Applications réparties : principes et propriétés (1)

Tarek Melliti
IBISC (Informatique, Biologie Intégrative et Systèmes Complexes)
LIS (Langage Interaction et Simulation)
tarek.melliti@ibisc.fr

Plan du cours

- **Introduction**
  - définitions
  - problématiques
  - architectures de distribution
- **Distribution intra-applications**
  - notion de processus
  - programmation multi-thread
- **Distribution inter-applications et inter-machines**
  - sockets
  - middlewares par appel de procédures distantes
  - middlewares par objets distribués (Java RMI, CORBA)
- **Conclusion**

Plan du cours

- **Introduction**
  - définitions
  - problématiques
  - architectures de distribution
- **Distribution intra-applications**
  - notion de processus
  - programmation multi-thread
- **Distribution inter-applications et inter-machines**
  - sockets
  - middlewares par appel de procédures distantes
  - middlewares par objets distribués (Java RMI, CORBA)
- **Conclusion**

Plusieurs exemples pour commencer

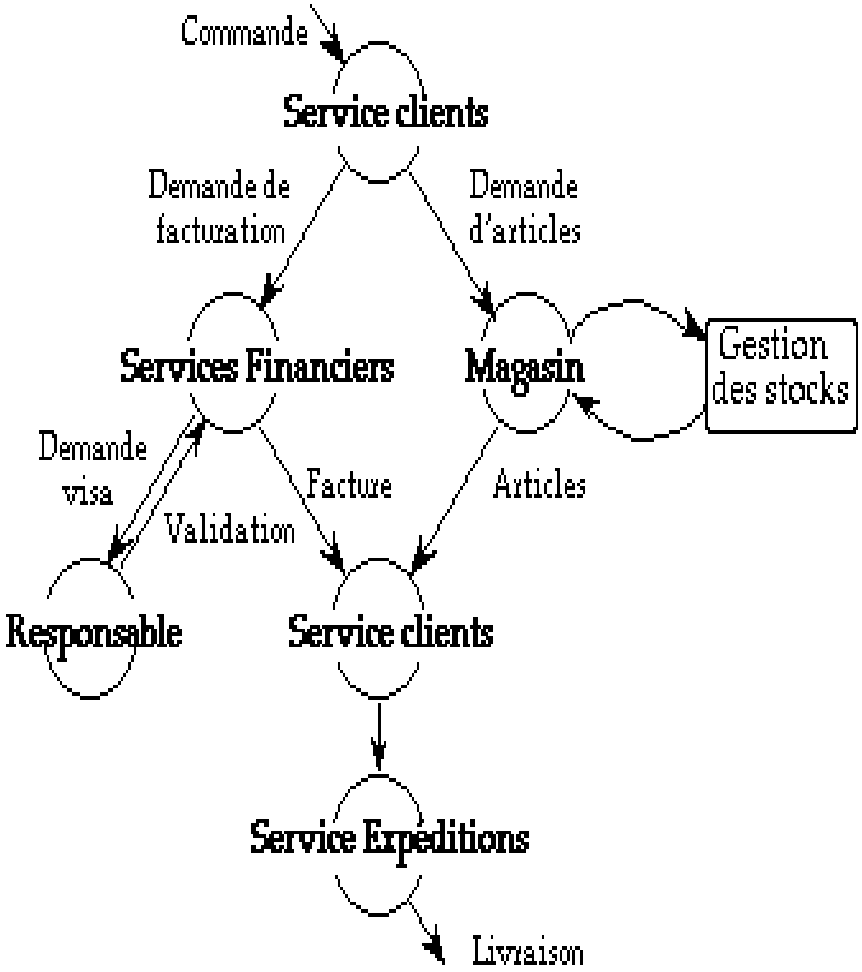
- **Coordination d'activités:**
  - Systèmes à flots de données («WorkFlow »)
- **Communication et partage d'information**
  - Bibliothèques Virtuelles
- **Collecticiels:**
  - édition coopérative
  - Téléconférence
  - Ingénierie Concourante (PSA)
- **Nouveau services grand public**
  - Presse électronique
  - Télévision interactive
  - Commerce électronique

Exemple 1 : Flot de données (*WorkFlow*)

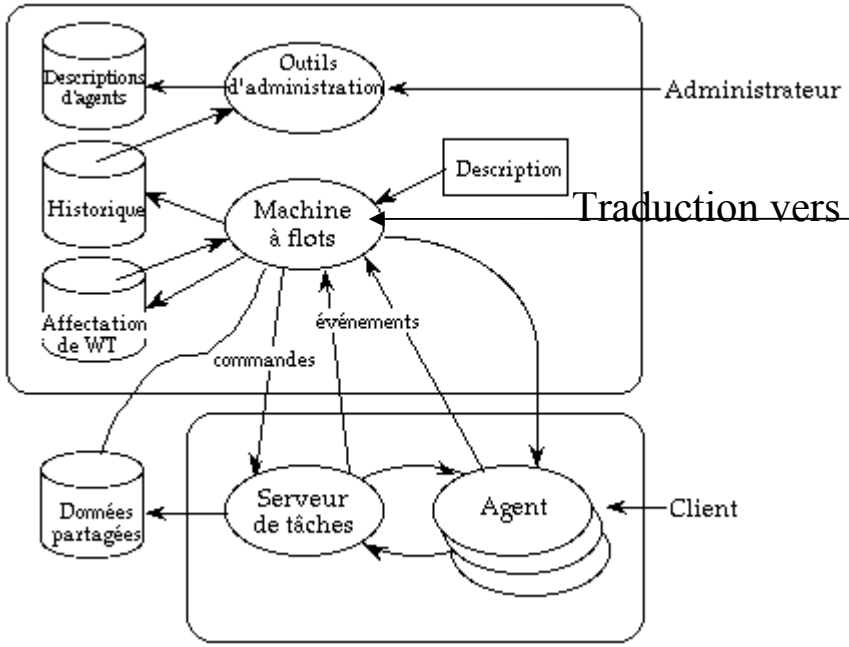
- **Une classe d'applications caractérisées par l'exécution coordonnée d'un ensemble de tâches sur différents organes de traitement.**
- **La coordination :**
  - spécification du séquencement des tâches
  - Spécification des données échangées entre les tâches
- **Exemple l'administration**
  - Service => tâche => nécessite des documents
  - Procédure administratif suite de tâches entre services et des documents échangées
- **Aspect important dans tel type d'application**
  - Systèmes à rôle
    - Dans un services des agents (fonctionnaire) sont assignées à des tâches
    - Dynamique ou statique
  - Systèmes réactifs
    - Souvent événementiel => l'arrivé d'un « document » déclenche la tâche
    - Utilisation des événements

Traitement d'une commande

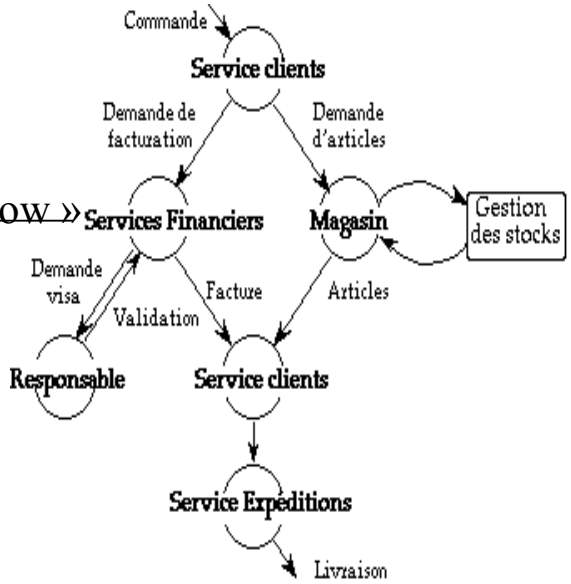
- **Une commande reçue est traitée par le Service Clients**
- **transmet aux Services Financiers une demande de facturation**
- **et au Magasin une demande de fournitures**
- **.....**



Architecture des systèmes à flot de données



Traduction vers un langage de « workflow »



- **Gestion centralisé de la coordination**
  - Conséquence peu de tolérance au panne
- **Nécessite un mécanisme d'allocation de tâches**
  - Dynamique et/ou statique
- **Les serveur de tâche peuvent être répartie**

Quelque remarques

- **Parmi les nombreux problèmes posés par les applications à flots de données, nous ne retenons que ceux qui concernent directement le support système et plus particulièrement la répartition.**
- **Coordination des tâches => nécessite des mécanismes de synchronisation et de communication.**
- **Gestion de groupes. L'association entre tâches et agents n'est en général pas définie à l'avance.**
- **Adaptation de la gestion des tâches à l'environnement.**
  - Centralisé;
  - Grappe de machine.
  - réseau local;
  - réseau à grande distance.

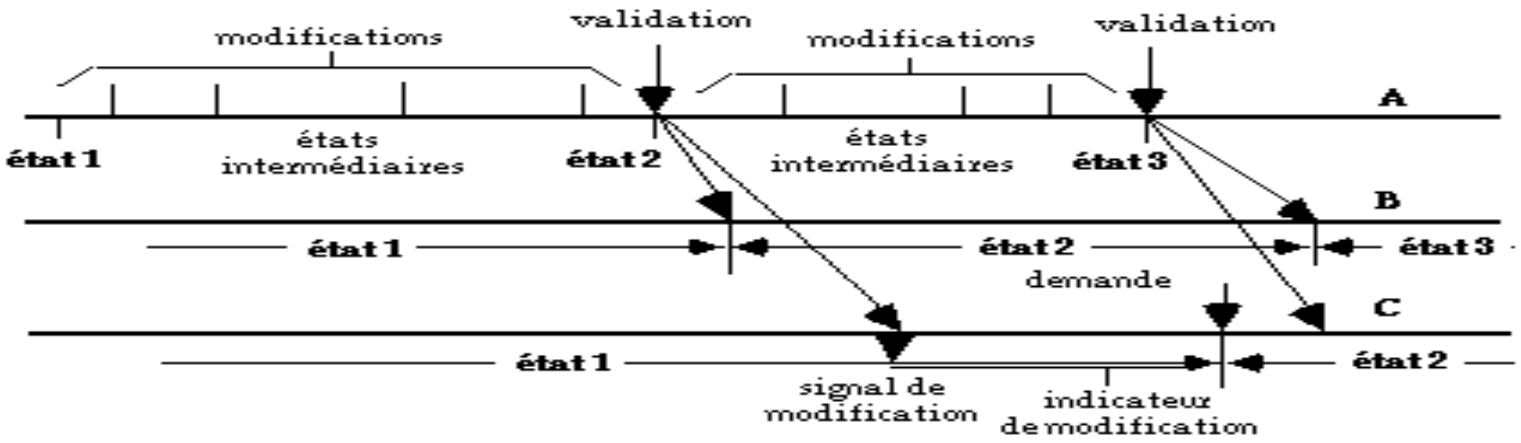
Exemple 2 : Les collecticiels (« Groupware »).

- **Logiciel pour travail coopératif entre plusieurs utilisateurs géographiquement réparties.**
  - Tâches commune
  - Ressource peuvent être communes
  - Les actions des autres peuvent être soit :
    - Immédiatement perceptible : synchrone.
    - Différé : asynchrone
- **Les exemples Multiple:**
  - Rédaction coopérative de document : asynchrone
  - Téléconférence : synchrones
  - Ingénierie coopérative : les deux quand c'est un objet virtuel
    - Conception coopératif d'une voiture (exemple PSA)
  - Télé-operation :
    - Les missions spatial.

