Remix

crystallize

Remix

Remix & high performance eCommerce.

With

crystallize

Sébastien Morel

CTO at Crystallize

- 👨🏻‍💻 Backend/DevOps developer since 2003
- 🥖 French living in California
- 🧬 Open Source Knight & Package Maintainer
- 🍼 Dad of a 1-year old
- 💯 Millisecond hunter

Plopix

San Francisco, CA

What's crystallize? A Complete Commerce Layer

An Headless eCommerce Toolbox. We got your back(end).

- PIM
- Order Management (OMS)
- Content Delivery Network
- Reporting & Analytics
- Search
- CMS
- Subscriptions
- DAM (Digital Assets)
- eCommerce
- Video Transcoding

Performance basics

Performances

Best practices

- Core Web Vitals
  - Largest Contentful Paint, **LCP**: should be less than 2.5s
  - First Input Delay, **FID**: should be under 100ms
  - Cumulative Layout Shift, **CLS**: visually stable and less than 0.1

- Frontend metrics
  - First Contentful Paint, **FCP**: under 1.8sec
  - Time to Interactive, **TTI**: within 50ms
  - Total Blocking Time, **TBT**: FCP-TTI, under 300ms
  - Speed Index: under 1.3s

Performances

Best practices

- Server Side
    - Time to First Byte, **TTFB**: under 600 ms
    - Minification, packing, compression, etc.
    - Properly sized images in the good format
    - Removed unused “everything”
    - Use HTTP 2
        - Preload, etc.
    - Use the edge
    - Use HTTP Cache
    - Asynchronous

Remix

Got your back!

What about ecommerce projects?

How are they different from other projects?

Performance wise

Everything is dynamic

Key components

When building an ecommerce

- Catalogue
- Product variations
- Stock
- Prices
- Sales
- Discount
- Cart
- Order Management
- Search

Back to the standard of the web

Core concepts

- Rendering
  - SSR?, SSG?, SPA?
- HTTP Cache
  - Browser
  - Reverse Proxies/CDN, ESI?
  - Expiration/Purge strategies
- Architecture
  - Queues and workers
  - Asynchronous

Rendering

Strategy

- SSG is not adapted for millions of pages/products
- SPA means downloading a big shell for the app and it’s bad for SEO and first load
- **SSR** enables HTTP caching and dynamism

- A progressive approach is key

Remix is really good for e-commerce

HTTP Cache

HTTP Cache, using the Edge

Strategy

- Caching is valid if we can use **Long TTL while being dynamic**
  - Expiration strategy is key
- Browser HTTP Cache can be short
- Shared cache HTTP Cache **must** be long
  - We need to protect the backend, and respond from the edge as much as we can

The goal is to attain more that 95% hit ratio.

HTTP Cache

With Remix

3 situations *(besides medias)*

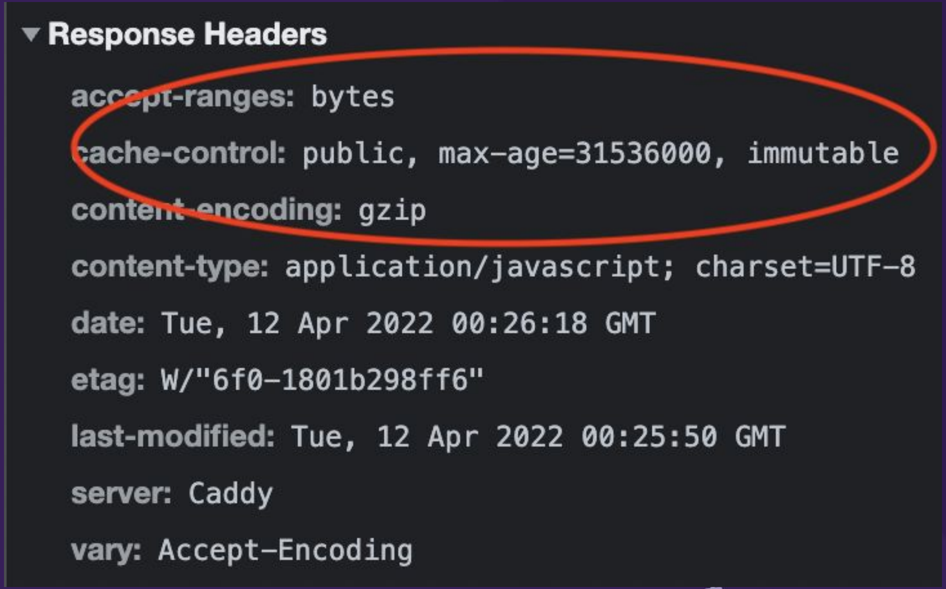
- Data loading *(Fetch)*
- First page rendering *(server side)*
- Page chunks *(scripts)*

Browser HTTP Cache

With Remix

Page chunk *(scripts)*

- Handle by Remix
- Immutable!



