INTERNET SYSTEMS

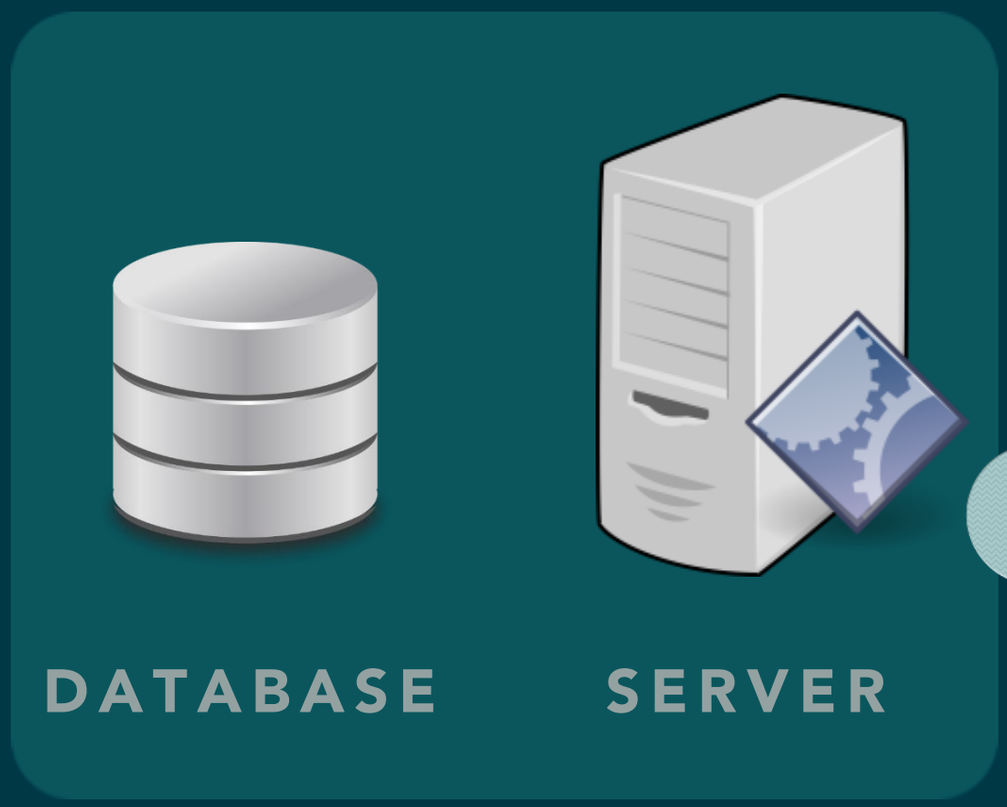
HTML 5

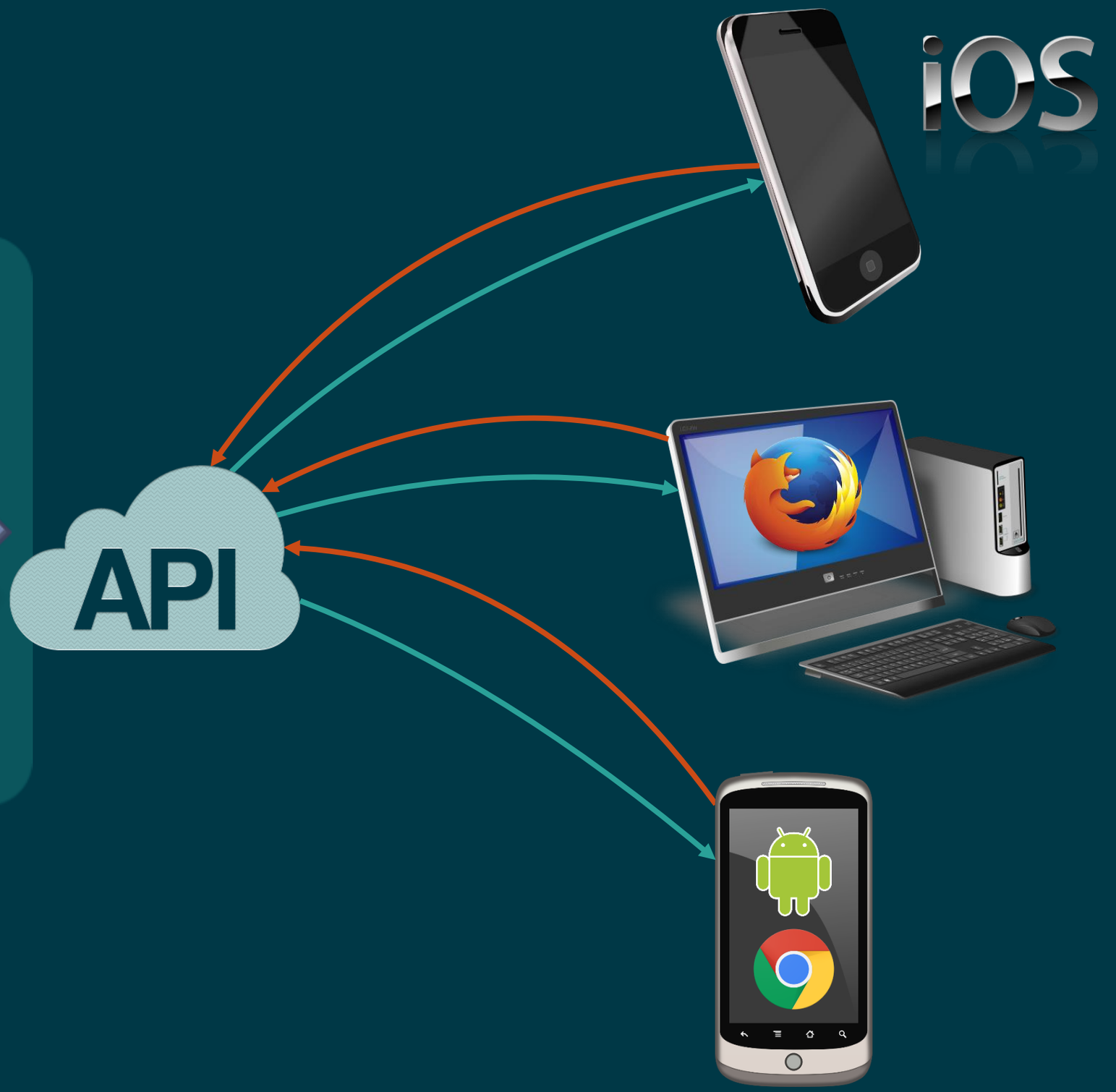
TOMASZ PAWLAK, PHD
MARCIN SZUBERT, PHD
POZNAN UNIVERSITY OF TECHNOLOGY, INSTITUTE OF COMPUTING SCIENCE

PRESENTATION OUTLINE

- History and Motivation
- HTML5 basics
- HTML5 features
  - Semantics
  - Connectivity
  - Offline & Storage
  - Multimedia
  - 2D/3D Graphics & Effects
  - Performance & Integration
  - Device Access
  - Styling

MODERN WEB APPLICATION





HISTORICAL PERSPECTIVE

- 1991 — *HTML Tags*, an informal **CERN** document
- 1993 — *HTML Internet Draft* published by the **IETF**
- 1995 — *HTML 2.0* (RFC 1866) published by the **IETF**
- 1997 — *HTML 3.2* published as a **W3C** Recommendation
- 1997 — *HTML 4.0* published as a **W3C** Recommendation:
  - **Transitional**, which allowed for deprecated elements
  - **Strict**, which forbids deprecated elements
  - **Frameset**, which allowed embedding multiple documents using frames
- 1998 — W3C decided to **stop evolving HTML** and instead begin work on an XML-based equivalent, called **XHTML**

HISTORICAL PERSPECTIVE — XHTML

- 2000 — *XHTML 1.0* published as W3C Recommendation:
  - reformulation of HTML 4 as an application of XML 1.0, offering **stricter rules** for writing and parsing markup: lower case tags, end tags for all elements, quoting attributes, escaping ampersands
  - new MIME type `application/xhtml+xml` enforces **draconian error handling** in web browsers.
    - combatibility guidelines: allowed serving pages as HTML (`text/html`) to continue using **forgiving error handling** in HTML parsers.
- 2002-2006 — W3C released working drafts of XHTML 2.0 which **break backward compatibility**.
- 2009 — W3C abandoned the work on XHTML 2.0.

HISTORICAL PERSPECTIVE — WHATWG

- 2004 — Opera, Mozilla and Apple formed Web Hypertext Application Technology Working Group (WHATWG):

  The group aims to develop specifications based on HTML and related technologies to ease the deployment of interoperable ***Web Applications*** *[...] for implementation in* ***mass-market Web browsers****, in particular Safari, Mozilla, and Opera; [the group] intends to ensure that all its specifications address* ***backwards compatibility*** *concerns [...] and specify* ***error handling behavior to ensure interoperability*** *even in the face of documents that do not comply to the letter of the specifications.*

  —IAN HICKSON, ON BEHALF OF THE WHATWG MEMBERS

- 2006 — W3C created HTML Working Group to participate in the development of the HTML5 specification:

  Some things are clearer with hindsight of several years. It is necessary to evolve ***HTML incrementally****. The attempt to get the world to* ***switch to XML****, including quotes around attribute values and slashes in empty tags and namespaces all at once* ***didn't work****... The plan is to charter a completely new HTML group.*

  —TIM BERNERS-LEE

WHATWG PRINCIPLES

- **Backwards compatibility** — web application technologies should be based on technologies authors are familiar with, including HTML, CSS, DOM, and JavaScript.

- **Well-defined error handling** — error handling in Web applications must be defined to a level of detail where User Agents do not have to invent their own error handling mechanisms or **reverse engineer** other User Agents'.

- **Users should not be exposed to authoring errors** — specifications must specify exact error recovery behaviour for each possible error scenario; error handling should for the most part be defined in terms of graceful error recovery, rather than obvious and catastrophic failure (as in XML).

