Intégration des Systèmes Clients/Serveurs

- André Freyssinet
  - GIE Bull/Inria Dyade
  - Téléphone: 04.76.61.52.75
  - Email: André.Freyssinet@inrialpes.fr
  - HTTP://dyade.inrialpes.fr/~freyssin

Plan du cours

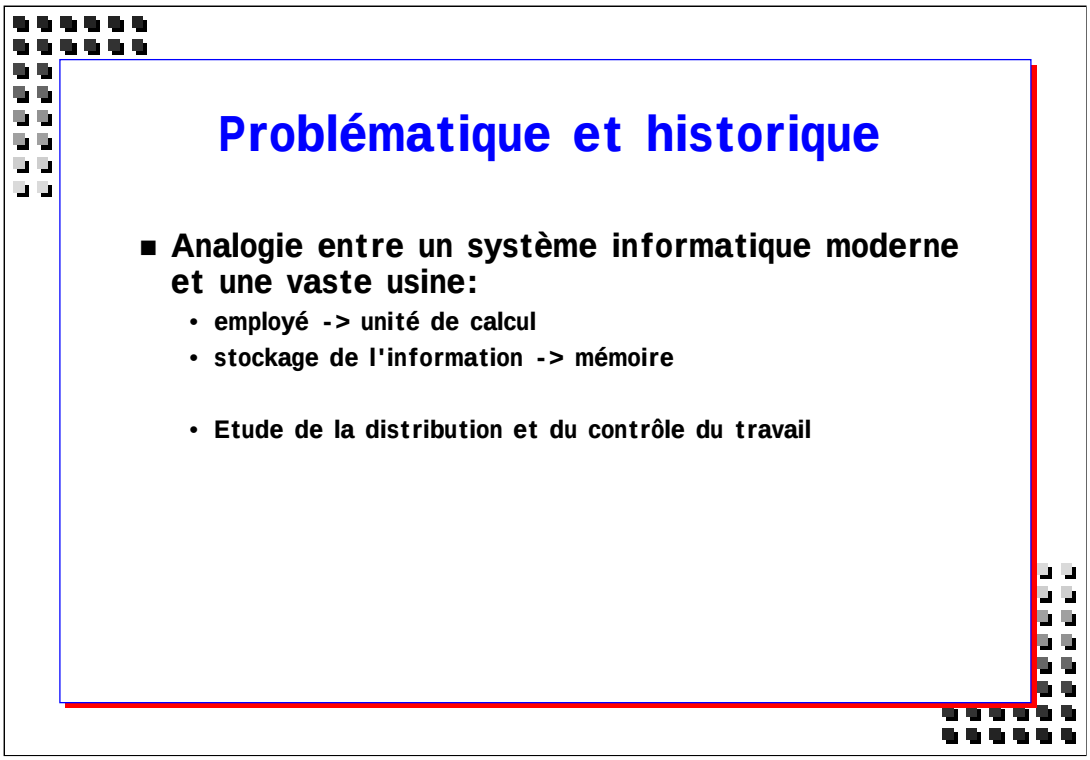
- **Introduction**
  - Analogie
  - Problèmes, propriétés
- **Modèles et mécanismes de structuration**
  - Messages,
  - Client/Serveur,
  - Code mobile, etc.

Plan du cours

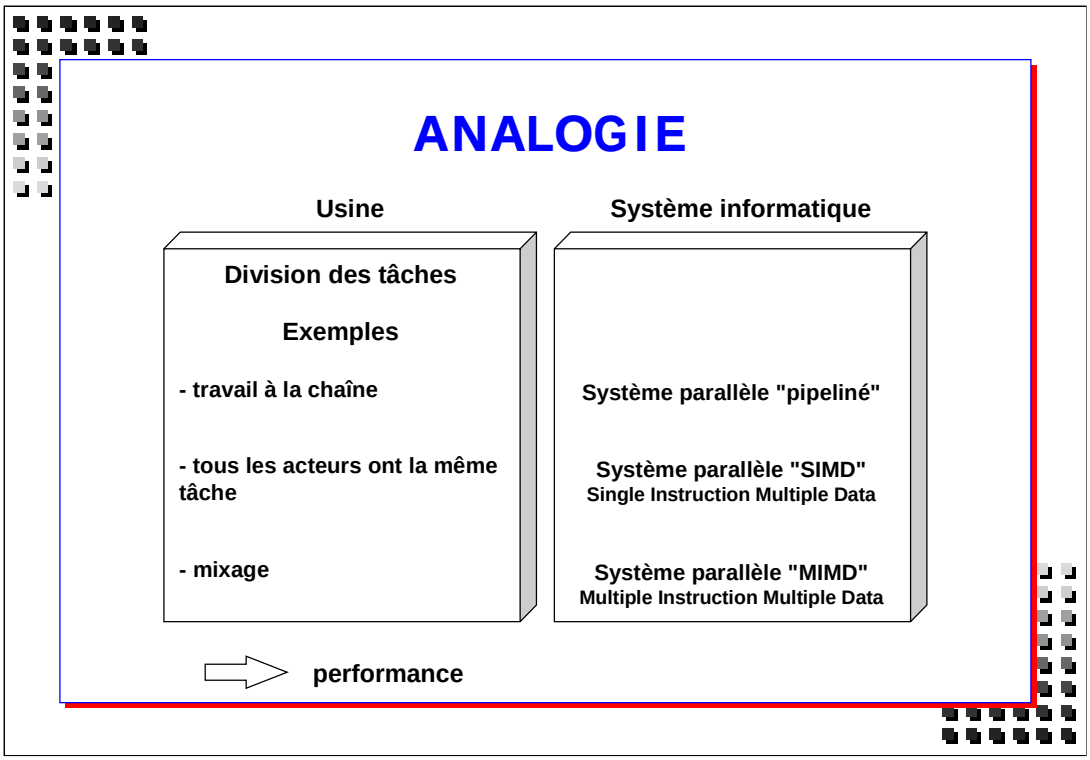
- **Ordonnancement - Synchronisation**
  - Exemple, problématique
  - Mécanismes de base
  - Exemple de solutions
  - Cohérence, etc.
- **SGF distribués**
  - NFS, DFS, etc.
- **Client/Serveur**
  - Présentation,
  - Mise en œuvre : Sockets, RPC, RMI, etc.

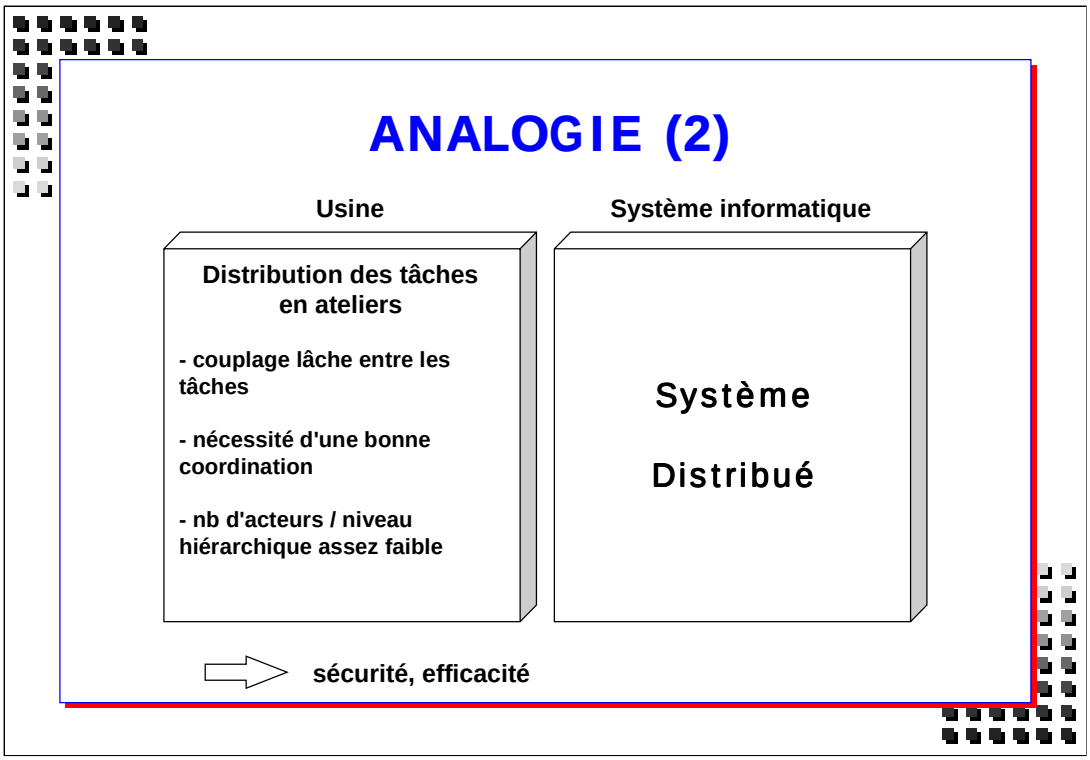
Plan du cours

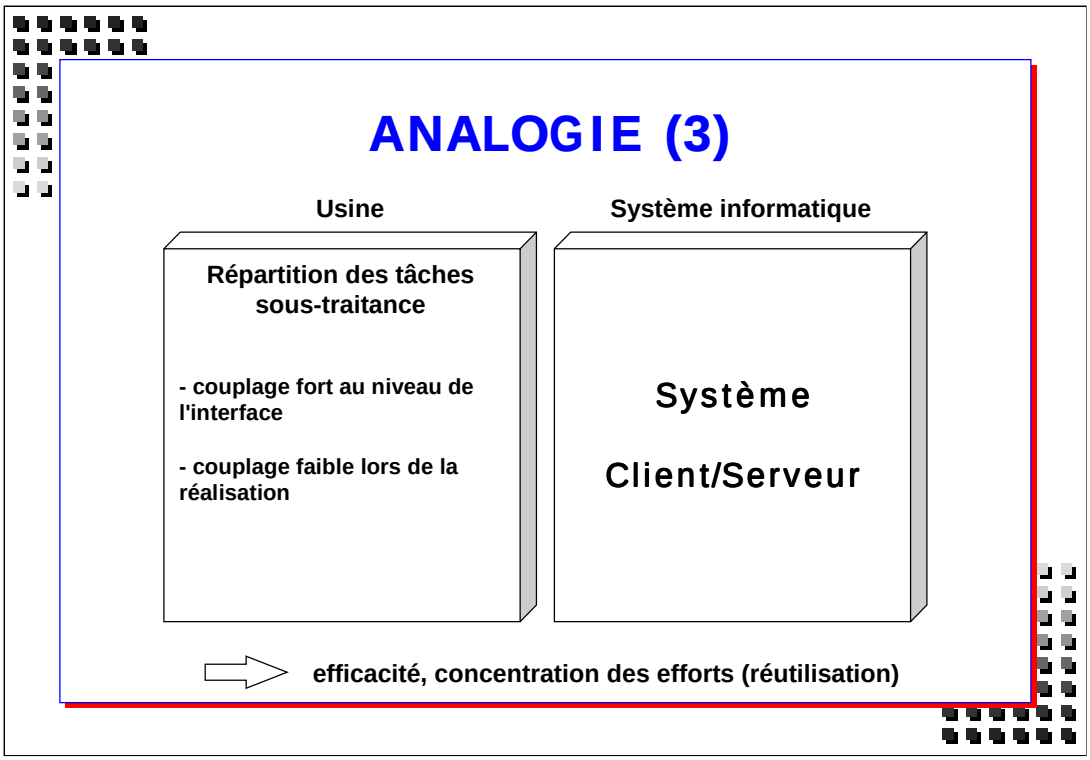
- **Message Oriented Middleware**
  - Présentation,
  - Java Message Service
- **Corba**
  - Présentation,
  - Exemple
- **Gestion répartie des transactions**
- **Tolérances aux fautes, Sécurité**
- **Internet, le Web, Java, etc.**



Le but de ces quatre premiers transparents est de présenter le concept de Client/Serveur au travers d'une analogie avec le monde réel. On développe donc une comparaison entre un système informatique parallèle (multiples UC) et une usine. Dans les deux cas on étudie les différentes solutions pour distribuer et contrôler le travail.


ANALOGIE
Usine
Système informatique
Division des tâches
Exemples
- travail à la chaîne
- tous les acteurs ont la même tâche
- mixage
Système parallèle "pipeliné"
Système parallèle "SIMD"
Single Instruction Multiple Data
Système parallèle "MIMD"
Multiple Instruction Multiple Data
performance


ANALOGIE (2)
Usine
Système informatique
Distribution des tâches en ateliers
- couplage lâche entre les tâches
- nécessité d'une bonne coordination
- nb d'acteurs / niveau hiérarchique assez faible
Système
Distribué
sécurité, efficacité


ANALOGIE (3)
Usine
Système informatique
Répartition des tâches
sous-traitance
- couplage fort au niveau de l'interface
- couplage faible lors de la réalisation
Système
Client/Serveur
efficacité, concentration des efforts (réutilisation)

Rappelons brièvement les principales motivations des systèmes informatique répartis:

• Faire communiquer des applications existantes, tout en respectant l'autonomie de chacune ; la principale fonction du système est alors la communication d'information.

• Adapter la structure du système à celle des apllications qu'il traite ; de nombreuses applications sont intrinsèquement réparties, soit géographiquement (utilsateurs, données, dispositifs de commande), soit fonctionnellement.

• Augmenter la disponiblité...

• Réduire le temps d'exécution par le traitement en parallèle de certaines tâches sur des machines différentes.

• Partager des ressources entre un ensemble d'utilisateurs.

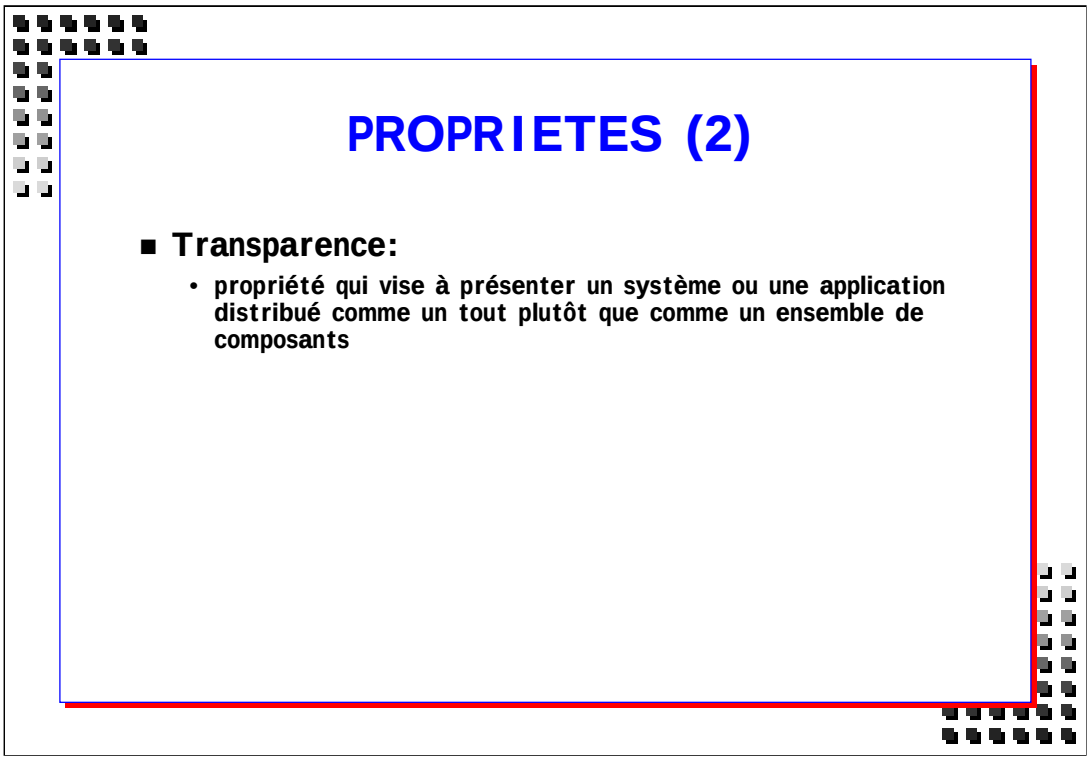
• Faciliter l'évolution progressive par la modularité d'un système constitué d'éléments indépendants.

Une application répartie pour répondre aux motivations énoncées doit posséder différentes propriétés, ces propriétés peuvent sembler inhérentes à l'aspect distribué de l'application, il n'en est rien :

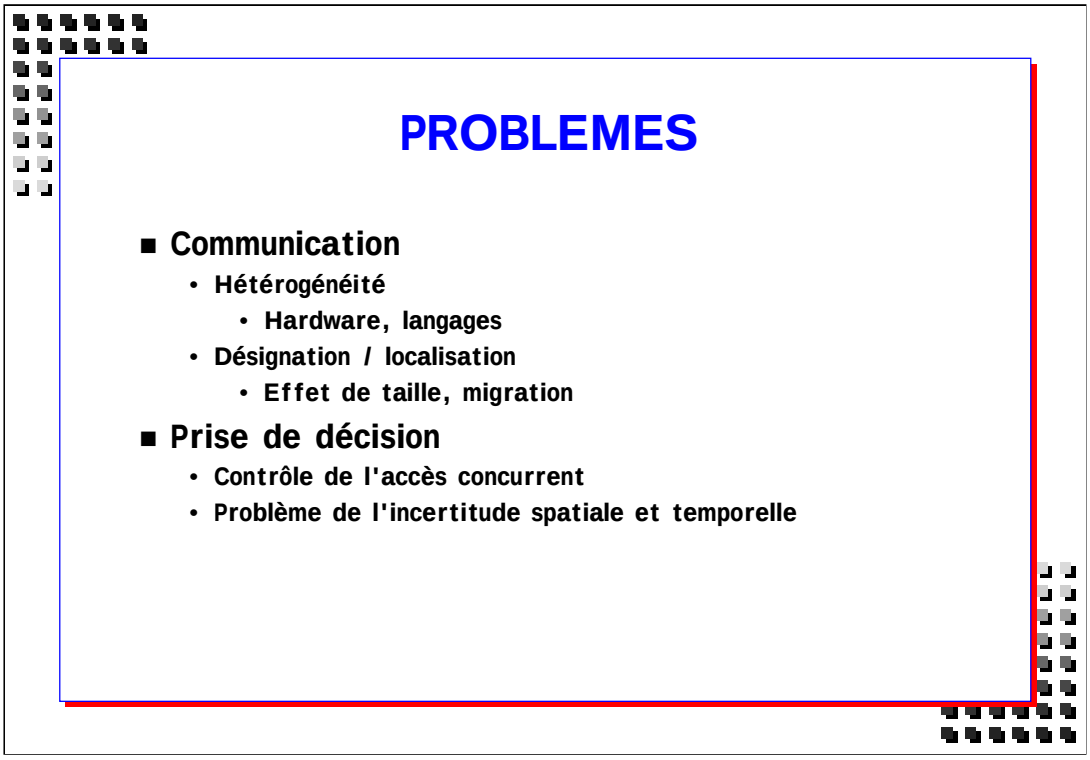
• **Parallélisme** : l'application est réalisée par de multiples processus (clients et serveurs) parallèles ; cependant son bon fonctionnement nécessite une synchronisation entre ces processus, une synchronisation trop forte peut conduire à la sérialisation de l'application.

• **Dimensionnement** (scalabilité) : cette propriété est proche de la précédente, elle exprime surtout le surcoût lié à l'ajout d'un client ou d'un serveur : par exemple le coût lié à la synchronisation ou à l'accès des données distantes peut devenir prohibitif à partir d'un certain nombre de processus.

• **Tolérance aux fautes** (fiabilité) : cette propriété exprime la capacité de l'application à s'exécuter dans un environnement dégradé (composants en panne, liens de communication absents, etc.). Il faut être vigilant et éviter d'isoler des fonctions vitales.



Cette propriété est très différente des précédentes, elle exprime surtout le caractère intégré de l'application. Une bonne transparence permettra de cacher la complexité de l'application.



Les problèmes liés à la réalisation d'applications réparties sont nombreux, on peut cependant citer les principaux. Dans la communication entre les différentes entités composant l'application, l'aspect passage d'information entre deux processus est simple (envoi de message par exemple : sockets), les difficultés sont liées d'une par à la désignation et à la localisation de ces entités, d'autre part à leurs différences :

• désignation, localisation d'une ressource, d'un serveur,

• hétérogénéité : différence de représentation des données donnant lieu à des interprétations différentes du contenu d'un message.

La prise de décision, et tous les problèmes proches (synchronisation, contrôle d'accès concurrents), est rendue difficile dans les systèmes répartis du fait que les différents acteurs ont une perception différente de l'état de l'application : en effet chacun perçoit l'état de tous les autres par le biais des messages échangés, il a donc au mieux une vision retardée de celui-ci.

L'ensemble de ces problèmes sera abordé ultérieurement, les différents outils et paradigmes permettant de solutionner le problème de la communication sont présentés ci-après, le problème de la prise de décision et ses solutions sont abordés ensuite.

Modèles et Mécanisme de Structuration

Problématique

- **Application répartie = traitements coopérants sur des données réparties**
- **Coopération = communication+synchronisation**
  - **modèle d 'exécution**
  - **interface de programmation (et/ou langage)**
  - **outils de développement**
  - **mise en œuvre : services systèmes**

Modèles d'exécution

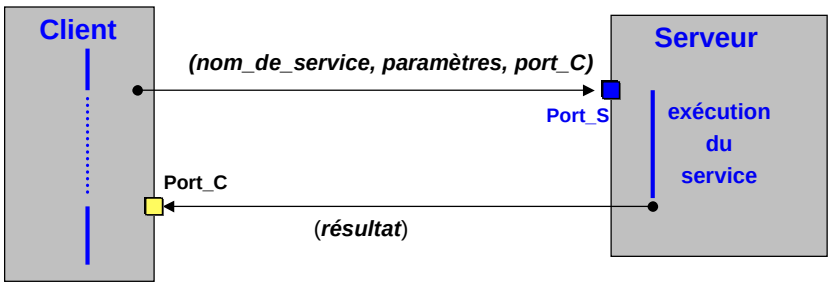
- communication par messages
- événements/réactions
- demande de service : mode client-serveur
  - client-serveur "traditionnel"
  - client-serveur "de données"
  - client-serveur "à objets"
  - client-serveur "à composants"
- code mobile
- mémoire virtuelle partagée
- autres
  - "peer to peer", transactionnel, . .

Communication par messages
Principes directeurs

- **Principes directeurs**
  - communication asynchrone
  - problème de désignation des entités coopérantes
  - messages éventuellement typés
- **Interface de programmation et mise en œuvre**
  - interface “socket” sur le niveau transport
  - primitives de communication élémentaires: send, recv.

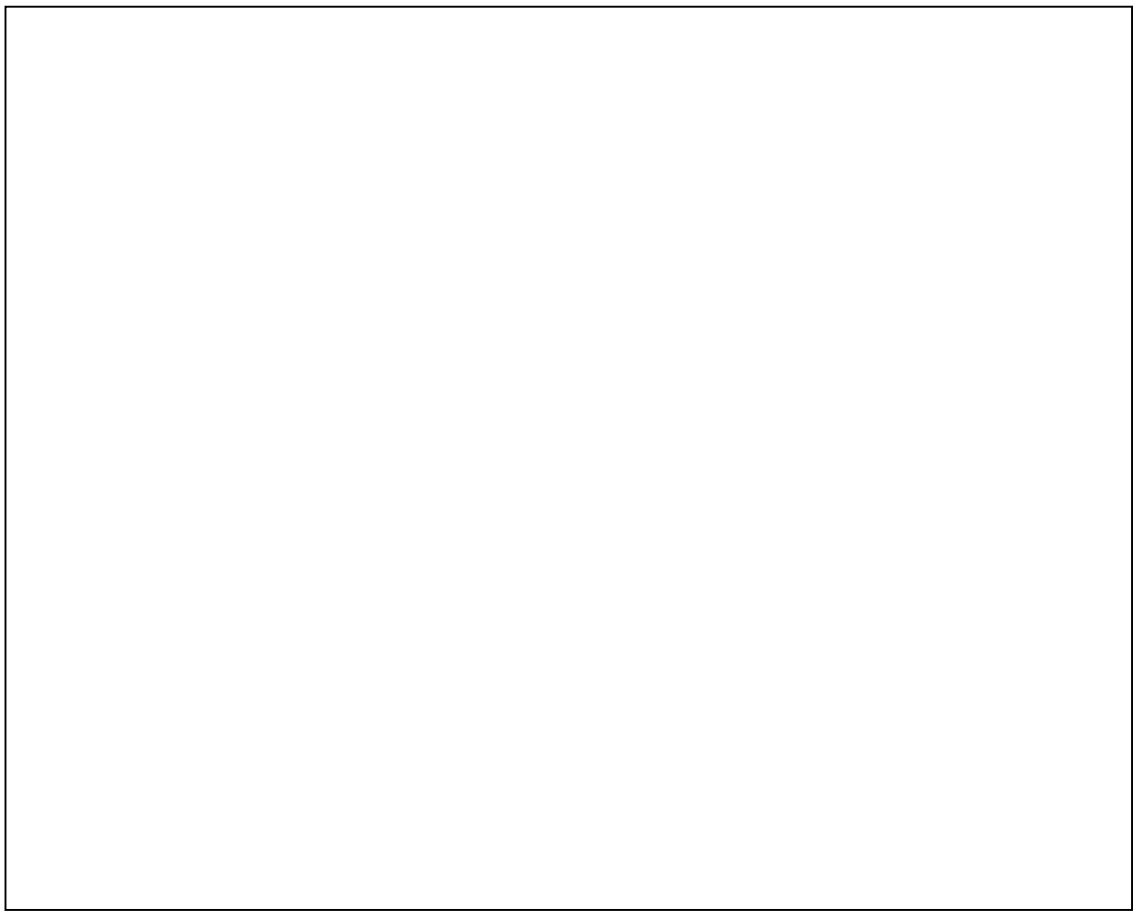
Communication par messages Exemple

- Réalisation d'une interaction de type « client-serveur »



"Middleware" à messages

- **Business Quality Messaging**
  - messages (attributs)
    - identification unique
    - typage, persistance
    - modes de délivrance
  - queues de messages
    - partagées par les applications
    - modes de réception variable
  - transactions
    - messages vus comme des ressources "transactionnelles"
  - sécurité



Communication par messages
Extensions du modèle

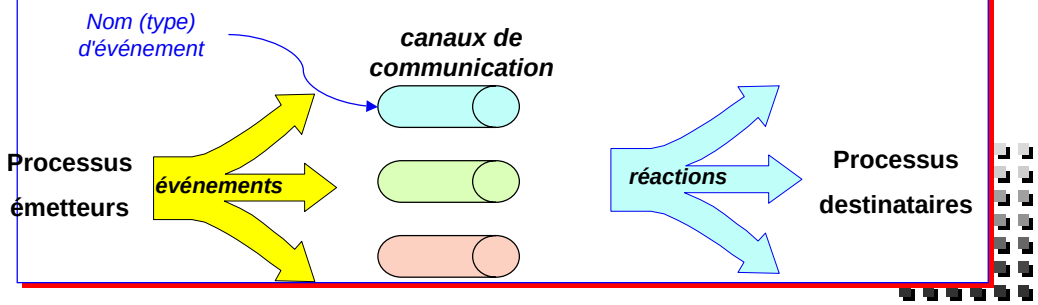
- **Communication de groupe**
  - groupe : ensemble de destinataires identifiés par un nom unique
  - gestion dynamique du groupe : arrivée/départ de membres
  - différentes politiques de service dans le groupe : 1/N, N/N
  - mise en œuvre : utilisation possible de IP multicast
  - exemple : Isis, Horus, Ensemble (Cornell university)
  - applications : tolérance aux fautes (gestion de la réplication), travail coopératif

Communication par messages
Extensions du modèle

- **Communication anonyme**
  - **désignation associative : les destinataires d'un message sont identifiés par leurs propriétés et non par leur nom**
  - **propriété : attribut du message ou identificateur externe**
  - **indépendance entre émetteur et récepteurs**

