Maximiliano Firtman
@firt   firt.dev

Browser Data Storage

Frontend Masters

mobile+web developer & trainer

Maximiliano Firtman

@firt   firt.dev

- HTML since 1996
- JavaScript since 1998
- Authored 13 books
  - Free **web.dev/learn/pwa**
- Published 150+ webapps

Let’s Start!

What we'll cover

State of Browser Storage	Debugging Tools
Quotas	Persistance
Web Storage	IndexedDB
Cache Storage	FileSystem

Pre-requisites

github.com/firtman/browser-storage

Questions?

1

Introduction

What we'll cover

State of Browser Storage	Debugging Tools
Quotas	Persistance
Web Storage	IndexedDB
Cache Storage	FileSystem

Why Browser Data Storage

- **Increase User Experience**
- **Increase Performance**
- **Offline support**
- **We can store:**
  - User-generated content
  - App's State
  - Cached assets
  - Authentication tokens
  - Analytics

How does it work?

- **Using JavaScript we store and retrieve data that is stored locally in user's device.**
- **Browsers manage the implementation and security details.**
- **We should always treat it as data that can disappear anytime.**
- **The data will persist between browsing sessions.**
- **On most APIs, we won't require any explicit permission from the user.**
- **It works also for PWAs and Hybrid apps.**
- **Data is NOT shared to the server* or with other webapps**
  (*cookies is the only exception)

Some Important Concepts

- Origin
- Web Client
- Device
- User

Some Important Concepts

- Origin
- Web Client
- Device
- User

Origin

- Quick and Dirty: **an Internet domain**
- **Protocol + Host + Port**
- http://firt.dev
  https://firt.dev
  https://www.firt.dev
  https://firt.dev:4000
  are all different origins
- **Be careful with**
  1. **www prefix**
  2. **country TLDs, such as:**
     amazon.com, amazon.es
  3. **Subdomains, such as:**
     firt.dev, learn.firt.dev

eTLD+1 groups

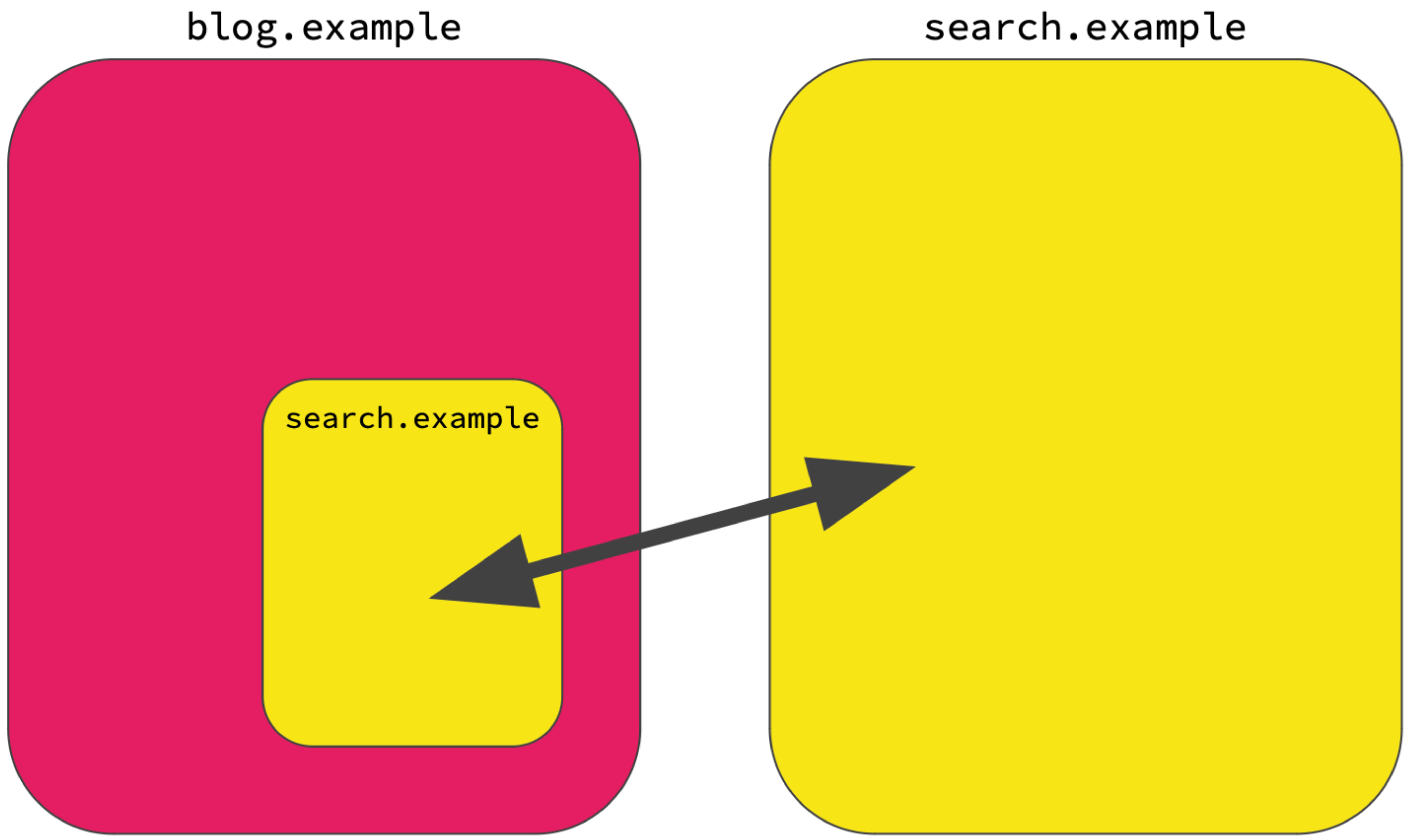
- Firefox doesn't see only origins and it generated different policies per eTLD+1 groups
- eTLD is a name for a public suffix (.com, .app, .co.uk, .ar, etc.)
- eTLD+1 is then, a registrable domain on an eTLD
- all subdomains of it will be part of the same group

eTLD+1 group sample

- .co.uk is an eTLD
- amazon.co.uk is an eTLD+1
- amazon.co.uk, www.amazon.co.uk, www.primevideo.amazon.co.uk are all part of the same eTLD+1 group.

