

# M1IF03

## Conception d'applications Web



### **PERFORMANCE DES APPLICATIONS WEB**

**LIONEL MÉDINI  
NOVEMBRE 2022**

# Plan du cours



- Introduction
- Concepts fondamentaux
- Méthodes d'optimisation
- Outils
- Conclusion
- Références

# Introduction



- Position du problème
  - Pas de phase d'installation pour une webapp
    - ➔ Il faut que ça marche tout de suite !
  - Temps de chargement variables en fonction des applications
    - ✦ <https://www.youtube.com/watch?v=gcSEA9FsxgM>
  - Des internautes exigeants

**47%** of consumers expect a web page to load in **2 seconds** or less.

**40%** of people abandon a website that takes more than **3 seconds** to load.

A **1 second** delay in page response can result in a **7%** reduction in conversions.

Source : [Akamai & Gomez.com](#), ([résumé en PDF](#))

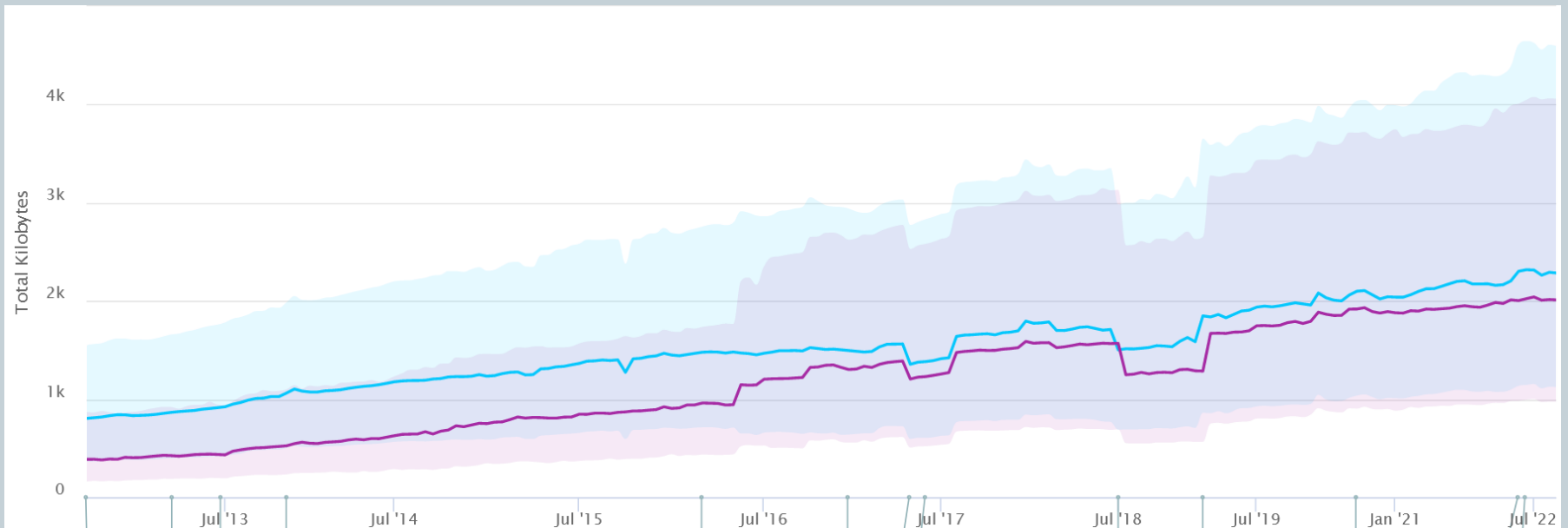
# Introduction



- **Constats**

- **Évolution des applications Web**

- ✦ **Augmentation de la taille des pages**



- ✦ **Source : [HTTP Archive](#)**

— Desktop — Mobile

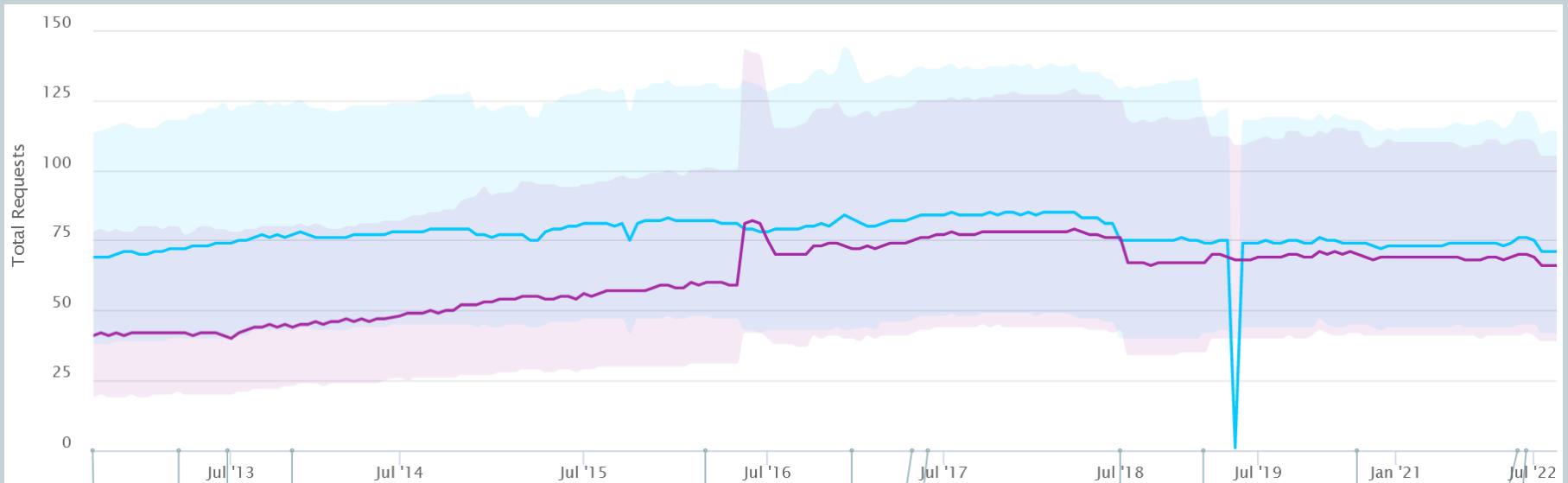
# Introduction



- **Constats**

- **Évolution des applications Web**

- ✦ **Augmentation de la complexité des pages**



- ✦ **Source : [HTTP Archive](#)**

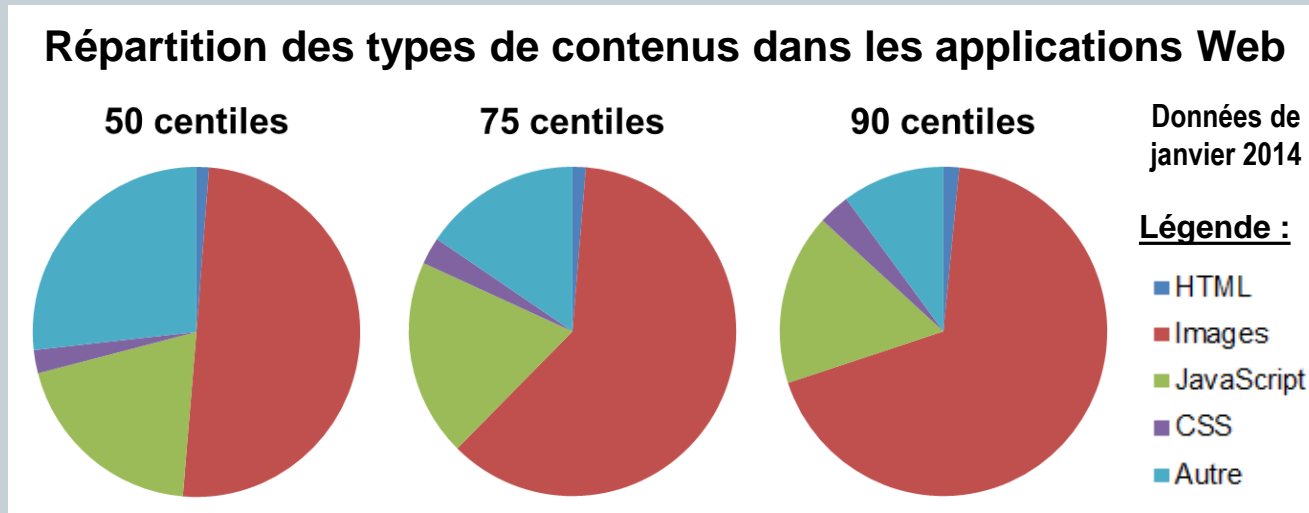
— Desktop — Mobile

# Introduction



- **Constats**

- Augmentation de la complexité des applications Web



- ✦ Données source : [Google Web Fundamentals](#)
- ➔ Ce qui est (en général) généré dynamiquement
  - ➔ HTML
  - ➔ Autres (données XML, JSON)

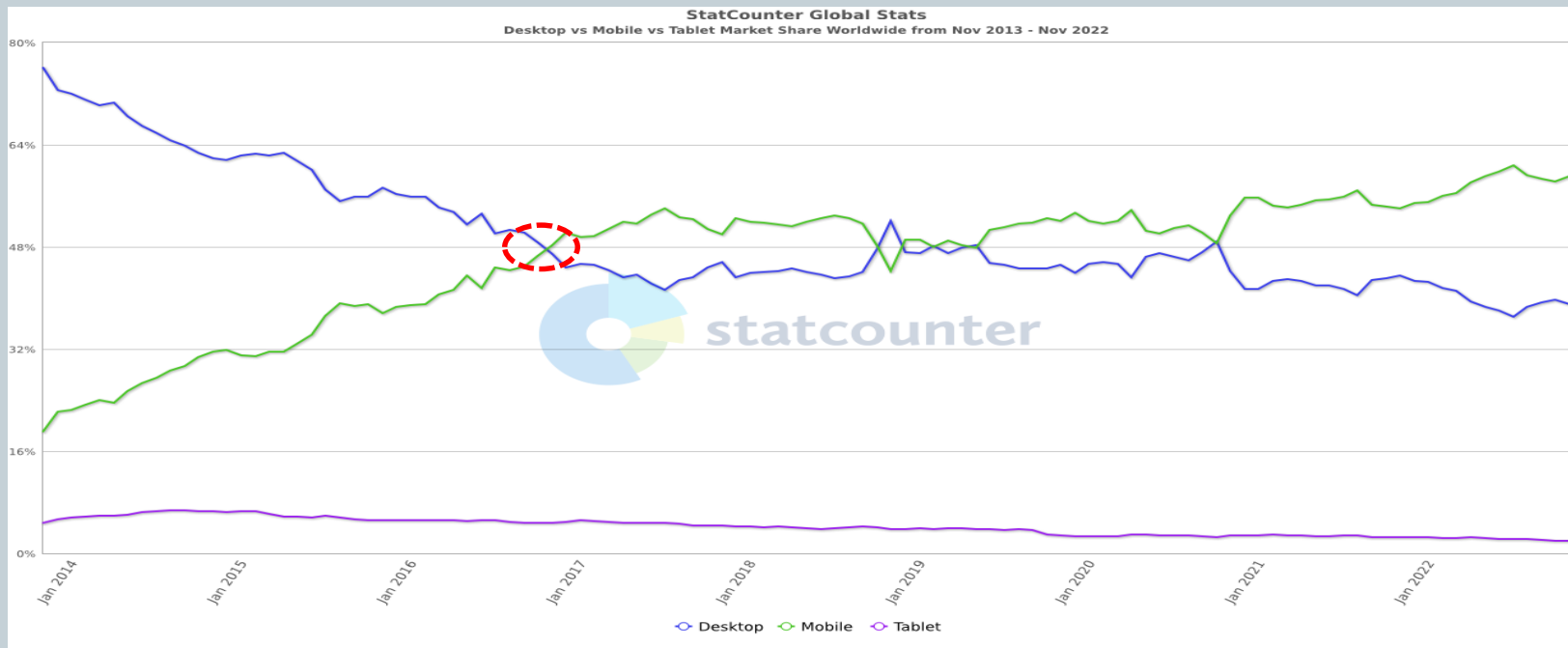
# Introduction



- **Constats**

- Utilisation du Web en mobilité

- ✦ Augmentation de la variabilité des situations d'utilisation



- ✦ Source : [StatCounter](https://www.statcounter.com)