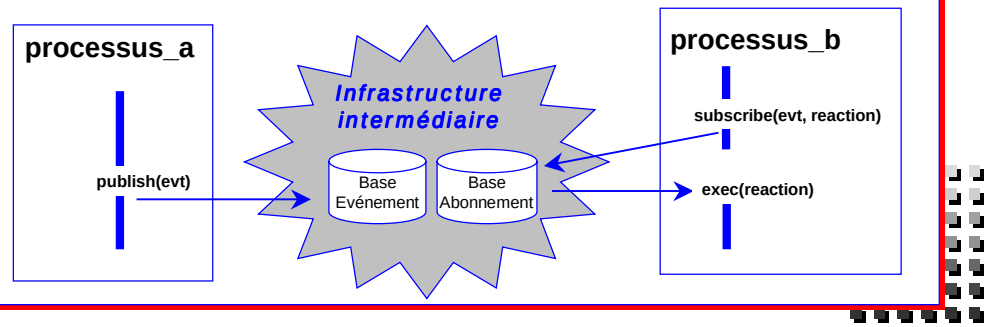
Communication événementielle principes de fonctionnement

- concepts de base : événements, réactions (traitements associés à l'occurrence d'un événement)
- principe d'attachement : association dynamique entre un nom d'événement et une réaction
- communication anonyme : indépendance entre l'émetteur et les "consommateurs" d'un événement



Communication événementielle : principes de réalisation

- serveur centralisé ("Hub and Spoke")
- serveur réparti ("Snowflake")
- service réparti (bus logiciel)



Communication événementielle : conclusion

- **Domaines d'application**
  - **génie logiciel (coopération entre outils de développement) : SoftBench, ToolTalk, etc.**
  - **Workflow : KoalaBus, etc.**
  - **diffusion de logiciels et d'information sur le Web : iBus, Castanet, Ambrosia, TIB/Rendezvous**
- **Infrastructures propriétaires**
  - **interface applicative et protocoles propres à chaque système -> problèmes de portabilité et d'interopérabilité**
  - **effort de normalisation (ECMA)**
- **Outils de développement**
  - **sommaires**

Modèle client-serveur
principes directeurs

Client

émet des requêtes (demandes de service)

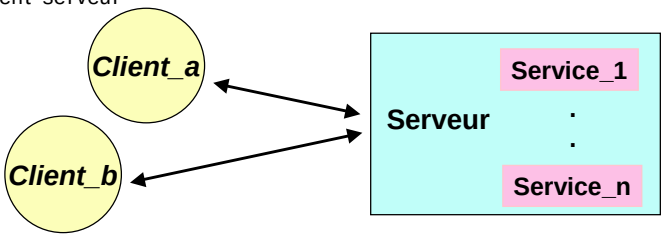
mode d'exécution *synchrone*

le client est l'initiateur du dialogue client-serveur

Serveur

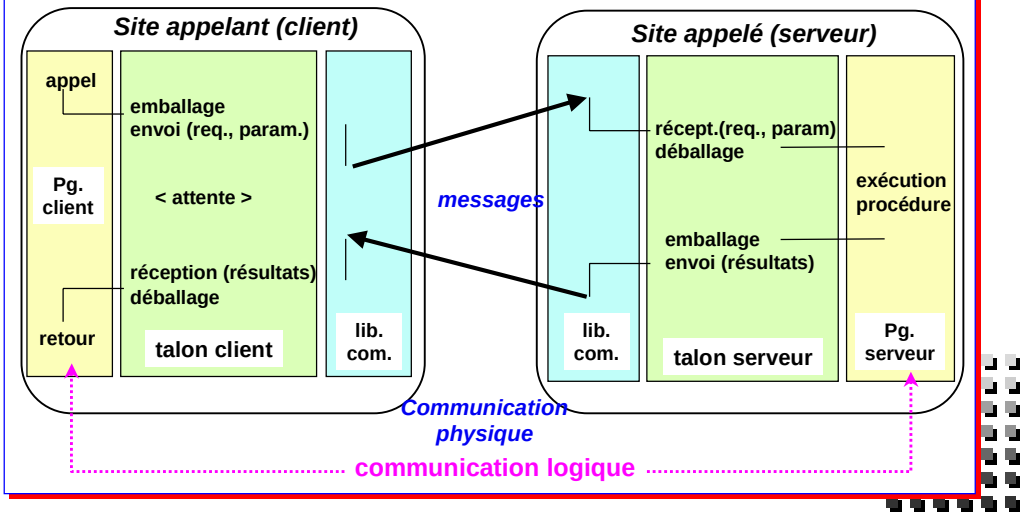
ensemble de services exécutables

exécute des requêtes émises par des clients --> exécution des services (exécution séquentielle ou parallèle)



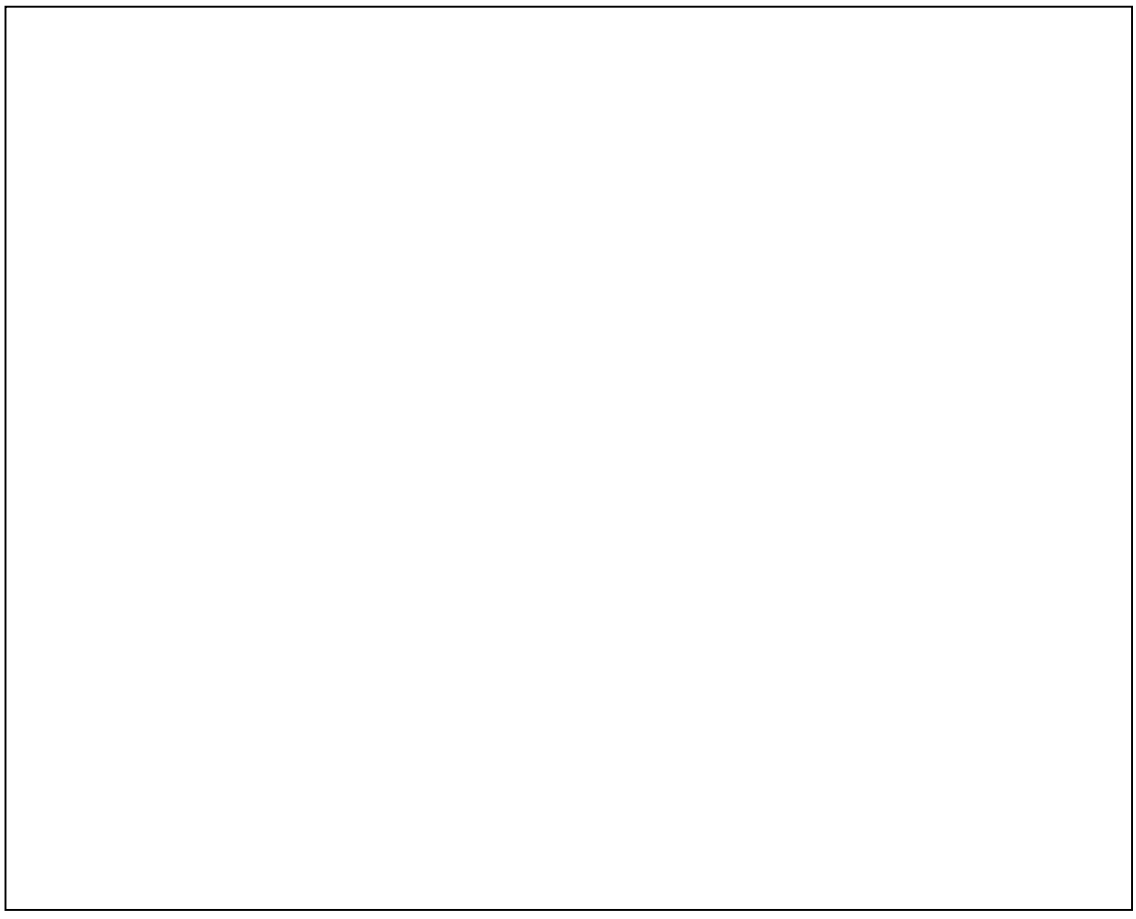
Exemples : **serveur de fichiers, serveur de base de données, serveurd'impression, serveur de noms (annuaire des services)**

Modèle client-serveur
Appel de procédure à distance



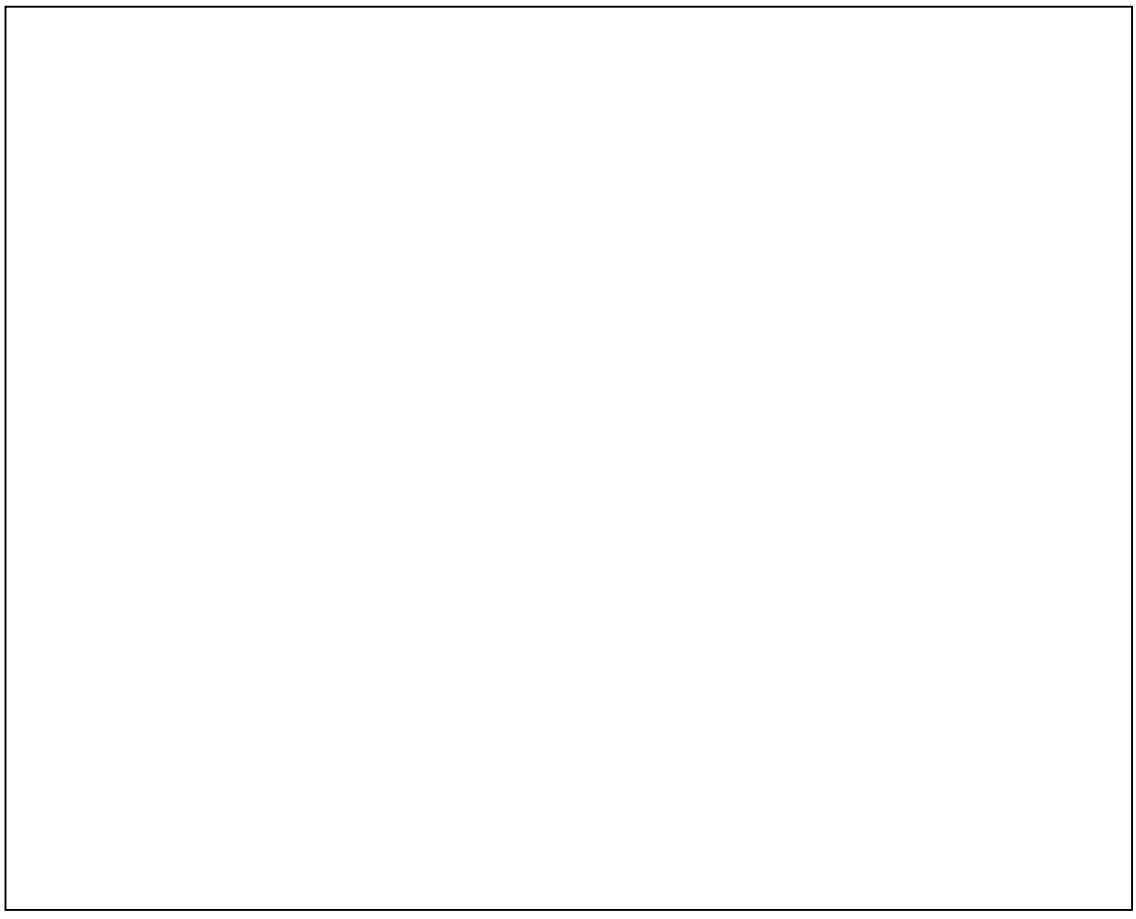
Modèle client-serveur
Appel de procédure à distance

- **fonction des talons**
  - emballage-déballage des paramètres et résultats
  - conversion de données
  - gestion des processus exécutants (serveur)
  - gestion des erreurs (mise en œuvre “sémantique” RPC)
- **fonction des modules de communication**
  - transmission des paramètres et résultats “emballés”
  - gestion des erreurs de communication
- **liaison (détermination de l’adresse de l’appelé)**
  - adresse fixe connue lors de la génération du talon
  - désignation logique + appel d’un serveur de noms



Modèle client-serveur
Outils de développement

- **Langages de description d'interface (IDL)**
  - spécification commune au client et au serveur
  - définit le type des paramètres et résultats
  - indépendant des langages de programmation
- **Compilateur d'interface**
  - à partir d'une définition d'interface (en IDL) produit les talons client et serveur pour un langage donné

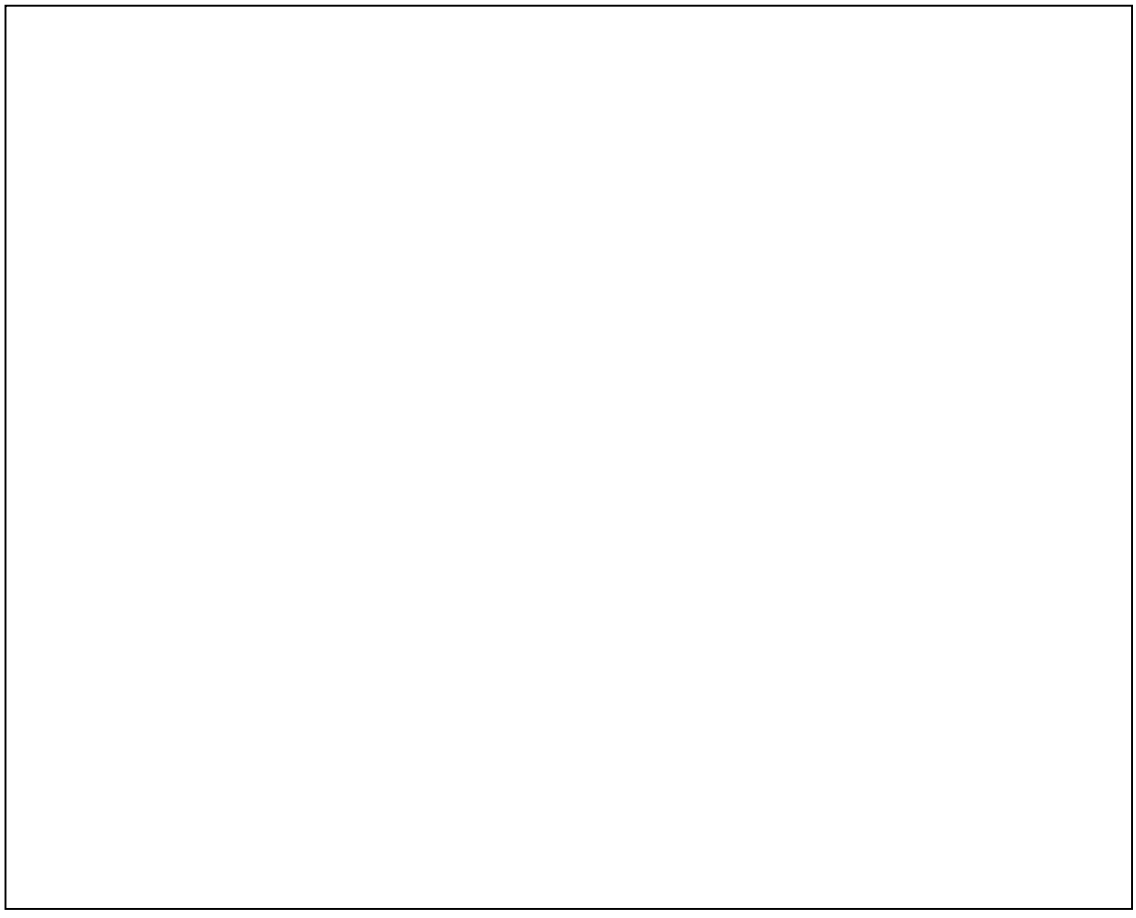


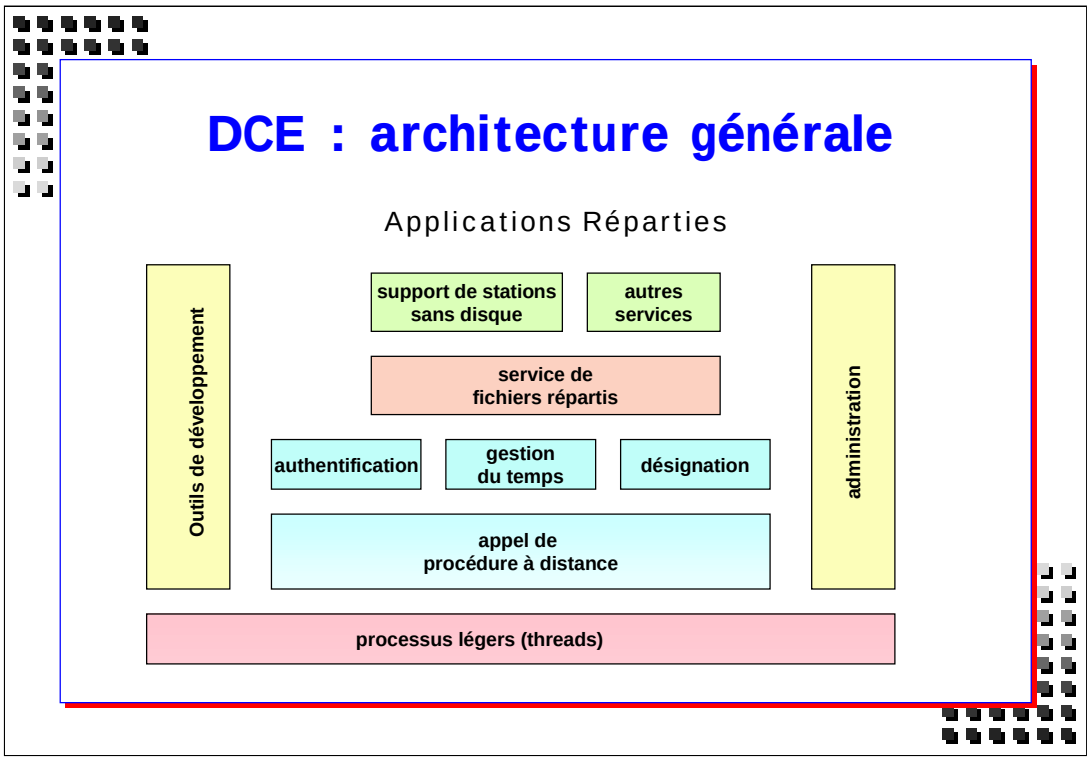
Modèle client-serveur
Les limites

- **Modèle de structuration**
  - permet de décrire l'interaction entre deux composants logiciels (absence de vision globale de l'application)
  - schéma d'exécution répartie élémentaire (synchrone, absence de propriétés portant sur la synchronisation, la protection, la tolérance aux pannes, etc.)
- **Services pour la construction d'applications réparties**
  - le RPC est un mécanisme de "bas niveau"
- **Outils de développement**
  - limités à la génération automatique des talons
  - peu d'outils pour le déploiement et la mise au point d'applications réparties

Environnements client-serveur

- environnement client-serveur "traditionnel"
  - Distributed Computing Environment : un environnement “intégré” d’applications réparties, fondé sur le modèle client-serveur
- environnement client-serveur "à objet"
  - CORBA : un environnement fondé sur un "courtier" d'objets répartis et un ensemble de services applicatifs
- environnement client-serveur "de données"
  - exécution de requêtes SQL
- environnement client-serveur sur le Web
- environnement client-serveur "à composant"
  - COM: un environnement pour la gestion de documents composites




DCE : architecture générale
Applications Réparties
Outils de développement
support de stations sans disque
autres services
service de fichiers répartis
authentification
gestion du temps
désignation
appel de procédure à distance
administration
processus légers (threads)

DCE : description des services

- **Processus légers (threads)**
  - définis pour assurer la portabilité des applications DCE
  - permet la multi-programmation des appels RPC
- **Appel de procédure à distance**
  - langage de description d'interface et son compilateur
  - mis en œuvre sur TCP/IP
- **Service distribué de gestion du temps**
  - synchronisation des horloges des différentes machines du système avec le temps universel coordonné.

DCE : description des services

- **Service de noms (Directory Service)**
  - gestion des informations (association nom/attributs) sur les ressources disponibles du système distribué.
  - service structuré en deux niveaux :
    - Un espace local à une cellule ou CDS qui maintient les informations locales à une cellule (groupe de machines)
    - Un espace global ou GDS qui permet d'interconnecter les espaces de noms des cellules entre eux dans une même hiérarchie

DCE : description des services

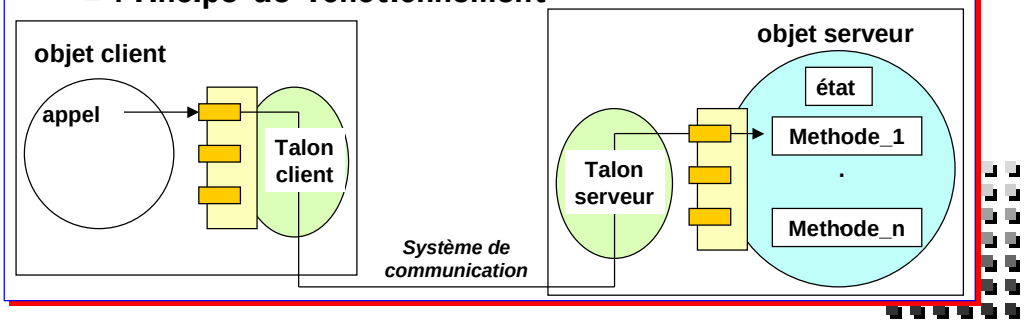
- **Service de sécurité**
  - authentification
  - sécurisation des communications
  - contrôle d'accès (autorisation)
- **Service de fichiers distribués (DFS)**
  - permet le partage des fichiers sans connaître la localisation physique.
    - espace de noms global
  - DCE DFS comprend un système physique de fichiers
    - qui permet la duplication, la journalisation et la gestion de listes d'accès.

Atouts et limites de DCE

- **architecture de type “boîte à outils”**
  - chaque fonction est utilisable via une interface “normalisée”
  - accès direct à des fonctions évoluées
    - ---> intégration “à gros grain”
- **“standard” de fait (diffusion dans la communauté industrielle)**
  - ex. : RPC, Service de noms, service d’authentification
    - ---> portabilité et interopérabilité
- **développement d’applications “à grain fin” laborieux**
  - ex. : utilisation des services de thread et du RPC (limitations équivalentes à celles du client-serveur)
  - peu d’outils de développement disponibles aujourd’hui

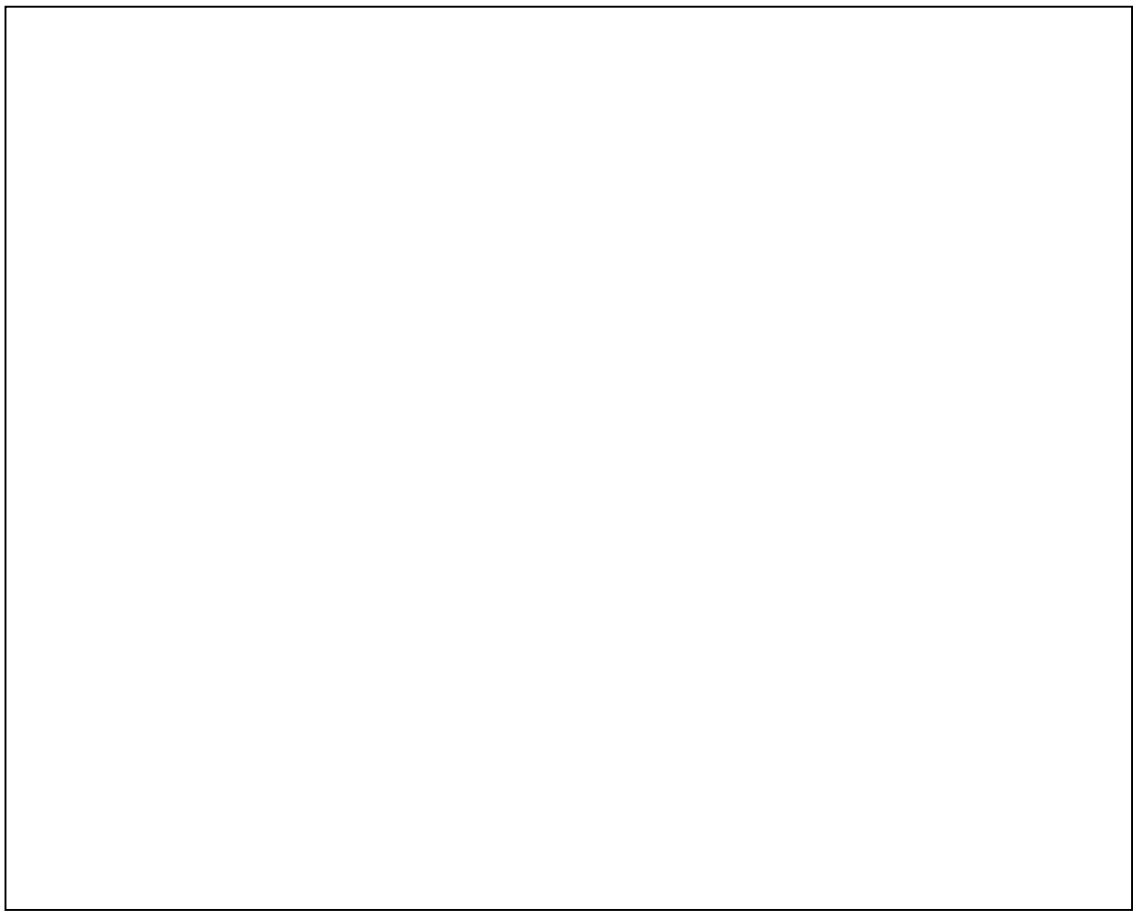
Client-serveur "à objet"

- **Motivations**
  - propriétés de l'objet (encapsulation, modularité, réutilisation, polymorphisme, composition)
  - objet : unité de désignation et de distribution
- **Principe de fonctionnement**

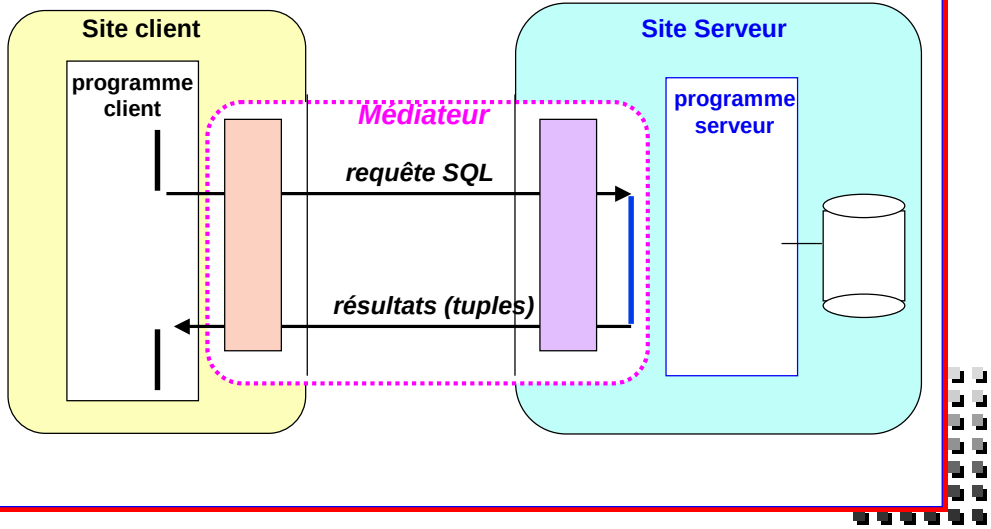


Appel de méthode à distance

- **éléments d'une "invocation"**
  - référence d'objet ("pointeur" universel)
  - identification d'une méthode
  - paramètres d'appel et de retour (et exception)
    - passage par valeur : types élémentaires construits
    - passage par référence
- **objets "langage"**
  - représentation propre au langage : instance d'une classe (exemple Java RMI)
- **objets "système"**
  - représentation "arbitraire" définie par l'environnement d'exécution (exemple CORBA)

