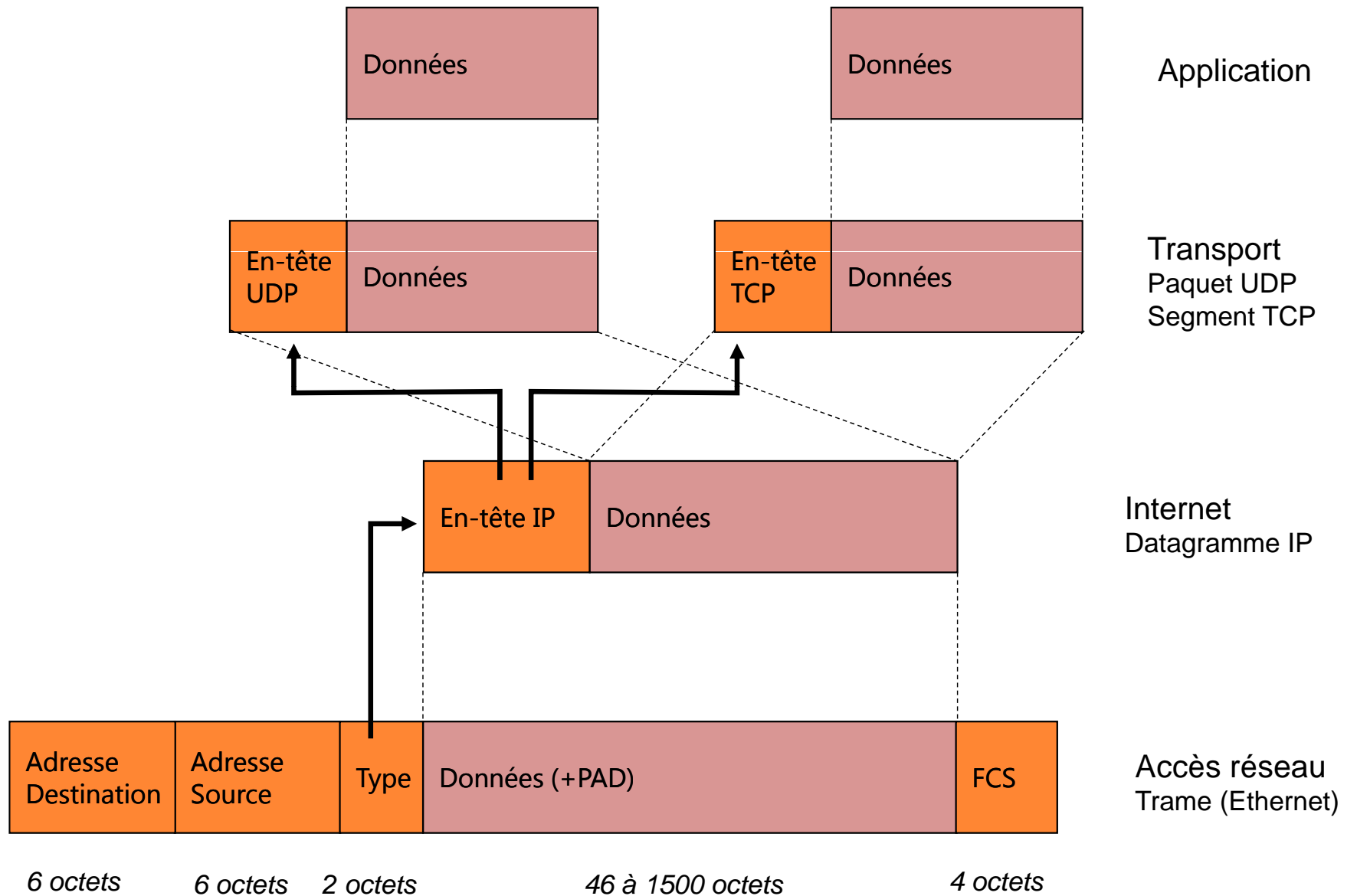


Le modèle client-serveur

Introduction

Encapsulation : rappel



Les applications réseau

- Les applications sont la raison d'être des réseaux informatiques
 - Partie visible pour l'utilisateur
 - Développement considérable de leur nombre depuis les débuts d'Internet
 - Ont évolué au fil des années
 - Sous forme textuelle au tout début : messagerie électronique, accès aux terminaux distants, transfert de fichiers ...
 - Multimédia aujourd'hui : diffusion radio/vidéo à la demande (podcasting), visioconférence, téléphonie sur IP (VoIP) ...
 - Sont basées sur la communication entre 2 entités
 - L'une appelée « **client** »
 - L'autre « **serveur** »