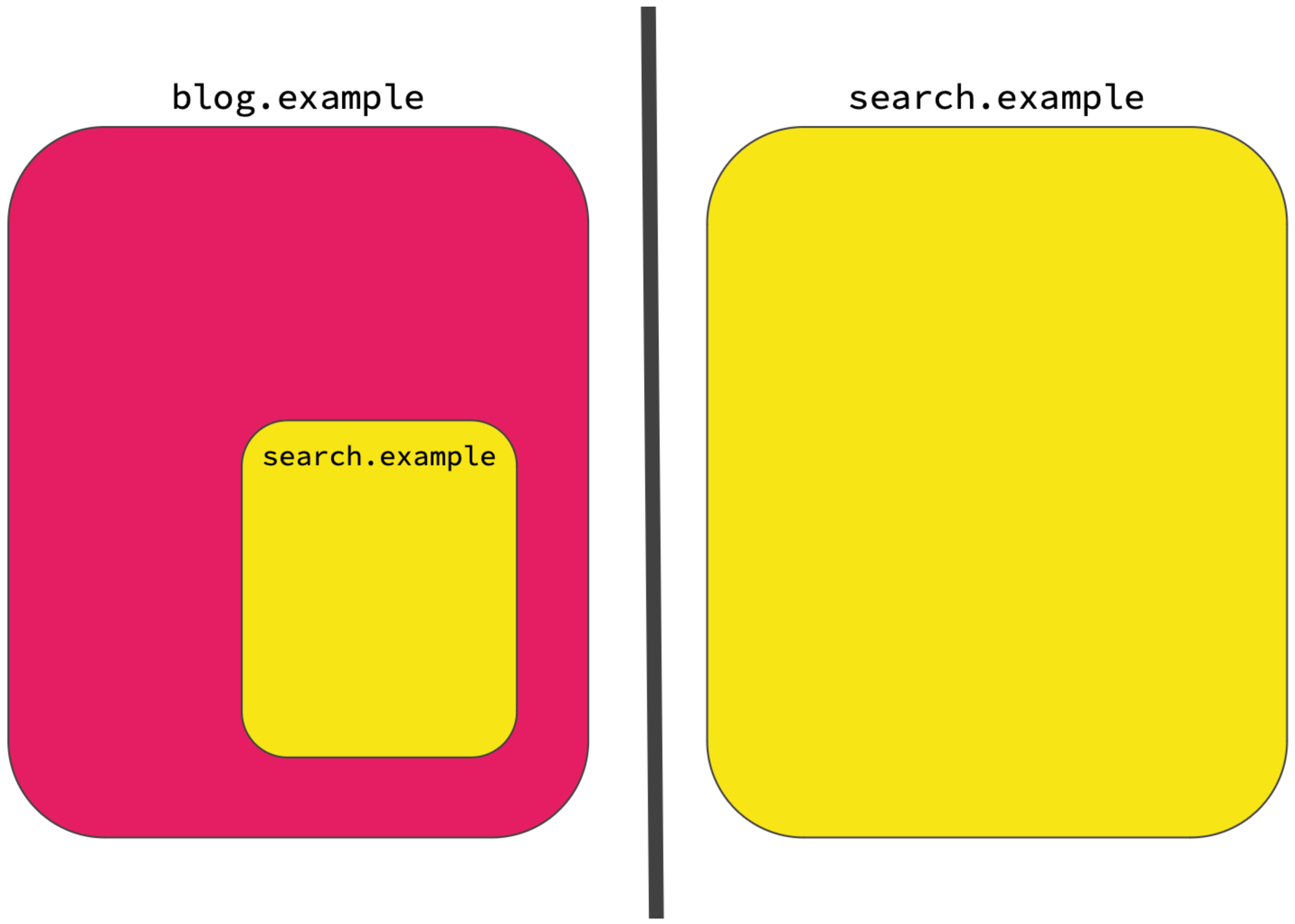
Partitions And Safari

- On most browsers, storage is per origin or eTLD+1
- In this case, search.example will use the same storage and data on both navigation cases



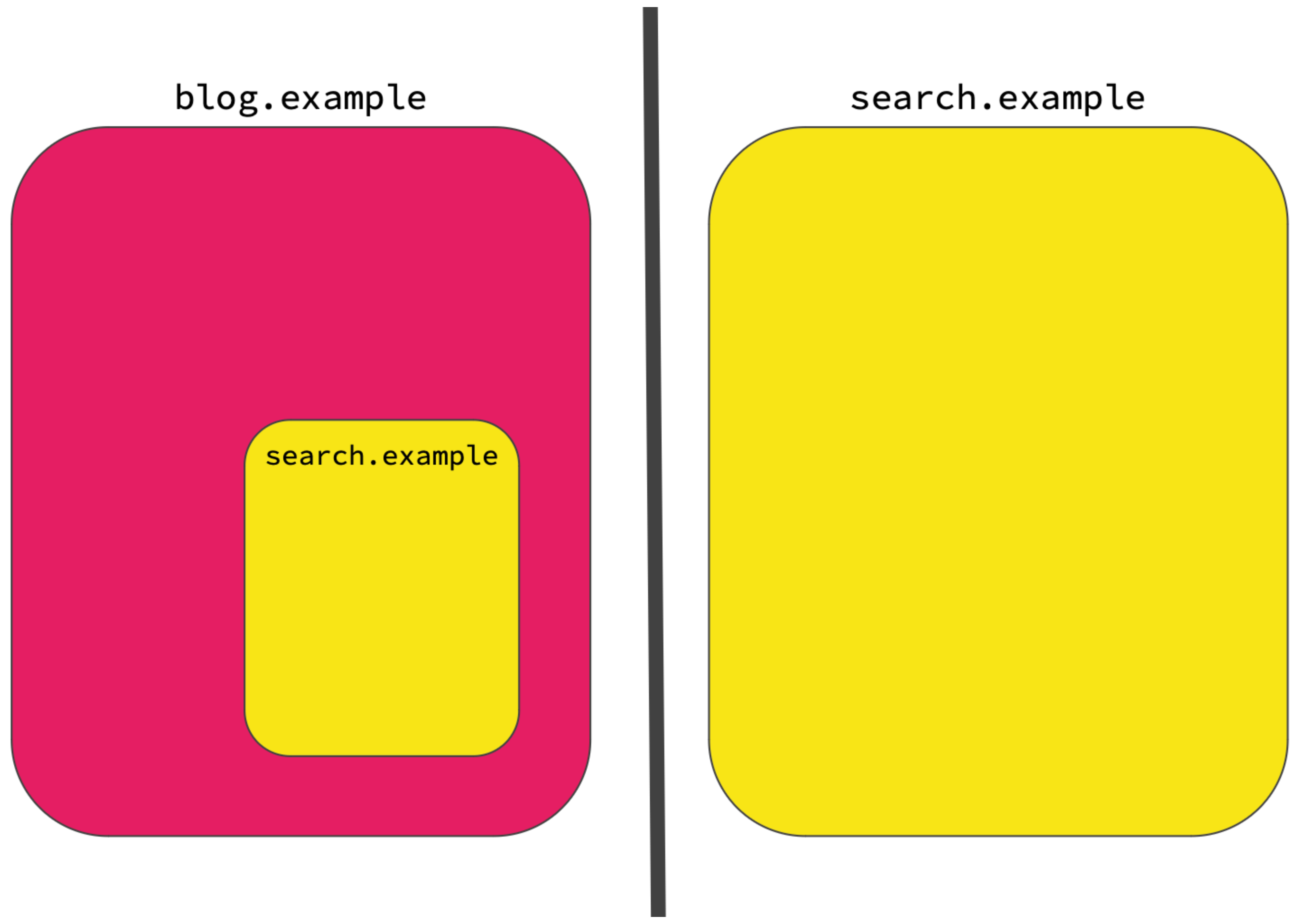
Partitions And Safari

- **WebKit make a partition on these cases to avoid fingerprinting and increase privacy**



Partitions And Safari

- search.example will not share storage in these cases.
- One partition is search.example and other is blog.example+search.example



Web Client

It's a piece of software than can navigate to a website

- Browser instance
- Progressive Web App (PWA) installed from a Browser
- Native app using a Web View
- Native app using an In-App browser taking advantage of a browser API
    - Custom Tabs (CT)
    - SafariViewController
- Store app using a Trusted Web Activity (TWA)

Web Client

Sometimes it's the same Web Client

- Chrome on desktop browsing twitter.com and an standalone Twitter PWA installed from the same Chrome.
- Safari on iOS browsing YouTube and the Twitter app In-App browser browsing YouTube (it uses SFViewController).
- Chrome on Android browsing TikTok, an installed app from the Play Store using a TWA to TikTok, and an app running a Custom Tab to TikTok.

