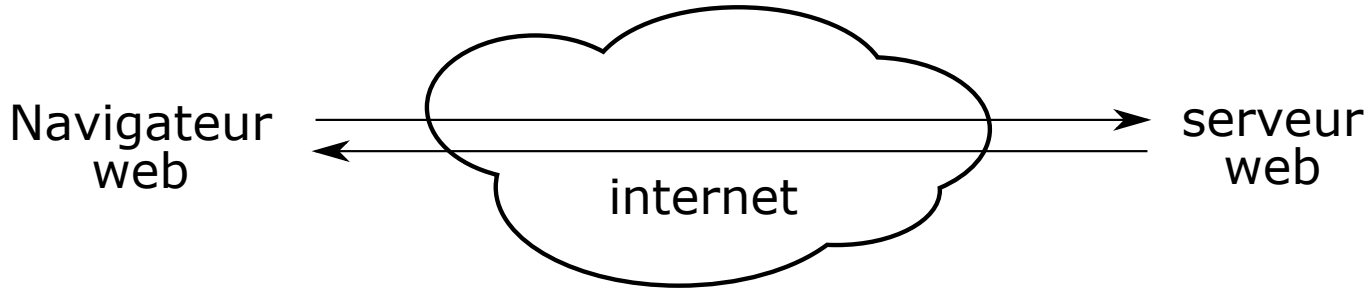
Cinquième partie

Technologies Web

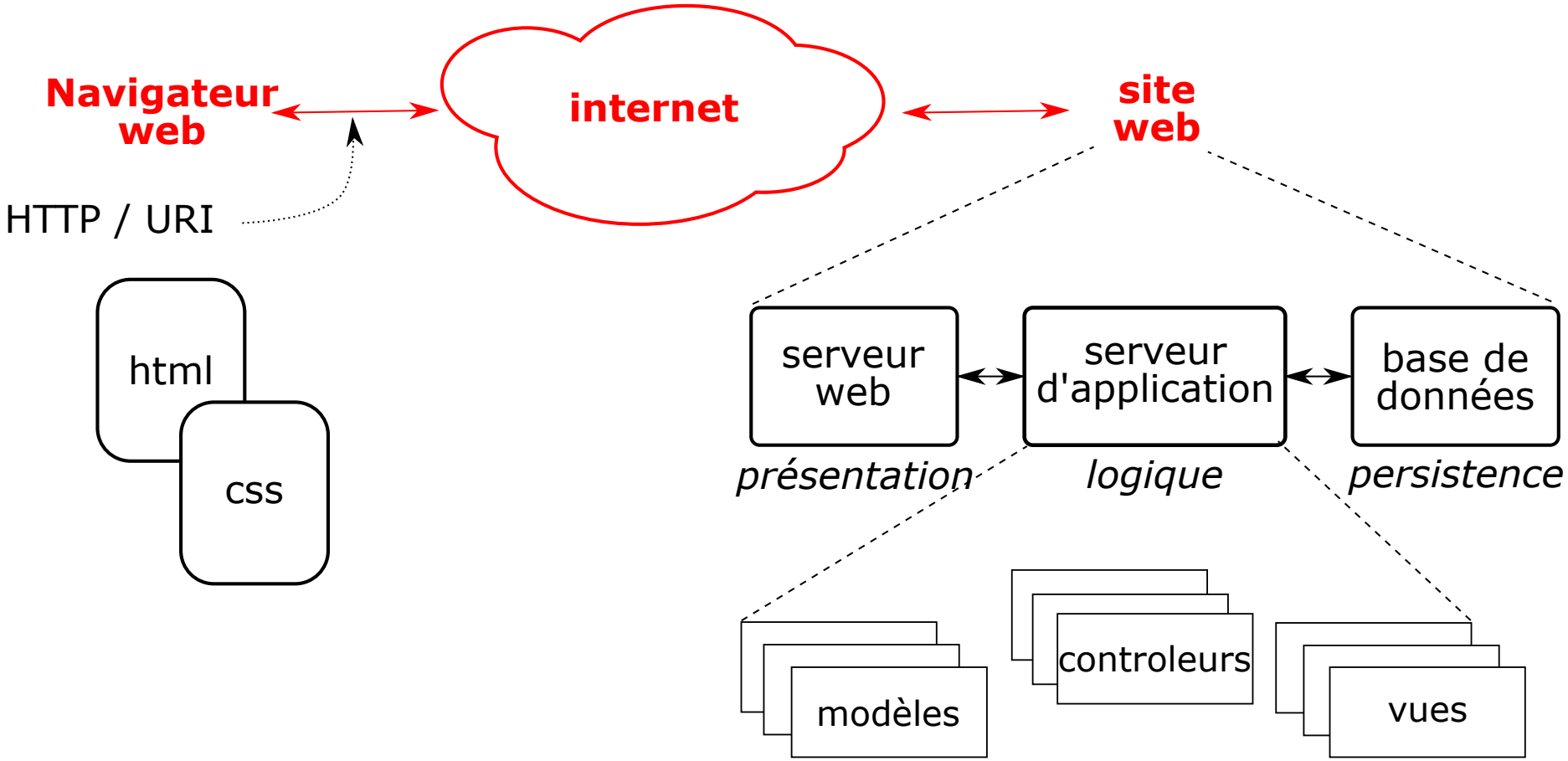
Plan

Le web vu de loin

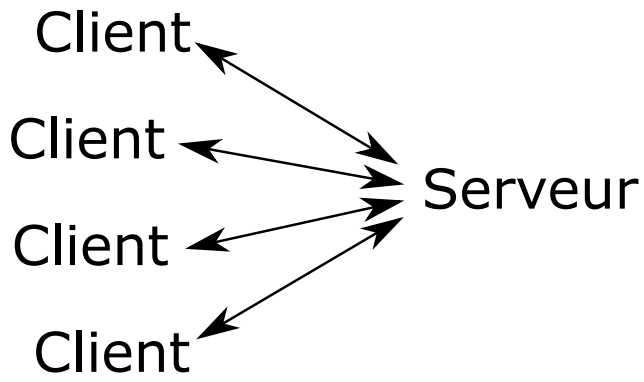
- Le web est une architecture client-serveur
- Essentiellement requête/réponse

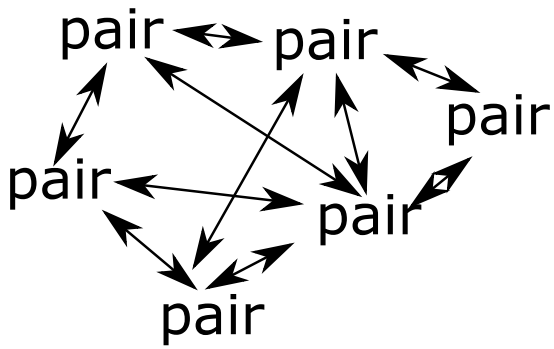


Architecture générale



Client-serveur vs pair à pair





- Architecture haut-niveau
- Client-serveur = patron de conception classique :
  Le client (usager ou navigateur) demande ;
  Le serveur attend les questions et répond au client ;
  Un serveur sert de nombreux clients indépendants.

Universal Resource Identifier/Locator

Ressource

- Ressource : entité hébergée par le serveur
- Le client interagit sur des représentations des ressources
- Ce qui constitue la ressource effectivement : fichier (web statique), résultat de l'exécution d'un programme (web dynamique)

URI/URL

Permet d'identifier et de trouver une ressource.

- extensible (nouveaux types de documents, nouveaux protocoles)
- complet
- "imprimable"

URL

protocol://[*user*[:*passwd*]@]*host*[:*port*]/*local-path*

protocol : ftp, file, http, mailto, news, nntp...
host : nom de machine
port : optionnel, dépend du protocole
local-path : dépend du protocole

http://*hostname*/*path-to-resource*
http://*hostname*/*path-to-resource*#*fragment*
http://*hostname*/*path-to-resource*?*param-list*

Exemples :

