

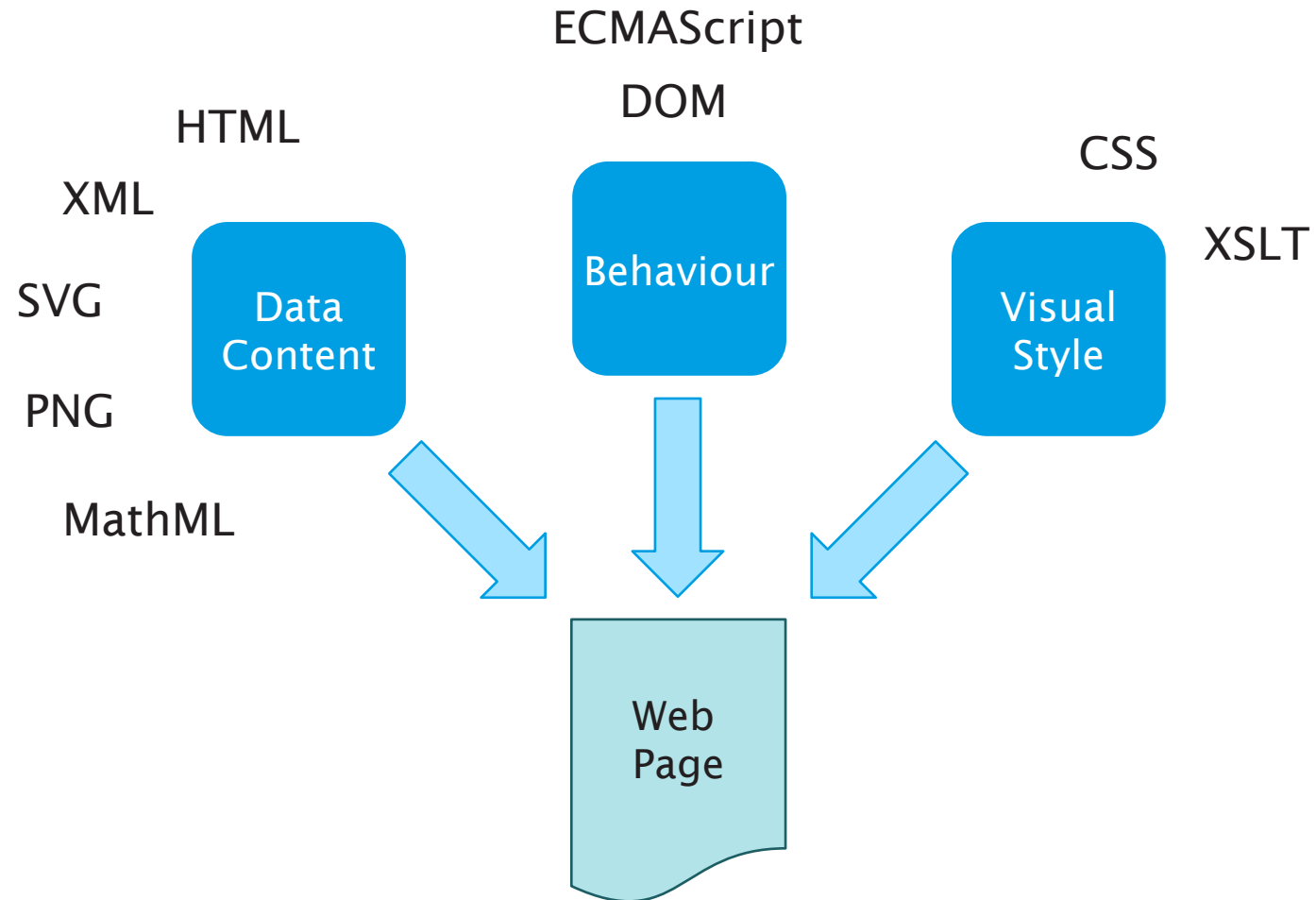
UNIVERSITY OF
Southampton

Web APIs

COMP3220 Web Infrastructure

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk

Content, Behaviour, Presentation



HTML as application platform

Web APIs fall into three broad classes:

1. Document content
 - Document Object Model, Canvas
2. Browser services
 - XMLHttpRequest, Fetch, WebSockets
 - Web Storage, IndexedDB
 - Web Workers, Service Workers
3. Hardware access
 - Geolocation, Media Capture, Vibration, Battery Status

