
# INP vs JS

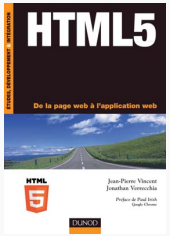
**Next/React, Nuxt/Vue, Angular, SvelteKit**

Jean-pierre Vincent – Expert Webperf indépendant

# Hello !

Jean-Pierre Vincent

BrainCracking


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

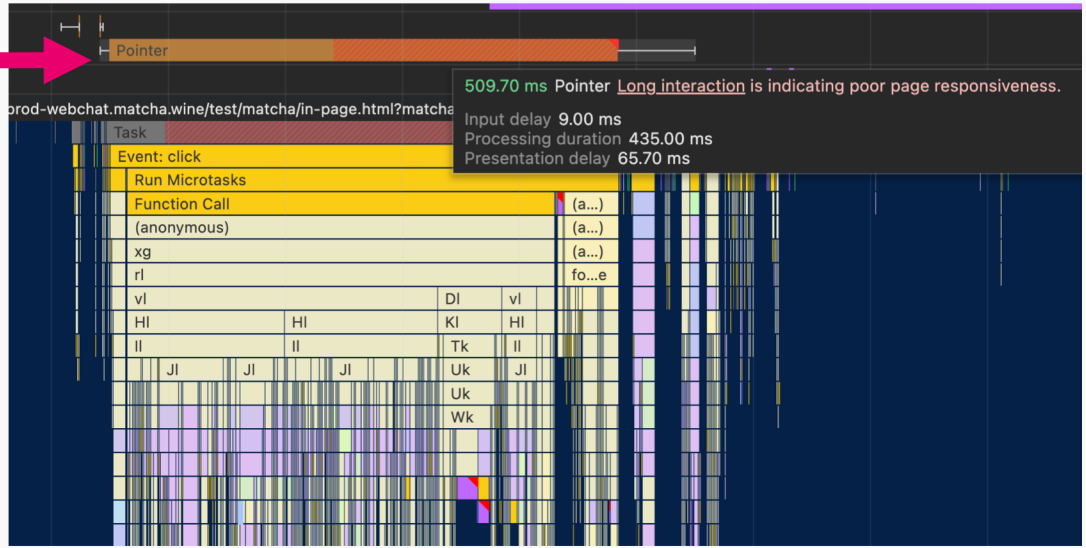
- Co-organisateur de we ♥ speed

  We Love Speed

@theystolemynick

# L'INP ?

Définition




Use case classique : tu cliques, ça bouge

Blockers classiques : ton framework, ton router, tes 3rd parties.

Sans surprise : trop de JS = mauvais INP

Surprise 1 :

- pas forcément des interactions prévues
- La charge JS joue parfois sur d'autres métriques

# Contenu VS JS

Apparemment tout va bien :

- Angular optimise correctement par défaut
- L'équipe était à l'aise avec les optimisations de chargement classiques
- Les tests en prod confirment que ça s'affiche vite

