

# Systèmes répartis et client-serveur

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

SRCS – Master 1 SAR 2017/2018

# Systèmes répartis



## Qu'est-ce ?

Un système composé de :

- Plusieurs unités de calcul distantes et autonomes
  - fiables/non fiables
  - homogènes/hétérogènes
- Un réseau physique les reliant
  - fiable/non fiable
  - communications synchrones/asynchrones

## Pour faire quoi ?

- Stockage de données géo-répliquées
- Calcul parallèle
- Partage de ressources
- Répartition de charge
- Tolérance aux pannes

⇒ **Passage à l'échelle**

## Ce qui est transparent pour l'utilisateur :

- Localisation des ressources
- Accès aux ressources
- Hétérogénéité des ressources
- Pannes des ressources
- Extensibilité des ressources

## Problèmes fondamentaux

- Diffusion fiable : "Diffuser un message à un ensemble de processus"
- Checkpointing : "Sauvegarder et retrouver un état stable du système"
- Réplication de donnée : "Gérer la cohérence des réplicas"
- Consensus : "Se mettre d'accord sur une valeur"
- Exclusion mutuelle : "Gérer les accès à une section critique "

# Architectures des Systèmes répartis

## Définition

Une architecture de système réparti définit :

- la structures de ses entités
- et leurs liens relationnels

Les 4 questions à se poser :

- "*Quelles sont mes entités communicantes ?*"
- "*Où sont placées mes entités sur mon infrastructure physique ?*"
- "*Quel paradigme de communication utiliser ?*"
- "*Quelles sont les rôles et les responsabilités de mes entités ?*"

# Entités communicantes

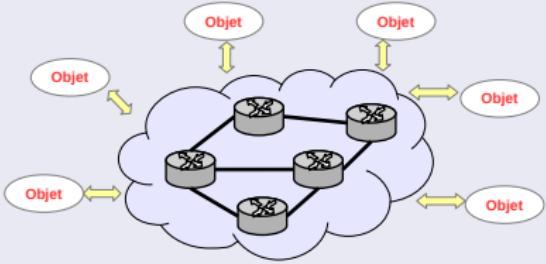
D'un point de vue "*Système*"

- Processus
- Threads
- Nœuds physiques



D'un point de vue "*Programmation*"

- Objets répartis
- Composants
- Web services



# Paradigmes de communication

## IPC : Inter-Process Communication

- Support bas niveau pour la communication entre processus
- Exemples : socket, primitives de passage de message
- Réglages fin, performances / lourd à programmer

## Communication indirecte

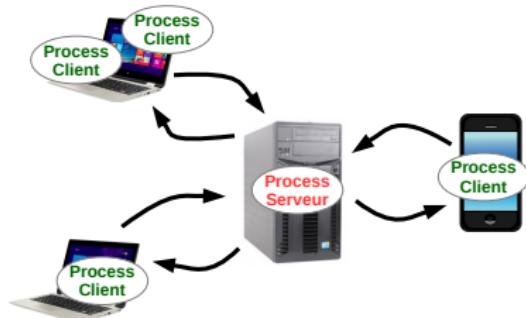
- La communication se fait par le biais d'une entité tierce
- Exemples : publish/subscribe, DSM, file de messages
- Découplage spatial et temporel / communications synchrones difficiles

## Invocation distante

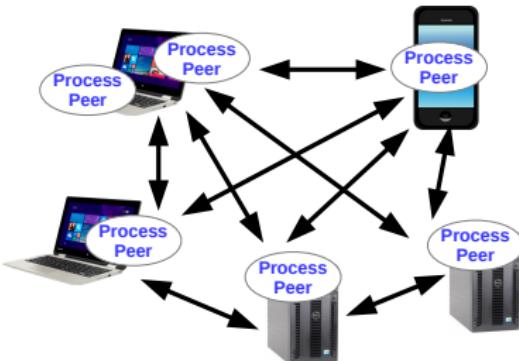
- L'échange se fait par l'appel d'une méthode/procédure distante
- Exemples : RPC, RMI, Corba, EJB
- Modèle simple, transparence des couches réseaux/optimisations difficiles

# Rôles, responsabilités et relations

## Architecture Client/Serveur



## Architecture Pair à pair (P2P)



2 groupes de processus :



