

Communication inter-processus par sockets

Armand Toguyéni

Professeur des Universités

Ecole Centrale de Lille

Bur. C341 Tel . 03-20-33-54-49

mel. : armand.toguyeni@centralelille.fr

Introduction

Introduction au modèle TCP/IP

Notion de Socket

domaine, protocole, adressage

Modèles de communication

DATAGRAM & STREAM

Sockets dans le domaine AF_UNIX

Mode datagram

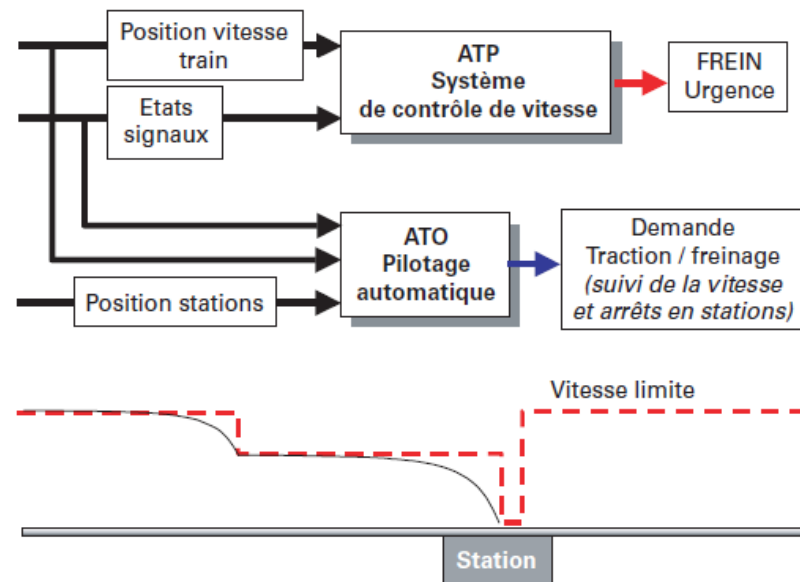
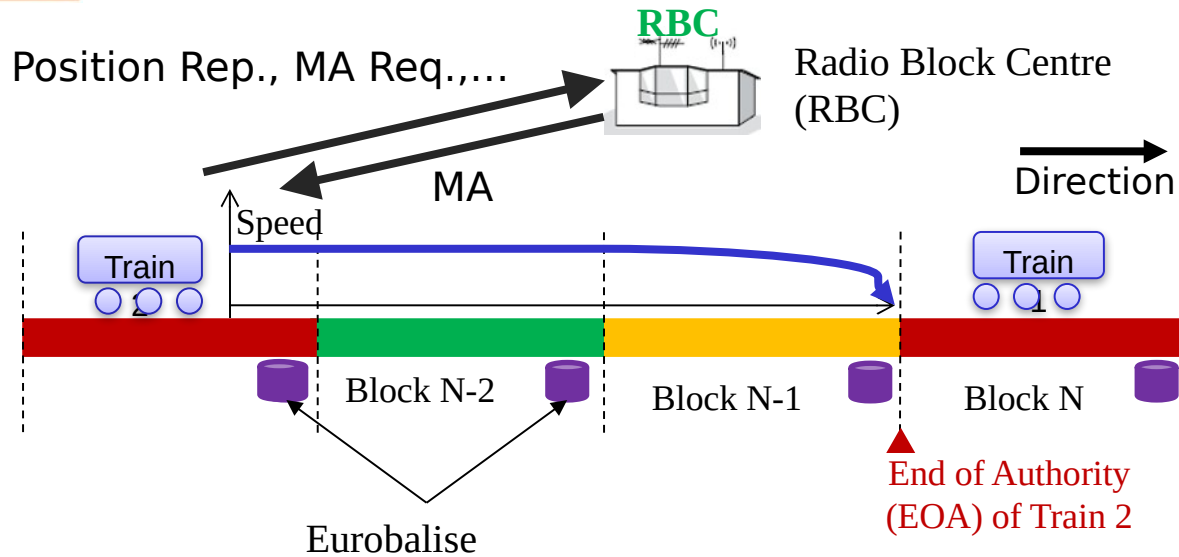
Mode stream

