# Architectures Client-Serveur

Bernard ESPINASSE
Professeur à Aix-Marseille Université (AMU)
Ecole Polytechnique Universitaire de Marseille

Octobre 2018

- **Introduction**
- **Couplage entre client et serveur**
- **Découpage d'une application, Middleware**
- **Différentes architectures de Client-Serveur**
- **Middleware en détail**
- **Performance/Optimisation en Client-Serveur**



## Plan





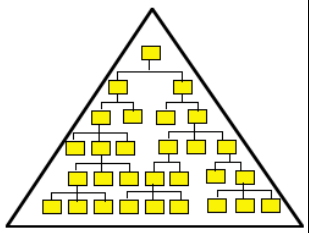
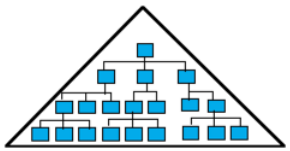
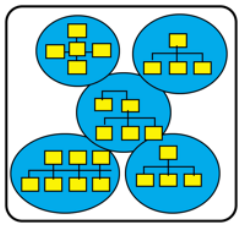
# 1. Introduction

- **Pourquoi le C/S**
- **Les enjeux**



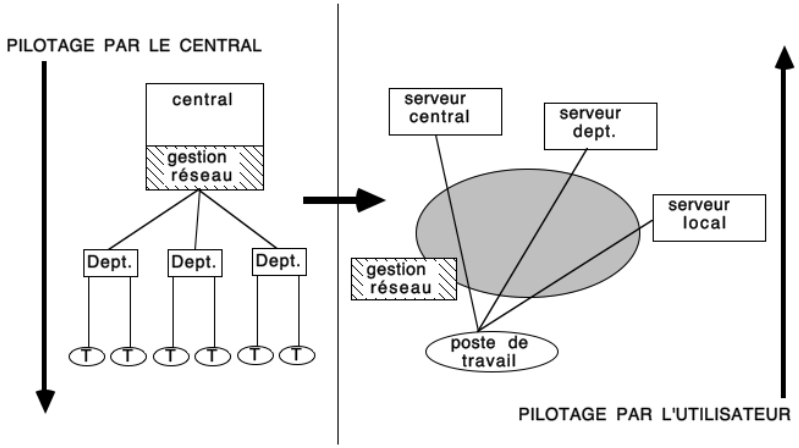
## Introduction : pourquoi le Client-Serveur ?

**Evolution des organisations :**

| 1980-1990 | 1985-1995 | 1995-2000 |
|---|---|---|
|  |  |  |
| **hiérarchique et centralisé** | **hiérarchique et décentralisé** | **réparti** ***(entreprise virtuelle)*** |

## Pourquoi le Client-Serveur ?

**Inversion de la pyramide :**






## Pourquoi le Client-Serveur ?

- **évolution des besoins :** production, informationnel, communication
- **évolution des techniques :** micro-informatique + réseaux locaux
- **contraintes des architectures hierachiques :** réseau centralisé, hétérogénité des postes de travail, rigidité des applications, ...

**→ le client-serveur :**

- **principes généraux :**
  - poste de travail multi-fonction
  - possibilité d'accés multi-serveur
  - localisation des données
  - répartition des traitements
  - pilotage par l'utilisateur
- **conséquences :**
  - définition de relation Client - Fournisseur
  - transformation des métiers
  - évolution des responsabilités
  - une mutation délicate



## Enjeux stratégiques du Client-Serveur ?

### Enjeux stratégiques :

- **productivité individuelle** (micro)
- **performance collective** : communication, disponibilité du patrimoine d'information personnalisable
- **réactivité** : réduction des délais, flexibilité des applications

### Enjeux organisationnels :

- **souplesse organisationnelle** par banalisation du couple "homme-machine" et "polyvalence" du personnel
- **fluidité de l'information**
- **communication** inter-services et inter-personnes

### Enjeux humains :

- **usage intuitif**, **appropriation de l'outil**, **enrichissement des tâches**, initiative, moindre résistance au changement, communication,...

### Enjeux techniques :

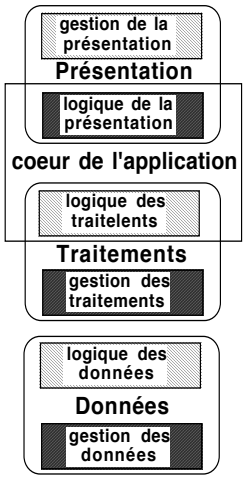
- **cohérence technique** : libres échanges des applications et des informations, …
- **définition d'architectures modulaires** : serveurs, poste de travail, …
- **édification de normes et standards** : chartes, guides, règles d'hygiène , …



# 2. Couplage client-serveur

- **Découpage d'une application**
- **Dialogue entre client et serveur**
- **Inter Process Communication (IPC) ou Middleware**

