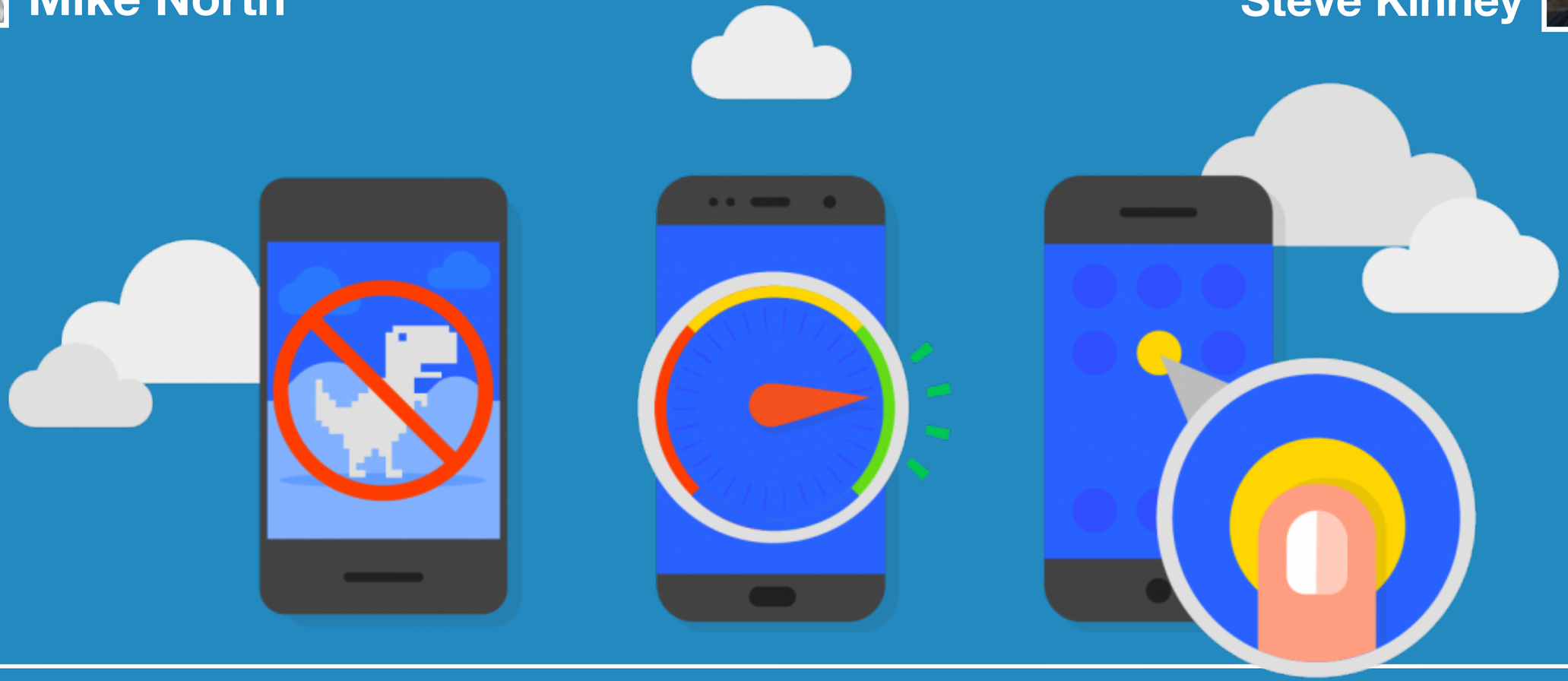
LinkedIn
Mike North

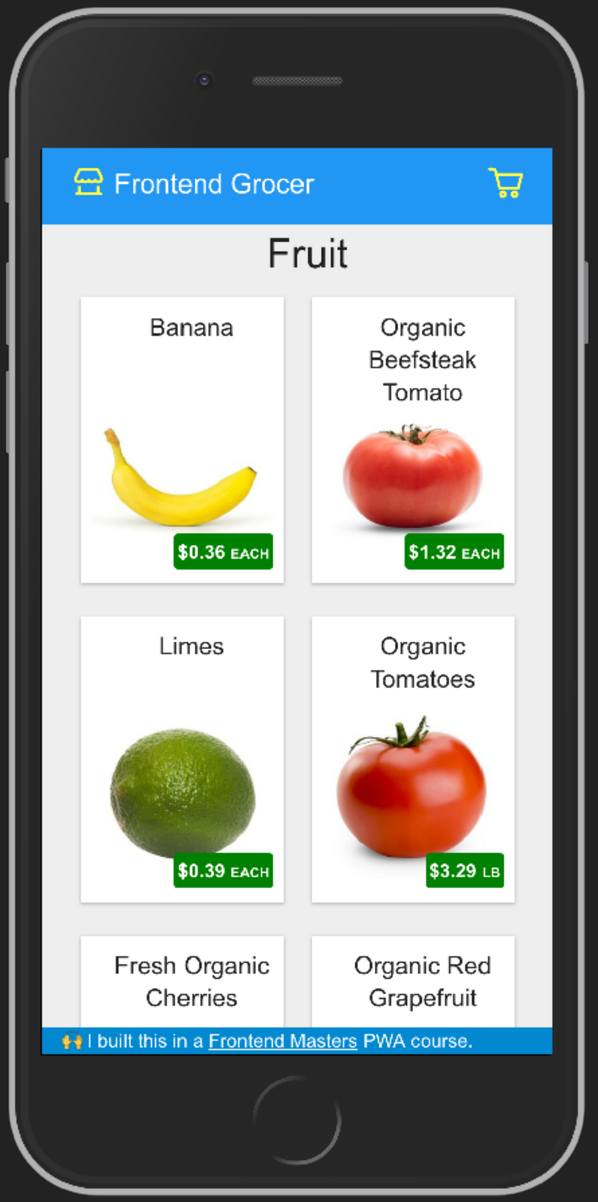
SendGrid
Steve Kinney



Progressive Web Fundamentals

What is a Progressive Web App?

- Use the latest advanced web technologies
- Adopt Progressive Enhancement as a core tenet
- Can transcend what users expect from the browser
- Offer a native-app-like experience



Characteristics of Progressive Web Apps

Progressive

Responsive

Works Offline

App-Like

Fresh

Safe

Discoverable

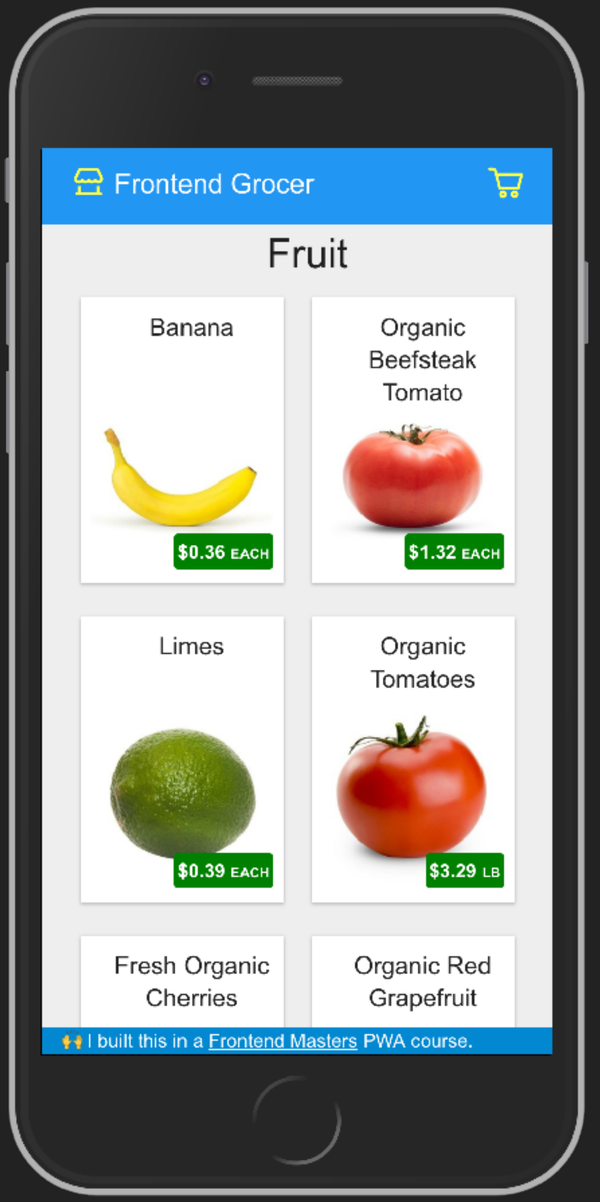
Re-Engageable

Durable

Linkable

Our Project: Frontend Grocer

- It's currently a "classic" single-page app
- We'll add support for
  - Background Tasks
  - Notifications
  - Offline Boot
  - Custom 404 images
  - Offline add-to-cart
  - Background Sync
  - Native-app look & feel
  - Offline on iOS



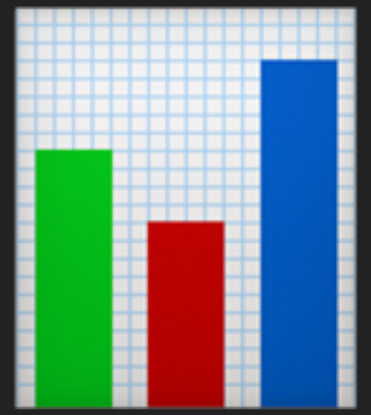
Measuring Apps

Important Metrics

Time to...

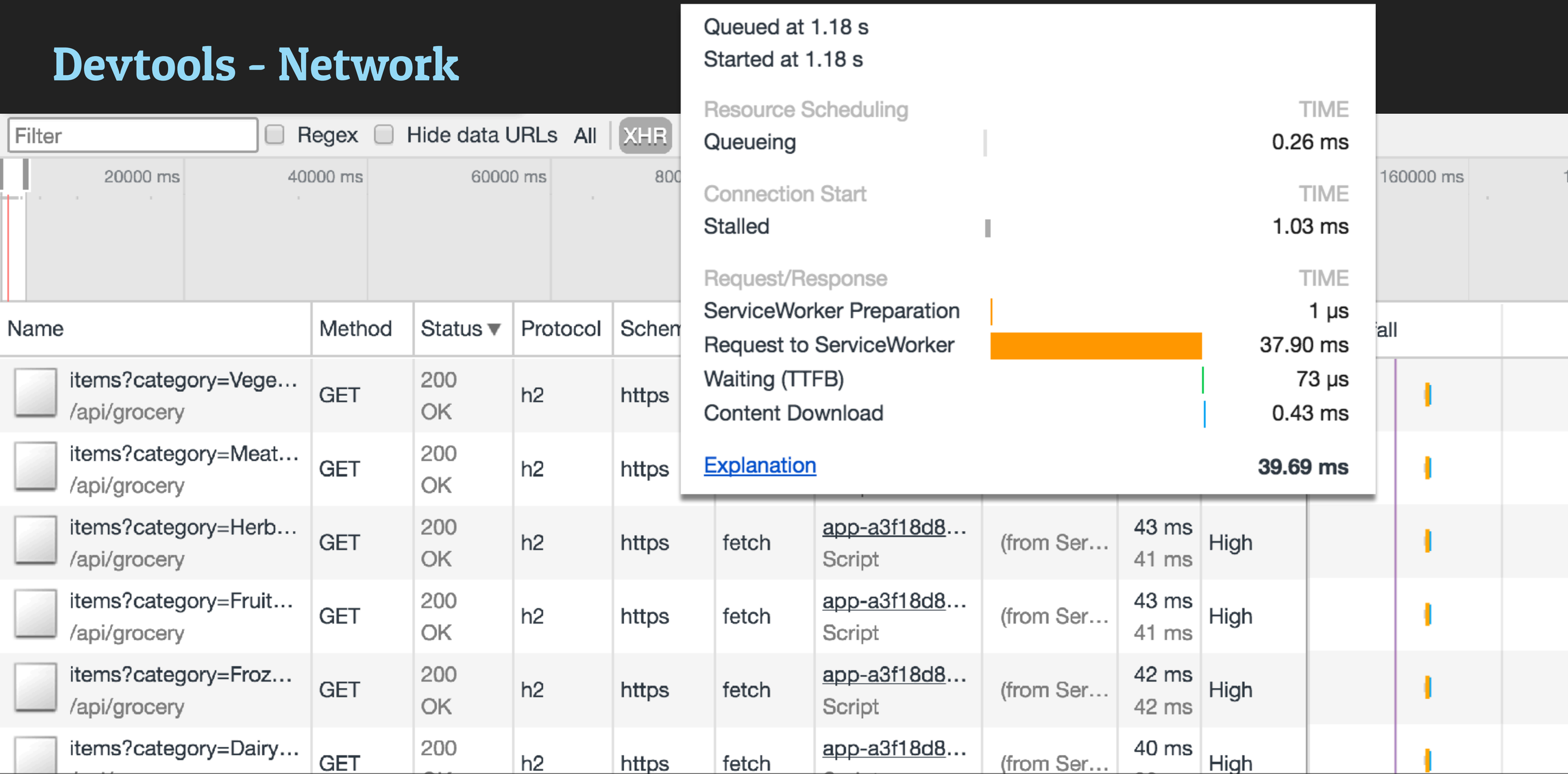
First Paint

Interactive



Data

Devtools - Network



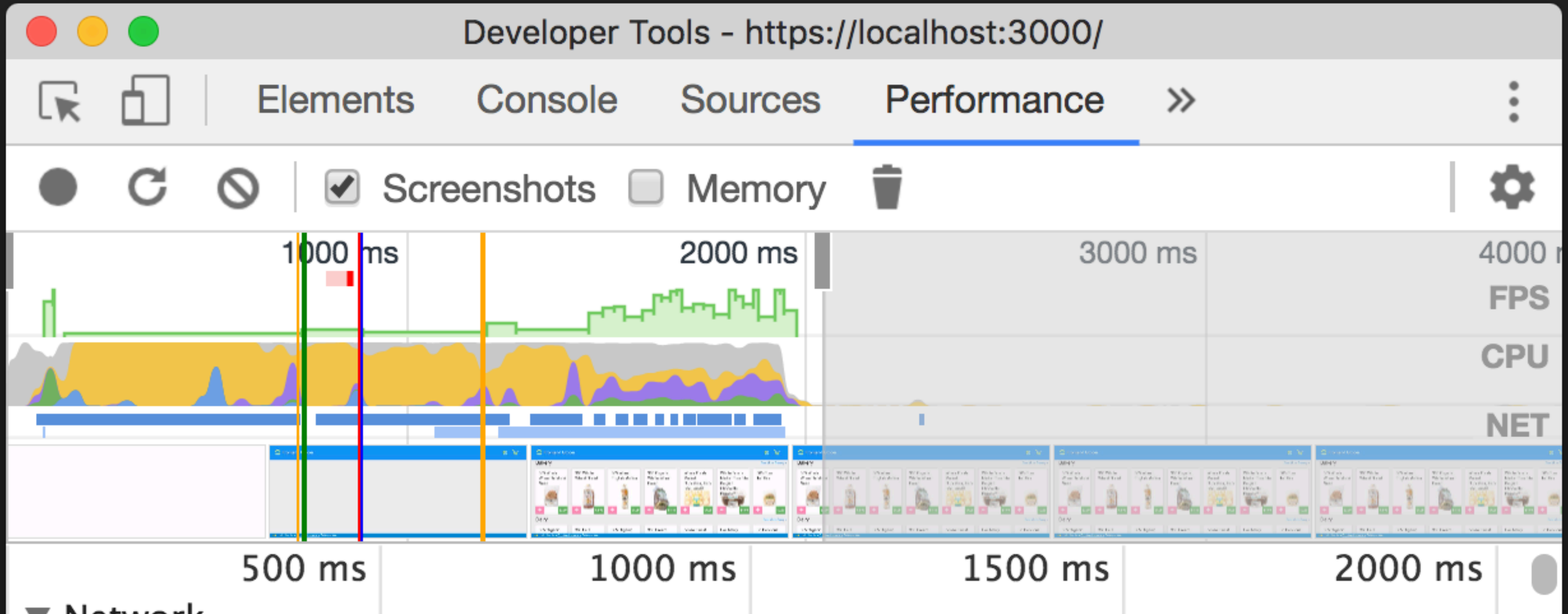
Devtools - Performance

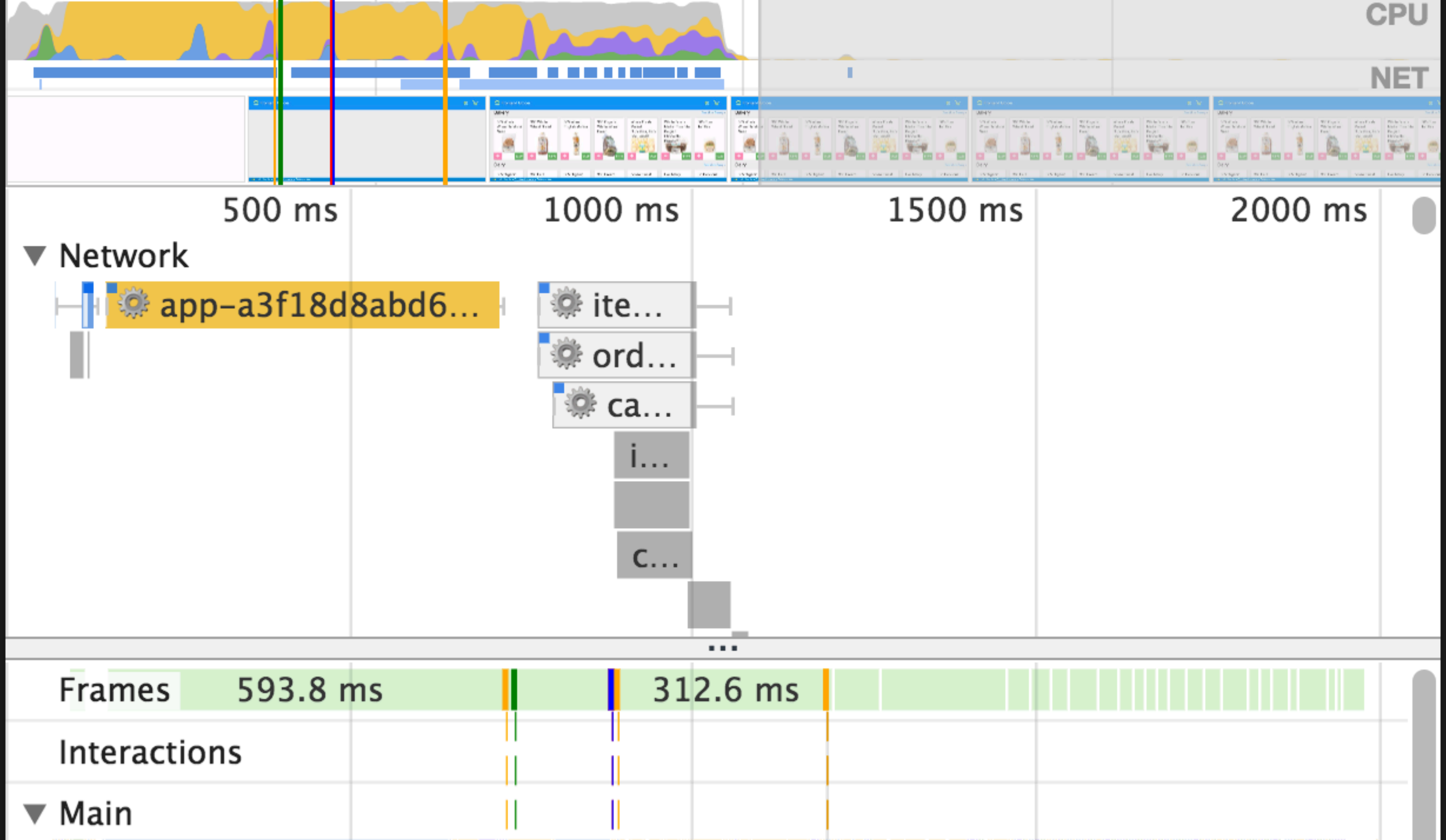
Timeline

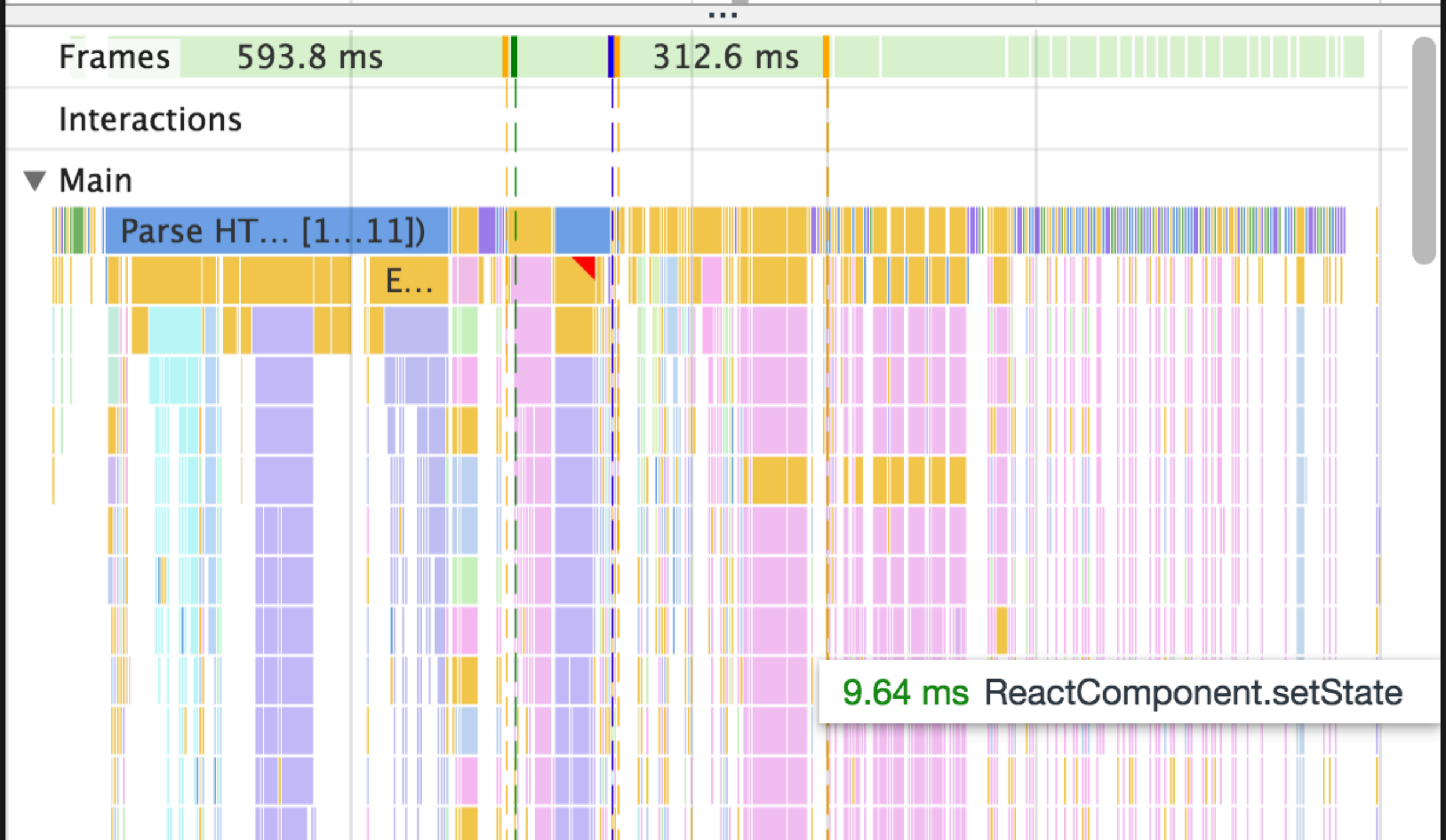
Network Usage

CPU Usage

Devtools - Performance




CPU
NET
500 ms
1000 ms
1500 ms
2000 ms
Network
app-a3f18d8abd6...
ite...
ord...
ca...
i...
c...
Frames
593.8 ms
312.6 ms
Interactions
Main


Frames
593.8 ms
312.6 ms
Interactions
Main
Parse HT… [1…11])
E…
9.64 ms ReactComponent.setState

Results for: https://www.mike.works/
Generated on: 3/26/2017, 12:01:45 PM PDT

Export...

99
100

Progressive Web App

These audits validate the aspects of a Progressive Web App. They are a subset of the PWA Checklist.

App can load on offline/flaky connections

Page load performance is fast

Users notice if sites and apps don't perform well. These top-level metrics capture the most important perceived performance concerns.

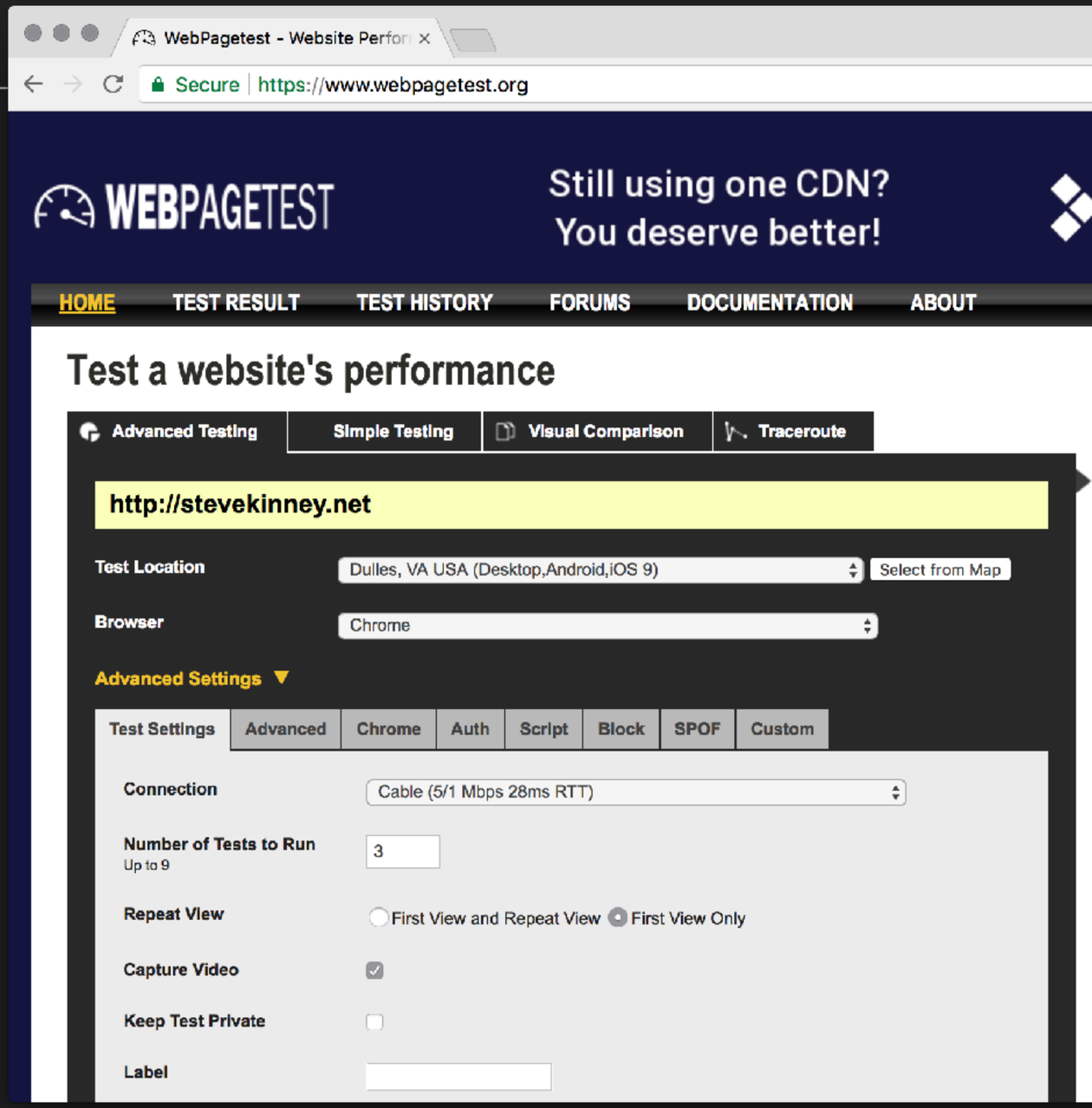
91 First meaningful paint: **2021.0ms** (target: 1,600ms)

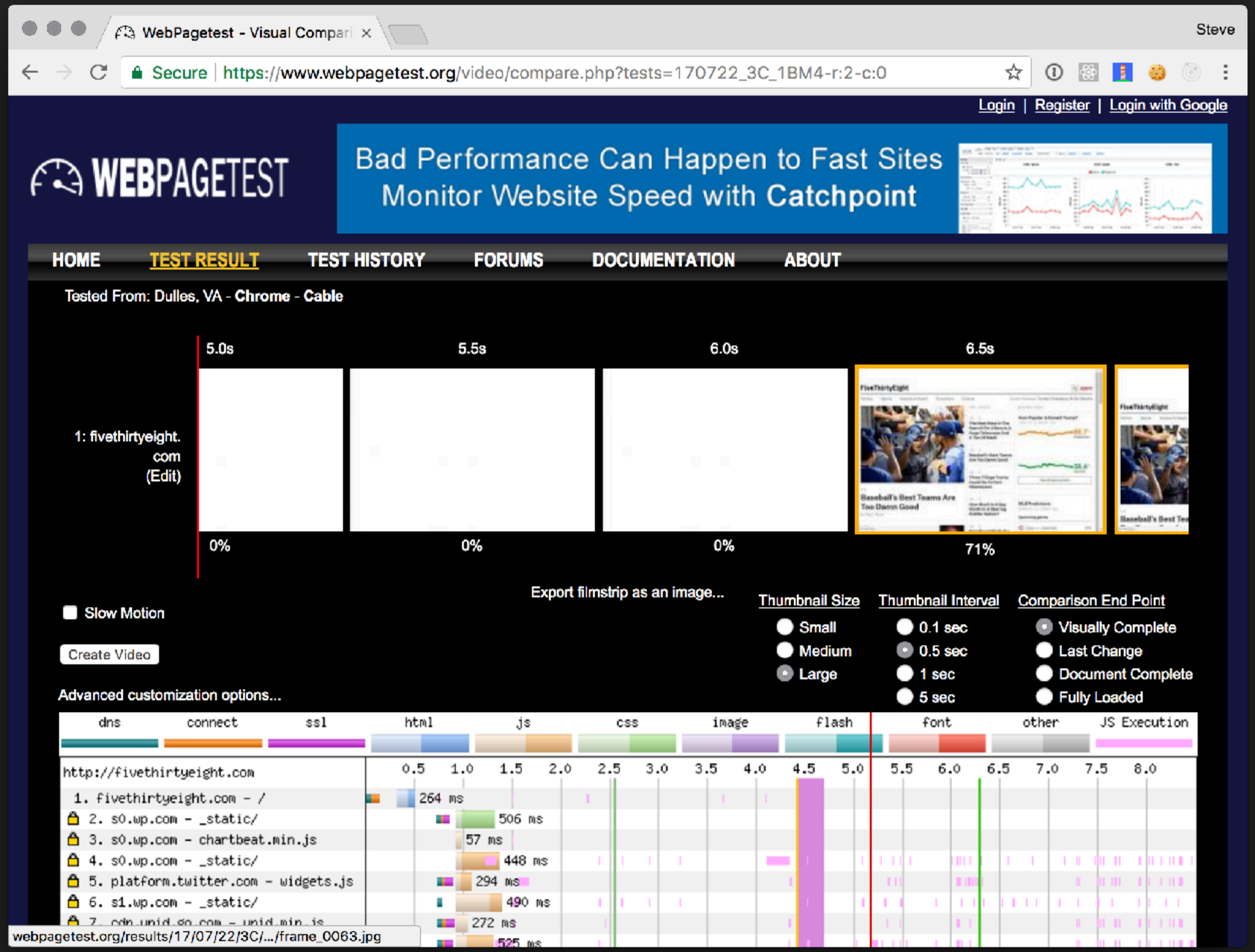
99 Perceptual Speed Index: **1069** (target: 1,250)

First Visual Change: **388ms**

webpagetest.org

- Allows you to run speed tests from various locations around the world
- Uses real browsers at real connection speeds
- Supports some pretty advanced features
  - Multi-step transactions
  - Video capture
  - Content blocking
  - Single Point of Failure Testing
  - Scripting



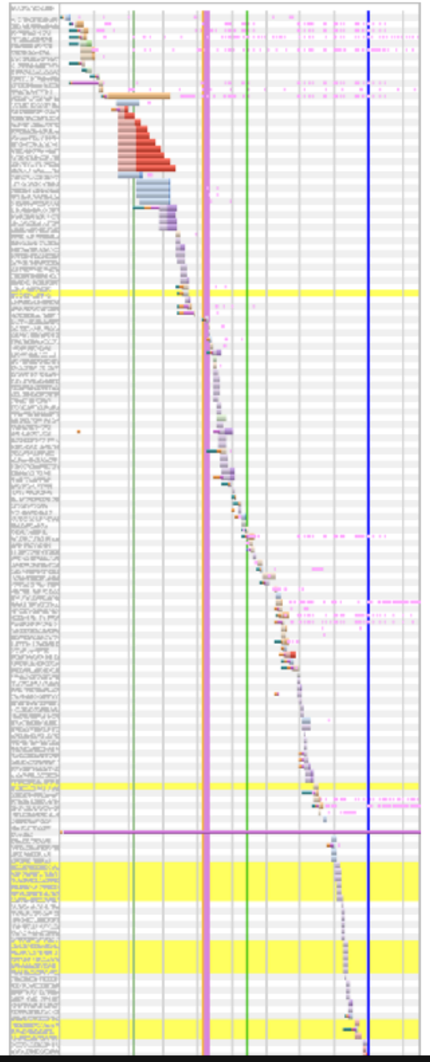


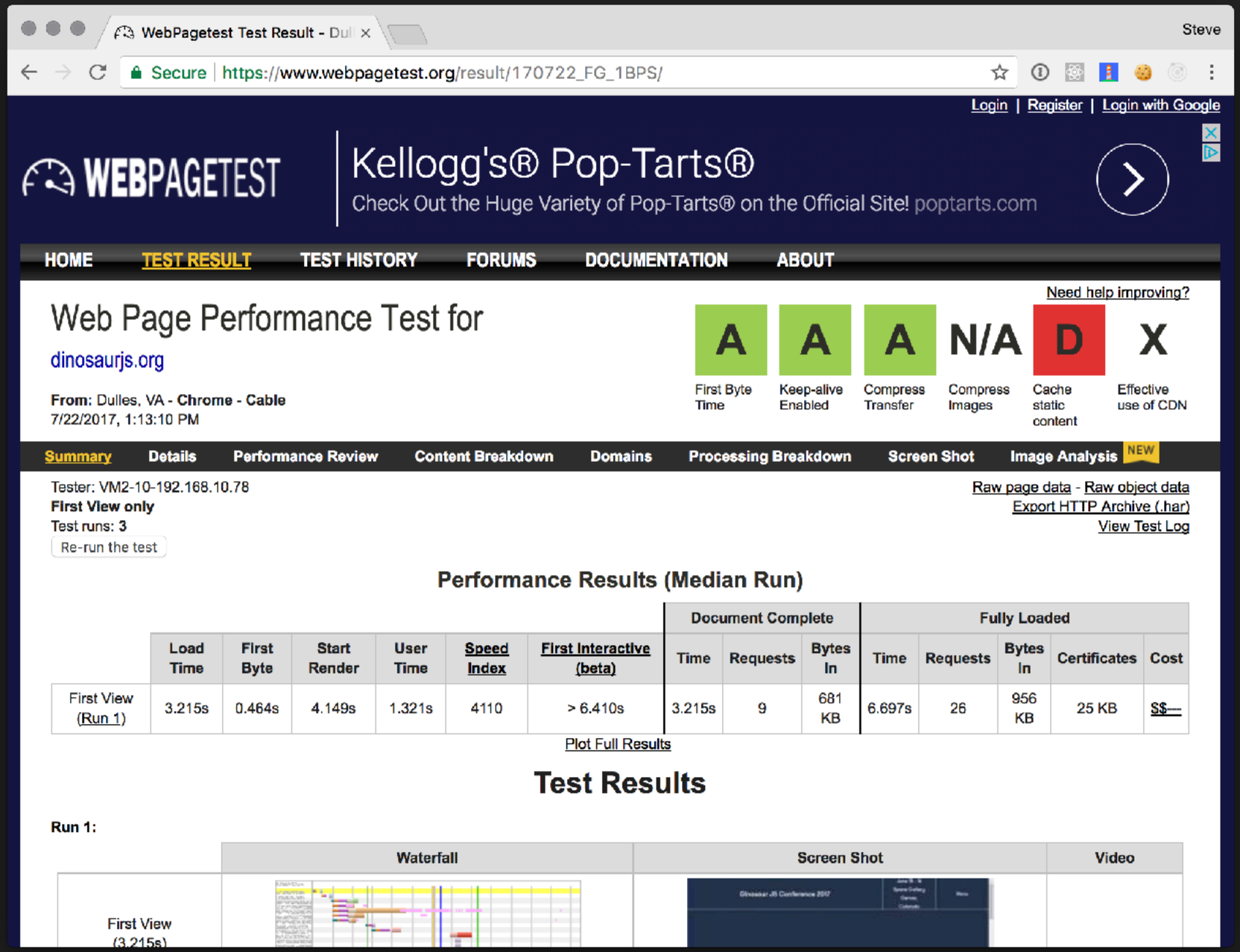
WebPagetest Test Result - Dul

Steve

Secure | https://www.webpagetest.org/result/170722_3C_1BM4/

Run 2:

	Waterfall	Screen Shot	Video
First View (8.989s) Timeline (view) Processing Breakdown Trace (view)			Filmstrip View - Watch Video



High Level Advice

Performance Results

Baseline

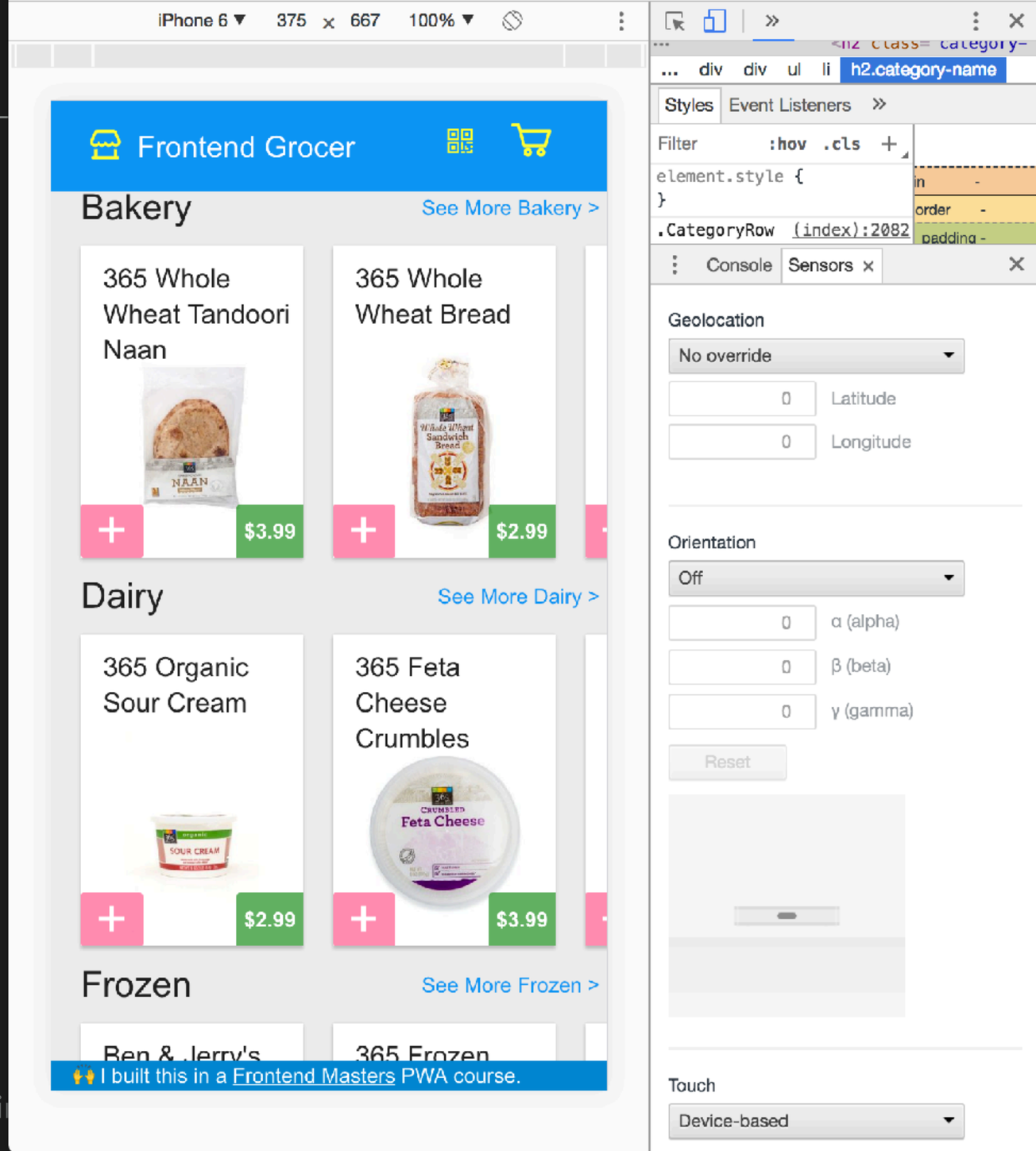
0

- Let's measure our app using Lighthouse
- Make sure you've read through README.md
- Make a production-optimized build of the app `npm run build:prod`
- Bundle analysis `ANALYZE=true npm run build:prod`
- Lighthouse will want you to run the app in HTTP/2 mode (we'll talk about what this means later) `./run serve --http2`
- **Things to note:** lighthouse score, gzipped app size, time to first paint, time to interactive

Mobile Simulation

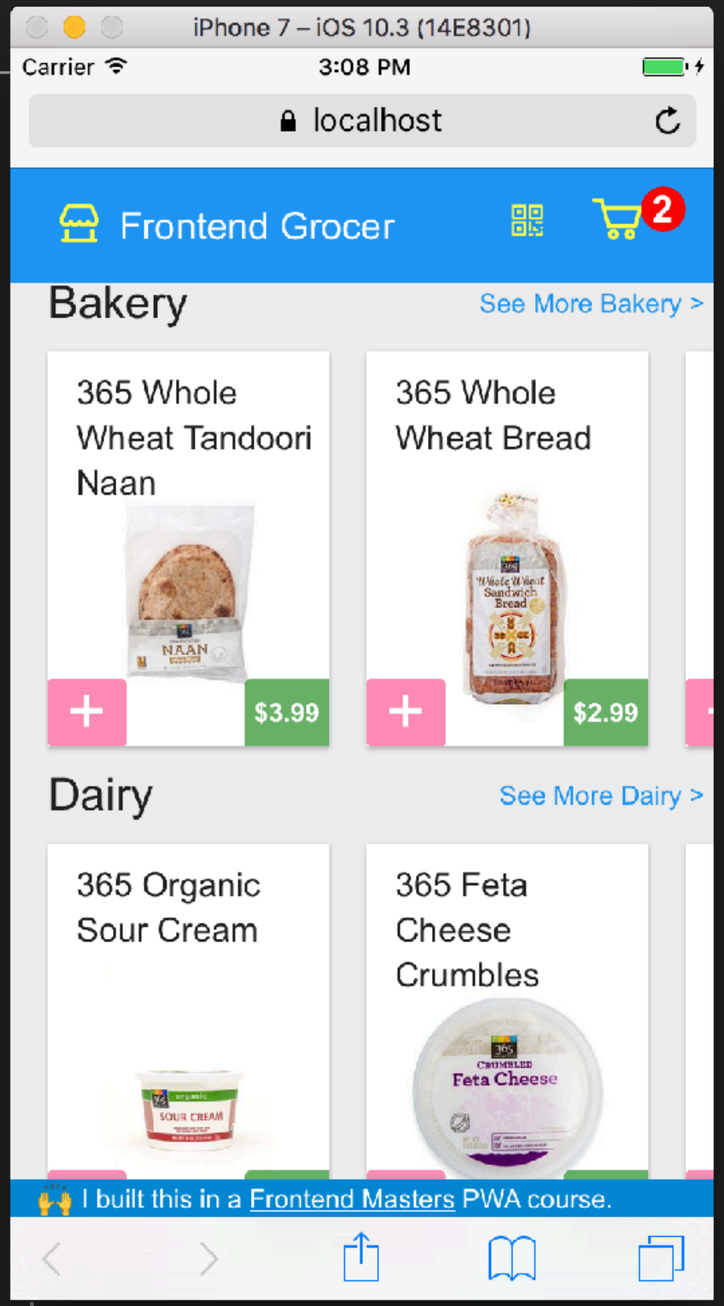
DevTools - Devices Mode

- Simulate device web experiences
- Simulate sensor input
- Closer to Android than iOS
- Responsive breakpoint visualization



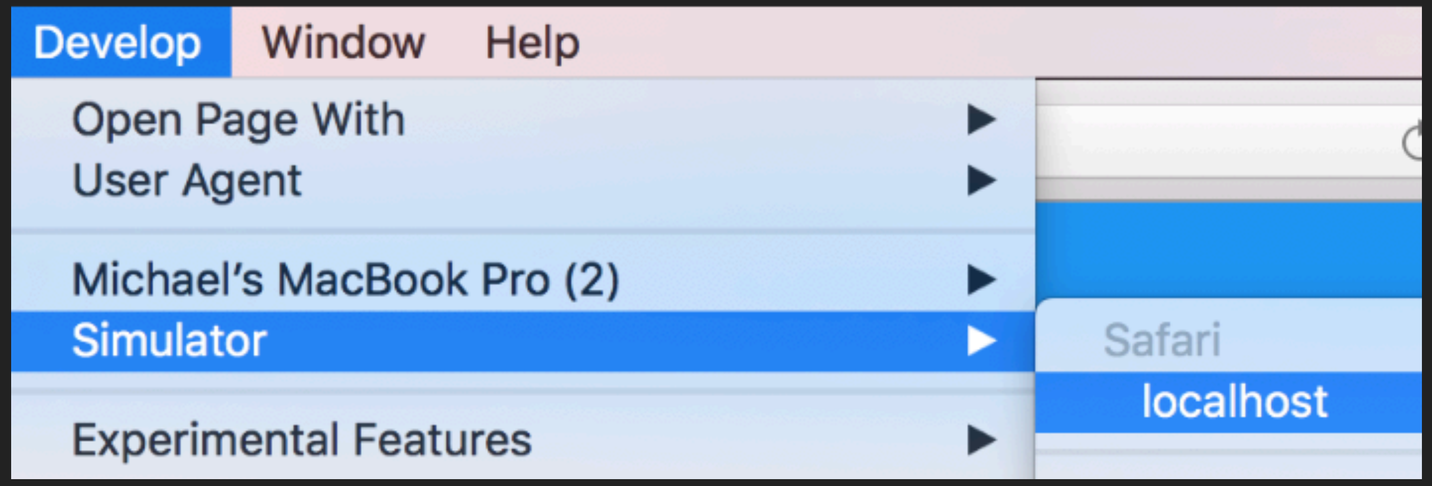
iOS Simulator

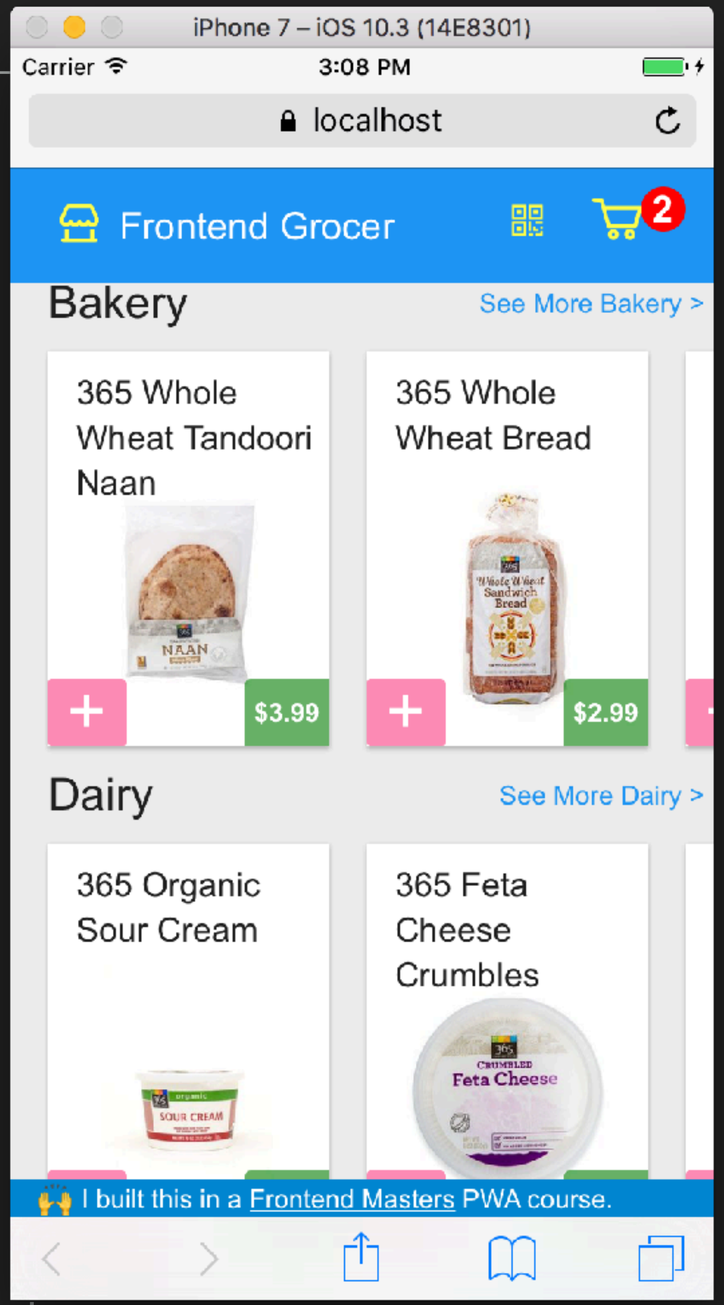
- Localhost is your machine
- Critical for testing apple-specific features



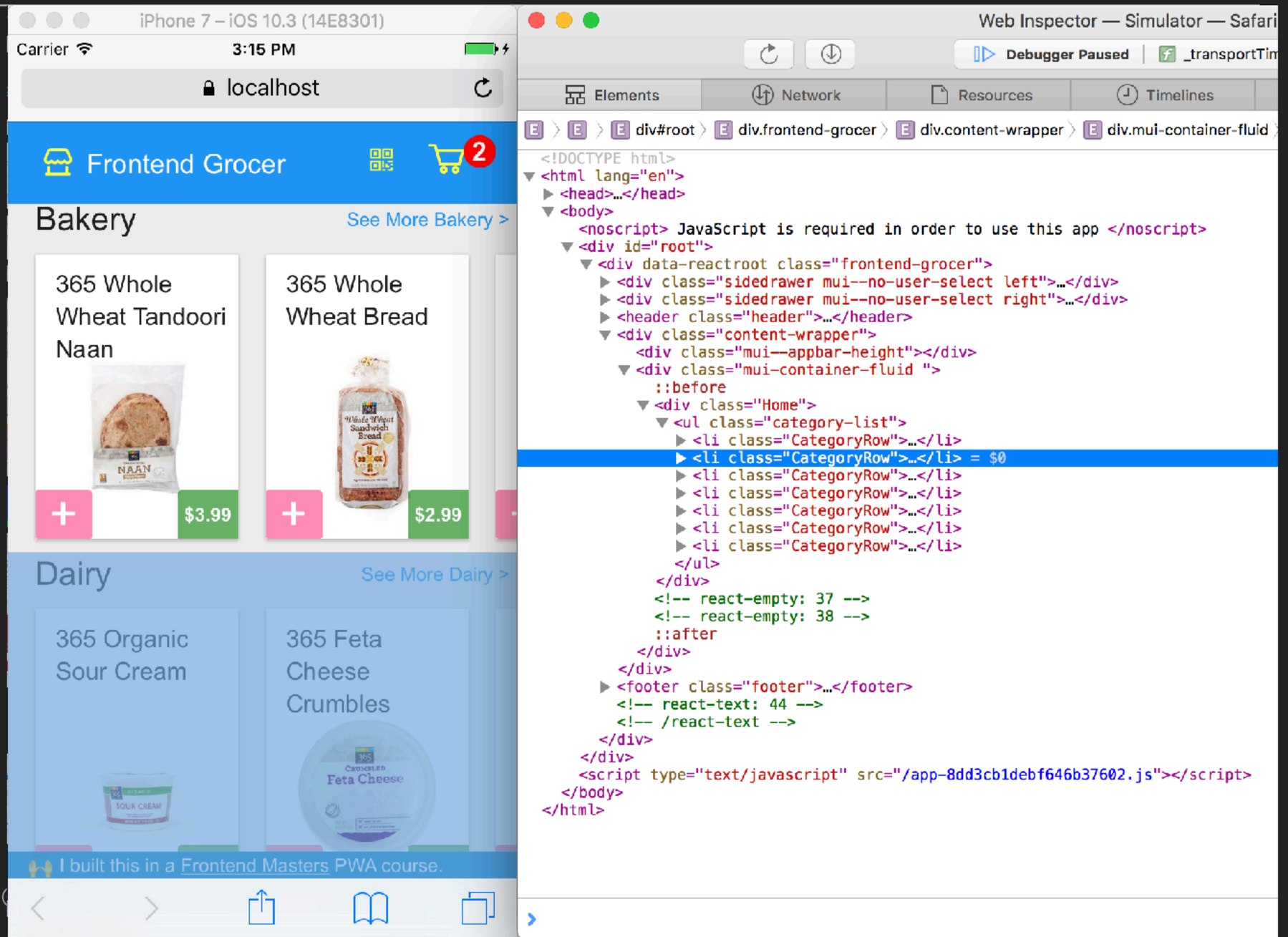
iOS Simulator

- Localhost is your machine
- Critical for testing apple-specific features
- Connect to Safari



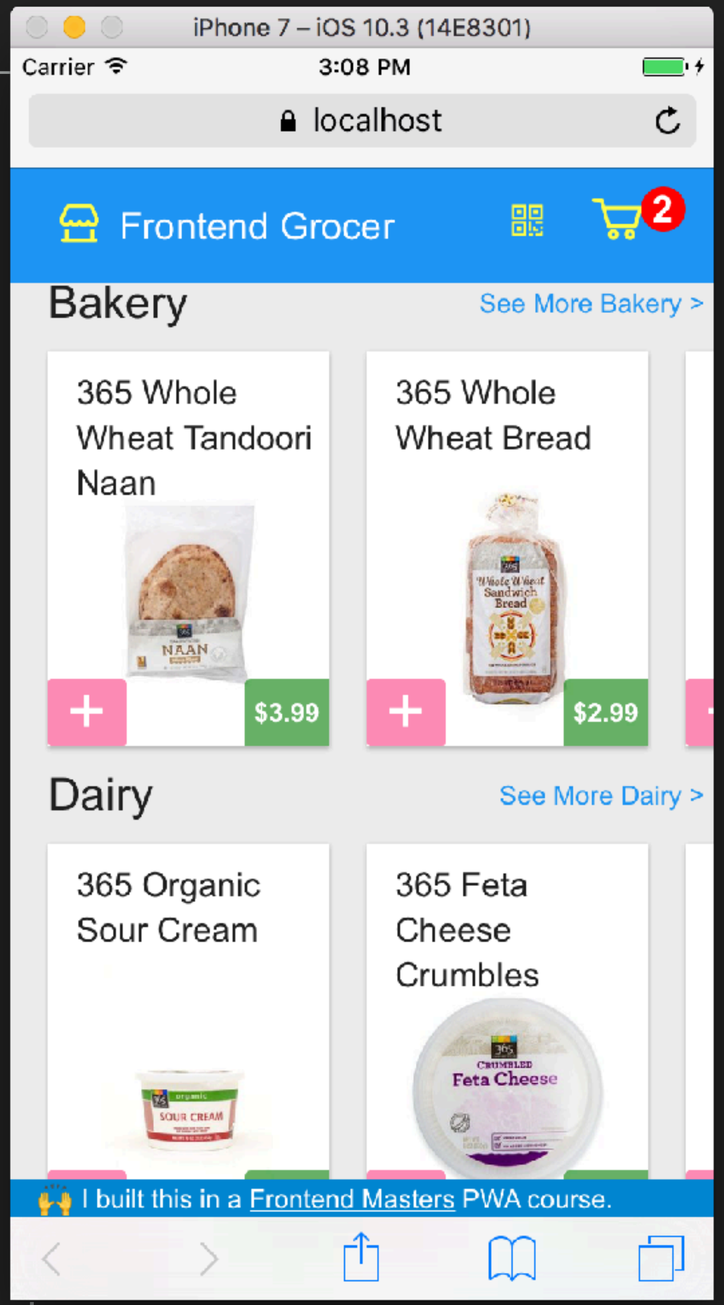


iOS Simulator



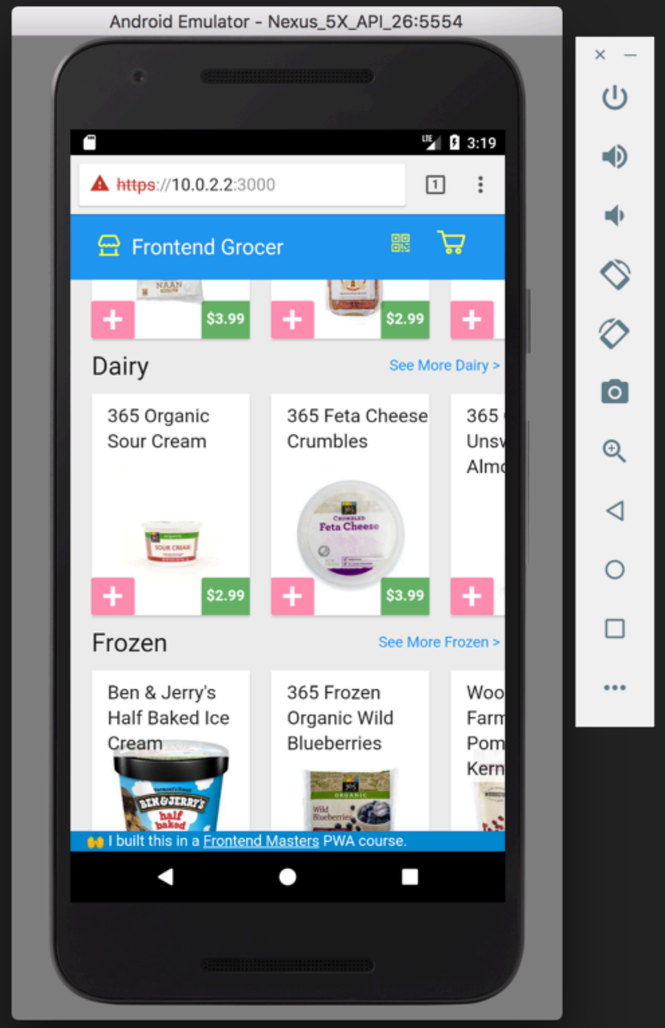
iOS Simulator

- Localhost is your machine
- Critical for testing Apple-specific features
- Connect to Safari
- Useful for validating against TWO iOS JavaScript engines
- Mobile Safari !== Desktop Safari



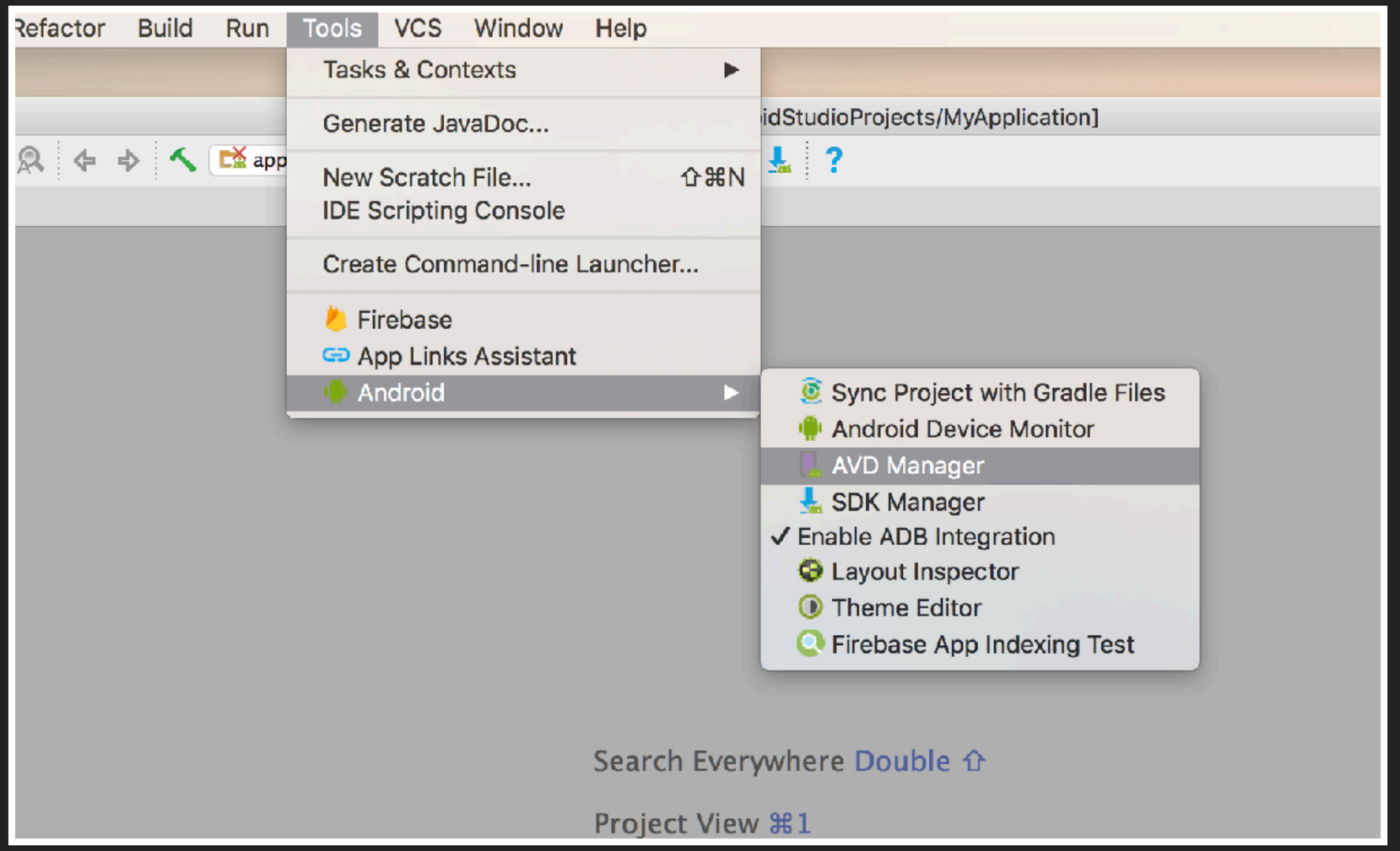
Android Simulator

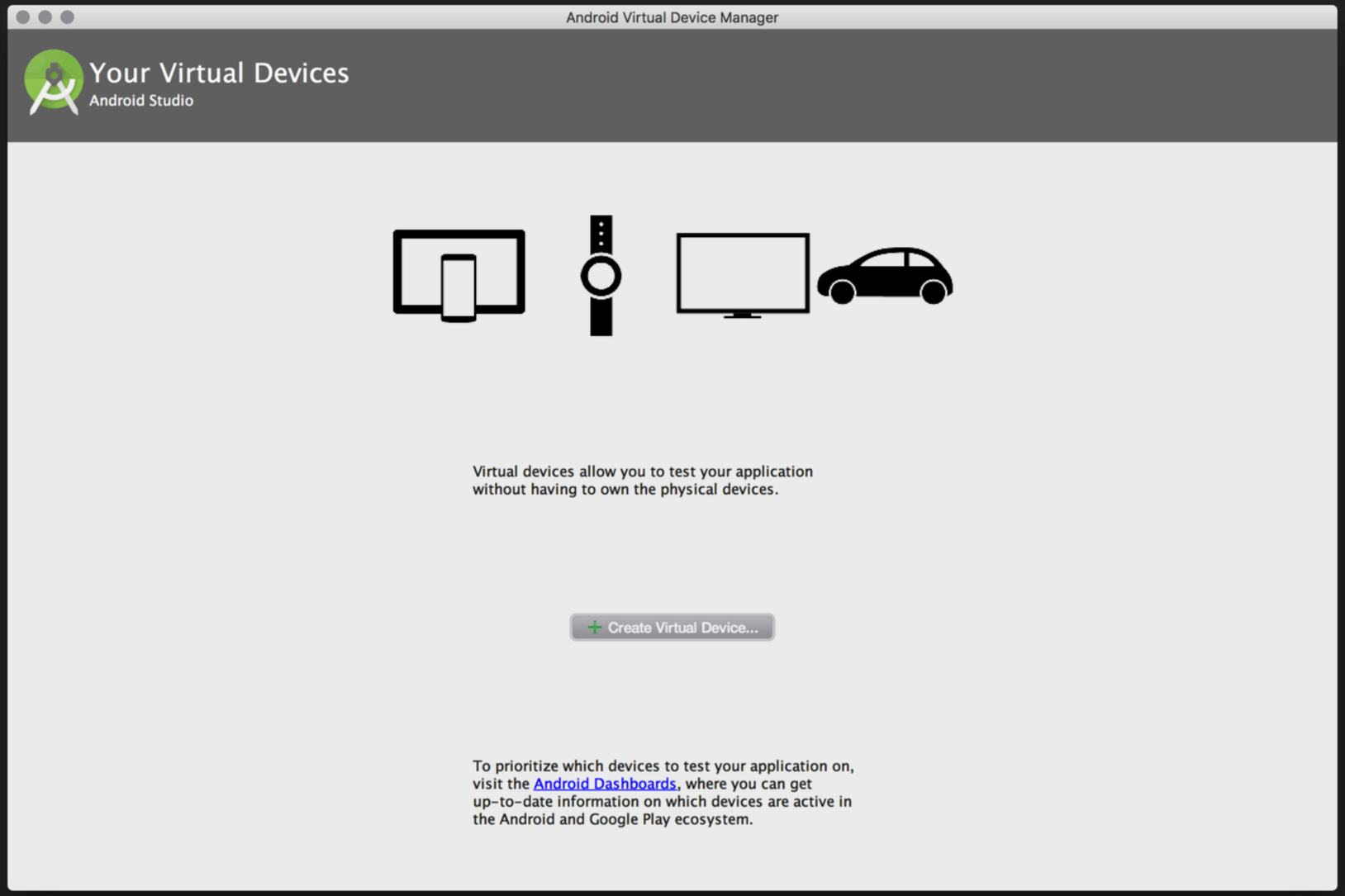
- 10.0.2.2 is your machine
- The simulator can be connected to Chrome's developer tools

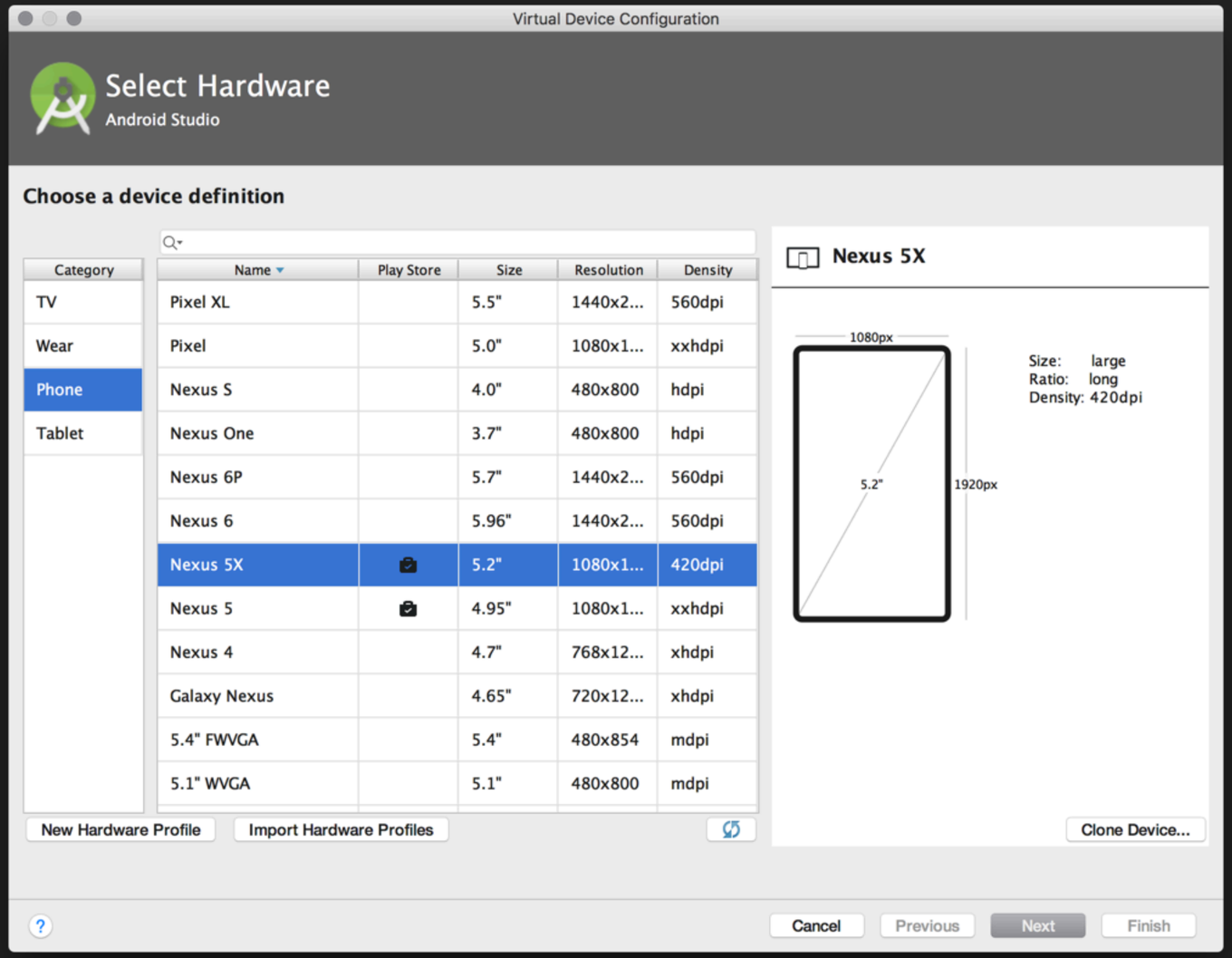


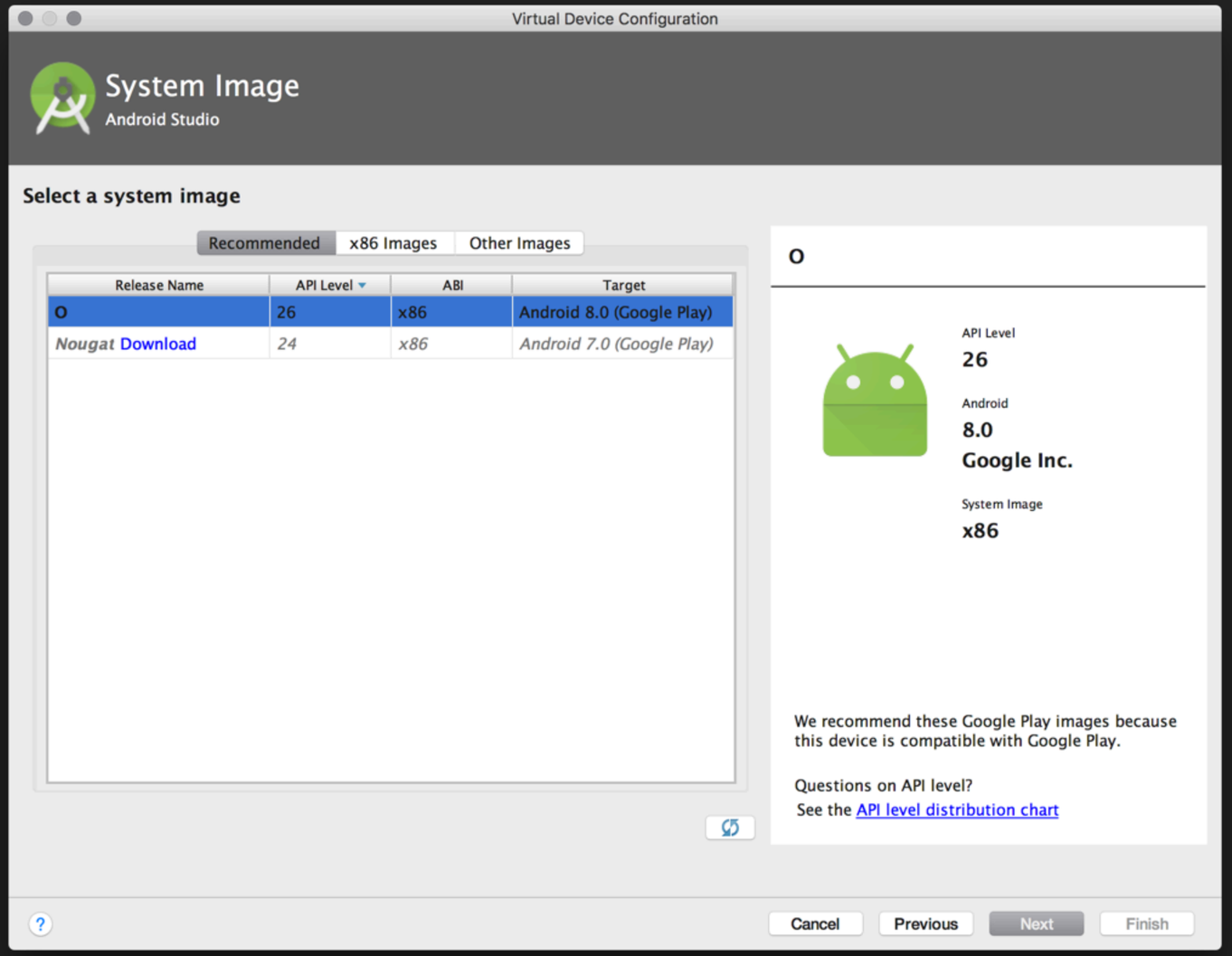
Installing the Android Emulator

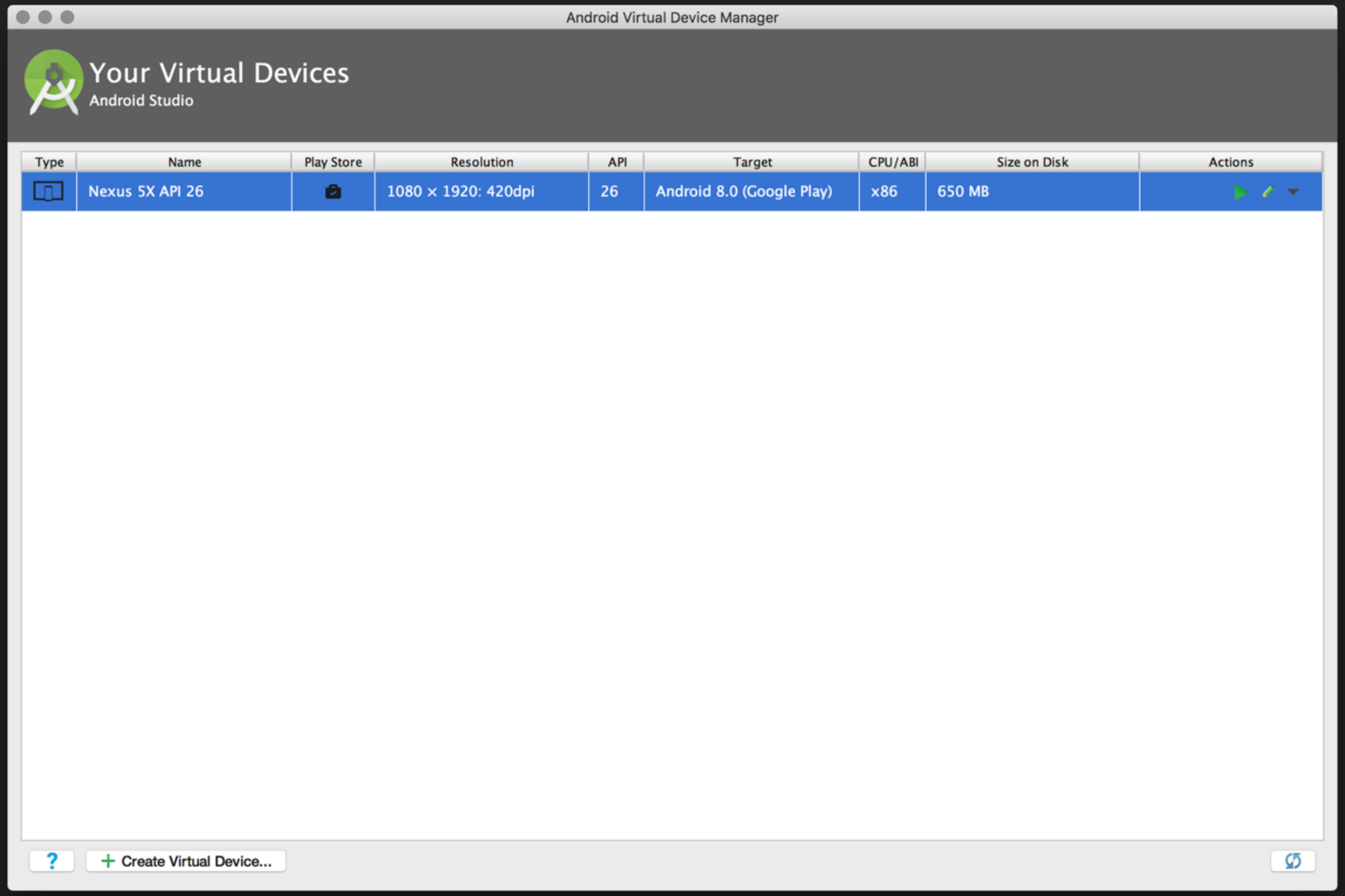
- Download Android Studio (https://developer.android.com/studio/install.html)
- Create a virtual device
- Connect Chrome to the virtual device by visiting chrome://inspect/#devices