http://queinnec.perso.enseeiht.fr/Ens/ens-so.html
http://www.foo.fr/bar?param1=val&p2=val1,val2&p3=val3

Protocole HTTP

Hypertext Transfer Protocol

- Protocole textuel, sans état : ouverture de la connexion, envoi de la requête, réponse, fermeture de la connexion
- Port par défaut : 80

Requête

type␣*url*␣HTTP/1.1 CRLF
[*motclef*:␣*valeur* CRLF]*
CRLF
[*contenu textuel*]

Réponse

HTTP/1.1␣*code-de-retour*␣*raison* CRLF
[*motclef*:␣*valeur* CRLF]*
CRLF
[*corps du document*]

Type de requête (méthode)

GET : demande la récupération de la représentation d'une ressource

HEAD : idem GET, mais sans le corps éventuel

PUT : fournit une répresentation pour une ressource, à gérer par le serveur

POST : soumission de données, pouvant créer une nouvelle ressource ou modifier une ressource existante

DELETE : demande d'effacer la ressource spécifiée

En-têtes

- Requête
  - `Host` : nom DNS du serveur
  - `From` : adresse e-mail (POST)
  - `If-Modified-Since` : date
  - `User-Agent` : chaîne décrivant le client
  - `Accept` : `image/gif, image/jpeg` ...
  - `Accept-Language` : `fr, en` ...
- Réponse
  - `Location` : URL (redirection)
  - `Server` : chaîne décrivant le serveur
- Entité (corps d'un message)
  - `Allow` : méthodes supportées par l'URL
  - `Content-Encoding` : x-gzip, x-compress
  - `Content-Length` : nombre d'octets du corps
  - `Content-Type` : type de média
  - `Last-Modified` : date