Principe du client / serveur

- Repose sur une communication d'égal à égal entre les applications ; communication réalisée par dialogue entre processus deux à deux
 - un processus client
 - un processus serveur
- Les processus ne sont pas identiques mais forment plutôt un système coopératif se traduisant par un échange de données
 - le client réceptionne les résultats finaux délivrés par le serveur
- Le client initie l'échange
- Le serveur est à l'écoute d'une requête cliente éventuelle
- Le service rendu = traitement effectué par le serveur

Serveur

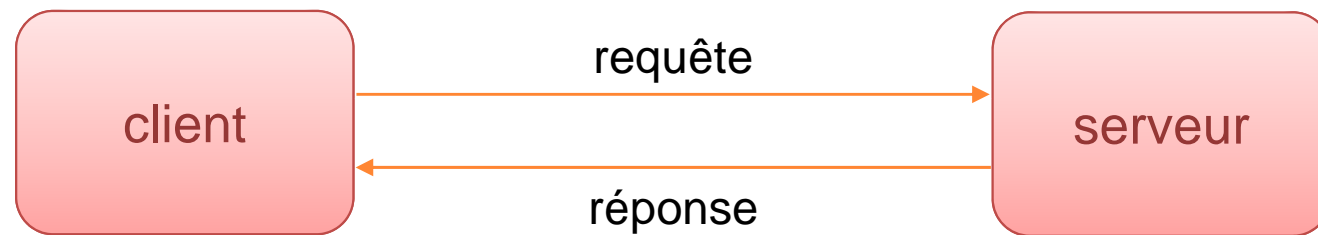
- Un programme serveur
 - tourne en permanence, attendant des requêtes
 - peut répondre à plusieurs clients en même temps
- Nécessite
 - une machine robuste et rapide, qui fonctionne 24h/24
 - alimentation redondante, technologie RAID ...
 - la présence d'administrateurs systèmes et réseau pour gérer les serveurs

Exemples de serveurs

- Serveur de fichiers (NFS, SMB)
- Serveur d'impression (lpd, CUPS)
- Serveur de calcul
- Serveur d'applications
- Serveur de bases de données
- Serveur de temps
- Serveur de noms (annuaire des services)

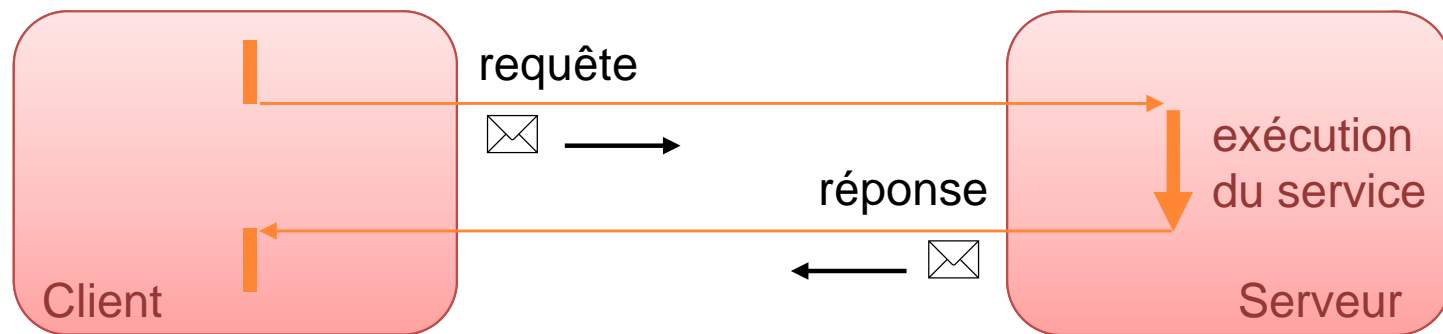
Le modèle client / serveur

- Le *client* demande l'exécution d'un service
- Le *serveur* réalise le service
- Client et serveur sont généralement localisés sur deux machines distinctes