On ”native” apps, client-side data is contained to that device and it goes to a cloud backup

On web apps, the world is much more complex

Many possibilities

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop
Tablet
Minnie
Mickey
www.disney.com

Data Storage

Chrome
Instagram
Chrome
Safari
PWA
Safari
PWA
Phone
Laptop (same OS user)
Tablet
Minnie
Mickey
www.disney.com

So the data will be available on same

Origin
Web Client
Device and OS account
Device and OS account
User

(exceptions apply)

The data we store **will** be available when

- **Navigating to the same origin, and same device, on same OS account in any time in the future* and**
  - **1) the same web client is used**
  - **2)** On desktop and Android: **a browser's tab and a PWA installed from the same browser are used**
  - **3)** On Android: **a browser's tab and a Play Store app using TWAs are used**
  - **4)** On Android, iOS and iPadOS:

    A default browser's tab and an In-App browser using CT or SafariVC are used

* some conditions apply

The data we store **won't** be available

- The same device using a different client
- On iOS and iPadOS, same device using Safari and icons in the home screen
- Same user using the same client on different device, even if logged in with same account
- A device restored from the cloud on most cases

We need to plan for the best-effort, worst case scenario:

the data won't be available

On most common use cases, **the data is there**

On some cases,
there will be many copies of the data

If you host several web apps in the same origin, **prefix storage names** to avoid conflicts

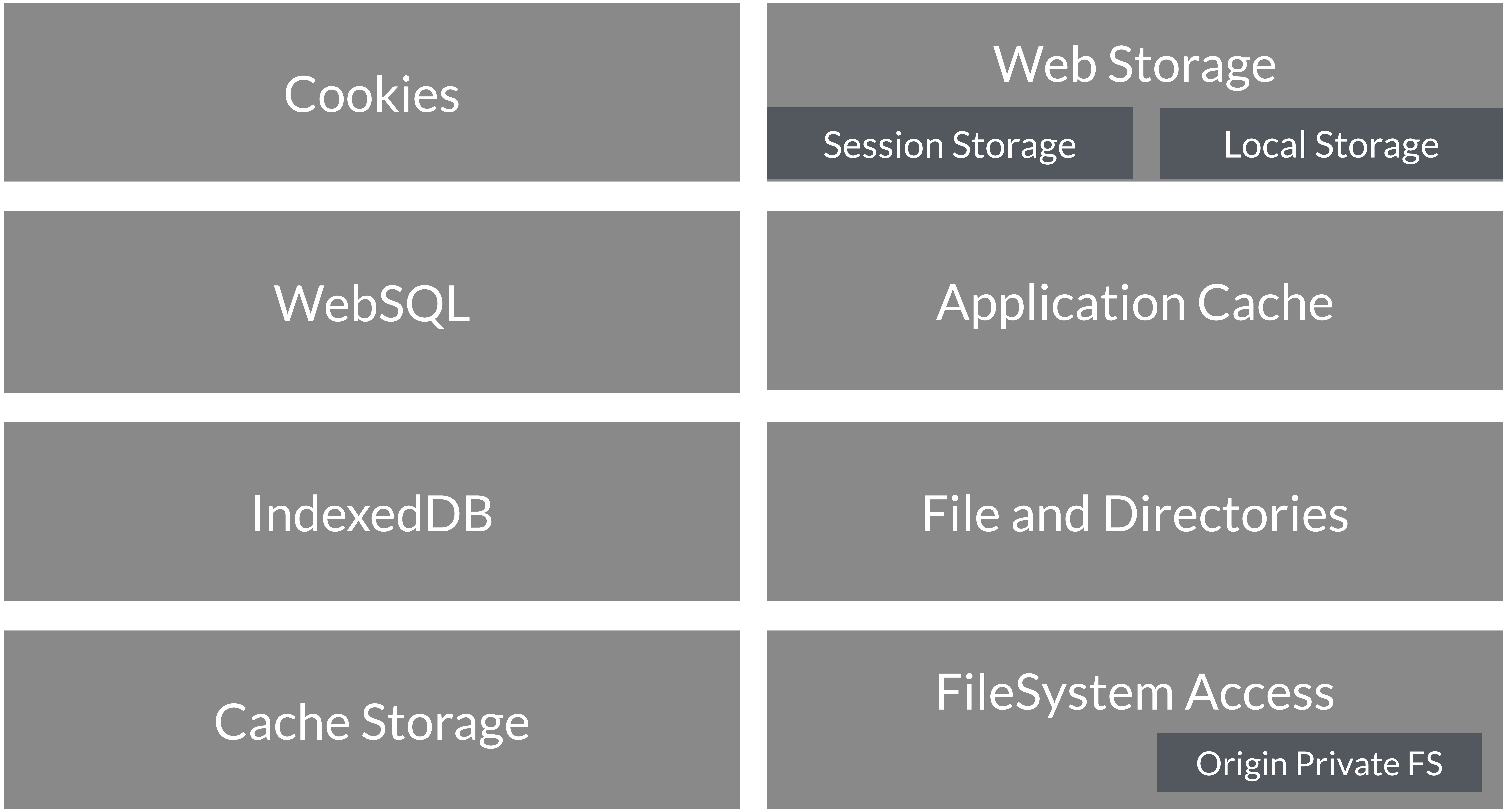
2

State of APIs

APIs for Browser Data Storage

Cookies	Web Storage
WebSQL	Application Cache
IndexedDB	File and Directories
Cache Storage	FileSystem Access

