HTTP
LE PROTOCOLE DU WEB

Développer des applications Web

Qu'est-ce que HTTP?

- HTTP (Hypertext Transfer Protocol) est un protocole de communication client-serveur développé pour le Web.
- HTTP est un protocole de la couche application et peut fonctionner sur n'importe quelle connexion fiable (généralement une connexion TCP).
- HTTP permet à un client et un serveur HTTP d'échanger des représentations de ressources (documents, images, sons, résultats de requêtes SQL, etc.).
- Rappel: Un protocole est l'ensemble des règles qui gouvernent une conversation entre un client et un serveur.

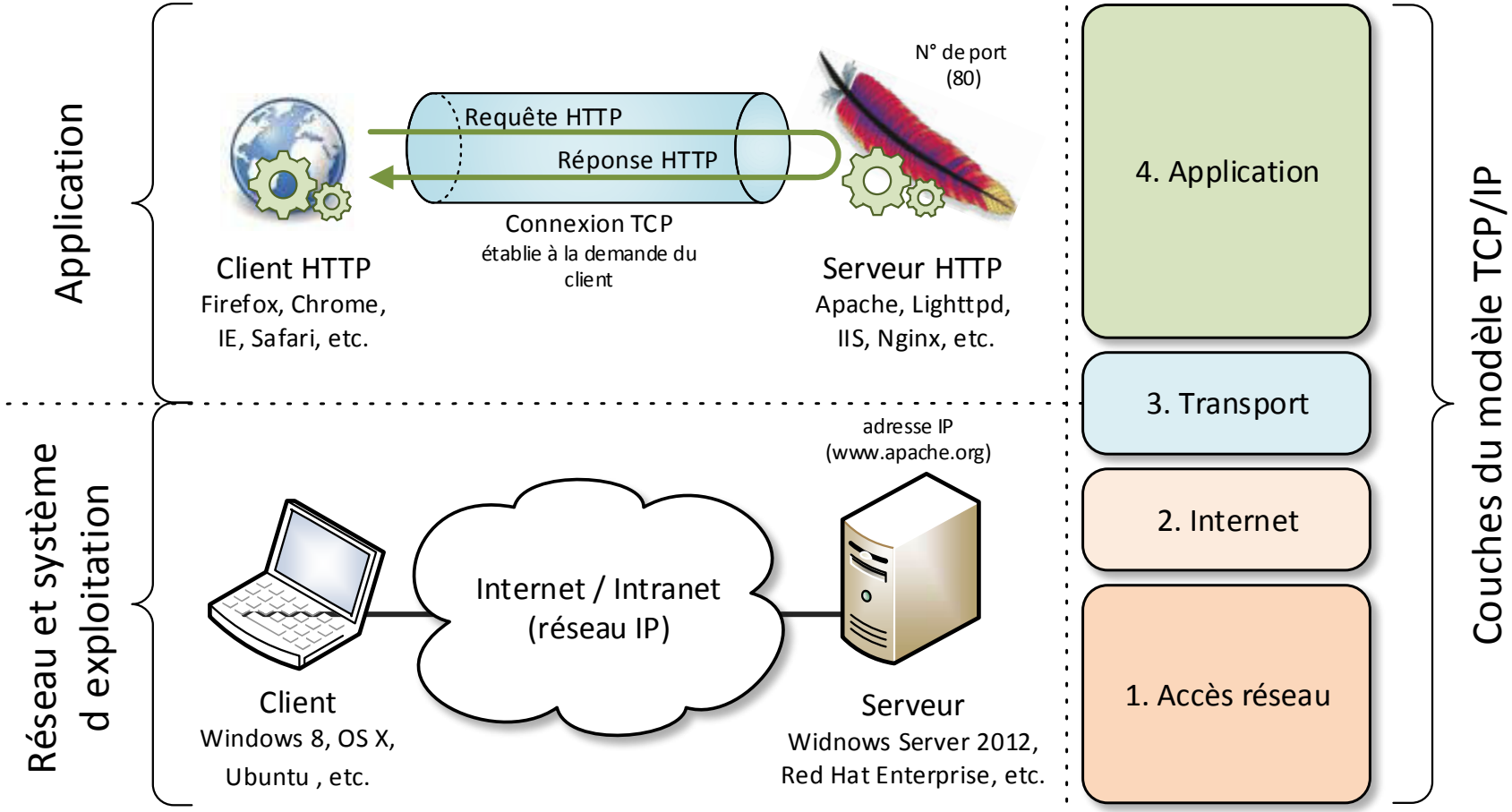
Client et serveur HTTP

- Client HTTP: programme, souvent un navigateur web (Firefox, Chrome, etc.), qui établit une connexion vers un serveur HTTP et lui envoie une ou plusieurs requêtes HTTP.
- Serveur HTTP: programme qui accepte les demandes de connexions des clients et répond à chacune de leurs requêtes HTTP par une réponse HTTP.
- Un programme peut être à la fois client et serveur.
- Remarques: un serveur HTTP n'est pas un ordinateur, mais un programme. Un client et un serveur peuvent fonctionner simultanément sur le même ordinateur.

