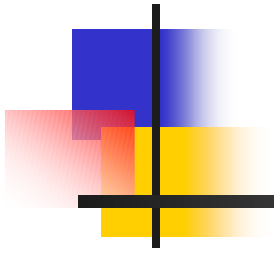


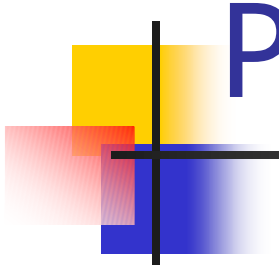
NSY107 - Intégration des systèmes client-serveur

Cours du 13/05/2006

(4 heures)

© Emmanuel DESVIGNE
<emmanuel@desvigne.org>

Plan du cours



- Introduction
- Historique
- Les différentes approches
- Architecture (initiation)
- Bilan

Rappels - Introduction [1/13]

- La notion de client-serveur est très floue. Intuitivement, on suppose qu'il existe un « client », souvent assimilé au poste de travail de l'utilisateur, qui interroge une « grosse machine » associée au serveur. Si ce modèle existe, nous verrons qu'il n'est pas unique (loin de là), et que la réalité est souvent plus complexe.

Rappels - Introduction [2/13]

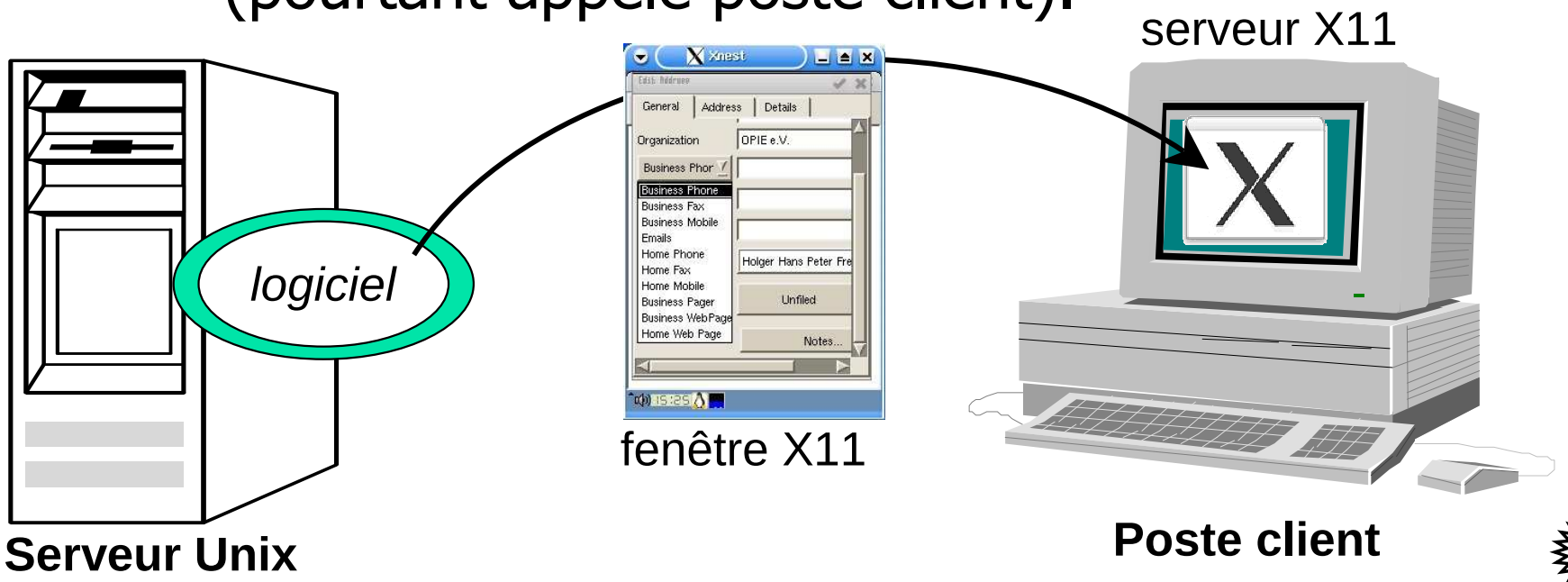
- En réalité, la notion de client-serveur peut se concevoir :
  - d'un « poste client » à un « serveur » (notion classique, quoi que…). Par exemple :
    - Un PC (Pentium 4 à 2 GHz) <-> un serveur de logiciel de paie (bi-proc 2 x Xéon 3GHz),
    - Un PC (Athlon 2600+) <-> serveur d'impression (parfois un serveur puissant sous Unix ou Windows, parfois un petit boîtier avec un processeur ARM à 200 MHz, voire un simple serveur intégré à l'imprimante),
    - Un client léger (tout petit PC avec un écran, avec un système d'exploitation minimal, une carte mère, pas de disque dur dont le rôle est de ne faire que de l'affichage) <-> serveur TSE ou Citrix (proche du concept Serveur/Terminal passif).

Rappels - Introduction [3/13]

- Mais aussi, d'un serveur à un autre serveur. Par exemple :
  - Une application métier (logiciel de paie, de compta, de dossier médical, de commerce en ligne, etc.) <-> serveur de Système de Gestion de Base de Données (SGBD),
  - Un serveur SGBD avec un serveur de fichiers,
  - Un serveur de fichiers avec un serveur d'authentification/habilitations,
  - Un serveur d'emails avec un serveur d'annuaires,
  - Etc.

Rappels - Introduction [4/13]

- Voire... d'un serveur à un client !?!? :
  - Ex : machine Unix (pourtant appelée serveur) sur laquelle tourne un programme scientifique qui envoie le contenu des fenêtres à afficher sur un serveur X11 situé... sur un poste de l'utilisateur (pourtant appelé poste client).



Rappels - Introduction [5/13]

- Oxymoron ?
  - en fait, ce dernier paradoxe vient d'un abus de langage. Le mot *serveur* est à la fois utilisé pour parler :
    - de la machine sur laquelle tourne le logiciel qui fourni les traitements ou les données sur lesquels travaille l'utilisateur,
    - et aussi pour parler du **service** X11, qui réalise l'affichage des fenêtres générées par le logiciel.

