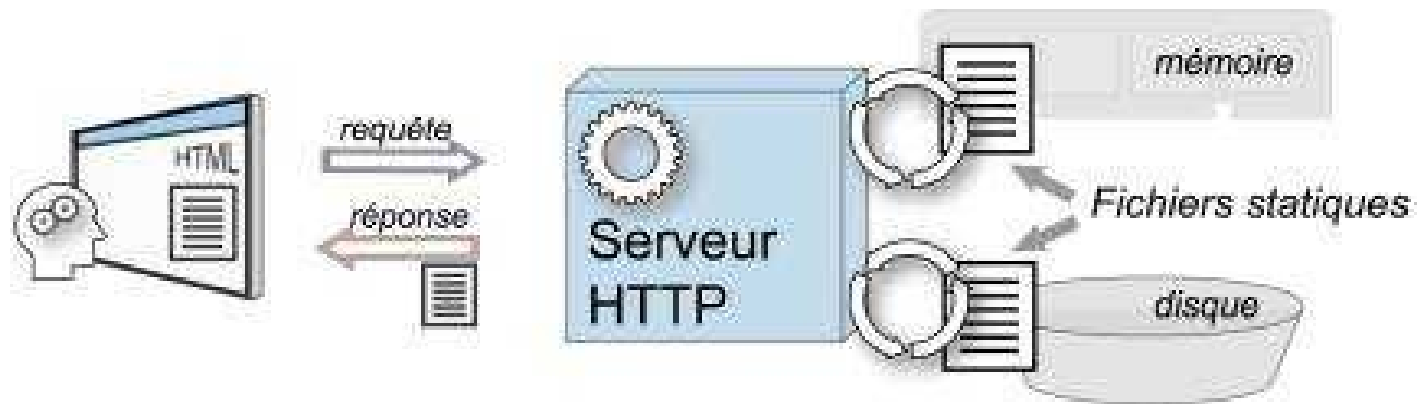


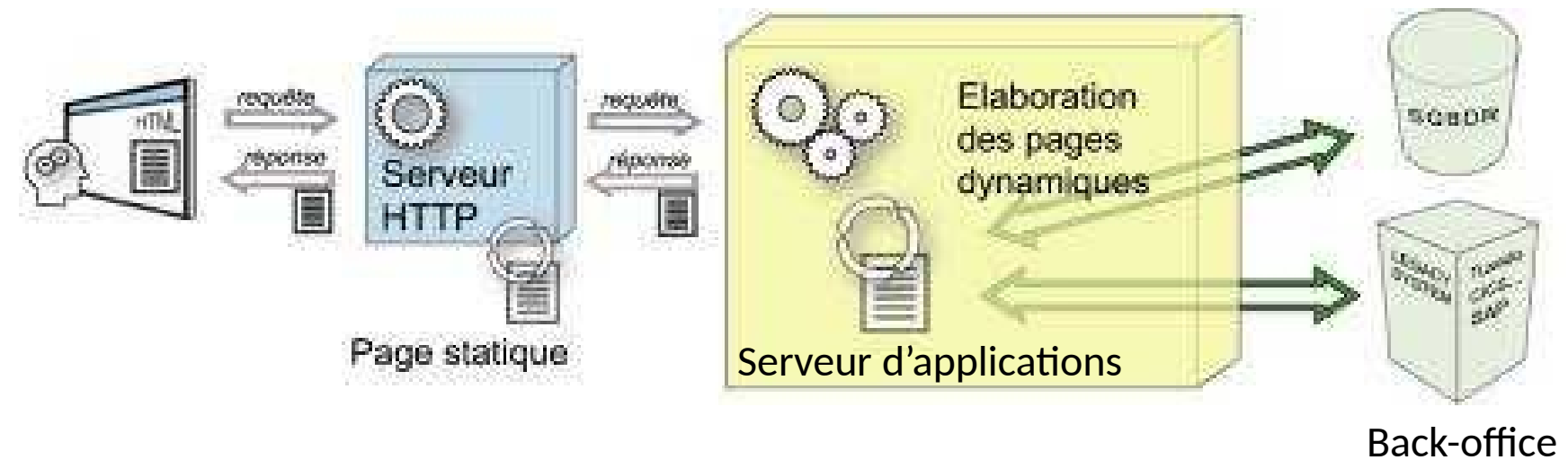


Évolution des architectures d'Application Web

Architecture Web statique

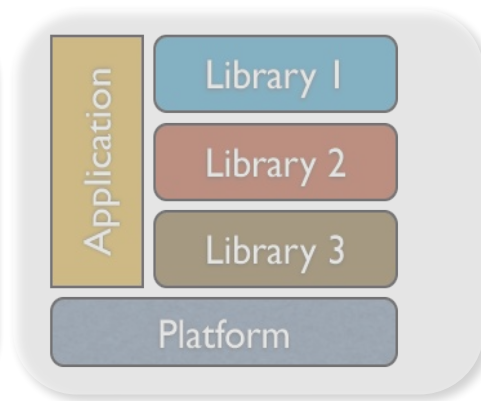
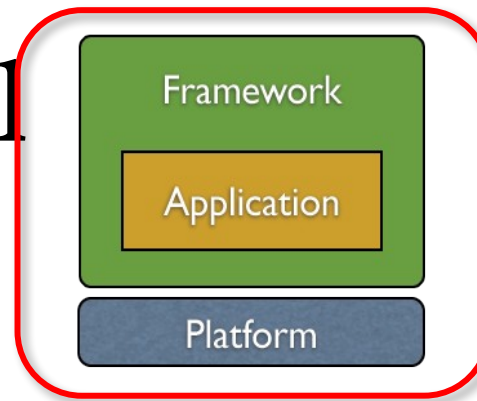


Architecture Web dynamique

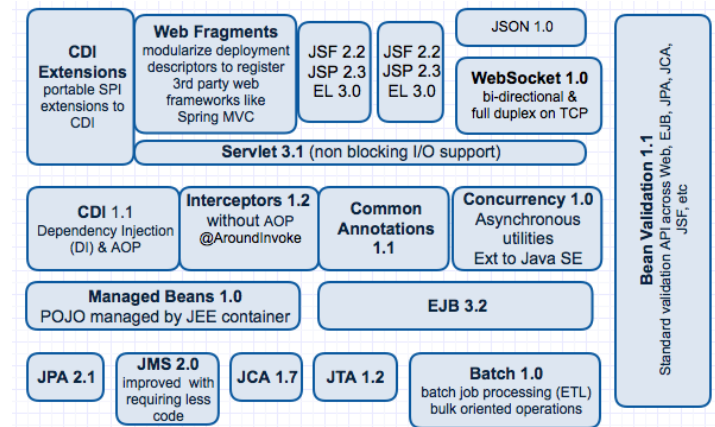


Cadre architectural

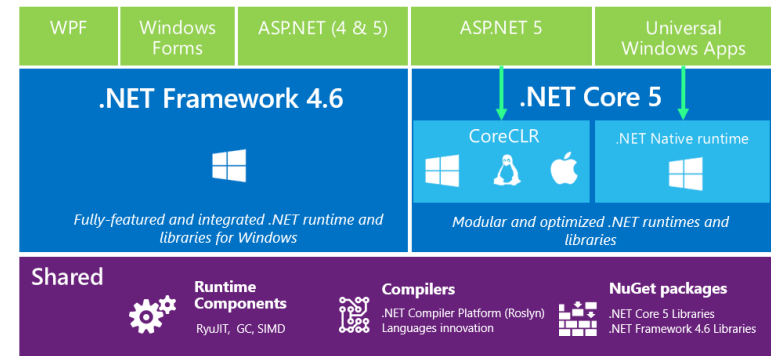
- Logiciel d'infrastructure offrant un contexte d'exécution isolé aux composants applicatifs



- La pile JEE (1995)
 - Multi plateforme
 - Mono langage (Java)



- La pile .NET (2002)
 - Mono plateforme (Windows)
 - Multi langage (VB, C#, C++ etc.)



