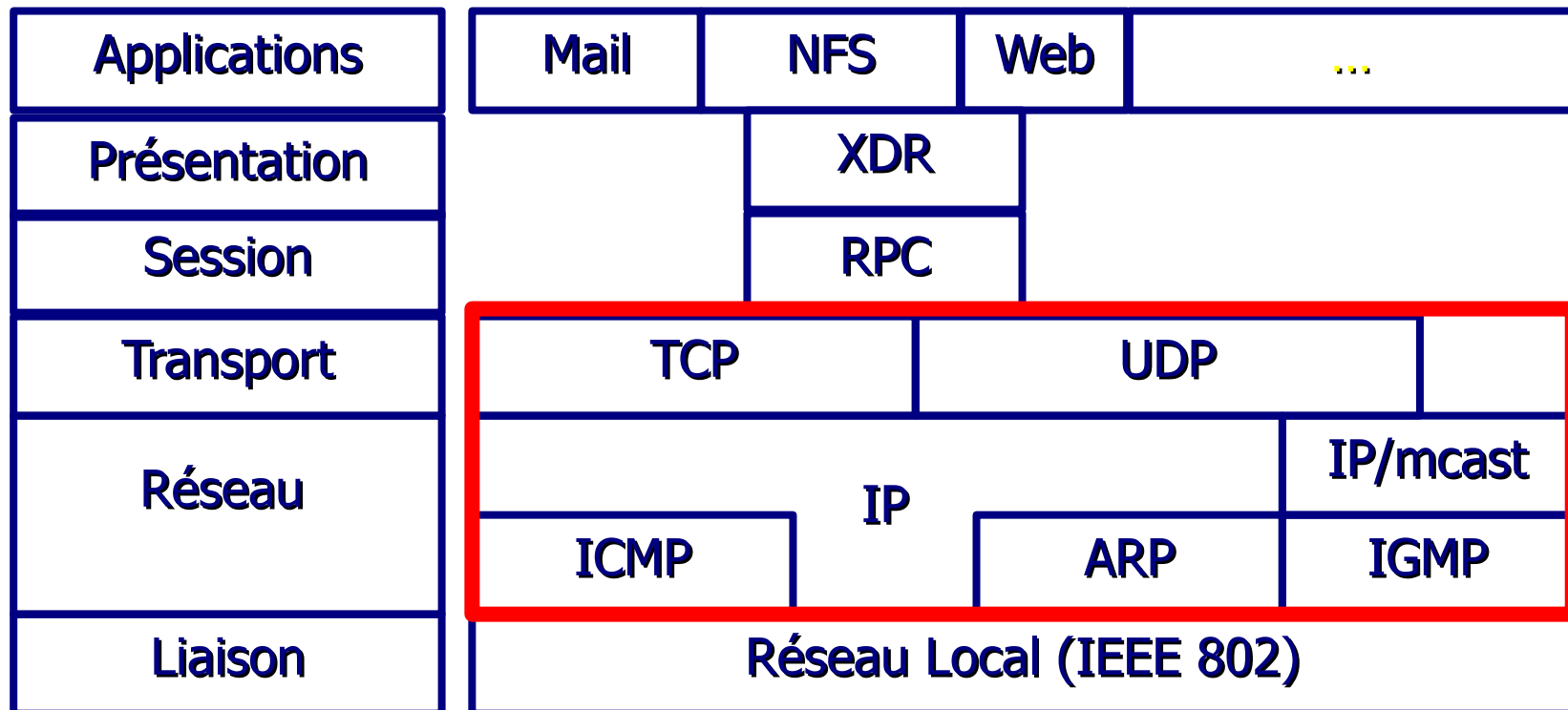
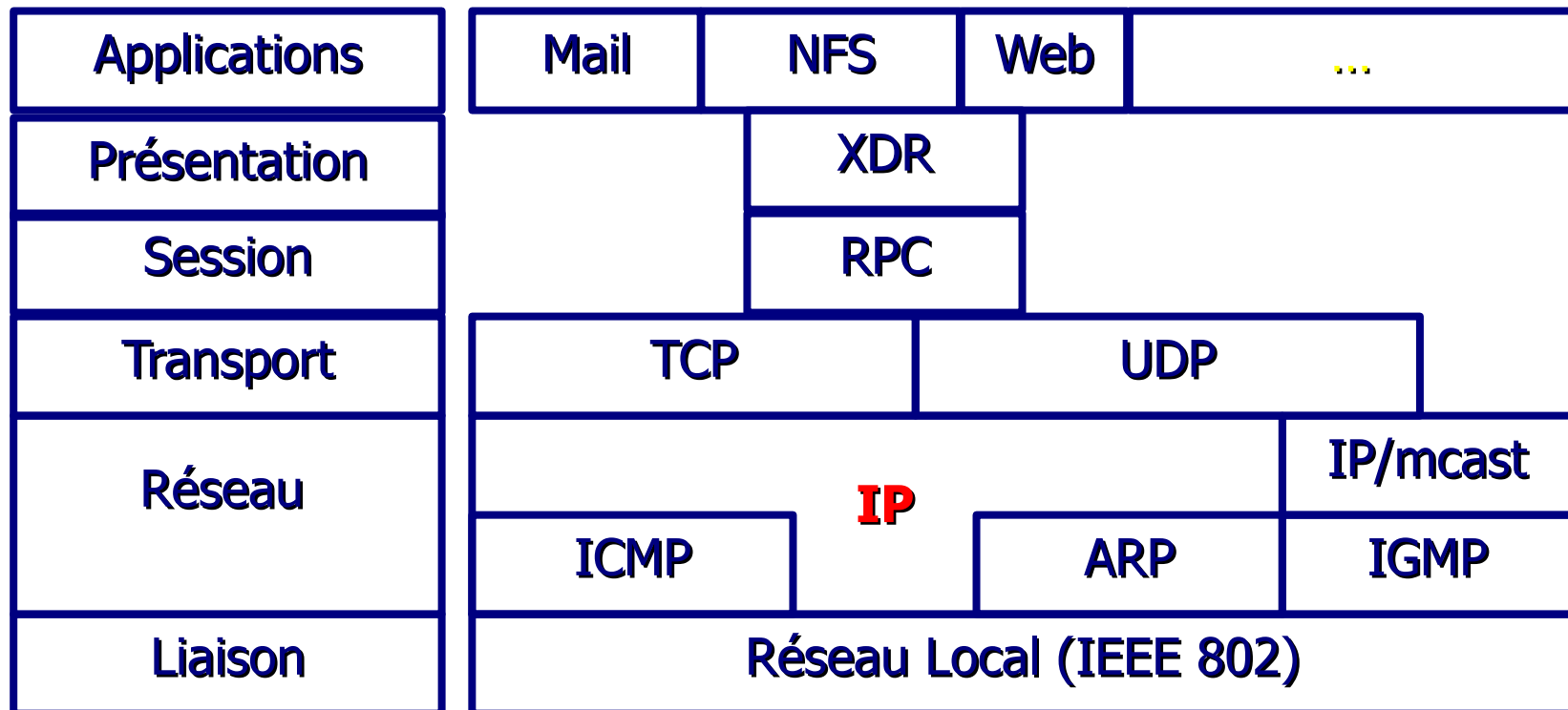


