

Hvorfor er nettsiden min treig?

Et dypdykk i ytelse på web

Knut Haugen 2025

Og ikke minst, hva kan vi gjøre med det?



Knut Haugen

Utvikler siden år 2000

NRK siden 2017

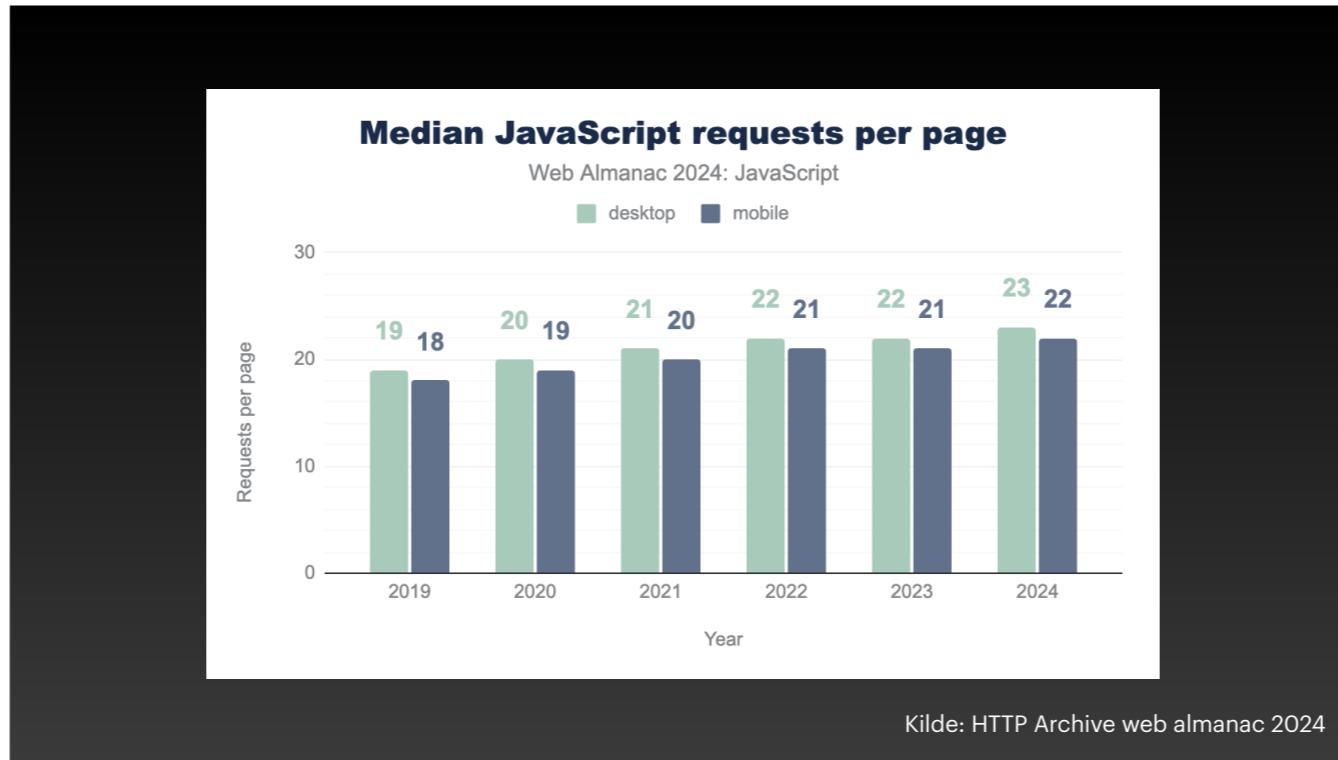
Jobber med strømming av video og avspillere

Et av mine mål i livet, er å ikke bry meg om ting jeg ikke kan gjøre noe med. Men ytelse kan jeg gjøre noe med.

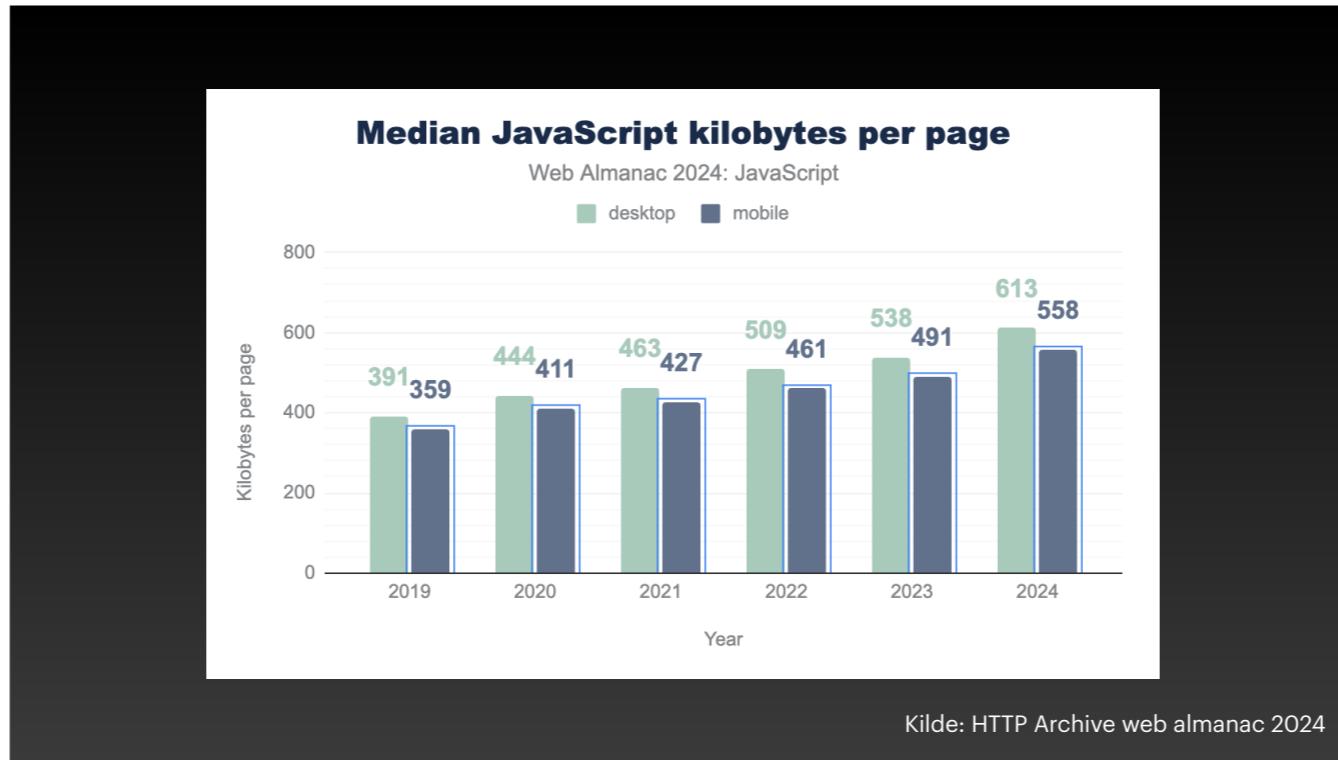
Jeg kommer til å snakke mye om chrome i dag. Selv om Chrome har en del problematiske sider ved seg, er det de som er helt i front når det gjelder ytelse, både verktøy og ytelses-relaterte api'er.

Hvordan står det til med webben?

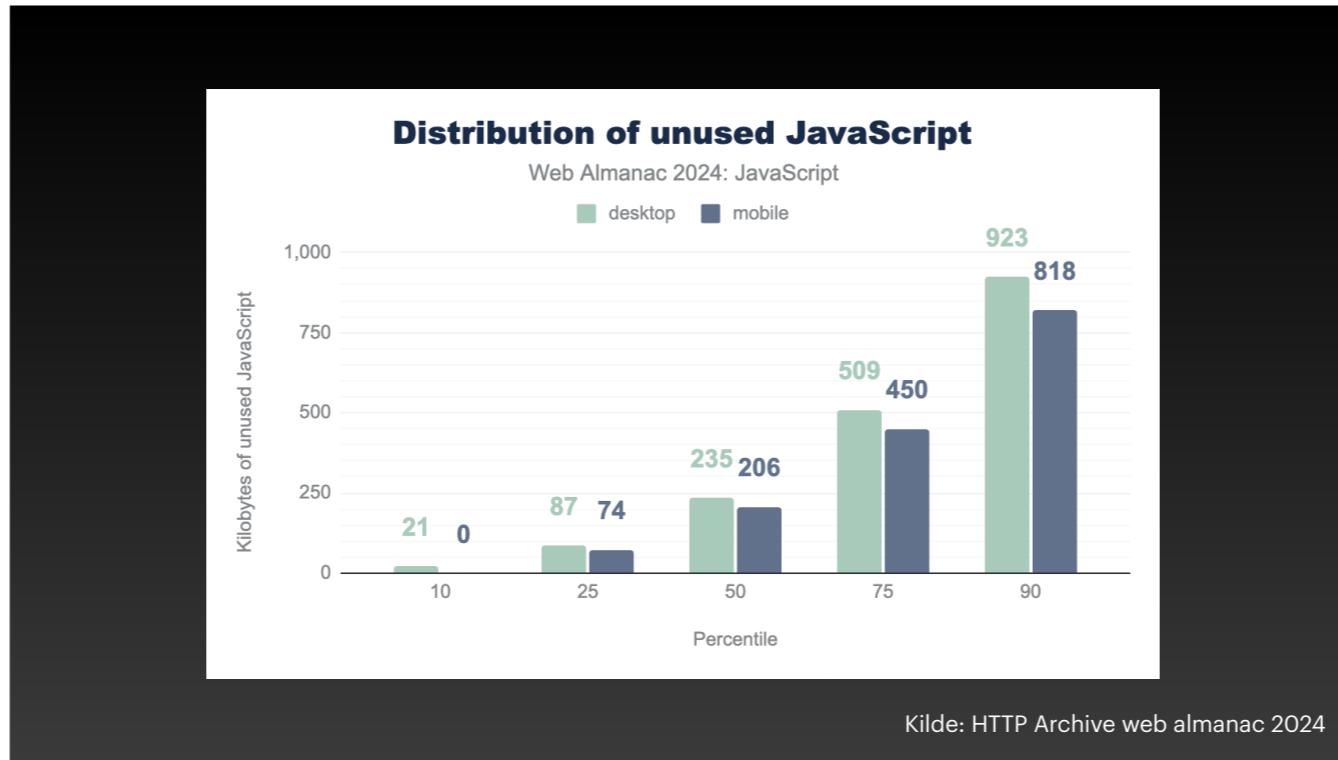
Det som har mest å si for ytelsen er javascript som kjører. Js som skal parses ved lasting, funksjoner som skal kalles, endringer som skal gjøres på siden etc. Og hva er det vi driver med, som utviklere? Vi skriver js.



Antall requester går opp, 90 percentilen har nesten 70 requests



antall bytes går opp.



Vi shipper mer js enn vi trenger også! Vi shipper det meste async eller deferred, men vi burde jo ikke sende det i det hele tatt?

Utviklere er ikke vanlige folk

Norge, kiaempers fødeland

Fritt etter Johan Nordahl Brun

Så hva med Norge? Vi har utfordringer vi også, som vi må ta hensyn til



~1 mill eldre

SSB 1. januar 2024

Eldrebølgen er over oss

315 182 flyktninger i Norge

SSB 1. januar 2024

Ukraina 79 624 alene (2025)

580 000 med lav inntekt

SSB 1. Januar 2024

I 2023. Under 285 000 kroner i årsinntekt (etter skatt). Dette er folk som er våre brukere i ulik grad.

Oppstartstid for on-demand video
~1.5s i Norge
~6s i Thailand

ca. 4.5MB lasting. Forskjellen er 200ms latency på alle nettverkskall. NRK har brukere i Thailand, og det betyr mest sannsynlig at dere også har brukere i Thailand.

Parsing av JSON-responser

1 ms på MacBook Pro

200 ms på samsung-tv

Med fem requester er totalen 5-10ms på MacBook, 1s på dårlig device.

**Vi må lage løsninger for de minst
privilegerte brukerne våre**

De privilegerte klarer seg alltid

Erkjennelse #1

Kjenn brukerne dine

Vite hvor de bor, hva de sliter med. Så hvordan skal vi gjøre det?

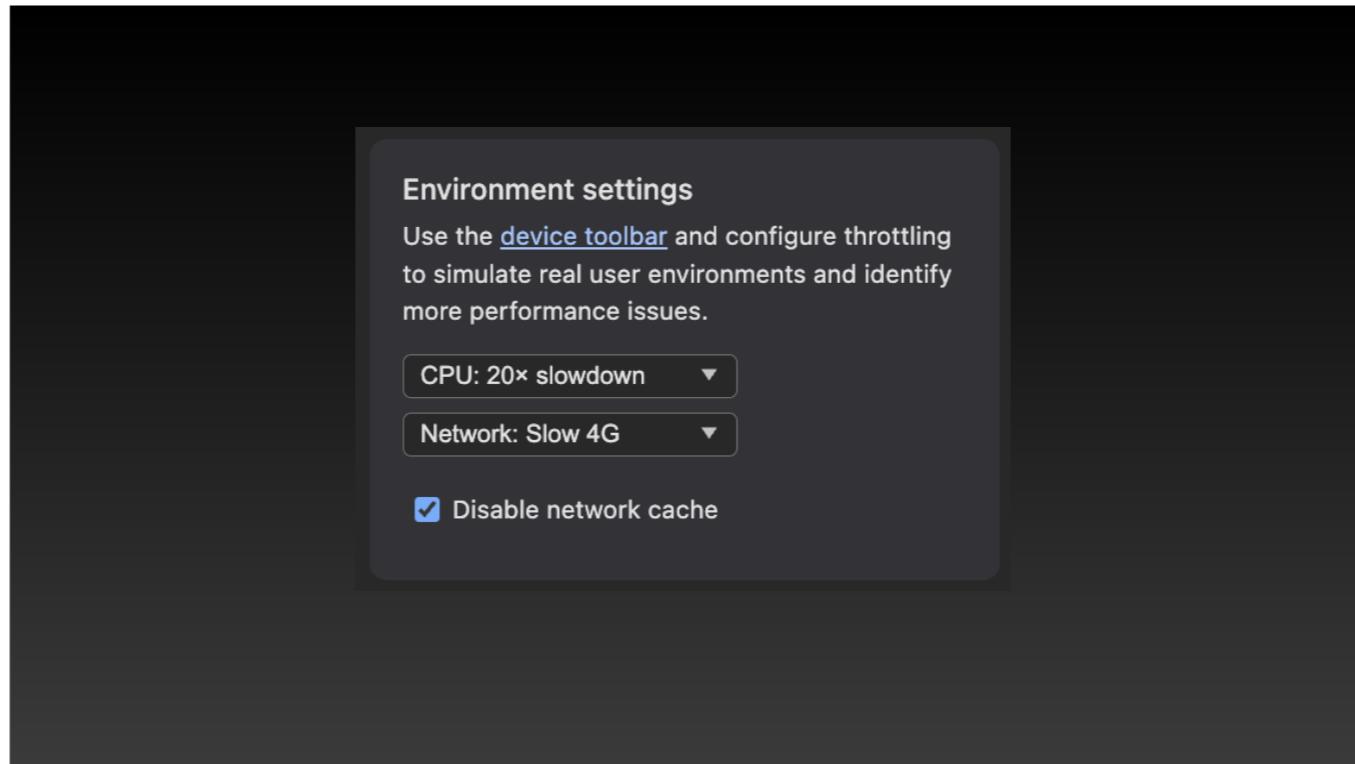
Ytelse er en feature

Og det hjelper på pagerank også. Men det er en feature som produkteiere sjeldent, om noen gang, ber om.

