Programmation réseau

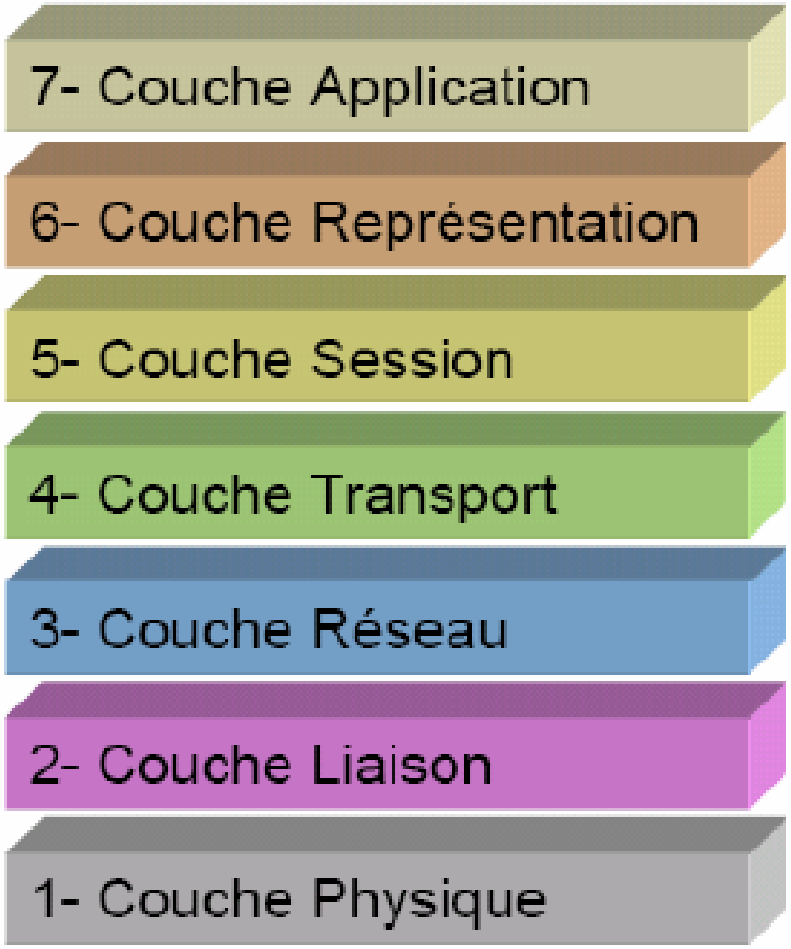
Les sockets

Préparé par : A. Begdouri
MST-SIR

Plan

- Introduction
- Sockets
  - Création, attachement, fermeture
- Sockets dans le domaine unix
  - Mode connecté
  - Mode non connecté
- Sockets dans le domaine internet
  - Mode connecté
  - Mode non connecté
  - Serveur parallèle

Le modèle OSI



Un ensemble de services standard : terminal virtuel, transfert de fichiers, ...

Codage des informations sous une forme standard

Gestion complète d 'une session de communications entre 2 utilisateurs.

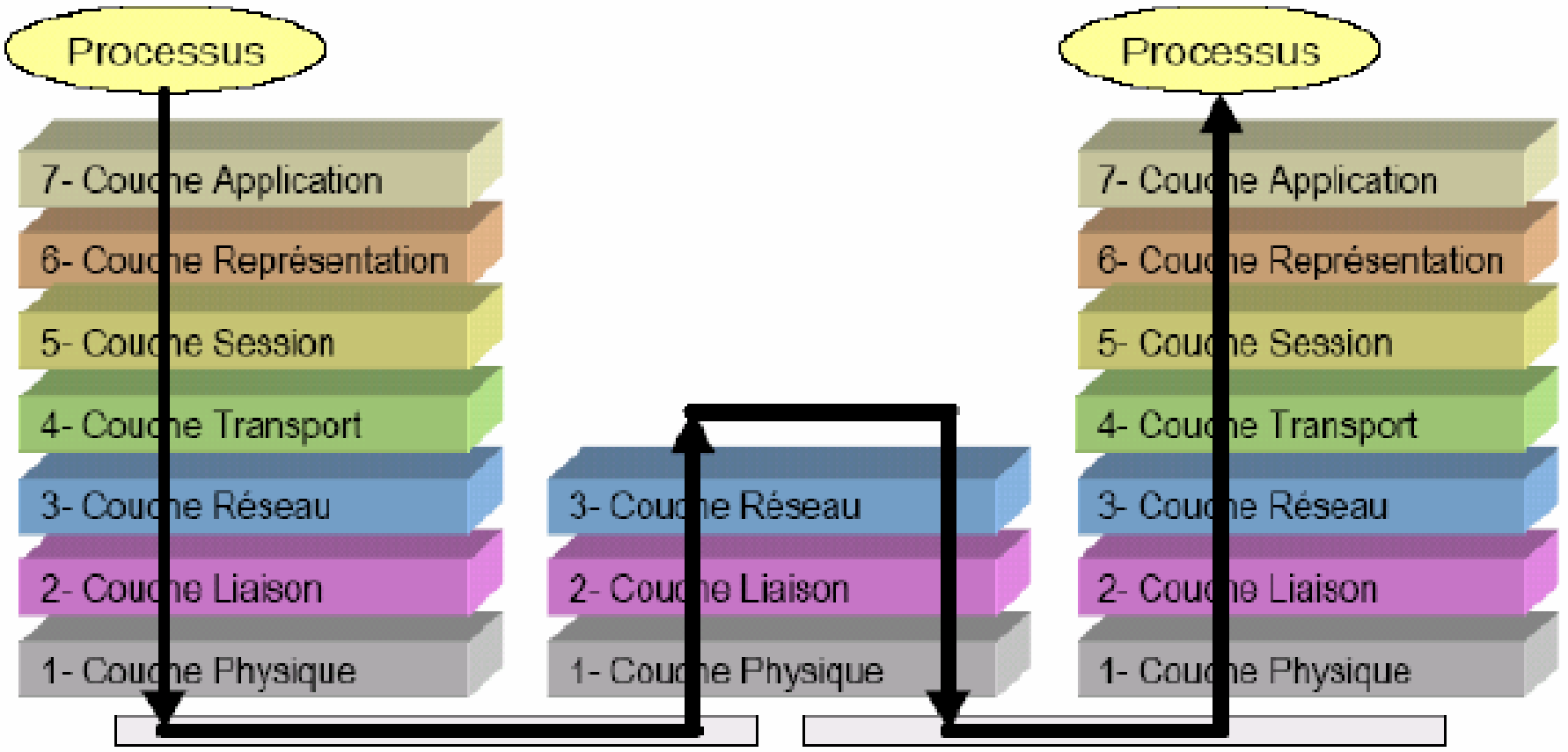
Transport de l 'information entre deux processus (Multiplexage)