# Introduction



- Les forces en présence

- Serveur

- ✦ La performance de la génération côté serveur est maîtrisée

**80-90% of the time spent by users waiting for pages to load is spent on the frontend,** all the work that needs to be done after the HTML document has arrived. Backend optimization is still critical - it drives down hardware costs and reduces power consumption. But if your goal is to make your pages faster for your users, the place to start is the frontend.

Source : [stevesouders.com](http://stevesouders.com)

- ✦ Problématique actuelle : scalabilité

→ Hors du scope de ce cours

- Réseau

- Client



# Introduction



- Les forces en présence
  - Serveur
  - Réseau
    - ✦ Non maîtrisé
      - Bande passante
    - ✦ Maîtrisé
      - Protocoles
      - Données
        - Quantité
        - Format
        - Types
  - Client

# Introduction



- Les forces en présence

- Serveur

- Réseau

- Client

- ✦ Non maîtrisé

- Ressources matérielles (mémoire, capacités de calcul)

- Ressources logicielles (navigateur, plugins...)

- ✦ Maîtrisé

- Structure, contenus de l'application

- Ordre de chargement des ressources

- Ordre de traitement des données

- ...

➔ **Rappel : c'est côté client que se mesure l'expérience utilisateur**

# Introduction



- Initiatives / ressources
  - [W3C Web Performance WG](#)
    - ✦ [Dates](#) : août 2010 → ... → juin 2020 → ...
    - ✦ Objectifs : “provide methods to measure aspects of application performance of user agent features and APIs”
    - ✦ Résumé : [Web Performance Timing APIs Primer](#) (Editor’s draft)
    - ✦ Nombreuses spécifications
      - Mesures temporelles
      - Événements liés au chargement et au traitement des ressources
      - Aide aux développeurs
  - [Google Fundamentals](#)
  - [Steve Souders](#)

