Séance 1 — Introduction aux systèmes répartis

Définitions, modèles, graphes, premiers algorithmes

M1 Informatique

Université — Cours de Systèmes Répartis

6 octobre 2025

Plan du cours

Plan

Définition d'un système réparti

Définition

Un **système réparti** est :

- un ensemble de processus (informatiques) autonomes ;
- géographiquement ou logiquement séparés ;
- qui coopèrent pour atteindre un **objectif** commun ;
- sans partager de **mémoire centrale** ni disposer d'une **horloge globale** ;
- et **échangent des messages** en utilisant un système de communication (**asynchrone**)

Définition d'un système réparti

Caractéristiques essentielles

- Chaque processus a son propre état local et évolue de manière indépendante.
- La communication se fait exclusivement par l'envoi et la réception de messages.
- Le système ne possède pas de point de contrôle unique, ce qui implique des défis de synchronisation et de cohérence.

Principale question

Comment réaliser une tâche commune dans un système réparti ?

Exemples de tâches communes dans un système réparti

Tâches classiques

- **Diffusion de messages** : garantir que tous les processus reçoivent la même information.
- **Accord / consensus** : décider collectivement d'une même valeur.
- **Exclusion mutuelle** : un seul processus à la fois accède à une ressource critique.
- **Élection d'un coordinateur** : choisir un « chef » pour organiser certaines décisions.
- **Ordonnancement des messages** : livraison en respectant la causalité ou un ordre total.
- **Réplication cohérente** : maintenir des copies identiques d'une donnée malgré les pannes.

Consensus distribué : l'exemple de la blockchain

Définition

La **blockchain** est un registre distribué où des nœuds doivent **se mettre d'accord** sur l'ordre des transactions et l'état global. C'est un cas emblématique de **consensus dans un système réparti**.

Objectifs du consensus

- Accord sur une même valeur (bloc/état) malgré délais réseau et pannes.
- Tolérance aux défaillances (crash, partitions réseau).
- Ordonnancement total des transactions (chaînage de blocs).
- mécanisme pour élection du prochain producteur d'un bloc

Élection de coordinateur : l'exemple d'un protocole de routage

Contexte

Dans les grands réseaux (entreprise, opérateur, Internet), plusieurs routeurs échangent des informations pour déterminer les meilleurs chemins de transmission. Certains protocoles de routage désignent un **coordinateur**, pour organiser les communications

- Dans le protocole **OSPF** (Open Shortest Path First), un **Designated Router (DR)** et un **Backup Designated Router (BDR)** sont élus automatiquement.
- L'élection se fait selon une priorité configurée et l'adresse IP la plus élevée en cas d'égalité.
- Si le DR tombe, le BDR prend automatiquement le relais sans interrompre la diffusion.

Rôle du routeur désigné

- Collecter les informations de topologie des autres routeurs du même segment.
- Diffuser les mises à jour de routes de manière ordonnée pour réduire le trafic.
- Centraliser temporairement les décisions de convergence.

Exclusion mutuelle : l'exemple d'un fichier partagé

Définition

L'exclusion mutuelle garantit qu'une seule machine accède à une **ressource critique** à la fois (ex. un fichier partagé ou une base de données).

Problématique

- En environnement distribué, il n'existe pas de mémoire ou verrou centralisé.
- Les processus doivent coopérer via des messages pour obtenir le droit d'accès.
- Il faut éviter blocages, famine et incohérence d'accès.

Exemples concrets

- Réservation d'un fichier dans un **système de versionnage** (Git, SVN).
- Accès exclusif à une ressource dans un **système de fichiers distribué** (NFS, HDFS).

Intérêt des systèmes répartis

Pourquoi les utiliser ?

