Architectures Réparties
Clients-Serveurs

Pr. Younès EL BOUZEKRI EL IDRISSI
Pr. Rachida AJHOUN

y.elbouzekri@gmail.com

Année universitaire 2017-2018

Objectifs

- Comprendre les concepts de base du modèle C/S;
- Comprendre les techniques de développement des applications C/S;
- Utilisation d'un Middleware;
- Étude de l'architecture Web;
- Futur Client/Serveur.

INTRODUCTION AU CLIENT-SERVEUR

3

Historique

- 1 étape:
  - 1 seule (énorme) machine, 1 seul utilisateur (son concepteur)
  - E/S: interrupteurs, voyants lumineux
- 2 étape:
  - Amélioration des E/S: lecteur (de cartes!), imprimante
  - Multi-utilisateurs, 1 programme à la fois
- 3 étape:
  - Montée en puissance : multi-exécution
  - Temps partagé, Apparition des terminaux
- 4 étape:
  - PCs des années 80
  - Emulateurs de terminaux pour les relier aux systèmes centraux
  - En // : évolution des réseaux pour permettre aux systèmes centraux de communiquer
- 5 étape:
  - Stations de travail, Terminaux X

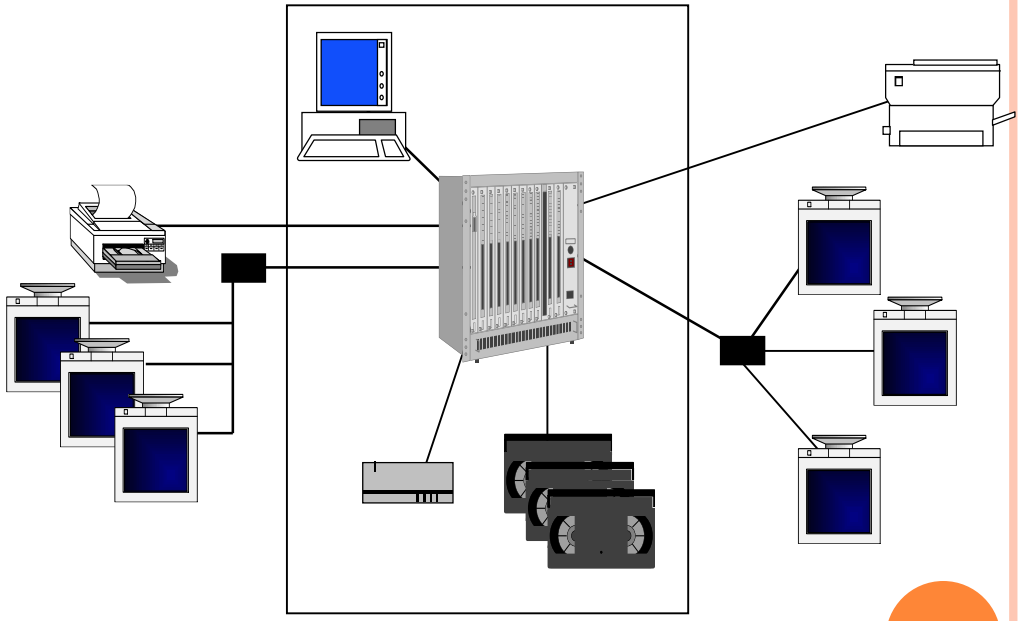
Historique

- Systèmes centralisés des années 1970

  (Gros systèmes)

Architecture centralisée:

- terminaux passifs, mono-technologie, mono OS, système propriétaire,…
- BD non relationnelles: IDS2, DB2,…
- langages de programmation: COBOL,…

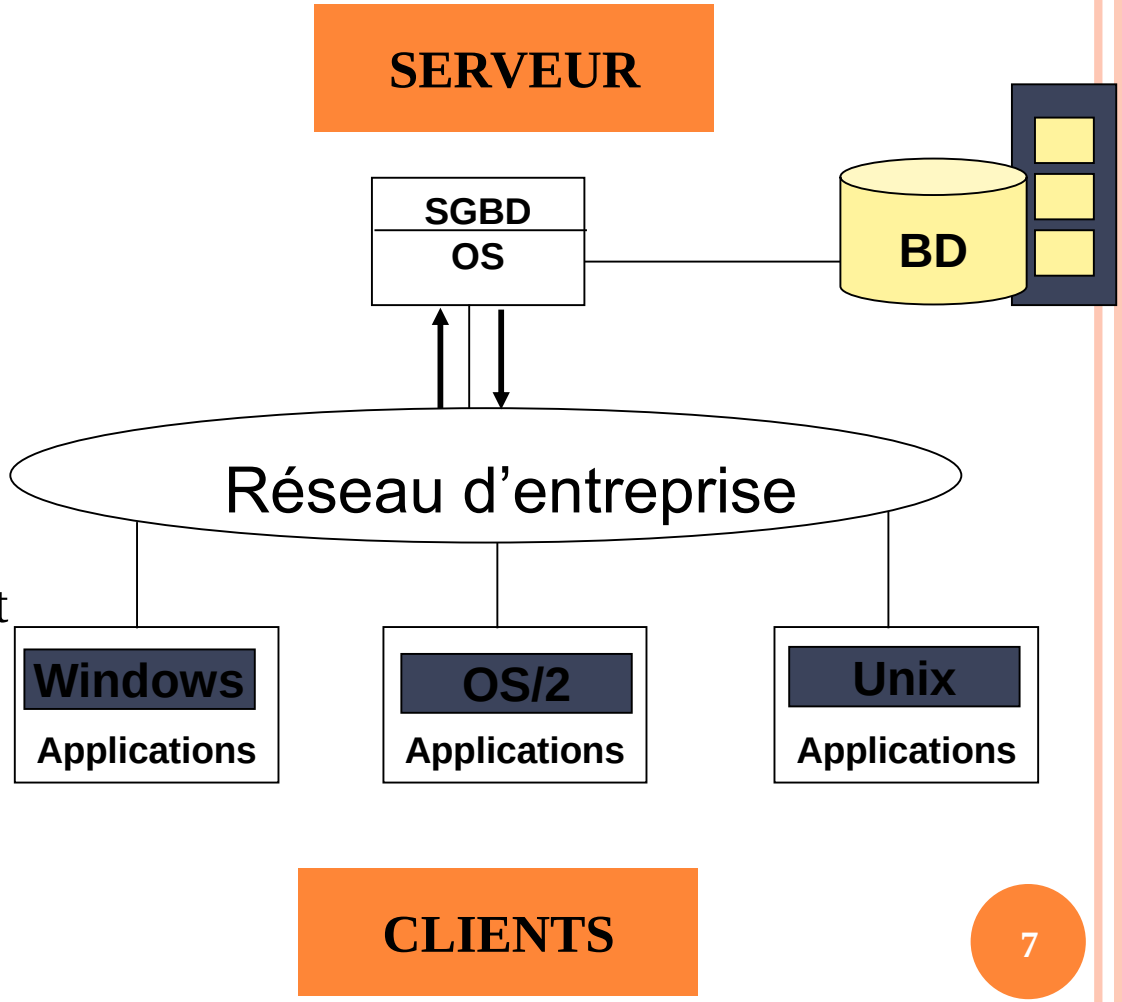


Besoins

- Consiste à réaliser une intégration de l'informatique personnelle dans le SI de l'entreprise avec les objectifs suivants:
  - Tout utilisateur dans l'entreprise doit pouvoir accéder à toute information utile à sa tâche dès lors que cet accès est autorisé par les règles de confidentialité et de sécurité.
  - L'accès doit être instantané et doit pouvoir être fait à partir de n'importe quel poste de travail.
  - L'accès à l'information doit avoir lieu par une interface aussi simple que possible, choisie par l'utilisateur.
  - Il faut préserver les investissements passés.

Evolution

- Architecture répartie:
  - Réseaux, PC plus puissants, OS ouverts
  - Interfaces et API standards,
  - BD relationnelles
  - Langages 4GL: SQL, …
  - Outils de développement
  - Outils de transport



La solution client/serveur

- L'architecture Client/Serveur est l'aboutissement d'un ensemble d'évolutions technologiques survenues dans les années 80:
  - capacités mémoires,
  - performances des processeurs et des réseaux,
  - évolutions des logiciels : interfaces graphiques, multimédia, des interfaces de communications.
- Définition : Modèle d'architecture applicative où les programmes sont répartis entre processus clients et serveurs communiquant par des requêtes avec réponses

N.B: *le réseau n'est pas vraiment nécessaire*

Downsizing

- Définition

  Migration d'une application d'une plate-forme informatique de type grand système vers une plate-forme plus petite et économique. Les modèles et fonctionnalités ne sont pas nécessairement modifiés par cette opération.

- Quel rapport avec C/S ?

  A priori aucun, le C/S est un modèle de *fonctionnement*, donc une technique alors que le downsizing est une *pratique*.

Caractéristiques du modèle C/S

- **Notion de service:** le serveur est fournisseur de service. Le client est consommateur de services.
- **Partage des ressources**: un serveur traite plusieurs clients en même temps et contrôle leurs accès aux ressources.
- **Redimensionnement**: il est possible d'ajouter et de retirer des stations clientes. Il est possible de faire évoluer les serveurs.
- **Intégrité**: les données du serveur sont gérées sur le serveur de façon centralisée. Les clients restent individuels et indépendants.

UNE GRANDE DIVERSITÉ D'OUTILS

- **Il comporte les composants suivants:**
  - Un système ouvert: basé sur les standards;
  - Un SGBD s'exécutant sur le serveur;
  - Des stations de travail personnelles avec interface graphique connectées au réseau;
  - Des outils de développement d'applications;
  - Des logiciels de transport de requêtes et réponses associés au SGBD ou indépendant;
  - Des outils de conception, de déploiement et de maintenance pour permettre le suivi du cycle de vie des applications;

Pourquoi le Client/Serveur pour l'entreprise ?

- Adapté à l'organisation des sociétés modernes

  ==> structurées en entités de moindre taille

  ==> nouveaux besoins de communication

- Gérer les contraintes de l'entreprise
  - mieux satisfaire les clients
  - faire face à l'évolution des règles
  - produire mieux et plus vite
- Mieux maîtriser le système d'informations
- Prendre en compte l'évolution des technologies

Exemples d'application

- Serveur d'impression
- Serveur Web
- Serveur d'autorisation de cartes bancaires
- Serveur de mail
- Traitement d'un appel (établissement,raccrochage...) dans un commutateur téléphonique
- Localisation d'un mobile GSM lors d'un appel vers ce mobile

ARCHITECTURE CLIENT-SERVEUR

1. DEFINITIONS
2. TECHNIQUES DE DIALOGUE CLIENT-SERVEUR
3. DIFFERENTS TYPES DE CLIENT-SERVEUR

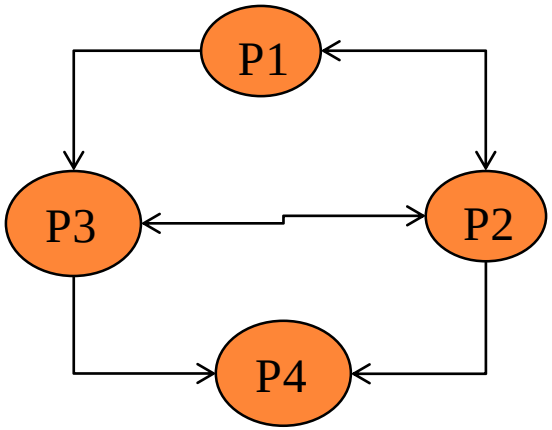
Rappel: Architectures classiques des Applications réparties

- **Couplage fort**
  - Ex : Architecture « peer-to-peer» (tous les processus ont le même rôle)
  - Grappes de calculateurs

- **Couplage faible**
  - Architecture client/serveur

COUPLAGE FORT

- Interdépendance entre les composants de l'application



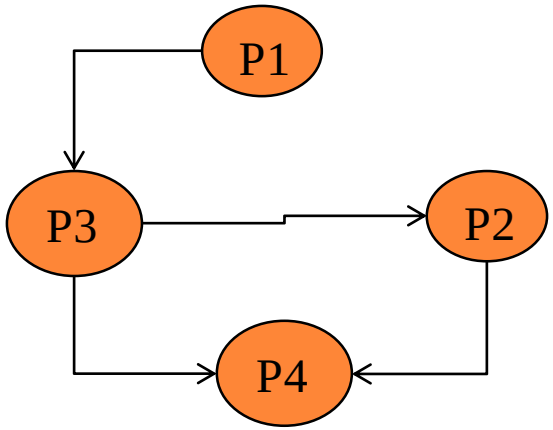
Il existe au moins un cycle dans le graphe de dépendances entre les composants de l'application

Ce type d'architecture pose problème tant pour le développement que pour la maintenance.

- A éviter tant que possible

COUPLAGE FAIBLE

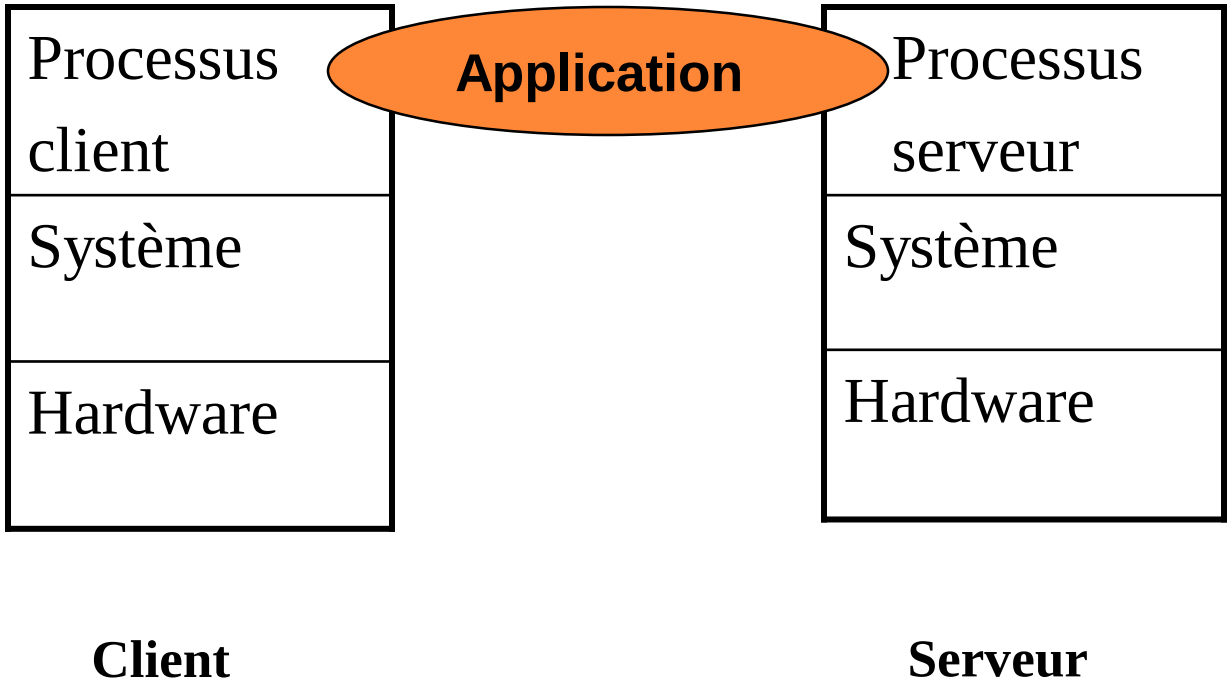
- Pas Interdépendance entre les composants de l'application



Pas de cycle dans le graphe de dépendances entre les composants de l'application

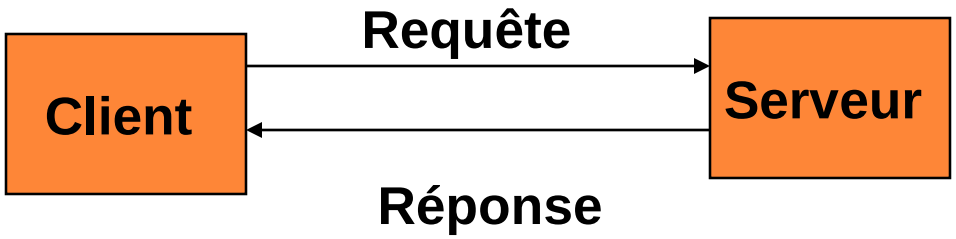
- Permet de maitriser la complexité de l'application:
  - pour le développement;
  - pour le test;
  - pour la maintenance.

Modèle client/serveur principe



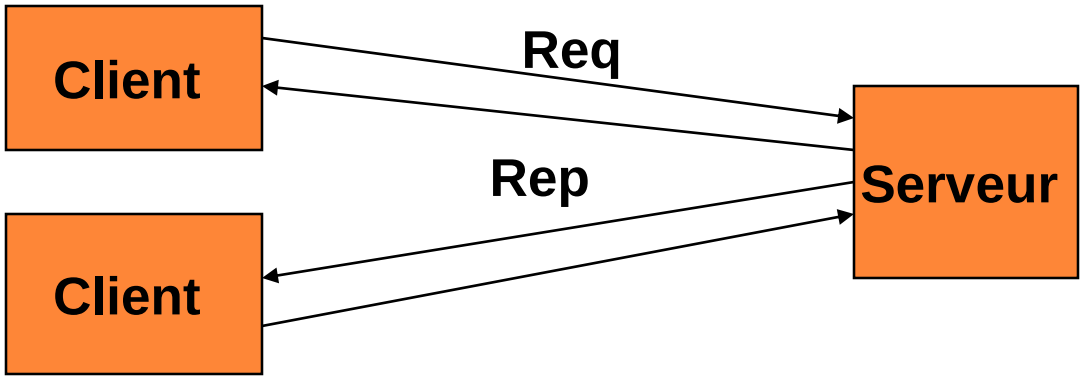
Client

- **Définition** :
  - Processus demandant l'exécution d'une opération à un autre processus
  - par envoi d'un message contenant le descriptif de l'opération à exécuter, et
  - attendant la réponse à cette opération par un message en retour.
  - Il est tjs l'initiateur du dialogue



SERVEUR

- **Définition :**
  - Processus accomplissant une opération
  - sur demande d'un client, et
  - transmettant la réponse à ce client.
  - Notion de session!!!



REQ & REP

- **Requête**:
  - Message transmis par un client à un serveur
  - décrivant l'opération à exécuter pour le compte du client

- **Réponse**:
  - Message transmis par un serveur à un client
  - suite à l' exécution d'une opération contenant les paramètres de retour de l'opération

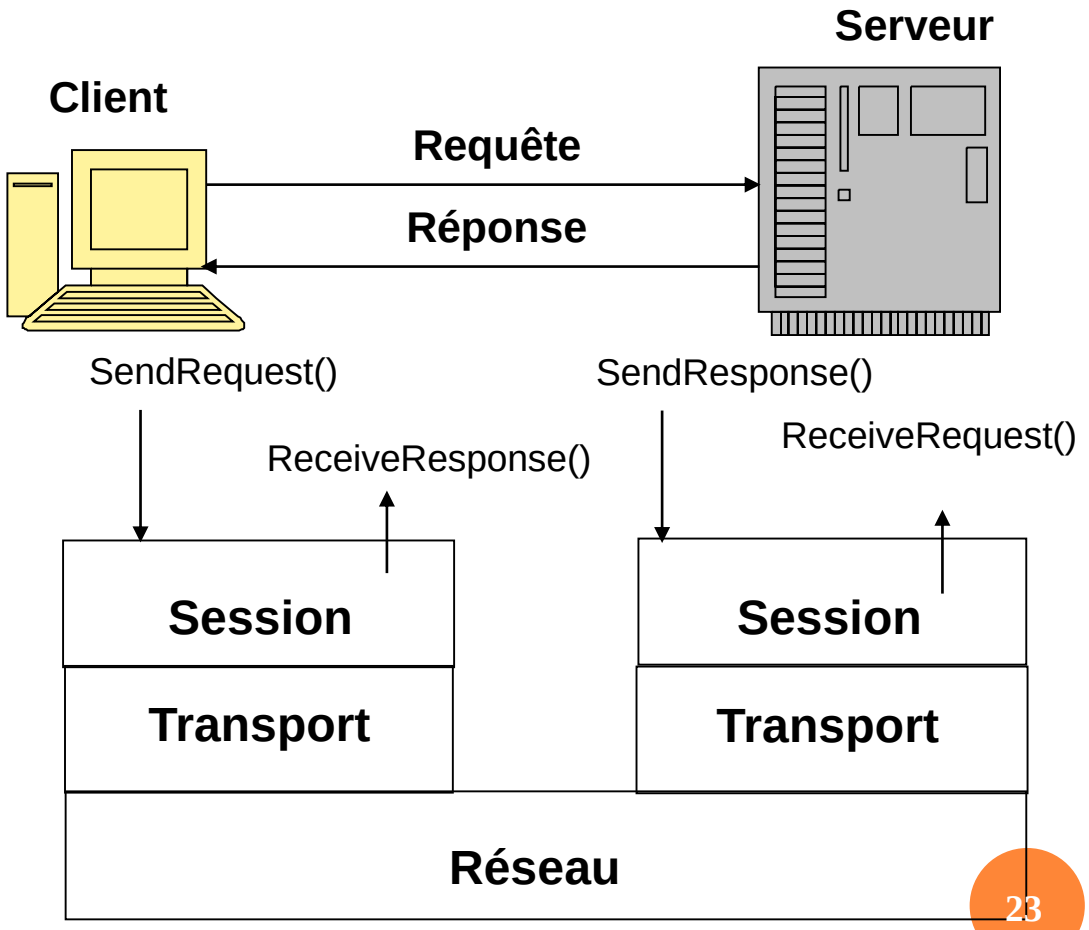
Architecture client/serveur

- **Points forts**
  - Couplage faible (le serveur n'a pas besoin des clients)
    - La maintenance et l'administration du serveur sont simplifiées
    - Souplesse : il est possible d'ajouter/supprimer dynamiquement des clients sans perturber le fonctionnement de l'application
  - Les ressources sont centralisées sur le serveur
    - Sécurisation simple des données : 1 seul point d'entrée
    - Pas de problème d'integrité/cohérence des données
- **Points faibles**
  - Un maillon faible : le serveur
  - Coût élevé : le serveur doit être très performant pour honorer tous les clients

Dialogue Client/Serveur

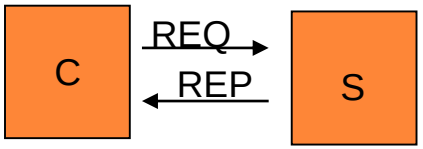
Primitives de service:

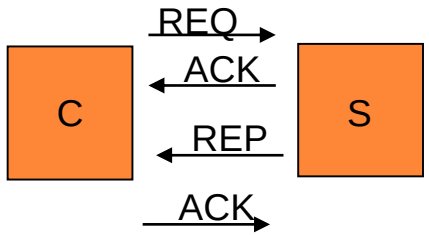
- SendRequest()
- ReceiveResponse()
- ReceiveRequest()
- SendResponse()

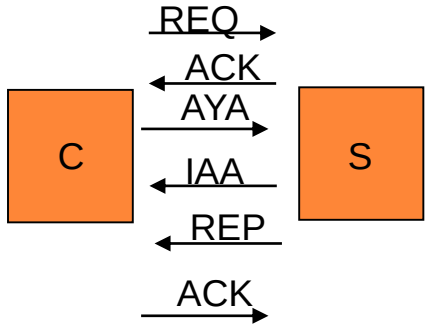


Messages Client/serveur

- En général on a trois types de messages: REQ, REP et ACK.
- Accessoirement, on peut avoir d'autres:
  - AYA (Are You Alive):
    - ✓ Utilisé par le client quand le serveur tarde à donner une réponse et que le client ne veut pas retransmettre sa requête.
  - IAA (I Am Alive):
    - ✓ Utilisé par le serveur pour demander au client d 'attendre un peu plus (ex. le serveur est occupé à faire un long travail).
    - ✓ Si le client perd patience (après plusieurs tentatives), il arrête tout.
  - TA (Try Again):
    - ✓ Utilisé si le message REQ ne peut être accepté (ex. boîte aux lettre trop pleine, adresse processus de destination inconnue)

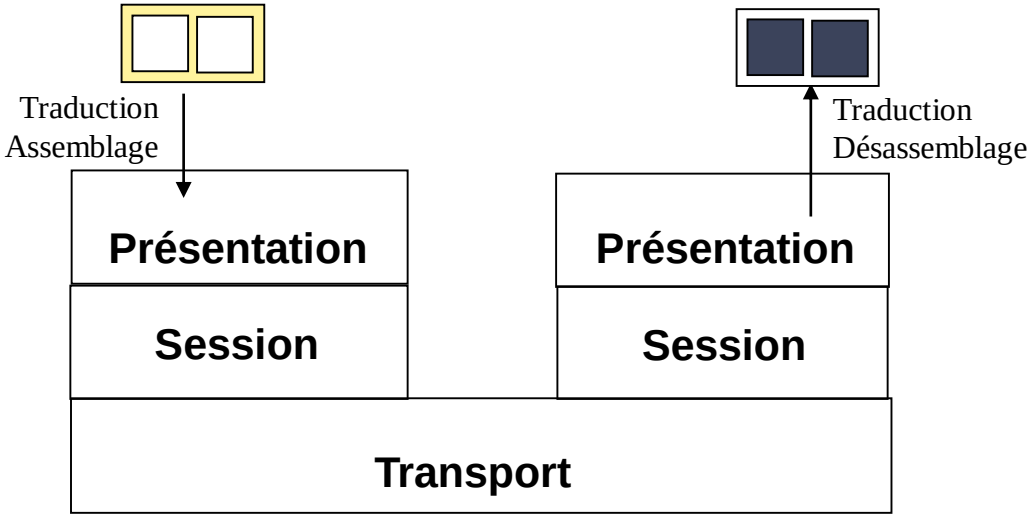






ÉCHANGES DE MESSAGES

- Dans un environnement hétérogène, on doit effectuer une présentation adéquate des données:
  - Traduction des données: XDR (SUN) , ASN.1(CCITT),…
  - Assemblage des paramètres émis (Marshalling) et des résultats
  - Désassemblage des paramètres reçus (Unmarshalling) et de résultats



ÉCHANGES DE MESSAGES (SUITE)

Assemblage (*Marshalling*):

- A l'émission de la requête, les paramètres sont arrangés et codés
- **Définition :**
  - Procédé consistant à prendre une collection de paramètres, à les arranger et à les coder en format externe pour constituer un message à émettre.

