

Algorithmes répartis

Applications réparties Chapitre 4

Polytech Marseille

Département Informatique
Formation par alternance / HUGo

5^{ème} année

Algorithmes répartis

Introduction

- ◆ Conception d'algorithmes répartis : plusieurs difficultés.
- ◆ ***Coordination sites distants***, exploration d'un réseau ou centralisation d'une information ne sont pas immédiates.
- ◆ ***État du système*** peut évoluer au cours de ces tâches que l'on aurait voulu "atomiques".
- ◆ Bien souvent, les sites doivent ***prendre une décision commune*** (élection d'un leader/coordonateur ou bien consensus : converger vers une valeur unique sur chaque site)
- ◆ Utilisation de ***ressources partagées*** entre les sites : exclusion mutuelle, gestion des interblocages, détection de terminaison, ...

Algorithmes répartis

Correction

- ♦ Un algorithme est **correct** lorsqu'il suit les spécifications qu'on lui a assignées.
- ♦ 2 types :
 - tâche ***dynamique*** (algorithmes définis par le comportement qu'ils doivent avoir),
 - tâche ***statique*** (algorithmes caractérisés par les résultats qu'ils doivent construire).

Algorithmes répartis

Propriétés

- ◆ Propriété de *sûreté* : permet notamment d'affirmer qu'un événement ou une situation dommageable pour le système ne se produira pas.
- ◆ Propriétés de *vivacité* : ont pour but de caractériser un changement d'état, c'est-à-dire la production d'un résultat par le système.

Algorithmes répartis

Complexité en temps

- ♦ Complexité en *temps* d'un algorithme réparti
- ♦ Temps nécessaire au déroulement d'un algorithme distribué se répartit entre le temps *dû aux calculs locaux* et celui *dû aux échanges de messages*.
- ♦ Mais hypothèse : les calculs locaux sont immédiats, donc seul le *temps nécessaire aux communications*.
- ♦ Dans système asynchrone : délai séparant le début et la fin de la plus grande séquence de messages consécutifs.
- ♦ Dans système synchrone : nombre de phases globales nécessaires à son accomplissement.

Algorithmes répartis

Complexité en mémoire

- ♦ Complexité en *mémoire* d'un algorithme réparti est égale à la *plus grande capacité mémoire nécessaire* à un site pour fonctionner.
- ♦ Préférence pour les algorithmes nécessitant le moins d'états internes, ou le moins de variables locales, etc.
- ♦ Utile en pratique que dans des cas particuliers, tels que les systèmes embarqués.
- ♦ L'unité considérée est le *bit* ou l'*octet*.

Algorithmes répartis

Complexité en nombre de messages

- ♦ Complexité en ***nombre de messages*** (évaluée globalement) : nombre total de messages émis lors d'une exécution (***en fonction du nombre de nœuds***).
- ♦ Si rôle identique (code uniforme, réseau assez "homogène"), cette mesure donne une idée du ***trafic requis*** par l'algorithme.
- ♦ Mesure difficilement exploitable si un nœud a un rôle particulier, ou bien si le réseau présente des nœuds ou des arcs qui constituent des passages obligés pour certaines communications inter-sites.
- ♦ Deux algorithmes ayant même complexité en nombre de messages : comparaison de la ***taille des messages***.

