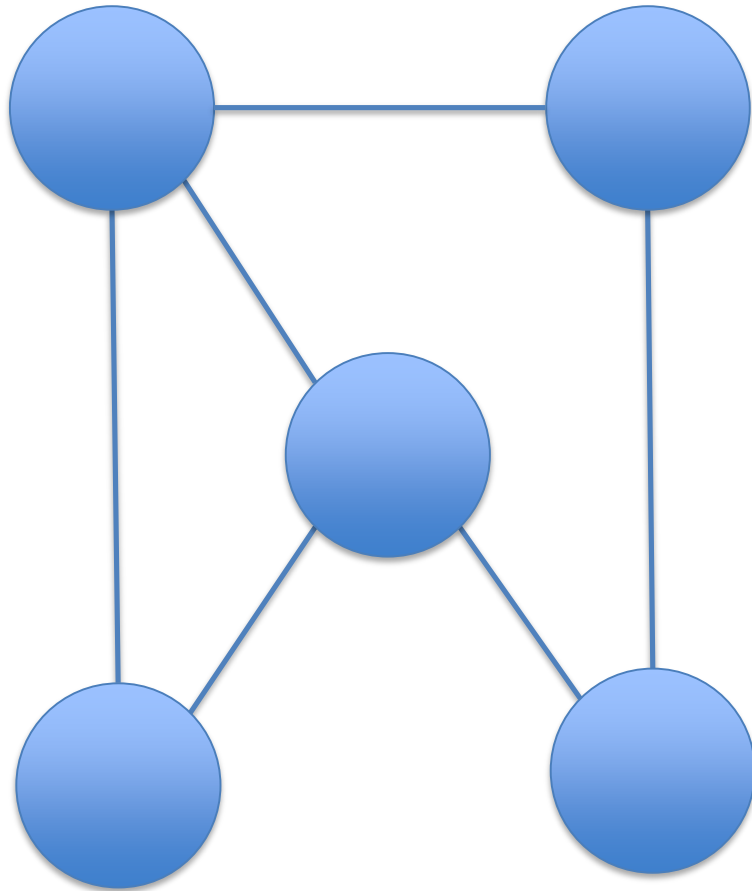
Numérotation des canaux



Exemple récapitulatif

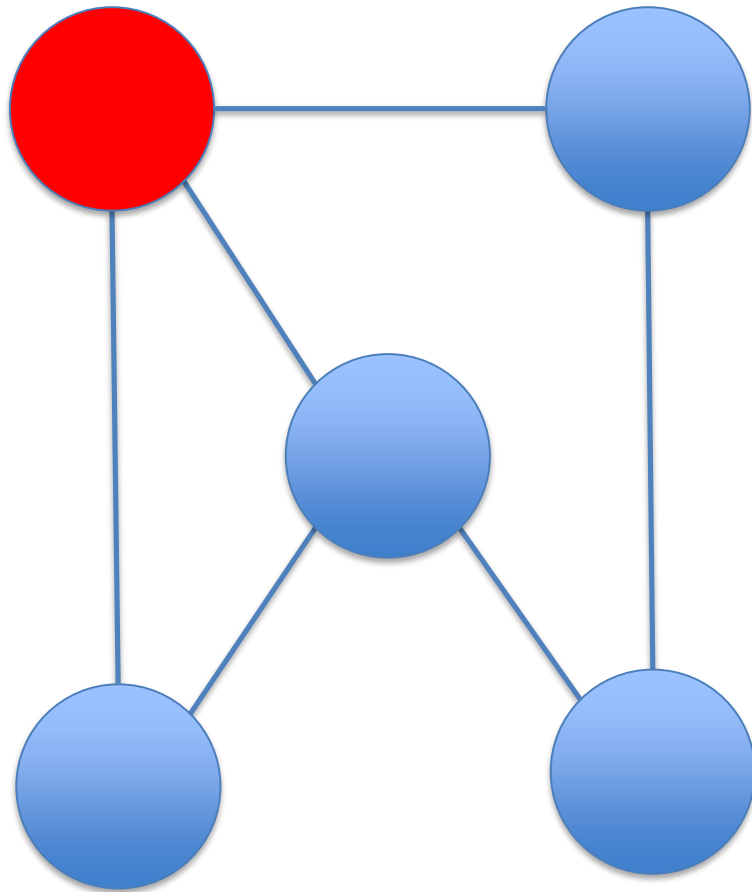
Algorithme Distribu 

Exemple : Calcul d'un arbre couvrant



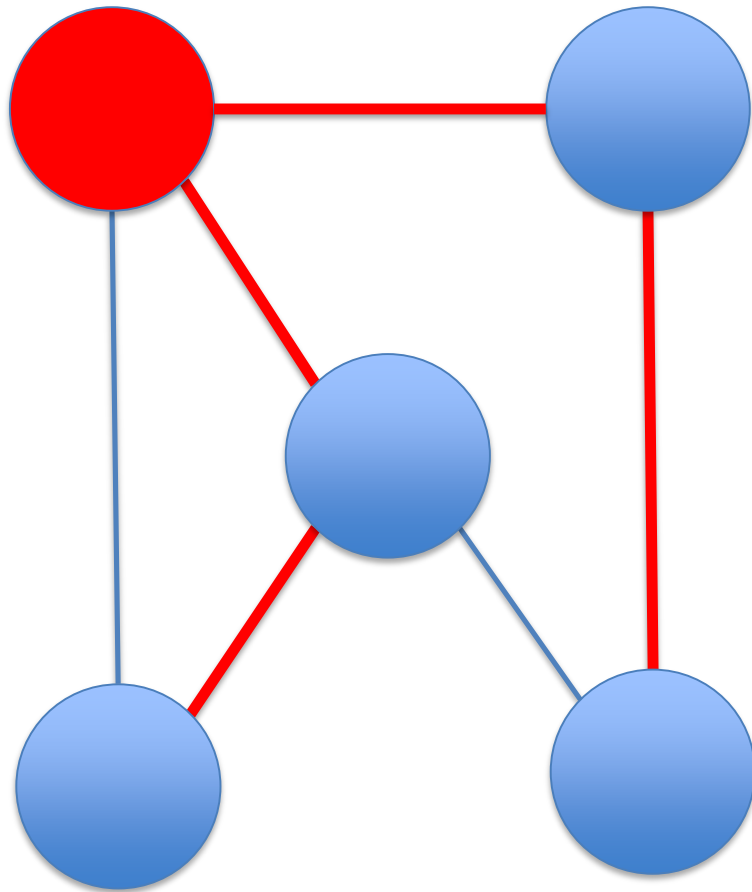
Algorithme Distribu 

Exemple : Calcul d'un arbre couvrant



Algorithme Distribu 

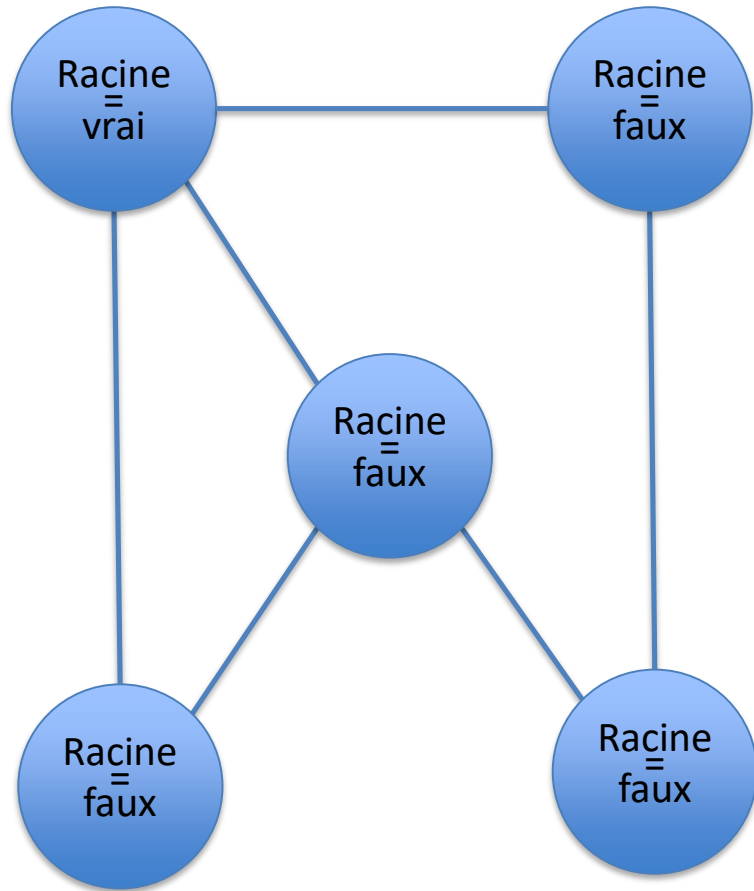
Exemple : Calcul d'un arbre couvrant



Algorithme Distribu 

Exemple : Calcul d'un arbre couvrant

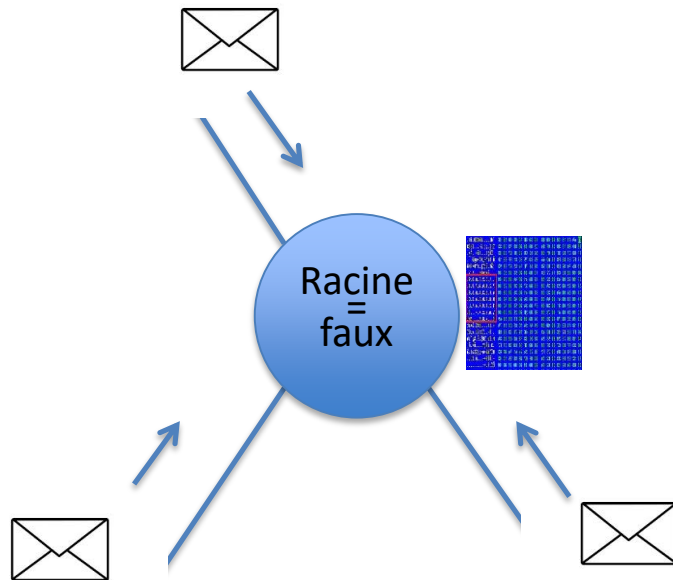
- Entr es r parties



Algorithme Distribu 

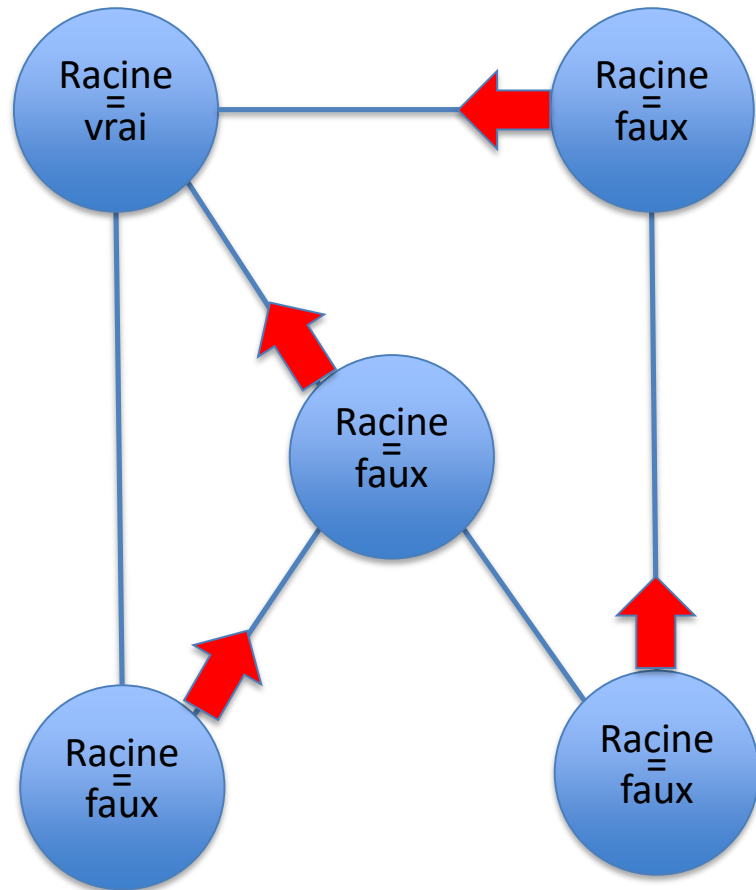
Exemple : Calcul d'un arbre couvrant

- Entr es r parties
- Calculs locaux
 - M moires locales
 - Programmes locaux
 - Envoi de messages
 - **D cision locale**



Algorithme Distribué

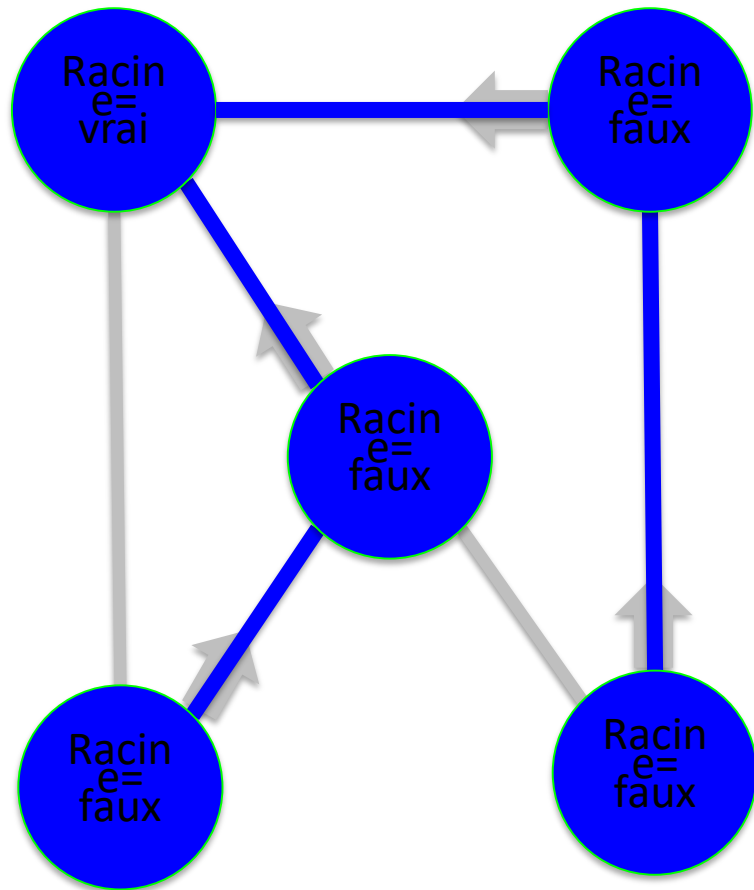
Exemple : Calcul d'un arbre couvrant



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

Algorithme Distribu 

Exemple : Calcul d'un arbre couvrant



- Entr es r parties
- Calculs locaux
 - M moires locales
 - Programmes locaux
 - Envoi de messages
 - **Decision locale**
- Sorties r parties
- T che 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

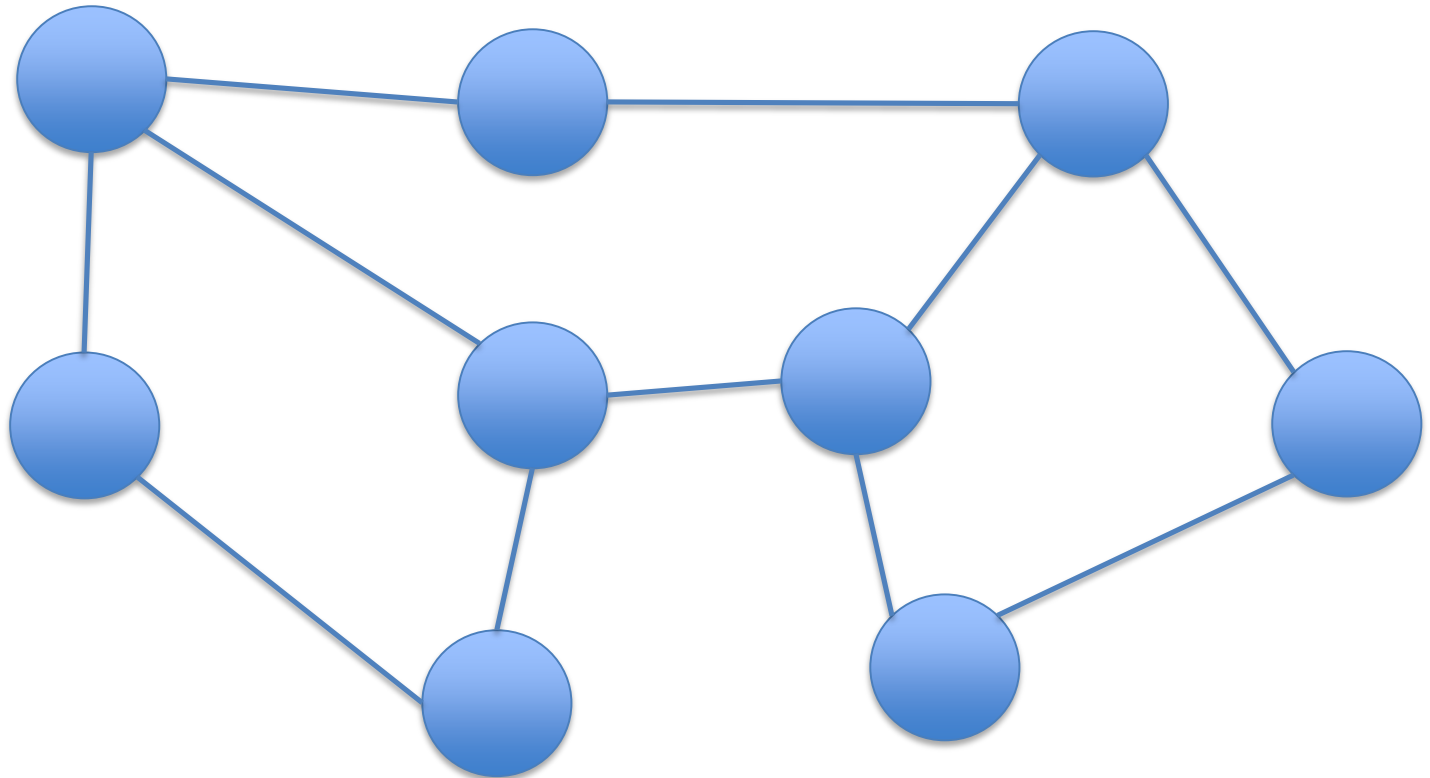
- Complexité en messages
- Complexité en volume
- Complexité en temps
 - Temps de traitement = 0, temps d'acheminement = 1
- Complexité en espace (bits ou états)

Problèmes classiques

Problèmes classiques

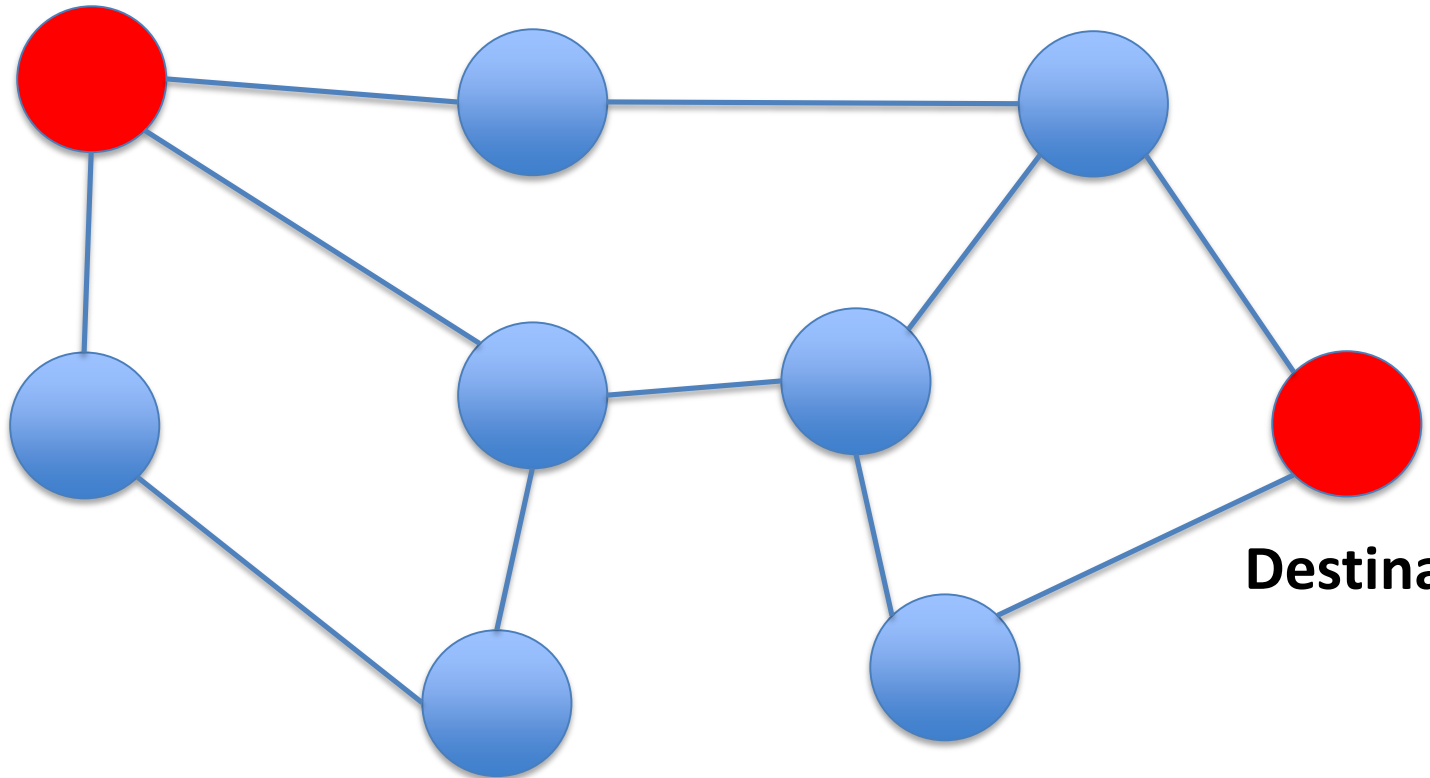
- **Echange de donnée** : routage, diffusion, ...
- **Accords** : consensus, élection, ...
- **Auto-organisation** : arbre couvrant, clustering
- **Allocation de ressources** : exclusion mutuelle, dîner des philosophes...