## Découpage d'une application : 3 niveaux




- **l'interface avec l'utilisateur**
  - **1** • **la gestion de la présentation** : concerne le fenêtrage (lié à un env. graphique d'exploitation (Window, XWindow,...))
  - **2** • **la logique de la présentation :** transmet à la gestion de l'affichage la description des éléments de présentation
- **les traitements**
  - **3** • **la logique des traitements** : contient l'arborescence algorithmique de l'application ( → lancement des procédures de la couche 4)
  - **4** • **la gestion des traitements** : procédures de traitements contenant des requêtes SQL de manipulation de données)
- **les données**
  - **5** • **la logique des données** : garantit le respect de la règle CRUDE pour les objets de la BD mis à jour par les procédures
  - **6** • **la gestion des données** : sélections ou mises à jour des enregistrements (généralement pris en charge par le SGBD)

**Modules 2 et 3 inséparables : coeur de l'application !!!**



## Découpages d'une application : grandes stratégies

**1 • Application en traitements coopératifs :**

→ **module de logique de traitement** :

- **éclaté sur plusieurs processeurs**
- **résidant sur plusieurs machines**

**2 • Application en client/serveur :**

→ **un ou plusieurs modules** sont **déportés sur un serveur** :

*Ex: le module de gestion des données est transformé en service et hébergé par un serveur (serveur de données)*



## Dialogue entre client et serveur

- Permettre l'échange de la **demande** et du **résultat** à cette demande
- S'effectue à travers le **réseau** qui relie les 2 machines :

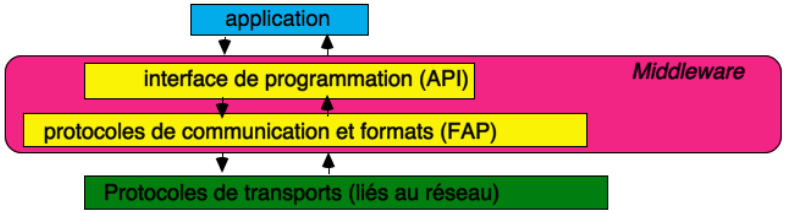
→ **Dialogue interprocessus : Inter Process Communication (IPC)** qui s'appuie côté client et côté serveur sur :

- **API : Application Programming Interfaces** (interface de programmation au niveau applicatif)
- **FAP : Format And Protocols** (protocoles de communication et formats de données)

**IPC = Middleware *(intergiciel)* = ensemble des couches logicielles qui s'interposent entre l'application et le réseau**



## Inter Process Communication (IPC) ou Middleware




- **FAP (Format And Protocols) pilote les échanges à travers le réseau :**
  - synchronisation des échanges selon un **protocole** de communication
  - mise en forme des données échangées selon un **format** connu de part et d'autre
- **API (Application Programming Interfaces) :**
  - les fonctions encapsulées dans l'API permettent à l'application de **faire appel aux services** proposés par le serveur

## Exemple de dialogue C/S

**SELECT libellé, date, sujet FROM dossiers WHERE responsable = mon_user**

- **l'application construit la requête et fait appel à des fonctions de l'API pour l'envoyer au serveur :**
  - fonction 1: remise à zéro de la zone tampon
  - fonction 2: écriture du message de requête, première partie
  - fonction 3: écriture du message de requête, seconde partie à mettre à la suite du précédent
  - fonction 4: fermeture de la zone tampon, le message est complet
  - fonction 5: vérification de la syntaxe de la requête (analyse du message par l'API (phrase en ASCII))
  - si OK : fonction 6: passer la main au FAP pour envoi du message au serveur. L'application se met en attente de la réponse ou effectue une autre tâche en attendant de consulter l'API pour récupérer le résultat
- **l'API passe la main au FAP :**
  - formatage du message : encapsulation du message dans une trame réseau (valise)
  - envoi du message formaté au serveur : selon le protocole de communication
  - à son arrivé, le résultat subit le même l'opération inverse



## Types de Middleware

- **caractérise la nature du dialogue :**
  - **synchrone / asynchrone** : obligation (/ou non) pour le client d'attendre la réponse du serveur après chaque envoi
  - **avec connexion / sans connexion (ou avec session):** nécessité (/ou non) d'établir une connexion entre le client et le serveur

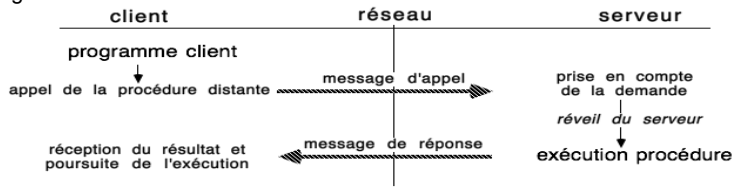
| Dialogue ... | *sans session* | *avec session* |
|---|---|---|
| *synchrone* | RPC | APPC ou RDA |
| *asynchrone* | *message queuing* | APPC ou RDA |

- **RPC** : Remote Procedure Call ou appels de procédure à distance (ex: DCE = IPC basé sur RPC)
- **APPC** : Application Program to Communication (l'APPC = élément de SNA d'IBM)
- **RDA** : Remote Data Access : norme de l'ISO pour l'accès distant aux BdD



## RPC : dialogue synchrone sans cession

- **le client fait une RPC et reste "suspendu" en attendant la réponse du serveur**
- **conversation synchrone très simple :**
  - le message d'appel contient **tous** les éléments nécessaires au serveur **(nom de la procédure et paramètres associés, données d'identification de l'appelant (pour vérifier autorisation))**
  - le message en retour, **en un seul flot, contient toute la réponse attendue** par le programme client

