



# Systemes distribués

Master 1 RID

# Plan

- **Partie I : Systèmes distribués**
  - Définition
  - Objectifs
  - Avantages
  - Inconvénients
  - Applications réparties
- **Partie II : Communication dans un système distribué**
  - Middleware (intergiciel)
  - Modèles de communication
    - Client/serveur
    - Diffusion de messages
    - Pair à pair (peer to peer)
  - Type de communication
    - Synchrones
    - Asynchrones

# Définition

- Système distribué en opposition à système centralisé (tout est localisé sur la même machine).
- **Définition 1** : ensemble d'ordinateurs indépendants qui apparaît à un utilisateur comme un système unique et cohérent.
- **Exemple Google** : le plus grand parc de serveurs au monde avec environ 1 million de serveurs répartis sur environ 36 sites.

# Définition

- **Définition 2** : ensemble d'entités logicielles communiquant entre-elles.
- Entités logicielles s'exécutent sur des machines reliées entre elles par un réseau.
- Exemple **Skype** : la téléphonie sur IP.

# Objectifs d'un système distribué

- **Transparence (masquer la répartition) :**
  - Localisation des ressources non perceptible.
    - URL [www.google.com](http://www.google.com).
  - Duplication de ressources non visible.
  - Concurrence d'accès aux ressources non perceptible : (ex. accès à un même fichier ou une même table dans une base de données, vidéo youtube, ...).
- **Sécurité :** confidentialité, intégrité, ..
- **Tolérance aux pannes :** permettant à un utilisateur de ne pas s'interrompre (ou même se rendre compte) à cause d'une panne d'une ressource.
- **Mise à l'échelle (scalability) :** fonctionne efficacement dans différentes échelles :
  - Deux postes de travail et un serveur de fichiers.
  - Réseau local avec plusieurs centaines de postes de travail et serveurs de fichiers.
  - Plusieurs réseaux locaux reliés pour former Internet.
- **Ouverture :**
  - **Extensibilité** (ajout/MAJ de composants sans en affecter les autres).
  - **Flexibilité** (facilité d'utilisation).

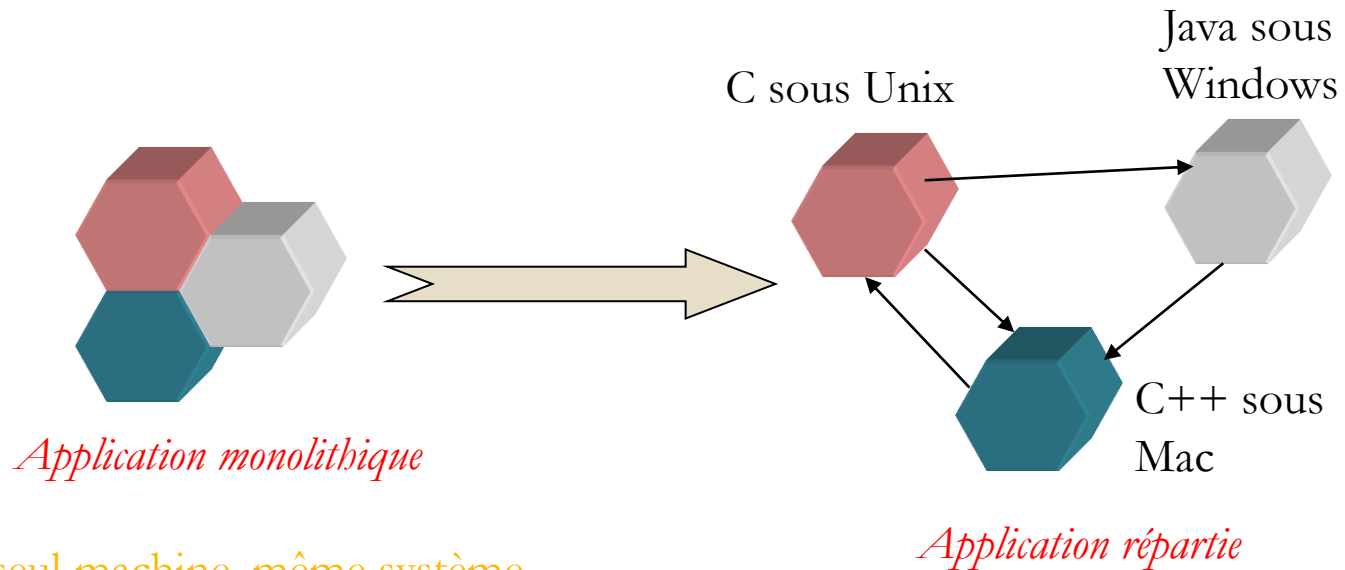
# Avantages des systèmes distribués

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

# Inconvénients / points faibles

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

# Qu'est ce qu'une application répartie ?




Une seule machine, même système  
et même langage de programmation

- Il s'agit d'une application découpée en plusieurs unités.
  - Chaque unité peut être placée sur **une machine différente.**
  - Chaque unité peut s'exécuter sur **un système différent.**
  - Chaque unité peut être programmée dans **un langage différent.**