Exemple 2 : rédaction coopératif de document(1)

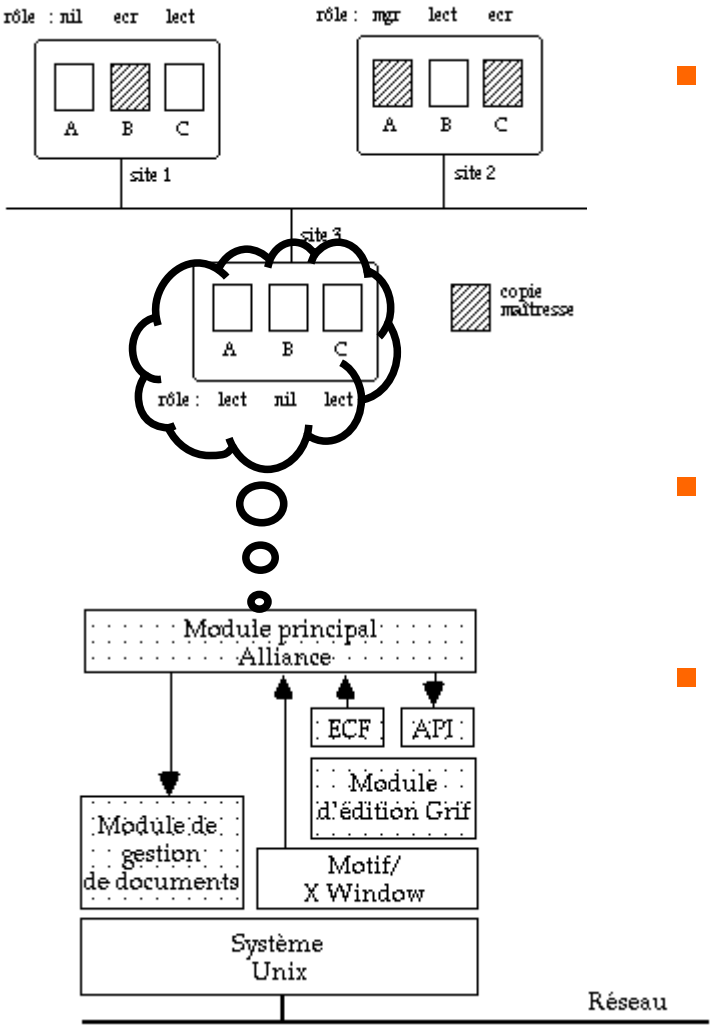
- **Un Document XML par exemple**
- **Comporte des parties (intro, chap1... concl)**
- **Un ensemble d'auteur doivent coopérer pour sa rédaction (un livre, un journal etc.)**
- **Le document est un objet abstrait.**
- **Chaque utilisateur a une vue différent:**
  - les parties (ou fragments) du document qui sont rendues visibles à l'utilisateur,
  - les droits de l'utilisateur sur chacun de ces fragments (lecture, écriture, etc.),
  - le degré de "fraîcheur" de chaque fragment (prise en compte des dernières modifications).

Exemple 3 : rédaction coopératif de document(1)



- **A Chaque partie du document correspond une copie maîtresse.**
  - L'administrateur du bout du document decide
- **Localement un auteur voie ce qui a le droit de voir**
- **Ce qu'il voit dépend :**
  - De la validation des modifications par le propriétaire du fragment
  - De son envie de le voir la dernière état du fragment
  - Voir exemple A valide les changement
    - B demande de voir l'état actuel mais pas C.

Architecture des collecticiels



- **Le module d'édition qui est accessible à travers deux interfaces.**
  - API (Application Programming Interface), permet d'appeler les fonctions d'édition ;
  - ECF (External Call Facility, répercute les événements significatifs (modifications du document via l'interface graphique).
- **Le module de gestion de documents**
  - contrôle la désignation, la distribution physique et le stockage des documents...
- **Le module principal, qui pilote l'ensemble de l'application.**

Quelques remarques

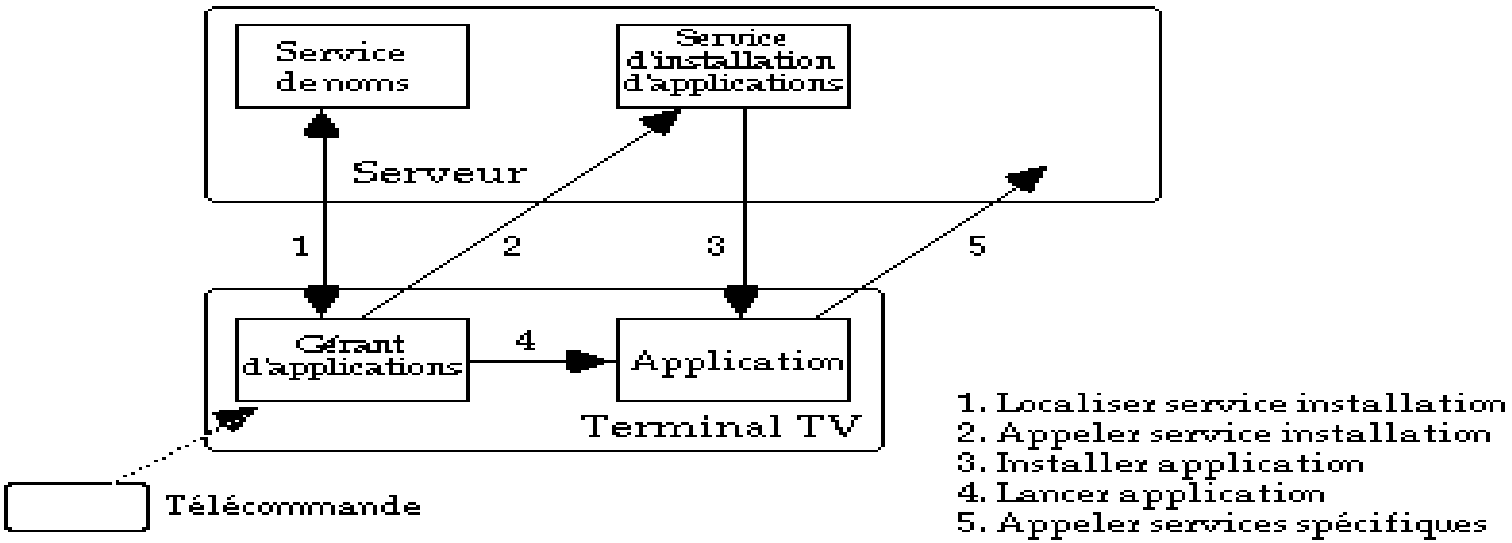
- **Cohérence de l'information**
  - Laisser à la charge des utilisateur (gestion lâche)
- **Structure du système**
  - Chaque utilisateur possède un exemplaire du document
  - Conséquence, action d'édition local : communication seulement pour la coordination
  - On est dans un cas de collecticiel Asynchrone
- **Cas Synchrone : e.g Vidéo conférence**
  - Tableau un objet de coopération où la cohérence doit être synchrone
  - Plus de communication pour assurer la cohérence
  - Données de nature Multimédia (quantité importante)
  - Problème pour assurer la qualité de services (Qos)
  - Nécessité d'un processus local pour chaque intervenant
  - La gestion de l'interaction synchrone se fait par interception local des événement et notification a des processus « représentants »

Exemple 3 : Télévision interactive

- **Systèmes de plus en plus présents**
  - Services TV de *Free* ou *wanadoo* ou autre....
  - Programme à la demande
  - Télé Achat
  - Jeux interactif
  - Pour cela il suffit d'un poste de télévision + un abonnement au réseau.
- **Il s'agit d'une application « Grand public »**
  - Le réseau doit être extensible
  - Et quelque soit le nombre des abonnées
  - La qualité de services doit rester inchangé
  - Ce qu'on appel une application qui supporte le Passage à l'echelle

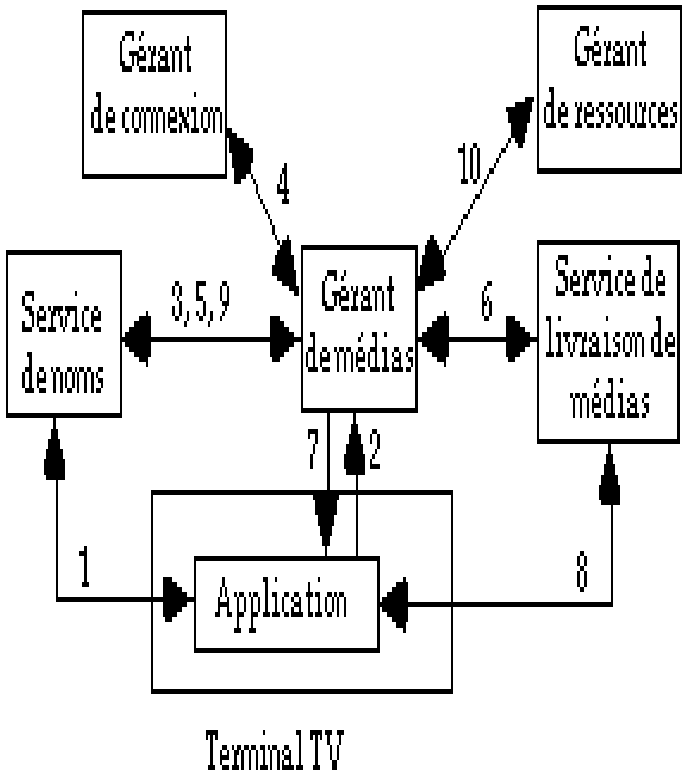
Architecture d'un système de TV interactif (1)

- **Organisation générale orientée Objet**
  - Interface un ensemble de méthode ou procédure
  - Chaque méthode est réalisé concrètement par un services implémenter sur un serveur.
  - Chaque objet est accessible par une référence
  - Un services de nom est donc par défaut nécessaire pour trouver les références d'un objet à partir d'un nom symbolique
  - La référence du services de nom est connue a l'initialisation du système.



Architecture d'un système de TV interactif (2)

1) la référence du gérant de médias (GM)
2) demande à GM un film
3) Le GM obtient la référence du gérant de connexion
4) lui demande d'ouvrir une connexion
5) Le gérant des médias obtient la référence du service de livraison des média (SLM)
6) lui demande la référence de l'objet représentant le film
7) transmet la référence objet film au terminal TV
8) Le Terminal TV appelle la méthode "jouer" sur l'objet filme, ce qui provoque la transmission des images depuis le SLM
9) Le GM obtient la référence du gérant de ressources
10) lui demande périodiquement l'état du terminal TV pour pouvoir réagir en cas de panne

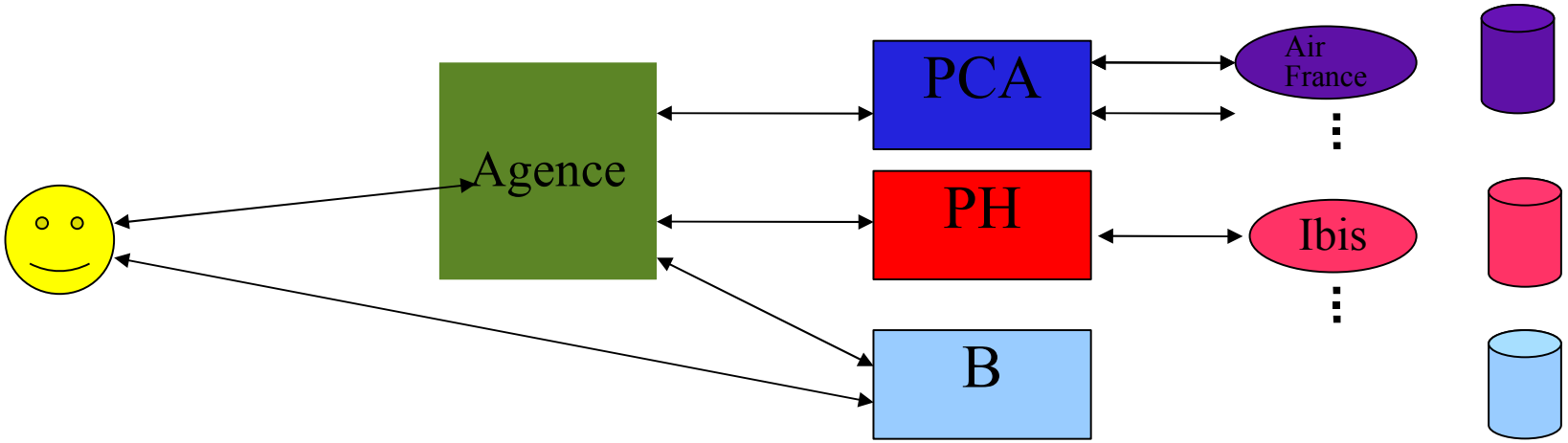


Quelque remarques

- **La disponibilité est une exigence essentielle pour ce système |**
- **généralement les services sont dupliqués sur plusieurs serveurs.**
  - Le service de noms étant central est le plus concerner par la duplication
  - Ça marche généralement suivant un schéma maître-esclaves
  - => mécanisme de sélection du serveur (critère géographique,...).
- **Ce système est représentatif des nouvelles applications destinées au grand public.**
  - la disponibilité,
  - les performances d'accès,
  - et la simplicité de l'interface.
- **Il reste à montrer que le système conserve ses qualités lors du passage à une plus grande échelle.**

Un dernier pour la route : commerce Electronique

- **Business to Consumer (B2C) : rapprocher le fournisseur du client**
- **Business to Business (B2B) : rapprocher les partenaires pour réaliser un plus value.**
- **Cas typique agence de voyage**
  - Des compagnies aérienne (Transport)
  - Des Hôtels (logement)
  - Services de location de voiture
  - Les services d'activité touristiques
  - Banques (services de payement)
  - Etc.



