Javascript

Crash Course - TSE

Why JavaScript ?

Yet another programing language

Standard things

```
> console.log("Hello world"); // Comment

> var i = 10; /* Another Comment */

> if (i == 10) { console.log("coucou"); }

> if (i == 100) { console.log("hello"); }

> while (i > 0) { console.log("Hello"); i--; }

> function hello(i) { console.log("->", i); }

> hello(5)

> tab = [1, 2, 3]

> console.log(" ouch " + tab.length)
```

Strange Things

```
> console.log("5")

> i = 10

> i = function (a) { console.log("hello\nbye\n") }

> i(10)

> i()

> 1 == true // true

> 1 === true // false

> { size: 10, name: "Stéphane", age: "25"} // This is an object

> a.size
```

What is Javascript ?

Really !!!

It is...

A 'non pure' functional language, but still functional

--> Everything can be seen as a function

An object oriented language without class keyword

--> But with prototype inheritance

A language for 'turn based' programming systems

--> As opposed to 'best-effort' systems

What is functional programming ?

Functions is standard type (nombre, chain, boolean...)

They can be passed as parameter to a function

AND

They can be returned by the end of an execution

Let's write a function that returns a function that can add a number with a fixed value.

Let's write a function that returns a function that can multiply a number with a fixed value.

Lets's compose these two functions

Hey, this is a lambda expression !

To see more...

https://www.youtube.com/watch?v=FITJMJjASUs

Functional manipulation of collection

x = Array.reduce(f (acc, cur), init)

Array = Array.map (f (cur))

Examples : [1, 2, 4, 6, 9],

Find the highest value of an Array

Returns an array that doubles each element value

See underscore.js / lodash.js

Function declaration

```
function square(number) { return number * number; }
```

Function expression

```
square = function (number) { return number * number; }
```

Scope et closure

```javascript
create = function () {
  var name = "hello";
  display = function () {
    console.log("-->", name);
    console.log("-->",
        global.process.title);
  }
  return display();
}
name = "bye";
create();
console.log("->", name)
```

```javascript
create = function () {
  var name = "hello";
  display = function () {
    console.log("-->", name);
    console.log("-->",
        global.process.title);
  }
  return display;
}
name = "bye";
create()();
console.log("->", name)
```

It is...

A 'non pure' functional language, but still functional

--> Everything can be seen as a function

An object oriented language without class keyword

--> But with prototype inheritance

A language for 'turn based' programming systems

--> As opposed to 'best-effort' systems

```
var thing = { 'name' : "stephane", 'size':10}
for (o in chose) {
  console.log( o + '->' + chose[o])
}

chose["size"];

-> 10

chose.name;
-> coucou

delete chose.hello;
chose
```

```
{
  "name": "Stéphane",
  "firstName": "Frénot",
  "children": [
    { "name": "Albert" },
    { "name": "Jeanne" },
    { "name": "Leon" }
  ],
  "age": 46.3,
  "isMan": true
}
```

```
person = {
  "name": "Stéphane",
  "firstName": "Frénot",
  "children": [
    { "name": "Albert" },
    { "name": "Jeanne" },
    { "name": "Leon" }
  ],
  "age": 46.3,
  "isMan": true,
  "sayHello": function () {
    return "Bonjour le "+ new Date();
  }
}

console.log(person.sayHello())
console.log(person.name)
```

```
person = {
  "name": "Stéphane",
  "firstName": "Frénot",
  "children": [
    { "name": "Albert" },
    { "name": "Jeanne" },
    { "name": "Leon" }
  ],
  "age": 46.3,
  "isMan": true,
  "sayHello": function () {
    reponse = "Bonjour le "+ new Date() + "\n";
    reponse += "J'ai " + this.nbChildren() + " enfants";
    return reponse
  },
  "nbChildren": function() {
    return this.children.length;
  }
}

console.log(person.sayHello())
console.log(person.name)
```

```
function Person (name, firstname, children) {
  this.name = name;
  this.firstname = firstname;
  this.children = children;

  this.sayHello = function () {
    reponse = "Bonjour le "+ new Date() + "\n";
    reponse += "J'ai " + this.nbChildren() + " enfants";
    reponse += elemSep;
    return reponse;
  }

  this.nbChildren = function() {
    return this.children.length;
  }

  var elemSep = " privé";

}

Person.prototype.toString = function() {
  return 'hello' + this.name;
}

var someOne = new Person("Frénot", "Stéphane", [{"name":"Albert"}, {"name":"Leon"}]);
console.log("" + someOne);
console.log(Object.getOwnPropertyNames(someOne));
console.log("name " + someOne.name);
console.log("name " + someOne.privateName);
console.log(Person.prototype);
console.log(someOne.sayHello());
console.log(someOne.elemSep);
```

```
Person = function () {
  function Person (name, firstname, children) {
    this.name = name;
    this.firstname = firstname;
    this.children = children;
  }

  Person.prototype.sayHello = function () {
      reponse = "Bonjour le "+ new Date() + "\n";
      reponse += "J'ai " + this.nbChildren() + " enfants";
      reponse += elemSep;
      return reponse;
    }

  Person.prototype.nbChildren = function() {
      return this.children.length;
    }

  Person.prototype.toString = function() {
    return 'hello' + this.name;
  }

  var elemSep = " privé";

  return Person;

}();

var someOne = new Person("Frénot", "Stéphane", [{"name":"Albert"}, {"name":"Leon"}]);
console.log("" + someOne);
console.log(Object.getOwnPropertyNames(someOne));
console.log("name " + someOne.name);
console.log("name " + someOne.privateName);
console.log(Person.prototype);
console.log(someOne.sayHello());
console.log(someOne.elemSep);
```

WAT !

It is...

A 'non pure' functional language, but still functional

--> Everything can be seen as a function

An object oriented language without class keyword

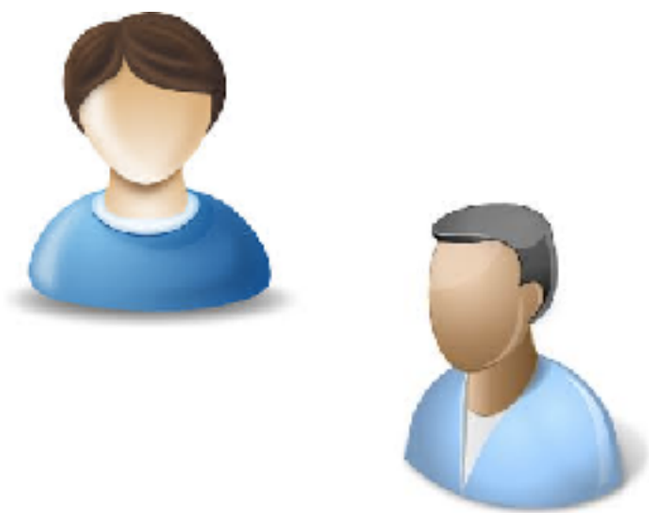
--> But with prototype inheritance

A language for 'turn based' programming systems

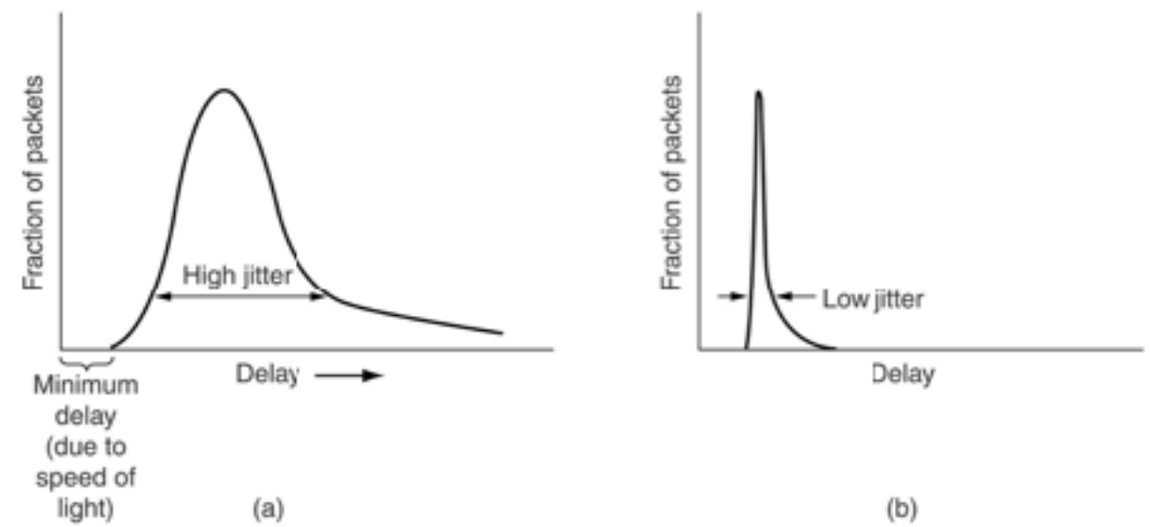
--> As opposed to 'best-effort' systems

What is a web browser ?

Web takes care of two elements



/



End-User

TCP network

Callbacks

"Don't wait for me, I will call you when I am ready !"

```javascript
var request = require('request');

request('http://www.google.com', function (error, response, body) {
  if (!error && response.statusCode == 200) {
    console.log(body)  }
})

console.log("hello");
```

THIS !!

and... bind

```
var Counter = function () {
  this.count = 0;
  this.tick = function () {
    this.count++;
    console.log(this.count);
  }
};

var myCounter = new Counter();
myCounter.tick()
```

We want one tick every second !!!

var timeoutID = window.setTimeout(func[, delay, param1, param2, ...]);

Let's talk about JavaScript

I am a programming language

single-threaded,
non blocking,
asynchronous,
concurrent.