Le modèle client / serveur

- Communication par **messages**
 - Requête : paramètres d'appel, spécification du service requis
 - Réponse : résultats, indicateur éventuel d'exécution ou d'erreur
 - Communication synchrone (dans le modèle de base) : le client est bloqué en attente de la réponse



Gestion des processus

- Client et serveur exécutent des processus distincts
 - Le client est suspendu lors de l'exécution de la requête (appel synchrone)
 - Plusieurs requêtes peuvent être traitées par le serveur
 - ➔ mode itératif
 - ➔ mode concurrent

Gestion des processus

- Mode de gestion des requêtes

→ itératif

- le processus serveur traite les requêtes les unes après les autres

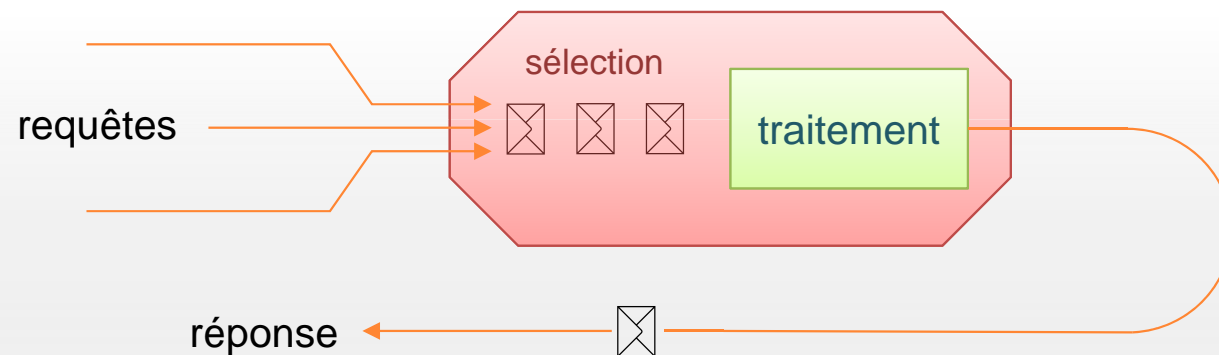
→ concurrent basé sur

- parallélisme réel
 - système multiprocesseurs par exemple
- pseudo-parallélisme
 - schéma veilleur-exécutants
- la concurrence peut prendre plusieurs formes
 - plusieurs processus (un espace mémoire par processus)
 - plusieurs processus légers (*threads*) dans le même espace mémoire

Gestion des processus dans le serveur

- Processus serveur unique

```
while (true) {  
  receive (client_id, message)  
  extract (message, service_id, params)  
  do_service[service_id] (params, results)  
  send (client_id, results)  
}
```



