Progressive Web Apps

Some facts

- Around 90% people on the world use Native Apps, only 10% use Web Apps
- Around 85% of the time spent in Native Apps is in user's TOP 3 apps (like YouTube, Facebook, Messanger)
- Avarage of new Native Apps installed per month by users is close to 0
- Web Apps have 3 Times bigger audience reach than Native Apps (because of Native Apps user's TOP 3)

Conclusion? Web Apps have much better audience reach but they lack all the cool features that provides Native Apps.

What is Progressive Web App?

Progressively enhance web apps to look and feel like native apps:

- Accessing device camera or location
- Fully working offline mode
- Displaying push notifications
- Installing app on your devices and adding home screen icon and splash screen
- Synchronizing data in the background

Progressive Web App should be:

- Reliable: load fast and provide offline functionality
- Fast: respond quickly to user actions
- Engaging: feel like a native app on mobile device

PWA Compability:

https://www.goodbarber.com/blog/progressive-web-apps-browser-support-compatibility-a883/

Differences between PWA and Native App

- Installation
- Cross platform availability
- Discovery
- Push notifications
- Security
- Device Features
- Cost

https://www.mobiloud.com/blog/progressive-web-apps-vs-native-apps/

Main requirements for the Web App to be PWA

- App served over https
- Has included manifest.json
- Has registered service worker script

PWA vs SPA

SPA:

- Powered by JavaScript
- Only one HTML file

PWA:

- Uses a lot of JavaScript but also works without it
- Works with multiple HTML files

Progressive Enhancement

Progressive Web App delivers a lot of features, but you don’t need to use all of them at once.

You might want to just add a manifest file so the users can add your app to the home screen.

Later on you can implement caching techniques to your service worker or any other PWA features like using camera device etc.

PWA browser support

https://www.goodbarber.com/blog/progressive-web-apps-browser-support-compatibility-a883/

Manifest file in depth

```
{
    ”name”: ”Full app name”,
    ”short_name”: ”app name”,
    ”start_url”: ”/index.html”,
    ”scope”: ”.”,
     ”display”: ”standalone”,
    ”background_color”: ”#fff”,
    ”theme_color”: ”#fff”,
    ”description”: ”App description”,
    ”dir”: ”ltr”,
    ”lang”: ”en-US”,
    ”orientation”: ”portrait-primary”,
    ”icons”: [{
            ”src”: ”/icon.png”,
            ”type”: ”image/png”,
            ”size”: ”48x48”
    }]
}
```

- Name visible on splashscreen
- Short name visible below icon
- Which page to load on startup
- Which page includes as PWA
- Should it be headless browser
- Background color of splashscreen
- Color of top bar in task switcher
- Description of your app
- Read direction of your app
- Main language
- Set and enforce default orientation
- Icons used for the splashscreen loading
  - -- Path to your icon
  - -- Type of file
  - -- Icon size, browser chooses best one for given device

PWA install banner requirements

https://developers.google.com/web/fundamentals/app-install-banners

Service worker

- Allows caching and running app offline
- Allows background synchronisation
- Allows to handle Push notifications

- Runs on separate thread
- Listening to the events
- Lives on after page is closed (background process)
- Manages all pages of given scope

