goto; conference

CHALLENGES FOR FRONT END DEVELOPERS OF LARGE WEB APPLICATIONS

Graham Hinchly
FT Labs

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

Me

app.ft.com

Some things on the web are hard

When you have a large amount of code they get really hard

But why is it harder on the web than elsewhere?

Languages lack
encapsulation

Browser rendering model designed for documents

Built around assumption of always on, stable internet connection

How do we deal with this?

Scream in the street for a bit. YMMV.

Modularisation

Performance

Offline



Modularisation

Encapsulation, managing dependencies and using components

Encapsulation is our friend

CommonJS Spec

- Declare dependencies at the top of the file with `require`
- Expose public API via `exports`
- Not supported by the browser ☹

```
// Declare dependencies
var depA = require("depA");
var depB = require("./../depB");


/**
Module code
**/

// Export public API
exports.foo = foo;
```

Allows you to write JavaScript like this

But what if I want to use bits of other peoples' code...

Cleanly manages your JavaScript dependencies

npm + browserify

- Install from npm registry
- …or specify git URL
- Great for breaking up a monolith

```
$ npm install fastclick --save
```

package.json 

```
{
  "name": "ft-app",
  "dependencies": {
    "fastclick": "^1.0.3",
      [...]
  }
}
```

someModule.js 

```
var fc = require("fastclick");
/**
Module code
**/
```

It’s not perfect (yet…)

- Git tags don’t guarantee repeatability
  - Use npm-shrinkwrap
- Registry introduces a single point of failure
  - We use a private lazy cache

CSS isn’t encapsulated either…

Leak-proof styling for reusable components

- Context agnostic
- Non-semantic naming – clear that it's reusable
- Classes prefixed with component name

Code example from smashingmagazine.com/2013/05/23/building-the-new-financial-times-web-app-a-case-study/

```
<div class="apple">
    <h2 class="apple_headline">…
    <h3 class="apple_subhead">…
    <div class="apple_body">…
</div>
```

```
.apple {}
.apple_headline {
    font-size: 40px;
}
.apple_subhead {
    font-size: 20px;
}
.apple_body {
    font-size: 14px;
    column-count: 2;
    color: #333
}
```

Works for one app – what about sharing components across an entire organisation?

FT Origami

Google™ Custom Search

SPEC REGISTRY

- **Overview**
  - Non-technical explainer
  - Statement of principles
  - Governance and policy
  - Roadmap and current activity
  - Spec changes newsfeed
- Component spec
- Syntax standards
- Developer guide
- Third party component A List

Origami is about empowering developers **of all levels** to build **robust**, **on-brand** products ranging from simple static sites through to rich, dynamic web applications, to do it **faster**, to do it **cheaper**, and leave them more **supportable** and more **maintainable**.

origami.ft.com

The future – fully encapsulated web components?

Performance

Maintaining smooth animations and a responsive UI

Perils of a single thread

Long running processes block user interactions

Perils of a single thread

Missed frames make animations, scrolling and swiping feel “janky”

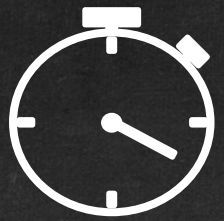
Synchronous tasks also block screen updates