## Introduction à la Programmation Sockets

### I. Les Protocoles TCP/IP

# Architecture TCP/IP



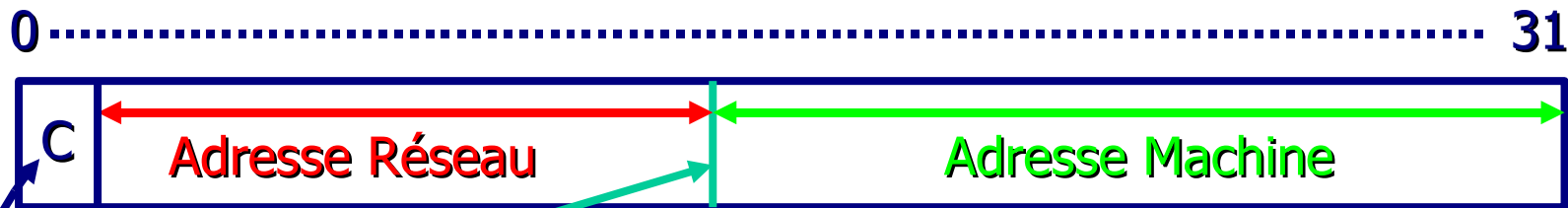


# Les Services de la couche IP

- ▶ Adressage unique dans le réseau IP (de réseaux)
  - ▶ Les réseaux utilisent des technologies différentes, qui ont leur propre espace d'adressage
- ▶ Transmission de "datagrammes" de bout en bout
  - ▶ Modèle du courrier postal :
    - ▶ datagramme = enveloppe : destinataire + expéditeur
- ▶ Segmentation et réassemblage
  - ▶ Modèle courrier postal : enveloppe limitée à 30g
    - ▶ Obligation d'envoyer les gros documents en plusieurs fois
- ▶ C'est tout !
  - ▶ "Raw IP" = Pas de fiabilité (contenu, acheminement), de contrôle de flux, ...

