Advanced Progressive Web Apps

UPDATING THE APPLICATION

Maximiliano Firtman
MOBILE+WEB DEVELOPER
@firt firt.mobi

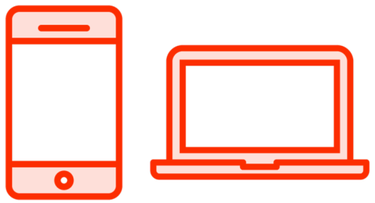
Overview

Updating the Application

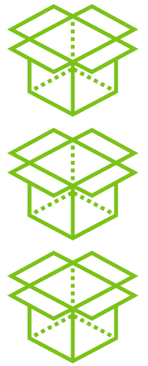
- Understand PWA's Updates
- Update Assets and Builds
- Update the App Launcher Meta Data
- Alert the User with App Badges
- Manage App's Storage

Understand PWA's Updates

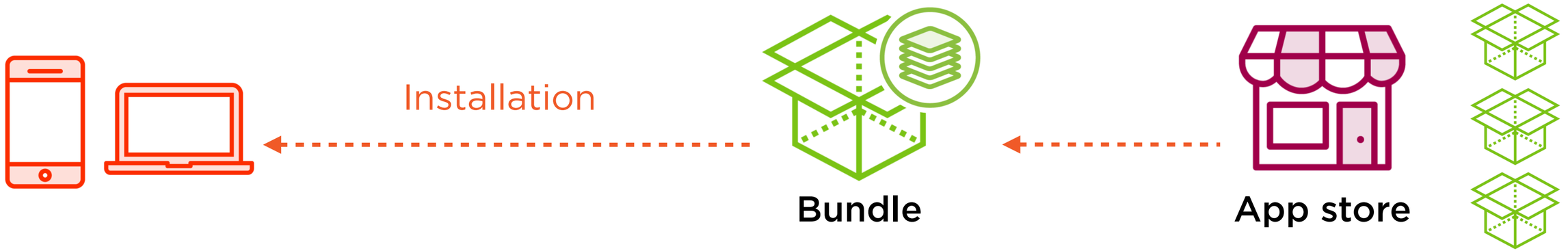
Native Apps



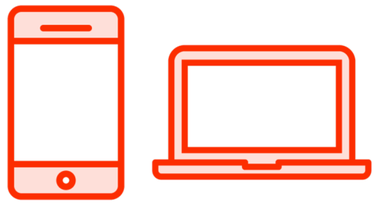
App store



Native Apps

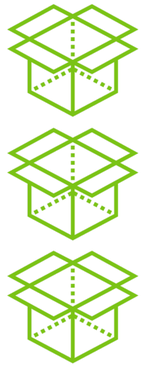


Native Apps

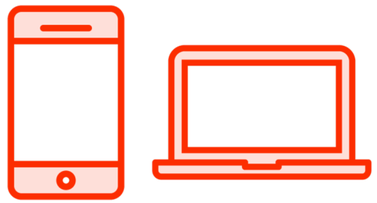


Bundle

App store

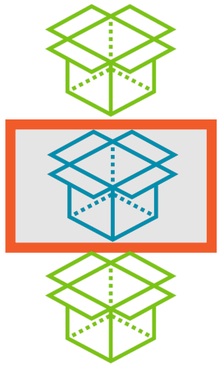


Native Apps

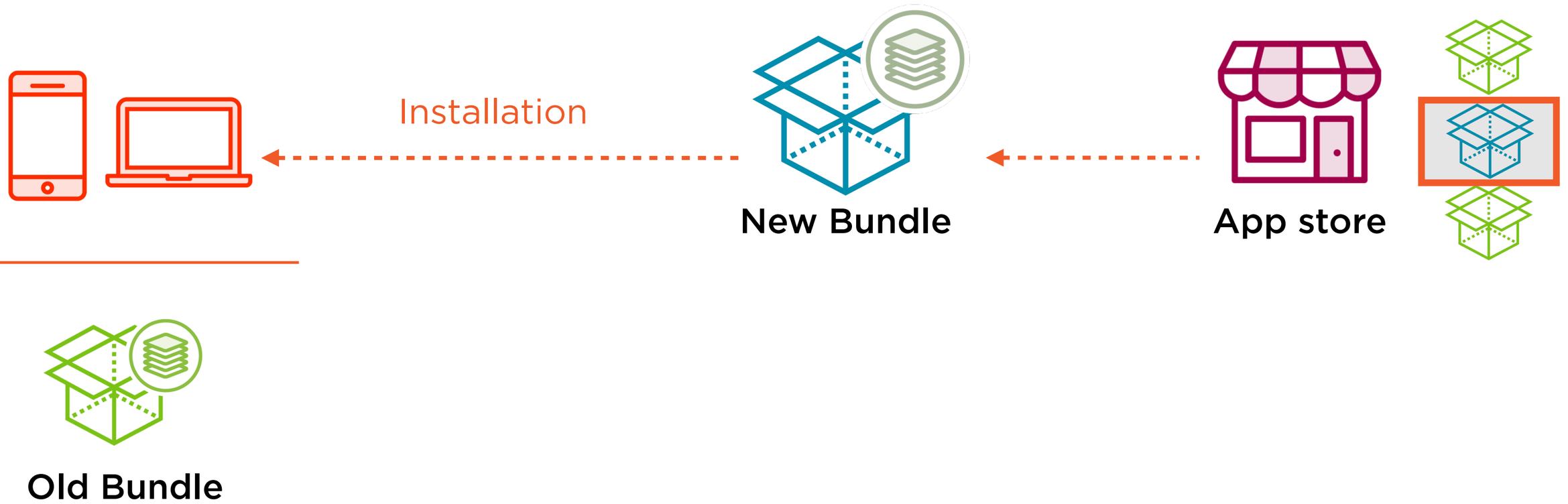


Old Bundle

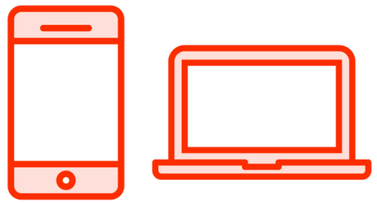
App store



Native Apps



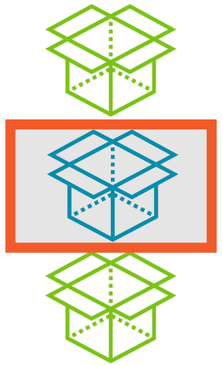
Native Apps



Installation

New Bundle

App store

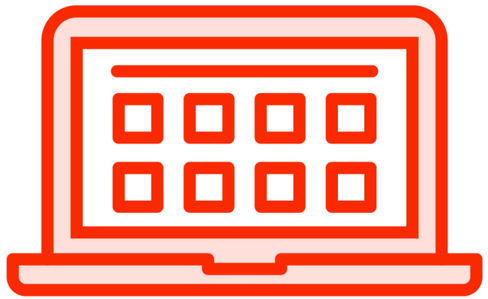


New Bundle

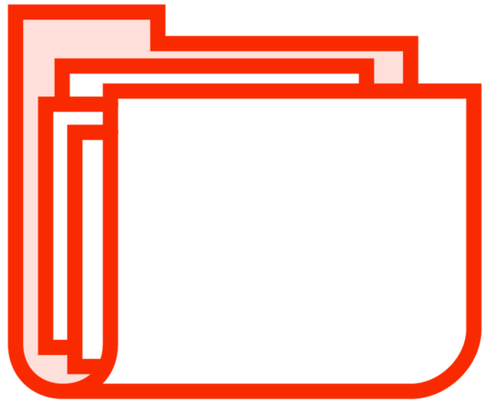
Native Apps must have a version number. You update all the package again when there is a new version.

