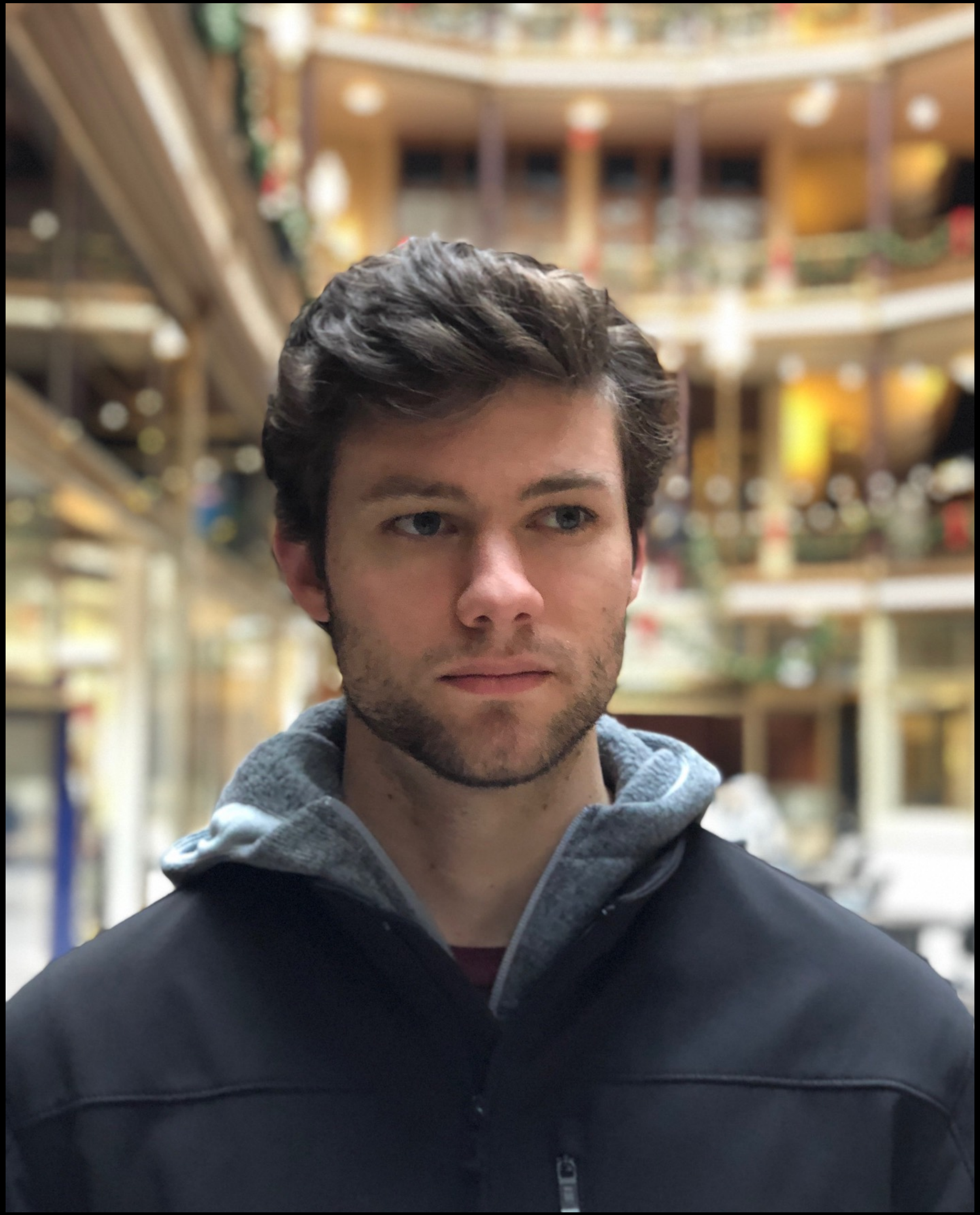
Hitchhiker’s Guide to Web Standards

Dominic Farolino

- Senior @ University of Cincinnati
- Previously Microsoft, Mozilla
- Incoming @ Google
- Chromium Committer
- WHATWG Editor

- twitter.com/domfarolino
- github.com/domfarolino
- dom@chromium.org



What is this talk?

Background & History

(this could really be its own talk)

Intro to technical bits of specs

How to get involved?

(you can do it!!)

What is a “standard”?

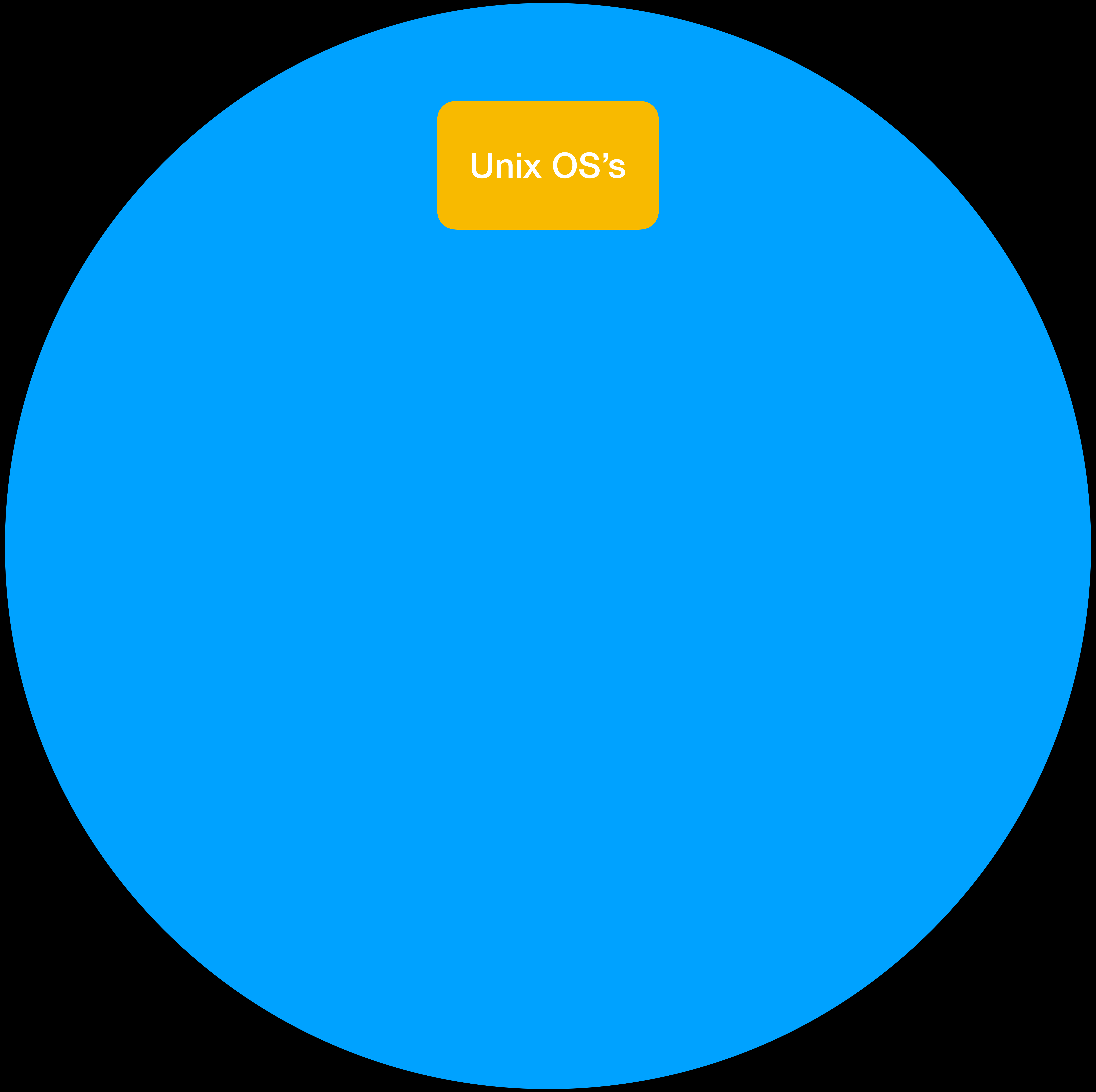
Depends on the context

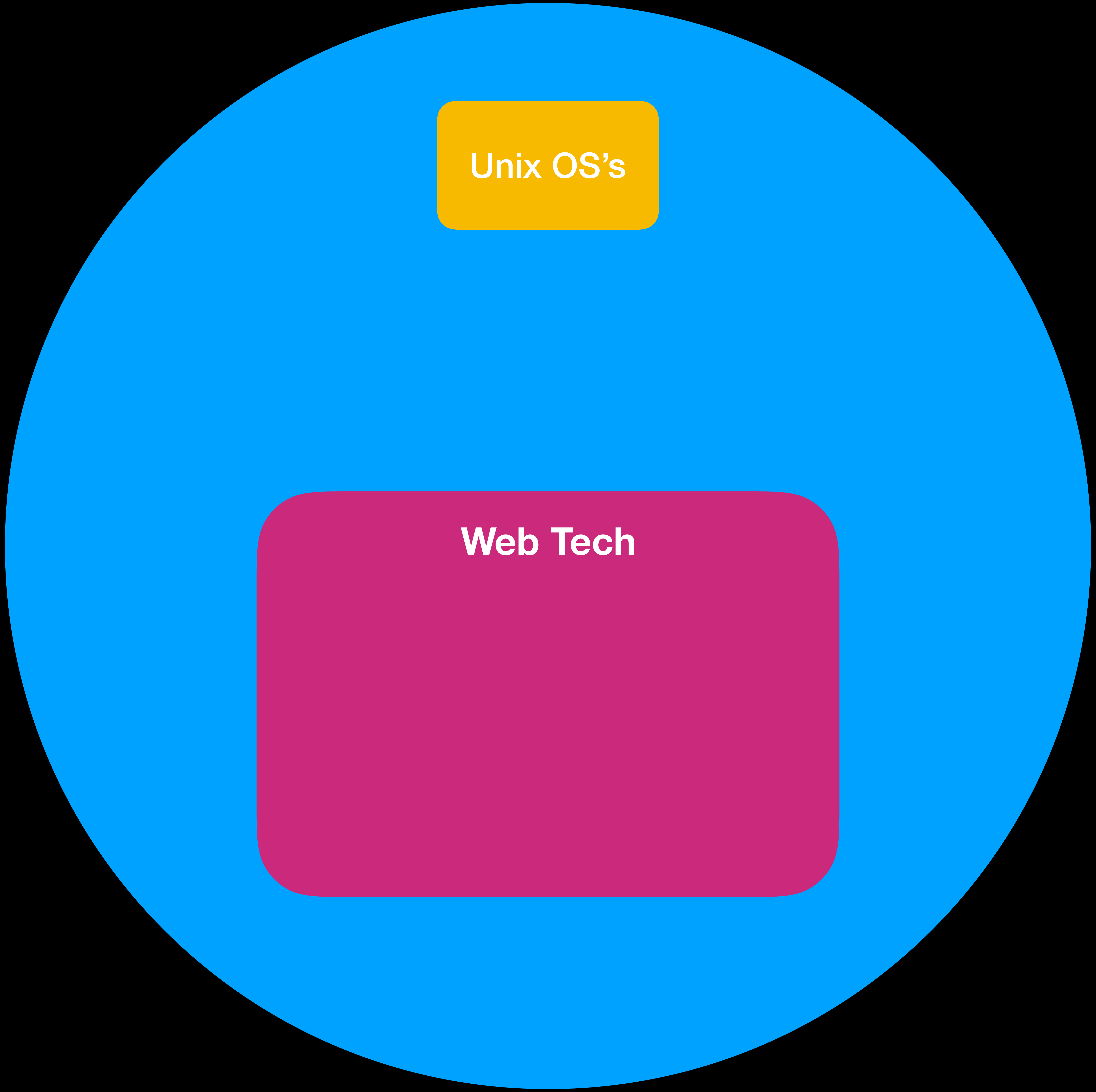
“A document specifying observable effects of tech with multiple independent implementations”

