Introduction au World Wide Web

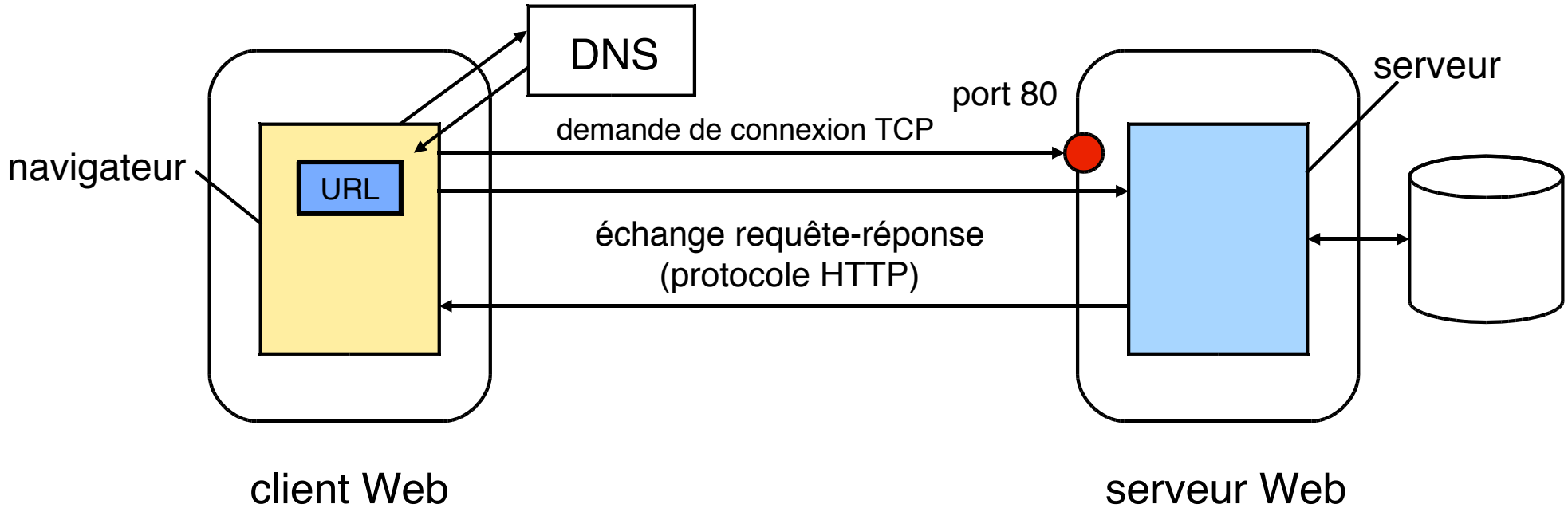
Olivier Dalle
Université de Nice - Sophia Antipolis
http://deptinfo.unice.fr/
D'après le cours original de
Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/~krakowia

World Wide Web : principes et composants

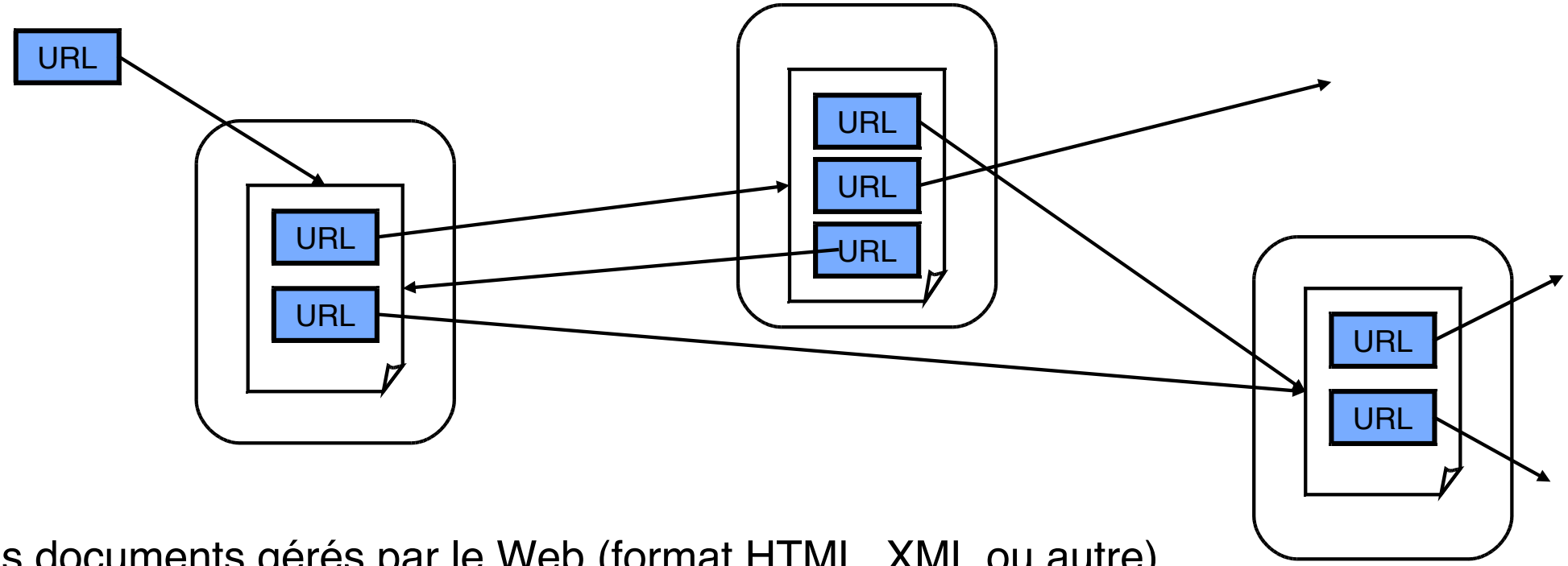
- **Bref historique**
  - Idée de base : ensemble de documents répartis reliés entre eux par des liens hypertexte.
  - Objectif initial (Tim Berners-Lee, CERN, 1989-90) : créer un outil pour le travail en collaboration, sur des données communes, pour une communauté répartie de physiciens
    - en fin 1993, 250 serveurs, 1% du trafic de l'Internet (10 fois plus qu'en début 1993)
  - Le vrai démarrage (1994)
    - les premiers navigateurs : Mosaic (NCSA), puis Netscape
    - les premiers moteurs de recherche : AltaVista, Yahoo!
    - création du World Wide Web Consortium (W3C) <www.w3c.org>
    - en fin 1994, environ 10 000 serveurs
  - Depuis, croissance explosive (l'application la plus utilisée de l'Internet) ~x10^9 pages web (x > 10)
- **Éléments de base du Web**
  - Un espace de noms global pour la désignation des ressources (URL, puis URI)
  - Un protocole (client-serveur) pour le transfert d'information : HTTP
  - Un langage de balisage (*markup*) pour la description de documents hypertextes : HTML
- **Extensions**
  - Langages de script (activation chez le client - *applets*, ou le serveur - *servlets*)
  - Types de données multiples ; descriptions génériques (XML), outils associés
  - Dans l'avenir : Web sémantique