Désassemblage (*Unmarshalling*):

- A l'arrivée, les résultats doivent être remis en format interne à partir du message reçu.
- **Définition :**
  - Procédé consistant à prendre un message en format externe et à reconstituer la collection de paramètres qu'il représente en format interne.

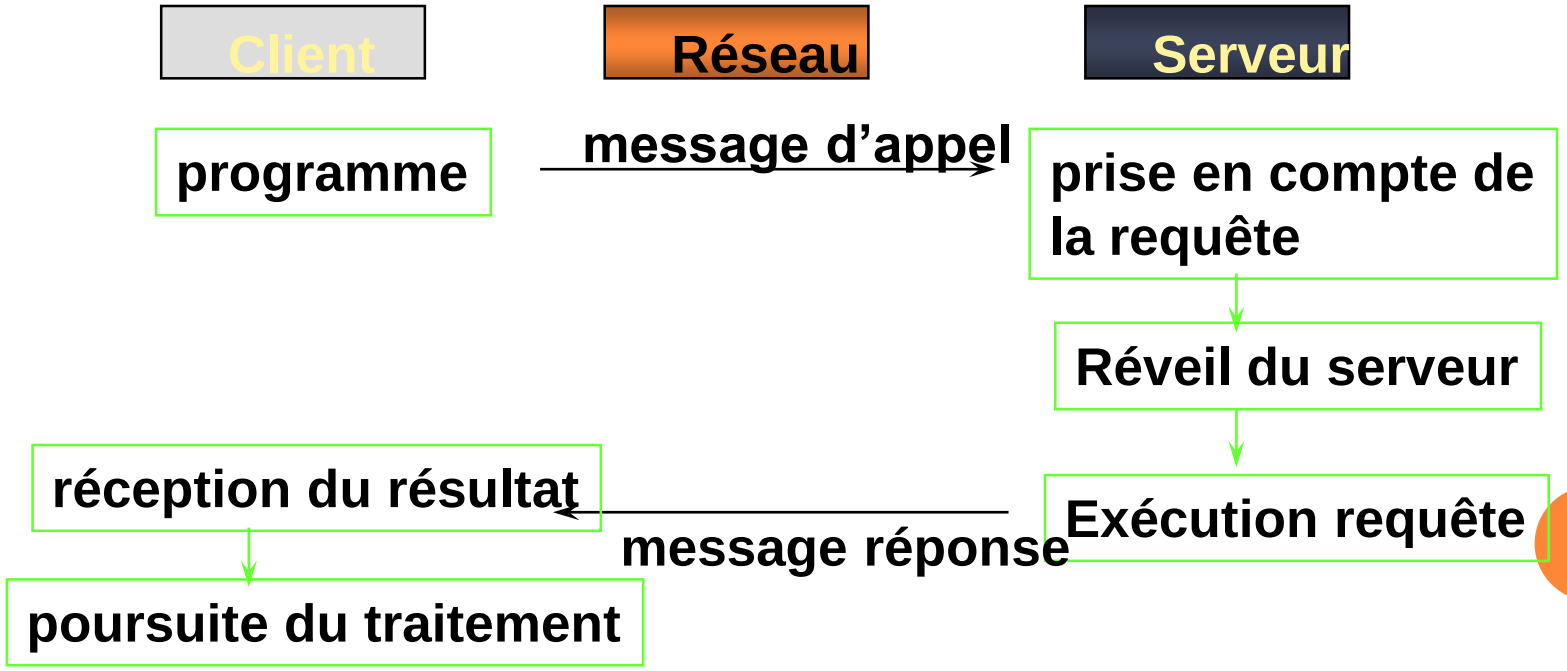
Modes de dialogue C/S

- Mode synchrone:
  - N'utilise pas de file d'Attente
  - Les messages sont émis sans attente.
  - Les commandes d'E/S sont bloquantes (ex. HTTP).

- Mode asynchrone:
  - Utilise une file d'attente
  - L'une au moins des commandes d'E/S est non bloquante
  - Favorise le multitâche (ex.: email, …)

Modes de communication

- **mode non-connecté:** l'arrivée des données + ordonnancement + non duplication ne sont pas garantis par le protocole ==> à gérer par l'application.
- l'approche non-connecté implique généralement une connexion synchrone

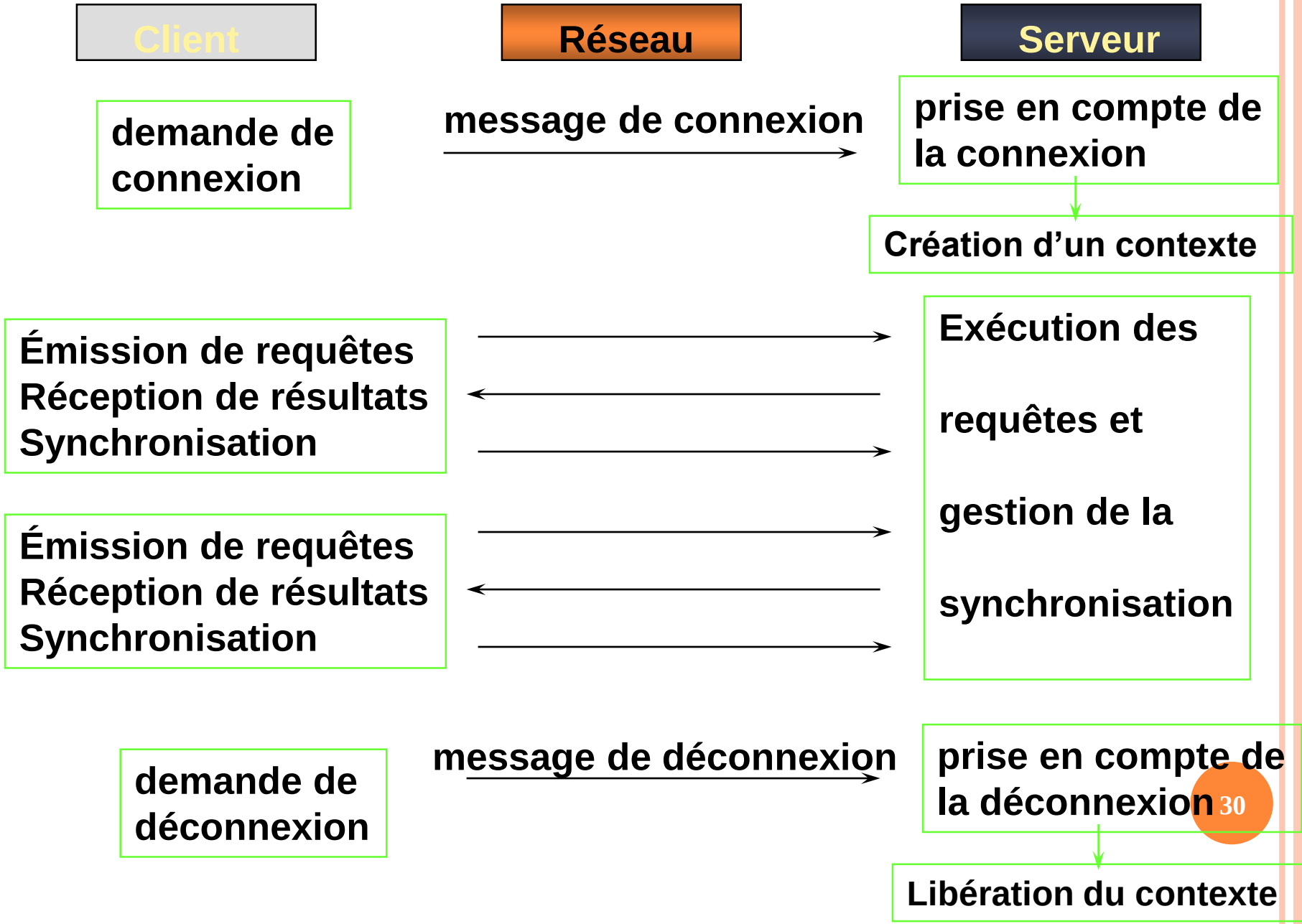


Modes de communication (suite)

- **le mode connecté:** implique une diminution des performances par rapport au mode non connecté: ceci est une contrainte pour certaines applications.

- le mode connecté permet une implémentation asynchrone des échanges, plus complexe mais plus fiable que le mode non-connecté.

Le mode connecté



Différentes combinaisons pour les dialogues C/S

mode comm. / dialogue	***Avec connexion***	***Sans connexion***
synchrone	**APPC** **RDA**	**RPC**
asynchrone	**pipes**	**Message queuing**

- RPC: (Remote Procedure Call) est préférable dans le cas ou les échanges sont sporadiques

- APPC: (Application Program to Program Communication) utile pour la gestion des transactions dans les bases de données.

- Message queuing: mécanisme simple mais manque de contrôle sur le délai d'obtention d'une réponse (ex: courrier électronique)

DIFFERENTS TYPES DE CLIENT-SERVEUR

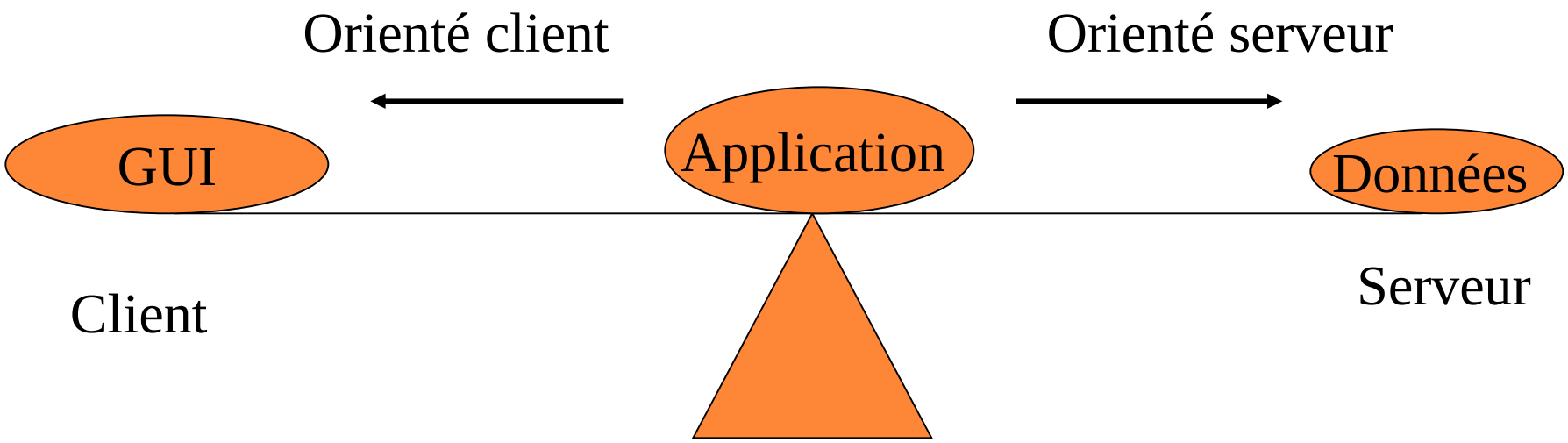
Couches d'une application

- Une application informatique est représentée selon un modèle en trois couches:
  - la couche présentation (interface Homme/Machine):
    - gestion de l'affichage (exemple Windows, X-window, etc.),
    - logique de l'affichage, partie intrinsèque de l'application qui transmet à la gestion de l'affichage, les éléments de présentation.
  - la couche traitement (qui constitue la fonctionnalité intrinsèque de l'application):
    - la logique des traitements : l'ossature algorithmique de l'application,
    - la gestion des traitements déclenchés par la logique de traitements qui réalise la manipulation des données de l'application (ex: procédures SQL)

Couches d'une application (suite)

- **la couche données** qui assure la gestion des données:
  - la logique des données constituant les règles régissant les objets de la base de données,
  - la gestion des données (consultation et mise à jour des enregistrements). Un système de type SGBDR, habituellement, est responsable de cette tâche.
- Le découpage permet de structurer une application en mode client/serveur;
- le module de gestion des données peut être hébergé par un serveur distant,
- le module de logique de l'affichage peut également être géré par un serveur distant (un Terminal X par exemple).

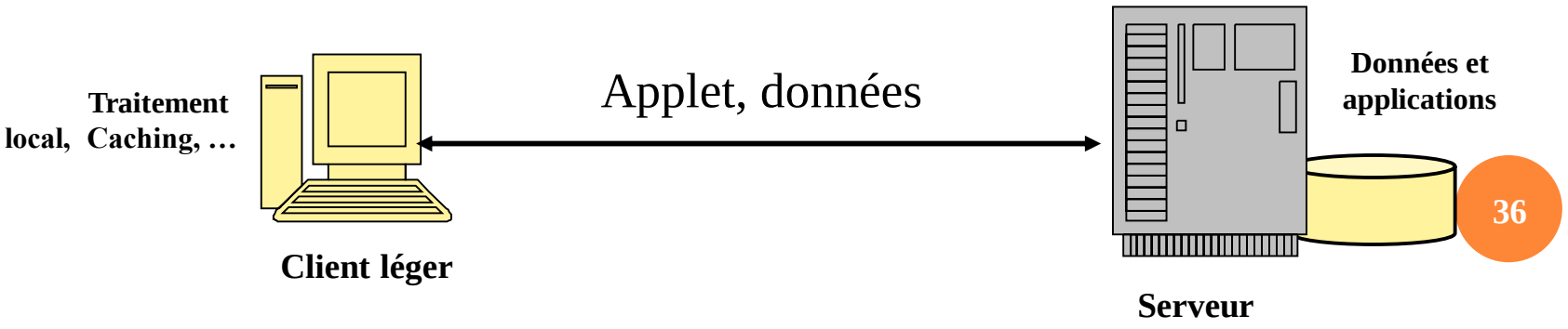
C/S ORIENTÉ CLIENT OU SERVEUR



- Client lourd (Fat Client):
  - Stocke les données et les applications localement. Le serveur stocke les fichiers mis à jour, …
  - Le client obtient une bonne partie du traitement: serveurs de bases de données, de fichiers, ...
  - Le serveur est plus allégé

C/S orienté client ou serveur

- Serveur lourd (Fat Server):
  - On met plus de charge sur le serveur: groupeware, transactions, objets,…
  - Plus facile à gérer car on peut enrichir le serveur sans trop affecter les clients.
- Client léger (Thin Client):
  - Client à fonctionnalité minimale: Terminaux X, stations de travail sans disque dur, Ordinateur Réseau (Network Computer), …
  - Beaucoup de charge sur le serveur

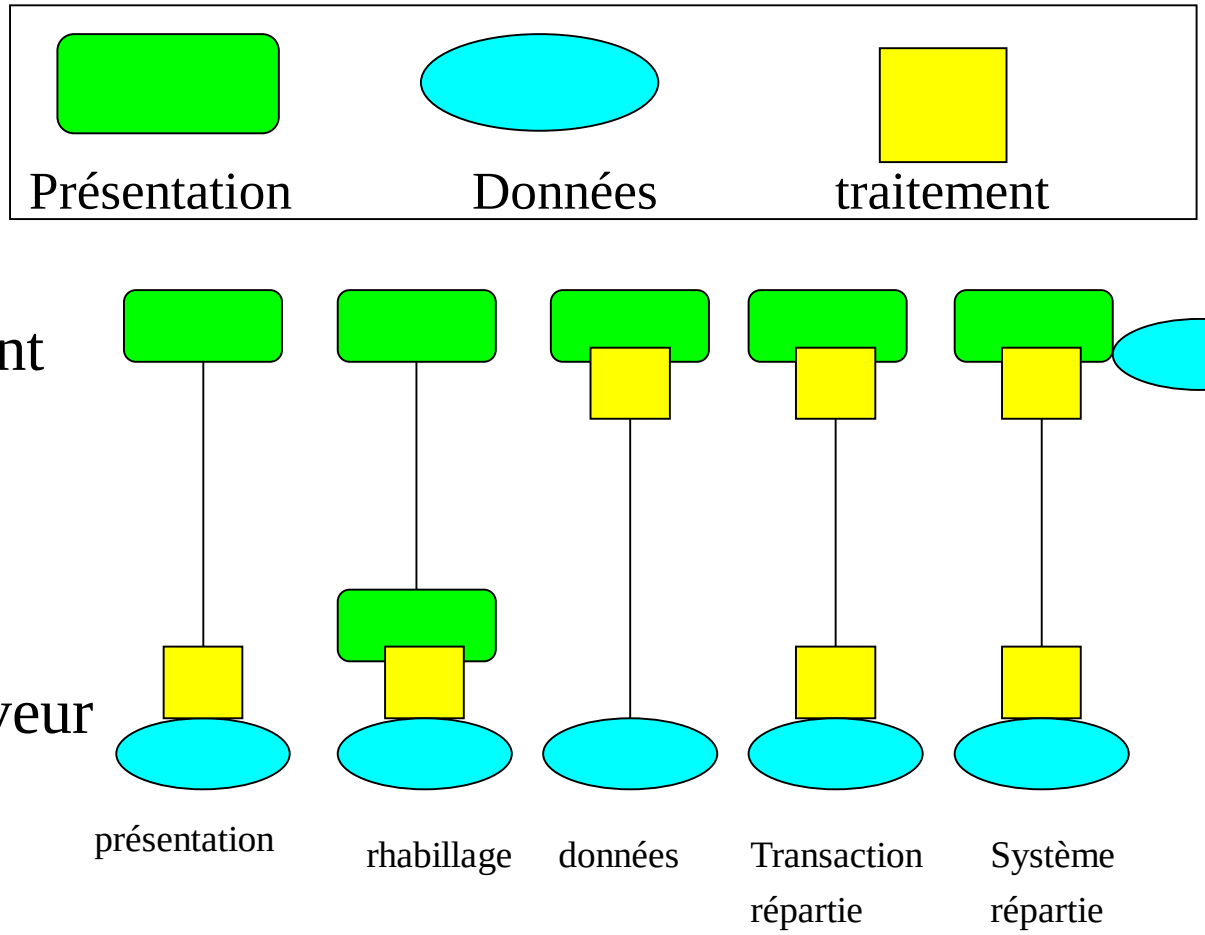


C/S orienté client ou serveur

- Client riche (Rich Client Platform):
  - Développer des applications traditionnelles, ou des applications type client-serveur.
  - Propose un environnement d'exécution comprenant des composants de base sur lequel seront déployées les applications.
  - Propose un Framework de développement et des composants de base pour faciliter le travail des développeurs.

Différents types de C/S

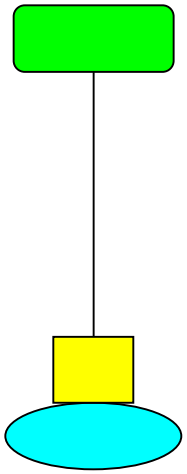
- Modèle de **Gartner** pour les systèmes à deux niveaux (2-tiers) :



Différents types de C/S (suite)

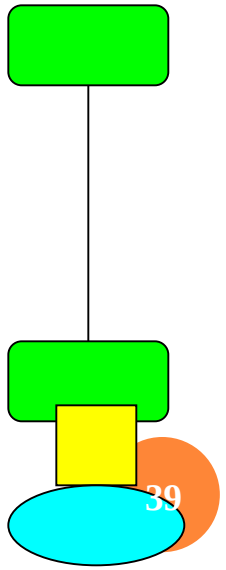
- **C/S de présentation**
  - Le client ne gère que les dialogues avec l'utilisateur
  - interface utilisateur

Emulation des terminaux



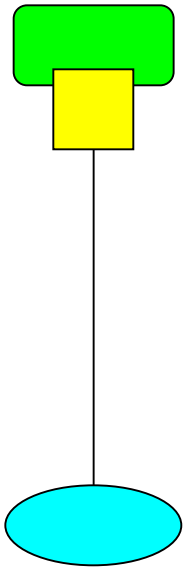
telnet

- **Rhabillage (revamping)**
  - Présentation repartie
  - On « rajoute » un client à une application existante
  - séparation clients
    - utilisateurs
    - Clients
  - Ex: X-Window



Différents types de C/S (suite)

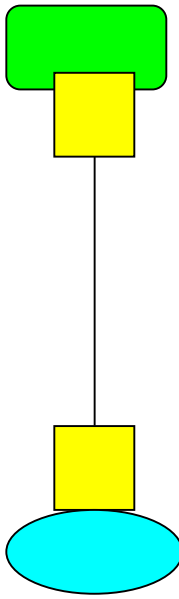
- **C/S de données**
  - Service de fichiers
  - Accès aux données
  - requêtes du type SQL



- **Transactions réparties**
  - Application des 2 côtés
  - sous-traitance
  - Ex: le Web

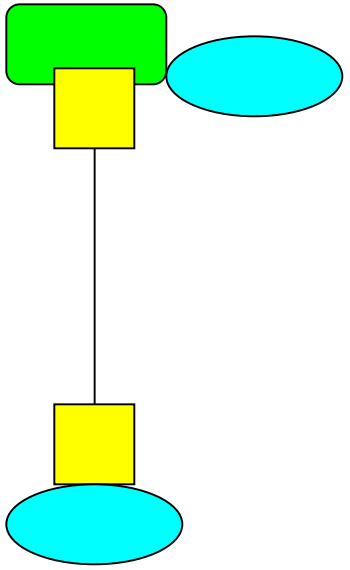
HTML, Java, JS

CGI, httpd server



Systèmes répartis

- tous des 2 côtés
- coopération
- pas les limites des modèles classiques :
  - réutilisation
  - transparence de la localisation
  - sécurité
  - persistance



Situation actuelle!!

