

# Advanced Progressive Web Apps

---

## MANAGING APP'S LIFECYCLE



**Maximiliano Firtman**

MOBILE+WEB DEVELOPER

@firt firt.mobi

# Overview

## Managing App's Lifecycle

- Understand the PWA Lifecycle
- Page Visibility and Page Lifecycle APIs
- Track Usage for Analytics
- Improve Performance with Navigation Preload
- Web Push Notifications
- Background Execution: Sync, Fetch and Periodic Sync



# Understand the Lifecycle

---

# Why do we need to understand the life cycle

**Improve User  
Experience**

**Save resources**

**Stop timers and  
pending requests**

App



App



App



App



---

Operating System

App



**Active App**



App



App



---

Operating System

App



**Active PWA**



App



App



---

Operating System

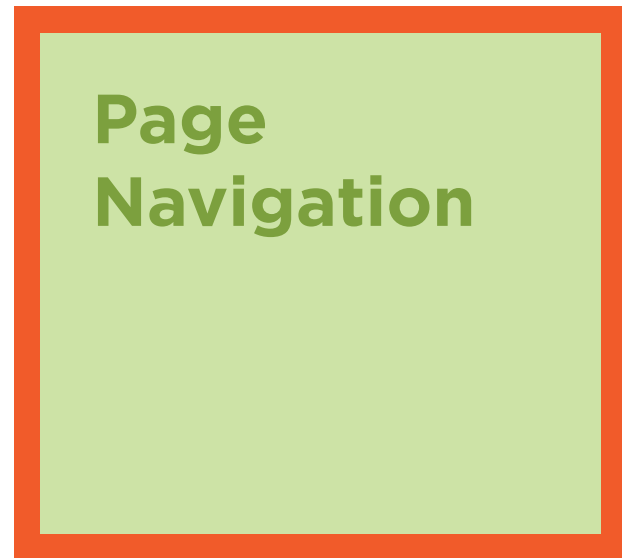


On Single Page Applications,  
only one Page Navigation  
instance will exist

App



**Active PWA**



App



App



---

Operating System

App



**Active PWA**  
Single Page App



App



App



---

Operating System

App



**Active PWA**  
Single Page App



App



App



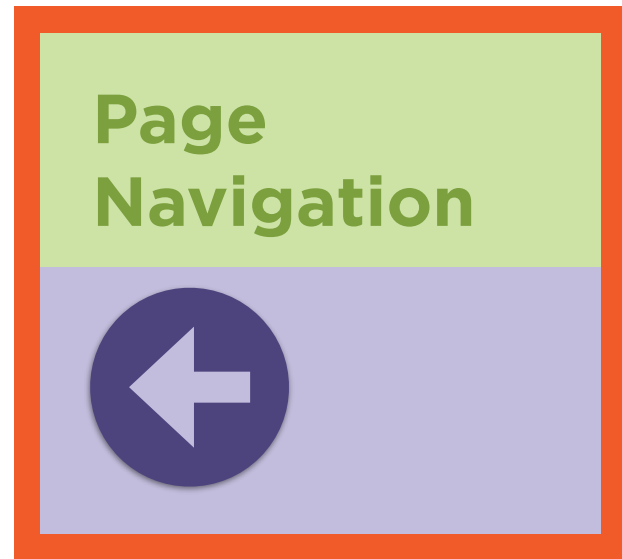
---

Operating System

App



**Active PWA**  
Single Page App



App



App



---

Operating System

App



**Active PWA**  
Single Page App



App



App



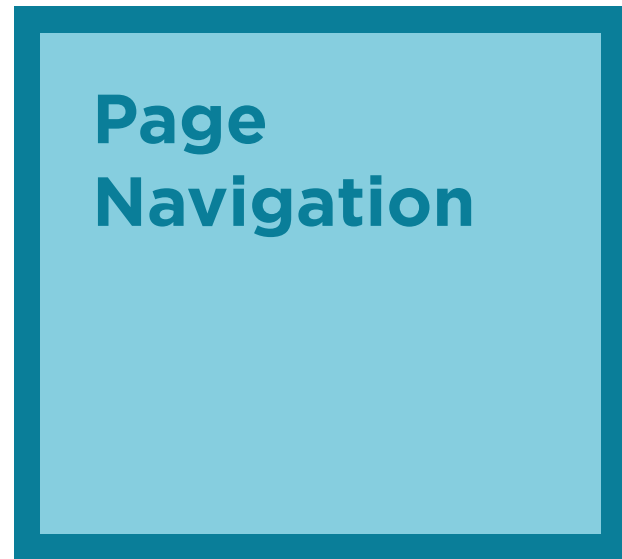
---

Operating System

App



PWA  
Single Page App



**App**



App



---

Operating System

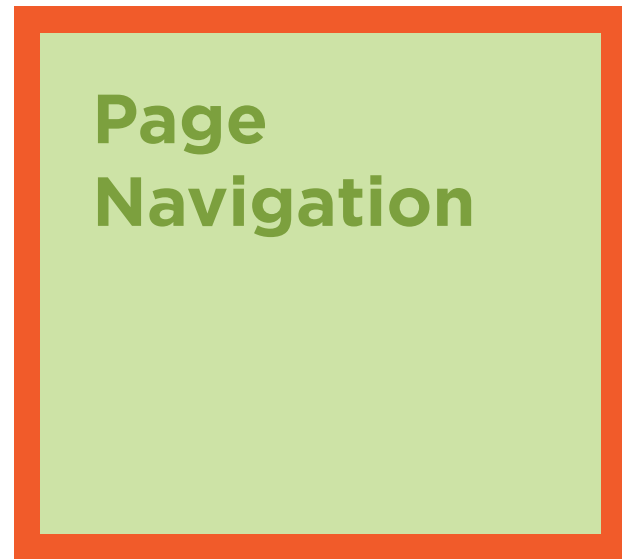
On Multi Page Applications,  
each navigation will have its  
own Page Navigation and  
Lifecycle



