

# ServiceWorker for Performance

[bit.ly/sw-breakout-tpac2024](https://bit.ly/sw-breakout-tpac2024)

IRC: #serviceworkers

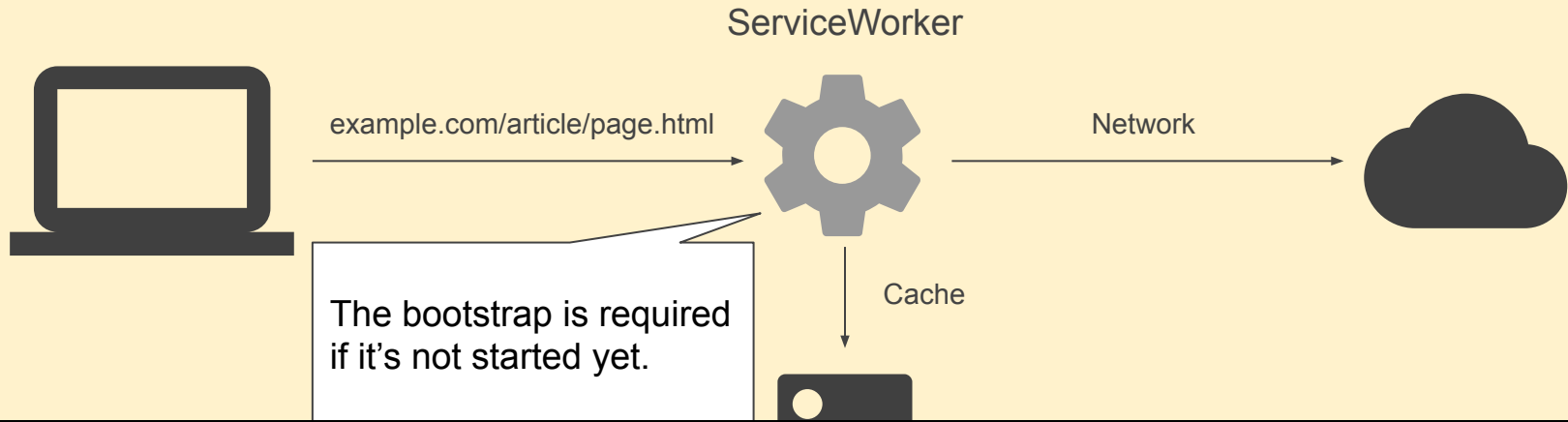
Shunya Shishido [sisidovski@chromium.org](mailto:sisidovski@chromium.org)  
Keita Suzuki [suzukikeita@chromium.org](mailto:suzukikeita@chromium.org)

Breakout in TPAC 2024 Anaheim, CA

# Agenda

1. Recap: Why ServiceWorker?
2. Updates from last year, Static Routing API
3. Newly incubated ideas
  - SWAutoPreload
  - SW Synthetic Response
4. Q&A, discussion

# Recap: How ServiceWorker works



**ServiceWorker is on the critical path of the navigation!**

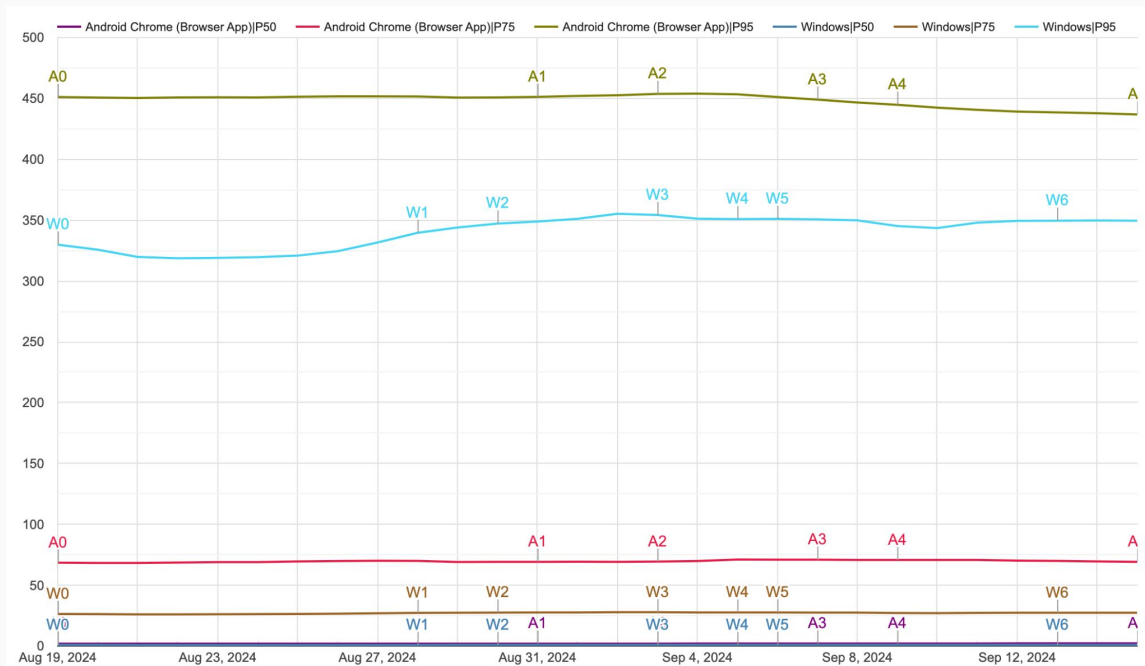
# Cost of ServiceWorker Bootstrap

## Android

- 3ms in p50
- 70ms in p75
- 436ms in p95

## Windows

- 1ms p50
- 27ms in p75
- 350ms in p95



# Cost of ServiceWorker

- What percentage of navigation needs to bootstrap service worker?
  - About 30% of ServiceWorker are not running
  - For cross origin navigation, it's about 50%
- What percentage of fetch handlers result in fallback\*?
  - On Windows, the fallback rate is 13%.
  - On Android, the fallback rate is 46%.

\*fallback: fetch handler is executed, but the fetch handler never respond the content. The browser ends up sending the network request with the regular network stack.

# Recent Updates

# Static Routing API

- Allows developers to register routing rules.
- Offload ServiceWorker tasks from the loading critical path.

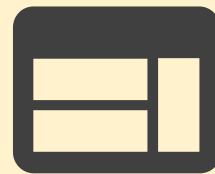
```
// Go straight to the network and bypass invoking fetch handlers for all URLs that start with '/images/'.  
addEventListener('install', (event) => {  
  event.addRoutes({  
    condition: {  
      urlPattern: {pathname: "/images/*"}  
    },  
    source: "network"  
  });  
});
```

# Static Routing API



In the install event handler, register routing info to the data structure associated with service worker.

```
addEventListener('install', (event) => {  
  event.addRoutes({  
    condition: {  
      urlPattern: {pathname: "/images/*"}  
    },  
    source: "network"  
  });  
});
```



example.com



# Static Routing API

Check if the request is matched with the registered routing rules.