# Spécifications

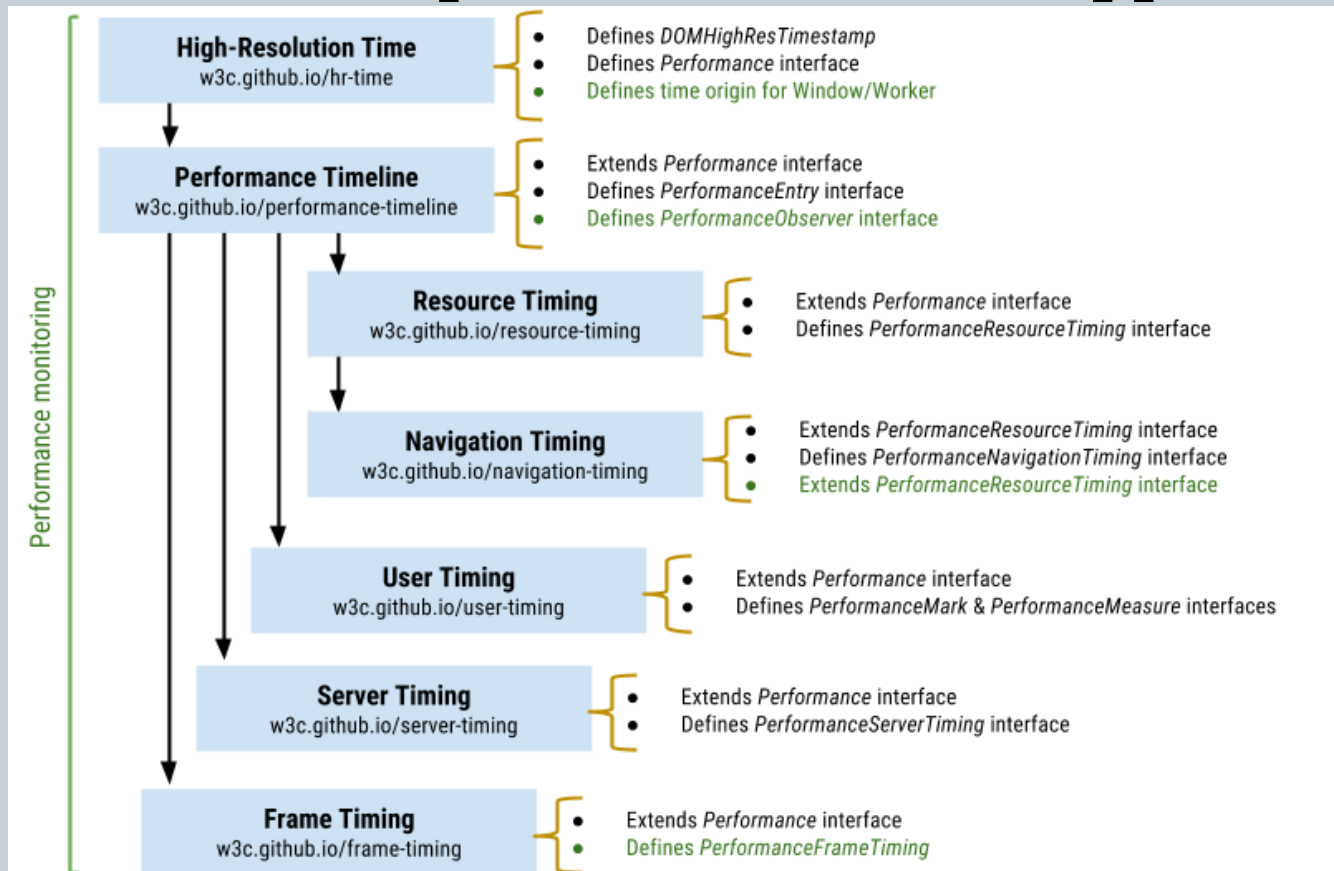


- Quelques [specs](#)
  - Navigation Timing [Level 1](#), [Level 2](#) (WD, sept. 2022)
    - ✦ “Racine” des spécifications
    - ✦ Traitement global de la page
  - Performance Timeline [Level 1](#), [Level 2](#) (WD, nov. 2022)
    - ✦ Sélection des sous-ressources à observer
  - Resource Timing [Level 1](#), [Level 2](#) (WD, nov. 2022)
    - ✦ Processus de chargement de chaque ressource
  - High-Resolution Time [Level 1](#), [Level 2](#), [Level 3](#) (WD, nov. 2022)
    - ✦ Mesures temporelles
  - User Timing [Level 1](#), [Level 2](#) (REC, fev. 2019)
    - ✦ Aide à la mesure de la performance pour les développeurs

# Spécifications



- APIs d'accès à la performance d'une application



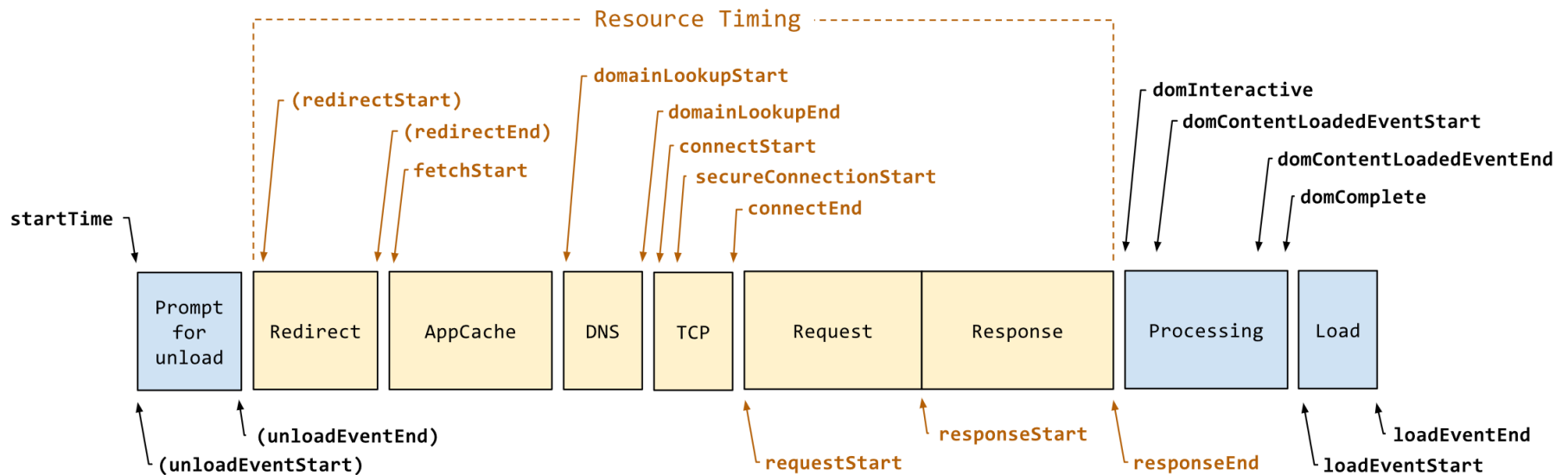
✦ Source : [Web Performance Primer](#)