Gestion des processus dans le serveur

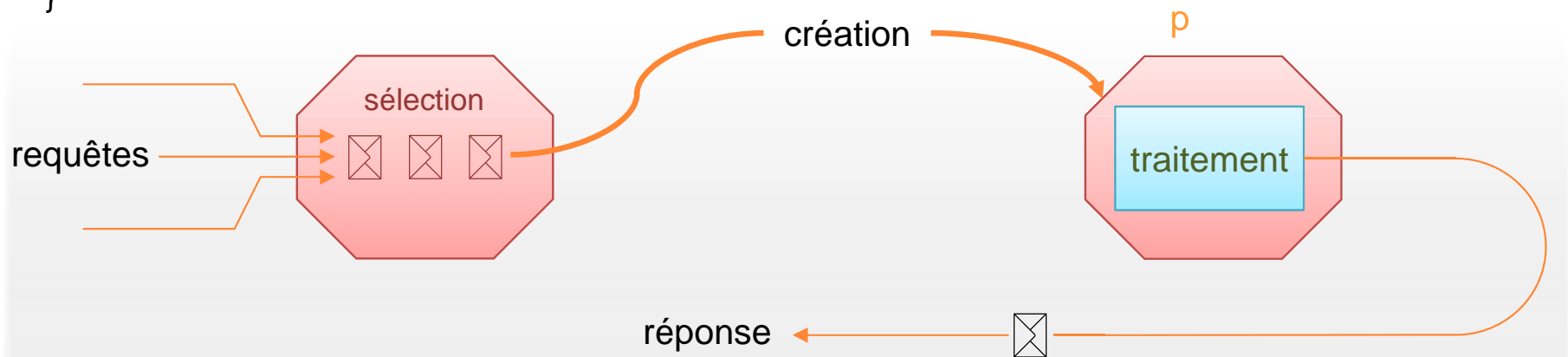
- Schéma veilleur-exécutants

Processus veilleur

```
while (true) {  
  receive (client_id, message)  
  extract (message, service_id, params)  
  p = create_thread (client_id, service_id,  
                    params)  
}
```

Création dynamique
des exécuteurs

```
programme de p  
do_service[service_id] (params, results)  
send (client_id, results)  
exit
```



Mise en œuvre du modèle client / serveur

- Besoin d'un support pour transporter les informations entre le client et le serveur
 - Bas niveau
 - Utilisation directe du transport : *sockets* (construits sur TCP ou UDP)
 - Haut niveau
 - Intégration dans le langage de programmation : RPC ou *Remote Procedure Call* (construits sur sockets)
- Nécessité d'établir un **protocole** entre le client et le serveur pour qu'ils se comprennent

Exemple de client / serveur

- Un serveur de fichiers
- Des clients qui demandent des fichiers
- Comment gérer la concurrence ?
 - un processus serveur \Leftrightarrow un client servi
 - plusieurs clients \Rightarrow plusieurs processus serveur
- Quel protocole utiliser ?
 - le client envoie le nom du fichier
 - le serveur renvoie la taille puis les données
 - comment gère-t-on les erreurs ?