GET example.com/images/test.png

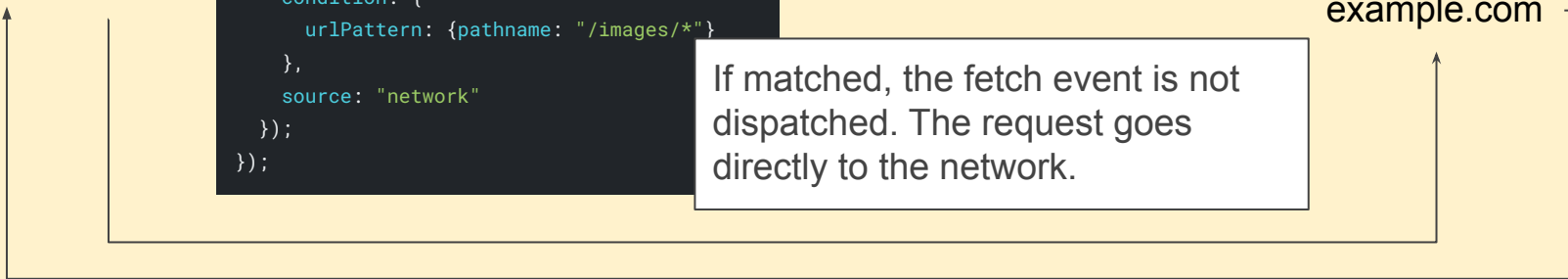


```
addEventListener('install', (event) => {  
  event.addRoutes({  
    condition: {  
      urlPattern: {pathname: "/images/*"}  
    },  
    source: "network"  
  });  
});
```

If matched, the fetch event is not dispatched. The request goes directly to the network.



example.com



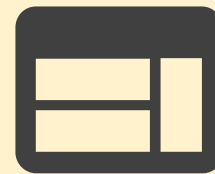
# Static Routing API



GET example.com/article/page.html



Network



example.com

```
addEventListener('install', (event) => {  
  event.addRoutes({  
    condition: {  
      urlPattern: {pathname: "/images/*"}  
    },  
    source: "network"  
  });  
});
```

Cache



If not matched, just go the through regular ServiceWorker fetch handler path.

# Conditions and Sources

## RouterCondition

- urlPattern
- requestMethod
- requestMode
- requestDestination
- runningStatus

## RouterSource

- fetch-event
- network
- cache
- race-network-and-fetch-handler

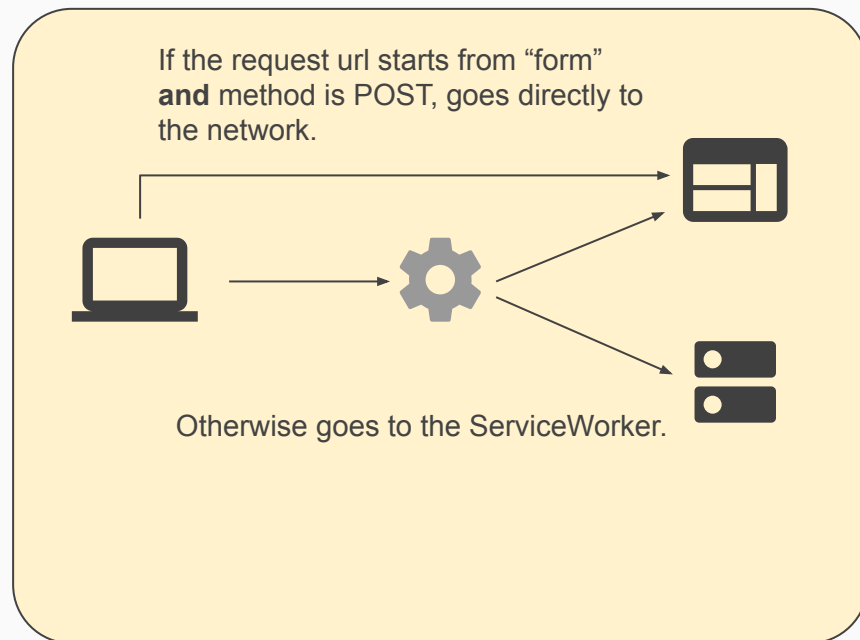
## Syntax Sugar

- “or” condition
- “not” condition

# Static Routing API common use case

Skip POST requests in matched URLs

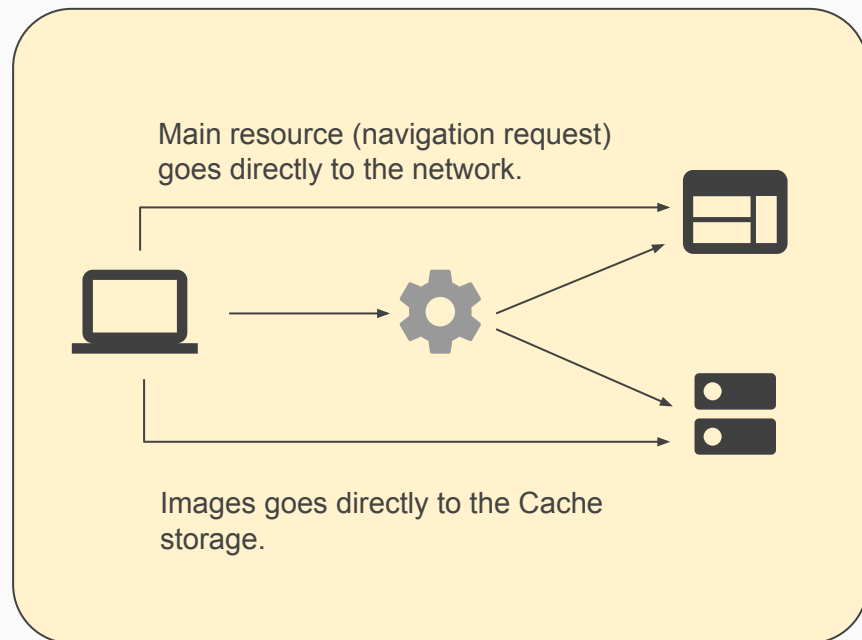
```
addEventListener('install', (event) => {  
  event.addRoutes({  
    condition: {  
      urlPattern: {pathname: "/form/*"},  
      requestMethod: "post"  
    },  
    source: "network"  
  });  
});
```



# Static Routing API common use case

Skip SW for navigation requests.  
Directly to the cache for subresources.

```
addEventListener('install', (event) => {
  event.addRoutes([
    {
      condition: {requestMode: "navigate"},
      source: "network"
    },
    {
      condition: {
        urlPattern: {pathname: "/images/*"}
      },
      source: "cache"
    }
  ]);
});
```



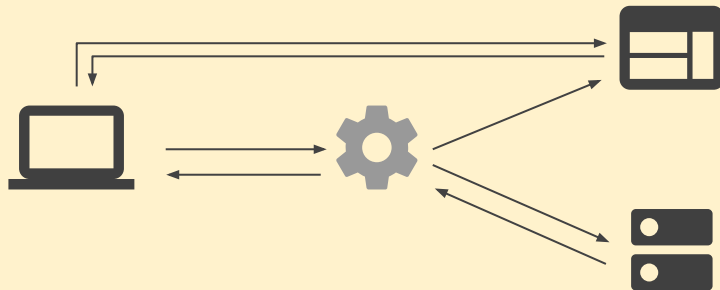
# Static Routing API common use case

If the request is a navigation request and the SW is not running, starts the race between the network and fetch handlers.

```
addEventListener('install', (event) => {  
  event.addRoutes({  
    condition: {  
      requestMode: "navigate",  
      runningStatus: "not-running",  
    },  
    source: "race-network-and-fetch-handler"  
  });  
});
```

“race-network-and-fetch-handler” is the router source starting the race between the network request and the ServiceWorker fetch handler.

