Université Paul Sabatier TOULOUSE III

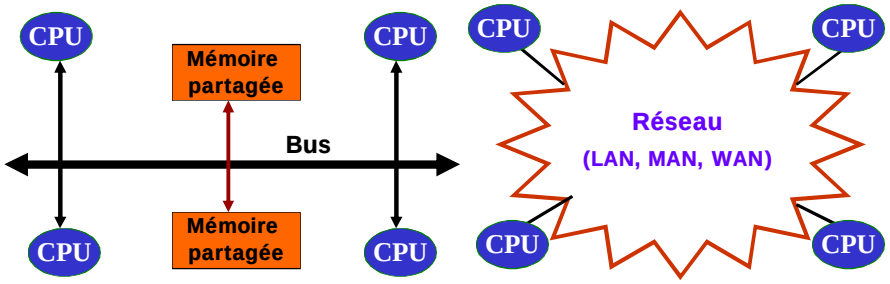
M1 Informatique
2004-2011

Concepts et fonctions de base des systèmes répartis

M1 Info – Cours de Systèmes répartis

Zoubir Mammeri

1. Introduction



- Systèmes fortement couplés (systèmes parallèles)
- Systèmes faiblement couplés (systèmes répartis)

Domaines d'applications

- **Très diversifiés**
  - Calcul scientifique
  - Simulation distribuée
  - Multimédia
  - Téléconférence
  - Travail coopératif
  - Installations industrielles
  - Robotique
  - Télécommunications
  - Bourse, finance
  - Réalité virtuelle
  - Intelligence artificielle, Multi-agents
  - Jeux en réseaux
  - ...

Objectifs de la répartition

- **Très variés**
  - Partage de ressources
  - Tolérance aux fautes (disponibilité, fiabilité...)
  - Plus de capacités de stockage
  - Plus de capacités de traitement
  - Répartition géographique de l'environnement du système informatique
  - Extensibilité et développement incrémental
  - Flexibilité
  - Réduction des coûts
  - ...

Définitions

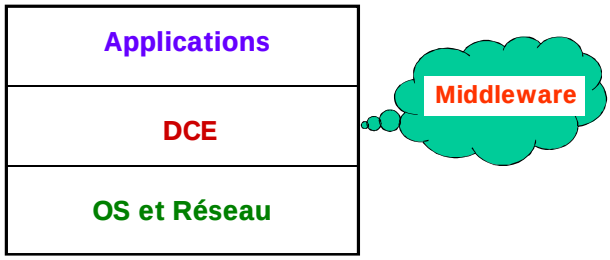
- « Un système réparti est un ensemble de processeurs (sites/nœuds) interconnecté par un réseau dans lequel chaque processeur a sa propre mémoire et ses propres périphériques ».
- « Un OS réparti est un OS qui apparaît aux utilisateurs comme un OS centralisé, mais qui s'exécute sur plusieurs CPU indépendantes et interconnectées ».
- « Un système réparti est un système dans lequel une machine dont vous n'avez jamais entendu parler auparavant vous empêche de travailler ».

Propriétés attendues d'un SR

- **Transparence**
  - d'accès aux informations
  - de localisation des objets
  - de migration des objets
  - de réplication des objets
  - des fautes
  - de performance
- **Fiabilité**
  - tolérance aux fautes
  - évitement de fautes
  - détection et recouvrement de fautes
- **Flexibilité :** facilité d'utilisation, de maintenance, de mise à jour…
- **Hétérogénéité** (du matériel et logiciel)
- **Performance** (optimisation de la gestion des ressources)
- **Sécurité**

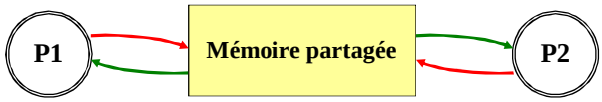
DCE (Distributed Computing Environment)

- **DCE =** ensemble de services et outils que l'on peut installer au-dessus d'un système existant pour exécuter des applications réparties

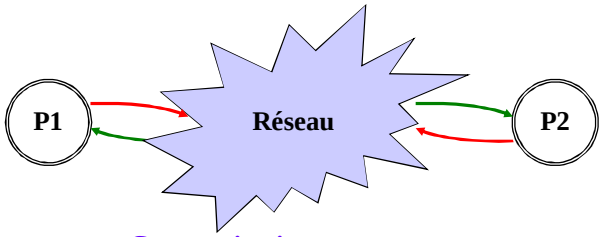


- **Composants usuels de DCE**
  - Service de gestion de threads
  - Appels de procédures distantes (RPC)
  - Service de temps
  - Service de noms
  - Service de sécurité
  - Service de fichiers répartis

Communication par messages (*Message passing*) (1/4)



Communication par mémoire partagée



Communication par message

Exemple de langage : MPI (*Message Passing Interface*) (2/4)

Fonction	Description
`MPI_send`	Envoi de message à un processus
`MPI_receive`	Réception de message
`MPI_Bcast`	Diffusion de message à tous les processus du groupe
`MPI_Init`	Initialisation de la librairie MPI
`MPI_Finalize`	Terminaison de la librairie MPI
`MPI_Comm_Size`	Retourne le nombre de processus MPI communicants
`MPI_Comm_rank`	Retourne l'identifiant du processus courant MPI

Fonctions de base de MPI

Exemple de code MPI (3/4)

```
#include <mpi.h>
void main(int argc, char **argv)
{
  int np, rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &np);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf(" Hello World du Processus %d sur
     un total de %d processus\n“, rank, np);
  MPI_Finalize();
}
```

Affichage du message *Hello World* en MPI par tous les processus du groupe MPI

`MPI_COMM_WORLD : constante MPI qui contient tous les processus du groupe`

Exemple de code MPI (4/4)

```
.....
intrank, size, i, buf[1];
MPI_Statusstatus;
MPI_Init( &argc, &argv);
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (rank == 0) {
          for (i=0; i<100*(size-1); i++) {
                    MPI_Recv( buf, 1, MPI_INT, MPI_ANY_SOURCE,
                    MPI_ANY_TAG, MPI_COMM_WORLD, &status );
                    printf( "Msgfrom %d with tag %d\n",
                    status.MPI_SOURCE, status.MPI_TAG); }
  }
else {
          for (i=0; i<100; i++)
          MPI_Send( buf, 1, MPI_INT, 0, i, MPI_COMM_WORLD ); }
MPI_Finalize();
....
```

