# INF344 – Données du Web
# Technologies côté serveur

Antoine Amarilli

# Serveurs

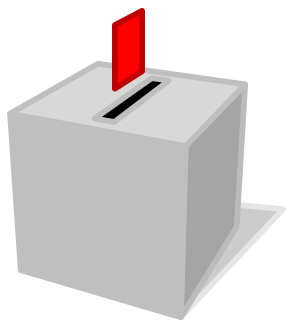
Traiter la requête (cas simples) ou la rediriger à un autre programme (cas complexes)

| | |
|---|---|
| **Apache** | Logiciel libre et gratuit, lancé en 1995 |
| **IIS** | Fourni avec Windows, propriétaire |
| **nginx** | Haute performance, libre et gratuit, lancé en 2002 |
| **GWS** | Google Web Server, interne (trafic de Google !) |
| **lighttpd** | Alternative légère à Apache |
| **Caddy** | Supporte HTTP/2 et HTTPS, lancé en 2015 |
| **Autres** | Rares, expérimentaux, embarqués... |

# SONDAGE : Serveur Web le plus populaire

Quel serveur Web a le plus de parts de marché ?

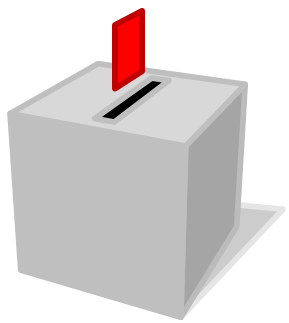
- **A**: Apache
- **B**: nginx
- **C**: Microsoft IIS
- **D**: Cloudflare Server



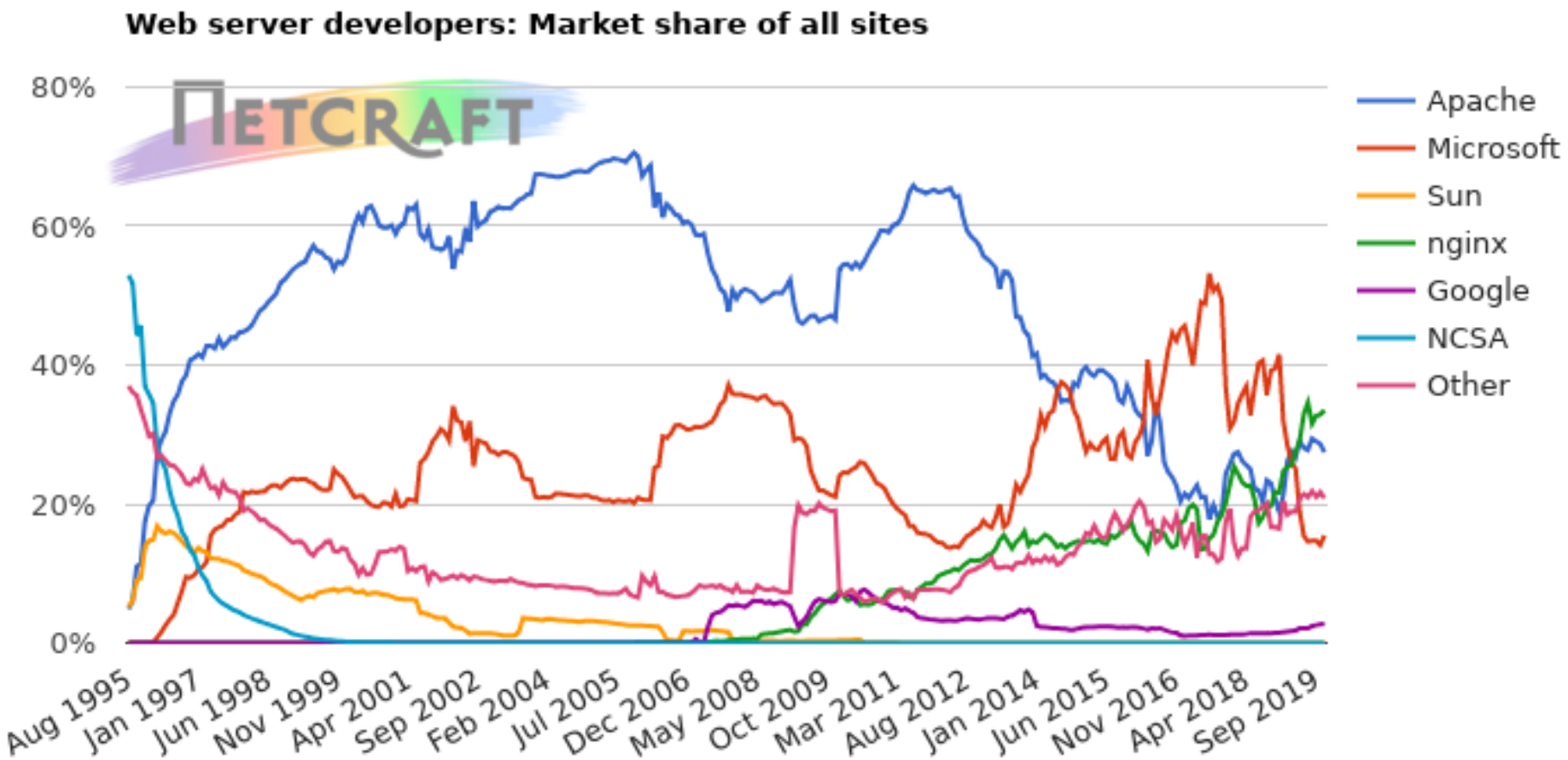
# SONDAGE : Serveur Web le plus populaire

Quel serveur Web a le plus de parts de marché ?

- **A**: **Apache**
- **B**: **nginx**
- **C**: Microsoft IIS
- **D**: Cloudflare Server



# Parts de marché




Mais W3Techs indique que Apache est utilisé par 39% des sites et Nginx seulement par 32%...

