

# Chapitre 2

## Protocoles de la couche Application

**LOGI00/GTII00** Programmation et Réseautique en génie logiciel/des TI

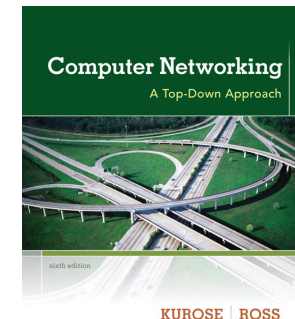
Le contenu de cette présentation est basé sur le livre de Kurose et Ross  
et de la documentation y jointe :

*Computer Networking: A Top Down Approach, 6<sup>ème</sup> édition.*

Jim Kurose, Keith Ross

Addison-Wesley, Mars 2012,

ISBN-13: 978-0132856201



# Chapitre 2: la couche application

## Objectifs:

- ❑ Comprendre le fonctionnement de quelques protocoles parmi les plus connus de la couche application
  - ❖ HTTP
  - ❖ FTP
  - ❖ SMTP / POP3 / IMAP
  - ❖ DNS
- ❑ Conception et implémentation des protocoles des apps réseaux
  - ❖ Modèles de services de la couche transport
  - ❖ Le paradigme client-serveur
  - ❖ Le paradigme poste-à-poste (P2P)

# Chapitre 2: la couche application

1. Principes des apps réseaux
  1. Architectures des apps
  2. Interfaces de connexion (Socket)
  3. Exigences des apps en termes de performance et de fiabilité
2. Web et HTTP
3. Protocole de transfert de fichier (FTP)
4. Courrier électronique (SMTP, POP3, IMAP)
5. Serveur de noms DNS

# Quelques apps réseaux

- ❑ e-mail
- ❑ Accès à distance
- ❑ Web
- ❑ Commerce électronique
- ❑ messagerie instantanée
- ❑ Partage de fichiers P2P
- ❑ Voix sur IP (Skype)
- ❑ Jeux interactifs
- ❑ Diffusion de vidéos (YouTube, Netflix)
- ❑ Réseaux sociaux (Facebook, Instagram, WeChat...)
- ❑ Application basée sur la localisation (Waze,
- ❑ Informatique nuagique (cloud computing: Amazon EC2, Google App engine, Azure...) ...

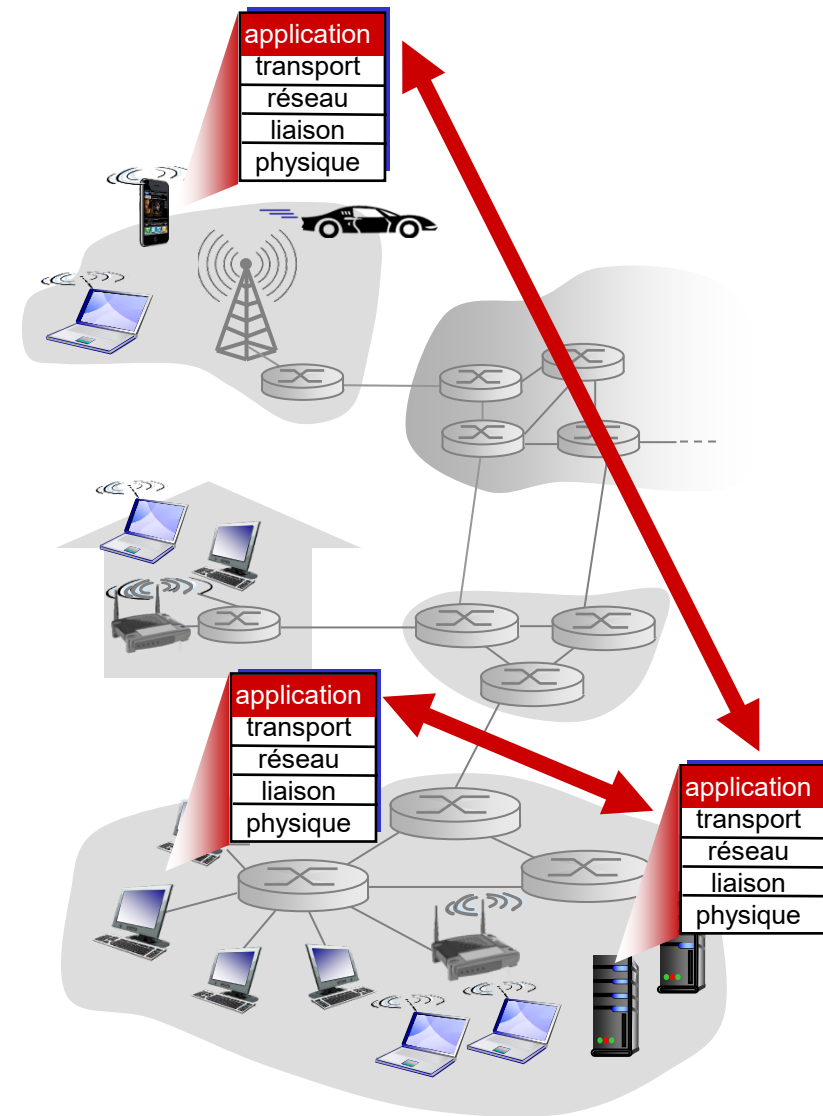
# Créer une app réseau

## Écrire des programmes qui

- ❖ Roulent sur des hôtes différents
- ❖ communiquent à travers le réseau
- ❖ Par exemple, un serveur web communique avec le navigateur web

## Pas besoin d'écrire du code pour les équipements du cœur du réseau

- ❖ Les équipements du cœur de réseau n'exécutent pas les applications des utilisateurs
- ❖ L'exécution des apps sur les hôtes seulement facilite leur développement et accélère leur adoption



# Architectures des apps

- Client-serveur

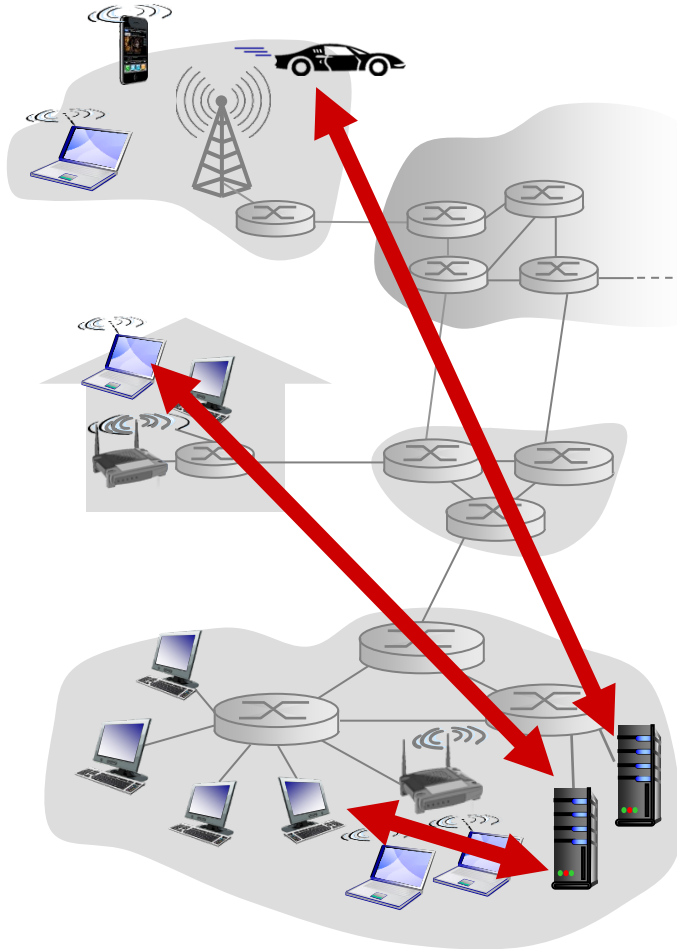
(applications web, messagerie électronique, accès à distance...)

- pair-à-pair (*peer-to-peer* - P2P)

(échange de contenu: Napster, BitTorrent, Gnutella;  
messagerie instantanée...)

# Architecture client-serveur

client/serveur

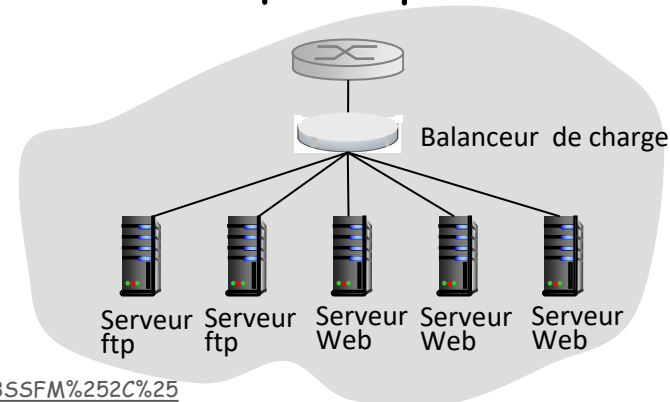


Serveur:

- ❖ Hôte toujours actif
- ❖ Adresse IP permanente
- ❖ Centre de données

Client:

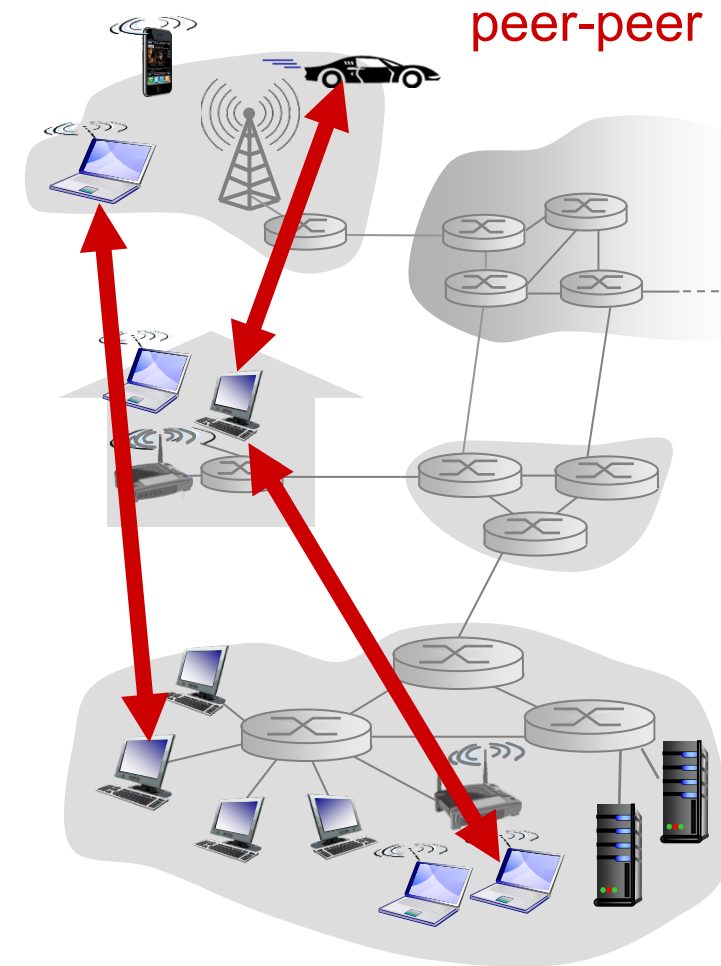
- ❖ Communique avec le serveur
- ❖ Connexions intermittentes
- ❖ Adresses IP peuvent être dynamiques
- ❖ Ne communiquent pas directement entre eux