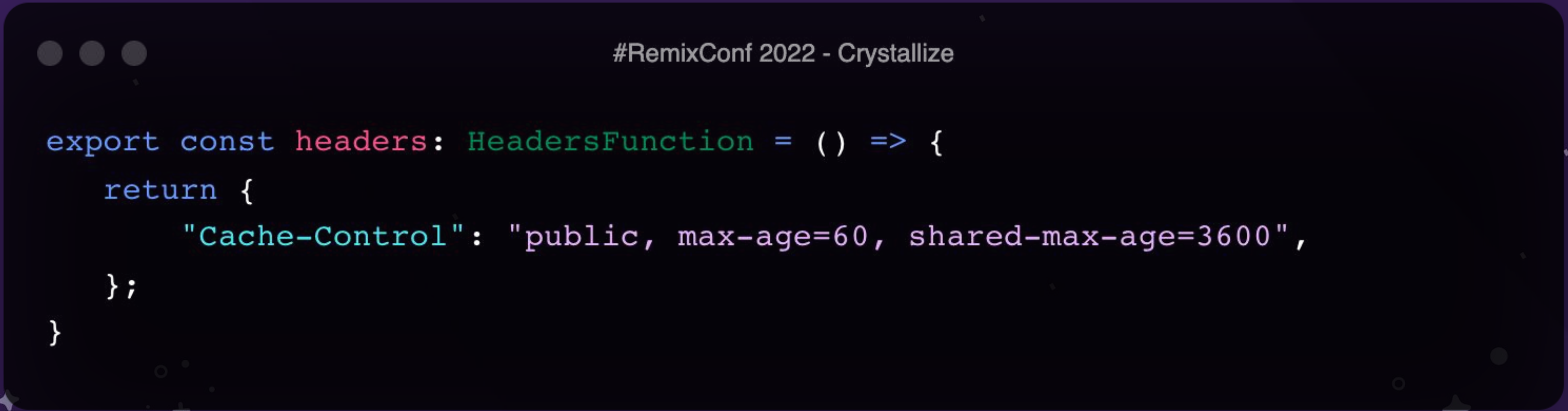
▼ Response Headers

```
accept-ranges: bytes
cache-control: public, max-age=31536000, immutable
content-encoding: gzip
content-type: application/javascript; charset=UTF-8
date: Tue, 12 Apr 2022 00:26:18 GMT
etag: W/"6f0-1801b298ff6"
last-modified: Tue, 12 Apr 2022 00:25:50 GMT
server: Caddy
vary: Accept-Encoding
```

HTTP Cache

With Remix

First page rendering *(server side)*



```
export const headers: HeadersFunction = () => {
    return {
        "Cache-Control": "public, max-age=60, shared-max-age=3600",
    };
}
```

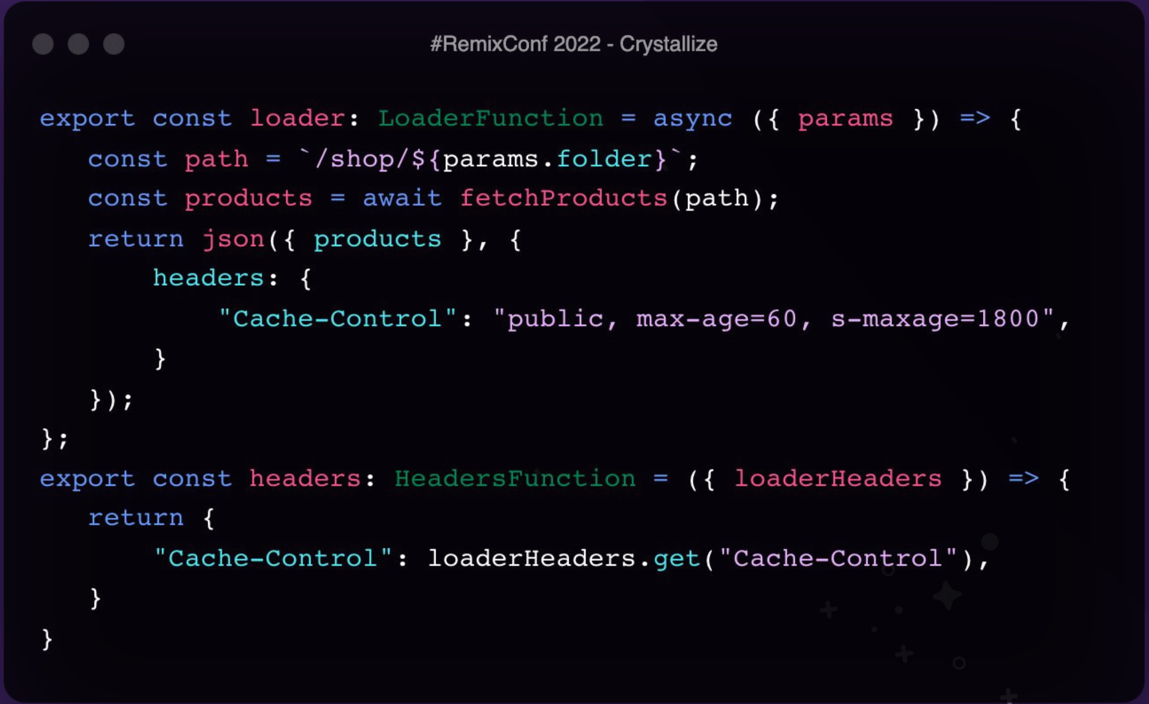
don’t forget **stale-while-revalidate=XXX**