Developer is in full control of how to cache and serve the resources of the PWA, and how to manage API calls.

Installation Steps for a PWA

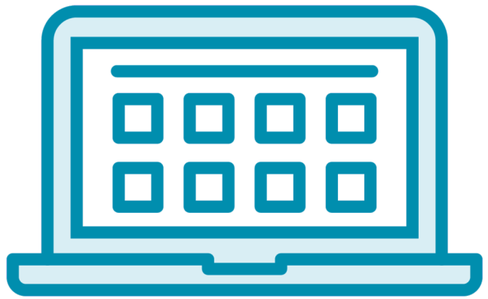


Launcher Icon



App Assets

Update Possibilities for PWAs



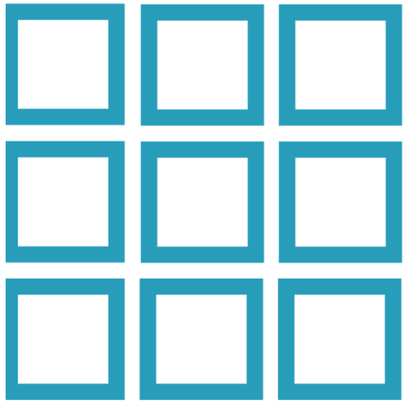
Launcher Icon

App Assets

PWAs are not forced to a version number. Several update design patterns are available.

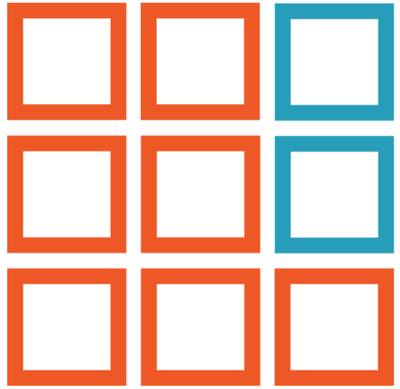
PWA Update Patterns

Every project can have a different technique



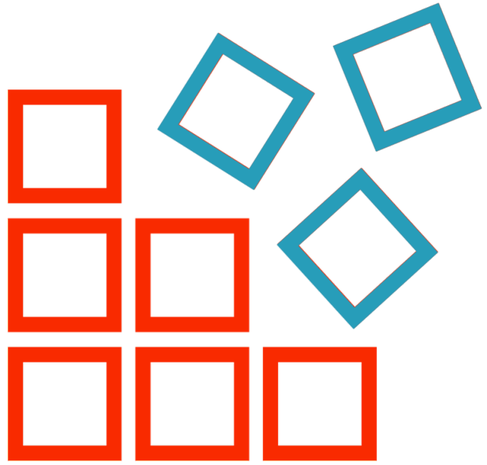
Full Update

We version all the app. We update all files always.



Partial Update

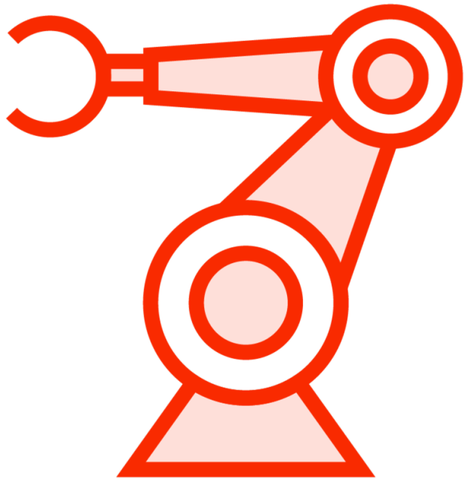
We version with delta changes. We update changes