Service worker events

Service Worker lifecycle – Service worker phase changes

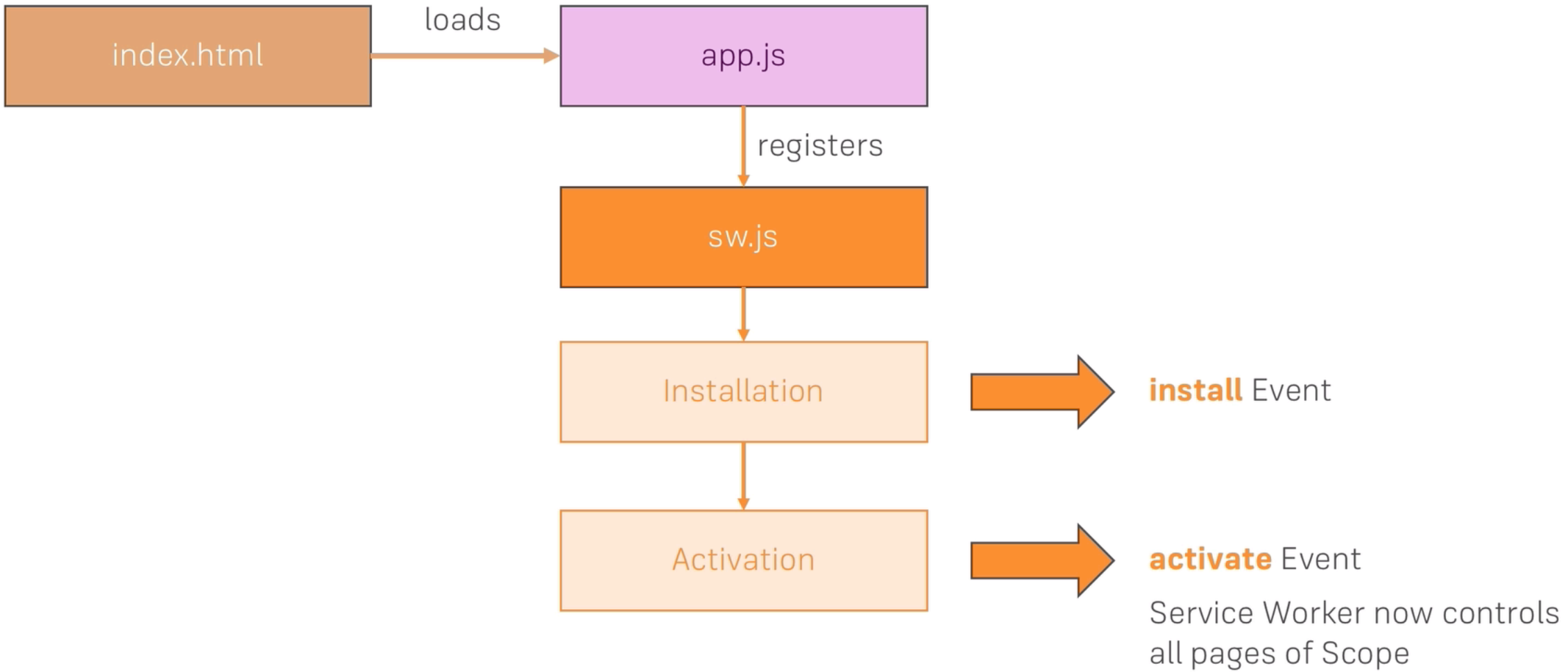
Fetch - Browser or Page-related JS initiates a Fetch request

Background Sync – Service worker receives Background sync event when connection is restored and is able to perform stored actions

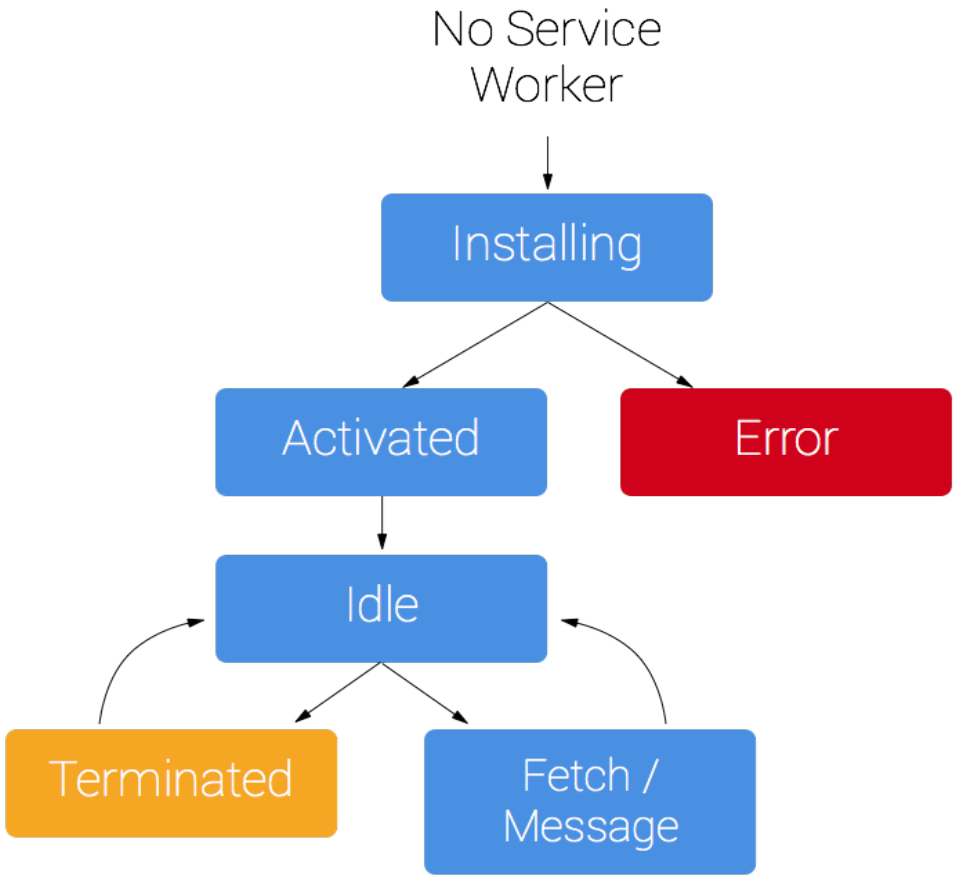
Push Notifications – Service worker receives Web Push Notification (from Server)

