

# **Systèmes et Applications Distribués**

---

## **Introduction et Définitions**

# Objectifs et Plan du cours

---

## ◆ Introduction

- ▶ Définitions
- ▶ Problématiques
- ▶ Architectures de distribution

## ◆ Distribution intra-applications

- ▶ notion de processus
- ▶ programmation multi-thread (concurrence)

## ◆ Distribution inter-applications et inter-machines

- ▶ Sockets
- ▶ Middlewares par appel de procédures distantes (RPC)
- ▶ Middlewares par objets/opérations distribués (Java RMI, gRPC)

## ◆ Conclusion

# Plan du cours

---

## ◆ Introduction

- ▶ Définitions
- ▶ Problématiques
- ▶ Architectures de distribution

## ◆ Distribution intra-applications

- ▶ notion de processus
- ▶ programmation multi-thread (Concurrence)

## ◆ Distribution inter-applications et inter-machines

- ▶ Sockets
- ▶ Middlewares par appel de procédures distantes (RPC)
- ▶ Middlewares par objets/opérations distribués (Java RMI, gRPC)

## ◆ Conclusion



# Plusieurs exemples pour commencer

---

## ◆ Coordination d'activités :

- ▶ Systèmes à flots de données («WorkFlow » )

## ◆ Communication et partage d'informations

- ▶ Bibliothèques Virtuelles

## ◆ Collecticiels:

- ▶ Édition coopérative
- ▶ Téléconférence
- ▶ Ingénierie Concourante (Stellantis, ex. PSA)

## ◆ Services grand public

- ▶ Presse électronique
- ▶ Télévision interactive
- ▶ Commerce électronique

# Exemple 1 : Flots de données (*WorkFlow*)

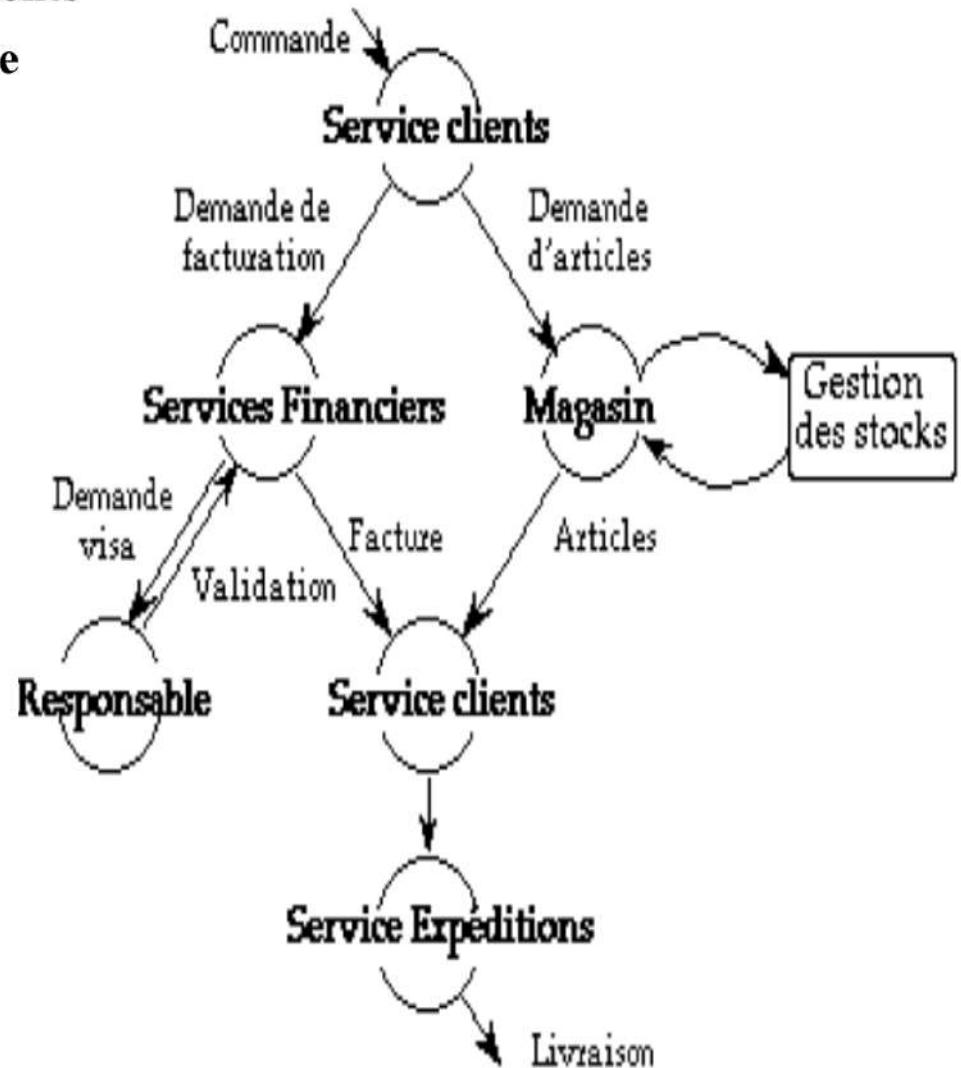
---

- ◆ 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 de tel type d'applications
  - ▶ Systèmes à rôle
    - Dans un service, des agents (fonctionnaires) sont assignés à des tâches
    - Dynamique ou statique
  - ▶ Systèmes réactifs
    - Souvent événementiel => l'arrivée d'un « document » déclenche la tâche
    - Utilisation des événements