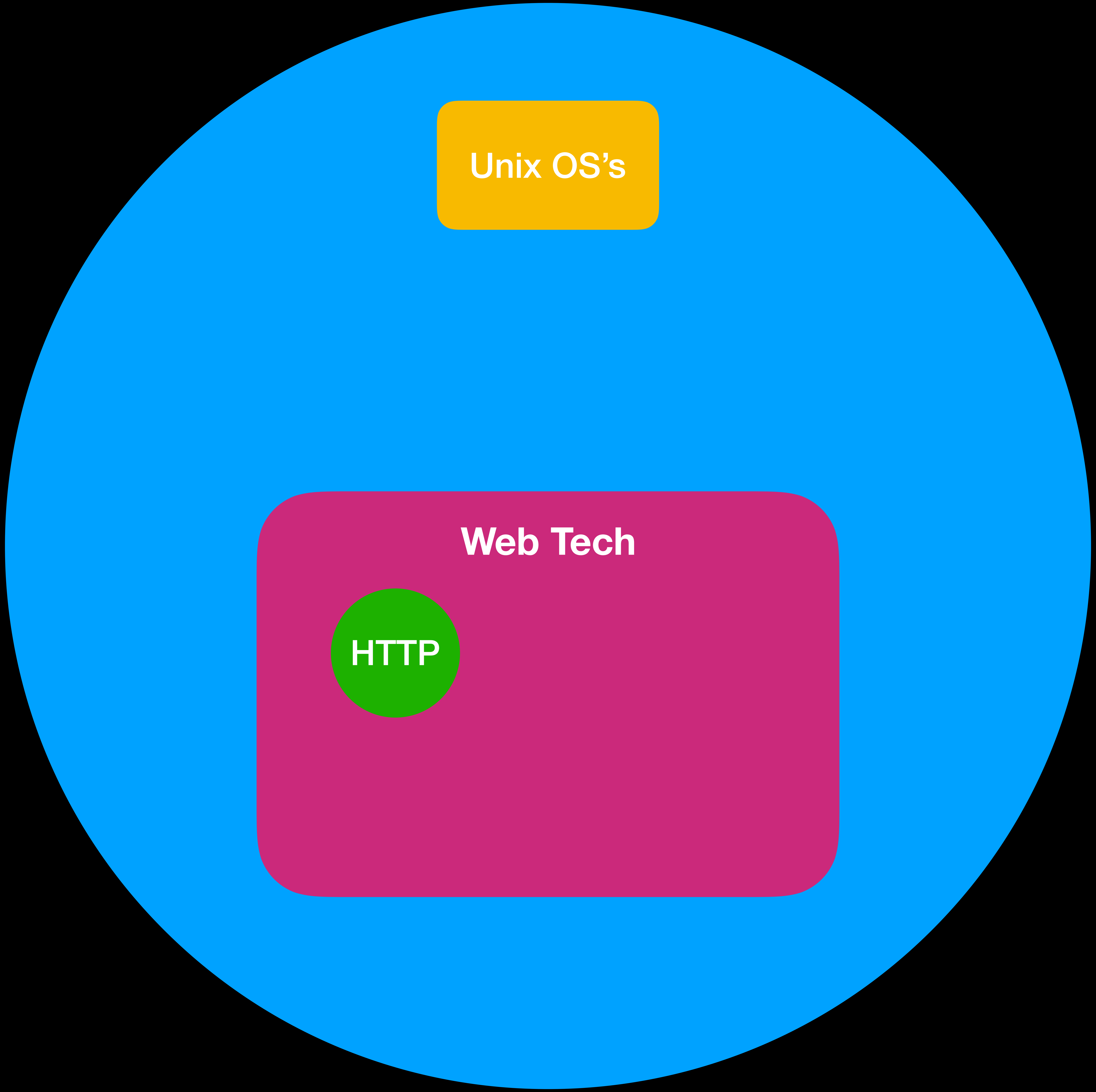
–yours truly

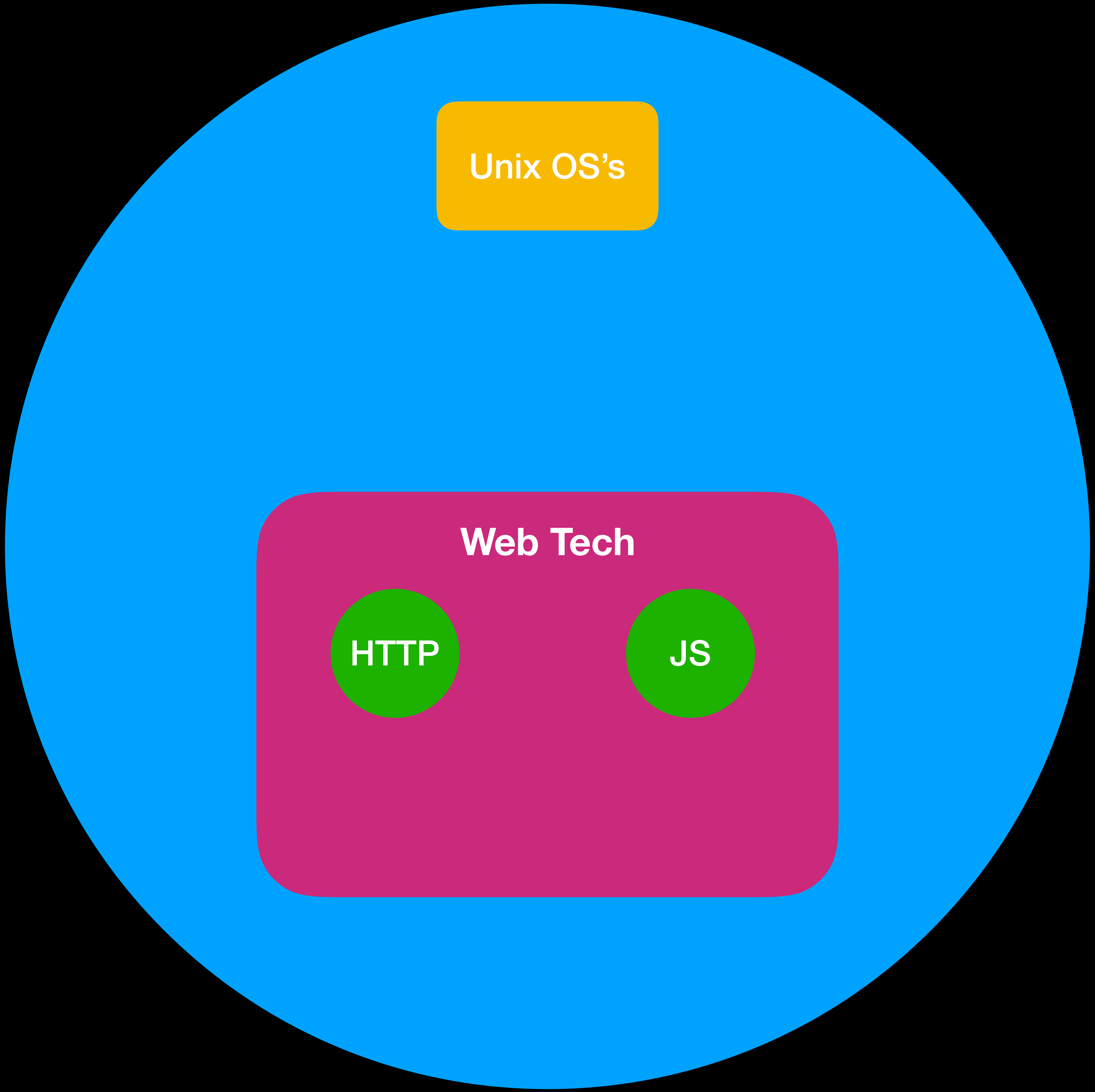
“A document specifying observable effects of tech with multiple independent implementations”