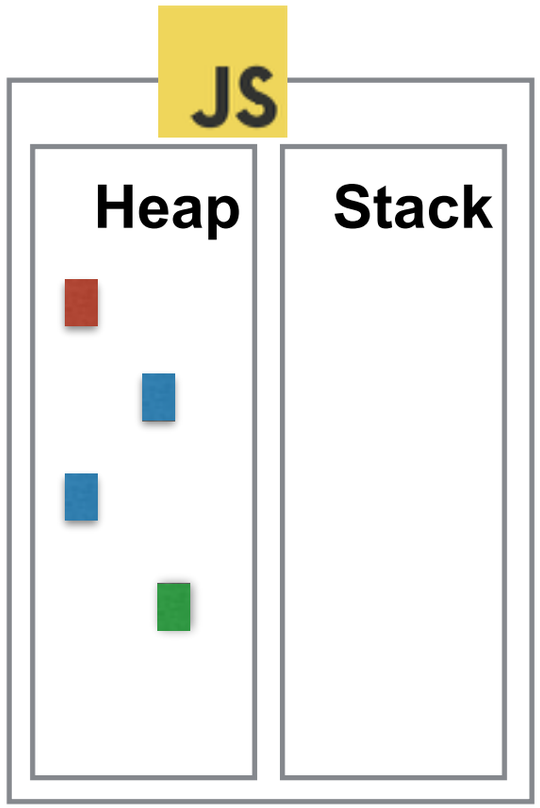
…that has

a call stack,
an event loop,
a callback stack,
some API,
and many other things

But

a complex object-oriented approach,
a strange type check mechanism,
a callbacks hell,
and limited to the browser !

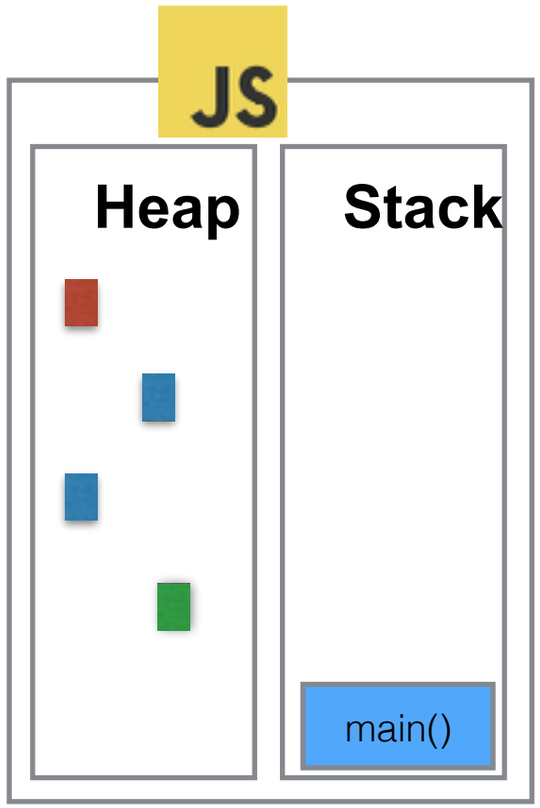
JavaScript




```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

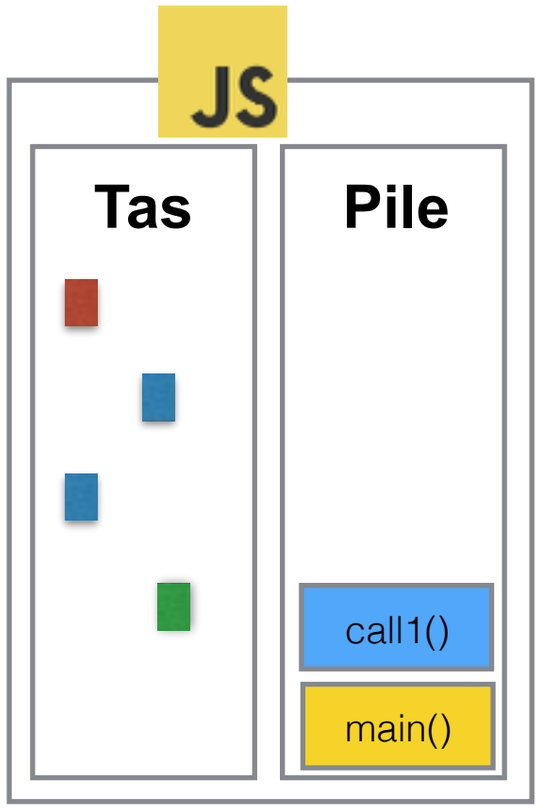



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 0

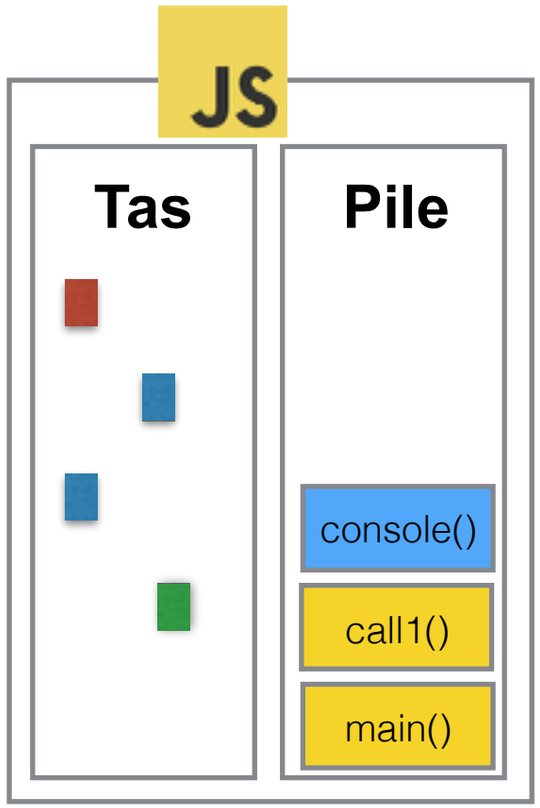



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 0

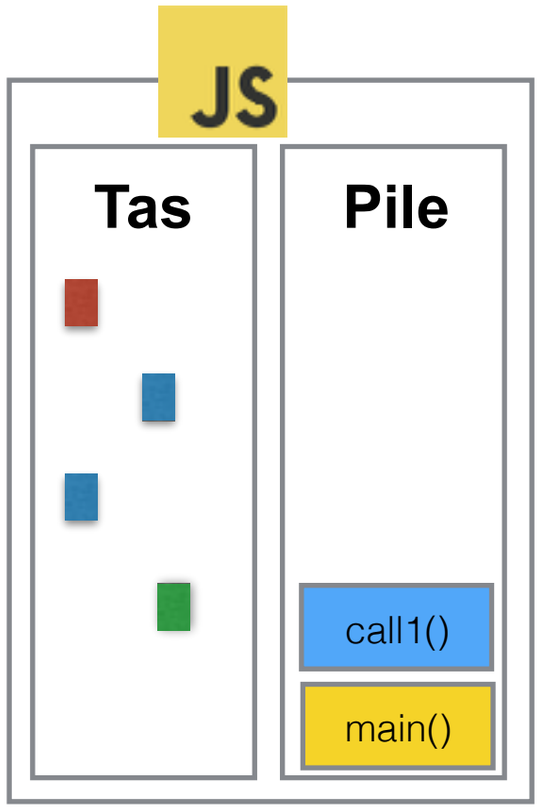



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 () {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 0

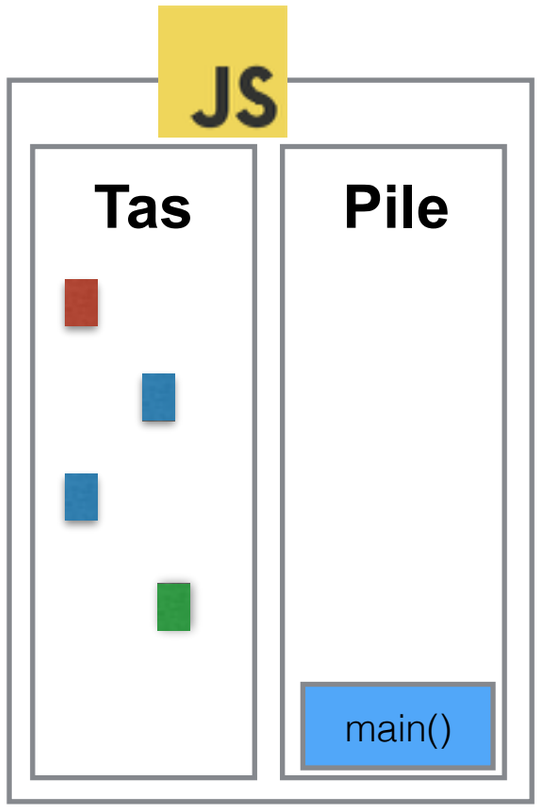



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 0

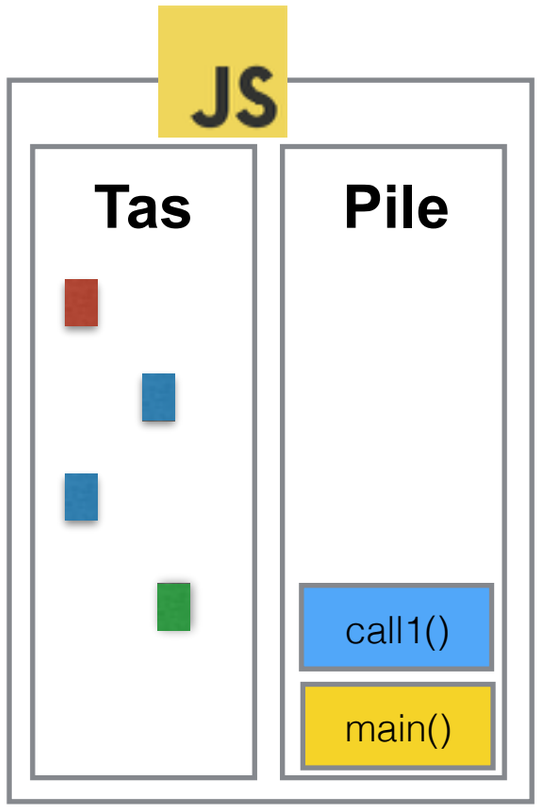



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 1

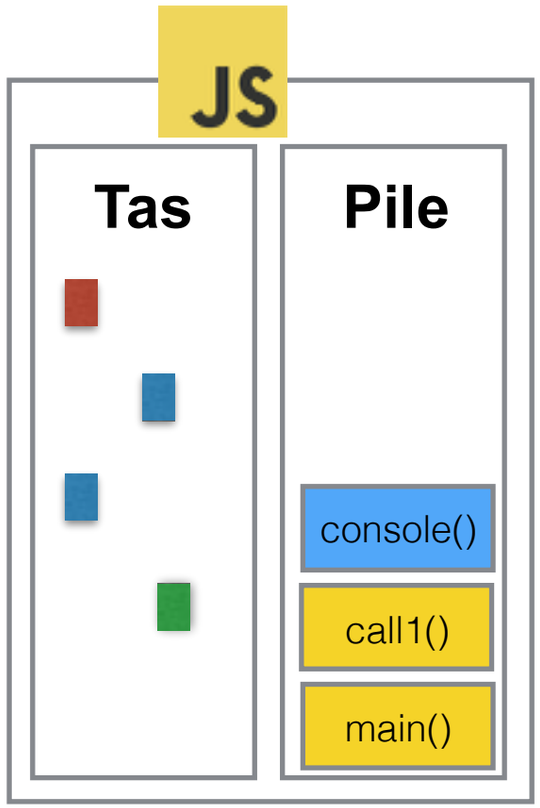



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 1

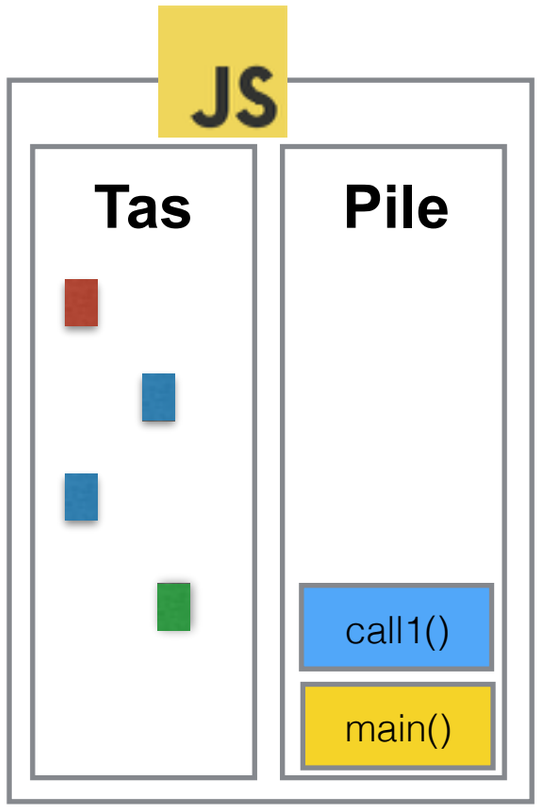



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 1

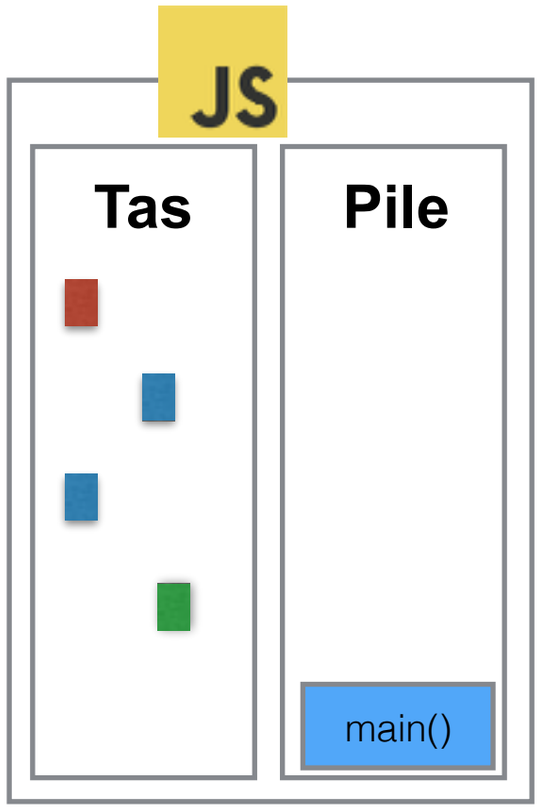