App



**Active PWA**  
Multi Page App



App



App



---

Operating System

App



**Active PWA**  
Multi Page App



App



App



---

Operating System

App



**Active PWA**  
Multi Page App



App



App



---

Operating System

App



**Active PWA**  
Multi Page App



App



App



---

Operating System

# App Lifecycle is Different



Desktop

Mobile

---

# Desktop Operating System

App



App



**App**

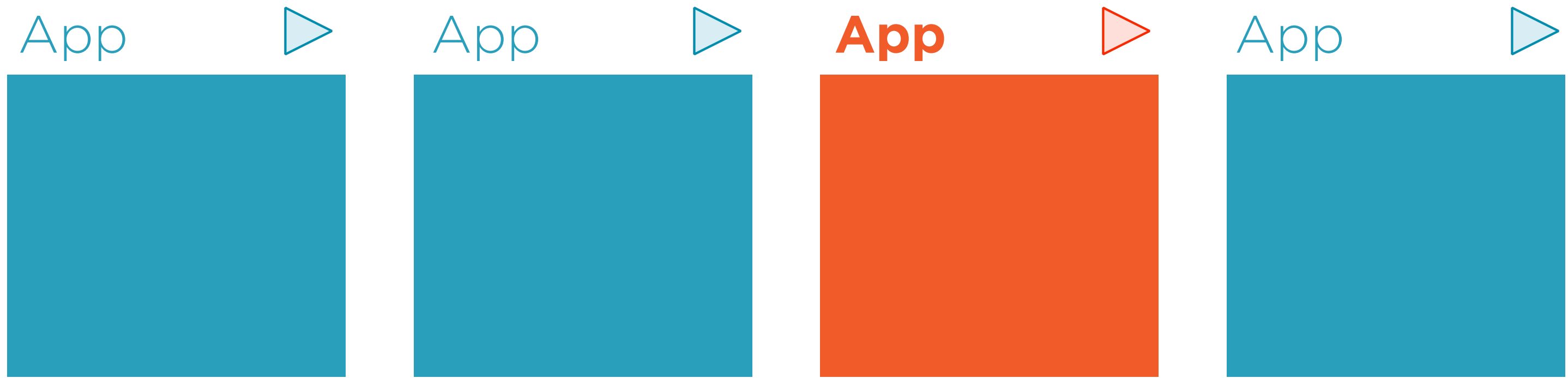


App



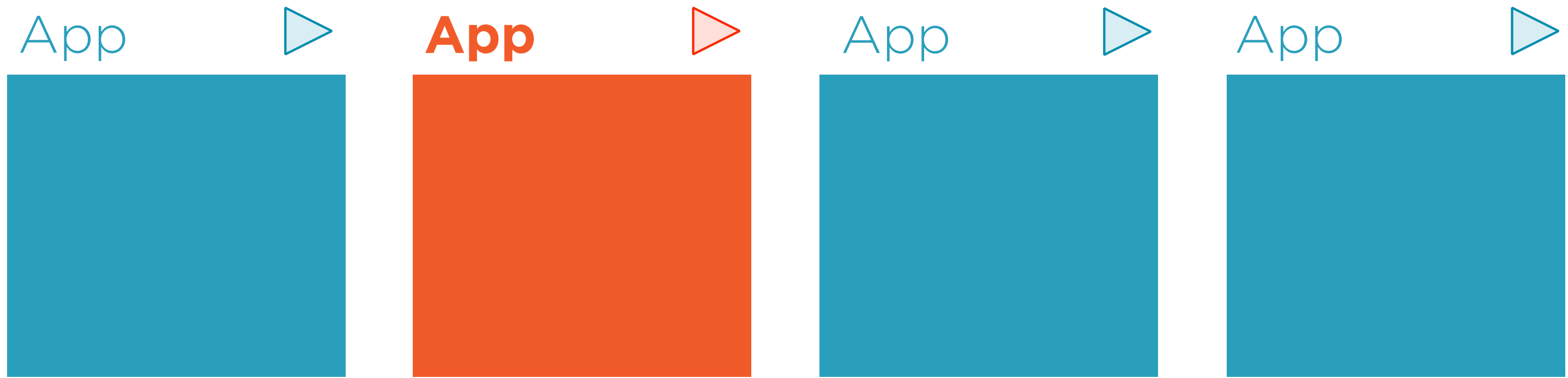
---

Desktop Operating System



Desktop Operating System





Desktop Operating System

---

# Mobile Operating System

App



App



**App**

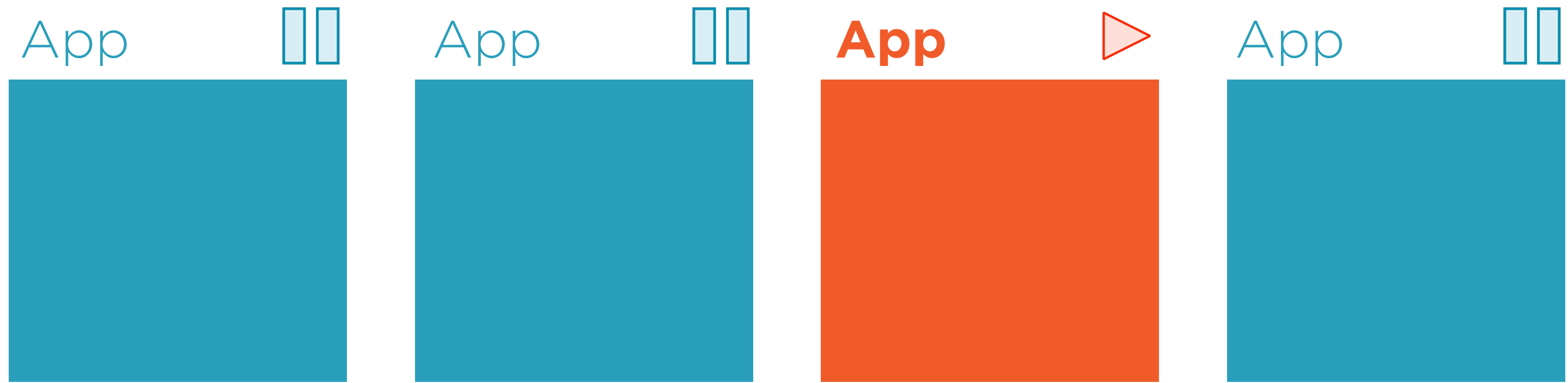


App

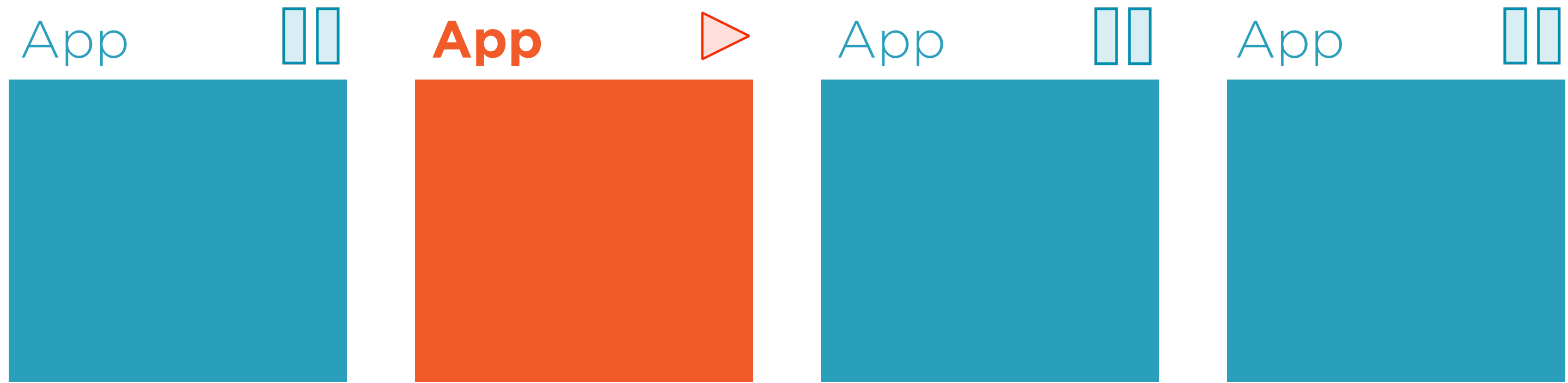


---

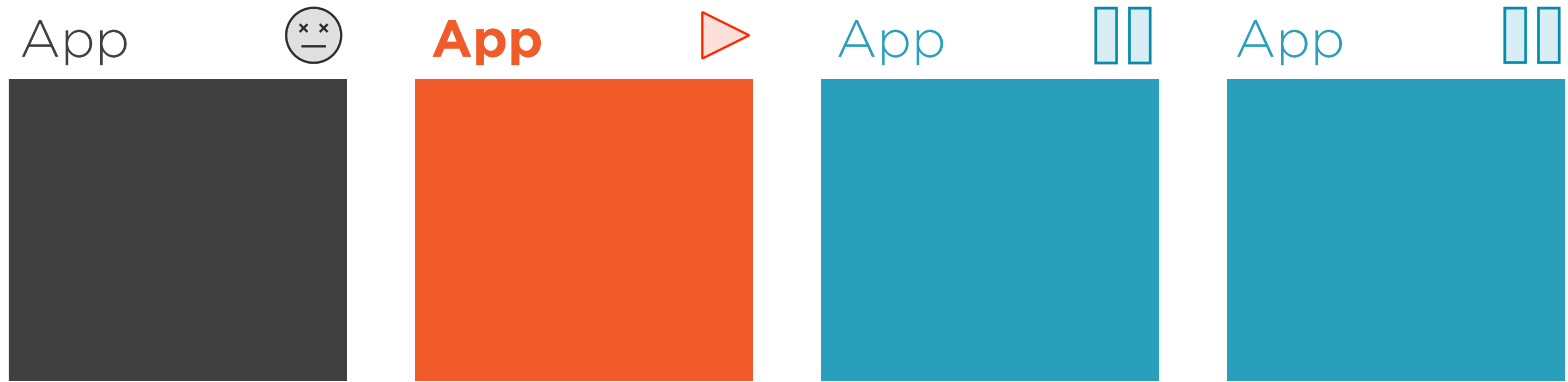
Mobile Operating System



Mobile Operating System

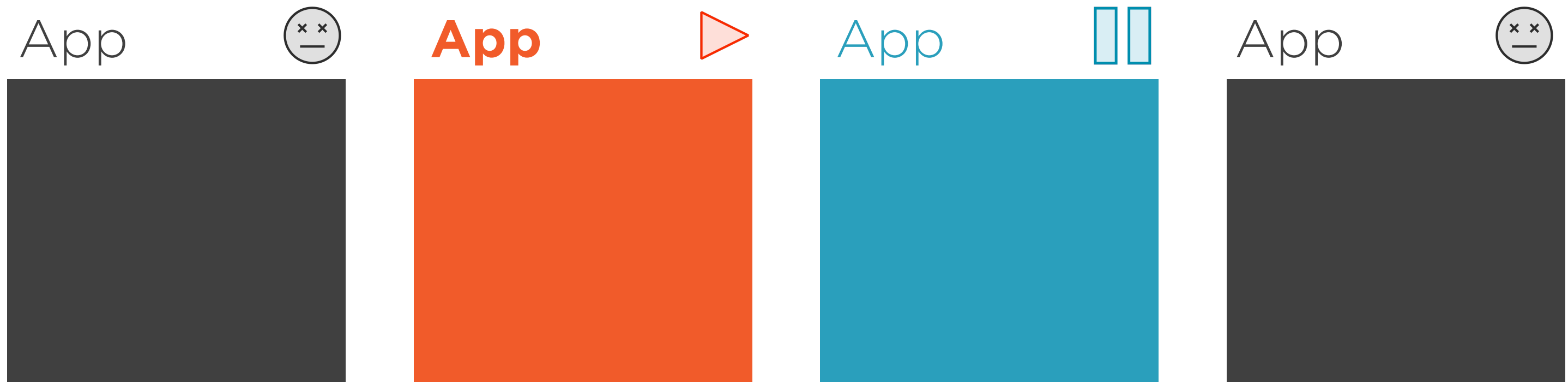


Mobile Operating System



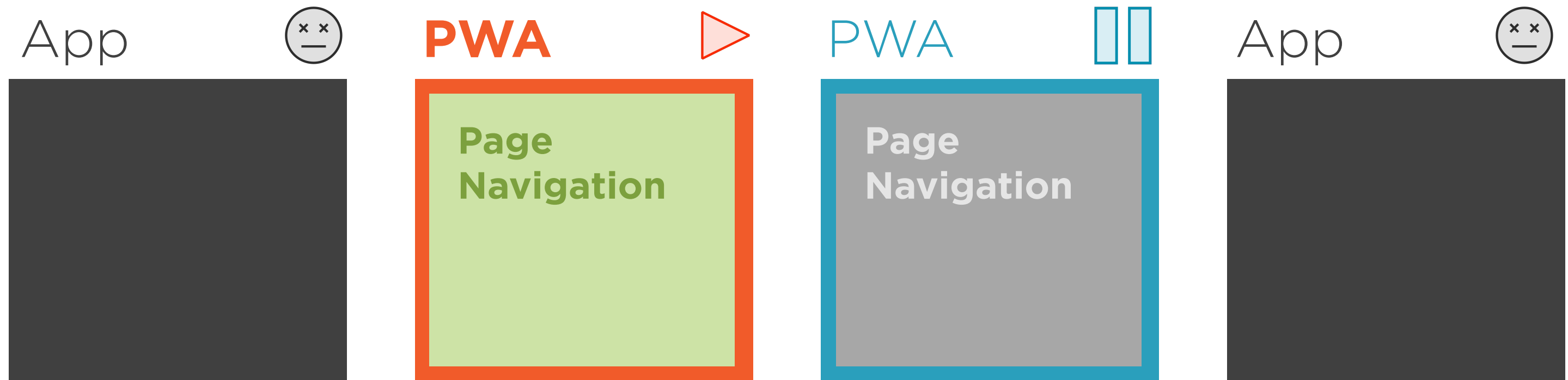
---

Mobile Operating System



---

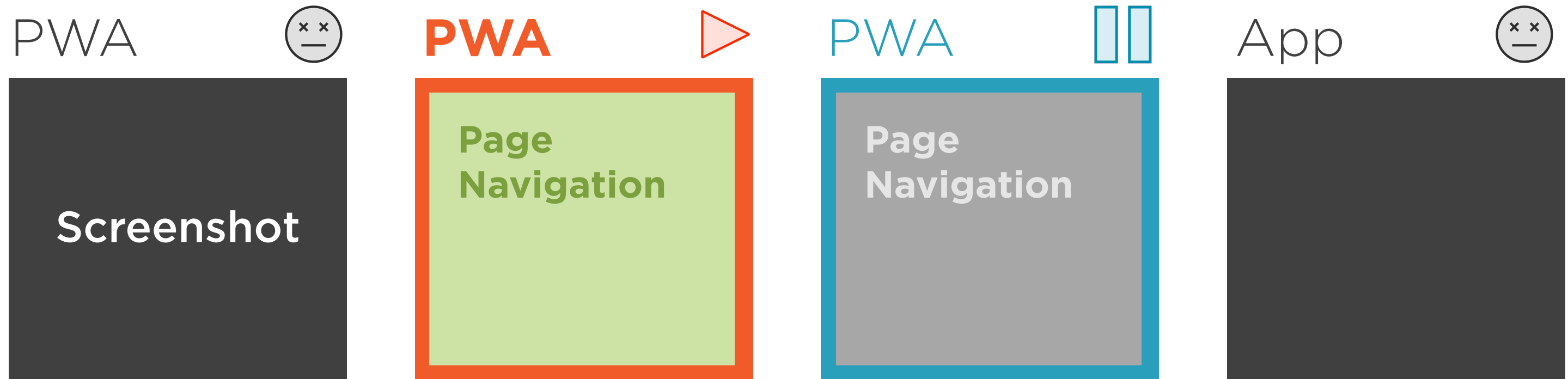
Mobile Operating System



---

Mobile Operating System

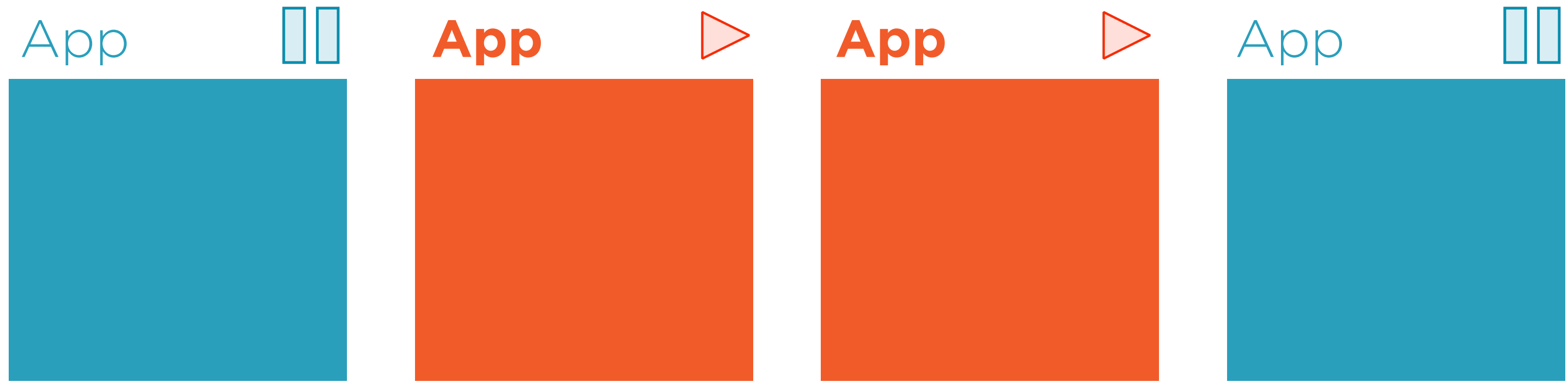




Mobile Operating System

---

# Mobile Operating System - Split Views



Mobile Operating System - Split Views

# Desktop

**Windows, macOS, Linux, Chrome OS**

**All PWAs opened (visible or not):**

- Are in memory

- Have execution rights

**PWAs not visible or in focus:**

- OS may limit timers

- OS may suspend them on rare situations

# Mobile

**Android, iOS, iPadOS**

**All PWAs visible on the screen**

Are in memory

Have execution rights

**PWAs in the background**

Could have their Page Navigations  
in memory or not

Don't have execution rights  
(exceptions apply)