Sockets dans le domaine AF_INET

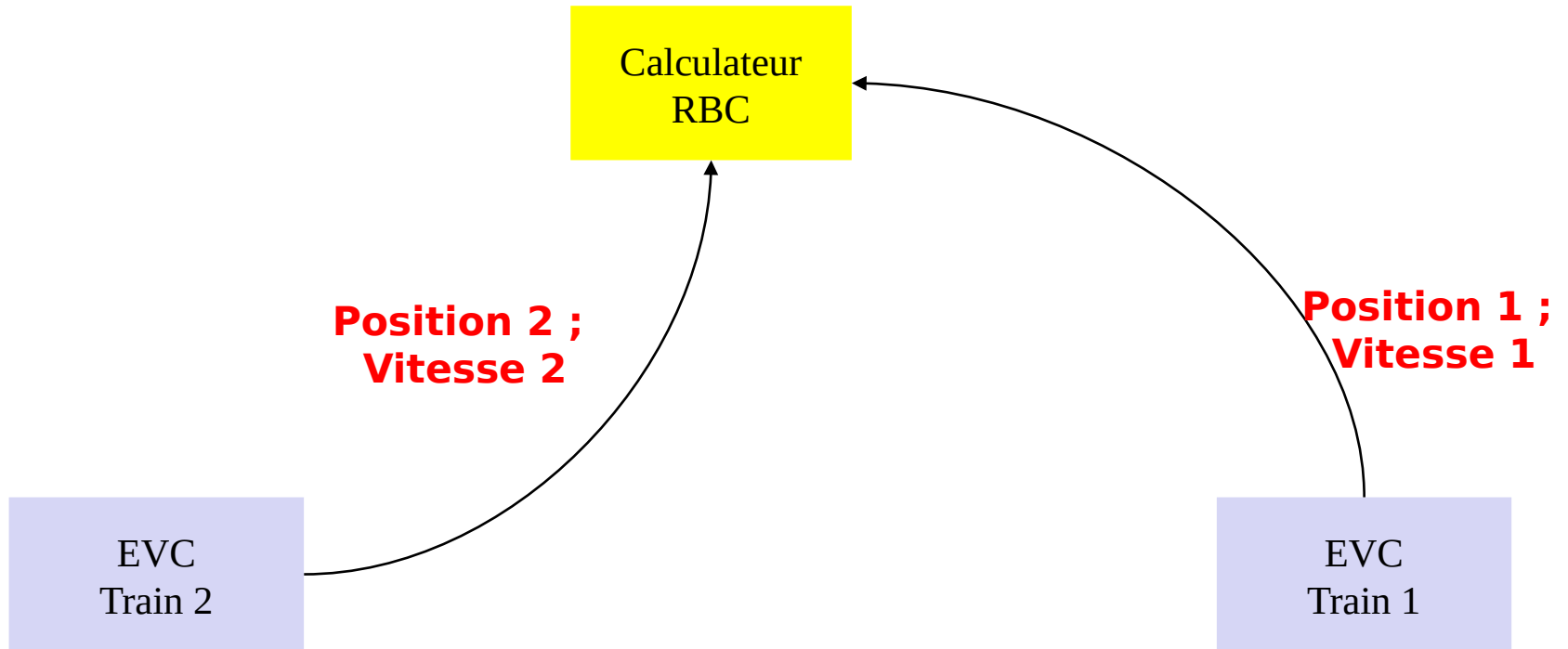
Mode datagram

Mode stream

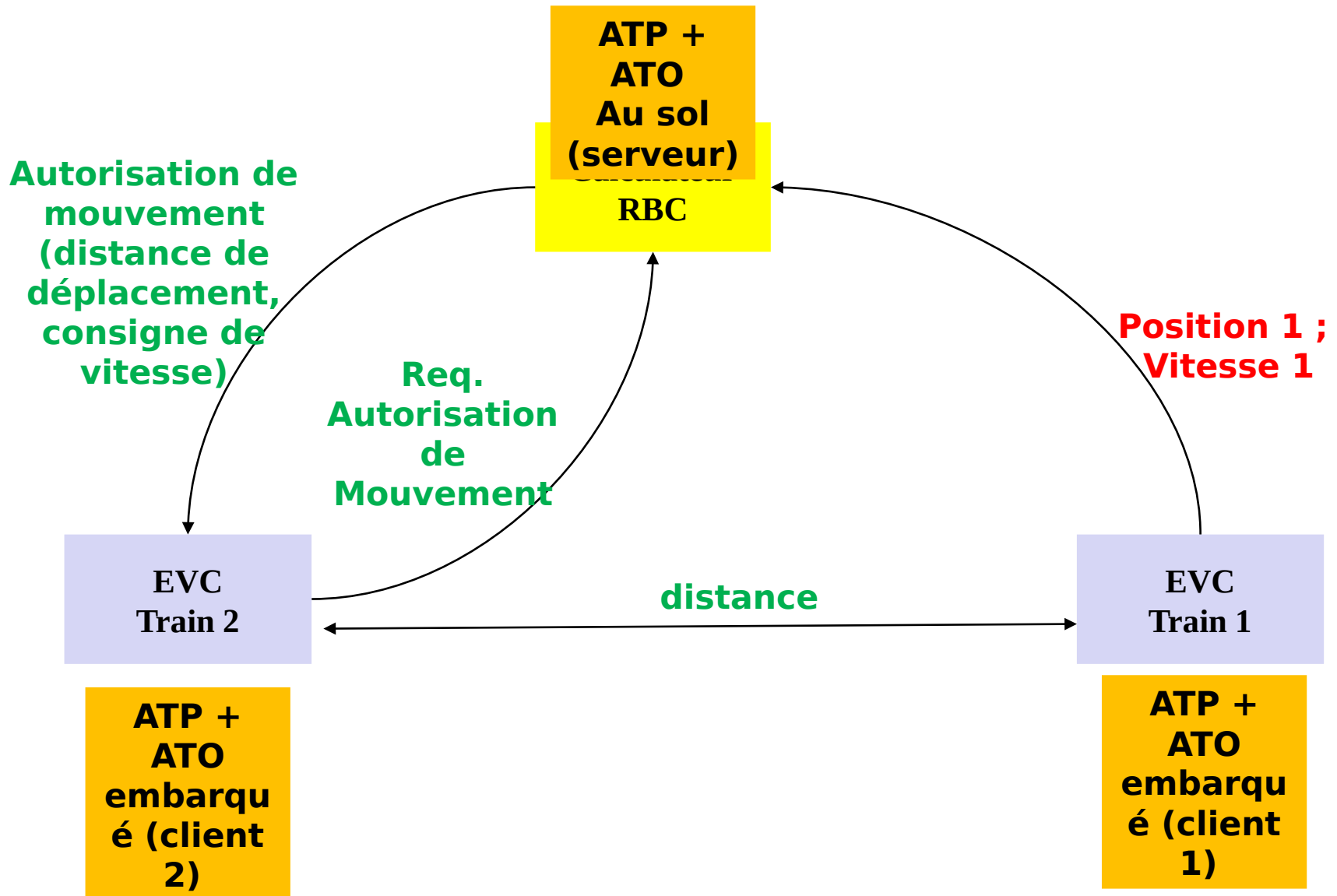
Introduction (1)



Introduction (2) : abstraction



Introduction (3) : abstraction



Introduction (4) : synthèse

□ Réalisation d'une application distribuée

◆ Serveur (RBC)

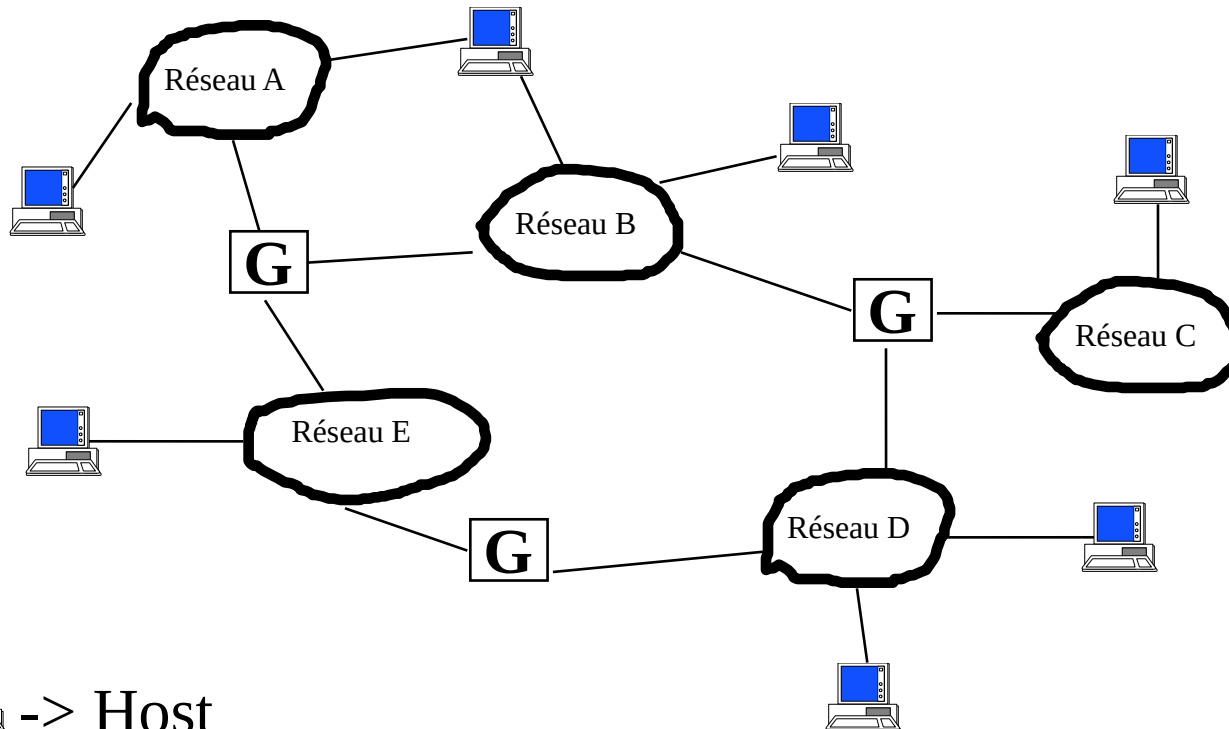
- ◆ Enregistrements de la position et de la vitesse de chaque train
- ◆ Calcule de l'autorisation de mouvement d'un train
- ◆ Envoi de l'autorisation au train concerné


◆ Client (Train)

- ◆ Caractérisé par un numéro de train
- ◆ Connait sa position et sa vitesse
- ◆ Envoi au RBC ses paramètres de localisation
- ◆ Demande une autorisation de mouvement
- ◆ Application de l'autorisation de mouvement

□ Besoins : Utilisation de sockets TCP/IP

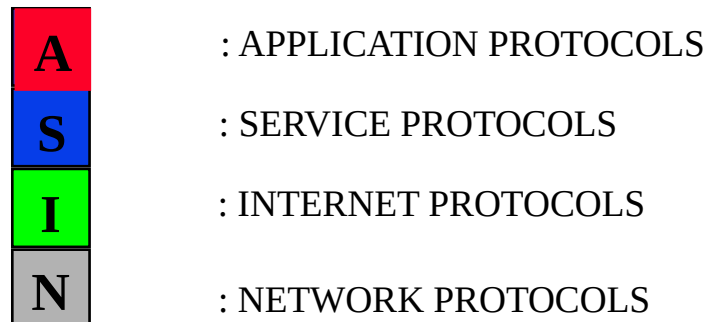
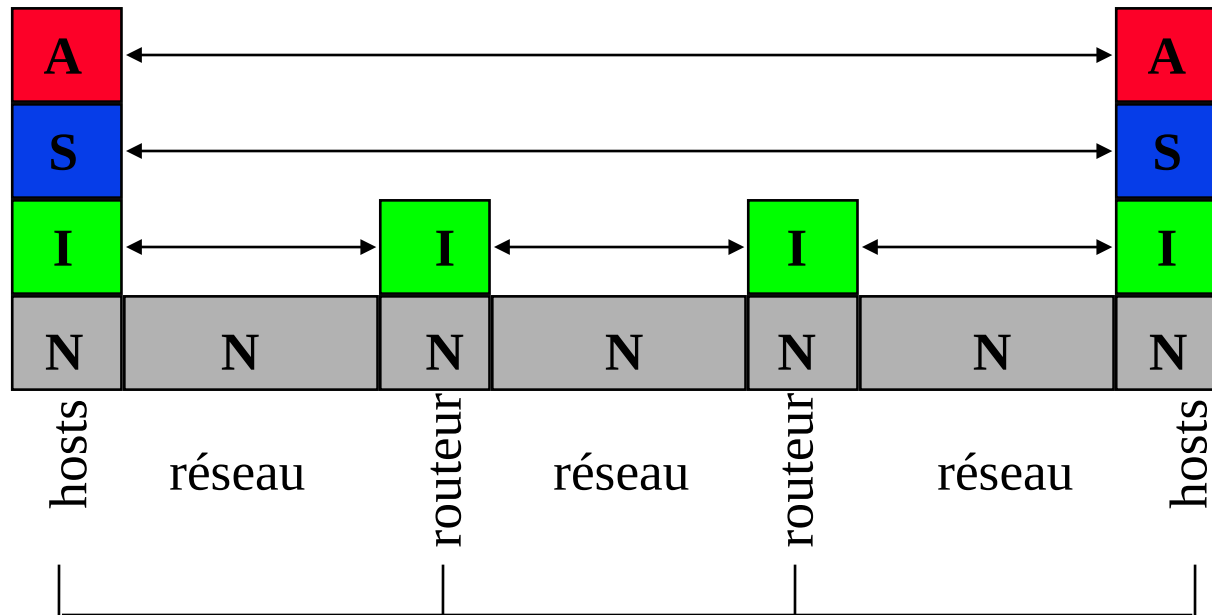
- INTERNET=RESEAU DE RESEAUX (pas de limite géographique)