# Spécifications



- **Navigation Timing**

- De l'initialisation de la navigation au traitement de la réponse



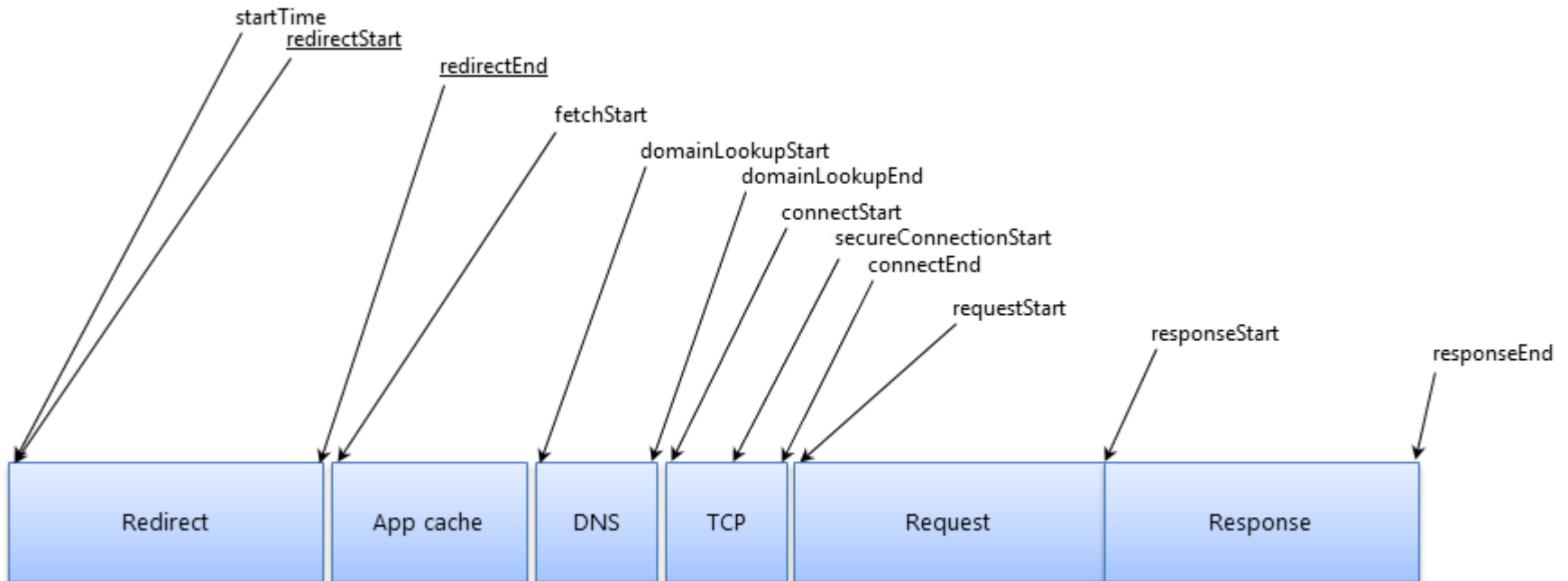
✦ Source : [Navigation Timing Level 2](#)

# Spécifications



- Resource Timing

- De la réception de l'URL à l'arrivée de la réponse



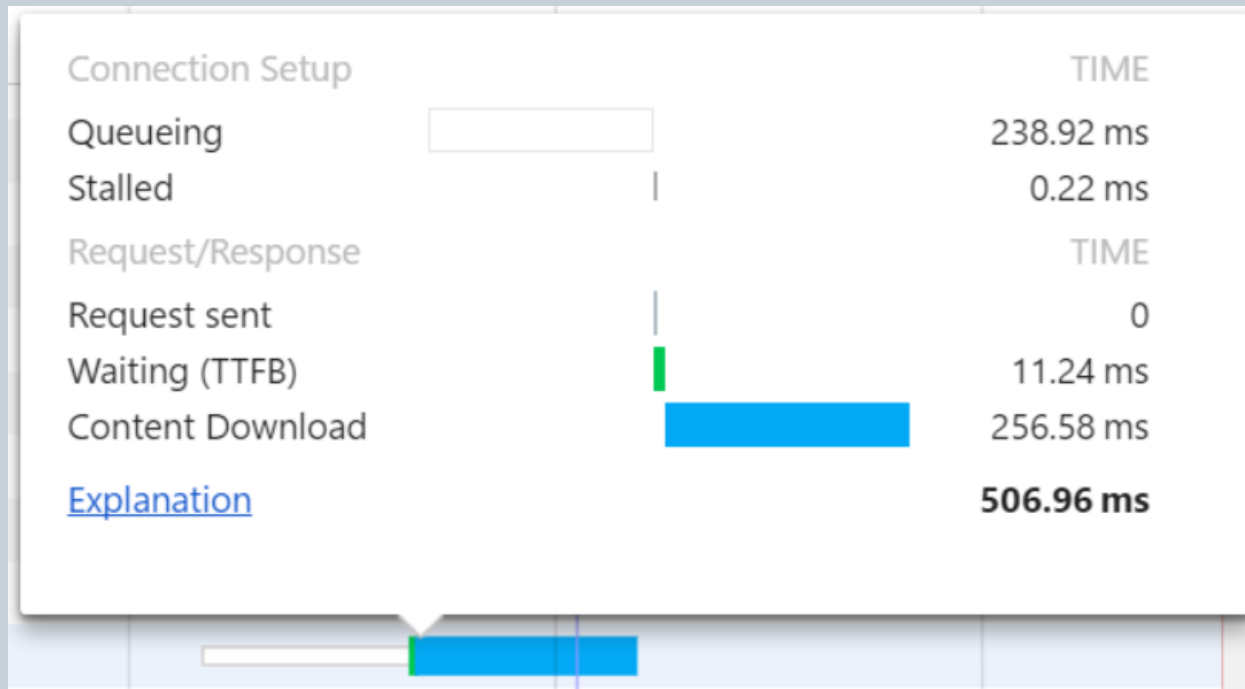
✦ Source : [Web Performance Timing API Primer](#)

# Spécifications



- **Resource Timing**

- Une fois la connexion TCP obtenue, plusieurs états pour la transaction



✦ Source : Google Chrome Dev Tools



# Spécifications



- Performance **interface**
- Récupérer les valeurs de “timing” pour les ressources

## WebIDL

```
partial interface Performance {  
    PerformanceEntryList getEntries ();  
    PerformanceEntryList getEntriesByType (DOMString type);  
    PerformanceEntryList getEntriesByName (DOMString name, optional DOMString type);  
};  
typedef sequence<PerformanceEntry> PerformanceEntryList;
```

- ✦ Source : [Performance Timeline](#)

# Spécifications



- Performance interface

- Exemples

- ✦ `performance.getEntries()`
- ✦ `performance.getEntriesByType("navigation")`
- ✦ `performance.getEntriesByType("resource")`

> `location.href`

< `'https://perso.liris.cnrs.fr/lionel.medini/enseignement/'`

> `performance.getEntriesByType("resource")`

< (13) [PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming, PerformanceResourceTiming] ⓘ

- ▶ 0: PerformanceResourceTiming {initiatorType: 'link', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 1: PerformanceResourceTiming {initiatorType: 'script', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 2: PerformanceResourceTiming {initiatorType: 'img', nextHopProtocol: '', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 3: PerformanceResourceTiming {initiatorType: 'img', nextHopProtocol: '', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 4: PerformanceResourceTiming {initiatorType: 'img', nextHopProtocol: '', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 5: PerformanceResourceTiming {initiatorType: 'script', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 6: PerformanceResourceTiming {initiatorType: 'script', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 7: PerformanceResourceTiming {initiatorType: 'link', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 8: PerformanceResourceTiming {initiatorType: 'fetch', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 9: PerformanceResourceTiming {initiatorType: 'css', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 10: PerformanceResourceTiming {initiatorType: 'css', nextHopProtocol: '', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 11: PerformanceResourceTiming {initiatorType: 'img', nextHopProtocol: '', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}
- ▶ 12: PerformanceResourceTiming {initiatorType: 'other', nextHopProtocol: 'http/1.1', workerStart: 0, redirectStart: 0, redirectEnd: 0, ...}

`length: 13`

> `[[Prototype]]: Array(0)`

# Spécifications



- Performance **interface**

- **Exemples**

- ✦ `performance.getEntriesByname("first-contentful-paint")`

```
> performance.getEntriesByName('first-contentful-paint')
< ▼ [PerformancePaintTiming] ⓘ
  ▼ 0: PerformancePaintTiming
    duration: 0
    entryType: "paint"
    name: "first-contentful-paint"
    startTime: 257.800000001192093
    ▶ [[Prototype]]: PerformancePaintTiming
    length: 1
    ▶ [[Prototype]]: Array(0)
```