Mobile devices don't let us  
execute any code in the  
background; and even on  
desktop, only opened PWAs  
have execution rights

The Service Worker can  
have some execution rights  
while the PWA is not active

# Service Worker Background Abilities

## APIs on top of the main spec

**Most of these APIs are available only on Chromium-based browsers**

- Web Push
- Background Sync
- Periodic Background Sync
- Background Fetch

Some of them requires user's permission







# Use Page Visibility and Page Lifecycle APIs



---

If you switch  
from a PWA to  
another App,  
Desktop or  
Home Screen

## Desktop

 ↔  PWA still in memory and running  
⚡ visibilitychange  
⚡ blur

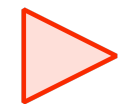


## Mobile and some desktop situations

 →  PWA goes to a suspended state  
⚡ visibilitychange  
⚡ freeze (Chromium only)




 →  At any time, it can be discarded

If you go back  
to the PWA  
using task  
manager or  
gesture

## Desktop

   Makes visible the PWA  
⚡ `visibilitychange`  
⚡ `focus`

## Mobile and some desktop situations

   Resuming a suspended state  
⚡ `visibilitychange`  
⚡ `resume` (Chromium only)

   PWA was discarded from memory  
⚡ `load`

`document.wasDiscarded == true`  
(Chromium only)

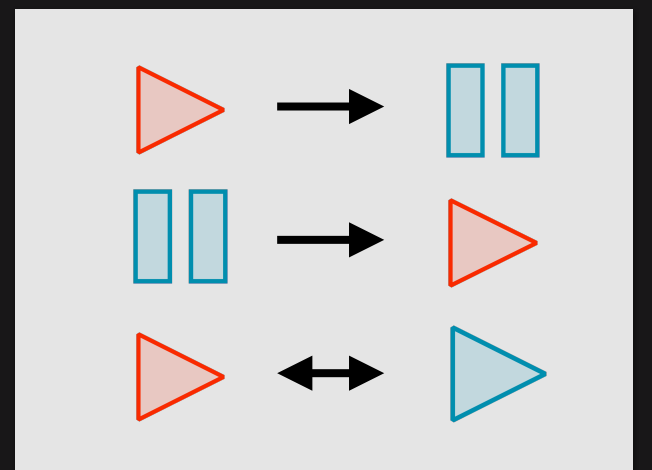
On desktop, PWAs work like  
browser's tabs

On mobile if you swipe out a PWA from the multi task manager, the page navigation will disappear in case it was still in memory

# Visibility Change Detection

All platforms

script.js



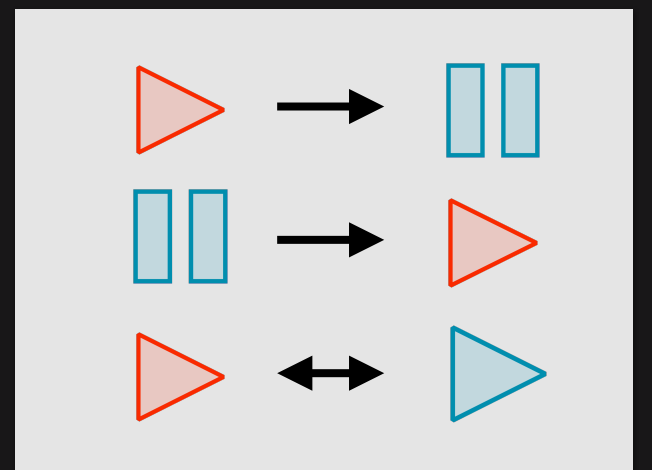
# Visibility Change Detection

All platforms

script.js

```
window.addEventListener('visibilitychange', event => {
```

```
});
```



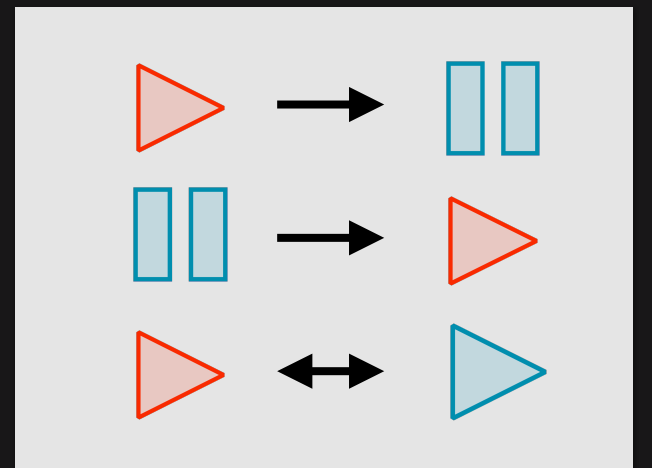


# Visibility Change Detection

All platforms

script.js

```
window.addEventListener('visibilitychange', event => {  
  if (document.visibilityState === 'hidden') {  
    track('lifecycle', 'hide');  
    // on some devices, last chance to save current state  
  } else {  
    track('lifecycle', 'show');  
  }  
});
```



# App State Change Detection

Some chromium browsers

script.js



# App State Change Detection

Some chromium browsers

script.js

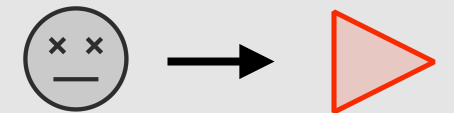
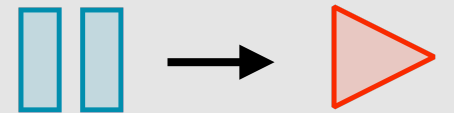
```
window.addEventListener('freeze', event => {  
  track('lifecycle', 'hide');  
  // we save current app's state (custom code)  
  saveState();  
});
```



# App State Change Detection

Some chromium browsers

script.js

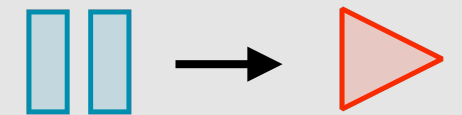


# App State Change Detection

Some chromium browsers

script.js

```
window.addEventListener('resume', event => {  
  track('lifecycle', 'resume'); // No need to restore  
});
```

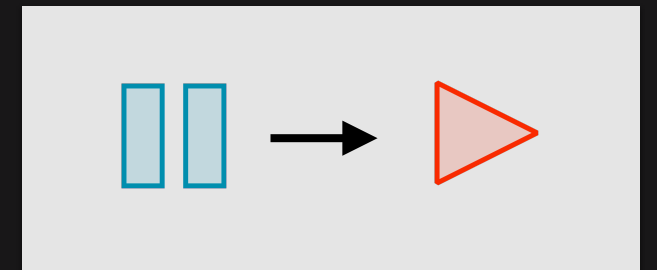


# App State Change Detection

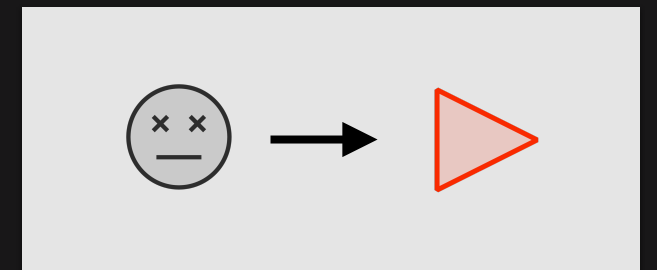
Some chromium browsers

script.js

```
window.addEventListener('resume', event => {  
  track('lifecycle', 'resume'); // No need to restore  
});
```



```
window.addEventListener('DOMContentLoaded', event => {  
  if (document.wasDiscarded) {  
    track('lifecycle', 'resume after discarded');  
    restoreState(); // our custom code  
  }  
});
```



On Multi Page Applications,  
each navigation should  
register events; changing  
page will also trigger similar  
events

# Freeze and Resume on mobile

**These events might not be available**

**You can save state**

On visibilitychange

Include a timestamp

**When should you restore state?**

When the page loads

Using the saved timestamp you can decide if you want to restore state or start a new navigation



To know if you are not visible, but still running, you can set a timer in `visibilitychange` and check how much time has passed when executed

## Conclusion

**On desktop, PWA lifecycle is similar to a tab**

**On mobile, a PWA in the background is:**

- **Typically Suspended**
- **Sometimes discarded from memory**

**Think about timers, pending requests and other situations when navigation resumes**

**For all browsers: Page Visibility API**

**For Chromium: Page Lifecycle API**



# MODULE 5-----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 goes into the same quota**

**Defined per origin**

**Quota is 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**
  - With user intervention

# iOS and iPadOS

## **With Safari, Best Effort**

*Eviction can happen:*

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

## **With Installed PWA, Persistent 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

## **Persistent 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, so using `await` remember to wrap those calls in an `async` function



# Track Usage for Analytics

---

# Measuring Impact for PWA Usage

**PWA loads from the launcher icon**

**Moving from browser's tab to standalone  
and viceversa**

Not every metric available on every platform



# Tracking App Launcher Usage

Different techniques are available to track when the user is accessing our PWA from the Launcher icon and not from the browser

# PWA Launcher Icon Detection

Option 1: Check media query matching manifest's display value

script.js

# PWA Launcher Icon Detection

Option 1: Check media query matching manifest's display value

script.js

```
window.addEventListener('DOMContentLoaded', event => {  
  if (window.matchMedia('(display-mode: standalone)').matches) {  
    event => track('pwaload', 'standalone')  
  }  
});
```



# PWA Launcher Icon Detection

Option 1: Check media query matching manifest's display value

script.js

```
window.addEventListener('DOMContentLoaded', event => {  
  if (window.matchMedia('(display-mode: standalone)').matches) {  
    event => track('pwaload', 'standalone')  
  }  
});
```

# PWA Launcher Icon Detection

Option 1: Check media query matching manifest's display value

script.js

```
window.addEventListener('DOMContentLoaded', event => {  
  if (window.matchMedia('(display-mode: minimal-ui)').matches) {  
    event => track('pwaload', 'minimal-ui')  
  }  
});
```

# PWA Launcher Icon Detection

Option 1: Check media query matching manifest's display value

script.js

```
window.addEventListener('DOMContentLoaded', event => {  
  if (window.matchMedia('(display-mode: fullscreen)').matches) {  
    event => track('pwaload', 'fullscreen')  
  }  
});
```

# PWA Launcher Icon Detection

Option 1: Check media query matching manifest's display value

script.js

```
window.addEventListener('DOMContentLoaded', event => {  
  if (window.matchMedia('(display-mode: browser)').matches) {  
    event => track('pwaload', 'browser')  
  }  
});
```

# PWA Launcher Icon Detection

Option 2: Specify a tracking argument in the manifest's start\_url

**app.webmanifest**

```
{  
  "name": "PWA Name",  
  "start_url": "/",  
  "theme_color": "#385770",  
  "display": "standalone",  
  "scope": "/",  
}
```

# PWA Launcher Icon Detection

Option 2: Specify a tracking argument in the manifest's start\_url

**app.webmanifest**

# PWA Launcher Icon Detection

Option 2: Specify a tracking argument in the manifest's start\_url

app.webmanifest

```
{  
  "name": "PWA Name",  
  "start_url": " /?utm_source=standalone",  
  "theme_color": "#385770",  
  "display": "standalone",  
  "scope": " /",  
}
```

There are use cases, like  
share target, a shortcut or  
deep links that can start a  
standalone navigation not  
coming from the launcher  
icon



# App Store Distribution - Add Tracking Argument

Trusted Web  
Activity (TWA)