Code réponse

- 1xx informationnel
- 2xx succès
  - 200 : ok, document renvoyé
  - 201 : créé (post accepté)
  - 204 : ok, mais pas de contenu
- 3xx redirection
  - 301 : déplacé permanent
  - 303 : déplacé temporaire
  - 304 : pas modifié
- 4xx erreur client
  - 400 : erreur de syntaxe
  - 401 : non autorisé
  - 403 : interdit
  - 404 : non trouvé
- 5xx erreur serveur
  - 500 : erreur interne
  - 501 : requête non implantée
  - 503 : service temporairement indisponible

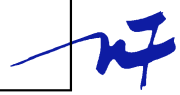
Exemple – get ok

```
posets> telnet queinnec.perso.enseeiht.fr 80
```

```
GET /Ens/ens-so.html HTTP/1.1
User-Agent: telnet
Host: queinnec.perso.enseeiht.fr
```

```
HTTP/1.1 200 OK
Date: Mon, 12 Mar 2012 12:43:24 GMT
Server: Apache/2.0.55 (Ubuntu)
Last-Modified: Mon, 14 Nov 2011 14:20:13 GMT
Content-Length: 9216
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="fr">
...
</html>
```



Exemple – get not found

```
posets> telnet queinnec.perso.enseeiht.fr 80
```

```
GET /foo HTTP/1.1
Host: queinnec.perso.enseeiht.fr
```

```
HTTP/1.1 404 Not Found
Date: Mon, 12 Mar 2012 13:09:51 GMT
Server: Apache/2.0.55 (Ubuntu)
Content-Length: 201
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /foo was not found on this server.</p>
</body></html>
```

Exemple – get not implemented

```
posets> telnet queinnec.perso.enseeiht.fr 80
```

```
LIRE /foo HTTP/1.1
Host: queinnec.perso.enseeiht.fr

```

```
HTTP/1.1 501 Method Not Implemented
Date: Mon, 12 Mar 2012 13:13:39 GMT
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Length: 209
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>501 Method Not Implemented</title>
</head><body>
<h1>Method Not Implemented</h1>
<p>LIRE to /foo not supported.<br /></p>
</body></html>
```

Gestion de session

Au niveau serveur, une session sert à repérer les connexions émises par l'instance particulière d'un navigateur.
Nécessaire pour la customisation, le suivi de connexion, le "tracking"...
Non géré au niveau HTTP.

Principe

1. Créer un identifiant au début de la session au niveau serveur
2. Le serveur donne l'identifiant au client
3. Le client rappelle l'identifiant dans les requêtes suivantes
4. L'identifiant sert à indexer des ressources au niveau du serveur.

Cookies

Le protocole HTTP est sans état.

Comment guider et suivre un utilisateur pour une succession de requêtes ?

- Adresse IP de la machine d'origine ?
  Non : partage multi-utilisateurs, proxy
- Mettre l'information de suivi dans l'URI ?
  Non : pollution des URI, caches
- $\Rightarrow$ cookies

Cookie

Cookie

Suite d'information envoyée par un serveur HTTP à un client, et stocké sur le client

- Positionné par le serveur dans l'en-tête :

