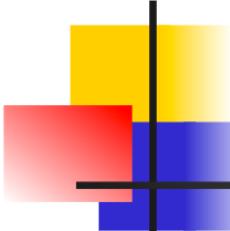


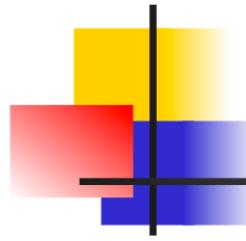
Client - Serveur

ESTEM 2016



Le visage de l'Internet (1)

- Un réseau de réseaux
- Un ensemble de logiciels et de protocoles
- Basé sur l' architecture TCP/IP
- Fonctionne en mode Client/Serveur
- Offre un ensemble de **services** (e-mail, transfert de fichiers, connexion à distance, WWW, ...)
- Une somme « d'inventions » qui s'accumulent
 - mécanismes réseau de base (TCP/IP)
 - gestion des noms et des adresses
 - des outils et des protocoles spécialisés
 - le langage HTML

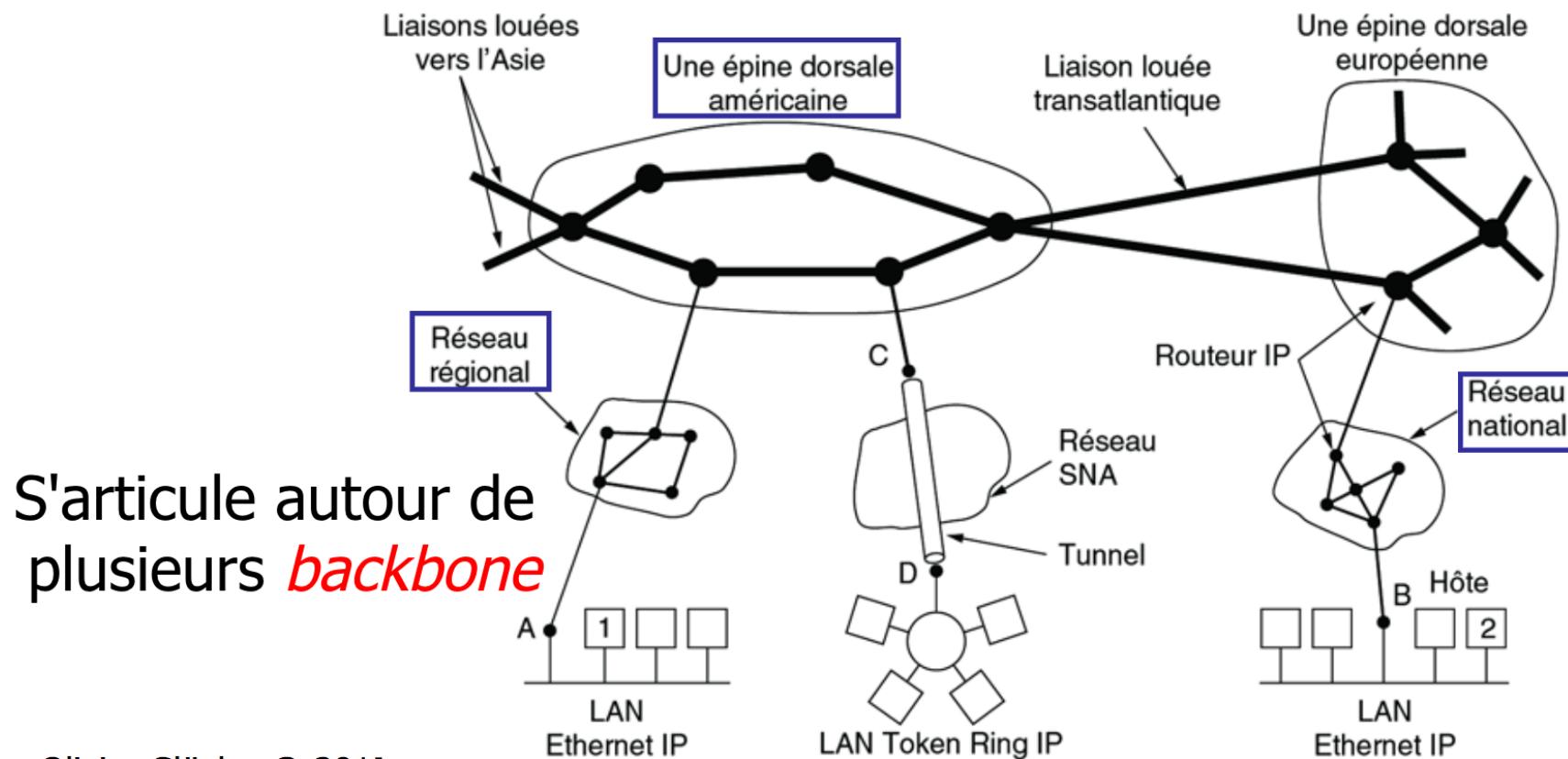


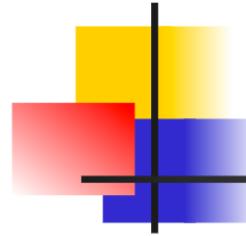
Le visage de l'Internet (2)

- Une construction à partir du « bas »
 - réseau local (laboratoire, département)
 - réseau local (campus, entreprise)
