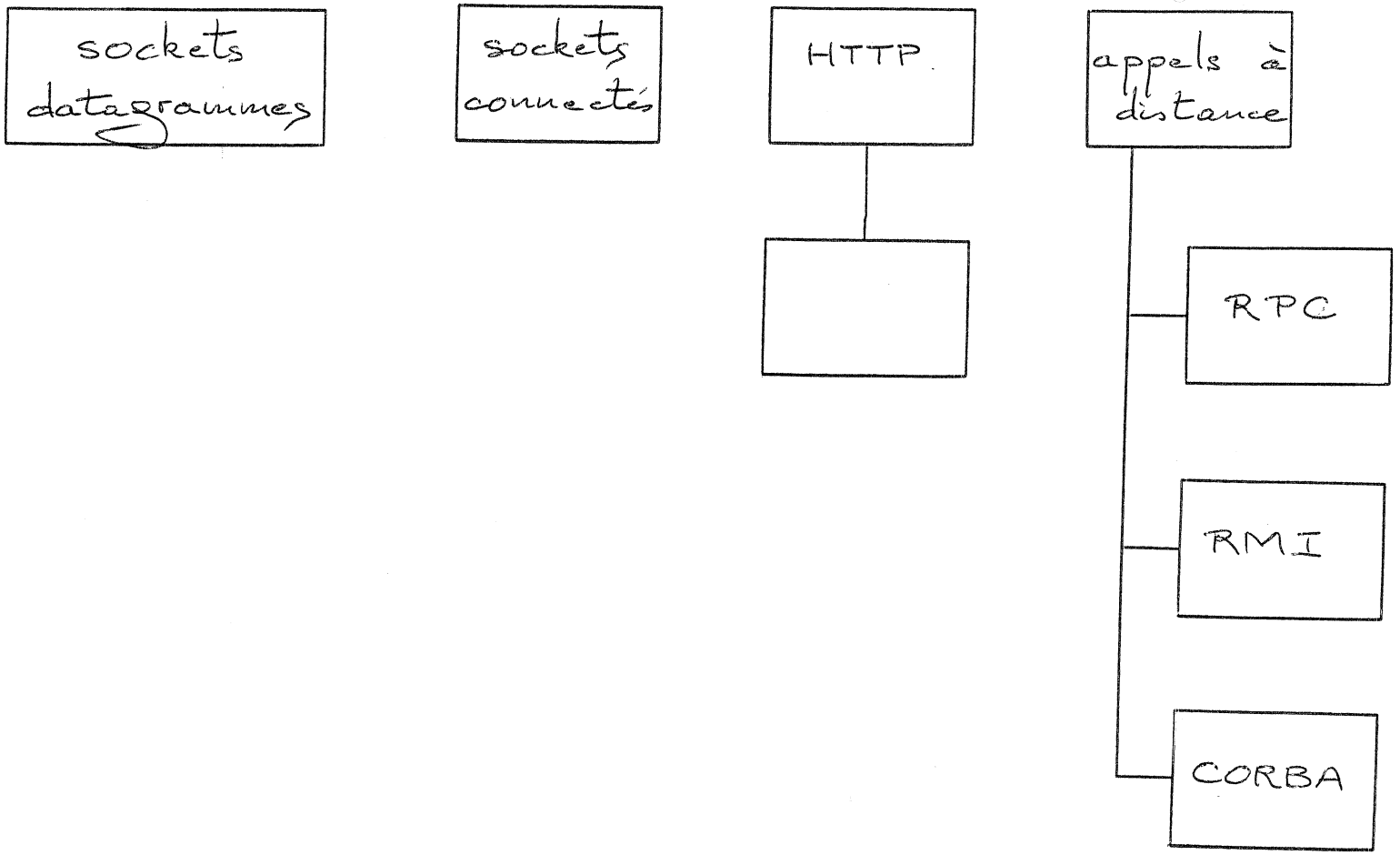
