Module 01 : Installation et premiers pas en PHP, rappels sur le Web

Portrait général de PHP

- PHP est **utilisable sur tous les OS les plus courants** (Windows, macOS, GNU/Linux, UNIX);
- PHP supporte de nombreux paradigmes de programmation : **procédural**, **orienté objet**, **fonctionnel**;
- PHP supporte **toutes les bases de données** (API `PHP Data Objects (PDO)`);
- PHP supporte et permet d'utiliser de nombreux **protocoles** de l'internet (HTTP, SMTP, FTP, etc.);
- Le moteur de PHP (*virtual machine*) dispose de nombreuses extensions pour étendre ses fonctionnalités;
- PHP dispose d'un gestionnaire de dépendances et de dépôts associés;
- PHP dispose de **plusieurs modes d'exécution configurables** (*SAPI*) pour le web : (mode CGI (cli), module Apache (`mod_php`, *déprécié*), PHP-FPM, FrankenPHP) le petit nouveau !

D'où vient PHP ?

- 1994 : PHP est crée par Rasmus Lerdorf (Groenland) comme une bibliothèque logicielle écrite en C pour faciliter le développement de son site web (*Personal Home Page*). PHP passe rapidement open source;
- 1997 : PHP3 est crée (refonte totale du moteur). Développé par Gutmans et Suraski, moteur Zend. Maintenu et supporté par la PHP Foundation et Zend;
- Novembre 2025 : PHP 8.5, version actuelle de PHP !

Plusieurs implémentations de PHP

- Zend Engine : une implémentation de PHP open source, par Zend Technologies. C'est celle que l'on va utiliser. Essayer `php -v` pour afficher la version de PHP et du Zend Engine;
- Hip Hop Virtual Machine for PHP (HHVM) : développé par Méta/Facebook, avec le langage Hack, un dialecte PHP fortement typé

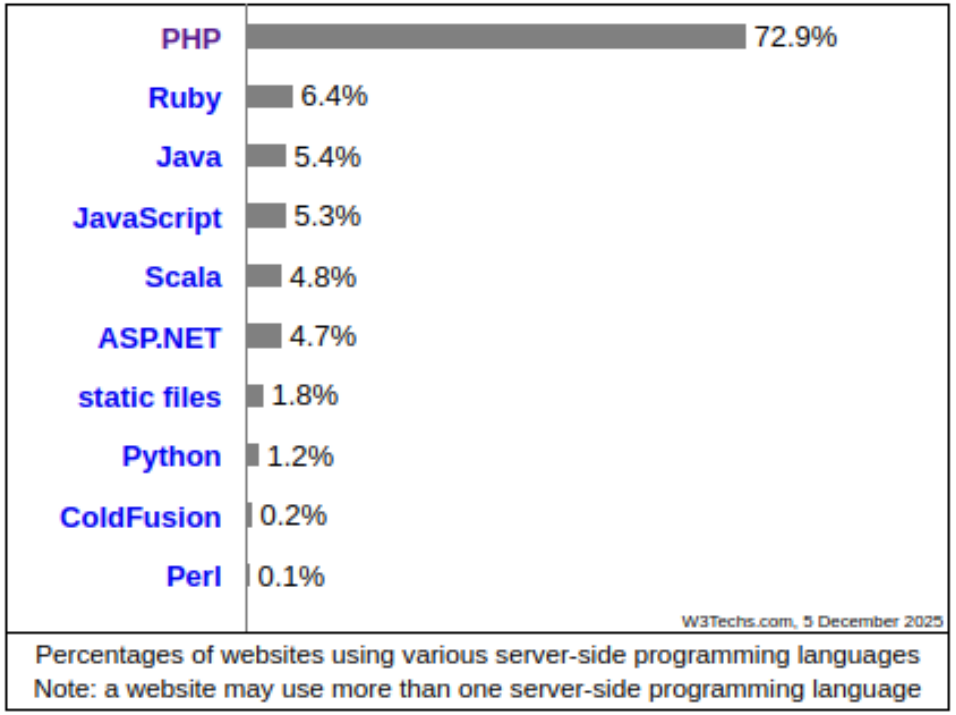
PHP dans le monde réel

{.marp-bg-img}

> PHP fait tourner le web mondial !

- En 2025, environ **73% des sites web** recensés sont *powered* par PHP (avec Wordpress notamment);
- PHP est en constante évolution et maturation. Écosystème **riche**, communautés **actives** :
  - PHP Framework Inter Group (PHP-FIG),
  - PHP Foundation,
  - AFUP,
  - Nombreux frameworks : Symfony, Laravel, API Platform, MediaWiki, **WordPress**, WooCommerce, bbpress, phpBB, etc.

> Source w3techs survey



Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

Qui utilise PHP ?

- Wikipédia ! Avec le moteur de wiki MédiaWiki, écrit en PHP. La plupart des moteurs de Wiki sont implémentés en PHP (Dokuwiki);
- Meta (Facebook), Etsy, Spotify, Canva, Slack, etc;
- Quasiment tous les **moteurs de forum** (phpBB, bbpress, etc.);
- **Content Managment System** (CMS) ! **Wordpress**, **Drupal**, Joomla, Magento, etc.
- **Administration de base de données** (phpMyAdmin, Adminer, etc.);
- Etc.

Pourquoi utiliser PHP ?

- PHP est à la fois *facile à prendre en main* (bien pour débutant·es) et *avancé* pour faire du développement de qualité professionnelle ;
- Déployer un site web = déployer des simples fichiers ;

Installer PHP

Suivre les instructions données sur le site officiel en fonction de l'OS de votre machine.

Suivre le tutoriel officiel.

Ajouter le répertoire où se strouve php.exe (répertoire issu de l'archive) sur votre PATH.

Ouvrir une invite de commande et tester l'installation :

```
php -v
PHP 8.4.15 (cli) (built: Nov 20 2025 19:00:52) (NTS)
Copyright (c) The PHP Group
Built by Debian
Zend Engine v4.4.15, Copyright (c) Zend Technologies
    with Zend OPcache v8.4.15, Copyright (c), by Zend Technologies
```

Sur macOS

1. **Installer** le gestionnaire de paquets brew ;
2. **Ajouter** le repértoire au PATH :

```
echo "export PATH=/opt/homebrew/bin:$PATH" >> ~/.bash_profile && source ~/.bash_profile
```

3. **Installer** PHP avec brew :

```
brew install php
```

Installer l'IDE Visual Studio code

- **Installer** Visualstudio Code ;
- Les fonctionnalités de VS Code peuvent être étendues via l'installation d'**extensions**. **Installer** les extensions suivantes :
  - PHP Intelephense, **indispensable**, permet d'avoir tout le **support pour le développement PHP** (formatage, documentation officielle *inline*, auto-complétion, snippets, etc.),
  - D'autres ? Pas pour l'instant...

Configurer son IDE, rendre sa vie meilleure

Dans File > Preferences > Snippets, **créer** un fichier `php.json`. Placez-y la configuration de *snippets* suivante :

```
{
 "php": {
  "prefix": "php",
  "body": [
   "<?php $0 ?>"
  ],
  "description": "php tag"
 },
 "ph": {
  "prefix": "ph",
  "body": [
   "<?php",
   "$0"
  ],
  "description": "php only one tag"
 },
}
```

> Ces snippets vous éviteront d'avoir à ouvrir et fermer des balises PHP à la main à chaque fois ! Testez-les.

Votre premier script PHP

Créer un fichier `hello-world.php` :

```
hello, world
```

Exécuter :

```
php hello-world.php
```

Votre premier script PHP, avec du code php

- Le code PHP est **toujours** inclus entre une **balise ouvrante** `<?php` et une **balise fermante** `?>`. Voici le contenu d'un fichier `index.php` :

```
<html>
    <body>
        <?php echo "hello, world"; ?>
    </body>
</html>
```

- Toute instruction PHP **se termine par un point-virgule** :

```
<?php echo "hello, world"; ?>
```

- **Exéxuter** le script `index.php` :

```
php index.php
```

- **Observer** la sortie.

Que s'est-il passé ?