 - réseau régional
 - réseau national
 - réseau mondial
- 3 niveaux d'interconnexion
 - postes de travail (ordinateur, terminal...)
 - liaisons physiques (câble, fibre, RTC...)
 - routeurs (équipement spécialisé, ordinateur...)

Le visage de l'Internet (3)

- Un ensemble de sous-réseaux indépendants (*Autonomous System*) et hétérogènes qui sont interconnectés (organisation hiérarchique)





L'architecture de TCP/IP (1)

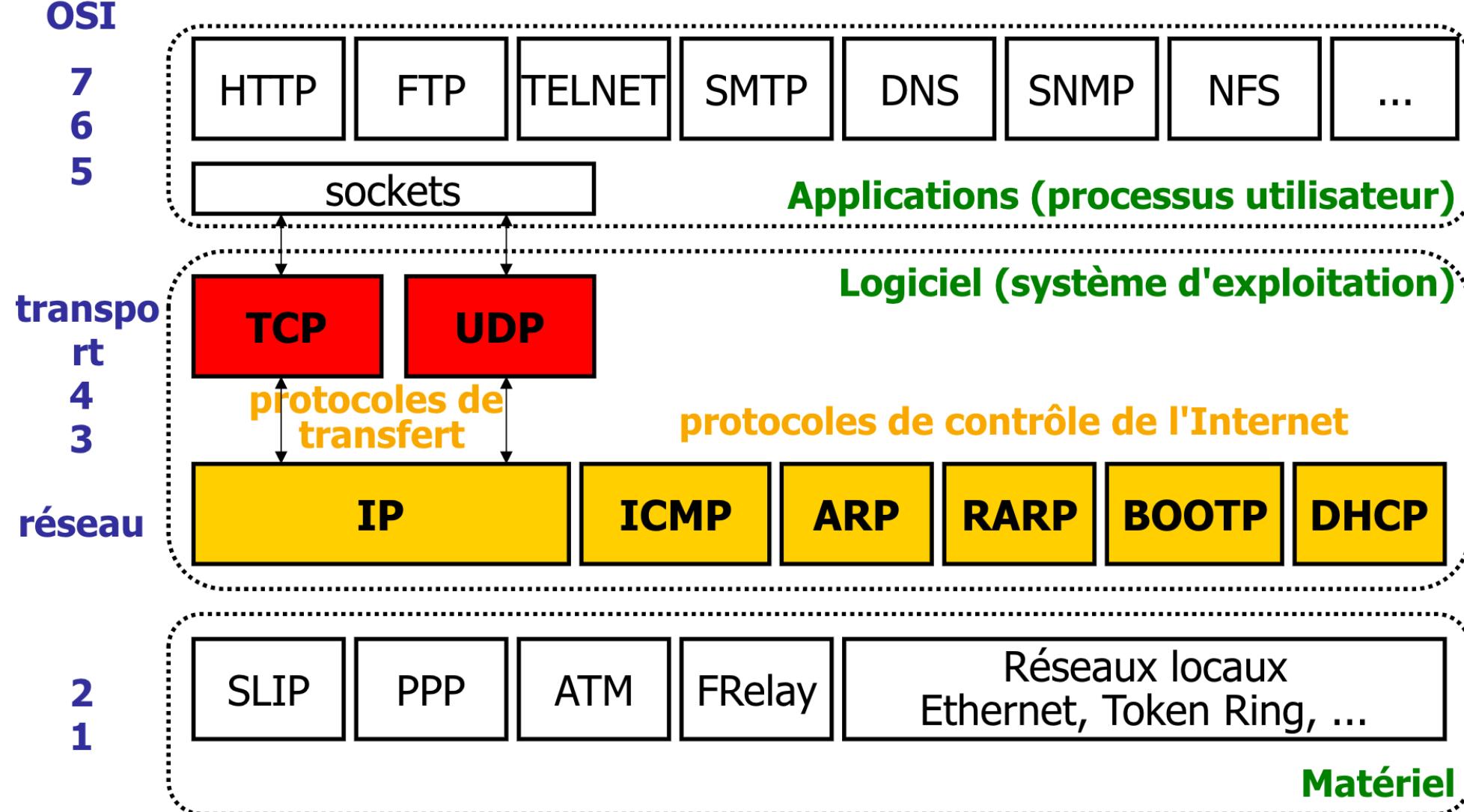
- Une version simplifiée du modèle OSI
 - Application FTP, WWW, telnet, SMTP, ...
 - Transport TCP, UDP (entre 2 processus aux extrémités)
 - TCP : transfert fiable de données en mode connecté
 - UDP : transfert non garanti de données en mode non connecté
 - Réseau IP (routage)
 - Physique transmission entre 2 sites

