Programmation réseaux
Sockets

Mohamed MAACHAOUI

m.maachaoui@gmail.com

Plan

- Rappel TCP/IP
- Modèle Client / Serveur
- Mode de communication
- Socket
- Exemple socket Linux
- TP : Serveur echo

L’architecture de TCP/IP (1)

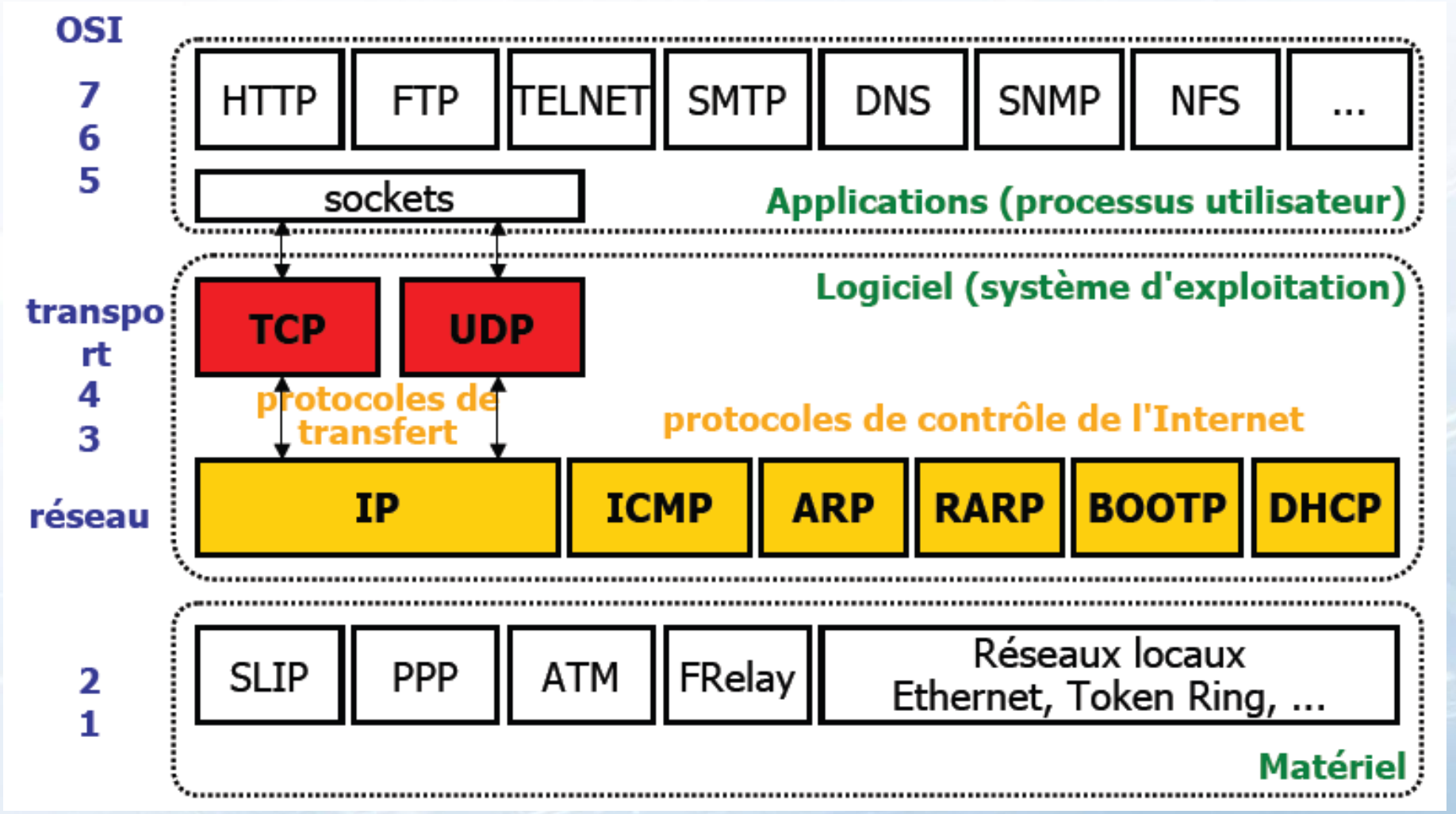
Une version simplifiée du modèle OSI

- **Application** : FTP, WWW, telnet, SMTP, …

- **Transport** : TCP, UDP (entre 2 processus aux extrémités)
  - TCP : transfert fiable de données en mode connecté
  - UDP : transfert non garanti de données en mode non connecté
- **Réseau** : Cette couche permet de gérer les sous réseaux (routage)

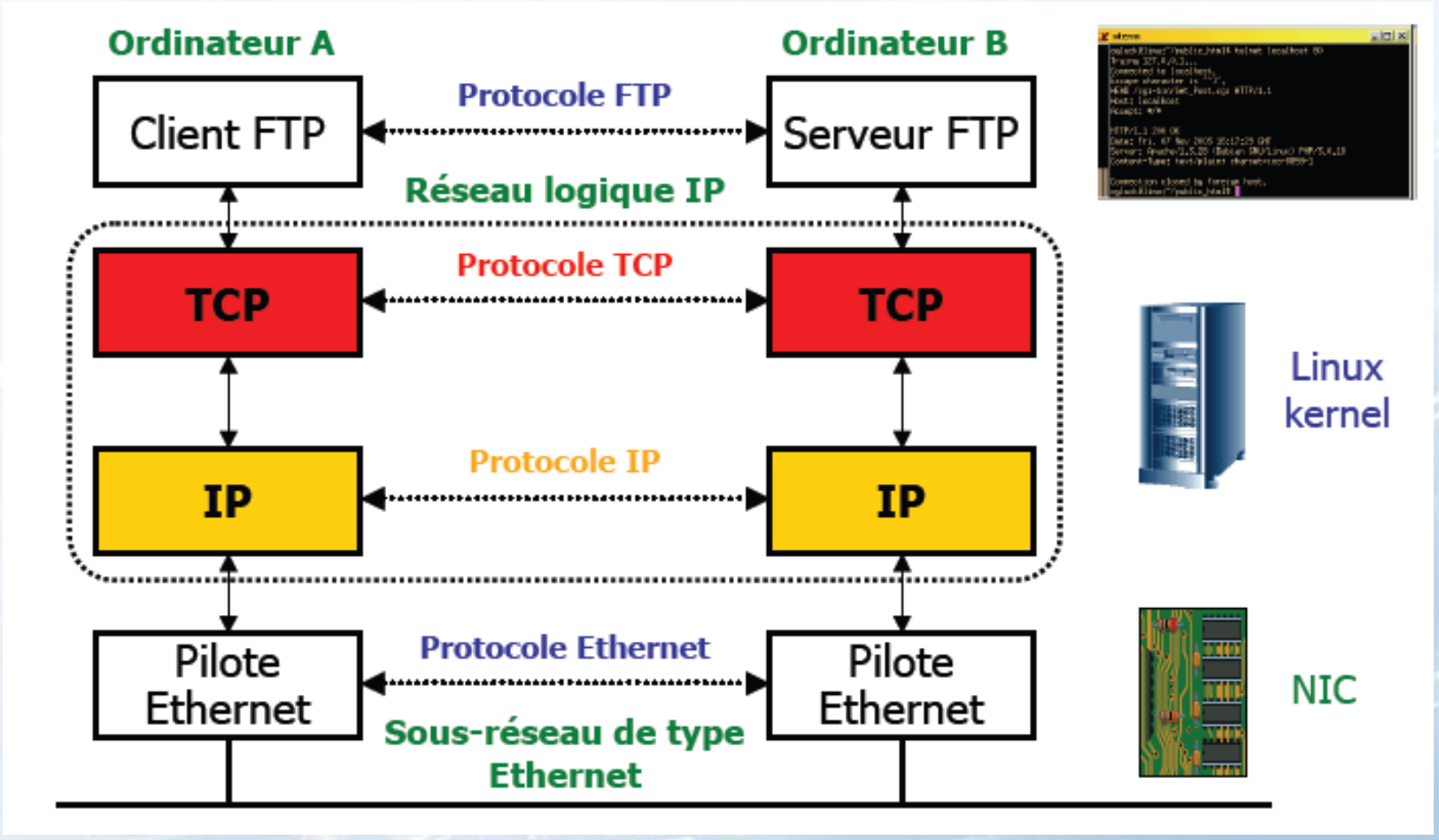
- **Physique** : Elle offre les services de l’interface entre l’équipement de traitement informatique et le support physique de transmission (Information élémentaire binaire)

L'architecture de TCP/IP (2)



