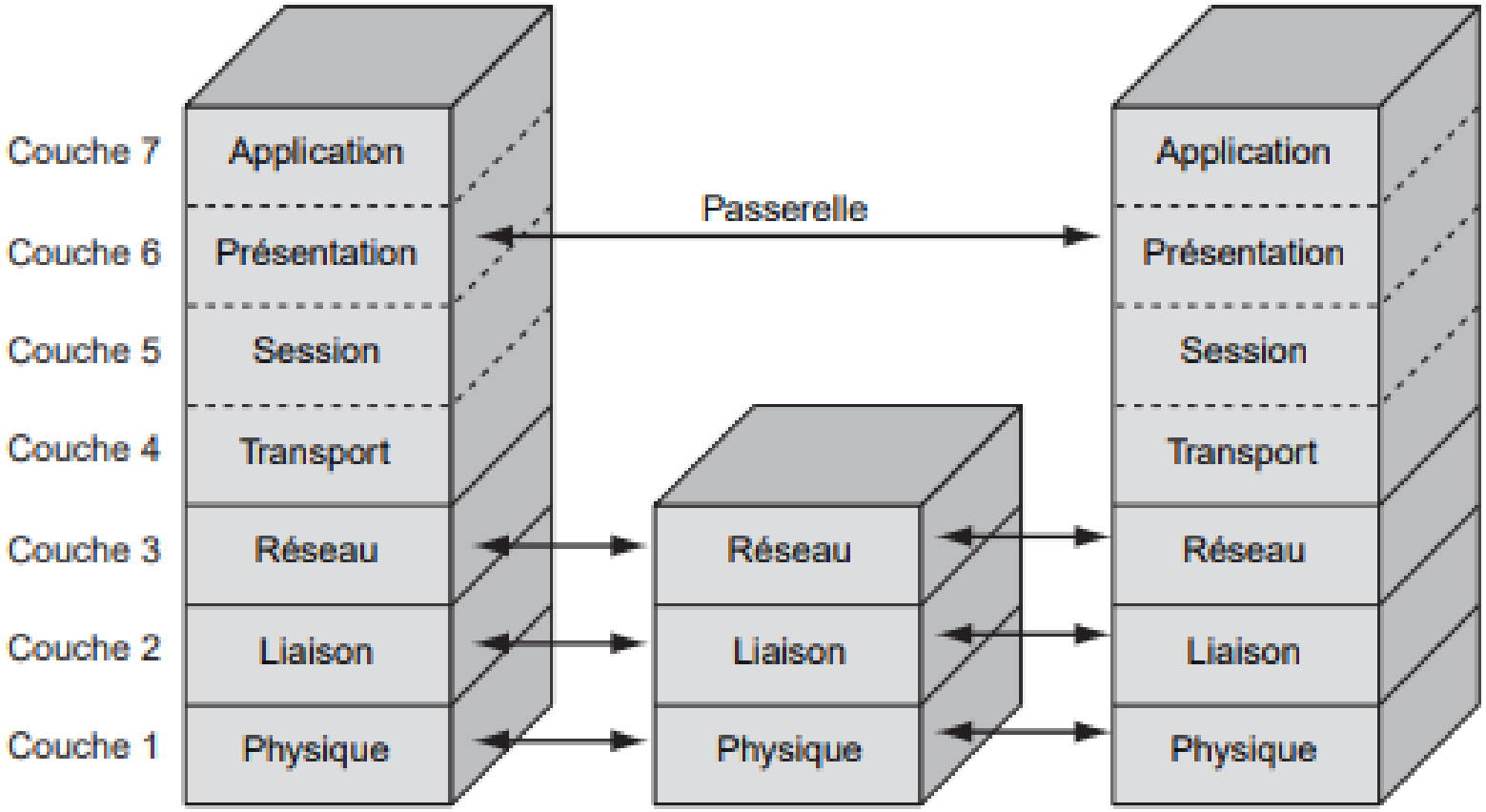
Programmation Système et Réseau

6 – Sockets

Equipe pédagogique

CY Tech
ING2 - GSI

- **Le modèle OSI**



Les niveaux paquet et message

Le niveau message concerne le transfert de bout en bout des données d'une extrémité à une autre d'un réseau.

Les données de l'utilisateur sont regroupées en messages.

Ces messages doivent être transportés de l'émetteur vers le récepteur.

C'est la raison pour laquelle ce niveau s'appelle également couche transport (couche 4).

Le message est une entité logique de l'utilisateur émetteur, sa longueur n'étant pas déterminée à l'avance

Les niveaux paquet et message

Sa définition est précise : garantir l'acheminement du message de l'émetteur au récepteur, éventuellement en traversant plusieurs réseaux.

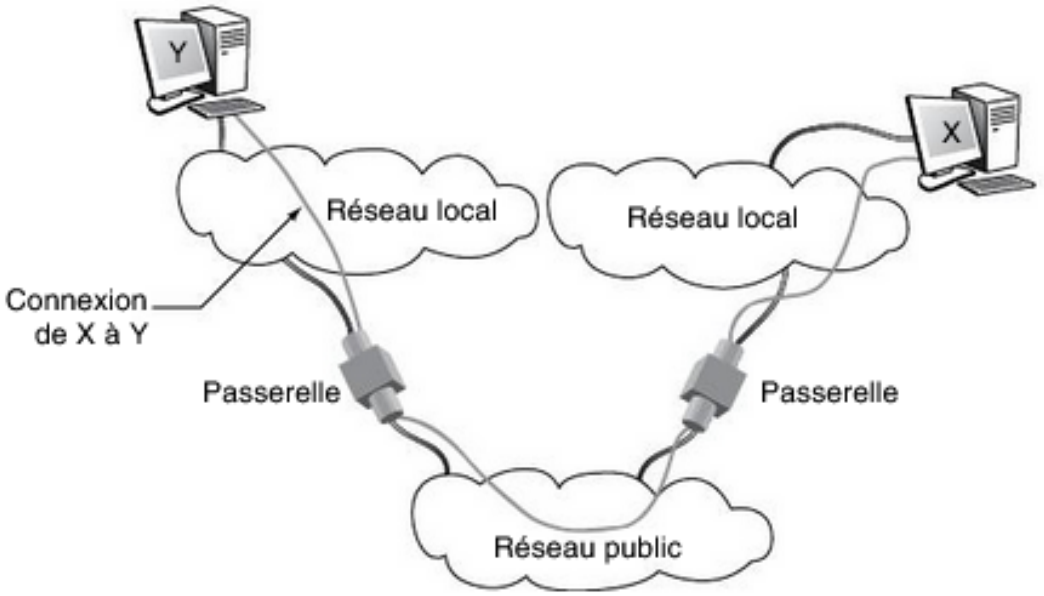
En comparaison, le niveau paquet n'a pour ambition que de faire le nécessaire pour assurer la traversée d'un réseau.

Par déduction, aucun niveau message ne doit être traversé avant d'atteindre l'équipement terminal de destination, sinon la transmission ne serait pas de bout en bout.

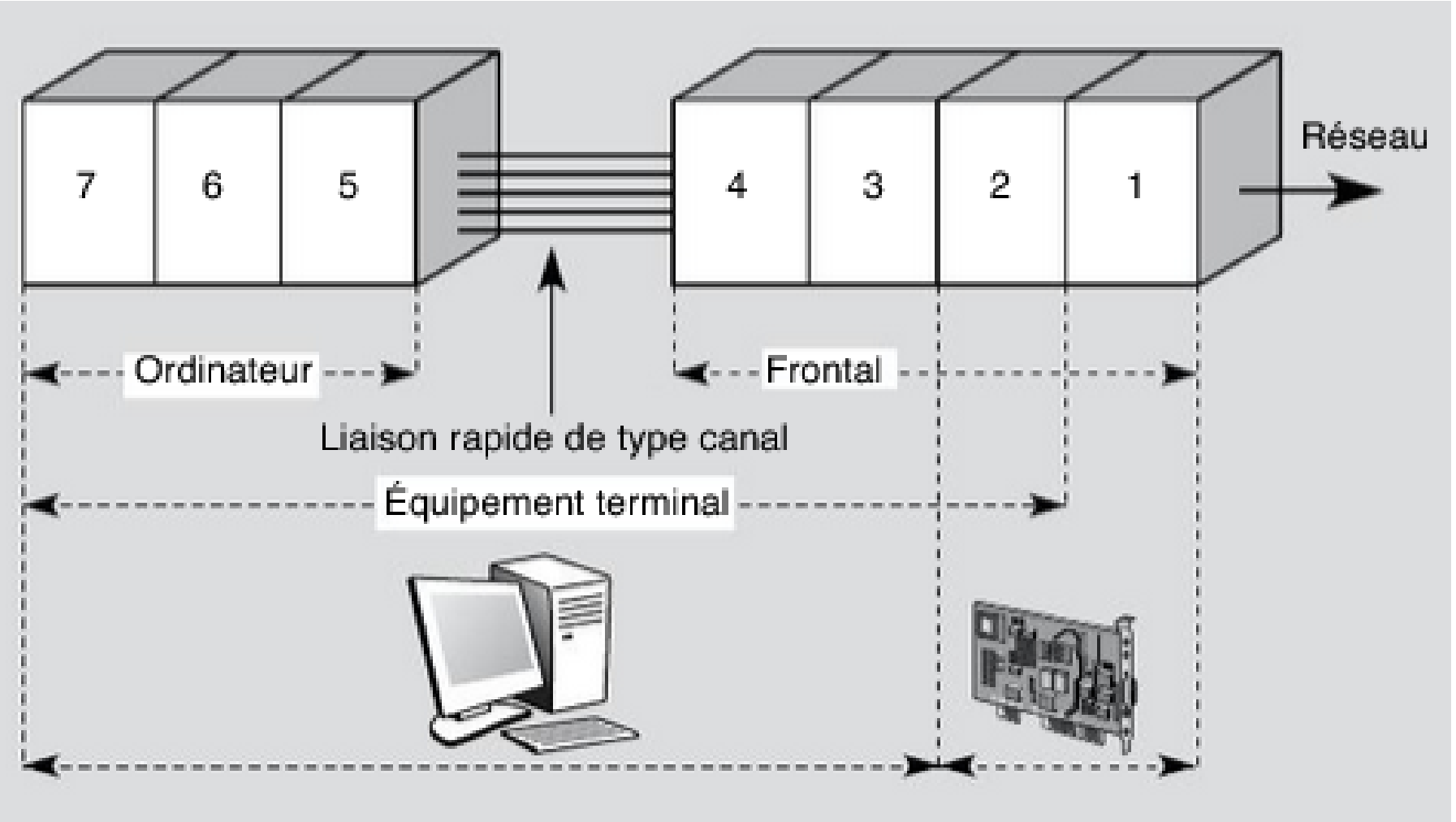
Les niveaux paquet et message

La couche transport doit permettre la communication entre deux utilisateurs situés dans des systèmes différents, indépendamment des caractéristiques des sous- réseaux sur lesquels s'effectue le transfert des données.

Un utilisateur du service de transport n'a pas la possibilité de savoir si un ou plusieurs réseaux sous-jacents sont mis en jeu dans la communication qui l'intéresse.



Couches de protocoles dans un système informatique



Protocoles niveau message

Le réseau Internet utilise le protocole IP au niveau paquet.