Moores lov er død

<https://cap.csail.mit.edu/death-moores-law-what-it-means-and-what-might-fill-gap-going-forward>

Den har egentlig ikke vært relevant siden 2016 omtrent. Nå fortiden er forbedringene ofte i form av cache, ikke ren cpu-ytelse. Og mobiler har mye mindre cpu-cache enn desktop-maskiner.



Vi må throttle ned utvikler-laptopen! Chrome can også enkelt lage profiler på devices med forskjellige karakteristikker og kalibrere cpu-throttle til den device

**Every little bit of uncaring makes
the world a little bit worse**

Nikhil Suresh

Vi må bry oss om dette, det skjer ikke av seg selv. Rammeverkenes default er å ikke gi deg god performance. Det blir mye javascript, mye som skjer på click og render, mye kode som lastes, css-i-js osv.

"I hold this truth to be self-evident: the larger the abstraction layer a web developer uses on top of web standards, the shorter the shelf life of their codebase becomes, and the more they will feel the churn."

Joeri Sebrechts, <https://plainvanillaweb.com/blog/articles/2024-09-30-lived-experience/>

Og de store rammeverkene koster oss tid i arbeid med å gjete dependencies



DX

Jeg tror vi bryr oss for mye om dette, og lar oss forlede inn i fristelsen når rammeverk lokker med god developer experience. Ikke det at ikke en god utvikler-opplevelse ikke er viktig, men er det noensinne viktigere enn brukerens behov?



Mange gode fotskytere i de store rammeverkene. You're doing it wrong-artikler på kode24. De er så komplekse! Ofte lager de flere problemer enn de løser (de løser alltid DX da). Mange har kjøpt argumentet om at DOM er tregt, men det er ikke det. Hvordan løser du layout trashing dersom det er rammeverket ditt som endrer css før den spør om størrelsen på elementet? Og den lager unødvendig stor DOM (som igjen koster tid i layout)

Erkjennelse #2

**Jobben er å lage gode
brukeropplevelser**

Og for å få til det, må vi bry oss om ytelse (11min)

Hvordan hentes og lastes en nettside?

Vi skal gå ganske dypt. Browseren er vår runtime, og de er dessverre ikke standardiserte (joda, neida)

Tight mode / 2 step waterfall

Innført av google for en tid tilbake, med veeldig lite fanfare rundt det. I hovedsak for å jobbe seg rundt servere som gjør http 2 og 3 feil og sender ting i tilfeldig rekkefølge.

Type/prioritet	Highest	High	Medium	Low	Lowest
HTML					
CSS (head)					
JS (head)					
JS (async)					
JS (defer)					
JS (body)					
Image (body)					
Image (5 første)					

Type/prioritet	Highest	High	Medium	Low	Lowest
HTML					
CSS (head)					
JS (head)					
JS (async)					
JS (defer)					
JS (body)					
Image (body)					
Image (5 første)					

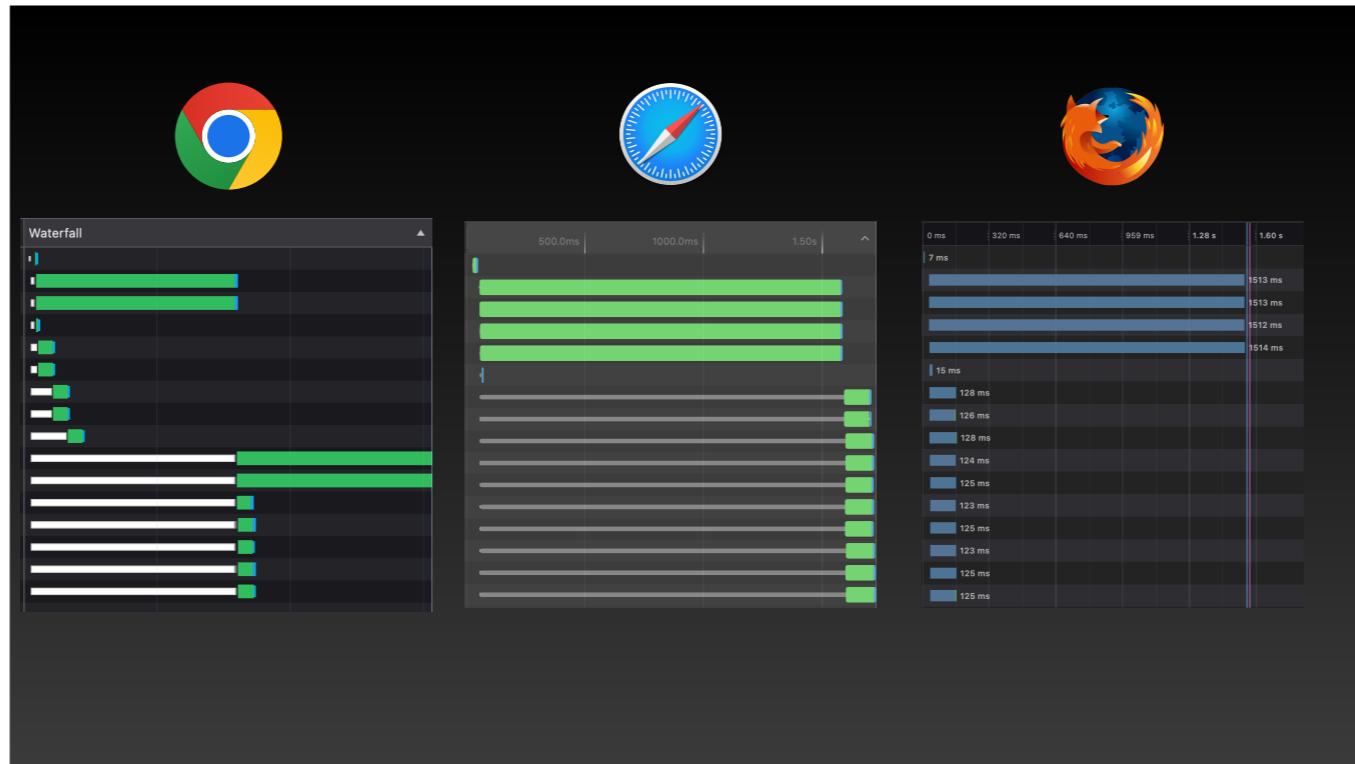
Tight mode Senere

Men, fra chrome 117 (september 2023) inkluderte google de fem først bildene i tight mode. Men bare 2 av gangen. Men de bare medium? Hva er greia. Logikken sier at de blir inkludert i tight mode likevel.

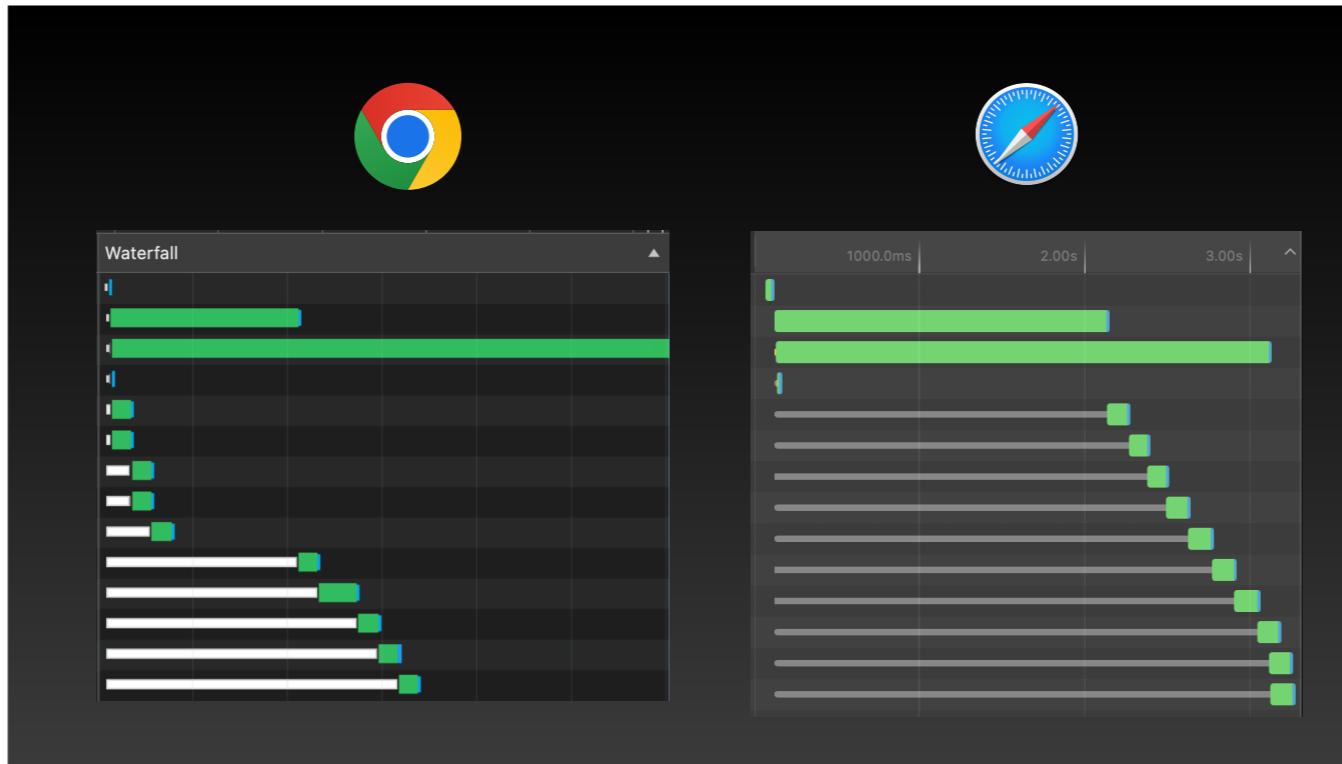
```
<html>
  <head>
    <script src="script.js?delay=1500&c=1"></script>
    <script src="script.js?delay=1500&c=2"></script>

    <script src="script.js?delay=1500&c=3" defer></script>
    <script src="script.js?delay=1500&c=4" defer></script>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <main>
      <h1>Delayed loading only</h1>
      
      
      
      
      
      
      
      
      
      
    </main>
  </body>
</html>
```

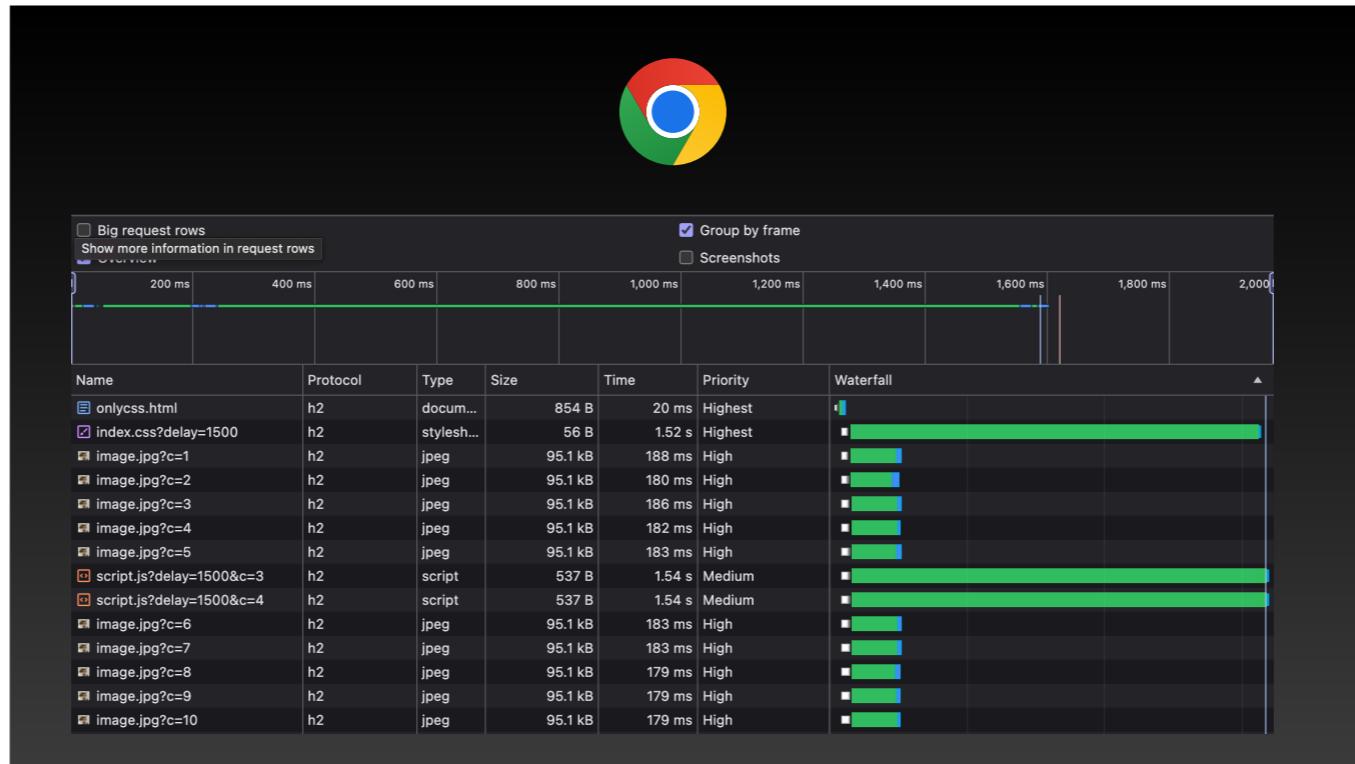
Hvordan kommer denne waterfallen til å se ut?



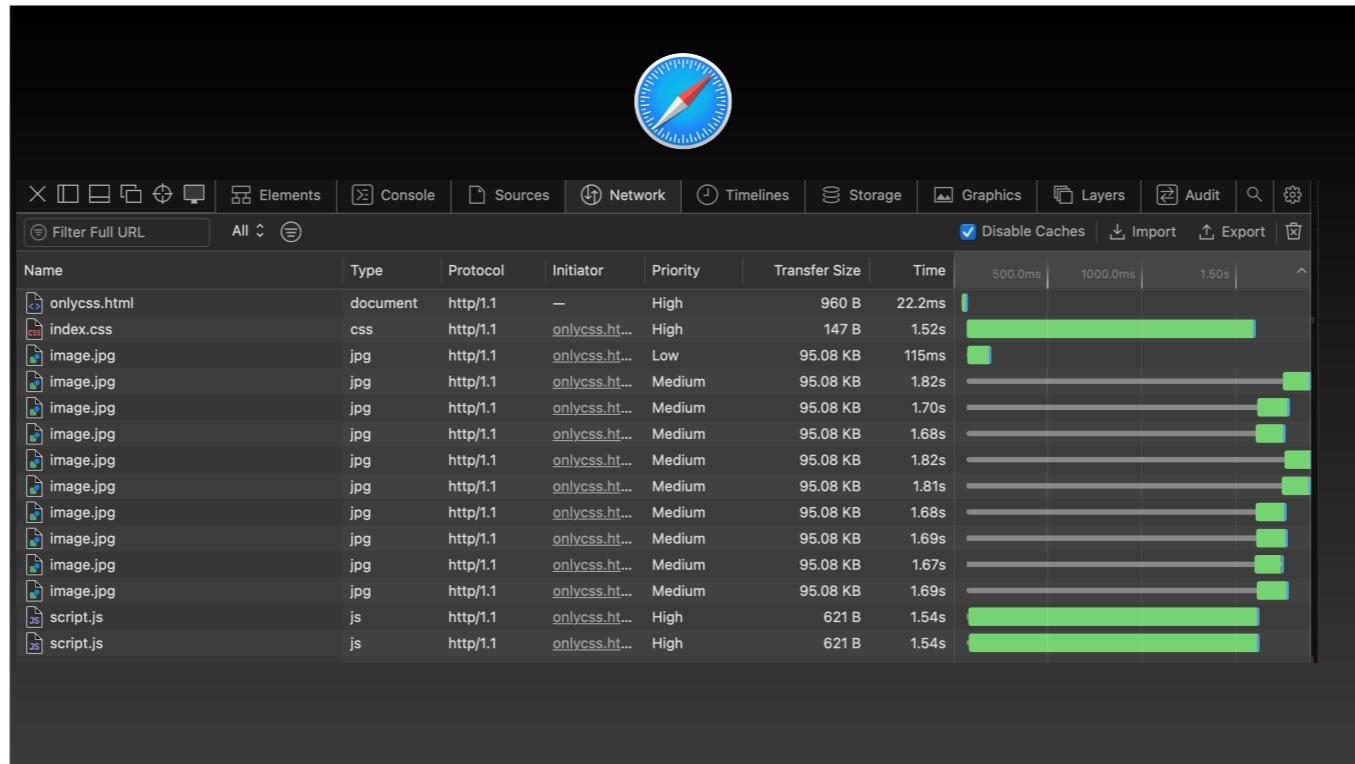
Chrome tar max to high pri request og 5 medium etterpå og kan ta to medium samtidig med høyere pri ressurser. Safari tar alt i head først (også deferred), så resten. Firefox tar alt på en gang. Og de er selvøflgelig ikke enige med hverandre!