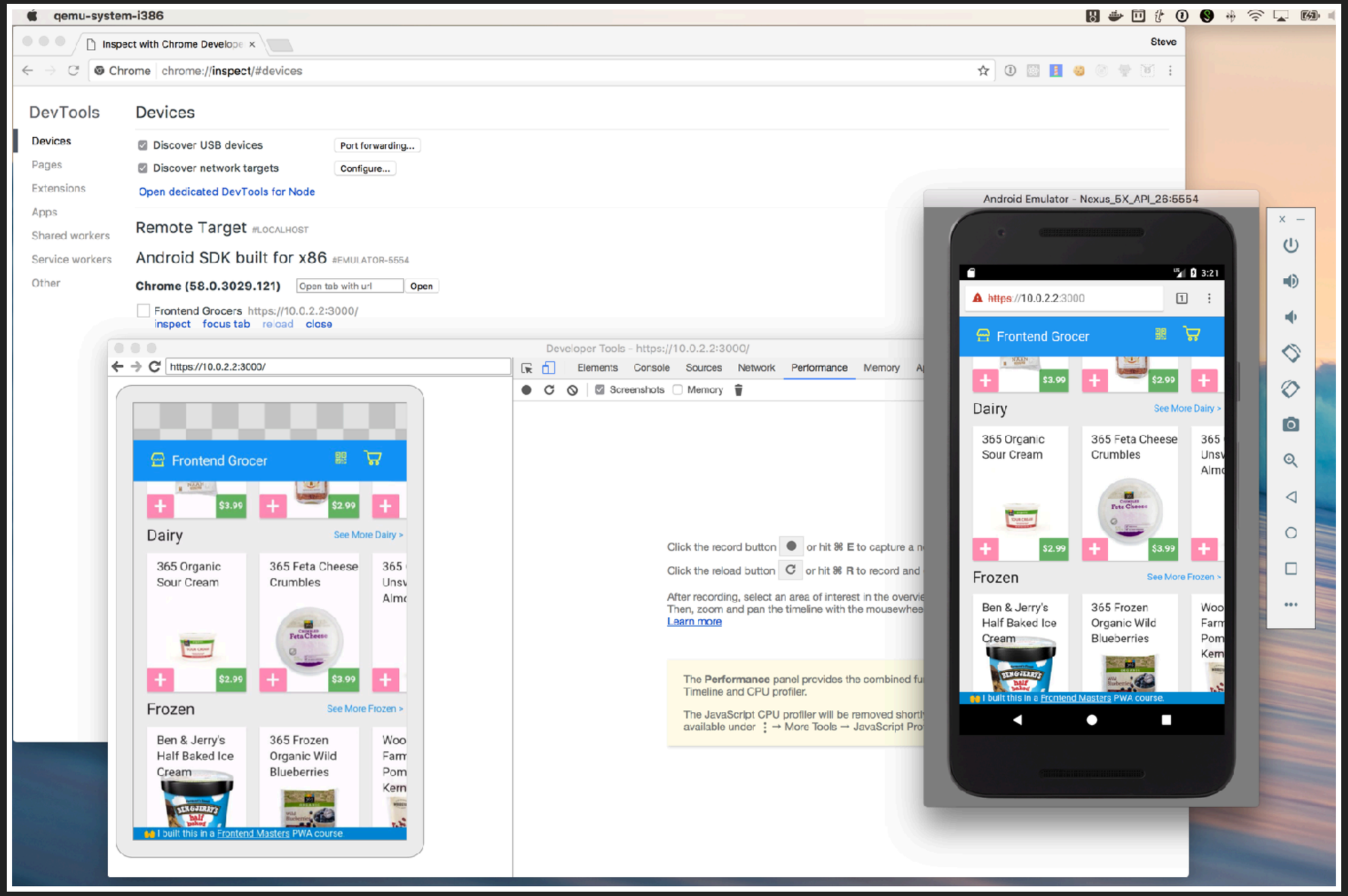












Progressive Metadata

Viewport

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

width	positive integer or ‘device-width‘
initial-scale	positive number between 1.0 and 10.0
maximum-scale	positive number between 1.0 and 10.0
minimum-scale	positive number between 1.0 and 10.0
user-scalable	yes or no

Fullscreen

```
<meta name="apple-mobile-web-app-capable" content="yes">
```

Black Status Bar

```
<meta name="apple-mobile-web-app-status-bar-style" content="black">
```

Home Screen Title

```
<meta name="apple-mobile-web-app-title" content="Frontend Masters">
```

manifest.json

```
<link rel="manifest" href="/manifest.json">
```

```
{
  "name": "Frontend Masters",     // App name
  "icons":[],                     // Various sizes of icon
  "theme_color": "#2d89ef",       // Title bar styling
  "background_color": "#2d89ef", // Used for startup screen
  "display": "standalone"         // Launch as an app
}
```

manifest.json

```
{
  "name": "Frontend Masters",     // App name
  "icons":[                       // Various sizes of icon

  ],
  "theme_color": "#2d89ef",       // Title bar styling
```

manifest.json

```
{
  "name": "Frontend Masters",      // App name
  "icons":[],                      // Various sizes of icon
  "theme_color": "#2d89ef",        // Title bar styling
  "background_color": "#2d89ef", // Used for startup screen
  "display": "standalone"          // Launch as an app
}
```

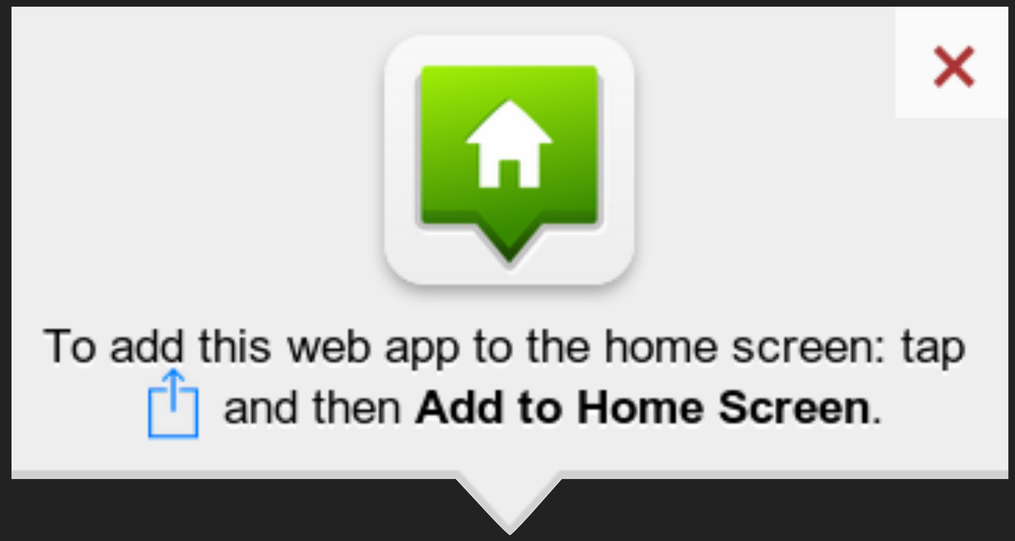
manifest.json - display

Display Mode	Description
`fullscreen`	All available display area is used
`standalone`	Look & feel like a standalone app
`minimal-ui`	“light browser” UI. Doesn’t have own window
`browser`	Conventional web app in a browser

Home Screen Icons

Size	Device
57	iPhone 1, 2
72	iPad 1, 2
114	iPhone 4 (Retina)
120	iPhone 6, 7, SE
144	iPad 3 (Retina)
152	iPad Air, Air 2, Mini
167	iPad Pro
180	iPhone 6+, 7+
128	Android Regular
192	Android High Res

```
<link href="https://placehold.it/152"
      sizes="152×152"
      rel="apple-touch-icon">
```



Home Screen Icons

https://github.com/cubiq/add-to-homescreen

```
addToHomescreen({
  appID: 'pwa.fundamentals', // local storage name (no need to change)
  message: 'Add to home?', // the message can be customized
  modal: false, // block UI until the message is closed
  mandatory: false,  // can't proceed if you don't add to homescreen
});
```

schema.org Metadata

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Product",
  "image": "dell-30in-lcd.jpg",
  "name": "Dell UltraSharp 30\" LCD Monitor",
  "offers": {
    "@type": "Offer",
    "price": "$1495"
  }
}
</script>
```

Mobile Metadata

1

Make the following enhancements to our app…

- Add to home screen
- "FEGrocer" should be used as the app name
- Home screen icon
- Remove browser UI (address bar, back button, etc…)
- Theme color is same as app-bar color
- Icons for 192, 48 and 96px
- start_url is "."
- Viewport suitable for mobile devices, and "app-like"

Mobile Metadata

1

- index.html is generated from `./client/index.ejs`
- We can get webpack to generate a file for us via file-loader

```
import 'file-loader?name=web-app-manifest.json!./web-app-manifest.json';
```

put this line at the top of ./client/index.js or ./client/app.jsx

- Use iOS simulator or at least "Device Mode" in a browser to verify your improvements
- BONUS: schema.org Product metadata for grocery items (involves updating `./client/components/grocery-item/index.jsx`)

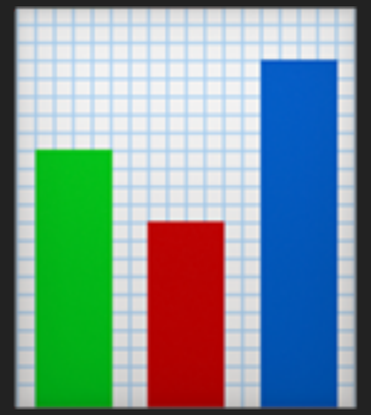
Server-Side Rendering

Important Metrics

Time to...

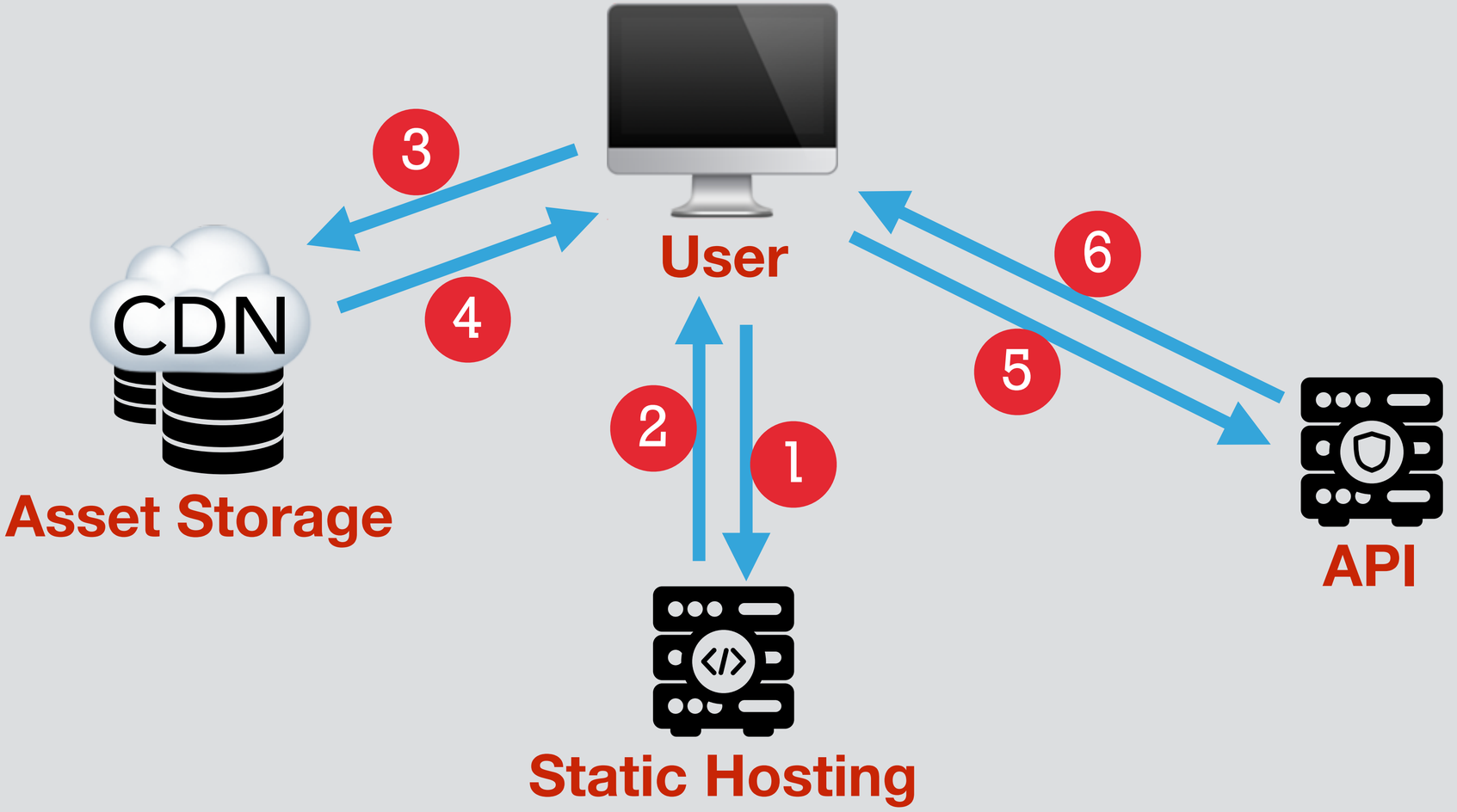
First Paint

Interactive



Data

Client Rendered SPA



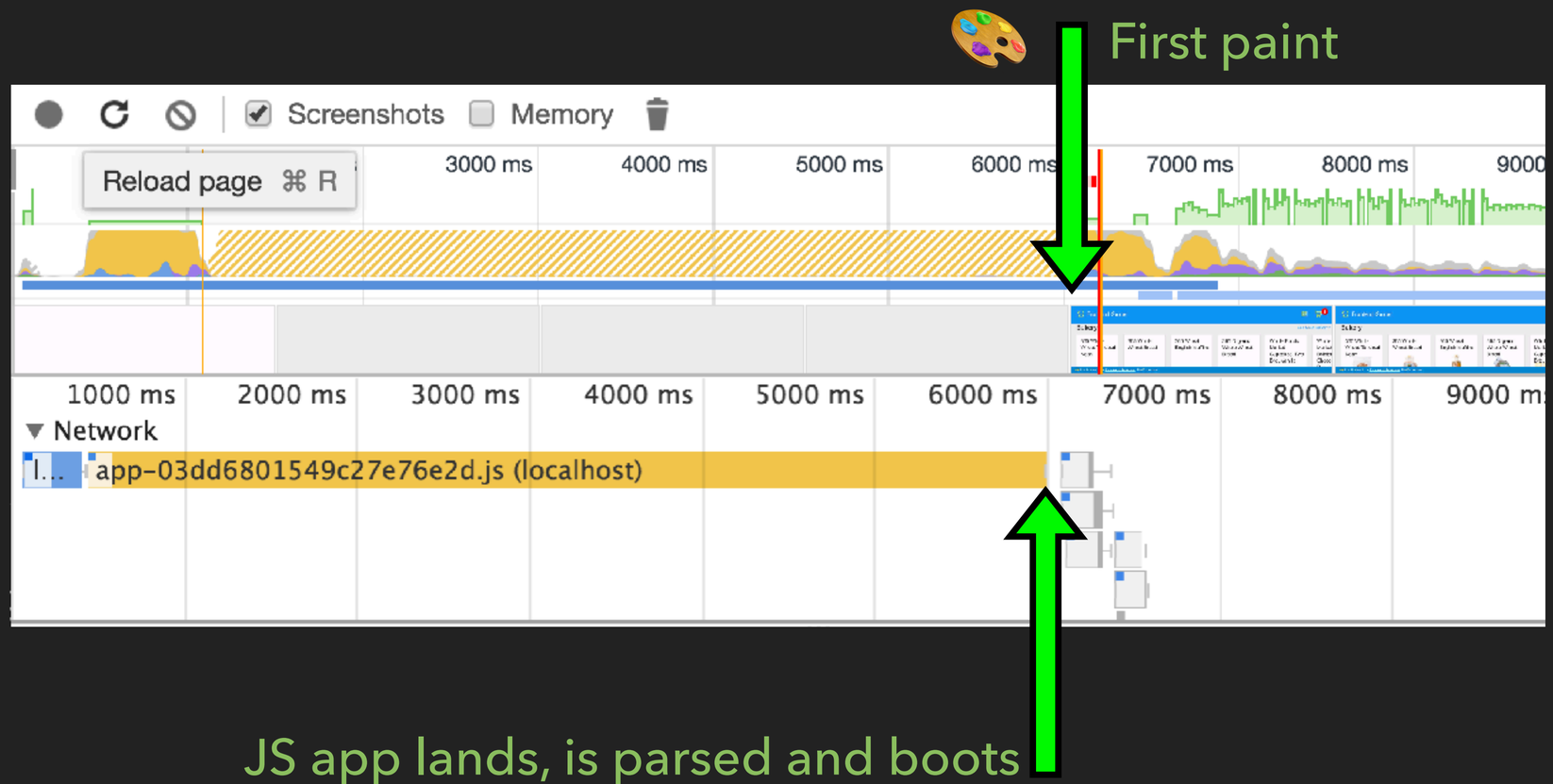
First Paint

Interactive

Data

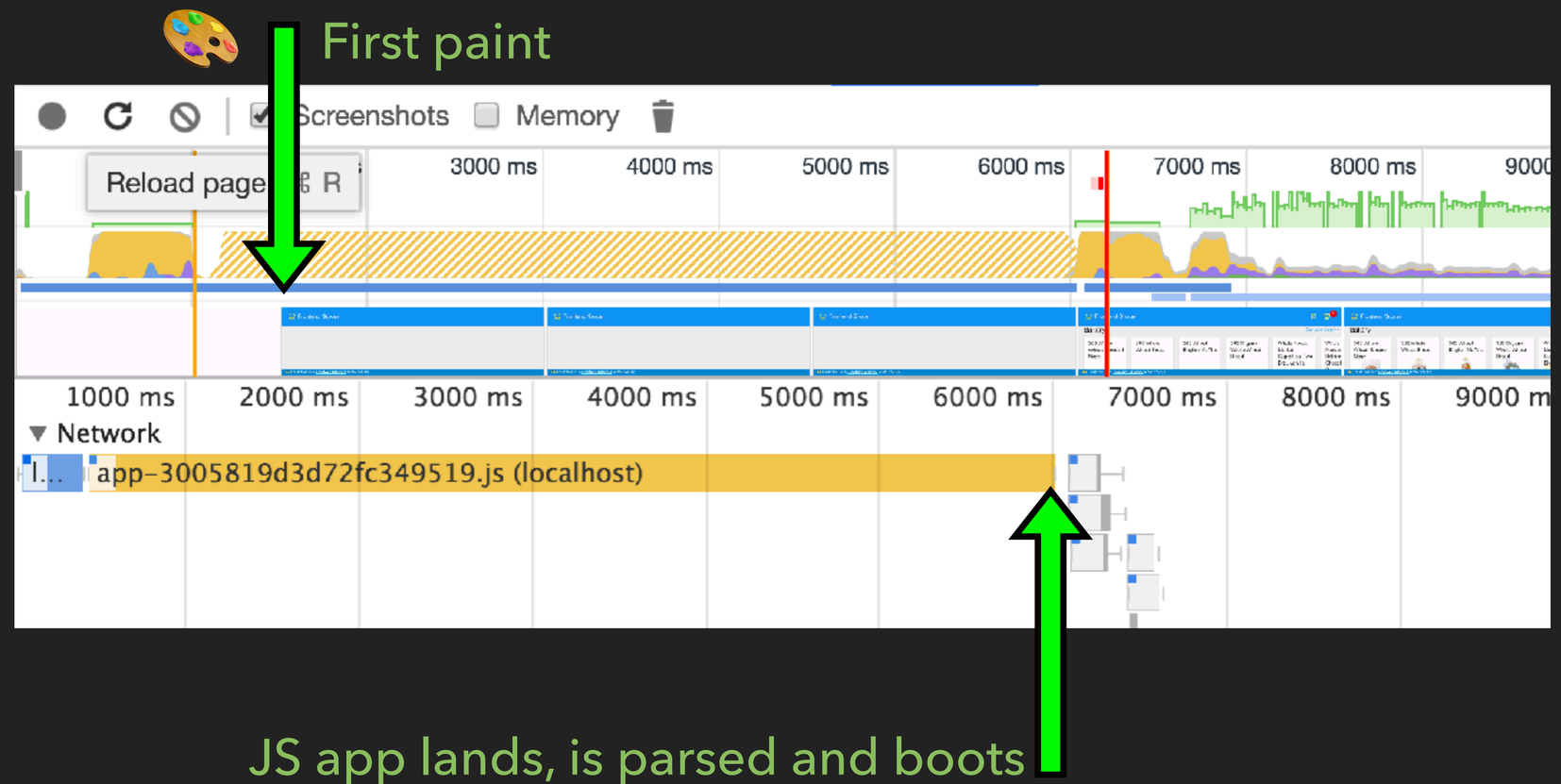
Client-Side Rendering: Perf Timeline

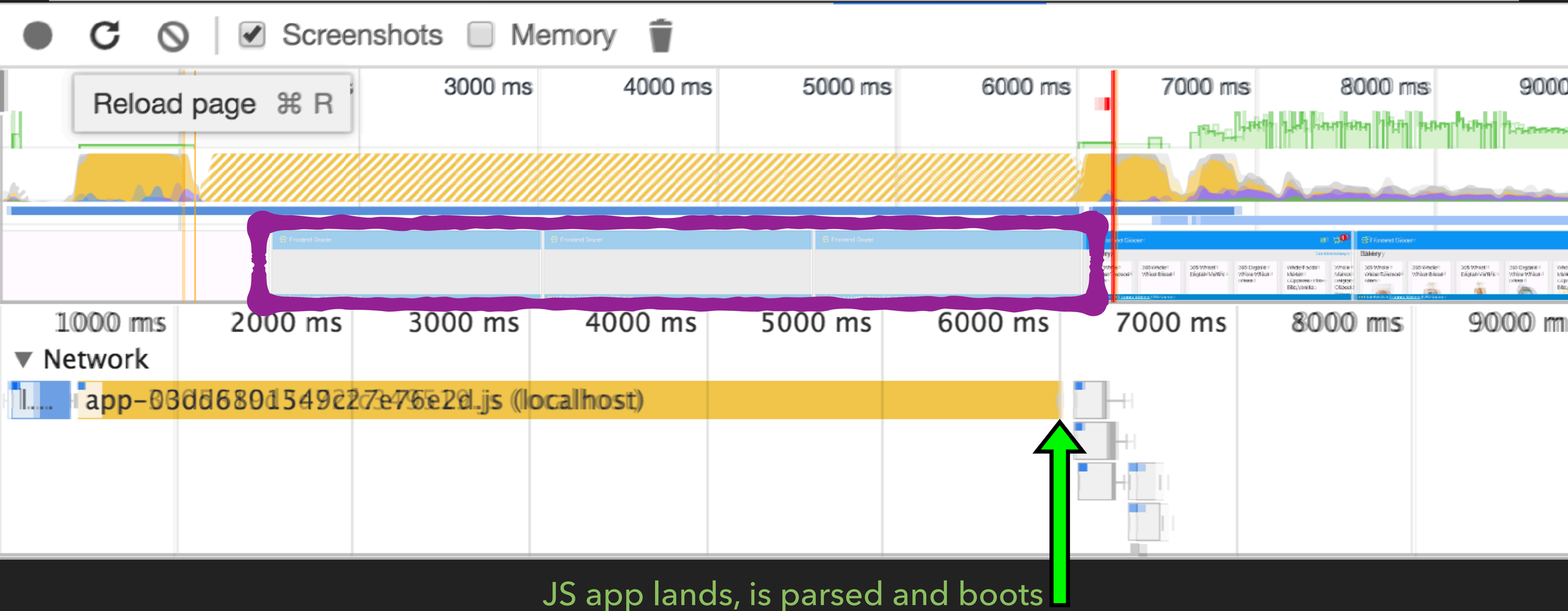
- Pure client-side rendering requires JS to land before anything interesting can happen



Enhanced Client-Side Rendering: Perf Timeline

- General solution for SS rendering is HARD
- Browser JS !== Node
- Sustainable and quick middleground for many apps

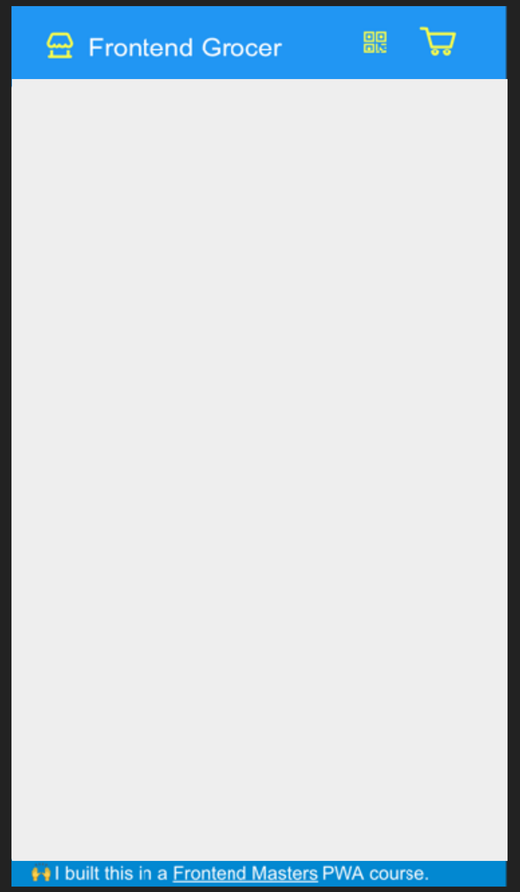




JS app lands, is parsed and boots

Enhanced Client-Side Rendering

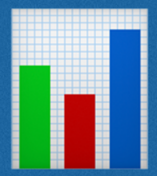
- Works for a wide range of apps with some URL-agnostic high-level UI
- Three steps:
  1. Boot the app with URL-agnostic HTML
  2. Embed critical CSS in your index.html
  3. Add a loading spinner



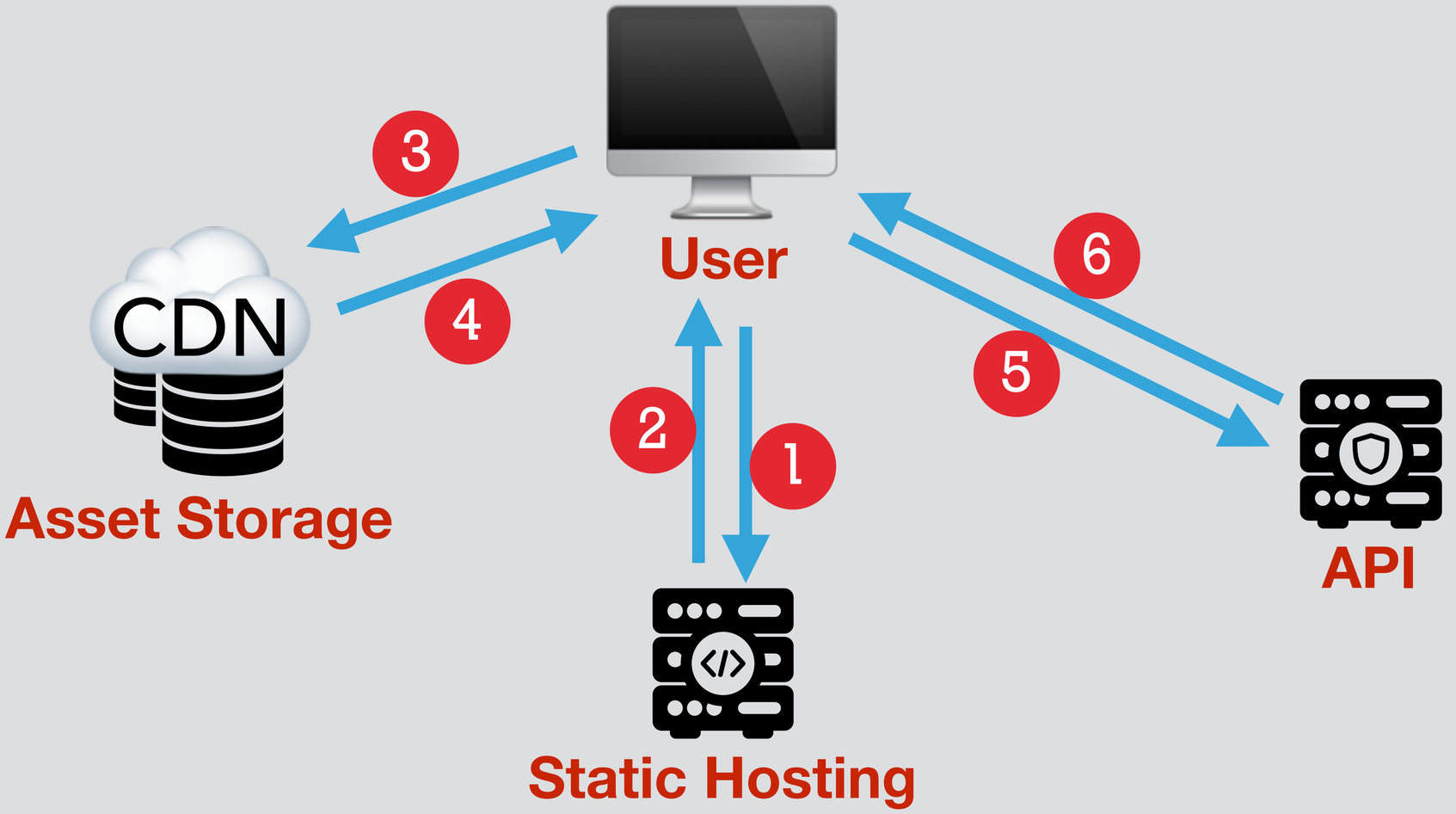
Enhanced Client Rendered SPA

First Paint

Interactive



Data



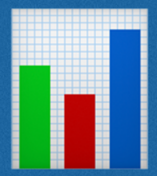
Server-Side Rendering

- Dynamic HTML responses on a per-request basis
- Involves special treatment of browser-only concepts
  - XMLHttpRequest
  - window, navigator, sensors, localstorage, etc...
- Initial HTML response is slowed down...
- ...but when it arrives, URL-specific content is ready!

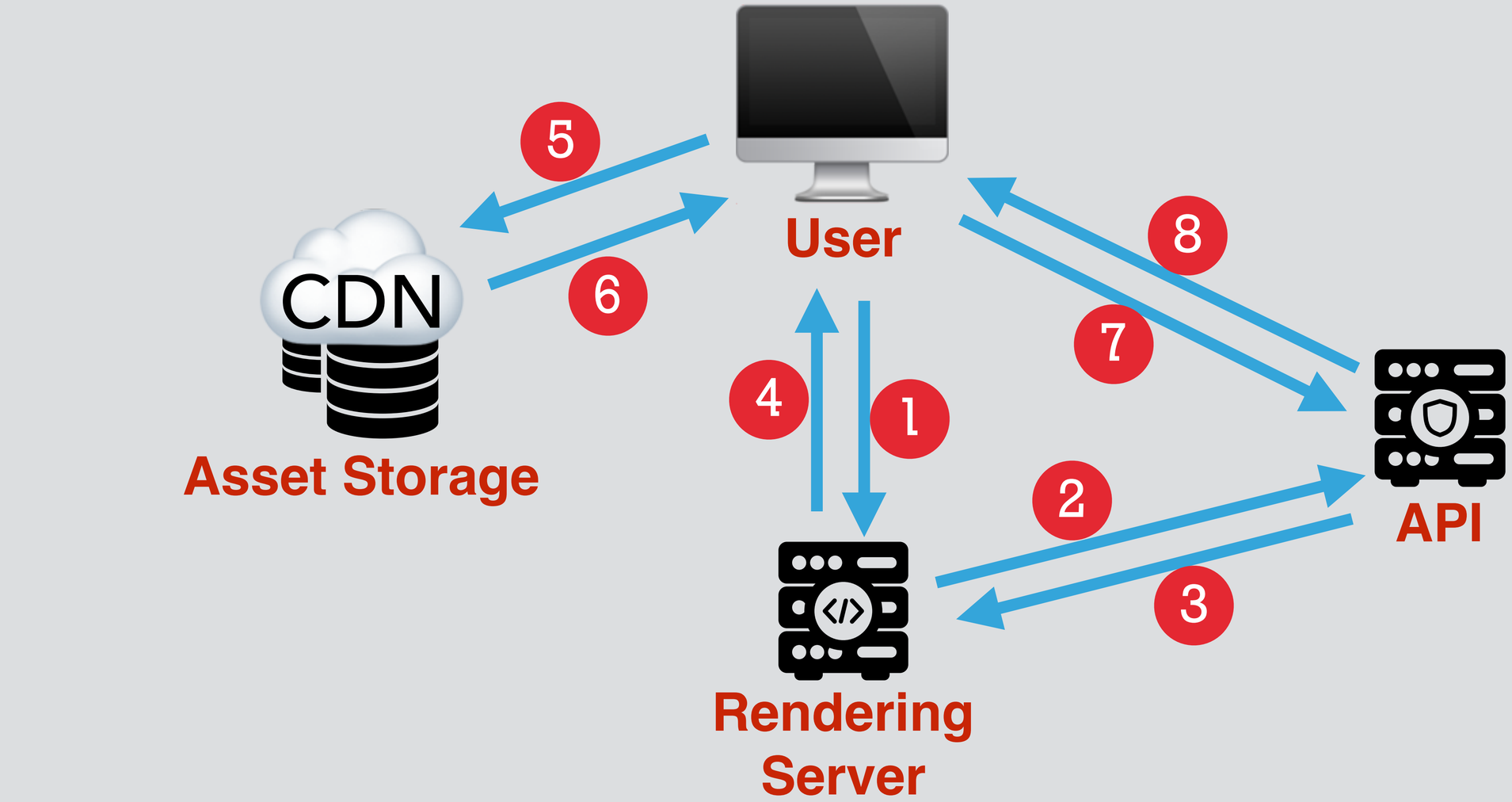
Server Rendered SPA

First Paint

Interactive



Data

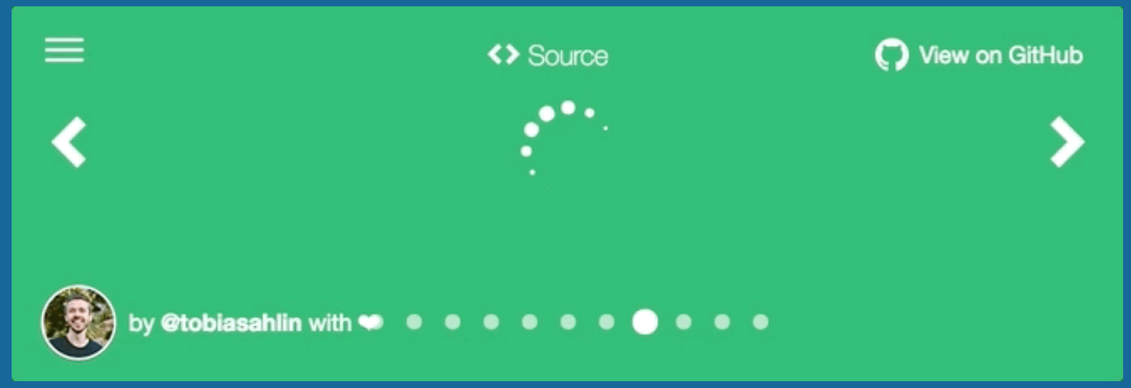


Don't boot blank!

2

A generic solution to server-side rendering is extremely complicated, particularly when combined with other performance-optimization techniques. However, for apps that look like ours, there's a little cheat!

- Identify HTML that's **always** rendered, regardless of URL
- Update `./client/index.ejs` so that we start with that HTML instead of a blank page
- BONUS: add a CSS spinner from http://tobiasahlin.com/spinkit/ and put css in `./client/index.scss`

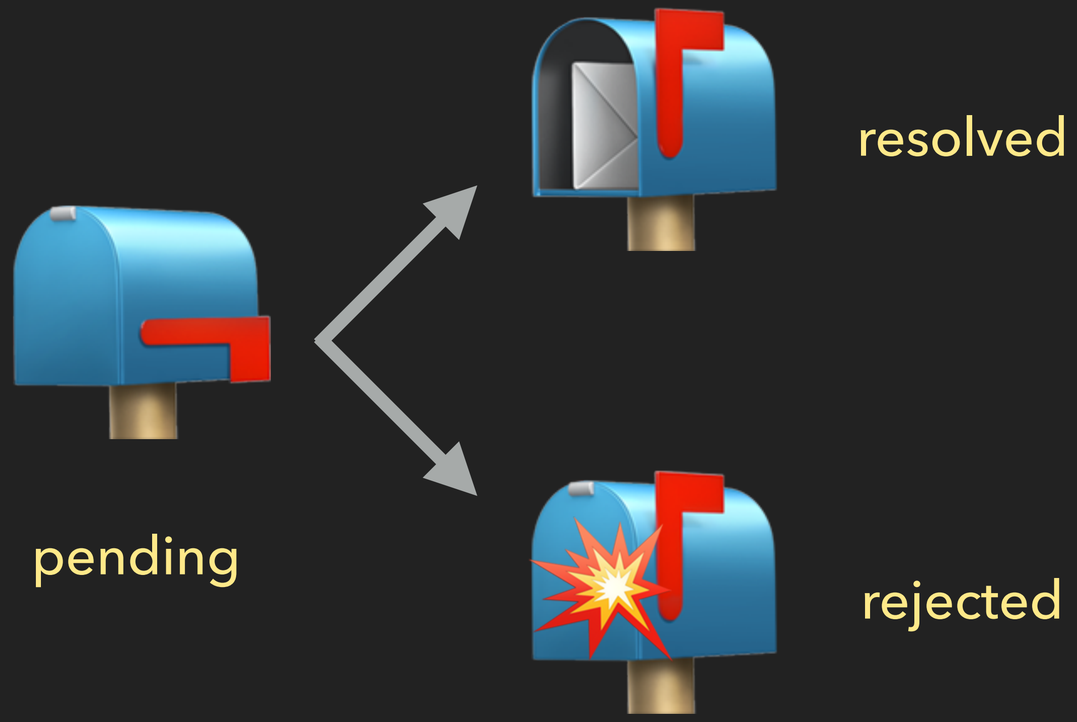


Promises

A Gentle (Re-)Introduction to Promises

- Promises can be thought of as an eventual value.
- They usually involve something asynchronous, that...
  - May or may not be "done"
  - If "done", may have either arrived at its resolved value, or encountered an error along the way
- You've probably already used promises (i.e., `$.getJSON()`)
- Promises are better than callbacks, and appear in PWAs all over the place

Promises - Mental Model



Promises - Basic Use

- We can listen for the eventual value by passing a callback to the `then` function
- We can listen for an error using the `catch` function
- There's a proposal for adding a `finally` function, which would be called in either case
- These are invoked immediately if the promise has already settled