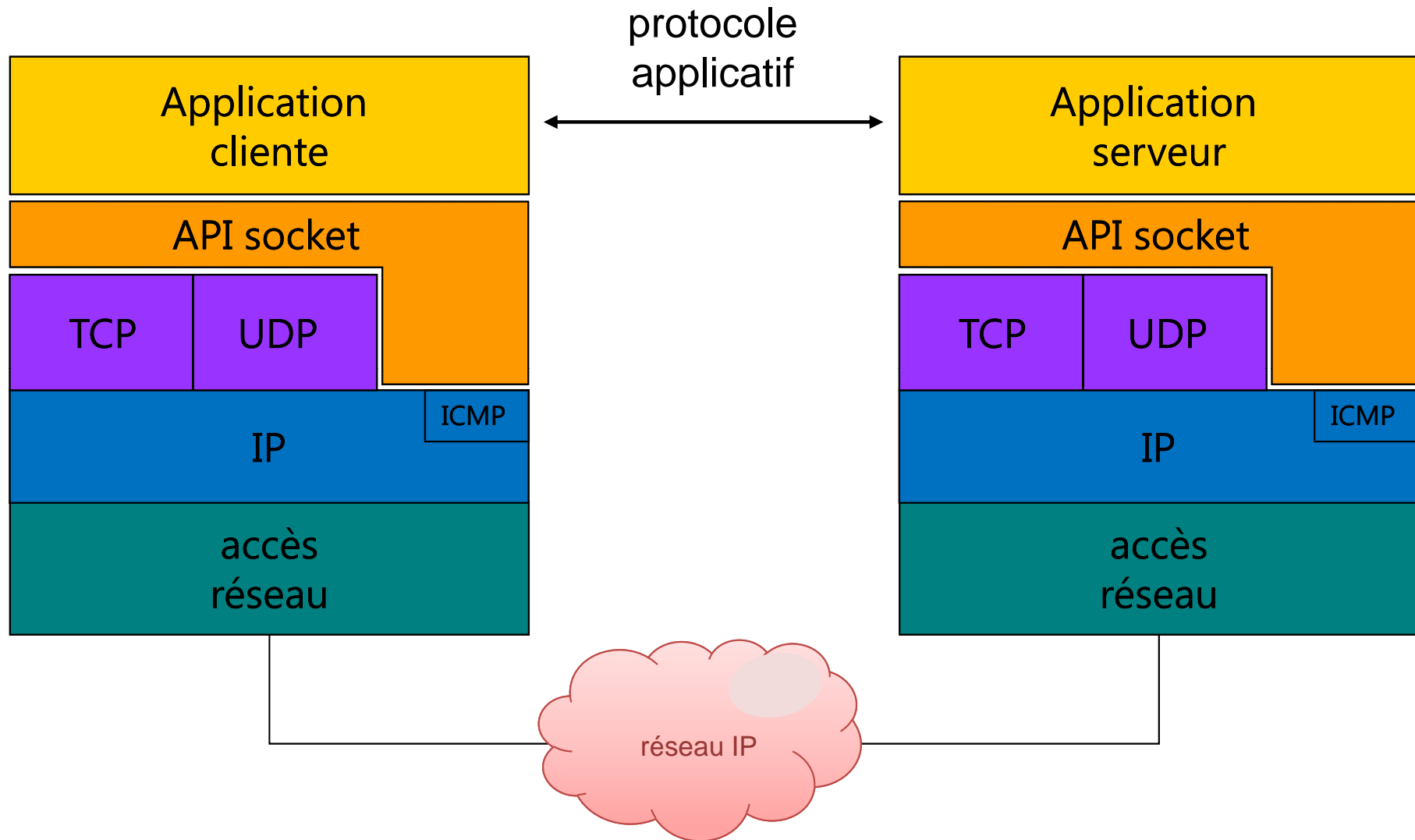
Les protocoles applicatifs

- Le protocole applicatif définit
 - Le format des messages échangés entre émetteur et récepteur (textuel, binaire, ...)
 - Les types de messages : requête / réponse / informationnel ...
 - L'ordre d'envoi des messages
- Ne pas confondre protocole et application
 - Une application peut supporter plusieurs protocoles (ex : logiciel de messagerie supportant POP, IMAP et SMTP)
 - Navigateur et serveur Web s'échangent des documents HTML en utilisant le protocole HTTP

Les sockets

- *API (Application Program Interface) socket*
 - mécanisme d'interface de programmation
 - permet aux programmes d'échanger des données
 - les applications client/serveur ne voient les couches de communication qu'à travers l'API socket (abstraction)
 - n'implique pas forcément une communication par le réseau
 - le terme « socket » signifie douille, prise électrique femelle, bref ce sur quoi on branche quelque chose
- L'API socket s'approche de l'API fichier d'Unix
- Développée à l'origine dans Unix BSD (*Berkeley Software Distribution*)