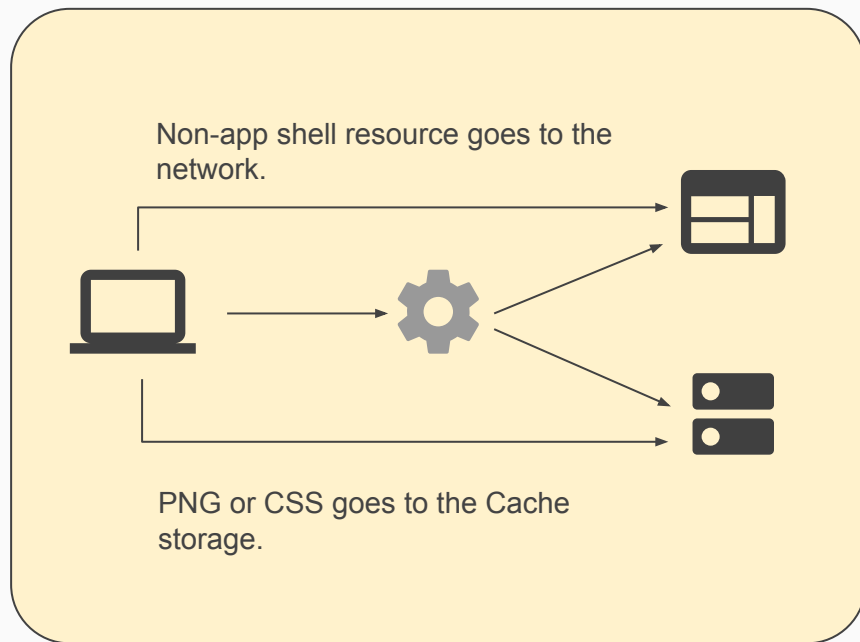
The browser uses the result of whichever was faster.



# Static Routing API common use case

Cache for png or css resources.  
Network for the URL is not /app-shell/\*

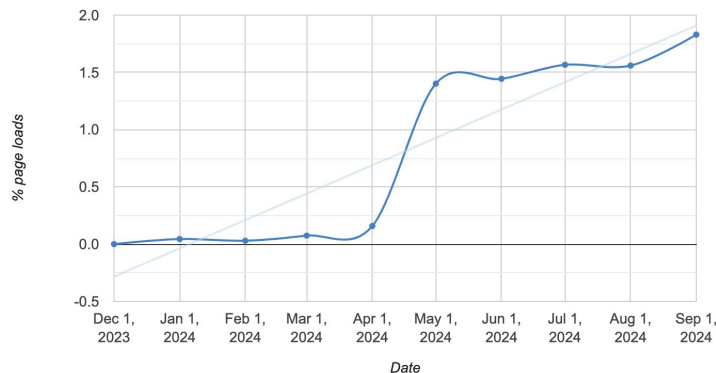
```
addEventListener('install', (event) => {
  event.addRoutes([
    {
      condition: {
        or: [{urlPattern: "**/*.png"}, {urlPattern: "**/*.css"}]
      },
      source: 'cache'
    },
    {
      condition: {
        not: {urlPattern: "/app-shell/*"}
      },
      source: 'network'
    }
  ])
});
```



# How Static Routing API is used today

As of Sep 2024, more than 1.5% of page loads use Static Routing API in Chrome.

One of the partner websites improved LCP by 80ms by enabling the API with 'race-network-and-fetch-handler' for navigation requests.



<https://chromestatus.com/metrics/feature/timeline/popularity/4711>



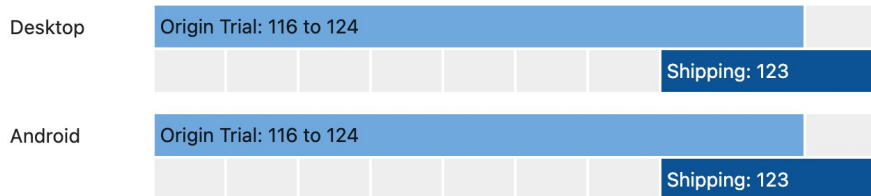
# Current Status

## Status in Chromium

Blink components: [Blink>ServiceWorker](#)

Implementation status: **Enabled by default** tracking bug

Estimated milestones:



<https://chromestatus.com/feature/5185352976826368>

The screenshot shows a GitHub pull request for the 'ServiceWorker static routing API #1701'. The pull request is merged and includes 59 commits. The author is yoshisatoyanagisawa. The pull request description states: 'This API allows developers to configure the routing, and allows them to offload simple things ServiceWorker'.

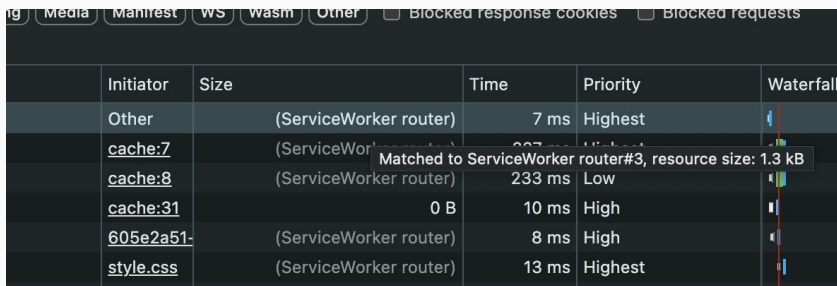
<https://github.com/w3c/ServiceWorker/pull/1701>

# Our Recent Focus

## Improving developer experience

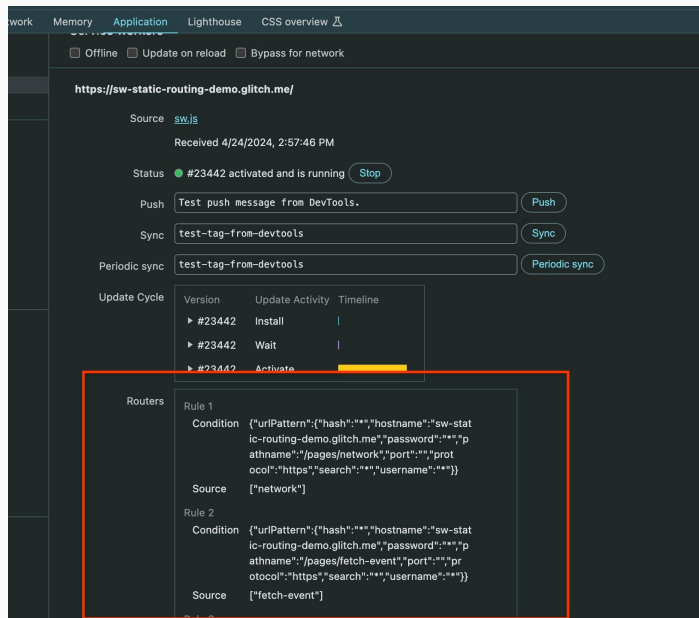
- More visibility in DevTools
- Resource Timing API

## More features are explored on the Static Routing API infra



	Initiator	Size	Time	Priority	Waterfall
	Other	(ServiceWorker router)	7 ms	Highest	
	cache:7	(ServiceWorker router)	233 ms	Low	
	cache:8	(ServiceWorker router)	233 ms	Low	
	cache:31	0 B	10 ms	High	
	605e2a51-	(ServiceWorker router)	8 ms	High	
	style.css	(ServiceWorker router)	13 ms	Highest	

Matched to ServiceWorker router#3, resource size: 1.3 kB



work Memory Application Lighthouse CSS overview

Offline Update on reload Bypass for network

https://sw-static-routing-demo.glitch.me/

Source sw.js

Received 4/24/2024, 2:57:46 PM

Status #23442 activated and is running Stop

Push Test push message from DevTools. Push

Sync test-tag-from-devtools Sync

Periodic sync test-tag-from-devtools Periodic sync

Update Cycle

Version	Update Activity	Timeline
#23442	Install	
#23442	Wait	
#23442	Activate	

Routers

```
Rule 1
Condition {"urlPattern":{"hash":"","hostname":"sw-static-routing-demo.glitch.me","password":"","pathName":"/pages/network","port":"","protocol":"https","search":"","username":""}}
Source ["network"]

Rule 2
Condition {"urlPattern":{"hash":"","hostname":"sw-static-routing-demo.glitch.me","password":"","pathName":"/pages/fetch-event","port":"","protocol":"https","search":"","username":""}}
Source ["fetch-event"]
```