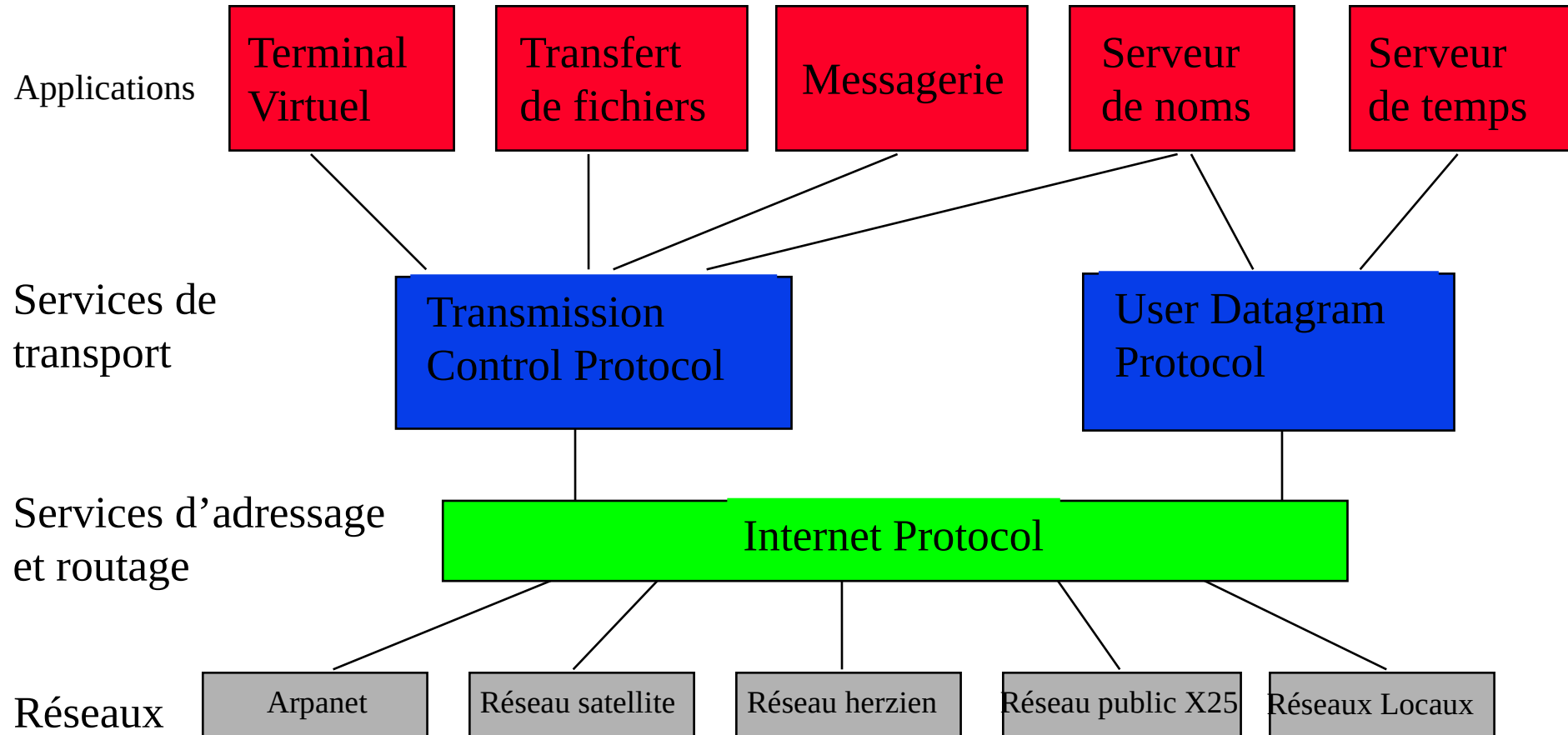
 -> Host

 -> Gateway (Routeur)