TCP *Transport Control Protocol*

UDP *User Datagram Protocol*

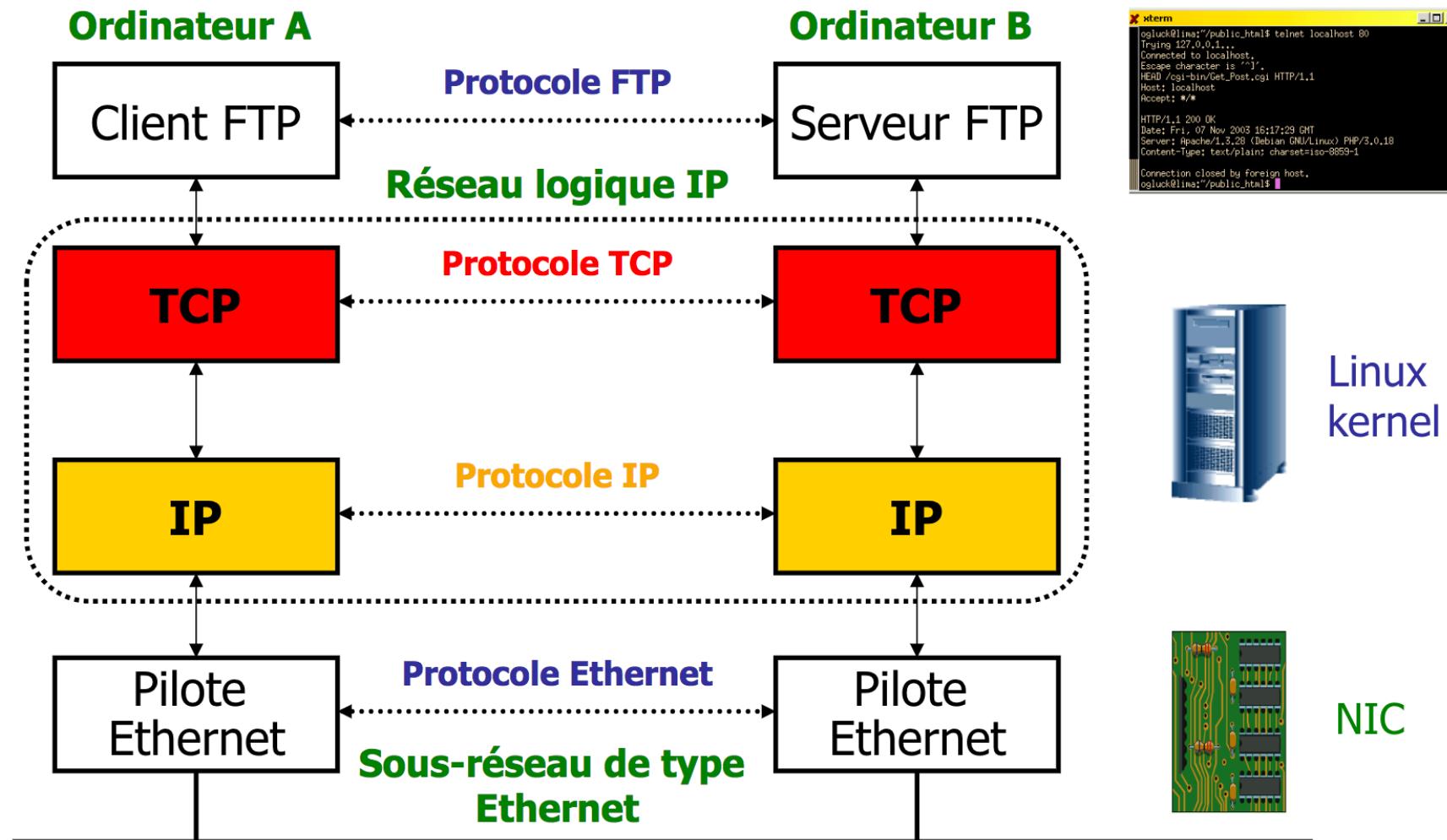
IP *Internet Protocol*

L'architecture de TCP/IP (2)