Un script PHP s'exécute sur la *Machine Virtuelle* (VM) ou *interpréteur* PHP. Il analyse le fichier *byte par byte* :

- Tant qu'il ne rencontre aucune balise PHP, les caractères sont considérés comme des **données brutes** et sont écrits sur la sortie standard (*stdout*) **sans être interprétés**;
- Lorsque l'interpréteur rencontre une balise ouvrante (`<?php`), il passe en **mode interprétation**. Le code jusqu'à la balise fermante (`?>`) est alors :
  - **Analysé** (*Tokenizing, Parsing*): Le **code est vérifié** pour sa syntaxe et **transformé** en une séquence d'instructions appelée Abstract Syntax Tree (AST),
  - **Compilé** (en *bytecode*): L'AST est ensuite **compilé** en **bytecode** appelé **Opcode**,
  - **Exécuté**: Le moteur d'exécution Zend **exécute** l'**Opcode**. Résultat écrit sur *stdout*.

PHP est spécialement adapté (conçu pour à la base) pour produire des documents

Grâce à son fonctionnement, PHP est parfaitement adapté pour **fabriquer des documents dynamiques** (contenu dynamique placé entre balises PHP) :

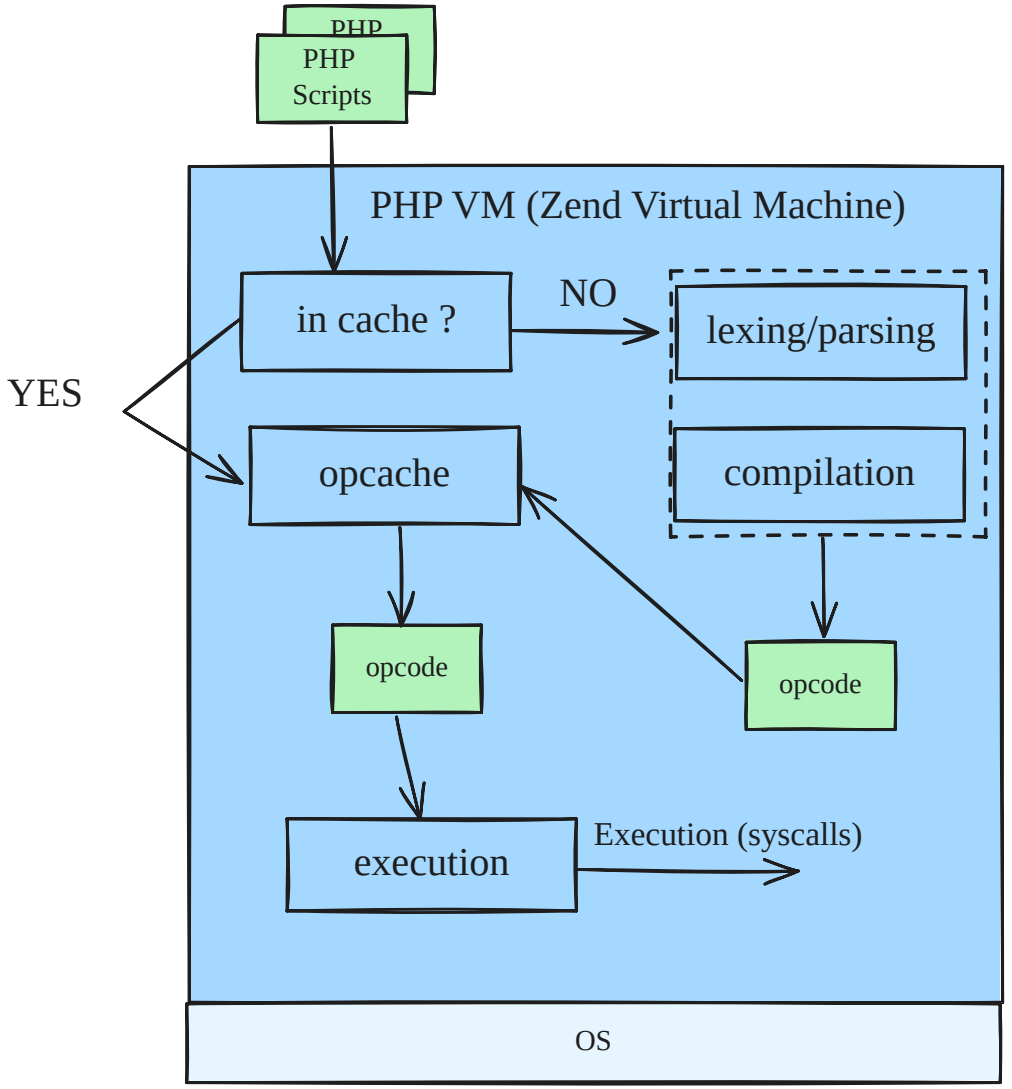
- Fichiers texte ASCII,
- **Pages HTML** (moteur de templates de site web),
- PDF,
- **Images** (PNG, SVG, etc.),
- Données structurées pour des services web (JSON),
- Documents XML,
- Email,
- Etc.

> N'importe quel format !

Modèle d’exécution

{.marp-bg-img}

- Chaque **script** s'exécute dans son **propre processus ou contexte** ;
- À la **fin** de **l’exécution**, toutes les variables, objets et **ressources allouées** sont automatiquement **libérées** ;
- Dans un contexte web, il n’y a **pas de mémoire partagée entre les requêtes** (nécessite un cache, fichiers, session ou **une base de données**).



PHP VM (Sans JIT)

Machine virtuelle PHP

La machine virtuelle PHP, dont le cœur est le Zend Engine, est conçue pour être à la fois hautement **configurable**, **modulaire** et **adaptable à différents contextes d'exécution**.

La machine virtuelle PHP :

- se **configure** via des **fichiers INI** (`php -i` pour afficher configuration actuelle) ;
- est **étendue** par un grand nombres d'**extensions** (modules) (`php -m` pour lister modules installés) ;
- possède **différentes API**, *Server Application Programming Interface (**SAPI**)*, intégration avec l'environnement hôte. Détermine l'architecture d'exécution, entrée/sorties, la configuration et la **gestion des processus** de la VM PHP :
  - **CLI**/CGI (**utilisé ici**),
  - **Module Apache**, *déprécié* mais utilisé pour le dev (stack *AMP),
  - *FastCGI Process Manager* (PHP-FPM) (standard actuelle en prod),
  - FrankenPHP, le petit nouveau (développé par Kevin Dunglas).

Configurer PHP au *runtime* (comportement à l'exécution) 1/2

Via des fichiers `php.ini` (format INI).

Dans un script, pour afficher la configuration complète de la VM PHP, **utiliser** la fonction `phpinfo()` :

Créer un fichier `index.php` avec le contenu suivant :

```php
<?php
//Afficher toute la configuration de PHP. Très pratique pour voir les fichiers ini chargés notamment,
//si l'upload de fichiers est autorisé, quelle taille max, etc. (cf WordPress)
phpinfo();
```

Configurer PHP au *runtime* (comportement à l'exécution) 2/2

Ou **directement** depuis le terminal :

```
# Affiche toutes les informations (idem à phpinfo())
php -i
# Liste tous les fichiers de configuration utilisés
php --ini
# Avec PHP 8.5 ! Affiche directement les valeurs qui diffèrent des valeurs par défaut.
php --ini=diff
```

Il existe de nombreuses directives pour configurer PHP à l'exécution, utiles notamment pour adapter le comportement en fonction des environnements et du site web.

> On reviendra pus tard dans la formation sur la configuration de PHP (*dev* vs *prod*), petites choses à savoir pour votre site WordPress...

PHP (CLI)

Exécuter directement le script dans le terminal :

```
php votre-script.php
```

La sortie (résultat) sera affichée dans le terminal (*sortie standard*).

Configuration personnalisée (fichier .ini)

Créer un fichier de configuration personnalisé `config.ini` dans le répertoire courant :

```
;Active la prise en charge des uploads de fichiers via HTTP.
file_uploads = On
;Limite la mémoire maximale qu'un script PHP peut consommer. Protège contre les scripts trop gourmands.
memory_limit = 64M
;Taille maximale acceptée pour l'ensemble du corps d'une requête POST. Doit être >= à upload_max_filesize.
post_max_size = 64M
;Taille maximale d'un fichier individuel envoyé via HTTP POST.
upload_max_filesize = 64M
```

Inspecter les configurations appliquées :

```
php -c config.ini --ini=diff
```

Utiliser une configuration personnalisée

Exécuter directement le script dans le terminal, en **chargeant le fichier de configuration** :

```
php -c config.ini index.php
```

Serveur web intégré de PHP

PHP dispose d'un **serveur web intégré**, utilisant toujours la SAPI CLI, comme le terminal.