A word on standardisation

Status of APIs is highly variable

- Some are W3C recommendations
- Some are WHATWG “living standards”
- Some are both (but equivalent)
- Some are both (but differ)
- Some are neither (e.g. WebGL)

Browser support is variable

- Write defensive code – check for API support before calling it

Document Object Model

Document Object Model (DOM)

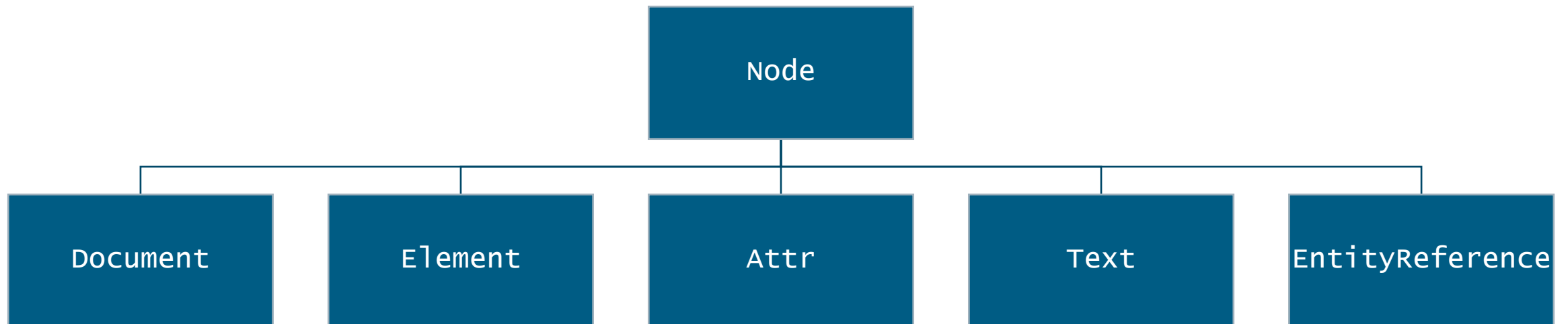
Standard API for accessing and manipulating XML and HTML

- Document represented as a hierarchy of objects of type Node
- Root of hierarchy is an object of type Document

Node interface is the key to understanding DOM

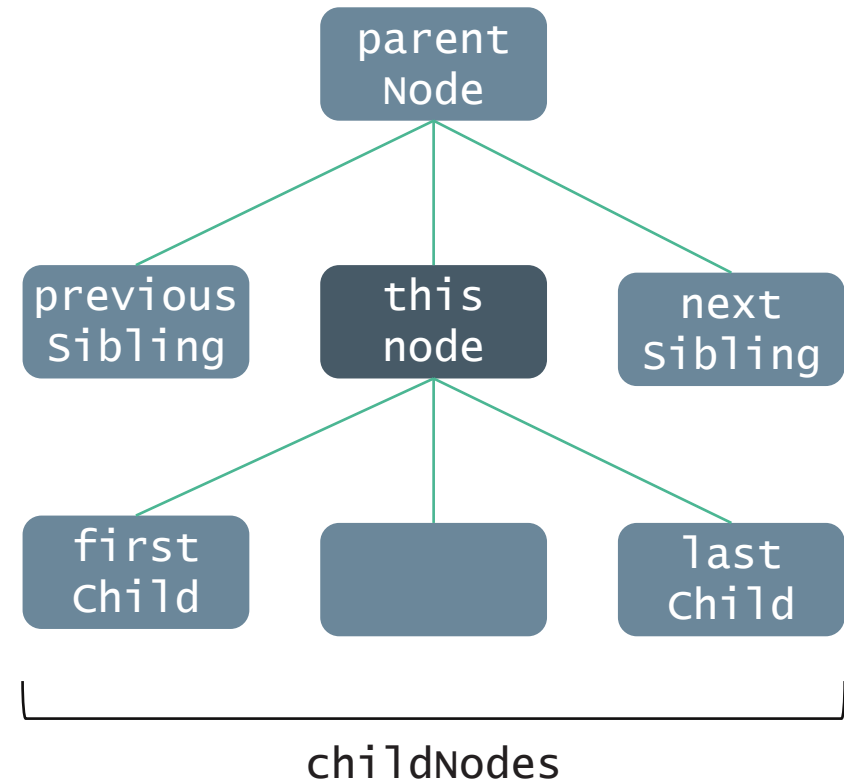
- Methods for basic access and manipulation of hierarchy
- Other types derived from Node add further methods

