

# Frameworks web

Retour d'expérience et enjeux

# Chris Woodrow



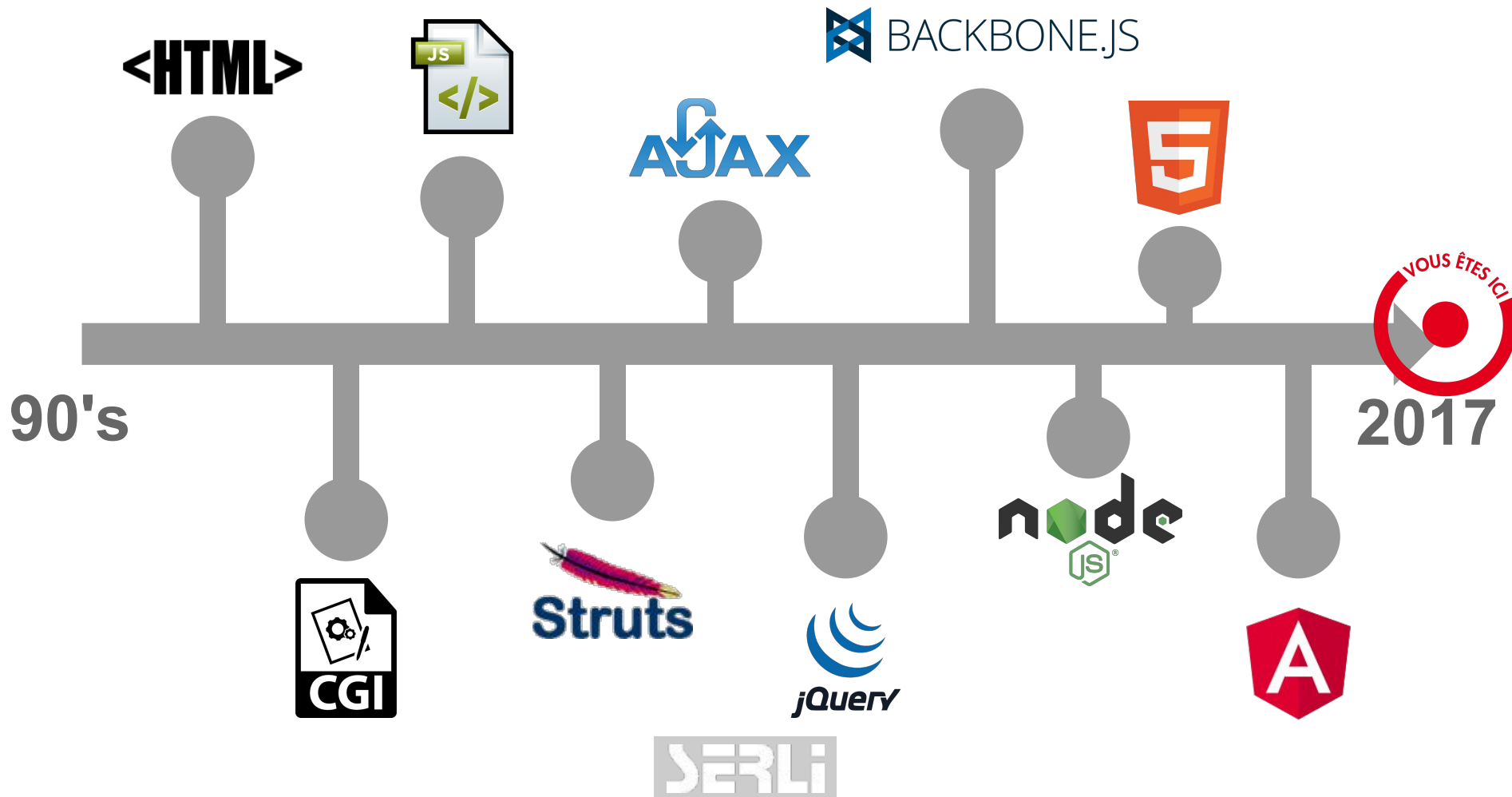
- **Développeur**
- Directeur Technique **@SERLI**
- Formateur
- Java, Archi, HA, NoSQL, ...
- **Data-Curious** (Big Data, ElasticSearch, DataViz ...)
- **Rock Addict**
- **@StrangeCousin** (*vraiment si vous avez du temps à perdre ... :D*)





Sondage

# Un peu d'histoire ...



# Frameworks MVC



...