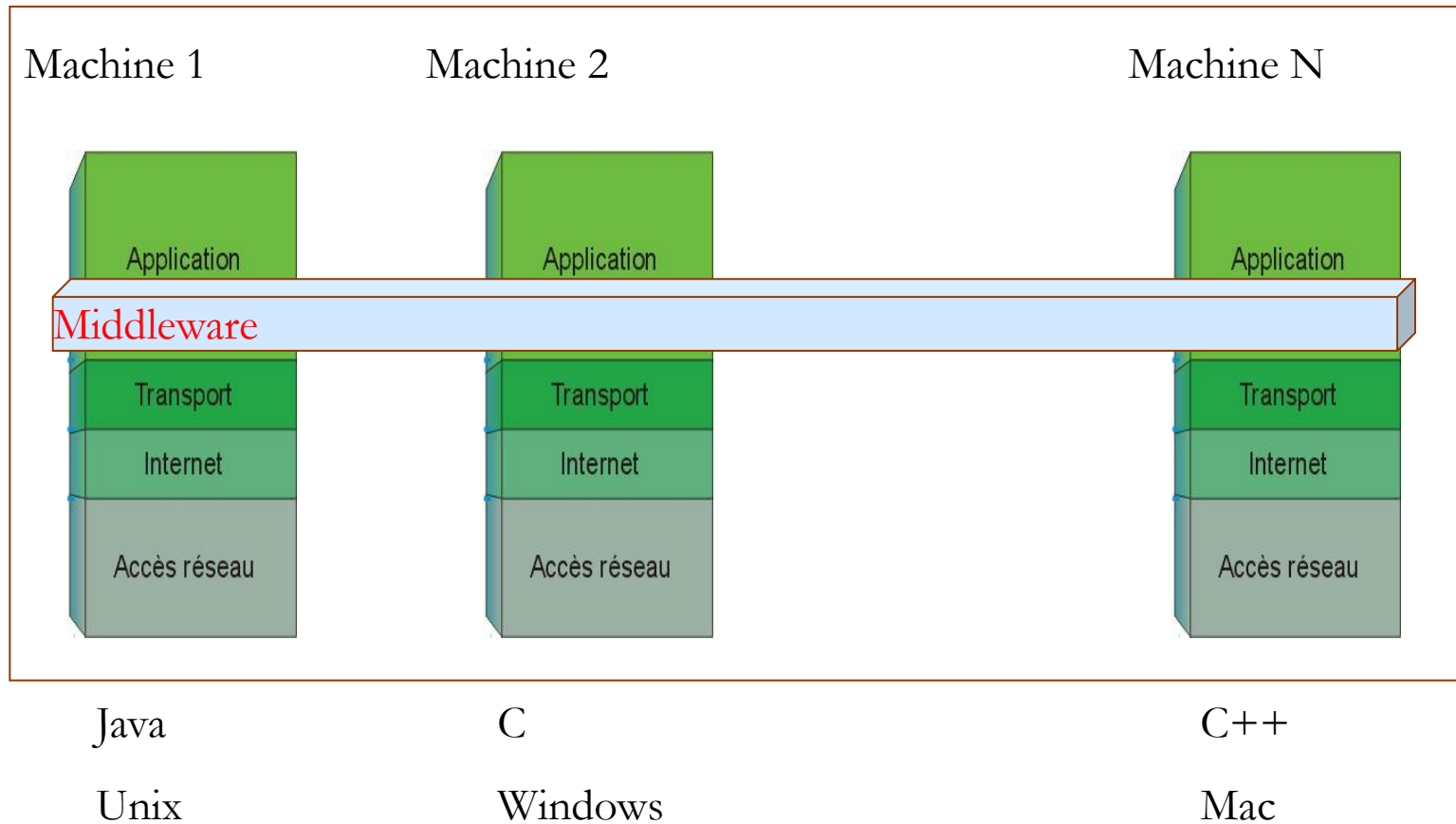
# Communication dans un système distribué

- **Objectif** : communiquer les éléments d'une application répartie.
  - **2 manières** :
    - **Bat niveau** : directement en appelant les services des couches TCP ou UDP.
      - *Exemple* : utilisation des sockets en Java.
    - **Haut niveau** : définition de couches offrant des services plus complexes.
      - Couche réalisée en s'appuyant sur les couches TCP/UDP.
      - *Exemple* : appel d'une méthode chez une entité distante.
- 
- **Notion de middleware (intergiciel).**

# Middleware (intergiciel)

- Le terme middleware vient de l'anglais middle (du milieu) et software (logiciel).
- Le middleware est une couche intermédiaire (couche logiciel) qui s'intercale entre l'infrastructure de communication d'un réseau et les éléments de l'application distribuée.