Particularité du commerce électronique

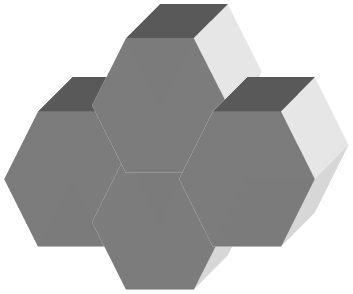
- **Protection des informations confidentielles, aussi bien pour le client (numéro de carte de crédit, etc.) que pour le fournisseur (conditions particulières de vente, etc.),**
- **Respect des règles de concurrence,**
- **Respect des garanties données au client par le fournisseur (sincérité de l'offre, exécution du contrat de vente, etc.),**
- **Respect des garanties données au fournisseur par le client (identité, paiement, etc.),**
- **Respect des droits de propriété (licences, droits d'auteur, etc.).**

Quelques remarques : la sécurité

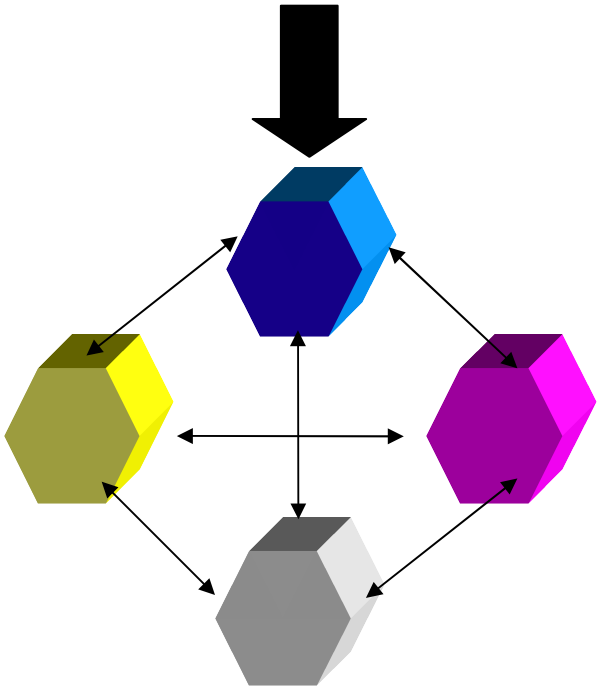
- **Cette application met en évidence des problèmes de sécurité et de tolérance aux fautes**
- **Sécurité:**
  - la confidentialité
    - assurer qu'une information n'est accessible qu'aux entités autorisées à la consulter
  - L'intégrité
    - Une information doit rester identique à elle-même si elle n'est pas explicitement modifiée (exp les message envoyé est reçu au même état).
  - L'authentification
    - garantit qu'une entité (personne, mais aussi programme s'exécutant pour le compte d'une personne, etc) est bien celle qu'elle prétend être (cryptographies, clés public)
  - Il est ainsi possible de garantir:
    - le destinataire ne peut nier avoir reçu le message,
    - l'émetteur ne peut nier avoir envoyé le message,
    - l'émetteur ne peut pas prétendre avoir envoyé, et le destinataire ne peut pas prétendre avoir reçu, un message différent de celui qui a été transmis.
    - l'identité de l'émetteur est établie et ne peut être contrefaite
- **Tolérance aux fautes**
  - Transaction : le systèmes doit garantir que le payement et la livraison du produit doivent être réalisés ensemble ou rien

Récapitulons : qu'est ce qu'une application répartie?

- Le système est constitué d'un ensemble de composants matériels (ordinateurs, organes d'entrée-sortie, processeurs spécialisés, dispositifs de commande ou de mesure, etc.), interconnectés par un réseau de communication.
- Les composants du système ne fonctionnent pas de manière indépendante, mais coopèrent pour l'exécution de tâches communes.
- Le système peut continuer à fonctionner (éventuellement en mode dégradé) malgré les défaillances partielles des composants ou du réseau de communication.

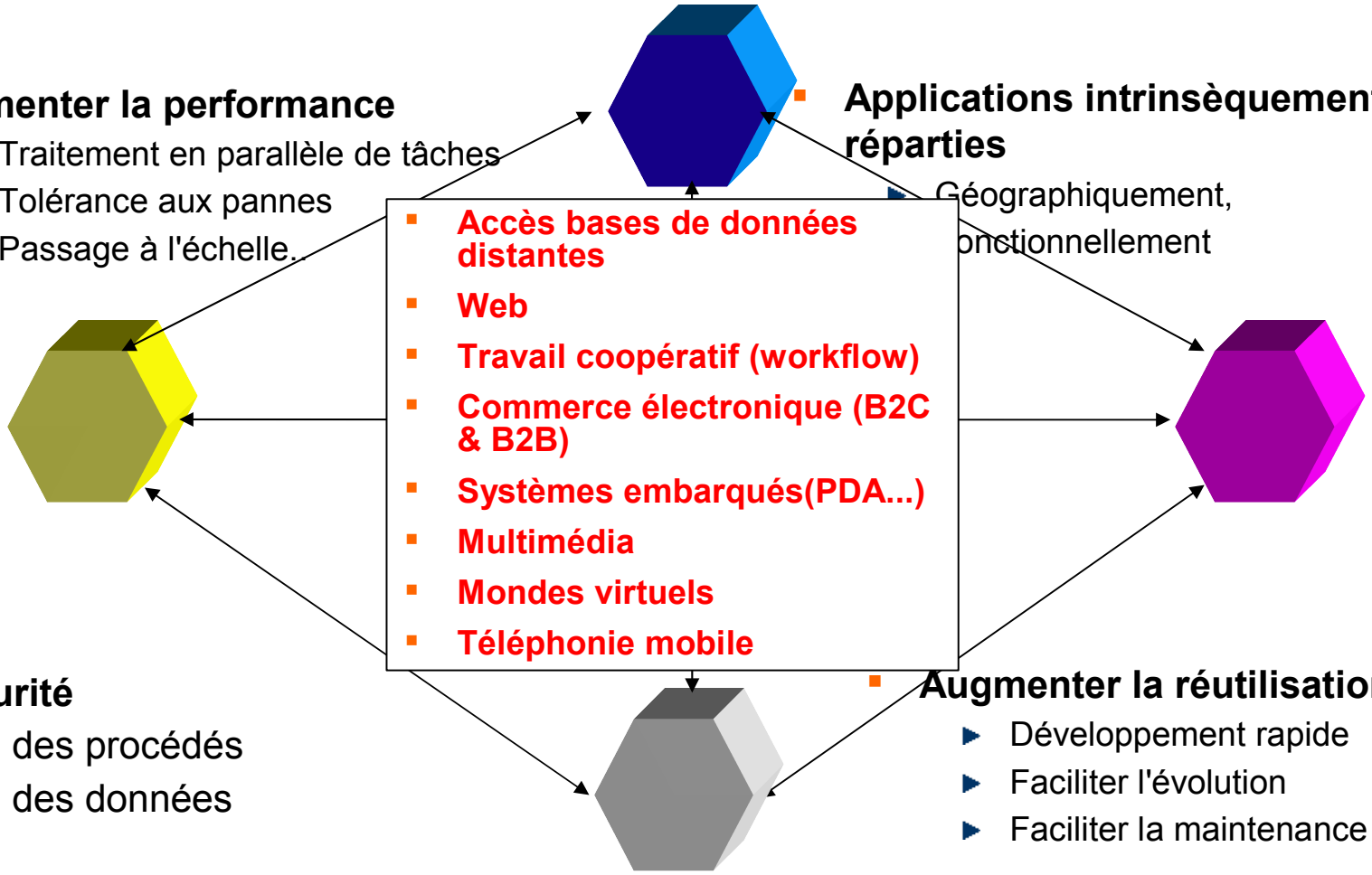


Application monolithique



Récapitulons : qu'est ce qui nous pousse à distribuer?

- **Augmenter la performance**
  - Traitement en parallèle de tâches
  - Tolérance aux pannes
  - Passage à l'échelle…
- **Applications intrinsèquement réparties**
  - Géographiquement,
  - onctionnellement
- **Sécurité**
  - des procédés
  - des données
- **Augmenter la réutilisation**
  - Développement rapide
  - Faciliter l'évolution
  - Faciliter la maintenance

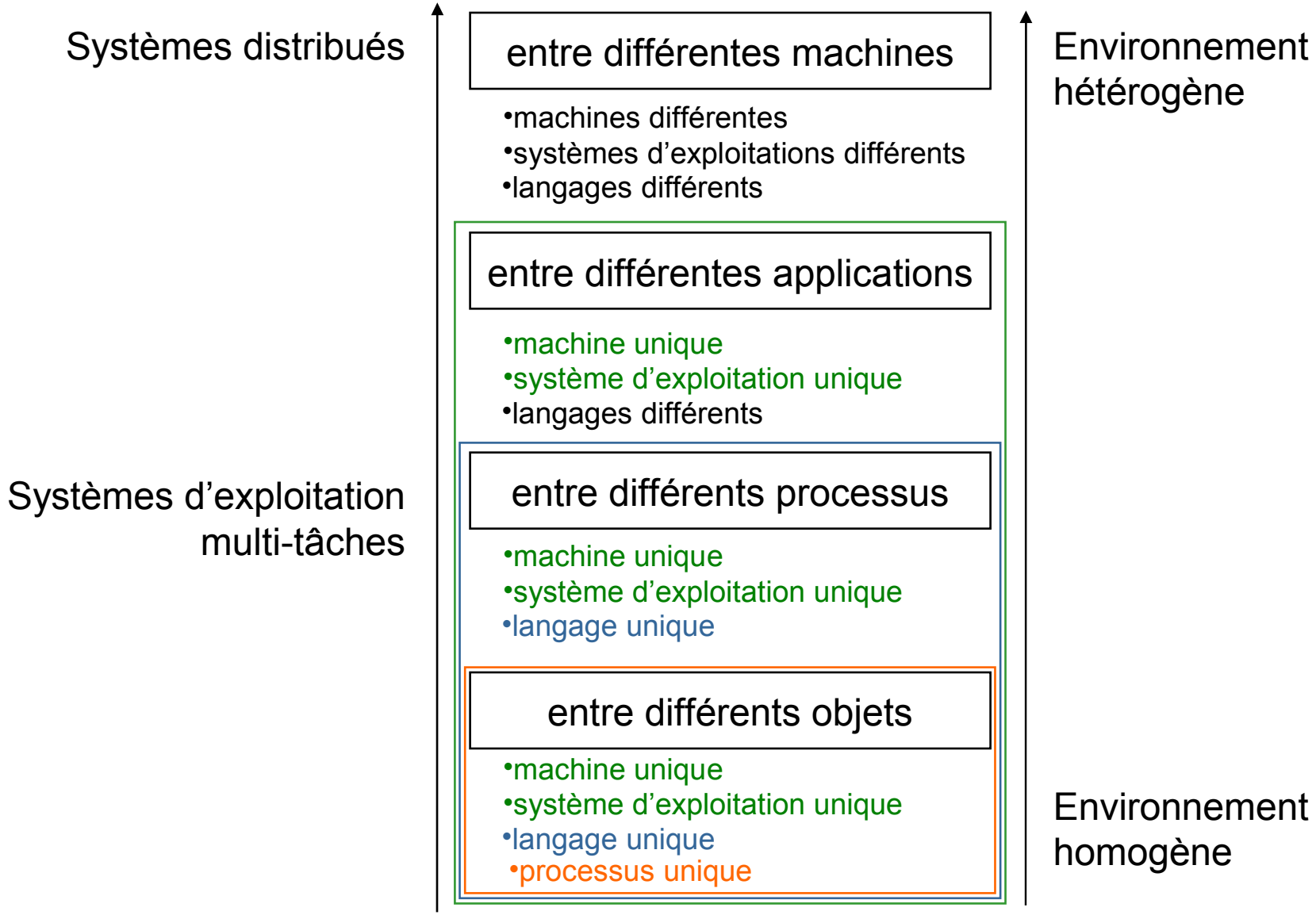


Application répartie

Différentes formes de distribution

- **Distribution physique**
  - fonctionnement en réseaux (LANs, WANs)
  - architectures multi-processeurs
- **Distribution structurelle**
  - programmation structurée
  - programmation objet
  - programmation par composants
  - programmation par aspects
- **Distribution fonctionnelle**
  - décomposition en services indépendants (affichage, calcul, stockage de données, etc.)
  - services Web

Différents niveaux de distribution



Définition et caractéristiques

- **Nombreuses définitions possibles**
- **Définition retenue**
  - *Un système distribué est une collection de processus ou d'ordinateurs* ***indépendants*** *et* ***coopératifs*** *qui apparaissent à l'utilisateur comme un seul et* ***unique*** *système* ***cohérent***
  - contraire = *système monolithique*