6 frames per second

Great for animated gifs

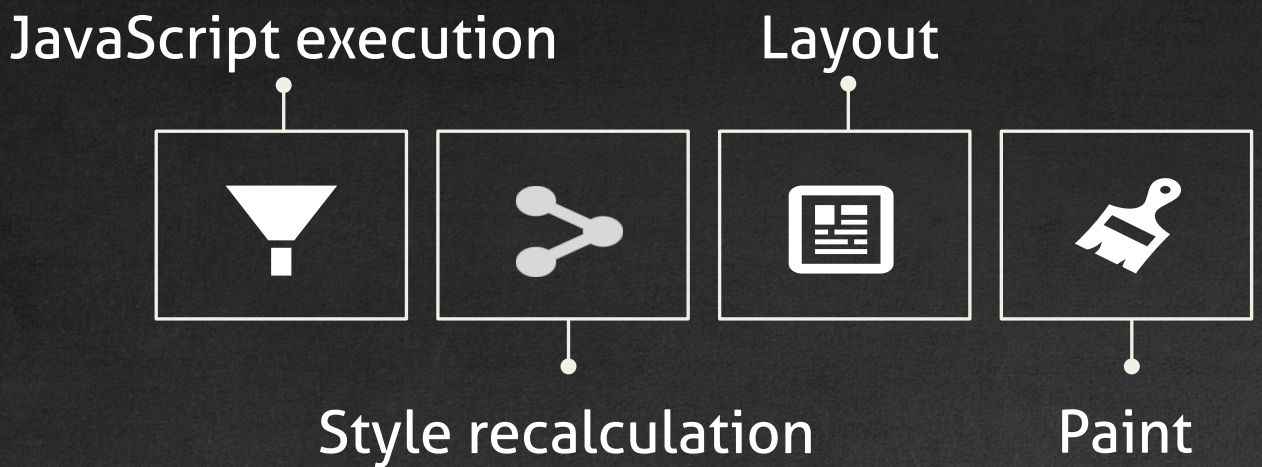
Rubbish for your app

Consistent frame rate

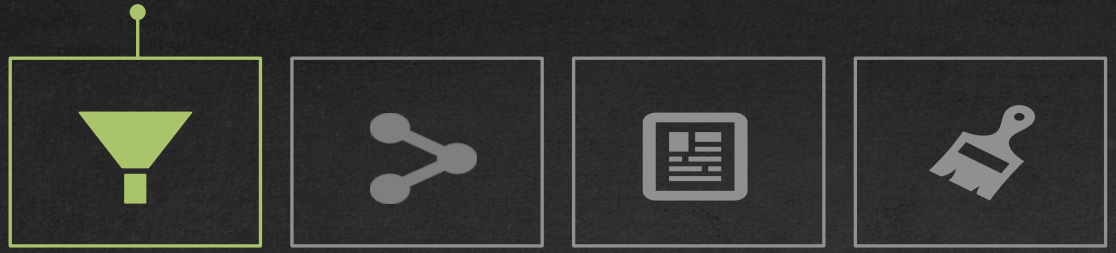


We want something that's silky smooth, so we aim for 60 frames per second. This gives us just 16.6ms between frames

What happens on this thread?

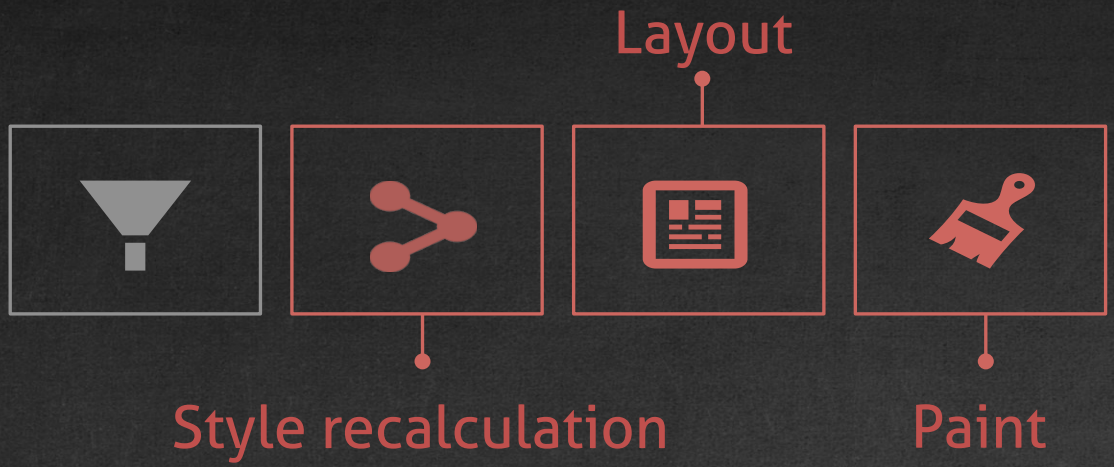


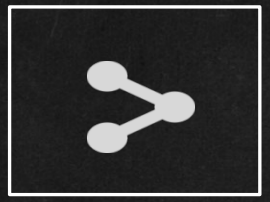
JavaScript execution



JavaScript execution is rarely the bottleneck

Which means we need to understand the other operations taking place (sorry!)