Le niveau message offre deux autres possibilités :

le protocole TCP (Transmission Control Protocol), qui introduit plusieurs fonctionnalités garantissant une certaine qualité du service de transport,

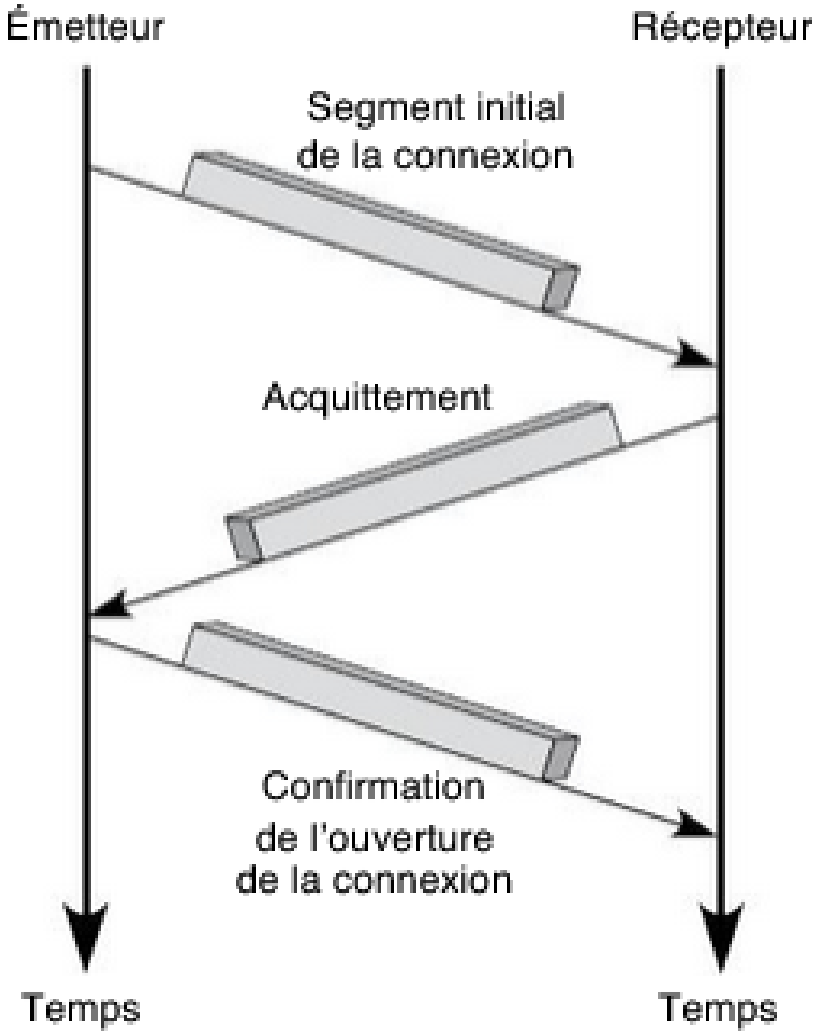
le protocole UDP (User Datagram Protocol), beaucoup plus simple, mais ne donnant aucune garantie sur le transport des messages.

La simplicité d'UDP offre en contrepartie des débits plus élevés.

Le protocole TCP offre un service de transport fiable.

Les données échangées sont considérées comme un flot de bits divisé en octets, ces derniers devant être reçus dans l'ordre où ils sont envoyés.

Le transfert des données ne peut commencer qu'après l'établissement d'une connexion entre deux machines.



Durant le transfert, les deux machines continuent à vérifier que les données transitent correctement.

Les programmes d'application envoient leurs données en les passant régulièrement au système d'exploitation de la machine.

Chaque application choisit la taille de données qui lui convient. Le transfert peut être, par exemple, d'un octet à la fois.

L'implémentation TCP est libre de découper les données en paquets d'une taille différente de celle des blocs reçus de l'application.

Pour rendre le transfert plus performant, l'implémentation TCP attend d'avoir suffisamment de données avant de remplir un datagramme et de l'envoyer sur le sous- réseau.

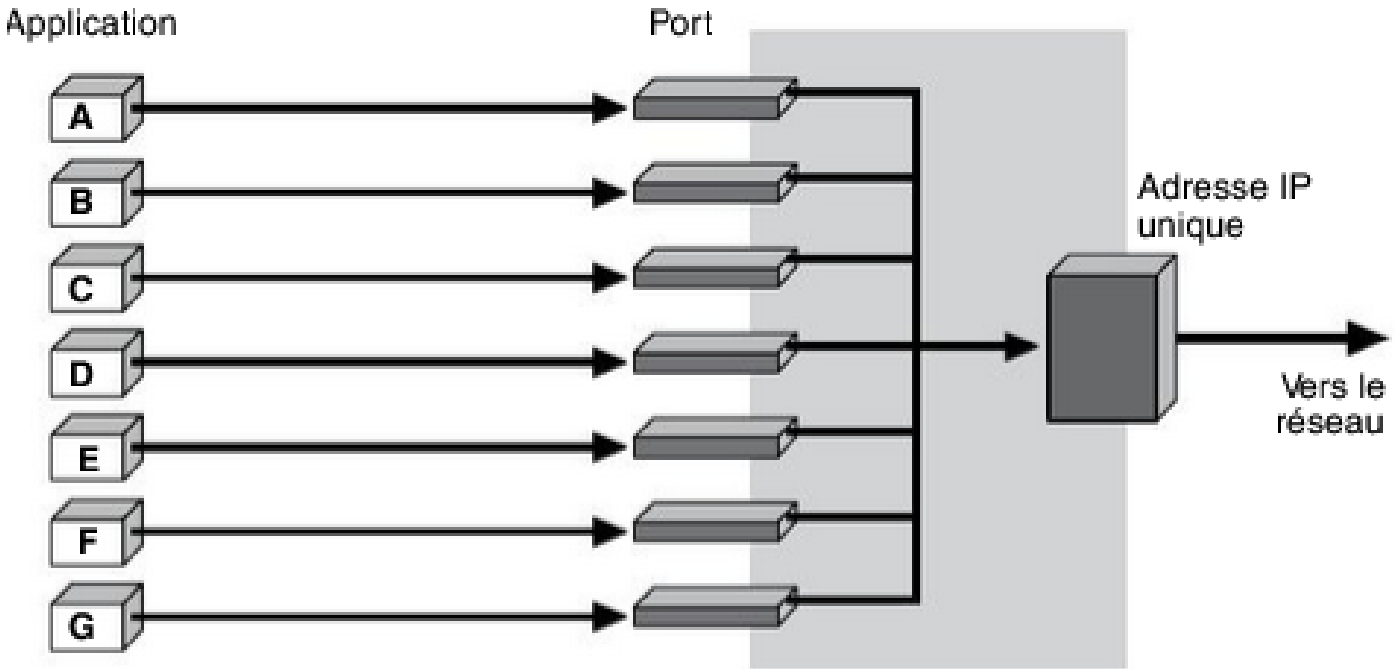
Ouverte dans les deux sens de transmission à la fois (full-duplex), la connexion garantit un transfert de données bidirectionnel, avec deux flots de données inverses, sans interaction apparente.

Il est possible de terminer l'envoi dans un sens sans arrêter celui dans l'autre sens. Cela permet d'envoyer des acquittements dans un sens de transmission en même temps que des données dans l'autre sens.

Le protocole TCP définit la structure des données et des acquittements échangés, ainsi que les mécanismes permettant de rendre le transport fiable. Il spécifie comment distinguer plusieurs connexions sur une même machine et comment détecter des paquets perdus ou dupliqués et remédier à cette situation.

TCP autorise plusieurs programmes à établir une connexion simultanée et à multiplexer les données reçues des différentes applications.

Il utilise pour cela la notion abstraite de port, qui identifie une destination particulière dans une machine (extrémité de connexion : @IP + n°port).



Protocoles niveau message

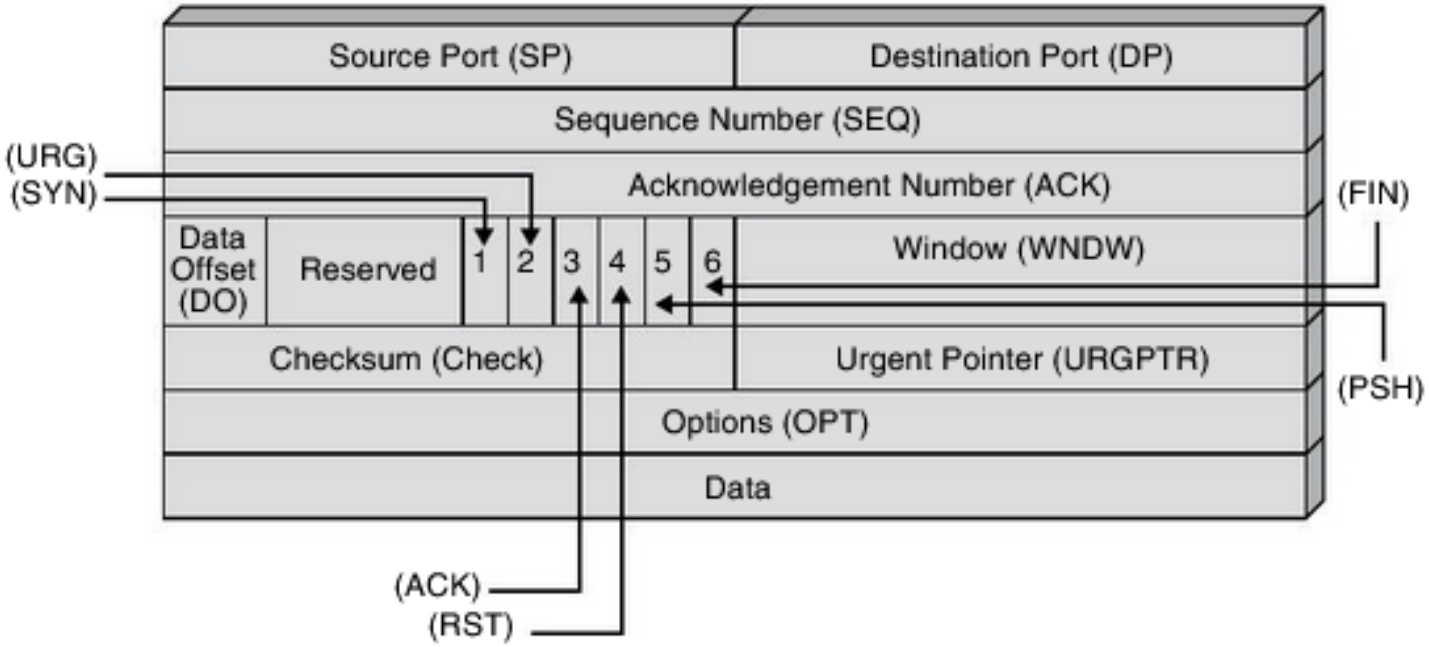
TCP est un protocole en mode avec connexion entre deux points extrémité d'une connexion.

Le programme d'une extrémité effectue une ouverture de connexion passive : il accepte une connexion entrante en lui affectant un numéro de port.

L'autre programme d'application exécute une ouverture de connexion active.

Dès la connexion établie, le transfert de données peut commencer.

Pour le protocole TCP, un flot de données est une suite d'octets groupés en fragments qui donnent naissance à un paquet IP.



Le protocole UDP permet aux applications d'échanger des datagrammes.

Il utilise pour cela la notion de port, qui permet de distinguer les différentes applications qui s'exécutent sur une machine.

Outre le datagramme et ses données, un message UDP contient un numéro de port source et un numéro de port destination.

Le protocole UDP fournit un service en mode sans connexion et sans reprise sur erreur.

Il n'utilise aucun acquittement, ne reséquence pas les messages et ne met en place aucun contrôle de flux. Il se peut donc que les messages UDP qui se perdent soient dupliqués, remis hors séquence ou qu'ils arrivent trop tôt pour être traités lors de leur réception.

Protocoles niveau message

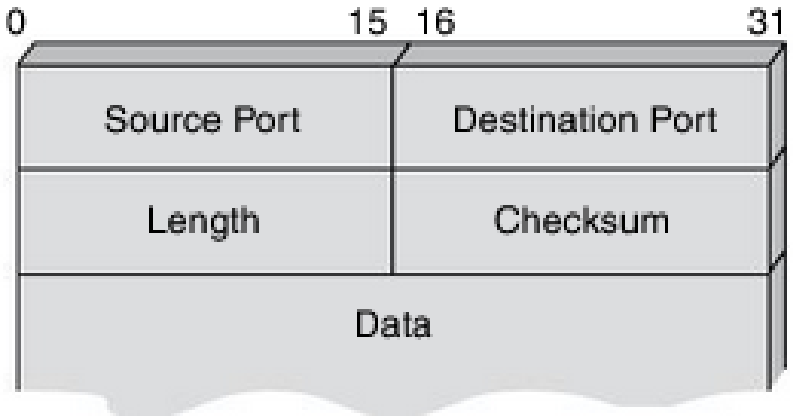
UDP est un protocole particulièrement simple du niveau message de l'architecture mais il présente l'avantage d'une exécution rapide, tenant compte de contraintes temps réel ou d'une limitation de place sur un processeur.