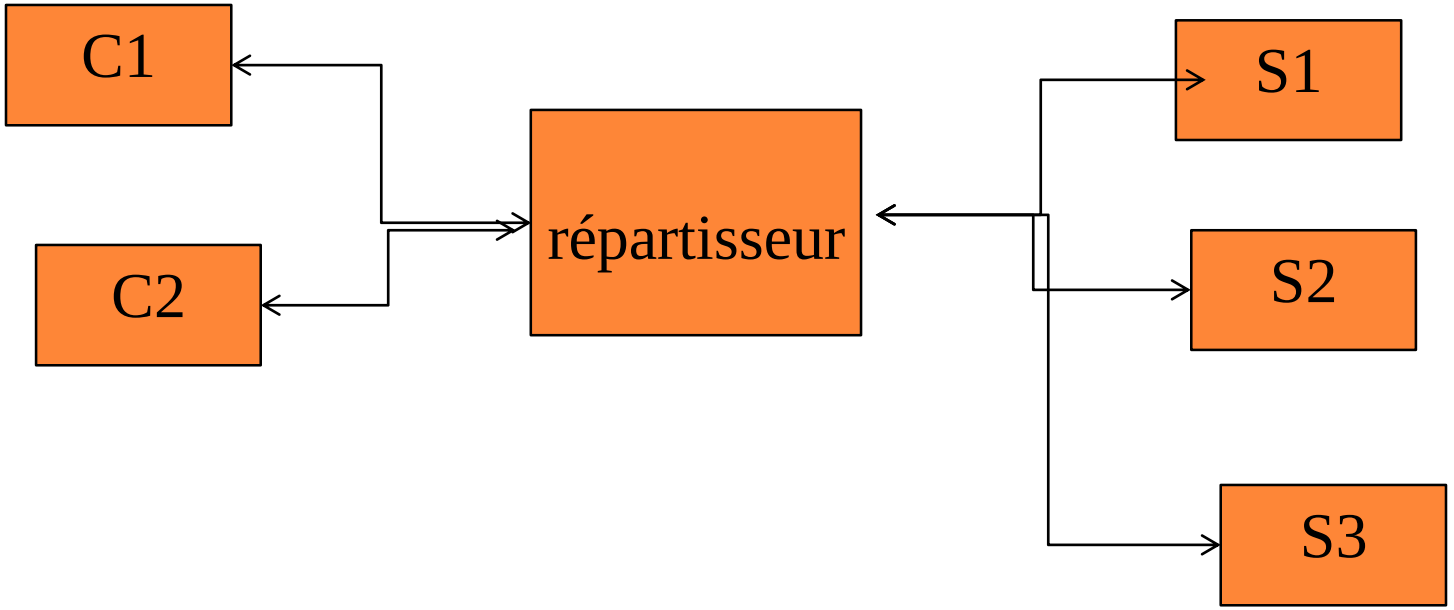
- Les Applications construites sont de plus en plus complexes :
  - Application de commerce en ligne
  - Radio/télévision numérique interactive
  - Moteur de recherche…
- Dans le modèle client/serveur toute la complexité est concentrée dans le serveur
  - Problème de performance/disponibilité
    - Répartition de charge (load balancing)
  - Problème pour maîtriser la complexité
    - Architectures multicouches (N- Tiers)

Répartition de charge Load balancing

- Un serveur traite simultanément les requêtes de plusieurs clients
- Les requêtes de deux clients sont indépendantes
- L'idée du « load balancing » est de paralléliser sur plusieurs serveurs identiques s'exécutant sur des machines différentes le traitement des requêtes concurrentes

Répartition de charge

- Les requêtes des clients passent par un répartisseur de charge qui les répartit sur N serveurs identiques.



Répartition de charge

- Rôle principal
  - Diriger les requêtes des clients en fonction de la charge de chacun des serveurs
- Rôle annexe
  - Gérer les sessions des clients (2 solutions)
    - Toutes les requêtes d'un client sont dirigées vers un seul serveur
    - Les données de session sont transmises avec la requête
    - Ex: gestion des données concernant un client
  - Gérer les défaillances :
    - Ne plus diriger de requêtes sur un serveur « crashé »
  - Assurer le passage à l'échelle (scalabilité)
    - Permettre l'ajout et le retrait de serveurs sans interruption de service

Répartition de charge

- Points forts
  - Transparent pour les clients
  - Scalable : le nombre de serveurs peut être adapté à la demande
  - Tolérant aux défaillances : la défaillance d'un serveur n'interrompt pas le service
  - Plus besoin de machines très chères : il suffit d'en mettre plus
- Point faible
  - Les données ne sont plus centralisées mais dupliquées
  - Le répartiteur de charge devient le « maillon faible »

Maîtriser la complexité

- L'application devient complexe :
  - Difficile à développer
  - Difficile à tester
  - Difficile à maintenir
  - Difficile à faire évoluer
- Solution : les architectures multicouches
  - Inspiré par le développement en couches des protocoles réseaux et des architectures à base de composants

Les niveaux d'une architecture C/S

- 2 niveaux:

  Plusieurs clients et un serveur.

  Découpage en deux de la charge de l'application

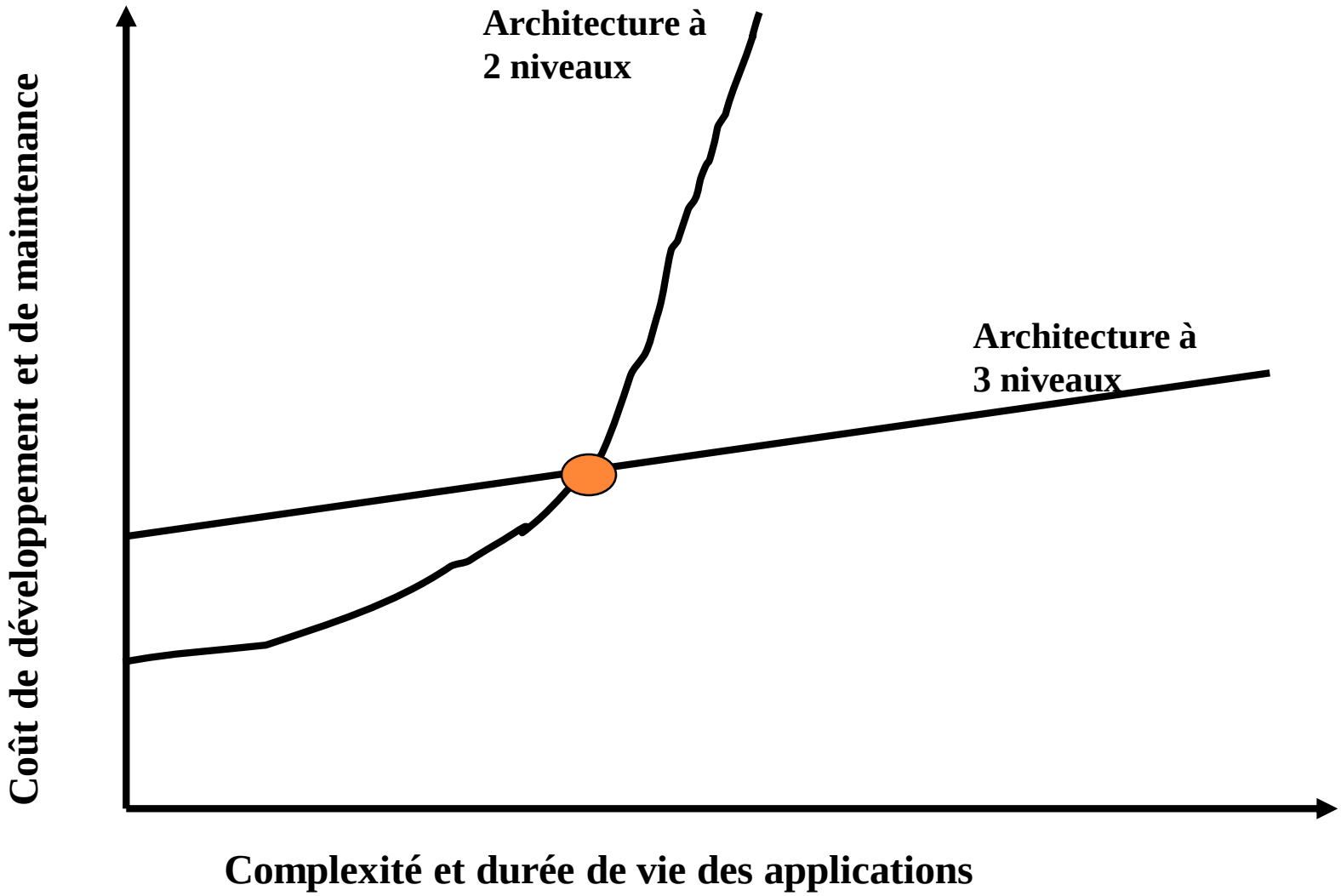
- **3 ou N niveaux:**

  Un client, plusieurs serveurs

  Répartition de la charge entre le client et plusieurs serveurs

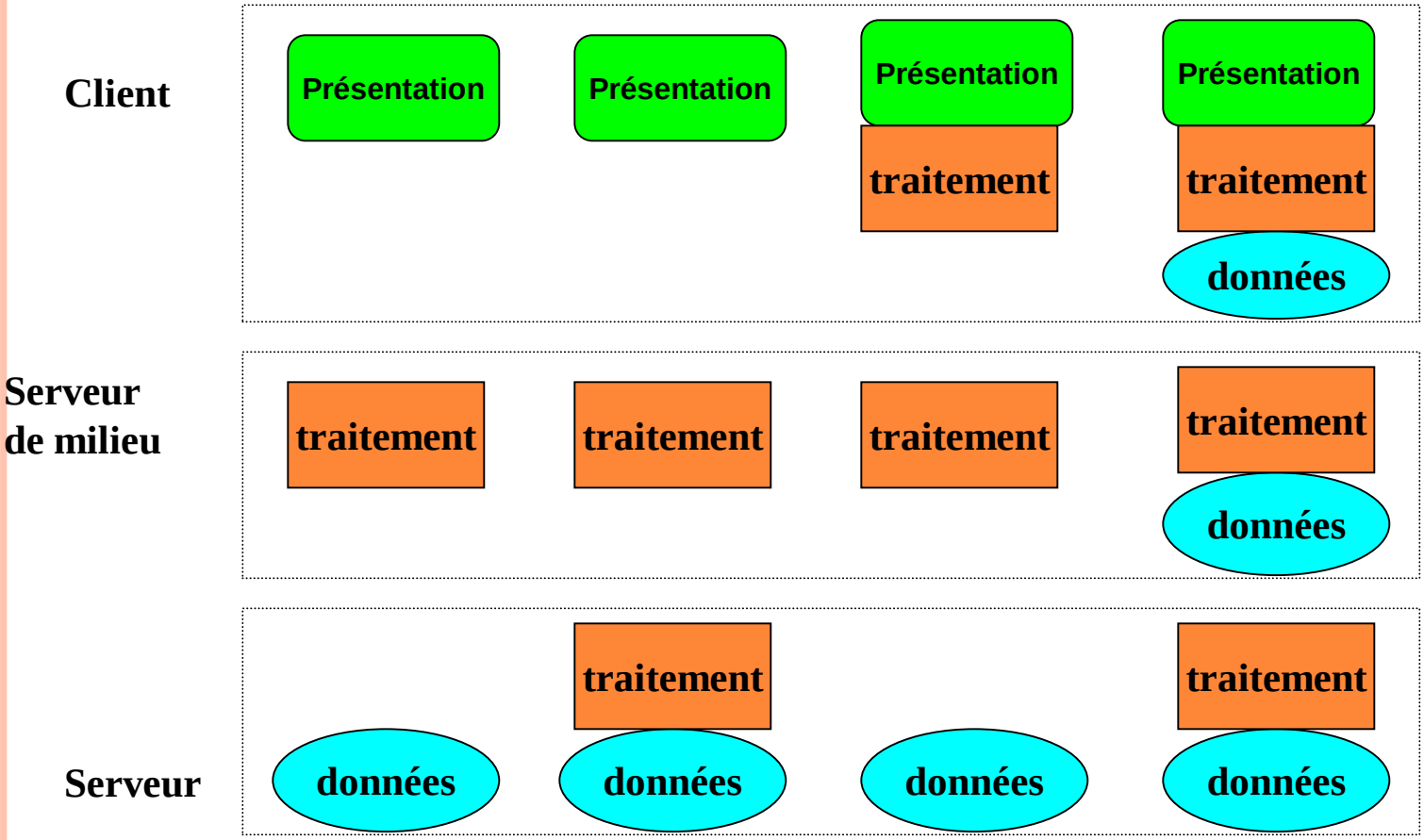
Quand utiliser un modèle à 3 niveaux?

- La réponse de Gartner Group
  - L'application comprend plus de 50 classes ou services
  - Les applications sont programmées dans des langages différents ou écrites par des organisations différentes
  - Il existe au moins deux sources de données hétérogènes, telles que deux SGBD différents
  - La durée de vie des applications est supérieure à 3 ans
  - La charge est élevée, plus de 50000 transactions par jour ou plus de 300 utilisateurs accédant à la même base
  - Les communications entre applications sont nombreuses.


Architecture à
2 niveaux
Architecture à
3 niveaux
Coût de développement et de maintenance
Complexité et durée de vie des applications

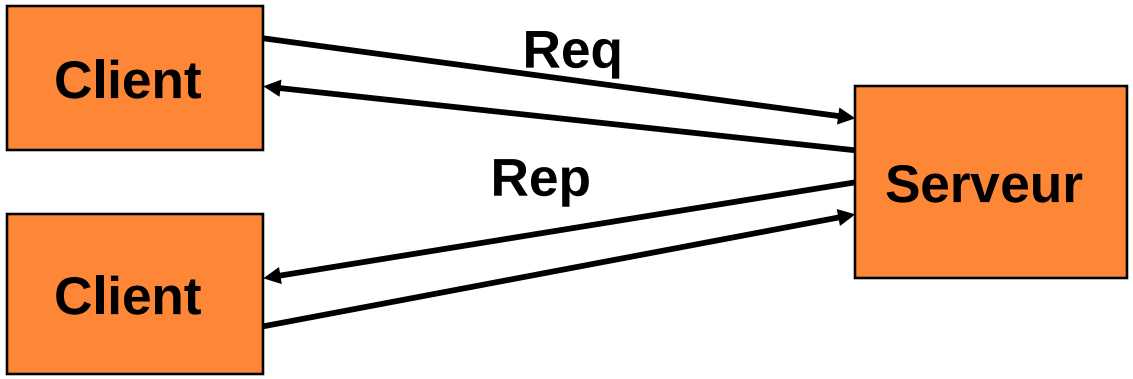
CLIENT/SERVEUR À 3 NIVEAUX

- **Modèle de Gartner pour les systèmes à 3 niveaux (3-tiers):**



4- FONCTIONNEMENT D'UN SYSTEME CLIENT-SERVEUR

FONCTIONNEMENT D'UN SYSTÈME C/S



- Le client émet une requête vers le serveur grâce à son adresse et le port, qui désigne un service particulier du serveur
- Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine client et son port

Architecture d'un client

- Une application cliente est moins complexe que son homologue serveur car:
  - la plupart des applications clientes ne gèrent pas d'interactions avec plusieurs serveurs,
  - la plupart des applications clientes sont traitées comme un processus conventionnel.
  - la plupart des applications clientes ne nécessitent pas de protection supplémentaires, le système d'exploitation assurant les protections élémentaires suffisantes.

Architecture d'un serveur

- **Processus serveur:**
  - Offre une connexion sur le réseau;
  - Entre indéfiniment dans un processus d'attente de requêtes clientes;
  - Lorsqu'une requête arrive, le serveur déclenche les processus associés à cette requête, puis émet la ou les réponses vers le client;
  - Problème : gérer plusieurs client simultanément.
  - Les types de serveurs:
    - **serveurs itératifs**: ne gèrent qu'un seul client à la fois;
    - **serveurs parallèles** : fonctionnent « en mode concurrent ».

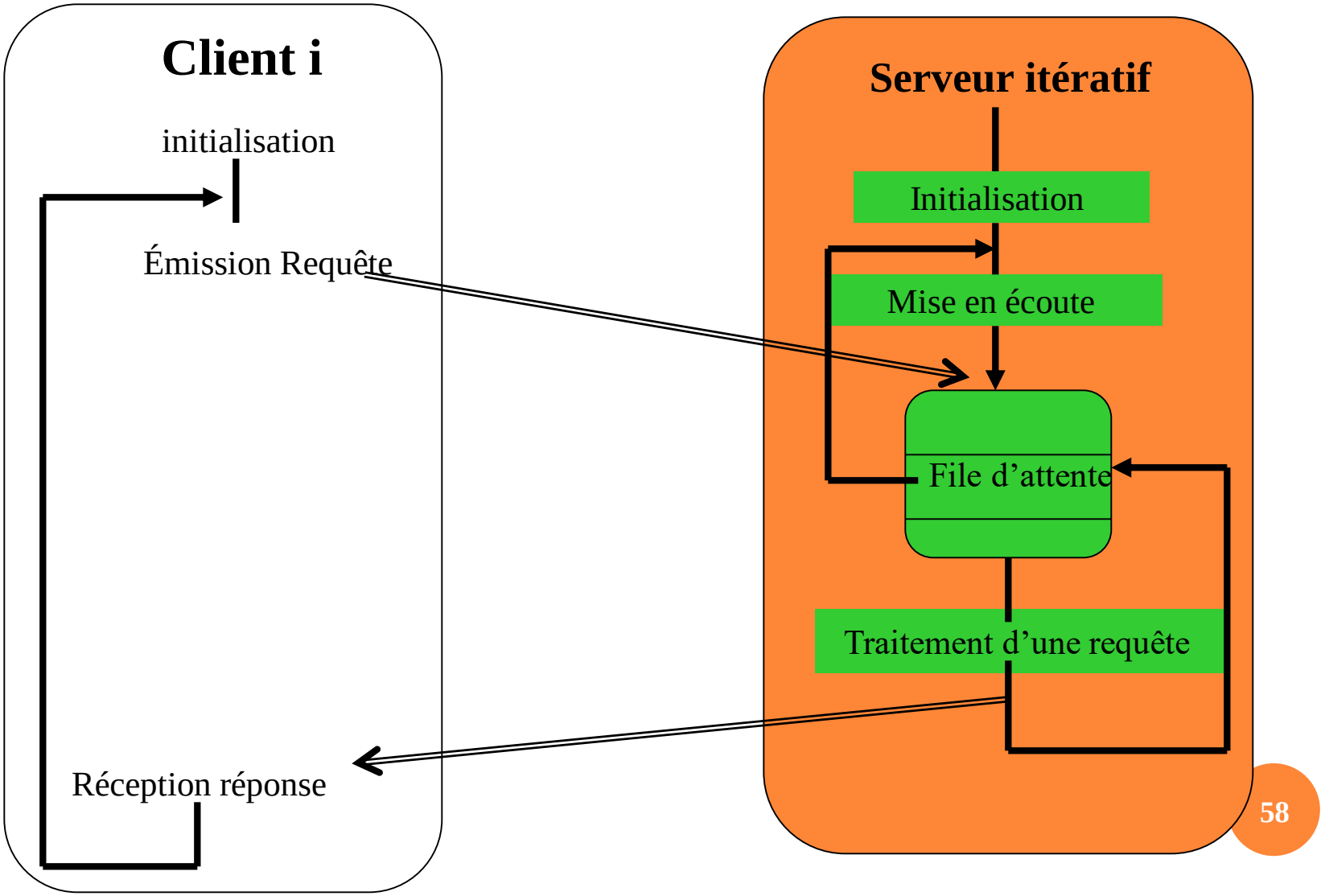
Serveur itératif

- Les requêtes sont traitées séquentiellement par le même processus. Mais pour éviter de perdre des requêtes qui arrivent pendant un traitement, l'attente et la mémorisation des requêtes dans la file d'attente se font en parallèle avec les traitements.

Serveur itératif (suite)

- 2 types : connecté ou non-connecté selon le protocole de transport
- le serveur itératif en mode non-connecté:
  - offre une interface de communication sur le réseau en mode non-connecté,
  - indéfiniment : réceptionne une requête client, formule une réponse, et renvoie le message réponse vers le client selon le protocole applicatif défini.
- le serveur itératif en mode connecté:
  - offre une connexion sur le réseau en mode connecté,
  - (*) réceptionne une connexion client,
  - offre une nouvelle connexion sur le réseau,
  - répétitivement : réceptionne une requête pour cette connexion, formule une réponse, et renvoie le message réponse vers le client,
  - lorsque le traitement pour ce client est terminé -->(*).

Serveur itératif (suite)



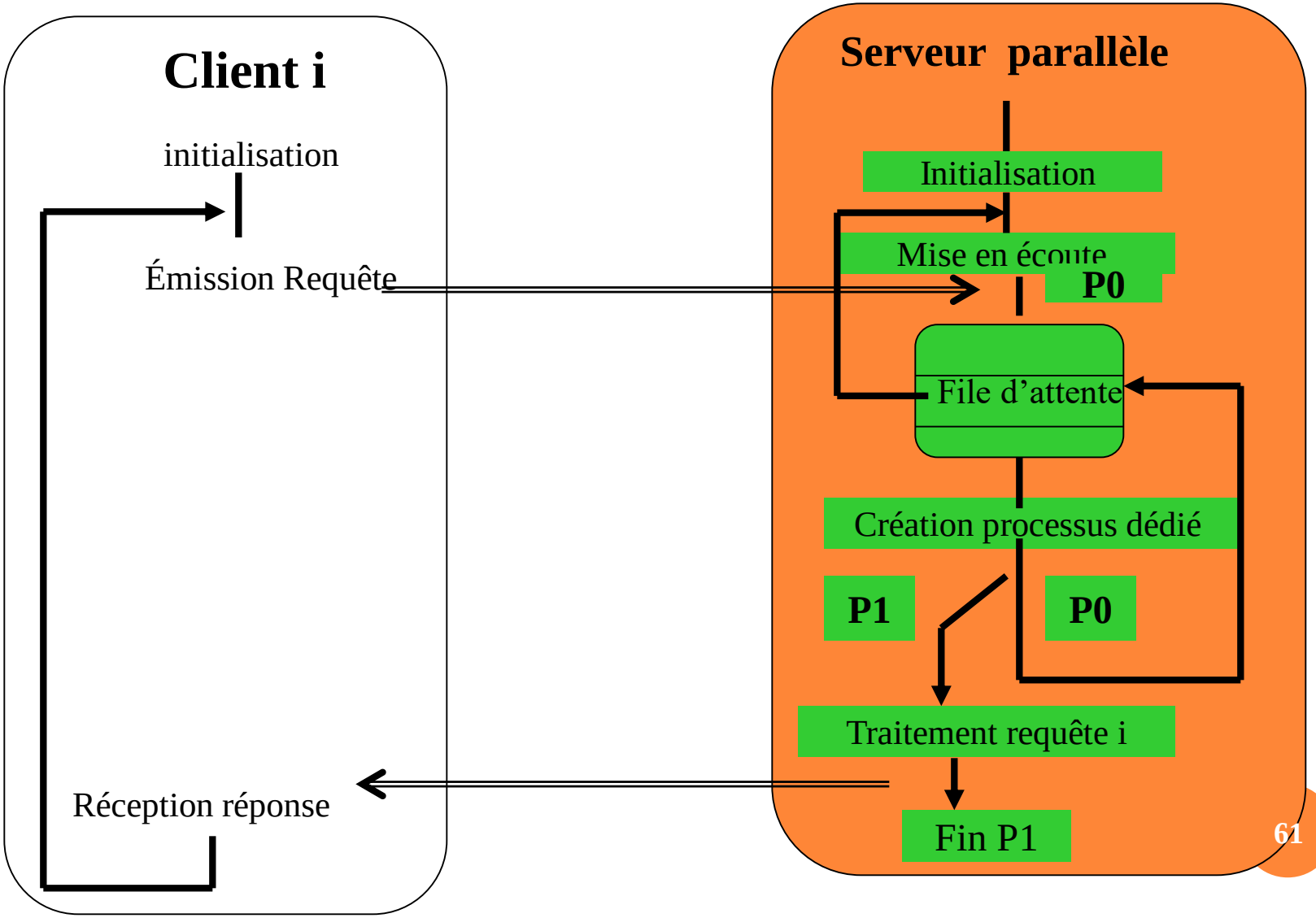
Serveur Parallèle

- Chaque requête est traitée par un processus fils dédié crée à cet effet. Une requête trop langue ne pourra plus retardé d'une façon trop importante d'autres traitements
- 2 modes de communication: connecté et non-connecté
- le serveur parallèle en mode non-connecté:
  - offre une interface de communication en mode non-connecté,
  - répétitivement : réceptionne la requête client; offre une nouvelle interface de communication sur le réseau, et crée un processus secondaire (PR. S.) chargé de traiter la requête courante.
  - (PR. S.) : formule une réponse à la requête client, et renvoie le message,
  - (PR. S.) : lorsque le traitement est terminé, libère la communication, Exit.

