M1IF03
Conception d'applications Web

PERFORMANCE DES APPLICATIONS WEB

LIONEL MÉDINI

OCTOBRE-DÉCEMBRE 2018

Plan du cours

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

Introduction

- Position du problème
  - Pas de phase d'installation pour une webapp

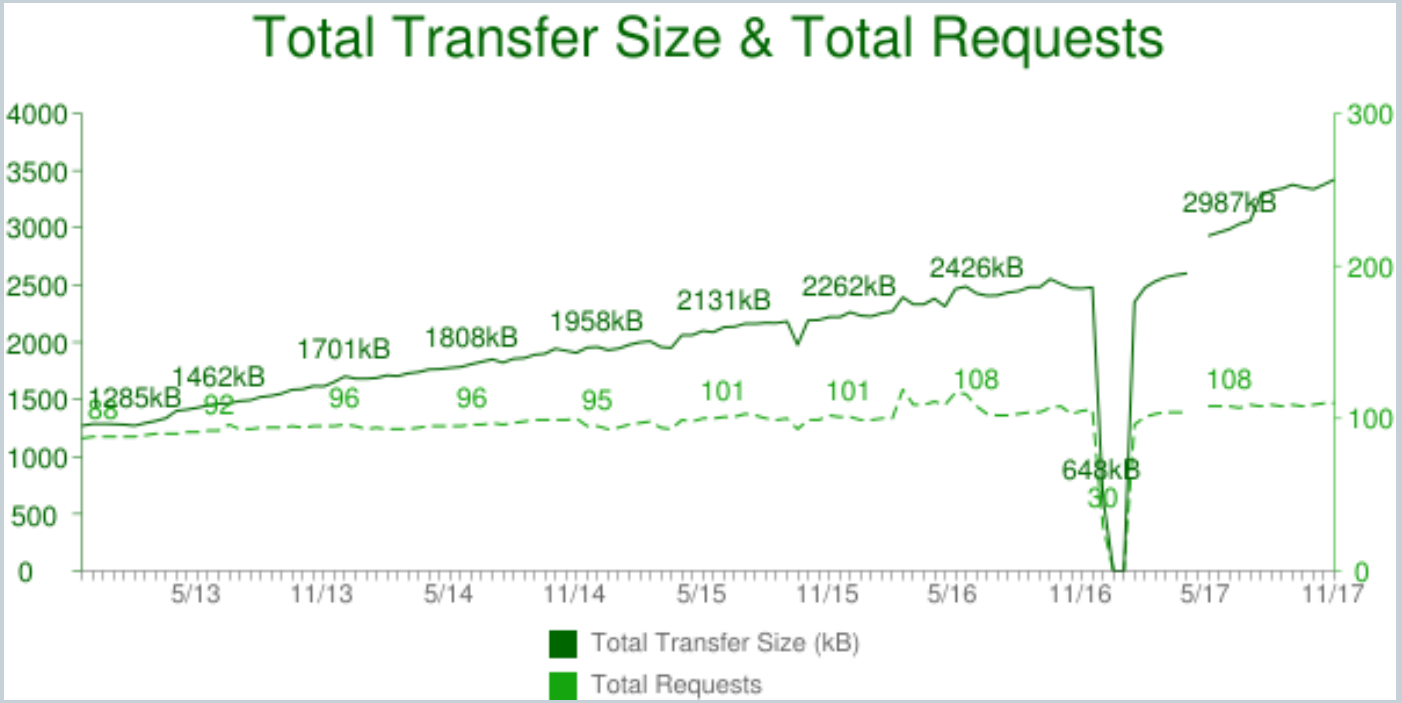
    ➔ Il faut que ça marche tout de suite !

  - 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

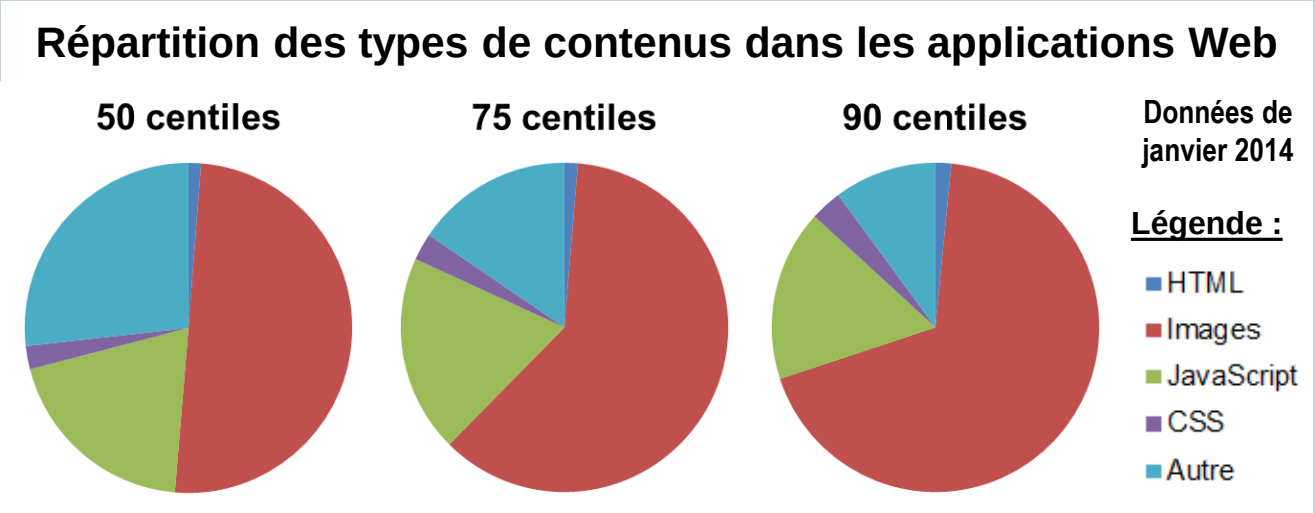
- Constats
  - Augmentation de la taille des applications Web