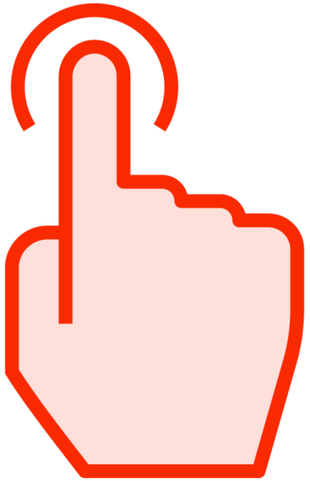
Continuous Update

We version and update assets independently

PWA Update Detection



Automatic



Manual

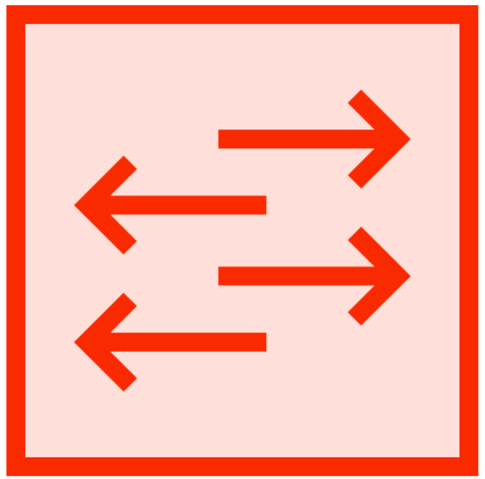
PWA Update Notifications

Ask the User

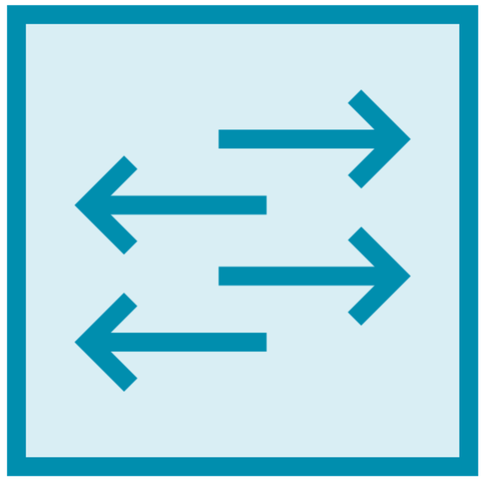
Silent

Notify After

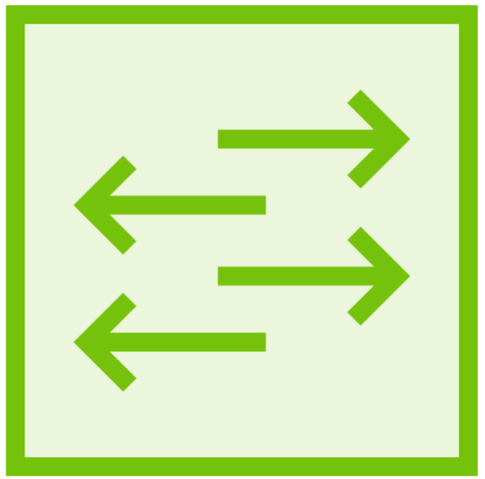
PWA Update Methods



Hot Swap



Next Load
Cold Swap



Background
Cold Swap

Update Assets and Builds

Assets and Builds

HTML & CSS

Images, Web Fonts

Resources

JavaScript

Service Worker

Assets and Builds

HTML & CSS

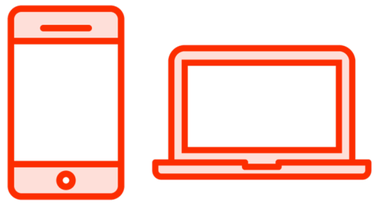
Images, Web Fonts

Resources

JavaScript

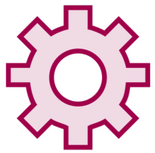
Service Worker

Progressive Web Apps



User's device

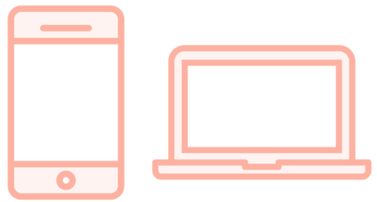
Web runtime



Service worker

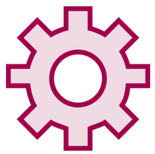
Web server

Progressive Web Apps



User's device

Web runtime

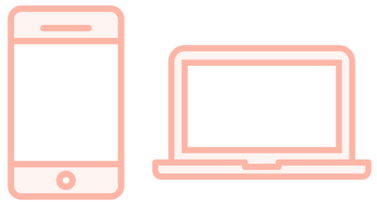


Service worker

Local cache

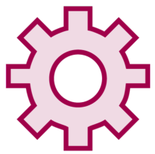
Web server

Progressive Web Apps

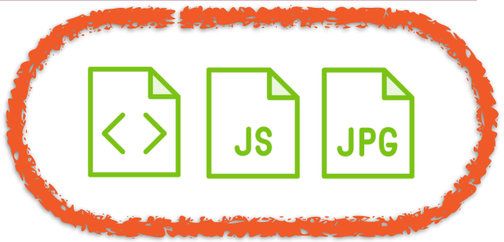


User's device

Web runtime

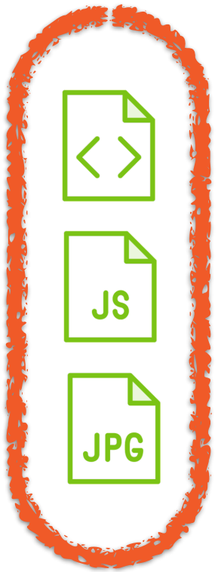


Service worker



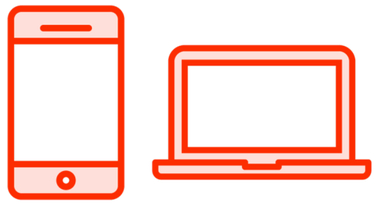
Local cache

Web server

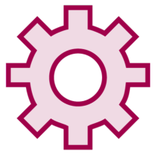


Server filesystem

Progressive Web Apps

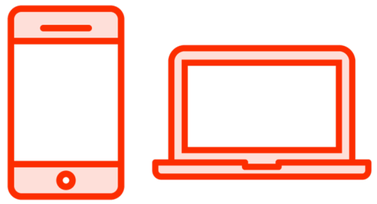


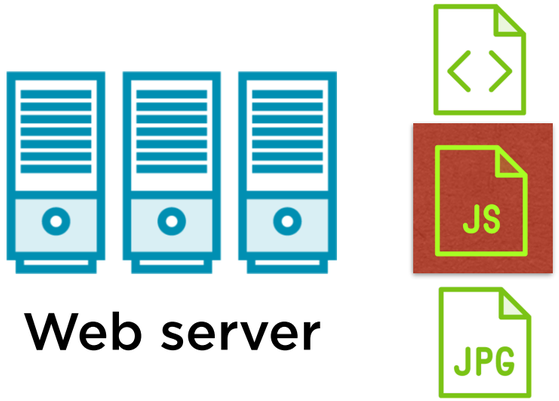
Web server



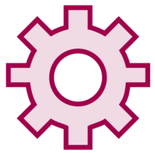
Service worker

Progressive Web Apps



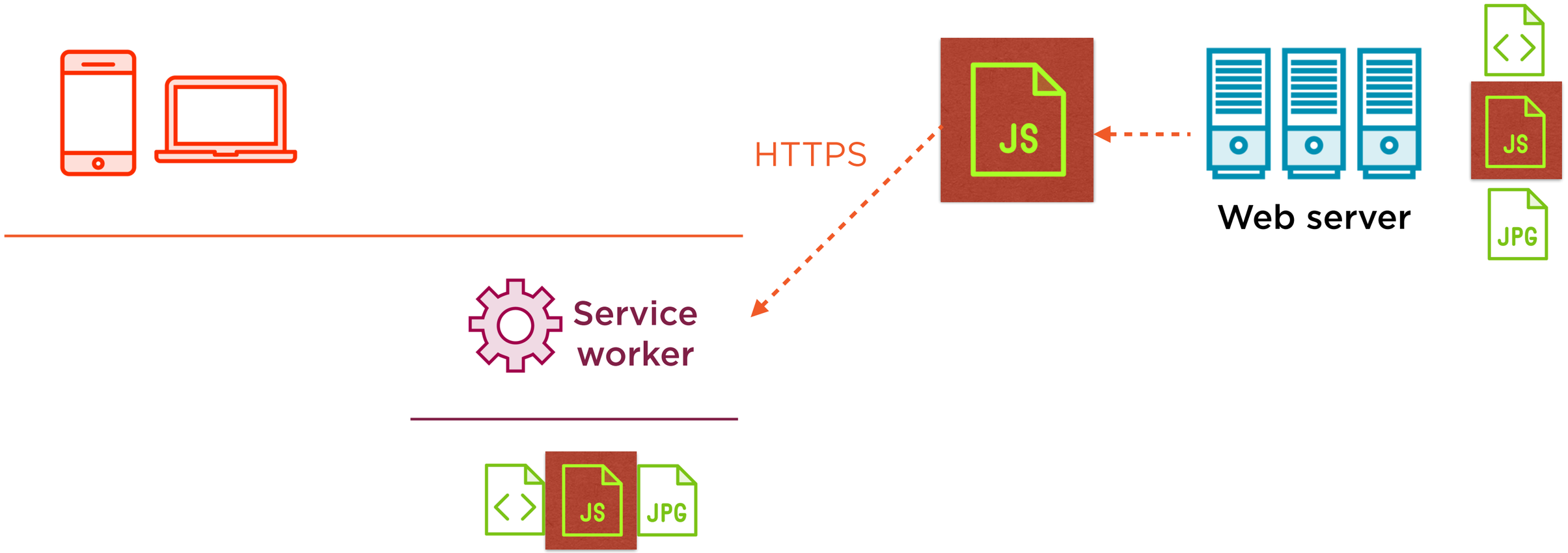


Web server

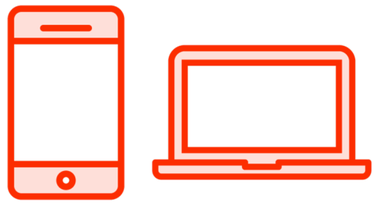


Service worker

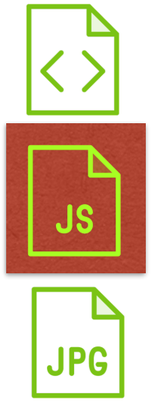
Progressive Web Apps



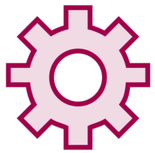
Progressive Web Apps



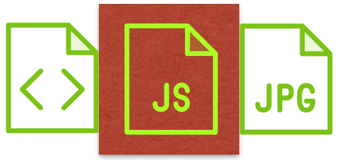
Web server



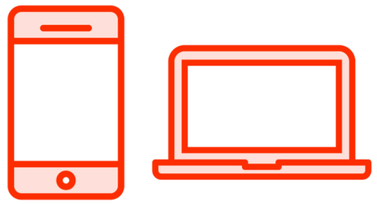
Web runtime

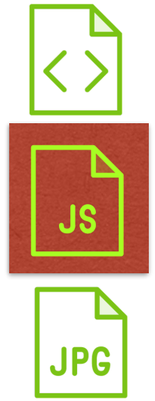


Service worker



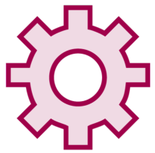
Progressive Web Apps



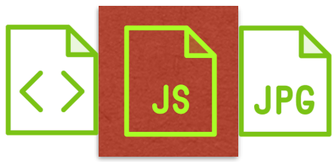


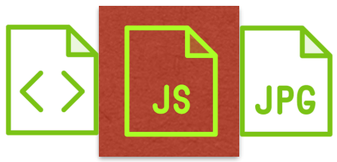
Web server

Web runtime



Service worker



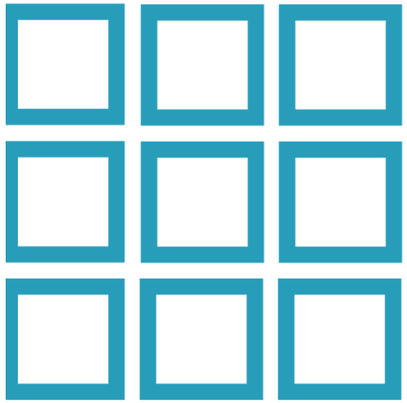


How to detect update in assets?