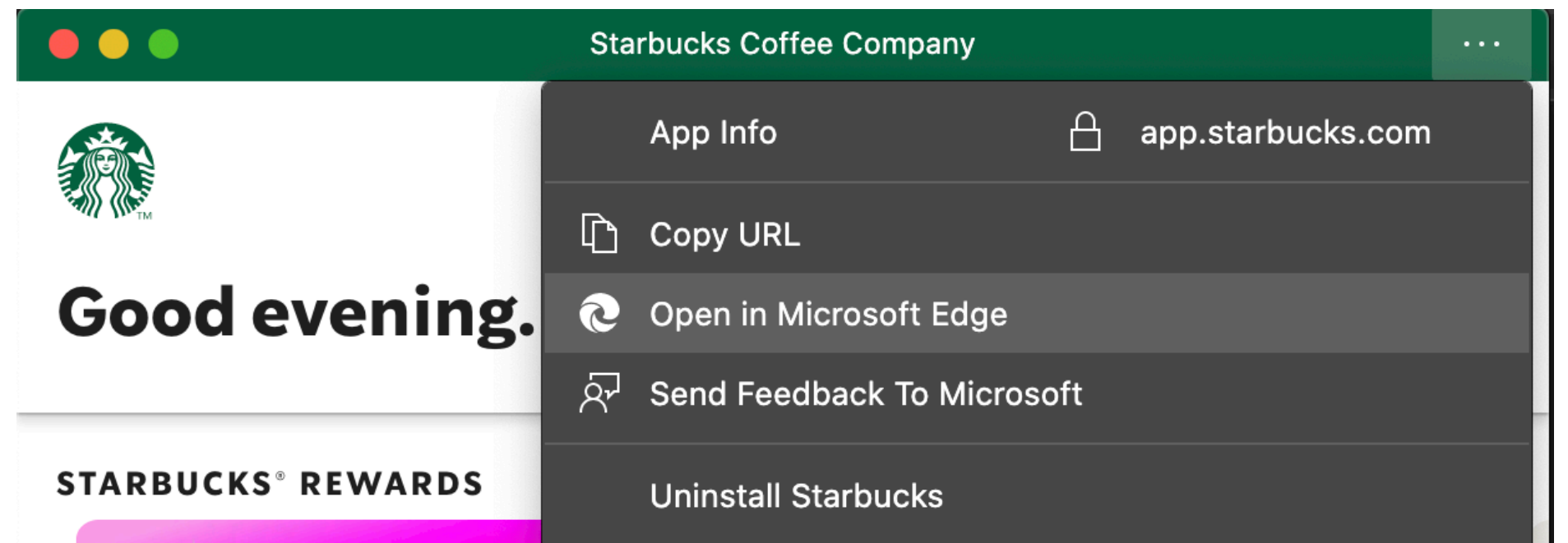
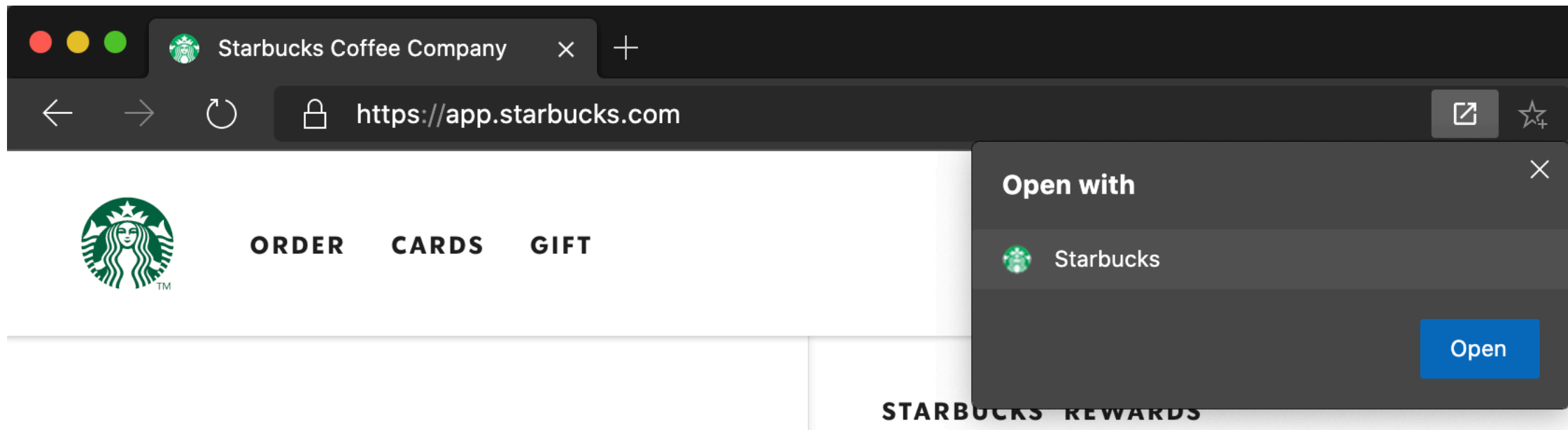
Web View

URL submitted



# Tracking Display Mode Transfer

On desktop operating systems, you can transfer  
current navigation from the browser's tab to/  
from the standalone PWA experience



# PWA Display Mode Change Tracking

Useful only on desktop when navigation moves between modes

script.js

# PWA Display Mode Change Tracking

Useful only on desktop when navigation moves between modes

script.js

```
window.addEventListener('DOMContentLoaded', () => {  
  window.matchMedia('(display-mode: standalone)')  
  
});
```

# PWA Display Mode Change Tracking

Useful only on desktop when navigation moves between modes

script.js

```
window.addEventListener('DOMContentLoaded', () => {  
  window.matchMedia('(display-mode: standalone)')  
    .addListener(event => {  
  
    });  
});
```

# PWA Display Mode Change Tracking

Useful only on desktop when navigation moves between modes

script.js

```
window.addEventListener('DOMContentLoaded', () => {  
  window.matchMedia('(display-mode: standalone)')  
    .addListener(event => {  
      if (event.matches) {  
        track('pwa-mode-change', 'standalone');  
      }  
    });  
});
```

# PWA Display Mode Change Tracking

Useful only on desktop when navigation moves between modes

script.js

```
window.addEventListener('DOMContentLoaded', () => {  
  window.matchMedia('(display-mode: standalone)')  
    .addListener(event => {  
    if (event.matches) {  
      track('pwa-mode-change', 'standalone');  
    } else {  
      track('pwa-mode-change', 'browser');  
    }  
  });  
});
```



# PWA Display Mode Change Tracking

Useful only on desktop when navigation moves between modes

script.js

```
window.addEventListener('DOMContentLoaded', () => {  
  window.matchMedia('(display-mode: minimal-ui)')  
    .addListener(event => {  
      if (event.matches) {  
        track('pwa-mode-change', 'minimal-ui');  
      } else {  
        track('pwa-mode-change', 'browser');  
      }  
    });  
});
```

On Android, navigation transfer between browser and PWA context is not available; new navigations are triggered

On iOS and iPadOS all  
navigations in the PWA  
context will be on **start\_url**  
Safari doesn't transfer  
navigation

# Summary

## Tracking Usage

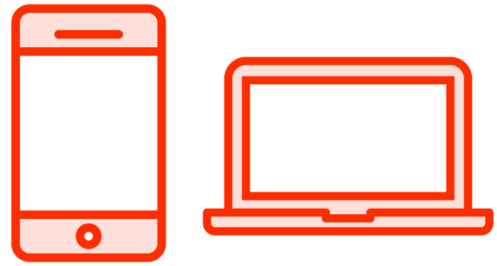
- Launching: using display mode media query or URL arguments
- Display Mode transfer (desktop)
- More events on different APIs



# Improve Performance with Navigation Preload

---

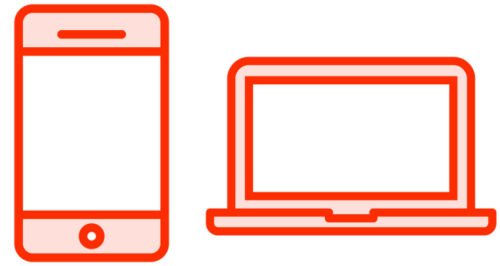
# Starting a PWA without Navigation Preload



**Web server**



# The Web Runtime Starts



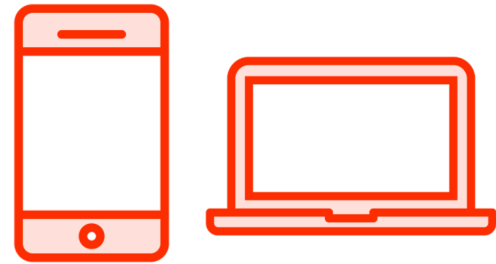
**Web server**



**Web  
runtime**



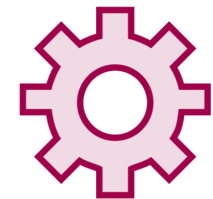
# The Service Worker Starts



**Web server**

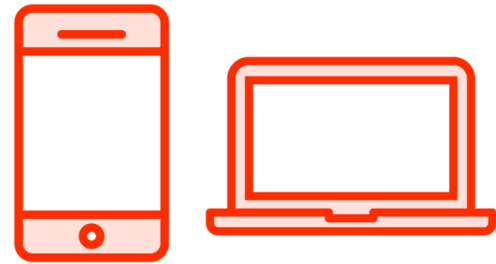


**Web  
runtime**



**Service  
worker**

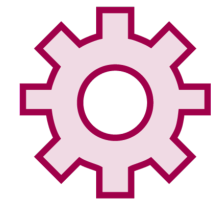
# The Service Workers check cache storage



**Web server**



**Web  
runtime**

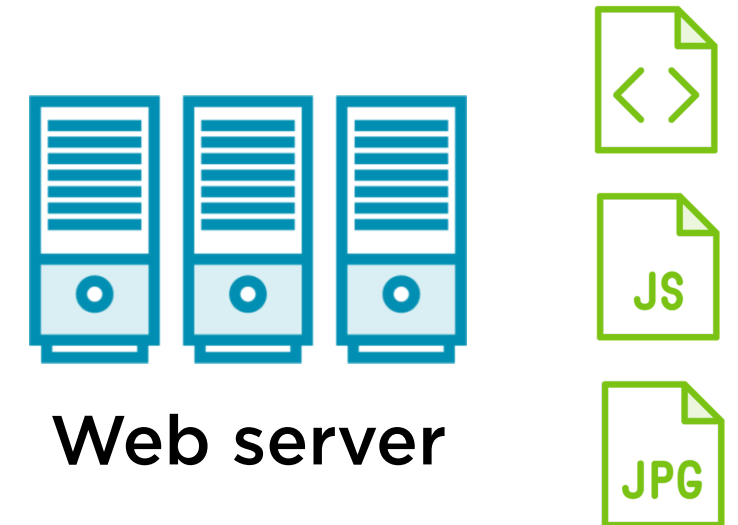
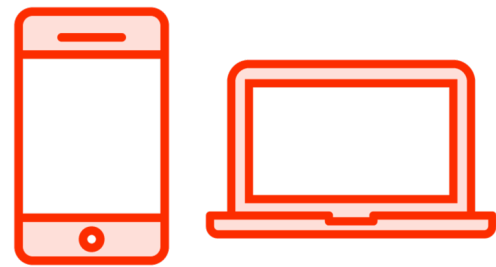