Transport de l 'information entre deux points du réseau (Routage)

Transport, sans erreur entre deux points, des blocs de données (trames)

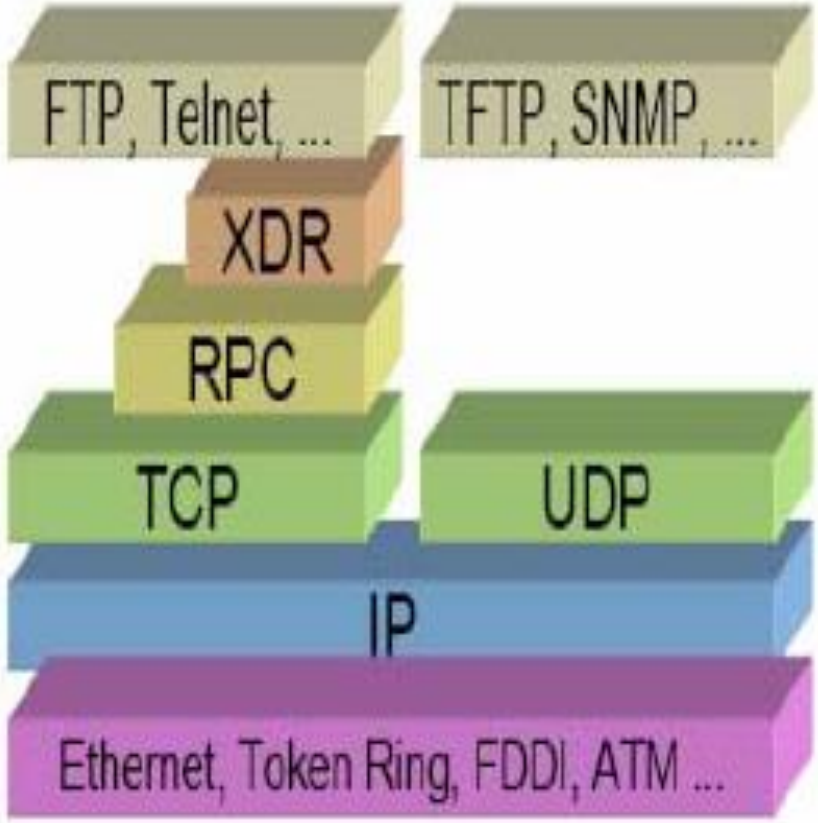
Transport de l 'information comme une suite de bits

Communication dans le modèle OSI



Routage entre plusieurs supports de communication

L'implémentation IP



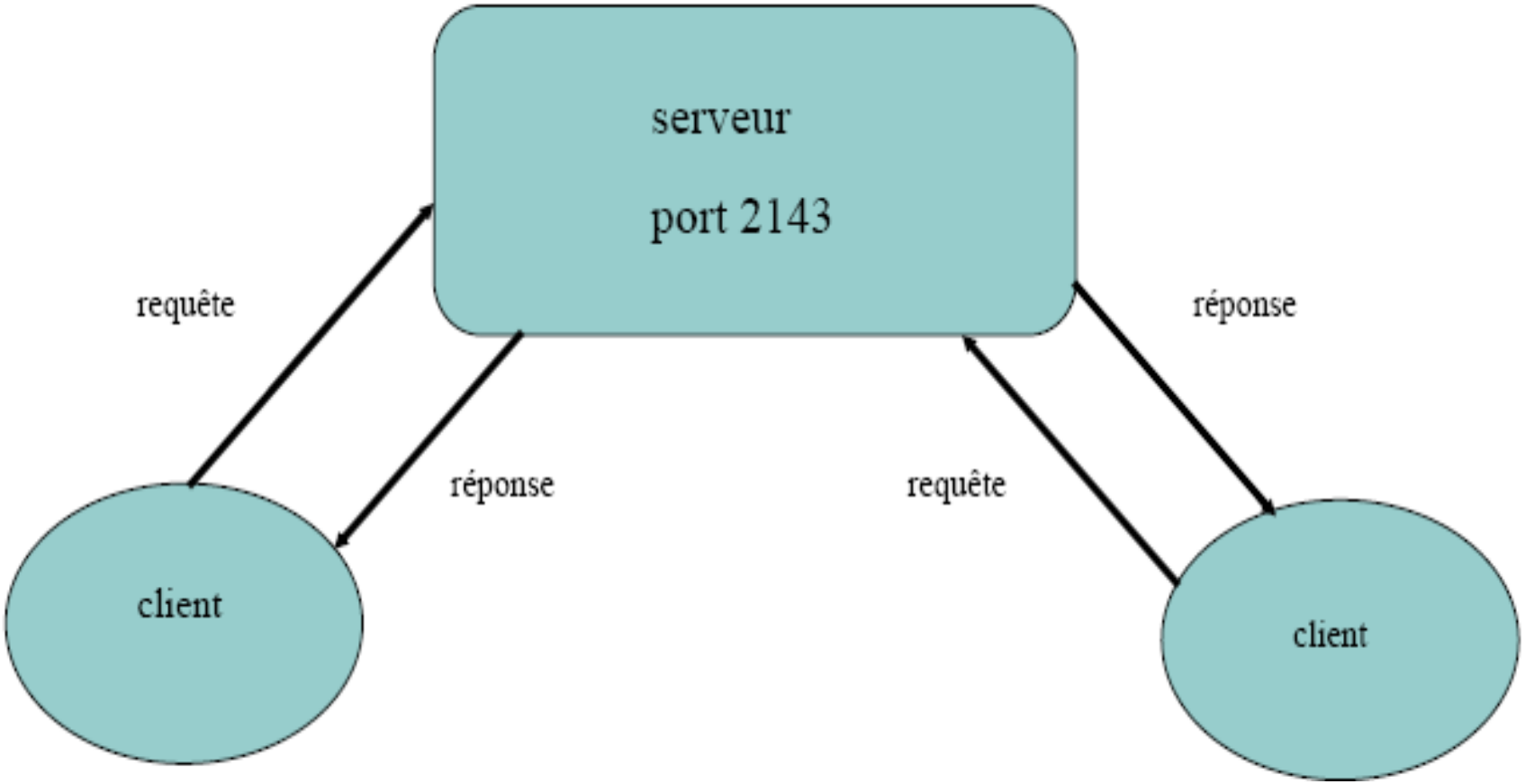
Architecture client /serveur

- Architecture d'applications réparties
- Principe :
  - Un serveur propose un ou plusieurs services (l'heure, transfert de fichiers, etc.)
  - Un ou plusieurs clients utilisent le service