The Service Worker API has no way to detect changes in individual resources

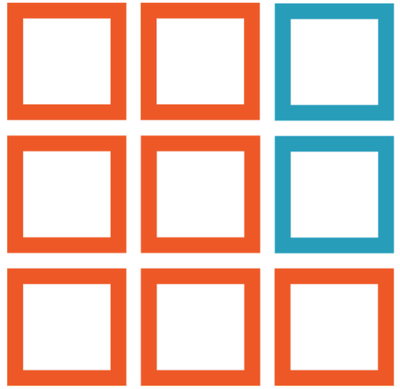
PWA Update Patterns

Every project can have a different technique



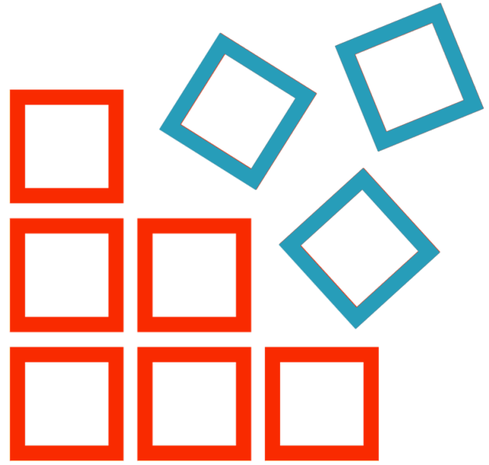
Full Update

We version the SW or the PWA through a configuration file



Partial Update

We version in an asset's manifest file for the PWA



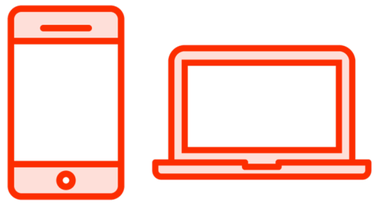
Continuous Update

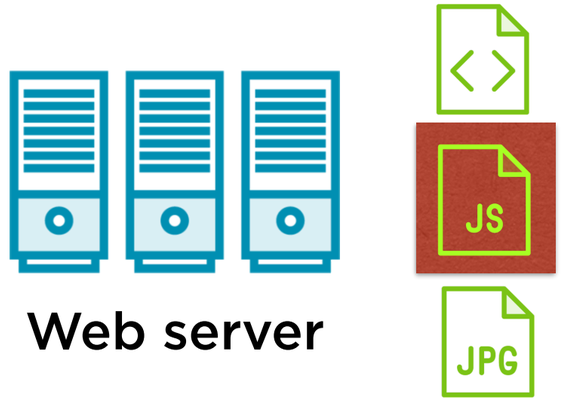
Stale-while-revalidate pattern

When to detect the change?

If we need to check if there is an update in the
PWA assets we have several moments

Progressive Web Apps



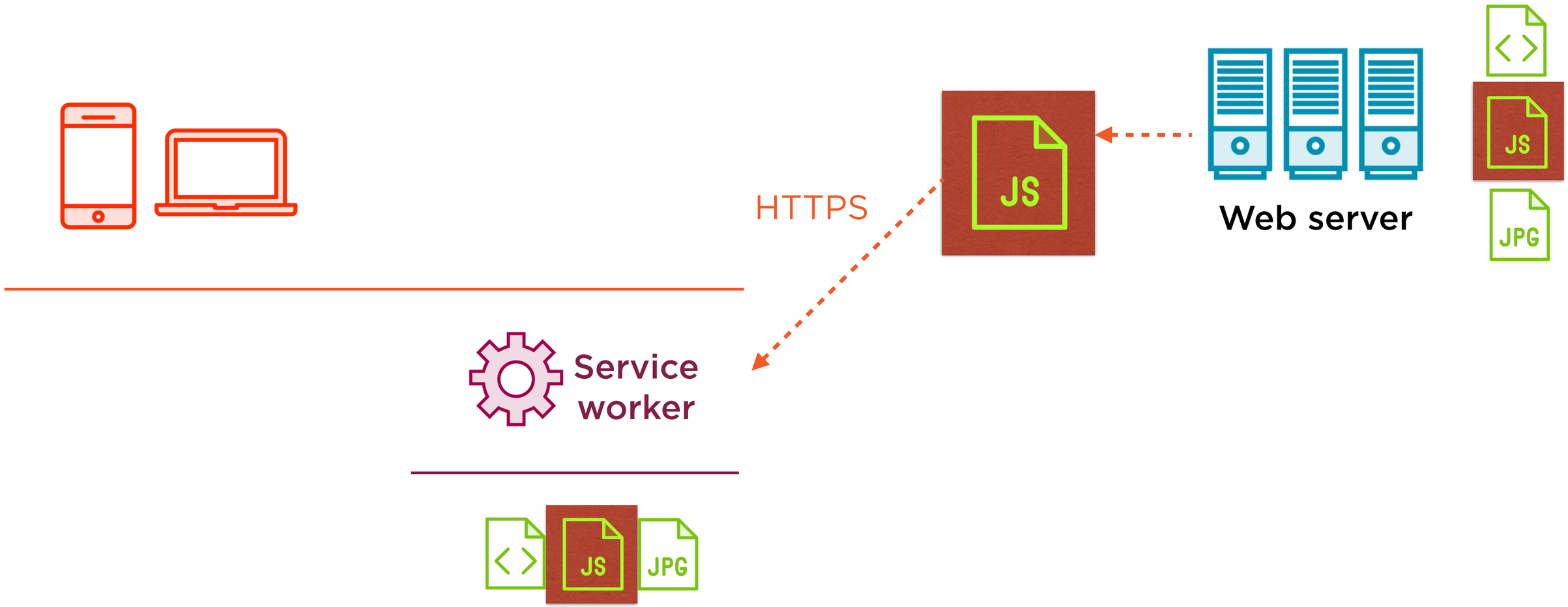


Web server

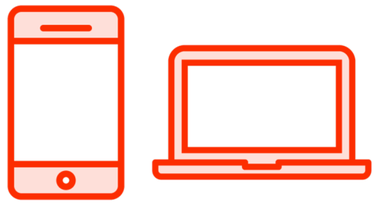
Service worker

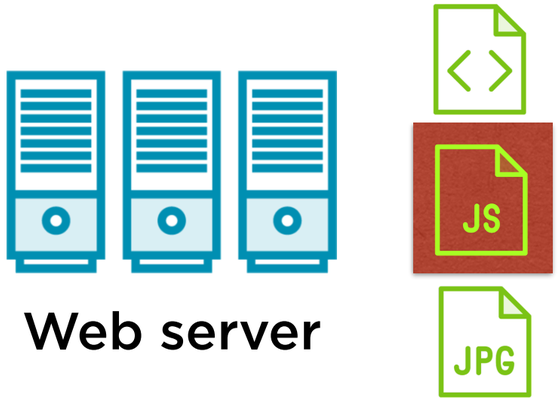
Asset's Manifest
PWA version
meta data

Progressive Web Apps



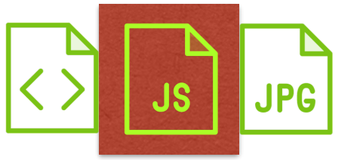
Progressive Web Apps





Web server

Service worker



When to check if updates in assets are available

- When the SW starts
- With a menu item in the UI
- When a push message is received
- Periodically using Background APIs

If you decide to make a hot swap, have in mind possible issues with dependencies and different versions of same resource in memory

Update the Service Worker

How Service Worker update works?