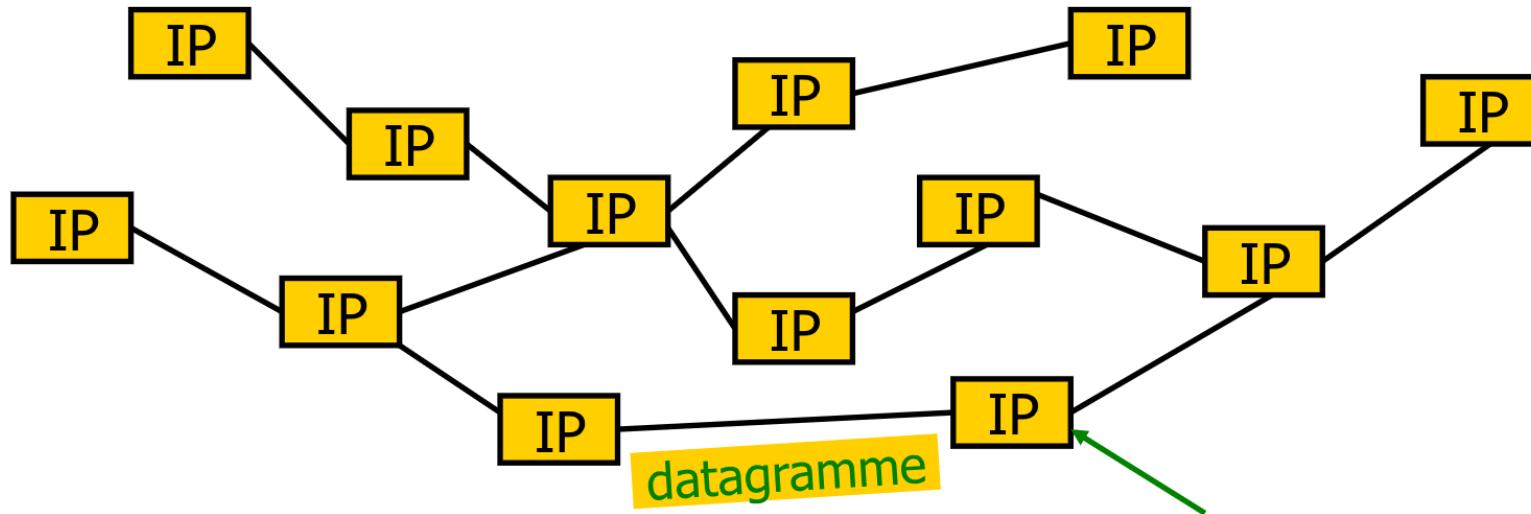
L'architecture de TCP/IP (3)

- Deux machines sur un même sous réseau IP



L' architecture de TCP/IP (5)

