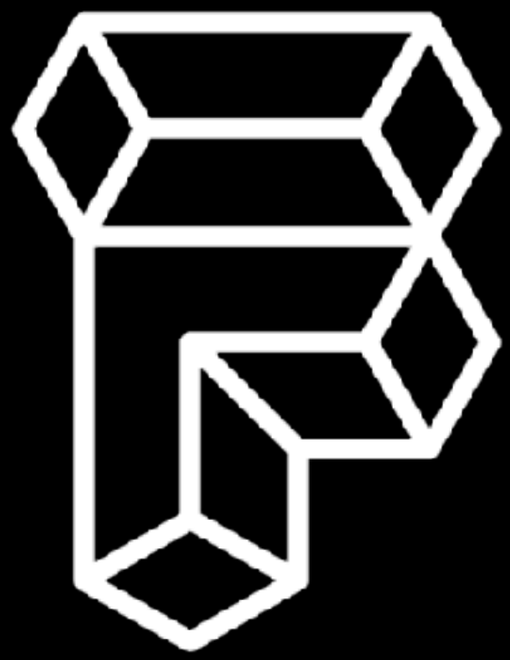
High-Performance Responsive Images

Chris Bolin
Formidable



Formidable℠

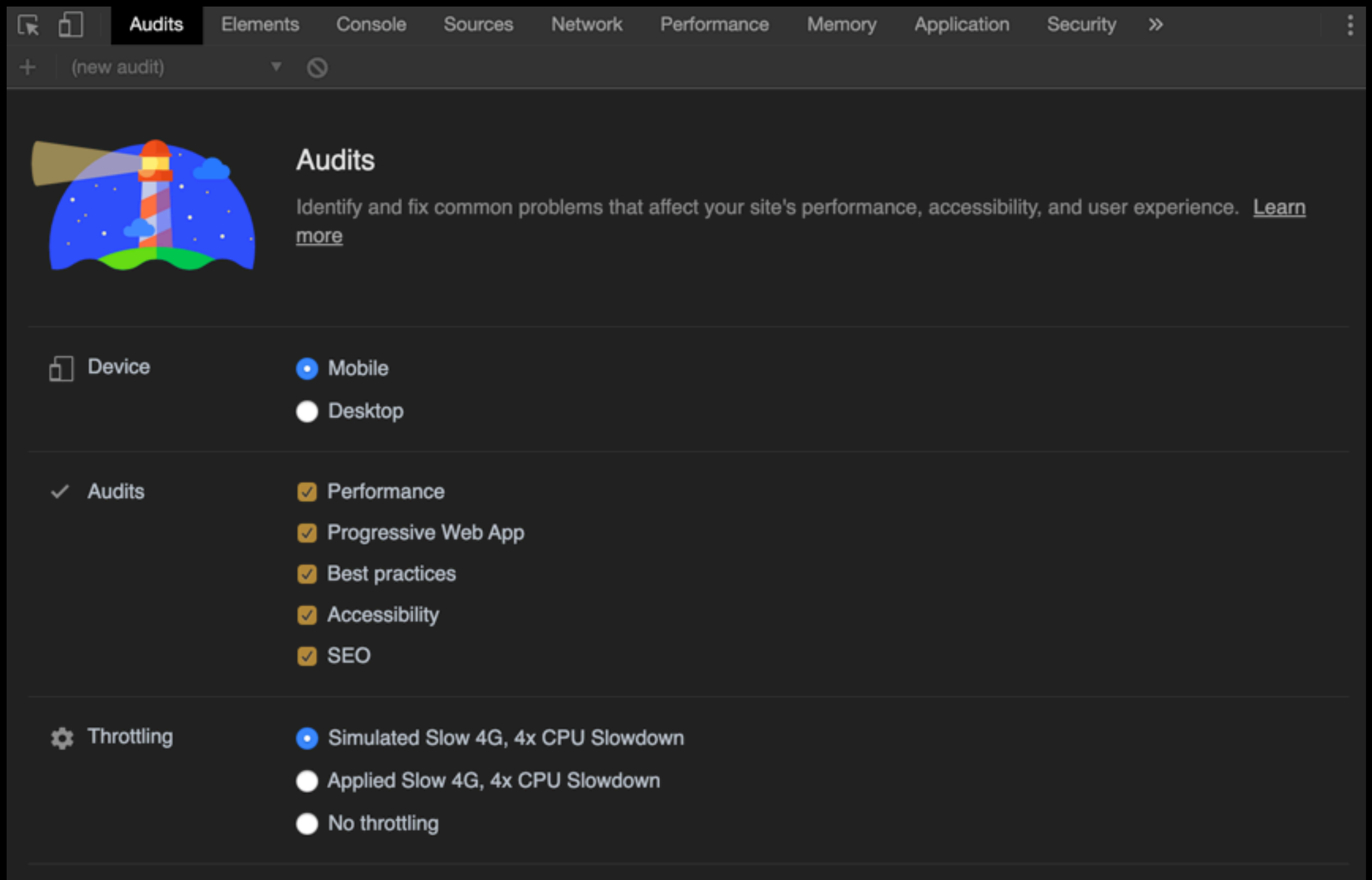
Images on the web

Images are about **half of the downloaded content** on modern pages

Images can have a very large impact on **performance**

Analyzing images can give **insight into browsers**—how they process content and display it

Assessing performance: Chrome Lighthouse



What makes an image both responsive and fast?

