BEYOND PROGRESSIVE WEB APPS WITH **SERVICE WORKERS**

@TrentMWillis

#ThunderPlains

Atwood's Law

Atwood's Law

*"Any application that **can** be written in JavaScript,*
***will** eventually be written in JavaScript."*

blog.codinghorror.com/the-principle-of-least-power


Winamp2-js
Secure
https://webamp.org
WINAMP
02:26
1. CAKE - OPERA SINGER
128 kbps 44 kHz
mono stereo
EQ PL
SHUFFLE
WINAMP EQUALIZER
ON AUTO
PRESETS
+12 db
+0 db
-12 db
PREAMP 60 170 310 600 1K 3K 6K 12K 14K 16K
WINAMP PLAYLIST
1. Cake - opera singer 4:06
2. Cake - meanwhile, rick james 3:57
3. Cake - shadow stabbing 3:07
4. Cake - short skirt long jacket 3:24
5. Cake - commisioning a symphony 2:59
6. Cake - arco arena 1:31
7. Cake - comfort eagle 3:40
8. Cake - long line of cars 3:23
9. Cake - love you madly 3:57
10. Cake - pretty pink ribbon 3:08
11. Cake - world of two 3:40
ADD REM SEL MISC
4:06/36:56
02:26
LIST OPTS


DoppioJVM: A JVM in JavaScri
https://plasma-umass.org/doppio-demo/
Fork me on GitHub
DoppioJVM
A Java Virtual Machine written in 100% JavaScript.
Read the academic paper »
Browse the code »
Shell Demo
Ruby.js
Not Secure
ruby-in-the-browser.my-codeworks.com
Ruby.js
Fork me on GitHub
Ruby in the browser
This is a collection of demos of what you can currently do in the area of using ruby as the scripting language of the browser. As a replacement, kinda, of javascript if you will.
This is an area where a lot of people have done a lot of work over the years

17:42

AA flipkart.com

Flipkart Lite
Explore Plus

Login

Search for Products, Brands and More

Offer Zone | Mobiles | Fashion | Electronics | Home | Beaut


Fashion
FROM ₹199
Top brands


oppo
~~₹11,990~~ ₹6,990
A3s (Up to 4GB)


SALE CONTINUES
Electronics & Acc.

Deals of the Day
View All

En Flipkart is now available in Hindi

Continue in English | Try Hindi

17:41

AA mobile.twitter.com

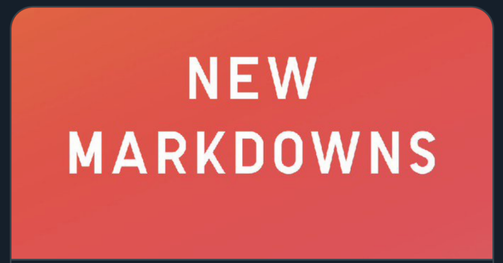
Home

David Pich Retweeted

Marco Rogers @polotek · Nov 5, 2014
Replying to @polotek
Social justice is about taking that empathy, that nuance of understanding different human needs, and using it to fix our institutions.

43 | 750 | 2.1K

UNIQLO @UniqloUSA
New markdowns just added! Shop the lowest prices of the season.



Shop Markdowns!
www.uniqlo.com

7 | 12 | 206

Promoted

17:43

AA starbucks.com

Exciting updates to Starbucks® Rewards »

Sign In | Join Now

Join Starbucks® Rewards today

Just sign up, keep drinking coffee, and you can get Rewards in as few as 2–3 visits.

JOIN NOW

PWA

PROGRESSIVE WEB APPS

BEYOND PROGRESSIVE WEB APPS WITH ***SERVICE WORKERS***

@TrentMWillis

Love The Web.
Love JavaScript.

JavaScript continues to *evolve*.

Web Hypertext Application Technology Working Group

WHATWG

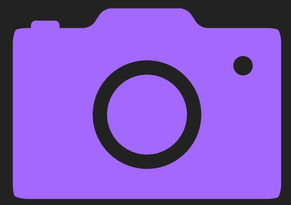
WHATWG Streams

WHATWG Streams

*"If installed inside the fetch hook of a service worker, this would allow developers to **transparently polyfill** new image formats."*

streams.spec.whatwg.org

Idea #1:
Transparently Polyfill New File Formats

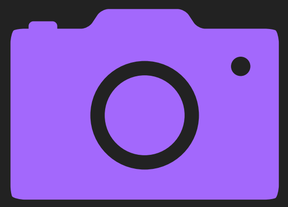


New Image Format

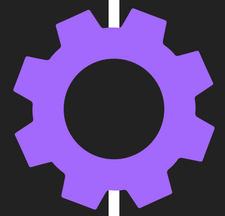
New Image Format

```
<img src="./my-image.nif">
```

