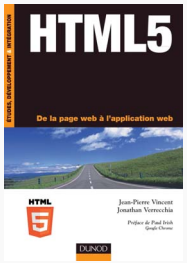
Performance Web
Il se passe quoi après le backend ?

Jean-pierre Vincent – Expert Webperf indépendant

Hello !

Jean-Pierre Vincent

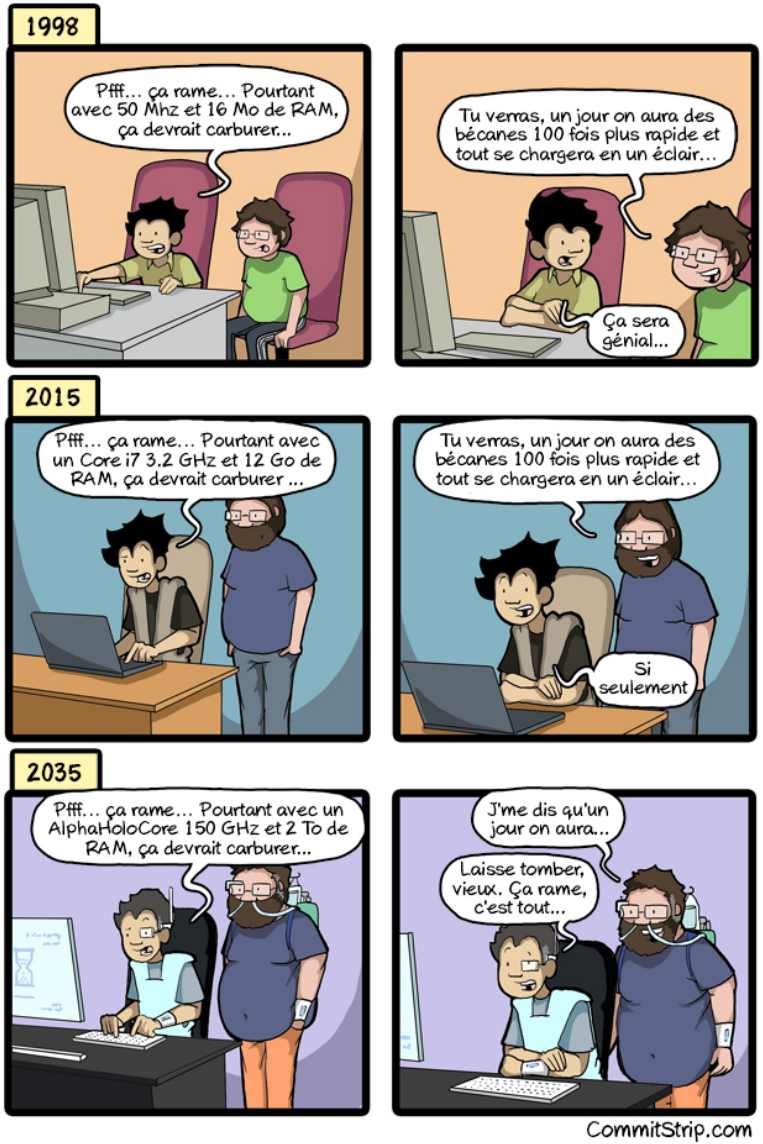
BrainCracking



- Consultant Webperf
- Formations Webperf
- Architecte JS
- Développeur Web

- Co-organisateur
  - Paris Webperf Meetups
  - Conférence FR le 20/09

we ♥ speed



http://www.commitstrip.com/fr/2015/08/28/murphys-take-on-moores-law/

POURQUOI LA PERF ?

Les enjeux de la Webperf

- L’expérience utilisateur 👨‍💻 👩‍💻
- L’écologie ♻️  = 🤑
- L’accessibilité ♿

🚀 = 🤑

- E-commerçants :
  - Walmart : **+1 s = -50% de conversion**
  - Rue Du Commerce : chargement ÷ 2 = **+56% de conversion**
- Modèles gratuits :
  - Financial Times : **1 s = -24% d'engagement**
  - Youtube : « video first » = **ouverture** à d'autres pays

wpostats.com

Performance Web ?

- On ne parlera **pas** backend :
  - ~~Tenue de charge~~
  - ~~Temps de réponse serveur~~
- À **surveiller** quand même car :
  - Pas de backend, pas de site …
  - Énorme influence **SEO** !

Au menu ce soir

1. Mesurer

- Que mesurer
- Outils

2. Optimiser

- JavaScript
- Les fonts
- Les images
- Les tiers
- HTTP/2

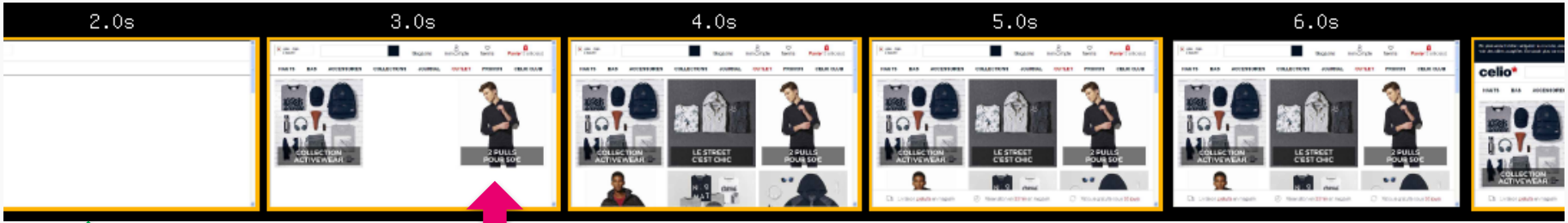
« On ne corrige pas ce qu’on ne voit pas »

Le gouvernement chinois

MESURE, SURVEILLER

1. L’utilisateur peut-il voir ?

SpeedIndex : 4780 ms

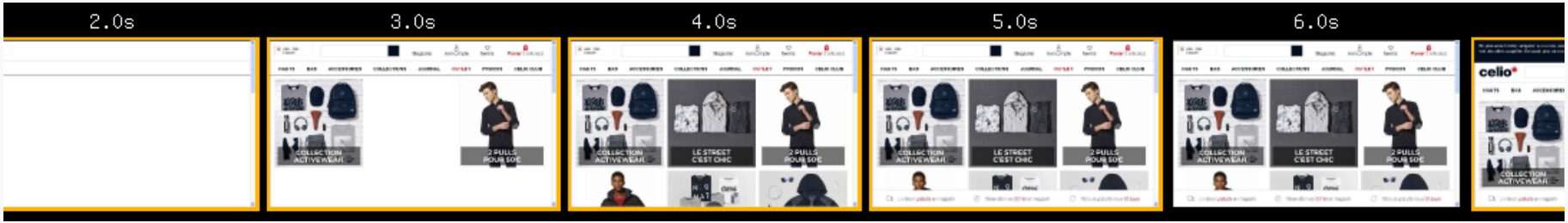


Custom metric : Images Produit : 3 s

First Paint / FCP : 2 s

WEBPAGETEST

2. L’utilisateur peut-il interagir ?



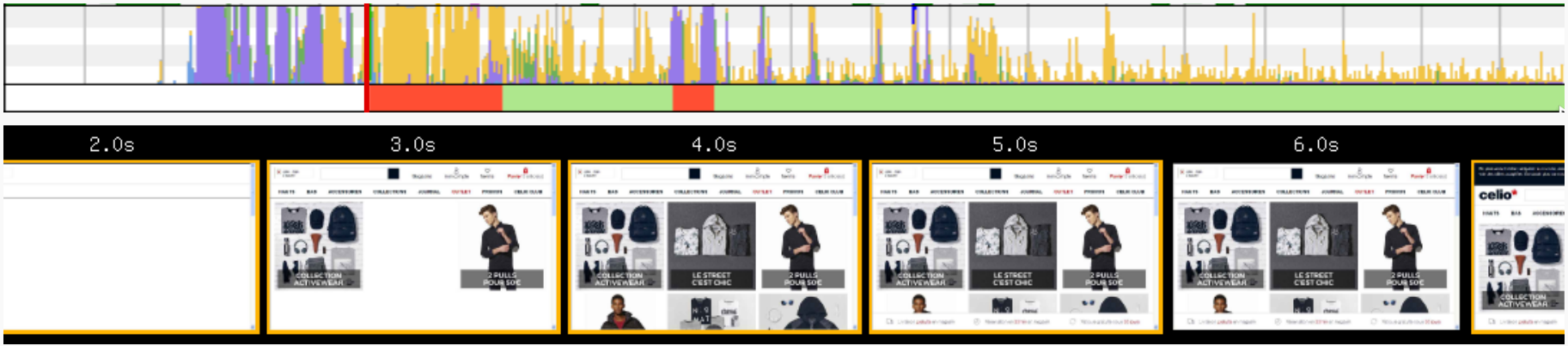
Custom metric
Moteur de recherche : 6,5 s