Notification interaction – User interacts with displayed Notification

Service Worker Lifecycle



Service worker lifecycle



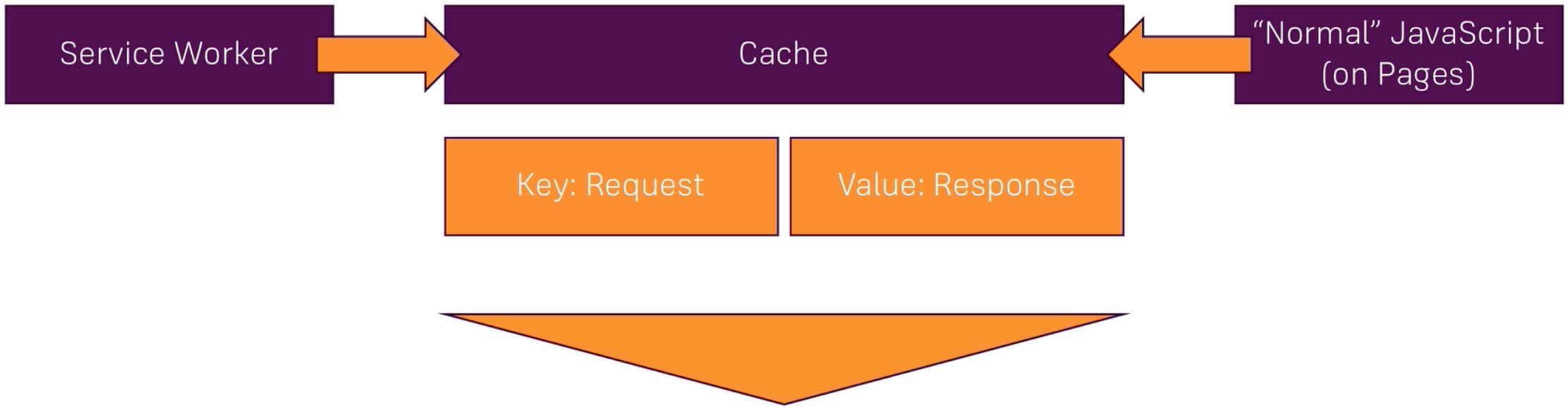
Types of caching

Static caching – caching app shell files, that don’t change that often

Dynamic caching – caching files like pictures that are not required for the app to launch but might change more frequently