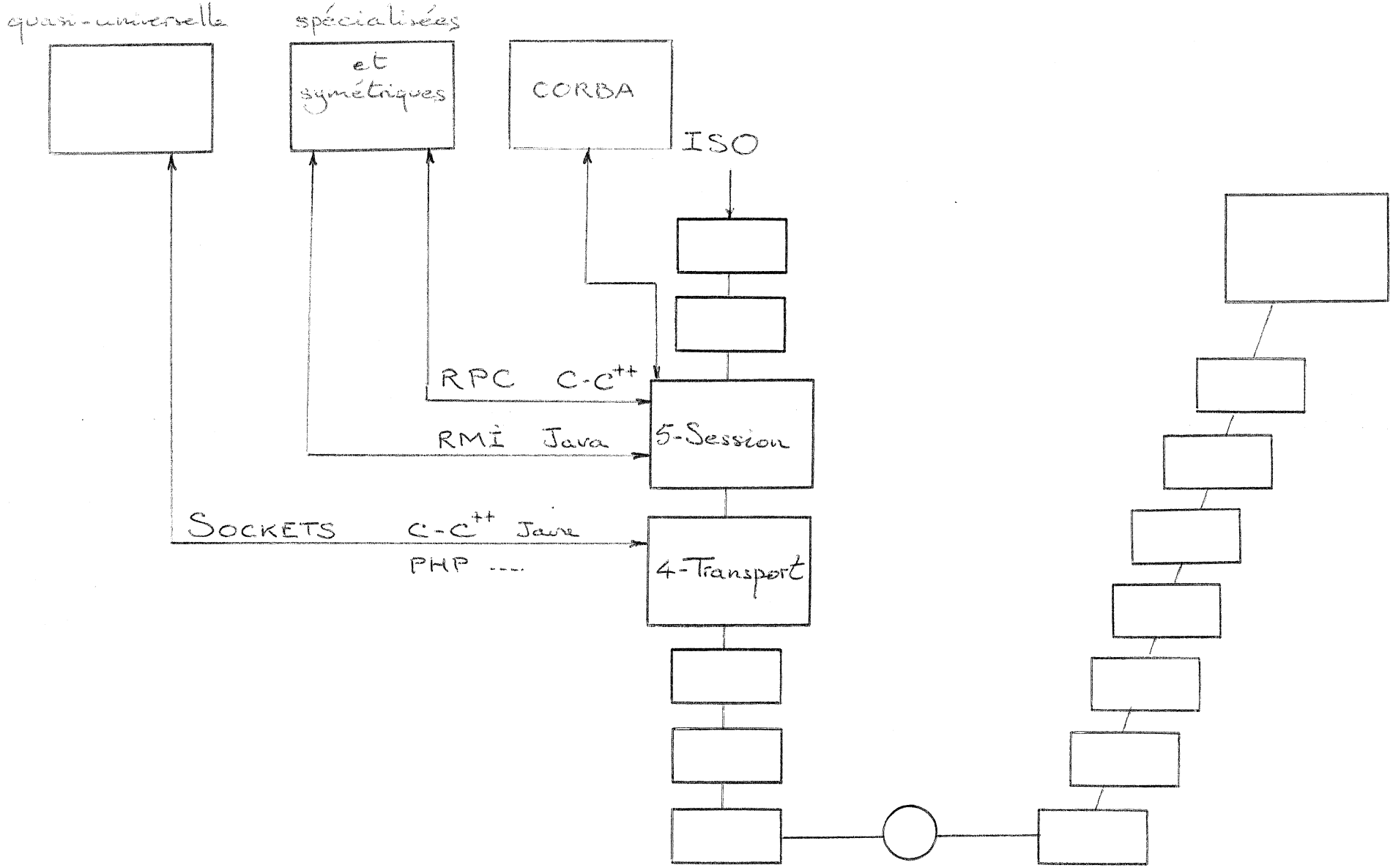
Construction de logiciels en Réseaux

