video 9

# How to optimize Network performance for Web Apps

## Network Optimization / Frontend Performance Optimization Techniques

- lazy loading
- loading js in async
- content visibility
- serving critical css
- Resource hints
- caching using service worker
- caching using CDN
- CSR, SSR
- compression techniques
- layout shifts & reflows

# Load JS Asynchronously

→ async / defer

# Lazy loading

## → Lazy Attribute

// Lazy loading for image, iframe, scripts.
// Probably for images outside of the viewport

```
<img loading="lazy" ..... />
<iframe loading="lazy" ..... />
```

// Prompt an early download of an asset
// for critical image eg hero images.

```
<img loading="eager" /....
<iframe loading="eager" /....
```

// When the browser assign "High" priority to an image
// but we don't actually want that.

```
<img src="less-important-image.svg" fetchpriority="low" />
```

// We want to initiate an early fetch for a resource,
// but also deprioritize it.

```
<link rel="preload" fetchpriority="low"
      href="/script.js" as="script" />
```

## → Intersaction Observer

```
let options = {
    root: null,
    rootMargin: '0px'
    threshold: 0.5
}

function callbackFunction(entries) { // array of
                                     // observing
  element
      entries.forEach(entry => {
        // The logic to apply for each entry.
      })
}

let observer = new IntersectionObserver(callbackFunction,
                                        options);
observer.observe(TARGET-ELEMENT)
```

## → Content-Visibility

I will render only those thing which are in viewport

By just adding `content-visibility: auto`

| baseline | content-visibility: auto |
| --- | --- |
| 232ms Rendering. | 30ms Rendering |

# Serving Critical CSS

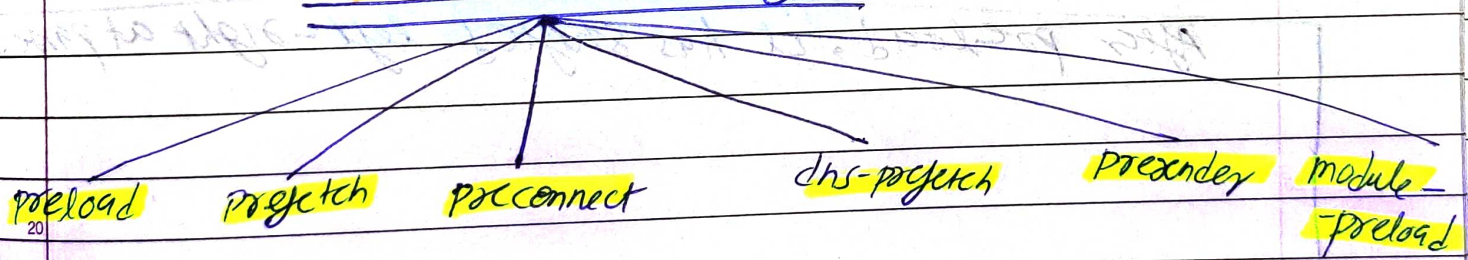
// Load critical CSS asynchronously

```
<link rel="stylesheet"
      href="critical.css" />
```

// Load CSS asynchronously, with low priority

```
<link rel="stylesheet"
      href="full.css"
      media="print"
      onload="this.media='all'" />
```

# Resource Hinting (29:45)



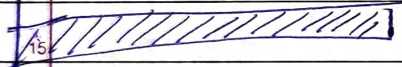
# Preload

In rendering if those, JS & CSS are required then we have to use - preload

eg: <link rel="preload" href="/style.css" as="style" />

Before preload, the network request started here

Performance



After preload, it has shifted left-right at parse time

→ Lift the loading of chunk at start of rendering of webpage itself.

Camlin Page
Date / /

## Prefetch

I want to load but I don't want to execute it.

1. → load
2. → Browser finds prefetch links
3. → Browser fetch requried content to display - prefetch pages
4. → Store content until user clicks link

```
<link rel="prefetch" href="/style.css" as="style" />
```

Some how similar to preload just it will not execute.