L'API socket



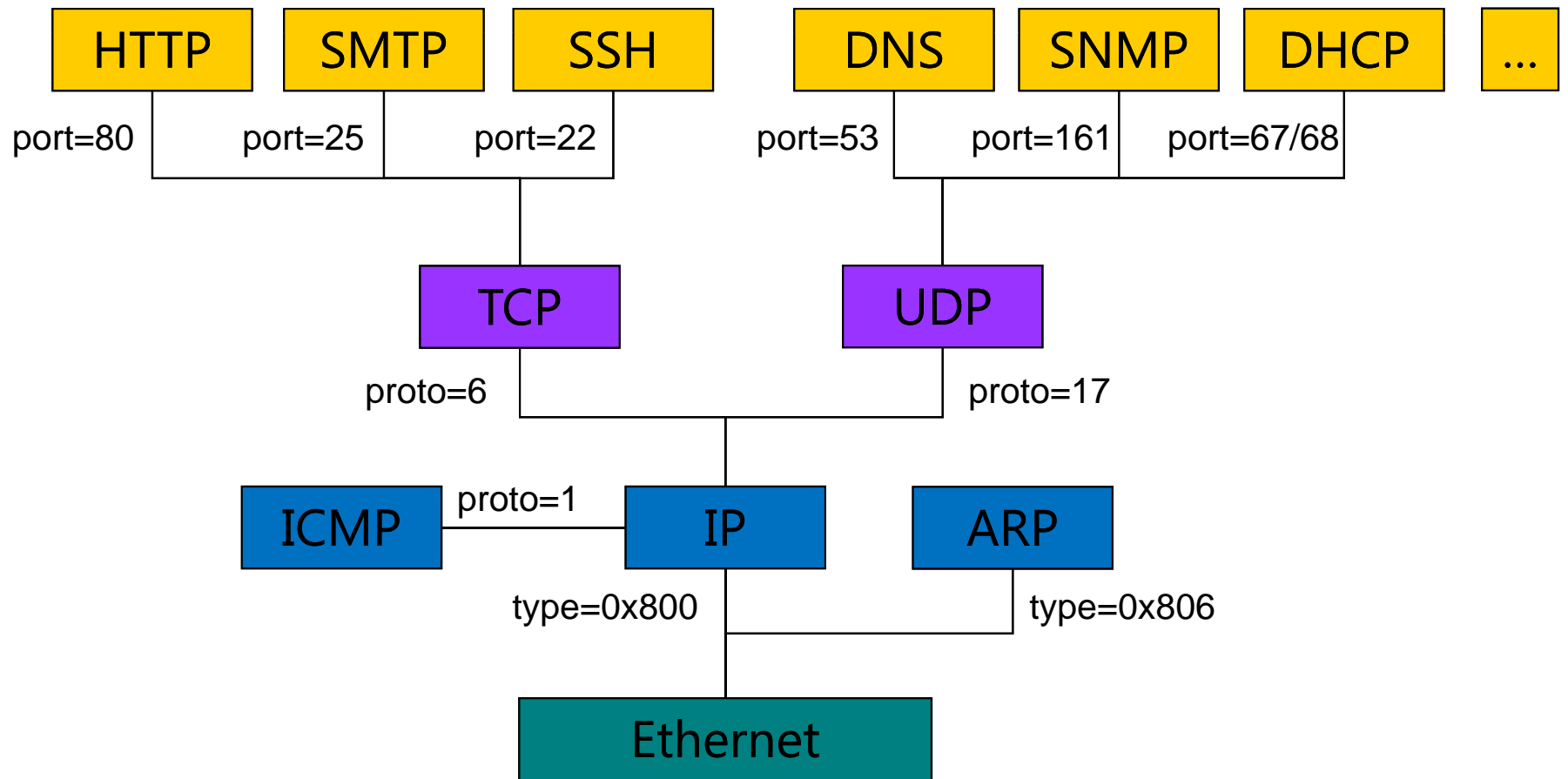
Les sockets

- Avec les protocoles UDP et TCP, une connexion est entièrement définie sur chaque machine par :
 - le type de protocole (UDP ou TCP)
 - l'adresse IP
 - le numéro de port associé au processus
 - serveur : port local sur lequel les connexions sont attendues
 - client : allocation dynamique par le système