Rappels - Introduction [6/13]

- Tentative de définition [*source Wikipedia*] :
  - L'architecture client-serveur désigne un mode de communication entre des ordinateurs ou des logiciels. Les mots « serveur » et « client » peuvent soit désigner :
    - les ordinateurs, on parle alors de **serveur informatique** et de **poste client** ;
    - soit désigner les logiciels fonctionnant sur ces ordinateurs, on parle alors de logiciel serveur [*ou service*] ou de logiciel client.
  - Le serveur est à l'écoute sur un **réseau informatique**, prêt à répondre aux requêtes envoyées par des clients.

Rappels - Introduction [7/13]

- Tentative de définition, suite :
  - Les clients « *sont pilotés par les utilisateurs** » et envoient des requêtes au serveur, puis attendent la réponse pour la donner à l'utilisateur.
  - [*Pas toujours vrai*] : un serveur est capable de servir plusieurs clients simultanément, jusqu'à plusieurs milliers.

(*) certains clients envoient des requêtes sans intervention directe de l'utilisateur (intelligence artificielle, tâches programmées, etc.)

Rappels - Introduction [8/13]

- Il existe deux grands modes de fonctionnement client/serveur (qui seront détaillés plus tard) :
  - Le mode connecté : un « canal de communication » se crée entre le client et le serveur (à la demande du client), et les échanges (ordres, accusés de réception, données...) transitent par ce canal, ou alors via d'autres canaux ouverts pour l'occasion (ex : HTTP, IMAP, FTP...) ;
  - Le mode « datagramme », ou « échanges de paquets » : il s'agit souvent de systèmes plus simples, où le client envoie sa requête (un ordre) dans un paquet, et le serveur lui répond dans un ou plusieurs paquets (ex: TFTP).

Rappels - Introduction [9/13]

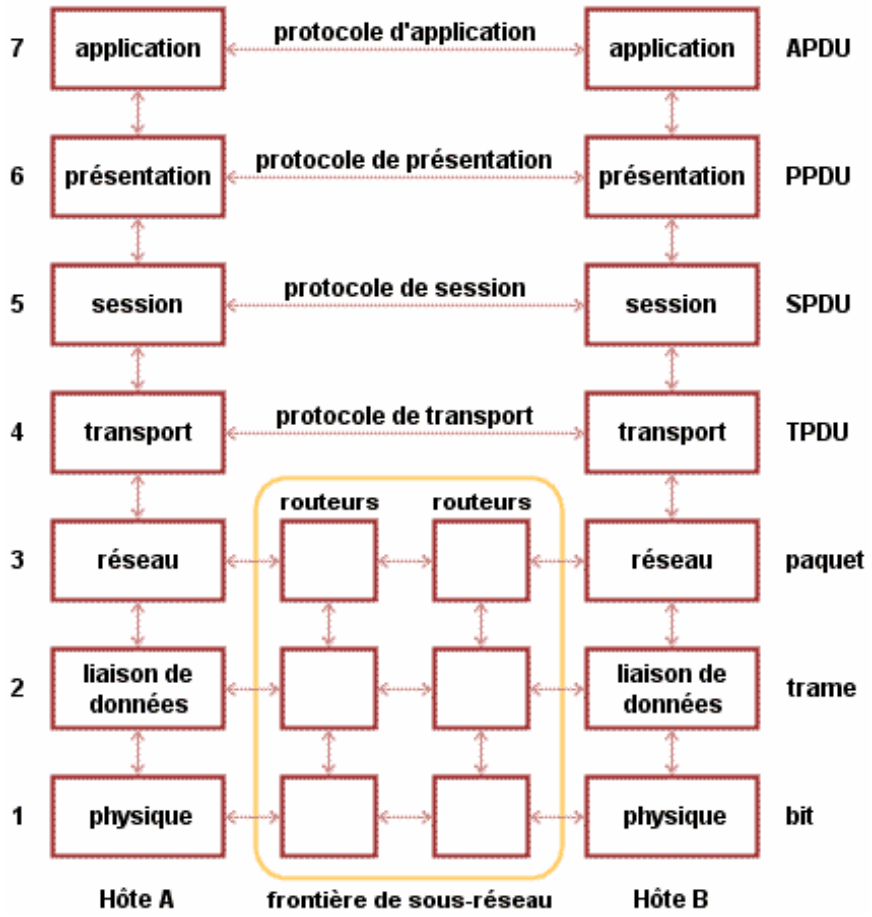
- La notion de client-serveur est intimement liée à deux concepts informatiques :
  - « **système** » : les systèmes [*déf :architecture physique + système d'exploitation*] modernes permettent :
    - le multi-tâche (voire le multi-thread), mono et multiprocesseurs,
    - la gestion des « événements » (interruptions).
  - « **réseau** » : si, sur le papier, il est possible d'imaginer du client/serveur monoposte, le client communiquant avec le serveur via des *IPC*, la notion de client-serveur n'a d'intérêt que pour plusieurs machines.

Rappels - Introduction [10/13]

- IPC (Inter-Process Communication, soit « communication inter-processus ») : ensemble de mécanismes mis à disposition par un système d'exploitation pour que plusieurs programmes concurrents (qui utilisent les mêmes ressources) tournant sur une même machine puissent communiquer entre-eux (échanges de données, synchronisation ou points de rendez-vous). Ex :
  - fichiers,
  - pipes (tubes),
  - sémaphores,
  - mémoire partagée (shared memory)

Rappels - Introduction [11/13]

- Petit rappel du modèle OSI :



Rappels - Introduction [12/13]

- Comparaison OSI-IP (Rq : il arrive que dans certaines revues, les couches 5 à 7 soient regroupées) :

