

# ARCHITECTURE CLIENT/SERVEUR

- UN MODE DE COMMUNICATION ENTRE PLUSIEURS COMPOSANTS D'UN RÉSEAU.
- CHAQUE ENTITÉ EST CONSIDÉRÉE COMME UN CLIENT OU UN SERVEUR.
- CHAQUE CLIENT PEUT ENVOYER DES REQUÊTES À UN SERVEUR.
- LE MODÈLE CLIENT/SERVEUR EST UN MODÈLE D'ARCHITECTURE LOGICIELLE QUI PERMET D'ORGANISER DES APPLICATIONS DISTRIBUÉES.
- LES SERVICES INTERNET SONT CONÇUS SELON L'ARCHITECTURE CLIENT/SERVEUR.

# LE SERVEUR

- UN SERVEUR EST UN PROGRAMME QUI OFFRE UN SERVICE SUR LE RÉSEAU.
- UN SERVEUR EST INITIALEMENT PASSIF, IL ATTEND (À L'ÉCOUTE), PRÊT À RÉPONDRE AUX REQUÊTES DES CLIENTS.
- LA RÉPONSE D'UNE REQUÊTE POUVANT ÊTRE SYNCHRONE OU ASYNCHRONE.
- UN SERVEUR EST GÉNÉRALEMENT CAPABLE DE SERVIR PLUSIEURS CLIENTS SIMULTANÉMENT

## EXEMPLES DE SERVEURS

- SERVEUR DE FICHIERS
- SERVEUR D'IMPRESSION
- SERVEUR DE CALCUL
- SERVEUR D'APPLICATIONS
- SERVEUR DE BASES DE DONNÉES
- SERVEUR DE NOMS (ANNUAIRE DES SERVICES)....

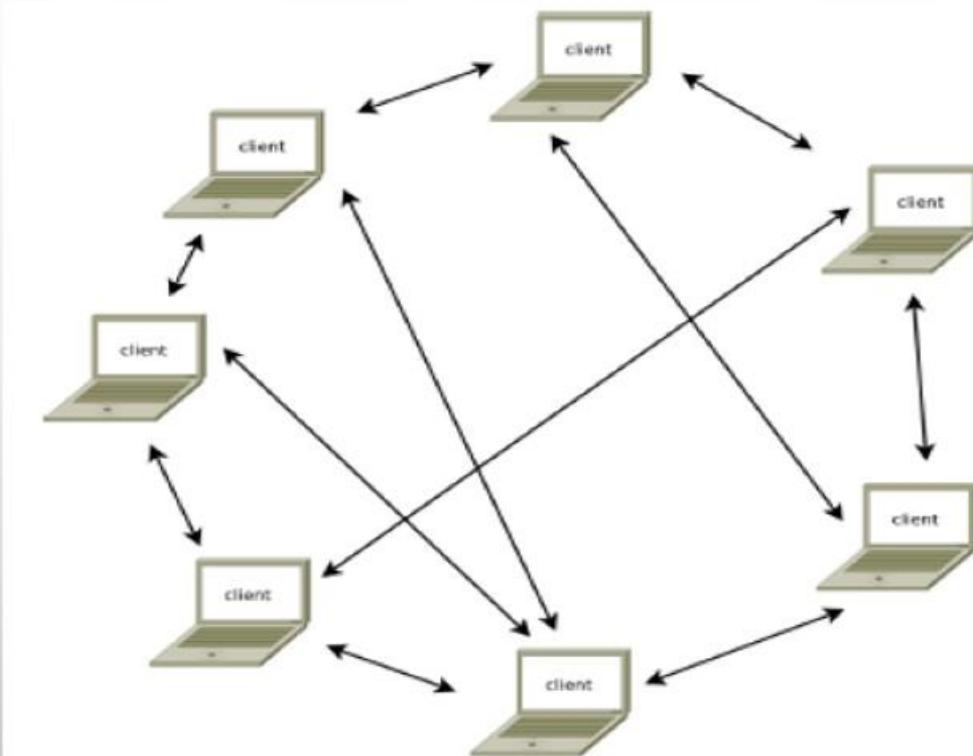
# LE CLIENT

- UN LOGICIEL CLIENT EST UN PROGRAMME QUI UTILISE LE SERVICE OFFERT PAR UN SERVEUR.
- IL EST D'ABORD ACTIF (OU MAÎTRE), IL ENVOIE DES REQUÊTES AU SERVEUR, IL ATTEND ET REÇOIT LES RÉPONSES DU SERVEUR.

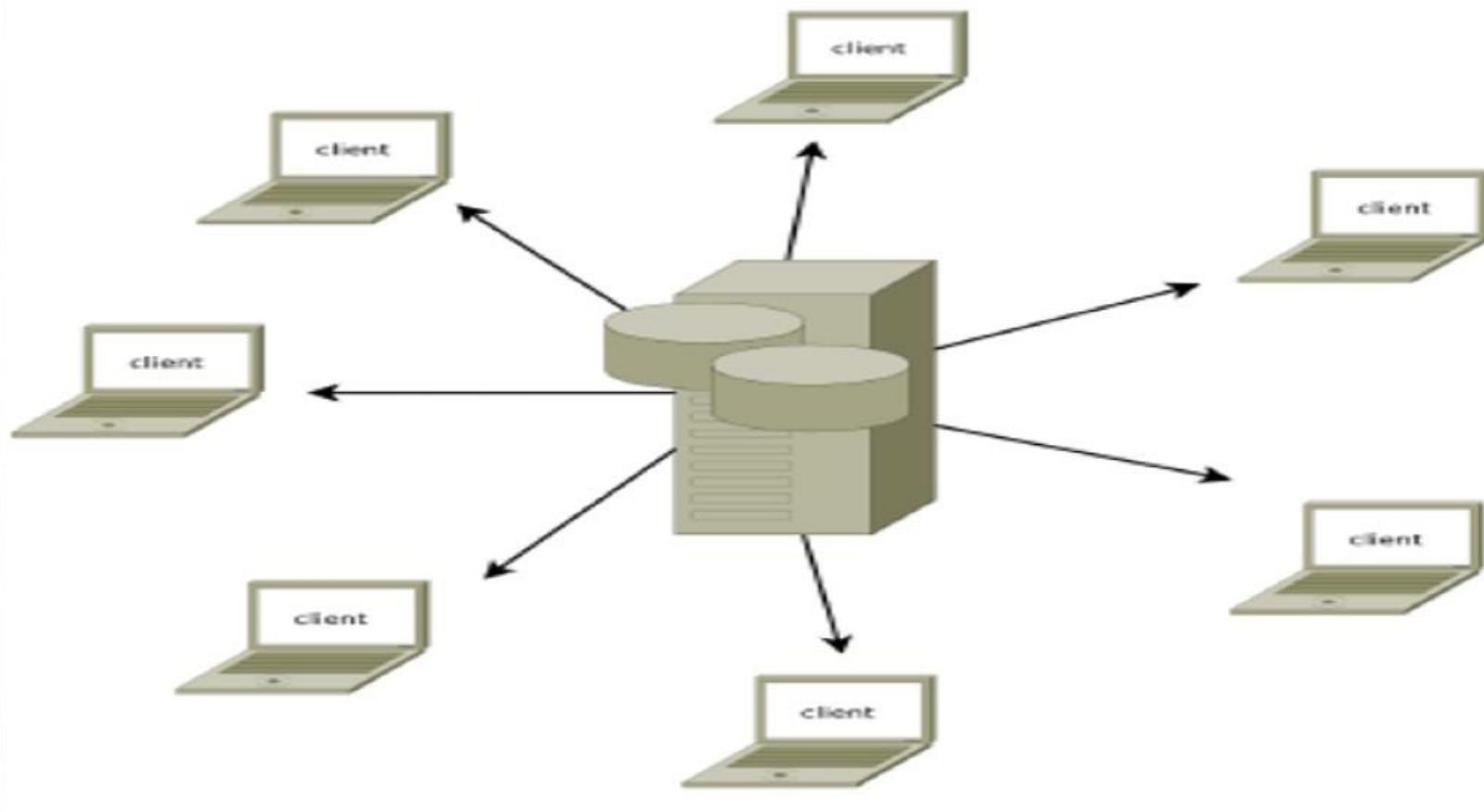
# ARCHITECTURE P2P

RÉSEAU PAIR À PAIR OU (PEER-TO-PEER - P2P).