  - Source : [HTTP Archive]

Introduction

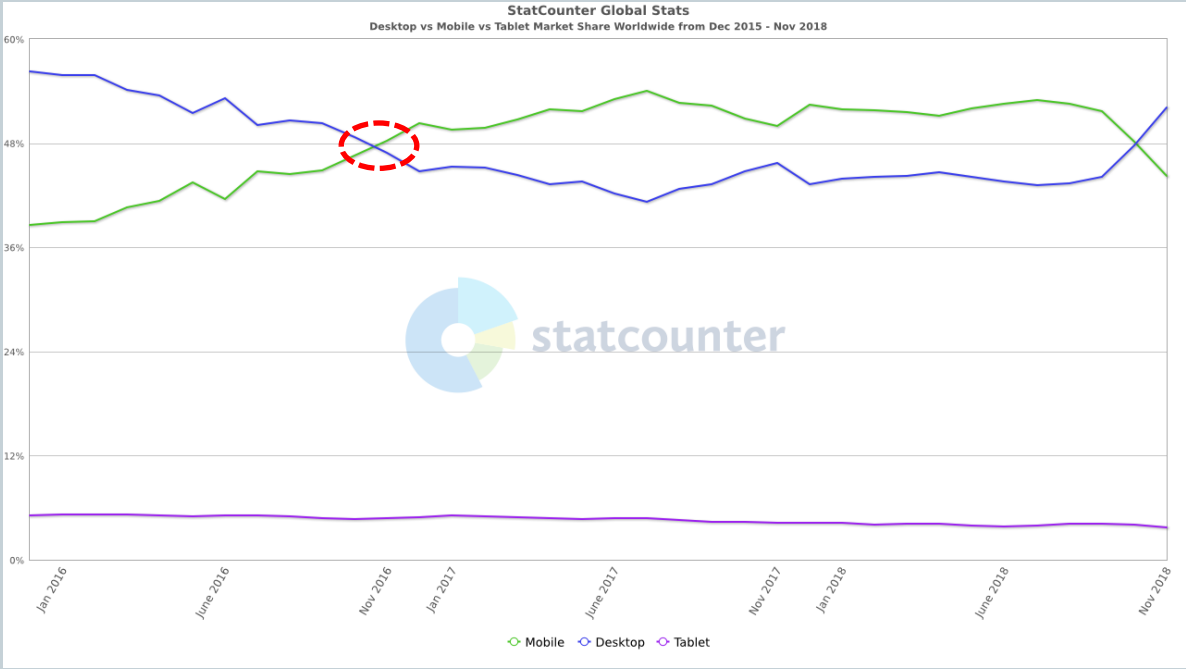
- Constats
  - Augmentation de la complexité des applications Web



    - 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]

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

    - 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
      - …

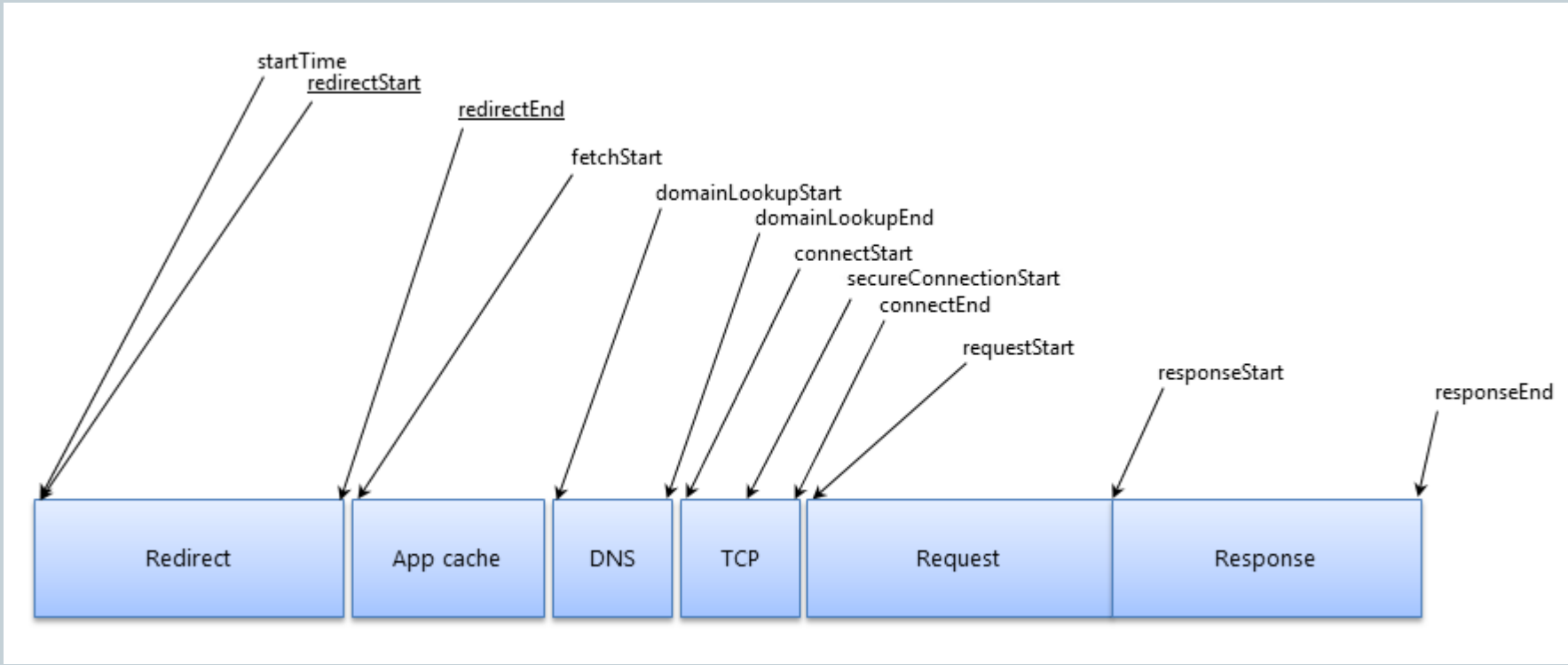
➔ **Rappel : c'est côté client que se mesure l'expérience utilisateur**