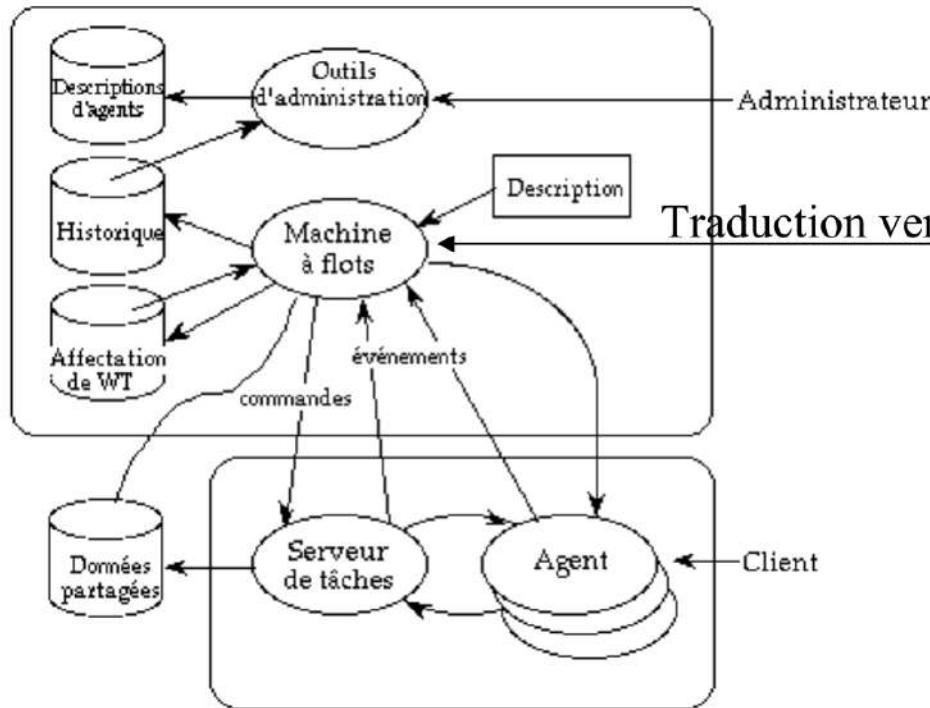
# Traitement d'une commande

---

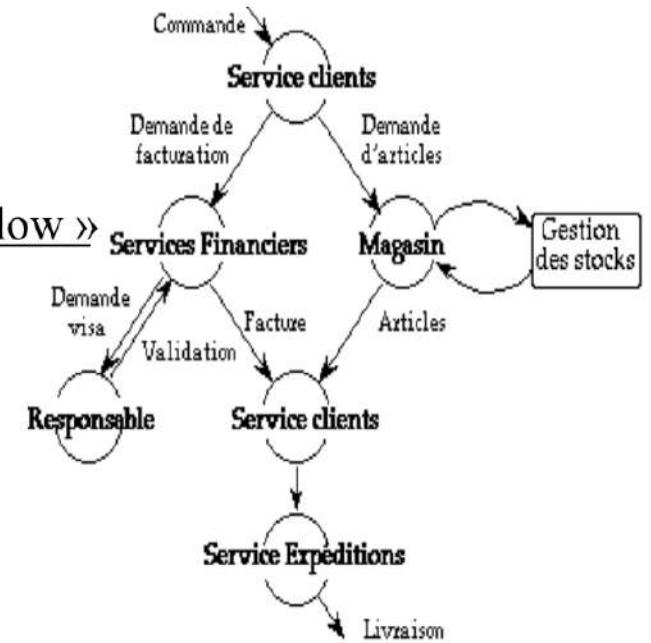
- ◆ Une commande reçue est traitée par le Service Clients
- ◆ Transmet aux Services Financiers une demande de facturation
- ◆ et au Magasin une demande de fournitures
- ◆ etc.



# Architecture des systèmes à flots de données



Traduction vers un langage de « workflow »



- ◆ **Gestion centralisée de la coordination**
  - ▶ Conséquence : peu de tolérance aux pannes
- ◆ **Nécessite un mécanisme d'allocation de tâches**
  - ▶ Dynamique et/ou statique
- ◆ **Les serveurs de tâches peuvent être répartis/distribués**

## Quelques 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 distribution (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 machines ;
  - ▶ réseau local;
  - ▶ réseau à grande distance.