Ce serveur web local vous permet de **lancer et développer un site web** comme si vous interrogiez un *vrai* site web (via HTTP)

Très utile pour le développement (et pour nous !)

> Ne pas utiliser en prod !

Préparons notre premier site web

Dans le fichier `index.php`, **placer** le code suivant :

```
<!DOCTYPE html>
<html lang="fr">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Mon site web</title>
</head>
<body>
 <h1>Notre premier site web</h1>
 <p>Nous sommes le <?php echo date( 'd/m/Y' ); ?> et il est <?php echo date( 'H:i' ); ?></p>
</body>
</html>
```

Utiliser le serveur web intégré de PHP

1. **Ouvrez** un terminal/invite de commandes ;
2. **Placez-vous** à la racine du projet à *servir comme un site web* (au même niveau) :

```
cd chemin/des/fichiers/php/a/servir
# Servir tout le contenu du dossier courant (attend un fichier index.php !)
php -S localhost:3000
# Alternative : servir le dossier 'public' (contient le site)
php -S localhost:3000 -t public
```

3. Avec votre navigateur favori, se rendre à l'URL http://localhost:300.

> Le serveur web intégré de PHP **attend un fichier `index.php` par défaut**. Penser à toujours utiliser un fichier `index.php` comme point d'entrée de votre site web.

Que s'est-il passé ?

1. Avec votre navigateur, vous avez **demandé** la **ressource** racine(`/`) à l'URL http://localhost:3000 ;
2. Votre navigateur a **envoyé une requête HTTP** `GET /` ;
3. Le serveur web intégré de PHP a reçu votre requête et **a exécuté** le script `index.php` **associé à la ressource demandée**(comportement par défaut) ;
4. Le script a **écrit son résultat** sur la *sortie standard* ;
5. Le serveur web a **récupéré ce résultat** (une chaîne de caractères, du texte !), l'a **placé dans le corps d'une réponse HTTP** et l'a retourné au navigateur ;
6. Le navigateur, à réception de la réponse HTTP, a **chargé le résultat** (le code HTML) puis l'a **peint** (pixels) sur l'écran ;
7. Vous **consultez** la page web !

Guide de survie dans le *shell* (terminal ou Invite de Commandes Windows)

> Inspecter et se déplacer

Action	UNIX (GNU/Linux, macOS)	Windows
Afficher où je suis (répertoire *courant*)	`pwd`	`cd`
Afficher le contenu du répertoire courant	`ls`	`dir`
Se déplacer	`cd /foo/bar`	`cd C:\foo\bar`

Pratique

1. **Ajoutez une feuille de style CSS** à votre site web pour ajouter de la mise en forme (couleurs, marges, fonts, etc.).
2. **Ajoutez** une (ou plusieurs) image(s) sur votre page web. Pour cela, **créer** un dossier `img` et **placez**-la dedans. Utilisez l'élément HTML `<img>`.
3. L'heure affichée n'est pas correcte, pourquoi ? **Réglez** le problème en définissant la *timezone* correcte. Pour cela, utilisez la fonction PHP `date_default_timezone_set` ;
4. **Ajouter** une *font* personnalisée pour votre document HTML. Allez sur Google Fonts, choisir une *font* qui vous plait, **téléchargez-là** et **utilisez**-là pour le texte de votre page web en utilisant le code CSS suivant :

```css
html{
    width: 80%;
    font-family: "myFont", serif;
}

@font-face {
  font-family: "myFont";
  src: url("votre-font.ttf");
}
```

**AMP stack* : Lamp, Wamp, Mamp

Le serveur local intégré de PHP est largement suffisant pour développer des petits sites web, tester ses idées, etc.

Mais si besoin de mettre en place **une base de données**, avoir un environnement complet *clé en main*, on peut **utiliser la stack AMP* :

- **L**(inux), **W**(indows), **M**(macOS) ;
- **A**pache (Serveur web) ;
- **M**ySQL (Base de données SQL) ;
- **P**HP (configuré et utilisé par Apache);

> Pour aujourd'hui, le serveur web intégré de PHP sera suffisant !

Homework

- Pour la prochaine session, **installez la stack *AMP sur votre machine** (pour qu'on gagne du temps !)
- Suivez **l'une de ces procédures** en fonction de votre OS .

Sur Windows (WAMP)

Installez Wamp Server et **suivez le guide**.

Sur macOS (MAMP)

Installez MAMP. Utilisez la documentation au besoin.

> Pas la version pro !

Sur GNU/Linux, Ubuntu/Debian (LAMP)

À vous d'installer Apache, MySQL et PHP (deja fait !) et configurer les *Virtual Hosts* d'Apache ! Suivez ce guide

Fonctionnement du web (en bref ?)

- Qu'est-ce que le *web* ?
- Comprendre **l'architecture client/serveur** du web ;
- Comprendre le rôle du *client* et du *serveur*.

> Se construire un modèle simplifié mais *utile*.

Internet *n'est pas* le web

{.marp-bg-img}

Internet :

- **Regroupement de réseaux publics**, reliant des machines (*Inter-Networks*) ;
- Descendant du réseau ARPANET (70s, période *Guerre Froide*), conçu et développé par la DARPA, agence de défense américaine ;
- Fonctionnement **décentralisé**, basé (entre autres) sur les protocoles TCP/IP, inventés en 1973.

Internet *n'est pas* le web

> Protocole : Règles et conventions régissant l'échange de données entre ordinateurs.

- De nombreux *protocoles* et usages d'Internet :
  - UUCP (premier réseau de forums Usenet en 1980) ;
  - SMTP, protocole pour envoyer des mails, années 70 ;
  - FTP : File Transfert Protocol ;
  - **HTTP** : **Le protocole du web, inventé dans les années 90** ;
  - IRC ;
  - SSL/TLS : protocole de transaction sécurisé ;
  - Etc.
- Sur Internet, **le protocole IP attribue à chaque machine une adresse** (*adresse IP*), un numéro qui l'identifie de manière unique.

Qu'est ce que le web ? Un système hypertexte distribué

> "Un système d'information **hypertexte universel** où les enjeux de **généralité** et de **portabilité** sont considérés comme les plus importants" (Tim Berners-Lee, inventeur du web)

- **Hypertexte** : Un document contenant des données texte et **liens vers d'autres documents** (**hyperliens**) ;
- **Hypermédia** (généralisation) : Extension de l'hypertexte aux données images, sons, vidéo + capacité de prendre des *inputs* (formulaire) pour envoyer des données à d'autres hypertextes afin des les modifier ;
- **Système hypertexte** : système constitué de *noeuds* contenant des informations et des liens représentant des relations entre ces *noeuds*. Un tel système est construit sur la **connection entre ses éléments**.

L'hypertexte, une invention plus ancienne que le web

> Hypertexte : un texte qui fournit des **liens** entre **des éléments clés** qui permettent aux utilisateurs de **naviguer à travers l’information** de **manière non linéaire** (sans tout lire dans l'ordre)

- L'*hypertexte* (lien et navigation entre documents) est une *vieille idée*. Par ex. les Indexs dans les livres dès le XIIème permettent de naviguer entre les pages, chapitres *via* des mot clefs ;
- Terme *hypertext* inventé en **1965** par Ted Nelson : "concept unifié d’idées et de données interconnectées, et de la façon dont ces idées et ces données peuvent être éditées sur un écran d’ordinateur" ;
- Systèmes hypertextes longtemps cantonnés au stade de prototype, *le Web constitue l’une de leurs implémentations* et sera, lui, *largement* adopté par le grand public ;

The Mother of All Demos : Un système hypertexte (et bien plus !) déjà présenté en 1965

{.marp-bg-img}

Douglas Engelbart, un spécialiste des interactions homme-machine, fait une démonstration avec ses équipes dispersées aux USA, restée dans l'histoire : "*The Mother of All Demos*" : souris et pointeur, GUI, visioconférence, courier électronique et un **système hypertexte**.

> Cette conférence montre déjà, dès 1965, l'essentiel (voire plus) de ce que nous faisons aujourd'hui avec les machines. En termes de démonstration publique, elle préfigure exactement le type de présentations dont Steve Jobs s'est inspiré pour lancer l'iPhone. Et le film/montage a une ambiance à la *Terry Gilliam*

Surfing the web, le web est *ouvert*

- Sur le web, **n'importe quel document hypertexte peut être édité** pour pointer sur un autre document (via les *hyperliens*), **sans avoir besoin de permission ou d'autorisation** ou faire quoi que ce soit d'autre ;
- Le web offre la liberté de faire **toutes les connections possibles**, éditer les documents, laisser les autres éditer des documents (via les *formulaires*), etc.
- Tout le système (et tout son contenu généré) repose sur les **liens** et les **formulaires** (appelés *hypermédias*) pour créer des chemins, experiences et **applications sans cesse nouvelles**.