Introduction

- Initiatives / ressources
  - W3C Web Performance WG
    - Dates : juillet 2016 → mars 2018
    - Objectifs : “user agent features and APIs to observe and improve aspects of application performance”
    - Résumé : Web Performance Timing APIs Primer (Editor’s draft)
    - Quelques specs
      - Performance Timeline ~~Level 1 (déc. 2013),~~ Level 2 (WD, nov. 2018)
      - Resource Timing Level 1 (CR, mar. 2017), Level 2 (WD, nov. 2018)
      - Navigation Timing ~~Level 1 (TR, déc. 2012),~~ Level 2 (WD, nov. 2018)
      - User Timing Level 1 (TR, déc. 2013), Level 2 (ED, nov. 2018)…
  - Google Fundamentals
  - Steve Souders

Concepts fondamentaux

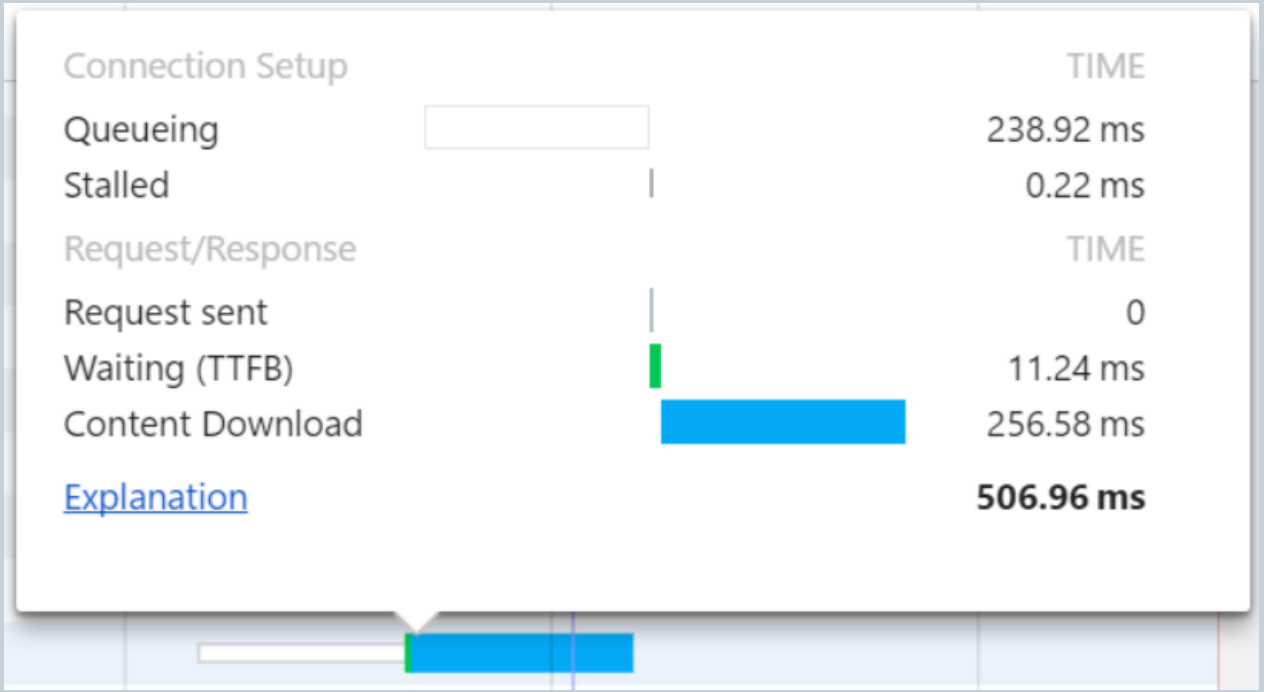
- Resource Timing
  - De la réception de l'URL à l'arrivée de la réponse



- Source : [Web Performance Timing API Primer]