## **Exemple 2 : Les collecticiels (« Groupware »).**

---

- ◆ **Logiciel pour travail coopératif entre plusieurs utilisateurs géographiquement réparties.**

- ▶ Tâches communes
- ▶ Ressources peuvent être communes
- ▶ Les actions des autres peuvent être soit :
  - Immédiatement perceptibles : synchrones
  - Différées : asynchrones

- ◆ **Les exemples Multiples :**

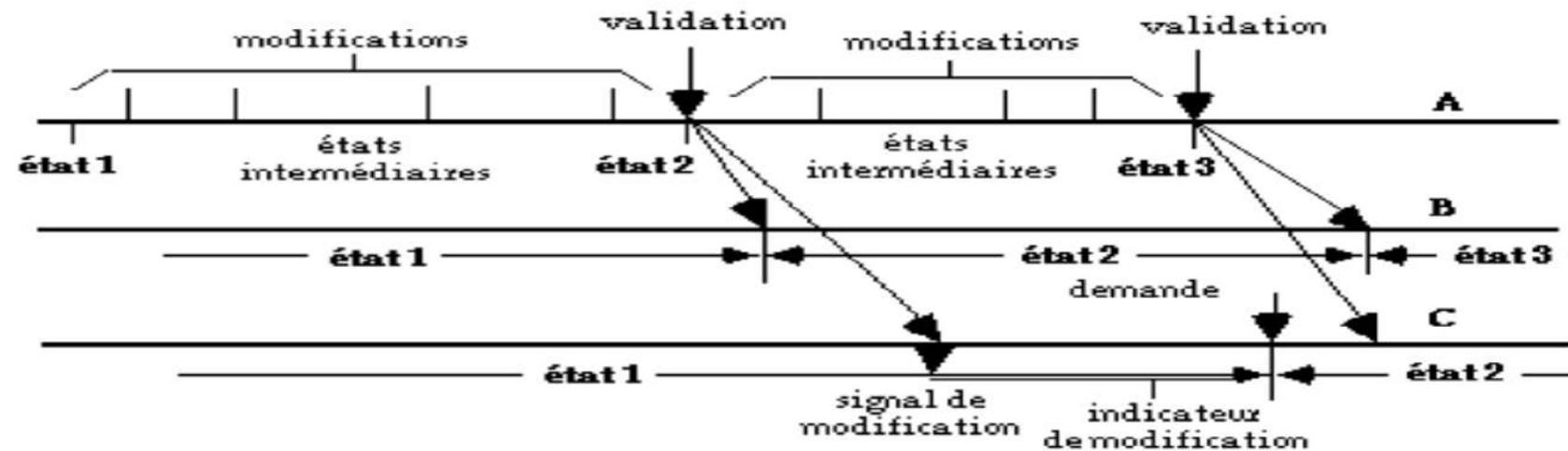
- ▶ Rédaction coopérative de documents : asynchrone
- ▶ Téléconférence : synchrone
- ▶ Ingénierie coopérative : les deux quand c'est un objet virtuel
  - Conception coopérative d'une voiture par exemple (plateforme PSA)
- ▶ Télé-opération :
  - Les missions spatiales.