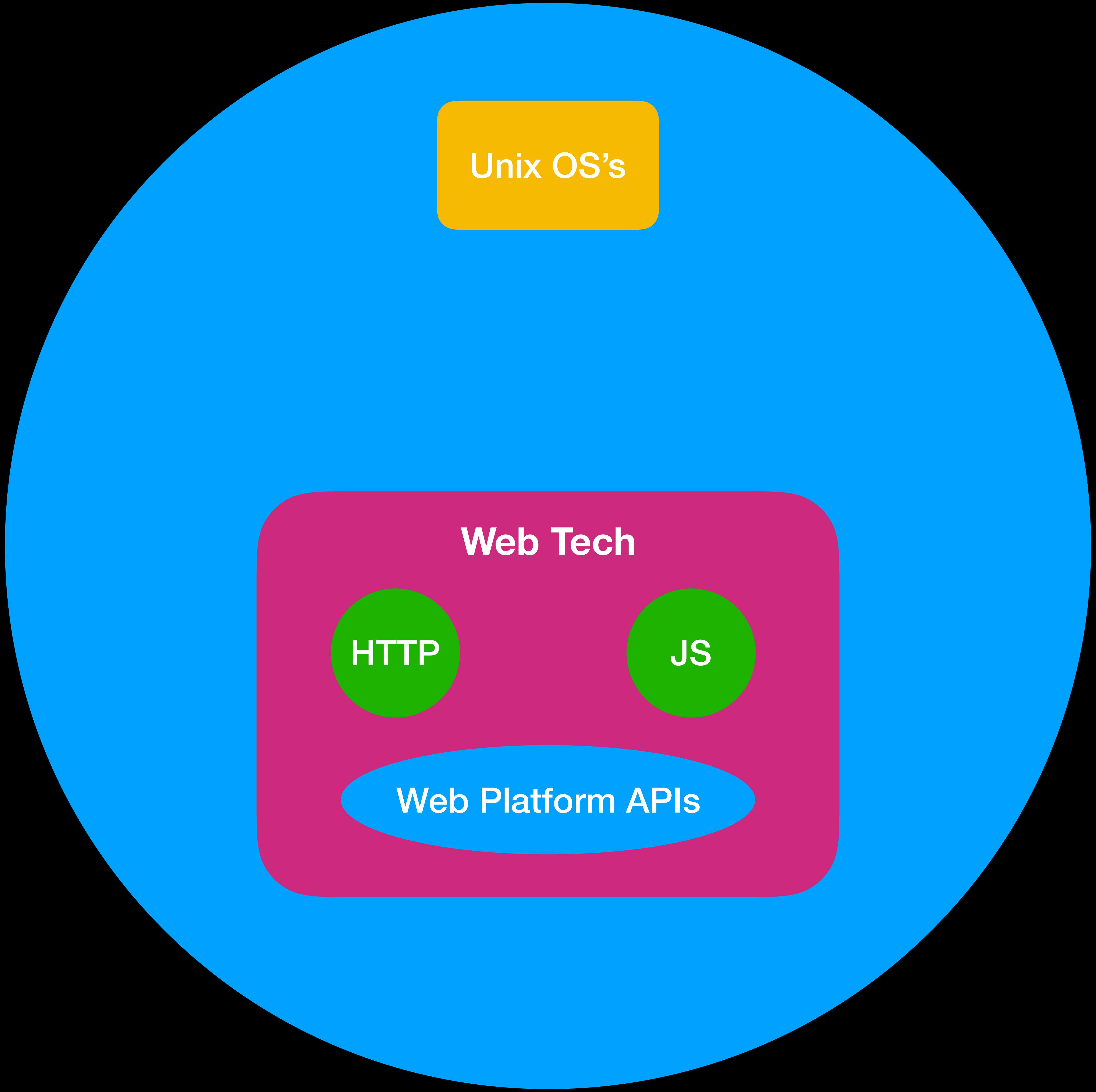
–yours truly

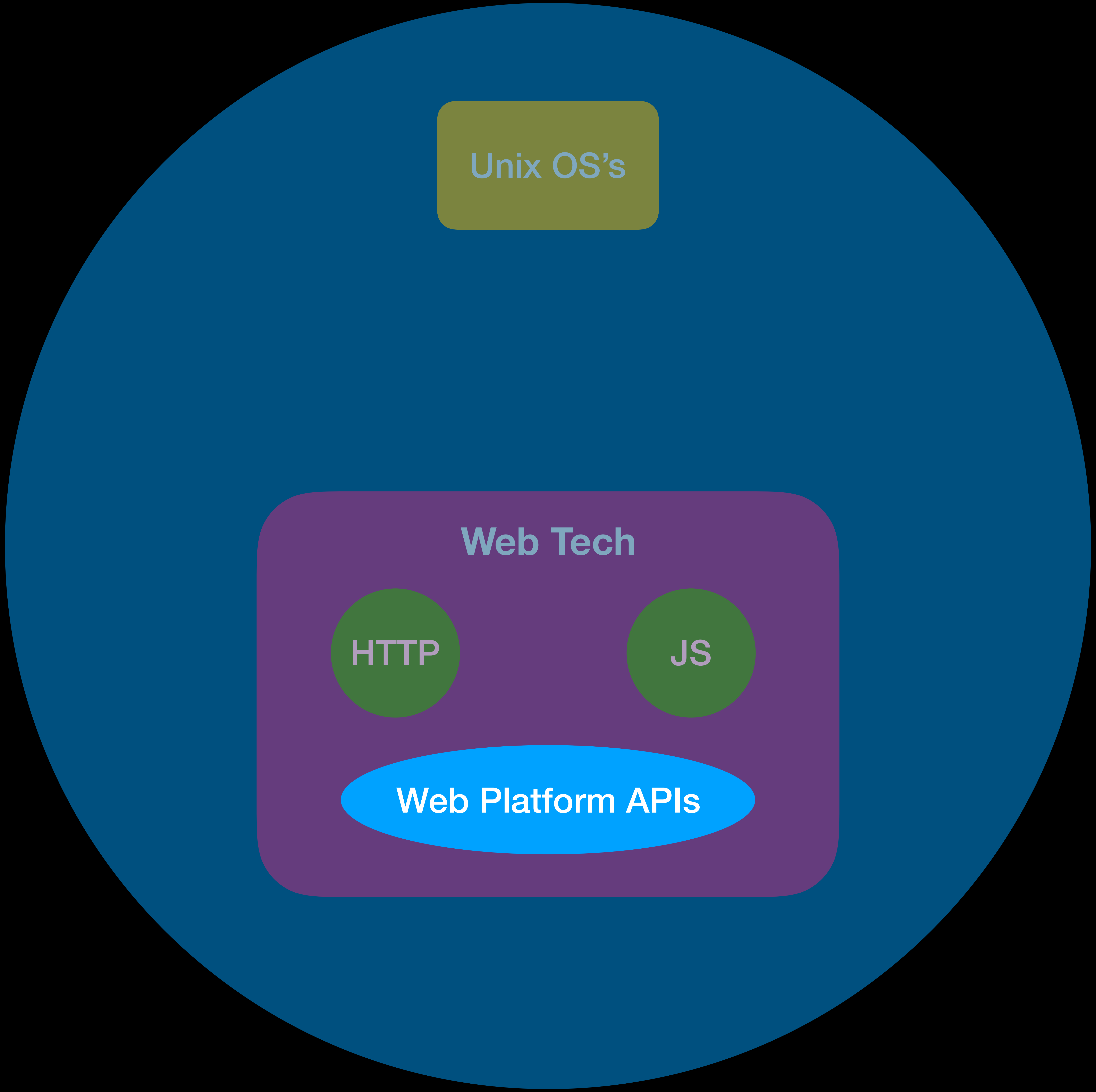

Unix OS’s

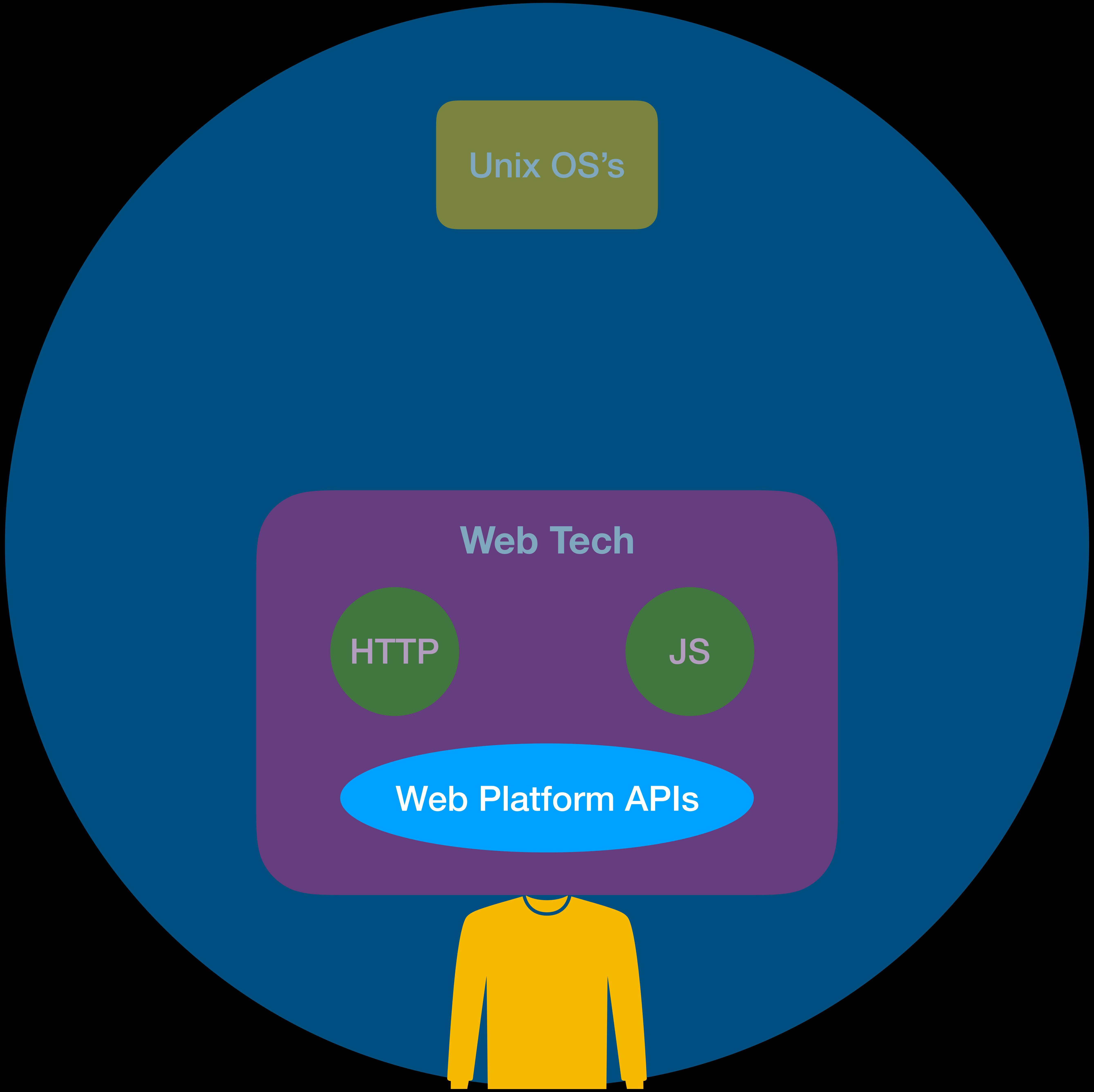

Unix OS’s
Web Tech

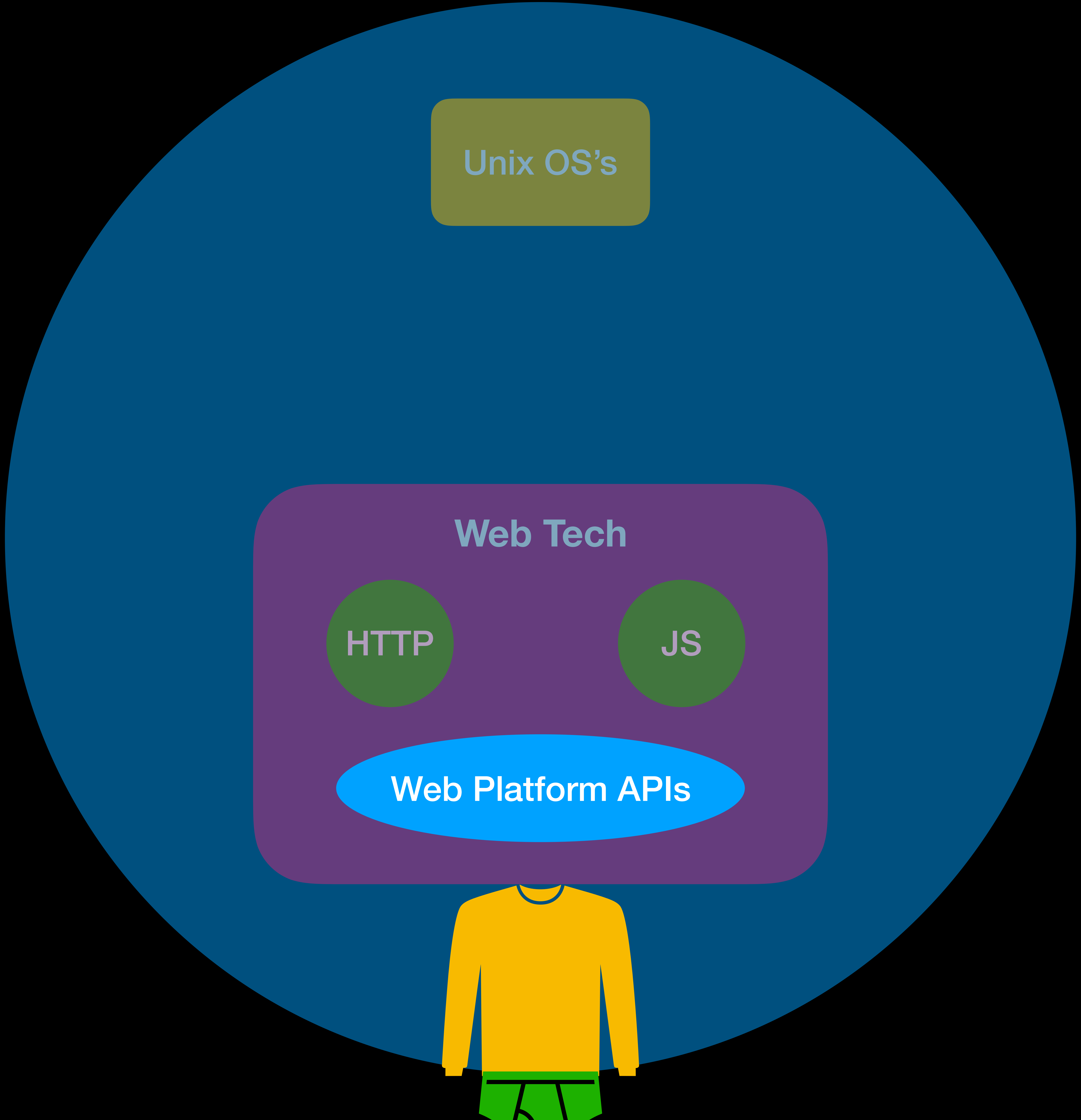

Unix OS's
Web Tech
HTTP

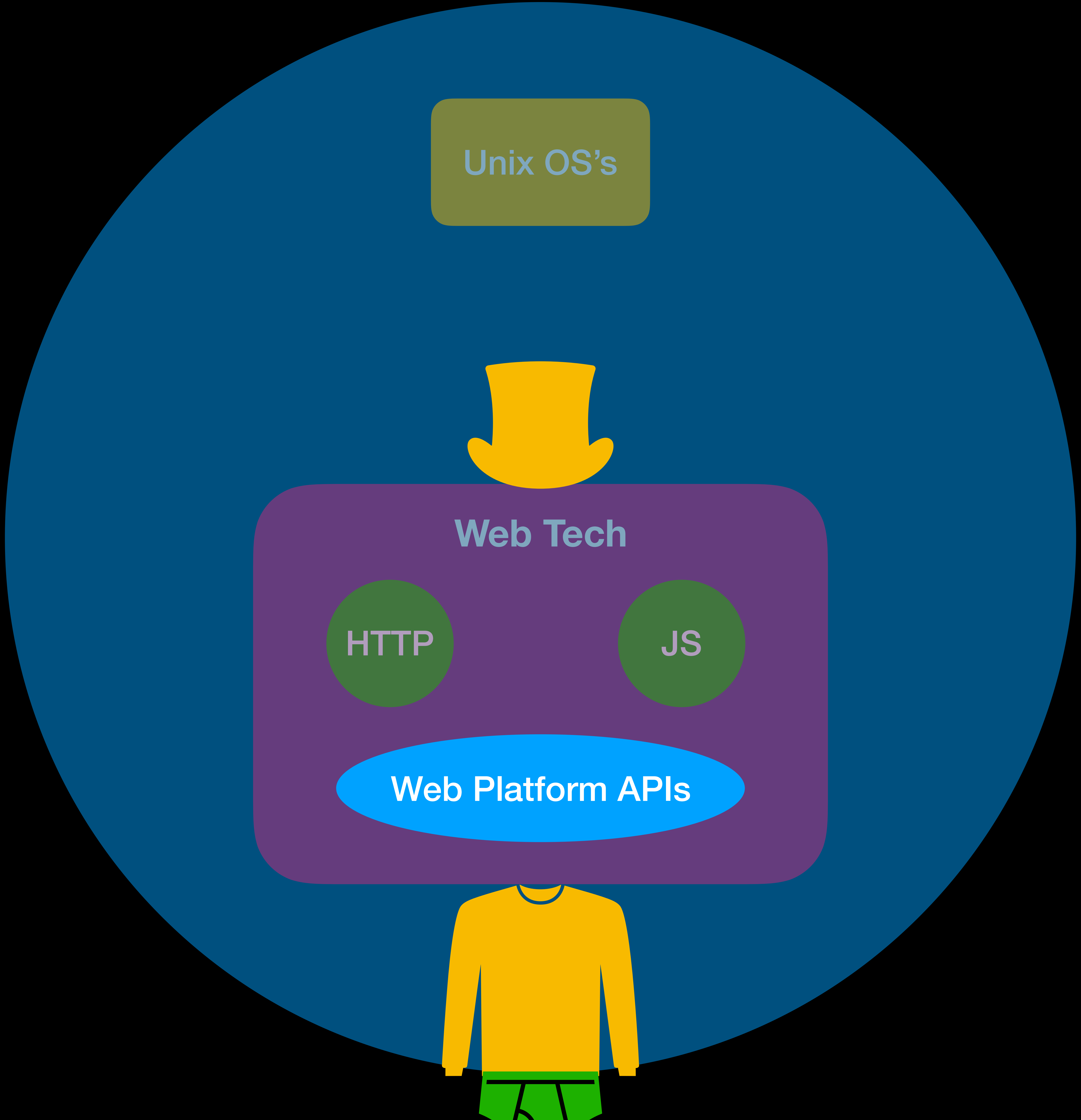

Unix OS's
Web Tech
HTTP
JS


Unix OS's
Web Tech
HTTP
JS
Web Platform APIs


Unix OS's
Web Tech
HTTP
JS
Web Platform APIs

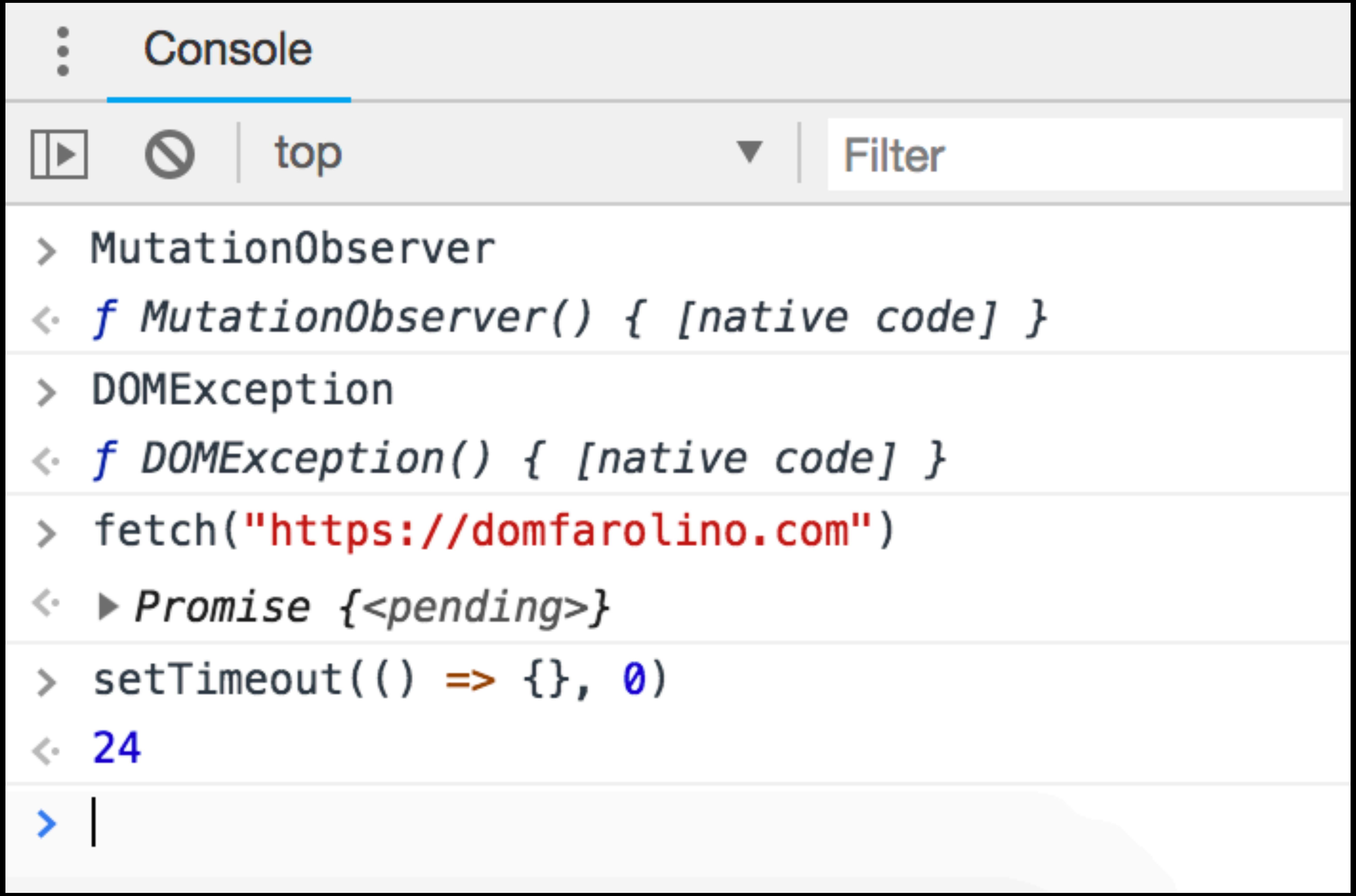

Unix OS's
Web Tech
HTTP
JS
Web Platform APIs


Unix OS's
Web Tech
HTTP
JS
Web Platform APIs

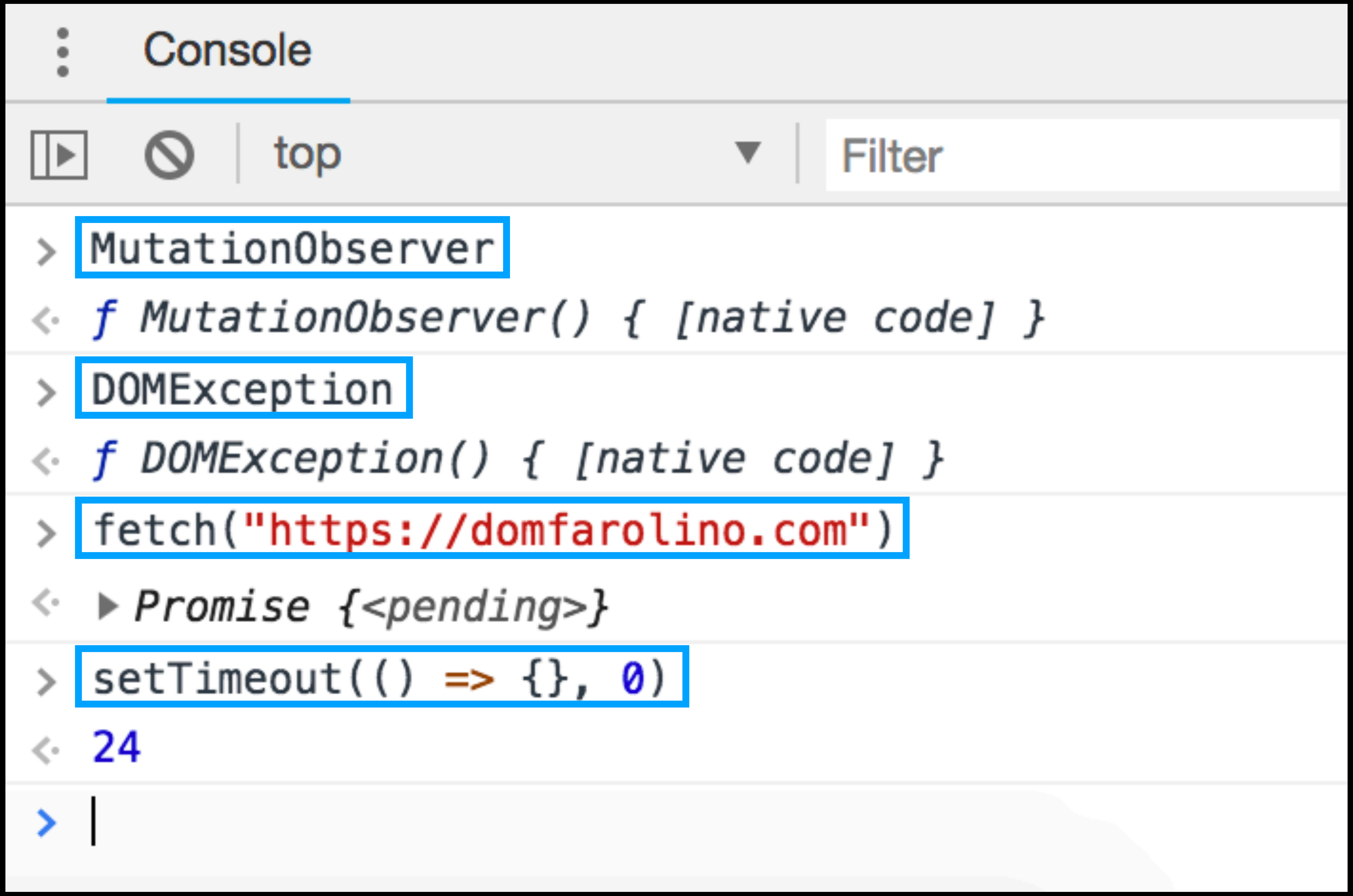

Unix OS's
Web Tech
HTTP
JS
Web Platform APIs

Where do Web APIs come from?

```
const logicalAnswer = “JavaScript”
```



```
> MutationObserver
<· ƒ MutationObserver() { [native code] }
> DOMException
<· ƒ DOMException() { [native code] }
> fetch("https://domfarolino.com")
<· ▶ Promise {<pending>}
> setTimeout(() => {}, 0)
<· 24
> |
```



```
> MutationObserver
<· ƒ MutationObserver() { [native code] }
> DOMException
<· ƒ DOMException() { [native code] }
> fetch("https://domfarolino.com")
<· ▶ Promise {<pending>}
> setTimeout(() => {}, 0)
<· 24
> |
```

What is JavaScript

- Just a language
- History
- Standardized by ECMAScript
- Multiple independent implementations exist

ES Engines

- V8 (Chrome)
- Chakra (Edge)
- SpiderMonkey (Firefox)
- JavaScriptCore (Safari/WebKit)

Search...

TABLE OF CONTENTS

Introduction
1 Scope
2 Conformance
3 Normative References
4 Overview
5 Notational Conventions
6 ECMAScript Data Types and Values
7 Abstract Operations
8 Executable Code and Execution Contexts
9 Ordinary and Exotic Objects Behaviours
10 ECMAScript Language: Source Code
11 ECMAScript Language: Lexical Grammar
12 ECMAScript Language: Expressions
13 ECMAScript Language: Statements and Declarations
14 ECMAScript Language: Functions and Classes
15 ECMAScript Language: Scripts and Modules
16 Error Handling and Language Extensions
17 ECMAScript Standard Built-in Objects
18 The Global Object
19 Fundamental Objects
20 Numbers and Dates
21 Text Processing
22 Indexed Collections
23 Keyed Collections
24 Structured Data
25 Control Abstraction Objects
26 Reflection
27 Memory Model
A Grammar Summary
B Additional ECMAScript Features for Web Browsers

Google Chrome

Draft ECMA-262 / November 1, 2018

ECMAScript® 2019 Language Specification

ecma INTERNATIONAL

Contributing to this Specification

This specification is developed on GitHub with the help of the ECMAScript community. There are a number of ways to contribute to the development of this specification:

GitHub Repository: https://github.com/tc39/ecma262
Issues: All Issues, File a New Issue