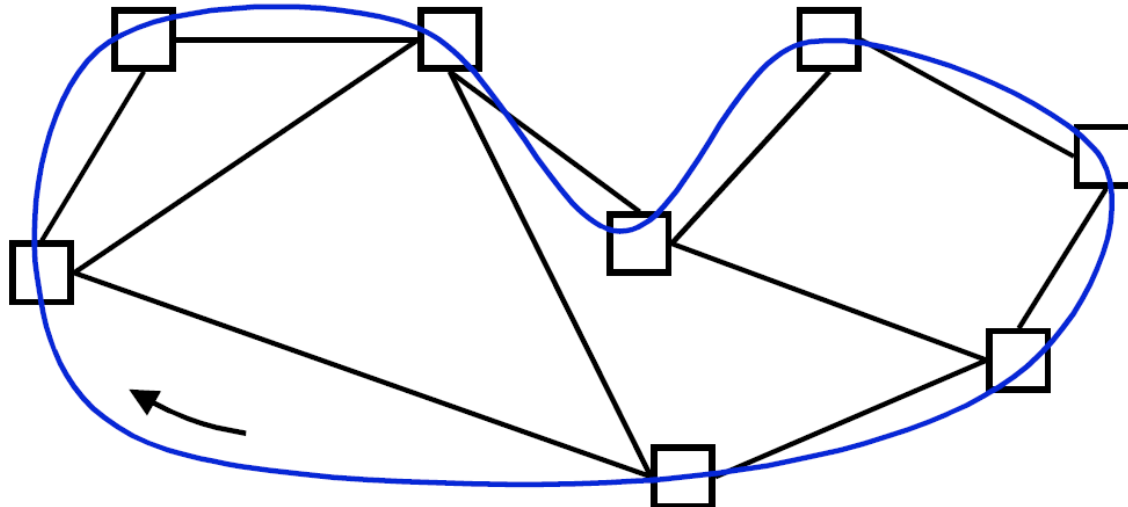
Algorithmes répartis

Généralités

- ◆ Quelques algorithmes de base (hors défaillances) :
 - Exclusion mutuelle, Élection, Terminaison, Consensus.
- ◆ Approches générales
 - Introduction de **contraintes** sur le déroulement des événements qui composent l'algorithme, pour faciliter le raisonnement.
 - Plusieurs types de contraintes
 - **Ordre des événements** (divers types d'horloges, voir annexe "Horloges logiques").
 - Contraintes sur la **topologie** de l'application (voies d'échange des messages).
 - ◆ Anneau virtuel ; Arbre ; Graphe acyclique.

Anneau virtuel

- ◆ Anneau virtuel : réseau superposé au réseau physique (overlay network), utilisant les protocoles du réseau physique.

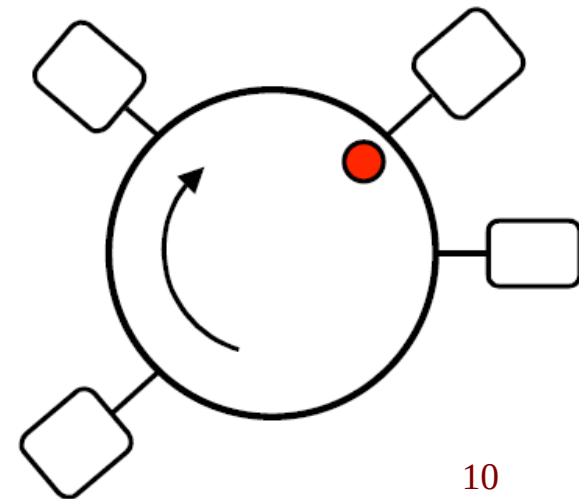


- ◆ Anneau virtuel défini en indiquant pour tout site i son successeur $succ[i]$; on peut aussi indiquer son prédécesseur $pred[i]$.

Exclusion mutuelle sur un anneau

- ◆ Inspiration : systèmes physiques de communication (*anneau à jeton* ou *token ring*)

- Les sites émettent et reçoivent des messages sur un bus unidirectionnel en anneau.
- Un site au plus peut émettre à tout instant.
- Un site ne peut émettre que s'il possède un jeton identifié par une configuration unique de bits.
- Un site garde le jeton pendant un temps limité.
- Il y a un jeton et un seul (invariant à maintenir en cas de défaillance).



Algorithmes à jeton

- ♦ De nombreux algorithmes répartis utilisent des jetons
 - La structure de communication sous-jacente peut être quelconque (anneau, arbre, graphe, non contrainte).
 - Le jeton peut être simple (seule est testée sa présence) ou porter une ou plusieurs valeurs.
- ♦ Problèmes communs
 - Garantir la **sûreté** : unicité (en général) et intégrité du jeton, atomicité des actions.
 - Garantir la **vivacité** : existence du jeton, circulation du jeton, régénération du jeton en cas de perte.
 - On cherche à développer des classes d'algorithmes génériques assurant ces propriétés.

Maintien de l'intégrité de l'anneau

- ♦ Hypothèses : réseau fiable ; défaillance des sites ; une défaillance ne partitionne pas le réseau.
- ♦ Principe : auto-surveillance (chaque site surveille son prédécesseur et/ou son successeur).
- ♦ En cas de détection de défaillance : reconfiguration (si besoin, régénération du jeton).
- ♦ Après réparation : réinsertion.
- ♦ Existence et unicité du jeton
 - Détection de la perte (délai de garde sur un tour, ou jeton de contrôle).
 - Perte détectée : choix d'un site (et d'un seul) pour régénérer un jeton : problème de l'**élection**.

Exclusion mutuelle

◆ Spécifications

- Imposer un ordre sur l'exécution des sections critiques,
- Algorithme symétrique, décentralisé,
- Algorithme équitable,
- Garantie de vivacité.

◆ Solutions

- Imposer un ordre global
- Imposer une topologie
 - Anneau virtuel,
 - Arbre.