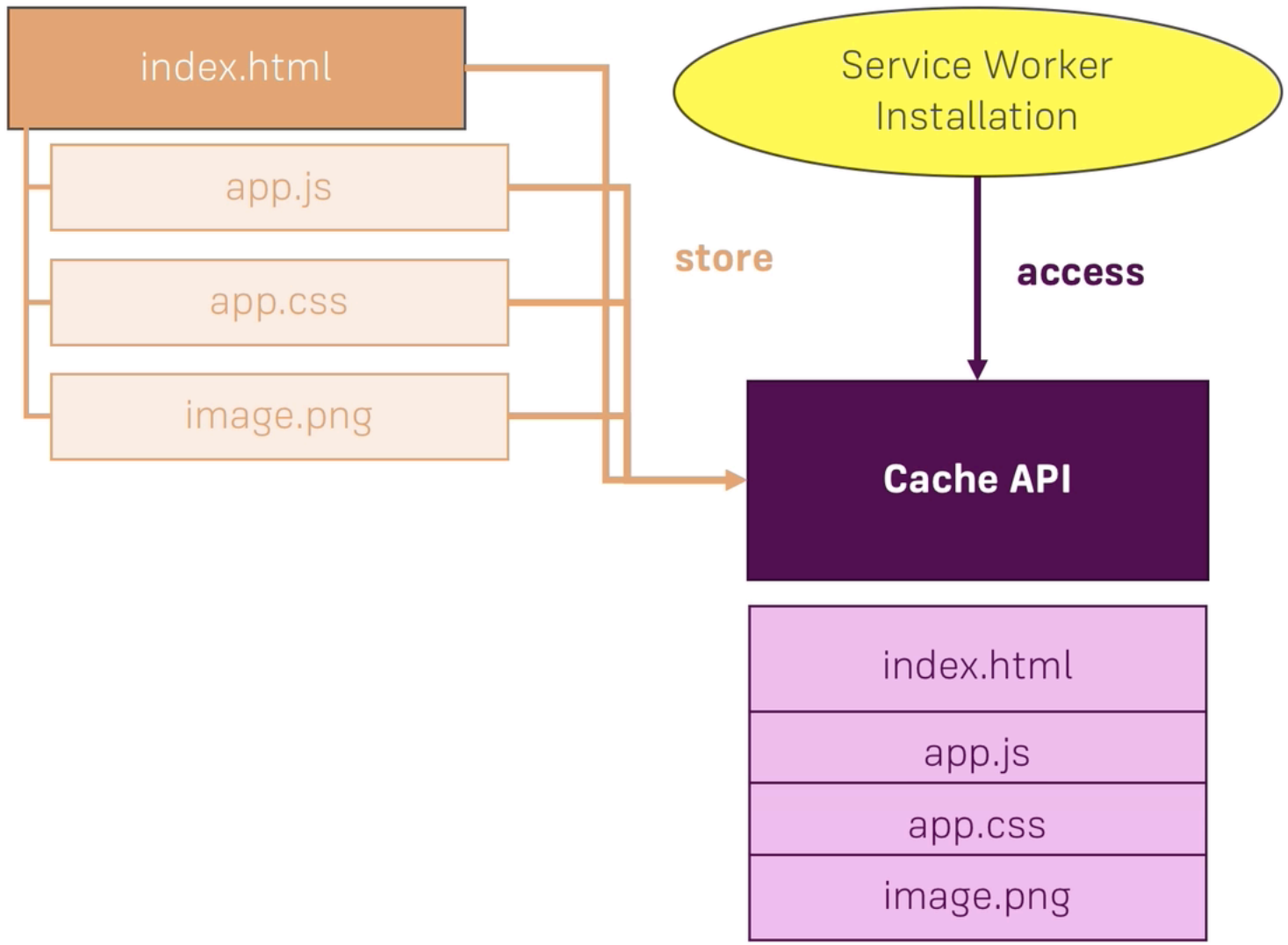
Dynamic content caching – for example caching JSON data returned from API

The Cache API



Cache Data can be retrieved instead of sending Network Request

Static Caching at Installation



Caching strategies – cache with network fallback

Caching every request, if something is not stored in the cache, then make a network request.