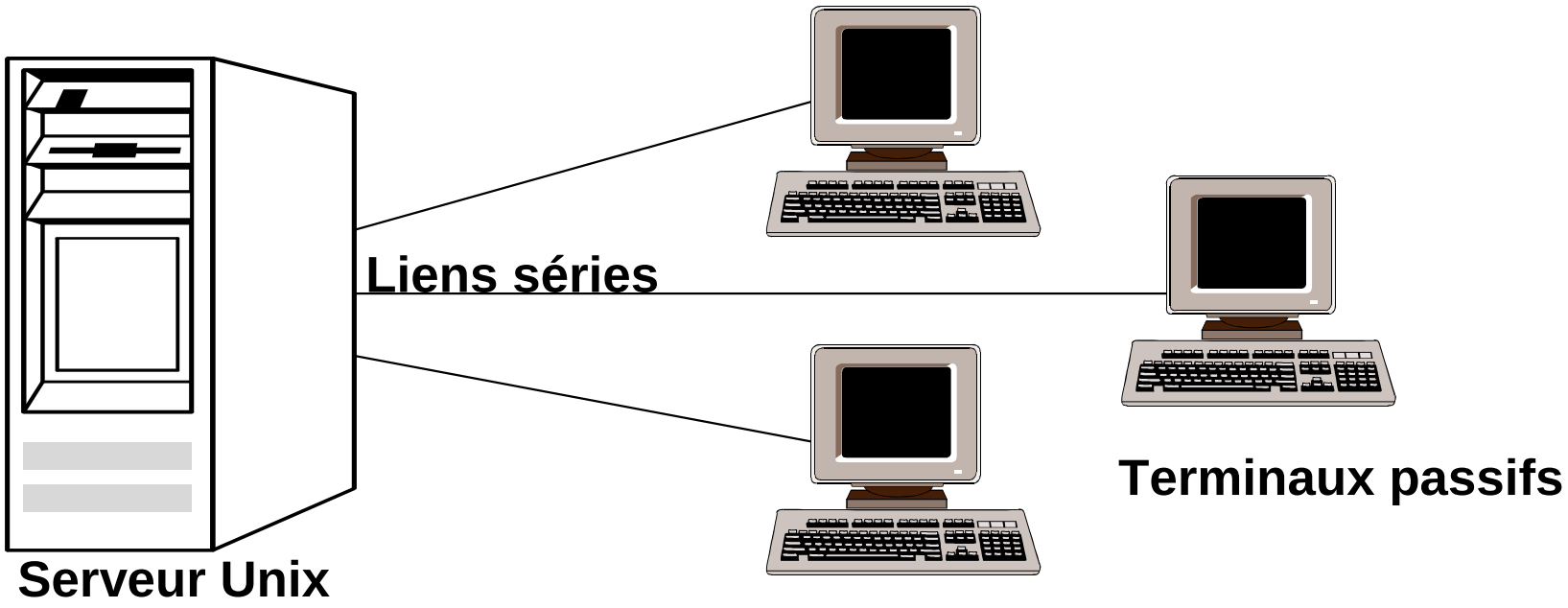
7	**Application**	ex. HTTP, SMTP, SNMP, FTP, Telnet, NFS
6	**Présentation**	ex. XDR, ASN.1, SMB, AFP
5	**Session**	ex. ISO 8327 / CCITT X.225, RPC, Netbios
4	**Transport**	ex. TCP, UDP, RTP, SPX
3	**Réseau**	ex. IP, ICMP, IGMP, X.25, CLNP, ARP, OSPF, RIP, IPX, DDP
2	**Liaison**	ex. Ethernet, Token Ring, PPP, HDLC, Frame relay, RNIS, ATM
1	**Physique**	ex. électronique, radio, laser

Rappels - Introduction [13/13]

- Relation « client-serveur » et « réseau » :
  - la notion de « client/serveur » se calque avec la notion de « protocole réseau ». En fait, fondamentalement, programmer du client/serveur revient à développer un protocole réseau,
  - depuis ~20 ans, divers mécanismes (RPC par exemple, que nous retrouverons plus tard) permettent de simplifier (voire, de rendre transparent) l'écriture de ces protocoles pour les programmeurs,
  - certains protocoles utilisent des échanges de flux en texte (ex : ASCII) et peuvent être compris par des humains (ex: SMTP, POP3, HTTP, etc.). D'autres utilisent des échanges de flux binaires (ex : DNS).

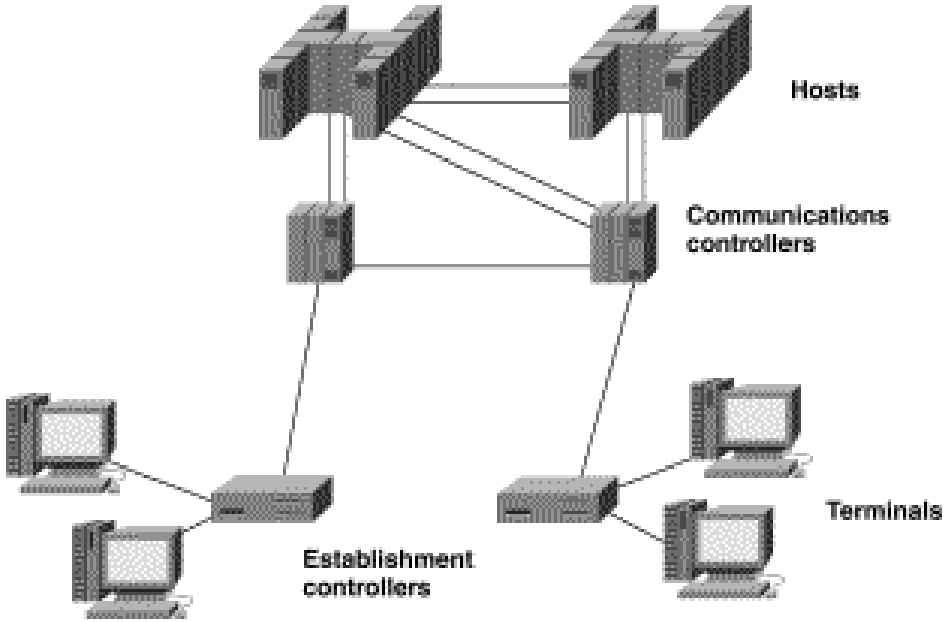
Rappels - Historique [1/5]

- à l'origine : systèmes centralisés :
  - Version simple : le serveur central avec terminal passif (ex : serveur Unix avec terminaux passifs):