[https://www.google.com/search?q=server+farm&client=firefox-b-d&tbm=isch&source=iu&ictx=1&fir=SnrmCKtsMxI1DM%253A%252CqX3WNGsor3SSFM%252C%252Fm%252F01n806&vet=1&usq=AI4\\_-kTTqisqp9E-yuEjJrHCKEdA1mQRiw&sa=X&ved=2ahUKEwi-kc3PhsTkAhXOVN8KHeLEDE4Q\\_B0wE3oECAUQAw#imgrc=SnrmCKtsMxI1DM:](https://www.google.com/search?q=server+farm&client=firefox-b-d&tbm=isch&source=iu&ictx=1&fir=SnrmCKtsMxI1DM%253A%252CqX3WNGsor3SSFM%252C%252Fm%252F01n806&vet=1&usq=AI4_-kTTqisqp9E-yuEjJrHCKEdA1mQRiw&sa=X&ved=2ahUKEwi-kc3PhsTkAhXOVN8KHeLEDE4Q_B0wE3oECAUQAw#imgrc=SnrmCKtsMxI1DM:)

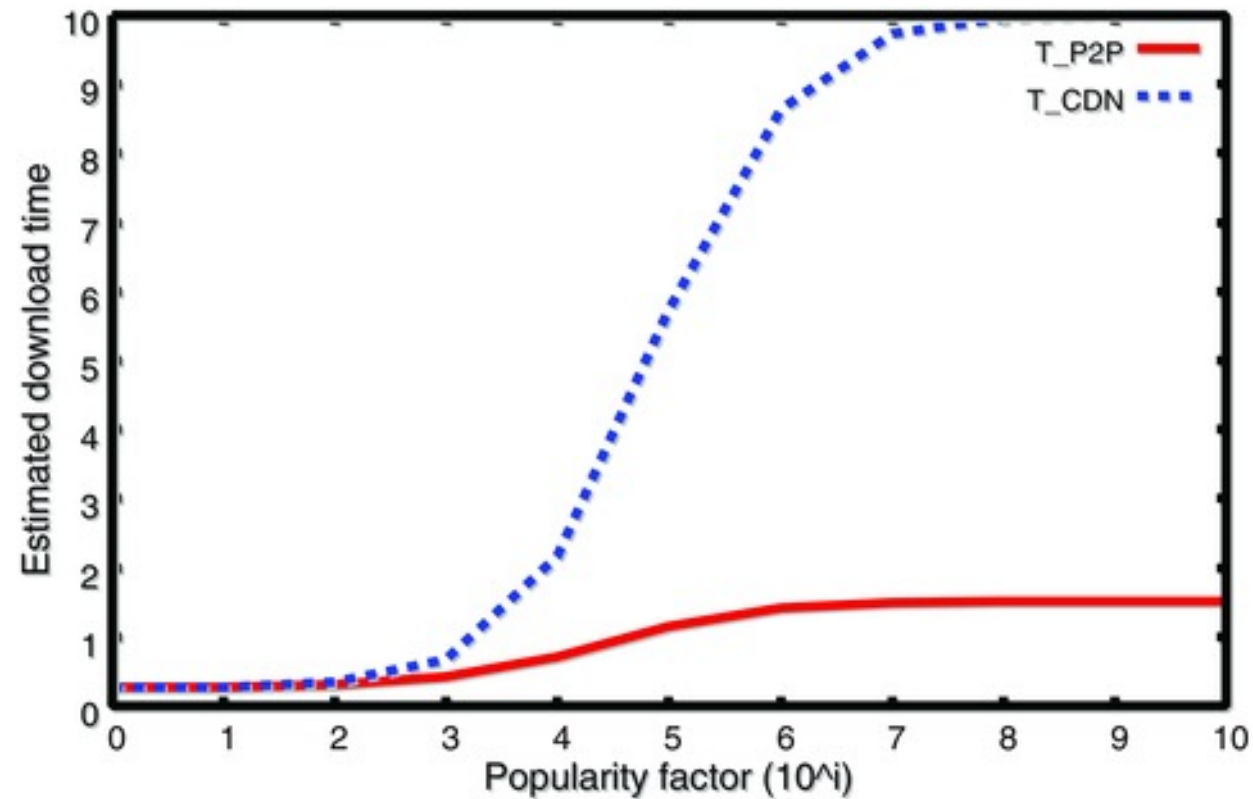
# Architecture P2P Pure

- ❑ Il n'y a pas de serveur qui est toujours actif
- ❑ Les hôtes communiquent directement d'une façon arbitraire
- ❑ Les hôtes "peers" demandent des services de la part des autres hôtes "peers"
- ❑ Chaque hôte a les deux processus client et serveur
- ❑ Les postes sont connectés d'une manière intermittente et peuvent changer d'adresses IP
- ❑ P2P Pure vs P2P hybride
  - ❑ P2P hybride présente un certain niveau de centralisation
  - ❑ **Hautement extensible mais difficile à gérer**
  - ❑ Ex: Bit Torrent, Skype





# Architecture P2P



[https://www.researchgate.net/publication/262077076\\_Analysis\\_and\\_design\\_of\\_peer-assisted\\_video-on-demand\\_services](https://www.researchgate.net/publication/262077076_Analysis_and_design_of_peer-assisted_video-on-demand_services)

# Chapitre 2: la couche application

1. Principes des apps réseaux
  1. Architectures des apps
  2. Interfaces de connexion (Socket)
  3. Exigences des apps en termes de performance et de fiabilité
2. Web et HTTP
3. Protocole de transfert de fichier (FTP)
4. Courrier électronique (SMTP, POP3, IMAP)
5. Serveur de noms DNS

# Communication entre processus

**Processus:** un programme exécuté sur un hôte.

- Dans le même hôte, deux processus communiquent avec **communication interprocessus** (définie par le système d'exploitation).
- Processus dans différents hôtes communiquent en échangeant des **messages**

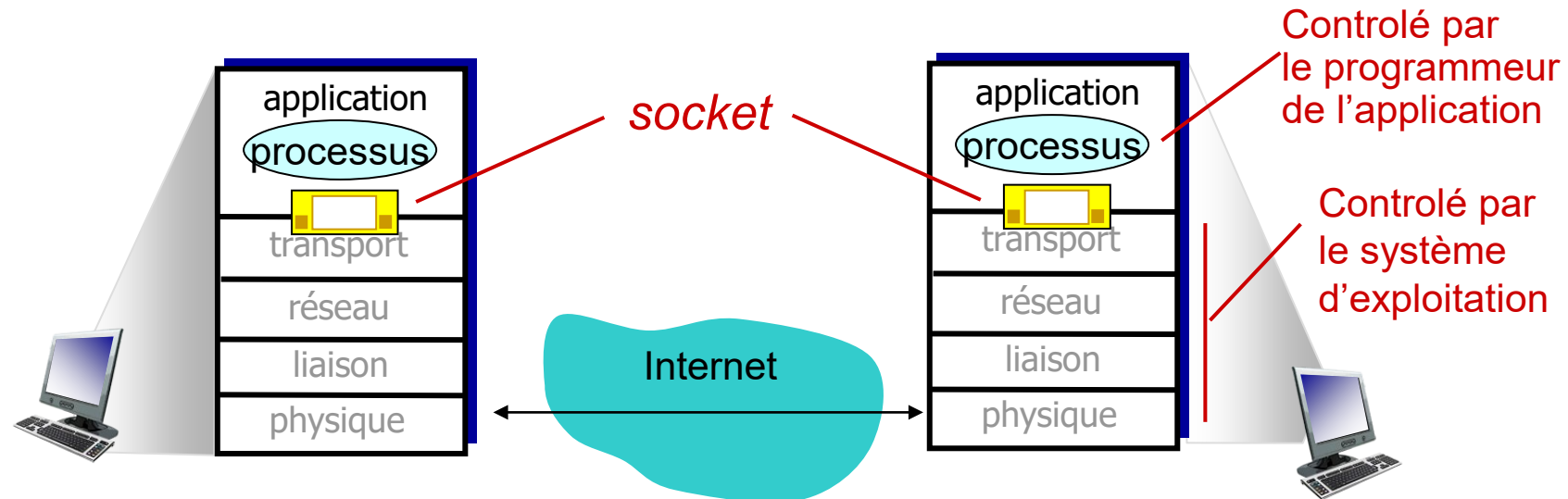
**Processus client:**  
processus qui initie la communication

**Processus serveur:**  
processus qui attend d'être contacté

- Note : les applications avec une architecture P2P ont des processus client et serveur

# Interface de connexion (Sockets)

- ❑ Processus émet/reçoit des messages à travers son **socket**
- ❑ Socket analogue à une porte
  - ❖ Processus d'émission met les messages à l'extérieur
  - ❖ Processus d'émission compte sur l'infrastructure transport de l'autre côté de la porte pour transférer les messages à la porte du côté réception



**Socket:** Interface entre un processus de la couche application et un protocole de la couche transport. Permet de différencier les données sortant d'un processus ou allant vers un processus.

# Donner une adresse à un processus

- ❑ Pour recevoir des messages, le processus doit avoir un *identifiant*
- ❑ L'équipement hôte a une adresse IP unique 32-bit (IPv4)
- ❑ Q: est ce que l'adresse IP suffit pour identifier le processus?
  - ❖ A: Non, plusieurs processus peuvent rouler sur le même hôte
- ❑ *L'identifiant* comprend l'**adresse IP** et le **numéro de port** associé au processus.
- ❑ Exemples de numéros de port:
  - ❖ serveur HTTP : 80
  - ❖ serveur courriel SMTP: 25
- ❑ Pour émettre des messages HTTP au serveur [www.etsmtl.ca](http://www.etsmtl.ca):
  - ❖ **Adresse IP**: 142.137.250.114
  - ❖ **Numéro de port**: 80
- ❑ Une connexion est définie par : **adresses IP source et destination + numéros de port source et destination + protocole de transport (TCP ou UDP)**

# Le protocole d'application définit

- ❑ Le type des messages échangés,
  - ❖ Par ex., requête, réponse
- ❑ La syntaxe du message :
  - ❖ Quels sont les champs contenu dans le message & comment ils sont délimités
- ❑ La sémantique du message
  - ❖ Sens de l'information contenu dans les champs
- ❑ Règles pour déterminer quand et comment les processus échangent des messages

## Protocoles ouverts:

- ❑ définis dans des RFCs
- ❑ Permettent l'interopérabilité
- ❑ Par ex., HTTP, SMTP

## Protocoles Propriétaires:

- ❑ Par ex., Skype

# Chapitre 2: la couche application

1. Principes des apps réseaux
  1. Architectures des apps
  2. Interfaces de connexion (Socket)
  3. Exigences des apps en termes de performance et de fiabilité
2. Web et HTTP
3. Protocole de transfert de fichier (FTP)
4. Courrier électronique (SMTP, POP3, IMAP)
5. Serveur de noms DNS

# Quels sont les services de transport dont une app a besoin?

## Fiabilité du transfert de données

- ❑ Il y a des applications exigeant une transmission **fiable** à 100%
  - ❑ par ex., transfert de fichier
- ❑ D'autres peuvent tolérer une certaine perte
  - ❑ par ex., audio

## Timing

- ❑ quelques apps exigent un faible délai pour être efficaces
  - ❑ par ex., téléphonie Internet, jeux interactifs

## Débit

- ❑ quelques apps (e.g., multimédia) exigent un débit moyen pour être "effectives"
- ❑ d'autres ("apps élastiques") utilisent le débit existant

## Sécurité et intégrité

- ❑ Encryptage, intégrité des données, ...