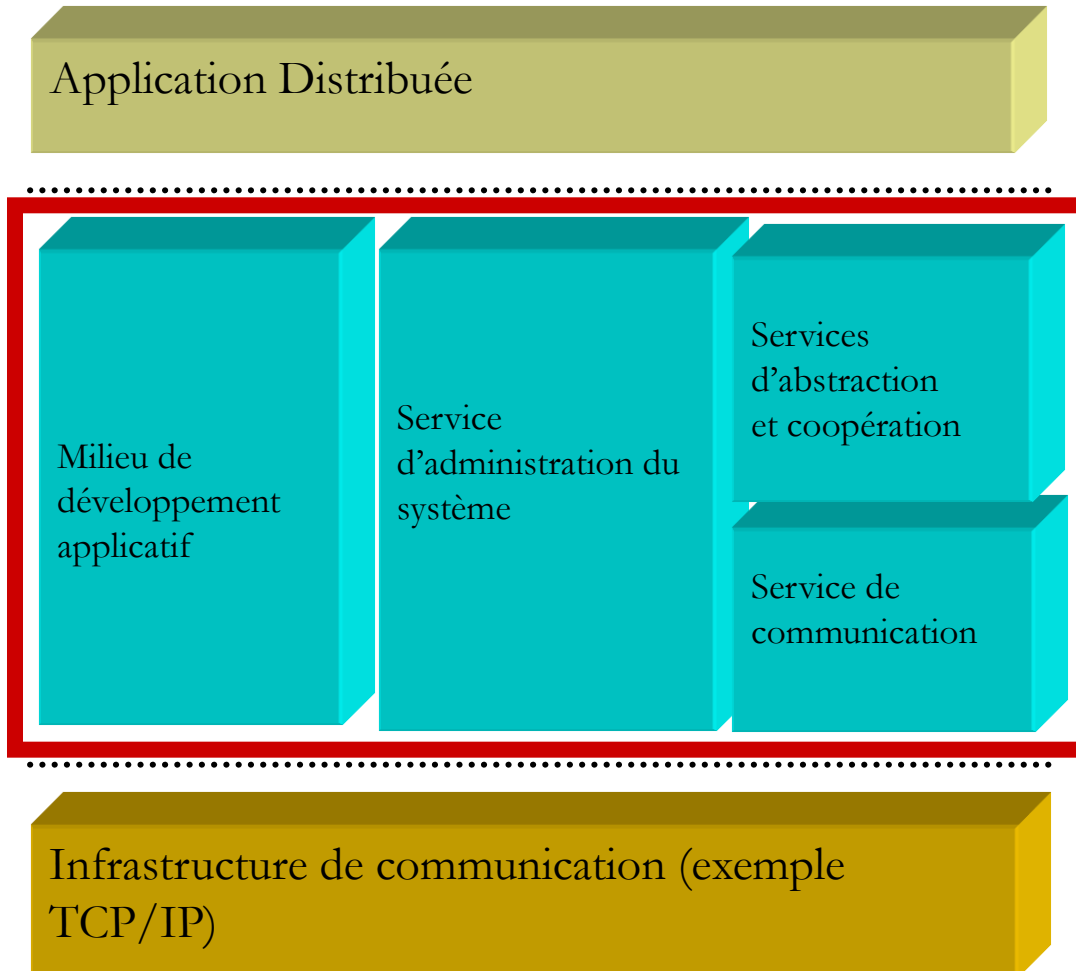
# Middleware (intergiciel)



# Middleware

- **Définition** : le middleware est un logiciel qui se compose d'un ensemble de services qui permettent à plusieurs entités (processus, objets etc.) résidents sur plusieurs ordinateurs, d'interagir à travers un réseau en dépit des différences dans les protocoles de communication, architectures des systèmes locaux, etc..
- Résoudre l'Hétérogénéité : Etre **indépendant** des **systèmes d'exploitation** et du **langage de programmation** des applications.
- Résoudre l'Interopérabilité : Unifier **l'accès** à des machines **distantes**.