HTTP Cache

With Remix

Data loading

(Fetch)


#RemixConf 2022 - Crystallize

```
export const loader: LoaderFunction = async ({ params }) => {
   const path = `/shop/${params.folder}`;
   const products = await fetchProducts(path);
   return json({ products }, {
       headers: {
           "Cache-Control": "public, max-age=60, s-maxage=1800",
       }
   });
};
export const headers: HeadersFunction = ({ loaderHeaders }) => {
   return {
       "Cache-Control": loaderHeaders.get("Cache-Control"),
   }
}
```

How do you make it dynamic?

Event driven HTTP Cache expiration

Event driven HTTP Cache expiration

Purging

What should be purged?

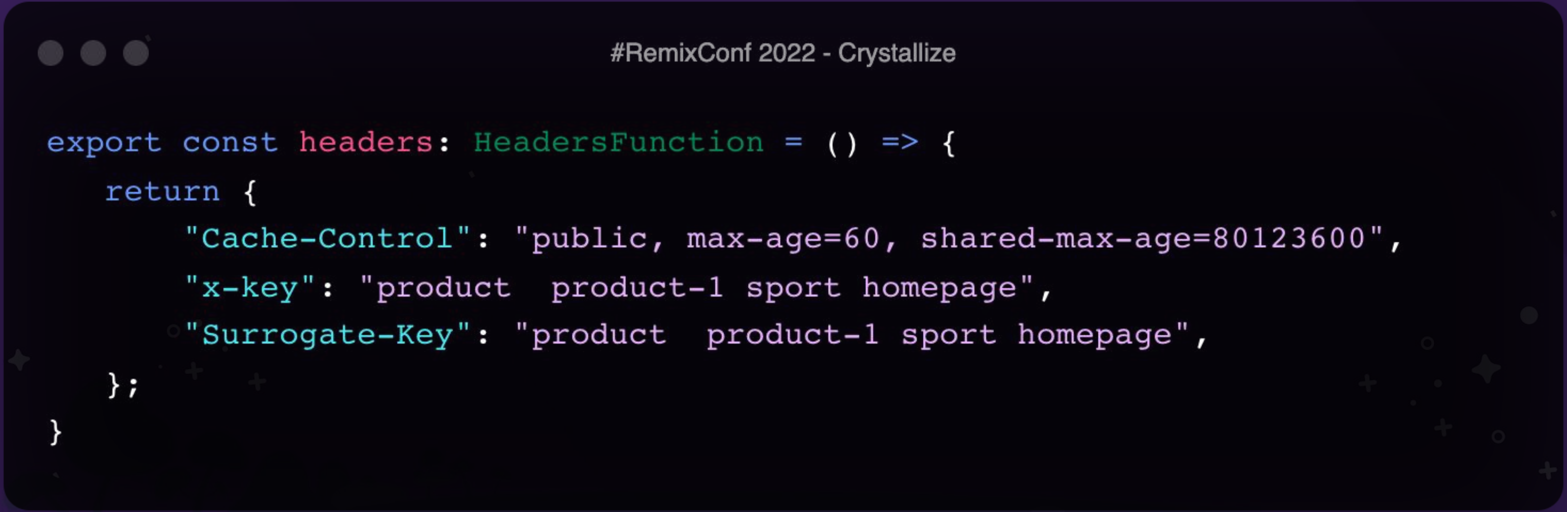
- URL? Which one?
- All on them?
- How do you the URL to purge?

Event driven HTTP Cache expiration

Purging

Purge by Headers!

