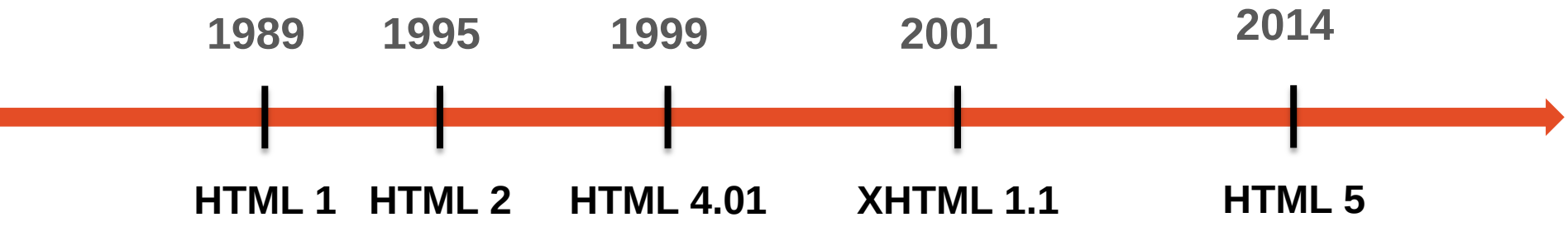
HTML

I've seen the
FUTURE
It's in my
BROWSER

coding_
academ

What is it?

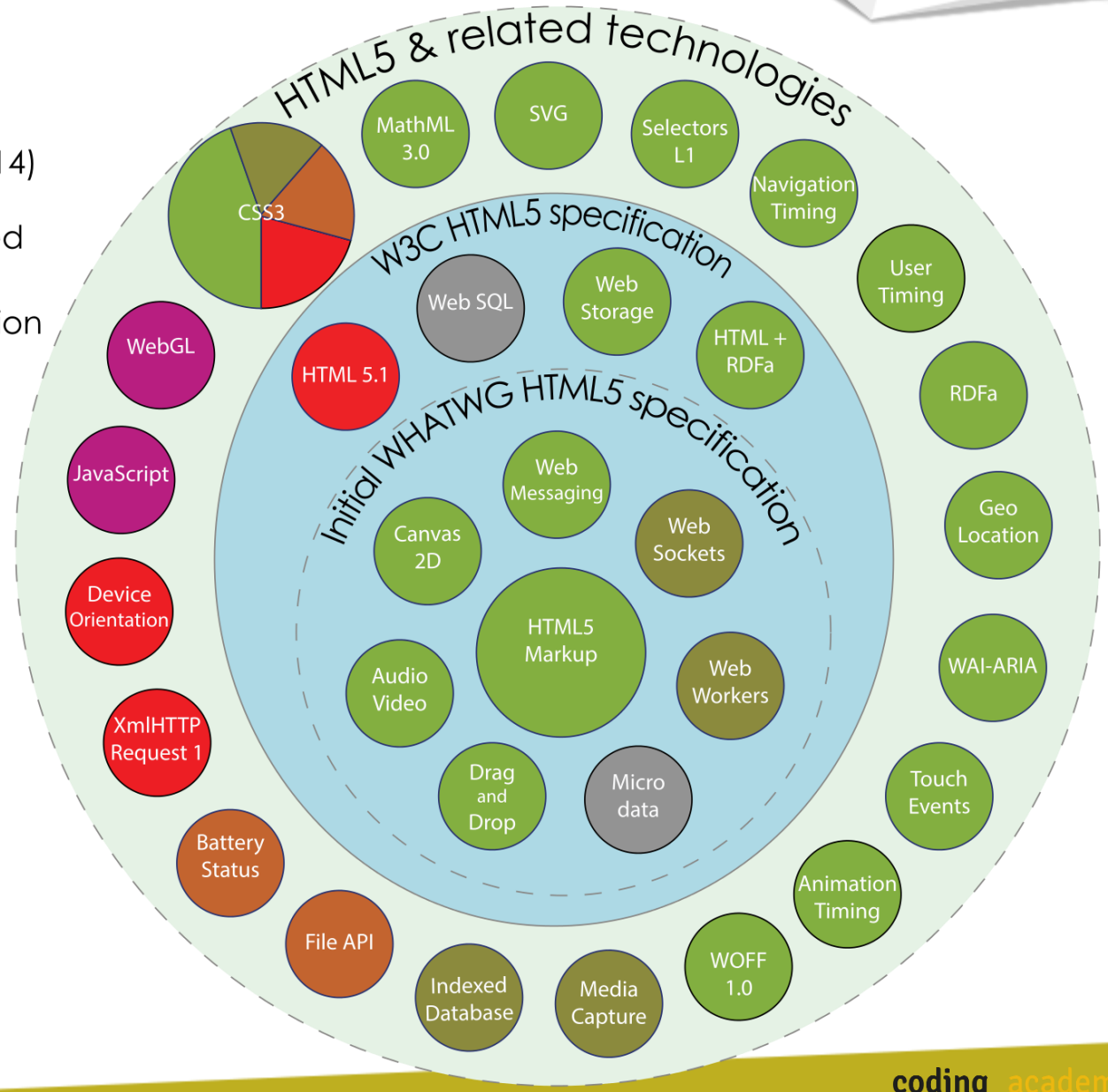
HTML5 is a markup language used for structuring and presenting content on the World Wide Web.