WHATWG PRINCIPLES

- **Practical use** — every feature that goes into the Web Applications specifications must be justified by a practical use case; the reverse is not necessarily true.

- **Scripting is here to stay** — but should be avoided where more convenient declarative markup can be used. Scripting should be device and presentation neutral.

- **Device-specific profiling should be avoided** — authors should be able to depend on the same features being implemented in desktop and mobile versions of the same UA.

- **Open process** — the Web has benefited from being developed in an open environment. Web Applications will be core to the web, and its development should also take place in the open.

HISTORICAL PERSPECTIVE — HTML5

- 2008 — *First Public Working Draft* of HTML5 for:
  - **web developers** — how to use new features, write correct documents and avoid bad habits.
  - **browser makers** — how to parse HTML, ensure backward compatibility, handle errors or obsolete elements.
- 2011 — W3C and WHATWG diverged:
  - WHATWG updates the *Living Standard* — whatwg.org/html
  - W3C uses traditional versioning — http://www.w3.org/TR/html/
- 2014 — HTML 5.0 specification released as a stable W3C Recommendation
- 2016 & 2017 — HTML 5.1 & 5.2 released as W3C Recommendation
- HTML 5.3 is under development as of 2020

FIVE THINGS YOU SHOULD KNOW ABOUT HTML5

1. *It's not one big thing — it is a collection of individual features.*

2. *You don't need to throw anything away — applications that worked yesterday in HTML 4 will still work today in HTML5.*

3. *It's easy to get started — upgrading to HTML5 can be as simple as changing your doctype.*

4. *It already works — HTML5 is already well-supported by most modern browsers.*

5. *It's here to stay — HTML5 is the future of web standards.*

—DIVE INTO HTML5, MARK PILGRIM

HTML5 IS NOT ONE BIG THING

- The term **HTML5** represents two different concepts:
  - a new version of the language HTML, with new elements, attributes, and behaviors;
  - a larger set of technologies that allows more diverse and powerful Web sites and applications. This set is sometimes called ***HTML5 & friends*** and often shortened to just HTML5.

- HTML5 is basically an attempt to evolve the Web to meet the demands of the way we use it today — Open Web Platform, New Exciting Web Technologies.

HTML5 FEATURES

Semantics — allows to describe more precisely what the content is.

Connectivity — allows to communicate with the server in innovative and efficient ways.

Offline & Storage — allows webpages to store data on the client-side and operate offline.

Multimedia — making video and audio first-class citizens in the web.

HTML5 FEATURES

2D/3D Graphics & Effects — allows a much more diverse range of presentation options.

Performance & Integration — providing speed optimization and better usage of hardware.

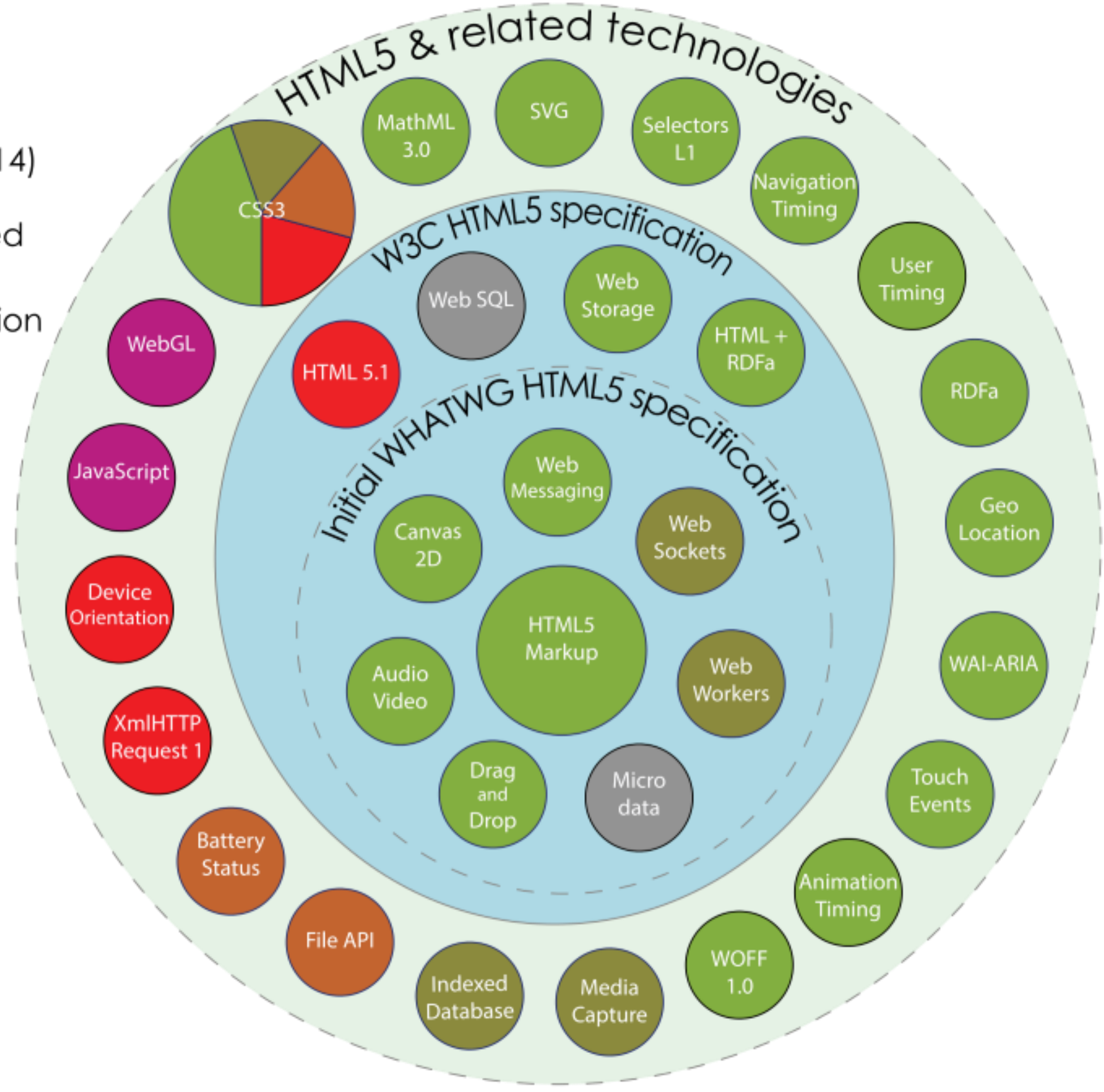
Device Access — allows the usage of various input and output devices.

Styling — allows authors to write more sophisticated themes.

HTML5

Taxonomy & Status (October 2014)