## Les demandes en service transport pour des apps typiques

Application	Perte	Débit	Sensibilité au temps
transfert de fichier	pas de perte	élastique	non
e-mail	pas de perte	élastique	non
documents Web	pas de perte	élastique	non
audio/vidéo temps réel	tolérant	audio: 5kbps-1Mbps vidéo:10kbps-5Mbps	oui, 100's msec
audio/vidéo stockés	tolérant	pareil	oui, qq secs
Jeux interactifs	tolérant	qq kbps	oui, 100's msec
messagerie instantanée	pas de perte	élastique	oui et non

# Les services des protocoles de transport de l'Internet

## service TCP :

- ❑ *transport fiable* : entre les processus client et serveur
- ❑ *Contrôle du flux* : l'émetteur ne submergera pas le récepteur
- ❑ *Contrôle de congestion* : réduire le débit de transmission quand le réseau est surchargé
- ❑ *Ne fournit pas de* : timing, garantie de débit minimal, sécurité
- ❑ *Orienté connexion* : création d'une connexion entre les processus client et serveur

## service UDP :

- ❑ Un transfert de données non fiable entre les processus client et serveur
- ❑ Pas de: établissement de connexion, fiabilité, contrôle de flux, contrôle de congestion, synchronisation, débit minimal, ou sécurité

Q: Pourquoi il y a UDP?

## Apps Internet: application, protocoles de transport

Application	Protocole d'application	Protocole de transport
courriel	SMTP [RFC 2821]	TCP
Accès terminal à distance	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Transfert de fichiers	FTP [RFC 959]	TCP
streaming multimédia	HTTP (par ex., YouTube), RTP [RFC 1889]	TCP ou UDP
téléphonie Internet	SIP, RTP, propriétaire (par ex., Skype)	TCP ou UDP

# Chapitre 2: la couche application

1. Principes des apps réseaux
  1. Architectures des apps
  2. Interfaces de connexion (*Socket*)
  3. Exigences des apps en termes de performance et de fiabilité
2. Web et HTTP
3. Protocole de transfert de fichier (FTP)
4. Courrier électronique (SMTP, POP3, IMAP)
5. Serveur de noms DNS

# Web et HTTP

## Un peu de jargon

- ❑ Une page Web contient des objets
- ❑ Un objet peut être un fichier HTML, une image JPEG, une applet Java, un fichier audio,...
- ❑ Une page web contient un fichier de base sous format HTML qui contient des références à des objets
- ❑ Chaque objet a une adresse URL (Unique Resource Locator)
- ❑ Exemple URL:

www.someschool.edu / someDept/pic.gif

Nome de l'hôte

Nom du chemin

# HTTP en bref

## HTTP : *hypertext transfer protocol*

- ❑ Protocole de la couche application pour le Web
- ❑ client/serveur utilisant HTTP :
  - ❖ *Le client*: fureteur qui envoie des requêtes, reçoit les objets Web et les affiche
  - ❖ *Le serveur*: le serveur Web envoie les objets Web comme une réponse à une requête



# HTTP en bref

## utilise TCP:

- ❑ Le client initie la connexion TCP (crée un socket) au serveur, port 80
- ❑ Le serveur accepte la connexion TCP du client
- ❑ Les messages HTTP échangés entre le navigateur (client HTTP) et le serveur web (serveur HTTP)
- ❑ Fermeture de la connexion TCP

## HTTP est "sans état"

- ❑ Le serveur ne garde aucune information sur les requêtes précédentes

### Les protocoles qui conservent "l'état" sont complexes!

- ❑ L'historique (état) doit être conservé
- ❑ si un serveur/client tombe en panne, leurs perceptions de "l'état" peuvent être inconsistantes et doivent être récupérées

# Connexions HTTP

## HTTP Non persistant

- ❑ Un seul objet est envoyé sur une connexion TCP.
- ❑ Une connexion TCP pour chaque objet

## HTTP Persistant

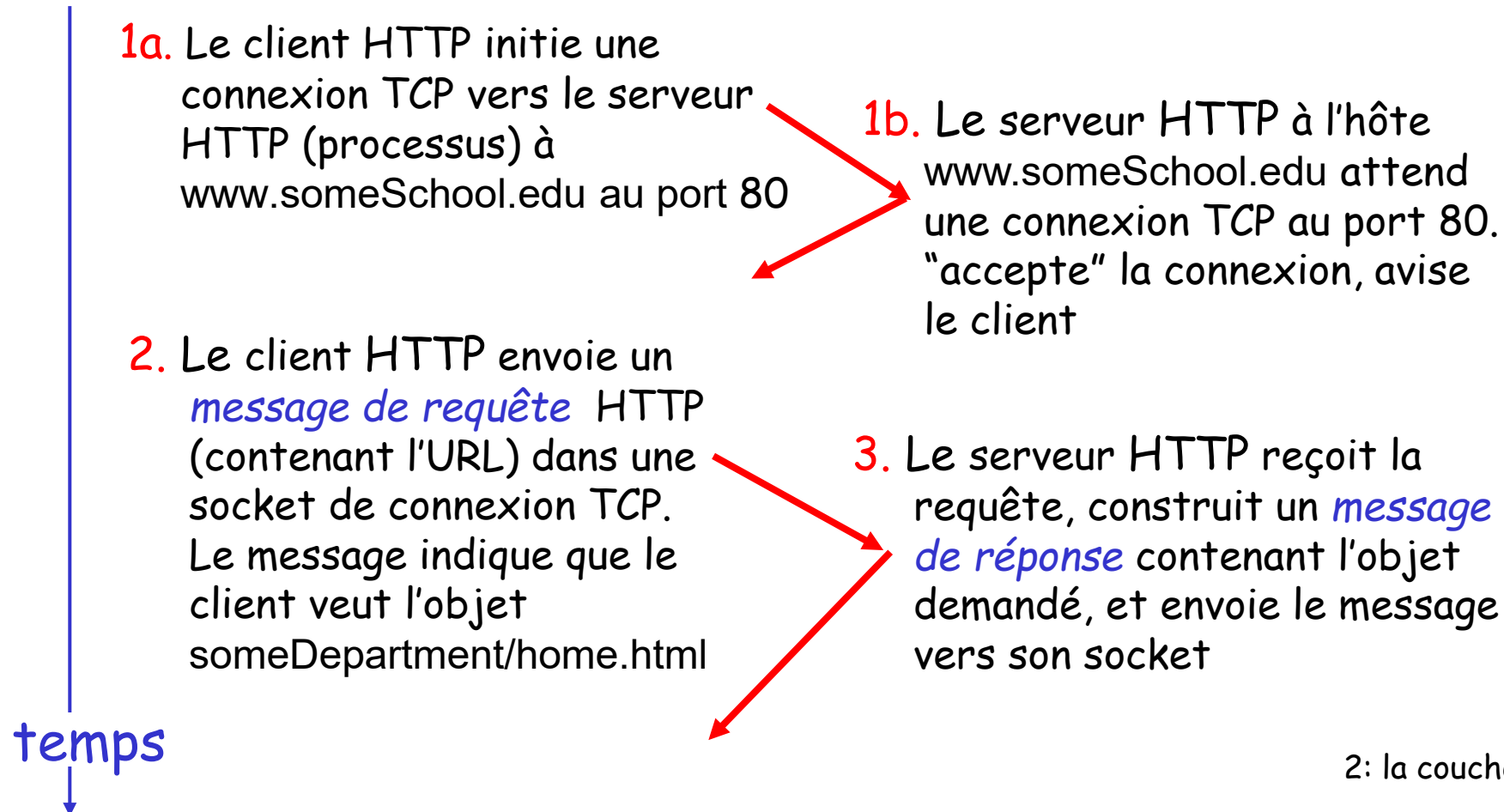
- ❑ Plusieurs objets peuvent être envoyés sur une seule connexion TCP entre le client et le serveur.



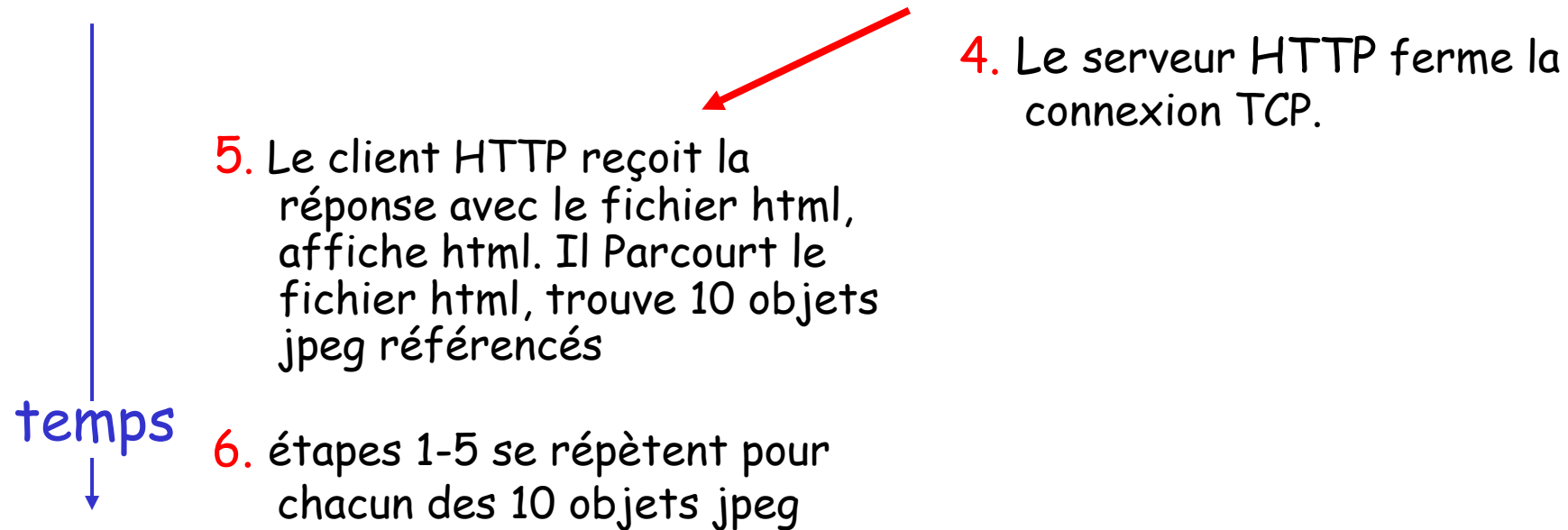
# HTTP Non persistant

Supposons l'utilisateur demande la page Web suivante qui contient des références à 10 images jpeg :