Pull Requests: All Pull Requests, Create a New Pull Request
Test Suite: Test262
Editors:

- Brian Terlson (@bterlson)
- Bradley Farias (@bradleymeck)
- Jordan Harband (@ljharb)

Community:

- Mailing list: es-discuss
- IRC: #tc39 on freenode

Refer to the colophon for more information on how this document is created.

Search...
Google Chrome

TABLE OF CONTENTS

Introduction
1 Scope
2 Conformance
3 Normative References
4 Overview
5 Notational Conventions
6 ECMAScript Data Types and Values
7 Abstract Operations
8 Executable Code and Execution Contexts
9 Ordinary and Exotic Objects Behaviours
10 ECMAScript Language: Source Code
11 ECMAScript Language: Lexical Grammar
12 ECMAScript Language: Expressions
13 ECMAScript Language: Statements and Declarations
14 ECMAScript Language: Functions and Classes
15 ECMAScript Language: Scripts and Modules
16 Error Handling and Language Extensions
17 ECMAScript Standard Built-in Objects
18 The Global Object
19 Fundamental Objects
20 Numbers and Dates
21 Text Processing
22 Indexed Collections
23 Keyed Collections
24 Structured Data
25 Control Abstraction Objects
26 Reflection
27 Memory Model
A Grammar Summary
B Additional ECMAScript Features for Web Browsers

Draft ECMA-262 / November 1, 2018

ECMAScript® 2019 Language Specification

Contributing to this Specification

This specification is developed on GitHub with the help of the ECMAScript community. There are a number of ways to contribute to the development of this specification:

GitHub Repository: https://github.com/tc39/ecma262
Issues: All Issues, File a New Issue
Pull Requests: All Pull Requests, Create a New Pull Request
Test Suite: Test262
Editors:

- Brian Terlson (@bterlson)
- Bradley Farias (@bradleymeck)
- Jordan Harband (@ljharb)

Community:

- Mailing list: es-discuss
- IRC: #tc39 on freenode

Refer to the colophon for more information on how this document is created.

ECMAScript

ECMAScript

- Needed to be a general standard

ECMAScript

- Needed to be a general standard
- Specifies *only* a language; separation of concerns
    - Syntax, semantics, constructs, primitives
    - Language should not know about its environment at all
    - No explicit knowledge of language’s host (DOM / fetch() / etc)

Web API Origins

- Not part of the language itself
- Effectively “mixins”, baked into UAs like browsers
- Browsers “support” implementations of these standards

Web Standards Bodies

WHATWG

WHATWG

- **W**eb **H**ypertext **A**pplication **T**echnology **W**orking **G**roup

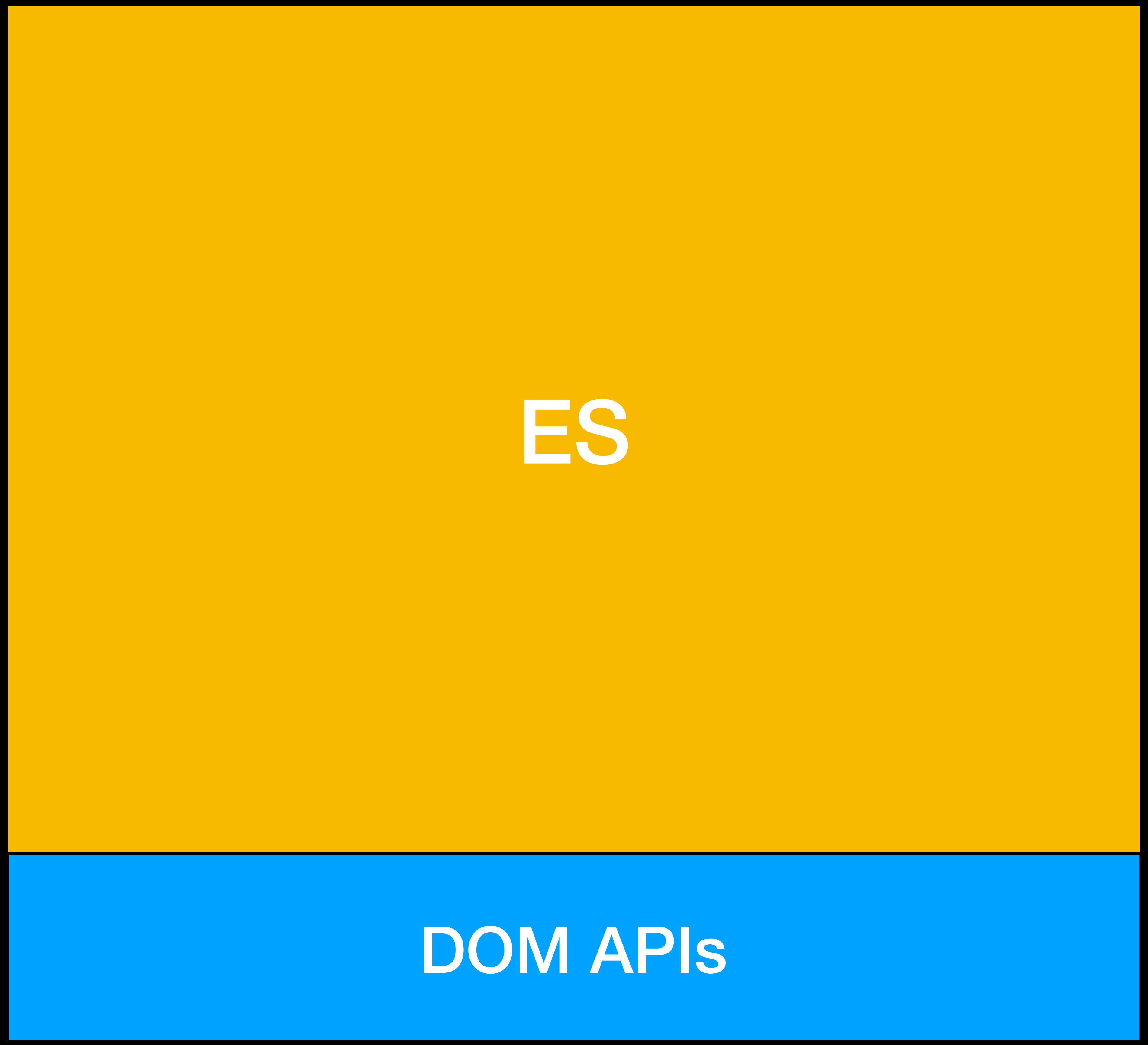
WHATWG

- **W**eb **H**ypertext **A**pplication **T**echnology **W**orking **G**roup
- Formed in 2004