Architecture client/serveur

- Serveur
  - processus rendant un service spécifique identifié
    - Par exemple, par un numéro de port particulier,
  - en attente sur une station identifiée par une adresse
    - Par exemple, adresse IP
- Client
  - processus appelant le serveur afin d'obtenir le service,
  - lancé à la demande à partir généralement de n'importe quelle station.
  - Émet une requête dans un protocole donné (TCP/IP, UDP/IP ou autre)
  - Précise l'adresse de la machine du serveur (@IP ou autre)
  - Précise le service souhaité (le port de communication dans le cas de service internet)

Architecture client/serveur



Socket : définition

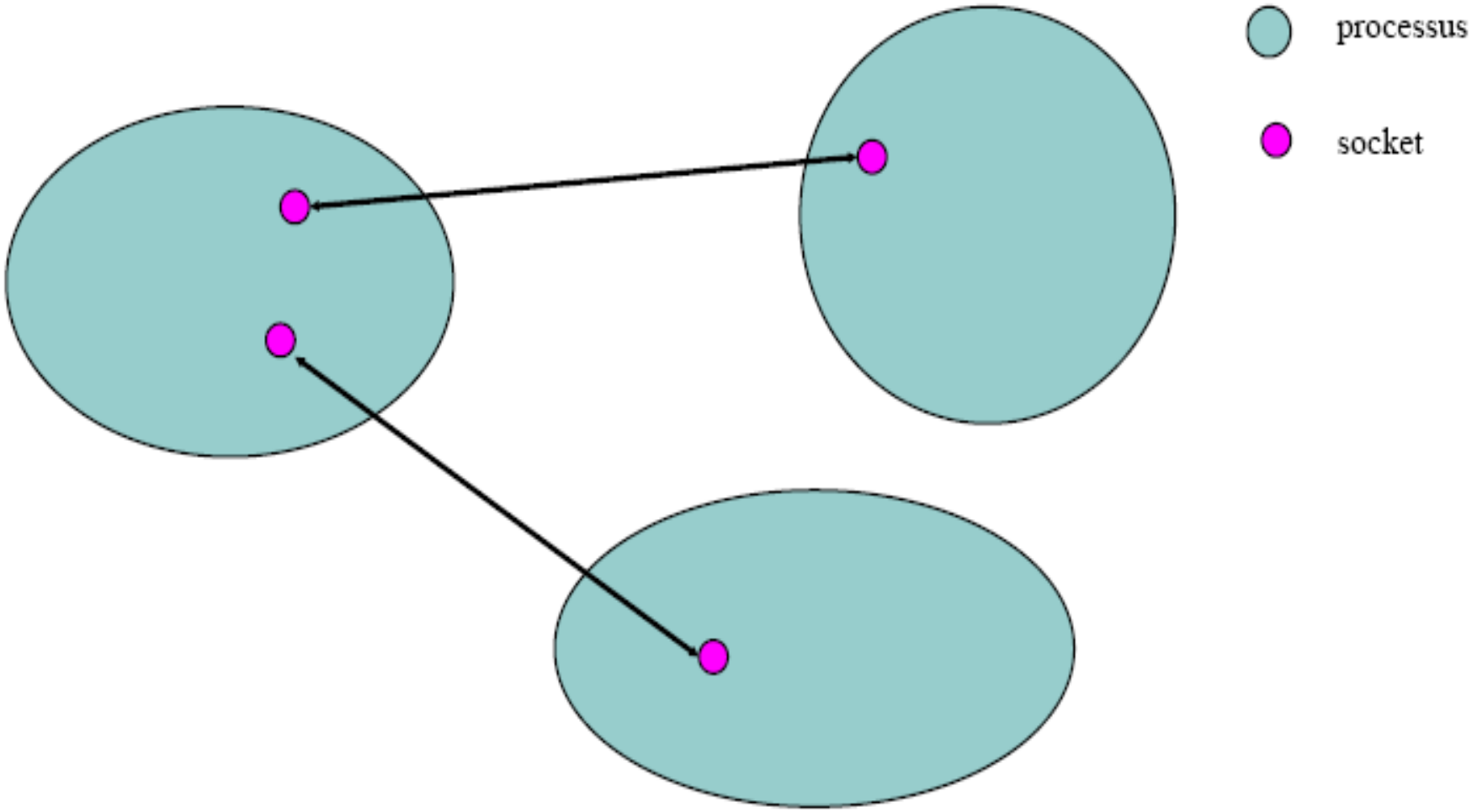
- Interface de programmation pour les communications
  - Ensemble de primitives assurant le service de communication,
  - Générique : s'adapte aux différents besoins de communication,
  - Indépendant des protocoles et des réseaux :
    - Mais développé à l'origine sous Unix 4BSD, pour Internet !
  - N'utilise pas forcément un réseau :
    - Programmation de communication locale (interne à une station) : domaine Unix !

Les sockets

- Interface de programmation proposée par l'unix de Berkeley
- Permettent des échanges bidirectionnels entre processus
- Une socket :
  - Point de communication par lequel un processus peut émettre ou recevoir des données
  - Dans le cas du domaine Internet:

    Un triplet (protocole, adresse, port)