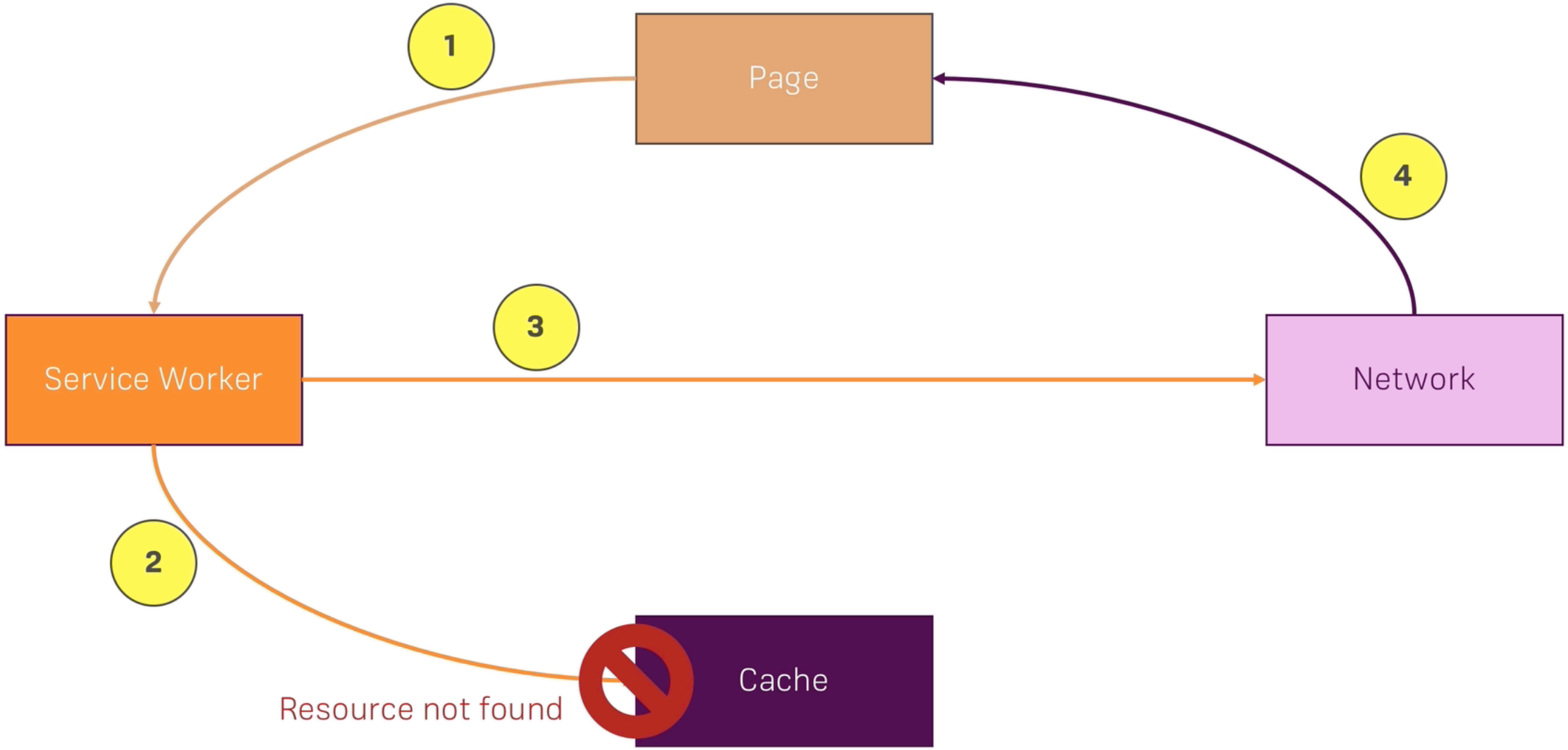
Pros:

- we can instantly load assets even if we have a network connection

Cons:

- assets never get an update!

Strategy: Cache with Network Fallback



Caching strategies – cache on demand

Cache on demand allows to cache resources as user interacts with our page.
New files are added to the cache through the front-end application.