WHATWG

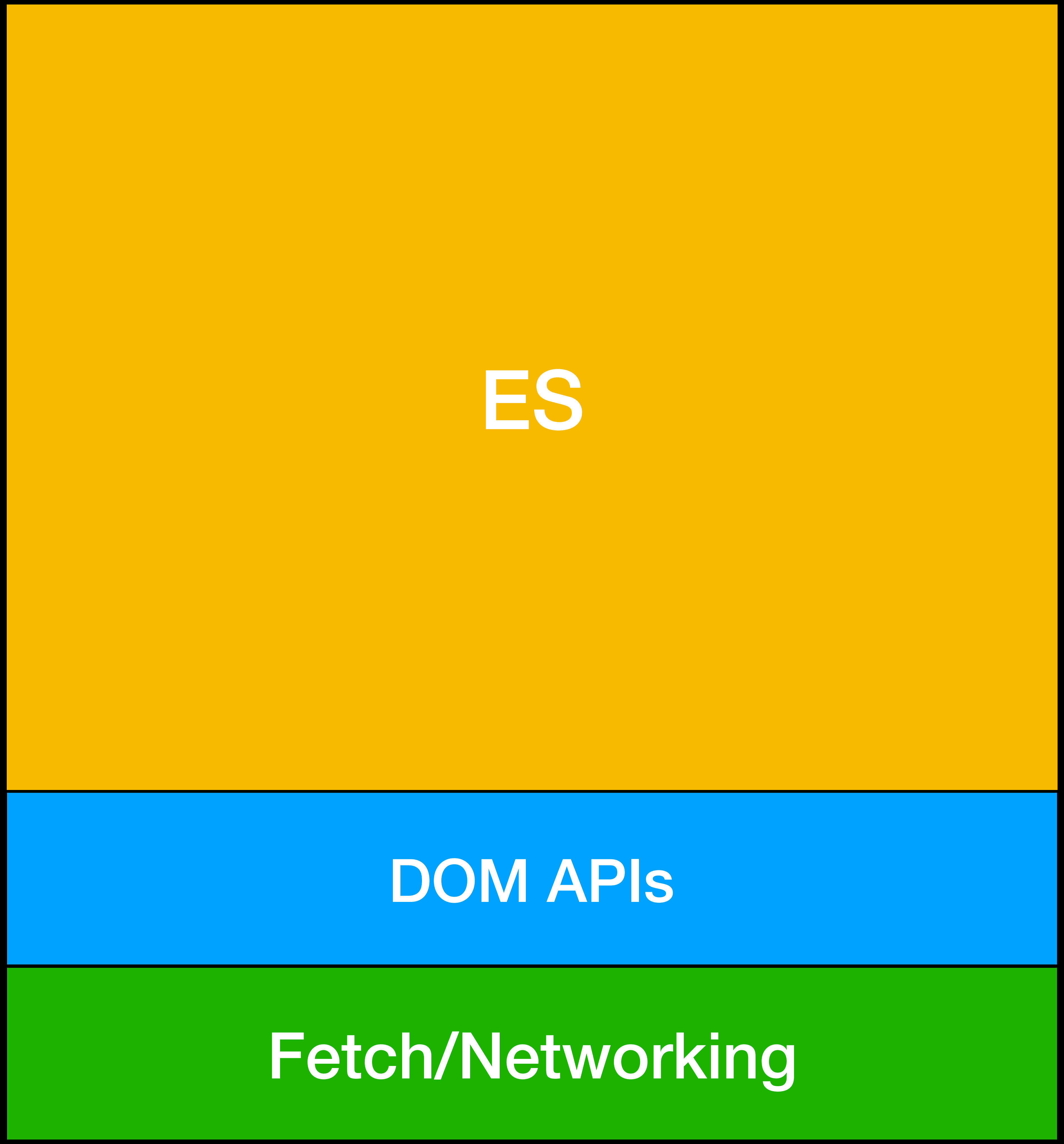
- **W**eb **H**ypertext **A**pplication **T**echnology **W**orking **G**roup
- Formed in 2004
- Canonical standards:
  - HTML
  - DOM
  - Fetch
  - Streams

`Array, WeakMap, Date`



`Array, WeakMap, Date`

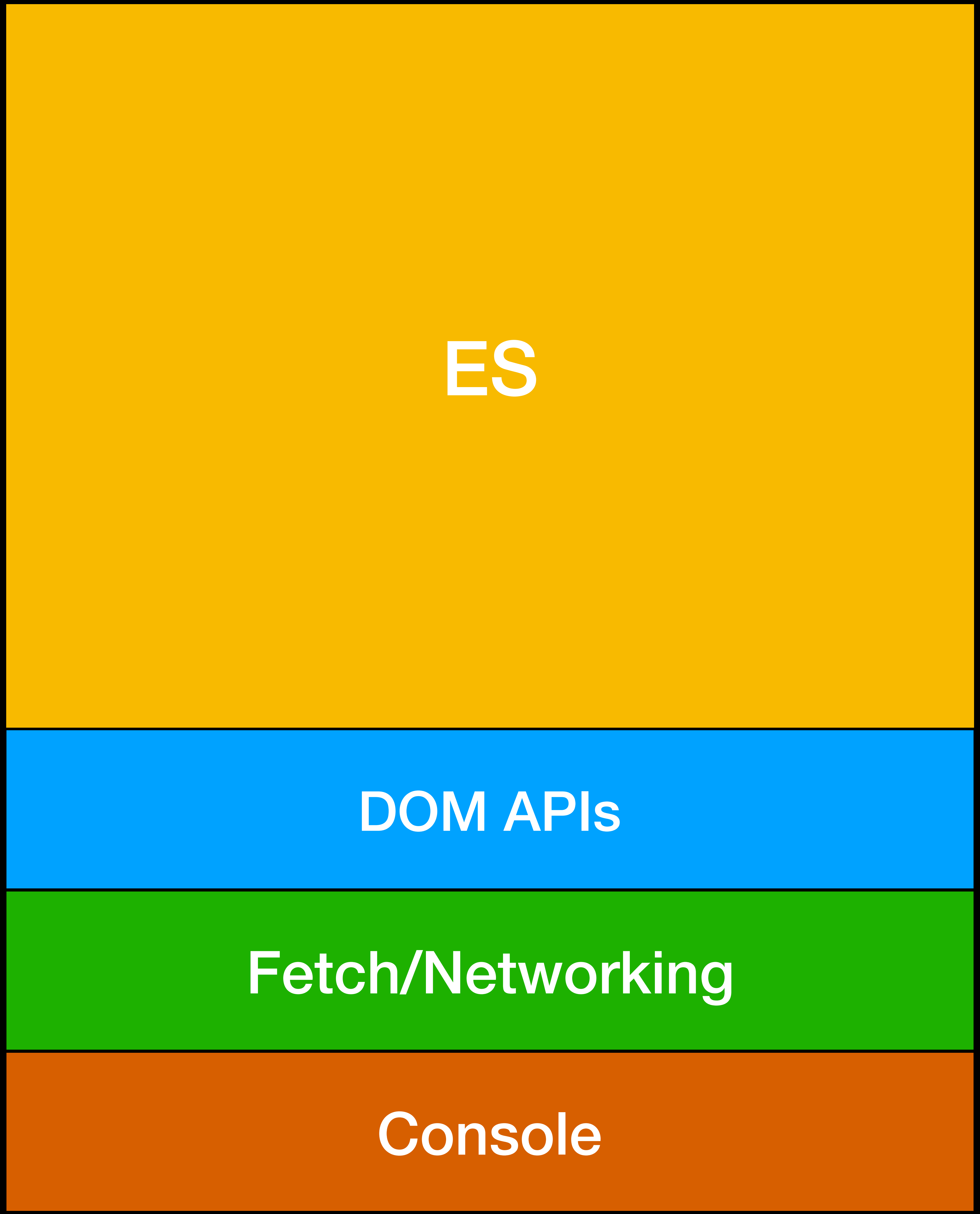
`document.querySelector`



`Array, WeakMap, Date`

`document.querySelector`

`fetch(), Request(), …`



`Array, WeakMap, Date`

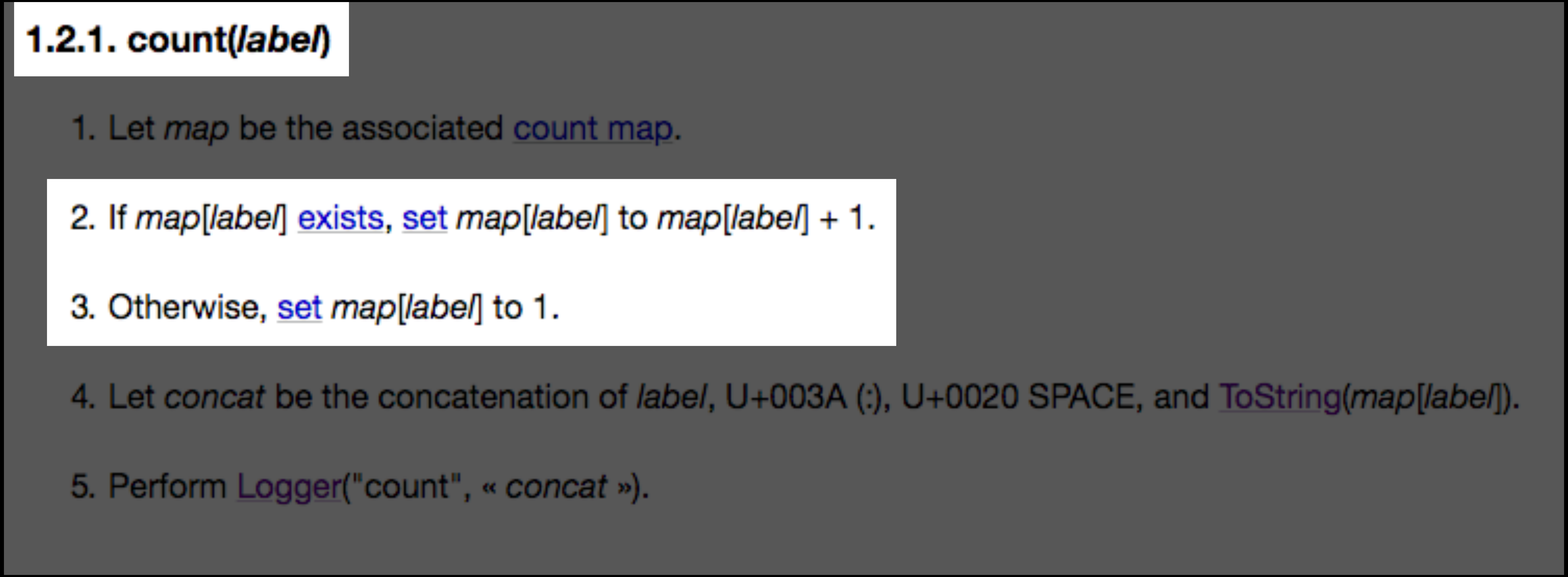
`document.querySelector`

`fetch(), Request(), …`

`console.{log, count, …}`

Anatomy of a standard

Algorithms



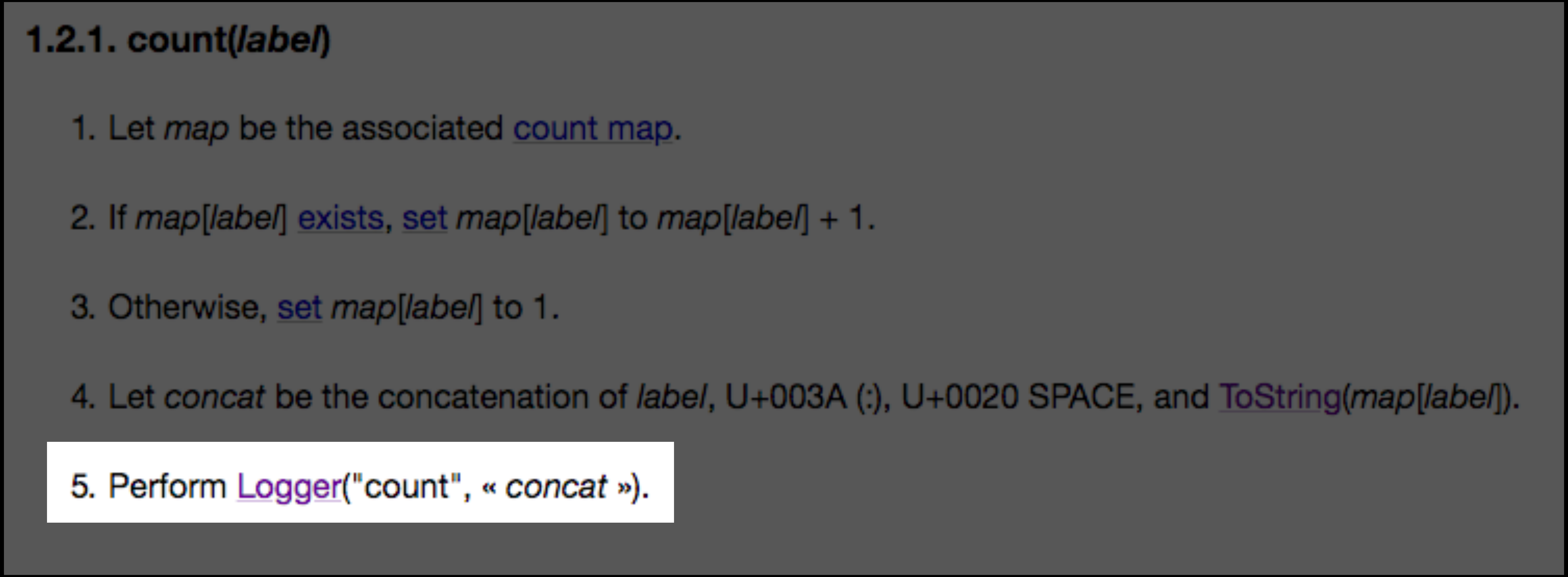
1.2.1. count(*label*)

1. Let *map* be the associated count map.
2. If *map*[*label*] exists, set *map*[*label*] to *map*[*label*] + 1.
3. Otherwise, set *map*[*label*] to 1.
4. Let *concat* be the concatenation of *label*, U+003A (:), U+0020 SPACE, and ToString(*map*[*label*]).
5. Perform Logger("count", « *concat* »).