# Many components exist in static routing API



**Hard to understand how the components are actually behaving**

Static Router

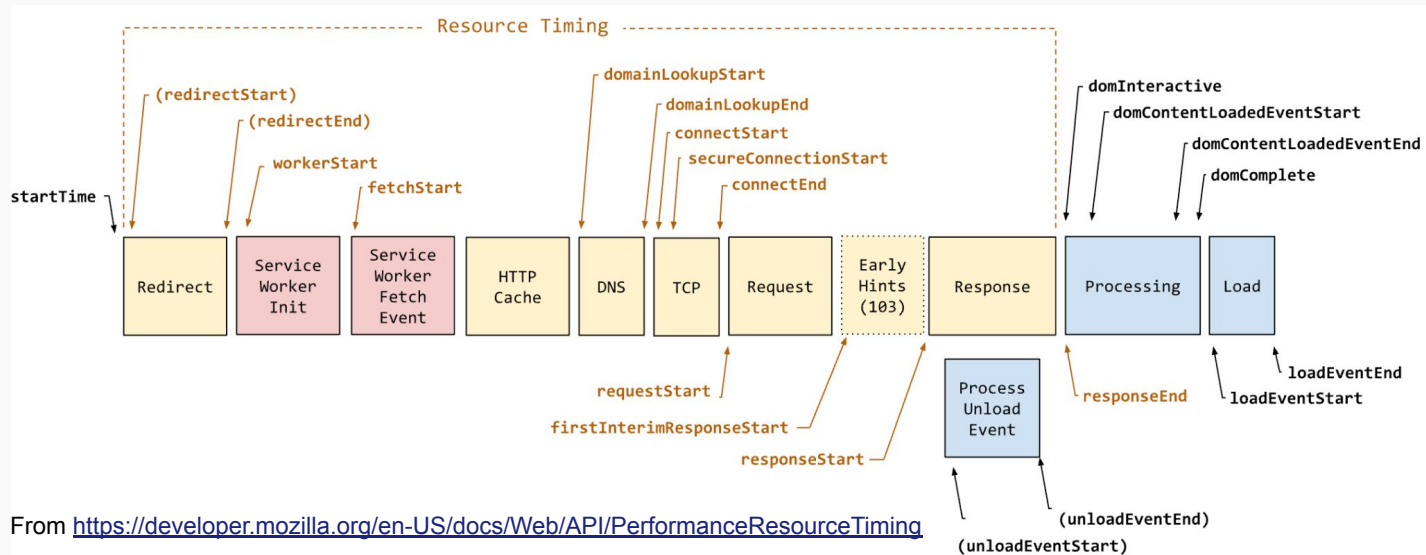
How long router eval took?

How long did cache lookup take?

Cache

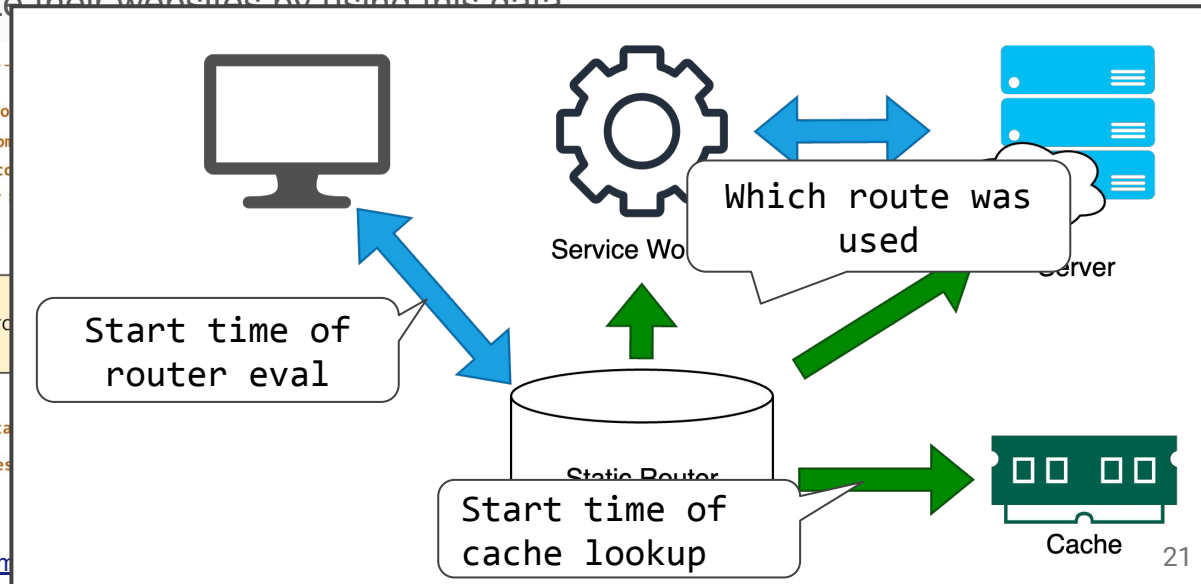
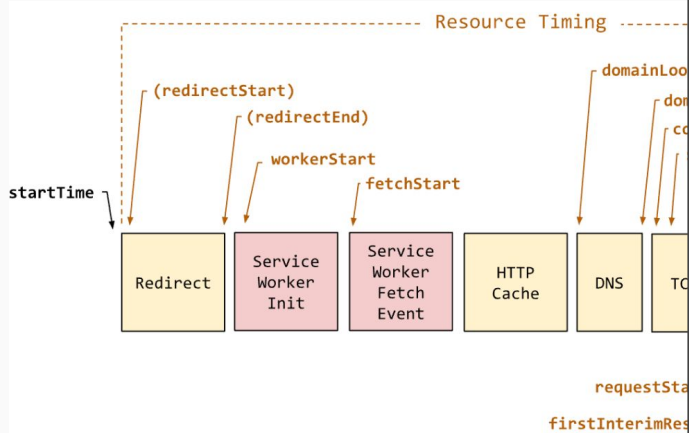
# Extending Resource Timing API

- **Provide loading-related behaviors to developers** via resource timing API
  - Developers can optimize their websites by using this data



# Extending Resource Timing API

- **Provide loading-related behaviors to developers** via resource timing API
  - Developers can optimize their websites by using this data

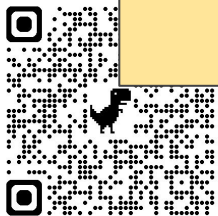
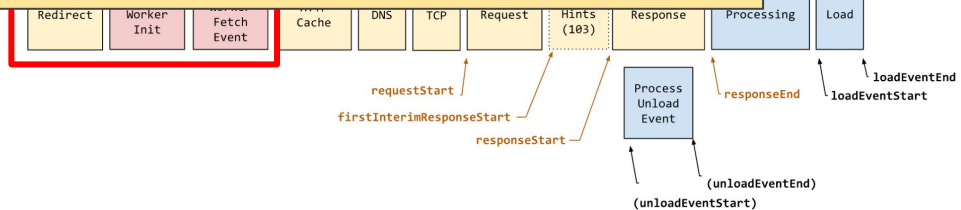
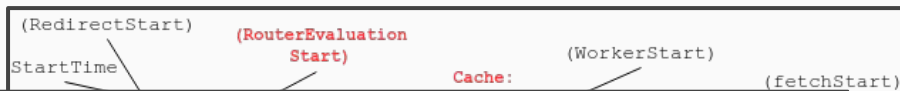


# Proposal in Resource Timing API (In discussion)

✓ Two new timings

✓ T

Currently in discussion at the Web Performance WG  
Discussion session 9/26 9:00AM ~



Explainer

# Feedback is welcomed

We need more feedback in general, but specifically...