`https://w3techs.com/technologies/overview/web_server`

# Sites Web statiques simples

- Les différentes ressources sont stockées dans des **fichiers** :
  → `/var/www/page.html`, `/var/www/style.css`…
- Les pages sont organisées en une **arborescence** de répertoires
- Les chemins demandés **correspondent** à l'arborescence :
  → `GET /a/b.html` correspond à `/var/www/a/b.html`
- Si un **répertoire** est demandé :
  - Servir `index.html` s'il existe
  - Sinon, produire une liste des fichiers du répertoire

→ **Pérennité** des URLs ?

# Sites Web statiques simples

- Les différentes ressources sont stockées dans des **fichiers** :
  → `/var/www/page.html`, `/var/www/style.css`…
- Les pages sont organisées en une **arborescence** de répertoires
- Les chemins demandés **correspondent** à l'arborescence :
  → `GET /a/b.html` correspond à `/var/www/a/b.html`
- Si un **répertoire** est demandé :
  - Servir `index.html` s'il existe
  - Sinon, produire une liste des fichiers du répertoire

→ **Pérennité** des URLs ?

(Démo : naviguer dans une arborescence)

# Extensions Apache

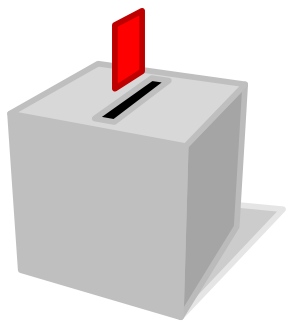
- Les fichiers `.htaccess` permettent de paramétrer Apache :
  - `deny from all` pour interdire un répertoire
  - Authentification HTTP
- URL rewriting :
  - `RewriteRule (.*\.png) /images/$1`
- Server Side Includes :
  - `<!-#include virtual="/footer.html" ->`

# Table des matières

# SONDAGE : Première technologie pour étendre les serveurs Web

Quelle est la technologie historique permettant à un navigateur d'invoquer un programme tiers ?

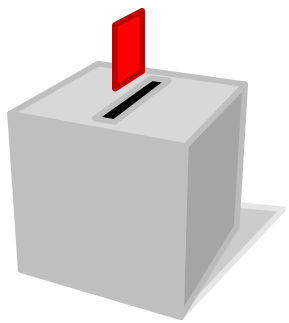
- **A**: Django
- **B**: CGI
- **C**: PHP
- **D**: `.htaccess`



# SONDAGE : Première technologie pour étendre les serveurs Web

Quelle est la technologie historique permettant à un navigateur d'invoquer un programme tiers ?

- **A**: Django
- **B**: **CGI**
- **C**: PHP
- **D**: `.htaccess`



# CGI

- Moyen historique pour un **serveur Web** d'invoquer un **programme externe** (n'importe quel langage)
- Exécute le **programme** sur les paramètres de la requête
- Le résultat de la requête est ce que **renvoie** le programme
- **Inconvénient** : créer un processus par requête est trop lourd
  - → **FastCGI** et autres mécanismes
  - → **Intégrer** le langage au serveur (par exemple PHP)

# PHP

- Lancé en 1995 ; des **centaines de millions** de sites l'utilisent [1]
- Langage de programmation **complet**
- S'ajoute aux pages HTML. Exemple :

```
<ul>
<?php
  $from = intval($_POST['from']);
  $to = intval($_POST['to']);
  for ($i = $from; $i < $to; $i++) {
    echo "<li>$i</li>";
  }
?>
</ul>
```

1. https://secure.php.net/usage.php

# Inconvénients de PHP

- Pas prévu comme un langage complet à l'origine
  - → À l'origine, Personal Home Page
  - → Difficile d'assurer la compatibilité…
- Encourage de mauvaises pratiques de sécurité
  - → Historiquement, `$_POST['from']` accessible comme `$from`!
  - → Facile d'oublier de (re)valider les données utilisateur…
- Mauvaises performances ?
  - → À l'origine, langage interprété
  - → Machine virtuelle avec compilation JIT (HHVM à Facebook)

# Autres langages côté serveur

**ASP** Microsoft, add-on à IIS, propriétaire

**ASP.NET** Microsoft, successeur d'ASP pour .NET

**ColdFusion** Adobe, commercial, propriétaire

**JSP** Intégration entre Java et un serveur Web

- **Servlet** classe Java suivant une API
- **Web container** serveur pour gérer les servlets (par exemple Apache Tomcat)

**node.js** Moteur JavaScript V8 (de Chrome) et serveur Web

→ Même langage pour le client et le serveur

**Python** Frameworks Web : Django, CherryPy, Flask

**Ruby** Frameworks Web : Ruby on Rails, Sinatra

# Table des matières

# Bases de données

- SGBD : Système de Gestion de Bases de Données
- Abstraction de la tâche de stocker de l'information, la mettre à jour, et la récupérer suivant des requêtes
- Modèle historique et le plus courant : modèle relationnel, SQL
- Avantages :
  - Gérer simplement les données sans s'occuper du stockage
  - Remplacer plutôt facilement un SGBD par un autre
  - Facilement déléguer la tâche à des machines séparées
  - Optimiser les SGBD plutôt que les usages ad hoc
- Déviations du modèle relationnel : NoSQL

# Modèle relationnel : structure

- **Relations** (tables), par exemple `Clients`, `Produits`…
- **Attributs** (colonnes), par exemple `Prénom`, `Identifiant`…
- **Enregistrements** (lignes)
- **Exemple** : table `clients` :

| id | prenom | nom | cp | ville | conseiller |
|---|---|---|---|---|---|
| 1 | Jean | Dupont | 75013 | Paris | 42 |
| 2 | François | Pignon | 75005 | Paris | 42 |