**Can Display PNG/JPEG/GIF, Not NIF*

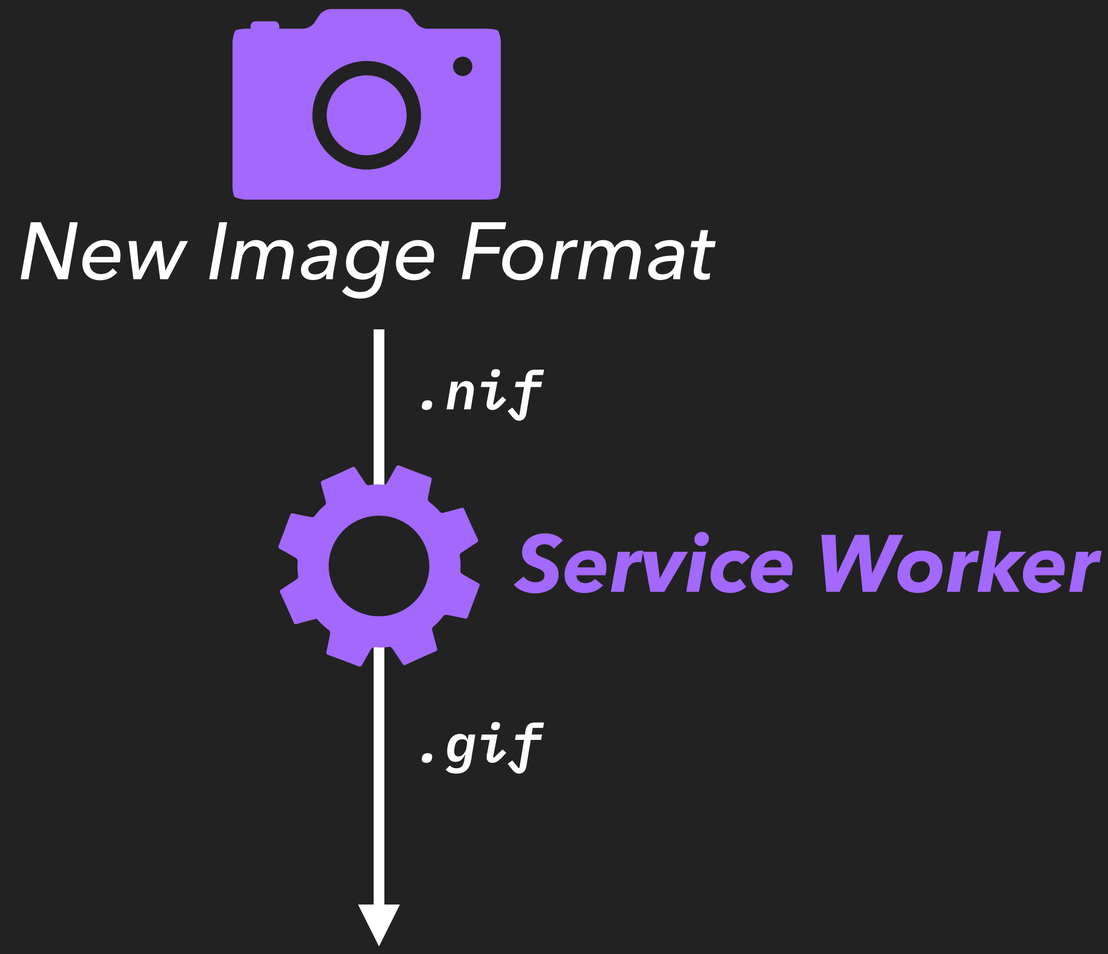


New Image Format

Proxy →  ***Service Worker***

```
<img src="./my-image.nif">
```

**Can Display PNG/JPEG/GIF, Not NIF*



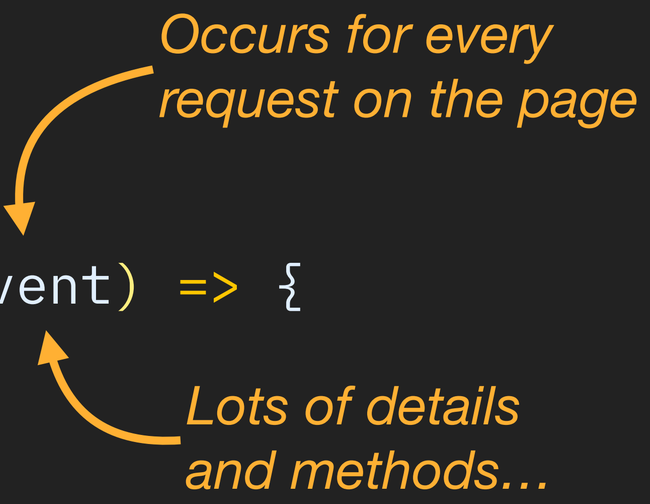

```
<img src="./my-image.nif">
```

*Can Display PNG/JPEG/GIF, Not NIF

All code will be available online

```
navigator.serviceWorker.register('./service-worker.js');
```

```
// service-worker.js
self.addEventListener('fetch', (event) => {
});
```



```
// service-worker.js
self.addEventListener('fetch', (event) => {

  event.respondWith();

});
```

Used to implement offline functionality

```
// service-worker.js
self.addEventListener('fetch', (event) => {

  event.respondWith(validFileFromPolyfilledFile(event.request));

});
```

```
// service-worker.js
self.addEventListener('fetch', (event) => {

  if (isPolyfilledFile(event.request)) {

    event.respondWith(validFileFromPolyfilledFile(event.request));

  }

});
```

```
const validFileFromPolyfilledFile = async (request) => {
};
```

```
const validFileFromPolyfilledFile = async (request) => {

  const originalResponse = await fetch(request);

};
```

```
const validFileFromPolyfilledFile = async (request) => {

  const originalResponse = await fetch(request);
  const originalData = originalResponse.blob(); // .text(), .json(), etc.

};
```