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



HTML5 IS EASY TO GET STARTED

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Example document</title>
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>
```

DOCTYPE SWITCHING

- The Document Type Definition is used for two things:
  - Web **browsers** use it to determine which rendering mode they should use (quirks, standard, almost standard).
  - Markup **validators** look at the doctype to determine which rules they should check the document against.

- New doctype `<!DOCTYPE html>`
  - triggers **standards** mode in all current and relevant legacy browsers.
  - intentionally contains no language version identifier so it will remain usable for all future revisions of HTML
  - is short and memorable to encourage its use.

- Upgrading to the HTML5 doctype won't break your existing markup, because obsolete elements will still render in HTML5; but it will allow you to use and validate new elements.

CHARACTER ENCODING

- Web developers are **required to declare** the **character encoding**. There are three ways to do that:
  - at the **transport level**; for instance, by using the HTTP header
  - using a **Unicode Byte Order Mark** (BOM) character at the start of the file. This character provides a signature for the encoding used.
  - using a **meta element** with a charset attribute that specifies the encoding, for instance: `<meta charset="UTF-8">` replaces the older syntax:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

GOOD HTML5 STYLE

- Including the optional **<html>**, **<body>**, and **<head>** elements. The **<html>** element is a handy place to define the page's **natural language**; and the **<body>** and **<head>** elements help to keep page content separate from the other page details.

- Using **lowercase tags** (like <p> instead of <P>). They're not necessary, but they're far more common and easier to type.

- Using **quotation marks** around **attribute values**. The quotation marks are there for a reason — to protect you from mistakes that are all too easy to make. Without quotation marks, one invalid character (>, = or a space) can break your whole page.

HTML5 ALREADY WORKS

- *You can't detect HTML5 support, but you can detect support for individual features, like canvas, video, or geolocation.*

- Before you commit to HTML5, you need to know how well it works with the **browsers your visitors are likely to use**.

- http://caniuse.com
- http://html5readiness.com
- http://gs.statcounter.com
- http://ranking.pl

DEALING WITH OLD BROWSERS

- For the next few years, some of your visitors' browsers won't support all the HTML5 features you want to use.
- But it doesn't need to prevent you from using HTML5 features (see also http://html5please.com):
  - **degrade gracefully** by ignoring nonessential frills, like some of the web form features (like placeholder text) and some of the formatting properties from CSS3 (like rounded corners and drop shadows).
  - **use fallback mechanism** — when a feature doesn't work supply another solution for older browsers, e.g., HTML5's new **<video>** element allows to supply an alternative video player that uses Flash.
  - **use JavaScript workarounds (polyfills)** — many of HTML5 new features can be replicated by using a good JavaScript library.

DEALING WITH OLD BROWSERS

- **Feature detection** with modernizr.com — an open-source, MIT-Licensed JavaScript library that checks which native HTML5 features are available in the current browser.

- *Modernizr* allows you to progressively enhance your pages with a granular level of control over the experience.

- *Modernizr* pairs extremely well with **polyfills** — scripts that replicate the standard API for older browsers, when native support is lacking.

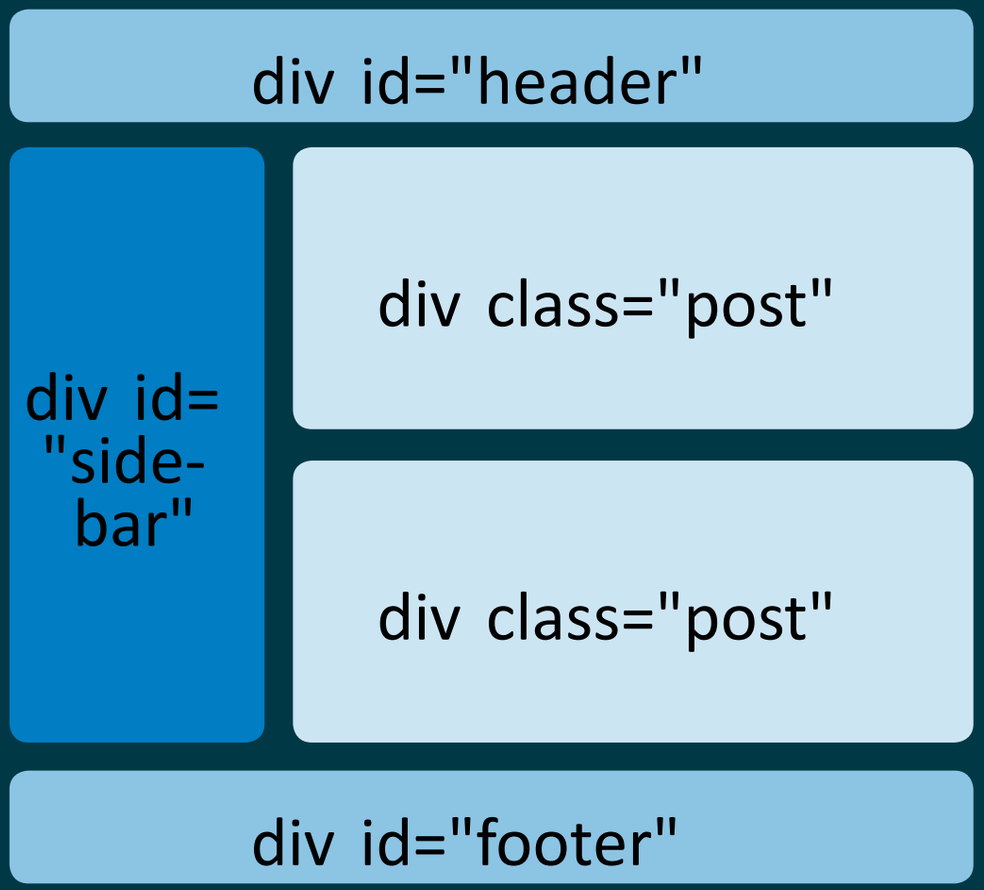
- *Just because you can use a polyfill doesn't mean you should!*

Modernizr

SEMANTICS

PAGE STRUCTURE IN HTML 4

- Most of the structure is entirely unknown to a browser.
- Only one HTML element used for all these important **page landmarks**.
- Semantically neutral ** ** — a generic mechanism for adding structure to documents.
- It’s a straightforward, **all-purpose container** that can be used to apply formatting anywhere.



SEMANTICS

- http://w3c.github.io/html-reference/elements.html
- The majority of new elements are **semantic elements** — these elements do not change anything besides giving extra meaning to the content they enclose.
- Semantics are all about adding meaning to your markup, and there are **several types of information** you can inject.
- How the semantic elements were chosen? https://web.archive.org/web/20160721214418/https://developers.google.com/webmasters/state-of-the-web/

CLASS NAMES

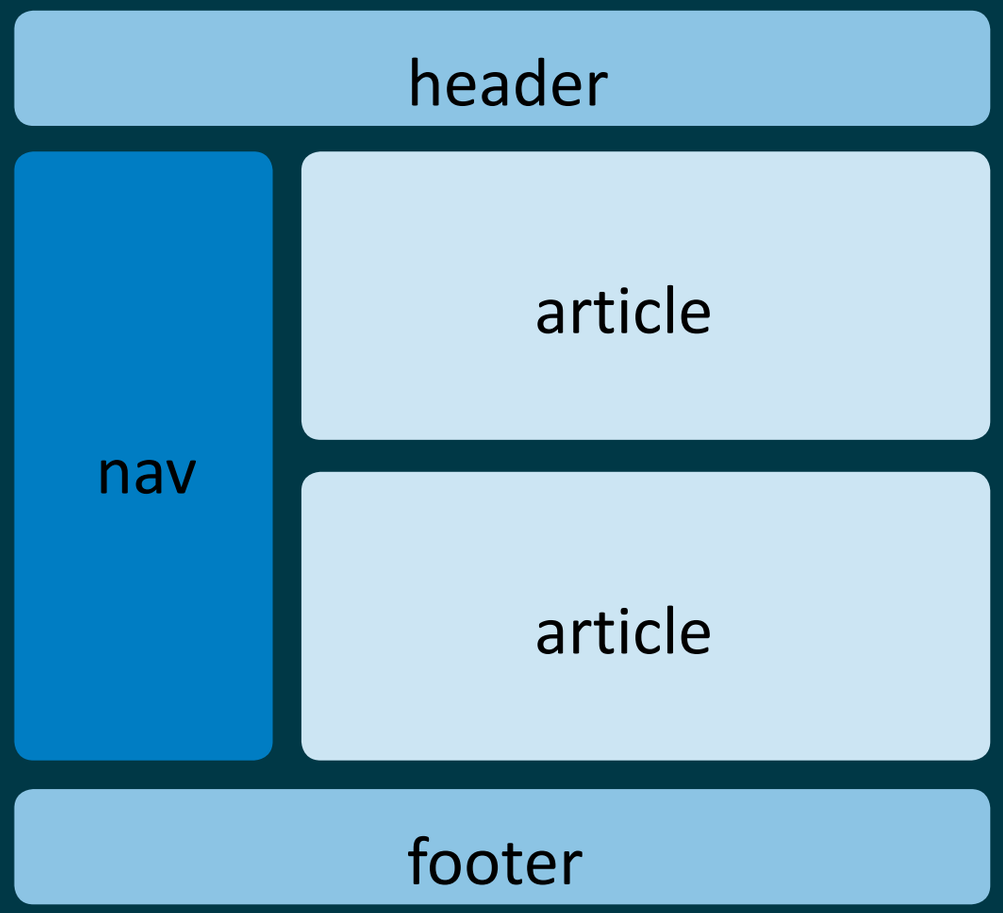
POPULARITY	VALUE	FREQUENCY
1	footer	179,528
2	menu	146,673
3	style1	138,308
4	msonormal	123,374
5	text	122,911
6	content	113,951
7	title	91,957
8	style2	89,851
9	header	89,274
10	copyright	86,979
11	button	81,503
12	main	69,620
13	style3	69,349
14	small	68,995
15	nav	68,634
16	clear	68,571
17	search	59,802
18	style4	56,032
19	logo	48,831
20	body	48,052

ID NAMES

POPULARITY	VALUE	FREQUENCY
1	footer	288,061
2	content	228,661
3	header	223,726
4	logo	121,352
5	container	119,877
6	main	106,327
7	table1	101,677
8	menu	96,161
9	layer1	93,920
10	autonumber1	77,350
11	search	74,887
12	nav	72,057
13	wrapper	66,730
14	top	66,615
15	table2	57,934
16	layer2	56,823
17	sidebar	52,416
18	image1	48,922
19	banner	44,592
20	navigation	43,664

PAGE STRUCTURE IN HTML5

- HTML5 gives us new semantic elements that **unambiguously** denote landmarks in a page.
- Most of semantic elements behave exactly like ** ** elements.
- They group a block of markup, they don’t do anything on their own, and let you apply formatting (using CSS).
- What happens in older browsers that don't understand these elements?



STRUCTURAL ELEMENTS

- **<header>** — typically contains the headings for a section (or the whole page) along with content such as introductory material or navigational aids for the section.

- **<section>** — represents a section of a document or a group of documents; an all-purpose container with a single rule: **the content it holds should begin with a heading**. Use it only if the other semantic elements don't apply.

- **<footer>** — represents the footer at the bottom of the page. This is a tiny chunk of content that may include small print, a copyright notice, and a brief set of links (for example, "About Us" or "Get Support").

STRUCTURAL ELEMENTS

- **<nav>** — represents a section of a document that links to other documents or to parts within the document itself, i.e., a section of navigation links.

- **<article>** — represents whatever you think of as an article; a section of self-contained content like a newspaper article, a forum post, or a blog entry; **the content it holds should begin with a heading.**

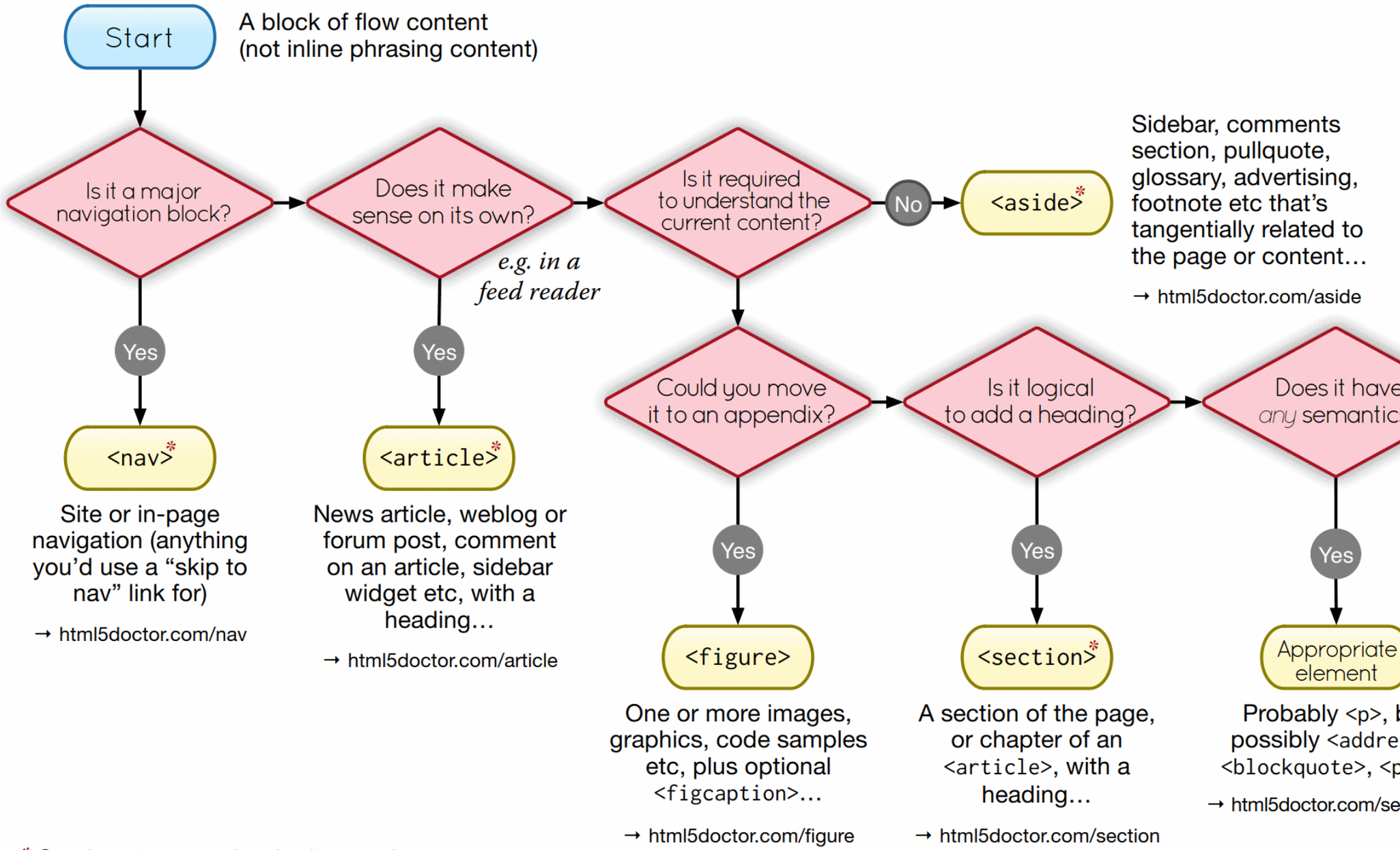
- **<aside>** — represents a complete chunk of content that's separate from the main page content. For example, it makes sense to use <aside> to create a sidebar with related content or links next to a main article.

HTML5 Doctor

HTML5 Element Flowchart

Sectioning content elements and friends

By @riddle & @boblet
www.html5doctor.com



* Sectioning content element

These four elements (and their headings) are used by HTML5's outlining algorithm to make the document's outline
→ html5doctor.com/outline

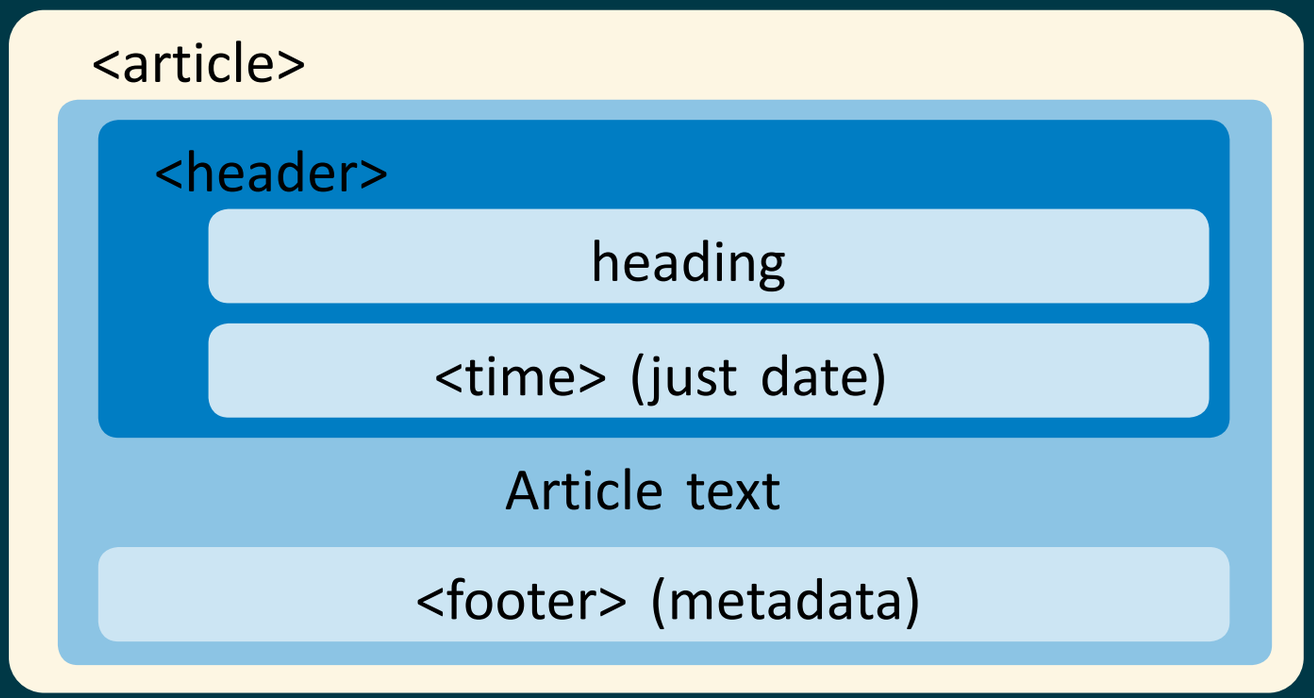
2011-07-22 v1.5
For more information:
www.html5doctor.com/semantics

STRUCTURING EXAMPLE