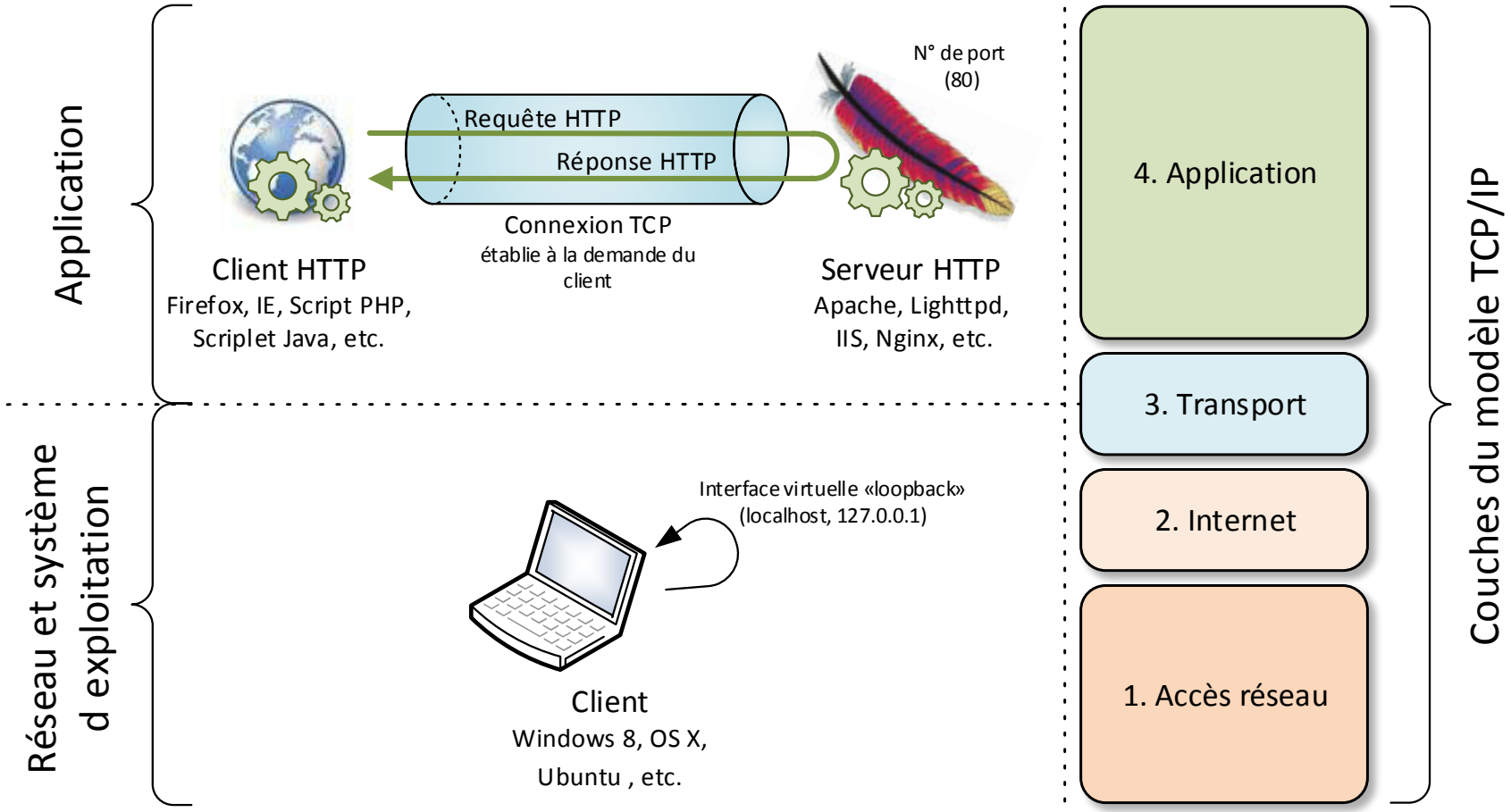
Modèles d'échange de messages

- Il existe deux grands modèles d'échanges de messages pour la communication client-serveur :
  - requête-réponse (*request-response*)
  - sens unique (*one-way*)
- HTTP utilise le modèle requête-réponse :
  1. Un échange commence lorsqu'un client envoie une requête sous la forme d'un message de requête (*request message*)
  2. Lorsqu'il reçoit le message de requête, le serveur y répond par un message de réponse (*response message*).