HTML5

Taxonomy & Status (October 2014)

- Recommendation/Proposed
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated or inactive



Semantic Elements

- <section> - a section in a document
- <article> - independent, self-contained content
- <nav> - a section with navigation links
- <main> - the main content
- <aside> - content aside from the content it is placed in
- <header> - container for introductory content
- <footer> - defines a footer for a document or section
- <time> - a human-readable date/time

Semantic Elements

- <figure>, <figcaption> - self-contained content, like illustrations, diagrams
- <mark> - highlighted text
- <summary> - visible heading for *<details>*
- <details> - additional details

Form Elements and Attributes

Input types

- tel – phone numbers
- search – search bars
- email – email addresses
- range – range of numbers
- color – color picker
- date – picker (datetime TZ, date-local, week, time, month)
- time – time picker
- file – upload **multiple** files
- url - links

Form Elements and Attributes

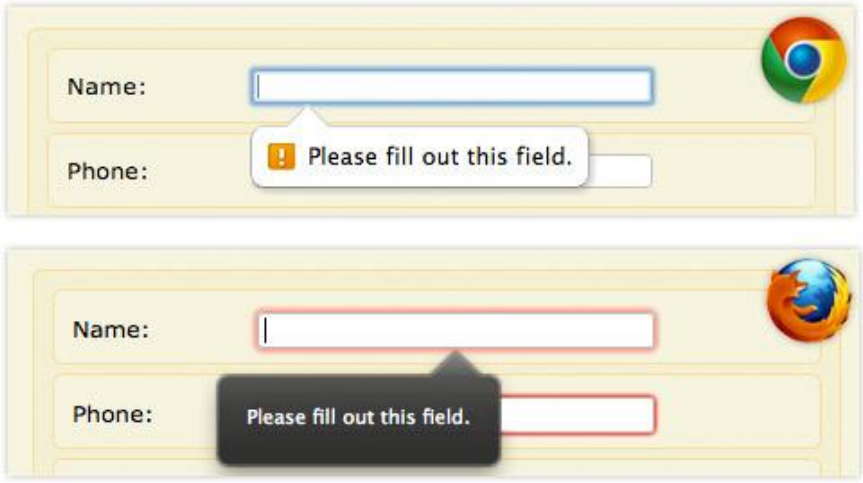
Attributes

- placeholder – sets place holder string for input elements
- data-* – custom element attributes
- autofocus – focus control during page load
- autocomplete (on / off) – stores input for future use
- multiple – allows multiple values in file / email inputs
- list – offers predefined values but accepts free text
- form – allows placing an input **outside** the form

Form Elements and Attributes

Validations

- required – valueMissing
- type – typeMismatch
- pattern – patternMismatch
- maxLength – tooLong
- min – rangeUnderflow
- max – rangeOverflow
- step – stepMismatch
- setCustomValidity(message) – customError



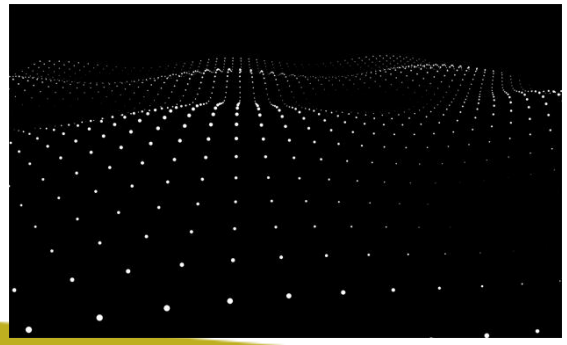
- Appearance can also be modified - (:invalid, :required)
- Turning form validation off
  - novalidate form attribute, Empty value formonvalidate submit button attribute