C'est grâce aux **hypermédias**, à la généralité et l'**uniformité de son interface** (et aux respect de certaines contraintes de *design*) que le web évolue depuis plus de 30 ans !

Surfing the web, le web est *simple*

Le web est un système *populaire*, car il permet à **des personnes sans connaissances techniques de réaliser des choses utiles**.

Sur le web, **un agent humain utilise les mêmes moyens qu'un agent programmable pour réaliser des actions** (clients et services web ou web API). Par ex., si vous utilisez des plateformes comme Make ou Zapier, les agents que vous programmez envoient, comme vous, des requêtes HTTP GET et POST. Ils *suivent* des liens et *soumettent* des formulaires.

> Le web *humain* est juste un (petit) "sous-ensemble" du web programmable.

Un peu d'histoire

- **1990**: Tim Berners-Lee invente le web au CERN pour **le partage de documents scientifiques**. Il invente :
  - Le protocole HTTP 1.0 ;
  - Le langage à balises HTML (structure et description du contenu);
  - Les URL ;
  - Le premier *serveur* web ;
  - Le 1er navigateur web (WorldWideWeb puis Nexus);
- **6 août 1991**: première page web publique

Un peu d'histoire 1/2

- **1993**: le web compte environ **4 millions d'utilisateur·ices**, **x2 chaque mois**, nombre de documents augmente de manière exponentielle. **Le web va s'effondrer !**
- **1993**: Roy Fiedling et al. pose le problème de *la montée en charge du web*. Spécifie/Design les **contraintes** *REST* que le web doit respecter pour **survivre à son succès** (*scaling*) : cache, système en couches, stateless, etc.
- **1994**: standardisation du protocole **HTTP 1.1** et respect des contraintes REST, avec Tim Berners Lee. Le web est *sauvé* !

Le web en quelques dates clefs (non exhaustif)

Date	Évènement
1990	Invention du web par Tim Berners-Lee, première page web en ligne, HTML 1 et HTTP 1.0
1993	Navigateur NCSA Mosaic (développé alors que le web ne contenait que 200 sites), **balise** `img`
1994	**HTTP 1.1**, Netscape Navigator (navigateur dominant dans les années 1990, supplanté par Internet Explorer), nombre de sites explose, annuaire web/moteur de recherche Yahoo!, création du World Wide Web Consurtium (W3C)
1995	Balise `table`, **création de** `JavaScript` (Netscape)
1997	HTML 4.0
1998	Le W3C crée le standard `XML` pour remplacer à terme le `HTML`, **création du** `CSS`, `XmlHttpRequest`, **Google**, **Netscape ouvre ses sources**
2000/2006	Évolutions du HTML, explosion de la bulle Internet
2001	Lancement de du projet **Wikipédia**
2004	Création du WHATWG, première version de **Firefox**
2004/2006	**E-commerce explose**, **Web 2.0**: AJAX, apparition de géants comme Amazon, Facebook, Youtube et autres réseaux sociaux
2010	**HTML5**, abandon `XHTML2`, **frameworks front-end**, CSS Grids, **Responsive web design**
2014	Le milliard de sites dépassé

Comment *fonctionne* le web ?

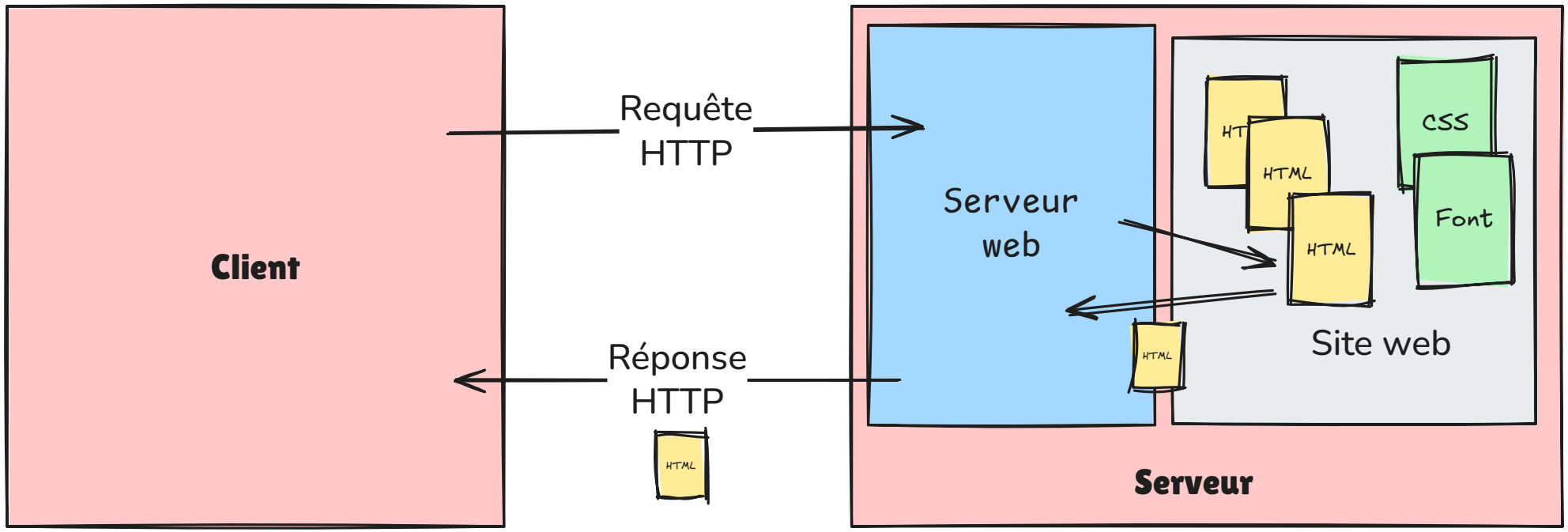
- Qu'est ce qu'un "site web" ?
- Qu'est ce qu'il se passe quand je "vais sur google.com" ?
- Qu'est ce qu'il se passe quand je clique sur "Me connecter" ou "Publier" sur un site web ?
- Ça veut dire quoi "être sur une page web" ?
- Qu'est ce qu'un *navigateur* web ?

Architecture du web : client/serveur

- Le **serveur** : **gère** et **expose** un ensemble de **ressources** (via des URL). Répond aux **demandes du client** (accepte ou refuse !) ;
- Le **client** : demande l'accès aux ressources du serveur via les hypermédias :
  - En *lecture* : consulter (balise `<a>`),
  - En *écriture* : éditer, créer, supprimer (balise `<form>`).

> Vision simplifiée mais un modèle mental utile

Architecture du web : client/serveur, modèle un peu plus détaillé



Sur quelles technologies est bâti le Web

Le web, techniquement, c'est *trois technologies* :

- Le protocole **HTTP**: protocole de communication entre un *client* et un *serveur* web. Ex: `GET / HTTP/1.1 Host: www.perdu.com` ;
- Adresses web (**URL**), hyperliens : `http://www.example.com/tim/page.html` ;
- **HTML**: langage markup. Ex: `<!DOCTYPE html><html><h1>Titre principal</h1><p>Un paragraphe</p></html>` ;

Brique 1 : Le protocole HTTP (les messages)

> Le plus important !

HyperText Transfer Protocol (HTTP) :

- **Protocole de communication du web** sous forme d'envoi d'*enveloppes*,
- Modèle **Requête/Réponse**,
- Méthodes HTTP: indiquent l'*intention* du client :
  - `GET` : "Donne moi une **copie** de la ressource",
  - `POST` : "**Modifie** l'état de tes ressources avec mes données que je t'envoie via un form"
- **Code d'erreurs**: 1XX, 2XX, 3XX, 4XX, 5XX,
- Port par défaut: `80` ;

> Concrètement c'est un format de message pour que le client et le serveur se comprennent ! Une requête HTTP 1.1 est juste une *chaîne de caractères*. Elle voyage sur le réseau Internet jusqu'à son destinataire.