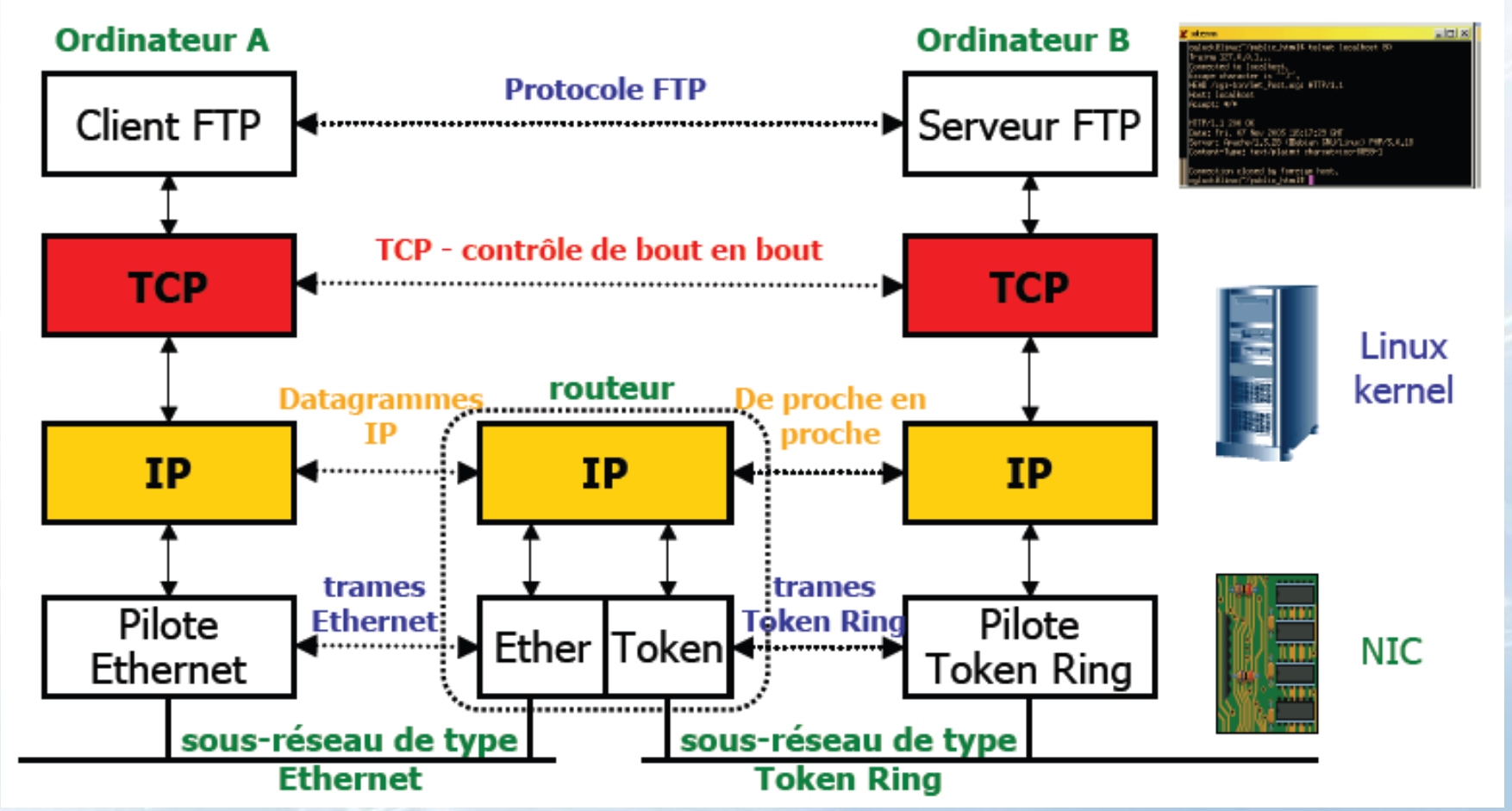
L’architecture de TCP/IP (3)

Deux machines sur un même sous réseau IP

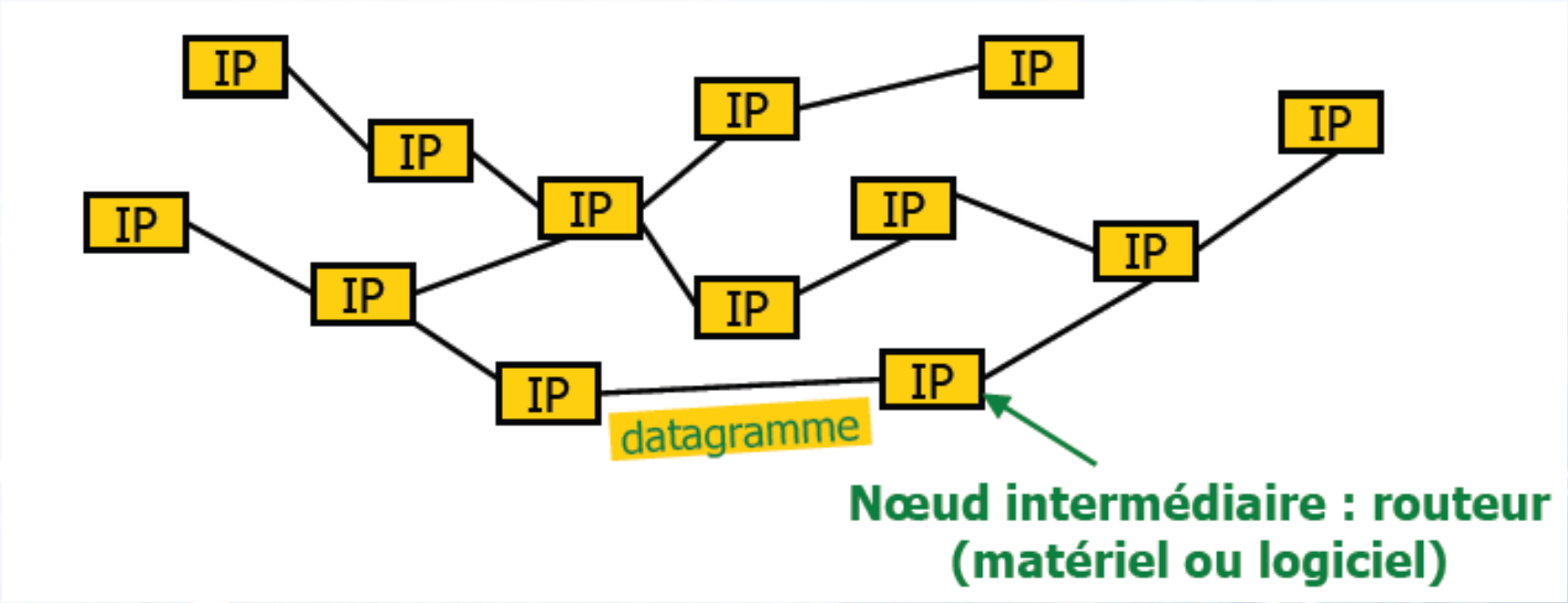


L’architecture de TCP/IP (4)

Prise en compte de l'hétérogénéité



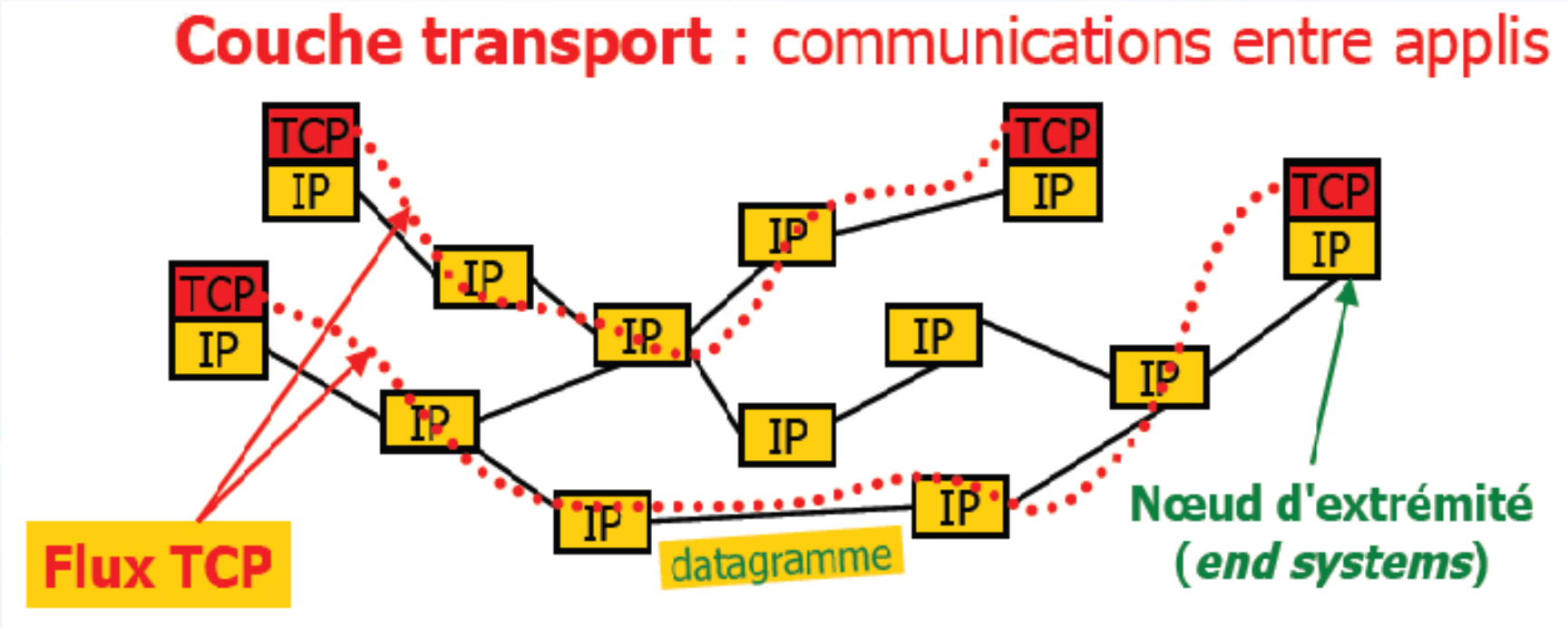
L’architecture de TCP/IP (5)



IP - protocole d'interconnexion, best-effort

- acheminement de **datagrammes** (mode non connecté)
- peu de fonctionnalités, pas de garanties
- simple mais robuste (défaillance d'un noeud intermédiaire)

L'architecture de TCP/IP (6)



TCP - protocole de transport **de bout en bout**

- uniquement présent aux extrémités
- transport **fiable** de **segments** (mode connecté)
- protocole complexe (retransmission, gestion des erreurs, séquencement, …)

L’architecture de TCP/IP (7)

- **Transport Control Protocol (RFC 793, 1122, 1323, 2018, 2581)**

- **Transport fiable en mode connecté**
  - point à point, bidirectionnel : entre deux adresses de transport (@IP src, port src) --> (@IP dest, port dest)
  - transporte un flot d'octets (ou flux)
  - l'application lit/écrit des octets dans un tampon
  - assure la délivrance des données en séquence
  - contrôle la validité des données reçues
  - organise les reprises sur erreur ou sur temporisation
  - réalise le contrôle de flux et le contrôle de congestion (à l'aide d'une fenêtre d'émission)