Style recalculation

Working out what things should look like from CSS & DOM

Layout

Working out what goes where on the screen

Paint

Putting the pixels onto the screen*

* Technically this is the browser painting to a bitmap and then uploading to the GPU rather than putting pixels directly on the screen

Use animation effects which avoid relayout/paint

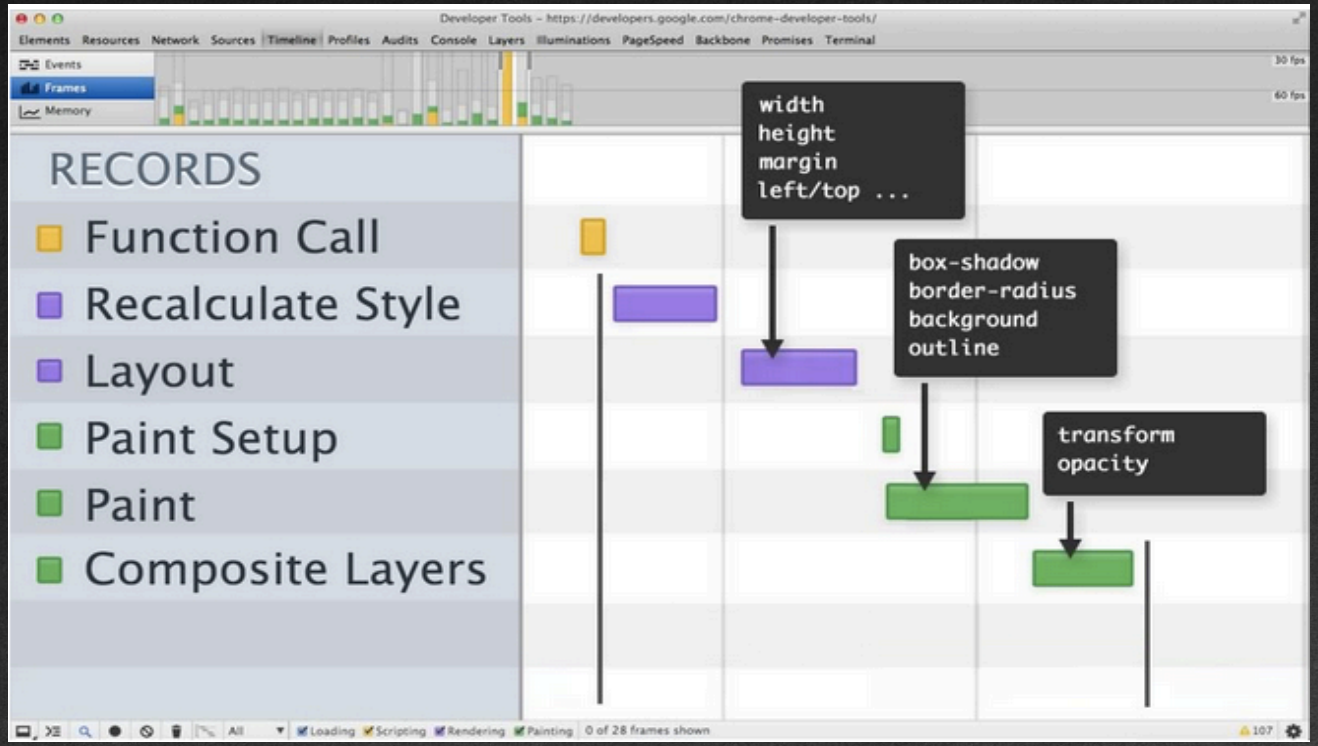
- transforms:
  - translate
  - scale
  - rotation
- opacity
- These use the GPU, which is optimised for just such a task

```
// Position/margin slow
style.top = x;
style.marginLeft = x;

// translate & translate3d fast
style[transform] =
“translate(“ + x + “px,“ + x +
“px)”;

/** may need to use “translateZ
hack” to manually force layer
creation **/
style[transform] =
“translateZ(0)”
```

If you can't eliminate, reduce time spent doing these tasks

Time for some detective work...

