Introduction au World Wide Web

Renaud Lachaize
Université Grenoble Alpes

renaud.lachaize @ imag.fr

Avril 2021

Ce cours est basé sur les transparents de Sacha Krakowiak

World Wide Web : principes et composants (1/2)

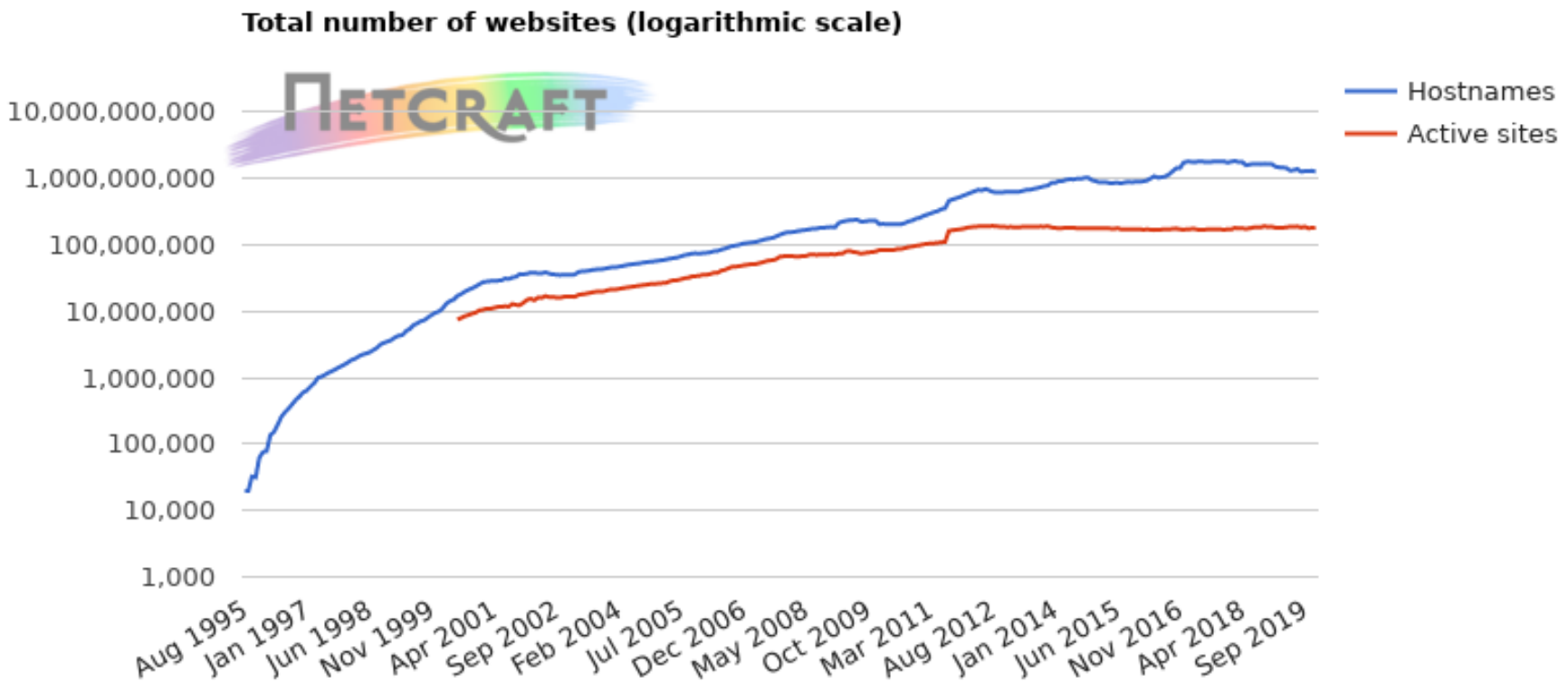
Bref historique

- 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
  - 1991 : ouverture du système à tous sur l'Internet
  - en fin 1993, 250 serveurs, 1% du trafic de l'Internet (10 fois plus qu'en début 1993)
- Le vrai démarrage (1993-1994)
  - les premiers navigateurs : Mosaic (NCSA), puis Netscape
  - les premiers moteurs de recherche : AltaVista, Yahoo!
  - création du World Wide Web Consortium (W3C) http://www.w3c.org
  - Première conférence internationale sur le Web
  - fin 1994, environ 10 000 serveurs

- Depuis, croissance explosive (l'application la plus utilisée de l'Internet, DNS excepté)

Croissance du nombre de sites (serveurs) Web



Sources :

- https://news.netcraft.com/archives/2019/12/10/december-2019-web-server-survey.html
- https://www.netcraft.com/active-sites/
- https://www.internetlivestats.com/total-number-of-websites/

Autres statistiques

- Évolution du nombre de pages Web (et de leur taille)
  - https://askwonder.com/research/web-pages-internet-presently-5l73c5yfg
  - https://www.worldwidewebsize.com
  - https://www.yottaa.com/a-brief-history-of-web-page-size/
  - http://www.websiteoptimization.com/speed/tweak/average-web-page/
- Nombre d'utilisateurs (pour Internet … mais ici assimilable au Web car c'est l'application la plus utilisée)
  - https://www.internetlivestats.com/internet-users/
  - https://www.internetlivestats.com/internet-users-by-country/
  - https://www.internetworldstats.com/stats.htm

World Wide Web : principes et composants (2/2)

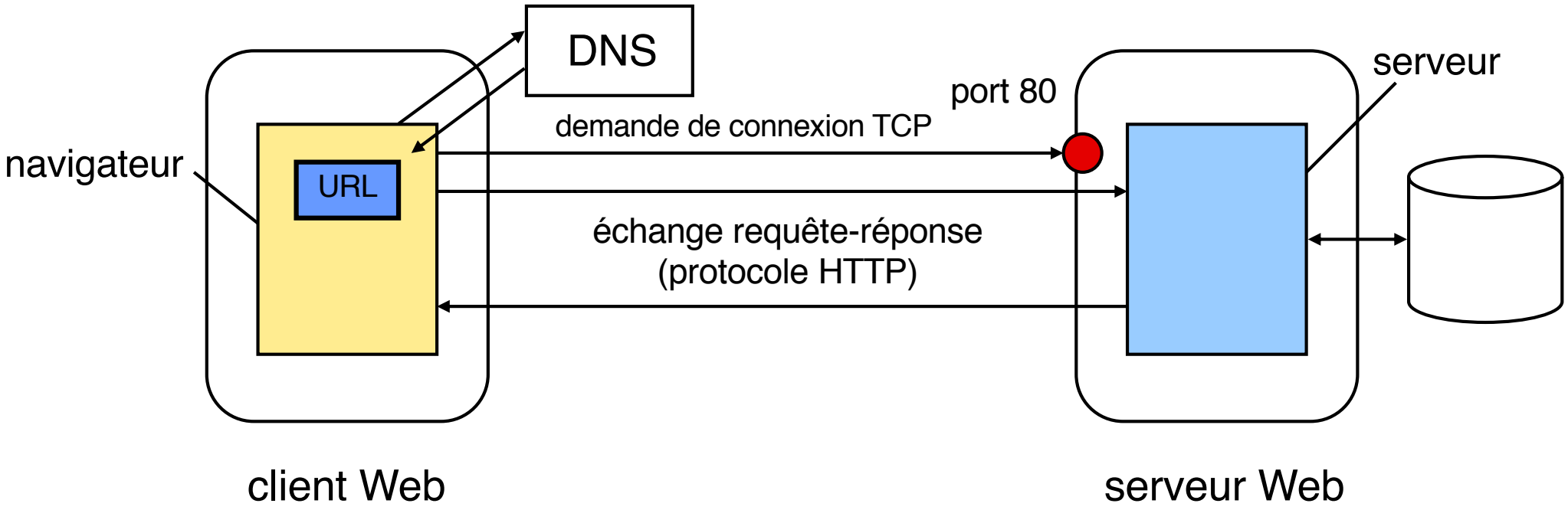
Éléments de base du Web

- Un espace de noms global pour la désignation des ressources (URL, puis URI)
- Un langage de balisage (*markup*) pour la description de documents hypertextes : HTML
- Un protocole (client-serveur) pour le transfert d'information : HTTP