COUCHES DE L'INTERNET



COMPOSANTS DES COUCHES



COMPARAISON INTERNET/MODELE OSI

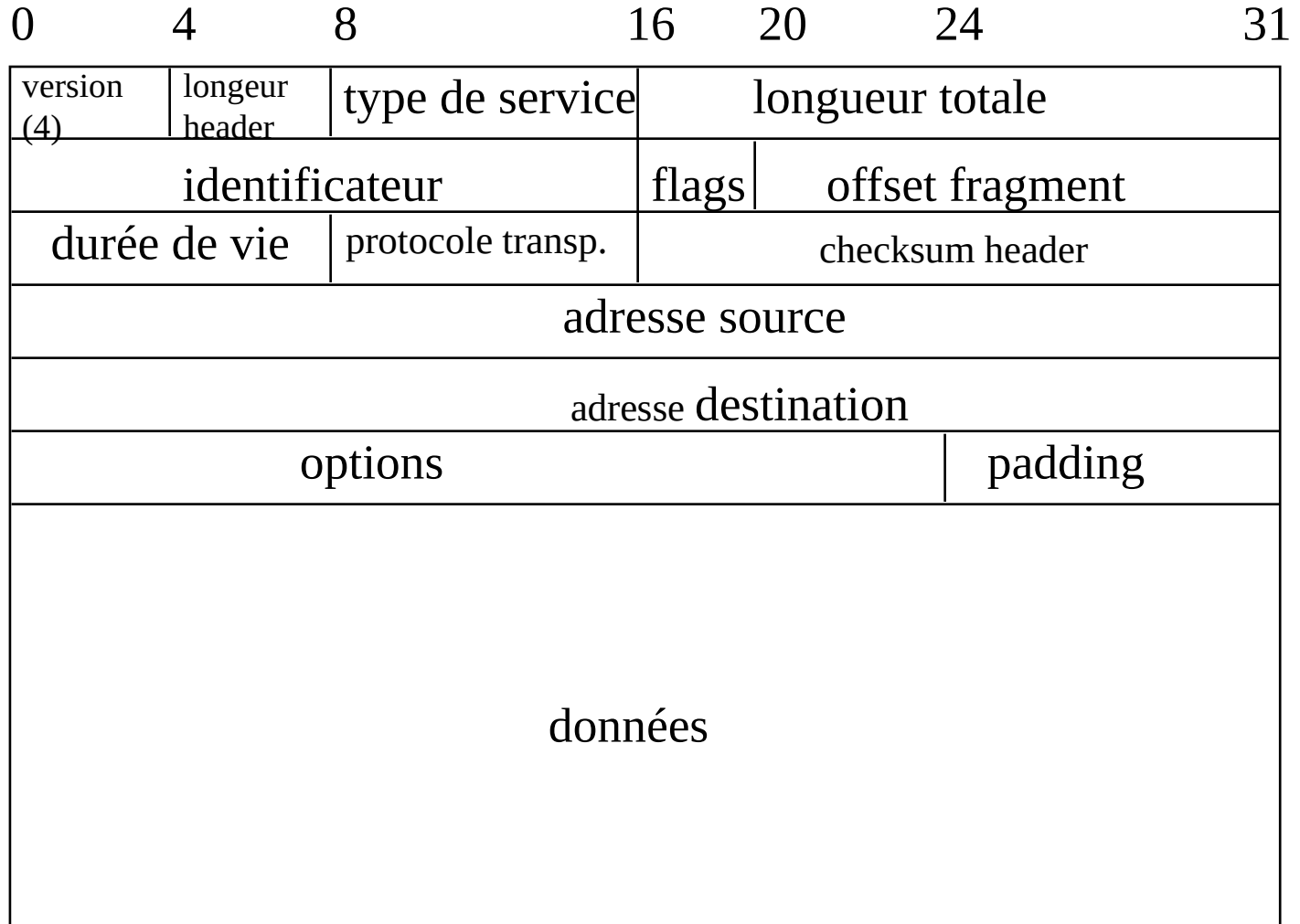
MODELE OSI

Application
Présentation
Session
Transport
Réseau
Liaison
Physique

INTERNET

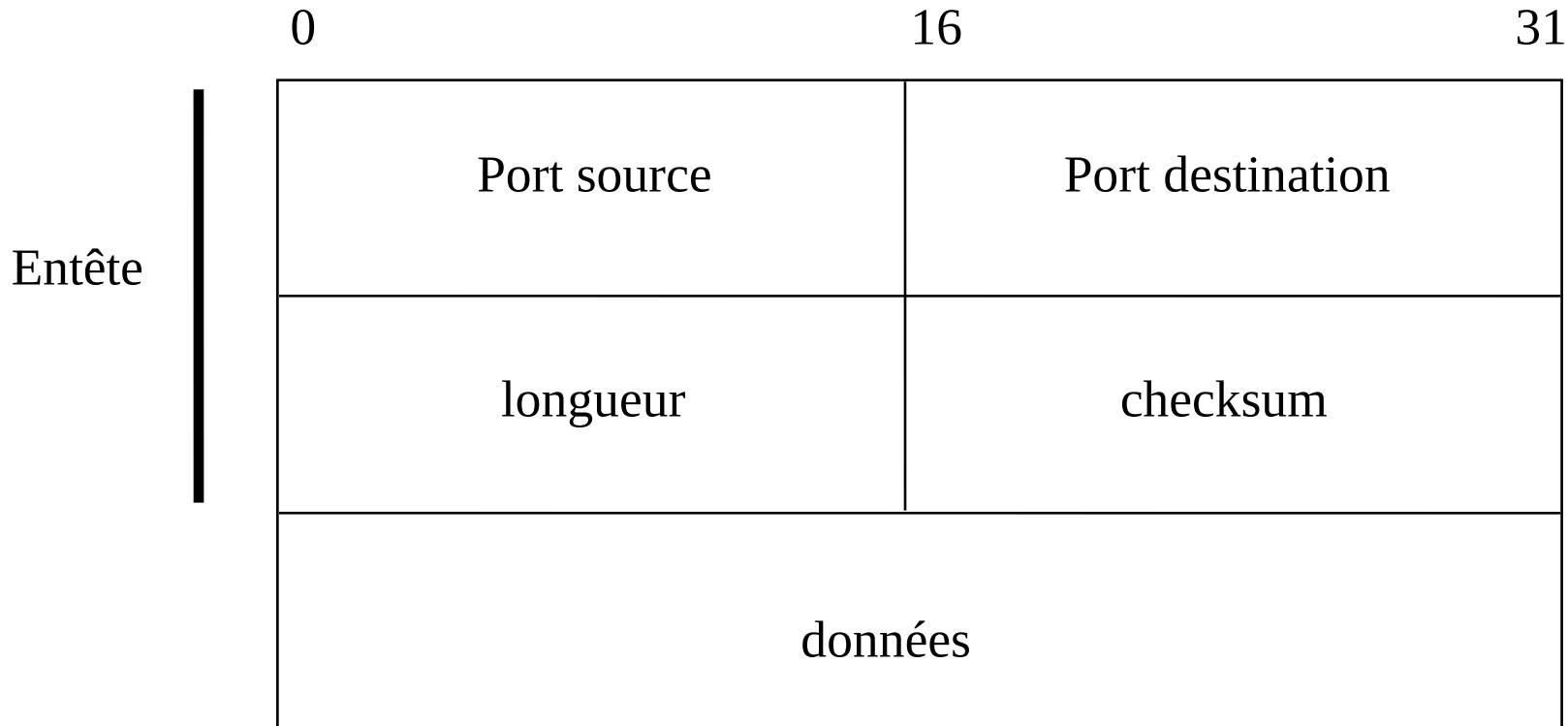
Application Protocols
TCP/UDP
Internet Protocols
Networks Protocol

FORMAT D'UN DATAGRAM IP



header

UDP : FORMAT DATAGRAM



longueur = entête + données (octets)

checksum = CRC sur pseudo entête

0

16

31

port source					port destination				
numéro de séquence									
numéro d'acquittement									
Offset données	Réservé	U R G	A C K	P S H	R S T	S Y N	F I N	Fenêtre	
Checksum					Pointeur données urgentes				
Options								Padding	
Données									

❑ 1 255 : réservés applications INTERNET

- 21 ftp
- 23 telnet
- 25 smtp
- 53 Name Server

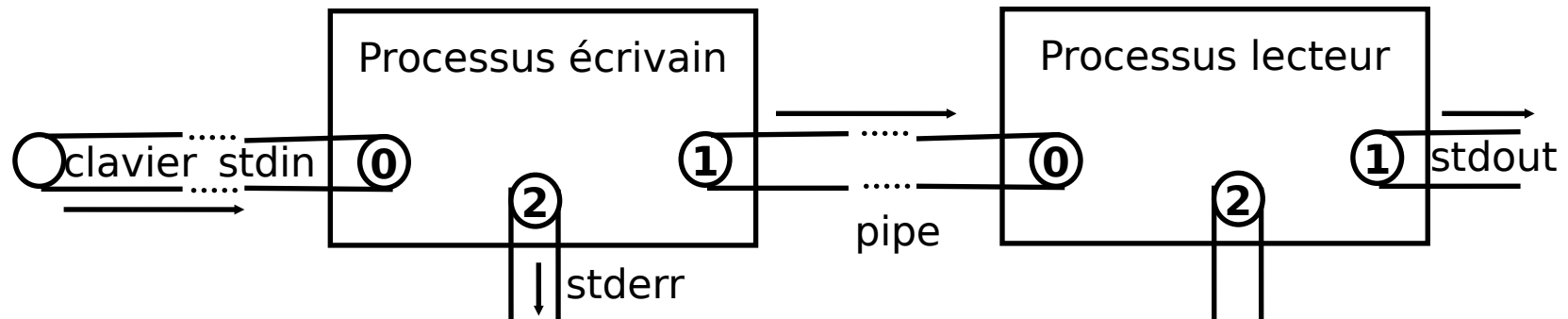
❑ 256 1023 : services UNIX

❑ > 1024 : développements

❑ /etc/services (exemple d'une ligne sous UNIX)

ftp 21/tcp 21/udp #protocole ftp

- ❑ La communication entre deux processus se fait par l'utilisation d'un canal logique (rappel : tout processus possède par défaut trois canaux logiques : *STDIN*, *STDOUT*, *STDERR*).
- ❑ Deux modes de communication sont possibles :
 - connecté (téléphone) : un canal de bout en bout (attaché)
 - non-connecté (courrier) : un canal non attaché
- ❑ On distingue trois types d'IPC :
 - fichier : entre deux processus distincts en mode non-connecté
 - pipe, pairsocket : entre un processus père et un processus fils
 - socket : entre deux processus distincts. Ce mécanisme prend tout son sens dans une application réseau (Client/Serveur).