```
let p = $.getJSON(url); // 📪

p.then((value) ⇒ { // 📪
  console.log('value is ', value);
});

p.catch((err) ⇒ { // 💥
  console.error(err);
});

p.finally(() ⇒ { // 📪 or 💥
  console.log('Done!');
});
```

Some Notes on Chaining

- Anything returned from a `.then()` is wrapped in a promise.
- This means that you can call `.then()` on that return value as well.
- It's turtles down the line.

kitchen-async.js — Desktop

```
1  const request = $.getJSON('/api/endpoint');
2
3  request.then(data => console.log(data));
4
```

Line: 3:41 | JavaScript | Soft Tabs: 2

kitchen-async.js — Desktop

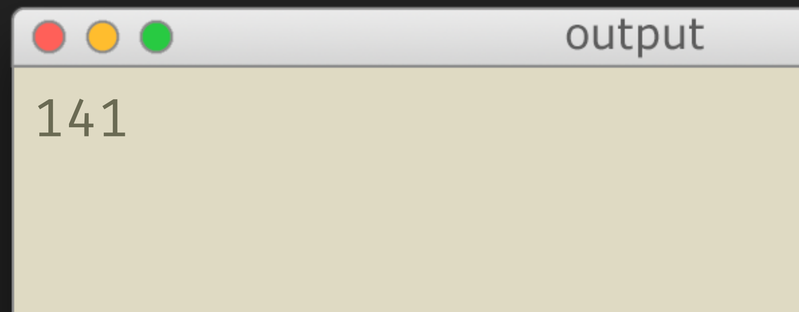
```
1 const request = $.getJSON('/api/endpoint');
2
3 request.then(data => console.log(data));
4
5 request.then(data => alert(data));
6
```

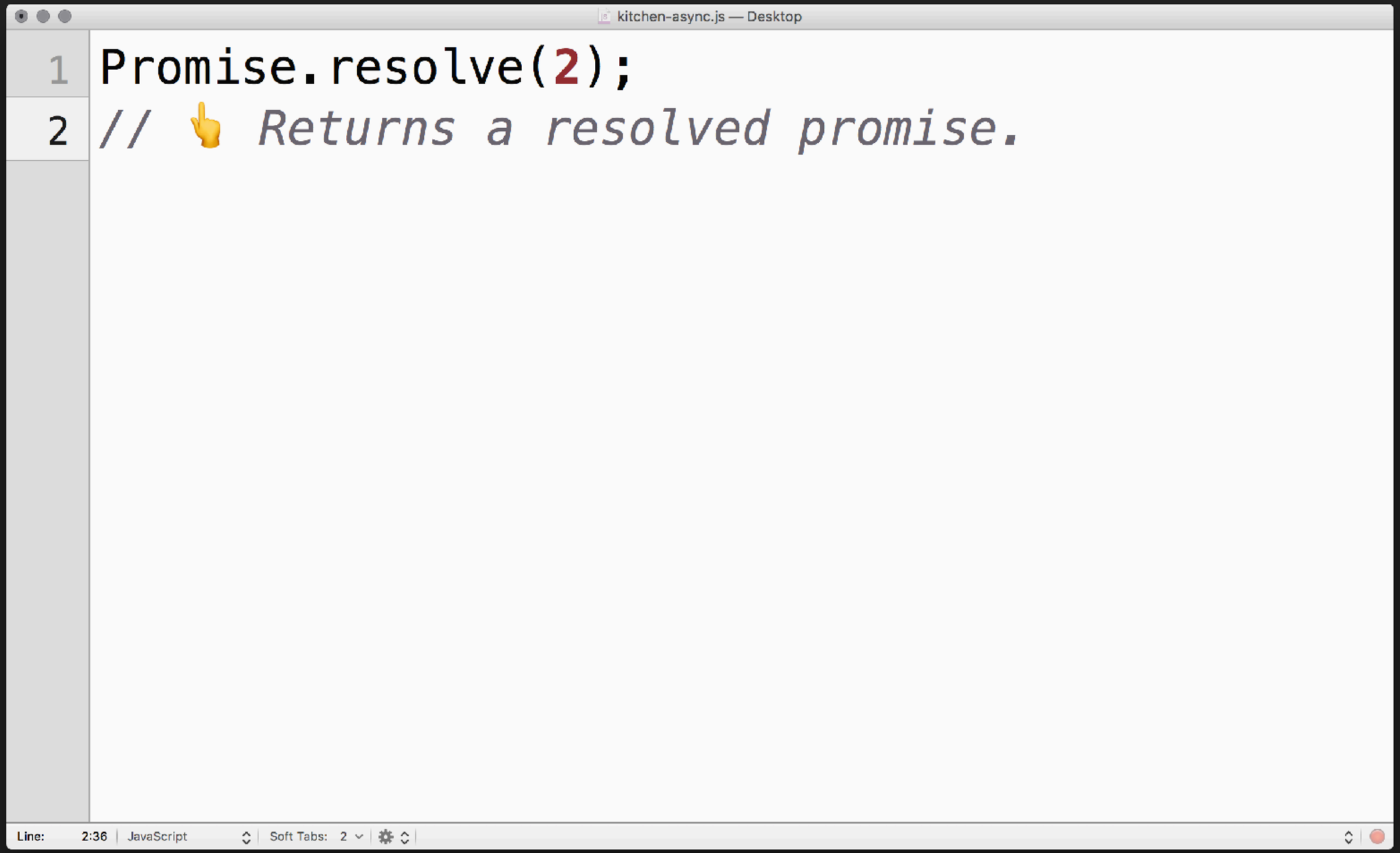
Line: 5:27 | JavaScript | Soft Tabs: 2

Promises - Chaining

- The `then` function returns a new promise, that resolves to whatever the callback returns.
- This allows us to chain multiple `then` functions together
- If the callbacks return a promise, the resolved value is passed down the chain (not the promise its self)
- Stick to a linear chain

```
let p = new Promise(
  (resolve) => resolve(31)
);

p.then((num) => num + 10)
 .then(
   (num) => Promise.resolve(num + 100)
 )
 .then((num) => {
   console.log(num);
 });
```





kitchen-async.js — Desktop

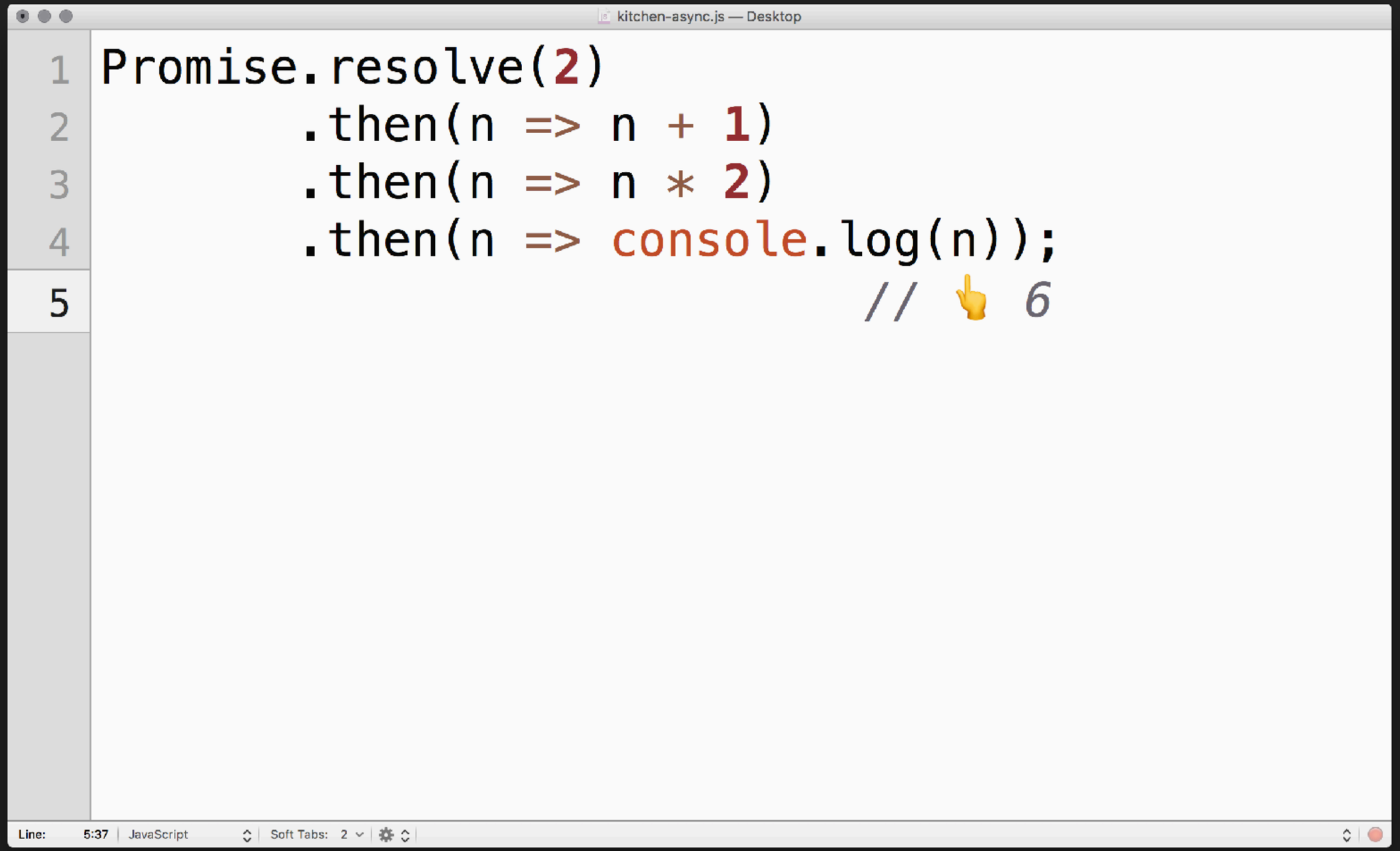
```
1  Promise.resolve(2)
2         .then(n => console.log(n));
3                                  // 👆 2
```

Line: 3:28 | JavaScript | Soft Tabs: 2

kitchen-async.js — Desktop

```
1  Promise.resolve(2)
2         .then(n => n + 1)
3         .then(n => console.log(n));
4                                  // 👆 3
```

Line: 4:37 | JavaScript | Soft Tabs: 2



kitchen-async.js — Desktop

```
1  Promise.resolve(2)
2         .then(n => n + 1)
3         .then(n => n * 2)
4         .then(n => Math.pow(n, 2))
5         .then(n => console.log(n));
6                                 // 👆 36
```

Line: 6:37 | JavaScript | Soft Tabs: 2

kitchen-async.js — Desktop

```
1  const futureNumber = Promise.resolve(2);
2
3  futureNumber
4    .then(n => n + 1)
5    .then(n => n * 2)
6    .then(n => Math.pow(n, 2))
7    .then(n => console.log(n));
8                              // 👆 36
```

Line: 8:33 | JavaScript | Soft Tabs: 2

kitchen-async.js — Desktop

```
1  const futureNumber = Promise.resolve(2);
2
3  futureNumber
4    .then(n => n + 1)
5    .then(n => n * 2)
6    .then(n => Math.pow(n, 2))
7    .then(n => console.log(n));
8                              // 👆 36
9
10 futureNumber.then(n => console.log(n));
11                                 // 👆 2
```

Line: 11:42 | JavaScript | Soft Tabs: 2

Promises - Creation

- Promises can be created via the `Promise` constructor, which takes a callback function as its single argument.
- This callback receives two functions as arguments
  - one to use if the promise resolves successfully (resolves)
  - another to use if it errors (rejects).

```
let p = new Promise(
  (resolve, reject) ⇒ {
    resolve(31);
  }
);
p.then((num) ⇒ {
  console.log(num);
});
```

How ′bout a more practical example?

kitchen-async.js — Desktop

```
1 const request = $.getJSON('/api/endpoint');
```

Line: 1:44 | JavaScript | Soft Tabs: 2

kitchen-async.js — Desktop

```
1  const request = $.getJSON('/api/endpoint');
2
3  const renderPost = (post) => {
4    return `
5      <article>
6        <h2>${post.title}</h2>
7        <p>${post.body}</p>
8      </article>
9    `;
10 };
```

Line: 3:31 | JavaScript | Soft Tabs: 2

kitchen-async.js — Desktop

```
1  const request = $.getJSON('/api/endpoint');
2
3  const renderPost = (post) => {…};
11
12 request.then(posts => post.map(renderPost));
13
```

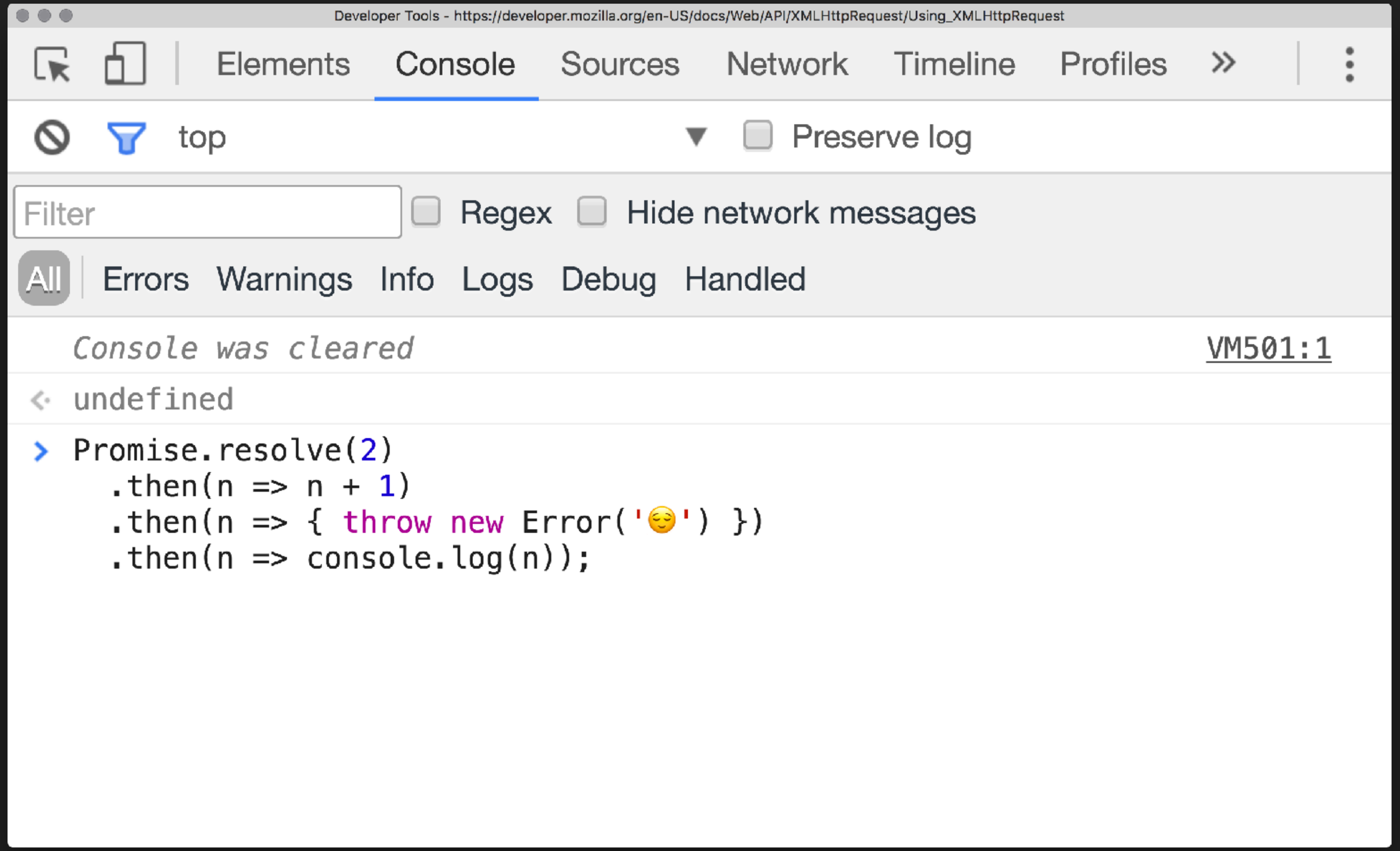
Line: 12:45 | JavaScript | Soft Tabs: 2

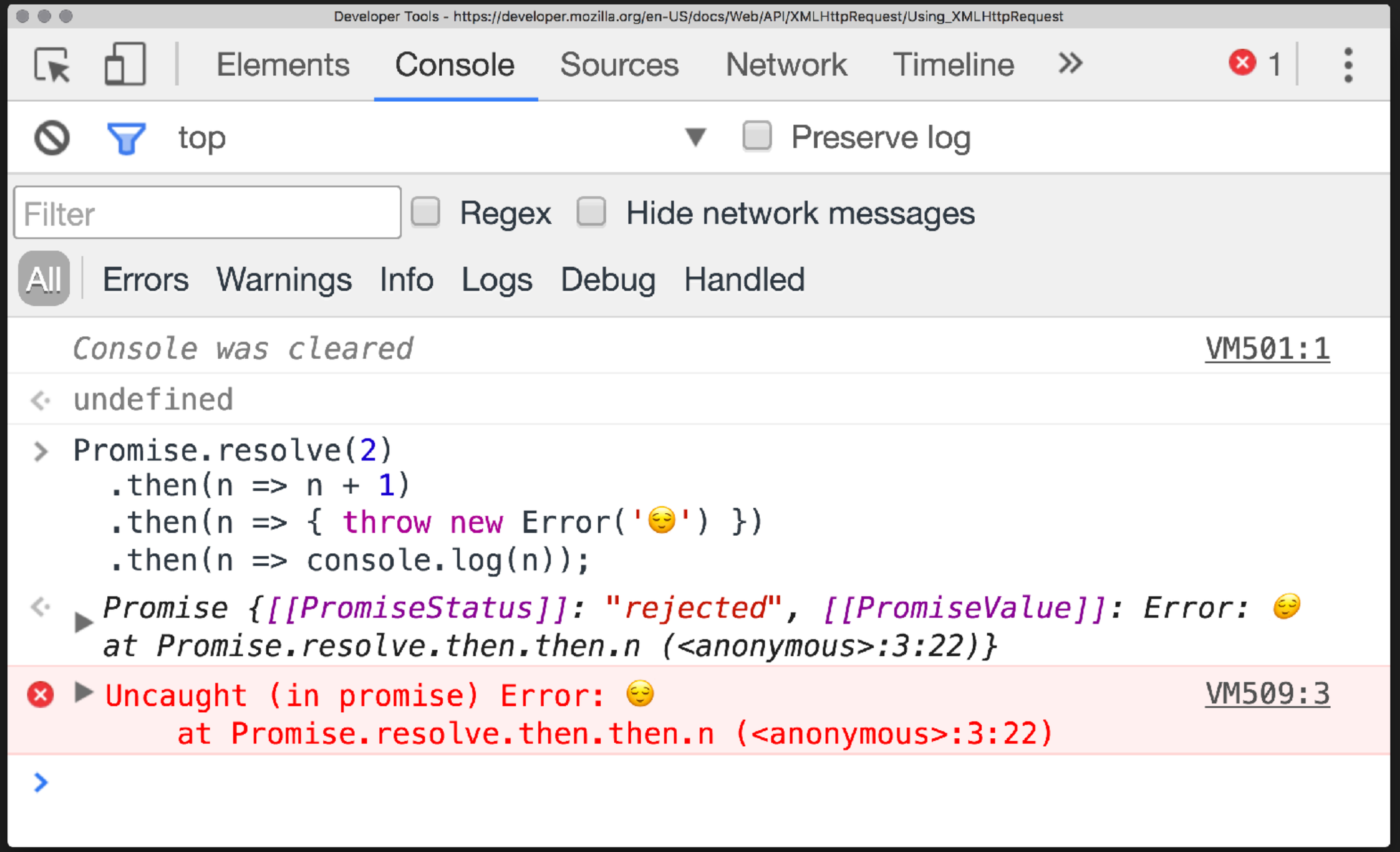
kitchen-async.js — Desktop

```
1  const request = $.getJSON('/api/endpoint');
2
3  const renderPost = (post) => {•••};
11
12 request.then(posts => post.map(renderPost))
13         .then(renderedPosts => {
14           // These are now the rendered posts!
15         });
16
```

Line: 15:11 | JavaScript | Soft Tabs: 2

How ‘bout when things go wrong?

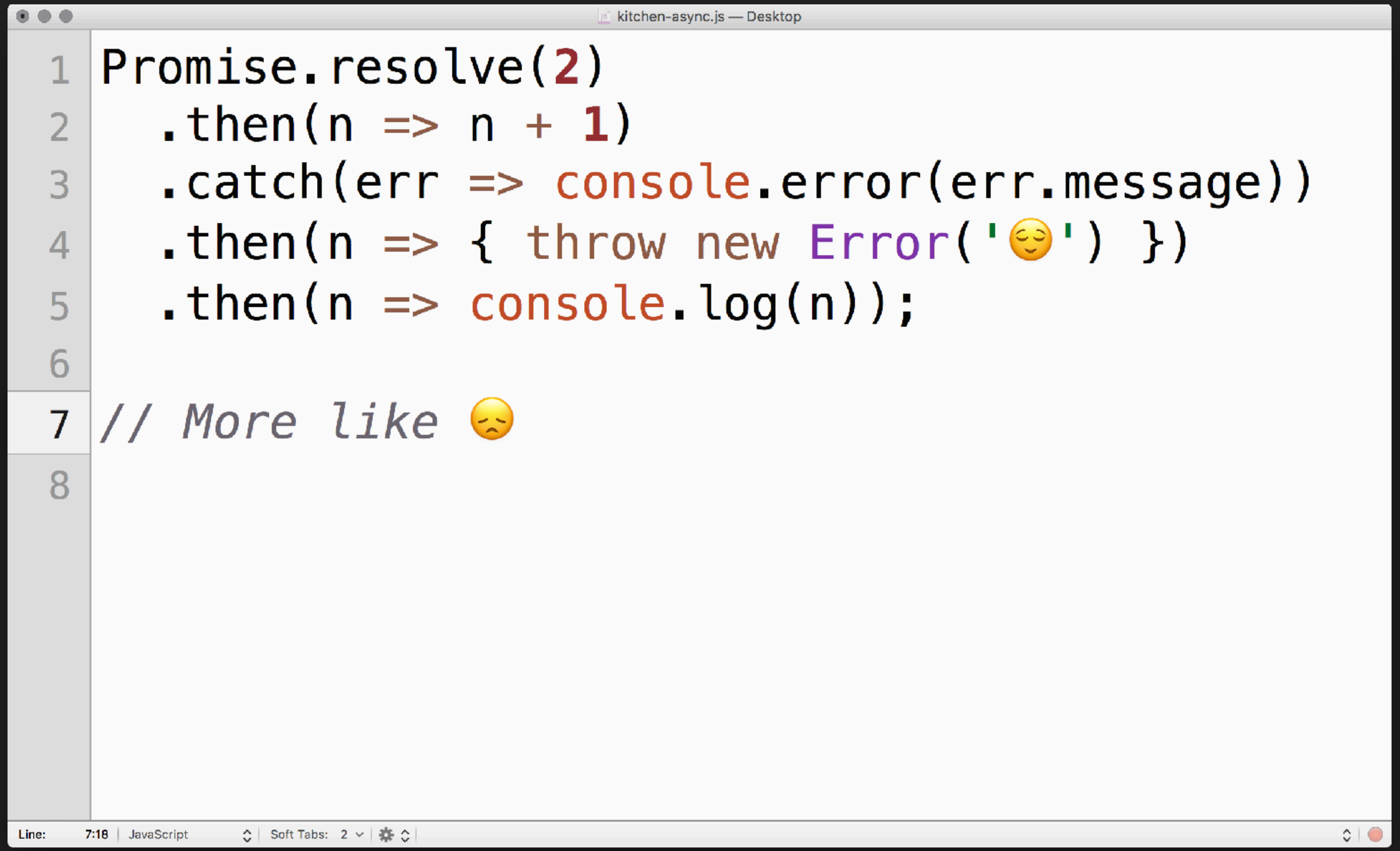




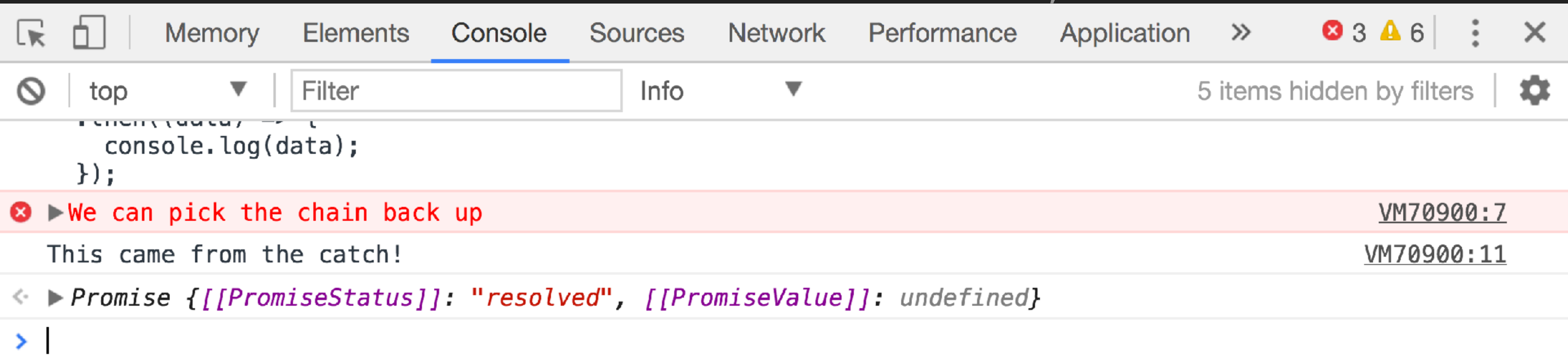
kitchen-async.js — Desktop

```
1  Promise.resolve(2)
2    .then(n => n + 1)
3    .then(n => { throw new Error('😔') })
4    .then(n => console.log(n))
5    .catch(err => console.error(err.message));
```

Line: 5:45 | JavaScript | Soft Tabs: 2



```
Promise.resolve('Wowowow')
  .then((data) => {
    // Let's pretend something goes wrong.
    throw new Error('Pretend something went wrong!');
  })
  .catch((error) => {
    console.error('We can pick the chain back up');
    return 'This came from the catch!';
```



JS Workers

JS Workers

- Parallel, not just concurrent
- Each is its own independent script
- Can send messages back to its creator
- Must obey same-origin policy

```
// app.js
let w = new Worker('worker.js');

w.postMessage({foo: 'bar'});
console.log('App Sent');

w.onmessage = (e) ⇒ {
  console.log(
    `App Received: ${e.data}`
  );
};
```

JS Workers