- Navigation page par page
- Premiers vrais standards du développement web (après Servlet, JSP, CGI)
- Separation of Concerns
- Templates côté serveur (JSP, Velocity, FreeMarker, ...)
- Applications "Old School"
- Dans la plupart des langages



Jusqu' ici tout va bien !

*Mais les applications créées ne sont pas très dynamiques et  
ne correspondent pas aux attentes*

# Ajax + JQuery



- Il n'est plus nécessaire de recharger la page pour faire une requête HTTP
- Manipulation du DOM en JS
- JQuery simplifie entre autres le développement multi-browsers
- Effer du développement JS
- Généralement un framework MVC + des touches d'Ajax
- Développement assez spécifique (pas de vrai standard JS)



Mais Javascript 1.x ...

*Maintenance très très compliquée ... (même avec JQuery)*



Et là, beaucoup de "bonnes" idées des Javaistes  
pour éviter les problèmes liés à JS ...

*Masquons la complexité !*

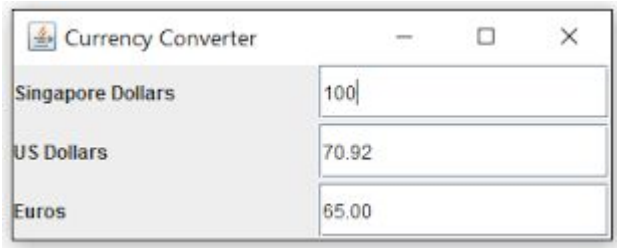
# GWT



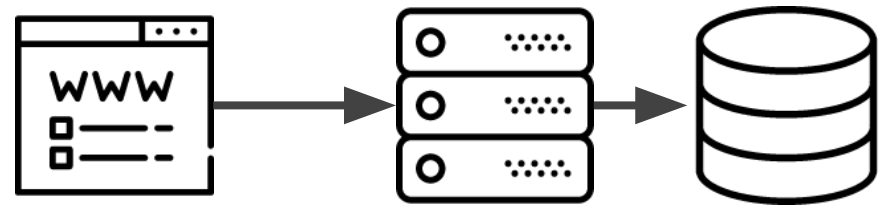
- Google Web Toolkit
- Utilisé par Google
  - Pas vraiment en fait ...
- Full Java
  - Transcodé en JavaScript
- Simple d'apparence pour les développeurs Java Swing
- Mais beaucoup de problèmes
  - Difficile à débbuger
  - Transcodage très long !

# Focus sur GWT

Point de vue du développeur



En vrai



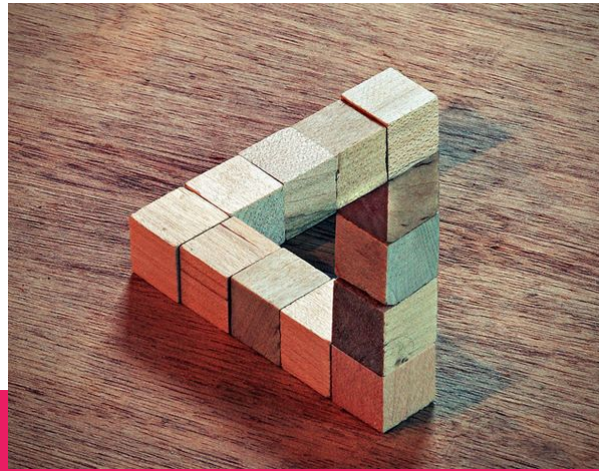
*"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."*