Hint: it’s a laundry list

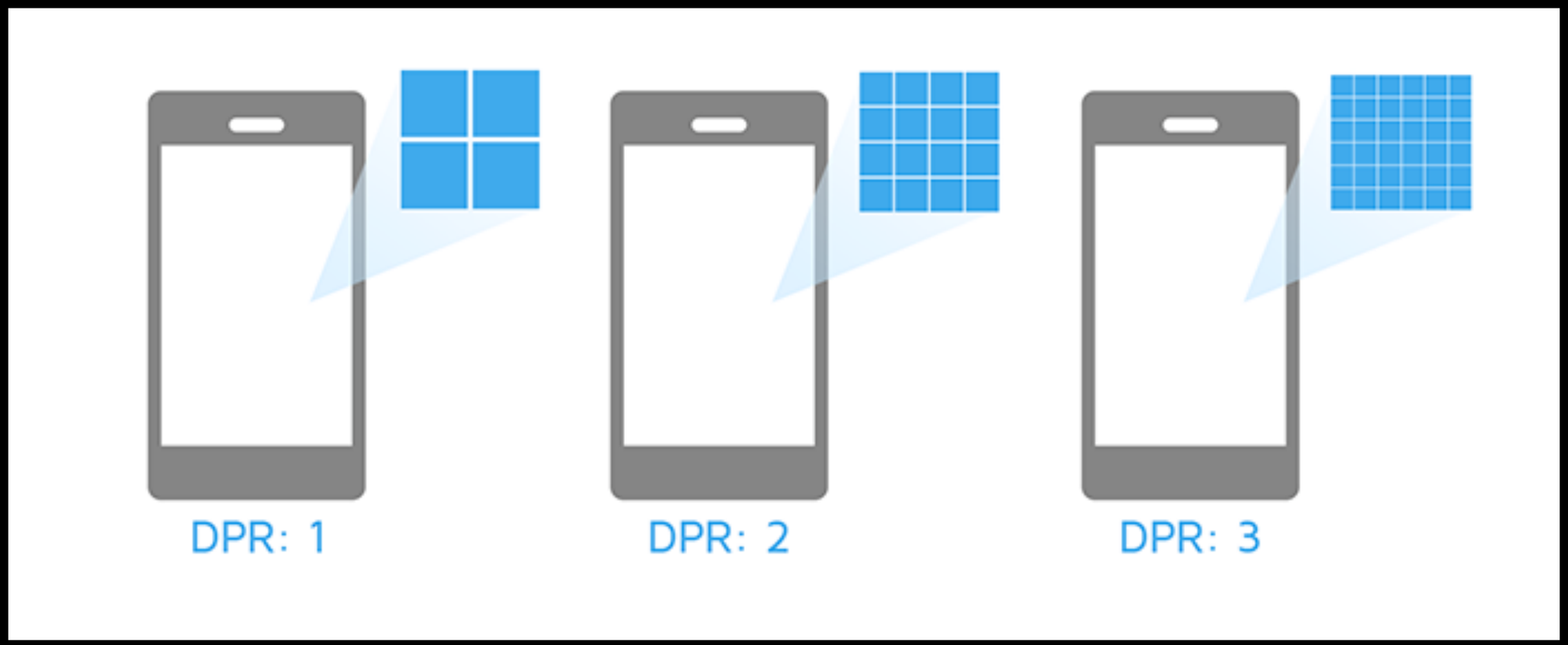
Correct dimensions for the viewport

- Avoid pixelation (image is too small)
- Avoid excessive download payloads (image is too big)
- *[e.g. document.body.clientWidth]*

Correct dimensions for the device's "pixel ratio"

- **Pixel Ratio** is the number of device pixels are in one CSS/JS/DOM pixel. *[window.devicePixelRatio]*

- Common DPR values: 1.0, 1.3, 1.4, 1.5, 2.0, 2.625, 3, 3.5, and 4.0 - "retina" is now far too ambiguous