Envoi et réception de message en MPI

Propriétés d'un bon système de passage de message

- **Simplicité d'utilisation**
- **Sémantique uniforme**
- **Efficacité (surcoût faible)**
- **Fiabilité**
- **Correction**
  - **Atomicité**
  - **Ordre de livraison**
  - **Terminaison**
- **Sécurité**
- **Portabilité**

Sources des problèmes (1/2)

- Pas de mémoire commune
- Pas d'horloge commune
- Latence du réseau
  - Vues décalées, incohérentes ou fausses
  - Livraison dans le désordre
- Fautes du réseau (pertes, duplications…)
- Adressage
  - Implicite/explicite
  - Statique/dynamique
- Mobilité, migration et localisation

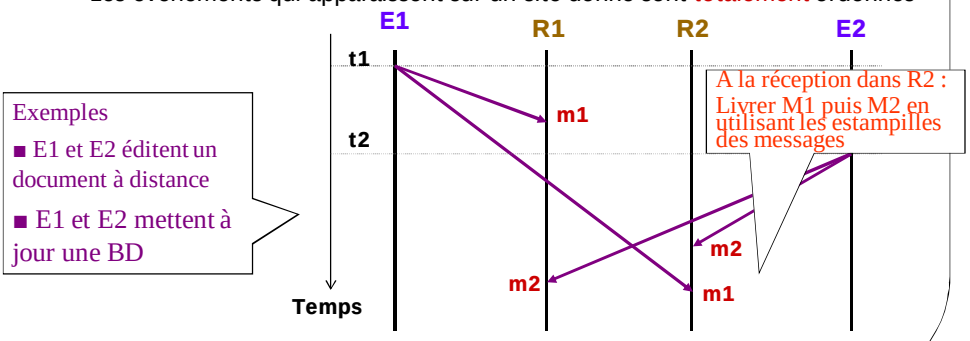
Sources des problèmes (2/2)

- Taille des buffers
  - Pas de buffer (appels bloquants)
  - Buffer à une place (appels bloquants)
  - Buffer à N places (appels bloquants)
  - Buffer à taille infinie
- Communication multipartenaire
  - Diffusion cohérente dans un groupe
  - Règles de formation et fonctionnement d'un groupe (droits et devoirs)
- Défaillances et tolérance aux fautes
- Sécurité

Protocoles et algorithmes pour cacher ces problèmes

2. Notions d'ordre

- **Ordre absolu** (ordre respectant le temps réel)
  - Livrer tous les messages à tous les récepteurs dans leur ordre d'émission
  - Si le réseau garantit une livraison ordonnée : Pas de problème d'ordre
  - Association d'estampilles aux messages
    Nécessité d'horloges synchronisées
  - Les événements qui apparaissent sur un site donné sont totalement ordonnés

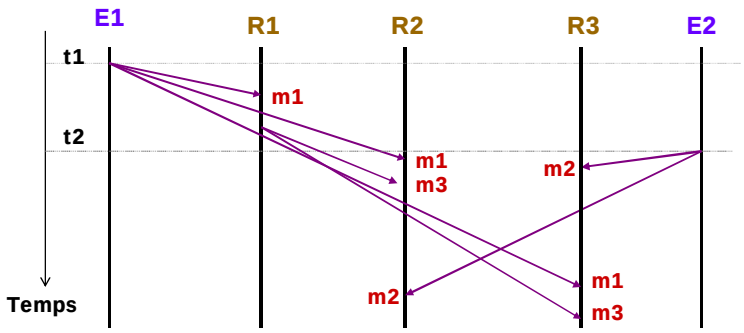


- **Ordre causal** (noté $\rightarrow$)
  - **deux événements sont liés par un ordre causal s'ils sont corrélés.**

  Ex. l'émission d'une réponse est "causalement" liée à la réception d'une requête

- $a \rightarrow b$ est vrai si l'une des conditions suivantes est vérifiée
  - les événements a et b ont eu lieu sur le même site et a s'est produit avant b.
  - a est l'émission d'un message et b la réception correspondante.
  - il existe un événement c tel que $a \rightarrow c$ et $c \rightarrow b$.
- La notion d'ordre causal traduit la notion de avant et de après (Lamport)
- L'ordre causal est un ordre partiel.
- $a \,||\, b$: Evénements concurrents (ou indépendants) si $a \rightarrow b$ et $b \rightarrow a$ sont faux.
- **Exemples d'utilisation de l'ordre causal**
  - Retrouver les causes de pannes ou d'anomalies de production de pièces
  - Retrouver des erreurs dans la conception d'un système réparti.
  - Tracer les événements dans un système de décision (bourse, ...)
  - Rechercher un diagnostic (de malade, de véhicule...)

Exemple d'ordre causal



E1 émet m1 vers R1, R2 et R3. E2 émet m2 vers R2 et R3. Sur réception de m1, R1 inspecte le message et crée un message m3 qu'il émet vers R2 et R3. Dans ce cas, l'émission de m3 est liée par un ordre causal à m1 :

Evt_émission(m1) → Evt_émission(m3)
Car on :
Evt_émission(m1) → Evt_réception(m1) ET Evt_réception(m1) → Evt_émission(m3)

Horloges logiques pour implanter l'ordre causal (Lamport)

- Chaque site i est doté d'un vecteur-horloge $VH_i[1..N]$ (N = nombre de sites). $VH_i[k]$ représente la connaissance qu'a le site i sur le nombre d'événements produits par le site k (k=1,..., N).