# L'adressage IP (v4)

- ▶ Chaque machine reçoit (au moins) une adresse : un entier sur 32 bits
- ▶ Les 32 bits sont divisés en 3 parties :

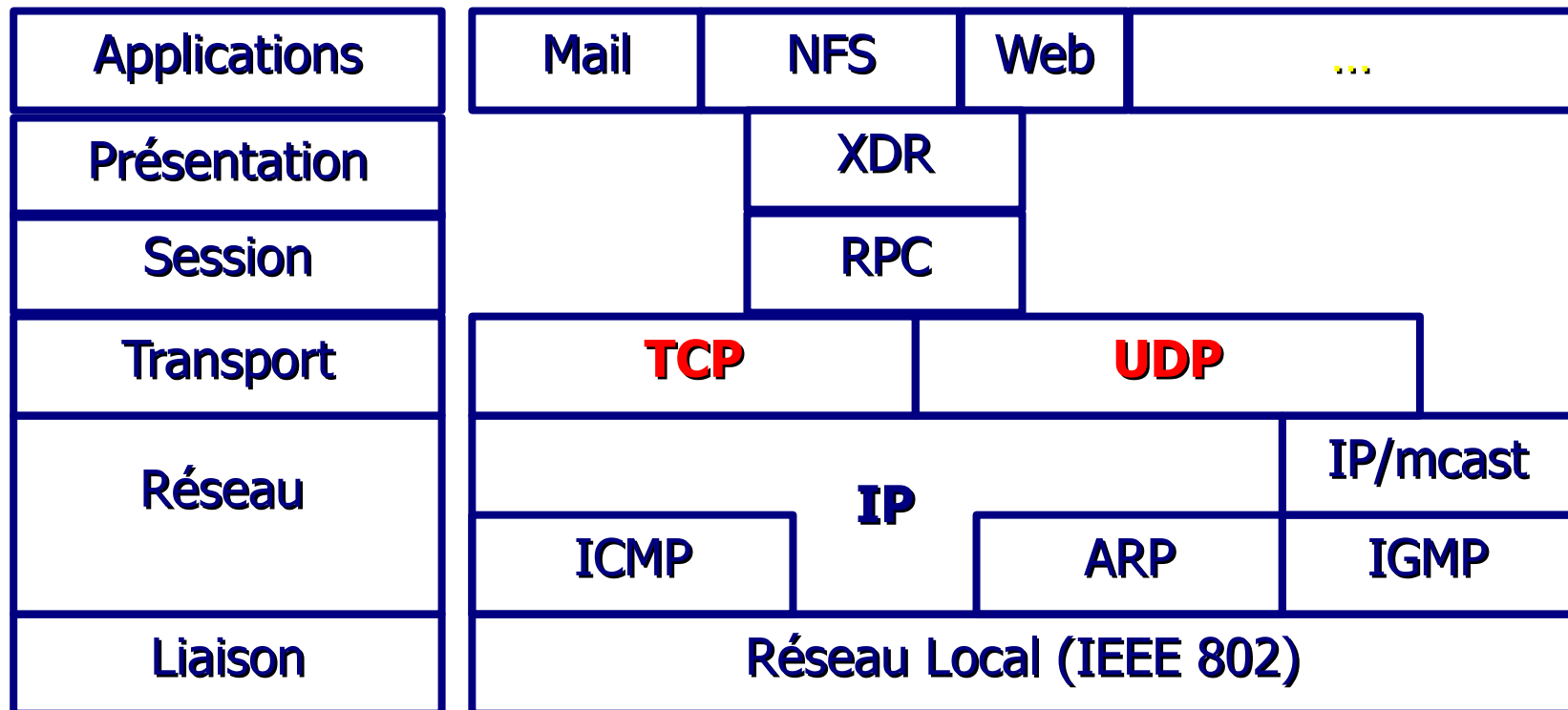


- ▶ Le nombre de bits consacrés à l'une ou l'autre partie dépend de la classe du réseau

# Les Classes de Réseaux TCP/IP

- ▶ Classe A (C= 0) : 7 bits pour le réseau, 24 bits pour l'adresse dans le réseau
- ▶ Classe B (C= 10) : 14 bits pour le réseau, 16 bits pour l'adresse dans le réseau
- ▶ Classe C (C= 110) : 21 bits pour le réseau, 8 bits pour l'adresse dans le réseau
- ▶ Classe D (C= 1110 : IP/multicast) : 4 bits pour la classe, 28 bits pour le réseau (pas d'adresse de machine dans le réseau !)
- ▶ Classe E (C= 11110) : Adresses réservées