```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

i -> 1



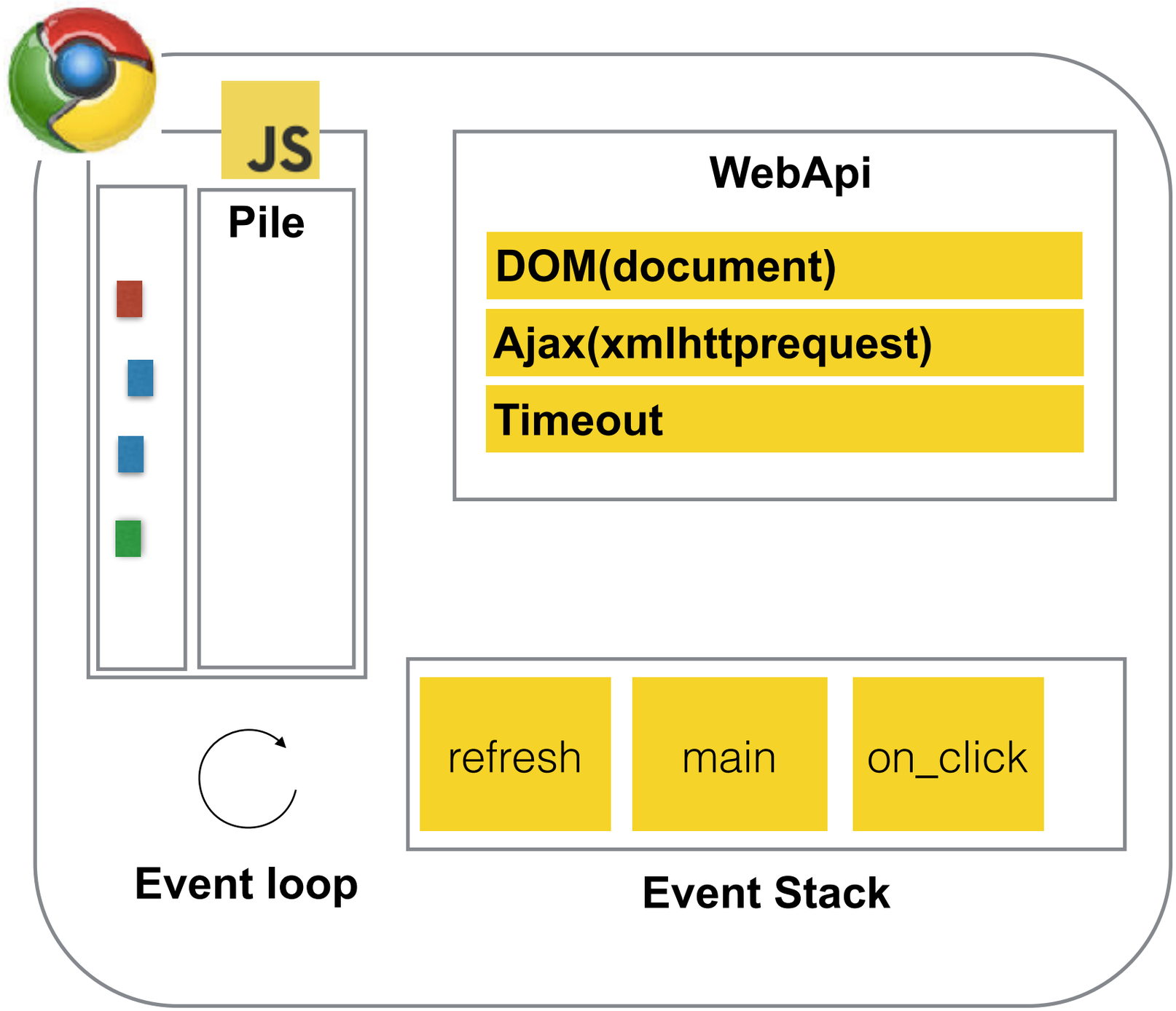

```
</html>
   <body>
    Hello World
    <script src="/test.js"></script>
    GoodBye World
  </body>
</html>
```

```
function call1 (val) {
  console.log("val -->" + val);
  return;