Ces contraintes ou limitations ne permettent pas toujours l'utilisation de protocoles plus lourds, comme TCP.

Les applications qui n'ont pas besoin d'une forte sécurité au niveau transmission, et elles sont nombreuses, ainsi que les logiciels de gestion, qui requièrent des interrogations rapides de ressources, préfèrent utiliser UDP.

Les demandes de recherche dans les annuaires transitent par UDP, par exemple.

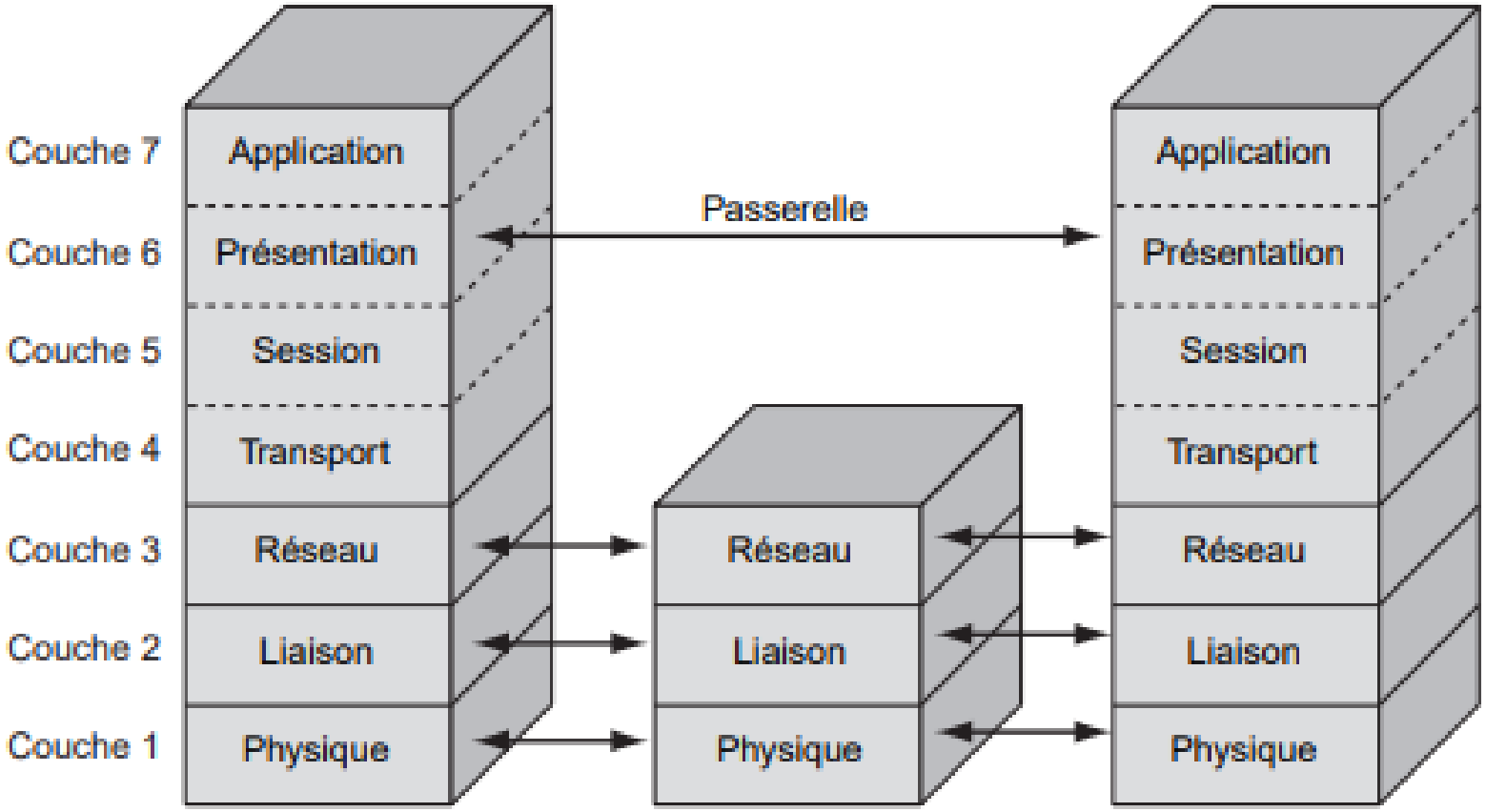
Pour identifier les différentes applications, UDP impose de placer dans chaque fragment une référence qui joue le rôle de port. Une référence identifie avec une pseudo-entête ce qui est transporté dans le corps du fragment : 8 octets avec 2o port source, 2o port destination, 2o longueur du datagramme exprimée en octets, 2o CRC sur la totalité du datagramme)



Les applications les plus importantes qui utilisent le protocole UDP correspondent aux numéros de port suivants :

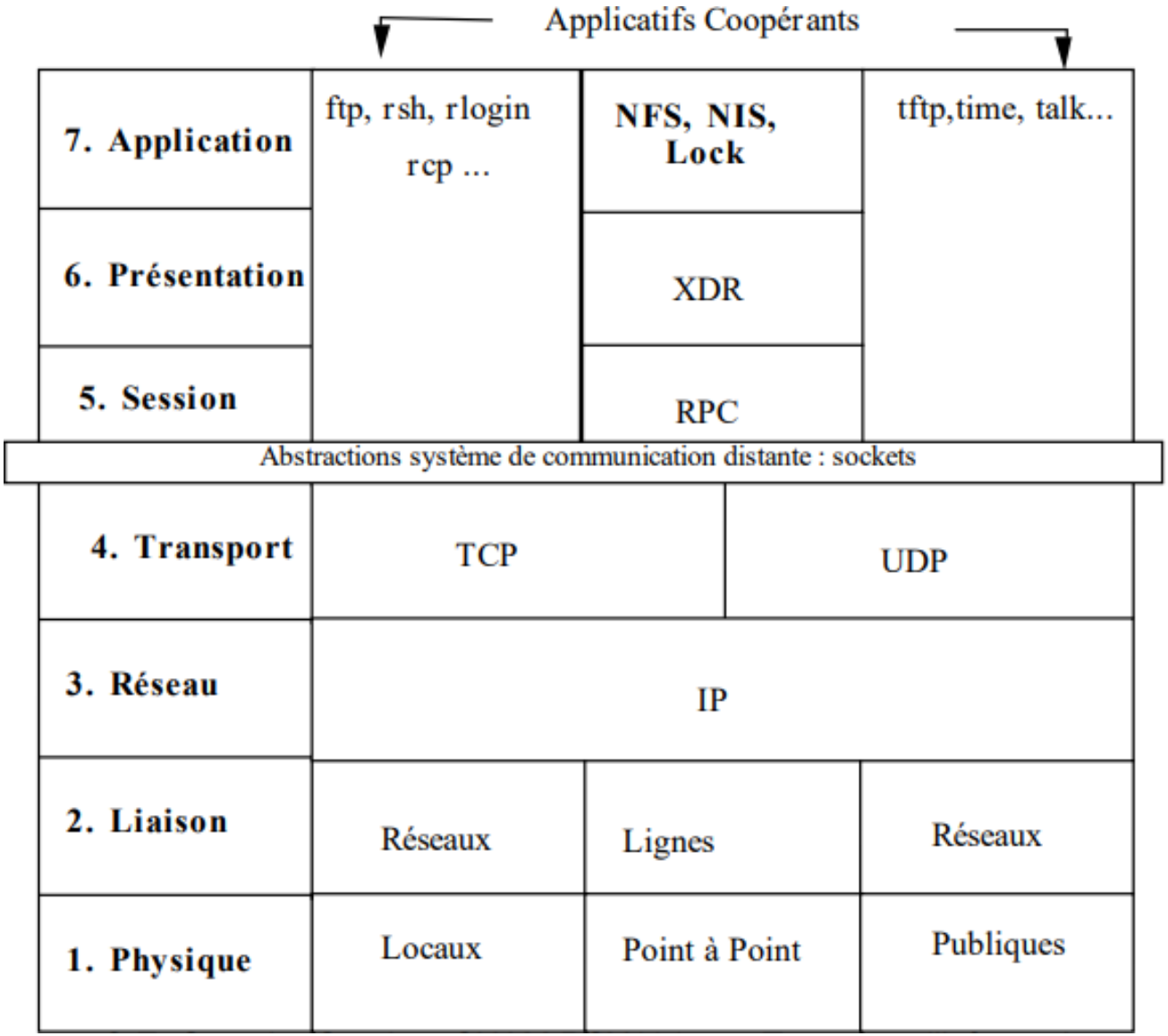
- 7 : service écho ;
- 9 : service de rejet ;
- 53 : serveur de nom de domaine DNS (Domain Name Server) ;
- 67 : serveur de configuration DHCP ;
- 68 : client de configuration DHCP.

Le modèle OSI



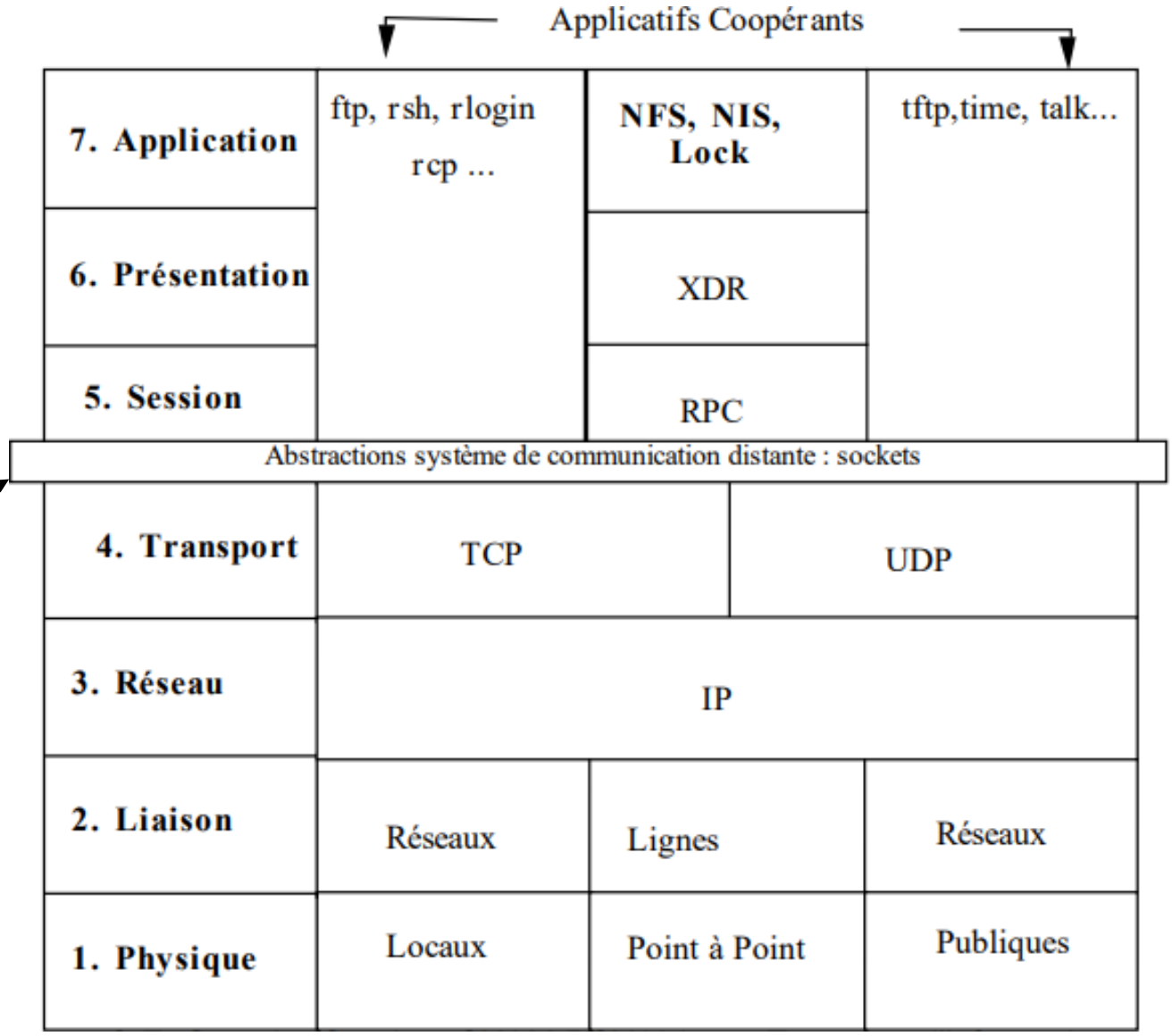
Les prises de communication (Socket)