- Les attributs ont un **type** (chaîne de caractères, nombre entier, décimal, date, blob binaire…)

# SQL

- Structured Query Language
- Sémantique et modèle propre, liens avec la logique
- Standardisé (1986, mis à jour en 2011) mais extensions...
- Sélection :

```
SELECT ville, COUNT(*)
FROM clients
WHERE conseiller=42
GROUP BY ville;
```

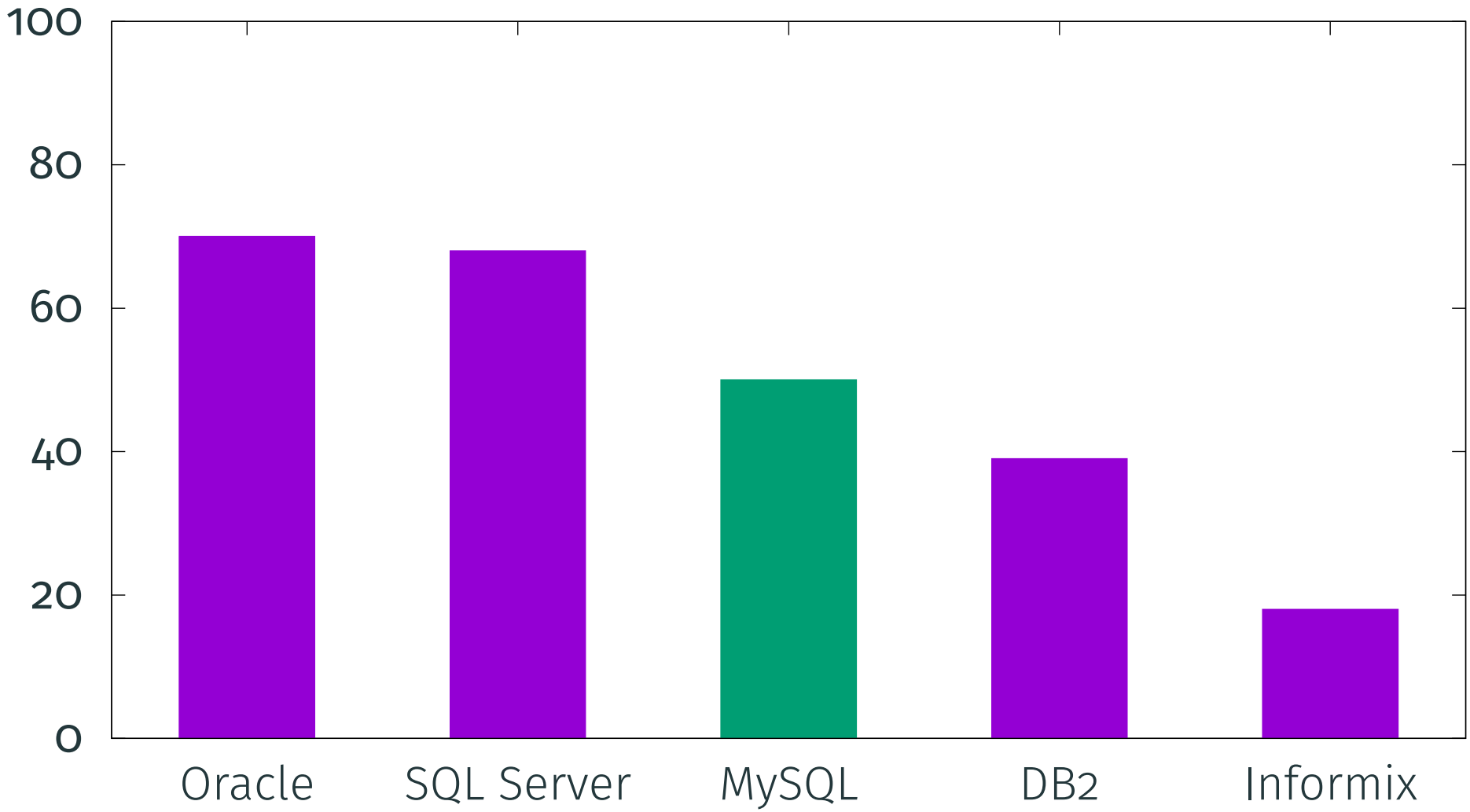
- Mise à jour :

```
UPDATE clients
SET conseiller=43
WHERE conseiller=42 AND cp=75013;
```

# SGBDs

| | |
|---|---|
| **Oracle** | Commercial et propriétaire, lancé en 1979 |
| **IBM DB2** | IBM, commercial et propriétaire, Lancé en 1983 |
| **Informix** | Autre système IBM, lancé en 1981 (rachat) |
| **SQL Server** | Microsoft, commercial et propriétaire, lancé en 1989 |
| **MySQL** | Libre (mais éditions commerciales), le plus courant sur le Web, lancé en 1995, racheté par Oracle |
| **MariaDB** | Fork de MySQL, lancé en 2009 |
| **PostgreSQL** | Libre, concurrent de MySQL, lancé en 1995 |
| **SQLite** | Librairie dans un programme, léger, lancé en 2000 |

# SGBDs, parts de marché en entreprise (cumulatives)




Source : étude Gartner de 2008 citée dans `https://www.mysql.com/why-mysql/marketshare/`

Autre source `db-engines.com` : PostgreSQL, SQLite, MS Access…

# Modèle relationnel : idées importantes

- Contraintes d'intégrité :
  - Clés primaires : identifiant unique, détermine l'enregistrement
  - Clés étrangères : valeur qui fait référence à une clé primaire
- Formes normales : éviter la répétition des données
- Procédures stockées : code pour des requêtes complexes
- Vues : table virtuelle définie par une requête (matérialisée ?)
- Triggers : répondre à un événement
- Transactions : atomicité pour éviter les incohérences
- Distribution : répartir entre plusieurs machines
- Concurrence : traiter des requêtes simultanées
- Performance : index, caches mémoire, etc.
- Utilisateurs et droits d'accès (toujours relégué à PHP ou au framework Web en pratique)