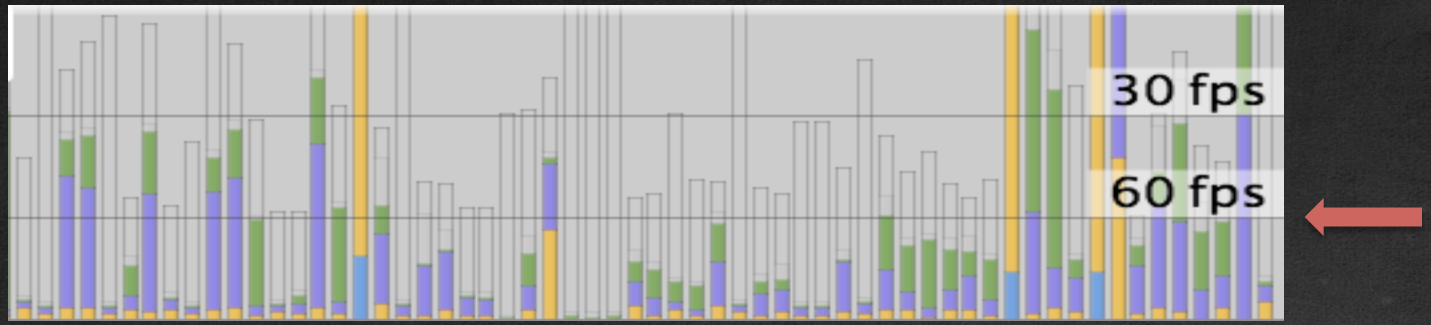
Timeline – shows us time spent in JS execution, layout and paint



From: http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

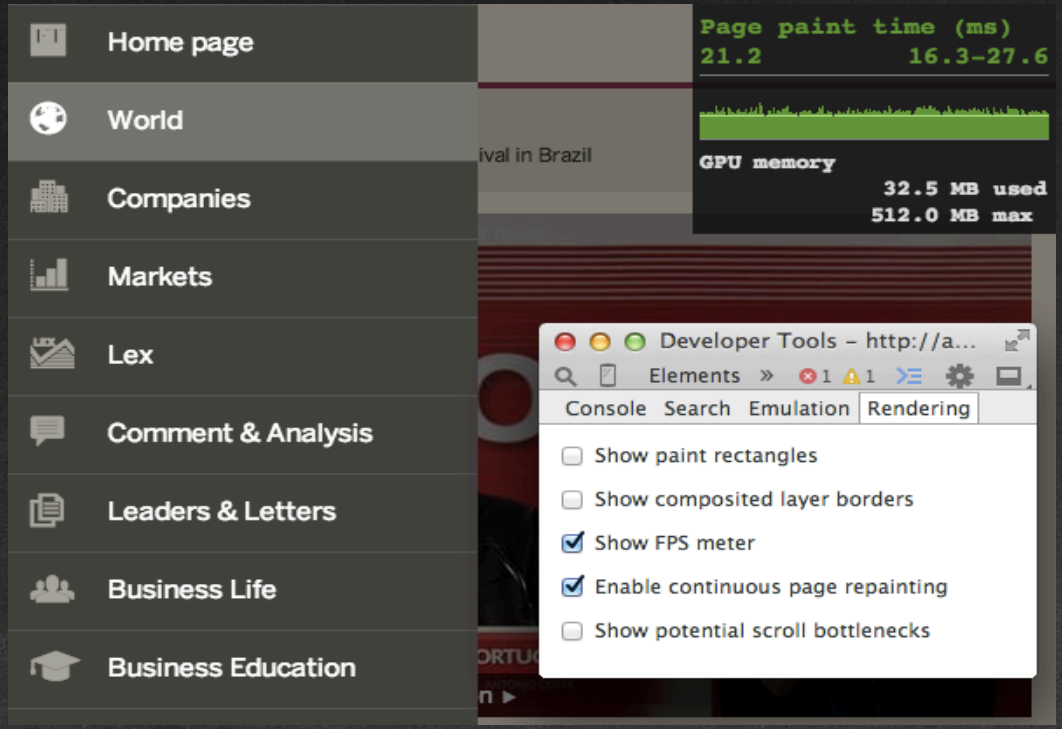
Timeline “frames view” shows amount of work required to render each frame