**Leslie Lamport**

# Frameworks orientés composants



- **Orientés composants**
  - Approche moderne
  - (vœux de) Réutilisabilité
- **Standard JavaEE (pour JSF)**
- **Stateful by design, berk !!!**
- **Le coût du développement augmente avec la complexité de l'application**
- **Là encore, la complexité sous-jacente est masquée**



Dans ce cas, masquer la complexité, c'est créer  
de la dette technique

*Accidents industriels ...*

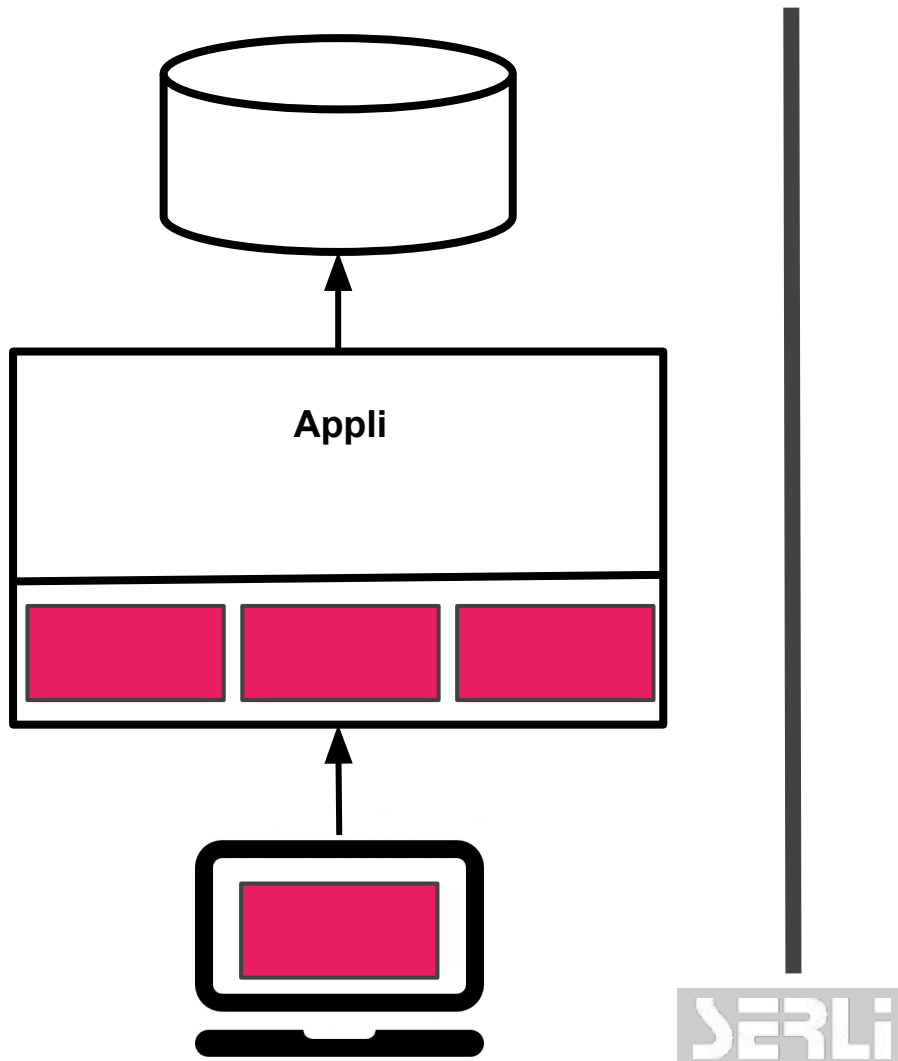