  3. L'échange se termine lorsque le client a reçu la réponse ou lorsque la connexion est interrompue.

Vue d'ensemble

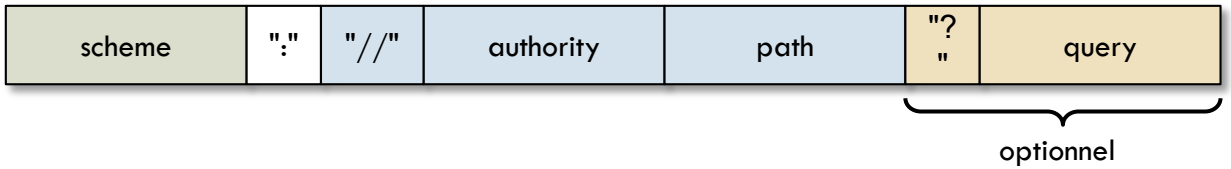


Vue d'ensemble (développement)



URI – Uniform Resource Identifier

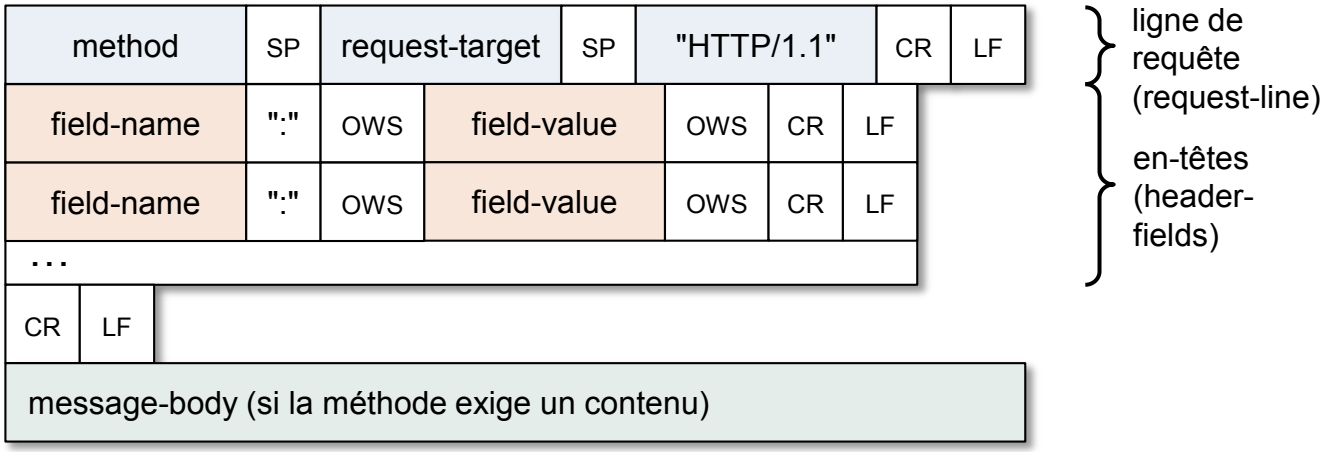
- Toutes les ressources du Web sont identifiées par des «noms» qui ont tous la même forme.
- Ces «noms» sont appelés URI (*uniform resource identifier*) ou URL (*uniform resource locator*).
- Pour les ressources du Web, un URI à la forme suivante :



