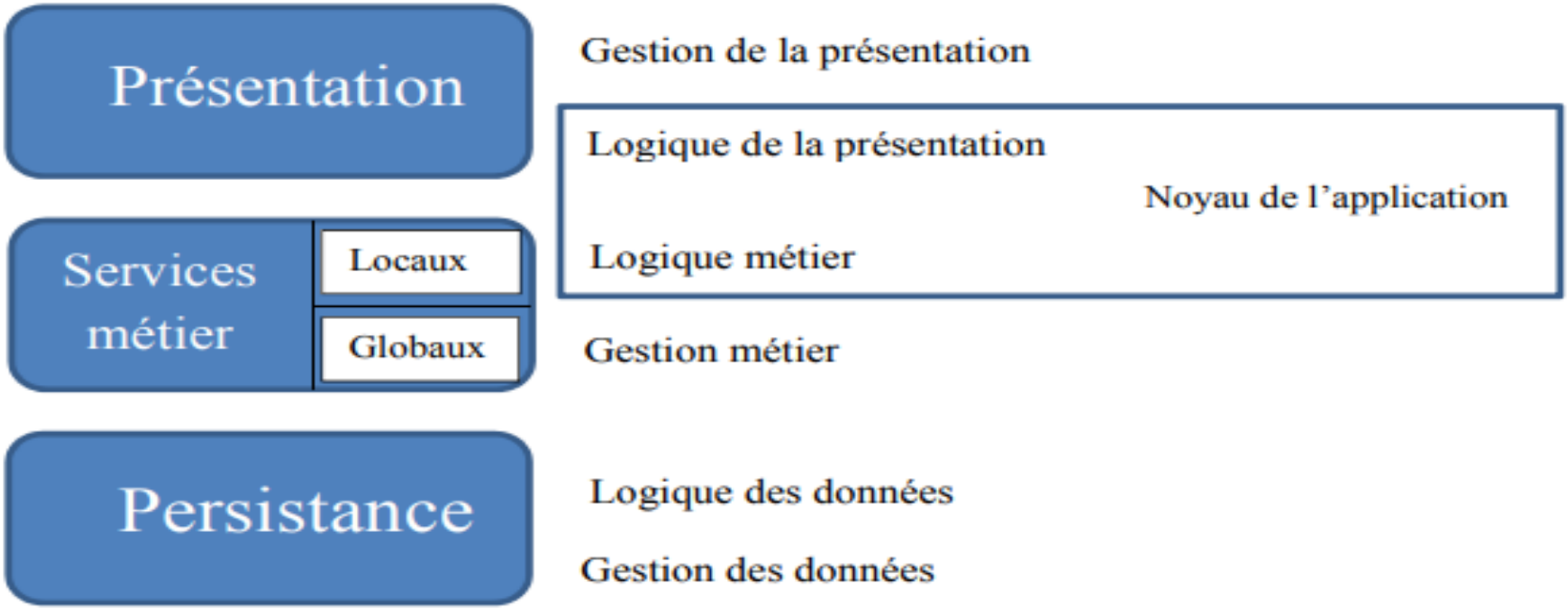
Chapitre 2
Architectures et modèles des systèmes répartis

Introduction

Couches fonctionnelles d'une application répartie

Toute application répartie est composée de 3 "couches" fonctionnelles:

• Présentation • Traitements métiers • Persistance de données



La répartition de ces couches entre les rôles de clients et serveurs qui permet de distinguer entre les différents types d'architectures.

Introduction

Couches fonctionnelles d'une application répartie

- **Présentation** : correspondant à l'affichage, la restitution sur le poste de travail, le dialogue avec l'utilisateur (= interface utilisateur).

- **Traitements métiers** : correspondant à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative.

- **Persistance de données**: correspondant aux données qui sont destinées à être conservées sur la durée, voire de manière définitive.

Introduction

Couches présentation

❖ Elle correspond à la partie de l'application visible et interactive avec les utilisateurs. On parle **d'Interface Homme Machine**.

❖ On conçoit facilement que cette interface peut prendre de multiples facettes sans changer la finalité de l'application. Dans le cas d'un système de distributeurs de billets, l'automate peut être différent d'une banque à l'autre, mais les fonctionnalités offertes sont similaires et les services sont identiques (fournir des billets, donner un extrait de compte, etc.).

❖La couche présentation relaie les requêtes de l'utilisateur à la couche métier, et en retour lui présente les informations renvoyées par les traitements de cette couche. Il s'agit donc ici d'un assemblage de services métiers et applicatifs offerts par la couche inférieure.

Introduction

Couches Traitements métiers

❖ Elle correspond à la partie fonctionnelle de l'application, celle qui implémente la “logique”, et qui décrit les opérations que l'application opère sur les données en fonction des requêtes des utilisateurs, effectuées au travers de la couche présentation.

❖ Les différentes règles de gestion et de contrôle du système sont mises en œuvre dans cette couche.

❖ La couche métier offre des services applicatifs et métiers à la couche présentation. Pour fournir ces services, elle s'appuie, sur les données du système, accessibles au travers des services de la couche inférieure. En retour, elle renvoie à la couche présentation les résultats qu'elle a calculés.