Echange de donnée : routage



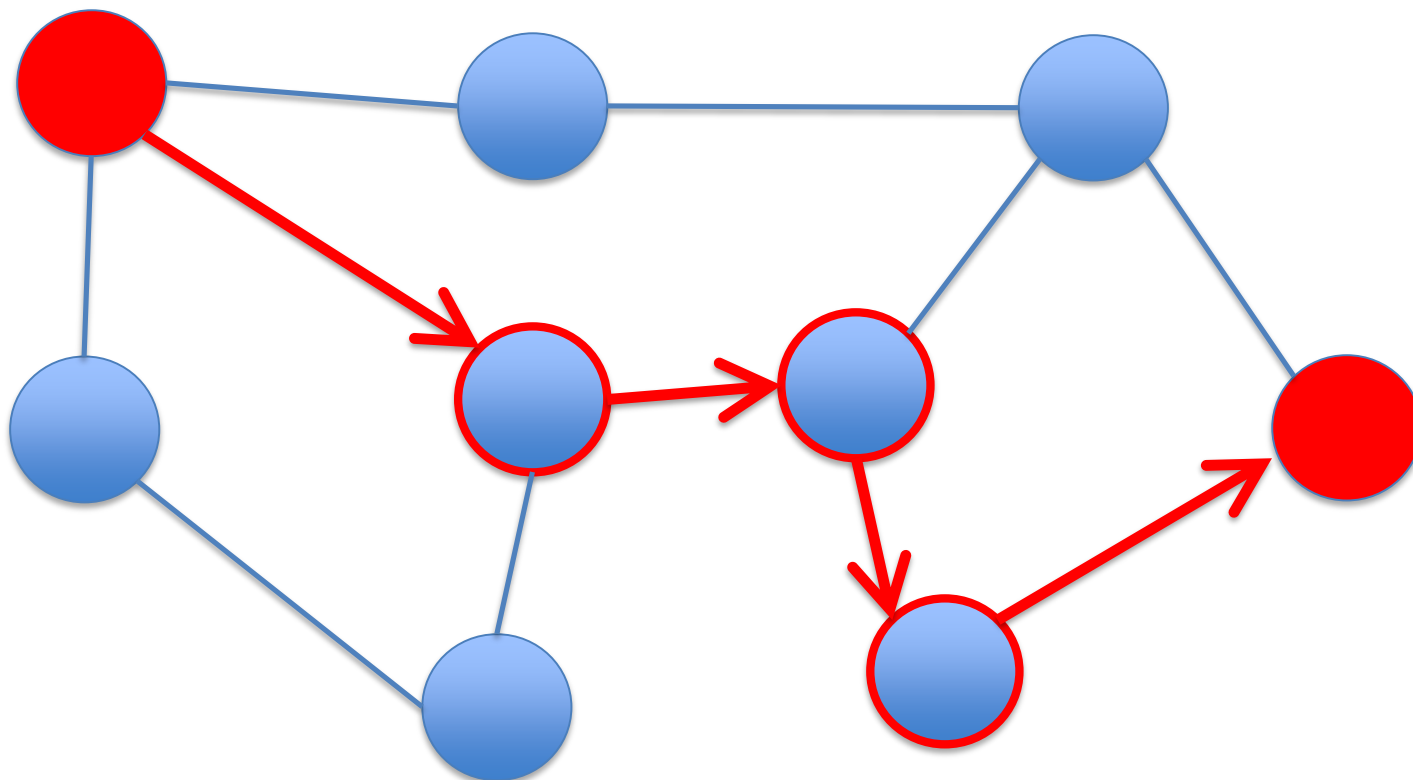
Echange de donnée : routage

Source



Destination

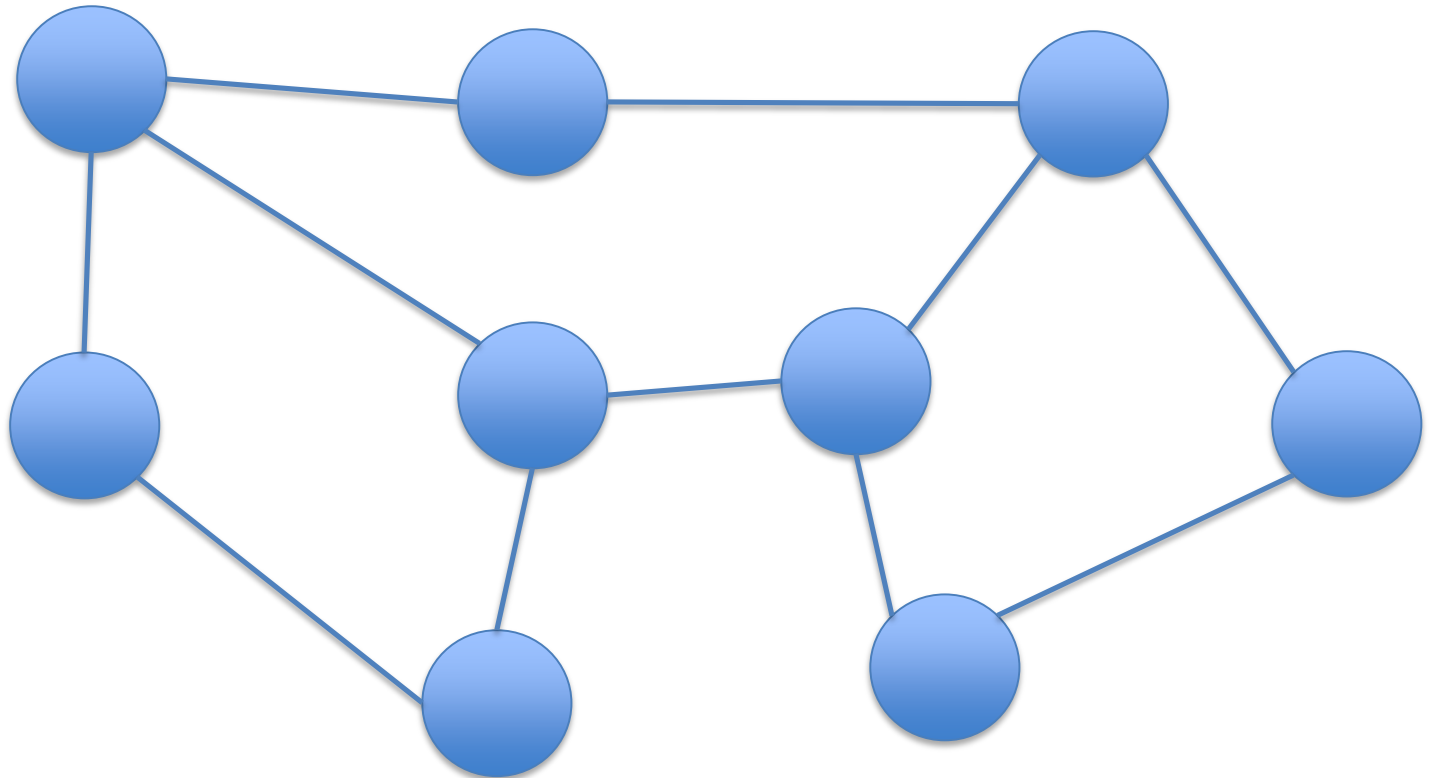
Echange de donnée : routage



Accord : élection

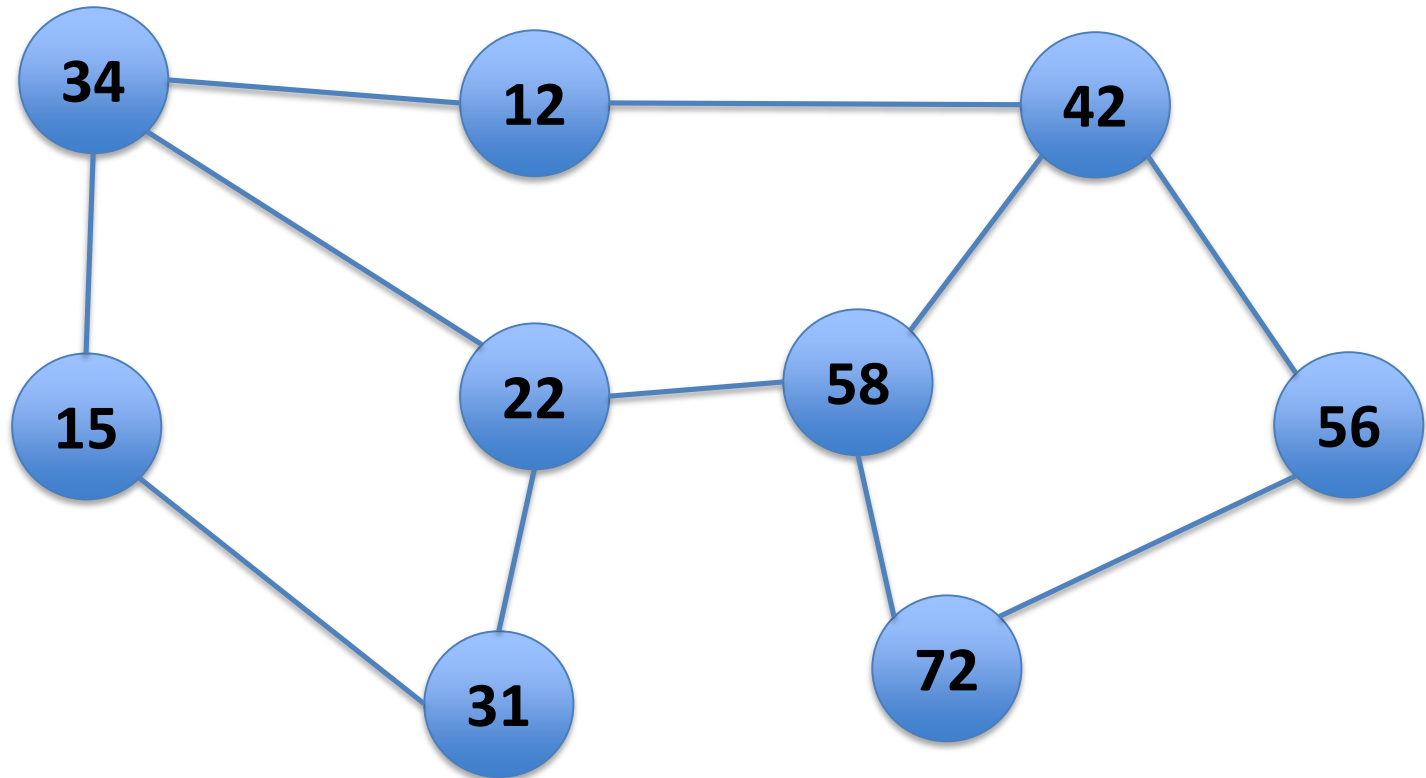
Calculer un chef !

(avec publication ou non)



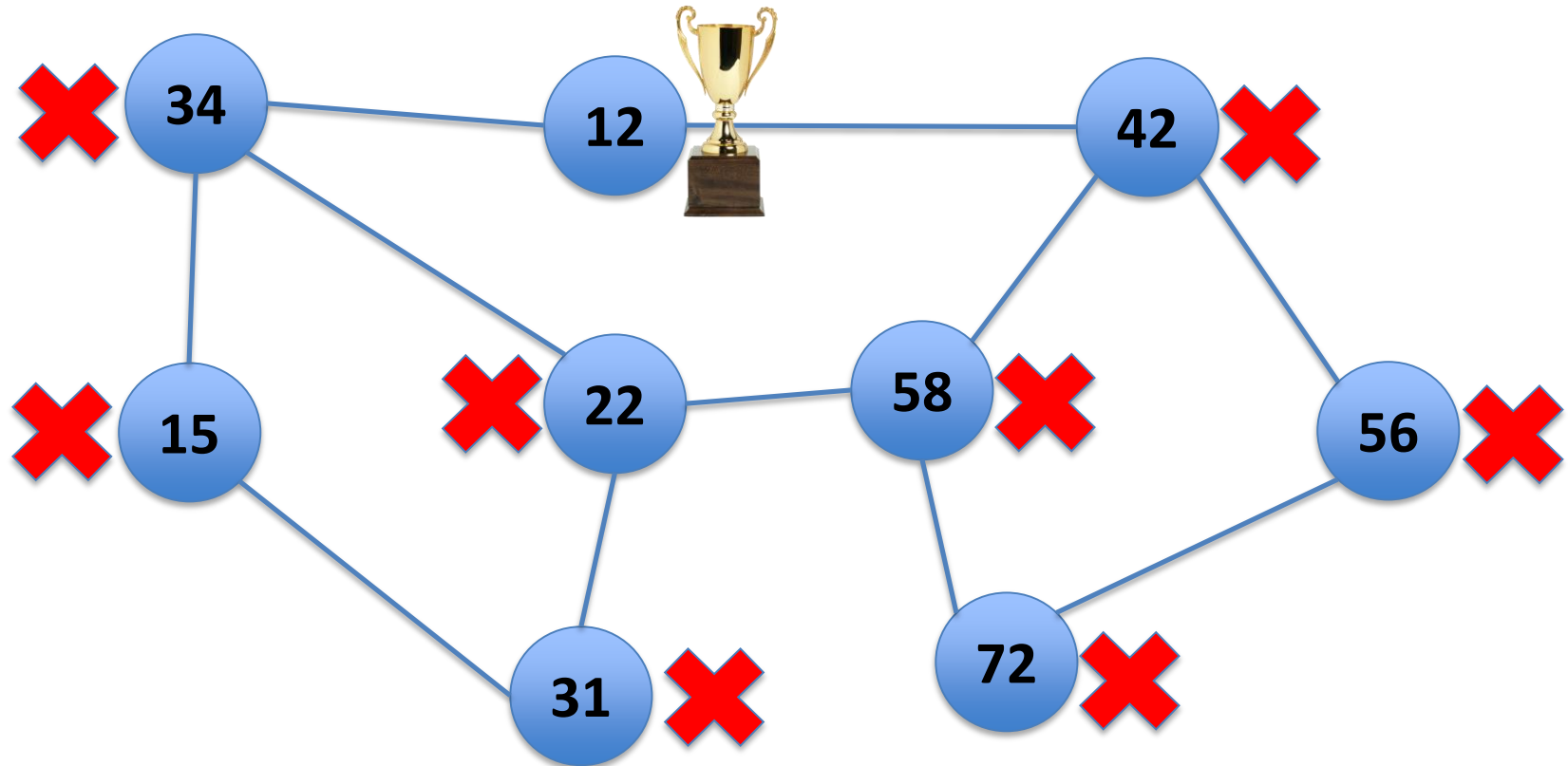
Accord : élection

Calculer un chef !



Accord : élection

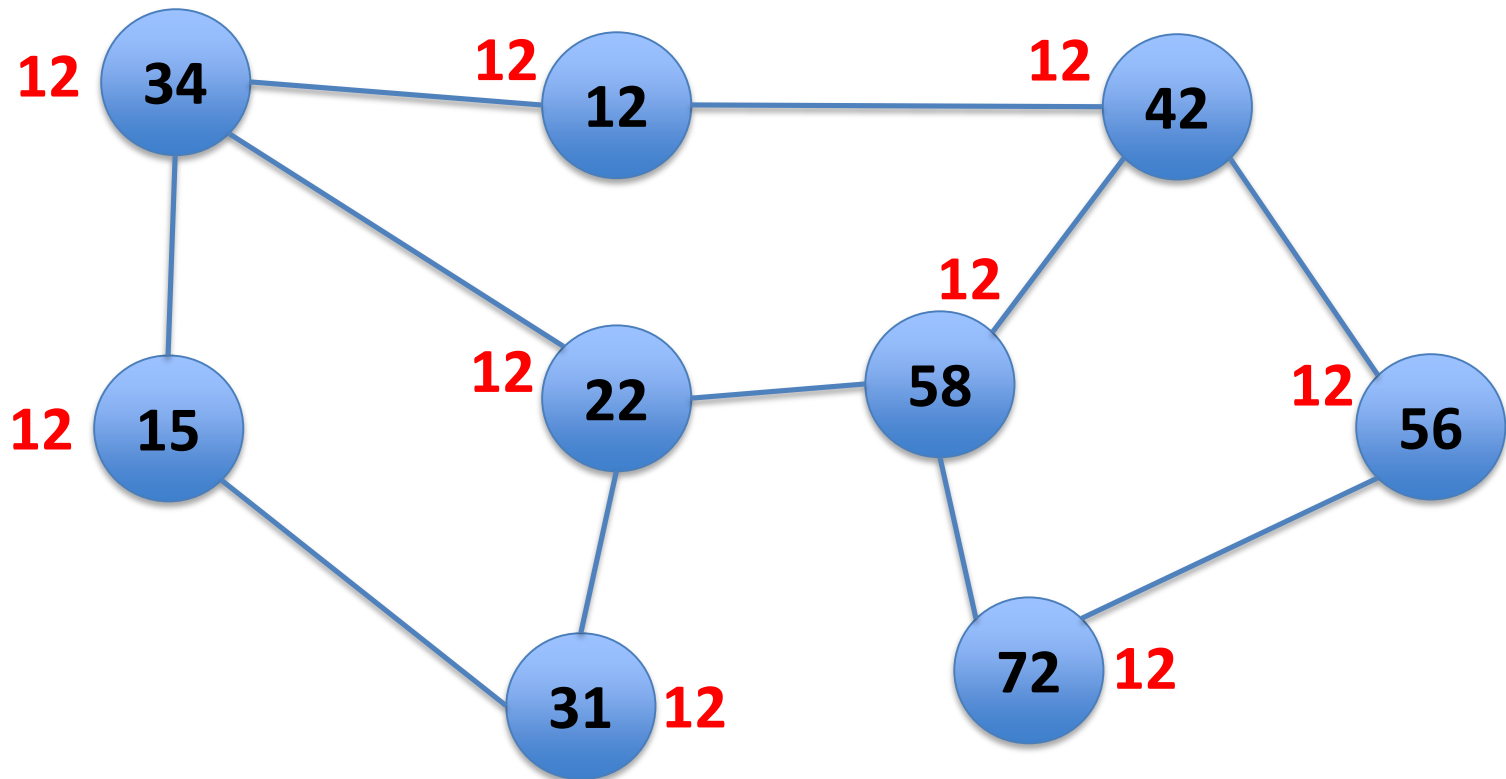
Calculer un chef !



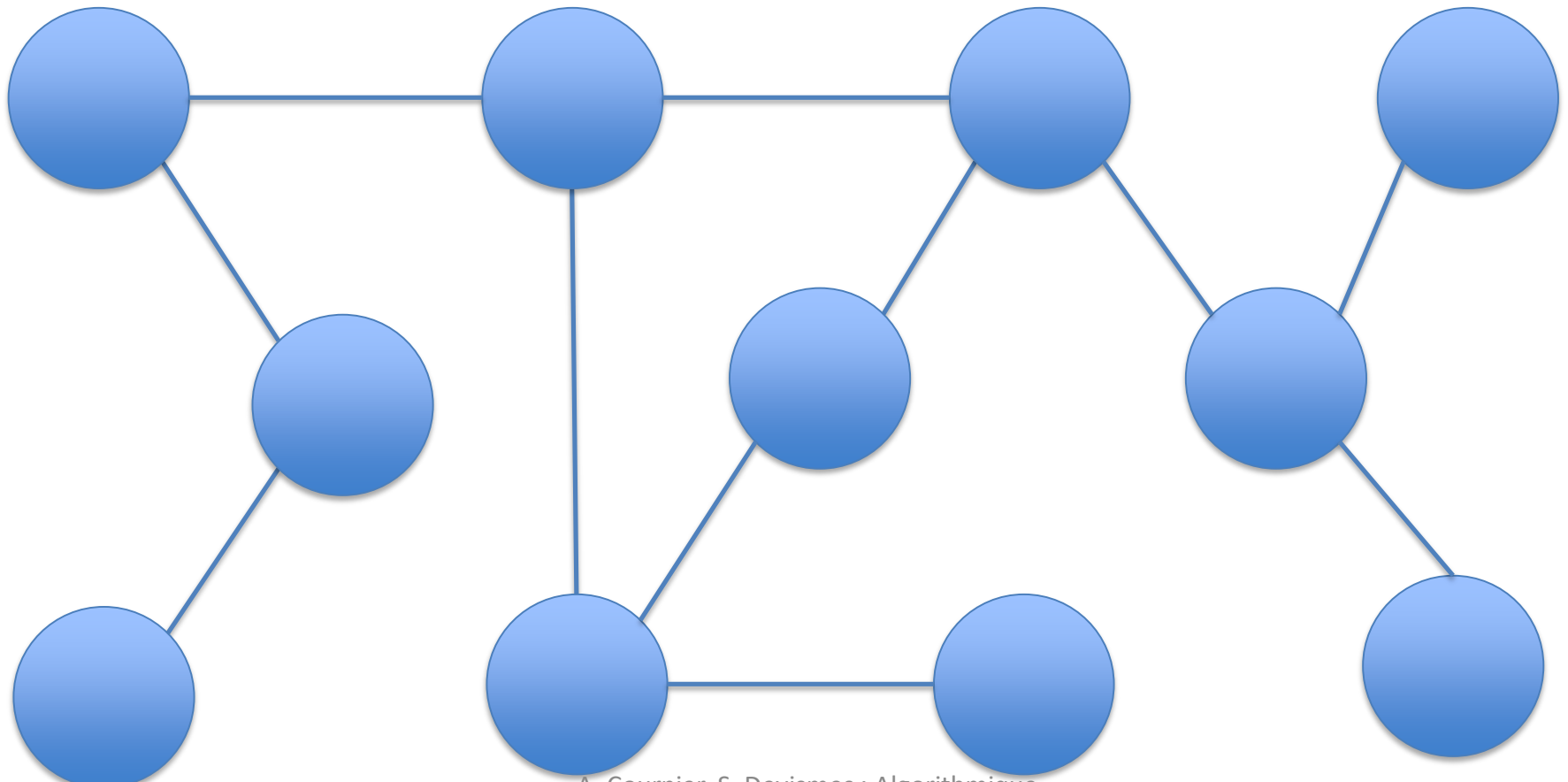
Accord : élection

Calculer un chef !

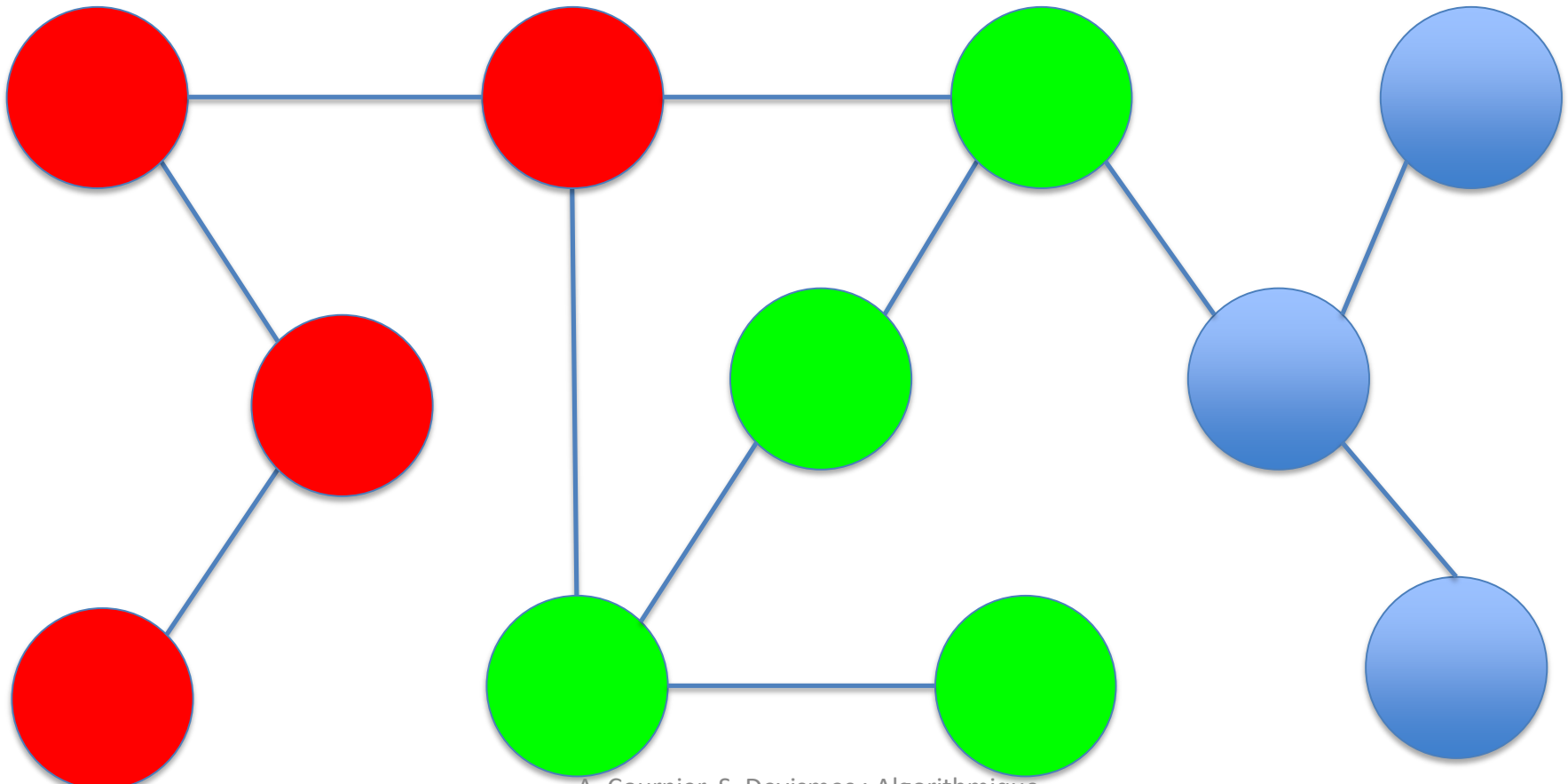
(avec publication)