APIs for Browser Data Storage



APIs for Browser Data Storage

Cookies

NOT SUITABLE

Web Storage

Session Storage

Local Storage

WebSQL

Application Cache

IndexedDB

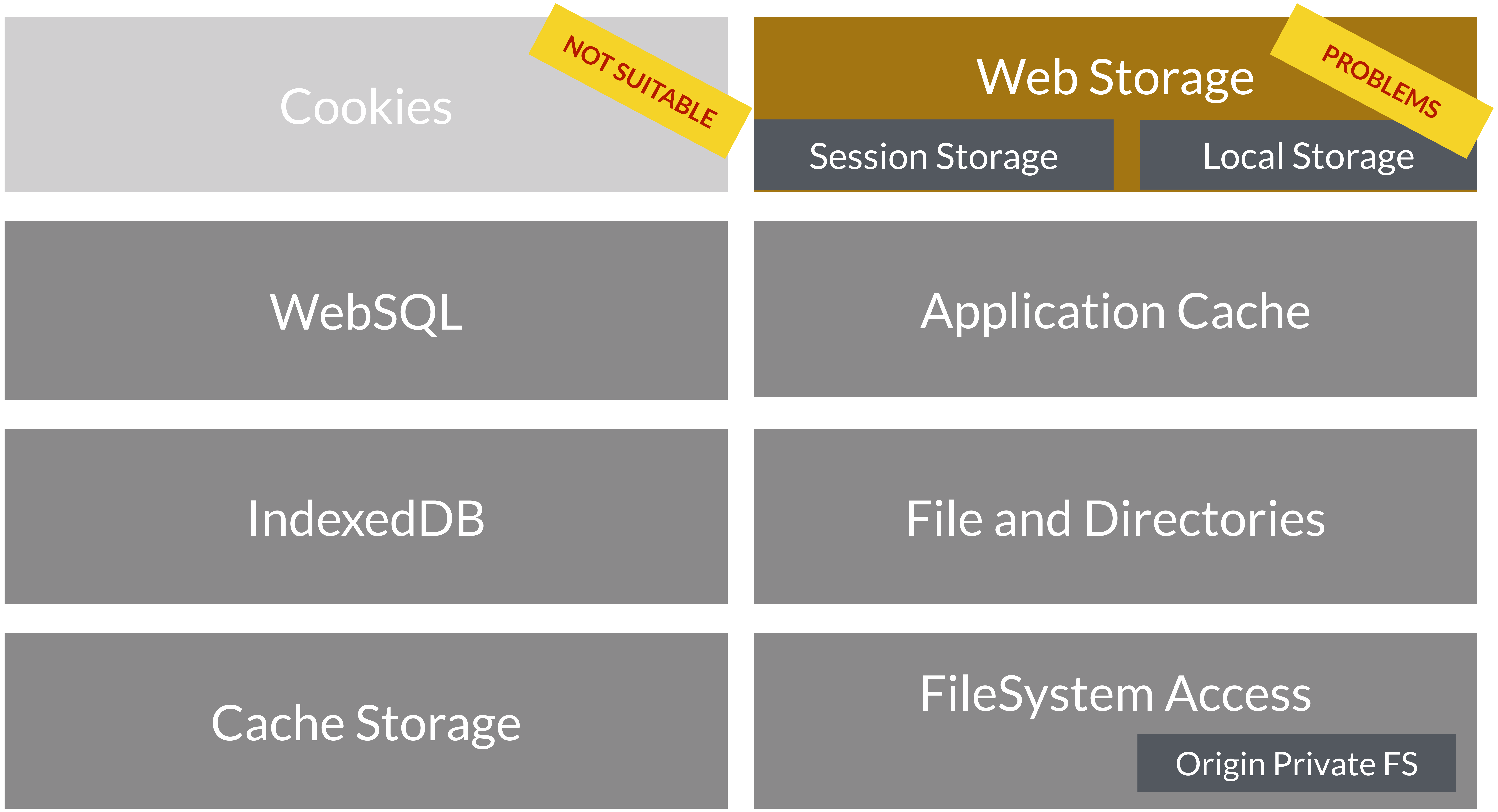
File and Directories

Cache Storage

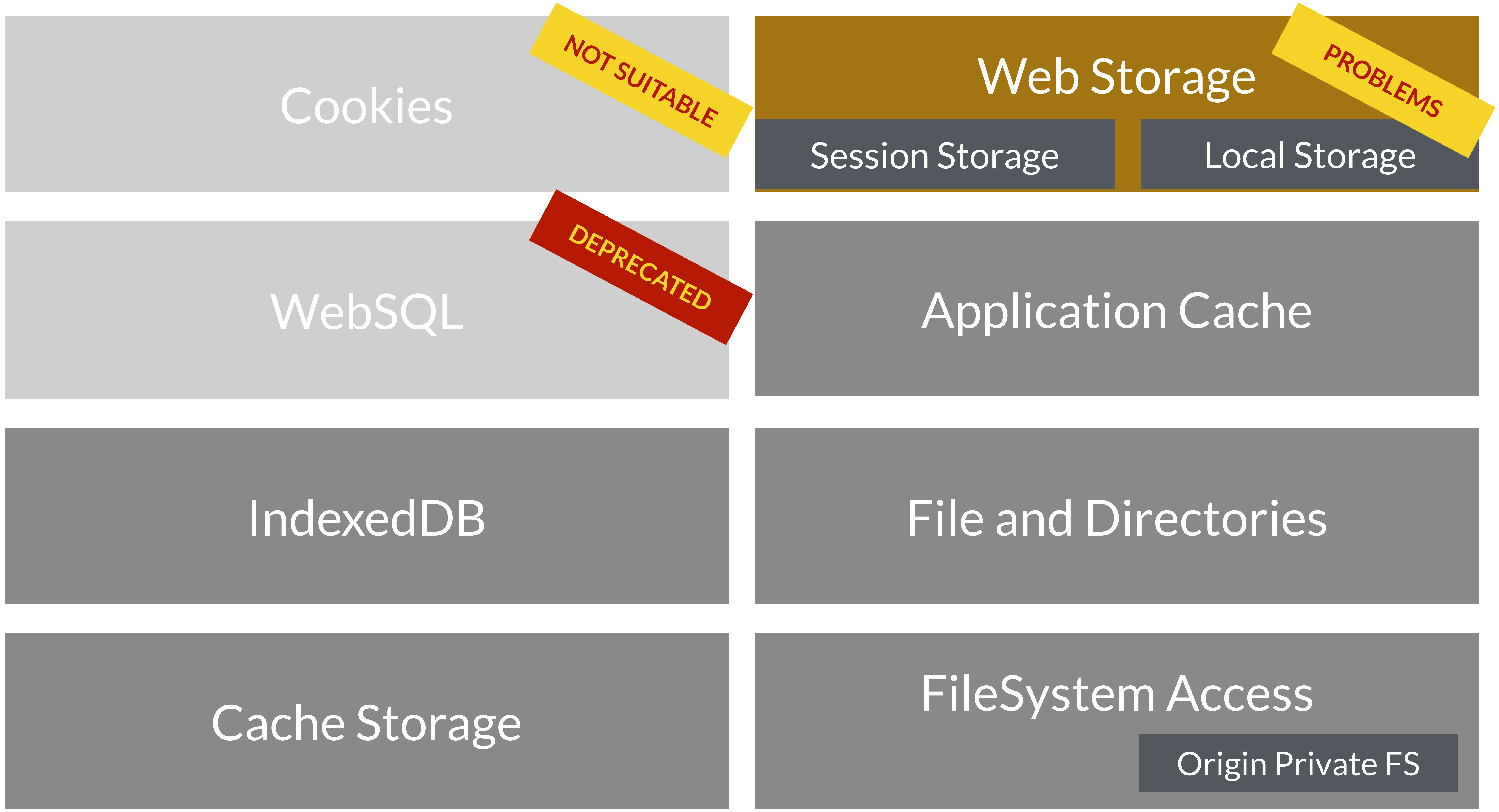
FileSystem Access

Origin Private FS

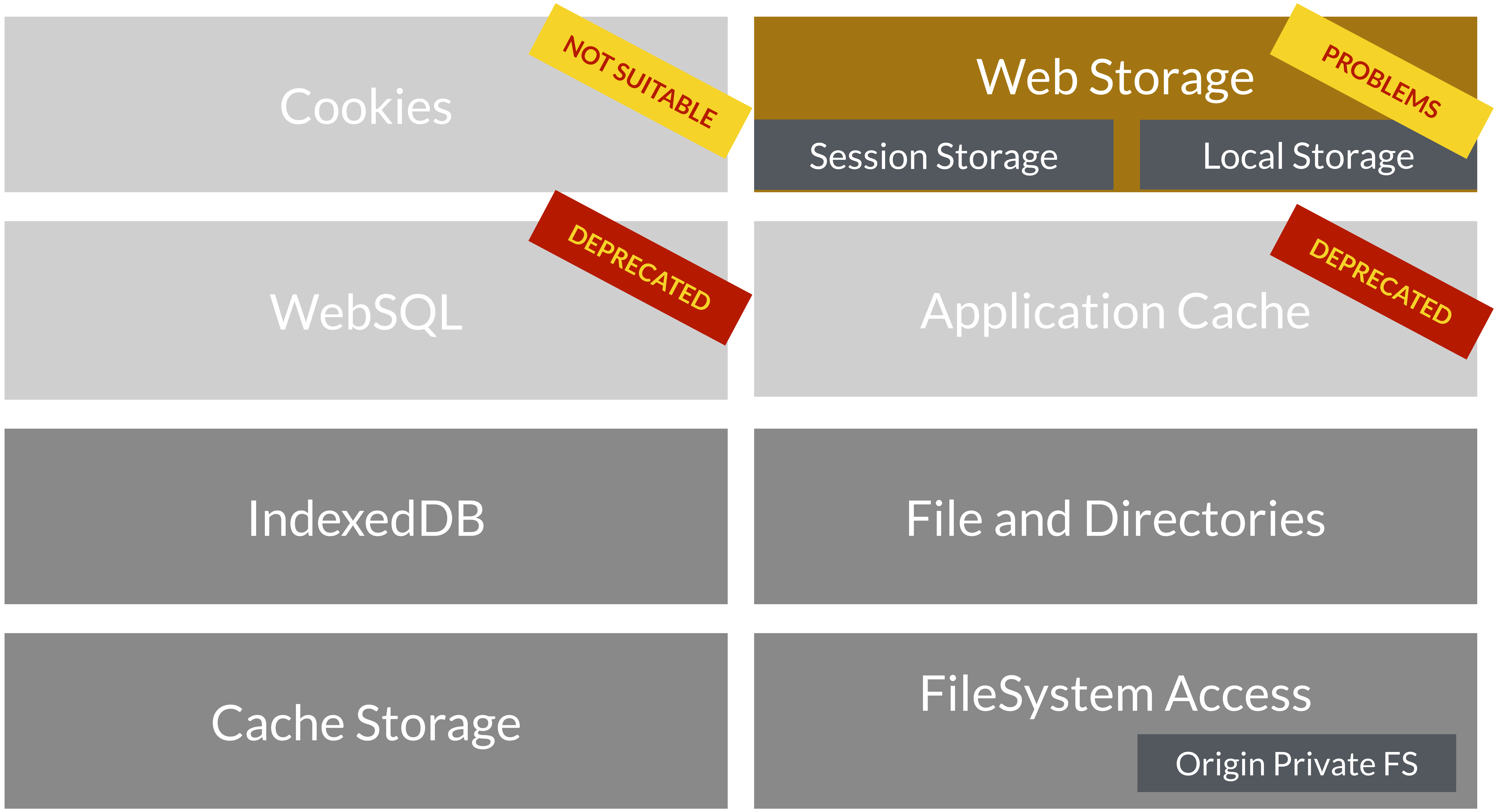
APIs for Browser Data Storage



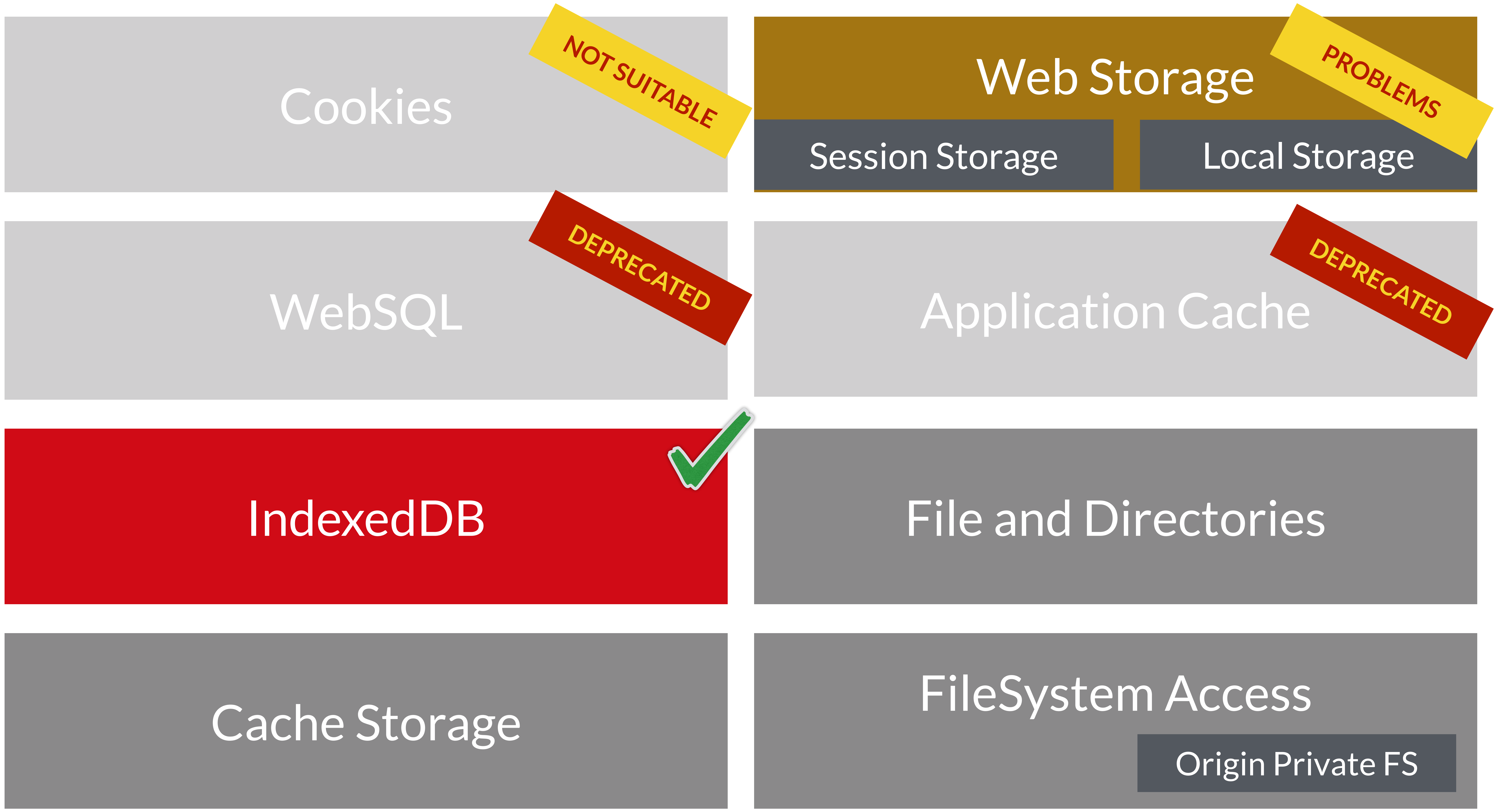
APIs for Browser Data Storage



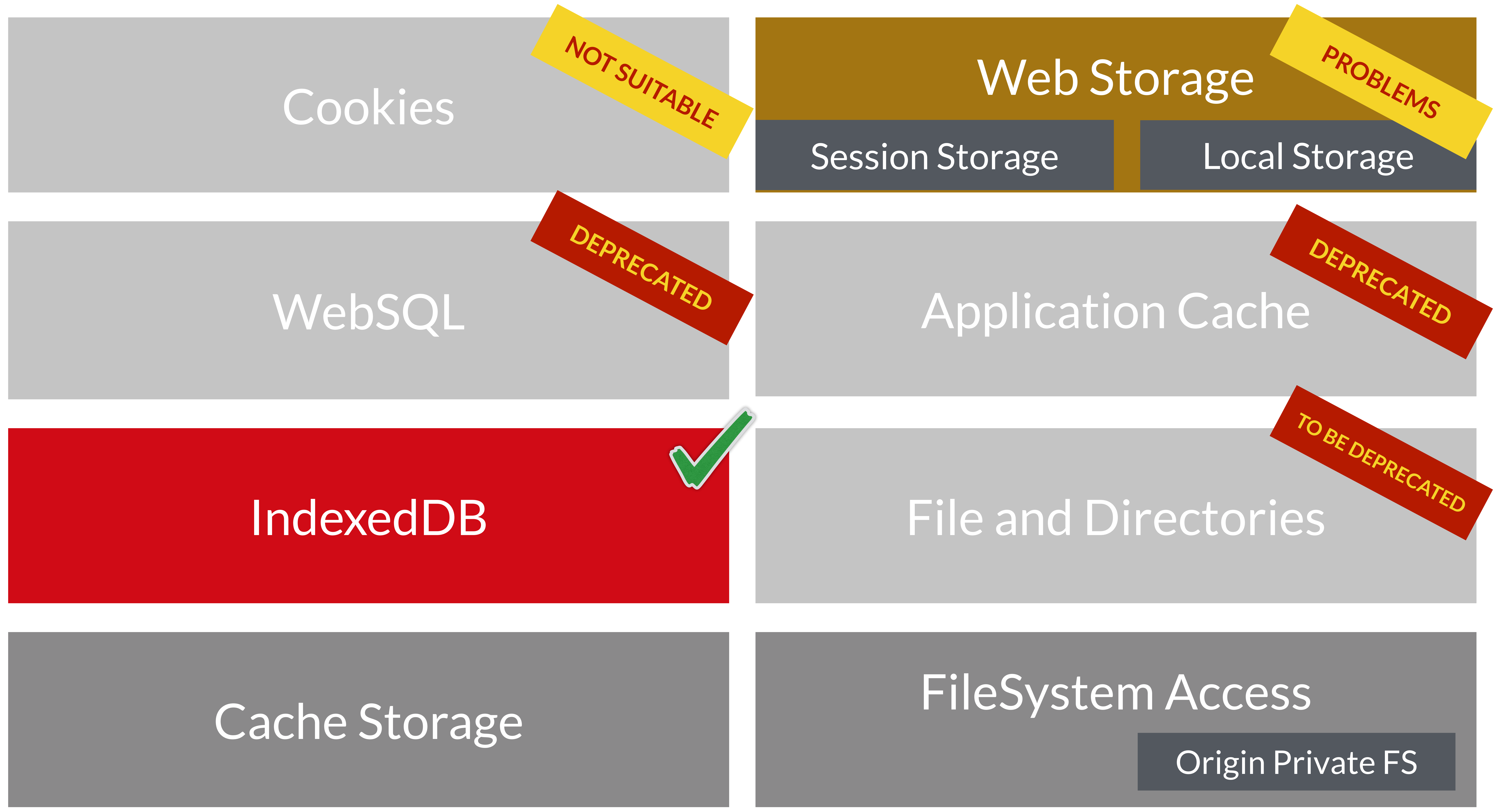
APIs for Browser Data Storage



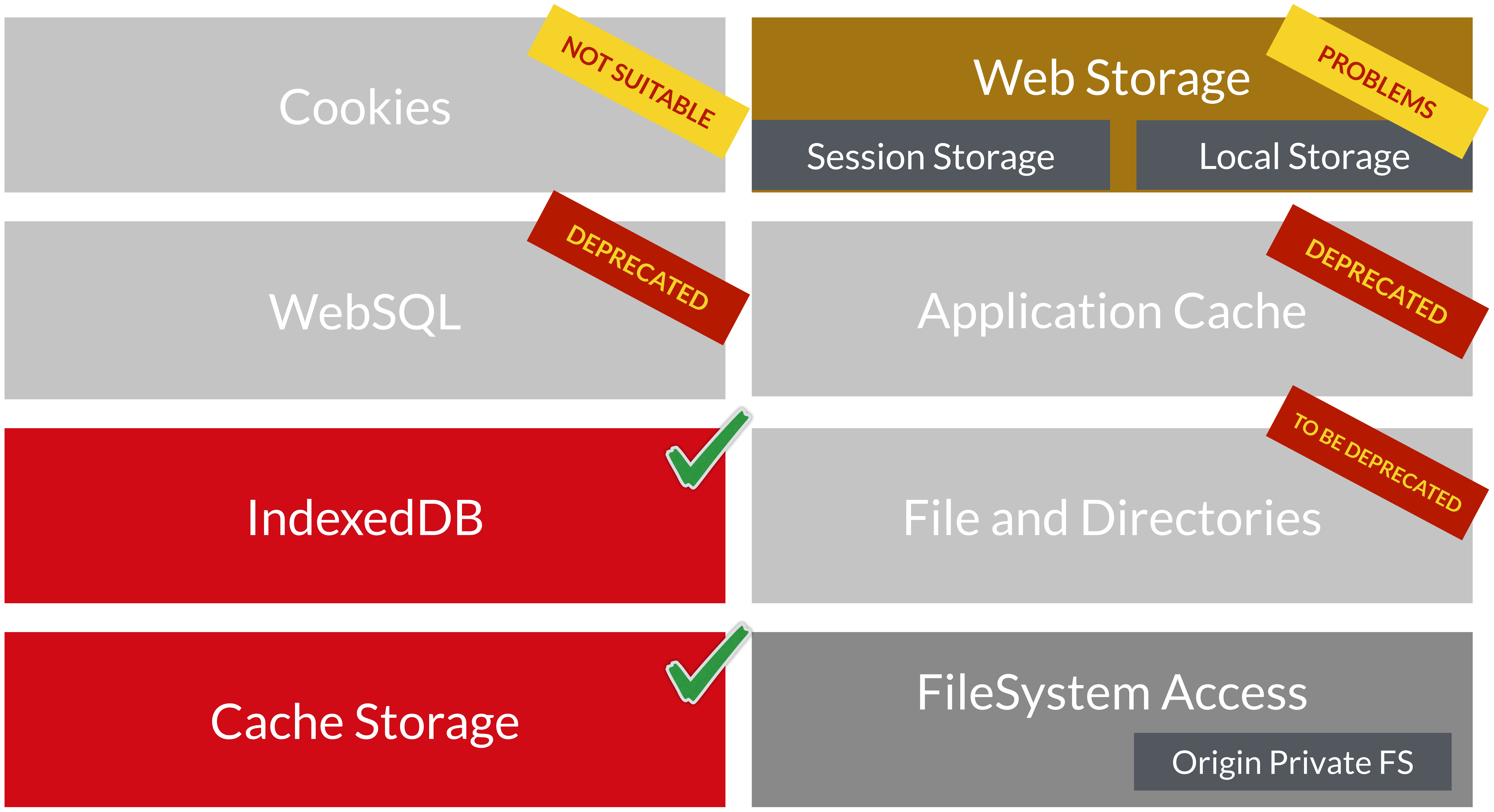
APIs for Browser Data Storage



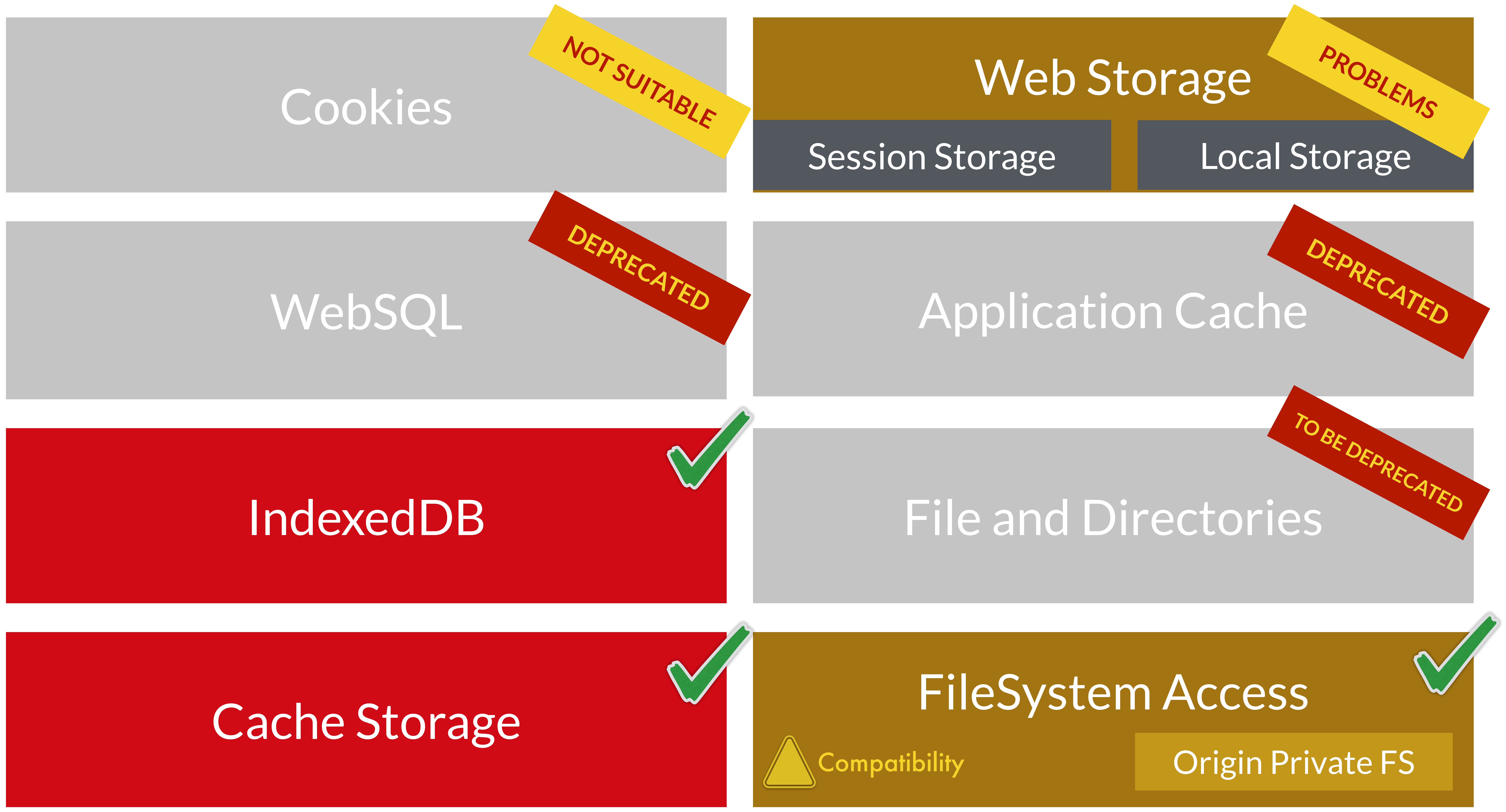
APIs for Browser Data Storage



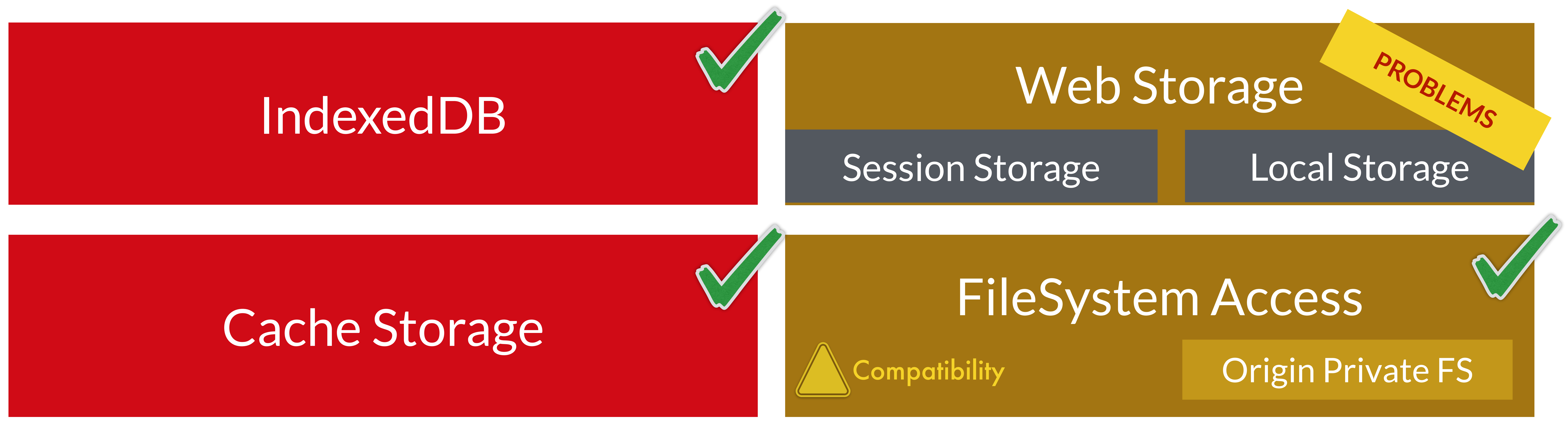
APIs for Browser Data Storage



APIs for Browser Data Storage



APIs for Browser Data Storage



Data Storage APIs Comparison

	Stores...	Using a key of...	Grouped in...	Up to...