Caractéristiques des application répartie

- **Hétérogénéité**
- **Ouverture**
- **Sécurité**
- **« Scalabilité »**
- **Tolérance aux pannes**
- **Concurrence**
- **Transparence pour l'utilisateur**

Hétérogénéité

- **au niveau des réseaux**
  - réseaux de différents types
- **au niveau du matériel informatique**
  - codage des données différent suivants les architectures
- **au niveau des systèmes d'exploitation**
  - protocoles standards mais APIs différentes d'un OS à l'autre
- **au niveau des langages de programmation**
  - différences pour le codage des caractères ou les structures de données
- **au niveau des implantations**
  - nécessité de suivre les standards établis

Hétérogénéité – approches possibles

- **protocoles standards de communication**
  - masquent l'hétérogénéite des systèmes
  - ex. : standards pour l'Internet TCP/IP
- **utilisation de middlewares**
  - modèles de programmation distribuée standardisés et uniformes accès aux BDs, appel d'objets à distance
  - implémentés au-dessus des standards de l'Internet
  - ex. : CORBA, Java RMI, Microsoft COM/DCOM, etc.
- **utilisation de *code mobile***
  - envoyé d'un ordinateur à l'autre
  - exécuté sur l'ordinateur distant
  - ex. : applets Java, agents mobiles

Ouverture

- **Possibilité d'extension ou de ré-implantation**
  - ajout et mise à disponibilité de ressources partagées
  - accès à d'autres applications
  - accès depuis d'autres applications
  - nécessite le libre accès aux spécifications et à la documentation des APIs pour les programmeurs
    - ex. : RFC pour Internet
- **Avantages**
  - développement collaboratif
  - permet une indépendance vis-à-vis des constructeurs et éditeurs de logiciels

Sécurité

Capacité à assurer la sécurité des données qui transitent, l'authentification des participants et à empêcher les intrusions

- **Sécurité des informations**
  - Confidentialité
    - protection contre les destinataires indésirables
  - Intégrité
    - protection contre l'altération ou la corruption des données
  - Authentification, signature électronique
    - identification des partenaires
    - non-déni d'envoi ou de réception
    - messages authentifiés
    - respect possible de l'anonymat
  - Disponibilité
    - protection contre les interférences aux moyens d'accès
- **Risques liés à la sécurité**
  - pénétration d'un Intranet via un firewall
  - envoi sécurisé de messages systèmes de cryptage
  - déni de service
  - sécurité des codes mobiles

Scalabilité

Capacité à rester efficace en cas de forte augmentation des ressources et des utilisateurs

- **Adapter le dimensionnement de l'application**
  - ajout/duplication de composants
    - ex. : augmentation du nb de clients duplication des serveurs
  - problème = coût de communication/synchronisation lié à l'augmentation du nb de processus
- **Implique la prise en compte**
  - du contrôle du coût des ressources physiques
    - idéalement, pour *n* utilisateurs, ressources en O(n)
  - du contrôle de la perte de performance
    - idéalement, si volume de données proportionnel au nb d'utilisateurs ou de ressources, temps d'accès < O(log n)
  - de la prévention de la diminution des ressources logicielles
    - ex. : nb d'adresses Internet limité par la représentation sur 32 bits
  - de la prévention des goulots d'étranglement
    - décentralisation nécessaire

Tolérance aux pannes

Capacité de l'application à s'exécuter en mode dégradé

- **Les pannes**
  - généralement partielles et d'origines diverses
    - panne d'un composant
    - panne d'un lien de communication entre composants
  - pb = limiter les conséquences liées à la panne d'un système matériel (ou logiciel)
    - éviter une trop forte centralisation
    - éviter d'isoler des fonctions vitales
- **Indicateur de performance = disponibilité**
  - proportion de temps pendant laquelle il est disponible pour utilisation
  - mesurée en pourcentage
  - disponibilité de 99,999% pour les systèmes les plus fiables

Tolérance aux pannes – approches possibles

- **la détection de pannes**
  - but = détecter que les données reçues ne sont pas celles attendues
  - ex. : bit de contrôle pour l'envoi de données
- **le masquage des pannes**
  - but = limiter l'impact des pannes
  - ex. : retransmission de messages, systèmes de cache
- **la tolérance aux pannes**
  - alerter si panne trop importante pour être corrigée
  - ex. : serveur Web en panne
- **la reprise sur erreur**
  - rétablir les données suite à une panne
  - ex. : systèmes de sauvegardes des données permanentes
- **la redondance**
  - duplication de composants pour en avoir toujours au moins un de disponible
  - ex. : plusieurs chemins entre routeurs, plusieurs BDs

Concurrence

Capacité de plusieurs processus à s'exécuter en parallèle

- **Concurrence**
  - accès simultané de plusieurs utilisateurs à la même ressource matérielle ou logicielle
  - ex. : base de données, serveur Web, etc.
- **Approches possibles**
  - redondance
    - ex. : duplication d'une base de données
    - pb de synchronisation entre les ressources
  - gestion des ressources par le système d'exploitation
    - ex. : accès à un fichier partagé
    - pb de synchronisation entre processus utilisation de sémaphores, mécanismes de synchronisation de threads
    - synchronisation trop forte = risque de « sérialisation »

Transparence (1)

Séparation des composants dans un système distribué de manière à ce que l'utilisateur perçoive ce système comme un tout plutôt que comme une interconnexion de composants actifs

- **caractère intégré de l'application, qui permet d'en cacher la complexité à l'utilisateur**
  - ressources
  - moyens de communication
  - architecture interne organisationnelle
- **Le standard ISO identifie huit types de transparences**
  - La transparence d'accès
    - les ressources locales et distantes doivent pouvoir être accessibles de la même manière
    - ex. : système de montage de volumes Unix
  - La transparence de localisation
    - les ressources doivent être accessibles quelle que soit leur localisation physique
    - ex. : URL Web

Transparence (2)

- La transparence de concurrence
  - plusieurs processus doivent pouvoir opérer de manière concurrentielle sans interférences entre eux
- La transparence de réplication
  - plusieurs instances des ressources doivent être déployées pour assurer la fiabilité du système
- La transparence de panne
  - une panne ne doit pas bloquer le fonctionnement global du système
- La transparence de mobilité
  - les ressources et clients doivent pouvoir être mobiles sans affecter le fonctionnement global
- La transparence de performance
  - le système doit pouvoir être reconfigurable pour assurer les montées en charge
- La transparence d'échelle
  - le système et les applications doivent pouvoir supporter les changements d'échelles sans modification interne des algorithmes par exemple

Applications réparties : *Mais où est le problème?*

- **Difficultés**
  - Propriété d'asynchronisme du système de communication (pas de borne supérieure stricte pour le temps de transmission d'un message)
    - Conséquence : difficulté pour détecter les défaillances
  - Dynamisme (la composition du système change en permanence)
    - Conséquences : difficulté pour définir un état global
    - Difficulté pour administrer le système
  - Grande taille (nombre de composants, d'utilisateurs, dispersion géographique)
    - Conséquence : la capacité de croissance (scalability) est une propriété importante, mais difficile à réaliser

Malgré ces difficultés, des grands systèmes répartis existent et sont largement utilisés le DNS (Domain Name System) (service de base) le World Wide Web (infrastructure)

Voies d'études des sytèmes répaties

- **Approche “descriptive”**
  - Étude des divers modèles de conception et de construction d’applications répartis (client-serveur, événements et messages, objets répartis, composants répartis, etc.)
  - Étude des diverses classes de systèmes, intergiciels et applications, et de leurs modes d’organisation et de fonctionnement
    - Modèles d'architecture
    - Modèles de communication
  - **C’est l’objet du cours “Construction d’applications réparties et parallèles” (CR)**
- **Approche “fondamentale”**
  - Étude des principes de base des systèmes répartis ; les problèmes fondamentaux (et leur origine), les solutions connues, les limites “intrinsèques”
  - Application de ces principes à quelques situations concrètes
  - **C’est l’objet du présent cours**

1. Approche Fondamentales : problématique des systèmes réparties

Aperçus sur les problèmes fondamentaux

- Comment décrire une exécution répartie ?
- Comment déterminer des propriétés globales à partir d'observations locales ?
- Comment coordonner des opérations en l'absence d'horloge commune ?
- Comment partager des données en l'absence de mémoire commune ?
- Quels sont les critères de qualité pour une application répartie ?
- Y a-t-il des problèmes intrinsèquement insolubles ? Et, dans une telle situation, que fait-on en pratique ?
- Comment définir, et maintenir, la cohérence d'informations réparties ?
- Comment garder un système en fonctionnement malgré des défaillances partielles ?

Voie d'approche pour les problèmes « fondamentaux »

- **Définir des modèles**
  - Un modèle est une représentation d'une partie du monde réel
  - Un modèle représente des éléments réels par des éléments abstraits (qui ignorent ou simplifient divers aspects du réel)
  - Un même système réel peut être représenté par des modèles différents
    - **selon le problème auquel on s'intéresse**
    - **selon le degré de détail souhaité**
  - Tout modèle a des limites (pour son adéquation à la réalité), qu'il ne faut pas oublier
- **Utiliser les modèles**
  - Pour observer et comprendre le comportement du système réel
  - Pour prédire le comportement du système réel dans certaines circonstances
  - Pour aider à commander le système réel (lui imposer un comportement souhaité)

Propriétés des systèmes et le cas répartie

- **On veut être capable de raisonner sur un système ou une application**
  - Définir des prédicats, ce qui implique d'accéder à un état
  - Coordonner des activités, ce qui implique de définir un ordre
- **Pourquoi est-ce différent (et plus difficile) en univers réparti ?**
  - Pas de mémoire commune (support habituel de l!état)
  - Pas d'horloge commune (qui définit le séquencement des événements)
  - Asynchronisme des communications
    - Pas de borne supérieure sur le temps de transit d'un message
    - Conséquence du mode de fonctionnement de la plupart des réseaux (dont l'Internet)
  - Asynchronisme des traitements
    - Pas de bornes sur le rapport relatif des vitesses d'exécution sur deux sites
    - Conséquence de la variabilité de la charge et de l'absence (en général) de garanties sur l'allocation des ressources

Sûreté et Vivacité

- **On est souvent amené à spécifier et à vérifier des propriétés d!un système dynamique (évoluant dans le temps)**
- **Ces propriétés sont en général classées sous deux rubriques:**
  - **Sûreté (safety)** : un événement indésirable n'arrivera jamais
    - Exemples : violation de l'exclusion mutuelle
    - incohérence dans les données
  - **Vivacité (liveness)** : un événement désirable finira par arriver
    - Exemples : un message sera délivré à son destinataire
    - une ressource demandée sera rendue disponible
    - un algorithme se termine (indécidable dans le cas général…)
- **Il est souvent impossible de fixer une borne supérieure à l'attente.**
- **=> Les propriétés de vivacité sont plus difficiles à établir que celles de sûreté**

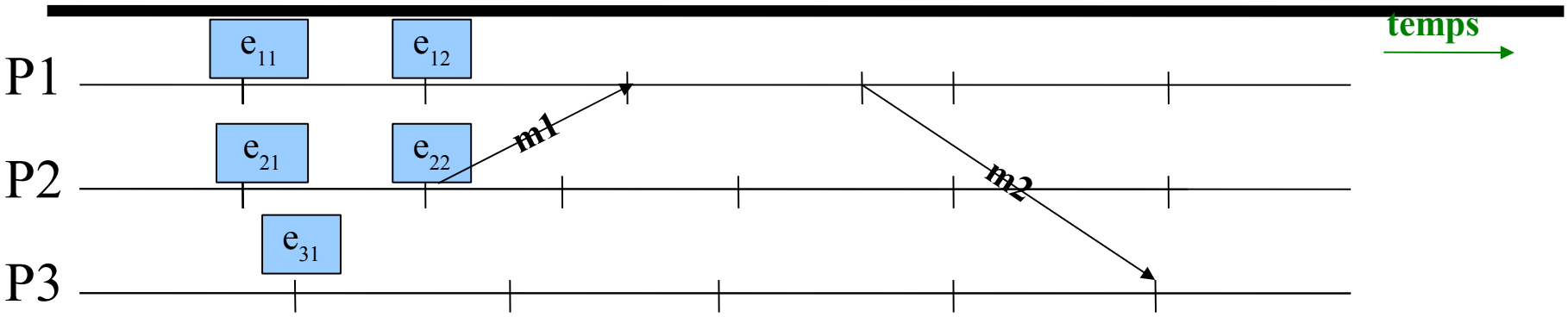
Un mot sur les messages

- **On suppose disponible un système de communication permettant d'envoyer des messages entre processus**
- **Propriétés :**
  - Un message finit par arriver à destination, mais son temps de transmission n'est pas borné; on modélise ainsi une panne (détectée) suivie d'une ou plusieurs réémission(s)
  - Un message arrive intact (non modifié) ; on suppose que des mécanismes de détection / correction d'erreur sont utilisés
  - Selon le cas, on fera ou non l'hypothèse que le canal est FIFO entre deux processus
- **Bien distinguer la réception d'un message de sa délivrance à son destinataire!**