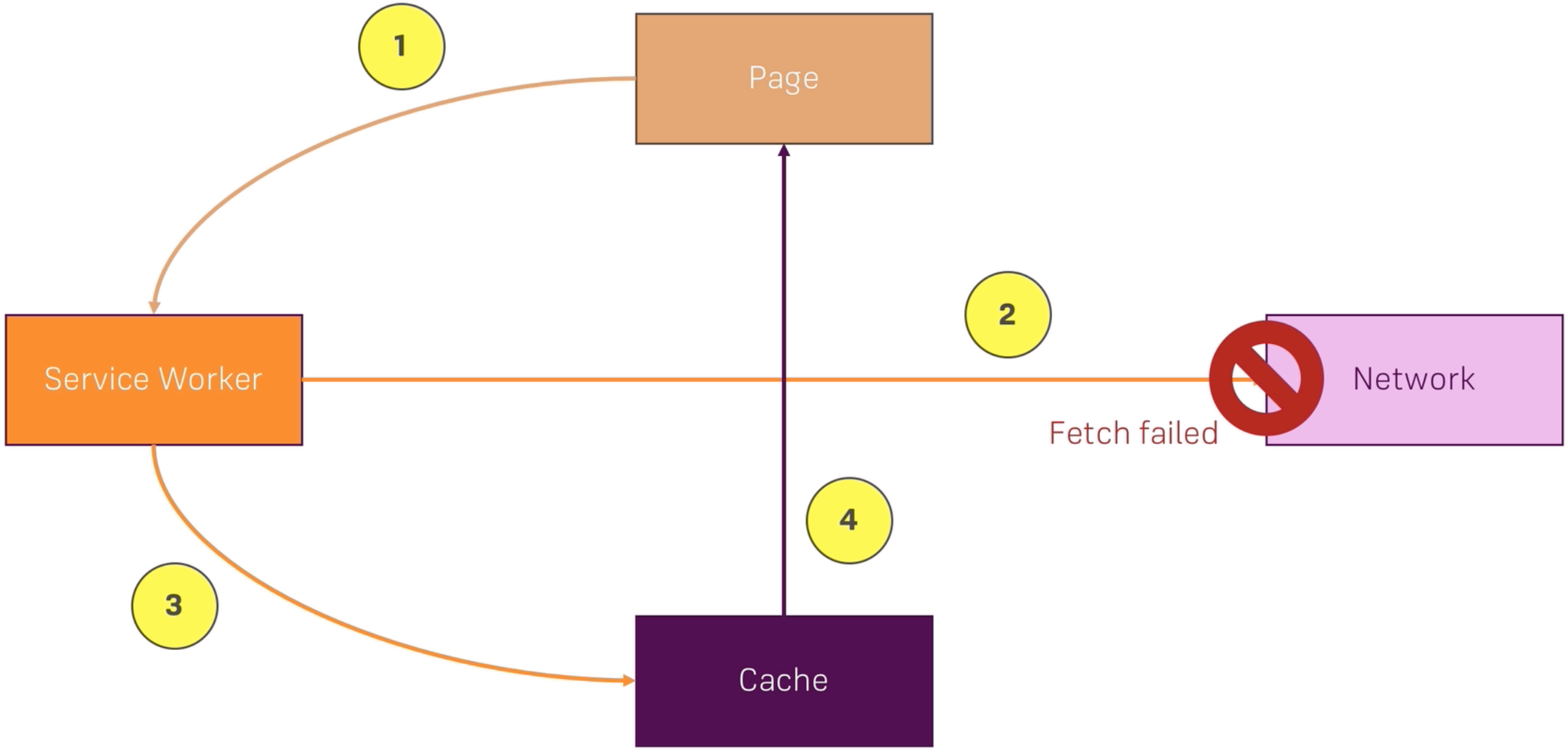
Caching strategies – network with cache fallback

Reaching to cache only as a fallback

Pros:
- we get always latest files
- we use cached files in case of offline mode or any error

Cons:
- we don’t get an advantage of cached files when network is available
- if request is getting 60 seconds timeout, user don’t see anything

Strategy: Network with Cache Fallback



Caching strategies – cache, then network

Serving content from the cache until assets gets fetched from the network.

