
# Introduction

# Algorithmes distribués

**Maurice Djibril FAYE, enseignant associé à l'UVS**



## Objectifs spécifiques :

A la suite de ce chapitre, l'étudiant doit être capable de:

1. Comprendre la plupart des concepts et paradigmes des systèmes distribués (SD)

2. Connaitre  les formalisations et modèles utilisés pour ces systèmes

# Bref historique des technologies informatiques

- 1945-1985 : ère des ordinateurs modernes.
  - ✓ mais, ils étaient encombrants, chers, indépendants (moyens de les connecter inexistants )

- A partir de1985 : des avancées majeurs
  - ✓ développement de puissants microprocesseurs
  - ✓ développement des technologies des réseaux à haut débit ( LAN,WAN )

  - Conséquences ==> ordinateurs puissants (CPU multi-cœurs aujourd'hui), pas encombrants, moins chers, et possibilités de les connecter à travers un réseau,

# Bref historique des technologies informatiques

**Les progrès dans la miniaturisation des composants électroniques, et au niveau des réseaux informatiques; couplés aux besoins de partager des ressources, de collaborer à distance,…**

- ont permis l'apparition de systèmes constitués d'applications sur des machines généralement distantes (ou de machines uniquement) qui coopèrent pour fournir un service.

✓ Les systèmes distribués étaient nés et sont maintenant omniprésents

**Mais c'est quoi un système distribué?**

# Définition d'un système distribué :

- C'est une collection d'entités de calcul, autonomes, interconnectées par un système de communication [1]

✓ Il est conçu et apparait à ses utilisateurs comme une entité unique et cohérente[2]

✓ Les entités (appelées aussi **nœuds**) coopèrent pour atteindre un objectif (notion de système). En général cet objectif est soit de fournir un service de calcul, soit assurer des fonctions de contrôle, lié à la distribution (exclusion mutuelle, détection terminaison d'un calcul/algorithme,….)

✓ Les entités ont des fonctions de traitement, de stockage (mémoire), de relation avec le monde extérieur (capteurs/réception, actionneurs/émission) :

✓ Aucune hypothèse n'est faite sur :

- la nature des entités de calcul : ordinateurs, téléphones, capteurs, objets connectés, processus, processus légers,....
- la manière de les interconnecter : réseaux physiques ( Ethernet, infiniband, wifi,…) et organisation logique ( anneau, arbre…)

# Définition d'un système distribué

- "Un système distribué est un système dans lequel la panne d'une machine dont vous ignorez l'existence peut vous empêcher de travailler avec votre machine." L. Lamport.

.

✓ l'interdépendance des différents éléments d'un système

✓ les différents processus collaborent pour fournir un service.

✓ panne DNS ==> accès à des pages web impossible même pour quelqu'un qui ignore l'existence du serveur DNS

# Exemples de SD :

- Internet  (www)
- DNS  (Domain Name System)
- Trafic aérien
- Réseaux de capteurs pour l'agriculture, la météorologie,…
- Cloud
- P2P (Peer to Peer / pair à pair)
- Système de défense anti-aérienne ( radars, missiles)
- Réseaux de téléphones mobiles
- Etc.

# Caractéristiques d'un SD :

- Les SDs partagent certaines caractéristiques comme :

✓ **Absence d'une mémoire centrale :** Ce qui rend difficile/impossible la possibilité de déterminer un état global du système. Chaque entité/processus n'a qu'une vue locale :

- Connaît seulement les événements qu'il a générés, les messages qu'il a émis/reçus
- Ne connaît pas l'état du réseau ( ce qui y circule, et qui lui est peut être destiné), ni l'état des autres processus.

✓ **Absence d'une horloge commune :** Ce qui rend difficile la coordination des actions.

- Il n'existe pas de relation naturelle d'ordre strict sur des événements ayant lieu sur des entités/processus différents.

**NB** : en plus de ces deux caractéristiques commune la plupart des SDs, certains sont en plus dynamiques ( des processus peuvent quitter ou joindre le système), ce qui rend encore une vision globale plus difficile.