Exclusion mutuelle

- ♦ Exclusion mutuelle : problème simple en centralisé, assez complexe en réparti.
- ♦ Sa solution illustre les principales classes d'algorithmes répartis :
 - Demande d'autorisation à tous les sites avec ordonnancement par estampilles [Ricart-Agrawal].
 - Jeton sans topologie sous-jacente (avec informations d'ordonnancement portées par le jeton) [Suzuki - Kasami].
 - Jeton sur anneau virtuel.
 - Jeton avec topologie en arbre reconfigurable [Raymond].
- ♦ Beaucoup d'autres algorithmes.
- ♦ En nombre de messages, algorithmes sur arbres sont les plus efficaces (Raymond, Naïmi-Tréhel) : $O(\log n)$ messages.
- ♦ Pour un nombre réduit de sites, l'anneau virtuel est le plus simple.

Élection

◆ Problème

- Parmi un ensemble de sites, en **choisir un et un seul** (et le faire connaître à tous)
 - Sûreté : un seul site élu,
 - Vivacité : un site doit être élu en temps fini.
- Élection peut être déclenchée par un site quelconque, éventuellement par plusieurs sites.
- Identité de l'élu est en général indifférente ; on peut donc (par exemple) choisir le site qui a le plus grand numéro.
- Difficulté : des sites peuvent tomber en panne pendant l'élection.

◆ Utilité

- Régénération d'un jeton perdu : un jeton et un seul doit être recréé (par l'élu).
- Dans les algorithmes de type "maître-esclave" : élection d'un nouveau maître en cas de défaillance du maître courant.

Détection de terminaison

- ◆ Comment savoir si un calcul, une application, ... réparti est terminé ?
- ◆ Problème de la détection de terminaison
 - Vérifier que le calcul est achevé.
 - Cela implique deux conditions sur l'état global du système.
 - Tous les sites sont au repos (passifs).
 - Aucun message n'est en transit.
 - En effet l'arrivée d'un message en transit peut relancer le calcul.

Détection de terminaison

- ◆ Méthodes pour détecter la terminaison
 - Méthode générale : analyse de l'état global.
 - La terminaison est une propriété stable.
 - On peut donc la détecter par examen d'un état global enregistré.
 - Méthodes spécifiques : applicables à un schéma particulier de communication.
 - Sur un anneau.
 - Sur un arbre ou un graphe orienté (avec arbre couvrant).

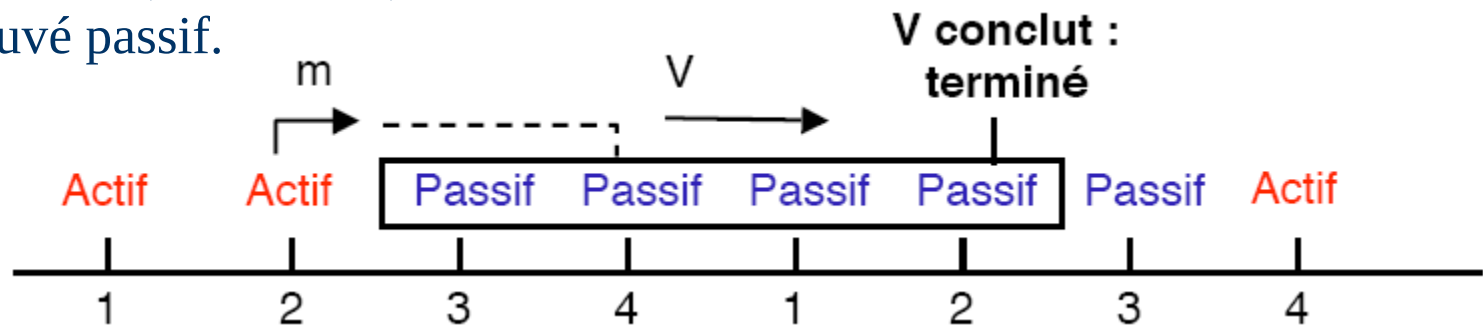
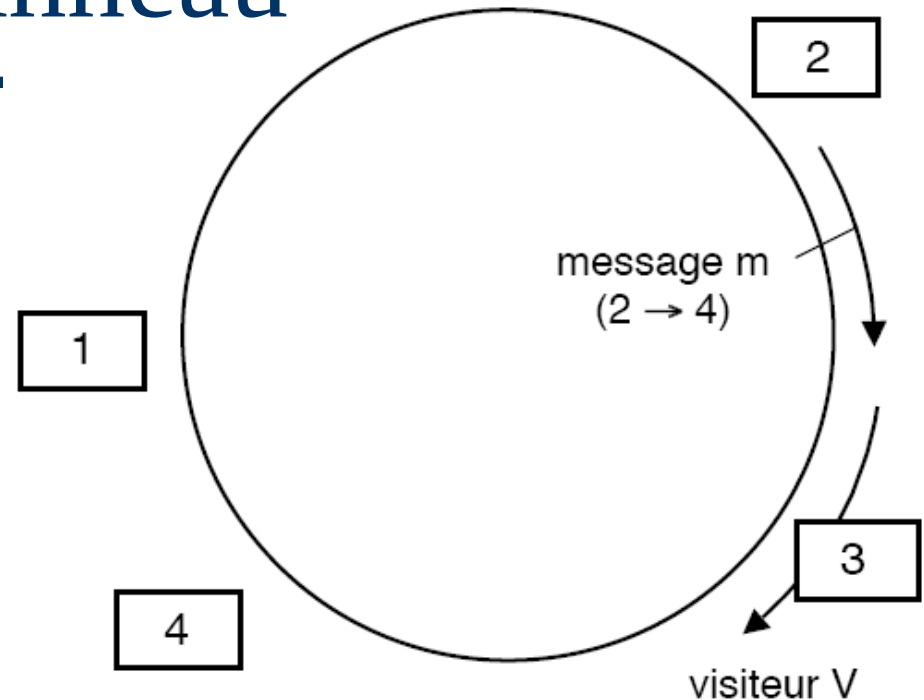
Détection de terminaison sur un anneau