Le modèle OSI



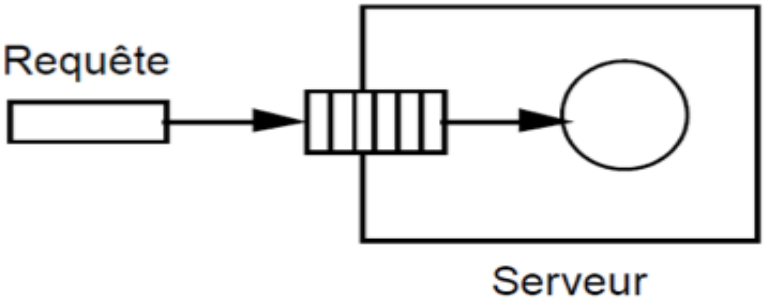
Les prises de communication (Socket)

Le modèle OSI

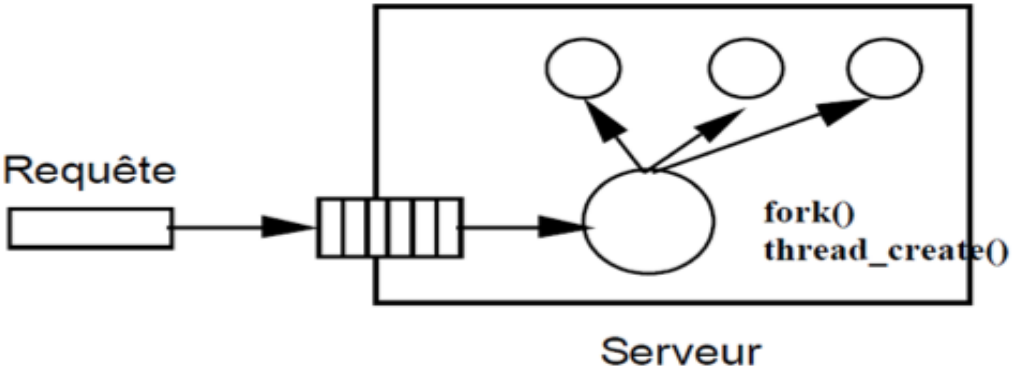


Les sockets : interface offrant un ensemble de fonctions qui permettent d'accéder aux couches de protocoles TCP/IP

L'interaction application Client et application Serveur



Serveur Itératif
Un seul processus effectue la réception, le traitement et l'émission



Serveur Parallèle
Le processus père effectue la réception. Il crée un fils pour réaliser le traitement et l'émission de la réponse.

Les prises de communication (Socket)

L’interaction application Client et application Serveur

On va utiliser des IPC (Inter-Process Communication) entre le client et le serveur

Indépendance du système d’exploitation, du matériel et du langage mais on préfère le langage C

Les API (Application Programming Interface) Socket datent des années 80 mais restent toujours d’actualité et elle a inspiré de nombreuses autres API

Socket, prise en anglais, c'est juste une boîte aux lettres.

Les prises de communication (Socket)

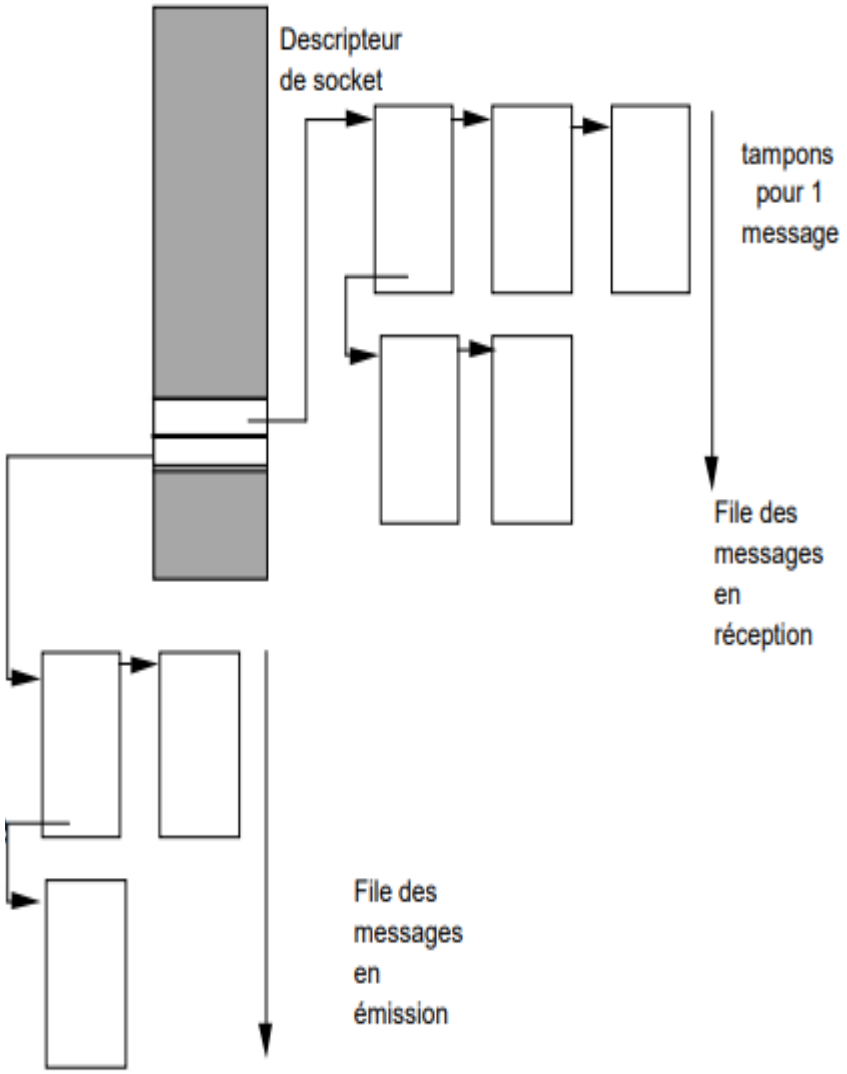
- **L'interaction Client/Serveur**

C'est essentiellement une file de tampons mémoire (buffers) en entrée et en sortie

Et des méthodes pour les manipuler, des attributs du système…

… Et c'est donc le cœur de la couche Transport finalement.

Un socket autorise la communication entre deux processus différents sur la même machine ou entre machines distantes.

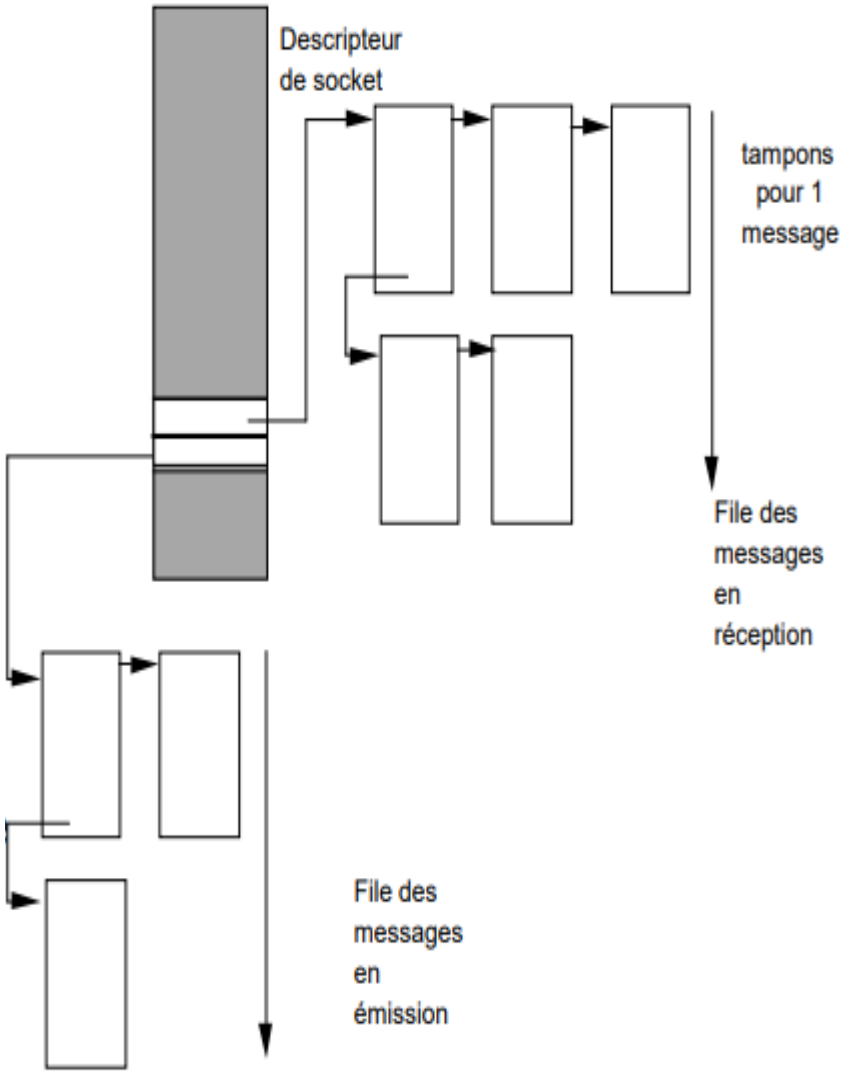


Les prises de communication (Socket)

- **L'interaction Client/Serveur**

Un(e) socket est un point de communication bidirectionnel par lequel un processus pourra émettre ou recevoir des informations.

On peut imaginer une communication entre personnes par courrier qui nécessite que les personnes disposent d'une boîte aux lettres qui est un point de communication ayant une adresse connue du monde extérieur.

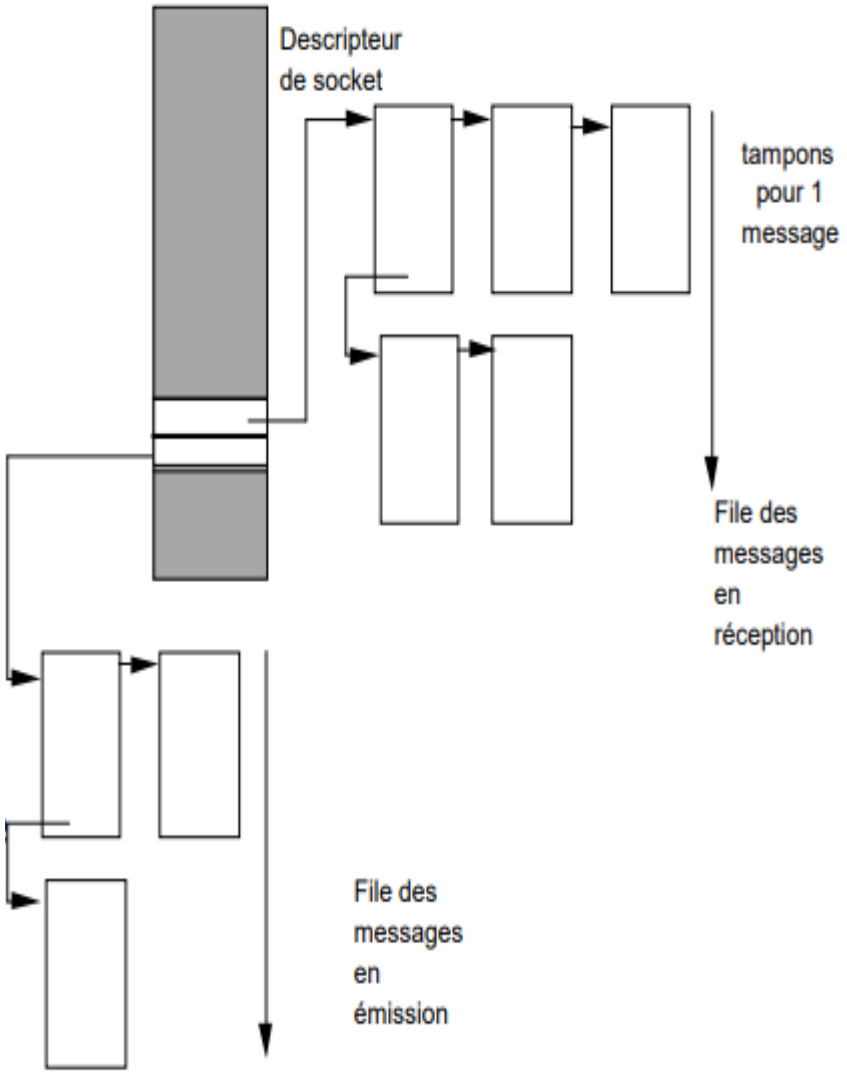


Les prises de communication (Socket)