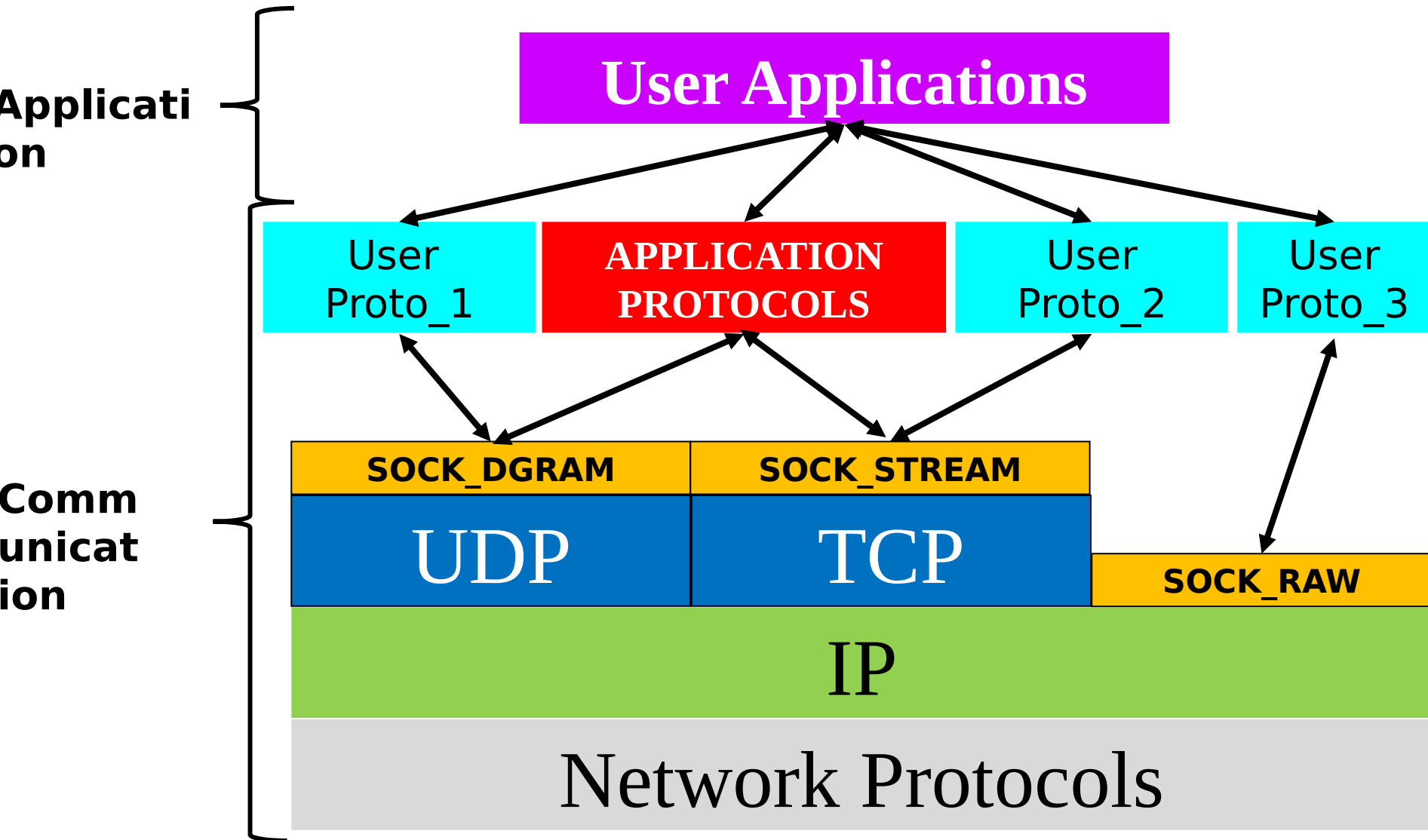
- ❑ Les pipes et les pairsockets offrent une communication entre un processus père et un processus fils : nécessité d'avoir un ancêtre commun.
- ❑ Deux processus créés séparément (éventuellement sur deux machines différentes : souvent le cas) qui veulent communiquer ensemble doivent utiliser le mécanisme de socket.
- ❑ La notion de socket est introduite par UNIX BSD pour banaliser la communication dans le monde Internet.
 - Fournir un modèle standard de communication entre deux processus quelconques.
 - La communication se fait en full-duplex.
 - Une socket peut être assimilée à un fichier réseau nécessité d'un mécanisme d'adressage universel.

- ❑ L'initiative de communication est initiée par un processus. Par conséquent, le destinataire est connu à l'avance.
- ❑ Une socket utilise un nom qui doit être traduit en adresse correspondante avant l'établissement de la communication.
- ❑ La nature des sockets utilisées dépend du domaine d'application :
 - `AF_UNIX`, `AF_INET`, `AF_ROUTE`, ...
- ❑ Chaque domaine nécessite un système d'adressage pour désigner le destinataire :
 - un chemin de fichier de type socket pour `AF_UNIX`,
 - une adresse IP (hôte) et un numéro de port (service applicatif) pour `AF_INET`,

Protocole & mode de communication d'une socket

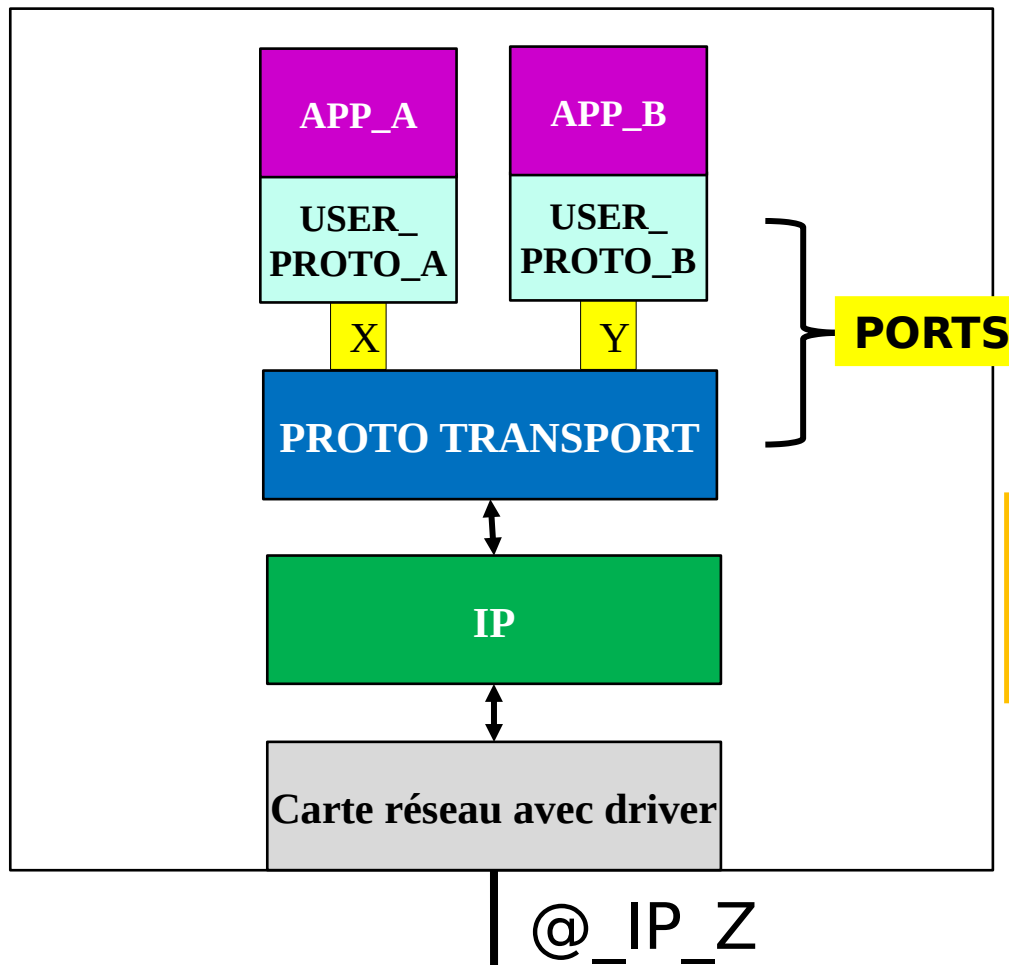
- ❑ Le protocole définit les formats de données, les règles et les conventions régulant la communication.
- ❑ Le protocole permet d'établir une connexion vers le destinataire et d'acheminer les données entre sockets.
- ❑ Trois modes de communication sont possibles :
 - SOCK_DGRAM : DATAGRAM ou mode non connecté (courrier).
 - SOCK_STREAM : STREAM ou mode connecté (Téléphone).
 - SOCK_RAW : administrateur (couche 3 : IP, ICMP, ...)
- ❑ Pour chaque mode de communication (STREAM, DATAGRAM) dans un domaine donné (AF_UNIX, AF_INET, ...), il n'existe généralement qu'un seul protocole de communication.
- ❑ Rappel : La création d'une socket nécessite la spécification du *domaine*, du *mode* et du *protocole*.