# Preconnect

→ It is to handshack whenever it call the api.

→ handshack basically

from source A to destination B

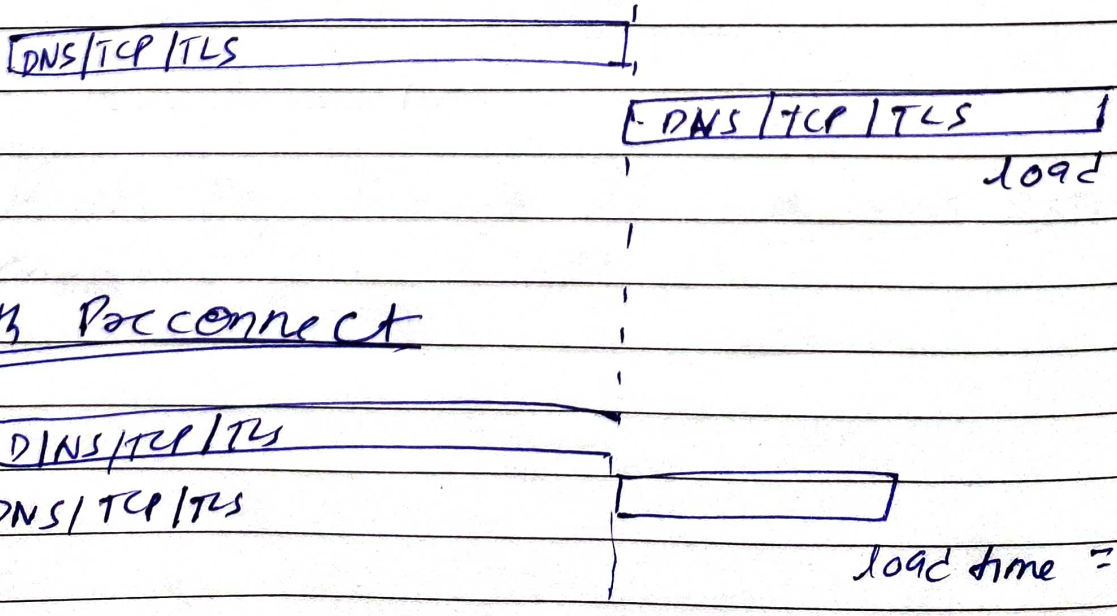
Api request sent

ack. back

} This is called handshack like saying Hi to domain.

→ by using preconnect we can provide pre-handshack to minimize the time.