Extensions

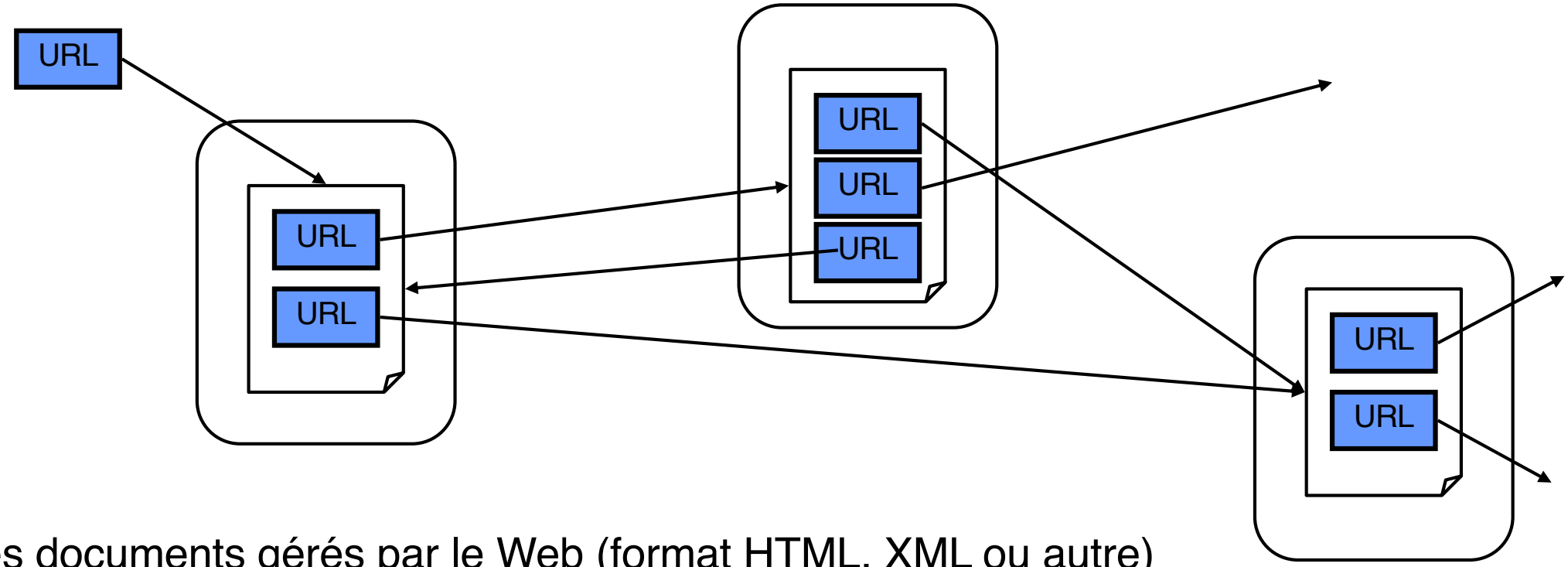
- Langages de script (activation chez le client - *applets*, ou le serveur - *servlets*)
- Types de données multiples ; descriptions génériques (XML), outils associés
- Interactions entre machines : services Web
- Dans l'avenir : Web sémantique

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)
Un serveur *concurrent* 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*)

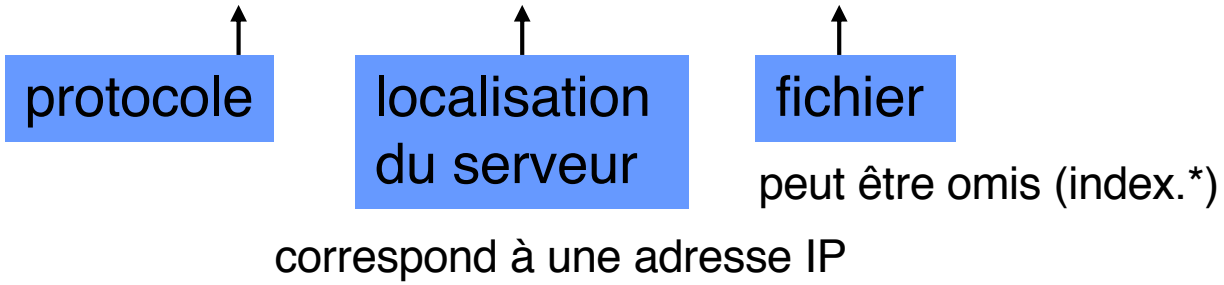
Les liens sont matérialisés par des URL (ou des URI, cf. plus loin)

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://boole.imag.fr/index.php**

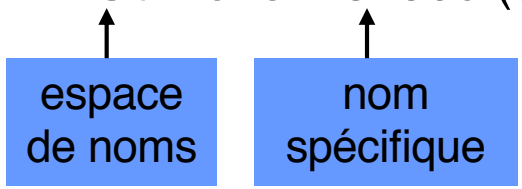


Autres protocoles :

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**)



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

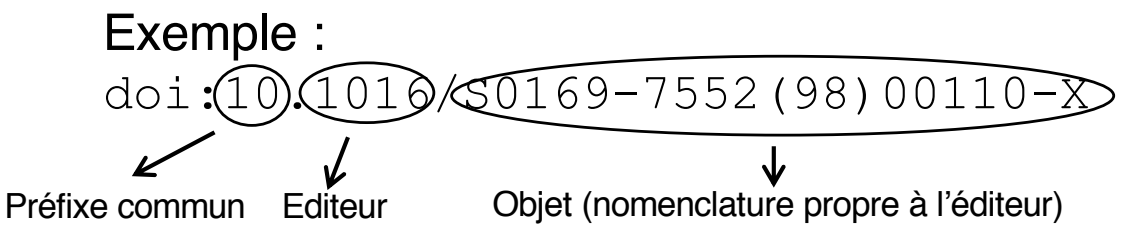
Désignation sur le Web (2)

Un autre exemple d'URI : **DOI (*Digital Object Identifier*).**
Une chaîne de caractères qui correspond à l'identifiant numérique d'un objet (tel qu'un document électronique ou physique ... mais bien d'autres types d'objets sont possibles).
Des métadonnées sont associées à l'identifiant ; elles incluent notamment la localisation de l'objet (URL). Ces métadonnées peuvent évoluer au cours du temps (contrairement au DOI, qui est pérenne).

Un système d'annuaire(s) de DOI permet d'obtenir les métadonnées qui correspondent à un identifiant donné. (voir par exemple l'interface Web: http://dx.doi.org/)

Format : une chaîne de caractères (unicode) insensible à la casse en deux parties (préfixe puis **/** puis suffixe) et on rajoute `doi:` devant
(ou bien `http://dx.doi.org/` pour embarquer le lien dans une URL)

Exemple :



Le premier article scientifique décrivant le moteur de recherche Google (1998)

Contenu Web (1/2)

Un serveur retourne du contenu à un client. On parle aussi d'*objets* ou de *ressources*.

Un objet peut correspondre à une page HTML, un fichier texte, un fichier multimédia (image, fichier audio/vidéo), un script/programme exécutable, etc.

Chaque objet est associé à un identifiant (URL) distinct.

Chaque requête/réponse HTTP est associée à un objet distinct.

Une page Web est composée d'un fichier HTML ainsi que des différents objets qu'il référence.

Contenu Web (2/2)

Un *objet* correspond à une séquence d'octets associé à un type (on parle de type MIME : *Multipurpose Internet Mail Extensions*)

Exemples :

text/html	document HTML
text/plain	document texte non structuré
application/pdf	document PDF
application/javascript	programme javascript
application/x-shockwave-flash	animation/programme Adobe Flash
application/octet-stream	flux de données arbitraire
application/vnd.(...)	formats spécifiques à certains éditeurs
image/gif	image encodée au format GIF
image/jpeg	image encodée au format JPEG
video/mp4	vidéo encodée au format MPEG4

Contenu statique ou dynamique (1/2)

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.

Contenu dynamique : exécuter un programme qui construit le contenu à la volée. Exemples : facturation sur un site marchand, réponse à une requête de recherche.