Tag your responses, so you can purge only what you want!


#RemixConf 2022 - Crystallize

```
export const headers: HeadersFunction = () => {
    return {
        "Cache-Control": "public, max-age=60, shared-max-age=80123600",
        "x-key": "product  product-1 sport homepage",
        "Surrogate-Key": "product  product-1 sport homepage",
    };
}
```

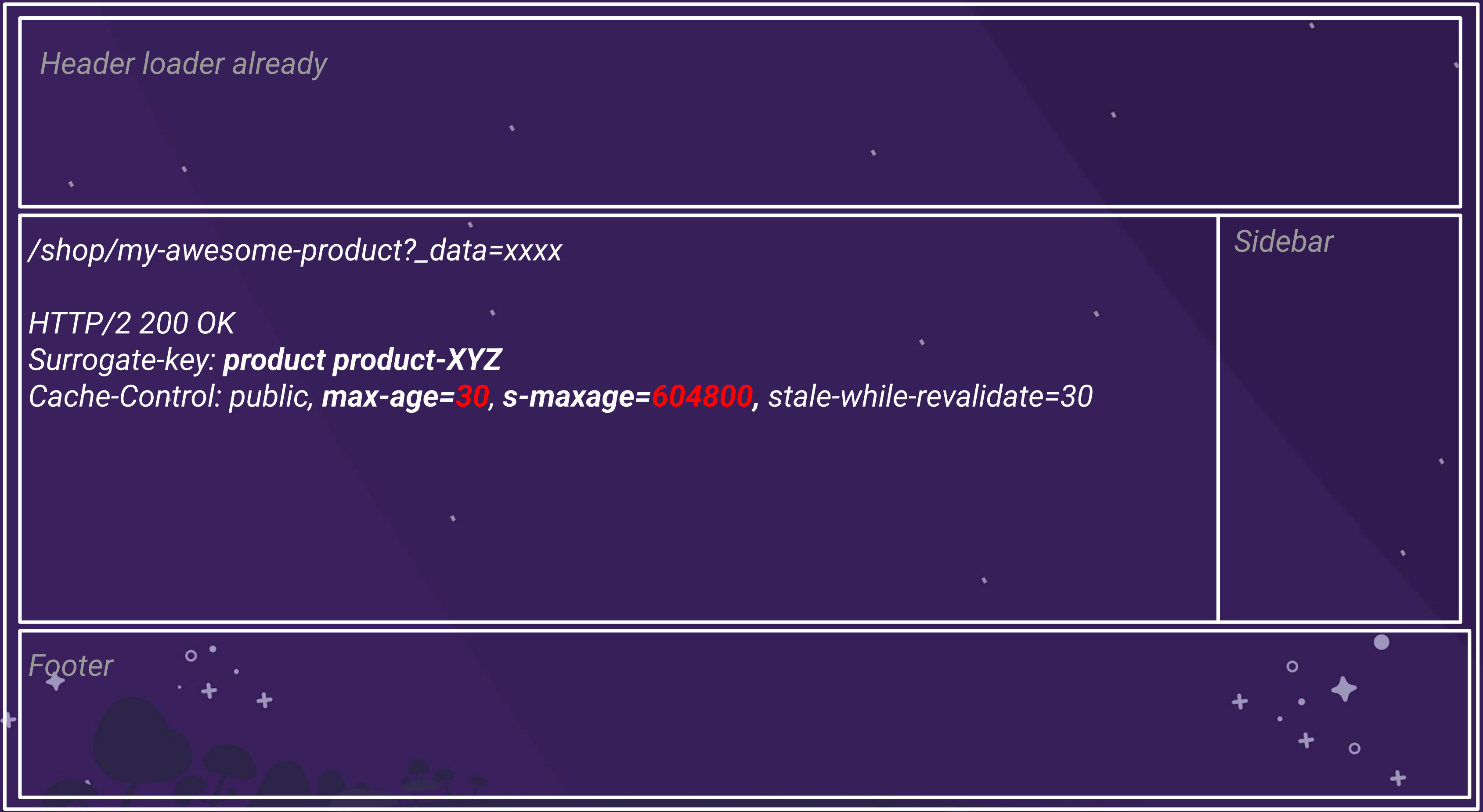
Event driven HTTP Cache expiration

Purging

Context:

- Application (chunks) are immutable (per build)
- SSR Pages and Data are tagged and cached
- You have an event mechanism to trigger purges on updates

/shop/my-awesome-product



```
/shop/my-awesome-product?_data=xxxx

HTTP/2 200 OK
Surrogate-key: product product-XYZ
Cache-Control: public, max-age=30, s-maxage=604800, stale-while-revalidate=30
```

User specific

Data

User specific data

Old school fetch mechanism enriched by Remix fetcher!