CHAQUE ORDINATEUR EST À LA FOIS CLIENT ET SERVEUR



# ARCHITECTURE CLIENT/SERVEUR



# CARACTÉRISTIQUES DE L'ARCHITECTURE CLIENTS/SERVEURS

- **SERVEUR**

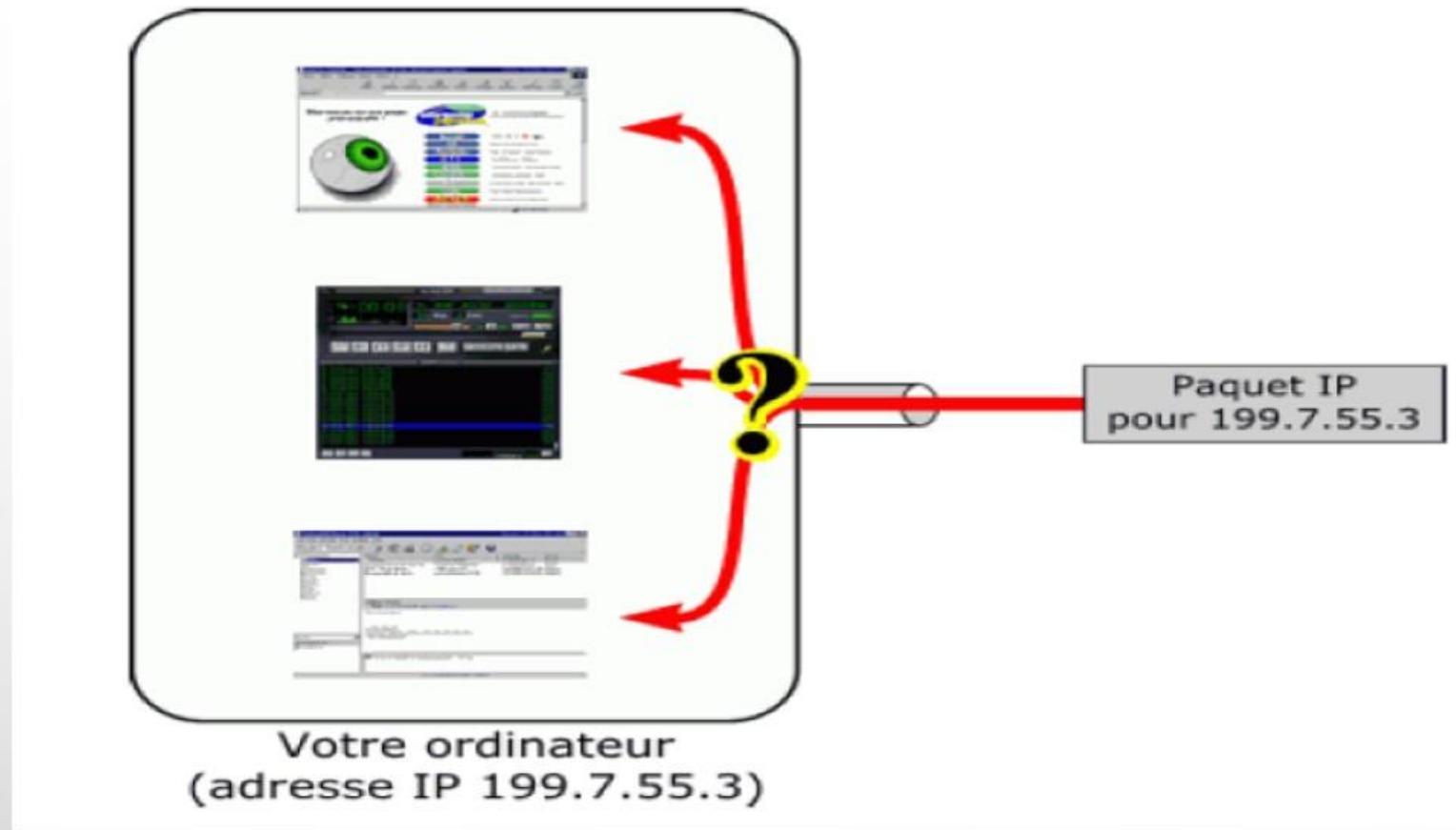
- INITIALEMENT **PASSIF** EN ATTENTE D'UNE REQUÊTE.
- À L'**ÉCOUTE**, PRÊT À RÉPONDRE AUX **REQUÊTES CLIENTS**.
- QUAND UNE REQUÊTE LUI PARVIENT IL LA **TRAITE** ET ENVOIE LA RÉPONSE

- **CLIENT**

- **ACTIF** EN PREMIER.
- ENVOIE DES **REQUÊTE AU SERVEUR**.
- ATTEND ET **REÇOIT LES RÉPONSE** DU SERVEUR.

LE CLIENTS ET LE SERVEUR DOIVENT UTILISER LE MÊME PROTOCOLE  
UN SERVEUR PEUT RÉPONDRE À PLUSIEURS CLIENTS EN SIMULTANÉ

# POURQUOI DES PORTS



des ports pour identifier les services

# POINT DE COMMUNICATION

- UNE APPLICATION FOURNIT UN SERVICE PARTICULIER SUR UNE MACHINE DONNÉE
- LE SERVICE EST IDENTIFIÉ PAR UN **PORT**
- LA MACHINE EST IDENTIFIÉE PAR UNE **ADRESSE**.
- UN COUPLE (ADRESSE,PORT) EST UN **POINT DE COMMUNICATION**
- TOUTE **COMMUNICATION** NE PEUT S'EFFECTUER QU'ENTRE AU MOINS **DEUX POINTS** DE COMMUNICATIONS
- L'**ÉMETTEUR**
- UN OU PLUSIEURS **RECEPTEUR**)

# FONCTIONNEMENT DE L'ARCHITECTURE CLIENTS/SERVEURS

- POUR RECEVOIR DES INFORMATIONS DU SERVEUR LE CLIENT **ÉMET UNE REQUÊTE AU SERVEUR** PASSANT PAR UN PORT DU PC (EXEMPLE : PORT 25 POUR LES MAILS, PORT 80 POUR LE WEB ET 21 POUR LE FTP).
- LE SERVEUR LUI ENVOI ENSUITE LES INFORMATIONS GRÂCE À L'**ADRESSE IP** DE LA MACHINE CLIENTE.
- LE CLIENT REÇOIT ET AFFICHE LES INFORMATION EN PROVENANCE DU SERVEUR

# EXEMPLES D'ARCHITECTURE CLIENTS/SERVEURS

- LA CONSULTATION DE PAGES SUR UN SITE WEB
- LES COURRIELS
- UNE BASE DE DONNÉES CENTRALISÉE.

## MISE EN ŒUVRE DE L'ARCHITECTURE CLIENT-SERVEUR

- HAUT NIVEAU : UTILISATION DIRECTE DU TRANSPORT : SOCKETS (CONSTRUIT SUR TCP OU UDP)
  - ❖ EXEMPLE : UTILISATION L'INTERFACE JAVA DES **SOCKETS** (PACKAGE **JAVA.NET**) OFFRE UN ACCÈS SIMPLE AUX **SOCKETS**
- BAS NIVEAU : INTÉGRATION DANS UN LANGAGE DE PROGRAMMATION
  - ❖ EXEMPLE : SOCKETS EN LANGAGE C.

# IDENTIFICATION DES MACHINES

## UN **NOM INTERNET** (PAS STRICTEMENT NÉCESSAIRE)

- PAR EXEMPLE : [WWW.SAMSUNG.COM](http://WWW.SAMSUNG.COM).
- UNE MACHINE PEUT POSSÉDER PLUSIEURS NOMS

## UNE **ADRESSE INTERNET** (TOUTE MACHINE CONNECTÉE AU RÉSEAU EN POSSÈDE UNE)

- EN RÉALITÉ IL S'AGIT DE L'ADRESSE D'UN DISPOSITIF RÉSEAU SUR UNE MACHINE (EX : 211.45.27.202)
  - UNE ADRESSE PAR DISPOSITIF
  - MAIS POSSIBLEMENT PLUSIEURS DISPOSITIFS POUR UNE MACHINE
- 
- IL EXISTE AUJOURD'HUI DEUX TYPES D'ADRESSES : IPV4 ET IPV6 (RESPECTIVEMENT 4 ET 8 OCTETS)

# PORT DE COMMUNICATION

- TOUTE COMMUNICATION NÉCESSITE L'UTILISATION D'UN PORT
- LES NUMÉROS DE PORT SONT REPRÉSENTÉES SUR 16 BITS CE QUI FAIT QUE CHAQUE MACHINE POSSÈDE DONC 65 536 PORTS.

IL EXISTE ESSENTIELLEMENT TROIS TYPES DE PORTS :

- **LES PORTS RECONNUS**, DE NUMÉRO COMPRIS ENTRE 0 ET 1023 : DES PORTS GÉNÉRALES PAR EXEMPLE LE PORT 80 EST UTILISÉ PAR LE PROTOCOLE HTTP, LE PORT 21 PAR FTP, ...
- **LES PORTS RÉSERVÉS**, DE NUMÉRO COMPRIS ENTRE 1024 ET 49151.
- **LES PORTS LIBRES**, DE NUMÉRO COMPRIS ENTRE 49 152 ET 65 535.

# LES PORTS RECONNUS (WELL-KNOWN PORTS)

SONT UTILISÉS PAR DES SERVICES RÉSEAU D'USAGE GÉNÉRAL ET COMMUN :

- ❖ LE 80 POUR HTTP
- ❖ LES PORTS 20 ET 21 POUR FTP.
- ❖ LE 25 POUR SMTP

CE QUI SIGNIFIE QUE POUR ÉTABLIR UNE CONNEXION AVEC UN SERVEUR WEB, IL FAUT S'ADRESSER AU PORT 80 DE LA MACHINE CONCERNÉE

EXTRAIT DU FICHIER /ETC/SERVICES

# EXTRAIT DU FICHIER /ETC/SERVICES

Extrait d'un fichier « /etc/services » :

```
...
chargen      19/tcp    ttyst source      #Character Generator
chargen      19/udp    ttyst source      #Character Generator
ftp-data     20/tcp    #File Transfer [Default Data]
ftp-data     20/udp    #File Transfer [Default Data]
ftp          21/tcp    #File Transfer [Control]
ftp          21/udp    #File Transfer [Control]
ssh          22/tcp    #Secure Shell Login
ssh          22/udp    #Secure Shell Login
telnet       23/tcp
telnet       23/udp
smtp         25/tcp    mail           #Simple Mail Transfer
smtp         25/udp    mail           #Simple Mail Transfer
...
...
```

## LES PORTS RÉSERVÉS (REGISTERED PORTS)

CORRESPONDENT À DES SERVICES D'USAGE MOINS GÉNÉRAL (SOUVENT DES SERVICES PROPRIÉTAIRES)

PAR EXEMPLE :

LE PORT 3306 UTILISÉ PAR LES BASES DE DONNÉES MYSQL.

LE PORT 17500 POUR DROPBOX.