◆ Principe :

- Visiter l'anneau (dans le sens de la communication) et vérifier que tous les sites sont passifs (après un tour complet).

◆ Difficulté :

- Un message émis après le passage du visiteur (et non visible par celui-ci) peut venir réactiver (derrière lui) un site trouvé passif.



Consensus

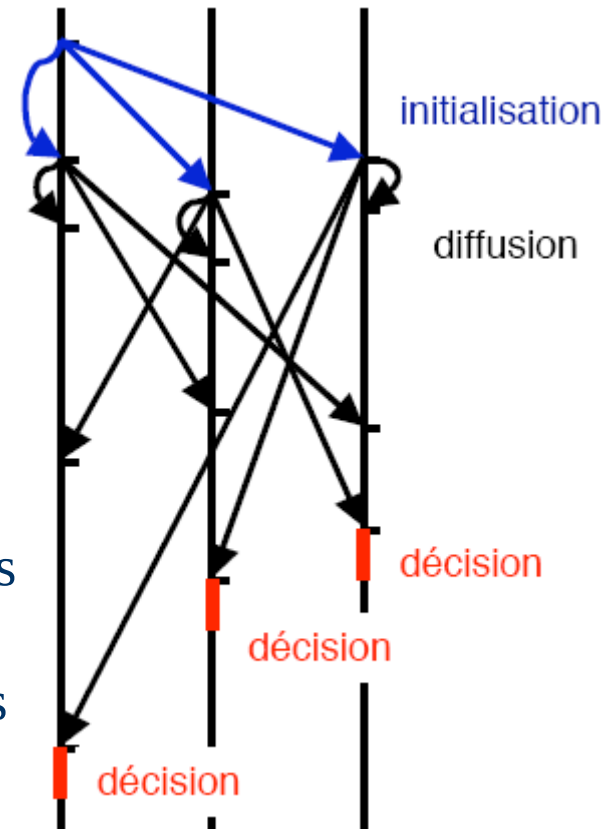
- ◆ Soit un ensemble de sites reliés par des canaux de communication.
 - Initialement : chaque site propose une valeur.
 - À la fin (si l'algorithme se termine) : chaque site décide d'une même valeur.
- ◆ Conditions :
 - **Accord** : la valeur décidée est la même pour tous les sites corrects.
 - **Intégrité** : tout site décide au plus une fois (une décision prise est définitive).
 - **Validité** : toute valeur décidée est l'une des valeurs proposées.
 - **Terminaison** : si au moins un site correct lance le consensus, tout site correct décide au bout d'un temps fini.

Consensus

- ♦ Hypothèses : communication fiable asynchrone, sites fiables (pas de défaillances).
- ♦ Une solution avec coordinateur.
 - Chaque processus p_i envoie sa valeur v_i à un coordinateur spécifié à l'avance.
 - Quand le coordinateur a reçu toutes les valeurs v_i , il choisit une de ces valeurs, soit d (sur un critère quelconque).
 - Le coordinateur envoie d à tous les p_i .
 - Après réception, chaque p_i décide d .
- ♦ En communication asynchrone, tous auront décidé en temps fini mais non borné.