scheme : le nom du protocole (`http` ou `https`)
authority : le nom du serveur (`www.exemple.org`) ou son adresse IP (`192.168.1.1`)
path : le chemin de la ressource dans le serveur (`/chemin/test.html`)
query : données à passer au serveur (`?arg1=val1&arg2=val2`)

Message de requête HTTP

- Format d'un message de requête :



SP : espace
CR : retour de chariot (carriage return)
LF : saut de ligne (line feed)
OWS : un nombre quelconque d'espaces/tabulations (optional white space)

Méthode (*method*)

- Le champ «*method*» prend généralement l'une des valeurs suivantes: GET, POST, PUT, DELETE ou HEAD.
- Un navigateur ne supporte que les méthodes GET et POST.
- Méthode GET
  - Permet d'obtenir une représentation pour une ressource; ne modifie pas l'état externe du serveur.
  - Utilisée par les éléments HTML qui ont un attribut `href` ou `src` et les boutons de navigation et la barre d'adresse du navigateur.
- Méthode POST
  - Permet d'envoyer des données au serveur en les plaçant dans le corps de la requête; modifie l'état externe du serveur.
  - Utilisé par l'élément HTML `<form>`.

Cible de la requête (*request-target*)

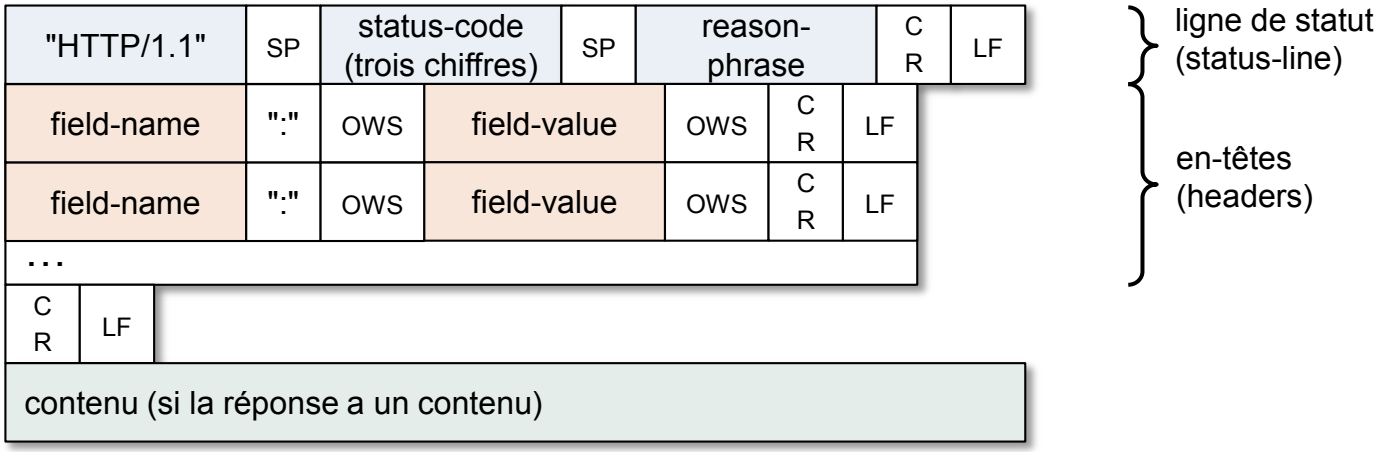
- En général, le champ «*request-target*» contient l'URI de la ressource sans le protocole et le nom du serveur.
- Le nom du serveur doit être spécifié à l'aide de l'en-tête «Host».

```
http://www.exemple.org/chemin/test?arg1=val1
```

```
GET /chemin/test?arg1=val1 HTTP/1.1
Host: www.exemple.org
```