Introduction

Couches persistance

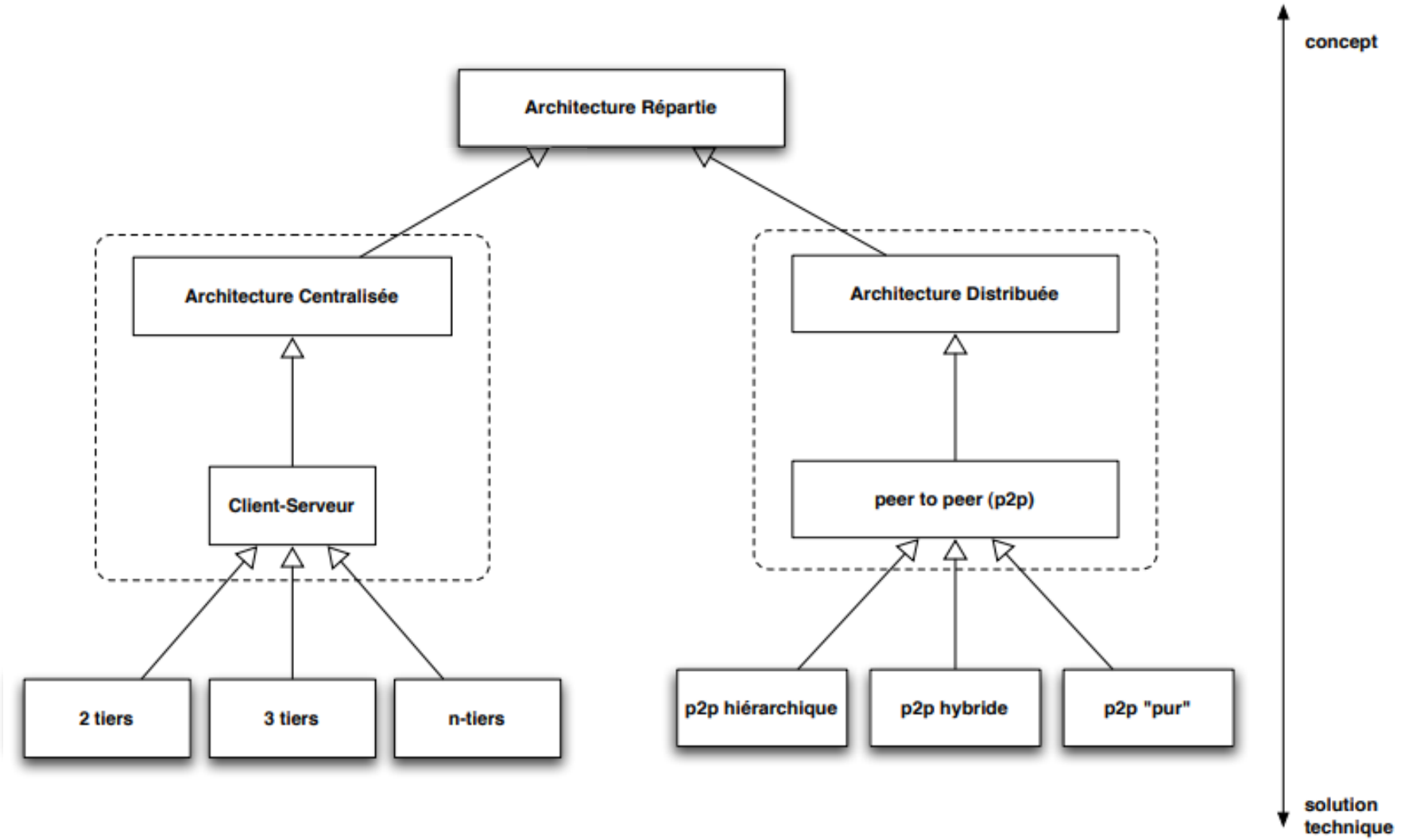
❖ Elle consiste en la partie gérant l'accès aux sources de données du système. Ces données peuvent être propres au système, ou gérées par un autre système. La couche métier n'a pas à s'adapter à ces deux cas, ils sont transparents pour elle, et elle accède aux données de manière uniforme.

➢ **Données propres au système**: ces données sont stables, car destinées à durer dans le temps, de manière plus ou moins longue, voire définitive. Elles peuvent être stockées indifféremment dans de simples fichiers texte, XML, ou encore dans une base de données.

➢ **Données gérées par un autre système**: elles ne sont pas stockées par le système considéré, il s'appuie sur la capacité d'un autre système à fournir ces informations.

Introduction

Architectures distribuées



Architecture Clients/Serveurs

Problématique

❖ Il s'agit de permettre à des programmes/applications/entités logicielles d'échanger de l'information entre plusieurs machines reliées par un réseau

- ➢ à large échelle (Internet)
- ➢ en local (Intranet)

❖Séparation des rôles inhérente à la plupart des systèmes informatiques :

- ➢ Certaines entités **fournissent des services** (de calcul, de données,...)
- ➢ Certaines entités souhaitent **accéder à ces services** (pour soumettre des requêtes, des calculs, afficher des données,...)