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

test.js

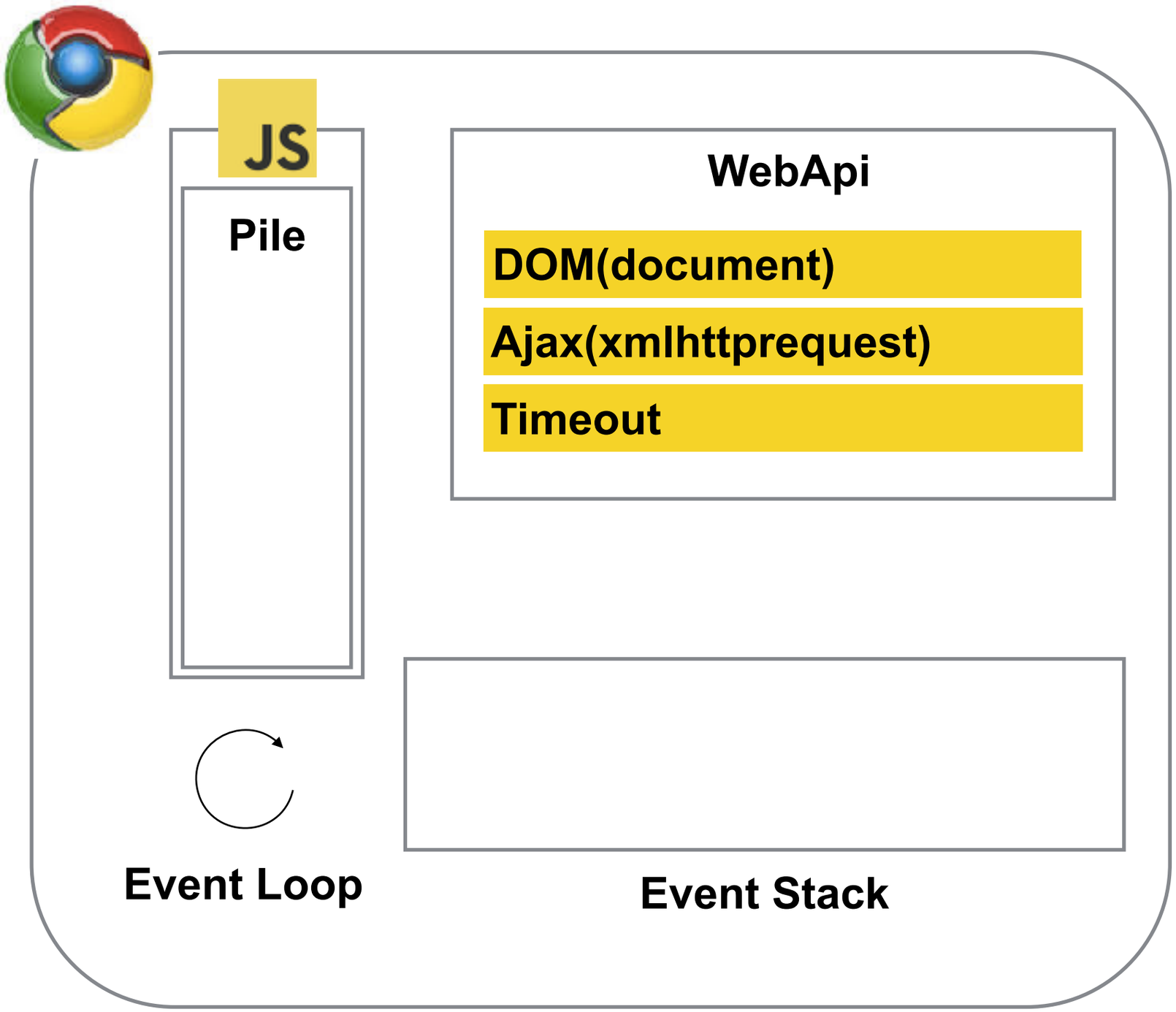
i -> ...

The Web Browser isn't javascript



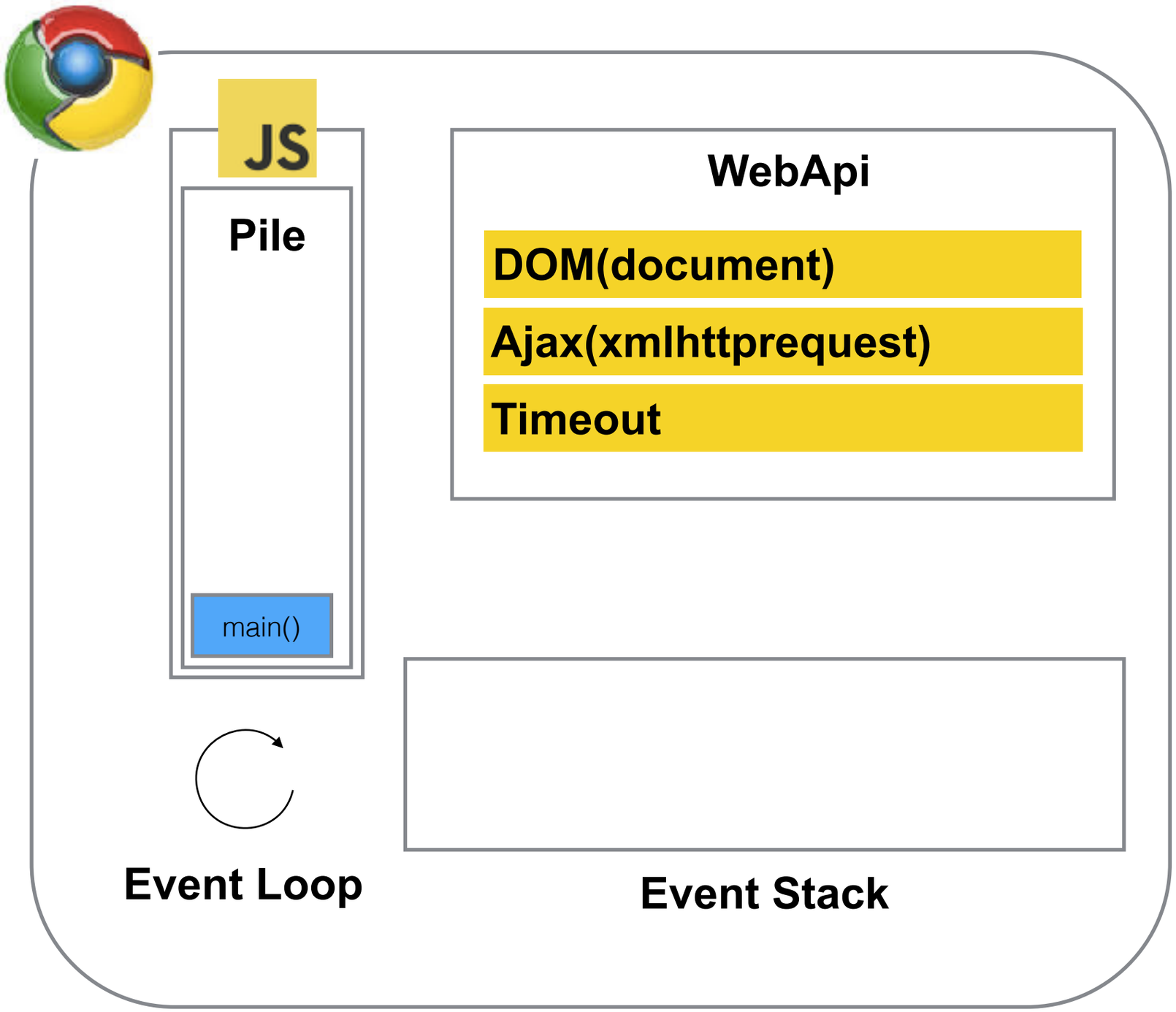
Turn Based System

Never wait
Never block
Finish fast



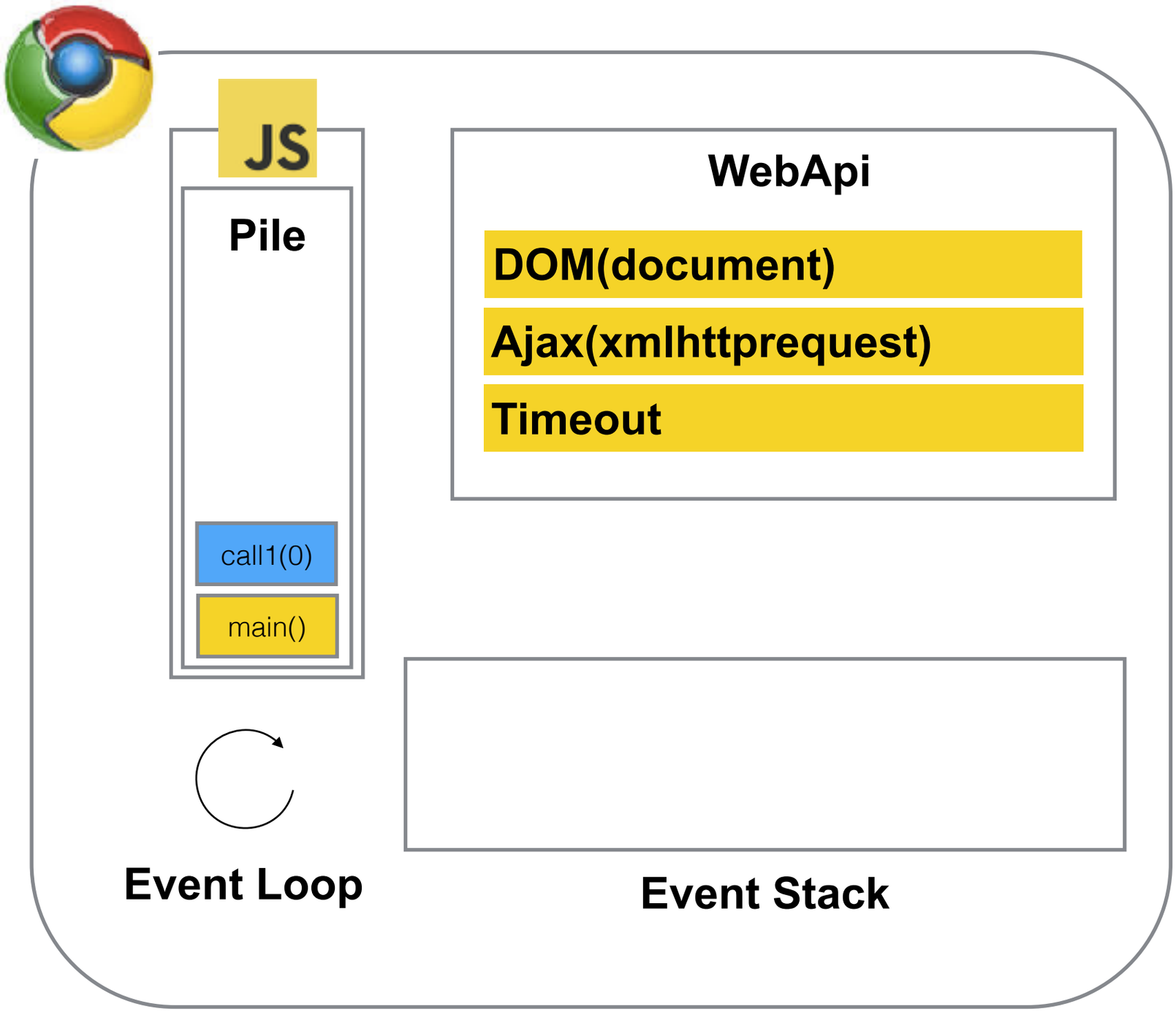