Correct image format for the browser

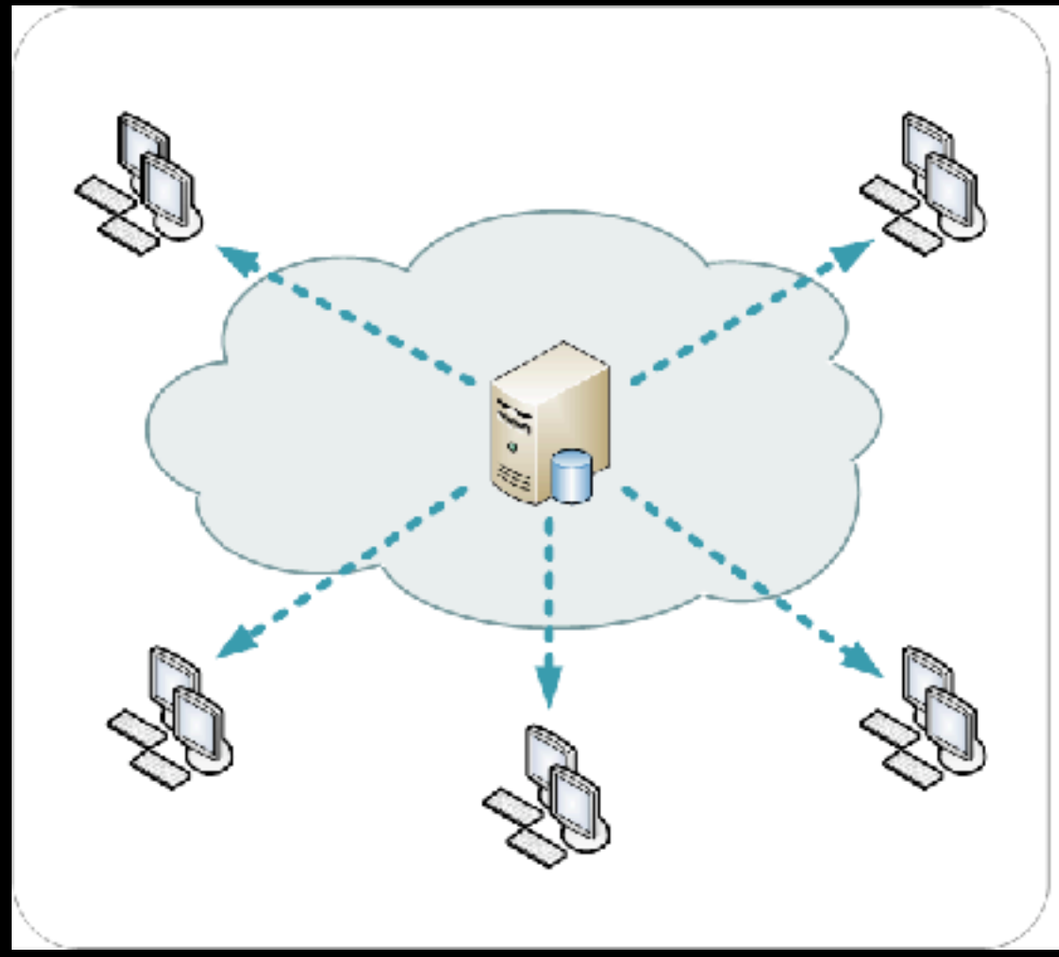
- Different browsers support JPEG, JPEG 2000, JPEG XR, and WebP.
- Different image formats can reduce download size and increase quality.

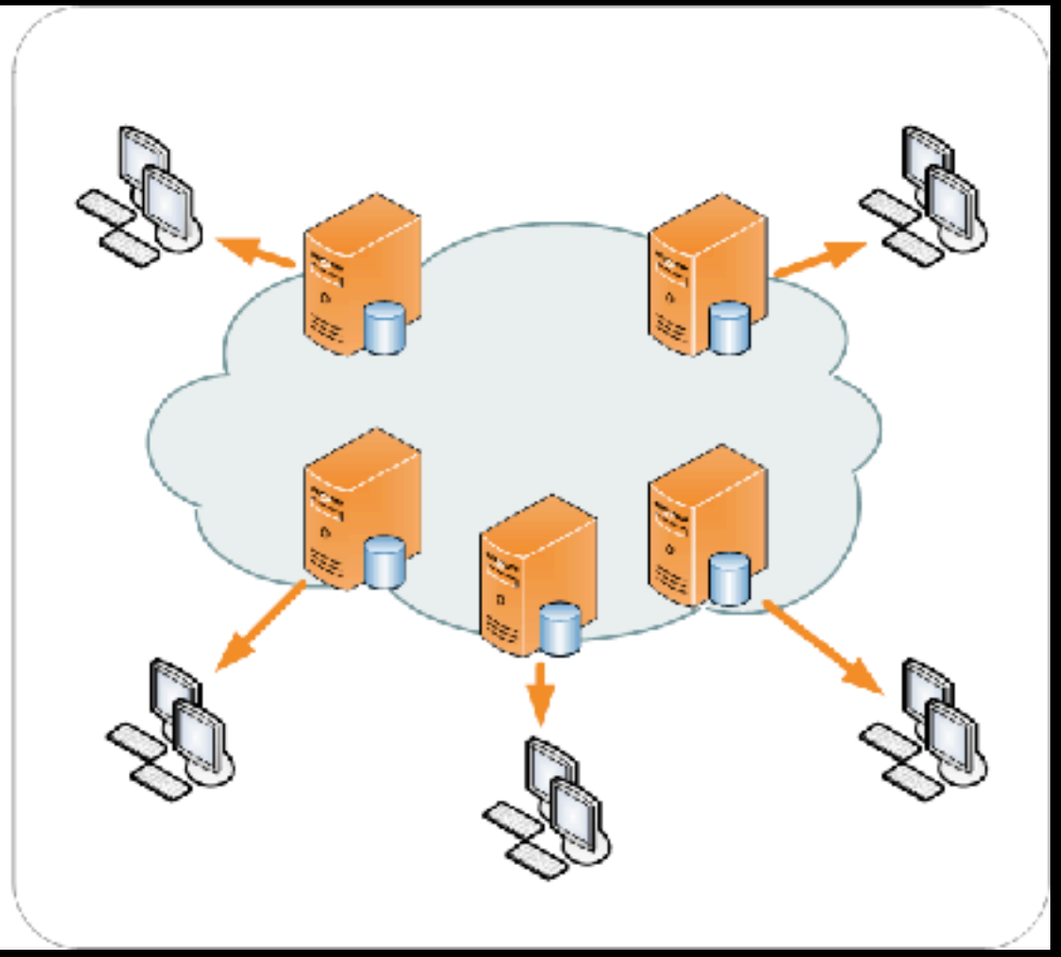
Adheres to the design

Image may be full-width, relative-width, or fixed-width depending on the screen size. This is called the “Art Direction” problem, using different image *variants* (not *sizes*)