# Middleware



middleware

# Middleware : services

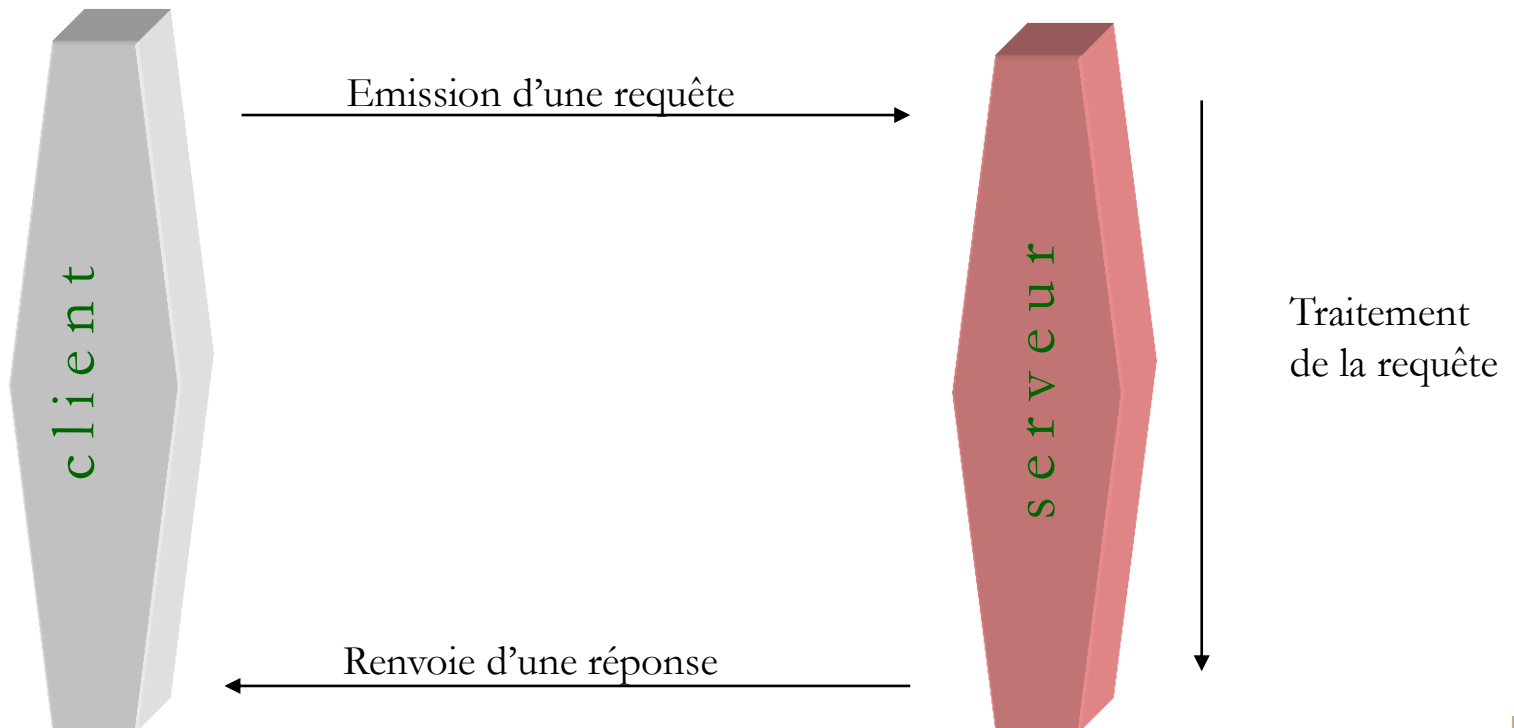
- **Service de communication** : ce service offre une interface qui permet à l'application distribuée d'échanger les informations entre ses composantes résidentes sur des ordinateurs avec des caractéristiques matériels et/ou logiciels différentes.
- **Services d'abstraction et coopération** : ces services représentent le coeur du middleware et ils comprennent entre autres :
  - **Directory Service** ou service de "pages blanches", qu'il pourvoit à l'identification et à la localisation d'entités distribuées.
  - **Security Service** pour la protection des éléments critiques comme les données et les services applicatifs à travers des techniques comme l'authentification et la cryptographie.

# Middleware : services

- **Services d'administration du système** : pour gérer la configuration.
- **Milieu de développement applicatif** : contient les instruments d'aide à l'écriture du code par exemple le langage IDL.
  - **IDL (Interface Definition Language)** : est un langage de définition et non un langage de programmation grâce auquel on définit des interfaces et des structures de données et non des algorithmes.
  - On peut donc, par exemple, écrire des applications clientes en JAVA et des applications serveurs en C++ et faire communiquer ce client et ce serveur.
  - IDL est défini par l'OMG (Object Management Group) et utilisé notamment dans le cadre d'applications CORBA.