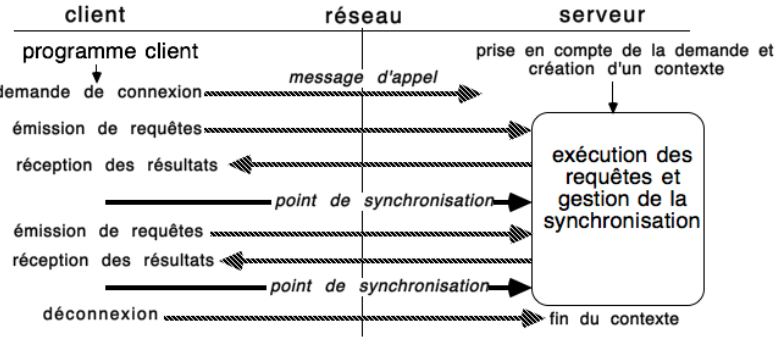



→ **Inconvénients :**
- **synchrone**
- **fiabilité médiocre** (si l'émission initiale échoue, le client n'est pas averti et pas de mécanisme de reprise interne au protocole sous-jacent),
- **pas de resynchronisation possible entre C et S**
- **pas de gestion de flux de retour** (tjs un seul flot)



## APPC/RDA : dialogue asynchrone avec session

- **la demande de connexion** émise par le programme client
- si le serveur accepte la connexion → **création d'un contexte** propre au programme client sur le serveur
- durant toute la conversation **asynchrone**, client et serveur vont échanger **3 types** de **messages : requêtes, résultats, points de synchronisation**



## APPC/RDA : dialogue asynchrone avec session

- **l'échange de points de synchronisation** (principalement C→S) permet de garantir un état stable au contexte du client
- **l'applicatif client défini et pilote les phases successives de l'échange**
- **serveur garantit le contexte** tel qu'il est perçu par le client
- un ordre **Commit** ou **Rollback** du client conduira à un **point de synchronisation**
- un **début** ou une **fin de transaction** client conduira à un **point de synchronisation**

**Avantages:**

→ **gestion de l'échange plus facile** à mettre en oeuvre qu'avec les RPC

**Inconvénients :**

→ plus **coûteux** en ressources que les RPC car tout au long de l'échange :

**- communication C/S maintenue**

**- le serveur crée et conserve le contexte**



## APPC/RDA : dialogue asynchrone avec session

- échange Client-Serveur utilisant IPC fournit par un éditeur de SGBDR (typiquement API-SQL et FAP RDA) :

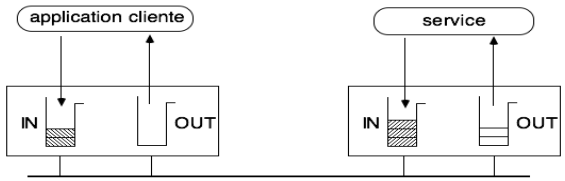
| Application cliente | Processeur serveur (SGBDR) |
|---|---|
| l'application veut adresser une requête SQL de type SELECT | |
| établissement de la connexion | création du curseur (notion de contexte propre au client connecté) |
| émission de la requête | compilation de la requête |
| | exécution de la requête, message de bonne fin |
| demande de la structure du résultat | envoi du descriptif de la structure du résultat |
| demande des n premières lignes composant le résultat | envoi des n premières lignes |
| demande des n lignes suivantes composant le résultat | envoi des n lignes suivantes composant le résultat |
| demande des n lignes suivantes composant le résultat | réponse : plus de lignes à envoyer |
| fin de connexion | destruction du curseur |



## Message queuing : dialogue asynchrone et sans session

- **Echanges fondamentalement asynchrones :**

*le client envoie un message à un destinataire (le service) désigné par un nom (plutôt qu'une adresse ou une localisation) sans se soucier de sa disponibilité*




**Avantages :**

- grande simplicité car l'API repose sur les 2 verbes {envoyer, recevoir}
- la technique "stocker et propager" (store and forward) garantit, quels que soit les événements, que le service appelé sera effectué une et une seule fois (utile dans applications financières)

**Inconvénients :**

- manque de contrôle sur le délai d'obtention d'une réponse



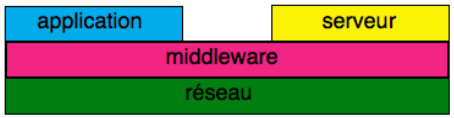
## Exemples d'IPC possibles

| APPLICATION | | | | | | |
|---|---|---|---|---|---|---|
| SQL | CPI-C | RPC | SQL | RPC | *interface de programmation* | API |
| RDA | APPC | DCE | RDA | APPC | *protocole de communication* | FAP |
| IPX | SNA | TCP-IP | TCP-IP | Netbios | *protocole de transport* | FAP |
| *exemple 1* | *exemple 2* | *exemple 3* | *exemple 4* | *exemple 5* | | |