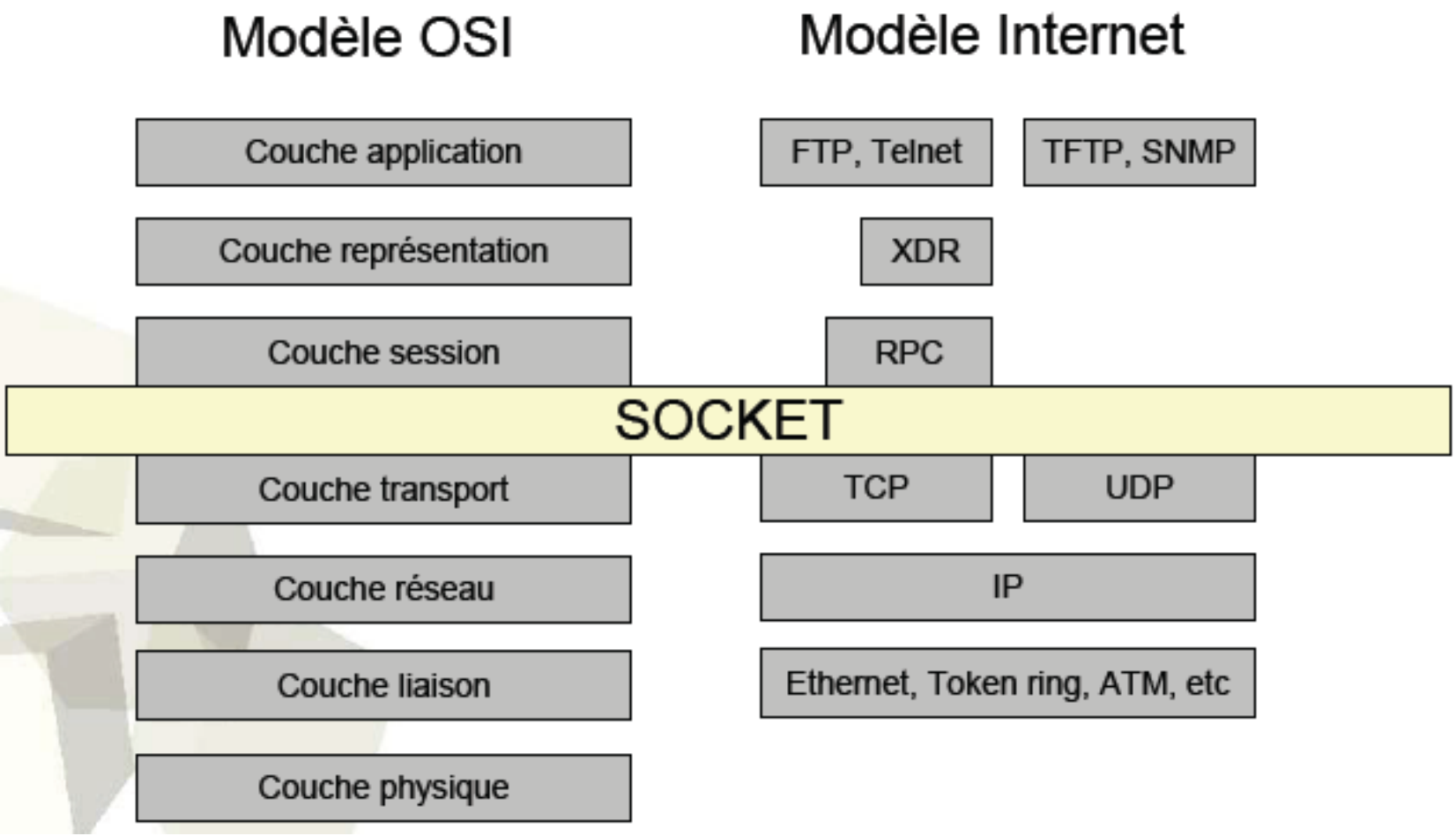
Processus / Socket



Socket

- Considérée comme un ***descripteur de fichier*** par le système d'exploitation
- À la différence
  - Socket:
    - on distingue la création de la socket : socket()
    - Du nommage: bind()
    - La connection: connect()
    - ….
  - Fichier:
    - Open() !

Programmation au niveau transport



Programmation sockets

- La librairie UNIX socket permet la programmation au niveau transport
  - Ensemble de primitives
- permet la gestion de 2 types de sockets:
  - « Datagram Socket » ➔ UDP
  - « Stream Socket » ➔ TCP

Création d'une socket

- *#include <sys/types.h>*

  #include <sys/socket.h>

  int socket(int domain, int type, int protocol);

- Retourne:
  - –1 en cas d'échec,
  - le descripteur de la socket en cas de réussite

Création d'une socket

int ***socket***(int *domain*, int *type*, int *protocol*);

descripteur de la socket, -1 si erreur

domaine de communication :
AF_UNIX, ou AF_INET, etc.
---
voir liste dans : **<sys/socket.h>**

mode de communication : SOCK_STREAM, SOCK_DGRAM, etc.
---
voir liste dans : **<sys/socket.h>**

protocole à utiliser, (0 => le système choisit le protocole le mieux adapté au mode).
voir dans : **<netinet/in.h>**

Arguments de la fonction socket

domain	**AF_UNIX** protocole local **AF_INET** protocole Internet **AF_INET6** protocole Internet IPv6
type	**SOCK_STREAM** Garantie l'intégrité de la transmission. **SOCK_DGRAM** Transmission sans connexion, sans garantie de datagrammes de longueur fixe. **SOCK_RAW** programmation de niveau 3, administrateur seulement
Protocol	**IPPROTO_TCP, IPPROTO_UDP, IPPROTO_RAW** En général, il y a un protocole par type et il n'est pas utile de le spécifier. S'il y en a plusieurs, on peut l'indiquer.

Exemples de création de sockets

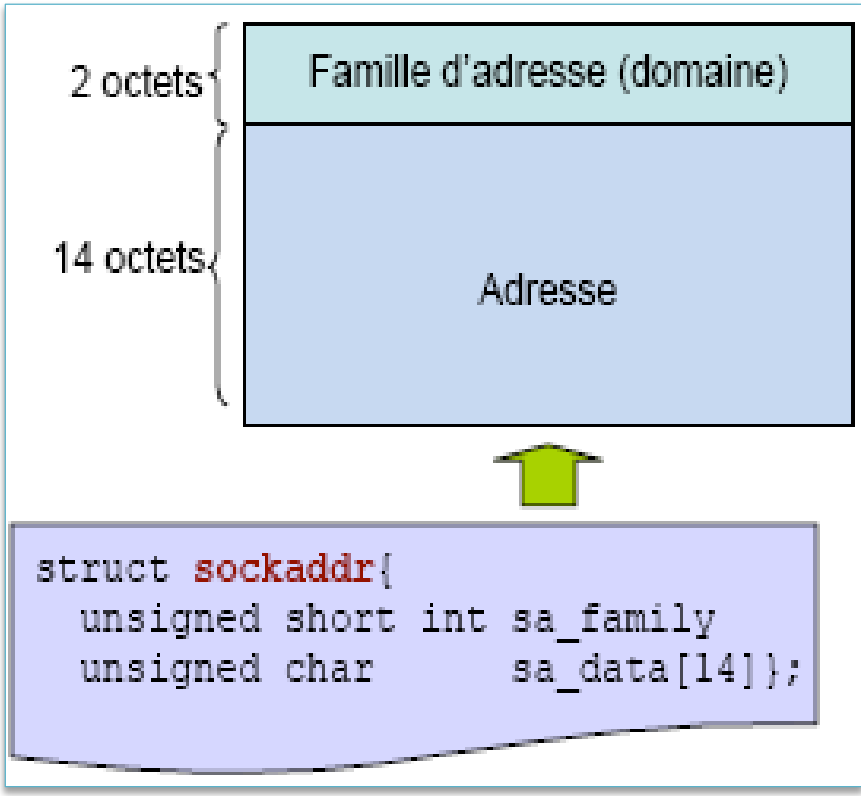
- Socket internet sans connexion (UDP/IP)
  - socket (AF_INET,SOCK_DGRAM,IPPROTO_UDP)
  - socket (AF_INET,SOCK_DGRAM,0)
- Socket internet avec connexion (TCP/IP)
  - socket (AF_INET,SOCK_STREAM,IPPROTO_TCP)
  - socket (AF_INET,SOCK_STREAM,0)
- Socket unix sans connexion
  - socket (AF_UNIX,SOCK_DGRAM,0)