Serveur Parallèle (suite)

- le serveur parallèle en mode connecté:
  - offre une connexion sur le réseau en mode connecté,
  - répétitivement : réceptionne une connexion client, offre une nouvelle connexion sur le réseau, créent un PR. S. chargé de traiter la connexion courante.
  - (PR. S.) : répétitivement : réceptionne une requête pour cette connexion, formule une réponse, et renvoie le message réponse vers le client selon le protocole applicatif défini,
  - (PR. S.) : lorsque le traitement est terminé (propre au protocole applicatif), libère la connexion, Exit.

Serveur Parallèle (suite)



Comment choisir un type

- serveur itératif en mode non-connecté :
  - Nécessite très peu de traitement par requête (pas de concurrence)
  - Exemple: serveur TIME
- serveur itératif en mode connecté :
  - Nécessite très peu de traitement par requête mais requièrent un transport fiable de type TCP. Peu utilisé.
- serveur concurrent en mode non-connecté si :
  - temps de création d'un processus extrêmement faible (dépend du système d'exploitation hôte) par rapport au temps de traitement d'une requête,
  - les requêtes nécessitent des accès périphériques importants (dans ce cas, la solution itérative est inacceptable).

Comment choisir un type (suite)

- serveur concurrent en mode connecté : offre un transport fiable et est capable de gérer plusieurs requêtes de différents clients simultanément
- Implémentation:
  - Multi-instanciation de processus avec un processus primaire et des processus secondaires traitant les connexions clientes,
  - avec un seul processus gérant les multiples connexions par l'intermédiaire de requêtes asynchrones et primitive d'attente d'évènements multiples.

Service avec ou sans état

- Service avec état:
  - le serveur conserve localement un état pour chacun des clients connectés : informations sur le client, les requêtes précédentes, …
- Service sans état:
  - le serveur ne conserve aucune information sur l'enchaînement des requêtes…
- Incidence sur les performances et la tolérance aux pannes dans le cas où un client fait plusieurs requêtes successives
  - performance --> service sans état
  - tolérance aux pannes --> service avec états
- Exemple : accès à un fichier distant
- RFS avec états, NFS sans état (pointeur de fichier…)

Hardware et Software

- Client
  - GUI: Graphical User Interface
  - OS: Windows98, XP, Vista,…
  - Video, audio,..
- Serveur
  - Grande capacité de stockage
  - Tolérance aux pannes et la disponibilité d'informations
  - Performant, rapide
  - Systèmes d'exploitation serveur :
    - Windows Server 2008
    - OS/2 SMP (IBM)
    - Solaris (Sun)
    - UnixWare2 (Novell)

MIDDLEWARE

1- Concepts de base

2- Types de Middleware

a- BD

b- RPC, Web

c- MOM

Définition: (middleware)

- Définition: ensemble des services logiciels construits au-dessus d'un protocole de transport afin de permettre l'échange de requêtes et des réponses associées entre client et serveur de manière transparente:
  - Transparence aux réseaux
  - Transparence aux serveurs
  - Transparence aux langages

Objectifs d'un Middleware

- Transports de requêtes et réponses;
- Simplification de la vision utilisateur:
  - Développement d'API transparente;
  - Intégration uniforme aux langages.
- Harmonisation des types de données;
- Performance:
  - Gestion de caches clients et serveurs;
  - Parcours des résultats complexes (ensembles, objets longs).
- Fiabilité:
  - Gestion de transaction intégrée;
  - Communication (réémission, détection de double émission).

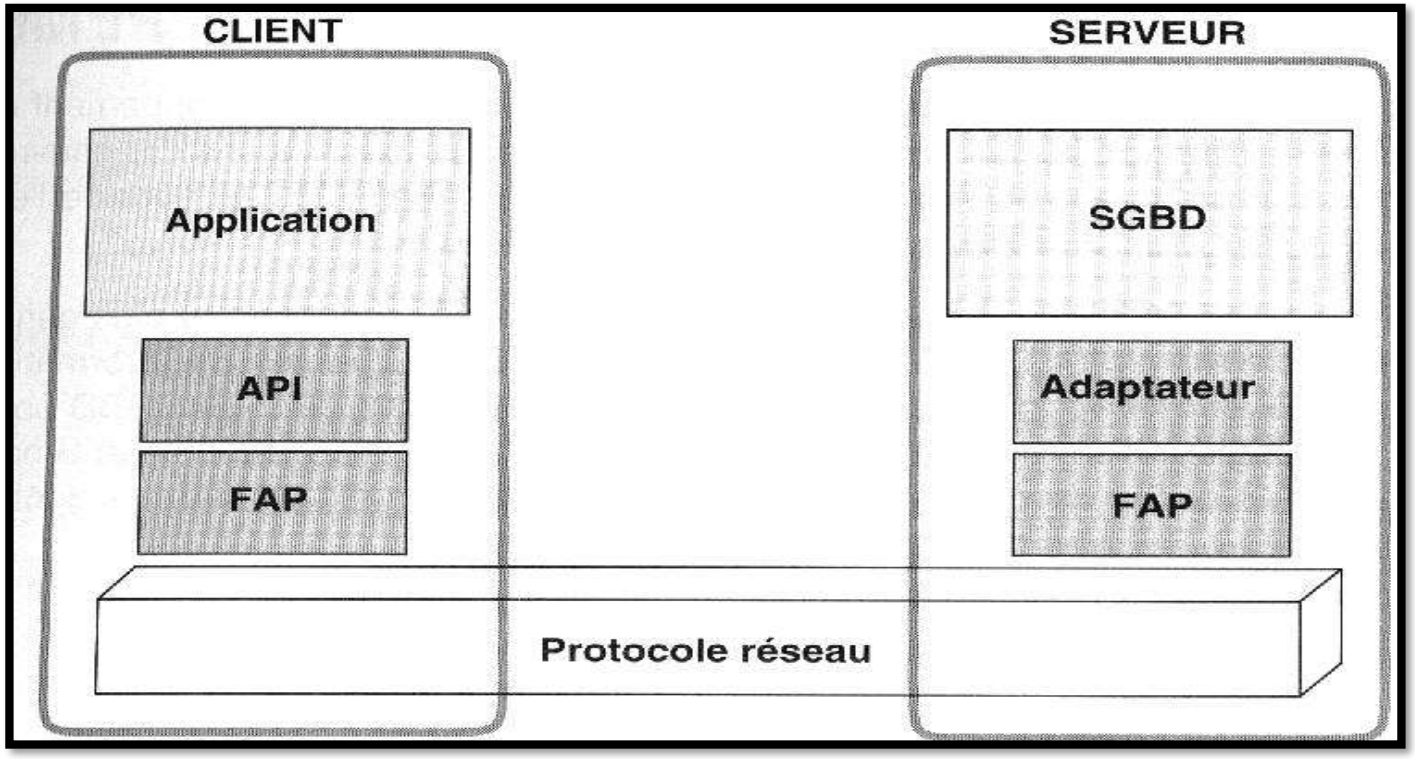
Fonctions d'un Middleware

- **Procédure de connexion**:
  - Opération permettant d'ouvrir un chemin depuis un client vers un serveur désigné par un nom, avec authentification de l'utilisateur associé par nom et mot de passe et identifie la BD.
- **Préparation de requêtes**:
  - Opération consistant à envoyer une requête avec des paramètres non instanciés à un serveur afin qu'il prépare son exécution.
- **Exécution de requêtes**
  - Opération consistant à envoyer une demande d'exécution de requête précédemment préparée à un serveur, en fournissant les valeurs des paramètres.

Fonctions d'un Middleware (suite)

- **Récupération des résultats:**
  - Opération permettant de ramener tout ou partie du résultat d'une requête sur le client.
- **Cache de résultats:**
  - Technique permettant de transférer les résultats par blocs et de les conserver sur le client ou/et sur le serveur afin de les réutiliser pour répondre à des requêtes.
- **Cache de requêtes:**
  - Technique permettant de conserver des requêtes compilées sur le serveur afin de les réutiliser pour répondre à des requêtes similaires.
- **Procédure de déconnexion:**
  - Opération inverse de la connexion, permettant de fermer le chemin ouvert depuis le client vers le serveur associé.

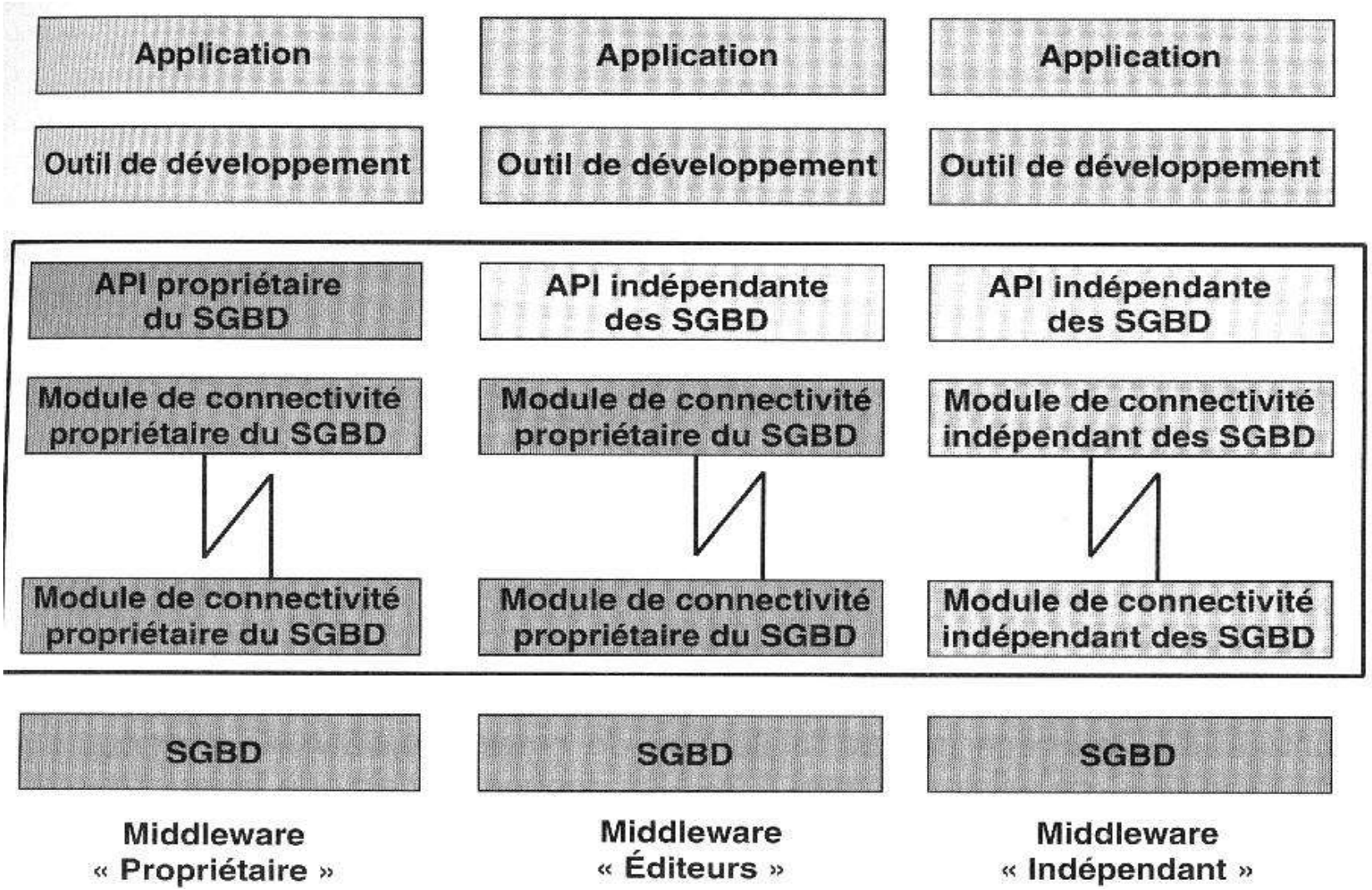
Modèle fonctionnel



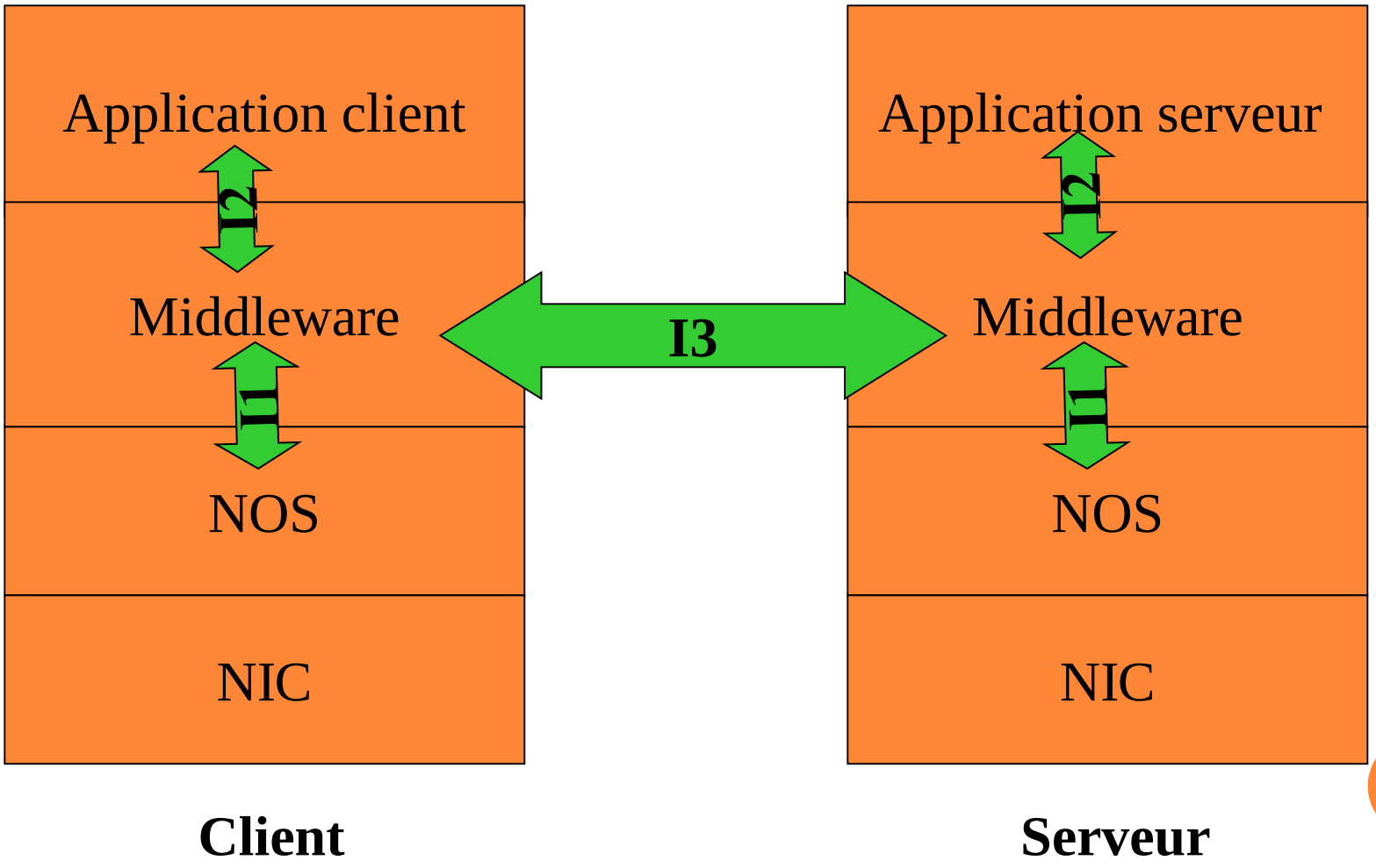
API: interface utilisateur

FAP: module de formatage et d'accès

Différents types de Middleware



Architecture du middleware



Architecture du middleware (suite)

- **I1** est l'interface la plus simple dans l'architecture: Un middleware est compatible a un OS ou non;
- **I2** représente l'intégration avec l'application: API;
- **I3** est une interface logique (la plus importante). Fournit une intégration entre les composantes du middleware de la machine client est la machine serveur;
- Les fonctionnalités représentées dans I2 sont limitées selon les possibilités de I3;
- Le choix d'un Middleware dépend de la façon dont ces interfaces (I1, I2, I3) adhèrent.

Liste compatibles d'interfaces

Interface	usage	Exemple
1- système d'exploitation	Client OS Serveur OS	Windows 7 Windows NT Server
2- application	Envt de développement	.NET, JEE,…
3- Communication	Technique de communication du Middleware	ODBC, RPC, CORBA, HTTP,…

Catégories de middleware

- Plusieurs implémentations du middleware
- Aider au choix d'un middleware: mettre en **catégorie**
- En se basant sur des critères d'ordre techniques et non marketing
- 2 catégories importantes:
  - Basée sur l'application (plates-formes)
  - Basée sur la communication (canaux)

Les catégories basées sur les applications

Catégorie	Type d'application	Technologie
Bases de données	Intégrer des BDs clients dans des serveurs de BD relationnelles	SQL ODBC
Legacy/application (héritées)	Intégrer un client dans des applications existants	TPM
Web	Intégrer des clients Web dans un serveur	CGI, ASP, PHP JSP,…

La méthodologie de communication

- Cette catégorie technique est indépendante de l'application supportée par le middleware

Catégorie	Exemple/standard
Remote Procedure Calls	DCE
Message-Oriented	Message Queuing Message Passing
TPM: moniteur transactionnel	CICS, IMS, ACMSxp
Orienté objet	CORBA DCOM

Middleware bases de données

- Middleware: comprendre l'application du programmeur pour accéder et manipuler les données stockées dans les serveurs BDs distants;
- Client qui formate les données pour la présentation;
- **Structured Query Language (SQL)** est un langage standard développé pour faciliter l'accès aux serveurs BD relationnel: ajout, mise a jour,...

SQL

- Est un standard ANSI (Americain National Standard Institute)
- 4 composantes principales:
  - Langage de définition de schéma (table, vues,…)
  - Langage de manipulation (sélection, mise a jour,…)
  - Spécification de modules appelables (procédures)
  - Intégration aux langages de programmation
- 4 verbes de base:
  - SELECT, INSERT, UPDATE, DELETE
- http://ceria.dauphine.fr/cours98/BD-wl-98.html

SQL (suite)

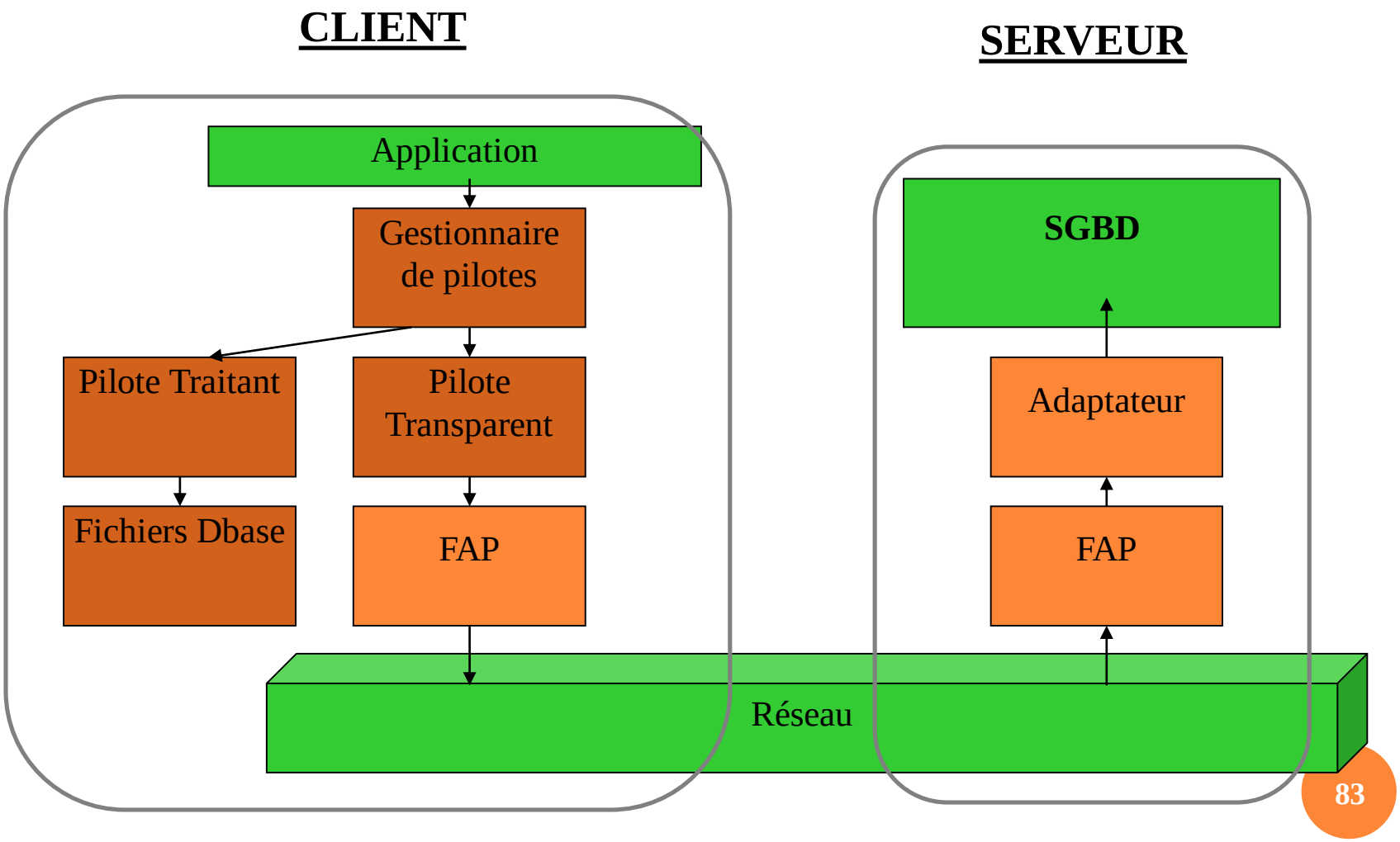
- La norme SQL était très restreinte et incomplète;
- Étendre la norme par les éditeurs de SGBDR dans les productions commerciales.

Vendeur BD	**Serveur BD**	**Middleware**
Oracle	Oracle	PL/SQL
CA-Ingres	Ingres	Open Ingres
IBM	DB2	SQL PL
Microsoft	SQL Server	Transact SQL

Open DataBase Connectivity (ODBC)

- **Objectif**: éliminer la relation entre l'application client et le serveur de BD. Développement des applications ouvertes
- Développée au début de 1992 et présentée comme une interface universelle d'accès aux services de données
- Conçue autour du modèle SQL

ARCHITECTURE DE ODBC



ODBC (suite)

- Architecture ODBC consiste en deux couches:
  - Gestionnaire de pilotes: définir le pilote adéquat pour la BD;
  - Pilote ODBC: traduit les requêtes SQL formulées par ODBC en message compréhensible par le SGBD cible.
- L'accès multi-source est possible sous certaines conditions mais il n'est pas transparent.

ODBC (suite)

- ODBC est devenu un standard par SAG (SQL Access Group);
- Il existe plus de 700 pilotes disponibles sur le marché;
- ODBC a amélioré la portabilité des applications clients;
- Evolution: RDO, DAO, OLE DB, ADO, ADO+
- Dans .NET ➔ ADO.NET.