**Service  
worker**

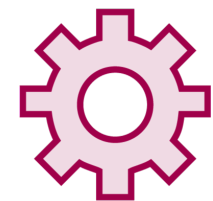


**Page**

# If the page is in cache, SW serves it from there



Web  
runtime



Service  
worker

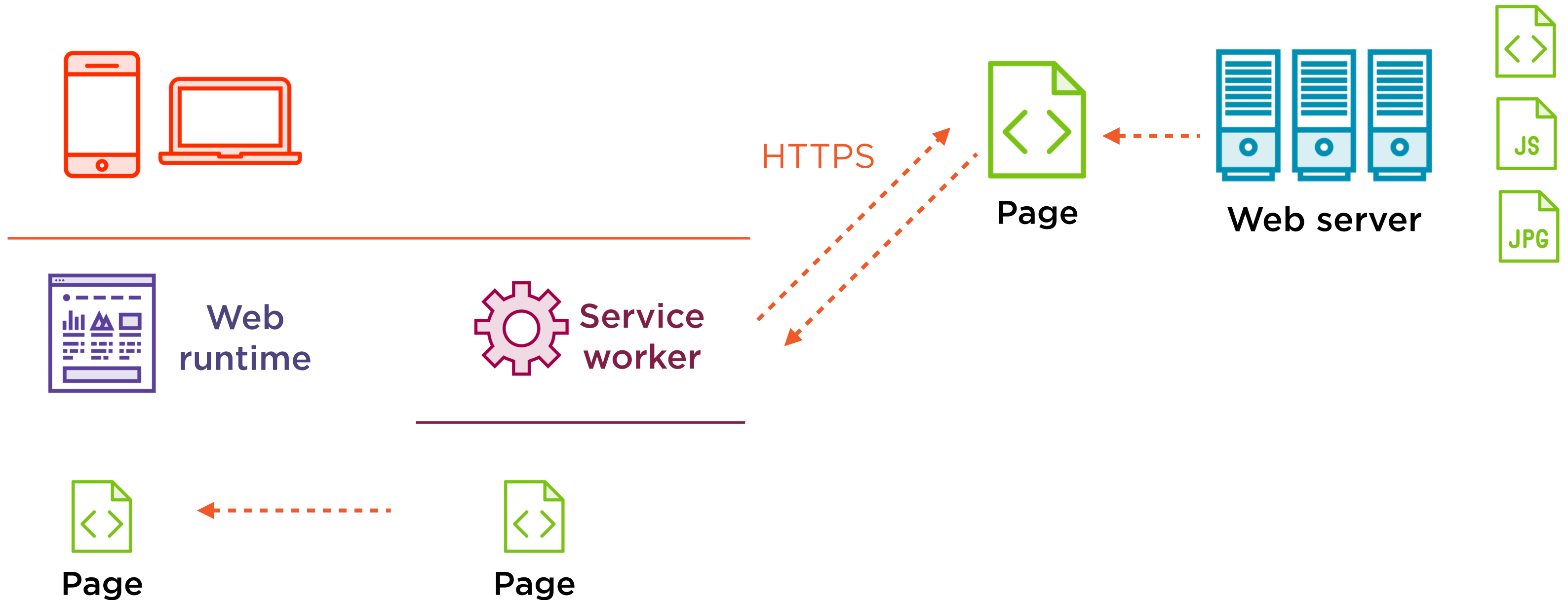


Page



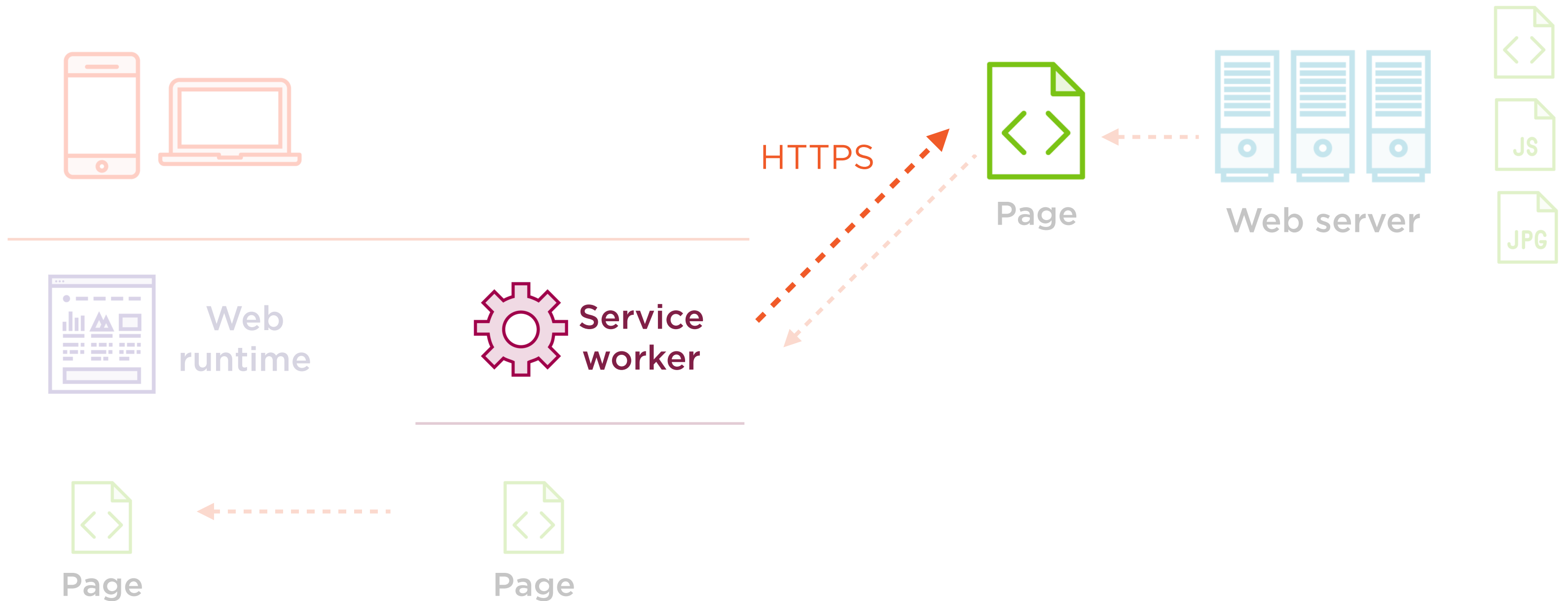
Page

# If the page is not in Cache, it starts a Request



What's the problem?  
On some slow CPU devices,  
all the process can take up  
to 500ms

# Delay from PWA launch: +20ms to +500ms



If the page is not in the  
cache or we are not  
expecting it to be we have a  
performance penalty to use  
a service worker

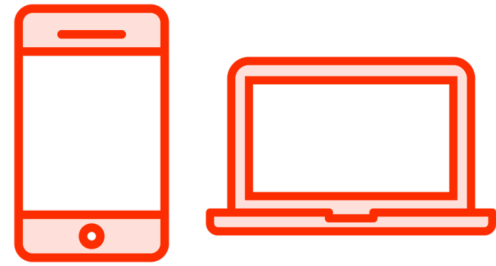


# Navigation Preload

It allows the web rendering engine to start the initial navigation request preventively while starting the service worker at the same time



# Start a PWA with Navigation Preload

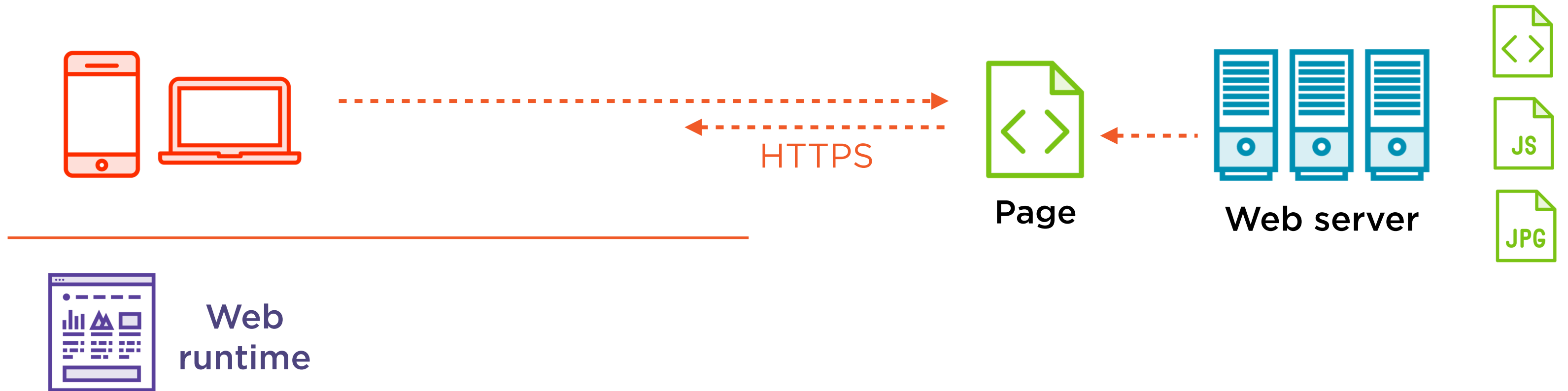


Web server

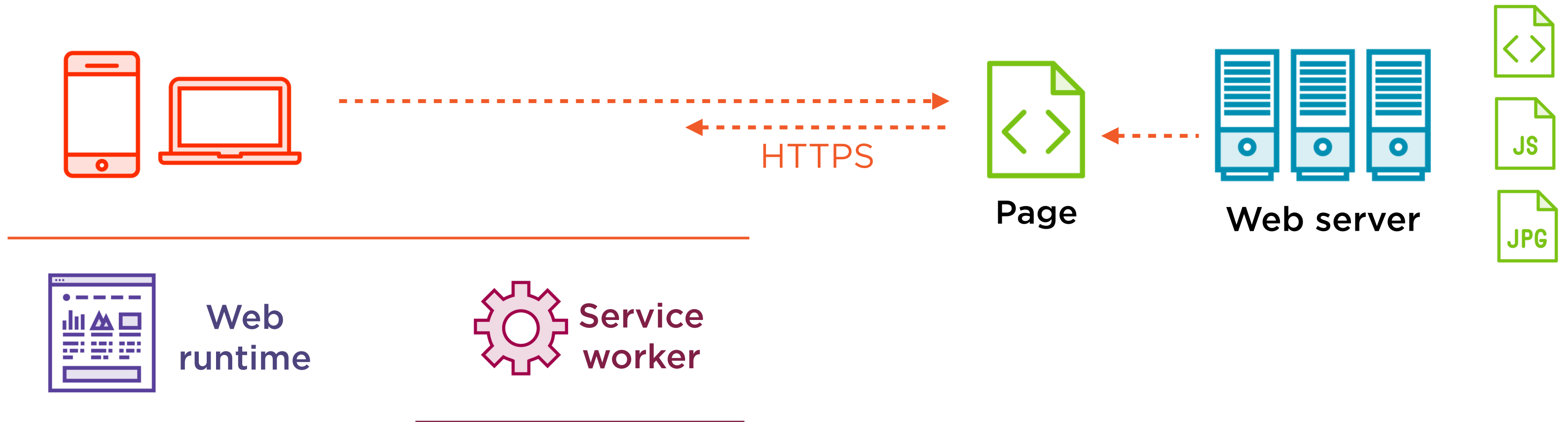


Web  
runtime

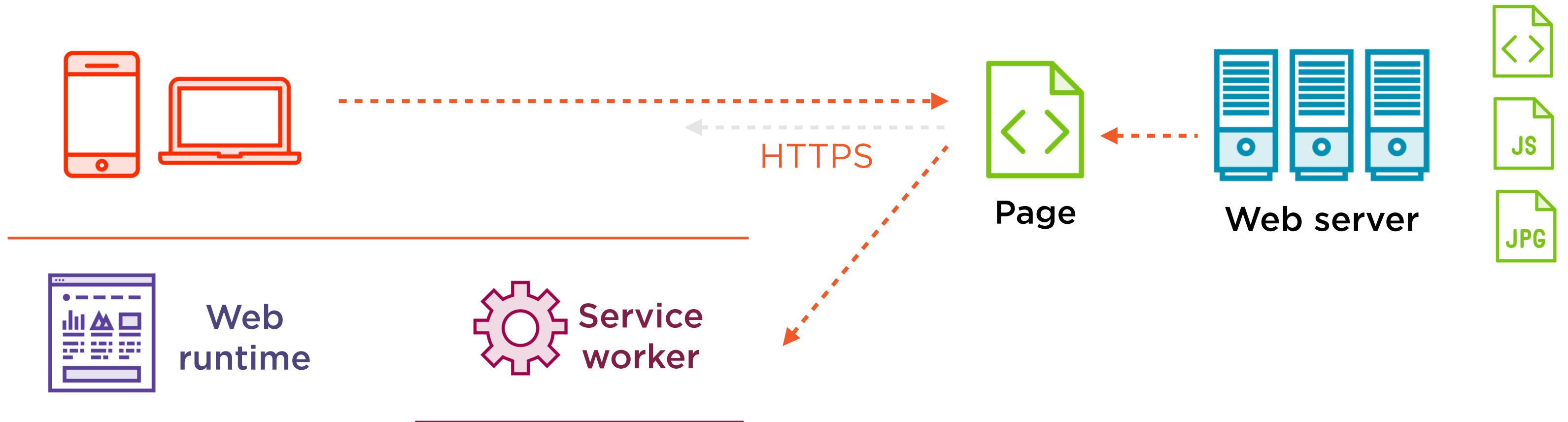
# The Web Rendering Engine Requests the Page



