Introduction

- Système distribué en opposition à système centralisé
- Système centralisé : tout est localisé sur la même machine et accessible par le programme
- Système logiciel s'exécutant sur une seule machine
- Accédant localement aux ressources nécessaires (données, code, périphériques, mémoire ...)
- Système distribué : une définition parmi d'autres :
- Ensemble d'ordinateurs indépendants connectés en réseau et communiquant via ce réseau
- Cet ensemble apparaît du point de vue de l'utilisateur comme une entité unique
- Vision matérielle d'un système distribué : architecture matérielle

Introduction

- Machine multi-processeurs avec mémoire partagée
- Cluster d'ordinateurs dédiés au calcul/traitement massif parallèle
- Ordinateurs standards connectés en réseau
- Vision logicielle d'un système distribué
- Système logiciel composé de plusieurs entités s'exécutant indépendamment et en parallèle sur un ensemble d'ordinateurs connectés en réseau

Pourquoi des SDs ?

Les systèmes répartis sont populaires pour plusieurs raisons:

- Aspects économiques (rapport prix/performance)
- Adaptation de la structure d'un système à celle des applications (géographique ou fonctionnelle)
- Besoin d'intégration (applications existantes)
- Besoin de communication et de partage d'information
- Réalisation de systèmes à haute disponibilité
- Partage de ressources (programmes, données, services)
- Réalisation de systèmes à grande capacité d'évolution

Pourquoi des SDs ?

- Accès distant: un même service peut être utilisé par plusieurs acteurs, situés à des endroits différents
- Redondance: des systèmes redondants permettent de pallier une faute matérielle, ou de choisir le service équivalent avec le temps de réponse le plus court
- Performance: la mise en commun de plusieurs unités de calcul permet d'effectuer des calculs parallélisables en des temps plus courts
- Confidentialité: les données brutes ne sont pas disponibles partout au même moment, seules certaines vues sont exportées

Qu'est-ce qu'un SD?

➢ Un système réparti (ou distribué, de «distributed system»), est:

- composé de plusieurs systèmes calculatoires autonomes (sinon, non réparti)
- sans mémoire physique commune (sinon c'est un système parallèle, cas dégénéré)
- qui communiquent par l'intermédiaire d'un réseau (quelconque)

Définitions

Définition [Lamport]

• ***A distributed system is one on which I can't do my work some computer has failed that I never heard of*.**

Un système réparti est un système qui vous empêche de travailler quand une machine dont vous n'avez jamais entendu parler tombe en panne.

Définitions

Un système distribué est un ensemble d'entités autonomes de calcul (ordinateurs, PDA, processeurs, processus, processus léger etc.) interconnectées et qui peuvent communiquer.

• Exemples:
– réseau physique de machines
– Un logiciel avec plusieurs processus sur une même machine.

Définitions

Définition générale

– C'est un ensemble de processus communiquants, répartis sur un réseau de machines le plus souvent hétérogènes, et coopérant à la résolution d'un problème commun.

« A distributed system is a collection of independent computers that appear to the users of the system as a single computer. » [A. Tanenbaum, Prentice-Hall, 1994]

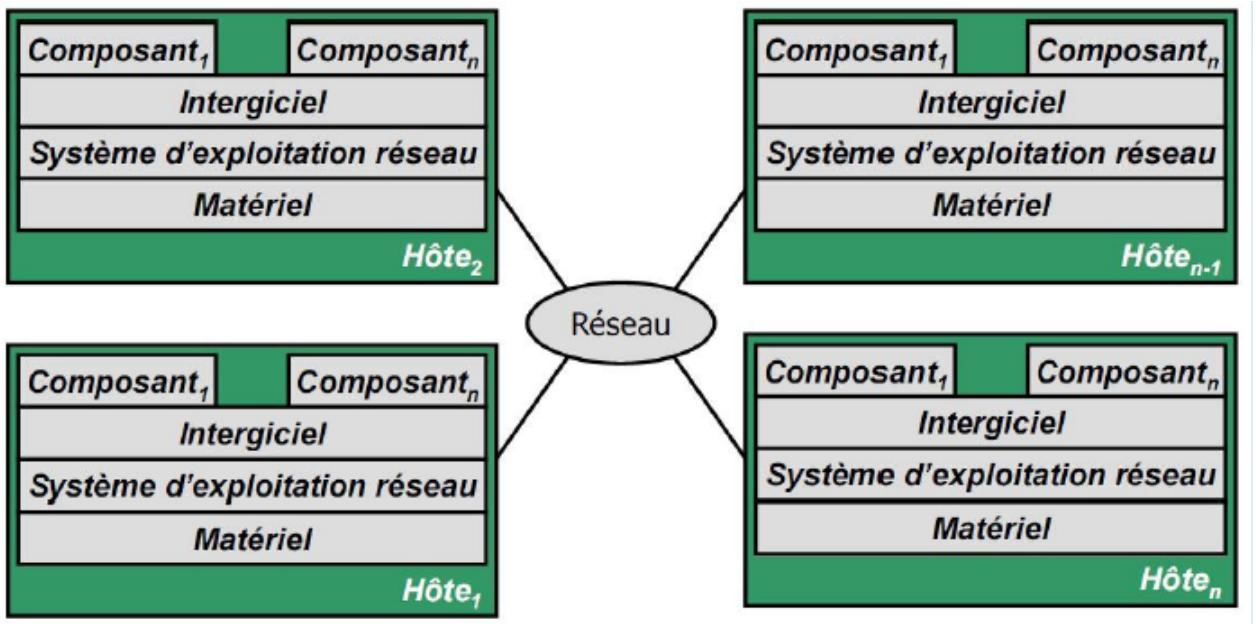
Définitions

Définition [Tanenbaum]: *Un ensemble d'ordinateurs indépendants qui apparaît à un utilisateur comme un système unique et cohérent*

- Les machines sont autonomes
- Les utilisateurs ont l'impression d'utiliser un seul système.

Plusieurs composantes réparties sur plusieurs machines et interconnectés grâce à un intergiciel (Middleware)

Définitions



- En résumé... Systèmes Répartis :

– Les systèmes répartis, c'est régler à plusieurs des problèmes qu'on n'aurait pas eus tout seul !

Intérêts des SDs

- Utiliser et partager des ressources distantes
- Système de fichiers : utiliser ses fichiers à partir de n'importe quelle machine
- Imprimante : partagée entre toutes les machines
- Optimiser l'utilisation des ressources disponibles
- Calculs scientifiques distribués sur un ensemble de machines
- Système plus robuste
- Duplication pour fiabilité : deux serveurs de fichiers dupliqués, avec sauvegarde
- Plusieurs éléments identiques pour résister à la montée en charge ...

Inconvénients/points faibles

- Si problème au niveau du réseau
- Le système marche mal ou plus du tout
- Bien souvent, un élément est central au fonctionnement du système : serveur
- Si serveur plante : plus rien ne fonctionne
- Goulot potentiel d'étranglement si débit d'information très important
- Gestion du système totalement décentralisée et distribuée
- Nécessite la mise en place d'algorithmes +/- complexes

Domaines d'application des SDs

- CFAO, Ingénierie simultanée
  - Coopération d'équipes pour la conception d'un produit
  - Production coopérative de documents
  - Partage cohérent d'information
- Gestion intégrée des informations d'une entreprise
  - Intégration de l'existant
- Contrôle et organisation d'activités en temps réel
- Centres de documentation, bibliothèques
  - Recherche, navigation, visualisation multimédia
- Systèmes d'aide à la formation

Exemples de SDs

➢ On rencontre des systèmes répartis dans la vie tous les jours:

- WWW, FTP, Mail
- Guichet de banque, agence de voyage
- Téléphones portables (et bornes)
- Télévision interactive
- Agents intelligents

• Contrôle du trafic aérien
• Banques
• Système de fichier distribué
• DNS
• Systèmes Pair-à-pair (P2P)

Exemples de SDs

 Serveur de fichiers

Accès aux fichiers de l'utilisateur quelque soit la machine utilisée

 Un serveur de fichier

 Physiquement : fichiers se trouvent uniquement sur le serveur

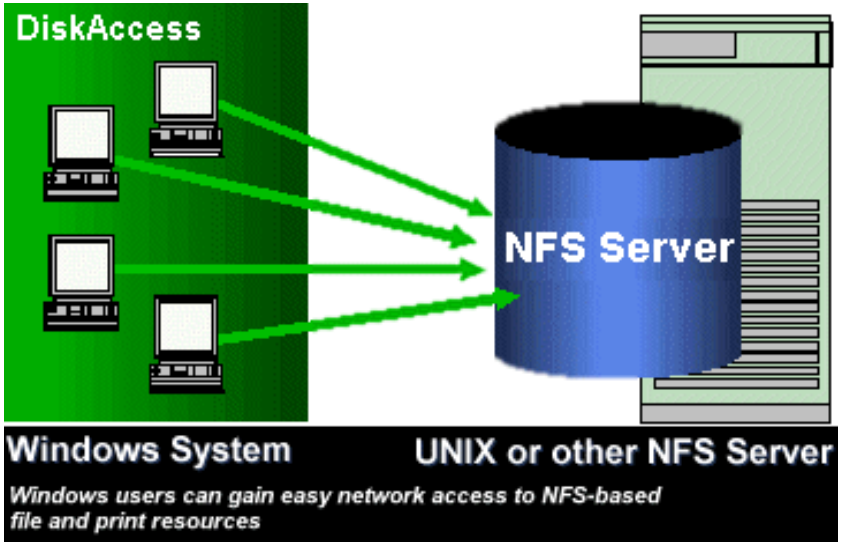
 Virtuellement : accès à ces fichiers à partir de n'importe quelle machine cliente en faisant « croire » que ces fichiers sont stockés localement

 Arborescence de fichiers Unix : arborescence unique avec :