Concepts fondamentaux

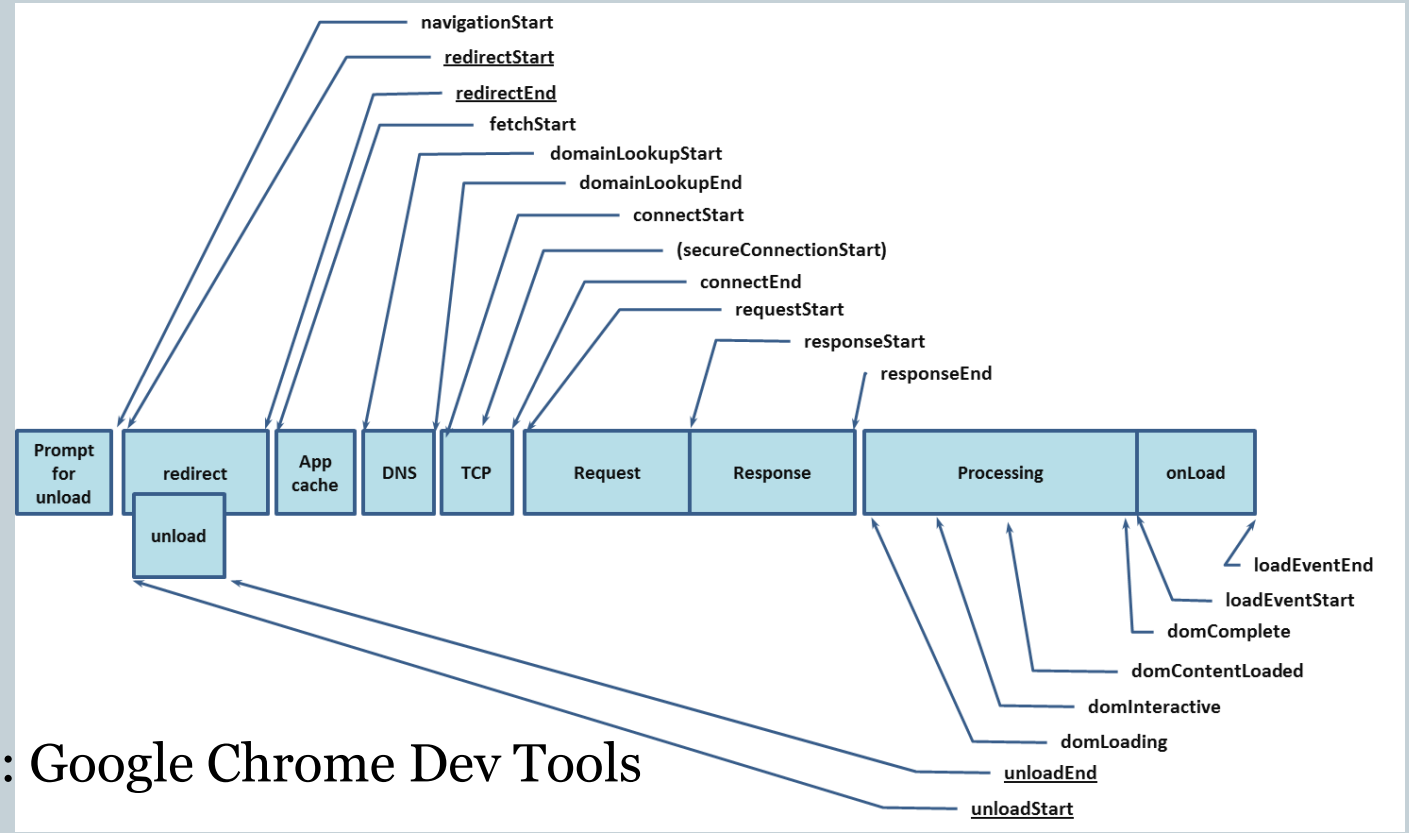
- Resource Timing
  - Une fois la connexion TCP obtenue, plusieurs états pour la transaction



- Source : Google Chrome Dev Tools

Concepts fondamentaux

- Navigation Timing
  - Événements liés aux différentes étapes de navigation



- Source : Google Chrome Dev Tools

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, les chargements se font en pipeline
  - Améliore le chargement des ressources de 30% (Steve Souders)