Edge cacheable

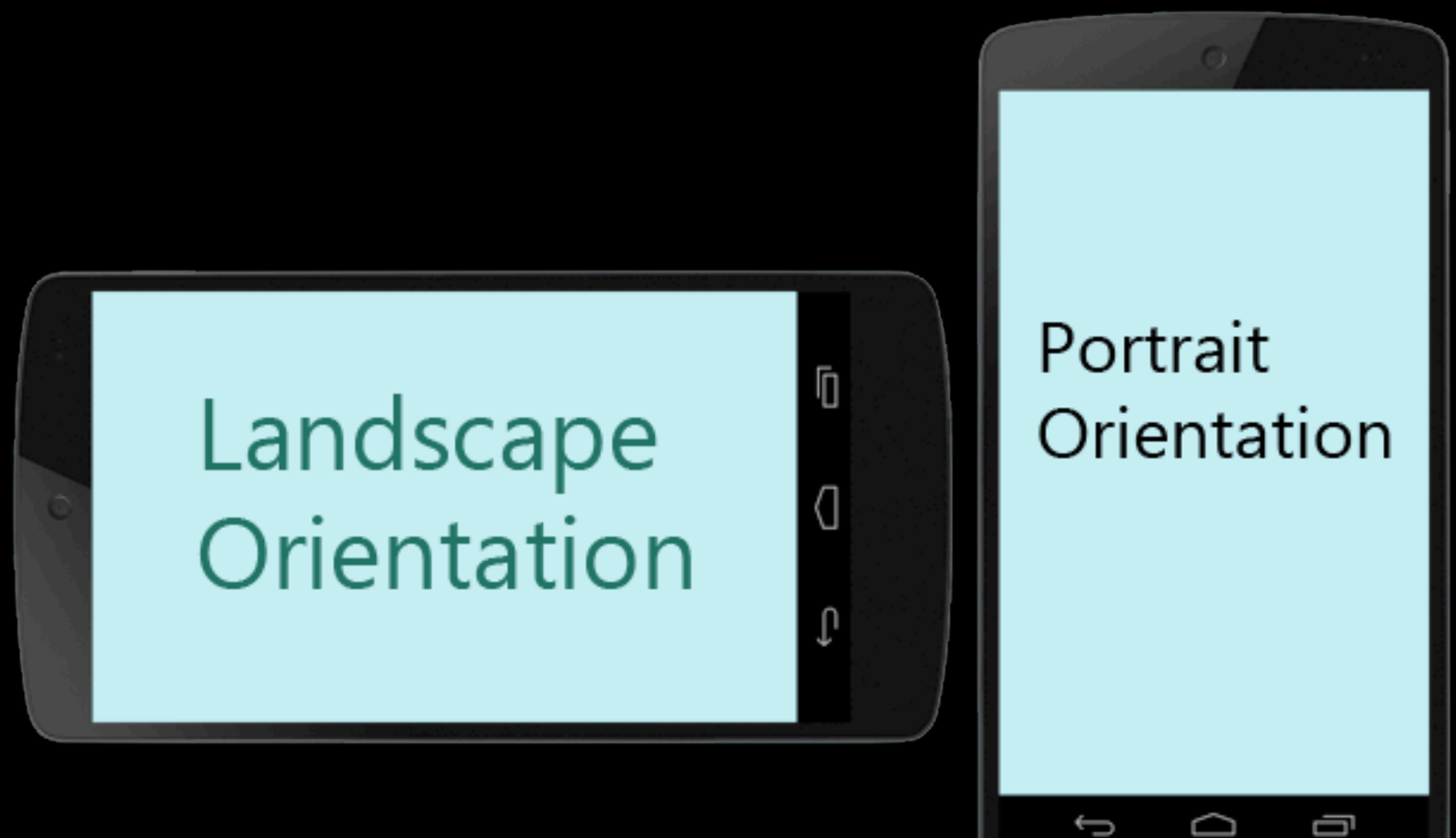
- Should be cacheable by a CDN.
- CDNs use URLs (and URLs only) to determine cache hits





Browser cacheable

- If a browser downloaded a larger image and cached it, just use that.
  [e.g. service worker's Cache.add(request)]
- The fastest network request is no network request.



Adapts to slow connections

- If user is on a slow connection, serve a low-res image
  e.g. *[navigator.connection.effectiveType]*

NetworkInformation.effectiveType

Web technology for developers › Web APIs › NetworkInformation › NetworkInformation.effectiveType

English ▾

This is an experimental technology

Check the Browser compatibility table carefully before using this in production.