 Répertoires physiquement locaux

Exemples de SDs

 Répertoires distants montés via le protocole NFS (Network File System)



Intérêts :

 Accès aux fichiers à partir de n'importe quelle machine

 Système de sauvegarde associé à ce serveur

Exemples de SDs

 Transparent pour l'utilisateur

Inconvénients :

 Si réseau ou le serveur plante : plus d'accès aux fichiers

WWW :

 Un serveur web auquel se connecte un nombre quelconque de navigateurs web (clients)

 Accès à distance à de l'information

 Accès simple

 Serveur renvoie une page HTML statique qu'il stocke localement

Exemples de SDs

Traitement plus complexe

- Serveur interroge une base de données pour générer dynamiquement le contenu de la page
- Transparent pour l'utilisateur : les informations s'affichent dans son navigateur quelque soit la façon dont le serveur les génère.
- Calculs scientifiques :
- Plusieurs architectures matérielles généralement utilisées
- Ensemble de machines identiques reliées entre elles par un réseau dédié et très rapide (cluster)
- Ensemble de machines hétérogènes connectées via un réseau local ou bien encore par Internet (grille)

Exemples de SDs

Grille de calcul

Une **grille de calcul** permet de faire du calcul distribué : elle exploite la puissance de calcul (processeurs, mémoires, ...) de milliers d'ordinateurs afin de donner l'illusion d'un ordinateur virtuel très puissant. Ce modèle permet de résoudre d'importants problèmes de calcul nécessitant des temps d'exécution très longs en environnement "classique".

- Un (ou des) serveur distribue des calculs aux machines clients
- Un client exécute son calcul puis renvoie le résultat au serveur

Exemples de SDs

Avantage :

- Utilisation d'un maximum de ressources de calcul

Inconvénient :

- Si réseau ou serveur plante, le système s’arrête

Exemples de SDs

Le domaine industriel

- Le calcul d'images (Films d'animation...),
- La simulation d'utilisation de matériaux (**Airbus**),
- La simulation d'utilisation de réseaux électriques (**EDF**),
- La simulation d'activités pétrolifères (**Institut Français du Pétrole**),
- Les simulations bancaires (Banques, intermédiaires boursiers),
- Les simulations militaires (**EADS**)

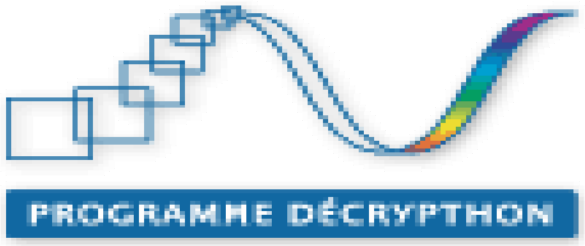
Exemples de SDs

Le domaine scientifique

Rechercher d'intelligence extraterrestre

Etudie les interactions entre le génome et les protéines

Etablir un lien entre des laboratoires de recherche qui ont besoin d'une énorme puissance de calcul



Réalisation de la Cartographie du protéome

L'étude des interactions protéine-protéine

Autres exemples de SDs

PVM

- Système populaire, prévu pour le calcul scientifique
- Basé sur l'envoi de message
- Supporte les messages de groupe
- Nécessite des appels explicites aux routines d'emballage et de déballage (pas de contrôle de type)

DCOM

- Objets répartis de Microsoft
- Concurrent direct de CORBA

RMI

- Objets répartis de Java
- Supporte la migration de code
- Doit devenir compatible avec CORBA

Autres exemples de SDs

Erlang

- Langage développé par Ericsson pour développer des autocommutateurs
- Basé sur l'envoi de messages typés entre processus, sans distinction de localité
- Transforme tout événement administratif (mort d'un processus, etc.) en message
- Supporte la migration de code

Autres exemples de SDs

XML-RPC

- Utilise des technologies normalisées:
  - XML
  - HTTP
- Bénéficie d'un niveau de sécurité normalisé
  - HTTPS (SSL+X509)

SOAP

- Basé sur XML-RPC
- Développé par Microsoft

Autres exemples de SDs

Les réseaux actifs

- Ne se contentent plus d'exécuter du code dans les nœuds finaux
- Transportent le code sur les routeurs qui en ont besoin (nouveaux protocoles)
- Permettent de localiser des services équivalents par l'anycast (imprimante sur un réseau par exemple)

Du centralisé vers le réparti (de Nouveaux problèmes)

Problèmes d'ordre et d'heure :

Un processus *P1* sur une station *A* veut coopérer avec un processus *P2* sur une station *B*. Comment faire pour savoir qu'un évènement ***e*** de *P1* s'est déroulé avant (ou après) un évènement ***e'*** de *P1* ?

- Absence de temps universel (ou horloge globale)
- Imprécision relative des horloges
- Grande variabilité des délais de transmission

Du centralisé vers le réparti (de Nouveaux problèmes)

Problèmes d'état et de prise de décision

À un instant donné, quel est l'état du système ?

- En centralisé l'état est représenté par des tables toujours disponibles
- En réparti, problèmes dus aux délais de transmission

Conséquence : il faut mettre de l'ordre
- Estampilles
- Anneau virtuel

Du centralisé vers le réparti (de Nouveaux problèmes)

Exemple 1:

Soit un parking. Les usagers sont en compétition pour l'utilisation des places.

- Parking avec un accès unique, surveillé par un seul gardien : connaissance partielle de la situation
- Refus d'entrer alors qu'une voiture est en route vers la sortie.
- Plusieurs accès avec un gardien à chaque accès : chaque gardien connaît avec retard les actions des autres gardiens
- 2 voitures sont autorisées à rentrer alors qu'il y a 1 seule place libre.

Du centralisé vers le réparti (de Nouveaux problèmes)

Exemple 2 : Producteur Consommateur

Le producteur *P* et le consommateur *C* sont sur des sites différents (*S1* et *S2*).

NP = nombre de productions et *NC* = nombre de consommations.

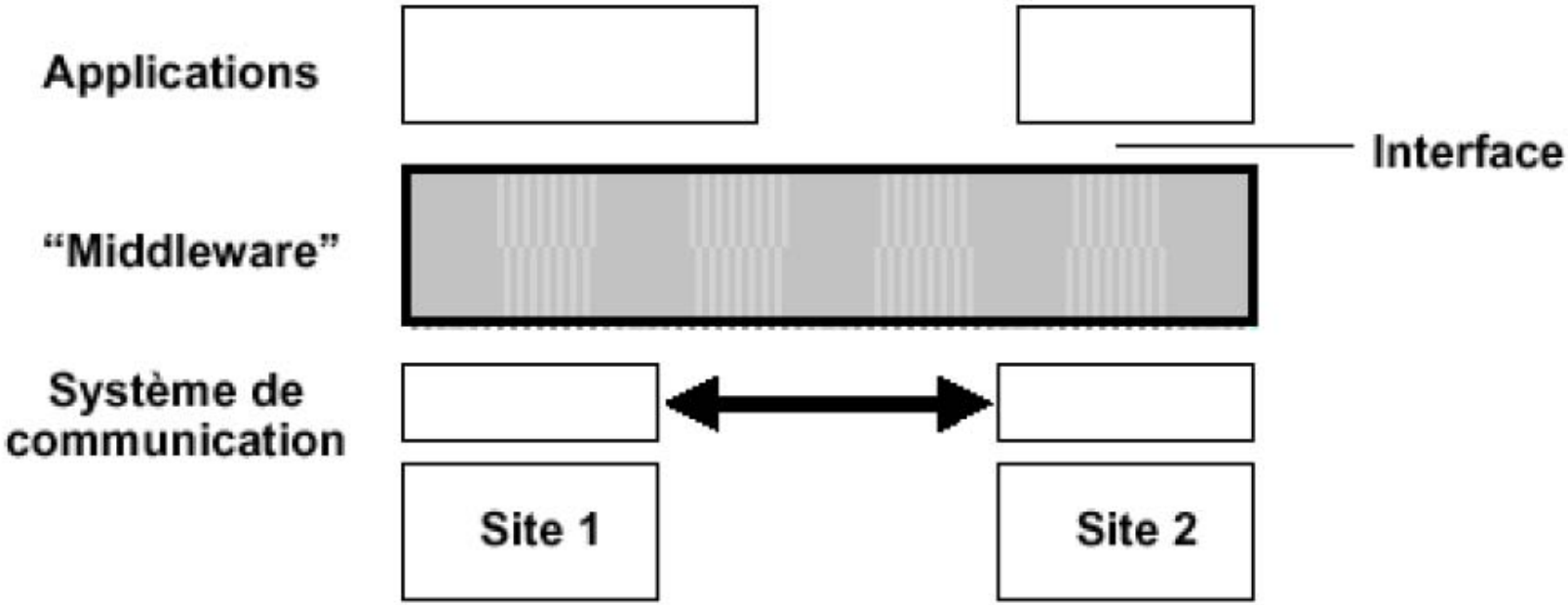
- *C* peut consommer ssi *NP-NC > 0*.
- *P* peut produire ssi *NP-NC < N*.