# Les objectifs d'un système distribué [2]

**Objectifs que se donne le concepteur d'un SD.**

- Faciliter l'accès aux ressources et leur partage:

✓ Les ressources peuvent être matérielles, logicielles, des services,…

✓ Partage de fichiers : P2P (BitTorrent)

✓ L'accès à des ressources de calcul et/ou de stockage ( cloud computing)

- Capacité de stockage : Google Drive, Microsoft OneDrive, Amazon
- Capacité de Calcul : Amazon EC2, projet SETI@home, plate-forme BOINC(Berkeley Open Infrastructure for NetworkComputing)

✓ Partager une imprimante dans un LAN au lieu d'avoir une imprimante pour chaque poste de travail

# Les objectifs d'un système distribué

- Assurer la transparence de la distribution du matériel et des traitements

✓ La transparence désigne ici le fait que cette distribution est cachée aux utilisateurs finaux ( humains ou applications )

✓ Cette transparence de la distribution peut concerner divers aspects d'un SD :

- **L'accès, la localisation, le déplacement, la réplication, la migration, l'accès concurrentiel à une ressource, la tolérance aux pannes**
  - **Ressource : physique ou logicielle (processus)**

# Les objectifs d'un système distribué

- La transparence peut concerner :

✓ **L'accès** : cacher les différences de représentations des données et de méthodes d'accès, l'hétérogénéité du matériel, des logiciels, des protocoles : Rendre les méthodes d'accès uniformes.

✓ **La localisation d'une ressource** : cacher la localisation géographique. Utilisation de noms logiques ( url , par exemple www.xyz.com; que le serveur soit hébergé au Sénégal , en Europe ou ailleurs ) qui ne fait pas référence à un endroit géographique. Un fichier sur Google Drive, l'utilisateur ignore sa localisation physique ( cette information n'est d'ailleurs pas nécessaire/pertinent pour utiliser son fichier).

✓ **Le déplacement d'une ressource** : cacher le fait qu'une ressource peut être déplacée , (changement de sa localisation ) pendant qu'elle est en cours d'utilisation. Le ressource est passive et subit l'opération. Un site web peut être déplacé sur un autre serveur sans que les utilisateurs ne s'en aperçoivent.

# Les objectifs d'un système distribué

## La transparence peut concerner (suite):

- **La réplication** : cacher le fait que plusieurs copies d'une ressource existent. Plusieurs copies d'un fichier; plusieurs processus travaillant en pool (quand un tombe en panne, un autre prend le relais). Il faut des mécanismes pour éviter les incohérence des données. Les copies ont le même nom.
- **La migration :** cacher le fait qu'une ressource est en train de se déplacer. Elle ne subit pas, elle est active. Ne pas se rendre compte du fait la personne avec qui on communique se déplace, son téléphone mobile aussi.
- **Les accès concurrentiels :** cacher le fait qu'une ressource peut être partagée par plusieurs utilisateurs indépendants :. Accès à un même fichier, à un même CPU, à une même table d'une Base de données. Ces accès concurrents ne doivent pas être perceptibles par l'utilisateur qui doit avoir l'impression d'être le seul utilisateur de la ressource)
- **La tolérance aux pannes :** cacher le fait qu'une partie du système ne fonctionne plus correctement , pour assurer une haute disponibilité. Peut être difficile, voire impossible.

# Les objectifs d'un système distribué

## La transparence de la distribution

- **Limites :**

✓ La transparence de la distribution est un objectif théorique, souhaitable quand c'est possible, mais qui repose sur des hypothèses très optimistes (réseau fiable, temps de traitements optimaux,…) qui ne sont pas toujours observées en pratique.

✓ Ce n'est pas toujours possible de la réaliser (partition du réseau, congestion du réseau pendant laquelle l'utilisateur va forcement noter des dysfonctionnements, la lenteur;..)

✓ Ce n'est pas aussi toujours souhaitable car pouvant causer des baisses de performances ( lorsqu'on est pas dans le cadre des hypothèses optimistes)

✓ Cela demande donc d'étudier les avantages et les inconvénients (il y a un débat entre les avantages et les inconvénients )

✓ Un système peut assurer une transparence de la distribution sur certains des aspects cités seulement, pas sur tous.