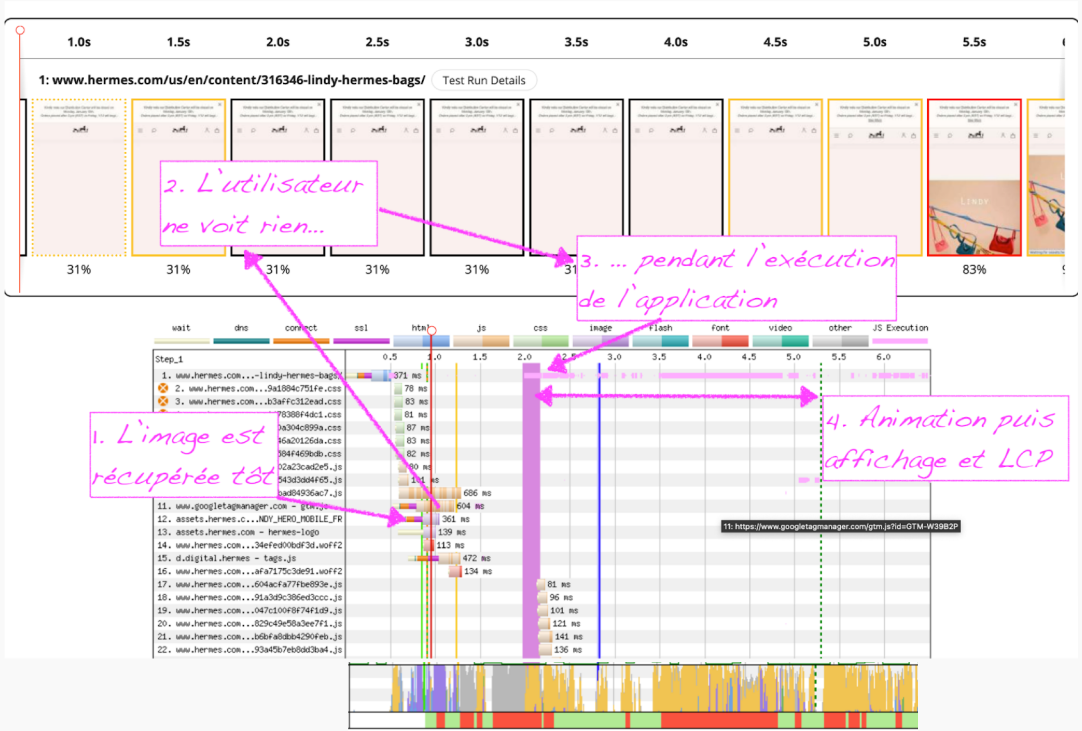
LightHouse était content,

- ✅Le SSR marche : l'image est bien dans le HTML
- ✅ JS est asynchrone
- Checklist d'optimisation d'images est faite
  - ✅ fetchpriority=high
  - ✅ Lazy-loading du reste
  - ✅ Responsive
  - ✅ Compression
- ❌ Mais pas les stats Google Crux...

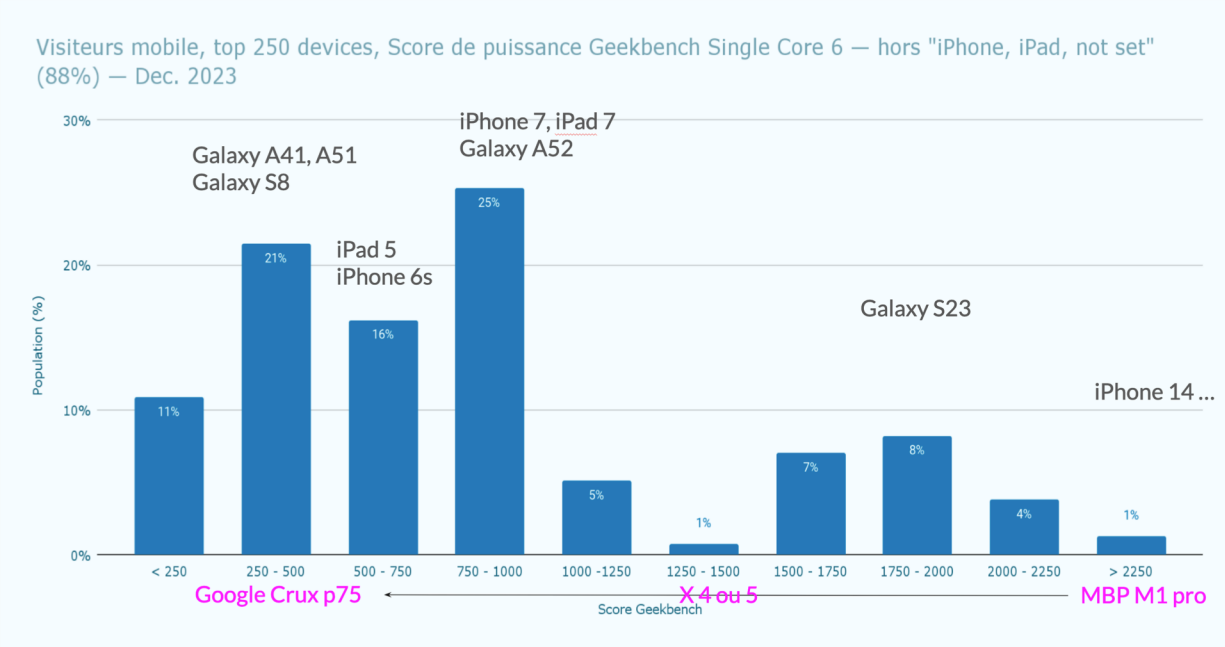
# Contenu VS JS

- Tester sur un mobile à 200€
- Ici il faut 750ms pour interpréter 500Ko de GTM et 1,5 Mo d'applicatif
- Puis Angular démarre vraiment et met 3 secondes pour démarrer l'animation de l'image.