- How to handle unsupported router features [issue28](#)
- The depth limit for the router registration [issue6](#)
- Visibility of registered router rules
- Interoperability

# New Ideas



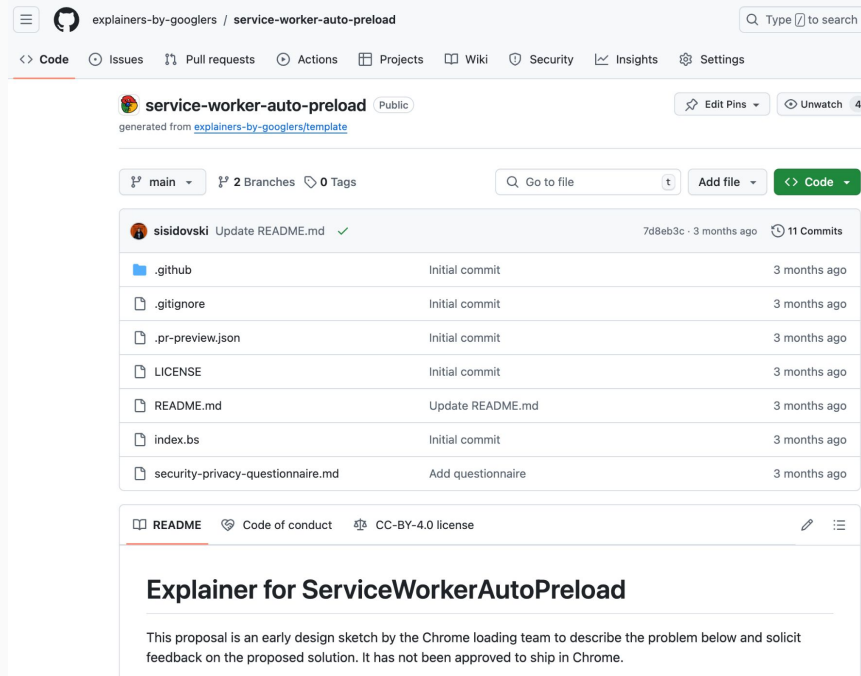
# ServiceWorkerAutoPreload

1. About half of SW fetch result is fallback.
2. Many websites just path-through responses from the network.

## What if...

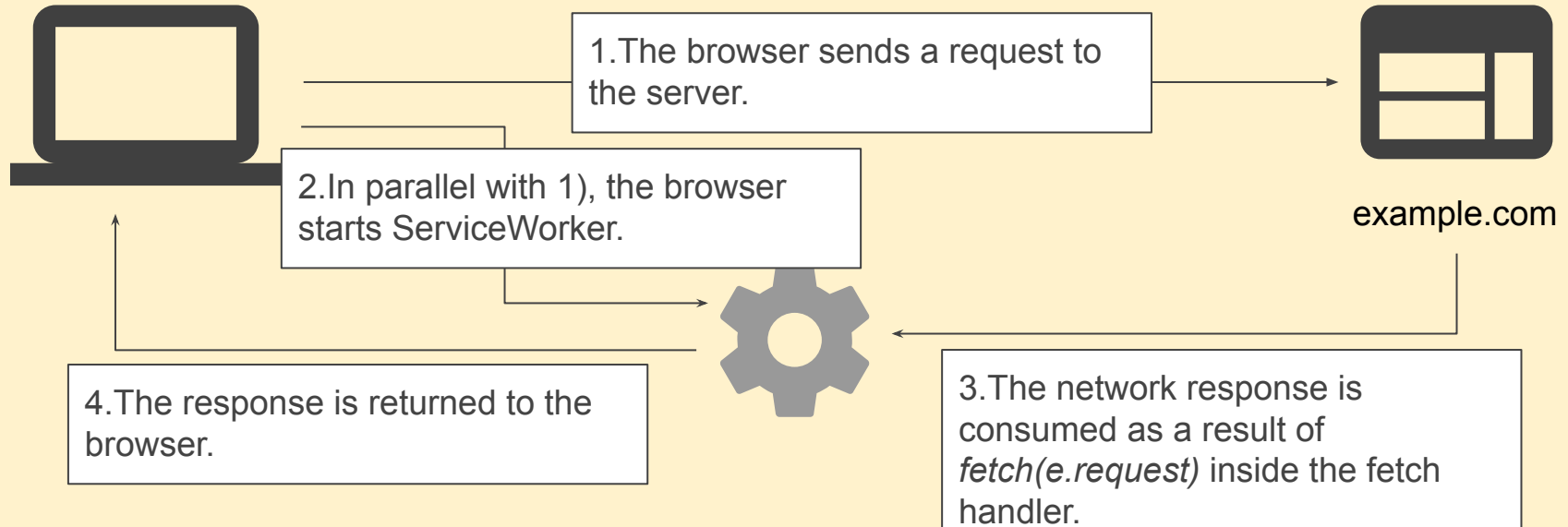
1. Automatically dispatches a network request before starting the SW?
2. Consumes the response inside the fetch handler, or as a fallback response?

<https://github.com/explainers-by-googlers/service-worker-auto-preload>

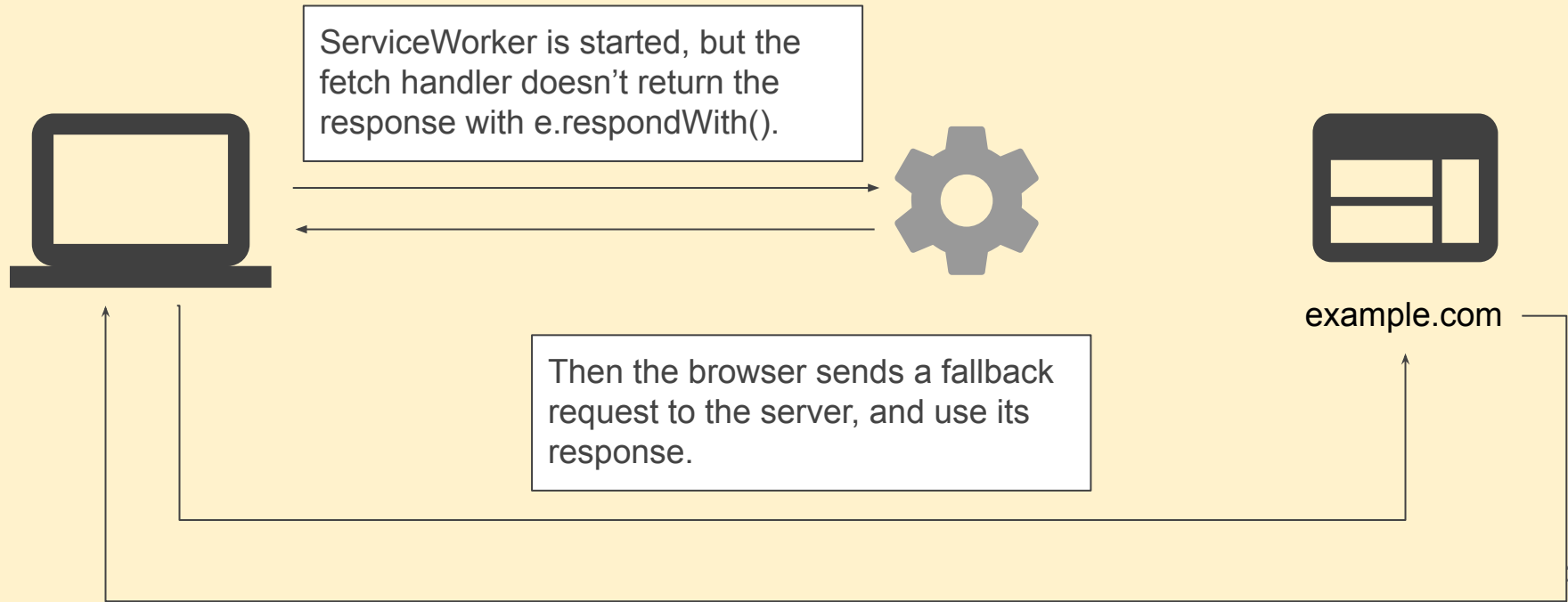


The screenshot shows the GitHub repository page for 'explainers-by-googlers / service-worker-auto-preload'. The repository is public and was generated from the 'explainers-by-googlers/template'. It has 2 branches and 0 tags. The commit history shows a recent update to the README.md by user 'sisidovski' 3 months ago. The file list includes .github, .gitignore, .pr-preview.json, LICENSE, README.md, index.bs, and security-privacy-questionnaire.md. The README section is visible, titled 'Explainer for ServiceWorkerAutoPreload', and contains the text: 'This proposal is an early design sketch by the Chrome loading team to describe the problem below and solicit feedback on the proposed solution. It has not been approved to ship in Chrome.'