# Les objectifs d'un système distribué

- **Etre ouvert :** Un système ouvert

  - ✓ offre des services basés sur des **standards**. Une possibilité == décrire le service ( syntaxe, sémantique) à l'aide d'un Langage de définition d'interface (IDL=Interface Definittion Language).
  - ✓ Assure l'**inter-opérabilité** (des matériels, logiciels,…). Par exemple l'IDL de CORBA permet de décrire un service fournit pas un composant logiciel (attributs,types, méthodes, exceptions,…), et ce service peut être implémenter dans divers langages (Java,C++,..), et ces différentes implémentations seront inter-opérables.
  - ✓ Facilite l'**intégration** des services et des matériels
  - ✓ Est **extensible**. L'ajout d'une ressource se fait aisément . Cela est facilité par les standards et l'inter-opérabilité.
  - ✓ **NB** : en pratique, il y a souvent des limites dans les possibilités d'inter-opérabilité (considérations économiques, humaines,…)

# Les objectifs d'un système distribué

- Etre capable de passer à l'échelle ( scalabilty)
  - ✓ Fonctionner efficacement à différents niveaux d'échelle :
    - un réseau social qui peut supporter efficacement (sans une détérioration perceptible de la qualité de service) une variation du nombre d'utilisateurs passant de quelques milliers à plusieurs millions d'utilisateurs en ligne.
    - Un système de réservation de billets d'avion, d'achat de produits en ligne; qui peut connaître des périodes de basse activité et des périodes d'intense activité comme les vacances, les fêtes, et qui est capable de gérer /supporter efficacement ces variations de charge.

# Les objectifs d'un système distribué

- Etre capable de passer à l'échelle ( scalabilty)

✓ Cette capacité peut concerner aussi bien la taille (les variations du nombre d'entités: passage de milliers à millions), que la distribution géographique (à l'échelle d'un pays, échelle d'un continent sans que les délais plus longs ne soient perceptible ).

- Taille : Les services à la demande sur les clouds : Le nombre de machines virtuelles et/ou capacité de stockage alloués à un utilisateur peuvent varier sensiblement en fonction des périodes de l'année, de besoins conjoncturelles,…mais le système doit s'efforcer de fournir la même qualité de service.
- Distribution géographique : requête sur un serveur web en local (machine locale, réseau local) ou distant : différences sur les temps de réponse non perceptibles.

# La tolérance aux pannes ( fault tolerance )

- La tolérance aux pannes vise à masquer les effets d'une défaillance et/ou à restaurer un comportement conforme [à sa spécification] pour un système qui a dévié de sa spécification à cause d'une faute [3].

- Capacité du système de continuer à fournir un service (même dégradé) lorsqu'une partie du système ne fonctionne plus correctement.

- Un système distribué peut être complexe, impliquant divers types de ressources autonomes (pouvant défaillir localement et de manière indépendante), géographiquement réparties, raison pour laquelle les fautes et les défaillances sont plus courantes que dans les systèmes centralisés. Une panne peut être locale et affecter le comportement d'une partie des autres nœuds du système sans affecter une autre partie.

# La tolérance aux pannes ( fault tolerance )

- Un système distribué est **en panne** lorsqu'il ne se comporte plus conformément à sa fonction (ce pour quoi il a été prévu) [4].
- **Une erreur** est une partie de l'état du système qui peut causé une panne.
- **Une faute** est ce qui cause une erreur.
- Être capable de détecter les fautes est donc d'une grande importance ( peut être difficile voire impossible)
- Ainsi, un système est tolérant aux pannes s'il peut continuer à fournir le service pour lequel il est prévu, et ceci même en présence de fautes (cela dépendra de la nature des fautes).

# La tolérance aux pannes : Modèles de fautes

- Lorsqu'on a une vue du système distribué de niveau processus, on peut distinguer les différents types de fautes au niveau processus [2, 3, 4].
  - ✓ **les arrêts** : un processus à l'arrêt cesse d'exécuter ses actions (interne, de communication, de lecture et d'écriture). L'arrêt peut être définitif ("crash stop") ou temporaire ("crash recovery");
  - ✓ **les omissions** : elles modélisent les fautes qui peuvent conduire à la perte de messages. Ce type de faute peut affecter les canaux de communication et se manifester sous la forme d'une rupture du lien (dû à une problème au niveau du réseau physique sous-jacent par exemple) rendant certaines communications impossibles. Les fautes au niveau des canaux peuvent aussi provoquer la perte, la duplication, la transmission hors délais des messages. Un canal qui peut perdre des messages peut être modélisé en considérant qu'un des processus au bout du canal échoue à transmettre ou à recevoir certains messages qu'il devait envoyer ou recevoir. Un autre moyen de modéliser les pertes de messages dans un système synchrone avec passage de messages est de permettre la perte d'au plus un certain nombre de messages à chaque round, mais les canaux sur lesquels ces pertes apparaissent peuvent changer d'un round en un autre;
  - ✓ **les pannes temporelles** : elles sont dues à un délai non respecté, par exemple dans un système temps réel où on exige que les actions soient terminées dans un intervalle de temps donné.