- Pour tester : utiliser la console du navigateur

# Spécifications



- User Timing
  - Permet de “marquer” des points particuliers du déroulement du code
  - Les marques se retrouvent dans la liste Navigation Timing

```
async function run() {
  performance.mark("startTask1");
  await doTask1(); // Some developer code
  performance.mark("endTask1");

  // Log them out
  const entries = performance.getEntriesByType("mark");
  for (const entry of entries) {
    console.table(entry.toJSON());
  }
}
run();
```

# Concepts fondamentaux



- “Look-ahead parser”
  - Lit le HTML pour trouver les ressources à télécharger
    - ✦ Images, scripts, CSS...
  - Décide de la **priorité** de chargement
    - ✦ Rappel : en HTTP/1.1, les chargements se font en pipeline
  - Améliore le chargement des ressources de 30%  
(Steve Souders)

# Concepts fondamentaux



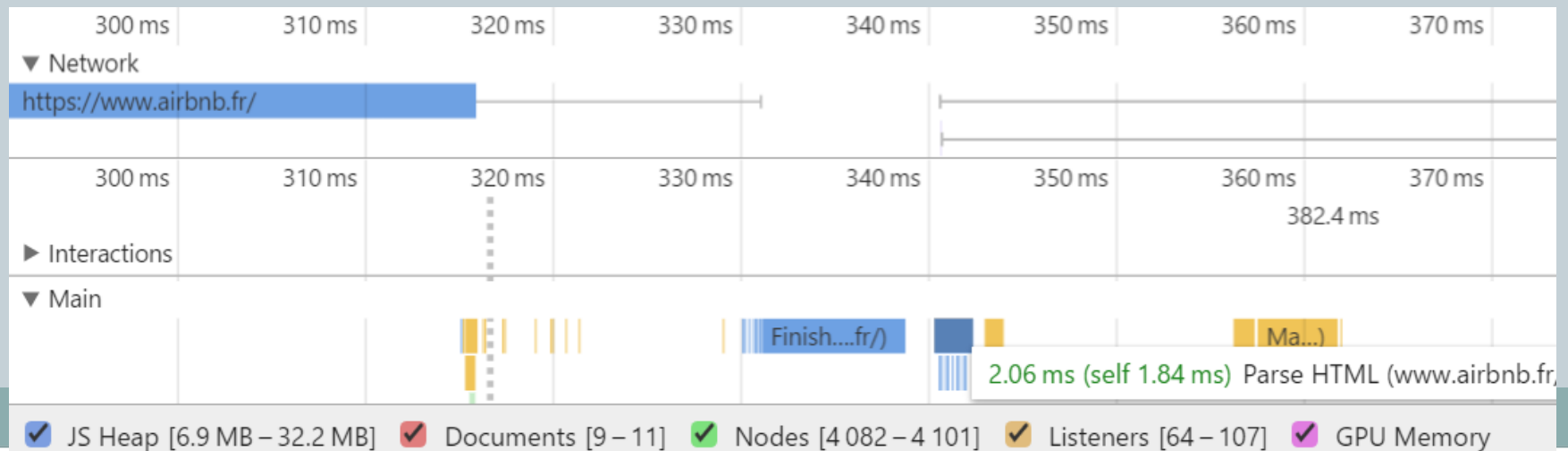
- Chemin Critique de Rendu (Critical Rendering Path)
  - De la réception des données (HTML, CSS, JS) à l'affichage des pixels à l'écran
    - ✦ Parsing HTML → composition du DOM
    - ✦ Parsing CSS → composition du CSSOM
    - ✦ Rendu → combinaison DOM / CSSOM, mise en page, affichage
  - Optimisation du CRP

**Optimiser le CRP** consiste à afficher le plus rapidement possible le contenu associé à la première action que l'utilisateur souhaite effectuer sur une page.

# Concepts fondamentaux



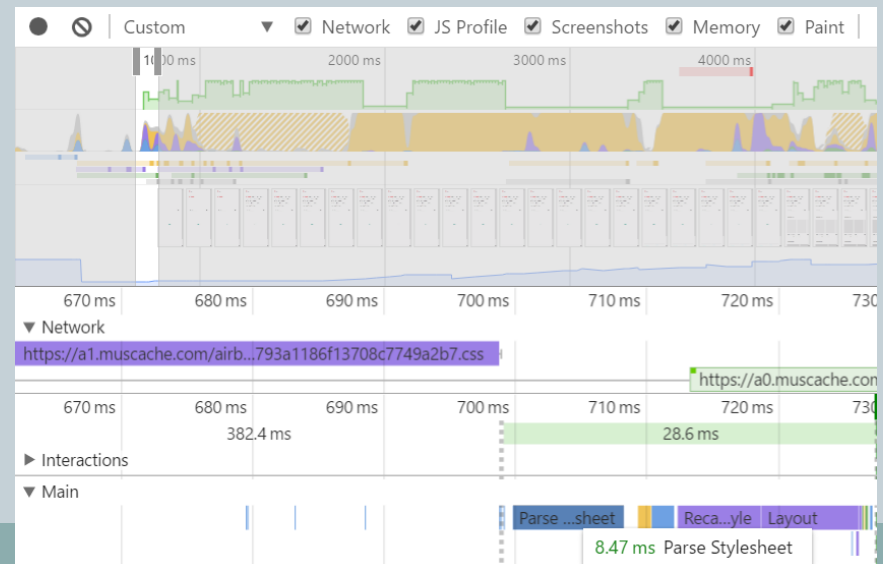
- Chemin Critique de Rendu (Critical Rendering Path)
  - Étapes de composition du DOM
    - ✦ Conversion : octets → caractères
    - ✦ Création de jetons (parsing, analyse syntaxique) : caractères → jetons
    - ✦ Analyse lexicale : jetons → objets (cf. spec. HTML5)
    - ✦ Construction du DOM : → arborescence d'objets
  - Exemple de visualisation (Chrome Dev Tools)



# Concepts fondamentaux



- Chemin Critique de Rendu (Critical Rendering Path)
  - Étapes de composition du CSSOM
    - ✦ Conversion : octets → caractères
    - ✦ Création de jetons (parsing, analyse syntaxique) : caractères → jetons
    - ✦ Analyse lexicale : jetons → objets (cf. spec. CSS3)
    - ✦ Construction du CSSOM : → arborescence d'objets
  - Exemple de visualisation (Chrome Dev Tools)