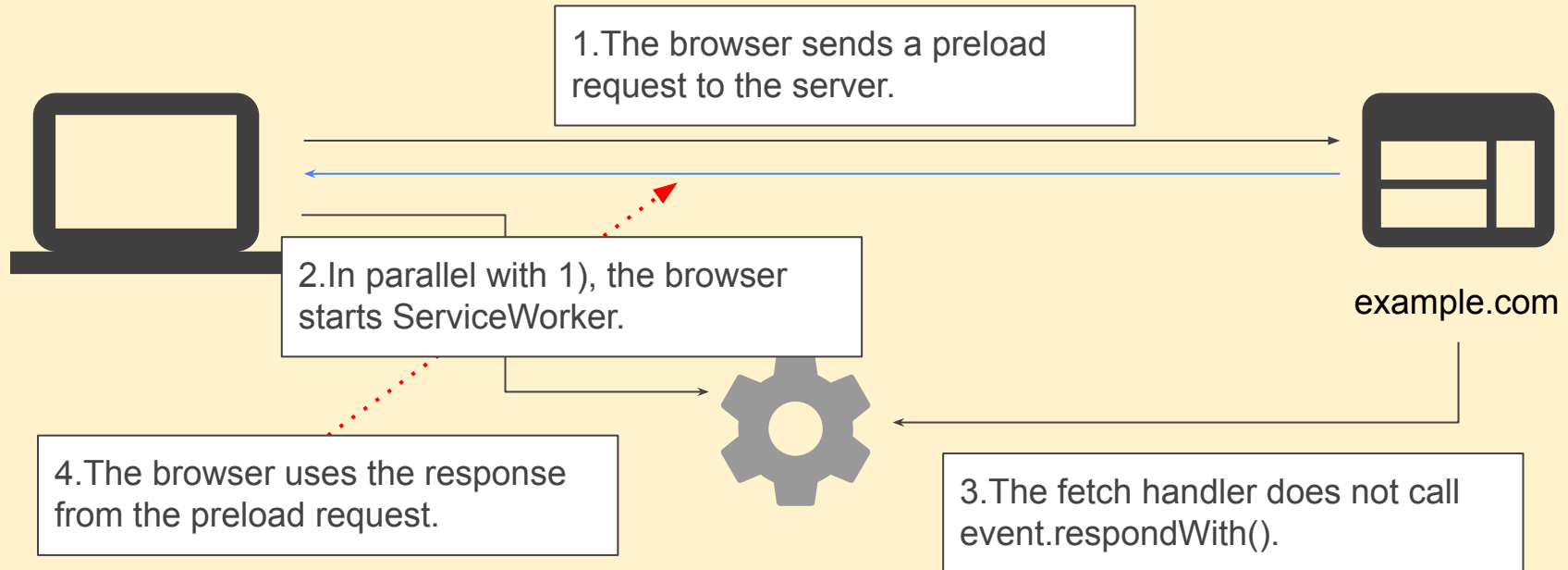
# How it works



# Fallback



# Fallback w/ SWAutoPreload



# Requests can be duplicated?

- The auto preload request is resolved with `fetch(e.request)` in the fetch handler.
- `fetch(e.request)` doesn't send a network request. Instead, it returns a promise that is resolved with the response from the auto preload request.
- Technically it's possible that requests are duplicated e.g. `request.clone()`
- It can be mitigated by applying `ServiceWorkerAutoPreload` only for websites that meet an eligibility criteria.

```
self.addEventListener('fetch', (event) => {  
  // This fetch() doesn't create a new network  
  // request. Instead, resolved with the response from  
  // the auto preload network request.  
  event.respondWith(fetch(event.request));  
  
  // This fetch() creates a network request.  
  event.respondWith(fetch(event.request.clone()));  
});
```

# Eligibility Criteria

- Planned criteria: higher rates of fetch handler results are fallback.
  - e.g. 98+% fallback
- We (Google Chrome team) plan to enable SWAutoPreload automatically for the sites met the criteria.
- The criteria may be revised in the future to behave more smartly.

# Opt-out

- Opting out can be done via the Static Routing API.
- By registering the router rule that matches all requests, and asking them to go to the fetch handler.

```
self.addEventListener('install', e => {  
  e.addRoutes({  
    condition: {  
      urlPattern: new URLPattern({})  
    },  
    source: "fetch-event"  
  });  
});
```

# Difference from NavigationPreload

Trying to solve the same problem, which is minimizing the cost of ServiceWorker bootstrap.

## NavigationPreload

- The response is resolved with `event.preloadResponse`.
- Explicitly enabled via `PreloadManager.enable()`.
- Prioritized when both features are enabled.

## ServiceWorkerAutoPreload

- The response is resolved with the regular response of `fetch(event.request)`.
- Enabled automatically by the browser criteria.

For more details, please see

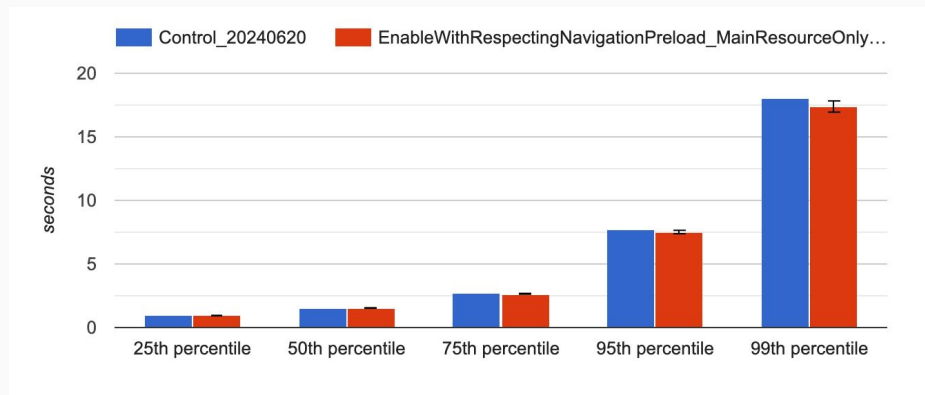
<https://github.com/explainers-by-googlers/service-worker-auto-preload#how-is-it-different-from-the-navigation-preload-api>



# Current Status

Prototyping, under the experiment.

From the Chrome Beta channel experiment, multiple loading metrics improvements were observed e.g. LCP on the ServiceWorker controlled page.



LCP on ServiceWorker controlled page

# ServiceWorkerAutoPreload

- Specified as an optional optimization that the browser can apply at its choosing.
- While it can provide performance improvements, it's observable via the server as additional requests.
- However, it can be mostly not observable as far as the browser limits this optimization only to ServiceWorkers in which the fetch handler returns the response always consistent with the network request.

# We need more feedback

General feedback, concerns are very welcomed.