Message de réponse HTTP

- Format général d'un message de réponse :



SP : espace
CR : retour de chariot (carriage return)
LF : saut de ligne (line feed)
OWS : un nombre quelconque d'espaces/tabulations (optional white space)

Code de statut HTTP

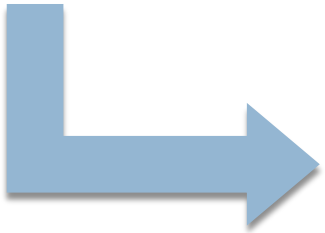
- Un serveur envoie une réponse pour chaque requête, même s'il ne peut pas y répondre avec succès.
- La réponse contient un champ «*status-code*» qui informe le client de ce qui s'est passé par un code de trois chiffres.
- Code 2xx: Succès
  - 200: Ok
  - 201: Une nouvelle ressource a été créée.
  - 204: Pas de contenu.

Code de statut HTTP (suite)

- Code 3xx: Redirections
  - 301: La ressource a été déplacée de façon permanente
  - 302: La ressource a été déplacée temporairement
  - 303: *See other* (utilisé pour une redirection après une requête POST)
  - 304: La ressource n'a pas été modifiée
- Code 4xx: Erreurs du côté client
  - 401: L'accès à la ressource n'est pas autorisé sans authentification.
  - 403: L'accès à la ressource est interdit.
  - 404: La ressource n'a pas pu être trouvée.
- Code 5xx: Erreurs du côté serveur
  - 500: Erreur Interne dans le serveur.

Exemple 1 – Requête GET


```
GET /alloffame.php HTTP/1.1
Host: www.epai-ict.ch
Connection: keep-alive
Accept: text/html
Accept-Language: fr, en
```



```
HTTP/1.1 200 OK
Date: Tue, 28 Oct 2014 23:55:11 GMT
Server: Apache
Cache-Control: no-cache
Content-Length: 3854
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<html>
    <head>
        <title>Hall of fame</title>
        <link rel="stylesheet" href="app.css">
    </head>
    <body>
        <h2>Hall of fame</h2>
        <div class="all-of-fame">
            <table>
                <th scope="row">Nom de naissance</th>
                <td>Timothy Berners-Lee</td>
[…]
```