L’architecture de TCP/IP (8)

- **UDP (RFC 768) - User Datagram Protocol**
  - protocole de transport le plus simple
  - service de type best-effort (comme IP)
    - les segments UDP peuvent être perdus
    - les segments UDP peuvent arriver dans le désordre
  - mode non connecté : chaque segment UDP est traité
  - indépendamment des autres
- **Pourquoi un service non fiable sans connexion ?**
  - simple donc rapide (pas de délai de connexion, pas
  - d'état entre émetteur/récepteur)
  - petit en-tête donc économie de bande passante
  - sans contrôle de congestion donc UDP peut émettre aussi rapidement qu'il le souhaite

L’architecture de TCP/IP (9)

Utilisation d’UDP

- **Performance sans garantie de délivrance**
- **Souvent utilisé pour les applications multimédias**
  - tolérantes aux pertes
  - sensibles au débit
- **Autres utilisations d'UDP**
  - applications qui envoient peu de données et qui ne nécessitent pas un service fiable
  - exemples : DNS, SNMP, BOOTP/DHCP
- **Transfert fiable sur UDP**
  - ajouter des mécanismes de compensation de pertes (reprise sur erreur) au niveau applicatif
  - mécanismes adaptés à l'application

Mode de connexion

- Définit par la norme ISO 7498
- Le mode de connexion définit le (ou les) processus utilisé par deux entités d'extrémité avant/après la phase d'échange d'informations applicatives
- Une communication entre des terminaux connectés sur un réseau peut se faire fondamentalement de 2 manières :
  - Avec (= orienté connexion) "Connection Oriented " = CO
  - Sans connexion : "Connection-less" = CL .

Le mode connecté

- C'est un mode durant lequel, le transfert d'info est obligatoirement précédé d'une phase de négociation
  - Etablissement d'un contexte applicatif
- Le mode connecté est un mode contextuel
  - Ex :
    - Accord sur la vitesse de transfert
    - Accord sur les adresses de transfert
    - Accord sur la qualité de service
      - TCP, ATM, RSVP, RTC, RNIS, X.25

Diagramme de flux du mode connecté

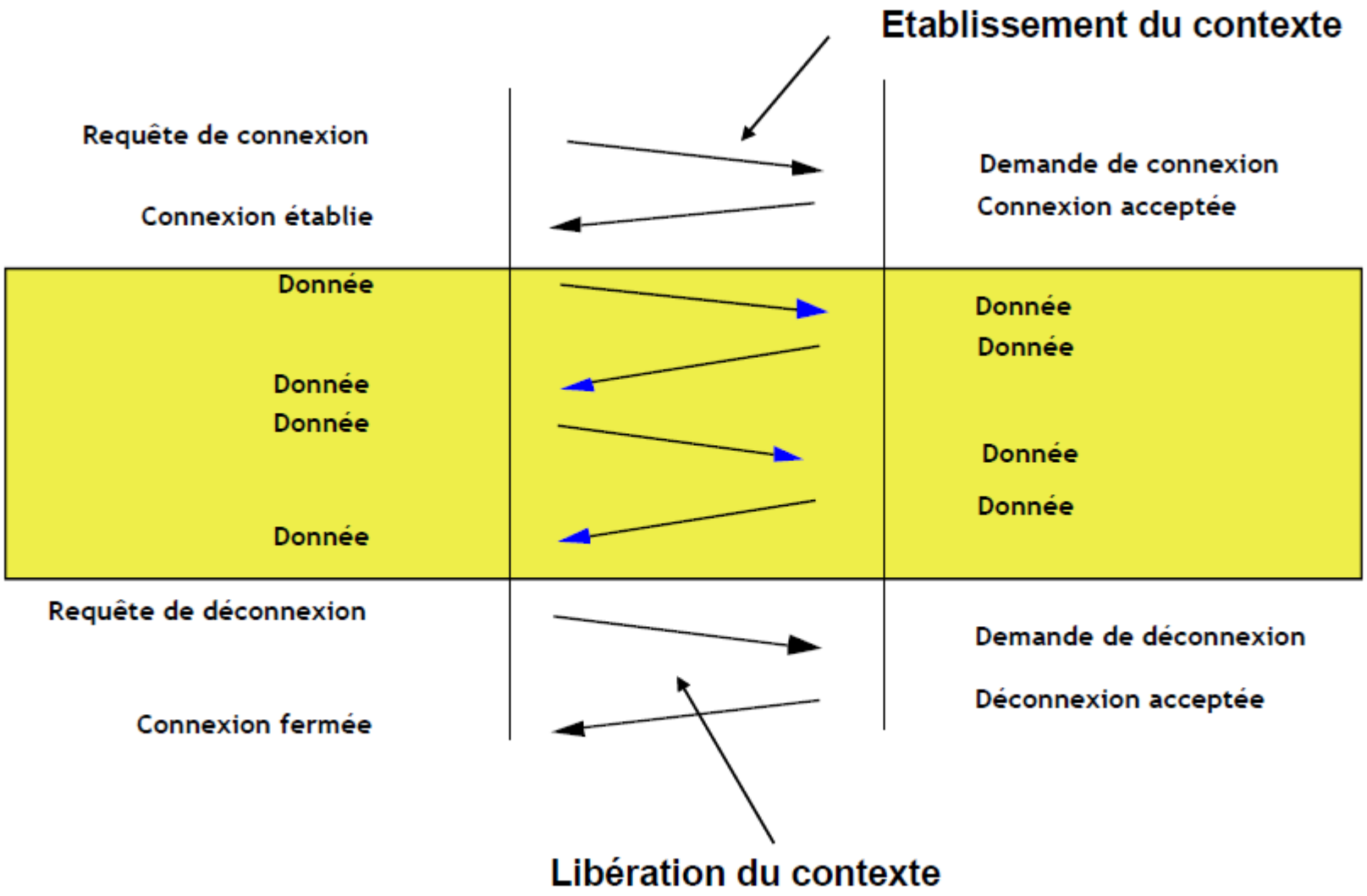
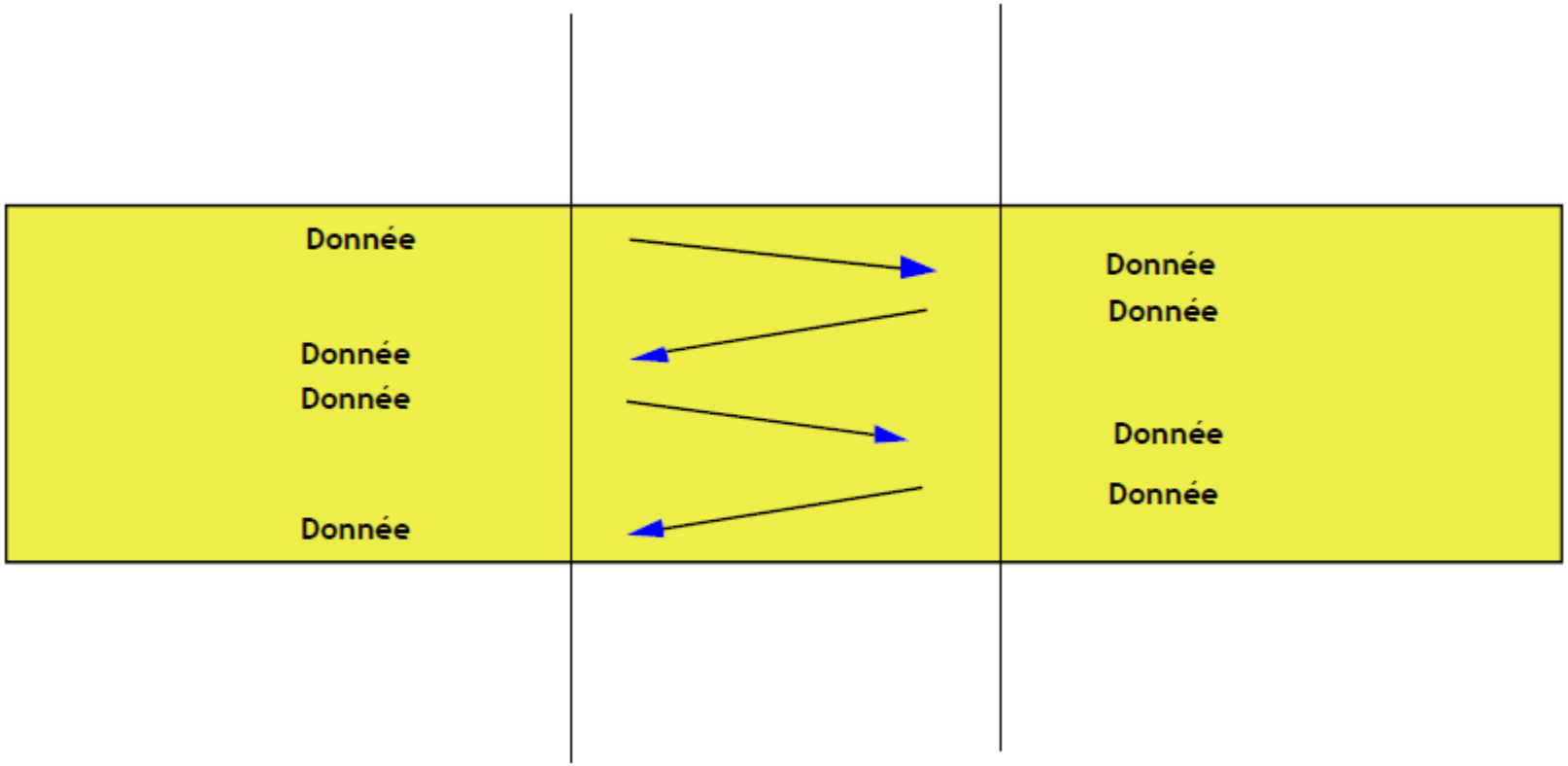


Diagramme de flux du mode non connecté



Comparaisons entre...