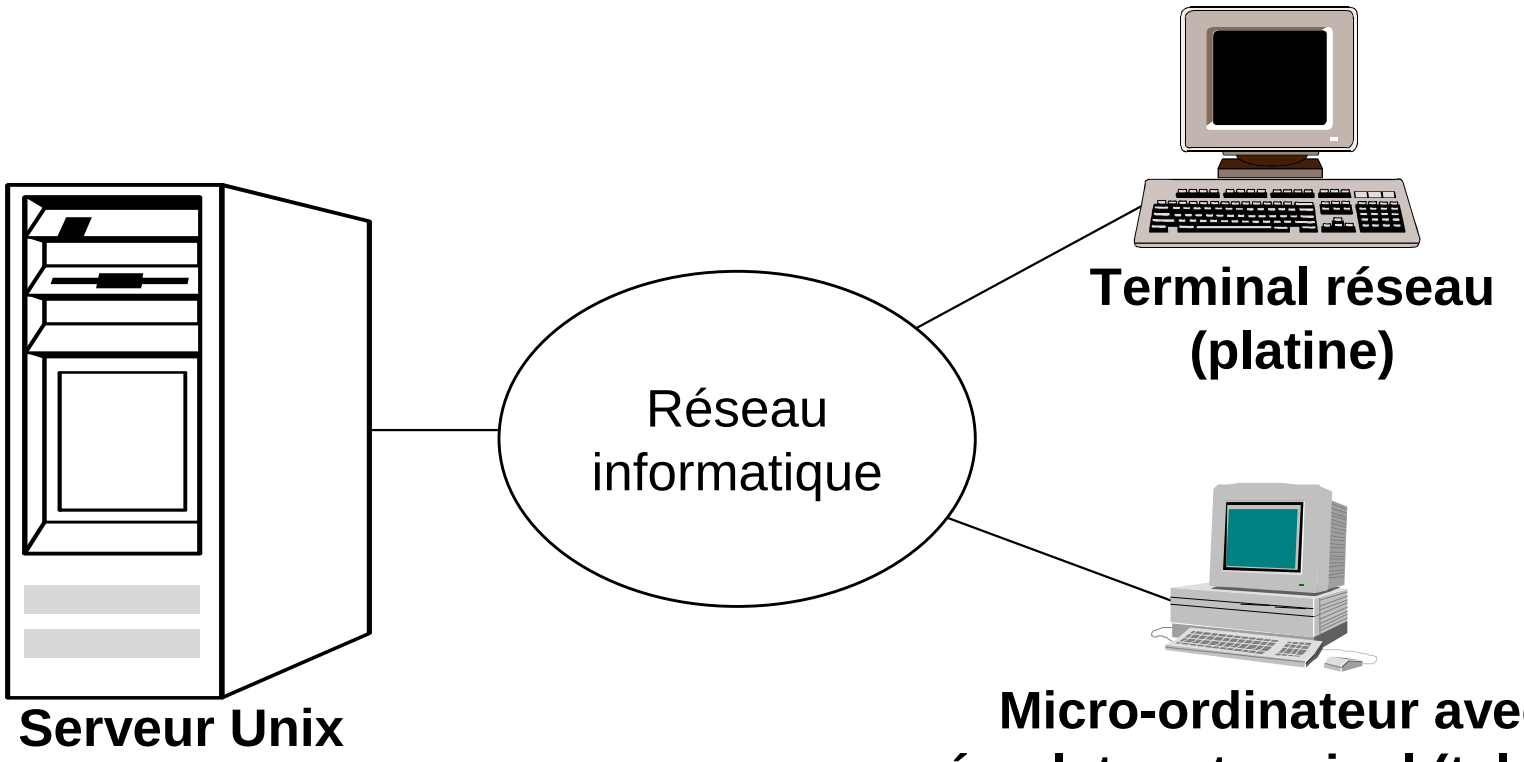
Rappels - Historique [2/5]

- évolution (initialement inventée par IBM dans le monde des mainframes et autres terminaux 3270): la gestion des terminaux n'est plus traitée par le serveur central (Rq : c'est déjà du client-serveur !) :



Rappels - Historique [3/5]

- autre évolution dans le monde Unix : le terminal qui cause à travers un réseau (telnet). Rq : c'est aussi du client serveur



Rappels - Historique [4/5]

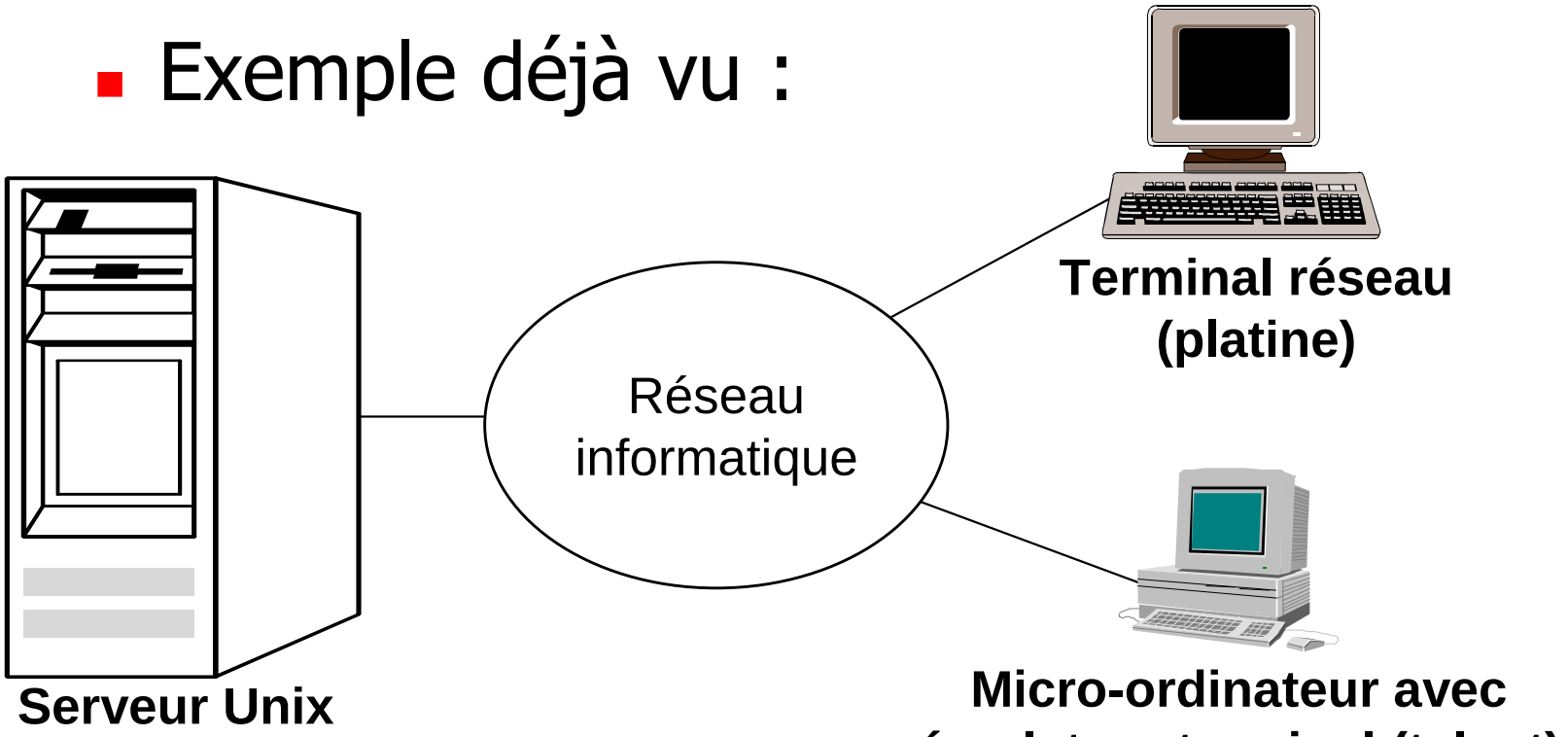
- La suite de l'histoire du client-serveur n'est pas linéaire. Il s'agit surtout de la naissance et de l'évolution de différents modèles, correspondant à différentes approches pour résoudre un problème. Parmi les idées développées :
  - certains proposent de concentrer la puissance dans le serveur, et de mettre des clients « idiots » (bien que nous verrons qu'ils deviennent de moins en moins idiots),
  - d'autres proposent de mettre l'intelligence dans le client, le serveur n'étant que la zone de stockage, et l'entité qui assure l'intégrité de l'ensemble,