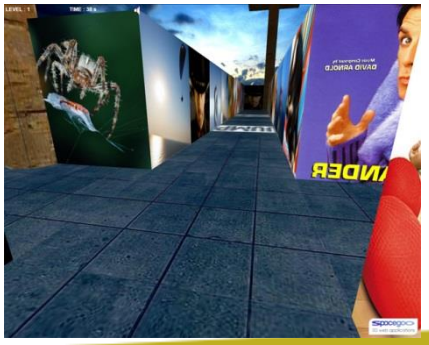
Canvas

Key features

- Web pages container for 2D graphics
- Pixel-by-pixel rendering resolution dependent
- Manipulated via scripting (Javascript)
- Content NOT in DOM – its just a bitmap

```
<canvas id="myCanvas" width="500" height="400">
    Update your browser to enjoy canvas!
</canvas>
```





SVG

SVG is an image format for vector graphics

Why to use it?

- Small file sizes that compress well
- Scales to any size without losing clarity (except very tiny)
- Looks great on retina displays
- Design control like interactivity and filters
- Here are some simple examples
- And a great article here!

```
<svg width="400" height="110">
   <rect width="300" height="100" style="fill:rgb(0,0,255);
         stroke-width:3;stroke:rgb(0,0,0)" />
   This browser does not support inline SVG
</svg>
```

SVG

SVG is an image format for vector graphics

Why to use it?

- Small file sizes that compress well
- Scales to any size without losing clarity (except very tiny)
- Looks great on retina displays
- Design control like interactivity and filters
- Here are some simple examples
- And a great article here!

```
<svg width="400" height="110">
   <rect width="300" height="100" style="fill:rgb(0,0,255);
         stroke-width:3;stroke:rgb(0,0,0)" />
   This browser does not support inline SVG
</svg>
```

Canvas vs SVG

Canvas	SVG
Pixel based (similar to image element)	Vector based (similar to Flash technology)
Single HTML element (the whole canvas)	Multiple graphical elements which become part of the Document Object Model (DOM)
Modified through **script** only	Modified through **script** and **CSS**
API does **NOT** support accessibility	SVG markup and object model directly supports accessibility
Resolution dependent	Resolution independent
No API for animation (Besides Repaint)	Good support for animations via scripting & CSS animations

Video & Audio

- Built in video/audio support
  - Codecs and fallbacks

```
<video  width="600" height="400" controls autoplay loop onclick="toggleVideo();">
    <source src="video/big_vid.mp4“ media="(min-device-width: 1024px)" />
    <source src="video/small_vid.mp4“ media="(max-device-width: 600px)" />
    <source src="video/normal_vid.mp4" />

   <source src="video/vid.ogv" />
   <source src="video/vid.webm" />
   <object type="application/x-shockwave-flash" width="600" height="400">
      <param name="movie" value="http://vimeo.com/moogaloop.swf?clip_id=1084537" />
   </object>
</video>
```

Video & Audio

JS API, such as:

Method	Description
load()	Re-loads the audio/video element
play()	Starts playing the audio/video
pause()	Pauses the currently playing audio/video
volume	Sets or returns the volume of the audio/video

Content Editable

- Turns an element into an editable area
- Can be applied to almost any element
- Can be applied even to <style> element
- Save / undo changes can be done by JavaScript

```
<p contenteditable>
    Some Content to Edit
</p>
```