LE PORT 1863 MSN MESSENGER

- N'IMPORTE QUELLE APPLICATION PEUT LES UTILISER

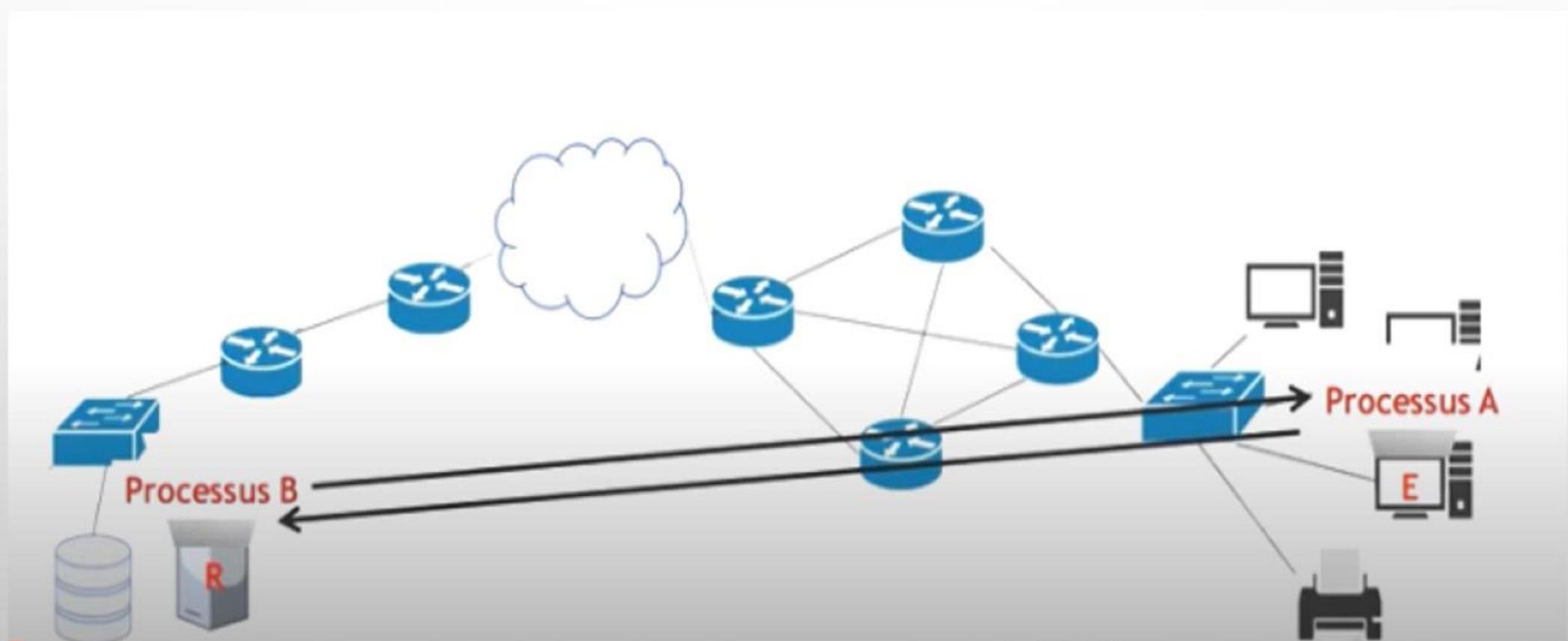
## LES PORTS LIBRES

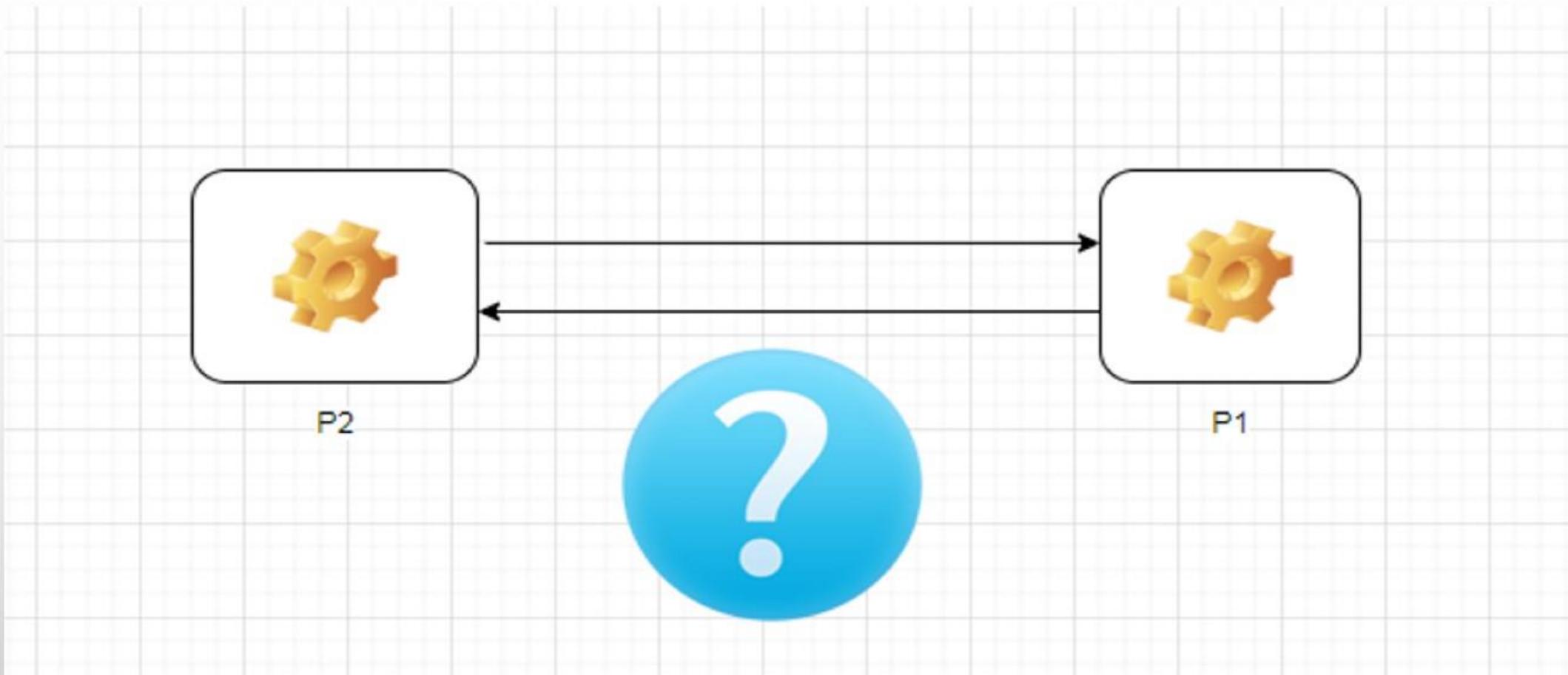
- LES **PORTS LIBRES** (DYNAMIC, PRIVATE OR EPHEMERAL PORTS)

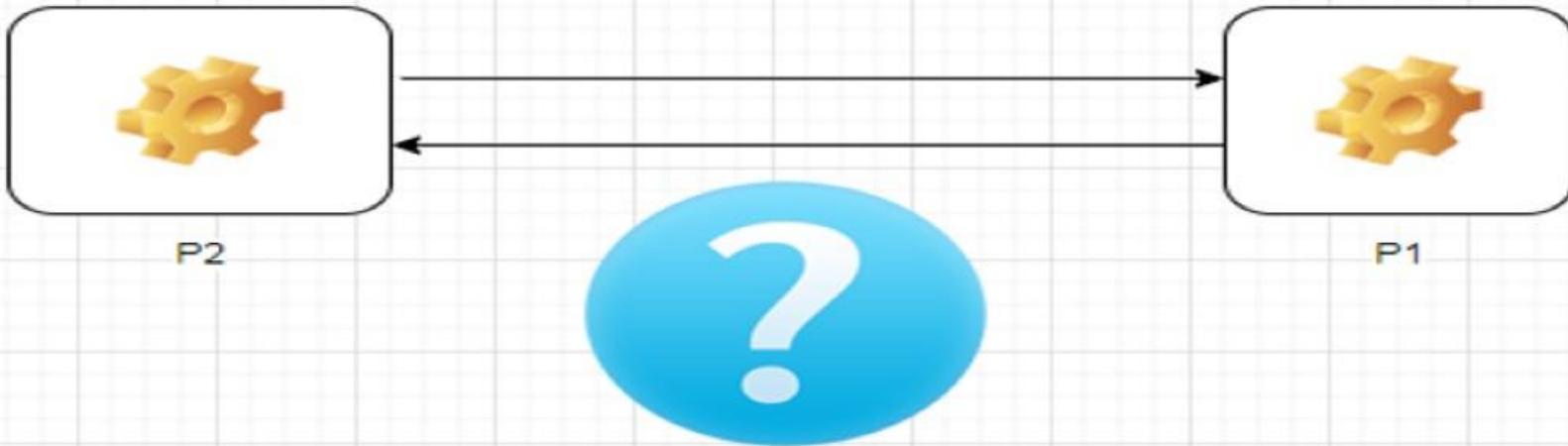
NORMALEMENT UTILISÉS POUR DES DURÉES LIMITÉES...

POUR LES SERVICES QUE VOUS DÉVELOPPEREZ, VOUS UTILISEREZ CES PORTS

# COMMUNICATION RÉPARTIE







Service de communication :

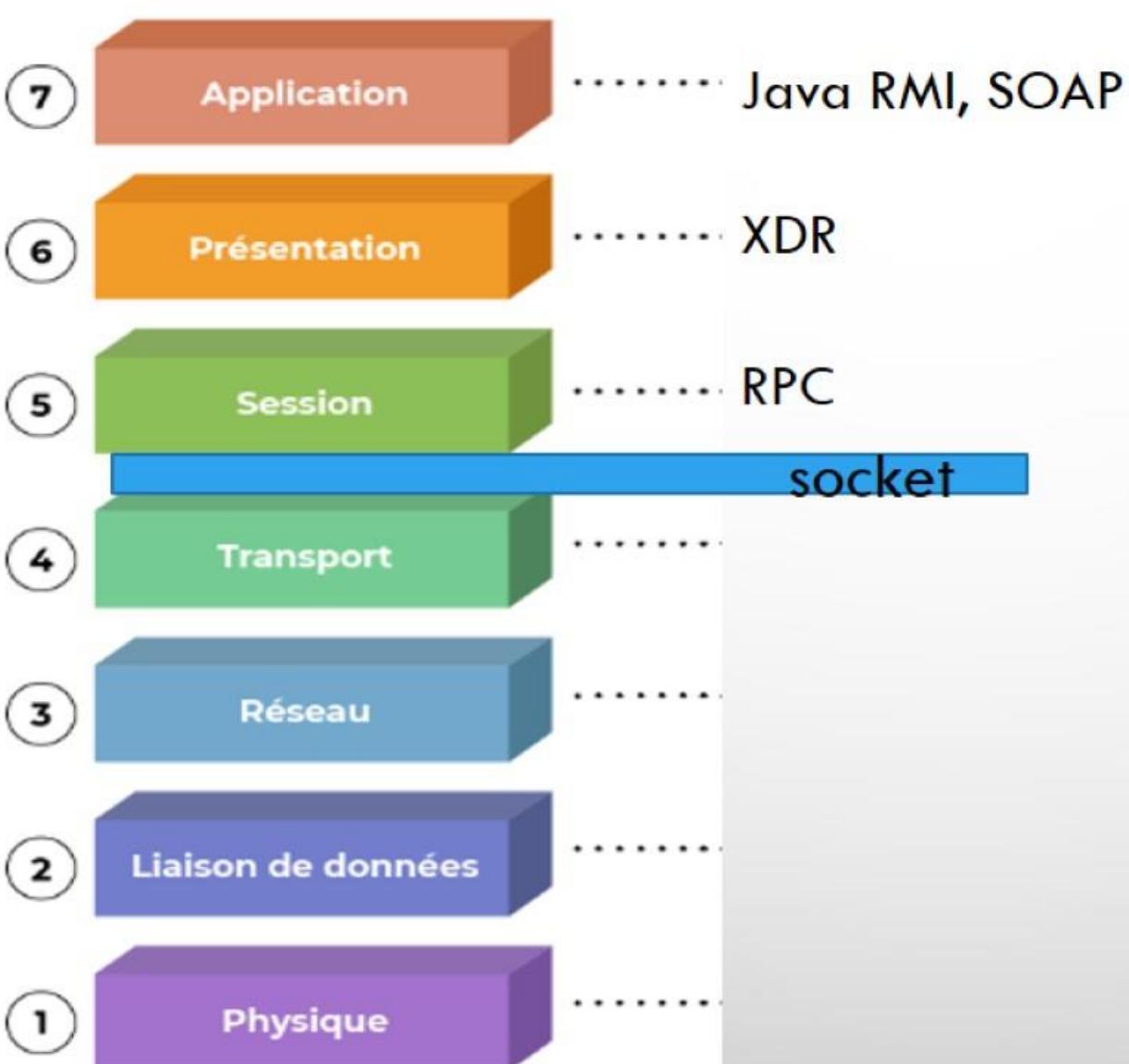
Envoi de messages : socket

Appel de procédure a distance : java RMI, CORBA

Invocation de service : SOAP

Etc....