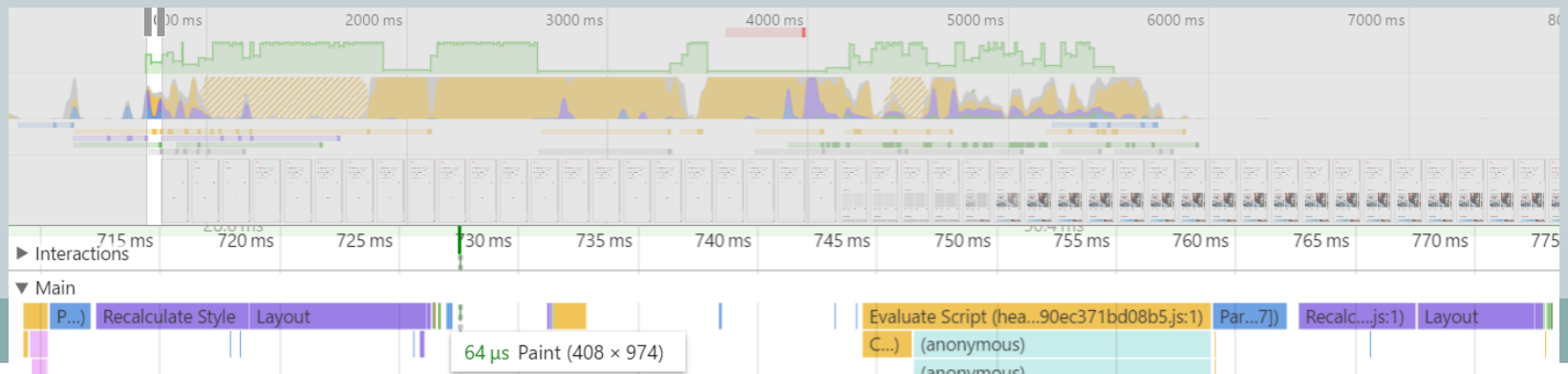




# Concepts fondamentaux



- Chemin Critique de Rendu (Critical Rendering Path)
  - Étapes de Rendu
    - ✦ Combinaison DOM / CSSOM → « Arborescence d’affichage » (Render Tree)
      - Arborescence des nœuds **visibles**, avec leurs styles calculés
    - ✦ Mise en page (Layout) → « modèle de zones »
      - Position et taille exacte de chaque élément dans la fenêtre
    - ✦ Affichage, « pixellisation » (Paint) → conversion des nœuds en pixels
    - ✦ Composition → assemblage des images des composants



# Concepts fondamentaux



- Chemin Critique de Rendu (Critical Rendering Path)
  - Remarques
    - ✦ Durée de chaque étape : de quelques ms à ...
      - En fonction
        - De la taille du document
        - De la complexité des styles
        - Des capacités de calcul de l'appareil
    - ✦ Ce processus est (partiellement) déclenché
      - Au premier chargement de la page
      - À chaque actualisation du DOM, du CSSOM
        - (Re-)chargement de données
        - Script
      - À chaque redimensionnement de la fenêtre

# Concepts fondamentaux



- Chemin Critique de Rendu (Critical Rendering Path)
  - Remarques
    - ✦ L'optimisation du CRP permet
      - D'accélérer l'affichage de la page au chargement
      - D'augmenter le taux d'actualisation (FPS) pour les contenus interactifs
    - ✦ Le CSS peut être bloquant (en fonction des media queries)
      - Il doit être disponible le plus rapidement possible
    - ✦ Les scripts sont bloquants
      - Le seul moyen pour le navigateur de le savoir est de l'exécuter
      - Le moteur JS « attend » l'arbre CSSOM

# Concepts fondamentaux



- Application shell (aka « app shell »)
  - Structure de l'application “nue”
  - Souvent fondé sur un framework JS
  - Optimise le chargement de la vue pour ses éléments principaux



“An **application shell** is the minimal HTML, CSS, and JavaScript powering a user interface. The application shell should:

- ✦ load fast
- ✦ be cached
- ✦ dynamically display content”
- ✦ Source : [Chrome developers blog](#) (Addy Osmany)

# Concepts fondamentaux



- **Rafrâichissement du rendu (layout)**

- Causes

- ✦ Action utilisateur
- ✦ Script



- Remarques

- ✦ Le layout remet en général en forme la totalité du document
  - ➔ Éviter les rafraîchissements
- ✦ La performance dépend du nombre d'éléments du DOM

- Consignes

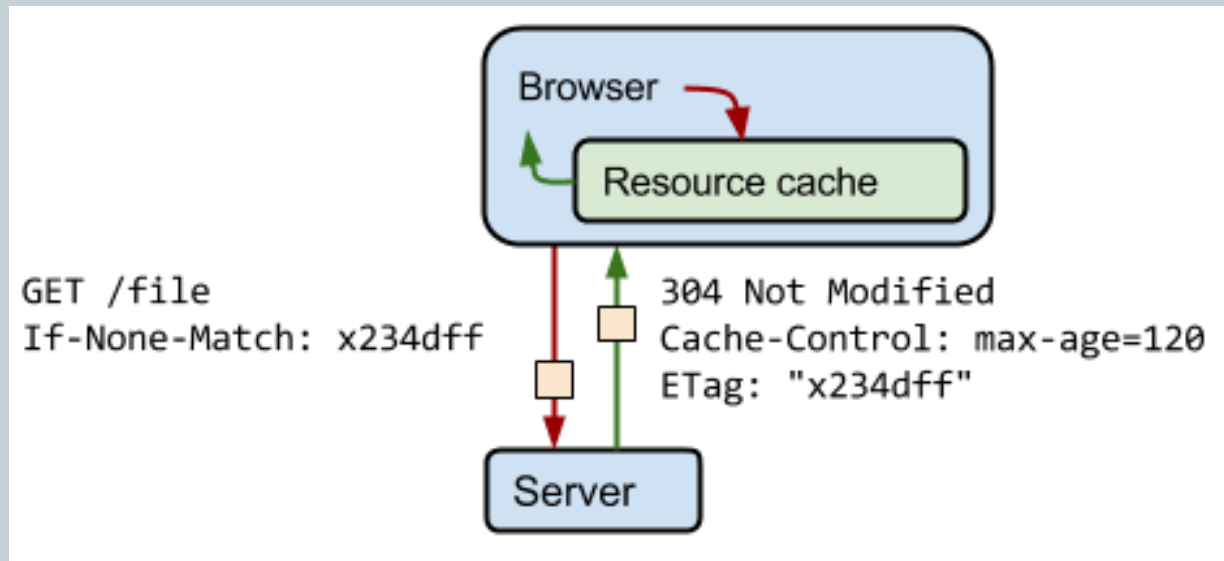
- ✦ Éviter les “forced synchronous layouts”

# Concepts fondamentaux



- Caching

- Rappel : un échange HTTP avec gestion du cache



✦ Source :

<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching>

# Concepts fondamentaux



- **Caching**
  - Rappel : les [directives Cache-Control](#)
  - Règles à respecter
    - ✦ Utiliser des URL cohérentes
    - ✦ S'assurer que le serveur fournit un ETag
    - ✦ Utiliser les CDN
    - ✦ Définir des règles optimales pour Cache-Control
    - ✦ Invalider et mettre à jour les réponses mises en cache
    - ✦ Découper les ressources en fonction de leur cycle de vie

# Méthodes d'optimisation



- Première chose à faire : capturer des métriques
  - Real User Measurements (RUM)
    - ✦ Outils de capture des requêtes
      - Permettent de connaître le trafic sur votre site
      - Permettent de connaître les temps de réponse
      - Permettent de connaître les **fonctionnalités utilisées**
    - ✦ Exemples
      - [Google Analytics](#)
      - [Piwik](#)
      - [Soasta mPulse](#)
      - [Episodes](#)
  - Synthèse des données
    - ✦ [WebPageTest](#)



# Méthodes d'optimisation



- Optimisation du chargement
  - Principe : optimiser chaque octet
    - ✦ La ressource la plus optimisée est celle qui n'est pas envoyée
    - ✦ Étudier la valeur de chaque ressource
      - Par rapport à son intérêt pour l'utilisateur
      - Par rapport à son intérêt pour les concepteurs
      - Pour savoir
        - Si elle doit être envoyée
        - Si elle doit faire partie du CRP

# Méthodes d'optimisation



- Optimisation du chargement

- Techniques

- ✦ Compresser les données

1. Choisir quelles données envoyer en fonction du contexte
2. Appliquer les optimisations spécifiques au contenu (HTML, CSS, JS)
3. Utiliser la compression binaire (GZIP)