Rappels - Historique [5/5]

- d'autres encore proposent de morceler l'application informatique en autant de clients et de serveurs nécessaires,
- Mais en fait... Il n'existe pas LA solution. Nous aborderons plus tard la notions d'« architecture », qui permet de mixer ces approches, afin de choisir le meilleur compromis à VOTRE problématique.

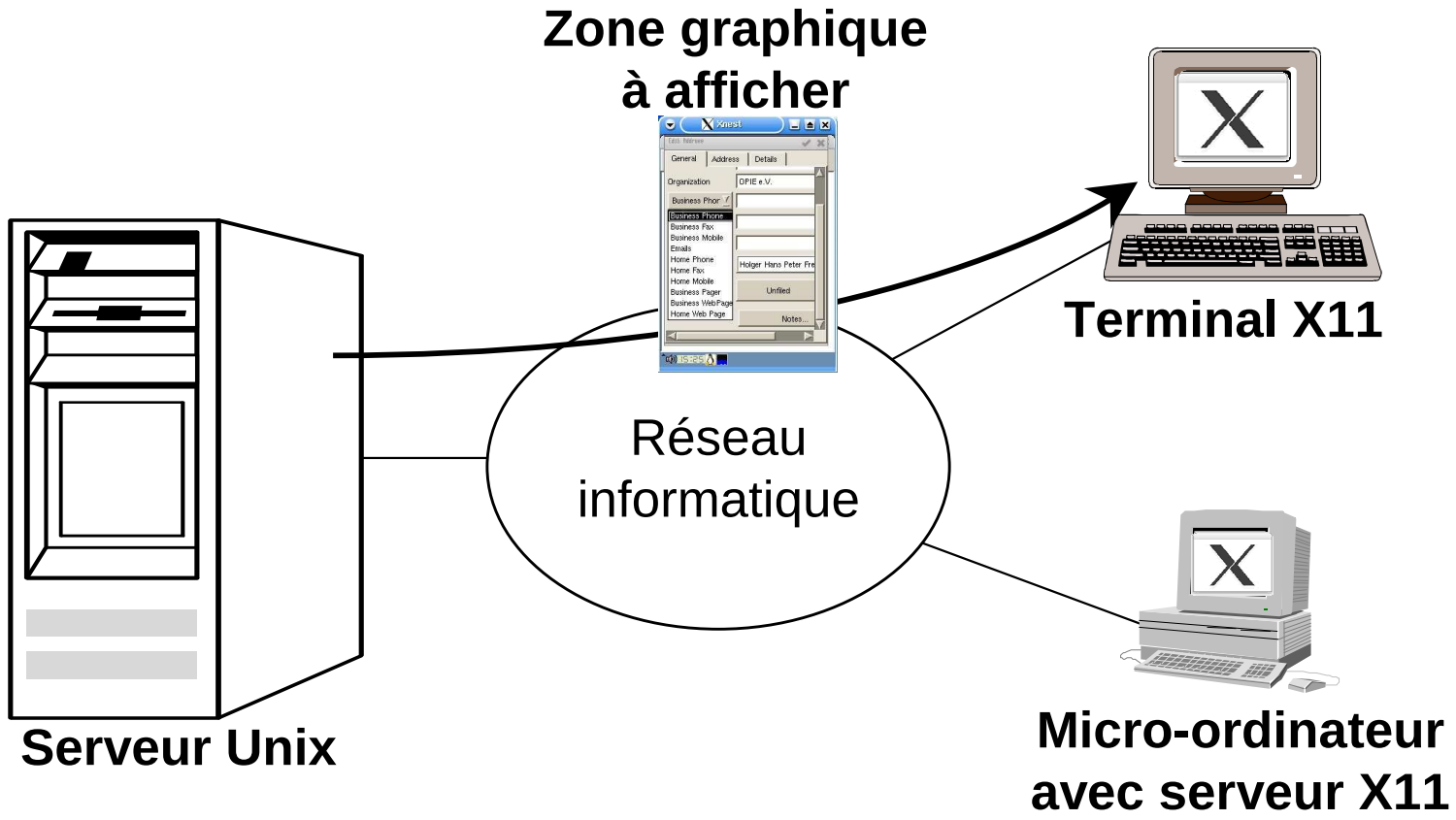
Les différentes approches [1/6]

- Une approche consiste à centraliser la puissance dans le serveur, et à avoir des clients « idiots » :
  - Exemple déjà vu :