Exemple 2 – Requête POST


```
[…]
<form action="login.php" method="POST">
    Nom:
    <input type="text" name="username"><br>
    Mot de passe:
    <input type="password" name="password"><br>
    <input type="submit" value="Login">
</form>
[…]
```

```
POST /login.php HTTP/1.1
Host: localhost
Connection: keep-alive
Content-Length: 29
Cache-Control: max-age=0
Accept: text/html
Origin: http://localhost
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded
Referer: http://localhost/~dev/Activity1/login.php
Accept-Language: fr, en

username=arthur&password=king
```

```
HTTP/1.1 303 See other
Date: Thu, 30 Oct 2014 21:21:05 GMT
Server: Apache
Location: http://localhost/guestbook.php
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Protocole «sans état»

- HTTP est un protocole «sans état» (*stateless protocol*) contrairement à SMTP ou FTP qui sont des protocoles dits «avec état» (*stateful protocol*).
- Un protocole «sans état» est un protocole avec lequel :
  - chaque échange de messages (une requête et une réponse) est traité comme une transaction indépendante.
  - l'état interne du serveur est exactement le même avant et après le traitement d'une requête, c.-à-d. qu'on pourrait redémarrer le serveur après une requête sans que cela n'affecte la réponse de la requête suivante.

État interne & état externe

- Un serveur HTTP (le programme qui traite les requêtes) est en principe sans état. Son état interne devrait être le même avant et après le traitement d'une requête.
- En revanche, un serveur HTTP peut avoir un état externe:
  - En interagissant avec un autre serveur (serveur SQL, serveur d'application, etc.) pour enregistrer des données.