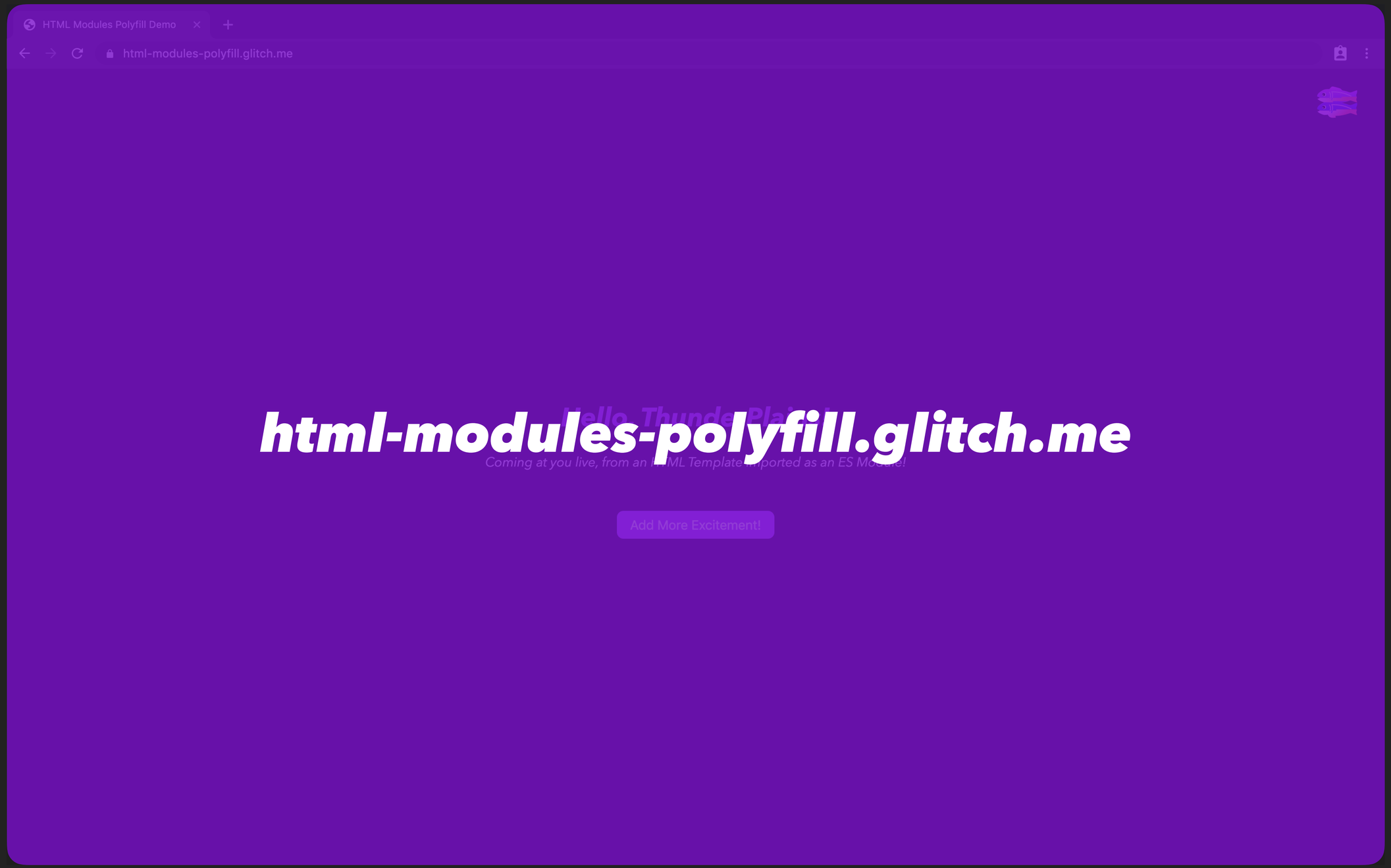
```
const validFileFromPolyfilledFile = async (request) => {

  const originalResponse = await fetch(request);
  const originalData = originalResponse.blob(); // .text(), .json(), etc.
  const modifiedData = polyfillTransform(originalData);

};
```

```
const validFileFromPolyfilledFile = async (request) => {

  const originalResponse = await fetch(request);
  const originalData = originalResponse.blob(); // .text(), .json(), etc.
  const modifiedData = polyfillTransform(originalData);
  const modifiedResponse = new Response(modifiedData);
  return modifiedResponse;

};
```


cryptogram-naive
cryptogram-naive.glitch.me
The image below is encrypted but is renderable by using a Service Worker.
cryptogram-naive.glitch.me

Polyfill new extensions to the web platform

Like importing HTML into JavaScript modules



html-modules-polyfill.glitch.me

Idea #2:
Compile Code At Runtime

Idea #2:
Compile **TypeScript** At Runtime

```
<script src="my-module.ts"></script>
```

```
<script src="my-module.ts" type="module"></script>
```

Cache API

Only compile changed files

```
const compileWithCache = async (response, compile) => {
};
```

```
const compileWithCache = async (response, compile) => {
  const cache = await caches.open('compile-cache');
};
```

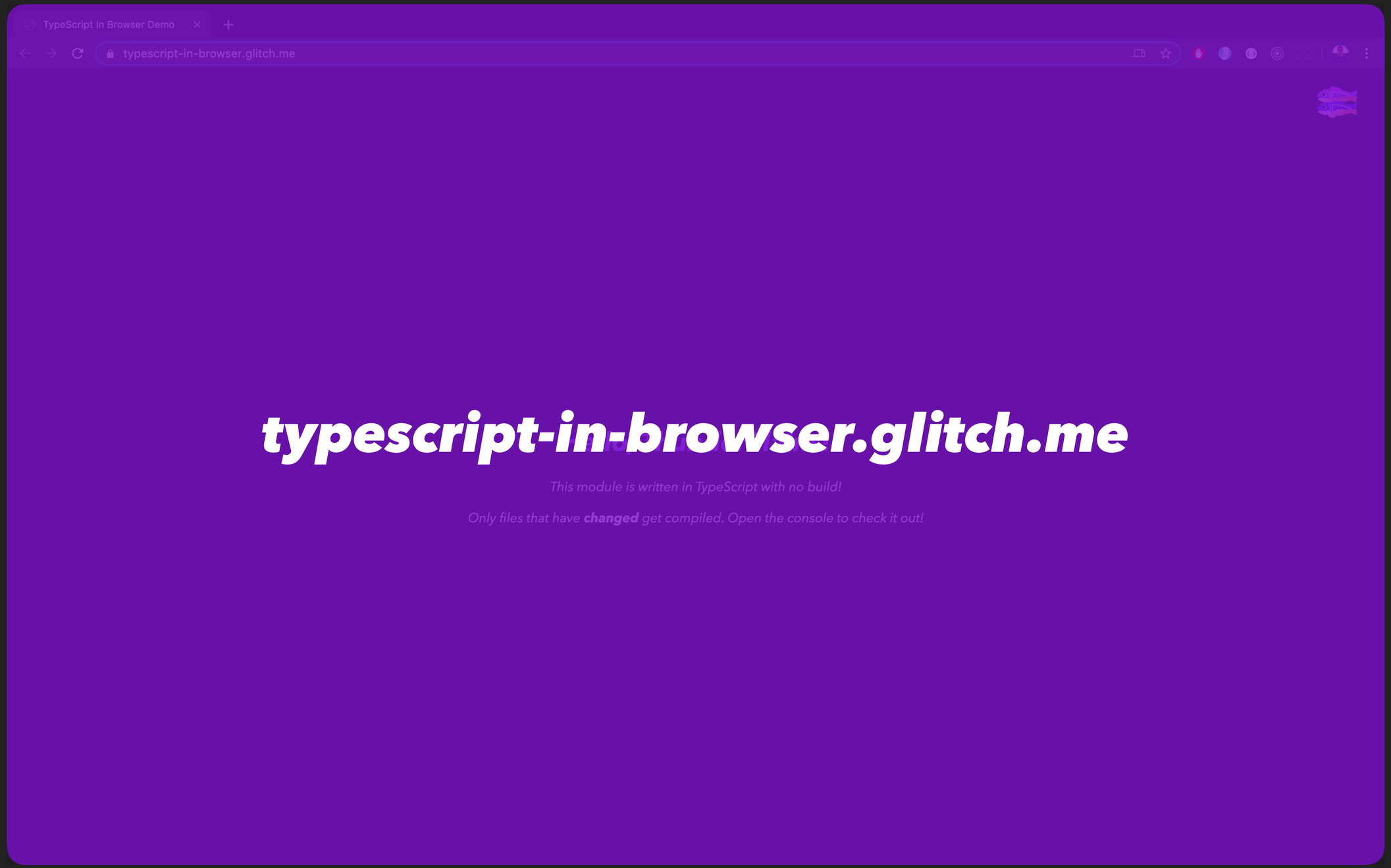
```
const compileWithCache = async (response, compile) => {

  const cache = await caches.open('compile-cache');

  if (response.headers.get('status') === '304') {
  }

};
```

```
const compileWithCache = async (response, compile) => {
  const cache = await caches.open('compile-cache');
  if (response.headers.get('status') === '304') {
    return cache.match(request.url);
  }
};
```