```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```



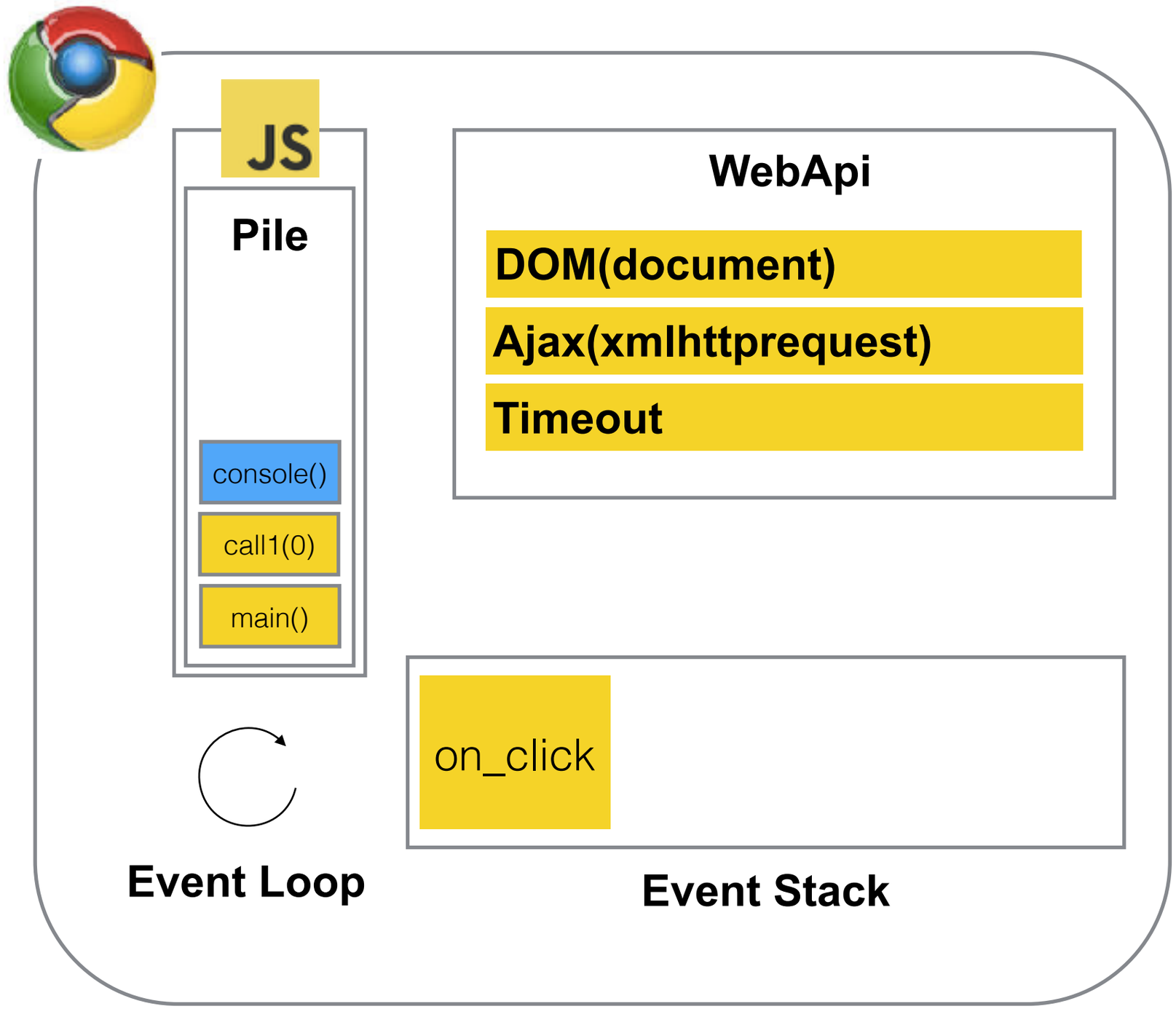

```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```



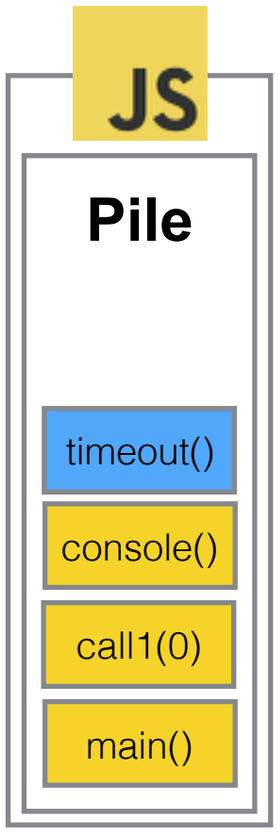

```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

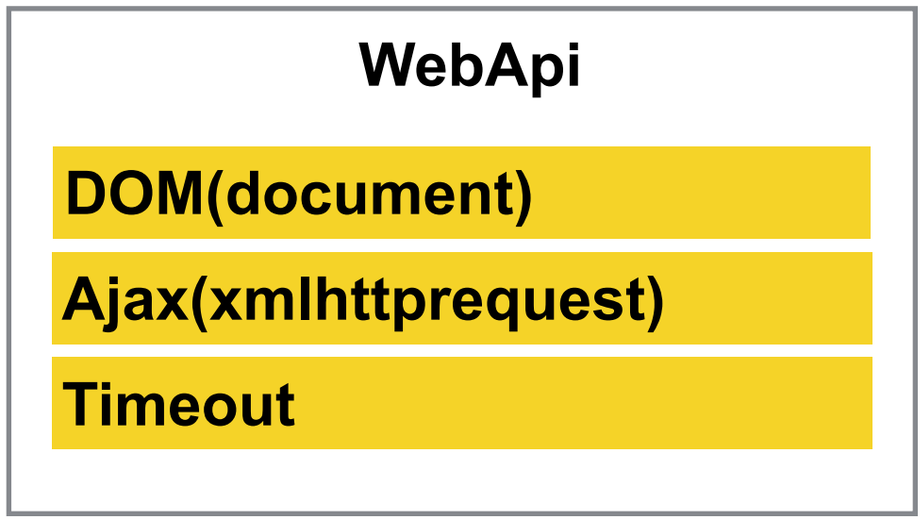
```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```

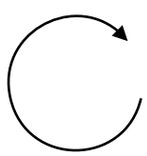