Organisation générale du Web (1)



Le Web utilise le schéma **client-serveur**
Le service demandé est localisé par une URL (cf plus loin)
Le **protocole** d'échange (requête-réponse) est **HTTP** (construit sur TCP)
Le serveur peut servir de nombreux clients simultanément

Organisation générale du Web (2)



Les documents gérés par le Web (format HTML, XML ou autre) utilisent des **liens hypertexte** (désignation d'un autre document) ; c'est cette structure maillée qui a donné son nom à la "toile" (*web*)

Les liens sont matérialisés par des URL (ou des URI, cf plus loin)

Désignation sur le Web

Une ressource est désignée sur le Web par un nom appelé **URI (*Uniform Resource Identifier*).** Les URI peuvent avoir différentes formes.

La forme d'URI la plus répandue, car la plus simple à mettre en œuvre, est l'**URL (*Uniform Resource Locator*)**, qui identifie une ressource par sa localisation et son protocole d'accès.

Exemple : **http://deptinfo.unice.fr/index.php3**

- protocole
- localisation du serveur — correspond à une adresse IP
- fichier — peut être omis (index.*)

Autres protocoles :

file	fichier local
ftp	fichier distant
mailto	mail (SMTP)
news	forums
…	

Une forme d'URI plus abstraite : **URN (*Uniform Resource Name*)**. Exemple :

urn:isbn:0131784560 (désigne un livre, de manière **unique**)

- espace de noms
- nom spécifique

Plus difficile à exploiter : il faudrait un annuaire d'ISBN indiquant une ou plusieurs localisations (URL) où le livre pourrait être trouvé

Contenu statique ou dynamique

Le contenu de la réponse à une requête de lecture (`GET`) peut être créé de manière statique ou dynamique.

Contenu statique : lire le contenu d'un fichier présent sur le site serveur. Exemple : catalogue d'un site marchand, document `pdf`

Contenu dynamique : exécuter un programme qui construit dynamiquement le contenu. Exemples : facturation sur un site marchand, réponse à une recherche sur Google, etc.

Exemple de contenu dynamique : scripts CGI (*Common Gateway Interface*)

Le programme exécutable est dans un répertoire `cgi-bin` sur le site du serveur. Les paramètres sont passés dans l'URI (derrière ?, séparés par &).

Exemple : un programme `adder` qui renvoie la somme de deux nombres, et dont le serveur attend sur le port 4321

```
http://www.unice.fr:4321/cgi-bin/adder?35&67
```

Dans les deux cas (statique ou dynamique), on doit trouver un fichier sur un serveur

Interprétation d'une URL

Le **client** interprète le début de l'URL pour déterminer

- quel protocole utiliser (`http`, `ftp`, etc.)
- la localisation du serveur (en utilisant DNS)
- le port du serveur (se déduit du protocole, ex. 80 : `http`, 21 : `ftp`, etc.) ou peut être indiqué explicitement (ex. :80 après le nom du serveur)

Le **serveur** interprète la fin de l'URL pour déterminer

- si le contenu est statique ou dynamique (pas de règle strictes, mais des indications, par exemple `cgi-bin`, `servlet`)
- le fichier recherché (contenu statique, programme exécutable)
- les paramètres d'un programme exécutable
- il y a des règles par défaut (chercher `index.html`, `Welcome.html`, etc.)