**www.someSchool.edu/someDepartment/home.html**



# HTTP Non persistant (suite)

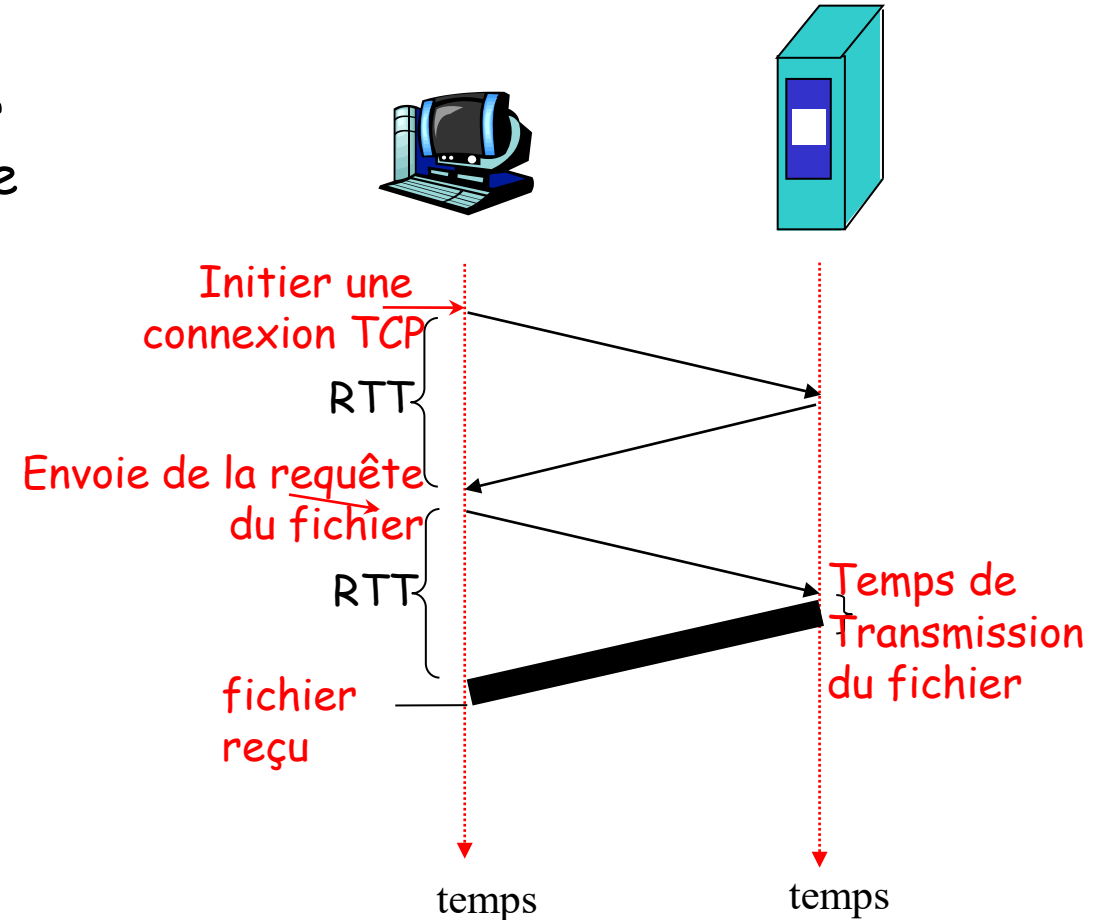


# HTTP Non-Persistant temps de réponse

**Définition du RTT (Round Trip Time):** le temps nécessaire à un petit paquet pour faire un aller-retour entre le client et le serveur.

## temps de réponse :

- ❑ un RTT pour initier la connexion TCP
- ❑ un RTT pour la requête HTTP et les premiers octets de la réponse HTTP
- ❑ Temps de transmission du fichier



**total = 2RTT + temps de transmission de l'objet demandé**

# HTTP Persistant

## Problèmes de HTTP Non-persistant :

- ❑ demande 2 RTTs par objet
- ❑ Surcharge le système d'exploitation pour chaque connexion TCP
- ❑ Les fureteurs souvent ouvrent plusieurs connexions TCP en parallèle pour télécharger les fichiers référencés

## HTTP persistant

- ❑ Le serveur laisse la connexion ouverte après l'envoi d'une réponse
- ❑ Les messages HTTP subséquents entre les mêmes client/serveur sont envoyés sur la même connexion
- ❑ Le client envoie des requêtes dès qu'il trouve un objet référencé
- ❑ Un RTT pour tous les objets référencés

# Message de requête HTTP

- Deux types de messages HTTP: *requête, réponse*
- *message de requête HTTP* :
  - ❖ ASCII (format lisible par les humains)

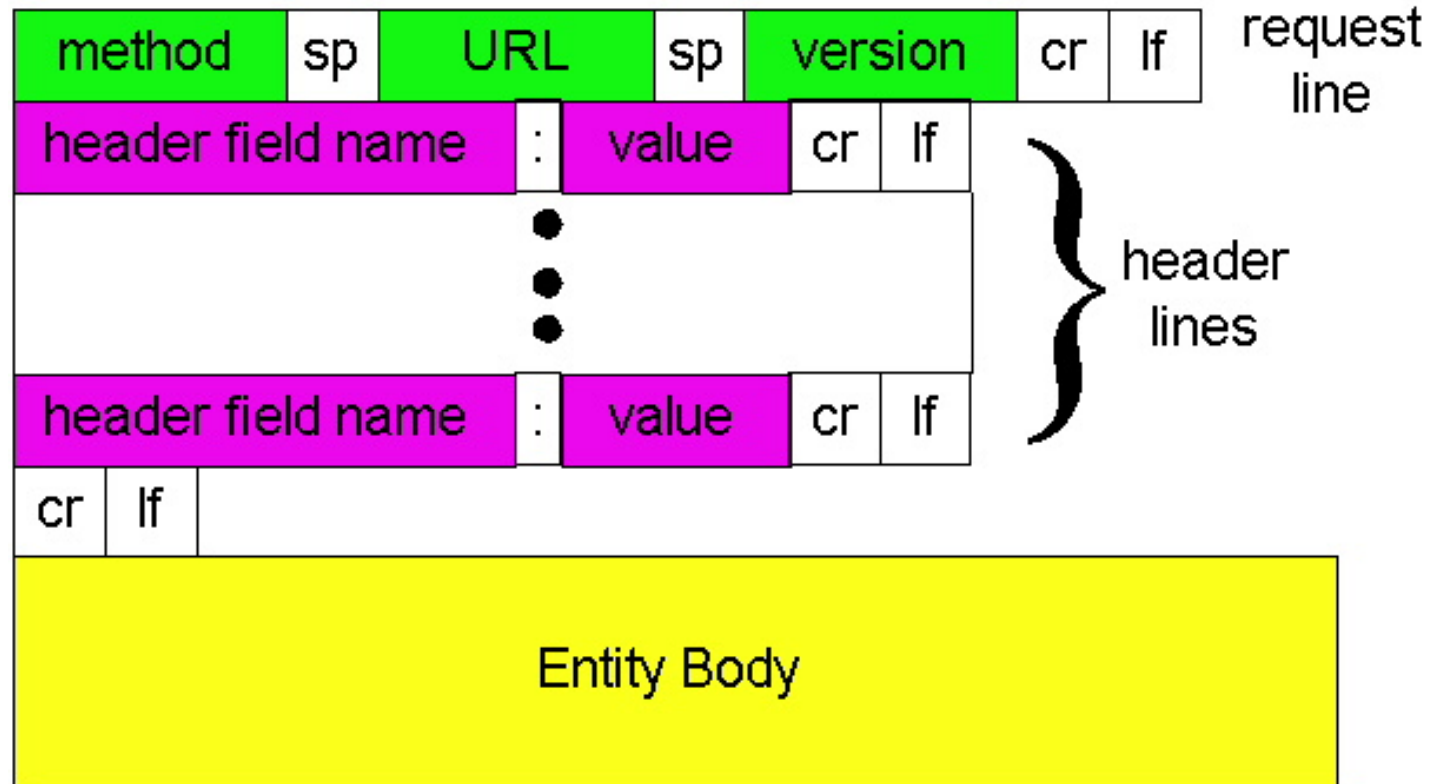
Ligne de requête  
(commandes GET,  
POST, HEAD)

Lignes de l'entête

Carriage return  
et line feed  
pour indiquer  
la fin du message

```
GET /somedir/page.html HTTP/1.1\r\n
Host: www.someschool.edu \r\n
User-agent: Mozilla/4.0\r\n
Connection: close\r\n
Accept-language: fr\r\n
\r\n
(extra carriage return, line feed)
```

# Message de requête HTTP: format général



# Envoie d'un formulaire d'entrée

## Méthode Post :

- ❑ La page Web comprend souvent un formulaire
- ❑ Les données saisies dans le formulaire sont envoyées au serveur dans le corps de l'entité

## Méthode GET :

- ❑ Utilise la méthode GET
- ❑ Les données du formulaire sont envoyées dans le champ URL de la requête :

```
GET /somedir/page.html?animal=monkey HTTP/1.1\r\n
```

# Types de méthodes

## HTTP/1.0

- ❑ GET
- ❑ POST
- ❑ HEAD
  - ❖ Demande au serveur de laisser les objets requis à l'extérieure de la réponse

## HTTP/1.1

- ❑ GET, POST, HEAD
- ❑ PUT
  - ❖ Téléverser (Uploader) le fichier dans le corps à un chemin spécifié dans l'URL
- ❑ DELETE
  - ❖ Effacer un fichier spécifié dans l'URL



# Message de réponse HTTP

Ligne d'état (protocole, Code d'état, message d'état)

→ HTTP/1.1 200 OK\r\n

Lignes  
d'entête

Connection close\r\n  
Date: Thu, 06 Aug 1998 12:00:15 GMT\r\n  
Server: Apache/1.3.0 (Unix)\r\n  
Last-Modified: Mon, 22 Jun 1998 2007 17:00:02 GMT\r\n  
Content-Length: 6821\r\n  
Content-Type: text/html\r\n  
\r\n

→ data data data data data ...

données (Par ex., le fichier HTML requis)

# Codes d'état de réponse HTTP

- ❖ Dans la première ligne de la réponse
- ❖ Quelques exemples de codes:

## **200 OK**

- ❖ Requête réussie, objet requis viendra plus tard dans le msg

## **301 Moved Permanently**

- ❖ L'objet requis est déplacé, la nouvelle localisation est spécifiée plus tard dans le message (Location:)

## **400 Bad Request**

- ❖ La requête n'est pas comprise par le serveur

## **404 Not Found**

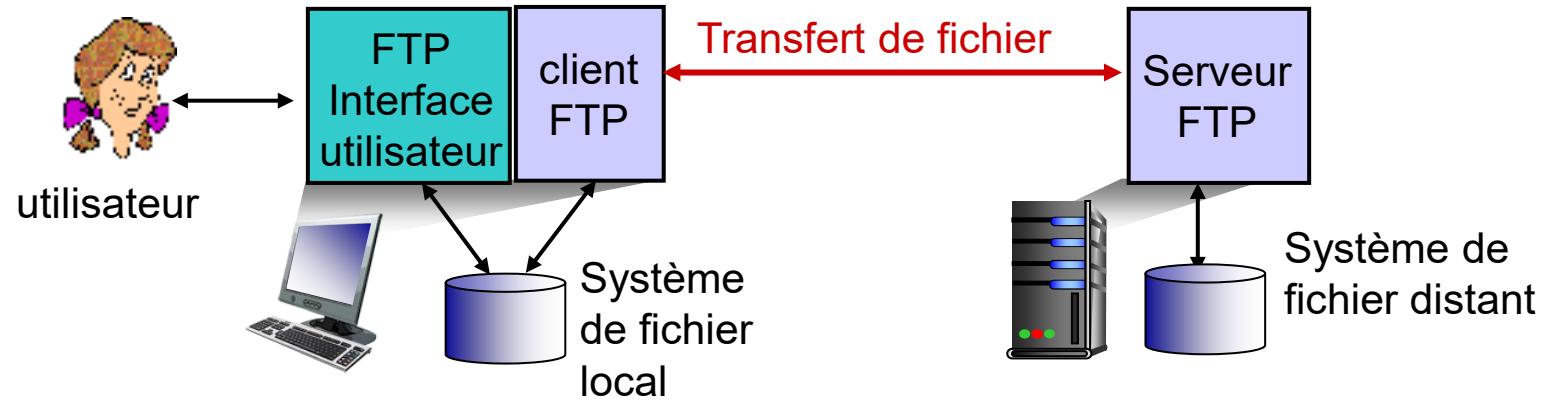
- ❖ Le document requis n'est pas sur ce serveur