Taller bars = slower



We want all our frames below the 60 FPS line

Let's see how much time the entire page would take to paint...

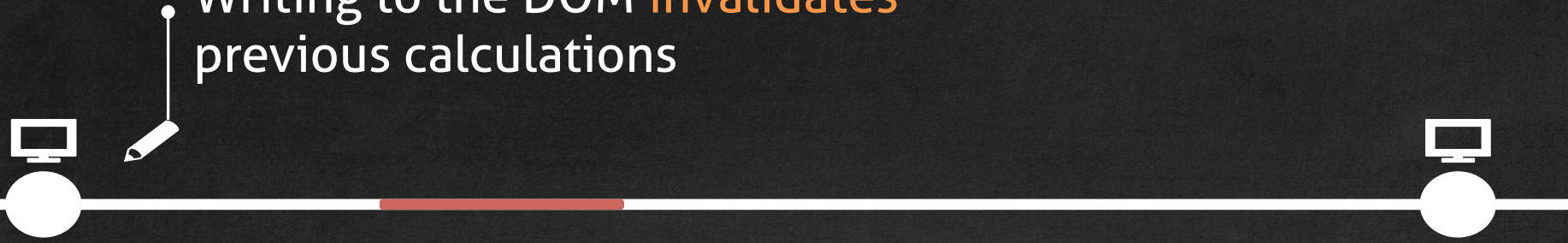


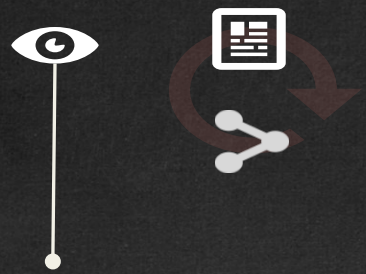
Keep your painting simple...

- Hide elements to see what impact they make on page paint time
- Common suspects: lots of box-shadow and border-radius

Reducing relayout

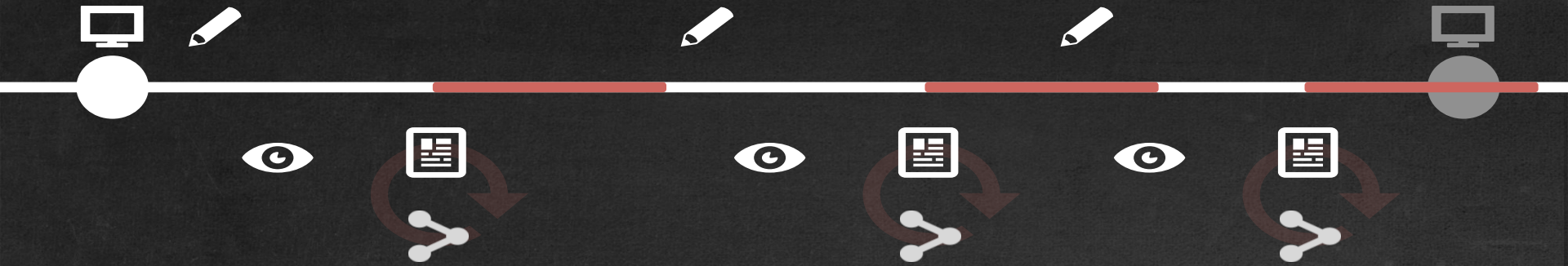
Writing to the DOM invalidates previous calculations