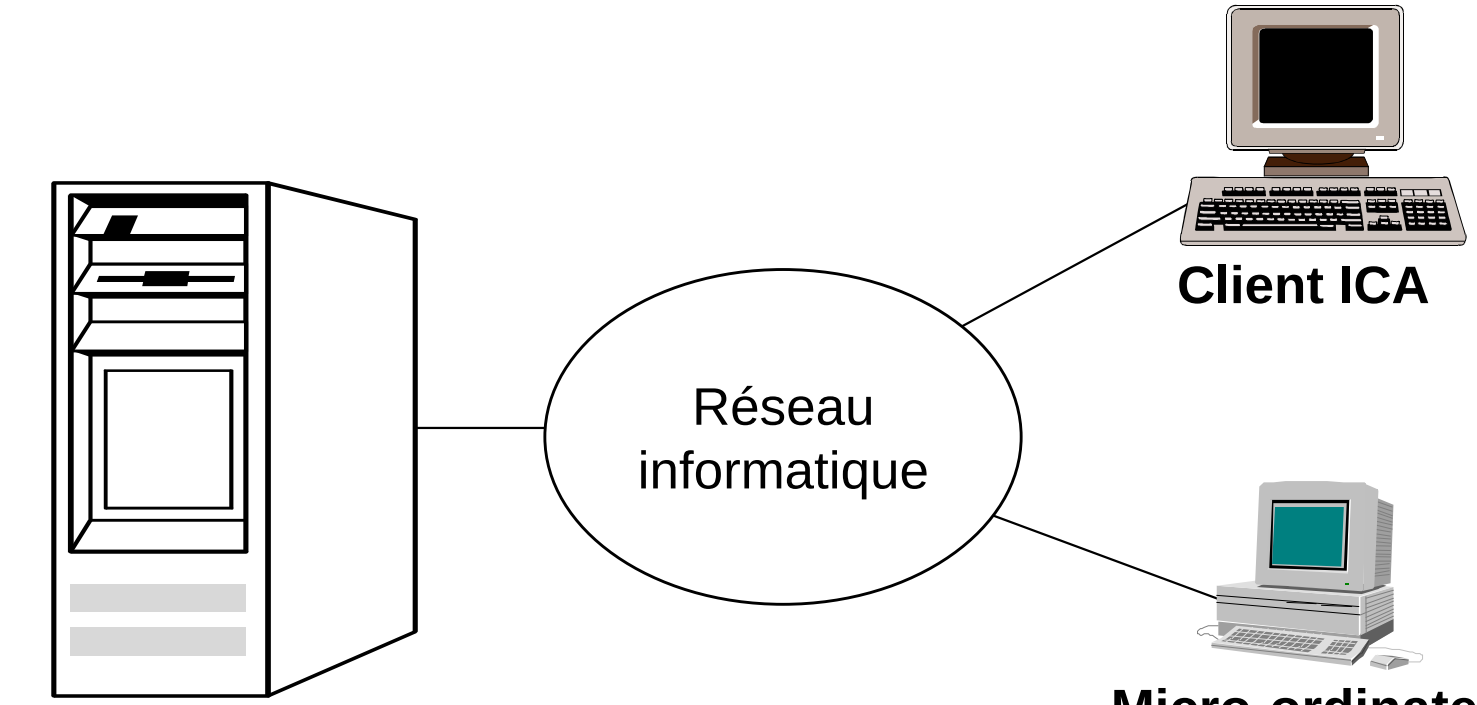
Les différentes approches [2/6]

- une extension de ce concept en version graphique : le terminal X11



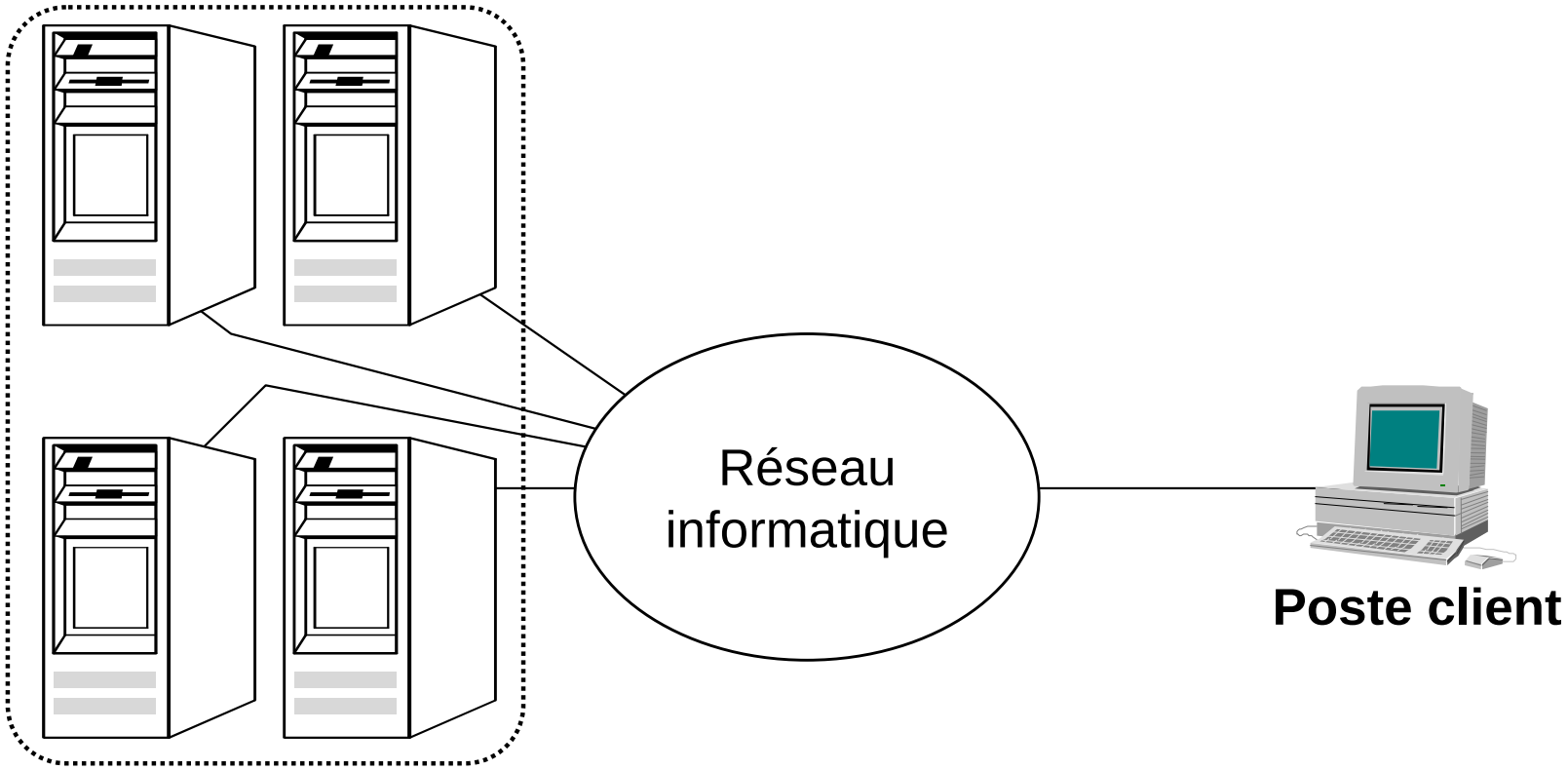
Les différentes approches [3/6]