Auto-organisation : k -Clustering

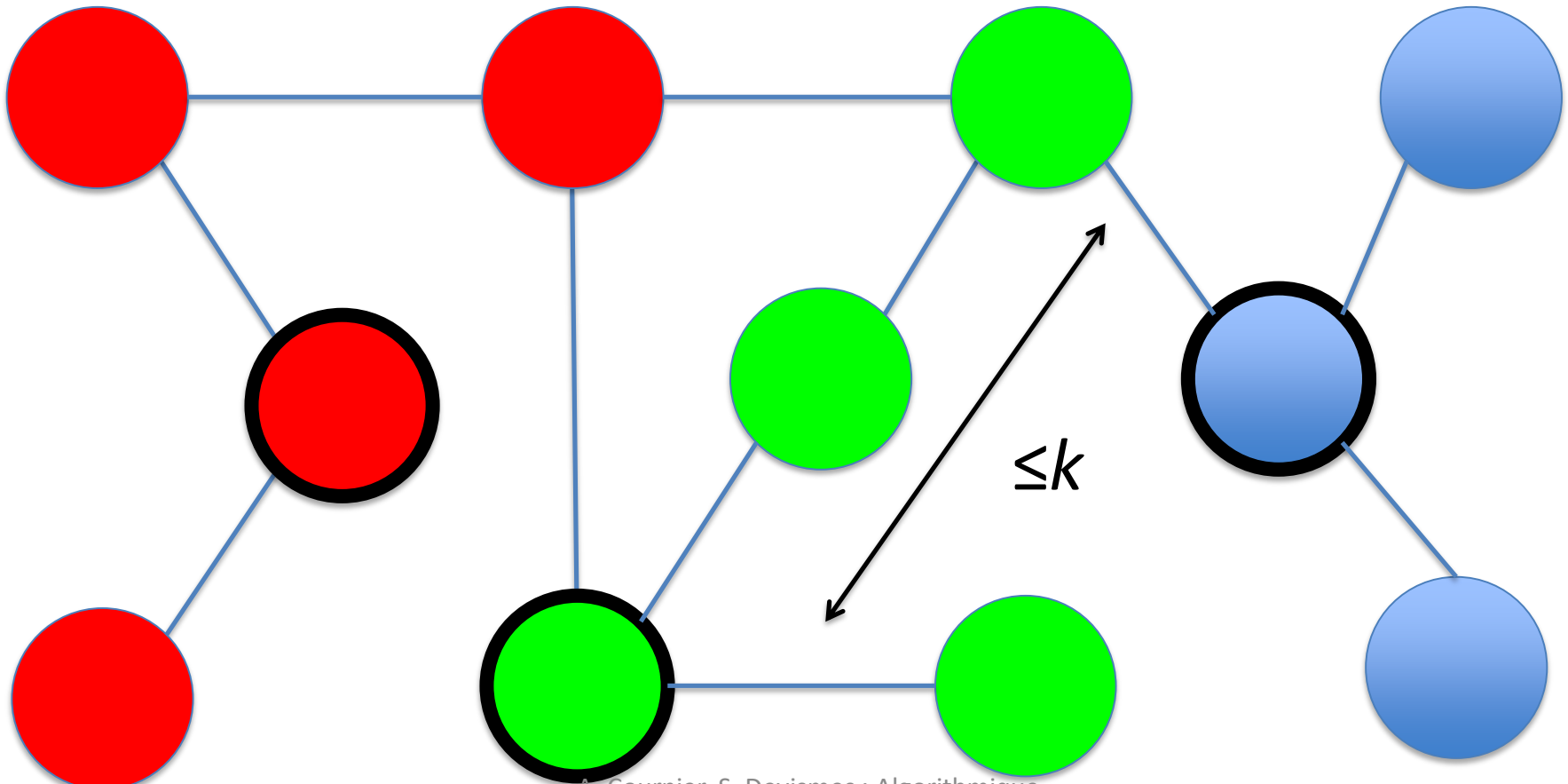


Auto-organisation : k -Clustering



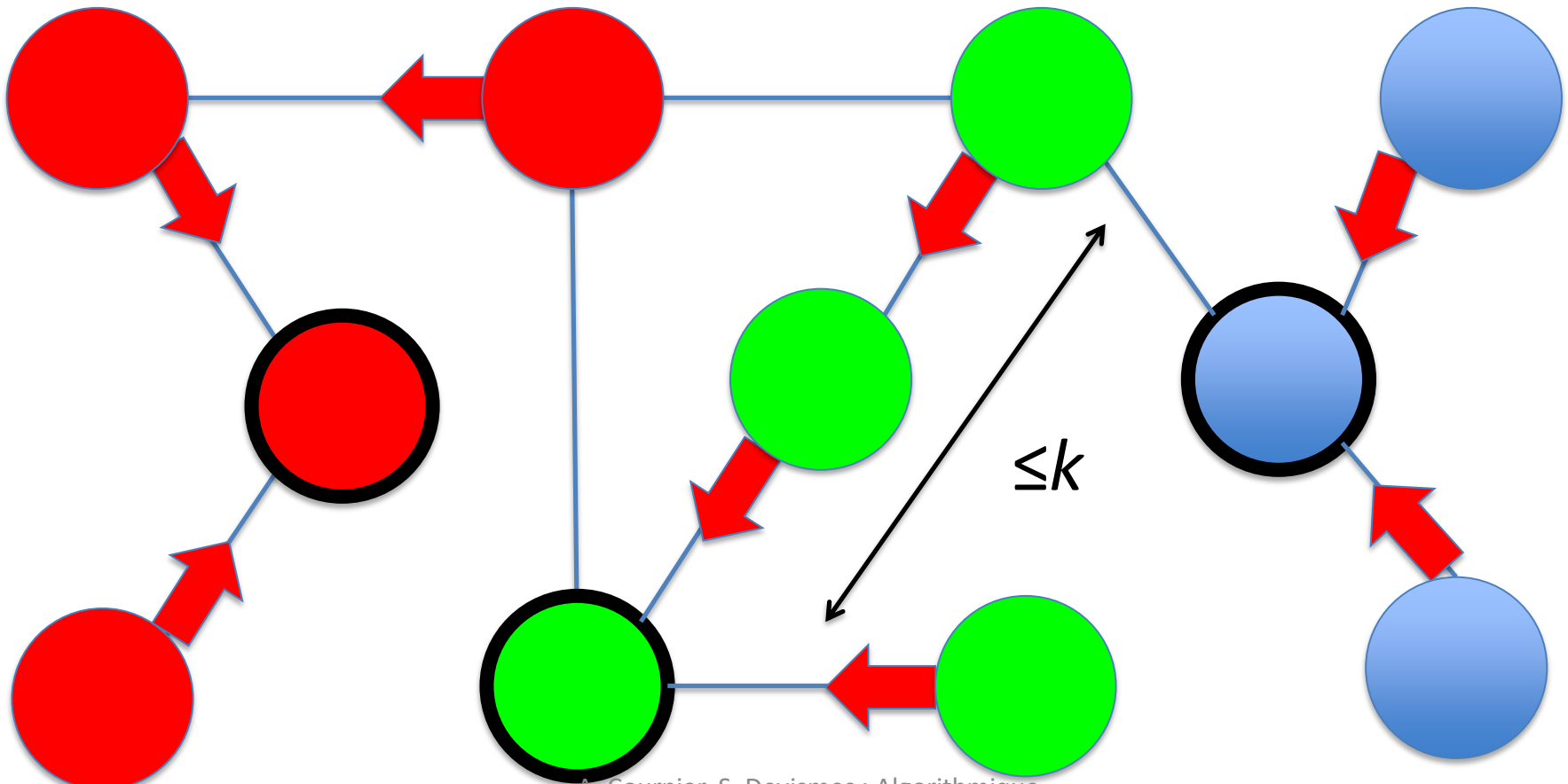
Auto-organisation : k -Clustering

- Ex. $k=2$

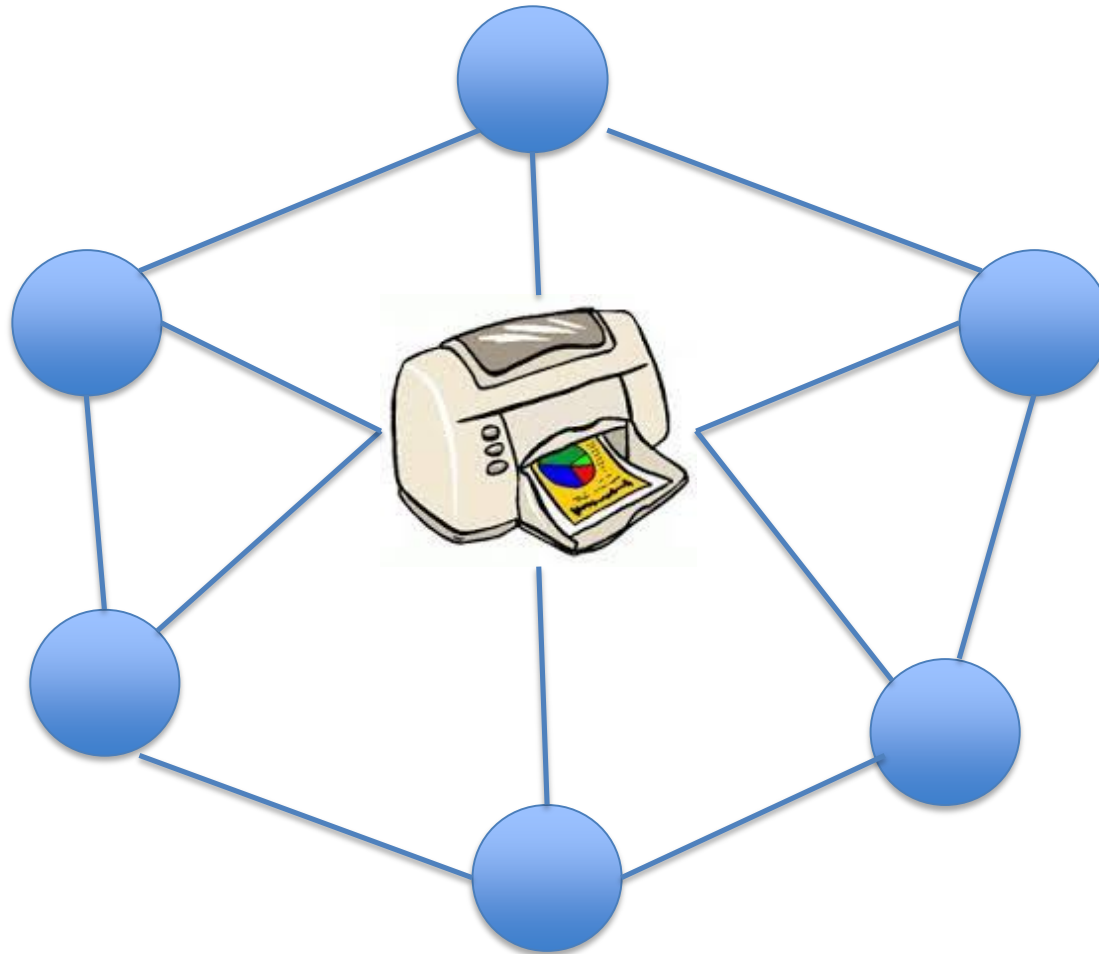


Auto-organisation : k -Clustering

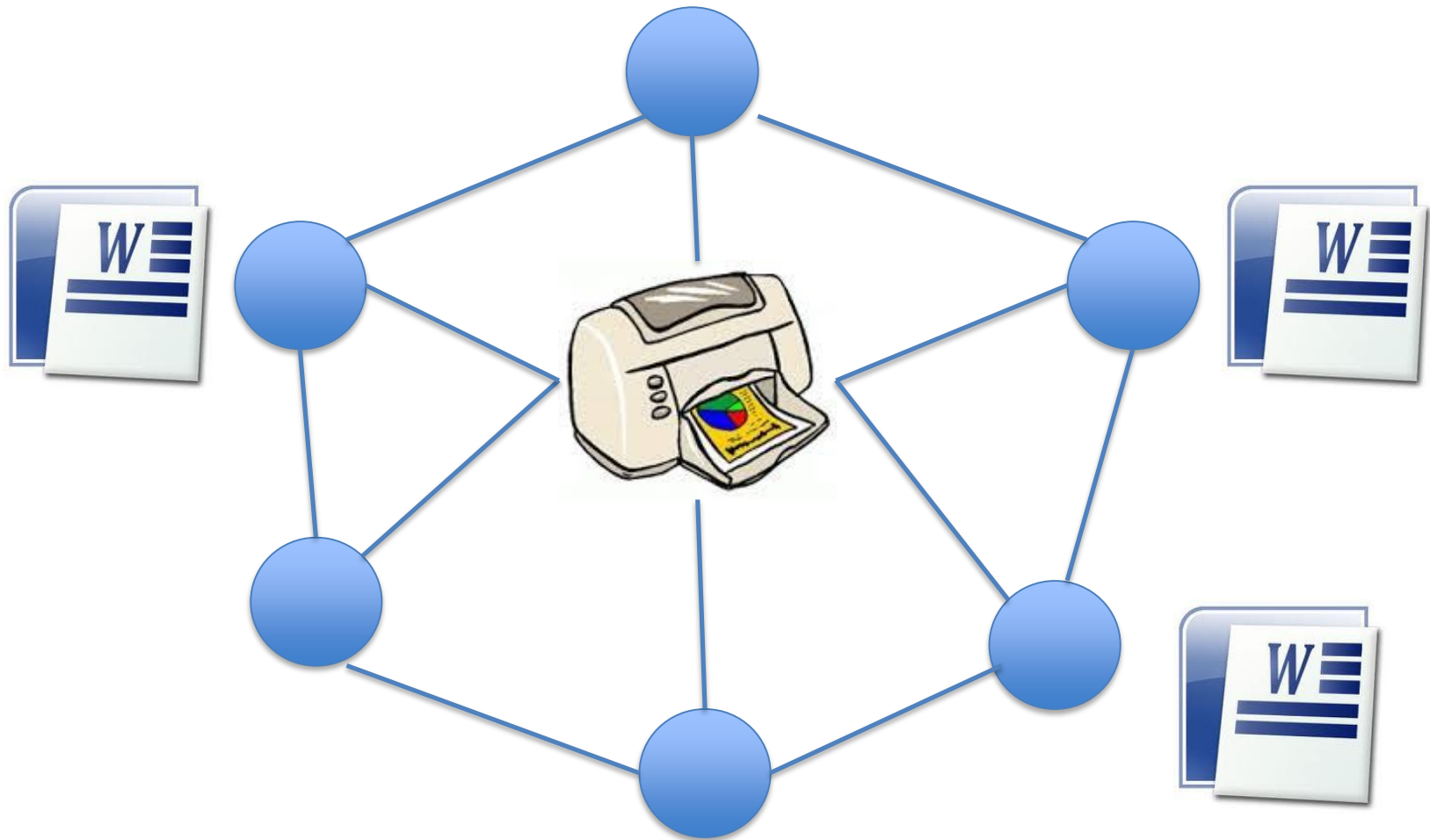
- Ex. $k=2$



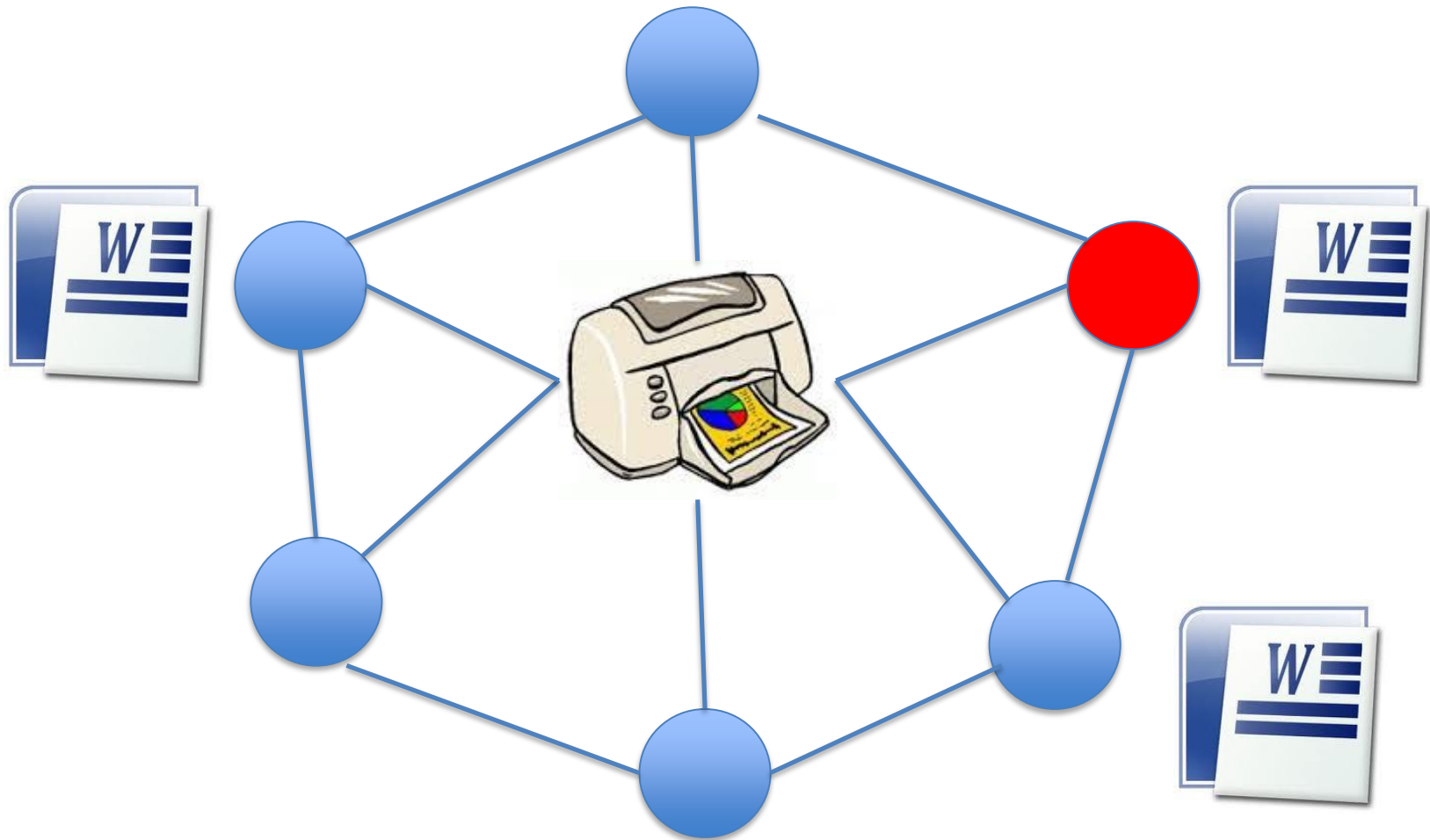
Allocation de ressources : exclusion mutuelle



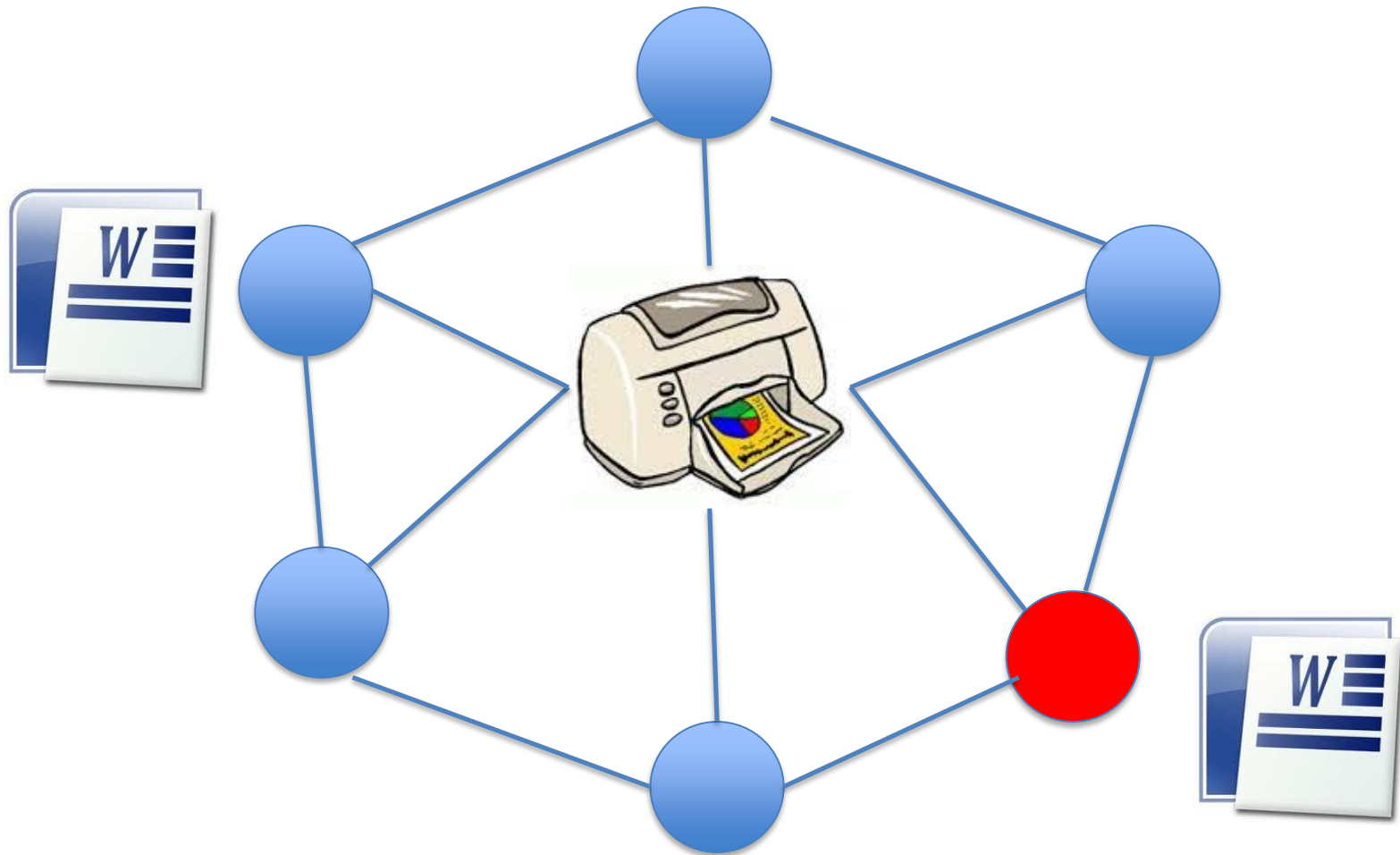
Allocation de ressources : exclusion mutuelle



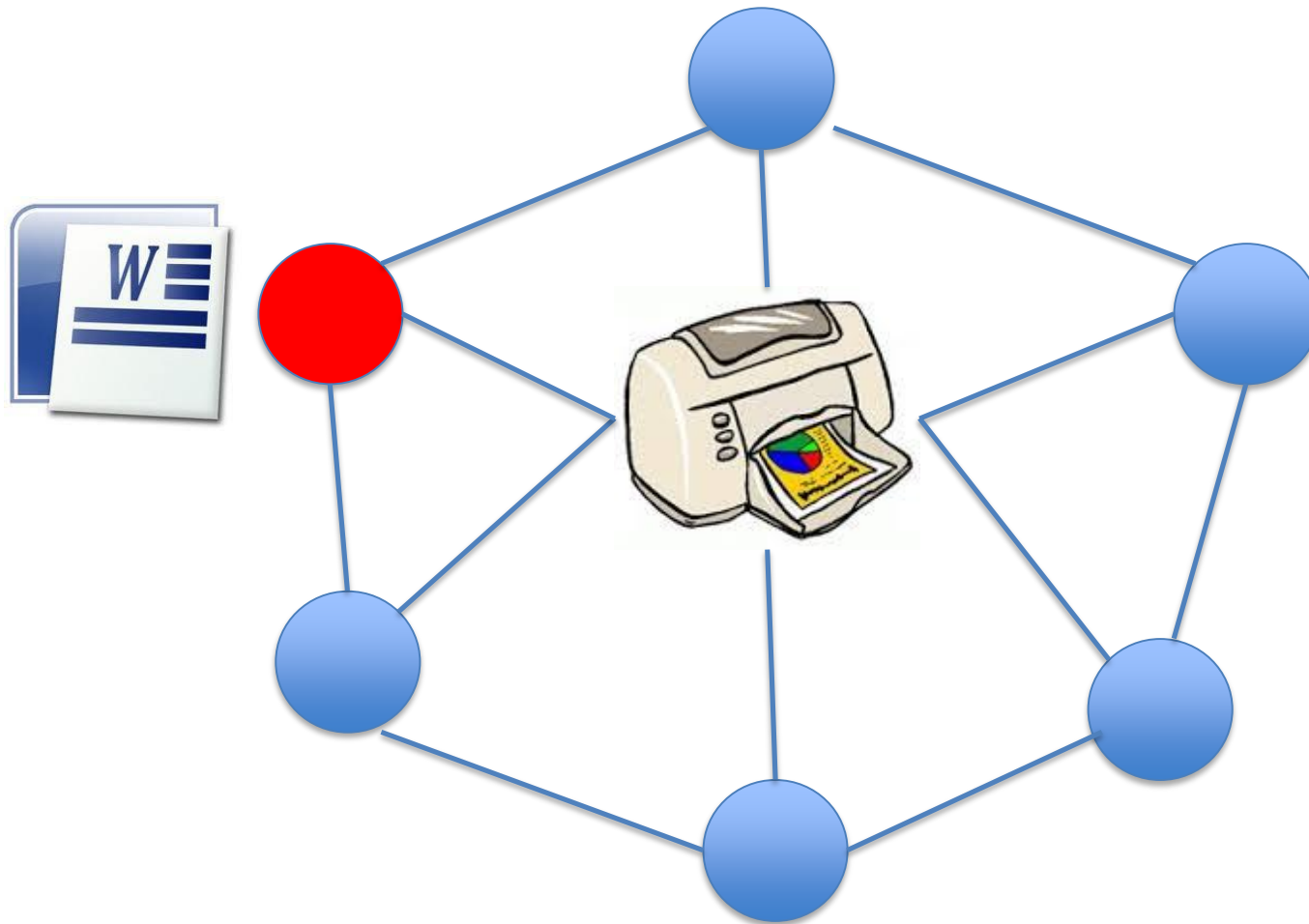
Allocation de ressources : exclusion mutuelle