Reading a geometric value from the DOM once it has been invalidated forces a relayout

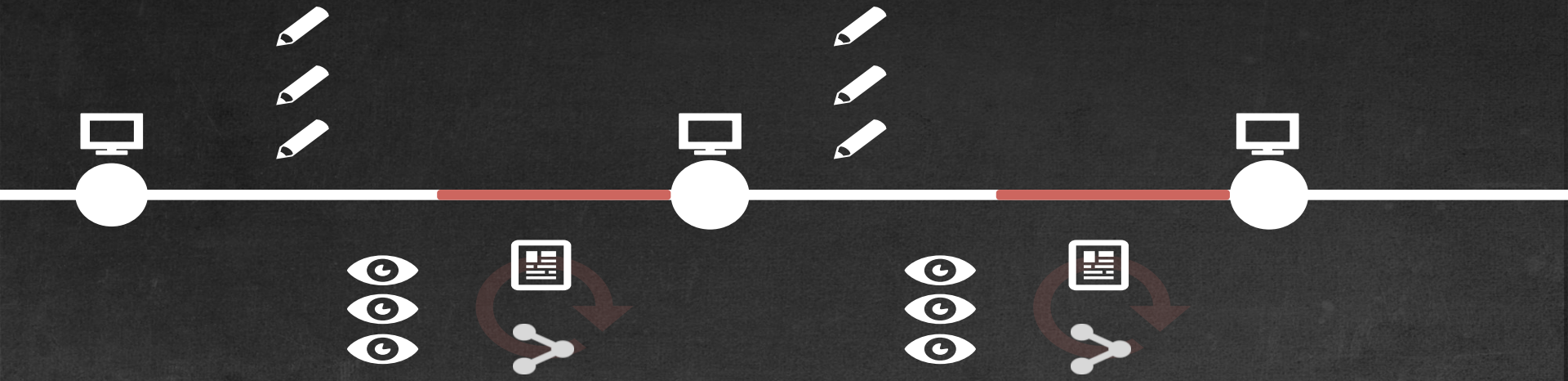
Reducing relayout



Doing this repeatedly prevents the browser from being able to render a frame, resulting in dropped frames

Batch DOM read/writes

Instead we can queue these reads and writes together, and execute them once per frame



This can be hard to do manually, especially with lots of components, but we can manage it with a library:

 wilsonpage/fastdom

Putting it all together:
Swiping on app.ft.com

ftlabs/ftscroller

More

- Videos
  - Debugging CSS & Render Performance
    - https://developers.google.com/events/io/sessions/324511365
- Lots of good tutorials/blogs
  - html5rocks.com/en/features/performance
  - jankfree.org/
  - Paul Lewis: aerotwist.com/

Offline

Client-side storage options and their limitations

Client Side Storage Options

- Cookies
- AppCache (flawed, but usable)
- LocalStorage (fast, but synchronous)
- IndexedDB (async, but tricky to use)

AppCache ☹

Cookies
AppCache
LocalStorage
IndexedDB

- Well intentioned, but flawed
- However, it *is* usable
  - We use it for bare minimum: bootstrap code, fonts, splash screen images