# SERVICE DE COMMUNICATION :



# INTRODUCTION AUX SOCKETS

- **SOCKETS**, EST UN [ENSEMBLE NORMALISÉ DE FONCTIONS](#) DE COMMUNICATION LANCÉ PAR L'[UNIVERSITÉ DE BERKELEY](#) AU DÉBUT DES ANNÉES 80 POUR LEUR [BERKELEY SOFTWARE DISTRIBUTION](#) (BSD). WIKIPÉDIA
- STANDARD AUJOURD'HUI.
- CETTE [INTERFACE DE PROGRAMMATION](#) EST PROPOSÉE DANS QUASIMENT TOUS LES LANGAGES DE PROGRAMMATION POPULAIRES (JAVA, C#, C++, ...)

# SOCKET

- SOCKET = PRISE DE COMMUNICATION
- SOCKET : EST UN POINT DE COMMUNICATION PAR LEQUEL UN PROCESSUS PEUT ÉMETTRE OU RECEVOIR DES INFORMATIONS.
- SOCKET = UNE **API** (APPLICATION PROGRAMMING INTERFACE OU « INTERFACE DE PROGRAMMATION D'APPLICATION ») = ENSEMBLE DE PRIMITIFS QUI PERMETTENT DE GÉRER L'ÉCHANGE DE DONNÉES ENTRE PROCESSUS

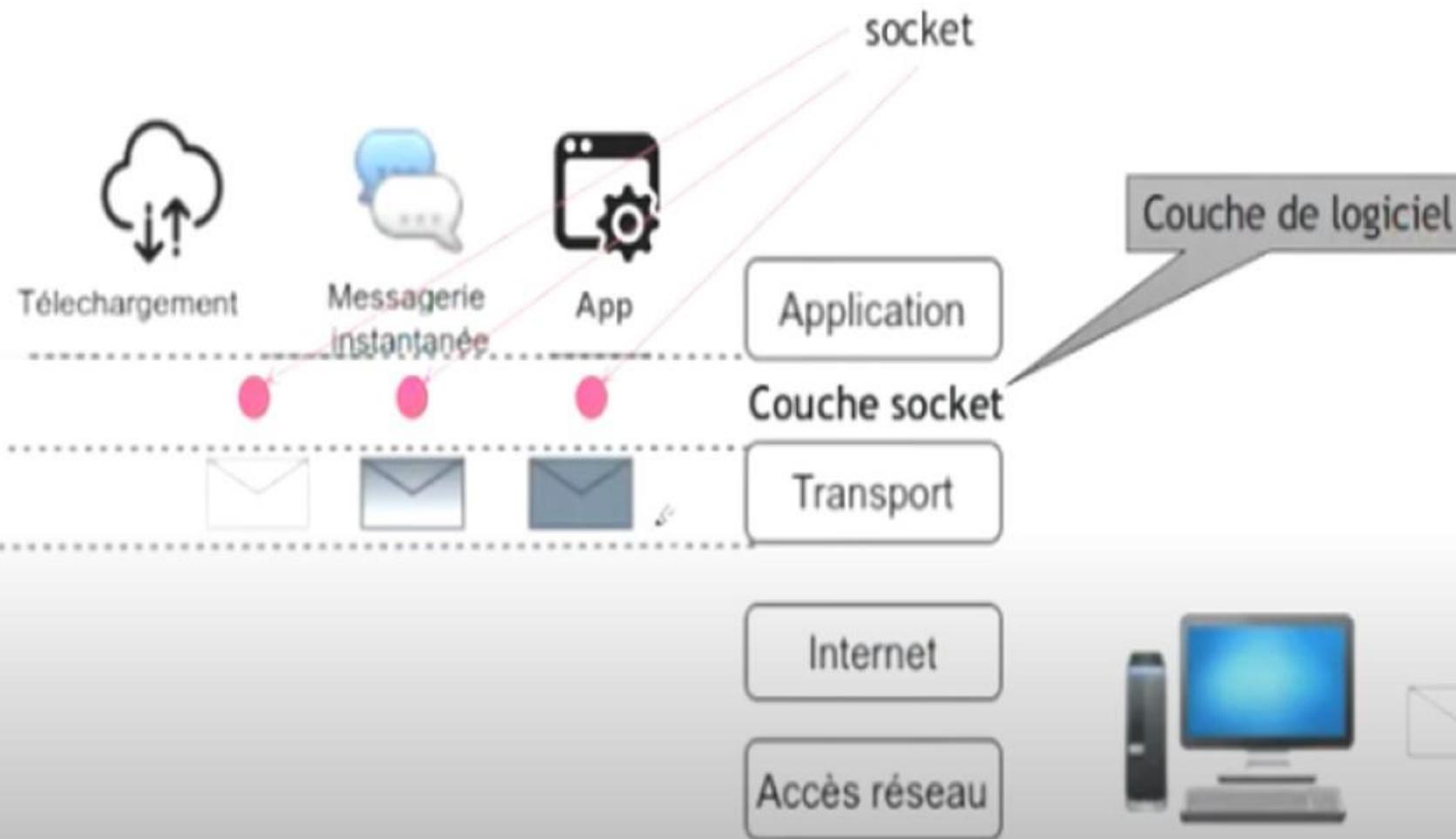


# SOCKET

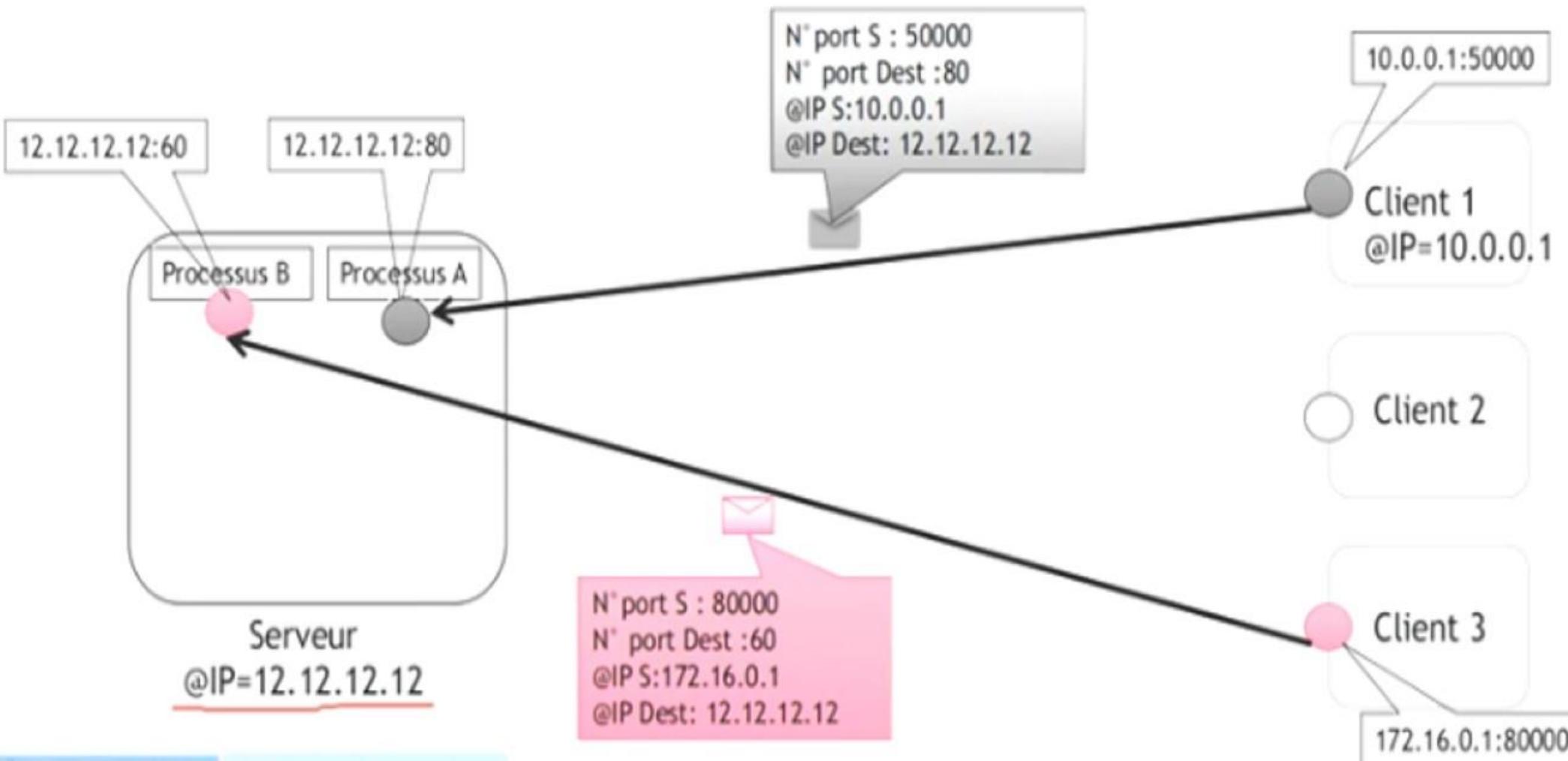
- UN SOCKET S'UTILISE COMME UN FICHIER :
  1. CRÉATION/DÉFINITION/OUVERTURE : SOCKET ET LA CANAL DE COMMUNICATION
  2. COMMUNICATION
  3. FERMETURE/LIBÉRATION.
- BASÉ SUR L'ARCHITECTURE CLIENT SERVEUR.