## without Preconnect

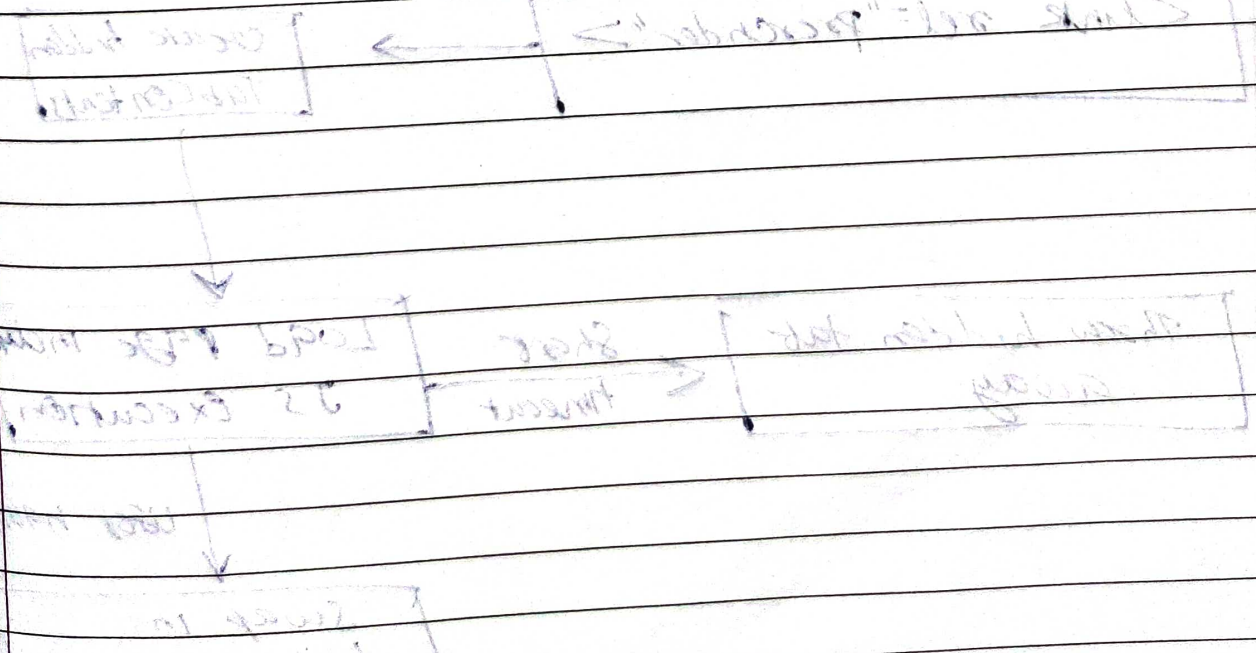



## With Preconnect

Camlin | Page
Date / /

preconnect may be not available in all browser as it has recently started.

<link rel = "preconnect" href = "http...." />



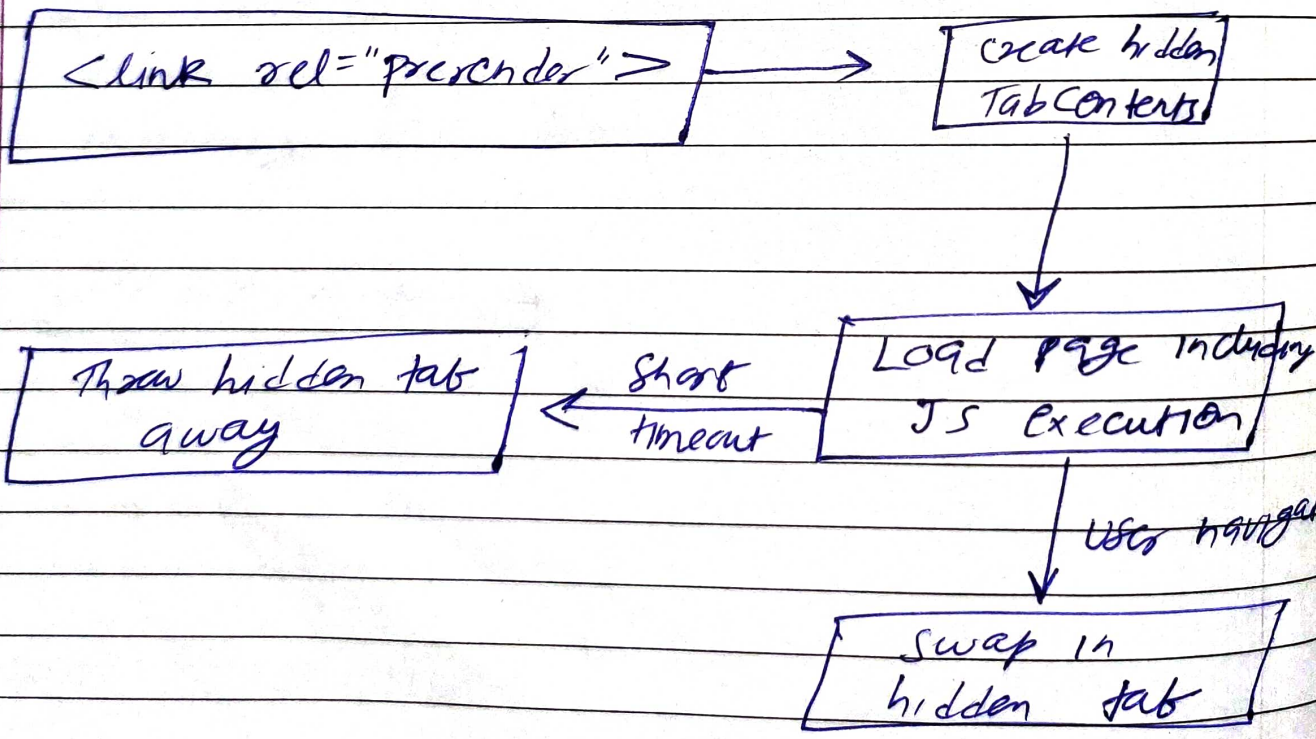
## Dns - prefetch (same as pre-connect) supports in all browser