## **Exemple 2 : rédaction coopérative de documents (1)**

---

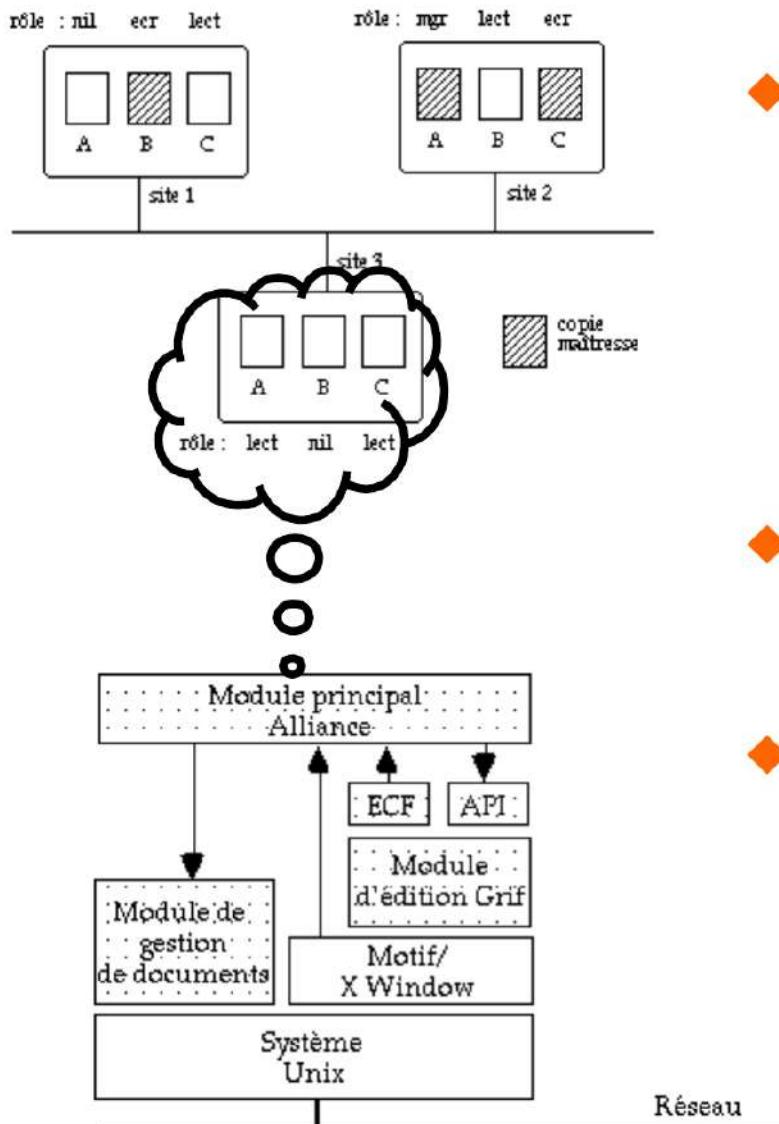
- ◆ **Un Document XML par exemple**
- ◆ **Comporte des parties (intro, chap1... concl)**
- ◆ **Un ensemble d'auteurs doivent coopérer pour sa rédaction (un livre, un journal, etc.)**
- ◆ **Le document est un objet abstrait.**
- ◆ **Chaque utilisateur a une vue différent :**
  - ▶ les parties (ou fragments) 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 coopérative de documents (2)



- ◆ **A Chaque partie du document correspond une copie maîtresse.**
  - ▶ L'administrateur du bout du document décide
- ◆ **Localement un auteur voit 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 voir le dernier état du fragment
  - ▶ Voir exemple A valide les changements
    - 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

- Laissée à la charge des utilisateurs (gestion lâche)

## ◆ Structure du système

- Chaque utilisateur possède un exemplaire du document
- Conséquence, action d'édition locale : communication seulement pour la coordination
- On est dans un cas de collecticiel **Asynchrone**

## ◆ Cas **Synchro**ne : e.g Vidéo conférence

- Tableau un objet de coopération où la cohérence doit être synchrone
- Plus de communications 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 locale des événements et notifications à des processus « représentants »

## Exemple 3 : Télévision interactive

---

### ◆ Systèmes de plus en plus présents

- Services TV de *Free* ou *orange* ou autre....
- Programme à la demande
- Télé Achat
- Jeux interactifs
- Pour cela, il suffit d'un poste de télévision + un abonnement au réseau.

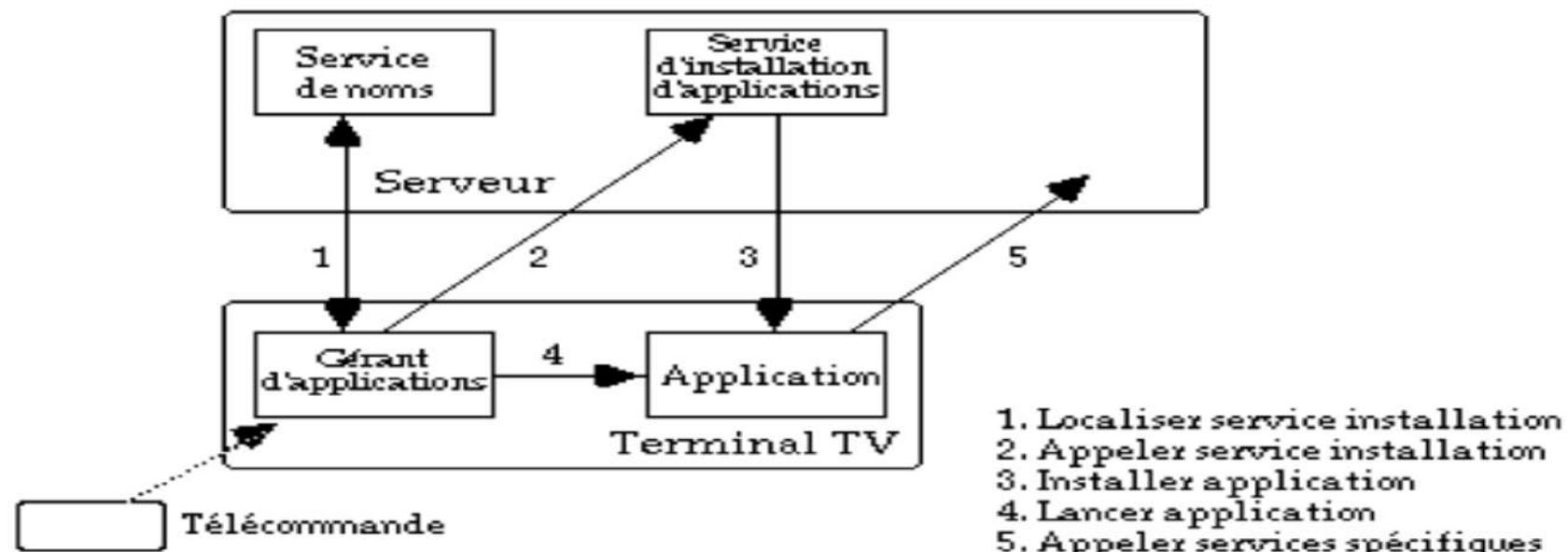
### ◆ Il s'agit d'une application « Grand public »