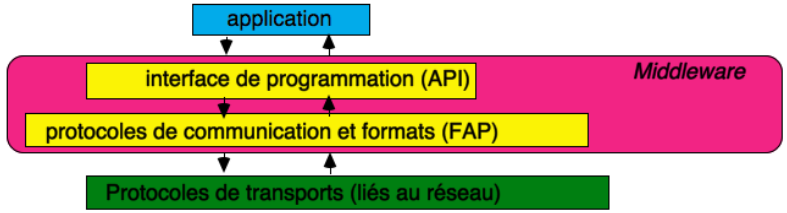
| exemple 1 | exemple 2 | exemple 3 |
|---|---|---|
| • API : type SQL fournies par les éditeurs de SGBD (Oracle, Sybase, ...)<br>• protocoles de com. et formats des données échangées aussi (s'inspire souvent de la norme RDA (ISO))<br>• protocole IPX pour accéder au serveur disposant d'un même IPC | • API : CPI-C (Common Programming Interface Communication - ISO (origine IBM))<br>• protocole APPC (Application Program to Program Communication)<br>• SNA (protocole de transport - LU6.2) | l'application a recours à des appels de procédures à distance<br>• API : de type RPC<br>• DCE (Distributed Computing Env. quasi standard)<br>• RDA avec un transport TCP-IP |

→ **attention :** toutes les combinaisons ne sont pas toujours possibles et toutes celles qui sont possibles n'ont pas nécessairement une implémentation disponible ...

## Le middleware C/S en détail




- **middleware (=IPC) = intelligence du réseau**
- permet d'unifier pour les applications l'accès et la manipulation de l'ensemble des services disponibles sur le réseau



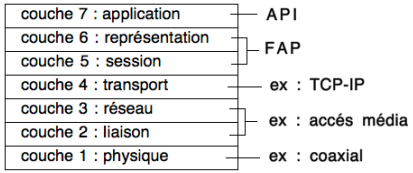

→ **middleware = clé de l'interopérabilité : élargir les fonctionnalités et aller vers les standards**



## Le middleware C/S en détail

| *Couche* | *Fonction* | *Exemple* |
|---|---|---|
| **API** | **interface de programmation**<br>point d'entrée unique pour les applications | |
| **FAP** | **protocole de communication**<br>gère l'ordonnancement de l'échange : ouvrir une connexion, envoyer une requête, récupérer résultats, créer les messages -> transport | |
| ***transport*** | **protocole de transport**<br>insère les messages et les insère dans une trame qui circulera sur le réseau | TCP-IP<br>Netbios<br>... |
| ***méthode d'accès au média*** | **protocole d'accès au média** | Ethernet<br>TokenRing |

**Middleware et couches ISO :**






## Les types de middlewares C/S

- **Fonctions assurées :**

| Elémentaire | Intermédiaire | Etendue |
|---|---|---|
| • gestion du protocole de communication | • passerelles vers SGBD | • catalogue complet des données |
| • transfert des requêtes et des résultats | • pas de catalogue des données | • support total des appels de fonctions |
| • transmission des codes d'erreurs et de statut | • support limité des appels de fonctions | • transparences de la localisation des données |
| | • administration restreinte du ou des serveurs | • administration complète des serveurs et des services associés |
| | | • sécurité étendue |

*(d'après A. Lefebvre)*

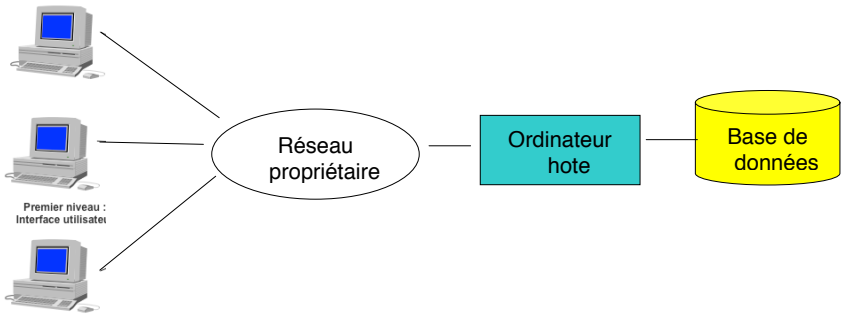
- **Marketing :**
  - proposés par les **éditeurs de serveurs** (SGBD (Oracle, ...))
  - proposés par les **éditeurs de middleware** (ex: Sequelink,...)
  - proposés par les **constructeurs** (DRDA/IBM, DDA/Bull, ...)