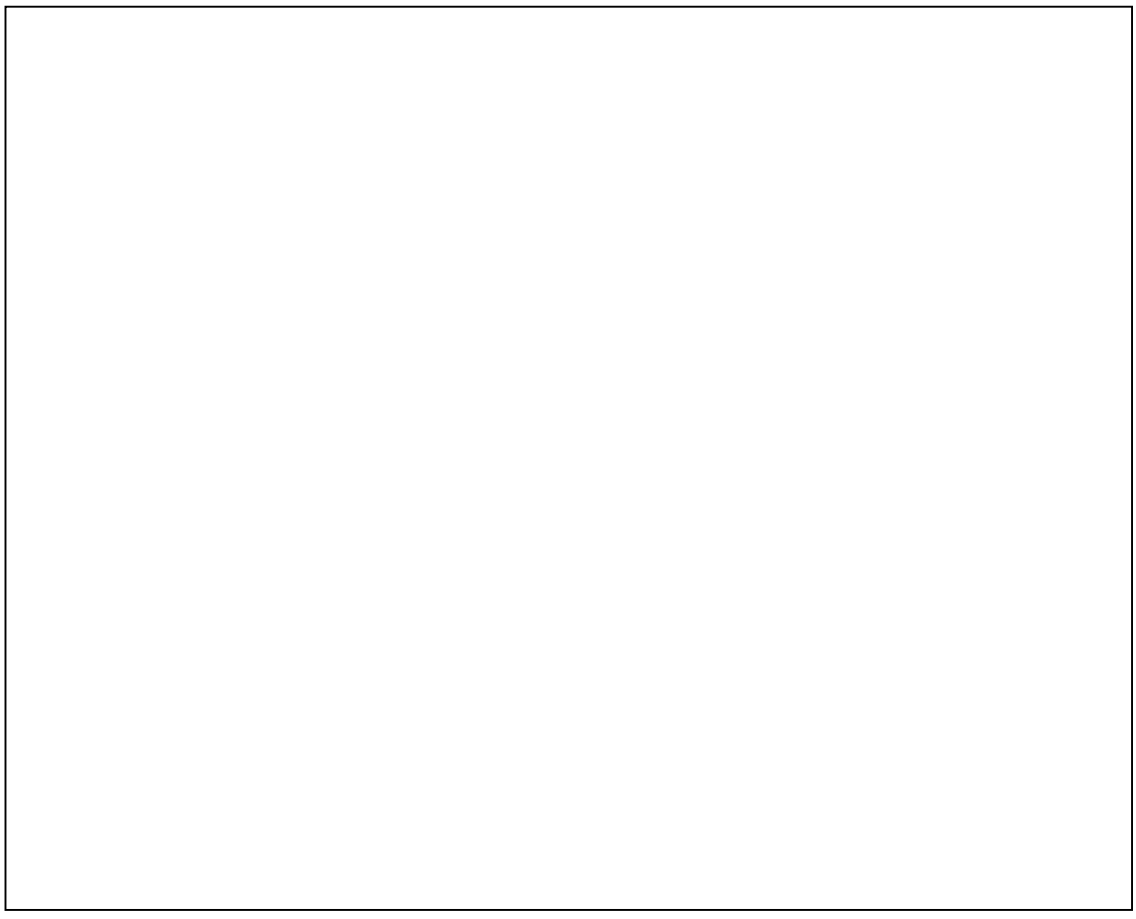


le client-serveur de données
Principes directeurs



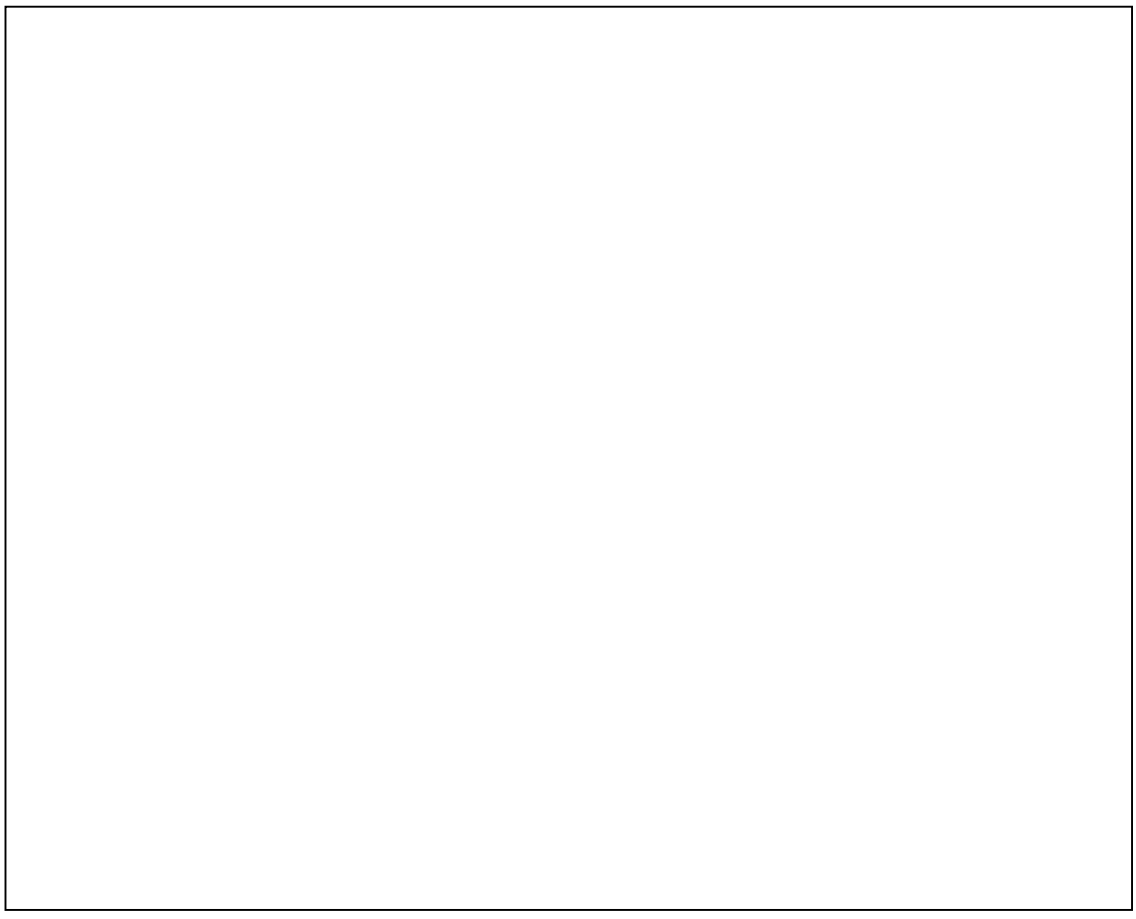
Le client-serveur de données
Principes directeurs

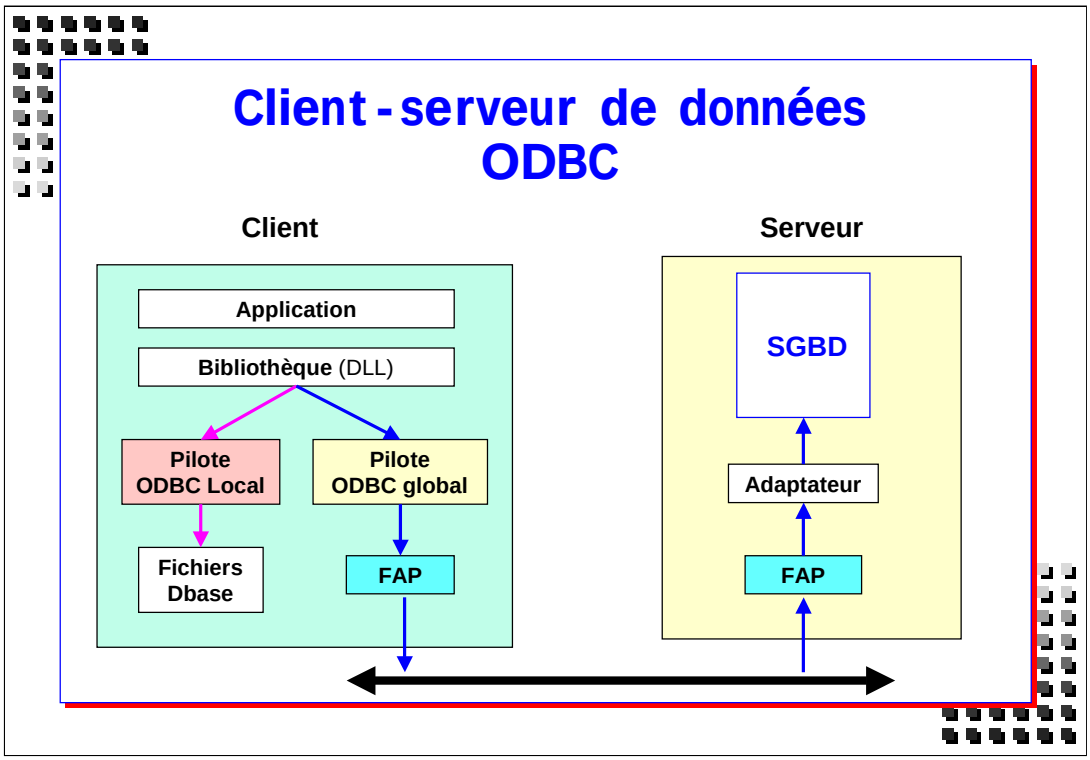
- **fonction du client**
  - code de l'application non lié aux données
  - dialogue avec l'utilisateur
- **fonction du serveur**
  - stockage des données, gestion de la disponibilité et de la sécurité
  - interprétation/optimisation des requêtes
- **fonctions du “médiateur”**
  - procédures de connexion/déconnexion
  - préparation/communication de requêtes
  - gestion de caches (requêtes et résultats)



Le client-serveur de données
Environnement applicatif

- **Environnement Applicatif Commun (CAE) de l'X/OPEN**
  - objectif : portabilité des applications
  - interface applicative CLI (Call Level Interface)
    - standardisation des appels SQL depuis un programme (C, Cobol, . .)
    - connexion/déconnexion, préparation/exécution des requêtes, . . .
- **Outils de développement**
  - L4G : accès aux tables, contrôle, affichage, saisie, …
    - indépendants des langages de programmation




Client - serveur de données ODBC
Client
Serveur
Application
Bibliothèque (DLL)
Pilote ODBC Local
Pilote ODBC global
Fichiers Dbase
FAP
SGBD
Adaptateur
FAP

Client-serveur de données "à objet" ODMG

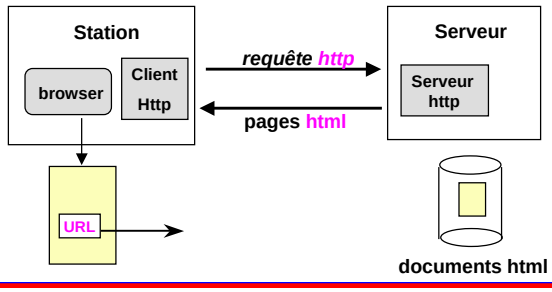
- **Objectifs**
  - Portabilité des sources d'une application sur divers "moteurs" de bases de données à objet
- **Principes directeurs**
  - modèle objet = sur-ensemble du modèle OMG
  - un langage de Définition de Données : ODL (basé sur l'IDL OMG)
  - un langage de Requêtes : OQL, évolution de SQL
  - intégration complète des mécanismes dans le langage hôte : C++ (avec propositions d'évolution de la norme) et Smalltalk

Environnements client-serveur
Le Web

- le Web “traditionnel”
  - accès à l'information répartie
- le Web comme support d’applications client-serveur
  - protocole d’échange de “haut niveau”
- les documents “actifs”
  - code mobile (ex. “applets” Java)
- le Web et les objets distribués
  - convergence de 2 technologies

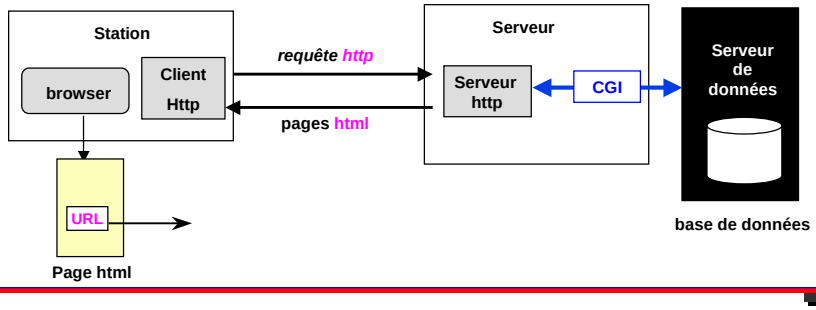
Le Web
Support pour l'accès à l'information

- un protocole client-serveur: http
- un système de désignation universel : URL
- une représentation de doc. hypertextes : html
- des navigateurs ("browser") évolués



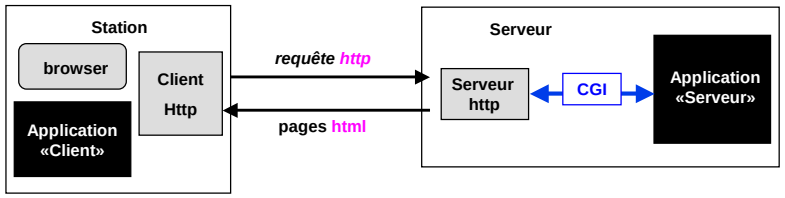
Le Web
Support pour l'accès à l'information

- **Accès à des serveurs "externes"**
  - utilisation de l'interface CGI (Common Gateway Interface) et/ou de "servlets"
    - requêtes à un serveur de données
    - transformation des résultats en pages html



Le Web
Support d'applications client-serveur

- utilisation du protocole http pour le dialogue client-serveur

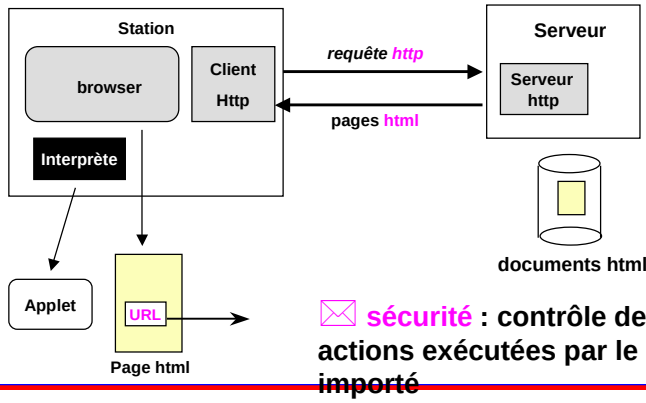


+ *avantages* :
  support universel de communication
  simplicité de la mise en œuvre

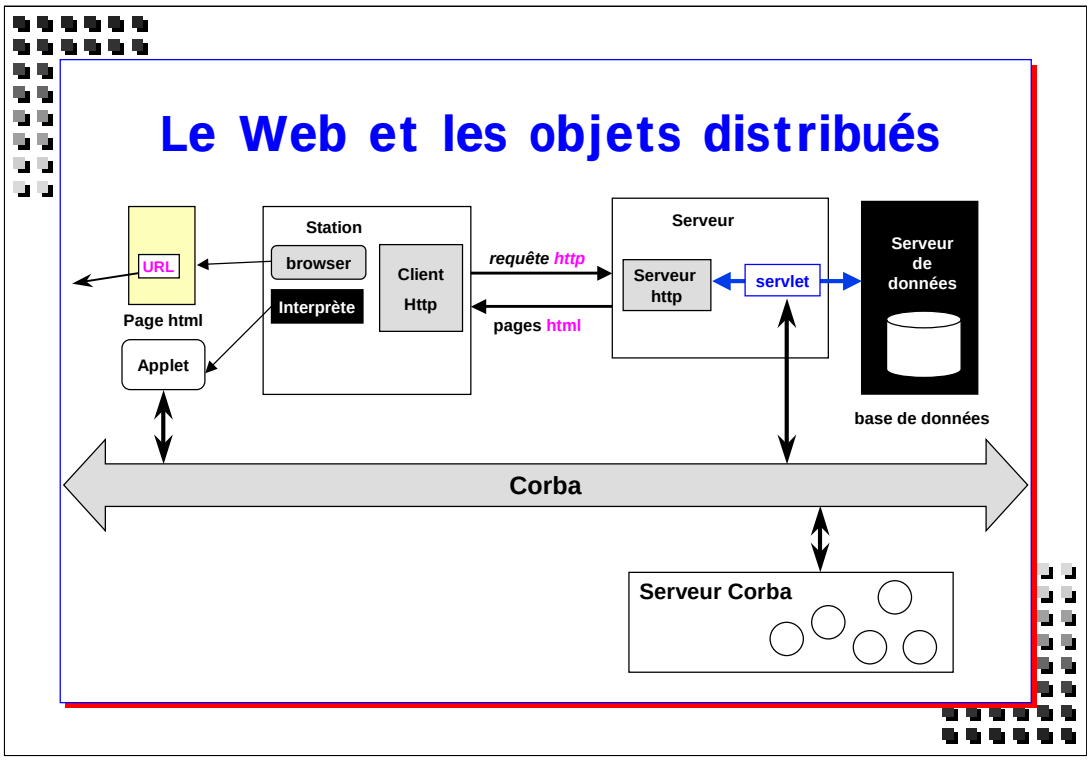
- *inconvénients* :
  primitives élémentaires (Get, Post, . .)
  serveur http non adapté

Le Web
Documents “actifs”

- séquences exécutables ("applets", "contrôles") dans les pages html

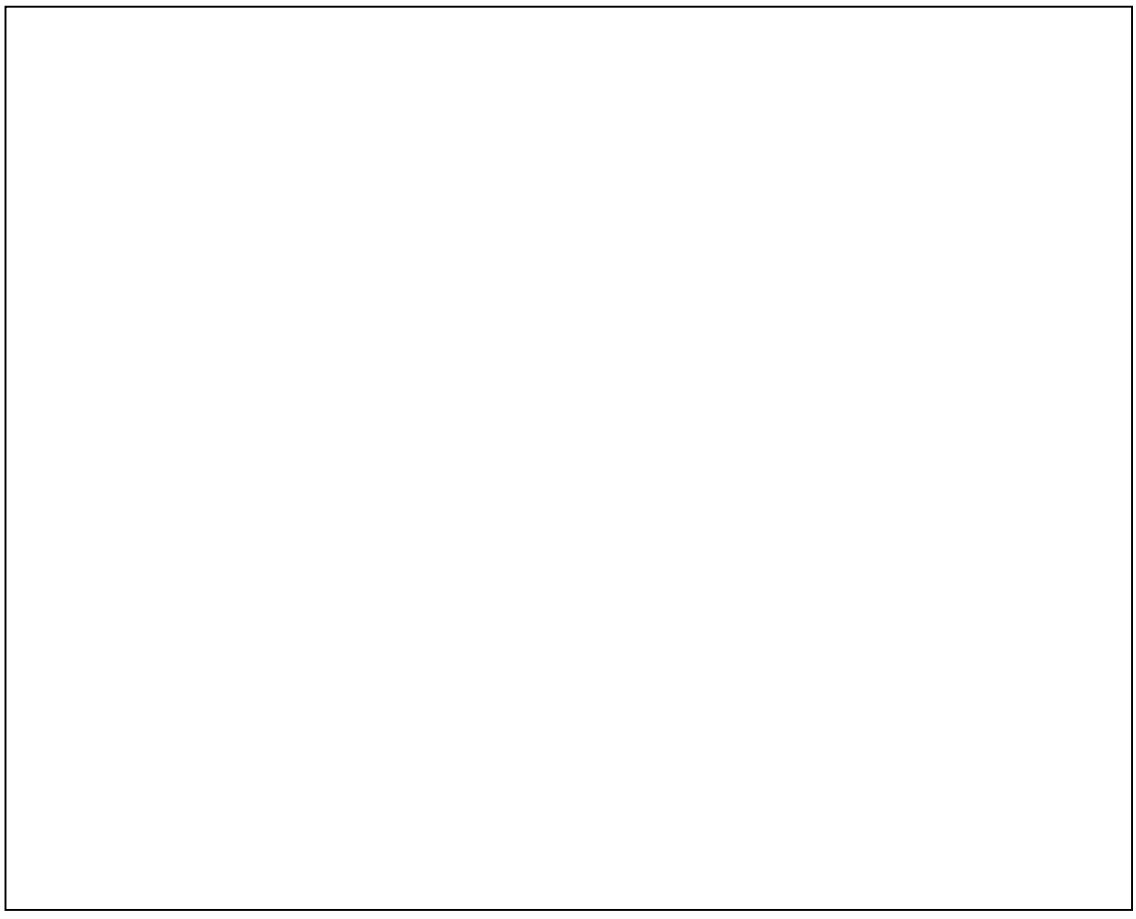


✉ sécurité : contrôle des actions exécutées par le code importé


Le Web et les objets distribués
URL
Page html
Applet
Station
browser
Interprète
Client
Http
requête http
pages html
Serveur
Serveur
http
servlet
Serveur
de
données
base de données
Corba
Serveur Corba

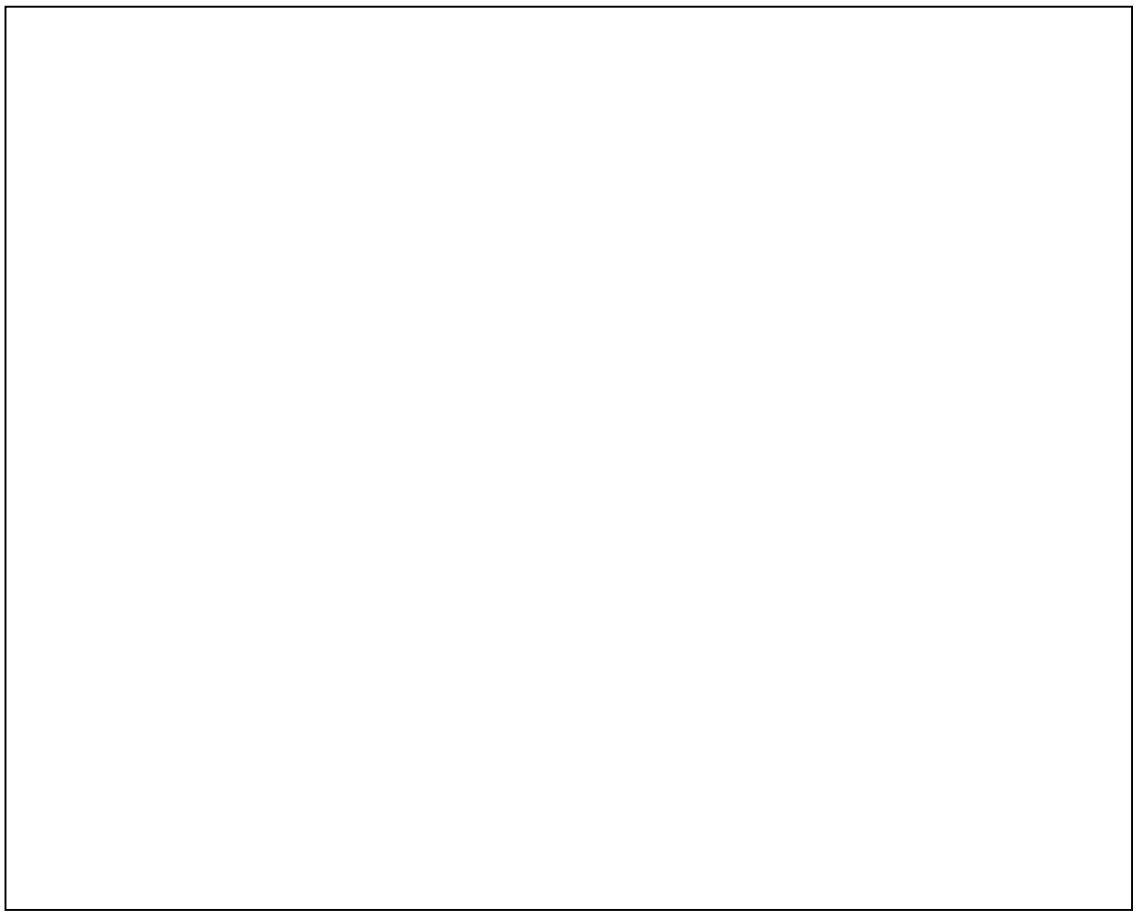
Client-serveur à "composants"

- **Motivations**
  - à l'origine, environnements pour la gestion de documents composites
    - ex. document composé de texte (word), tableaux (excel), données extraites d'une base de données, données trouvées sur le Web
- **Approche**
  - modèle et mécanismes pour réaliser l'interopérabilité entre "composants" logiciels
    - réutilisation de composants binaires

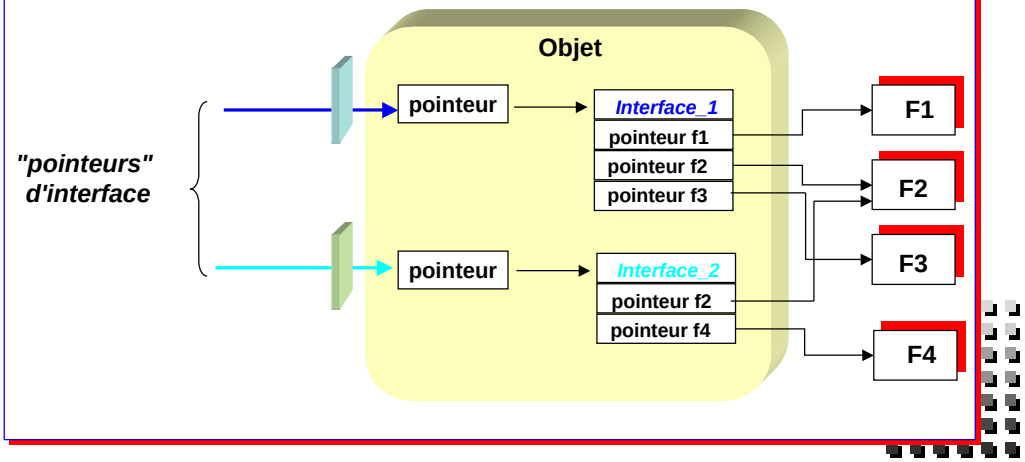


COM

- **Concepts**
  - objet/composant : encapsulation de données, manipulées par des fonctions
  - interface(s) d'un objet/composant : ensemble de fonctions d'accès à un objet/composant
    - un composant peut avoir plusieurs interfaces
    - les interfaces sont non modifiables ("immutable")
    - les interfaces sont désignées par un identificateur global GUI : Global Unique Identifier (128 bits)
  - classe : ensemble de services réalisés par un serveur (EXE) ou une bibliothèque (DLL)
    - un objet/composant peut faire référence à plusieurs classes

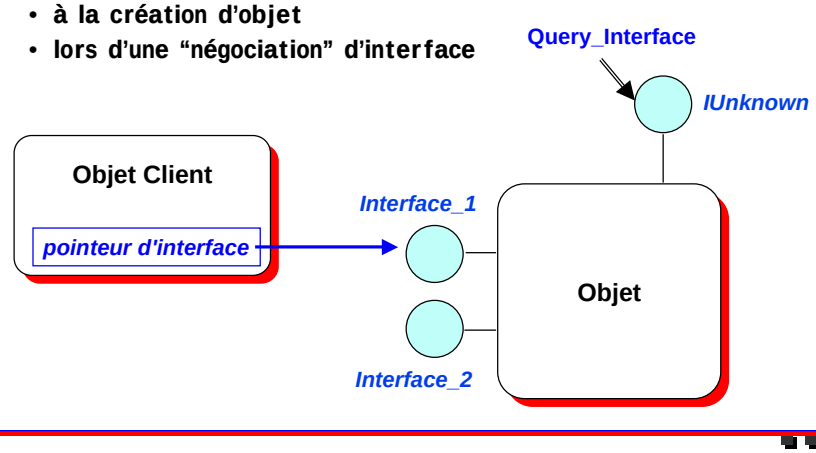


COM
Représentation d'un objet composant



COM
Relation entre objets

- un pointeur d'interface peut être obtenu ...
  - à la création d'objet
  - lors d'une "négociation" d'interface