# BACK TO THE JAVASCRIPT

## Frameworks Javascript client

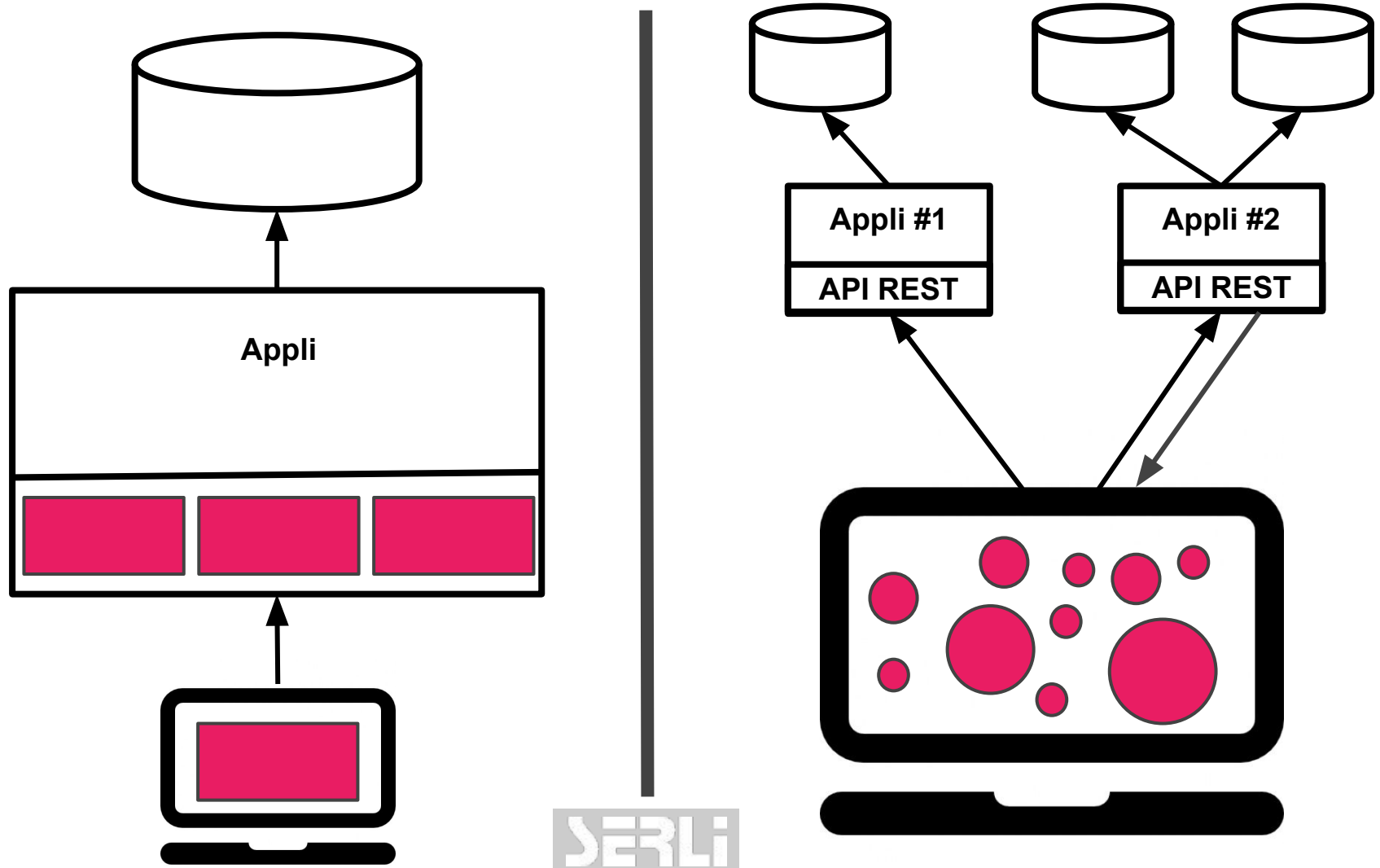


- Plusieurs projets ont émergé avec une grosse part de JS
  - Backbone, Ember, ...
  - Single Page Applications
- ES5 : amélioration du langage
- Le Push server s'est démocratisé et standardisé
- NodeJS nous apporte un outillage de qualité
- Le standard Web Components : tentative de normalisation, malheureusement infructueuse à ce jour