```
<div class="post">
    <h2>Post title</h2>
    <small>January 24th, 2010</small>

    <div class="entry">
        <p>Blog post text</p>
    </div>

    <p class="postmetadata"><a href="respond">No Comments</a></p>
</div>
```

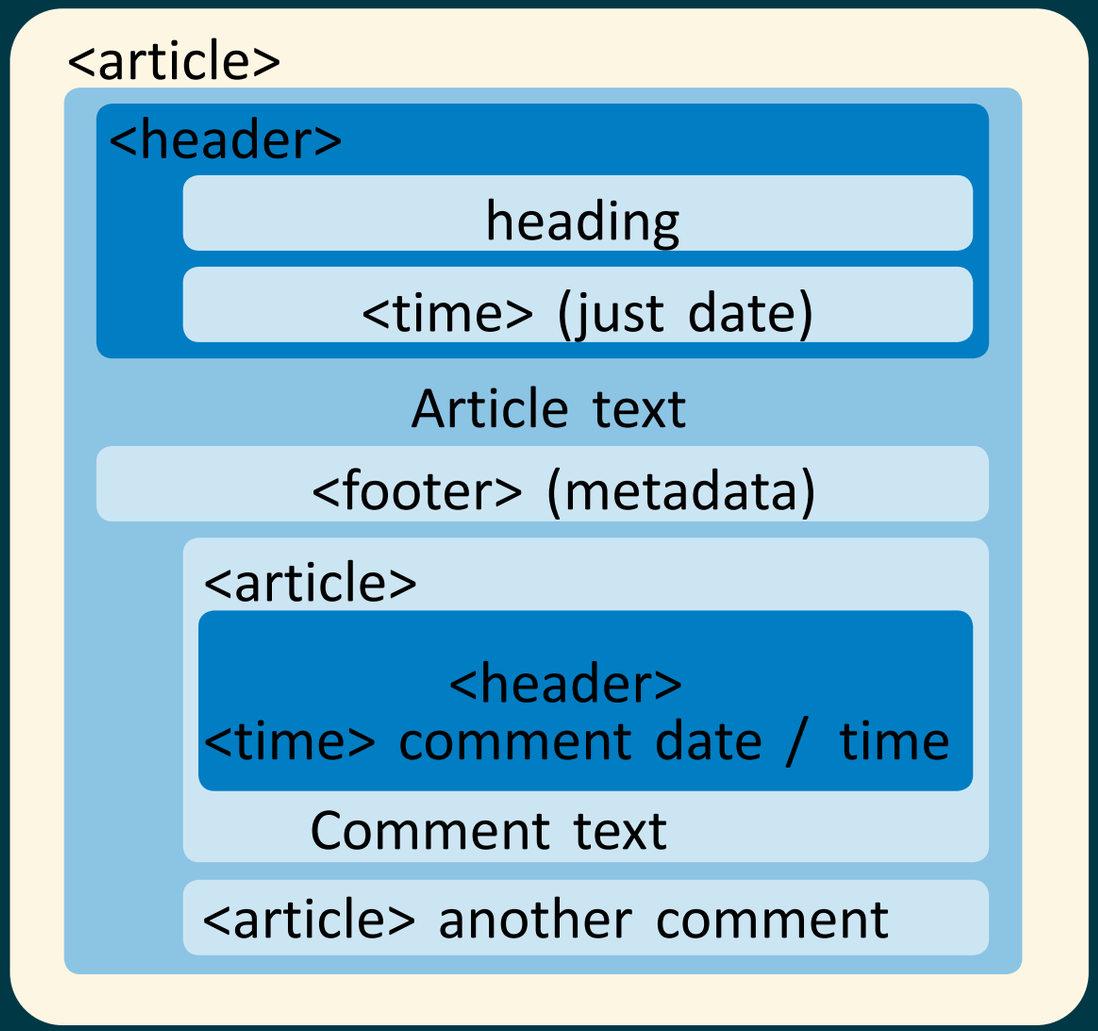
STRUCTURING EXAMPLE

```
<article class="post">
    <header>
        <h2>Post title</h2>
        <time datetime="2010-01-24">
            <small>January 24th, 2010</small>
        </time>
    </header>

    <div class="entry">
        <p>Blog post text</p>
    </div>

    <footer class="postmetadata">
        <a href="respond">No Comments</a>
    </footer>
</article>
```



STRUCTURING EXAMPLE

- When **article elements** are **nested**, the inner article elements represent articles that are in principle **related** to the contents of the outer article.
- For instance, the **comments** to a blog entry could be represented as article elements nested within the article element for the blog entry.



INLINE SEMANTIC ELEMENTS

- Semantics can also include **text-level information**, which you add to explain and point out much smaller pieces of content.

- **<time>** — used for unambiguously encoding dates and times for machines, while still displaying them in a human-readable way.