DomContentLoadedEnd : 5 s

Custom metric
Navigation : 5 s

WEBPAGETEST

3. L’interaction est-elle fluide ?



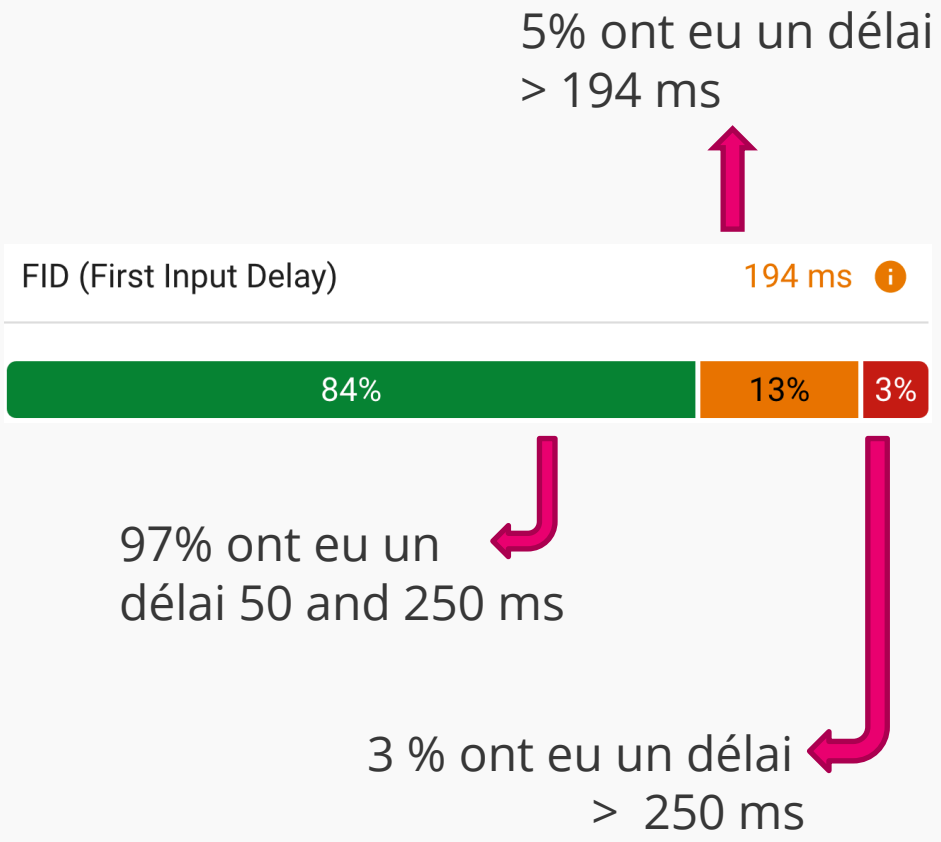
- Temps de réaction < 100 ms
  - **First Input Delay** ➔ PageSpeed Insights
- Animations fluides : **60 FPS**

PageSpeed Insights – Chrome User eXperience Report

Google Page Speed Insights nous dit si l'utilisateur subit des gels de l'interface.

Voici comment lire ce graphique :

Les utilisateurs de Chrome mobile , sur les 30 derniers jours, au moment d'interagir, ont eu un eu un délai de réaction :



developers.google.com/speed/pagespeed/insights/

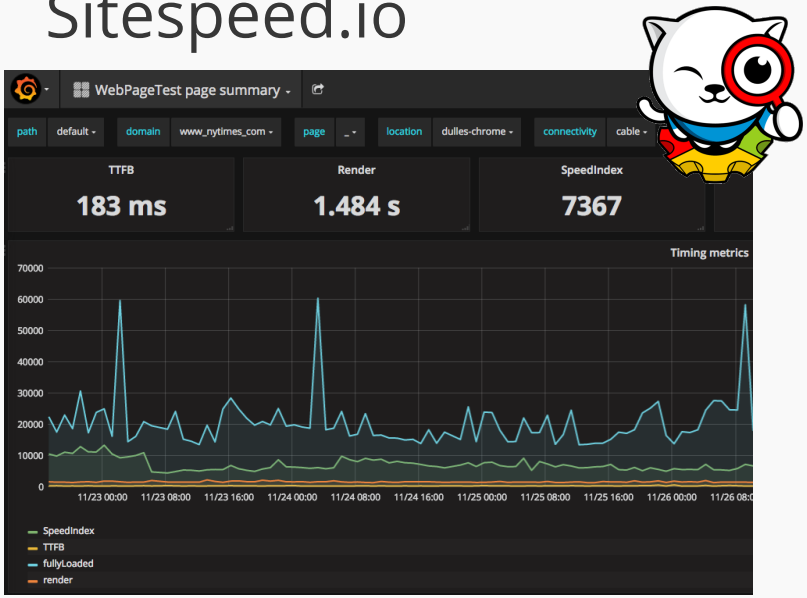
Custom metrics, via le standard

- `<img src="hero-image" onload="performance.mark('hero image')" />`
- `performance.getEntriesByType('paint')`
- `performance.getEntriesByType('resource')`
  - `[{ name: "https://…", responseEnd: 304.1, transferSize: 42000, … }]`
- `performance.getEntriesByType('longtask')`

Surveiller / Visualiser

En synthétique

- Sitespeed.io

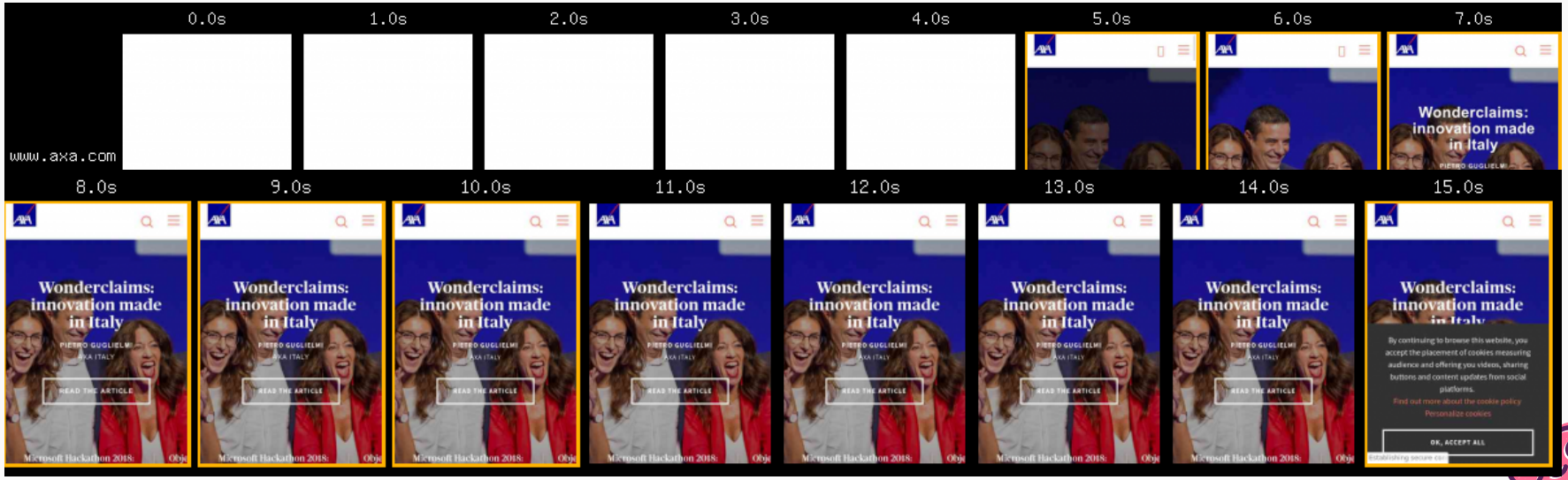


Les vrais utilisateurs

- Gratuitement
  - Google Analytics
  - PageSpeed Insights
  - BoomerangJS +
    - BasicRUM
    - Piwik
- Saas

Exploiter ses analytics, leur donner du sens

1.	France	**1 133 171** (32,07 %)	2,83	4,01	5,41
2.	United States	**330 187** (9,35 %)	4,35	5,44	6,83
3.	United Kingdom	**323 294** (9,15 %)	2,78	3,57	4,88
4.	India	**190 226** (5,38 %)	6,30	9,51	13,20
5.	Germany	**143 497** (4,06 %)	3,42	4,20	5,41



Conclusion : mesurer l'UX

- Mesurer
  - Investir dans les **custom metrics**
  - Métriques génériques : à sélectionner
- Analyser et automatiser
  - **Webpagetest** : le plus complet
- Surveiller
  - **Sitespeed.io**, ou solutions Saas payantes
  - Real User Monitoring : analytics, PageSpeed Insights, Saas

Au menu

✓ **Mesurer**

✓ Que mesurer

✓ Outils

Optimiser

- JavaScript
- Les fonts
- Les images
- HTTP/2
- Les tiers

On n’a pas tous un iPhone X

L’OPTIMISATION DE JAVASCRIPT

@theystolemynick

Théorie : combien coûte 1 Mo de JS ?

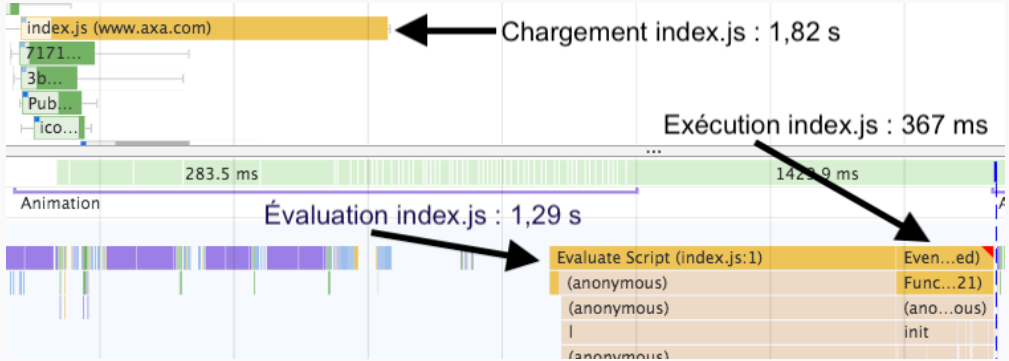
	Temps d'exécution	Prix 2018
Motorola G4	1 800 ms	200 €
Galaxy S7	1 000 ms	400 €
iPhone 7	350 ms	650 €

5 fois plus lent

D'autres stats, par Addy Osmani : http://bit.ly/jscost-wpt

JavaScript

- Un gros fichier JS est une double peine :
  - Long à **charger**
  - Long à **exécuter**