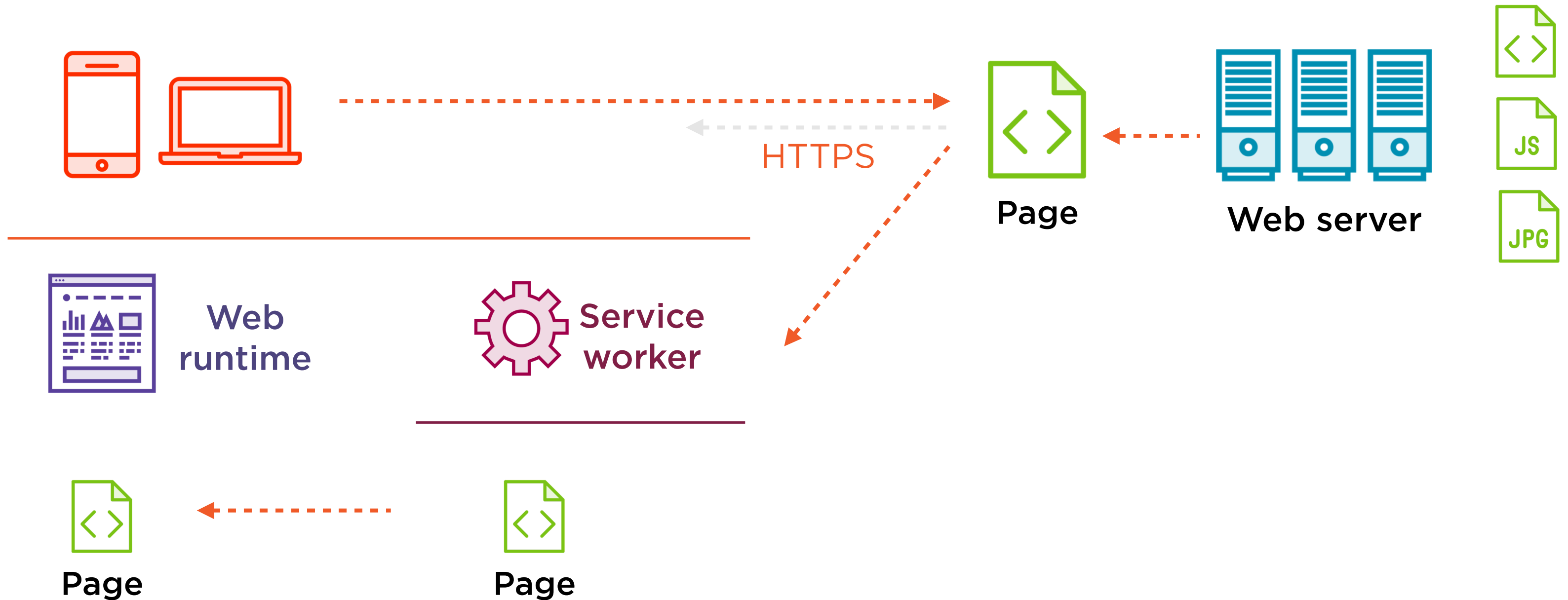
# At the Same Time, It Starts the Service Worker



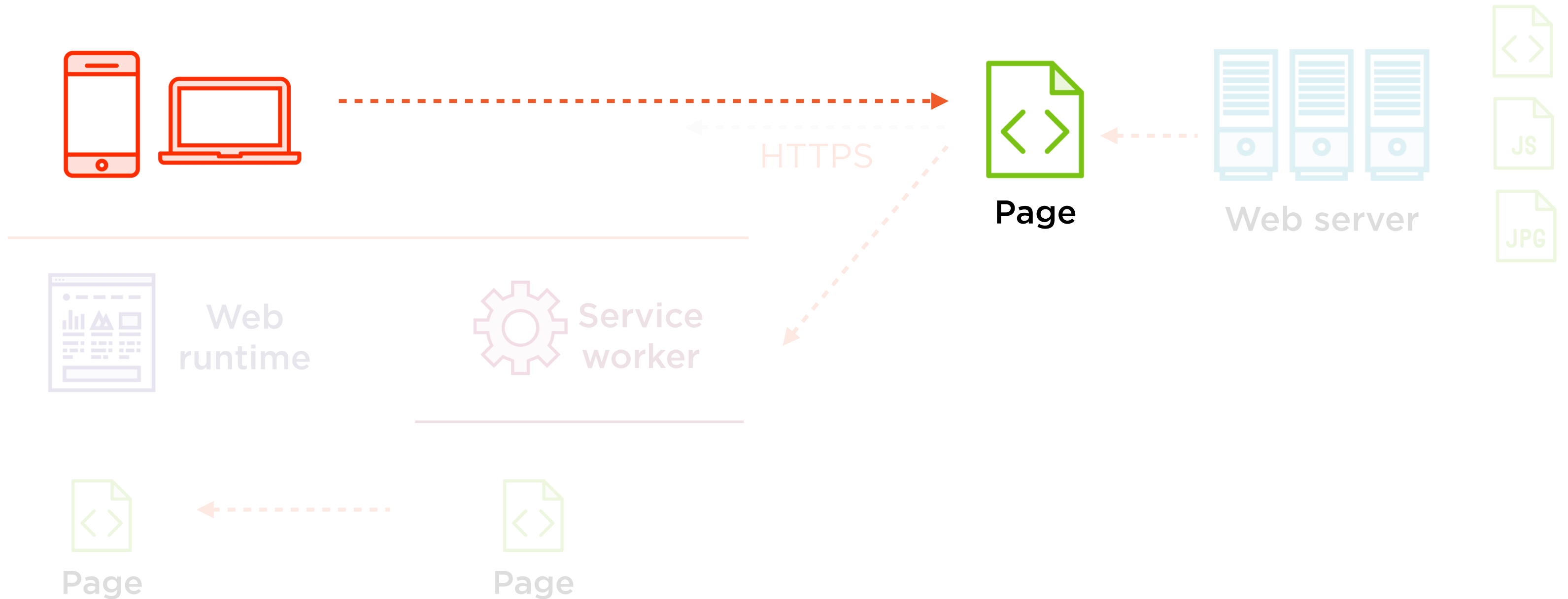
# The Service Worker Reuses the Request



# If the page is not in Cache, it starts a Request



# No Delay from PWA launch



# Step 1: Enable Navigation Preload

Available only on some browsers

**serviceworker.js**





# Step 1: Enable Navigation Preload

Available only on some browsers

serviceworker.js

```
self.addEventListener('activate', event => {  
  event.waitUntil(async function() {  
    if (self.registration.navigationPreload) {  
  
    }  
  })();  
});
```

# Step 1: Enable Navigation Preload

Available only on some browsers

serviceworker.js

```
self.addEventListener('activate', event => {  
  event.waitUntil(async function() {  
    if (self.registration.navigationPreload) {  
      // Enable navigation preload  
      await self.registration.navigationPreload.enable();  
    }  
  })();  
});
```

# Step 2: Use the Preloaded Response if Available

Available only on some browsers

**serviceworker.js**

# Step 2: Use the Preloaded Response if Available

Available only on some browsers

serviceworker.js

```
self.addEventListener('fetch', event => {  
  event.respondWith(async function() {  
  
    }());  
});
```

# Step 2: Use the Preloaded Response if Available

Available only on some browsers

serviceworker.js

```
self.addEventListener('fetch', event => {  
  event.respondWith(async function() {  
    const responseInCache = await caches.match(event.request);  
    if (responseInCache) return responseInCache;  
  
  })();  
});
```

# Step 2: Use the Preloaded Response if Available

Available only on some browsers

serviceworker.js

```
self.addEventListener('fetch', event => {  
  event.respondWith(async function() {  
    const responseInCache = await caches.match(event.request);  
    if (responseInCache) return responseInCache;  
  
    return fetch(event.request);  
  })();  
});
```

# Step 2: Use the Preloaded Response if Available

Available only on some browsers

serviceworker.js

```
self.addEventListener('fetch', event => {  
  event.respondWith(async function() {  
    const responseInCache = await caches.match(event.request);  
    if (responseInCache) return responseInCache;  
    const responseNavPreload = await event.preloadResponse;  
    if (responseNavPreload) return responseNavPreload;  
    return fetch(event.request);  
  })();  
});
```

If you enable Navigation  
Preload, you must use the  
response from it; if not, we  
might end up duplicating  
requests



The request goes with a header in case you want to return something different from the server

`Service-Worker-Navigation-Preload: true`



# Notify the User in the Background with Push

---



# Web Push

With user permission, the web app can send messages from the server not matter if the PWA is active or not

# Web Push Notifications

**PWA asks permission to send notifications to the user**

**If granted, now the service worker can create a notification from the background**

**Also, the PWA can subscribe the user to Push**

**A 'push' event will be fired in the Service Worker to handle a push**

**Firefox, Chrome, Edge, Samsung Internet  
Safari?**

Safari on macOS supports  
notifications with its own  
non-standard API

# Step 1: Ask for notification permissions

Don't do this when the page loads. Follow design patterns.

script.js

# Step 1: Ask for notification permissions

Don't do this when the page loads. Follow design patterns.

script.js

```
if ('Notification' in window) {  
  if (Notification.permission === "granted") {  
    track('notification', "granted");  
  }  
  
}
```



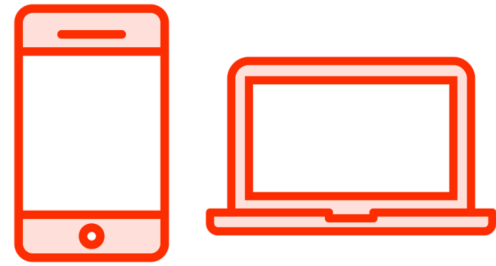
# Step 1: Ask for notification permissions

Don't do this when the page loads. Follow design patterns.

script.js

```
if ('Notification' in window) {  
  if (Notification.permission === "granted") {  
    track('notification', "granted");  
  }  
  status = await Notification.requestPermission();  
  // it can be 'granted', 'denied', 'default'  
  track('notification-request', status);  
}
```