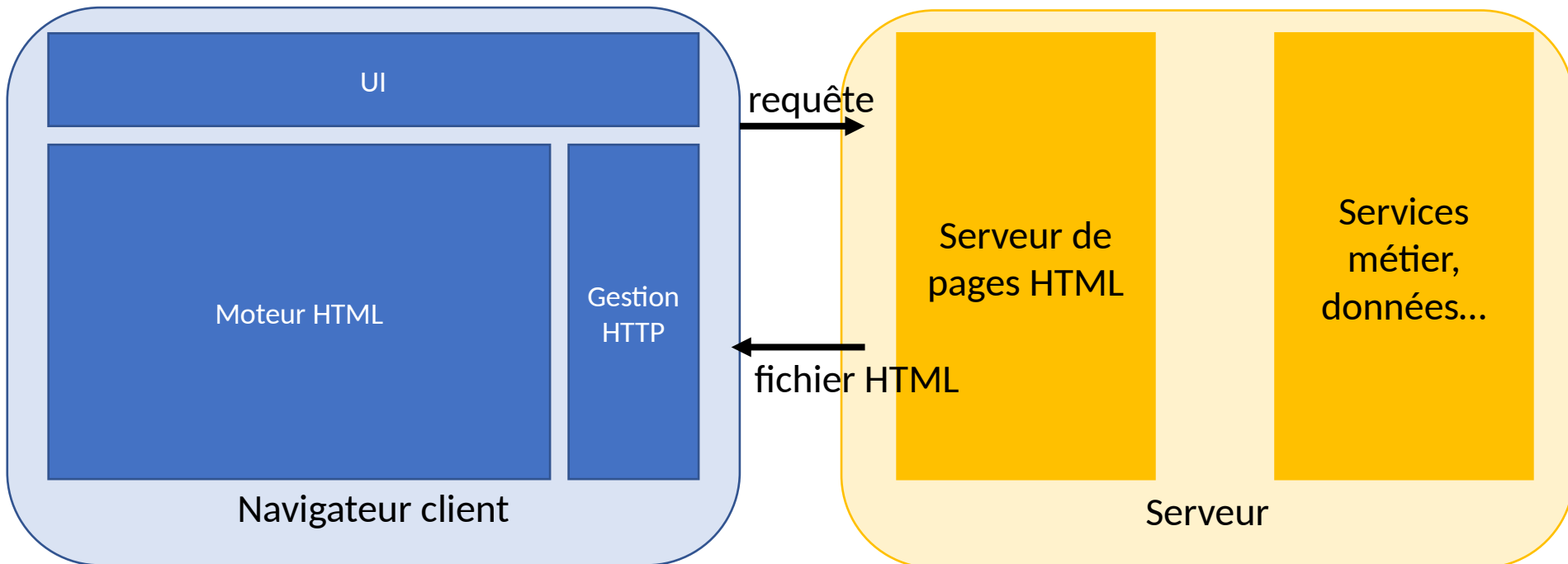
Bénéfices

- Évite l'utilisation de code « technique » dans le code applicatif / distinction entre la partie développement (métier) et la partie technique (administration)
- Principaux services fournis aux applications
 - accès aux BD
 - connexion aux applications d'entreprise
 - gestion des transactions, des sessions, des reprises sur panne
 - sécurité
 - passage à l'échelle
 - gestion par console d'administration unifiée
 - ...
- Support du déploiement de composants

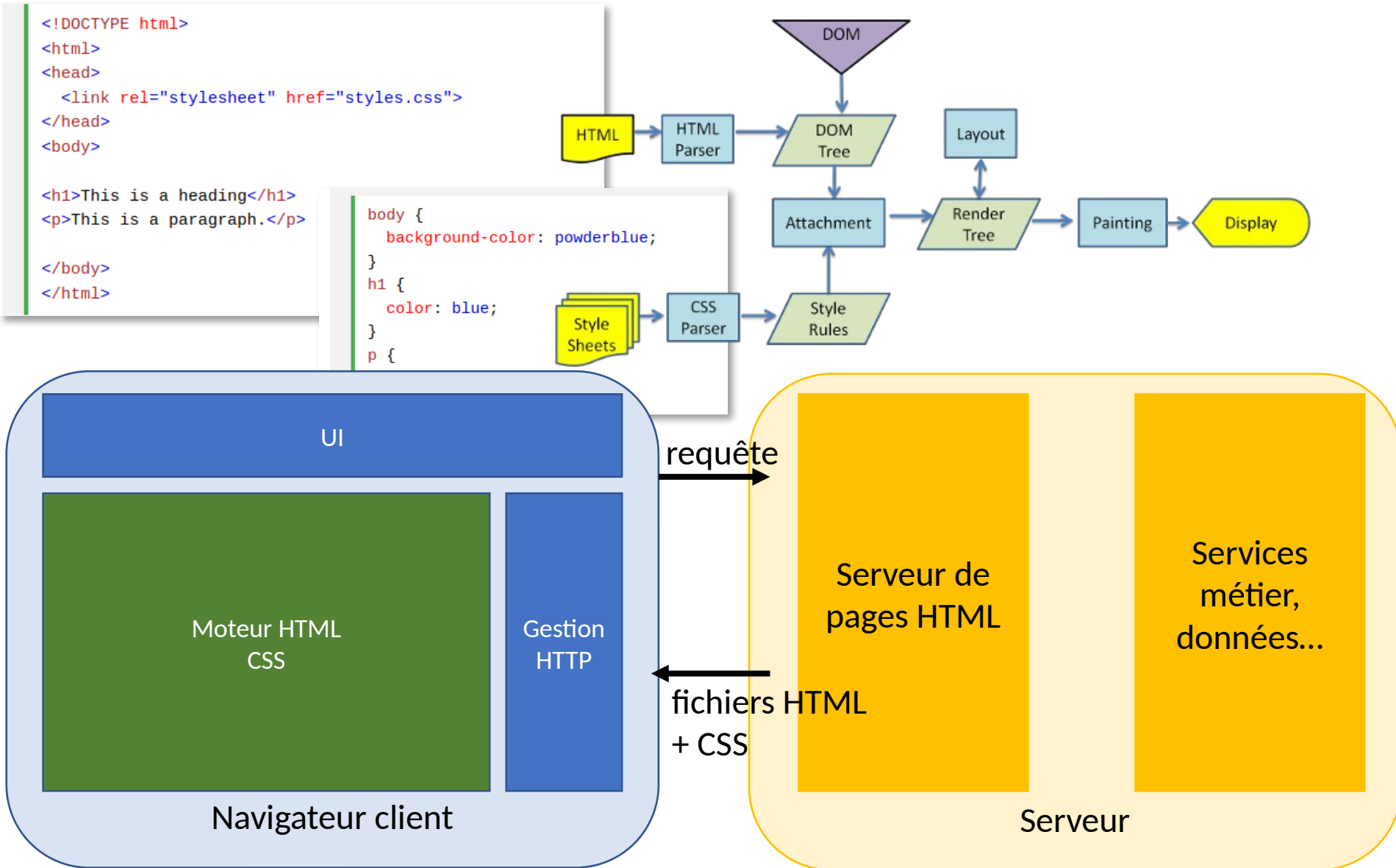
→ Pas important vers du code applicatif plus réutilisable !