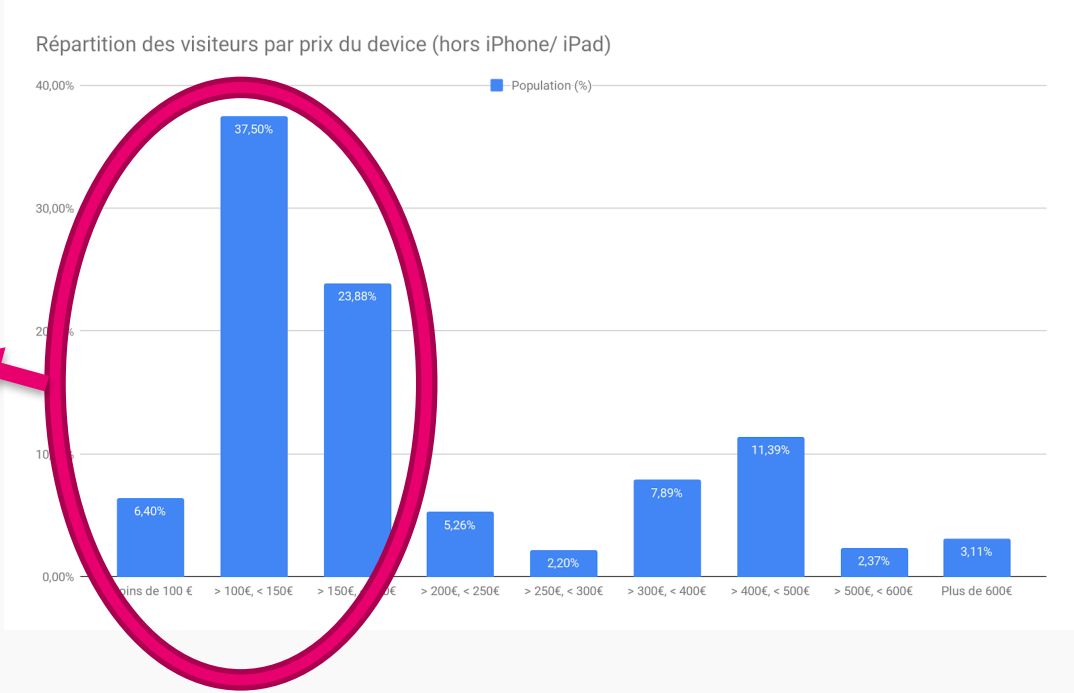
- Ex: Bundle Axa 2018 :
  - 1,3 Mo à charger
  - 4,2 Mo à exécuter

Galaxy Note 3 (200€)

Connaître ses utilisateurs

Axa.com (cible mondiale)

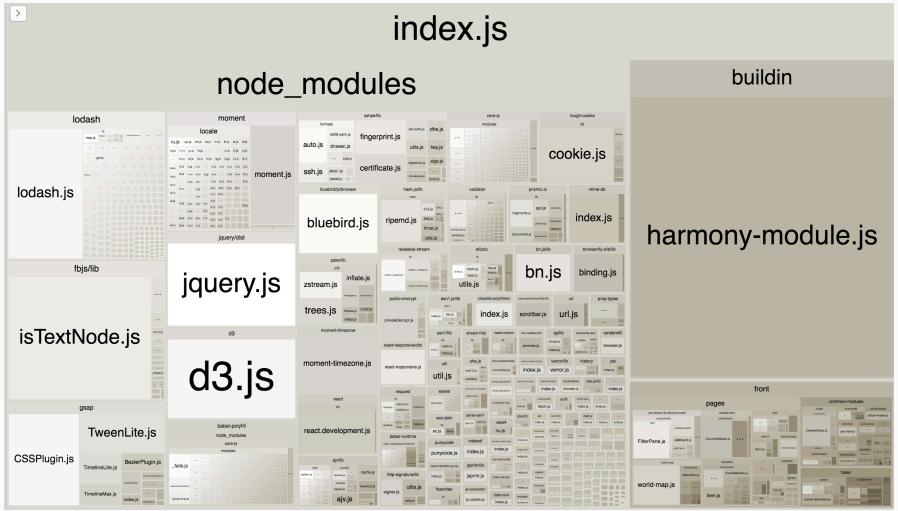
- iPhone ou iPad : 51% des visiteurs
- 30% des visiteurs ont un smartphone valant **moins de 200 €**



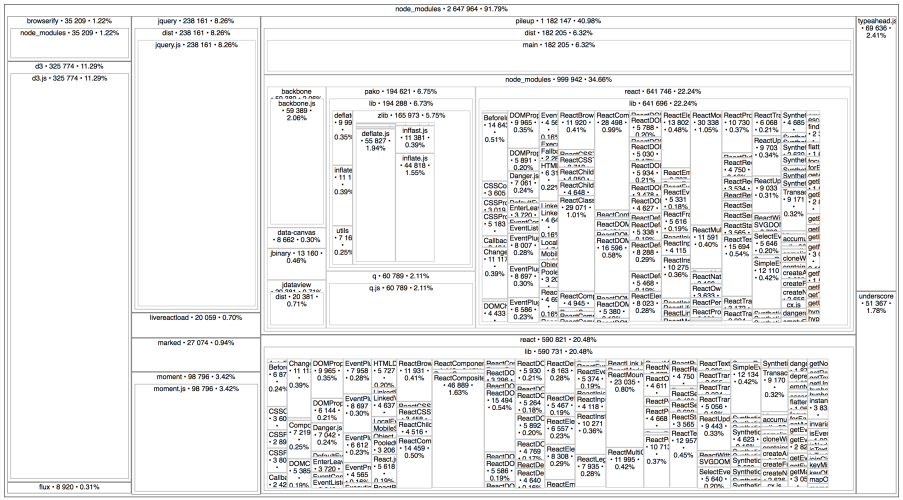
Pour janvier-octobre 2018, modèles collectés par GA (top 100), sur Axa.com, prix collectés manuellement.

Outils d’analyse

Webpack Bundle Analyzer

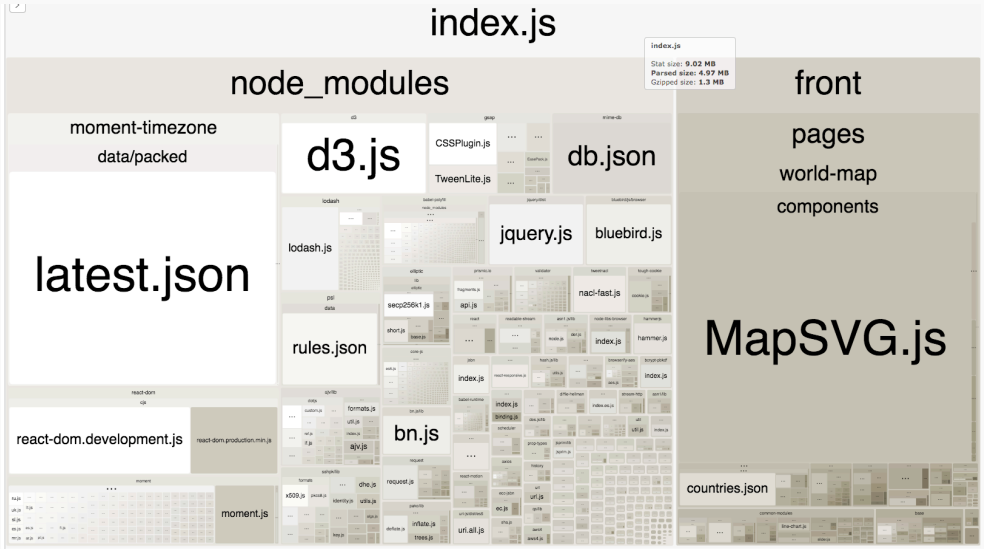


Source map explorer



Bundle phase 1 : dégraisser

- Sortir les SVG du bundle
  - Bonus : jamais chargée sur mobile ➔ -1 Mo
- Passage du build en mode optimisé
  - React ➔ -200 Ko
- Upgrade des outils de build
  - webpack, babel, polyfill, coreJS ➔ -100 Ko
- Optimisations propres aux dépendances :
  - MomentJS, ➔ - 180 Ko
  - moment-timezone ➔ - 900 Ko
  - Lodash, ➔ -90 Ko
  - Suppression request et bluebird ➔ -275 Ko
  - jQuery ➔ -15 Ko
  - …



webpack-bundle-analyzer

Bundle phase 1 : aider le tree-shaking

Savoir importer

- ~~`import d3 from 'd3'`~~
- `import {scaleLinear} from "d3-scale"`
- **babel-plugin-transform-imports**
  - Ré-écrit les `import`
  - **Configuration individuelle** des librairies
- Deep tree-shaking
  - Rollup : de base
  - Webpack : tenter deep-scope-analysis-plugin

Chercher les version éclatées des librairies

Versions **modules ES**

- underscoreJS / lodash ➔ lodash-es
- asyncJS ➔ async-es
- Indice: champs `"module"` dans `package.json`

Versions **lite**

- Gsap : versions *Lite, à aller chercher dans des fichiers séparés
  - ~~`import { TweenMax } from 'gsap'`~~
  - `import { TweenLite } from 'gsap/all'`
- jQuery, version lite

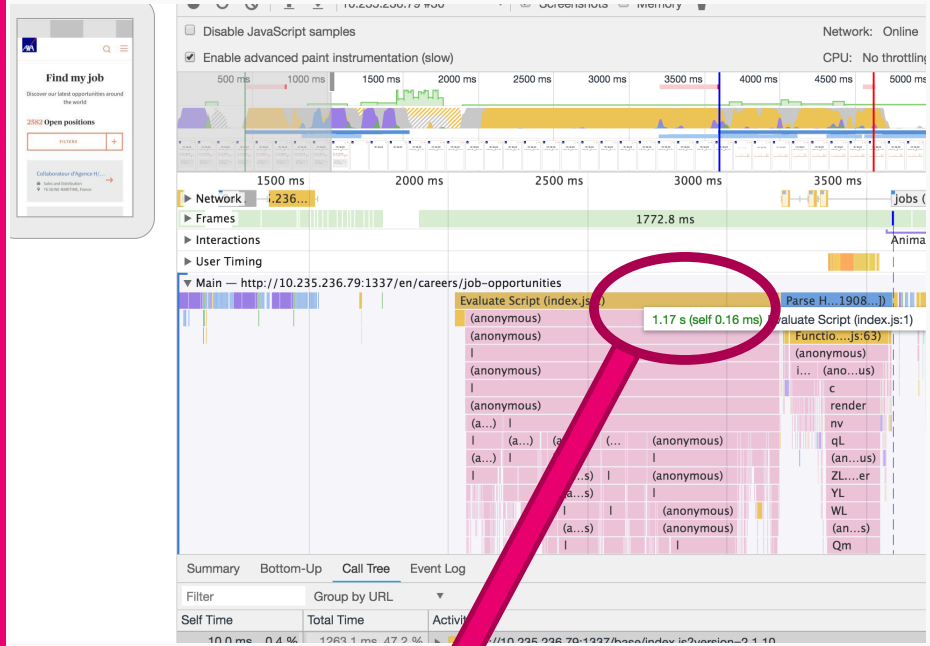
Bundle phase 2: alléger lib par lib ...