AF_INET (1) : interface des sockets



AF_INET (2) : Architecture d'un poste

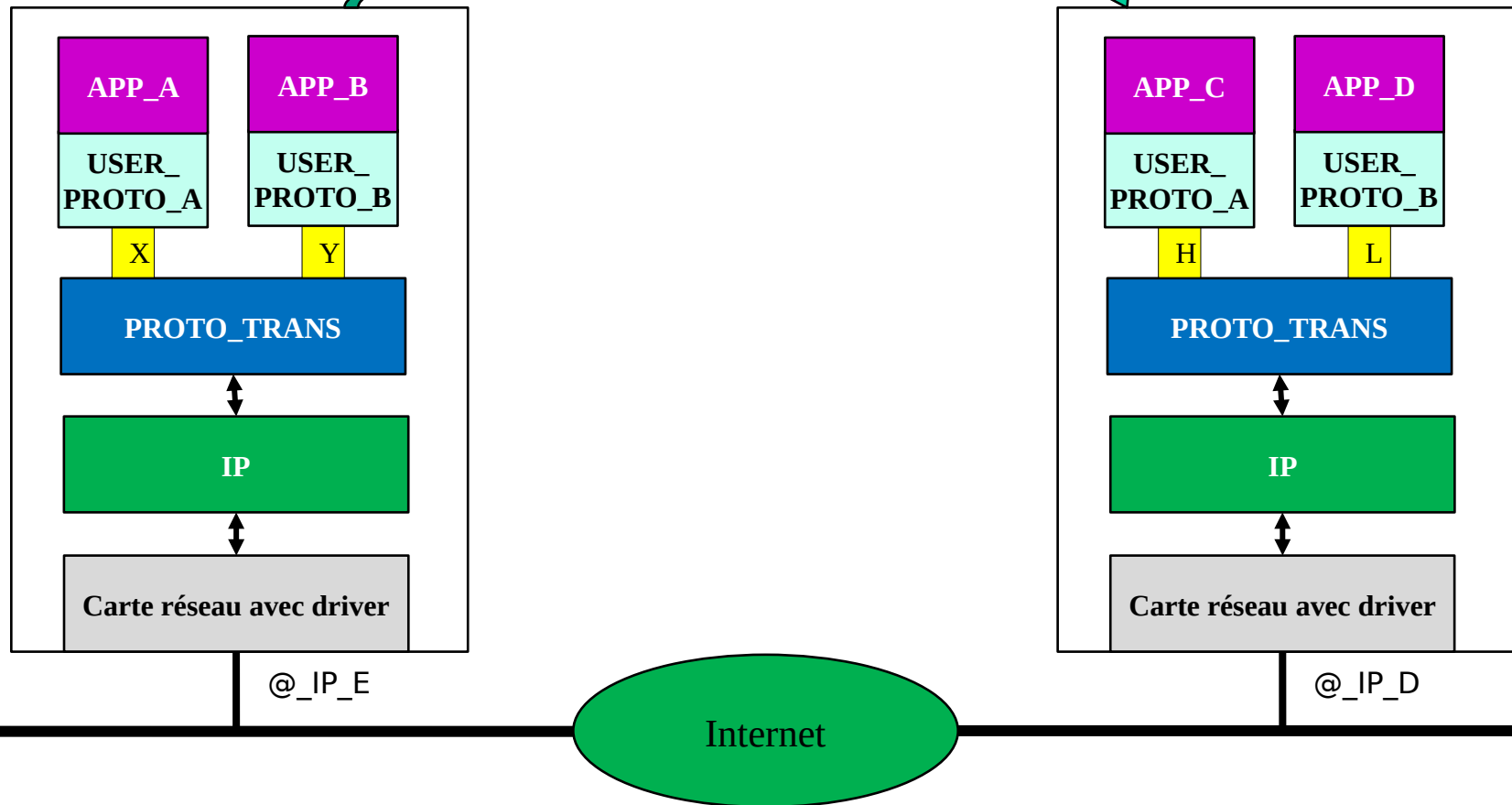
Désignation d'un point de communication d'une application par le triplet : @_IP, Proto de transport, num_port



Exemple :
<123.15.19.12, UDP,
3000>

AF_INET (3) : Communication

(PROTO_Trans, @_IP_E, Port_E, @_IP_S, Port_S)



Exemple : <TCP, 123.15.19.12, 3000, 67.34.98.11, 5200>

Création d'une socket

- ❑ La création d'une socket se fait par l'appel système :

```
#include <sys/socket.h>
sock = socket(int domain, int mode, int protocol);
/*sock=numéro du canal logique attribué*/
```

- ❑ La donnée *domain* indique la famille qui sera utilisée : *AF_UNIX*, *AF_INET*, ...
- ❑ La donnée *mode* indique le mode d'échange qui sera utilisé : *SOCK_DGRAM*, *SOCK_STREAM*, *SOCK_RAW*.
- ❑ Le *protocole* est implicite en fonction du domaine et mode qui seront choisis :
 - la valeur 0 permet d'utiliser le protocole par défaut
 - protocole « *UDP* » pour *AF_UNIX* et *AF_INET* en mode *SOCK_DGRAM*
 - protocole « *TCP* » pour *AF_INET* en mode *SOCK_STREAM*
- ❑ En cas d'échec l'appel renvoie -1 et positionne *errno* à :
 - *INVALID_SOCKET*.
 - *ENOBUFS* : pas assez de mémoire
 - *EPROTOTYPE* : protocole non supporté

Fermeture d'une socket

- ❑ A la fin de l'utilisation, une socket doit être fermée par l'appel

```
close(sock);  
int sock;
```

- ❑ En cas d'erreur `errno` vaut *SOCKET_ERR*.

- ❑ Dans le cas d'utilisation d'une communication en mode connecté, le système va continuer à écrire des données sur la socket. Il est donc préférable d'utiliser l'appel :

```
shutdown(sock, how);  
int sock, how;
```

- ❑ L'argument `how` vaut :
 - 0 : plus de lecture
 - 1 : plus d'écriture
 - 2 : plus de lecture/écriture

AF_INET (4) : Adressage

- ❑ Les sockets de type AF_INET sont une implémentation des protocoles DARPA Internet :
 - adressage IP
 - protocole TCP ou UDP.
- ❑ Les sockets AF_INET sont utilisées pour une communication entre deux processus tournant sur des machines différentes mais également sur la même machine (adresse de bouclage).
- ❑ La communication se fait via un canal réseau de type socket.
- ❑ Structure d'une socket en AF_INET : `#include <netinet/in.h>`

```
struct sockaddr_in {
    short                sin_family;
    u_short              sin_port;
    struct in_addr       sin_addr;
    char                 sin_zero[8]; }
```

- ❑ *sockaddr_in est une spécialisation de sockaddr*

```
struct in_addr {
    in_addr_t s_addr; /* the
};
```


AF_INET (5) : Adressage

- Structure d'une socket en AF_INET : `#include <netinet/in.h>`

```
struct sockaddr_in {  
    short                sin_family;  
    u_short              sin_port;  
    struct in_addr sin_addr;  
    char                 sin_zero[8]; }
```

- *`sockaddr_in` est une spécialisation de `sockaddr`*

```
struct in_addr {  
    in_addr_t s_addr; /* the IP address  
in network byte order */  
};
```

AF_INET (6) : Association d'un adressage à une socket

- ❑ Une socket créée doit être adressée avant toute utilisation par l'appel système :

```
int bind(sock, addr, addrlen);  
int sock, addrlen;  
struct sockaddr *addr;
```

- ❑ Cet appel en cas d'échec renvoie -1 et `errno` vaut ***SOCKET_ERROR***
- ❑ En AF_INET, la donnée *addr* n'est pas obligatoire (utilisation de l'interface réseau par défaut). Pour une machine ayant plusieurs interfaces, *addr* vaut :
 - soit l'adresse IP de l'interface utilisée,
 - soit ***INADDR_ANY*** pour toutes les interfaces.
- ❑ Si dans l'adressage, on utilise 0 comme valeur de port, alors le système attribuera un numéro de port libre (cf. *getsockname()*).
 - ◆ Un serveur doit nécessairement utiliser une valeur de port prédéfinie !!!