# Injections SQL

- Une application serveur fabrique souvent des requêtes à partir de **données utilisateur** :

```
SELECT nom, telephone FROM employes WHERE nom='$nom'
```

- Si l'utilisateur soumet `toto`, on veut exécuter la requête :

```
SELECT nom, telephone FROM employes WHERE nom='toto'
```

- Maintenant, imaginons qu'un utilisateur soumette :

```
toto' UNION ALL SELECT nom, mdp FROM employes --
```

- La base de données va recevoir et exécuter :

```
SELECT nom, telephone FROM employes WHERE nom='toto'
UNION ALL SELECT nom, mdp FROM employes -- '
```

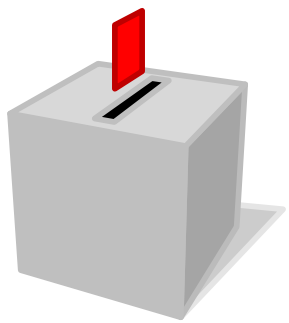
→ **Préparation** pour placer les paramètres dans les requêtes

→ Sinon, **échapper** ou **retirer** les caractères dangereux

# SONDAGE : NoSQL

Quel est le système NoSQL le plus populaire (d'après `db-engines.com` et la Stack Overflow Developer Survey 2019) ?

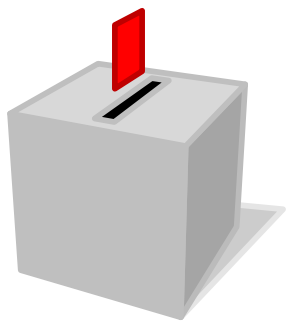
- **A**: Cassandra
- **B**: Redis
- **C**: Firebase
- **D**: MongoDB



# SONDAGE : NoSQL

Quel est le système NoSQL le plus populaire (d'après `db-engines.com` et la Stack Overflow Developer Survey 2019) ?

- **A**: Cassandra
- **B**: Redis
- **C**: Firebase
- **D**: **MongoDB**



# NoSQL

- Beaucoup d'**expressivité** avec le modèle relationnel
- Structure très **contrainte** et peu modifiable
- Exigences fortes de **cohérence** sur du relationnel distribué

→ **NoSQL** : renoncer au modèle relationnel pour de meilleures performances et un meilleur passage à l'échelle

- Quelques exemples :
  - Stockage de **documents** entiers peu structurés
  - Stockage de **graphes**
  - Stockage de couples **clé-valeur**
  - Stockage de **triplets** pour le Web sémantique
- Système le plus populaire[2] : **MongoDB**, utilisé par Craigslist, le CERN, Shutterfly, Foursquare, etc.

2. `https://db-engines.com/en/ranking`

# Table des matières

# Frameworks

- Ensemble de fonctions et d'outils, organisé autour d'un langage, pour les applications Web
- Intégration AJAX, production de code JavaScript…
- MVC :

**Modèle** La structure des données de l'application et les fonctions pour les manipuler

**Vue** La présentation des données destinée au client

**Contrôleur** Le contrôle de l'interaction avec les données du modèle à travers la vue

# ORM

- Object Relational Mapping
- Les frameworks Web utilisent souvent la programmation **objet**
- **Persistence** des objets dans une base de données relationnelle
- Ne concerne pas les **méthodes** !
- Historiquement : tentatives de bases de donnée **objet**

# Exemple d'un ORM (Django)

Définir une classe (table) avec des attributs :

```
from django.db import models
class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()
```

Créer un objet (insérer un enregistrement) :

```
b = Blog(name='Beatles', tagline='Latest Beatles news.')
b.save()
```

Récupérer un objet (faire une requête) :

```
b = Blog.objects.filter(name='Beatles')
```

# Templates

- Modèles avec **champs** de Pages HTML
- **Exemple** (avec Jinja2) :

```
<h1>Résultats pour "{{ recherche }}"</h1>
<ul>
  {% for objet in resultats %}
    <li>
      <a href="details/{{ objet.id }}">
        {{ objet.nom }}
      </a>
    </li>
  {% endfor %}
</ul>
```

# Routage des URLs

- Router suivant le chemin et la méthode
- Exemple (avec Flask) :

```
@app.route('/')
def index():
  pass # Préparer la page d'accueil

@app.route('/message/<int:message_id>')
def message(message_id):
  pass # Préparer l'affichage du message <message_id>

@app.route('/upload', methods=['POST'])
def upload():
  pass # Traiter un upload
```

# CMS

- Content Management System
- Faire un site sans programmation :
  - Édition de page avec un texte riche, ou langages simplifiés (Markdown, Textile, BBCode…)
  - Hébergement d'images, vidéos, etc.
  - Gestion d'utilisateurs
  - Choix du thème graphique
- Différents types :

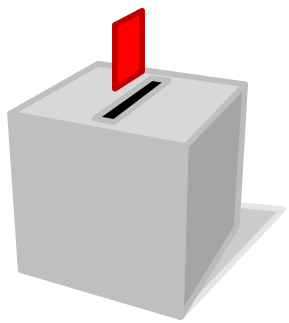
| | |
|---|---|
| **Wikis** | MediaWiki, MoinMoin, PmWiki… |
| **Forums** | phpBB, PunBB, Phorum, vBulletin… |
| **Blogs** | WordPress, Movable Type, Drupal, Blogger… |
| **QA** | (comme StackOverflow) : Shapado, OSQA, AskBot |
| **Commerce** | Magento, PrestaShop… |

- Autres services hébergés : Weebly, Wix, etc.