# La tolérance aux pannes : Modèles de pannes

- Les différents types de pannes peuvent être classés dans des catégories de plus haut niveau :

✓ **les pannes transitoires** : une panne transitoire peut perturber l'état d'un processus d'une manière arbitraire. Elles capturent les effets de l'environnement, dont **la durée est limitée**. L'élément responsable de la panne peut n'être actif que pendant un temps limité, mais l'effet produit sur l'état global du système reste. Les omissions sont un cas de panne transitoire, lorsque l'état d'un canal est perturbé;

✓ **les pannes byzantines** : elles modélisent un comportement arbitraire des processus. Ce dernier modèle est utile pour simuler des attaques et situations dans lesquelles les fautes sont difficiles à caractériser. Un algorithme dans le modèle avec fautes byzantines doit donc fonctionner correctement (atteindre son but) quel que soit le comportement des processus.

# La tolérance aux pannes ( fault tolerance )

- Un système distribué peut être complexe, impliquant divers types de ressources autonomes (pouvant défaillir localement et de manière indépendante), géographiquement réparties,

- Pour cette raison les fautes et les défaillances sont (potentiellement) plus courantes que dans les systèmes centralisés. Une panne peut être locale et affecter le comportement d'une partie des autres nœuds du système sans affecter une autre partie.

# Notion d'intergiciel ( Middleware)

- Les SDs sont caractérisés par une hétérogénéité du matériel, des logiciels de bas de niveau ( systèmes d'exploitation par exemple).

- Pour cacher cette hétérogénéité à l'utilisateur, et fournir un service standard ( Même interfaces d'accès au service), une couche logicielle est généralement utilisée. Elle est logiquement située entre le niveau application (utilisateur) et le niveau matériel ,dans une architecture à 3 couches.

- Cette couche logicielle est appelée **Intergiciel** (Middleware en anglais, middle=milieu/centre)

- Elle offre des facilités de communications inter-application, de sécurité, de tolérance aux pannes,…

- Par analogie, l'intergiciel joue pour un SD (environnement réseau), un rôle comparable à celui que joue un système d'exploitation pour un ordinateur (environnement local).

# Modélisation d'un SD

Un SD peut être :

- Un réseaux d'entités de calcul physiques: ordinateurs, téléphones, capteurs, objets connectés,…
  - ✓ Réseaux téléphoniques, réseaux de capteurs,…
- Un réseau de processus formant un logiciel distribué, les processus s'exécutant sur la même machine ou sur des machines distantes interconnectées.
  - ✓ Dans ce cas, on confond le processus et la machine sur laquelle il s'exécute.

Les SDs sont implémentés de diverses manières, mais peuvent être modélisés par un Graphe ➔

# Modélisation des SD :

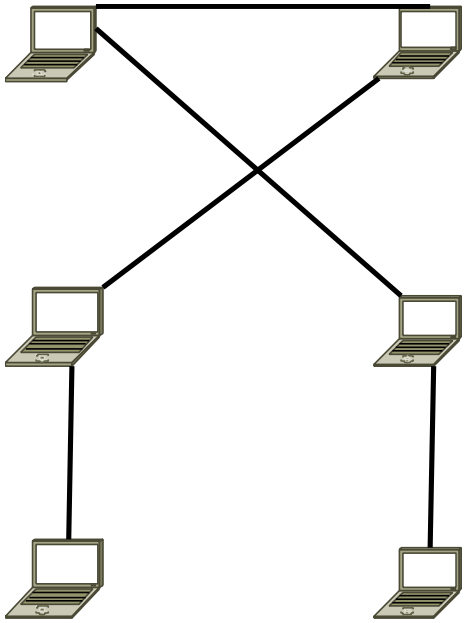
Un SD peut être modélisé par graphe (non orienté,connexe) G=(V,E) où :

- **V** = l'ensemble des sommets, qui désignent les **processus/machines** (dans la suite processus et machines sont confondues).