- Le vecteur-horloge VH_i progresse selon les deux règles :

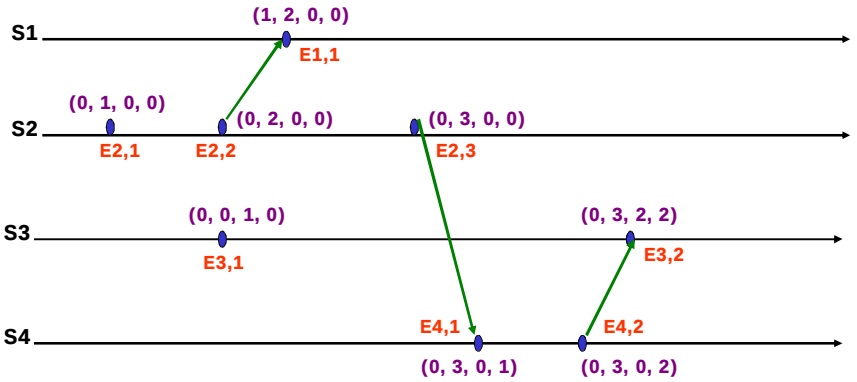
  R1 : Lorsque le site i produit un événement (interne, d'émission ou de réception) il incrémente $VH_i[i]$ de 1 au préalable.

  R2 : A tout message M est associé la valeur du vecteur-horloge VH_k du site k au moment de l'émission de M par ce site k.
  A la réception de (M, VH_k), le site récepteur i incrémente $VH_i[i]$ de 1 et recale son vecteur-horloge VH_i avec la connaissance qu'il vient d'acquérir :

  $$\forall x \in [1..N] : VH_i[x] := \max\{VH_i[x], VH_k[x]\}$$

- Les règles R1 et R2 garantissent :
  - $VH_i[k]$ monotone croissant $\forall i, k$.
  - $VH_i[k] \leq VH_k[k]$ /* Ceci est une conséquence du fait que l'état du site k parvient au site i avec un certain décalage */

Exemple de progression des horloges logiques



- a et b deux événements estampillés par deux vecteurs Va et Vb :
  - $a \rightarrow b \Leftrightarrow Va < Vb$.
  - a et b indépendants $\Leftrightarrow Va \,||\, Vb$ ($Va < Vb$ et $Vb < Va$ sont faux)

Exemple : E2,2 $\rightarrow$ E1,1 et E3,1 || E2,1

Algorithme d'implantation de l'ordre causal
Protocole CBCAST (Causal multicast) - Birman 87

1. Chaque site k du groupe maintient un vecteur V_k à N composants (N : nombre de sites). Le composant $V_k[i]$ (i=1,…,N) indique le n° associé au dernier message reçu en provenance du site i.

2. Pour transmettre un message M, le site émetteur E incrémente son propre composant (i.e. $V_E[E]$) et envoie son vecteur V_E avec le message M.

3. Lorsque un message est reçu par un site R, il est mémorisé. Ensuite, le site R teste si le message satisfait les conditions (a) et (b) ci-dessous pour être délivré ou s'il doit attendre pour respecter l'ordre causal :

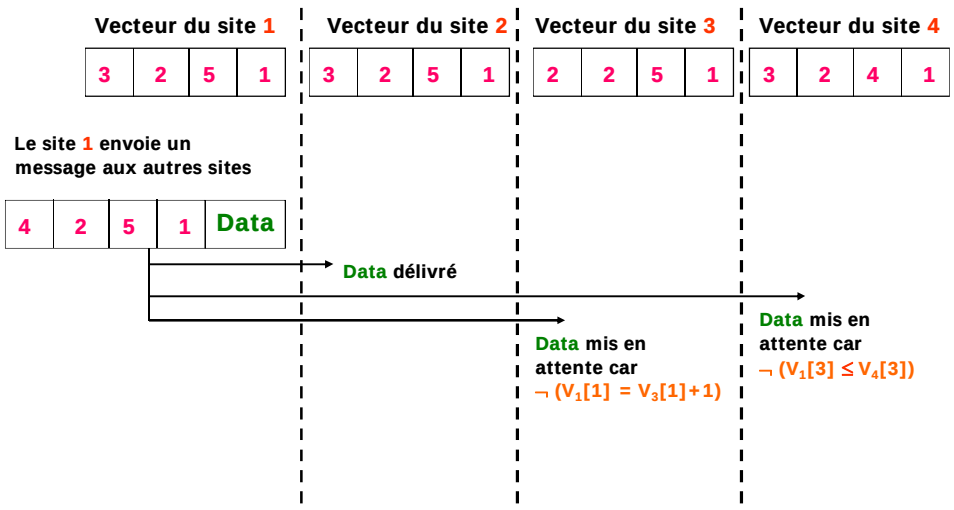
(a) : $V_E[E] = V_R[E] + 1$

(b) : $V_E[j] \leq V_R[j]$ pour tout $j \neq E$

La condition (a) garantit que le récepteur R n'a raté aucun message en provenance du site E (ceci est requis car : deux messages en provenance d'un même émetteur vers un même récepteur sont toujours causalement liés).

La condition (b) garantit que l'émetteur E n'a reçu aucun message que le site R n'a pas encore reçu (ceci est requis pour être sûr que le message de l'émetteur E n'est pas causalement lié à un message que le site R n'a pas encore reçu).

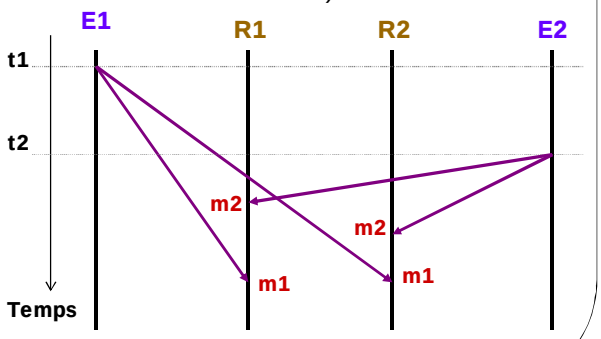
Exemple de fonctionnement du CBCAST



- **Ordre consistant** **(utilisé pour la diffusion d'un message à un groupe)**
  - Les événements qui apparaissent sur des sites différents peuvent être partiellement ordonnés
  - L'ordre absolu est difficile à obtenir car il nécessite la synchro des horloges.
  - Il n'est pas toujours indispensable (ex. deux transactions de m.à.j d'une BD peuvent être prises en compte dans n'importe quel ordre par le serveur de BD)
  - Ordre consistant = livrer tous les messages dans le même ordre (cet ordre pouvant être différent de celui de l'émission)

Exemple :

Une équipe doit traiter des problèmes posés par des clients différents (discuter d'un sujet, concevoir un produit, vendre des actions…). Mais pour des raisons de coopération, tous les membres de l'équipe doivent se focaliser sur un même problème avant de passer au suivant.