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

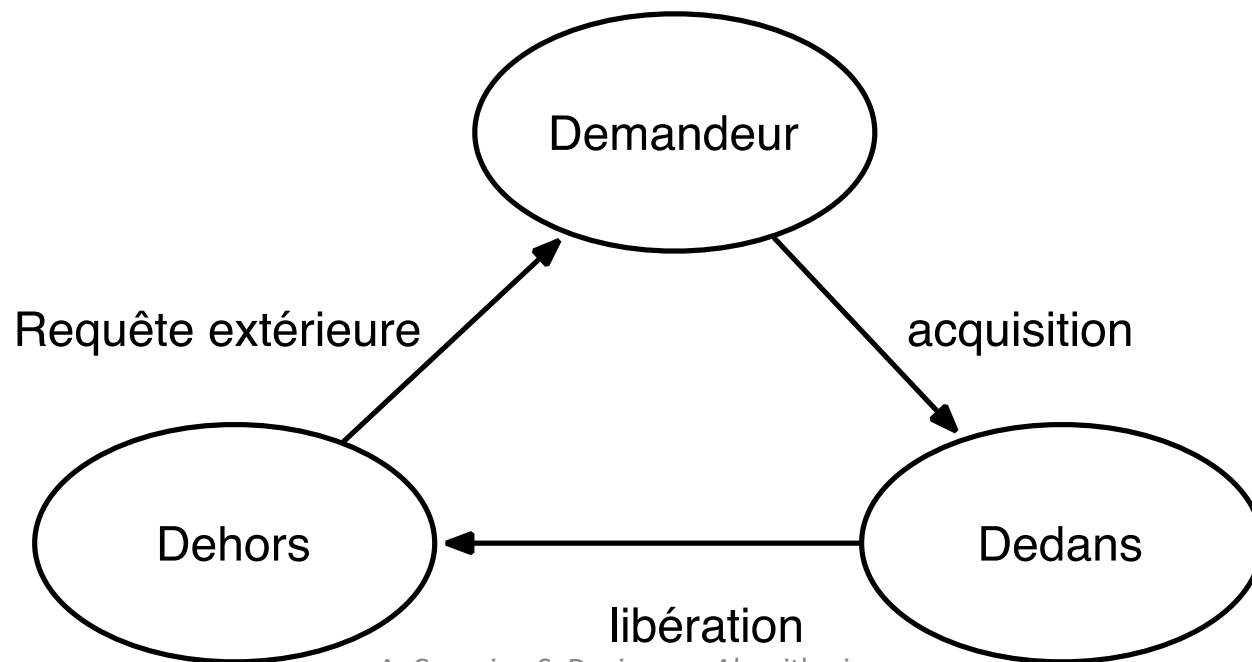
- Enoncé formel du problème que l'algorithme doit résoudre
- **Sûreté (Safety) + Vivacité (Liveness)**
 - **Sûreté** : l'ensemble des propriétés qui doivent être vérifiées à chaque instant de l'exécution de l'algorithme.
« Rien de mal ne doit arriver. »
 - **Vivacité** : L'ensemble des propriétés qui doivent être vérifiées à certains instants de l'exécution de l'algorithme.
« Quelque chose de bien finit par arriver. »

Preuve (de correction) d'un algorithme

- **Preuve** : raisonnement logique pour déduire une **conclusion** à partir d'**hypothèses**
- Ici la conclusion c'est la **spécification** de l'algorithme qui doit être déduite à partir des **hypothèses sur le système**
- **Remarque** : une preuve est un enchainement de *faits élémentaires* obtenus en appliquant des *règles logiques* sur des *hypothèses de départ* et/ou des *faits élémentaires déduits précédemment dans la preuve*
- **Règles logiques** : induction, contraposée, tiers exclu, réduction par l'absurde, modus ponens, modus tollens, *etc*

Spécification de l'exclusion mutuelle

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

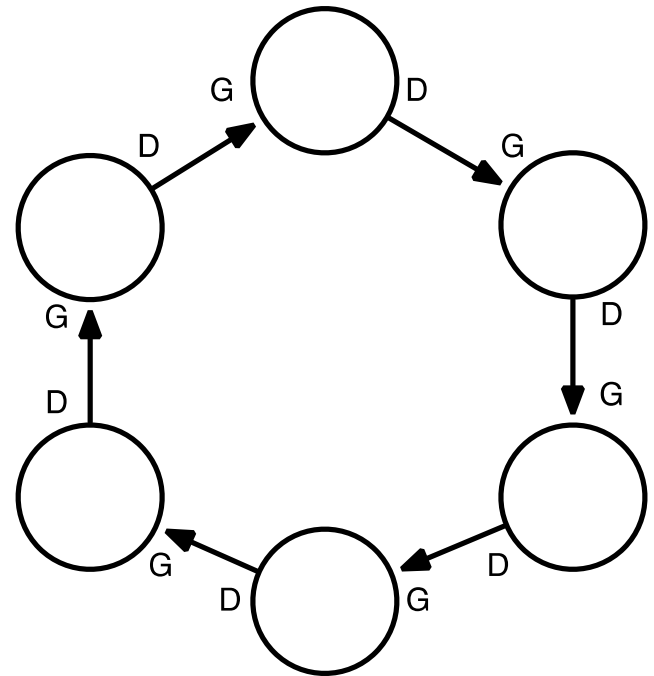


Spécification de l'exclusion mutuelle

- **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**)

Algorithme 1 Exclusion mutuelle de Le Lann

Variables

1: $Etat \in \{dehors, demandeur, dedans\}$ initialisé à *dehors*
/* Entrée-Sortie, modifiée aussi par la couche applicative */

Spontanément /* Démarrage de l'algorithme */
/* exécuté en 1^{er} par l'initiateur uniquement */

2: Envoyer $\langle J \rangle$ à D

Réception de $\langle J \rangle$ de G

3: **Si** $Etat = demandeur$ **alors**

4: $Etat \leftarrow dedans$

5: SC /* Section critique */

6: $Etat \leftarrow dehors$

7: **Fin Si**

8: Envoyer $\langle J \rangle$ à D

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 : Au démarrage, 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 lors du démarrage 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. \square

Preuve de correction

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

Preuve.

1. Le démarrage 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. \square

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

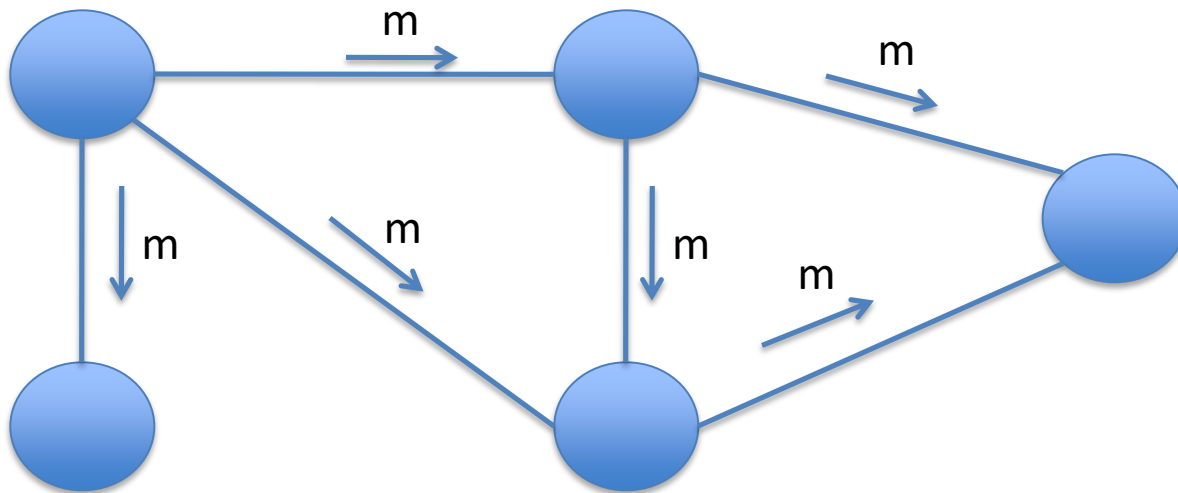
est un

algorithme à vague

Algorithme à vague

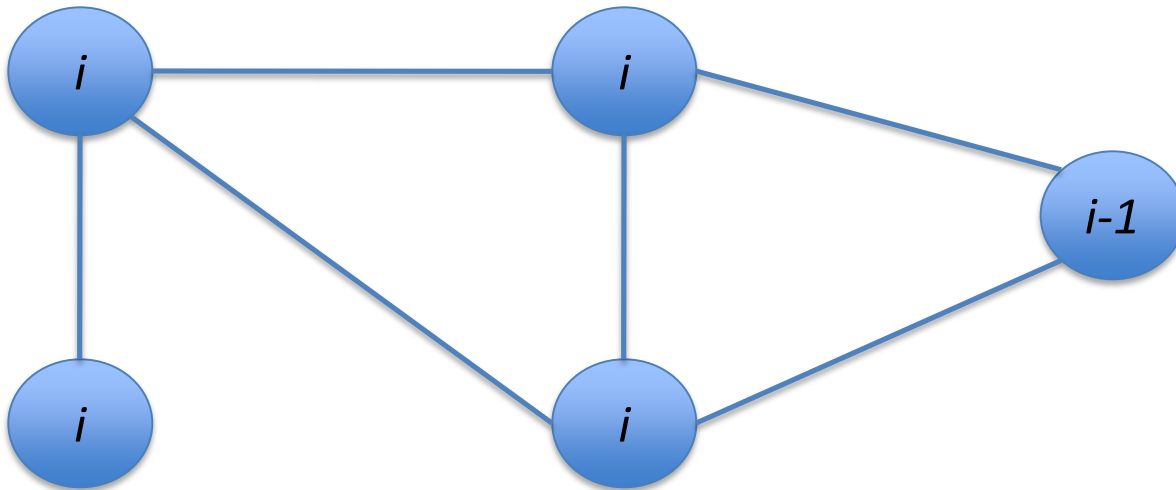
Introduction

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



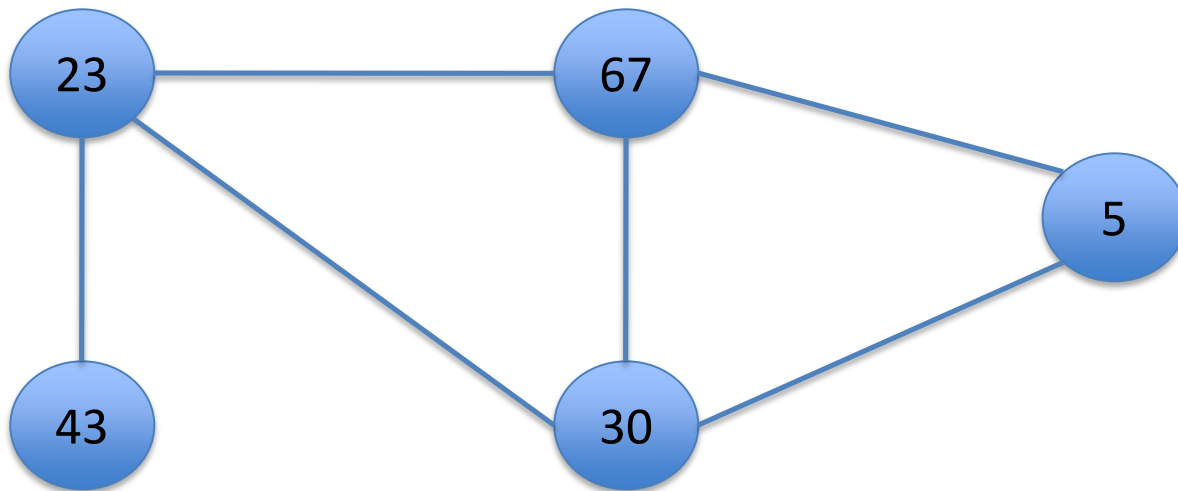
Introduction

- 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 : prise d'instantané, détection de terminaison, calcul d'infimum, *etc.*

Instanciación

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

Instanciación

- 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

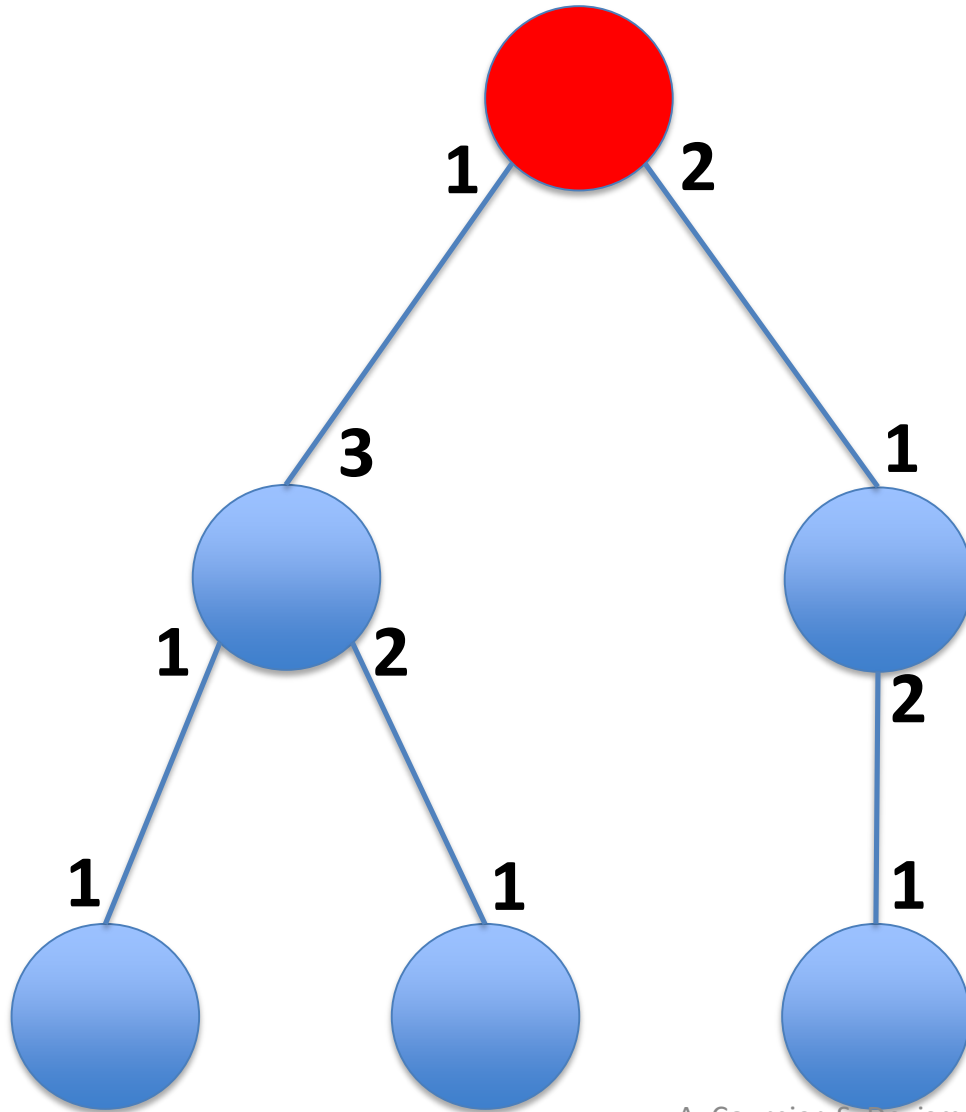
Hypothèses pour notre circulation de jeton

- 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

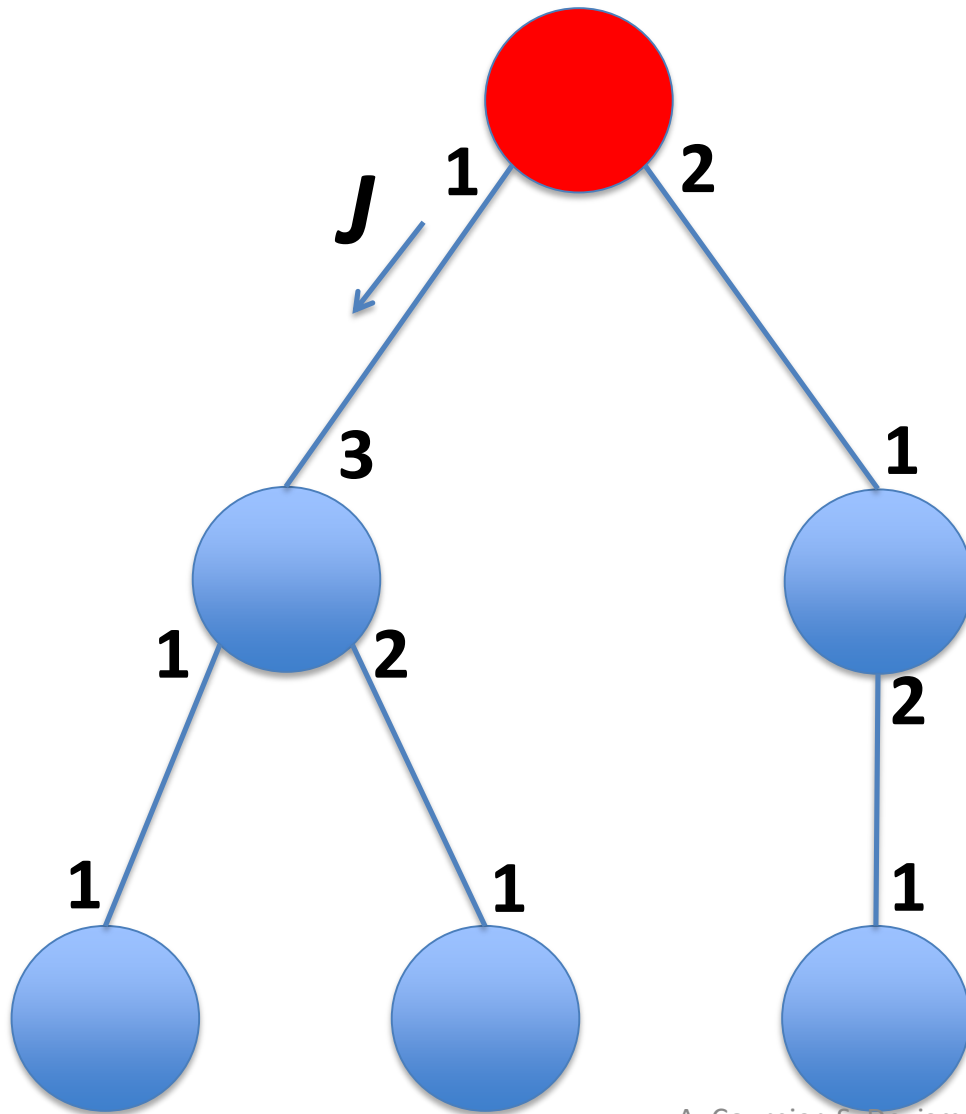
Rappel

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

Circulation d'un jeton dans un arbre

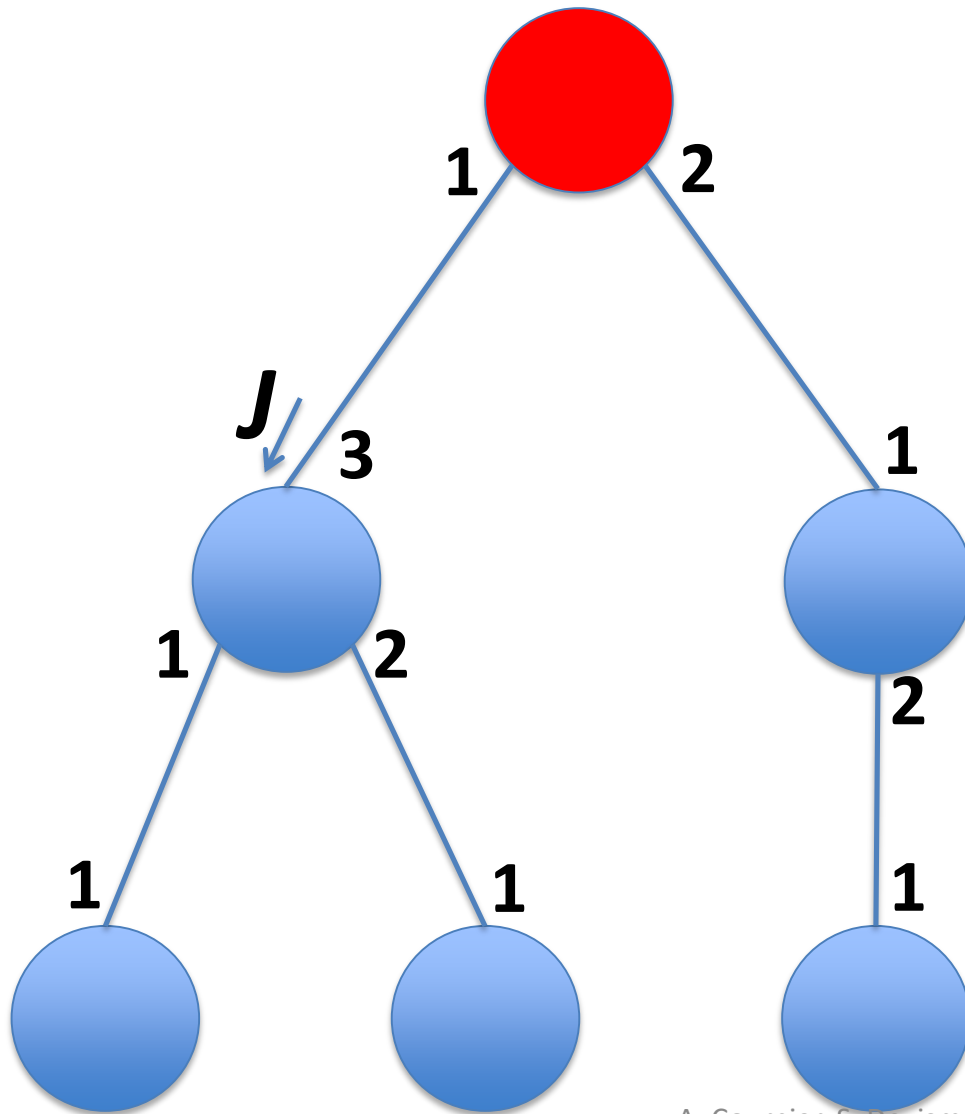


Circulation d'un jeton dans un arbre



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

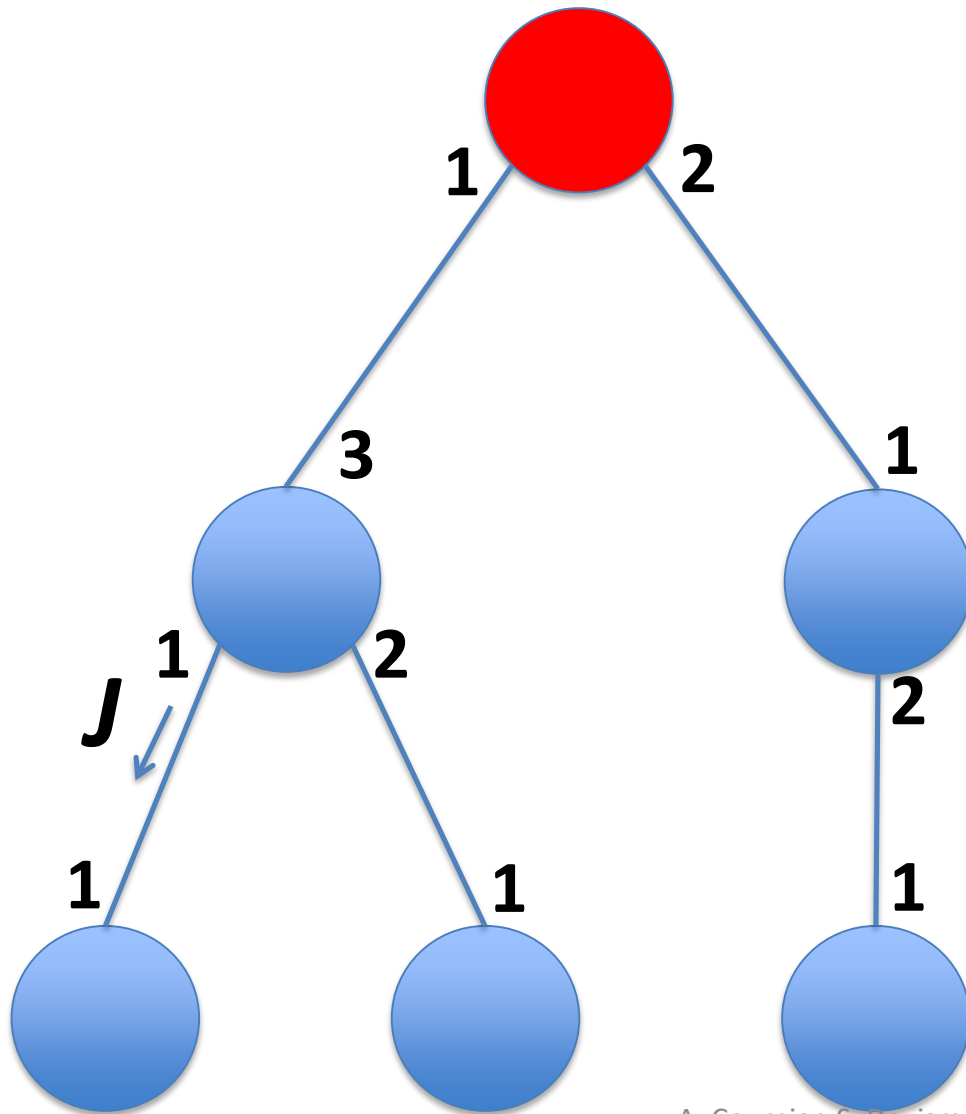
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 3$
- $(3 \bmod 3) + 1 = 1$