- Le réseau doit être extensible
- Et quel que soit le nombre des abonnés
- La qualité de service doit rester inchangée
- Ce qu'on appelle une application qui supporte le **Passage à l'échelle**

# Architecture d'un système de TV interactive (1)

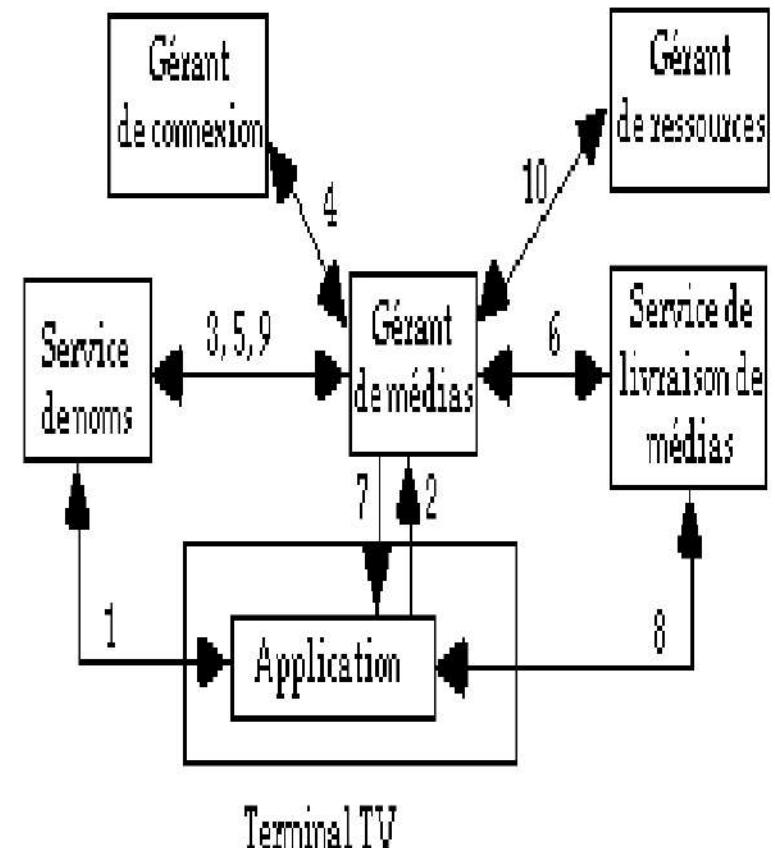
## ◆ Organisation générale orientée Objet

- Interface : un ensemble de méthodes ou procédures (opérations)
- Chaque méthode est réalisée concrètement par un service implémenté sur un serveur.
- Chaque objet est accessible par une référence
- Un service de noms est donc par défaut nécessaire pour trouver les références d'un objet à partir d'un nom symbolique
- La référence du service de noms est connue à l'initialisation du système.



# Architecture d'un système de TV interactive (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 film, 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