Couche réseau : communications entre machines

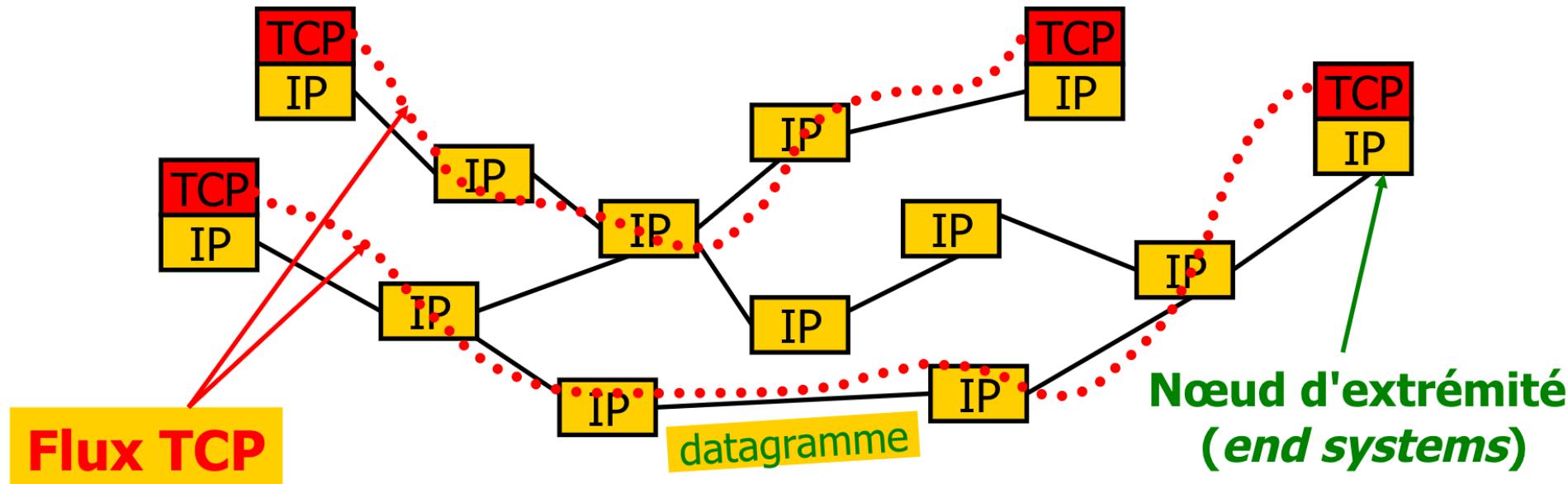


Nœud intermédiaire : routeur
(matériel ou logiciel)

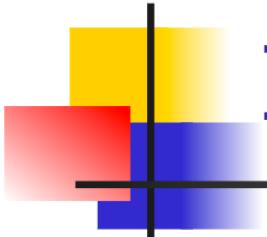
- IP - protocole d'interconnexion, best-effort
 - acheminement de **datagrammes** (mode **non connecté**)
 - peu de fonctionnalités, pas de garanties
 - simple mais robuste (défaillance d'un nœud intermédiaire)

L' architecture de TCP/IP (6)

Couche transport : communications entre applis

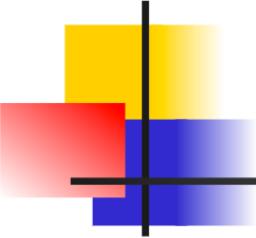


- TCP - protocole de transport **de bout en bout**
 - uniquement présent **aux extrémités**
 - transport **fiable** de **segments** (mode connecté)
 - protocole complexe (retransmission, gestion des erreurs, séquencement, ...)



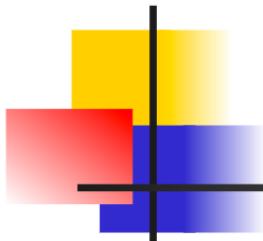
Identification des protocoles (2)

- Une adresse de transport = une adresse IP + un numéro de port (16 bits) -> adresse de socket
- Une connexion s'établit entre une socket source et une socket destinataire -> une connexion = un quintuplé (proto, @src, port src, @dest, port dest)
- Deux connexions peuvent aboutir à la même socket
- Les ports permettent un multiplexage ou démultiplexage de connexions au niveau transport
- Les ports inférieurs à 1024 sont appelés **ports réservés**