Adresse d'attachement d'une socket

- Chaque socket créée doit s'attacher à une adresse sur laquelle elle écoute les données entrantes
- La structure d'adresse dépend du domaine de la socket
  - Domaine unix:
    - l'adresse est une référence unix
    - Le pathname de la socket
  - Domaine internet:
    - L'adresse est sous forme d'un couple (numéro de port, adresse IP)

Adressage : structure sockaddr

- Fournit un type générique compatible avec les types spécifiques (AF_UNIX, AF_INET, etc.)
- Toutes les primitives de manipulation des sockets sont génériques
  - Elles prennent en paramètre le type sockaddr
  - Il faut penser à faire « le casting »



Adressage: Structure sockaddr

- On manipule des structures d'adresses dont le type générique est: *struct sockaddr*
- Les adresses manipulées par toutes les primitives sont prototypées avec ce type
- En fonction du domaine, on utilisera des structures d'adresses particulières:
  - *struct sockaddr_un :* Dans le domaine Unix ➔ AF_UNIX
  - *struct sockaddr_in :* Dans le domaine Internet ➔ AF_INET

Attachement socket / adresse

- Après sa création, la socket n'est connue que du processus qui l'a créé (et de ses descendants)
- Elle doit être assignée par une adresse pour pouvoir être contactée de l'extérieur : par d'autres processus locaux ou distants

Int bind (sock, p_adresse, lg)

```
Int sock ;                          /*descripteur de socket*/
Struct sockaddr * p_adresse ;       /*pointeur en mémoire sur l'adresse*/
Int lg ;                            /*longueur de l'adresse*/
```

- Associe l'adresse locale ***adresse*** à la socket sock.
- Le paramètre ***lg*** est la taille de la structure adresse.
- Retourne -1 en cas d'erreur, 0 sinon

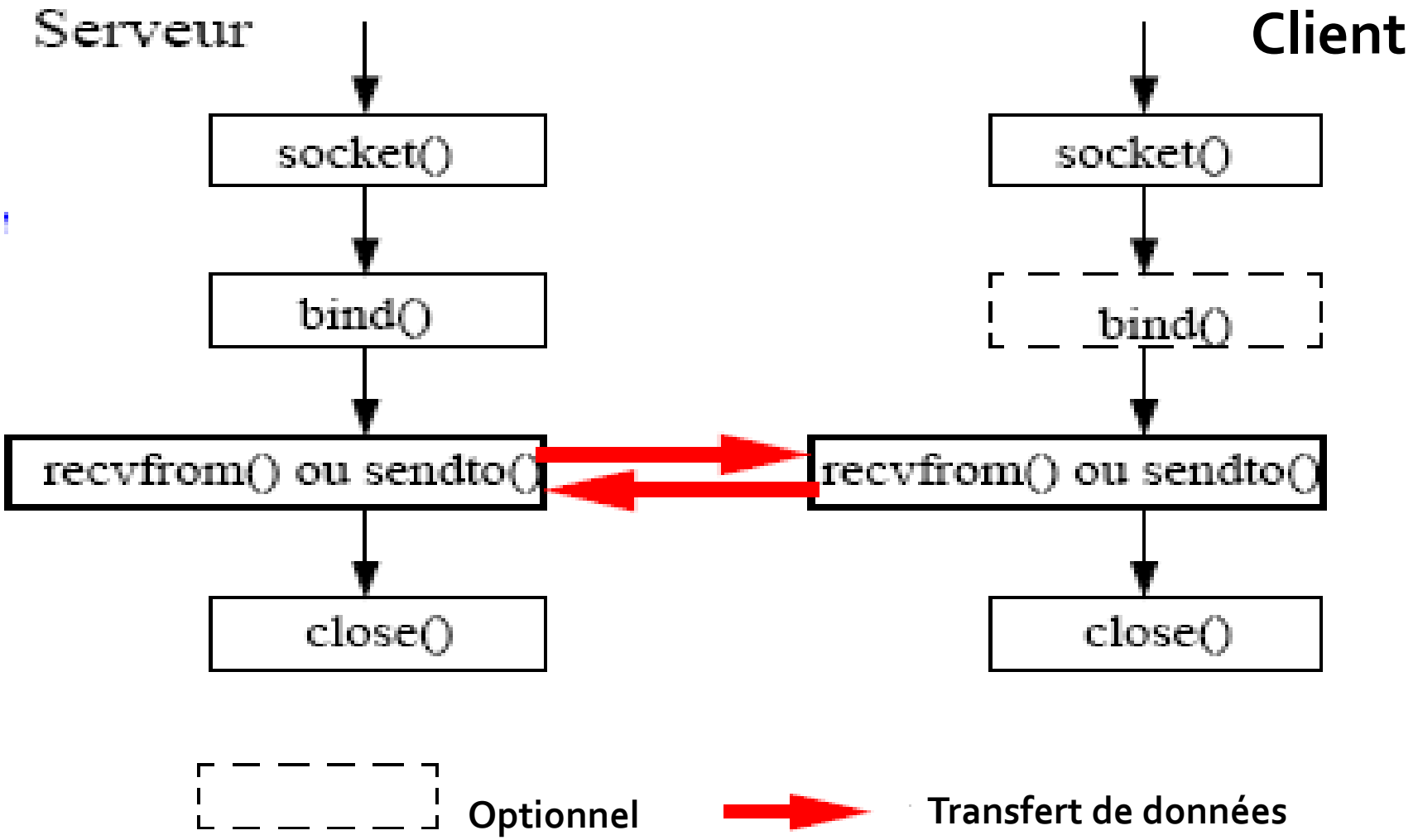
Fermeture d'une socket

- libération des entrées dans les différentes tables et des tampons alloués par le système en relation avec la socket