- **Tolérance aux pannes** : en cas de panne locale d'un nombre réduits de noeuds, le système global continue de fonctionner.
- **Évolutivité / élasticité** : on peut ajouter à la volée des ressources (machines, serveurs) pour gérer plus de charge, ou au contraire en enlever.
- **Performance** : paralléliser les calculs complexes / coûteux, ou rapprocher les services web des utilisateurs permet de réduire la latence.
- **Disponibilité** : les services peuvent être accessibles en continu grâce à la réplication et la répartition de charge.
- **Partage de ressources** : possibilité de mutualiser les données, ou leur stockage entre plusieurs utilisateurs (exemple des réseaux de torrents).
- **Robustesse et flexibilité** : le système s'adapte aux variations du trafic et aux défaillances.

Algorithme réparti / distribué

Définition d'un algorithme distribué

Description d'un calcul à effectuer par des entités autonomes, ayant pour but de solutionner un même problème.

- Chaque entité possède une partie de l'information
- Chaque entité exécute le même algorithme que les autres enttiés du Système réparti
- Chaque entité du système réparti peut communiquer uniquement avec ses voisins.

Absence de contrôle centralisé

Dans un **algorithme distribué**, il n'existe pas de site ou de processus jouant le rôle d'arbitre global. Le système fonctionne sans autorité centrale : les décisions émergent de la coopération entre les sites.

Contrôle centralisé vs Algorithme réparti / distribué

Contrôle centralisé

- Un site particulier joue le rôle de **chef arbitre**.
- Il prend toutes les décisions et collecte des informations de tout le système.
- Le reste des sites agit comme des exécutants.
- Risque : **point de défaillance unique**, surcharge, manque d'évolutivité.

Contrôle réparti / distribué

- Aucune entité n'a de rôle privilégié.
- Tous les sites sont **égaux en droit et en devoir**.
- Chaque site agit à partir d'une **connaissance locale** et partielle du système.
- La coordination se fait via des échanges de messages entre voisins.

→ *Le contrôle est réellement distribué : chaque site contribue à la décision globale sans autorité centrale.*
→ *Certains algorithmes distribués élisent d'abord un chef qui peut faire office de contrôle centralisé.*

Défis majeurs des systèmes répartis

Problématiques fondamentales

- **Absence d'horloge globale** : difficile d'ordonner les événements et de définir un état global cohérent.
- **Pannes et tolérance aux fautes** : crash de machines, pertes de messages.
- **Synchronisation** : coordonner plusieurs processus sans mémoire partagée ni temps global.
- **Communication incertaine** : délais variables, messages dupliqués ou perdus.
- **Sécurité** : authentification, intégrité et confidentialité des échanges entre nœuds potentiellement distants.
- **Scalabilité** : maintenir performance et cohérence quand le nombre de nœuds augmente.
- potentielle présence de **Comportements byzantins**

Comportements byzantins dans les systèmes répartis

Définition

Un processus adopte un **comportement byzantin** lorsqu'il se comporte de manière arbitraire ou malveillante : il peut envoyer des informations incorrectes, contradictoires ou agir différemment vis-à-vis de chaque pair.

Origine du terme

Le terme vient du **problème des généraux byzantins** (Leslie Lamport, 1982) : Des généraux de l'armée byzantine campent autour d'une cité ennemie. Ils ne peuvent communiquer qu'à l'aide de messagers et doivent établir un plan de bataille commun, faute de quoi la défaite sera inévitable.
Cependant un certain nombre de ces généraux peuvent se révéler être des traîtres, qui essayeront donc de semer la confusion parmi les autres.
Le problème est donc de trouver un algorithme pour s'assurer que les généraux loyaux arrivent tout de même à se mettre d'accord sur un plan de bataille.

Comportements byzantins dans les systèmes répartis

Exemples

- Un nœud envoie une valeur x à un processus et une valeur différente y à un autre.
- Un processus altère ou falsifie les messages reçus avant de les retransmettre.
- Un acteur malveillant tente de perturber le consensus (ex. attaques sur une blockchain).
- Silence sélectif : un processus choisit de répondre à certains nœuds mais pas à d'autres, créant une vision incohérente du système.
- Fabrication de messages inexistants : un nœud invente de faux messages « reçus » pour influencer le consensus ou induire les autres en erreur.