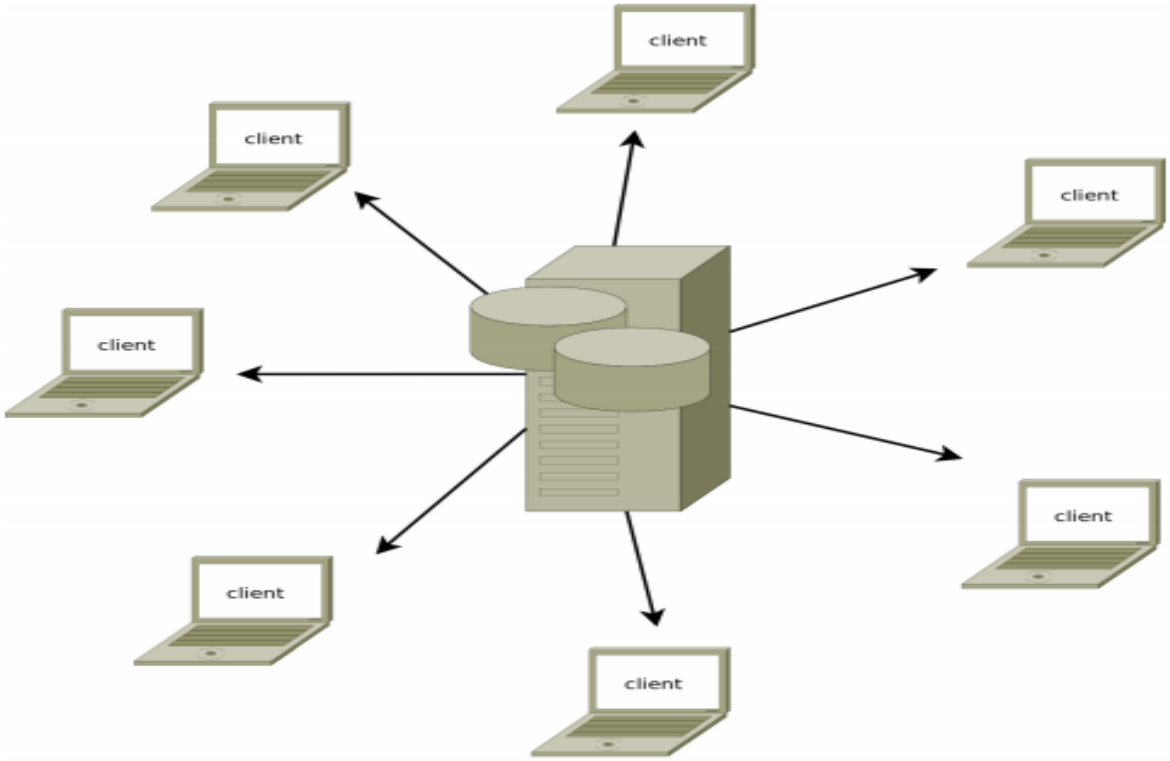
❖ Comment cela se traduit-il du point de vue de l'architecture logicielle ?

Architecture Clients/Serveurs

Définition

❖L'architecture client serveur s'appuie sur un **poste central**, le serveur, qui **envoie des données** aux machines clientes.

❖Des programmes qui accèdent au serveur sont appelés **programmes clients** (client FTP, client mail, navigateur).



Architecture Clients/Serveurs

Définition

❖ L'architecture client-serveur fait une distinction stricte entre les **rôles** de **client** d'une part et de **serveur** d'autre part.

- ➢ D'un point de vue matériel : machines clientes et machines serveurs)
- ➢ D'un point de vue logiciel: entités clientes et entités serveurs d'une même application.

❖ Les interprétations matérielles et logicielles sont souvent liées: **une application client-serveur est communément répartie sur plusieurs machines connectées par un réseau.**

Architecture Clients/Serveurs

Qu'est-ce qu'un serveur ?

- Un serveur est un programme qui offre un service sur le réseau.
- Le serveur accepte des requêtes, les traite et renvoie le résultat au demandeur.
  - Une requête est un appel de fonction, la réponse éventuelle pouvant être synchrone ou asynchrone (le client peut émettre d'autres requêtes sans attendre)
  - Les arguments et les réponses sont énoncés dans un protocole
- Le terme serveur s'applique à la machine sur lequel s'exécute le logiciel serveur.
- Pour pouvoir offrir ces services en permanence, le serveur doit être sur un site avec accès permanent .

Architecture Clients/Serveurs

Qu’est-ce qu’un client?

❖ Un logiciel client est un programme qui utilise le service offert par un serveur.

❖ Le client envoie une requête et reçoit la réponse.

❖ Il peut-être raccordé par une liaison temporaire.

❖ Il interagit généralement directement avec un utilisateur final en utilisant une IHM

Communication

❖C’est la description du fonctionnement coopératif entre le serveur et le client.

❖Les services internet sont conçus selon cette architecture. Chaque application est composée de logiciel serveur et logiciel client.

❖ Un logiciel serveur, peut correspondre plusieurs logiciels clients développés dans différents environnements: Unix, Mac, PC... la seule obligation est le respect du protocole entre les deux processus communicants.

❖Ce protocole étant décrit dans un RFC (Request For Comment).

Architecture Clients/Serveurs

Caractéristiques

- **Serveur**
  - initialement passif en attente d'une requête I à l'écoute,
  - prêt à répondre aux requêtes clients
  - quand une requête lui parvient il la traite et envoie la réponse

- **Client**
  - actif en premier
  - envoie des requête au serveur
  - attend et reçoit les réponse du serveur

- Le clients et le serveur doivent utiliser le même protocole
- Un serveur peut répondre à plusieurs clients en simultané

Avantages/Inconvénients

Avantages

- Unicité de l'information : toutes les données sont stockées sur un même serveur.
- Meilleure sécurité : simplification des contrôles de sécurité.
- mise à jours : mise à jour centralisé aussi bien des données et logiciels.
- Meilleure fiabilité : En cas de panne, seul le serveur fait l'objet d'une réparation.
- Facilité d'évolution : architecture évolutive, il est très facile de rajouter ou d'enlever des clients ou des serveurs.
- Architecture plus mature que les autres

Avantages/Inconvénients

Inconvénients

- Un coût d'exploitation élevé (bande passante, câbles, ordinateurs surpuissants).