```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```





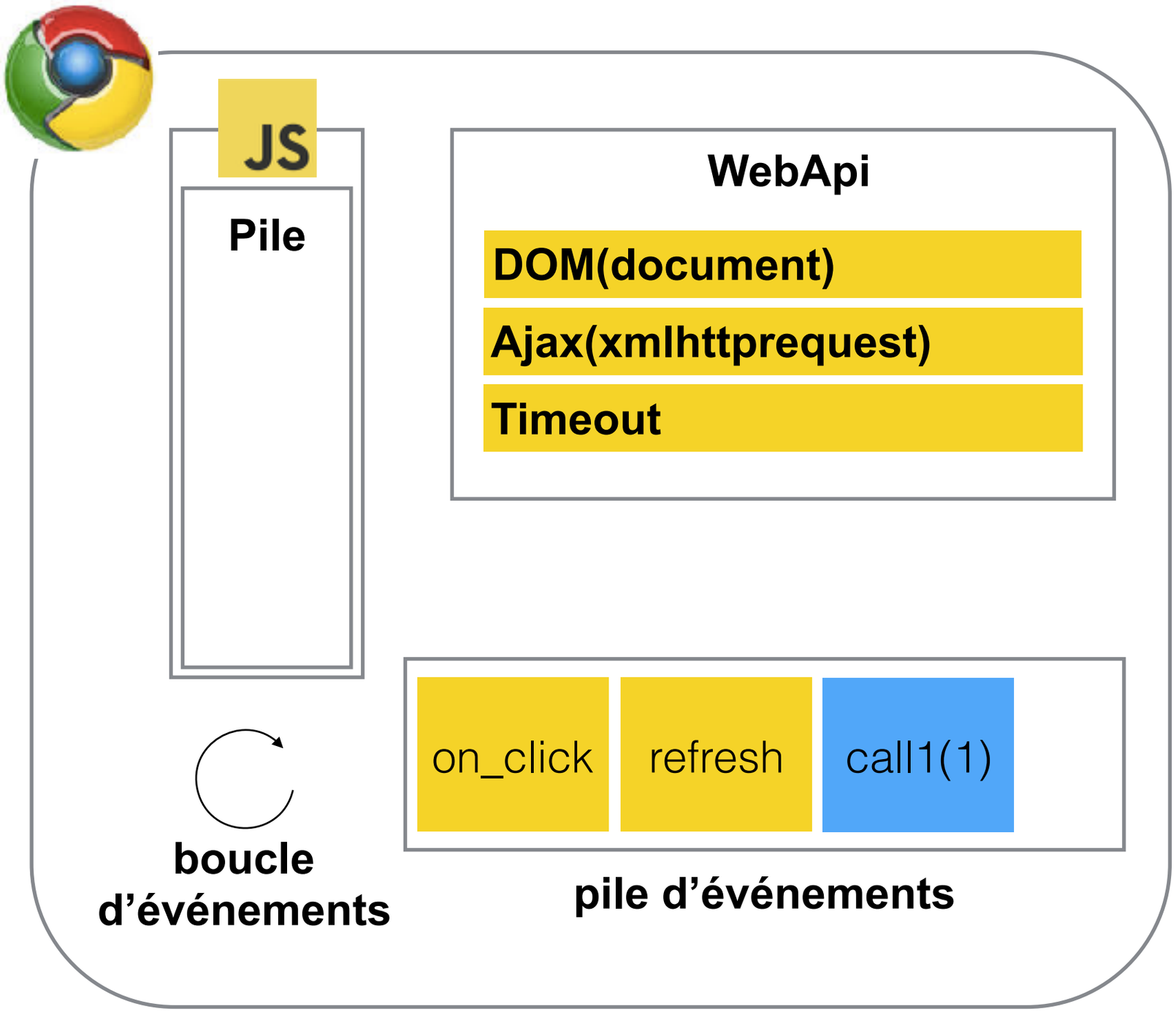


boucle d'événements

pile d'événements

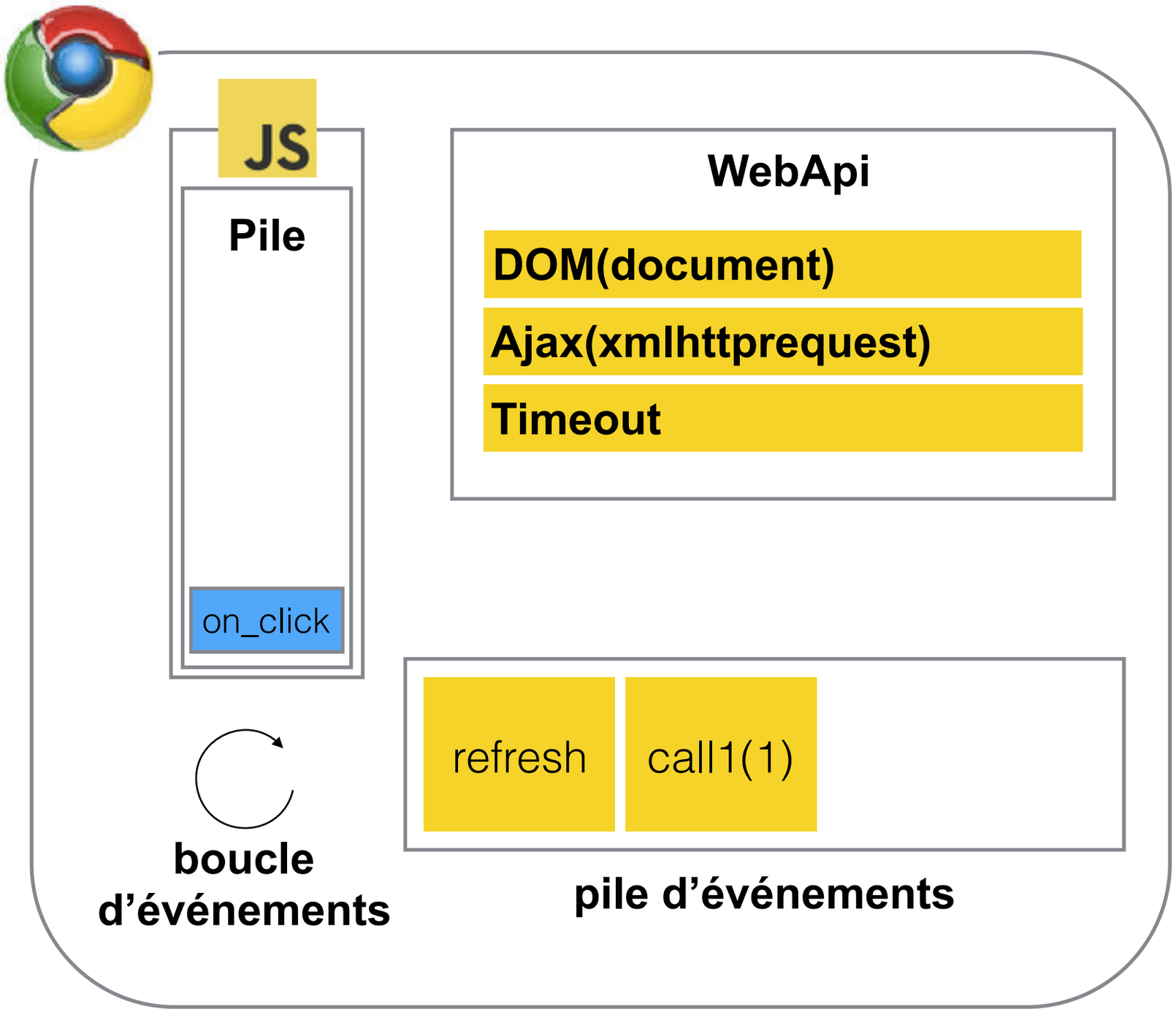
```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```




```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```




```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```

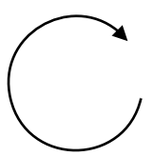
JS

Pile

refresh

WebApi

- **DOM(document)**
- **Ajax(xmlhttprequest)**
- **Timeout**



boucle d'événements

call1(1)

pile d'événements

```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```

JS

Pile

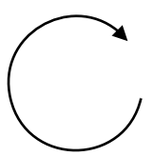
call1(1)

WebApi

DOM(document)

Ajax(xmlhttprequest)