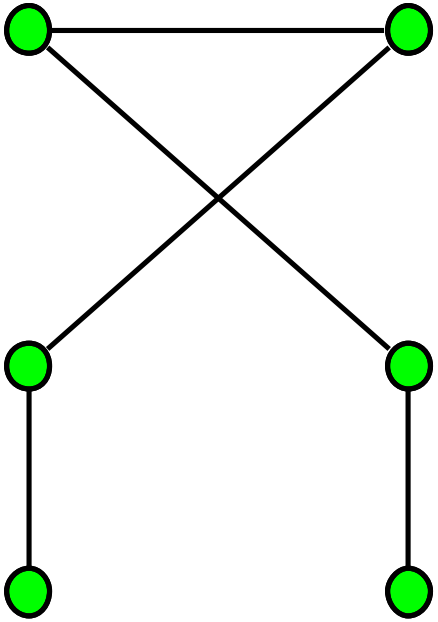
- **E** = l'ensemble des arêtes (une arête relie deux sommets), désignant le réseau d'interconnexion des processus. Une arête désigne un **canal de communication**. La structure du réseau désigné par E, est appelé topologie.

# Modélisation des SD :

Représentation simplifiée du SD



- Abstraction par un graphe(sommet = machine/processus)



algorithme distribué local : algorithme qui s'exécute sur chaque sommet (en utilisant uniquement le contexte local) et
Qui peut envoyer/recevoir des messages,

# Modélisation des SD : La Topologie du réseau

- La topologie du réseau caractérise le maillage des liaisons de communication.

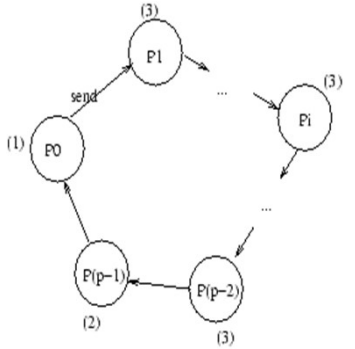
- Ce maillage peut être **logique**, et dans ce ca, obtenu grâce à une couche logicielle qui crée un réseau logique au dessus du réseau physique. Ce réseau logique est appelé « overlay » en anglais.

- Exemple de topologie fréquemment rencontrées: anneau, étoile, hypercube, graphe complet,… ➔

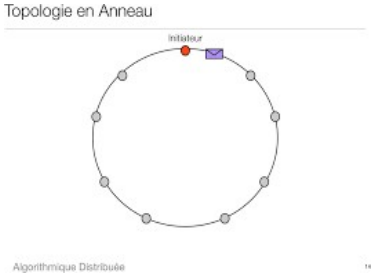
# Modélisation des SD : La Topologie du réseau

- **Anneau** :

- Un processus ne peut communiquer directement qu'avec
  - ✓ son voisin immédiat (anneau unidirectionnel) selon l'orientation de l'anneau,
  - ✓ Ses deux voisins immédiats ( anneau bidirectionnel).




  - Si n processus ( $P_1$,…,$P_n$) :
    - $P_i$ ( i ≠ n) ne communique qu'avec $P_{i+1}$ et $P_n$ avec $P_1$ ➔ unidirectionnel
    - Il existe un canal entre chaque $P_i$ et ses deux voisins immédiats ➔ bidirectionnel

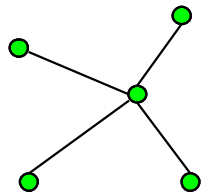


# Modélisation des SD : La Topologie du réseau

- **Étoile** :

- Il existe un processus particulier $P_1$ tel que

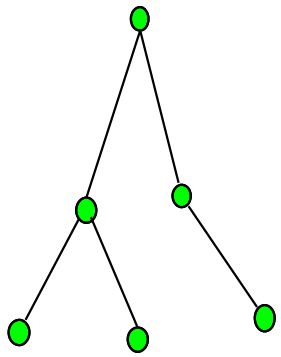
  ✓ Tout $P_i$ ( $i \neq 1$) ne communique qu'avec $P_1$ et $P_1$ communique tous les autres