Le modèle asynchrone (fiable)

- **C'est une tentative pour modéliser certains aspects du monde réel**
  - **Asynchronisme des communications et des traitements**
- **C'est le modèle (fiable) le plus "faible"**
  - Les contraintes sont les plus fortes
  - Donc les résultats sont les plus généraux
    - Bornes sur les coûts
    - Résultats d'impossibilité
  - **Le modèle peut être renforcé (en levant certaines contraintes**
    - **Exemple : borne supérieure sur le traitement et/ou la durée de** transmission

Système répartie : Modèle Asynchrone



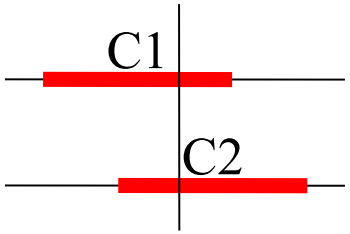
- **Les processus (sur différents sites) ne communiquent que par messages**
- **Trois type d'évènement :**
  - Locaux
  - Envoies de message
  - Réception de message

- **Les processus sur le même site**
- **Les processus sur des sites différents**
  - Pas d'horloge globale
  - Pas de borne:
    - La réception de message
    - le rapport des vitesses d'exécution des processus

Événement , histoire et Synchronisation

- **L'exécution d'un processus est une suite d'événements (local, émission, réception) appelée histoire (ou trace) du processus**
  - pour p1 : e_{11}, e_{12}, e_{13}, …, e_{1k}, …
- **Cette suite est ordonnée par l'horloge locale du processus**
- **Que veut dire "synchroniser deux processus ?"**
  - Exemple : l'exclusion mutuelle
- **=> Imposer un ordre entre des événements appartenant à ces deux processus**

fin(C2) précède deb(C1) ou fin(C1) précède deb(C2)



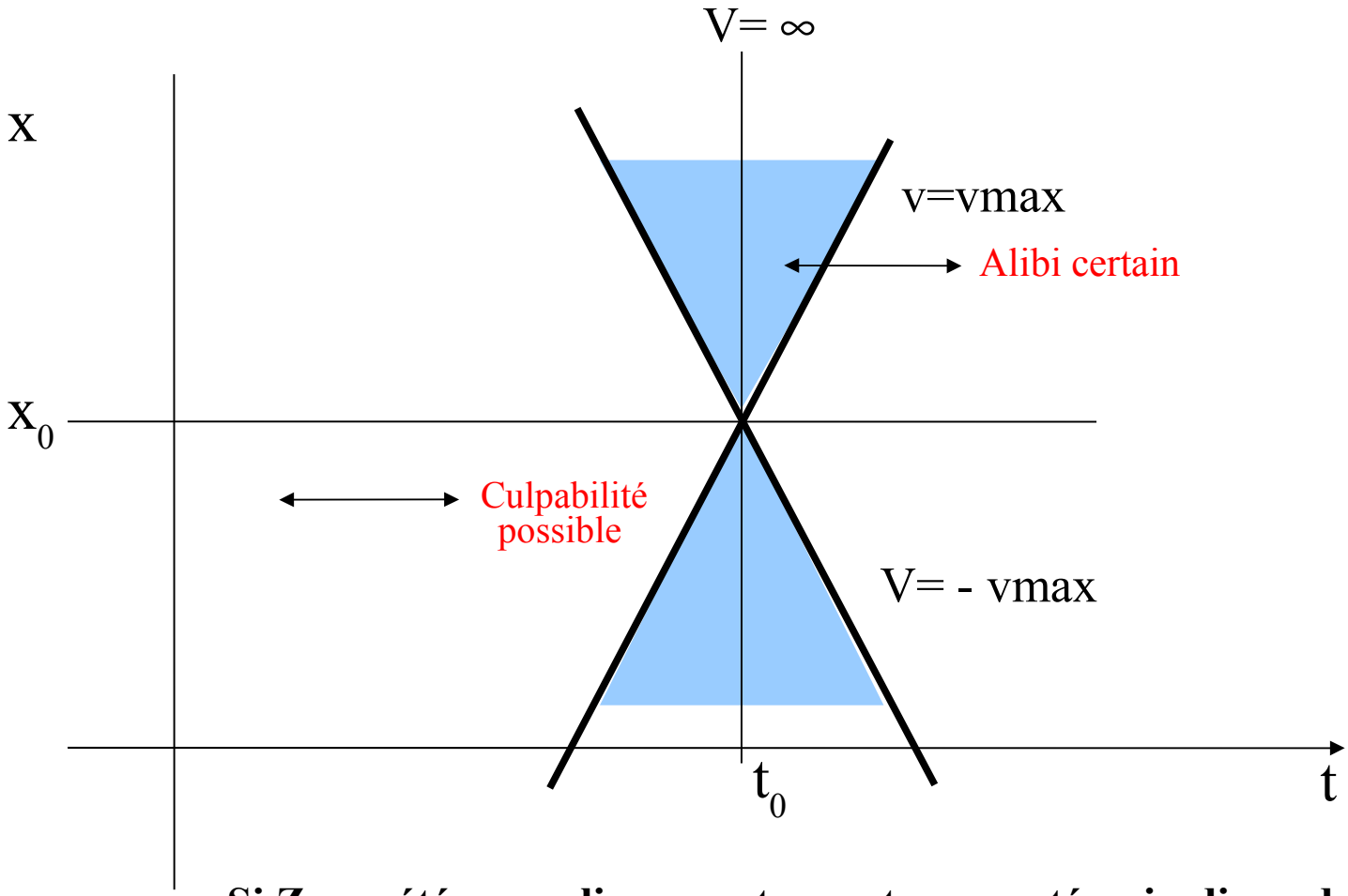
Rélation de pércédence

- **Le problème :**
  1) Définir une relation globale de précédence (donner un sens l'opérateur précède ci-dessus)
  2) Définir un ordre entre deux événements sur la seule base d!informations locales
- **Solution pour 1) utiliser le principe de causalité : la cause précède l'effet.**
  - **Une relation de précédence est dite causale si elle est** compatible avec ce principe. Application ici :
    - **Sur un processus :** un événement local ne peut agir localement que sur les événements postérieurs
    - **Entre deux processus :** l'envoi d'un message précède sa réception
    - **Composition :** la relation de causalité est transitive

Causalité selon Lamport[78]

- *e* précède causalement *e′ (e -> e′)* si :
  - *e* précède localement *e′* (sur 1 site, dans 1 processus), ou
  - ∃ un message *m* tel que *e = émission(m)*, *e′ = réception(m)*, ou
  - ∃ *e″* tel que (*e* précède *e″*) et (*e″* précède *e′*)
- La relation de précédence causale définit en fait une causalité potentielle (par négation)
  - Si *e -> e′*, on peut simplement dire qu'on ne viole pas le principe de causalité en disant que *e* est la cause de *e′*. On peut donc dire que e est une cause potentielle de *e′*, mais pas que e est effectivement une cause de *e′* (il faudrait pour cela analyser la sémantique de l!application)
  - En revanche, on peut dire avec certitude que *e′* ne peut pas être la cause de *e* le futur, jusqu'à nouvel ordre, n'agit pas sur le passé)

Représentation amusante de la causalité

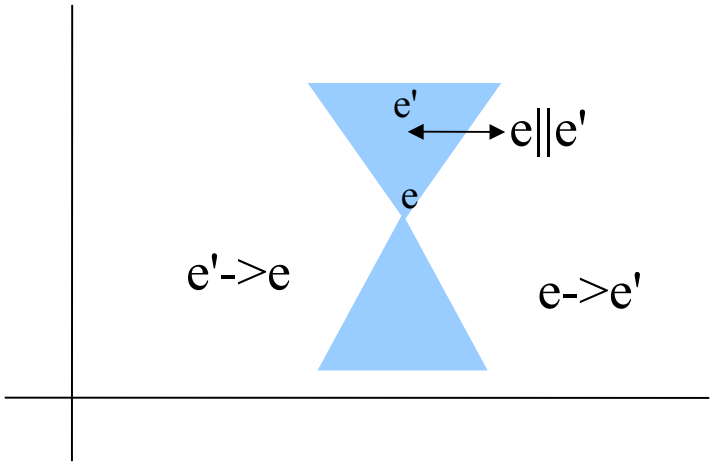


Si Z… a été vu au lieu x au temps t par un témoin digne de foi, peut-il avoir été au lieu x_0 au temps t_0 ?

N.B. t peut être antérieur ou postérieur à t_0

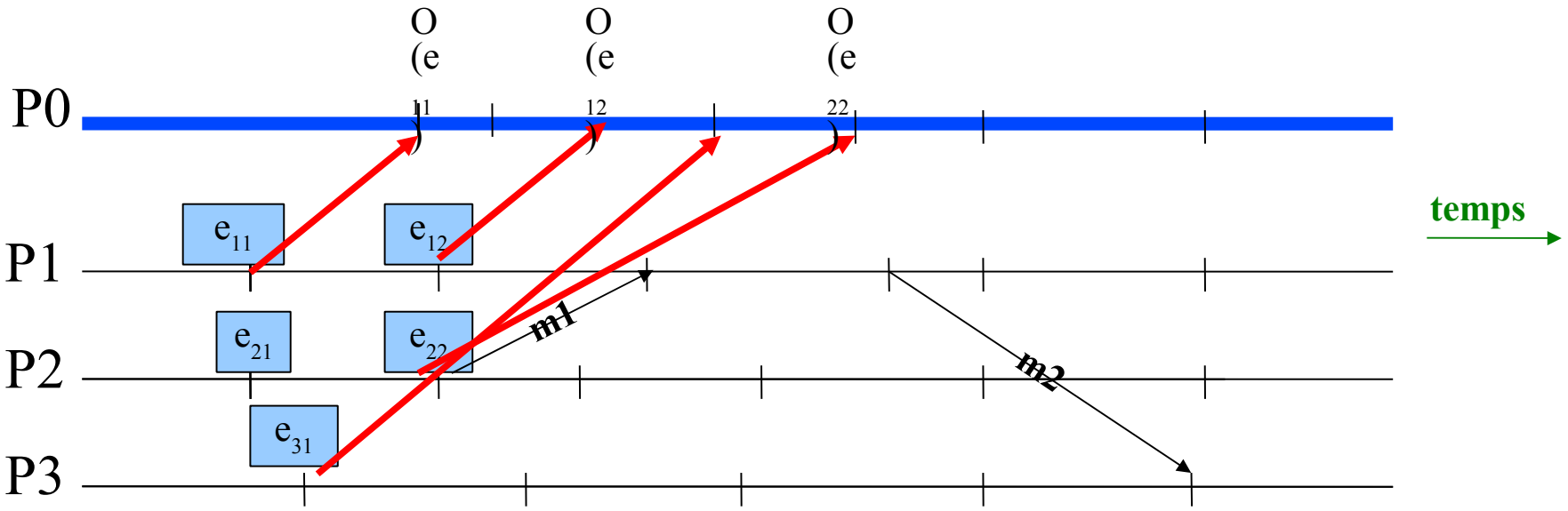
Histooir et causalité

- **Définition : passé (ou historique) d!un événement e**
  - ***hist(e)* = l'ensemble des *e′* tels que *e′* -> *e* ∪{*e*}**
- **Seul le passé strict de *e* peut influencer *e***
  - **si *e′* -> *e*, alors e' peut influencer e**
  - **si ¬ (e' -> e), alors il est certain que e' ne peut pas influencer e**
  - Si ¬ *(e -> e') et* ¬ *(e′ -> e)*, on note *e* || *e'* et on dit que *e* et *e′* sont causalement indépendants (aucun des deux n!appartient au passé de l'autre, aucun des deux ne peut influencer l'autre