# SOCKET



# SOCKET



# MODES DE COMMUNICATION

Application

Socket

Transport

Internet : IP

Application

Socket  
mode  
connecté

Socket  
mode non  
connecté

TCP

UDP

transport

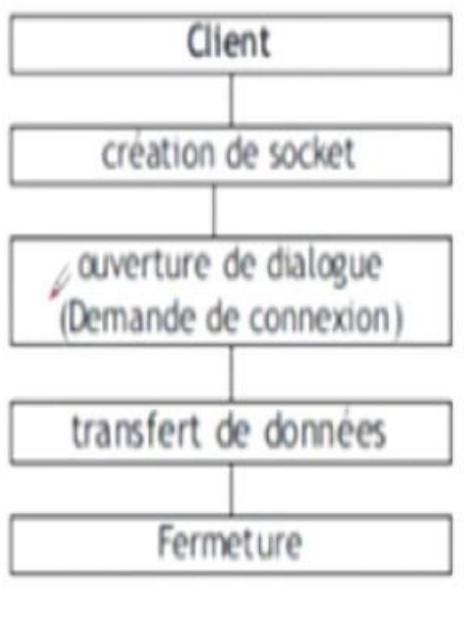
Internet : IP

# MODES DE COMMUNICATION

IL EXISTE AU MOINS DEUX GRANDS MODES DE COMMUNICATION :

- **PAR PAQUET (DATAGRAM) :**
- DANS CE MODE ON EST PAS CONNECTÉ
- ON EST JUSTE ATTEIGNABLE
- **EN FLUX (STREAM) :** DANS CE MODE ON EST CONNECTÉ

# MODES DE COMMUNICATION



Mode connecté

Mode non connecté

# MODE DATAGRAMME UDP

DANS LE MODE DATAGRAMME :

- IL N'EXISTE PAS D'ORDRE DANS LA DÉLIVRANCE DES PAQUETS.
- UN PAQUET DÉLIVRÉ EN PREMIER PEUT ARRIVER EN DERNIER.
- IL N'EXISTE PAS NON PLUS DE FIABILITÉ : UN PAQUET ENVOYÉ PEUT ÊTRE PERDU.
- INTÉRÊT : SOUPLE ET LÉGER...

## MODE FLUX TCP

DANS LE MODE **FLUX** :

- LES INFORMATIONS SONT REÇUES DANS L'ORDRE EXACT DE LEUR ÉMISSION
- IL N'Y A PAS DE PERTE.
- INCONVÉNIENT : NÉCESSITE L'ÉTABLISSEMENT D'UNE CONNEXION ET CONSOMME DONC DES RESSOURCES POUR SA GESTION

# LANGAGE DE PROGRAMMATION

- C/C++ , JAVA, JAVASCRIPT , C SHARP, PHP, .....

# LES TYPES POSIX

Type de données	Description	Fichier d'en-tête
<code>int8_t</code>	Entier 8-bit signé	<code>&lt;sys/types.h&gt;</code>
<code>uint8_t</code>	Entier 8-bit sans signe	<code>&lt;sys/types.h&gt;</code>
<code>int16_t</code>	Entier 16-bit signé	<code>&lt;sys/types.h&gt;</code>
<code>uint16_t</code>	Entier 16-bit sans signe	<code>&lt;sys/types.h&gt;</code>
<code>int32_t</code>	Entier 32-bit signé	<code>&lt;sys/types.h&gt;</code>
<code>uint32_t</code>	Entier 32-bit sans signe	<code>&lt;sys/types.h&gt;</code>
<code>sa_family_t</code>	Famille d'adresse	<code>&lt;sys/socket.h&gt;</code>
<code>socklen_t</code>	Entier 32-bit sans signe	<code>&lt;sys/socket.h&gt;</code>
<code>in_addr_t</code>	Adresse IPv4, <code>uint32_t</code>	<code>&lt;netinet/in.h&gt;</code>
<code>in_port_t</code>	Port TCP ou UDP, <code>uint16_t</code>	<code>&lt;netinet/in.h&gt;<sup>34</sup></code>

# LITTLE ENDIAN ET BIG ENDIAN

LES HÔTES DIFFÉRENT DANS LA FAÇON DONT ILS STOCKENT LES DONNÉES

- OCTET3, OCTET2, OCTET1, OCTET 0 LITTLE ENDIAN
- OCTET0, OCTET1, OCTET2, OCTET 3 BIG ENDIAN
- LES TYPES ENTIERS DE 16 BITS ET ENTIERS DE 32 BITS APPELÉS RESPECTIVEMENT (SHORT ET LONG) :
- UTILISEZ **htons ()** ET **htonl ()** POUR CONVERTIR EN ORDRE D'OCTETS DE RÉSEAU
- UTILISEZ  **ntohs ()** ET  **ntohl ()** POUR CONVERTIR EN ORDRE D'HÔTE
- **htonl** ET  **ntohl** PERMETTENT LA MANIPULATION DES ADRESSES, ET **htons** ET  **ntohs** PERMETTENT CELLES DES NUMÉROS DE PORT :

```
#include <arpa/inet.h>

short ntohs(short); /* NETWORK TO HOST SHORT */
short htons(short); /* HOST TO NETWORK SHORT */
long ntohl(long); /* NETWORK TO HOST LONG */
long htonl(long); /* HOST TO NETWORK LONG */
```

- EN JAVA, CE N'EST PAS NÉCESSAIRE, L'API FAIT LA CONVERSION ELLE-MÊME CAR LE CODAGE DES ENTIERS EN JAVA EST STANDARDISÉ

# LES FICHIERS D'INCLUSION (LINUX)

- LA DÉFINITION DES STRUCTURES ET DÉCLARATIONS DE FONCTIONS DES SOCKETS SONT CONTENUES DANS LES FICHIERS D'INCLUDE :

```
#include <sys/types.h> /* TYPES DE BASE */  
  
#include <sys/socket.h> /* DEFINITIONS DE BASE DES SOCKETS */  
  
#include <netdb.h> /* FONCTIONS D'INFORMATION SUR LE RESEAU */  
  
#include< netinet/in.h> /* SOCKADDR_IN */  
  
#include <arpa/inet.h> /* FONCTIONS DE MANIPULATION D'ADRESSE */
```

# ADRESSE DE SOCKET EN C

- CARACTÉRISE PAR :
  - UNE ADRESSE INTERNET
  - NUMÉRO DE PORT
  - TYPE DE COMMUNICATION

# LES SOCKETS

POUR ÉTABLIR UNE COMMUNICATION VERS UNE MACHINE DISTANTE, IL FAUT :

1. S'ATTRIBUER (OU LAISSER LE SOIN AU SYSTÈME D'ATTRIBUER) UN NUMÉRO DE PORT
2. DÉTERMINER L'ADRESSE INTERNET DE LA MACHINE AVEC LAQUELLE LES ÉCHANGES VONT S'EFFECTUER
3. CONNAÎTRE OU DÉTERMINER LE NUMÉRO DE PORT DU SERVICE DISTANT.

# SOCKET UDP

- COMMUNICATION NON FIALE
- PAS DE CONNEXION PERMANENTE ENTRE LE CLIENT ET LE SERVEUR
- TYPE : SOCK\_DGRAM

Serveur

## ► Receveur

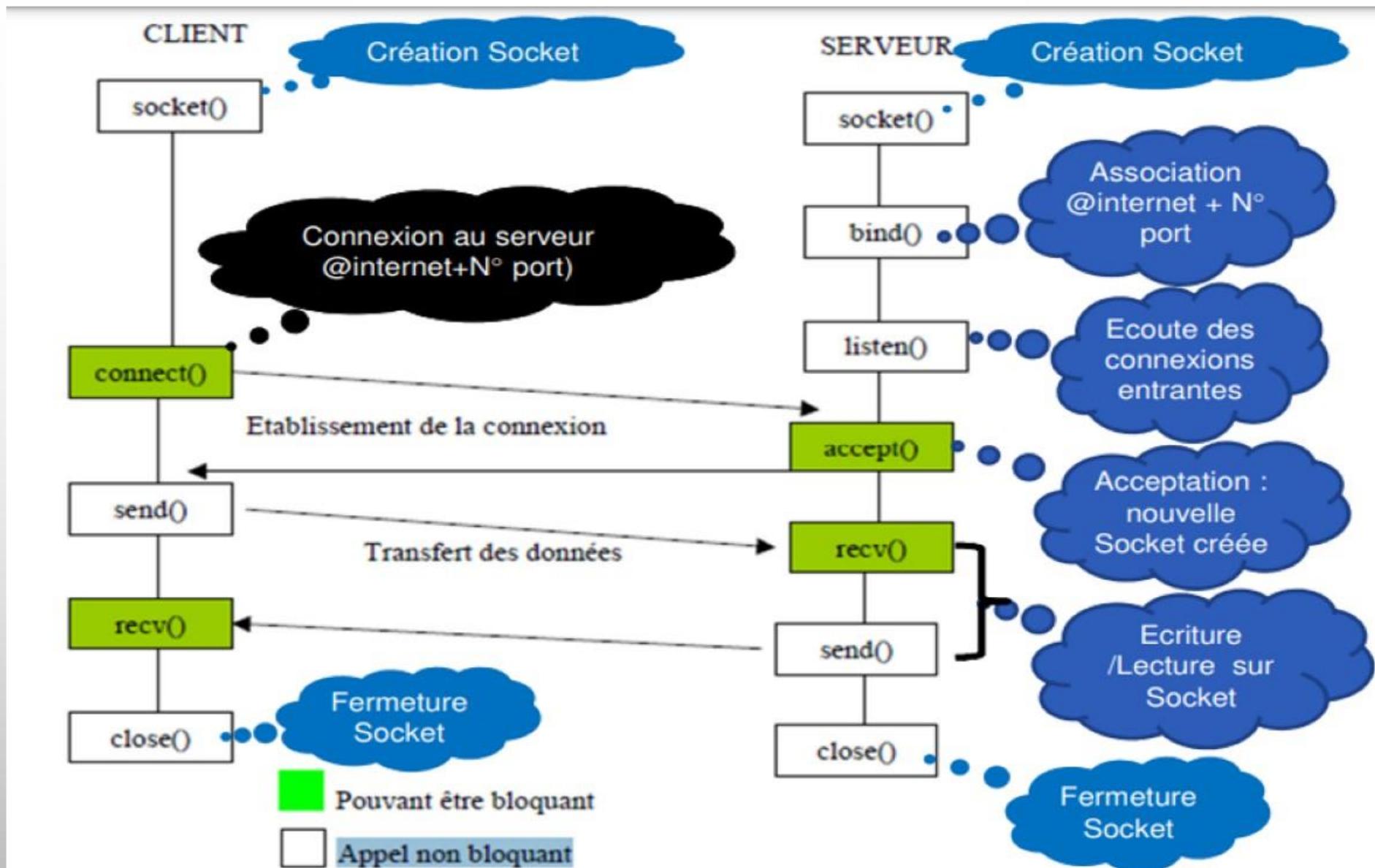
- socket
- bind
- sendto/recvfrom
- close

client

## ► Emetteur

- socket
- sendto/recvfrom
- close

# SOCKET TCP



# SOCKET

POUR CRÉER UNE SOCKET, ON DOIT PRÉCISER LE DOMAINE DE TRAVAIL :

1. PF\_INET TCP/IP (IPV4)
2. PF\_INET6 TCP/IP (IPV6)
3. PF\_UNIX COMMUNICATION UNIX
4. PF\_APPLETALK PROTOCOLE APPLE
5. PF\_IPX IPX PROTOCOL NOVELL
6. .....