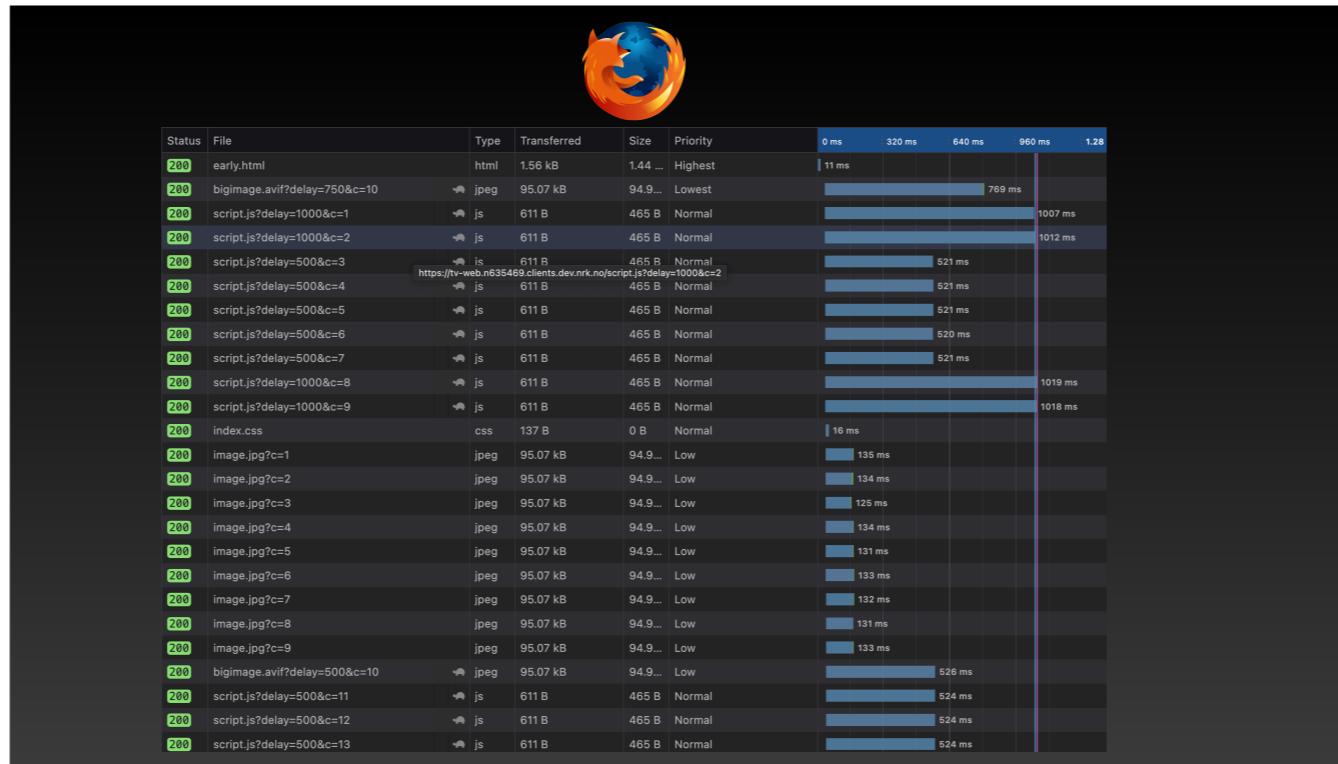
Dersom et script henger og tar mye mere tid enn de andre, lastes bare en ressurs med lavere prioritet av gangen, fordi vi har en “in-flight request”.



HTML uten blocking javascript i head. Det gjelder bare for blocking javascript, ikke css.



Men Safari ser på css også, og gjøre tight mode for dem også. Og Safari blokker uansett hvor javascriptet er, ikke bare i head!



Firefox gir ikke styre med dette og stoler helt på server-implementasjonene av http2 og 3, og requester alt sammen med en gang den finner det. Yolo liksom, og det kan faktisk være raskere enn de andre.

fetchpriority

Fetchpriority-attributt på de fleste typer ressurser i html, som vil øke eller senke prioritet. Man setter den ikke absolutt, bare nudger den i den retningen du ønsker. Du kan få ting inn i tight mode, i hvert fall i chrome og for bilder i safari. Nyttig om du ikke får flyttet det i html'en. Fetchpriority er nå i alle browsere (firefox siden sent 2024)



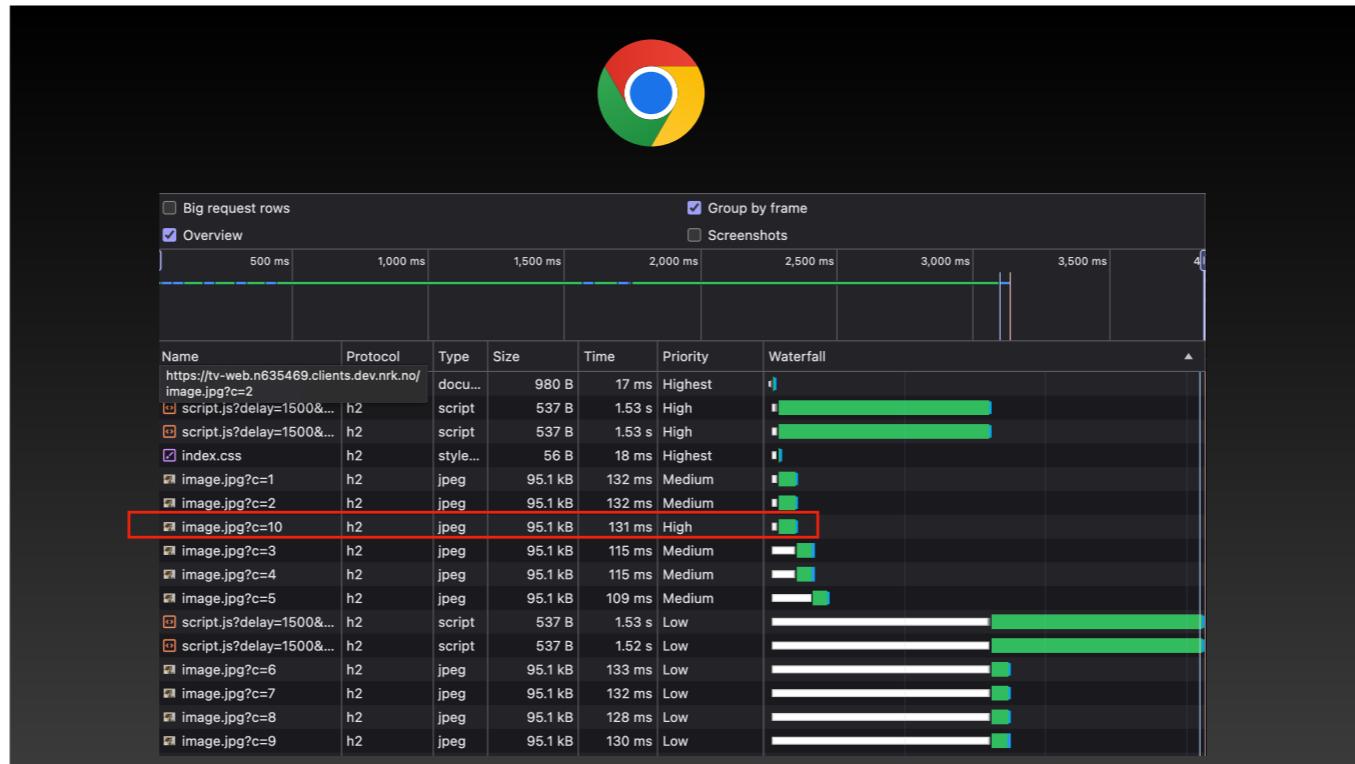
fetchpriority=high

Få bilder (for safari eller chrome) eller andre ressurser (chrome) inn i tight mode.



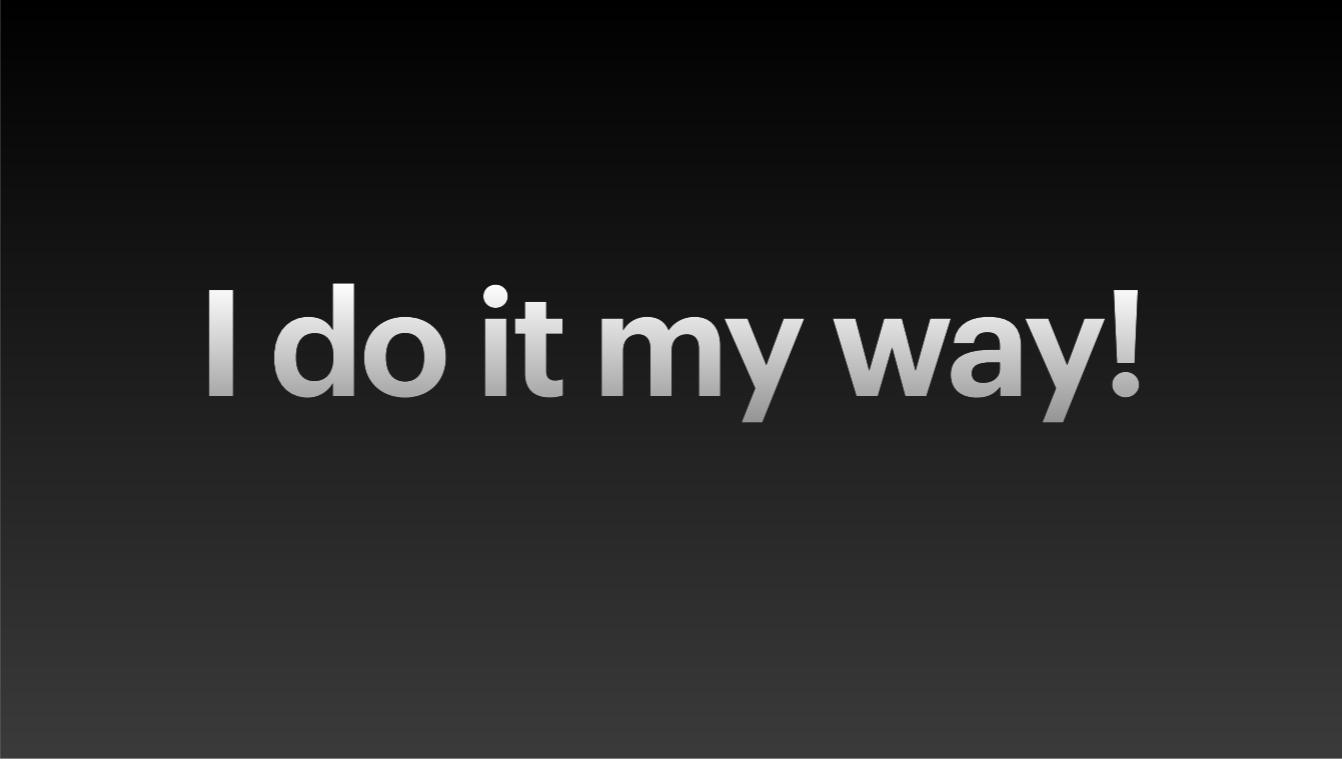
fetchpriority=low

Senke prioriteten for å vente med å laste en ting.



```
fetch('https://example.com/', {priority: 'low'})  
  .then(data => {  
    ...  
  }) ;
```

Gjelder også for fetch-kall vi selv lager, for det er det samme api'et i bunn.



I do it my way!

Ingen browsere gjør det helt likt.

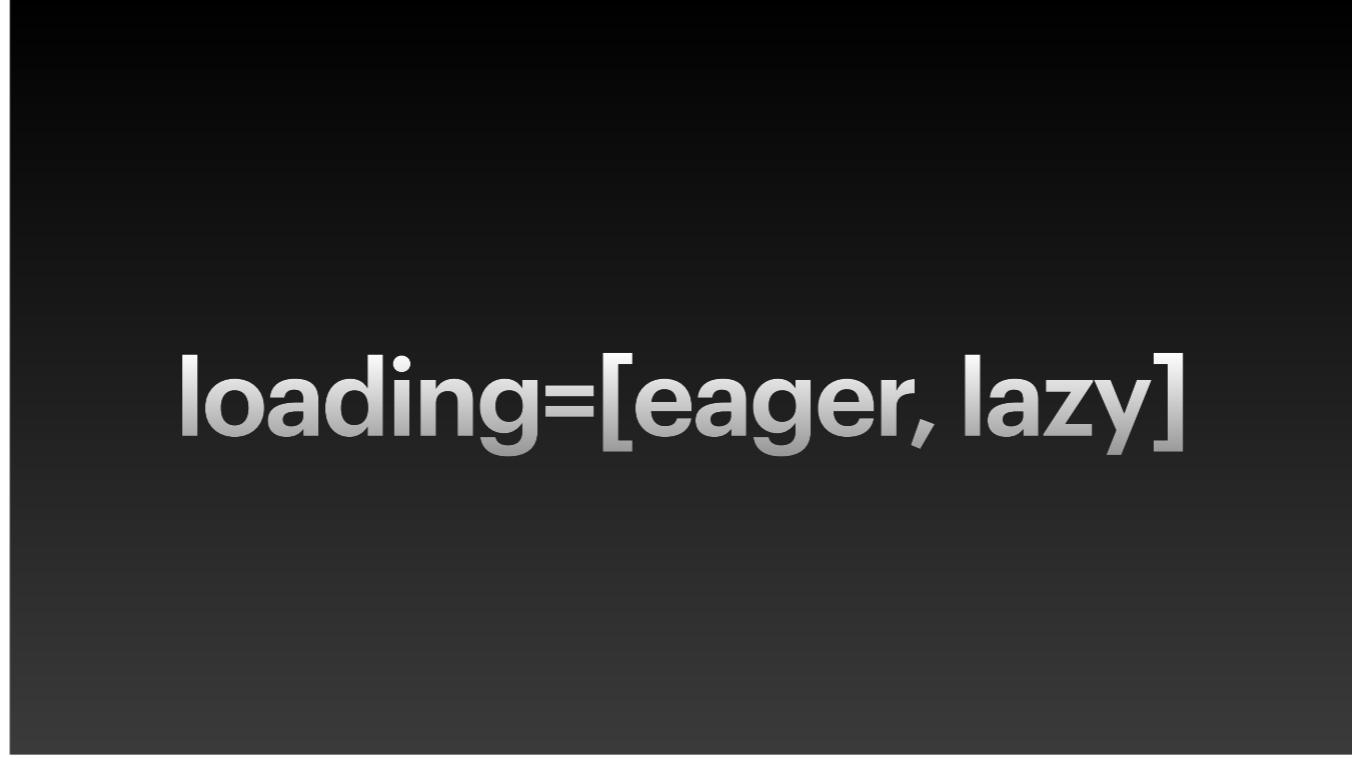
Eksempel: fonter

Type/prioritet	Highest	High	Medium	Low	Lowest
Font (@font-face)					
Font preload		  			

Fonter!