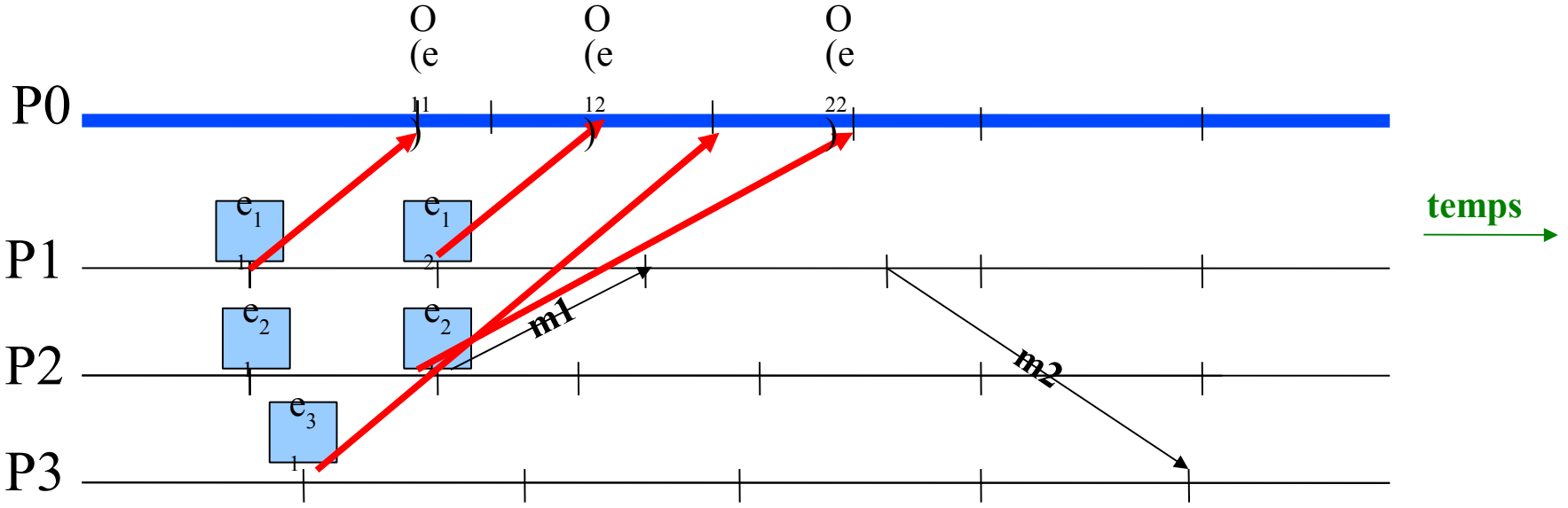
Vers un système de datation

- Le Problème
  - Vérifier nécessite une observation local
  - Distribution plusieurs temps....
- Solution
  - Mettre en place un Observateur P0
  - Vision global mais il faut construire un système de datation des événements compatible avec la causalité
- Garantie la validité de l'observation par rapport a ce qui sepasse pour pouvoir vérifier réellement le système



Vers un système de datation : observation et état global

- Le Problème
  - Garantie la validité de l'observation par rapport a ce qui sepasse pour pouvoir vérifier réellement le système
  - La complétude être sûr d'avoir toute observé
- **Et c'est ce qu'on va voir dans ce cours**
  - Horloge virtuelle
  - Estampillage par de Horloge locaux
  - Sauvegarde des états cohérent d'un système Répartie
  - Etc...



2- Approche Descriptive : Solution conceptuel et technique

Modèles architecturaux &

Modèle de communications

Modèles d'architecture logique : Mais avant le physique ou matériel

- **Systèmes Multi-Processeurs (SMP)**
  - mémoire partagée avec accès direct pour tous les CPUs
  - mémoire cache pour assurer un accès rapide aux données et éviter les goulots d'étranglement entre les calculs
- **Machines parallèles homogènes**
  - noeuds de calcul montés dans des « racks » reliés par un réseau d'interconnexion unique et très haute performance
  - ex. : super-calculateurs, clusters
- **Machines parallèles hétérogènes**
  - ordinateurs variés en termes de processeurs, de mémoire, de bande passante, etc.
  - généralement associés à une couche logicielle type middleware

Modèles d'architecture logique

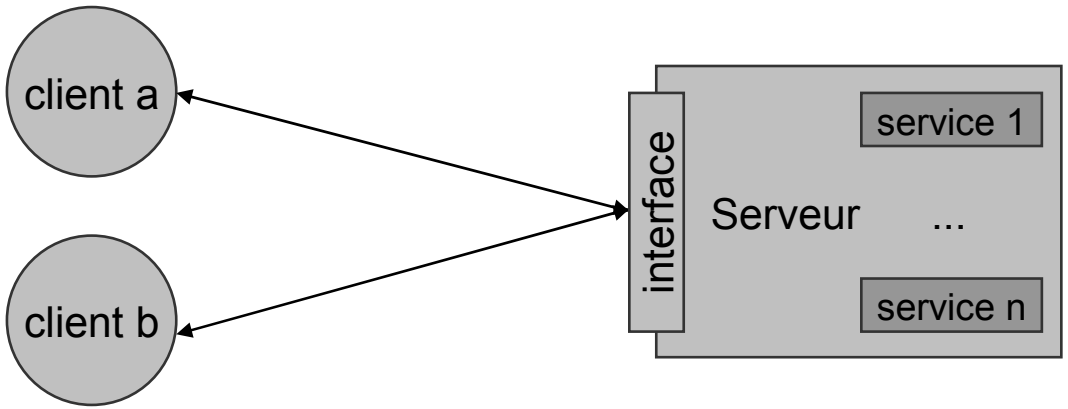
- **Définition**
  - L'architecture d'un système est sa structure en termes de composants spécifiquement séparés, conçue dans le but d'assurer un bon fonctionnement en fonction des demandes présentes mais aussi futures
  - Le système doit être fiable, gérable, adaptable et tout cela à un coût réaliste. Un tel modèle est décrit par :
    - le placement des entités sur le réseau (distribution des données, charge effective)
    - les relations existantes entre ces entités (rôles fonctionnels, communications)
- **Principales architectures**
  - client/serveur
    - architectures 2-tiers
    - architectures 3-tiers
    - architectures n-tiers
  - métacomputing
    - P2P
    - clusters
    - grilles
    - agents autonomes

Le modèle client-serveur (1)

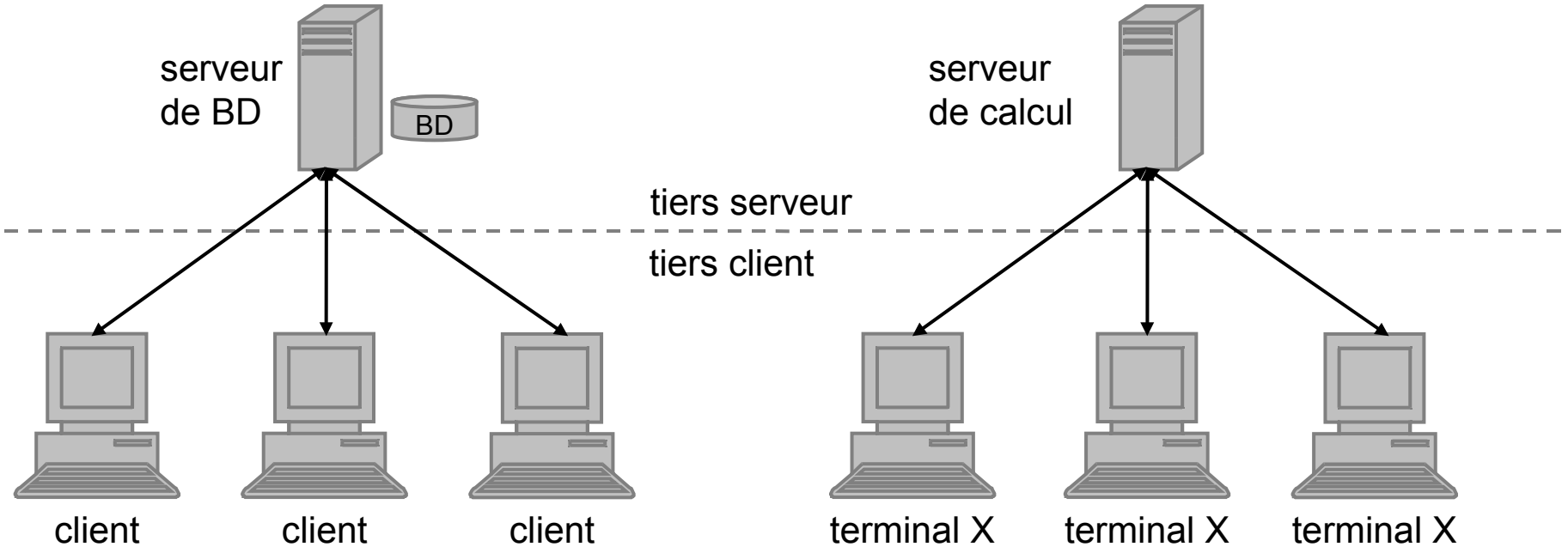
- **Analogie : le modèle interviewer / interviewé**
  - communication entre 2 entités
    - toute communication concernant les éléments d'un groupe peut être décomposée en un ensemble d'échanges entre deux entités
  - une entité a l'initiative du dialogue (l'interviewer) et l'autre est dans l'attente d'une requête (l'interviewé)
  - l'entité interviewée est programmée pour répondre à un ensemble très précis de requêtes
    - la liste des requêtes autorisées doit être parfaitement définie
    - cette liste est l'***interface*** de l'entité interviewée
    - à chaque requête correspond un service
- **Le modèle client/serveur**
  - l'interviewé fournit des services c'est le ***serveur***
  - l'interviewer requiert des services c'est le ***client***

Le modèle client-serveur (2)

- **Fonctionnement**
  - le client émet un message contenant une requête à destination d'un serveur
  - le serveur exécute le service associé à la requête émise par le client
  - le serveur retourne au client un message contenant le résultat du service effectué



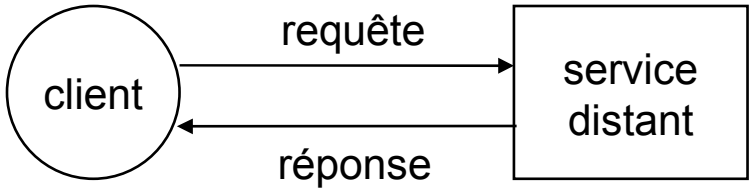
Architecture 2-tiers



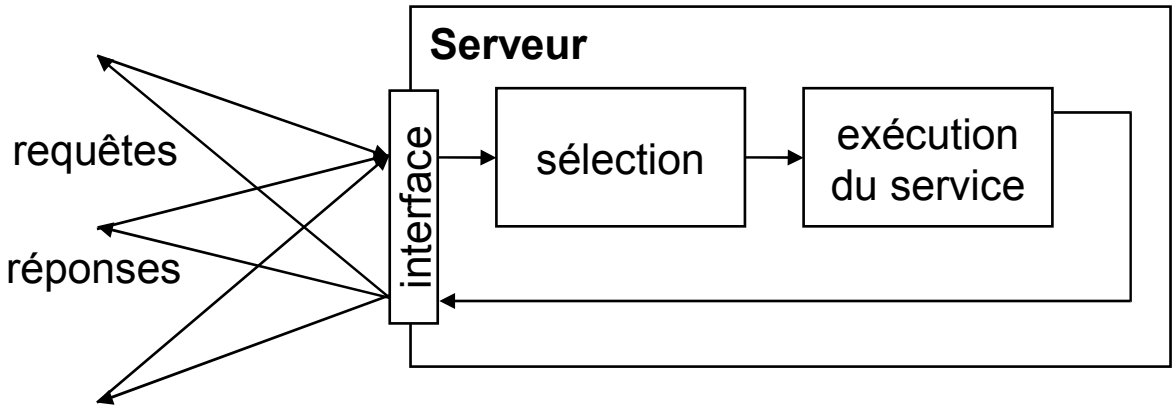
- Autres exemples de serveurs
  - serveur de fichiers
  - serveur de noms
  - serveur Web
  - serveur d'impression
  - etc.