Consensus

- ◆ Une autre méthode (solution symétrique).
 - Chaque processus p_i diffuse sa valeur v_i à tous.
 - Quand un processus p_i a reçu toutes les valeurs v_j , il applique un algorithme de choix d'une de ces valeurs (par exemple définir un ordre sur un critère quelconque et prendre la première).
 - L'algorithme de choix est le même pour tous les p_i .
- ◆ Si le protocole de diffusion garantit que tous les destinataires reçoivent le message (sans difficulté en l'absence de pannes), alors tous les p_i décident la même valeur.
- ◆ En communication asynchrone, tous auront décidé en temps fini mais non borné.



Conclusion

- ◆ Il existe beaucoup d'autres algorithmes répartis.
- ◆ Quelques techniques
 - Ordonnancement par estampilles.
 - Structures de communication (anneau, arbre) imposant un mode de propagation.
 - Utilisation d'un jeton.
- ◆ Propriétés
 - Sûreté,
 - Vivacité (progrès, terminaison),
 - Capacité de croissance,
 - Tolérance aux fautes.



ANNEXE

Modélisation des systèmes répartis

Définition (rappel)

- ♦ Un **système réparti** peut être défini comme un réseau d'*entités* "calculantes" ayant le même *but commun* : celui de la réalisation d'une *tâche globale* à laquelle chaque entité contribue par ses calculs locaux et les *communications* qu'elle entreprend.

Caractéristiques

- ♦ La *topologie* du réseau, nombre de sites connu ou non
- ♦ L'*uniformité* ou non des codes (codes identiques d'un nœud à un autre)
- ♦ La *synchronisation* des nœuds,
- ♦ La *tolérance aux pannes*,
- ♦ Le type de *communications*, le temps de délivrance d'un message fixé ou non, l'ordre des messages sur un lien est connu ou aléatoire.

Généralités

Site

- ♦ **Site** : caractérisé par son programme, ses variables, sa connaissance de l'extérieur.
- ♦ Selon les contextes, un site est appelé : processeur, entité calculante, *processus*, *nœud*, agent, etc.
- ♦ Pour un site, la seule façon de recevoir de l'information sur l'état du système ou sur une autre entité consiste à recevoir un "*message*" au sens large.

Généralités Réseau

- ♦ **Réseau** : liens qui véhiculent les messages d'un site à un autre.
- ♦ Terme générique désignant les moyens de communication
- ♦ Caractérisation du réseau :
 - (qualitative) la *topologie*, la *connectivité*, la *synchronisation* des envois de messages, les communications uni- ou bi-directionnelles, etc.
 - (quantitative) le diamètre du réseau, sa bande passante, sa latence, la taille maximale d'un message, etc.

Généralités

Connaissance globale

- ♦ Accès éventuel à une certaine connaissance globale du système réparti est un paramètre important.
 - Horloge globale, numérotation des entités, état global périodiquement distribué (synchronisation, terminaison, etc.), identifiant d'un nœud particulier (élection d'un leader), etc.
- ♦ Distinction : systèmes dont chaque nœud connaît tous les autres ; systèmes où les nœuds n'ont qu'une connaissance partielle des autres.
- ♦ **L'identifiant des nœuds** est un paramètre central.
 - Si pas moyen de distinguer deux nœuds quelconques
 - système **anonyme** (plusieurs problèmes classiques sont alors insolubles) dans un système anonyme.

Généralités

Mode de communication

- ♦ Mode de communication : caractéristique importante d'un système réparti
- ♦ Distinction : communications par *échange de messages* ; communications par *mémoire partagée*
- ♦ *Tampons de réception* sont de taille finie ou non
- ♦ Messages peuvent être désordonnés ou non (propriété *FIFO*).
 - Si les communications ne sont pas effectuées en mode connecté, alors les messages peuvent arriver par des routes différentes, et dans un ordre différent de celui d'émission.

Modélisation

- ◆ Système réparti : un *ensemble d'ordinateurs autonomes géographiquement dispersés et interconnectés* par un réseau de lignes de communication.
- ◆ *Chaque nœud* : un processeur, une horloge, une mémoire centrale (volatile), une mémoire secondaire (non volatile) et une interface raccordée au réseau de communication.
- ◆ Les nœuds ne partagent aucun composant et ils communiquent exclusivement à travers le réseau de communication.
- ◆ Formalisation du *modèle physique* par un *modèle logique*

Modélisation

- ♦ **Modèle logique** : programmes qui s'exécutent sur le système.
- ♦ Ensemble de processus s'exécutant de manière concurrente et coopérant pour réaliser une tâche commune.
- ♦ **Processus** : exécution d'un programme sur un nœud.
- ♦ Coopération entre les processus par *échange de messages*.
- ♦ **Message** : ensemble d'information qu'un processus désire communiquer à un ou plusieurs processus.
- ♦ Échange de messages permet aux processus de prendre des *décisions concernant l'état du système*.
- ♦ Transmission d'un message d'un processus à un autre via un canal de communication.
- ♦ **Canal** : connexion logique entre deux processus.