https://console.spec.whatwg.org/#count

@domfarolino

dom@chromium.org



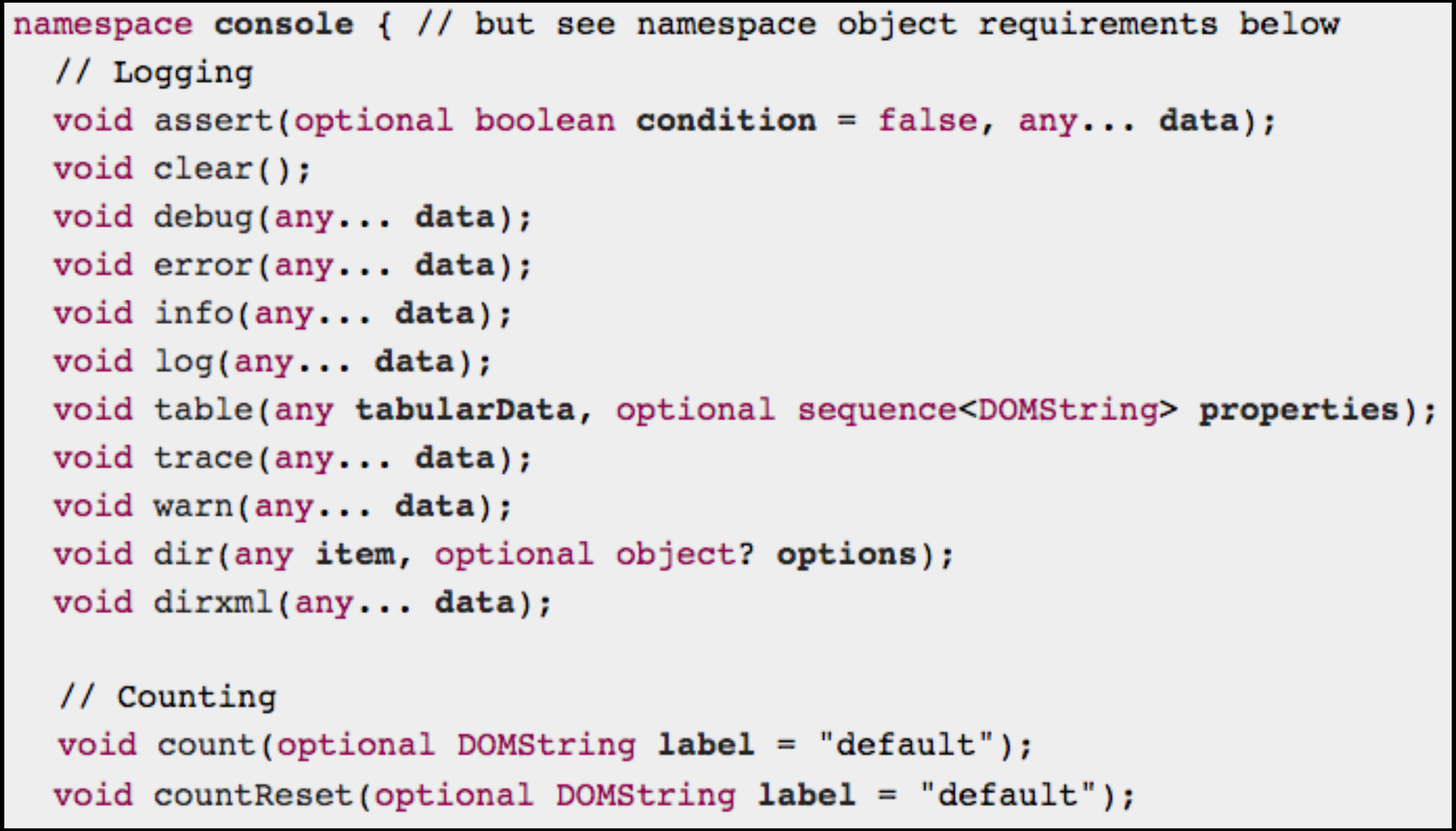
1.2.1. count(*label*)

1. Let *map* be the associated count map.
2. If *map*[*label*] exists, set *map*[*label*] to *map*[*label*] + 1.
3. Otherwise, set *map*[*label*] to 1.
4. Let *concat* be the concatenation of *label*, U+003A (:), U+0020 SPACE, and ToString(*map*[*label*]).
5. Perform Logger("count", « *concat* »).

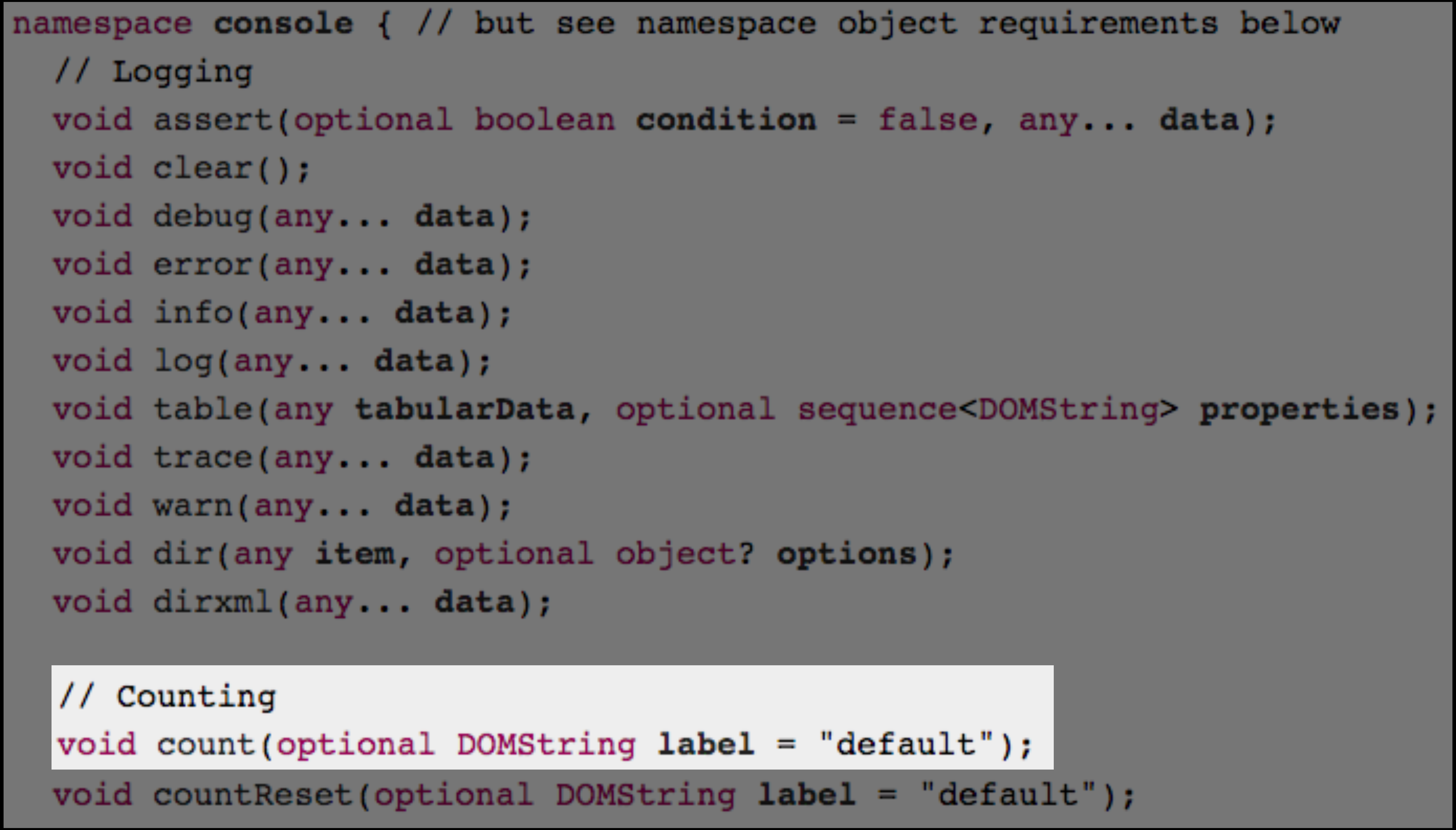
https://console.spec.whatwg.org/#count

@domfarolino dom@chromium.org

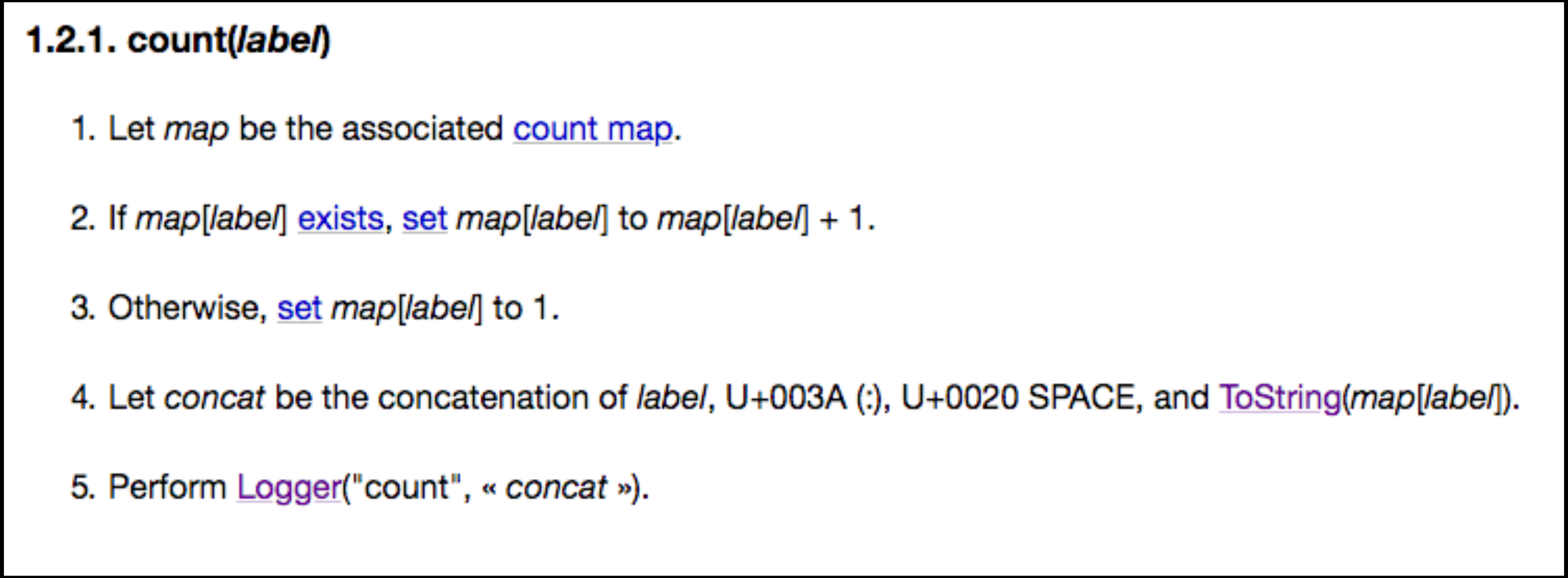
Look and Feel



```
namespace console { // but see namespace object requirements below
  // Logging
  void assert(optional boolean condition = false, any... data);
  void clear();
  void debug(any... data);
  void error(any... data);
  void info(any... data);
  void log(any... data);
  void table(any tabularData, optional sequence<DOMString> properties);
  void trace(any... data);
  void warn(any... data);
  void dir(any item, optional object? options);
  void dirxml(any... data);

  // Counting
  void count(optional DOMString label = "default");
  void countReset(optional DOMString label = "default");
```



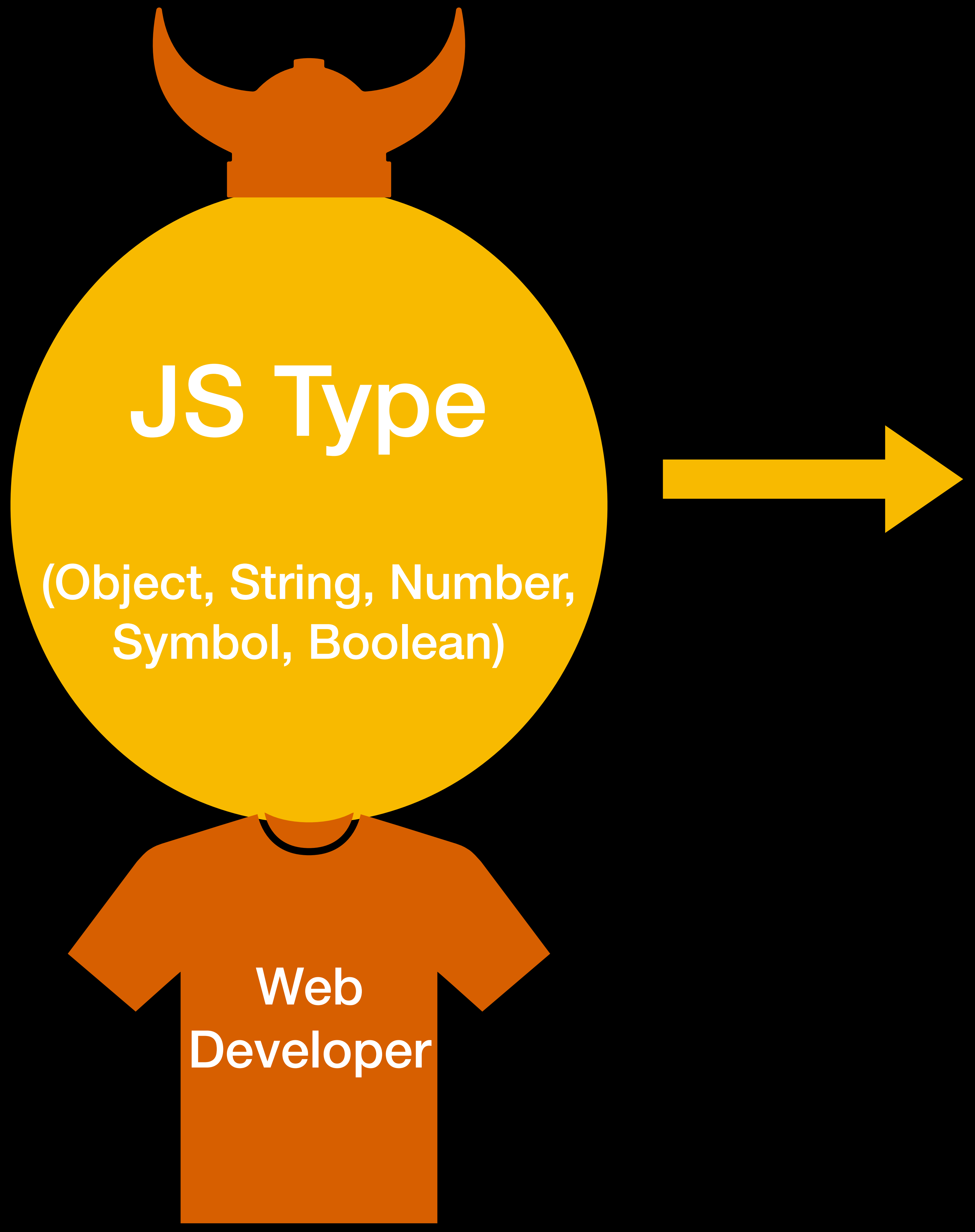
```
namespace console { // but see namespace object requirements below
  // Logging
  void assert(optional boolean condition = false, any... data);
  void clear();
  void debug(any... data);
  void error(any... data);
  void info(any... data);
  void log(any... data);
  void table(any tabularData, optional sequence<DOMString> properties);
  void trace(any... data);
  void warn(any... data);
  void dir(any item, optional object? options);
  void dirxml(any... data);

  // Counting
  void count(optional DOMString label = "default");
  void countReset(optional DOMString label = "default");
```