The browser manages updating the service worker registration with a new one

Service Worker's Update Algorithm

Managed automatically by the browser

Algorithm differs per browser

Typically, the algorithm checks for an update

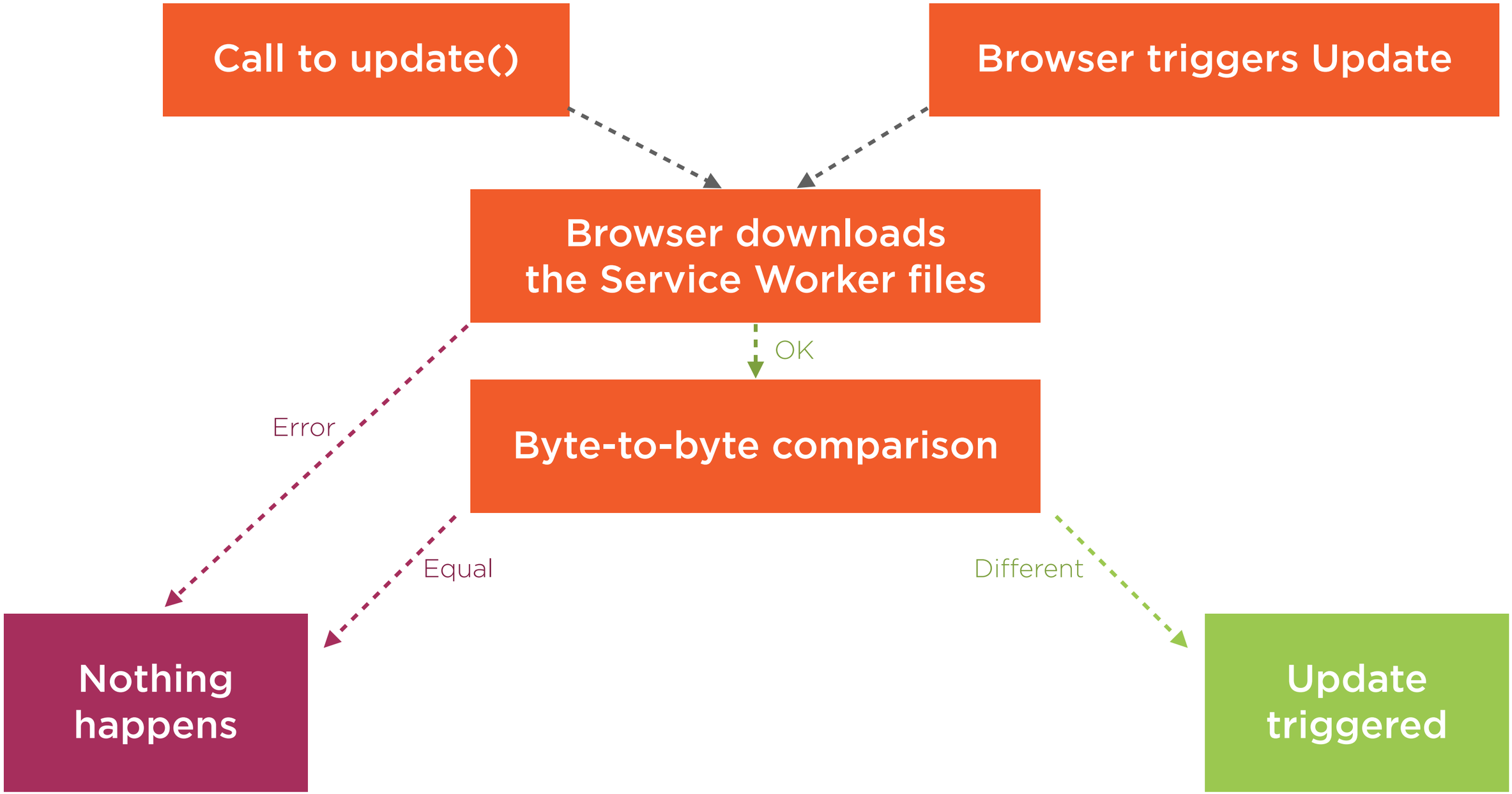
- On every new navigation to a PWA
- On a background event, such as push or sync, once every 24 hours
- If you call update() on the SW's registration

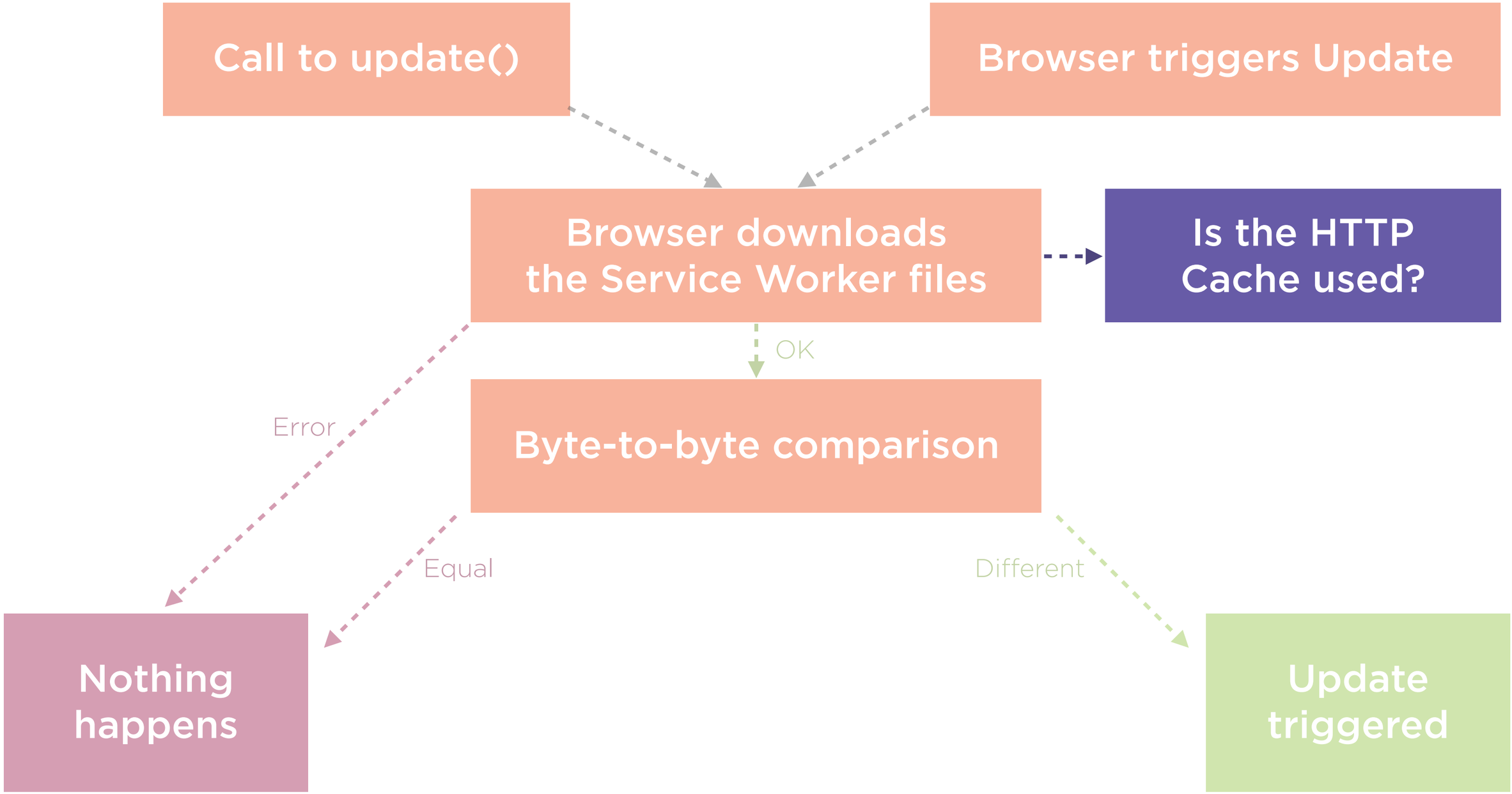
Request a Service Worker's Update Check

Not available on every browser

script.js

```
const registration = await navigator.serviceWorker.ready;
if ('update' in registration) {
    registration.update();
}
```