CONNECTÉ

C.O.N.S. *(Connection Oriented Network Services)*

CIRCUIT *(Virtuel)*

- Etablissement d'une connexion
- Paquet de communication sans adresse
- Libération de la connexion en fin de session
- Réception de paquets séquencés
- Présence destinataire obligatoire
- Diffusion difficile

Norme ISO 8205 pour X.25 (paquets)

NON CONNECTÉ

C.L.N.S. *(Connection Less Network Services)*

DATAGRAMME

- **Pas de connexion**
- **Adresses dans chaque datagramme**
- **Pas de procédure de libération**
- **Datagrammes non séquencés à réception**
- **Présence destinataire non nécessaire**
- **Diffusion aisée**

Norme ISO 8473 pour I.P. (Internet Protocol)

Les applications réseau (1)

- Applications = la raison d'être des réseaux infos
- Profusion d'applications depuis 30 ans grâce à l'expansion d'Internet
  - années 1980/1990 : les applications "textuelles"
    - messagerie électronique, accès à des terminaux distants, transfert de fichiers, groupe de discussion (forum, *newsgroup*), dialogue interactif en ligne (chat), la navigation Web
  - plus récemment :
    - les applications multimédias : vidéo à la demande (*streaming*), visioconférences, radio et téléphonie sur Internet
    - la messagerie instantanée (ICQ, MSN Messenger)
    - les applications *Peer-to-Peer* (MP3, ...)

Les applications réseau (2)

- L'application est généralement répartie (ou distribuée) sur plusieurs systèmes
- Exemples :
  - L'application Web est constituée de deux logiciels communiquants : le navigateur client qui effectue une requête pour disposer d'un document présent sur le serveur Web
  - L'application *telnet* : un terminal virtuel sur le client, un serveur *telnet* distant qui exécute les commandes
  - La visioconférence : autant de clients que de participants
- --> Nécessité de disposer d'un protocole de communication applicatif !

Terminologie des applications réseau

- Processus :
  - une entité communicante
  - un programme qui s'exécute sur un hôte d'extrémité
- Communications inter-processus locales :
  - communications entre des processus qui s'exécutent sur un même hôte
  - communications régies par le système d'exploitation (tubes UNIX, mémoire partagée, ...)
- Communications inter-processus distantes :
  - les processus s'échangent des **messages** à travers le réseau selon un **protocole** de la couche applications
  - nécessite une infrastructure de transport sous-jacente

Protocoles de la couche Applications

- Le protocole applicatif définit :
  - le format des messages échangés entre les processus émetteur et récepteur
  - les types de messages : requête, réponse, ...
  - l'ordre d'envoi des messages
- Exemples de protocoles applicatifs :
  - HTTP pour le Web, POP/IMAP/SMTP pour le courrier électronique, SNMP pour l'administration de réseau, ...
- Ne pas confondre le protocole et l'application !
  - Application Web : un format de documents (HTML), un navigateur Web, un serveur Web à qui on demande un document, un protocole (HTTP)

Le modèle Client / Serveur

- Idée : l'application est répartie sur différents sites pour optimiser le traitement, le stockage...
- Le client
  - effectue une demande de service auprès du serveur (**requête**)
  - initie le contact (parle en premier), ouvre la session
- Le serveur
  - est la partie de l'application qui offre un service
  - est à l'écoute des requêtes clientes
  - répond au service demandé par le client (**réponse**)

Le modèle Client / Serveur

- Le client et le serveur ne sont pas identiques, ils forment un système coopératif
  - les parties client et serveur de l'application peuvent s'exécuter sur des systèmes différents
  - une même machine peut implanter les côtés client ET serveur de l'application
  - un serveur peut répondre à plusieurs clients simultanément

Communication client/serveur

Client Dialogue Serveur

Demande

Attend

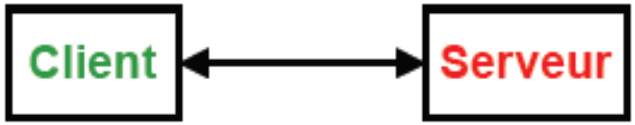
Réalise/ Exécute

Reçoit

Envoie

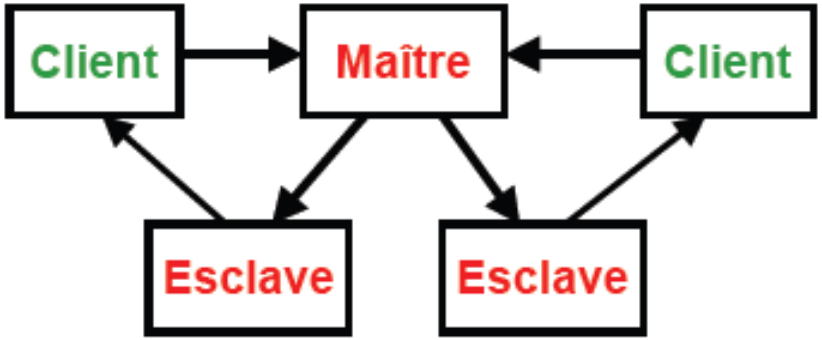
Des clients et des serveurs...

Un client, un serveur :



Requête/Réponse

Plusieurs clients, un serveur :



Le serveur traite plusieurs requêtes simultanées

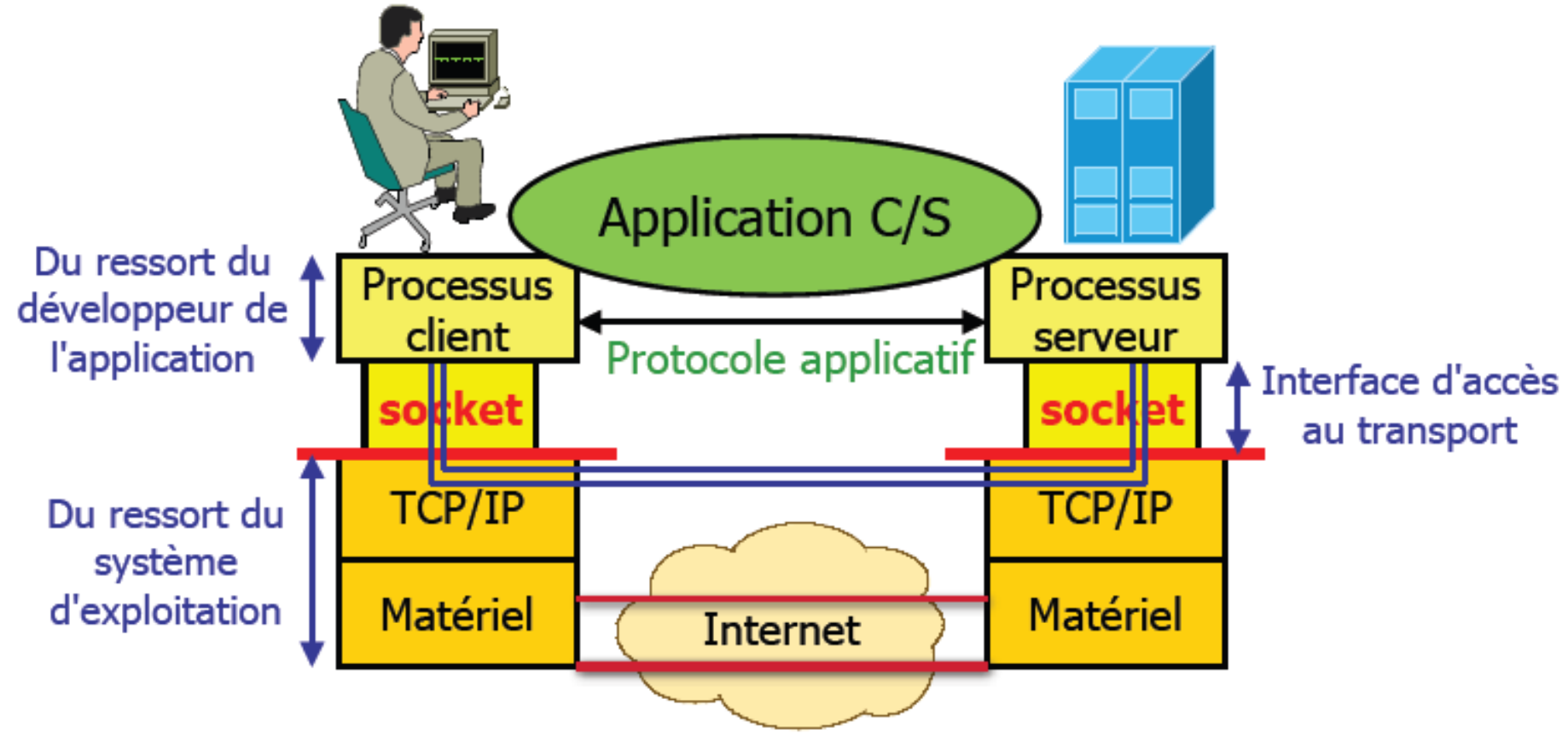
Un client, plusieurs serveurs :

Le serveur contacté peut faire appel à un service sur un autre serveur (ex. SGBD)

Interface de programmation réseau

- Il faut une interface entre l'application réseau et la couche transport
  - le transport n'est qu'un tuyau (TCP ou UDP dans Internet)
  - l'API (*Application Programming Interface*) n'est que le moyen d'y accéder (interface de programmation)
- Les principales APIs de l'Internet
  - les sockets
    - apparus dans UNIX BSD 4.2
    - devenus le standard de fait
  - les RPC : Remote Procedure Call - appel de procédures distantes

Interface de programmation réseau



Une socket : interface locale à l'hôte, créée par l'application, contrôlée par l'OS