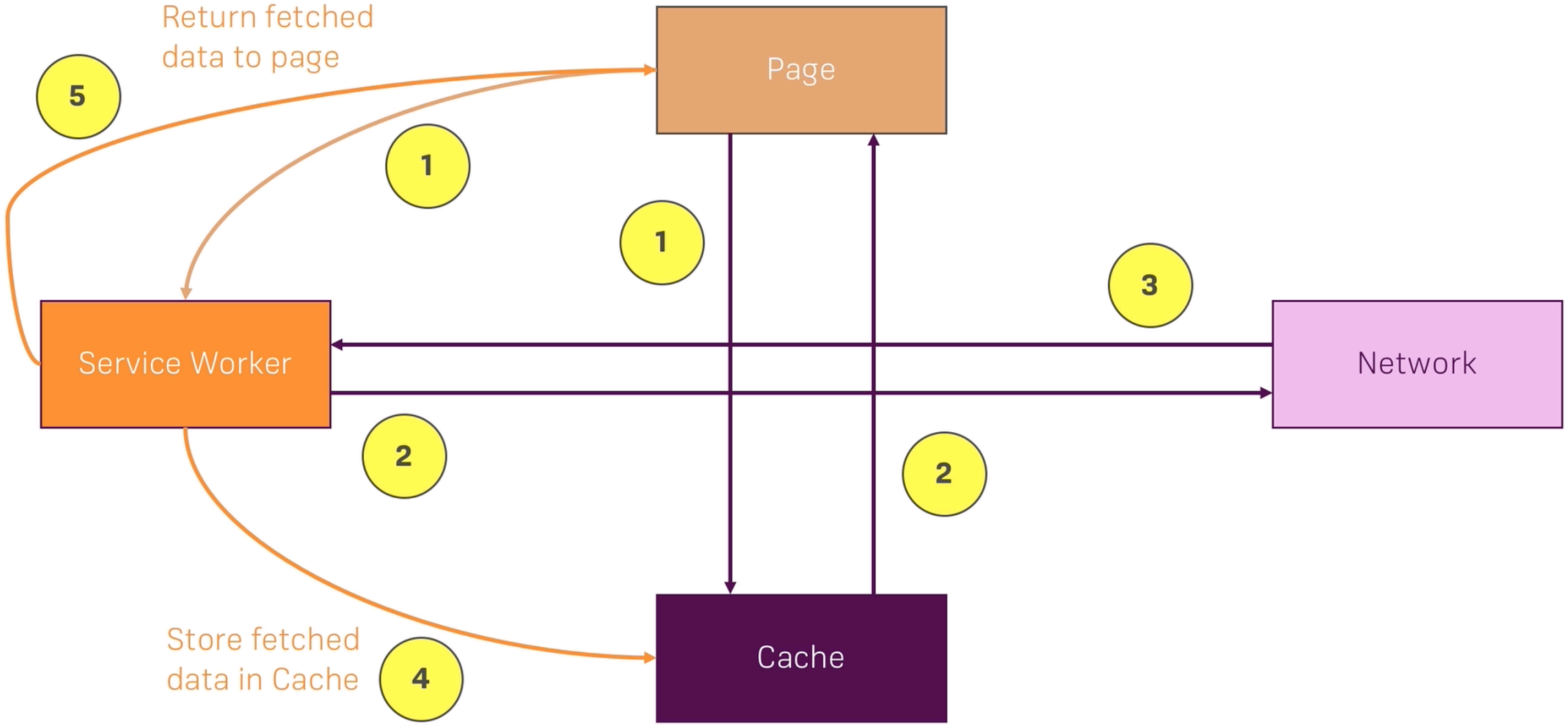
Pros:

- user sees the content very quickly from the cache, then gets updated version asap from the network

Cons:

- needs to be combined with *CACHE WITH NETWORK FALLBACK* or *CACHE ONLY* strategies for offline support

Strategy: Cache, then Network



Caching dynamic content vs Dynamic cache

What's the difference between caching dynamic content and dynamic caching?

dynamic caching – we use it for file assets like scripts, images, css, html etc.

caching dynamic content – dynamic content is coming from servers, mostly as JSON or XML format