Du centralisé vers le réparti (de Nouveaux problèmes)

Autres problèmes :

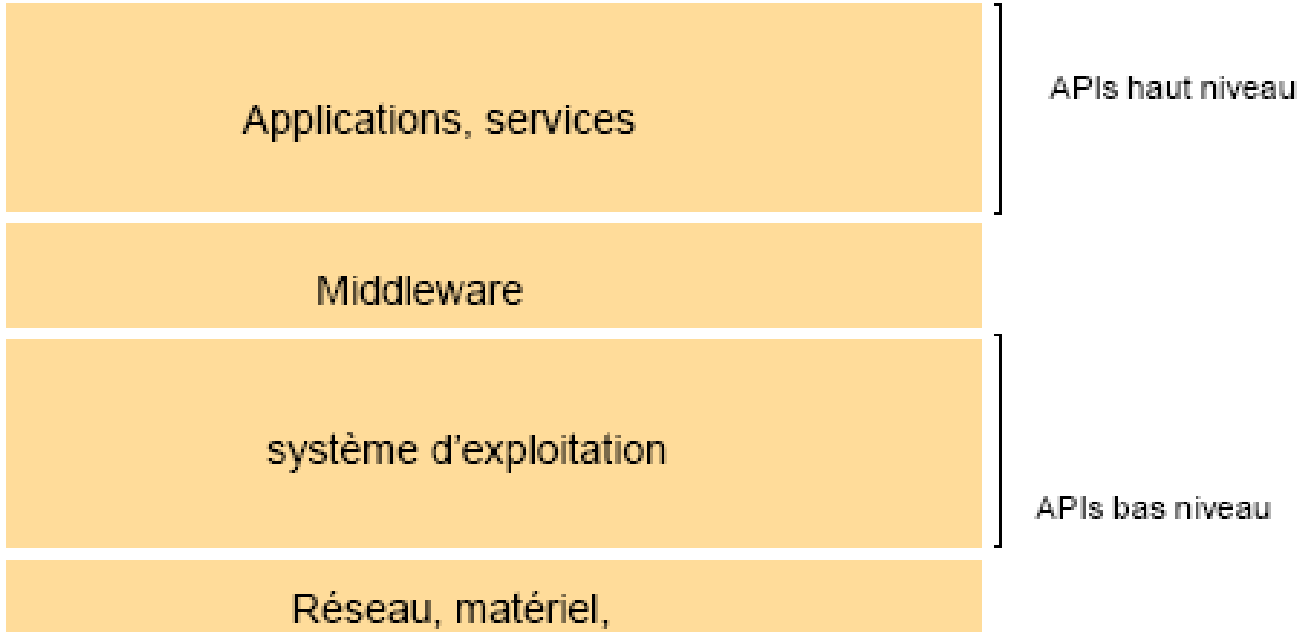
- Localisation/Nommage
- Transparence
- Interopérabilité, ...

Intergiciel “middleware”



Définition : L’intergiciel (*middleware*) est la couche logicielle située entre les couches basses (systèmes d'exploitation, protocoles de communication) et les applications dans un système informatique réparti (CORBA, EJB, COM, etc.).

Intergiciel “middleware”

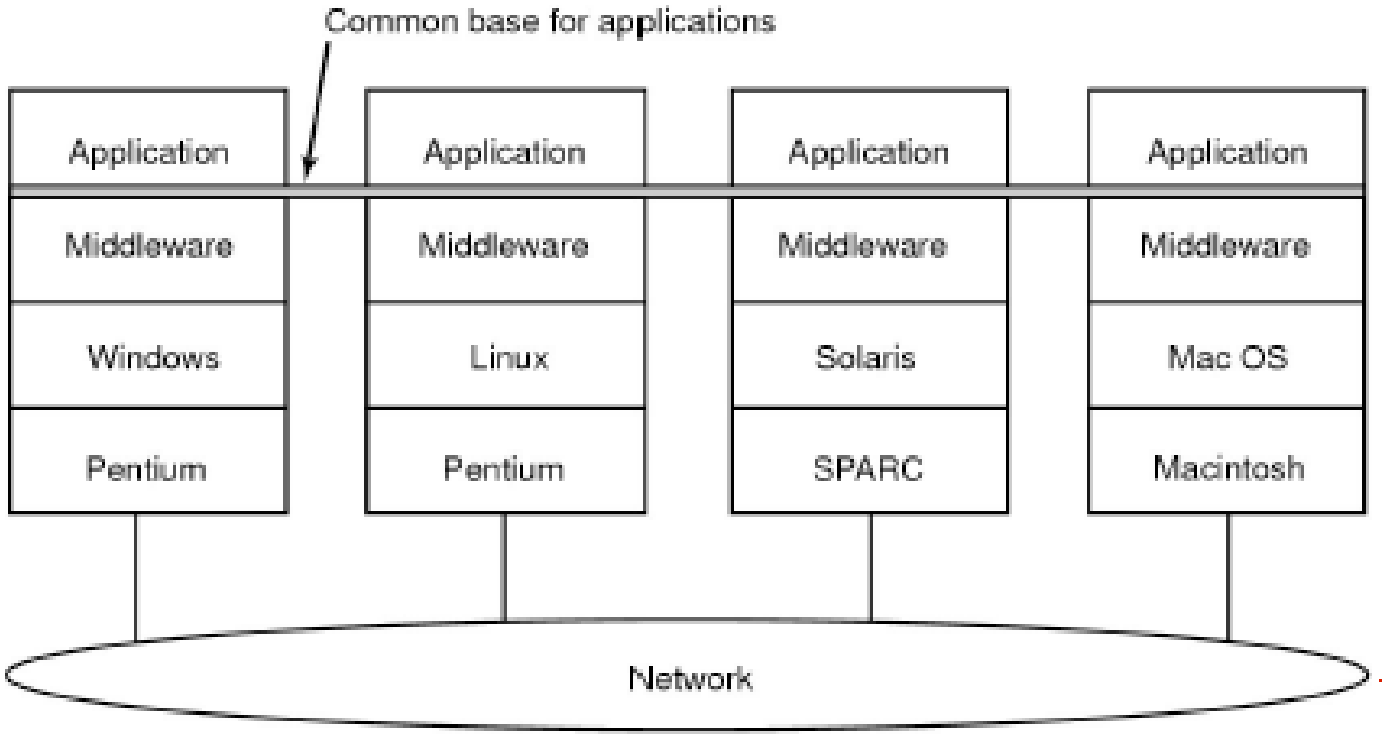


Couche logicielle intermédiaire située entre l'application et le système d'exploitation de la machine servant à concevoir/développer/déployer une application répartie.

Objectifs d'Intergiciel

- Faciliter la conception, le développement et le déploiement des applications réparties (développement, évolution, réutilisation, portabilité des applications)
- Masquer la complexité et l'hétérogénéité de l'infrastructure sous-jacente
- Fournir des services communs
- Fournir une interface ou API de haut niveau aux applications
- Rendre la répartition aussi invisible (transparente) que possible

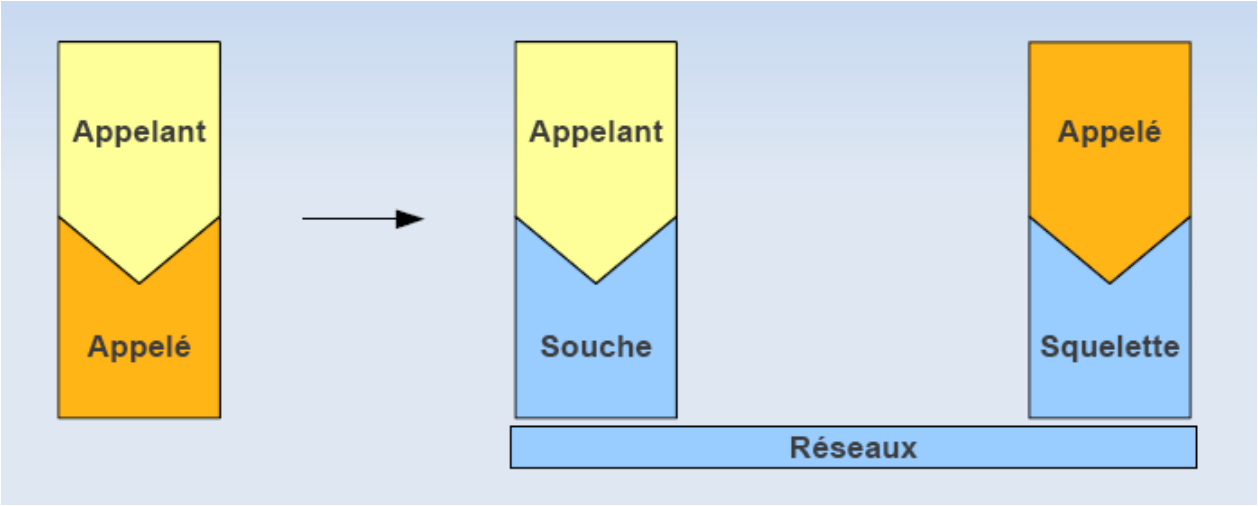
Objectifs d'Intergiciel



Les Classes d'Intergiciel

- Orientés objets distribués
  - Java RMI, CORBA, ...
- Orientés composants distribués
  - EJB, CCM, ...
- Orientés transactionnels
  - JTS de Sun, MTS de Microsoft
- Orientés messages
  - JMS de Sun, MSMQ de Microsoft, MQSeries de IBM,..
- Orientés Web
  - Web Services (XML-RPC, SOAP)
- Orientés SGBD / SQL
  - ODBC (Open DataBase Connectivity), JDBC de Sun