Timeout

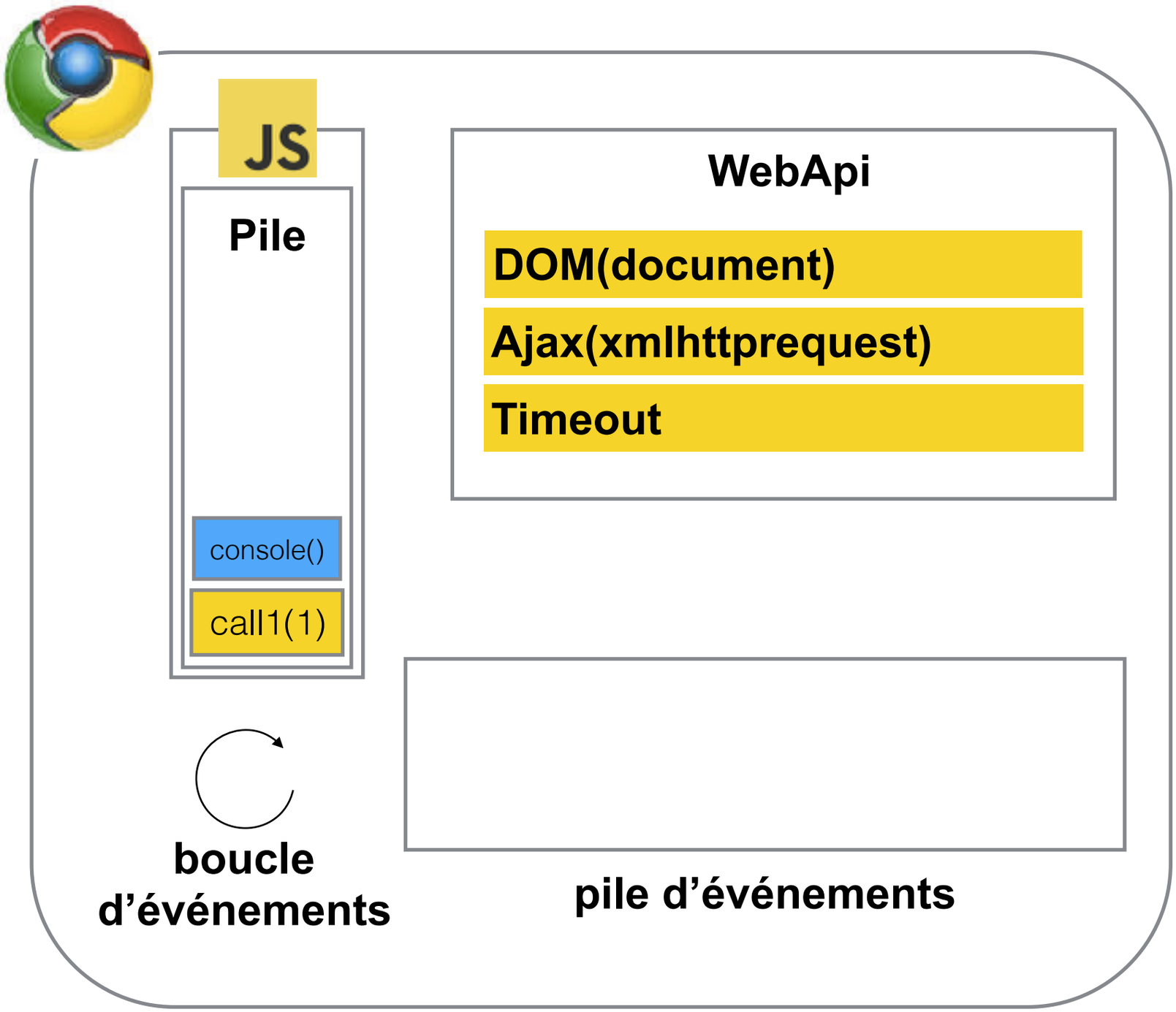


boucle d'événements

pile d'événements

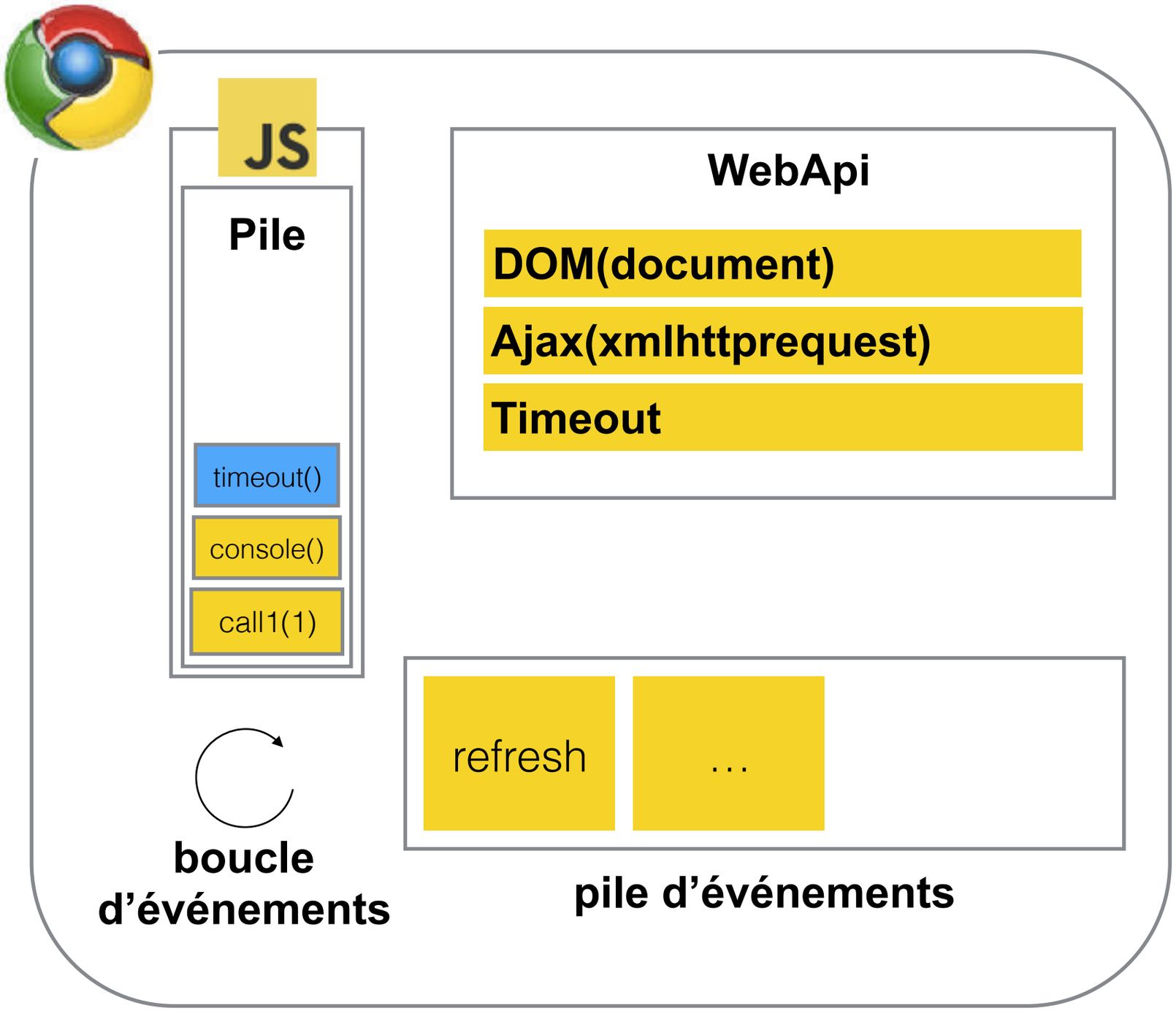
```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```



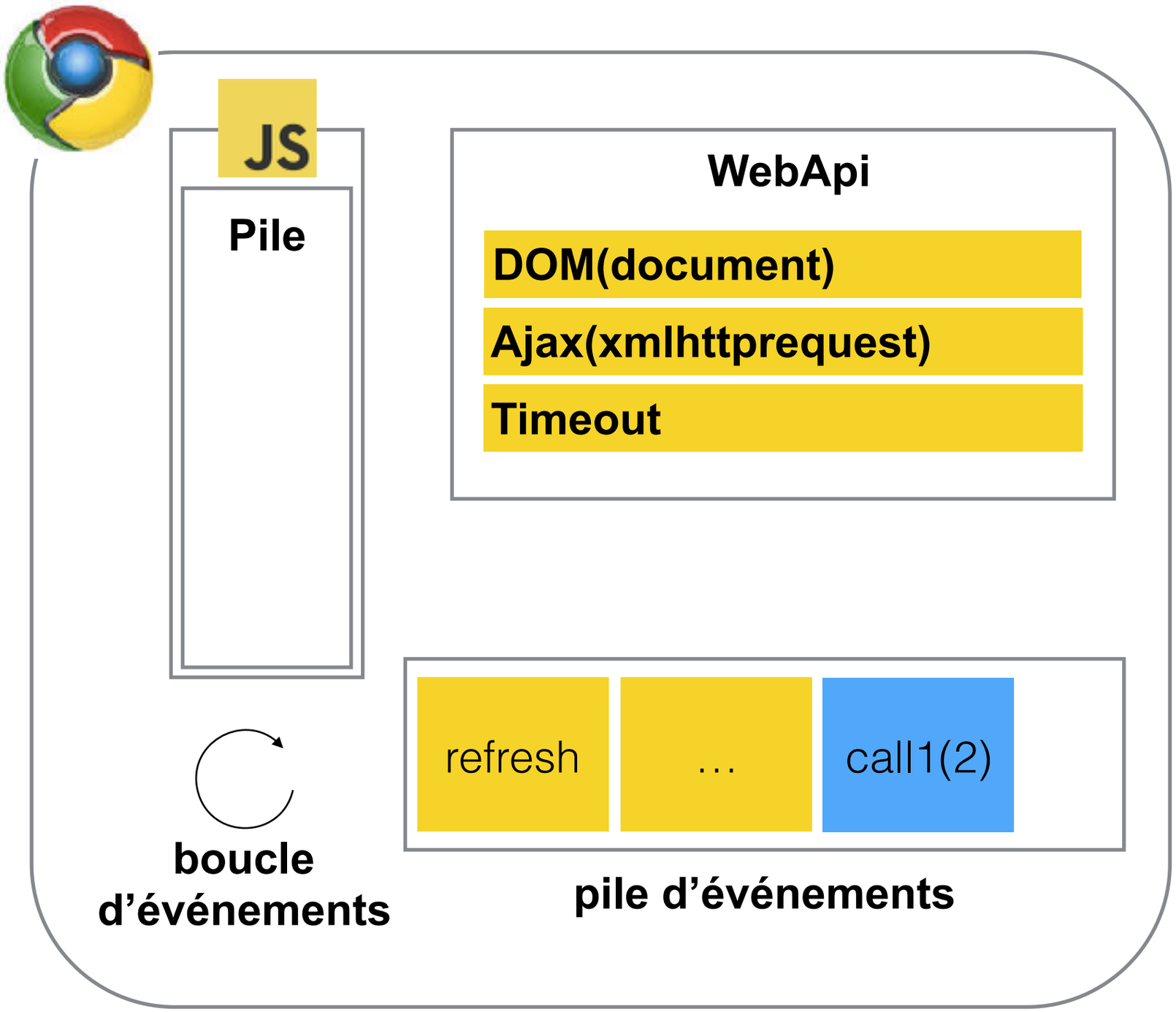