Procédés Réseaux



Greffe des programmes



Protocoles de transport (couche 4)

Services de transport

Sans connexion	Avec connexion
UDP	**TCP**
Métaphore : carte postale	Métaphore : téléphone
∃ pertes	∄
∃ doublons	∄
∃ désordre	∄
∄ erreurs de caractères	∄

*) **TCP** possède un cas particulier: **HTTP**

Conversation = 1 Aller + 1 Retour

*) Dans chaque CPU les programmes décident d'utiliser
un transport (tcp ou udp) et "occupent"
un numéro de port/service ($< 2^{16}$) numéro sur 16 bits

Un port/service ne peut être occupé que par 0..1 pgm

En cours de communication

Le serveur (de connexion)

attend à une adresse convenue

Le client prend l'initiative
et interpelle le serveur
à l'adresse convenue

⇒ Le client et le serveur

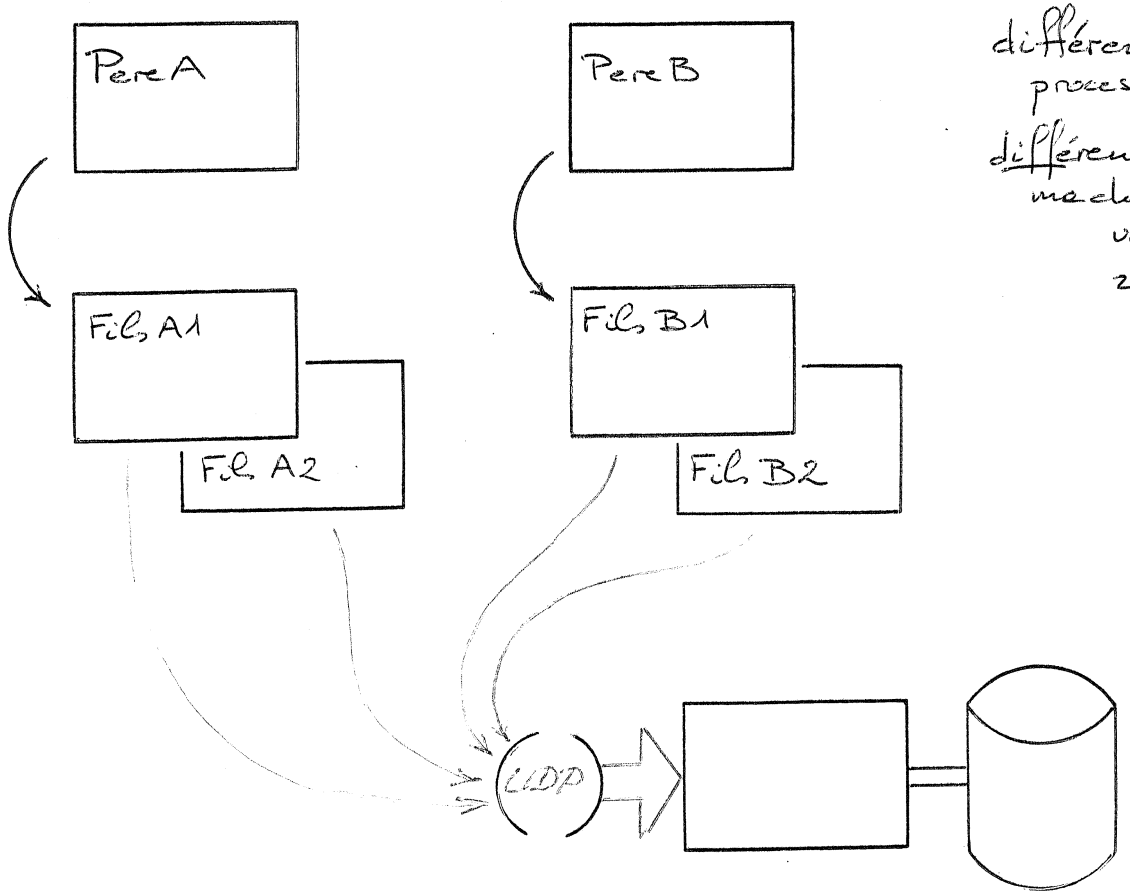
doivent connaître l'adresse du serveur:

*) le serveur pour s'y mettre,

*) le client pour y aller

	T.C.P.		U.D.P.	
	SERVEUR	CLIENT	SERVEUR	CLIENT
Initialisation	socket bind listen	socket (bind)	socket bind	socket (bind)
Fonctionnement	accept send reco close	connect reco send close	 recofrom sendto close	 sendto recofrom close
Variante	fdopen --- fclose	fdopen --- fclose		

Une coopération tcp-udp : la journalisation



différents processus,
différentes machines virtuelles, réelles,

Que paramétrer ?