Modélisation

Aspect temporel

- ♦ Deux approches : approche *synchrone* et approche *asynchrone*.
- ♦ Approche synchrone consiste à supposer que :
 - Existence borne supérieure connue comme **délai de transmission d'un message**. (temps nécessaire de l'émission, la transmission et la réception du message)
 - Chaque processus a une **horloge logique**.
 - Existence borne inférieure et borne supérieure connues au temps nécessaire à un processus pour exécuter une instruction de son programme.
 - Notion de **délai de garde** (*timeout*)

Modélisation Topologie

- ◆ Système réparti S : ensemble de processus P_u, P_v, \dots reliés par des liens de communications
- ◆ Si communications *bi-directionnels*, alors topologie modélisée à travers un **graphe non orienté** $G=(V,E)$
- ◆ Si communications sont *unidirectionnelles*, alors utilisation d'un **graphe orienté**
- ◆ Processus P_u : modélisé par le **sommet** u
- ◆ Lien de communication entre P_u et P_v : modélisé par **arête** (u,v)

Modélisation Topologie

- ◆ Si topologie ***non connexe***, alors aucune communication d'une composante connexe à une autre. (étude en plusieurs morceaux indépendants)
- ◆ Par la suite, topologie ***connexe***.
- ◆ Topologie *influence fortement* les propriétés générales d'un système
 - Techniques de diffusion, tolérance aux fautes, complexité...
- ◆ Souvent plus facile de raisonner sur des topologies bien connues : **arbre** ou **anneau**

Modélisation Communications

- ◆ Communications par envoi de messages ou par rendez-vous
- ◆ Si envoi de messages :
 - *délai* entre l'envoi et la réception du message
 - forme de stockage des messages émis, qu'ils soient en transit ou en attente de traitement.
 - cet espace de stockage est appelé une **queue**
- ◆ **Capacité** d'un lien de communication = taille de la queue
 - En pratique, taille finie, mais modélisation avec capacité infinie.

Modélisation Communications

- ♦ Si queue de réception peut contenir plus d'un message
→ **FIFO** (si les messages sont reçus dans le même ordre que celui de leur émission)
- ♦ Possible que les messages arrivent ainsi dans un ordre quelconque → important de les **réordonner** à l'arrivée (en utilisant par exemple un numéro unique par message)
- ♦ **Délai de délivrance** : peut être connu ou non, borné ou non. On admet que ce délai est fini, i.e., la communication est sûre (le message arrivera tôt ou tard à destination)

Modélisation Processus

- ◆ Système réparti ***uniforme*** si les codes locaux de chacun des processus sont identiques.
- ◆ Si aucun moyen de distinguer les processus, le système est ***anonyme***.
- ◆ Noms (***identifiants***) des sites : généralement choisis pour être ***comparés*** (des *nombres*). Si absence d'un ordre implicite, comparaison de deux identifiants en utilisant l'*ordre lexicographique*.
- ◆ L'ensemble des identifiants peut donc toujours être considéré comme un ***ensemble totalement ordonné***.

Modélisation Synchronisation

- ◆ Présence ou non de synchronisation conditionne les algorithmes, les preuves et les implémentations.
- ◆ Système ***synchrone*** : si son fonctionnement est organisé en phases globales.

Processus synchronisés pour que leurs actions aient lieu simultanément durant une même phase globale.

- ◆ Système ***asynchrone*** : si pas de telles synchronisations.
- ◆ Si absence de telles phases globales, forme de synchronisation *induite par les communications*.

Modélisation Synchronisation

- ◆ Systèmes asynchrones : distinction envois de messages synchrones et envois asynchrones.
- ◆ ***Communication synchrone : si l'envoi et la réception sont bloquantes***

La communication synchronise l'expéditeur et le destinataire (rendez-vous).

- ◆ ***Communication asynchrone*** : l'envoi n'est pas bloquant.

Temps de délivrance du message est arbitraire, mais fini.

Réception peut être bloquante ou non, selon les implémentations.