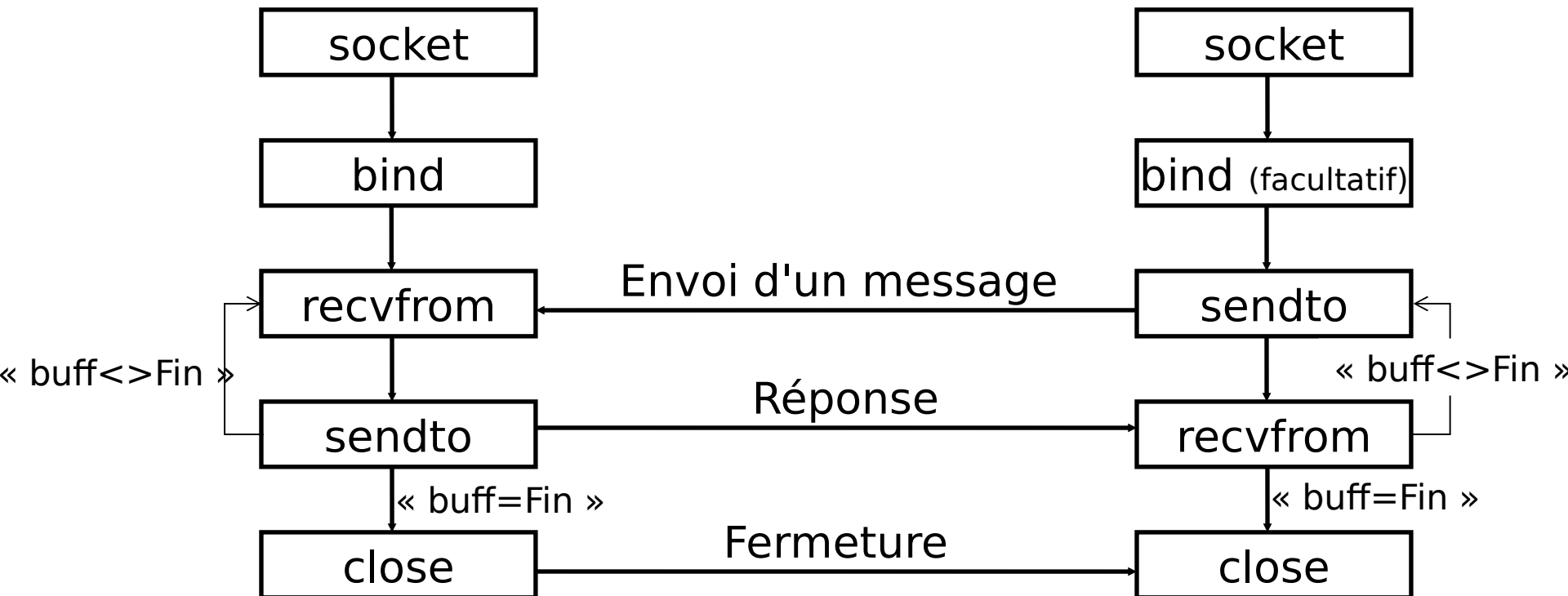
DATAGRAM (1) : caractéristiques

- ❑ Ce mode ne nécessite pas de connexion.
- ❑ L'échange se fait par l'envoi d'un message entier, appelé datagram.
- ❑ Aucune gestion des erreurs de transmission n'est effectuée.
- ❑ Ce mode est rapide mais il n'est pas fiable.
- ❑ Remarque : Ce qui permet de spécifier le protocole de transport utilisé c'est :
 - ◆ La nature de la socket,
 - ◆ Les fonctions de communication utilisées !!!

DATAGRAM (2) : modèle de communication

LECTEUR

ECRIVAIN



DATAGRAM (3) : Réception/envoi de message

- ❑ La réception de données sur une socket non connectée se fait par l'appel :

```
nbcar = recvfrom(sock, buff, buflen, flag, from, fromlen));  
int sock, buflen, flag; int *fromlen; char *buff;  
struct sockaddr *from;
```

- ❑ L'argument *from* sert à mémoriser l'adresse de l'émetteur. Cette information est utile pour renvoyer une réponse à l'émetteur.

- ❑ L'envoi de données sur une socket non connectée se fait par l'appel :

```
nbcar = sendto(sock, buff, buflen, flag, to, tolen));  
int sock, buflen, flag, tolen; char *buff;  
struct sockaddr *to;
```

- ❑ L'argument *to* sert à spécifier l'adresse du destinataire. Cette information est utile pour que le datagramme trouve sa destination.

- ❑ En cas d'échec, ces fonctions renvoient la valeur -1.

DATAGRAM(4) : inet_aton

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct
in_addr *inp);
```

- **Description** : convertit l'adresse Internet de l'hôte *cp* depuis la notation IPv4 décimale pointée vers une forme binaire (dans l'ordre d'octet du réseau), et la stocke dans la structure pointée par *inp*. Renvoie une valeur non nulle si l'adresse est valide, et zéro sinon.

- **Exemple**

```
struct sockaddr_in lect ;
inet_aton("192.168.1.10", &(lect.sin_addr));
```

DATAGRAM(5) : inet_addr

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
in_addr_t inet_addr(const
char *cp);
```

- **Description :** Elle convertit l'adresse internet de l'hôte cp depuis la notation numérique pointée IPV4 en une donnée binaire dans l'ordre des octets du réseau. Si l'adresse est invalide, INADDR_NONE (généralement -1) est renvoyé.

- **Exemple :**

```
struct sockaddr_in lect ;
lect.sin_addr.s_addr =inet_addr("192.168.1.10")
```

DATAGRAM(6) : inet_addr

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
in_addr_t inet_addr(const
char *cp);
```

- **Description** : La fonction `inet_ntoa()` convertit l'adresse Internet de l'hôte `in` donne dans l'ordre des octets du réseau en une chaîne de caractères dans la notation numérique pointée. La chaîne est renvoyée dans un tampon alloué statiquement, qui est donc écrasé à chaque appel.

- **Exemple**

```
struct sockaddr_in lect ;
lect.sin_addr.s_addr =inet_addr("192.168.1.10")
```


DATAGRAM(6) : inet_addr

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
char *inet_ntoa(struct
in_addr in);
```

- **Description :** La fonction `inet_ntoa()` convertit l'adresse Internet de l'hôte `in` donne dans l'ordre des octets du réseau en une chaîne de caractères dans la notation numérique pointée. La chaîne est renvoyée dans un tampon alloué statiquement, qui est donc écrasé à chaque appel.
- **Exemple :**

```
struct sockaddr_in lect ;
printf("L'adresse IP de la machines est %s \n :
inet_ntoa(ecriv.sin_addr)");
```