Défis associés

- Détection et tolérance aux comportements arbitraires.
- Besoin d'algorithmes de **consensus byzantin**.
- Coût en communication et en ressources souvent plus élevé.

Plan

Modèle de communication

Plusieurs modèles de communication peuvent être considérés

- **Synchrone** : bornes connues sur temps de calcul et délais réseau.
- **Asynchrone** : pas de borne connue ; messages peuvent être arbitrairement retardés.
- **Partiellement synchrone** : bornes existent mais inconnues / valides après un temps.

Impact : possibilités de consensus, détection de pannes, timeouts, preuves.

Types de fautes

- **Défaillance de processus** : crash / arrêt.
- **Défaillances réseau** : perte d'envoi/réception de message, perte d'un lien réseau.
- **Comportement bizantins** (arbitraire/malveillants), information altérée

Modéliser les fautes oriente la conception et les garanties.

Modélisation d'un système réparti par un graphe

Principe de modélisation

Un **système réparti** peut être représenté sous forme de **graphe** (orienté, non orienté, pondéré), afin de décrire les entités de calcul et leurs relations de communication.

Nœuds du graphe

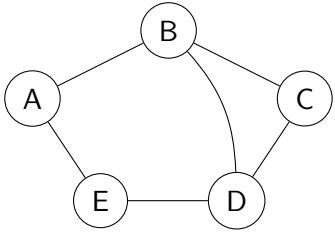
- Représentent les **entités de calcul** : nœuds, sites, processus, processeurs, machines, etc.
- Chaque nœud peut avoir un **identifiant unique** : P1, P2, ... ou A, B, C...
- Attention : ce sont les **processus** qui exécutent des calculs et échangent des messages.

Arêtes du graphe

- Représentent les **liens de communication** entre entités de calcul.
- Ces liens peuvent être appelés : **canaux de communication**, **liens**, ou **arêtes**.
- Une arête (Pi, Pj) signifie qu'un échange de messages est possible entre Pi et Pj.

Graphe de communication (non orienté)

- Nœuds V : **processus** ; Arêtes E : **canaux**.
- Graphe **non orienté** ; éventuellement **pondéré** (latence, capacité).
- **Dynamique** : arêtes pouvant apparaître / disparaître.

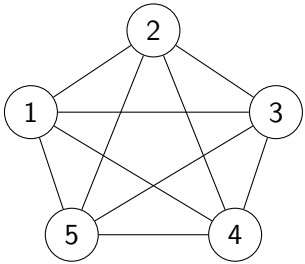


Un graphe spécial : le **graphe complet**

Définition

Un **graphe complet** K_n relie chaque paire de nœuds par une arête.

- Connectivité maximale, diamètre $= 1$.
- Coût en liens $= \frac{n(n-1)}{2}$ (non orienté).
- Très robuste mais **peu réaliste** à grande échelle.



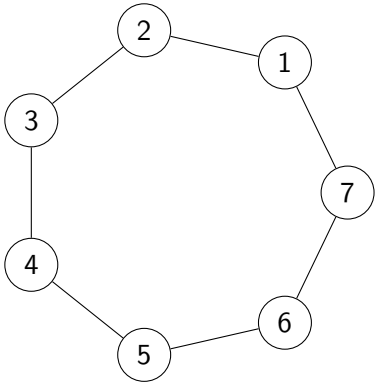
K_5

Topologie en **anneau** (7 nœuds) — vision locale

Hypothèses locales

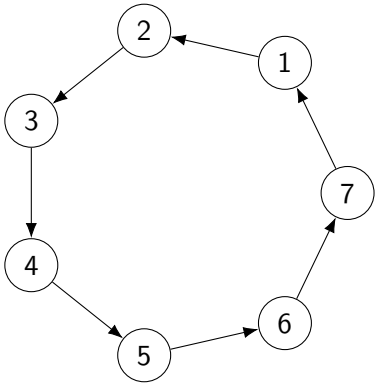
- Chaque nœud connaît uniquement son **prédécesseur** et son **successeur**.
- Un nœud **ne connaît pas** le nombre total de nœuds du réseau.
- Les décisions (diffusion, élection, etc.) s'appuient sur des **échanges locaux**.