Geo Location

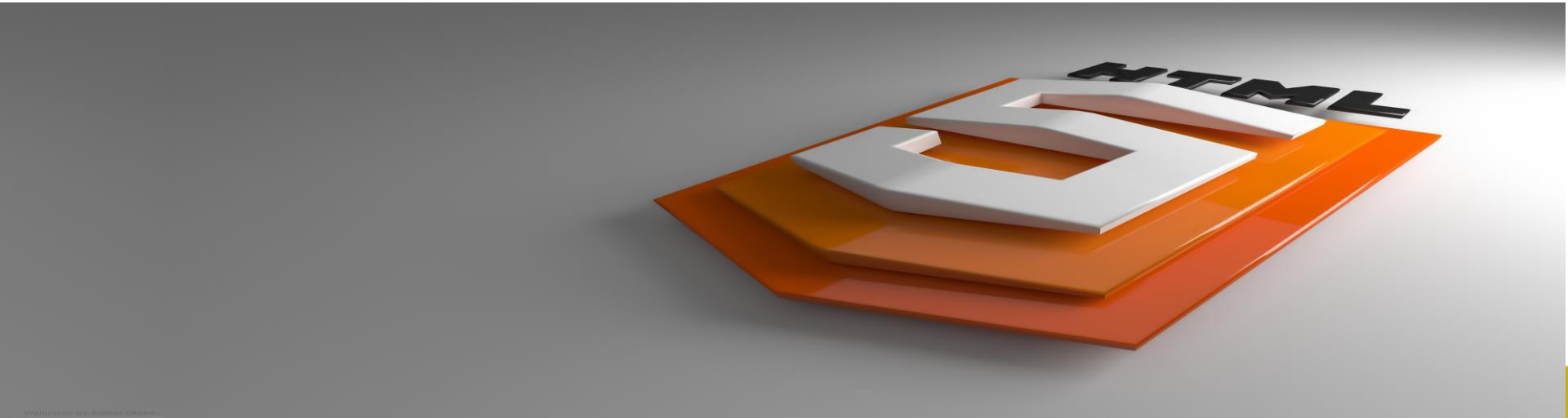
navigator.**geolocation**.getCurrentPosition(*showLocation*, *handleLocationError*);

Geo Location

- **IP based**: old-fashion, inaccurate & unreliable but always available
- **GPS**: Very accurate but is only available on open spaces
- **Wi-Fi based**: Very accurate and available even in buildings but requires several Wi-Fi spots
- **Cellphone**: Location is set by its distance from cellular antennas

HTML5 Offline

- More people are experiencing the **web in motion**
- This increase the likelihood of **connectivity issues**
- We should build web apps that give users a **continuous experience**
  - Caching API: HTML, CSS, JS & images
  - Web Storage
  - Service Workers!



Web Storage

localStorage & sessionStorage

- Store string typed key-value pairs on client side
- Unlike cookies, data is not sent to server
- ~5MB per domain (4KB per cookie)
- sessionStorage: short-term, tab (window)
- localStorage: long-term, browser
- Doesn’t work in private-mode
- Don’t use for sensitive data store!

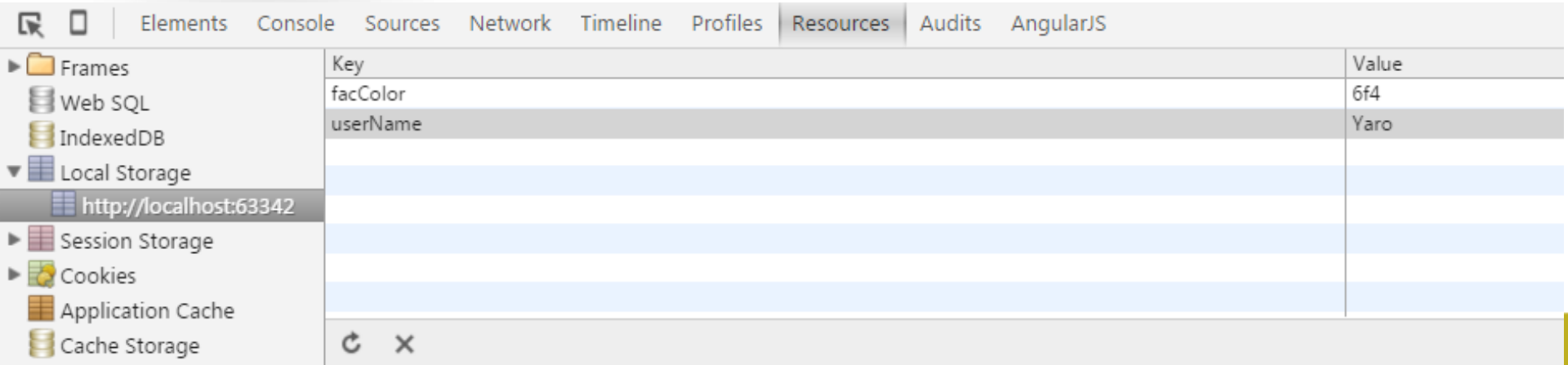
```
localStorage.setItem(“favColor", favColor);
localStorage.favColor = favColor;

var favColor = localStorage.getItem("favColor");
var favColor = localStorage.favColor;
```