Requêtes HTTP GET

{.marp-bg-img}

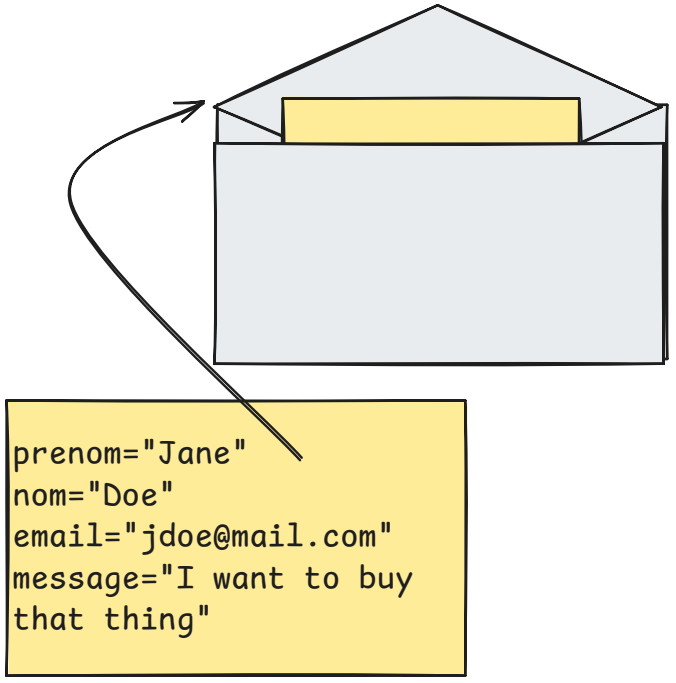
Sur le web humain, vous utilisez que deux méthodes HTTP :

`GET`

- **Intention du client** : "Donne moi une copie" (*lecture*);
- Méthode utilisée quand *on clique sur un lien* (`<a>`) ;
- Toutes les informations sont sur l'enveloppe ;
- Le contenu de l'enveloppe (le **corps** de la requête) reste *vide* ;

Requête HTTP GET

Requête HTTP POST

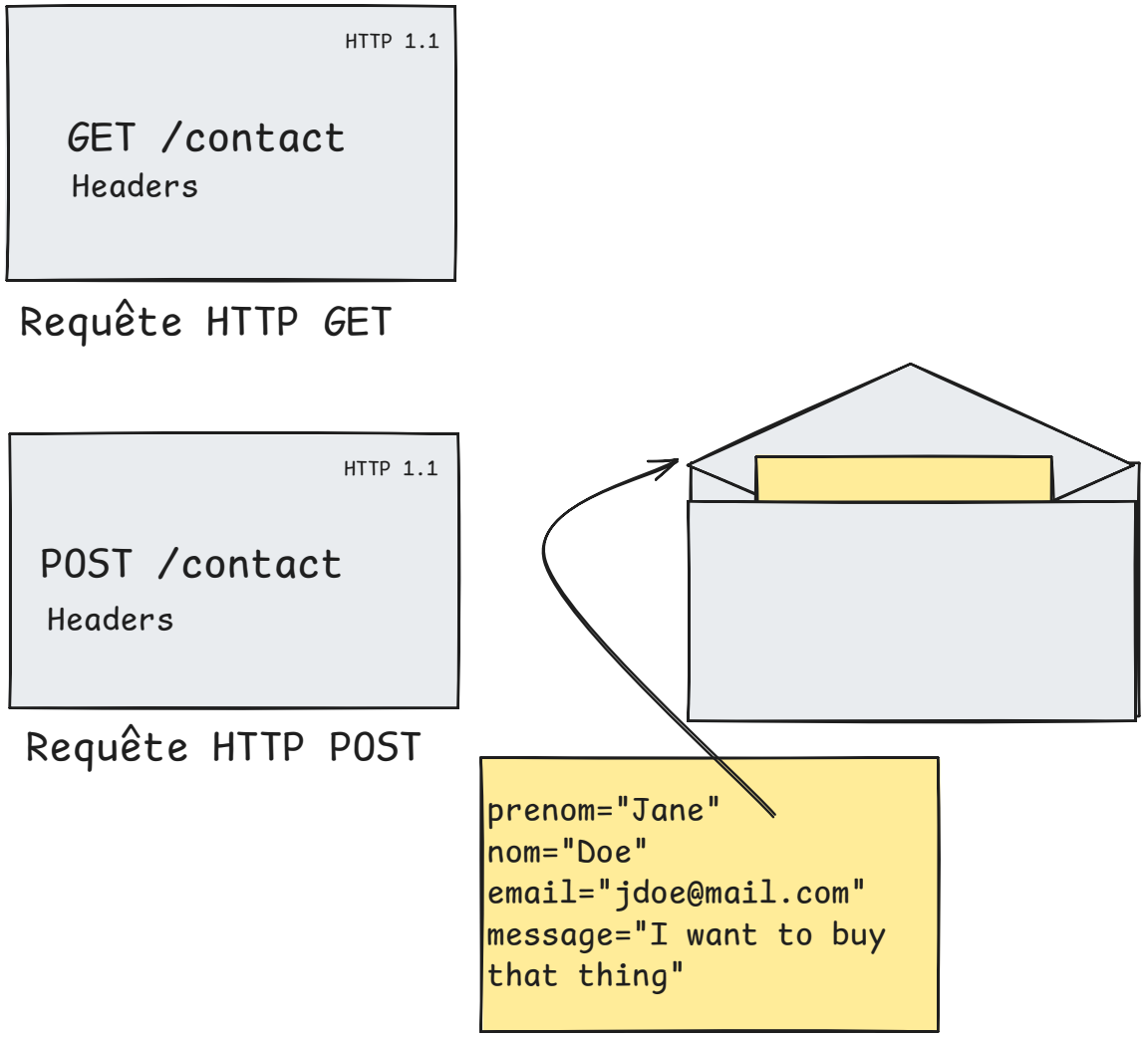


Requêtes HTTP POST

{.marp-bg-img}

POST

- **Intention du client** : "Modifie l'état de tes ressources" (*écriture*) ;
- Méthode utilisée **quand on soumet un formulaire** (`<form>`) ;
- Les informations sur l'URL soumettre le contenu du formulaire (données) sur l'envoloppe ;
- Les données du formulaire placées **dans l'enveloppe** (corps de la requête) ;
- Format des données : `name=valeur`. Le nom de chaque donnée est définie par l'attribut `name` de l'input HTML.



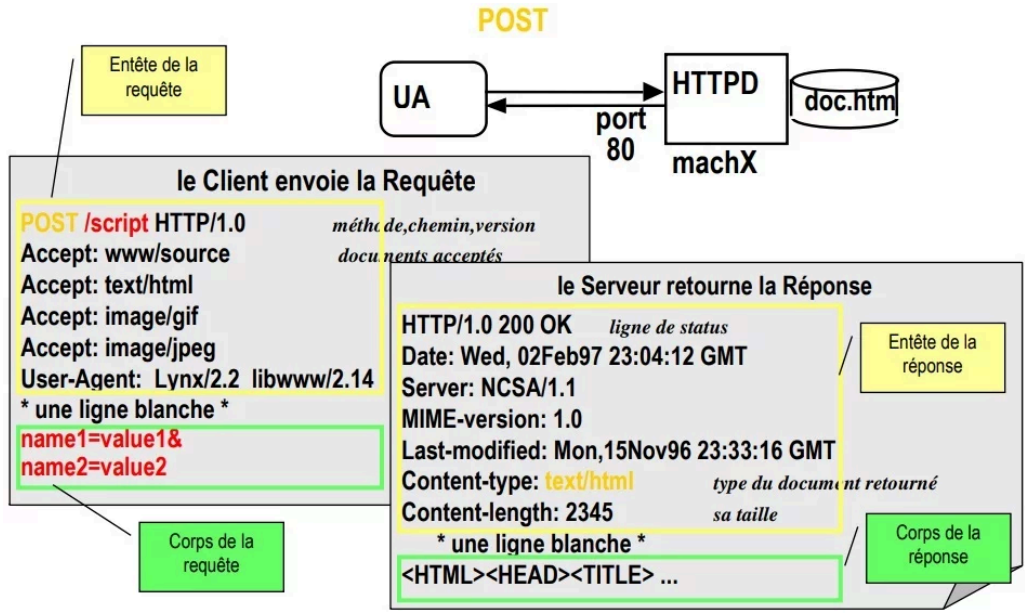
Requête HTTP

{.marp-bg-img}

Contenu d'une **Requête HTTP** (envoyée par le client) :