Algorithme d'implantation de l'ordre consistant
Protocole ABCAST (Atomic multicast) - Birman 87

1. Le processus émetteur (E) affecte un n° de séquence temporaire à son message M et l'envoie à tous les membres du groupe. Ce n° de séquence doit être plus grand que tous les n° d'accord que l'émetteur E a déjà vus.

2. A la réception du message M, chaque membre i du groupe:

   i) Met le message M en file d'attente avec le statut « non livrable »

   ii) Renvoie un n° de séquence qu'il propose à l'émetteur E calculé comme suit :

$$\#seq(i, M) = \max(F_{max}, P_{max}) + 1 + i/N$$

   N : nombre de sites /* i/N garantit l'unicité du n° d 'accord. */

   P_{max} : le plus grand n° de séquence que le site i peut proposer (il y a différentes possibilités de gestion de Pmax ; Pmax prend la valeur de #seq(i) proposée par le site i pour le message M-1, Pmax peut tenir compte de changement d'état interne au site i...)

   F_{max} : le dernier plus grand n° d'accord sur lequel les sites se sont mis d'accord pour la diffusion du message précédent (i.e. pour diffuser le message M-1).

Algorithme d'implantation de l'ordre consistant
Protocole ABCAST (Atomic multicast) - Birman 87 (suite)

3. Lorsque le processus E reçoit tous les n° proposés par les membres du groupe, il choisit le plus grand des n° proposés et le diffuse à tous les membres pour leur indiquer le n° de séquence final (dit n° de séquence d'accord).

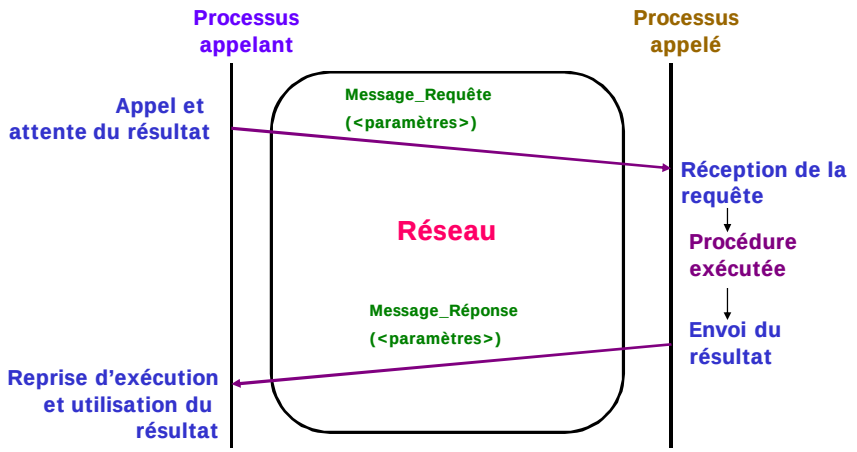
$$\#accord(M) = \max(\#seq(k, M), k=1,..., n)$$

4. A la réception du n° d'accord (#accord(M)), chaque membre du groupe attache au message M ce n° d'accord et marque le message M avec le statut « livrable ».

5. Les messages de données livrables sont passés à la couche application selon l'ordre des n° d'accord qui leur sont attachés.

3. RPC (Remote Procedure Call)

- **Mécanisme le plus répandu pour la communication entre processus distants (IPC pour processus distants)**

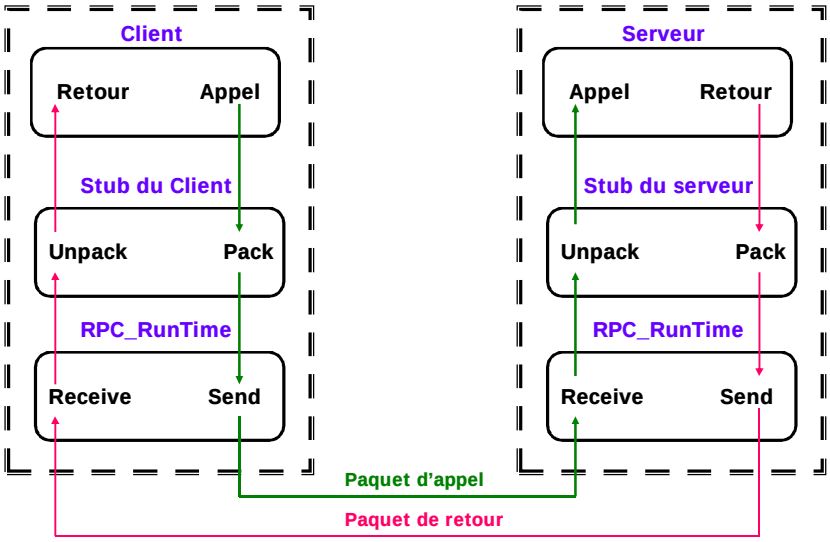


Propriétés des RPC

- Syntaxe simple
- Concept familier aux programmeurs
- Interface avec une sémantique claire et simple
- Versatile (pour la communication distante ou locale)

⇒ Disponible sur la plupart des OS

Implantation de RPC (1)



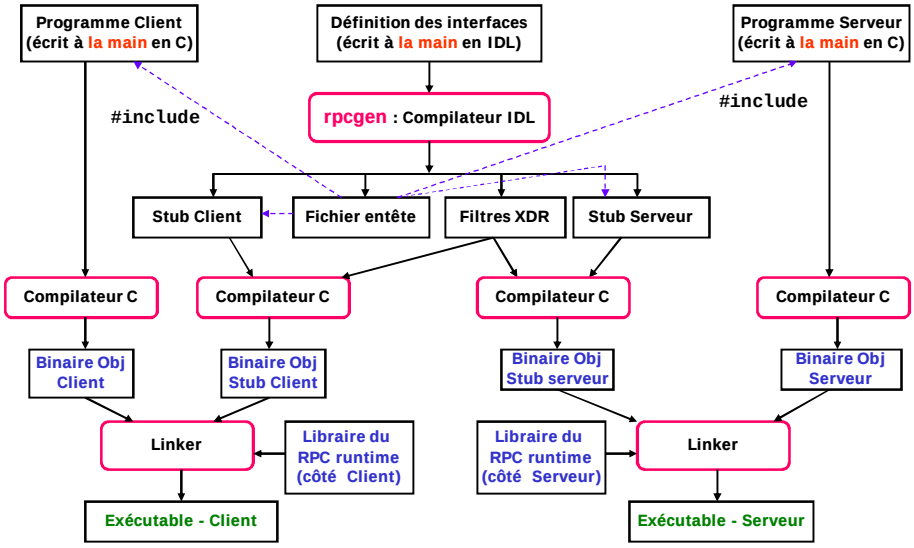
Implantation de RPC (2)