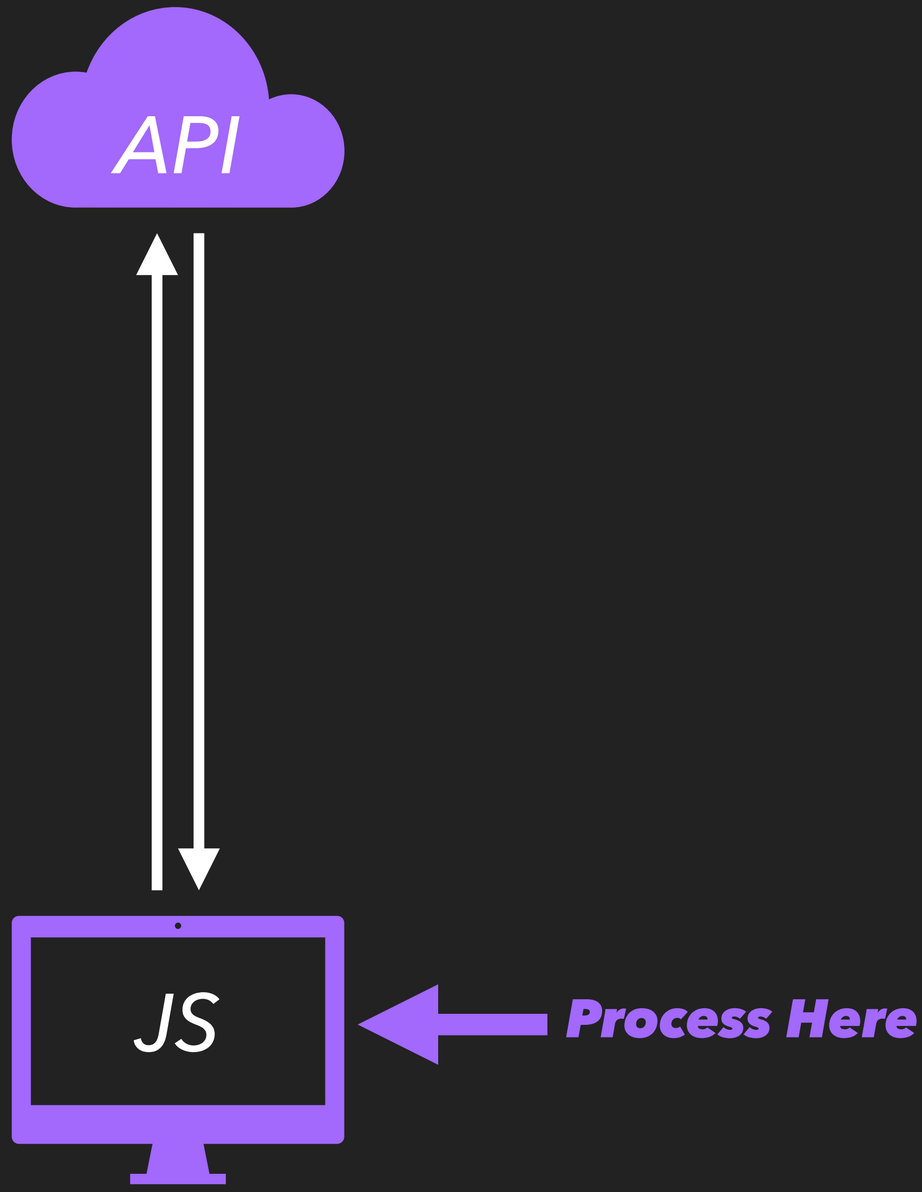
```
const compileWithCache = async (response, compile) => {
  const cache = await caches.open('compile-cache');
  if (response.headers.get('status') === '304') {
    return cache.match(request.url);
  }
  const compiledResponse = compile(response);
};
```