Selected Node type hierarchy



Node interface - attributes

- parentNode
- childNodes
- firstChild
- lastChild
- previousSibling
- nextSibling
- attributes



Node interface – methods

`insertBefore(newChild, refChild)`

- Inserts `newChild` into list of children after `refChild`

`replaceChild(newChild, oldChild)`

- Replaces `oldChild` in list of children with `newChild`

`removeChild(oldChild)`

- Removes `oldChild` from list of children

`appendChild(newChild)`

- Adds `newChild` to the end of the list of children

`cloneNode(deep)`

- Returns a duplicate of the node (which may be a deep copy)

Document interface – methods

`getElementsByTagName(tagname)`

- Get a list of all elements with the given tag name

`getElementById(elementId)`

- Get the element with the given ID

`createElement(tagName)`

`createAttribute(name)`

`createTextNode(data)`

`createEntityReference(name)`

Element interface – methods

`getAttribute(name)`

- Returns value of named attribute

`setAttribute(name, value)`

- Sets value of named attribute

`getElementsByTagName(name)`

- Get a list of all descendant elements with the given tag name

Canvas

Canvas 2D Context

API for drawing graphics via JavaScript

- Uses <canvas> element as container for 2d context
- Animation via JavaScript
(compare with declarative animation in SVG)

Canvas example - canvas.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Canvas example</title>
  </head>
  <body>
    <canvas id='canvas' width='600' height='300'>
      Canvas not supported
    </canvas>
    <script src='canvas.js'></script>
  </body>
</html>
```

external script

fallback content

Canvas example – canvas.js

```
var canvas = document.getElementById('canvas');  
var context = canvas.getContext('2d');
```

```
context.fillStyle = 'red';  
context.fillRect(10,10,110,60);  
context.font = '32pt Lucida Sans';  
context.strokeStyle = 'blue';  
context.strokeText("Lorem Ipsum", 40, 40);
```



WebGL

Low-level 3D graphics API based on OpenGL

Defined as an additional context for the canvas element

```
var canvas = document.getElementById('canvas');  
var context = canvas.getContext('webgl');
```

XMLHttpRequest and Fetch

XMLHttpRequest

API for fetching representations of resources

Asynchronous

- Register onreadystatechange handler function for response
- AJAX – Asynchronous JavaScript and XML

XMLHttpRequest example

```
function handler() {  
    if (this.status == 200 && this.response != null {  
        // do something with the resource  
    } else {  
        console.error("Request failed: HTTP status: " + this.status);  
    }  
}
```

```
var client = new XMLHttpRequest();  
// Register handler  
client.onload = handler;  
// Construct request  
client.open("GET", "http://example.org/picture.png");  
// Send request  
client.send();
```

Fetch

Modern replacement for XMLHttpRequest

- Makes extensive use of promises for handling asynchrony

`fetch(resource, init)`

`init` is an optional object containing custom settings:

- `method` - GET, POST, etc
- `headers` - headers to be added to request
- `body` - request body
- `mode` - cors, no-cors, same-origin

Fetch example

```
fetch("http://example.org/picture.png")
  .then(response => {
    if (!response.ok) {
      throw new Error("HTTP Status: " + response.status);
    }
    return response.blob();
  })
  .then(blob => {
    // do something with the fetched resource
  })
  .catch(error => {
    console.error("Fetch failed:", error);
  });
```

Web Sockets

Web Sockets

Three issues with XMLHttpRequest:

1. Connection is not persistent
Repeated requests require TCP setup and teardown
2. Communication always initiated by client
No pushing of messages from the server
3. Bound only to HTTP/HTTPS

Web Sockets is a modern replacement for XMLHttpRequest

- Supports multiple transport protocols

Web Sockets example

```
var connection = new WebSocket('ws://example.org/srv',  
                               ['http', 'xmpp']);  
  
connection.onmessage = function (e) {  
    console.log('Server: ' + e.data);  
};  
  
connection.send('...data...');
```

Web Storage

Web Storage

Cookies used to store key-value data in the browser

- HTTP-based mechanism (Cookie: header)
- Breaks stateless nature of HTTP

Web Storage is a more principled replacement

- Separate storage area for each origin (web page)
- Non-persistent storage (`window.sessionStorage`)
- Persistent storage (`window.localStorage`)

Web Storage example