- ➔ action rapide : sortir l'animation CSS d'Angular ⚡
- ➔ action longue : optimiser l'intégralité de l'applicatif 😅



# La puissance disponible chez nos utilisateurs



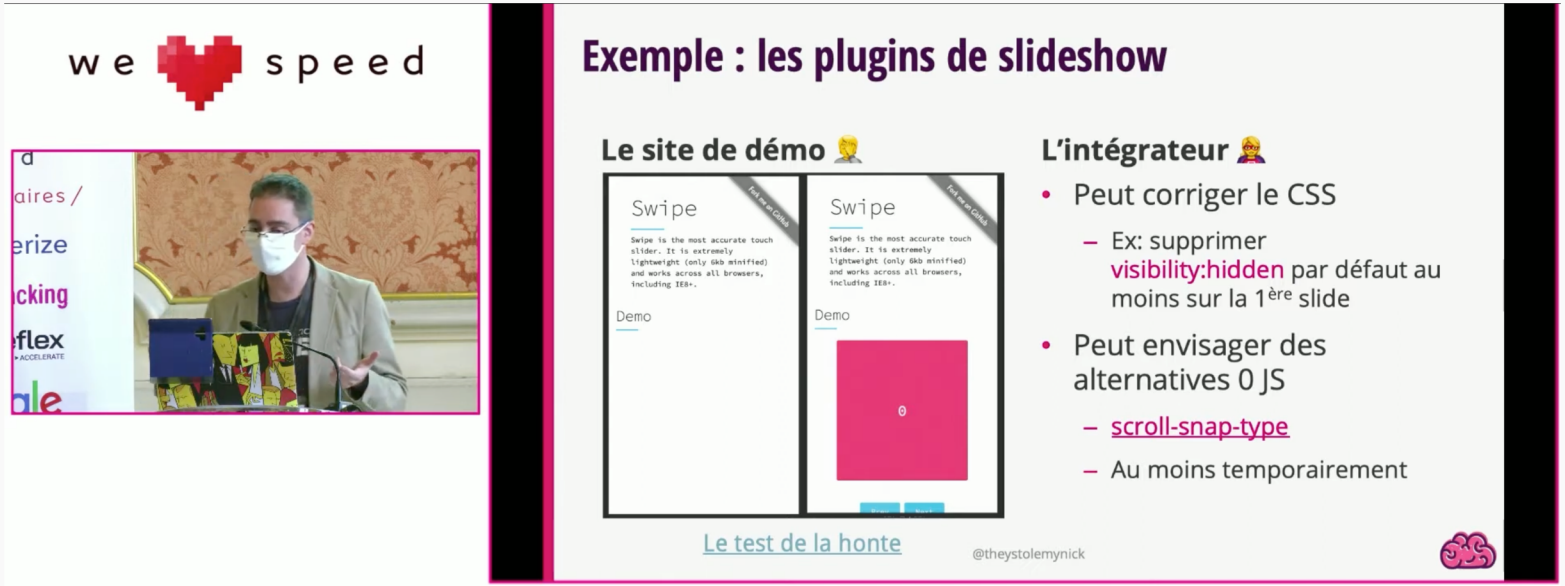

Différence centile 75 ➔ machine du dev : **4 à 10 fois** plus lent
➔ tester sur un « vrai » mobile

# Privilégier le natif dès que possible

- Images, Vidéos responsive ➔
  - `<source src="large.mp4" type="video/mp4" media="(min-width: 600px)">`
- Lazy-loading d'images, d'iframe
  - `<img|iframe loading=lazy …>`
- Preload, prerender de pages

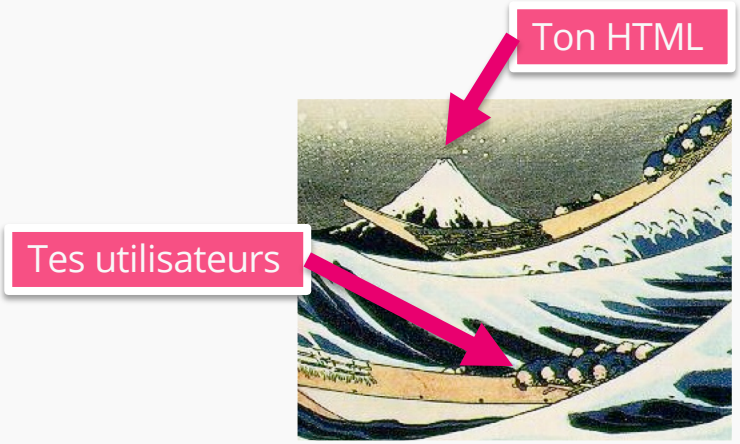
- ➔ pourquoi pas un spécialiste HTML/CSS dans ta team de dev React ? (vidéo l'intégrateur ce héro)