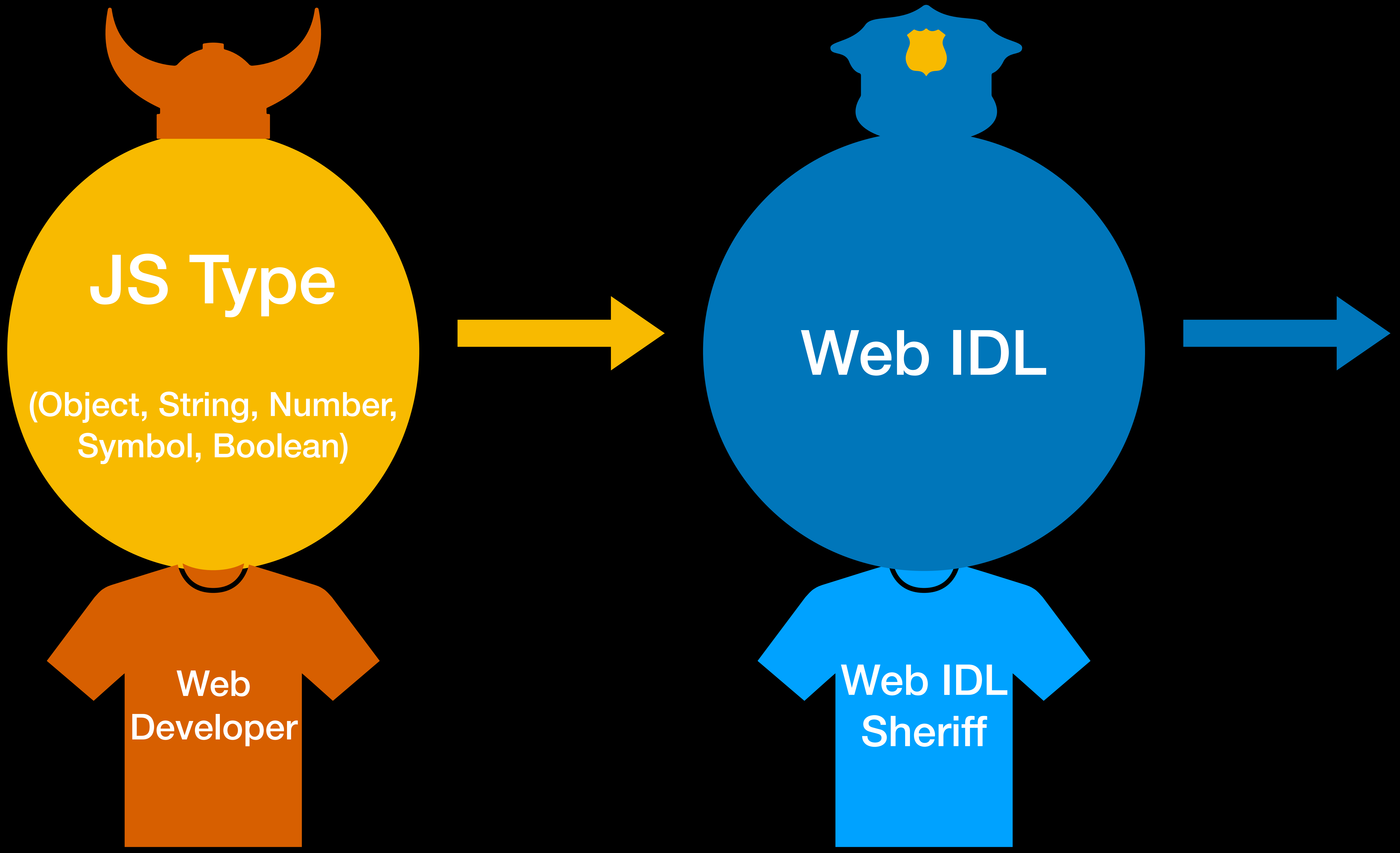
1.2.1. count(*label*)

1. Let *map* be the associated count map.
2. If *map*[*label*] exists, set *map*[*label*] to *map*[*label*] + 1.
3. Otherwise, set *map*[*label*] to 1.
4. Let *concat* be the concatenation of *label*, U+003A (:), U+0020 SPACE, and ToString(*map*[*label*]).
5. Perform Logger("count", « *concat* »).

Web IDL Conversion

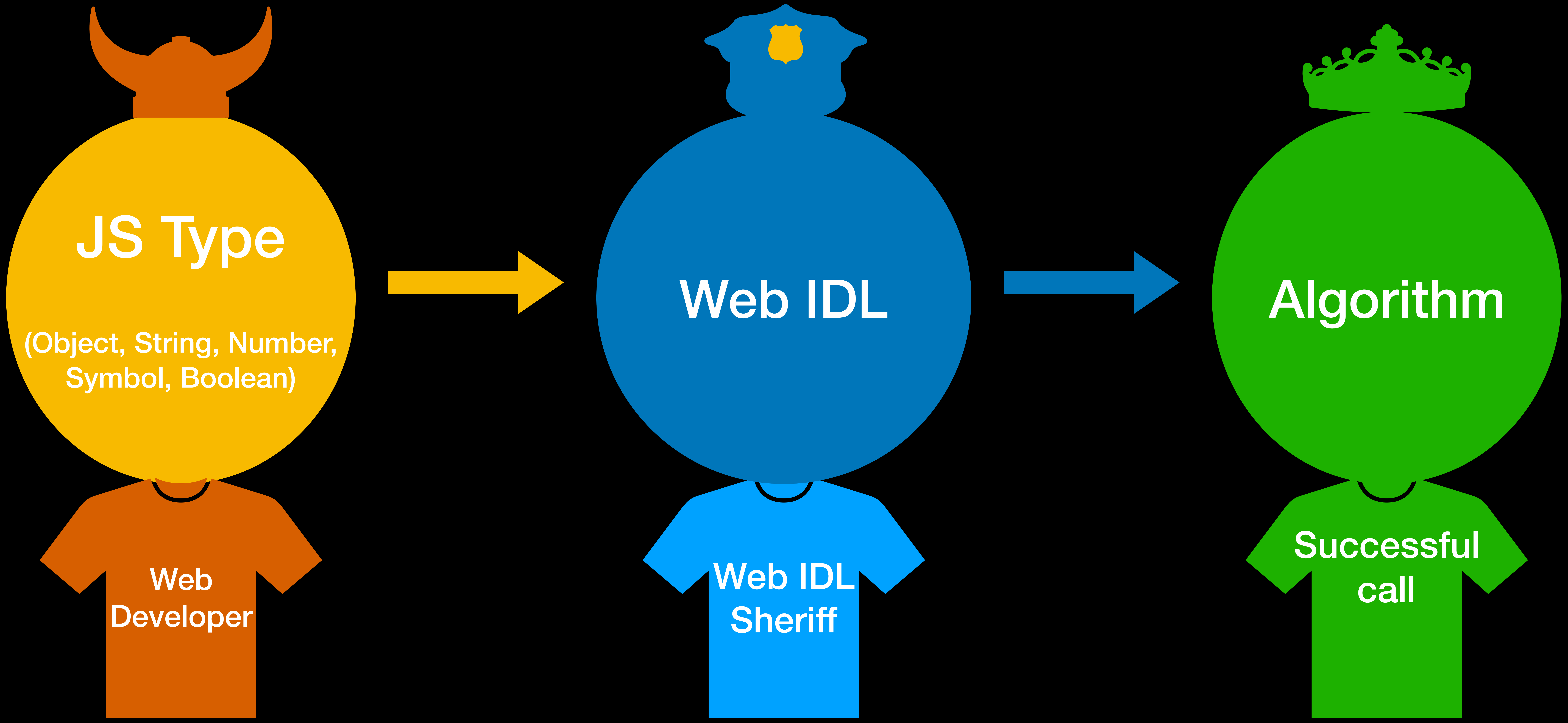


Web IDL Conversion



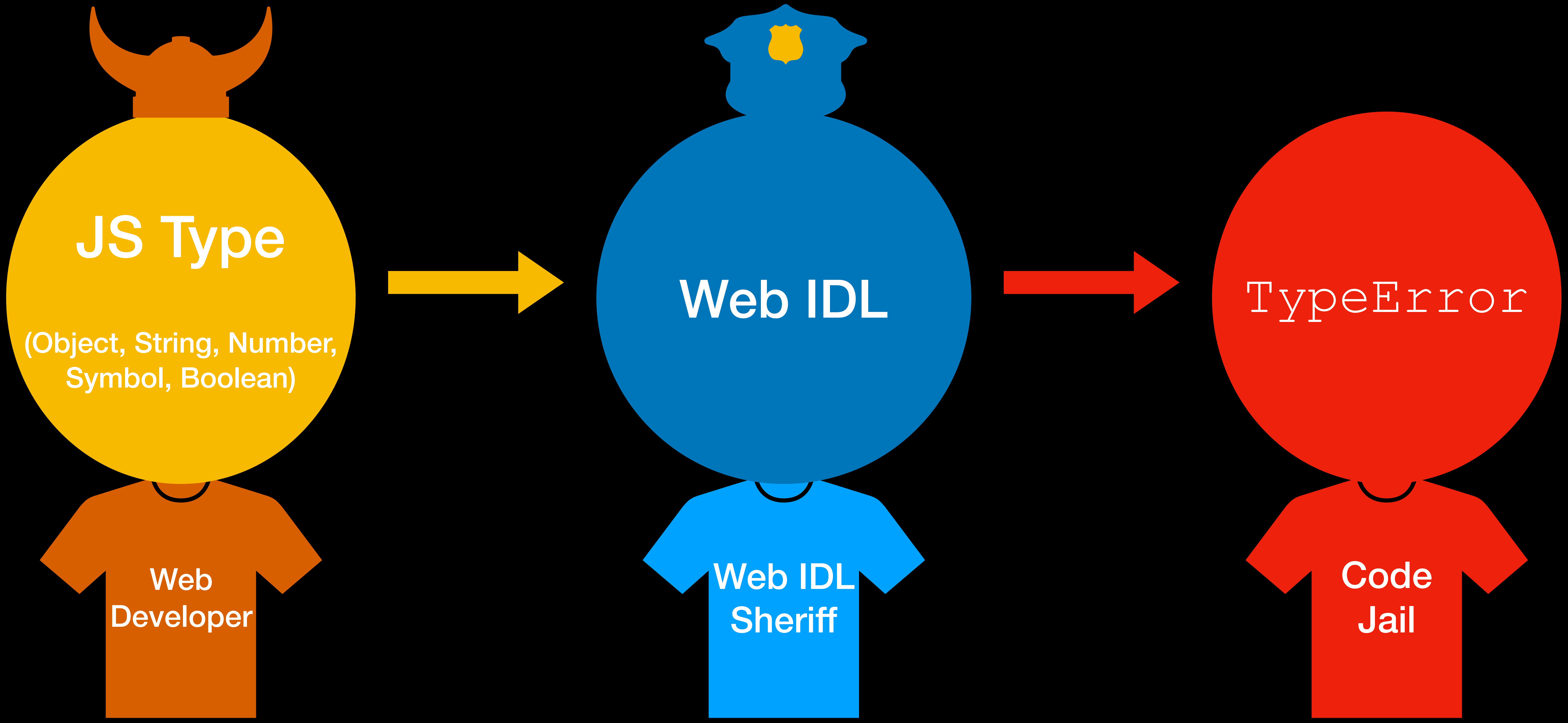
```
void count(optional DOMString label = "default")
```

Web IDL Conversion



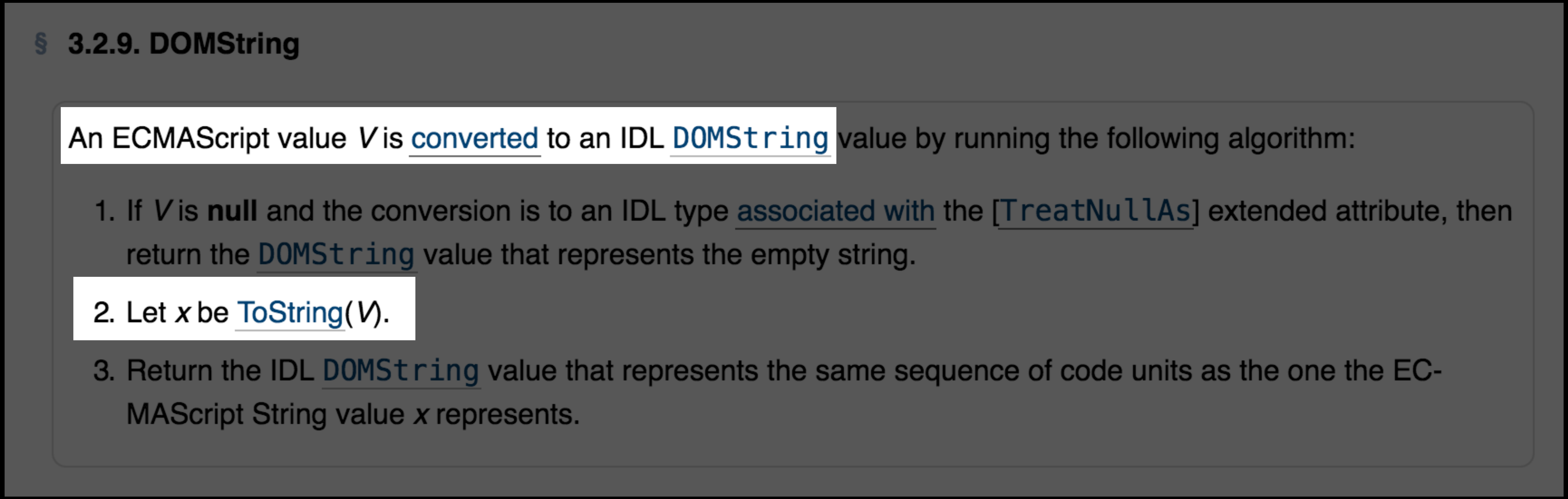
```
void count(optional DOMString label = "default")
```