- **Client :** processus utilisateur qui initie l'appel de procédure distante
- **Stub client :** construction de message de requête à partir des paramètres d'appel et récupération des résultats à partir du message de réponse. Il peut être généré de manière manuelle ou automatique.
- **RPC RunTime :** partie de l'OS qui assure le transport de paquets.
- **Stub serveur :** construction de message de résultat à partir des paramètres de résultat et récupération des paramètres d'appel à partir du message de requête. Il peut être généré de manière manuelle ou automatique.
- **Serveur :** processus utilisateur qui exécute la procédure appelée.

Liaison Client-Serveur (Binding)

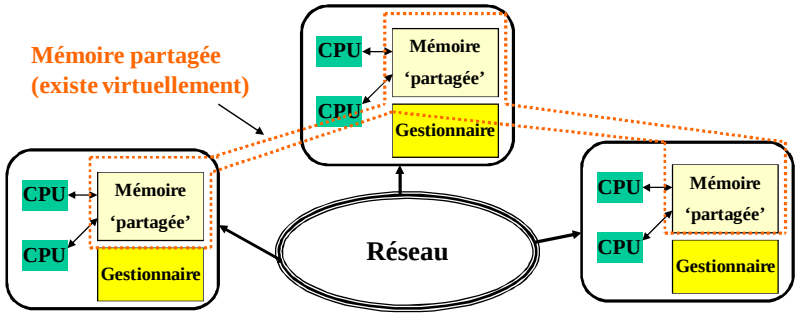
- **Le client doit établir un lien avec le serveur avant l'appel de procédure**
- **La liaison traite les questions suivantes**
  - Comment le client désigne-t-il (nomme-t-il) le serveur ?
  - Comment le service de liaison localise-t-il le serveur ?
  - La liaison est-elle faite à la compilation ? à l 'édition de lien ? à l'exécution ? Peut-elle changer durant l'exécution ?
  - Un client peut-il être lié à plusieurs serveurs simultanément ?
- **Il existe différentes réponses aux questions précédentes donnant lieu à différentes implantations possibles des RPC.**

Utilisation des RPC de SUN



4. Gestion de mémoire partagée

- **Une abstraction pour permettre à des processus distants d'accéder à une même mémoire**



X:= read(adresse/nom_variable)

write(adresse/nom_variable, Y)

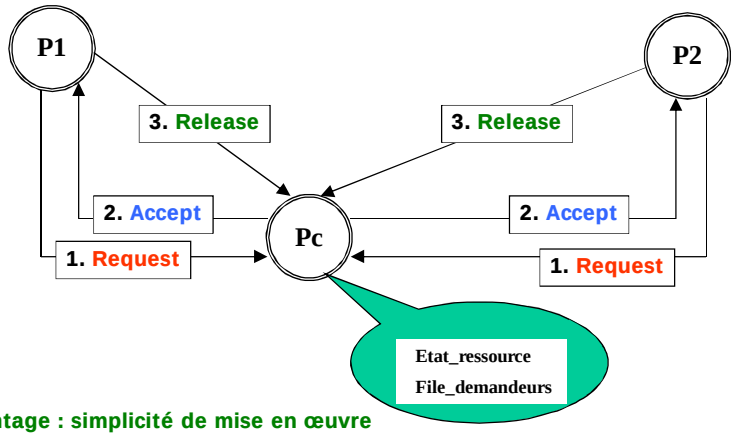
- **Problèmes à résoudre**
  - **Granularité (taille de blocs à gérer)**
  - **Localisation physique des données**
  - **Cohérence (spatiale et temporelle) des données (exacte ou approchée)**
  - **Contrôle des accès (réparti ou centralisé)**
  - **Migration de blocs mémoire (possible ou impossible)**
  - **Réplication et cohérence des copies (possible ou impossible)**
  - **Hétérogénéité matérielle des mémoires (possible ou impossible)**
  - **...**

5. Exclusion mutuelle

- Protection des ressources partagées
- En centralisé : sémaphore, signaux, verrous, moniteurs...
- En réparti, implanter l'exclusion mutuelle par :
  - Contrôle centralisé
  - Contrôle réparti
  - Utilisation de jeton circulant

Contrôle centralisé

- Toutes les requêtes d'accès sont adressées et gérées de manière centralisée



- Avantage : simplicité de mise en œuvre
- Inconvénient : non tolérante aux fautes

Contrôle décentralisé

- Coopération entre processus pour entrer en section critique
- Beaucoup de solutions ont été proposées
- Solution de Ricart et Agrawala

– Chaque processus qui veut entrer en SC envoie un message de demande à tous les autres. Le message contient l'Id du demandeur, le nom de la SC et une estampille.

– Quand un processus reçoit un message de demande d'entrer en SC, il répond ou non à ce message de manière suivante :

- Si le récepteur est lui-même dans la SC demandée, il met la demande en attente.

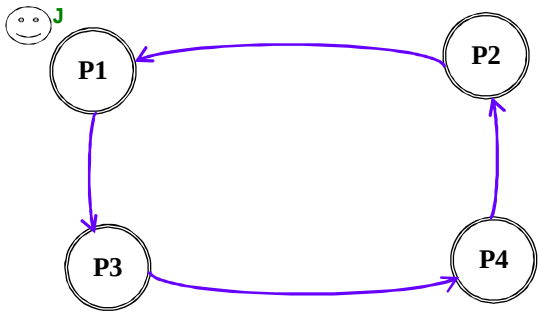
- Si le récepteur est lui-même intéressé par la SC (et qu'il a déjà envoyé sa demande), il compare son estampille à celle de l'émetteur. Si elle est plus grande, il répond à l'émetteur pour lui permettre d'entrer en SC. Sinon, il met la demande en attente.