```
// app.js
let w = new Worker('worker.js');

w.postMessage({foo: 'bar'});
console.log('App Sent');

w.onmessage = (e) => {
  console.log(
    `App Received: ${e.data}`
  );
};
```

```
// worker.js
const log = console.info.bind(console);

self.onmessage = ({ data }) => {
  log('Worker received', data);
  doWork(data.foo);
}

function doWork(value) {
  self.setTimeout(() => {
    let x = value.toUpperCase();
    log('Worker sending to app', x);
    self.postMessage(x);
  }, 2000);
}
```

Why should we care?

Dedicated vs. Shared Workers

- Dedicated Workers can only communicate with the "script" (frame) that spawned them
  - Multiple JS files are still one "script"
- Shared Workers can communicate between multiple frames on the same origin
  - Multiple tabs/windows/frames

Worker Limitations & Features

- No DOM/cookies/localStorage
- Limited `location` and `navigator` objects
- `setInterval` and `setTimeout` (and respective clears*)
- `fetch` and `Promise`.
- Cache Storage API
- WebSocket
- IndexedDB

Terminating a worker

- You may have multiple dedicated or shared worker instances, based off of the same script file
- They'll each have their own scope, and take up non-trivial resources
- You may kill them from application-land

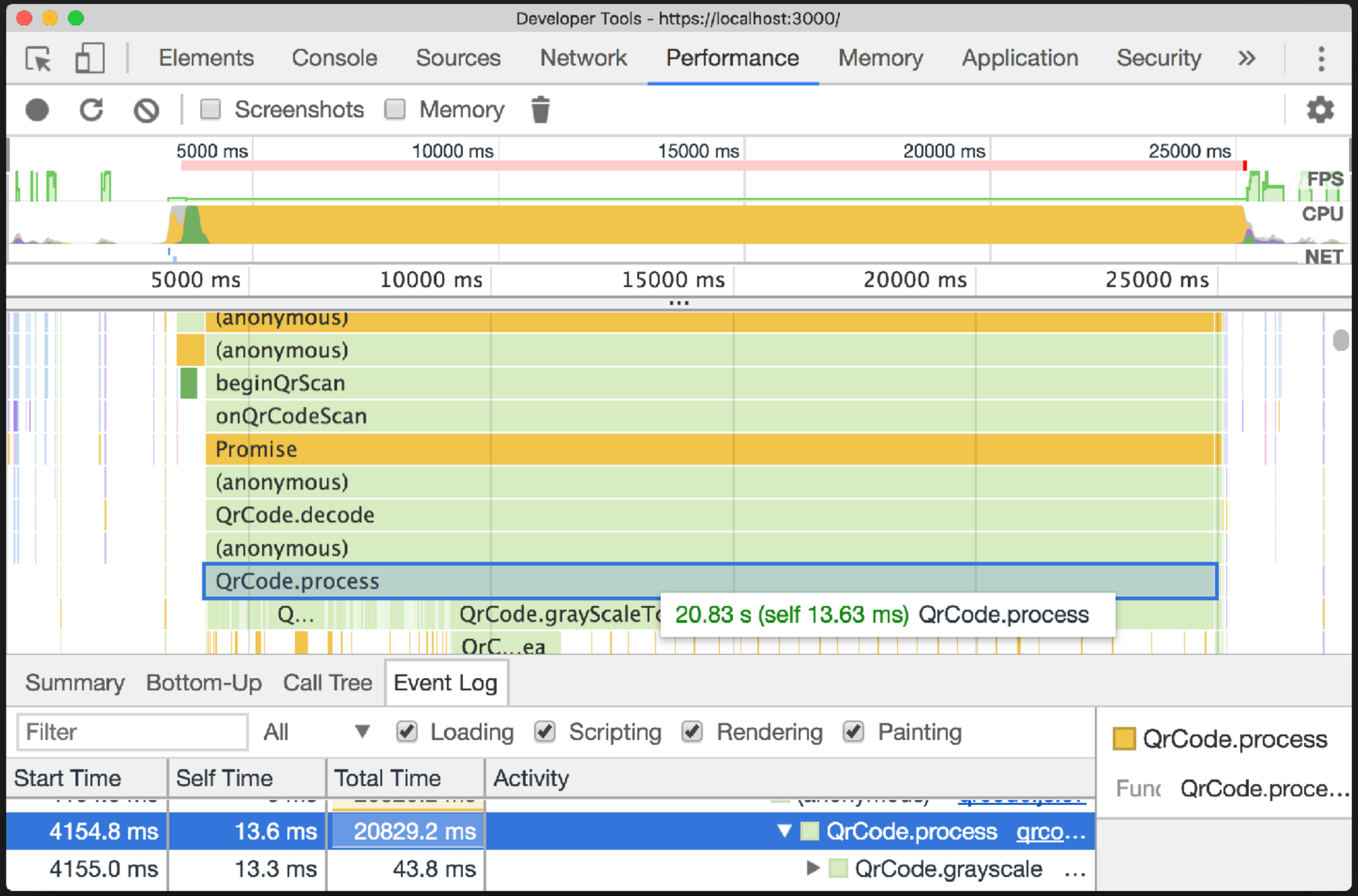
```
let task = new Worker('do-chores.js');
task.terminate();
```

- This is a great tool for the "do something and then die" worker pattern
- You can't terminate a Service Worker this way

Web Workers

3

- Users want to be able to scan groceries into their cart via a QR code
- Our QR code reader performs some complex image analysis to find the code and interpret it
- This is heavy work! Current implementation causes the app to hang during processing
- Your task: move the call to qr.decode into a web worker
- ./banana.png



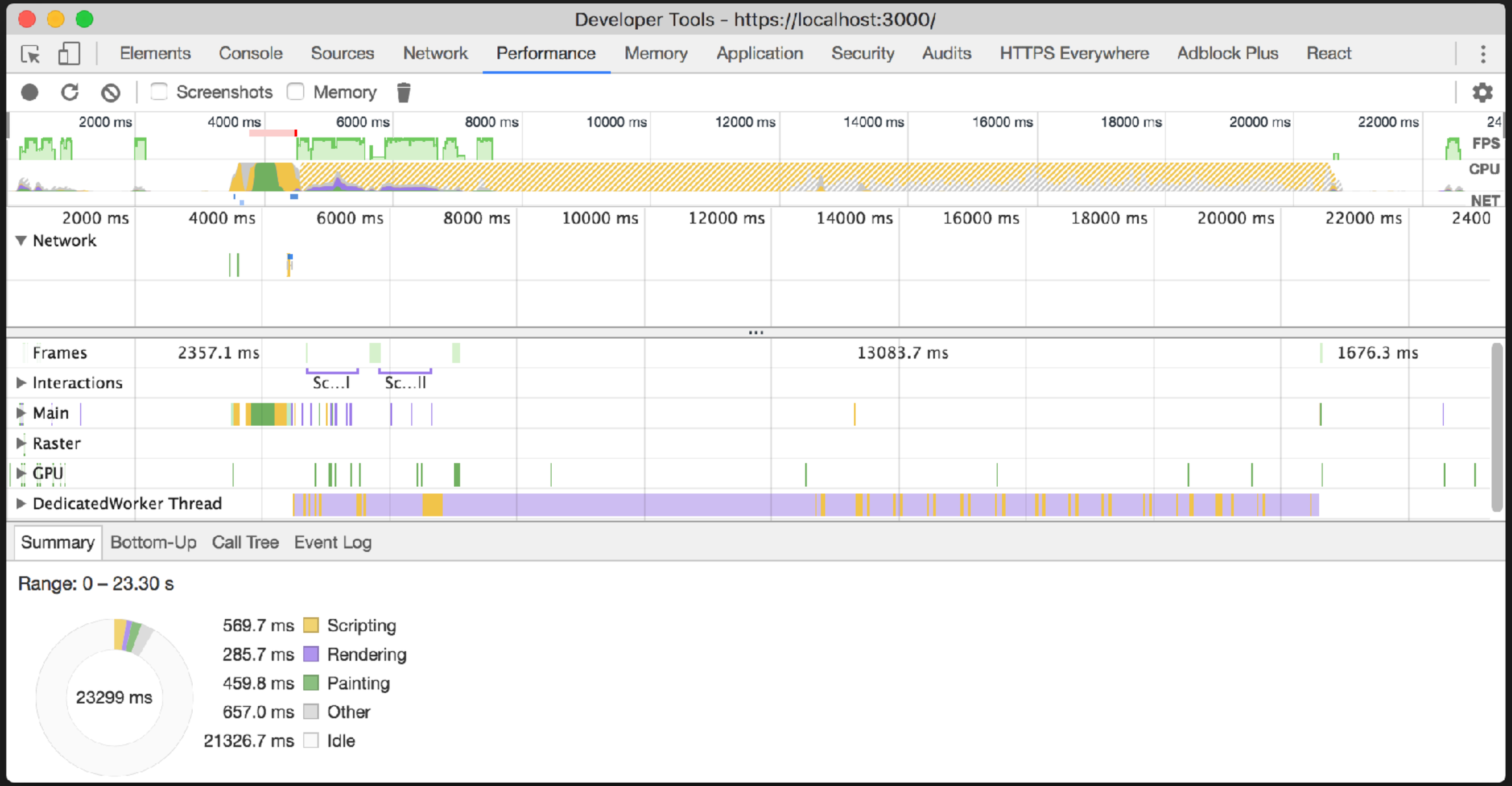
Web Workers

3

- We need a new file. Add this import to `./client/app.jsx`.

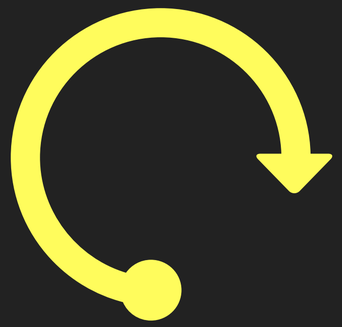
```
import 'worker-loader?name=./qrwork.js!./qrwork.js';
```

- We'll take advantage of mobile device convention for `<input type="file" />`
- Most of your work will be done in `./client/util/qrcode.js`.
- `npm run build:prod` Before and after to compare



WOW 🙌

- We're not abusing the main thread
- Our gzipped app just got smaller
- We're taking advantage of multi-core
- Doing this kind of work on the client enables rich offline feature sets



Async Data

Getting Data: Ancient Edition™

```
const xhr = new XMLHttpRequest();
xhr.open('GET', url);
xhr.responseType = 'json';

xhr.onload = () ⇒ { console.log(xhr.response); };

xhr.onerror = () ⇒ { console.error("🤡"); };

xhr.send();
```

Getting Data: Classic Edition™

```
$.ajax(‘http://example.com').then((data) ⇒ {
   …
});
```

- Not much control over request/response
- A browser-only construct
- XMLHttpRequest wasn’t really made for how we typically use it

Getting Data: Fetch

```
fetch('http://example.com/')
  .then((response) ⇒ response.json())
  .then((jsonData) ⇒ {
    ...
  });
```

- Can be polyfilled today
- First-class treatment of request/response objects
- More Node.js friendly

Getting Data: Fetch from the Future

```
const getData = async (url) ⇒ {
  try {
    const response =  await fetch(url);
    const data = await response.json();
    // Do stuff…
  } catch (error) {
    // Handle error…
  }
}
```

So, Fetch is like AJAX, right?

- The promises returned by a Fetch request will not reject if the response has a 400- or 500-level status code.
- It only rejects on network failure. (This is actually a good thing for us today.)
- By default, Fetch does not send or receive any cookies.

Fetch is Much More Than AJAX

- Typically with AJAX, we're just sending and receiving JSON or XML.
- The later is even baked into the name.
- With Fetch, we can access a wider range of data types.

The Three Things You Need To Know

- How the `fetch()` method works
- Configuring `Request` objects
- Working with `Response` objects

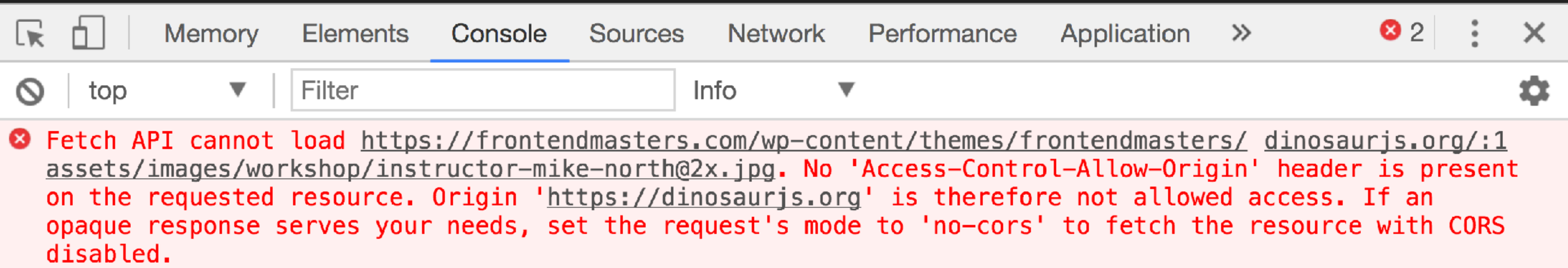
A Basic Fetch Request

```
const imageElement = document.querySelector('img');

fetch('beatles.jpg').then(response => {
  return response.blob();
}).then(response => {
  const objectURL = URL.createObjectURL(response);
  imageElement.src = objectURL;
});
```

Are you going to talk about CORS?

- Yes.
- CORS rules apply, but you *can* request things like images that you don't plan on reading outright.
- This applies to images, audio, video, scripts.
- Rule of thumb: anything you might pass into the `src` attribute of an HTML element.

```
const mike = 'https://frontendmasters.com/wp-content/
themes/frontendmasters/assets/images/workshop/
instructor-mike-north@2x.jpg';

fetch(mike).then(response ⇒ {
 console.log(response);
});
```



```
const mike = 'https://frontendmasters.com/wp-content/
themes/frontendmasters/assets/images/workshop/
instructor-mike-north@2x.jpg';

fetch(mike, { mode: ‘no-cors' }).then(response ⇒ {
  console.log(response.type); // opaque
  const objectURL = URL.createObjectURL(response);
  imageElement.src = objectURL;
});
```

```
const headers = new Headers();

const options = { method: 'GET',
                  headers: headers,
                  mode: 'cors',
                  cache: 'default' };

const request = new Request('beatles.jpg', options);
```

```
const request = new Request('beatles.jpg', options);

fetch(request).then(response => {
  return response.blob();
}).then(response => {
  const objectURL = URL.createObjectURL(response);
  imageElement.src = objectURL;
});
```

```
const mike = 'https://frontendmasters.com/wp-content/
themes/frontendmasters/assets/images/workshop/
instructor-mike-north@2x.jpg';

const request = new Request(mike, {
  mode: 'no-cors'
});

fetch(request).then(response ⇒ {
  console.log(response.type); // opaque
});
```

Please make a mental bookmark here.

```
const content = 'I am a banana';
const contentLength = content.length.toString();
const headers = new Headers();

headers.append('Content-Type', 'text/plain');
headers.append('Content-Length', contentLength);
```

```
const content = 'I am a banana';
const contentLength = content.length.toString();
const headers = new Headers({
  'Content-Type': 'text/plain',
  'Content-Length': contentLength
});
```

What About Responses?

- Fetch works with more than just JSON and XML.
- We need to let it know what we're dealing with here.

```
fetch(request)
  .then(response ⇒ response.json())
  .then(doStuffWithResponse);
```

```
response.blob();
response.json();
response.text();
response.formData();
response.arrayBuffer();
```

A Quick Word on Reading the Body of a Request or Response

- You can only read the body of a Request or a Response object once.
- You can however, clone the object if you think you'll need to read it again.
- More on this later.

Using Fetch I

4

- Currently the user's shopping cart is stored in memory – it's never persisted to some durable media
- In the `CartStore` class defined in `./client/data/cart-store.js`, enhance the private function `_saveCart` so our cart is persisted to our API
- Look at `./API_PLAYGROUND.http` for examples of API usage.
  Hint: `PUT https://localhost:3100/api/cart/items`

Using Fetch II

5

- Once a user "checks out", the contents of their cart should be persisted so that an "order" is created. Hint: `POST https://localhost:3100/api/order`
- In the `CartStore` class defined in `./client/data/cart-store.js`, enhance the function `doCheckout` so that the order gets created in our back end
- After the API call is completed, call `_restoreCart` to update the client's "view" of any changes that may have happened as a result of order creation

```
( ... )
  .then(this._restoreCart)
  .then((newItems) ⇒ {
    this._items = newItems;
    this._onItemsUpdated();
  });
```

Service Worker

Resource Caching Strategies

“Check with Server”

- Content at a URL can change
- Cache-Control: “no-cache”
- Use Last-Modified or ETag
- Involves extra network requests 😥

```
index.html
├app.js
├app.css
└logo.png
```

These slides are what I typically use to rip on the limitations of HTTP caching, setting us up for service worker being a thing people should care about

Resource Caching Strategies

Immutable Content

- Content at a URL is never changed
- max-age=year
- index.html is the key
- Offline-friendly
- Often involves a checksum baked into the filename

```
index.html
├─app-ab4182c.js
├─app-cd014c.css
└─logo-aaff10.png
```

This is... limiting

- Timestamps and version change notifications are not enough for offline apps to work
- There's no place for imperative logic, based on request particulars, environmental conditions, etc..
- Appcache tried to solve this, but it's awful. We'll talk about this at the end, and then rock on the floor in a fetal position until we feel clean again

The New Offline

- Not a prescriptive solution this time – more like "ingredients" for a solution
- High degree of customizability. We don't all have the same needs!
- Built on existing modern web concepts that are broadly supported

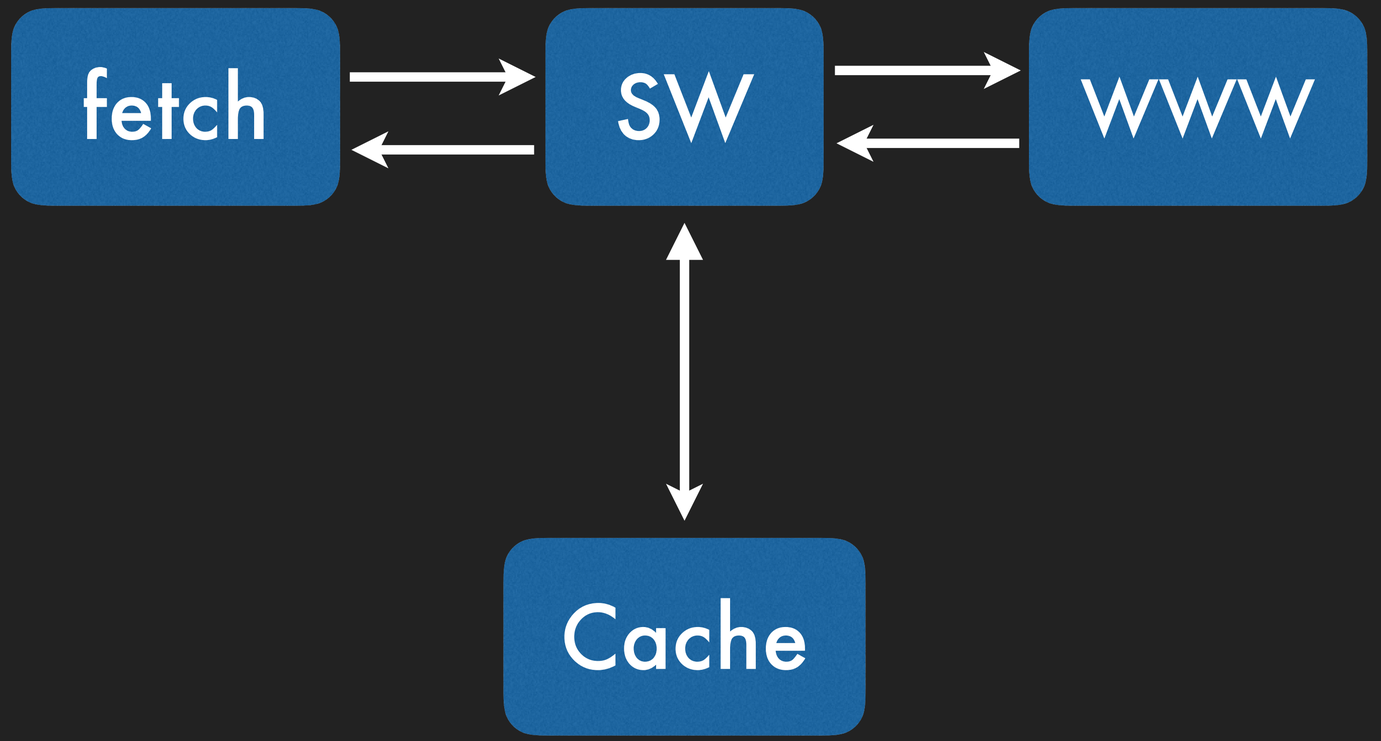
Service Worker

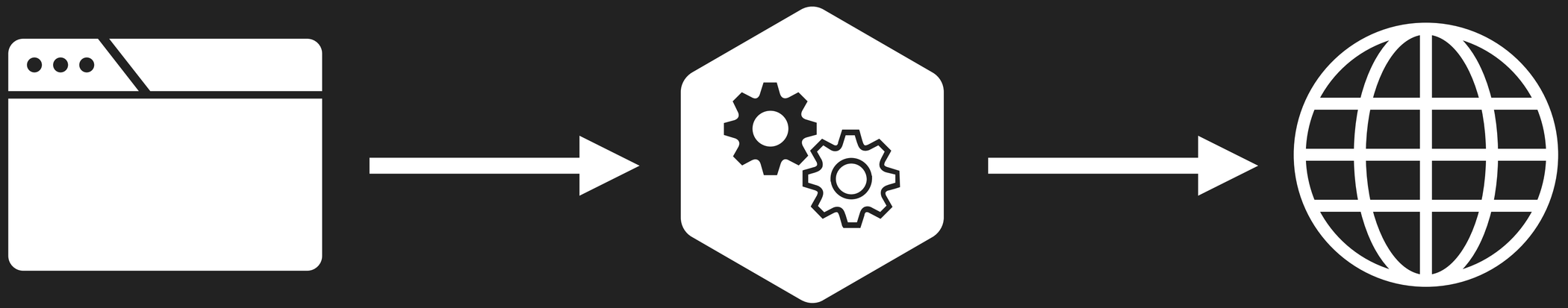
What even is a Service Worker?

- Service Workers run on their own thread just like a Web Worker or Shared Worker.
- They can intercept network requests.
- This means you can catch the request if the client is offline and either provide a fallback or use a resource from cache.
- Service worker can run the background, even when the web application is not open.

Service Worker

- A programmable network proxy
- A JavaScript worker
- Only works over secure domains
- Has a predictable lifecycle
- Necessary for many app-like capabilities, including "offline", notifications, background sync and more!





Service Worker

ServiceWorker enthusiasm

The first thing any implementation needs.

Chrome: Shipped.

Firefox: Shipped.

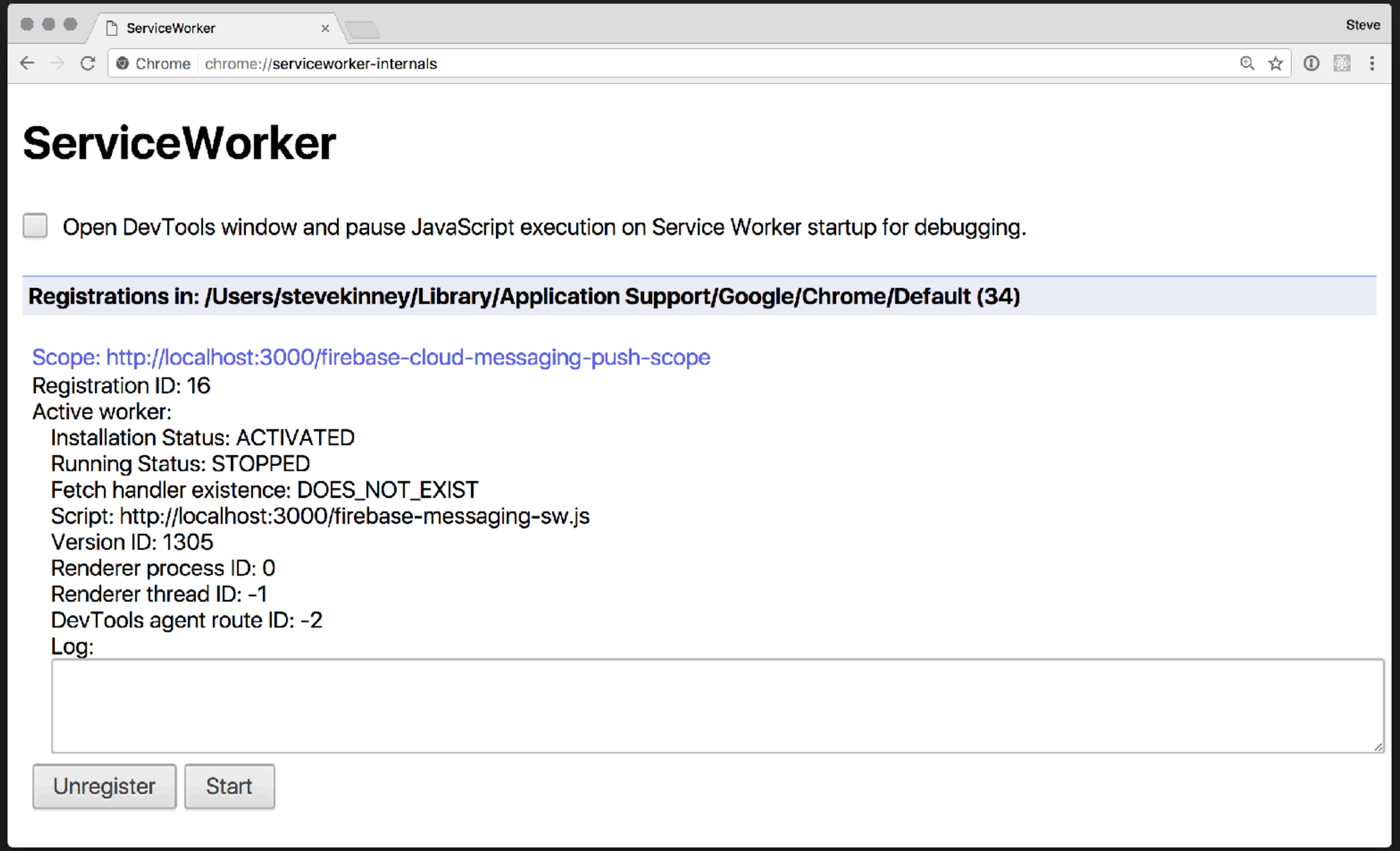
Samsung Internet: Shipped. Based on Chromium 44.2403 with some additions and changes. (See "Service Worker" section.)

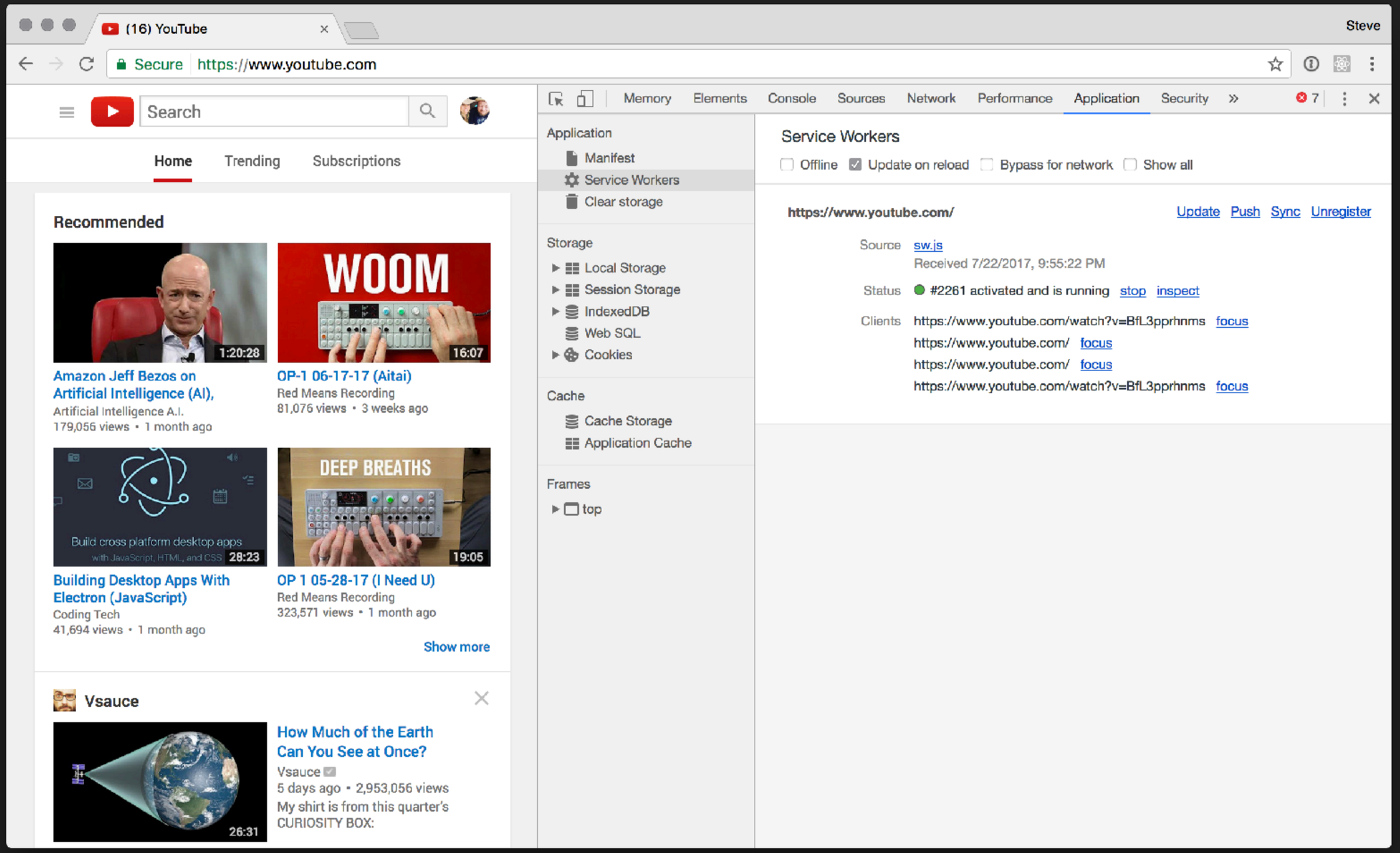
Safari: Under consideration, Brief positive signals in five year plan.

Edge: In development.

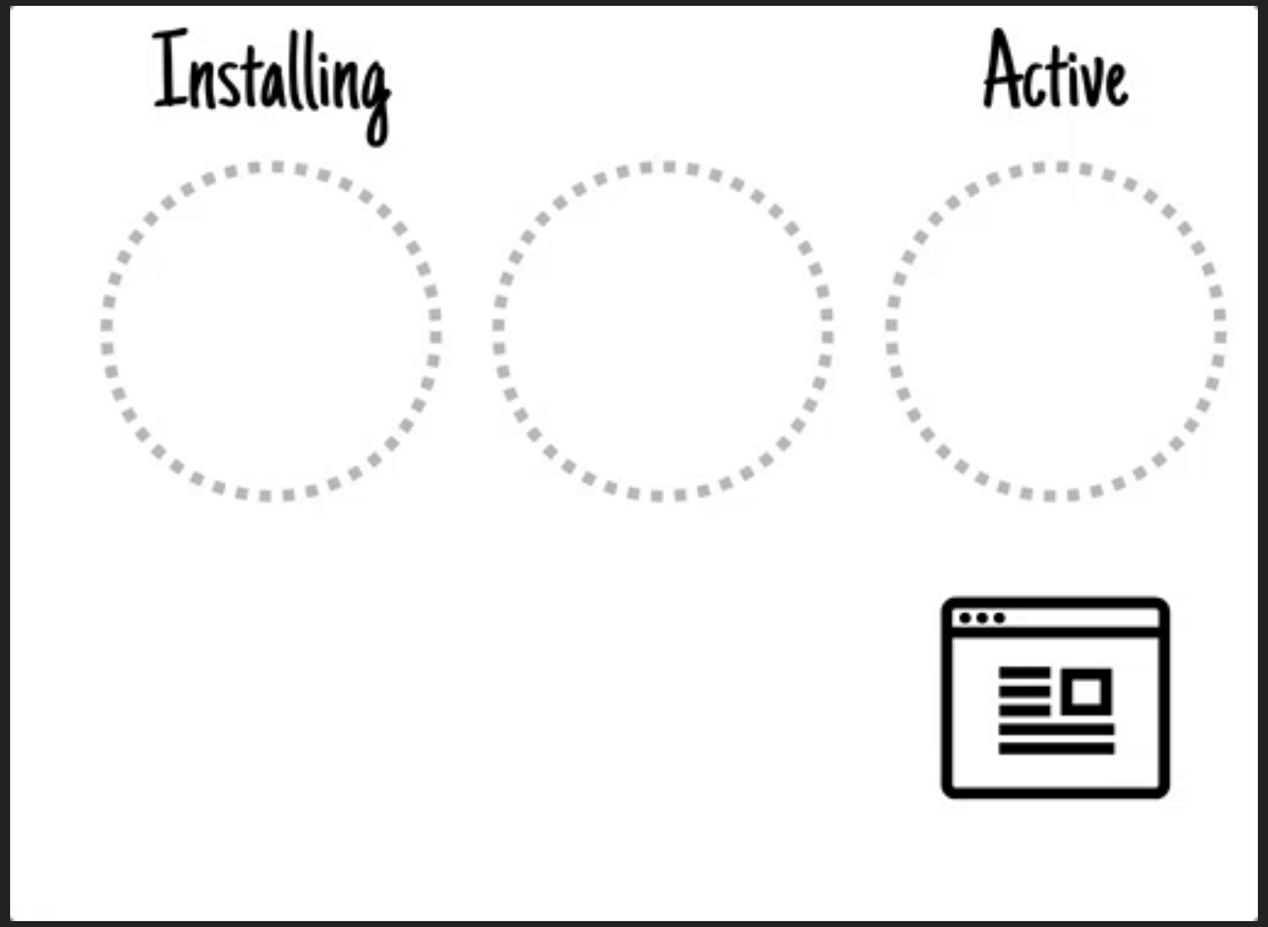
Support does not include iOS versions of third-party browsers on that platform (see Safari support).