Porte de communication entre le processus client et le processus serveur

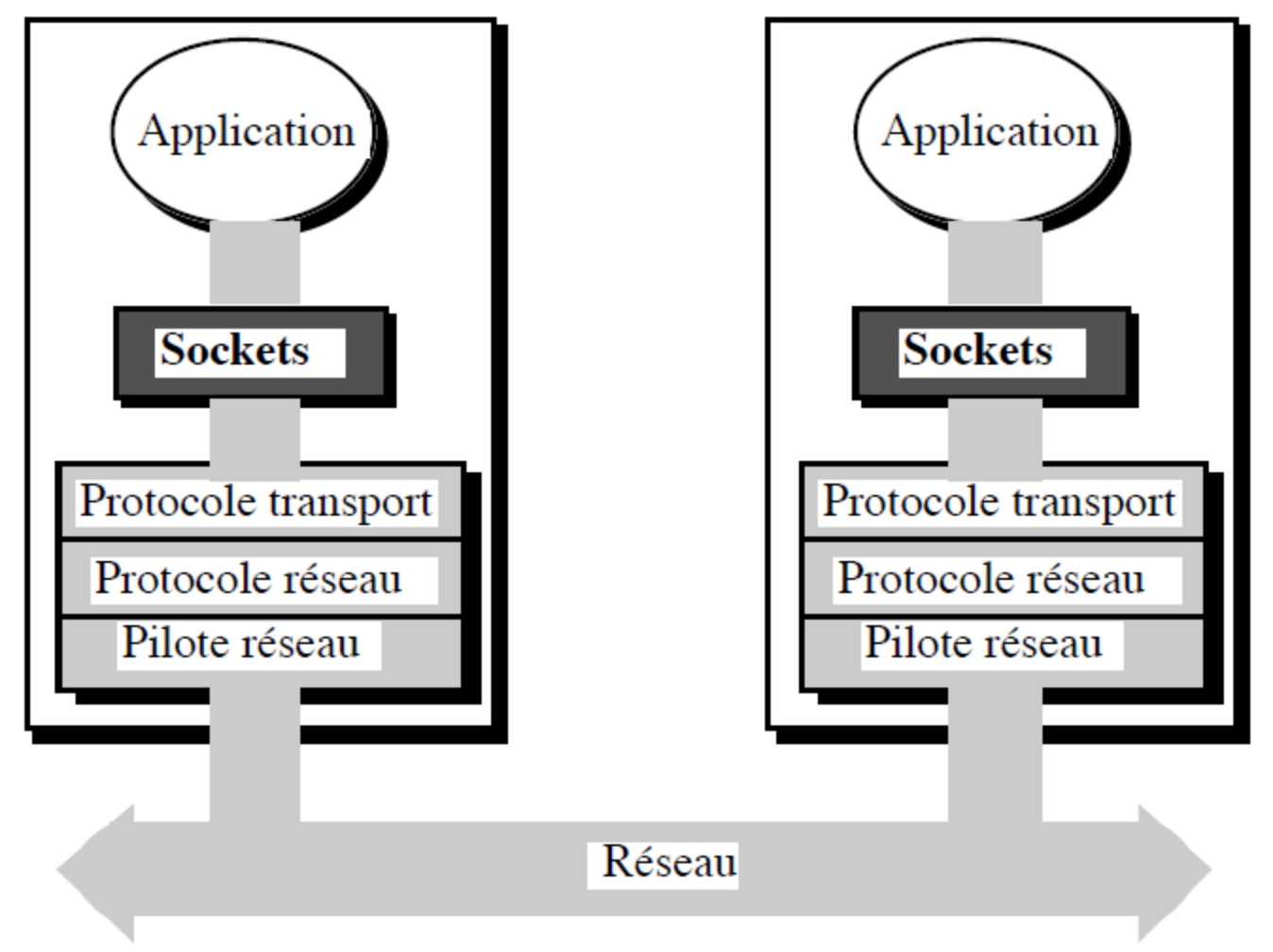
Application C/S - récapitulatif

- Une application Client/Serveur, c'est
  - **une partie cliente** qui exécute des requêtes vers un serveur
  - **une partie serveur** qui traite les requêtes clientes et y répond
  - **un protocole applicatif** qui définit les échanges entre un client et un serveur
  - **un accès via une API** (interface de programmation) à la couche de transport des messages
- Bien souvent les parties cliente et serveur ne sont pas écrites par les mêmes programmeurs (Navigateur Netscape/Serveur apache) --> rôle important des RFCs qui spécifient le protocole !

Généralités interface "socket"

- **1982** : API réseaux pour la version UNIX BSD.
- *Deux programmes différents*
  - » Le **serveur** se met en attente de demandes (passif)
  - » Le **client** initie le dialogue par une demande (actif)
- **Objectifs**
  - Fournir des moyens de communications entre processus (*IPC*) **utilisables en toutes circonstances**: échanges *locaux* ou *réseaux*.
  - *Pour programmeur*
    - offre moyen d'utiliser des *fonctions du SE semblables à celles qui régissent l'accès aux fichiers*, e.g., lire&écrire données sur réseau comme il ferait sur système fichiers
  - *Pour usagers*
    - cacher *détails d'implantation* des couches transport.
    - Si possible, cacher les **différences entre protocoles de transport hétérogènes** sous une même interface (TCP, Novell XNS, OSI)

Généralités interface "socket"



Généralités interface "socket"

- Interface de programmation pour les communications
- Ensemble de primitives assurant ce service,
- Générique : s'adapte aux différents besoins de communication,
- Indépendant de protocoles et de réseaux particuliers :
  - Mais développé à l'origine sous Unix 4BSD.
- N'utilise pas forcément un réseau :
  - Par exemple : communication locale (interne à une station).
- point de communication par lequel un processus peut émettre ou recevoir des données
- Homogène avec les identificateurs d'E/S :
  - l'identificateur (de descripteur) de Socket est compatible avec l'identificateur (de descripteur) de fichiers.

Sockets

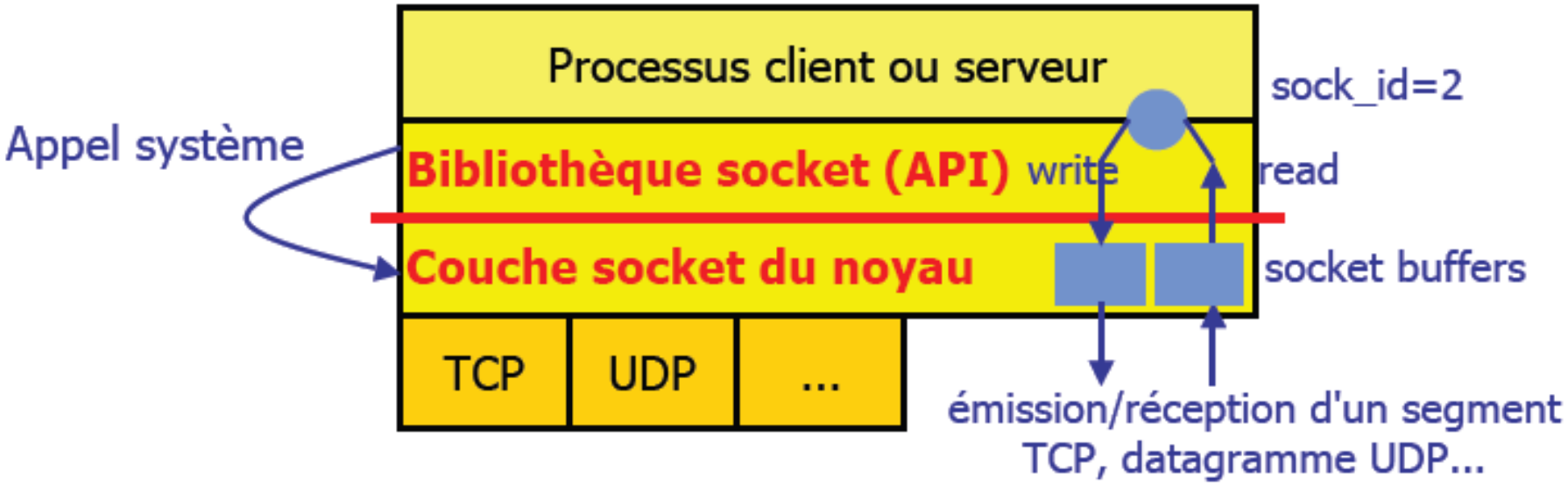
✓ Le terme " socket " désigne à la fois une ***bibliothèque d'interface réseau*** et ***l'extrémité d'un canal de communication*** (point de communication) par lequel un processus peut émettre ou recevoir des données.

✓ L'interface socket est un ensemble de ***primitives*** qui permettent de gérer ***l'échange*** de données entre ***processus***, que ces processus soient exécutés ou non sur la même machine.

✓ La bibliothèque socket ***masque*** l'interface et les ***mécanismes*** de la ***couche transport*** : un appel socket se traduit par plusieurs requêtes transport.