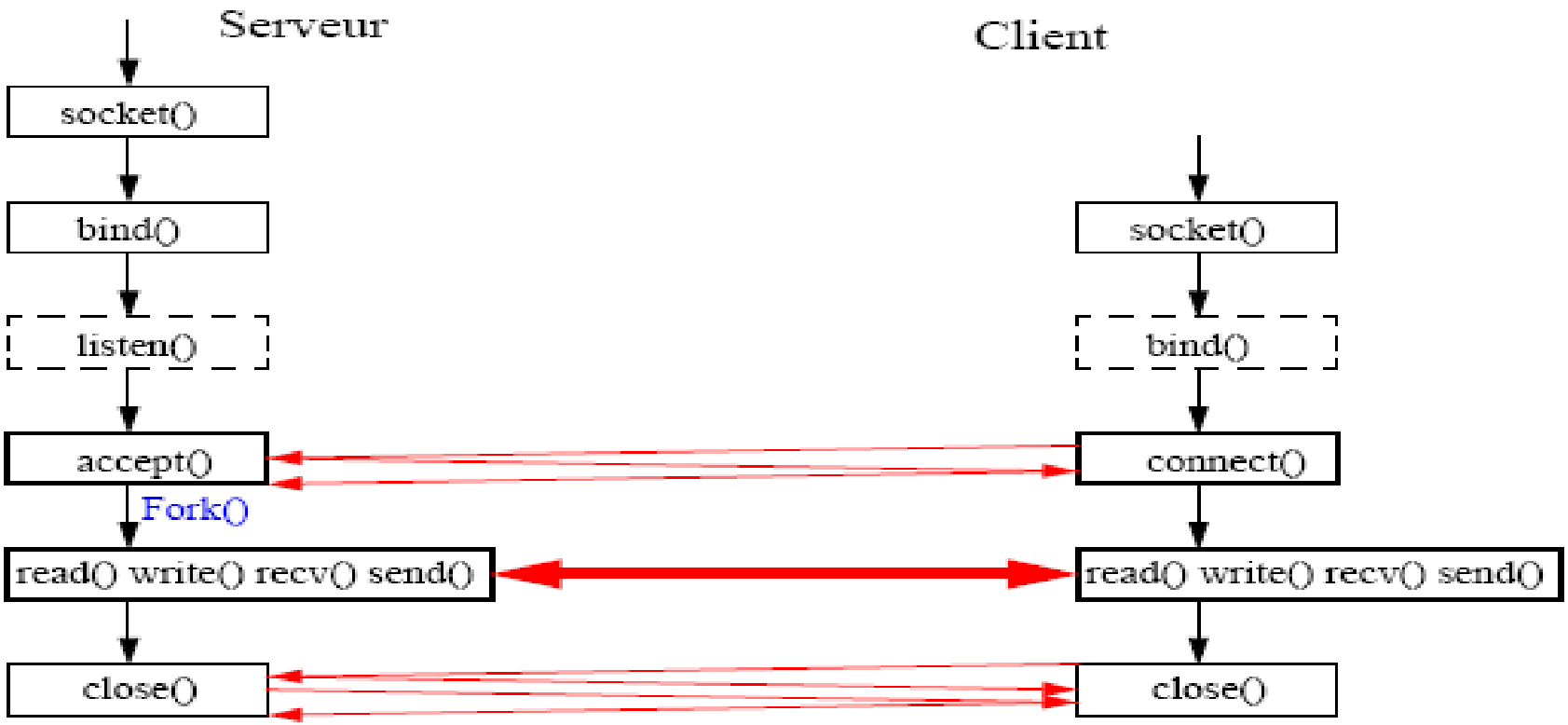
int close(int fd);

Retourne 0 ou –1 en cas d'échec

Enchainement des primitives : Le mode non connecté



Enchainement des primitives : Le mode connecté (TCP)



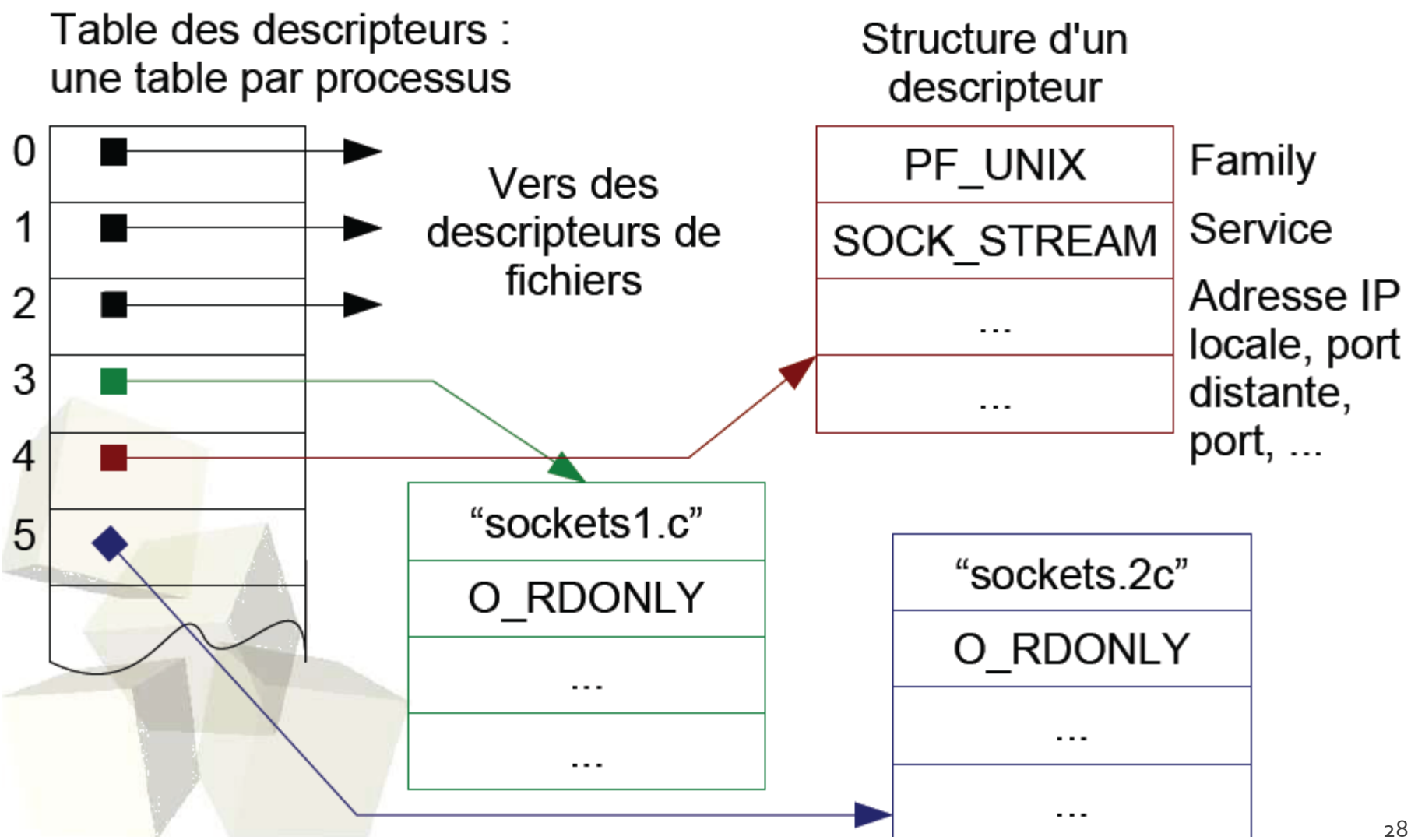
Chaque close() ne ferme qu'un seul sens de la communication