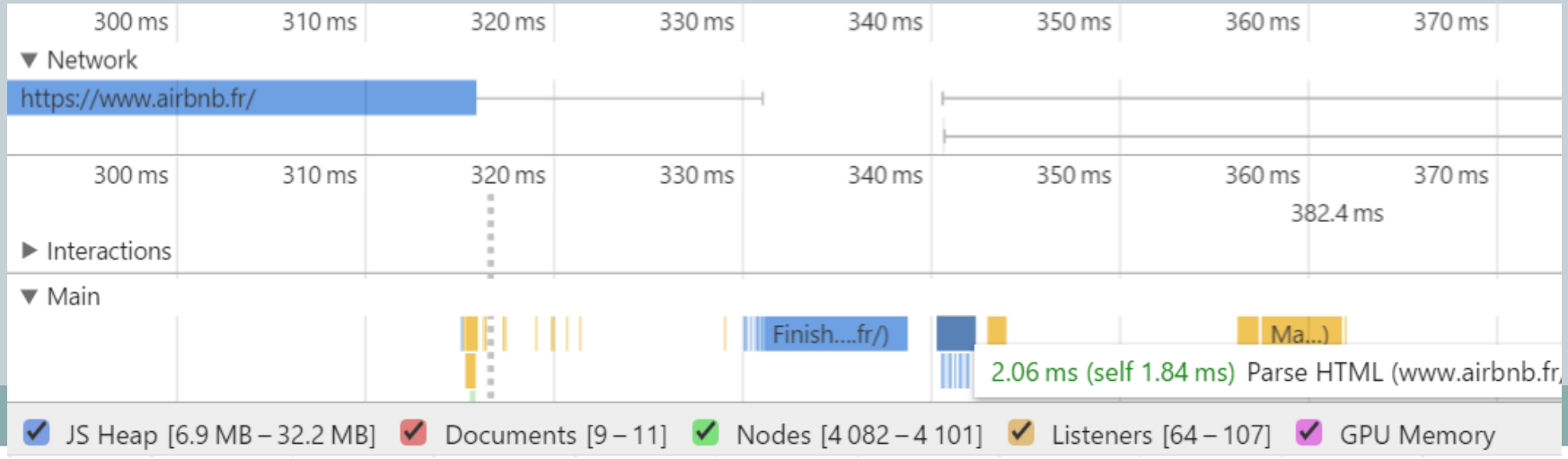
Concepts fondamentaux

- 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 CSSDOM
    - Rendu → combinaison DOM / CSSDOM, 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.

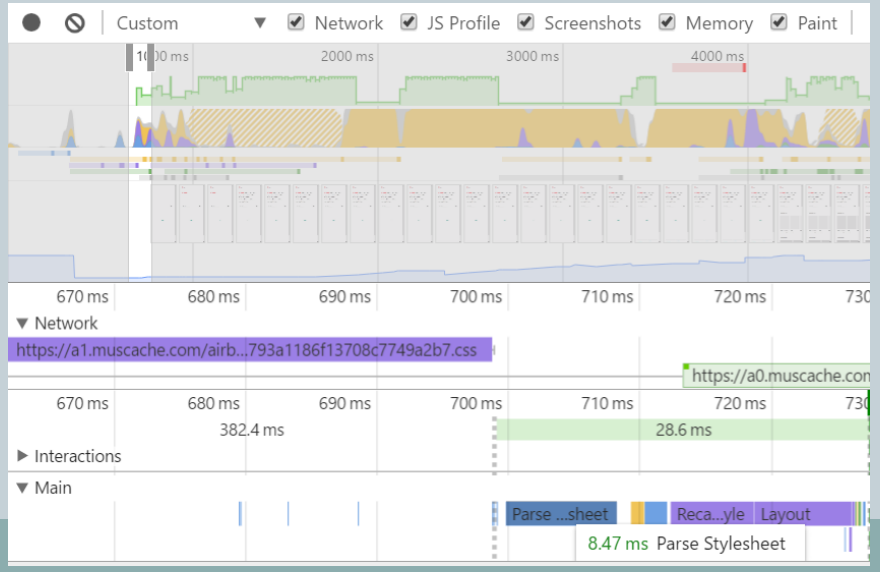
Concepts fondamentaux

- 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)



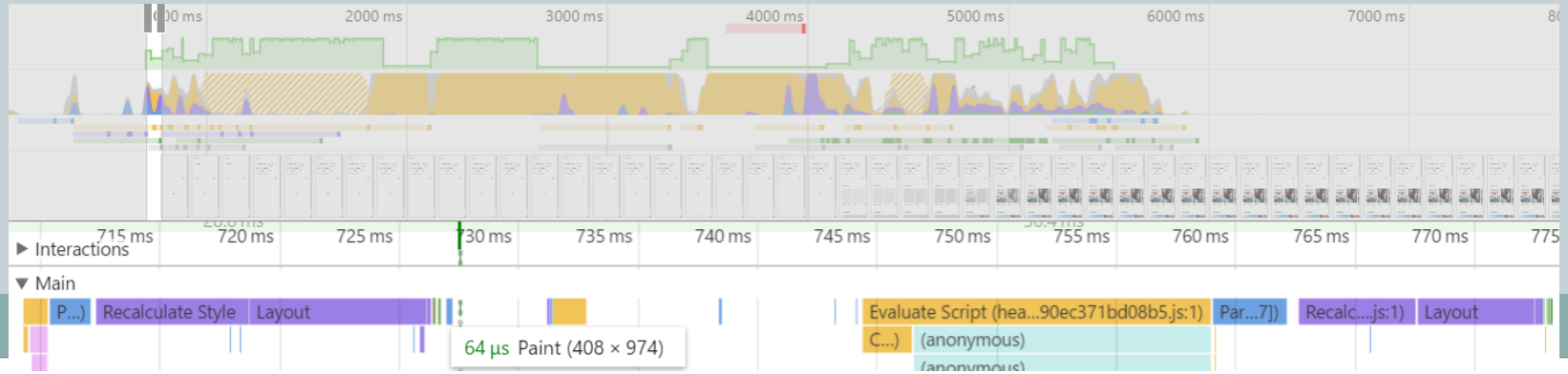
Concepts fondamentaux

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



Concepts fondamentaux

- Chemin Critique de Rendu (Critical Rendering Path)
  - Étapes de Rendu
    - Combinaison DOM / CSSDOM → « 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