L'application Web classique des années 90

- Un client léger (navigateur rudimentaire)
- Un protocole simple et sans état
- Exécution essentiellement sur le serveur

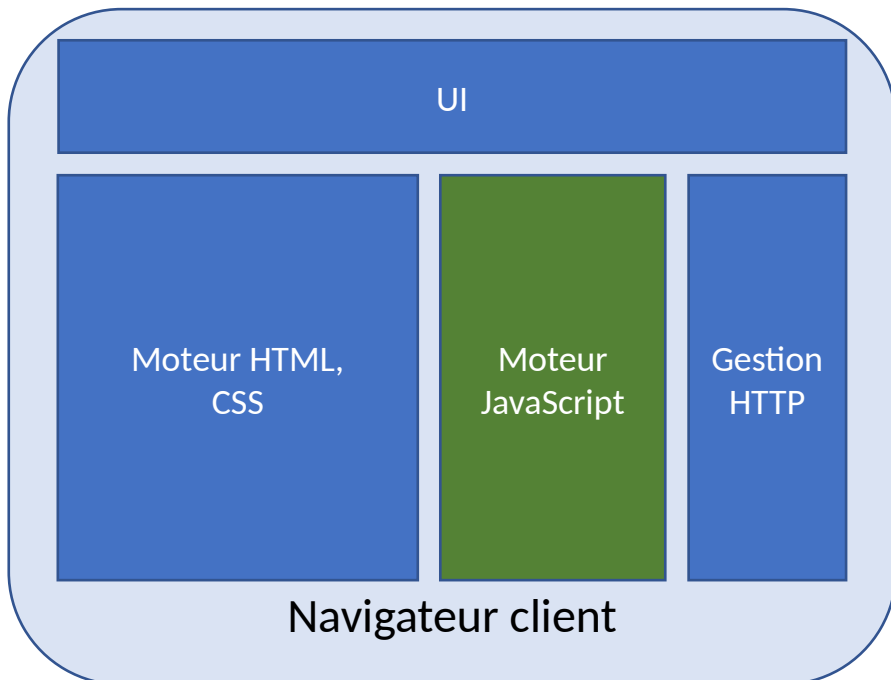


Mi-90's : une partie de la présentation est transférée au client



1996-97 : scripts interprétés par le client

- balise script référençant une petite application compilée
 - exécutée par le navigateur dans un espace réservé
- Dynamic HTML (DHTML)



```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<p>JavaScript can change the content of an HTML element:</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo">This is a demonstration.</p>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
}
</script>
```

My First JavaScript

JavaScript can change the content of an HTML element:

Click Me!

This is a demonstration.