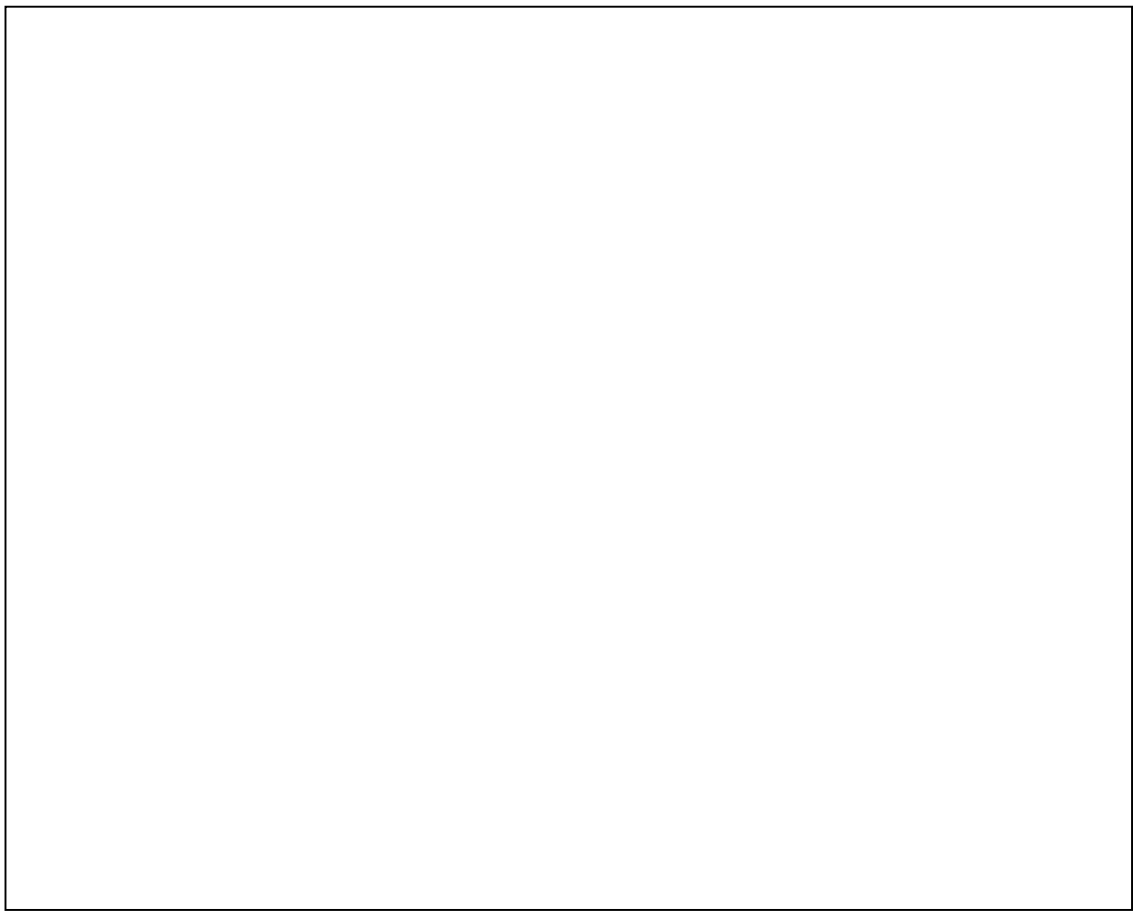


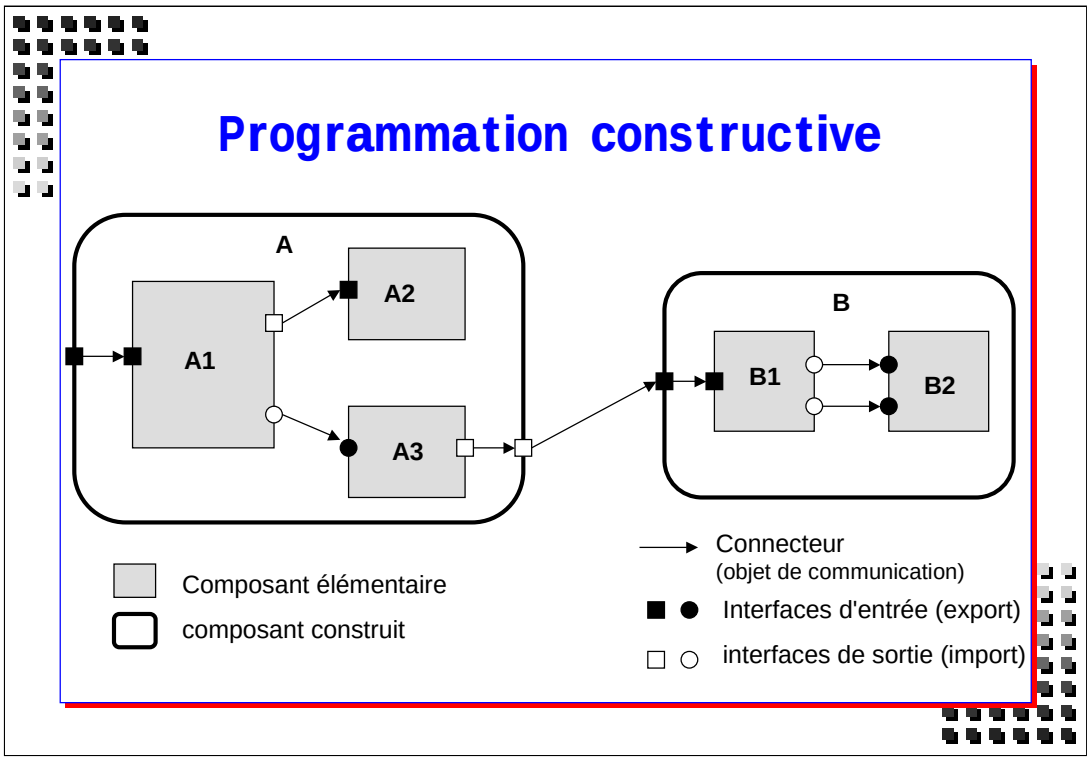
Programmation par composants JavaBeans

- **Bean : objet Java présentant certaines caractéristiques syntaxiques**
  - **attributs publics pouvant être positionnés de l'extérieur :**
    - **par d'autres Beans**
    - **par des outils**
  - **modèle de communication de type événement/réaction**

Programmation constructive

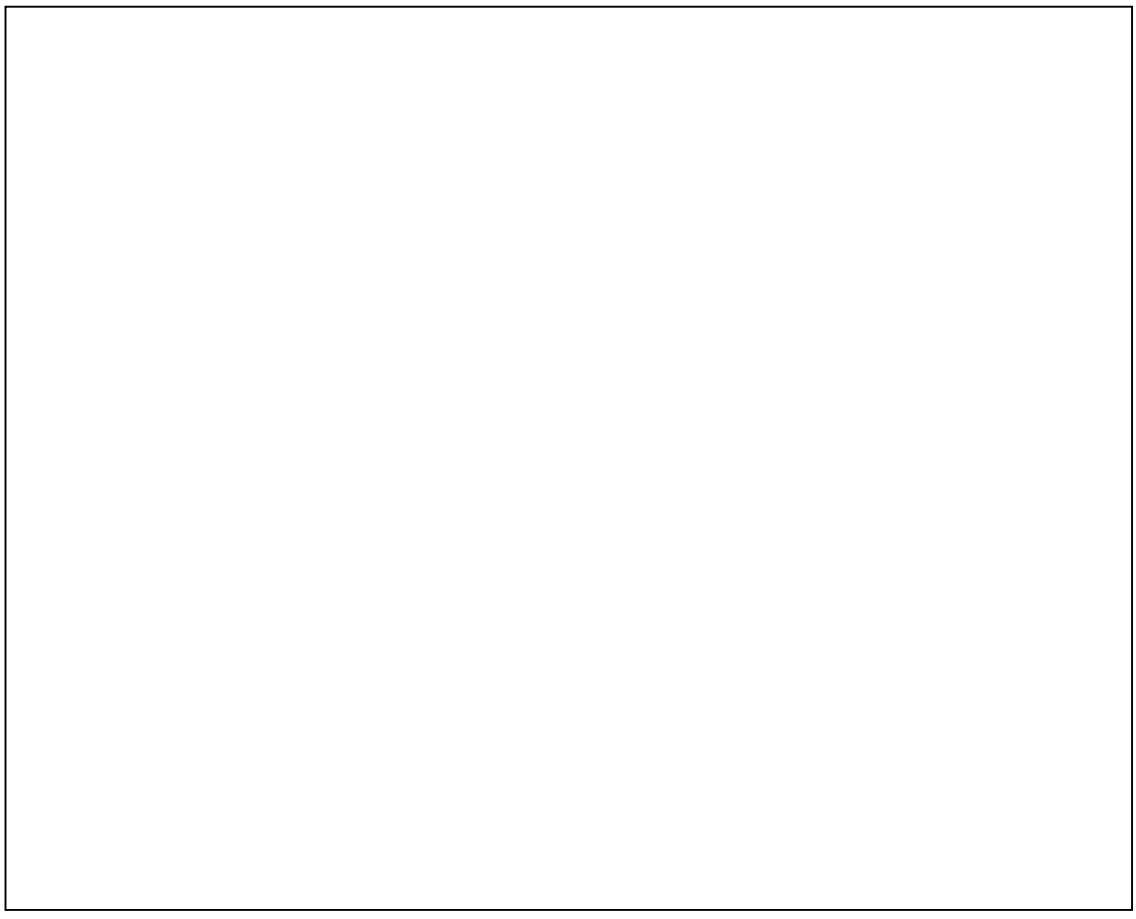
- **Motivation : réutilisation de logiciel**
  - intégration de modules logiciels existants
  - construction d'applications réparties par assemblage de modules logiciels existants
  - programmation à gros grain ("programming in the large")
- **Approche:**
  - description de l'architecture de l'application à l'aide d'un langage déclaratif
    - composant : interfaces, attributs, implémentation
    - description des interactions entre composants (connecteurs)
    - description de variables d'environnement (placement, regroupement, sécurité, etc.)




Programmation constructive
A
A1
A2
A3
B
B1
B2
Composant élémentaire
composant construit
Connecteur
(objet de communication)
Interfaces d'entrée (export)
interfaces de sortie (import)

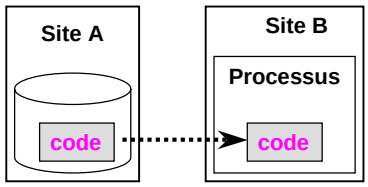
Code mobile

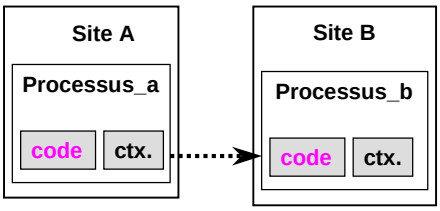
- **Définition**
  - programmes pouvant se déplacer d'un site à un autre
  - exemples : requête SQL, "applet" Java
  - Caractéristiques : code interprétable
- **Motivations**
  - rapprocher le traitement des données
    - réduire le volume de données échangées sur le réseau
    - partage de charge
  - function shipping versus data shipping
- **Problèmes**
  - Sécurité
  - Gestion de l'état, addressage des agents (communication)



Modèles d'exécution pour la mobilité

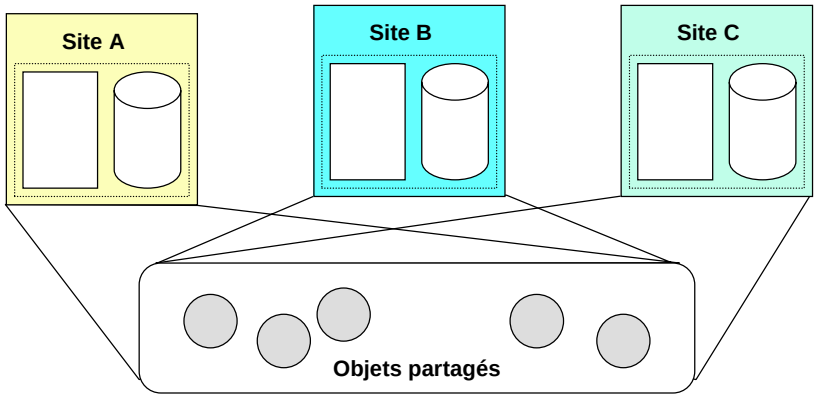
- **code "à la demande"**
  - exemple : "applet" Java
- **agents mobiles**
  - mobilité "faible" (Aglets)
    - code exécutable
    - données modifiées
  - mobilité "forte" (AgentTcl)
    - code exécutable
    - données
    - contexte d'exécution
  - Problème :
    - gestion des adhérences au système hôte (fichier, canaux ouverts, etc.)





Mémoire Virtuelle Distribuée

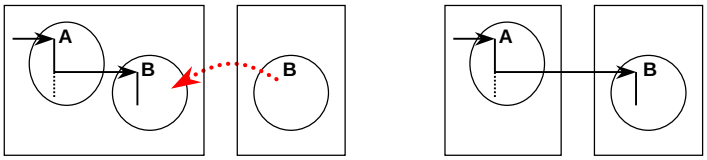
- **Principes de fonctionnement**
  - "simulation" d'une mémoire globale d'objets partagée



Mémoire Virtuelle Distribuée
Principes de mise en œuvre

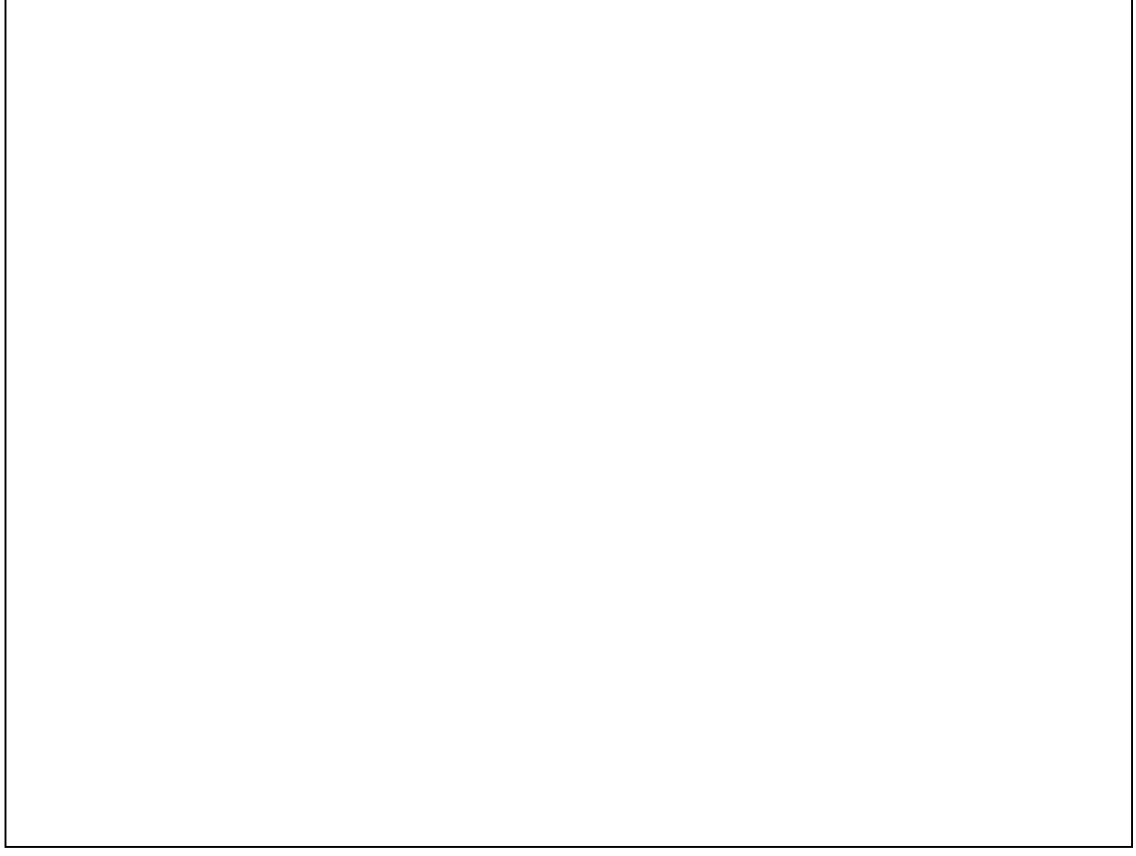
- **schéma d'exécution**
  - objets "actifs" versus objets "passifs"

  - migration des objets versus migration du contrôle

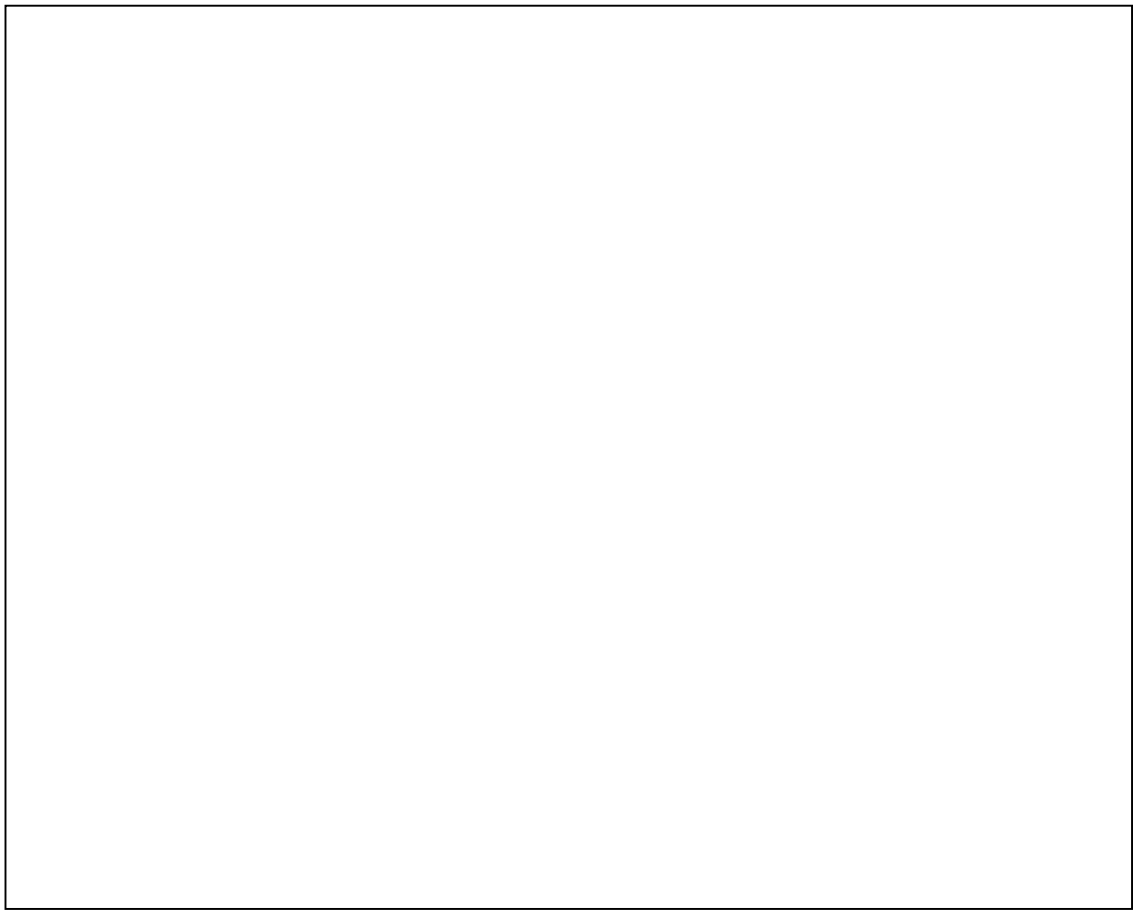


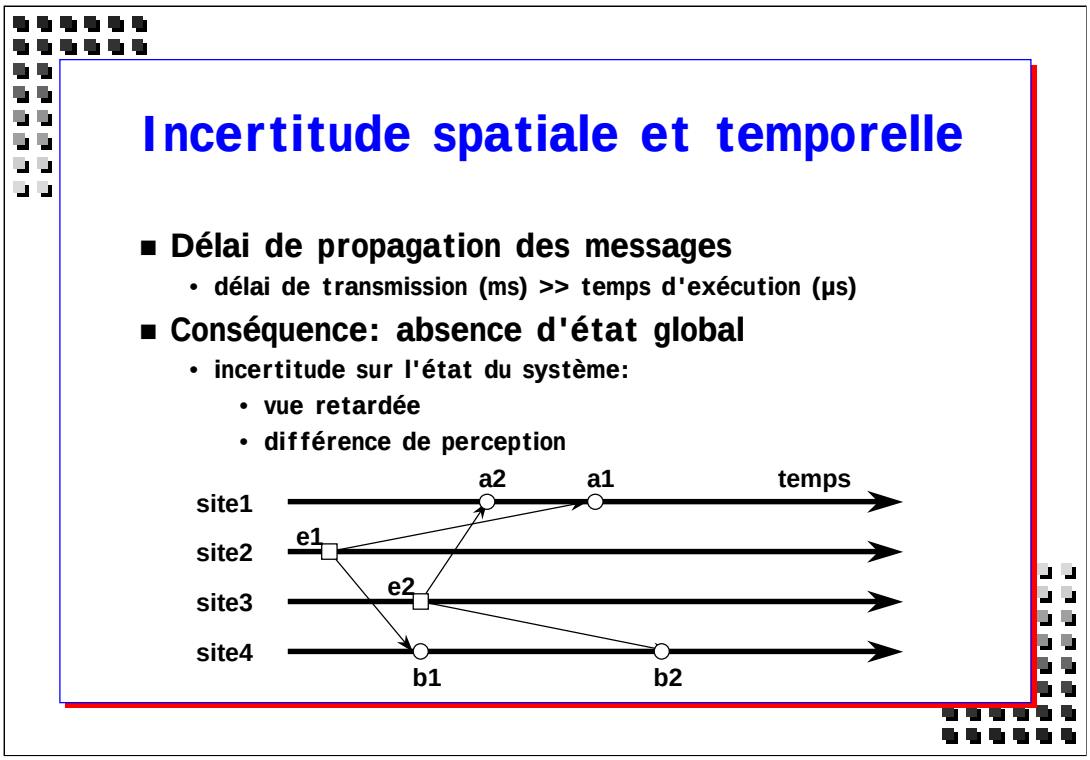
Mémoire Virtuelle Distribuée
Principes de mise en œuvre

- gestion du partage :
  - synchronisation
  - image unique d'un objet versus copies d'un objet
  - cohérence des images
- désignation
- gestion de la persistance



Synchronisation
Gestion de la Cohérence

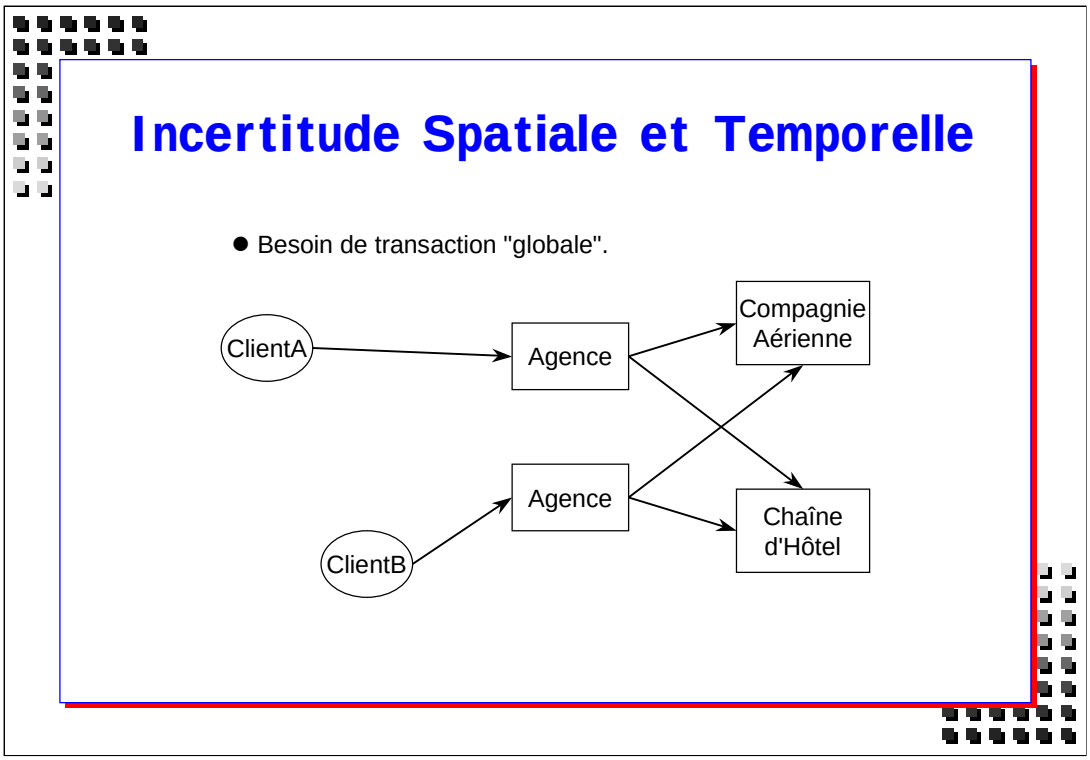




Pour appréhender le problème de l'incertitude spatiale et temporelle, il faut garder à l'esprit que tous les acteurs d'une application répartie perçoivent l'état des autres acteurs par le biais des messages qu'ils échangent :

• Le fait est que le temps de transmission d'un message est très supérieur au temps d'exécution d'une instruction ; le temps qu'un message arrive à son destinataire, son émetteur a donc considérablement progressé dans son exécution, le message ne délivre donc qu'une vue retardée de l'état de l'émetteur au destinataire.

• L'autre problème est que les réseaux actuels (au moins les WAN), ne garantissent pas l'ordonnancement des messages ; deux acteurs de l'application peuvent ainsi recevoir des messages dans des ordres différents, il en résulte des incohérences dans la perception qu'ils ont de l'état global de l'application.



Lors de la mise en place d'un déplacement, un client passe par une agence qui effectue diverses actions : en particulier, elle réserve d'une part le transport (avion par exemple) et d'autre part l'hôtel. Lorsque plusieurs clients effectuent des demandes similaires, leurs actions auprès des compagnies aériennes et des chaînes d'hôtel peuvent être perçues différemment. Ces différences peuvent affecter le bon fonctionnement du système.

Synchronisation de tâches réparties

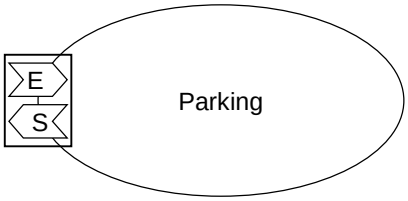
- Coordination d'un ensemble de tâches
- Exemple: le parking
  - N places de parking
  - Système formé d'automobiles (= processus)
    - entrer dans le parking → e
    - sortir du parking → s
  - Evénements e et s de durée nulle
- Synchronisation
  - Cadencement de l'évolution des processus en fonction des événements passés
  - Ordonnancement ⇗ séquence des événements observés

Synchronisation de tâches réparties

- **Séquence légale**
  - Composée de préfixe tels que le nombre de voitures entrées (e) ne dépasse pas le nombre de voitures sorties (s) de plus de N (nombre de places)
  - Exemple: N=3
    - eesees est légale
    - eeeesse n'est pas légale
- **Remarque:**
  - Problème d'allocation de ressources

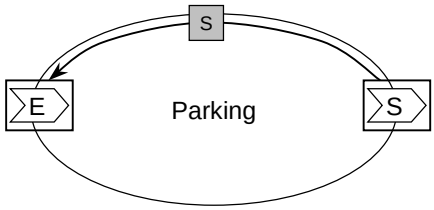
Synchronisation de tâches réparties

- **Solution: utilisation de compteurs**
  - **E = nombre de voitures entrées**
  - **S = nombre de voitures sorties**
  - **E - S < N (E - S > 0 par construction)**
- **Accès unique => pas de problèmes**
  - **perception globale, vue unique.**



Incertitudes spatiales et temporelles

- **le parking possède 2 accès:**
  - **1 entrée et 1 sortie**
  - **sortie -> S**
  - **entrée -> E, S' (image retardée de S)**
  - **condition d'entrée suffisante: E - S' < N**
    - **E - S < E - S'**

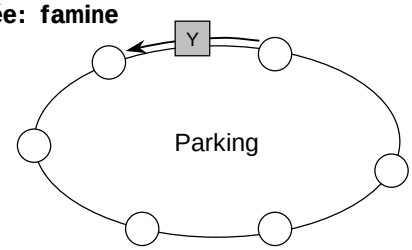


Incertitudes spatiales et temporelles

- **Le parking possède plusieurs accès:**
  - entrée et sortie.
  - Usage d'un compteur Y de places libres en accès exclusif
  - Circulation du compteur entre les accès selon une boucle virtuelle
  - Un accès permet l'entrée d'un véhicule lorsqu'il reçoit le compteur Y avec une valeur positive
  - Chaque accès remet à jour le compteur en fonction des sorties qu'il a constaté depuis son dernier passage

Incertitudes spatiales et temporelles

- **Restrictions**
  - Un seul accès en entrée à un instant donné
  - Perte du compteur
    - Détection
    - algorithme de régénération (1 et 1 seul)
  - Ordre d'entrée ≠ ordre d'arrivée: équité
  - Pas de garantie d'entrée: famine

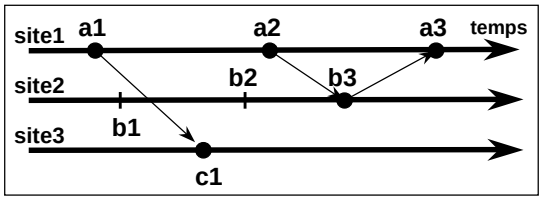


Ordonnancement

- **Définition d'un ordre**
  - ordonnancement de tous les événements d'un processus
  - précédence de l'événement "envoi d'un message par un processus" sur l'événement "réception de ce message par un autre processus"
  - chaque site peut alors situer tout événement dans une séquence ordonnée

Ordonnancement

- a1«a2«a3, b1«b2«b3
- a1«c1, a2«b3«a3
- a1«b1«a2«b2«b3«c1«a3 ≠ b1«b2«a1«a2«b3«c1«a3

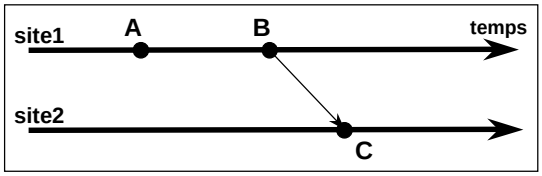


- horloge logique, estampille

Précédence causale

- A "*précède directement*" B ssi:
  - A et B arrivent sur le même site, et A est antérieur à B sur ce site

    ou
  - A est l'envoi d'un message sur un site, et B est la réception de ce message sur un autre site



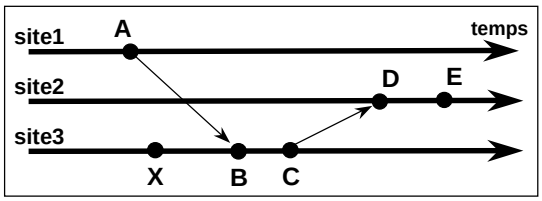
Cette définition définit le principe de causalité ; la seule interaction entre 2 sites étant l'échange de messages, les liens de causalité entre événements se produisant sur des sites différents utilisent forcément l'échange de messages entre ces sites.