```
The party starts <time datetime="2014-03-21">March 21<sup>st</sup></time>.
```

- **<output>** — a placeholder that your JavaScript code can use to show a piece of calculated information (the result).

- **<mark>** — a section of text that's highlighted for reference. Can be used to flag important content or keywords, as search engines do when showing matching text in your search results

HTML5 SEMANTIC ELEMENTS BENEFITS

- **Easier editing and maintenance** — interpreting the markup in a traditional HTML page is difficult; using HTML5's semantic elements allows to provide extra structural information.
- **Accessability** — HTML5 can provide a far better browsing experience for disabled visitors.
- **Search-engine optimization** — search bots already check for some of HTML5's semantic elements to glean more information about the pages they're indexing.
- **Future features** — new browsers and web editing tools are sure to take advantage of semantic elements.

OTHER SEMANTICS STANDARDS
ARIA

- Accessibility of web content requires **semantic information** about **widgets**, **structures**, and **behaviors**, in order to allow assistive technologies to convey appropriate information to persons with disabilities.

- **Accessible Rich Internet Applications** (ARIA) provides an ontology of roles, states, and properties that define accessible user interface elements and can be used to improve the accessibility and interoperability of web content and applications.

- These semantics are designed to allow an author to properly convey user interface behaviors and structural information to assistive technologies in document-level markup

- ARIA was invented before HTML5 and later incorporated into HTML5. The semantic elements of HTML5 have default values for ARIA attributes.

ARIA EXAMPLE

```
<!-- The role attributes describe the tab list and each tab. -->
<ol role="tablist">
    <li id="ch1Tab" role="tab">
        <a href="#ch1Panel">Chapter 1</a>
    </li>
    <li id="ch2Tab" role="tab">
        <a href="#ch2Panel">Chapter 2</a>
    </li>
    <li id="quizTab" role="tab">
        <a href="#quizPanel">Quiz</a>
    </li>
</ol>
<div>
<!-- The role and aria-labelledby attributes describe these panels. -->
    <div id="ch1Panel" role="tabpanel" aria-labelledby="ch1Tab">
        Chapter 1 content goes here
    </div>
    <div id="ch2Panel" role="tabpanel" aria-labelledby="ch2Tab">
        Chapter 2 content goes here
    </div>
    <div id="quizPanel" role="tabpanel" aria-labelledby="quizTab">
        Quiz content goes here
    </div>
</div>
```

OTHER SEMANTICS STANDARDS MICRODATA

- **Microdata** is a specification used to nest metadata within existing content on web pages.

- Search engines, web crawlers, and browsers can extract and process Microdata from a web page and use it to provide a richer browsing experience for users.

- Web developers can design a **custom vocabulary** or use vocabularies available on the web.

- A collection of commonly used **markup vocabularies** are provided by **schema.org** schemas which include: *Person, Event, Organization, Product, Review*.

MICRODATA EXAMPLE

```
<section itemscope itemtype="http://schema.org/Person">
       Hello, my name is
       <span itemprop="name">John Doe</span>,
       I am a
       <span itemprop="jobTitle">graduate research assistant</span>
       at the
       <span itemprop="affiliation">University of Dreams</span>.
       My friends call me
       <span itemprop="additionalName">Johnny</span>.
       You can visit my homepage at
       <a href="http://www.JohnnyD.com" itemprop="url">www.JohnnyD.com</a>.
       <section itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
           I live at
           <span itemprop="streetAddress">1234 Peach Drive</span>,
           <span itemprop="addressLocality">Warner Robins</span>,
           <span itemprop="addressRegion">Georgia</span>.
       </section>
   </section>
```

Source: https://en.wikipedia.org/wiki/Microdata_%28HTML%29

HTML5 FORMS

- Before HTML5 the **text input** was used for all sorts of textual input (In the same way that the ** ** element was employed for all sorts of block content).

- HTML5 provides a number of new and backward-compatible **input types** to improve the **semantics of data input.**

- In addition to new form field types, HTML5 introduces **ten common attributes**, that allow to alter the behavior of a given field.

- Some attributes allow the browser to perform **client-side validation** without JavaScript. For example, the `required` attribute specifies that a field must be populated, or the browser will produce an error.

HTML5 FORMS

- Input tag

```
<input name="name" type="type" field-specific-attributes/>
```

- Examples

```
<input name="name1" type="text" value="abc"/>

<input name="name2" type="radio" value="male" checked/>Male
<input name="name2" type="radio" value="female" />Female
```

- Types and support
  - http://www.wufoo.com/html5/

PRESENTATION OUTLINE

- History and Motivation
- HTML5 basics
- **HTML5 features**
  - Semantics
  - **Connectivity**
  - Offline & Storage
  - Multimedia
  - 2D/3D Graphics & Effects
  - Performance & Integration
  - Device Access
  - Styling

CONNECTIVITY

CONNECTIVITY

- Before HTML5 there was one main tool that allows a web page to speak to a web server — **`XMLHTTPRequest.`**
- HTML5 provides **two new ways** for web pages to talk with a web server:
  - **server-sent events** — allows a server to push events to a client, rather than the classical paradigm where the server could send data only in response to a client's request.
  - **web sockets —** allows creating a permanent connection between the page and the server and to exchange non-HTML data through that means.

CONNECTIVITY
XML HTTP REQUEST

- **`XMLHttpRequest (XHR)`** is a **JavaScript** object that provides an easy way to retrieve data from a URL **without a full page refresh**.

- A web page can update just a **part of the page** without disrupting what the user is doing — requests take place **asynchronously**, the web page stays responsive.

- `XMLHttpRequest` is used heavily in **AJAX** (Asynchronous JavaScript and XML) programming.

- **`XMLHttpRequest` Level 2** extends the functionality of XMLHttpRequest object, including, **progress events**, support for **cross-site requests**, and the **handling of byte streams**.

CONNECTIVITY
FETCH

- **`Fetch API`** is a HTML5 replacement for `XMLHttpRequest`
- Supports all features of `XMLHttpRequest` plus Promises
  - Promise is JavaScript object that represents future result of activity and fires events when activity is done
- Simpler to use than `XMLHttpRequest`
- Supported in all major browsers
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

CONNECTIVITY
CROSS-ORIGIN RESOURCE SHARING

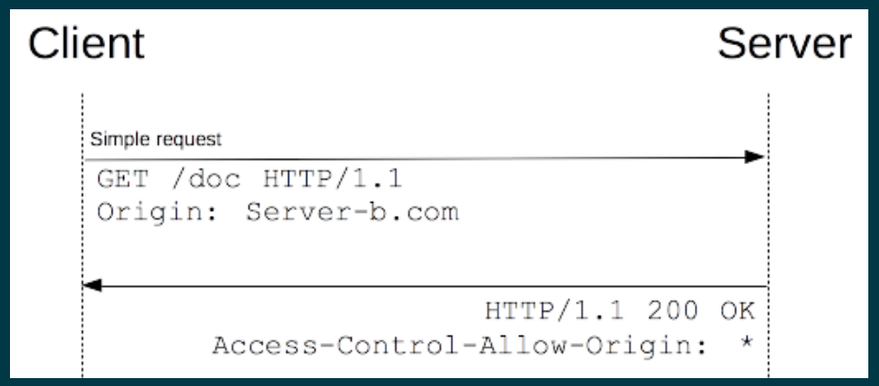
- **Cross-site HTTP requests** using the `XMLHttpRequest` and `Fetch` objects have been subject to the **same-origin policy** — a web application could only make requests to the domain it was loaded from, and not to other domains.

- **Cross-Origin Resource Sharing (CORS)** mechanism provides a way for web servers to support cross-site access controls, which enable **secure cross-site data transfers**.

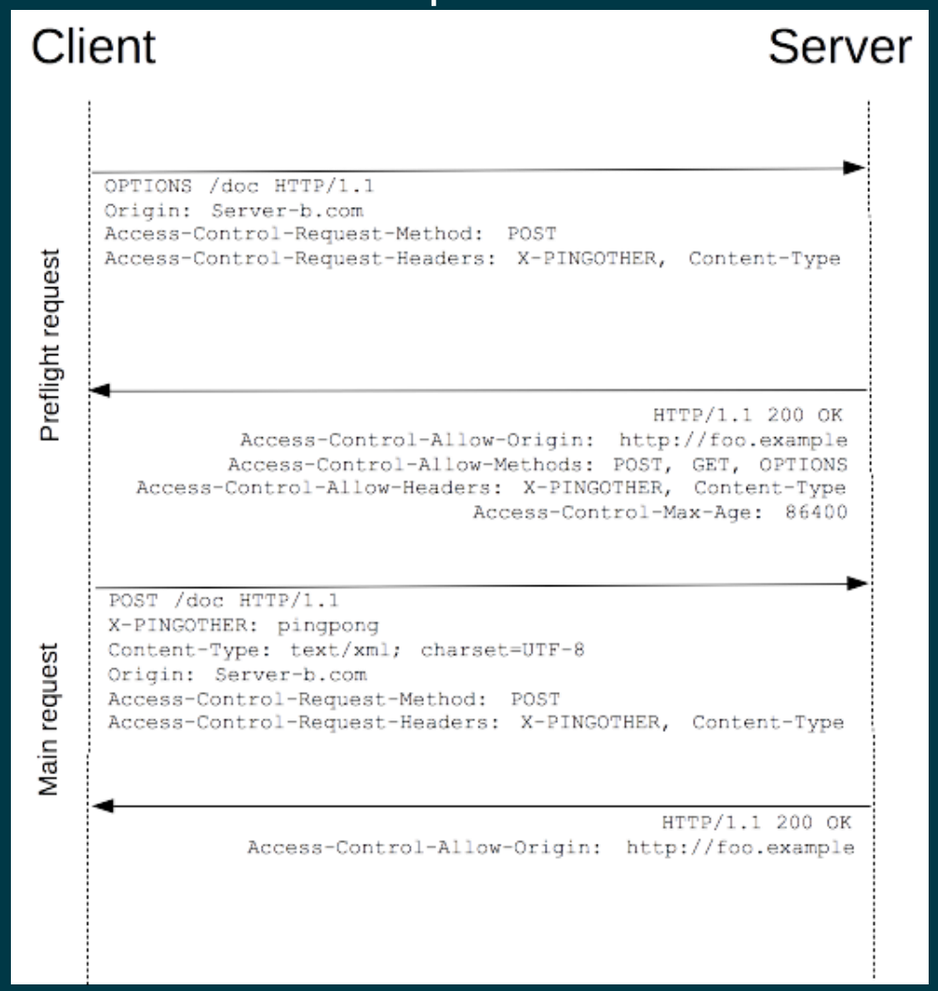
- The Cross-Origin Resource Sharing standard works by adding new **HTTP headers** that allow **servers** to describe the **set of origins** that are **permitted** to read the information.

CONNECTIVITY
CROSS-ORIGIN RESOURCE SHARING

- Safe HTTP requests (GET, HEAD, POST without payload):



- Other HTTP requests:



- The CORS header is verified by the browser, if it is not valid, the response is not available for a script
- Browser uses OPTIONS preflight request to acquire CORS headers before running actual request

https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

CONNECTIVITY
SERVER-SENT EVENTS

- With `XMLHttpRequest` and `Fetch` once the server provides its response, the interaction is over — there's no way for the web server to wait a moment and send another message with an **update**.

- One approach to maintain a **longer-term** web server **relationship** is to use **polling** — periodically checking the web server for new data (or long polling, i.e. **hanging GET**).

- HTML5 provides **server-sent events (SSE)**, which let a web page hold an open connection to the web server.

- With SSE a browser receives updates from a server via HTTP connection and processes them using **`EventSource`** API.

CONNECTIVITY
SERVER-SENT EVENTS

- Unlike `XMLHttpRequest` and `Fetch`, the standard of the server-sent events doesn't let you send just **arbitrary data**.

- You need to follow a **simple but specific format**:

```
event: userconnect
data: {"username": "bobby", "time": "02:33:48"}

event: usermessage
data: {"username": "bobby", "time": "02:34:11", "text": "Hi everyone."}

event: userdisconnect
data: {"username": "bobby", "time": "02:34:23"}

event: usermessage
data: {"username": "sean", "time": "02:34:36", "text": "Bye, bobby."}
```

CONNECTIVITY
SERVER-SENT EVENTS

- Subscribe to events using JavaScript:

```
var source = new EventSource("demo_sse.php");
source.onmessage = function(event) {
    document.getElementById("result").innerHTML += event.data + "<br/>";
};
```

CONNECTIVITY
WEB SOCKETS

- With server-sent events the communication is completely **one-sided** — there's no way for the browser to **respond**, or to enter into a **more complex dialogue**.

- If a web application requires **bidirectional communication**, one approach is to use the `XMLHttpRequest` or the `Fetch` object, but:
  - it is difficult to determine order of messages if you send **multiple asynchronous messages** back and forth very quickly (e.g. in a **chat application**).
  - there's **no way to associate one call with the next**, so every time the web page makes a request, the web server needs to sort out who you are all over again.

CONNECTIVITY
WEB SOCKETS

- The **HTML5 WebSockets** specification defines an **API** that enables web pages to use the WebSockets protocol for **two-way communication** between the user's browser and a server.

- WebSocket is a protocol providing **full-duplex** communication channels over a single TCP connection.

- WebSockets provides an enormous step forward in the **scalability** of the **real-time**, **event-driven web applications**.

- Web Sockets can provide even a **1000:1 reduction in HTTP header traffic** and **3:1 reduction in latency**.

  - Source: https://websocket.org/quantum.html

WEB SOCKETS EXAMPLE

```
if ("WebSocket" in window) {
    alert("WebSocket is supported by your Browser!");

    // Let us open a web socket
    var ws = new WebSocket("ws://localhost:9998/echo");

    ws.onopen = function () {
        // Web Socket is connected, send data using send()
        ws.send("Message to send");
        alert("Message is sent...");
    };

    ws.onmessage = function (evt) {
        var received_msg = evt.data;
        alert("Message is received...");
    };

    ws.onclose = function () {
        // websocket is closed.
        alert("Connection is closed...");
    };
}
else
{
    // The browser doesn't support WebSocket
    alert("WebSocket NOT supported by your Browser!");
}
```

Source: http://www.tutorialspoint.com/html5/html5_websocket.htm

OFFLINE & STORAGE

OFFLINE & STORAGE
SERVICE WORKERS

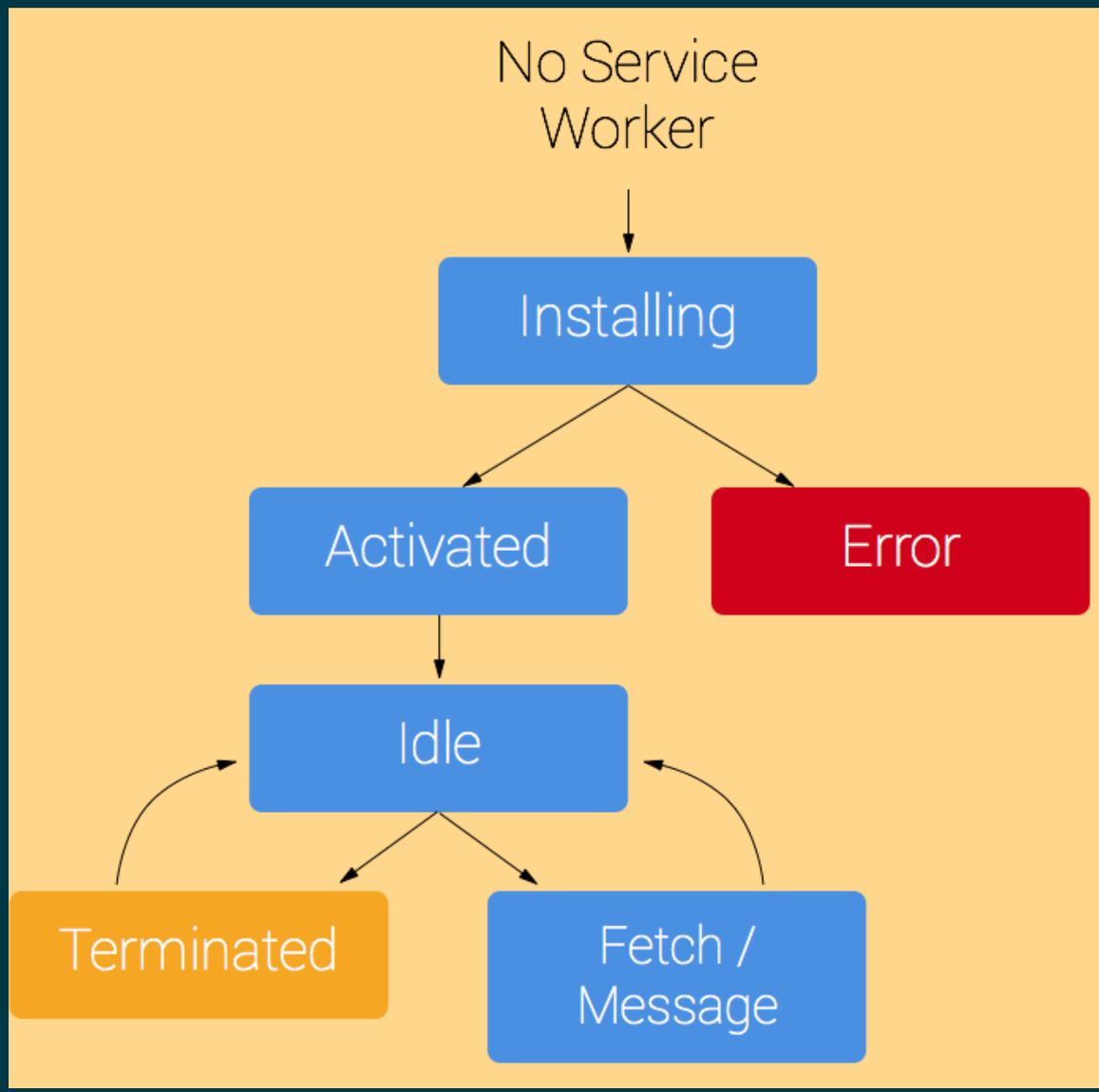
- A service worker is a script that a browser runs in the background, separate from a web page and without user interaction. Web workers are intended for **on-demand caching**, **push notifications** and **background sync**.

- A service worker is like a proxy between a web page and a server. It takes over the requests raised by the web page and serves them using its custom behavior.

- Things to note about a service worker:
  - It can't access the DOM directly. Instead, a service worker can communicate with the pages it controls by responding to messages sent via the `postMessage` interface, and those pages can manipulate the DOM.
  - It's terminated when not in use, and restarted when it's next needed, so you cannot rely on global state within a service worker's `onfetch` and `onmessage` handlers. If there is information that you need to persist and reuse across restarts, service workers do have access to the IndexedDB API.
  - It handles HTTPS connections only.

OFFLINE & STORAGE
SERVICE WORKER LIFECYCLE

- The first event a service worker gets is `install`. It's triggered once as soon as the worker executes. After updating the service worker script, the new script gets its `install` event.
- Once a service worker is ready to control clients and handle functional events like `push` and `sync`, it gets an `activate` event.
- A service worker won't receive events like `fetch` and `push` until it successfully finishes installing and becomes "active".
- By default, page's fetches won't go through the service worker unless the page request itself went through a service worker. So you'll need to refresh the page to see the effects of the service worker.
- `clients.claim()` can override this default, and take control of non-controlled pages.



OFFLINE & STORAGE
ON-DEMAND CACHING EXAMPLE

- An HTML page