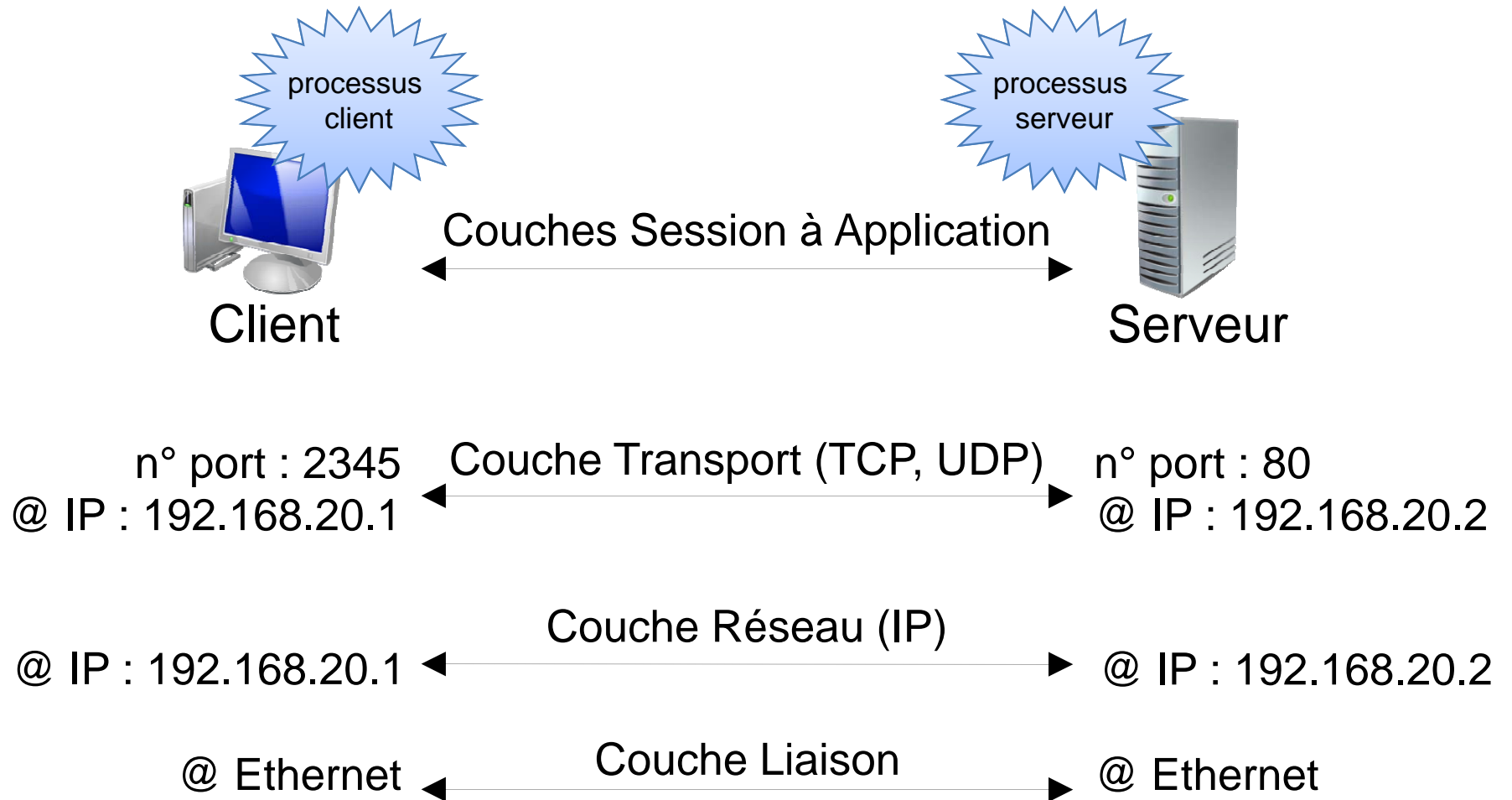
Notion de port

- Un service rendu par un programme serveur sur une machine est accessible par un port
- Un port est identifié par un entier (16 bits)
 - de 0 à 1023
 - ports **reconnus** ou **réservés**
 - sont assignés par l'IANA (*Internet Assigned Numbers Authority*)
 - donnent accès aux services standard : courrier (SMTP port 25), serveur web (HTTP port 80) ...
 - > 1024
 - ports « **utilisateurs** » disponibles pour placer un service applicatif quelconque
- Un service est souvent connu par un nom (FTP, ...)
 - La correspondance entre nom et numéro de port est donnée par le fichier `/etc/services`