- En cas de panne du serveur, les clients ne peuvent accéder aux informations.

- si trop de clients veulent communiquer avec le serveur ce dernier risque de ne pas supporter la charge.

Architecture Clients/Serveurs

Mise en œuvre

❖ Besoin d'un support pour transporter les informations entre le client et le serveur

- **Bas niveau**

  Utilisation directe du transport : **sockets** (construits sur TCP ou UDP)

- **Haut niveau**

  Intégration dans le langage de programmation : **RPC** ou Remote Procedure Call (construits sur sockets)

❖ Nécessité d'établir un protocole entre le client et le serveur pour qu'ils se comprennent

Architecture n-tiers

Architecture 2-tiers

❖Client / serveur de base, avec 2 éléments

1. **Client** : présentation, interface utilisateur.
2. **Serveur** : partie persistance, gestion physique des données.

❖ Les services métier / la partie applicative peuvent être :

➢ Soit entièrement coté client, intégrés avec la présentation . La partie serveur ne gère que les données .

Ex : traitement de texte avec serveur de fichiers distants
Ex : application accédant à une BDD distante

Architecture 2-tiers

➢ Soit entièrement coté serveur.

✓La partie cliente ne gère que l'interface utilisateur .

✓ L'interface utilisateur peut même être exécutée sur le serveur.

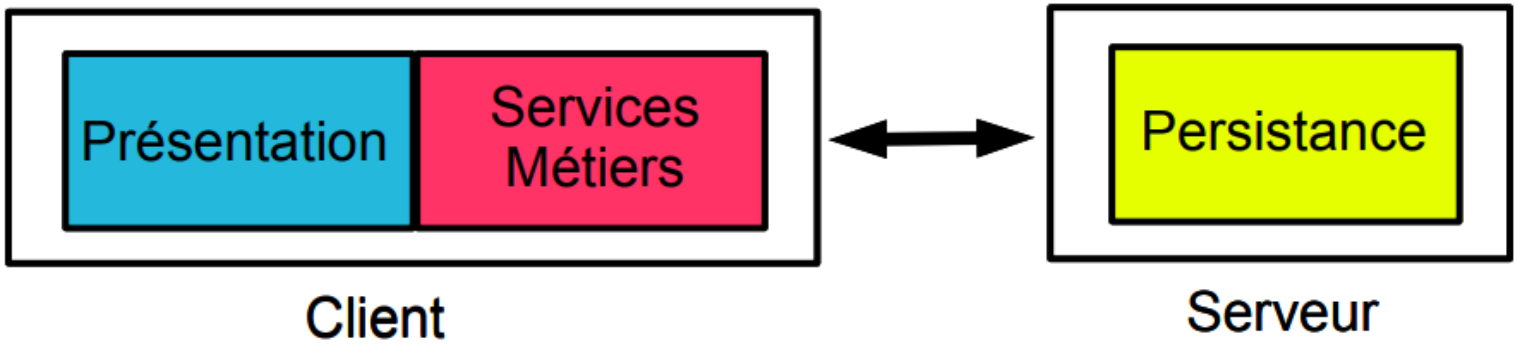
- Fonctionnement mode terminal / mainframe
- L'utilisateur a simplement devant lui écran / clavier / souris pour interagir à distance avec l'application s'exécutant entièrement sur le serveur