- **L'interaction Client/Serveur**

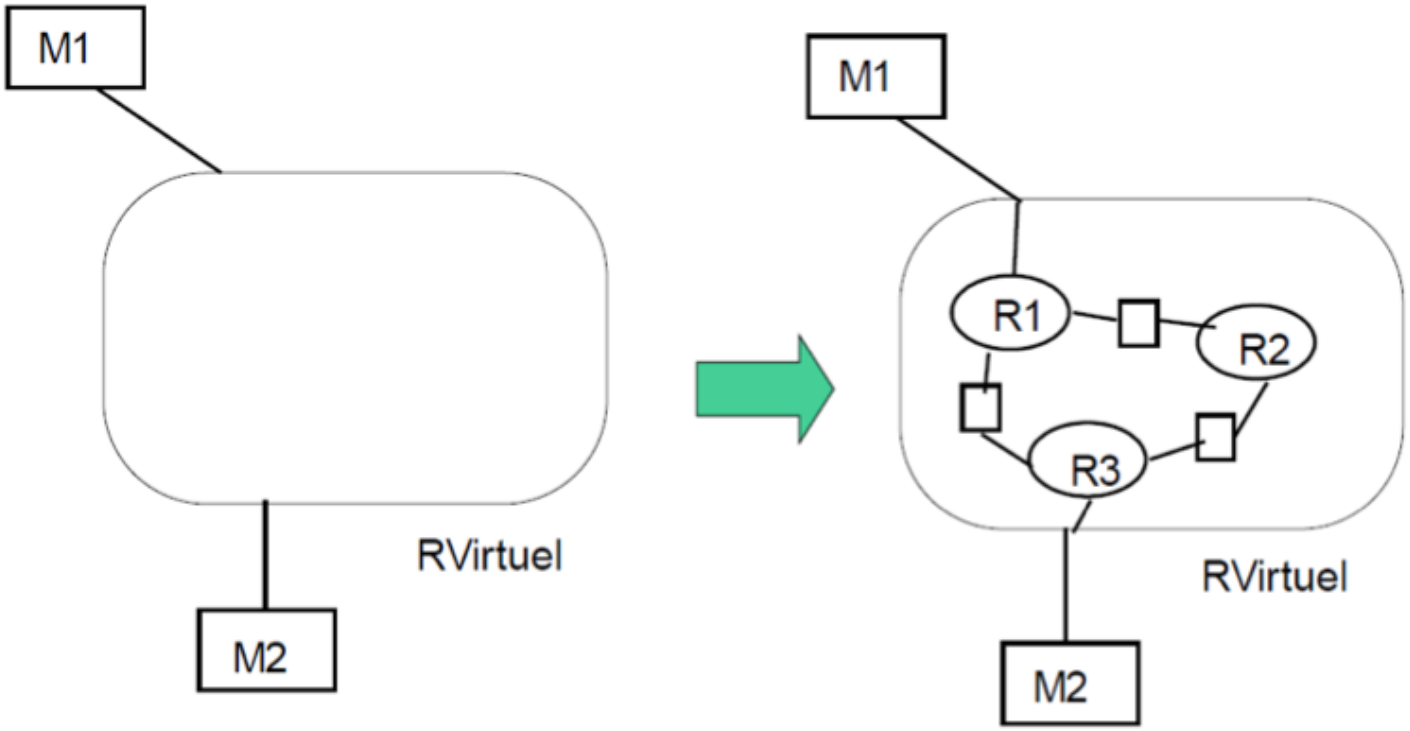
Quand une communication entre deux sockets est établie, on ne se préoccupe pas de savoir comment est réalisée la liaison réelle.

Si par exemple la communication se fait grâce à des machines intermédiaire, le programme utilisant les sockets l'ignore complètement.



Sockets

- Interconnexion de réseau



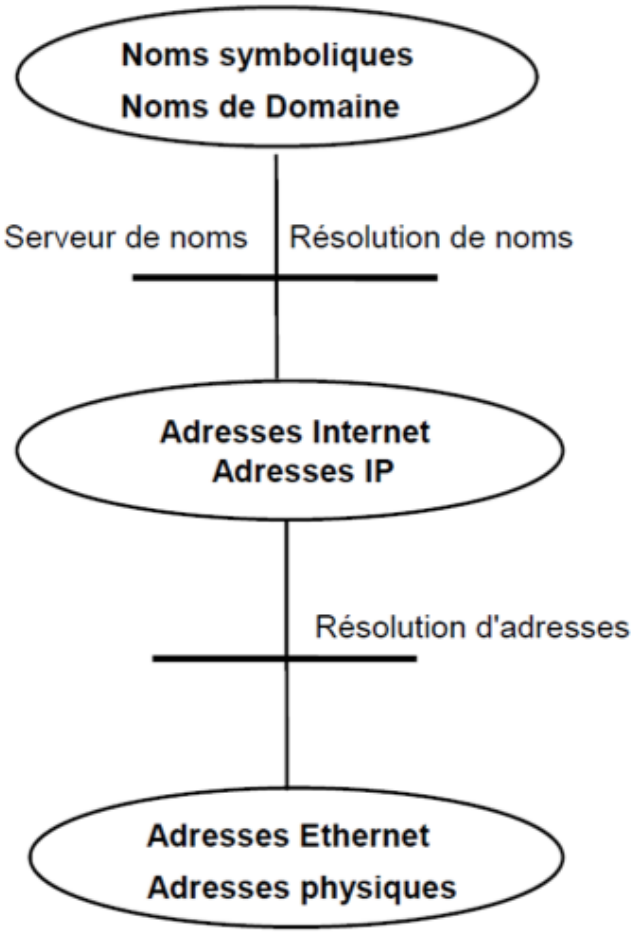
- Construire un réseau virtuel
  - adressage (adresse IP, noms symboliques)
  - transport de bout en bout (protocole TCP/IP)

Sockets

- Interconnexion de réseau
  - Structure hiérarchique fondée sur **le domaine** (DNS)

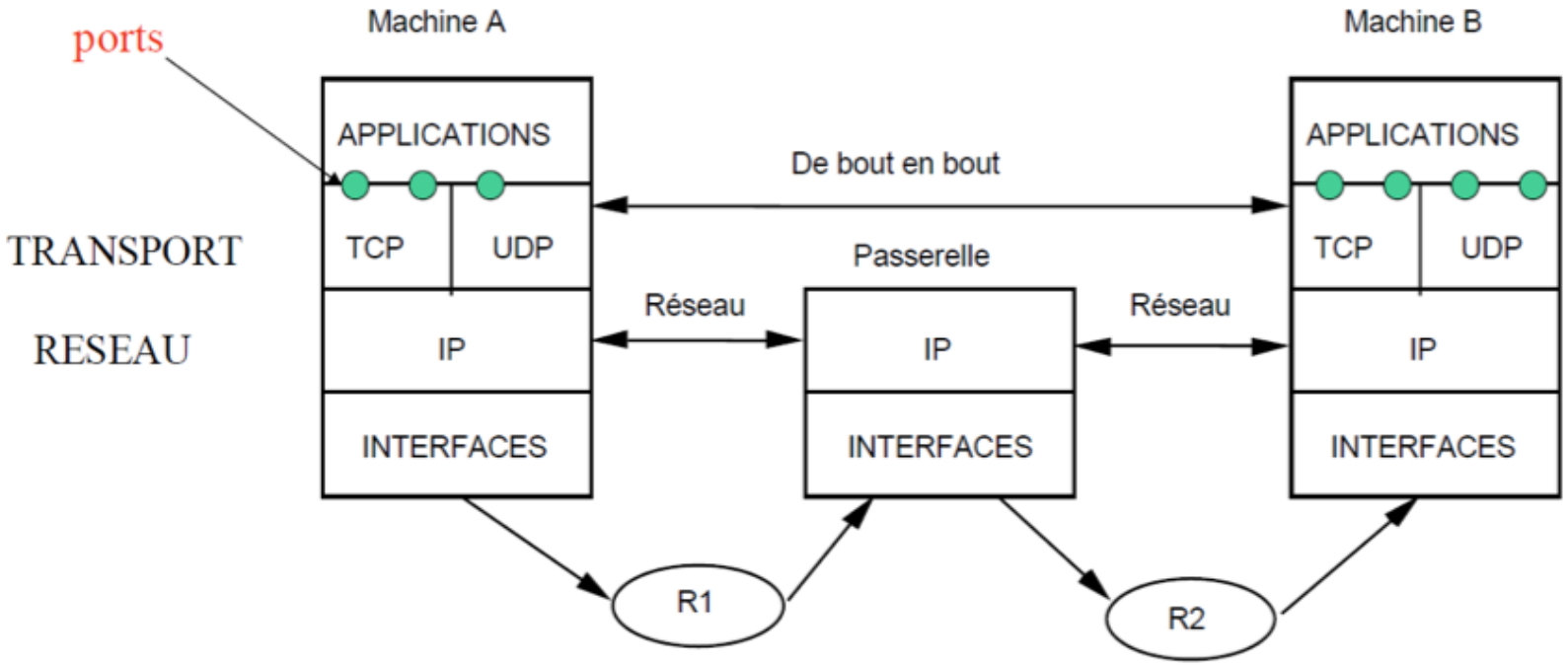
    objet.sous-domaine.domaine
    ex: fermi.cnam.fr
    objet : nom d'une machine.
    domaine : géographique (fr, jp)
    ou institutionnel (com, mil, edu, name)
  - **Adresses IP** : 32 bits
    Paire (adresse réseau
    , adresse machine dans le réseau)

    Forme pointée :
    10000000 00001010 00000010 00011110
    128.10.2.30



Sockets

Interconnexion de réseau

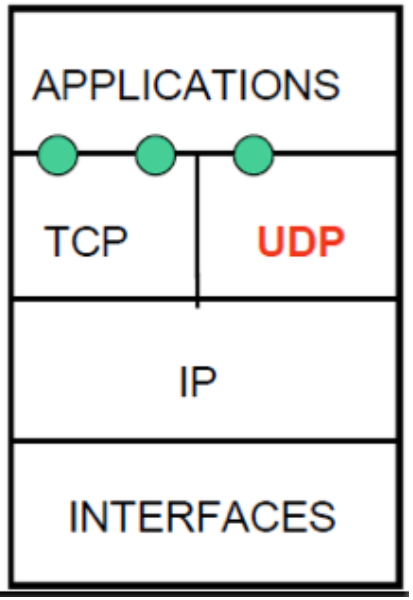


- **IP** : adressage de machine à machine via adresse IP
- **TCP/UDP** : adressage d'applications à applications
  - notion de ports (entier 16 bits)

Sockets

Interconnexion de réseau : couche UDP
User Datagram Protocol

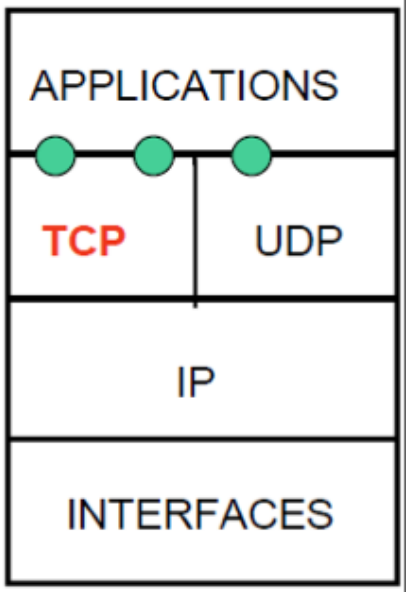
- Protocole transport de bout en bout
  - Adressage d'application à application via les ports UDP
  - Ports UDP réservés
  - exemple : port 513 application who
- Protocole transport non fiable basé sur IP
  - Datagrammes indépendants
  - Perte de datagrammes
  - Datagrammes dupliqués
  - Pas de remise dans l'ordre d'émission
  - exemple : le courrier