```
if ('serviceWorker' in navigator) {
  // Okay, the browser supports service workers.
  navigator.serviceWorker.register('service-worker.js')
  .then(registration ⇒ {
    // The service worker is registered.
  })
  .catch(err ⇒ {
    // No dice.
  });
} else {
  // Service worker is not a thing, apparently.
}
```

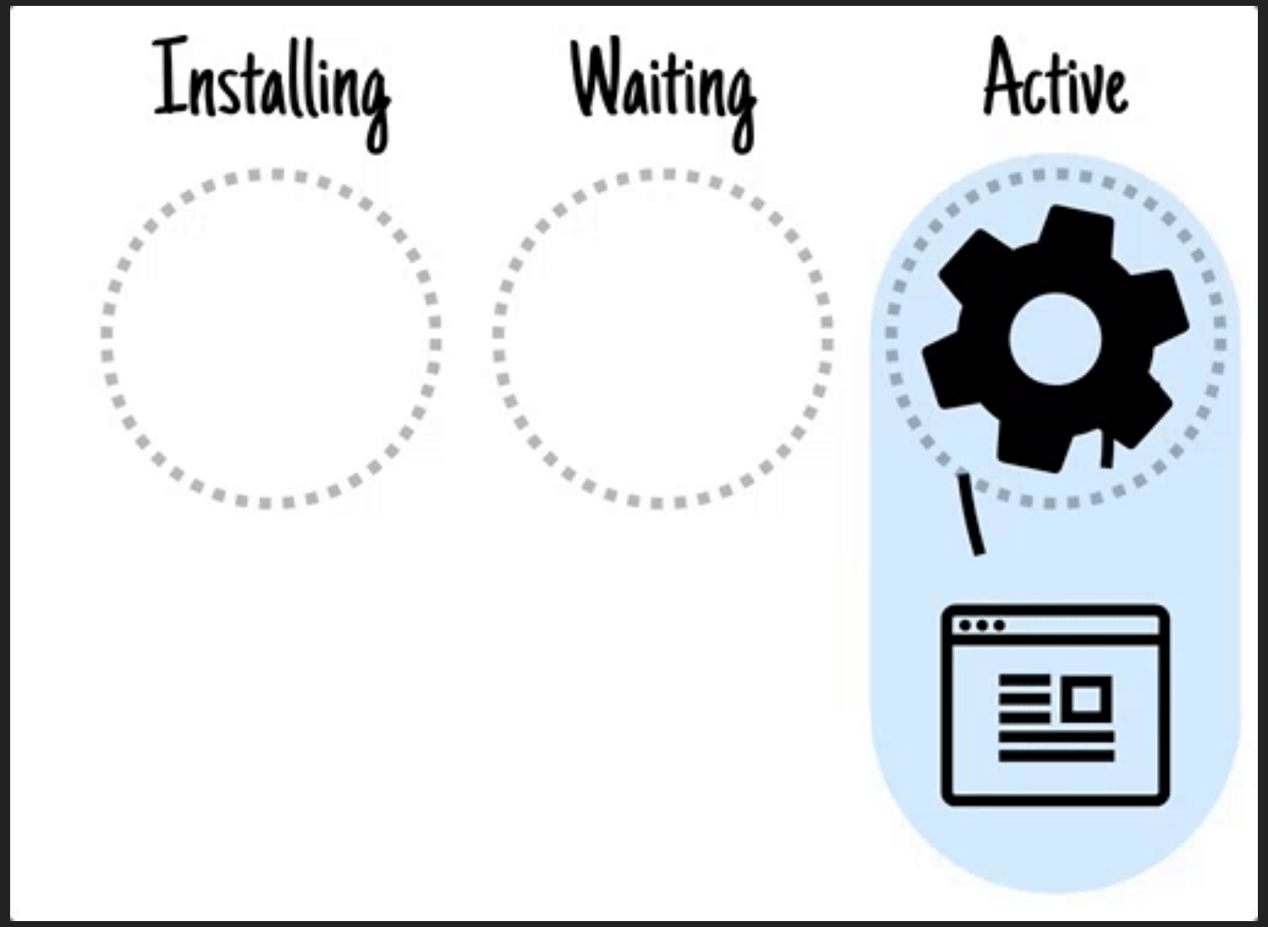




Service Worker Lifecycle

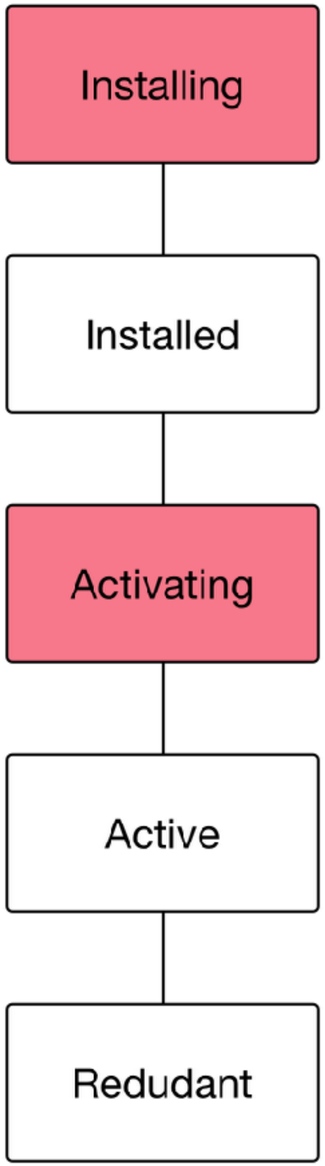


Service Worker Lifecycle



But, why have a lifecycle at all?

- Make offline-first something that's possible
- Allow the new service worker to get installed and ready without disturbing the current one
- Ensure that there is one and only one active service worker
  - Imagine if two open tabs were running two different service workers. 😨



The “install” event has fired, but it hasn’t completed yet.

The service worker is installed, but it’s waiting for another service worker to be unloaded.

The “activate” event has fired, but it hasn’t completed yet.

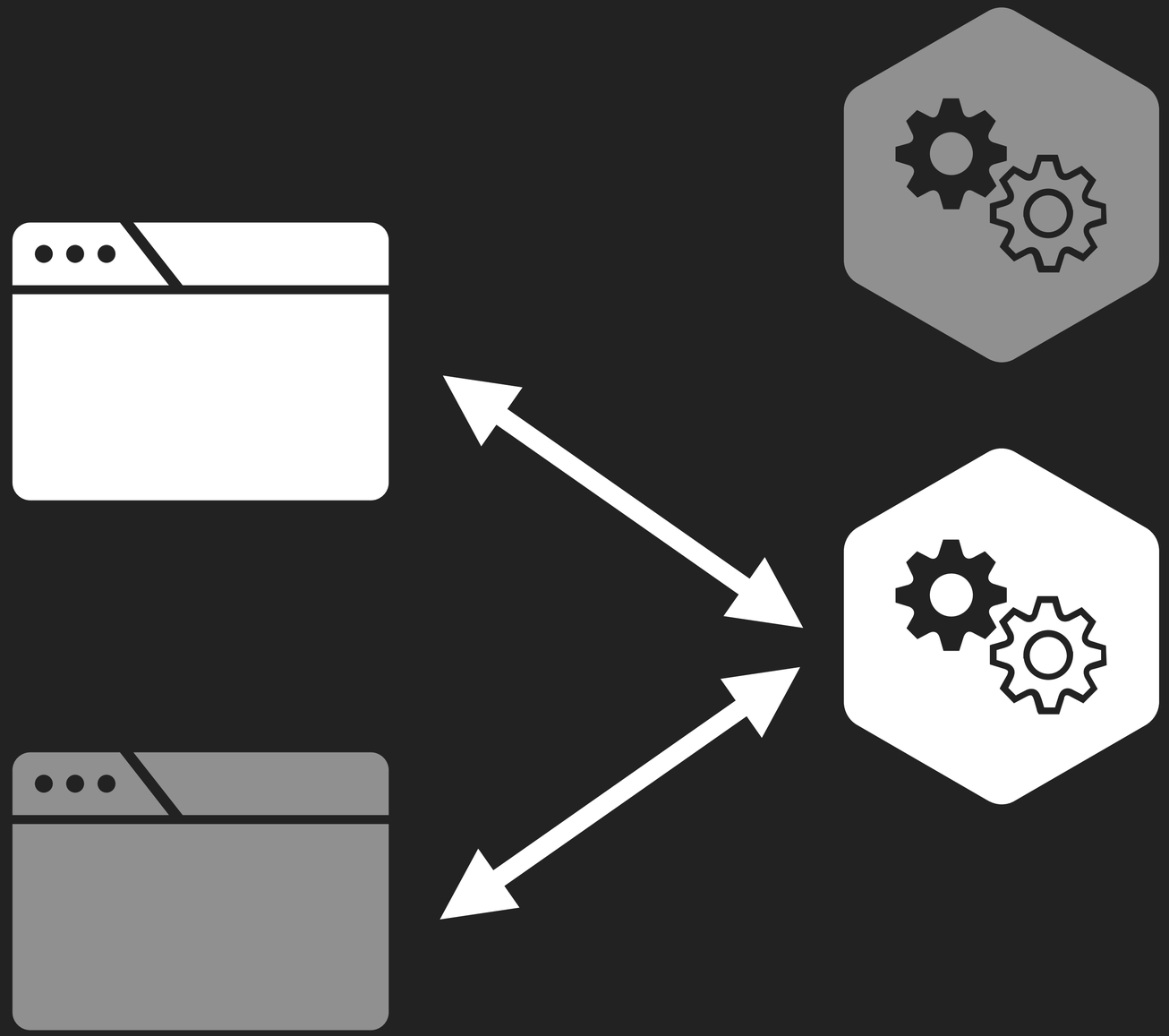
The service worker is installed an ready to begin handling events.

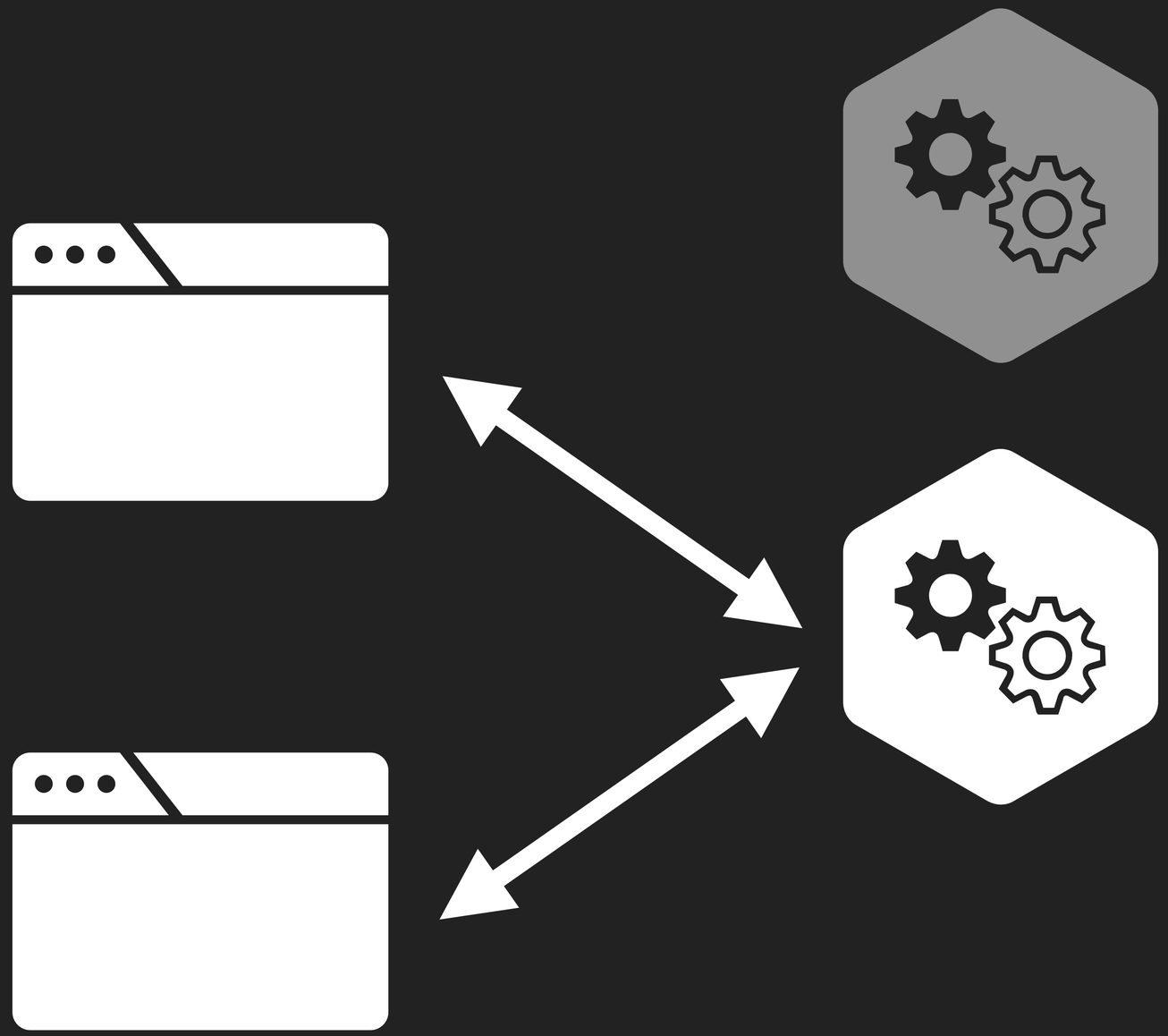
The service worker has been replaced by another one.

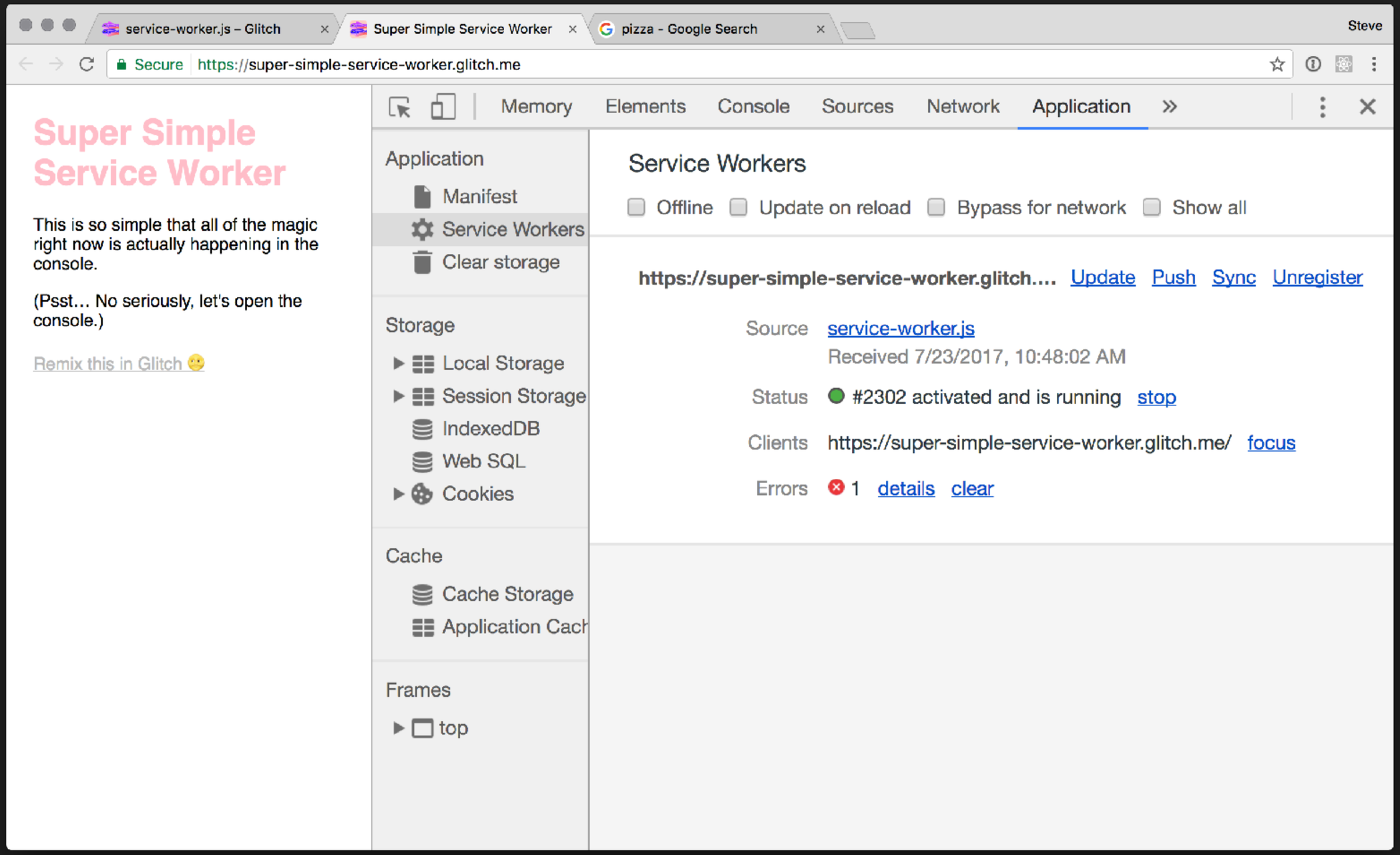
```
self.addEventListener('install', event ⇒ {
  // Do the stuff needed at install time.
  // This may not be the active service worker yet.
});

self.addEventListener('activate', event ⇒ {
  // We are ready to take the stage.
});

self.addEventListener('fetch', event ⇒ {
  // Intercept a network request.
});
```

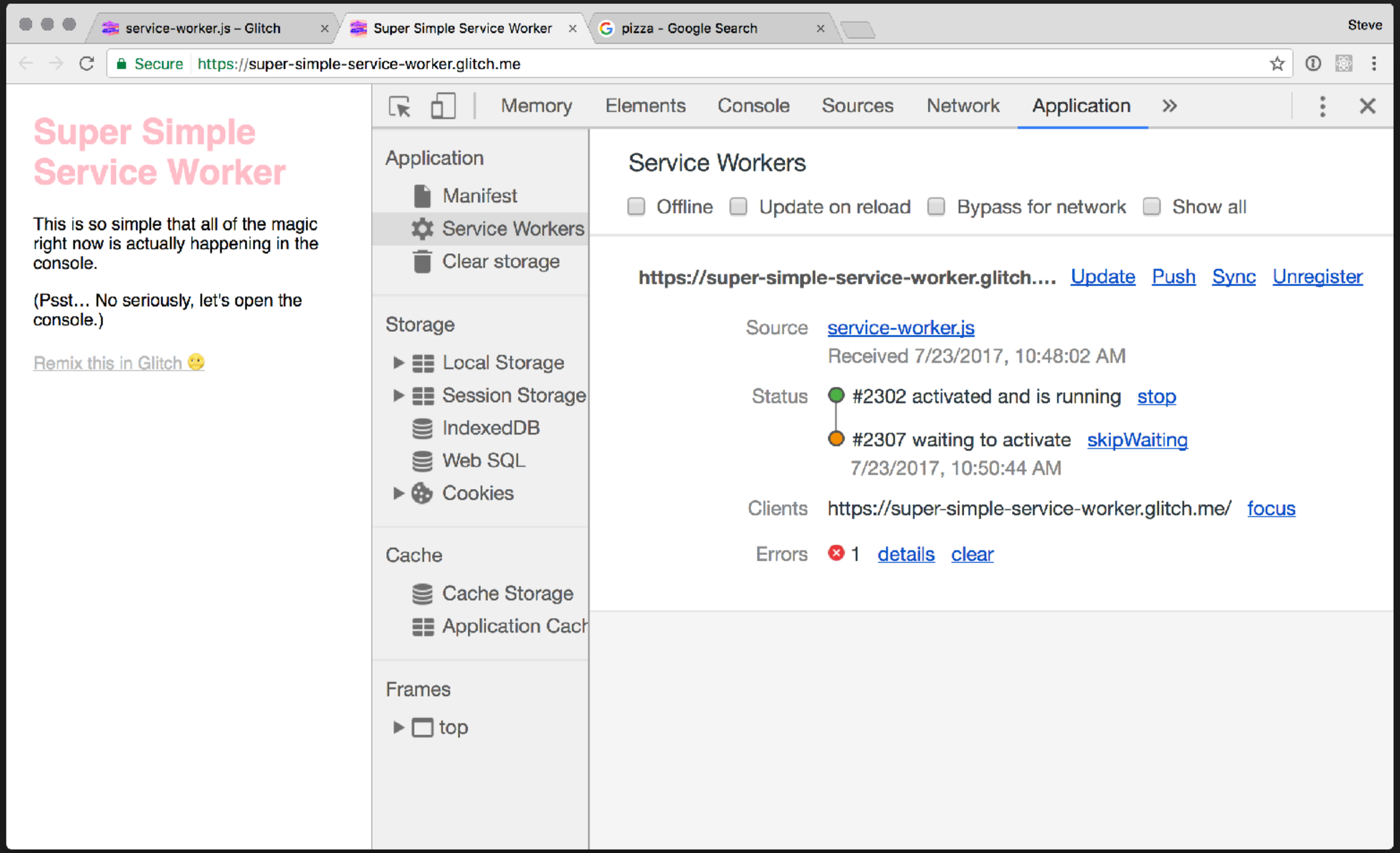
Conditions for Replacing a Service Worker

- Imagine this, you update your wonderful service worker.
- You proudly reload your page. Nothing changes.
- Your new service worker will not take over until all other instances of the application have been closed.
- It's like when a desktop application auto-updates. You don't get the new one until you totally quit out of the application and start it back up.



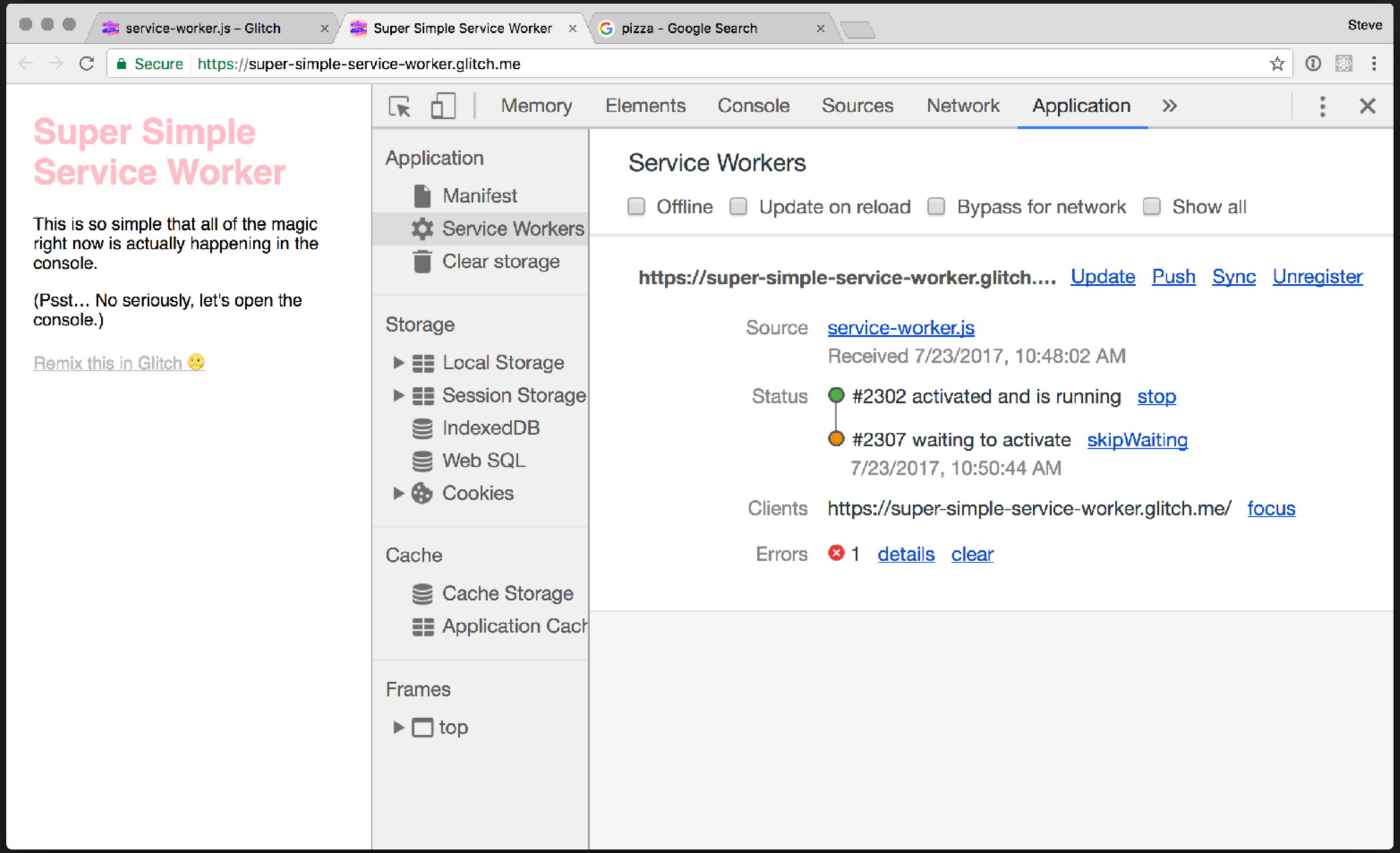


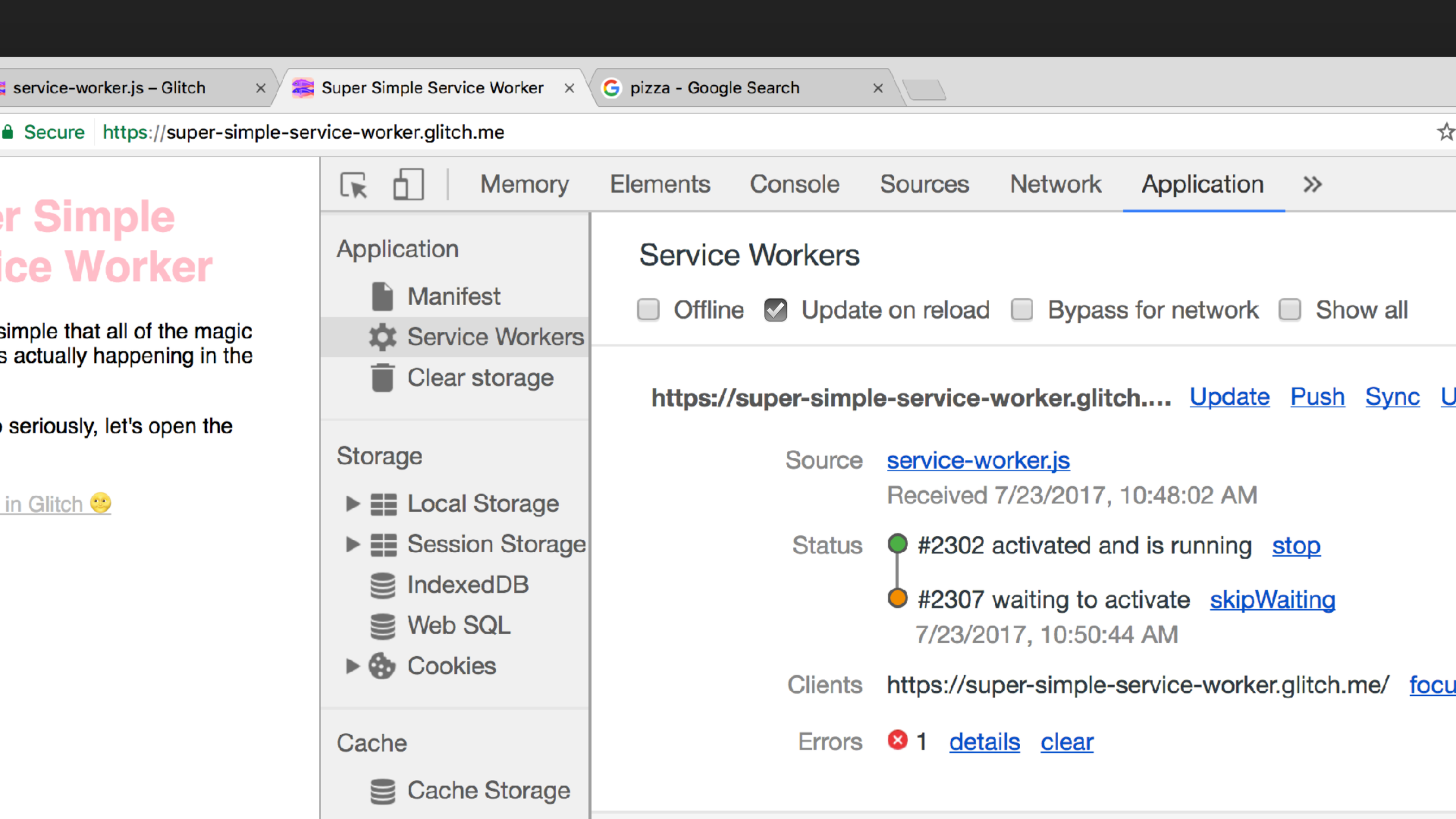




Conditions for Replacing a Service Worker

- If you don't want this in your application for some reason, you can call `event.skipWaiting()`.
- If you only want this in development, then the Chrome Developer Tools are your friend.




service-worker.js – Glitch
Super Simple Service Worker
pizza - Google Search
Secure
https://super-simple-service-worker.glitch.me
r Simple
ice Worker
simple that all of the magic
s actually happening in the
seriously, let's open the
in Glitch
Memory
Elements
Console
Sources
Network
Application
Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Cache
Cache Storage
Service Workers
Offline
Update on reload
Bypass for network
Show all
https://super-simple-service-worker.glitch....
Update
Push
Sync
Source
service-worker.js
Received 7/23/2017, 10:48:02 AM
Status
#2302 activated and is running
stop
#2307 waiting to activate
skipWaiting
7/23/2017, 10:50:44 AM
Clients
https://super-simple-service-worker.glitch.me/
Errors
1
details
clear

Super Simple Service Worker

6

https://super-simple-service-worker.glitch.me/

- We're going to implement this in our grocery application, but let's start with the simplest possible implementation.
- Make sure to test your code where service workers are/aren't supported
- Observe what happens when an error is thrown in the handlers for install and activate events.
- Count the number of times each event is invoked, and `console.log` the "current" counts in your event handlers

Service Worker 201

- One of the nice things about Service Worker is that it is very much influenced by the Extensible Web Manifesto.
- A core tenet is that web developers should have fine-grained access to how things work.
- As such, we an inspect and control Service Workers in ways that were not really possible in previous technologies.

```
self.addEventListener('install', event ⇒ {
  self.skipWaiting();

  // Immediately oust the current service worker
  // and install this one as the new king.
});
```

- This is kind of dangerous because you're replacing the service worker on other currently opened instances of your application—without replacing their client-side code.
- We somewhat regret even showing you this.

```
self.addEventListener('install', event ⇒ {
  // Push the current service worker out of the way
  // and take over.
  return self.skipWaiting();
});

self.addEventListener('activate', event ⇒ {
  // Take over all of the unclaimed clients
  return self.clients.claim();
});
```

```
navigator.serviceWorker
  .register('/service-worker.js')
  .then(registration ⇒ {
    registration.installing;
    registration.waiting;
    registration.active;
  });
```

- As part of the registration process, we can inquire about active Service Worker as well as the any that are presently installing or waiting to in line to supplant the active worker.

```
navigator.serviceWorker
  .register(‘/service-worker.js')
  .then(registration ⇒ {
    registration.addEventListener('updatefound', () ⇒ {
      // A new challenger has emerged!
      const newWorker = registration.installing;

      newWorker.state;
      // installing, installed, activating, activated,
      // or redundant

      newWorker.addEventListener('statechange', () ⇒ {
        // The state of this new worker has changed.
      });
    });
  });
```

```
navigator.serviceWorker
         .addEventListener('controllerchange', () => {
           // A new service worker just took over.
           // This probably happened via
           // event.skipWaiting();
         });
```

Intercepting Network Requests with Service Worker

```
self.addEventListener('fetch', event ⟹ {
  console.log(event.request.url);
});
```

- Listening for the “fetch” events will not only give us XHR requests, but *all* network requests.
- That includes images, CSS, HTML, and other JavaScript.

Working with the FetchEvent

- **event.request** → the request object
  - **event.request.url** → the original URL request
  - **event.request.method** → the HTTP method used
- **event.respondWith** → allows you to respond with something other than what was being requested

```
self.addEventListener('fetch', event => {
  if (event.request.url == new URL(location).href) {
    event.respondWith(
      new Response('<p>Hello!</p>', {
        headers: {'Content-Type': 'text/html'}
      })
    )
  }
});
```

So, we can intercept requests and send custom responses— but what should we send?

Quick Demonstration

- https://intercepting-fetch.glitch.me/

Introducing: Cache API

What is the Cache API?

- It's very cool that we can intercept requests, but that doesn't get us any closer to the Progressive Web App™ stamp of approval.
- The Cache API gives us fine-grain, programatic control over caching assets from inside of a service worker.

```
const cacheName = 'assets-v1';

caches.open(cacheName).then(cache ⟹ {
  // Do stuff with the cache.
});
```

- Opening the cache returns a promise.

You can tell your cache about a bunch of things you'd like it to hold on to.

```
const cacheName = 'assets-v1';

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(cacheName).then(cache => {
      return cache.addAll(
        [
          '/index.html',
          '/style.css',
          '/client.js',
        ]
      );
    })
  );
});
```

```
const cacheName = 'assets-v1';

self.addEventListener('install', event ⇒ {
  event.waitUntil(
    caches.open(cacheName).then(cache ⇒ {
      cache.add('/not-critical.css');
      return cache.addAll(
        [
          '/index.html',
          '/style.css',
          '/client.js',
        ]
      );
    })
  );
});
```

Wait—what was event.waitUntil?

- The budding Service Worker can be killed at any time by the browser.
- `event.waitUntil()` keeps the “install” process going until the promise it was handed resolves.
  - This allows us to be assured that by the time the activate event is fired, the preceding install process has completed successfully.
  - If the promise it was given rejects, the Service Worker installation will fail, it will be abandoned, and the currently-active worker—if any—will remain in charge.

The Cache API stores pairs of request and response objects.

The Cache API in its Relation to Fetch

- Under the hood, the Cache API is a Map where `Request` objects are the keys and `Response` objects are the values.
- Put another way: this means that you can store responses and just serve them back if you receive a another matching request.

```
caches.open('band-assets').then(cache => {
  const request = new Request('/beatles.png');
  const response = new Response('/oasis.png');
  cache.put(request, response);
});
```

```
caches.open('band-assets').then(cache ⇒ {
  const request = new Request('/beatles.png');
  cache.add(request);
});
```

```
caches.open('band-assets').then(cache ⇒ {
  const request = new Request('/beatles.png');
  const response = fetch(request);
  cache.match(request, response);
});
```

```
self.addEventListener('install', event => {
  console.log(`
    This is a great time to set up any caches
    that your new service worker will need.
  `);
});

self.addEventListener('activate', event => {
  console.log(`
    This is a great time to remove any caches
    that your retired service worker was using.
  `);
});
```

How do we keep track of the current cache and clean up old ones?

```
const currentCache = 'assets-v1';

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(currentCache).then(cache => {
      return cache.addAll(
        [
          '/index.html',
          '/style.css',
          '/client.js',
        ]
      );
    })
  );
});
```

```
const currentCache = 'assets-v2';

self.addEventListener('activate', event ⇒ {
  event.waitUntil(
    caches.keys().then(cacheNames ⇒ Promise.all(
      cacheNames.filter(cacheName ⇒ {
        return cacheName ⥽ currentCache
      }).map(cacheName ⇒ caches.delete(cacheName))
    ))
  );
});
```

The Cache API is not just for Service Workers

- You can access the Cache API from the browser context as well.
- This can be useful if you want to allow users to manually add items to the cache.
- (Think: “Save for Offline”)

Fallback Image

7

https://intercepting-fetch.glitch.me/

- In the last example, we set up a Service Worker, but it was kind of pointless. All it did was talk about the requests being made.
- In this example, we'll provide our own fallback image, in the event we encounter a 404 response status for image requests.
- All requests for things other than images should be unaffected by the service worker
- The fallback image is https://localhost:3100/images/generic-fallback.jpg
- **BONUS 1:** Retrieve and store the image during the Service Worker installation phase
- **BONUS 2:** Respond with the fallback image, not only if a 404 is encountered, but also if a request for an image takes longer than 100ms

Fallback Image

7

```
// Get the Accept header from the request
let acceptHeader = event.request.headers.get('accept');
// Build a URL object from the request's url string
let requestUrl = new URL(event.request.url);

if (acceptHeader.indexOf('image/*') >= 0 && // if it's an image
  requestUrl.pathname.indexOf('/images/') === 0) { // and the url looks right
  // do something interesting
  console.log('this is a product image!');
}
```

Cache API: Strategies & Use

Cache-Only

```
self.addEventListener('fetch', event ⇒ {
  event.respondWith(caches.match(event.request));
});
```

- I hope you like being offline, because this service worker will try to pull everything from cache and never get to the network.
- It's supposed to be "offline-first" not "offline-only." 😡

Network-Only

```
self.addEventListener('fetch', event ⟹ {
  event.respondWith(fetch(event.request));
});
```

- Wait, what? Why?
- I'm going somewhere with this. I promise.

Neither of these make us feel particularly, great—maybe we can mix and match?

Cache with a Network Backup

```
self.addEventListener('fetch', event ⇒ {
  event.respondWith(
    caches.match(event.request).then(response ⇒ {
      return response || fetch(event.request);
    })
  );
});
```

Service Worker Caching Strategies

Precache

I have a list of important resources. I'll fetch them all and cache them.

Cache

App

I need item.png

Of course! Here it is.

Demonstration

https://cache-money.glitch.me/

Network with Cache Backup

```
self.addEventListener('fetch', event ⟹ {
  event.respondWith(
    fetch(event.request).catch(() {
      return caches.match(event.request);
    })
  );
});
```

Service Worker Caching Strategies

Network or Cache

App

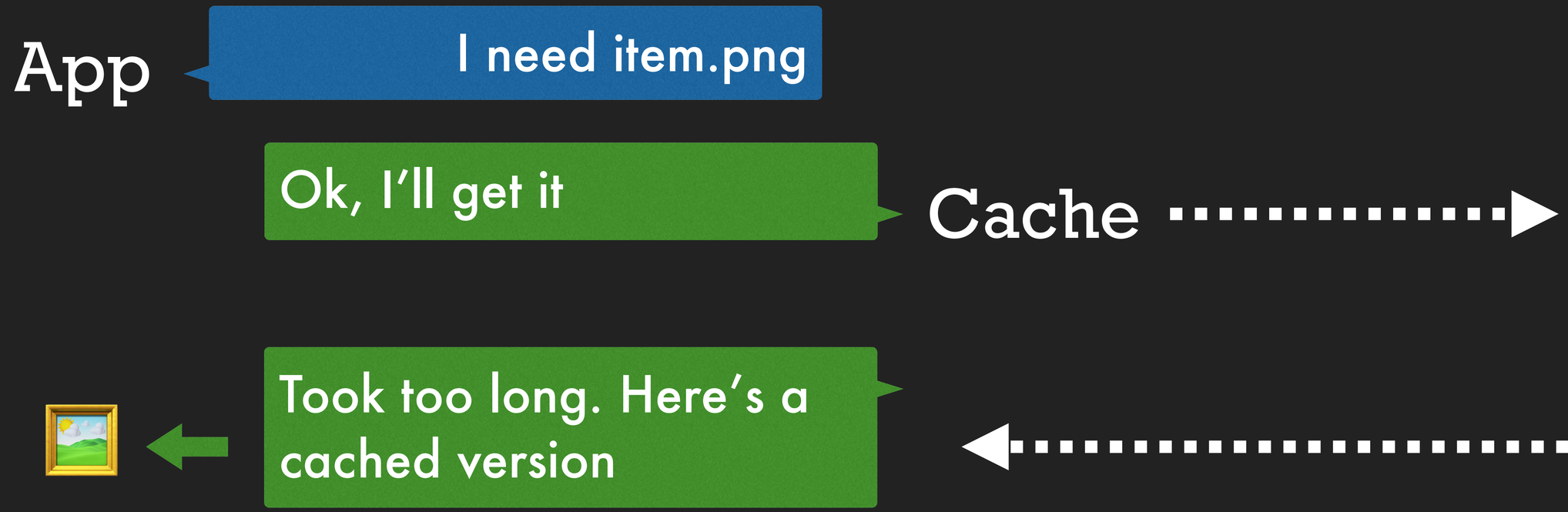
I need item.png

Ok, I'll get it

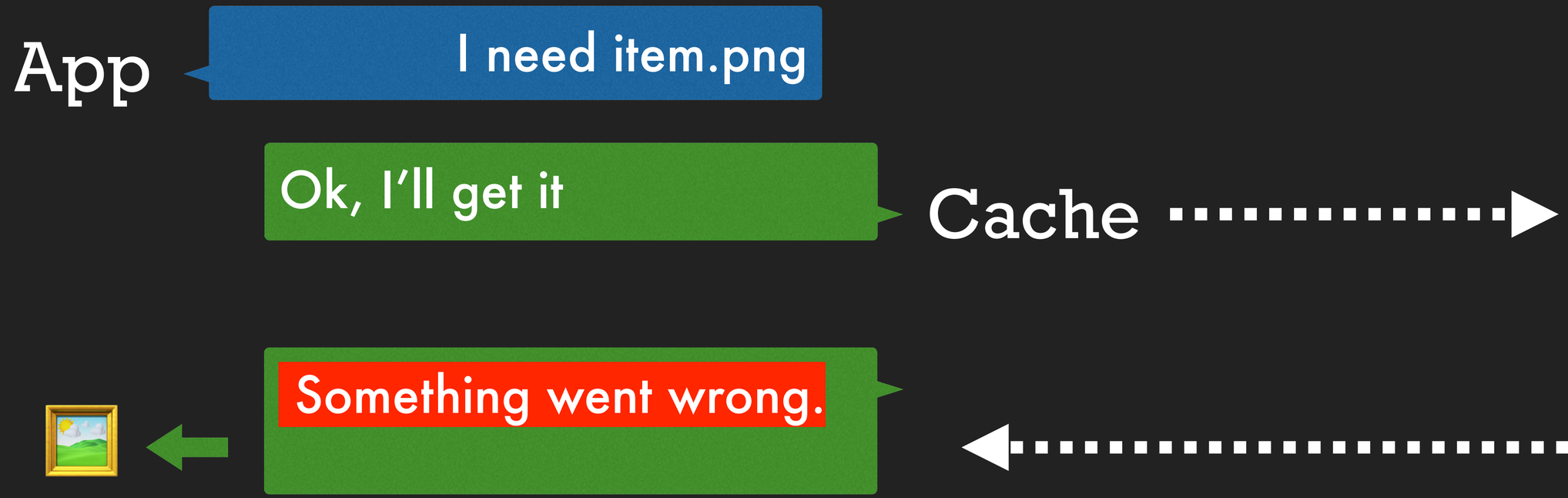
Cache

Just got it for you!

Service Workers
Network or Cache



Service Workers
Network or Cache



This is getting better, but we're still iterating towards something here it feels like.

Cache then Network

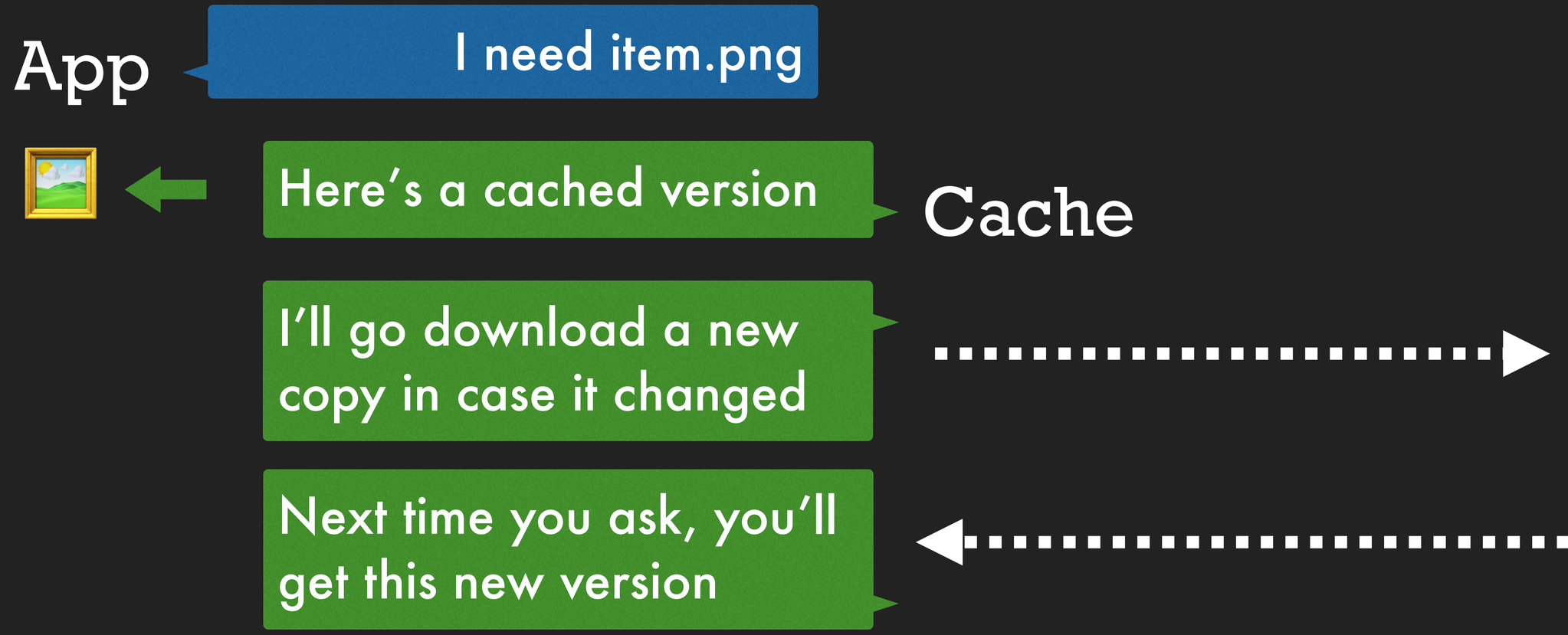
```
let didWeReceiveFreshNetworkData = false;
const doSomethingWithData = () ⇒ { … };
```

Cache then Network