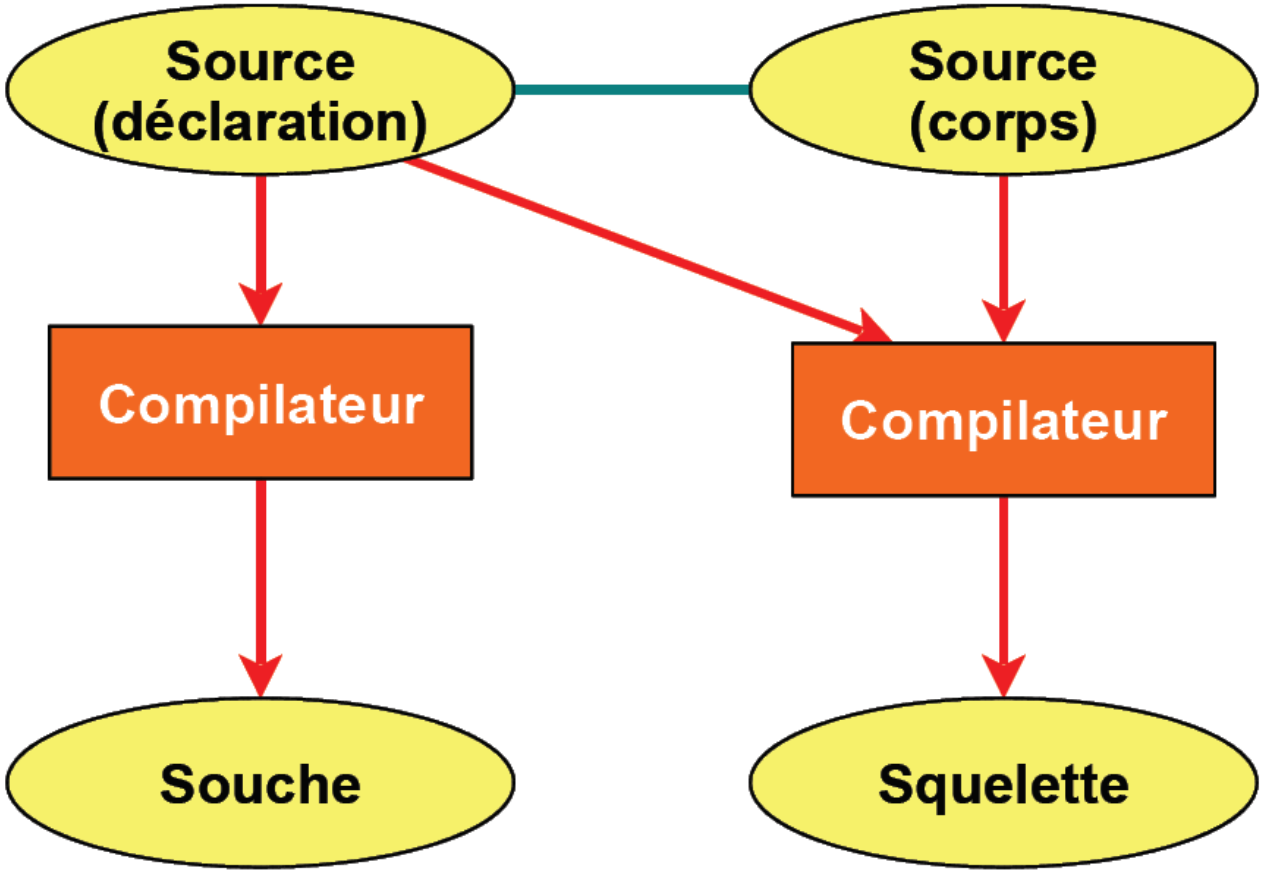
Souches et squelettes



- A partir d'une spécification d'interface, on peut générer une souche
  - Emballage des paramètres et envoi de la requête
  - Attente et déballage du résultat
- On peut également générer un squelette
  - Réception de la requête et déballage des arguments
  - Appel du sous-programme réel

Souches et squelettes

- Emballage et renvoi du résultat



Particularités des SDs

- Transparence
- Tolérance aux pannes
- Fiabilité
- Sécurité
- Disponibilité
- Extensibilité
- Scalability
- Ouverture

Transparence

Le but est de cacher l'architecture, le fonctionnement de l'application ou du système distribué pour apparaître à l'utilisateur comme une application unique cohérente

L'ISO définit plusieurs transparences (norme RM-ODP)
Accès, localisation, concurrence, réplication, mobilité, panne, performance, échelle

Transparence d'accès :

- Accès à des ressources distantes aussi facilement que localement
- Accès aux données indépendamment de leur format de représentation

Transparence

Transparence de localisation :

Accès aux éléments/ressources indépendamment de leur localisation

Transparence de concurrence :

Exécution possible de plusieurs processus en parallèle avec utilisation de ressources partagées

Transparence de réplication : Possibilité de dupliquer certains éléments/ressources pour augmenter la fiabilité

Transparence de mobilité : Possibilité de déplacer des éléments/ressources

Transparence à l'emplacement : L'utilisateur ne connaît pas où sont situées les ressources

Transparence

Transparence à la migration : Les ressources peuvent être déplacées sans modification de leur nom

Transparence au parallélisme : Des tâches peuvent s'exécuter en parallèle sans que les utilisateurs le sachent

Transparence de panne : Doit supporter qu'un ou plusieurs éléments tombent en panne

Transparence de performance :

Possibilité de reconfigurer le système pour en augmenter les performances

Transparence d'échelle : Doit supporter l'augmentation de la taille du système (nombre d'éléments, de ressources ...)

Tolérance aux pannes

Pannes : Pannes franches ; Pannes byzantines

Un SD doit être conçu pour masquer les pannes aux utilisateurs. La panne de certains serveurs (ou leur réintégration dans le système après la réparation) ne doit pas perturber l'utilisation du système en terme de fonctionnalité.

Détection de pannes : Difficulté et même impossibilité de détection pour certains systèmes, suspicion de machines ; e.g. connexion par un navigateur à un serveur distant qui ne répond pas !!

Tolérance aux pannes

Correction d'erreurs : De fichiers/messages corrompus.

Reprise sur pannes : Techniques de journalisation dans les BD (éventuellement le système est dégradé)

Problèmes :

- La communication est en général assez lente dans les SDs

- Le mécanisme de tolérance aux pannes requiert beaucoup d'opérations "inutiles" si pas de panne !

Tolérance aux pannes

Solutions :

Minimiser les échanges de message par exemple en faisant en sorte de :

- Maximiser des granules grosses (gros grains) et éviter des grains fins pour les calculs à distance

- Réduire le champ d'application des mécanismes de reprises sur pannes (car cela engendre des transmissions d'information, et donc, les réserver aux parties critiques)

Tolérance aux pannes

<u>Exemple transaction bancaire :</u>

- Compte A donne ordre de virement de X$_{DA}$ vers le compte B
- X ajouté à B
- X retiré de A

<u>Atomicité :</u> Tout faire ou rien

Mécanisme permettant de faire un rollback

Fiabilité des SDs

 Nombreux points de pannes ou de problèmes potentiels

Réseau

- Une partie du réseau peut-être inaccessible
- Les temps de communication peuvent varier considérablement selon la charge du réseau
- Le réseau peut perdre des données transmises

Machine

- Une ou plusieurs machines peut planter, engendrant une paralysie partielle ou totale du système
- Peut augmenter la fiabilité par redondance, duplication de certains éléments, mais rend plus complexe la gestion du système

Sécurité des SDs

 Nature d'un système distribué fait qu'il est beaucoup plus sujet à des attaques
 Communications à travers le réseau peuvent être interceptées
 On ne connaît pas toujours avec qui on communique

Les ressources doivent être protégées contre des utilisations abusives et malveillantes. En particulier le problème de piratage des données sur le réseau de communication

Solutions :

 Connexions authentifiées et sécurisées
 Cryptage de la communication sur le réseau
 Protection contre les falsifications et les corruptions

Disponibilité / Extensibilité

La disponibilité : Est le temps pendant lequel le système est utilisable.

Pour améliorer la disponibilité :

- Limiter le nombre des composants critiques

- Dupliquer les parties clés des composants logiciels et matériels (redondance) ; Mais on doit savoir maintenir la cohérence des copies

Extensibilité*:* Ajout/MAJ de composants sans en affecter les autres)

Scalability / Ouverture

Mise à l 'échelle (scalability) : Un SD doit fonctionner efficacement à différentes échelles:

- Deux postes de travail et un serveur de fichiers ;

- Réseau local avec plusieurs centaines de postes de travail et serveurs de fichiers ;

- Plusieurs réseaux locaux reliés pour former un Internet

Ouverture : Services offerts selon des règles standards qui décrivent la syntaxe et la sémantique de ces services (Interfaces publiées, ex. IDL)

Défis des SDs

- Comment modéliser et exprimer les SDs ? Comment les prouver ?
- Comment les analyser, calculer leur complexité, les classifier etc…?
- Pas de connaissance de l'état global !
- Non déterminisme (lié souvent au problème du synchronisme) !
- Pas de modèle universel et standard pour les SDs
- Réutiliser les codes patrimoniaux ("legacy code") !
- Hétérogénéité :

  Différents vendeurs, différents langages, différents protocoles, différentes machines, ... !

Niveaux d'abstraction des SDs

- *La couche vue, ou présentation :* encore appelée IHM.

- Permet l'interaction de l'application avec l'utilisateur (Gère les actions de la souris, du clavier, de l'écran, la présentation de l'information etc.).

- On tente au maximum de rendre cette couche accessible et ergonomique.

- La logique application, les traitements : décrivent les tâches à réaliser par l'application. On peut les découper en 2 sous-parties ;

- Les traitements locaux : concernent les traitements du dialogue avec l'IHM, principalement pour faciliter leur manipulation ; et

Niveaux d'abstraction des SDs

- Les traitements globaux : également appelées *Business Logic* (logique applicative) concernent l'application elle-même ; Ils contiennent les règles élémentaires et internes qui régissent une entreprise.

- Les données : ou plus particulièrement, l'accès aux données, regroupent l'ensemble des mécanismes permettant la gestion des informations stockées par ou pour l'application.

Niveaux d'abstraction des SDs

- On imagine bien que ces trois niveaux peuvent être imbriqués ou répartis entre différentes machines physiques et/ou virtuelles.

- Le noyau de l'application est constitué de la couche présentation et de la logique application, l'accès aux données étant délégué.