# Web Push Subscription Architecture



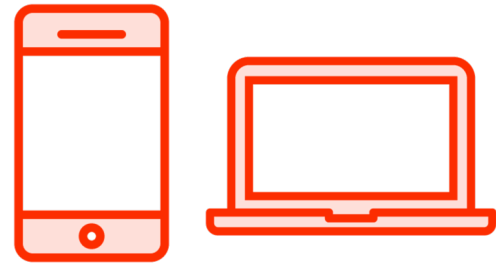
**Web server**



**Push server**

owned by browser

# Web Push Subscription Architecture



**Web server**



**Service  
worker**

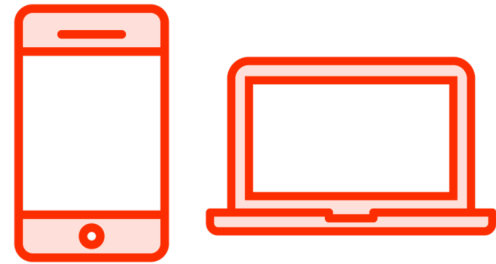


**Web  
runtime**



**Push server**  
owned by browser

# Web Push Subscription Architecture



**Web server**



**Web  
runtime**

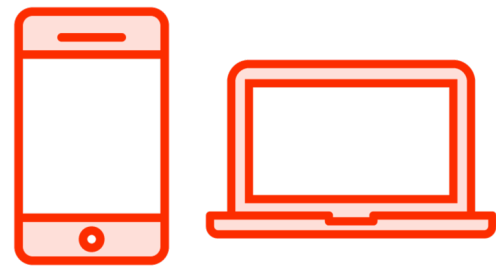
**Push Request**



**Push server**

owned by browser

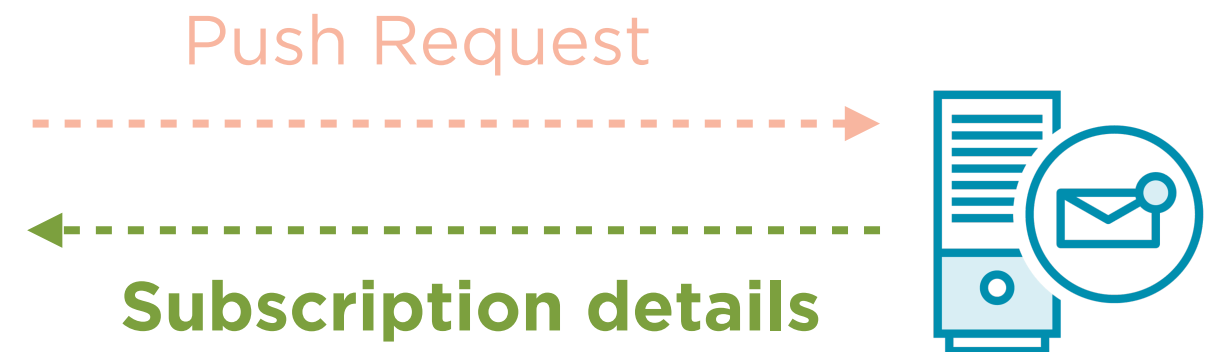
# Web Push Subscription Architecture



**Web server**

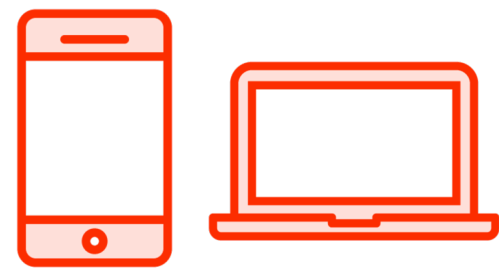


**Web  
runtime**



**Push server**  
owned by browser

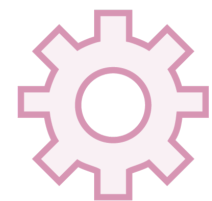
# Web Push Subscription Architecture



Subscription  
details



Web server



Service  
worker



Web  
runtime

Push Request



Subscription details



Push server

owned by browser

# Step 2: Register a Push Subscription

script.js

## Step 2: Register a Push Subscription

Check course's source code for instructions for get a key

script.js

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



## Step 2: Register a Push Subscription

Check course's source code for instructions for get a key

script.js

```
if ('PushManager' in window) {  
  const registration = await navigator.serviceWorker.ready;  
  const pushData = await registration.pushManager  
    .subscribe(  
      { userVisibleOnly: true, applicationServerKey: 'KEY' }  
    );  
  
}
```

## Step 2: Register a Push Subscription

Check course's source code for instructions for get a key

script.js

```
if ('PushManager' in window) {  
  const registration = await navigator.serviceWorker.ready;  
  const pushData = await registration.pushManager  
    .subscribe(  
      { userVisibleOnly: true, applicationServerKey: 'KEY' }  
    );  
  if (pushData) {  
    // We have the Push Subscription details to save in our server  
  }  
}
```

```
{
  "endpoint": "https://pushserver.com/unique-id",
  "keys": {
    "p256dh" :
      "BNcRdreALRFXTk00UHK1EtK2wtaz5Ry4YfYCA_0QTpQtUbV1U1s0VJXg7A8u-
      Ts1XbjhazAkj7I99e8QcYP7DkM=",
    "auth"    : "tBHItJI5svbpez7KI4CCXg=="
  }
}
```

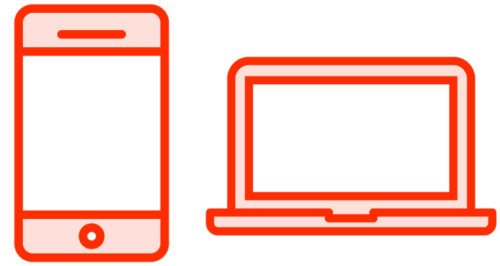
## Subscription details from the Push server

It includes an end-point, a unique ID for that user in that browser and public keys we have to use later to encrypt messages to that user

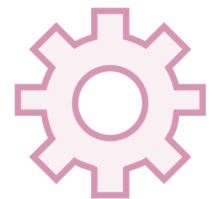
We need to store this data safely in our web server and assign it to the user

If you want to send a  
personalized message to  
each user you need to assign  
these details to the right  
user

# Web Push Message Architecture



**Web server**



**Service  
worker**



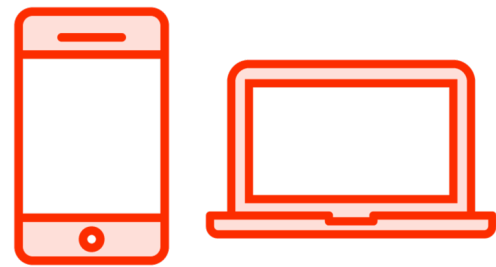
**Web  
runtime**



**Push server**

owned by browser

# Web Push Message Architecture



**Web server**



**Service  
worker**



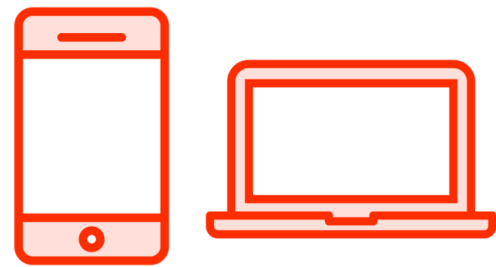
**Web  
runtime**



**Push server**

owned by browser

# Web Push Message Architecture



**Web server**



**Service  
worker**



**Web  
runtime**

**Push Message**



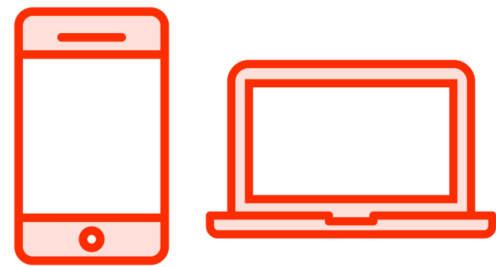
Subscription  
details + Payload



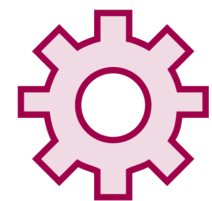
**Push server**

owned by browser

# Web Push Message Architecture



**Web server**



**Service  
worker**



**Web  
runtime**

Push Message

Subscription  
details + Payload

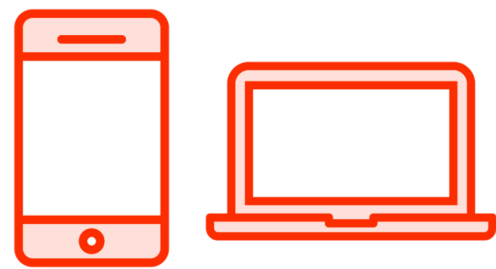


**Push server**

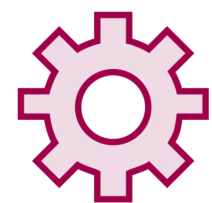
owned by browser



# Web Push Message Architecture



**Web server**



**Service  
worker**



**Web  
runtime**

Push Message

Subscription  
details + Payload



**Push server**

owned by browser



**Notification**

# Step 3: Receive the push and create notification

**serviceworker.js**



## Step 3: Receive the push and create notification

serviceworker.js

```
self.addEventListener('push', event => {  
  event.waitUntil(  
    self.registration.showNotification('Title',  
      {  
        body: event.data.text(),  
        icon: "/icon.png"  
      })  
  )  
});
```

Push messages can arrive  
when a PWA is active or  
when it's closed

You are not forced to notify  
the user with the payload's  
body, but you are forced to  
notify something



# Background Execution: Sync, Fetch and Periodic Sync

---





# Background Sync

Without user permission, the web app can mark sync operations to do in a background thread, even if the user closes the PWA

# Background Sync

**PWA defers a sync action until the device has a stable connection to the server**

**A 'sync' event will be fired in the Service Worker to handle a pending sync:**

- Immediately if network is stable
- Later, when network go back to stable
- If battery is in good level

**We access the network and fulfill the sync or leave it pending**

**Execution happens in the background**

Remember execution  
happens in the background;  
maybe no PWA's page is  
currently loaded.  
Use Web Notifications for  
messaging the user

# Step 1: Register a Background Sync Action

Available from the window or service worker's contexts

**script.js**

# Step 1: Register a Background Sync Action

Available from the window or service worker's contexts

script.js

```
if ('SyncManager' in window) {  
  const registration = await navigator.serviceWorker.ready;  
  registration.sync.register("tag-name");  
}
```

# Step 2: Handle the Sync Action

This happens in the service worker's context at any moment

**serviceworker.js**

## Step 2: Handle the Sync Action

This happens in the service worker's context at any moment

serviceworker.js

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



# Periodic Background Sync

If the site engagement score is good, the web app can execute code periodically in the background



# Periodic Background Sync

**PWA asks user for permission to periodically execute code in the background**

**A 'periodicsync' event will be fired in the Service Worker:**

- On a synchronization time interval
- If battery and network conditions are met

**We typically access the network on each execution, but it's not mandatory**

**Execution happens in the background**

Execution frequency will be  
honored based on a Site  
Engagement Score defined  
by the browser

# Step 1: Ask for Periodic Sync Permission

Permission will be granted based on Site Engagement Score

script.js

# Step 1: Ask for Periodic Sync Permission

Permission will be granted based on Site Engagement Score

script.js

```
const permissionStatus = await navigator.permissions.query({  
  name: 'periodic-background-sync',  
});
```

# Step 1: Ask for Periodic Sync Permission

Permission will be granted based on Site Engagement Score

script.js

```
const permissionStatus = await navigator.permissions.query({
  name: 'periodic-background-sync',
});
if (permissionStatus.state === 'granted') {
  track('periodic-sync', 'granted');
} else {
  track('periodic-sync', 'denied');
}
```

# Step 2: Register a Background Sync Operation

It requires a tag and a synchronization interval in milliseconds

**script.js**

# Step 2: Register a Background Sync Operation

It requires a tag and a synchronization interval in milliseconds

script.js

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

await registration.periodicSync.register('sync-tag', {
  minInterval: 24 * 60 * 60 * 1000 // One day
});

}
```

# Step 2: Register a Background Sync Operation

It requires a tag and a synchronization interval in milliseconds

script.js

```
const registration = await navigator.serviceWorker.ready;
if ('periodicSync' in registration) {
  try {
    await registration.periodicSync.register('sync-tag', {
      minInterval: 24 * 60 * 60 * 1000 // One day
    });
  } catch (error) { }
}
```



# Step 3: Handle the Periodic Sync Operation

It will be executed in the Service Worker

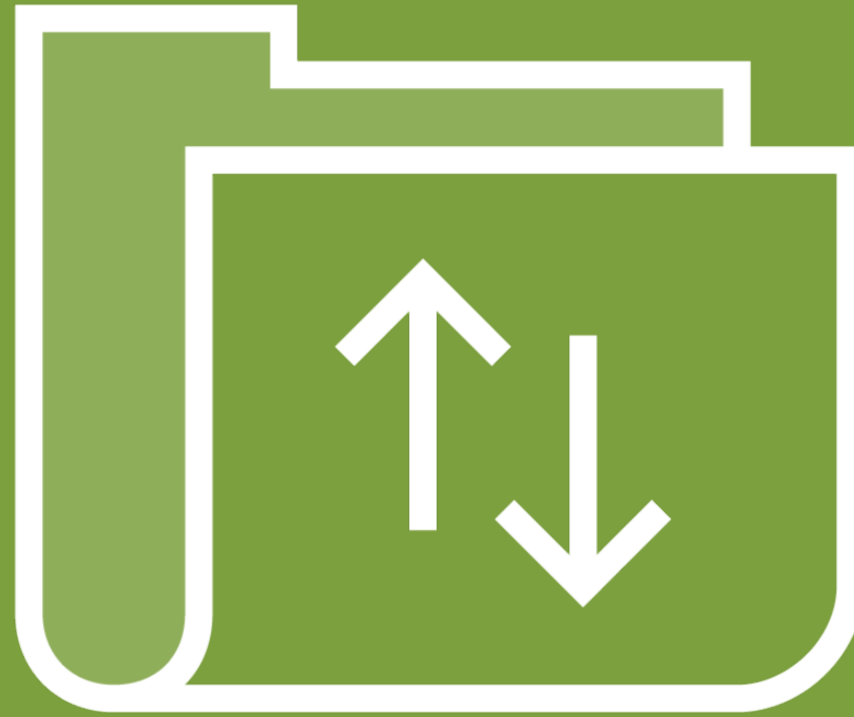
**serviceworker.js**

# Step 3: Handle the Periodic Sync Operation

It will be executed in the Service Worker

serviceworker.js