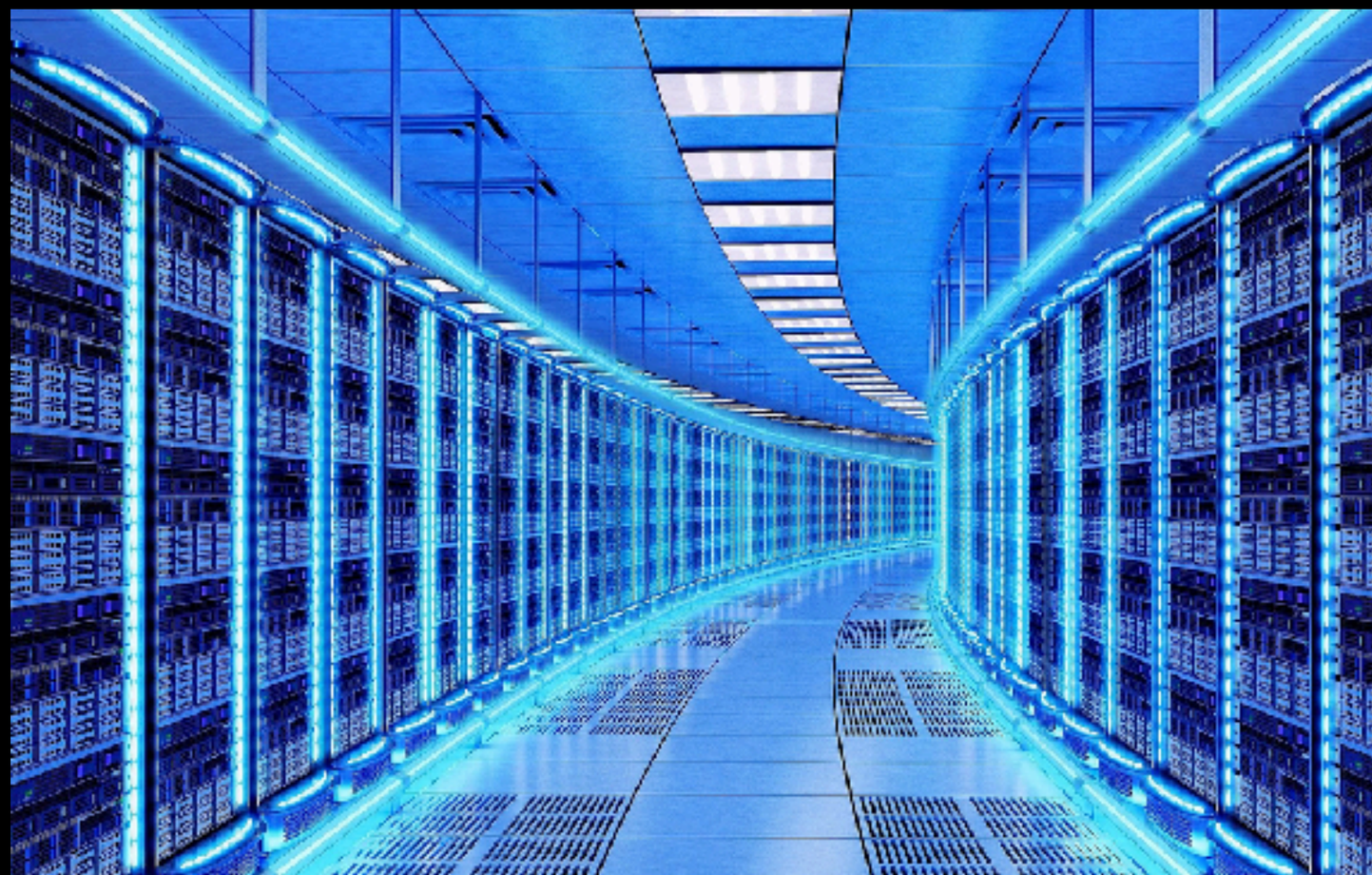
The `effectiveType` read-only property of the `NetworkInformation` interface returns the effective type of the connection meaning one of 'slow-2g', '2g', '3g', or '4g'. This value is determined using a combination of recently observed, round-trip time and downlink values.

Respects server side rendering (SSR)

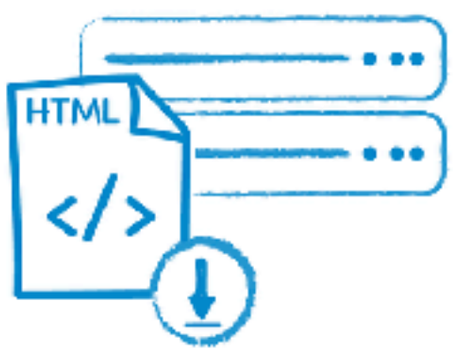
- The same image is downloaded from server- and client-rendered DOM.

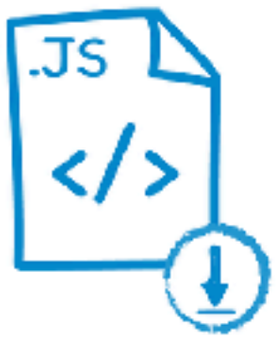
- This means the server code needs to know <u>everything</u> the client does

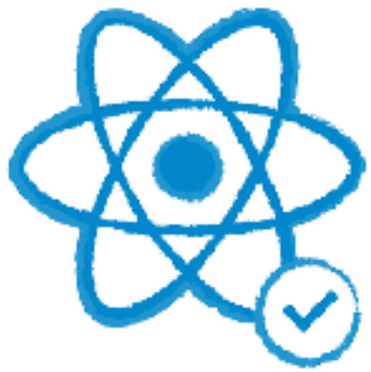
But what is SSR?