Le protocole UDP

- 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



Les utilisations d'UDP

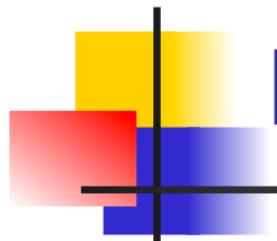
- Performance sans garantie de délivrance
- Souvent utilisé pour les applications multimédias
 - tolérantes aux pertes
 - sensibles au débit
- Autres utilisations d'UDP
 - applications qui envoient peu de données et qui ne nécessitent pas un service fiable
 - exemples : DNS, SNMP, BOOTP/DHCP
- Transfert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



Le protocole TCP

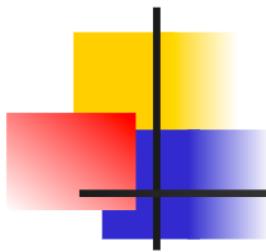
- Transport Control Protocol (RFC 793, 1122, 1323, 2018, 2581)

Attention: les RFCs ne spécifient pas tout - beaucoup de choses dépendent de l'implémentation
- 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)



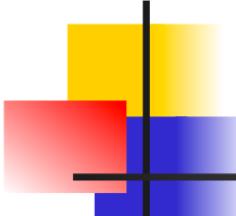
Exemples de protocole applicatif (1)

- HTTP - HyperText Transport Protocol
 - protocole du web
 - échange de requête/réponse entre un client et un serveur web
- FTP - File Transfer Protocol
 - protocole de manipulation de fichiers distants
 - transfert, suppression, création, ...
- TELNET - TELetypewriter Network Protocol
 - système de terminal virtuel
 - permet l'ouverture d'une session distante



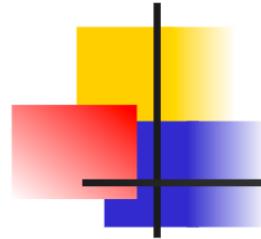
Exemples de protocole applicatif (2)

- SMTP - Simple Mail Transfer Protocol
 - service d'envoi de courrier électronique
 - réception (POP, IMAP, IMAPS, ...)
- DNS - Domain Name System
 - assure la correspondance entre un nom symbolique et une adresse Internet (adresse IP)
 - bases de données réparties sur le globe
- SNMP - Simple Network Management Protocol
 - protocole d'administration de réseau (interrogation, configuration des équipements, ...)
- Les sockets - interface de programmation permettant l'échange de données (via TCP ou UDP)



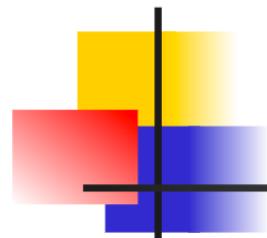
Les applications réseau (1)

- 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, ...)



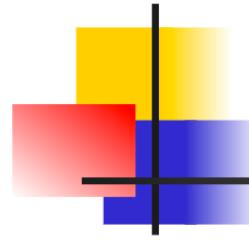
Les applications réseau (2)

- 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 !



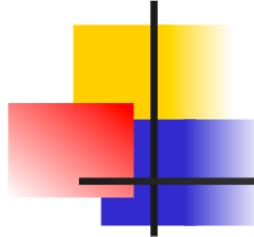
Protocoles de la couche Applications

- 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)



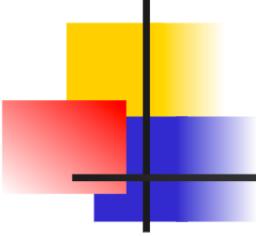
Le modèle Client / Serveur

- 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**)



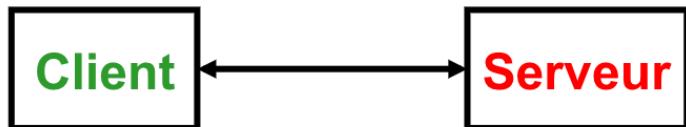
Le modèle Client / Serveur

- 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



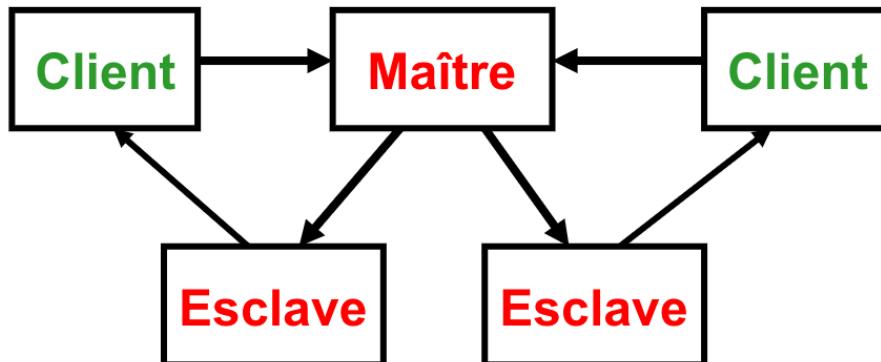
Des clients et des serveurs...