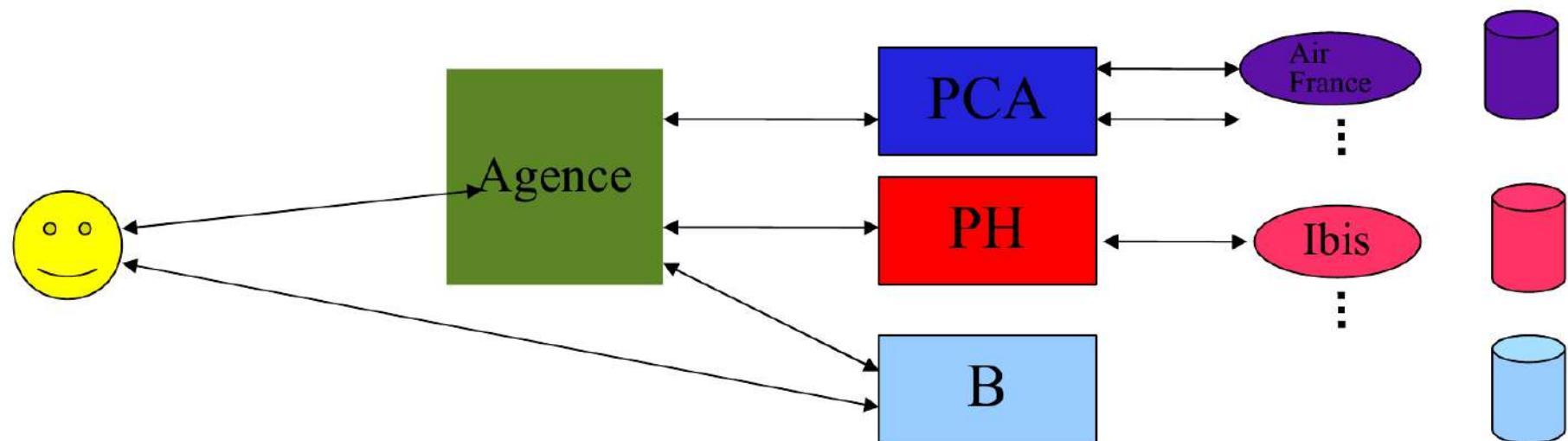
## Quelques remarques

---

- ◆ **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 concerné 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/services destinés 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 Électronique

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

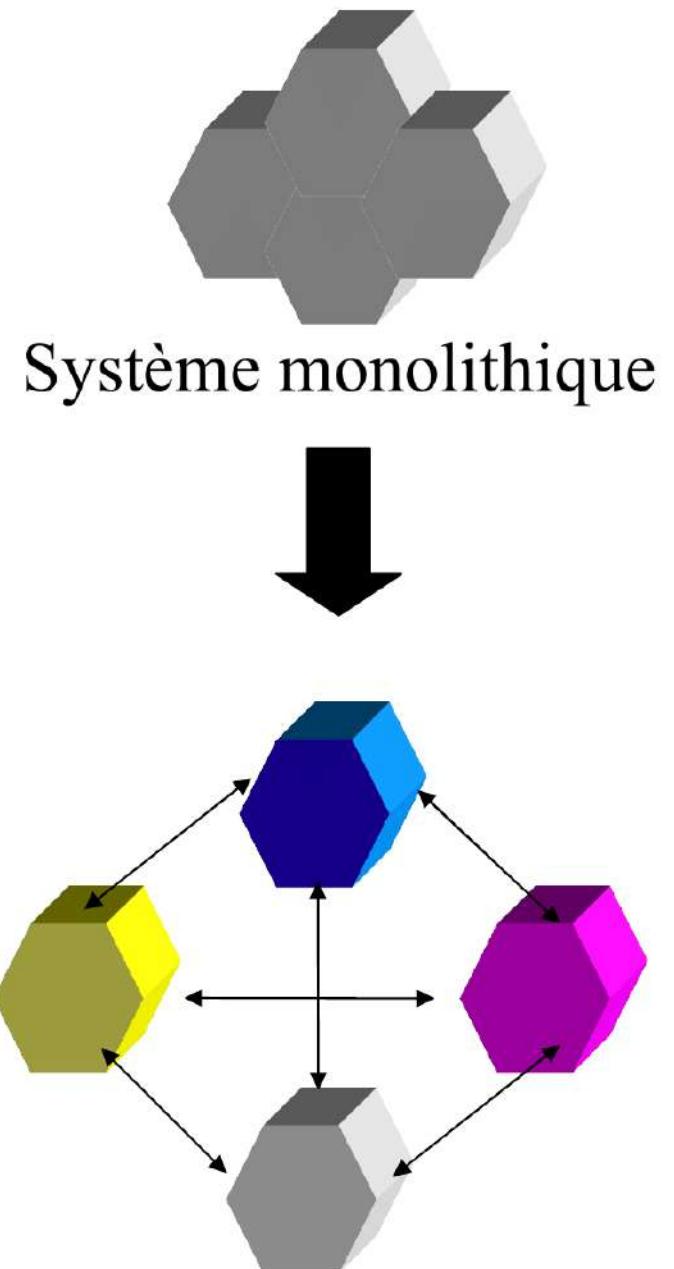
---

- ◆ **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 système doit garantir que le paiement et la livraison du produit doivent être réalisés ensemble ou rien