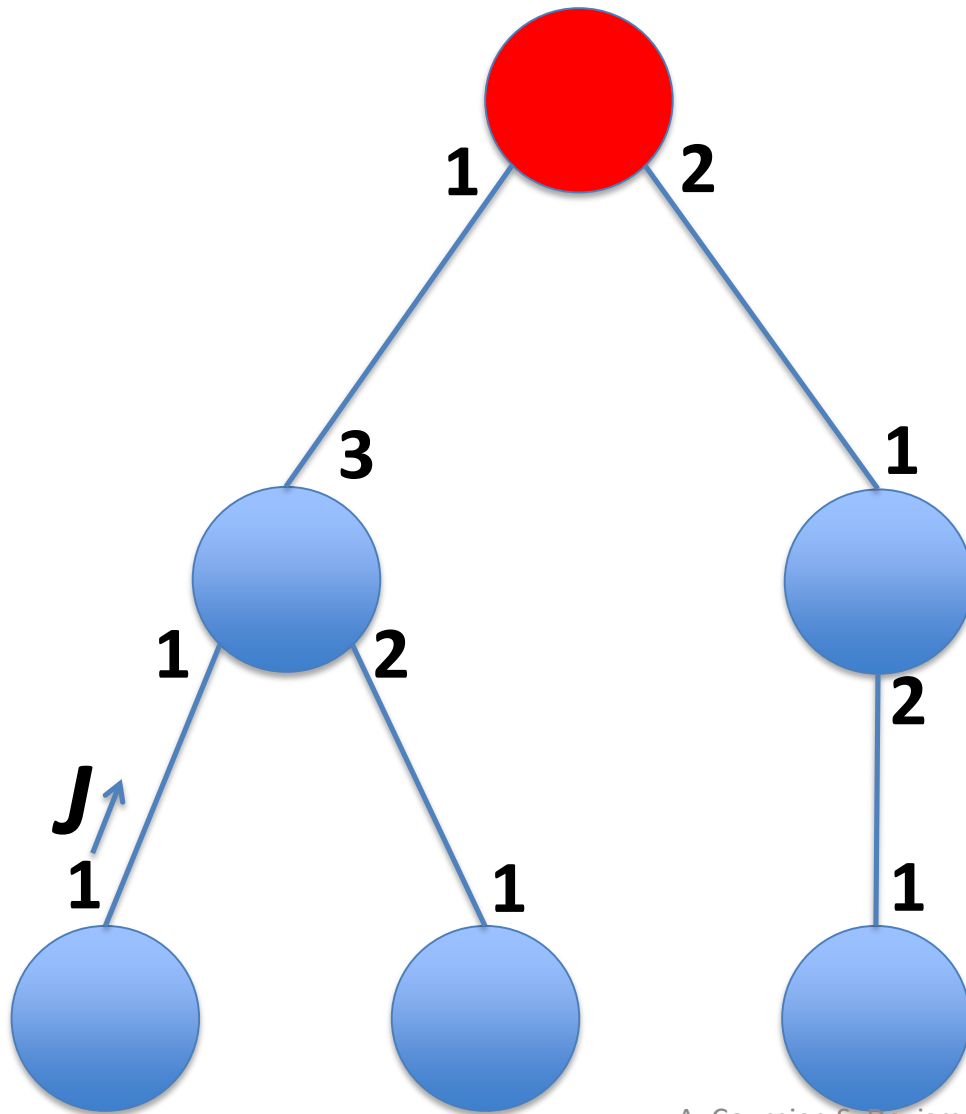
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 3$
- $(3 \bmod 3) + 1 = 1$

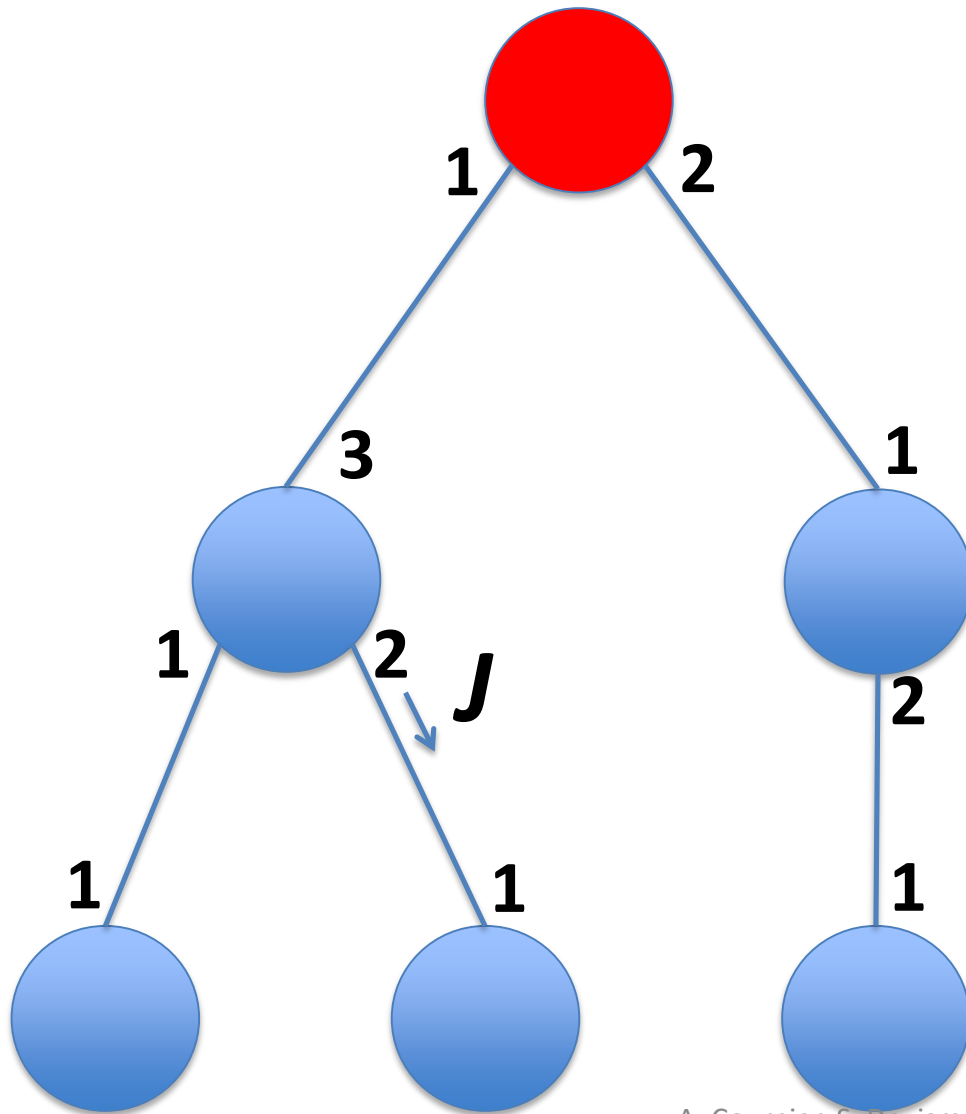
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 1$
- $(1 \bmod 1) + 1 = 1$

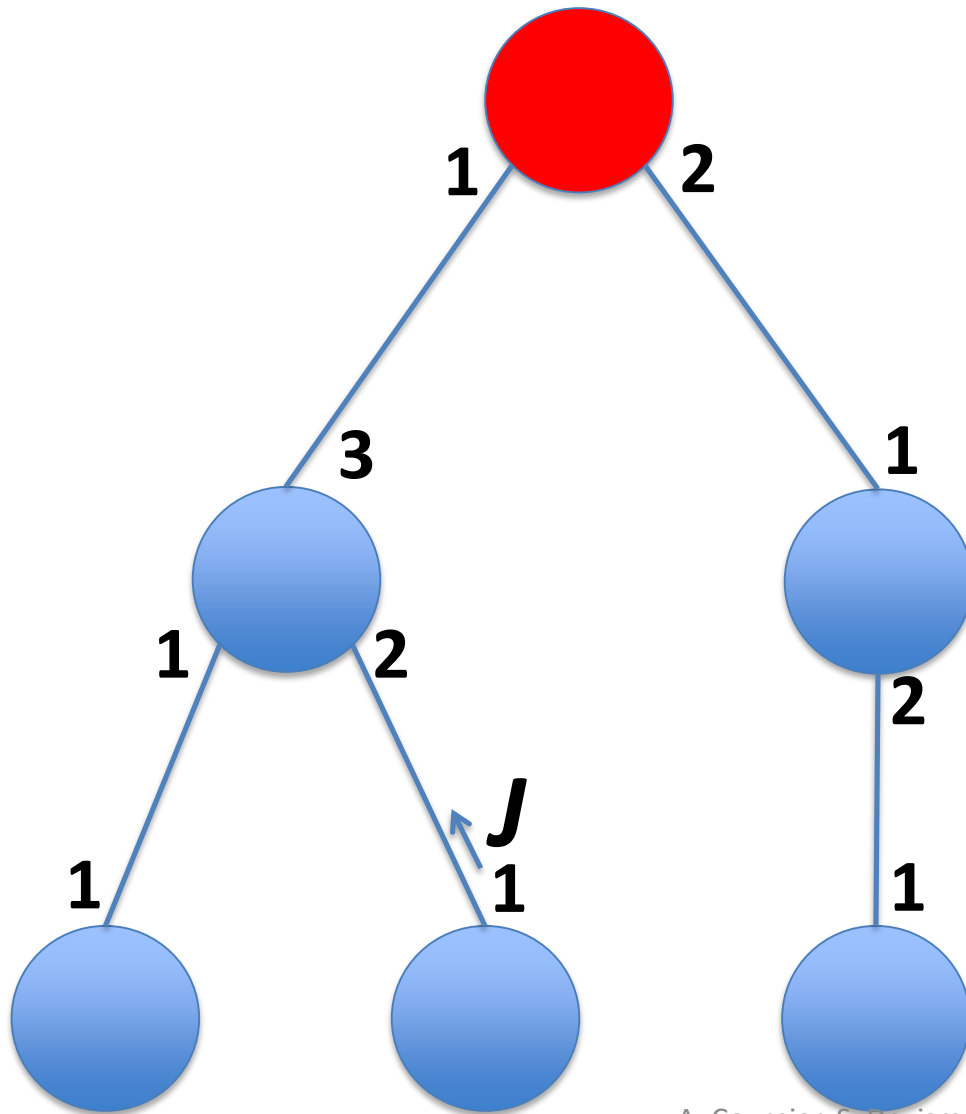
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 3$
- $(1 \bmod 3) + 1 = 2$

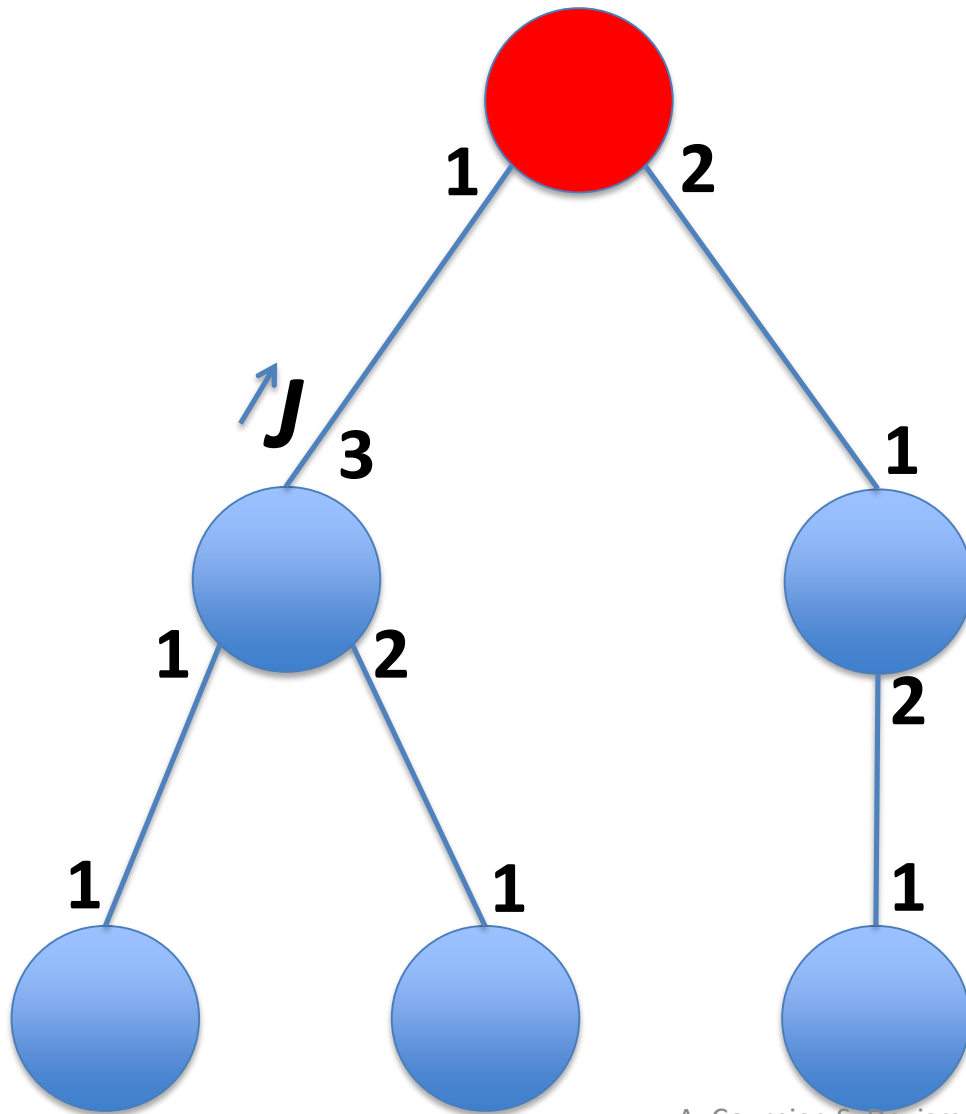
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 1$
- $(1 \bmod 1) + 1 = 1$

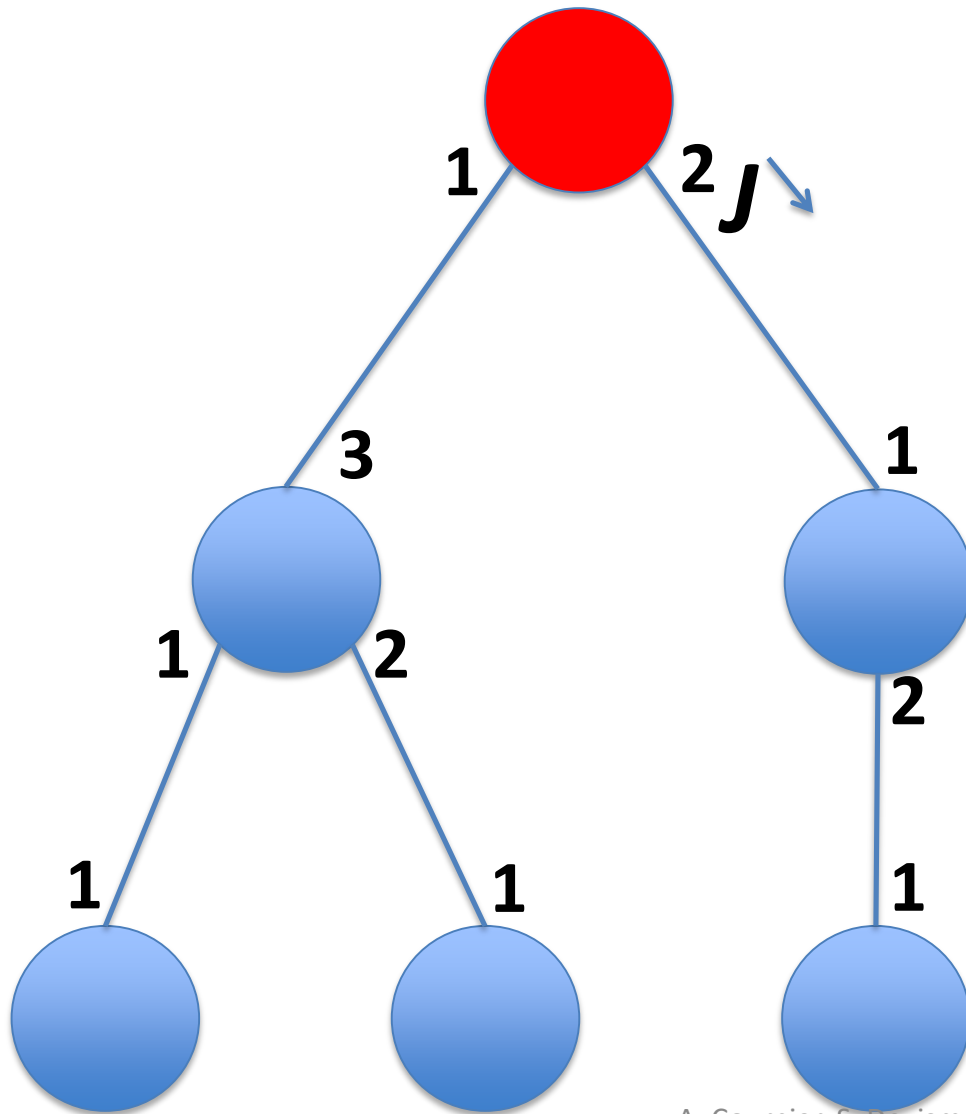
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 3$
- $(2 \bmod 3) + 1 = 3$

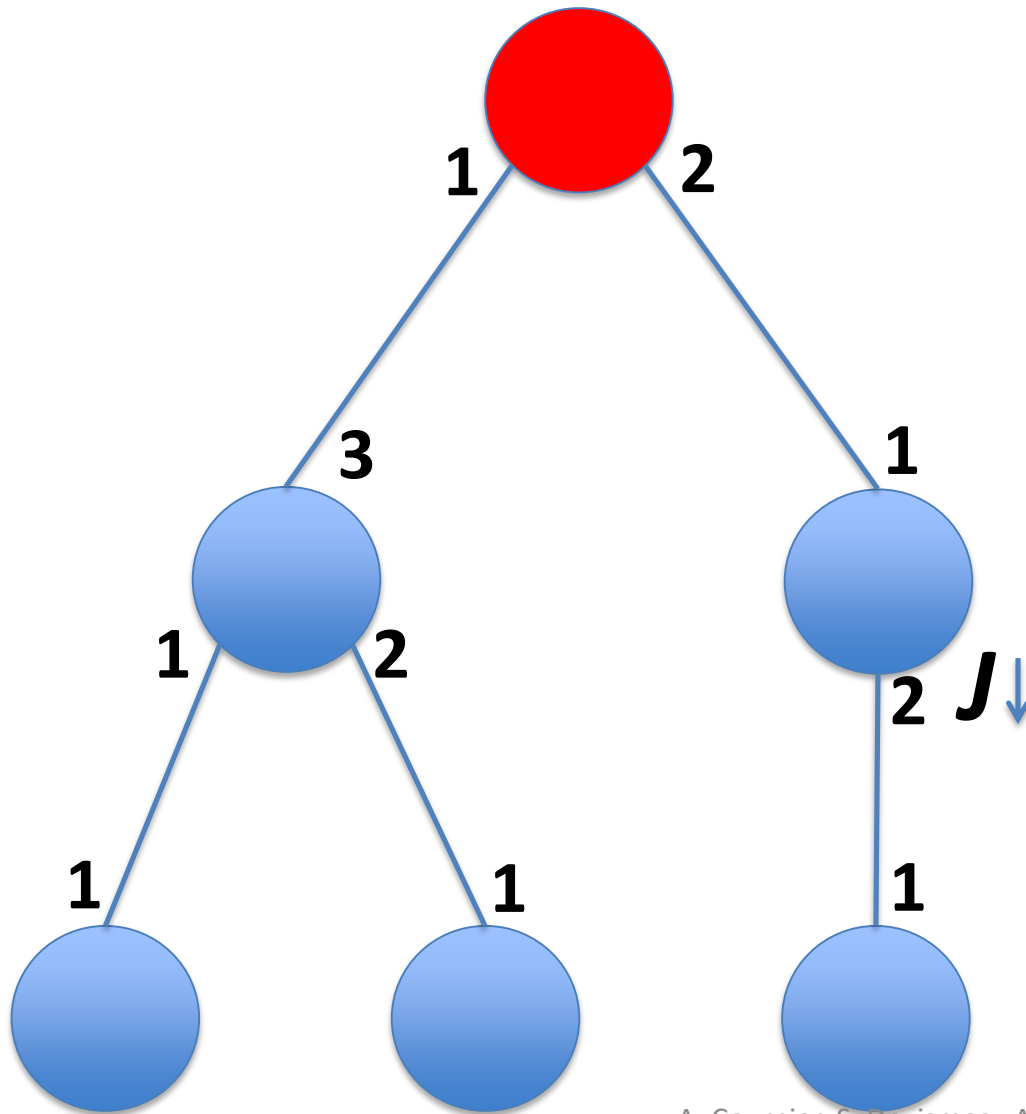
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 2$
- $(1 \bmod 2) + 1 = 2$

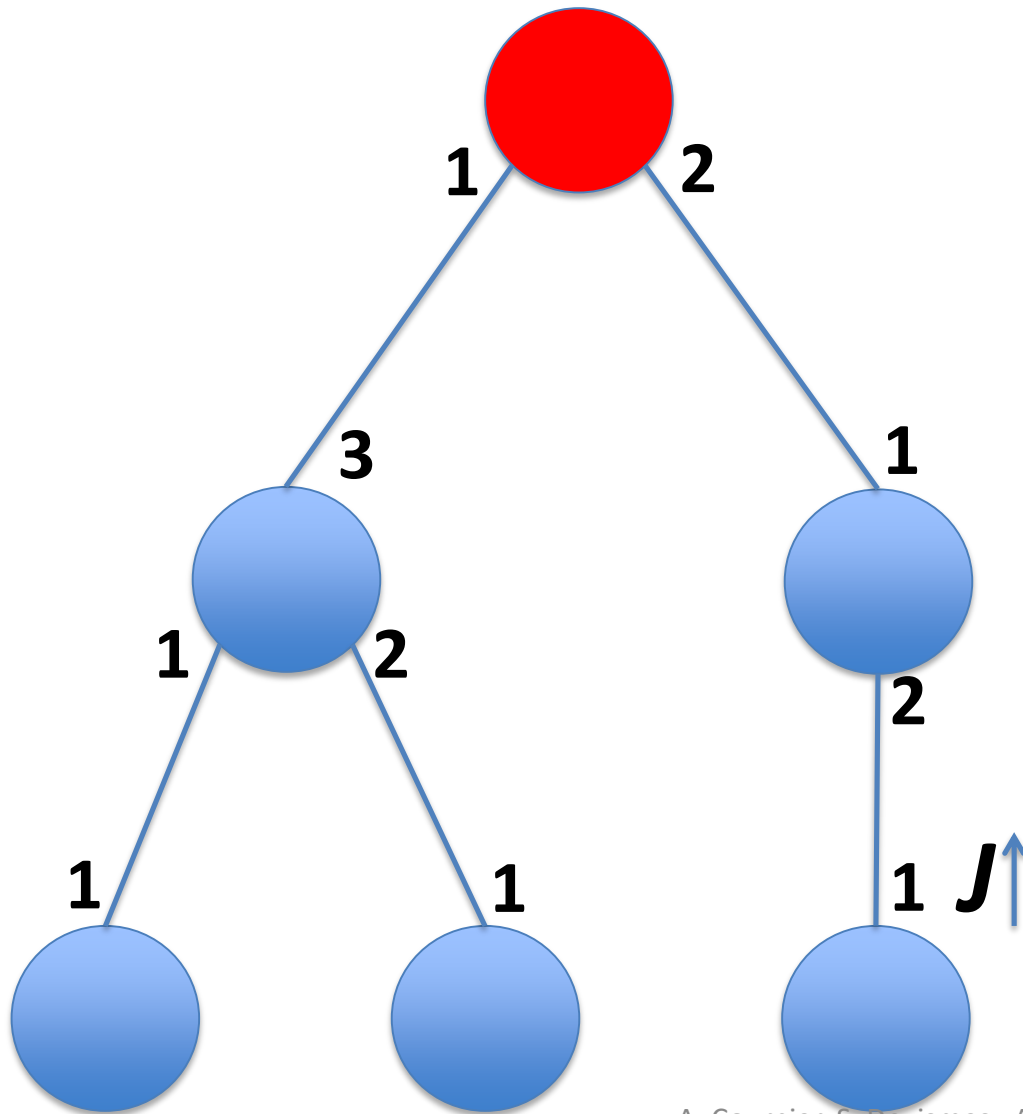
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 2$
- $(1 \bmod 2) + 1 = 2$