➢ Soit découpés entre la partie serveur et la partie client

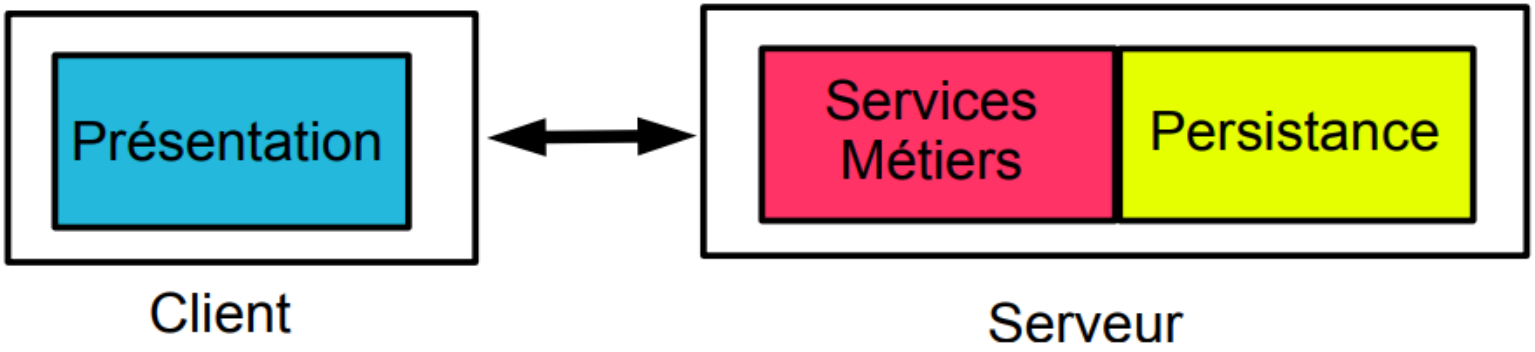
Architecture n-tiers

Architecture 2-tiers

- Client : présentation + applicatif

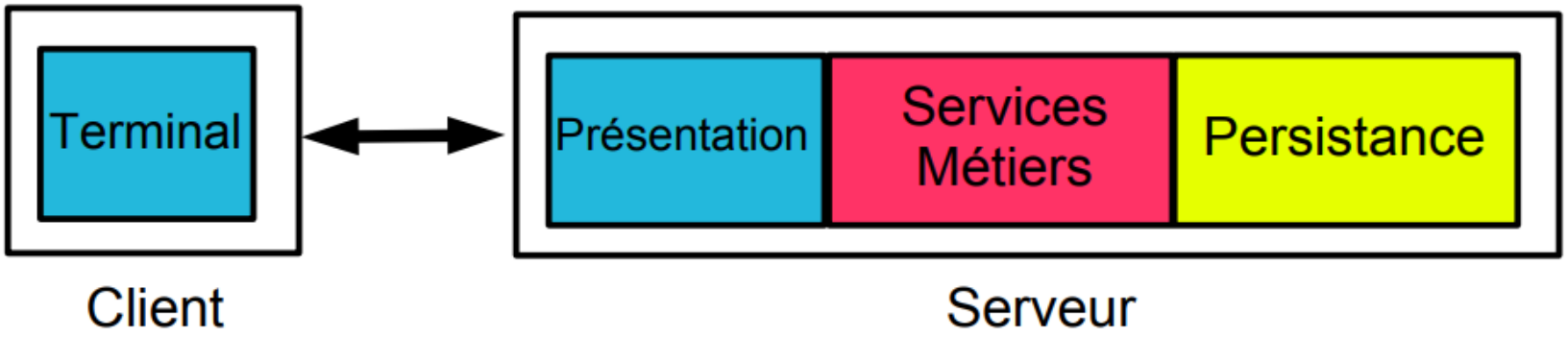


- Serveur : applicatif + gestion données

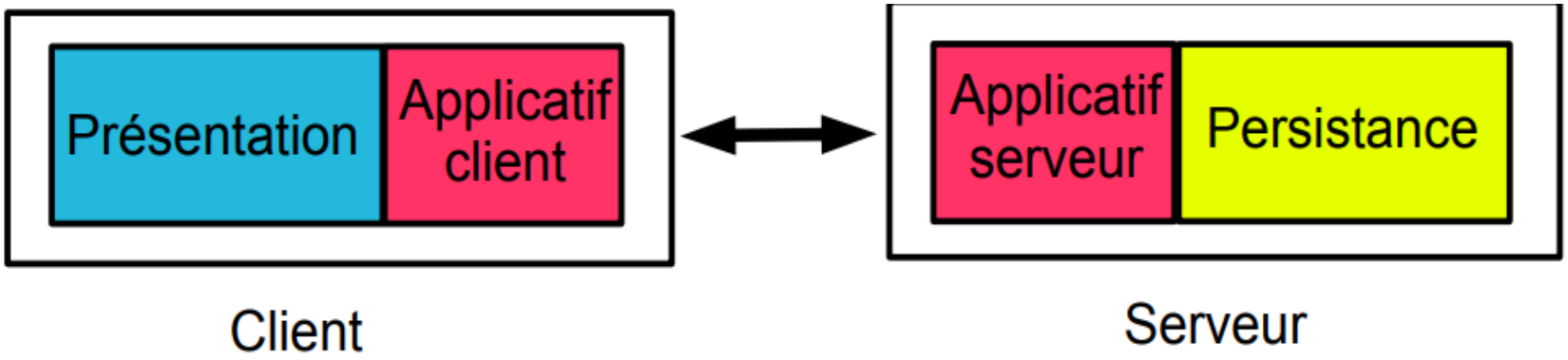


Architecture 2-tiers

- Terminal : client intègre un minimum de la partie présentation



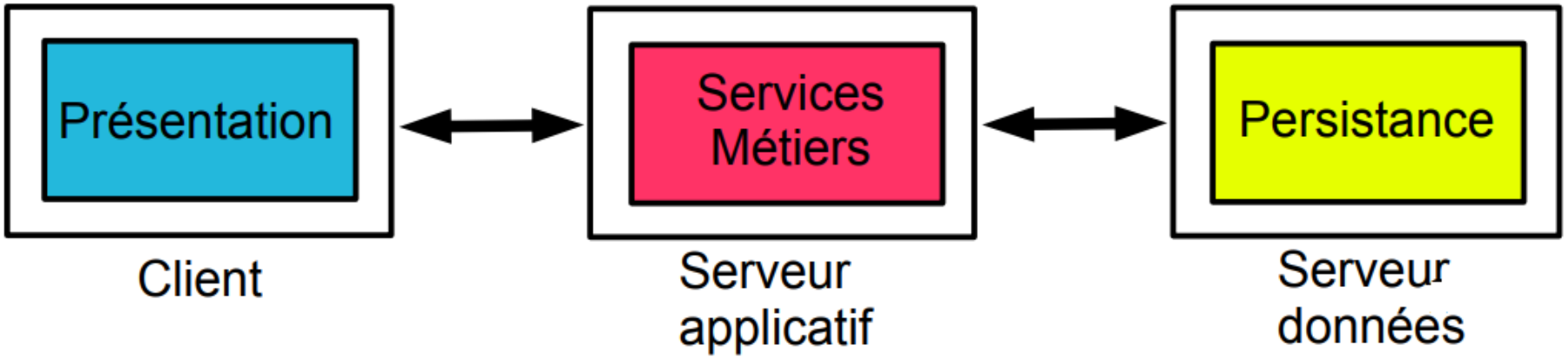
- Applicatif : découpé entre client et serveur



Architecture 3-tiers

➢Les 3 principaux tiers s'exécutent chacun sur une machine différente :

- ➢ Présentation : Machine client
- ➢ Applicatif / métier : Serveur d'applications
- ➢ Persistance : Serveur de (base de) données



Architecture n-tiers

Architecture 3-tiers

Très développée de nos jours, avec généralement un fonctionnement au dessus du Web

➢ **Couche présentation**:

• Navigateur web sur machine cliente , Client léger .