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

Contenu statique ou dynamique (2/2)

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://mandelbrot2.e.ujf-grenoble.fr:4321/cgi-bin/adder?35&67
```

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 (liées au chemin ou à l'extension du fichier)
- 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 du protocole
  - **GET** <URI> : demande au serveur indiqué dans l'URI d'envoyer la ressource (statique ou dynamique) désignée par l'URI. Option : n'envoyer la page que si elle a changé depuis une date spécifiée
  - **HEAD** <URI> : demande au serveur d'envoyer l'en-tête de la ressource (contenant des informations diverses : titre, date, etc.)
  - **PUT** <URI> <contenu> : envoie une ressource 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> <contenu> : comme PUT, mais intègre les nouvelles données à celles existant déjà (dépend de la nature des données). Permet aussi de demander du contenu dynamique – voir détails plus loin
  - **DELETE** <URI> : supprime la ressource 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 en-tête et éventuellement un corps

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 des *sockets* (port serveur 80 par défaut)

Première version : **HTTP 1.0 (spécification 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 **HTTP 1.1 (spécification 1997-2001)** – La plus utilisée jusqu'au milieu des années 2010

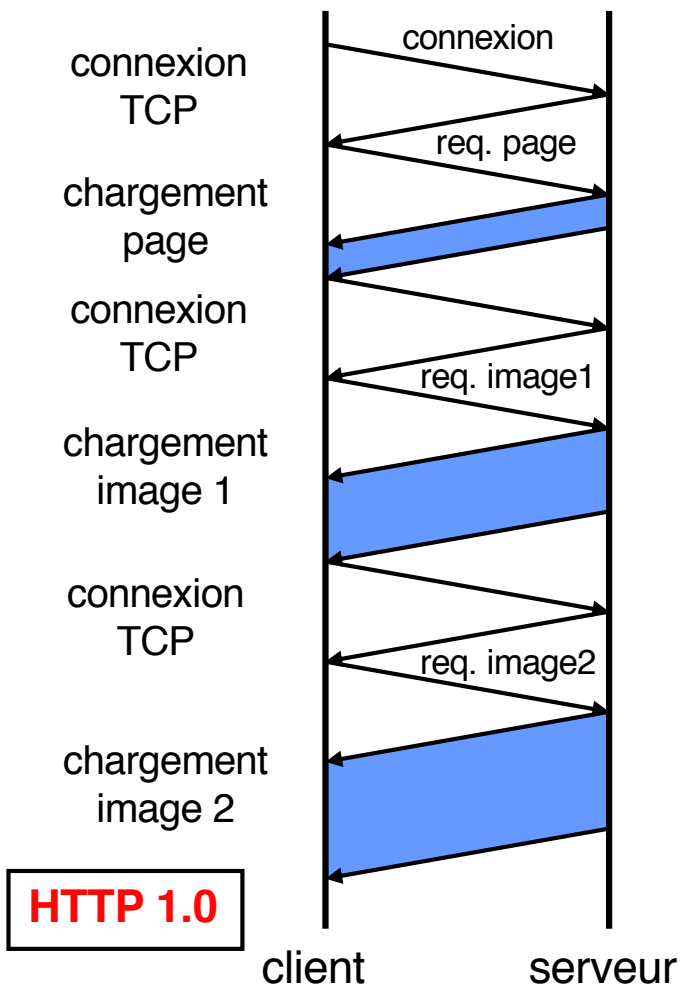
Une connexion est créée pour la durée d'une session, et peut servir pour une série de requêtes successives entre un client et un serveur.
Néanmoins il est toujours possible de fonctionner en mode HTTP 1.0 (la connexion est fermée à la fin de chaque requête)

Et aussi :

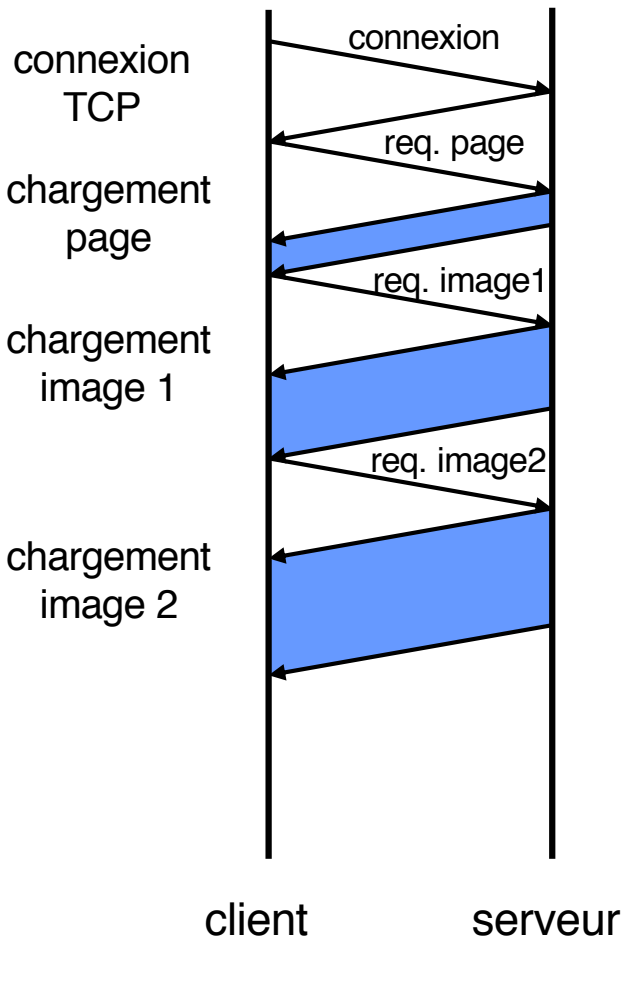
- Une nouvelle version (HTTP/2, anciennement nommée HTTP 2.0) dont la spécification est plus récente (2014-2015).
- HTTP/3 – en cours de spécification

Performances du Web

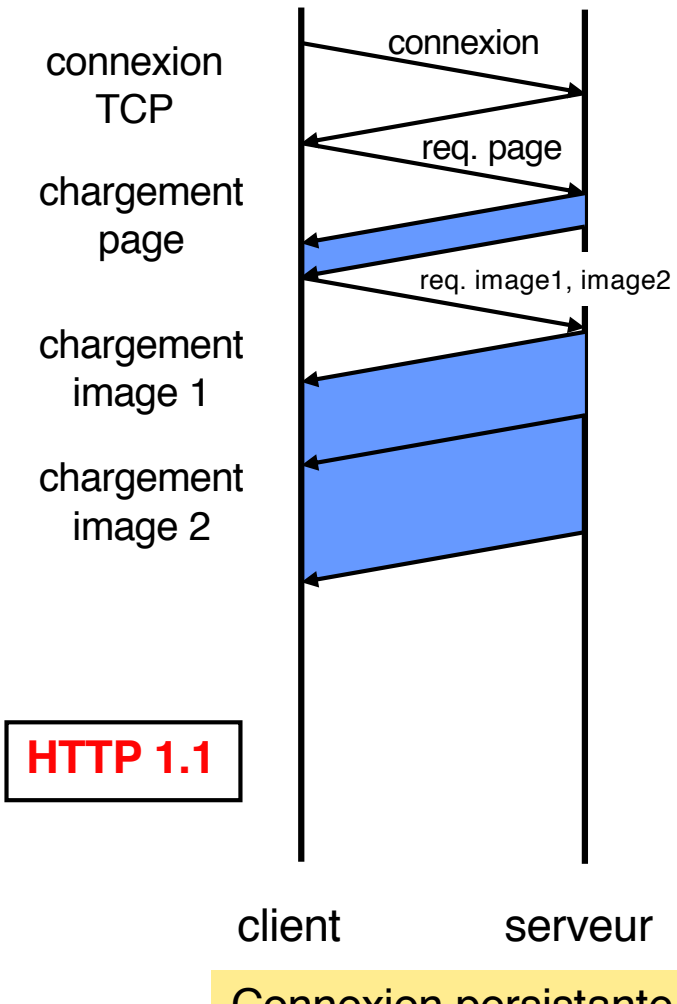
Exemple simple : 1 page HTML avec 2 images incluses



Connexion non persistante



Connexion persistante

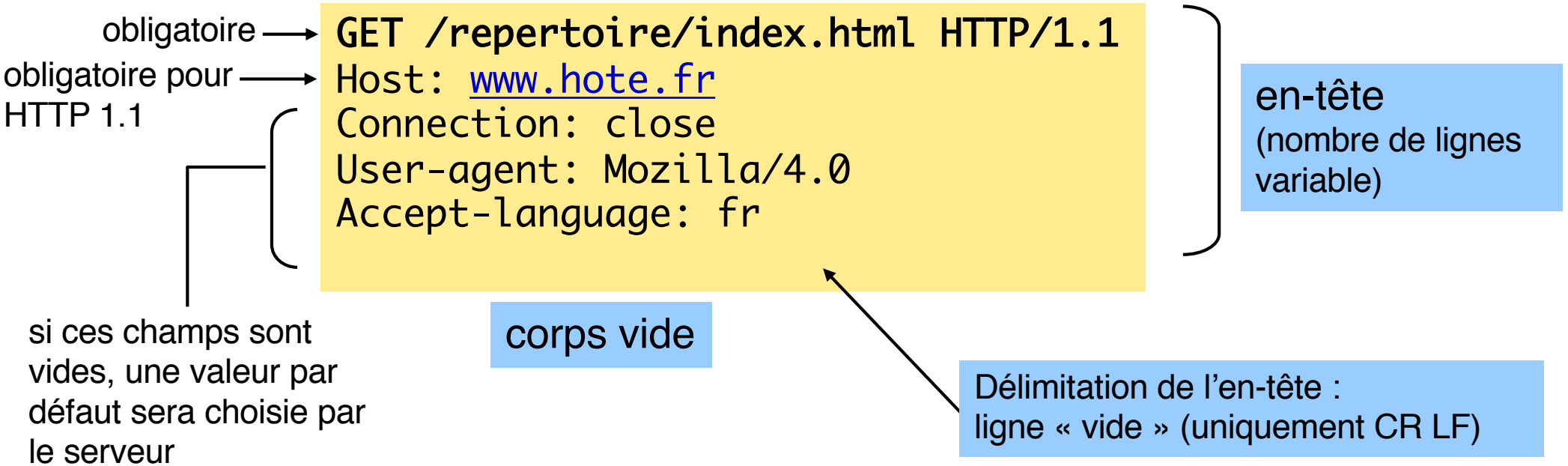


Connexion persistante avec pipeline

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 *corps/contenu* (obligatoire ou facultatif selon le type de requête/réponse).
Les en-têtes sont directement lisibles (texte ASCII)
L'encodage du corps dépend du type d'objet.
Chaque ligne d'en-tête est terminée par la séquence : CR LF

Exemple de requête :



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

Délimitation de l'en-tête : ligne « vide » (uniquement CR LF)

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 optionnel 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
500	erreur interne
505	version non gérée

HTTP : exemples (4)

Une requête POST