Les sockets en pratique

Un descripteur de socket (sock_id) n'est qu'un point d'entrée vers le noyau



- la bibliothèque socket est liée à l'application

- la couche socket du noyau réalise l'adaptation au protocole de transport utilisé

Analogie avec le téléphone

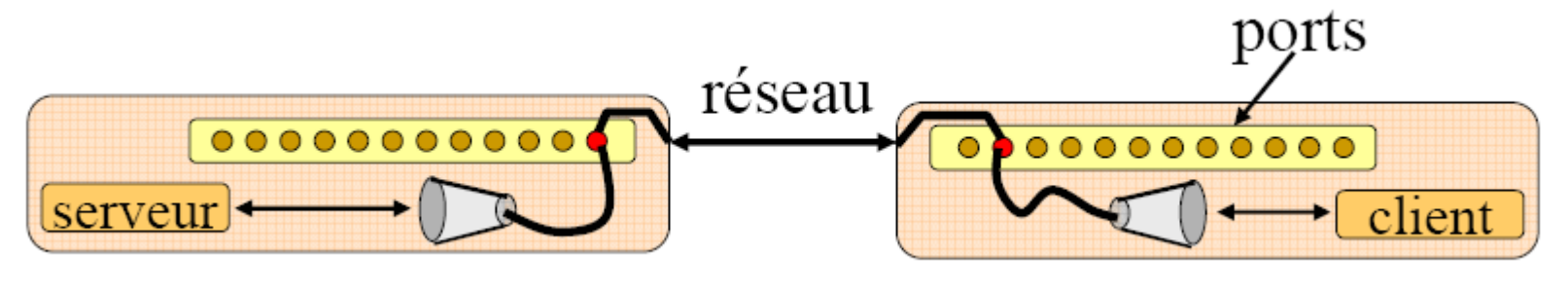
- **Socket** = point d'accès au réseau pour couches transport
  - désigne souvent l'ensemble min des infos permettant à un logiciel d'écouter les tentatives d'établissement de connexion réseau,
    - e.g., @IP, protocole (souvent UDP ou TCP) et port ...
  - **Comparable à un téléphone**
  - C'est l'*extrémité d'un canal* de communication *permettant l'échange de donnée* entre deux entités (les utilisateurs du téléphone)
- *utilisateurs* des téléphones — *applis/process* qui utilisent ces sockets
  - Utilisateur *demandeur* — c'est le client, c'est lui *qui compose num*
  - Utilisateur *en attente coup fil* — c'est le serveur, il *décroche*
- Une fois la communication établie, elle est bi-directionnelle et symétrique

Les primitives de l'interface socket

Exemple en langage C en UNIX.

Sockets

1. Chaque machine crée une socket, **socket()**
2. Chaque socket sera associée à un port de sa machine hôte, **Bind()**
3. Les deux sockets seront explicitement connectées si on utilise un protocole en mode connecté…,
4. Chaque machine lit et/ou écrit dans sa socket, **write()/Read()**
5. Les données vont d'une socket à une autre à travers le réseau,
6. Une fois terminé chaque machine ferme sa socket. **Close()**



Sockets fonctions de base

Une socket =
{une famille ; un mode de communication ; un protocole}

- ✓ **Exemples de familles de sockets :**
  - ✓ processus sur la même station Unix :
    → sockets locales (AF_UNIX)
  - ✓ processus sur des stations différentes à travers Internet :
    → sockets Internet (AF_INET)
- ✓ **Exemples de modes de communication :**
  - ✓ Datagrames – ou mode non connecté (SOCK_DGRAM)
  - ✓ Flux de données – ou mode connecté (SOCK_STREAM)
- ✓ **Exemples de protocoles de sockets : IP, UDP, TCP, …**

Toutes les combinaisons ne sont pas possibles !!

Sockets fonctions de base

Association d'une socket à un port :

```
int bind(
int descripteur,                    /* socket */
struct sockaddr *ptr_adresse,       /* pointeur sur adresse */
int lg_adresse                      /* taille adresse */
);
```

Fermeture et suppression d'une socket:

```
int close(
int descripteur, /* socket */
);
```

Structure de base employée pour stocker des informations sur les adresses de sockets :

```
struct sockaddr {
unsigned short sa_family;                    // address family, AF_xxx
char sa_data[14];                            // 14 bytes of protocol address
};
```

/* Adresse Internet d'une socket */

```
struct sockaddr_in {
short sin_family ;                           /* AF_INET */
u_short sin_port ;                           /* Port */
struct in_addr sin_addr;                     /* Adresse IP */
char sin_zero[8] ;
};
```

/* Adresse Internet d'une machine */

```
struct in_addr {
        u_long    s_addr ;
};
```

Sockets UDP en C-Linux

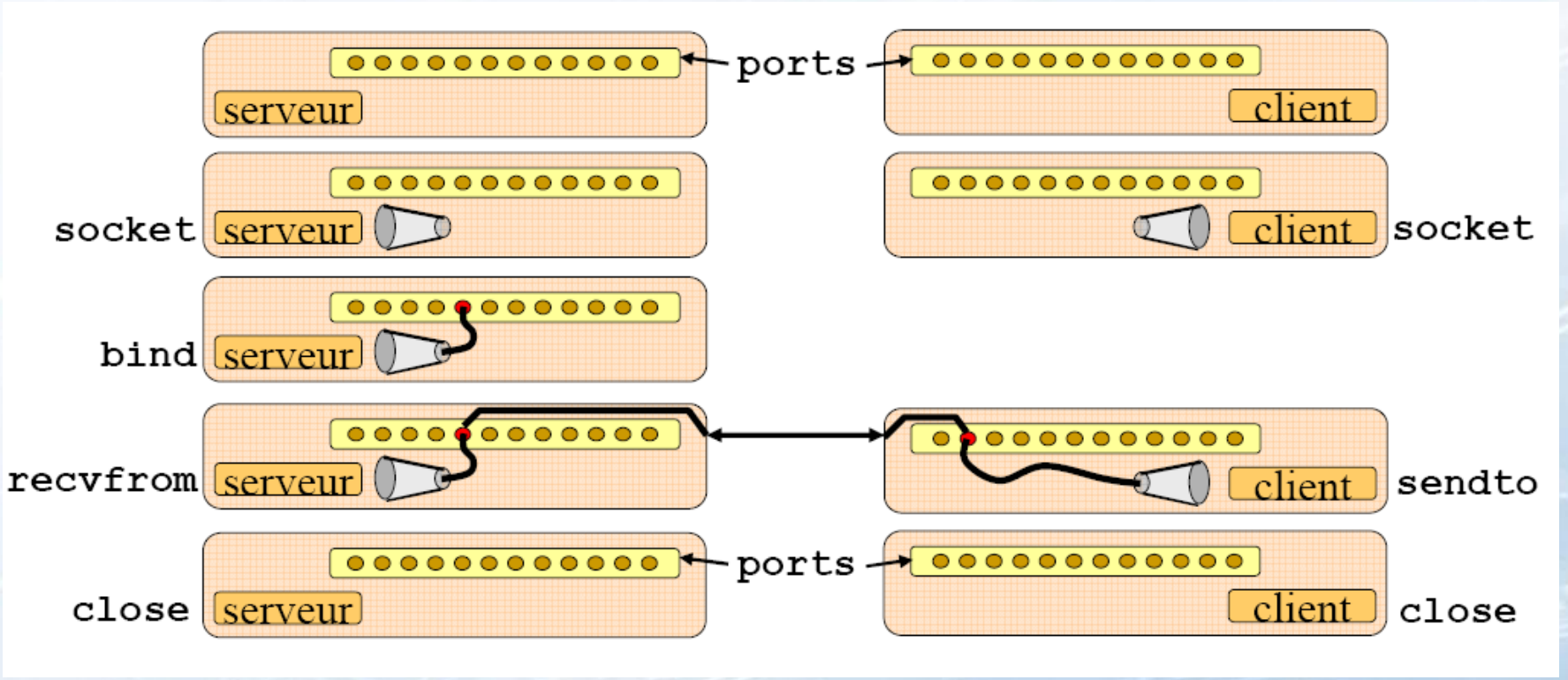
Étapes d'une connexion client-serveur en UDP :

- ✓ le serveur et le client ouvrent chacun une « socket »
- ✓ le serveur la nomme (il l'attache à un de ses ports (un port précis))
  - ✓ le client ne nomme pas sa socket (elle sera attachée automatiquement à un port lors de l'émission)
- ✓ le client et le serveur dialogue : **sendto(…)** et **recvfrom(…)**
- ✓ finalement toutes les sockets doivent être refermées

Les deux extrémités n'établissent pas une connexion :

- ✓ elles ne mettent pas en oeuvre un protocole de maintien de connexion
- ✓ ***si le processus d'une extrémité meurt l'autre n'en sait rien !***

Sockets UDP en C-Linux



Sockets fonctions de base

```
int sendto(
int descripteur,                        /* Id de socket émetteur */
void *message,                          /* message à envoyer */
int longueur,                           /* taille du message */
int option,                             /* 0 pour DGRAM */
struct sockaddr *ptr_adresse,           /* destinataire */
int lg_adresse                          /* taille adr destinataire */
);
```

```
int recvfrom(
int descripteur,                        /* Id de socket récepteur */
void *message,                          /* pointeur sur message reçu */
int lg_message,                         /* taille du msg à recevoir */
int option,                             /* 0 ou MSG_PEEK */
struct sockaddr *ptr_adresse,           /* adresse émetteur */
int *ptr_lg_adresse                     /* taille adresse émetteur */
);
```