My First JavaScript

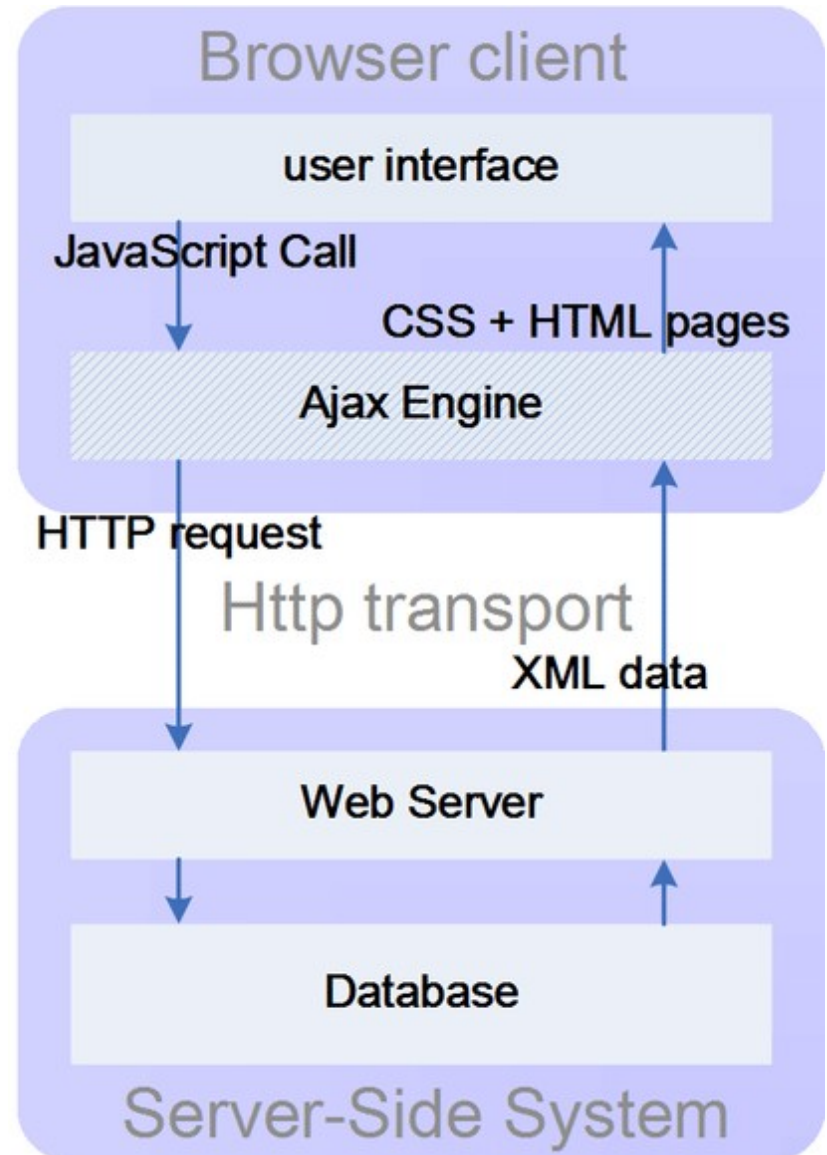
JavaScript can change the content of an HTML element:

Click Me!

Hello JavaScript!

2005 : patron « AJAX » : Vue et Contrôleur déportés sur le client

- *Asynchronous JavaScript + XML*
- Lorsqu'une action engendre la récupération de nouvelles données
 - La requête est faite au serveur en arrière-plan
 - A la réception, seule la portion de page impactée est rechargée
- Atouts
 - Moins de trafic et de charge serveur
 - Application sans effet de page
 - Possibilité d'applications très riches (par agrégation de