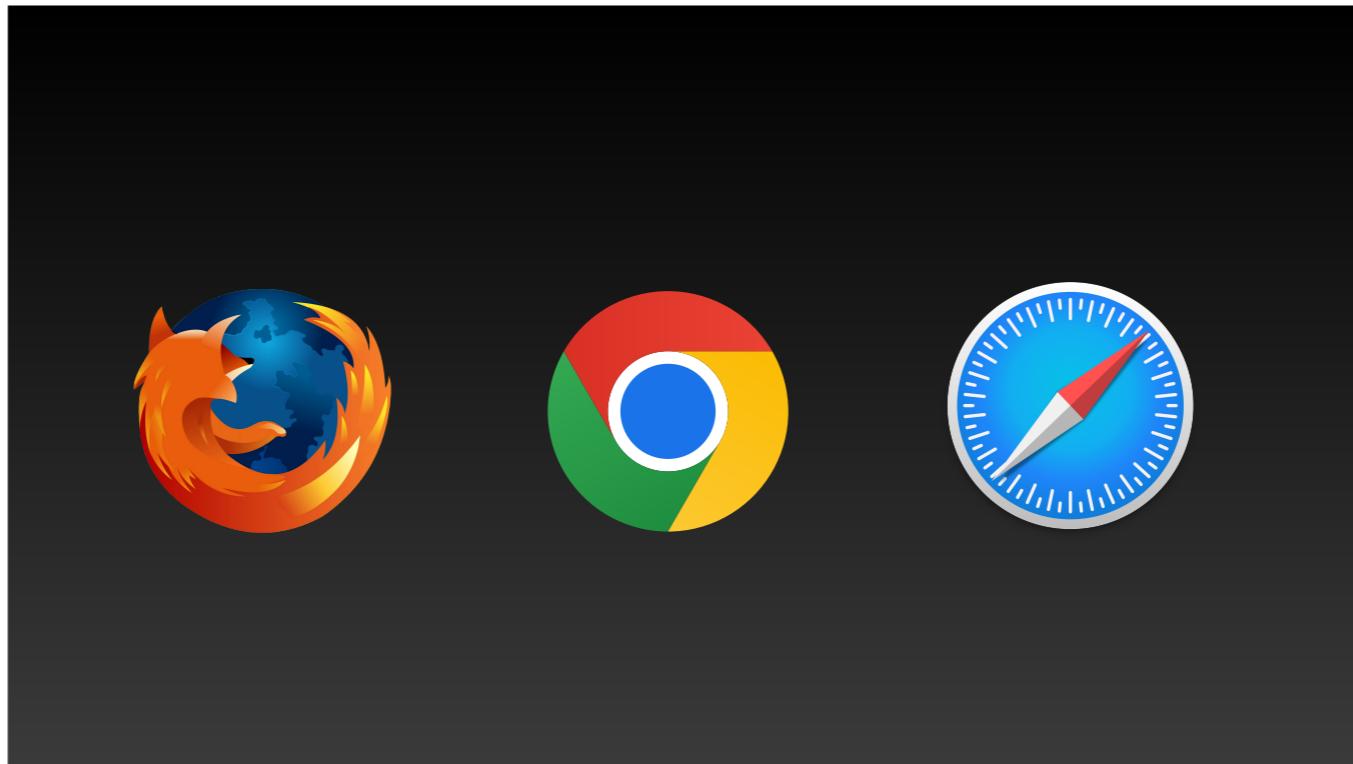
Type/prioritet	Highest	High	Medium	Low	Lowest
Font (@font-face)					
Font preload					
Font preload fetchpriority=high					
Font preload fetchpriority=low					

Fonter!



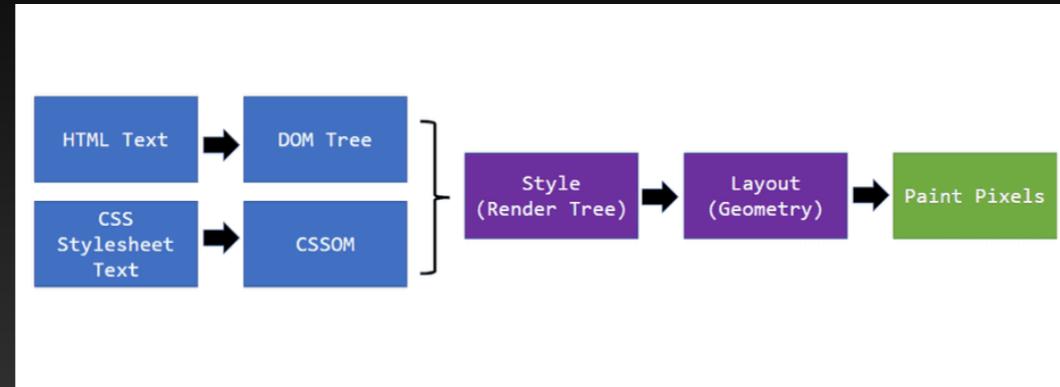
loading=[eager, lazy]

Loading-attributtet på bilder er på ingen måte det samme! loading=eager er default og gjør ingenting med prioriteten. Lazy vil derimot forsinke henting til bildet er i viewporten. Loading styrer når bildet hentes, fetchpriority styrer hvordan



Neste umulig å lage en nettside som performer likt i alle browsere, i dagens økosystem.

Rendering pipeline



Denne kommer inn i bildet etter at ressurser er lastet, men den skal vi ikke snakke om her i dag fordi den er veldig mye bedre dokumentert enn tight mode. Men om du ikke kjenner, denne bruk url'en.

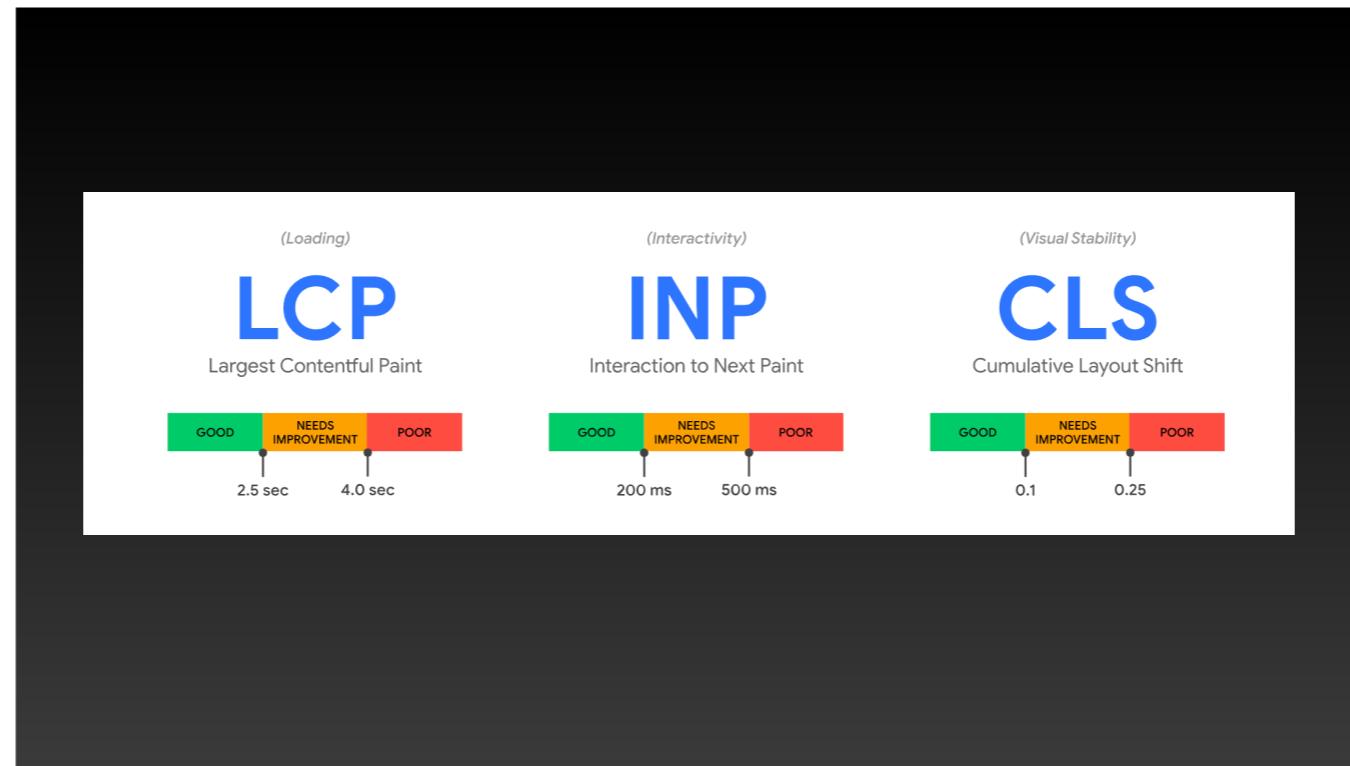


Everything is a trade-off

Så hvordan skal vi finne ut av dette? (23min)

Core Web Vitals

Tracking-bibliotek og initiativ fra Google for å måle relevante data hos brukerne, Baserer seg på ulike performance-api'er under panseret.



fokuserer på INP (interaktivitet), LCP (lasting) og CLS (visuell stabilitet).

Grensene er for lave!

The screenshot shows the web.dev Core Web Vitals tool within the Chrome DevTools Performance panel. The main area displays three key metrics:

- Largest Contentful Paint (LCP):** 1.13 s (good)
- Cumulative Layout Shift (CLS):** 0.38 (poor)
- Interaction to Next Paint (INP):** 48 ms (good)

Below each metric is a detailed breakdown of the elements contributing to the score. The LCP section includes a performance histogram:

Performance Range	Count
Good	1
Needs Improvement	1
Poor	1

Time markers: 2.5 sec and 4.0 sec.

The INP section also provides a breakdown of interactions.

On the right, there's a "Next steps" sidebar with environment settings (CPU: 6x slowdown, Network: Fast 4G, Disable network cache), recording options (Record, Record and reload), and a "Learn more" link.

Enkleste approach er med performance tab i chrome devtools. Og bryter ned tallene delvis og gir info om interaksjoner og layout-skifter. Gir egentlig informasjon om _det meste_ og er de samme målingene som biblioteket gir oss

≡  web.dev

Sign in

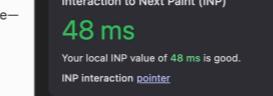
Core Web Vitals

Core Web Vitals are the subset of Web Vitals that apply to all web pages, should be measured by all site owners, and will be surfaced across all Google tools. Each of the Core Web Vitals represents a distinct facet of the user experience, is measurable [in the field](#), and reflects the real-world experience of a critical [user-centric](#) outcome.

The metrics that make up Core Web Vitals will [evolve](#) over time. The current set focuses on three aspects of the user experience—*loading*, *interactivity*, and *visual stability*—and includes the following metrics (and their respective thresholds):

(Loading) **LCP**
Largest Contentful Paint

Your local LCP value of **1.13 s** is good.
LCP element [p](#)

(Interactivity) **INP**
Interaction to Next Paint

Your local INP value of **48 ms** is good.
INP interaction pointer

Local metrics

Largest Contentful Paint (LCP) **1.13 s**
Your local LCP value of **1.13 s** is good.
LCP element [p](#)

Cumulative Layout Shift (CLS) **0.38**
Your local CLS value of **0.38** is poor.
Worst cluster [3 shifts](#)

Learn more about local and field data

Interactions Layout shifts

Layout shift score	Element
0.0002	devsite-search
0.3402	div.devsite-article-body.clearfix
0.0001	devsite-search devsite-appearance-selector devsite-user#devsite-user
0.0365	div.devsite-actions devsite-toc.devsite-nav.devsite-toc-embedded div.devsite-article-body.clearfix

Next steps

Environment settings

Use the [device toolbar](#) and configure throttling to simulate real user environments and identify more performance issues.

CPU: 6x slowdown Network: Fast 4G

Show custom tracks [Learn more](#)

Disable network cache

Record [E](#)

Record and reload [E](#)



Lab versus RUM

<https://web.dev/articles/lab-and-field-data-differences>

Lab eller syntetiske data er bra for å kontrollere flest mulig variabler og få like forhold hver gang. Dette er nyttig når man forsøker å debugge og optimalisere. RUM-data er fra dine faktiske brukere, der de er, i daler og på holmer overalt i Norge. Så hva sier RUM oss som lab-data ikke forteller?

Forskjellige LCP-elementer

Viewport-størrelse, personalisering, fonter etc kan alle påvirke hva som er brukerens faktiske LCP-element.

Caching

Caching kan påvirke alle målingene i det fri. Også bfcache i nettleseren, som vi skal se på litt senere.

INP krever ekte brukerinteraksjon

Hva er det brukerne faktisk gjør?

What element was interacted with

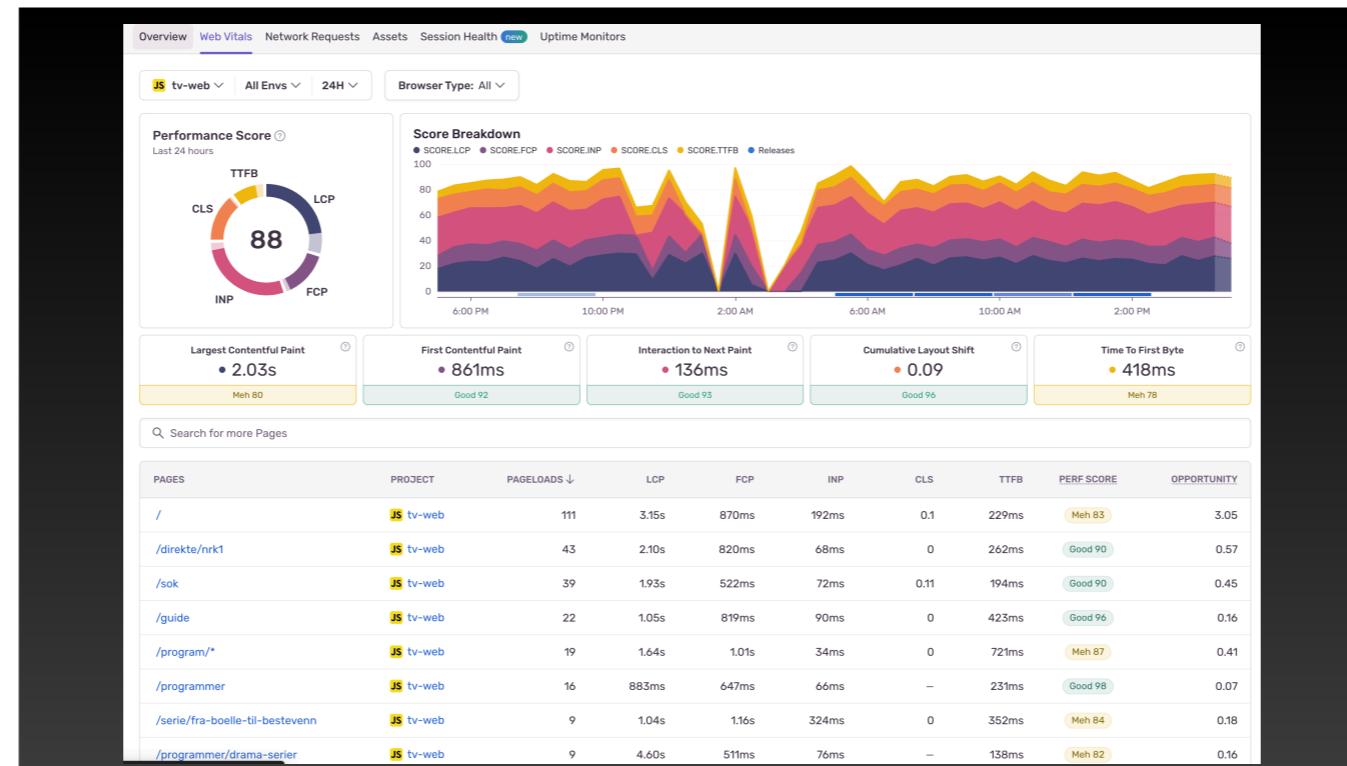
Why type of interaction it was

When that interaction took place

En ofte synder for høy INP er cookie-bannere fra 3. parter. Pass på dem!