Remote Procedure Call

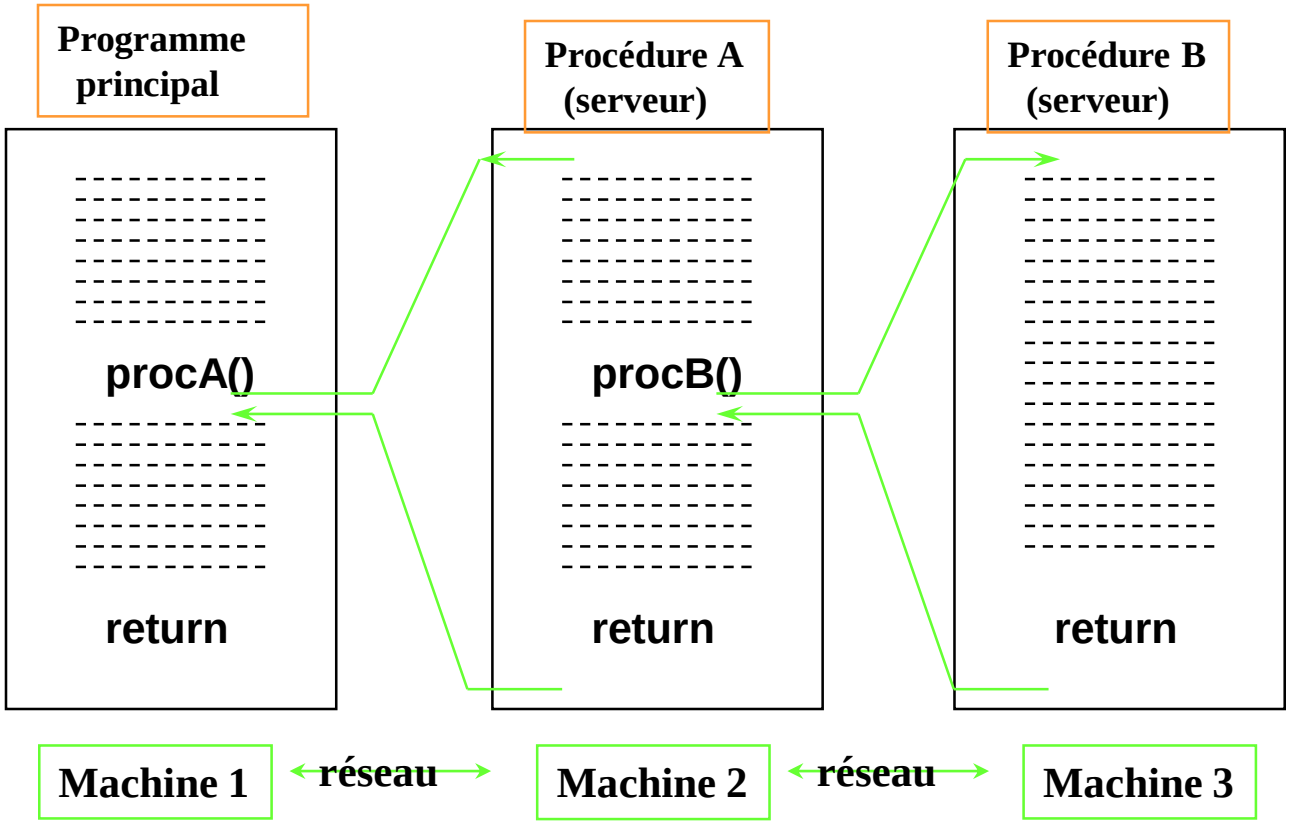
- RPC a été développé pour UNIX. Elle a été largement utilisée pour le développement des applications C/S;
- Objectif: l'invocation d'un service situé sur une machine distante (analogue à celui d'un appel de procédure local);
- Middleware basé sur RPC: **DCE** (Distributed Computing Environment) de l'OSF;
- DCE propose des standards de services pour la mise en place d'application C/S:
  - Service de sécurité (login et authentification);
  - Service de répertoires des ressources (annuaire);
  - Gestion de fichiers distribués (DFS);
  - RPC.
- RMI (J2EE), .NET Remoting (Microsoft .NET)

RPC

- Le modèle RPC utilise l'approche "conception orientée application" et permet l'exécution de procédure sur des sites distants;
- L'appel de la procédure distante constitue la requête cliente, le retour de la procédure constitue la réponse serveur;
- but : conserver le plus possible la sémantique associée à un appel de procédure conventionnel, alors qu'il est mis en oeuvre dans un environnement totalement différent;
- Un appel de procédure obéit à fonctionnement synchrone: une instruction qui suit un appel de procédure ne peut pas s'exécuter tant que la procédure appelée n'est pas terminée.

RPC : LE MODÈLE

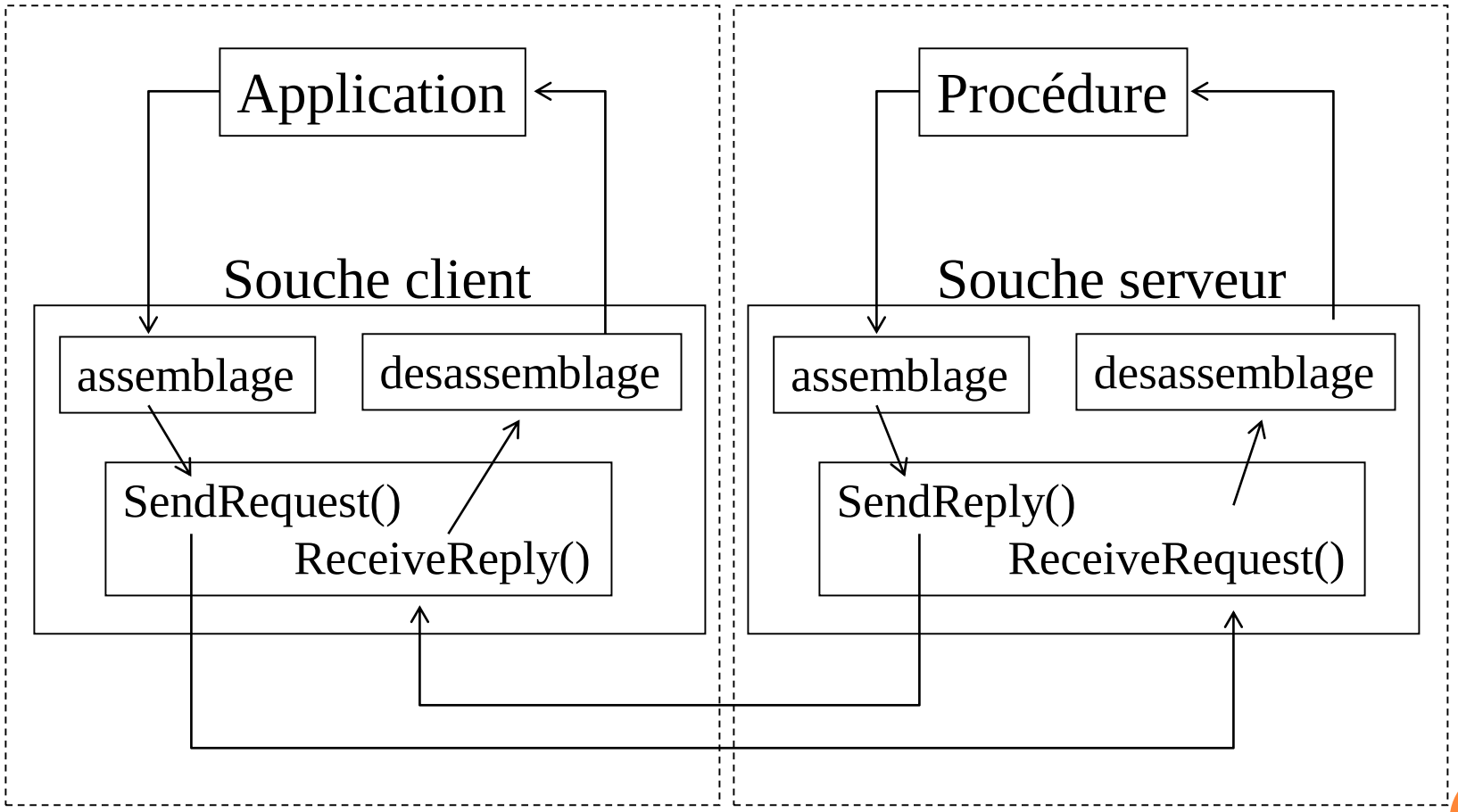
➢ Encapsulation du protocole de communication dans un appel de procédure:



Différences entre procédures distantes et procédures locales

- les temps de délais dûs au réseau peuvent engendrer des temps d'exécution considérablement plus long;
- Un appel de procédure distant ne peut contenir d'argument de type pointeur;
- Les descripteurs d'entrée/sortie ne sont pas accessibles au procédures distantes, leur interdisant l'utilisation de périphériques locaux (exemple écriture de messages d'erreur impossible).

Fonctionnement du RPC



Composants d'une application RPC

- Client
  - Localiser le serveur et s'y connecter
  - Faire des appels RPC (requêtes de service)
    - Emballage des paramètres
    - Soumission de la requête
    - Désemballage des résultats

Stub client + Primitives XDR

- Serveur
  - Attendre les requêtes du client et les traiter
    - Désemballage des paramètres
    - Appel local du service (procédure) demandé(e)
    - Emballage des résultats

Stub serveur + Primitives XDR

Les problèmes à résoudre

- Gestion du processus serveur:
  - Création;
  - Désignation et liaison;
- Transmission des paramètres et des résultats:
  - Espace d'adressage client et serveur disjoints;
  - Hétérogénéité des langages et des machines;
  - Passage de structures dynamiques (tableaux de taille variable, listes, arbres, graphes, etc.).
- Gestion des défaillances;
- Transparence.

Identification d'une procédure

- Principe: regrouper différentes procédures relatives à un même domaine ou à un service particulier en un *programme RPC*
- Les paramètres d'identifications sont:
  - Numéro du programme
  - Numéro de la version du programme à utiliser
  - Numéro de la procédure à appeler
  - Les différents paramètres de la fonction et leur type
- Un processus démon se charge de lancer le dialogue avec le *processus de service* qui exécutera la procédure souhaité

La mise en œuvre sous UNIX

chaque machine offrant des programmes RPC dispose d'un service d'association de port dynamique: le *port mapper*.

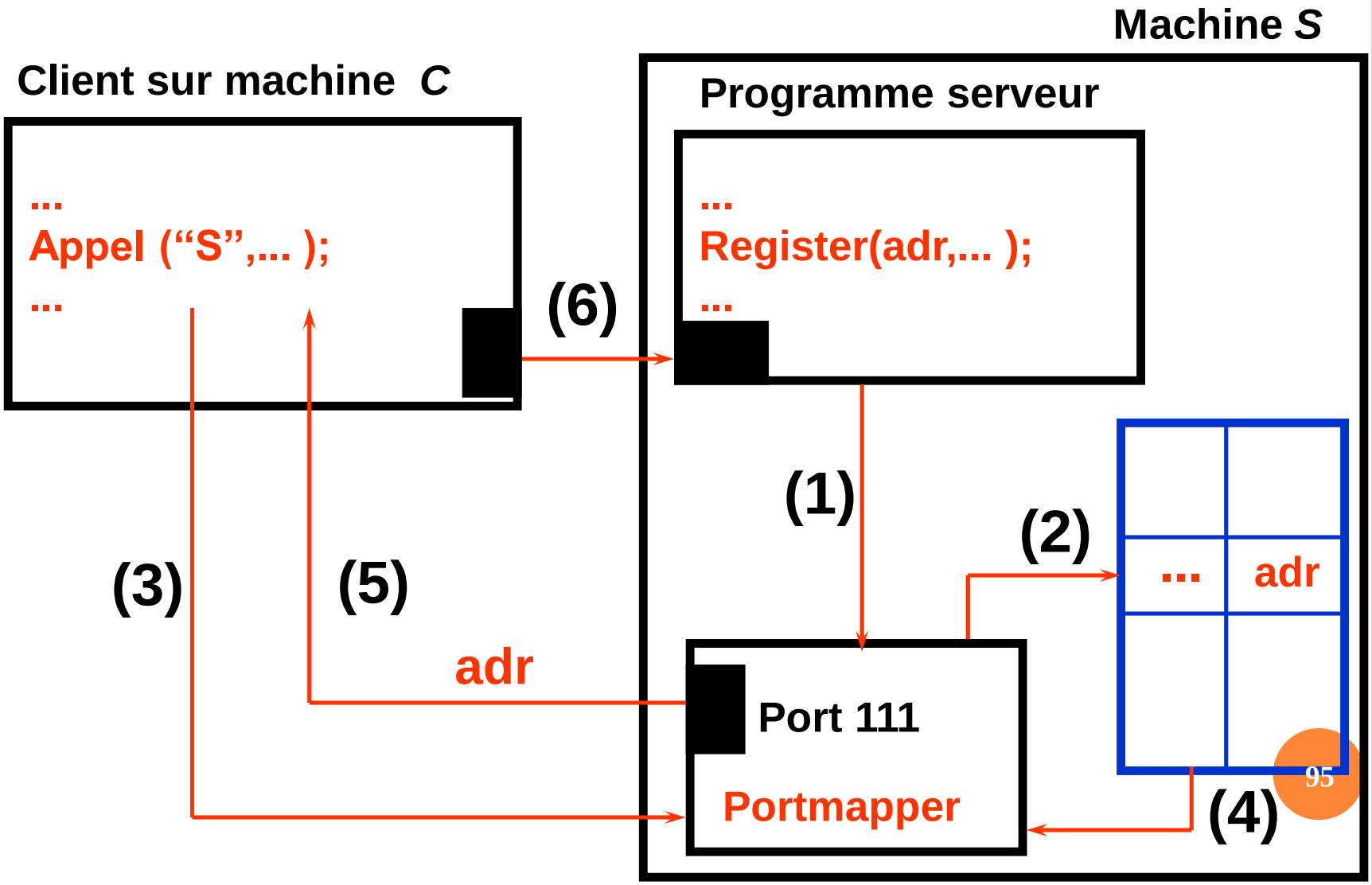
Lorsqu'un programme RPC (serveur) démarre, il alloue dynamiquement un numéro de port local, puis contacte le *port mapper* de la machine sur laquelle il s'exécute, puis informe ce dernier de l'association (identifier le programme RPC / numéro de port).

Le *port mapper* maintient une base de données renseignant les associations.

Lorsqu'un client désire contacter un programme RPC sur une machine M, il s'adresse au préalable au *port mapper* de M afin de connaître le port de communication associé.

Le *port mapper* s'exécute toujours sur le port de communication 111.

Nommage Dynamique



Problème d'hétérogénéité

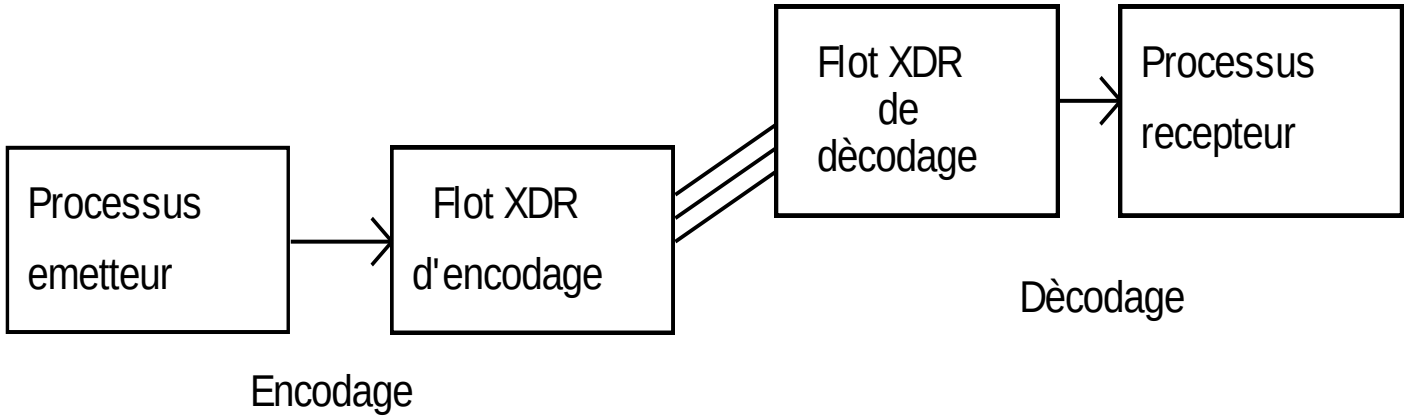
- Solution de Sun Microsystems
  - Format *eXternal Data Representation* ou *XDR*
  - Librairie XDR (types de données XDR + primitives de codage/décodage pour chaque type)
  - rpc/xdr.h

XDR: eXternal Data Representation

- Objectif: représentation standard des données
  - définit comment les données doivent être véhiculées sur un réseau.
  - permet d'échanger des données entre machines ayant des représentations internes différentes.
- Pratiquement:
  - propose des fonctions de conversion d'un objet typé de sa représentation locale vers une représentation normalisée et vice versa

PRINCIPE GÉNÉRAL DE XDR

- Les processus émetteur et récepteur associent respectivement leur sortie et leur entrée standard à un flot XDR d'encodage et de décodage
- XDR: le type d'un descripteur de flot



Opérations de l'encodage/décodage

- Principe:
  - À chaque type d'objet correspond une fonction spécifique agissant comme filtre
  - Un encodage, un décodage ou même une libération d'espace alloué au cours d'un décodage précédent sera réalisé selon la nature du flot XDR
  - Pour chaque type d'objet élémentaire *type* un filtre xdr-type est associé avec 2 paramètres:
    - Un pointeur sur un descripteur de flot de type XDR
    - Un pointeur sur un objet du type correspondant

Le traitement de type de base/1

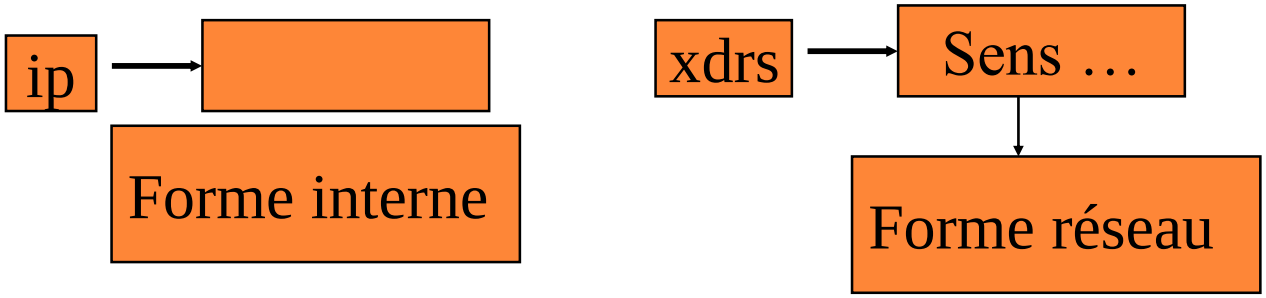
- Plusieurs fonctions existent:
  - ***Char*:** xdr_char (XDR *, char *);
  - ***Int*** : xdr_int(XDR *, int *);
  - ***Enum*** : xdr_enum(XDR *, enum_t *);
  - ***String*** : xdr_string(XDR **ptr_xdr*, char ***ptr*, const unsigned int *longueur_max*);
  - ***Vector*:** xdr_vector(XDR **ptr_xdr*, void **ptr*, unsigned int *longueur*, const unsigned int *taille_element*, xdrproc_t *xdr_type*);
  - ***Array*:** xdr_array(XDR **ptr_xdr*, void ***ptr*, unsigned int *ptr_longueur*, const unsigned int *longueur_max*, const unsigned int *taille_element*, xdrproc_t *xdr_type*)

Le traitement de type de base/2

- xdr_int (xdrs, ip)

  XDR *xdrs

  Int *ip



- xdr_array (xdrs, arrp, sizep, maxsize, elsize, elproc)

  *arrp: pointeur vers tableau

  *sizep: nombre d'éléments de taille elsize

  maxsize: nombre max d'éléments

  elproc: filtre xdr pour convertir l'élément

Structure définie par l'utilisateur

Struct *structure {*

type_1 champ_1;

type_2 champ_2; .....

type_n champ_n;

*xdr_structure(XDR *ptr_xdr,* **struct** *structure *ptr_objet) {*

return *(xdr_type_1(ptr_xdr, &ptr_objet->champ_1)*

&&(xdr_type_2(ptr_xdr, &ptr_objet->champ_2) ....

&&(xdr_type_n(ptr_xdr, &ptr_objet->champ_n));}

Niveaux de programmation RPC

- Couche haute:
  - Le réseau, le S.E et la machine sont transparents;
  - Utilisation des fonctions existantes uniquement.
- Couche intermédiaire:
  - Plus intéressante pour le développeur;
  - Connaissance minimal de XDR et RPC;
  - Aucune connaissance des sockets.
- Couche basse
  - Si les options choisies dans C.I sont insuffisantes (utilisation de TCP);
  - Beaucoup plus complexe.

RPC: Couche haute

- L'utilisation du RPC la plus simple
- Les fonctions:
  - **Getrpcport**:
    - obtenir le numéro de port associe a une version d'un programme donné
  - **Rusers**: renvoie le nombre d'utilisateurs connectés
  - **Rnusers**: fournit des informations plus complètes sur les utilisateurs connectés
  - **Rwall**: permet de demander l'envoi d'un message à tous les utilisateurs de la machine spécifiée

RPC: Couche intermédiaire

- Le point de vue du serveur
  - Fonction d'enregistrement

```
Int registerrpc (
    Unsigned long numero-programme,
    Unsigned long numero-version,
    Unsigned long numero-procedure,
    Void *(*fonction)(),
      xdrproc_t xdr-parametres,
     xdrproc_t xdr_resultat
);
```

  - L'attente de clients

```
void svc_run()
```

Couche intermédiaire (suite)

- L'effacement d'un service:

```
Void pmap_unset(
        unsigned int numero_programme,
        unsigned int numero_version
);
```

- Le point de vue des clients:
  - L'appel d'une fonction:

```
int callrpc(
    char *nom_machine,
    unsigned long numero_programme,
    unsigned long numero_version ,
    unsigned long numero_procedure ,
    xdrproc_t xdr_parametres,
    void *ptr-parametres
    xdrproc_t xdr-resultats,
    void *ptr_resultat
);
```

Couche intermédiaire (suite)

- Limitations:
  - Les applications s'appuient sur le protocole UDP. Cela limite la taille des informations.
  - Aucune procédure d'authentification des requêtes n'est mise en œuvre
  - Pas de diffusion

RPC: Couche basse

- Cette interface n'est utilisée que lorsque l'une au moins des options adoptées par la couche intermédiaire n'est pas adaptée au service à implémenter
- Les fonctionnalités fournies:
  - Création d'un service UDP et TCP
  - Appel de fonctions diffusé (broadcast);
  - Intégration des mécanismes d'authentification
  - Permettre des appels asynchrones
  - Charger le processus **inetd** de créer dynamiquement le processus d'un service RPC lorsqu'un client sollicite ce service

Gestion des défaillances/1

- Le client est incapable de localiser le serveur
  - Le serveur est en panne
  - L'interface du serveur a changé
  - Solutions : retourner -1 !!!, exceptions, signaux
- La requête du client est perdue
  - Temporisation et ré-émission de la requête
- La réponse du serveur est perdue
  - Temporisation et ré-émission de la requête par le client

Gestion des défaillances/2

- Problème : risque de ré-exécuter la requête plusieurs fois (opération bancaire !!!!)
- Solution : un bit dans l'en-tête du message indiquant s'il s'agit d'une transmission ou retransmission

- Le serveur tombe en panne après réception d'une requête
  - (i) Après exécution de la requête et envoi de réponse
  - (ii) Après exécution de la requête, avant l'envoi de la
  - réponse
  - (iii) Pendant l'exécution de la requête
  - Comment le client fait-il la différence entre (ii) et (iii) ?

Gestion des défaillances/3

- Trois écoles de pensée (sémantiques)
  - Sémantique *une fois au moins* : le client ré-émet jusqu'à avoir une réponse (RPC exécuté au moins une fois)
  - Sémantique *une fois au plus* : le client abandonne et renvoie un message d'erreur (RPC exécuté au plus une fois)
  - Sémantique *ne rien garantir* : le client n'a aucune aide (RPC exécuté de 0 à plusieurs fois)
- Le client tombe en panne après envoi d'une requête
  - Requête appelée *orphelin*. Que doit-on en faire ?

Gestion des défaillances/4

- Solutions de *Nelson*
  - Extermination : Le client utilise un journal de trace et tue les orphelins : solution coûteuse en espace et complexe
  - Réincarnation : Définition de périodes d'activité incrémentale du client. Après une panne, il diffuse un message indiquant une nouvelle période. Ses orphelins sont détruits.
  - Réincarnation douce : variante de la précédente. Un orphelin est détruit seulement si son propriétaire est introuvable.
  - Expiration : Chaque RPC dispose d'un quantum *q* de temps pour s'exécuter. Il est détruit au bout de ce quantum. Pb : valeur de q ?
  - Problème de ces solutions : si l'orphelin détruit a verrouillé des ressources ?!!!