- Si le récepteur n'est ni en SC ni intéressé par y entrer, il répond immédiatement au demandeur.

– Quand le demandeur reçoit les réponses de tous les autres sites, il entre en SC.

Contrôle avec jeton

- Idée analogue à celle de gestion du MAC dans les réseaux locaux



Interblocage (deadlock) dans les systèmes répartis

- **Problème analogue à celui connu dans les systèmes centralisés.**
- **Evitement d'interblocage** (en ligne)
  - Recenser les utilisations de ressources
  - Ne pas allouer de ressource (libre) s'il y a un risque d'interblocage
  - Le test d'allocation de ressources se fait en ligne.
- **Prévention d'interblocage** (hors ligne)
  - Concevoir l'application de sorte que l'interblocage devient impossible
  - Les situations d'interblocage sont écartées à la conception (i.e. hors ligne)
  - Techniques utilisées pour éviter l'interblocage :
    - + Toutes les ressources sont demandées en une seule fois
    - + Demandes de ressources ordonnées
    - + Préemption

- **Détection d'interblocage**
  - Construction du graphe d'attente
  - **Détection centralisée**
    - + Un site coordinateur reçoit les infos
    - + Infos échangées en continue, périodiquement ou à la demande
    - + Construction du graphe d'attente et détection d'interblocage
  - **Détection décentralisée**
    - + Tous les sites s'échangent des infos
    - + Infos échangées en continu, périodiquement ou à la demande
    - + Construction du graphe d'attente et détection d'interblocage
  - **Détection hiérarchique**
    - + Partitionnement de l'ensemble des sites
    - + Un site coordinateur par partition
    - + Les coordinateurs de partitions fonctionnent en décentralisé
    - + Permet de réduire les infos échangées

6. Problème d'élection

- **Beaucoup d'algorithmes répartis nécessitent l'élection d'un site pour contrôler (coordonner) certains aspects.**
- **Algorithme d'élection : sélection d'un membre élu par un groupe. Il doit garantir l'unicité de l'élu.**
- **Différentes formes d'élection**
  - Utilisation de priorités connues a priori pour élire le représentant
  - L'élection se fait selon un ordre connu d'avance (selon une boucle...)
  - …
- **Plus l'algorithme d'élection est équitable, plus il est coûteux à mettre en œuvre. On cherche surtout l'unicité de l'élu.**

Algorithme de Garcia-Molina

- **Existence d'un site coordinateur (choisi par une procédure d'initialisation)**
- **Chaque site a une priorité fixe et connaît les priorités des autres sites.**
- **De manière périodique ou non, chaque site S_i envoie une requête de présence au coordinateur. S'il ne reçoit pas de réponse, le site S_i considère que le coordinateur est hors service et déclenche une procédure d'élection en envoyant un message *Election* aux sites de priorité supérieure.**
- **Si le site S_i ne reçoit aucune réponse à son message *Election* après un certain temps, il considère qu'il a la plus haute priorité et s'octroie le statut de coordinateur. Ensuite, il diffuse aux sites de plus faible priorité que lui le message *je suis coordinateur*.**
- **Si le site S_i reçoit une réponse à son message *Election*, il considère qu'un autre site plus prioritaire que lui veut devenir coordinateur et se désiste en sa faveur.**
- **Si un site S_k reçoit un message *Election*, il envoie un message *je suis en service* et reprend la procédure d'élection du coordinateur (sauf s'il a déjà déclenché lui-même cette procédure).**

Algorithme de Silberschatz – Structure en anneau

- **Les sites sont organisés en anneau (logique ou physique)**
- **Chaque site a une priorité fixe.**
- **De manière périodique ou non, chaque site S_i envoie une requête de présence au coordinateur. S'il ne reçoit pas de réponse, le site S_i considère que le coordinateur est hors service et déclenche une procédure d'élection en envoyant un message *Election* (contenant la priorité de S_i) aux sites dans l'ordre de l'anneau.**
- **Chaque site actif qui reçoit le message *Election*, rajoute son numéro de priorité au message et le passe au suivant.**
- **Quand le message *Election* revient à son site source (S_i), celui-ci choisit le site ayant la plus haute priorité contenue dans le message reçu et envoie, sur l'anneau, le message *le nouveau coordinateur est* ... Quand ce message lui revient, après un tour de l'anneau, le site S_i le retire. Tous les autres sites connaissent alors le nouveau coordinateur.**

7. Problème de placement

- **Comment placer les processus et les données pour optimiser un ensemble de critères ?**
- **Critères à optimiser**
  - Equilibrage de charge
  - Minimisation de la communication entre processeurs
  - Distribution équitable des données
  - Temps d'accès aux données
  - Temps de réponse des transactions
  - Tolérance aux fautes
  - Sécurité
  - …
- **Algorithmes de placement :** statique/dynamique, avec ou sans migration, à l'initiative du demandeur/offreur, centralisé/distribué...

  ⇒ PROBLEME d'OPTIMISATION AVEC 1 ou N CRITERES

Exemple d'algorithme de placement

- **Objectif : Placement optimal pour minimiser le temps d'exécution et coût de communication**

- **Principe**

– Construire un graphe tel que :

+ **Les processeurs (P_1, ... P_n) et les tâches/processus (T_1, ...T_m) constituent les sommets du graphe**

+ **Le poids de l'arc entre deux tâches T_i et T_k représente le coût de communication $C_{i,k}$ entre ces deux tâches.**

$C_{i,k}$ = 0 Si les deux tâches sont placées sur le même processeur

+ **Le poids de l'arc entre une tâche T_i et un processeur P_k représente le coût d'exécution de T_i sur P_k , noté E_{iK}.**