- Bypass the cache
- Specific to the user
  - Requires 1 more request
  - Combine everything
-

Use Remix Fetcher: Optimistic UI

HTTP/2

Server Push

First PR in progress: https://github.com/remix-run/remix/pull/3200

HTTP2

Server Push

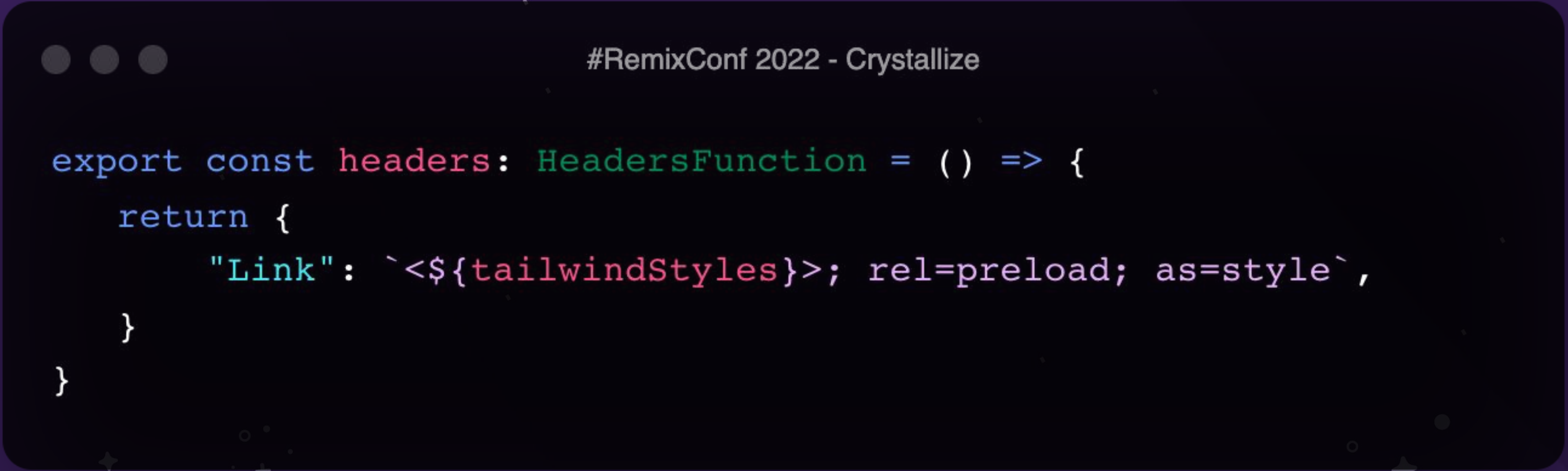
No more “Inline Resources” !

- Link: </css/styles.css>; rel=preload; as=style

- Link: </css/styles.css>; rel=preload; as=style, </js/scripts.js>; rel=preload; as=script, </img/logo.png>; rel=preload; as=image