CSR





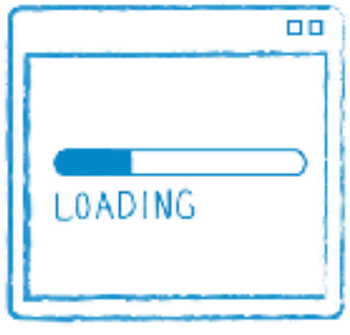


Server Sending Response to Browser

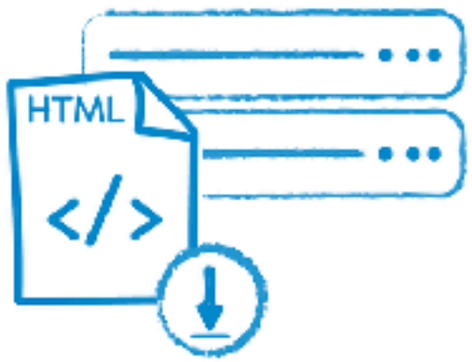
Browser Downloads JS

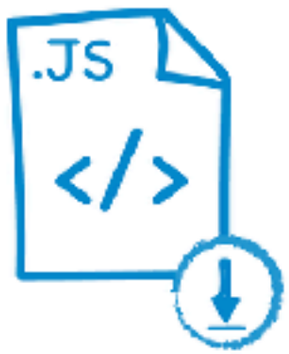
Browser executes React

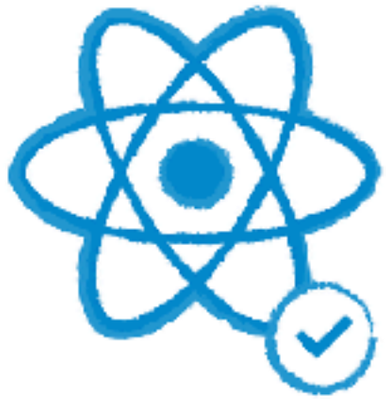
Page Now **Viewable** and **Interactable**



SSR





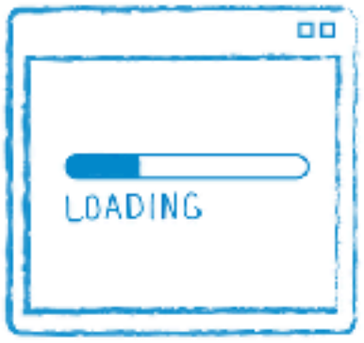


Server Sending Ready to be rendered HTML Response to Browser

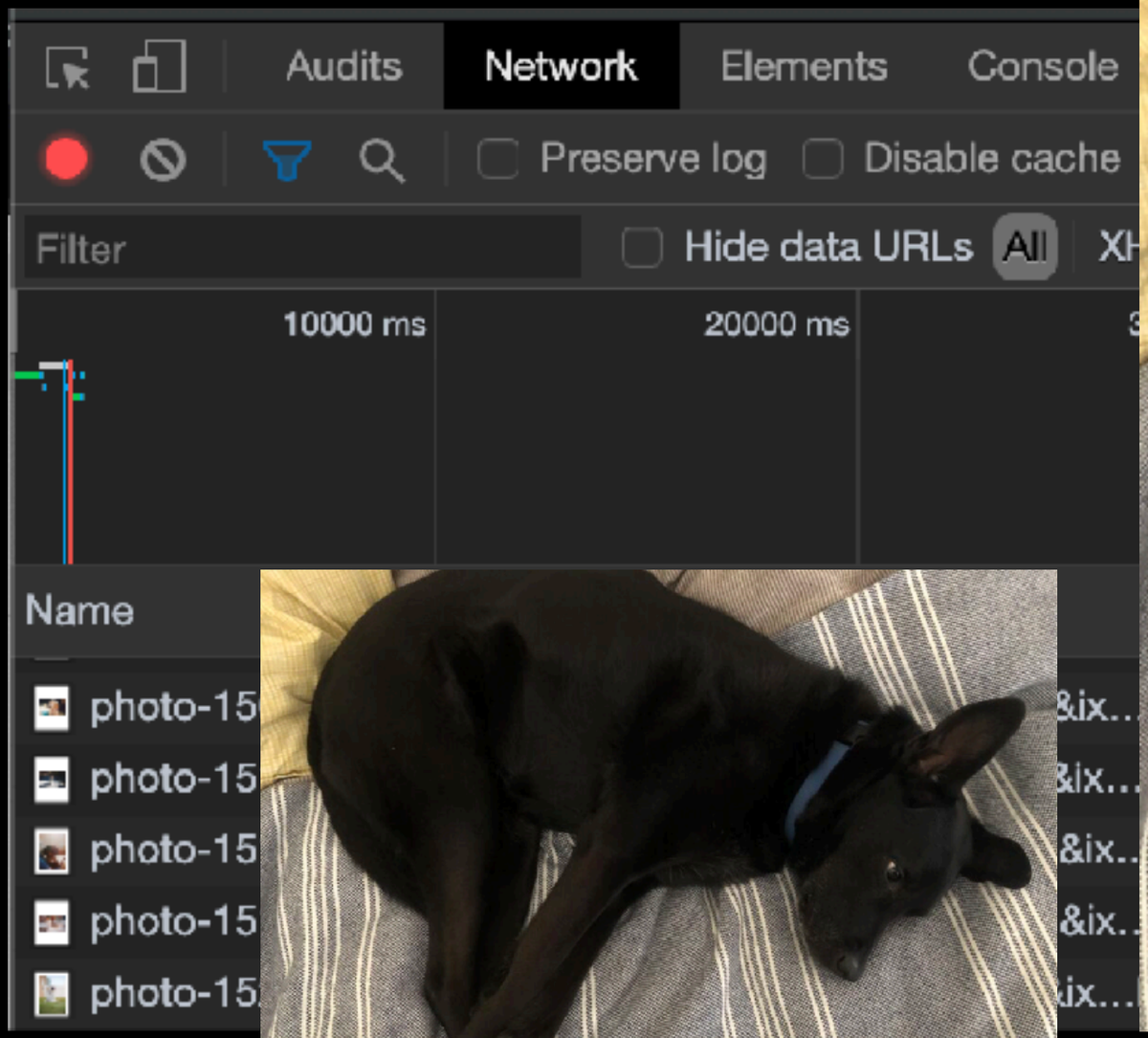
Browser Renders the page, **Now Viewable**, and Browser Downloads JS

Browser executes React

Page Now **Interactable**



A very strange bug: Images downloaded twice



img/harley-dog-800.jpeg

img/harley-dog-1600.jpeg

Building the perfect responsive image system

- Server:
  - Parse user agent
  - Look up client viewport width (TODO)
  - Look up device pixel ratio (TODO)
  - Determine device orientation (TODO)
  - Set reasonable fallbacks for new devices (TODO)
  - Implement effectiveType polyfile… *on the server* (TODO)
- Client:
  - Cache images in a Service worker, but account for device orientation and viewport width! (TODO)
- Determine all the other things we need to do (TODO)

~~Use JavaScript~~
Use the Platform

- You *might* be able to create a very elaborate system to handle this, but **please don't**.