- Le découpage et la répartition de ce noyau permettent de distinguer des modes d'interactions (*Approches des SDs*) et des architectures applicatives qui vont de l'architecture un-tiers à *n*-tiers

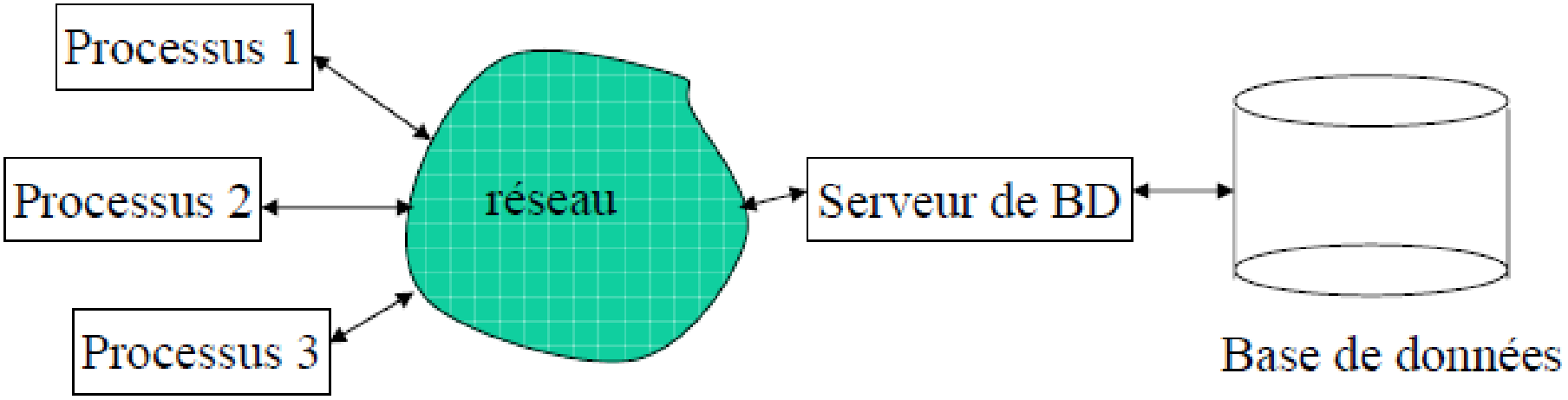
Approches des SDs

- Approches centrées sur les données
- Approches centrées sur les messages
- Approches centrées sur les procédures
- Approches centrées sur les objets

NB Cette classification est artificielle du point de vue des produits sur le marché (qui traversent les frontières de cette classification), mais elle permet de comprendre le domaine.

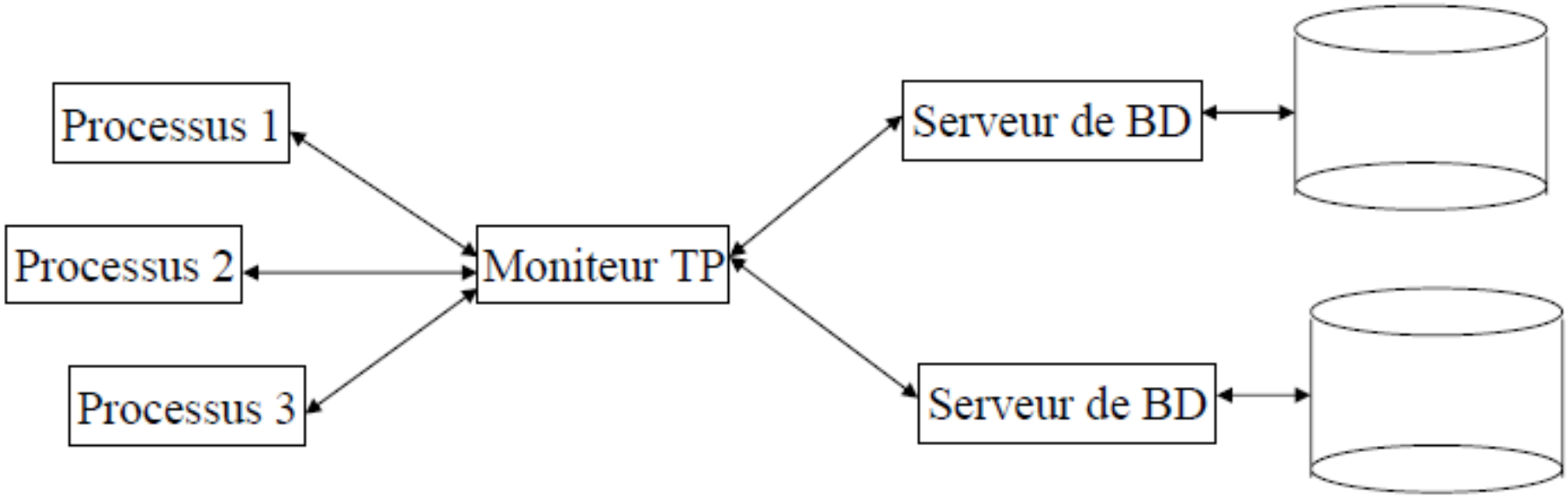
Approches centrées sur les données

- Les différentes parties d'une application répartie communiquent par l'intermédiaire d'une base de données.
- Solution la plus simple, adéquate pour certaines applications réparties.



Approches centrées sur les données

• Plusieurs bases de données: un logiciel appelé moniteur transactionnel (TP monitor) peut servir d'intermédiaire entre les processus-clients et le serveur de bases de données:



Approches centrées sur les données

Avantage du moniteur TP:

- Cache au processus clients les spécificités des différentes bases de données.
- Permet aux processus clients d'exécuter des transactions qui accèdent plusieurs bases de données.

Les propriétés transactionnelles (Atomicité, Consistance, Isolation, Durabilité) sont assurées par le moniteur TP.

- Nombreux produits sur le marché: CICS (IBM), IMS (IBM), Tuxedo (BEA), Encina (Transarc), MTS (Microsoft)

Approches centrées sur les messages (MOM)

• Les différentes parties d'une application répartie communiquent par échange de messages

• On distingue deux types de MOM (Message Oriented Middleware)

– MOM basés sur des *queues de messages*

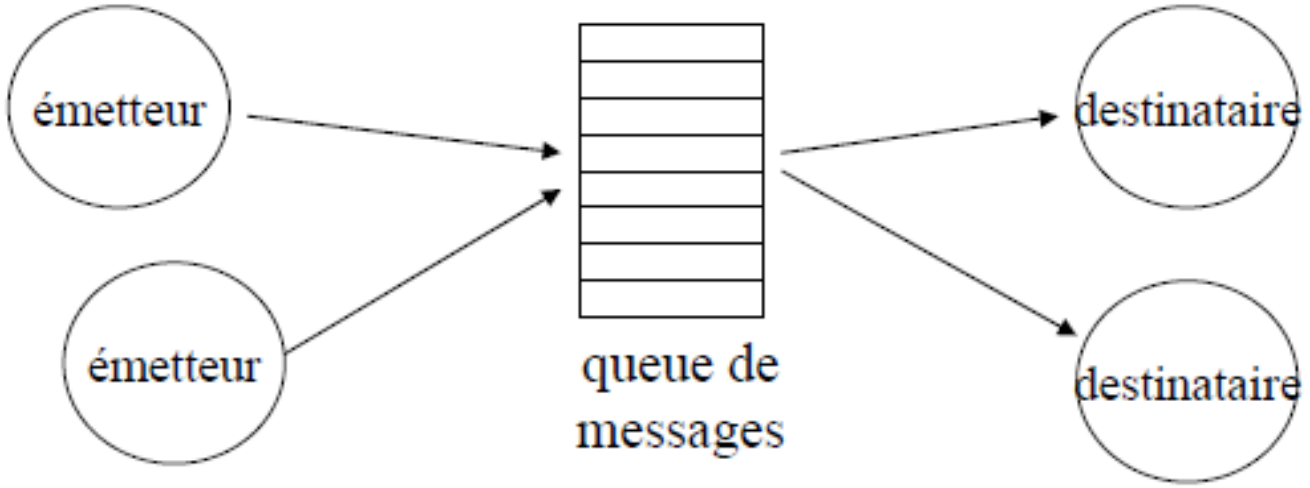
– MOM basés sur le paradigme *publish/subscribe*

• Java Messaging Service (JMS) est une API Java de SUN qui offre les interfaces pour les queues de message et publish/subscribe

MOM: Queues de messages

• La commication entre émetteur du message et destinataire du message utilise une queue de messages gérée par un processus-tiers (schéma producteur-consommateur).

Avantage: la communication ne nécessite pas que l'émetteur du message et le destinataire soient en ligne simultanément.



MOM: Queues de messages

- Nombreux produits sur le marché.
- Les queues de messages sont souvent offertes dans le contexte de moniteurs TP.
- Produits indépendants de moniteurs TP: MQSeries (IBM), messageQ (DEC-Compaq-HP), MSMQ (Microsoft), RTR (DEC-Compaq-HP)

MOM: Publish/Subscribe

• Basé sur le modèle des *news groups*:

– Un processus peut s'abonner à une liste de sujets.

– Un processus peut publier un message correspondant à un sujet.

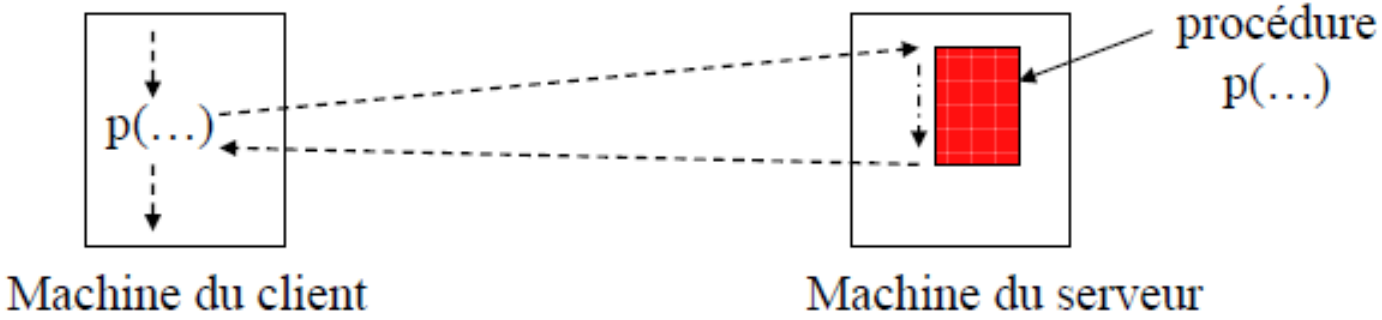
Le message sera reçu par tous les processus abonnés au sujet.