HTTP2

Server Push

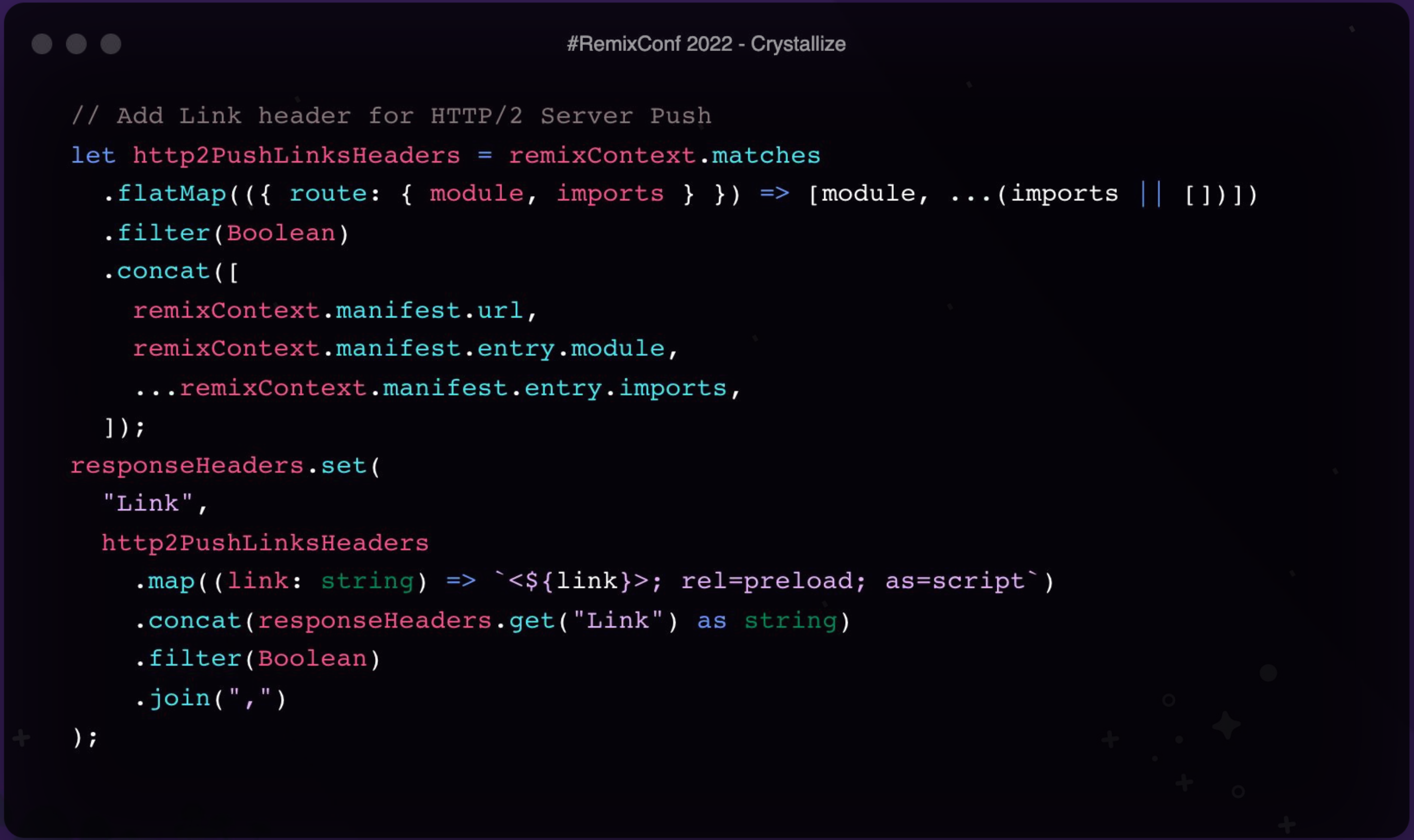


#RemixConf 2022 - Crystallize

```
export const headers: HeadersFunction = () => {
    return {
        "Link": `<${tailwindStyles}>; rel=preload; as=style`,
    }
}
```

HTTP2

Server Push



#RemixConf 2022 - Crystallize

```
// Add Link header for HTTP/2 Server Push
let http2PushLinksHeaders = remixContext.matches
  .flatMap(({ route: { module, imports } }) => [module, ...(imports || [])])
  .filter(Boolean)
  .concat([
    remixContext.manifest.url,
    remixContext.manifest.entry.module,
    ...remixContext.manifest.entry.imports,
  ]);
responseHeaders.set(
  "Link",
  http2PushLinksHeaders
    .map((link: string) => `<${link}>; rel=preload; as=script`)
    .concat(responseHeaders.get("Link") as string)
    .filter(Boolean)
    .join(",")
);
```

Application Cache

Stock/Inventory Management

Application cache

Stock/Inventory Management

Assuming **1 warehouse**, **3** different types of **Stock**

- What you have in the warehouse: ***onHand***
- What has been ordered (reserved): ***onHold***
- What is therefore available on the website (onHand-onHold)