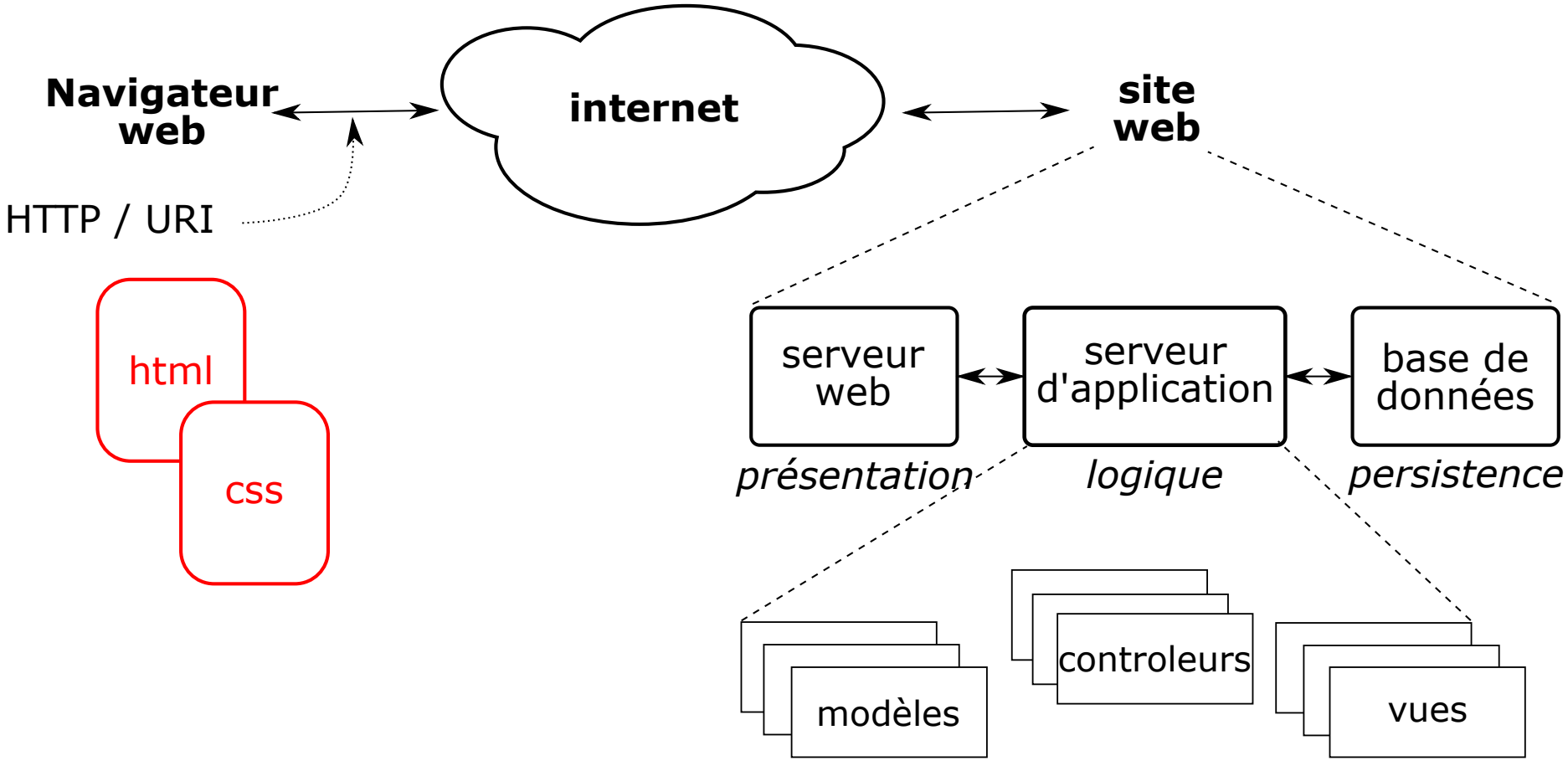
```
Set-Cookie: nom=valeur; expires=date; path=/; domain=.exemple.org
```

- Stocké par le client
- Renvoyé par le client pour le domaine et le chemin spécifié :

```
Cookie: name=value
```

- Soumis à la bonne volonté du client
- Facilement interceptables (sécurité)

Présentation de contenu



Exemple

```
<h1>Introduction</h1>
<p><i>HTML</i> est un
format de structuration du
texte. Il contient:</p>
<ul>
<li>des éléments</li>
<li>des commentaires</li>
</ul>
....
```

Introduction

HTML est un format de structuration du texte. Il contient :

- des éléments
- des commentaires

...

HyperText Markup Language

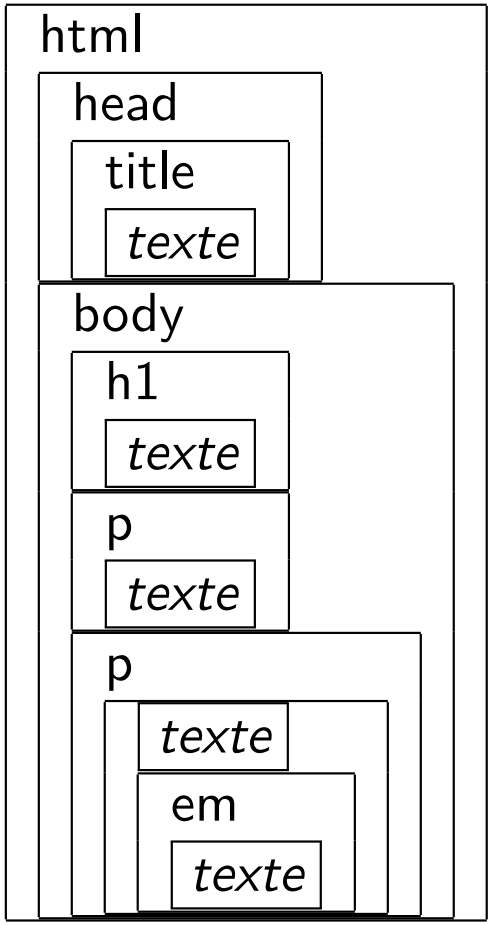
- Descendant de SGML (Standard Generalized Markup Language)
- Proche de XML
- Document = collection hiérarchique d'éléments
  - inline (titres, tables, listes...)
  - embedded (images, JavaScript code...)
  - formulaires : interaction simple avec l'utilisateur (saisie de texte, boutons, menus...)
- Tout élément peut avoir des attributs (généralement optionnels) et certains éléments peuvent avoir du contenu.
  - Attributs d'intérêt pour l'apparence : `id` et `class`
  - `id` : identification unique d'un élément
  - `class` : classification logique d'éléments
  - un élément peut avoir plusieurs classes, une classe peut contenir plusieurs éléments
  - ex : `<p id="intro" class="important encadre">`