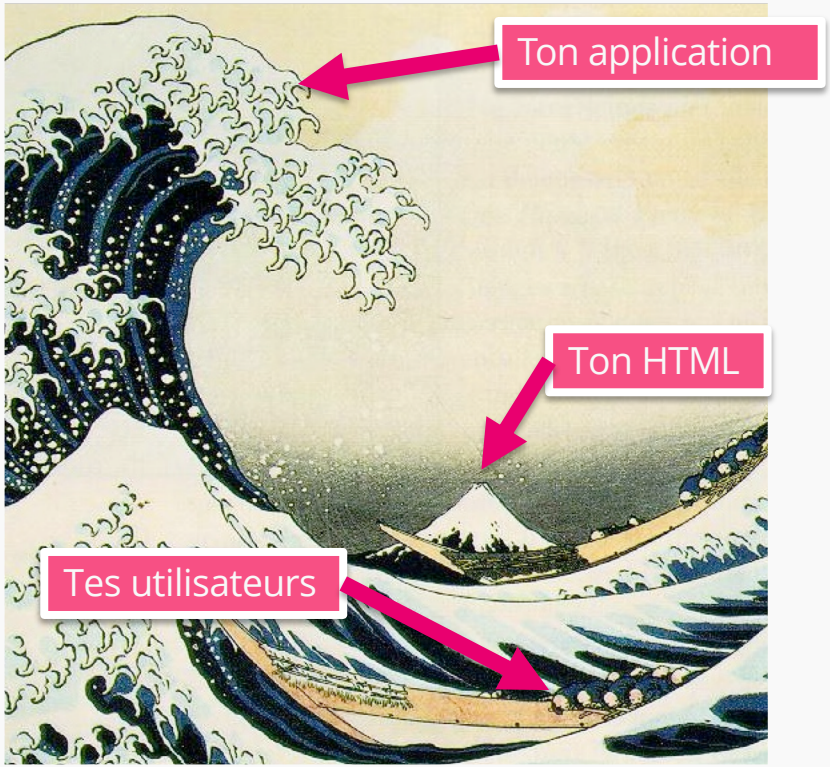



2021 – youtube : l'intégrateur ce héro

Ton HTML


Ton HTML
Tes utilisateurs

# De la réhydration au tsunami 🌊

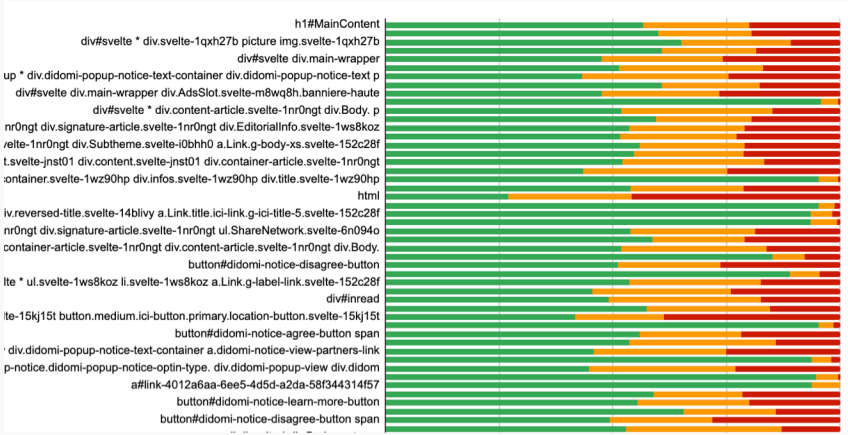



- Réhydration :

- Ton appli moderne :

# Lien rehydration – INP ?

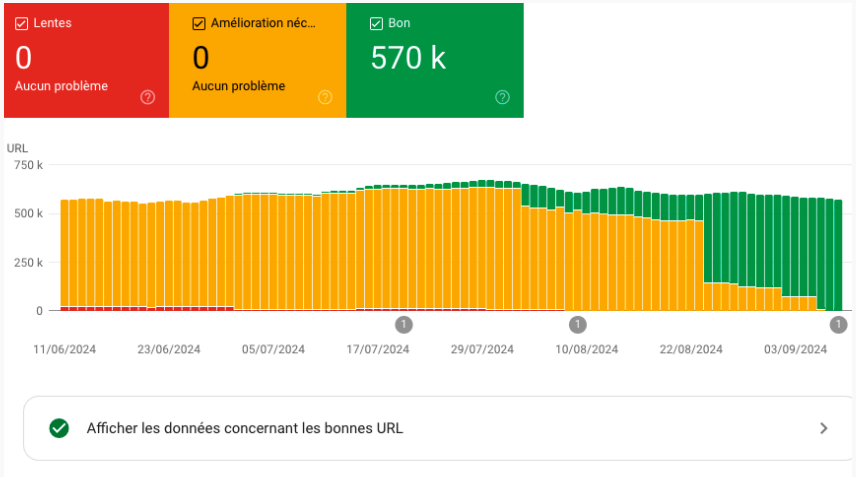
- INP = Interaction to Next Paint… mais pas que !
- Mise en place d’un monitoring maison
  - API Long Animation Frame, via la librairie Google web-vitals ou un RUM
- **Les interactions les plus lentes ne sont pas forcément des interactions possibles**
  - Liens, CMP et pub mais aussi … bouts de texte, titres et images sans interactivité.