Identification des protocoles

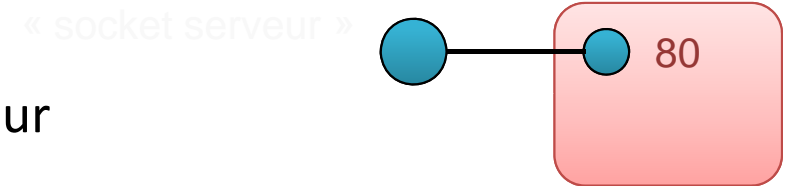


Les sockets



Principes de fonctionnement (mode concurrent)

- ❶ Le serveur crée une « socket serveur » (associée à un port) et se met en attente




- ❷ Le client se connecte à la socket serveur
 - Deux sockets sont alors créés
 - Une « socket client » côté client
 - Une « socket service client » côté serveur
 - Ces sockets sont connectées entre elles




- ❸ Le client et le serveur communiquent par les sockets
 - L'interface est celle des fichiers (read, write)
 - La socket serveur peut accepter de nouvelles connexions

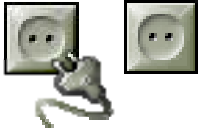
Mode connecté / non connecté

- Deux réalisations possibles
 - Mode connecté (protocole TCP)
 - Mode non connecté (protocole UDP)
- Mode **connecté** 
 - Ouverture d'une liaison, suite d'échanges, fermeture de la liaison
 - Le serveur préserve son état entre deux requêtes
 - Garanties de TCP : ordre, contrôle de flux, fiabilité
 - Adapté aux échanges ayant une certaine durée (plusieurs messages)

Mode connecté / non connecté

- Mode **non connecté** 
 - Les requêtes successives sont indépendantes
 - Pas de préservation de l'état entre les requêtes
 - Le client doit indiquer son adresse à chaque requête (pas de liaison permanente)
 - Pas de garanties particulières (UDP)
 - gestion de toutes les erreurs à la main : il faut réécrire la couche transport !!!
 - Adapté
 - aux échanges brefs (réponse en 1 message)
 - pour faire de la diffusion

Mode connecté / non connecté

- Points communs 
 - Le client **a l'initiative** de la communication
 - le serveur doit être à **l'écoute**
 - Le client doit connaître la référence du serveur (adresse IP, numéro de port)
 - en la trouvant dans un annuaire, si le serveur l'y a enregistrée au préalable
 - en utilisant les numéros de ports préaffectés par convention
 - Le serveur peut servir plusieurs clients

Utilisation du mode connecté



- Caractéristiques
 - Établissement préalable d'une connexion (circuit virtuel) : le client demande au serveur s'il accepte la connexion
 - Fiabilité assurée par le protocole (TCP)
 - Mode d'échange par **flots d'octets** : le récepteur n'a pas connaissance du découpage des données effectué par l'émetteur
 - Possibilité d'émettre et de recevoir des caractères urgents
 - Après initialisation, le serveur est « passif » : il est activé lors de l'arrivée d'une demande de connexion du client
 - Un serveur peut répondre aux demandes de service de plusieurs clients : les requêtes arrivées et non traitées sont stockées dans une file d'attente
- Contraintes
 - Le client doit avoir accès à l'adresse du serveur (adresse IP, numéro de port)

Utilisation du mode non connecté



- Caractéristiques
 - Pas d'établissement préalable d'une connexion
 - Pas de garantie de fiabilité
 - Adapté aux applications pour lesquelles les réponses aux requêtes des clients sont courtes (1 message)
 - Le récepteur reçoit les données selon le découpage effectué par l'émetteur
- Contraintes
 - Le client doit avoir accès à l'adresse du serveur (adresse IP, numéro de port)
 - Le serveur doit récupérer l'adresse de chaque client pour lui répondre

Généralisation du schéma client / serveur

- Les notions de client et de serveur sont relatives
 - Un serveur peut faire appel à d'autres serveurs dont il est le client
 - Architecture à 3 niveaux
 - Exemple : un serveur Web faisant appel à un serveur de base de données
 - Clients et serveurs jouent parfois un rôle symétrique
 - Exemple : postes des réseaux Microsoft Windows
 - Systèmes de pair à pair (Peer to Peer, P2P)