# Qu'est ce qu'un système distribué ?

---

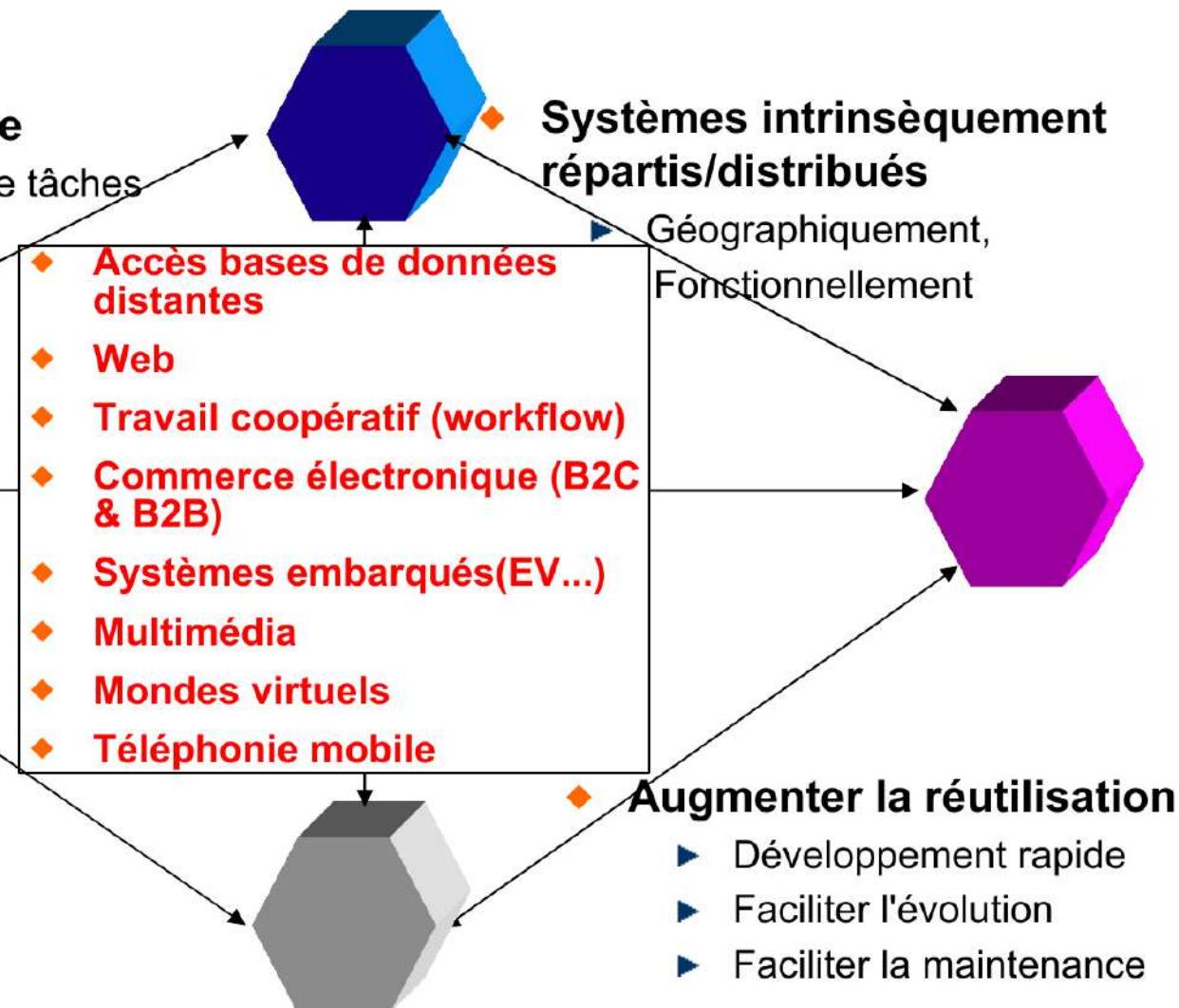
- ◆ Un système distribué est constitué d'un **ensemble de composants** logiciels ou matériels (ordinateurs, organes d'entrée-sortie, processeurs spécialisés, dispositifs de commande ou de mesure, etc.), **interconnectés par un réseau de communication**.
- ◆ Les composants du système distribué ne fonctionnent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



# Qu'est ce qui nous pousse à distribuer?

- ◆ **Augmenter la performance**

- ▶ Traitement en parallèle de tâches
- ▶ Tolérance aux pannes
- ▶ Passage à l'échelle.