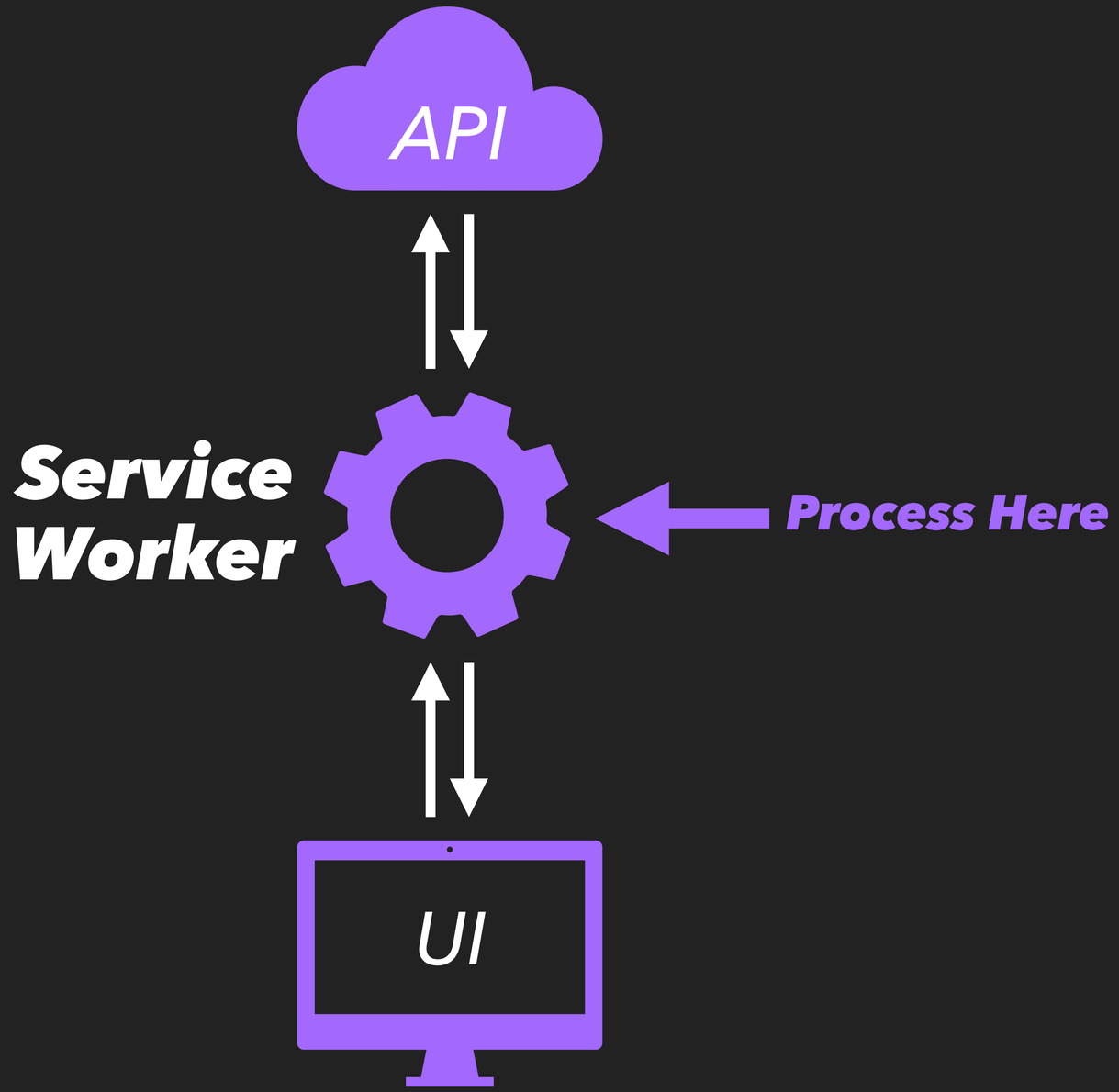
```
const compileWithCache = async (response, compile) => {
  const cache = await caches.open('compile-cache');
  if (response.headers.get('status') === '304') {
    return cache.match(request.url);
  }
  const compiledResponse = compile(response);
  cache.put(request.url, compiledResponse.clone());
};
```

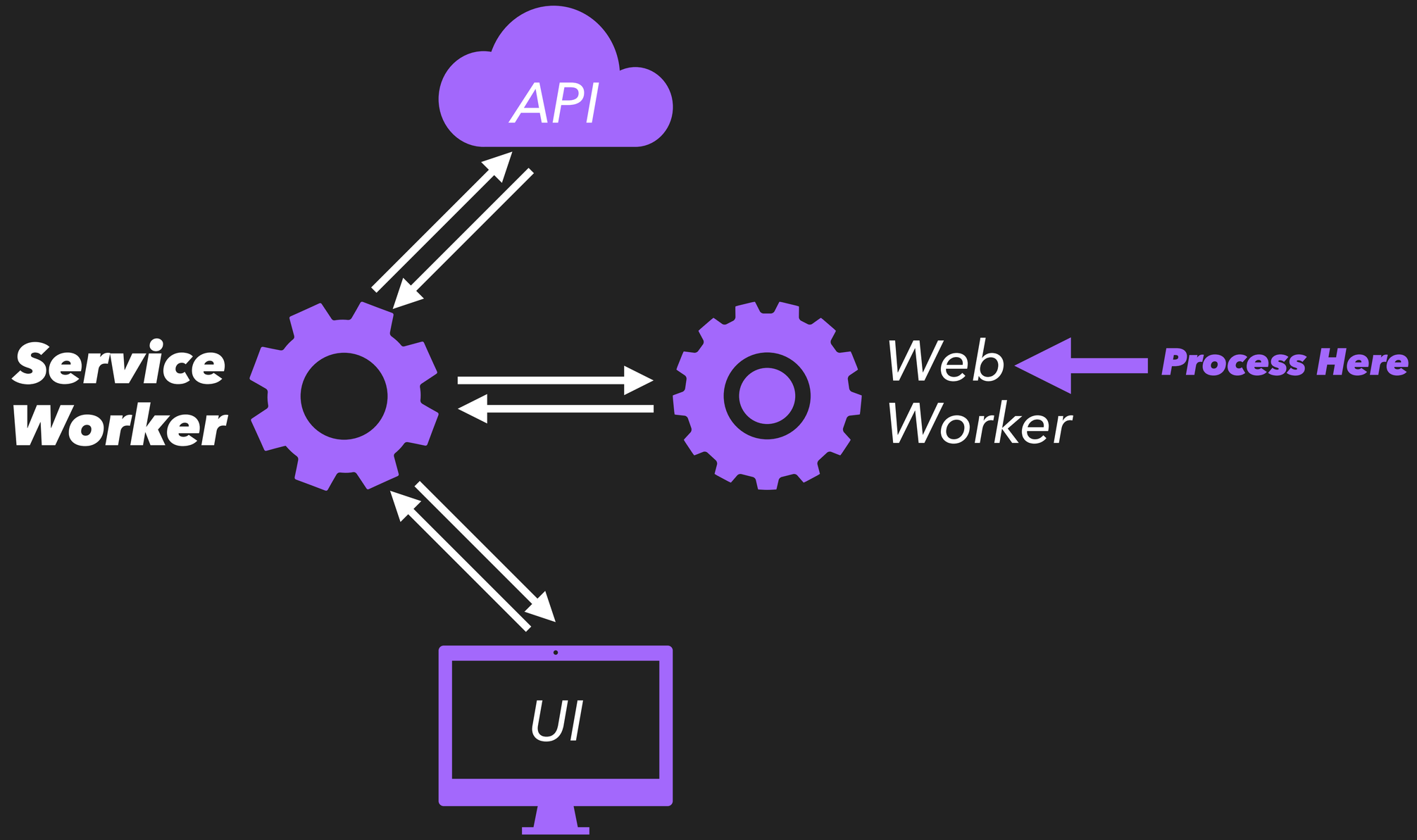
```
const compileWithCache = async (response, compile) => {
  const cache = await caches.open('compile-cache');
  if (response.headers.get('status') === '304') {
    return cache.match(request.url);
  }
  const compiledResponse = compile(response);
  cache.put(request.url, compiledResponse.clone());
  return compiledResponse;
};
```


TypeScript In Browser Demo
typescript-in-browser.glitch.me
typescript-in-browser.glitch.me
This module is written in TypeScript with no build!
Only files that have changed get compiled. Open the console to check it out!