Prioriter RUM-data

Jeg vil anbefale å måle dem selv, faktisk (verktøy finnes, som sentry). Du kan ikke forbedre det du ikke kan måle.



Sentry-dashboard for [tv.nrk.no](#)

Web-vitals
+
Litt-kode
+
Influxdb
==
RUM-dashboard





Dypere i Web Vitals

Vi skal se litt på attribution dataene i web-vitals.vitals.html
Performance script timing (script)
Server-Timing (siden 2023)

The screenshot shows the Chrome DevTools Performance tab. At the top, there are several configuration options: Disable JavaScript samples (unchecked), Enable advanced paint instrumentation (slow) (unchecked), Enable CSS selector stats (slow) (unchecked), CPU: No throttling, Network: Fast 4G, and Show custom tracks (checked). Below this, the "Local metrics" section displays:

- Largest Contentful Paint (LCP): 1.15 s (good)
- Cumulative Layout Shift (CLS): 0 (good)
- Interaction to Next Paint (INP): 496 ms (needs improvement)

The "INP interaction" section shows a pointer interaction with a button element. The timeline details the following phases and their local duration:

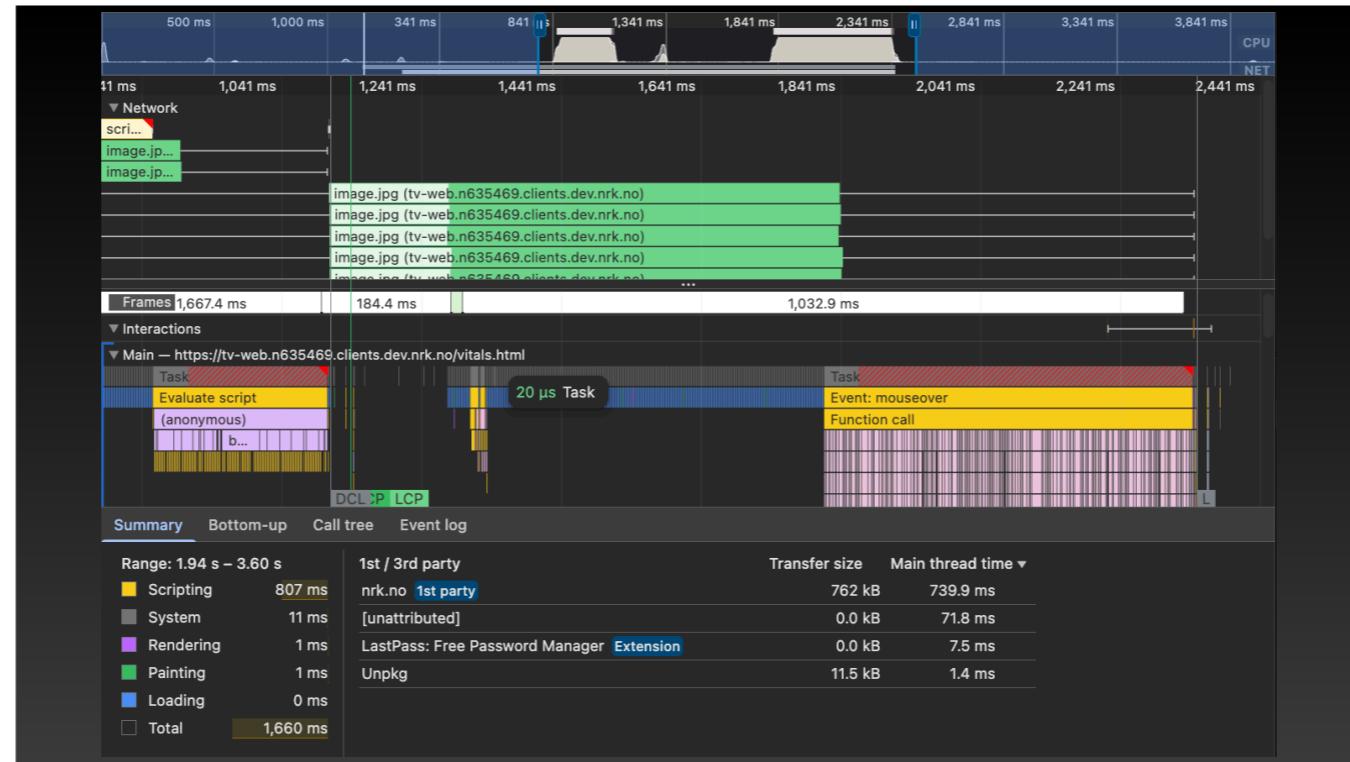
Phase	Local duration (ms)
Input delay	469
Processing duration	1
Presentation delay	26

Below this, a keyboard interaction with the same button element has a local duration of 32 ms.

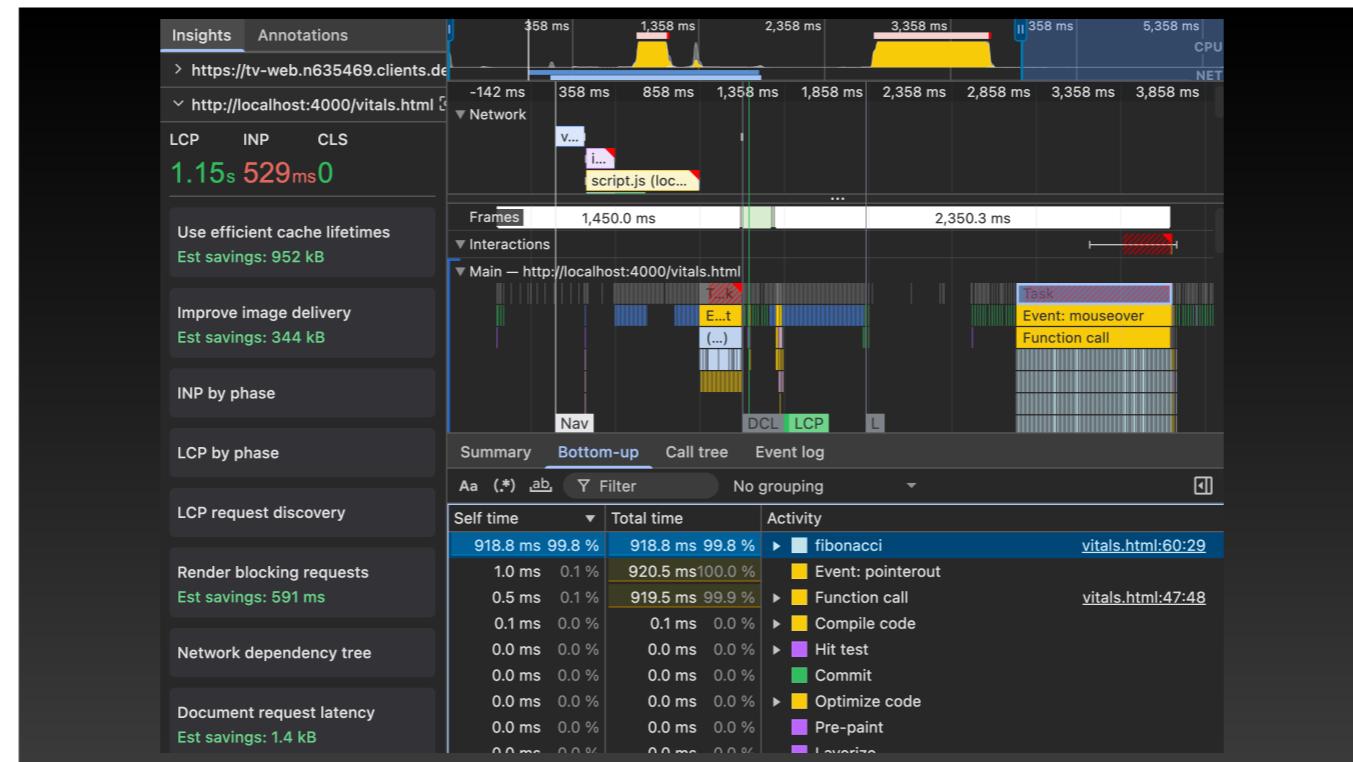
The "Next steps" section includes:

- Field data:** See how your local metrics compare to real user data in the [Chrome UX Report](#). (Set up button)
- Environment settings:** Use the [device toolbar](#) and configure throttling to simulate real user environments and identify more performance issues. (CPU: No throttling, Network: Fast 4G, Disable network cache checkboxes)
- Record:** (⌘ E)
- Record and reload:** (⌘ ⌘ E)

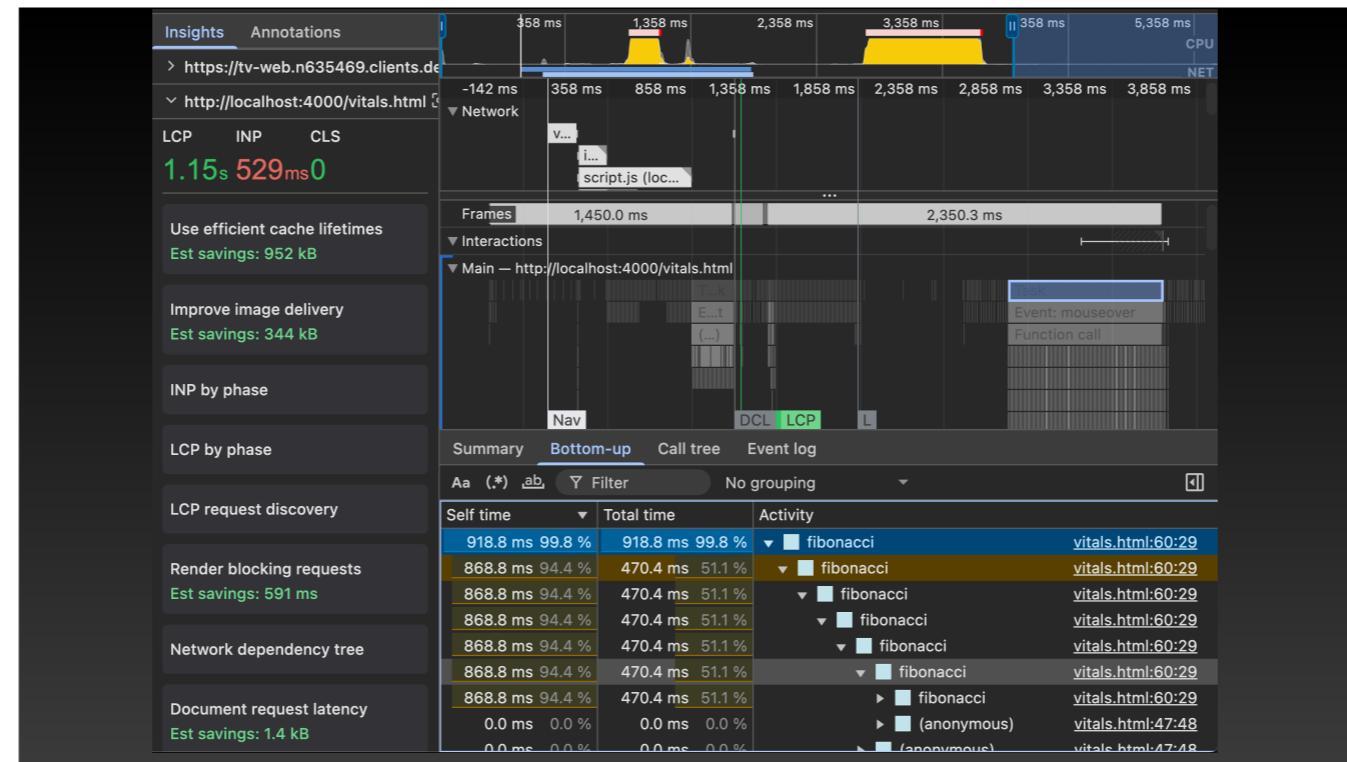
Eksempel fra en lasting av en test-side med en dårlig INP-score. La oss se om vi kan finne ut hva det skyldes.



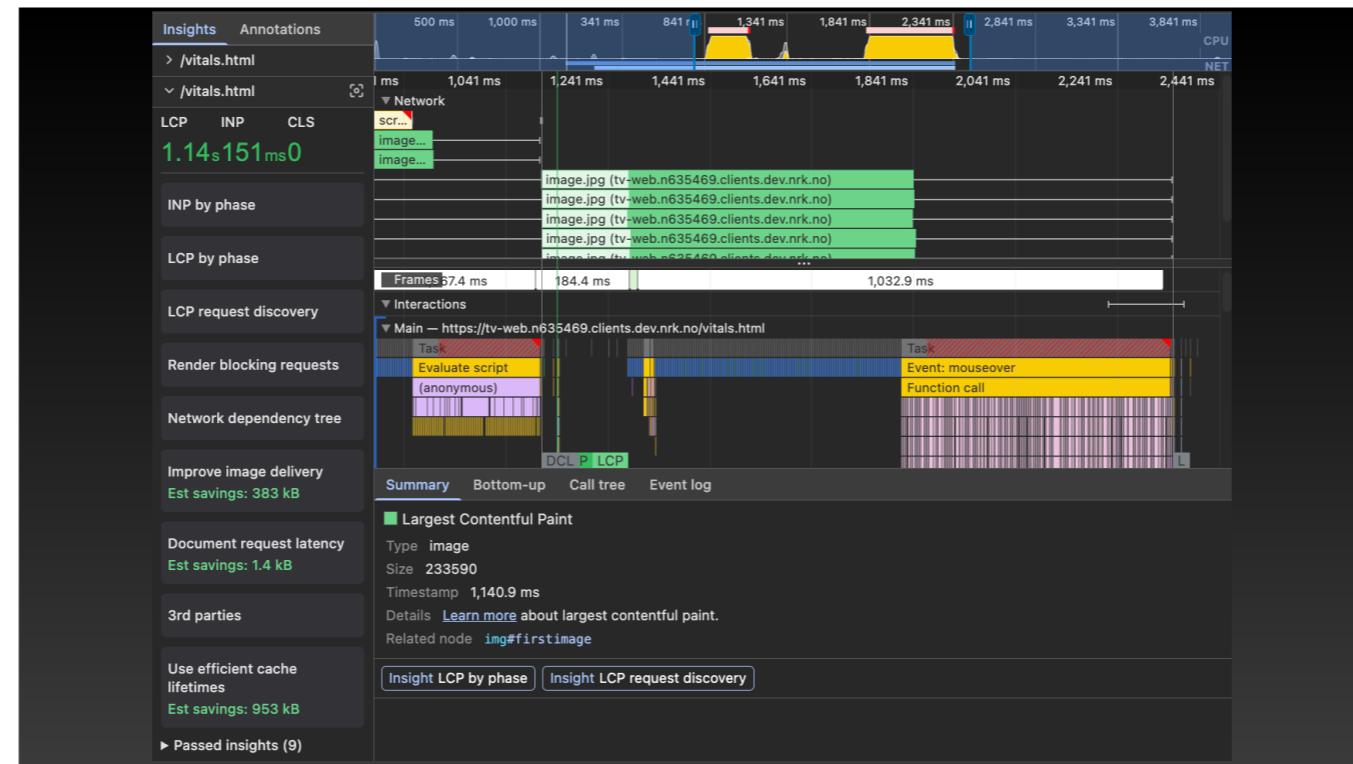
Vi har to tasks som er markert med rødt, som long tasks, den ene på oppstart, men den andre er på mouseover!