🤨 Origines mauvais INP sur une page article

# L’INP est bousculé par la charge de JS

- 🤓 Plus la page est visible tôt, plus il est probable qu’ils cliquent pendant la grosse charge initiale
- 🦯 Les utilisateurs cliquent partout
- 🤯 **L’INP mesure l’incapacité du navigateur à réagir, pas forcément une interaction prévue**
- ♻️ Rappel : le JS de la page est ré-interprété à chaque navigation normale (ni BFCache, ni soft nav)
- ➔ 🗜️ on n’a pas fini d’optimiser le poids des bundles et la rehydration




Poids du bundle décompressé : ↘️ de 1,7 Mo à 700 Ko
INP ↘️ de 270 ms à 145 ms

Lazy-hydration

No-hydration

# NE PAS EXÉCUTER TOUT, TOUT DE SUITE



# Le tree-shaking 🍂, c'était pas automagique ? 🧙

- Système de routing des SSR : oui ils marchent !

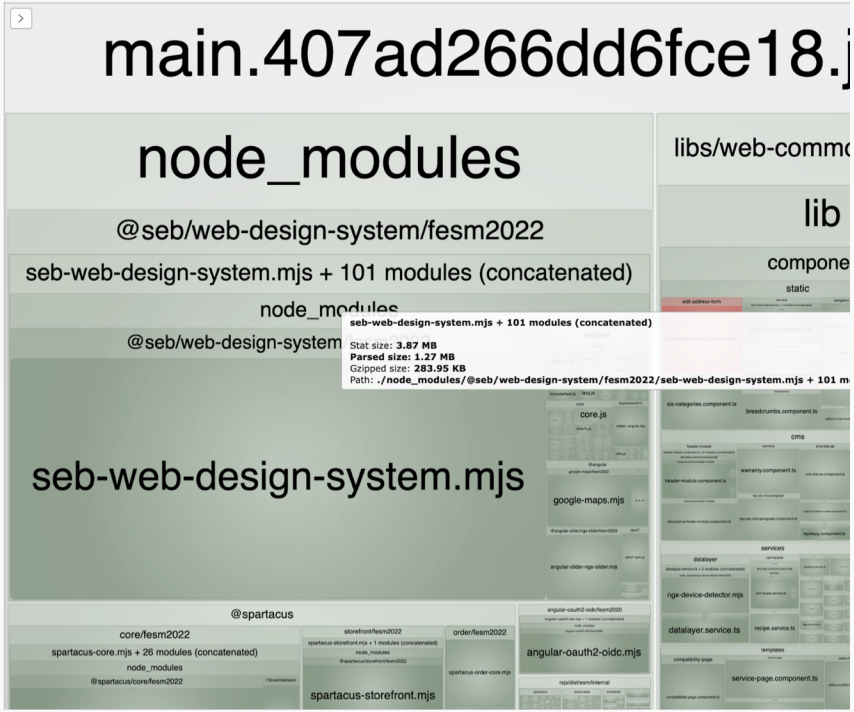
  Sauf quand tu leur manques de respect

- Toi-même tu connais la route qui inclue tout car tu sais pas ce que le CMS peut t'injecter
  - Template typique : page fiche produit, page article, zone de mise en avant géré par le marketing …
  - ➔ chunk manuel des composants peu fréquents ou lourds