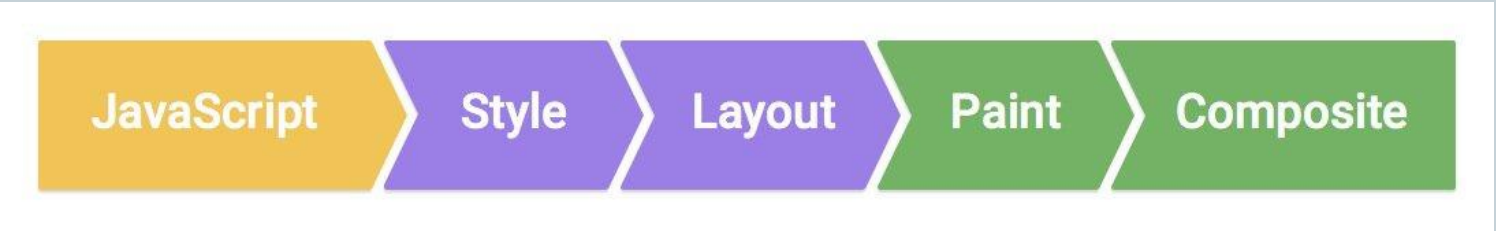
- Chemin Critique de Rendu (Critical Rendering Path)
  - Remarques
    - Durée de chaque étape : de quelques ms à …
      - En fonction
        - De la taille du document
        - De la complexité des styles
        - Des capacités de calcul de l'appareil
    - Ce processus est (partiellement) déclenché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

Concepts fondamentaux

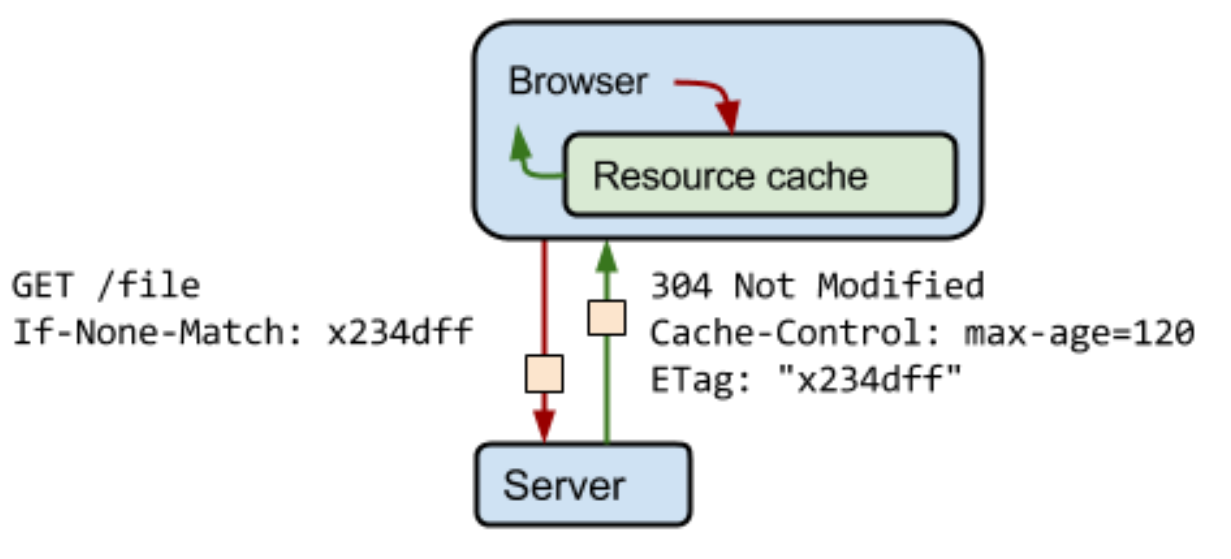
- Rafraîchissement du rendu (layout)
  - Causes
    - Action utilisateur
    - Script



  - 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 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 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
    - « Asynchoniser »
    - 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

Méthodes d'optimisation

- Le pattern PRPL
  - Proposé par Google au Google I/O 2016
  - Dédié aux "Progressive Web Apps"
  - 4 étapes

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

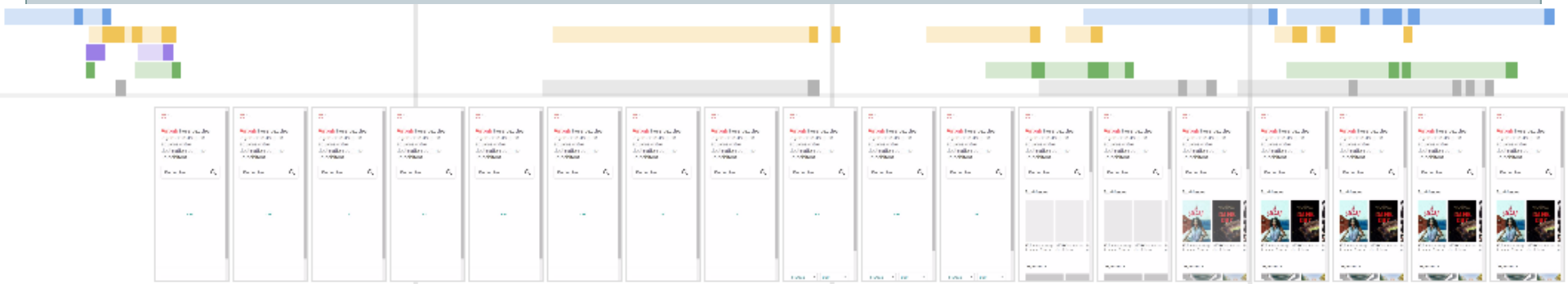
  - Remarque
    - Peut être appliqué partiellement
  - Source : Google Fundamentals