C'est pourquoi la relation $\rightarrow$ est appelée "relation de précédence causale", il s'agit en fait plus exactement d'une dépendance potentielle: pour que A puisse être la cause de B, il faut que A $\rightarrow$ B.

Deux événements sont dits concurrents (ou causalement indépendants) si aucun des 2 ne peut influer sur l'autre.

Précédence causale

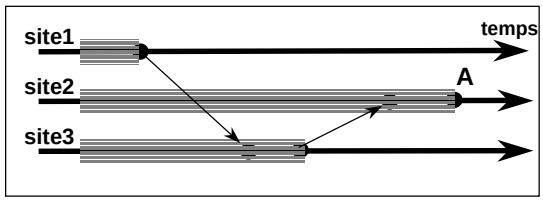
- La relation $\rightarrow$ (précède) est la fermeture transitive de la relation "*précède directement*"
  - Relation d'ordre partiel

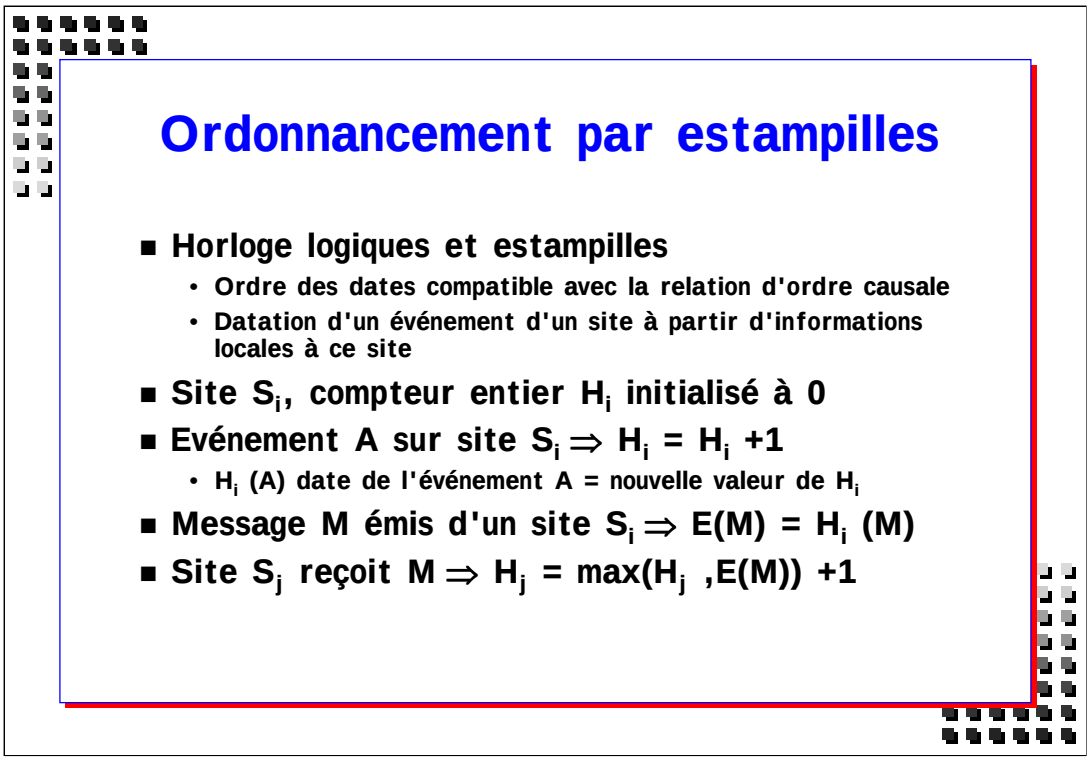


- Evénements concurrents
  - A || B $\Leftrightarrow \neg$ (A $\rightarrow$ B) et $\neg$ (B $\rightarrow$ A)

Précédence causale

- **Passé de l'événement A**
  - **ensemble des événements B tel que B → A**





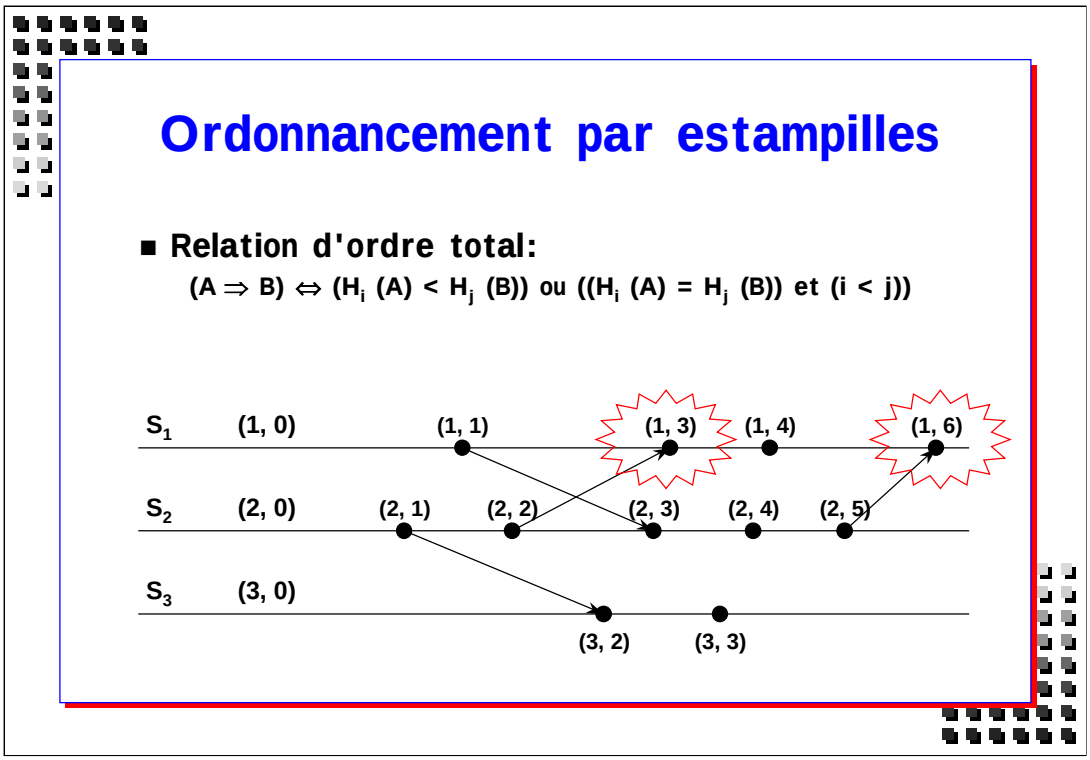
Une première méthode d'ordonnancement repose sur l'emploi d'horloge logiques et d'estampilles. Ce mécanismes réalise une datation avec les propriétés suivantes:

• l'ordre des dates est compatible avec la relation de précédence causale,

• la datation d'un événement sur un site ne met en œuvre que des informations locales à ce site.

On définit sur chaque site S_i un compteur logique H_i de valeur entière, initialisé à 0, qui sert à dater les événements du site. Chaque fois qu'un événement Ase produit sur ce site, H_i est incrémenté de 1. On note H_i (A) la date de l'événement A qui est égal par définition à la nouvelle valeur de H_i.

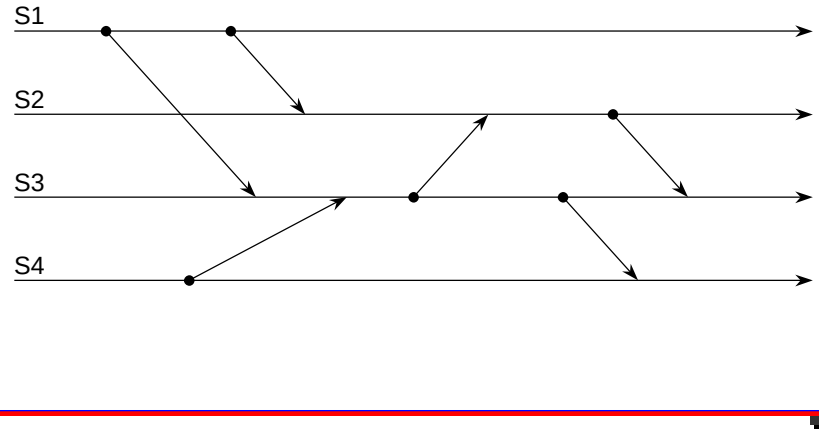
Pour garantir le respect de la précédence causale, tout message M émis par un site S_i porte une estampille E(M) qui est égale à sa date d'émission. Lorsqu'un site S_j reçoit le message M il met àjour son horloge logique.

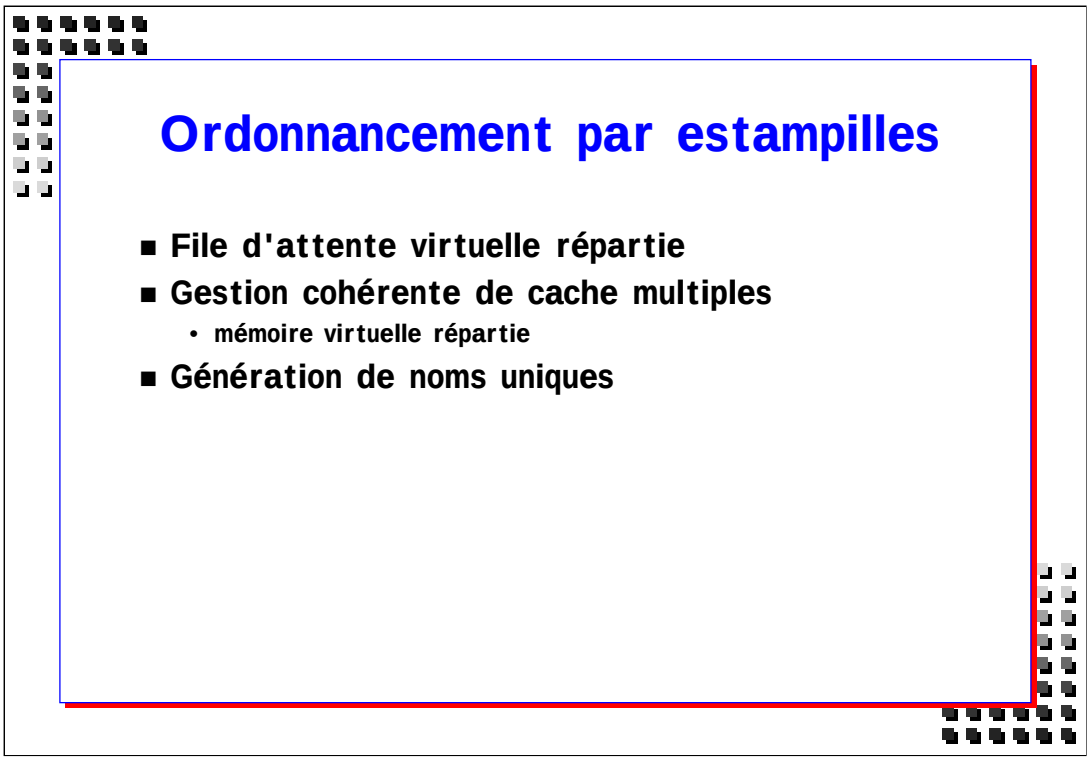


La relation ainsi définit n'est pas un ordre strict, pour obtenir un ordre strict il suffit de définir un ordre arbitraire entre les sites et on peut alors définir la relation d'ordre total strict ...

Chaque événement est maintenant daté par un couple (n° de site, estampille).

Exercice

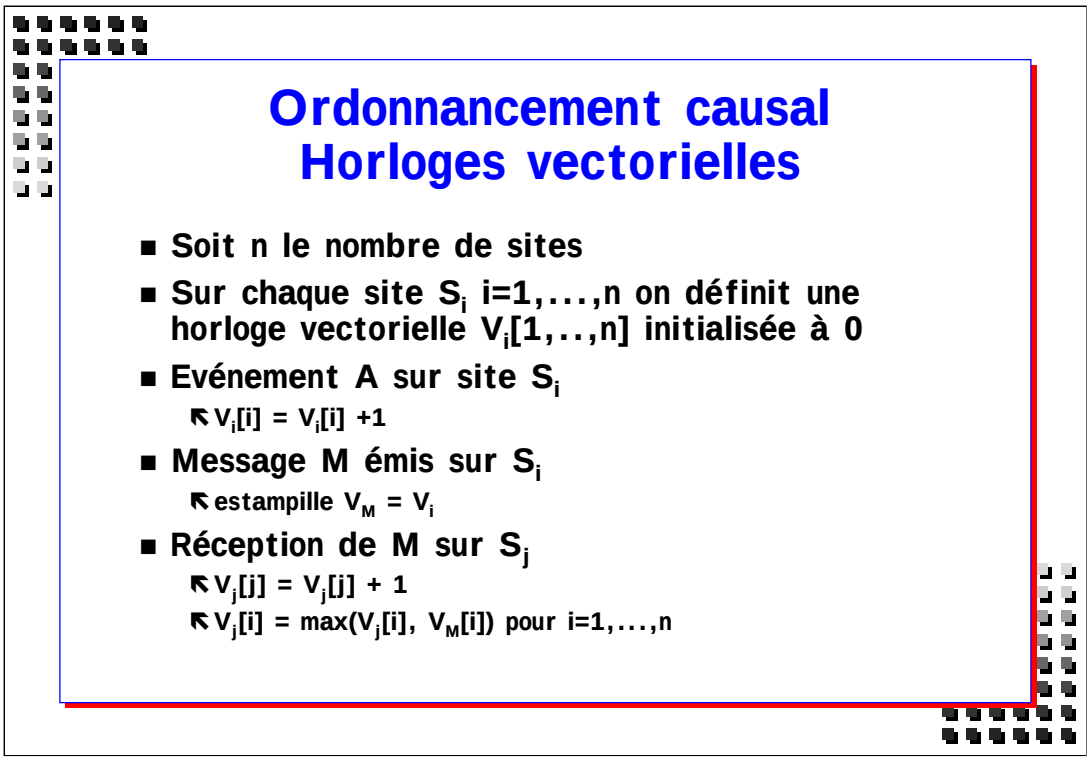




L'ordonnancement des événements par estampilles est une technique d'usage universel dans les systèmes répartis. Quelques usages courants sont:

• Les algorithmes qui mettent en œuvre une file d'attente répartie. Une telle file possède une représentation partielle ou totale sur chaque site concerné, et l'ordre global des éléments de la file est défini par cette relation.

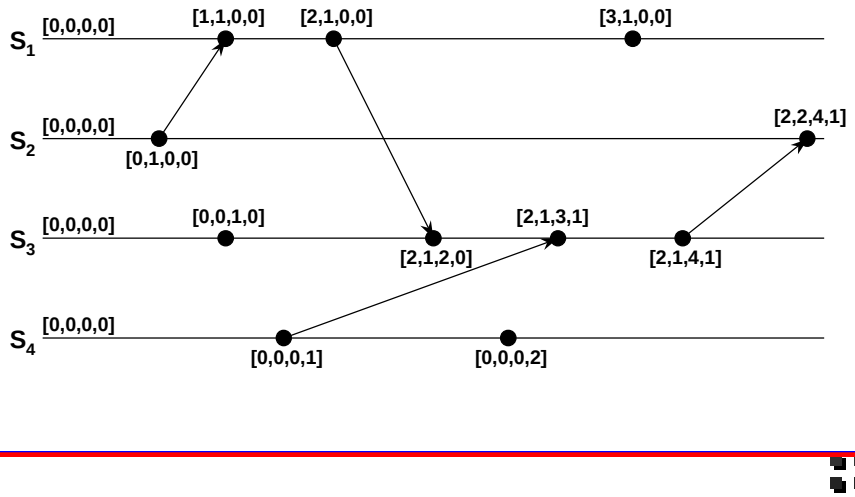
• etc.



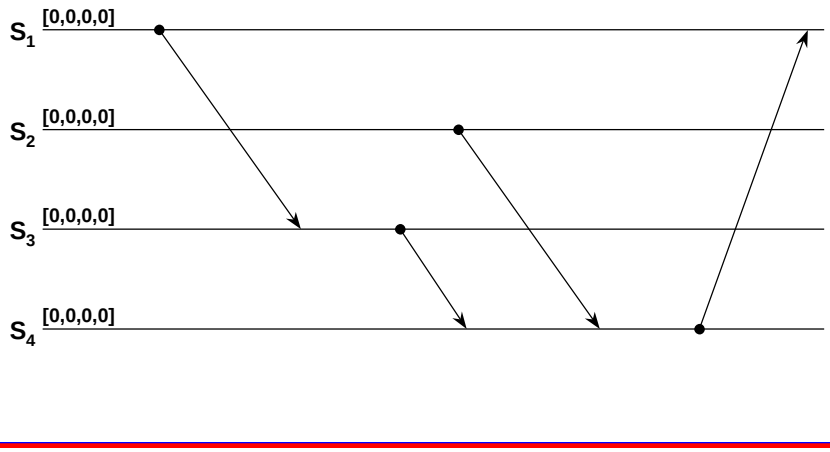
La relation d'ordre définie par les estampilles ne suffit pas pour établir une relation de causalité entre 2 événements. On peut simplement dire que cet ordre ($\Rightarrow$)est compatible avec l'ordre partiel de précédence causale ($\rightarrow$).

Il est néanmoins utile de pouvoir déterminer la dépendance (ou l'indépendance) causale entre 2 événements. Le mécanisme des horloges vectorielles a été introduit pour caractériser la dépendance causale.

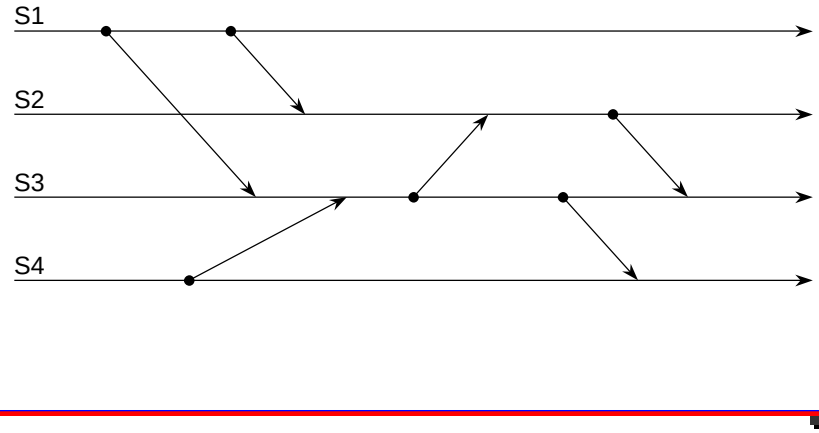
Ordonnancement causal
Horloges vectorielles



Exercice

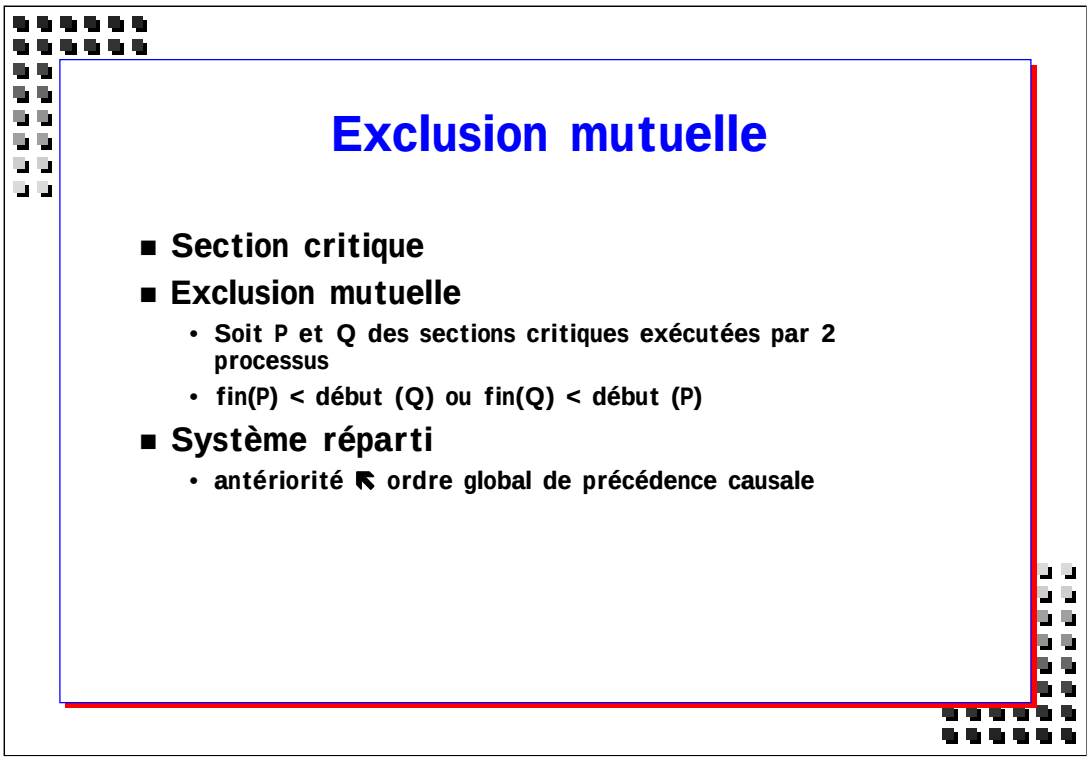


Exercice



Ordonnancement causal
Horloges vectorielles

- **Passé d'un événement A**
  - $V_A[j]$ = nombre d'événements du passé de A sur le site Sj
  - $\Sigma V_A[j]$ = nombre total d'événements du passé de A
- **Relation d'ordre partiel**
  - $V \leq W \Leftrightarrow \forall j : V[j] \leq W[j]$
  - $V \,||\, W \Leftrightarrow \neg(V \leq W)$ et $\neg(V \geq W)$
- **Propriété fondamentale**
  - $A \rightarrow B \quad \Leftrightarrow \quad V_A < V_B$
  - A et B causalement indépendant $\Leftrightarrow \quad V_A \,||\, V_B$

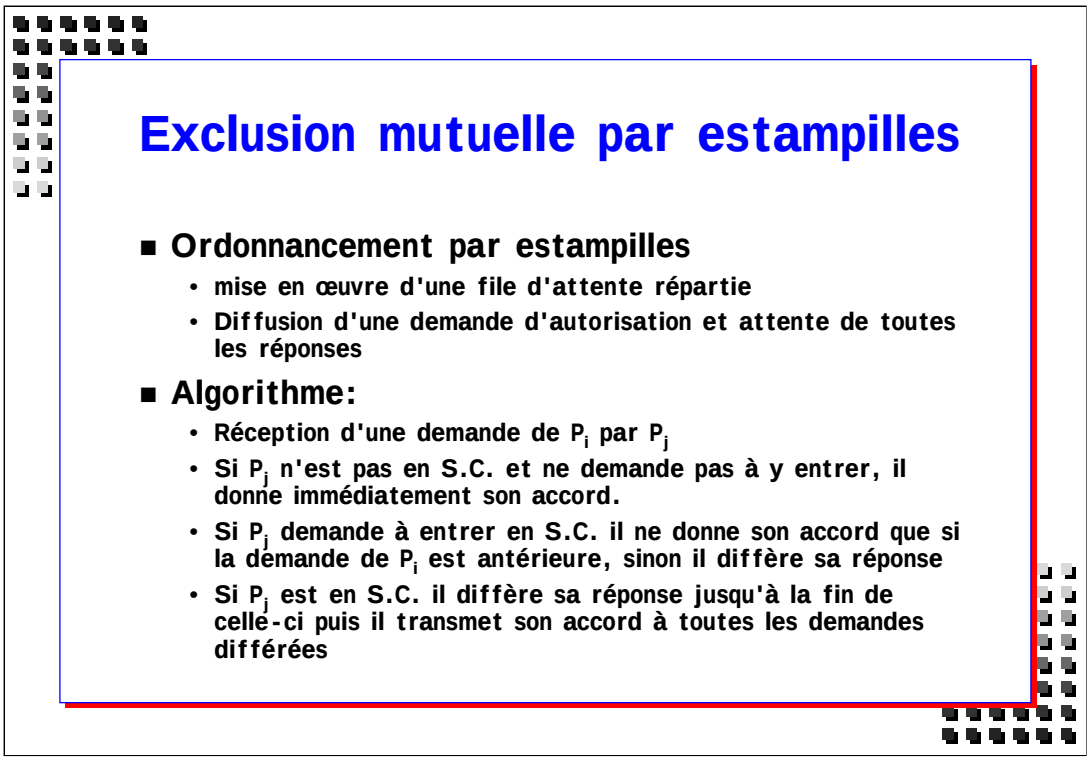


Rappelons la notion d'exclusion mutuelle dans un système centralisé: étant donné n processus, on définit dans le programme de chacun d'eux une séquence appelée *section critique* et on impose la contrainte qu'un processus au plus peut être dans sa section critique à un instant donné.

Dans un système centralisé, les outils de synchronisation permettent de contrôler l'accès à la S.C. : instruction T.A.S. (Test And Set), mutex, sémaphores, etc.