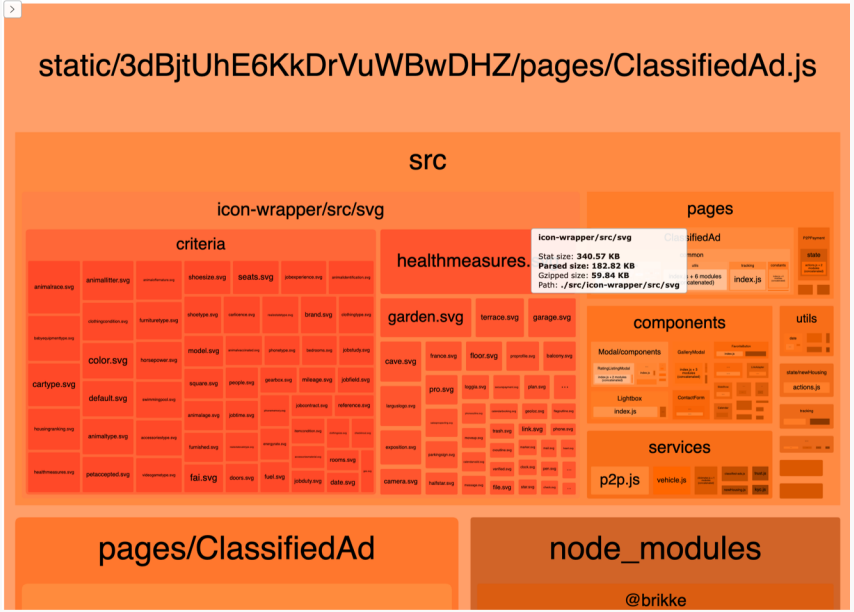
# Les dépendances non tree-shakable

- Éviter les « barrel files » (import/export)
  - ➔ `import { arrayMatch } from 'utils'`
- Faire passer babel dans les node_modules
- Les dépendances externes
  - Vérifier la compat ESM, forcer un peu avec des plugins ou des alternatives
- La librairie de composants interne
  - 😵 Inclure un logo = récupérer tout le design system



# Les SVG

- Les SVG comme composant
  …
  - À traiter comme des assets externes 95% du temps
  - Besoin de variables ? ➔ CSS !



# Lazy-exécution de composants

## React < 18

- Un import dynamique du composant
  - import(/*webpackChunkName:"Footer"*/ '../footer')
- @loadable/component ou Next/dynamic
- Une condition d'affichage
  - Une condition métier (isModalOpened…)
  - Un HOC comme hydration-on-demand ou lazy-hydrate

```
<LazyHydrate whenVisible>
  <Footer …
<LazyHydrate
```

## Angular @defer

```
@defer (on viewport) {
  <wc-footer></wc-footer>
}
```

- 🤬 Pas compatible SSR
- uniquement composants standalone (v15)

## Vue / Nuxt

- Vue : import() + condition (vue-lazy-hydration)
- Nuxt 2+ : préfixe Lazy + v-if



# Absence de rehydration

### React < 18

- Les HOC comme hydration-on-demand ou lazy-hydrate
  - `<LazyHydrate ssrOnly>`
    `<Articletext …`
    `</LazyHydrate`

### Next 13 (React 18)

- Server components
  - `"use client"`

### Angular

- Il existe bien
  - <articletext **ngSkipHydration** />
- Mais la doc le déconseille car ça force les re-render 🙄
- ➔ à tester