•Affichage de contenu HTML .

➢**Couche applicative /** métier :

• Serveur d'applications

1. Serveur HTTP exécutant des composants / éléments logiciels qui génèrent dynamiquement du contenu HTML
2. Via des requêtes à des BDD distantes

➢**Couche persistance :**

• Serveur(s) de BDD de données

Architecture n-tiers

- Rajoute des étages / couches en plus
- La couche applicative n'est pas homogène
- ✓ Peut s'appuyer et interagir avec d'autres services

✓Composition horizontale

Service métier utilise d'autres services métiers

✓Composition verticale

Les services métiers peuvent aussi s'appuyer sur des services techniques : Sécurité, Transaction,...

✓Chaque service correspond à une couche, d'où le terme de N-tiers

Architecture n-tiers

Intérêts d'avoir plusieurs services / couches (3 ou plus) Réutilisation de services existants :

- Découplage des aspects métiers et technique et des services entre eux : meilleure modularité
- Facilite évolution : nouvelle version de service
- Facilite passage à l'échelle : évolution de certains services
- On recherche en général un couplage faible entre les services
- Permet de faire évoluer les services un par un sans modification du reste de l'application.

Inconvénients:

En général, les divers services s'appuient sur des technologies très variées : nécessite de gérer l'hétérogénéité et l'interopérabilité

- Utilisation de framework / outils supplémentaires
- Les services étant plus découpés et distribués, pose plus de problèmes liés à la distribution

Architectures distribuées

Peer to peep

Dans les architectures distribuées, tous les nœuds du réseau partagent théoriquement les mêmes rôles : ils sont simultanément clients et serveurs et disposent de tout ou partie de l'information répartie. Les nœuds du réseau sont en relation d'égal à égal.

Les caractéristique habituelles de ces architectures sont les suivantes :

- Pas de connaissance globale du réseau.
- Pas de coordination globale des nœuds.
- Chaque nœud ne connaît que les nœuds constituant son voisinage.
- Toutes les données sont accessibles à partir de n'importe quel nœud.
- Les nœuds sont très volatiles (ils peuvent disparaître ou apparaître à tout moment).

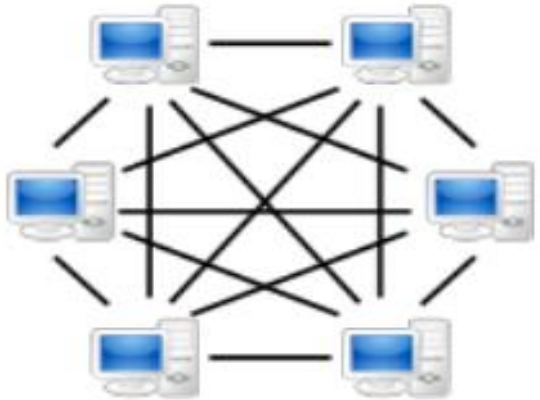
Architectures distribuées

Peer to peep

• Les technologies Peer-to-Peer (P2P, ou Pair-à-Pair) constituent le pendant technologique des architectures distribuées.

• La parfaite homogénéité des rôles vue précédemment correspond rarement à la réalité des réseau P2P déployés de part le monde. De ce fait, on procède à leur classification en différentes grandes familles (cf. transparent suivant).

Architecture centralisée
(client-serveur en étoile)