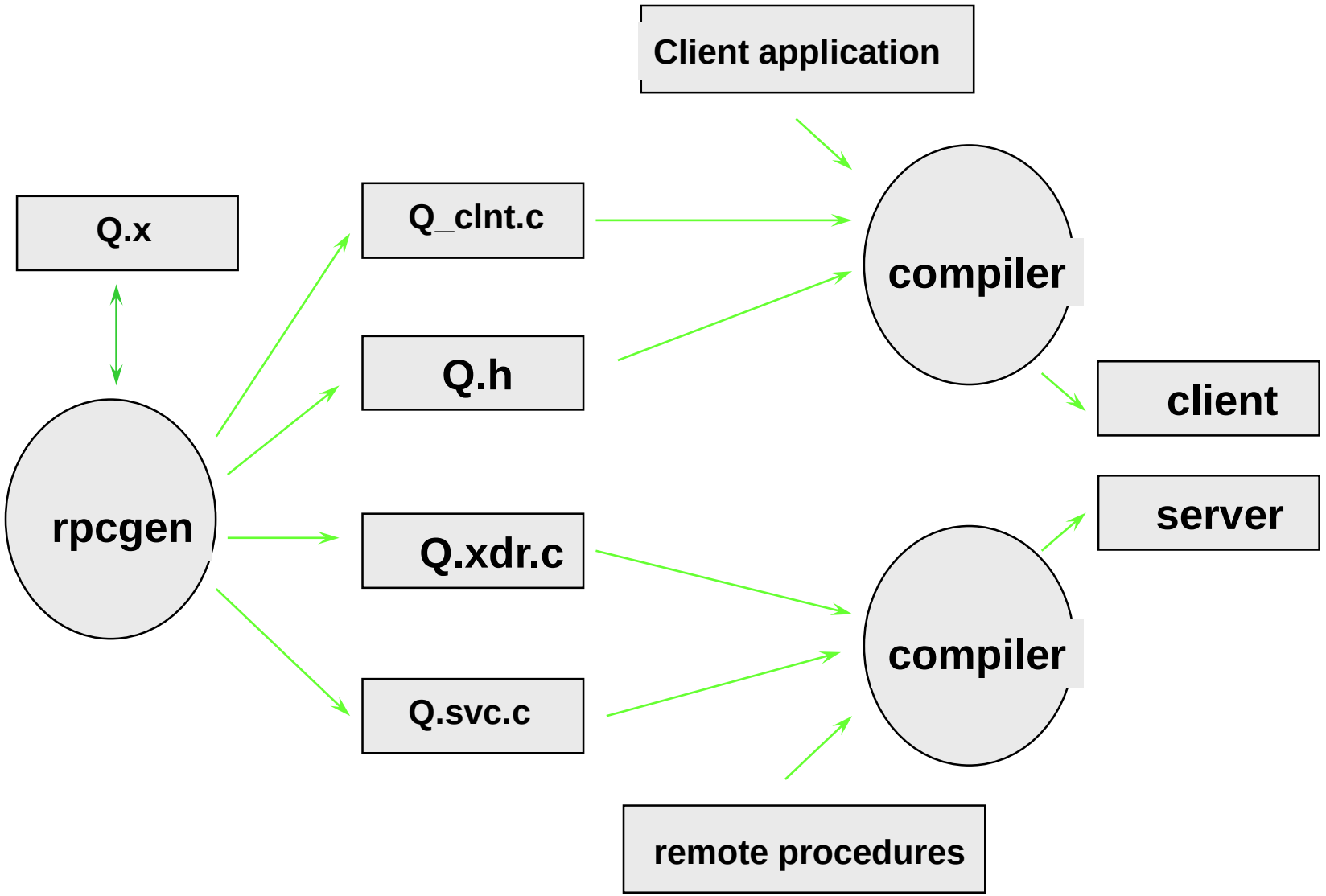
RPC : RPCGEN

- Rpcgen est un outil de génération de logiciel produisant
  - le talon client,
  - Le talon serveur,
  - les procédures XDR pour les paramètres et les résultats,
  - un fichier contenant les définitions communes.
- SUN fournit une méthodologie complète assistée par:
  - les routines de conversion XDR pour les types simples,
  - les routines XDR qui formatent les types complexes (tableaux et structures) utilisés dans la définition de messages RPC,
  - les fonctions run-time RPC qui permettent à un programme d'appeler une procédure distante, enregistrer un service auprès du *port mapper* , dispatcher une requête d'appel de procédure vers la procédure associée, à l'intérieur du programme distant;

RPC : RPCGEN

- rpcgen produit quatre fichiers source dont les noms sont dérivés du nom de la spécification en entrée. Si le fichier en entrée est Q.x, les fichiers générés sont:
  - Q.h : déclarations des constantes et types utilisés dans le code généré pour le client et le serveur,
  - Q_xdr.c : procédures XDR utilisés par le client et le serveur pour encoder/décoder les arguments,
  - Q_clnt.c : procédure « stub » côté client,
  - Q_svc.c : procédure « stub » côté serveur.

RPC : RPCGEN


Client application
Q.x
Q_clnt.c
compiler
Q.h
client
rpcgen
server
Q.xdr.c
compiler
Q.svc.c
remote procedures

LANGAGE RPCL

- spécification du prog, de la version et des fonctions

  program *nom-programme {*
  listes-versions
  }=numero-programme;

  Listes-versions est la concaténation de:

  Version *nom-version {*
  liste-procedures
  }=numero-version

  Listes-procedures est la concaténation de:

  Type-resultat nom-procedure(type-parametre)=numero-prcedure,

- La définition de constantes et de types

  Const identificateur=valeur-entiere;

- Les structures sont définies de la même manière qu'en C

Exemple RPC
La couche intermédiaire

- L'addition de deux entiers en utilisant une procédure add à distance
  - En entrée: 2 entiers
  - En sortie: un entier (la somme de deux entiers de l'entrée)
- Méthode:
  - La mise en œuvre de cette application nécessite trois fichiers:
    - Fichier commun
    - Fichier serveur
    - Fichier client

Fichier commun xdr_couple.c

```
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#define ADD_PROG 0x23333333
#define ADD_VERS1 1
#define ADD_PROC 1
struct couple {
int e1;
int e2;};
xdr_couple(xdrp, p)
XDR *xdrp;
struct couple *p;
{return (xdr_int(xdrp, &p->e1)&&(xdr_int(xdrp, &p->e2));
}
```

FICHIER SERVEUR

```
#include "xdr_couple.c"
struct couple *p;
Char *add(p)
{ static int res;
res=p->e1+p->e2;
Return((char*)&res);
}
main()
{ int rep;
 rep=registerrpc(ADD_PROG, ADD_VERS1,ADD_PROC,add, xdr_couple,
     xdr_int);

  if (rep==-1)

    { fprintf("erreur registerrpc (ass)\n");
      exit(2);}
Svc_run();
Exit(3);
}
```

Fichier client

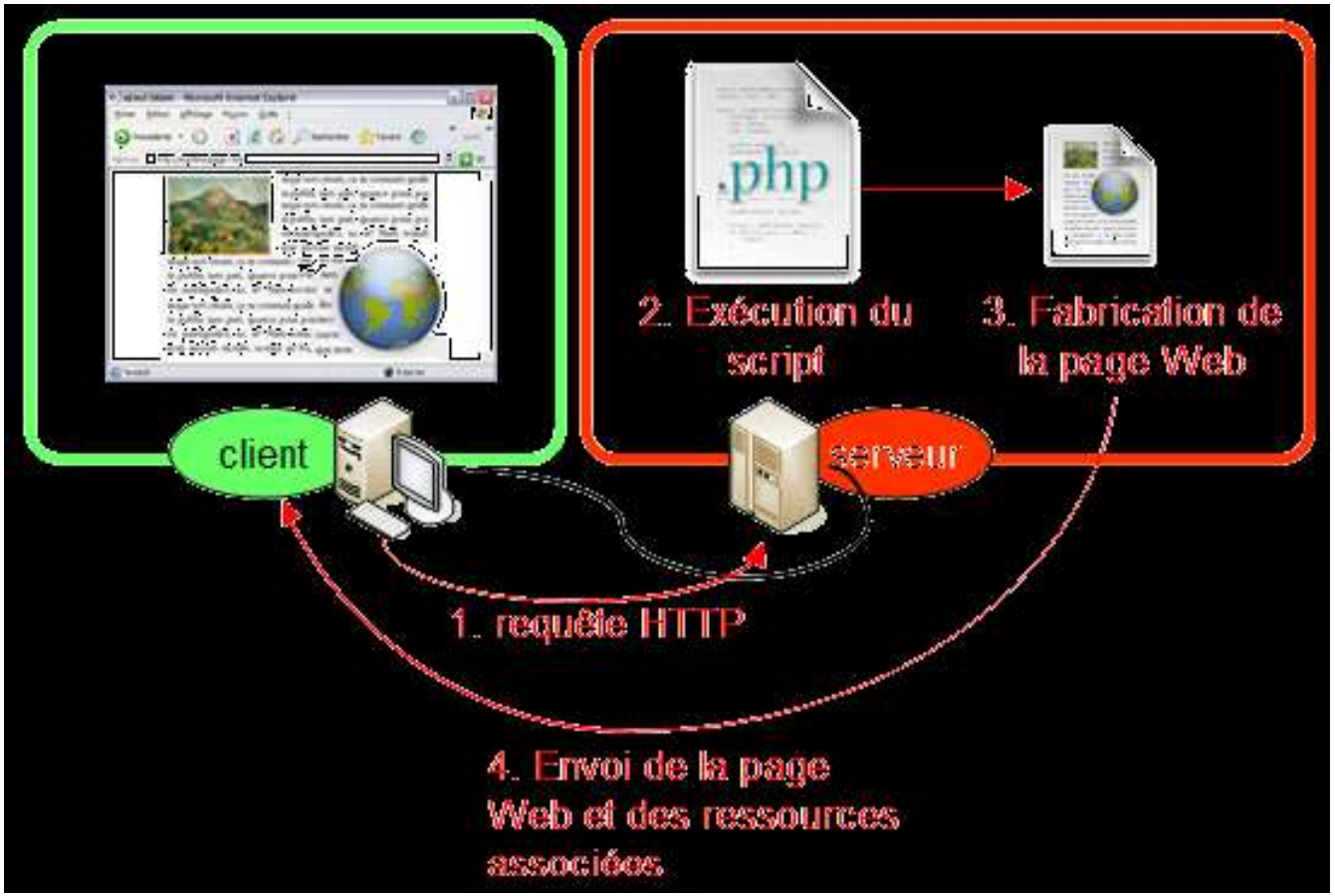
```
#include "xdr_couple.c"
Main(n,v)
Char *v[];
Int n;
{
Struct couple don;
Int *res;
Int op, m;
don.e1=12;
don.e2=5;
m=callrpc(v[1], ADD_PROG, ADD_VERS1, ADD_PROC, xdr_couple, &don,
    xdr_int, &res);
If (m!=0)
//visualiser la somme de e1 et e2
Else
//message d'erreur
```

Modèle Web

- Historique
  - Début 1990: apparition du premier prototype au CERN (Conseil Européen pour la Recherche Nucléaire) d'un système pour améliorer la diffusion des informations internes (1986 les bases, 1991 premier serveur).
    - Création d'un logiciel "World Wide Web" en C par Tim Berners-Lee capable de consulter un serveur HTTPD (HyperText Transfer Protocol Daemon)
    - World Wide Web s'appuie sur un protocole applicatif permettant de passer automatiquement du document consulté à un autre document lié: l'hypertexte
    - But affiché: être en mesure de centraliser les applications et de les consulter sur un poste client à l'IHM évoluée

Architecture

- Exemple:



World Wide Web

- 1991- Lancement officiel du Web
- 1991- Premier navigateur en mode texte
- 1992- Développement des premiers navigateurs graphiques
  - violawww, Lynx
- 1993- Lancement de Mosaic
- 1994- Lancement de Netscape Navigator
- 1994- Fondation du W3C
- 1995-…- Commercialisation du Web

W3C

- www.w3.org
- Processus de recommandation en quatre principales étapes:
  - Working Draft (WD)
  - Candidate Recommendation (CR)
  - Proposed Recommendation (PR)
  - W3C Recommendation (REC)
- Exemples de spécifications:
  - HTML, CSS, XML, XHTML, SOAP, WSDL, ….

Les acteurs du Web

- CERN (Conseil Européen pour la Recherche Nucléaire)
- NCSA (National Center for Supercomputing Applications)
- W3C (World Wide Web Consortium) fondé en 1994. initialement MIT+CERN+INRIA+Université de Keio puis de nombreux autres
  - Unifier les développement
  - Proposer une recommendation
- IETF: créer les standards (RFC)
- Microsoft, IBM, Sun,…
  - Standards de fait (frames, cookies, plug-in,…)

Les normes

- Le Web s'appuie sur des "normes":
  - HTTP comme protocole de transfert d'informations entre client et serveur
    - RFC 1945 (septembre 1997): HTTP 1.0
    - RFC 2616 (Juin 1999): HTTP 1.1
  - Une syntaxe d'adressage des ressources: les URL/URI/URN
    - RFC 3986 (Janvier 2005): définition de la syntaxe des URI
  - Un langage de description d'actions à exécuter:
    - ECMA-357 (décembre 2005): 2ième version de l'ECMAScript (Netscape et Microsoft ont chacune leurs variantes)

La norme HTTP: fonctionnement

- Basé sur un paradigme requête/réponse
  - Etablissement par le client d'une connexion puis envoi d'une requête sous la forme:

    Méthode-URI-version du protocole utilisé par le client

    Entête de la requête

    Corps de la requête
  - Réponse du serveur sous la forme:

    Ligne de statut

    Entête de la réponse

    Corps de la réponse
  - Déconnexion du serveur

HTTP: LES MÉTHODES

- Méthode GET:
  - Récupère le contenu de la ressource identifié par l'URI
  - Peut être associée à l'entête « if-modified-since » pour une récupération conditionnelle (emploi du cache)
- Méthode HEAD
  - Identique à GET mais récupère seulement l'entête de la réponse
  - Utilisée a des fins d'hyperliens (vérification d'existence)
- Méthode POST
  - Envoi de données au serveur sur l'URI spécifiée

HTTP: LES MÉTHODES

- Méthode PUT:
  - Demande au serveur d'enregistrer l'entité sous l'URI visé
- Méthode DELETE:
  - Demande de suppression de ressource
- Méthode LINK:
  - Demande d'établissement de liaisons entre l'URI visée et d'autres ressources
- Méthode UNLINK:
  - Suppression de liaisons entre l'URI visée et d'autres ressources

HTTP: L'ENTÊTE DE REQUÊTE

- Permettent la transmission d'information complémentaires
  - Sur la requête
  - Sur le client lui-même
- Agissent comme modificateurs de la requête
- Format: nom_entête: valeur de l'entête
- Des entêtes sont normalisés, le reste est considéré comme des champs d'entête propres à l'entité

Exemple

- Requête:

Connect to 196.200.135.4 on port 80 … ok

GET / HTTP/1.1

Host: www.ensias.ma

Connection: close

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Accept-Encoding: gzip

Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7[

Cache-Control: no

Accept-Language: de,en;q=0.7,en-us;q=0.3

Referer: http://web-sniffer.net

Exemple

- Réponse:

  HTTP/1.1 200 OK

  Date : Wed, 04 Feb 2009 04:51:56 GMT

  Server : Apache/2.2.3 (CentOS)

  X-Powered-By: PHP/5.1.6

  Set-Cookie: 3a43d2b0aad33b43e1269c3f820247b5=-; path=/

  Content-Type : text/html; charset-UTF-8

  Expires: Mon, 26 Jul 1997 05:00:00 GMT

  Last-Modified: Wed, 04 Feb 2009 04:59:26 GMT

  ……

- Outil de visualisation des entêtes http: http://web-sniffer.net/

HTTP: L'ENTÊTE DE REQUÊTE

- MIME (Multipurpose Internet Mail Extensions)
  - Sert à typer les documents transférés par HTTP
  - Format: Content-type: type_mime_principal/sous_type_mime
  - Exemple: Content-type:image.gif
  - Les grands types MIME: TEXT, APPLICATION, IMAGE, AUDIO, VIDEO, MESSAGE, MULTIPART (mixed, alternative, parallel,…)

HTTP: La réponse du serveur

- Les principaux codes d'état:
  - 1xx: indique une réponse provisoire
  - 2xx: les succès
  - 3xx: les redirections
  - 4xx: les erreurs clients
  - 5xx: les erreurs serveurs

HTTPS

- HTTPS n'est pas une norme mais une encapsulation d'HTTP au sein du protocole SSL décrit dans la RFC 2246
  - Les données envoyées sont encapsulées dans une structure
  - Le destinataire de la structure déchiffre la structure de données de manière à avoir les données originales

HTTP: LES URI

- URI pour Uniform Ressource Identifier
- Permet d'identifier la ressource que l'on souhaite consulter
- Format: nom de schema: chemin d'accès à la ressource
- Exemples:
  - ftp://ftp.iana.com:rfc/rfcxxx.txt
  - mailto:webaster@ensias.ma
  - http://www.google.fr
- Le schema HTTP
  - Format: http://nom-serveur ou adresse IP[:port]/chemin-absolu
  - Port=80 par défaut

HTTP: LES URI

- Liste des schemas gérés par l'IANA (Internet Assigned Numbers Autority) sur http://www.iana.org/numbers.html
  - Exemple:
    - RFC 2817: HTTP
    - RFC 4289: MIME
- Possibilité de récupérer un fragment d'une URI:
  - http://www.ensias.ma/index.htm#section
- Un chemin au sens HTTP est différent de son emplacement physique
  - Définition des chemins par les logiciels serveurs

Java RMI RemoteMethodInvocation

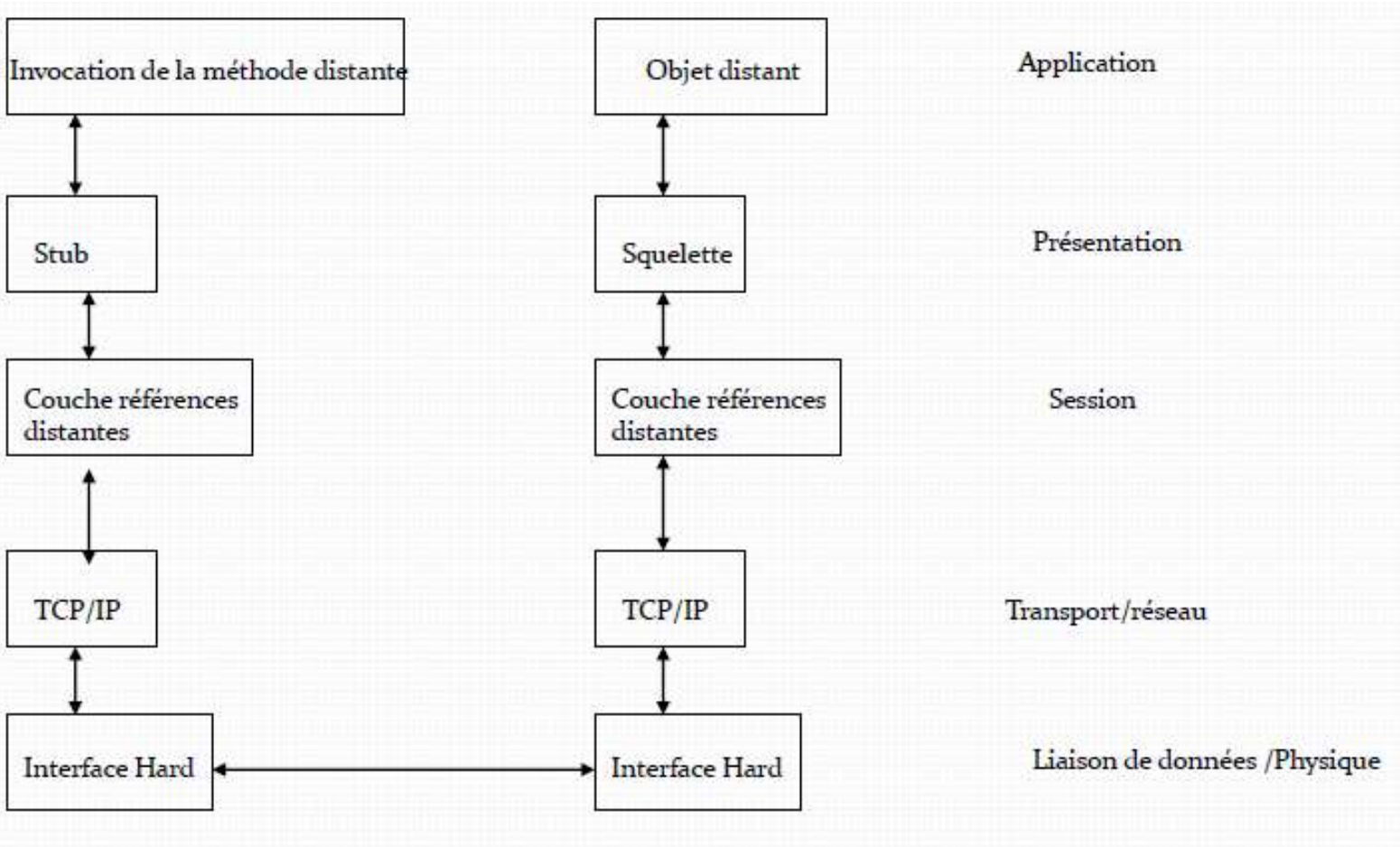
RMI Introduction

- RMI est un système d'objets distribués constitué uniquement d'objets java ;
- RMI est une **Application ProgrammingInterface (intégrée au JDK 1.1 et plus) ;**
- Développé par JavaSoft;

RMI

- Mécanisme qui permet l'appel de méthodes entre objets Java qui s'exécutent éventuellement sur des JVM distinctes ;
- L'appel peut se faire sur la même machine ou bien sur des machines connectées sur un réseau ;
- Utilise les sockets ;
- Les échanges respectent un protocole propriétaire :**RemoteMethodProtocol ;**
- RMI repose sur les classes de sérialisation.