Un client, un serveur :



Requête/Réponse

Plusieurs clients, un serveur :

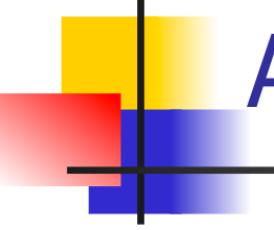


Le serveur traite plusieurs requêtes simultanées

Un client, plusieurs serveurs :

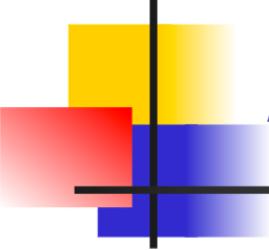


Le serveur contacté peut faire appel à un service sur un autre serveur (ex. SGBD)



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 !



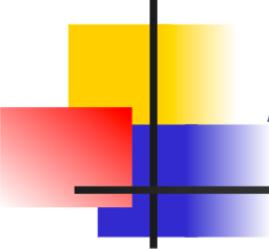
Exemple : Serveur Socket PHP

```
<?php
// Ce serveur attend une connexion avec Telnet sur le port 12000. Il envoie un message
au client et quitte

$address = '0.0.0.0';
$port = '12000';
$sock = socket_create(AF_INET, SOCK_STREAM, 0);
socket_bind($sock, $address, $port);
socket_listen($sock);

// On Attend une connexion
$client = socket_accept($sock);
// On écrit au client un message de bienvenue
socket_write($client,"Bienvenue sur le serveur\n");
// On écrit sur la console du serveur un message
echo(" --> Un client vient de se connecter\n");

// On Ferme et on quitte
socket_close($client);
socket_close($sock);
?>
```



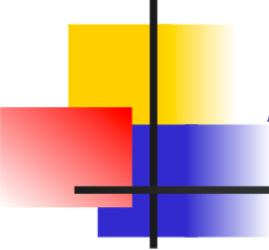
Exemple : Serveur Socket PHP

```
<?php
// Ce serveur reste permanent. Il reçoit des connexions, envoi un message, ferme la
connexion et attend la suivante

$address = '0.0.0.0';
$port = '12000';

$sock = socket_create(AF_INET, SOCK_STREAM, 0);
socket_bind($sock, $address, $port);
socket_listen($sock);

// Ici on fait une boucle pour que le serveur reste à l'écoute même après une connexion
client
while(true){
    $client = socket_accept($sock);
    socket_write($client,"Bienvenue sur le serveur\n");
    echo(" --> Un client vient de se connecter\n");    socket_close($client);
}
socket_close($sock);
?>
```



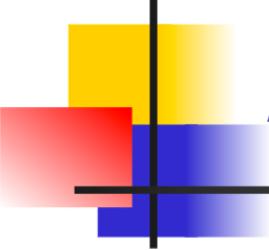
Exemple : Serveur Socket PHP

```
// Ici on remplace la boucle. Quand un client se connecte on laisse la connexion ouverte

while(true){
    $client = socket_accept($sock);
    echo(" --> Un client vient de se connecter\n");
    $sortieClient = false;
    socket_write($client,"Tapez votre commande : \n\n");

    // Cette boucle garde la connexion ouverte avec notre client
    $sortieClient = false;
    while($sortieClient === false){
        $input = trim(socket_read($client, 100000));
        echo "Le client a tapé : ".$input."\n";
        $sortieClient = true;
        socket_close($client);
    }
}

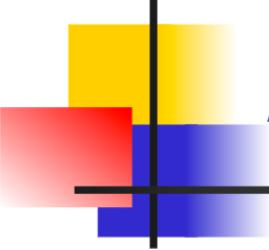
// L'inconvénient est que nous ne pouvons pas quitter, sauf en « tuant » le serveur
```