ANNEXE

Horloges logiques

Ordre des événements et horloges logiques

- ♦ Dans un système réparti : ***pas d'horloge globale.***
- ♦ Pour avoir une horloge, il faut imposer un ***ordre des événements.***
- ♦ Utilisation d'***horloges logiques.***

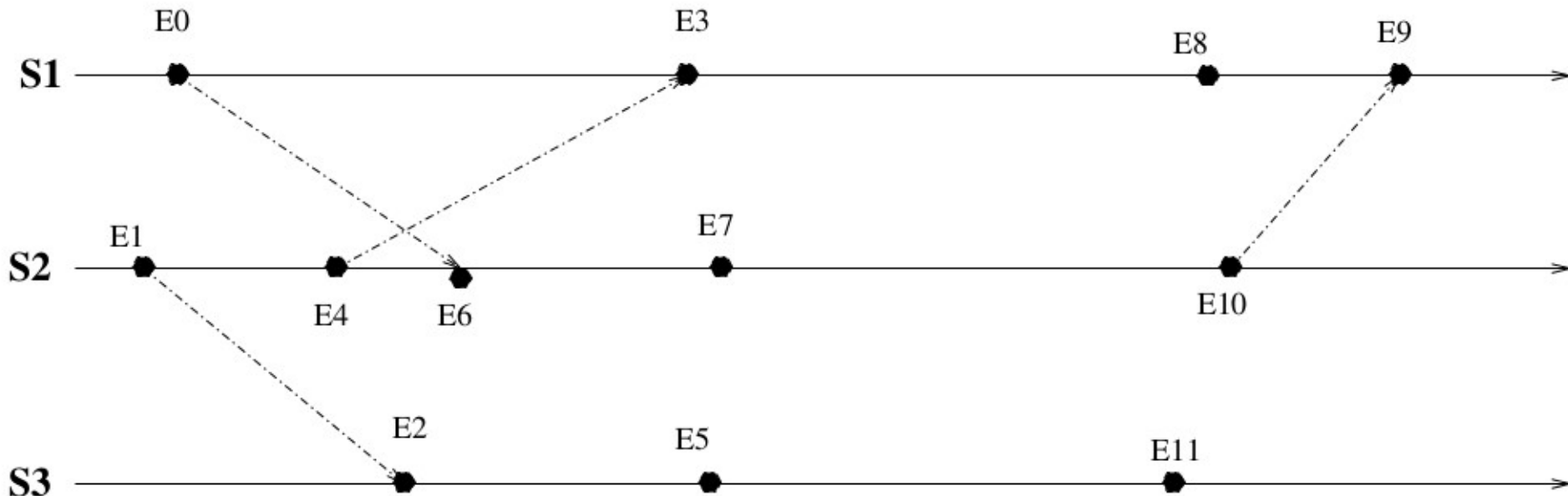
Basées sur les communications.

- ♦ 3 types d'événements :
 - émission,
 - réception,
 - événement interne.

Ordre des événements et horloges logiques

- ♦ ***Chronogramme*** décrit l'ordonnancement temporel des événements des processus et des échanges de messages
- ♦ Chaque processus est représenté par une ligne
- ♦ 3 types d'événements signalés sur une ligne
 - Émission d'un message à destination d'un autre processus
 - Réception d'un message venant d'un autre processus
 - Événement interne dans l'évolution du processus
- ♦ Messages échangés doivent respecter la topologie de liaison des processus via les canaux

Ordre des événements et horloges logiques



Ordre des événements et horloges logiques

- ◆ Soient deux événements x et y .
- ◆ x est ***causalement dépendant*** de y (noté $x \rightarrow y$) si :
 - x et y sont des événements locaux (dits internes) et x s'exécute avant y .
 - x est l'émission d'un message et y sa réception sur un site différent.
 - si $x \rightarrow z$ et $z \rightarrow y$ alors $x \rightarrow y$.
- ◆ $x \rightarrow y$ signifie que x ***précède temporellement*** y .

Ordre des événements et horloges logiques

- ♦ La ***relation de causalité*** est invariante par toute transformation des temps locaux qui
 - préserve les ordres locaux, et
 - respecte la causalité pour les messages (i.e., réception après émission).
- ♦ ***Horloges entières*** (e.g., horloge de Lamport)
- ♦ ***Horloges vectorielles*** (e.g., horloge de Mattern)
- ♦ Autres...