Idea #3:
Process Data Off The Main Thread


API
JS
Process Here


API
Service Worker
Process Here
UI


API
Service Worker
Web Worker
Process Here
UI

The page makes a request…


```
const transform = async (request, clientId) => {
  const [response, port] = await Promise.all([
    fetch(request),
    getPortToWebWorkerFromClient(clientId)
  ]);
};
```

```
const getPortToWebWorkerFromClient = async (clientId) => {
  const client = await self.clients.get(clientId);
};
```

```
const getPortToWebWorkerFromClient = async (clientId) => {
  const client = await self.clients.get(clientId);
  return new Promise(resolve => {
  });
};
```

```
const getPortToWebWorkerFromClient = async (clientId) => {
  const client = await self.clients.get(clientId);
  return new Promise(resolve => {
    client.postMessage({});
  });
};
```

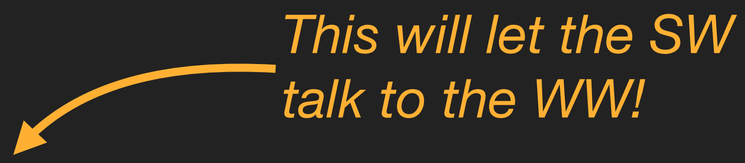
Ask the page how to communicate with the web worker

```
const webWorker = new Worker('web-worker.js');

navigator.serviceWorker.onmessage = () => {
};
```

```
const webWorker = new Worker('web-worker.js');

navigator.serviceWorker.onmessage = () => {

  const channel = new MessageChannel();

};
```

```
const webWorker = new Worker('web-worker.js');

navigator.serviceWorker.onmessage = () => {

  const channel = new MessageChannel();
  const serviceWorker = navigator.serviceWorker.controller;
  serviceWorker.postMessage({port: channel.port2}, [channel.port2]);

};
```

```
const webWorker = new Worker('web-worker.js');

navigator.serviceWorker.onmessage = () => {

  const channel = new MessageChannel();
  const serviceWorker = navigator.serviceWorker.controller;
  serviceWorker.postMessage({port: channel.port2}, [channel.port2]);
  webWorker.postMessage({port: channel.port1}, [channel.port1]);

};
```



```
const getPortToWebWorkerFromClient = async (clientId) => {
  const client = await self.clients.get(clientId);
  return new Promise(resolve => {
    client.postMessage({});
  });
};
```

```
const getPortToWebWorkerFromClient = async (clientId) => {
  const client = await self.clients.get(clientId);
  return new Promise(resolve => {
    self.onmessage = (msg) => {
      self.onmessage = null;
      resolve(msg.data.port);
    };

    client.postMessage({});
  });
};
```

Now we know how to talk to the Web Worker

```
const transform = async (request, clientId) => {
  const [response, port] = await Promise.all([
    fetch(request),
    getPortToWebWorkerFromClient(clientId)
  ]);
};
```

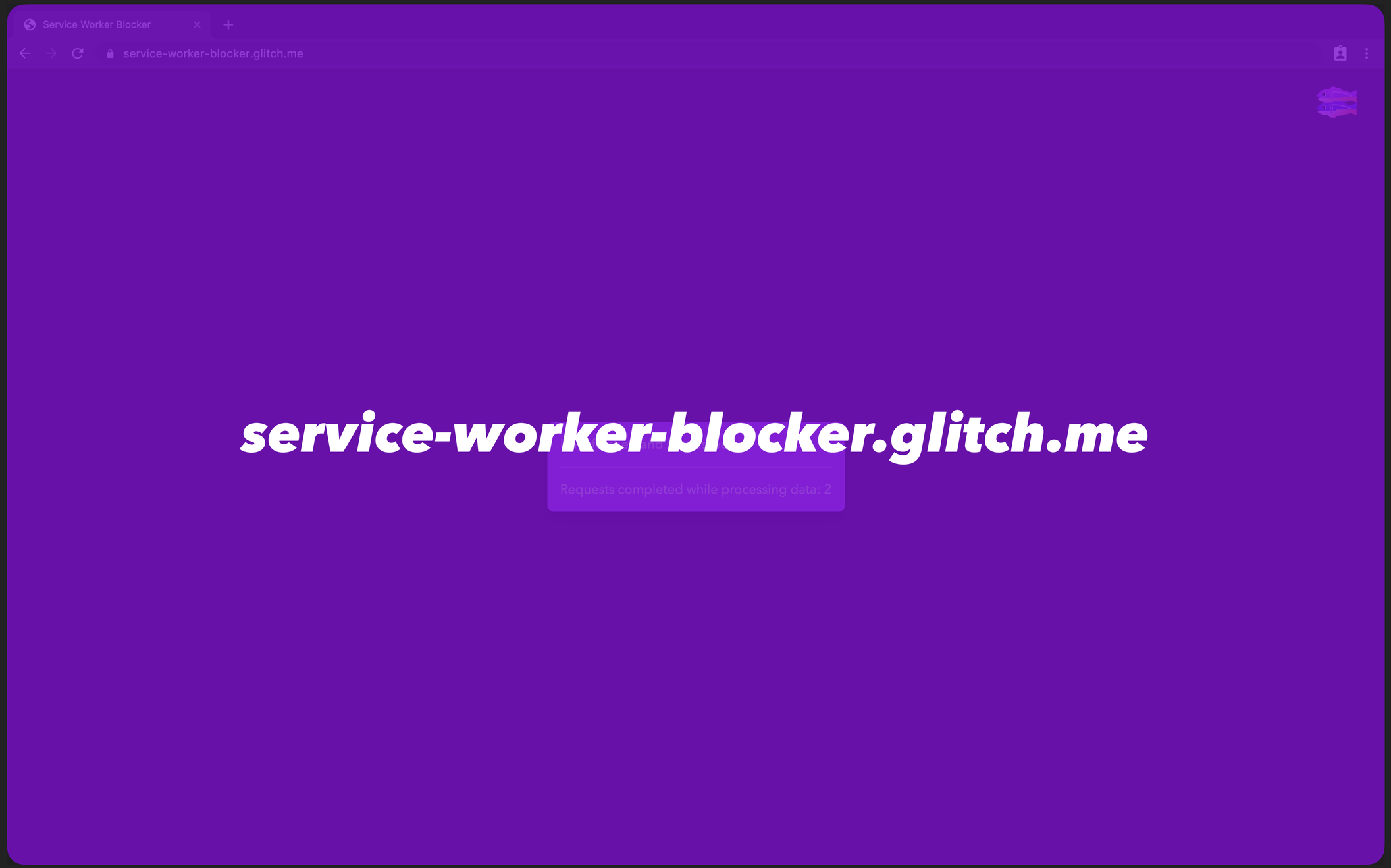
```
const transform = async (request, clientId) => {
  const [response, port] = await Promise.all([
    fetch(request),
    getPortToWebWorkerFromClient(clientId)
  ]);

  return new Promise(async (resolve) => {
  });
};
```

```
const transform = async (request, clientId) => {
  const [response, port] = await Promise.all([
    fetch(request),
    getPortToWebWorkerFromClient(clientId)
  ]);

  return new Promise(async (resolve) => {
    port.postMessage(await response.json());
  });
};
```