*) Réseaux

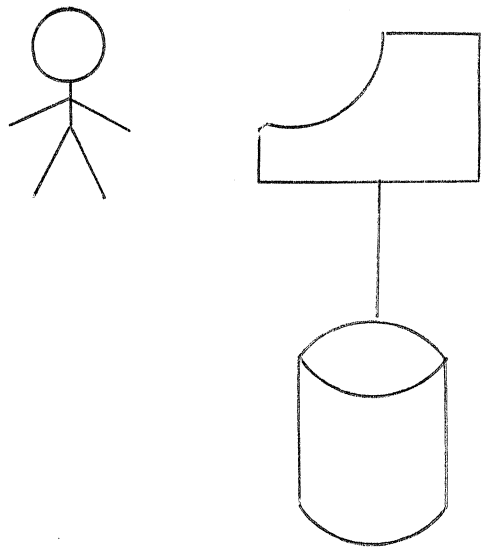
*) Protocoles

*) Services / Ports

*) Machines

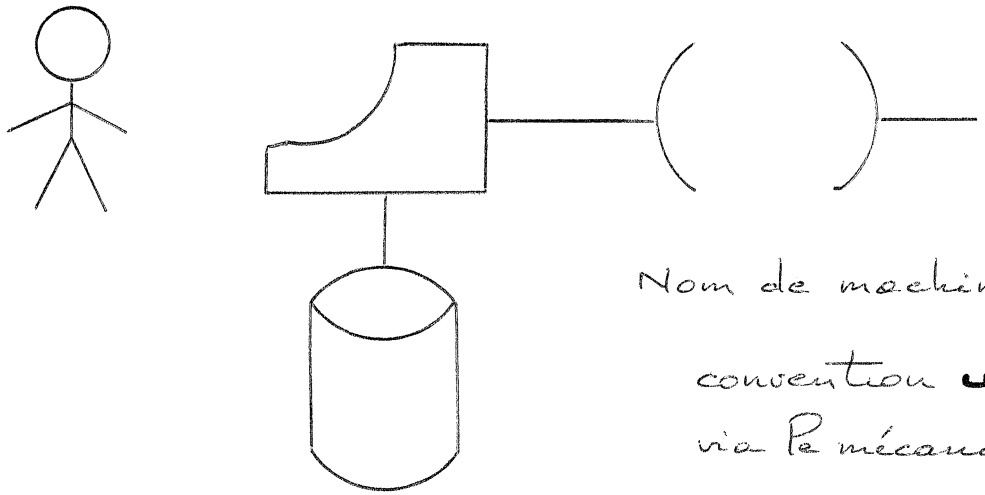
~~> fichiers ascii ...

Conventions de nommage



Nom de service :