# La couche Transport



# Les Services de la Couche Transport

- ▶ Ajoute la notion de port à une adresse IP (Multiplexage)
  - ▶ Port= entier 16 bits
    - ▶ Analogie modèle postal : « à l'attention de Mr X. »
  - ▶ 2 principaux protocoles : TCP et UDP
- ▶ Problème : peu d'informations disponibles sur les ports actifs...
  - ▶ Certains ports sont « bien connus » (telnet, ssh, ftp,...)
  - ▶ Pour les autres : besoin de solution "externe" (à TCP/IP)
    - ▶ Exemple de solution : RPC portmapper
      - ▶ Service présent sur le port « bien connu » 111
      - ▶ Répertoire des (autres) services présents



# Le Protocole UDP

- ▶ UDP = Datagramme
  - ▶ Envoi d'un message = 1 à 64Ko
    - ▶ Une adresse IP de destination
    - ▶ Un numéro de port de destination
  - ▶ Mode non connecté :
    - ⇒ Aucune garantie que le datagramme arrive !
    - ⇒ Mais si le datagramme arrive, il n'est pas altéré (contrôle CRC actif par défaut)

# Le Protocole TCP

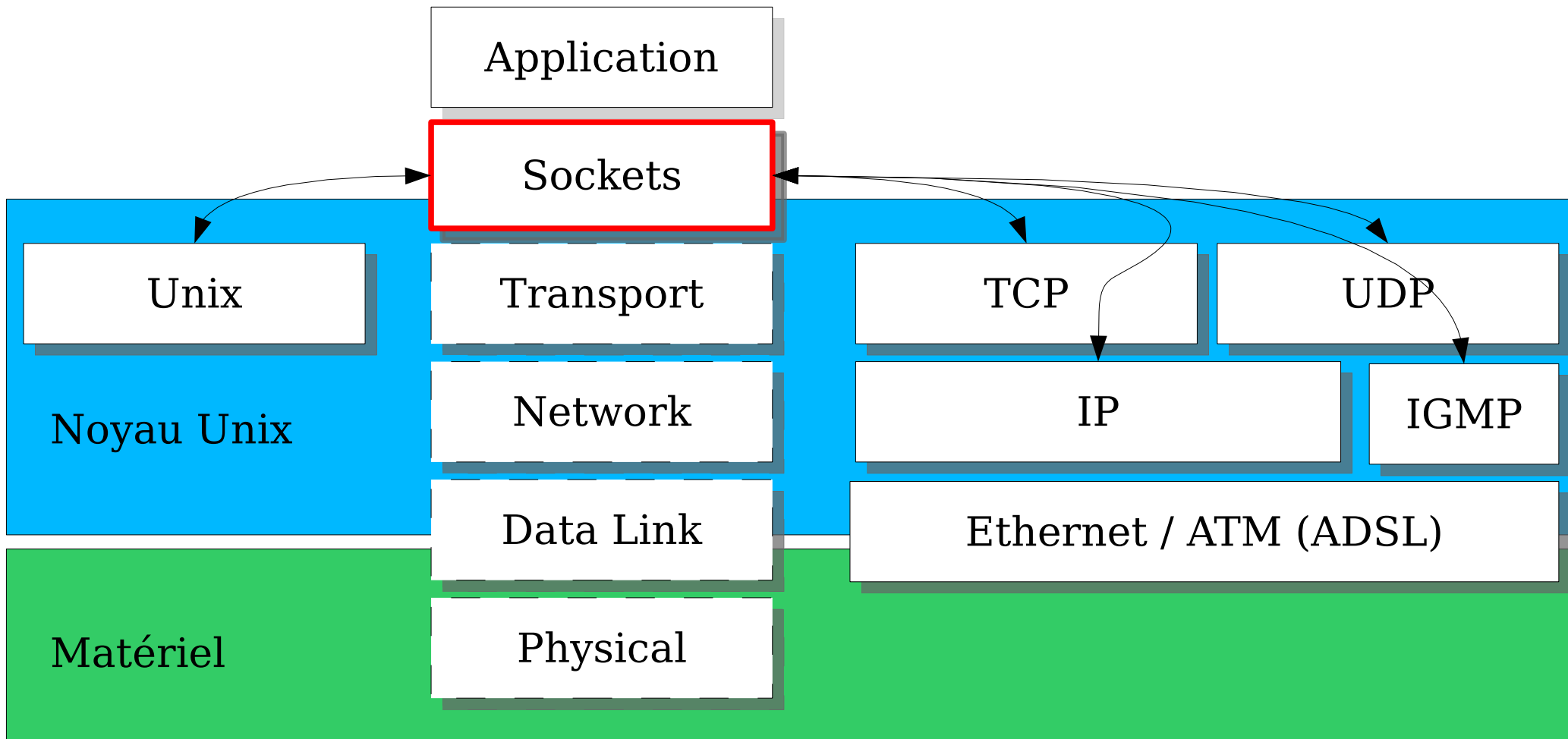
- ▶ TCP = flot de caractères
  - ▶ Mode connecté (donc phase initiale de connexion ... )
  - ▶ Flot de caractères (frontières de message non préservée ... )
  - ▶ Garanties :
    - ▶ Séquence préservée
    - ▶ Aucune perte
    - ▶ Aucune altération
  - ▶ Protocole « socialement correct » (évite la congestion)