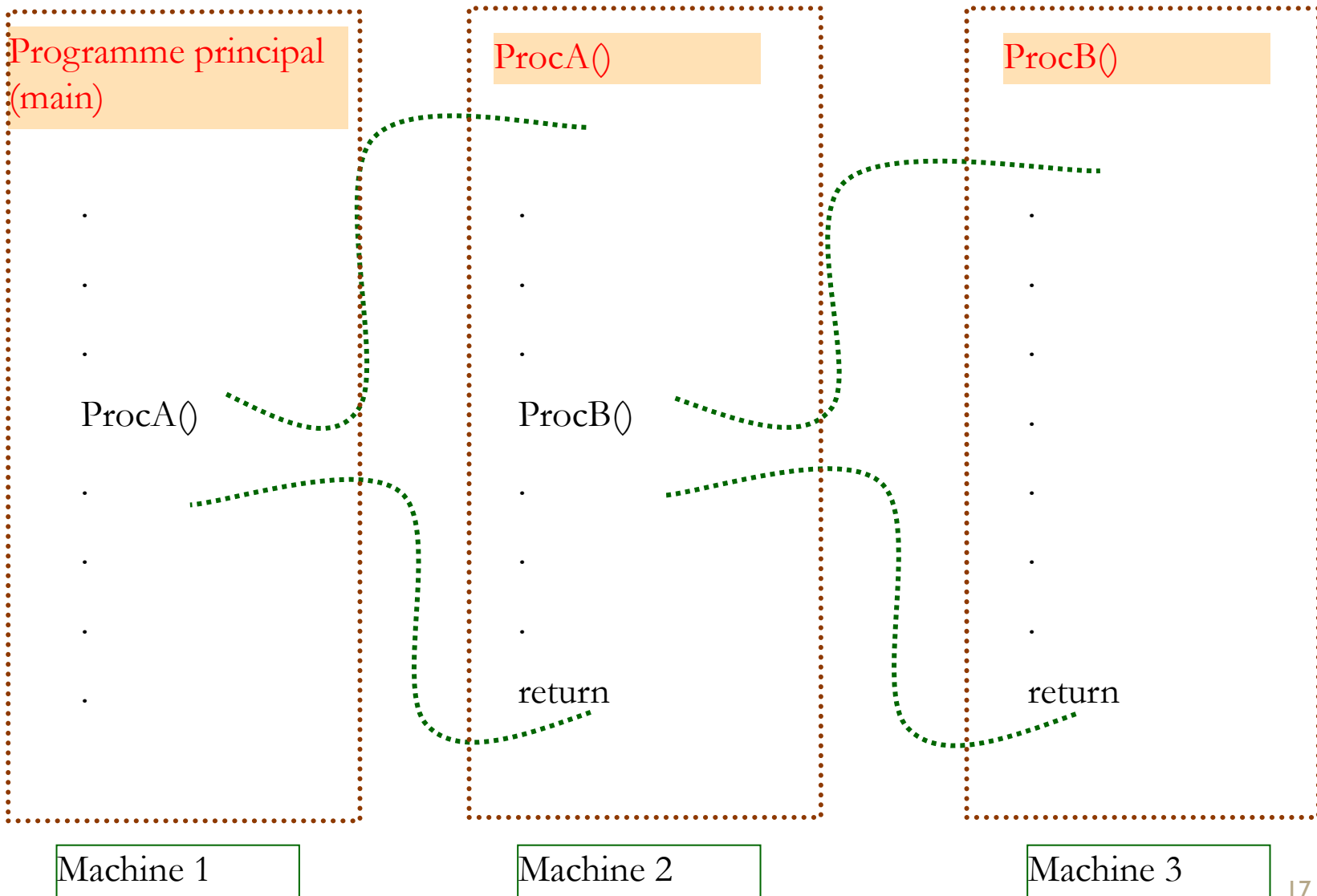
# Middleware : mécanisme de base

- Les environnements répartis sont basés (pour la plupart) sur **un mécanisme RPC (Remote Procedure Call)** (Appel de procédure à distance).
- Ce mécanisme fonctionne en mode **requête / réponse**.
  - Le client effectue une requête (demande un service ),
  - Le serveur traite la demande puis retourne une réponse au client.





# RPC : principe



# Exemples de middleware

- **RPC (Remote Procedure Call).**
  - Pas d'objets.
- **JAVA RMI (Remote Method invocation).**
  - Solution purement java.
- **CORBA (Common Object Request Broker Architecture).**
  - Langues de programmation distincts.