Call to update()
Browser triggers Update
Browser downloads the Service Worker files
OK
Byte-to-byte comparison
Error
Equal
Different
Nothing happens
Update triggered


Call to update()
Browser triggers Update
Browser downloads the Service Worker files
Is the HTTP Cache used?
OK
Error
Byte-to-byte comparison
Equal
Different
Nothing happens
Update triggered

Some browsers ignore the HTTP headers that defines service worker's file cache policy. Some other browsers use them with a maximum of 24 hours

Update Algorithm Policy for Dependencies

The Service Worker can have dependencies on other files

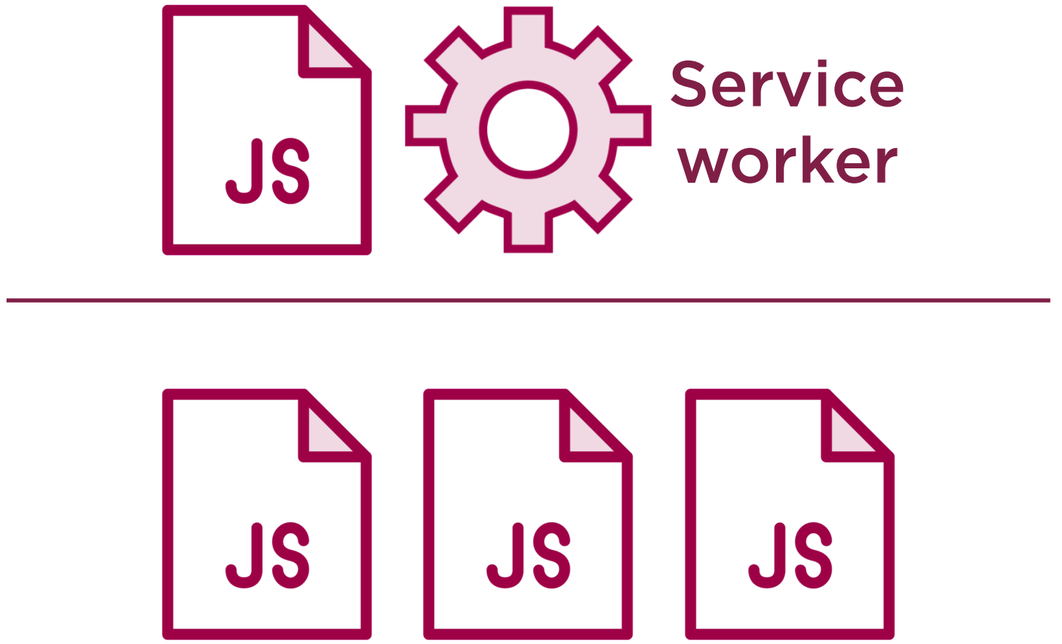
script.js

```
navigator.serviceWorker
      .register('sw.js');
```

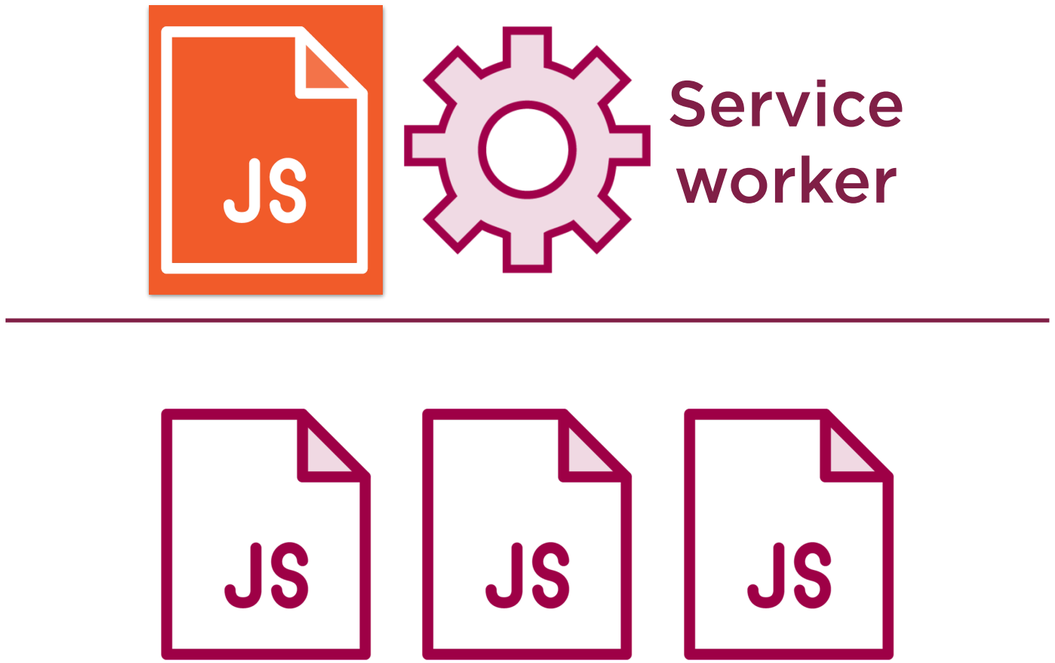
sw.js

```
importScripts("sw-events.js");
importScripts("sw-fetch.js");
importScripts("sw-push.js");
```