Le domaine Unix : AF_UNIX

TP : envoi de message

- Objectif:
- Écrire les programmes serveur et client dans le ***domaine unix*** en ***mode non connecté*** permettant:
  - au client d'envoyer au serveur une chaine de caractères (par exemple: "Bonjour")
  - Au serveur de répondre en renvoyant la même chaine de caractères.

Descripteur de socket



La structure sockaddr_un

```
#include<sys/un.h>
Struct sockaddr_un
{
    Short       sun_family ;        /*domaine unix: AF_UNIX*/
    Char        sun_path[108];      /*reference unix*/

};
```

La structure sockaddr_un

- Type associé aux adresses dans le domaine Unix
- Sun_path est le chemin pour accéder à la socket

```
#include<sys/un.h>
Struct sockaddr_un
{
        Short           sun_family ;        /*domaine unix: AF_UNIX*/
        Char            sun_path[108];      /*reference unix*/

};
```

Le mode non connecté

- Point de vue du serveur
  - création et attachement de la socket (socket et bind)
  - une boucle infinie dans laquelle:
    - Il attend une requête,
    - traite la requête,
    - met la réponse en forme,
    - et envoie la réponse.
- Point de vue du client
  - Création, éventuellement attachement, de la socket locale
  - envoi du message ;
  - attente du résultat ;
  - exploitation du résultat

Primitive d'envoi de données mode non connecté

- sendto

```
Int sendto (sock, msg, lg, option, p_dest, lgdest)

        Int  sock ;              /* descripteur de la socket d'émission*/
        Char  *msg ;             /* adresse du message à envoyer*/
        Int  lg ;                /* longueur du message*/
        Int  option ;            /* = 0 pour le type SOCK_DGRAM*/
        Struct sockaddr *p_dest ;        /* pointeur sur adresse socket destination*/
        Int lgdest ;                     /* longueur de l'adresse de la socket destinataire*/
```

Primitive de réception de données mode non connecté

- recvfrom

```
Int  recvfrom ( sock, msg, lg, option, p_exp, p_lgexp)

        Int sock;                       /* descripteur de la socket de réception*/
        Char *msg ;                     /* adresse de récupération du message reçu*/
        Int lg ;                        /* taille de l'espace alloué à l'adresse msg*/
        Int option ;                     /* 0 ou MSG_PEEK*/
        Struct sockaddr * p_exp ;       /* pour récupérer l'adresse d'expédition*/
        Int *p_lgexp ;                  /* taille de l'espace réservé à p_exp et longueur
                                           du résultat*/
```

TP: étape 1

- Ecrire un programme serveur qui :
  - Crée sa socket
  - Attache sa socket à une adresse
  - Affiche: " Serveur prêt à recevoir des données! "

- Localisez la socket créée dans l'arborescence linux?
- Elle est répertoriée sous quel type?

TP: étape 2

- Ecrire un programme client qui:
  - Crée sa socket
  - Affiche: " Client prêt à envoyer des données! «
- Quelles sont les étapes qui manquent chez le client et le serveur pour que :
  - le client puisse envoyer des données?
  - le serveur puisse les recevoir?
- Améliorez les programmes client et serveur

TP: Amélioration: échange de messages

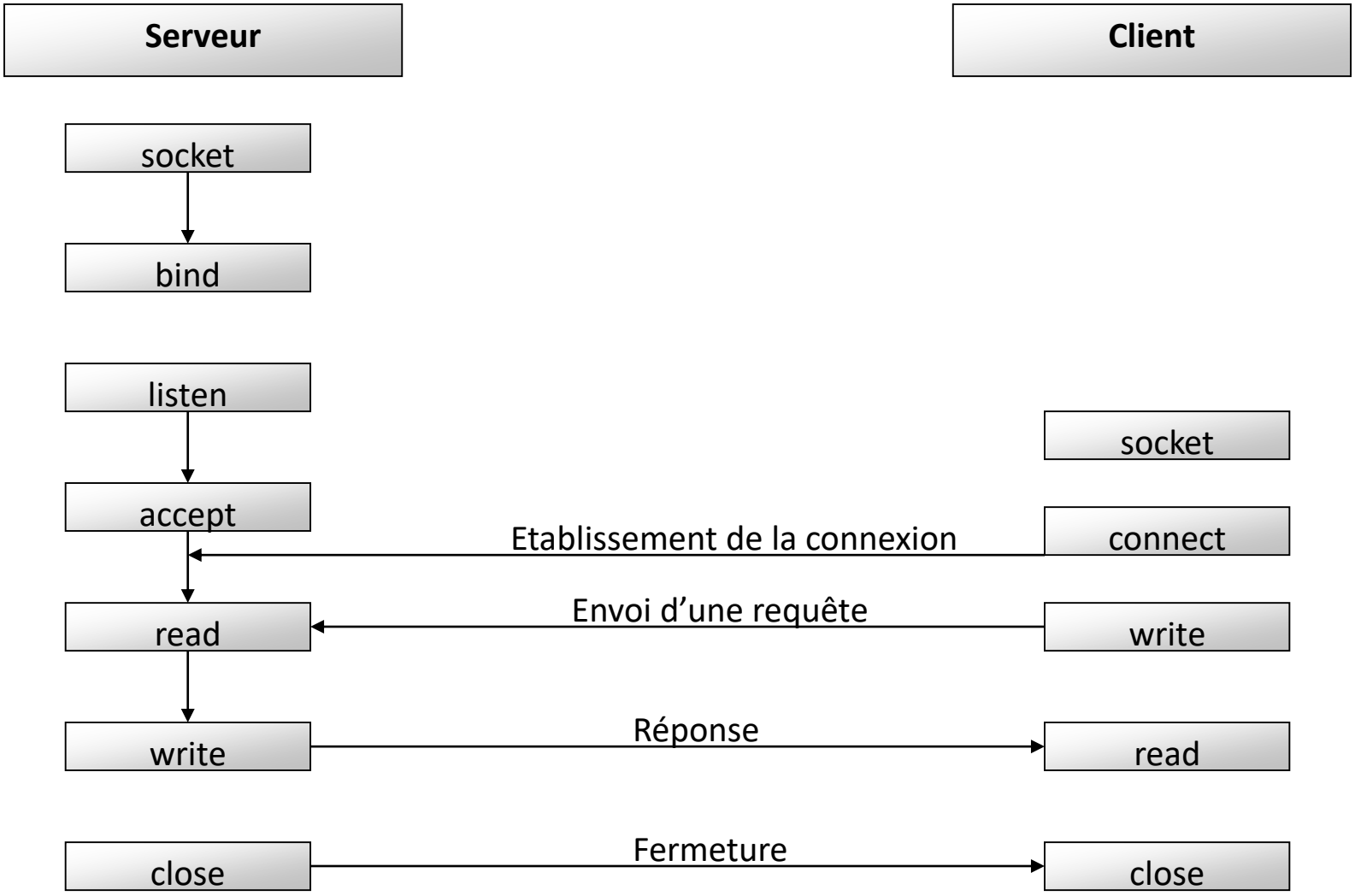
- Lorsque le serveur veut envoyer son message vers le client de qui il vient de recevoir un message, est ce que c'est possible?
- Est-ce qu'il connait l'adresse de la socket client?
  - Le client doit avoir fait auparavant un attachement de sa socket locale pour que le serveur puisse récupérer l'adresse du client dans son appel à recvfrom