```
<!DOCTYPE html>
An image will appear here in 3 seconds:
<script>
  navigator.serviceWorker.register('/sw.js')
    .then(reg => console.log('SW registered!', reg))
    .catch(err => console.log('Boo!', err));

  setTimeout(() => {
    const img = new Image();
    img.src = '/dog.svg';
    document.body.appendChild(img);
  }, 3000);
</script>
```

- A service worker (sw.js)

```
self.addEventListener('install', event => {
  console.log('V1 installing…');

  // cache a cat SVG
  event.waitUntil(
    caches.open('static-v1').then(cache => cache.add('/cat.svg'))
  );
});

self.addEventListener('activate', event => {
  console.log('V1 now ready to handle fetches!');
});

self.addEventListener('fetch', event => {
  const url = new URL(event.request.url);

  // serve the cat SVG from the cache if the request is
  // same-origin and the path is '/dog.svg'
  if (url.origin == location.origin && url.pathname == '/dog.svg') {
    event.respondWith(caches.match('/cat.svg'));
  }
});
```

OFFLINE & STORAGE
BACKGROUND SYNC EXAMPLE

- An HTML page

```
<!DOCTYPE html>
<script>
   // Register your service worker:
   navigator.serviceWorker.register('/sw.js');

   // Then later, request a one-off sync:
   navigator.serviceWorker.ready.then(function(registration) {
     return registration.sync.register('myFirstSync');
   });
</script>
```

- A service worker (sw.js)

```
self.addEventListener('sync', function(event) {
  if (event.tag == 'myFirstSync') {
    event.waitUntil(doSomeStuff());
  }
});
```

- In the above, `doSomeStuff()` should return a promise indicating the success/failure of whatever it's trying to do. If it fulfills, the sync is complete. If it fails, another sync will be scheduled to retry. Retrying syncs wait for connectivity, and employ an exponential back-off.

OFFLINE & STORAGE
CLIENT-SIDE STORAGE

- Certain types of data should be stored on the **client-side**:
  - **user preferences** (setting how the web page tailors its display)
  - **application state** (a snapshot of where the web application is right now, so the web visitor can pick up at the same spot later on).

- Before HTML5, the only way to get local storage was to use **cookies**, a mechanism originally devised to transmit small bits of identifying information.
  - Most browsers limit the **size of cookies** to 4KB, and the **number of cookies** stored per domain to 20.
  - Cookies are sent in every HTTP request, adding **overhead** to transactions and **slowing down** page loads.

OFFLINE & STORAGE
CLIENT-SIDE STORAGE

- HTML5 provides alternatives to cookies allowing the application to **persistently** save a **larger amount** of data on the client device, **without sending** it to the web server:
  - **Web Storage** simply provides a key-value mapping.
  - **Indexed Database** supports indexes like those of relational databases, so searching objects matching a particular field is fast.
- Client-side storage is a particularly useful for:
  - making **self-sufficient offline applications** that can store all the information they need, even when there's **no web connection**.
  - enhancing **performance** — e.g., in applications that continually retrieve the same data or generate data using complex calculations.

OFFLINE & STORAGE
WEB STORAGE

- Web Storage is an example of a **NoSQL key-value store**.
- There are two types of web storage that can be accessed via **globally available JavaScript objects**:
  - **Local storage** uses the `localStorage` object to store data **permanently** and make it accessible to any page in your website; most browsers allow users to clear out local storage.
  - **Session storage** uses the `sessionStorage` object to store data **temporarily**, for a single browser window; the data remains until the window is closed, at which point the session ends and the data disappears.
- Present implementations only support **string-to-string mappings**, so you need to serialize and deserialize other data structures.
- Web Storage supports events fired when a data is written. The events are fired in all browser windows sharing the same storage, thus can be used as inter-window communication mechanism.

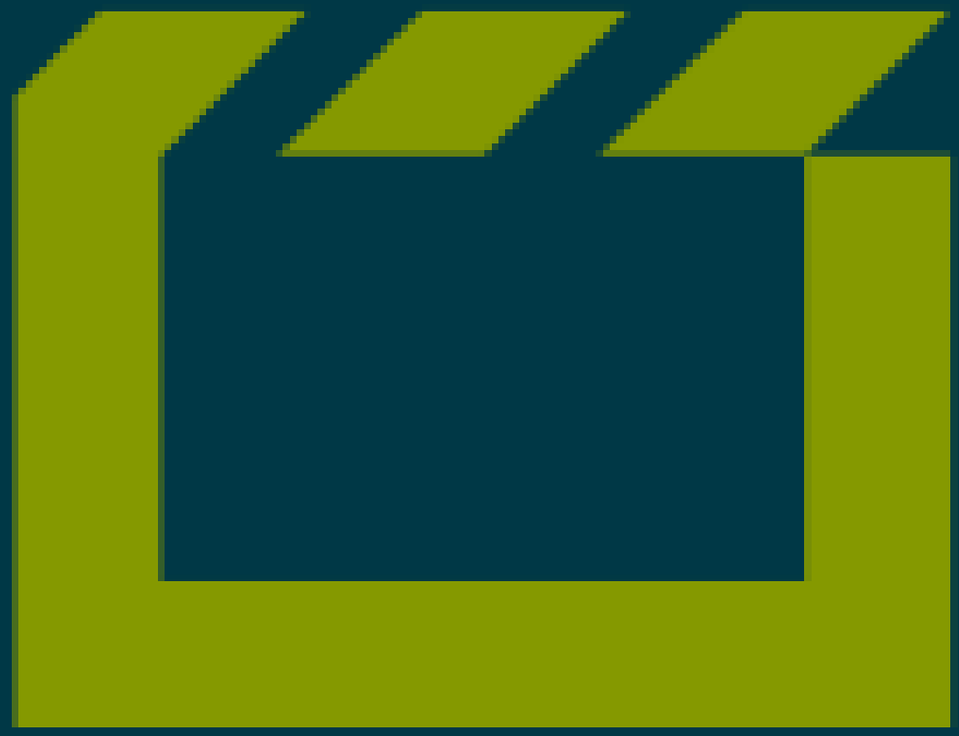
OFFLINE & STORAGE
INDEXED DB

- Although the Web Storage can store **megabytes of data** (browsers limit capacity in their own ways), it's not ideal for storing **complex data structures** that would typically be stored in a **database**.

- The **IndexedDB API** provides developers with a means of storing **significant** amounts of **structured** data in a **client-side JavaScript object-oriented database**.

- In IndexedDB the values are fully **indexable**, making it a viable solution for any application where you need to **search** or **filter data**.

- IndexedDB uses an **asynchronous model** — database tasks happen in the background, without stalling your code or locking up the page.



MULTIMEDIA

MULTIMEDIA

- The majority of **internet bandwidth** in recent years has been driven by the delivery of multimedia content.

- Cisco reports that the trend isn't slowing down, estimating that a staggering **73%** of all internet traffic was **video** in **2016**, and predicting increase of that value to **82%** in **2021**. The fastest growing market segment is **live video streaming**.

- **Adobe Flash** was *de facto* standard of showing video on the web — it works everywhere the plug-in is installed, and currently that's on more than **99** percent of connected computers.

- Problems with Flash:
  - markup that uses the **<object>** and **<embed>** elements.
  - **security** and **performance** of Flash as a platform for video.
  - **lack of support** for Flash on some mobile devices.

MULTIMEDIA

- HTML5 introduces built-in media support via the **<audio>** and **<video>** elements, offering the ability to easily embed media into HTML documents.

- New elements allow supported multimedia files to be played **natively** by the **browser**, with **no third-party plug-ins required**.

- Controlling an HTML5 players to **play**, **pause**, increase and decrease **volume** using some Javascript is straightforward.

- www.youtube.com/html5

ATTRIBUTES	METHODS	EVENTS
error state	load()	loadstart
error	canPlayType(type)	progress
network state	play()	suspend
src	pause()	abort
currentSrc	addTrack(label, kind, language)	error
networkState		emptied
preload		stalled
buffered		play
ready state		pause
readyState		loadedmetadata
seeking		loadeddata
controls		waiting
controls		playing
volume		canplay
muted		canplaythrough
tracks		seeking
tracks		seeked
playback state		timeupdate
currentTime		ended
startTime		ratechange
duration		
paused		
defaultPlaybackRate		
playbackRate		
played		
seekable		
ended		
autoplay		
loop		
width [video only]		
height [video only]		
videoWidth [video only]		
videoHeight [video only]		
poster [video only]		

MULTIMEDIA SUPPORTED FORMATS

- The HTML5 specification **does not specify** which video formats browsers should support.

- User agents are **free** to support **any video formats** they feel are appropriate, but content authors **cannot assume** that any video will be **accessible** by all complying user agents.

- The ideal format would:
  - be **royalty-free**,
  - have good **compression** and image **quality**
  - a **hardware video decoder** should exist for the format, as many embedded processors do not have the performance to decode video.

MULTIMEDIA SUPPORTED FORMATS

- The result has been the polarisation of HTML5 video:
  - **industry-standard but patented** formats (MP4, H.264)
  - **free, open formats** (WebM, Ogg Theora, Ogg Vorbis)

- A web page can provide video in multiple formats — the browser will choose automatically which file to download:

```
<video poster="movie.jpg" controls>
  <source src="movie.webm" type='video/webm; codecs="vp8.0, vorbis"'>
  <source src="movie.ogv" type='video/ogg; codecs="theora, vorbis"'>
  <source src="movie.mp4" type='video/mp4; codecs="avc1.4D401E, mp4a.40.2"'>
  <p>This is fallback content to display for user agents that do not support
  the video tag.</p>
</video>
```

GRAPHICS & EFFECTS

GRAPHICS & EFFECTS
CANVAS

- HTML5 defines the **<canvas>** element as
  "a resolution-dependent bitmap canvas that can be used for rendering graphs, game graphics, or other visual images on the fly."

- A canvas is a rectangular **drawing surface** in your page where you can use **JavaScript** to draw anything you want.

- HTML5 defines a set of functions (the canvas API) for drawing **shapes**, defining **paths**, creating **gradients**, adding **text** and applying **transformations** without additional plug-ins.

- The API also provides developers with a way to **export** the current **content of the canvas** as a PNG or JPG format image.

GRAPHICS & EFFECTS
CANVAS 3D / WEBGL

- **Web Graphics Library** (WebGL), a JavaScript API for creating 3D graphics using the **<canvas>** element (without the use of plug-ins).

- WebGL is based on the **Open Graphics Library for Embedded Systems** (OpenGL ES) standard, which was designed for implementing 3D on embedded devices including mobile phones.

- It provides developers with an API that allows them to control graphics hardware at a **low level** using **shader**, **buffer**, and **drawing** methods.

- WebGL programs consist of control code written in JavaScript and special effects code (shader code) that is executed on a computer's **Graphics Processing Unit** (GPU)

- Examples
  - http://carvisualizer.plus360degrees.com/threejs/
  - http://hexgl.bkcore.com/play/

GRAPHICS & EFFECTS
SCALABLE VECTOR GRAPHICS

- **Scalable Vector Graphics** (SVG) is an XML language for describing two-dimensional vector graphics.

- HTML5 specification gives you the ability to use SVG directly in your HTML markup.

- SVG maintains a tree that represents the current state of all the objects drawn on-screen — every graphical object is also a **DOM object**, so you ca attach **JavaScript event handlers** or modify them later.

PERFORMANCE & INTEGRATION

PERFORMANCE & INTEGRATION
WEB WORKERS

- A **web worker** is a script executed from an HTML page that runs in the **background**, independently of other **user-interface** scripts that may also have been executed from the same HTML page.
- Web workers are able to utilize **multi-core** CPUs more effectively.
- Using web workers allows web pages to remain **responsive** at the same time as they are **running long tasks** in the background.

PERFORMANCE & INTEGRATION
WEB WORKERS

- Web workers can do **complex mathematical calculations**, **make network requests**, **access IndexedDB** while the main web page responds to the user scrolling, clicking, or typing.
- Web workers **do not have access to DOM** and are executed in separate context from the “normal” JS code (don’t have access to objects defined there)
- There is no synchronization mechanism between web workers
  - E.g., no locks, mutexes, semaphores etc.
- To communicate between web worker(s) and normal JS code use message queues and events.

PERFORMANCE & INTEGRATION
DRAG-AND-DROP

- Lack of **drag-and-drop interactivity** had been an issue that has plagued web application developers for a long time.

- This type of functionality has been prevalent in **desktop applications** for as long as graphical UIs have been around.

- Up until now, developers had to rely on using **JavaScript libraries** (e.g. Dojo) to provide web apps with decent drag-and-drop features (e.g. for rearranging the order of a list or moving documents in content management systems).

- In HTML5, a full Drag and Drop API has been specified.

PERFORMANCE & INTEGRATION
DRAG-AND-DROP API

- To use drag and drop in HTML5, you can use the `draggable` attribute on an element to explicitly define that **element as draggable** (many elements, such as images, are draggable by default.)

- You can then use a series of **events** to **listen for changes** as the user **drags** the element into and out of other elements and indeed when the user **drops** the element.

- The API allows you to set the **data** you want to **associate** with the **drag operation** and then to read this back when dropped.

- A **new feature** of HTML5 drag and drop (combined with **File API**) is the ability to **drag files from your computer** and drop them into a web application: an example of this functionality can be seen in **Gmail**.

PERFORMANCE & INTEGRATION
FULLSCREEN API

- The **Fullscreen API** provides an **easy** way for web content to be presented using the user's entire screen.
- The Fullscreen API allows to make an **arbitrary HTML element** "full-screen", **hiding** the **browser's UI** and stretching the element to encompass the entire screen.
- This API is particularly useful for HTML5 **video** and **games**.
- The API is not consistently implemented — there are subtle **presentation differences** between the browsers.

PERFORMANCE & INTEGRATION
HISTORY API

- The HTML5 History API gives developers the ability to **modify** a website's **URL without a full page refresh.**

- This is particularly useful in **Single Page Applications** for loading portions of a page with AJAX, such that the content is significantly different and warrants a new URL.

- The History API provides two new methods:
  - `history.pushState()` — adds a new entry in the history stack
  - `history.replaceState()` — replaces current history value

- HTML5 History API also allows us to build applications in an **SEO-friendly** manner.

PERFORMANCE & INTEGRATION
PAGE VISIBILITY API

- With tabbed browsing, there is a reasonable chance that any given webpage is in the background and thus not visible to the user.

- The **Page Visibility API** provides events you can watch for to know when a document becomes visible or hidden, as well as features to look at the current visibility state of the page.

- When the user minimizes the window or switches to another tab, the API sends a `visibilitychange` event to let listeners know the state of the page has changed.

- http://daniemon.com/tech/webapps/page-visibility/

DEVICE ACCESS

DEVICE ACCESS
GEOLOCATION

- The **Geolocation API** allows the user to provide their **location** to web applications (if **GPS isn't available**, devices can fall back to other means of tracking location, such as **IP address**).

- For **privacy** reasons, the user is asked for **permission** to report location information.

- The Geolocation API can also support features like:
  - **tracking** user movement over set time **intervals**
  - obtaining the user's **altitude**, **heading**, and **speed**
  - **limiting** GPS use when **battery life** is a concern

DEVICE ACCESS
MOBILE DEVICES API

- The **Device Orientation API** delivers events to your web page that correspond to the movement of the device:
  - `DeviceOrientationEvent` — sent when the accelerometer detects a change to the orientation of the device; allows to respond to **rotation** and **elevation** changes
  - `DeviceMotionEvent` — listening for changes in acceleration (instead of orientation)

- The **Battery API** allows to adjust the amount of processing depending on the state of the battery, e.g. you could avoid doing any heavy processing or reduce the number of network connections when the battery is low.

- Mobile devices offer alternative input — the **Vibration API** provides access to a mobile's built-in vibration function.

DEVICE ACCESS
CAMERA API

- Through the **Camera API**, it is possible to take **pictures** with your **device's camera** and **upload** them into the current web page.

```
<input type="file" id="take-picture" accept="image/*">
```

- When users choose to activate this HTML element, they are presented with an option to choose a file, where the device's camera is one of the options.

- If they select the camera, it goes into picture taking mode. After the picture has been taken, the user is presented with a choice to accept or discard it. If accepted, it gets sent to the `<input type="file">` element and its `onchange` event is triggered.

DEVICE ACCESS
MEDIA CAPTURE AND STREAMS

- The `MediaDevices.getUserMedia()` method prompts the user for permission to use a media input which produces a `MediaStream` with tracks containing the requested types of media.

- That stream can include, for example,
  - a video track produced by either a hardware or virtual video source such as a camera, video recording device, screen sharing service, and so forth,
  - an audio track produced by a physical or virtual audio source like a microphone, A/D converter, or the like,
  - possibly other track types.
- https://webrtc.github.io/samples/
- https://appr.tc/

DEVICE ACCESS CERTIFICATES AND SMART CARDS

- The **Web Crypto API** is an interface allowing a script to use cryptographic primitives in order to build systems using cryptography

- Web Crypto API methods are available through `Crypto.subtle` property.

- The SubtleCrypto API provides the following cryptography functions:
  - `sign()` and `verify()`: create and verify digital signatures.
  - `encrypt()` and `decrypt()`: encrypt and decrypt data.
  - `digest()`: create a fixed-length, collision-resistant digest of some data.

STYLING

STYLING
CASCADING STYLE SHEETS

- Cascading Style Sheets (CSS), is a **stylesheet language** used to describe the **presentation** of a document written in HTML or XML, including elements such as the **layout**, **colors**, and **fonts**.

- More information in the next lecture!

CONCLUSIONS

HTML5 = NEXT GENERATION FEATURES FOR MODERN WEB DEVELOPMENT

1. *It's not one big thing — it is a collection of individual features.*
2. *You don't need to throw anything away — applications that worked yesterday in HTML 4 will still work today in HTML5.*
3. *It's easy to get started — upgrading to HTML5 is simple.*
4. *It already works — it is already well-supported by most browsers.*
5. *It's here to stay — HTML5 is the future of web standards.*

REFERENCES

- *Dive into HTML* — Mark Pilgrim
  http://diveintohtml5.info

- HTML5 - Web Developer Guide, Mozilla Developer Network
  https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

- http://slides.html5rocks.com

- http://html5doctor.com

- *HTML5 in Action — Rob Crowther, Joe Lennon, Ash Blue, Greg Wanish. Manning Publications, 2014*

- *Introducing HTML5 — Bruce Lawson and Remy Sharp. New Riders: Pearson Education, 2014*

- *Service Workers: an Introduction — Matt Gaunt*
  https://developers.google.com/web/fundamentals/primers/service-workers/

- *The Service Worker Lifecycle — Jake Archibaldhttpshttps://developers.google.com/web/fundamentals/primers/service-workers/lifecycle*

- *Introducing Background Sync — Jake Archibaldhttps://developers.google.com/web/updates/2015/12/background-sync*