HTTP (*HyperText Transfer Protocol*)

- HTTP : le protocole standard du World Wide Web
  - Protocole client-serveur, construit au-dessus de TCP
  - Utilisation principale : entre navigateur et serveur Web, mais peut être utilisé de manière autonome par toute application
- Principales commandes ("verbes") du protocole
  - 99% GET <URI> : demande au serveur indiqué dans l'URI d'envoyer la page désignée par l'URI. Option : n'envoyer la page que si elle a changé depuis une date spécifiée
  - 1%
    - HEAD <URI> : demande au serveur d'envoyer l'en-tête de la page (contenant des informations diverses : titre, date, etc.)
    - PUT <URI> <page> : envoie une page au serveur spécifié pour la rendre disponible sur ce serveur à l'URI indiquée ; replace le contenu courant de cet URI s'il existe
    - POST <URI> <page> : comme PUT, mais intègre les nouvelles données à celles existant déjà à l'URI (dépend de la nature des données)
    - DELETE <URI> : supprime la page figurant à l'URI indiqué
  - Toutes ces commandes sont soumises à autorisation, en fonction des droits du client demandeur et des protections associées aux ressources sur le serveur
  - La réponse à une commande comporte un code (OK ou type d'erreur) et éventuellement un résultat (contenu de page pour GET, etc.)
  - Convention standard (MIME) pour les données non textuelles

HTTP : versions

HTTP est un protocole du niveau application. Il est construit au-dessus de TCP (protocole de transport en mode connecté). Les clients et serveurs utilisent en général les *sockets* (port serveur 80)

Première version : **HTTP 1.0 (1993-1996)**

Pas de connexion permanente : après un échange (requête-réponse), la connexion TCP est fermée. L'échange suivant doit ouvrir une nouvelle connexion.

Version la plus récente : **HTTP 1.1 (1997-2001)**

Une connexion est créée pour la durée d'une session, et peut servir pour une série de requêtes successives entre un client et un serveur. Néanmoins il est toujours possible de fonctionner en mode HTTP 1.0 (la connexion est fermée à la fin de chaque requête)

HTTP : exemples (1)

Le protocole HTTP définit les formats des requêtes et des réponses. Une requête ou une réponse se compose d'un en-tête (obligatoire) et d'un contenu (facultatif). Les en-têtes sont directement lisibles (ASCII)

Exemple de requête :

```
GET /repertoire/index.html HTTP/1.1
Host: www.hote.fr
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
```

en-tête

+ 2 retour chariot

contenu vide

si ces champs sont vides, une valeur par défaut sera choisie

Expérience : on peut envoyer une requête par connexion directe à l'hôte par

```
telnet hote.organisation.fr 80
GET /index.html HTTP/1.0
```

+ 2 retour chariot

HTTP : exemples (2)

Exemple de réponse à une requête `GET` :

```
HTTP/1.1 200 OK
Connection: close
Date: Fri, 11 Mar 2005 11:04:43 GMT
Server: Apache/1.3.0 (Unix)
Last-modified: Thu, 10 Mar 2005 09:45:22 GMT
Content-length: 8765
Content-type: text/html

.... des données ....
```

en-tête

contenu (le fichier demandé)

`Connection: close` signifie que le serveur va fermer la connexion après ce message

`Content-type: text/html` sert au navigateur pour choisir le programme qui va afficher les données (par ex. `image.gif` appelera le programme d'affichage d'image approprié (*plugin*))

HTTP : exemples (3)

Autre exemple (message d'erreur, requête incorrecte)

```
HTTP/1.1 400 Bad Request
Date: Sat, 12 Mar 2005 14:36:24 GMT
Server: Apache/2.0.52 (Gentoo/Linux) PHP/4.3.10
Content-Length: 330
Connection: close
Content-Type: text/html; charset=iso-8859-1

.... des données ....
```

en-tête

contenu (le message d'erreur en HTML, qui sera affiché par le navigateur)

Principaux codes renvoyés dans la réponse

200	OK, requête sans erreur
301	le fichier a changé d'emplacement
400	requête incorrecte (non comprise par le serveur)
403	opération interdite (protection)
404	fichier pas trouvé

HTTP : exemples (4)

Une requête POST

```
POST /repertoire/fichier HTTP/1.0
Content-Length: 330

.... des données ....
```

en-tête

contenu (le fichier à mettre à jour ou à inclure)