```
POST /repertoire/fichier HTTP/1.0
....
Content-Length: 328

.... 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 fil de discussion)
- 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)

Gestion de contenu dynamique

- La gestion du contenu dynamique est déléguée par le serveur à un processus fils

- Ce processus fils exécute un programme
  - 1er cas : le programme de génération de contenu dynamique correspond à du code natif => il est exécuté directement par le fils
  - 2ème cas : le programme de génération de contenu dynamique correspond à du code interprété => le processus fils exécute l'interpréteur de code correspondant (ce cas ne sera pas approfondi dans ce cours)

- Questions à considérer
  - Passage des arguments entre le client et le serveur
  - Passage des arguments (et d'autres informations) entre le serveur et le fils
  - Transfert du contenu produit vers le client
  - La prise en charge de ces aspects est définie dans la spécification CGI (décrite ci-après)
    - Il existe des variantes/optimisations. Par exemple FastCGI (FCGI) et SCGI.

Gestion de contenu dynamique
CGI : Common Gateway Interface

- La spécification CGI définit un standard simple pour le transfert des information entre un client Web (navigateur), un serveur et un processus fils (chargé de produire le contenu dynamique).

- Le code exécuté pour la génération dynamique de contenu est appelé *programme CGI* (car il adhère à la spécification CGI).

- Comme de nombreux programmes CGI sont écrits dans des langages interprétés, on les désigne souvent sous le nom de *scripts CGI*.

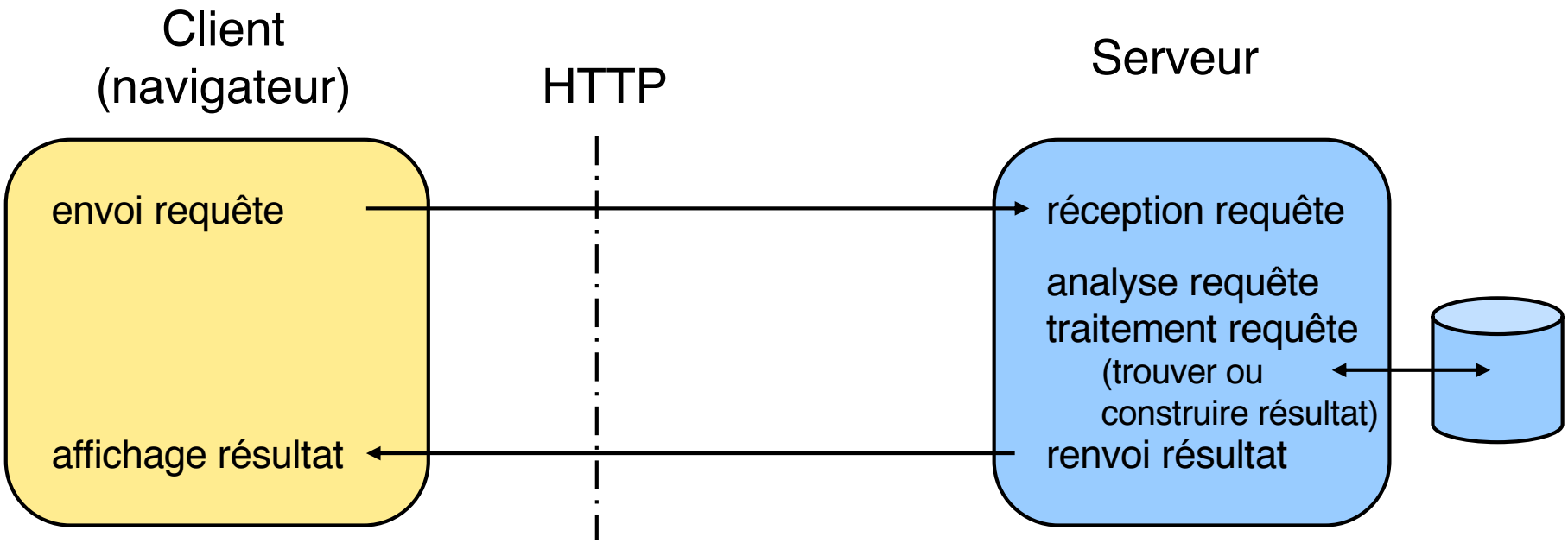
- Dans le modèle de base, le traitement de chaque requête dynamique nécessite la création d'un processus fils.

- Spécification complète : RFC 3875
  - http://tools.ietf.org/html/rfc3875

CGI : Common Gateway Interface

- Passage des arguments entre client et serveur
  - Concaténation derrière l'URI (ou bien dans le corps de la requête, dans le cas de POST)
  - Début de la liste avec « ? », séparateur « & » entre arguments, caractère *espace* représenté par « + » ou «%20 »
- Passage d'informations entre serveur et fils
  - Arguments : via la variable d'environnement QUERY_STRING
    - Une seule chaîne de caractères contenant tout ce qui suit le marqueur « ? »
  - Autre informations (concernant le serveur, le client, le type de requête ...) : via des variables d'environnement aux noms prédéfinis
    - Voir détails dans la spécification
  - Transfert du contenu produit vers le client
    - Redirection (par le père) de la sortie du fils