Sockets

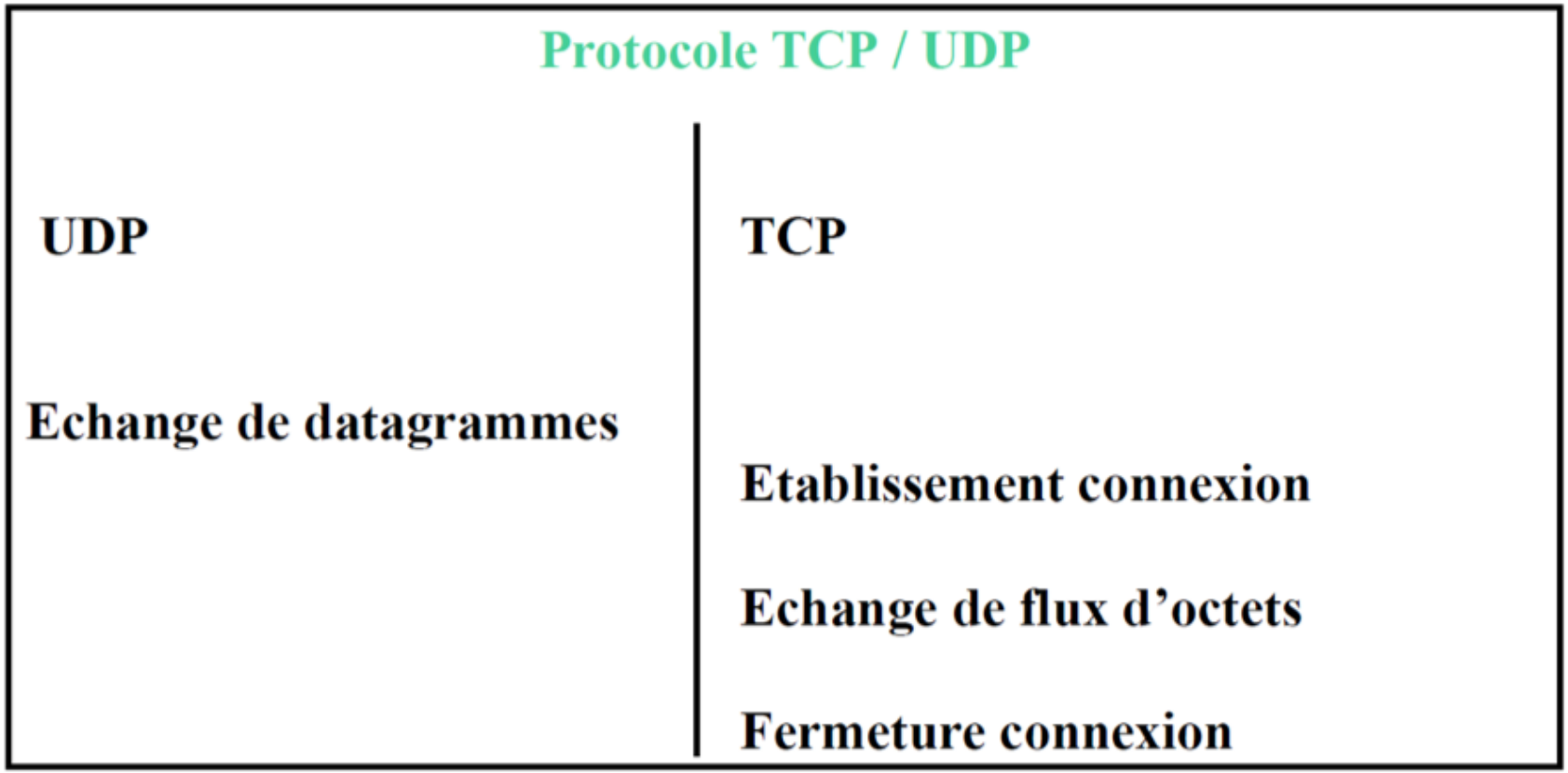
Interconnexion de réseau : couche TCP
Transfert Control Protocol

- Protocole transport de bout en bout
  - Adressage d'application à application via les ports TCP
  - Ports TCP réservés
  - exemple : port 21 application FTP
- Protocole transport fiable orienté connexion basée sur IP
  - Connexion / Flux d'octets
  - Pas de perte de messages
  - Pas de duplication
  - Remise dans l'ordre d'émission
  - Analogie : le téléphone



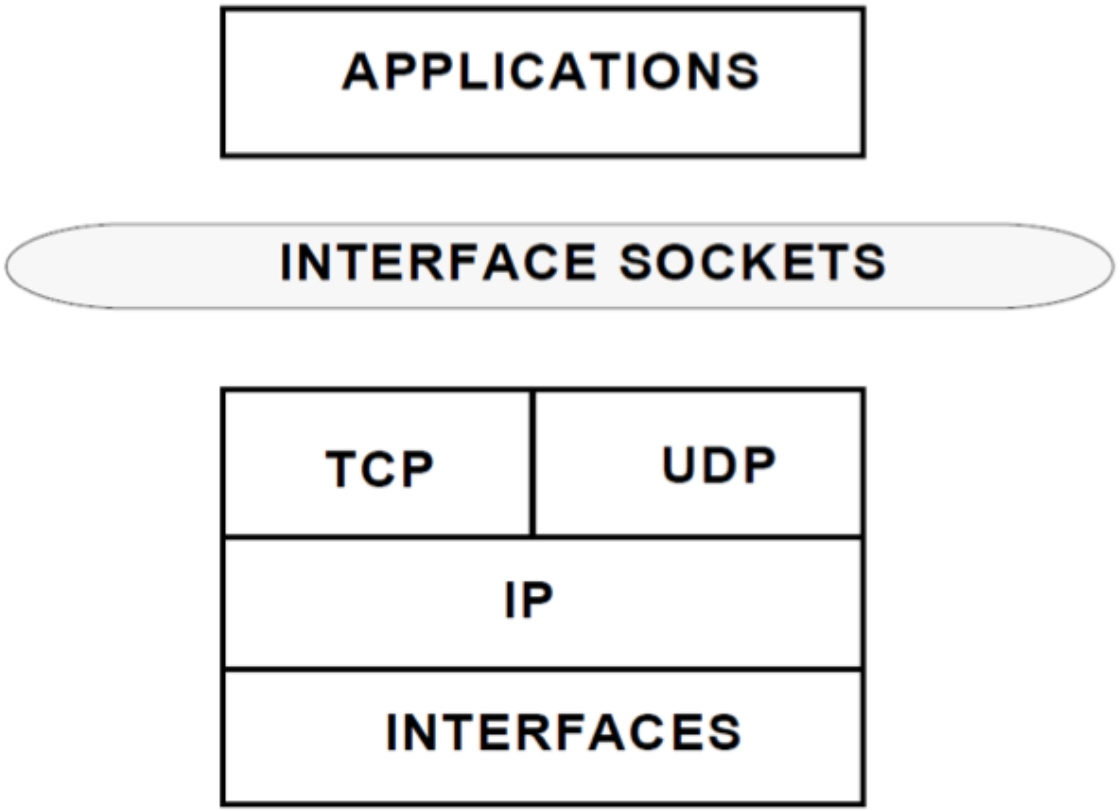
Sockets

- Interconnexion de réseau



Sockets

- Interface socket



- Interface de programmation (ensemble de primitives)
- Point de communication (adresse d'application)
- Compatible SGF

Sockets

- Interface socket : Création
  - `sock = socket (af, type, protocole);`
    - Famille
      - TCP-IP : `AF_INET`
      - UNIX : `AF_UNIX`
    - Type de service : `SOCK_STREAM`, `SOCK_DGRAM` ou `SOCK_RAW`
    - Protocole : `IPPROTO_TCP`, `IPPROTO_UDP` ou `0`
      - ```
        int socket (af, type, protocole)
        int af;
        int type;
        int protocole;
        ```
    - Retourne un descripteur de socket ayant les mêmes propriétés qu'un descripteur de fichier (héritage)
    - Accessible par le créateur et les fils de celui-ci
        ```

Sockets

- Interface socket : Attachement d'une @ d'application
  - `int bind (int sock`
    `, struct sockaddr_in *p_adresse`
    `, int lg);`
    - `sock` : descripteur de la socket
    - `p_adresse` : Pointeur en mémoire sur l'adresse, dans domaine TCP/IP `struct sockaddr_in`
    - `lg` : Longueur de la structure adresse
  - L'opération d'attachement permet d'étendre le groupe des processus pouvant accéder à la socket

Sockets

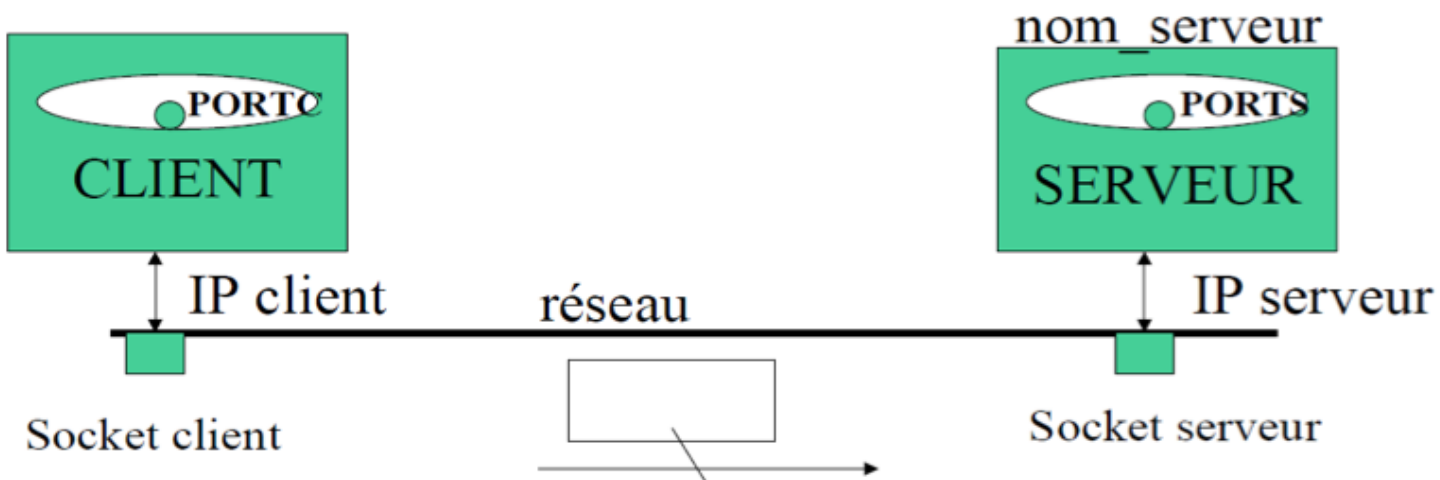
- Interface socket : Attachement d'une @ d'application
  - ```
    struct sockaddr_in {
        short sin_family; -- AF-INET
        ushort sin_port; -- n° de port
        struct in_addr sin_addr; -- @ IP
        char sin_zero [8]; -- Huit 0
    }
    ```
    - Le numéro de port est un entier en dehors des numéros de port réservés (>`IPPORT_RESERVED`). Le champ peut être mis à 0 (Choix laissé au système).
    - L'adresse IP de la machine :
      - constante `INADDR_ANY`
      - utilisation du fichier `/etc/hosts` ou serveur de noms (DNS)
    ```

Sockets

- Interface socket : Attachement d'une @ d'application
  - `/etc/hosts` : Correspondance adresse IP et nom symbolique
    `-- Adresse IP, nom symbolique, alias, commentaire`
    `163.173.128.6 asimov.cnam.fr cnam iris`
  - ```
    struct hostent {
        char *h_name; /* nom de la machine */
        char **h_aliases; /* alias */
        int h_addrtype; /* type de l'adresse */
        int h_length; /* longueur de l'adresse */
        char **h_addr_list; /* liste des adresses IP DNS */
        #define h_addr h_addr_list[0] /* 1er de la liste IP */
    };
    ```
  - Obtenir l'entrée `/etc/host` de la machine de nom "nom"
    ```
    struct hostent *gethostbyname (nom)
        char *nom;
    ```
    ```