## **505 HTTP Version Not Supported**

# Chapitre 2: la couche application

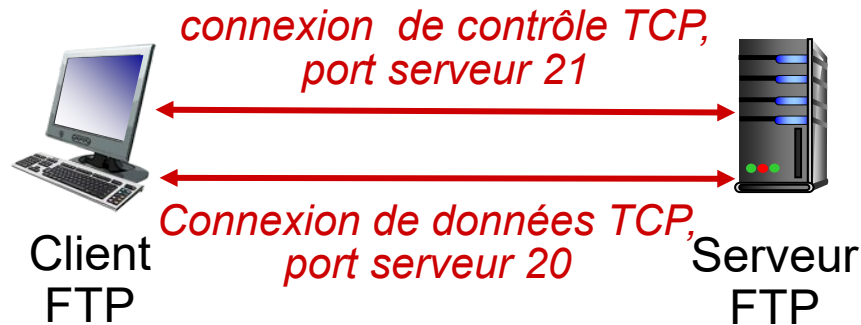
1. Principes des apps réseaux
  1. Architectures des apps
  2. Interfaces de connexion (*Socket*)
  3. Exigences des apps en termes de performance et de fiabilité
2. Web et HTTP
3. Protocole de transfert de fichier (FTP)
4. Courrier électronique (SMTP, POP3, IMAP)
5. Serveur de noms DNS

# FTP: protocole de transfert de fichiers



- ❑ Transfert de fichier vers/de une machine distante
- ❑ Modèle client/serveur
  - ❖ *Le client* : initie le transfert (de ou vers une machine distante)
  - ❖ *Le serveur* : machine distante
- ❑ ftp: RFC 959
- ❑ Le client et le serveur ftp utilisent les **ports 21 et 20**

# FTP: sépare les connexions: contrôle, données



- ❑ Connexion de contrôle: est utilisée pour envoyer les commandes
  - ❑ « Hors-bande » (out of band)
- ❑ Chaque fois que le serveur a un fichier à transmettre, il ouvre une nouvelle connexion TCP appelée connexion de données
- ❑ Le serveur FTP maintient "l'état": répertoire actuel, authentification précédente

1. Le client FTP contacte le serveur FTP au port 21 à travers TCP (connexion de contrôle)
2. Le client est authentifié sur la connexion de contrôle
3. Le client parcourt le répertoire de fichier distant en envoyant des commandes sur la connexion de contrôle.
4. Quand le serveur reçoit une commande de transfert d'un fichier,
  - ❑ il ouvre une 2<sup>ème</sup> connexion TCP (appelée connexion de données) avec le client pour transmettre le fichier
  - ❑ Après le transfert du fichier, le serveur ferme la connexion de données.

# Commandes et réponses FTP

## Ex. commandes:

- ❑ Envoyées en texte ASCII sur la connexion de contrôle
  - ❑ `USER username`
  - ❑ `PASS password`
  - ❑ `LIST` donne la liste des fichiers dans le répertoire actuel
  - ❑ `RETR filename` demande un fichier (gets)
  - ❑ `STOR filename` stocke (puts) le fichier sur la machine distante

## Ex. codes de retour

- ❑ Code d'état et message d'état (comme en HTTP)
  - ❑ 331 Username OK, password required
  - ❑ 125 data connection already open; transfer starting
  - ❑ 425 Can't open data connection
  - ❑ 452 Error writing file

# Chapitre 2: la couche application

1. Principes des apps réseaux
  1. Architectures des apps
  2. Interfaces de connexion (*Socket*)
  3. Exigences des apps en termes de performance et de fiabilité
2. Web et HTTP
3. Protocole de transfert de fichier (FTP)
4. Courrier électronique (SMTP, POP3, IMAP)
5. Serveur de noms DNS

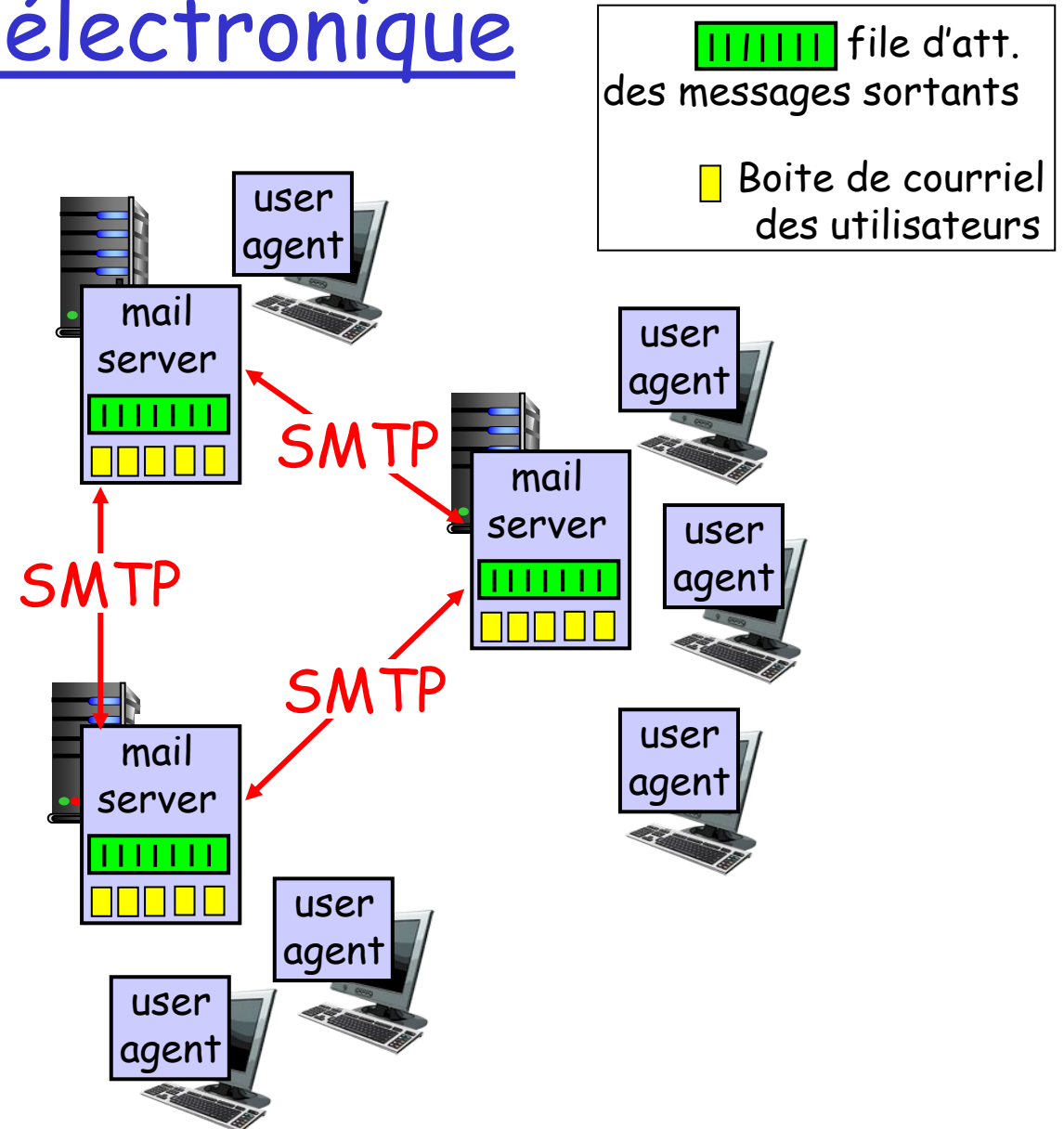
# Système de Messagerie électronique

## 3 composantes majeures:

- ❑ Agent utilisateur (*User agent*)
- ❑ Serveurs de courriel (*mail server*)
- ❑ Protocole Ex. Simple Mail Transfer Protocol (SMTP)

## Agent utilisateur (User Agent)

- ❑ "lecteur de courriel"
- ❑ Composer, éditer, lire les courriels
- ❑ Par ex., Outlook, Thunderbird
- ❑ Les messages entrants et sortants sont stockés sur le serveur

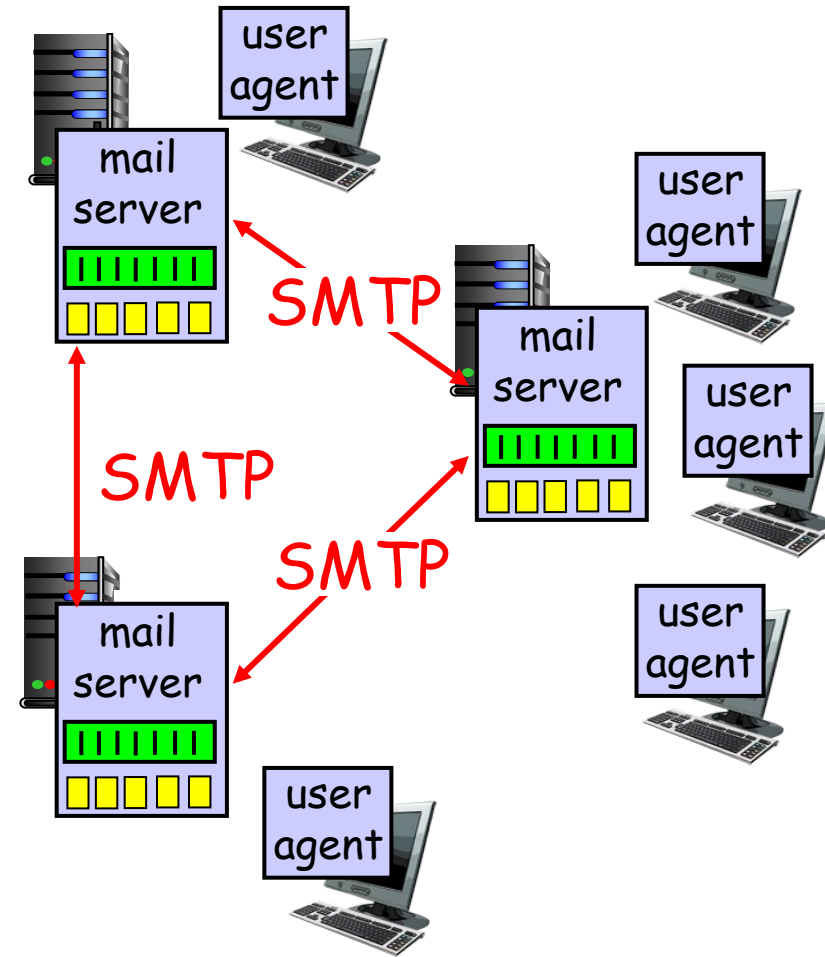




# Système de Messagerie électronique

## Serveurs de courriels (mail servers)

- ❑ **Boîte aux lettres** contient les messages entrants pour l'utilisateur
- ❑ **File d'attente des messages** pour les messages à envoyer
- ❑ **Protocole SMTP** entre les serveurs de courriels pour envoyer les messages
  - ❖ client: le serveur qui envoie les courriels
  - ❖ serveur: le serveur qui reçoit les courriels