# CRÉATION DES SOCKETS

POUR CRÉER UNE SOCKET, ON DOIT PRÉCISER :

2. LE TYPE DE PROTOCOLE DE COMMUNICATION :

- **SOCK\_DGRAM** : PROTOCOLE DE COMMUNICATION NON CONNECTÉ (UDP DANS LE DOMAINE PF\_INET).
- **SOCK\_STREAM** : PROTOCOLE DE COMMUNICATION NON CONNECTÉ , ENVOI DE FLUX D'OCTETS (TCP DANS LE DOMAINE PF\_INET).
- - **SOCK\_RAW**: UTILISÉ POUR LE DÉVELOPPEMENT DE NOUVEAUX PROTOCOLES DE COMMUNICATION,
- 3. **PROTOCOLE** PERMET DE SPÉCIFIER UN PROTOCOLE PERMETTANT DE FOURNIR LE SERVICE DÉSIRÉ. DANS LE CAS DE TCP/IP IL N'EST PAS UTILE, ON LE METTRA AINSI TOUJOURS À 0
  - A. IPPROTO\_TCP POUR LE TCP
  - B. IPPROTO\_UDP POUR LE UDP
  - C. 0 LE SYSTÈME CHOISIT LE PROTOCOLE (SI PF\_INET + SOCK\_STREAM = . IPPROTO\_TCP) (SI PF\_INET + SOCK\_DGRAM = . IPPROTO\_UDP)

- `int socket(famille , type , protocole)`

PF\_INET  
PF\_INET6  
PF\_UNIX

AF\_xxx = Adress Family  
PF\_xxx = Protocol Family

AF\_INET  
AF\_INET6  
AF\_UNIX

Struct sockaddr {  
.....  
**u\_short sa\_family } •**

- `bind(int descripteur , sockaddr localaddr ,int addrlen).`

```
File Edit View Search Terminal Help
#define AF_RXRPC          33      /* RxRPC sockets
#define AF_ISDN           34      /* mISDN sockets
#define AF_PHONET          35      /* Phonet sockets
#define AF_IEEE802154       36      /* IEEE802154 sockets
#define AF_CAIF            37      /* CAIF sockets
#define AF_ALG              38      /* Algorithm sockets
#define AF_NFC              39      /* NFC sockets
#define AF_VSOCK            40      /* vSockets
#define AF_KCM              41      /* Kernel Connection Multiplexor */
#define AF_QIPCRTR          42      /* Qualcomm IPC Router */
#define AF_SMC              43      /* smc sockets: reserve number for
                                * PF_SMC protocol family that
                                * reuses AF_INET address family
                                */

#define AF_MAX              44      /* For now.. */

/* Protocol families, same as address families. */
#define PF_UNSPEC           AF_UNSPEC
#define PF_UNIX              AF_UNIX
#define PF_LOCAL             AF_LOCAL
#define PF_INET              AF_INET
```

# LA FONCTION SOCKET() (TCP ET UDP)

- PERMET LA CRÉATION D'UN SOCKET.
- **int socket(famille , type , protocole)**
- **FAMILLE** REPRÉSENTE LA FAMILLE DE PROTOCOLE UTILISÉ
- **TYPE** LE TYPE DE PROTOCOLE DE COMMUNICATION **SOCK\_STREAM, SOCK\_DGRAM**  
**PROTOCOLE** : DANS LE CAS DE TCP/IP IL N'EST PAS UTILE, ON LE METTRA AINSI TOUJOURS À 0
- LA FONCTION **SOCKET()** RENVOIE UN ENTIER QUI CORRESPOND À UN **DESCRIPTEUR** DU SOCKET. EN CAS D'ERREUR, LA FONCTION **SOCKET()** RETOURNE -1.
- **Les socket doivent être associées à un point de communication (@+port) = bind**

# EXEMPLE

Coté client

```
int descripteurC = socket(AF_INET, SOCK_STREAM, 0);
```

Coté serveur

```
int descripteurS = socket(AF_INET, SOCK_STREAM, 0);
```

# STRUCTURE sockaddr

- L'INTERFACE SOCKET DÉFINIT UNE **STRUCTURE STANDARD** (SOCKADDR PERMETTANT DE REPRÉSENTER L'ADRESSE D'UNE SOCKET:
- STRUCTURE GÉNÉRIQUE POUR LES SOCKETS

```
struct sockaddr {  
  
    u_char sa_len; // longueur effective de l'adresse.  
  
    u_char sa_family; //famille de protocole (généralement AF_INET pour tcp/ip).  
  
    char sa_data[14]; //l'adresse complete.  
};
```

# STRUCTURE `sockaddr`

- UNE ADRESSE EST STOCKÉE DANS UNE STRUCTURE SOCKADDR
- DÉPEND DE L'IMPLÉMENTATION
- CONTIENT UN CHAMP SA\_FAMILY
- –POUR CHAQUE SA\_FAMILY DIFFÉRENT, UNE SOUS-STRUCTURE :
- `struct sockaddr_in (AF_INET)`
- `struct sockaddr_in6 (AF_INET6)`
- `struct sockaddr_un (AF_UNIX)`

# STRUCTURE sockaddr\_in

- LA **STRUCTURE** UTILISÉ AVEC **TCP/IP** EST UNE ADRESSE **AF\_INET** QUI UTILISENT UNE STRUCTURE **sockaddr\_in** : SOCKET DU DOMAINE IPV4. Domaine internet

```
struct sockaddr_in {  
  
    u_char sin_len; // longueur effective de l'adresse.  
  
    short sin_family; // famille de protocole (AF_INET) .  
  
    u_short sin_port; // numero de port.  
  
    struct in_addr sin_addr; //adresse internet.  
  
};  
  
/* Adresse Internet */  
  
struct in_addr { uint32_t s_addr; /* Adresse dans l'ordre d'octets réseau */ };
```

**structure sockaddr\_in**

<b>sin_len</b>	<b>sin_family</b>	<b>sin_port</b>	<b>sin_addr</b>
<b>sa_len</b>	<b>sa_family</b>	<b>sa_data</b>	

**structure sockaddr**

# LA FONCTION BIND (TCP ET UDP)

- **BINDING (ASSOCIATION)** : PERMET DE LIER LA SOCKET À UN POINT DE COMMUNICATION LOCAL DÉFINI PAR UNE ADRESSE ET UN PORT.

pointeur sur l'adresse à affecter au socket

- **bind(int descripteur , sockaddr \*localaddr ,int addrlen).**
- **descripteur** : REPRÉSENTE LE DESCRIPTEUR DU SOCKET NOUVELLEMENT CRÉÉ
- **Localaddr**: EST UNE STRUCTURE QUI SPÉCifie L'ADRESSE LOCALE À TRAVERS LAQUELLE LE PROGRAMME DOIT COMMUNIQUER
- **Addrlen** : INDIQUE LA TAILLE DU CHAMP *localaddr*. ON UTILISE GÉNÉRALEMENT *sizeof(localaddr)*.

## EXEMPLE D'ADRESSAGE

```
struct SOCKADDR_IN serveur;
```

```
serveur.sin_family = AF_INET;
```

```
serveur.sin_port = htons(50000);
```

```
serveur.sin_addr.s_addr = inet_addr("192.168.1.10");
```

Inet-addr() : permet de convertir de la notation numérique pointé IPV4 en codage réseau

# LA FONCTION BIND()

- IL EXISTE 4 POSSIBILITÉS D'UTILISER LE BIND() :

1) EN SPÉCIFIANT L'ADRESSE IP ET LE NUMÉRO DE PORT,

```
struct SOCKADDR_IN serveur;  
  
serveur.sin_family = AF_INET;  
serveur.sin_port = htons(50000);  
  
serveur.sin_addr.s_addr = inet_addr("192.168.1.10");
```

# LA FONCTION BIND()

- IL EXISTE 4 POSSIBILITÉS D'UTILISER LE BIND() :
- 1) EN SPÉCIFIANT L'ADRESSE IP ET LE NUMÉRO DE PORT,
  - 2) EN SPÉCIFIANT L'ADRESSE IP ET EN LAISSANT LE SYSTÈME CHOISIR UN NUMÉRO DE PORT LIBRE INUTILISÉ

```
struct SOCKADDR_IN client;  
  
client.sin_family = AF_INET;  
client.sin_port = 0; // port alloué dynamiquement  
  
client.sin_addr.s_addr = inet_addr("192.168.1.10");
```

# LA FONCTION BIND()

- IL EXISTE 4 POSSIBILITÉS D'UTILISER LE BIND() :

1) EN SPÉCIFIANT L'ADRESSE IP ET LE NUMÉRO DE PORT,

2) EN SPÉCIFIANT L'ADRESSE IP ET EN LAISSANT LE SYSTÈME CHOISIR UN NUMÉRO DE PORT LIBRE INUTILISÉ

3) EN UTILISANT L'ADRESSE IP : *INADDR\_ANY* QUI SIGNIFIE QUE LA SOCKET PEUT-ÊTRE ASSOCIÉE À N'IMPORTE QUELLE ADRESSE IP DE LA MACHINE LOCALE (S'IL EN EXISTE PLUSIEURS) ET À UN NUMÉRO DE PORT SPÉCIFIQUE.

```
struct SOCKADDR_IN client;  
  
client.sin_family = AF_INET;  
client.sin_port = htons(50000);  
  
client.sin_addr.s_addr = INADDR_ANY;
```

# LA FONCTION BIND()

- IL EXISTE 4 POSSIBILITÉS D'UTILISER LE BIND() :

1) EN SPÉCIFIANT L'ADRESSE IP ET LE NUMÉRO DE PORT,

2) EN SPÉCIFIANT L'ADRESSE IP ET EN LAISSANT LE SYSTÈME CHOISIR UN NUMÉRO DE PORT LIBRE INUTILISÉ

3) EN UTILISANT L'ADRESSE IP : *INADDR\_ANY* QUI SIGNIFIE QUE LA SOCKET PEUT-ÊTRE ASSOCIÉE À N'IMPORTE QUELLE ADRESSE IP DE LA MACHINE LOCALE (S'IL EN EXISTE PLUSIEURS) ET À UN NUMÉRO DE PORT SPÉCIFIQUE.