• Marché en pleine expansion

• Produits: SmartSockets (Talarian), TIBCO (TIBCO Software), MQSeries (IBM), MSMQ (Microsoft)

Approches centrées sur les procédures

- Approche basée sur l'abstraction appelée RPC (Remote Procedure Call).
- Pour le client, similaire à un appel de procédure (mais la procédure exécutée à distance, sur le serveur).
- Communication synchrone entre client et serveur: client bloqué en attente de la réponse (MOM: communication asynchrone)
- Produits: ONC (Sun), NCA (HP), DCE (OSF)



Approches centrées sur les objets

• Généralisation de la notion de procédure distante (RPC) à la notion d'objet distant.

• Permet à un processus client d'appeler une méthode d'un objet situé sur une machine distante.

• Produits basés sur la norme CORBA, définit par l'OMG (Object Management Group) qui regroupe plus de 700 compagnies (y compris Microsoft).

NB : CORBA de moins en moins utilisé

• Autres produits: .NET (Microsoft), Java RMI.

Architecture des SDs

Architecture: détermine « *qui communique avec qui* » (dimension «orthogonale» à la classification précédente)

- Architecture un-tiers
- Architecture deux-tiers (Architecture Client-Serveur)
- Architecture trois-tiers (Client-Serveur Distribué)
- Architecture *n*-tiers (Multi-tiers)

Architecture un-tiers

*- Dans un contexte **simple-utilisateur :***

Les trois couches sont fortement et intimement liées, et s'exécutent sur la même machine ; on ne peut pas parler d'architecture client-serveur mais d'informatique centralisée.

*- Dans un contexte **multi-utilisateurs :***

Deux types d’applications mettant en œuvre des architectures un-tiers :

- Applications sur site central ;

- Applications réparties sur des machines indépendantes communiquant par partage de fichiers.

Architecture un-tiers

- Application sur site central (ou mainframe)

- Les utilisateurs se connectent aux applications exécutées via le serveur central à l'aide de terminaux passifs.

- On délaisse l'intégralité des traitements au serveur central, y compris l'affichage qui se voit simplement déporté sur des terminaux passifs.

Avantages :

- Facilité d'administration et haute disponibilité.

- Utilisation optimale des ressources.

Architecture un-tiers

- Application un-tiers déployée

- Application un-tiers sur plusieurs machines indépendantes.

- Répond aux besoins d'une seule personne isolée ou d'un petit groupe de personne.

- Une mise en œuvre dans un environnement multi-utilisateurs est envisageable.

-L'envoie intégral des données nécessaires à l'exécution d'une requête.

- On arrivera toujours à une saturation du réseau.

Architecture un-tiers

- Des conflits lors de la consultation ou de la modification simultanée qui peuvent altérer l'intégrité des données.

- Ce type de solution est donc à réserver à des applications non critiques exploitées par de petits groupes de travail.

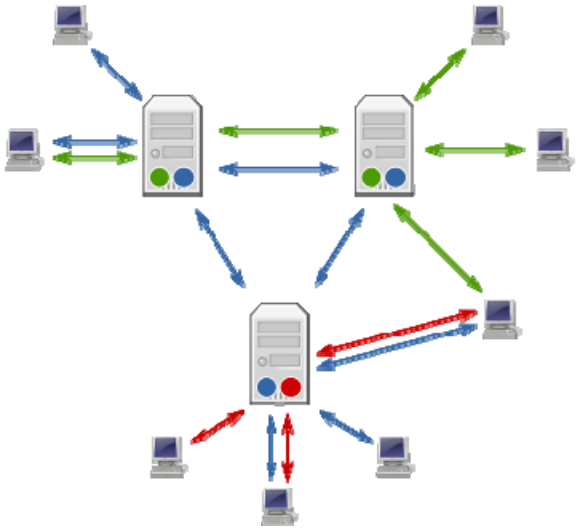
Architecture deux-tiers (Client-Serveur)

- Dans une architecture deux-tiers - encore appelée client-serveur de première génération ou client-serveur de données - le poste client se contente de déléguer la gestion des données à un service spécialisé.

- Ce type d'application permet d'utiliser toute la puissance disponible des ordinateurs présents sur le réseau et permet de fournir à l'utilisateur une interface riche, tout en garantissant la cohérence des données, qui restent gérées de façon centralisée.

- La gestion des données étant prise en charge par un SGBD centralisé, sur un serveur dédié. On interroge ce serveur à travers un langage de requête ; le plus courant étant SQL.

Architecture deux-tiers (Client-Serveur)



Les points rouge/vert/bleu correspondent à des applications hébergées sur différents serveurs

→ 2 rôles distincts ***Client*** et ***Serveur***.

Client :

Machine avec processus client demande que des requêtes ou des services lui soient rendus (consommateur d'un service).

Architecture deux-tiers (Client-Serveur)

- Il est actif le premier (ou maître) ;

- Il envoie des requêtes au serveur ;

- Il attend et reçoit les réponses du serveur.

Serveur :

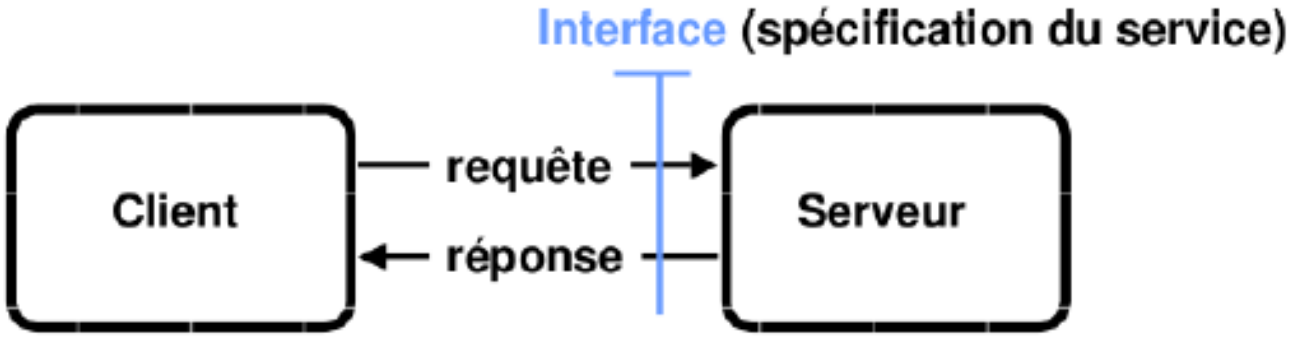
Machine avec processus serveur (par exemple base de données) répond aux requêtes des clients (fournisseur de service).

- Il est initialement passif (ou esclave, en attente d'une requête) ;

- Il est à l'écoute, prêt à répondre aux requêtes envoyées par des clients ;

Architecture deux-tiers (Client-Serveur)

- Dès qu'une requête lui parvient, il la traite et envoie une réponse.



Exemple : Serveur Web

- Client : navigateur Web de l'utilisateur
- Requêtes : récupérer le contenu d'une page HTML gérée ou générée par le serveur

Exemples : Client-Serveur

- Consultation des pages Web (HTTP)
- Envoi & réception des e-mails (SMTP, POP, IMAP)
- X Window : serveur muni écran (protocole X basé sur IP)
- NFS (protocole basé sur Sun RPC)

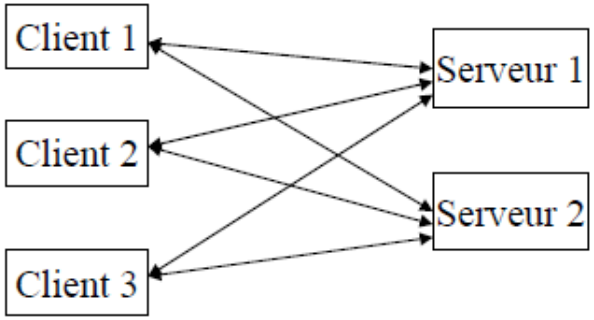
Particularités du modèle Client-Serveur

- Liens forts entre le client et le serveur ;
- Un client peut aussi jouer le rôle de serveur (et vice-versa) dans une autre interaction ;
- Le serveur tourne en permanence, attendant des requêtes et peut répondre à plusieurs clients en même temps ;
- Nécessité de machine robuste et rapide, qui fonctionne 24h/24 (grande mémoire, disques suffisants, sécurité des disques, …) ;
- Besoin en administrateurs réseau pour gérer les serveurs ;
- Nécessité généralement pour le client de connaître précisément le serveur (sa localisation).

Ex : URL du site Web ; DNS.

Le dialogue Client-Serveur

L'architecture Client / Serveur désigne un mode de communication entre plusieurs ordinateurs d'un réseau qui distingue un ou plusieurs clients du serveur: chaque logiciel client peut envoyer des requêtes à un serveur.