# Modélisation des SD : La Topologie du réseau

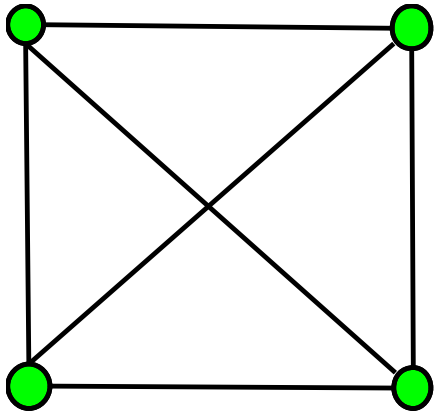
- **Arbre :**

  - ✓ $P_1$ ne communique qu'avec ses fils ($P_1$ est la racine)
  - ✓ Si $P_i$ ( $i \neq 1$) a des fils; alors il communique qu'avec ses fils et son père.
  - ✓ Si $P_i$ ( $i \neq 1$) n' a pas de fils; alors il communique qu'avec son père ( une feuille)



# Modélisation des SD : La Topologie du réseau

- **Graphe complet** : Il existe un canal entre chaque paire de processus.



- Etc…

# Modélisation des SD : Notion de Processus

- Un processus correspond à un fichier exécutable en exécution. Il présente un flot de contrôle unique, composé d'instructions atomiques, exécutées de manières séquentielle.

- Il est caractérisé par un code, une pile et tas permettant de stocker les données nécessaires à son bon fonctionnement, un identifiant unique, une priorité.

- Les instructions peuvent être de deux types:
  - ✓ Communications : envoi/ délivrance de message
  - ✓ Calcul local : toute autre activité interne ( calcul d'une valeur,modification de la valeur d'une variable,démarrage d'un timer,…)

# Modélisation des SD : Notion de Processus

Dans un algorithme distribué, chaque processus a une vision locale ( connait son état local) :

- Cet état local englobe :
  - ✓ Les événements/actions générés par le processus
  - ✓ Les messages émis et reçus

- Mais peut aussi englober ( réseau anonyme ou non )
  - ✓ Son identifiant
  - ✓ L'identifiant de ses voisins immédiats, et dans certaines situations la taille du réseau ( nombre de nœuds).

NB : Toute autre information voulue doit être recherchée de temps à temps.

# Modélisation des SD : Hypothèses fréquentes

Quand on modélise un SD, ou quand on doit écrire un algorithme distribué, des hypothèses sont faites, pouvant concerner :

- Les délais / le passage du temps ( synchrone/asynchrone)
- Les Canaux de communication

# Modélisation des SD : Hypothèses sur les délais/temps

- Un aspect fondamental dans la caractérisation d'un SD est le comportement de ses processus et canaux par rapport au passage du temps.

- Les hypothèses faites sur les délais de transmission des messages et sur la vitesse (relative) des processus (temps de traitement des informations échangées) sont d'une grande importance.

- Certains problèmes difficiles dans les SD sont liés aux hypothèses faites : Un problème peut avoir une solution dans avec certaines hypothèses, et être impossible à résoudre avec d'autres hypothèses.

- Selon les hypothèses faites , un système est dit **synchrone** ou **asynchrone**.

# Modélisation des SD : Hypothèses sur les délais/temps

- **Systèmes asynchrones**

  - ✓ Dans ce modèle, il n y a pas de borne sur les délais de propagation des messages et de traitement des messages par les processus.
    - Les processus peuvent donc être arbitrairement lents.

  - ✓ Conséquence ➔ impossibilité pour un processus de faire la différence entre un processus lent et un processus en panne.

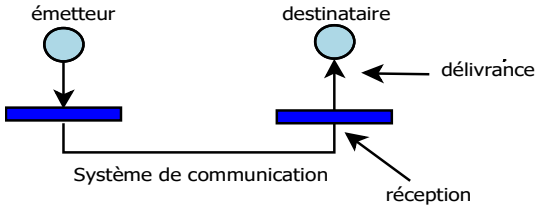
  - ✓ Un algorithme qui fonctionne sous ces hypothèses, fonctionnera aussi avec un système synchrone

# Modélisation des SD : Hypothèses sur les délais/temps

- **Systèmes synchrones :** Cette hypothèse peut concerner plusieurs aspects d'un SD.