Sockets TCP en C-Linux

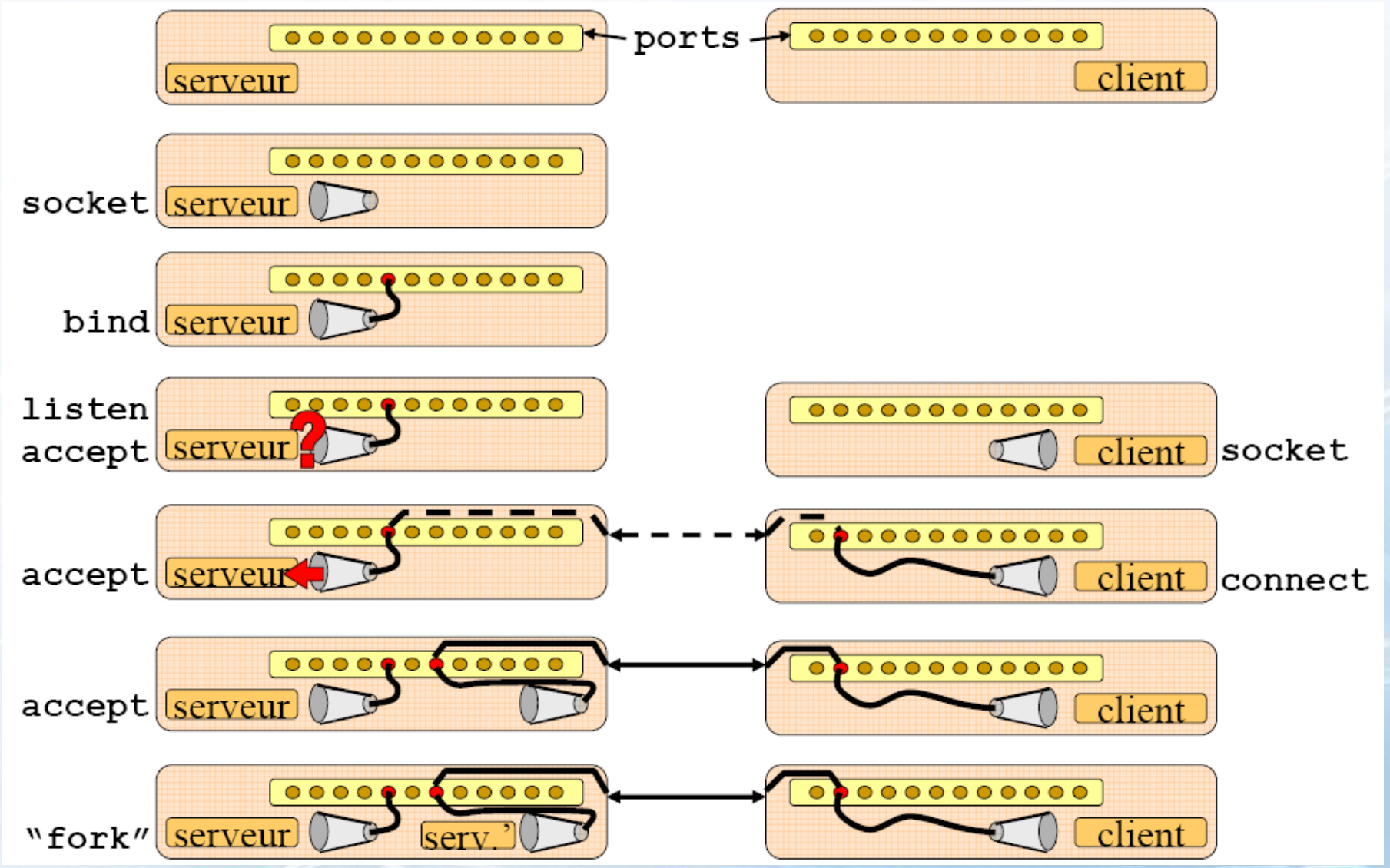
Etapes d'une connexion client-serveur en TCP :

- ✓ le serveur et le client ouvrent chacun une « socket »
- ✓ le serveur la nomme (il l'attache à un de ses ports (un port précis))
  - ✓ le client n'est pas obligé de la nommer (elle sera attachée automatiquement à un port lors de la connexion)
- ✓ le serveur écoute sa socket nommée ***Listen()***
  - ✓ le serveur attend des demandes de connexion
  - ✓ le client connecte sa socket au serveur et à un de ses ports (précis) ***Connect()***

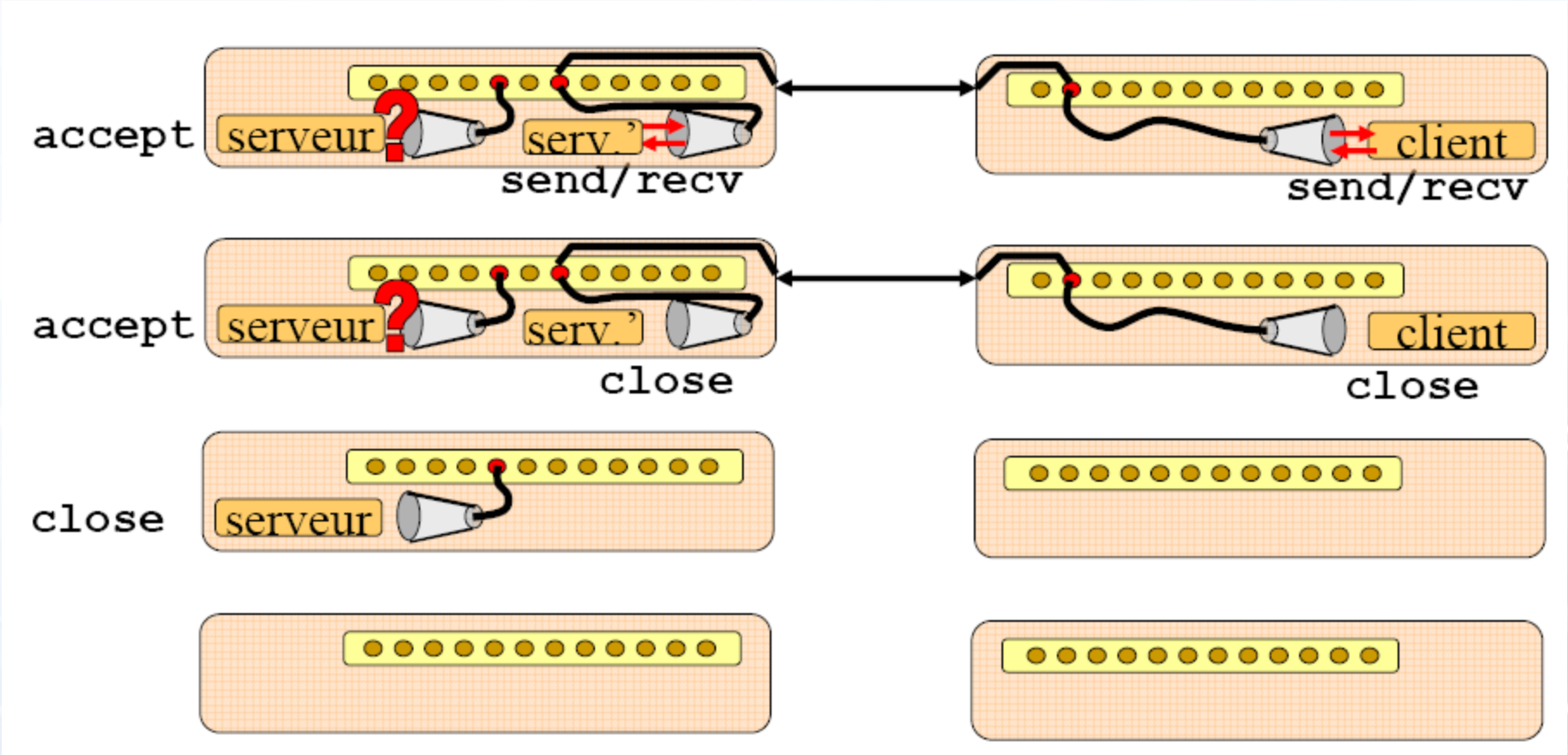
Sockets TCP en C-Linux

- ✓le serveur détecte la demande de connexion
  - ✓une nouvelle socket est ouverte automatiquement ***Accept()***
- ✓le serveur crée un processus pour dialoguer avec le client
  - ✓le nouveau processus continue le dialogue sur la nouvelle socket
  - ✓le serveur attendre en parallèle de nouvelles demandes de connexions
- ✓ finalement toutes les sockets doivent être refermées