# Évolution des architectures web



# Évolution des architectures web

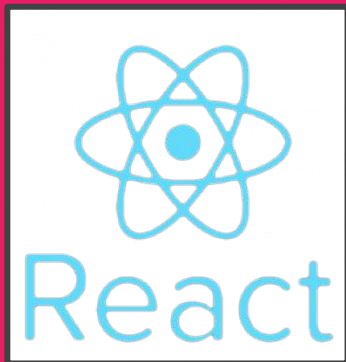


# Angular



- Open sourcé par Google (et utilisé, ex : *Google Trends*)
- Le premier framework web moderne qui a été popularisé
- Simple au premier abord
- Très (trop?) complet
- Approche MV\* puis orientée components (1.5+)
- Quelques problèmes de performances, corrigés depuis la v1.5+
- A partir de la v2, utilisation de TypeScript

# React.js



- Open sourcé par Facebook
- Orienté composants
- Pas de magie
- Basé sur un DOM virtuel : très performant !
- Bullet Proof (en prod sur Facebook)
- Gère uniquement la partie vue
  - Redux : gestion d'états, unidirectional dataflow
- React Native pour le développement mobile

# Vue.js



- Dernier arrivé, plutôt bien adopté
- Orienté Components
- Reprends des concepts du standard Web Components
  - Templates, CSS et logique dans les composants
- Beaucoup de points communs avec React



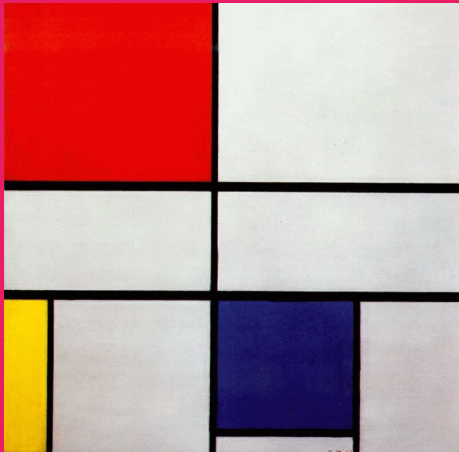
Quelques points d'attention

# #1 Apprenez le JS



- JS est LE langage du développement web
- Un des seuls éléments qui sera encore dans là dans 5 ans
- Développement productif (feedback rapide)
- C'est un langage outillé
  - Transcodage (TypeScript) ... ça reste proche et rapide ...
  - Linter (analyse statique du code : style & bugs)
  - Tests automatisés (unitaires, end-to end)

## #2 Attention à l'abstraction



- Une abstraction est un nouveau paradigme pour quelqu'un qui connaît le web
- Un peu d'abstraction ne fait pas de mal, il faut savoir doser :D
- Choisissez une technologie qui reste proche du web

# #3 Utilisez les bons outils



- Utiliser un framework JS moderne permet d'organiser son code (components)
- Les gestionnaire d'états (Redux) permettent de gérer des applications complexes
- Les outils de constructions sont désormais standardisés et simples (webpack)
- Les outils sont au service des développeurs et non l'inverse

## #4 Attention à l'écosystème



- L'écosystème est particulier, les outils sont vite adoptés et vite abandonnés
- La communauté JS à tendance à "réinventer". Par exemple : beaucoup de tentatives niveau de la gestion des dépendances et du build
- Tout n'est pas toujours sec, il n'est pas rare d'utiliser une version 0.x en production

# #5 KISS\*

\* Keep It Stupid, Simple



- Optez pour une approche micro-services :
  - Complexité raisonnable
  - Couplage faible
  - Réécriture possible
  - Évolutivité de la stack
- L'écosystème qui évolue très vite
  - La veille est indispensable !

# #6 Misez sur les compétences



- En 2017, un développeur est polyglotte, il adapte l'outil au besoin et non l'inverse
- Les développeurs fullstack développent à la fois le frontend et le backend
- Attention : développeur != intégrateur web



Questions ?