  ✓ Calcul/traitement synchrone : Il existe une borne supérieure **connue** sur le temps de traitement d'une phase. Les traitements se font phase par phase. Une phase est constituée de la consommation/délivrance d'un message (envoyé par un autre processus et qui peut être vide), d'un calcul local, et de l'envoi d'un message (qui peut être omis).

  - Ainsi le temps pris pour exécuter une phase, quel que soit le processus, est toujours inférieur à cette borne.
  - Il peut y avoir un délai entre la réception d'un message et sa délivrance.



# Modélisation des SD : Hypothèses sur les délais/temps

- **Systèmes synchrones :** Cette hypothèse peut concerner plusieurs aspects d'un SD.

✓ Communications / canaux synchrones : Il existe une borne supérieure **connue** sur les délais de transmission des messages.

- Ainsi le délai entre l'instant où le message a été envoyé, et l'instant où le message a été consommé/délivré, est toujours inférieur à cette borne.
- $P_1$ Si le message envoyé par n'est pas délivré par $P_2$ dans l'intervalle de temps attendu, alors $P_1$ suppose l'existence de pannes.

# Modélisation des SD : Hypothèses sur les canaux

Propriétés des canaux : elles sont diverses. Les plus fréquentes.

- H1 : la transmission sur le canal se fait sans duplication de message
- H2 : la transmission sur le canal se fait sans altération de message
  - ✓ Canal fiable
- H3 : pour tout couple de processus, l'ordre de délivrance des messages est le même que leur ordre d'émission.
  - ✓ Canal FIFO
- H4 : le délai d'acheminement des messages est fini. Tout message est envoyé est reçu au bout d'un temps fini.
  - ✓ Asynchrone : modélise les situations où le message peut être perdu, problème détecté, et message ré-émis.
- H5 : le délai d'acheminent des messages est borné. Si un message n'est pas reçu après le délai connu, il est perdu.
  - ✓ Synchone.

# Problèmes communs aux SDs

- Bien que les SDs se présentent sous différentes facettes, un certain nombre de problèmes fondamentaux leur sont communs.
- Un concepteur d'un application distribuée, peut être amené à devoir trouver une solution ou à utiliser les algorithmes/solutions existants, concernant un ou plusieurs de ces problèmes.
- Parmi ces problématiques fondamentaux:
  - ✓ Synchronisation : comment coordonner (??ordre???) des opérations en l'absence d'une horloge commune ?
  - ✓ Election d'un leader :
  - ✓ Exclusion mutuelle
  - ✓ Calculer un état global
  - ✓ Consensus, terminaison d'une exécution, auto-stabilisation,...
  - ✓ Gestion répartie des données
  - ✓ Tolérance aux pannes
  - ✓ Comment décrire une exécution répartie ?
  - ✓ Y a-t-il des problèmes intrinséquement insolubles, et le cas échéant, que fait on en pratique ?

# Conclusion

- La conception d'algorithmes distribués nécessite :

  - ✓ De donner les propriétés structurelles et comportementales des liaisons

  - ✓ De choisir la technique à utiliser ( jeton, vagues ou calcul diffusant)

  - ✓ Ecrire le code

  - ✓ Valider l'algorithme ( preuve, simulations,…)

# Introduction

## Bibliographie


1. H. Attiya and J. Welch. Distributed Computing: Fundamentals, Simulations and Advanced Topics. Wiley-Interscience, 2004
2. Tanenbaum and van Steen, Distributed Systems , 3rd Edition, 2017
3. Sukumar Ghosh. Distributed systems: an algorithmic approach. CRC press, 2007.
4. Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Dependability and its threats: a taxonomy. In Building the Information Society, pages 91–120. Springer, 2004
5.Gerard Tel : “Introduction to distributed systems”
6.Paul Sivilotti : “introduction to distributed systems”, OHIO State University, 2007
7. C Cachin et al. “Introduction to reliable and secure distributed programming”, Springer-Verlag Berlin Heidelberg,2011

Support de cours du Pr Sacha Krakoviak (http://lig-membres.imag.fr/krakowia/ ) (2018)
Support de cours du Pr Ousmane Thiaré (http://ousmanethiare.com/fr/enseignement) (2018)
.Support de cours du Pr Bernard Pottier (http://wsn.univ-brest.fr/moodle/) connexion anonyme (2018)
. :