# SONDAGE : Parts de marché CMS

Quel est le CMS le plus populaire ?

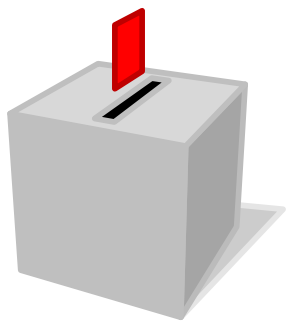
- **A**: Joomla
- **B**: WordPress
- **C**: Drupal
- **D**: Squarespace



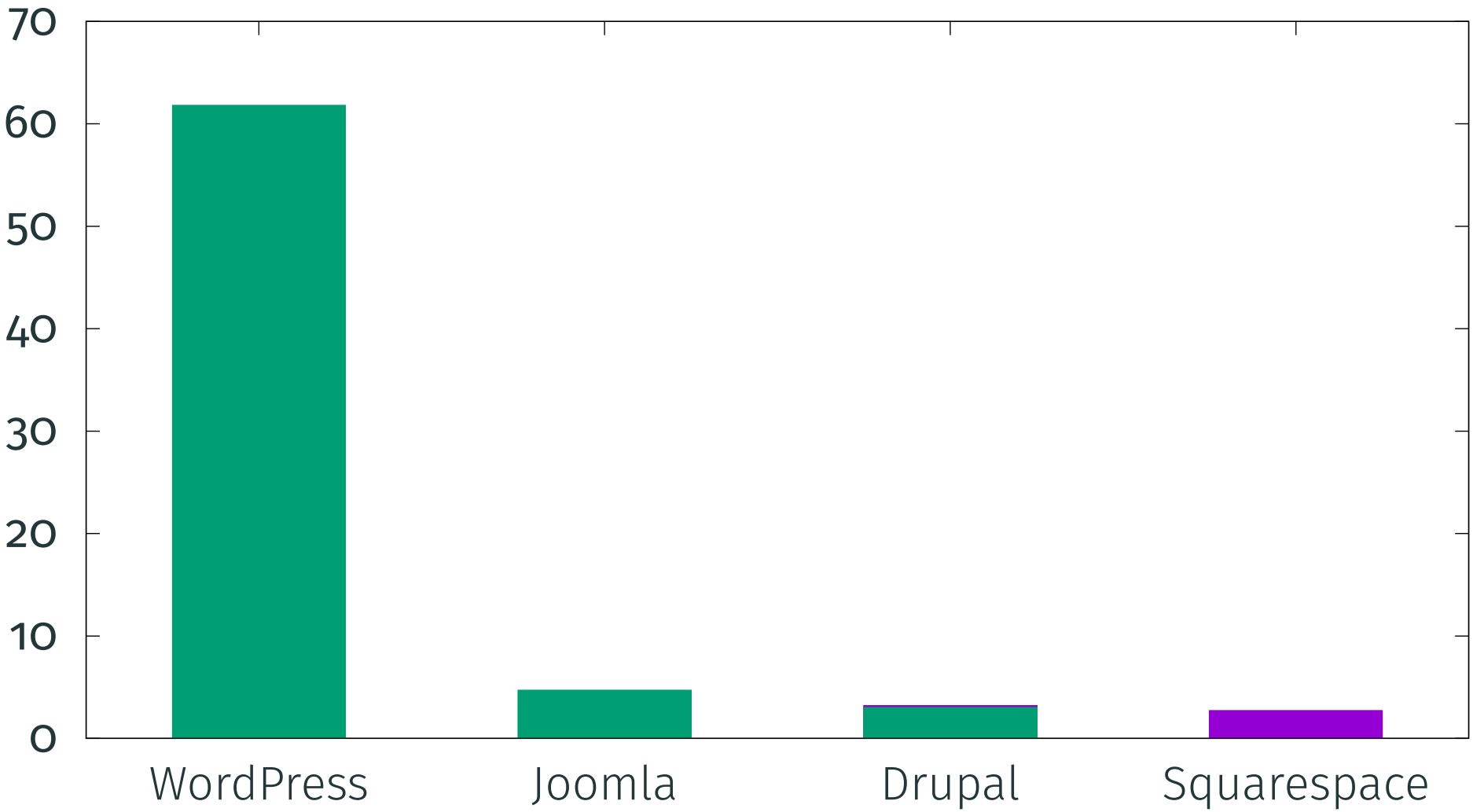
# SONDAGE : Parts de marché CMS

Quel est le CMS le plus populaire ?

- **A**: Joomla
- **B**: **WordPress**
- **C**: Drupal
- **D**: Squarespace



# Parts de marché




Sites avec chaque CMS (novembre 2018) ; tous en PHP.

Source : https://w3techs.com/technologies/overview/content_management/all

# node.js

- Plateforme **MEAN** :
  - **MongoDB**
  - **Express.js** (framework minimal pour Node.js)
  - **Angular**
  - **Node.js**
- Avantage : **JS isomorphe**, même code sur le client et le serveur
  - **Idée :** d'abord faire tourner le code sur le **serveur** pour calculer la vue
  - **Puis :** envoyer le code Javascript sur le client en arrière-plan et l'exécuter
  - Autres **avantages** pour la réutilisation de code (e.g., validation de l'entrée)
- Gestionnaire de paquets pour les bibliothèques : `npm` (ou `yarn`, `bower`)

# Meteor

- Solution intégrée par dessus **node.js**
- **Database everywhere :** Avoir un **cache partiel** de la base de données sur le client qui est synchronisé automatiquement avec le serveur
  - → Plus efficace et plus simple à coder
  - → Règles d'accès pour limiter ce que le client peut éditer
- **Compensation de la latence :** faire des **changements optimistes** sur le client et les synchroniser en arrière-plan avec le serveur
- Gestion des **sessions** et des **utilisateurs**
- Diverses **librairies de routage** : FlowRouter, IronRouter, ...