# 3. Architectures Client-Serveur

- **Grandes générations de C/S**
- **Modèle du Gardner Group – C/S 2 tiers**
- **Modèle du Gardner Group – C/S 3 tiers**

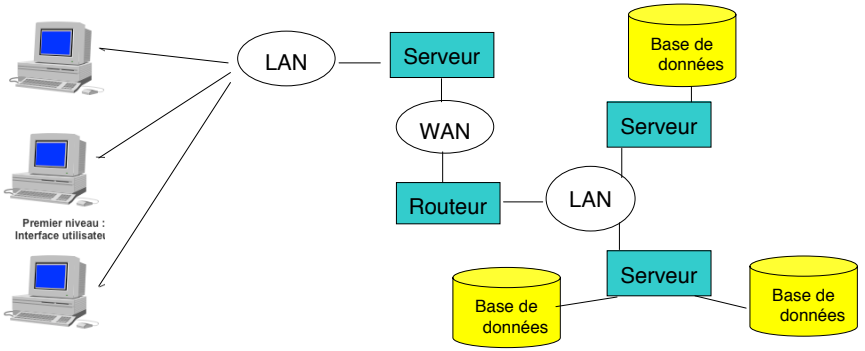
## C/S de 1ère Génération (dès 1970)




- Terminaux passifs pas ergonomiques
- **Client** : gestion présentation
- **Serveur** : réalisation



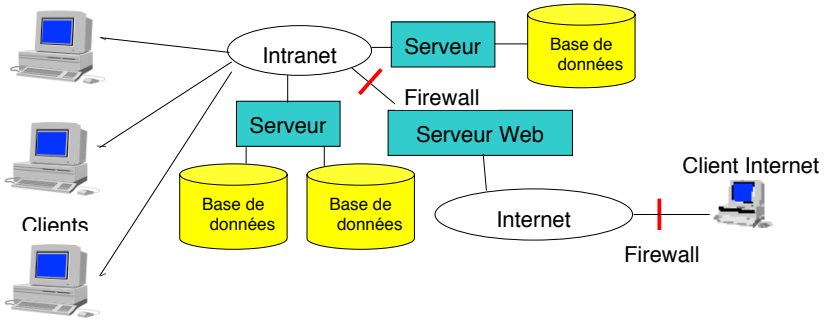
## C/S de 2ième Génération (dès 1980)




- Terminaux ergonomiques
- LAN : Large Area Network - WAN : Wide Area Network
- **Client** : gestion présentation + Portage traitements applicatifs
- **Serveur** : gestion accès BD



## C/S de 3ième Génération (dès 1990)




- **Clients** (Unix, Windows, …) : gestion présentation
- **Serveur applicatif** : lien entre client et plusieurs serveurs de BD
- **Serveur de données** : gestion accès BD

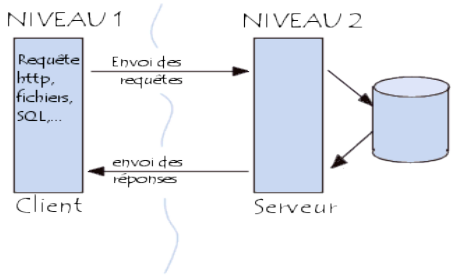


## Client-serveur 2 tiers

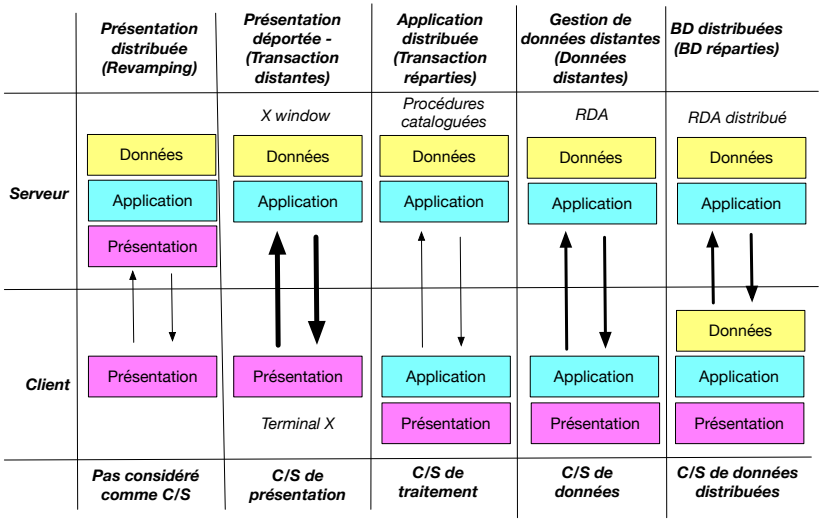
En anglais « tier » signifiant *rangée*, *couche*

**L'architecture C/S à 2 tiers :**

- C/S pour lesquels le client demande une ressource et ***le serveur la lui fournit directement, en utilisant ses propres ressources***.
- C/S dont le serveur ***ne fait pas appel à une autre application*** pour fournir une partie du service.

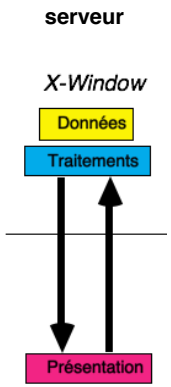


## Le modèle du Gartner Group – 2 tiers

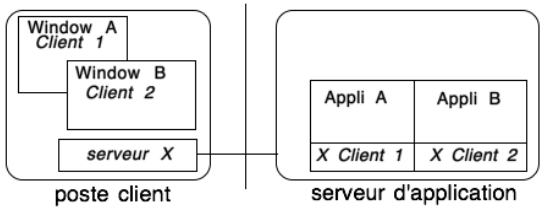





## Client-Serveur de présentation

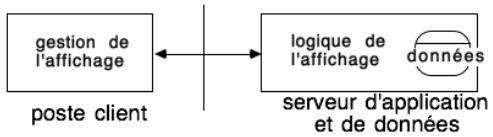



- suppose de pouvoir **séparer** la **gestion de l'affichage** de la **logique de l'affichage**
- **le C/S de présentation nécessite un gestionnaire de fenêtres**
  - → **interface s'appuyant sur un gestionnaire de fenêtres (Window Manager) :** ***par exemple Motif sous Unix s'appuie sur le gestionnaire de fenêtre X-Window***
  - → **interface ne s'appuyant pas sur un gestionnaire de fenêtres et reposant sur le système d'exploitation :** ***par exemple Windows de Microsoft***
- **X-Window =** C/S de présentation en mode natif = serveur d'affichage ou "serveur X"