- **Serveurs** : fournissent un service spécifique à leurs clients
- **Clients** : requêtent le service

Les processus collaborent :

- en exécutant le même programme
- en fournissant la même interface
- sans distinction client/serveur



# Rôles client/serveur

## Le client

- Le processus qui envoie la requête
- Si la requête est synchrone, attendre la réponse du serveur

## Le serveur

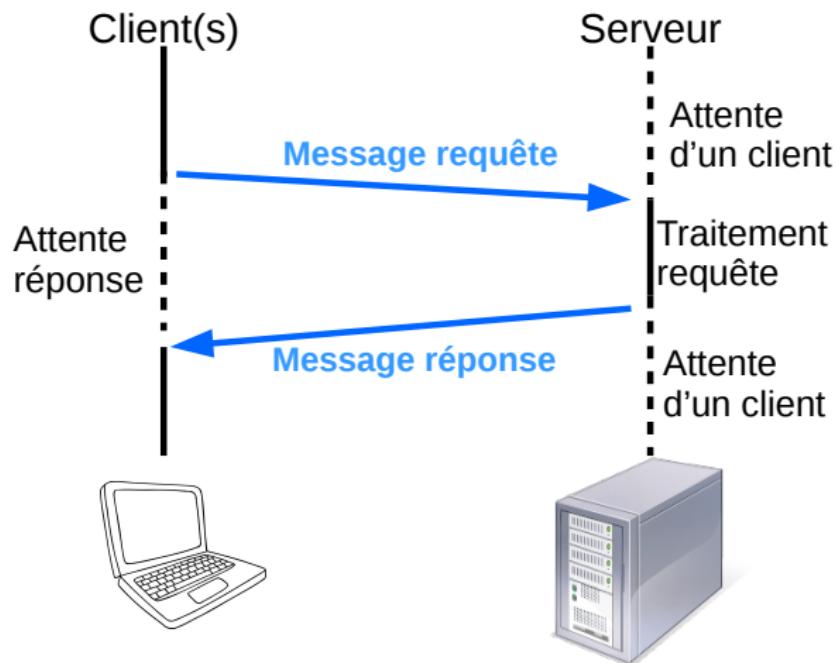
- Enregistre son service auprès du système
- Se met en attente de la réception d'une requête
- Traite la requête à la réception
- Renvoie une réponse au client

## Attention

- Les rôles client-serveur ne se font que par rapport à une requête.
- Le serveur peut être client d'un autre service.

*Serveur Web client d'un serveur DNS ou d'un SGBD*

# Interaction synchrone client/serveur



# Avantage-Inconvénient du modèle client/serveur

## Avantages

- ✓ Structure simple :
  - Séparation spécification service et de son implémentation
  - Client et serveur peuvent être modifiés indépendamment
- ✓ Partage des ressources du serveurs entre plusieurs clients
- ✓ Centralisation des traitements
  - ⇒ Simplification des contrôles de sécurités, de l'administration et des mises à jour