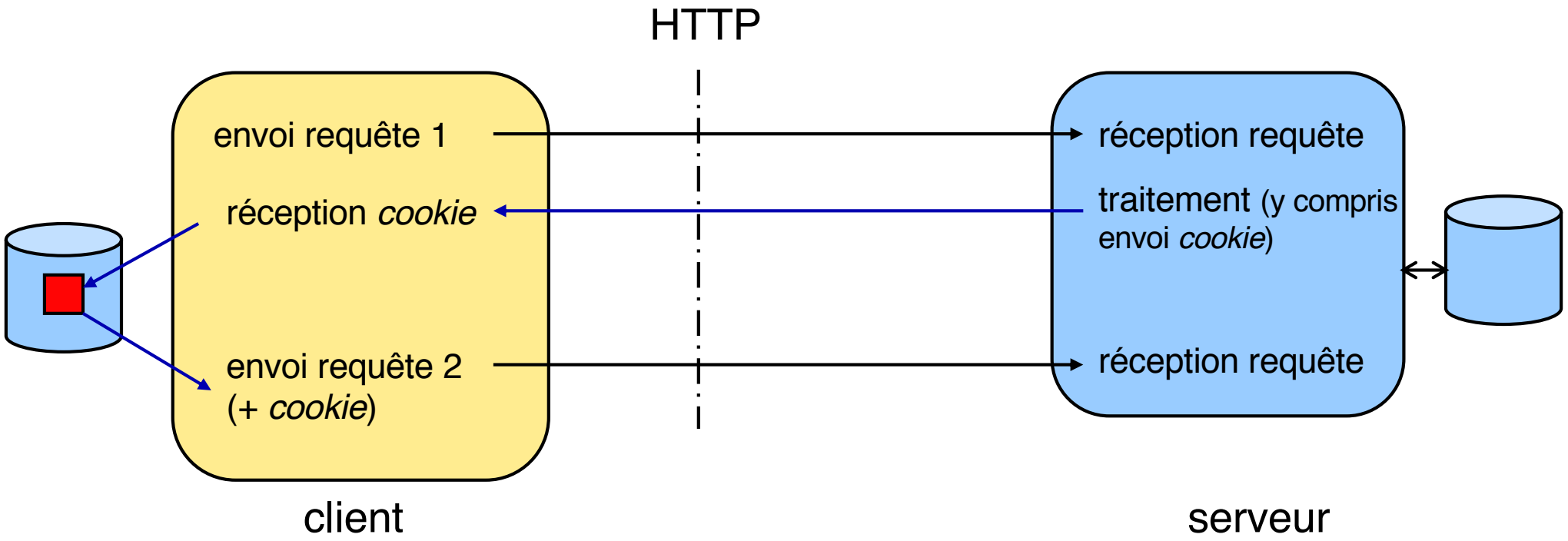
Échange sur le Web : schéma de base



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 (au sein d’une même connexion et a fortiori dans des connexions différentes).

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

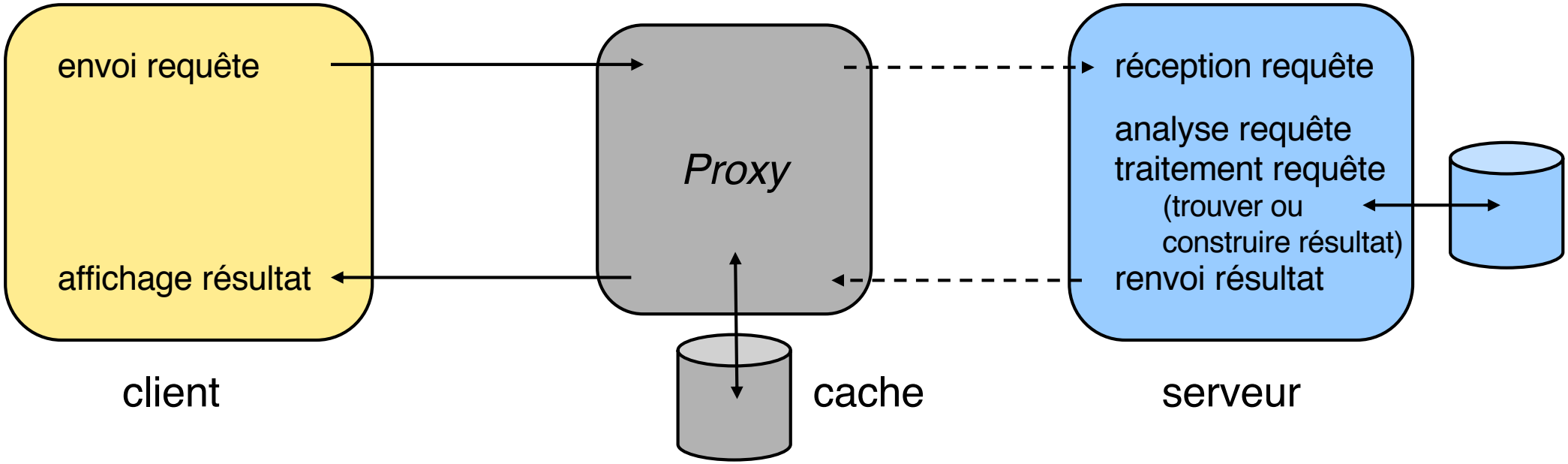
Le mécanisme de cookies permet à une application serveur d'implémenter une notion de session au dessus du protocole HTTP sans état.
Un *cookie* reçu dans une réponse du serveur est renvoyé par le client lors des interactions suivantes avec le même serveur.



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

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).
- **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 (`smtp`), etc.

Schéma d'un serveur Web

Étapes du traitement d'une requête (HTTP) :

1. **Lire et analyser le message HTTP** ; extraire nom 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 avec une architecture concurrente (pour servir de nombreux clients en parallèle, cf. cours sur client-serveur).

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

Quelques références sur l'histoire et l'avenir du Web

- Web 25th anniversary: https://www.w3.org/webat25/
- Discours de Tim Berners Lee (Mai 2018) suite à l'obtention du prix Turing (Turing Award lecture)
  - https://www.youtube.com/watch?v=BaMa4u4Fio4

Utiliser HTTP avec Python (1/2)

- La bibliothèque standard Python fournit un ensemble de fonctionnalités pour intégrer du code client-serveur HTTP dans des applications.
  - Voir notamment les documentations suivantes:
    - https://docs.python.org/3/library/internet.html
    - https://docs.python.org/3/library/http.client.html
    - https://docs.python.org/3/library/socketserver.html
    - https://docs.python.org/3/library/http.server.html
    - https://docs.python.org/3/library/cgi.html
    - https://realpython.com/courses/python-requests/
    - https://realpython.com/python-requests/
  - Remarque : l'étude du code source peut aussi être utile pour mieux comprendre le protocole (les pages de documentation ci-dessus contiennent les liens vers le code source)
- Pour HTTP/2, voir :
  - https://www.python-httpx.org
  - https://python-hyper.org/projects/h2/en/stable/

Utiliser HTTP avec Python (2/2)

- Il existe des specifications pour la definition d'interfaces pour la programmation d'applications "web" (c'est-à-dire d'applications serveur basées sur HTTP) en Python

- Ces interfaces concernent la jonction entre un serveur HTTP "générique" (par exemple Apache) et le code d'une application serveur écrite en Python

- Exemples :
  - WSGI: Web Server Gateway Interface
  - ASGI: Asynchronous Server gateway Interface

Annexes

Exemple de formulaire HTML

- Un premier exemple d'interaction

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

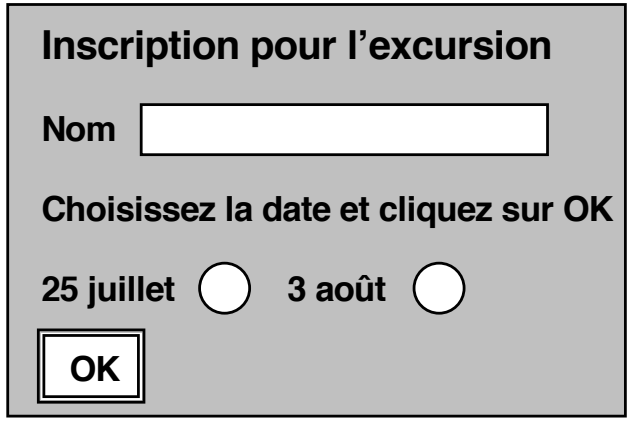
  On souhaite aussi permettre l'interaction entre client et serveur

  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



ce qui est envoyé (par exemple)

client=Dupont&date=2507

c'est le programme (script) indiqué dans le paramètre ACTION qui traitera cette entrée
et ce qui recevra une requête dont le type de méthode (PUT, POST, etc.) est défini dans le champ METHOD

Requêtes GET conditionnelles (1/2)

- Optimisation permettant de ne récupérer le contenu d'un objet que si ce dernier a été modifié par rapport à une version antérieure déjà récupérée par le client
  - Gain de temps (temps de transfert inutile)
  - Moindre utilisation des liens réseau
- Spécifiée via une ligne d'en-tête particulière
- Plusieurs critères possibles
  - Date de dernière modification: *If-Modified-Since: <date>*
  - Contenu de l'objet: *If-None-Match: <tags>*