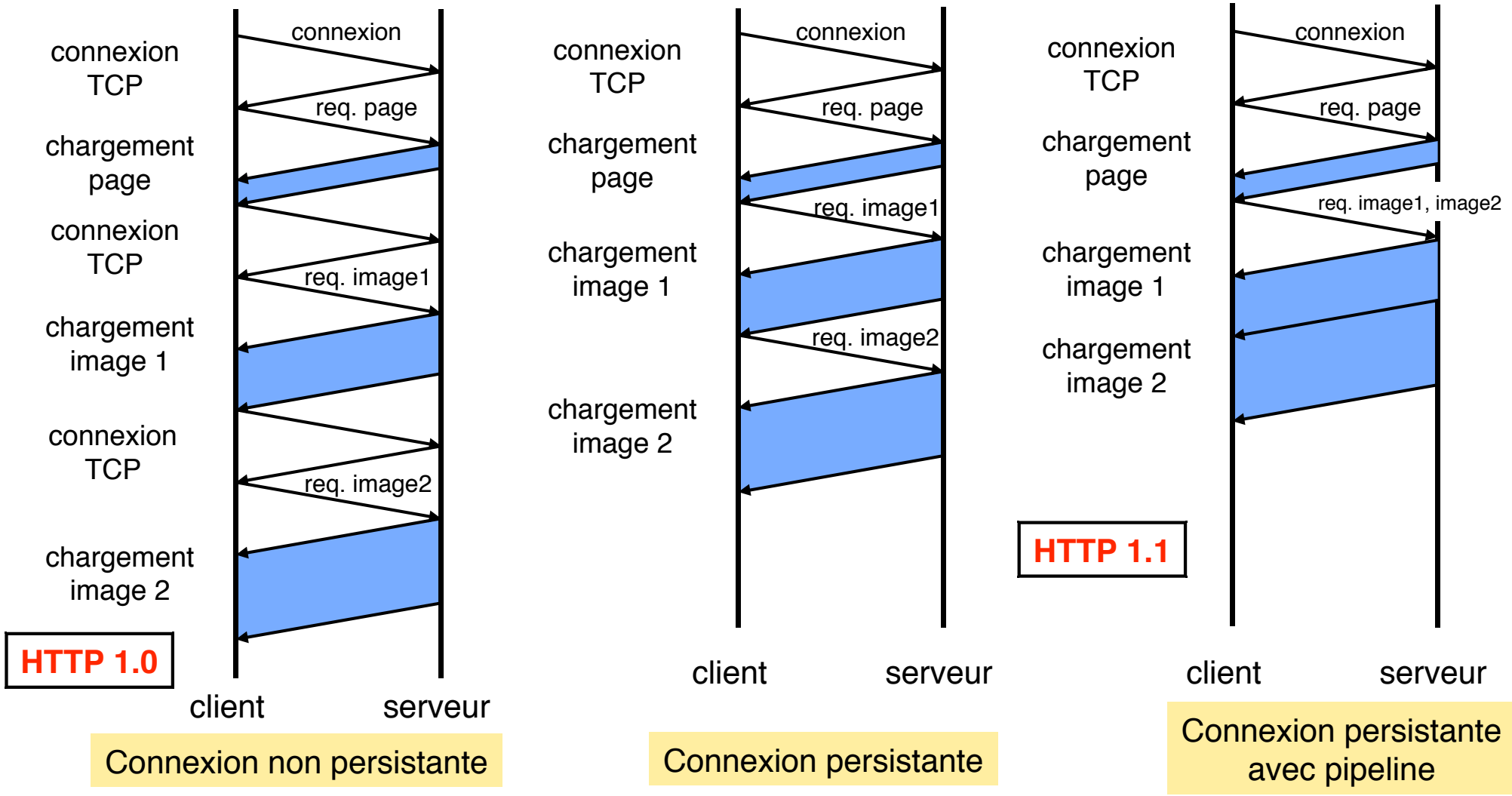
La méthode POST peut servir :

- à modifier un fichier existant (par exemple ajouter un message dans un fichier de news)
- à inclure un nouveau fichier dans un répertoire
- à exécuter un script en lui passant comme paramètres le contenu de la requête (noter la différence avec un script activé par GET où les paramètres sont passés dans l'URL)

Comme elle peut modifier les données du serveur, la méthode POST est généralement soumise à autorisation

Performances du Web

Exemple simple : 1 page HTML avec 2 images incluses (.gif ou .jpg)



Connexion non persistante

Connexion persistante

Connexion persistante avec pipeline

HTML (*HyperText Markup Language*)

- HTML est un langage de “balisage” (*markup*)
  - Un tel langage comporte des marques (balises) insérées dans le texte et destinées à donner des indications de formatage (présentation, interprétation du texte). Exemples plus loin
  - Un langage de balisage très général, utilisé dans l’édition de documents, est SGML (*Standard Generalized Markup Language*) ; HTML en est inspiré.
  - Intérêt du balisage : permet de séparer le contenu de la présentation ou de l’interprétation, et donc permettre des interprétations différentes selon (par exemple) les capacités d’affichage d’une station de travail
  - HTML est en évolution constante (version 4.0) - normalisé par le W3C <www.w3c.org>
- Comment sont produits les documents HTML ?
  - “À la main”. Pas recommandé, il est préférable d’utiliser un des outils qui suivent
  - Par un éditeur de documents (pour l’écriture de pages Web)
    - directement (frappe du texte, insertion d’images, etc.)
    - par traduction depuis un autre format de document (LaTeX, Word, ou autre)
  - Par un générateur spécialisé, à partir (par exemple) du résultat d’une requête sur une base de données. Chaque application peut construire son générateur.