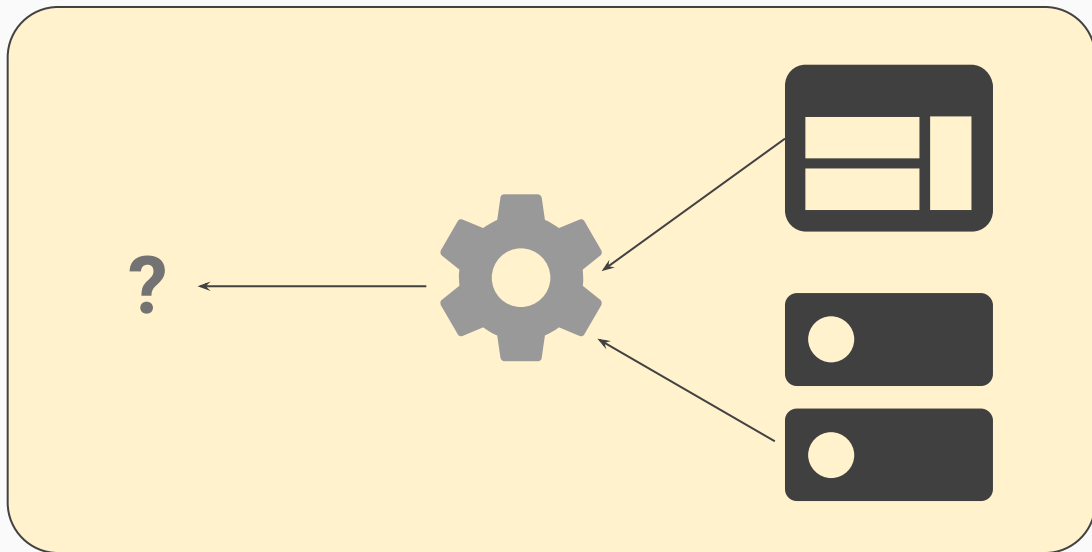
Is the current criteria reasonable?

# ServiceWorker Synthetic Response

# ServiceWorker Synthetic Response

A new idea as the part of the Static Routing API.

Synthetic Response provides faster navigation by starting the page load earlier in the render process.



Currently, the renderer process starts the commit after receiving the response from the server.

- (Android) The commit navigation task takes median 40ms, 78ms in p75, 194ms in p95.
- (Android) From the navigation start to receive the response, it takes median 360ms, p75 759ms, p95 2350ms.

Can we parallelize sending a network request and the commit navigation?

Brow

Load  
Stop

Browser's Network stack

Start url  
request

Read response body

Renderer Process

Commit

Load

# ServiceWorker Synthetic Response

What if the browser could know the set of HTTP response headers in advance for the upcoming navigation request? e.g.

- content-type is text/html
- Non-204 response
- No content disposition
- COOP/COEP headers
- etc

**=> The renderer process can speculatively start the navigation commit without waiting for the response from the network.**

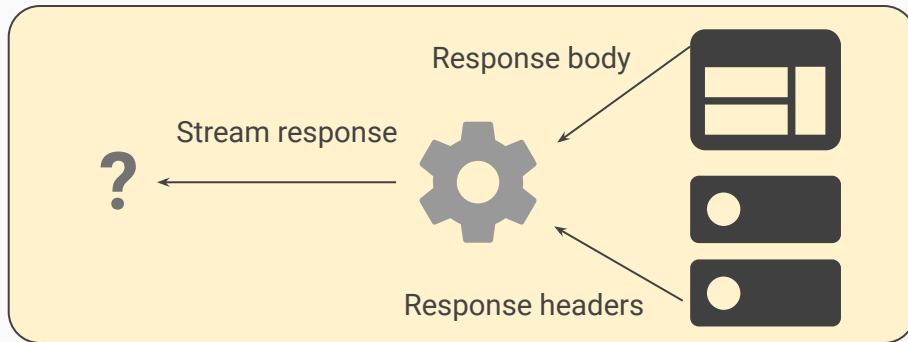
# ServiceWorker Synthetic Response

We can extend the Static Routing API so that it can store the set of response headers in the install phase.

```
self.addEventListener("install", e => {
  e.addRoutes({
    condition: {requestMode: "navigate"},
    source: {
      predefinedHeaders: new Headers({
        "Content-Type": "text/html",
        "Cache-Control": "no-cache"
      })
    }
  });
});
```

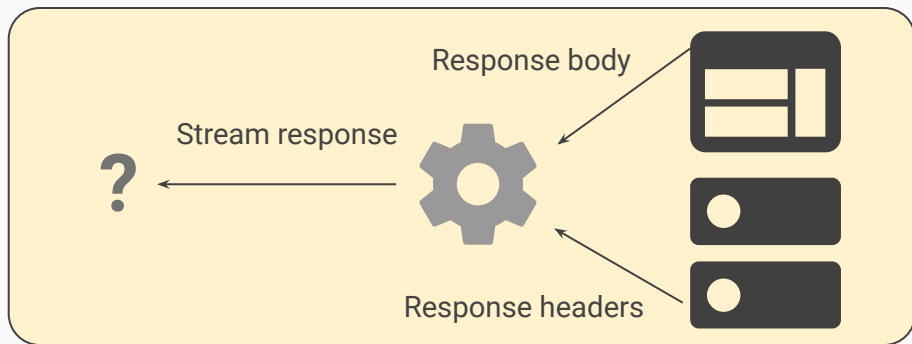


# ServiceWorker Synthetic Response



```
self.addEventListener("install", e => {
  e.addRoutes({
    condition: {requestMode: "navigate"},
    source: {
      predefinedHeaders: new Headers({
        "Content-Type": "text/html",
        "Cache-Control": "no-cache"
      })
    }
  });
});
```

# ServiceWorker Synthetic Response

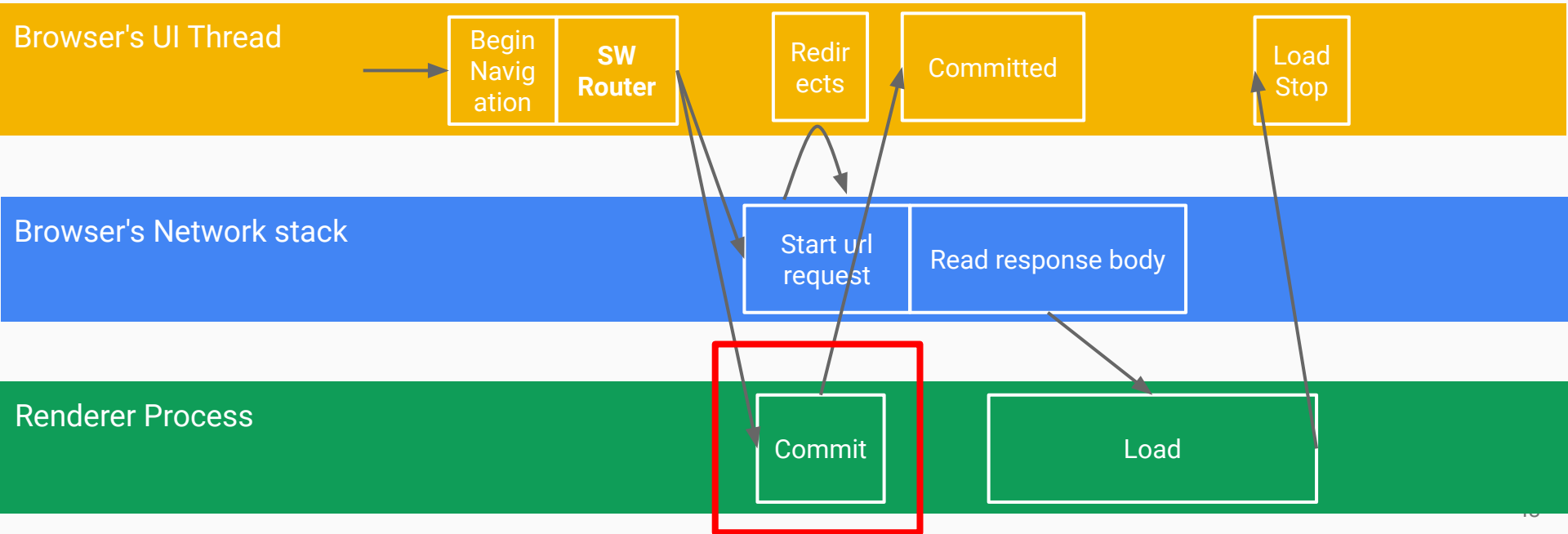


Possible polyfill:

```
onfetch = (event) => {  
  event.respondWith(  
    new Response(synthetic_header, synthetic_body)  
  );  
  fetch(event.request)  
    .then(res => { plumb res.body to synthetic_body; })  
    .catch(e => { Set error_text to synthetic_body; });  
};
```

The Static Routing API returns an early part of the response without waiting for the network response. The rest of response from the network will be appended to the stream.