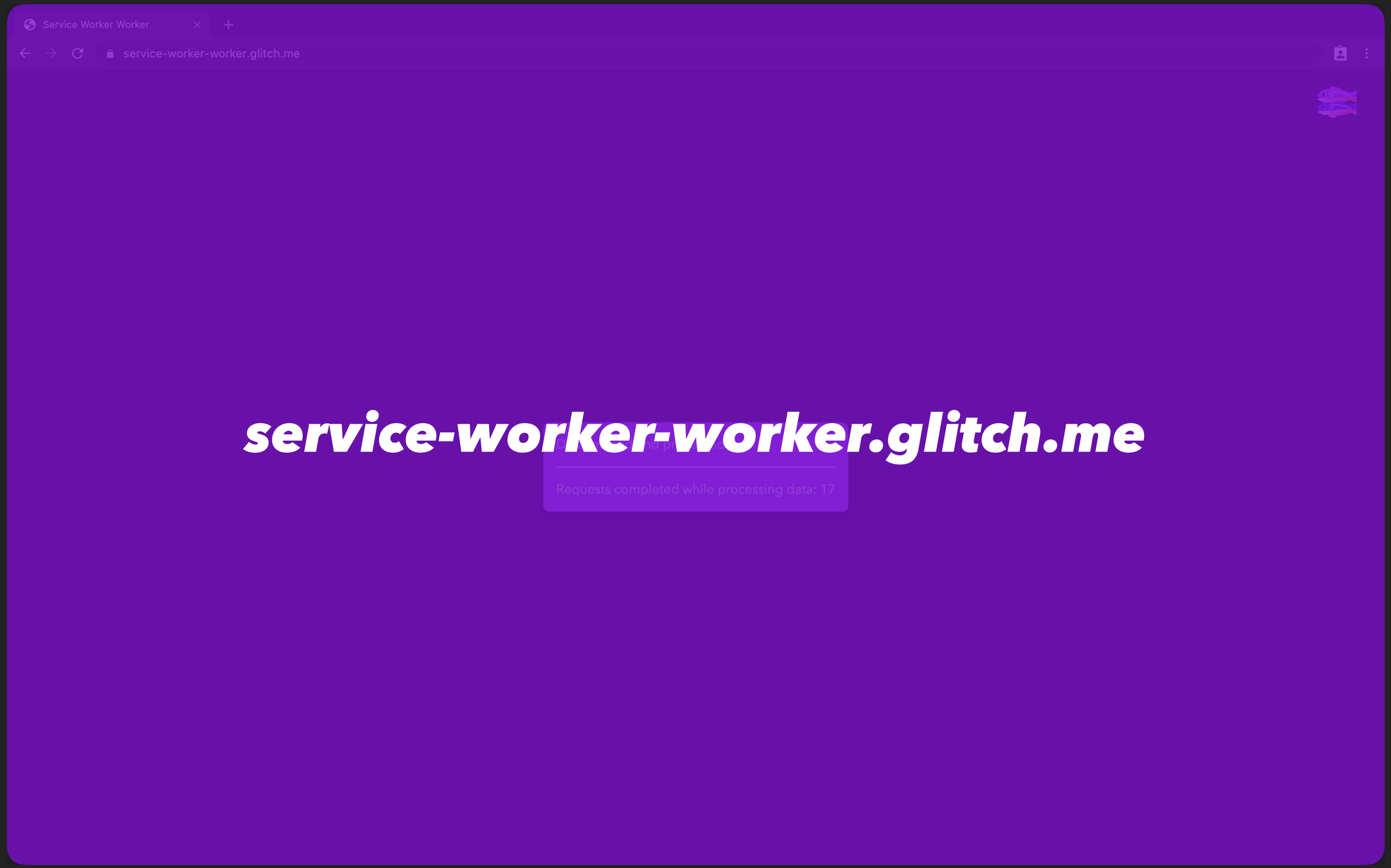
Send the data to the Web Worker

```
const transform = async (request, clientId) => {
  const [response, port] = await Promise.all([
    fetch(request),
    getPortToWebWorkerFromClient(clientId)
  ]);

  return new Promise(async (resolve) => {
    port.onmessage = (msg) => {
      resolve(new Response(`[${msg.data.toString()}]`));
    };
    port.postMessage(await response.json());
  });
};
```