- The browser can do it for you!
  - **img** element - **srcset** and **sizes** attributes
  - **picture** element - **source** and **img** child

Full-Width Images

```
<img srcset=“hero-480.jpg 480w,
             hero-800.jpg 800w,
             hero-1600.jpg 1600w"
     src=“hero-800.jpg”>
```

hero-480.jpg - image url

480w - image width in **actual pixels**

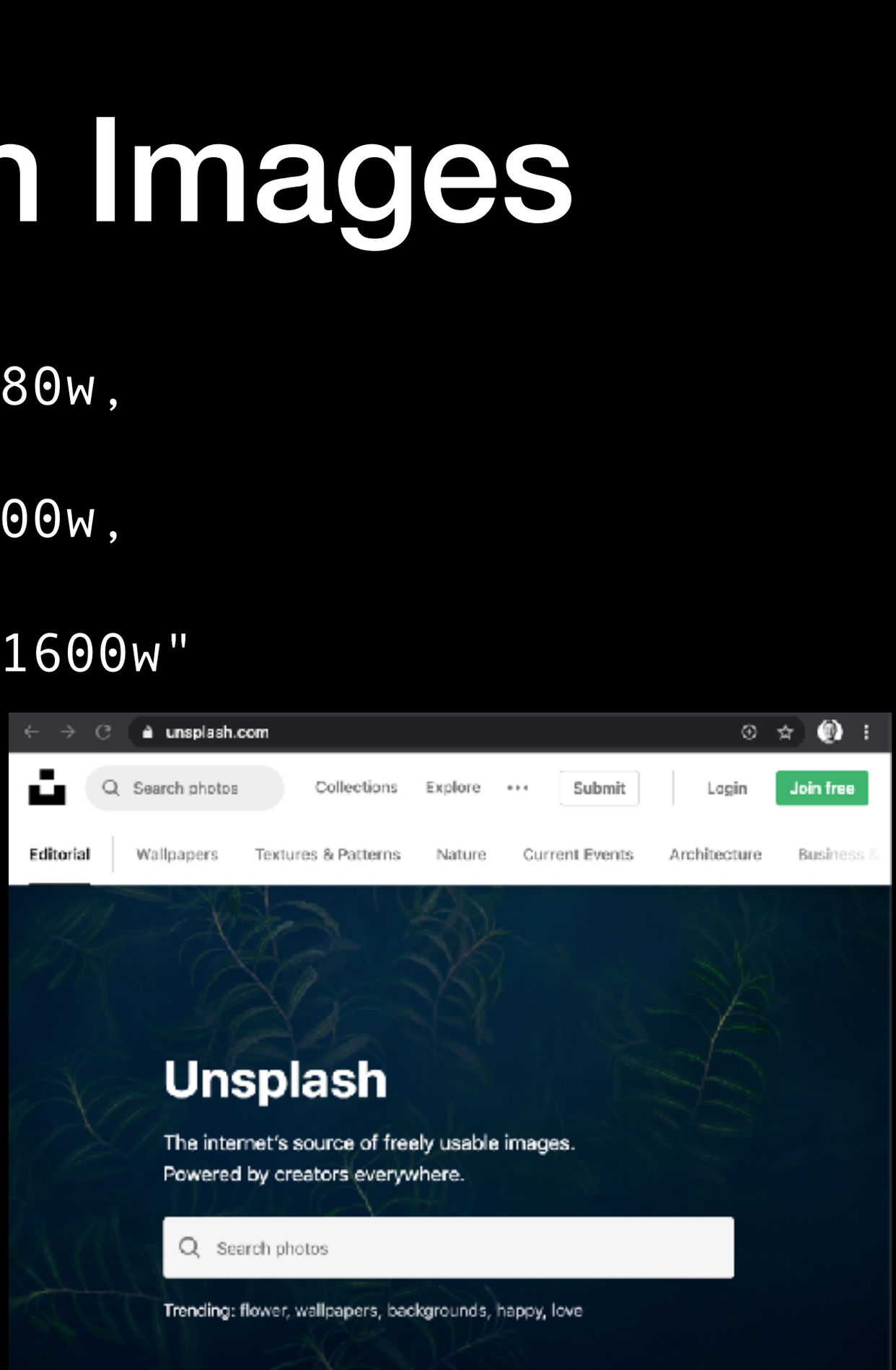


Image slot width

Not every image is full width!

```
<img srcset="cat-480.jpg 480w,
             cat-800.jpg 800w,
             cat-1600.jpg 1600w"
sizes="(max-width: 320px) 90vw,
       (max-width: 800px) 45vw,
       20vw">
```