# Système de Messagerie électronique

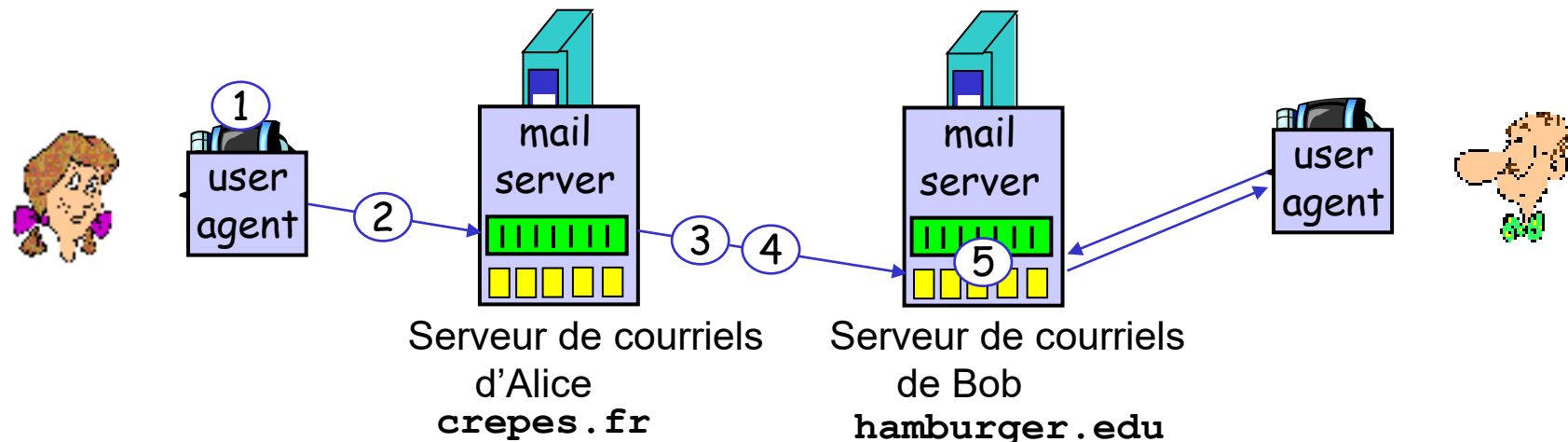
## SMTP [RFC 2821]

- ❑ Utilise le TCP pour transférer sans erreur les messages email du client au server, port 25
- ❑ Transfert direct: du serveur d'émission vers le serveur de réception
- ❑ Trois phases de transfert
  - ❖ handshaking (bonjour)
  - ❖ transfert des messages
  - ❖ fermeture
- ❑ L'interaction commande/réponse (comme http et ftp)
  - ❖ commandes: texte ASCII
  - ❖ réponse: le code d'état et message d'état
- ❑ Les messages doivent être en ASCII 7-bit

# Scénario: Alice envoie un message à Bob

- 1) Alice utilise UA pour composer le message "à" bob@hamburger.edu
- 2) l'agent utilisateur d'Alice envoie un message à son serveur mail crepes.fr le message est placé dans une file d'attente et attend son tour pour être envoyé

- 3) Le client SMTP dans le serveur crepes.fr ouvre une connexion TCP avec le serveur mail de Bob
  - 4) Il envoie le message d'Alice sur la connexion TCP au serveur SMTP résidant à hamburger.edu
  - 5) Le serveur mail de Bob place le message dans la boîte aux lettres de Bob
- Maintenant, si Bob veut récupérer son message, il utilise son agent utilisateur pour se connecter à son serveur mail et récupérer le message



# Exemple d'une interaction SMTP

```
S: 220 hamburger.edu SMTP service ready
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

S: Serveur C: Client
-------------------------

# SMTP : Résumé

- ❑ SMTP utilise des connexions persistantes
- ❑ SMTP exige que le message (entête & corps) soit en ASCII 7-bit
- ❑ Le serveur SMTP utilise CRLF.CRLF pour déterminer la fin du message

## Comparaison avec HTTP:

- ❑ HTTP: pull protocol (tirez)
- ❑ SMTP: push protocol (poussez)
- ❑ Les deux ont des interactions basées sur commande/réponse en ASCII, codes d'état
- ❑ HTTP : chaque objet est encapsulé dans son propre message de réponse
- ❑ SMTP : plusieurs objets envoyés dans un message multi-parties

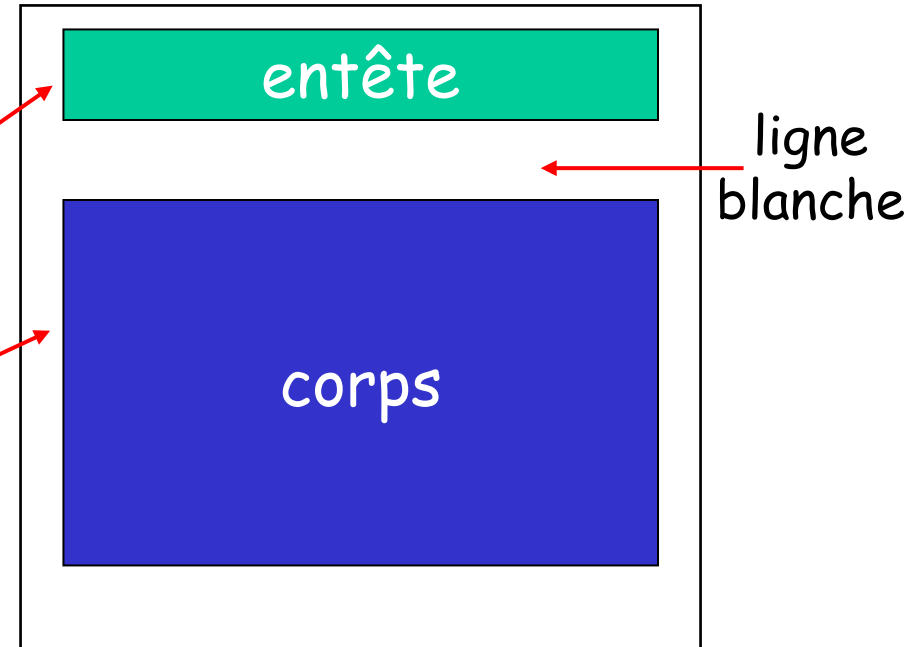
# Le format du message Mail

- ❖ **SMTP**: le protocole pour échanger des courriels
- ❖ **RFC 822**: le standard pour le format texte du message:

- Lignes d'entête, par ex.,
  - To:
  - From:
  - Subject:

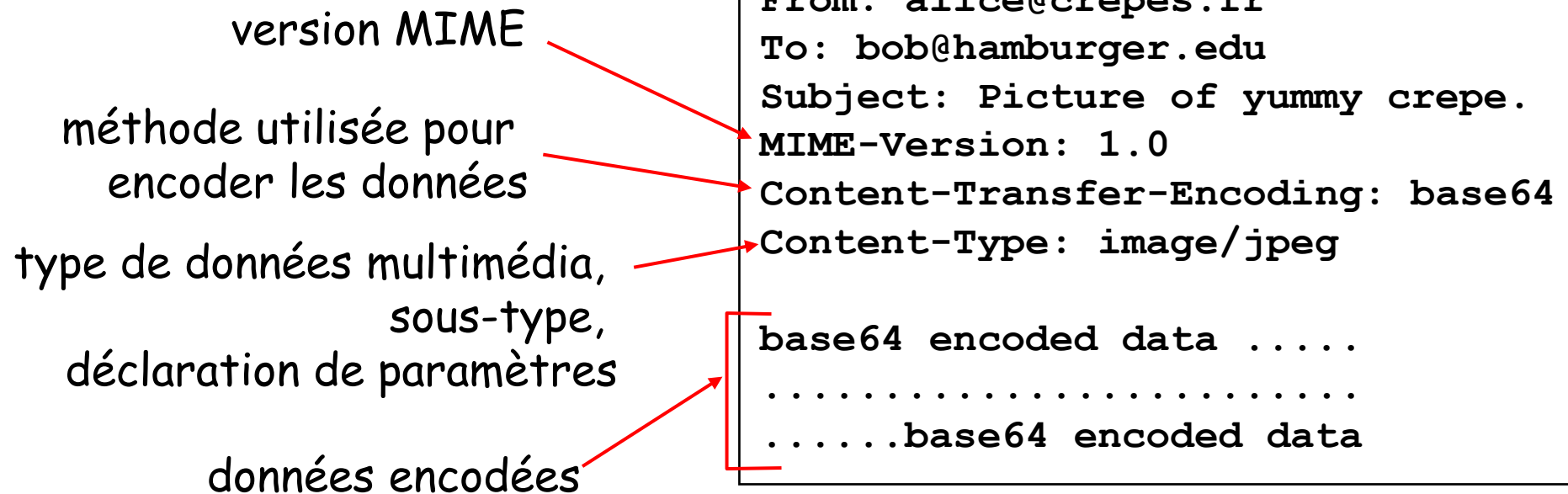
*Différentes des commandes SMTP!*

- corps
  - le "message", est en caractère ASCII seulement

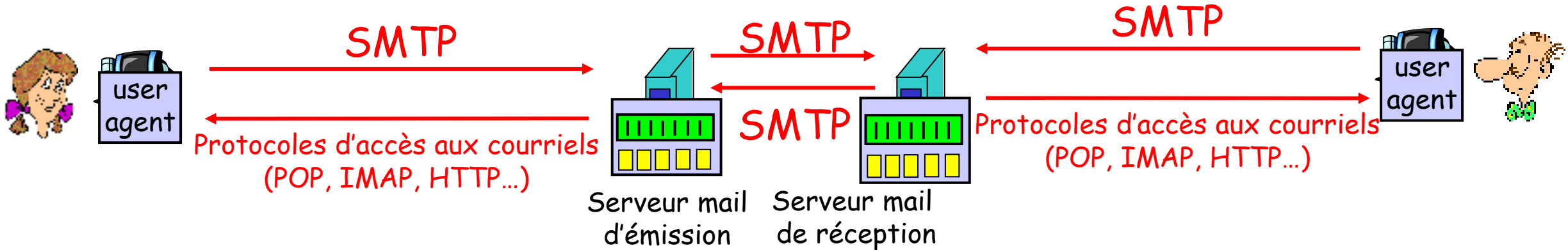


# Format du message: extensions multimédia

- ❑ MIME: extension mail pour les multimédia, RFC 2045, 2056
- ❑ Des lignes additionnelles dans l'entête du message déclarent le type du contenu MIME



# Protocole d'accès Mail



- ❑ SMTP est utilisé seulement pour livrer le courriel au serveur de réception
- ❑ Protocoles d'accès aux courriels: récupérer les courriels à partir du serveur
  - ❖ POP : Post Office Protocol [RFC 1939] : autorisation d'accès et téléchargement des courriels
  - ❖ IMAP : Internet Mail Access Protocol [RFC 1730]: Plus d'options (plus complexe) incluant la manipulation des messages stockés sur le serveur
  - ❖ HTTP : gmail, Hotmail, Yahoo! Mail, etc.



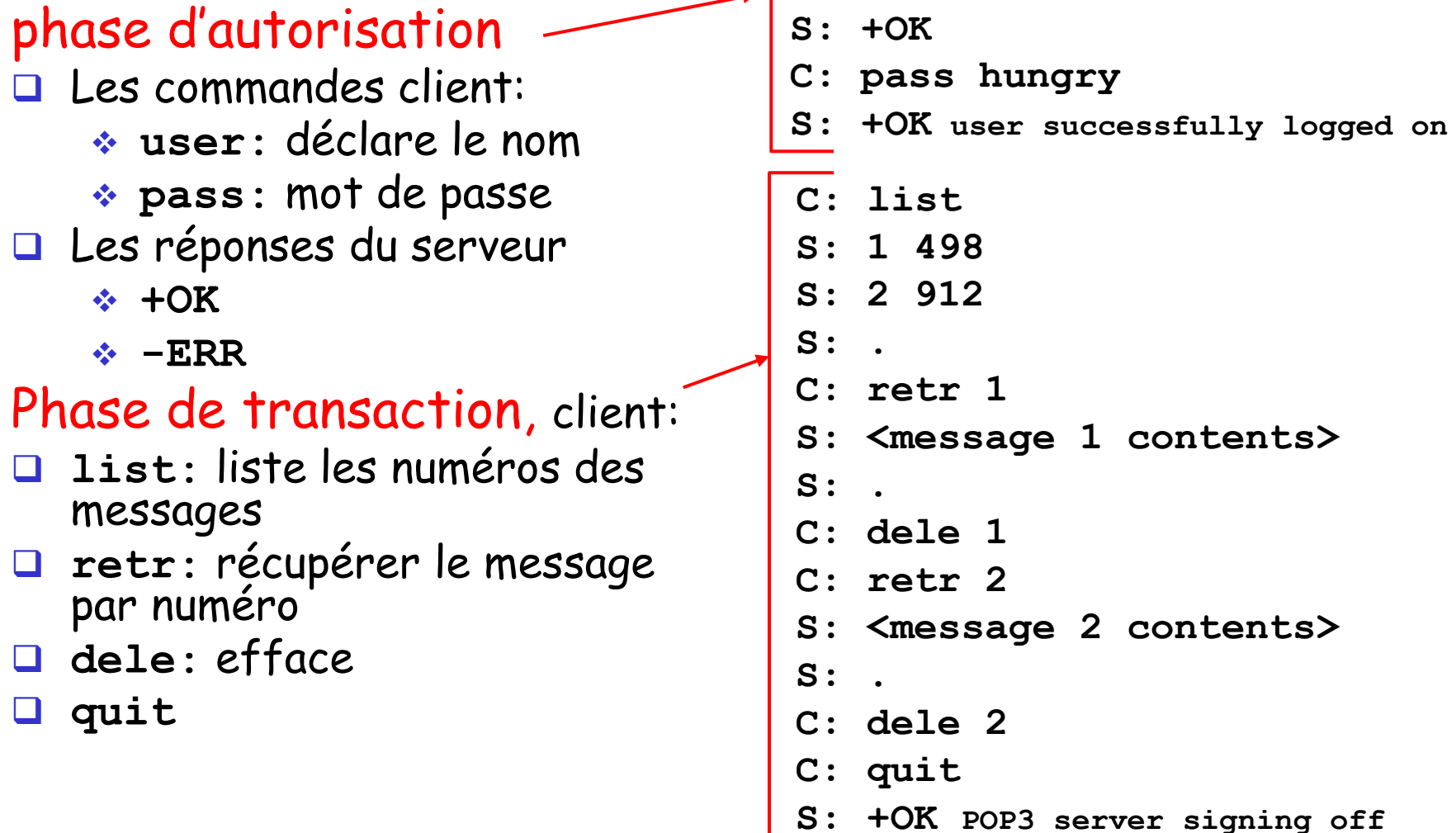
# Le protocole POP3

## phase d'autorisation

- ❑ Les commandes client:
  - ❖ user: déclare le nom
  - ❖ pass: mot de passe
- ❑ Les réponses du serveur
  - ❖ +OK
  - ❖ -ERR

## Phase de transaction, client:

- ❑ list: liste les numéros des messages
- ❑ retr: récupérer le message par numéro
- ❑ dele: efface
- ❑ quit