HTML structure d'un document

Source HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Exemple de document</title>
</head>
<body>
  <h1>Premier titre</h1>
  <p>Du texte</p>
  <p>Du texte <em>important</em></p>
</body>
</html>
```

Modèle (Document Object Model)



Cascading Style Sheets

- Important : le document HTML ne devrait contenir aucune information d'apparence.
- L'apparence visuelle d'un document html est décrit dans un autre document
  - accessibilité
  - variété des terminaux (postes fixes vs mobiles)
  - séparer le travail de concepteur graphique de celui de fournisseur de contenu
- Cascade ($\sim$ héritage) : un style s'appliquant à un élément s'applique à tous les éléments qu'il contient

CSS exemple

```
/* Défaut */
body { color:black; background:#EEEEEE; }

/* Titre de niveau 1 centré et en rouge. */
h1 { text-align: center; color: red; }

/* Titre de niveau 2 en bleu sauf ceux de la classe 'enavant' */
h2 { color: blue; }
h2.enavant { color: orange; }

/* Les listes contenues dans des listes en texte plus petit
   que le normal. */
li li { font-size: smaller; }

/* Tout élément de la classe copyright en petit texte. */
.copyright { font-size:small; }
```

CSS en pratique

Désignation d'une feuille de style

Dans l'élément `<head>` du document :

```
<link rel="stylesheet" href="http://.../mystyle.css"/>
```

Contenu d'une feuille de style

La feuille de style spécifie l'apparence pour :

- toutes les instances d'un élément donné :
  `h1 { text-align:center; color:red; background:#EEEEEE }`
- tous les éléments d'une classe donnée :
  `p.important { font-style:italic; color:red }`
- un élément particulier identifié par son `id` :
  `div#auteur {font-weight:bold;text-decoration:underline;}`
- sélection arbitrairement précise (tous les éléments de la classe `cher` sous l'élément `ul` d'identité `ingredients`) :
  `ul#ingredients .cher { color:red }`

Javascript

Langage de programmation universellement reconnu par les navigateurs, embarquable dans une page web.
Il s'exécute dans le navigateur du client.
Souvent utilisé pour réagir à un événement :

- appui sur un bouton
- survol d'une zone
- fin du chargement de données (page, images, autres)

Il permet :

- de changer l'apparence du document (changement de style)
- de changer ou construire le document (changement du DOM)
- d'interroger le serveur et de récupérer des données

$\Rightarrow$ document dynamique

Plan

Contenu statique

Autrefois, un site web était constitué de pages statiques, stockées comme des fichiers.
Un serveur web est alors simplement :

1. attendre les connexions socket sur 80
2. créer un thread ou un processus à chaque connexion
3. lire la requête HTTP, extraire l'URL `path-to-resource`
4. ouvrir le fichier `server-root/path-to-resource`
5. le renvoyer

Mais actuellement, la majorité des pages sont créées à la volée par un programme.

Implantation d'un serveur web (1)

AUCUN CONTRÔLE D'ERREUR !

```
int main ()
{
    struct sockaddr_in soc_in;
    int s;
    s = socket (AF_INET, SOCK_STREAM, 0);
    soc_in.sin_family = AF_INET;
    soc_in.sin_addr.s_addr = INADDR_ANY;
    soc_in.sin_port = htons(4000);
    bind (s, &soc_in, sizeof(soc_in));
    listen (s, 1);
    while (1) {
        int f = accept (s, NULL, NULL);
        handle_request(f);
    }
}
```

Implantation d'un serveur web (2)

AUCUN CONTRÔLE D'ERREUR !