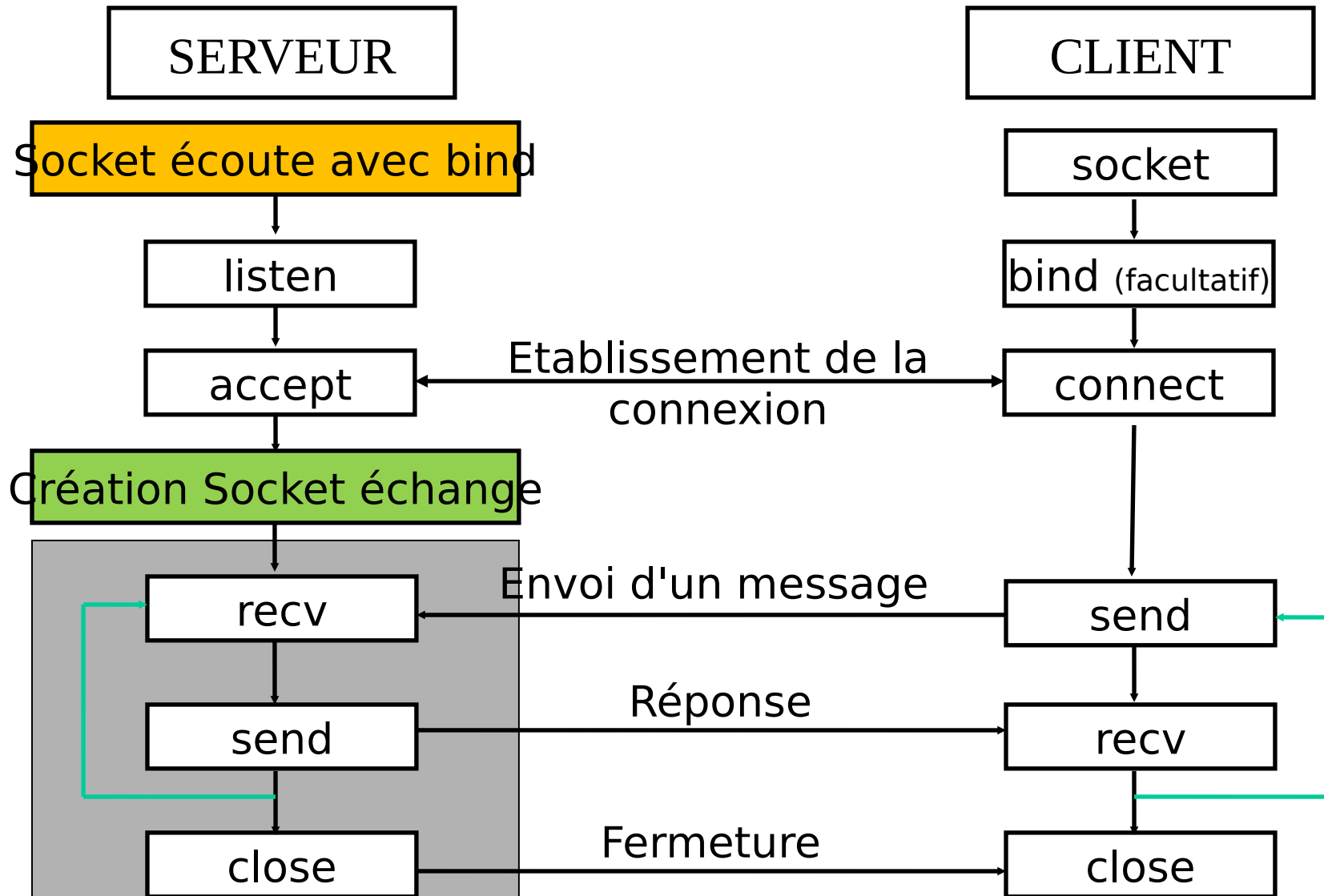
DATAGRAM(4) : Tchat

- On désire réaliser un système de “tchat”. Ce tchat doit permettre à 2 utilisateurs de s’envoyer des messages , jusqu’à ce que l’un des deux envoie le message fin. Dans ce cas, les deux programmes communicants s’arrêtent. L’un des utilisateurs aura le rôle d’écrivain, l’autre celui de lecteur. Au départ le lecteur est suppose être en attente de communication. C’est alors à l’écrivain de prendre l’initiative de la communication en envoyant un premier message au lecteur.
 - ◆ Ecrire le programme du lecteur ?
 - ◆ Ecrire le programme de l’écrivain ?
 - ◆ Un écrivain peut-il écrire simultanément à 2 lecteurs ?
 - ◆ Un lecteur peut-il communiquer simultanément avec plusieurs écrivains ?

STREAM (1) : caractéristiques

- ❑ Ce mode nécessite l'établissement d'une connexion.
- ❑ Ce mode offre un transfert de données fiable :
 - garantie d'acheminement des données,
 - respect de l'ordre des paquets : reconstitution du message entier avant délivrance,
 - envoi des données urgentes.
- ❑ Une gestion automatique des erreurs de transmission (error-free) est assurée par le mode STREAM.
- ❑ Ce mode nécessite un protocole se caractérisant par une transmission de messages segmentés et d'une gestion des erreurs .

STREAM (2) : modèle de communication



STREAM (3) : établissement d'une connexion

- ❑ En mode connecté, le serveur doit établir une connexion avant tout échange.
- ❑ Le serveur doit être en attente de réception de requêtes de connexion de la part des clients.
- ❑ Demande de connexion par le client :

```
etat = connect(sock, addr, addrlen);  
int etat, sock, addrlen;  
struct sockaddr *addr;
```

- ❑ Un serveur multiple doit indiquer le nombre de clients mis en attente de connexion par l'appel :

```
listen(sock, 5);
```

- ❑ Acceptation d'une requête de connexion par le serveur :

```
newsock = accept(sock, from, fromlen);  
int newsock, sock, *fromlen;  
struct sockaddr *from;  
/*sock est la socket d'écoute et newsock celle de la  
connexion*/
```

STREAM (4) : Réception de messages en mode connecté

- ❑ La réception des données se fait par l'appel :

```
nbcar = read(sock, buff, buflen); /*nbcar = nb de caract. reçus */  
int sock, buflen;  
char *buff;
```

- ❑ L'appel *read()* ne peut être utilisé qu'en mode STREAM (connecté) et renvoie le nombre de caractères reçus, ou 0 si le processus distant a fermé la connexion.

- ❑ L'appel *read()* est construit à partir de l'appel :

```
nbcar = recv(sock, buff, buflen, flag);  
int sock, buflen, flag;          /* flag = 0 dans le cas du read */  
char *buff;
```

- ❑ L'argument *flag* permet de configurer la connexion :

- *MSG_OOB* : out of band lecture de messages urgents.
- *MSG_PEEK* : lecture sans retrait du tampon.

- ❑ Erreurs de lecture :

- *ENOTCONN* : socket non connectée

STREAM (5) : Envoi de message en mode connecté

- ❑ L'envoi des données se fait par l'appel :

```
write(socket, buff, buflen);  
int socket, buflen;  
char *buff;
```

- ❑ L'appel *write()* ne peut être utilisé qu'en mode STREAM (connecté) et renvoie -1 en cas d'erreur.
- ❑ L'appel *write()* est construit à partir de l'appel :

```
send(sock, buff, buflen, flag);  
int sock, buflen, flag;          /* flag = 0 dans le cas du write */  
char *buff;
```

- ❑ L'argument *flag* permet de configurer la connexion :
 - *MSG_OOB* : out of band écriture de messages urgents.
 - *MSG_DONTROUTE* : message à destination d'un réseau local.
- ❑ Erreurs d'écriture :
 - *ENOTCONN* : socket non connectée
 - *EPIPE* : socket cassée

- ❑ La programmation des applications réseau nécessite le respect de standards.
- ❑ Les données doivent être converties au format réseau avant leur transmission.
- ❑ La résolution de noms doit passer par l'utilisation de mécanismes communs : DNS, NIS, base d'adresses locale.
- ❑ Passage d'une représentation humaine d'une adresse IP (en décimal) à une représentation machine (4 octets).
- ❑ Les services offerts par une machine sont nommés (cf /etc/services) utilisation des noms au lieu des numéros de port.

- ❑ Les données avant transmission sur le réseau doivent être converties dans un format standard appelé "network order".

- ❑ On dispose des primitives suivantes :
 - **htons(val) : host order to network order short**
 - **htonl(val) : host order to network order long**
 - **ntohs(val) : network order to host order short**
 - **ntohl(val) : network order to host order long**

Primitives de résolution de noms de machines

❑ Structure d'un adressage IP :

```
struct hostent {  
    char    *h_name;  
    char    **h_aliases;  
    int      h_addrtype;      /* ~ AF_INET */  
    int      h_length;        /* 4 */  
    char    **h_addr_list; }  
#define h_addr h_addr_list[0];
```

❑ Nom d'une machine Adresse IP :

```
struct hostent gethostbyname (char *name);
```

❑ Adresse IP Nom d'une machine :

```
struct hostent gethostbyaddr (struct in_addr addr);
```

❑ Adresse IP décimale Adresse IP machine :

```
char *inet_ntoa (struct in_addr addr);
```

❑ Adresse IP machine Adresse IP décimale :

```
struct in_addr inet_addr (char *addr_mach);
```

Primitives de résolution de noms de services

- ❑ Un service réside sur un port et possède un protocole.

- ❑ Structure d'un service IP :

```
struct servent {  
    char    *s_name;  
    char    **s_aliases;  
    int     s_port;  
    char    *s_proto; }
```

- ❑ Nom d'un service Numéro de port :

```
struct servent getservbyname (char *name, *proto);
```



- ❑ Numéro de port Nom d'un service :

```
struct servent getservbyport (port; proto);  
int port; char *proto;
```

Création d'un processus fils

- ❑ Un serveur multiple doit détacher un processus fils pour répondre à une requête de connexion cliente.
- ❑ Le serveur (processus père) exécute alors l'appel système `fork()`.
- ❑ Cet appel crée un deuxième processus (fils ou clone) à l'identique du père. La seule différence résulte dans le résultat renvoyé par le `fork()` .
- ❑ Le `fork()` renvoie le **numéro du processus fils** dans le cas du père et la valeur **0** dans le cas du fils.
- ❑ En cas d'erreur, le résultat vaut **-1**.
- ❑ Les deux processus continuent leur évolution en parallèle et ce à partir de l'appel du `fork()`.

- ❑ Attention à la création en boucle : un premier fils créé par une boucle, créera à partir de cet instant autant de fils que son père.

- [1] Andrew Tannebaum, Réseaux, Ed; InterEditions, 1997
- [2] Laurent Toutain, Réseaux locaux & Internet, Ed. Hermes, 1998
- [3] Samuel Pierre, Réseaux Locaux : Fondements , implantation et études de cas , Ed. Eyrolles, 1991
- [4] C. Machi, J.F. Guilbert, et 13 Co-auteurs, Téléinformatique : transport et traitement de l'information dans les réseaux et systèmes téléinformatiques et télématiques, Ed. Dunod, 1987.
- [5] G. Pujolle, D. Seret, D. Dromard, E. Horlait, Réseaux et Télématique, Tome1, Edition Eyrolles, 1987.
- [6] P. Rolin, Réseaux Locaux : normes et protocoles, Ed. Hermes, 1993