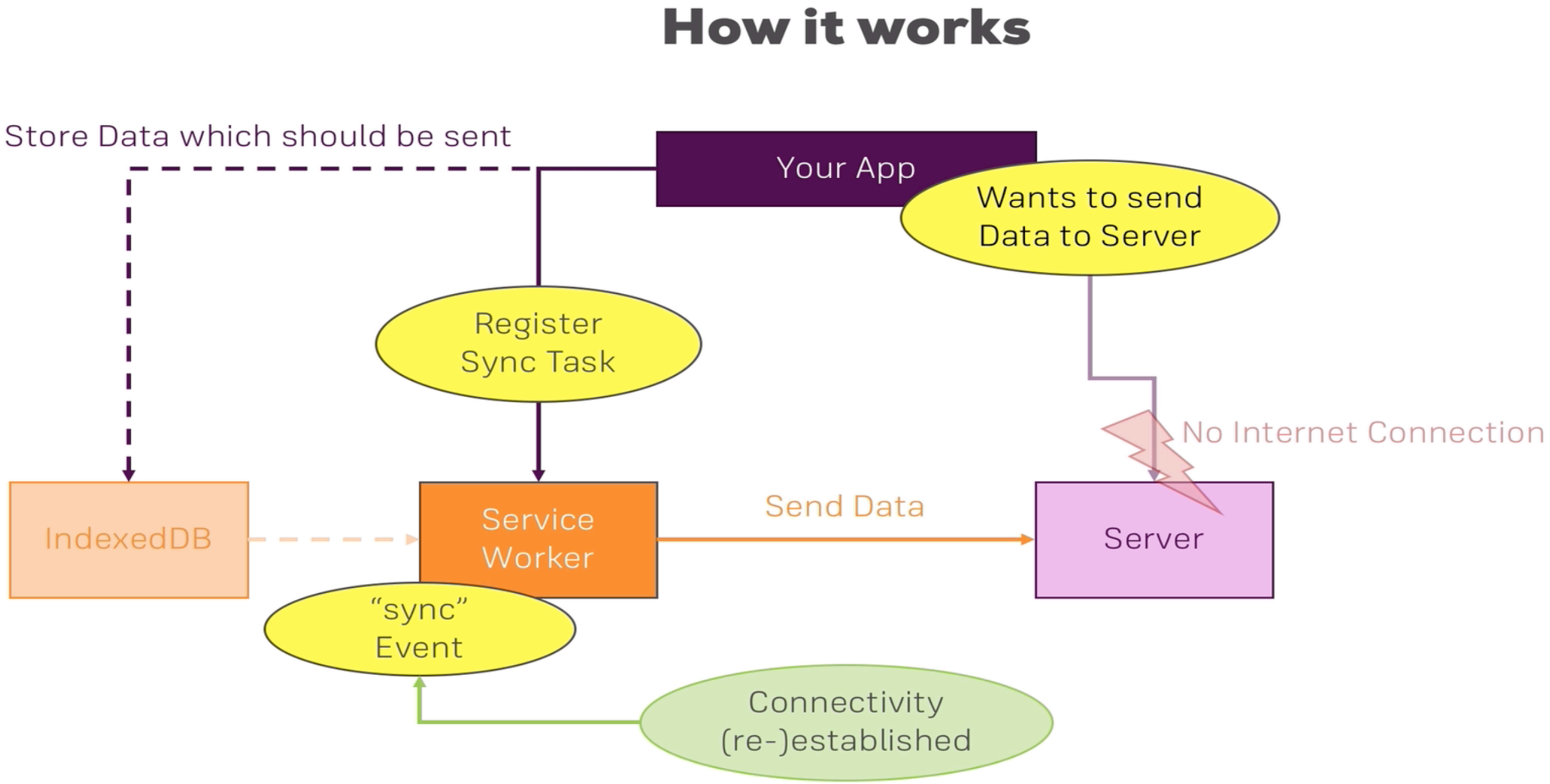
IndexedDB for caching dynamic content

IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API ***uses indexes to enable high-performance searches*** of this data.

IndexedDB is a ***transactional*** database system, like an SQL-based RDBMS. However, unlike SQL-based RDBMSes, which use fixed-column tables, ***IndexedDB is a JavaScript-based object-oriented database***. IndexedDB lets you store and retrieve objects that are indexed with a key; any objects supported by the structured clone algorithm can be stored. ***You need to specify the database schema***, open a connection to your database, and then ***retrieve and update data within a series of transactions***.

Source:
https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

Background sync



Push Notification