convention **Pocale**

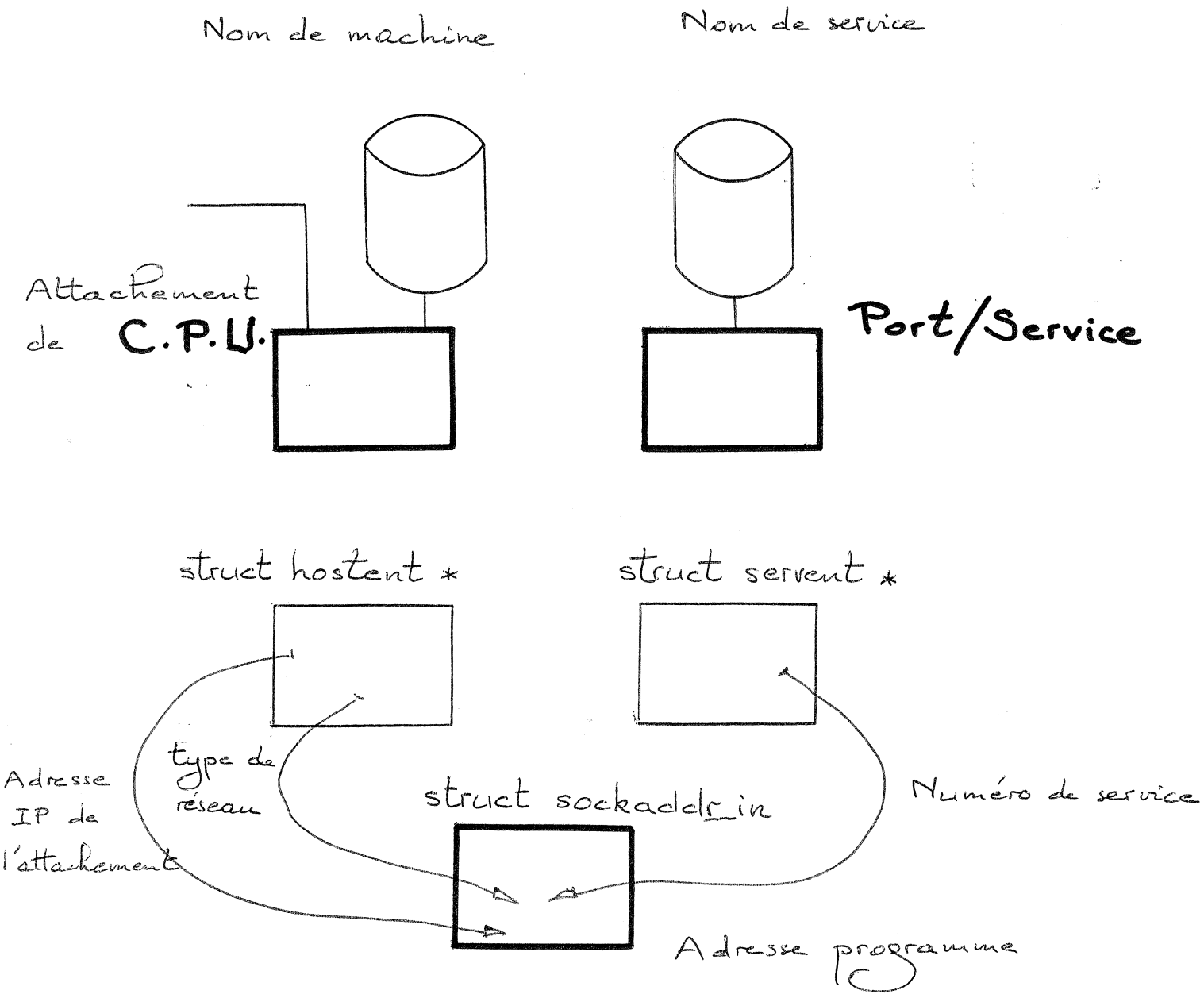


Nom de machine :

convention **universelle**

via le mécanisme du D.N.S.

Paramétrage (FACULTATIF !)