Le mode connecté
Domaine Unix

Le mode connecté

- Point de vue du serveur
  - création et attachement de la socket (socket et bind)
  - Se mettre à l'écoute de connexions entrantes (listen)
  - Acceptation d'une connexion (accept)
  - Lecture et écriture sur la socket:
    - Il attend une requête du client,
    - traite la requête,
    - met la réponse en forme,
    - et envoie la réponse.
- Point de vue du client
  - Création, éventuellement attachement, de la socket locale
  - Connexion au serveur (connect)
  - envoi du message ;
  - attente du résultat ;
  - exploitation du résultat

Communication en mode connecté



La primitive d'écoute: listen

- Attendre des connexions sur une socket
- Retourne -1 en cas d'échec, 0 sinon

```
Int  listen (sock, nb)

        Int  sock ;          /* descripteur de la socket d'écoute*/
        Int  nb ;            /*nombre maximal de demandes de connexions en attente*/
```

La primitive de connexion

- Demande d'une pseudo-connexion sur une socket distante
- Retourne -1 en cas d'échec, 0 sinon

Int connect (sock, p_adr, lgadr)

```
Int  sock ;                    /* descripteur de la socket locale*/
Struct sockaddr  *p_adr ;      /* adresse de la socket distante*/
Int  lgadr ;                   /* longueur adresse distante*/
```

La primitive d'acceptation de connexion

- Attente bloquante d'une demande de connexion (chez le serveur)
- Si une demande de connexion vient d'un client, une nouvelle socket est créée, appelée socket de service

```
Int  accept (sock, p_adr, p_lgadr)
        Int  sock ;                    /* descripteur de la socket*/
        Struct  sockaddr  *p_adr ;     /* adresse de la socket connectée*/
        Int  *p_lgadr ;                /*pointeur sur la taille  de la zone allouée à p_adr*/
```

- Retourne:
  - -1 en cas d'échec
  - Le descripteur de la nouvelle socket en cas de réussite
  - L'adresse de la socket du client avec laquelle la connexion est établie
    - Dans la zone pointée par p_adr

Primitive d'envoi de données mode connecté

- Primitive write

```
Int  write  (sock, msg, lg)

        Int sock;                   /* descripteur de la socket locale*/
        Char  *msg ;                /* adresse en mémoire du message à envoyer*/
        Int  lg ;                   /* longueur du message*/
```

Primitive d'envoi de données mode connecté

- La primitive send
- Le fonctionnement est le même que celui de write, sauf qu'elle permet d'utiliser des options de transfert

```
Int send  (sock, msg, lg, option)

        Int sock ;          /* descripteur de la socket locale*/
        Char *msg ;         /* adresse en mémoire du message à envoyer*/
        Int lg ;            /* longueur du message*/
        Int  option ;       /* 0 ou MSG_OOB*/
```

Primitive de lecture de données mode connecté

- La primitive read

```
Int  read (sock, msg, lg)

        Int sock;          /* descripteur de la socket locale*/
        Char  *msg ;       /* adresse en mémoire de sauvegarde du message*/
        Int  lg ;          /* longueur de la zone allouée à l'adresse msg*/
```

Primitive de lecture de données mode connecté

- La primitive recv
- Le fonctionnement est le même que celui de read, sauf qu'elle permet d'utiliser des options de transfert

```
Int recv (sock, msg, lg, option)

        Int sock ;          /* descripteur de la socket locale*/
        Char *msg ;         /* adresse en mémoire de sauvegarde du message*/
        Int lg ;            /* longueur de la zone allouée à l'adresse msg*/
        Int  option ;       /* 0 ou MSG_PEEK ou MSG_OOB*/
```

TP2: échange de messages en mode connecté

- Réécrire les programmes client et serveur permettant l'échange de messages dans le domaine unix en ***mode connecté***

TP2: Étape 1

- Écrire le programme serveur qui:
  - Crée sa socket
  - Remplit sa structure d'adresse locale
  - Attache sa socket à cette adresse
  - Affiche le message :

    « Serveur prêt: attend qu'un client se connecte »
  - Reçoit une demande de connexion et affiche:

    « un client vient de se connecter »
  - Reçoit le message de ce client et l'affiche sur son écran

TP2: étape 2

- Écrire le programme client qui:
  - Crée sa socket
  - Localise le serveur (Remplit la structure d'adresse du serveur)

    Et affiche: « tentative de connexion au serveur »
  - Se connecte au serveur et affiche:

    « connexion établie avec le serveur »
  - Envoie au serveur le message du client

TP2: étape 3

- Le serveur envoie son message après réception de celui du client.
- Le client affiche le message du serveur sur son écran.