## Client-Serveur de présentation



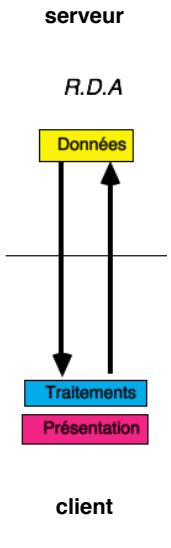

| logique de l'affichage | gestion de l'affichage |
|---|---|
| émission : affichage d'une fenêtre avec telles dimensions, telles caractéristiques et tel emplacement initial | |
| | réception et exécution de la requête, émission d'un accusé de réception |
| | l'utilisateur déplace et redimentionne la fenêtre |
| | l'utilisateur clique sur un des boutons de la fenêtre |
| | émission : événements de type "click" sur l'objet "bouton_1" |
| réception du message, prise en compte de l'événement, préparation de la requête suivante | |
| émission : affichage d'un dialogue... | |

→ **avantages** : indépendance entre logique de présentation et l'interface graphique utilisée (les standards X11 et NFS)

→ **inconvénients** : trafic réseau généré par le protocole X11 important, X11 pas une norme, stabilité ?

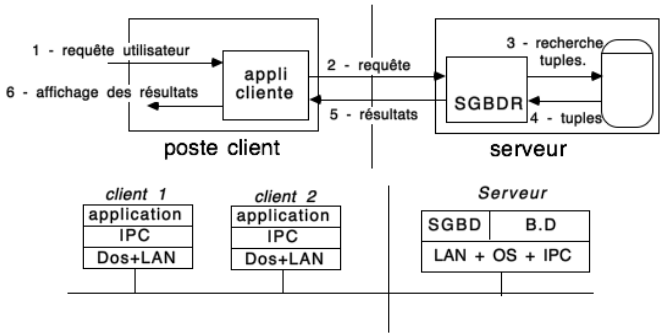


## Client-Serveur de données




- le serveur abrite la **gestion des données**
- très tôt les **SGBDR** ont proposé des IPC pour accéder à leurs données (Oracle, Ingres, ... 1985)
- le serveur assure aussi **l'intégrité des données**
- les SGBD modernes proposent des **mécanismes** permettant de déclencher des traitements de contrôle :
  - s'assurant de la validité des mises à jour des BD,
  - indépendamment des applications
  - incontournables
  - → *préservation de la cohérence des données*
  - → *plus d'efficacité*
  - → *moins de maintenance*
  - → *plus de sécurité*

## Client-Serveur de données




**Avantages :**

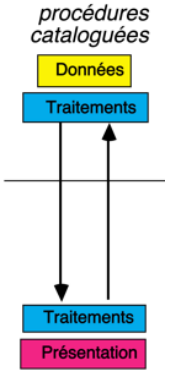
- **facile** à mettre en oeuvre,
- **largement disponible**,
- **bien adapté** aux utilisateurs de type consultation/décision

**Inconvénients :**

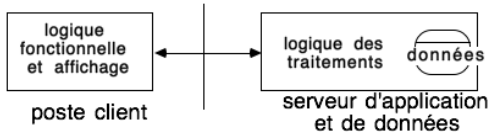
- **pas complètement normalisé**,
- **inadapté aux exigences du transactionnel intensif**



## Client-Serveur de traitements




- **meilleure répartition des traitements** sur le client et le serveur → **charge plus faible sur le réseau**
- nécessite à un **découpage fin du noyau de l'application**, demande beaucoup de savoir-faire pour sa mise en oeuvre → encore peu répandu

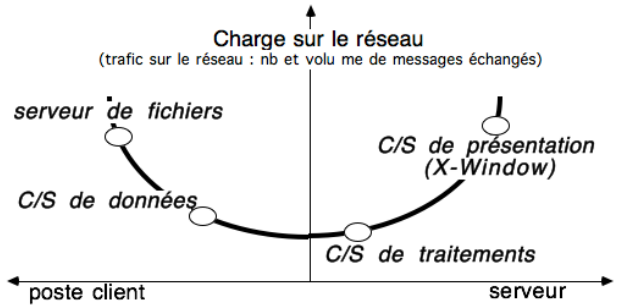



- le module "logique fonctionnelle" de l'application client envoie des requêtes SQL mais aussi des **appels de procédures (procédures cataloguées) au serveur** qui les exécute et renvoie les résultats
- sur le serveur, le module d'exécution des procédures n'est pas obligatoirement associé aux modules d'intégrité et de gestion des données
- peut être mis en oeuvre avec un mécanisme de type **RCP** entre client et serveur