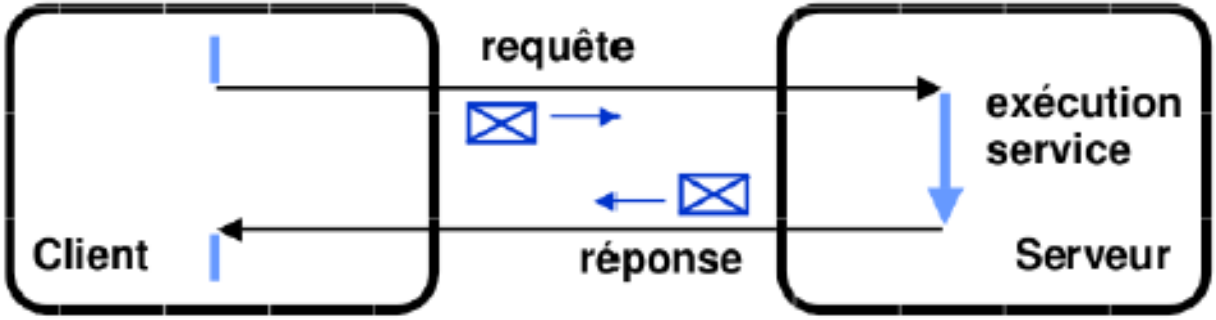
Un serveur peut être spécialisé en serveur d'applications, de fichiers, de terminaux, ou encore de messagerie électronique.

Le client et le serveur doivent bien sûr utiliser le même protocole de communication. Un serveur est généralement capable de servir plusieurs clients simultanément.

Le dialogue Client-Serveur

Client : il provoque le dialogue (l'établissement d'une conversation) afin d'obtenir des données ou un résultat de la part du serveur; il envoie une requête et reste bloqué en attente de la réponse (communication synchrone).

Serveur : il se contente seulement de répondre aux requêtes du client ; il attend une requête cliente, exécute le service demandé et renvoie une réponse au client.



Le dialogue entre le client et le serveur se résume donc à l'envoi de requêtes et aux données, en réponse.

Le dialogue Client-Serveur

Communication par échange de messages

- Message du client vers le serveur pour faire une requête

- Exécution d'un traitement par le serveur pour répondre à la requête

- Message du serveur vers le client avec le résultat de la requête

Cet échange de message transite à travers le réseau reliant les deux machines. Il met en œuvre des mécanismes relativement complexes qui sont, en général, pris en charge par un intergiciel (*middleware*).

Notion de Service

Client : programme demandant un *Service* d'un autre programme.

Serveur : programme fournissant des *Services* à d'autres programmes.

Service : Comportement d'un programme qui peut être appelé par une requête suivant un certain protocole.

Exemple :

- Le service DNS permet les traductions noms / adresses IP ;
- Requête «donne-moi la traduction de www.yahoo.fr envoyée suivant le protocole DNS.

Notion de Socket

Socket :

- Entrée réseau de la machine sur laquelle un serveur «écoute » en attendant des connexions / requêtes d'un client.

- « Tuyau » entre deux programmes

- Quintuplet : (machine1, port1, protocole transmission, port2, machine2)

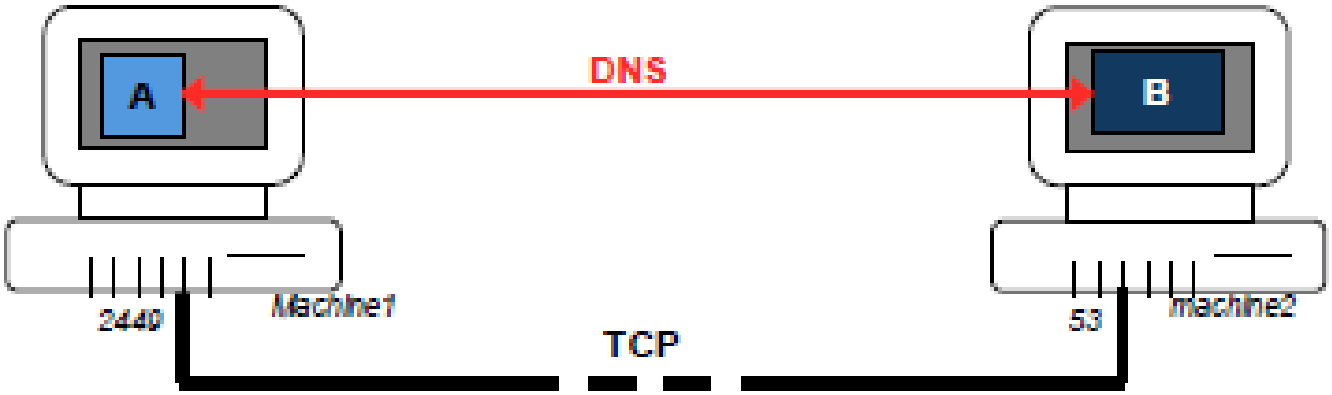
Exemple 1 : DNS

- Client sur machine 1 appelle serveur sur machine 2 / port 53.

- La connexion s'établit, le canal de communication est ouvert

Notion de Socket

- Il devient possible de communiquer suivant un protocole d'application (par exemple DNS)



- Un serveur DNS peut être un client pour un autre DNS

<u>Exemple 2 :</u> Transfert de fichiers

- Protocole : FTP
- Clients : outils de gestion de transfets FTP (Ws_FTP, FileZilla, etc.)
- Serveurs : serveurs FTP (port = 21-22)

Notion de Socket

Exemple 3 : Web

- Protocole : HTTP

- Clients : navigateurs web (Mozilla, IE, Firefox, Opera...)

- Serveurs : serveur web (IIS, Apache, ...) (port = 80)

Exemple 4 : Peer to peer

- Protocoles : envoi de fichier, échange d'informations, ...

- Plient : client P2P

- Serveurs : client P2P, serveurs de métadonnées

Notion de Session

Session : Connexion maintenue entre un logiciel client et un serveur.

Par exemple : Identification sur un intranet,

- Navigation sans donner à nouveau mon login/mdp ;
- Le lien entre le client et le serveur est maintenu même quand il ne se passe rien ;
- Une session est en général coupée si elle dure trop longtemps.

Exemple : Le protocol http

http (HyperText Transfert Protocol) défini par le W3C (World Wide Web consortium) envoi de documents web d'un serveur web (serveur HTTP) vers un client web (navigateur).

Notion de Session

Principe

- Requête du client au serveur demander une ressource web (page, image, service) ;
- Réponse du serveur au client ; envoyer une ressource (page web, image, réponse) ;
- HTTP 1.0 , HTTP 1.1 gère les sessions (permet de garder une connexion).

Référence RFC 2616 : (Request For Comments)

Requête client

- Contenu de la requête (type + URI + version protocole) ;
- En-têtes ;
- Ligne vide (indique fin de requête).

Notion de Session

Réponse serveur

- Code réponse (version HTTP + code + chaine) ;
- En-têtes ;
- Ligne vide (indique fin en-tête) ;
- Contenu de la réponse (souvent le document qu'on veut).

Requêtes :

- GET : demande de document ou de service ;
- POST : demande de service avec envoi de paramètres ;
- HEAD : demande d'information concernant un document ;
- ...

Notion de Session

Réponses : Codées

- 2xx : succès
  - 200 : ok
- 3xx : redirection
  - 304 : document inchangé
- 4xx : erreur client
  - 401 : non autorisé
  - 404 : inexistant
- 5xx : erreur serveur
  - 500 : erreur dans l’exécution d’un service
  - 505 : version HTTP non supportée

Notion de Session

En-têtes de requêtes

- From : adresse email (Non envoyée par la majorité des clients pour des raisons de confidentialité)
- Accept : liste de types MIME (Exemples : audio/mid, image/jpeg, application/pdf)
- Accept-Encoding : liste de méthodes de codage MIME (Exemples : compress, x-gzip, x-zip)
- Accept-Langage : liste des langues acceptées
- User-Agent : l'identification du logiciel client (Permet de répondre différemment suivant le client, ne devrait pas être le cas (car il y a des normes qui devraient être respectées par tous les clients)

Notion de Session

- Referer : page d'où l'on vient (Peut être utile pour faire des statistiques de parcours dans le site)
- Autorization : login password (Niveau faible de sécurité, passe en clair)
- If-Modified-Since : date (Ne transmet la page que si elle a été modifiée depuis la date spécifiée, utile pour les caches
- etc.

<u>En-têtes de réponses</u>

- Server : type du serveur ;
- Date : date du traitement de la requête ;
- Last-Modified : date utile pour le cache ;

Notion de Session

- Content-Type : type MIME du document renvoyé (Doit faire partie en théorie de la liste des types acceptés dans la requête *Accept*) ;
- Content-length : longueur des données (octets), On peut savoir quand (et si) le transfert est fini, permet au navigateurs d'indiquer des barres de progression (Non obligatoire) ;
- Content-Encoding : encodage MIME (Doit faire partie en théorie des méthodes spécifiées dans la requête Accept-Encoding) ;
- Content-Langage : langue ;
- etc.

Notion de Session

Exemple

eguerin >telnet bat710 80

Trying 134.214.88.10...

Connected to bat710.univ-lyon1.fr.

Escape character is '^]'.

HEAD / HTTP/1.1

Host: www710.univ-lyon1.fr

Connection: close

HTTP/1.1 200 OK

Date: Mon, 09 Sep 2010 14:50:22 GMT

Server: Apache/1.3.9 (Unix) Debian/GNU

Notion de Session

Last-Modified: Thu, 11 Jul 2002 09:36:01 GMT
ETag: "27ec6-1811-3d2d5181"
Accept-Ranges: bytes
Content-Length: 6161
Connection: close
Content-Type: text/html; charset=iso-8859-1
Connection closed by foreign host.
eguerin >

Avantages de l’architecture deux-tiers

- Ressources centralisées : les serveurs sont au centre du réseau gèrent les ressources communes à tous les utilisateurs, et permettent d'éviter les problèmes de redondance et de contradiction ;

- Meilleure sécurité : faible nombre de points d’entrée pour l'accès aux données ;

- Administration centralisée au niveau des serveurs : les clients ne sont pas des ressources critiques ;

- Réseau évolutif : ajouter/enlever des clients sans perturber le réseau.