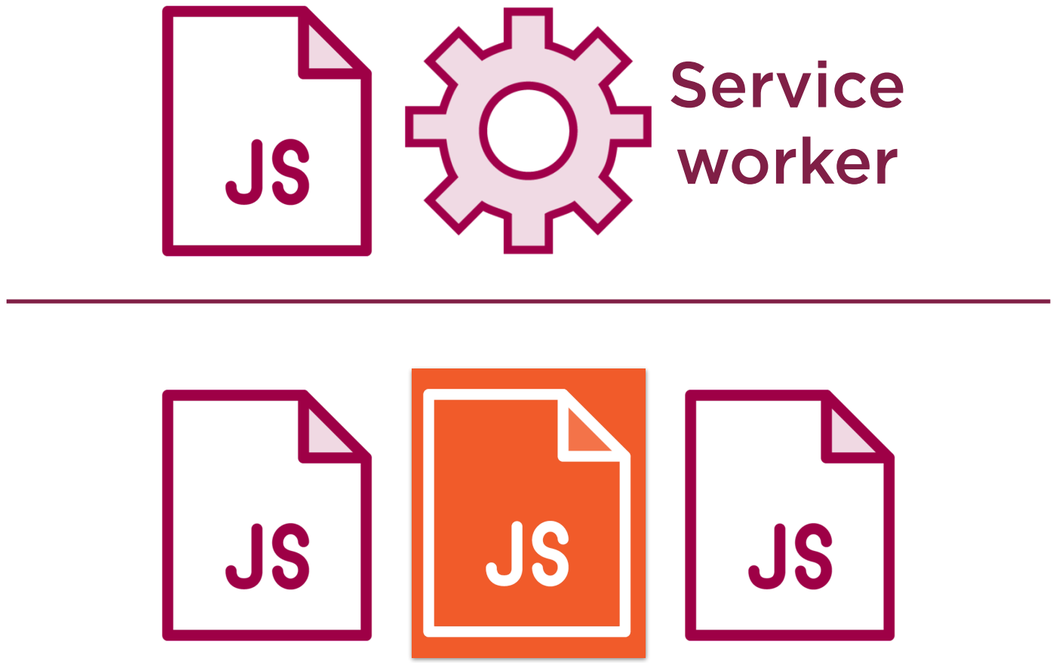
Service Worker's Dependencies



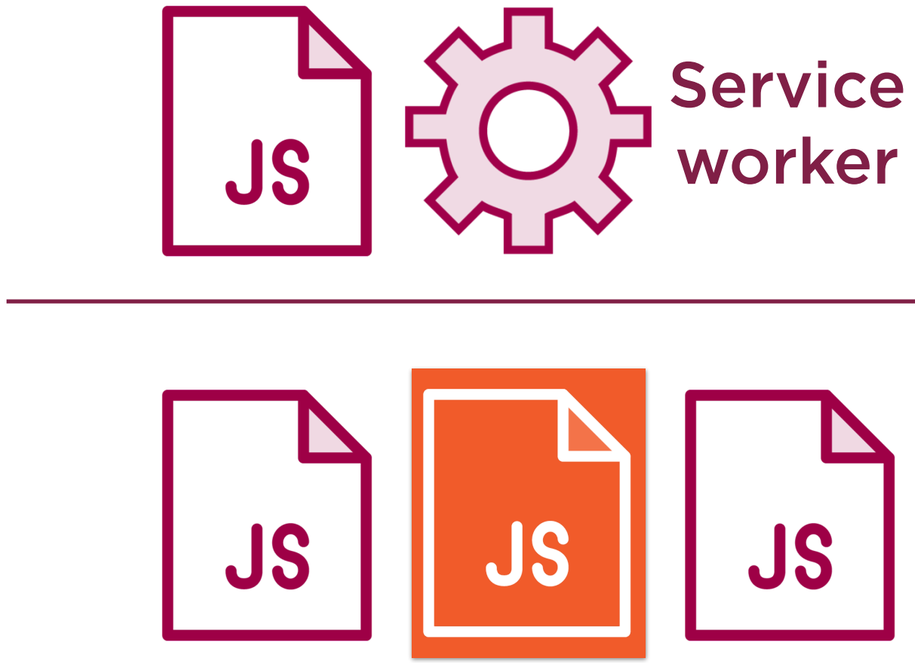
Service Worker's Dependencies



Service Worker's Dependencies



Service Worker's Dependencies



The HTTP Headers defining cache policies might be a problem

Update Algorithm Policy for Dependencies

Comparison will include dependences but using HTTP Cache

script.js

```
navigator.serviceWorker
    .register('sw.js');
```

sw.js

```
// The HTTP Cache WON'T be used
// for the Service Worker file
// The HTTP Cache WILL be used
// for dependencies on update

importScripts("sw-events.js");
importScripts("sw-fetch.js");
importScripts("sw-push.js");
```

Update Algorithm Policy for Dependencies

Comparison will include dependences but using HTTP Cache

script.js

```
navigator.serviceWorker
    .register('sw.js', {
        updateViaCache: 'imports'
    });
```

sw.js

```
// The HTTP Cache WON'T be used
// for the Service Worker file
// The HTTP Cache WILL be used
// for dependencies on update

importScripts("sw-events.js");
importScripts("sw-fetch.js");
importScripts("sw-push.js");
```

Update Algorithm Policy for Dependencies

Comparison will include dependences but using HTTP Cache

script.js

```
navigator.serviceWorker
    .register('sw.js', {
        updateViaCache: 'all'
    });
```

sw.js

```
// The HTTP Cache WILL be used
// for the Service Worker file
// The HTTP Cache WILL be used
// for dependencies on update

importScripts("sw-events.js");
importScripts("sw-fetch.js");
importScripts("sw-push.js");
```

Update Algorithm Policy for Dependencies

Comparison will include dependences but using HTTP Cache

script.js

```
navigator.serviceWorker
    .register('sw.js', {
        updateViaCache: 'none'
    });
```

sw.js

```
// The HTTP Cache WON'T be used
// for the Service Worker file
// The HTTP Cache WON'T be used
// for dependencies on update

importScripts("sw-events.js");
importScripts("sw-fetch.js");
importScripts("sw-push.js");
```

What happens after a Service Worker is updated?

It stays in 'waiting' state until a next PWA usage makes it active

Make the new Service Worker Active ASAP

Be careful!

serviceworker.js

```
self.addEventListener('install', event => {
  self.skipWaiting();
  // The rest of your install event
});
```

If you don't want to update all the assets, don't cache them on the install event without browsing your cache first

Be careful with skipWaiting as you might have a PWA loaded with an old service worker that will continue working with a new version from that moment

How to detect when a SW update happens?

In the Service Worker, install event is enough signal; but in the UI we can also detect it

Detect when a Service Worker's update was installed

It let us know in the UI that an update is ready to be activated

script.js

```
const registration = await navigator.serviceWorker.ready;

registration.addEventListener("updatefound", event => {









})
```

Detect when a Service Worker's update was installed

It let us know in the UI that an update is ready to be activated

script.js

```
const registration = await navigator.serviceWorker.ready;

registration.addEventListener("updatefound", event => {
    const newSW = registration.installing;
    newSW.addEventListener("statechange", event => {




    });
})
```

Detect when a Service Worker's update was installed

It let us know in the UI that an update is ready to be activated

script.js

```
const registration = await navigator.serviceWorker.ready;

registration.addEventListener("updatefound", event => {
    const newSW = registration.installing;
    newSW.addEventListener("statechange", event => {
        if (newSW.state=="installed") {
            track("sw-update", "installed and waiting");
            // You can show a user a notification here
        }
    });
})
```

Detect when a new Service Worker is now activated

Let a page knows that now is owned by a new controller

script.js

```
navigator.serviceWorker
    .addEventListener("controllerchange", event => {
       track("sw-update", "activated and controlling the PWA");
    });
```

Update Summary

There are no silver bullets

We need to define our custom Update Policy

Service Worker's update will be managed automatically by the browser

Asset's updates need manual management

- You update all assets with a new SW
- You create an assets manifest and update files in groups
- You use stale-while-revalidate pattern

You can update silently or asking the user, with hot swap or cold swap, in the foreground or in the background

Update App Launcher Meta Data

You want to update

App's Name

Icons

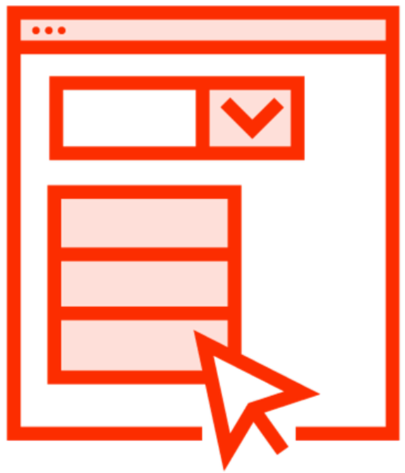
URL

Colors

App Shortcuts

Other meta data

Distribution Models



Browser

Enterprise

App store

Updates to App Launcher for Enterprise and App Stores

You need to rebuild your PWA Launcher

Upload it to the store with a new version number

Wait for approval

Wait for your users to get the new version

Updates to App Launcher for Browser-installed PWAs

We need to publish a new version of the Web App Manifest

What will happen with installed PWAs?