***onHold* is application cache**

Architecture

Don’t do anything on request

Synchronous way

Queues and Workers

1/ Buyer is on the Checkout Page

2/ Buyer places the Order

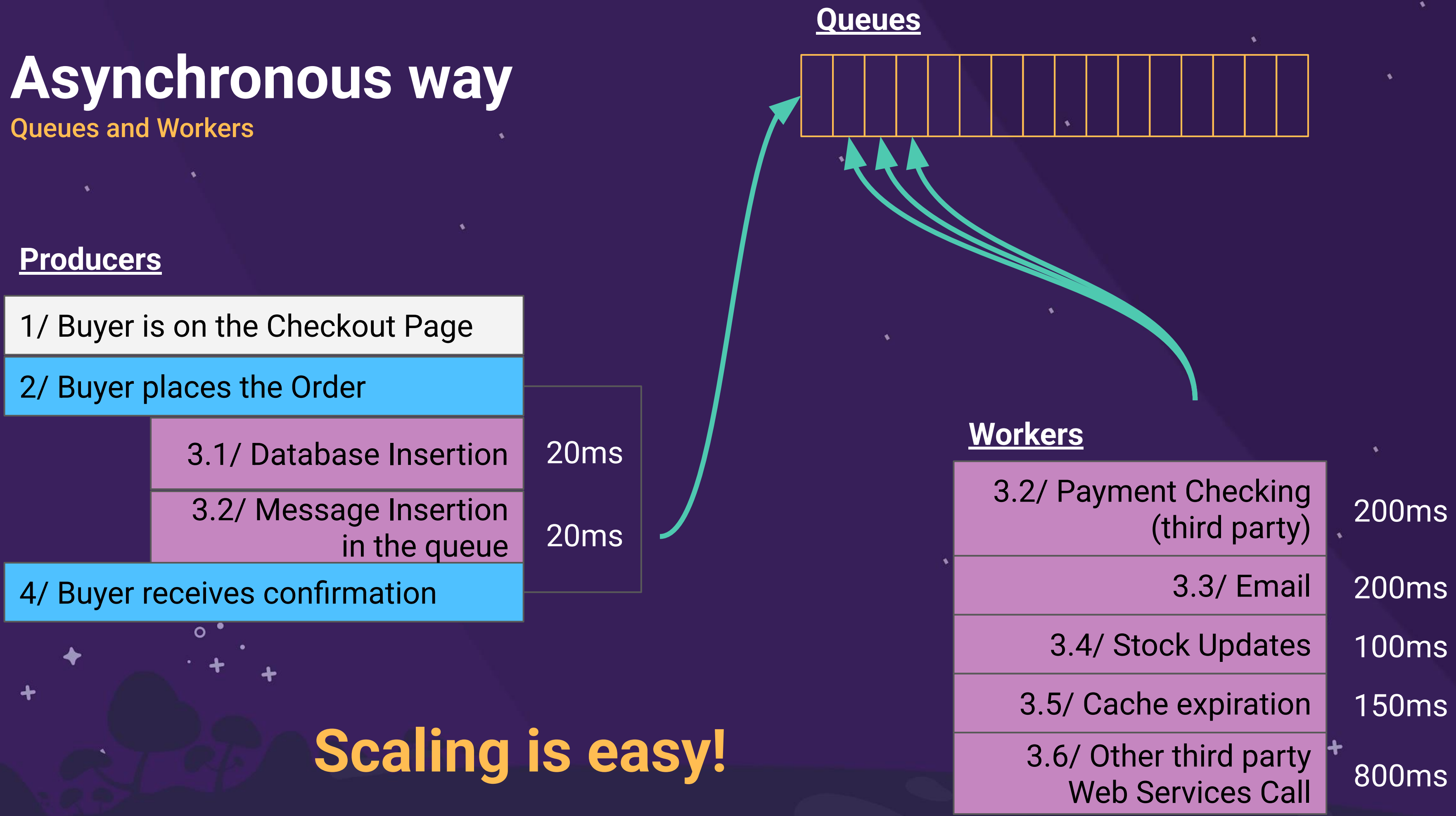
3.1/ Database Insertion	20ms
3.2/ Payment Checking (third party)	200ms
3.3/ Email	200ms
3.4/ Stock Updates	100ms
3.5/ Cache expiration	150ms
3.6/ Other third party Web Services Call	800ms

4/ Buyer receives confirmation

Easily > **1 sec**

Asynchronous way

Queues and Workers



Producers

1/ Buyer is on the Checkout Page		
2/ Buyer places the Order		
	3.1/ Database Insertion	20ms
	3.2/ Message Insertion in the queue	20ms
4/ Buyer receives confirmation		

Workers

3.2/ Payment Checking (third party)	200ms
3.3/ Email	200ms
3.4/ Stock Updates	100ms
3.5/ Cache expiration	150ms
3.6/ Other third party Web Services Call	800ms

Scaling is easy!

Conclusion

Wrapping up

- Use **Web Standards**
- Use **HTTP2**, **Cache** and **CDN**s
  - with **expiration** method
- **Cache** is part of your application, test it
  - Include everything in your local & in your CI/CD
- **User specific data** can be **fetched afterwards**
- **Asynchronous** is key
  - For scalability and thus performances