  - En ajoutant à la réponse, des données que le client est chargé de retenir et de renvoyer avec les requêtes suivantes (Cookie).
- Dans les deux cas, l'état qui a changé est celui d'un autre programme (serveur SQL ou navigateur).

Pourquoi «pas d'état»?

- En général, un serveur traite des requêtes pour un grand nombre de clients, il est donc important que le programme soit robuste.
- Un serveur sans état à plusieurs avantages du point de vue de la robustesse:
  - À qualité égale, un programme plus simple est plus robuste et un serveur sans état est plus simple (pas de gestion de session).
  - Il peut être redémarré à la fin de n'importe quelle requête si une consommation de ressource excessive est détectée.
  - Les requêtes peuvent être traitées par plusieurs instances du serveur (répartition de charge, redondance).

HTTP Cookie

- Avec les cookies (*HTTP State Management Mechanism*), l'IETF propose un moyen qui permet à un serveur d'enregistrer des données dans la mémoire du client.
- Il est ainsi possible de conserver l'état d'une session sans avoir à sacrifier la simplicité d'un serveur sans états.
- Principe:
  - Le serveur demande au client d'enregistrer des données à l'aide de l'en-tête `set-cookie` de la réponse.
  - Si le client supporte les cookies, il enregistre les données dans un conteneur (fichier, base de données) dédié à ce serveur.
  - Lorsque client fait une nouvelle requête à ce même serveur, il renvoie les données stockées à l'aide de l'en-tête `cookie`.

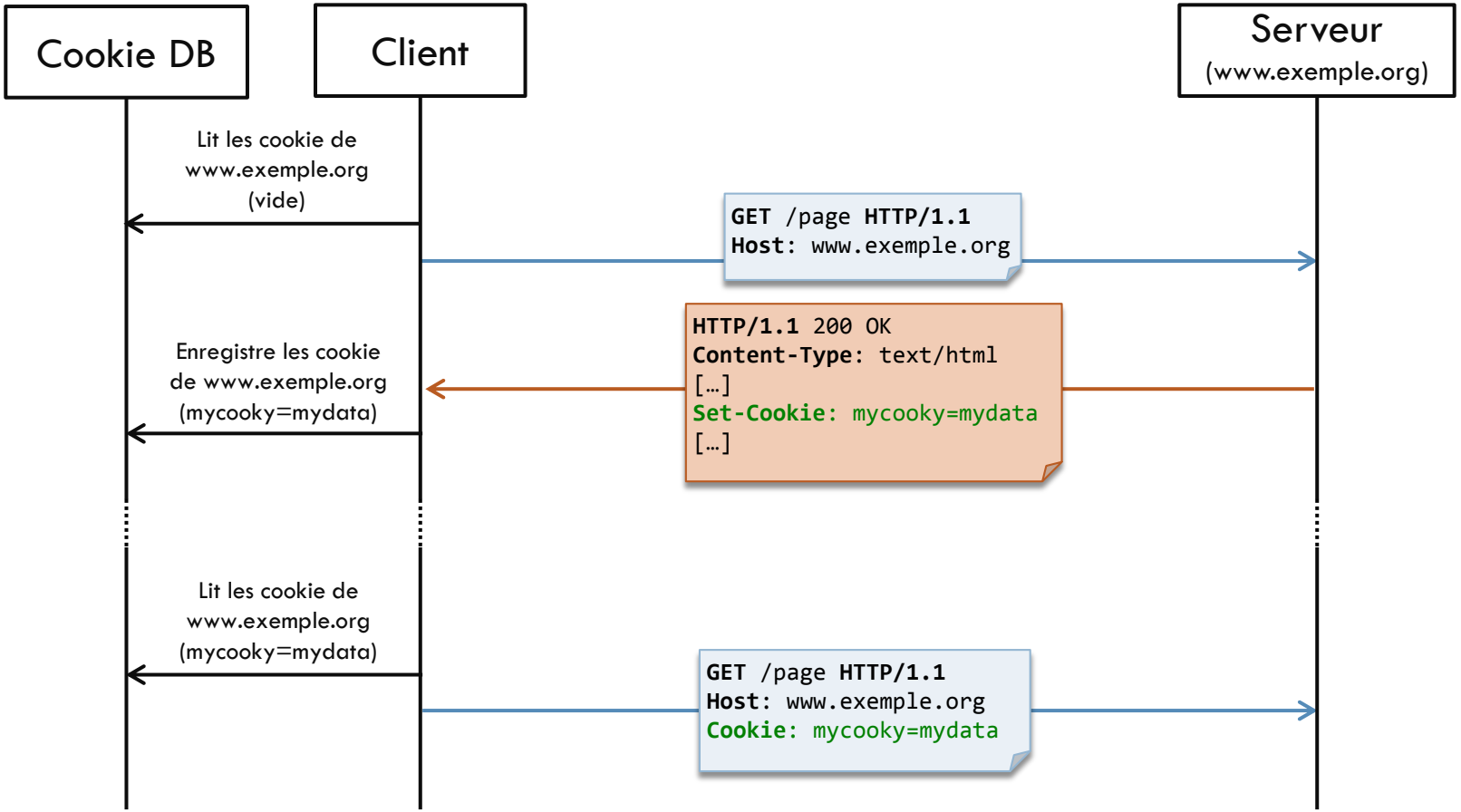
HTTP Cookie – Format

- Format de l'en-tête `Set-Cookie` :