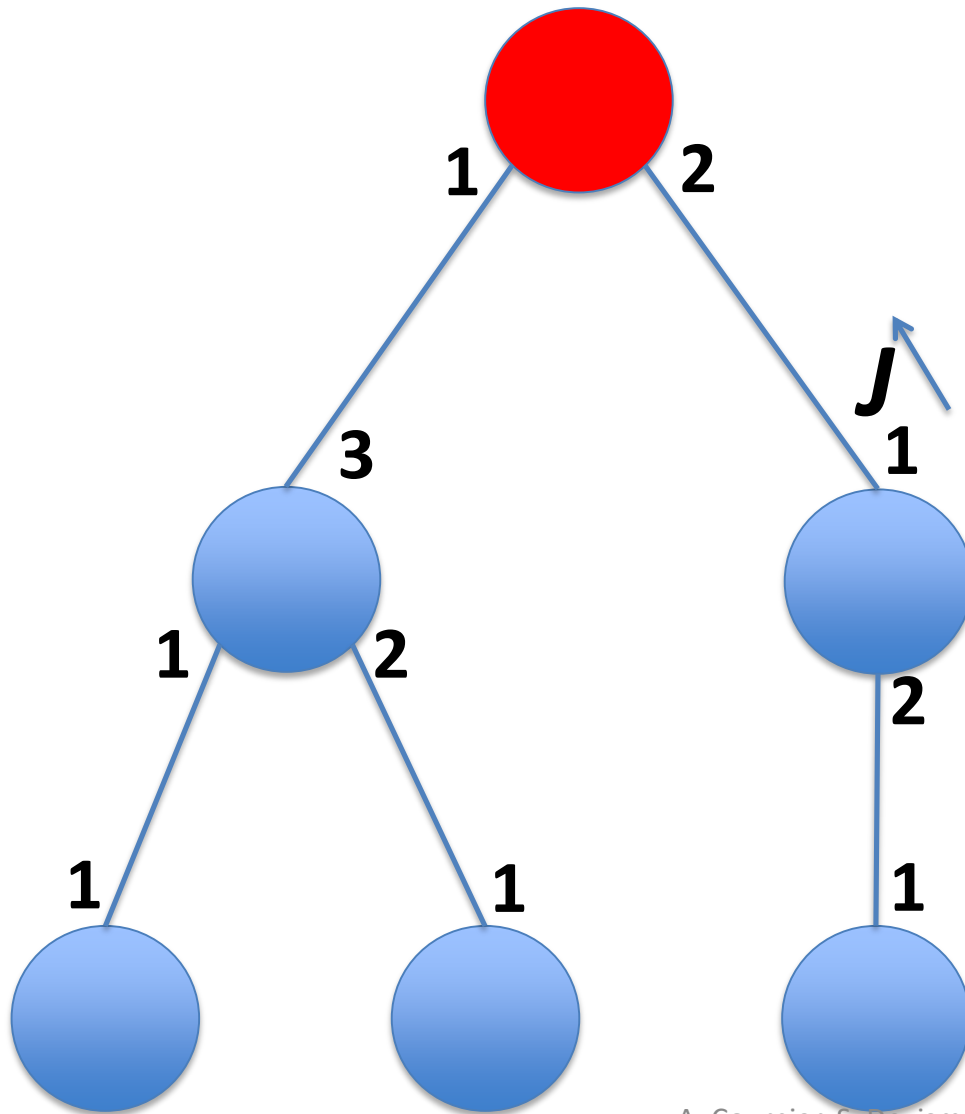
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 1$
- $(1 \bmod 1) + 1 = 1$

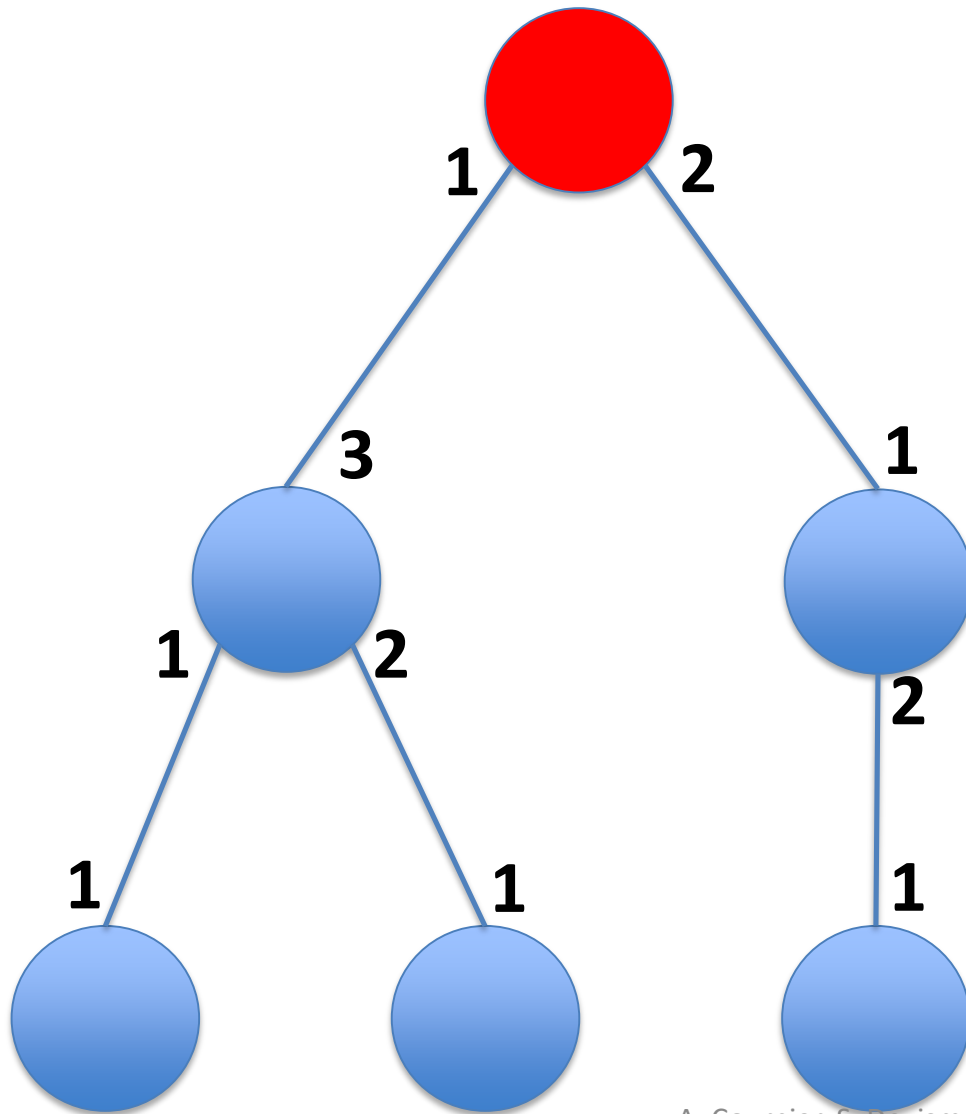
Circulation d'un jeton dans un arbre



- Sur réception du canal i , un non-initiateur renvoie le jeton sur le canal $(i \bmod \delta) + 1$

- Ici $\delta = 2$
- $(2 \bmod 2) + 1 = 1$

Circulation d'un jeton dans un arbre



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

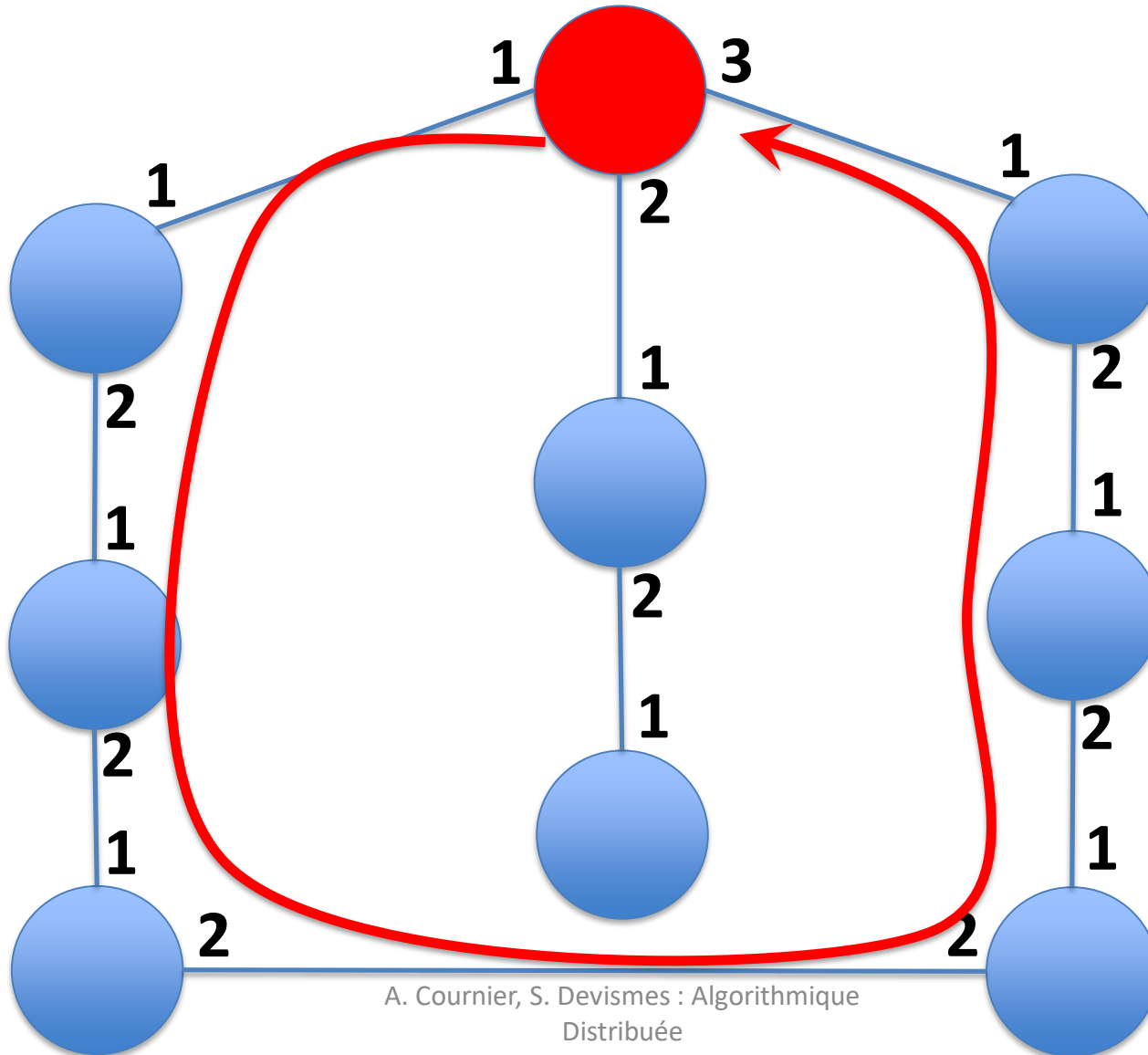
- Ici $\delta = 2$

Circulation d'un jeton dans un réseau quelconque ?

- Est-ce que l'algorithme précédent fonctionne ?

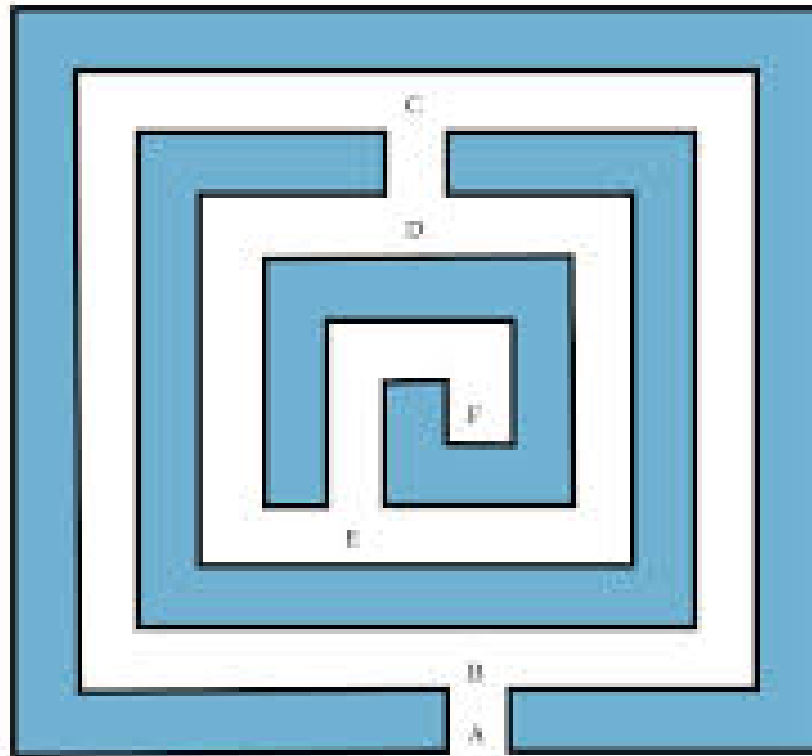
NON !

Exemple



Solution

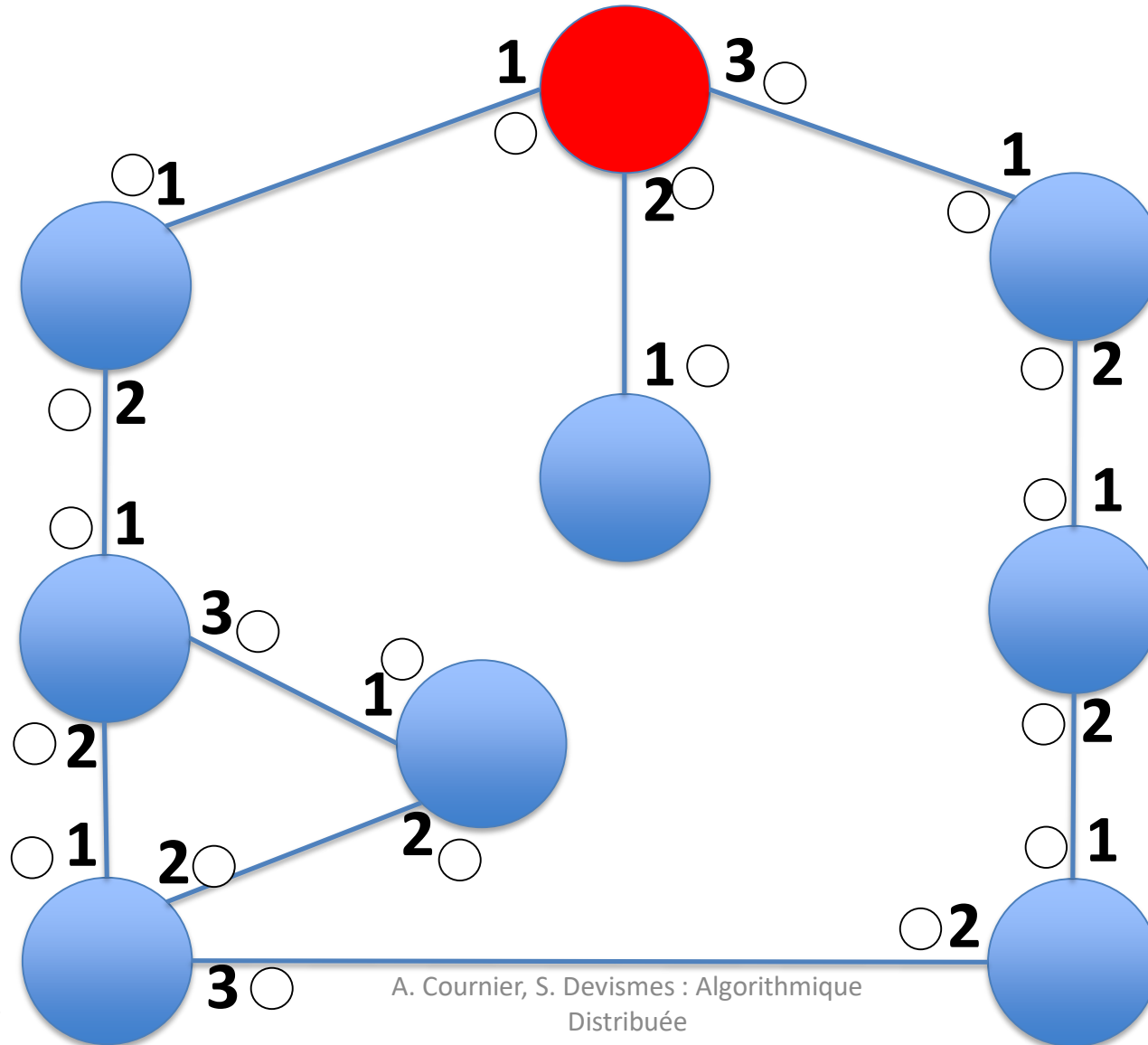
- Algorithme de Tarry (1885)
- Problème de Labyrinthe
- « Ne reprendre l'**allée initiale** qui a conduit à un **carrefour** pour la première fois que lorsqu'on ne peut pas faire autrement »
- Sommets = carrefours
- Liens = allées entre les carrefours



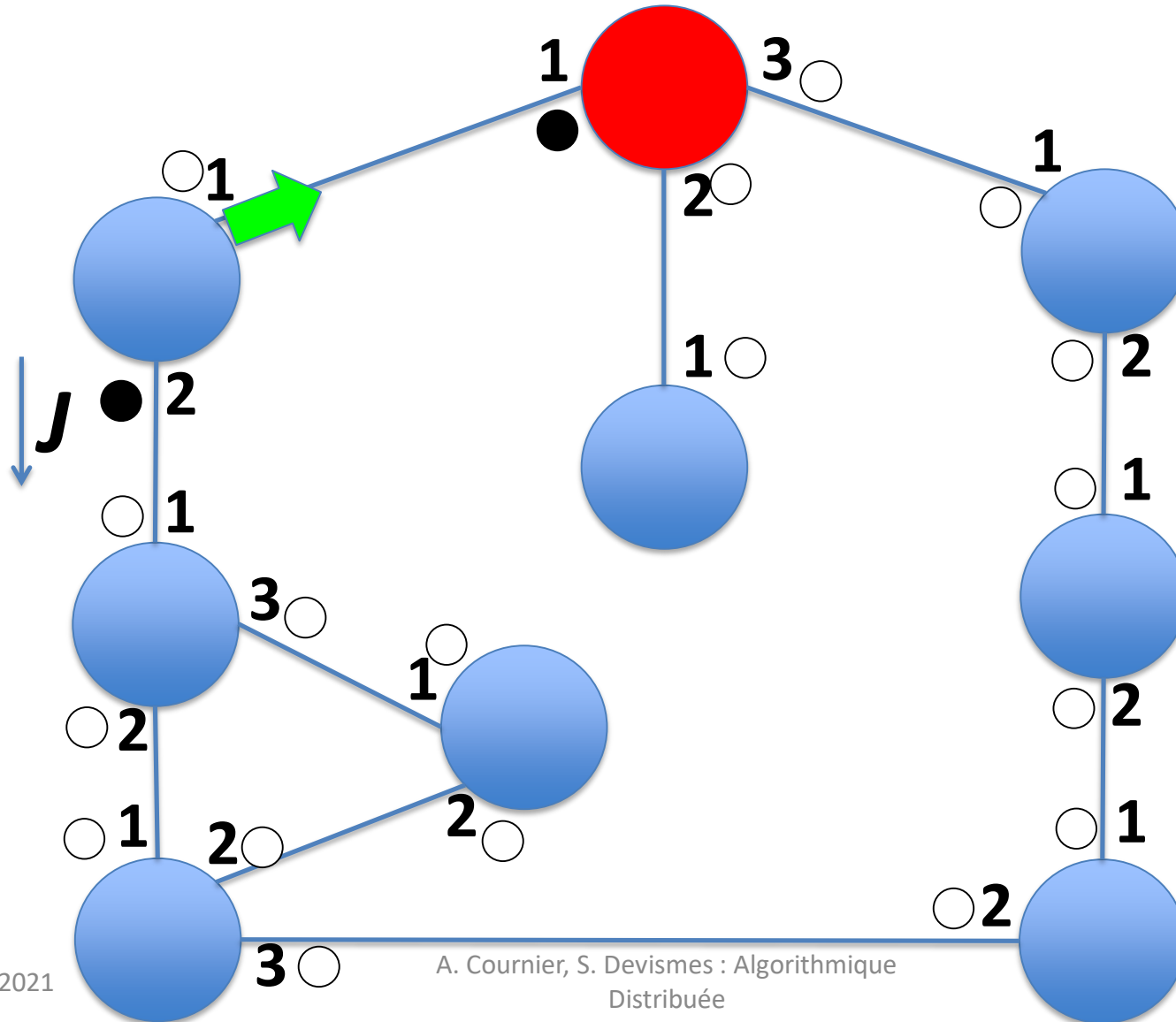
Variables

- Pour chaque processus
 - Un pointeur **Père** $\in \{\top, \perp\} \cup \{1 \dots \delta\}$ initialisé à \perp
 - Lorsque l'initiateur démarre spontanément **Père** := \top
 - Un tableau de Booléen **Visite**[$1 \dots \delta$], initialement toutes les cases sont à faux.