```
self.addEventListener('periodicsync', (event) => {  
  if (event.tag === 'sync-tag') {  
    event.waitUntil(doPeriodicSyncOperation());  
  }  
});
```



# Background Fetch

The web app can download files and content  
even when the user gets out of the PWA

# Background Fetch

**PWA asks the web engine to make some fetch download requests**

**The browser will download the requests in the background while showing an OS notification about the process**

**Events will be fired in the Service Worker when:**

- Download finishes
- If the user has clicked the notification
- On abort or failure
- On Progress

# Step 1: Start a Background Fetch Request

The operation will be handled entirely by the browser

**script.js**

# Step 1: Start a Background Fetch Request

The operation will be handled entirely by the browser

# script.js

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

```
const fetch = await registration.backgroundFetch.fetch(
```

$$);$$

# Step 1: Start a Background Fetch Request

The operation will be handled entirely by the browser

script.js

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

const fetch = await registration.backgroundFetch.fetch(
  'fetch-name', ['url-1', 'url-2', 'url-3'],

);
```

# Step 1: Start a Background Fetch Request

The operation will be handled entirely by the browser

script.js

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

const fetch = await registration.backgroundFetch.fetch(
  'fetch-name', ['url-1', 'url-2', 'url-3'],
  {
    title: 'Offline Content',
    icons: [{ sizes: '300x300', src: 'icon.png', type: 'image/png' }],
    downloadTotal: 20 * 1024 * 1024,
  }
);
```



# Step 1: Start a Background Fetch Request

The operation will be handled entirely by the browser

script.js

```
const registration = await navigator.serviceWorker.ready;
if ('backgroundFetch' in registration) {
  const fetch = await registration.backgroundFetch.fetch(
    'fetch-name', ['url-1', 'url-2', 'url-3'],
    {
      title: 'Offline Content',
      icons: [{ sizes: '300x300', src: 'icon.png', type: 'image/png' }],
      downloadTotal: 20 * 1024 * 1024,
    }
  );
}
```

# Step 2: Handle events in Service Worker

**serviceworker.js**

## Step 2: Handle events in Service Worker

serviceworker.js

```
self.addEventListener('backgroundfetchsuccess', async event => {  
  const downloadedFiles = await event.registration.matchAll();  
});
```

## Step 2: Handle events in Service Worker

serviceworker.js

```
self.addEventListener('backgroundfetchsuccess', async event => {  
  const downloadedFiles = await event.registration.matchAll();  
});
```

```
self.addEventListener('backgroundfetchclick', event => {  
  clients.openWindow('/download-status');  
});
```

## Step 2: Handle events in Service Worker

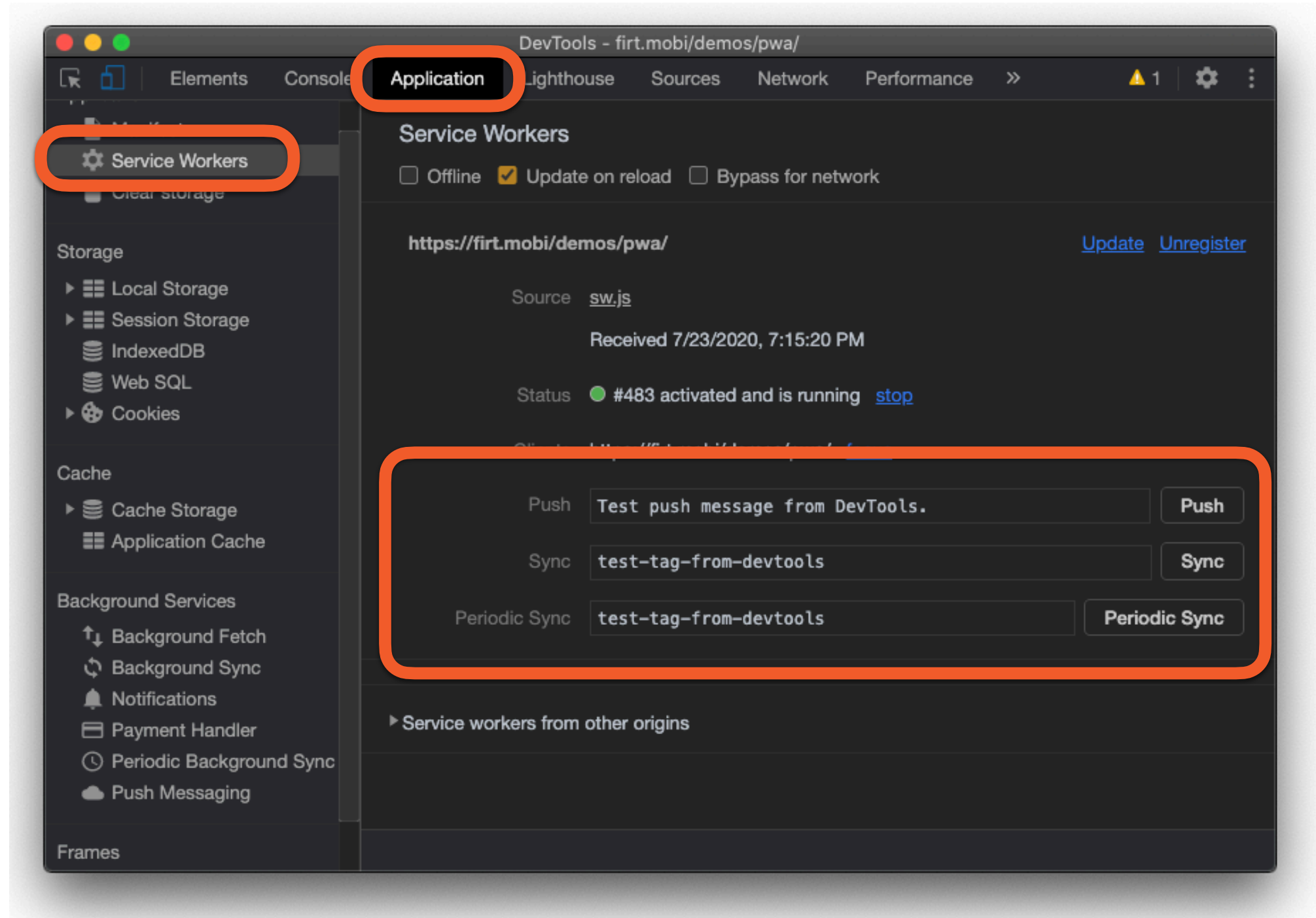
serviceworker.js

```
self.addEventListener('backgroundfetchsuccess', async event => {  
  const downloadedFiles = await event.registration.matchAll();  
});
```

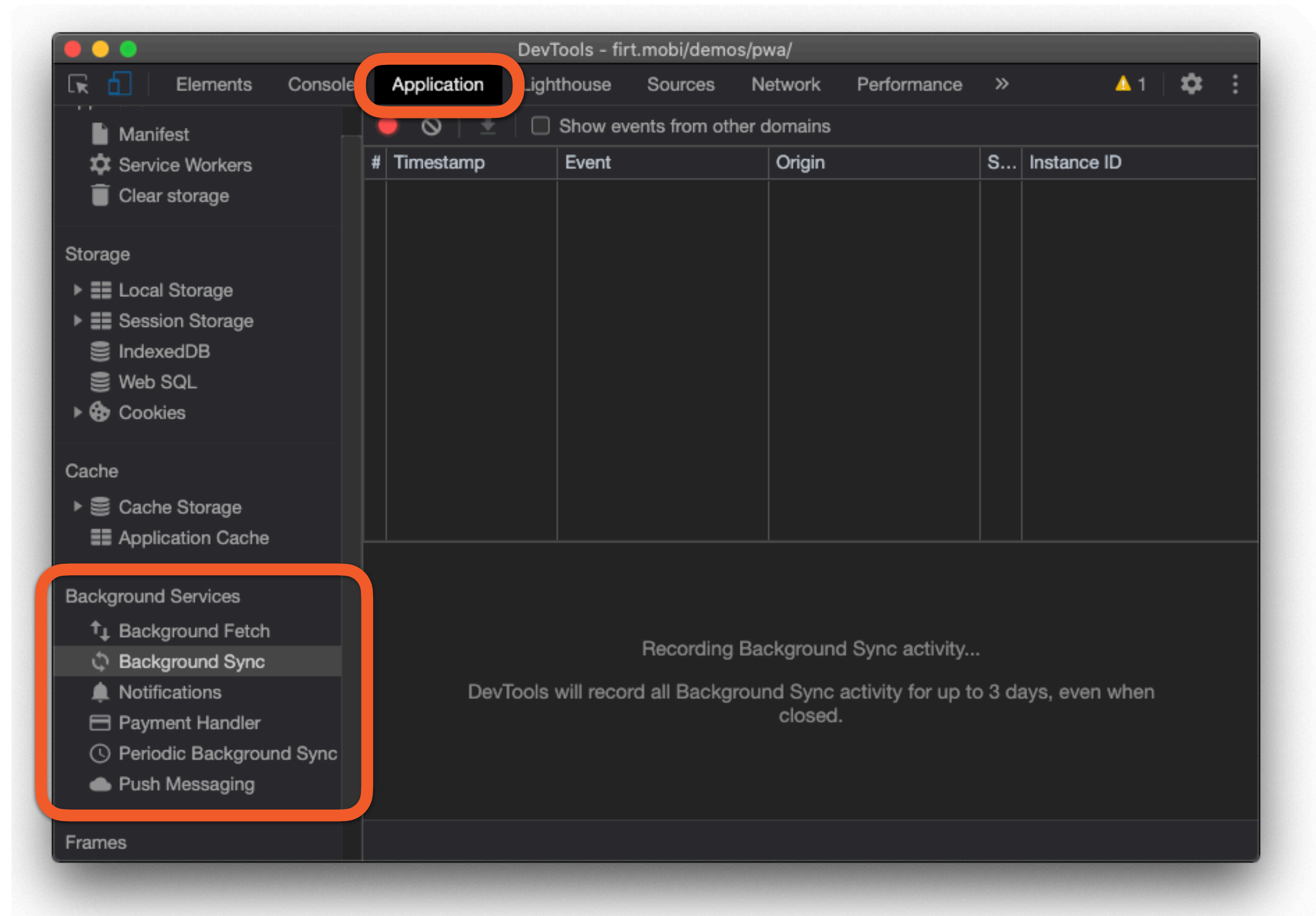
```
self.addEventListener('backgroundfetchclick', event => {  
  clients.openWindow('/download-status');  
});
```

```
self.addEventListener('backgroundfetchfailure', event => {  
});
```

# Debugging Background Execution



# Debugging Background Execution







# Summary

## Managing App's Lifecycle

- Understand the PWA Lifecycle
- Page Visibility and Page Lifecycle APIs
- Track Usage for Analytics
- Improve Performance with Navigation Preload
- Web Push Notifications
- Background Execution: Sync, Fetch and Periodic Sync

Up Next:  
Updating the Application

---