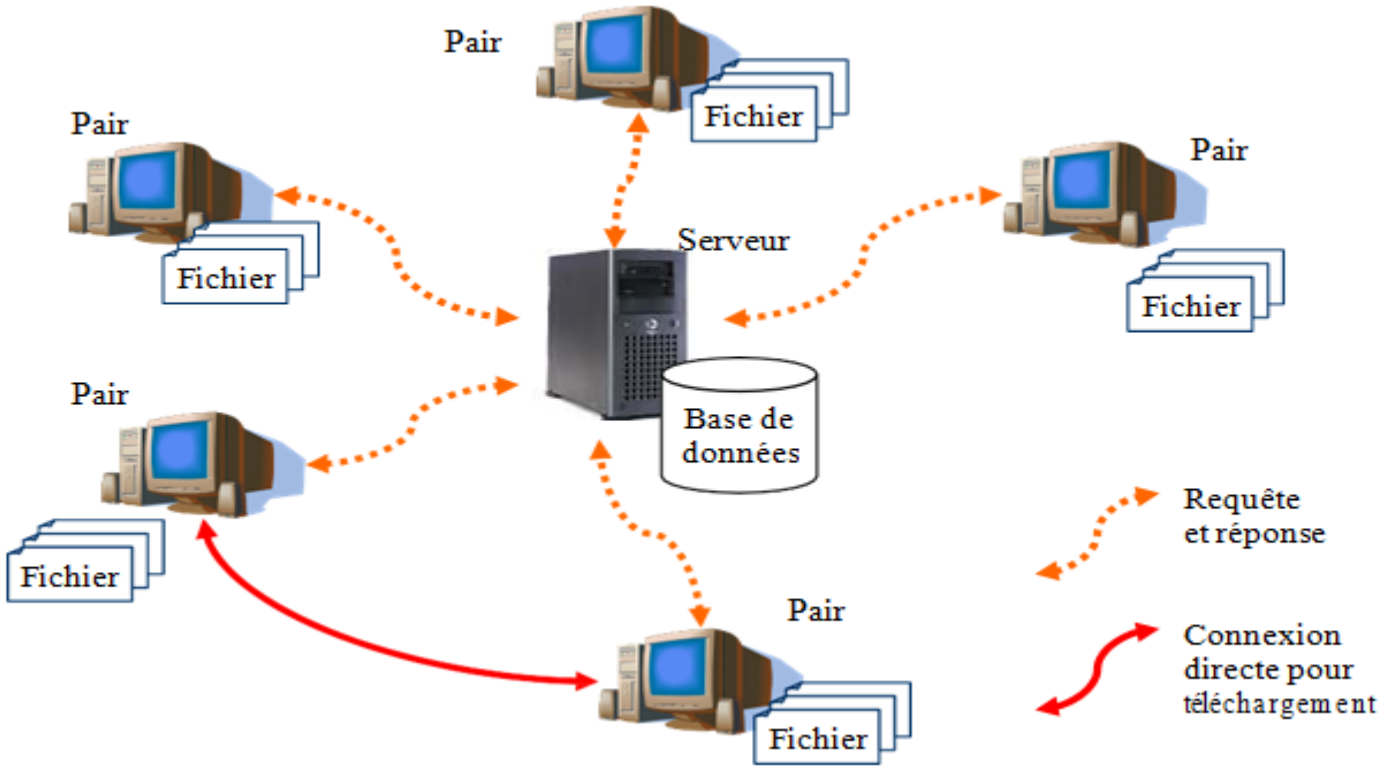


Architecture distribuée
(Peer-to-Peer pur)

Architectures distribuées

Peer to peep

Architecture centralisée

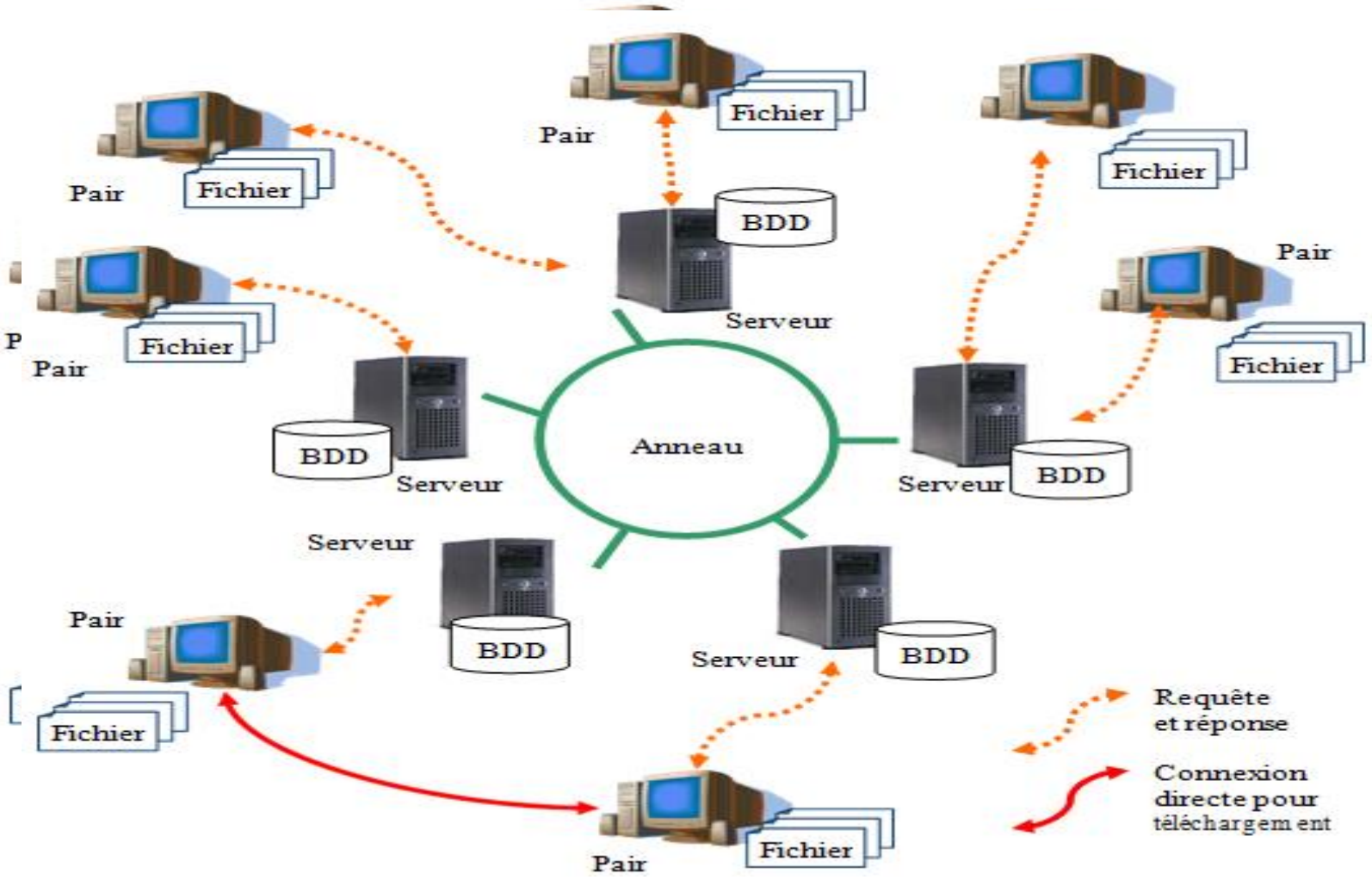


e-Mule: est la version actuelle de e-Donkey qui est né en septembre 2000. Il adopte une architecture centralisée avec une multitude de serveurs et permet le transfert de tous types de fichiers en utilisant des serveurs pour les plateformes windows et Unix.