<link rel="dns-prefetch" href="https://...">

## prerender

→ load the data which your are sure that user is going to load it - after immediate.

→ It will give smooth behaviour of the page




<link rel="prerender" href="https://.....">

# Resource Hinting (modulepreload)

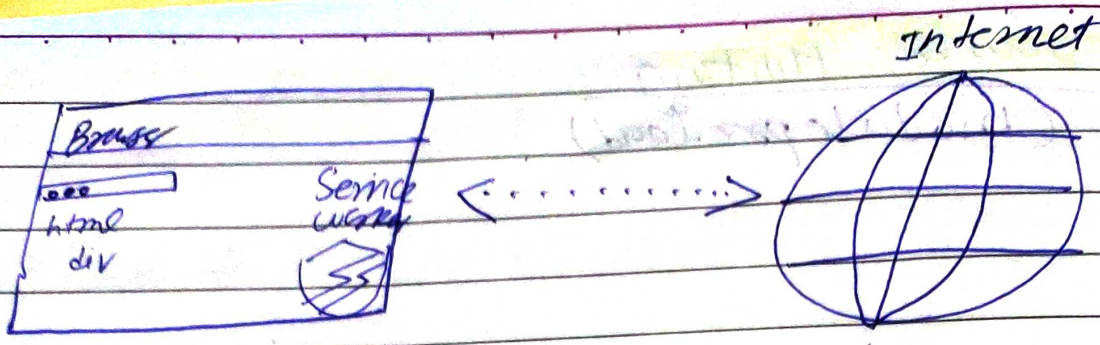
<link rel="modulepreload" href="/static/Header.js" />

<link rel="modulepreload" href="/static/Header.js" as="service worker" />

→ It required with pwa service worker. Basically u can leverage this.

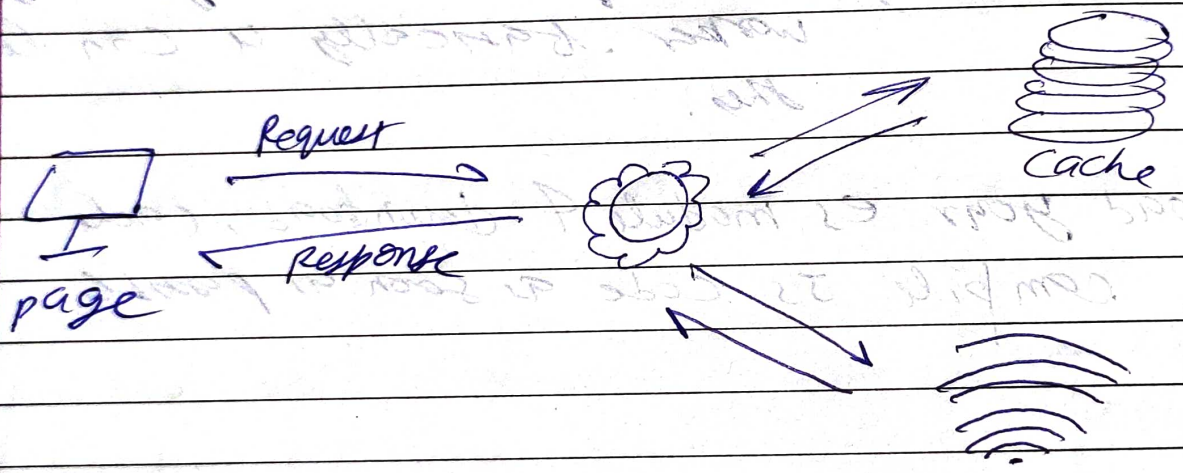
load your es module & download, cache & compile JS code as soon as possible

# Service Worker




→ Service worker are something like proxy server which sit between browser & Internet (webpage).