```
localStorage.setItem('email', 'fred@example.org');  
localStorage.getItem('visitCount');  
  
sessionStorage.getItem('query');
```

IndexedDB

IndexedDB

Web Storage API only useful for key-value data

IndexedDB is a more sophisticated web browser database:

- Asynchronous
- Transaction support
- Structured (JSON) data (c.f. CouchDB, MongoDB, etc)

IndexedDB example

```
var db;
var request = indexedDB.open("books");

request.onsuccess = function() {
  db = request.result;
};

var trans = db.transaction("books", "readwrite");
var store = trans.objectStore("books");

store.put({title: "HTML5 for Dummies", isbn: 123456});
store.put({title: "Starting HTML5", isbn: 234567});
store.put({title: "HTML5 Revealed", isbn: 345678});

trans.oncomplete = function() {
  // Called when all requests have succeeded
  // (the transaction has committed)
};
```

Web Workers and Service Workers

Web Workers

Trend in Web scripting is towards asynchrony

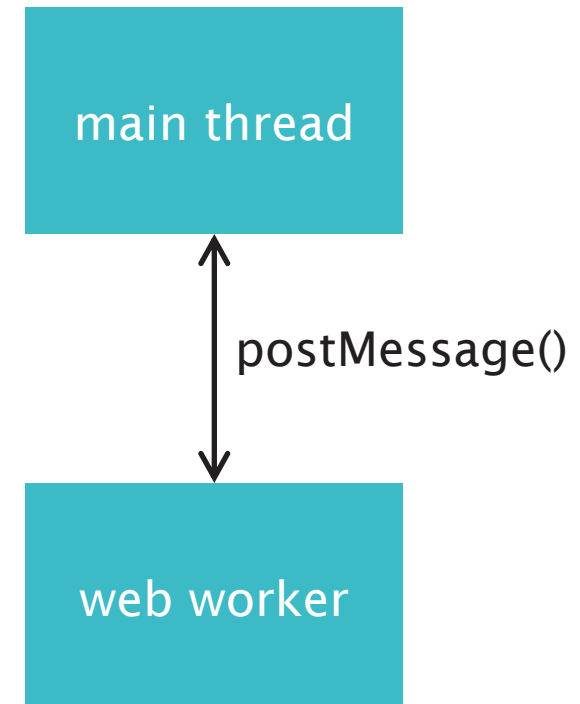
- XMLHttpRequest
- Web Sockets
- Web Storage

JavaScript browser environment is single-threaded

- Compute-intensive tasks affect responsiveness of scripts

Web Workers provides multi-threading for JavaScript

- Asynchronous handling of results



Web Workers example

```
// Main thread:
```

```
const searcher = new Worker('searcher.js');  
searcher.onmessage = function (event) {  
  // process response from the worker thread  
};
```

```
// send message to worker  
searcher.postMessage(query);
```

```
// searcher.js:
```

```
onmessage = function (event) {  
  // process message received from the main thread  
  