ARCHITECTURE



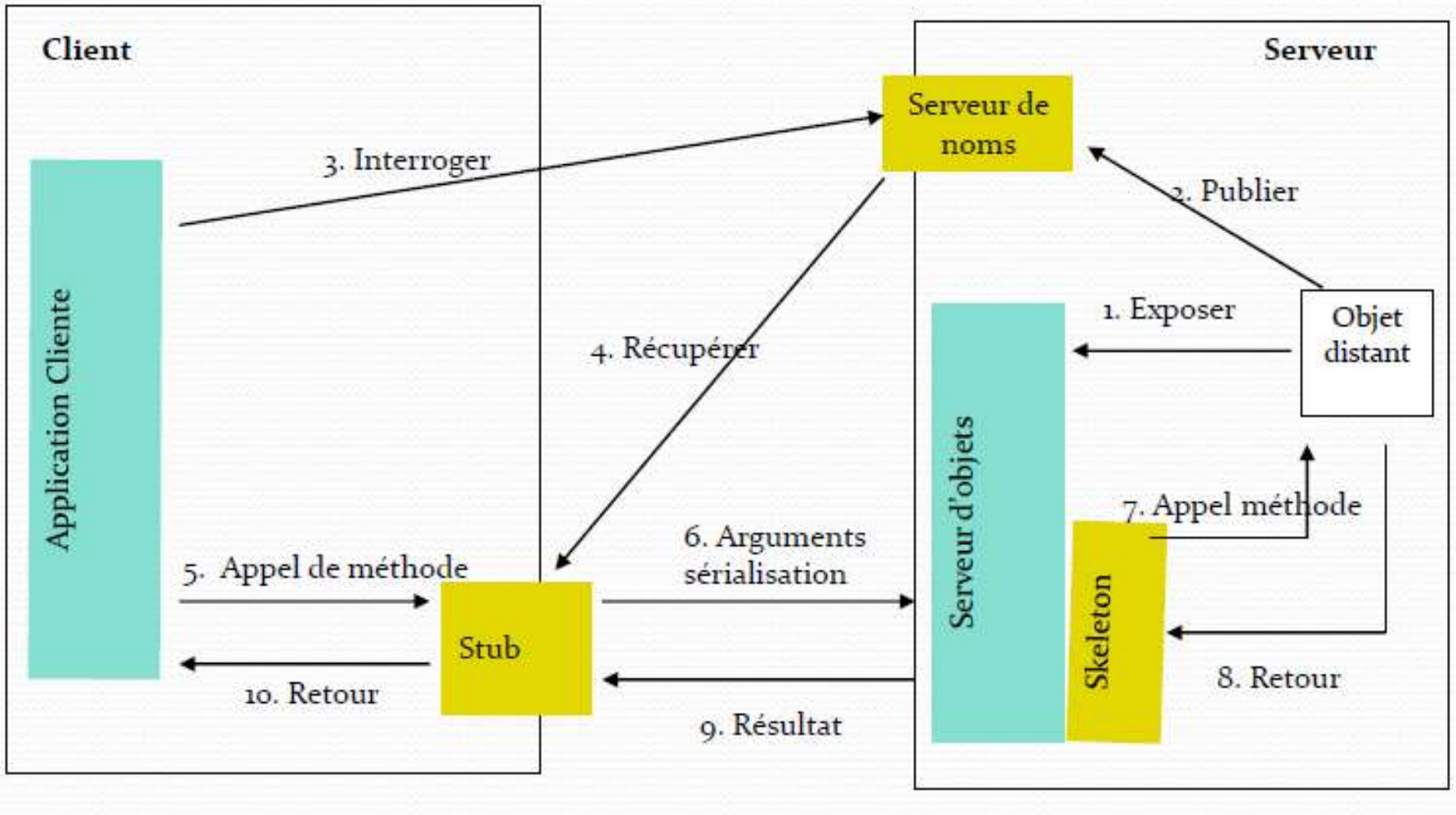
STUB/SKELETON

- Elles assurent le rôle d'adaptateurs pour le transport des appels distants;
- Elles réalisent les appels sur la couche réseau;
- Elles réalisent l'assemblage et le désassemblage des paramètres (*marshaling, marshaling);*
- Une référence d'objets distribué correspond à une référence d'amorce;
- Les amorces sont créées par le générateur **rmic.**

La couche des références d'objets
RemoteReferenceLayer

- Permet d'obtenir une référence d'objet distribué à partir de la référence locale au stub ;
- Cette fonction est assurée grâce à un service de noms **rmiregister** (les clés sont des noms et les valeurs sont des objets distants) **;**
- Un unique **rmiregisterpar JVM ;**
- **Rmiregister** s'exécute sur chaque machine hébergeant des objets distants ;
- **Rmiregister** accepte des demandes de service sur le port 1099.

ETAPES D'UN APPEL DE MÉTHODE DISTANTESTUB



Mise en oeuvre

1. Définir une interface distante **(Xyy.java) ;**
2. Créer une classe implémentant cette interface **(XyyImpl.java) ;**
3. Compiler cette classe **(javacXyyImpl.java) ;**
4. Créer une application serveur **(XyyServer.java) ;**
5. Compiler l'application serveur ;
6. Démarrage du registre avec **rmiregistry;**
7. Lancer le serveur pour la création d'objets et leur enregistrement dans **rmiregistry**
8. Créer une classe cliente qui appelle des méthodes distantes de l'objet distribué

(XyyClient.java) ;

10. Compiler cette classe et la lancer.

Inversion d'une chaîne de caractères à l'aide d'un objet distribué

- Invocation distante de la méthode **reverseString() d'un objet** distribué qui inverse une chaîne de caractères fournie par l'appelant. On définit :
  - **ReverseInterface.java:** interface qui décrit l'objet distribué;
  - **Reverse.java:** qui implémente l'objet distribué;
  - **ReverseServer.java:** le serveur RMI
  - **ReverseClient.java:** le client qui utilise l'objet distribué

FICHIERS

Côté Client

- l'interface : `ReverseInterface`
- le client : `ReverseClient`

Côté Serveur

- l'interface : `ReverseInterface`
- l'objet : `Reverse`
- le serveur d'objets : `ReverseServer`

Définition de l'interface distante

- Format
  - l'interface étend **java.rmi.Remote**
  - les méthodes doivent pouvoir lever **java.rmi.RemoteException**
- Exemple

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface ReverseInterface extends Remote
{ String reverseString(String chaine) throws
   RemoteException;
}
```

Implémentation de l'objet distant

- La classe de l'objet distant doit :
  - **implémenter l'interface distante**
  - **étendre une des sous-classes de java.rmi.server.RemoteServer comme java.rmi.server.UnicastRemoteObject**
- java.rmi.server.UnicastRemoteObject
  - sous classe le plus souvent utilisée;
  - appelle la classe squelette pour le désassemblage des paramètres ;
  - utilise TCP/IP pour la couche transport.

IMPLÉMENTATION DE L'OBJET DISTRIBUÉ DE NOTRE EXEMPLE

```
import java.rmi.*;
import java.rmi.server.*;
public class Reverse extends UnicastRemoteObject implements
ReverseInterface
{
public Reverse() throws RemoteException {
    super();
}
public String reverseString (String ChaineOrigine) throws
RemoteException {

    int longueur=ChaineOrigine.length();
    StringBuffer temp=new StringBuffer(longueur);
    for (int i=longueur; i>0; i--)
    {
        temp.append(ChaineOrigine.substring(i-1, i));
    }
    return temp.toString();
} }
```

RMIRegistry

- Implémentation d'un service de nommage
- Fourni en standard avec RMI
- Permet d'enregistrer des références sur des objets de serveur afin que des clients les récupèrent
- On associe la référence de l'objet à une clé unique (chaîne de caractères)
- Le client effectue une recherche par la clé, et le service de nommage lui renvoie la référence distante (le stub) de l'objet enregistré pour cette clé.

RMIREGISTRY

- Programme exécutable fourni pour toutes les plates formes
- S'exécute sur un port (1099 par défaut) sur la machine serveur
- Pour des raisons de sûreté, seuls les objets résidant sur la même machine sont autorisés à lier/délier des références
- Un service de nommage est lui-même localisé à l'aide d'une URL

RMIRegistry: La classe Naming

- La classe Naming du package java.rmi
  - permet de manipuler le RMIRegistry
  - supporte des méthodes statiques permettant de
    - Lier des références d'objets serveur
      - Naming.bind(...) et Naming.rebind(...)
    - Délier des références d'objets serveur
      - Naming.unbind(...)
    - Lister le contenu du Naming
      - Naming.list(...)
    - Obtenir une référence vers un objet distant
      - Naming.lookup(...)

Le serveur

- Programme à l'écoute des clients ;
- Enregistre l'objet distribué dans rmiregistry**Naming.rebind("rmi://Serveur:1099/Reverse", rev);**
- On installe un gestionnaire de sécurité si le serveur est amené à charger des classes (inutile si les classes ne sont pas chargées dynamiquement)**System.setSecurityManager(new RMISecurityManager());**

Le serveur de notre exemple

```
import java.rmi.*;
import java.rmi.server.*;

public class ReverseServer {
public static void main(String[] args)
{
    try {
   System.out.println( "Serveur : Construction de l'implémentation ");
   Reverse rev= new Reverse();
   System.out.println("Objet Reverse lié dans le RMIregistry");
   Naming.rebind("rmi://Serveur:1099/MyReverse", rev);
  System.out.println("Attente des invocations des clients …");
  }
   catch (Exception e) {
       System.out.println("Erreur de liaison de l'objet Reverse");
       System.out.println(e.toString());
        }
} // fin du main
} // fin de la classe
```

Le client

- Le client obtient un stub pour accéder à l'objet par une URL RMI**ReverseInterface ri = (ReverseInterface)Naming.lookup("rmi://Serveur:1099/MyReverse");**
- Une URL RMI commence par **rmi://, le nom de machine, un numéro de port optionnel et le nom de l'objet distant.rmi://hote:2110/nomObjetPar défaut, le numéro de port est 1099 défini (ou à définir)**

Le client de notre exemple

```
import java.rmi.*;

public class ReverseClient
{
    public static void main (String [] args)
    {   System.setSecurityManager(new RMISecurityManager());
        try{
         ReverseInterface rev = (ReverseInterface) Naming.lookup
               ("rmi://Serveur:1099/MyReverse");
         String result = rev.reverseString (args [0]);
         System.out.println ("L'inverse de "+args[0]+" est "+result);
        }
        catch (Exception e)
        {
            System.out.println ("Erreur d'accès à l'objet distant.");
            System.out.println (e.toString());
        }
    }

}
```

Le client

- Pour que le client puisse se connecter à rmiregistry, il faut lui fournir un fichier de règles de sécurité **client.policy.**

client.policy