It depends on the platform and browser

iOS and iPadOS: re-install needed,
launcher won't update

Android: based on the app launcher type

Desktop: re-install needed,
in the future it may regenerate automatically

Updating the App Manifest on Android Browsers

Google Chrome and WebAPK

- Every new day you open the PWA, the browser downloads the App Manifest and compare properties' values
- If there is an update in one property, the WebAPK will be marked as needing update
- After you closed the App, Chrome will request a new WebAPK to the mint server
- It will install the new app silently when the device is plugged and on Wi-Fi

Updating the App Manifest on Android Browsers

Samsung Internet and WebAPK

- Check updates when you open PWA
- WebAPK is updated on Wi-Fi in the next 24 hours

Other browsers and Google Chrome with Shortcuts

- re-install is needed, launcher won't update

In case the manifest can't be retrieved and the WebAPK can't be generated a couple of times, Chrome will move to a minimum 30 day period to check for updates again

WebAPK debug information

Google Chrome

(Android)

chrome://webapks

Samsung Internet

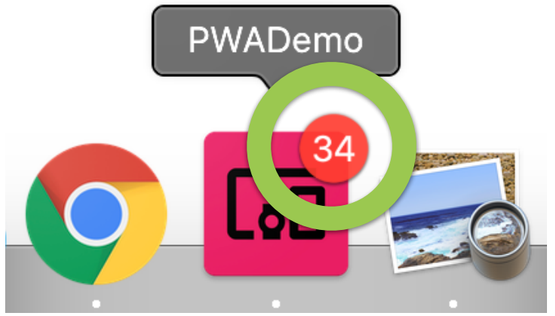
(on Samsung devices only)

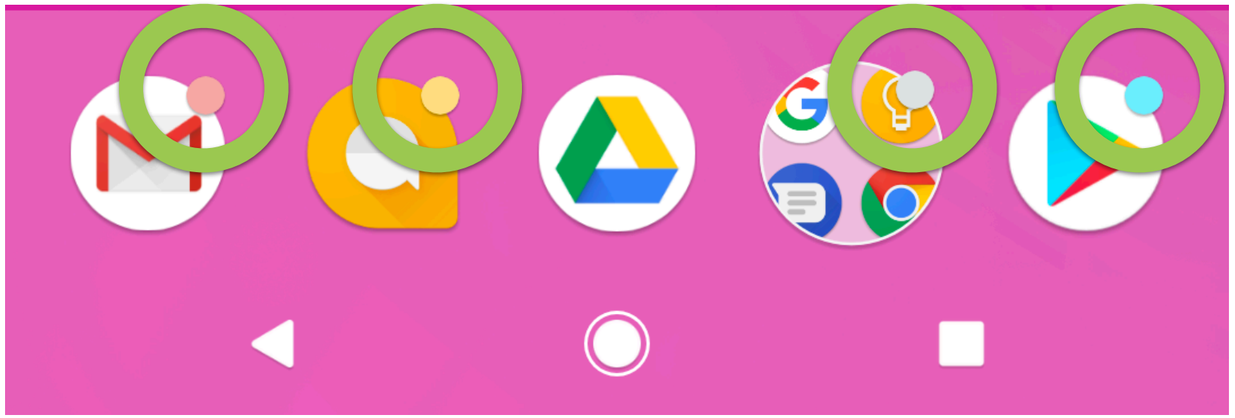
internet://webapks

If you change the manifest's URL from your HTML, the installed PWA is considered obsolete and it won't be updated in the future

Alert the User with App Badging

What's Icon Badging





Web Badging

It lets us alert the user that there is new content or pending tasks in our app

It's not yet an official standard

It's available in Google Chrome and Microsoft Edge

- Android
- Windows
- macOS

When to Update the Badge

When you are using the PWA

Using Periodic Sync

When a Web Push Message is received

Define a New Badge for the App's Icon

We can do this in the Page or in the context of a Service Worker

script.js

```
if ('setAppBadge' in navigator) {
  navigator.setAppBadge(24)
}
```

Define a New Badge for the App's Icon

We can do this in the Page or in the context of a Service Worker

script.js

```
if ('setAppBadge' in navigator) {
  navigator.setAppBadge(24).catch(error => {
    // Do something on error
  });
}
```

Define a New Badge for the App's Icon

We can do this in the Page or in the context of a Service Worker

script.js

```
if ('clearAppBadge' in navigator) {
  navigator.clearAppBadge().catch(error => {
    // Do something on error
  });
}
```

Final Results

Showing the number is not guaranteed

Setting the badge as zero will clear it

Platforms are using a best effort scenario

Some platforms, such as some Android devices don't support badges, so they use a notification dot instead

Other platforms might crop or display large numbers in different ways, such as showing 99+ in the badge instead of 1285

Manage App's Storage

The App's Storage

Service Worker's Registration

App Assets in Cache Storage

Data in Web Storage or IndexedDB

Storage details

Today, all storages are under the same quota

Defined per origin

Quota are different per browser:

- **Chrome:** 60% of total disk space
- **Firefox:** 2GB
- **Safari:** 750MB with increments of 200Mb with user's permission

Storage life

- **Storage can be defined as "Best Effort" (default) or "Persistent"**
- **Best Effort can clear the storage**
  - On Storage Pressure (low storage)
  - After some time of inactivity
  - With user intervention
- **Persistent will keep storage unless**
  - User intervention happens

iOS and iPadOS

With Safari, Best Effort

Eviction can happen:

- On Storage Pressure
- After 7 days of inactivity
- Settings ➡ Safari ➡ Clear

With Installed PWA, Persistant Storage

Eviction can happen:

- Settings ➡ Safari ➡ Clear

Firefox and Chromium-based browsers

By default, Best Effort

Eviction can happen:

- On Storage Pressure
- Using Settings ➡ Clear
- When uninstalling the PWA, the user may have the option to delete the data

Persistant Storage can be requested by API

Eviction can happen:

- Using Settings ➡ Clear
- When uninstalling the PWA, the user may have the option to delete the data

Persistent Storage Request

Firefox will ask the user, Chromium will grant or deny based on criteria

script.js

Persistent Storage Request

Firefox will ask the user, Chromium will grant or deny based on criteria

script.js

```
const granted = await navigator.storage.persist();
track('storage-persist-request', granted);
```

Persistent Storage Request

Firefox will ask the user, Chromium will grant or deny based on criteria

script.js

```
if (navigator.storage && navigator.storage.persist) {
  const granted = await navigator.storage.persist();
  track('storage-persist-request', granted);
}
```

Ask Current Persistent Storage Status

script.js

Ask Current Persistent Storage Status

script.js

```
if (navigator.storage && navigator.storage.persist) {
  const isPersisted = await navigator.storage.persisted();
  track('storage-persisted', isPersisted);
}
```

Ask Quota Information

Available on some browsers

script.js

Ask Quota Information

Available on some browsers

script.js

```
const q = await navigator.storage.estimate();
track('quota available', q.quota);
track('quota usage', q.usage);
```

Ask Quota Information

Available on some browsers

script.js

```
if (navigator.storage && navigator.storage.estimate) {
  const q = await navigator.storage.estimate();
  track('quota available', q.quota);
  track('quota usage', q.usage);
}
```

The Storage APIs return promises and we are using await; remember to wrap those calls in an async function

There is no way to disable persistent storage once it was granted

Chromium criteria for Persistent Storage

Persistent Storage will be granted if

- It's an installed PWA
- It's in the bookmarks
- Push permission has been granted
- It has high site engagement

Summary

Updating the Application

- Understand PWA's Updates
- Update Assets and Builds
- Update the App Launcher Meta Data
- Alert the User with App Badges
- Manage App's Storage

Up Next:
Integrating with Hardware and Platforms