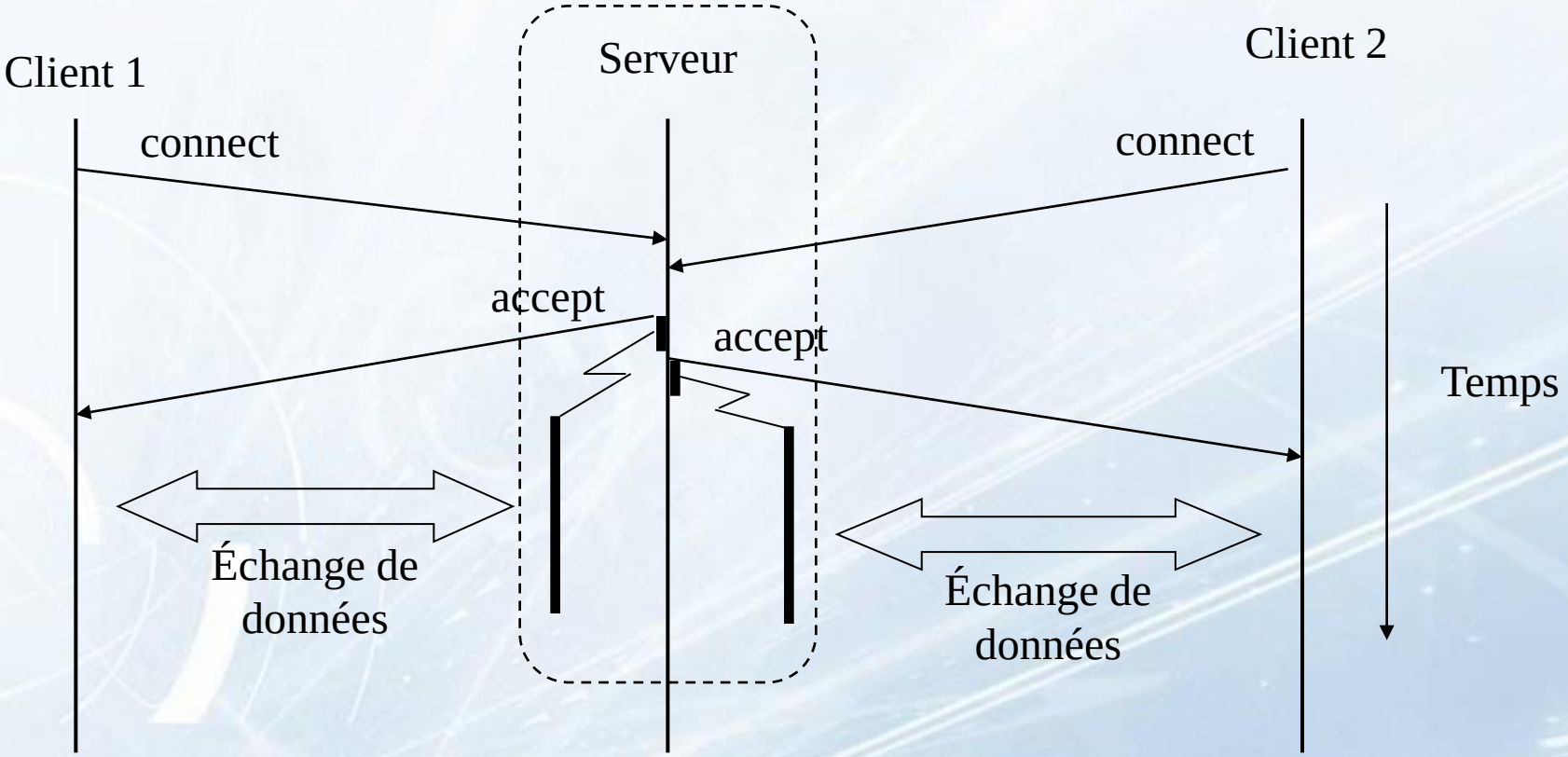
Sockets TCP en C-Linux



Sockets TCP en C-Linux



Sockets TCP en C-Linux



Création de processus fils
Utilisation bloquante de la socket

Sockets fonctions de base

```
int connect(
int descripteur,                    /* Id de socket client */
struct sockaddr *ptr_adresse,       /* adresse serveur */
int lg_adresse                      /* taille adresse serveur */
);
```

```
int listen(
int descripteur,                    /* socket d'écoute */
int nb_pendantes                    /* nb max connexions en */
);
```

```
int accept(
int descripteur,                    /* socket d'écoute */
struct sockaddr *ptr_adresse,       /* adresse client */
int *ptr_lg_adress,                 /* taille adr client */
);
```

Sockets fonctions de base

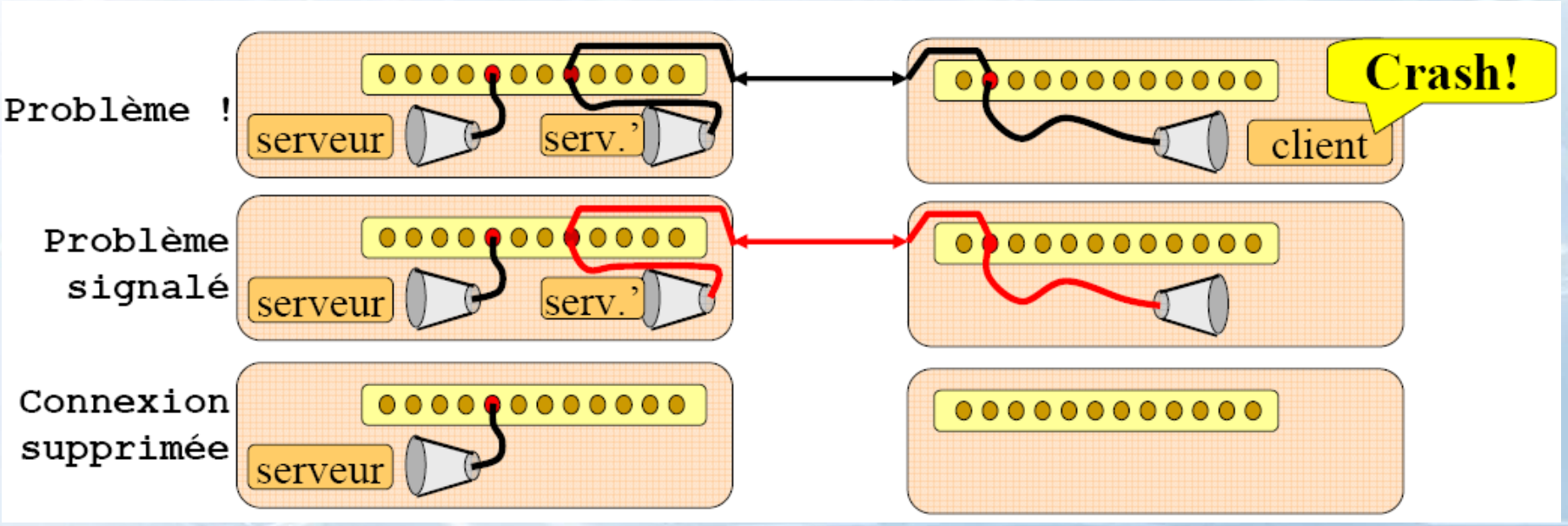
```
int send(
int descripteur,                    /* Id socket émetteur */
void *ptr_caracteres,               /* Adr données à émettre */
size_t nb_caracteres,               /* Nbr caractères à émettre*/
int option                          /* 0 ou MSG_OOB */
);
```

```
int recv(
int descripteur,                    /* Id socket récepteur */
void *ptr_caracteres,               /* Adr stockage des données*/
size_t taille,                      /* Nbr données à recevoir */
int option                          /* 0, MSG_OOB, ou MSG_PEEK */
);
```

Sockets TCP en C-Linux

Détection de fin de connexion

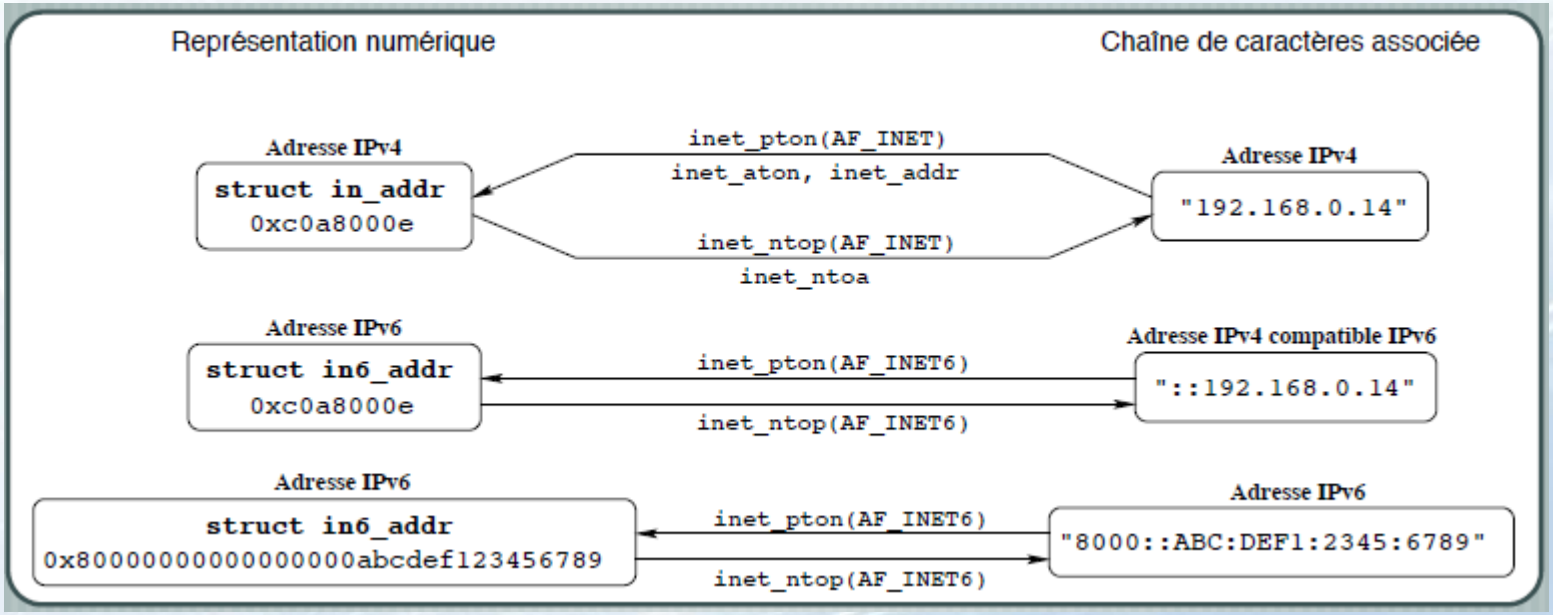
- ✓ Protocole de maintien de connexion
- ✓ ***Si le processus d'une extrémité meurt l'autre en est averti***
- ✓ ***Si le client meurt la nouvelle socket automatiquement ouverte sur le serveur est détruite***



Le problème de la langue

- Données ne sont pas représentées de la même façon suivant processeurs
  - ==> Il faut passer par un traducteur avant de les envoyer sur réseau
- *short int htons (short int x)*
  - retourne l'entier court (2 octets) à la norme réseau de x (passé en norme machine) ===> Home to Network
- *short int htonl (short int x)*
  - retourne l'entier long (4 octets) à la norme réseau de x
- *short int ntohs (short int x)*
  - Retourne l'entier court (2 octets) à la norme machine de x (passé en norme réseau) ===> Network to Host
- *short int ntohl (short int x)*
  - retourne l'entier long (4 octets) à la norme réseau de x

Le problème de la langue



Socket: Accès à son @ et numéro port

- **Récupération de sa propre adresse**
  - Permet de définir la socket locale
  - *int* ***gethostname*** *(*char **nom,* int *longueur_nom)*
    - renseigne le nom de la machine sur laquelle s'exécute la procédure
  - On utilise ensuite ***gethostbyname*** pour avoir l'adresse associée

- **Récupération d'un numéro de port alloué dynamiquement**
  - *int* ***getsockname****(*int *socket,* struct *sockaddr_in *p_ad_s,* int **len*)
    - ****len*** doit contenir la longueur de la structure ***sockaddr_in***

Exemples de programmation en C-Linux

Serveur "UDP/TCP" Echo