Get the result back from the Web Worker



service-worker-blocker.glitch.me



service-worker-worker.glitch.me

Idea #4:
Process Data As It Streams

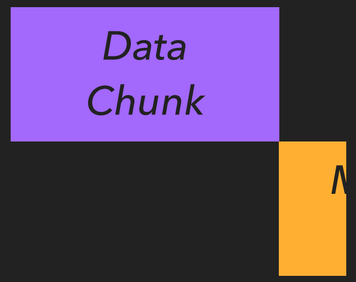
Original Data

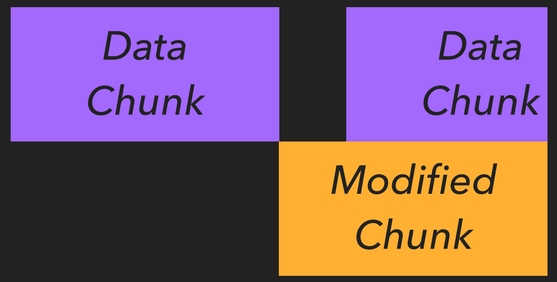

Original Data
Modified Data

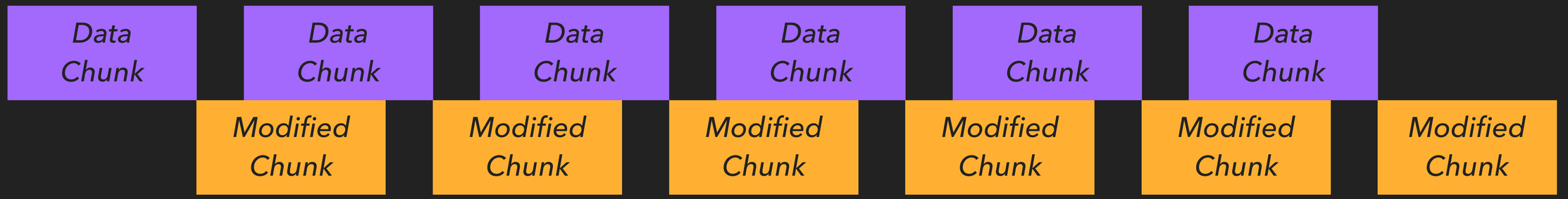
BATCH PROCESSING

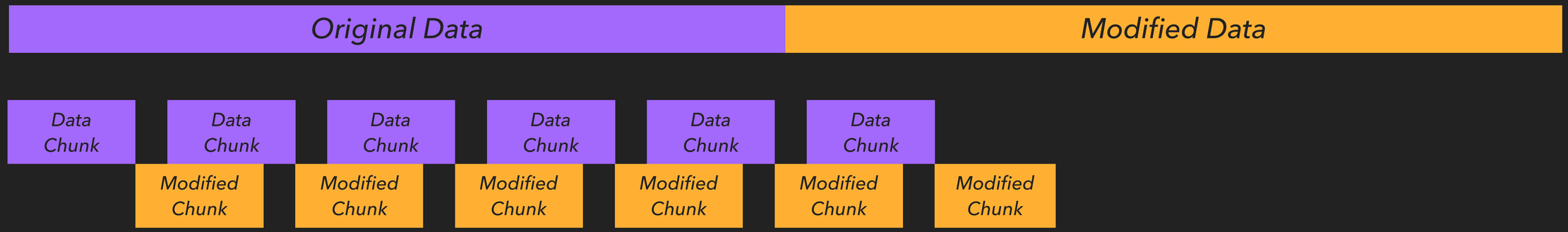
STREAM PROCESSING


Data
Chunk


Data
Chunk
M


Data
Chunk
Data
Chunk
Modified
Chunk

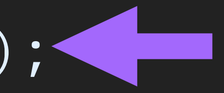

Data Chunk
Data Chunk
Data Chunk
Data Chunk
Data Chunk
Data Chunk
Modified Chunk
Modified Chunk
Modified Chunk
Modified Chunk
Modified Chunk
Modified Chunk


Original Data
Modified Data
Data Chunk
Data Chunk
Data Chunk
Data Chunk
Data Chunk
Data Chunk
Modified Chunk
Modified Chunk
Modified Chunk
Modified Chunk
Modified Chunk
Modified Chunk

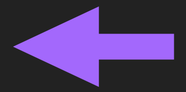
```
const validFileFromPolyfilledFile = async (request) => {

  const originalResponse = await fetch(request);
  const originalData = originalResponse.blob(); // .text(), .json(), etc.
  const modifiedData = polyfillTransform(originalData);
  const modifiedResponse = new Response(modifiedData);
  return modifiedResponse;

};
```

```
const validFileFromPolyfilledFile = async (request) => {;

  const originalResponse = await fetch(request);
  const transformStream = new TransformStream(new Transformer());
  const transformedBody = originalResponse.body.pipeThrough(
    transformStream
  );
  const transformedResponse = new Response(transformedBody);
  return transformedResponse;

};
```

```
const validFileFromPolyfilledFile = async (request) => {;

  const originalResponse = await fetch(request);
  const transformStream = new TransformStream(new Transformer());
  const transformedBody = originalResponse.body.pipeThrough(
    transformStream
  );
  const transformedResponse = new Response(transformedBody);
  return transformedResponse;

};
```

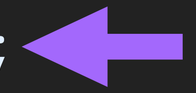


```
const validFileFromPolyfilledFile = async (request) => {;

  const originalResponse = await fetch(request);
  const transformStream = new TransformStream(new Transformer());
  const transformedBody = originalResponse.body.pipeThrough(
    transformStream
  );
  const transformedResponse = new Response(transformedBody);
  return transformedResponse;

};
```

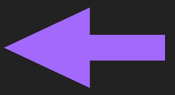


```
const validFileFromPolyfilledFile = async (request) => {;

  const originalResponse = await fetch(request);
  const transformStream = new TransformStream(new Transformer());
  const transformedBody = originalResponse.body.pipeThrough(
    transformStream
  );
  const transformedResponse = new Response(transformedBody);
  return transformedResponse;

};
```

```
const validFileFromPolyfilledFile = async (request) => {;

  const originalResponse = await fetch(request);
  const transformStream = new TransformStream(new Transformer());
  const transformedBody = originalResponse.body.pipeThrough(
    transformStream
  );
  const transformedResponse = new Response(transformedBody);
  return transformedResponse;

};
```



```
const validFileFromPolyfilledFile = async (request) => {;

  const originalResponse = await fetch(request);
  const transformStream = new TransformStream(new Transformer());
  const transformedBody = originalResponse.body.pipeThrough(
    transformStream
  );
  const transformedResponse = new Response(transformedBody);
  return transformedResponse;

};
```



```
class Transformer {
  async start() {}

  async transform() {}

  async flush() {}
}
```

cryptogram-streaming

cryptogram-streaming.glitch.me

The image below is encrypted but is renderable by using a Service Worker.

cryptogram-streaming.glitch.me

Idea #1: Transparently Polyfill New File Formats

Idea #2: Compile Code At Runtime

Idea #3: Process Data Off The Main Thread

Idea #4: Process Data As It Streams

Service Workers, Streams, MessageChannels, Web Workers, Caches, Web Assembly…

Service Workers, Streams, MessageChannels, Web Workers, Caches, Web Assembly…these are all just tools to build things, ***so go build something!***

EXPERIMENT WITH JAVASCRIPT AND **HAVE FUN!**

*ALSO, USE **GLITCH**

glitch.com/@trentmwillis/fun-with-service-workers