Sockets

- Interface socket : Attachement d'une @ d'application

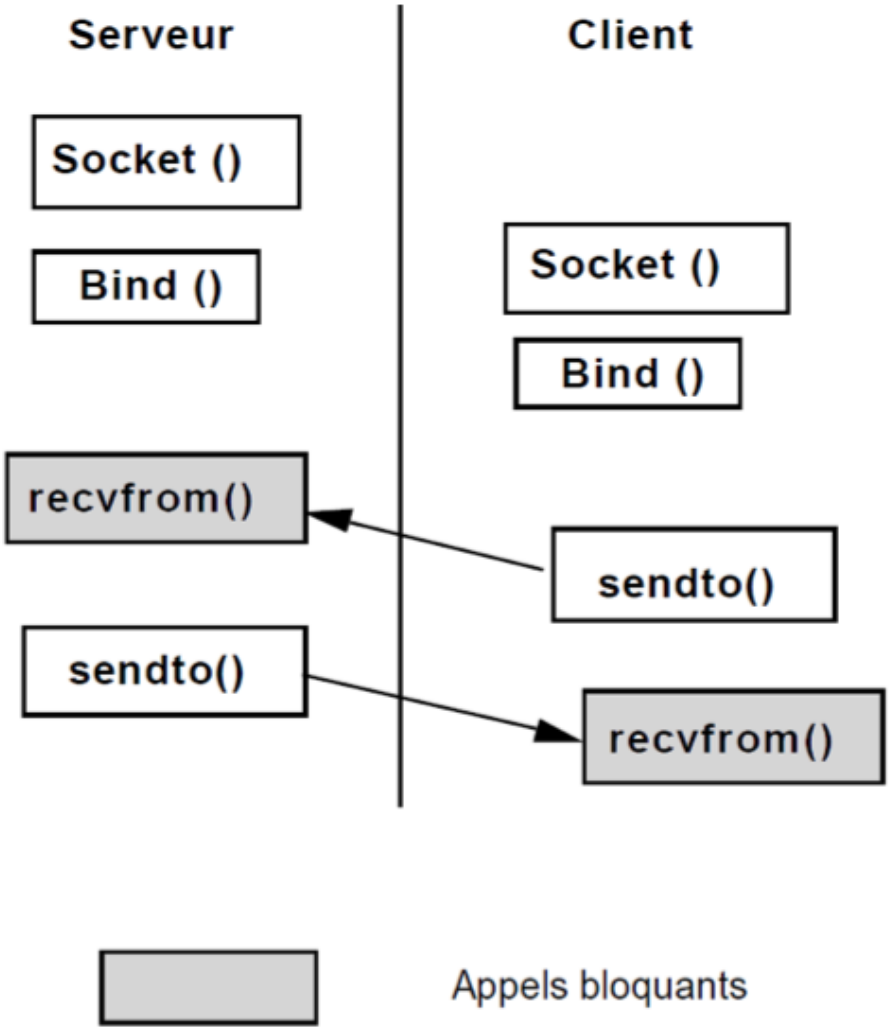


Adresse locale
Socket client
Port ← PORTC
Adresse IP ← INADDR_ANY

Adresse distante
Port ← PORTS
Adresse IP ← gethostbyname (nom_serveur)

Sockets

- Interface socket : Communication en UDP



Sock = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP)

Pas d'établissement de connexion

→ Chaque message échangé contient l'adresse du destinataire

Sockets

- Interface socket : Communication en UDP
  - `int sendto(sock, msg, lg, option, p_dest, lgdest);`
    - `int sock;` -- **socket d'émission**
    - `char *msg;` -- **@ zone mémoire contenant le message**
    - `int lg;` -- **taille en octets du message**
    - `int option;` -- **0**
    - `struct sockaddr_in *p_dest;` -- **@ du destinataire**
    - `int lgdest;` -- **taille de l'adresse du destinataire**
  - `int recvfrom(sock, msg, lg, option, p_exp, p_lgexp);`
    - `int sock;` -- **socket de réception**
    - `char *msg;` -- **@ zone mémoire pour recevoir le message**
    - `int lg;` -- **taille en octets du message**
    - `int option;` -- **0 ou MSG_PEEK**
    - `struct sockaddr_in *p_exp;` -- **@ de l'expéditeur**
    - `int *p_lgdest;` -- **taille de l'@ de l'expéditeur**

Sockets

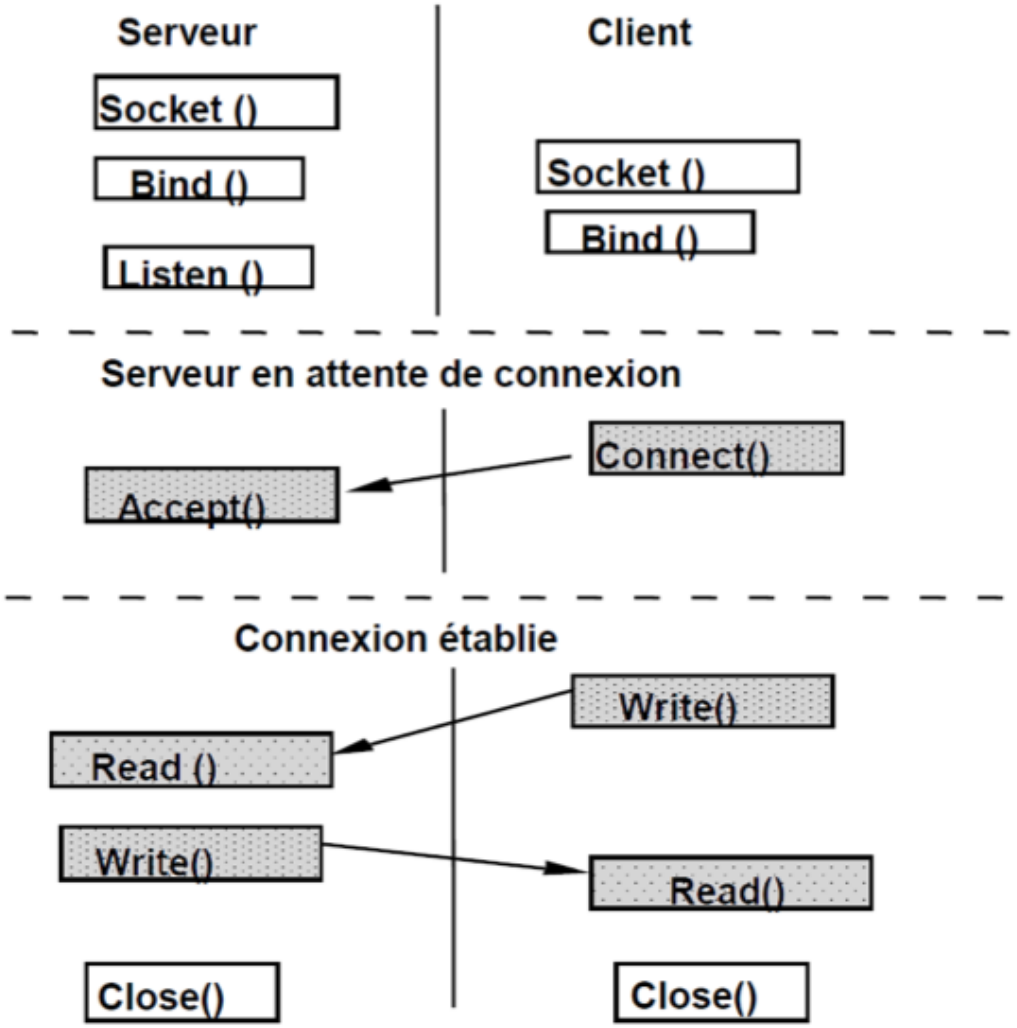
```
#include <stdio.h>   #include <sys/types.h>
#include <sys/socket.h>  #include <netinet/in.h>
#include <netdb.h>  #include <string.h>
#define PORT 2058
int main() { // serveur
int sock, lg, n; char buf[20]; struct sockaddr_in adr_s, adr_c;
sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP); // Creation socket
bzero(&adr_s,sizeof(adr_s));  adr_s.sin_family = AF_INET; adr_s.sin_port = htons(PORT);
adr_s.sin_addr.s_addr = htonl(INADDR_ANY);
bind (sock, (struct sockaddr *) &adr_s, sizeof(adr_s)); // Attachement socket
while (1) { lg = sizeof(adr_c);
                n = recvfrom (sock, buf, 20, 0, (struct sockaddr *) &adr_c, &lg);
                printf("Message reçu : %s \n", buf); } close(sock); }
```

Sockets

```
#include <stdio.h>   #include <sys/types.h>
#include <sys/socket.h>   #include <netinet/in.h>
#include <netdb.h>   #include <string.h>
#define PORT 2058
int main() { // client
int sock; char buf[20]; struct sockaddr_in adr_s, adr_c;
sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP); // Creation socket
bzero(&adr_c,sizeof(adr_c)); adr_c.sin_family = AF_INET; adr_c.sin_port = htons(PORT);
adr_c.sin_addr.s_addr = htonl(INADDR_ANY);
bind (sock, (struct sockaddr *) &adr_c, sizeof(adr_c)); // Attachement socket
bzero(&adr_s,sizeof(adr_s)); adr_s.sin_family = AF_INET; adr_s.sin_port = htons(PORT);
adr_s.sin_addr.s_addr = htonl(INADDR_ANY);
sendto (sock, "bonjour, serveur !!!", 20, 0, (struct sockaddr *) &adr_s, sizeof(adr_s)); close(sock); }
```

Sockets

- Interface socket : Communication en TCP



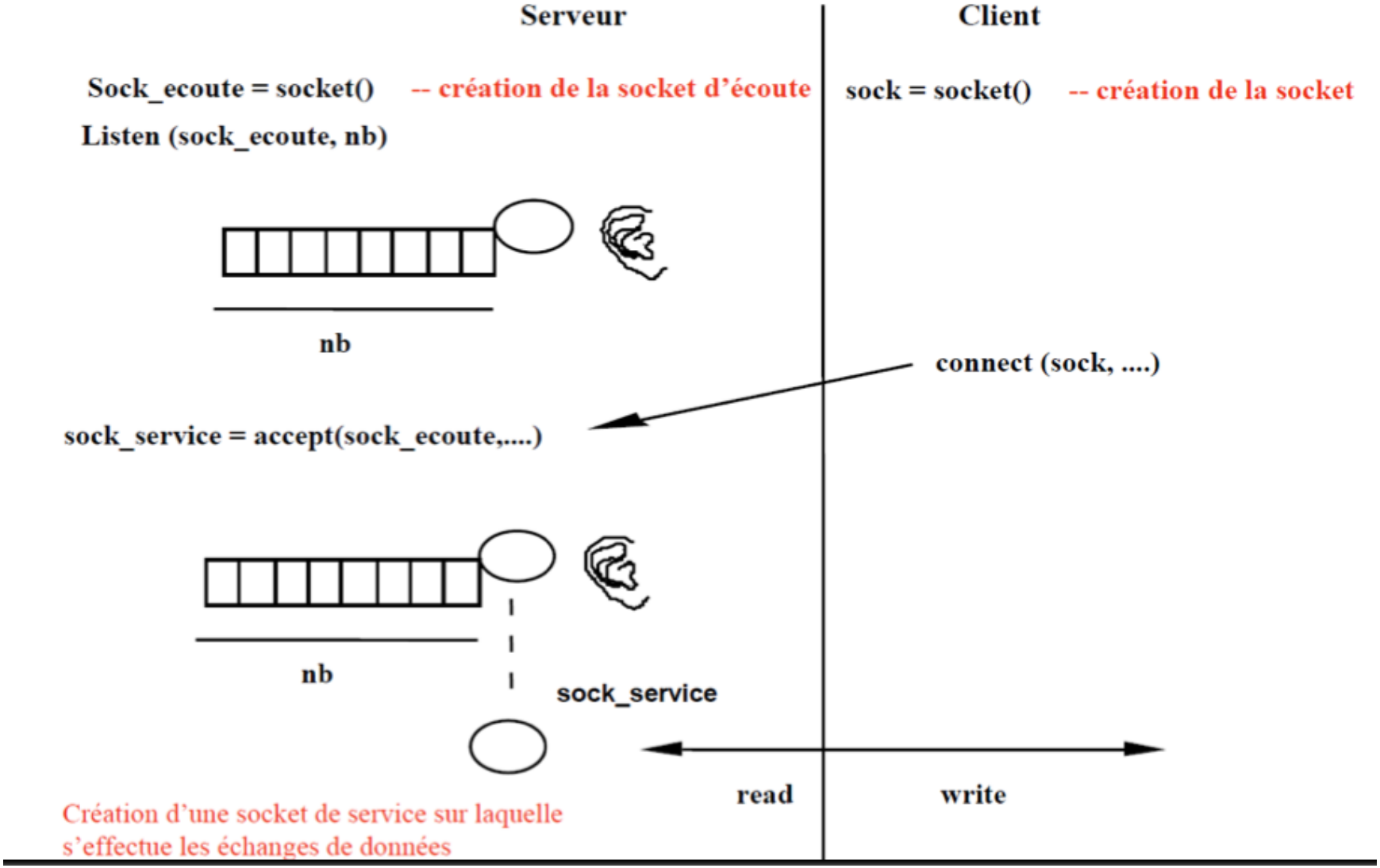
Sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)