- **Ligne de requête au format** `{Méthode} {URL de la ressource} {Version HTTP}`
  - Méthode (verbe) : Exprime l'*intention* sur la ressource (GET ou POST) ;
  - URL : Quelle ressource ?
- En-têtes (*Headers*)
- **Corps** (les données envoyées via des formulaires HTML)

> Les Headers HTTP permettent de *moduler* la demande (préciser des choses)

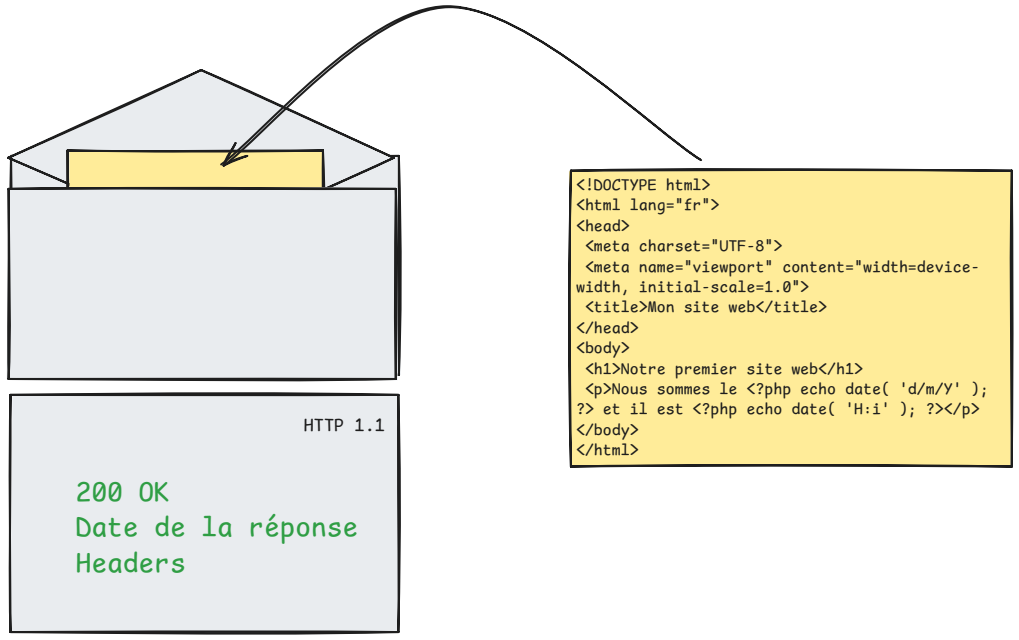


Réponse HTTP

{.marp-bg-img}

Contenu de la **Réponse HTTP** (envoyée par le serveur) :

- Sur l'enveloppe :
  - **Code de statut** (`200` pour dire OK, `404` pour dire "*Vous* vous êtes trompés !", `500` pour dire "*Nous* avons un problème") ;
  - Date et heure de la réponse ;
  - Headers ;
- Dans l'enveloppe (corps de la réponse) :
  - **Page HTML demandée** ;

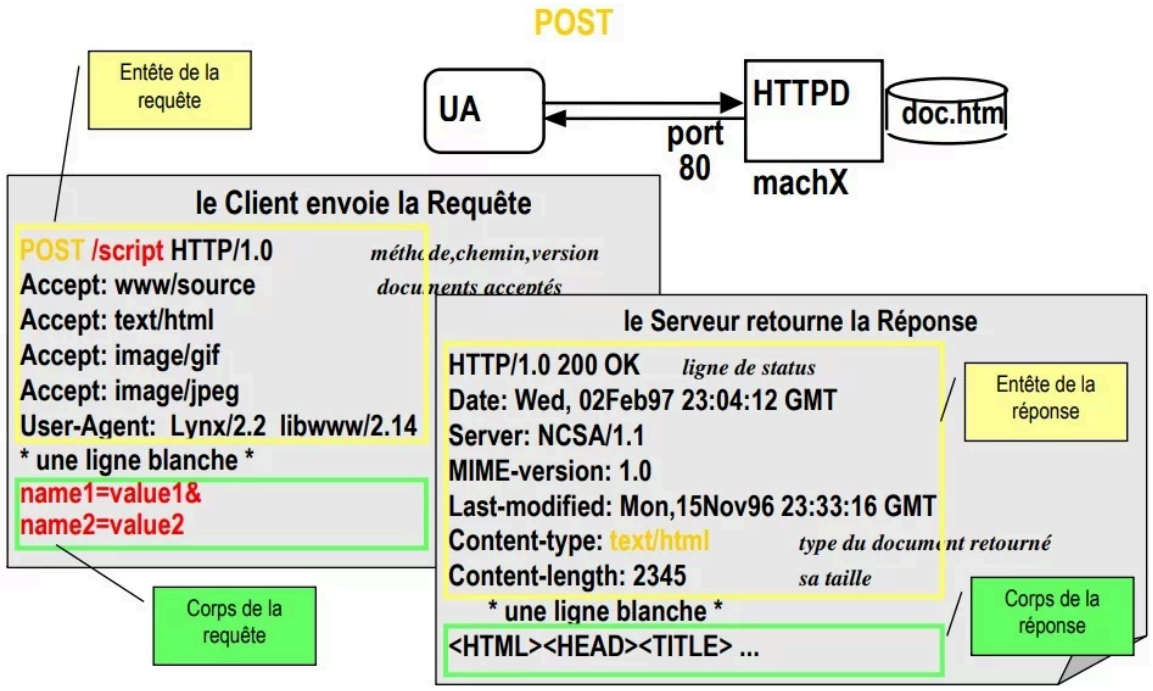


Réponse HTTP

Requête et réponse exemples plus détaillés

{.marp-bg-img}

Le serveur analyse la méthode HTTP (intention) et l'URL (ressource demandée), puis **produit une réponse**.



Le protocole HTTPS

HyperText Transfer Protocol Secure (HTTPS) :

- HTTP + *SSL/TLS*: **couche de chiffrement**,
- Permet de **vérifier l'identité du serveur** (certificat SSL) + **transmission chiffrée des données entre les deux machines** (ex. données de formulaire),
- Port par défaut: `443`.

Brique 2 : Anatomie des URLs

URL : *Uniform Resource Locator*, une chaîne de caractères décrivant **le nom et l'emplacement d'une ressource** (adresse d'un document sur le web)

Exemple :

```
http://example.com/un/chemin/page.html?firstname=tim&lastname=berners%20lee
```

- `http` : protocole
- `example.com` : nom de la machine hôte (ou nom de domaine) associé à une IP dans un enregistrement DNS
- `/un/chemin/page.html` : emplacement de la ressource sur la machine. Peut correspondre au path sur la machine (emplacement réel des fichiers) ou non
- `firstname=tim&lastname=berners%20lee` : **arguments d'URL** (*query parameters*) sous forme de clefs valeurs. Commence par `?`, séparés par `&`. Caractères spéciaux encodés (ex: le caractère *espace* est encodé par la chaîne de caratères `%20`)

Brique 3 : HTML

Vous connaissez déjà il me semble, on passe ?