→ Everything is intercept by service worker.




## worker lifecycle

1. install
2. installed
3. Activating
4. Activated
5. Redundant

1. Registration
2. Installation
3. Activation
4. fetch

Camlin Page
Date / /

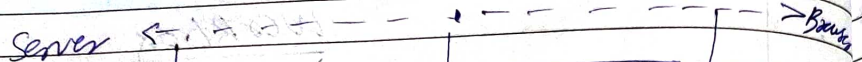
## WORKBOX

- Pre caching
- Runtime caching
- Strategies
- Request routing
- Background sync
- Helpful debugging

CSR, SSR, CSR + SSR, static rendering, pre rendering

Client side Rendering

Server side Rendering




| | Server rendering | Static SSR | SSR with (Re) hydration | CSR with Prerendering | Full CSR |
|---|---|---|---|---|---|
| Overview | An application where I/P is navigation request & the O/P is HTML in response to them. | Build as a single Page App, but all pages prerendered to static HTML as a build step & the JS is removed. | Build as a single Page App. The Server prerenders page but the full app is also booted on the client | A single Page App, where the initial shell/skeleton is prerendered to static HTML at build time. | A single Page App. All logic rendering & booting is done on the client. HTML is essentially just script & style tags. |
| Authoring | Entirely server-side | Built as client-side | Built as client-side | Client side | Client-side. |
| Rendering | Dynamic HTML | static HTML | Dynamic HTML & JS/DOM | Partial static HTML, then JS/DOM | Entirely JS/DOM |
| Under the hood | controls all aspect | Delivers static HTML | Renders page | Delivers static HTML | Delivers static HTML |
| Props | TTI = FCP fully streaming | fast TTFB TTI = FCP fully streaming | flexible | flexible fast TTFB | flexible fast TTFB |
| Example | Gmail, HTML, Hacker News | Documentation, Netflix | Next.js, Razzle | Gatsby, Vue | Most apps, |