Etablissement de connexion avec échange des adresses participantes

→ Les messages échangés ne contiennent pas l'adresse du destinataire

Sockets

- Interface socket : Communication en TCP



Sockets

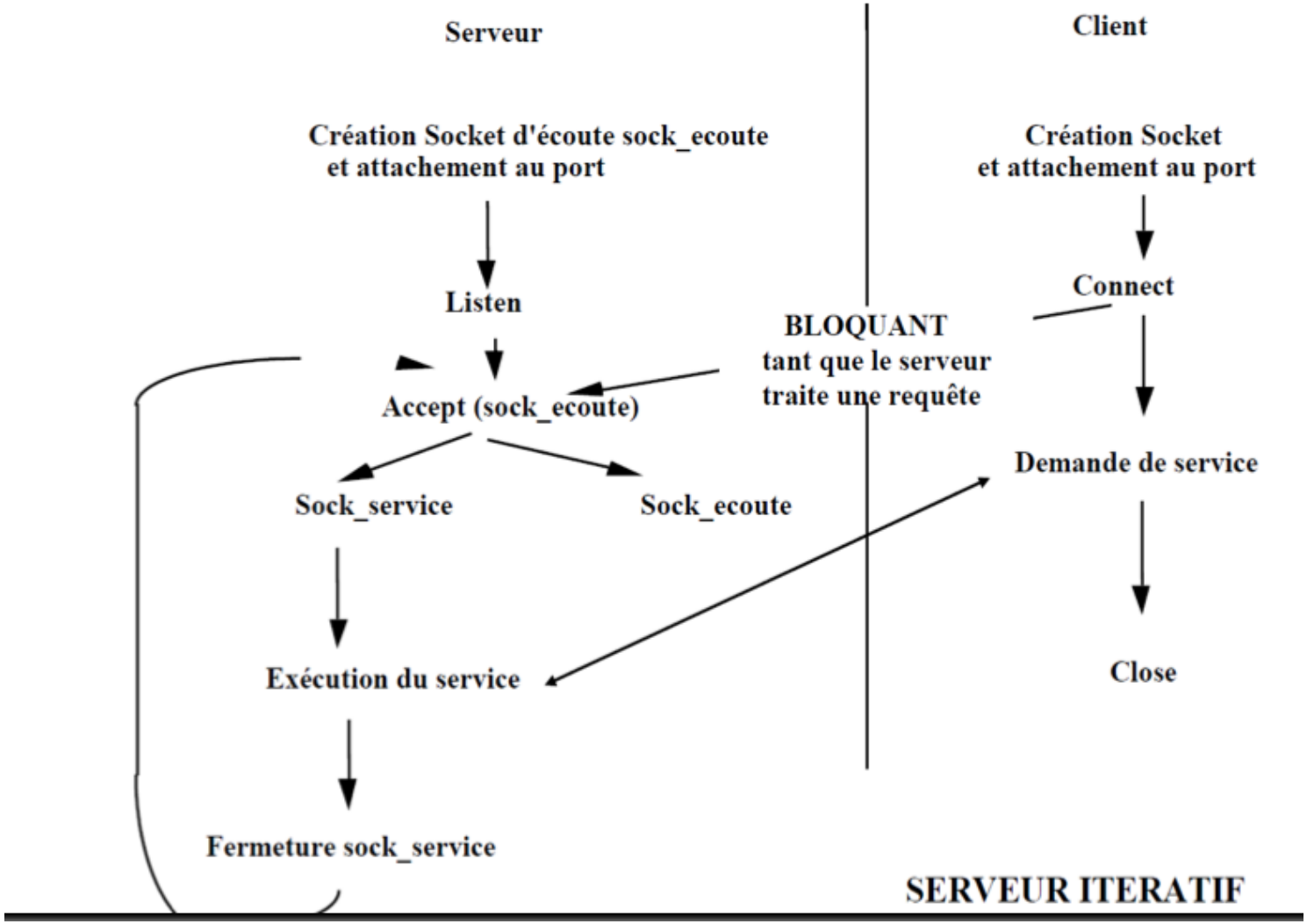
- Interface socket : Communication en TCP
  - `int` **`listen`** `(sock, nb);`
    - `int sock;` -- **socket d'écoute**
    - `int nb;` -- **connexions pendantes max**
  - `int` **`accept`** `(sock, p_adr, p_lgadr);` -- **retourne un descripteur socket de service**
    - `int sock;` -- **socket d'écoute du serveur**
    - `struct sockaddr_in *p_addr;` -- **@ socket connectée client**
    - `int *p_lgadr;` -- **taille de l'adresse**
  - `int` **`connect`** `(sock, p_adr, lgadr);`
    - `int sock;` -- **socket client**
    - `struct sockaddr_in *p_addr;` -- **@ socket du serveur avec lequel s'établit la connexion**
    - `int lgadr;` -- **taille de l'adresse**

Sockets

- Interface socket : Communication en TCP
  - `int` **`write`** `(sock, msg, lg);`
    - `int sock;` -- **socket locale**
    - `char *msg;` -- **@ mémoire de la zone contenant le message**
    - `int lg;` -- **taille du message en octets**
  - `int` **`read`** `(sock, msg, lg);`
    - `int sock;` -- **socket locale**
    - `char *msg;` -- **@ mémoire de la zone où recevoir le message**
    - `int lg;` -- **taille du message en octets**

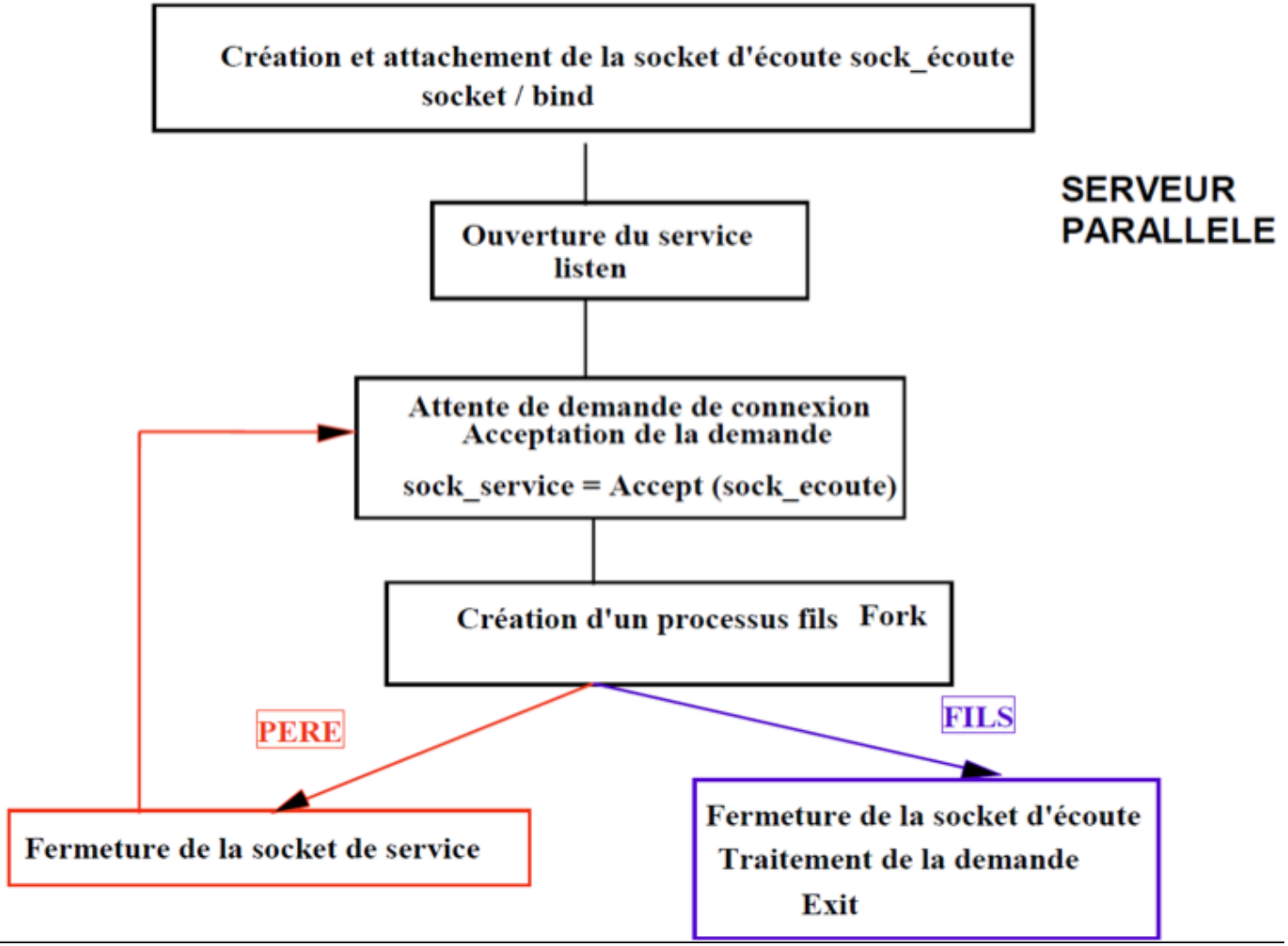
Sockets

- Interface socket : Communication en TCP



Sockets

- Interface socket : Communication en TCP



Sockets

```
#include <stdio.h> #include <stdlib.h>
#include <sys/types.h> #include <sys/socket.h>
#include <netinet/in.h> #include <netdb.h>
#include <string.h>    #define PORT 2058
int main() { // serveur
int sock, nsock, lg, n; pid_t pid; char buf[20]; struct sockaddr_in adr_s, adr_c;
sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); // Creation socket
bzero(&adr_s,sizeof(adr_s)); adr_s.sin_family = AF_INET; adr_s.sin_port = htons(PORT);
adr_s.sin_addr.s_addr = htonl(INADDR_ANY);
bind (sock, (struct sockaddr *) &adr_s, sizeof(adr_s)); listen(sock, 5);
while (1) { lg = sizeof(adr_c);
            nsock = accept(sock, (struct sockaddr *) &adr_c, &lg);
            pid = fork();
            if (pid==0) {close(sock); read(nsock,buf,20); close(nsock); printf("Message reçu : %s \n", buf);
                        exit(0);}
```

Sockets

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#define PORT 2058
int main() { // client
int sock; struct sockaddr_in adr_s, adr_c;
sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); // Creation socket
bzero(&adr_s,sizeof(adr_s)); adr_s.sin_family = AF_INET; adr_s.sin_port = htons(PORT);
adr_s.sin_addr.s_addr = inet_addr("192.168.0.12");
connect(sock, (struct sockaddr *) &adr_s, sizeof(adr_s));
write(sock, "bonjour, serveur !!!", 20);
close(sock);  }
```