CRYSTALLIZE

Quick note about ESI

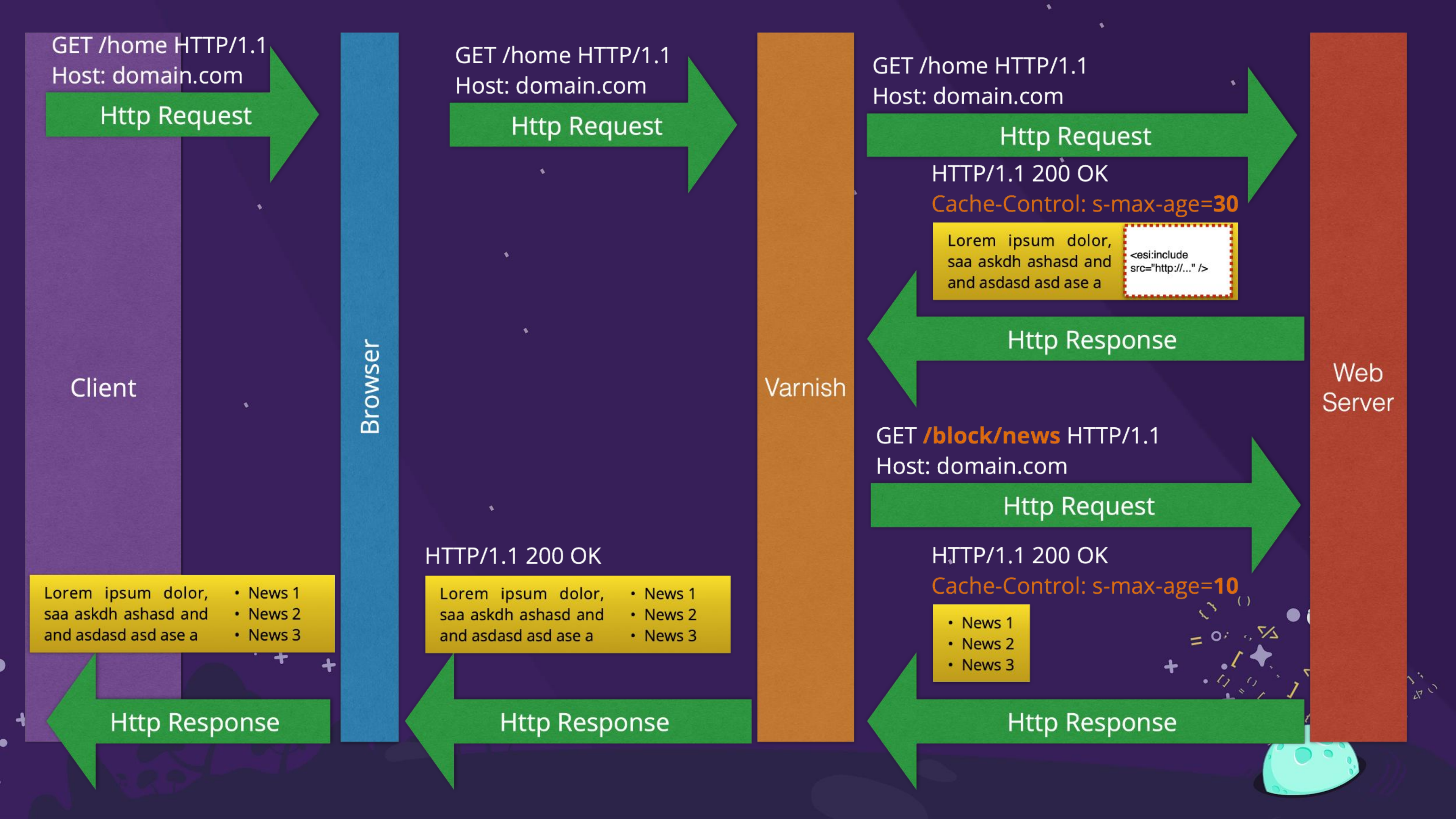
Edge Side Include

Edge Side Include

With Remix

Reverse Proxy / CDN are able to fetch data (blocks that are sub http request) and cache the result with another TTL before to return the fully built page.

Some kind of diagram here



```
GET /home HTTP/1.1
Host: domain.com
```

```
GET /home HTTP/1.1
Host: domain.com
```

```
GET /home HTTP/1.1
Host: domain.com
```

```
HTTP/1.1 200 OK
Cache-Control: s-max-age=30
```

Lorem ipsum dolor, saa askdh ashasd and and asdasd asd ase a

```
<esi:include src="http://..." />
```

```
GET /block/news HTTP/1.1
Host: domain.com
```

```
HTTP/1.1 200 OK
Cache-Control: s-max-age=10
```

- News 1
- News 2
- News 3

```
HTTP/1.1 200 OK
```

Lorem ipsum dolor, saa askdh ashasd and and asdasd asd ase a

- News 1
- News 2
- News 3

Lorem ipsum dolor, saa askdh ashasd and and asdasd asd ase a

- News 1
- News 2
- News 3

Edge Side Include

With Remix

Not really useful with Remix, as you can customize TTL for each data loader and each page

It could be used outside of the Remix App layout
For a banner, the footer etc.

Mention the hydration problem

crystallize