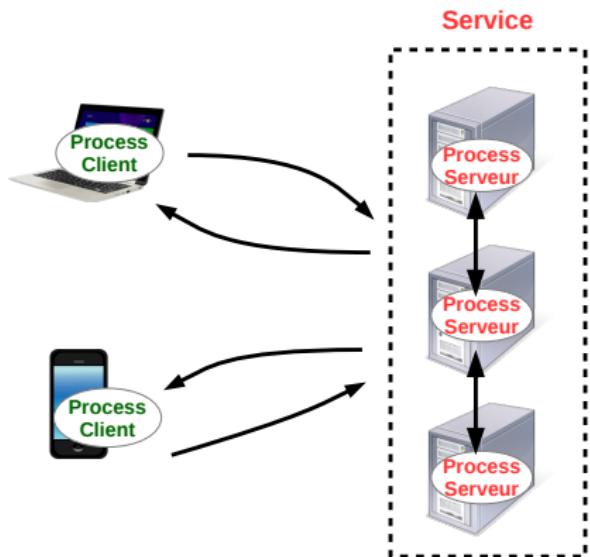
## Inconvénients

Le serveur est un goulot d'étranglement

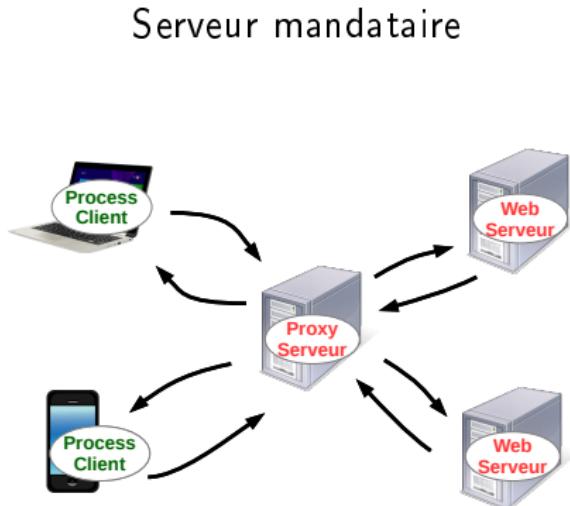
- ✗ Trop de clients ⇒ Surcharge du serveur ⇒ Service indisponible
- ✗ Si le serveur tombe, le système ne fonctionne plus
- ✗ Pas de communication directe entre les clients

# Exemple d'architecture client/serveur

Service répliqué



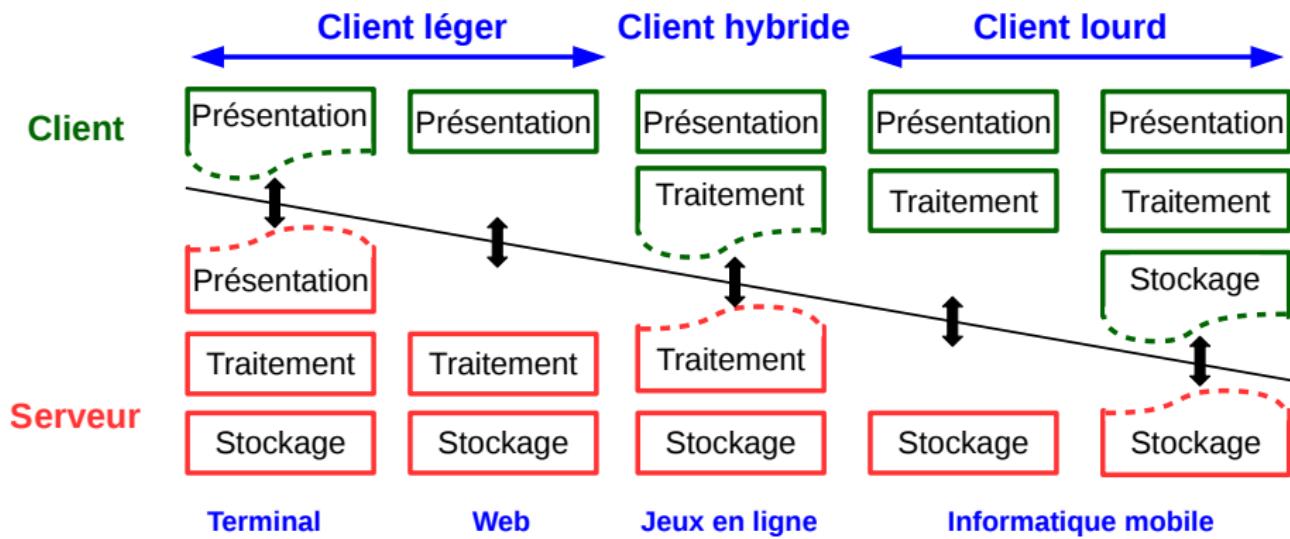
Serveur mandataire



# Application client/serveur : qui doit faire quoi ?

## Les trois niveaux de spécification d'un service

- Présentation des données (IHM, affichage, ...)
- Traitement des données (ce que fait l'Application)
- Stockage des données (persistance, indexation, cohérence)



# Client léger vs. client lourd

## Client léger

### Avantages

- Adaptable à plusieurs service
- Administration simple
- Peu de ressources coté client

### Inconvénients

- Charge accrue sur serveur et réseau
- Client doit toujours être connecté

## Client lourd

### Avantages

- Meilleures performances coté client
- Service disponible hors ligne
- Moins de congestion serveur

### Inconvénients

- Administration et contrôle plus difficiles
- L'application est dépendante de la plate-forme cliente