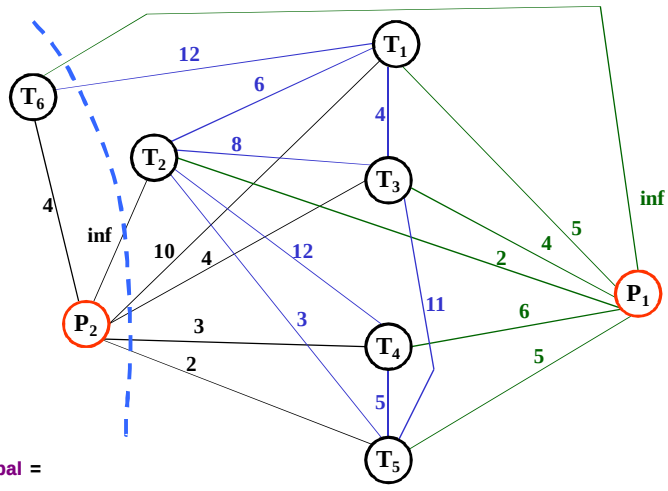
– Recherche de coupe minimale :

+ **C'est un problème de théorie des graphes**

+ **Cas de 2 processeurs : solution polynomiale**

+ **Cas de N >2 processeurs : problème NP-complet**

On appelle **coupe minimale** du graphe associée à un flot, une partition des sommets en 2 ensembles disjoints.



Placement

T_1, T_2, T_3, T_4, T_5 sur P_1

T_6 sur P_2

Coût d'exécution global =

$E_{11} + E_{21} + E_{31} + E_{41} + E_{51} + E_{62} = 26$

Coût de communication global = $C_{12} + C_{13} + C_{16} + C_{23} + C_{25} + C_{35} + C_{24} + C_{45}$ =

$0 + 0 + 12 + 0 + 0 + 0 + 0 + 0 = 12$

Coût global du placement = 26 +12 = 38 (coût minimum)

8. Gestion de l'information distribuée

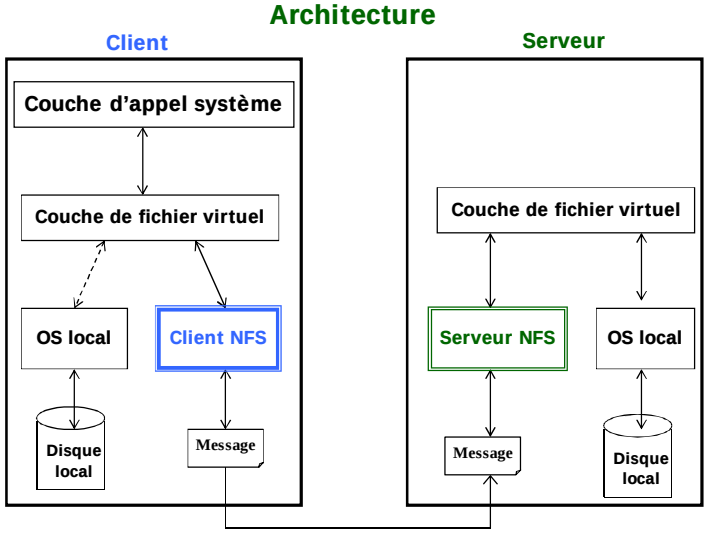
- **Fichiers distribués (NFS, ...)**
- **Bases de données réparties (avec ou sans duplication)**
- **Problèmes à résoudre**
  - Représentation des données (structures de données abstraites)
  - Accès aux données dans un environnement hétérogène
  - Utilisation de caches pour accélérer les accès
  - Copies multiple et leur cohérence
  - Validation des transactions
  - Sécurité et contrôle d'accès (granularité du contrôle)
  - Migration des données
  - Mobilité des données
  - ...

Exemple : NFS (Network File System de Sun) – RFC 3530

- **NFS = système de fichiers distribué.**
- **Idée de base = pouvoir partager entre plusieurs clients et serveurs hétérogènes un même système de fichiers.**
- **Une machine peut être à la fois Client et Serveur.**
- **Un serveur NFS exporte ses répertoires pour qu'ils soient accessibles par des clients.**
- **Si un répertoire est exporté, c'est tout le sous-arbre qui est exporté.**
- **Sous Unix : liste des répertoires exportés dans `/etc/exports`.**

Exemple : NFS (Network File System de Sun)

Architecture

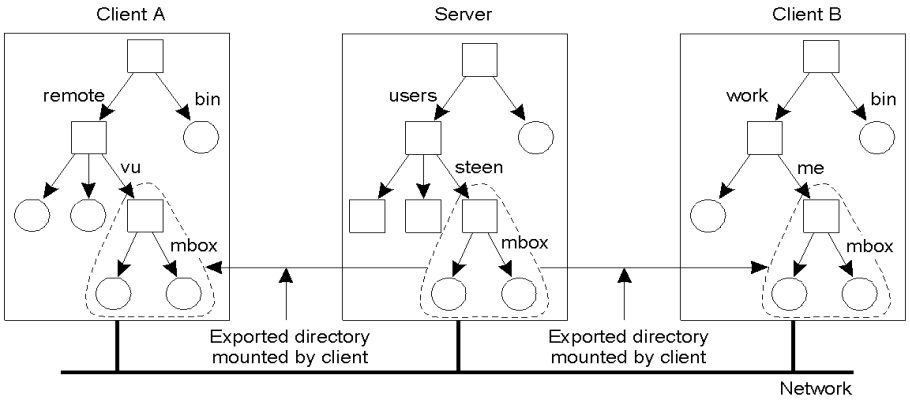


Exemple : NFS (Network File System de Sun)

Architecture

- Un client qui veut accéder à un répertoire distant doit le monter dans sa propre hiérarchie.
- Une station avec un disque local aura une hiérarchie en partie locale et distante.
- Pour les programmes du client pas de différence entre fichiers locaux ou distants.
- Si deux clients ont monté le même répertoire, ils en partagent les fichiers (pb. de verrouillage).
- Deux protocoles : Mounting et Directory

Protocole *Mounting*



Protocole *Mounting*

- **Un client qui veut accéder à un répertoire distant doit le monter dans sa propre hiérarchie.**
- **Le client envoie un chemin d'accès (le nom du répertoire à monter) au serveur et demande la permission de monter le répertoire chez lui.**
- **L'endroit où le client va monter le répertoire n'est pas important pour le serveur.**
- **Si le chemin d'accès est correct et si le répertoire se trouve dans /etc/exports, le serveur renvoie un file *handle* au client. Ce handle contient :**
  - type du système de fichiers
  - n° du disque
  - numéro de i-node du répertoire
  - infos de sécurité (droits d'accès)