```
grant
{
permission java.security.AllPermission;
};
```

EXECUTION

- Lancer le serveur :**C:>java ReverseServer&Serveur :Construction de l'implémentationObjet Reverse lié dans le RMIregistryAttente des invocations des clients …**
- Exécuter le client :**C:>java -Djava.security.policy=client1.policyReverseClient AliceL'inverse de Alice est ecilA**

MAIS COMMENT FAIRE POUR CES APPLICATIONS?

- surveillance de l'état des machines, des systèmes d 'exploitation et d 'applications;
- flot continuel de données en provenance de sources diverses sur le réseau;
- les éléments du système peuvent apparaître, disparaître, migrer, etc.
- les administrateurs doivent pouvoir accéder à l 'information quel que soit leur localisation.

Solution traditionnelle

- interrogation régulière des éléments à surveiller par l ’application d ’administration et m-à-j d’une base de données centralisée:
  - utilisation d’une configuration complexe afin de connaître l’ensemble des éléments à surveiller;
  - maintien de cette configuration lorsque des machines ou des applications rejoignent, quittent ou se déplacent dans le système.
- interrogation par les administrateurs de la base centrale.

MOM

- **MOM (**Message Oriented Middleware)
- **Solution basée sur les messages:**
  - les différents éléments administrés émettent des messages:
    - chgt d'état et de configuration;
    - alertes, statistiques.
  - un ou plusieurs démons reçoivent ses notifications et maintiennent l'état courant du système :
    - suivi des chgts de configuration dynamique.
    - émission de messages signalant les changements d'état significatifs ou les mises à jour.
- analogie:
  - mail (asynchronisme), liste de diffusion (multicast), News (subject based)

MOM (SUITE)

- Objectif: transport de l'information sous forme des messages indépendants;
- Supporte les communications synchrones et asynchrones;
- Les middleware basés sur les messages sont de 3 catégories:
  - Message passing;
  - Message queuing;
  - Publication/abonnement.

Message Passing

- Modèle de communication directe: program-to-program;
- Intéressant pour les applications fortement liées et dépendant du temps;
- supporte les modes synchrone et asynchrone;
- supporte des connexions concurrentes vers plusieurs serveurs.
- Exemple:
  - PIPES de PeerLogic.
  - MPI (*Message Passing Interface*) est un standard:
    - bibliothèques de communication par passage de messages.

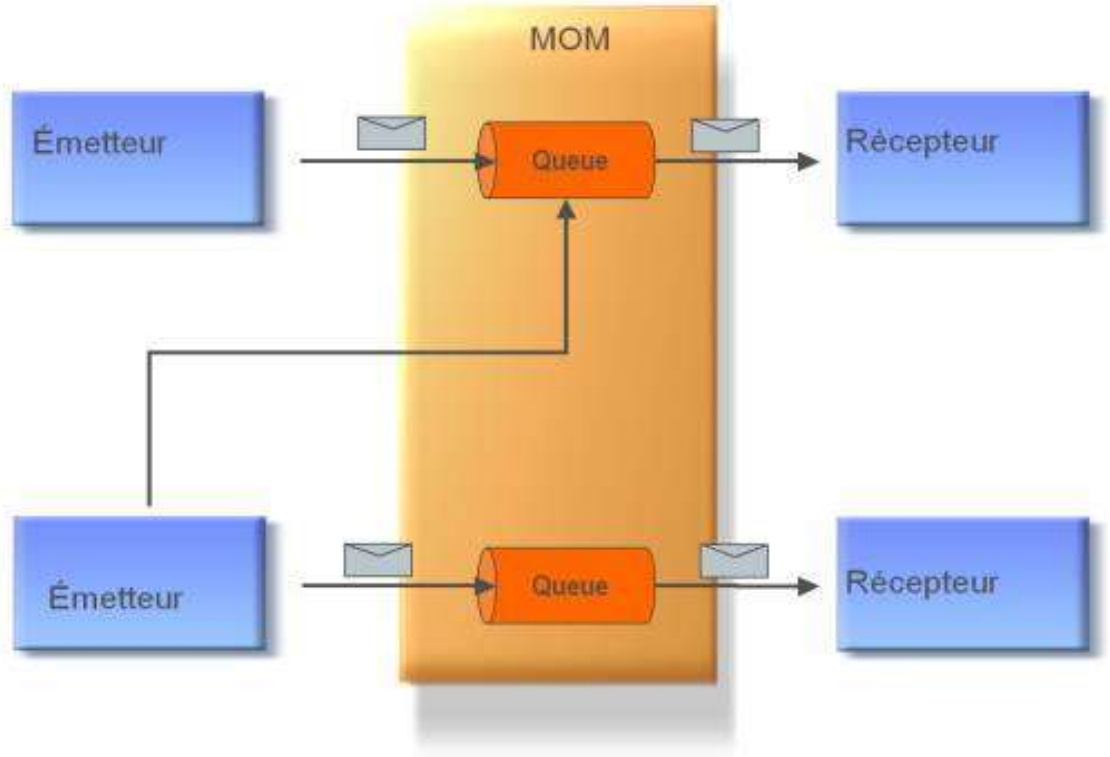
Message Queuing

- Remplace la connexion directe entre les applications par une queue de messages;
- Communication point à point;
- indépendance de l'émetteur et du destinataire;
- Anonymat:
  - l'émetteur n 'a pas à reconnaître l 'identité du destinataire; il peut le designer par un nom logique, de groupe ou par un ensemble de propriétés.
- gestion de l'hétérogénéité:
  - des données, des systèmes et des systèmes de communication.

Message Queuing

- Cela facilite:
  - Implémentation des politiques de priorité;
  - Balancement de la charge (+ serveurs prélèvent messages de la même queue);
  - Gestion d'applications tolérantes aux pannes;
  - Amélioration des performances (client non bloquant).

Message Queuing



Gestionnaire de messages

- Responsable de la gestion de queues de messages
- services fournis:
  - délivrance fiable des messages
  - garantir la délivrance des messages
  - garantir la non-duplication des messages délivrés
- Gestion de la priorité
- ordonnancement

Gestionnaire de messages

- La gestion de la persistance des messages:
  - les queues de messages **non-persistants** sont perdus après un arrêt du système ;
  - les queues de messages **persistants** sont stockés sur le disque .
- **performance**: les queues de messages non-persistants;
- **fiabilité**: les queues de messages persistants ;
- le choix dépend de l'application .

Exemples

- Message Queuing est un modèle de communication très utilisé dans les MOM actuels:
  - MSMQ (Microsoft)
  - MQSeries (IBM)
  - JMS (SUN)
- Modèle facile à utiliser pour interconnecter les applications

Limitations

- Message queuing a ses limites:
  - l'émetteur doit connaître le nom et l'adresse du destinataire
  - envoi à un groupe: plusieurs envois, un pour chaque destinataire

Publication/Abonnement

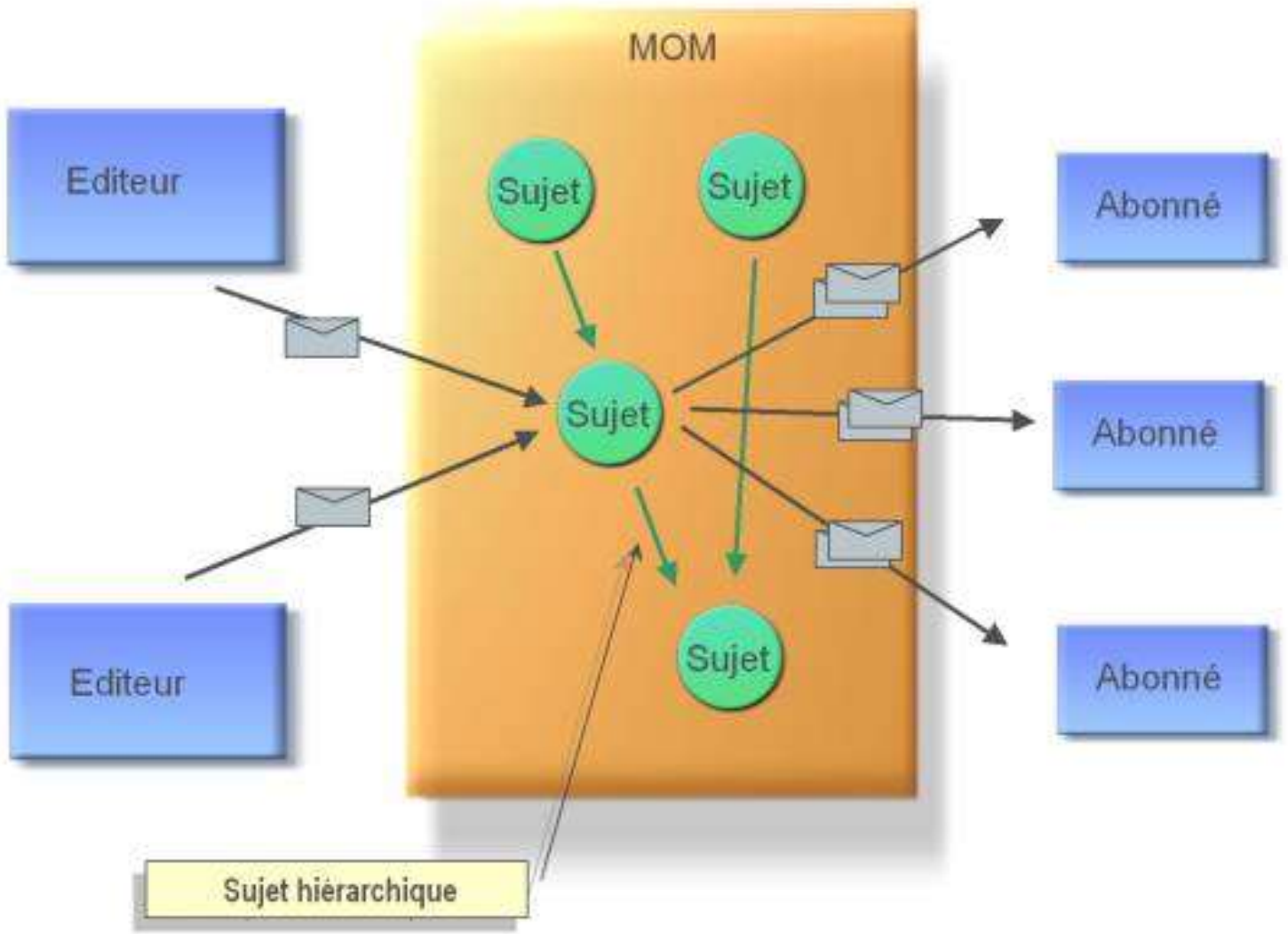
- Publication/Abonnement est similaire à l'abonnement dans un journal;
- le message est envoyé à un gestionnaire qui se charge de le répartir à ceux qui se sont abonnés à ce type de message;
- Communication multipoint .

Publication/abonnement (suite)

- Permet aux clients (ou aux composants serveur) de faire connaître leur intérêt pour certains messages en se faisant enregistrer auprès d'un gestionnaire d'événements.
- Il y a 2 types de participants:
  - fournisseurs (providers) qui envoient des événements dans le système.
  - consommateurs (consumers) qui s'abonnent à certaines catégories d 'événements du système sur certains critères.

Critères d'abonnement

- Il existe deux politiques d 'abonnement:
  - *subject-based*: la plus répondue et la plus utilisée. proposée par l 'interface JMS (Java Message Service) de SUN
  - *content-based*: une alternative à la technique subject-based. Elle permet un filtrage parmi les différentes valeurs
- *content-based* est plus général car il peut être utilisé pour implémenter le *subject-based*


MOM
Editeur
Editeur
Sujet
Sujet
Sujet
Sujet
Abonné
Abonné
Abonné
Sujet hiérarchique

Les principales caractéristiques

- **Découplage** des applications:
  - possibilité d'ajouter ou d'enlever un éditeur ou un abonné sans impacter le fonctionnement du système.
- **Rapidité** d'échange des messages :
  - utilisé dans environnement à fort volumétrie et à temps contraint, il utilise généralement des protocoles de transport naturellement dédiés à ce mode de communication tels que IP multicast.

Exemple : Tibco

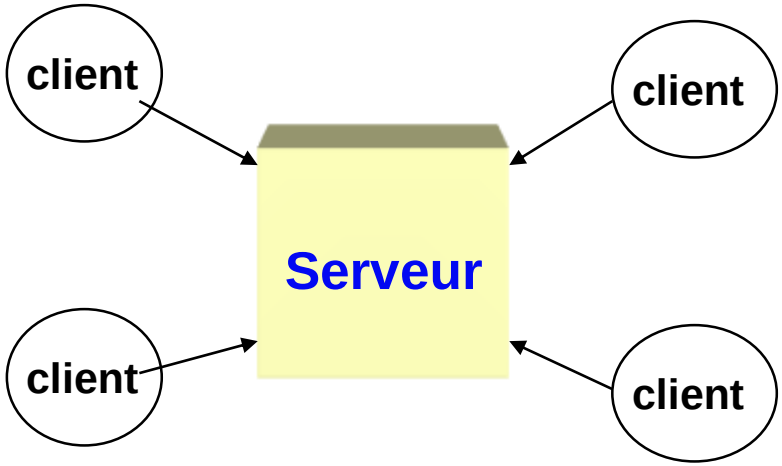
- Inventeur de la technologie: http://www.tibco.com
- Architecture: bus de messages
  - Protocole fiable de multicast
  - Un démon par host pour émettre et recevoir
  - Routeurs intelligents de messages
  - Propriétés:
    - Routage de messages en fonction de leurs sujets
    - Fiable et garantie de délivrance

Synthèse

	Communication synchrone	Communication asynchrone	Concentration de messages	Sécurité	diffusion
Message passing	●			●	
Message Queuing		●	●	●	
Publication/Abonnement		●	●		●

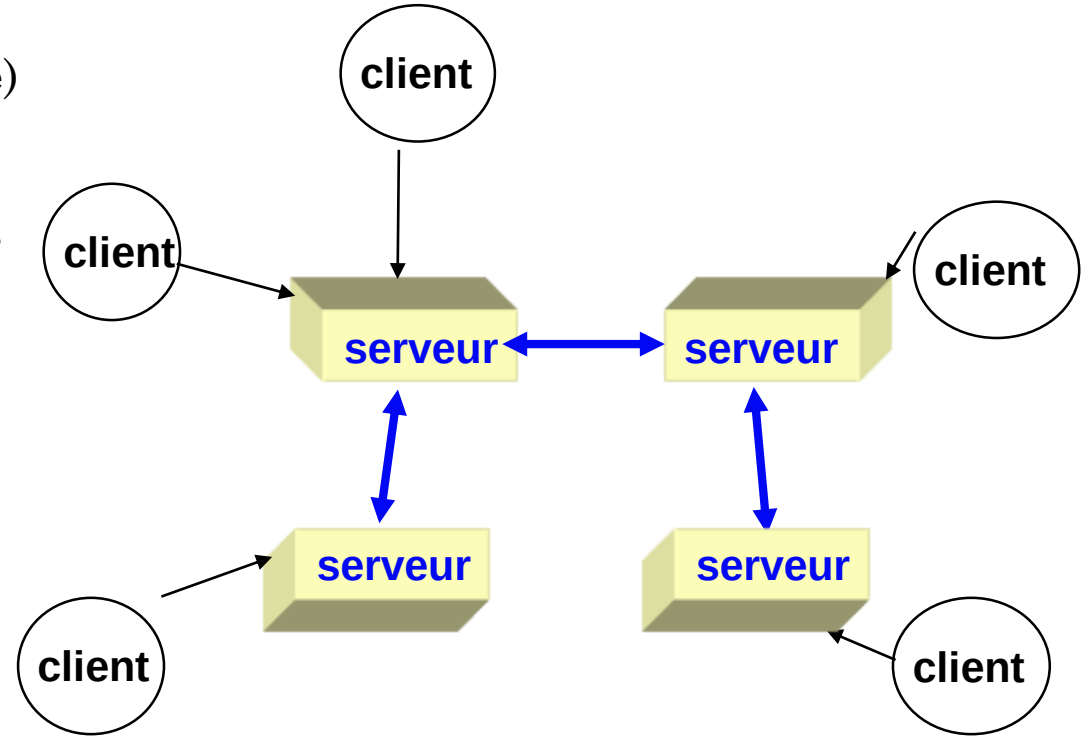
Implantation

- Serveur centralisé (Hub & spoke)
  - Simplicité de mise en œuvre
  - Peu tolérant aux pannes
  - Passage à l'échelle difficile



Implantation

- Serveur réparti (Snow flake)
  - Chaque serveur connaît un ensemble d'autres serveurs
  - Routage des messages
  - Répartition de la charge
  - Fiabilité relative
  - Passage à l'échelle



Support des MOMs: J2EE et .NET

- J2EE
  - Spécification JMS (Java Messaging Service)
  - Support les MQ et Pub/abon.
  - Composants logiciels: JORAM, OpenJMS (OpenSource)
- .NET
  - **MSMQ** (**M**icro**S**oft **M**essage **Q**ueuing)
  - Service a fait son apparition dans Windows NT4
  - Support uniquement les MQ

TPM: Transaction Process Monitors

- Contrôle le traitement des transactions dans les bases de données
- implémenté comme middleware à 3 niveaux
- Gestion des communications client/serveur:
  - Invocation des composantes de l'application au moyen de plusieurs paradigmes :RPC, MOM,...

Propriétés des transactions

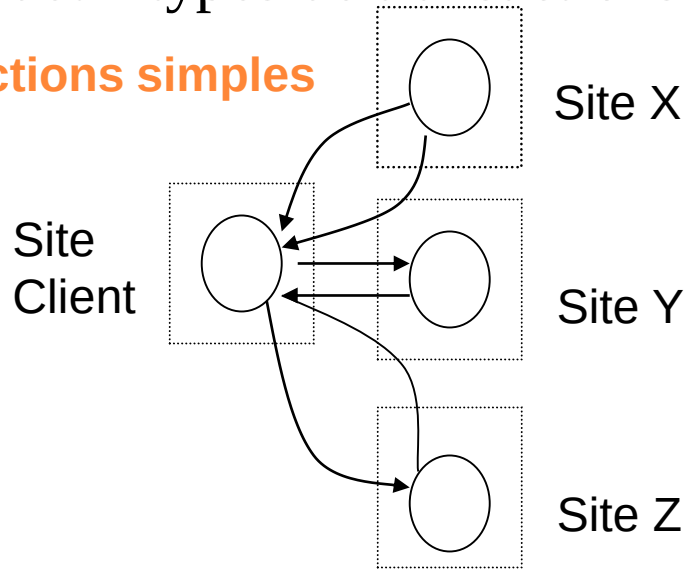
- Principe ACID:
  - **A**tomicité: exécution selon le principe du tout-ou-rien.
  - **C**ohérence: l'exécution garde la cohérence des données de la BD
  - **I**solation: les mises à jours d 'une transaction n'ont pas d'effets visibles avant sa fin.
  - **D**urabilité: l'effet d'une transaction doit être durable. Même après une panne.

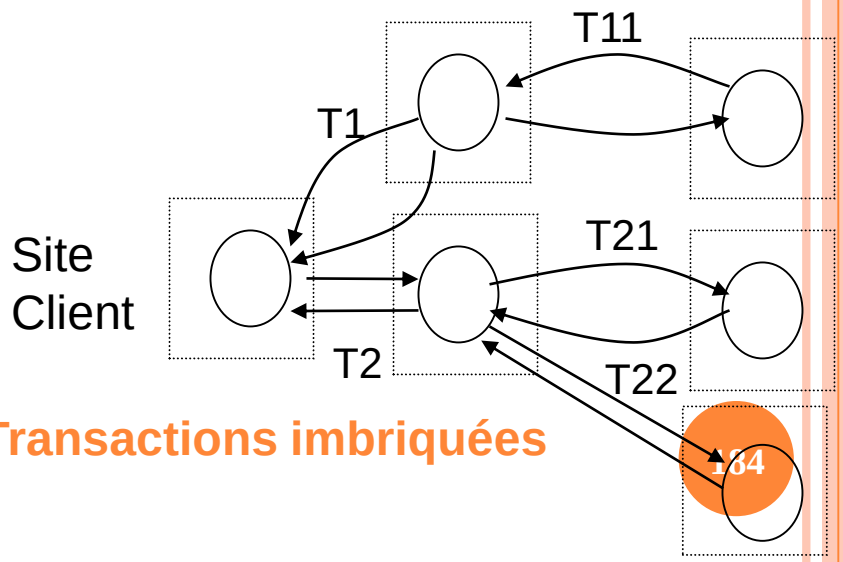
TRANSACTIONS RÉPARTIES

- Une transaction répartie implique plusieurs sites.
  - Un client demande le transfert d'argent entre deux de ses comptes
  - Un client demande le transfert d'argent entre comptes dans des banques différentes.

- On a deux types de transactions réparties:

Transactions simples

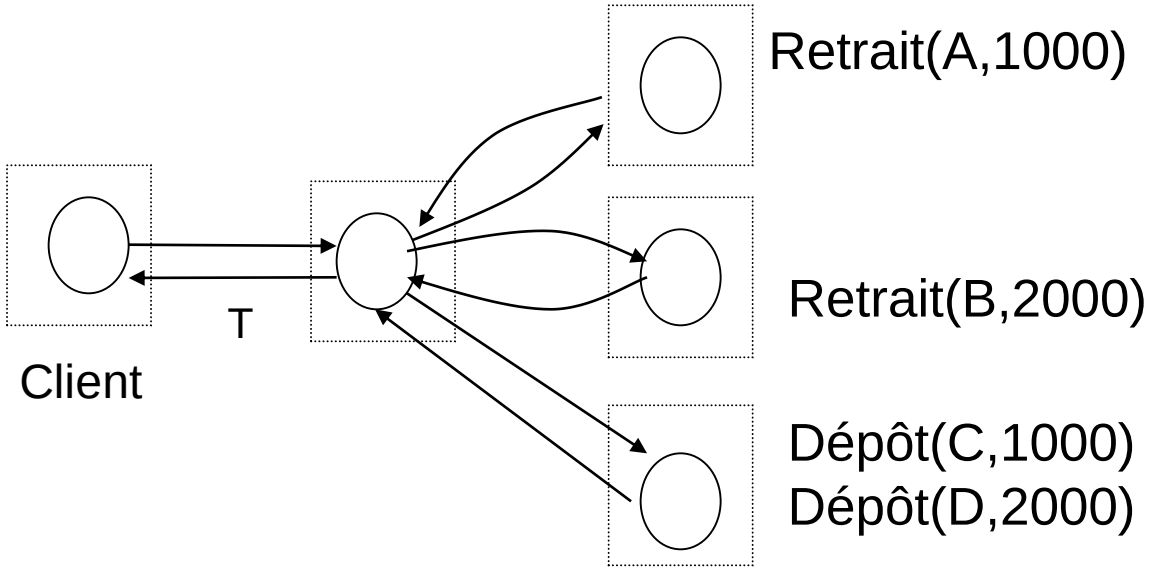




Transactions imbriquées

Transactions imbriquées

- Utilisées lorsqu ’une transaction nécessite la collaboration de plusieurs sites:



Protocole tow-phase commit

- Protocole de validation à 2 phases;
- Utile pour les opérations sur plusieurs sites;
- Synchronise les mises à jour;
- Standard OSI-TP de l'ISO définit l'implémentation du protocole: 1992;

Quand faut-il utiliser un TPM

- D'après Standish Group, un TPM est intéressant à utiliser pour Les applications C/S :
  - qui ont plus de 100 clients;
  - Traite plus de 50 transactions par minute ;
  - Utilise plus de 3 serveurs physiques et/ou plus d'une base de données.

Support TPM sur J2EE et .NET

- J2EE
  - Spécification **JTS** (Java Transaction Service)
  - Spécification **JTA** (Java Transaction API) décrit la communication de JTS avec les applications

- .NET
  - **MTS** (**M**icro**S**oft Transaction Server).
  - **DCOM** est le mécanisme responsable de la communication entre les clients et les composants MTS;

Middleware Orienté Objet

- Les middlewares les plus connus sont:
  - CORBA (Common Object Request Broker Architecture) de OMG;
  - DCOM (Distributed Common Object Model) de Microsoft;
  - OTM (Object Transaction Monitor) utilise la technologie objet en plus de la robustesse et les performances des moniteurs TP.

JMS: Java Message Service

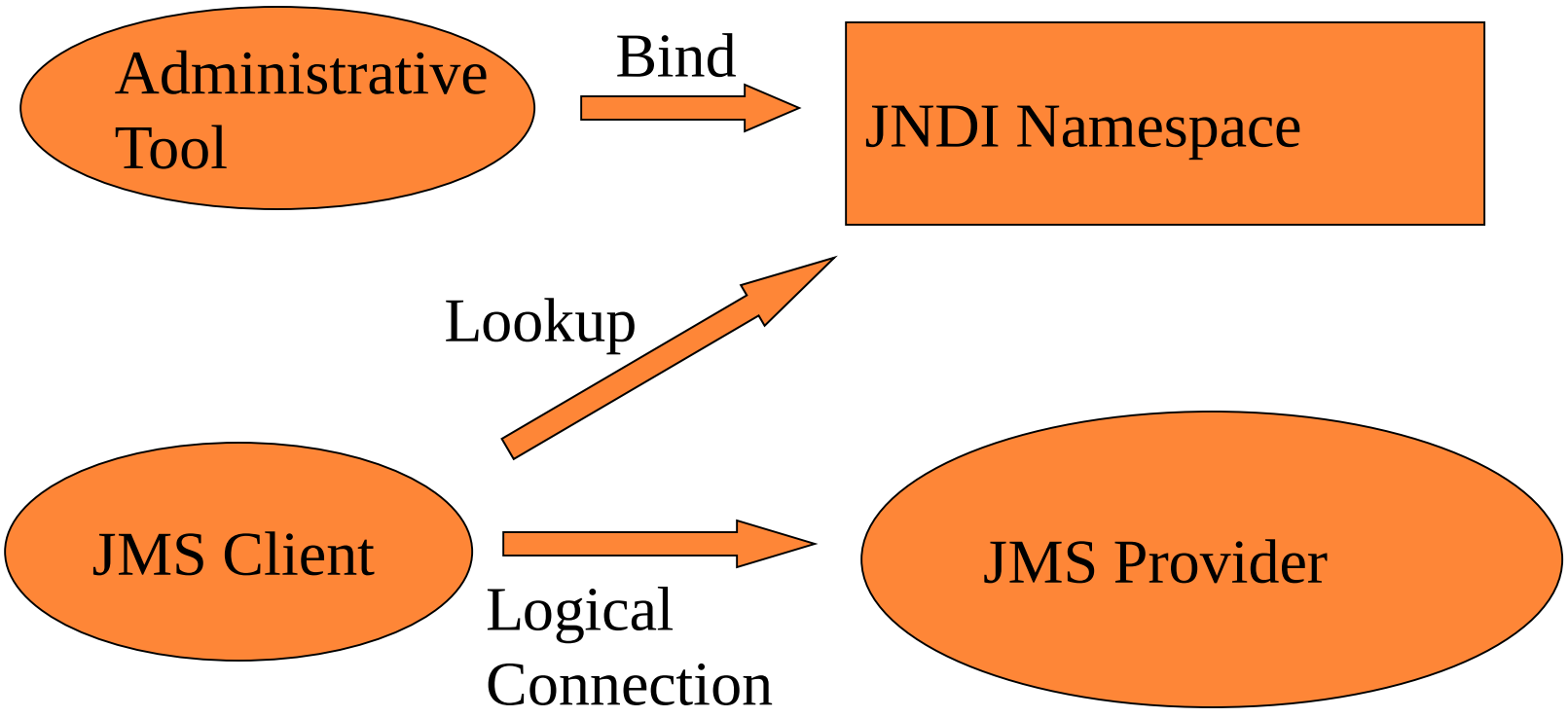
Specification

- Une démarche permettant à des programmes Java, de créer, d'envoyer et de recevoir des messages d'entreprise distribués.
- Communication faiblement couplée;
- Synchrone et asynchrone message;
- Fiabilité: Ack, pas de redondance de message;

Application JMS

- JMS Clients
  - Programme Java qui envoi et reçoit des messages;
- Messages
- Administered Objects
  - Objet JMS configuré crée par un administrateur
  - ConnectionFactory, Destination (queue or topic)
- JMS Provider
  - Système d'administration de message

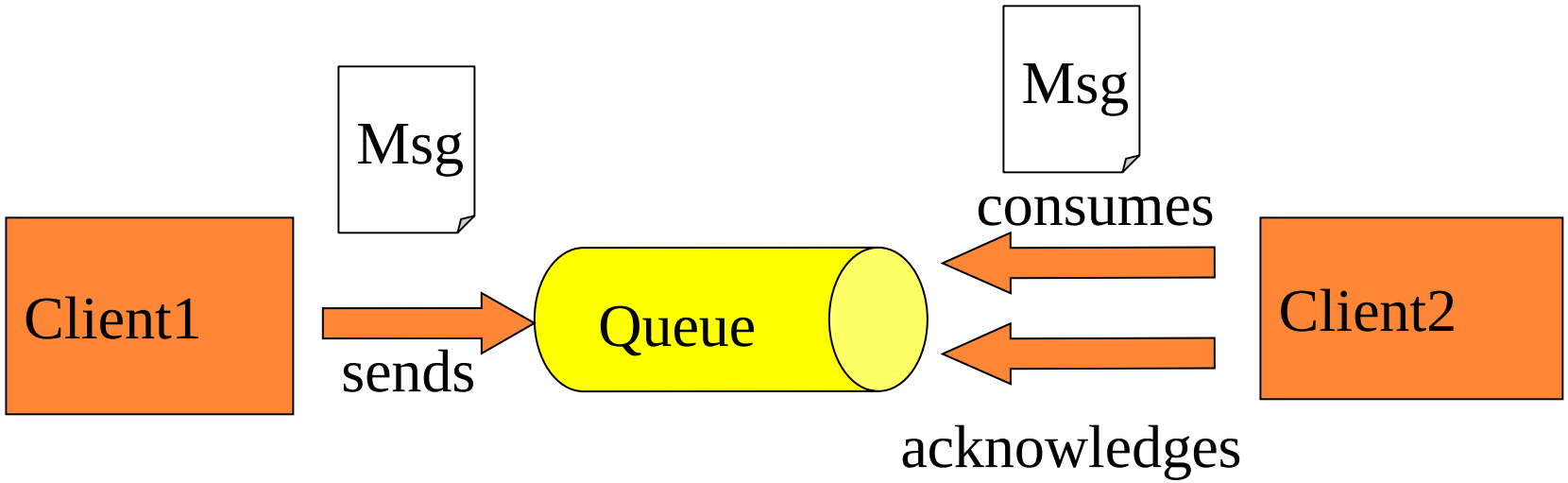
JMS Administration



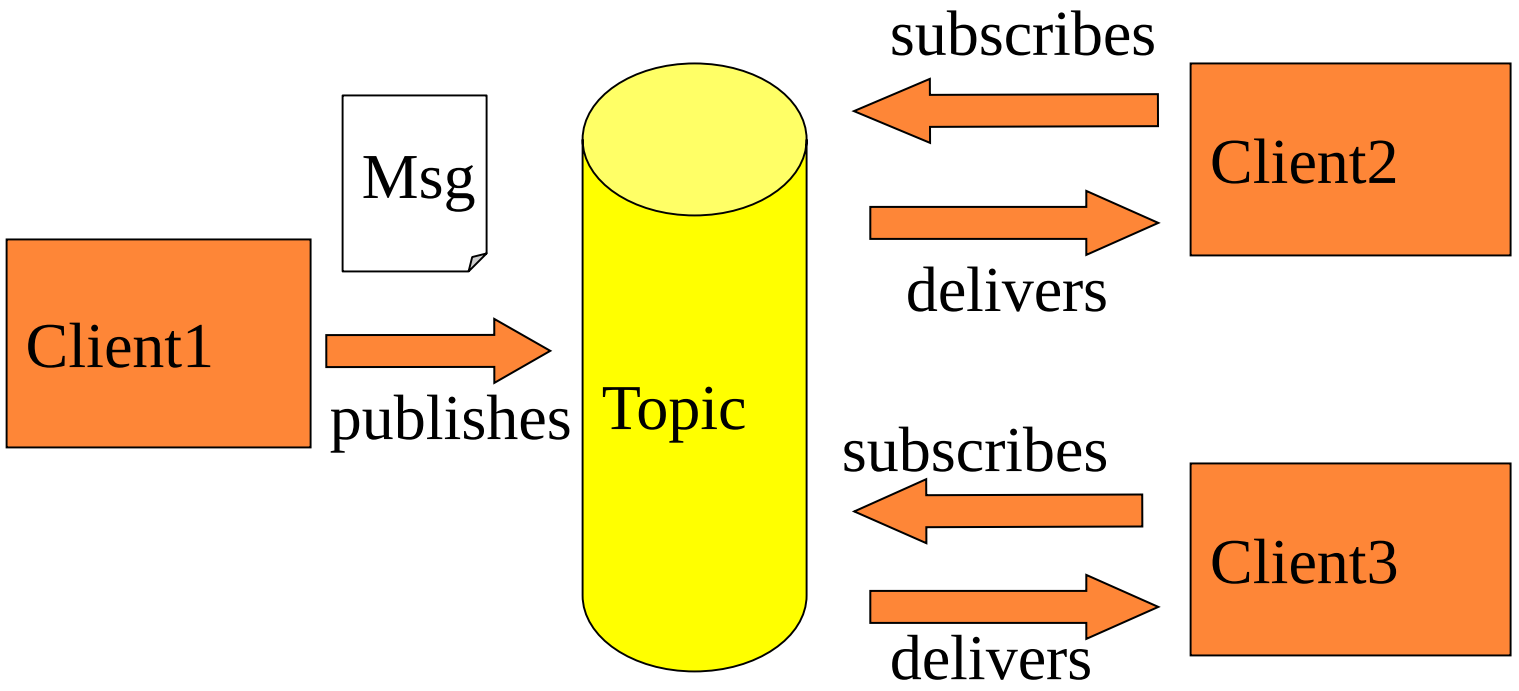
JMS Messaging Domains

- Point-to-Point (PTP)
  - Implémentation du MQ;
  - Chaque message est envoyé à un demandeur.
- Publish-Subscribe systems
  - Utilise le concepte du subject-based;
  - Chaque message est envoyé à pluisieurs consommateur.

Point-to-Point Messaging



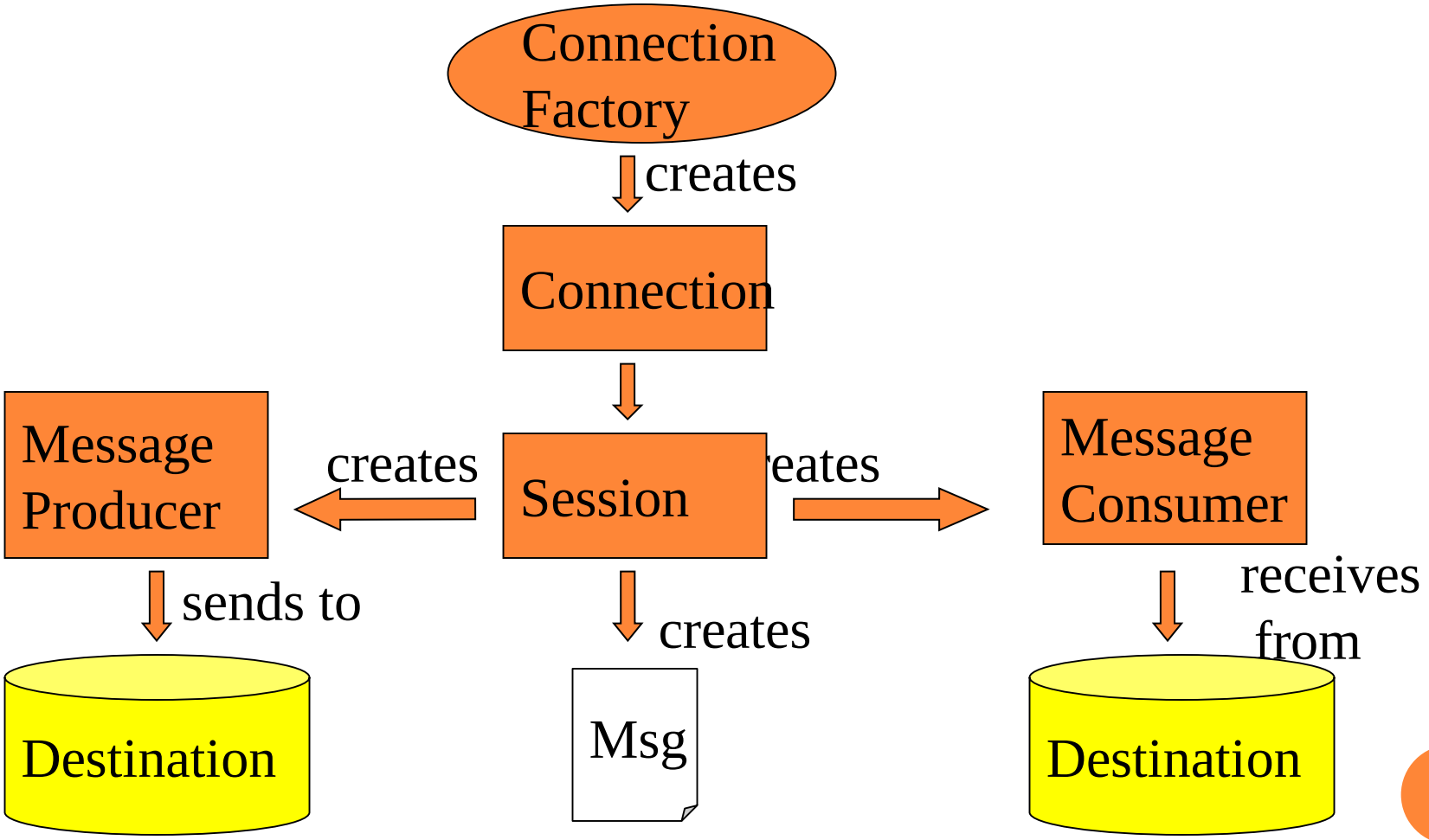
Publish/Subscribe Messaging



Consommation de message

- Synchrone
  - Un abonné ou un récepteur extrait explicitement le message de la destination en appelant la méthode de réception.
  - La méthode de réception peut bloquer jusqu'à ce qu'un message arrive ou peut expirer si un message n'arrive pas dans un délai spécifié.
- Asynchrone
  - Un client peut enregistrer un écouteur de message auprès d'un consommateur.
  - Chaque fois qu'un message arrive à destination, le fournisseur JMS délivre le message en appelant la méthode onMessage () de l'auditeur.

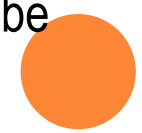
JMS API: Modele de Programmation



Exemple de Client

- Création d'une connnexion et une session

```
InitialContext jndiContext=new InitialContext();
//look up for the connection factory
ConnectionFactory cf=jndiContext.lookup(connectionfactoryname);
//create a connection
Connection connection=cf.createConnection();
//create a session
Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
//create a destination object
Destination dest1=(Queue) jndiContext.lookup("/jms/myQueue"); //for PointToPoint
Destination dest2=(Topic)jndiContext.lookup("/jms/myTopic"); //for publish-subscribe
```



Producer Exemple

- Création de connexion et de session
- Création de producer

```
MessageProducer producer=session.createProducer(dest1);
```

- Envoi de message

```
Message m=session.createTextMessage();
m.setText("just another message");
producer.send(m);
```

- Fermeture de connection

```
connection.close();
```

Consommateur

- Création de connexion et de session
- Création de consommateur

```
MessageConsumer
consumer=session.createConsumer(dest1);
```

- Reception de messages

```
connection.start();
Message m=consumer.receive();
```