Web IDL Conversion



```
void count(optional DOMString label = "default")
```

Web IDL



§ 3.2.9. DOMString

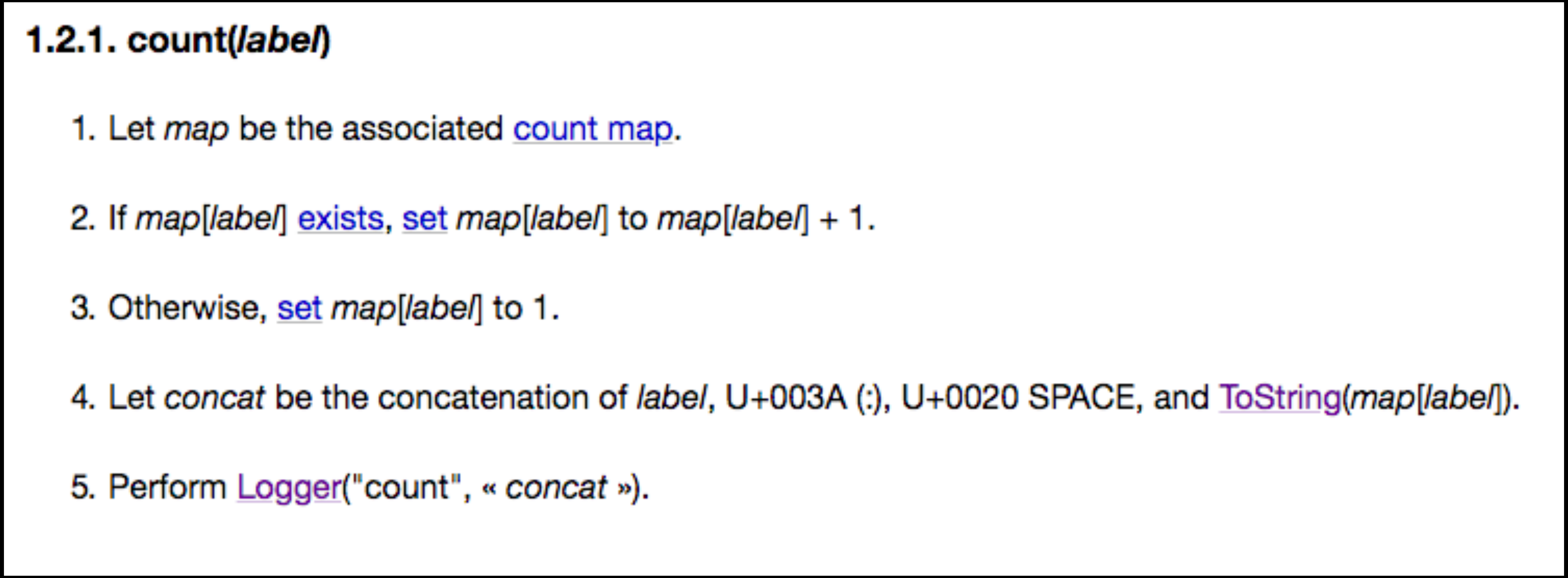
An ECMAScript value *V* is converted to an IDL `DOMString` value by running the following algorithm:

1. If *V* is **null** and the conversion is to an IDL type associated with the [`TreatNullAs`] extended attribute, then return the `DOMString` value that represents the empty string.
2. Let *x* be ToString(*V*).
3. Return the IDL `DOMString` value that represents the same sequence of code units as the one the EC-MAScript String value *x* represents.

https://heycam.github.io/webidl/#es-DOMString

@domfarolino

dom@chromium.org



1.2.1. count(*label*)

1. Let *map* be the associated count map.
2. If *map*[*label*] exists, set *map*[*label*] to *map*[*label*] + 1.
3. Otherwise, set *map*[*label*] to 1.
4. Let *concat* be the concatenation of *label*, U+003A (:), U+0020 SPACE, and ToString(*map*[*label*]).
5. Perform Logger("count", « *concat* »).

Why use Web IDL?

Why use Web IDL?

- Abstraction over ECMAScript text

Why use Web IDL?

- Abstraction over ECMAScript text
- Takes care of things for us:
  - Property init (Prototype chain, property descriptors, …)
  - Type conversion
  - Where to expose interfaces

Why use Web IDL?

- Abstraction over ECMAScript text
- Takes care of things for us:
  - Property init (Prototype chain, property descriptors, …)
  - Type conversion
  - Where to expose interfaces
- Don’t have to use it

How I got involved?

@domfarolino

dom@chromium.org

Overview Repositories 179 Stars 261 Followers 3.6k Following 67

Domenic Denicola
domenic

Unfollow

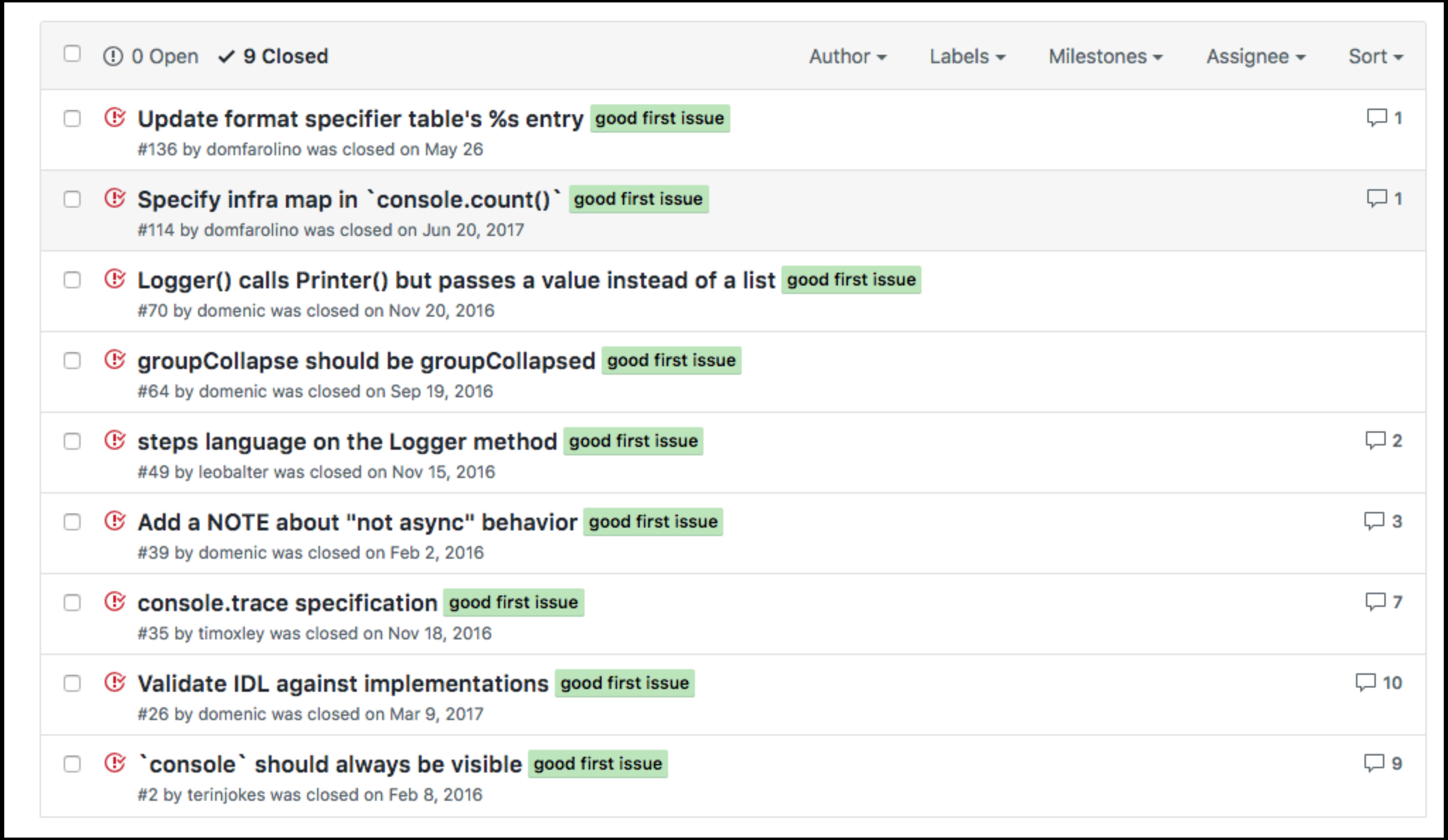
Block or report user

Google
New York, NY

https://domenic.me/

Pinned repositories

whatwg/html HTML Standard ● HTML ★ 2.2k 755	**whatwg/streams** Streams Standard ● HTML ★ 839 94
jsdom/jsdom A JavaScript implementation of the WHATWG DOM and HTML standards, for use with node.js ● JavaScript ★ 11k 1k	**jsdom/whatwg-url** An implementation of the WHATWG URL Standard in JavaScript ● JavaScript ★ 124 42
tc39/proposal-async-iteration Asynchronous iteration for JavaScript ● HTML ★ 672 32	**worm-scraper** Scrapes the web serial Worm into an eBook format ● JavaScript ★ 60 12


0 Open ✓ 9 Closed Author Labels Milestones Assignee Sort

Update format specifier table's %s entry good first issue 1
#136 by domfarolino was closed on May 26

Specify infra map in `console.count()` good first issue 1
#114 by domfarolino was closed on Jun 20, 2017

Logger() calls Printer() but passes a value instead of a list good first issue
#70 by domenic was closed on Nov 20, 2016

groupCollapse should be groupCollapsed good first issue
#64 by domenic was closed on Sep 19, 2016

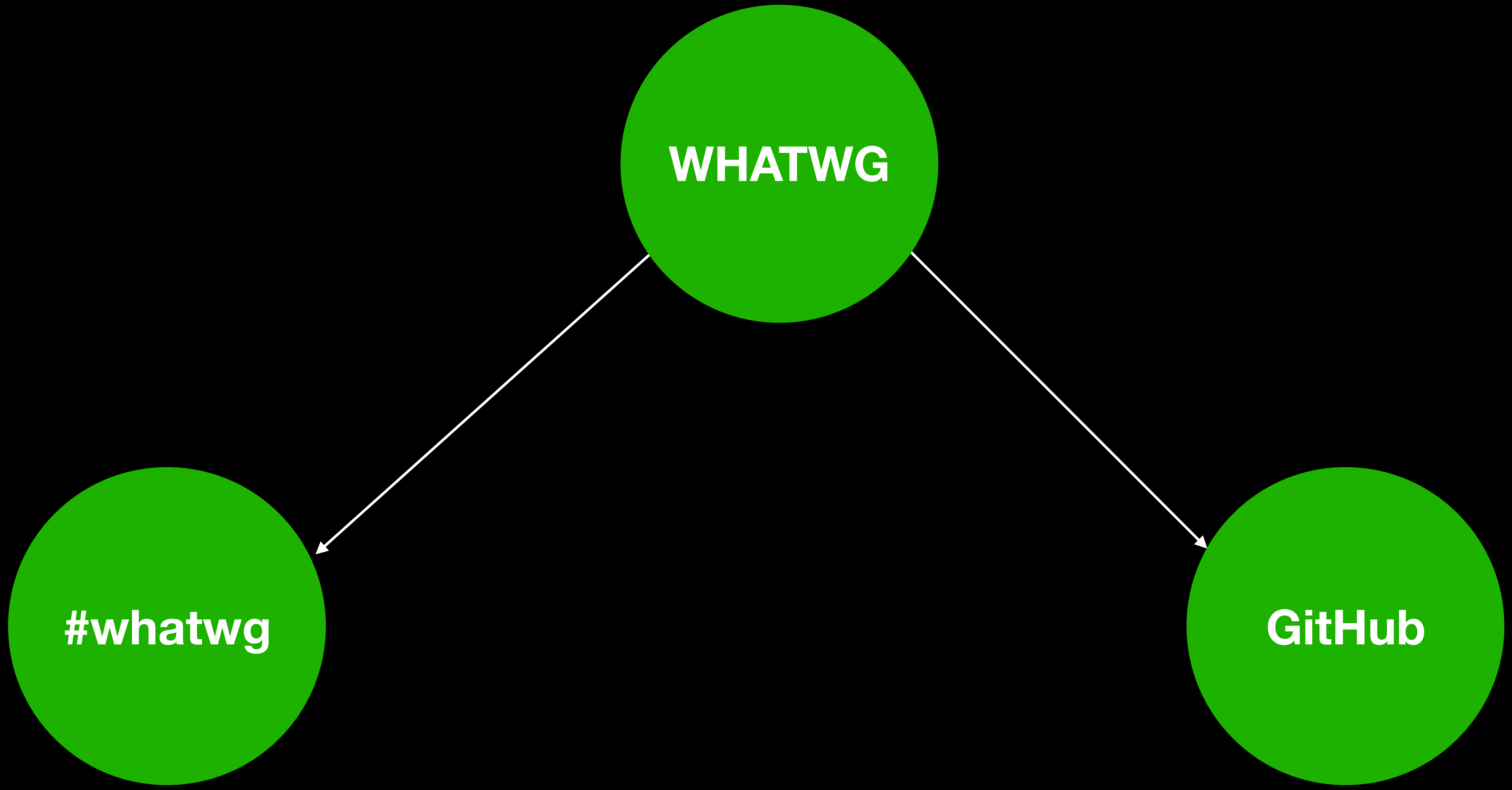
steps language on the Logger method good first issue 2
#49 by leobalter was closed on Nov 15, 2016

Add a NOTE about "not async" behavior good first issue 3
#39 by domenic was closed on Feb 2, 2016

console.trace specification good first issue 7
#35 by timoxley was closed on Nov 18, 2016

Validate IDL against implementations good first issue 10
#26 by domenic was closed on Mar 9, 2017

`console` should always be visible good first issue 9
#2 by terinjokes was closed on Feb 8, 2016


$$\begin{tikzcd}
 & \text{WHATWG} \arrow[dl] \arrow[dr] & \\
\text{\#whatwg} & & \text{GitHub}
\end{tikzcd}$$

Why?

domfarolino/cascadiajs

Things I’ve done

Things I’ve done

- Changed how fetch() works

Things I’ve done

- Changed how fetch() works
- Changed how module scripts are fetched

Things I’ve done

- Changed how fetch() works
- Changed how module scripts are fetched
- `<script referrerpolicy=“”>`

Things I’ve done

- Changed how fetch() works
- Changed how module scripts are fetched
- `<script referrerpolicy=“”>`
- Implemented console APIs

Things I’ve done

- Changed how fetch() works
- Changed how module scripts are fetched
- `<script referrerpolicy=“”>`
- Implemented console APIs
- Priority Hints