  // send response to main thread  
  self.postMessage();  
};
```

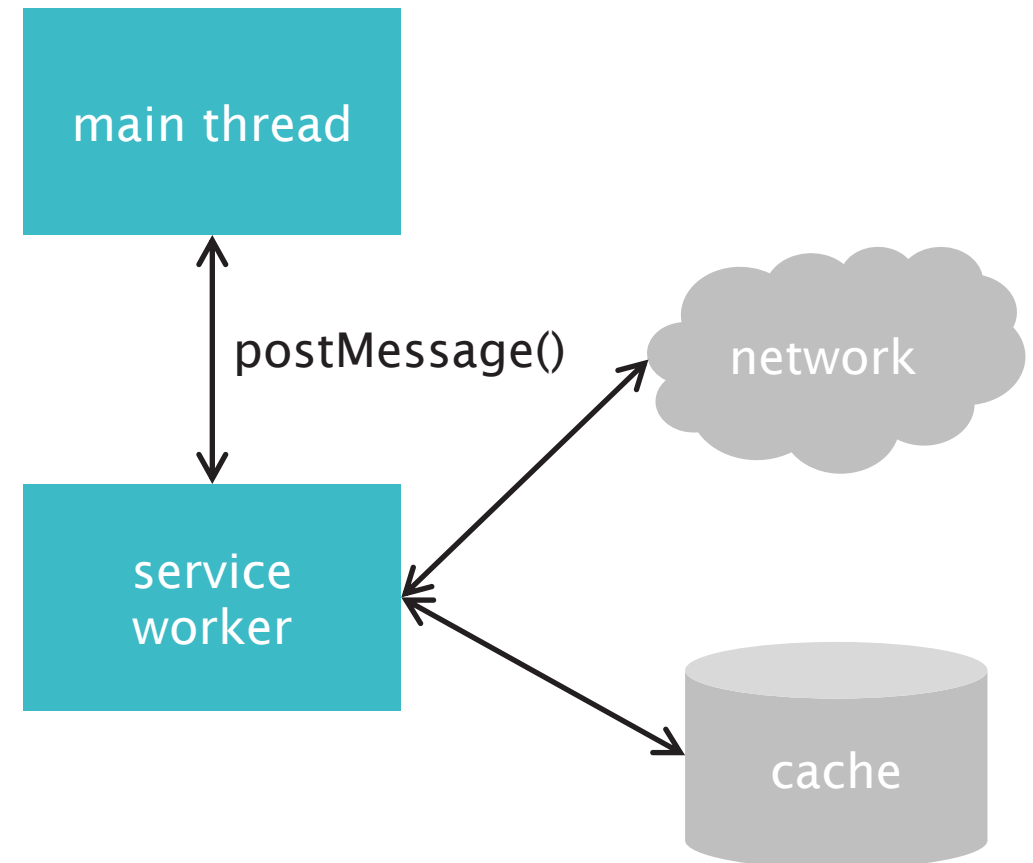


Service Workers

Background script similar to Web Workers

Designed to proxy fetch requests

- Service workers registered with a scope
- fetch requests for resources within that scope are passed to the worker



Service Workers example

```
//main.js
```

```
navigator.serviceWorker.register("/worker.js")  
  .then(reg => { console.log("Registration succeeded for scope:", reg.scope); })  
  .catch(err => { console.error("Registration failed:", err)});
```

```
// worker.js
```

```
self.addEventListener("install", event => {  
  // code that runs when the service worker is registered (set up cache, etc)  
});
```

```
self.addEventListener("fetch", function(event) {  
  // code that runs when a fetch is executed in the scope of this service worker  
  // (return cached resource if available, otherwise execute fetch)  
});
```

Geolocation

Geolocation

Allows a script to determine client location

- One-off (`getCurrentPosition()`)
- Ongoing (`watchPosition()`)

Location information service independent

- GPS, GSM/CDMA, Wi-Fi

```
navigator.geolocation.getCurrentPosition(success);
```

```
function success(pos) {  
    console.log("Latitude: " + pos.coords.latitude);  
    console.log("Longitude: " + pos.coords.longitude);  
}
```

Further reading

WHATWG (2020) *DOM Living Standard*.

<https://dom.spec.whatwg.org/>

Jackson, D. and Gilbert, J. (2020) *WebGL Specification*. Beaverton, OR: Khronos Group.

<https://www.khronos.org/registry/webgl/specs/latest/1.0/>

WHATWG (2020) *XMLHttpRequest Living Standard*.

<https://xhr.spec.whatwg.org/>

WHATWG (2020) *Fetch Living Standard*.

<https://fetch.spec.whatwg.org/>

Fette, I. and Melnikov, A. (2011) *The Web Socket Protocol*. RFC6455.

<https://tools.ietf.org/html/rfc6455>

WHATWG (2020) *HTML Living Standard: Web Sockets*.

<https://html.spec.whatwg.org/multipage/web-sockets.html>

Further reading

WHATWG (2020) *HTML Living Standard: Web Storage*.

<https://html.spec.whatwg.org/multipage/webstorage.html>

Alabbas, A. and Bell, J. (2018) *Indexed Database API 2.0*. W3C Recommendation.

<https://www.w3.org/TR/IndexedDB-2/>

WHATWG (2020) *HTML Living Standard: Workers*.

<https://html.spec.whatwg.org/multipage/workers.html>

Russell, A. et al (2019) *Service Workers*. W3C Candidate Recommendation.

<https://www.w3.org/TR/service-workers/>

Popescu, A. (2018) *Geolocation API Specification 2nd Edition*. W3C Recommendation.

<https://www.w3.org/TR/geolocation-API/>

Next Lecture: Web Assembly