HTML - quelques exemples (1)

Principe du balisage

Les balises vont en général par paires, encadrant un texte à interpréter
balise début : <xxx paramètres éventuels> - balise fin : </xxx>

Structure d'un document HTML (indicatif)

<HTML>
<HEAD> en-tête </HEAD> -- contient le titre, la date, d'autres méta-informations
<BODY> corps </BODY> -- contient le document proprement dit
</HTML>

Quelques balises de présentation (exemples)

Présentation de caractères

- <B> texte </B> : caractères gras (*bold*) ; <I> texte </I> : caractères italiques
- Caractères accentués. Exemples : é = é à = à Ê = Ê etc.

Titres

- <H1> texte du titre </H1> : titre de 1-er niveau (idem pour H2, H3, …)

Paragraphe

- <P> texte </P> : paragraphe ;
 retour à la ligne
- <HR> coupure du texte (trait horizontal)

En fait, l'interprétation précise des balises de présentation peut être définie séparément (feuilles de style). En modifiant la feuille de style, on modifie la présentation sans changer le document

HTML - quelques exemples (2)

- Autres informations de présentation
  - Tables (nombreux attributs possibles : disposition relative des cases, couleur du fond, épaisseur des traits, etc.)
  - Listes (numérotées ou non)
- Inclusion d'images
  - <IMG SRC = "le fichier ou l'URL contenant l'image" – autres paramètres (échelle, alignement par rapport au texte, affichage de texte alternatif si l'image ne peut être affichée, etc.) >
- Liens hypertexte
  - <A HREF="l'URI associée au lien" NAME="le nom" autres paramètres (affichage dans une fenêtre autonome, etc.) > le texte ou l'image qui constitue l'hyperlien </A>
  - Ce lien peut être affiché de manière particulière par le navigateur (par exemple souligné en bleu)
  - Un "clic" de souris sur ce lien est interprété par le navigateur comme : demander le chargement (GET) du document désigné par l'URI du paramètre HREF

HTML - quelques exemples (3)

Un premier exemple d'interaction

Jusque là ont été décrites des caractéristiques uniquement liées à l'affichage

On souhaite aussi permettre l'interaction entre client et serveur

Exemple : remplir un formulaire simple

le texte HTML

```
<HTML> <HEAD> <TITLE> Inscription </TITLE></HEAD>
<BODY>
<H1>Inscription pour l'excursion</H1>
<FORM ACTION = "http://sejourvacances.com/cgi-bin/choix" METHOD = POST>
  Nom <INPUT NAME ="client" SIZE = 20><BR>
  Choisissez la date et cliquez sur OK<BR>
  25 juillet <INPUT NAME="date" TYPE=RADIO VALUE="2507"
  3 ao&ucirc;t  <INPUT NAME="date" TYPE=RADIO VALUE="0308" <BR>
  <INPUT TYPE=SUBMIT VALUE = "OK">
</FORM></BODY></HTML>
```