# Outillage Javascript

- Minification et obfuscation
  - → Rendre ça réversible pour le débogage avec des source maps
- Linting : vérifier que le code est conforme à un manuel de style
- Documentation : JSDoc (analogue de Doxygen)
- Empaqueter le code et ses dépendances : webpack
- Élimination du code mort ou tree shaking
- Transpilation vers d'autres variantes de Javascript, e.g., Babel, Google Closure Tools
- Système de gestion de ces tâches : grunt, gulp

# Identifier les technologies serveur

Sur le client, le code HTML, JavaScript, CSS est **accessible** (modulo minification et obfuscation). Sur le serveur, on n'a rien !

- `whois` : base de données fournissant (souvent) des infos sur le propriétaire d'un nom de domaine
- Localisation **géographique** des IP des serveurs
- `traceroute`, pour connaître le **chemin réseau** vers un hôte
- Outils de scan (`nmap`) pour localiser les machines et identifier le système d'exploitation (fingerprinting)
- Header `Server`, possiblement faux
- Forme des **identifiants de session**, cookies…
- Chemins d'accès et **extensions** : `.php`, `.asp`, …
- **Commentaires** dans le code HTML

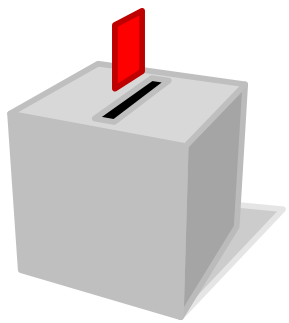
→ Utilisé par les **pirates** pour identifier des services vulnérables

# Table des matières

# SONDAGE : Hébergement bon marché

Combien cela coûte-t-il au minimum d'héberger un site Web peu gourmand sur un VPS ?

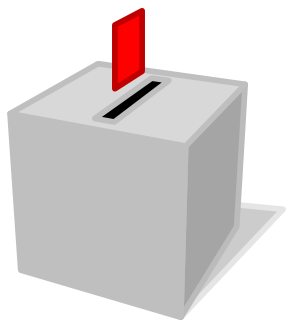
- **A**: Moins de 10 EUR par an
- **B**: Entre 10 et 50 EUR par an
- **C**: Entre 50 et 100 EUR par an
- **D**: Plus de 100 EUR par an



# SONDAGE : Hébergement bon marché

Combien cela coûte-t-il au minimum d'héberger un site Web peu gourmand sur un VPS ?

- **A**: Moins de 10 EUR par an
- **B**: **Entre 10 et 50 EUR par an**
- **C**: Entre 50 et 100 EUR par an
- **D**: Plus de 100 EUR par an



# Comment se faire héberger un site Web

- Solution la plus simple : solution **hébergée**, e.g., Wordpress, Weebly,
- Les fournisseurs d'accès proposent souvent un hébergement avec **espace disque limité**, parfois PHP/MySQL
- On peut louer un **VPS** ou un **serveur dédié** (quelques EUR/mois)
- On peut héberger un site Web **chez soi** derrière une connexion fibre optique avec une adresse IP fixe
- On peut aussi utiliser le **cloud** (serverless computing)

# Infrastructure d'hébergement

- Serveur : machine qui reste allumée pour répondre aux requêtes
- Centre de données : local pour serveurs avec une bonne connexion, alimentation électrique, climatisation, sécurité physique…
- Serveurs virtuels : machine virtualisée, imite une vraie machine
- Cloud : location simple à grande échelle de machines
  - → Possibilité d'ajuster le nombre de machines selon la charge
- Content delivery network : services de proxy intermédiaire
- Répartition de charge entre plusieurs machines
  - → Au niveau DNS : géographique et round-robin
  - → Au niveau logiciel

# Comment héberger un site Web soi-même

- Louer un nom de domaine (environ 15 euros par an)
- Avoir un serveur (VPS, ou dédié, ou chez soi)
- Configurer SSH pour se connecter à la machine et l'administrer
- Installer un serveur Web votre framework, CMS, etc.

# Un exemple concret : Wikimedia

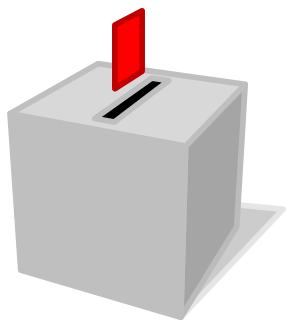
`wikipedia.org`, 13e site le plus visité au monde en 2020 d'après `http://www.alexa.com/topsites`

WIKIPEDIA
The Free Encyclopedia

# SONDAGE : Nombre d'employés

Quel est l'ordre de grandeur du nombre de pages servies par seconde en moyenne par les projets Wikimedia ?

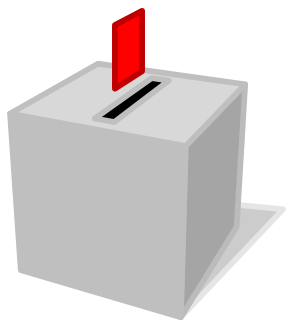
- **A**: 100 pages par seconde
- **B**: 1 000 pages par seconde
- **C**: 10 000 pages par seconde
- **D**: 100 000 pages par seconde



# SONDAGE : Nombre d'employés

Quel est l'ordre de grandeur du nombre de pages servies par seconde en moyenne par les projets Wikimedia ?