# Socket = Prise

- ▶ Un moyen de « brancher » des processus entre-eux
  - ▶ Communication dite « par échange de messages », (*Message Passing*)
    - ▶ Message = 1 caractère (stream) ou plusieurs (datagramm)
    - ▶ S'oppose à d'autres formes tq. Mémoire (Virt.) Partagée
  - ▶ Connexion de processus locaux (IPC)
    - ▶ Protocol interne « Unix »
      - ▶ Ex: DISPLAY=:0.0
  - ▶ Connexion de processus distants
    - ▶ Protocoles de la famille TCP/IP
      - ▶ Telnet, rlogin, ssh, ftp, http, nslookup, ...
    - ▶ Autres protocoles possibles
      - ▶ Mais rarement utilisés car « TCP/IP » standard de fait

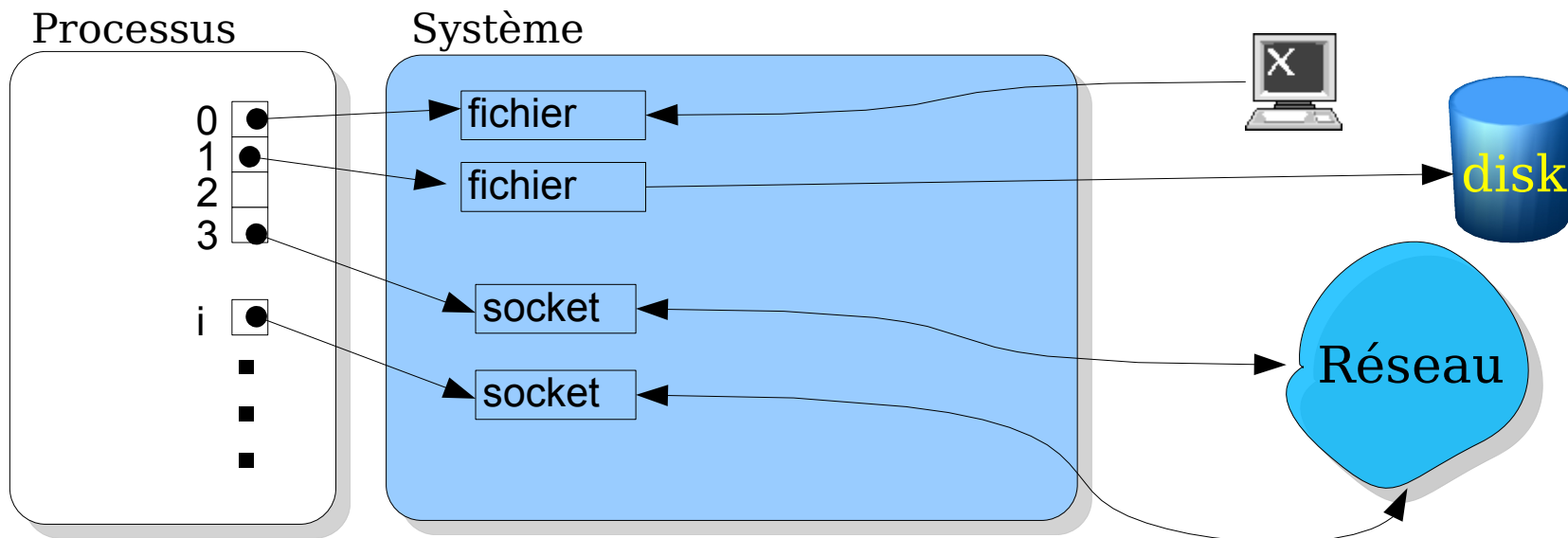
# Socket = API (issue de BSD)