IndexedDB	JS Objects and binary data	A keyPath within the object	Object Stores in Databases	Available Quota
Cache Storage	HTTP Responses	HTTP Request	Caches	Available Quota
Web Storage: *Session* *Local*	Strings	String	N/A	12MB 5MB
FileSystem Access	Files	N/A	N/A	N/A

New ideas!

Do you still want to use SQL?

Do you want to create your own API?

Thanks to WebAssembly and
IDB or FS APIs it's possible!

Web Storage

- **Simple API**
  - It stores only one **string** per key
  - The key for entries is also a **string**
- **Synchronous API**
  - ⚠️ performance issues
  - ⚠️ not available on Workers or Service Workers
- **We should try to avoid using it today**
- **You can emulate them with IndexedDB**

Web Storage

It offers the same API on localStorage and sessionStorage global objects

script.js

```
localStorage.setItem("key", "value");
const data = localStorage.getItem("key");

localStorage.removeItem("key");
localStorage.clear();
```

`localStorage`

It persist data between navigation and browser sessions

Quota is typically 5MB per origin

Strings are stored in UTF-16

⚠️ At the end, it's around 2.5MB per origin `sessionStorage`

It persist data within a browser's session

- Include page reloads and restores
- ⚠️ What's a session on mobile?

Quota is typically between 5MB and 12MB

To increase performance, quota and reachability, let's use **IndexedDB** instead of Web Storage.

Browser Data Storage

3

Debugging Tools

Workshop time

storage-quota.glitch.me

⚠️

All browser data storage
is public to the user

4

Quotas and Persistence

Quota includes

- **One quota for all storages:**
  - **All the data from APIs:**
    - Local Storage
    - IndexedDB
    - Cache Storage
    - FileSystem (Origin Private FS)
  - **Service Worker registrations**
  - **Web App Manifests from installed PWAs**

Quota does not include

- Cookies
- Files cached by the browser
- Session Storage
- Files created with the FileSystem Access API (on the real FS)

Quotas per browser

- **Chromium:** 60% of total disk space per origin
- **Firefox:** 50% of total disk space with a maximum of 2GB per group (eTLD+1)
- **Safari:** 1GB per partition with increments of 200Mb with user's permission

More complexity

- Chrome Incognito mode: **5% total disk space**
- Chrome with "Clear cookies and site data when you close all windows": **300MB**
- Other browsers Private mode: **from zero storage (ephemeral) to APIs not available**

Workshop time

filldisk.com

Storage per origin can be defined as

Best Effort

Persistent

Best Effort

- **It's the default state per origin**
- **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

Persistent

- **Persistent will keep storage unless**
  - User intervention happens
  - Device is reset