- momentJS ➔
  - Suppression des locales embarquées (plugin webpack)
  - Remplacement par Luxon ou date-fns
- moment-timezone
  - Hack pour réduire le nombre de timezones
  - Après IE : utiliser l'API du navigateur
- React ➔ Preact, Inferno, Nerv ...
- Ramda ➔ plugin babel
- request ➔ préférer whatwg-fetch ou Axios, avec des dépendances moins lourdes
- Bluebird ➔ Promise native ou en polyfill
- Validator, react-motion :
  - Préparer le tree-shaking ne suffit pas : `import { isEmail } from 'validator'`
  - Configurer le tree-shaking avec **babel-plugin-transform-imports**
- ...

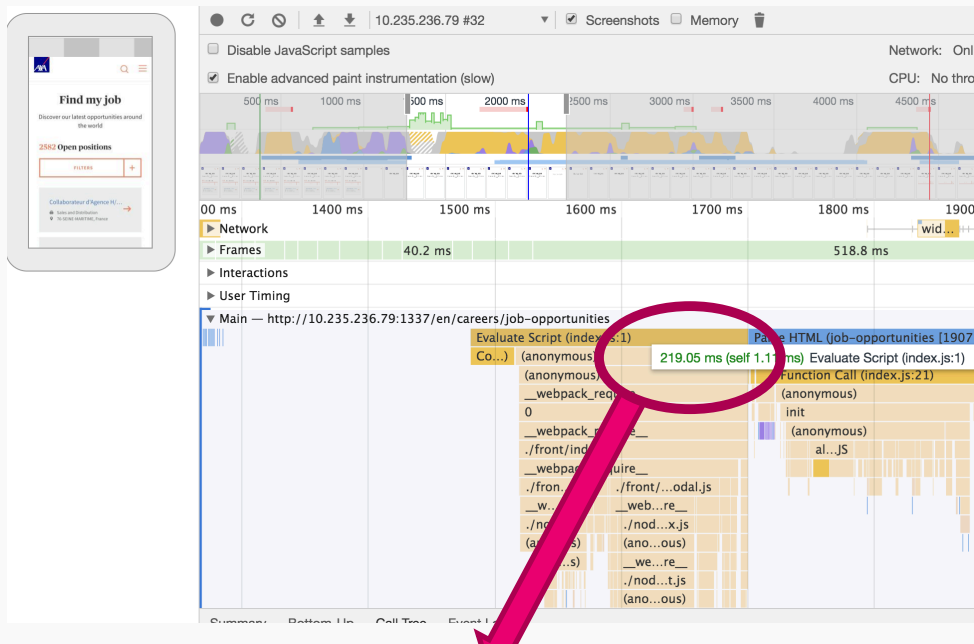
Bundle phase 3 : ES 2019 ➔ ES 5 ?

- Upgrade aux dernières versions de Webpack (4) et babel (7)
- Mode production de webpack
  - Minifie, compresse, externalise sourcemaps …
  - React et d'autres passent aussi en mode production
- Ne plus importer babel-core ou babel-polyfill
  - Configurer ses navigateurs cible (browserlist)
  - Utiliser @babel/preset-env
- Pourquoi pas : differential serving (1 bundle par niveau de support ES2019 du navigateur)

Résultat ?



1,17 seconds d'exécution pour 3,8 Mo



200 ms d'exécution pour 973 Kb

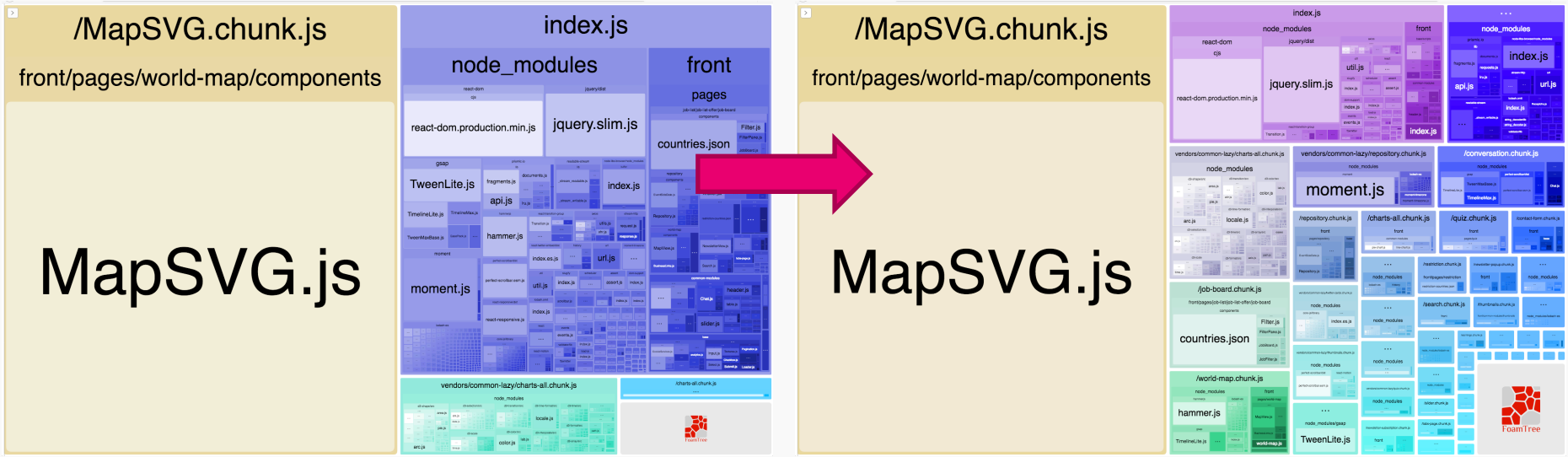
Samsung Galaxy Note 3 (200€)

Bundle phase finale : diviser pour régner

Options habituelles :

- 1 bundle par page / route
  - Réservé aux sites avec beaucoup de code métier
  - Avec HTTP/2 + ES `import()` + plugin babel + Webpack chunk + , **un mini-bundle par module devient réaliste et efficace**
- 1 bundle par module
  - React : Lazy + Suspense, React-loadable
  - VueJS, JS natif …

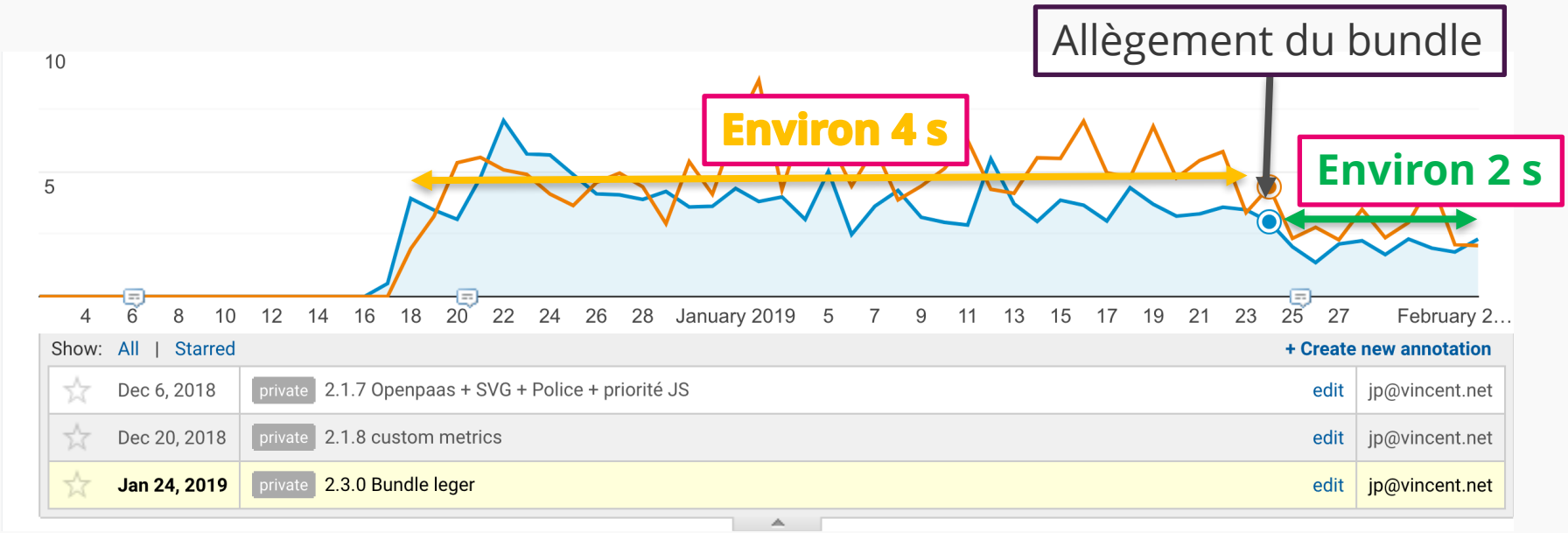
Bundle Axa phase 3 : diviser pour régner



Chaque page charge entre **2 et 4 fois plus** de fichiers qu'avant.