- ▶ Interface de programmation
- ▶ Interface avec le système

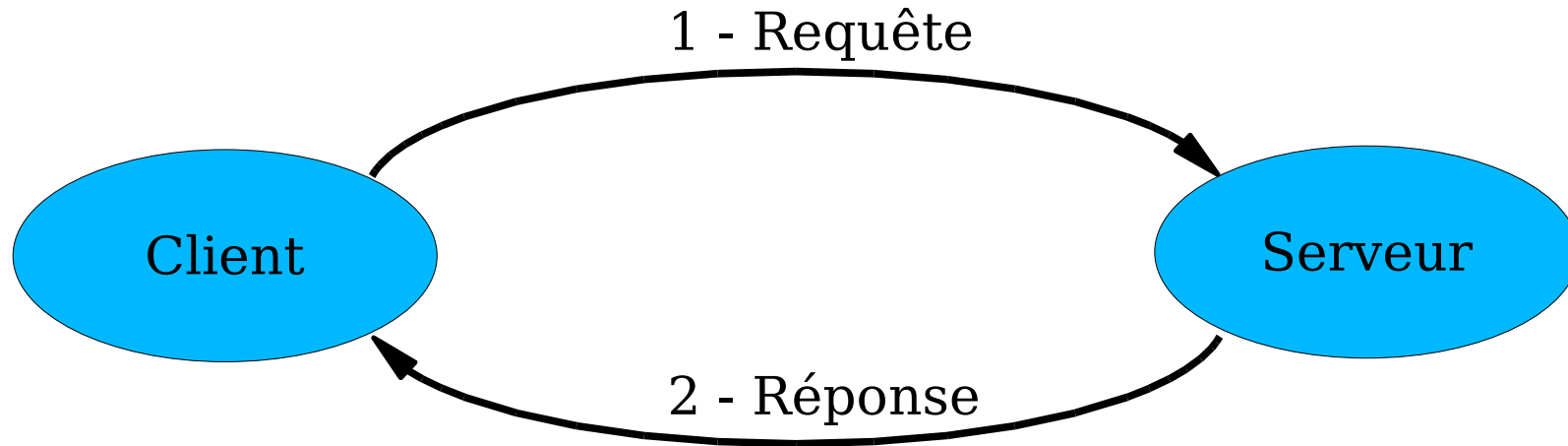


# Principe de Base de l'API Socket

- ▶ Unix = Fichiers + Processus
  - ▶ Communiquer au travers de pseudo-fichiers
    - ▶ Chaque socket est associé à un descripteur
  - ▶ Fonctionnement similaire aux FIFO (bi-directionnelles)
    - ▶ Envoyer = écrire dans le descripteur
    - ▶ Recevoir = lire dans le descripteur