Le modèle client-serveur (3)

- **Du point de vue du client**



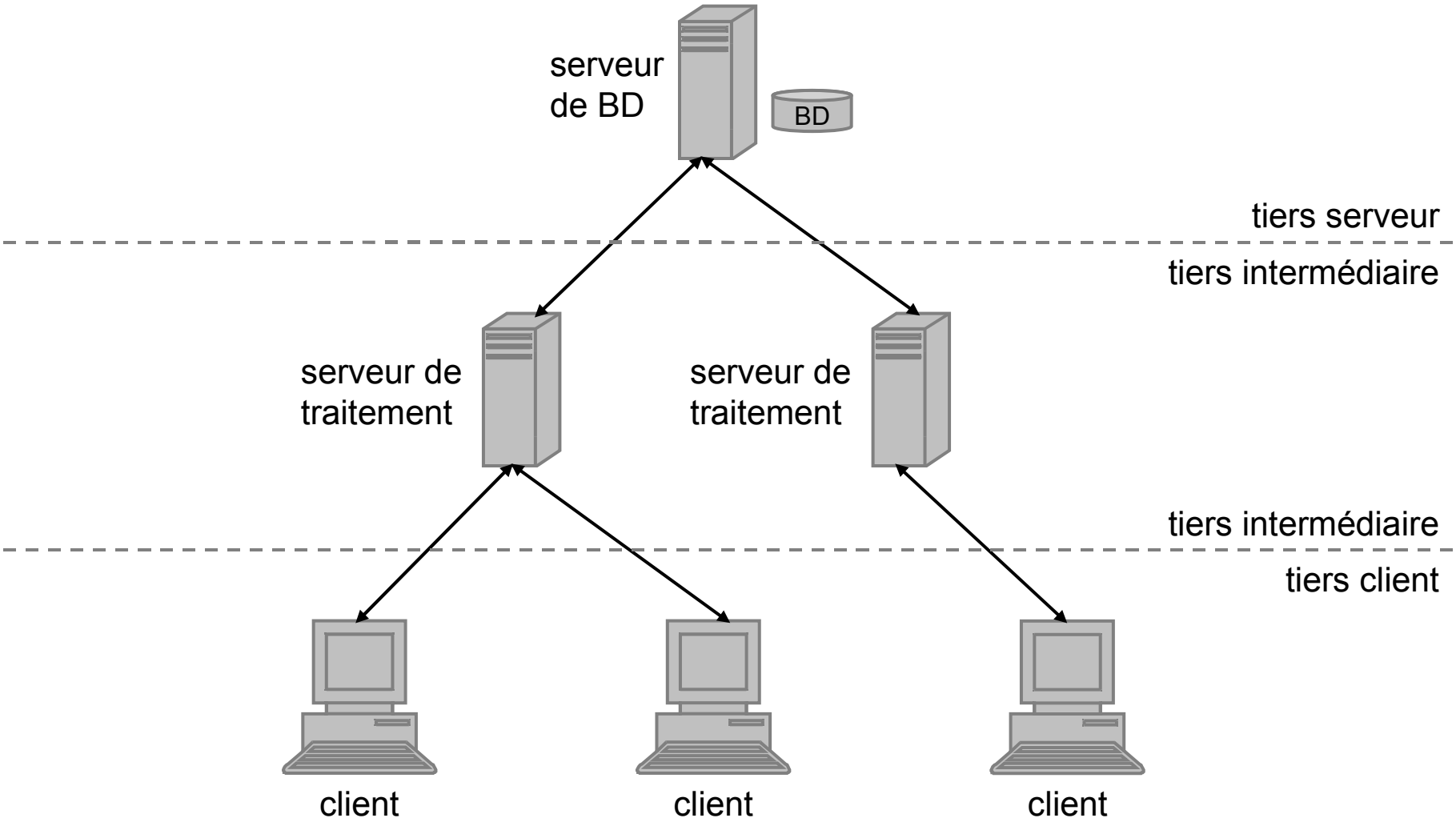
- **Du point de vue du serveur**
  - gestion des requêtes (priorité)
  - exécution du service (séquentiel, concurrent)
  - mémorisation ou non de l'état du client



Le modèle client-serveur – Remarques

- la communication est toujours initiée par le client
  - le serveur est en mode réactif
- mode d'exécution synchrone
- chaque entité peut jouer les deux rôles (client et serveur)
- on ne se préoccupe pas de la manière dont la communication est réalisée
  - on suppose seulement qu'il existe un moyen d'échanger des messages
  - la plupart des protocoles classiques (FTP, HTTP, etc.) sont basés sur ce principe
- l'interface du serveur est un élément essentiel de la communication
  - définit les requêtes autorisées
- concept simple mais base théorique pour des architectures beaucoup plus compliquées

Architecture 3-tiers



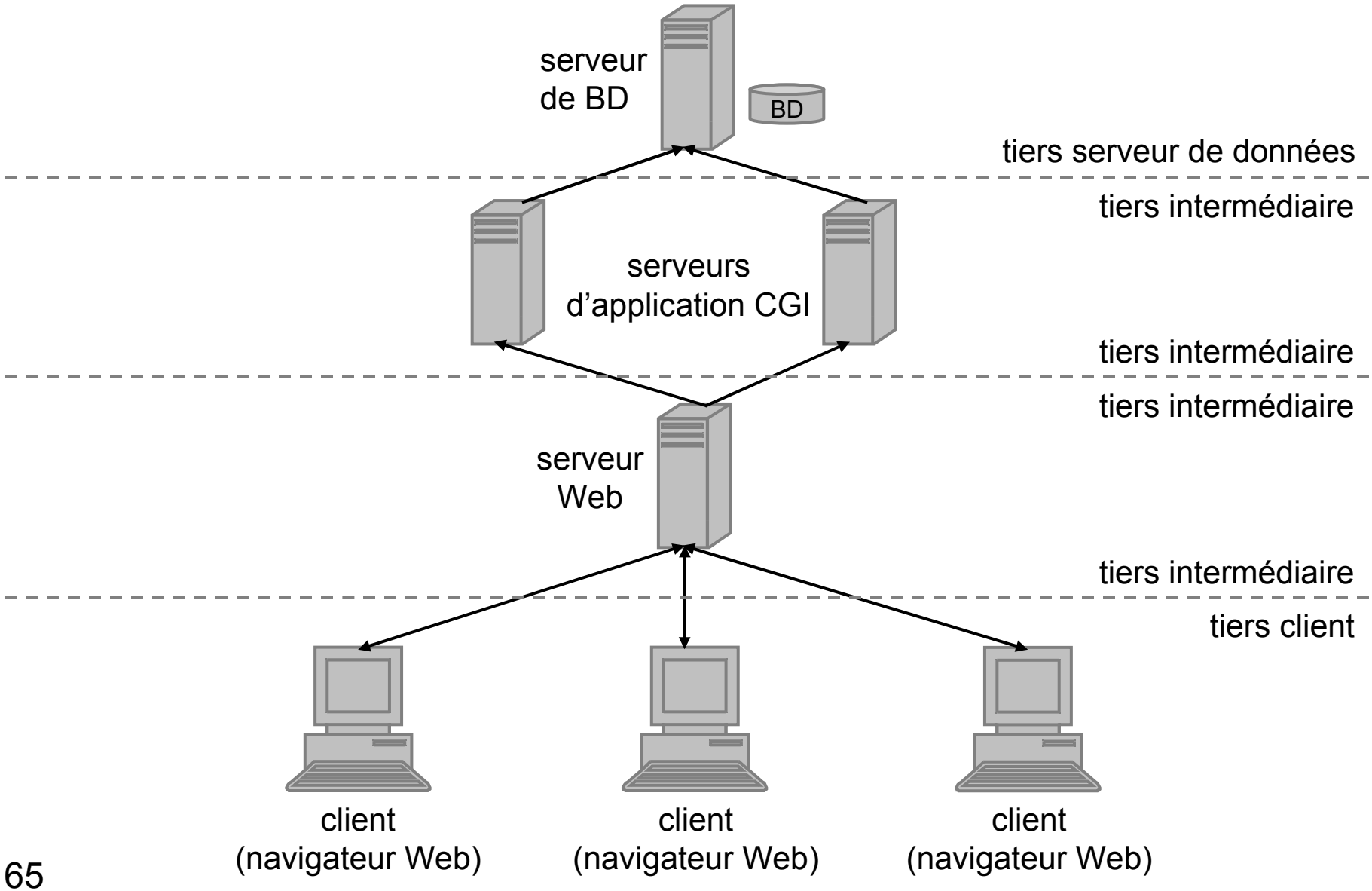
Apport des architecture 3-tiers

- **logique métier dans le tiers intermédiaire**
  - performance
    - traitement métier proche du tiers de données, quelque soit la localisation du tiers client
  - scalabilité
    - possibilité d'ajouter des tiers métiers pour satisfaire la demande de nouveaux clients
  - modularité
    - tiers client peut être développé indépendamment du tiers de données
  - cohérence et évolutivité
    - centralisation de la logique métier dans le tiers intermédiaire
    - évite les incohérences liées à des implantations différentes de la logique métier dans les différents clients
    - permet la prise en compte facile de changements de logique métier

Vers les architecture n-tiers

- **Dans les architectures 3-tiers :**
  - tiers client = présentation de l'application
  - tiers serveur = gestion des données
  - tiers intermédiaire = ensemble de services fournis au client pour consulter/modifier/etc. les données
    - peut être considéré comme un ensemble de tiers, chacun correspondant à un service particulier
- **Exemple : application Web**
  - tiers 1 : tiers client = présentation par navigateur Web
  - tiers 2 : tiers intermédiaire = distribution par serveur Web
  - tiers 3 : tiers intermédiaire = service par scripts côté serveur
  - tiers 4 : tiers de données = stockage par BD relationnelle

Exemple d'architecture n-tiers : application Web



Modèles du « meta-computing »

super-ordinateur virtuel permettant le partage de ressources au travers d'un réseau

- **Schéma opératoire**
  - connecter des ressources hétérogènes de manière à former une très grande capacité de calcul
  - ces ressources peuvent être distribuées le long de très grands réseaux, voire sur l'Internet tout entier

CaractŽristiques	P²P	Clusters	Grilles
Type de noeuds	PCs de bureau	PCs dŽdiŽs, racks	PCs de bureau, stations de travail, serveurs
Participants	Un, plusieurs	Plusieurs	Plusieurs
Gestion des utilisateurs	CentralisŽe, dŽcentralisŽe	CentralisŽe	DŽcentralisŽe
Gestion des ressources	DistribuŽe	CentralisŽe	DistribuŽe
Allocation / PrŽvision	DŽcentralisŽe	CentralisŽe	DŽcentralisŽe
Taille	Millions	¹··s	Milliers^ millions
CapacitŽ en calcul	Haute, mais peut varier	Garantie	Haute, mais peut varier
Bande passante	Basse	Tr□s haute	Basse

P2P et « Internet computing »

- **Peer to Peer (P2P)**
  - Peer = entité disposant de capacités similaires aux autres entités d'un système
  - outils de stockage et d'échange de données
- **Principes**
  - éviter les vulnérabilités du réseau par la répartition
  - découvrir les ressources par un procédé de diffusion
    - méta-description qui permettent de les retrouver par des moteurs de recherche
- **P2P « purs » et « hybrides »**
  - P2P « purs » = tous les noeuds participants sont des Peers et aucun serveur central n'est utilisé pour contrôler, coordonner ou gérer les échanges entre Peers
  - P2P « hybrides » = serveur central qui effectue un certain nombre de fonctions essentielles

Avantages du P2P

- réduction du coût global par répartition
- accroissement de la scalabilité et de la sûreté de fonctionnement
  - pas d'autorité centrale
- agrégation des ressources et interopérabilité des systèmes
- accroissement de l'autonomie
  - les utilisateurs ne dépendent pas d'un seul fournisseur centralisé
- anonymat et respect de la vie privée
- très grande dynamicité
  - les ressources entrent et sortent du système de manière totalement continue
  - la communication ne s'en trouve généralement pas affectée.

- **« Internet Computing »**
  - applications P2P parallélisables
    - une même tâche est effectuée sur chaque Peer avec différents paramètres
  - utiliser les ressources inutilisées des noeuds
  - si un noeud détecte une inactivité, il informe un client maître
  - activité complètement transparente pour l'utilisateur
- **Exemples d'implantations**
  - P2P pur : KaZaa, FreeHeaven, JXTA, Gnutella, Freenet
  - P2P hybride : Napster
  - Internet Computing : SETI@Home, RSA-135

Clusters

ensemble d'ordinateurs complets (processeur, mémoire, périphériques d'E/S) en général interconnectés par l'intermédiaire d'un LAN

- utilisé comme une seule ressource de calcul unifiée
- les composants d'un cluster sont typiquement des stations de travail ou des PCs et sont généralement homogènes
- premier cluster « Beowulf » créé en 1994 pour la NASA

- **Architecture**
  - tiers d'accès = fournit les services d'accès et d'authentification
  - tiers de gestion = responsable des fonctionnalités basiques du cluster (service de fichiers, gestion des sauvegardes, ...)
  - tiers de calcul = fournit la puissance de calcul du cluster
    - travaux exécutés sur un ou plusieurs noeuds
  - deux services critiques
    - « Batch Queing System » = reçoit les demandes de travaux
    - « Job Scheduler » = détermine l'ordre d'exécution des travaux en fonction de la charge processeur moyenne, la charge mémoire moyenne, ...