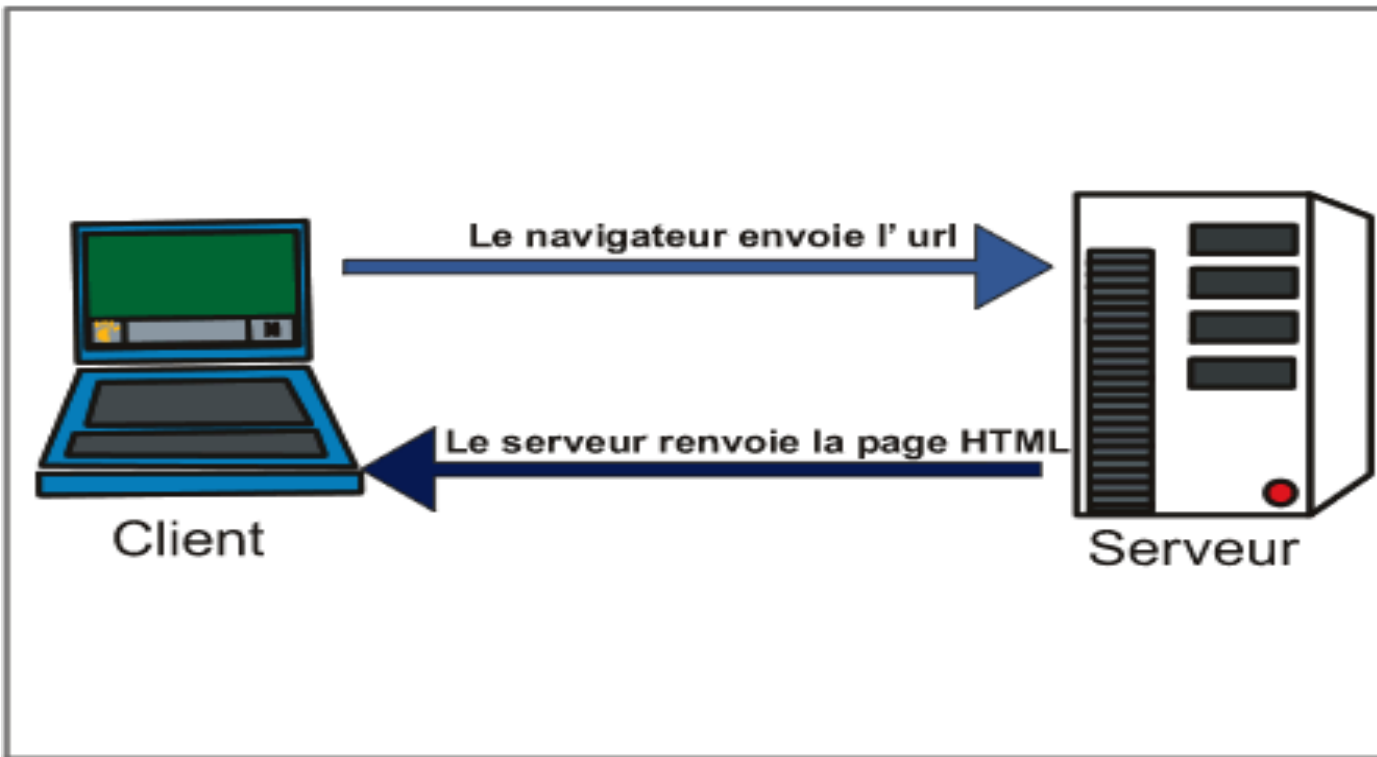
# Modèles de communication

- Les éléments distribués interagissent, communiquent entre eux selon plusieurs modèles possibles :
  - **Client/Serveur.**
  - **Diffusion de messages.**
  - **Pair à pair (peer to peer).**
  - Modèle à base de composants.
    - EJB (Enterprise JavaBeans).
  - Modèle à base d'agents mobiles.
  - Modèles à mémoires « virtuelles » partagées.

# Client/Serveur

- **Modèle le plus répandu.**
- **2 rôles distincts :**
  - **Client :** demande que des requêtes ou des services lui soient rendus.
  - **Serveur :** répond aux requêtes des clients.
    - Liens forts entre le client et le serveur.
- **Interaction :**
  - Message du client vers le serveur pour faire une requête.
  - Exécution d'un traitement par le serveur pour répondre à la requête.
  - Message du serveur vers le client avec le résultat de la requête.
- **Interaction de type « 1 vers 1 »**

# Client/Serveur



Client/Serveur WEB

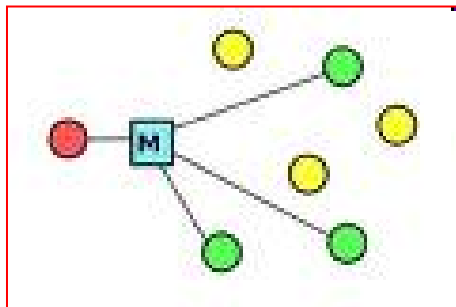
# Types de Client/Serveur

- **Client/Serveur traditionnel :**
  - **RPC (Remote procedure call).**
- **Client/Serveur à objets :**
  - **RMI (Remote Method Invocation) :** communique des objets java.
  - **CORBA :** n'importe quel langage.
  - ...
- **Client/Serveur de données :**
  - Requêtes SQL pour communiquer avec une base de donnée MySQL par exemple.
- **Client/Serveur WEB :**
  - **Pages web statiques avec HTML.**
  - **Pages web dynamiques avec PHP.**
  - **Servlets (\*.java)** sont des classes Java exécutées par le serveur en réponse à une requête du client (en utilisant le protocole http).
  - **JSP (JavaServer Pages) (\*.jsp)** représentant un code HTML dans lequel du code Java est appelé.