ce qui est affiché

Inscription pour l'excursion

Nom

Choisissez la date et cliquez sur OK

25 juillet ○ **3 août** ○

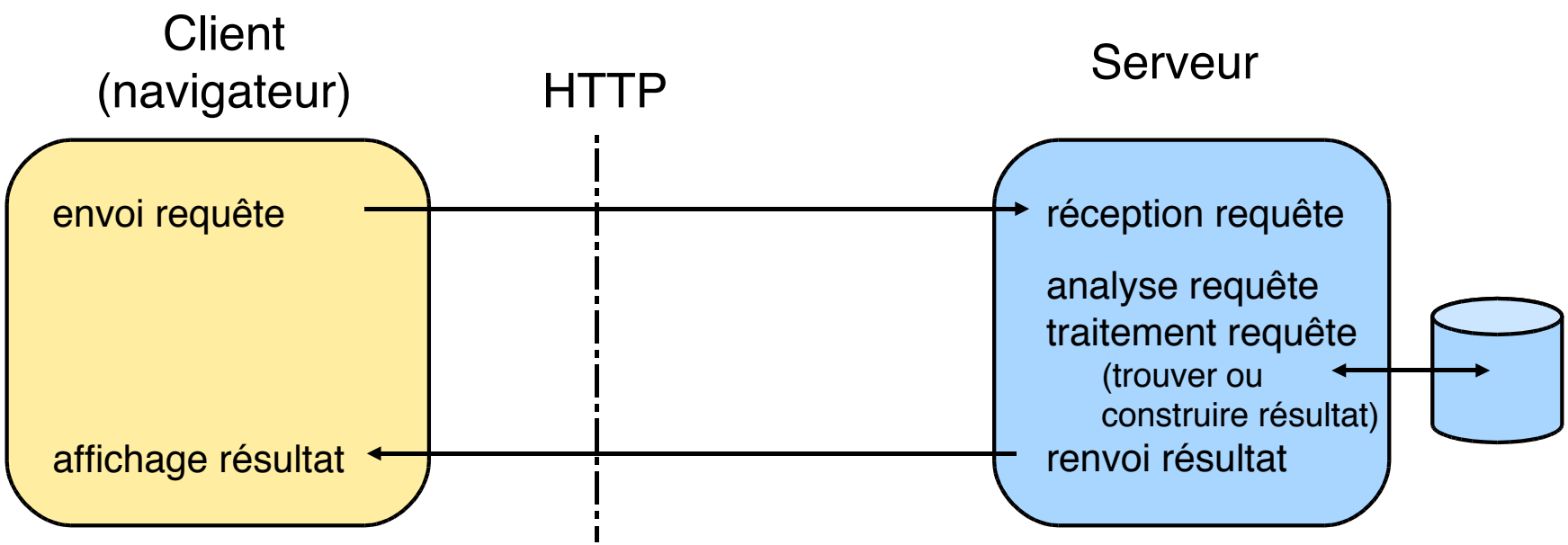
OK

ce qui est envoyé (par exemple)

client=Dupont&date=2507

c'est le programme (script) indiqué dans le paramètre ACTION qui traitera cette entrée

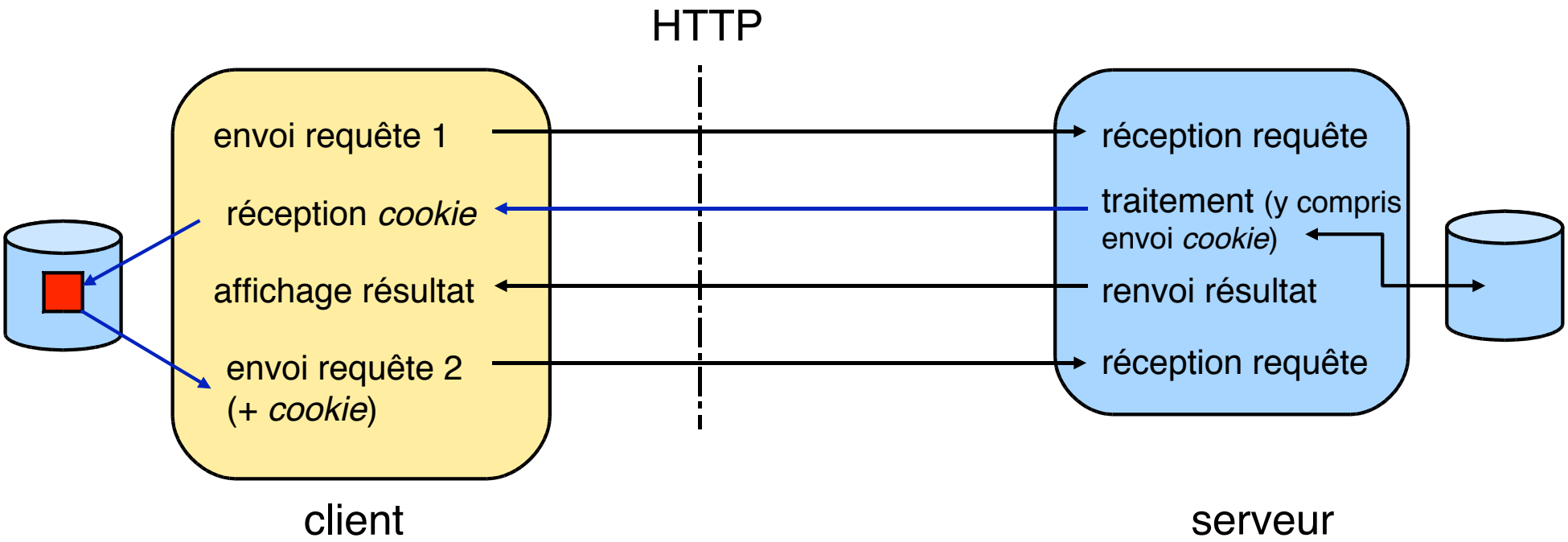
Échange sur le Web : schéma de base



Dans ce schéma simple, le serveur est “sans état” (le serveur ne conserve pas d’informations relatives au client) : les requêtes successives entre un client et un serveur sont indépendantes entre elles.

Schéma d'un échange sur le Web : extensions (1)