- une extension du terminal X11 : le client léger (protocoles RDP ou ICA pour Windows, LNA pour Linux, ...). Exemple :



Les différentes approches [4/6]

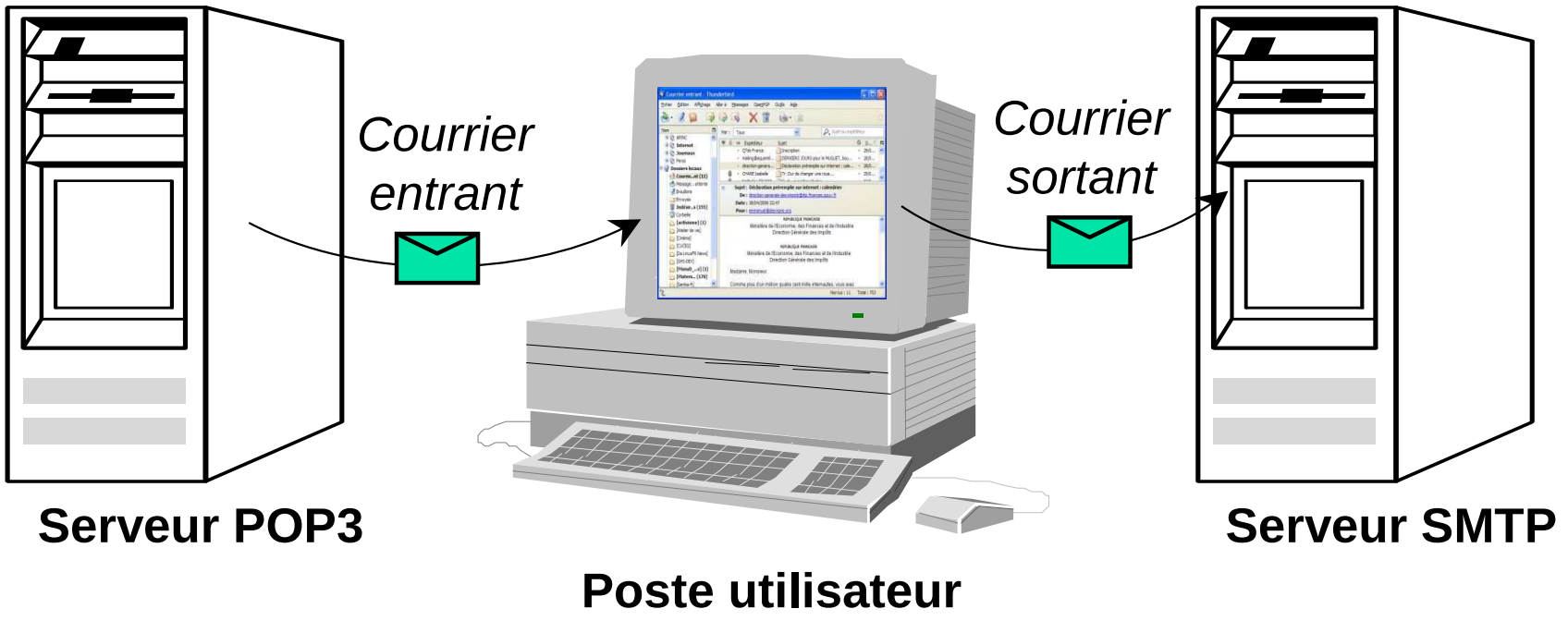
- Extension du modèle client-serveur appliqué aux serveurs : le « cluster »



Grappe de machines (cluster)
vue comme un seul serveur

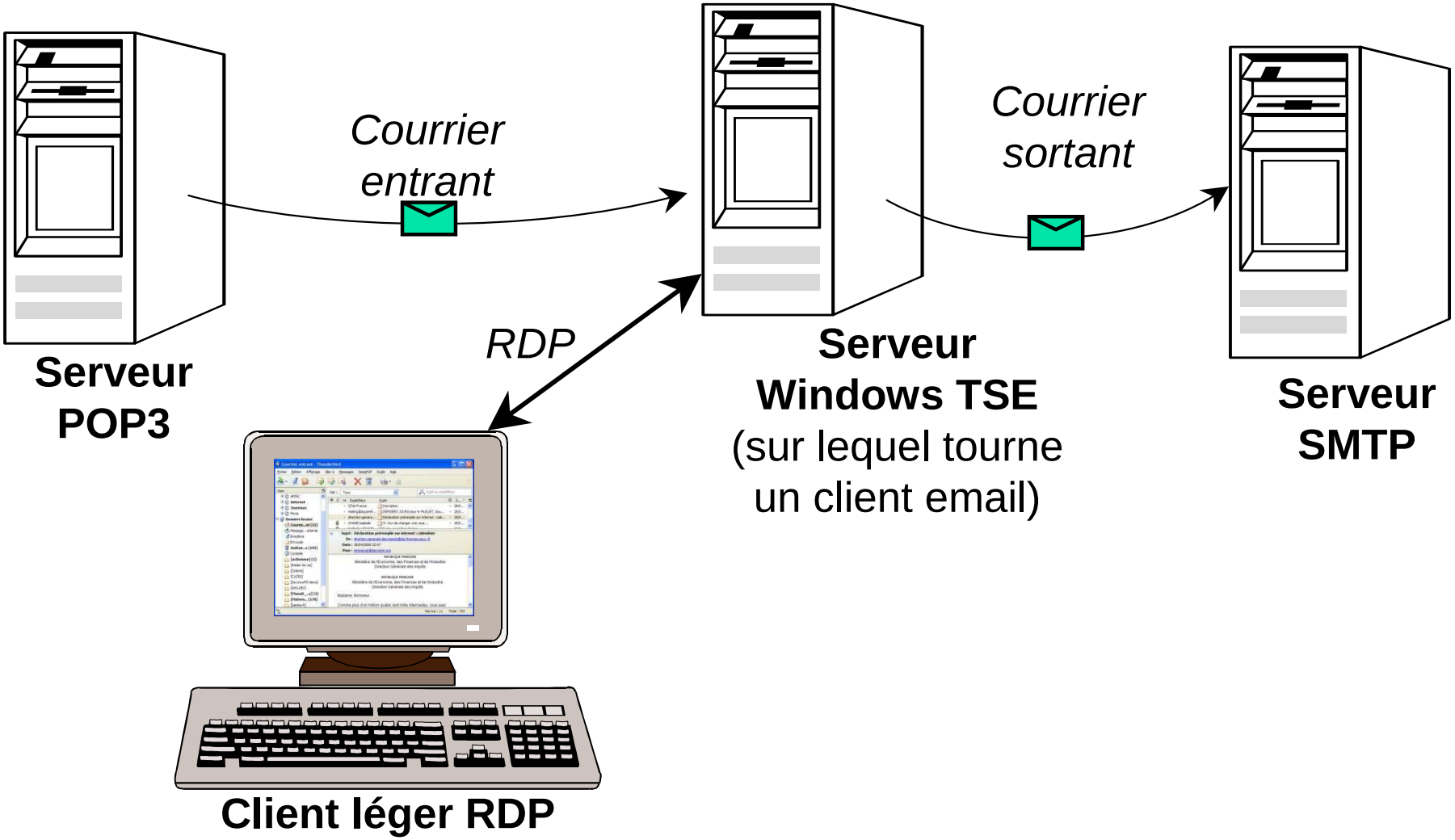
Les différentes approches [5/6]