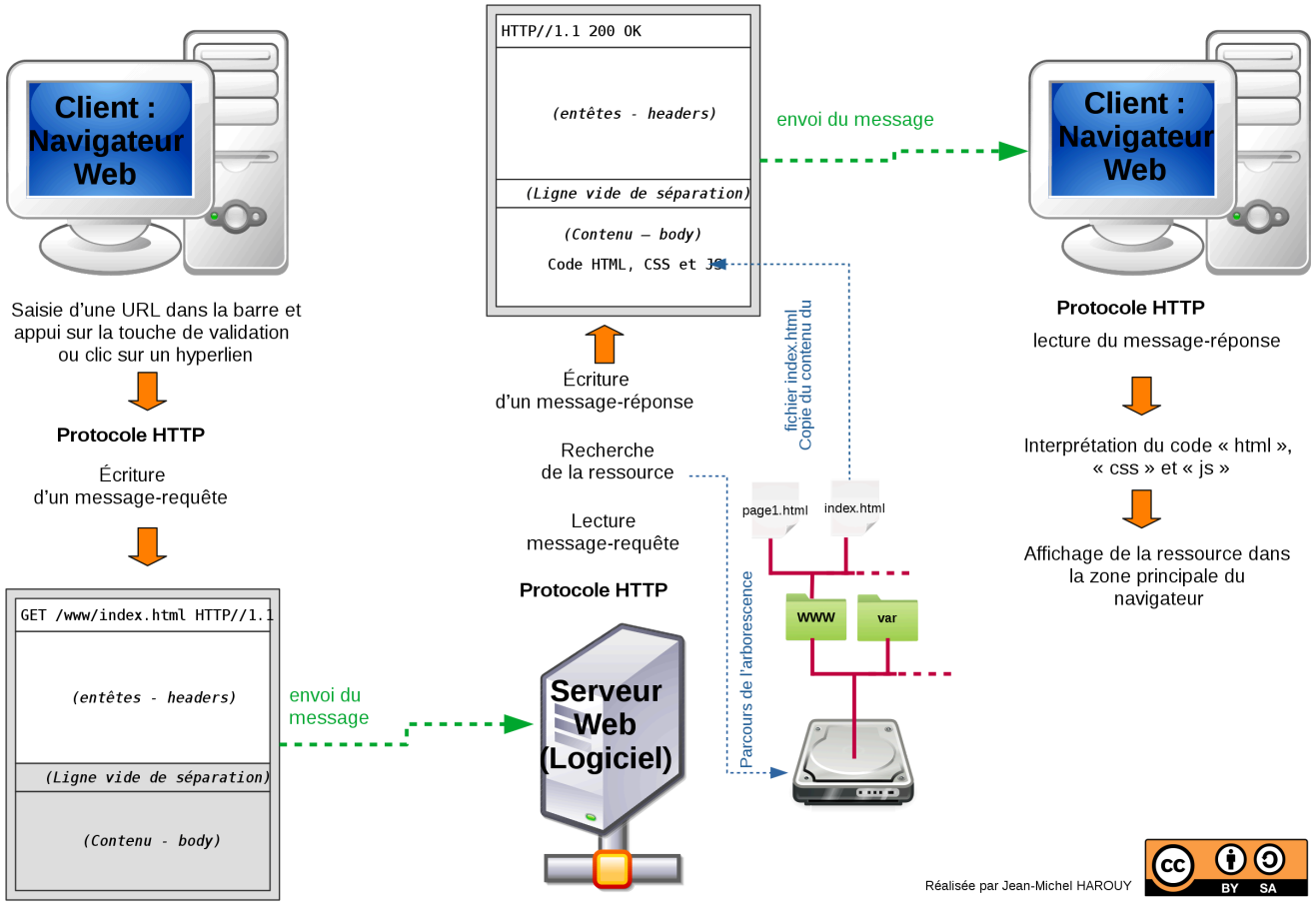
Les briques cousines

Les autres langages du web, interprétables par votre navigateur (*client*) :

- `CSS` : *Cascading Style Sheets* : langage permettant de décrire la présentation des pages HTML ;
- `Javascript` : langage interprété de type script execute dans le navigateur. Né chez Netscape, en 1995. Norme ECMASCRIPT pour uniformiser les langages de script coté client. **Le serveur vous délègue du code à exécuter chez vous**.

> RIP Adobe Flash

Pour résumer



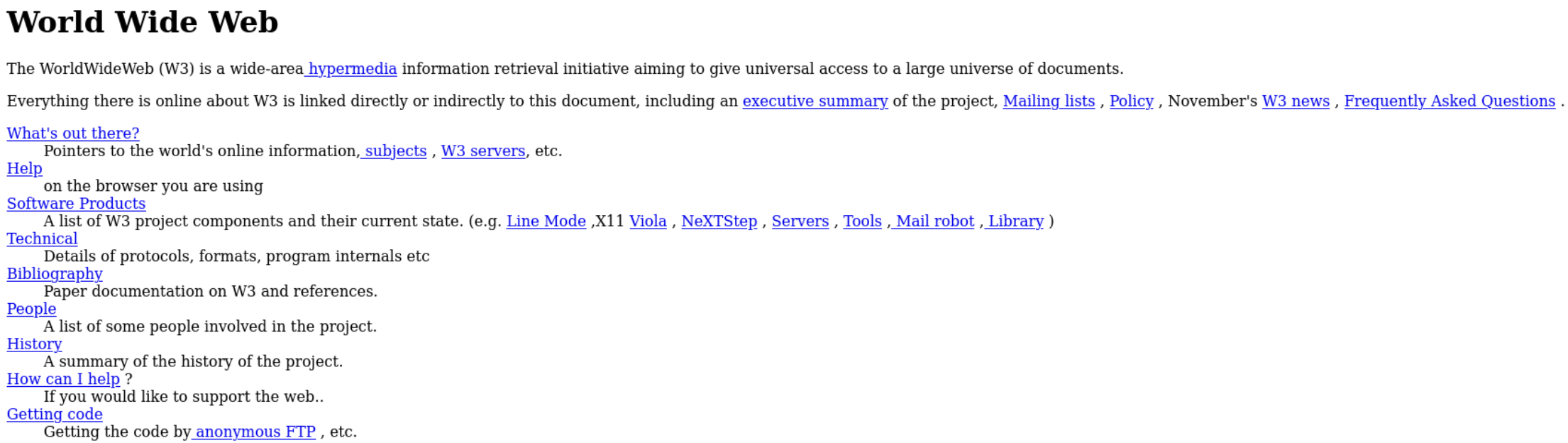
Navigateurs web (client HTTP + rendering + exec JS)

- Un navigateur web est un *client HTTP très sophistiqué* ;
- Permet d'**accéder à des ressources web** (requêtes GET), de les **récupérer** et de les **afficher** (rendu HTML+CSS) ;
- Préférer le navigateur natif à votre OS (consomme beaucoup de ressources) et qui *respecte votre vie privée* ;
- Tester vos pages web sur différents navigateurs (navigateurs supportés à préciser dans les specs d'un projet pro !), même si la standardisation avance bien (voir projet Web Platform Baseline)
- Liens utiles: W3C Markup Validation Service, caniuse

Web statique vs Web dynamique

D'un système hypertexte, orienté documents, aux applications web.

Page web *statique*



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

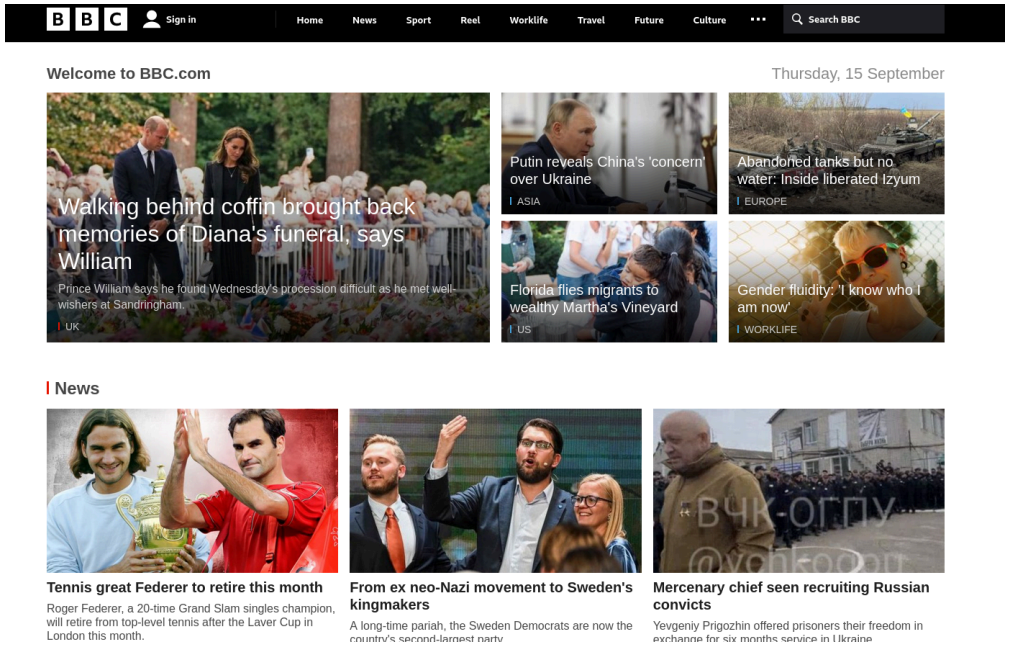
What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

- Page dont le contenu de change pas (on sert un document) ;
- Hyperliens pour naviguer d'une page à l'autre ;
- Contenu qui ne bouge *pas trop*.