Conséquences : algorithmes itératifs, messages circulants, détection de terminaison non triviale.



Topologie en **anneau orienté** (7 nœuds)

- Les canaux suivent une **direction unique** (sens horaire).
- Utile pour des **algorithmes par jeton** (token) ou des tours d'ordonnancement.
- Même **vision locale** : chaque nœud voit seulement son prédécesseur/successeur.



Remarques sur la mise en œuvre d'un algorithme réparti

Influence du modèle

- Le comportement d'un algorithme réparti dépend fortement du **graphe de communication** (topologie, connectivité) et des **hypothèses sur les canaux** : FIFO, fiabilité, synchronisme, etc.
- Ces hypothèses conditionnent la validité et les performances des algorithmes.

Liens physiques vs liens logiciels

- Faire la distinction entre :
  - les **liens physiques** (câbles, routeurs, réseau IP),
  - et les **liens logiciels** (connexions logiques entre processus).
- Le graphe modélise les liens **logiques**, pas nécessairement les connexions matérielles.

Déploiement d'un système réparti en TP

Création du graphe

- Le graphe sera construit à l'aide de **connexions sockets TCP**.
- Avant d'exécuter un algorithme, il faut créer les **processus**, et établir les **canaux de communication**.

→ *L'infrastructure logicielle (processus + liens) doit exister avant tout calcul réparti.*

Procédure

- Le lancement des sites se fait au préalable : mise en relation chaque site avec son voisinage avant qu'il ne commence son calcul / exécution de l'algorithme.
- La mise en place d'un graphe (+ éventuelles initialisations des sites) sera centralisée, i.e. contrôlée par un processus supplémentaire à qui la topologie est donnée.
- Ce processus se termine une fois le graphe d'interconnexion construit

→ *Objectif en TP : une solution **réutiilsable** pour mettre en place un graphe quelconque, possiblement de grande taille*

Plan

Plan

Conception d'un algorithme réparti (1/2)

Principe général

Un algorithme réparti décrit la tâche exécutée par un site P_i, avec $1 \leq i \leq N$. Tous les sites exécutent le **même algorithme**, mais sur leurs **variables locales** respectives.

Convention de notation

- Les variables locales portent le même nom sur tous les sites.
- L'algorithme est **orienté évènement** :
  - il réagit à la **réception de messages** ;
  - il peut **envoyer des messages** à d'autres sites.

Conception d'un algorithme réparti (2/2)

- Structure : pseudo-code classique avec des structures de contrôle (`if`, `while`, etc.)
- Organisation en 3 **blocs** :

Initialisation

- initiation des variables locales
- Mention des données d'entrée et informations connues

Envoi de message

- contenu du message + destinataire(s) ciblé(s)

Réception de message (évenement)

- traitements opérés à la réception d'un message
- un bloc par type de message reçu

Plan

Exemple : un algorithme de diffusion d'informations

Description d'un algorithme réparti (notations, structure, comportements) sans analyse.

Contexte / problématique

- Diffusion dans un **réseau non structuré** : graphe $G = (V, E)$ quelconque de n sites.
- Chaque site S_i, $1 \leq i \leq n$, connaît l'ensemble de ses voisins $N(S_i)$ et sait les distinguer.
- Les communications se font en **mode point à point** et sont **asynchrones**.
- Initialement, un sous-ensemble de sites possède l'information à diffuser.

Principe de l'algorithme

1. Un site disposant d'une information initiale l'envoie à **tous ses voisins**.
2. Lorsqu'il reçoit un message pour la **première fois**, il le diffuse à **l'ensemble de ses voisins sauf l'émetteur du message**.
3. S'il reçoit un message **déjà reçu**, il l'ignore.