### Nuxt 3

- <NuxtIsland>



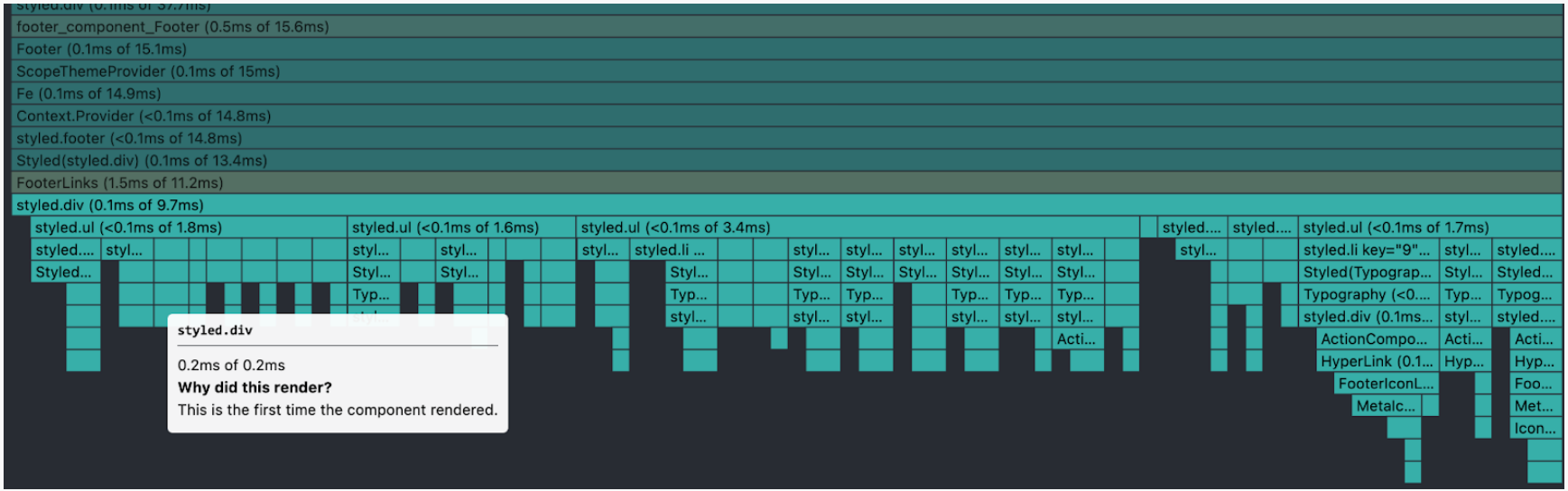
# CSS IN JS

@theystolemynick

# CSS in JS, côté client 💥


Application **lit-element**, habituellement très légère …
mais temps multiplié par 5 à cause de postCSS utilisé côté client !

# CSS in JS, côté client 💥

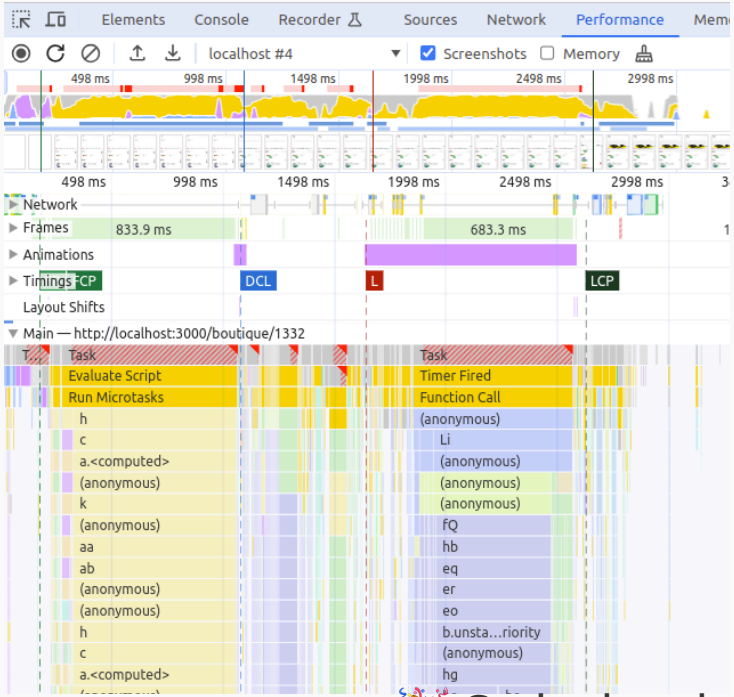



React + Styled + Theme ➔ remplacer theme par des variables CSS
Génération de CSS au build : vanilla-extract, stitches, Linaria, compiled

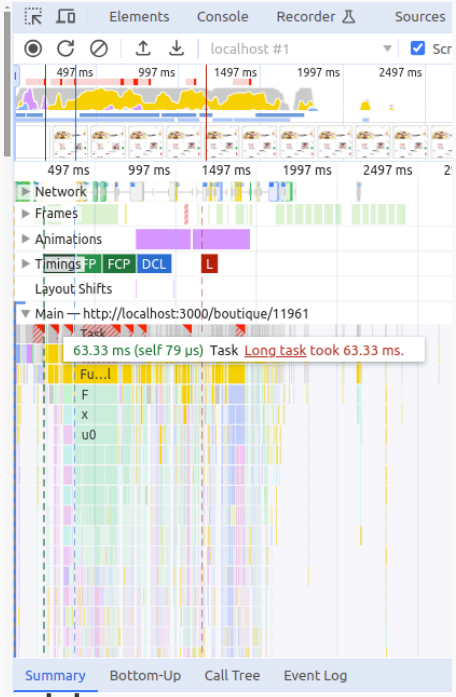
# UPGRADE DES FRAMEWORKS ?