Exemple : Serveur Socket PHP

```
// Dans la boucle qui gère le client, nous pouvons gérer les commandes qu'il tape

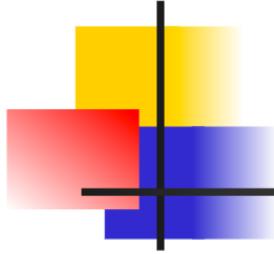
while($sortieClient === false){
    $input = trim(socket_read($client, 100000));
    echo "Le client a tapé : ".$input."\n";
    switch($input){
        case "bonjour":
            socket_write($client,"Bonjour mon ami\n");
            break;
        case "bye":
            $sortieClient = true;
            socket_write($client,"A bientot\n");
            socket_close($client);
            break;
    }
}
// L'utilisateur peut maintenant taper « bye » pour fermer sa connexion
```



Exemple : Serveur Socket PHP

```
// Nous allons maintenant interroger un webservice externe. Rajoutons un case dans
// votre switch

case "priere":
    socket_write($client,"Nous lançon la requete ... un peu de patience\n");
    $lien = 'http://muslimsalat.com/casablanca.json';
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $lien);
    curl_setopt($curl, CURLOPT_COOKIESESSION, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $return = curl_exec($curl);
    $salat = json_decode($return);
    socket_write($client,"Fajr : ".$salat->items[0]->fajr
                  ."\\nShurooq : ".$salat->items[0]->shurooq
                  ."\\nShuhru : ".$salat->items[0]->dhuhr
                  ."\\nAsr : ".$salat->items[0]->asr
                  ."\\nMaghrib : ".$salat->items[0]->maghrib
                  ."\\nIsha : ".$salat->items[0]->isha
                  ."\\n=====\\n");
    curl_close($curl);
break;
```



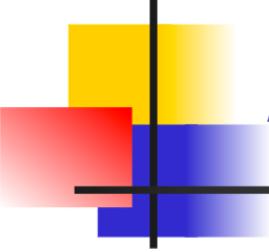
Qu'est ce qu'un Service Web

Un service web est l'interface entre l'application et la base de données.

Dans les bonnes pratiques, l'application n'accédera JAMAIS à la base de données directement mais passera par un service Web.

Il existe différentes normes pour développer les services Web.

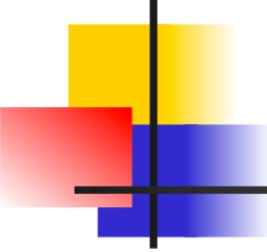
Un service Web peut être mis à disposition du public. C'est par exemple le cas des API de Facebook, Twitter



Service Web REST

Une architecture REST doit respecter les contraintes suivantes :

- Client-serveur : les responsabilités sont séparées entre le client et le serveur. L'interface utilisateur est séparée de celle du stockage des données. Cela permet aux deux d'évoluer indépendamment.
- Sans état : chaque requête d'un client vers un serveur doit contenir toute l'information nécessaire pour permettre au serveur de comprendre la requête, sans avoir à dépendre d'un contexte conservé sur le serveur. Cela libère de nombreuses interactions entre le client et le serveur.
- Mise en cache : le serveur envoie une réponse qui donne l'information sur la propension de cette réponse à être mise en cache, comme la fraîcheur, sa date de création, si elle doit être conservée dans le futur. Cela permet à des serveurs mandataires de décharger les contraintes sur le serveur et aux clients de ne pas faire de requêtes inutiles. Cela permet également d'améliorer l'extensibilité des serveurs.
- Une interface uniforme ; cette contrainte agit selon quatre règles essentielles :
 - l'identification des ressources : chaque ressource est identifiée unitairement ;
 - la manipulation des ressources à travers des représentations : les ressources ont des représentations définies ;
 - un message auto-descriptif : les messages expliquent leur nature. Par exemple, si une représentation en HTML est codée en UTF-8, le message contient l'information nécessaire pour dire que c'est le cas ;
 - Hypermédia comme moteur d'état de l'application : chaque accès aux états suivants de l'application est décrit dans le message courant.



Service Web REST : Exemple

<http://api.football-data.org/documentation>

<http://muslimsalat.com/>