Algorithme 1 : Diffusion d'une information (site S_i)

Pseudo-code

Algorithme 1 : Algorithme de diffusion exécuté par le site S_i

Entrée : N : ensemble des voisins du site S_i
Sortie : Diffusion de la donnée d à tous les sites connectés

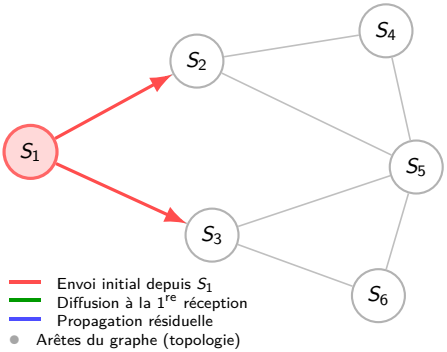
▷ `Initialisation`
`source` : **vrai** si S_i une source de l'information d_i à diffuser, sinon **faux** ;
`setMsg` : ensemble des messages reçus, initialisé à $\varnothing$.

▷ `Procédure diffuser_donnees(d)`
pour chaque $u_k \in N(s_i)$ **faire**
  envoyer le message $\langle d, i \rangle$ à u_k

▷ `À la réception d'un message` $\langle M, emetteur \rangle$ `depuis un voisin` S_j :
  si $\langle M, emetteur \rangle \notin$ `setMsg` **alors**
    ajouter $\langle M, emetteur \rangle$ à `setMsg`
    pour chaque $S_k \in N(S_i) - \{S_j\}$ **faire**
      envoyer $\langle M, emetteur \rangle$ à S_k

Exemple : déroulé sur un réseau)

version simplifiée

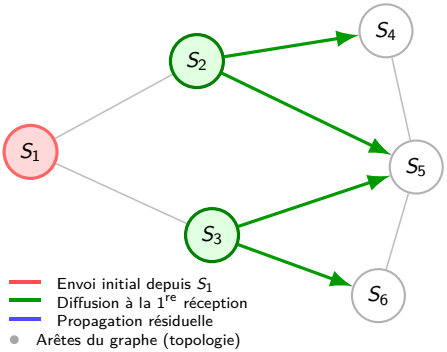


Déroulement (rappel des règles)

1. La source envoie à **tous** ses voisins.
2. À la **première réception**, un site relaye à **tous ses voisins sauf l'émetteur**.
3. Un message **déjà reçu** est ignoré.

Exemple : déroulé sur un réseau)

version simplifiée

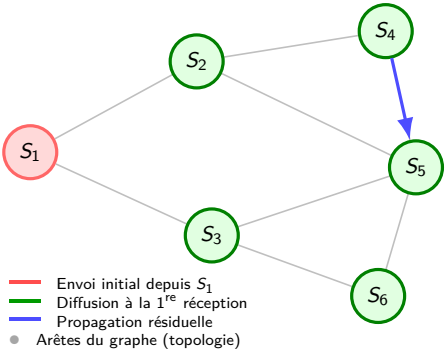


Déroulement (rappel des règles)

1. La source envoie à **tous** ses voisins.
2. À la **première réception**, un site relaye à **tous ses voisins sauf l'émetteur**.
3. Un message **déjà reçu** est ignoré.

Exemple : déroulé sur un réseau)

version simplifiée



Déroulement (rappel des règles)

1. La source envoie à **tous** ses voisins.
2. À la **première réception**, un site relaye à **tous ses voisins sauf l'émetteur**.
3. Un message **déjà reçu** est ignoré.

Exemple : déroulé sur un réseau

Déroulé de l'exemple précédent

- l'exemple précédent donne une fausse sensation de diffusion "par vagues"
- système asynchrone !
- il n'est pas possible de déterminer dans quel ordre les noeuds ont reçu le message : S2 avant ou après S3 ?
- Une nouvelle exécution pourrait donner un ordre totalement différent

Comportement d'un algorithme réparti

Le comportement global d'un système dépend des événements locaux dans le temps

- **Envoi d'un message**
- **Réception d'un message** = arrivée du message dans la file d'entrée du site (niveau communication).
- **Délivrance d'un message** = traitement effectif du message par le programme réparti (niveau applicatif).
- **Calcul interne** = temps de traitement interne, algorithmique interne ...