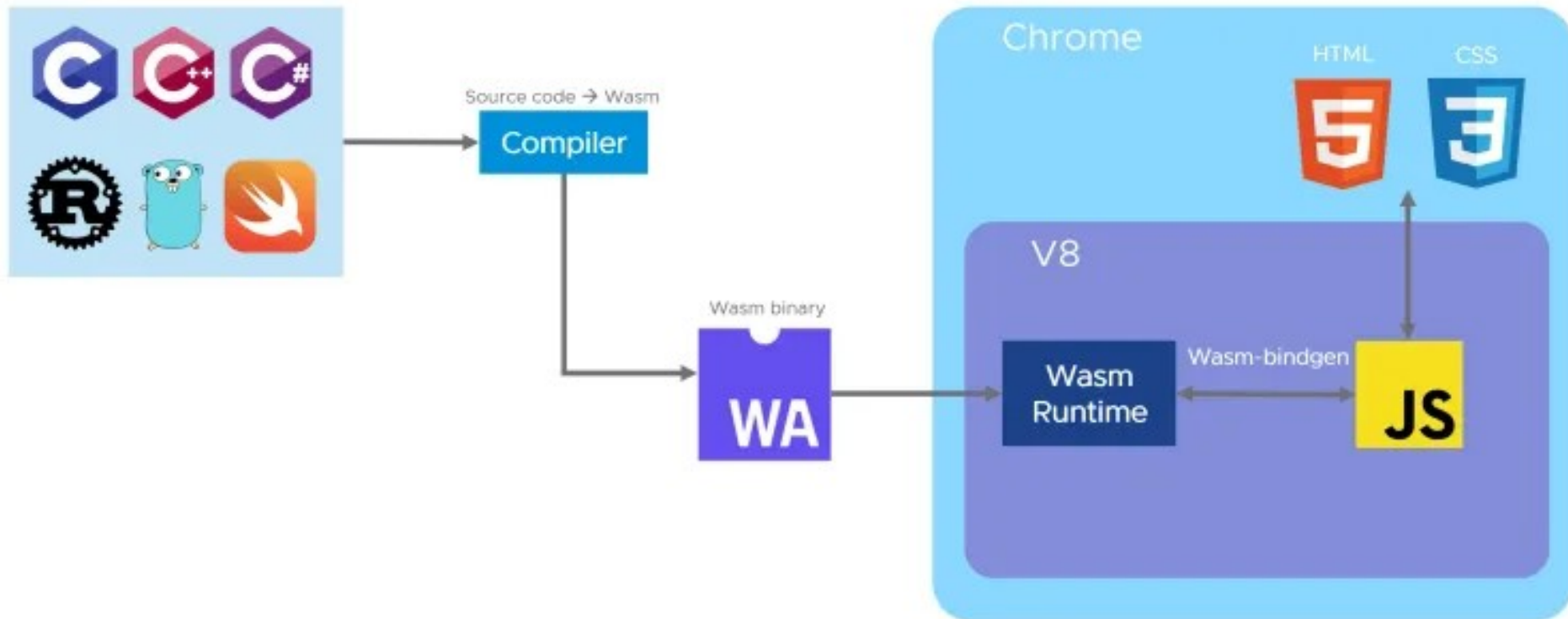


2010-15 : déclin des applets et apparition d'HTML 5

- Flash/Silverlight : problèmes de sécurité, interopérabilité et navigabilité → prise en charge abandonnée
- HTML5 : fonctionnalités pour des applications plus riches et interactives :
 - Contenus multimedia
 - Formulaires
 - Graphisme
 - Stockage de données local
 - Web Workers (scripts JavaScript qui s'exécutent en arrière-plan, en parallèle avec le thread principal de l'application avec lequel ils communiquent par messages)
 - Mobile : API de périphériques (Géolocalisation, caméra et micro, autres capteurs...)

2015 : Wasm (WebAssembly)

- Binaire exécutable dans les navigateurs
- Alternative ou complément à JavaScript



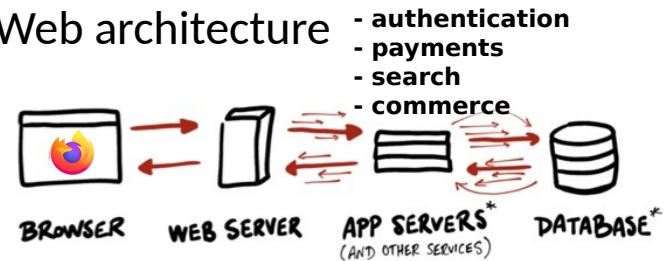
2015 : JAMstack

Transfère la charge du serveur vers le client autant que possible

Sites pré-rendus en pages statiques et données récupérées via des APIs

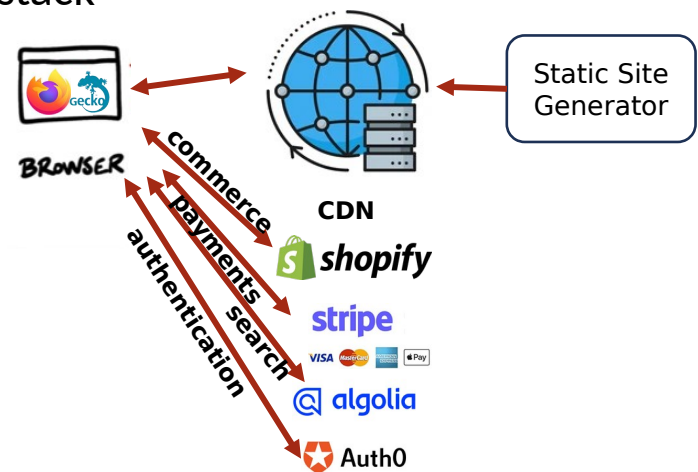
- **JavaScript** pour que le navigateur prenne en charge l'aspect et le comportement d'un site
- **APIs** pour que le client requête directement les données
- **Markup** statique chargé dans un CDN

Traditional Web architecture



* Typically ONE WEB REQUEST RESULTS IN LOTS OF APP AND DATABASE REQUESTS.