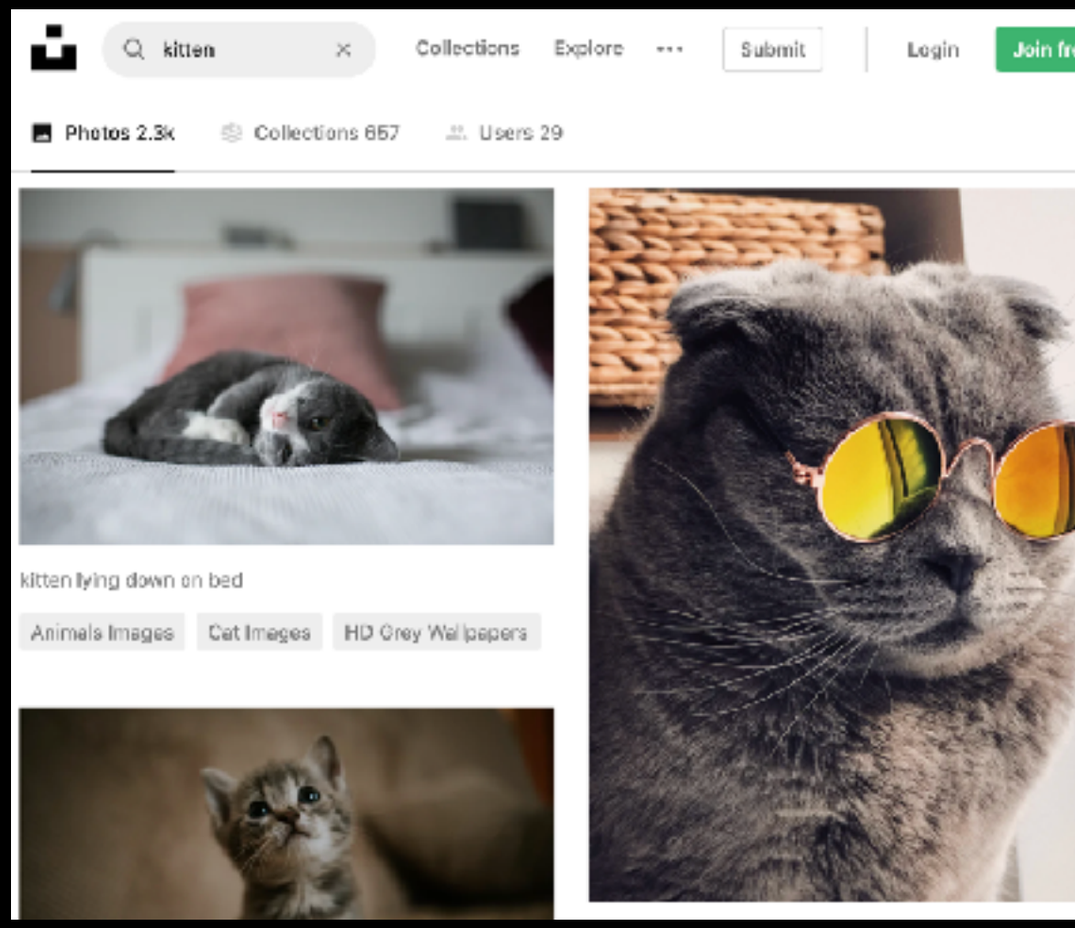


Image Services

(Adobe Scene7, Cloudinary, Akamai image manager)

Image services can resize images on-demand,
usually via a query search param.

```
<img srcset="fluffy-dog-hero?wid=480 480w,
```

```
fluffy-dog-hero?wid=800 800w,

fluffy-dog-hero?wid=1600 1600w">
```

Image Formats

```
<picture>

  <source type="image/webp"
          srcset=“oh-hi.webp”>

  <source type="image/svg+xml"
          srcset="oh-hi.svg">

  <img src=“oh-hi.jpeg”>

</picture>
```

Art Direction

```
<picture>

  <source media="(max-width: 799px)”
          srcset="dog-portrait.jpg">

  <source media="(min-width: 800px)"
          srcset=“dog-landscape.jpg">

  <img src=“dog-landscape.jpg”>

</picture>
```

Safari: the new IE

- Safari (iOS and desktop) download the image in *both* <source> and <img>

- No optimizations for handling an *img* that is built by JS and contains *src, srcset,* and *sizes.*

- **Solution**:

  - Only include *src* for the browser that needs it: IE 11 (Will mean slightly slower IE 11 experience)

  - *sizes* prop should always come before *srcset* prop.

```
const source = document.createElement('source');
// set the attributes...
const img = document.createElement('img');
// set the attributes...
const picture = document.createElement('picture');
picture.appendChild(source);
picture.appendChild(img);
```

Choosing Image Widths

- This very much depends on user base. For our project (USA and Canada)…
    - 320, small image on desktops or slow connections
    - 750, phones 2x (iPhone 6,7,8)
    - 1080, phones 2.6x (iPhone 6,7,8 Plus)
    - 1125, phones 3x (iPhone X)
    - 1280, desktops 1x, iPhone 3x, phones 3x
    - 1440, desktops 1x, phones 3x
    - 1600, desktops 1x, iPad 2x, phones 4x
    - 2420, desktops 2x
    - 2644, 2x grid max

There’s so much more to talk about!

- Lazy image loading
- Image resizing / unsharp masks
- Offscreen images
- Placeholder images

…but only so much time.

ICYMI

- Images can have a **huge effect on performance**
- **Lighthouse** is a great performance benchmarking and troubleshooting tool
- Use the platform for responsive images:
  img element - **srcset** and **sizes** attributes
  picture element - **source** and **img** child
- **Questions?**