Entre la réception et la délivrance d'un message, plusieurs scénarios peuvent se produire :

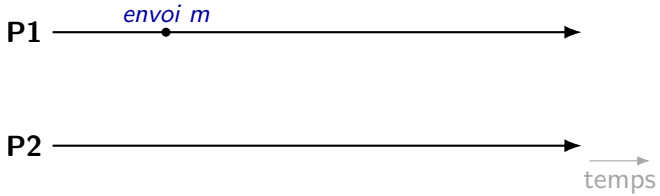
- réordonnancement des messages ;
- files d'attente internes ;
- traitements différés (ex. causal delivery) ;
- garanties de cohérence (FIFO, causalité, etc.).

Réception vs Délivrance d'un message

Distinction dans le comportement d'un algorithme réparti

Définitions essentielles

- **Réception :** arrivée physique du message dans la couche communication du site destinataire. Le message est placé dans une *file d'attente locale*.
- **Délivrance :** prise en compte effective du message par le processus applicatif — déclenche une action locale (mise à jour, émission, calcul...).

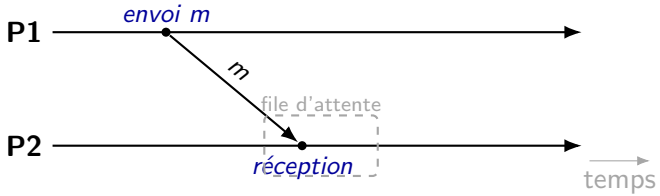


Réception vs Délivrance d'un message

Distinction dans le comportement d'un algorithme réparti

Définitions essentielles

- **Réception :** arrivée physique du message dans la couche communication du site destinataire. Le message est placé dans une *file d'attente locale*.
- **Délivrance :** prise en compte effective du message par le processus applicatif — déclenche une action locale (mise à jour, émission, calcul...).

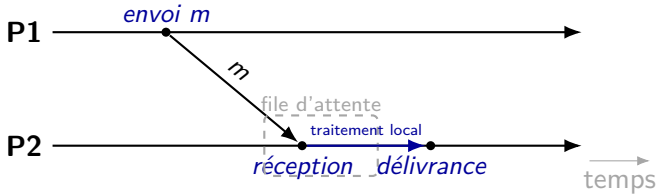


Réception vs Délivrance d'un message

Distinction dans le comportement d'un algorithme réparti

Définitions essentielles

- **Réception :** arrivée physique du message dans la couche communication du site destinataire. Le message est placé dans une *file d'attente locale*.
- **Délivrance :** prise en compte effective du message par le processus applicatif — déclenche une action locale (mise à jour, émission, calcul...).

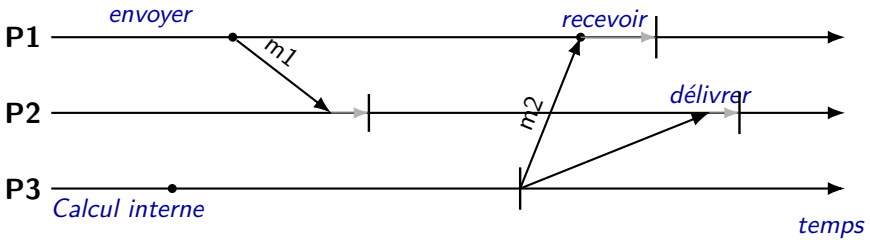


Comportement d'un algorithme réparti

Graphe de précédence immédiate

Graphe de précédence immédiate

- Capture d'un comportement global du système
  - ordre possible d'évènements (locaux) dans le temps ;
  - les messages en transit entre sites voisins ;
  - etc.
- Évènement local : envoi / réception / délivrance d'un message ou calcul interne.