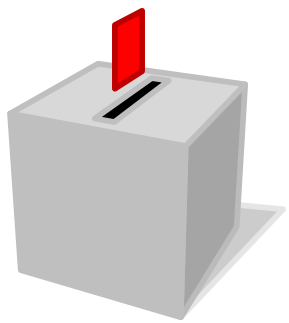
- **A**: 100 pages par seconde
- **B**: 1 000 pages par seconde
- **C**: **10 000 pages par seconde**
- **D**: 100 000 pages par seconde



# SONDAGE : Coûts techniques

Quels sont les dépenses en hébergement Internet de la fondation Wikimedia en 2018–2019 ?

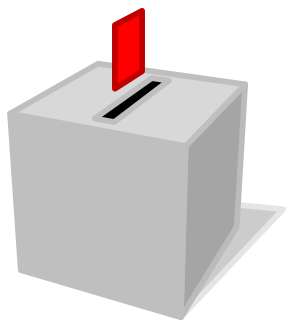
- **A**: 200 000 USD
- **B**: 2 000 000 USD
- **C**: 20 000 000 USD
- **D**: 200 000 000 USD



# SONDAGE : Coûts techniques

Quels sont les dépenses en hébergement Internet de la fondation Wikimedia en 2018–2019 ?

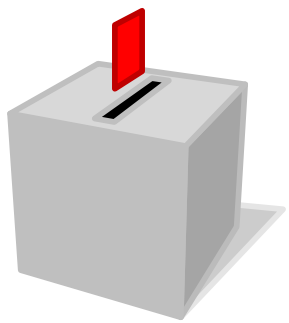
- **A**: 200 000 USD
- **B**: **2 000 000 USD**
- **C**: 20 000 000 USD
- **D**: 200 000 000 USD



# SONDAGE : Coûts humains

Quels sont les dépenses en salaires de la fondation Wikimedia en 2018–2019 ?

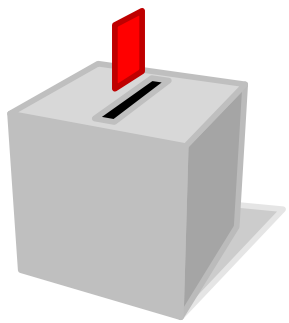
- **A**: 400 000 USD
- **B**: 4 000 000 USD
- **C**: 40 000 000 USD
- **D**: 400 000 000 USD



# SONDAGE : Coûts humains

Quels sont les dépenses en salaires de la fondation Wikimedia en 2018–2019 ?

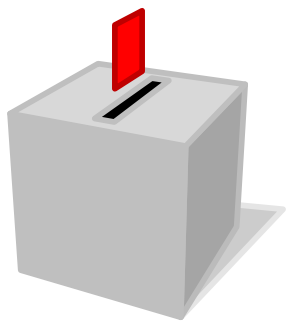
- **A**: 400 000 USD
- **B**: 4 000 000 USD
- **C**: **40 000 000 USD**
- **D**: 400 000 000 USD



# SONDAGE : Coûts humains

Quelles sont les dépenses totales de la fondation Wikimedia sur l'année 2018–2019 ?

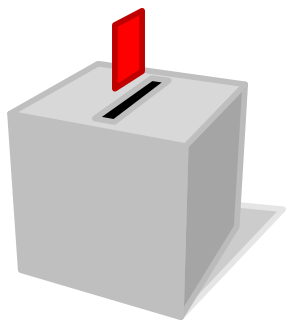
- **A**: 800 000 USD
- **B**: 8 000 000 USD
- **C**: 80 000 000 USD
- **D**: 800 000 000 USD



# SONDAGE : Coûts humains

Quelles sont les dépenses totales de la fondation Wikimedia sur l'année 2018–2019 ?

- **A**: 800 000 USD
- **B**: 8 000 000 USD
- **C**: **80 000 000 USD**
- **D**: 800 000 000 USD



# Finances de Wikimedia

- Organisation caritative, environ 350 employés en 2020
- 120 millions de dollars de revenus en 2018–2019 (surtout des dons)
- Coûts techniques en 2018–2019 : quelques millions de dollars

# Statistiques générales

- À peu près **un millier** de serveurs au total (2013)
- **137 115** utilisateurs actifs sur `en.wikipedia.org` [3]
- Près de **1000** éditions par minute au total [4]
- **16 milliards** de pages vues par mois sur tous les projets [5]
- Environ **7 000** par seconde en moyenne mais pointes à **50 000** [6]
- **825 millions** d'appareils uniques par mois sur la Wikipedia anglophone [7]

---

3. https://en.wikipedia.org/wiki/Wikipedia:Wikipedians#Number_of_editors
4. https://grafana.wikimedia.org/dashboard/
5. https://stats.wikimedia.org/v2/
6. https://arstechnica.com/information-technology/2008/10/wikipedia-adopts-ubuntu-for-its-server-infrastructure/
7. https://stats.wikimedia.org/v2/

# Infrastructure générale

- Centres de données :
  - → Site principal : Ashburn, Virginia (Equinix)
  - → Pour l'Europe (réseau et cache), Amsterdam (EvoSwitch, Kennisnet)
  - → Cache : San Francisco (United Layer), Singapour (Equinix)
  - → Autres sites : Dallas, Chicago
  - → Centre de données de secours : Carrollton, Texas (CyrusOne)
- Serveurs Dell sous Ubuntu [8] et Debian
- `puppet` pour gérer la configuration des serveurs
- Logiciels de monitoring : Icinga, Grafana
  - → `grafana.wikimedia.org`
  - → `status.wikimedia.org`

8. `https://insights.ubuntu.com/2010/10/04/wikimedia-chooses-ubuntu-for-all-of-its-servers/`