Méthodes d'optimisation

- 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")

Méthodes d'optimisation

- 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"
          - Structure de l'application "nue"
          - Souvent fondé sur un framework JS
      - Optimise le chargement de la vue pour ses éléments principaux



Méthodes d'optimisation

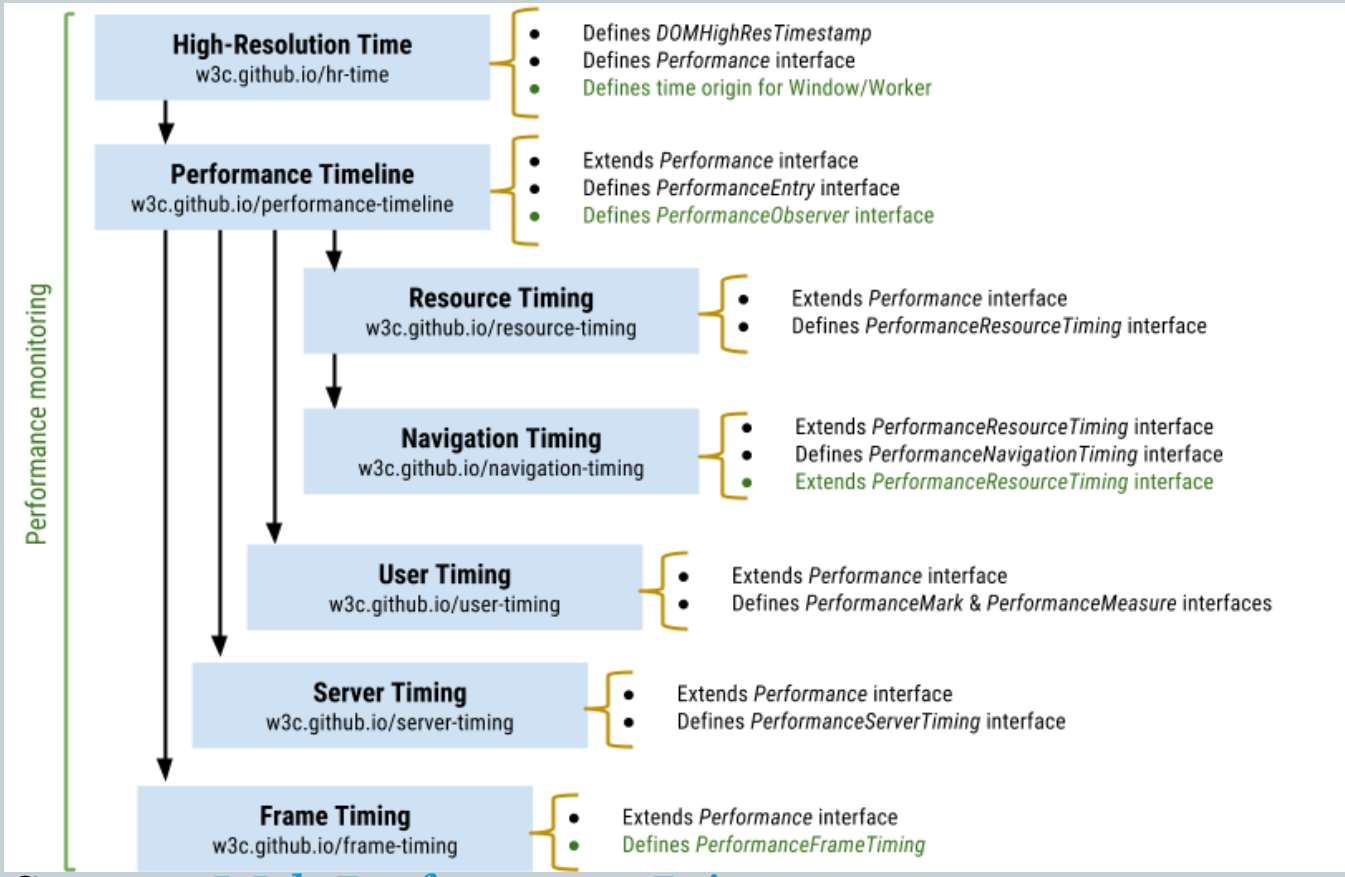
- Le pattern PRPL
  - Pré-rendu des routes secondaires
    - Suite du principe d'optimisation du CRP
    - Une fois que la route principale est chargée
    - Nécessite de structurer "découper" l'application en composants
      - Fonctionnels
      - De vue
        - Souvent, la vue a été préchargée (cf.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