Requêtes GET conditionnelles (2/2)

- Exemple

```
GET /repertoire/index.html HTTP/1.1
....
```

```
HTTP/1.1 200 OK
....
Date: Fri, 31 Dec 2013 23:59:59 GMT
....
```

```
GET /repertoire/index.html HTTP/1.1
....
If-Modified-Since: Fri, 31 Dec 2013 23:59:59 GMT
....
```

```
HTTP/1.1 304 Not modified
....
```

HTTP/2 (ancien nom : HTTP 2.0)

- Nouvelle version du protocole
  - Spécification récente (2014-2015)
  - Partiellement inspirée du protocole SPDY (proposé par Google en 2009)

- Principales caractéristiques
  - Motivation : **optimisation des performances**
  - **Ne modifie pas les fonctionnalités définies dans HTTP 1.1** (méthodes disponibles et sémantique correspondante, etc.)
  - Les évolutions concernent essentiellement la façon dont les messages (requêtes/réponses) sont transmis entre le client et le serveur
    - Les messages « classiques » (HTTP 1.1) sont encapsulés dans un nouveau protocole intermédiaire / format binaire, qui permet des optimisations
    - Lors de l'établissement d'une connexion entre client et serveur, une phase de négociation permet de basculer de HTTP 1.1 vers HTTP 2.0 (activation de la couche intermédiaire)

HTTP/2 (suite)

- Multiplexage des communications
  - **Possibilité d'utiliser une seule TCP pour véhiculer plusieurs flots d'interaction entre un client et un serveur.**
  - Permet d'éviter l'utilisation de connexions parallèles.
    - Économie de ressources et de temps
    - Meilleure régulation des performances par TCP
  - Une connexion TCP peut encapsuler plusieurs flots (*streams*) bidirectionnels. Un flot est un canal de communication virtuel qui s'appuie sur la connexion TCP sous-jacente.
  - On peut assigner des **priorités** différentes aux flots au sein d'une connexion.
  - Chaque *message* échangé au sein d'un flot peut éventuellement être découpé en plusieurs fragments (*frames*)
  - Les fragments associés à différents flots HTTP peuvent être entrelacés de manière arbitraire au sein d'un flot d'octets TCP

HTTP/2 (suite)

- Format d'encapsulation
  - Les messages « classiques » (avec en-tête sous forme textuelle) sont encapsulés dans un format binaire qui permet un traitement plus efficace par le récepteur
  - Exemples :
    - Identification rapide de la partie « en-tête » et de la partie « contenu » d'un message (sans nécessité de parsing)
    - Encodage des lignes d'en-tête de façon à éviter de renvoyer les lignes identiques à celle des messages précédents

- Réponses multiples à l'initiative du serveur (*server push*)
  - Le serveur peut envoyer plusieurs réponses/objets en réaction à une requête, en anticipation des prochaines requêtes du client
  - Exemple : client demande une page HTML, serveur renvoie page HTML ainsi que les fichiers référencés par cette page (feuille de style CSS, images, etc.)

HTTP/2 (suite)

- Pour plus de détails, voir :
  - **Résumé des principales caractéristiques de HTTP/2 :** I. Grigorik. Making the Web Faster with HTTP 2.0. ACM Queue, 2013. http://queue.acm.org/detail.cfm?id=2555617

- Et aussi :
  - **Groupe de travail sur la spécification HTTP/2:**
    - http://http2.github.io
    - http://trac.tools.ietf.org/wg/httpbis/trac/wiki
  - **Description de SPDY** (l'un des principaux protocoles ayant inspiré HTTP 2.0) : B. Thomas, R. Jurdak, and I. Atkinson, "SPDYing Up the Web", Communications of the ACM, 55(12):64-73, December 2012. http://jurdak.com/CACM12.pdf

HTTP/2 - Illustrations

- Partage de connexion TCP et entrelacement des flots

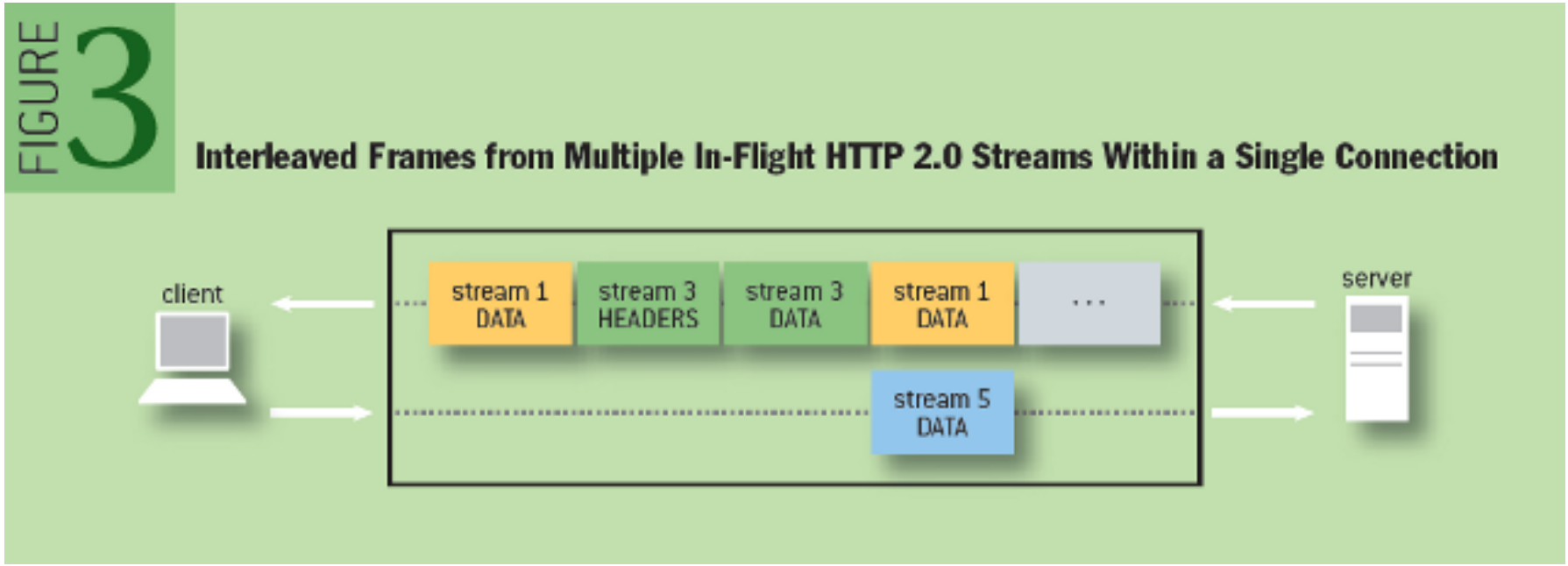


Schéma extrait de : I. Grigorik. Making the Web Faster with HTTP 2.0. ACM Queue. 2013.