```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# POP3 et IMAP

## Plus sur POP3

- ❑ L'exemple précédent utilise le mode "télécharge et efface".
  - ❑ Bob ne peut pas relire les courriels s'il change de client
- ❑ "télécharge-et-garde": des copies des courriels sur des clients différents
- ❑ POP3 est sans état à travers les sessions

## IMAP

- ❑ Garde tous les courriels dans la même place: le serveur
- ❑ Permet à l'utilisateur d'organiser les messages en répertoires
- ❑ IMAP garde l'état de l'utilisateur à travers les sessions :
  - ❖ Les noms de répertoires et le mappage entre les IDs des messages et le nom du répertoire

# Chapitre 2: la couche application

1. Principes des apps réseaux
  1. Architectures des apps
  2. Interfaces de connexion (*Socket*)
  3. Exigences des apps en termes de performance et de fiabilité
2. Web et HTTP
3. Protocole de transfert de fichier (FTP)
4. Courrier électronique (SMTP, POP3, IMAP)
5. Serveur de noms DNS

# DNS: Domain Name System

**Humains** : plusieurs identifiants :

- ❖ NAS, nom, passeport

**hôtes Internet et routeurs** :

- ❖ Adresse IP (32 bit) - utilisée pour l'adressage des datagrammes
- ❖ "nom", par ex., www.yahoo.com - utilisé par les humains

**Q:** comment faire la correspondance entre un nom et une adresse IP?

❖ **Système de noms de domaines (Domain Name System) :**

- ❑ Base de données distribuée implémentée en hiérarchie de plusieurs serveurs de noms et qui maintiennent les correspondances : (nom de domaine → adresse IP)

❖ **Protocole Domain Name Service (DNS) :**

- ❑ Les hôtes et routeurs doivent communiquer avec les serveurs de noms à travers le protocole DNS pour trouver la correspondance (nom, adresse IP)
- ❑ Protocole de niveau application qui utilise soit TCP soit UDP au niveau transport

Note : la résolution des noms de domaines est une fonction de base dans l'Internet, mais implémentée comme un protocole de niveau application

# DNS

## Les services DNS

- ❑ traduction du nom d'une machine en une adresse IP
- ❑ dénomination d'hôte
  - ❑ Nom canonique
  - ❑ Alias
- ❑ attribution d'un alias à un serveur de messagerie
- ❑ distribution de charge
  - ❖ Plusieurs instances du même serveur Web : plusieurs adresses IP pour le même nom

## Pourquoi ne pas centraliser le DNS?

- ❑ fragilité d'un site central unique
- ❑ volume de trafic
- ❑ Une base de données centralisée peut être située trop loin par rapport à quelques utilisateurs
- ❑ maintenance

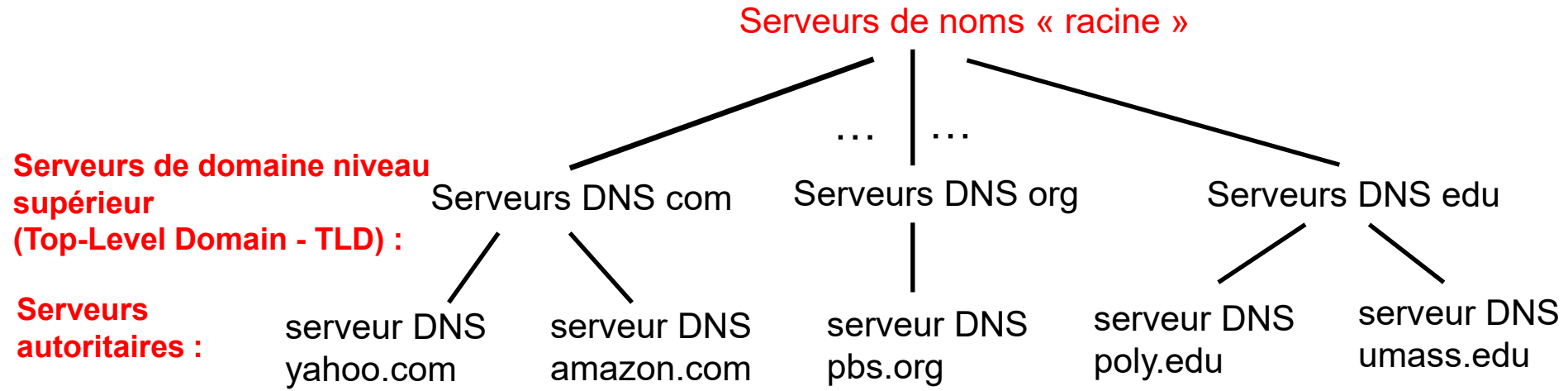
Pas de mise à l'échelle!

Les serveurs de nom de domaine écoutent sur le **port 53**

# Les noms de domaines

- ❑ Noms des hôtes :
  - ❑ `www.yahoo.com`, `mail.yahoo.com`, `help.yahoo.com`
  - ❑ `signets.etsmtl.ca`, `ens.etsmtl.ca`, `profs.etsmtl.ca`, `safirh.etsmtl.ca`
- ❑ Un domaine est un ensemble d'ordinateurs :
  - ❑ `yahoo.com`, `etsmtl.ca`, `google.ca`
- ❑ Il est possible de créer des sous-domaines :
  - ❑ `logti.etsmtl.ca`, `departements.etsmtl.ca`
- ❑ Le domaine de niveau supérieur (Top-Level Domain - TLD)
  - ❑ `.ca`, `.fr`, `.com`, `.org`

# Base de données répartie, hiérarchique



## Un client veut l'adresse IP de [www.amazon.com](http://www.amazon.com) :

1. Le client demande au serveur DNS racine de trouver le serveur DNS .com
2. Le client demande au serveur DNS .com de trouver le serveur DNS de amazon.com
3. Le client demande au serveur DNS de amazon.com de trouver l'adresse IP de [www.amazon.com](http://www.amazon.com)

# TLD et serveurs DNS autoritaires

## ❑ Serveurs de noms autoritaires:

- ❖ Serveur DNS appartenant à l'organisation, donne la correspondance nom-adresse IP pour les serveurs de l'organisation (par ex., Web, mail).
- ❖ Peut être maintenu par une organisation ou par un fournisseur d'accès