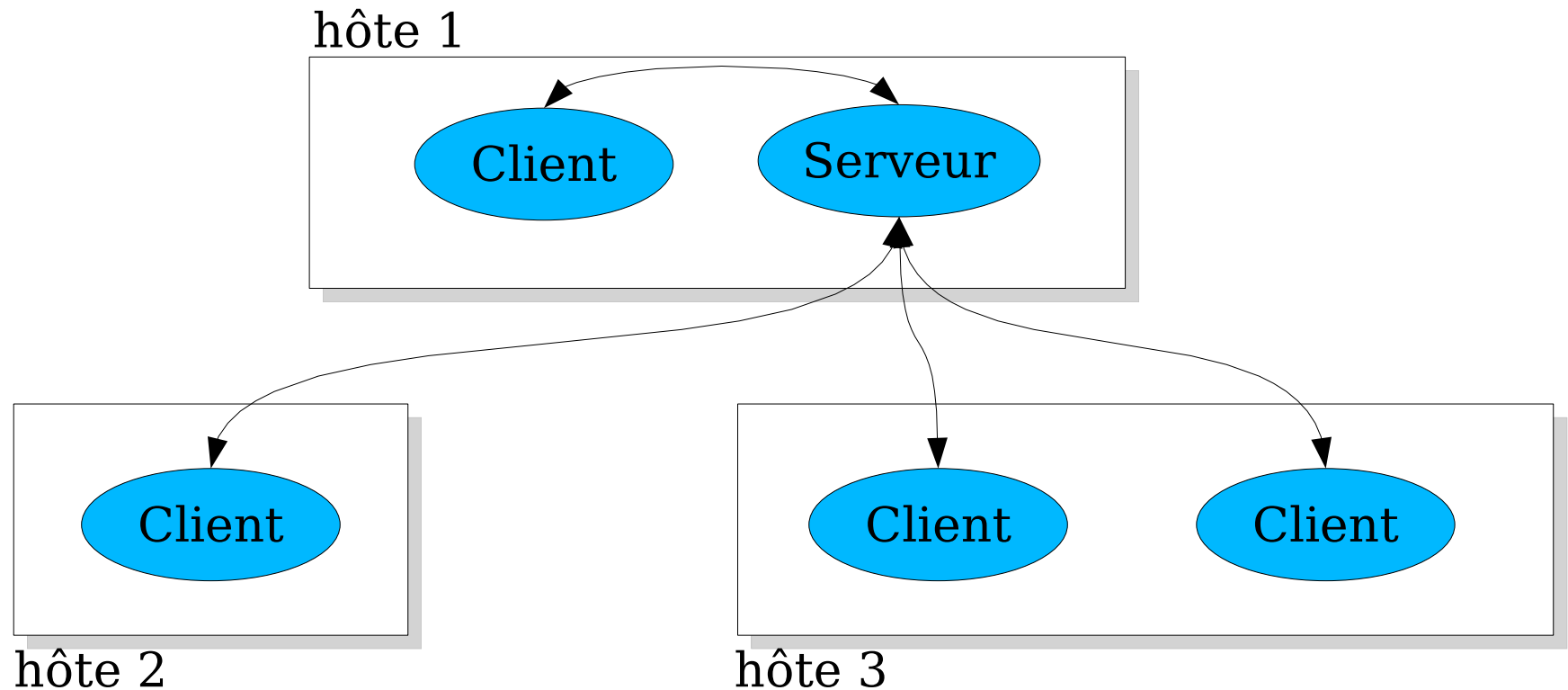
# Modèle Client/Serveur



- ▶ Programmation socket = modèle client serveur
  - ▶ Une application socket = **2 programmes** (au moins)
    - ▶ Un client
      - ▶ Etablit la connexion
      - ▶ Envoie une requête (demande service)
    - ▶ Un serveur
      - ▶ Attend connexion(s)
      - ▶ Traite requête(s)

# Client/Serveur : un modèle Asymétrique

- ▶ **Serveur** : normalement capable de (prévu pour) traiter plusieurs requêtes
  - ▶ Eventuellement en parallèle !
- ▶ **Client** : connexion (ponctuelle) vers un serveur



# Modes de Fonctionnement

- ▶ Deux modes de connexion
  - ▶ **Mode connecté** : l'interlocuteur (au bout de la prise) ne change pas
    - ▶ Idée du « téléphone rouge »
  - ▶ **Mode non connecté** : réutilisation d'un *socket* pour joindre (successivement) différents interlocuteurs
- ▶ Deux types de **flots de données**
  - ▶ Flot de **caractères** (mode « *stream* »)
    - ▶ Une lecture peut récupérer les données envoyées par plusieurs écritures (et inversement)
  - ▶ Flot de **messages** (mode « *datagramm* »)
    - ▶ Une écriture = Un message = Une lecture



## Quel mode de connexion choisir ?

- ▶ Mode connecté : logique avec protocole connecté
  - ▶ En particulier TCP
- ▶ Mode non connecté : logique avec protocole non connecté
  - ▶ En particulier UDP
- ▶ **MAIS** : Mode connecté possible avec UDP !
  - ▶ Connexion virtuelle
  - ▶ Evite (à l'expéditeur) d'avoir à répéter le destinataire à chaque envoi...
    - ▶ Allège la programmation dans certains cas

## Utilisation du mode « Flot de Messages »

- ▶ Frontière de message conservée = **information implicite**
  - ▶ Exemple :
    - ▶ expéditeur : envoi (écrit) 10 octets
    - ▶ destinataire : lit 20 octets
    - ▶ Résultat ?
      - ▶ Destinataire **ne récupère que 10 octets !!**
  - ▶ Utile lorsque les transmissions sont de longueur variable
    - ▶ Pb. destinataire : combien de données lire la prochaine fois ??
    - ▶ Solution : (tenter de) lire le maximum
      - ▶ Aucun risque de déborder sur la réception suivante
  - ▶ Attention : possibilité de perte « silencieuse » de données
    - ▶ Lorsque le destinataire ne lit pas assez de données...

## Utilisation du mode « Flot de Caractères »

- ▶ Lectures et écritures sont décorrélées
  - ▶ Le destinataire peut lire en une fois le produit de plusieurs écritures
  - ▶ Transmissions de longueur variables ?!
    - ▶ A la charge du programmeur (concepteur du protocole)
    - ▶ Ex : en-tête = longueur(contenu) + contenu
    - ▶ Pas forcément plus compliqué...