Web Storage

Tips

- surround with try…catch block to catch QUOTA_EXCEEDED_ERR
- use JSON.parse() / .stringify() for anything (e.g. booleans)
- Use browsers debugging tools

```
try {
  localStorage.setItem("key", "some data");
} catch (e) {
  if (e == QUOTA_EXCEEDED_ERR) {
    // do something nice to notify your users
  }
}
```



Web Storage – Indexed DB

Key Concept

- NOT a Relational Database
- API is mostly asynchronous
- Store key-value pairs: The value is an object
- Enforces the same origin constraint
- Does not use SQL: uses queries on an index that produces a cursor, which you use to iterate across the result set.

- Talking is Cheep, Show me the Code.

Messaging API

API for posting messages

Iframe Messaging

- Target origin is required and must match the origin of the iframe; Otherwise a security error will be thrown
- Wait for the target to finish loading

```
// Page 1 :
var target = document.getElementById("iframe");
target.contentWindow.postMessage("Wassap?",  "http://www.target.com");

// Page 2 :
window.addEventListener("message", function(e) {
        if (e.origin == "http://www.source.com")
            e.source.postMessage("Roger that!", e.origin);
});
```

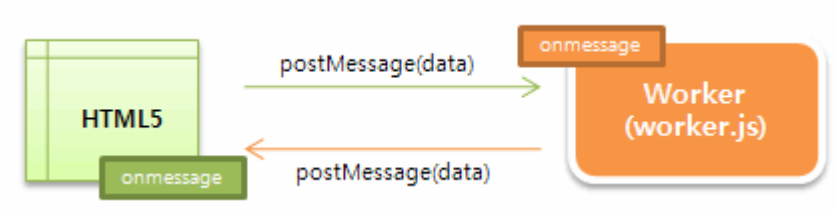
Web Workers

- Problem: When executing scripts in an HTML page, the page becomes unresponsive until the script is finished

- Solution: Web Workers provide a simple means for web content to run scripts in background threads.

  The worker thread can perform tasks without interfering with the user interface.

  Once created, a worker can send messages to the JavaScript code that created it by posting messages to an event handler specified by that code (and vice versa.)



Web Workers

- A worker is an object created using a constructor (e.g. Worker()) that runs a named JavaScript file — this file contains the code that will run in the worker thread
- Workers have no access to the window or the document

```
// Create a new worker object
var worker = new Worker("primesWorker.js");

// Register to the onmessage event of the worker, that will read the prime number
// sent from the worker and add it to the listbox
worker.onmessage = function (e) {
    var lstPrimes = document.getElementById("lstPrimes");
    var opt = document.createElement("option");
    opt.innerHTML = e.data;
    lstPrimes.appendChild(opt);
}
```

Web Workers

- Thread safety: web workers have carefully controlled communication points with other threads - no concurrency problems.
- Data passed between the main page and workers is **copied**, not shared, Objects are serialized / deserialized
- Worker threads have access to a global function, importScripts()
  - Load external script: *importScripts(file1.js‘[, ‘file.js‘, …]);*
  - The browser loads each listed script (asynch) and executes it by the specified order. Any global objects from each script may then be used by the worker.
  - importScripts() does not return until all the scripts have been loaded and executed.
- Catch worker errors: *worker.addEventListener('error', onError, false);*
- Terminate the worker:
  - Main page: *worker.terminate();*
  - Worker script: *self.close();*

Web Workers - SharedWorker

- A shared worker is accessible by multiple scripts — even if they are being accessed by different windows, iframes or even workers.
- One big difference is that with a shared worker you have to communicate via a port object — an explicit port is opened that the scripts can use to communicate with the worker

```
var myWorker = new SharedWorker("SharedWorker.js");
myWorker.port.start();

// SharedWorker.js:
onconnect = function(e) {
    var port = e.ports[0];
    port.onmessage = function(e) {
        var workerResult = 'Multiply Result: ' + (e.data[0] * e.data[1]);
        port.postMessage(workerResult);
    }
    port.start();  // not necessary since onmessage event handler is being used
}
```