Une autre manière de définir l'exclusion mutuelle consiste à dire que l'exécution des sections critiques par les processus est sérialisée. Dans un système réparti, c'est cette définition qui permet de spécifier l'exclusion mutuelle ; il faut pouvoir alors dater les événements de début et de fin de S.C. afin de vérifier la condition :

• fin(P) < début (Q) ou fin(Q) < début (P)



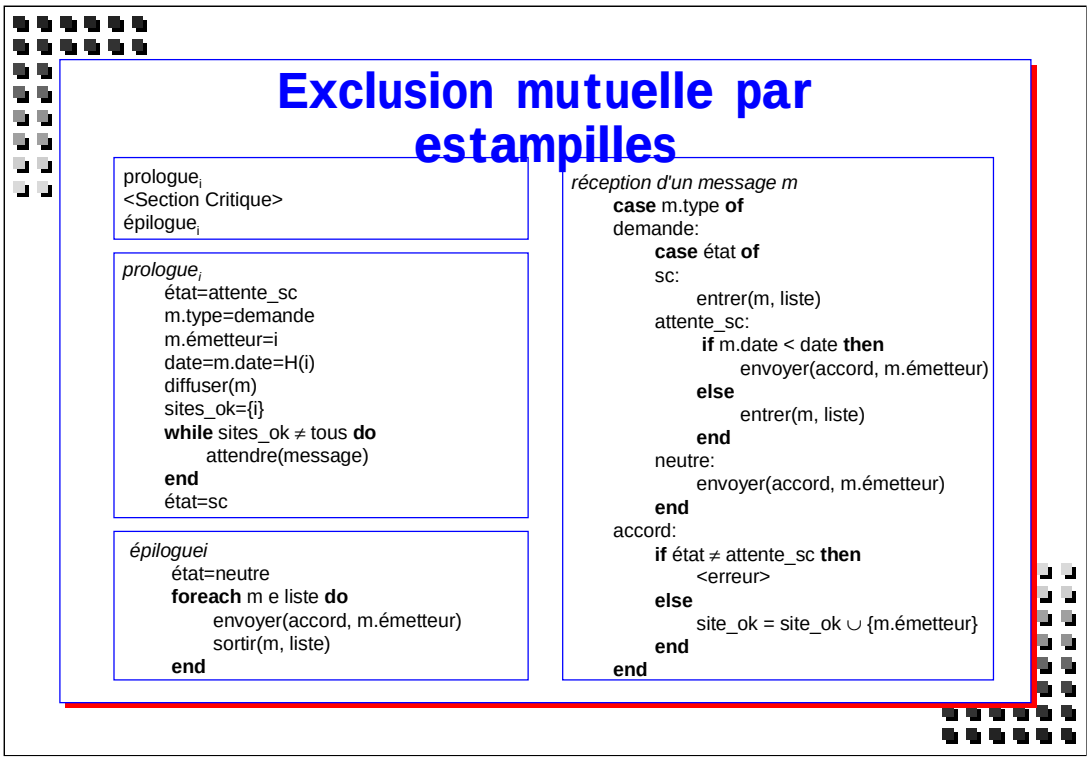
Cet algorithme est une application directe de l'ordonnancement des événements par estampilles, il est caractéristique de la gestion d'une file d'attente répartie dont il existe une représentation partielle sur chaque site. Cette file est ordonnée globalement par l'heure logique des demandes d'entrée.

Principe: Un processus qui désire entrer en S.C. diffuse une demande d'autorisation à tous les autres, et attend leur accord. Il ne peut entrer dans la S.C. que lorsqu'il a reçu toutes les réponses. Un processus Pj qui reçoit une demande d'autorisation de Pi réagit comme suit :

• si il n'est pas en concurrence pour entrer un S.C. il donne son accord,

• si il es en S.C. il mémorise les demandes, puis lorsqu'il sort de la S.C. il donne son accord à tous les prétendants,

• si Pi et Pj sont en concurrence pour entrer en S.C. alors chacun va examiner la demande de l'autre et évaluer laquelle est antérieure. Si on veut que le conflit se règle il est impératif que tous les processus en cause prennent une décision identique! Cela justifie l'utilisation d'horloges logiques telles que nous les avons présentées auparavant pour assurer l'absence d'interblocage et de privation.



L'entrée d'un processus en section critique nécessite 2(n-1) messages. En l'absence de pannes, cet algorithme réalise l'exclusion mutuelle sans interblocage ni privation.

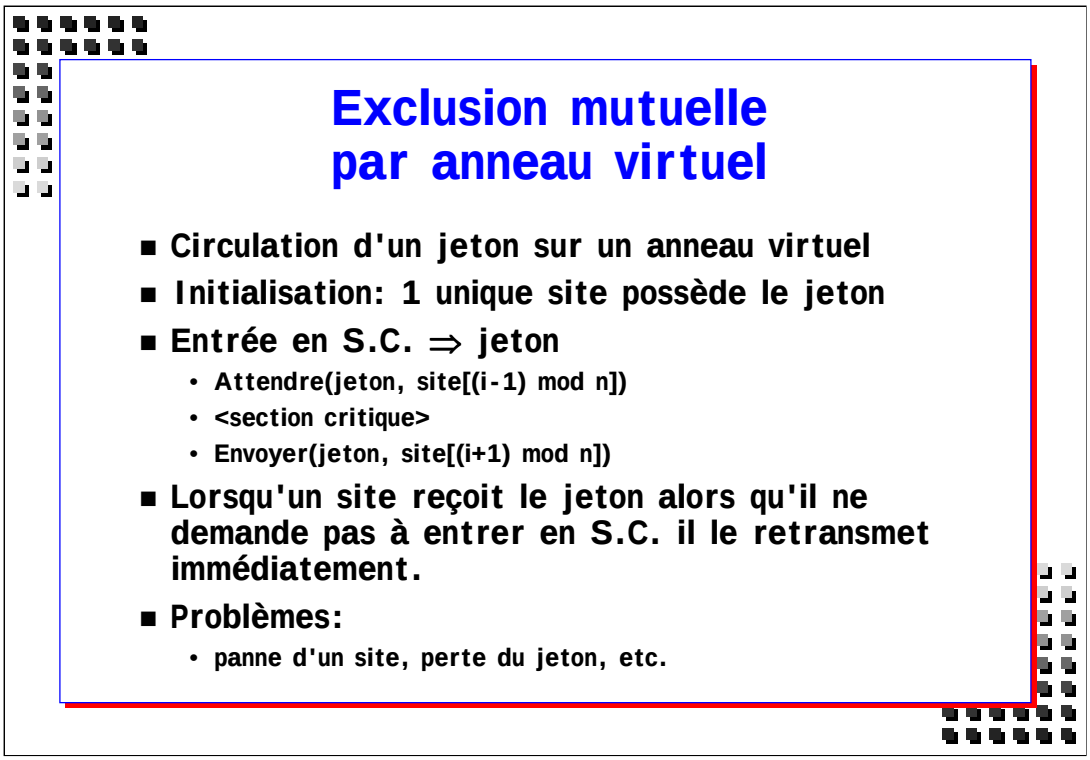
Exclusion mutuelle: lorsqu'un site Si qui a demandé à entrer en S.C. a reçu une réponse de tous les autres, alors:

• sa demande est la plus ancienne de toutes celles en attente.

• toutes les S.C. demandées antérieurement sont terminées.

Absence d'interblocage: garanti par l'ordonnancement global des demandes.

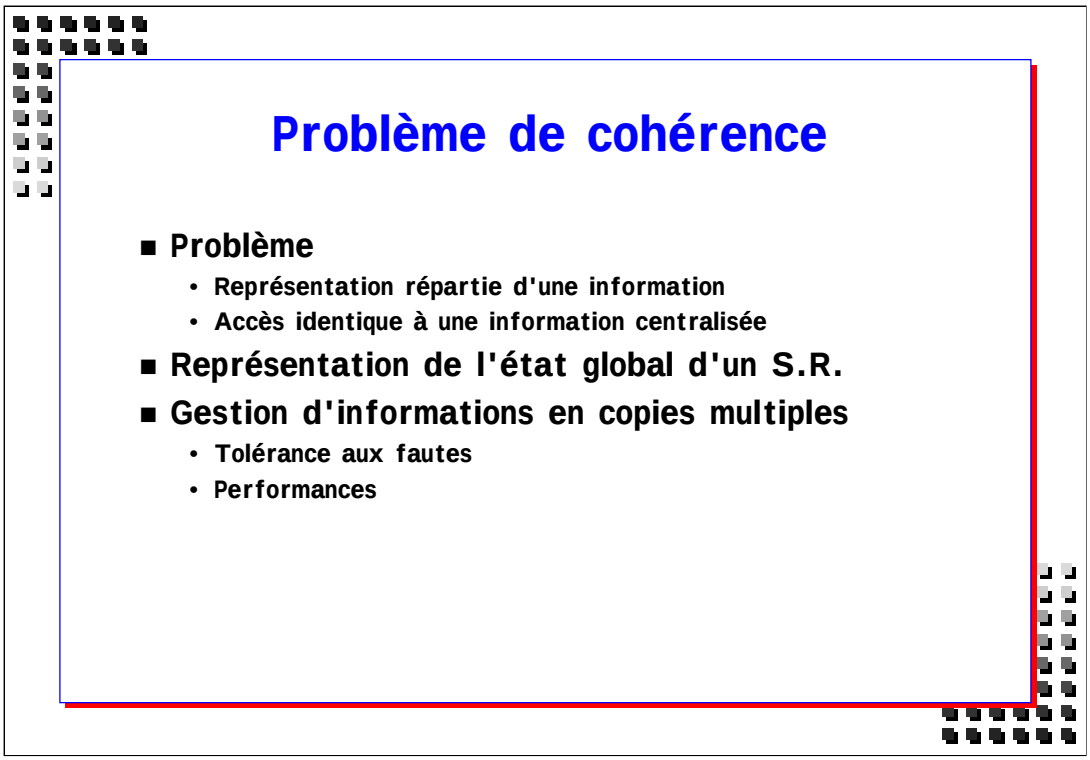
Absence de privation: tout site ayant fait une demande obtient un accord au bout d'un temps fini (si toutes les S.C. durent un temps fini) puisque les demandes sont traitées dans l'ordre (l'algorithme peut même être déclaré **équitable** si l'ordre des messages est conservé).



Une autre méthode repose sur la circulation d'un jeton entre les sites organisés en anneau virtuel.

Cet algorithme est identique à l'allocation de la voie sur un anneau virtuel, sa validité repose sur l'existence d'un jeton unique qui peut être mis en cause par la panne d'un site ou du système de communication.

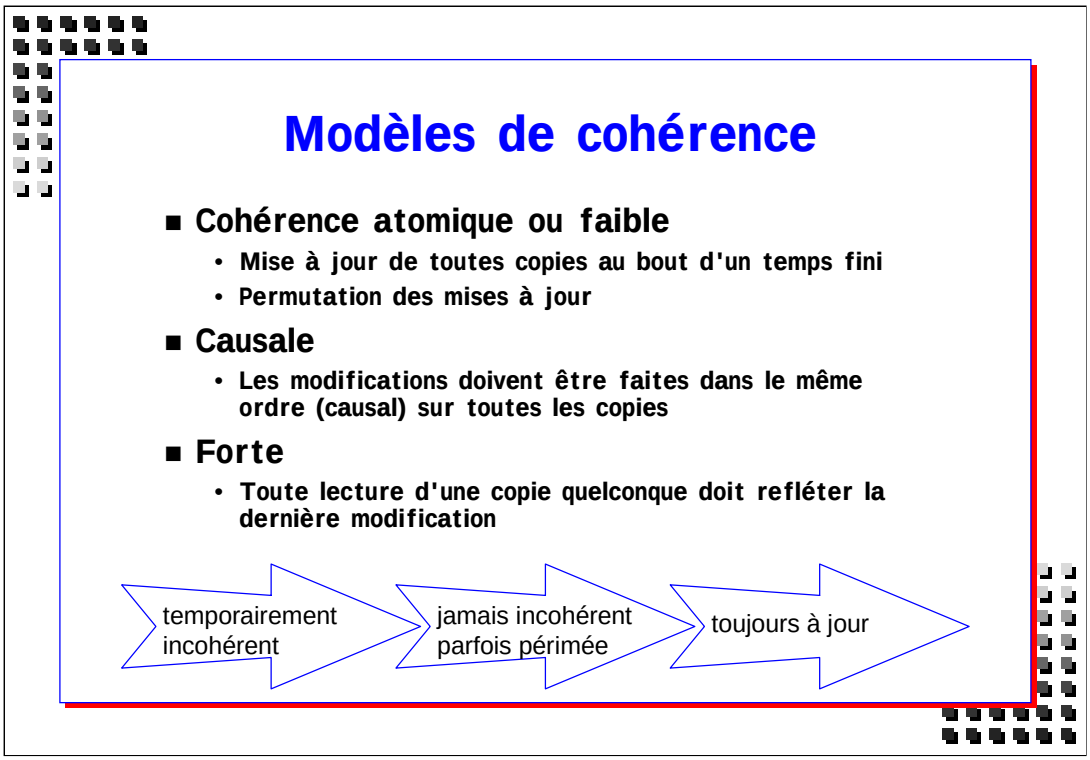
Nous ne traitons pas de la reconfiguration de l'anneau en cas de panne, de l'insertion ou du retrait d'un site. Le principe de la détection de la perte du jeton, et de sa régénération est examiné ci-après.



Les problèmes de la cohérence d'informations dans un système réparti peuvent se résumer à permettre l'accès à une information répartie comme si elle était centralisée, c'est à dire représentée sur un site unique et manipulée par des processus de ce site.

Ce problème recouvre plusieurs cas, dont la représentation de l'état global d'un système réparti ou la gestion de copies multiples.

représentation de l'état global d'un S.R.: Il est impossible de connaître l'état global **instantané** d'un S.R., car les informations reçues sur l'état de sites distants sont périmées au moment de leur réception, et que leur ordre d'arrivée n'est pas forcément compatible avec l'ordre causal. On peut néanmoins chercher à accéder



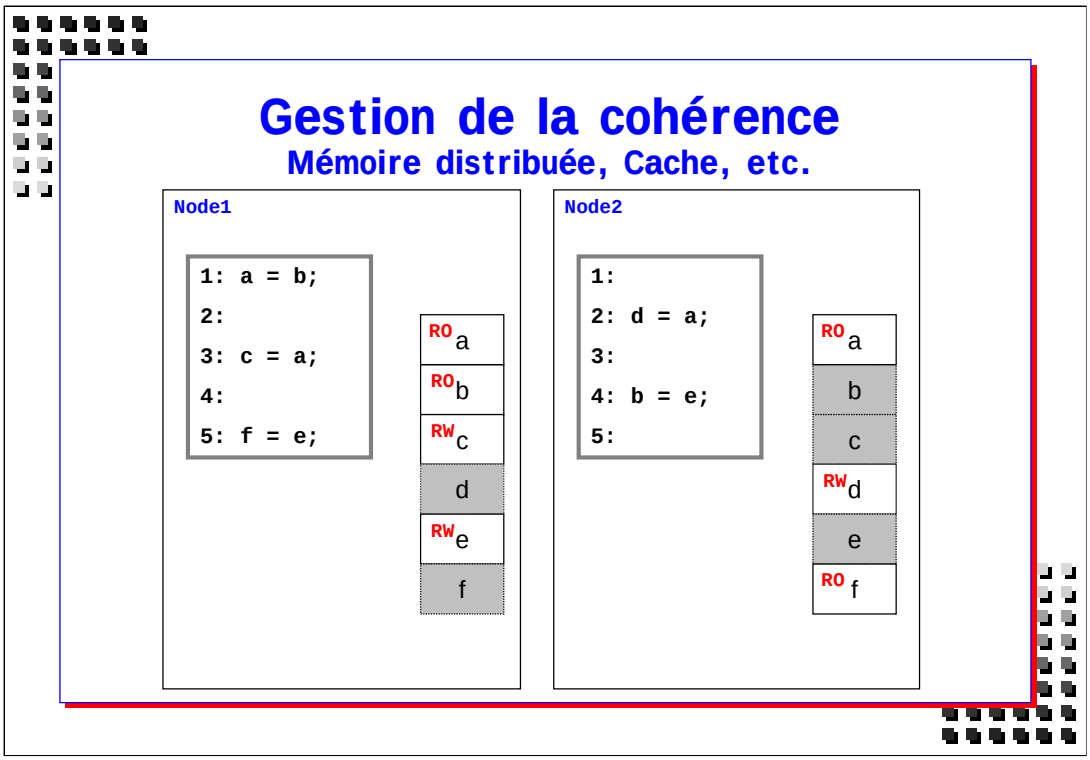
Cohérence faible: la mise à jour de toute copie doit avoir lieu au bout d'un temps fini ; néanmoins, il n'y a pas de contrainte sur l'ordre des mises à jour. Ce type de cohérence n'est acceptable que si les mises à jour peuvent être permutées sans conséquences (par ex. l'insertion d'éléments dans une table).

Cohérence causale: Pour la cohérence causale, on ajoute une contrainte sur les mises à jour: les modifications de toutes les copies doivent être faites dans le même ordre, défini par l'ordre causal.

Cohérence forte: Toute consultation d'une copie doit refléter le résultat de toutes les modifications antérieures (au sens causal).

Avec la cohérence faible, une copie peut être temporairement incohérente ; avec la cohérence causale, une copie est toujours cohérente mais elle peut être périmée ; avec la cohérence forte, les copies sont toujours à jour.

L'expérience montre que la cohérence causale, relativement peu couteuse à mettre en œuvre, est en fait suffisante pour la plupart des applications.



Afin de permettre un accès "distribué" à l'information, chaque site gère des copies de celle-ci. Lors d'une tentative d'accès (défaut de page dans une mémoire distribuée, opération de lecture/écriture dans un système de fichiers, etc.), le système permet l'accès direct à l'information si celle-ci est présente (avec les bons droits), sinon il en demande le transfert au site détenteur. Les conditions de transfert et de partage d'un objet déterminent le degré de cohérence mis en œuvre.

Cohérence Forte

- **Identique à un partage local**
- **1 rédacteur / n lecteurs**
- **Toujours à jour**

1: **S1** Prise du droit d'écriture sur l'objet A
   S2 Invalidation de la copie de l'objet A
2: **S1** Invalidation du droit d'écriture sur l'objet A
   S2 Récupération d'une copie de l'objet A (RO)
3: R.A.S.
4: **S1** Invalidation de la copie de l'objet B
   Invalidation du droit d'écriture sur E
   S2 Récupération d'une copie des l'objets B (RW) et E (RO)
5: **S1** Récupération d'une copie de l'objet F (RW)
   S2 Invalidation de la copie de l'objet F

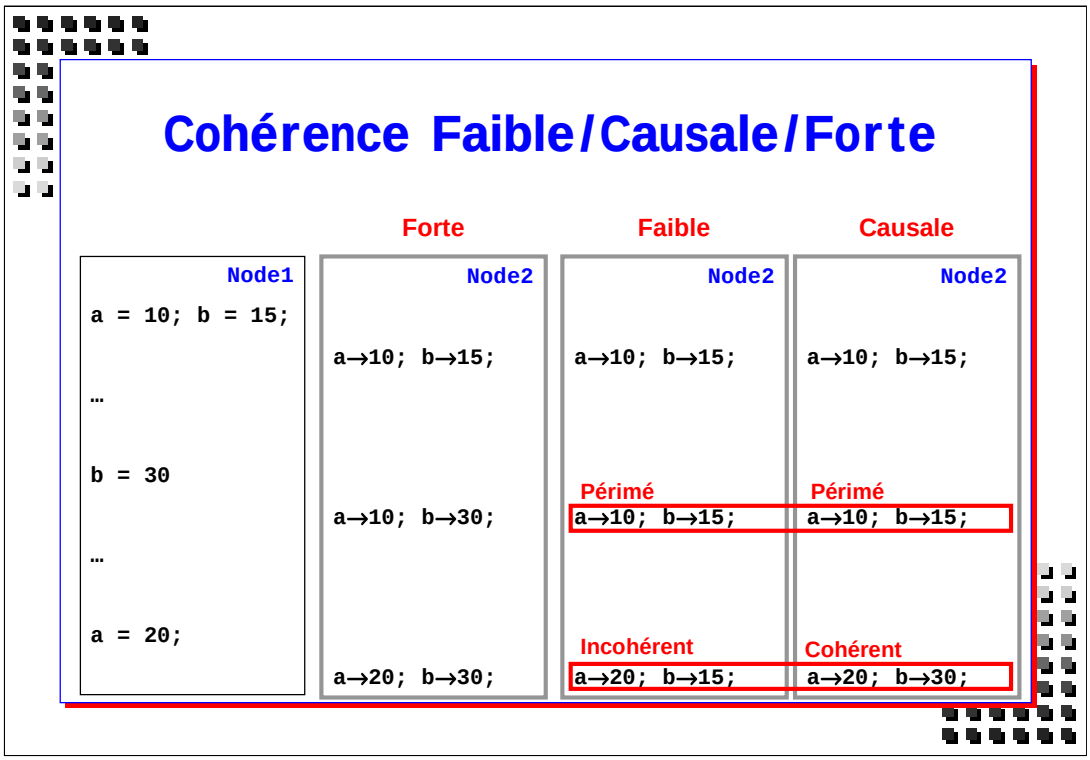
Node1	Node2
1: a = b;	1:
2:	2: d = a;
3: c = a;	3:
4:	4: b = e;
5: f = e;	5:

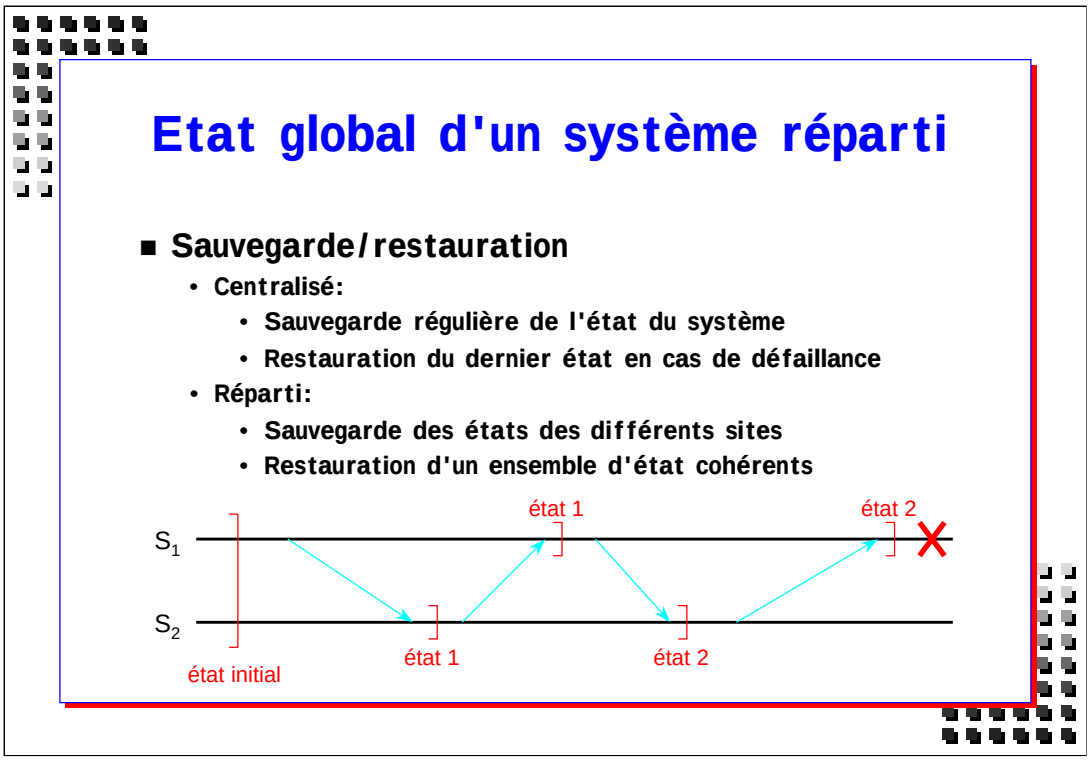
Cohérence Faible

1: R.A.S.
2: R.A.S.
3: R.A.S.
4: **S2** Récupération d'une copie des l'objets B et E
5: **S1** Récupération d'une copie de l'objet F (RW)

Node1	Node2
1: a = b;	1:
2:	2: d = a;
3: c = a;	3:
4:	4: b = e;
5: f = e;	5:

Cohérence Faible/Causale/Forte





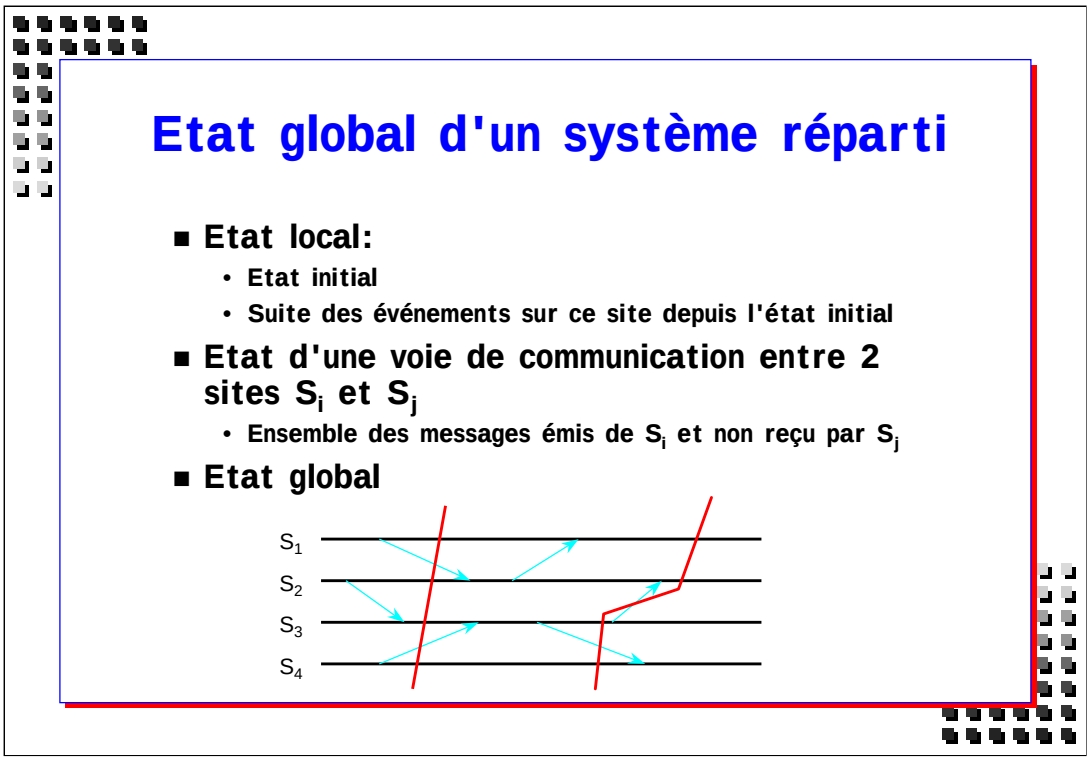
Nous avons déjà noté la difficulté de définition de l'état global d'un S.R., en raison des incertitudes sur les transmissions d'information entre sites. Nous allons maintenant définir de manière rigoureuse la cohérence d'un état global. Cette définition a de nombreuses applications, comme par exemple la sauvegarde/restauration de l'état d'un système afin de permettre une reprise du fonctionnement après une défaillance.

Pour ce faire, on sauvegarde régulièrement l'état du système ; lors de la reprise, on restaure le dernier état sauvegardé avant la défaillance.

Dans un S.R., il faut que l'ensemble des états partiels qui servent de base à une reprise constitue un état global cohérent.

Exemple: Dans cet exemple, une défaillance du processus 1 après le point de reprise 2 oblige la restauration du système dans l'état initial. En effet, dans chacun des états sauvegardés l'un des processus a reçu un message dont l'autre n'a pas enregistré l'envoi (une restauration dans cet état ferait que le processus reémettrai le message lors de la reéxécution, alors que l'autre l'a déjà traité):

• La restauration de S1 dans l'état 2, entraîne celle de S2 dans l'état 2, qui entraîne celle de S1 dans l'état 1, etc.