HTTP/2 - Illustrations

- Encapsulation binaire

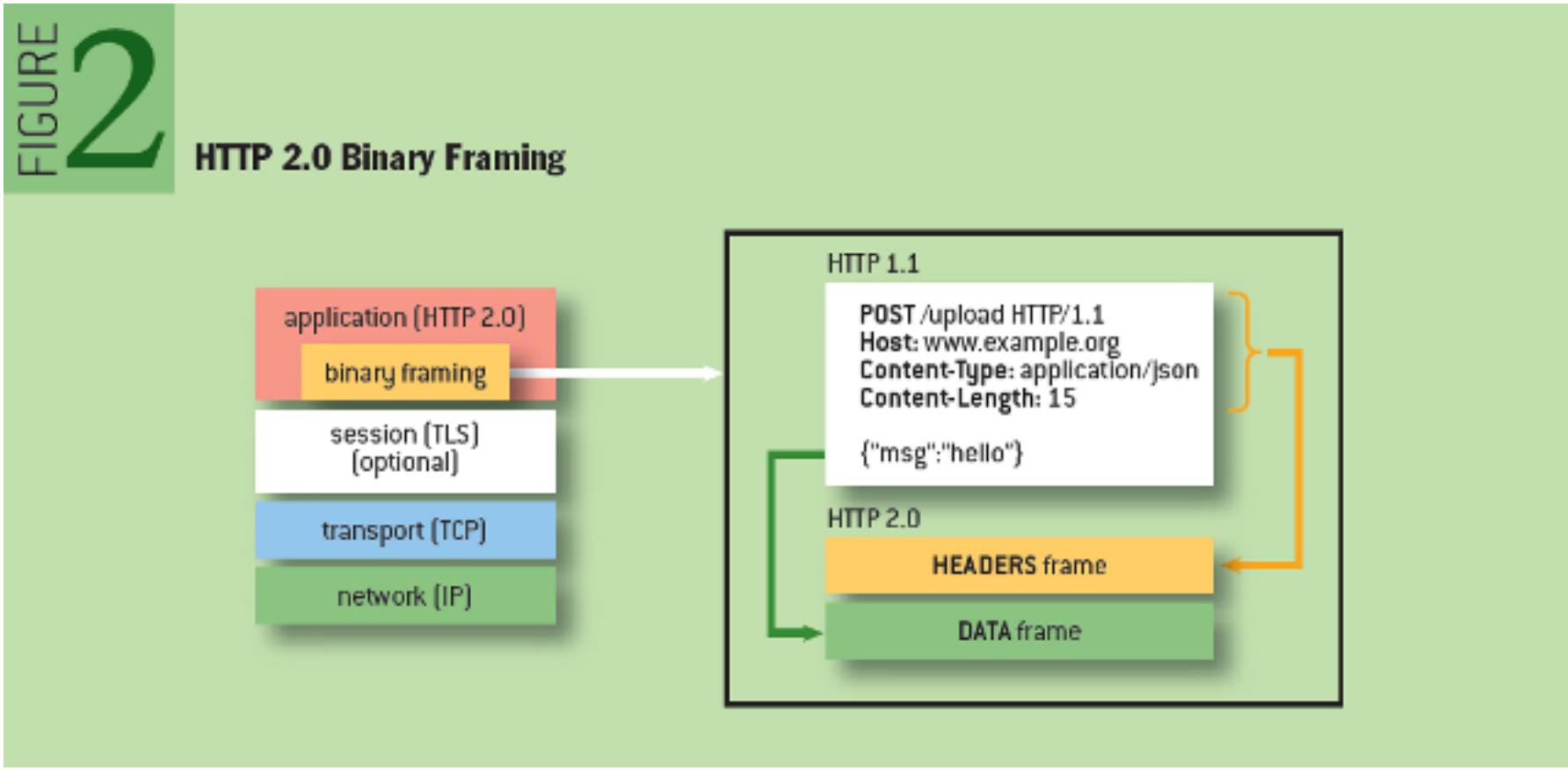


Schéma extrait de : I. Grigorik. Making the Web Faster with HTTP 2.0. ACM Queue. 2013.

HTTP/2 - Illustrations

- Encapsulation binaire (suite)

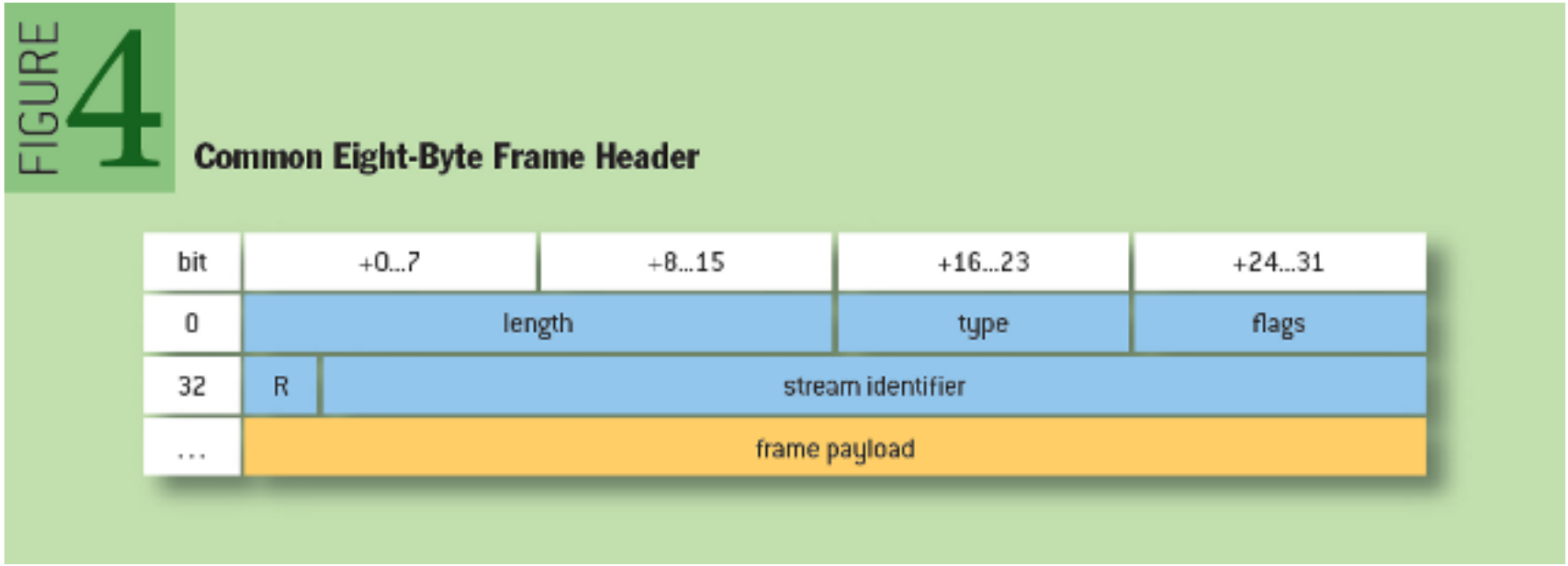


Schéma extrait de : I. Grigorik. Making the Web Faster with HTTP 2.0. ACM Queue. 2013.