## Client-Serveur de traitements

**Répartition du processus :**




**Avantages :**

- **meilleures performances,**
- **trafic réseau réduit**

**Inconvénients :**

- nécessite un **développement côté serveur**,
- **ne convient pas pour les applications "haddock" type infocentre**

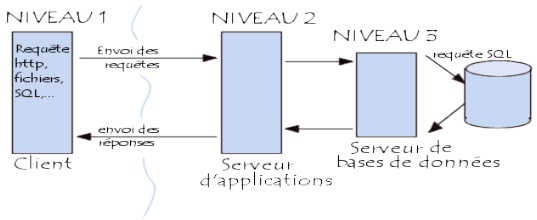


## Client-Serveur 3 tiers

Un serveur peut utiliser les services d'un ou plusieurs autres serveurs pour fournir son propre service.

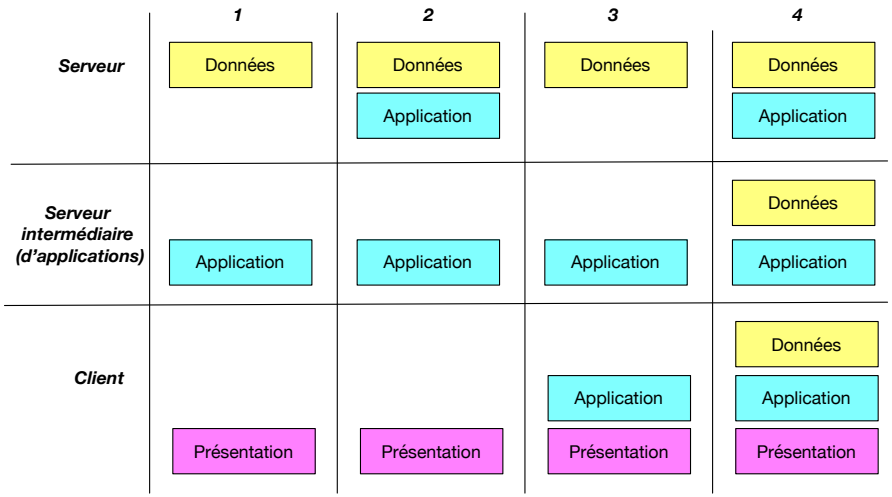
**On a généralement une architecture partagée entre :**

1. Un **client**, c'est-à-dire l'ordinateur demandeur de ressources, équipée d'une interface utilisateur (généralement un navigateur web chargée de la présentation
2. Le **serveur d'application** chargé de fournir la ressource en faisant appel à un autre serveur
3. Le **serveur de données**, fournissant au serveur d'application les données dont il a besoin



## Le modèle du Gardner Group - 3 tiers

Différentes configurations possibles :






## Client-Serveur 2 tiers Versus 3 tiers

*(Source : M. Cilia – Darmstadt Univ. )*

**Les architectures à 2 tiers sont typiques :**

- des environnements avec peu de clients
- d'environnements homogènes
- d'environnements fermés (par exemple des SGBD)

**Les architectures à 3 tiers sont nécessaires pour :**

- le passage à des milliers de clients (scalabilité)
- l'accès à des sources de données hétérogènes
- la maintenabilité (mise à jour du logiciel sur quelques serveurs d'applications au lieu de milliers de clients)

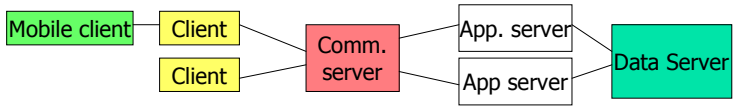


## Client-Serveur N tiers (Multi-tiers)

*(Source : M. Cilia – Darmstadt Univ. )*

**Les architectures à N niveaux résultent lorsque :**

- la fonctionnalité est déléguée à des ***serveurs spécialisés*** (communication, Web, application, serveur de données, …)
- des ***clients mobiles*** sont considérés (client de bureau pourrait servir de serveur au client mobile)
- on considère des ***systèmes d'objets distribués*** dans lequel chaque serveur peut agir en tant que client sur un autre serveur.

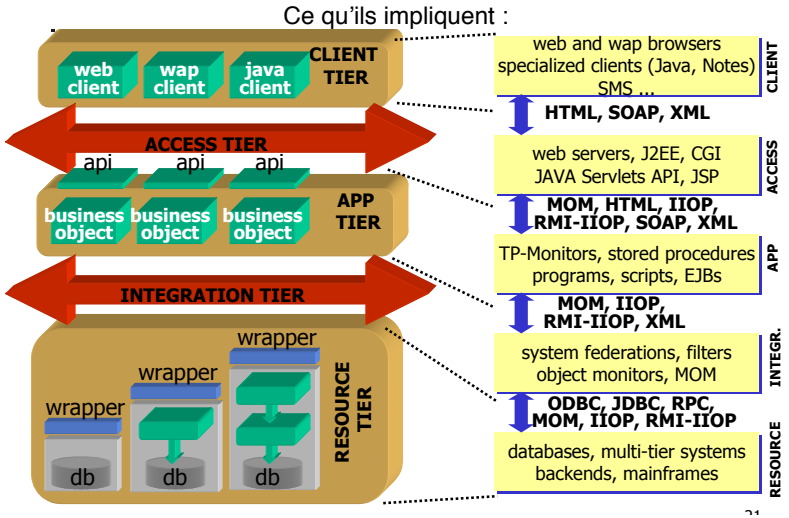





## Client-Serveur N tiers (Multi-tiers)

*(Source : M. Cilia – Darmstadt Univ. )*

Ce qu'ils impliquent :