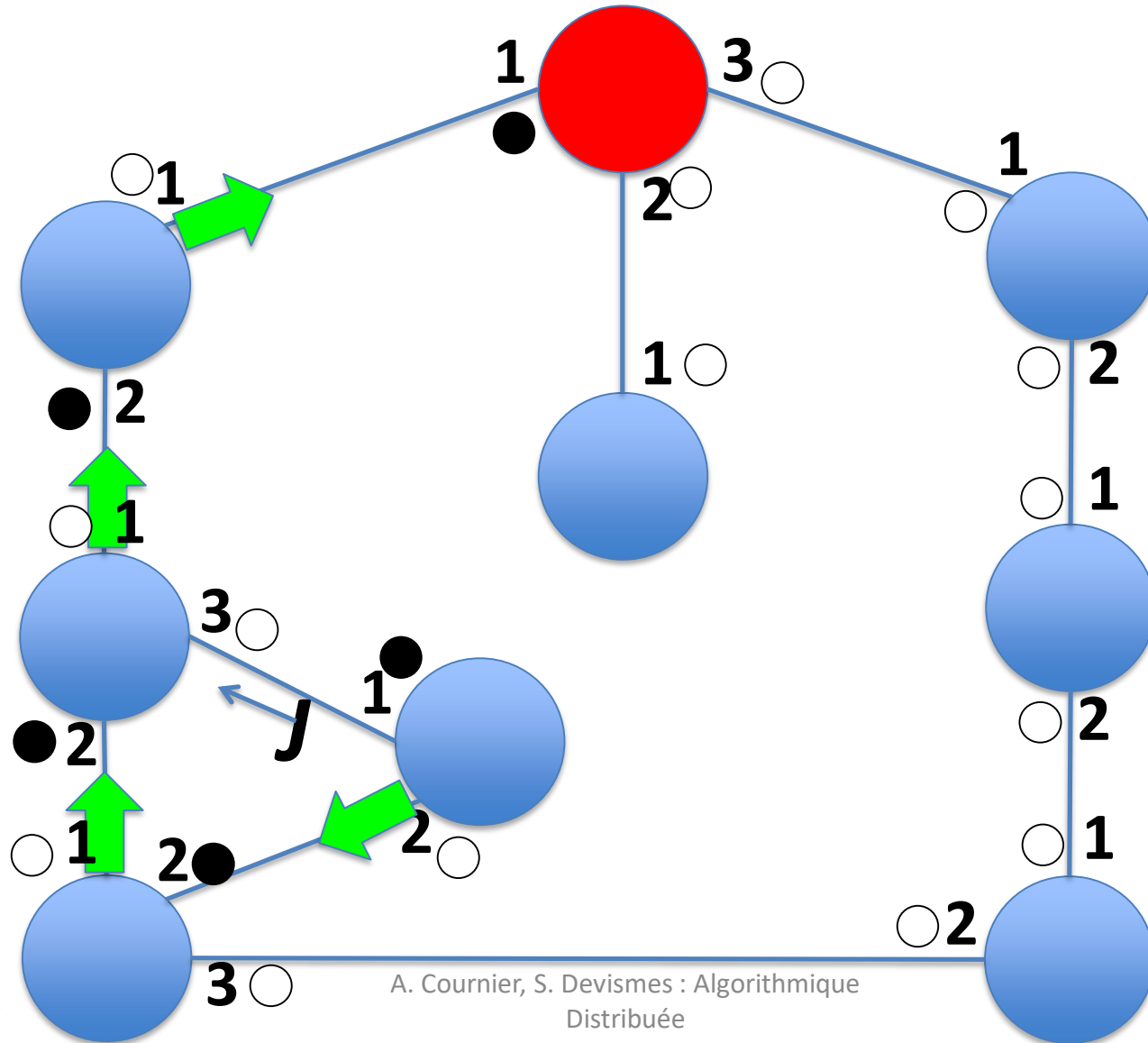
Exemple



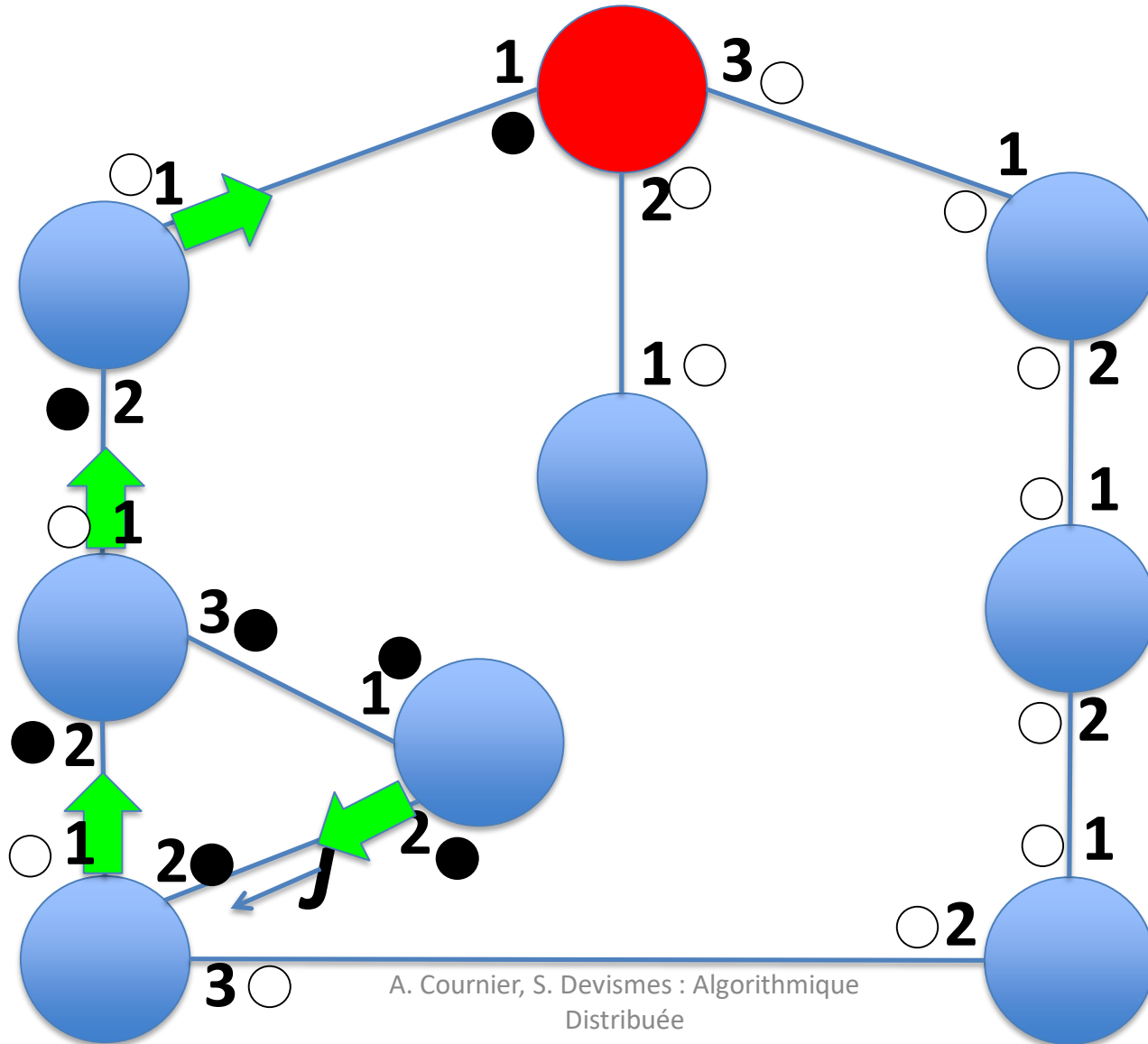
Exemple



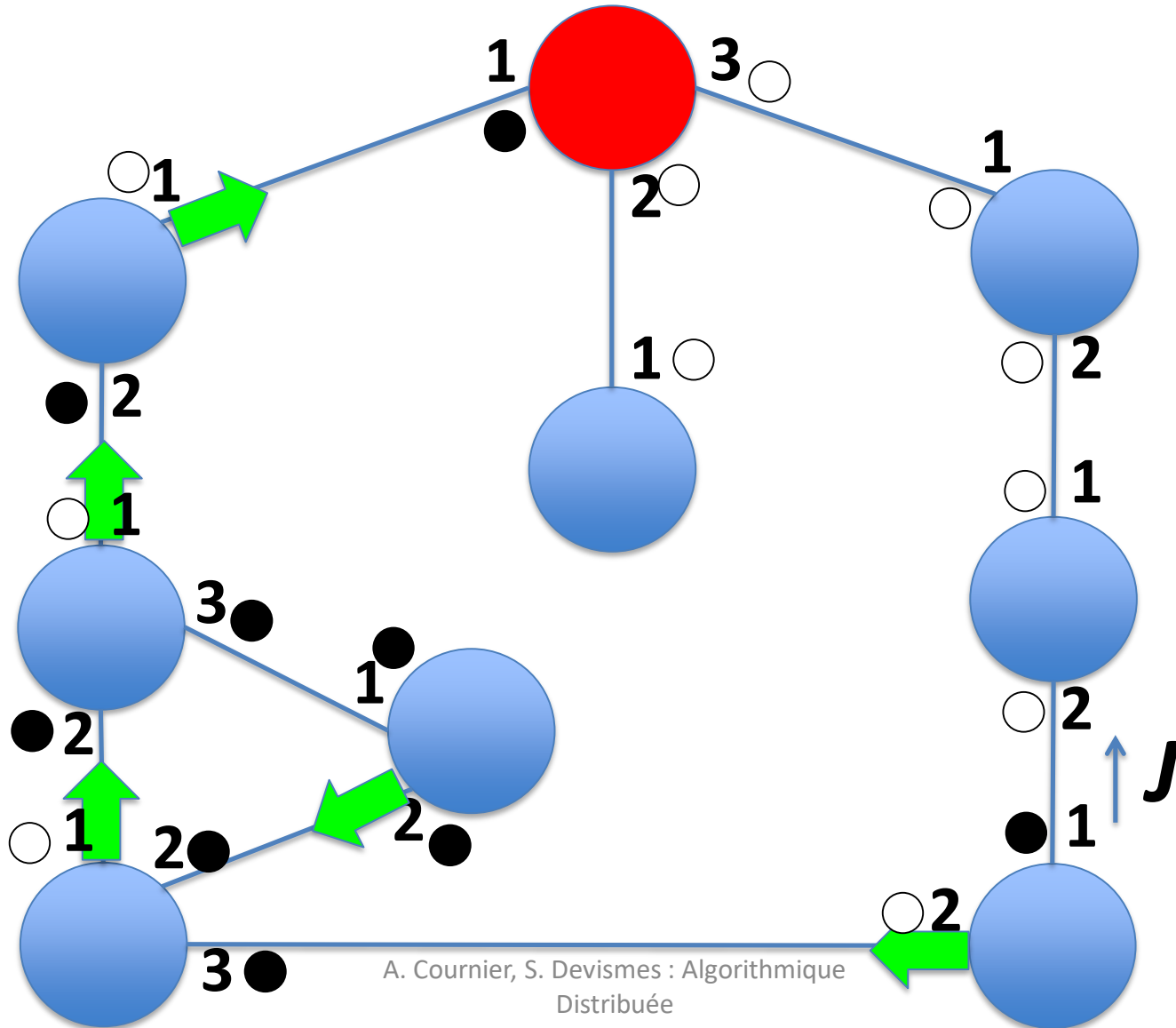
Exemple



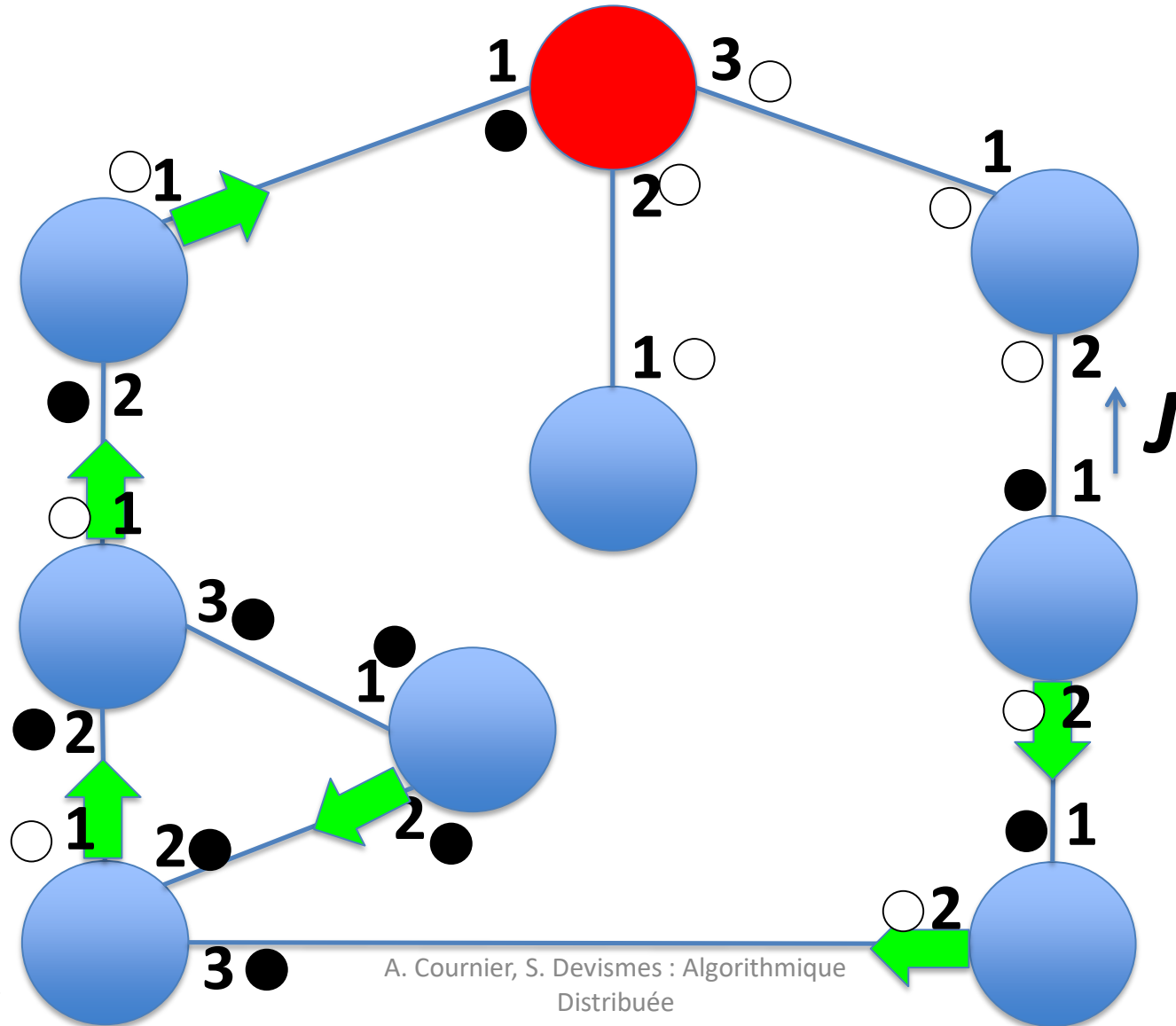
Exemple



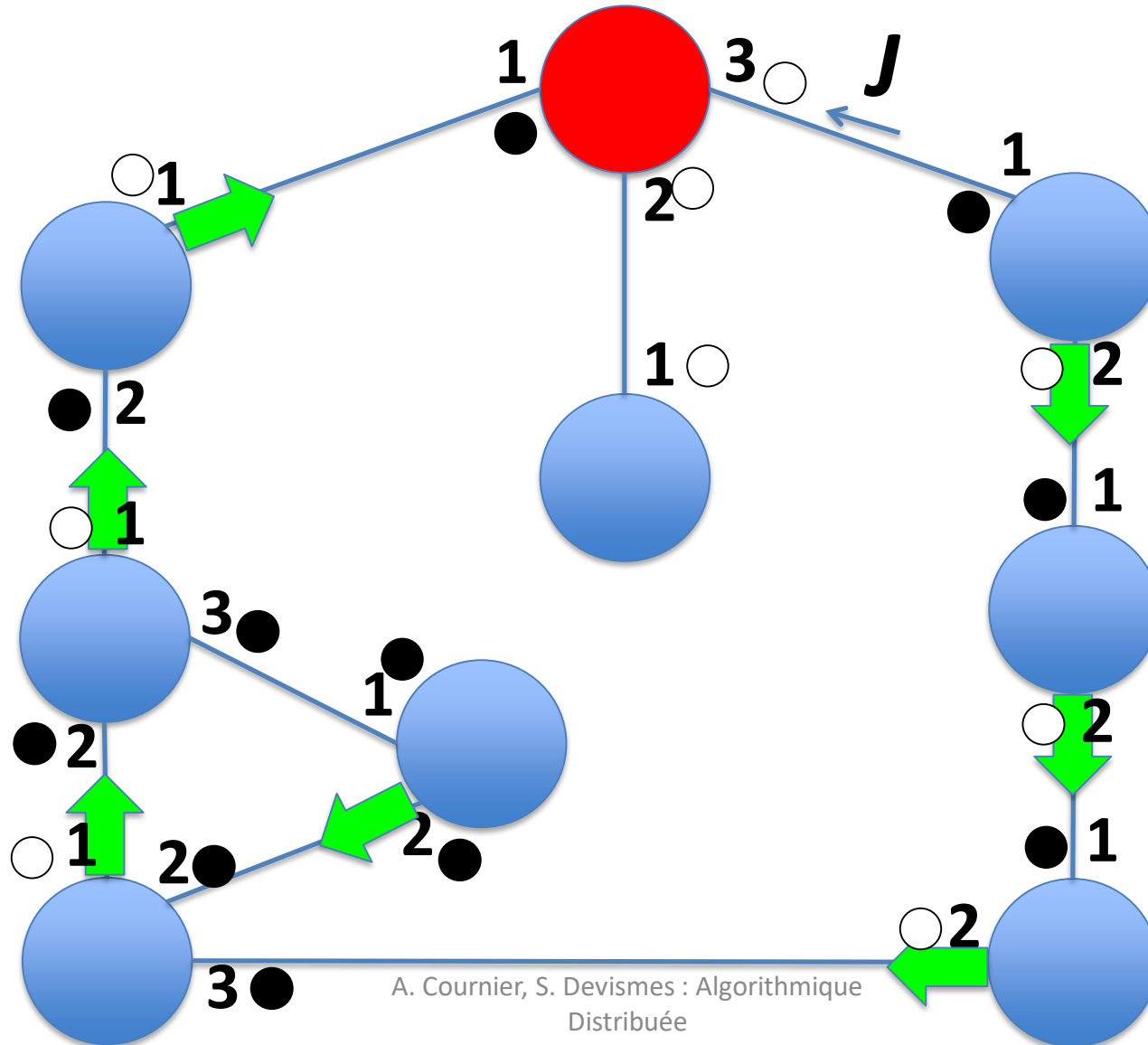
Exemple



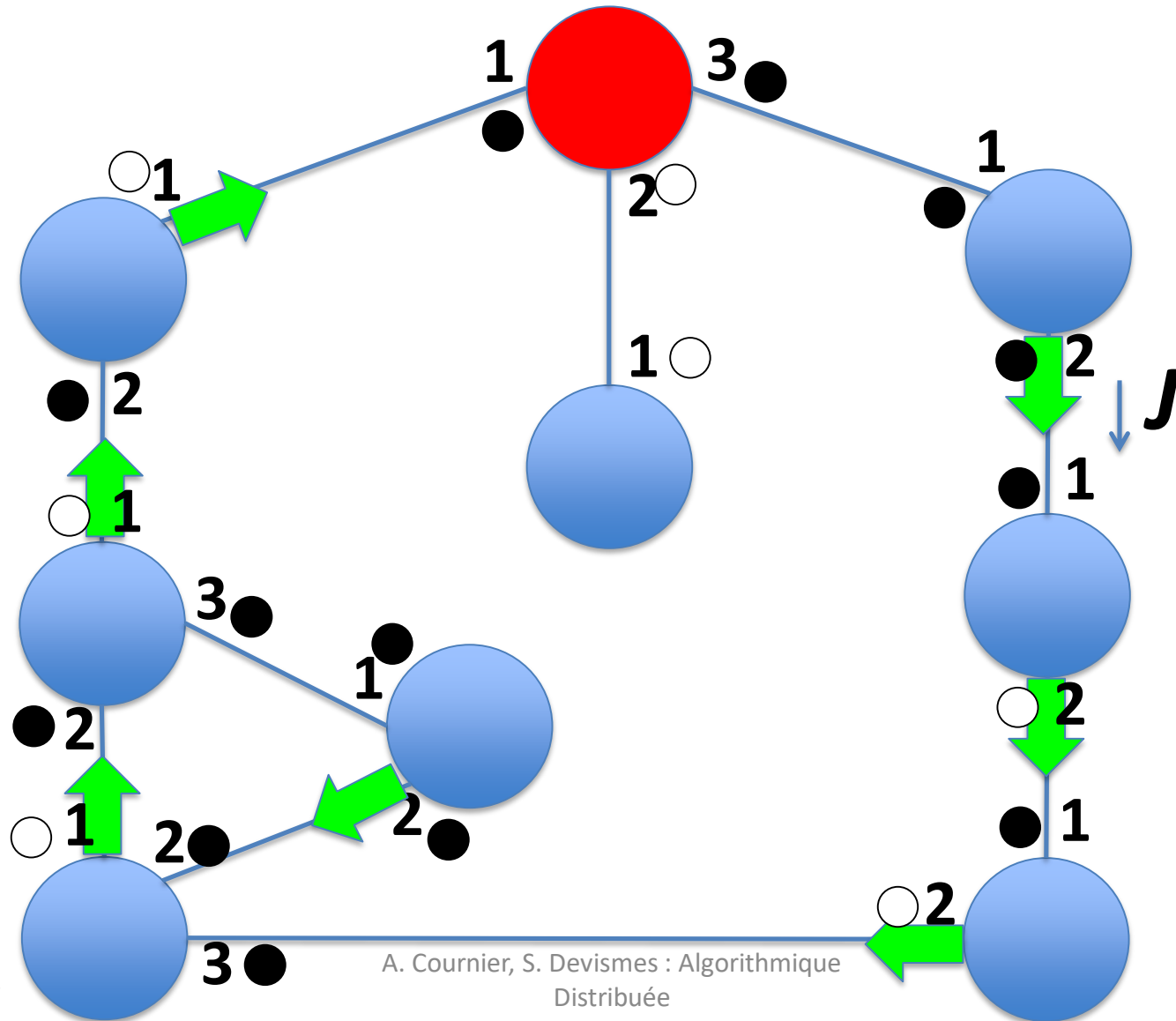
Exemple



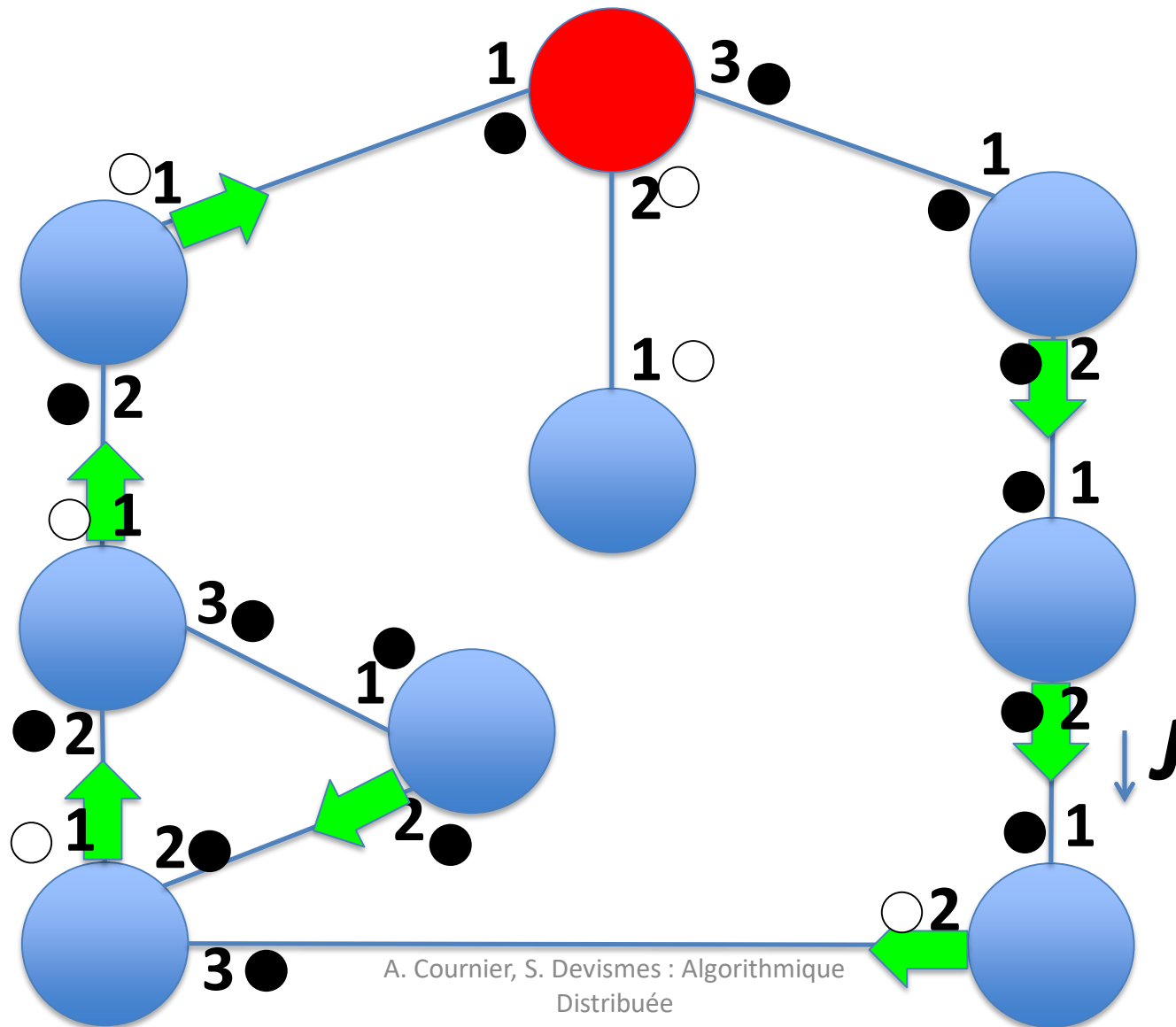
Exemple