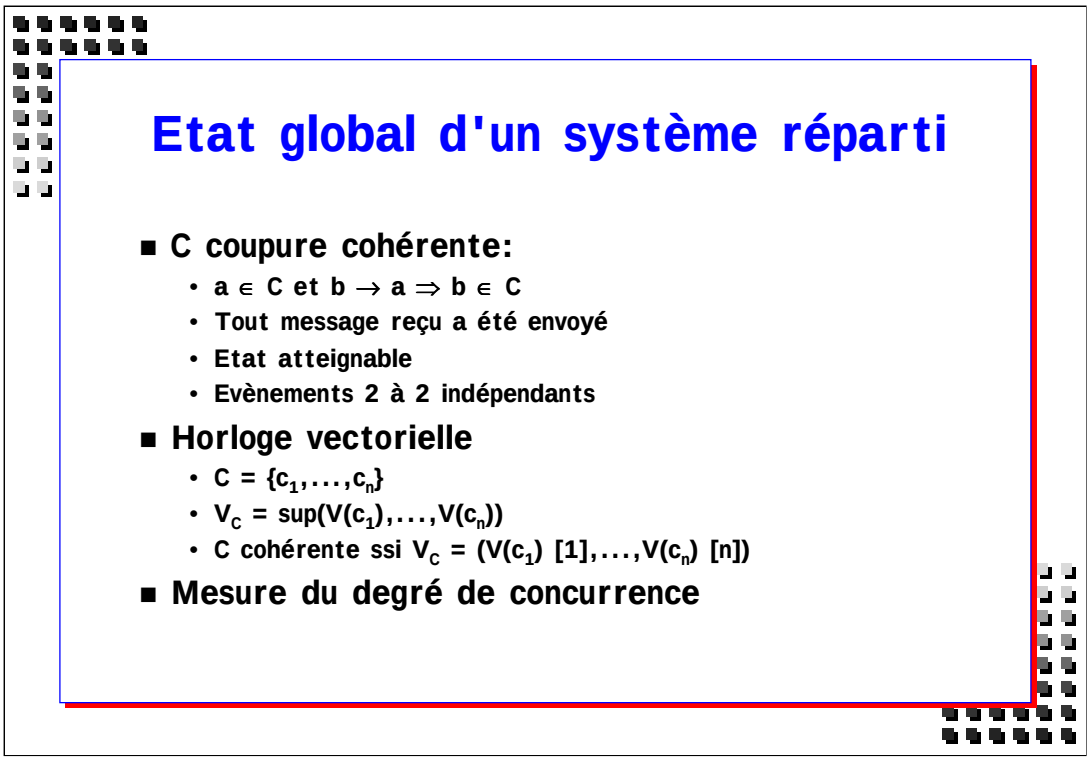
La relation de précédence causale permet de définir la cohérence de l'état global d'un S.R.:

• Sur chaque site on définit l'état local par l'état initial et la suite des événements sur ce site.

•On définit en outre l'état d'une voie de communication entre 2 sites Si et Sj par l'ensemble des messages émis par Si en non reçu par Sj.

• L'état global est alors défini par l'ensemble des états locaux des sites et des voies de communication qui les relient.

Si l'on représente l'évolution de l'état des sites par des axes horizontaux, et les messages par des flèches joignant des points sur ces axes, on peut représenter un état global par une coupure (ligne joignant les points de sauvegarde des différents états).



On peut définir une coupure comme l'ensemble des événements qui constituent son passé. La cohérence s'exprime alors comme une propriété de fermeture: une coupure est cohérente ssi elle est fermée par la relation de précédence causale, c-à-d:

- $a \in C$ et $b \rightarrow a \Rightarrow b \in C$

On peut alors montrer que tout message reçu a été envoyé, que la coupure constitue un état atteignable : un état par lequel l'application a pu passer, et donc un état possible de reprise.

Les événements définissant la coupure sont 2 à 2 indépendants.

On peut dater les coupures au moyen d'horloges vectorielles. Soit une coupure définie par les événements c1,...,cn ; on définit la date Vc de C par:

- $V_C = \sup(V(c_1),...,V(c_n))$
- C cohérente ssi $V_C = (V(c_1)\ [1],...,V(c_n)\ [n])$.

Outre son intérêt conceptuel, et son application pratique à de nombreux algorithmes répartis, la notion de coupure fournit une mesure du degré de concurrence d'un système ; on peut en effet démontrer que le nombre de coupures cohérentes possibles est égal au nombre d'éléments 2 à 2 concurrents (||).

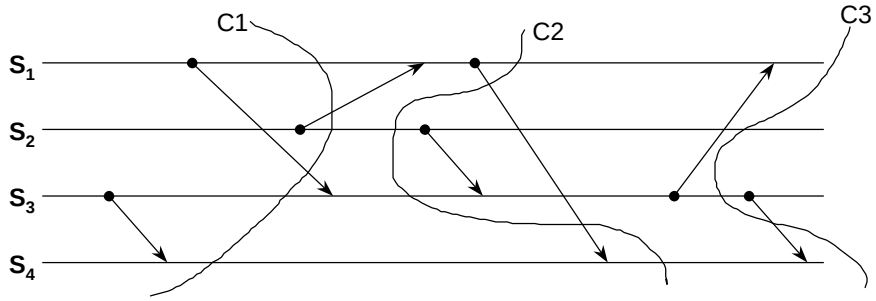
Exercice

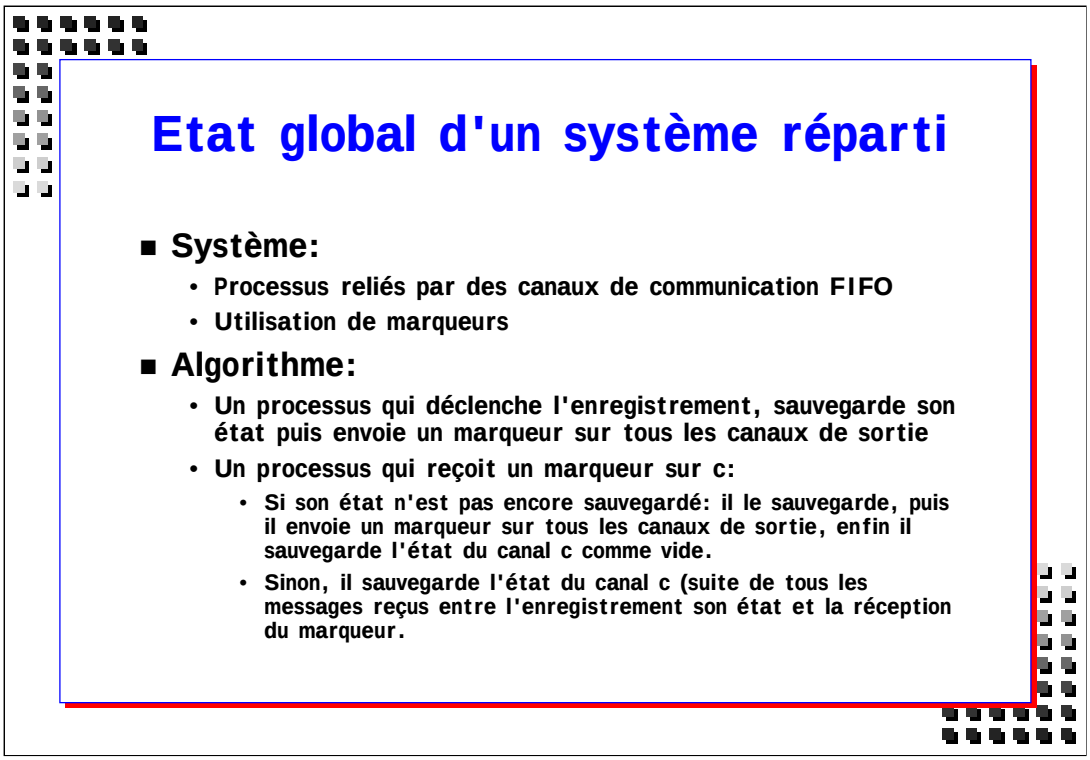
Soit 4 sites, et soit la coupure définie par les événements A, B, C et D sur chacun de ces sites. V_A, V_B, V_C et V_D désignant les dates respectives de A, B, C et D, la coupure ainsi définie est elle cohérente?

a) V_A=[2,0,0,0], V_B=[2,1,0,0] , V_C=[1,0,2,1] , V_D=[0,0,0,4]

b) V_A=[2,0,0,0], V_B=[2,3,3,1] , V_C=[1,0,3,1] , V_D=[1,0,4,2]

Les coupures C1, C2 et C3 des schémas ci-dessous sont elles cohérentes?

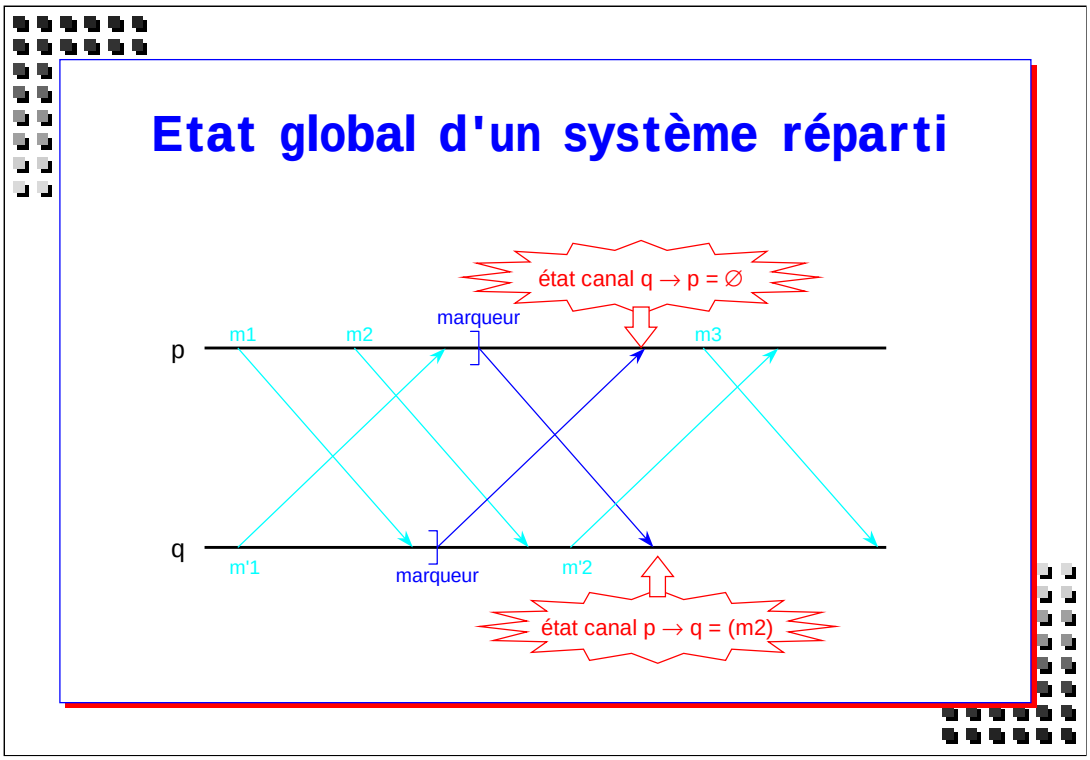




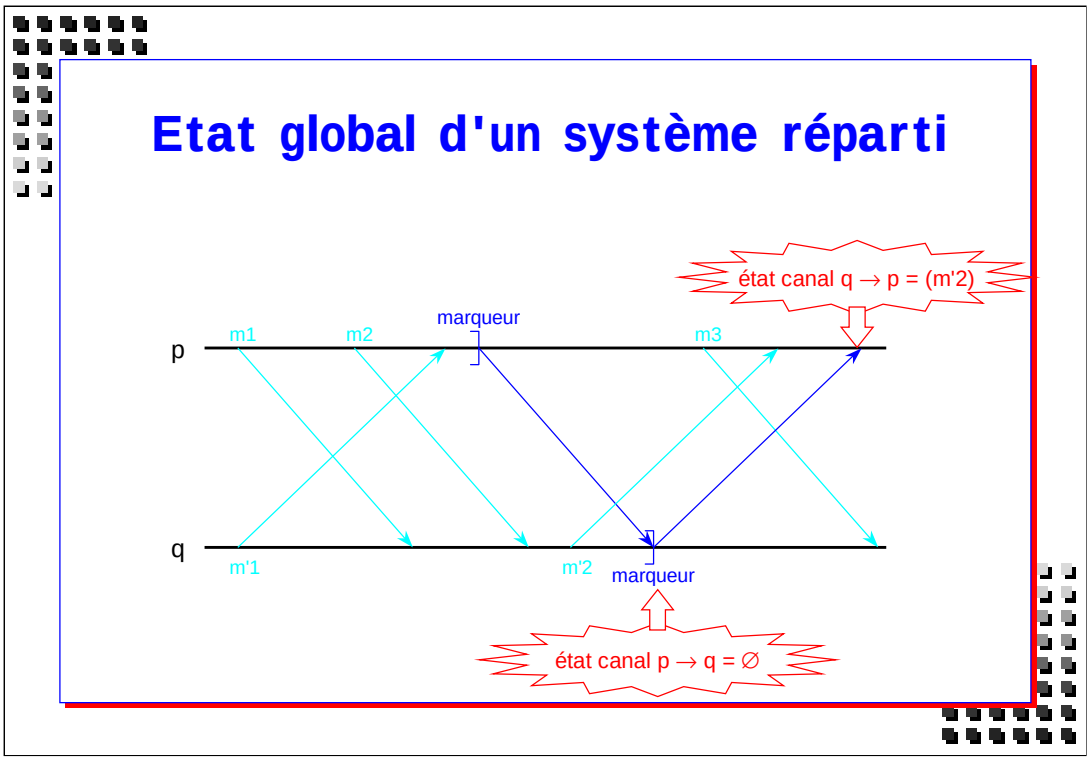
Nous décrivons maintenant le principe d'un algorithme pour l'enregistrement de l'état global d'un S.R. L'enregistrement peut-être déclenché par un processus quelconque, et éventuellement par plusieurs processus en parallèle.

Le système est constitué d'un ensemble de processus reliés par des canaux de communications sur lesquels les messages sont reçus dans l'ordre d'émission (FIFO). L'algorithme repose sur l'utilisation d'un message particulier, nommé marqueur, qui est envoyé sur chaque canal et permet de séparer les messages envoyés avant et après la sauvegarde de l'état local par le processus émetteur.

Fonctionnement de l'algorithme: cf. ci-dessus.



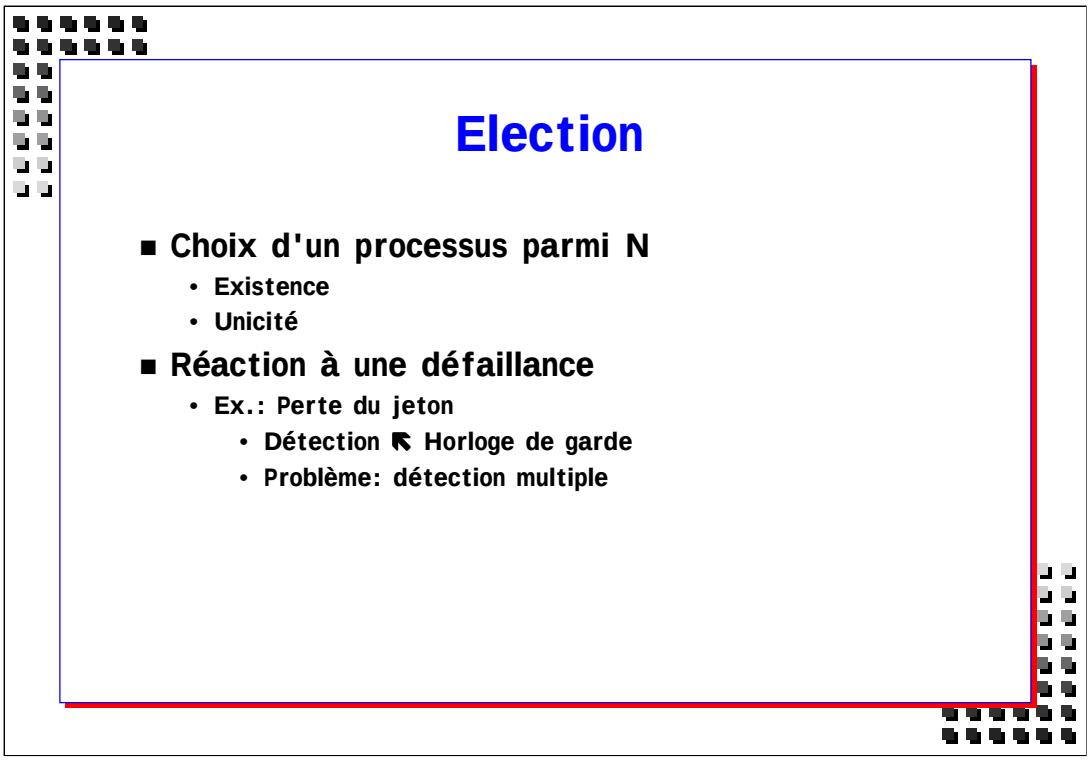
Cet algorithme garantit que l'état initial d'un canal de p vers q est bien enregistré comme la suite des messages émis par p avant l'enregistrement de p et reçus par q entre l'enregistrement de son état et la notification de l'enregistrement de p. En d'autres termes, l'état du canal est constitué des messages qui "traversent" la coupure définie par l'enregistrement des états de p et q.



On peut noter que l'état global enregistré ne correspond pas nécessairement à un état effectivement atteint par le système. Néanmoins, cet état est cohérent dans le sens ou il est état intermédiaire dans une séquence d'événements qui est une permutation légale d'une séquence réelle.

Certains algorithme permettent de supprimer la propriété FIFO sur les canaux de communication.

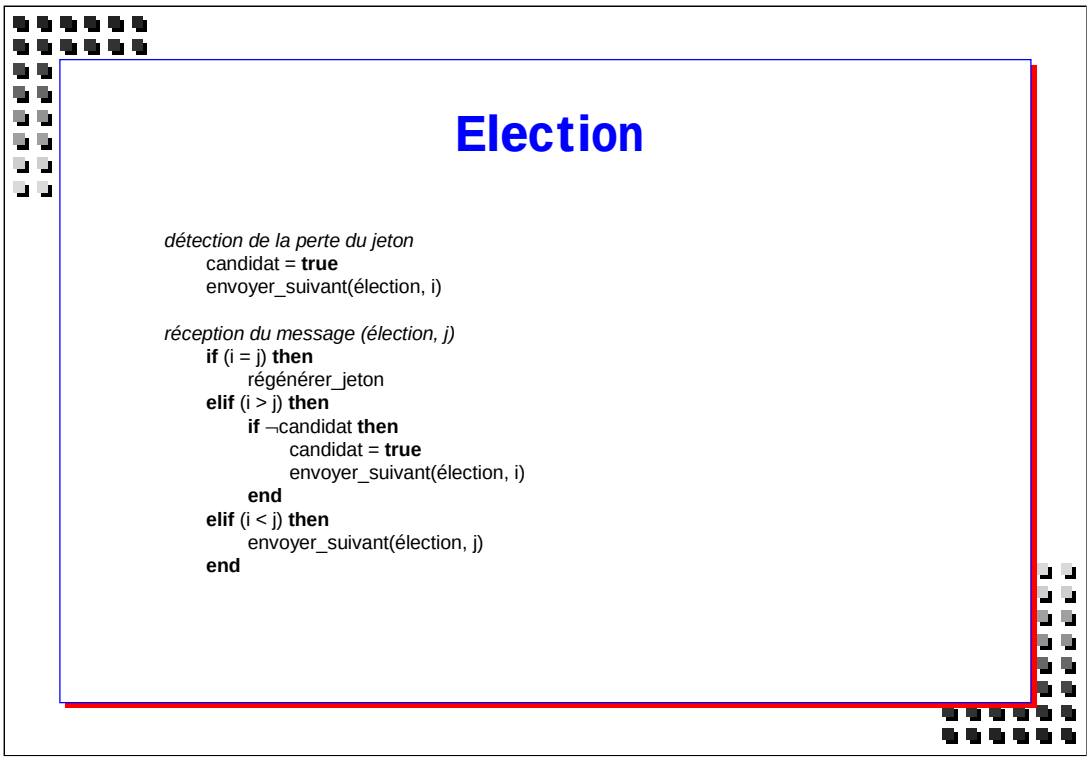
Algorithmique répartie



Le problème de l'élection consiste à choisir un processus et un seul parmi plusieurs, l'identité de processus élu étant indifférente: les propriétés requises sont l'existence et l'unicité.

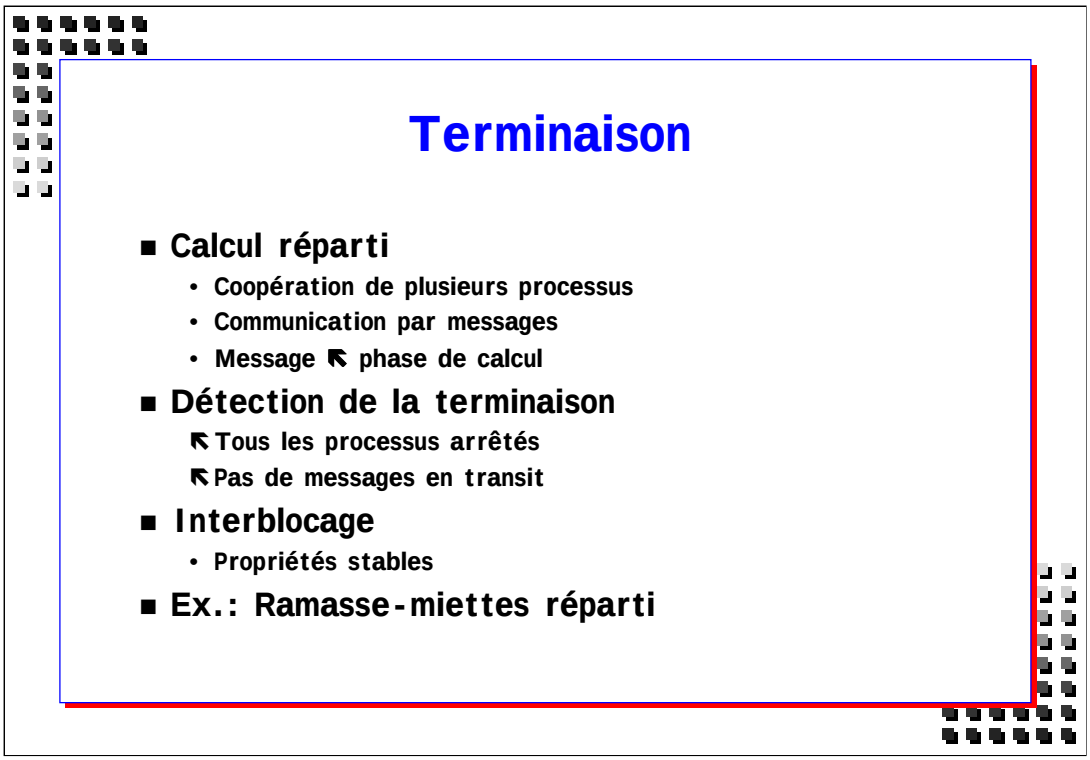
L'exemple typique de l'utilisation d'algorithme d'élection est la régénération d'information lors d'une défaillance.

Chaque site peut détecter la perte du jeton au moyen d'une horloge de garde ; le choix du délai de garde nécessite de connaître une limite supérieure à la durée d'un tour du jeton. Il faut alors éviter que plusieurs procesus régénère le jeton.



Comme le choix de l'élu est arbitraire, on choisit le processus restant de plus grand numéro. L'algorithme présenté procède par filtrage: chaque processus qui détecte la défaillance transmet un message de candidature qui contient son numéro.

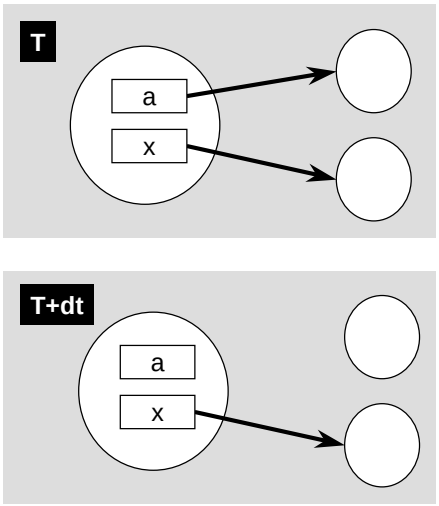
Lorsqu'un processus reçoit un tel message, il l'arrête s'il porte un numéro inférieur au sien et il devient lui même candidat si ce n'était pas déjà le cas. Dans le cas contraire il retransmet le message. Seul le message du processus de plus grand numéro fait donc un tour complet et revient à son émetteur. Celui-ci peut alors régénérer le jeton.

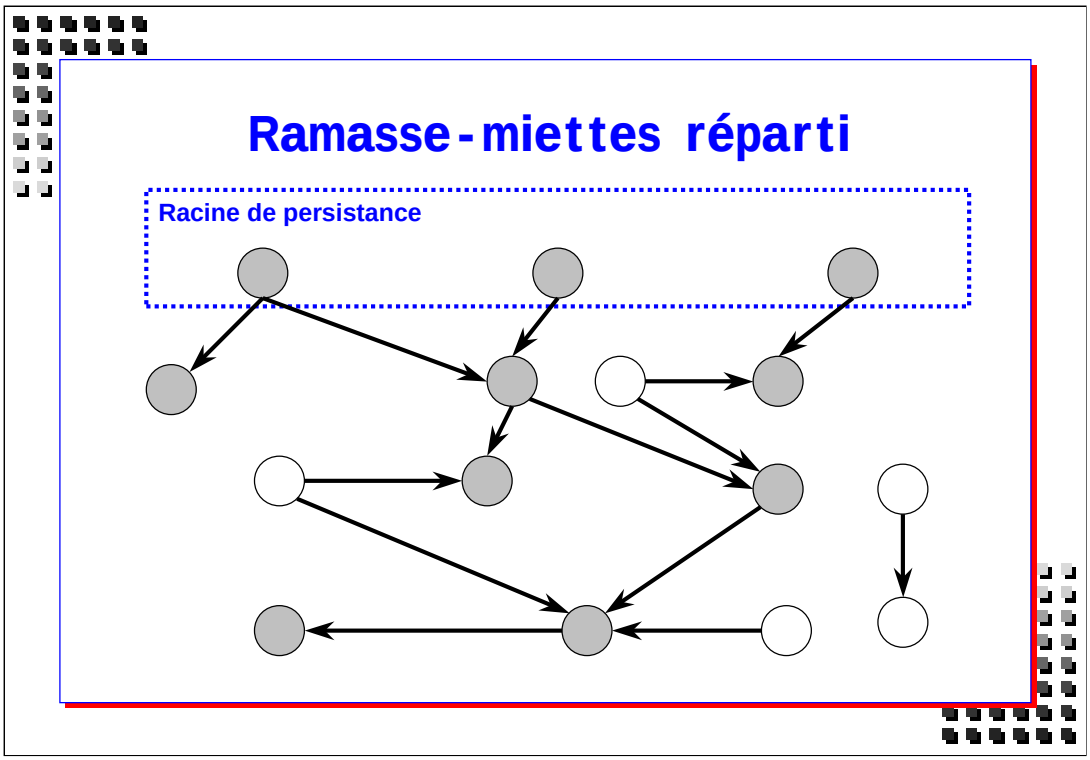


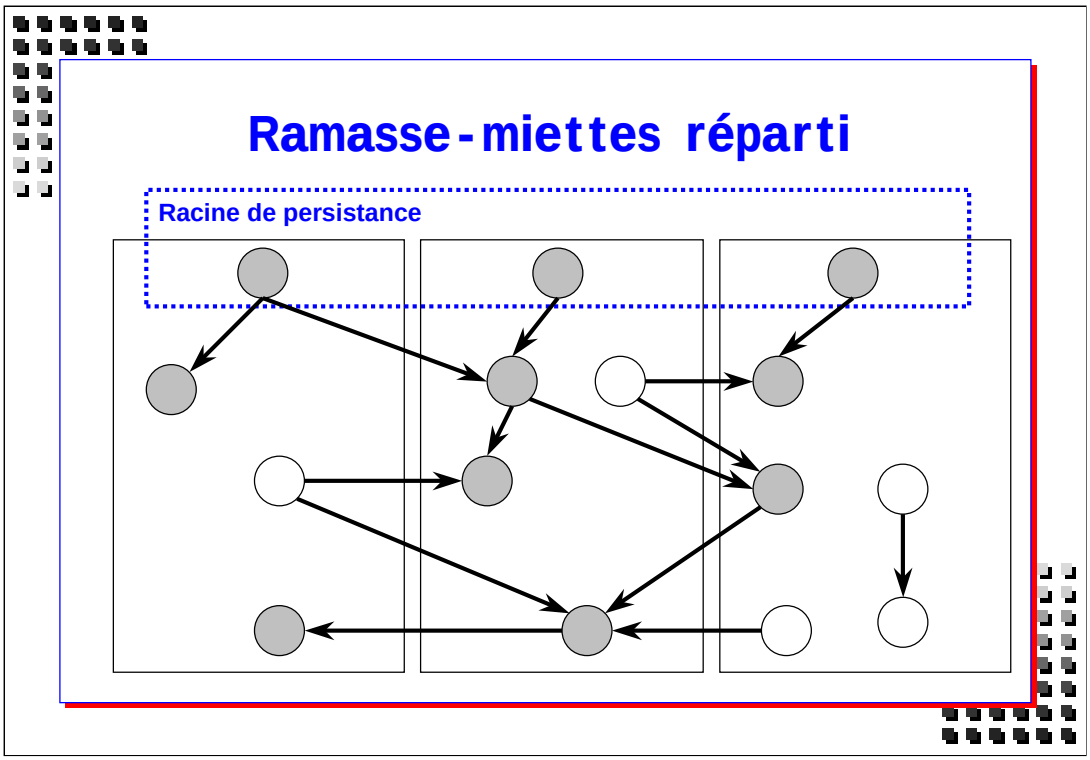
Le problème de la terminaison est celui de la fin d'un calcul réparti dans lequel plusieurs processus coopèrent.