Fichiers de base

open
pipe
dup
socket (SOCK_STREAM)
accept

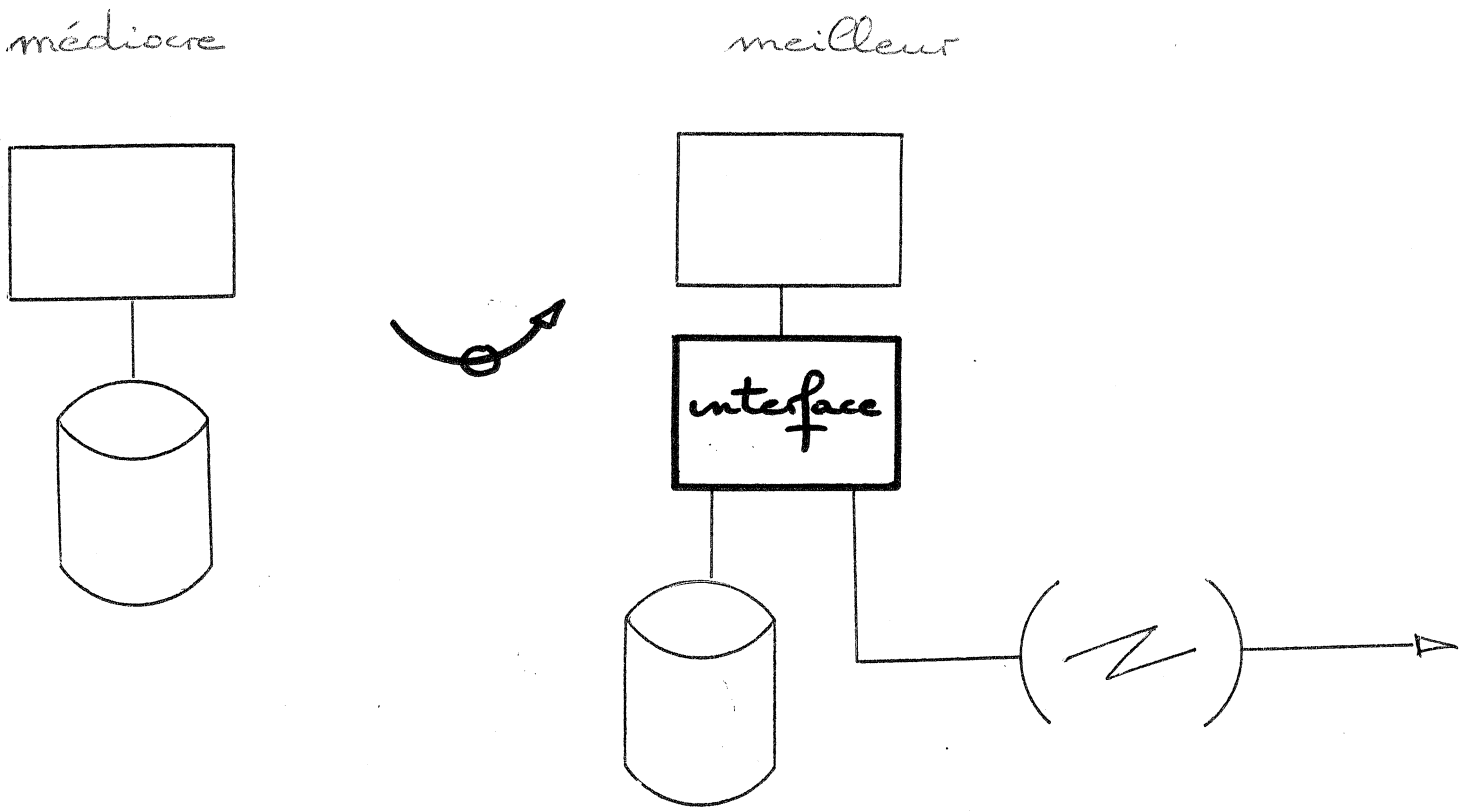
Files d'attente

msgget (et al)

socket (SOCK_DGRAM)

Convention d'indication d'erreur

Une fonction qui renvoit	signale une erreur en renvoyant
un pointeur	NULL
un comptage (int ≥ 0)	-1
rien de particulier	-1


médiocre
meilleur
interface

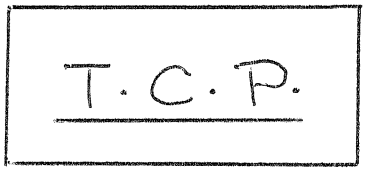
Interfaces d'accès

*) Séquentiel

setxxxent()
getxxxent()

*) Accès direct

getxxxbyname()


T.C.P.

Une bizarrerie de la fonction "socket"

```
nº fichier de base = socket
                     (
                       AF_INET,            /* = internet */
                       SOCK_STREAM,        /* = protocole TCP */
                       SOCK_DATAGRAM,      /* = protocole UDP */
                     0 );                  /* = sous-protocole,
                                              prévu et jamais utilisé! */
```

l'un ou l'autre

Une bizarrerie de la fonction "listen"

Son nom, particulièrement trompeur.

Ici il s'agit seulement (et obligatoirement) de prévoir combien d'appels simultanés, et pas combien de connexions !