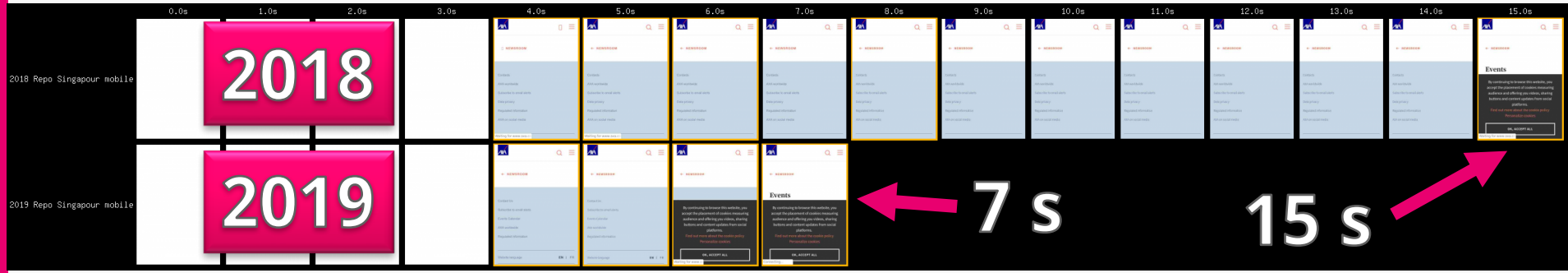
Mais chaque page charge entre **2 et 4 fois moins de poids**.
Le temps de parsing est **divisé par 2 ou 3**.
Même sans HTTP/2, notre **code métier s'exécute plus tôt**.

Résultats



Moyenne de l'affichage du dernier composant (mondiale, moyenne hebdo)

Concrètement, sur une page React

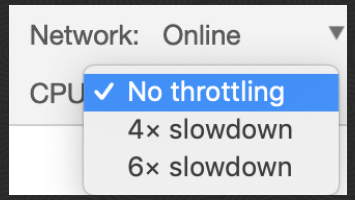


Simulation mobile, Inde, avant / après

Conclusion JS

- Connaître vos utilisateurs
  - Quel mobile ?
  - Quelle connexion ?

- Vérifier les performances sur de **vrais mobiles**
  - chrome dev tools + cable USB = ✨ ✨
  - Psst, ceci est un petit mensonge ➔



Au menu

✓ **Mesurer**

✓ Que mesurer

✓ Outils

Optimiser

✓ JavaScript

- Les fonts
- HTTP/2
- Les tiers
- Les images

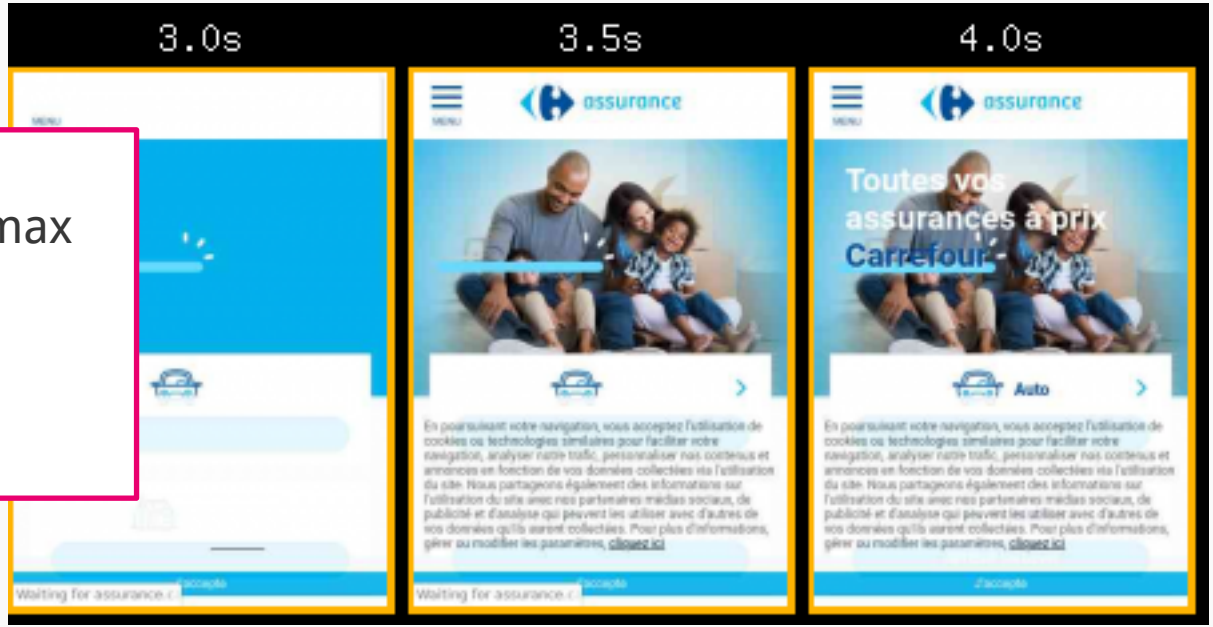
LES POLICES DE CARACTÈRES

Que faire en attendant la police ? 🚓

Chrome, Fx : **texte invisible** pendant **3 s** max

iOS : **page blanche**

IE 9+ : **swap**



Textes invisibles

1 font = 1 fichier

Optimiser les fichiers

- Utiliser les formats :
  - Eot : servir compressé
  - Woff : déjà compressé
  - **Woff 2** : +30% de compression
- Réduire le nombre de caractères à une culture : « **subsetting** »
- Les héberger sur **ses serveurs** ➔ 

fontsquirrel.com/tools/webfont-generator

Déclaration optimale

```
@font-face {
  font-family: 'Police';
  font-weight: normal;  font-style: normal;
  src:
      url('police.woff') format('woff');
}
…
.element { font-family: 'Police'          ; }
```

Tenter sa chance

L'ordre de déclaration compte

Stratégie d'attente.
Swap = affichage immédiat

Police de repli

Que faire en attendant la police ? 🚓

Fichiers optimisés : - 1 seconde

Quelques vérifications

- Les variantes sont-elles utiles ?
  - ✓ Tester « faux bold » et « faux italic »
  - ✓ Le fondeur fournit-il une version « variable font » ?
- Le fichier est-il léger ?
  - ✓ Objectif : < 25Ko
- Licence :
  - ✓ L'hébergement sur nos serveurs est-il autorisé ?
  - ✓ Peux-t-on modifier les fichiers ?

Conclusion Polices

- Choisir une stratégie d’affichage
  - La plus commune : swap
- Enlever l’inutile
  - Les polices ou les variantes qui n’amènent rien
  - Les caractères d’autres cultures

Au menu

- ✓ **Mesurer**
- ✓ Que mesurer
- ✓ Outils

Optimiser

- ✓ JavaScript
- ✓ Les fonts
- ✓ HTTP/2
- ✓ Les tiers
- • Les images

HTTP/2

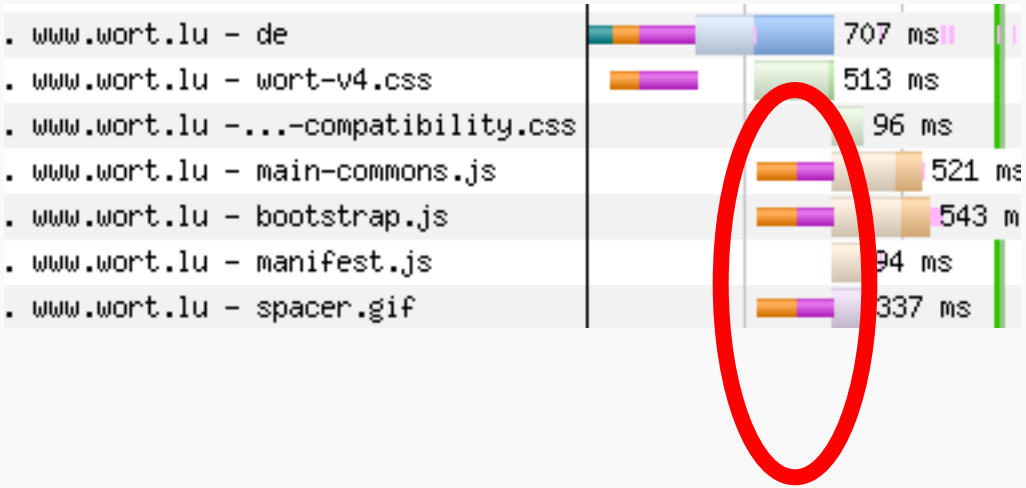
Vous avez aimé le 1 ?

Le 3 est mieux !