- Une autre approche est de mettre une grande partie de la puissance dans le client. Exemple : client de courrier électronique avec POP3 et SMTP

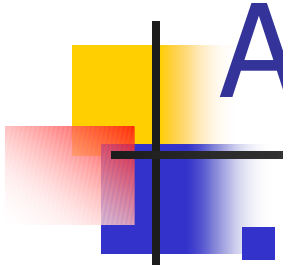


Les différentes approches [6/6]

- Mix des solutions



Architecture (initiation) [1/3]



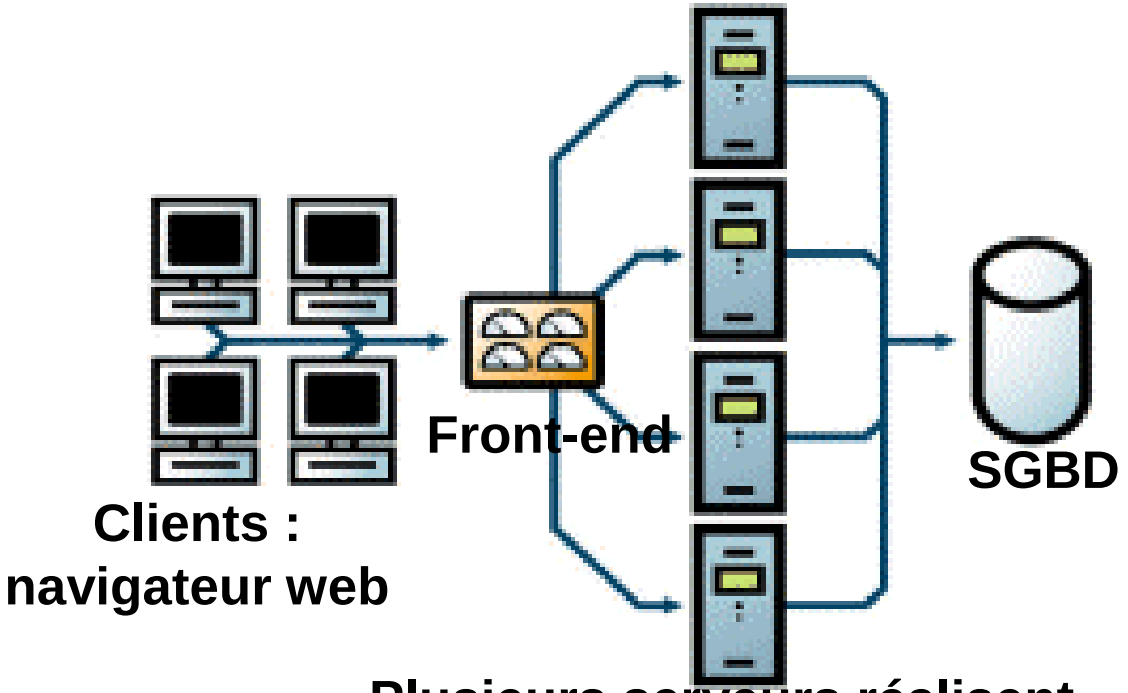
- Pour faire simple : c'est l'art de découper tout une application en composantes clients et en serveurs, de mixer les différentes approches client-serveur vues précédemment.
- Tout comme il n'existe pas UNE approche client-serveur parfaite, il n'existe pas UNE architecture unique ayant tous les avantages. Tout dépend de vos besoins.

Architecture (initiation) [2/3]

- Exemple (classique) : architecture 3-tiers :
  - **Couche présentation (ou cliente)** : correspondant à l'affichage, la restitution sur le poste de travail, le dialogue avec l'utilisateur, etc. (interface homme-machine)
  - **Couche métier (ou applicative)** : traitements propres à l'application (exemple : pour un logiciel de compta, gestion de stock, de la paie, gestion des états, de la balance…)
  - **Couche accès aux données** : correspondant aux données qui sont destinées à être conservées sur la durée, voire de manière définitive. Pour faire simple : mécanismes de cache + SGBD

Architecture (initiation) [3/3]

- Architecture N-tiers (multi-tiers) : la couche applicative est elle-même composée de plusieurs niveaux



Bilan [1/2]

- Intérêts de l'approche client-serveur (liste non exhaustive) :
  - Gagner de l'argent (Ooops…)
  - Optimiser l'achat de grosses et de petites machines en fonction du prix du marché (de moins en moins vrais)
  - Réutilisation des serveurs (un même service peut être utilisé par plusieurs applicatifs)
  - Répartition de la charge plus simple, et facilité d'ajouter des machines pour répondre à une montée en charge
  - Sécuriser (redondance de serveurs)

Bilan [2/2]

- Inconvénients (liste non exhaustive) :
  - Plus il y a de maillons, plus il y a risque de panne d'un élément dans la chaîne (mais il y a des solutions : redondance, virtualisation)
  - Problème de sécurité, à plusieurs niveaux (authentifications, gestion des droits d'accès, cryptage des données, et risque de trous de sécurité dans les APIs).

Questions/Réponses ?