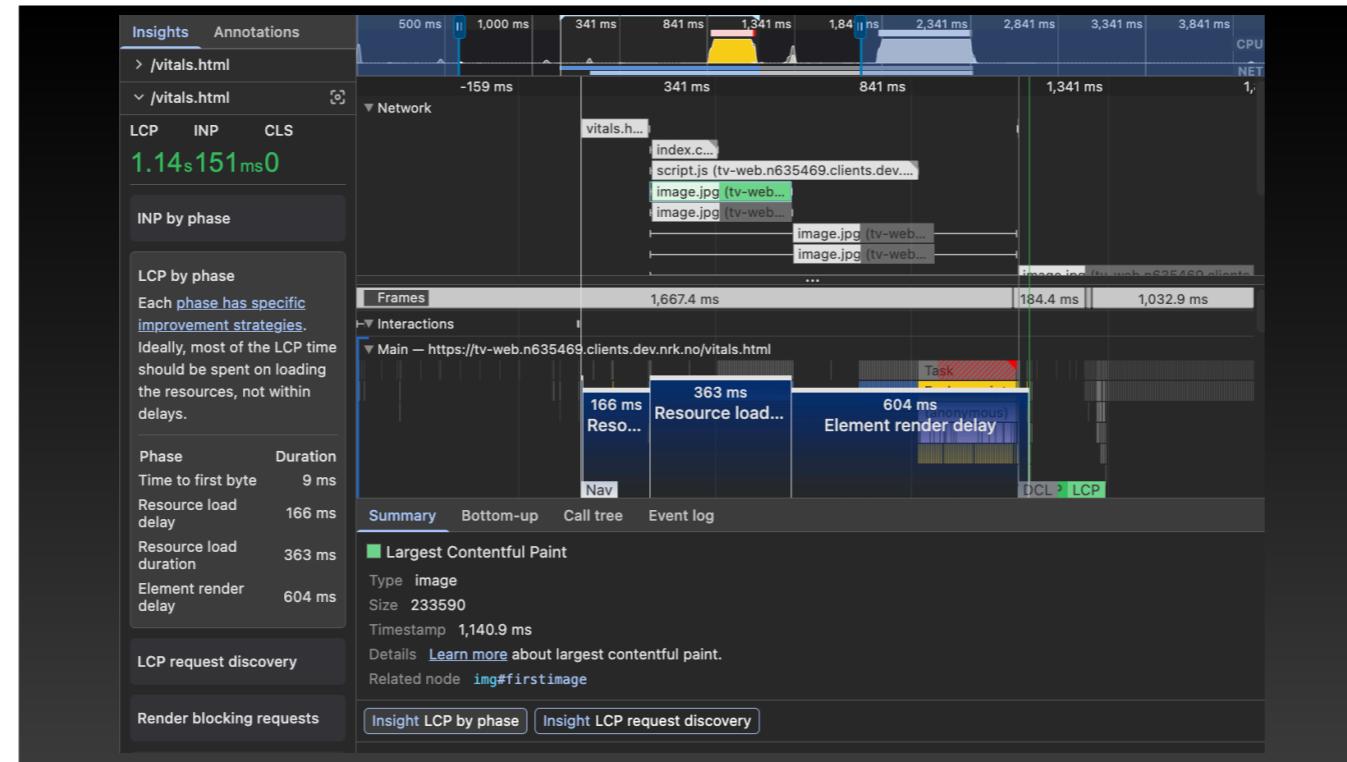
Trykker vi på den ser vi hvilket metodekall som lager den langsomme interaksjonen. Tallene er litt annerledes pga. forskjellige runs.



Viser seg at vi har rekursiv kalkulering av fibonacci-rekken på onmouseover. Ikke den beste tingen å kjøre påmouseover kanskje, men det illustrerer et poeng. Ikke kjør tunge ting på event handlere



Vi kan se på LCP



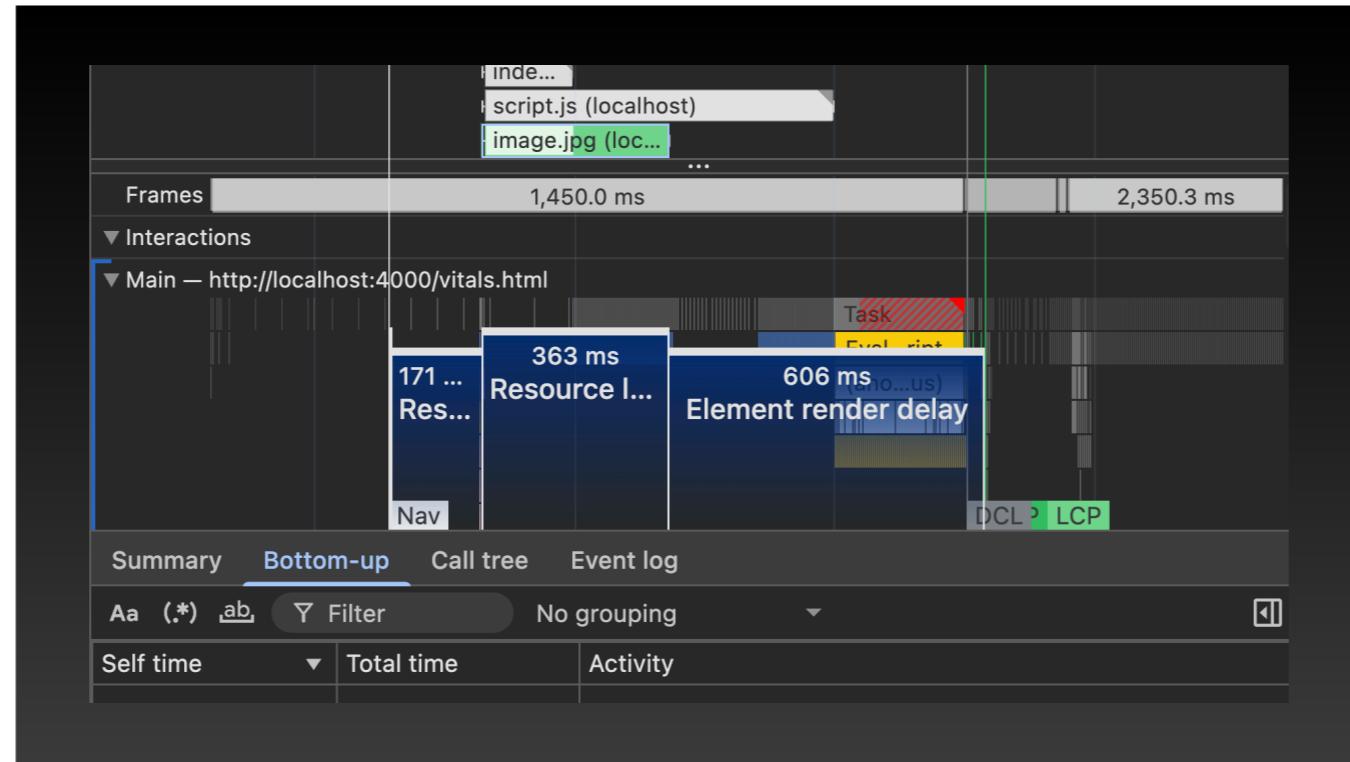
Og på fasene ser vi at vi har noen forsinkelser i ressurs-lasting

```
▼ {name: 'LCP', value: 1144, rating: 'good', delta: 1144, entries: Array(1), ...} ⓘ
  ▼ attribution:
    elementRenderDelay: 606.7000000476837
    ▶ lcpEntry: LargestContentfulPaint {renderTime: 1144, loadTime: 1108.8999999761581, size: 1144, ...}
    ▶ lcpResourceEntry: PerformanceResourceTiming {initiatorType: 'img', nextHopProtocol: 'HTTP/2', ...}
    ▶ navigationEntry: PerformanceNavigationTiming {unloadEventStart: 173.5, unloadEventEnd: 173.5, ...}
    resourceLoadDelay: 171.10000002384186
    resourceLoadDuration: 357.6999999284744
    target: "#firstimage"
    timeToFirstByte: 8.5
    url: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
    ▶ [[Prototype]]: Object
  delta: 1144
  ▶ entries: [LargestContentfulPaint]
  id: "v5-1747316711050-2575798612027"
  name: "LCP"
  navigationType: "reload"
  rating: "good"
  value: 1144
```

I consolet har vi skrevet ut de faktisk web vitals-objektene vi har fått. Her er det mye data

```
▼ {name: 'LCP', value: 1144, rating: 'good', delta: 1144, entries: Array(1), ...} ⓘ
  ▼ attribution:
    elementRenderDelay: 606.7000000476837
    ▼ lcpEntry: LargestContentfulPaint
      duration: 0
      ▶ element: img#firstimage
      entryType: "largest-contentful-paint"
      id: "firstimage"
      loadTime: 1108.8999999761581
      name: ""
      renderTime: 1144
      size: 233590
      startTime: 1144
      url: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
      ▶ [[Prototype]]: LargestContentfulPaint
```

LCP er på første bildet, men det ble lastet etter 1100ms. Hvorfor det? Vi har en renderDelay på 606 ms (som er pausen fra det ble lastet, til det er ferdig rendret).



Her ser vi den grafiske framstillingen. Vi har en long task akkurat idet bildet vårt ble lastet!

```
    ▼ lcpResourceEntry: PerformanceResourceTiming
      connectEnd: 174.60000002384186
      connectStart: 174.60000002384186
      decodedBodySize: 94923
      deliveryType: ""
      domainLookupEnd: 174.60000002384186
      domainLookupStart: 174.60000002384186
      duration: 362.699999284744
      encodedBodySize: 94923
      entryType: "resource"
      fetchStart: 174.60000002384186
      finalResponseHeadersStart: 287.89999997615814
      firstInterimResponseStart: 0
      initiatorType: "img"
      name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
      nextHopProtocol: "h2"
      redirectEnd: 0
      redirectStart: 0
      renderBlockingStatus: "non-blocking"
      requestStart: 179.60000002384186
      responseEnd: 537.2999999523163
      responseStart: 287.89999997615814
      responseStatus: 200
      secureConnectionStart: 174.60000002384186
    ▼ serverTiming: Array(1)
      ▶ 0: PerformanceServerTiming {name: 'image', duration: 100, description: ''}
        length: 1
      ▶ [[Prototype]]: Array(0)
      startTime: 174.60000002384186
      transferSize: 95223
      workerStart: 0
```

```
▼ LcpResourceEntry: PerformanceResourceTiming
  connectEnd: 174.6000002384186
  connectStart: 174.6000002384186
  decodedBodySize: 94923
  deliveryType: ""
  domainLookupEnd: 174.6000002384186
  domainLookupStart: 174.6000002384186
  duration: 362.6999999284744
  encodedBodySize: 94923
  entryType: "resource"
  fetchStart: 174.6000002384186
  finalResponseHeadersStart: 287.89999997615814
  firstInterimResponseStart: 0
  initiatorType: "img"
  name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  renderBlockingStatus: "non-blocking"
  requestStart: 179.6000002384186
  responseEnd: 537.2999999523163
```

```
▼ LcpResourceEntry: PerformanceResourceTiming
  connectEnd: 174.6000002384186
  connectStart: 174.6000002384186
  decodedBodySize: 94923
  deliveryType: ""
  domainLookupEnd: 174.6000002384186
  domainLookupStart: 174.6000002384186
  duration: 362.699999284744
  encodedBodySize: 94923
  entryType: "resource"
  fetchStart: 174.6000002384186
  finalResponseHeadersStart: 287.89999997615814
  firstInterimResponseStart: 0
  initiatorType: "img"
  name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  renderBlockingStatus: "non-blocking"
  requestStart: 179.6000002384186
  responseEnd: 537.2999999523163
```

```
▼ LcpResourceEntry: PerformanceResourceTiming
  connectEnd: 174.6000002384186
  connectStart: 174.6000002384186
  decodedBodySize: 94923
  deliveryType: ""
  domainLookupEnd: 174.6000002384186
  domainLookupStart: 174.6000002384186
  duration: 362.6999999284744
  encodedBodySize: 94923
  entryType: "resource"
  fetchStart: 174.6000002384186
  finalResponseHeadersStart: 287.89999997615814
  firstInterimResponseStart: 0
  initiatorType: "img"
  name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  renderBlockingStatus: "non-blocking"
  requestStart: 179.6000002384186
  responseEnd: 537.2999999523163
```

```
▼ LcpResourceEntry: PerformanceResourceTiming
  connectEnd: 174.6000002384186
  connectStart: 174.6000002384186
  decodedBodySize: 94923
  deliveryType: ""
  domainLookupEnd: 174.6000002384186
  domainLookupStart: 174.6000002384186
  duration: 362.6999999284744
  encodedBodySize: 94923
  entryType: "resource"
  fetchStart: 174.6000002384186
  finalResponseHeadersStart: 287.89999997615814
  firstInterimResponseStart: 0
  initiatorType: "img"
  name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  renderBlockingStatus: "non-blocking"
  requestStart: 179.6000002384186
  responseEnd: 537.2999999523163
```

```
▼ LcpResourceEntry: PerformanceResourceTiming
  connectEnd: 174.6000002384186
  connectStart: 174.6000002384186
  decodedBodySize: 94923
  deliveryType: ""
  domainLookupEnd: 174.6000002384186
  domainLookupStart: 174.6000002384186
  duration: 362.6999999284744
  encodedBodySize: 94923
  entryType: "resource"
  fetchStart: 174.6000002384186
  finalResponseHeadersStart: 287.89999997615814
  firstInterimResponseStart: 0
  initiatorType: "img"
  name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  renderBlockingStatus: "non-blocking"
  requestStart: 179.6000002384186
  responseEnd: 537.2999999523163
```

```
▼ LcpResourceEntry: PerformanceResourceTiming
  connectEnd: 174.6000002384186
  connectStart: 174.6000002384186
  decodedBodySize: 94923
  deliveryType: ""
  domainLookupEnd: 174.6000002384186
  domainLookupStart: 174.6000002384186
  duration: 362.6999999284744
  encodedBodySize: 94923
  entryType: "resource"
  fetchStart: 174.6000002384186
  finalResponseHeadersStart: 287.89999997615814
  firstInterimResponseStart: 0
  initiatorType: "img"
  name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  renderBlockingStatus: "non-blocking"
  requestStart: 179.6000002384186
  responseEnd: 537.2999999523163
```

```
▼ LcpResourceEntry: PerformanceResourceTiming
  connectEnd: 174.6000002384186
  connectStart: 174.6000002384186
  decodedBodySize: 94923
  deliveryType: ""
  domainLookupEnd: 174.6000002384186
  domainLookupStart: 174.6000002384186
  duration: 362.6999999284744
  encodedBodySize: 94923
  entryType: "resource"
  fetchStart: 174.6000002384186
  finalResponseHeadersStart: 287.89999997615814
  firstInterimResponseStart: 0
  initiatorType: "img"
  name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
  nextHopProtocol: "h2"
  redirectEnd: 0
  redirectStart: 0
  renderBlockingStatus: "non-blocking"
  requestStart: 179.6000002384186
  responseEnd: 537.2999999523163
```

```
entryType: "resource"
fetchStart: 174.6000002384186
finalResponseHeadersStart: 287.89999997615814
firstInterimResponseStart: 0
initiatorType: "img"
name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
nextHopProtocol: "h2"
redirectEnd: 0
redirectStart: 0
renderBlockingStatus: "non-blocking"
requestStart: 179.6000002384186
responseEnd: 537.2999999523163
responseStart: 287.89999997615814
responseStatus: 200
secureConnectionStart: 174.6000002384186
serverTiming: Array(1)
  ▶ 0: PerformanceServerTiming {name: 'image', duration: 100, description: null, startTime: 174.6000002384186, transferSize: 85322}
    length: 1
    [[Prototype]]: Array(0)
```

Duration tok 362 ms, og server-timing sier at det tok 100ms på serveren (header). Men hvor kan vi se den long-tasken i web-vitals dataene? Hold på den tanken...