Une seule différence entre "bind" et "accept"

bind : on connaît tout !

code retour = bind (n° fichier,
emplacement mémoire de l'adresse réseau,
longueur de cette adresse)

accept : on apprend (presque) tout !

nouveau fichier = accept (n° fichier,
emplacement mémoire de l'adresse réseau,
& longueur maximale (avant) et réelle (après) de cette adresse)

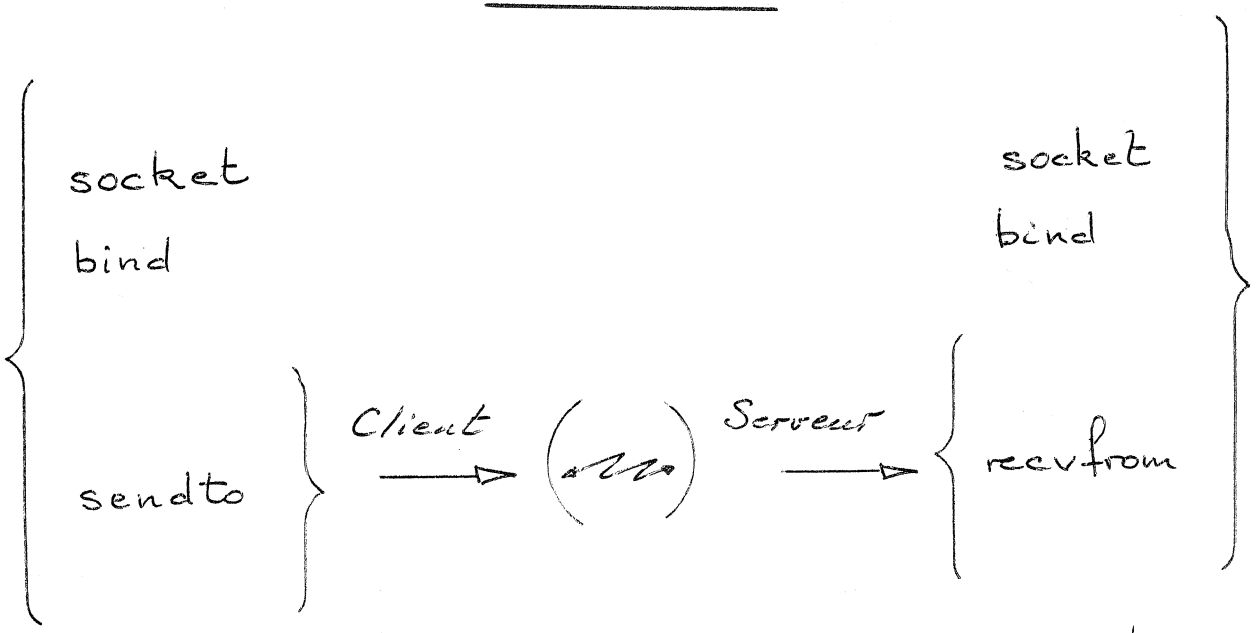
```
read (noFic, tabChar, lmax)

recv (noFic, tabChar, lmax, 0)

recvfrom(noFic, tabChar, lmax, 0, tabAdr, &laddmax)
```

U.D.P.

U-D-P-



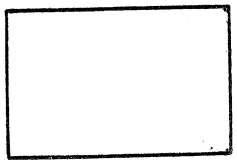
int sendto (int , char * , int , int , struct sockaddr * , int)

Nombre d'octets manipulés	Socket	Message proprement dit emplacement, lgr	priorité (mettre à zéro)	Adresse du correspondant emplacement, lgr emplacement, adr lgr

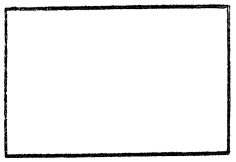
int recvfrom (int, char *, int , int , struct sockaddr_in * , int *)

int : réelle
int : max
struct sockaddr_in * : max (avant) réelle (après)

/* pseudo-code */



```
DatagramSocket ds = new
                    DatagramSocket(1234);
```



```
DatagramPacket in = new
                    DatagramPacket(char[] _, int _);

                out = new
                    DatagramPacket(char[] _, int _,
                                   InetAddr _, int _);
```

```
ds.receive(in)

ds.send(out)
```

```
in.getAddress()
in.getData()
in.getPort()
```

U.D.P. prestataire

Machine
ou
Réseau Local