4) EN LAISSANT LE SYSTÈME CHOISIR UNE ADRESSE IP ET UN NUMÉRO DE PORT QUELCONQUE . UTILISER DANS LES PROGRAMMES CLIENTS . « SANS UTILISATION DE LA PRIMITIF BIND »

# LA FONCTION LISTEN (TCP)

- **MISE D'UN SERVEUR À L'ÉTAT D'ÉCOUTE** : INDIQUE QUE LE SERVEUR PEUT ATTENDRE AU MAXIMUM QLEN REQUÊTES DES CLIENTS.
- LA FONCTION ***LISTEN()*** PERMET DE METTRE UNE SOCKET EN ATTENTE DE CONNEXION.
- **int listen (int socket, int qlen )**
- **socket** PRÉSENTE LA SOCKET PRÉCÉDEMMENT OUVERTE
- **qlen** PRÉSENTE LE NOMBRE MAXIMAL DE CONNEXIONS POUVANT ÊTRE MISES EN ATTENTE <5.
- **qlen = 0** , LAISSER LE SYSTÈME TROUVER UNE VALEUR APPROPRIÉE.
- LA FONCTION ***LISTEN()*** RETOURNE LA VALEUR **-1** EN CAS DE PROBLÈME, SINON ELLE RETOURNE **0**.
- **LISTEN (S , 3);**

# LA FONCTION CONNECT() (CLIENT TCP)

FONCTION **connect()** PERMET DE DEMANDER L'ÉTABLISSEMENT DE CONNEXION AVEC LE SERVEUR :

```
int connect(int socket, struct sockaddr * addr , int * addrlen)
```

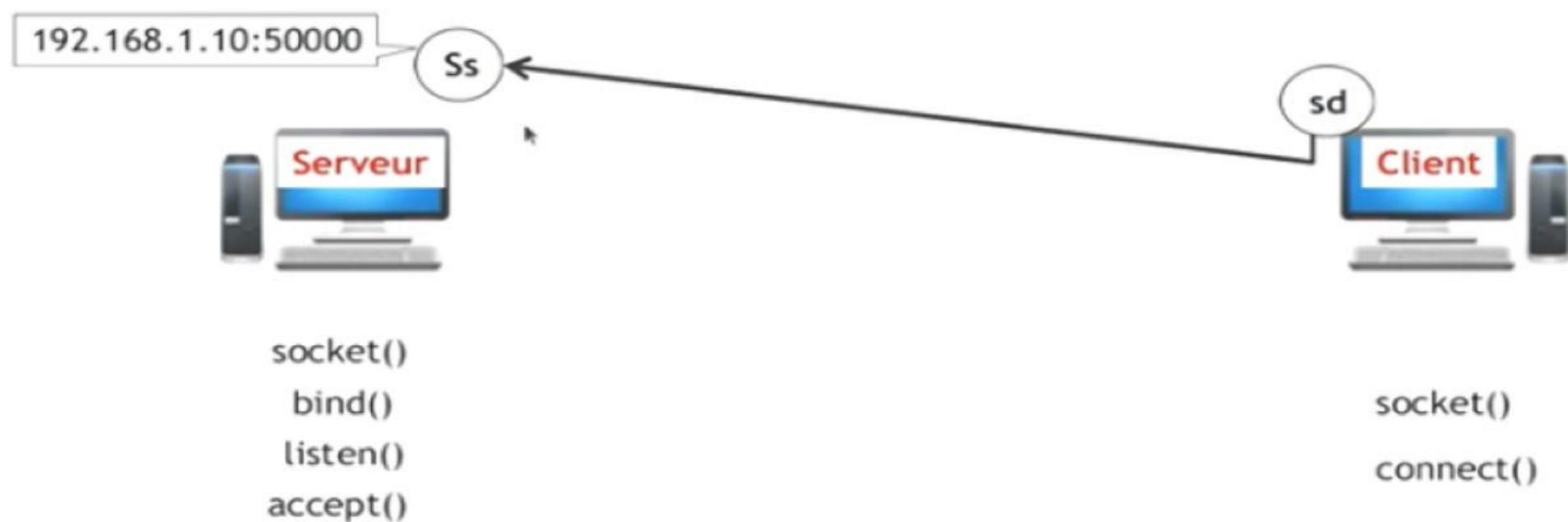
- **socket** REPRÉSENTE LA SOCKET PRÉCÉDEMMENT OUVERTE.
- **addr** REPRÉSENTE L'ADRESSE DE L'HÔTE À CONTACTER (SERVEUR). POUR ÉTABLIR UNE CONNEXION, LE CLIENT NE NÉCESSITE PAS DE FAIRE UN **BIND()**
- **addrlen** REPRÉSENTE LA TAILLE DE L'ADRESSE DE L'HÔTE À CONTACTER
- LA FONCTION **CONNECT()** RETOURNE **0** SI LA CONNEXION S'EST BIEN DÉROULÉE, SINON **-1**.

```
struct SOCKADDR_IN serveur;  
serveur.sin_family = AF_INET;  
serveur.sin_port = htons(50000);  
serveur.sin_addr.s_addr = inet_addr( "192.168.1.10 ");
```

```
Connect(sd, (struct sockaddr *) &serveur, sizeof (serveur) )
```

## LA FONCTION ACCEPT (SERVEUR TCP)

- LA PRIMITIVE ACCEPT() RETOURNE UN NOUVEAU DESCRIPTEUR DE SOCKET, QUI SERA UTILISÉ POUR L'ÉCHANGE DE DONNÉES AVEC LE CLIENT.
- LA FONCTION **ACCEPT()** PERMET LA CONNEXION EN ACCEPTANT UN APPEL :
- **int accept(int socket , struct sockaddr \* addr , int \* addrlen)**
- **SOCKET** REPRÉSENTE LA SOCKET PRÉCÉDEMMENT OUVERTE (LA SOCKET LOCALE)
- **ADDR** REPRÉSENTE UN TAMPON DESTINÉ À STOCKER L'ADRESSE **DE L'APPELANT**
- **ADDRLen** REPRÉSENTE LA TAILLE DE L'ADRESSE **DE L'APPELANT**
- LA FONCTION **ACCEPT()** RETOURNE UN IDENTIFICATEUR DU SOCKET DE RÉPONSE. SI UNE ERREUR INTERVIENT LA FONCTION **ACCEPT()** RETOURNE LA VALEUR **-1**.



192.168.1.10:50000



socket()

bind()

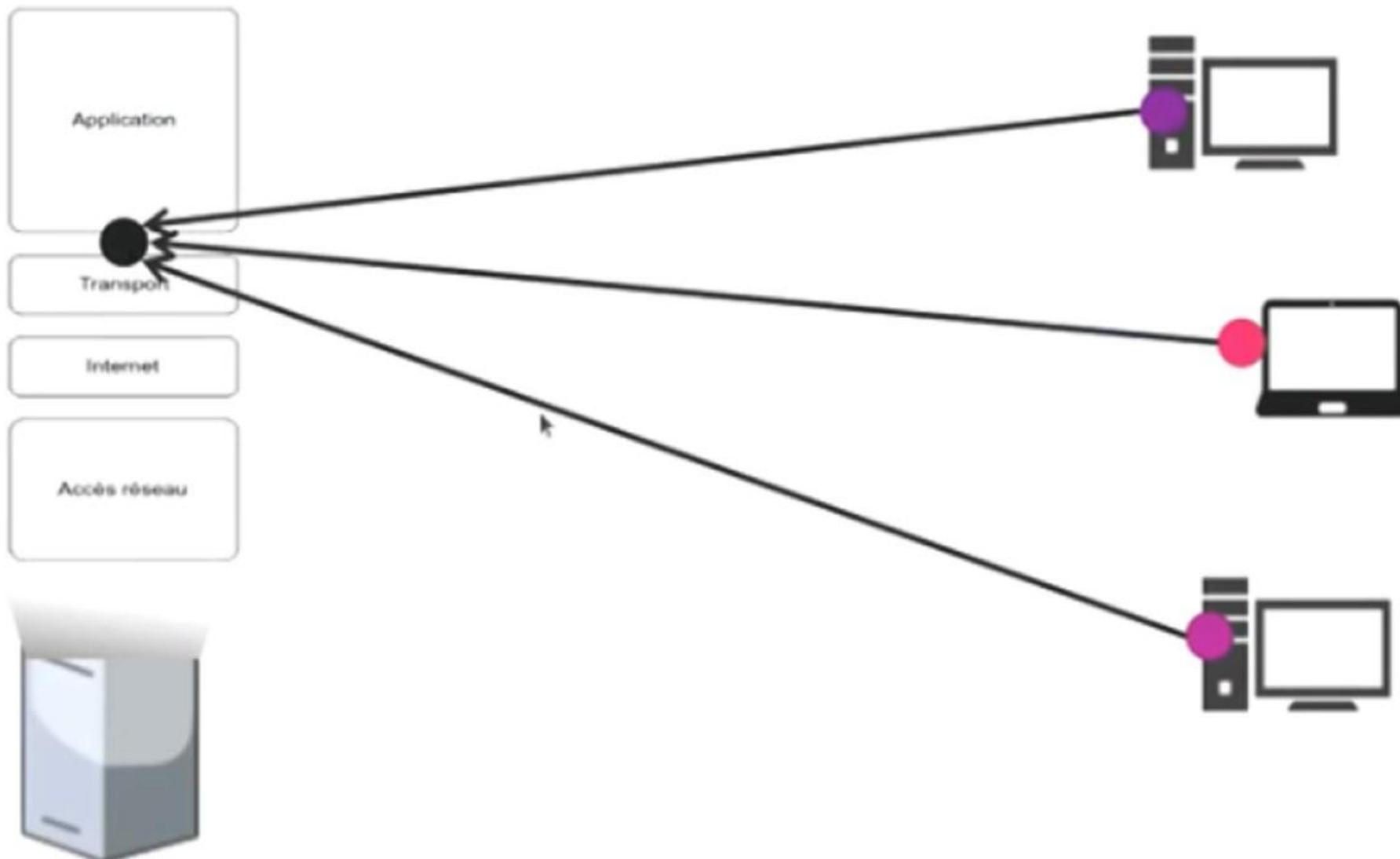
listen()

int Sock\_Dis=accept()