```
void handle_request (int sock)
{ char l[512]; char c;
  int i = 0; int fdrep;
  /* Lire la première ligne */
  while ((read(sock, &c, 1) != -1) && (c != '\n')) l[i++] = c;
  l[i] = 0;
  if (strncmp(l, "GET ", 4) != 0) {/* erreur */};
  *(strchr(l+4, ' ')) = 0;   /* GET filename HTTP/1.1 */
  fdrep = open(l+4, O_RDONLY);
  if (fdrep == -1) { /* Erreur : inconnu ou interdit ou... */
      write(sock, "HTTP/1.1 400 Not Found\n\nNot Found\n", 33);
  } else { /* Ok : renvoyer le fichier */
      write(sock, "HTTP/1.1 200 Ok\n\n", 17);
      while (read(fdrep, &c, 1) != 0)
          write(sock, &c, 1);
  }
  close(fdrep); close(sock);
}
```

Contenu dynamique – côté serveur

Basiquement, page html avec du code embarqué (p.e. PHP, JSP). Lors d'une requête, le code est isolé et exécuté par le serveur, ce code émet du HTML, le code est remplacé dans la page par le HTML qu'il produit, et le résultat est envoyé au client.

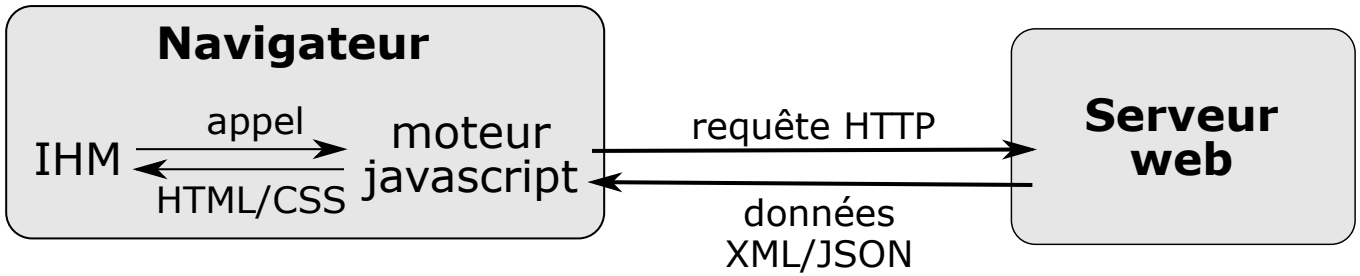
```
<html>
<h1>Titre</h1>
<p>
<?php
  if (strtolower($_POST['lang']) === 'fr') echo "Bonjour";
  else echo "<b>Hello</b>";
?>
et aussi du contenu statique.</p>
</html>
```

Illisible, peu maintenable $\Rightarrow$ isoler la logique applicative de la présentation.

Contenu dynamique – côté client