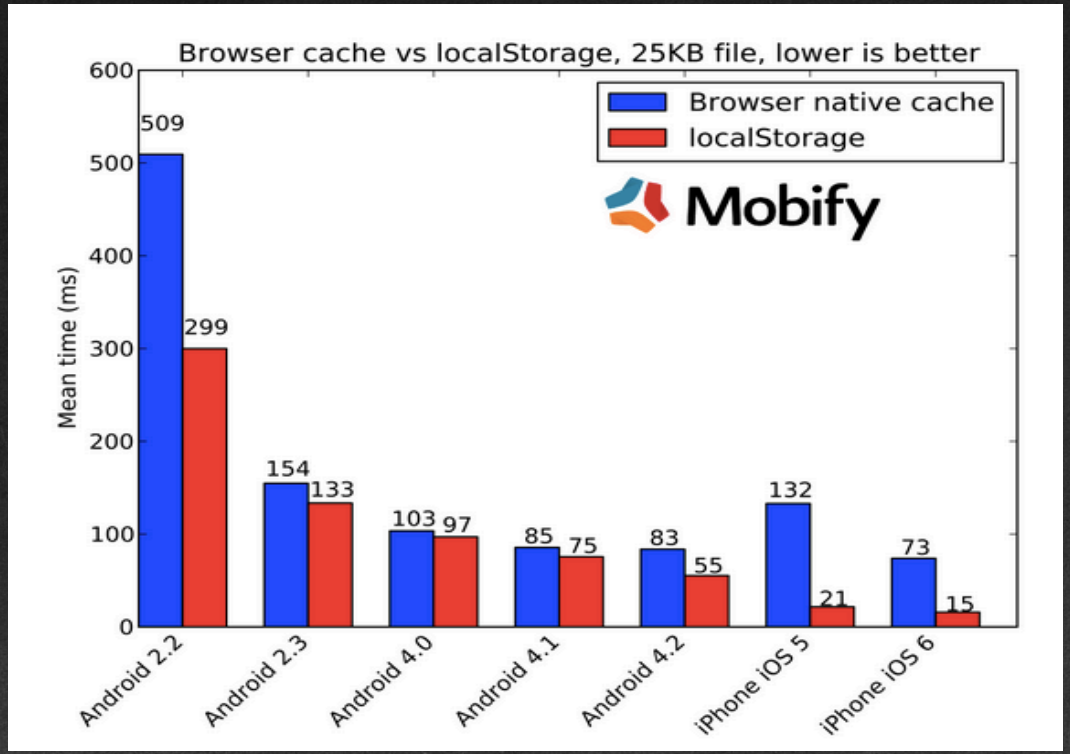
LocalStorage 

- Simple API
- Fast…?

Cookies
AppCache
LocalStorage
IndexedDB

Faster than cache...

- Cookies
- AppCache
- LocalStorage
- IndexedDB



http://www.mobify.com/blog/smartphone-localstorage-outperforms-browser-cache/

LocalStorage

Cookies
AppCache
LocalStorage
IndexedDB

- But:
  - Limited storage
  - Synchronous
    - File I/O for persistence means it can have variable performance
  - Odd behaviour in Safari private browsing
  - We use a lightweight wrapper called Superstore by Matt Andrews

matthew-andrews/superstore

IndexedDB 😐

- Async key value object store
  - We use this for articles and images
- Not supported everywhere - use polyfill [1] to support (long deprecated) WebSQL
- Managing versions and migrations can be awkward
- Documentation is variable

Cookies
AppCache
LocalStorage
IndexedDB

[1] http://nparashuram.com/IndexedDBShim/ or https://github.com/mozilla/localForage

Future: ServiceWorker

- Sits in the middle of browser and network
- Lots of good things:
  - Extensible w/ low level, granular control
  - “Cache API” for storage
  - Async
- But:
  - No access to localStorage
  - HTTPS only

More

- Tutorial: Building an offline web app
  labs.ft.com/2012/08/basic-offline-html5-web-app/
- Storage quotas:
  html5rocks.com/en/tutorials/offline/quota-research
- Maximising storage by using UTF-8 instead of UTF-16:
  labs.ft.com/2012/06/text-re-encoding-for-optimising-storage-capacity-in-the-browser/
- Using ServiceWorker today:
  jakearchibald.com/2014/using-serviceworker-today/

Summary

A quick recap....

- Modularisation
  - npm + browserify works well for managing client side JS dependencies
  - Prefixed class names for CSS component elements
- Performance
  - Good tools, profile your own use case. Look out for relayout and paint as bottlenecks – batch DOM read/writes and stick to known fast animations
- Offline
  - Hard with limited options, prefer async IndexedDB, look out for ServiceWorker

Thanks!☻

 grahamhinchly

 graham.hinchly@ft.com

 labs.ft.com/jobs

 ftlabs | financial-times