Plan

Conception d'un bon algorithme réparti (1/2)

Critères d'évaluation

Objectif

Un bon algorithme réparti doit assurer une **exécution correcte**, **efficace** et **robuste** dans un environnement distribué incertain (pannes, délais, asynchronie...).

Critères de correction

- **Cohérence** : tous les sites ont une vision compatible de l'état global.
- **Terminaison** : chaque opération se termine dans un temps fini (en absence de pannes irréversibles).
- **Sécurité (Safety)** : rien de « mauvais » ne se produit (ex. pas de double accès critique).
- **Vivacité (Liveness)** : quelque chose de « bon » finit par se produire (ex. progression des requêtes).

Conception d'un bon algorithme réparti (2/2)

Critères de performance

- **Complexité en messages** : nombre total de messages échangés.
- **Délai (latence)** : temps nécessaire pour atteindre un état stable ou obtenir une réponse.
- **Charge locale** : quantité de calcul effectuée par chaque site.

Critères de robustesse et d'évolutivité

- **Tolérance aux fautes** : maintien de la cohérence malgré les pannes partielles.
- **Scalabilité** : adaptation à un grand nombre de sites ou à des charges variables.

Analyse de l'algorithme de diffusion

Analyse rapide

- **Correction :** si le graphe est connexe, tous reçoivent.
- **Messages :** $\mathcal{O}(|E|)$ utiles (au plus un aller utile par arête).
- **Asynchronie :** supportée (pas d'horloge globale).
- **Temps** : $T \leq$ diamètre du graphe (si rondes synchrones).
- **Inconvénients** : nombreux doublons ; pas d'arbre explicite.

Plan

Élection d'un coordinateur — Anneau orienté (ring)

- Processus arrangés logiquement en anneau ; chaque nœud a un ID unique.
- Idée : chaque nœud envoie son ID ; ne relayer que l'ID maximal observé.
- Le gagnant est celui dont l'ID revient à l'émetteur initial.

Pseudocode — Élection sur anneau

Algorithme 2 : Algorithme d'élection en anneau (version simple)

▷ `Initialisation`
me $\leftarrow$ $ID(p)$; actif $\leftarrow$ vrai;
envoyerCandidature(*me*) au successeur;

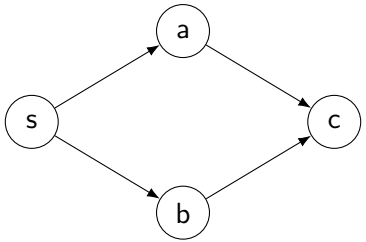
▷ `Procédure` *envoyerCandidature*(*id*)
envoyer le message de candidature de *id* au successeur;

▷`À la réception d'un message de candidature` (x)
si $x = me$ **alors**
  élu$\leftarrow$ me ; actif$\leftarrow$ faux ; // elu
sinon si $x > me$ ***et*** *actif* **alors**
  envoyer (cand, x) au successeur
sinon si $x < me$ ***et*** *actif* **alors**
  envoyer (cand, me) au successeur

Analyse — Élection

- **Messages** : $O(n \log n)$ pour des variantes optimisées ; $O(n^2)$ pour la version naïve.
- **Temps** : $O(n)$ tours de l'anneau.
- **Pré-requis** : IDs uniques ; fiabilité des canaux (au moins FIFO ou fiable).

Exemple visuel — Diffusion sur un graphe



Étapes : s émet $\Rightarrow$ a, b relaient $\Rightarrow$ c reçoit (doublon ignoré).

Détection de terminaison (intuition)

- Problème : savoir quand la diffusion/collecte est terminée.

Plan

Ressources

- Attiya, Bar-Noy, Dolev — *Distributed Computing* (textbook).
- Lynch — *Distributed Algorithms*.
- Tannenbaum, van Steen — *Distributed Systems*.
- Notes de cours et dépôts d'exemples.