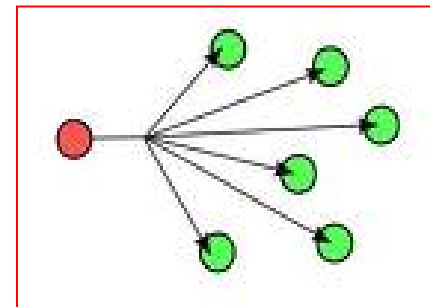
RPC, RMI, CORBA, JSP.. : Modèles d'exécution pour le client / serveur

# Diffusion de messages (broadcast)

- 2 rôles distincts :
  - **Emetteur** : envoie des messages (ou événements) à destination de tous les récepteurs.
    - Possibilité de préciser un sous-ensemble de récepteurs (multicast).
  - **Récepteurs** : reçoivent les messages envoyés.
- Interaction :
  - Emetteur envoie un message.
  - Le middleware s'occupe de transmettre ce message à chaque récepteur.
- **Modèle d'exécution : Message Oriented Middleware (MOM).**
- **Package javax.jms** permet d'implémenter une architecture de type MOM.
  - JMS : Java Message Service.



Multicast



Broadcast

# Diffusion de messages (broadcast)

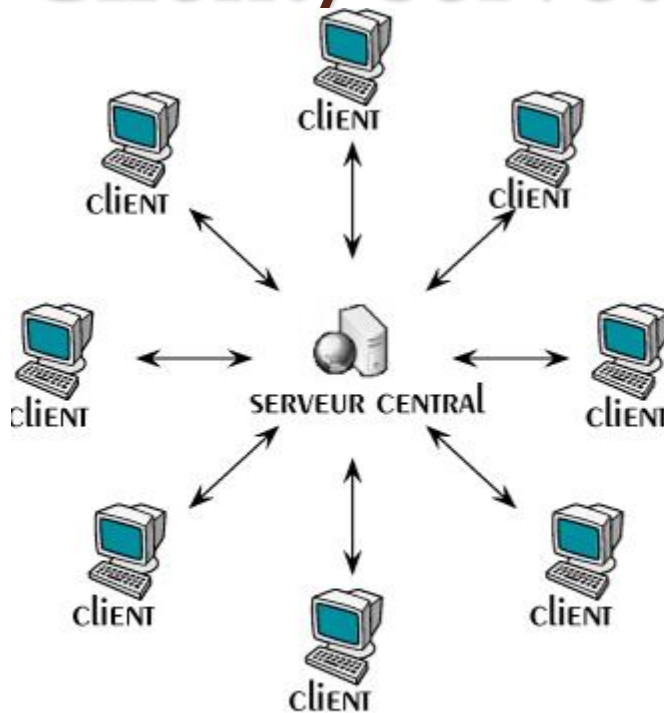
- 2 modes de réception :
  - Le récepteur va vérifier lui-même qu'il a reçu un message.
    - **Boîte aux lettres.**
  - Le récepteur est prévenu que le message est disponible et il lui est transmis.
    - **Le facteur sonne à la porte pour remettre en main propre le courrier.**
- Particularités du modèle :
  - Dépendance plus faible entre les participants.
    - Pas besoin pour l'émetteur d'être directement connecté aux récepteurs ni même de savoir combien ils sont.
  - Interaction de type « 1 vers N ».