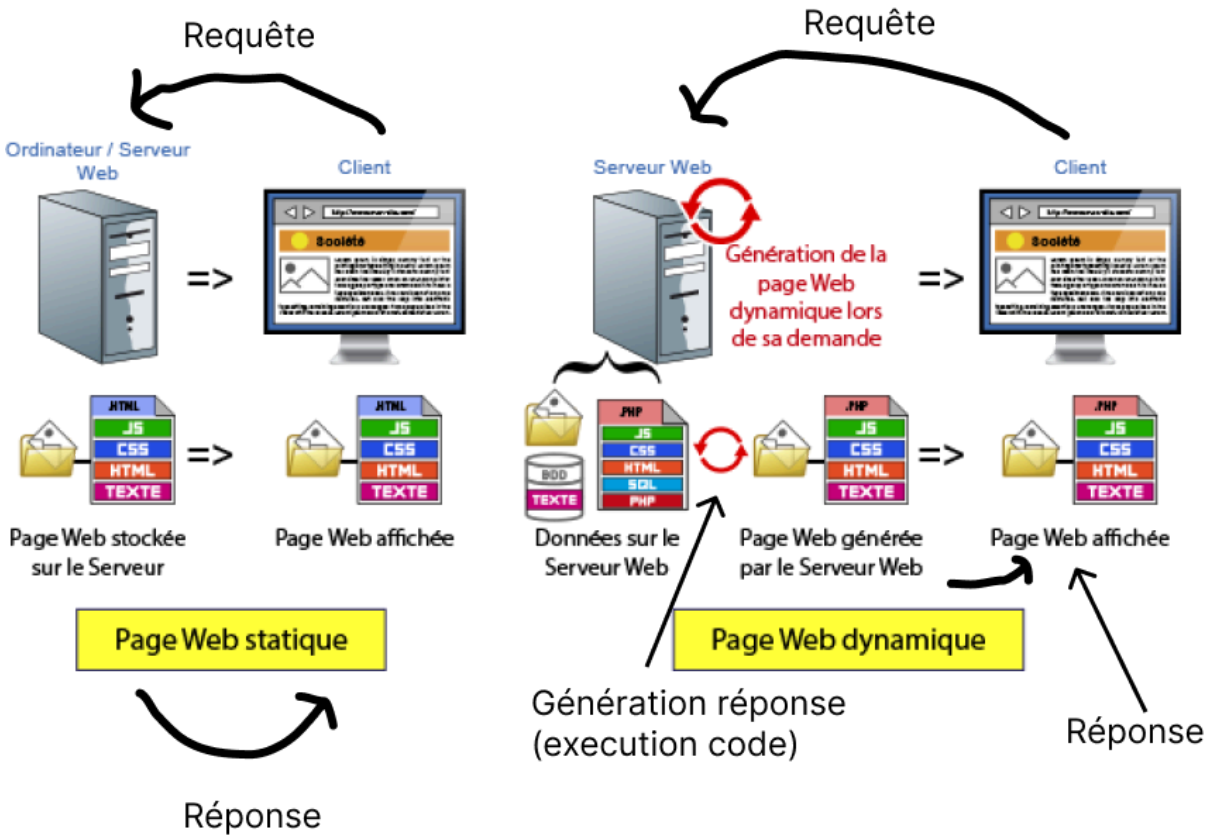
Page web *dynamique*

{.marp-bg-img}

- **Page HTML crée à la demande** (tout le monde n'a plus le *même* web) ;
- Contenu qui **bouge beaucoup** (infos, météo, etc.) ;
- Site web soutenu par un base de données ;
- **Contenu change** en fonction de : la date, la localisation (langue, culture), l'utilisateur·ice (role, préférences), paramètres d'URL, etc.
- Web "2.0"
- Client HTTP embarqué côté client (code Javascript qui met à jour la page de manière dynamique).



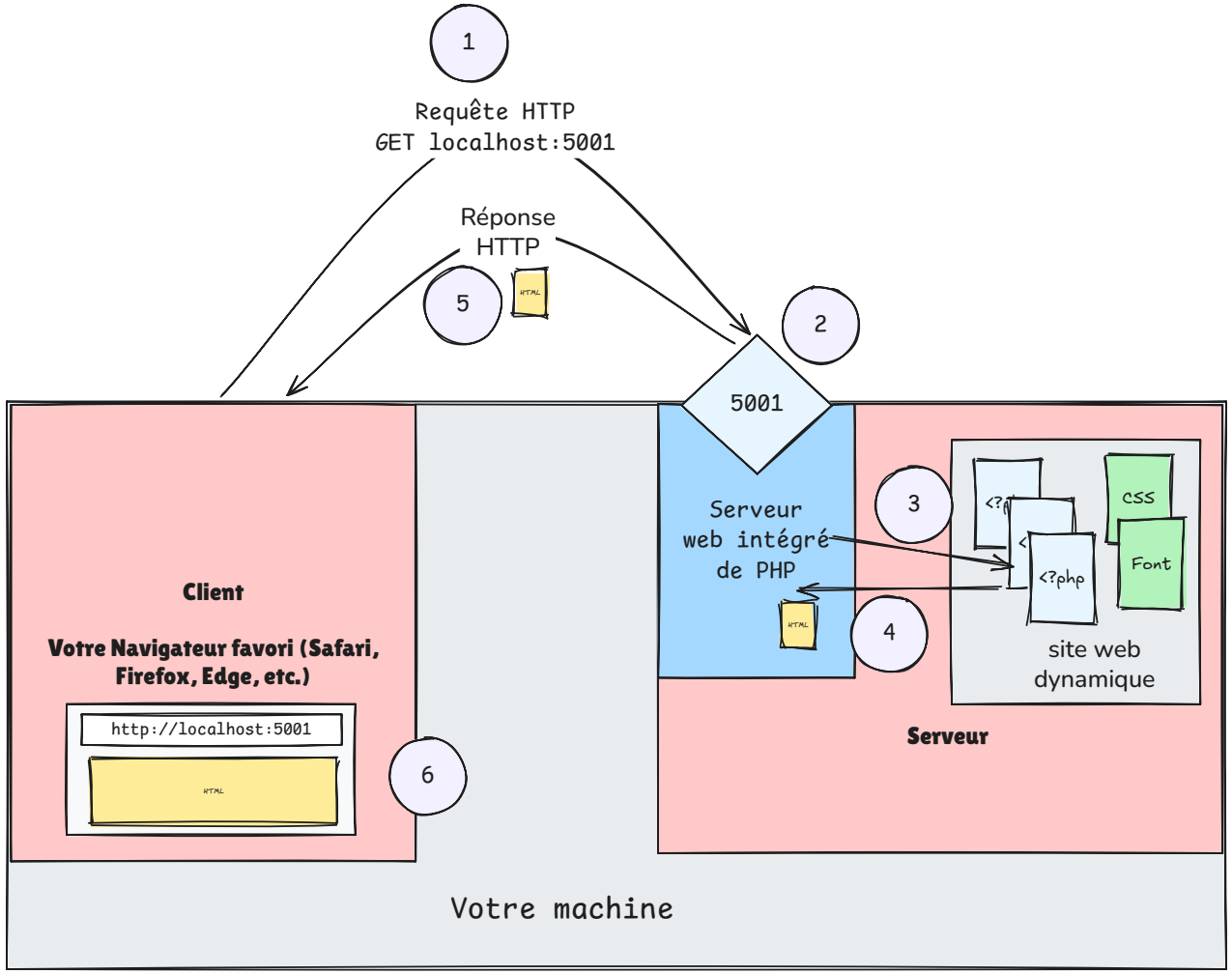
Page web *statique* et *dynamique*, en coulisses



Retour sur notre environnement local

Nous sommes en train de développer **un site web dynamique** (avec PHP) sur **un serveur local**. Dans cet *environnement de développement*, le **client** et le **serveur** sont **tous les deux sur notre machine**.

{.marp-bg-img}



En résumé

- Vous savez *installer* et *configurer* PHP ;
- Vous savez *à quoi sert* PHP et *comment* il fonctionne ;
- Vous savez *écrire* des premiers scripts PHP ;
- Vous savez *lancer un serveur web local* pour *développer* des sites web en PHP ;
- Vous comprenez ce *qu'est* le web : HTTP, HTML et adresses web (URL) ;
- Vous connaissez *l'architecture du web client/serveur* et le *rôle* de chacun dans cette relation ;
- Vous avez des *notions suffisantes* du protocole HTTP ;
- Vous comprenez la *structure d'une URL* ;
- Vous savez ce qu'est un *navigateur* ;
- Vous avez tous les outils pour *travailler en local*.