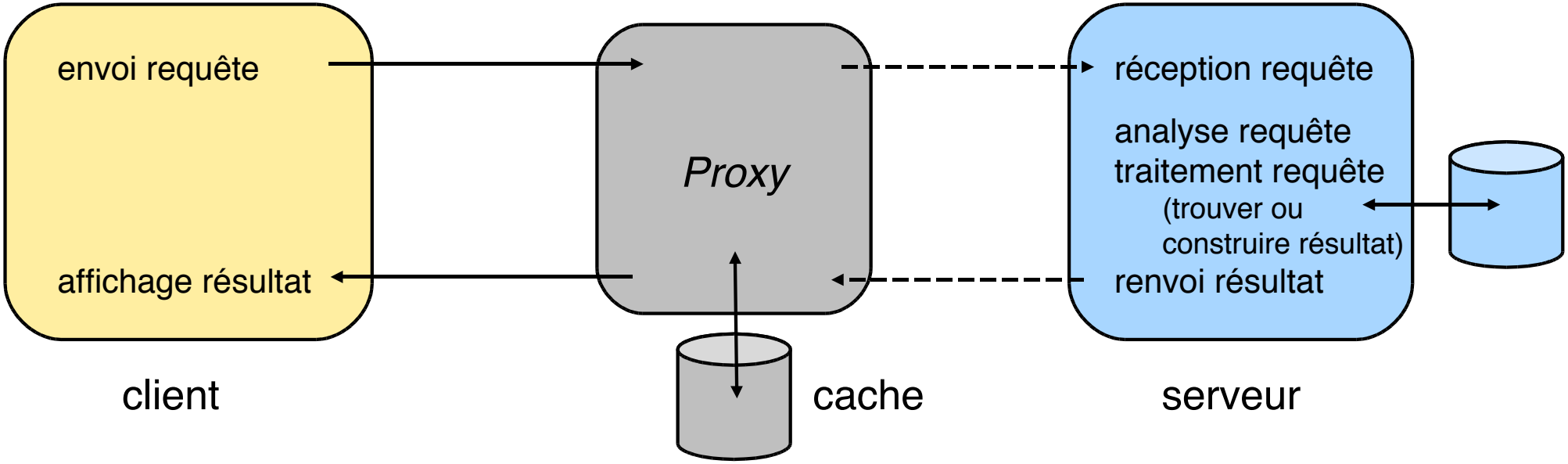
Le mécanisme des *cookies* permet au serveur de maintenir un état en le stockant chez le client. Un *cookie* fabriqué lors du traitement d'une requête est renvoyé lors de requêtes suivantes vers le même serveur



Le client peut restreindre ou interdire l'usage des *cookies*

Schéma d'un échange sur le Web : extensions (2)

Les échanges mettent le plus souvent en jeu un organe intermédiaire : le ***proxy*** (mandataire)



Le *proxy* joue le rôle de serveur pour le client web et de client pour le serveur web (interposition "transparente")

Ses rôles majeurs sont d'améliorer

- l'**efficacité** (conservation des résultats dans un **cache**)
- la **sécurité** (contrôle de droits d'accès)

Un *proxy* est en général commun à de nombreux clients - cf plus loin

Schéma d'un serveur Web

Étapes du traitement d'une requête (HTTP)

1. Lire et analyser le message HTTP ; extraire verbe de la requête (par ex. : `GET`) et l'URI de la ressource demandée (par ex. `./index.html`)
2. Déterminer le nom du fichier en utilisant celui du répertoire de base (par ex. `/www/index.html`, ou `/www/cgi-bin/prog`)
3. Déterminer si la requête est autorisée (par ex. en examinant les droits d'accès au fichier trouvé ci-dessus, ou en demandant un mot de passe)
4. Engendrer la réponse (avec en-tête HTTP). Ce peut être simplement le fichier trouvé ci-dessus (contenu statique), ou le résultat de l'exécution d'un programme (contenu dynamique), ou un message d'erreur
5. Renvoyer la réponse au client

Le serveur est organisé en veilleur-exécutant (pour servir de nombreux clients en parallèle, souvent sur une "grappe" de machines

Exemple : un serveur très populaire en *open source*, Apache (`http://www.apache.org`)

Proxies et caches pour le Web

Un *proxy* web est interposé entre le client et le serveur. Il a différents rôles.

- **Gestion de caches** (fonction principale), pour pouvoir récupérer plus vite des informations encore valides (et en faire profiter d'autres clients). Détails plus loin.
- **Sécurité et protection**
  - filtrage de certaines requêtes, authentification de clients
  - anonymat (supprimer l'identification du client par le serveur)
- **Adaptations diverses**
  - transformation de protocoles (entre HTTP 1.0 et 1.1)
  - traduction de requêtes et / ou de réponses dans plusieurs langues
- **Médiation** vers d'autres services (non HTTP)
  - service de fichiers (`ftp`), de messagerie (`snmp`), etc.

Dans tous les cas, le *proxy* est "transparent" pour le client et le serveur (le serveur le voit comme un client et le client comme un serveur)

Proxy avec fonction de cache pour le Web

- Fonctions générales d'un cache (rappel)

  Introduire un niveau intermédiaire, d'accès rapide, entre le lieu de stockage d'une information et celui de son utilisation.