HTTP/2 - Illustrations

- Encodage différentiel des en-têtes

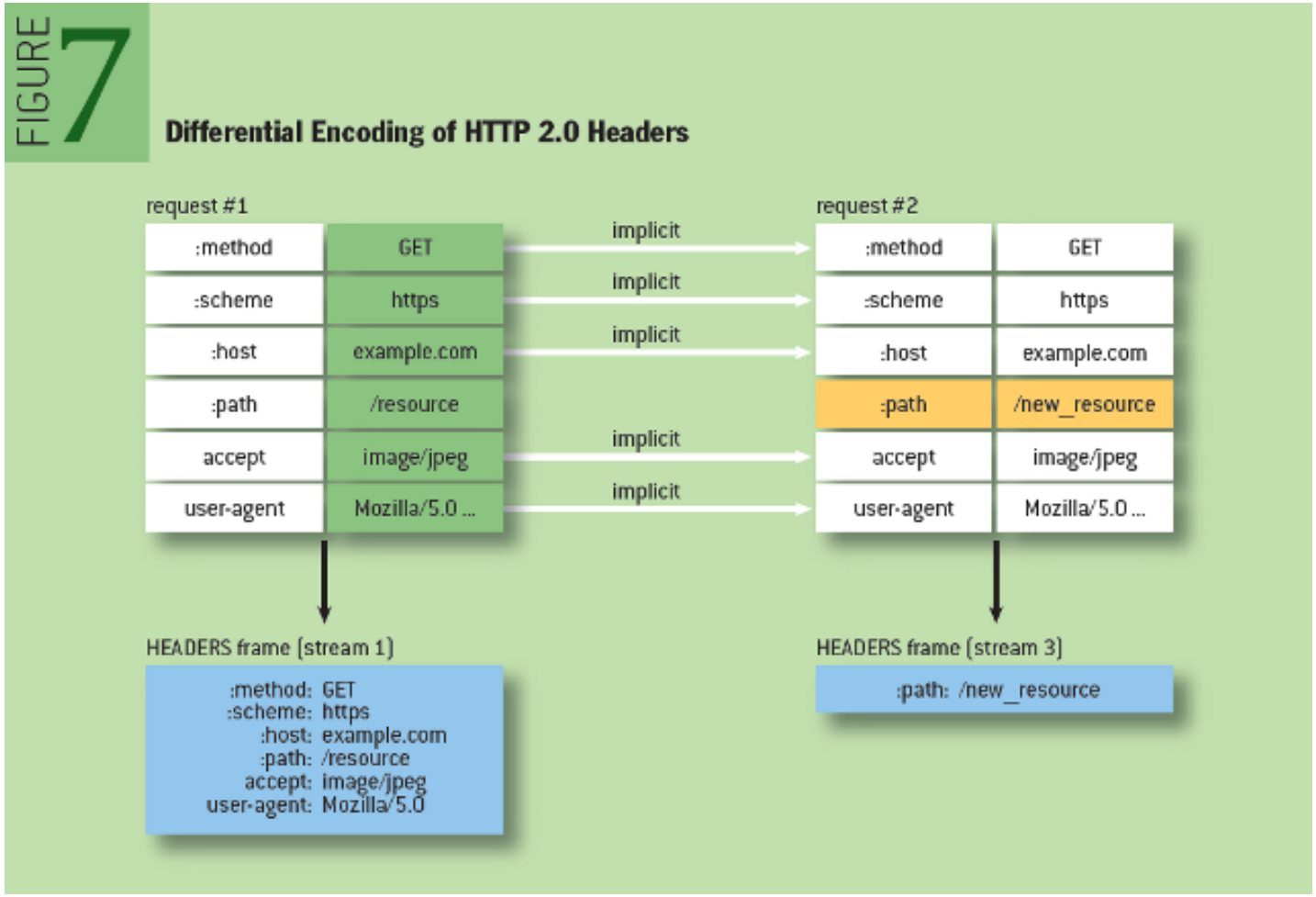


Schéma extrait de : I. Grigorik. Making the Web Faster with HTTP 2.0. ACM Queue. 2013.

HTTP/2 - Illustrations

- Négociation / basculement vers HTTP/2

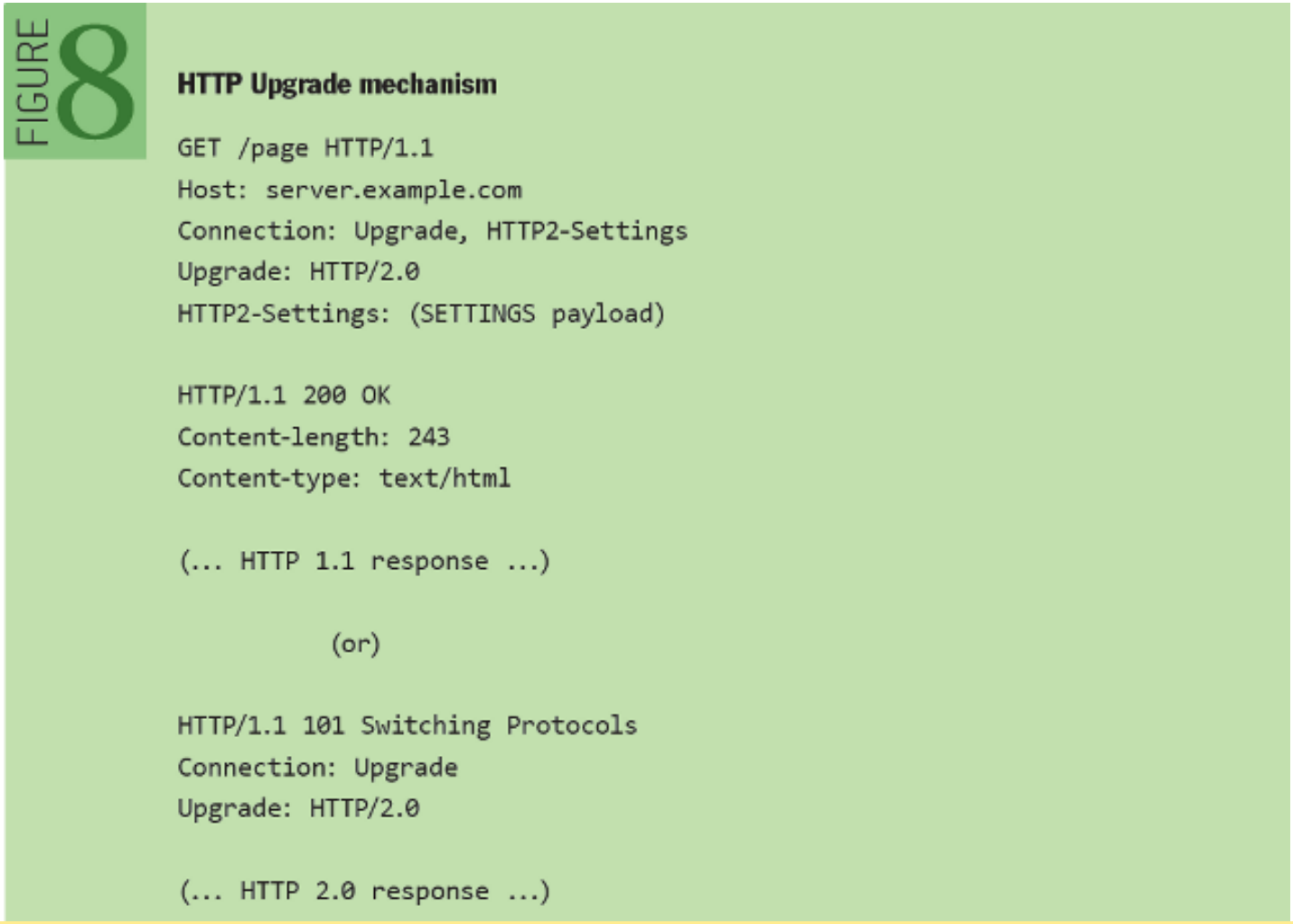


Schéma extrait de : I. Grigorik. Making the Web Faster with HTTP 2.0. ACM Queue. 2013.

HTTP/3

- Une nouvelle version du protocole, en cours de développement et de standardisation

- **La principale modification concerne le changement de protocole de niveau 4 (transport) sous-jacent :**
  - Utilisation de QUIC (au lieu de TCP)
  - HTTP/3 était d'ailleurs précédemment connu sous le nom "HTTP over QUIC"

- Motivations
  - Amélioration de performances
    - Meilleurs temps de réponse
    - Meilleur comportement lors de pertes de paquets
  - Amélioration de la confidentialité

QUIC

- Un nouveau protocole de transport
  - Nom initialement associé à un acronyme : “Quick UDP Internet Connection”
  - Proposé par Google (et déjà utilisé dans bon nombre de ses applications)
  - En cours de standardisation par l’IETF (Internet Engineering Task Force)
  - Principalement imaginé pour améliorer les performances du protocole applicatif HTTP

- Objectifs
  - Fournir des garanties similaires à TCP : fiabilité des communications, contrôle de flux, contrôle de congestion
  - Fournir de meilleures performances
    - Multiplexage de connexions « logiques »
    - Établissement rapide de connexions sécurisées
    - Évitement du problème de *head-of-line blocking* de TCP (en cas de perte/délais de certains paquets)
    - Évitement de certaines retransmissions (grâce à code correcteur d’erreurs)
    - Gestion efficace de la mobilité des connexions
  - Garantir une meilleure sécurité des communications
  - Permettre une évolution rapide du protocole malgré l’ossification du cœur de l’Internet

- Caractéristiques originales
  - Implémenté au dessus d’un autre protocole de transport : UDP
  - Implémenté (actuellement) en mode utilisateur (plutôt que dans le noyau du système d’exploitation)

Références pour HTTP/3 & QUIC

Spécification HTTP/3

- NB : Pas encore finalisée.
- **Document de travail** (mars 2021) : https://quicwg.org/base-drafts/draft-ietf-quic-http.html
- **Tutoriel multilingue** : HTTP/3 explained. https://http3-explained.haxx.se

QUIC

- **Article de recherche :** A. Langley et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. Proceedings of SIGCOMM 2017. http://www.audentia-gestion.fr/Recherche-Research-Google/46403.pdf
- **Spécification :** J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-27. IETF Secretariat, Feb. 2020. https://datatracker.ietf.org/doc/draft-ietf-quic-transport/
- **Tutoriel :** Maxime Piraux, Quentin De Coninck, François Michel. QUIC Tutorial. July 2019. https://docs.google.com/presentation/d/1HHrbra6OaJ8zHAnDSnu8zbCaQ_HqnkjZ0auX2DMluoY/edit#slide=id.p