Méthodes d'optimisation

- 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

Autres outils

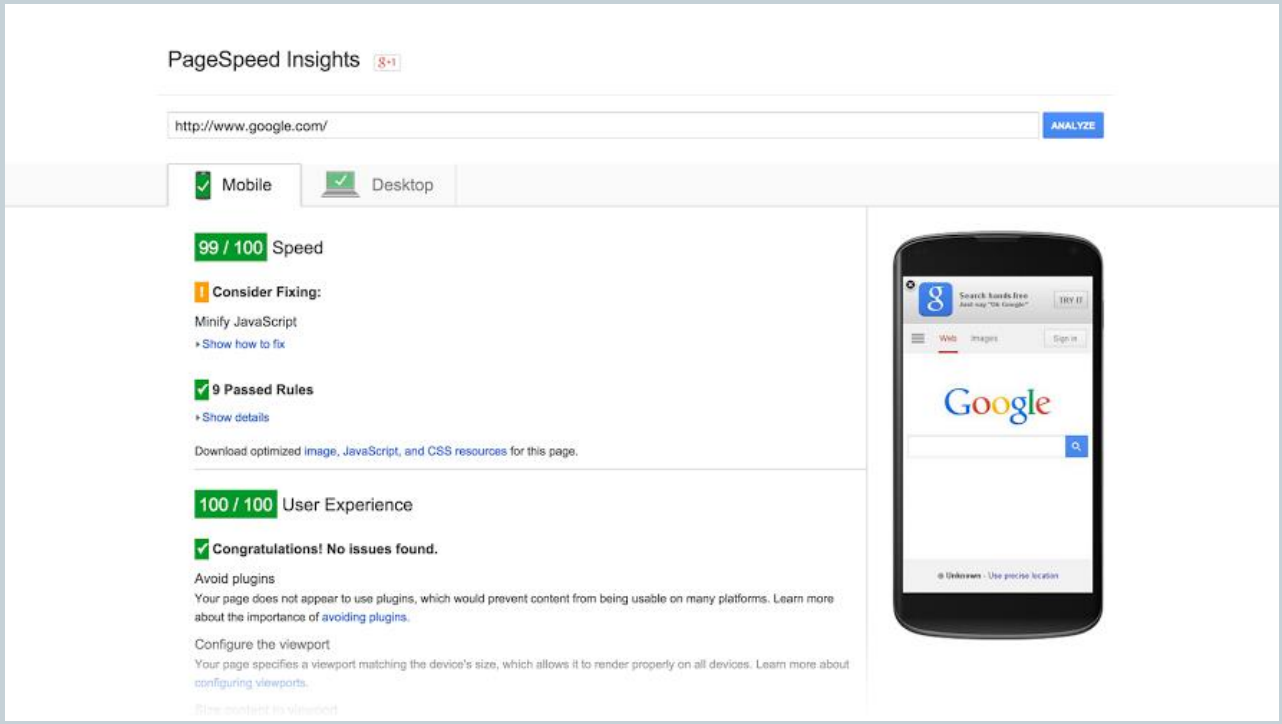
- APIs d'accès à la performance d'une application



- Source : [Web Performance Primer]

Autres outils

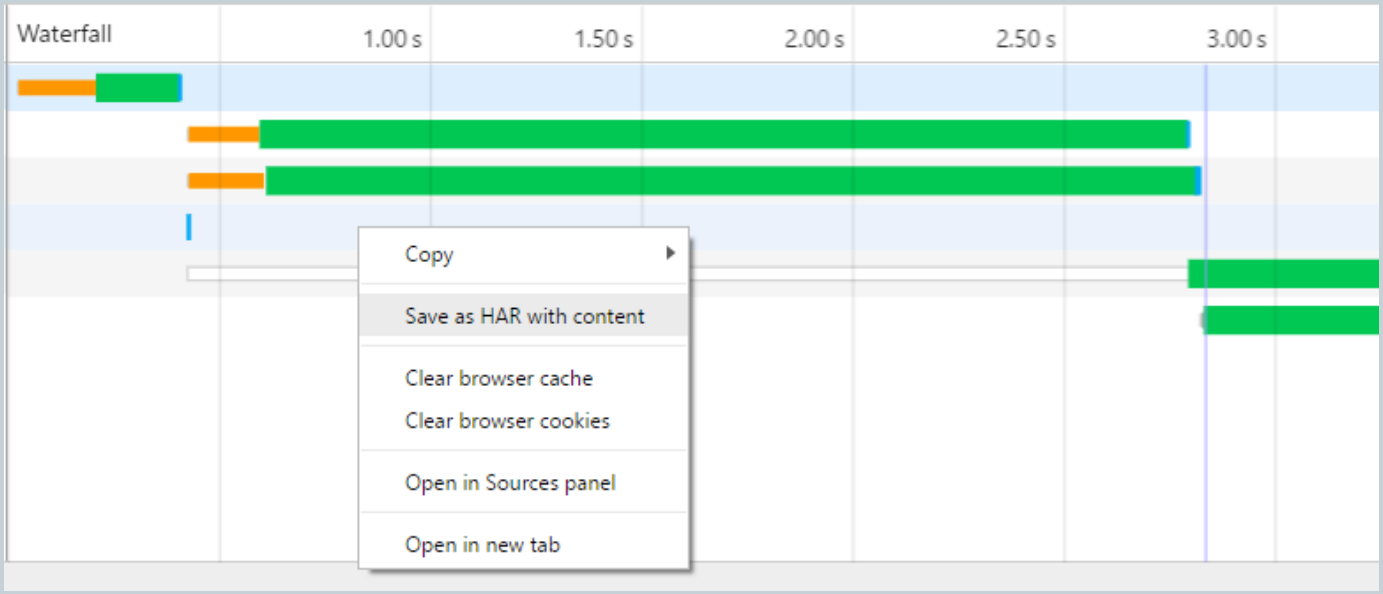
- Google PageSpeed
  - Analyse et optimisation d'un site en fonction des bonnes pratiques



- Source : [PageSpeed]

Autres outils

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



Autres outils

- Exemple d'ordonnancement de ressources
  - Visualiser la façon dont les ressources sont chargées dans un navigateur
  - Télécharger en asynchrone (parallèle) les ressources "lourdes"
    - Images…
  - Démo : Cuzillion

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/