Protocole *Directory and Access*

- **Le client envoie des messages pour manipuler des répertoires, lire et écrire des fichiers et leurs attributs (taille, date de modification, propriétaire, ...).**
- **Tous les appels systèmes sont pris en charge par NFS sauf OPEN et CLOSE.**
- **Pour OPEN et CLOSE**
  - **Pour chaque opération READ ou WRITE, le client d'abord envoie une demande LOOKUP qui renvoie un file *handle*, le serveur ne garde pas de trace de cette demande.**
  - **Une opération READ ou WRITE est accompagné du *handle*.**

9. Nommage et localisation

- **Nommage : Comment nommer une entité dans un environnement distribué ?**
- **Formes de noms**
  - Noms destinés à usage humain
  - Chaînes de bits, caractères, de chiffres
  - Noms structurés : adresses IP, groupes de noms,....
  - …
- **Gestion de noms**
  - Une table de correspondance locale
  - Un serveur centralisé de noms (service DS)
  - Serveurs distribué
  - …
- **Mobilité : Nouveaux problèmes à résoudre**

- **Localisation : Comment localiser une entité dans un environnement distribué ?**

- **Méthodes de localisation**
  - Préfixer chaque nom par sa localisation (ne permet pas la mobilité)
  - Broadcast général (demander à tous les sites)
  - Demander aux sites voisins/proches
  - S'adresser à un nœud précis (racine)
  - Utiliser une chaîne de pointeurs de localisation (on contacte d'abord le créateur de l'objet, ensuite on remonte le chemin de migration en passant par les pointeurs).
  - Utilisation de mémoire cache
  - GPS, serveur de localisation
  - ...

10. Synchronisation d'horloges

- **Fournir un temps global approximativement le même pour un ensemble de sites formant un "groupe" , malgré:**
  - les dérives des horloges physiques des sites
  - les défaillances des sites et de leurs horloges
  - les défaillances du réseau de communication

- Le temps physique fourni par une horloge synchronisée (HS) doit être une bonne approximativement du temps réel.

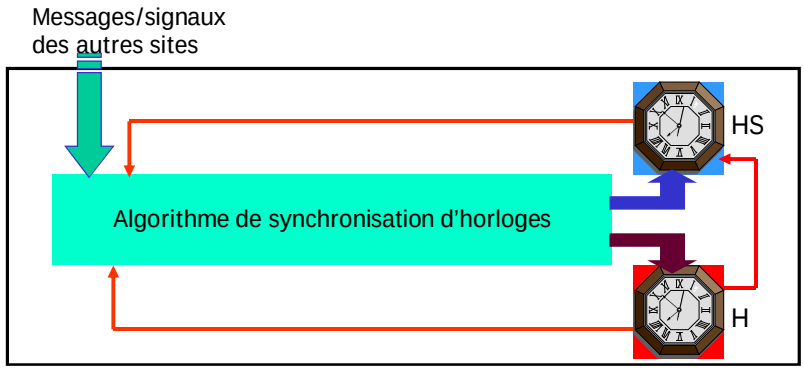
- **Propriété de précision d'accord (PA) des horloges synchronisées**

$$\left|HS_i(t) - HS_j(t)\right| \leq PA \qquad 1 \leq i \leq n, \quad 1 \leq j \leq n$$

- **Propriété d'exactitude (Ex) des horloges synchronisées**

$$\exists Ex << 1, \qquad (1 - Ex) \leq \frac{HS_i(t2) - HS_i(t1)}{t2 - t1} \leq (1 + Ex) \qquad t2 \geq t1 + IM$$

Principe de base de synchronisation d'horloges



Exemple 1 : Algorithme CNV de Lamport (« Interactive CoNVergence algorithm »)

- **Algorithme décentralisé**
- **Principe de l'algorithme** (pour tout site *i*)

```
Toutes les V unités de temps faire
  MonHorloge = HSi
 /* Lecture des horloges distantes */
      Pour k=1 à n et k≠i répéter
        HSk = Lire_Horloge(k)
        si | HSk - MonHorloge | > Seuil
             Alors HSk = MonHorloge
      Fin
 /* Calcul de la référence */
  R = moyenne(HS1, HS2, …, HSn)
 /* Ajustement */
  HSi = R
Fin
```

Exemple 2 : Algorithme Gusella et Zatti

- **Principe de l'algorithme pour le site privilégié** (noté site *p*)

```
Toutes les V unités de temps faire  /* Lecture des horloges distantes */
    Pour k=1 à n et k≠p répéter
        envoi de la demande de lecture au site k ; t1,k = Hp(t)
        réception de la valeur de Hk ; t3,k = Hp(t)
    Fin
    Sélectionner les valeurs d'horloges qui diffèrent les
    unes des autres d'au maximum Z  /* Z est un seuil fixé */
    somme = 0 ; NbHcorrectes = 0
    Pour k dans l'ensemble des horloges sélectionnées répéter
        Δp,k = ((t3,k + t1,k )/2) - Hk ;  /* Estimation de la différence */
        somme = somme + Δp,k ; NbHcorrectes = NbHcorrectes + 1
    Fin
    R = somme/NbHcorrectes  /* Calcul de la référence */
    Pour k=1 à n et k≠p répéter
        Ak = R - Δp,k      /* Calcul de la donnée d'ajustement du sit k */
        envoi de Ak au site k
    Fin
    Ap = R  /* Donnée d'ajustement du site privilégié */
Fin
```


11. Sécurité

- **Sécurité : un besoin pour tous**
- **Formes des attaques**
  - Passives
  - Actives (virus, déni de service, destruction de données…)
- **Mécanismes de la sécurité**
  - Identification et authentification
  - Contrôle d'accès (granularité de protection des données)
  - Signature digitale
  - Chiffrement (cryptographie)
    - + Utilisation de clés
    - + Puissance de chiffrement
    - + Politique de distribution des clés
    - + …