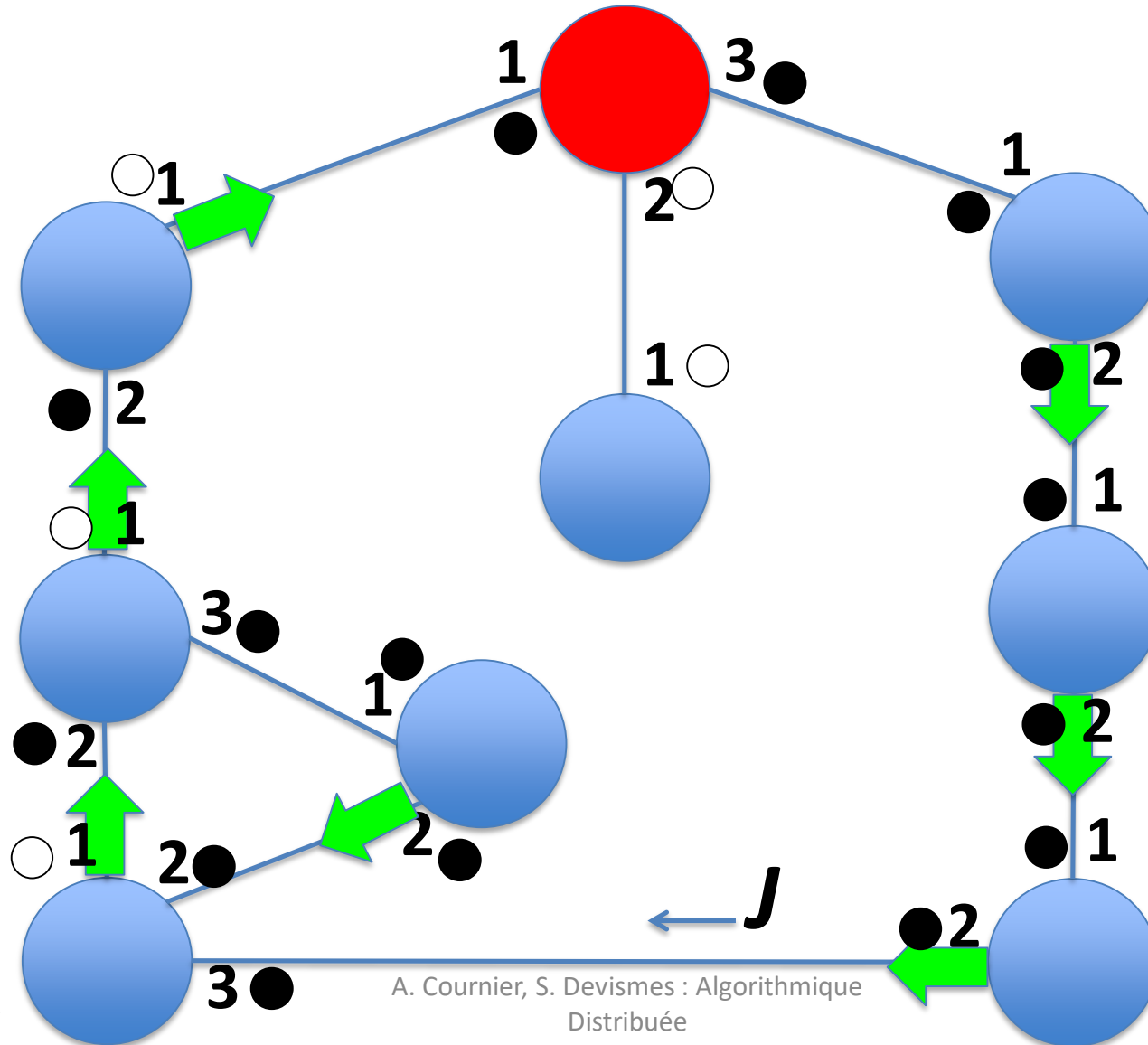
Exemple



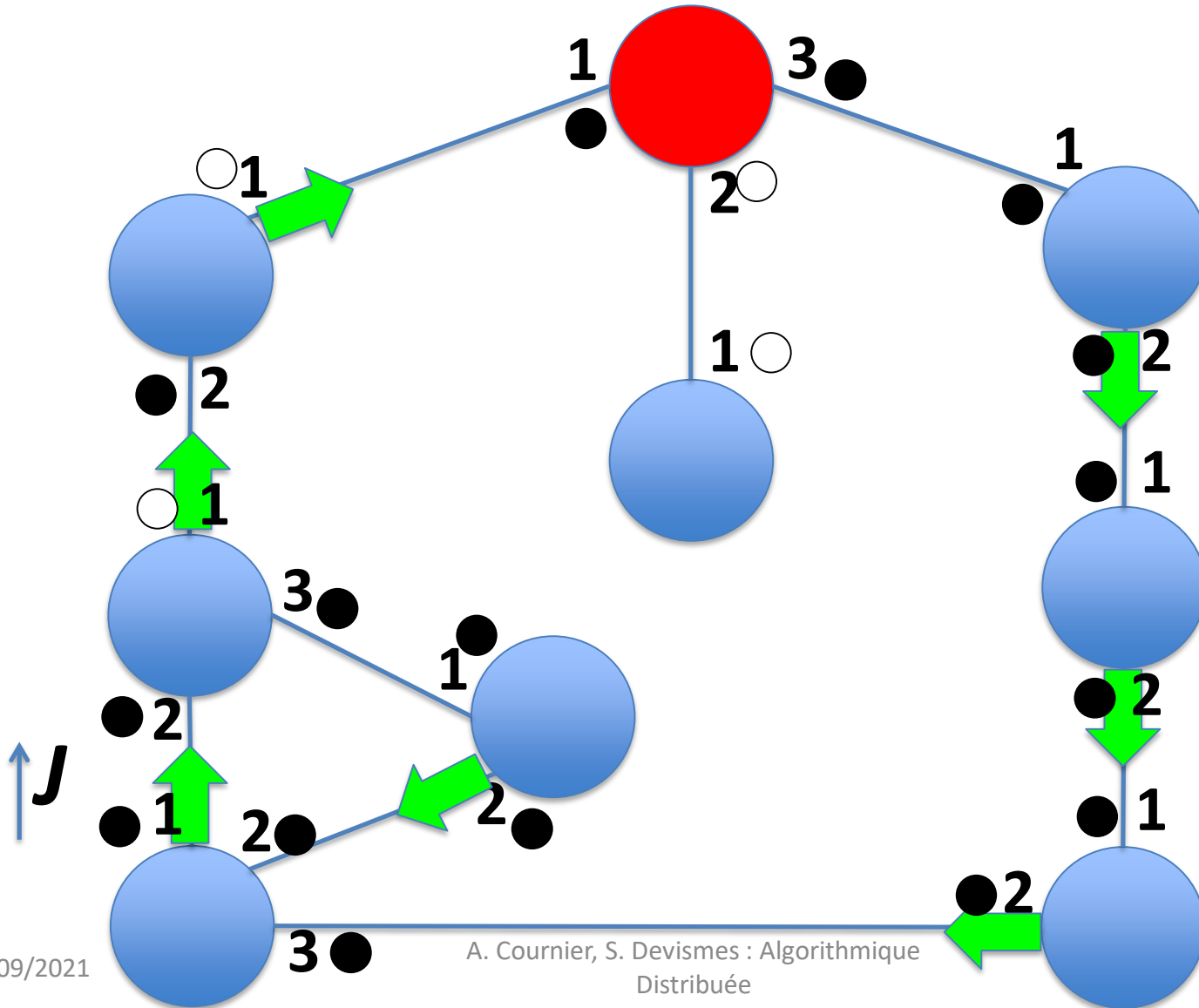
Exemple



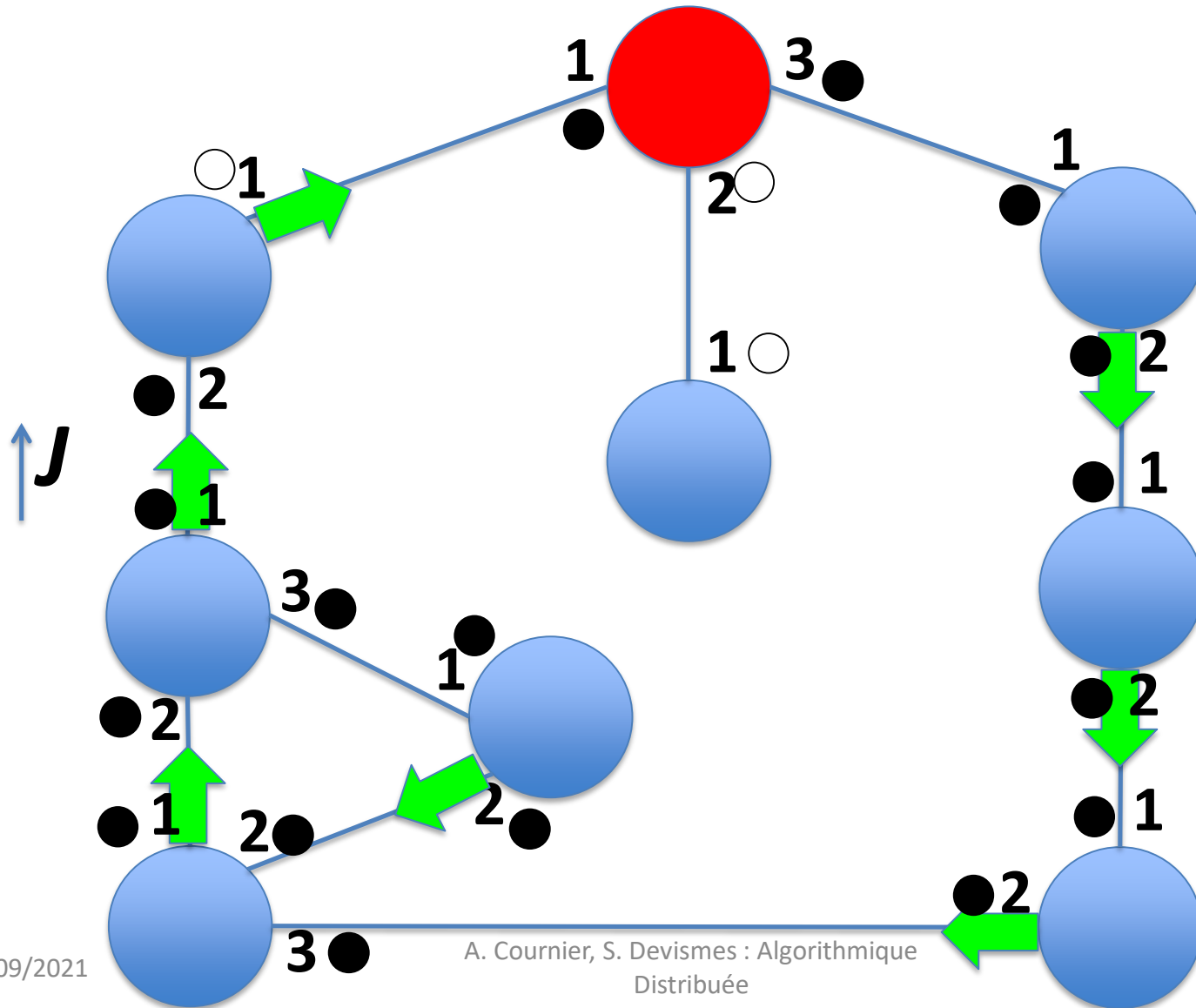
Exemple



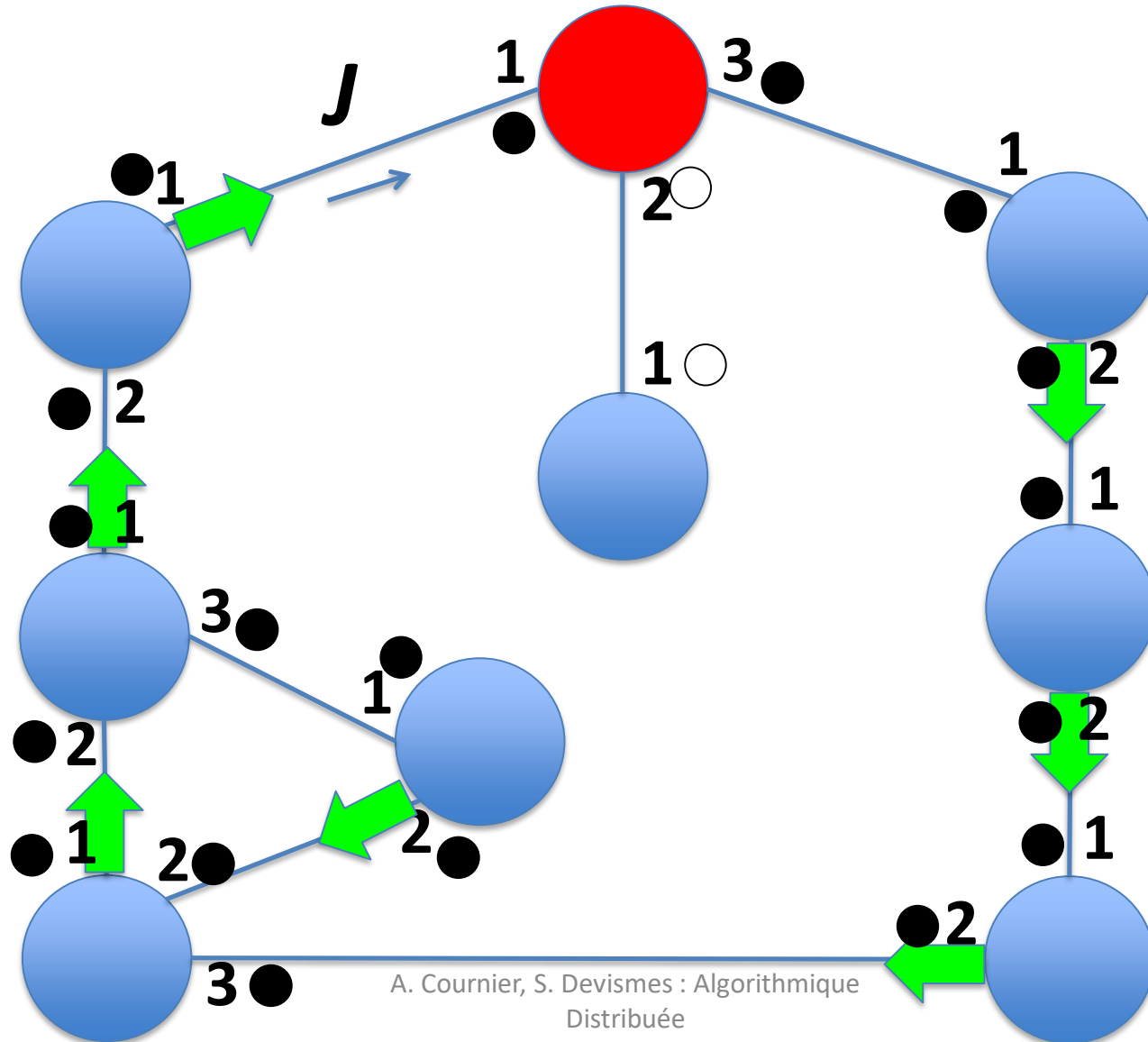
Exemple



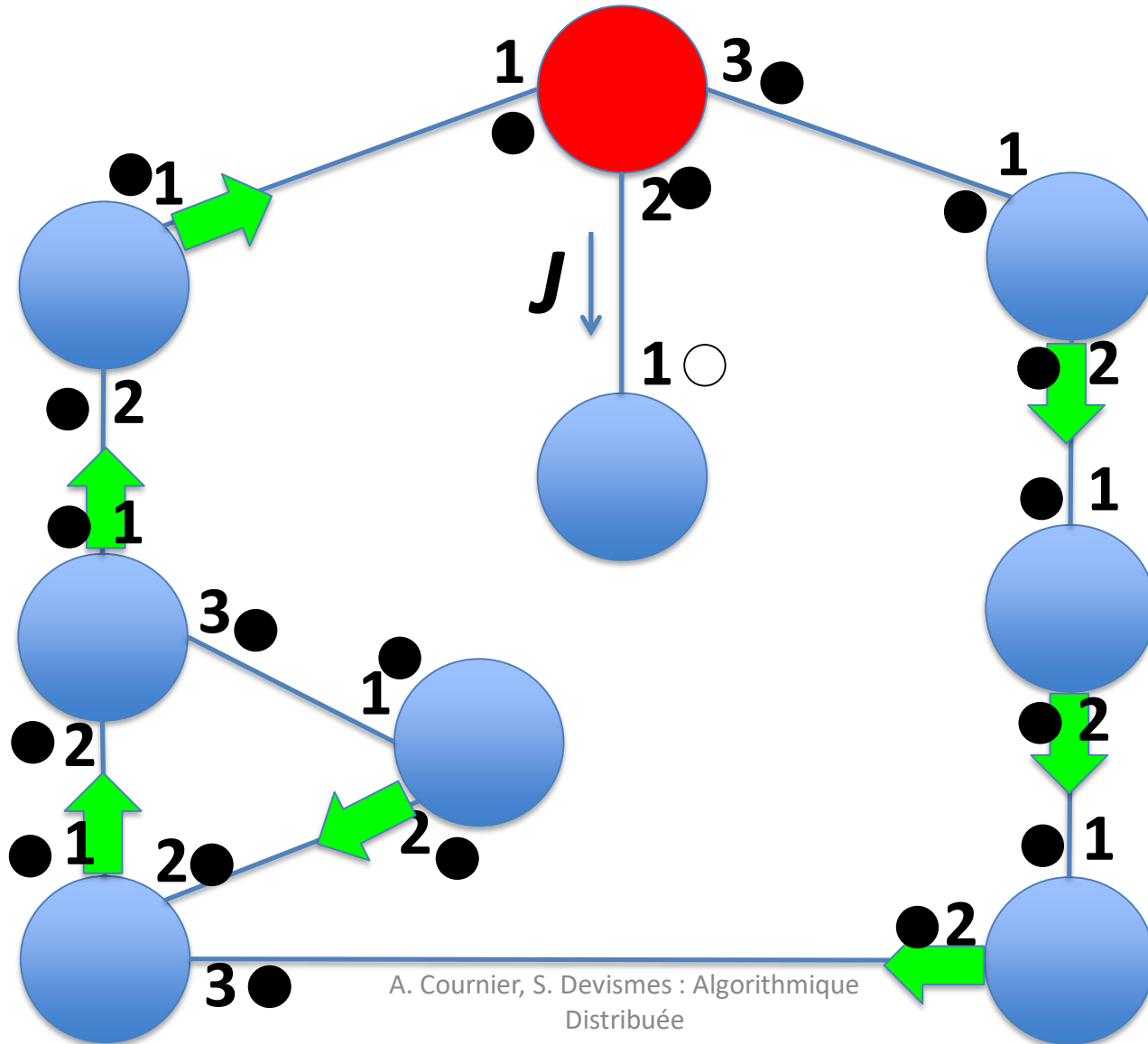
Exemple



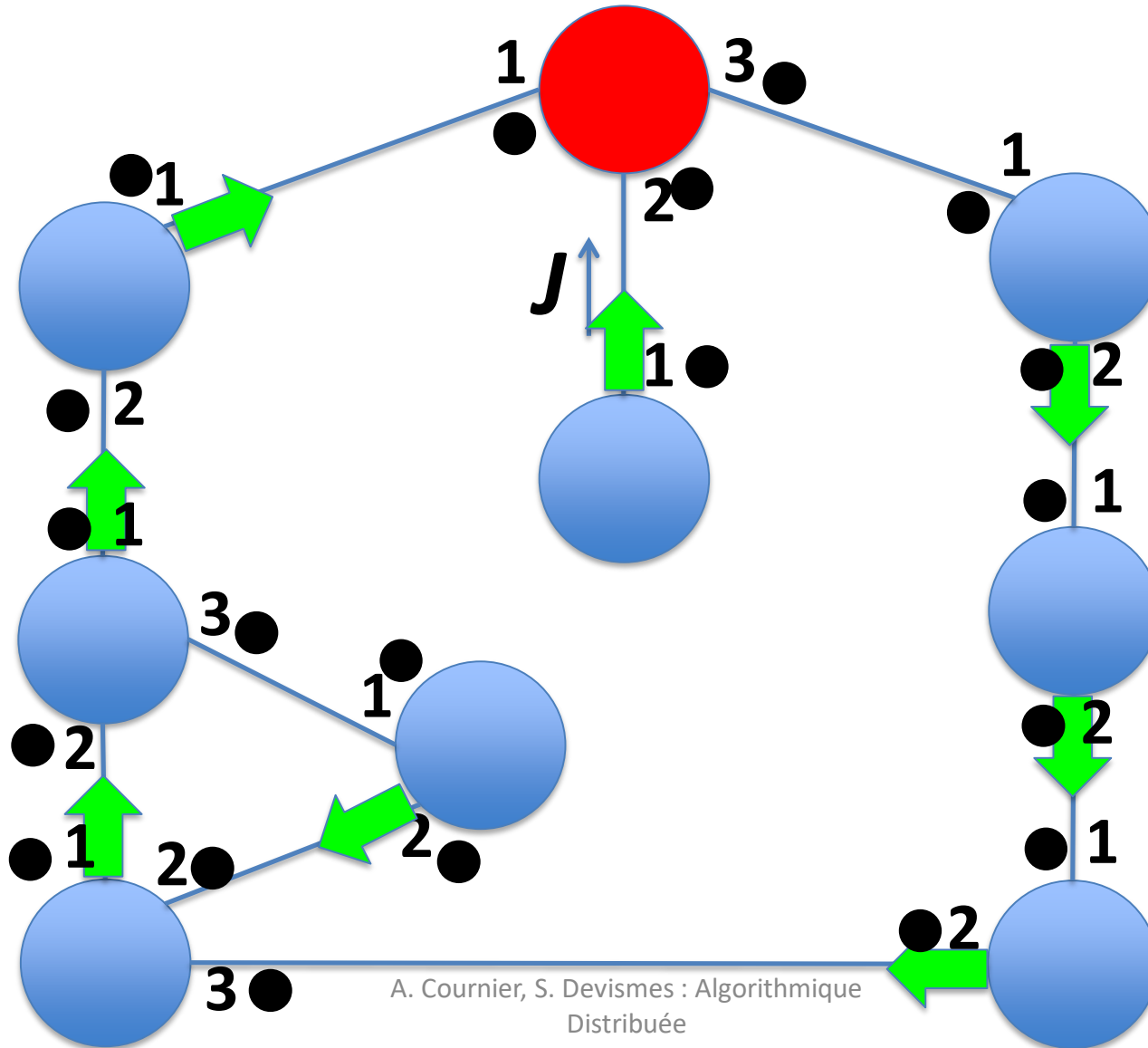
Exemple



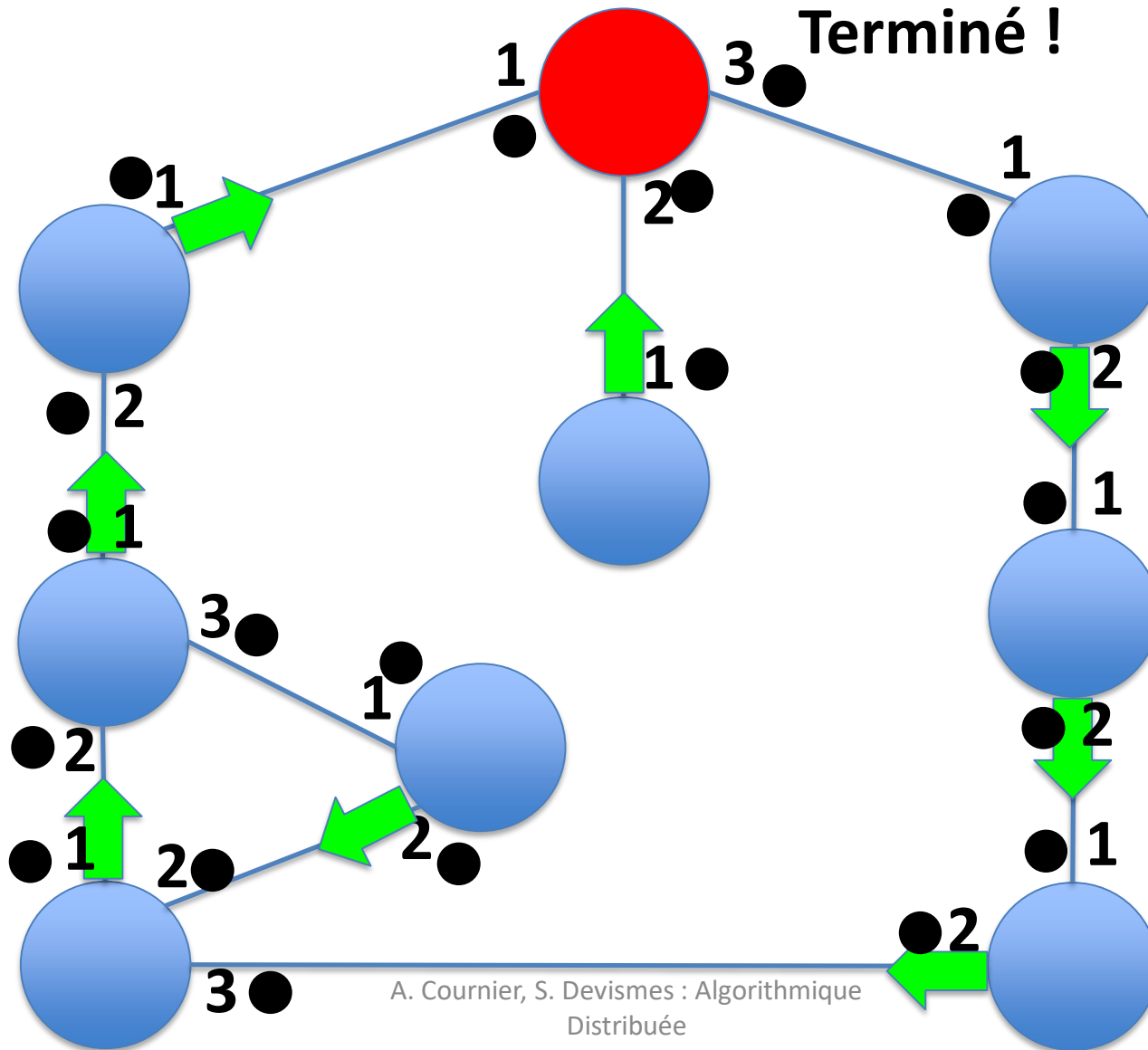
Exemple



Exemple



Exemple



La suite en TD !