Jamstack

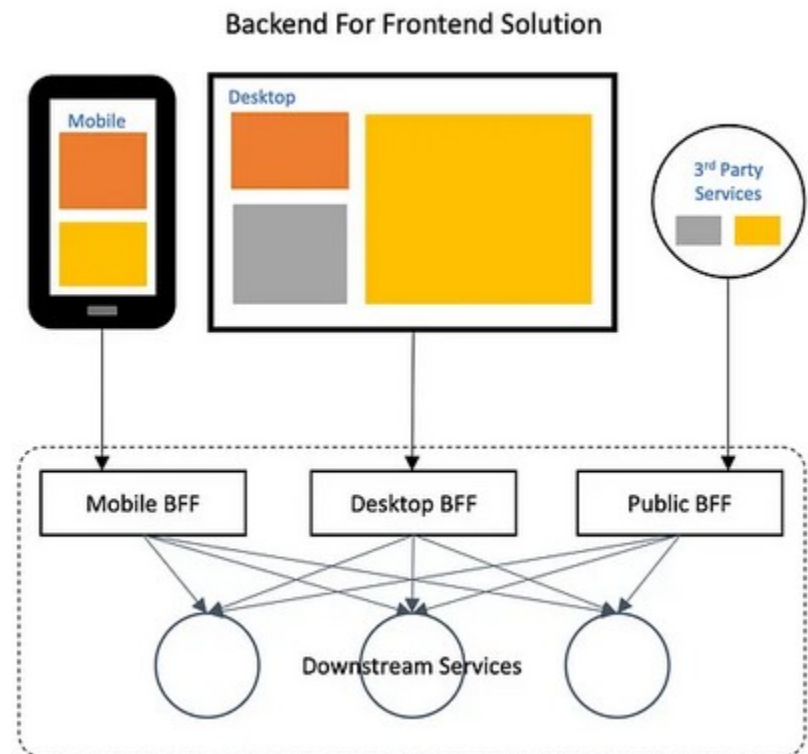
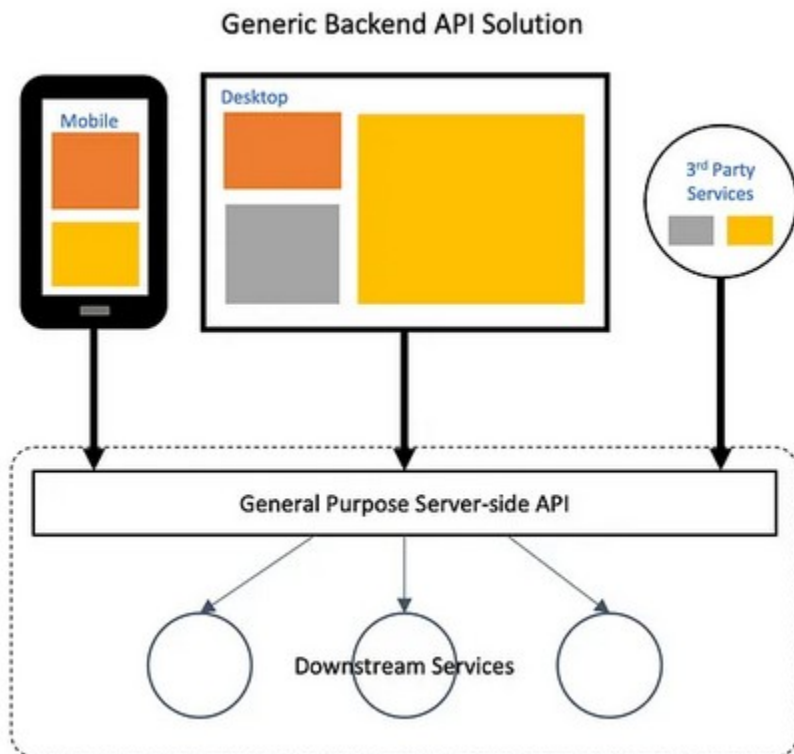


Jamstack

- Avantages :
 - Chargement rapide
 - Passage à l'échelle plus simple et moins cher
 - Les failles de sécurité du front ne menacent pas le back
- Inconvénients :
 - Mise à jour moins conviviale pour les éditeurs de contenus
 - Dépendant des systèmes tiers
 - Mal adapté aux sites demandant des fonctions complexes et dynamiques

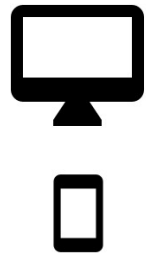
Backend For Frontend (BFF)

- Spécialise les données de l'API pour un type de front
 - Moins de manipulations par le navigateur
 - Moins de requêtes depuis le navigateur



2015 : GraphQL

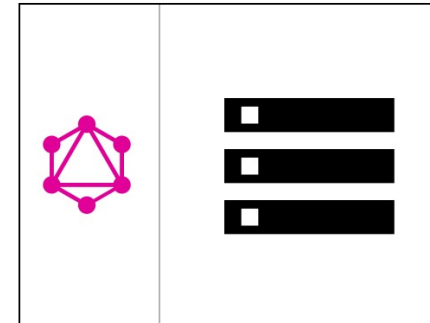
- Au lieu de points de terminaison spécifiques le back expose un schéma de données
- Le client requête uniquement les données dont il a besoin



```
query {  
  User(id: "er3tg439frjw") {  
    name  
    posts {  
      title  
    }  
    followers(last: 3) {  
      name  
    }  
  }  
}
```