Og dette er data som kan være tilgjengelig som RUM, ikke bare i lab.

```
entryType: "resource"
fetchStart: 174.6000002384186
finalResponseHeadersStart: 287.89999997615814
firstInterimResponseStart: 0
initiatorType: "img"
name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
nextHopProtocol: "h2"
redirectEnd: 0
redirectStart: 0
renderBlockingStatus: "non-blocking"
requestStart: 179.6000002384186
responseEnd: 537.2999999523163
responseStart: 287.89999997615814
responseStatus: 200
secureConnectionStart: 174.6000002384186
serverTiming: Array(1)
  ▶ 0: PerformanceServerTiming {name: 'image', duration: 100, description: null, startTime: 174.6000002384186, transferSize: 65322}
    length: 1
    [[Prototype]]: Array(0)
```

```
entryType: "resource"
fetchStart: 174.6000002384186
finalResponseHeadersStart: 287.89999997615814
firstInterimResponseStart: 0
initiatorType: "img"
name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
nextHopProtocol: "h2"
redirectEnd: 0
redirectStart: 0
renderBlockingStatus: "non-blocking"
requestStart: 179.6000002384186
responseEnd: 537.2999999523163
responseStart: 287.89999997615814
responseStatus: 200
secureConnectionStart: 174.6000002384186
serverTiming: Array(1)
  ▶ 0: PerformanceServerTiming {name: 'image', duration: 100, description: null, startTime: 174.6000002384186, ...}
    length: 1
  ▶ [[Prototype]]: Array(0)
startTime: 174.6000002384186
transferSize: 65322
```

```
entryType: "resource"
fetchStart: 174.6000002384186
finalResponseHeadersStart: 287.89999997615814
firstInterimResponseStart: 0
initiatorType: "img"
name: "https://tv-web.n635469.clients.dev.nrk.no/image.jpg?c=1"
nextHopProtocol: "h2"
redirectEnd: 0
redirectStart: 0
renderBlockingStatus: "non-blocking"
requestStart: 179.6000002384186
responseEnd: 537.2999999523163
responseStart: 287.89999997615814
responseStatus: 200
secureConnectionStart: 174.6000002384186
serverTiming: Array(1)
  ▶ 0: PerformanceServerTiming {name: 'image', duration: 100, description: null, startTime: 174.6000002384186, transferSize: 65322}
```

**What looks bad in the lab
looks terrible in your hand**

Alex Russel, performance.now() 2024

Erkjennelse #3

Mål ekte brukerdata

Hva med mobil?

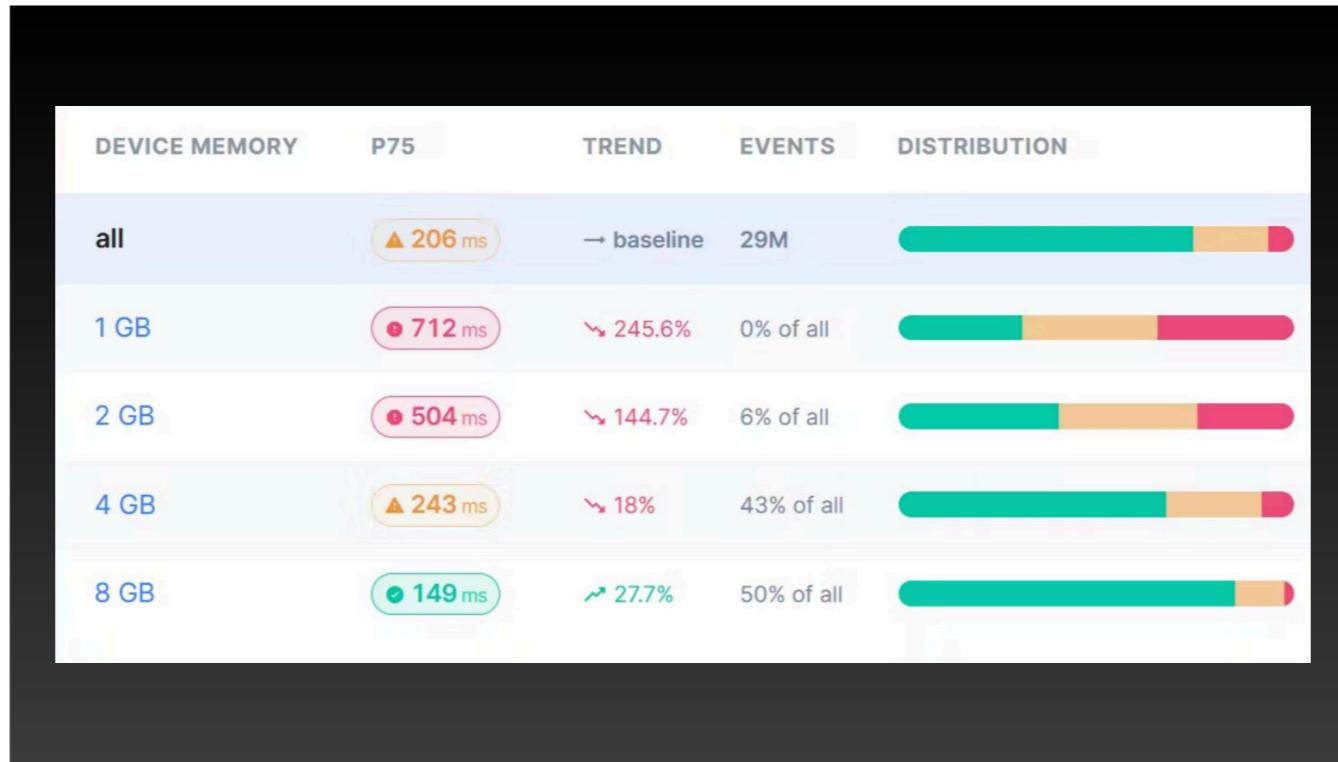
Safari og chrome for android støtter ikke core web vitals og har dårlig verktøy-støtte for å debugge ytelse, men man _kan_ generalisere noe ut av en chrome for desktop som er throttlet ned og med dårlig nett.

lav ram og lav cpu-cache gir høy inp.

Høy INP har mest korrelasjon med devicen man bruker

javascript påvirker INP mest. (Rumvision)

Og det er ikke 5g i alle deler av Norge heller. Det er fortsatt solide hull i dekningskartene til Telenor og Telia.



Fra rumvision på performance.now i 2024 en graf som viser korrelasjon mellom device ram og INP verdier. Lav ram har stor påvirkning! Man kan måle dette selv via navigator.deviceMemory, om man vil.

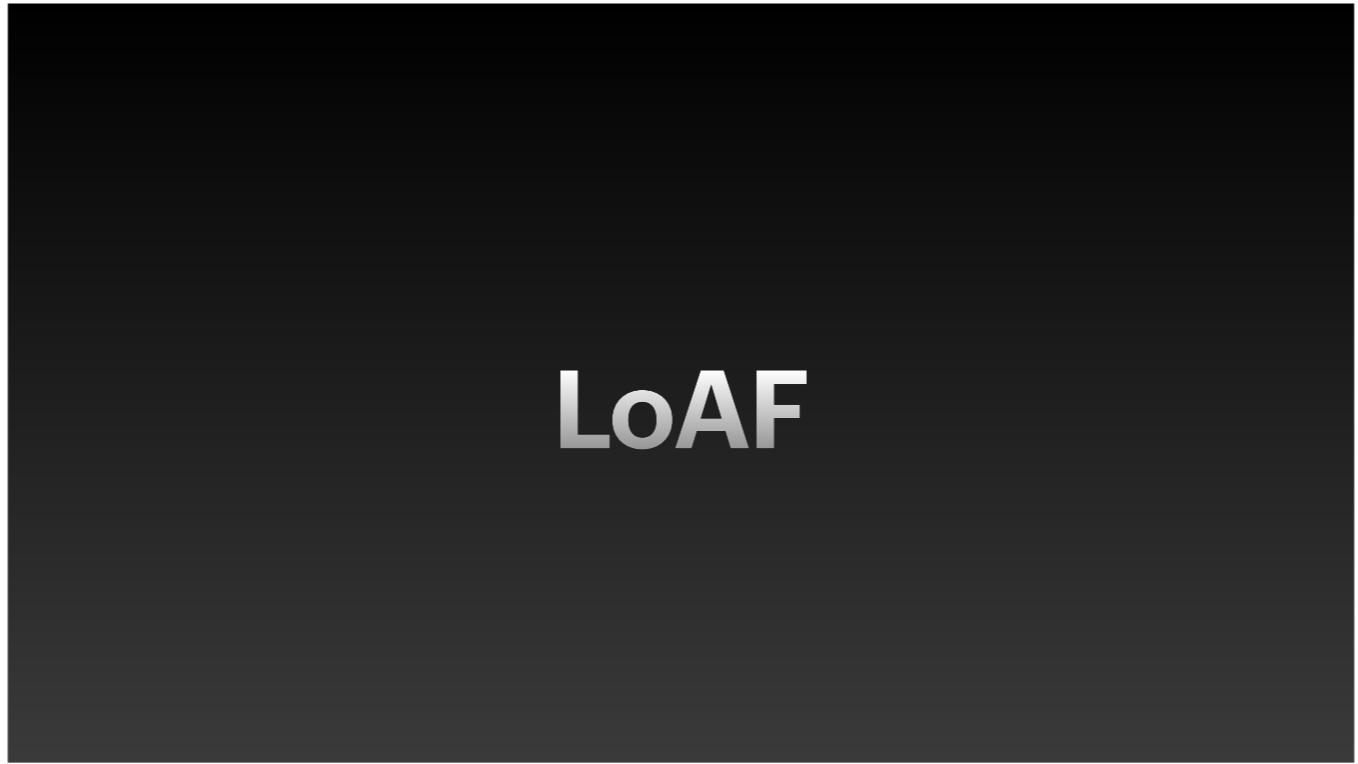
Jo lavere spec på device, jo mer betyr detaljene! På dårlige smart-tv'er har box-shadow (versus border) og easing-funksjoner på animasjoner noe å si. Og lazy-loading av data som virket som en god ide, er ikke lenger en god ide når man scroller raskt og har dårlig ytelse.

bfcache

Cache-api som tar snapshot av hele sider i minnet og kan vise forrige/neste helt uten å gjøre requests, finnes i alle browdere nå (ble innført i chrome 96, 2022).

```
window.addEventListener("pageshow", (event) => {
  if (event.persisted) {
    console.log("Restored from bfcache.");
  } else {
    console.log("Normal load from server");
  }
});
```

Man kan sjekke det enkelt (og logge det som rum-data!)



LoAF

Long animation frame API, innført i chrome 123 i mars 2024 (og finnes bare i edge og chrome foreløpig). I siste versjon av web vitals er long-animation-frame data inkludert i INP attributions.

```
const observer = new PerformanceObserver((list) =>
{
  for (const entry of list.getEntries()) {
    console.log(entry);
  }
}) ;

observer.observe({ type: "long-animation-frame",
buffered: true });
```

Identifiserer langkjørende tasks som kan blokkere interaktivitet. En long frame er over 50ms

Demo LoAF

Loaf-eksempelet.

Se på

loaf i devtools, tidssammenligningene

Tasks, detaljer, blocking time etc.

Man må bryte opp lange tasks i mindre biter.

```
▼ PerformanceLongAnimationFrameTiming {renderStart: 1105.6000000000001,
  firstUIEventTimestamp: 0, blockingDuration: 616.82, scripts:
    blockingDuration: 616.82
    duration: 932.5
    entryType: "long-animation-frame"
    firstUIEventTimestamp: 0
    name: "long-animation-frame"
    renderStart: 1105.6000000238419
  ▶ scripts: [PerformanceScriptTiming]
  startTime: 173.70000004768372
  styleAndLayoutStart: 1105.6000000238419
  ▶ [[Prototype]]: PerformanceLongAnimationFrameTiming
```

Script-propertyen.

```
▼ PerformanceLongAnimationFrameTiming {renderStart: 1105.6000000000001, firstUIEventTimestamp: 0, blockingDuration: 616.82, scripts: [PerformanceScriptTiming], startTime: 173.70000004768372, styleAndLayoutStart: 1105.6000000238419, [[Prototype]]: PerformanceLongAnimationFrameTiming}
  ▶ blockingDuration: 616.82
  ▶ duration: 932.5
  ▶ entryType: "long-animation-frame"
  ▶ firstUIEventTimestamp: 0
  ▶ name: "long-animation-frame"
  ▶ renderStart: 1105.6000000238419
  ▶ scripts: [PerformanceScriptTiming]
  ▶ startTime: 173.70000004768372
  ▶ styleAndLayoutStart: 1105.6000000238419
  ▶ [[Prototype]]: PerformanceLongAnimationFrameTiming
```

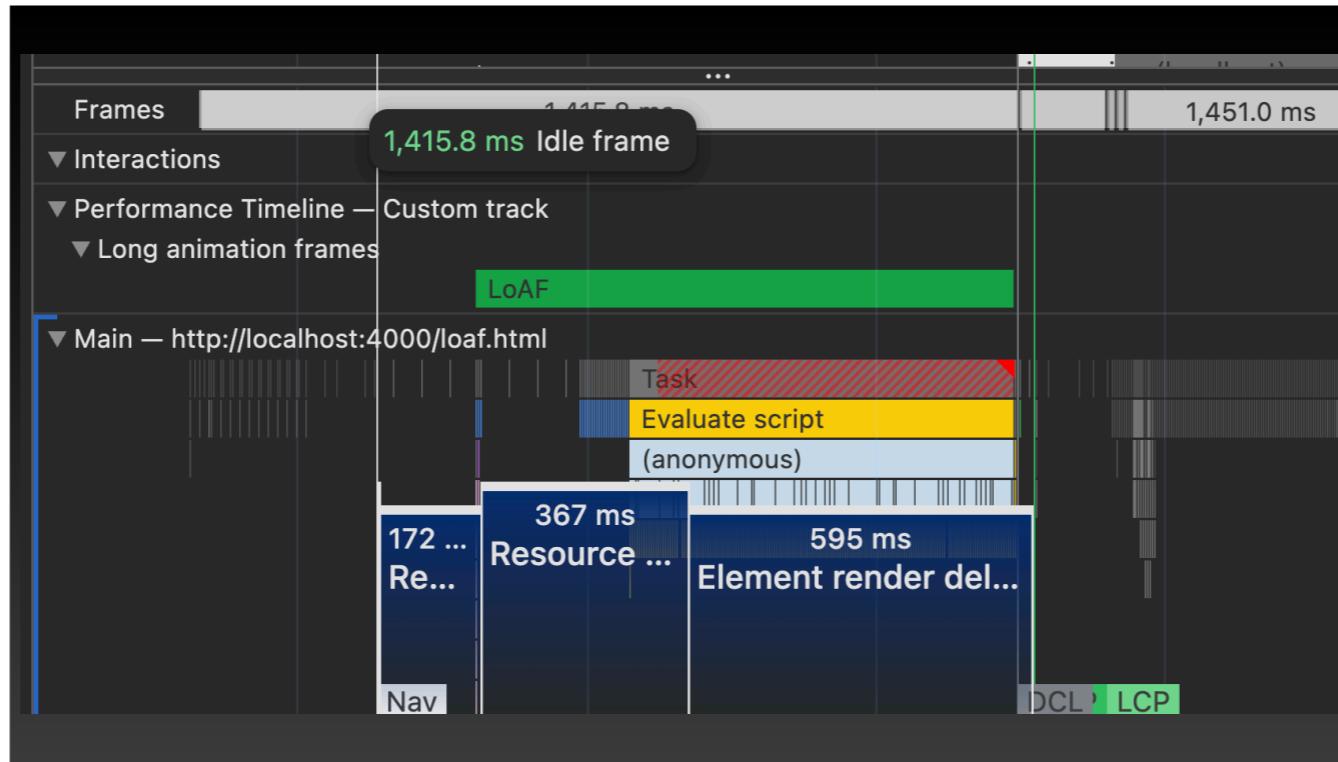
```
▼ PerformanceLongAnimationFrameTiming {renderStart: 1105.6000000000001,
  firstUIEventTimestamp: 0, blockingDuration: 616.82, scripts:
    blockingDuration: 616.82
    duration: 932.5
    entryType: "long-animation-frame"
    firstUIEventTimestamp: 0
    name: "long-animation-frame"
    renderStart: 1105.6000000238419
  ▶ scripts: [PerformanceScriptTiming]
  startTime: 173.70000004768372
  styleAndLayoutStart: 1105.6000000238419
  ▶ [[Prototype]]: PerformanceLongAnimationFrameTiming
```

```
▼ 0: PerformanceScriptTiming
  duration: 666
  entryType: "script"
  executionStart: 438
  forcedStyleAndLayoutDuration: 0
  invoker: "http://localhost:4000/script.js?c=2&delay=250&clientdelay=666"
  invokerType: "classic-script"
  name: "script"
  pauseDuration: 0
  sourceCharPosition: 0
  sourceFunctionName: ""
  sourceURL: "http://localhost:4000/script.js?c=2&delay=250&clientdelay=666"
  startTime: 437.89999997615814
  ▶ window: Window {window: Window, self: Window, document: document, name: '',
    windowAttribution: "self"
  ▶ [[Prototype]]: PerformanceScriptTiming
```