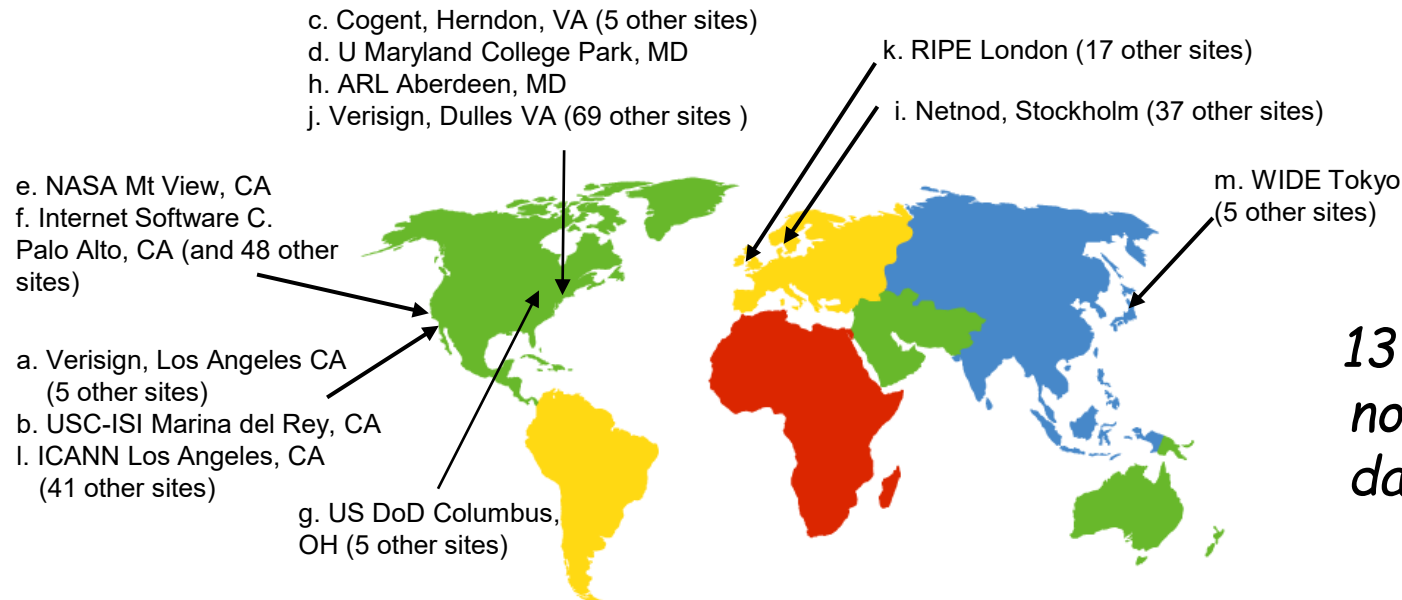
## ❑ Serveurs de domaines de niveau supérieur (Top-Level Domain servers - TLD servers):

- ❖ responsables pour les domaines génériques tels que: com, org, net, edu, ... et tous les domaines de niveau supérieur des pays tels que: ca, uk, fr.
- ❖ Environ 560 domaines TLD, 260 nationaux et 300 génériques



# DNS: serveurs de nom « racine »

- ❑ Contacté par le serveur de nom local qui ne peut pas trouver l'adresse IP qui correspond à un nom
- ❑ Serveur de nom « racine » (*root name server*) :
  - ❖ Retourne l'adresse(s) de serveur(s) TLD responsable d'un domaine (edu, ca, fr...)



*13 serveurs de nom « racine » dans le monde*

# Serveur DNS local

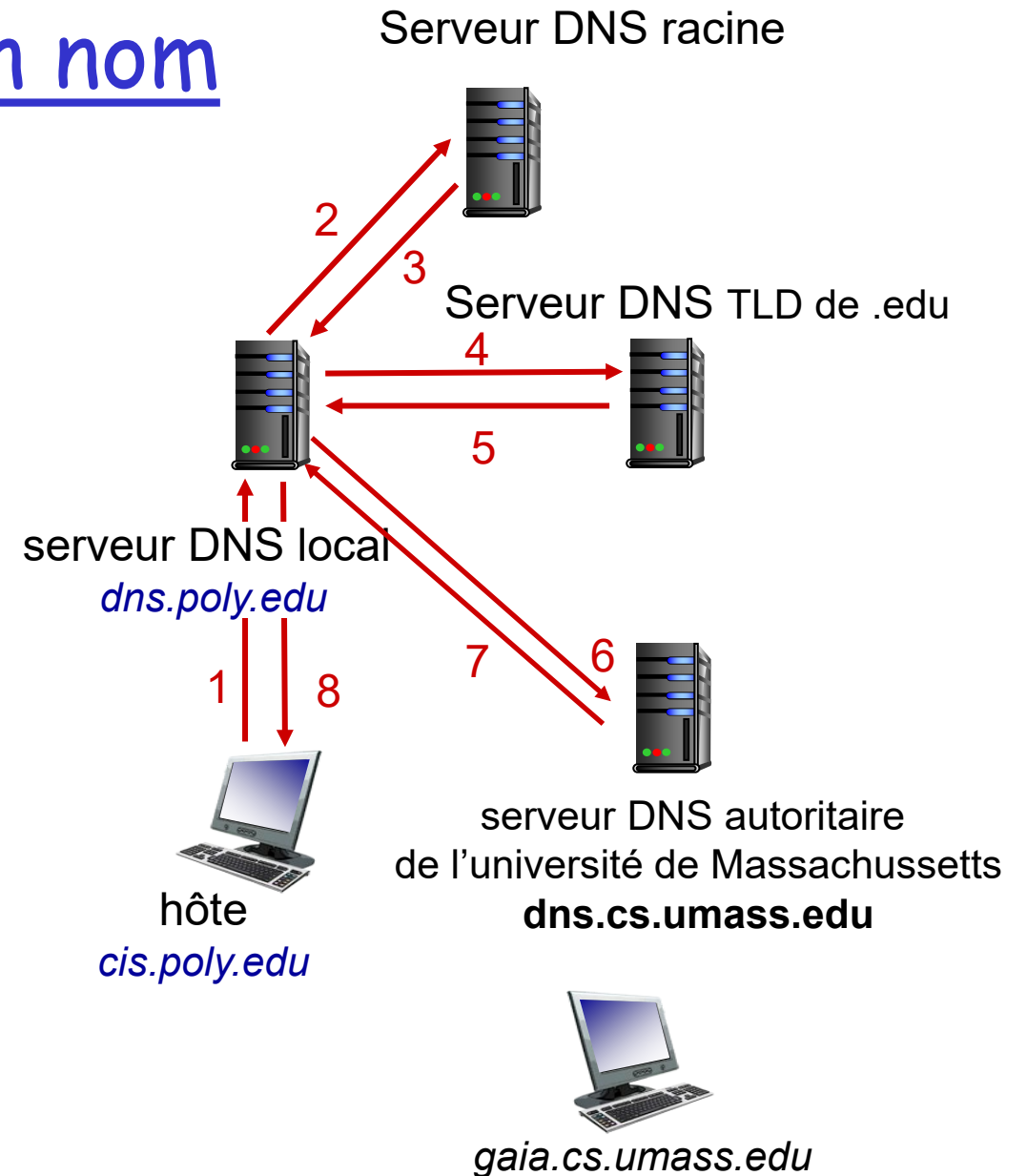
- ❑ N'appartient pas à la hiérarchie
- ❑ Chaque FAI (FAI résidentiel, compagnie, université) en a un.
  - ❖ appelé aussi "serveur de noms par défaut"
- ❑ Quand un hôte envoie une requête DNS, la demande est envoyée à son serveur DNS local
  - ❖ Il possède un cache local contenant les couples (nom, adresse IP) récemment demandés.
  - ❖ Il agit comme proxy, transfère les demandes dans la hiérarchie

# Exemple de résolution d'un nom

- Un hôte `cis.poly.edu` souhaite l'adresse IP de `gaia.cs.umass.edu` (`umass`: university of Massachusetts)

## recherche itérative:

- Le serveur DNS contacté répond avec le nom du serveur DNS à contacter :  
"je ne connais pas ce nom  
mais demande à ce serveur"

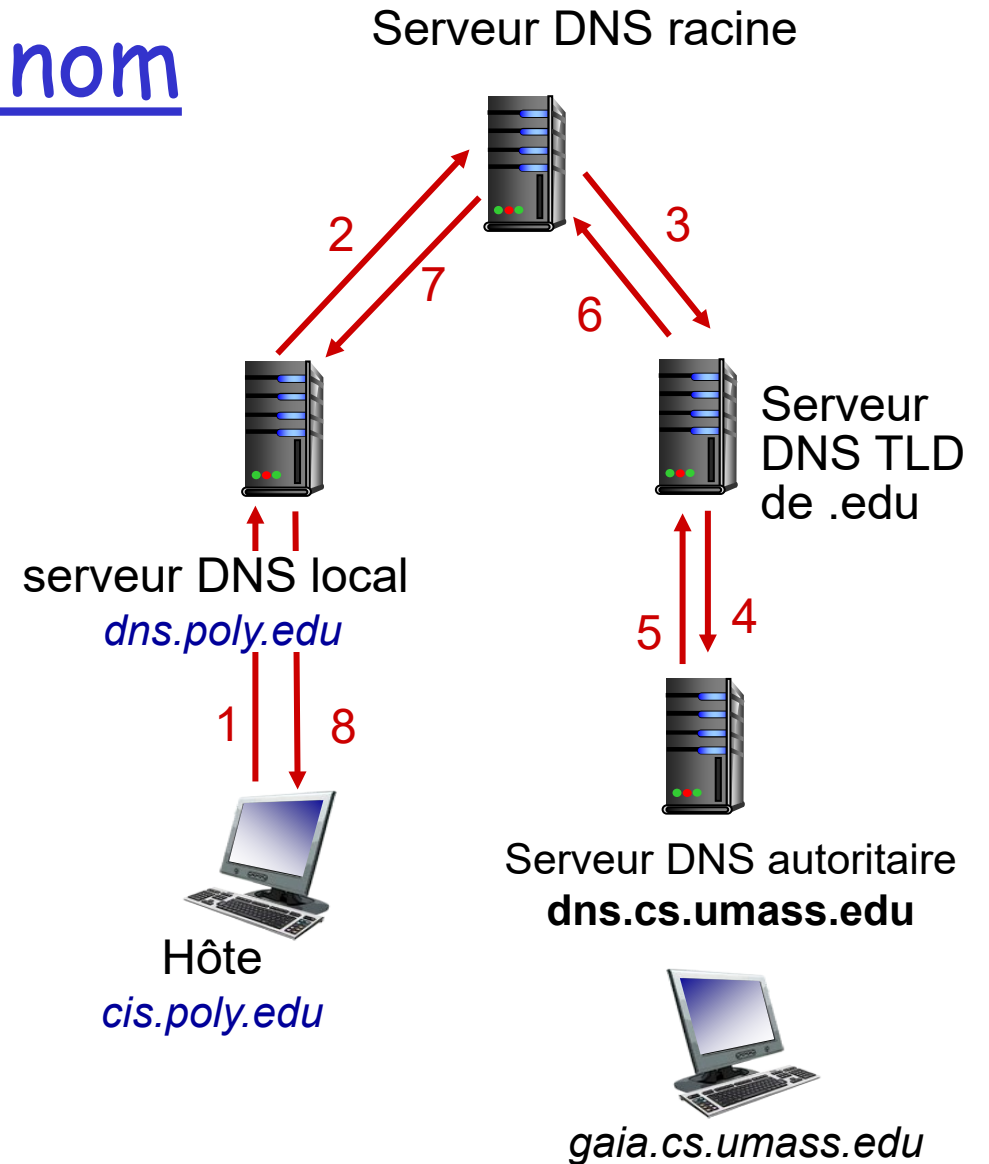


`gaia.cs.umass.edu`

# Exemple de résolution d'un nom

## recherche récursive:

- ❑ Mettre le fardeau sur le serveur DNS contacté
- ❑ grande charge sur les serveurs DNS racine?



# DNS: le cache et les mises à jour

- Lorsqu'un serveur DNS trouve l'adresse IP correspondante à un nom, il la place dans sa mémoire cache
  - ❖ Chaque entrée dans le cache est supprimée (disparaît) après un certain temps (TTL)
  - ❖ Les adresses des serveurs TLD sont typiquement stockées dans la mémoire cache du serveur DNS local
    - Donc le serveur racine n'est pas fréquemment visité
- Les mécanismes de mise à jour et de notification sont décrits dans le RFC 2136

Questions?