It doesn't clear the data on storage pressure

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

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

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);
}
```

Quotas are estimations; they will never give you exact data.

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

Safari criteria for Persistent Storage

Not specified ☹️

Our Project

- PWA for a Coffee
- Coffee Store
- Vanilla JavaScript
- Download assets and coding help

github.com/firtman/browser-storage

Workshop time

github.com/firtman/browser-storage

5

IndexedDB

IndexedDB

- It's a NoSQL data store
- We will be using IndexedDB 2.0
- It stores JavaScript objects or bytes
- Every entry has a key
- The API is asynchronous
- No permission needed from user
- It's available on Windows, Workers and Service Workers
- When storing objects, IDB clones them, and cloning happens synchronously

IndexedDB

- The API is event-based
- With a thin wrapper we can convert it in a Promise-based API
- It supports transactions
- It supports DB versioning

On top of IDB

- **SQL on IDB**
  JsStore, sqlite-worker
- **Web Storage on IDB**
  idb-localstorage, localforage
- **Other APIs for IDB**
  dexie, IndexedDB ORM, idb

Origin
Database
Database
Object Store
Object Store
Object Store
Object Store
Object
Object
Object
Object
Object
Object
Object
Object
Object
Object

Open a IDB database with name and version number
Does it exists with that name?
NO
YES
🔄 Upgrade event
Is the version greater than the browser version?
YES
NO
Is the version equal to the browser version?
NO
YES
⚠️ Error
✅ Success

Opening a DB

Standard API (non-event based)

script.js

```
let db;
const request = indexedDB.open(name);

request.onerror = (event) ⇒ {

};
request.onsuccess = (event) ⇒ {
  db = event.target.result;
};
```

Opening a DB

Using the idb Promise-based wrapper

script.js

```
// Open a DB
const db = await idb.openDB(name, version);

// Open a DB and handle upgrade
const db = await idb.openDB(name, version, {
   upgrade(db, oldVersion, newVersion, tx, event) { }
   // more event-based functions such as `blocked`
});
```

Creating an Object Store

script.js

```
// No key
const objectStore = await db.createObjectStore(name);

// With keyPath
const objectStore = await db.createObjectStore(name,
        { keyPath: property_name } );

// With Key generator
const objectStore = await db.createObjectStore(name,
        { autoIncrement: true } );
```

Deleting a DB

script.js

```
// Delete a DB
await idb.deletedb(name);

// Delete a DB and handle block
const db = idb.deletedb(name, {
   blocked(db) { }
});
```

Keys for Data Stores

Key Path

Key Generator

Indices for Data Stores

Key Path (`keyPath`)	Key Generator (`autoIncrement`)	Description
No	No	This object store can hold any kind of value, even primitive values like numbers and strings. You must supply a separate key argument whenever you want to add a new value.
Yes	No	This object store can only hold JavaScript objects. The objects must have a property with the same name as the key path.
No	Yes	This object store can hold any kind of value. The key is generated for you automatically, or you can supply a separate key argument if you want to use a specific key.
Yes	Yes	This object store can only hold JavaScript objects. Usually a key is generated and the value of the generated key is stored in the object in a property with the same name as the key path. However, if such a property already exists, the value of that property is used as key rather than generating a new key.

Quick Transactions

script.js

```
// New value/object
await db.add(storeName, value);

// Define a value/object in a store with a key
await db.put(storeName, value, key);

// Delete a value
await db.delete(storeName, key);

// Delete all values
await db.clear(storeName);
```

Quick Transactions

script.js

```
// Get count of values/objects in a store
const count = await db.count(storeName);

// Get all values/objects in a store
const values = await db.getAll(storeName);

// Get one value/object by key
const value = await db.get(storeName, key);
```

Workshop time

Simple IDB storage

Workshop time

Database

Creating an Index

script.js

```
// Index without unique values
objectStore.createIndex(name, property_name,
{ unique: false });

// Index with unique values enforcement
objectStore.createIndex(name, property_name,
{ unique: true });
```

Quick Transactions from Indexes

script.js

```
// Get all values/objects from an index
const values = await db.getAllFromIndex(storeName,
                    indexName, valueFromIndex);

// Get one value/object from an index
const value = await db.getFromIndex(storeName,
                    indexName, valueFromIndex);
```

Advanced IDB

- Transactions
- Cursors
- Filters for Cursors
- Performance

6

Cache Storage

Cache Storage

- It's part of the Service Worker spec, but not tied to the SW's scope
- We can create different storages (caches) under a name
- Every cache and store HTTP responses (headers + body)
- It stores them under an HTTP request key
- The API is asynchronous
- No permission needed from user
- We can store, update, delete and query HTTP responses by URL or request
- While typically we use it within a Service Worker, it's available in the Window's scope

Common Scenarios

- Pre-cache Assets
- Cache Assets on the fly
- Serve assets from a Service Worker for performance and offline access
- Query assets available for offline usage
- Create an offline page

Serving Resources

The service worker will respond for every request the PWA make

It can serve from the cache

It can forward the request to the network

It can synthesize a response

Any mixed algorithm is possible

Workshop time
Caching images

Your PWA
Service Worker
Cross-Domain Servers / CDNs
Your Server
Client
Proxy Middleware
Servers
User's Device - Browser
Cloud

Workshop time
Service Worker

makes a request
1
Your PWA
Service Worker
cache miss
forwards the request
to the network
3b
Network
same-origin and cross-origin
cache hit
responds with cached
response
3a
2
searches in cache
Cache
User's Device - Browser

Cache Serving Strategies

Cache first

Network first

Stale while revalidate

Updating Resources

Files are saved in the client

Updating files in the server won't trigger any automatic change in the client

We need to define and code an update algorithm

It will need a process within your build system for hashing or versioning files

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

Workshop time
Delivering Assets

Workshop time
Caching App Shell

7

FileSystem

FileSystem Access API

- We can read and write files in the real filesystem in user's device
- It will requiere user's permission
- It's Chromium-only
- It's an asynchronous API
- It doesn't count for the Quota
- It has an extension known as Origin Private FileSystem (OPFS) that is implemented by Safari

Opening a File

script.js

```
// Have the user select a file.
const [ handle ] = await window.showOpenFilePicker();

// Get the File object from the handle.
const file = await handle.getFile();

// Get the file content.
// Also available, slice(), stream(), arrayBuffer()
const content = await file.text();
```

Writing to an opened File

script.js

```
// Make a writable stream from the handle.
const writable = await handle.createWritable();

// Write the contents of the file to the stream.
await writable.write(contents);

// Close the file and write the contents to disk.
await writable.close();
```

Writing to a New File

script.js

```
const handle = await window.showSaveFilePicker({
    types: [{
        description: "Test files",
        accept: {
        "text/plain": [".txt"],
        },
    }]
});
const writable = await handle.createWritable();
await writable.write(contents);
await writable.close();
```

Workshop time

8

Best Practices

Database & Performance

It's better to store small objects

Remember you can use Web Workers

You can create custom indexes for faster access to collections of objects

Serverless ideas

Export data using FileSystem

Export/Import data using QR codes

Blockchain-based data storage

Being a Good Citizen

Don't store what you won't use

Clear the storage when it's not needed

Best-effort First

Capture quota errors and clear data

Offer the user a way to get user-generated content outside

Data Sync

In case you also store data on the server, many sync algorithms are available

Master Service Workers and sync APIs

Think about versions and data migration

Security

Remember all browser data storage is public

It's insecure by definition

Don't store private or sensitive data

If you store authentication data, it should be a token that can be revoked easily

Where to continue

Web Workers

Service Workers

Sync APIs

IndexedDB performance

WebAssembly-based DBs

What we've covered

State of Browser Storage	Debugging Tools
Quotas	Persistance
Web Storage	IndexedDB
Cache Storage	FileSystem

Fot

hi@firt.dev

@firt