Grilles

partage de ressources et de puissance de calcul dans de très larges organisations multi-institutionnelles

- infrastructure capable d'unifier diverses ressources
- met en oeuvre un grand nombre de ressources hétérogènes
- peuvent s'accroître de quelques ressources à plusieurs millions
- probabilité de ressources défaillantes élevée

- **Architecture**
  - une fabrique de grille
    - toutes les ressources (PCs, SANs, clusters,...) distribués géographiquement
  - un middleware de grille
    - offre les services de base (gestion des processus distants, co-allocation des ressources, accès au stockage, ...)
  - un environnement de développement de grille
    - offre des services de très haut niveau permettant aux développeurs de développer des applications et des brokers agissant de manière globale
  - des applications de grille et des portails de grille
  - ex. : Globus, Legion, CERN Data Grid ou Unicore

Agents autonomes [Wooldridge]

- **Définition**

  entités matérielles ou logicielles dotées des propriétés suivantes

  - ***autonomie***

    les agents agissent sans l'intervention directe d'humains ou d'autres agents, et ont le contrôle de leurs actions et de leur état interne

  - ***capacité sociale***

    les agents interagissent avec d'autres agents (et éventuellement des humains) grâce à des langages de communication agent

  - ***reactivité***

    les agents perçoivent leur environnement (le monde physique, un utilisateur via une interface graphique, un ensemble d'autres agents, Internet, une combinaison de tout cela), et répondent en permanence aux changements qui s'y produisent

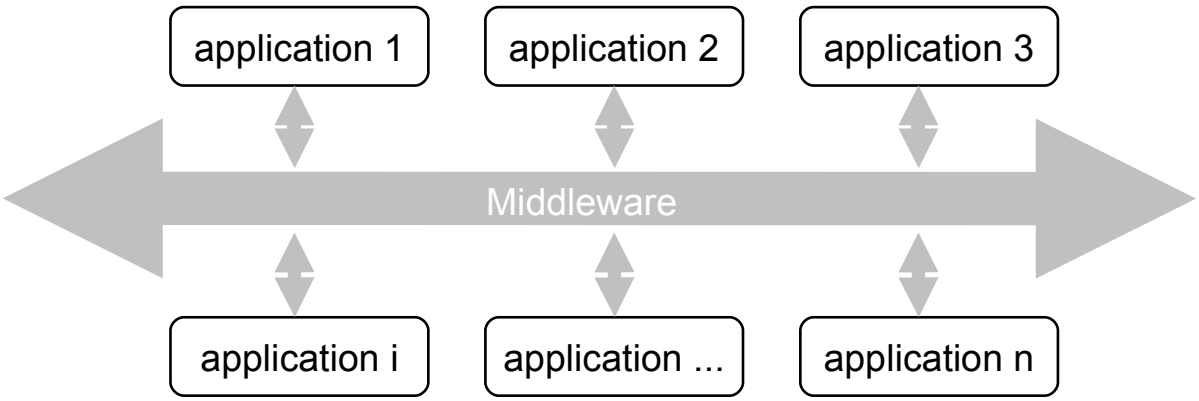
  - ***pro-activité***

    les agents n'agissent pas seulement en réponse aux sollicitations de l'environnement mais peuvent exhiber un comportement orienté par un but en prenant l'initiative d'agir

- **Caractéristiques**
  - à la frontière des systèmes distribués et de l'IA
  - inspiration sociologique

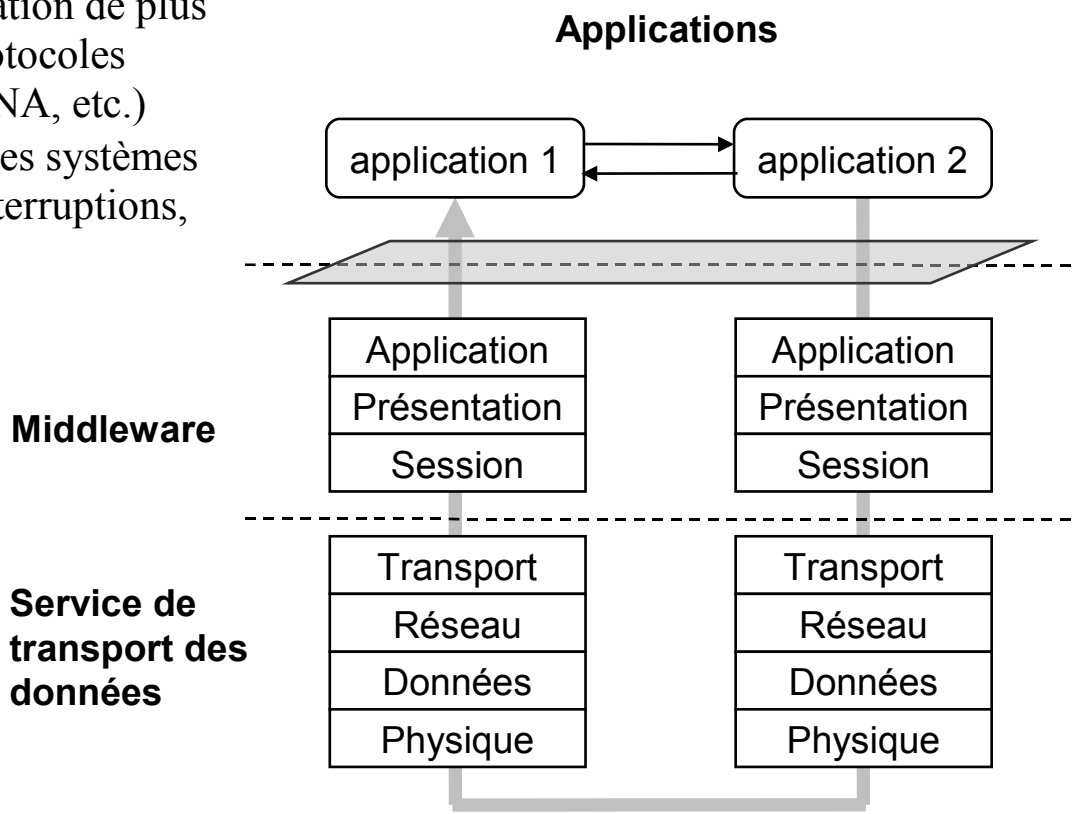
Modèle de communication - les middlewares

- **But** = cacher l'hétérogénéité des plate-formes mises en oeuvre pour les applications
  - intergiciel = bus de communication auquel les applications se connectent par l'intermédiaire d'une interface



Positionnement du middleware

- couche indépendante
  - des systèmes d'exploitation
  - des machines
  - du réseau de transmission
- repose sur
  - les structures de communication de plus bas niveau telles que les protocoles réseau (TCP/IP, DECnet, SNA, etc.)
  - les mécanismes offerts par les systèmes d'exploitation (gestion d'interruptions, etc.)



Services du middleware

- **Disponibilité sur différentes machines**
- **Fiabilité du transfert**
  - assurance que le message...
    - atteindra le destinataire
    - en un seul exemplaire
    - même en cas de panne d'une machine ou de liens réseau
- **Adaptation au trafic**
  - variation du nb d'applications, du nb et du type de machines
- **Support de différents schémas de communication**
  - communication 1-1, 1-n
  - communication synchrone/asynchrone
- **Service de nommage**
  - conversion d'un nom en adresse physique
- **Support de la notion de transaction**
  - si plusieurs entités appartiennent à une transaction, toutes doivent pouvoir exécuter leur travail ou alors aucune d'elles
- **etc.**

Différentes approches possibles

- **considérer tous les composants en tant que fichiers**
  - périphériques traités en tant que fichiers : Plan 9
  - systèmes de fichiers distribués : NFS
  - pb = transparence limitée à l'échange de fichiers
- **appel de procédures à distance : RPC**
  - appel d'une fonction dont l'implantation est sur une autre ressource
- **objets distribués : RMI, CORBA, DCOM, etc.**
  - dissociation entre l'interface et l'implantation
  - seule l'interface est distribuée
- **échange de messages : JMS**
  - envoi de messages entre les participants
- **agents mobiles : Aglet, AgentTcl, etc.**
  - le code à exécuter et les données se déplacent d'une ressource à une autre sur le réseau

Echange de messages : JMS

- **Bus logiciel à messages (MOM)**
  - asynchronisme émetteur/récepteur : envoi de messages
  - désignation explicite ou anonyme du destinataire
  - connexion 1-1 ou 1 parmi N ou diffusion
- **Propriétés du bus**
  - priorité, ordre des messages
  - durée de vie des messages
  - filtrage des messages
    - émetteur, type de message, priorité
    - attributs
    - contenu
  - notification des erreurs
- **Avantage**
  - simplicité du modèle, pas d'interblocage
  - problèmes : propagation des erreurs

Appel de procédure à distance : RPC

- **Remote Procedure Call (RPC)**
  - Birrel et Nelson, 1984
  - basé sur l'échange explicite de messages entre processus
  - capacité de cacher le réseau sous-jacent en permettant à un processus d'appeler une fonction dont l'implantation se situe sur une autre ressource
- **Principe**
  - un processus A appelle une procédure sur une machine B
    - le processus appelant sur A est suspendu
    - les paramètres d'invocation sont transmis à B
    - l'exécution de la procédure sur B est lancée, le résultat est calculé
    - le résultat est renvoyé au processus appelant
  - comme si le processus sur A avait appelé une procédure locale
  - aucun échange de message n'est visible pour le programmeur
- **Problèmes**
  - les procédures appelantes et appelées ne partagent pas le même espace d'adressage mémoire
  - les paramètres doivent absolument transiter en même temps que l'appel, ce qui peut poser différents problèmes d'implantation

Objets distribués (Java RMI, CORBA, DCOM, etc.)

- **Objets distribués**
  - encapsulation des attributs et des méthodes
  - séparation entre les interfaces et les objets les implémentant
  - l'interface est placée sur une machine tandis que l'objet lui-même réside sur une autre
- **Principe**
  - quand un client s'associe à un objet distribué
    - une implantation de l'interface de l'objet appelée proxy est chargé dans l'espace d'adressage du client
    - ce proxy assure l'invocation des méthodes en communiquant avec l'implantation réelle de l'objet qui peut se trouver n'importe où dans le système au moyen d'un squelette agissant comme un médiateur
- **Important**
  - l'état du système n'est pas distribué : il réside sur une machine et une seule
  - seules les interfaces des objets rendues disponibles sont accessibles par les processus tournant sur les autres machines.

Agents mobiles (Aglet, MOA, AgentTcl, etc.)

- **Code mobile**
  - programme pouvant se déplacer d'un site à un autre sur un réseau
  - ex. : Postscript, SQL, applets, etc.
  - caractéristique = 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

Agents mobiles (Aglet, MOA, AgentTcl, etc.)

- **Agent mobile**
  - processus, incluant du code et des données, pouvant se déplacer entre des machines pour réaliser une tâche
  - Principe de migration (en Java)
    - sérialisation de l'état de l'agent
    - envoi du code et de l'état de l'agent
    - destruction de l'agent sur le site origine
    - création d'un *thread* sur le site de destination
    - chargement du code de l'agent
    - dé-sérialisation de l'agent
    - exécution
- **Limites**
  - coût important de la migration
    - intéressant si réseaux lents et données à traiter volumineuses
  - problèmes de sécurité
  - problèmes d'autonomie
  - gestion de l'état, adressage des agents

Les services Web

- **Les architectures orientées service**
  - Approche conceptuelle par service : un service est plus stable, plus cohérent, plus fonctionnel, …
  - Par rapport à l'objet un services specifie de qui offre comme méthode mais aussi ses besoin en terme d'invocation
- **Les services Web est une implémentation d' la SOA sur le Web**
  - Technologies rendu possible grace à l'ubiquité de XML
  - Des effort de standardisation notamment
  - Un protocole abstrait SOAP structure de message pas de transport (c'est de texte il peut être transporté n'importe comment).
  - WSDL interface de services
  - UDDI un standard de publication localisation.

Quelques références

- **http://rangiroa.essi.fr/cours/**
  - collection très importante de supports de cours et liens en tous genres sur
    - la programmation
    - les réseaux, Internet
    - les systèmes d'exploitation
    - les systèmes répartis et la construction d'applications réparties
- **http://www-mips.unice.fr/~baude/Systemes-Distribues/index.html**
  - systèmes d'exploitation et applications réparties
- **http://spe.univ-corse.fr/bernardiweb/cours.htm**
  - supports de cours de
    - systèmes distribués
    - programmation système
    - CORBA