# 4. Performance/optimisation en C/S

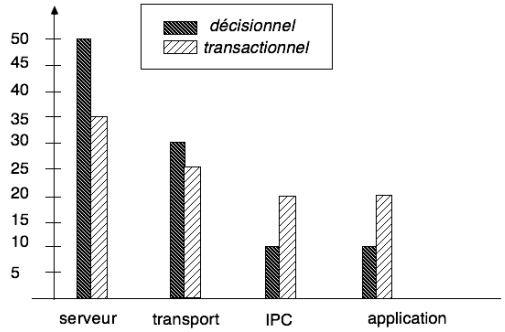
- **Performance**
- **Optimisation**



## Les performances

*(Source : étude Dipro )*

***(Application : C sous Windows; requête SQL lecture; SGBD sous Unix):***




**→ le serveur consomme la plupart du temps nécessaire à l'exécution complète d'une requête (simple ou complexe)**

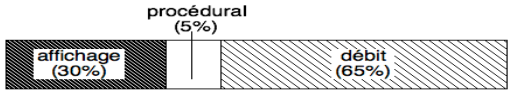


## Les performances

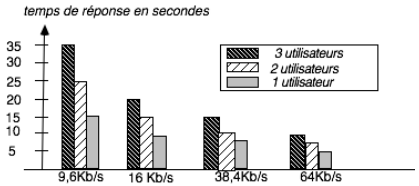
**Les performances d'une application cliente dépendent de 3 critères :**

- le **débit** : doubler le débit (réseau) des liaisons améliore généralement les temps de réponse de 30%
- **l'affichage :** lié à la performance de l'environnement graphique
- **l'exécution procédurale :** importance de la vitesse d'exécution du langage

**Importance relative des critères (en général) :**




**Influence du Nb d'utilisateurs :**

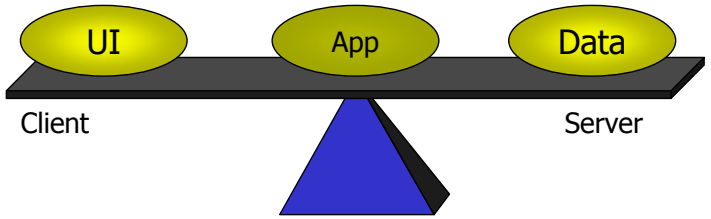





## Optimisation Client/Serveur : une question d'équilibre

*(Source : M. Cilia – Darmstadt Univ. )*

**Client lourd <-** **I** **-> Serveur lourd**



## Performance Client/Serveur : client lourd

*(Source : M. Cilia – Darmstadt Univ. )*

**Intégrité et cohérence**

- Vérification des types de données (Data type, plages de valeurs, etc. – intégrité)
  *Ex : Vérification de la date de naissance*
- Vérification de l'existence de données – intégrité référentielle (clés étrangères)
  *Ex : l'existance d'un département*
- Calculer le numéro d'employé suivant automatiquement [Autonumber] (read last; add one; return it)
  *Ex : Employee number (unique)*

**Business Rules (règles d'affaire) :**

- Ils pourraient être spécifiques à l'application
- Règles métier dispersées dans de nombreuses applications
- Exprimé sous forme de code de programmation
  *Ex : Bonus du département (10% pour tous les programmeurs)*



## Performance Client/Serveur : serveur lourd

*(Source : M. Cilia – Darmstadt Univ. )*

**Intégrité et cohérence**

- Basé sur les procédures cataloguées (Stored Procedures)
- Une version unique de la vérification est maintenue du coté du serveur
- « Autonumber » est une procédure cataloguée qui contrôle l'unicité

**Business Rules (règles d'affaire) :**

- Basé sur les procédures cataloguées (Stored Procedures) et des Triggers
- Les « Business rules » sont localisées à un seul endroit : sur le serveur
  Ex :
  - Department Bonus (10% pour tous les programmeurs)
  - Table Employe (colonne dept) a un Trigger qui est exécuté sur un UPDATE ou in INSERT et qui exécute une procédure cataloguée.



## Performance Client/Serveur : conclusion

*(Source : M. Cilia – Darmstadt Univ. )*

- La ***séparation entre le client et le serveur*** d'application n'est jamais nette :
  - Il n'y a pas de recette
  - Mais des expériences !
- Tout dépend du ***type d'application***
- Cela dépend aussi du ***matériel*** (client et serveur)
- Les ***données mises en cache*** sont nécessaires lorsqu'une partie de la logique d'application est située sur le client vérifiant les contraintes d'intégrité du côté client au moment de la saisie des données.
- Dépend du type d'interaction