AJAX Asynchronous JavaScript And XML

Construction dynamique de la page, côté client, par une suite de requêtes (éventuellement asynchrones) au serveur.

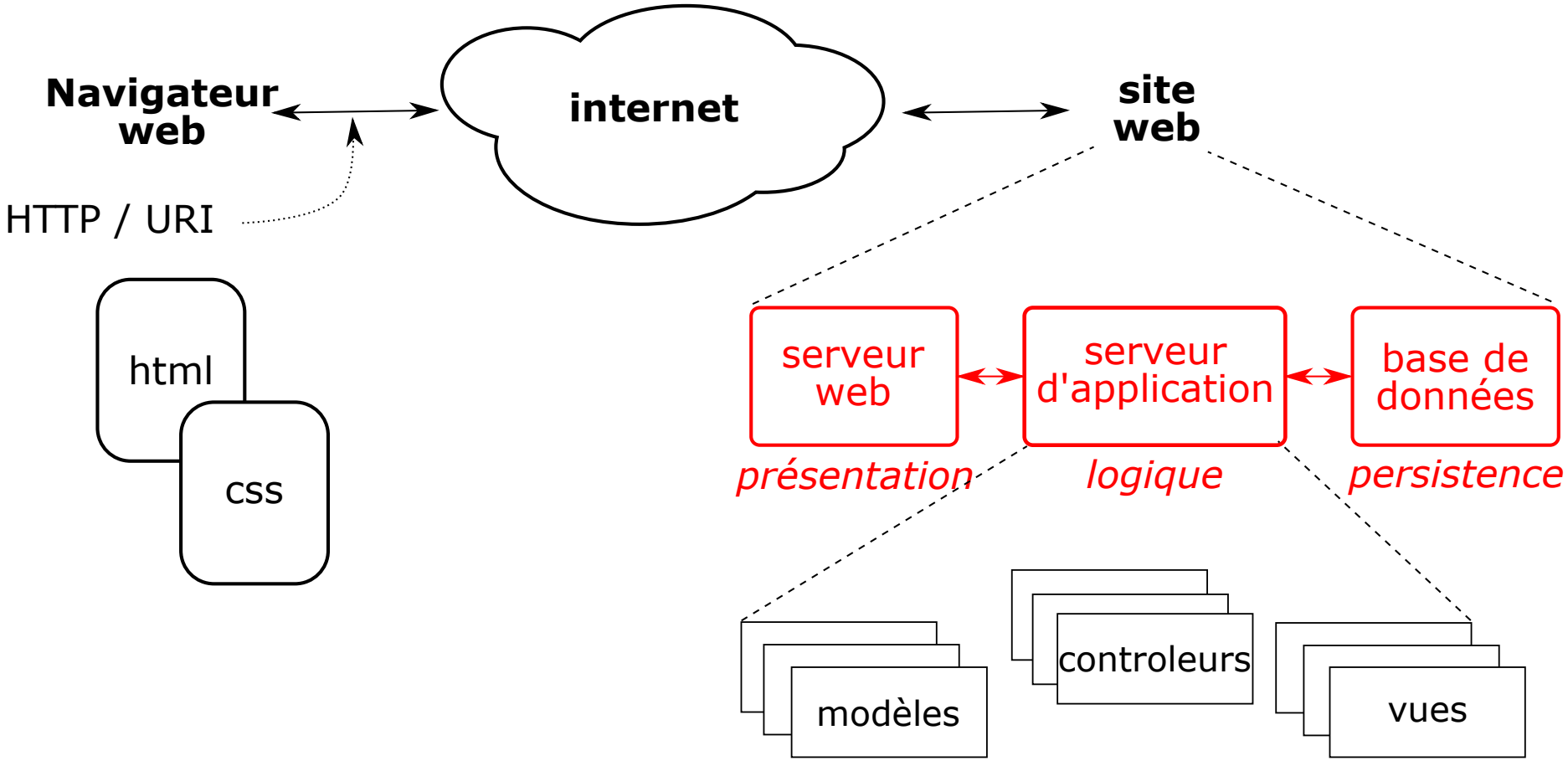


Au chargement de la page ou sur action de l'utilisateur :

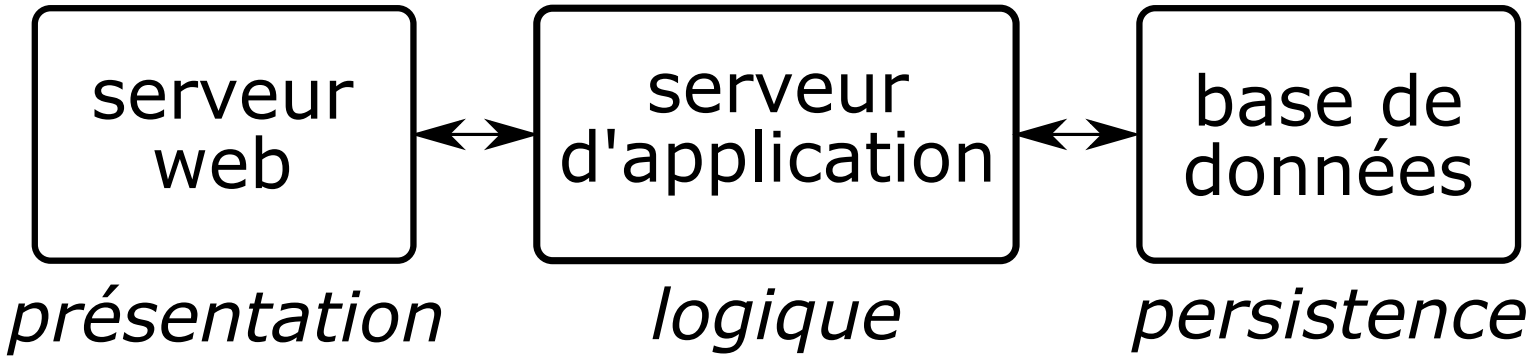
1. Le code javascript construit et envoie une ou plusieurs requêtes au serveur,
2. Le serveur répond des données (format XML ou JSON),
3. De manière asynchrone, le code javascript construit des morceaux de la page (manipulation du DOM de la page),
4. qui sont affichées à l'écran.

$\Rightarrow$ bonne interactivité, coût des échanges + coût de la construction pour le navigateur

Architecture *3-tier*



Framework

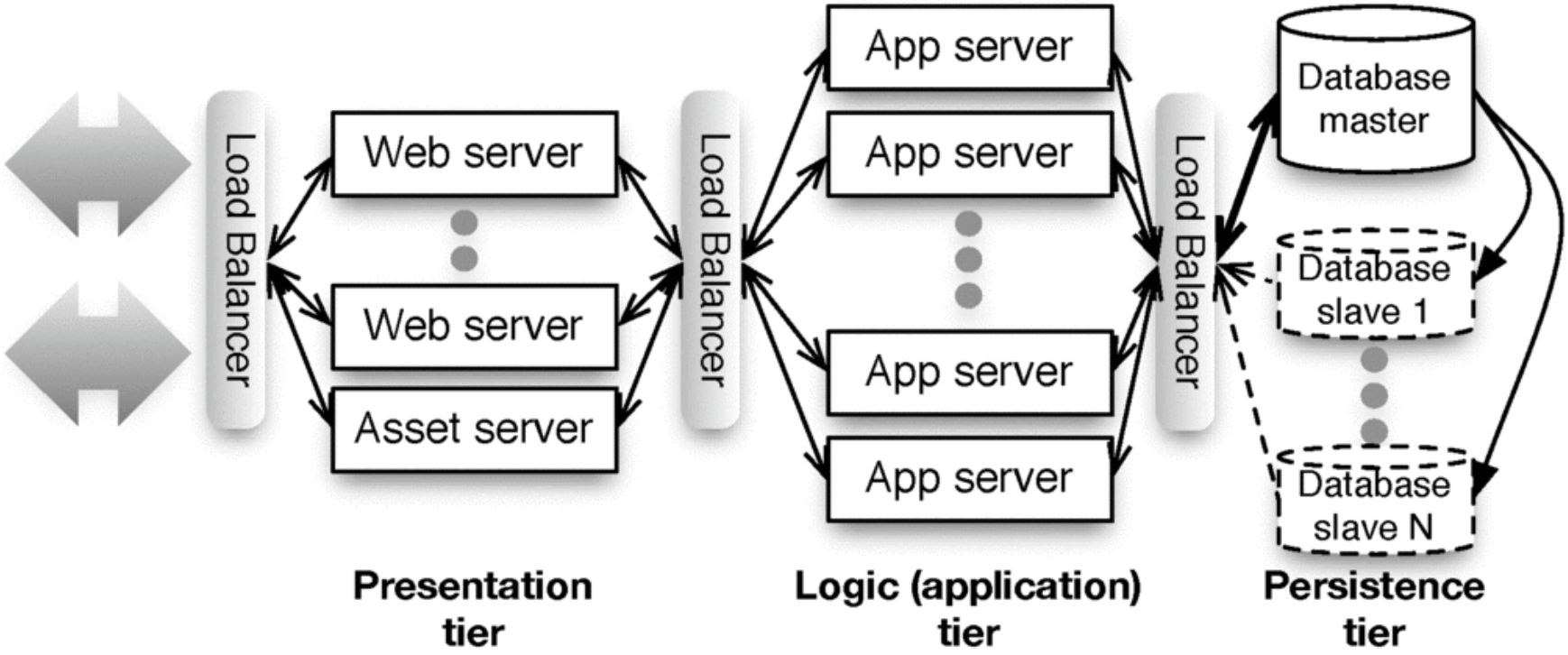


Comment

- faire correspondre un URI avec un programme et ses arguments ?
- transmettre les arguments ?
- activer le programme ?
- gérer les données persistantes ?
- gérer les informations de sessions (cookies) ?
- gérer les erreurs ?
- construire la page résultat ?

$\Rightarrow$ *frameworks*

Architecture *3-tier scalable*



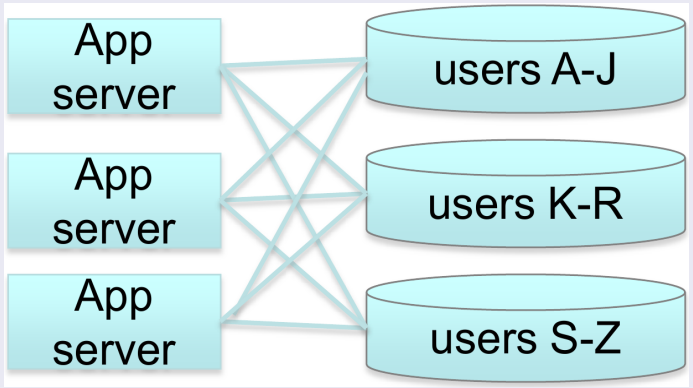
Source : cours « Software Engineering for Software as a Service », Berkeley

Base de données : répartition de charge

Partionnement horizontal (sharding)