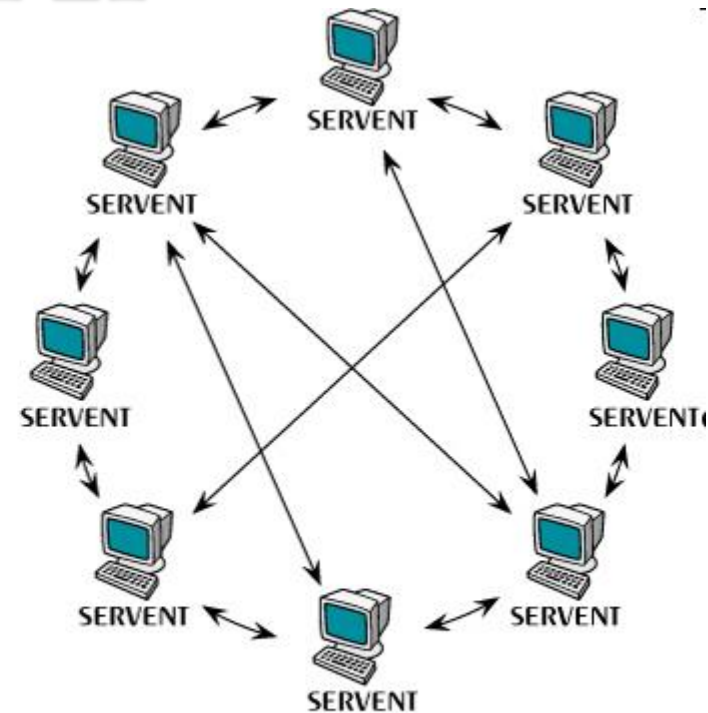
# Modèle pair à pair (peer to peer / P2P)

- **Un seul rôle** : pas de distinction entre les participants.
  - Chaque participant est connecté avec tous les participants d'un groupe et tout le monde effectue les mêmes types d'actions pour partager des données.
- **Exemples** :
  - Modèles d'échanges de fichiers (**e-mule, bit-torrent**).
  - Avec parfois un mode hybride client/serveur – P2P.
  - Serveur sert à connaître la liste des fichiers et effectuer des recherches.
  - Le mode P2P est utilisé ensuite pour les transferts.
  - Chacun envoie une partie du fichier à d'autres participants.
  - **Skype** emploie une technique **P2P VoIP** (mélange de P2P et de VoIP (voix sur IP)) pour se connecter avec les autres utilisateurs de Skype.

# Client/serveur vs P2P



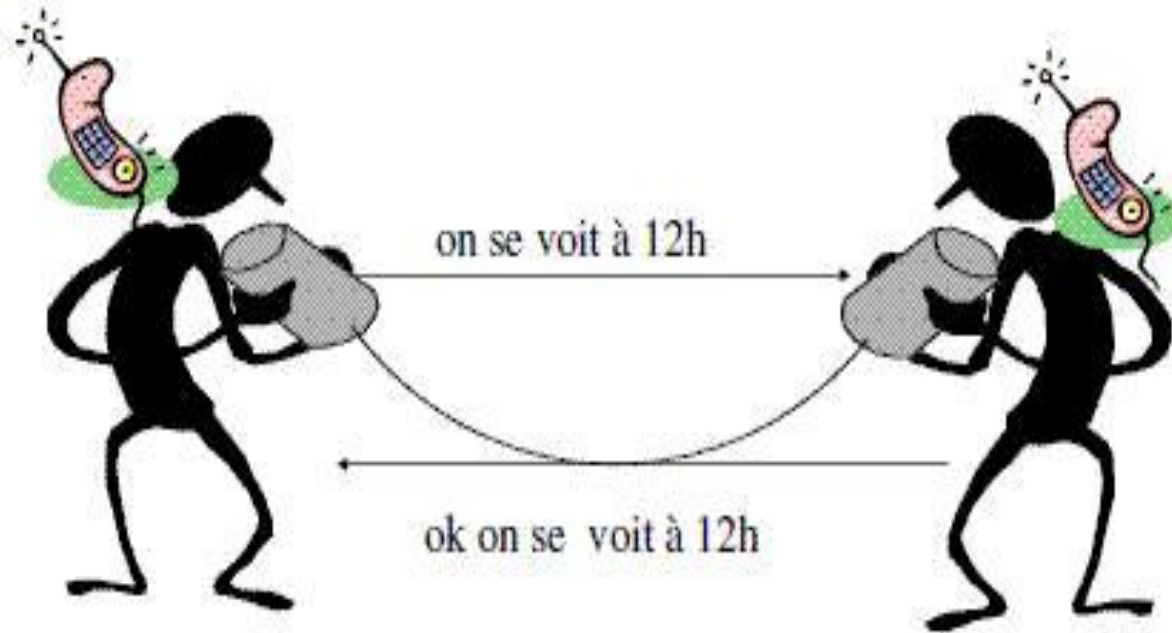
ARCHITECTURE CLIENT-SERVEUR



ARCHITECTURE PAIR-À-PAIR

- **Servent** est la contraction du mot **serveur** et **client**. Ce terme est souvent utilisé pour désigner les logiciels P2P.

# Type de communication (synchrone/asynchrone)



- **Communication synchrone** : même notion de temps, transmission instantanée, généralement bornée.

# Type de communication (synchrone/asynchrone)

- Communication asynchrone