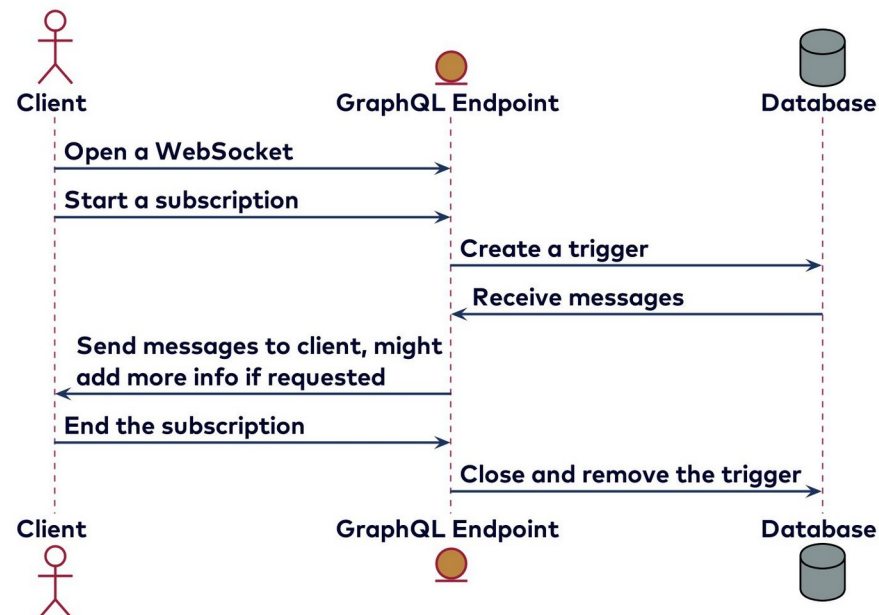
HTTP POST

```
{  
  "data": {  
    "User": {  
      "name": "Mary",  
      "posts": [  
        { title: "Learn GraphQL today" }  
      ],  
      "followers": [  
        { name: "John" },  
        { name: "Alice" },  
        { name: "Sarah" },  
      ],  
    }  
  }  
}
```



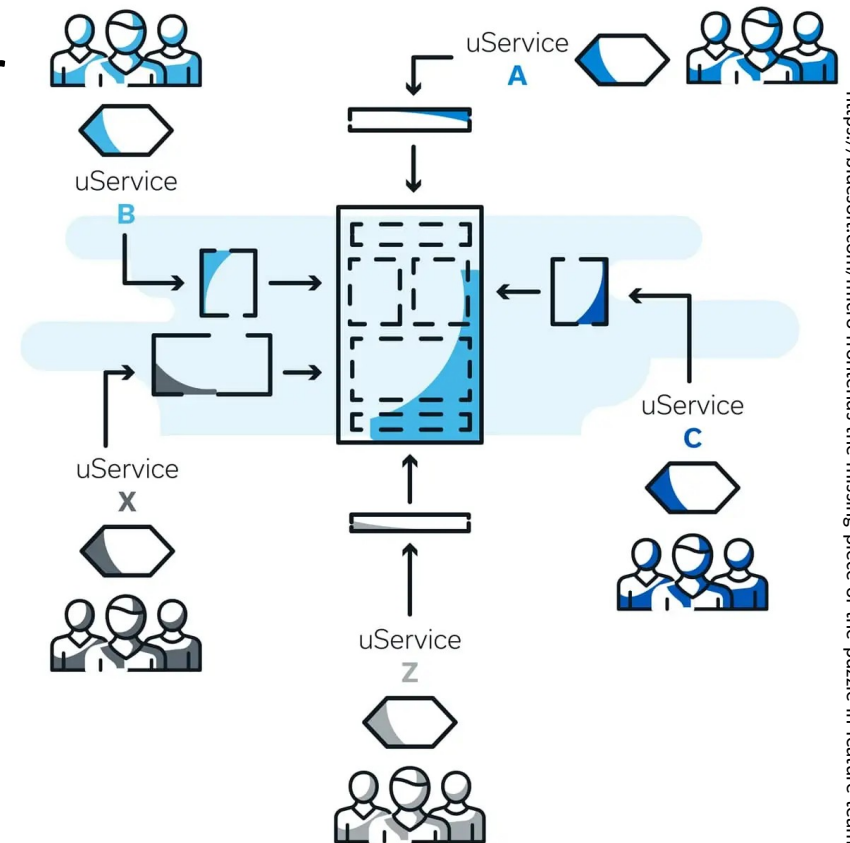
2016 : GraphQL Subscriptions

- Le client peut s'abonner à des événements spécifiques ; dès qu'un événement se produit sur le serveur, le serveur envoie des mises à jour automatiquement
- Exemple d'utilisation :
 - Notifications en temps réel
 - Flux de données dynamiques
 - Mises à jour collaboratives



2015 : micro frontends

- Page web assemblage de micro composants
- Chaque équipe développe son micro composant de la BD à l'UI
- Nécessite coordination pour
 - Homogénéité visuelle
 - Authentification partagée



Bilan : Architecture MACH

- M – Microservices
- A – API-first
- C – Cloud-native
- H – Headless
- Une tendance qui s’installe malgré les défis
 - Plus de composants, plus d’interactions... plus de complexité
 - En cas de bug, il faut naviguer entre plusieurs logs, services...
 - Sécurisation indépendante de chaque service + authentification et autorisation qui doivent être cohérentes à l’échelle du système
 - Nécessite des équipes plus complètes
 - Coûts cachés : plus d’abonnements SaaS, “pay-as-you-go” du cloud qui peut vite coûter cher sans surveillance