Her ser vi delayen på rundt 600ms fra forrige eksempel. Vi har en long task som forsinke rendering av lcp-bildet

```
▼ 0: PerformanceScriptTiming
  duration: 666
  entryType: "script"
  executionStart: 438
  forcedStyleAndLayoutDuration: 0
  invoker: "http://localhost:4000/script.js?c=2&delay=250&clientdelay=666"
  invokerType: "classic-script"
  name: "script"
  pauseDuration: 0
  sourceCharPosition: 0
  sourceFunctionName: ""
  sourceURL: "http://localhost:4000/script.js?c=2&delay=250&clientdelay=666"
  startTime: 437.89999997615814
  ▶ window: Window {window: Window, self: Window, document: document, name: '',
    windowAttribution: "self"
  ▶ [[Prototype]]: PerformanceScriptTiming
```

```
▼ 0: PerformanceScriptTiming
  duration: 666
  entryType: "script"
  executionStart: 438
  forcedStyleAndLayoutDuration: 0
  invoker: "http://localhost:4000/script.js?c=2&delay=250&clientdelay=666"
  invokerType: "classic-script"
  name: "script"
  pauseDuration: 0
  sourceCharPosition: 0
  sourceFunctionName: ""
  sourceURL: "http://localhost:4000/script.js?c=2&delay=250&clientdelay=666"
  startTime: 437.89999997615814
  ▶ window: Window {window: Window, self: Window, document: document, name: '',
    windowAttribution: "self"
  ▶ [[Prototype]]: PerformanceScriptTiming
```



Grafikken beviser det. De grønne loaf-boksene er forresten custom og her kan man legge inn hva slags data man vil i devtools.

Lange tasks må brytes opp i mindre biter.

Bryt opp lange tasks

Gir bedre inp-verdier og bedre interaktivitet.

scheduler.yield()

Eget API for å a yielde slik at main thread kan gjøre noe annet (chrome/edge 129, ingen andre enn så lenge. Men kom i Firefox nightly heromdagen!). Finnes en polyfill for den.

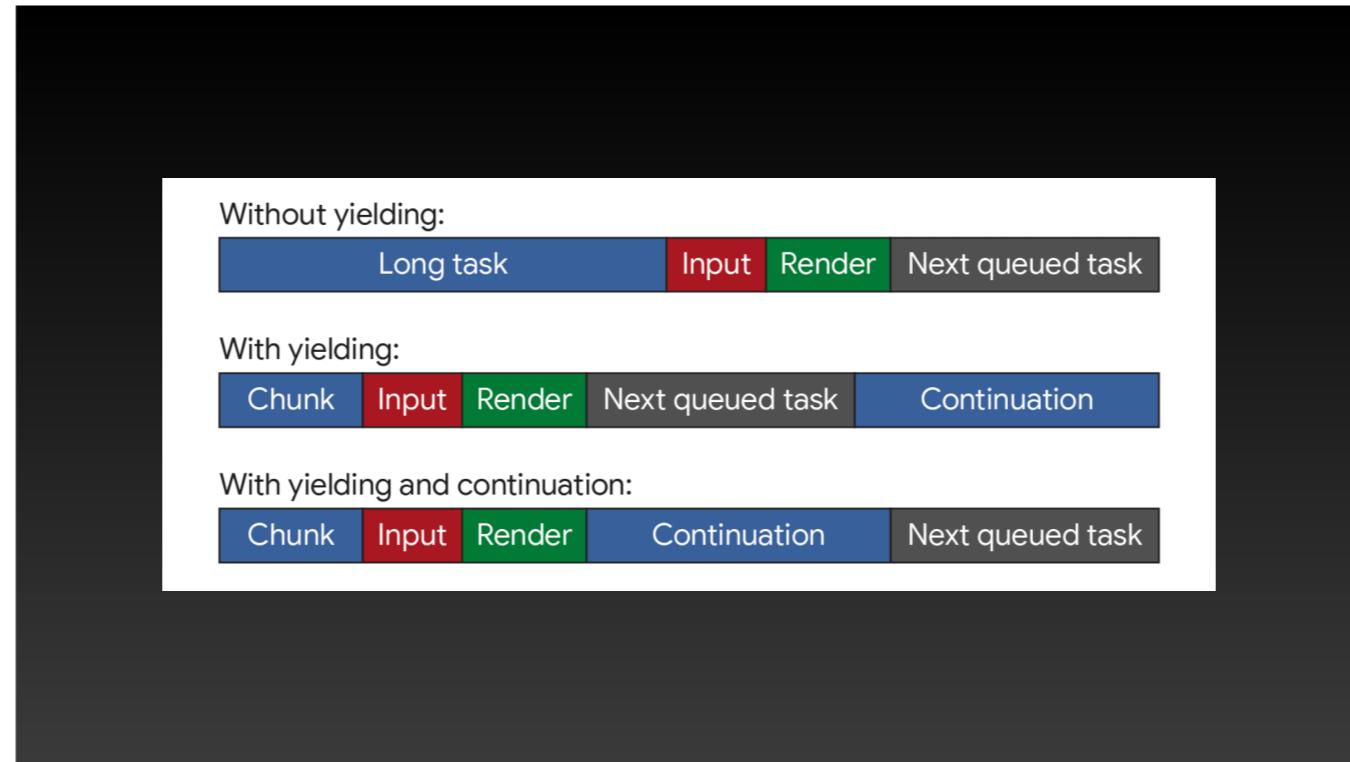
```
function saveSettings() {  
    validateForm();  
    showSpinner();  
    saveToDatabase();  
    updateUI();  
    sendAnalytics();  
}
```

```
function saveSettings() {  
  // Do critical work that is user-visible:  
  validateForm();  
  showSpinner();  
  updateUI();  
  
  // Defer work that isn't user-visible  
  // to a separate task:  
  setTimeout(() => {  
    saveToDatabase();  
    sendAnalytics();  
  }, 0);  
}
```

```
async function saveSettings() {
    // Do critical work that is user-visible:
    validateForm();
    showSpinner();
    updateUI();

    // Yield to the main thread:
    await scheduler.yield();

    // Work that isn't user-visible,
    // continued in a separate task:
    saveToDatabase();
    sendAnalytics();
}
```



Continuation blir prioritert etter waiten, så det vil komme før “ neste ” ting.

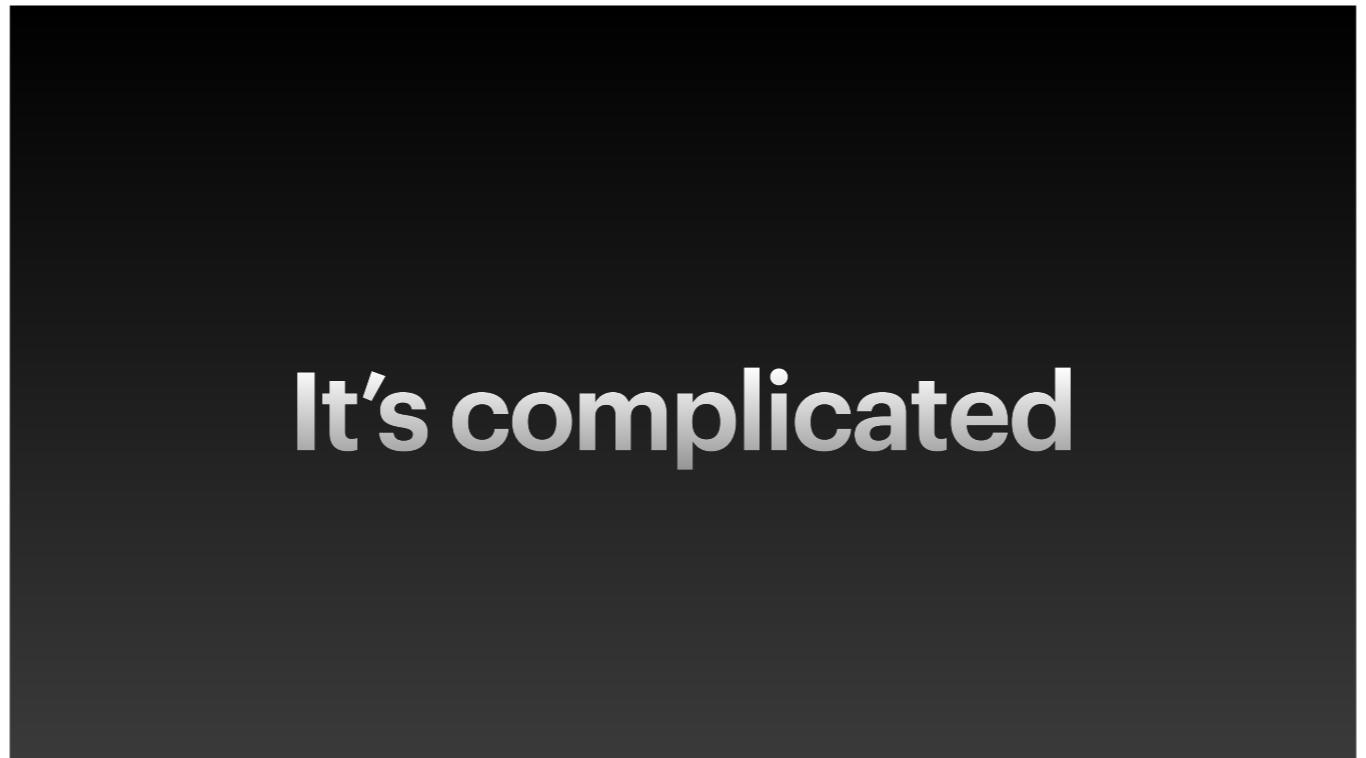
Alt jeg ikke rakk

**Browser Rendering Pipeline
Speculation rules API
HTTP 103 Early hints
Priority-headere i http 2/3
Whiskers i dev-tools**

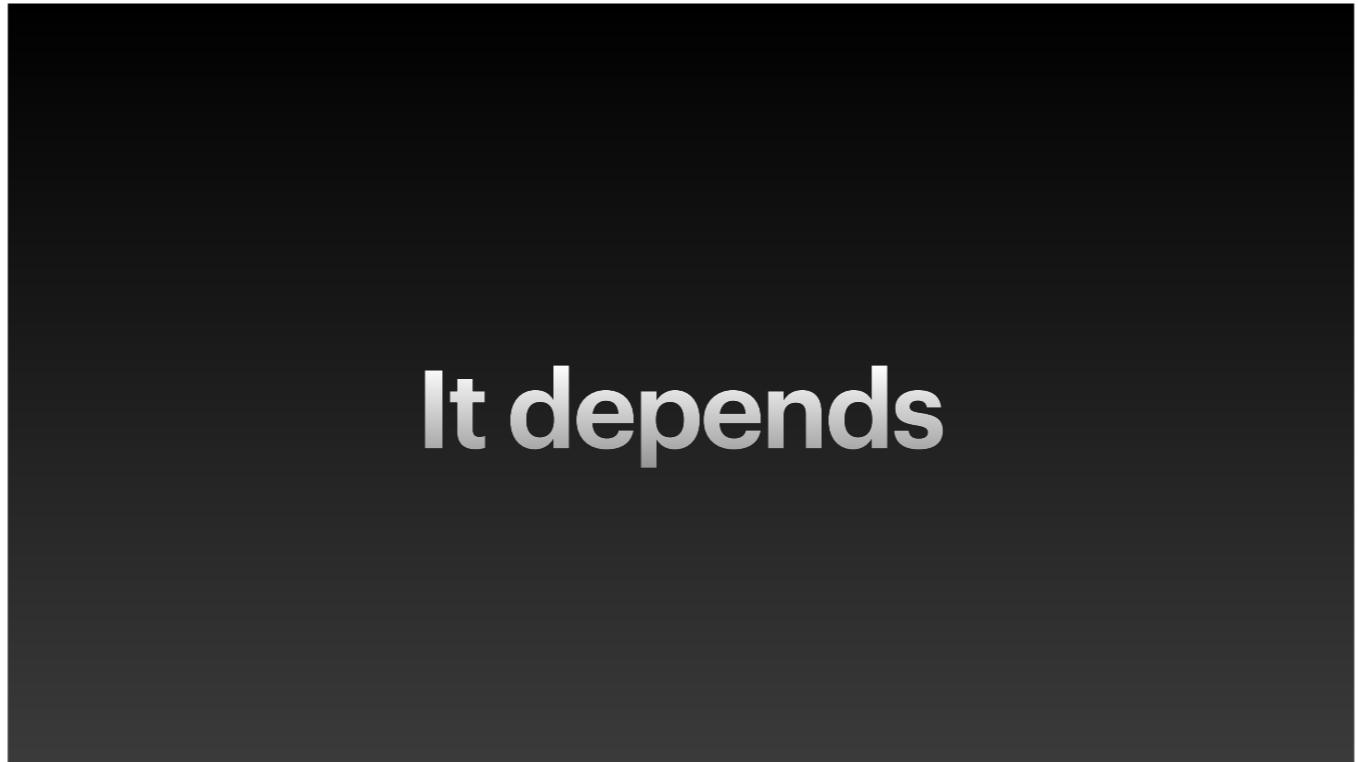
Og sikkert mye annet.

Oppsummert

- **Kjenn brukerne dine**
- **Jobben er å lage gode brukeropplevelser**
- **Mål ekte brukerdata**
- **Kjenn browserene og browser-API'ene**
- **Du må bry deg!**



It's complicated



It depends

Takk for meg!

<https://blog.knuthaugen.no/>

 knuthaug.bsky.social

Slack: Offentlig frontend

Slack: offentlig-paas-no