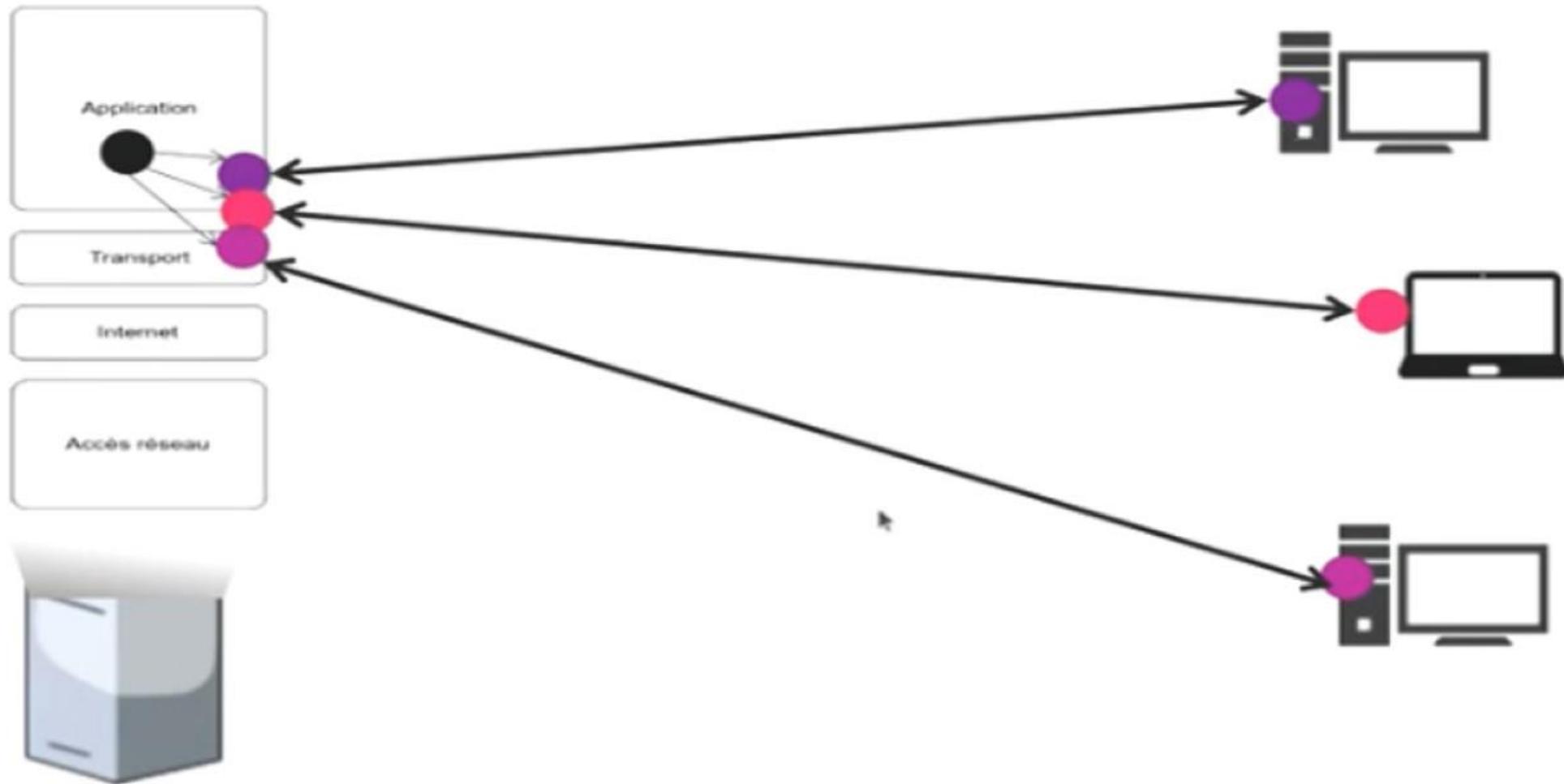
socket()

connect()

Modèle TCP/IP



**Modèle TCP/IP**



# LA FONCTION SEND() EN MODE TCP

- LA FONCTION SEND() PERMET D'ÉCRIRE DANS UN SOCKET (ENVOYER DES DONNÉES) EN MODE CONNECTÉ (TCP)
- **int send (int socket,char \* buffer,int len,int flags)**
- **SOCKET** PRÉSENTE LA SOCKET PRÉCÉDEMMENT OUVERTE
- **BUFFER** PRÉSENTE UN TAMPON CONTENANT LES OCTETS À ENVOYER AU CLIENT
- **LEN** INDIQUE LE NOMBRE D'OCTETS À ENVOYER
- **FLAGS** CORRESPOND AU TYPE D'ENVOI À ADOPTER : LE FLAG 0 INDIQUE UN ENVOI NORMAL, **MSG\_OOB** INDIQUE UN ENVOI URGENT :
- LA FONCTION **SEND()** RENVOIE LE NOMBRE D'OCTETS EFFECTIVEMENT ENVOYÉS.

```
Const char * message = "bonjour";
Send(s , message, strlen(message), 0);
```

## LA FONCTION RECV() EN MODE TCP

- LA FONCTION `RECV()` PERMET DE LIRE DANS UNE SOCKET EN MODE CONNECTÉ (TCP)
- **`int recv (int socket,char * buffer,int len,int flags)`**
- **socket** REPRÉSENTE LA SOCKET PRÉCÉDEMMENT OUVERTE
- **buffer** REPRÉSENTE UN TAMPON QUI RECEVRA LES OCTETS EN PROVENANCE DU CLIENT
- **len** INDIQUE LE NOMBRE D'OCTETS À LIRE
- **flags** CORRESPOND AU TYPE DE LECTURE À ADOPTER :LE FLAG 0 INDIQUE UN ENVOI NORMAL

# LA FONCTION SENDTO() EN MODE UDP

- LA FONCTION **SENDTO()** PERMET D'ÉCRIRE DANS UNE SOCKET (ENVOYER DES DONNÉES) EN MODE NON CONNECTÉ (UDP)
- **int sendto (int socket, char \* buffer, int len, int flags, sockaddr \* to, int tolen)**
- **socket** PRÉSENTE LA SOCKET PRÉCÉDEMMENT OUVERTE.
- **buffer** PRÉSENTE UN TAMPON CONTENANT LES OCTETS À ENVOYER AU CLIENT.
- **len** INDIQUE LE NOMBRE D'OCTETS À ENVOYER.
- **flags** CORRESPOND AU TYPE D'ENVOI À ADOPTER :LE FLAG 0 INDIQUE UN ENVOI NORMAL
- **to** CORRESPOND À L'ADRESSE D'UNE STRUCTURE QUI CONTIENDRA L'ADRESSE DU DESTINATAIRE
- **tolen** ENTIER QUI INDIQUE LA TAILLE DE LA STRUCTURE DE L'ADRESSE DU DESTINATAIRE
- LA FONCTION **sendto()** RENVOIE LE NOMBRE D'OCTETS EFFECIVEMENT ENVOYÉS.

# LA FONCTION RECVFROM() EN MODE UDP

- LA FONCTION **RECVFROM()** PERMET DE LIRE DANS UNE SOCKET EN MODE NON CONNECTÉ (UDP)
- **int recvfrom (int socket,char \* buf, int len, int flags, sockaddr \* from, int \* frlen)**
- **socket** PRÉSENTE LA SOCKET PRÉCÉDEMMENT OUVERTE
- **buf** PRÉSENTE UN TAMPON QUI RECEVRA LES OCTETS EN PROVENANCE DU CLIENT
- **len** INDIQUE LE NOMBRE D'OCTETS À LIRE
- **flags** CORRESPOND AU TYPE DE LECTURE À ADOPTER: LE FLAG 0 INDIQUE UNE LECTURE NORMALE.
- **from** CORRESPOND À L'ADRESSE D'UNE STRUCTURE QUI CONTIENDRA L'ADRESSE DE L'ÉMETTEUR
- **frlen** ADRESSE D'UN ENTIER QUI INDIQUE LA TAILLE DE LA STRUCTURE DE L'ADRESSE DE L'ÉMETTEUR
- LA FONCTION **recvfrom()** RENVOIE LE NOMBRE D'OCTETS LUS. DE PLUS CETTE FONCTION BLOQUE LE PROCESSUS JUSQU'À CE QU'ELLE REÇOIVE DES DONNÉES

# LES FONCTIONS CLOSE() ET SHUTDOWN()

- LA FONCTION **CLOSE()** PERMET LA FERMETURE D'UN SOCKET EN PERMETTANT AU SYSTÈME D'ENVOYER LES DONNÉES RESTANTES (POUR TCP) :
- **int close(int socket)**
- LA FONCTION **shutdown()** PERMET LA FERMETURE D'UN SOCKET DANS UN DES DEUX SENS (POUR UNE CONNEXION FULL-DUPLEX) :
- **int shutdown(int socket,int how)**
  - SI **how** EST ÉGAL À **0**, LA SOCKET EST FERMÉE EN RÉCEPTION
  - SI **how** EST ÉGAL À **1**, LA SOCKET EST FERMÉE EN ÉMISSION
  - SI **how** EST ÉGAL À **2**, LA SOCKET EST FERMÉE DANS LES DEUX SENS
- **close()** ET **shutdown()** RETOURNENT **-1** EN CAS D'ERREUR, **0** SI LA FERMETURE SE DÉROULE BIEN.

# PROGRAMMATION IP V6

- LE DOMAINE EST AF\_INET6
- LE CHANGEMENT MAJEUR EST LA TAILLE DES ADRESSES
- DEUX NOUVELLES PRIMITIVES `inet_ntop()` ET `inet_pton()` DE CONVERSION.
- LES FONCTIONS `socket()`, `bind()`, `connect()`, `accept()`, `sendto()`, `recvfrom()`... NE CHANGENT PAS, MAIS IL FAUT FAIRE ATTENTION AUX ADRESSES UTILISÉES.

**FIN**