# Tâches principales (chiffres en 2013)

- Logiciel de gestion du wiki : **MediaWiki**, en PHP
- Serveur **Apache**, utilise HHVM [9]
  - → **192** machines (à Ashburn)
- Base de données : **MariaDB**
  - → **54** machines pour la base de données
  - → **10** machines de stockage avec 12 disques de 2 TB en RAID10
- Stockage de fichiers distribué : **Ceph** (anciennement **Swift**)
  - → **12** serveurs
- Serveurs de tâches asynchrones (base de données NoSQL **Redis**)
  - → **16** serveurs

9. https://blog.wikimedia.org/2014/12/29/how-we-made-editing-wikipedia-twice-as-fast/

# Caches (chiffres de 2013)

- Squid
  - → 8 machines pour le multimédia
  - → 32 machines pour le texte
- Varnish
  - → 8 machines
- Invalidation du cache avec MediaWiki
- Memcached entre MediaWiki et la base de données
  - → 16 machines

→ 90% du trafic n'utilise que le cache et non Apache.[10]

10. https://blog.wikimedia.org/2013/01/19/wikimedia-sites-move-to-primary-data-center-in-ashburn-virginia/

# Autres services (chiffres de 2013)

- Proxies de terminaison SSL avec **nginx** :
  → 9 machines
- **Load balancing** avec LVS (Linux Virtual Server) :
  → 6 serveurs
- **Indexation** pour la fonction de recherche :
  **Lucene** 25 serveurs
  **Solr** 3 serveurs
- **Redimensionnement** de fichiers multimédia :
  **Images** 8 serveurs
  **Vidéos** 2 serveurs
- **Statistiques** : 27 serveurs
- **Traitement des paiements en ligne** : 4 serveurs
- Serveurs DNS, services divers, snapshots, etc.

# Exemple de rack (2015)

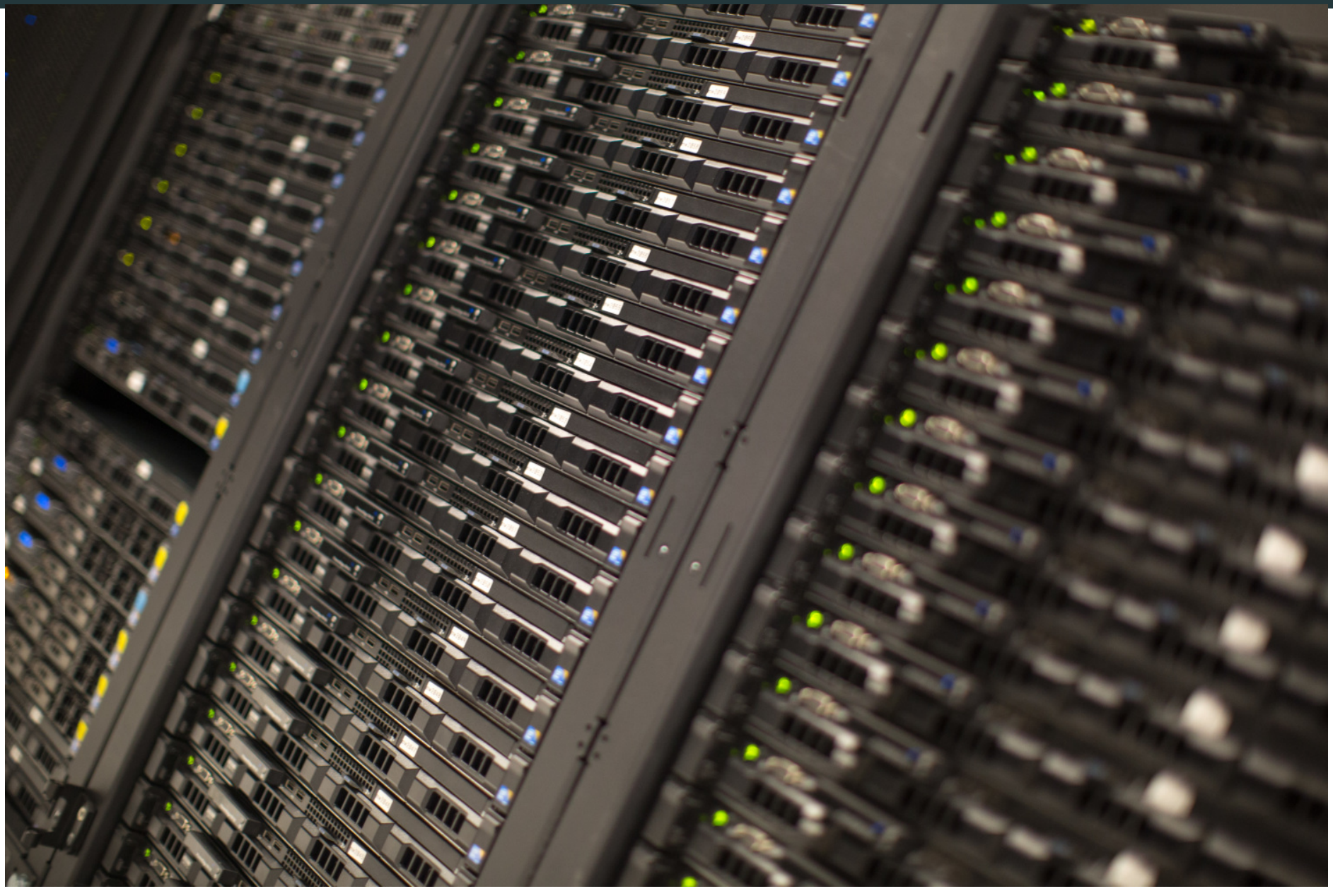


Image credits : `https://meta.wikimedia.org/wiki/File:Wikimedia_Foundation_Servers_2015-90.jpg`

# Crédits

- Matériel de cours inspiré de notes par Pierre Senellart
- Merci à Pierre Senellart pour sa relecture
- Transparent 4 : chiffres Netcraft `https://news.netcraft.com/archives/category/web-server-survey`