- ✦ Choisir les formats optimaux

- Cf. images, slide suivant

- ✦ Réévaluer régulièrement la valeur des ressources

# Méthodes d'optimisation



- Optimisation du Chemin Critique de Rendu
  1. Analyser le CRP
    - ✦ Ressources bloquantes et non bloquantes
    - ✦ Ressources asynchrones
    - ✦ Redéclenchement du rendu
  2. Optimiser le CRP
    - ✦ Réduire le nombre de ressources critiques
    - ✦ Optimiser l'ordre de chargement des ressources critiques restantes
    - ✦ Optimiser le nombre d'octets critiques
      - Nombre de ressources (allers et retours réseau)
      - Tailles des ressources

# Méthodes d'optimisation



- Optimisation du Chemin Critique de Rendu
  - Rappel : ressources critiques (bloquantes)
    - ✦ CSS
      - Placer le code / la balise *link* dans l'entête du document
      - Éviter les directives CSS *@import*
      - Éventuellement, mettre le code CSS directement dans le document
    - ✦ JS
      - Retarder le chargement (attribut *async*)
      - Retarder l'exécution (attribut *defer*)
      - Le placer en fin de page
      - Séparer les scripts (pas plus de 50 ms)
      - Utiliser des Web Workers / Service Workers pour les traitements longs

# Méthodes d'optimisation



- **Optimisation JavaScript**
  - Démo du fonctionnement interne du moteur JS V8
    - ✦ [IRHydra2](#) (déprécié)
  - Mise à jour de la vue
    - ✦ Utiliser le mécanisme de rafraîchissement du navigateur
      - Éviter `setTimeout()` et `setInterval()`
      - Utiliser `requestAnimationFrame()`
  - Réduire la complexité des calculs en tâche principale
    - ✦ Modulariser
    - ✦ « Asynchroniser »
    - ✦ Passer en tâche de fond (workers)
  - Prendre en compte la “JS frame tax”
    - ✦ À chaque rafraîchissement
    - ✦ Les micro-optimisations sont souvent contre-productives

# Méthodes d'optimisation



- Images

- Pistes d'optimisation

- ✦ Préférer les styles / animations CSS3 aux images
- ✦ Choisir entre vectoriel ou matriciel
- ✦ Choisir le meilleur format (matriciel)
  - En fonction des besoins (animations)
  - En fonction des navigateurs (formats )
    - JPEG XR (IE)
    - WebP (Blink)
- ✦ Choisir le niveau de qualité nécessaire
- ✦ Compresser
- ✦ Automatiser le processus de renvoi par le serveur

# Méthodes d'optimisation



- Images (matricielles)
  - Progressive rendering
    - ✦ Par défaut
      - Rendu de haut en bas, au fur et à mesure de l'arrivée des données
    - ✦ Optimisation
      - Rendu interlacé
        - 1D (GIF)
        - 2D (PNG)
        - Coût : +20% sur un PNG
    - ✦ Explication sur [Coding Horror](#)
  - Bonne pratique
    - ✦ Dans tous les cas, préciser hauteur et largeur pour “réserver” l'espace dans la page

# Méthodes d'optimisation



- Comportement en cas de mauvaise connectivité
  - Caractéristiques
    - ✦ Low Bandwidth
    - ✦ High Latency
  - Outils de test
    - ✦ Depuis des localisations différentes
    - ✦ Diminution volontaire de la bande passante
      - Throttling
  - Outil de remédiation
    - ✦ Mettre des timeouts

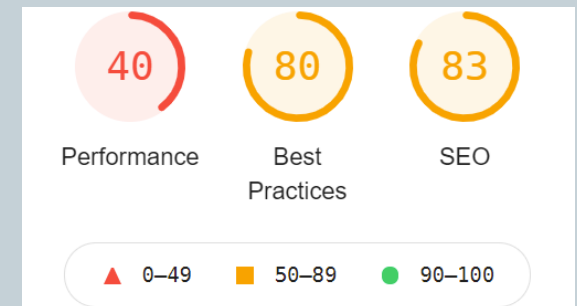
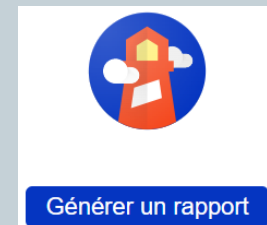


# Outils



- Google PageSpeed / LightHouse
  - Analyse et optimisation d'un site en fonction des bonnes pratiques

The screenshot shows the Google PageSpeed Insights interface. At the top, the URL 'http://www.google.com/' is entered in the search bar, with an 'ANALYZE' button to the right. Below the search bar, there are two tabs: 'Mobile' (selected) and 'Desktop'. The main content area is divided into two columns. The left column displays performance metrics: '99 / 100 Speed' with a 'Consider Fixing' status, listing 'Minify JavaScript' as a suggestion; '9 Passed Rules'; and '100 / 100 User Experience' with a 'Congratulations! No issues found.' status. The right column features a mobile device simulator showing the Google homepage.



✦ Source : [PageSpeed](https://pagespeed.web.dev/)

# Outils

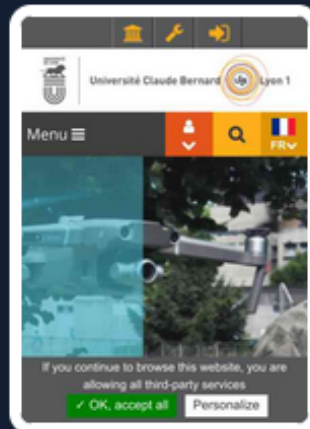


- WebPageTest

- Audit de performance (similaire à LightHouse)
- Requêtage depuis différentes zones géographiques
- Comparaison de performances entre sites

## Webpage Performance Test Result

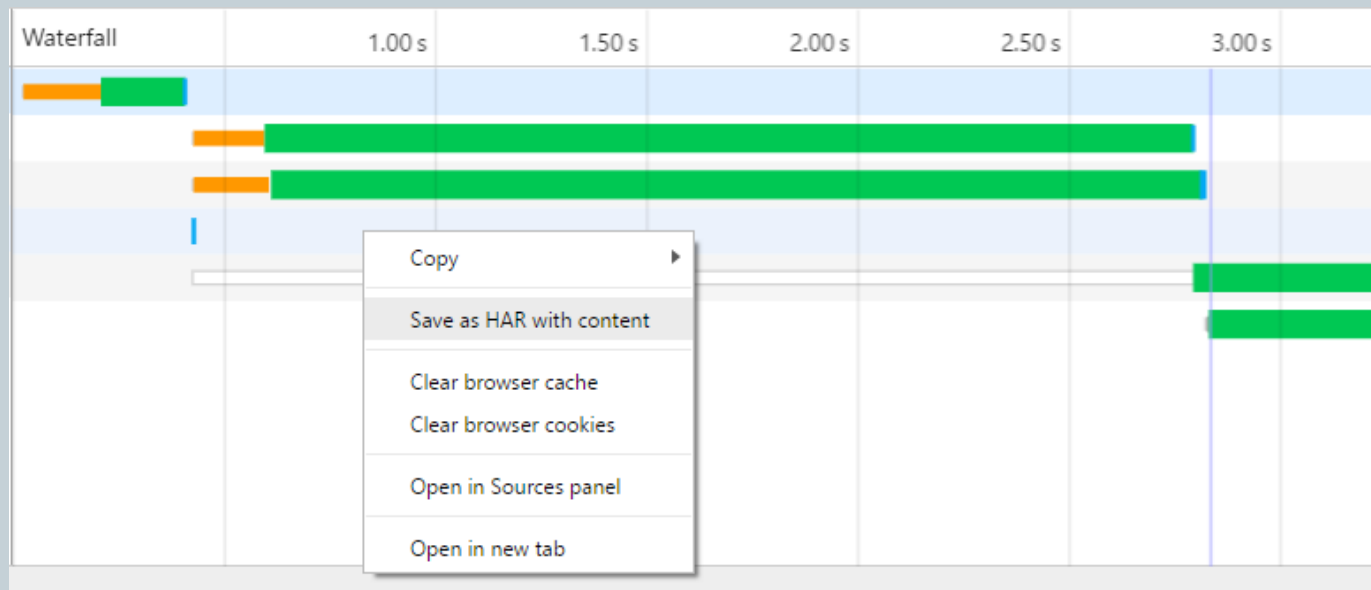
SETTINGS: MOTOG4



# O Outils



- HTTP Archive Format (HAR)
  - Permet d'enregistrer les données de performance d'un site
  - Objectifs
    - ✦ Partager avec les auteurs du site