# Upgrades de React ?

## React < 18



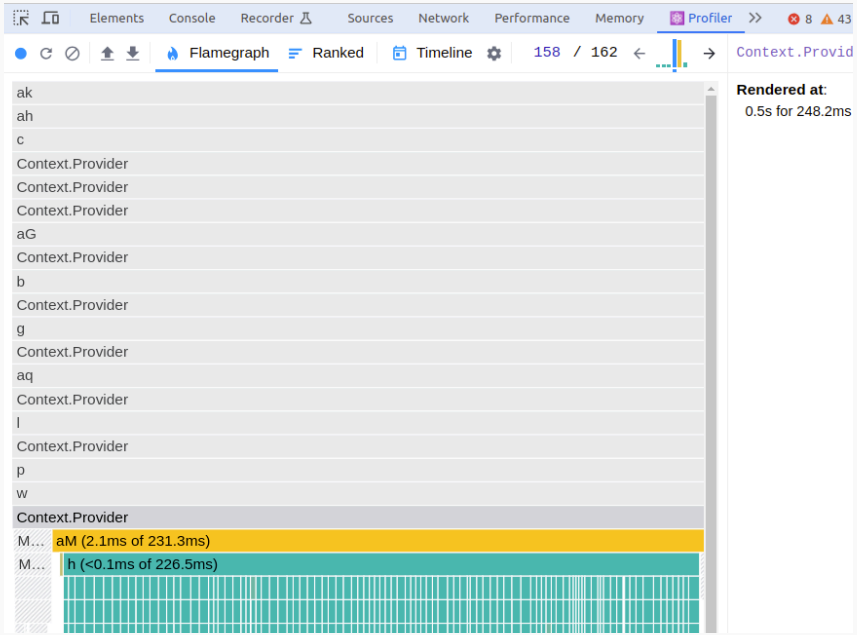

## React 18



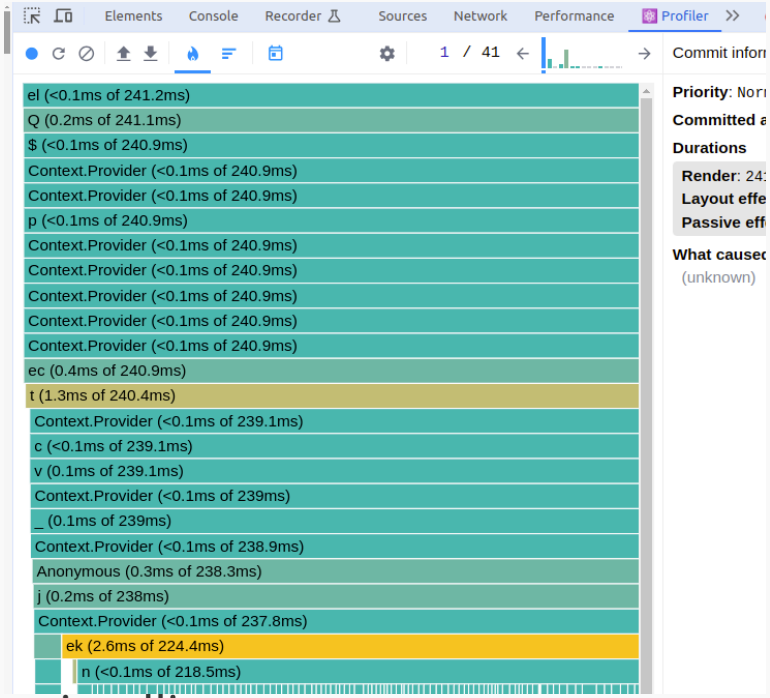

🎉 Cycle de rehydration moins bloquant

# Upgrades de React ?

## React < 18




## React 18




🎉 Memoisation plus intelligente

# Toujours upgrader ?

- Svelte : peu mature, donc probablement oui
- Angular : probablement oui
- Vue / Nuxt : probablement oui

- Pour React / Next :
  - Par défaut, meilleur
  - Mais les hacks permettant d'améliorer manuellement la rehydration ne marchent plus
  - ➔ amélioration perf imprévisible, donc utilisez la version qui vous arrange par ailleurs

# CONCLUSION



# Comment maintenir la réactivité ? ⚡

- **Monitoring**
  - **I**nteraction to **N**ext **P**aint en RUM
  - **T**otal **B**locking **T**ime, en CI
  - Métriques métiers custom
    - Render des composants importants (useEffect)
    - Temps avant l'appel pub
    - …
  - Poids du bundle en CI
- Outillage de profiler en dev
  - Pouvoir activer les outils sur la version de prod
  - Tester sur un mobile réaliste



# Questions ? 🙋

BrainCracking

- Aller plus loin
  - L'atelier de 12H50 en salle 1
  - Le stand pendant les pauses

- Slides et vidéo :
  - Twitter : @_welovespeed ou @theystolemynick
  - Mastodon : @WeLoveSpeed@piaille.fr ou @jpvincent@mamot.fr
  - Youtube.com/@WeLoveSpeed
  - LinkedIn
- Formation, audit, accompagnement ? jp@braincracking.fr