```
const fetchFromNetwork = fetch('/very-important-data.json')
  .then(response ⟹ {
    return response.json();
  }).then(data ⟹ {
    didWeReceiveFreshNetworkData = true;
    doSomethingWithData(data);
  });
```

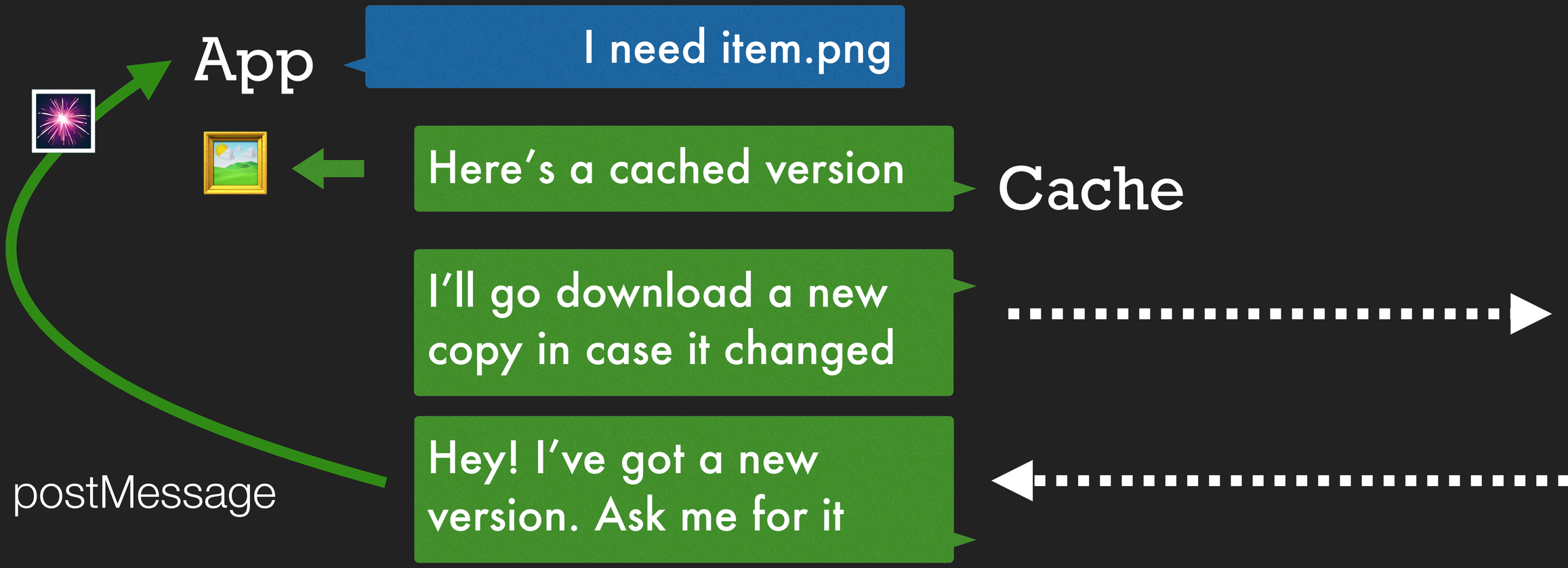
Service Worker Caching Strategies

Cache and Update



Service Worker Caching Strategies

Cache, Update and Refresh



Cache then Network

```
caches.match('/very-important-data.json')
  .then(response ⇒ {
    return response.json();
  }).then(data ⇒ {
    if (!didWeReceiveFreshNetworkData) {
      doSomethingWithData(data);
    }
  }).catch(() ⇒ {
    return fetchFromNetwork;
  });
```

Fallback Image

7

https://intercepting-fetch.glitch.me/

- In the last example, we set up a Service Worker, but it was kind of pointless. All it did was talk about the requests being made.
- In this example, we'll provide our own fallback image, in the event we encounter a 404 response status for image requests.
- All requests for things other than images should be unaffected by the service worker
- The fallback image is https://localhost:3100/images/generic-fallback.jpg
- **BONUS 1:** Retrieve and store the image during the Service Worker installation phase
- **BONUS 2:** Respond with the fallback image, not only if a 404 is encountered, but also if a request for an image takes longer than 100ms

Generic Fallback

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request);
    }).catch(() => {
      return caches.match('/placeholder.png');
    })
  );
});
```

We didn't even talk about the fact we might want to handle caching differently for different types of assets.

What was the browser requesting?

```
self.addEventListener('fetch', event ⇒ {
  const acceptHeader = event.request.headers.get('accept');
  console.log('Fetching', acceptHeader, event.request.url);

  event.respondWith(
    caches.match(event.request).then(response ⇒ {
      return response || fetch(event.request);
    })
  );
});
```

```
if (acceptHeader.indexOf("image/*")) {
  'I am an image.';
}
if (acceptHeader.indexOf('text/css')) {
  'I am a CSS file.';
}
if (acceptHeader.indexOf('text/html')) {
  'I am an HTML file.';
}
if (acceptHeader.indexOf('*/*')) {
  'YOLO!';
}
```

Different types of files in different parts of your application will have different needs.

Precache for Quick Boot

8

- We've had an asset manifest available this whole time!

https://localhost:3000/asset-manifest.json

- In your install event handler, fetch the asset manifest, and then iterate over this object to "precache" all of the assets it names.
- In your activate event handler, iterate over all of the caches, and delete any that are no longer useful
- In your fetch event handler, try to serve these assets from the cache (if possible) and fall back to retrieving them from the internet

Caching dynamic data

9

- While our app is now able to boot without an internet connection, as soon as we need JSON or product images, we're in trouble if we're offline
- Implement a fetch event handler that implements the "cache fallback" strategy for all GET requests that aren't Precached in advance
  - Core idea: use fresh data from a fetch if possible, and fall back to the cached data if something goes wrong (i.e., user is offline)
- Store these responses in a separate cache than the preached responses

SPA treatment of index.html

10

- Single-page apps have to pay special attention index.html because some portion of the URL's path may be used for client-side routing
- Precache index.html in your install handler

```
const INDEX_HTML_PATH = '/';
const INDEX_HTML_URL = new URL(INDEX_HTML_PATH, self.location).toString();
```

- For local (same origin) GET requests that have the appropriate Accept header, attempt to fetch a fresh index.html, and fall back to the precached response if something goes wrong

```
let isHTMLRequest = request.headers.get('accept').indexOf('text/html') !== -1;
  let isLocal = new URL(request.url).origin === location.origin;
```

IndexedDB

IndexedDB

- 🔢 Versioned
- 🗃 Indexable
- ⚙️ Worker-Friendly
- 📆 Durable
- 🔣 Supports values of many types
- ⚠️ Not a SQL, or a relational DB

IndexedDB

```
// Open (or create) the database
let open =
  indexedDB.open('MyDatabase', 1);

// Create and/or Migrate the Schema
open.onupgradeneeded = (evt) ⇒ { ... };


// Transactions
open.onsuccess = () ⇒ { ... };
```

IndexedDB

```
// Create and/or Migrate the Schema
open.onupgradeneeded = (evt) => {

    let db = open.result;
    let store = null;
    switch (evt.oldVersion) {
      case 0: // Upgrade from 0
        //  ...
      case 1: // Upgrade from 1
        //  ...
    }

};
```

IndexedDB

```
// Open (or create) the database
let open =
  indexedDB.open('MyDatabase', 1);


// Create and/or Migrate the Schema
open.onupgradeneeded = (evt) ⇒ { ... };


// Transactions
open.onsuccess = () ⇒ { ... };
```

IndexedDB

```
// Transactions
open.onsuccess = () ⇒ {

  // Start a new transaction
   let db = open.result;
   let tx = db.transaction('MyObjectStore', 'readwrite');
   let store = tx.objectStore('MyObjectStore');

   // ⟵ Transaction Particulars ⟶ //

   // Close the db when the transaction is done
   tx.oncomplete = () ⇒ db.close();

};
```

IndexedDB

```
let index = store.index('NameIndex');

 // Add some data
 store.put({
   id: 12345,
   name: { first: 'Mike', last: 'North' },
   age: 33
 });

 // Query the data
 let getJohn = store.get(12345);
 let getBob = index.get(['North', 'Mike']);
```

IndexedDB

IndexedDB - REC

Method of storing data client-side, allows indexed database queries.

Global	74.13% + 8.26% =	82.39%
unprefixed:	74.06% + 8.07% =	82.13%
All Web Site Data	90.25% + 2.68% =	92.93%
unprefixed:	90.25% + 2.64% =	92.89%

Current aligned | Usage relative | Date relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.3	
			51					4.4	
			54			2 9.3		4.4.4	
1 11	1 14	50	55	10	42	10.2	all	53	55
	1 15	51	56	TP	43				
		52	57		44				
		53	58						

IDB: IndexedDb Promisified

```
const dbPromise = idb.open('keyval-store', 1, upgradeDB ⇒ {
  upgradeDB.createObjectStore('keyval');
});

dbPromise.then(db ⇒ {
  const tx = db.transaction('objs', 'readwrite');
  tx.objectStore('objs').put({
    id: 123456,
    data: {foo: "bar"}
  });
  return tx.complete;
});
```

IndexedDB

11

- In your service worker's activate handler, populate an IndexedDB store with grocery store items
- To fetch a large number of items from the API, you can use limit=9999 as a queryParam

```
GET https://localhost:3100/api/grocery/items?limit=99999
```

- As part of the installation process, store this data in IndexedDb, lazily creating an appropriate store on the fly if it doesn't exist yet
- You may use raw IndexedDb or idb (recommended).

IndexedDB

11

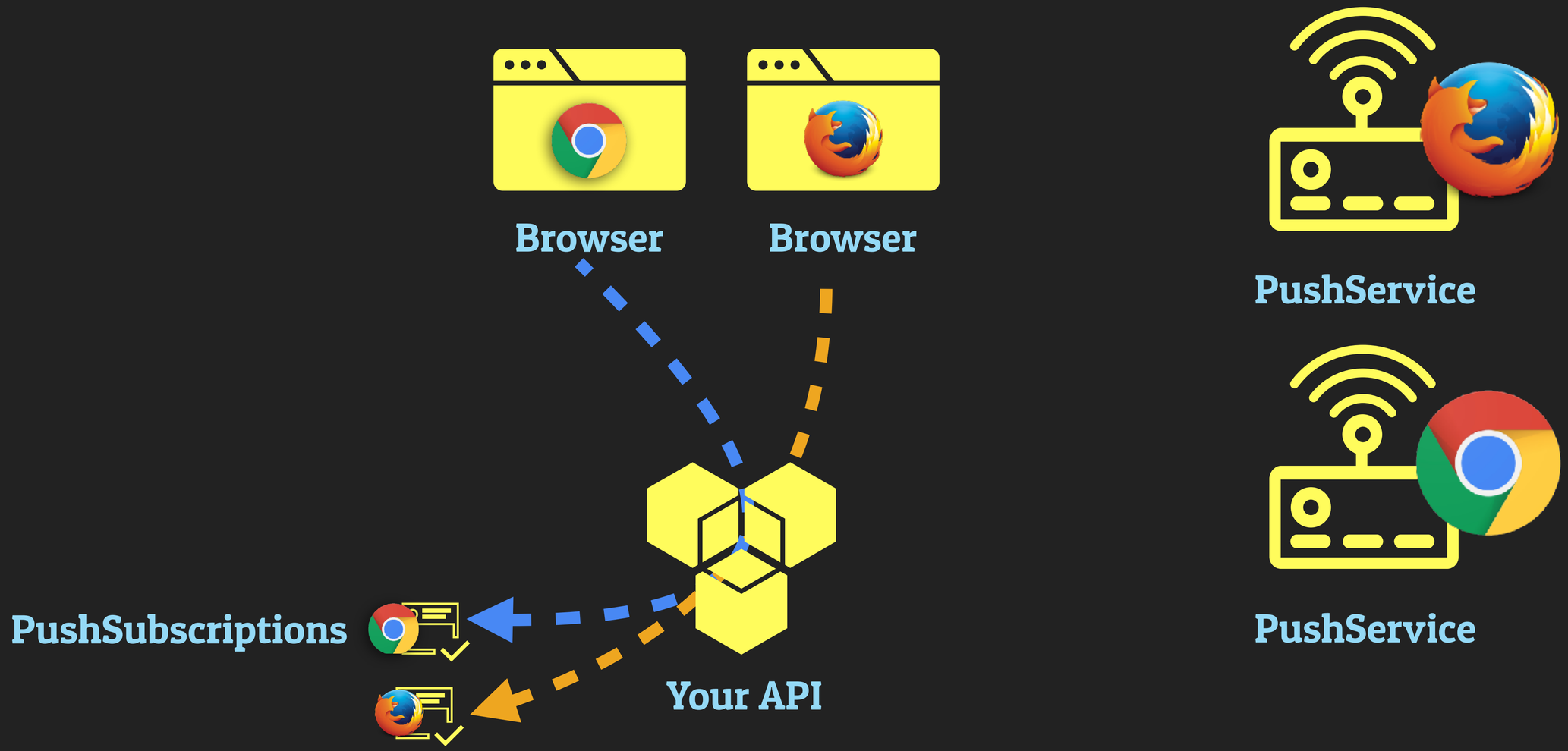
- BONUS: Upgrade your fallback images, so that they are provided on a per-category basis, depending on the image.
  - This will involve taking the id of the record (i.e., 124.jpg is the image for product with id=124), and then...
  - Using IndexedDB to identify the value item's "category" attribute, and finally...
  - Serving up the preached response for the appropriate placeholder image

Web Push

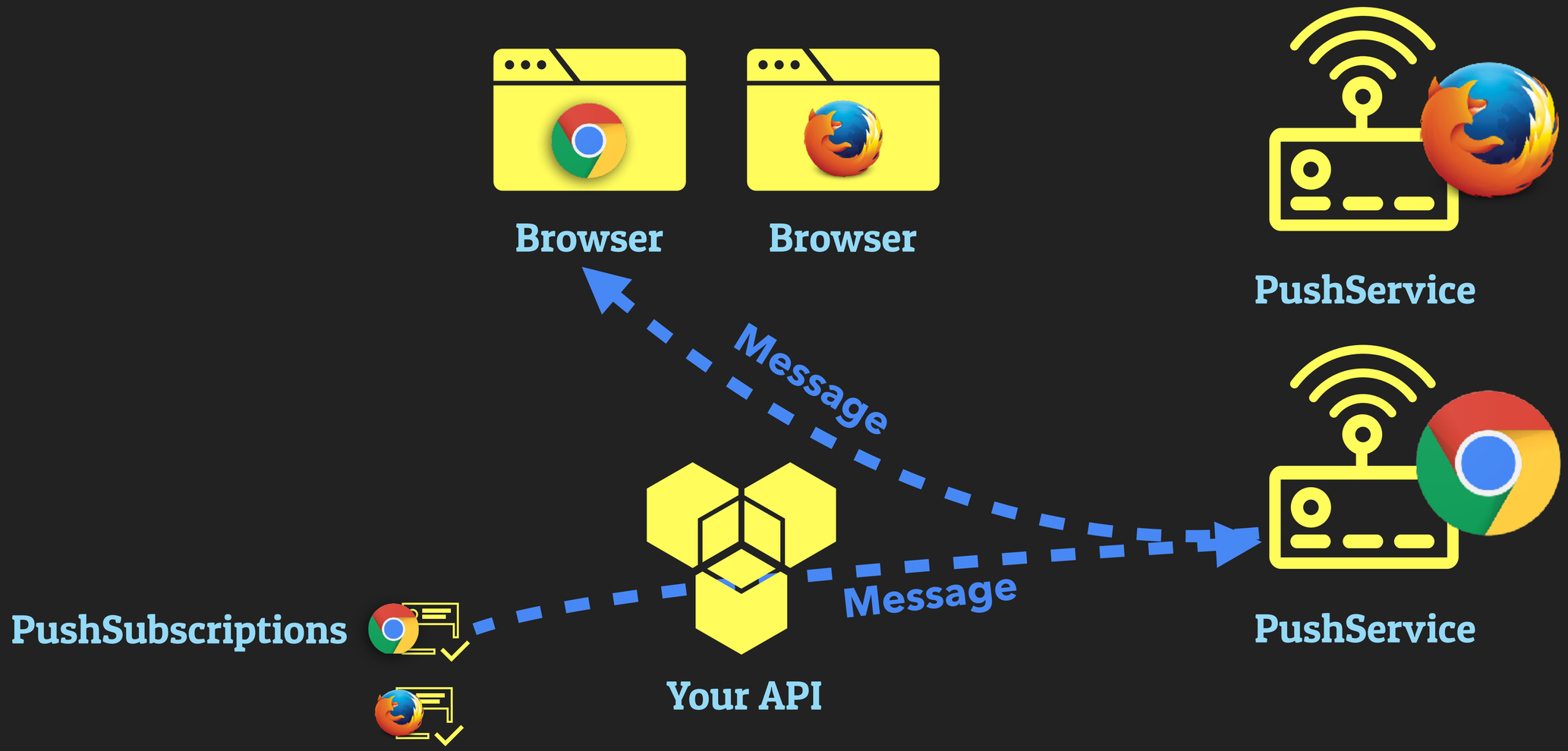
Web Push - High-Level Overview

- Provide the ability to send messages to a user's browser
- The user will receive the message regardless of whether the app is open or in the foreground
- Describes an open standard for a push service (that you don't have to worry about)
- Requires a service worker

Web Push - Storing a PushSubscription for each browser



Web Push - Sending a web push message



Web Push - Asking the user for a PushSubscription

- Requires a ServiceWorkerRegistration
  - Can happen at the time the SW is registered, or in a SW event handler
  - Requires same permission as Notifications

```
// app.js
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('./sw.js')
    .then((registration) => {
      // Ask for a push subscription
    })
}
```

Web Push - Asking the user for a PushSubscription

```
// app.js
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('./sw.js')
    .then((registration) => {
      // Ask for a push subscription
      let subscribeOptions = {
        userVisibleOnly: false,
        applicationServerKey: urlBase64ToUint8Array(
          'BE ... 5o'
        )
      };
      return registration.pushManager.subscribe(subscribeOptions);
    })
}
```

VAPID

Voluntary Application Server Identification for Web Push

- Nice for the push services
- Helps them distinguish between legitimate and bogus traffic (i.e., DDOS attacks)
- You SHOULD use VAPID
- Not about securing the rights to send a message to a user (that's the PushSubscription's purpose)

```
function urlBase64ToUint8Array(base64String) {
  const padding = '='.repeat((4 - base64String.length % 4) % 4);
  const base64 = (base64String + padding)
    .replace(/-/g, '+')
    .replace(/_/g, '/');

  const rawData = window.atob(base64);
  const outputArray = new Uint8Array(rawData.length);

  for (let i = 0; i < rawData.length; ++i) {
    outputArray[i] = rawData.charCodeAt(i);
  }
  return outputArray;
}
```

Generating VAPID keys

```
npm install -g web-push
web-push generate-vapid-keys
```

- Public key is used to generate an appropriate PushSubscription (in the browser)
- Private key is used to send a push message from your back end to the PushService

Web Push - What does a PushSubscription look like?

```
JSON.stringify(subscription);
{
  endpoint: 'https://android.googleapis.com/gcm/send/f1LsxkKph ... YeZ8kq_A',
  keys: {
    p256dh: 'BLc4xRzKlKORKWlbdgFaBrrPK3yd ... Wk8udnW3oYhIO4475rds=',
    auth: '5I2Bu2oKdyy9CwL8QVF0NQ=='
  }
};
```

Push Server Endpoint

Public Key / Authentication Secret

Web Push - Back End & Subscriptions

- Users may have more than one push subscription PER BROWSER.
- You won't be notified about users "unsubscribing" or "expiring" until you attempt to send a push event
- This is fine - Browser vendors ensure your users get no more than one notification per browser (i.e., one for Chrome, one for Chrome Canary)

Web Push - Sending a notification

- You do not want to do this without a back end library of some sort
- Debugging problems in this process is profoundly painful
- This pain is not unique to web push – most services like these have poor developer visibility

NODE.JS
github.com/web-push-libs/web-push

JAVA
github.com/web-push-libs/webpush-java

C#
github.com/web-push-libs/web-push-csharp

PHP
github.com/web-push-libs/web-push-php

Web Push - Receiving a Message

- Another event in your service worker

```
self.addEventListener('push', (event) => {
  let eventData = event.data.text();
  /* Do something with the data */
});
```

```
PushEvent {
  data: PushMessageData
}
```

Web Push

12

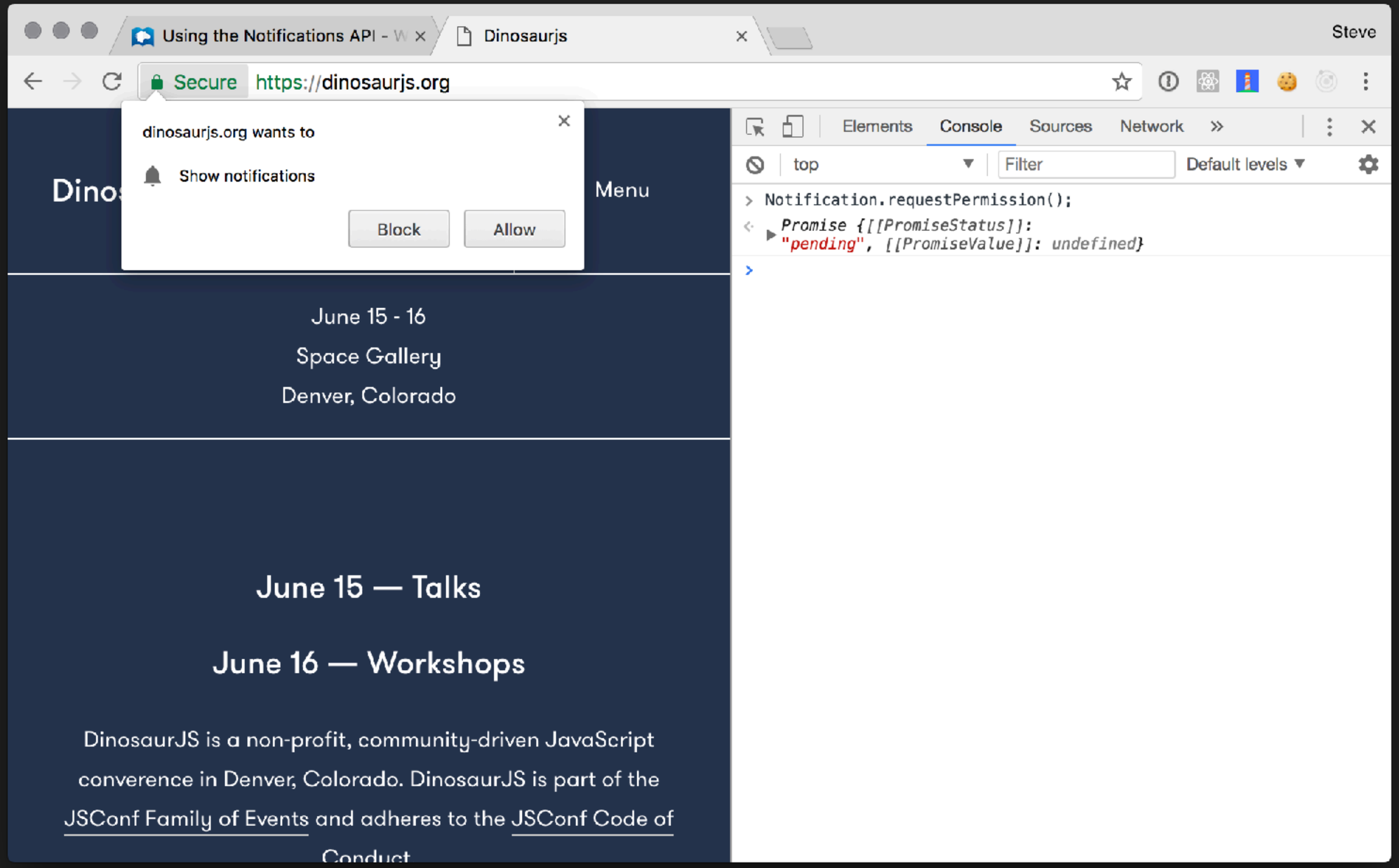
- Generate a web push subscription immediately following service worker registration
- Save this subscription to our API
  Hint: `POST https://localhost:3100/api/push-subscription`
- Now, after placing an order, you should start to receive push messages
- Implement a push event handler that unregisters the service worker in the event that it receives a message that's the string "TERMINATE"

```
self.registration.unregister().then((didUnregister) ⇒ { ... });
```

Notifications

```
Notification.requestPermission().then(permission ⇒ {
  console.log(permission);
});
```

- Notification's require a user's permission.
- Requesting permission returns a promise.
- Permission is either "granted" or "denied" in the promise.
- You can check the permission later using Notification.permission.
- Notification.permission will be "default" if you haven't prompted the user.



```
if ('Notification' in window) {
  // Now do your notification stuff…
}
```

- Let’s do some feature detection to make sure we’re not calling methods on objects that don’t exist.

```
if ('Notification' in window) {
  if (Notification.permission === 'default') {
    Notification.requestPermission().then(permission => {
      const notification = new Notification(
        'Thanks for granting permission!',
        {
          body: 'We promise not to abuse it.',
        }
      );
    });
  }
}
```

```
if (Notification.permission === 'granted') {
  const notification = new Notification(
    'Hello Frontend Masters',
    {
      body: 'PWAs are pretty sweet.',
      icon: 'important-icon.png'
    }
  );
}
```

```
if (Notification.permission === 'denied') {
  alert('You will never escape me');
}
```

```
self.addEventListener('activate', event ⇒ {
  const notification = new Notification();
  // You will never hear from me.
});
```

```
self.addEventListener('activate', event ⇒ {
  self.registration.showNotification('Hello World');
});
```

Notification Options

- Notifications take two arguments
  - A string for the **title**
  - An object full of **options**

Notification Options

- The options fall into three main categories
  - Visual options
  - Behavioral options
  - Informational options

Visual Options

- body (string, effectively the subtitle of the notification)
- icon (URL, a small image to go along with your notification)
- image (URL, a larger image)
- badge (URL, only used on Chrome for Android)
- vibrate
- sound
- dir (for left-to-right or right-to-left text, “auto” is also a valid option)

Behavioral Options

- tag (a short string, used for grouping notifications)
- data (anything)
- requireInteraction (boolean)
- renotify (boolean, forces the notification to stay visible)
- silent (boolean)

Talking about notifications...

- ...is kind hard to capture in slides.
- ...fairly boring to cover in slides.
- So, let's play around with them in a contrived safe space
  - https://notify-the-authorities.glitch.me

Notifications

13

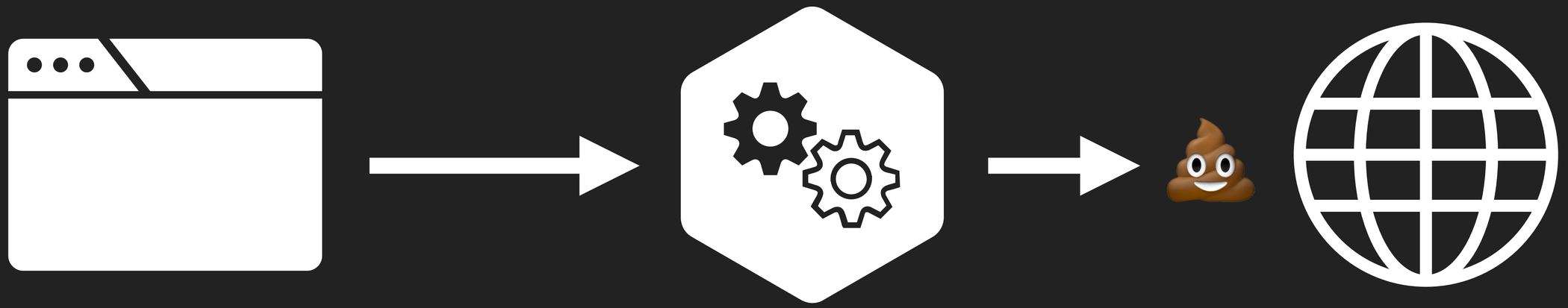
- You may notice that some of the web push payloads you're receiving are JSON objects with a top-level key: "notification". These are intended to become push notifications!
- Using the Notification API, from within your service worker, create a new notification when a web push payload that looks like this is received
- Handle the Service Worker's notificationClick event to open a new window, using the code on the next page

Notifications

13

```
self.addEventListener('notificationclick', function(event) {
  console.log('On notification click: ', event.notification.tag);
  event.notification.close();

  // This looks to see if the current is already open and
  // focuses if it is
  event.waitUntil(self.clients.matchAll({
    type: 'window'
  }).then(function(clientList) {
    for (var i = 0; i < clientList.length; i++) {
      var client = clientList[i];
      if (client.url == 'https://localhost:3000/' && 'focus' in client)
        return client.focus();
    }
    if (self.clients.openWindow)
      return self.clients.openWindow('https://localhost:3000/');
  }));
});
```

Background Sync



All Internet is Not Equal

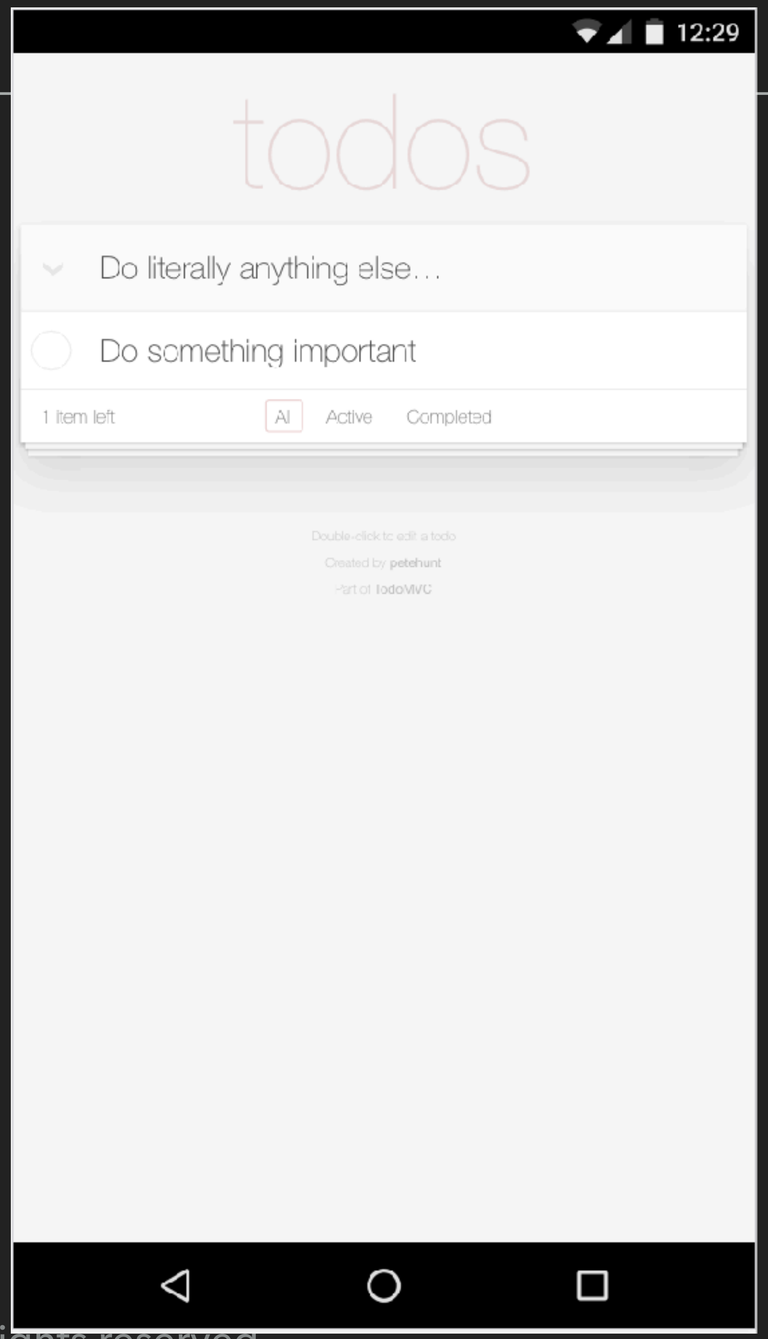
- What happens when you think you're connected to the Internet, but your connection is flaky.
- You think you did something, but… nope, it was dropped.
- Yea, you want to try again later—but you're very busy moving and shaking.

If only we had some way to do stuff when the user wasn't actively using the page, right?

We could like do it later when we had a reliable connection, right?