  Objectifs :

  - réduire le temps moyen d'accès, en conservant les informations les plus utilisées
  - réduire le trafic entre les niveaux de stockage (pour le web : trafic sur l'Internet)

  Hypothèse de travail (souvent vérifiée) : localité d'accès (réutilisation des informations)

- Le web se prête bien à l'usage de caches

  Les informations changent relativement peu souvent

  On peut travailler sur des regroupements de demandes (à plusieurs niveaux)

  - cache individuel sur disque
  - cache local pour un département, une entreprise, etc.
  - cache régional pour un ensemble de réseaux

- Problèmes à résoudre

  Politique de gestion du cache (quoi garder, quoi éliminer, etc)

Un cache populaire (*open source*) : Squid. Voir `http://squid.nlanr.net/Squid/`

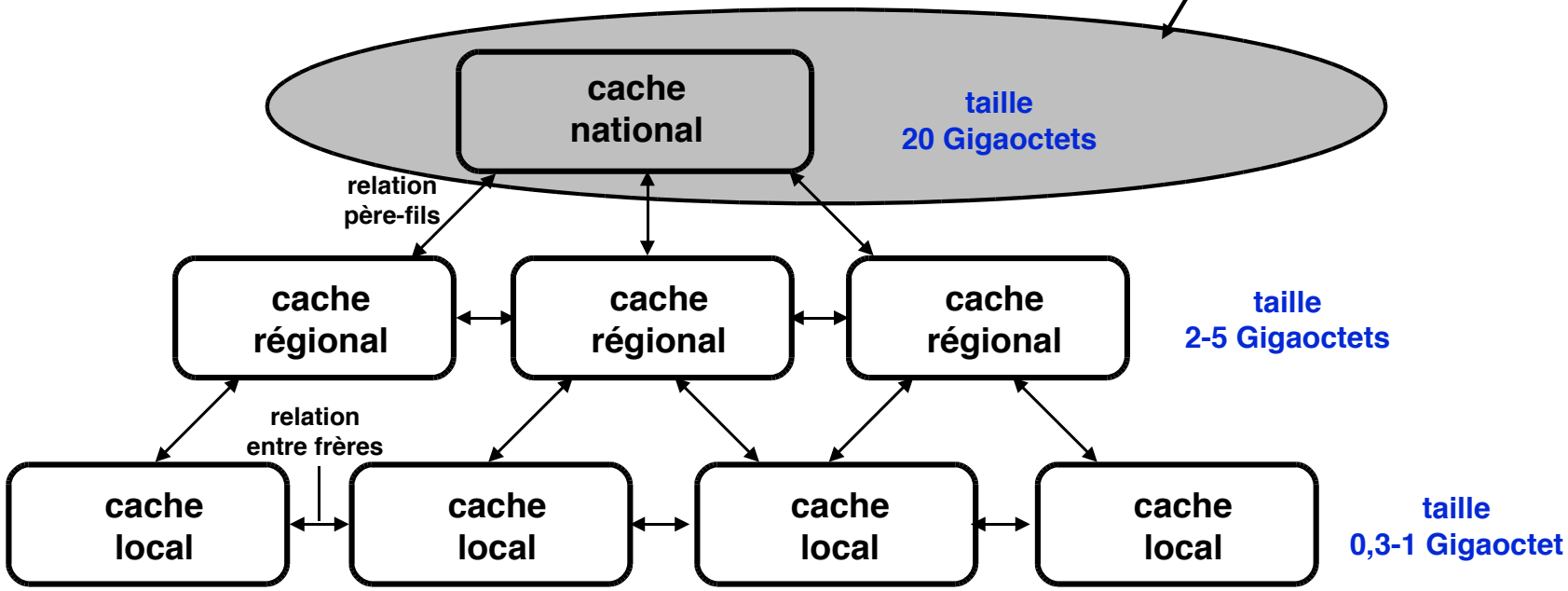
Problèmes des caches web

- Politique de remplacement (quels documents éliminer quand on a besoin de place)
  - FIFO (dans l'ordre des arrivées) : simple à réaliser, peu intéressant
  - SIZE : éliminer le document le plus gros (pour gagner de la place) : gain à court terme, mais risque de perte si le document éliminé était très demandé
  - LRU (*Least Recently Used*) : fondé sur l'hypothèse de localité, souvent utilisé
- Cohérence (comment garantir que les documents du cache sont à jour)
  - Invalidation : le serveur prévient le cache quand l'original est modifié
    - idéal, mais grosse charge de gestion pour le serveur (doit garder trace des copies)
  - TTL (*Time To Live*) : durée de vie limitée ; élimination ou rappel serveur à l'expiration
  - Autre solution : durée de vie proportionnelle à l'âge du document
- Coopération entre caches
  - Hiérarchie : tout cache a un "parent", auquel il transmet la requête s'il ne peut la résoudre
    - Le parent fait de même (ou contacte le serveur s'il n'a pas de parent), puis répond au fils
  - Entre égaux : un cache transmet la requête aux autres caches "frères" et au serveur ; il prend la première réponse qui arrive
  - Le mode de coopération entre deux caches n'est pas fixé a priori et peut dépendre de la nature des requêtes

Exemple de hiérarchie de caches Web (projet ancien)

Hiérarchie de caches du réseau Renater

Le cache national a été arrêté en 2000 car rapport coût-efficacité insuffisant



Rendement espéré : local 25%, régional 20%, national 15%

Voir : `http://www.serveurs-nationaux.jussieu.fr/cache/`

Résumé de la séance 10

- Introduction au World Wide Web
  - Principes, fonctions de base
  - Désignation
- Protocole HTTP
- Format HTML
- Organisation d'un serveur Web
  - Contenu statique
  - Contenu dynamique
- *Proxies* et caches pour le Web