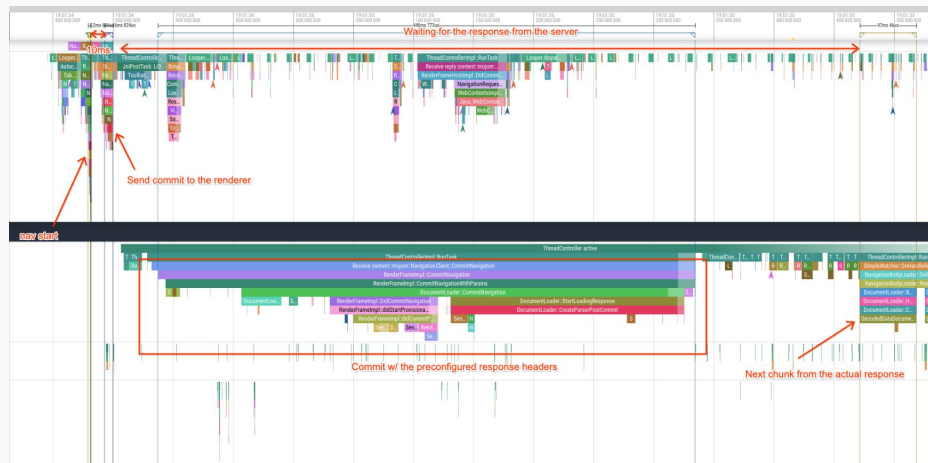
Note: The SW router lookup doesn't involve the SW bootstrap in the navigation critical path.



# ServiceWorker Synthetic Response traces



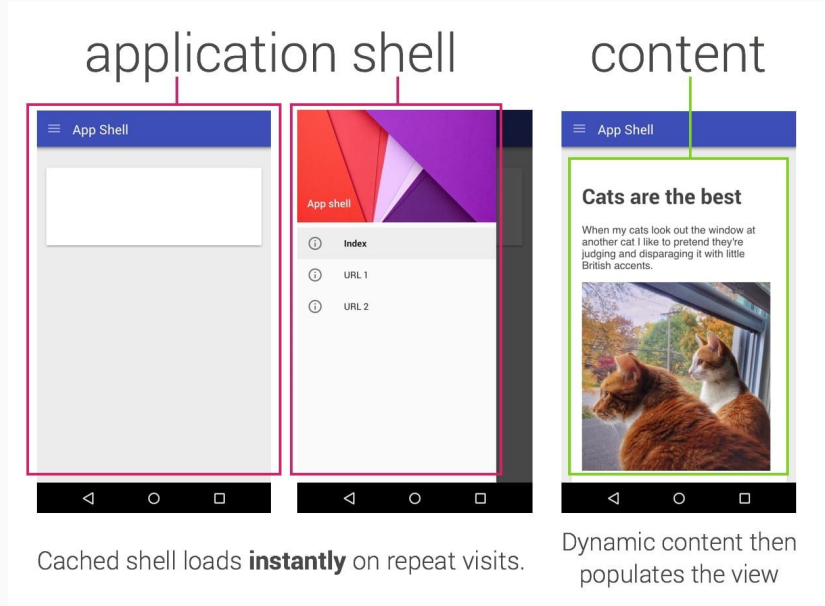
w/o Synthetic Response



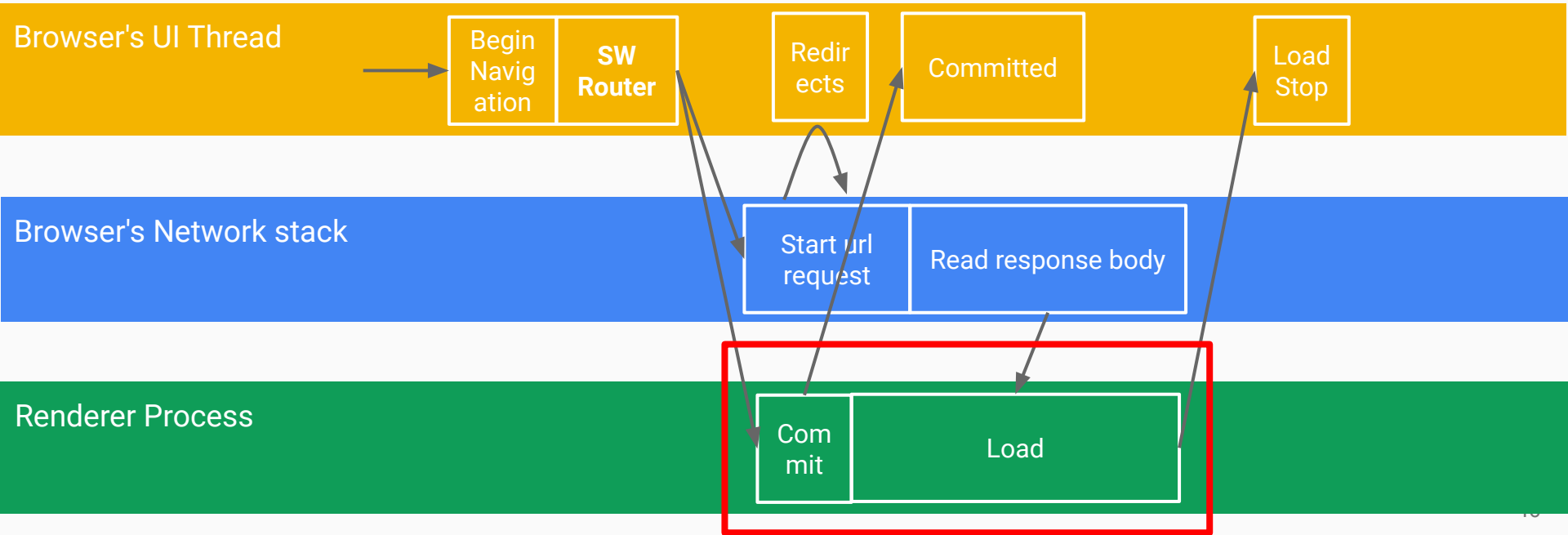
w Synthetic Response

# App Shell via Synthetic Response?

More beneficial if the cached response contains the body, in addition to headers e.g. App Shell.



# How Navigation Works



# ServiceWorker Synthetic Response

- Chrome team is exploring, under prototyping.
- Very early stage, still we have lots of unclear points.
  - What are necessary headers for the navigation commit?
  - Are headers be merged?
  - etc

- Open to any feedback, comments.

- Participate:

<https://github.com/WICG/service-worker-static-routing-api/issues/32>

# Resources

## Static Routing API

- [Explainer](#)
- [Spec](#)
- [Explainer for the Resource Timing](#)
- [Developer Instructions | Chrome for Developers](#)
- [DEMO](#)

## ServiceWorkerAutoPreload

- [Explainer](#)
- [chromestatus](#)

## Synthetic Response

- [Github Issue](#)