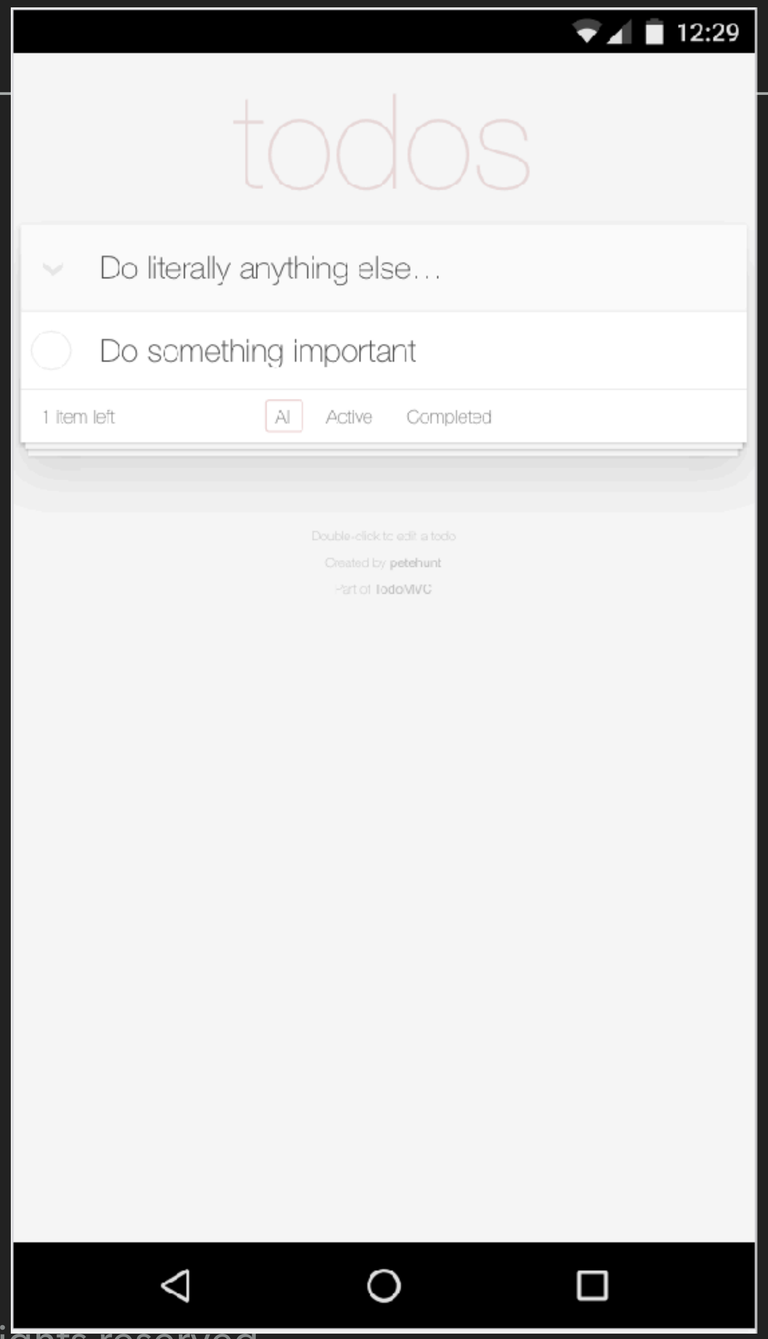
The Case of Lost Todos

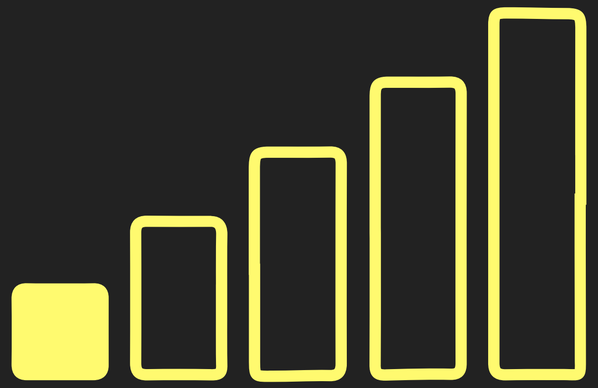
- Logan has been using a native Todo application for the last year and things have been going swimmingly.
- Whenever she is on the go and needs to remember to do something she jots it down and moves along.
- At the behest of her husband, who is all excited about the Open Web, she decides to give a web-based application a shot.



The Case of Lost Todos

- She picks one of the many identical todo apps on the web and goes on about her day.
- She takes for a swing at home and it works as expected.
- She checks her list on the subway and sees that the PWA even loads offline.
- Then she tries to add a new todo from outside of her son's school, which has notoriously spotty connection.

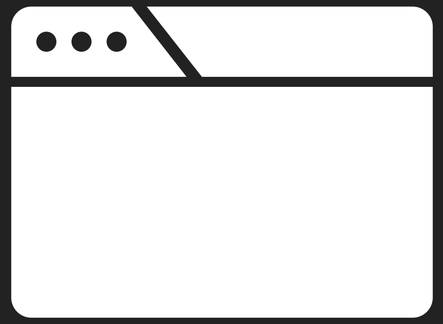




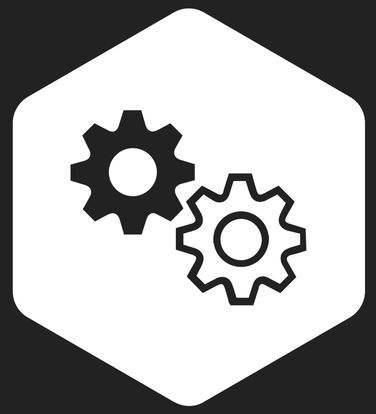
Background Sync

- Background Sync allows you to defer an action until you have a successful connection again.
- You can schedule an action.
- Your Service Worker will periodically try it out.
- If it works: awesome. 🤠
- If it doesn't work: try again later. ⏱

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.ready.then(registration ⇒ {
    registration.sync.register('your-event-name');
  });
}
```



```
self.addEventListener('sync', event => {
  if (event.tag == 'your-event-name') {
    event.waitUntil(handleYourEvent());
  }
});
```



Some Considerations

- You can't pass in any state when you register your background sync.
  - IndexedDB is a great choice as a place to store this data.
  - When your Service Worker attempts a sync, read the data out of IndexedDB.
- If you register multiple sync events with the same tag, the most recent will override the previous.
  - This could be a good thing.

Background Sync Strategies

- Just like caching, there is no one-size-fits-all solution for syncing
- In some situations, we might only want to schedule a sync if the network fails.
- In other situations, this might be out first choice.

Consider a Twitter Application

- If the Twitter application is offline and you like a tweet, the application will optimistically assume that everything is going to work out.
- It might keep a reference to each tweet that you liked in IndexedDB and replay your actions when it connects.
- It can do this silently, without alerting you.

Background Sync

14

- We've got a lot of offline functionality working, so it's possible that users may update their cart while offline. Using what we've learned about background sync, send the contents of the cart to our back end whenever connectivity permits.
- In the `CartStore` class defined in `./client/data/cart-store.js`, enhance the private function `_saveCart` so that it:
  - Persists the current cart into IndexedDB.
  - Registers a sync events.
  - Listens for that sync event in the Service Worker and fires out the request to the API.

The joys of appcache

Service Worker

ServiceWorker enthusiasm

The first thing any implementation needs.

Chrome: Shipped.

Firefox: Shipped.

Samsung Internet: Shipped. Based on Chromium 44.2403 with some additions and changes. (See "Service Worker" section.)

Safari: Under consideration, Brief positive signals in five year plan.

Edge: In development.

Support does not include iOS versions of third-party browsers on that platform (see Safari support).

Extra Credit

- When you get home, go ahead and do a Google search for "jake archibald appcache"

Some Tasting Notes

- The HTML page itself isn't listed in the manifest. Pages that associate with a manifest become part of it.
- Anything in the AppCache is cache-only—even if you're on the network.
- The cache will only update if the text of the manifest has changed

Some Tasting Notes

- But, you could still get an version because removing something from AppCache doesn't necessarily remove it from the regular cache.
- The manifest could get cached in which part, you won't get the the new manifest anyway, which means you'll still have the cached data even though you changed the manifest.

Some Tasting Notes

- Non-cached assets will not load at all on a cached paged (unless you exempt them from the manifest) even if you're online
- There is no logic for presenting a fallback if you're offline. Those resources you excepted above will just fail to load.

Steve

Mozilla Foundation [US] https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache

MDN web docs
moz://a

Technologies ▾ References & Guides ▾ Feedback ▾

Sign in

‹ HTML

Languages Edit

Using the application cache

See also

- Events
  - cached
  - checking
  - downloading
  - noupdate
  - obsolete
  - updateready

In This Article

Deprecated

This feature has been removed from the Web standards. Though some browsers may still support it, it is in the process of being dropped. Avoid using it and update existing code if possible; see the compatibility table at the bottom of this page to guide your decision. Be aware that this feature may cease to work at any time.

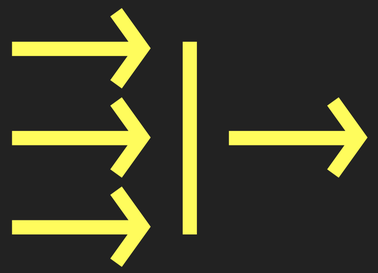
Using the *applicationcaching* feature described here is at this point highly discouraged; it's in the process of being removed from the Web platform. Use Service Workers instead. In fact as of Firefox 44, when AppCache is used to provide offline support for a page a warning message is now displayed in the console advising developers to use Service workers instead (bug 1204581).

Introduction

HTML5 provides an *application caching* mechanism that lets web-based applications run offline. Developers can use the

So, let's talk about it anyway.

```
CACHE MANIFEST
# v1 (Ideally, you can change this.)

CACHE:
app.css
app.js

NETWORK:
*

FALLBACK:
/download-virus.html /not-available.html
images/ images/offline.jpg
```



HTTP/2

What is HTTP/2

- An upgrade to the HTTP transport layer
- Fully multiplexed—send multiple requests in parallel
- Allows servers to proactively push responses into client caches

What's wrong with HTTP/1.1

- Websites are growing: more images, more JavaScript
- Sure, bandwidth has gotten a lot better, but roundtrip time hasn't
- It takes just as long to ping a server now as it did 20 years ago

What's wrong with HTTP/1.1

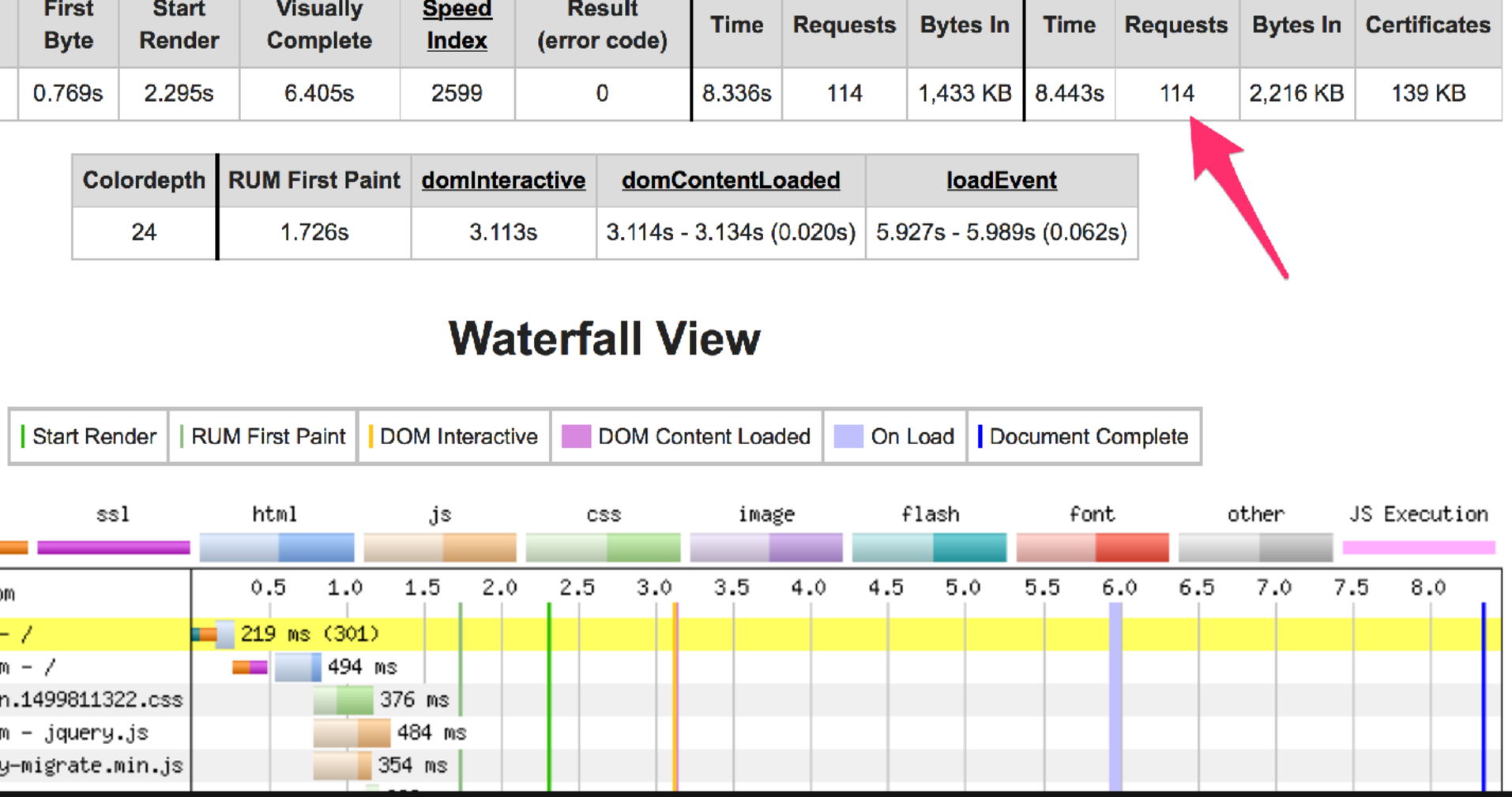
- That's right: one file at a time per connection
- No big deal. It's not like we are building websites that request 100 files to something.

www.webpagetest.org/result/170722_NX_1FXD/1/details/#waterfall_view_step1

Custom Metrics

						Document Complete			Fully Loaded			
Load Time	First Byte	Start Render	Visually Complete	Speed Index	Result (error code)	Time	Requests	Bytes In	Time	Requests	Bytes In	Certificates
8.336s	0.769s	2.295s	6.405s	2599	0	8.336s	114	1,433 KB	8.443s	114	2,216 KB	139 KB

Colordepth	RUM First Paint	domInteractive	domContentLoaded	loadEvent
24	1.726s	3.113s	3.114s - 3.134s (0.020s)	5.927s - 5.989s (0.062s)



What's wrong with HTTP/1.1

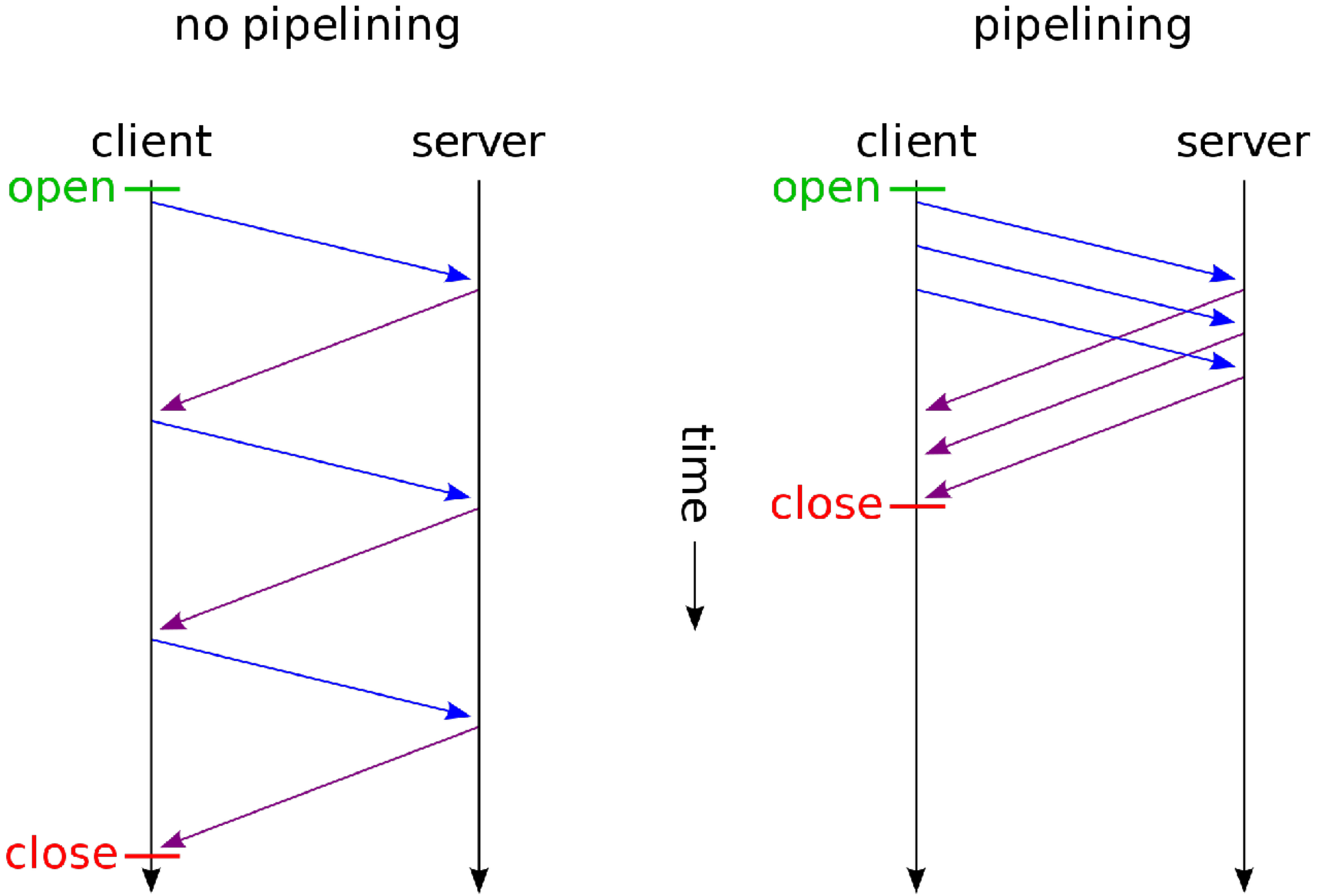
- We have a bunch of hacks to get around this:
  - Reusing connections
  - Creating off multiple (up to six) connections at the same time
  - Sharding our assets across multiple servers (in order to send more than six in parallel)
  - Combining images into sprites

What's not different?

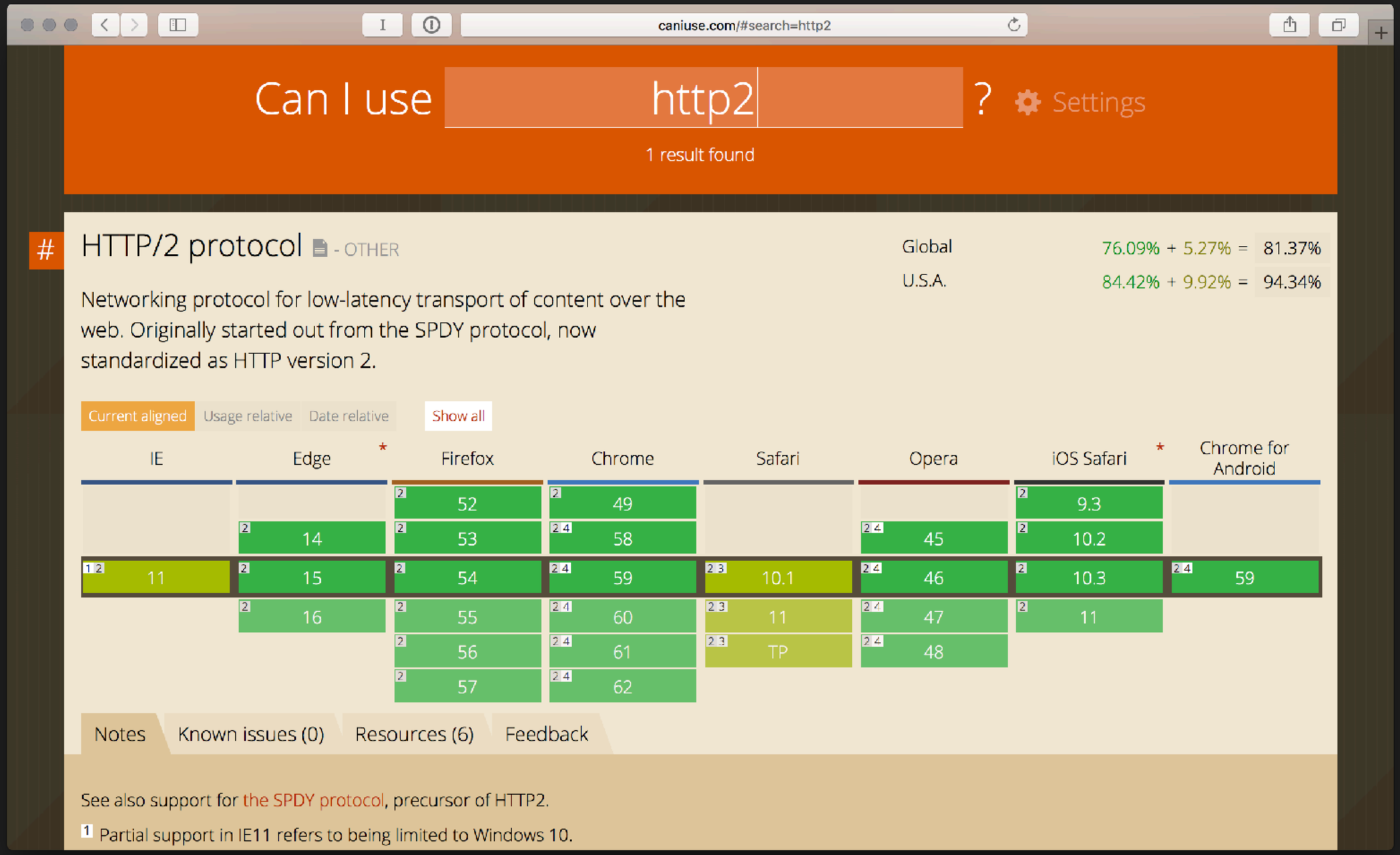
- The fundamentals are all still the same
  - Same "HTTP methods" (GET, POST, PUT, etc.) that you know and love
  - Same headers
  - Same paths
  - Same semantics

What is different?

- Multiplexing: multiple files can be sent simultaneously over a single connection.
- A well configured server can know about what other files you will need based on the one you asked for and start sending it to you before you even ask.



This sounds too good to be true. Okay, so what about browser support?



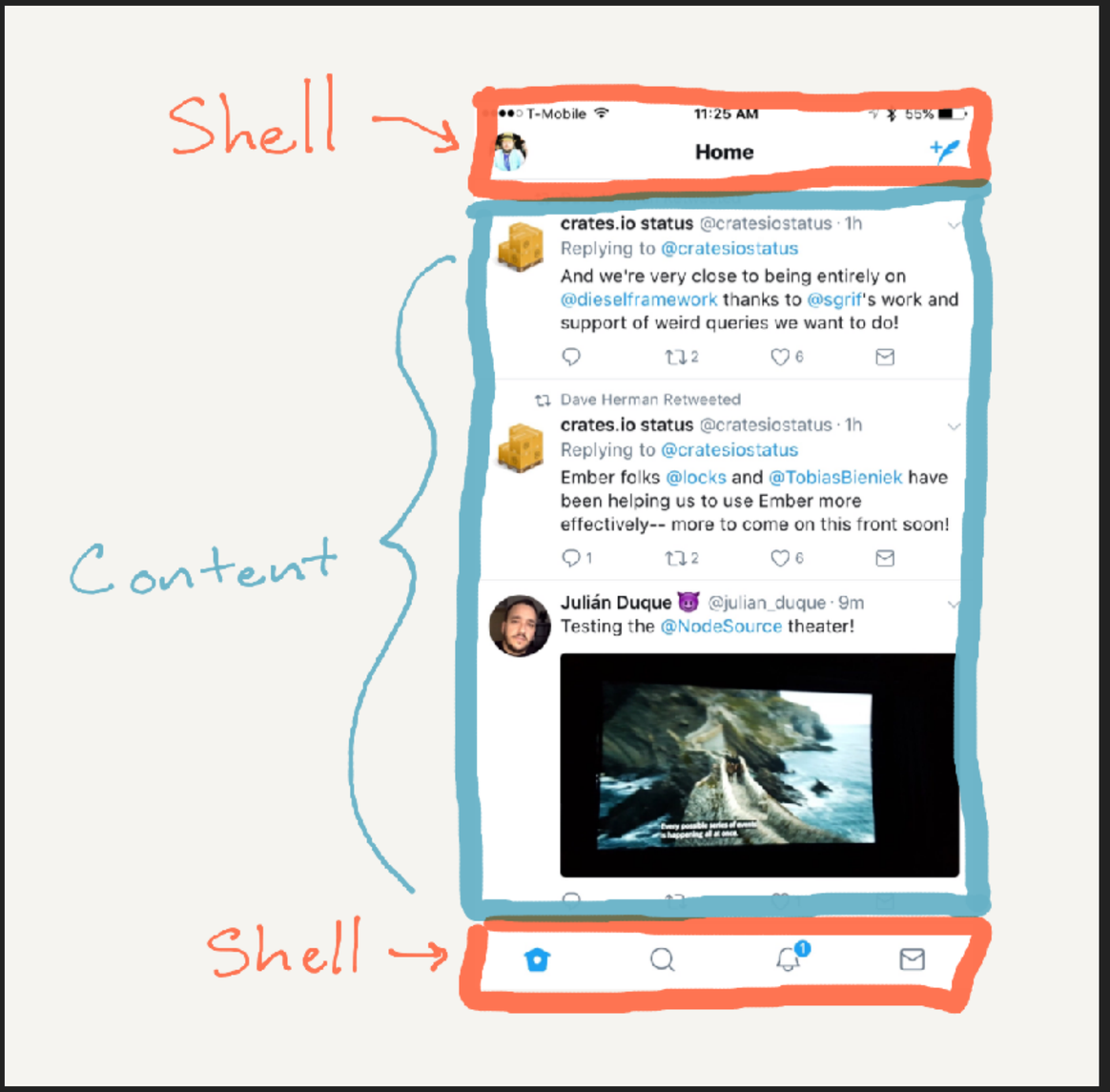
Patterns for PWAs

What is the Application Shell Pattern?

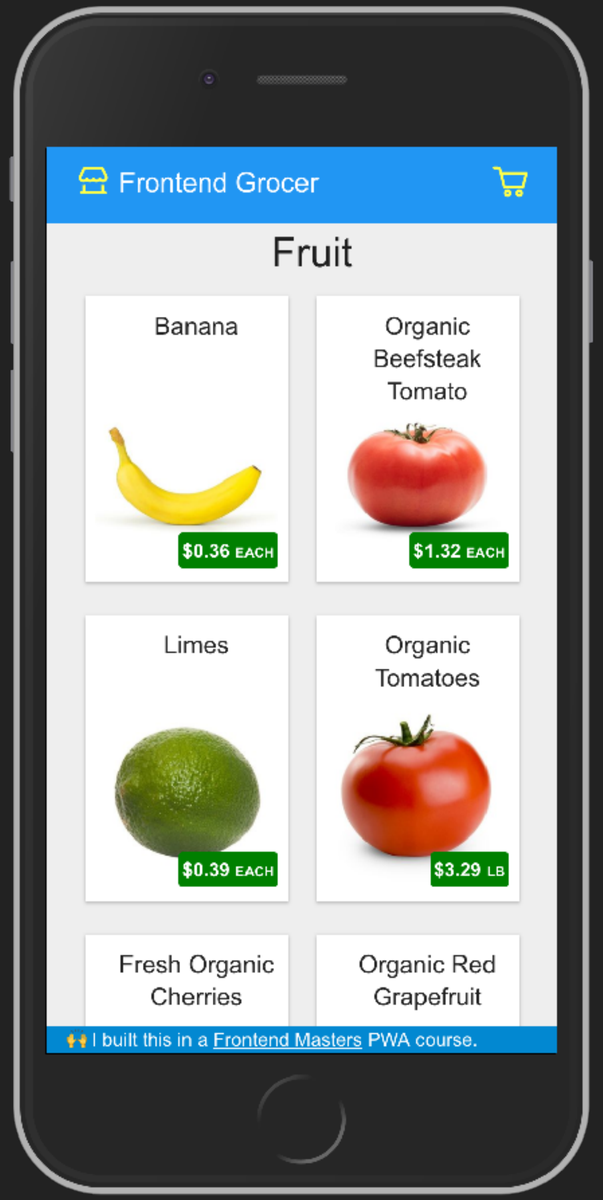
- As you might have guessed from the title of this section, it's a architectural pattern more than it is a technology.
- It's based in the idea that your application has two distinct components:
  - The user interface elements of your application.
  - The content displayed within your user interface.

The Application Shell Pattern

- Think about Twitter for a second.
- In a native application, you would already have the basic UI, you just need to go get some tweets.
- But, a traditional web application would have to fetch HTML, CSS, JavaScript, and images along with the content to build the UI.
- The App Shell pattern makes PWAs more like native apps in this regard.



Let’s squint at our application, where is the line between the shell and the content?



You might already be using an application shell without even thinking about it.

Pro tip: cache the application shell using a Service Worker!

PRPL Looks Like an Acronym. What does it stand for?

- **Push** critical resources for the route that is being requested
- **Render** the initial route
- **Pre-cache** the remaining routes
- **Lazy-load** the remaining routes when they are

PRPL

- The goal of PRPL is to
  - **minimize** the time-to-interactive
  - **maximize** caching efficiency

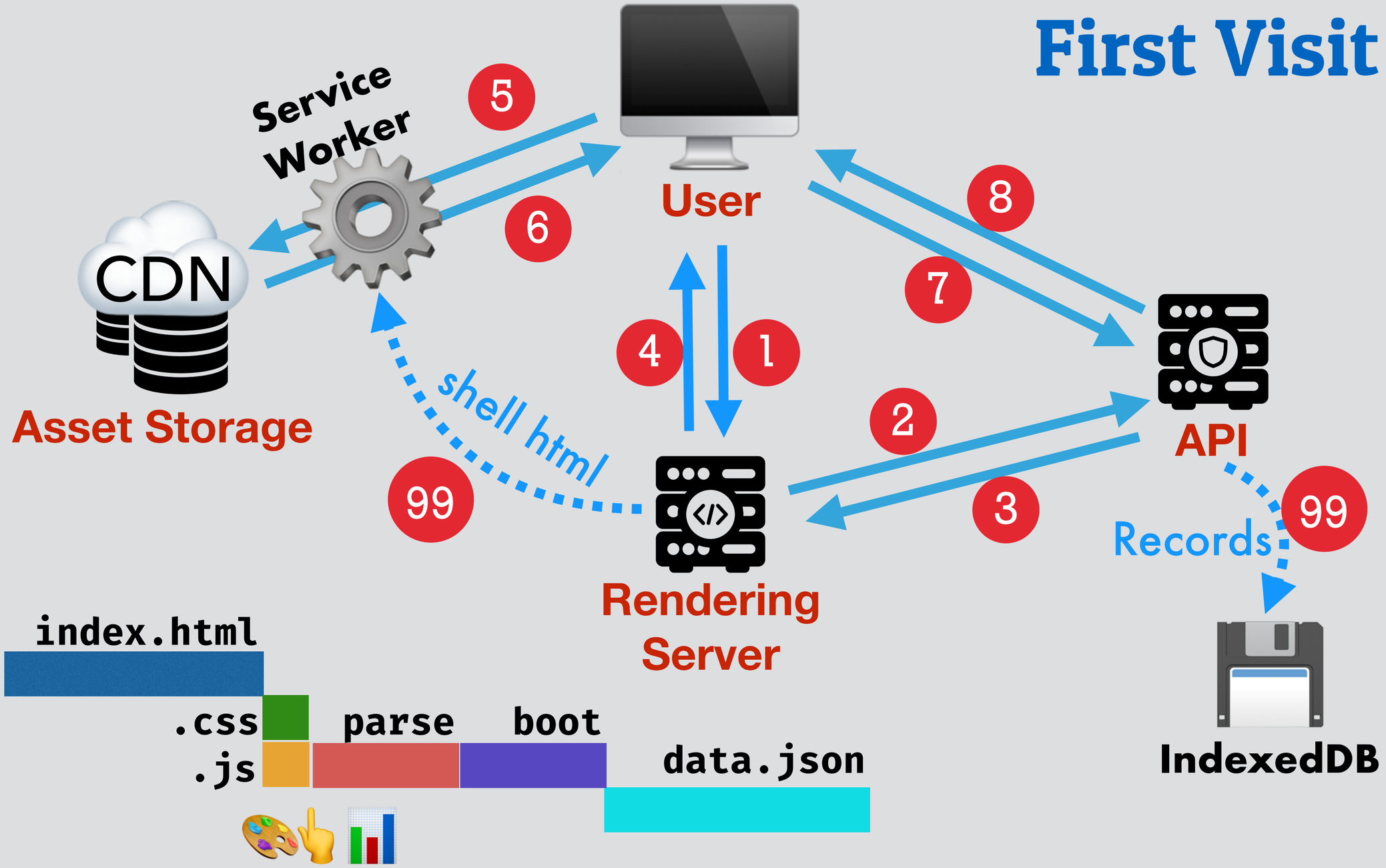
Server Rendered PWA v2

First Paint

Interactive

Data

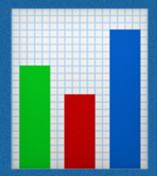
First Visit



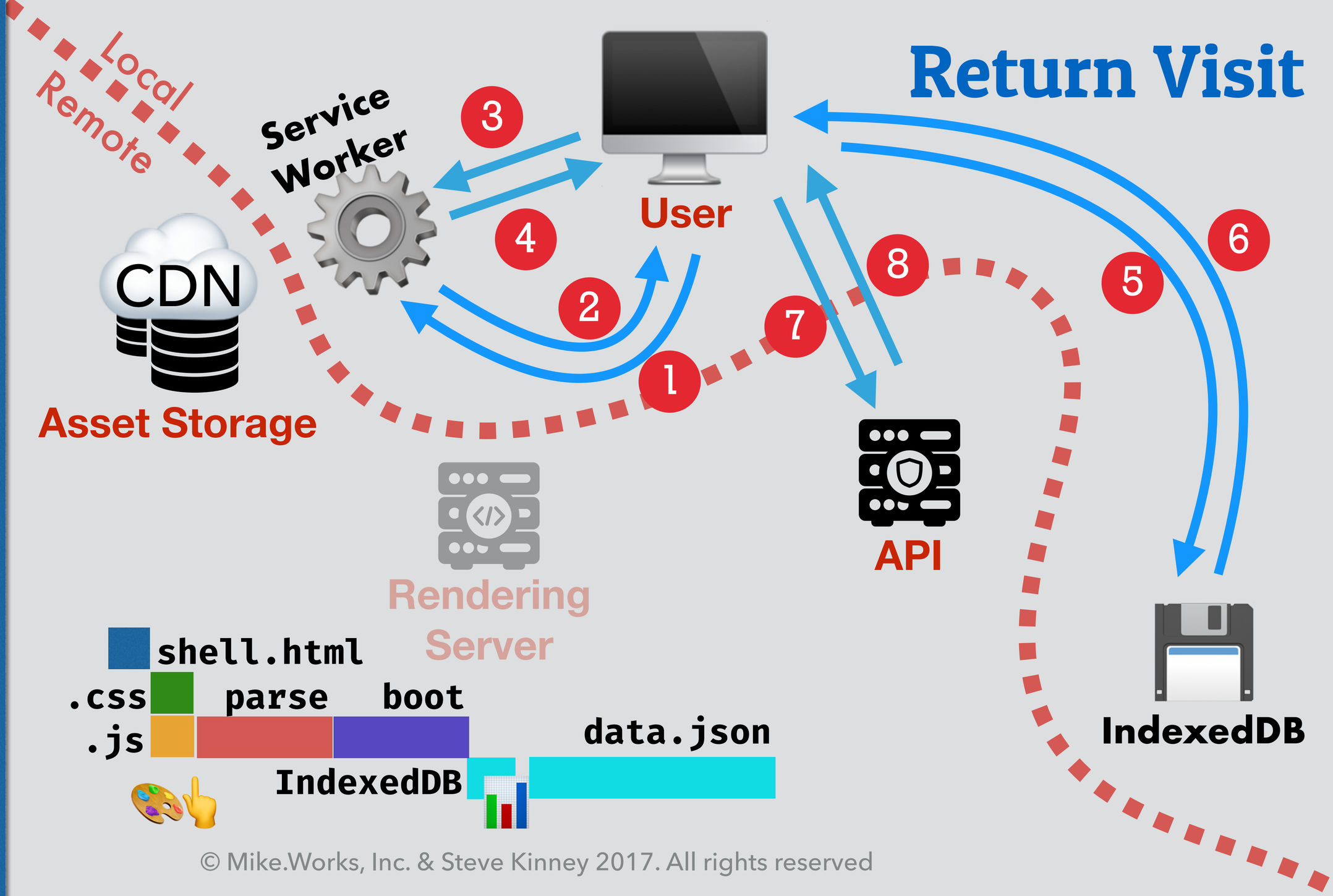
Server Rendered PWA v2

First Paint

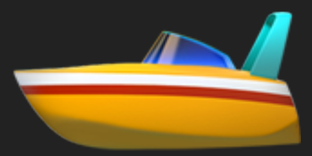
Interactive



Data



Return Visit

 Final Measurements

 How far we've come 

Metric	Before	After
Lighthouse	?	?
Gzipped app.js	?	?
Time to first paint	?	?