HTTP/2 en théorie : pure magie 🧙‍♀️

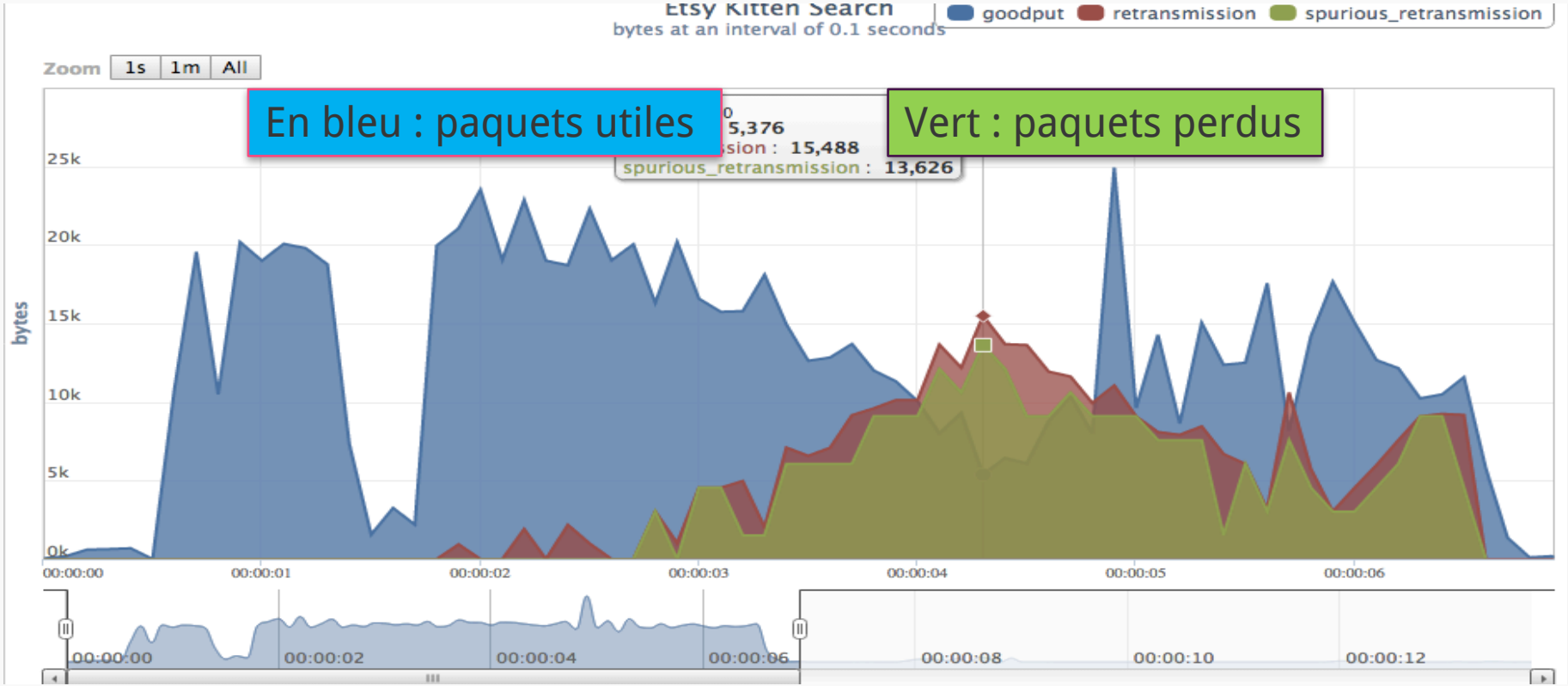
http://www.http2demo.io/

HTTP/2 en pratique : pas trop mal



H2 va « juste » supprimer ces renégociations TLS

HTTP/2 parfois : 💥



L'exemple d'Etsy : HTTP/2 + **multi-domaine** === congestion réseau

HTTP/2 PUSH : opportunité

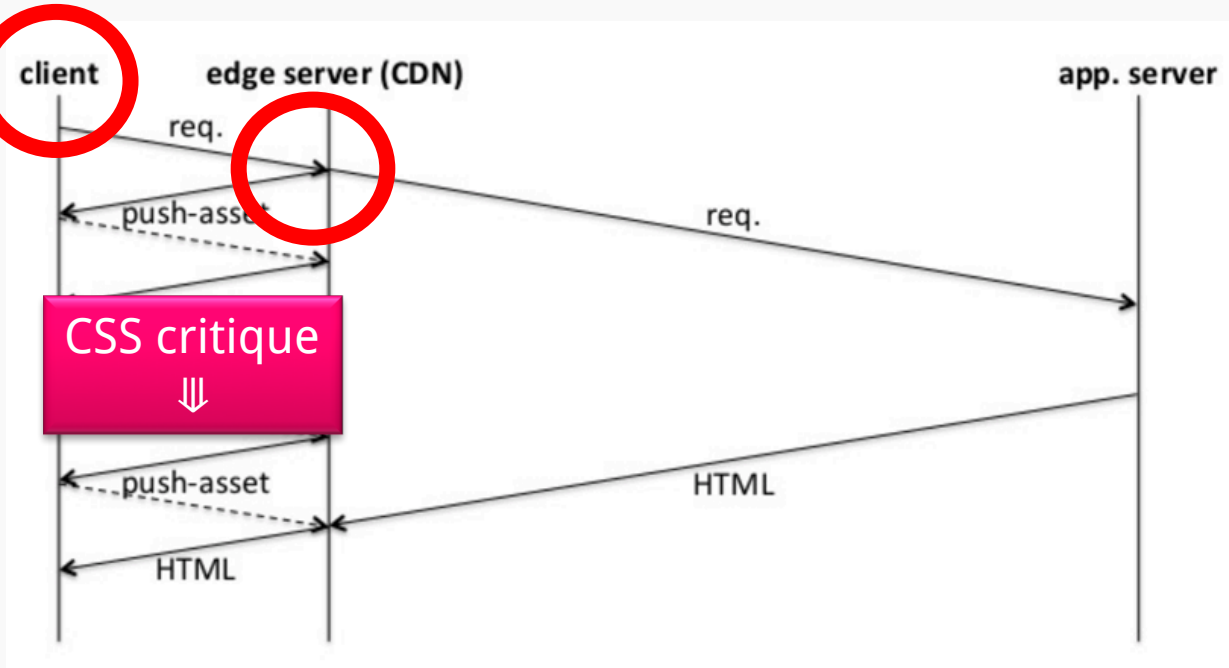


Schéma du serveur H2O

HTTP/2 PUSH : opportunité loupée

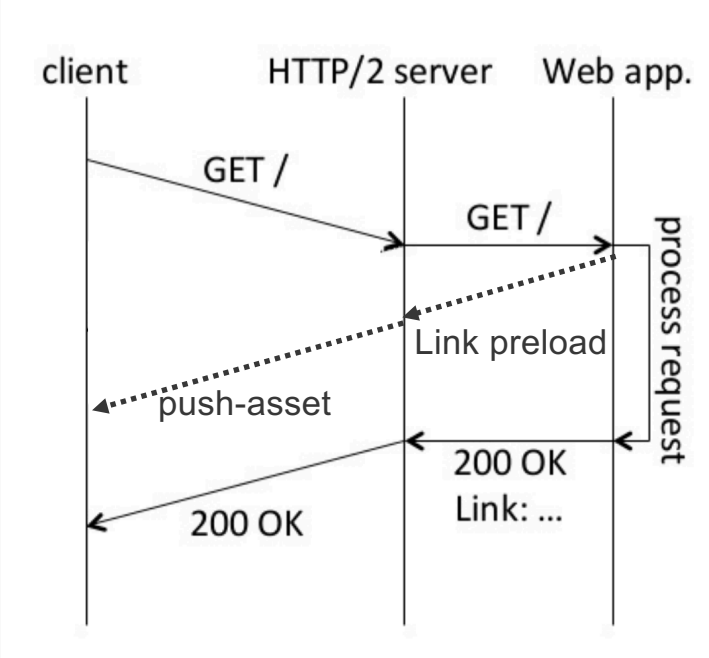
- Le serveur applicatif peut répondre avec un header

```
Link: </critical.css> rel=preload;
as=style;
```

- Le serveur H/2 intercepte ce header et fait le PUSH

(marche sur H2O, nghttp2, Apache, nginx patch)

- Problème côté applicatif : envoyer ce header AVANT de passer du temps à forger une réponse complète



HTTP/2 PUSH : danger

Trop de fichiers, trop lourd = saturation pendant le TCP Slow Start

Recommandation Chromium :

- Pas plus d'**un fichier**
- Pas plus de **100-150 Ko** sur un réseau 4G

Étude Chromium sur les bonnes pratiques de PUSH : tinyurl.com/bp-push

HTTP/2 priorities en théorie

Théorie

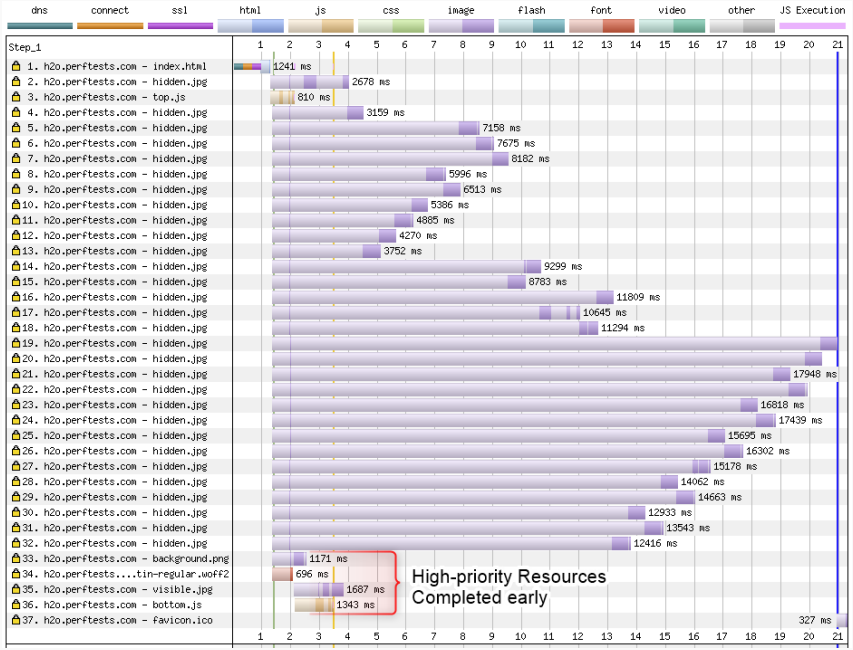
- Le navigateur indique les requêtes importantes
- Pendant le transfert, le serveur envoie plus de trames des fichiers jugés critiques