Ex : exp souhaite envoyer chaîne « truc »

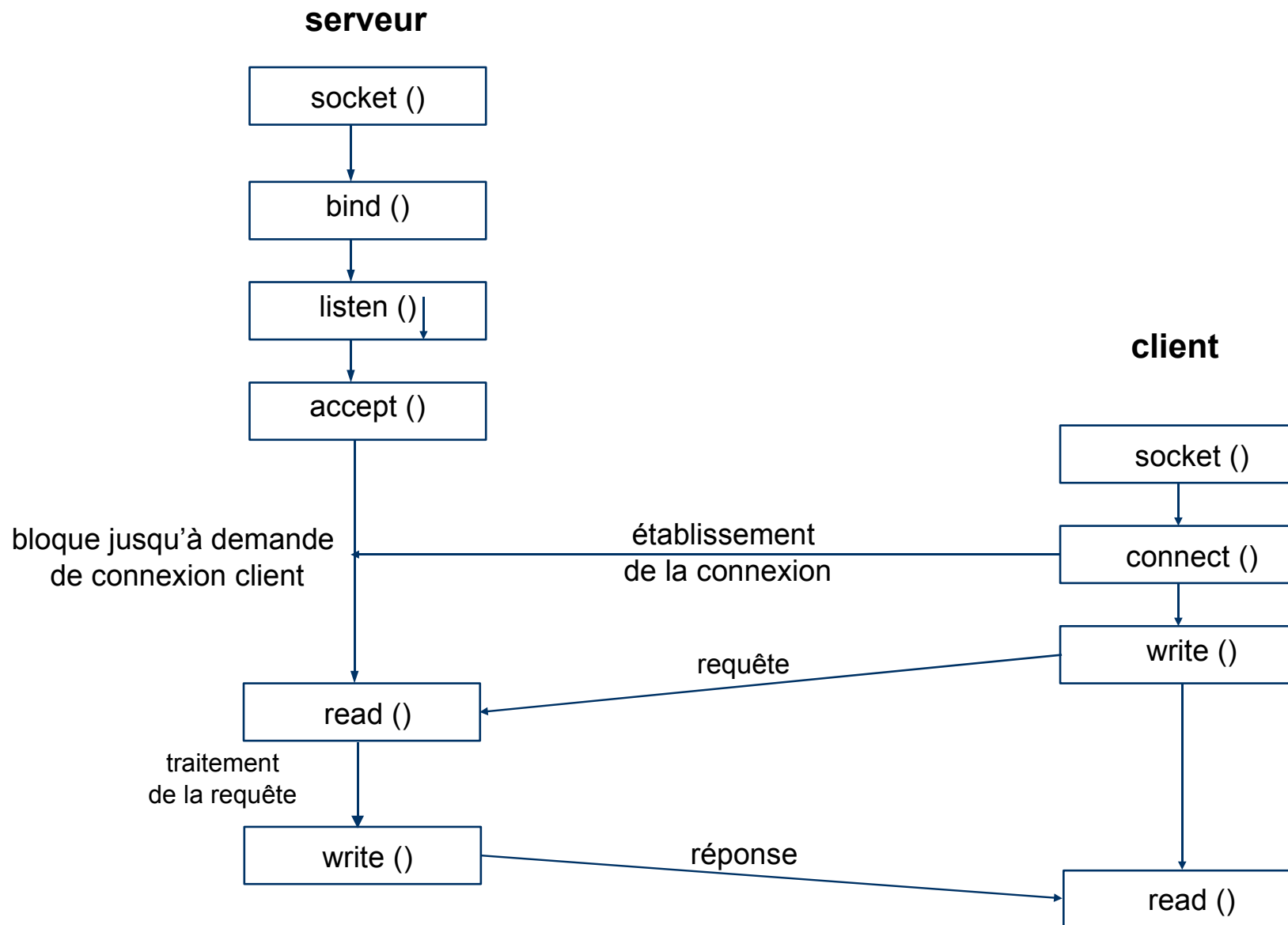
```
▶ exp: envoi os.write(fd, "%2d" %(len("truc")))
▶ exp: envoi « truc » os.write...
▶ dest : reçoit msg_len = int(os.read (fd, 2))
▶ dest : fread(truc_buf,msg_len, ...)
```

Attention à éviter les décalages intempestifs !

# Synchronisation Source/Destination

- ▶ Mode de fonctionnement Asynchrone
  - ▶ Capacité **limitée** de stockage à destination
    - ▶ Contrôle de flux ?
      - ▶ Aucun : à la charge du protocole sous-jacent
      - ▶ Possibilité de perdre des messages avec UDP ...
    - ▶ Capacité de stockage (un peu) configurable
  - ▶ Le destinataire lit (consomme) les données reçues quand il le souhaite
    - ▶ Eventuellement très longtemps après leur réception...
  - ▶ Possibilité de blocage
    - ▶ Si contrôle de flux (TCP)
    - ▶ Mode bloquant débrayable

# Synoptique en mode connecté



# Synoptique en mode non connecté