Limites de l'architecture deux-tiers

- On ne peut pas soulager la charge du serveur qui supporte déjà tous les traitements applicatifs ;

- Le serveur est fortement sollicité, il devient de plus en plus complexe et nécessite des mises à jour régulière, ce qui est contraignant ;

- Le client et le serveur sont assez bruyants, ce qui s'adapte mal à des bandes passantes étroites ; ce qui cantonne ce type d'application à des réseaux locaux ;

- Il est difficile de modifier l'architecture initiale, les applications supportent donc mal les fortes montées en charge ;

Limites de l'architecture deux-tiers

- La relation forte et étroite entre le programme du client et l'organisation de la partie serveur complique les évolutions de cette dernière ;

- Ce type d'architecture est grandement rigidifié par les coûts et la complexité de maintenance ;

- Coût élevé des machines serveurs, car fiabilité vitale ;

- Sécurité des échanges réseau ;

- Protection des données qui circulent ;

- Cryptage des données ;

- Protection des données sur les machines ;

Limites de l'architecture deux-tiers

- Identification ;
- Protection contre les attaques ;
- Firewall ;
- Antivirus.

Architecture trois-tiers (Client-Serveur Distribué)

- Le processus Client n’interagit pas directement avec le processus Serveur ;

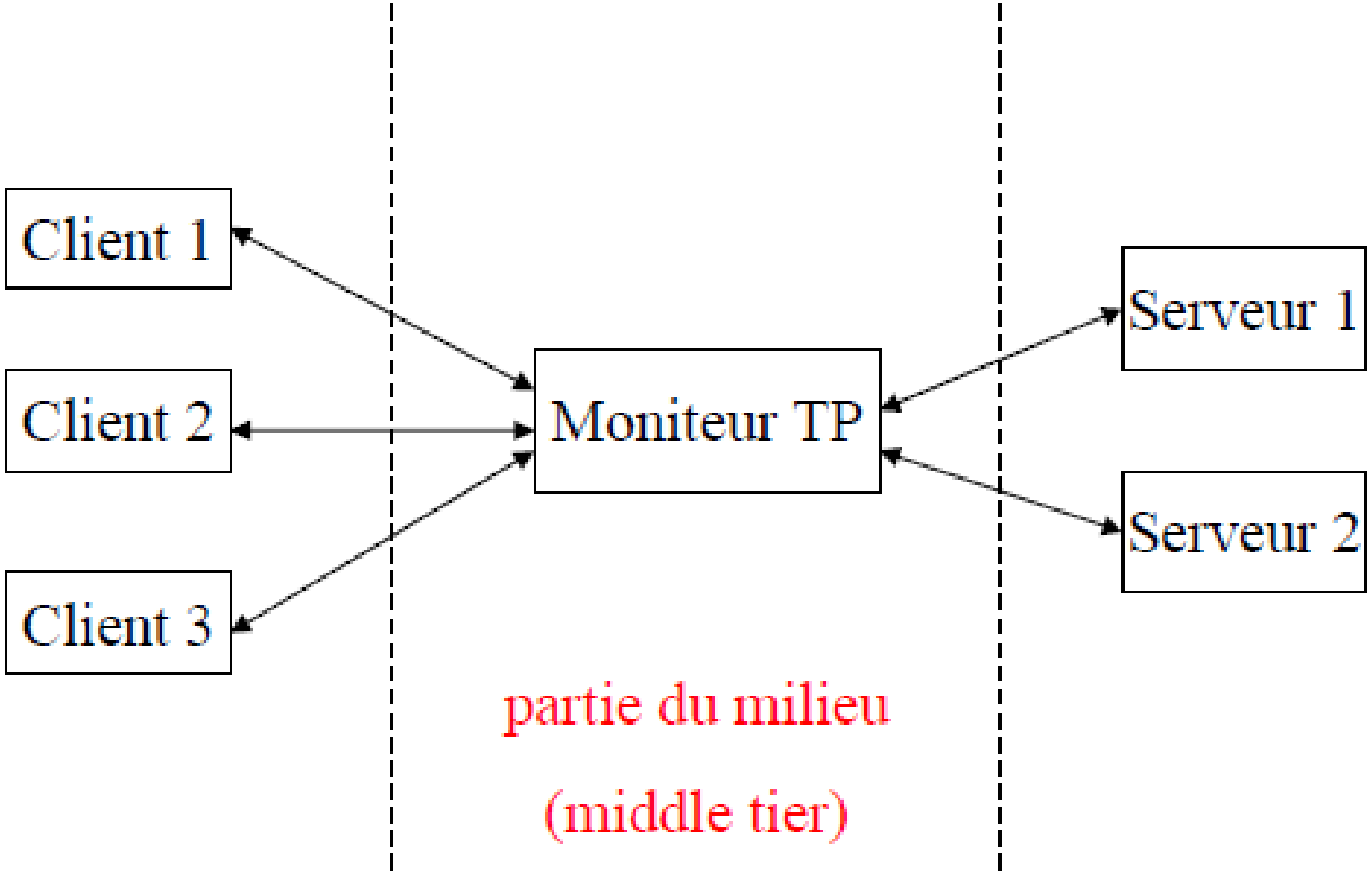
Trois étages : (“tiers” en anglais)

- Étage présentation

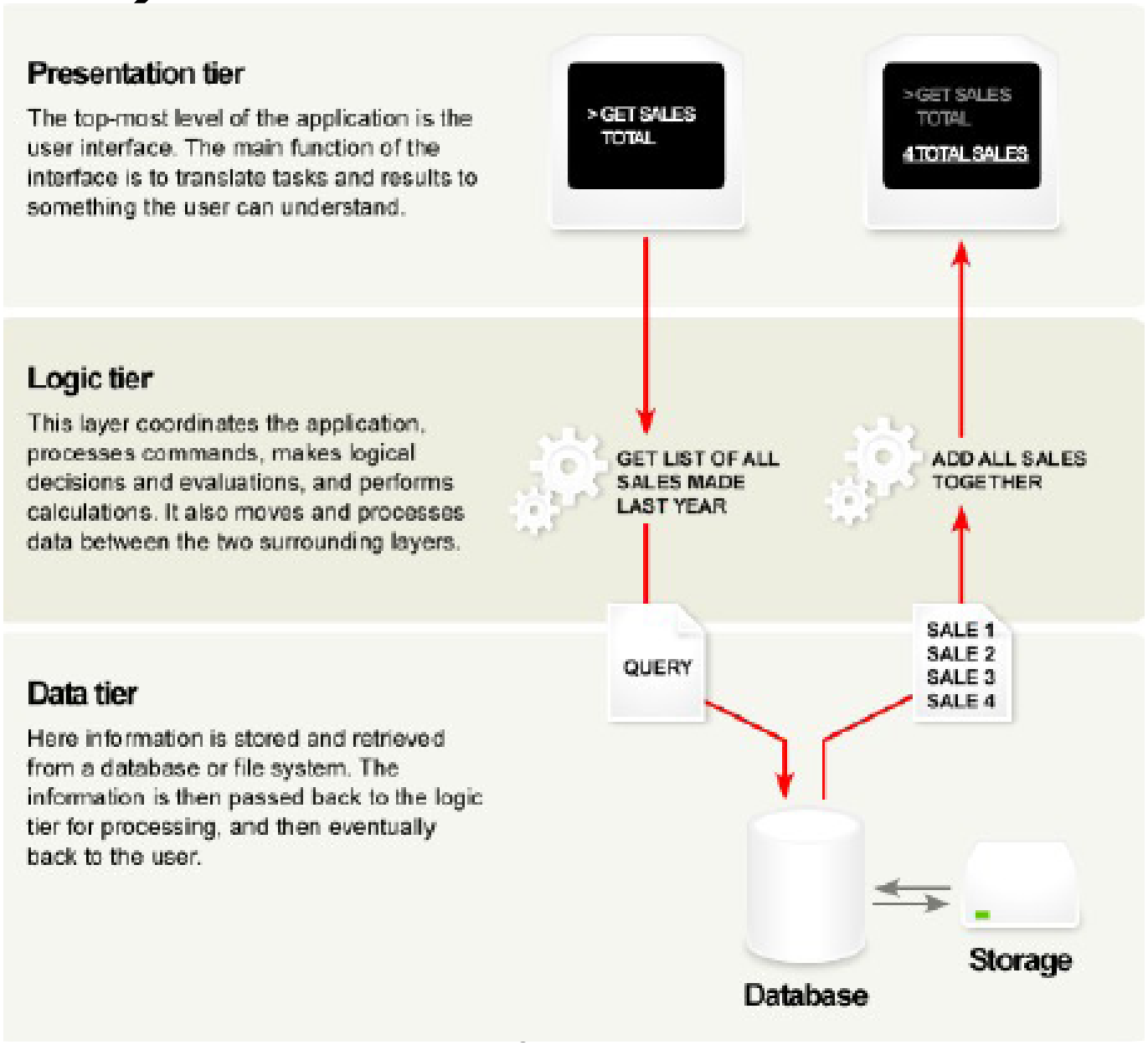
- Étage logique ou “métier”

- Étage données persistantes (SGBD)

Architecture trois-tiers (Client-Serveur Distribué)



Architecture trois-tiers (Client-Serveur Distribué)



Le dialogue trois-tiers

- Chaque niveau ne communique qu'avec ses voisins immédiats au travers une interface de communication bien définie ;

- Pas de communications directes entre la couche présentation et données !

Avantages de l’architecture trois-tiers

- Séparation de la couche présentation et métier, trop souvent imbriquées dans les architectures client/serveur classiques ;

- Meilleure maintenance : isolement des fonctionnalités (e.g. changer facilement l'IHM) ;

- Allègement du poste de travail client (par rapport aux clients/serveur de données).

Architecture n-tiers (Multi-tiers)

- Architecture Peer-to-Peer;
- Architecture totalement distribuée;
- Pas de serveur centralisé;
- Chaque client peut être en même temps un serveur