Ex: chrome, 2016

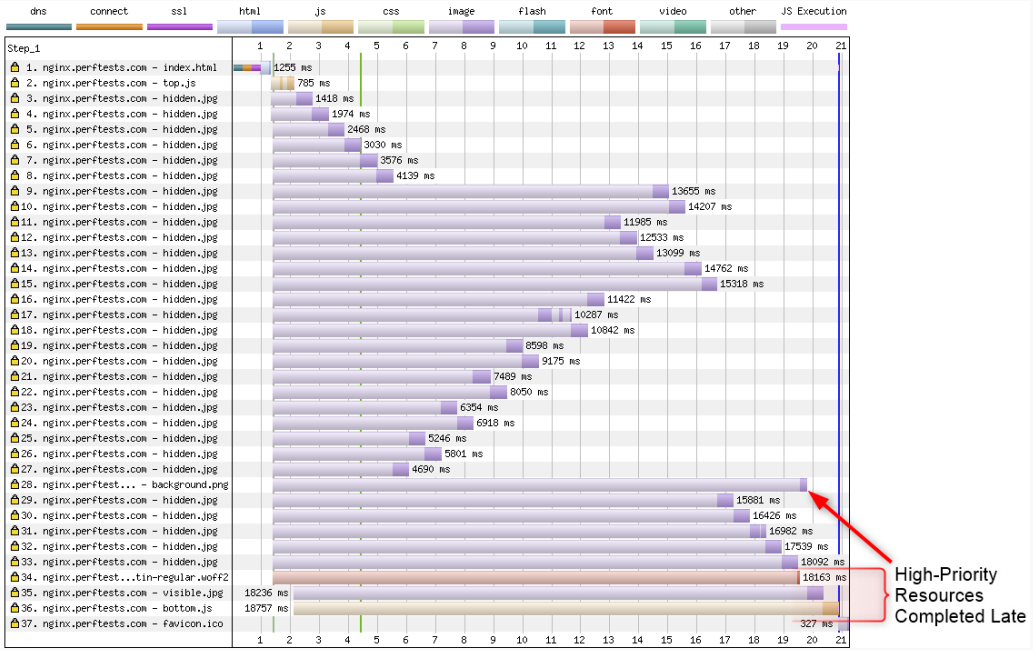
Layout-blocking	Load in layout-blocking phase	Load one-at-a-time in layout-blocking phase		
Highest	**Medium**	**Low**	**Lowest**	**Idle**
Main Resource (HTML)	Script (early** or not from preload scanner)	Script (late**)	Script (async)	
CSS (match)	@import			CSS (mismatch)
Font	Font (preload)			
	Image (in viewport)		Image	
			Media	
			SVG Document	
	Preload*			Prefetch
XHR (sync)	XHR/fetch* (async)			
		Favicon		

HTTP/2 priorities en pratique

Bon serveur



Serveur YOLO



Liste des bons et mauvais CDN : https://github.com/andydavies/http2-prioritization-issues

Page de test : https://github.com/pmeenan/http2priorities

@theystolemynick

Avec ou sans H2 : Optimiser HTTPS

SSL / TLS 1.2

Configurer :

- Keep-alive
- Session Resumption
- OCSP stapling
- False Start (ALPN/NPN + Forward Secrecy)
- Si multi-domaine : connection coalescing

TLS 1.3

- Handshake en 1 RTT
- Resumption en 0 RTT
- TCP Fast Open
  - Reconnexion en 0 RTT

Vérifier sa config serveur : https://www.ssllabs.com/ssltest/

Conclusion HTTP/2

- Juste un outil de plus : pas de miracle
- Utiliser PUSH, avec précaution
- Bien bench son serveur
- Pré-requis :
  - Optimisation de HTTPS
  - Un seul domaine (ou connection coalescing)

- Vivement HTTP/3 !

Au menu

- ✓ **Mesurer**
- ✓ Que mesurer
- ✓ Outils

Optimiser

- ✓ JavaScript
- ✓ Les fonts
- ✓ HTTP/2
- ✓ Les tiers
- • Les images

L’Enfer, c’est les autres

LES TIERS

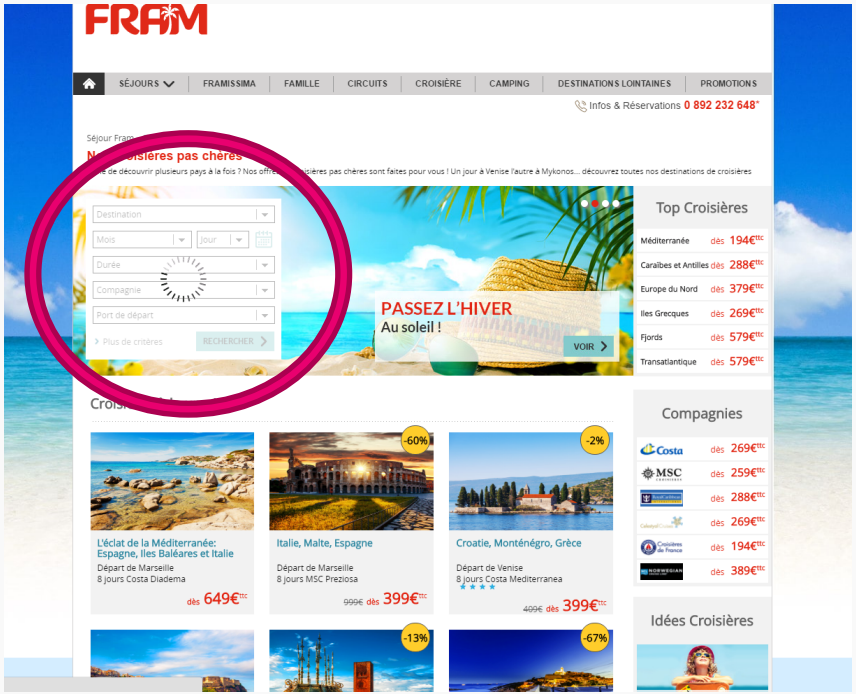
Quand le PDG t’appelle un samedi 😱

20 secondes de page blanche

Vas-tu répondre « Chef, c’est pas mes serveurs »

Simulation d’un domaine qui ne répond pas : Webpagetest.org, Advanced Settings > SPOF

DomContentLoaded



- N'importe quel script tiers va retarder le code métier
- Ici le développeur attend domContentLoaded, via jQuery
- Cette fois-ci, c'était Google Ads, pourtant en asynchrone (`defer`)

DomContentLoaded

- DomContentLoaded attend :
  - `<script src>`
  - `defer`
  - `document.write()`

- Le surveiller
  - Chez les utilisateurs :
    - `performance.timing.domContentLoadedEventEnd`
    - Google Analytics
  - De manière synthétique avec n’importe quel outil

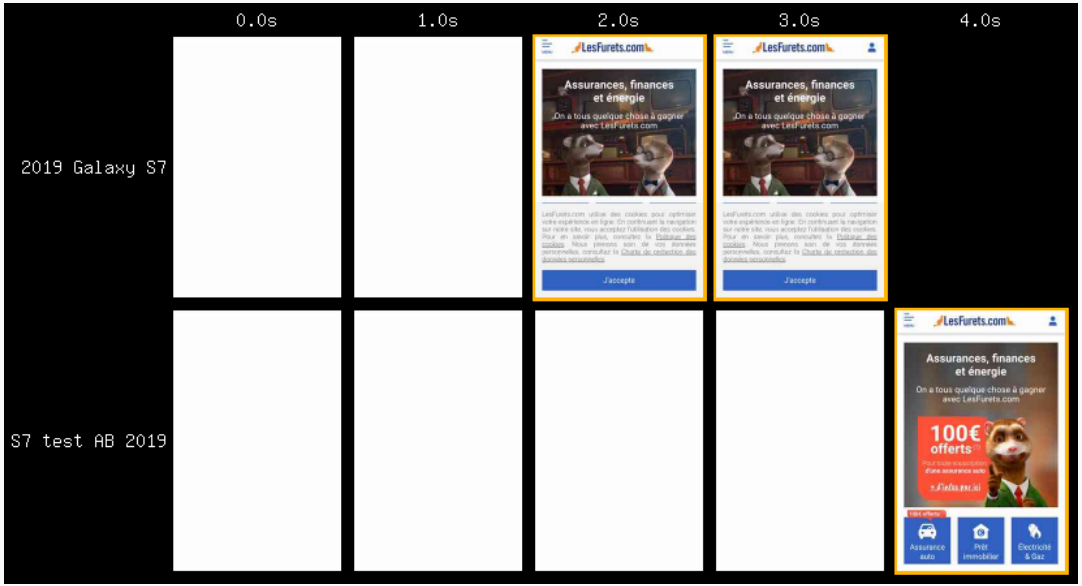
Combien coûte un test A/B ?

Tests A/B côté client :

- Minimum **1s de page blanche**
- ⬈ taux de rebond

Cas lesfurets.com : 1seule redirection côté client

- **-50% de trafic SEO**



Savoir inclure les tiers

Rapatrier les dépendances :

- Fonts 

- JS/CSS

Pubs, tests A/B, analytics, trackers …

- En `iframe`
- Préférer `async` à `defer`
- Exécuter après le code métier (`onload` ?)
- Utiliser un reverse-proxy.

Au menu

- ✓ **Mesurer**
- ✓ Que mesurer
- ✓ Outils

Optimiser

- ✓ JavaScript
- ✓ Les fonts
- ✓ HTTP/2
- ✓ Les tiers
- Les images

Compression et format

Responsive et lazy-loading

IMAGES ET VIDÉOS

Connaître ses formats

Aplats de couleurs, bords nets, texte, transparence, petites dimensions …

- Surement **PNG**, probablement **SVG**
- ~~GIF~~

Pas de texte, beaucoup de détails

- **JPEG** et consors

Les 2 en un seul fichier ? Dirty, dirty hack

Connaître ses formats

- Gif animé ➔ video

```
<video autoplay loop muted
playsinline>
 <source src="load.mp4">
 <img src="load.gif">
</video>
```

VOGUE

Codec	Poids
GIF	1.7 Mb
H.264	800 Kb
H.265	-15%
VP9	-20%
AV1	-15%

JPEG

- Meilleur compresseur actuel : Mozjpeg

La compression niveau 80 a toujours été une légende …

Chercher le meilleur ratio compression / qualité

- Avec tes yeux : “Save for the Web”™
- Automagique : cjpeg-dssim , Guetzli

- À comparer avec WebP : surprise !