# Modèles et patterns



- **Le pattern PRPL**

- Proposé par Google au [Google I/O 2016](#)
- Dédié aux “Progressive Web Apps”
  - ✦ Mais applicable aussi aux SPA...
- 4 étapes

1. Push critical resources for the initial URL route
2. Render initial route
3. Pre-cache remaining routes
4. Lazy-load and create remaining routes on demand

- Remarque
  - ✦ Peut être appliqué partiellement
- Source : [Google Fundamentals](#)

# Modèles et patterns

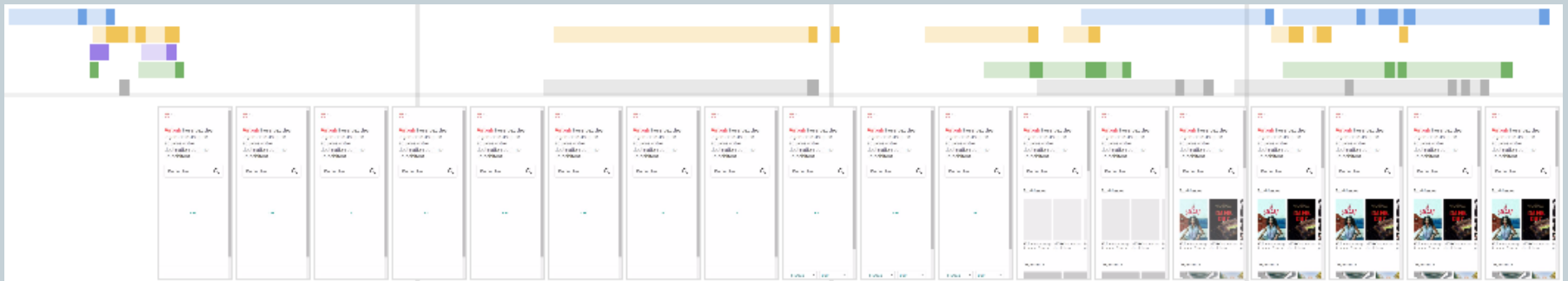


- Le pattern PRPL
  - HTTP/2
    - ✦ Server push
      - Le serveur “sait” de quoi le client va avoir besoin
      - Il peut “pousser” les données avant qu’il les demande
        - ➔ Nécessite de prendre “la bonne décision”
    - ✦ Parallélisation des transactions
      - Plusieurs requêtes-réponses peuvent être envoyée en parallèle
        - ➔ Gain de temps (suppression de l’état “Queuing”)

# Modèles et patterns



- Le pattern PRPL
  - Rendu de la route initiale
    - ✦ Revient à optimiser le CRP
    - ✦ [Progressive Web Apps](#)
    - ✦ Single Page Applications
      - Une seule vue côté client
      - Application Shell



# Modèles et patterns



- Le pattern PRPL
  - Pré-rendu des routes secondaires
    - ✦ Suite du principe d'optimisation du CRP
    - ✦ Une fois que la route principale est chargée
    - ✦ Nécessite de structurer “découper” l'application en composants
      - Fonctionnels
      - De vue
        - Souvent, la vue a été préchargée (cf. CSS bloquant)

# Modèles et patterns



- Le pattern PRPL

- Lazy-loading

- ✦ Permet de ne demander des contenus que quand l'utilisateur en a besoin
- ✦ Souvent associé à un événement utilisateur
  - Changement de route client (hash)
  - Scroll
- ✦ Requête asynchrone sur une partie des données
  - Données paginées
  - Nécessite de savoir “dimensionner” les ressources

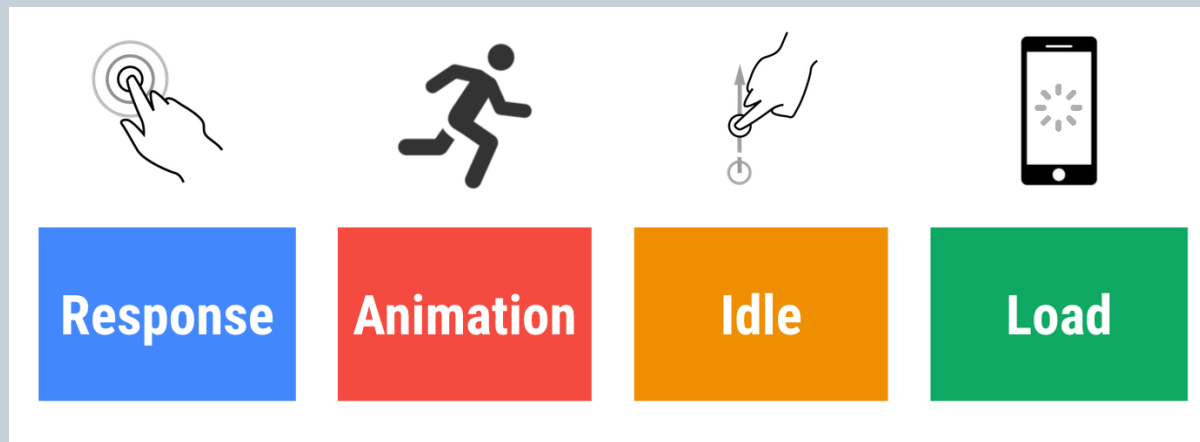


# Modèles et patterns



- Le modèle [RAIL](#)

“RAIL is a **user-centric** performance model that provides a **structure for thinking about performance**. The model breaks down the user's experience into key actions (for example, tap, scroll, load) and helps you **define performance goals** for each of them.”



“[...] Users have different performance expectations for each of these contexts, so **performance goals are defined based on the context** and [UX research on how users perceive delays](#).”

# Modèles et patterns



- Le modèle RAIL
  - Response: process events in under 50ms
    - ✦ Max = 100ms : 50ms input handling, 50ms idle
  - Animation: produce a frame in 10 ms
    - ✦ “Produce each frame in an animation in 10ms or less.”
  - Idle: maximize idle time
    - ✦ “Maximize idle time to increase the odds that the page responds to user input within 50 ms.”
  - Load: deliver content and become interactive in under 5 seconds
    - ✦ “Optimize for fast loading performance relative to the device and network capabilities”

# Conclusion



- Garder à l'esprit que la performance n'est pas l'objectif principal du site
- ...mais une condition de son utilisation
  - Jusqu'où dégrader les fonctionnalités pour la performance ?
  - Jusqu'où dégrader la performance pour les fonctionnalités ?
- La barrière des “3 secondes” n'est pas absolue
  - La vraie vitesse de chargement optimale est :

**“faster than your competitors”**

(Steve Souders)

# Références



- <https://www.w3.org/2016/07/webperf>
- <https://developers.google.com/web/fundamentals/performance/>
- <https://developer.yahoo.com/performance/>
- <http://stevesouders.com/>
- <https://developers.google.com/speed/pagespeed/>
- <http://www.perfplanet.com/>