# Différentes formes de distribution

---

## ◆ Distribution physique

- ▶ fonctionnement en réseaux (LANs, WANs)
- ▶ architectures multi-processeurs

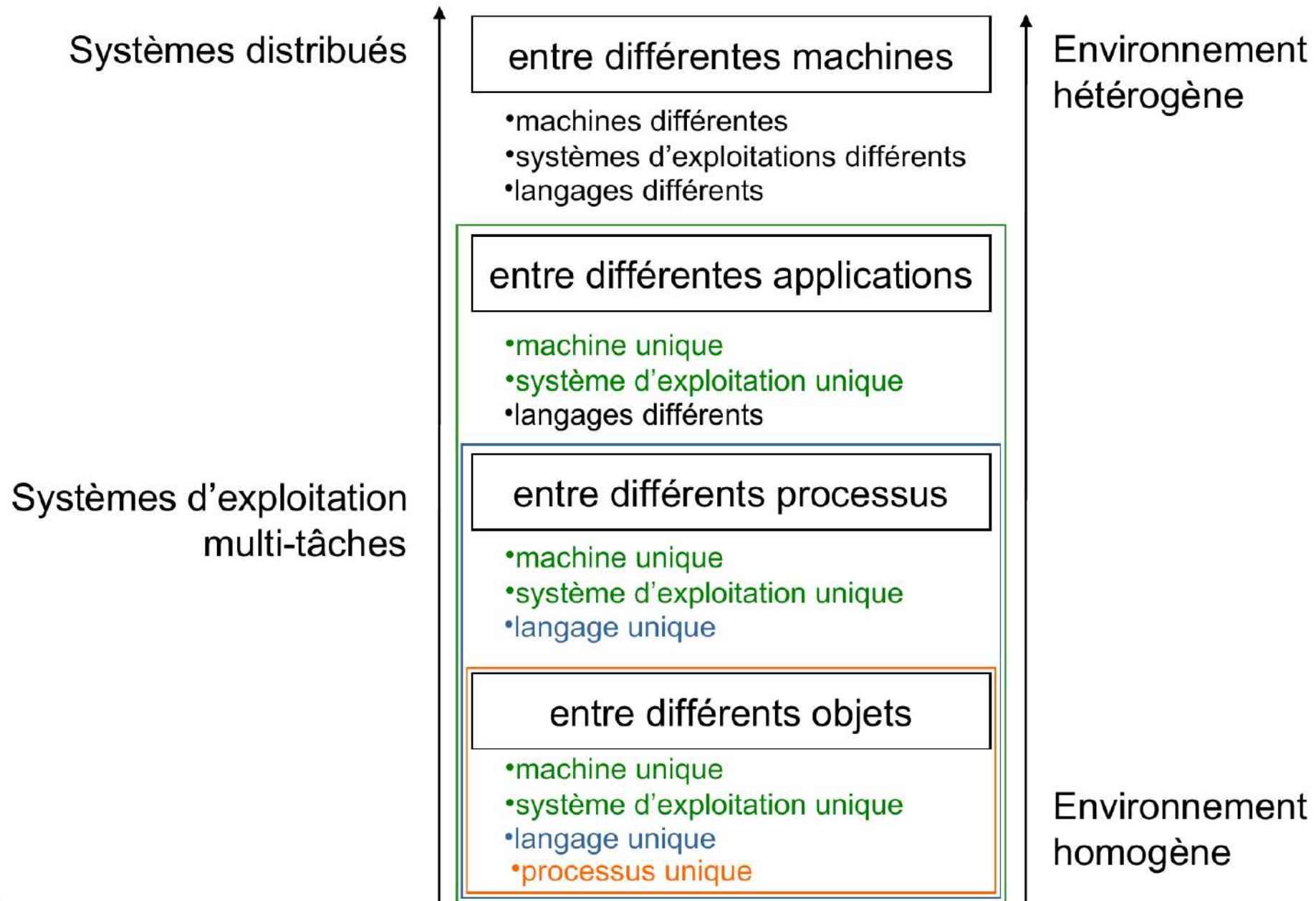
## ◆ Distribution structurelle

- ▶ programmation structurée (procédures/fonctions)
- ▶ 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 (pas totalement) et coopératifs qui apparaissent à l'utilisateur comme un seul et unique système cohérent*
- ▶ contraire = *système monolithique*
- ▶ Partenaires partiellement indépendants et non totalement indépendants

# Caractéristiques des systèmes distribués

---

- ◆ **Hétérogénéité**
- ◆ **Ouverture (Évolutivité)**
- ◆ **Sécurité**
- ◆ **Scalabilité et RéPLICATION**
- ◆ **Tolérance aux pannes (Disponibilité)**
- ◆ **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'hétérogénéité des systèmes
- ▶ ex. : standards pour l'Internet TCP/IP

### ◆ utilisation de middlewares

- ▶ modèles de programmation distribuée standardisés et uniformes  accès aux BDs, appel de méthodes d'objets à distance
- ▶ implémentés au-dessus des standards d'Internet
- ▶ ex. : gRPC, Java RMI, CORBA, 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 et Évolutivité

---

### ◆ Possibilité d'extension ou de ré-implantation

- ▶ ajout et mise à disposition 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 des 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 avec systèmes de cryptage
- ▶ déni de services
- ▶ sécurité des codes mobiles

## Scalabilité et RéPLICATION

---

*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 nombre 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 goulets d'étranglement
  - décentralisation nécessaire

## Tolérance aux pannes et Disponibilité

---

*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 et Dispo. – 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 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 distribués : *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 (scalabilité) est une propriété importante, mais difficile à réaliser

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

# Voies d'études des systèmes distribués

---

## ◆ Approche “descriptive”

- Étude des divers modèles de conception et de construction d'applications distribuées (client-serveur, événements et messages, objets distribués, composants distribués, 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

## ◆ Approche “fondamentale”

- Étude des principes de base des systèmes distribués ; 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**

## Aperçus sur les problèmes fondamentaux

---

- ◆ Comment décrire une exécution distribuée ?
- ◆ 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 ?