Architectures distribuées

Peer to peep

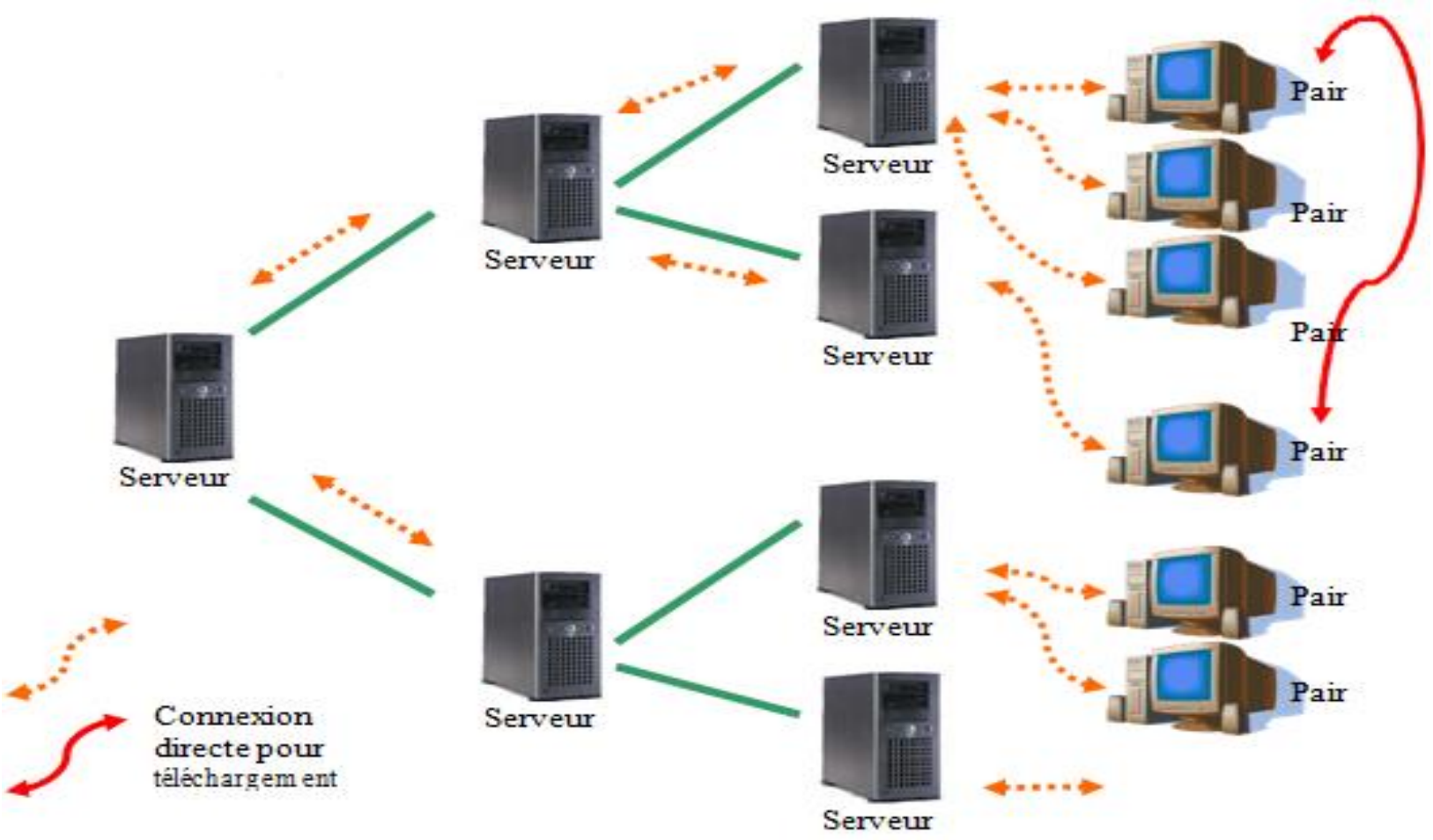
Architecture en anneau



Architectures distribuées

Peer to peep

Architecture hiérarchique



Applications: Super-Peer, Kazaa

Architectures distribuées

Peer to peep

Avantages :

• Tous les pairs fournissent des ressources (bande passante, stockage, puissance de calcul,...). On obtient ainsi des architectures qui supportent beaucoup mieux la montée en charge (“scalability”) que les architectures centralisées.

• La distribution augmente la robustesse du réseau dans le cas d’une panne par la réplication des données sur plusieurs pairs. Et, dans le cas d’un P2P pur, il n’y pas de point central de vulnérabilité (l’annuaire).

Inconvénients :

• Mauvaise réputation du “P2P”, invariablement associé au téléchargement musical → faible adoption par l’industrie.

• Les architectures distribuées amènent leur lot de problématiques spécifiques (concurrence entre les pairs, fragmentation des données, ...).

• Elles ne permettent pas un contrôle avancé des échanges d’information entre les pairs (inconvénient ou avantage ?).