Horloge de Lamport

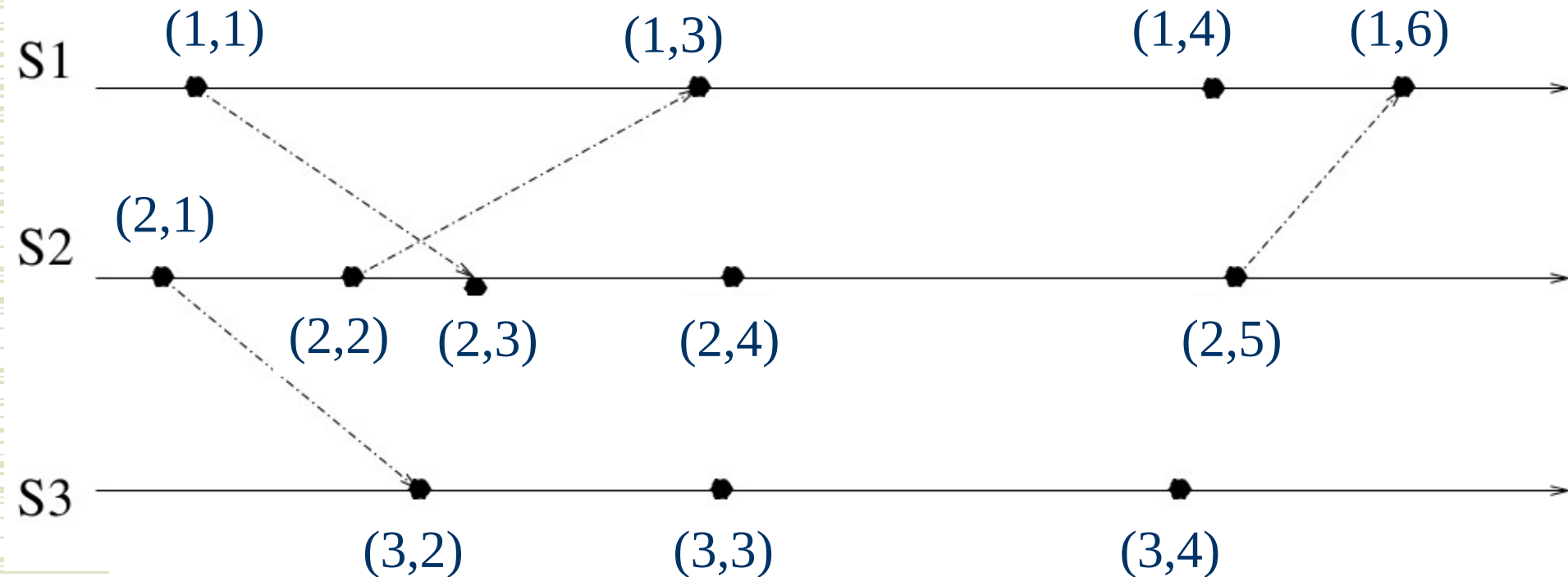
- ♦ Horloge entière : *horloge de Lamport*
- ♦ Chaque processus/site : variable entière *cpt* (un compteur)
- ♦ ***Date*** ou ***estampille*** : couple (*site*, *cpt*)
- ♦ Algorithme d'estampillage de Lamport consiste à ***affecter à chaque événement une estampille.***
- ♦ Pour un site *s* donné, le premier événement est estampillé (*s*,1).

Horloge de Lamport

- ♦ Algorithme d'estampillage de Lamport :
 - Tout message envoyé porte l'estampille de l'événement émetteur;
 - Soit e un événement interne ou un événement d'émission du site s . Soit f l'événement précédent immédiatement e sur le site s . Si f est estampillé (s, cpt) alors e est estampillé $(s, cpt+1)$;
 - Soit e un événement de réception d'un message estampillé (s', cpt') . Soit f l'événement précédent immédiatement e sur le site s . Si f est estampillé (s, cpt) , alors e porte l'estampille $(s, 1 + \max(cpt, cpt'))$.

Horloge de Lamport

- ◆ Exemple d'estampillage avec horloge de Lamport :



Horloge de Lamport

- ♦ Respecte les dépendances causales
 $e \rightarrow e' \Rightarrow H(e) < H(e')$
où $H(\dots)$ est la valeur de l'horloge
- ♦ Ordre total global
- ♦ Mais avec e et e' tq $H(e) < H(e')$, on ne peut rien dire sur la dépendance entre e et e'

Horloge de Mattern

- ◆ Assure la réciprocité
 $H(e) < H(e') \Rightarrow e \rightarrow e'$
- ◆ Permet de voir si deux événements sont en parallèle
- ◆ Par contre, pas d'ordre global total
- ◆ Chaque site S_i a un vecteur V_i initialisé à 0
 $V_i[j]$ contient la valeur de l'horloge de S_j

Horloge de Mattern

- ♦ Sur S_i , si évt interne, émission ou réception, alors $V_i[i] = V_i[i] + 1$
et si émission d'un message, alors V_i est envoyé avec
- ♦ Sur S_i , à la réception d'un message m contenant V_m , mise à jour de V_i :
qqsoit $j \neq i$ $V_i[j] = \max(V_m[j], V_i[j])$

Horloge de Mattern

- ◆ Exemple d'estampillage avec horloge de Mattern :