```
Set-Cookie: <cookie-name>=<value> *(; SP <attribute-name>=<value>)
```

- Les cookies supportent les attributs suivants:
  - `Expires` : la date à laquelle expire le cookie (date RFC-1123)
  - `Max-Age` : l'âge maximal, en seconde, du cookie
  - `Domaine` : les sous-domaines dans lesquels le cookie est valide
  - `Path` : le chemin de l'URI pour lequel le cookie est valide
  - `Secure` : indique que le cookie ne doit être transmis qu'avec HTTPS
  - `HttpOnly` : indique que le cookie ne peut pas utilisé en JavaScript
- Dans la même réponse HTTP, un navigateur devrait accepter jusqu'à vingt en-têtes `Set-Cookie`.

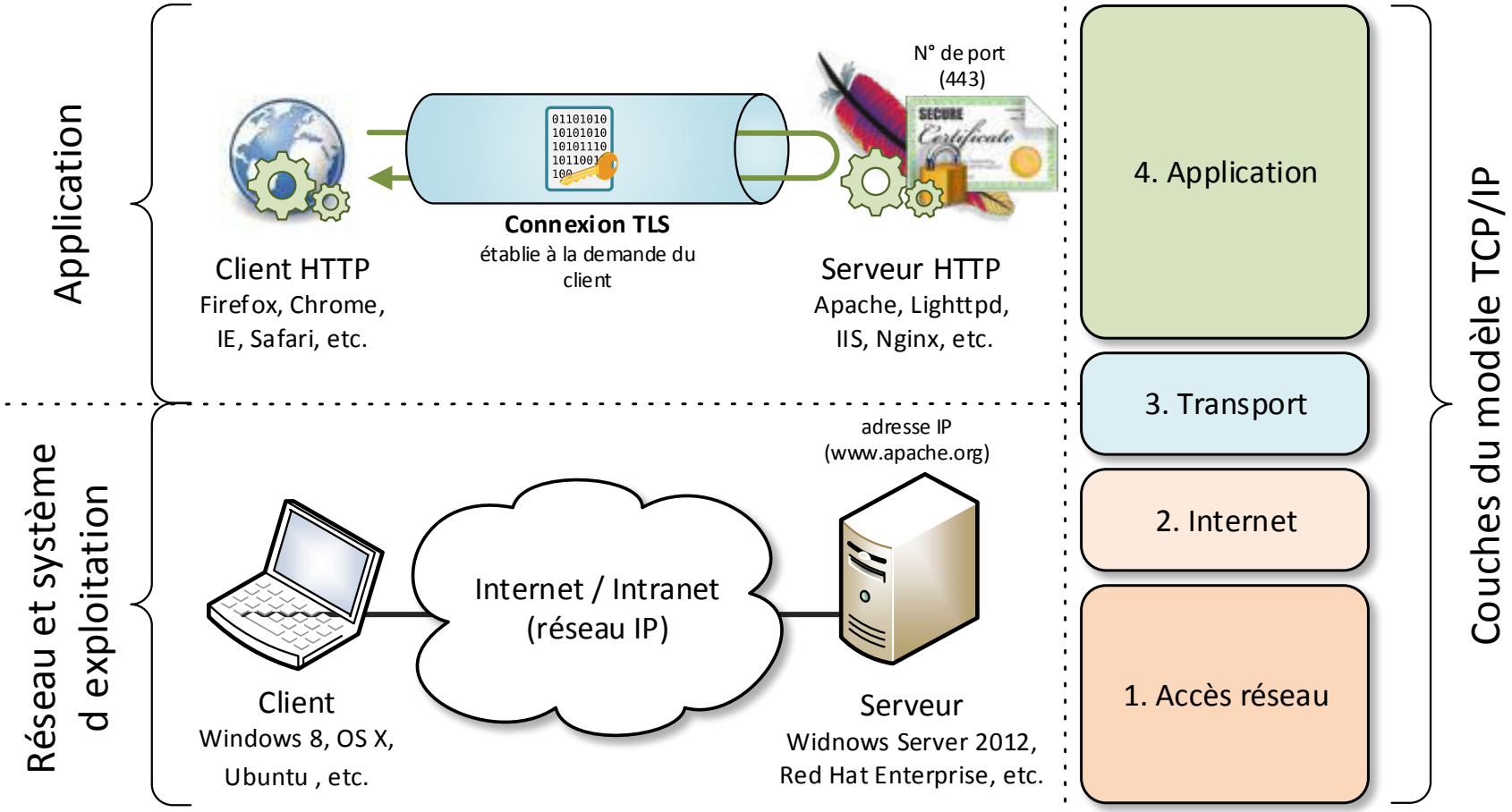
HTTP Cookie – exemple



HTTPS

- Les messages du protocole circulent sous forme de texte dans une connexion TCP.
- Si un tiers intercepte le trafic, des données sensibles pourraient donc être compromises.
- Une solution est le chiffrage du canal de transport à l'aide du protocole TLS (Transport Layer Security) souvent encore appelé SSL (Secure Socket Layer).
- Une fois que la connexion sécurisée établie, les échanges de messages HTTP se déroulent de la même manière.
- L'utilisation de HTTPS nécessite un certificat SSL qui peut être obtenu auprès d'une autorité de certification (CA).

HTTPS – Vue d'ensemble



Quelques liens

- RFC 7230
  HTTP/1.1: Message syntax and Routing
- RFC 6265
  HTTP State Management Mechanism (HTTP Cookie)
- Let's Encrypt
  Autorité de certification libre et gratuite soutenue par la fondation Linux