# Compression
## (Brotli VS GZIP)

Response Headers

cache-control: max-age=473607

content-encoding: gzip

content-length: 38988

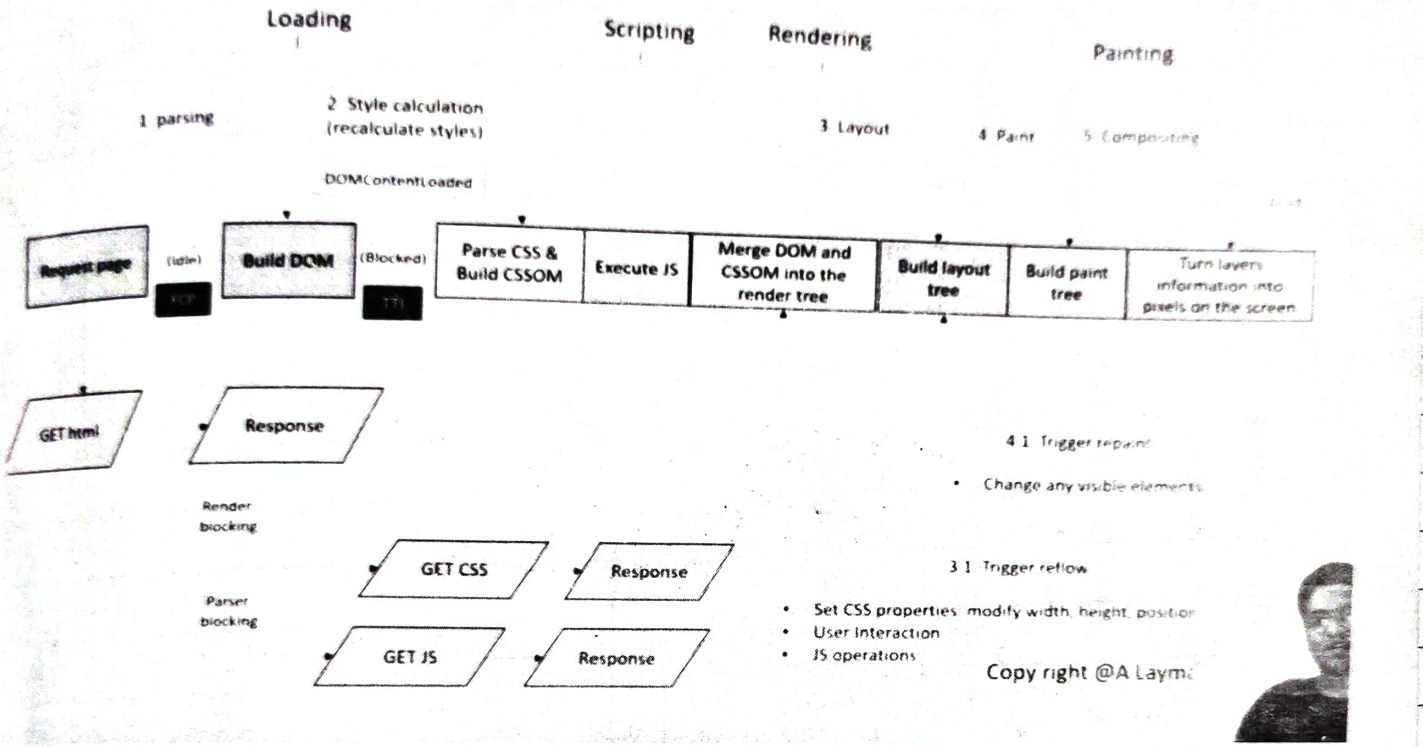
Response Headers

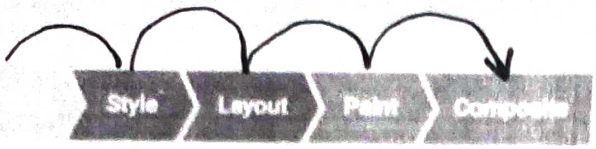
cache-control: max-age=2811

content-encoding: br

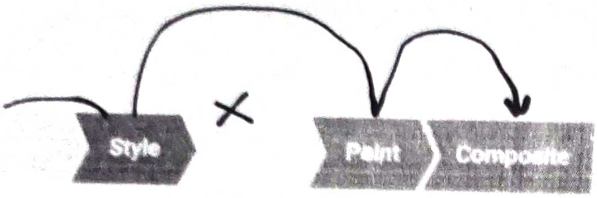
content-length: 30942.

# Layout Shifts & Repaints

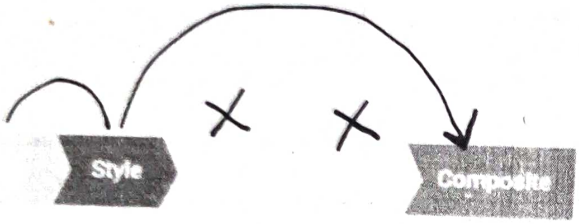





+




Background-color, color, shadows, visibility




Transform & opacity

Element

MouseEvent

Window

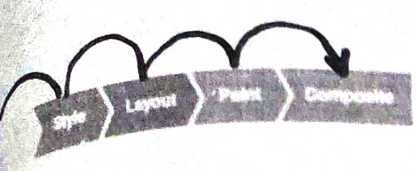
Frame, Document & Image

15

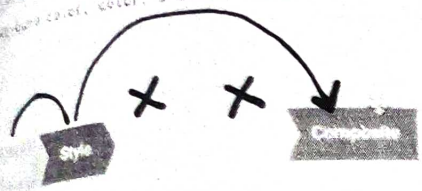
20

25

30









1/60 = 16.67ms

10ms