Encore mieux que la compression

- Réduire les dimensions
  - Stratégie des « responsive images »
- Charger moins d’images c’est mieux (Captain Obvious)
  - Technique de « lazy-loading »

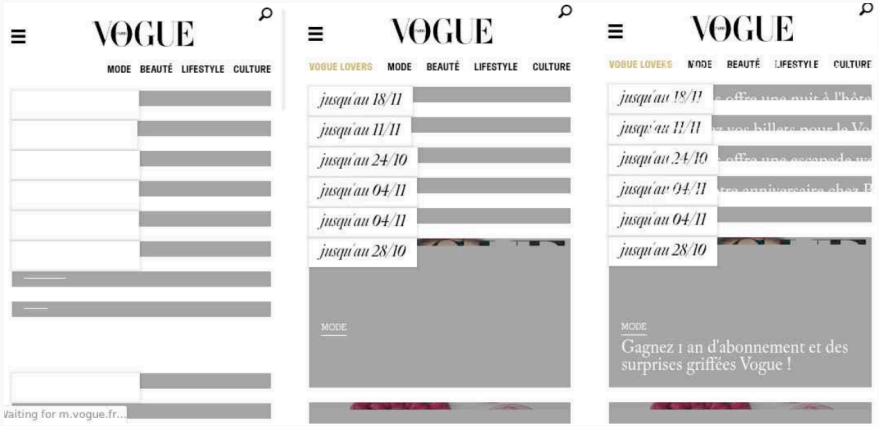
Chargement à la demande

Ne charger les images que lorsqu'elles vont être visibles.

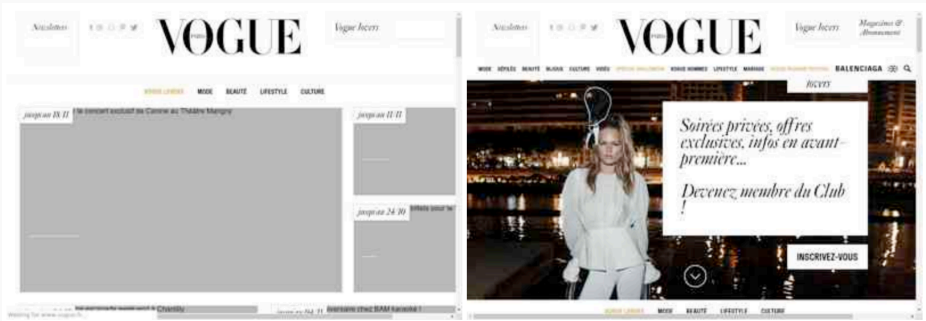
1. Ton propre code avec l'API **`IntersectionObserver`**
2. Libs JS : Lozad, Yall, LazySizes
3. `<img loading="lazy">`

Prévoir les phases de chargement

ÉVITER le contenu qui saute

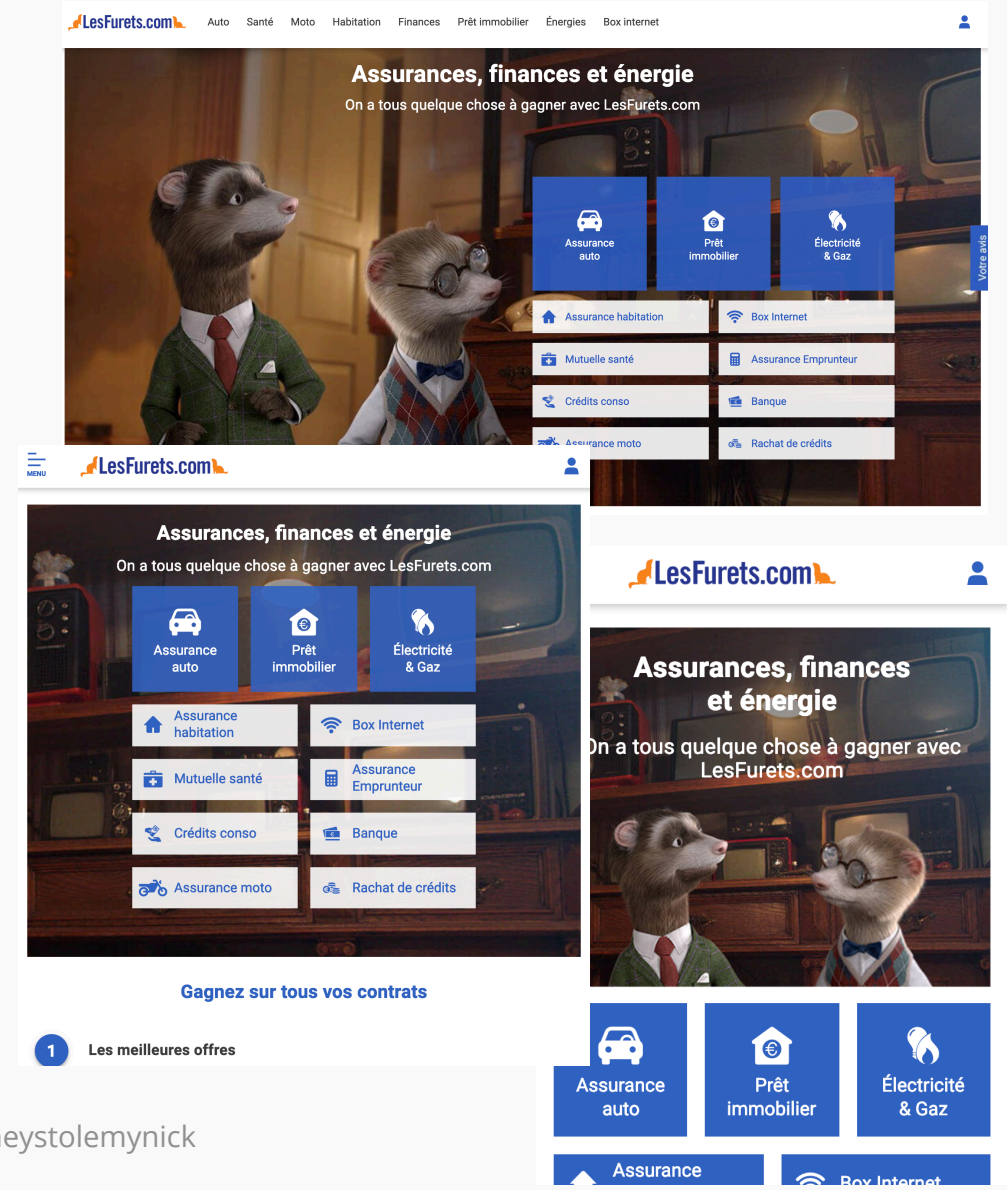


Bon (ou presque)



Responsive image

- Art Direction
  - Bureau: image de fond HD + mascottes HD
  - Tablette : seulement l'image de fond
  - Mobile : Image de fond + mascottes légères
- Résolution adaptée
  - L'image de fond n'a pas la même dimension, parfois pas le même ratio



Tag `<img>` === télécharger vite

Risques

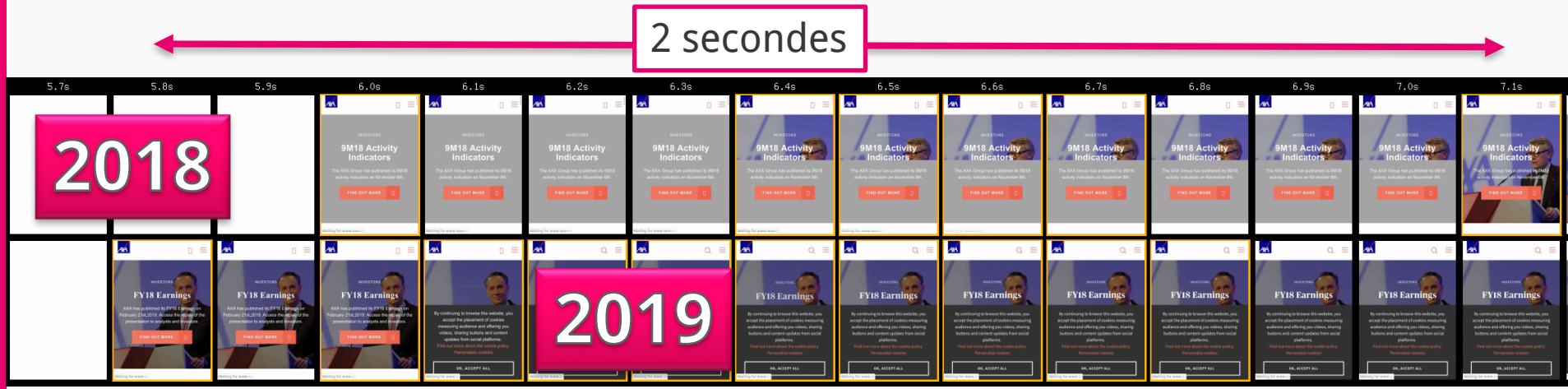
- `display: none;` ➔ chargées
- Images cachées dans un sous-menu ➔ chargées

Opportunités

- Les images de fond sont chargées en **priorité basse**
- Pour les **charger plus vite**, utiliser `<img src>`

Donner la priorité à ce qui est visible

- On affiche plus vite nos grandes images.



NB: sur cette page, l'image 2019 est pourtant 2 fois plus lourde que la 2018 !

Conclusion images

1. Techniques d’évitement
   - Chargement à la demande
   - Stratégie responsive

2. Optimisations fines
   - Connaître ses formats
   - Compresser

Recommandations 2005 ➔ 2019

- Compresser en gzip🗜
  - Et maintenant en **brotli** (+20%)
- Utiliser le cache client !
  - Et ajouter la gestion de **l'offline** ✈ (mot-valise : PWA)
- Utilise le serveur, Luke
  - Et renomme ça « **Server Side Rendering** » sur ton CV
- Tout concaténer ?
  - Mais pas trop parce que « **ça dépend** »©

Conclusion

1. Mesurez ! 
2. Connaissez vos utilisateurs  
3. Optimisez 
4. GOTO 1

Questions ? 🤔

- Conférence Webperf :
  19 sept. 2019

BrainCracking

- Audit, formation ?

➔Mail jp@braincracking.fr