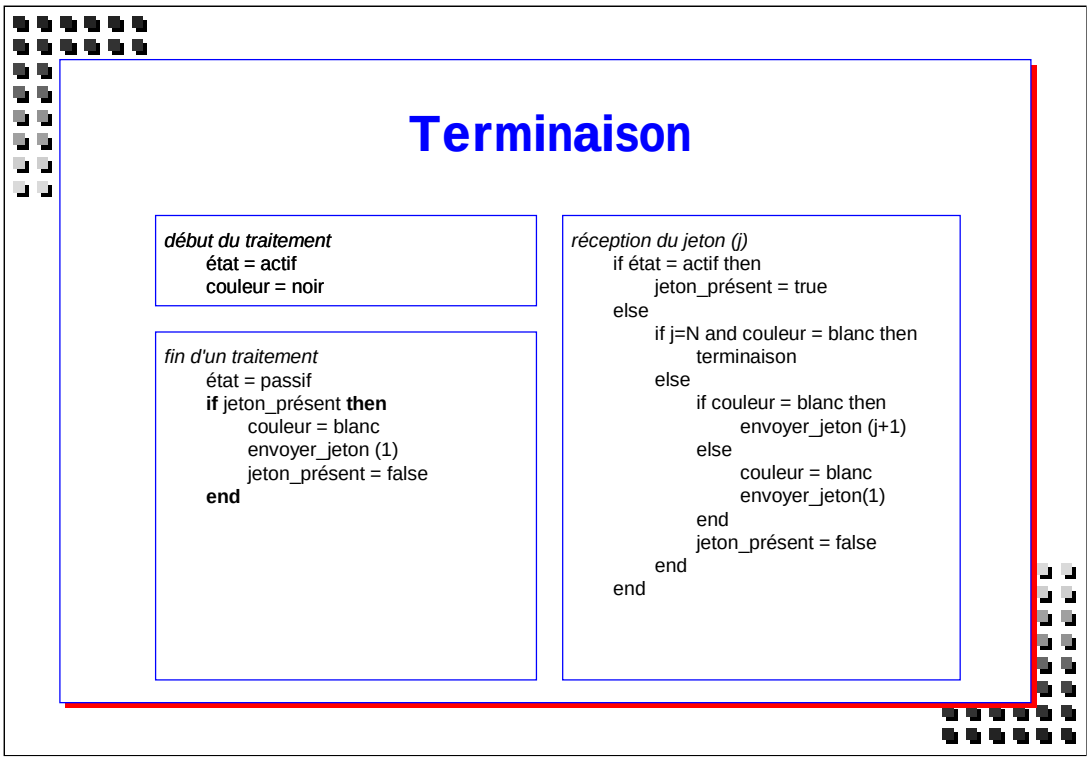
Ramasse-miettes réparti

T	a = new Object();
…	…
T+dt	a = null;




Ramasse-miettes réparti
Racine de persistance


Ramasse-miettes réparti
Racine de persistance

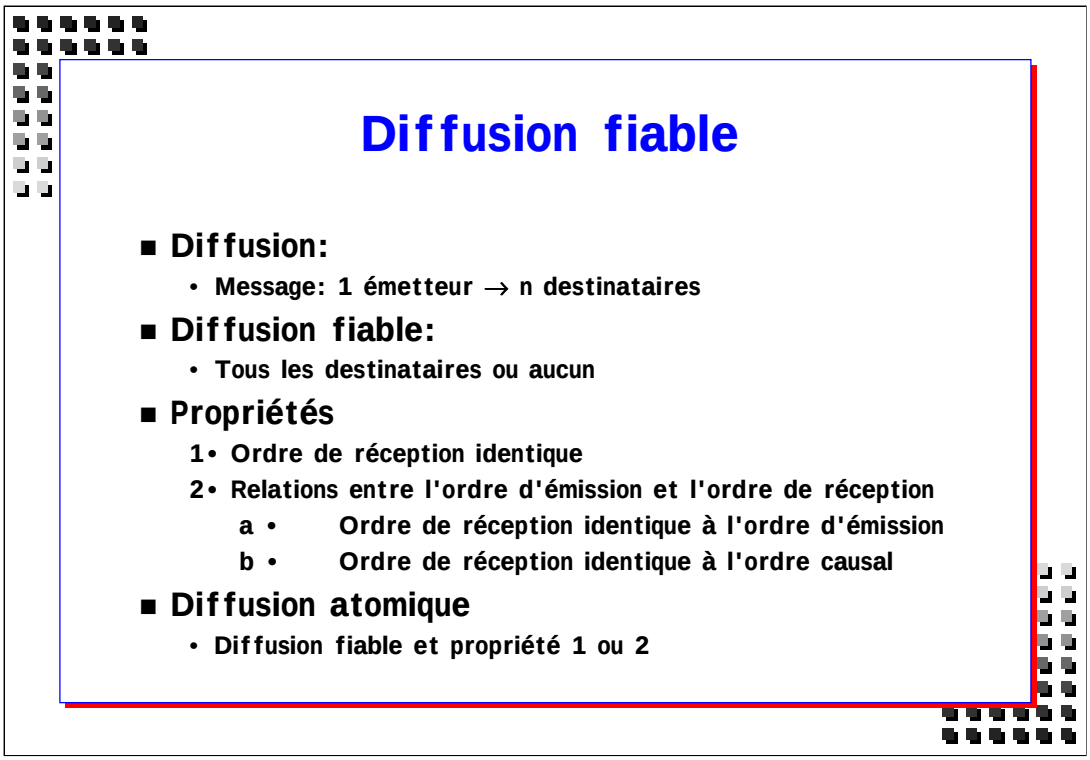


Nous présentons un algorithme de détection de la terminaison utilisant un jeton. L'ensemble des processus est organisé en anneau virtuel, qui est le seul moyen de communication entre les sites ; on suppose que les messages ne peuvent pas se doubler sur l'anneau.

Si chacun des sites a été visité 2 fois de suite par le jeton et qu'il est resté passif entre les 2 passages, on peut affirmer que la terminaison est détectée.

En effet, les messages en transit lors de la 1ère visite lui sont nécessairement parvenu lors de la seconde et il ne peut plus y en avoir en transit puisque les autres sites ont été observés passifs.

On associe à chaque site une couleur (blanc ou noir). Un site qui reçoit le jeton ne le renvoie que lorsqu'il est inactif, le site est alors mis à blanc. Lorsqu'un site devient actif, il est mis à noir. Lorsqu'un site blanc reçoit le jeton, on peut affirmer qu'il est resté passif depuis la dernière visite. Le jeton est muni d'un compteur qui enregistre le nombre de sites trouvés à blanc, la terminaison est détectée lorsque ce compteur est égal au nombre de sites.



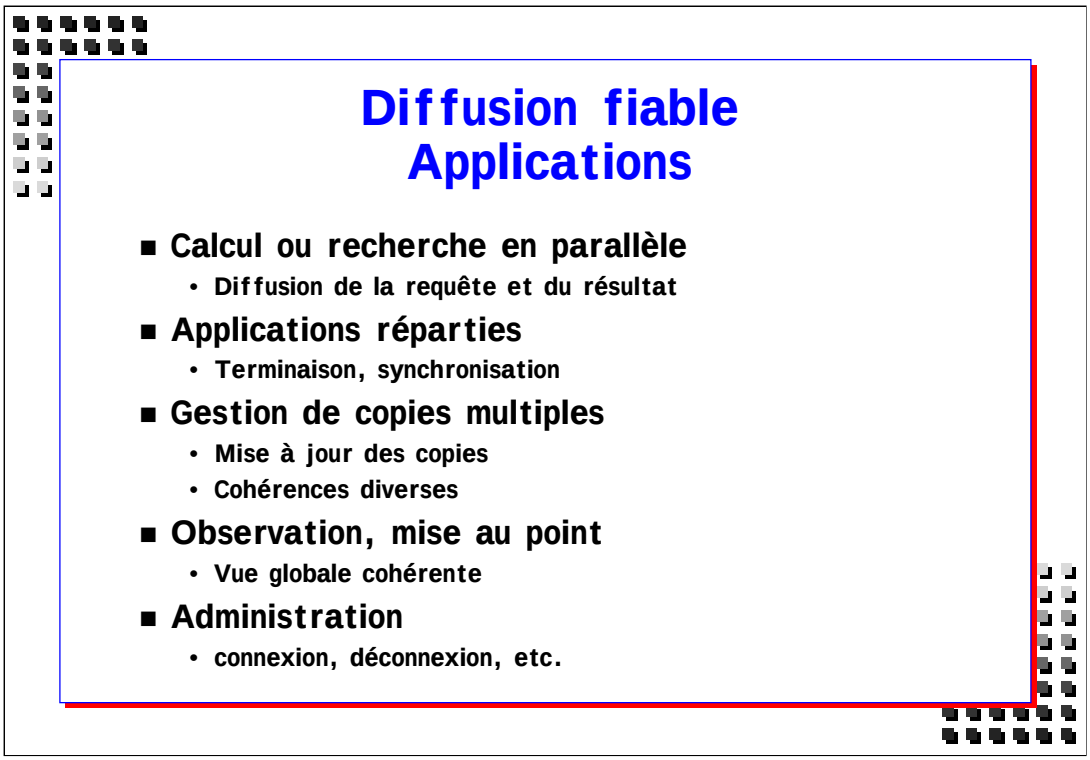
Etant donné n sites reliés par un réseau de communication, la diffusion d'un message est l'envoi de ce message à destination de chacun des sites du groupe (l'émetteur peut ou non appartenir au groupe).

Pour que la diffusion soit fiable, tous les destinataires du message (qui ne sont pas en panne) doivent recevoir le message (en cas de panne de l'émetteur durant la diffusion, aucun ne doit le recevoir). Un protocole de diffusion fiable doit résister aux pannes du système de communication et des sites.

La propriété de fiabilité ne spécifie rien sur l'ordre d'émission et de réception des messages ; on peut spécifier les propriétés suivantes:

- Ordre de réception identique pour tous les destinataires:
    - 1émetteur et 1 groupe destinataire
    - 1 ensemble d'émetteur et 1 groupe destinataire
    - 1 ensemble d'émetteur et 1 ensemble de groupe destinataire
- Relation entre l'ordre d'émission et l'ordre de réception
    - ordre de réception identique à l'ordre global d'émission
    - ordre de réception identique à l'ordre causal d'émission

Cette dernière propriété (2b) est indépendante des propriétés d'uniformité (1). En effet, si 2 processus diffusent chacun un message vers un groupe de destinataires et si ces 2 diffusions sont des événements causalement indépendants, la propriété 2b n'impose aucune contrainte sur l'ordre relatif de réception de ces 2 messages.



L'intérêt de dissocier l'ordre causal de la propriété d'uniformité, réside dans le fait qu'un protocole causal peut être réalisé de manière beaucoup plus efficace si l'uniformité n'est pas imposée. En effet, les protocoles qui respectent l'uniformité imposent en général un ordre total sur l'ensemble des messages, alors que l'ordre causal est un ordre partiel.

• Calcul ou recherche en parallèle: diffusion de la requête, du résultat.

• Applications réparties: utilisation de la diffusion fiable pour réaliser des algorithmes de terminaison ou de synchronisation.

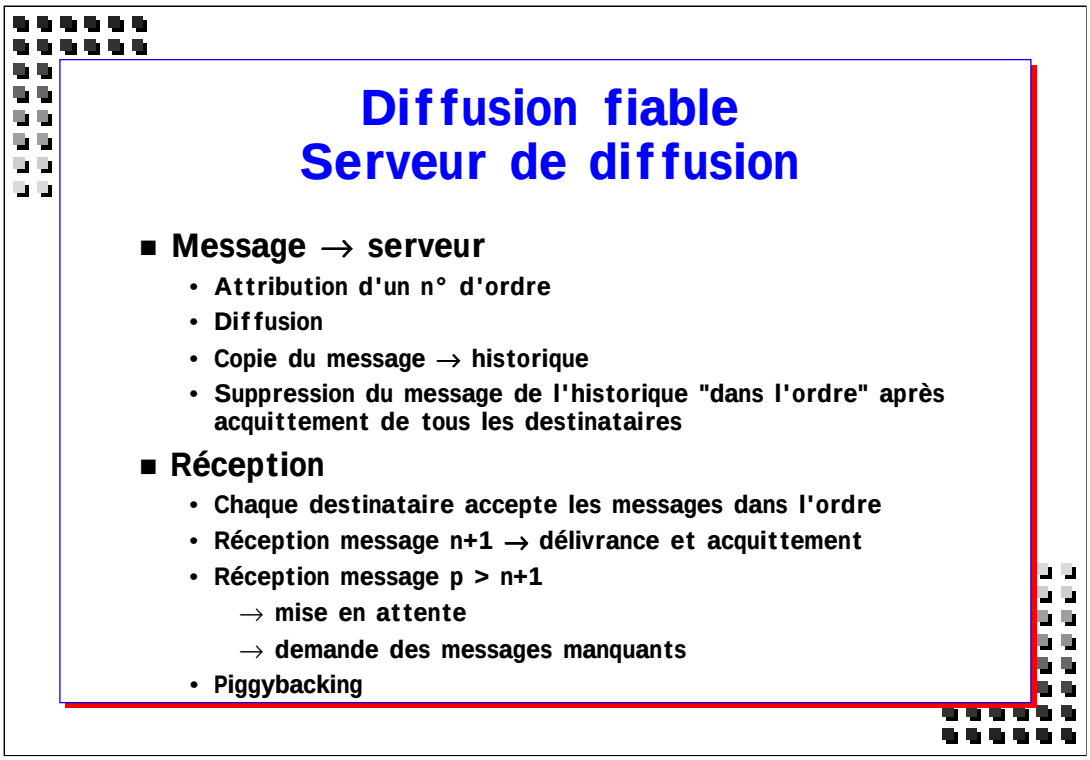
• Gestion de copies multiples: mise à jour des copies. Les propriétés de conservation de l'ordre permettent de mettre en place les diverses politiques de cohérences.

• Observation, mise au point: lorsque l'ordre causal est respecté, la diffusion fiable permet à un utilisateur d'avoir une vue globale cohérente de l'ensemble du système.

• Administration: réalisation de protocoles de connexion, déconnexion de sites, etc.

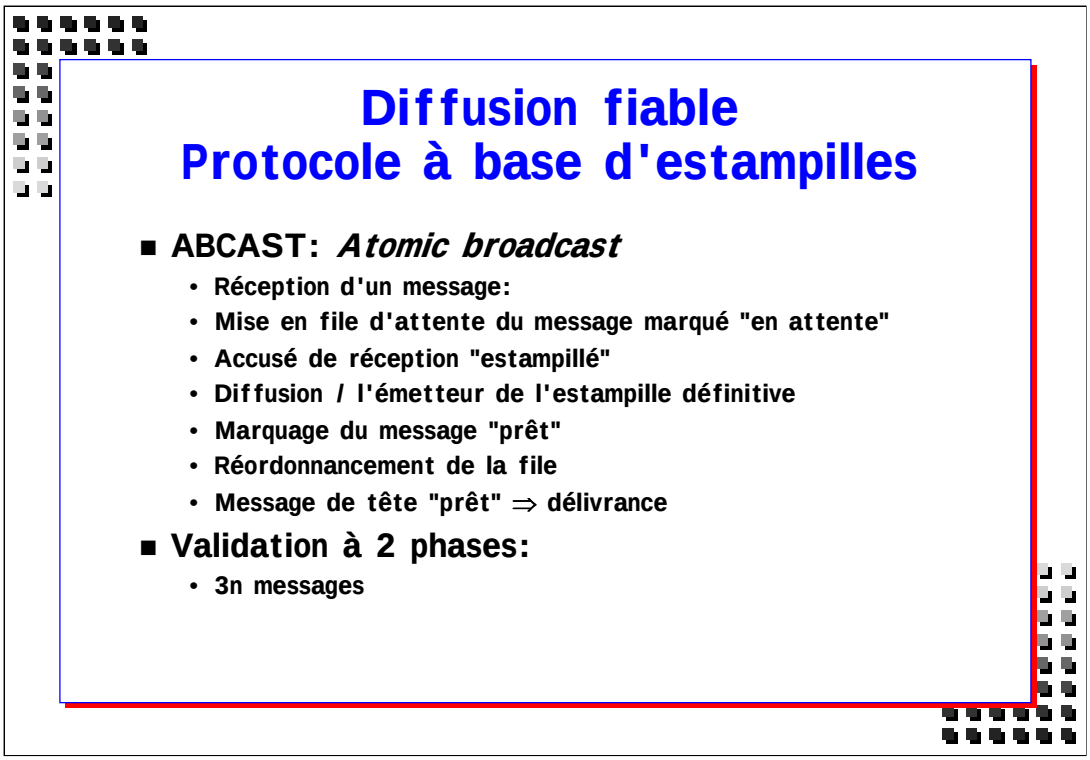
Par ailleurs, nous pouvons noter que différents systèmes de communication (l'anneau ou le bus à diffusion) réalisent déjà une diffusion à un niveau élémentaire.

Nous allons étudier maintenant différents protocoles de diffusion fiable.



Pour diffuser un message, un processus l'envoie au serveur de diffusion. Celui-ci lui attribue un n° d'ordre, le diffuse à tous les sites et le conserve en mémoire dans une file historique.

Chaque site conserve le n° d'ordre du dernier message reçu ; lorsqu'un destinataire reçoit un message, si ce message est son successeur direct il le délivre et envoie un acquittement



Le protocole ABCAST résiste aux défaillances des sites et assure un ordre de réception uniforme par ordonnancement global des messages. Cet ordonnancement est réalisé au moyen d'estampilles.

Un site qui reçoit un message le met dans une file d'attente locale et le marque "en attente". Il renvoie à l'émetteur un accusé de réception estampillé par la date de réception du message.

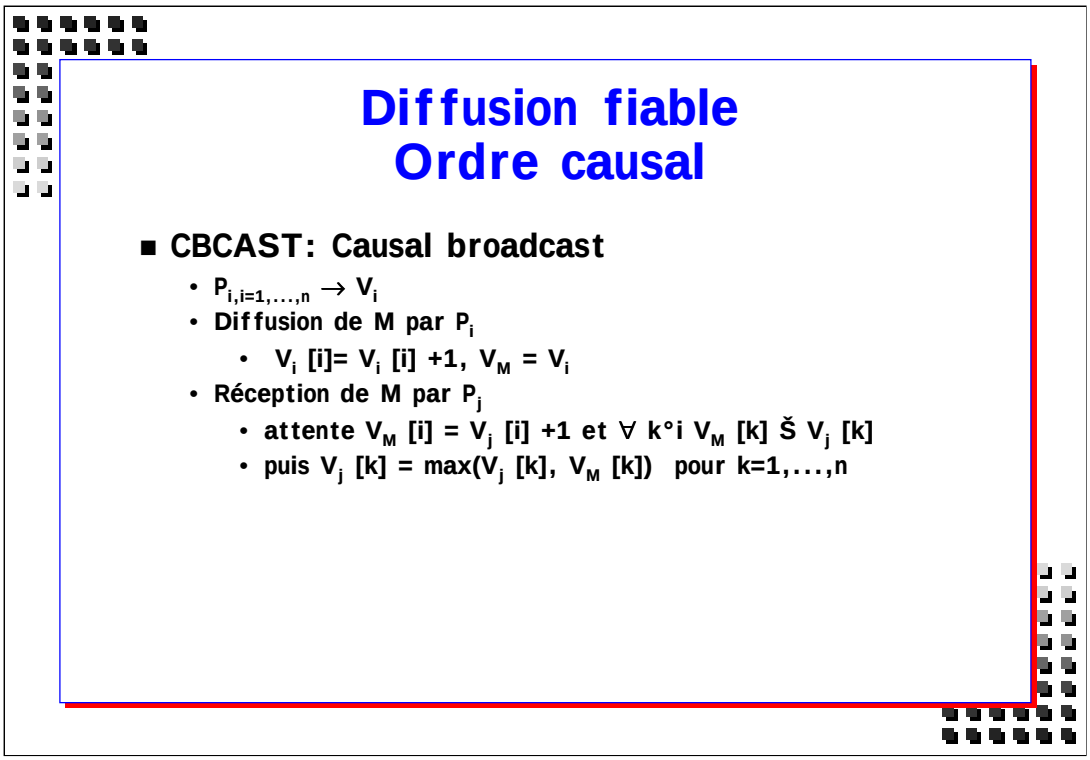
Lorsque l'émetteur a reçu tous les accusés de réception, il attribue une estampille définitive au message égale à la plus grande des estampilles renvoyées. Puis il diffuse cette estampille à tous les sites.

Chaque site affecte alors cette estampille au message en file d'attente, marque celui-ci "prêt" et réordonne la file dans l'ordre croissant des estampilles. Si le message de tête est marqué prêt, il est sorti de la file et délivré au processus destinataire.

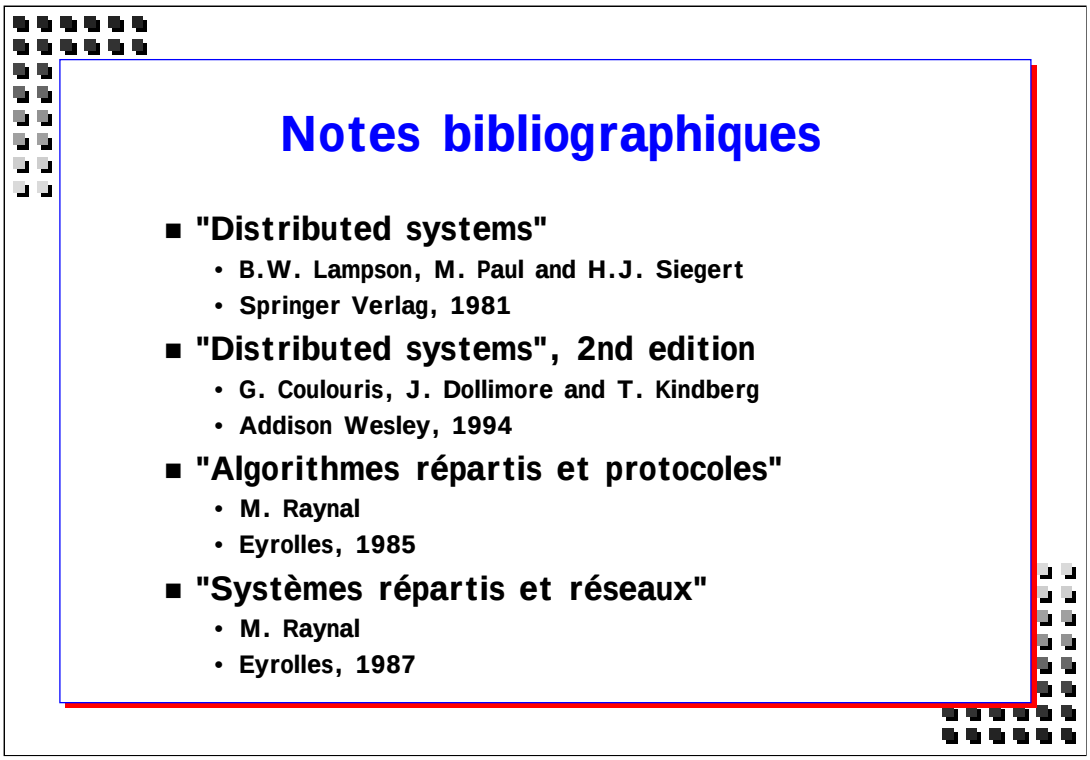
Ce protocole assure que 2 messages se trouvent dans le même ordre dans toutes les files d'attente ou ils se trouvent (uniformité). Il nécessite 3n messages pour une diffusion vers n sites en l'absence de pannes.

En cas de panne de l'émetteur avant la fin de la diffusion de l'estampille, un des sites destinataires est élu pour terminer le travail. A cet effet, pour chacun des messages dans sa liste il interroge les autres processus ; si l'un d'eux possède le message dans l'état prêt il diffuse alors la valeur correspondante de l'estampille sinon il rediffuse le message comme s'il n'avait jamais été diffusé.

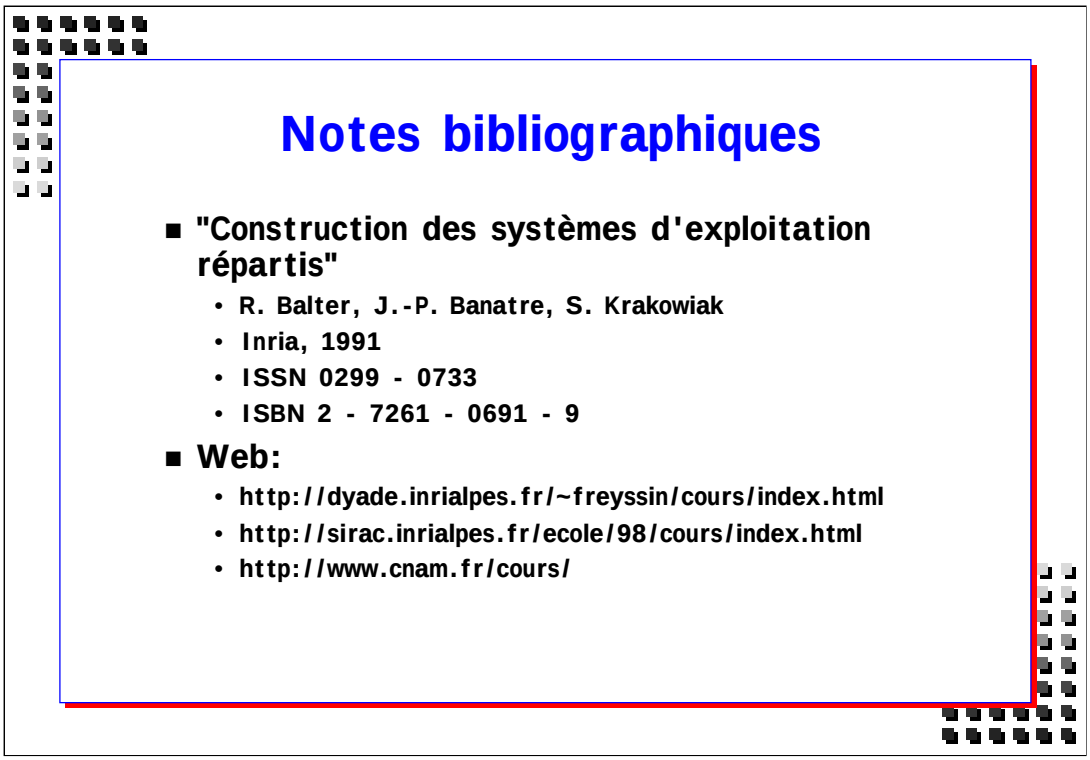
Notons que ce protocole de reprise nécessite de conserver tous les messages, et de les détruire ultérieurement par un algorithme de ramasse-miettes.



Le protocole CBCAST est un protocole de diffusion qui respecte la dépendance causale. Une première réalisation reposait sur la transmission avec tout message d'un historique des messages reçus et émis par le site émetteur ; diverses optimisations ont permis de réduire cet historique en remplaçant



[Lampson81] et plus récemment [Coulouris94] sont des ouvrages généraux sur les systèmes répartis. Pour l'algorithmique répartie on peut se reporter à [Raynal85] et [Raynal87].



Le livre "Construction des systèmes d'exploitation répartis" est le résultat d'une école sur la construction des systèmes distribués, de nombreux exposés de mon cours en sont issus.