History API

- **Push / pop** history items to the browser
- **Attach data** to each entry
- **Manipulate** URL in browser's location bar without refresh
- **Listen** to browser's **Back** button

```
window.history.go(-1);
window.history.back();

window.history.go(1);
window.history.forward();

// Suppose http://misterbit.me/foo.html executes the following JS:
var stateObj = { prop: "value" };
history.pushState(stateObj, "State title, currently ignored", "puk.html");
// This will cause the URL bar to display http://misterbit.me/puk.html
// but WON'T cause the browser to load puk.html or even check that it exists.

// Suppose now that the user now navigates to http://google.com, then clicks back.
// At this point, the URL bar will display http://misterbit.me/puk.html,
// and the page will get a popstate event whose state object contains a copy of stateObj.
// The page itself will look like foo.html, although the page might modify its contents during the popstate event.

// If we click back again, the URL will change to http://misterbit.me/foo.html,
// and the document will get another popstate event, this time with a null state object.
// Here too, going back doesn't change the document's contents from what they were in the previous step,
// although the document might update its contents manually upon receiving the popstate event.
```

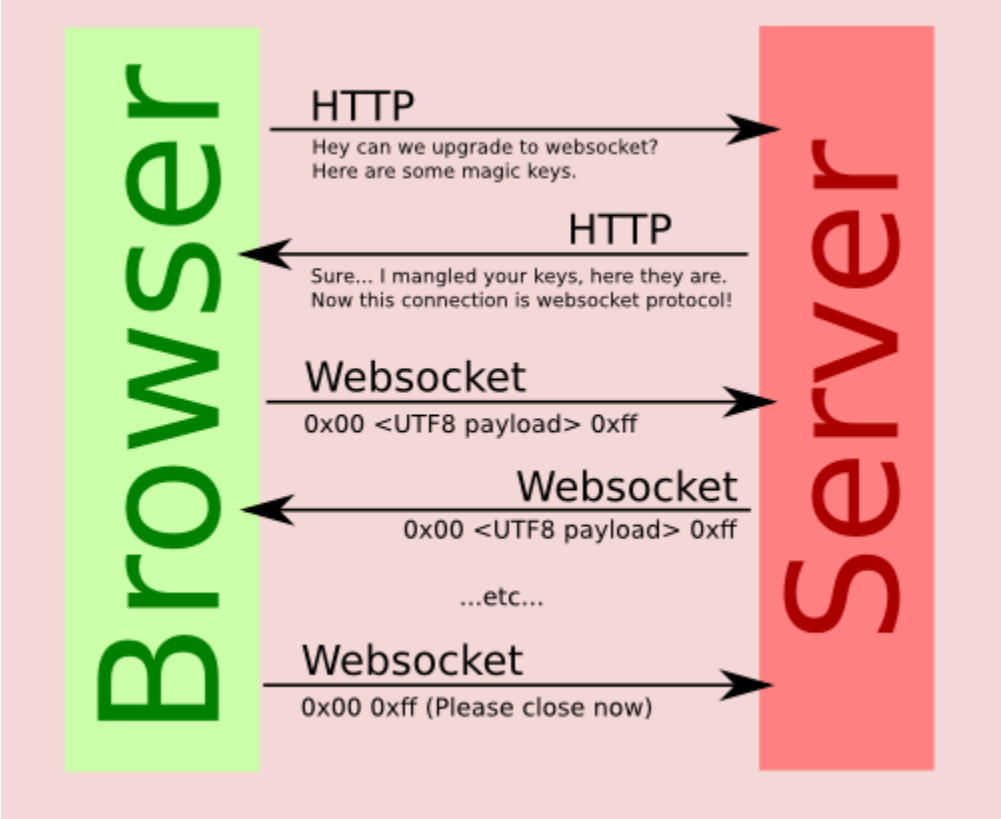
Web Sockets

an API establishing "socket" connections between a web browser and a server. This enables a “Push” technique.

Check it out here:
http://www.html5rocks.com/en/tutorials/websockets/basics/

Basic Echo Test:

http://www.websocket.org/echo.html



Summary

- HTML5 gives us some new super powers
- Enjoy it!

- Next Steps:
  - Go Build an awesome web application

Questions?


MisterBit
misterBIT.co.il
Keep on diving
oin us:
eetup
The Future is made in Javascript