Partitionner les données

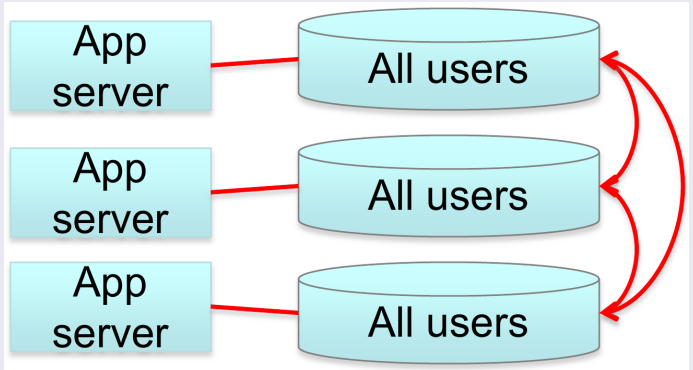
+ passage à l'échelle
− performance si opérations sur plusieurs tables/données



Réplication

Répliquer les données

+ requêtes complexes rapides
− performance des écritures ⇒ incohérence temporaire



Figures : cours « Software Engineering for Software as a Service », Berkeley

Lien Modèles - données persistantes

Correspondance entre objet en mémoire (p.e. objet java avec attributs) et stockage persistant (p.e. ligne dans une BD).
Opérations élémentaires génériques : CRUD (Create, Read, Update, Delete).

Base de données relationnelle

1 type de modèle = 1 table

- 1 ligne dans la table = 1 instance de modèle
- chaque colonne = valeur d'un attribut du modèle
- nécessité d'une clef primaire unique

id	titre	année	durée
8	2001, l'odyssée de l'espace	1968	3h04
10	Barry Lyndon	1975	2h21
11	Shining	1980	2h22
...	...	...	...

Schéma

Schéma de la BD = l'ensemble des tables et leur structure.

NoSQL – stockage structuré

- Stockage simple, type clef–valeur, structuré en arbre ou tables de hachage réparties
- Pas de tables, pas de requêtes complexes (jointure), pas de schéma à respecter
- Pas de propriétés transactionnelles (ACID – atomicité, cohérence, isolation, durabilité)
- Performant, *scalable*