```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```




```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

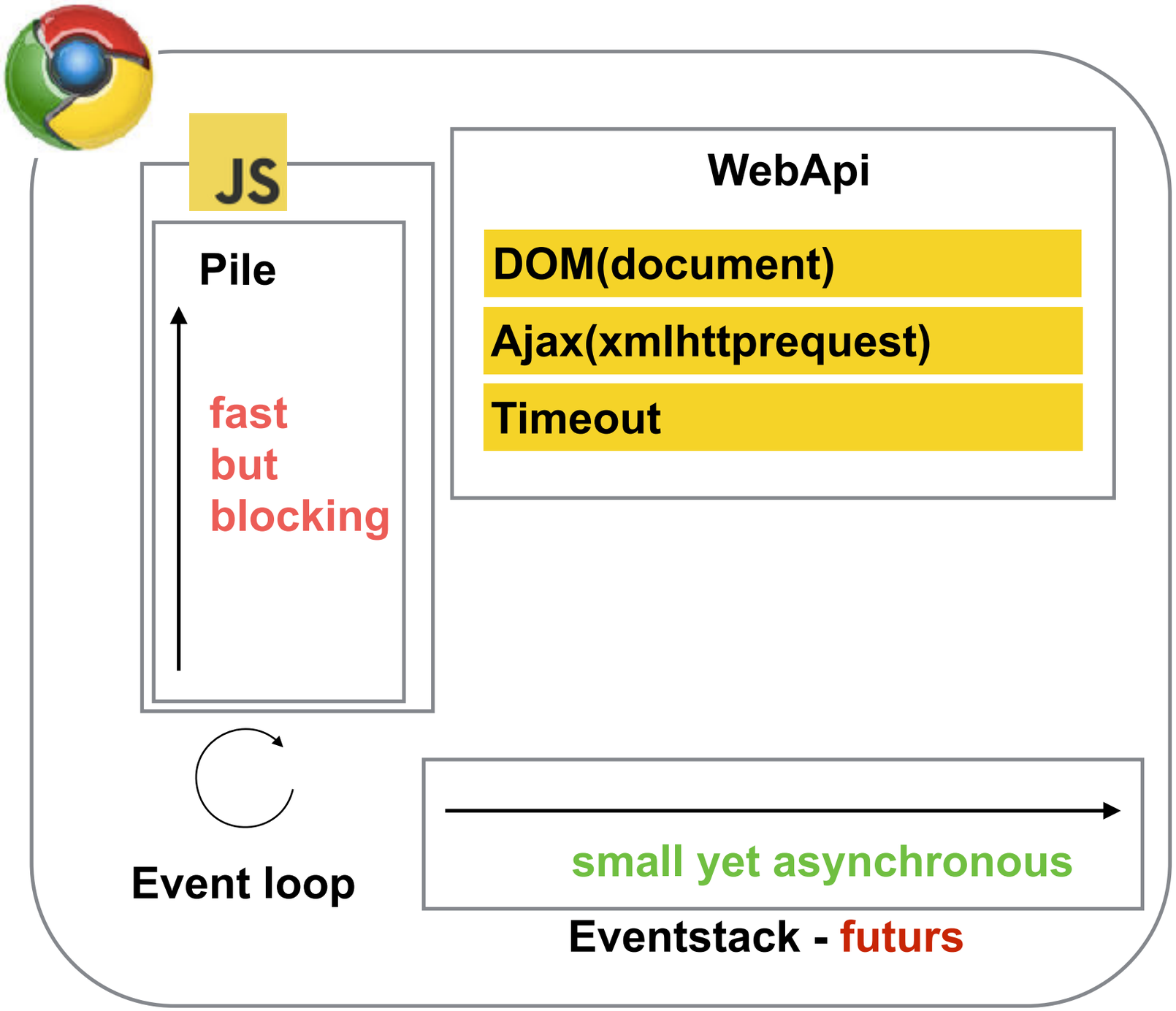
```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```




```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```

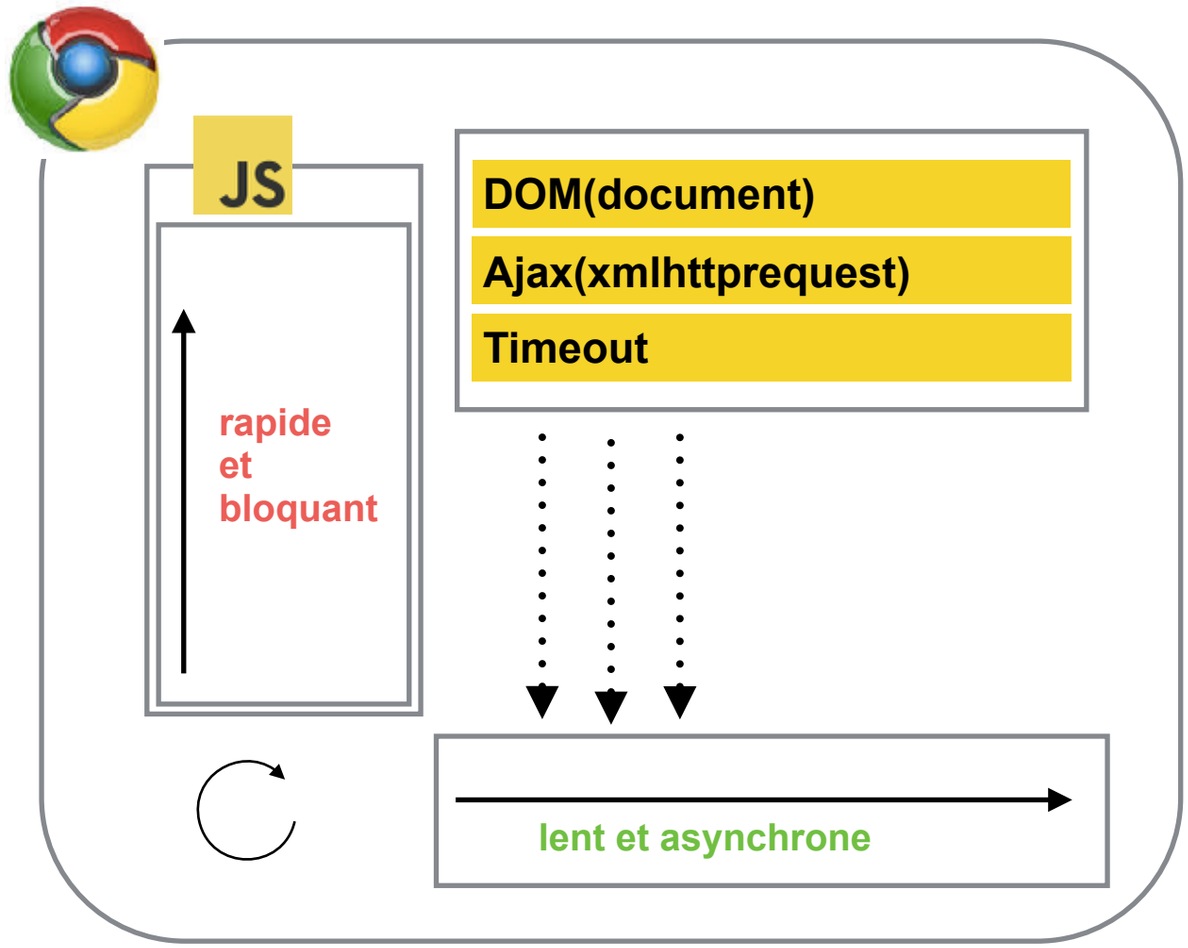
Programming in the future




```
function call1 (val) {
  console.log("val -->" + val);
}

for (i = 0; i < 200000; i++) {
  call1(i);
}
```

```
function call1 (i) {
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```

Let's really talk about JS

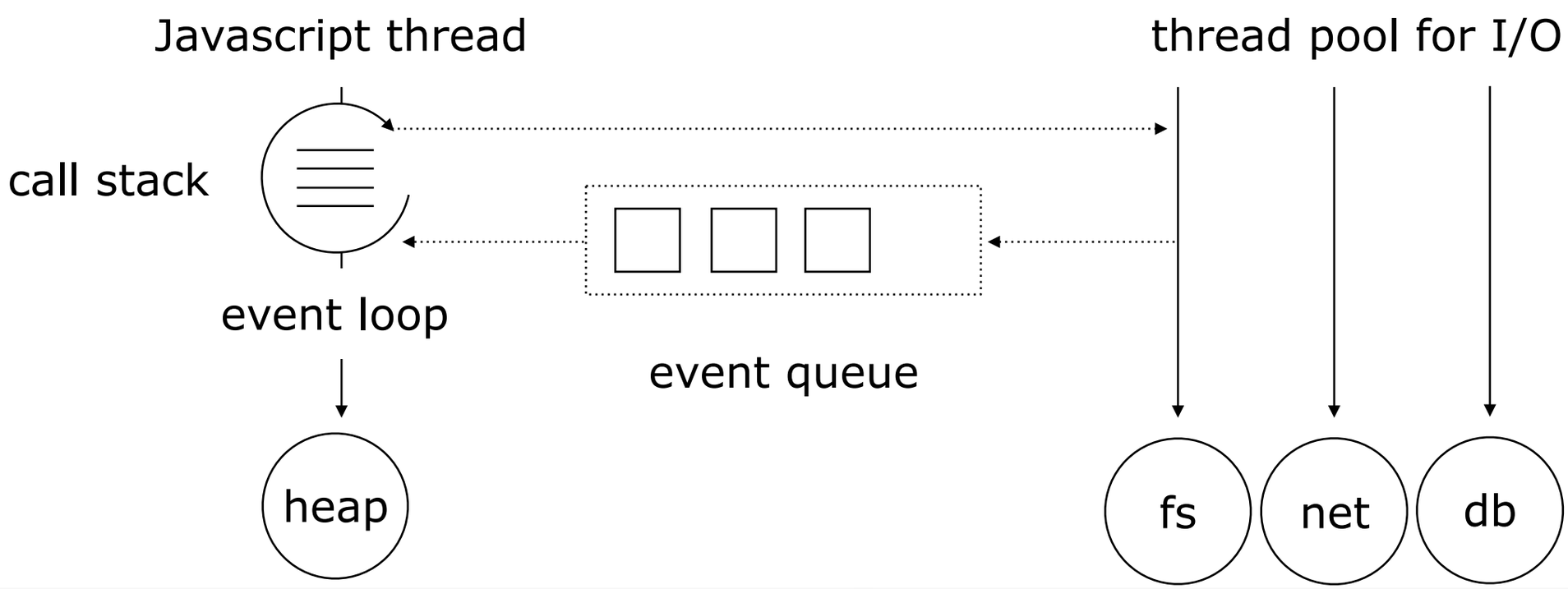



```
function call1 (i)(
  if (i < 200000) {
    i++;
    console.log(i);
    setTimeout(call1, 0, i);
  }
}

call1(0);
```

Assembly language of the Web
Promises development
Functional Approach
No synchronization API

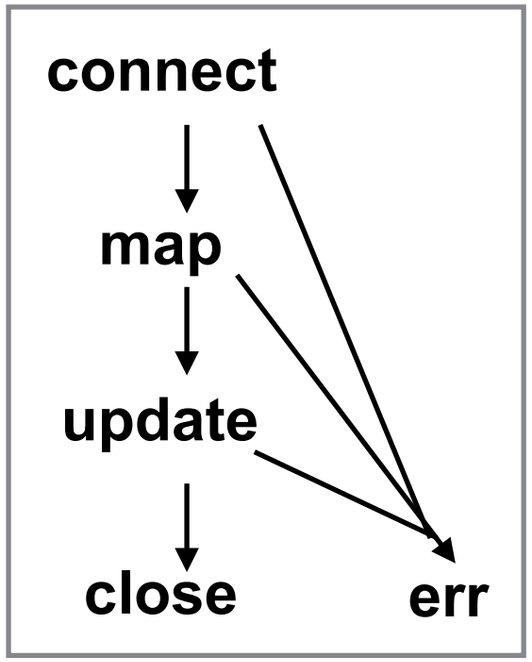
Moreover...

1st efficient functional language/ V8
1er programming language with latency in mind

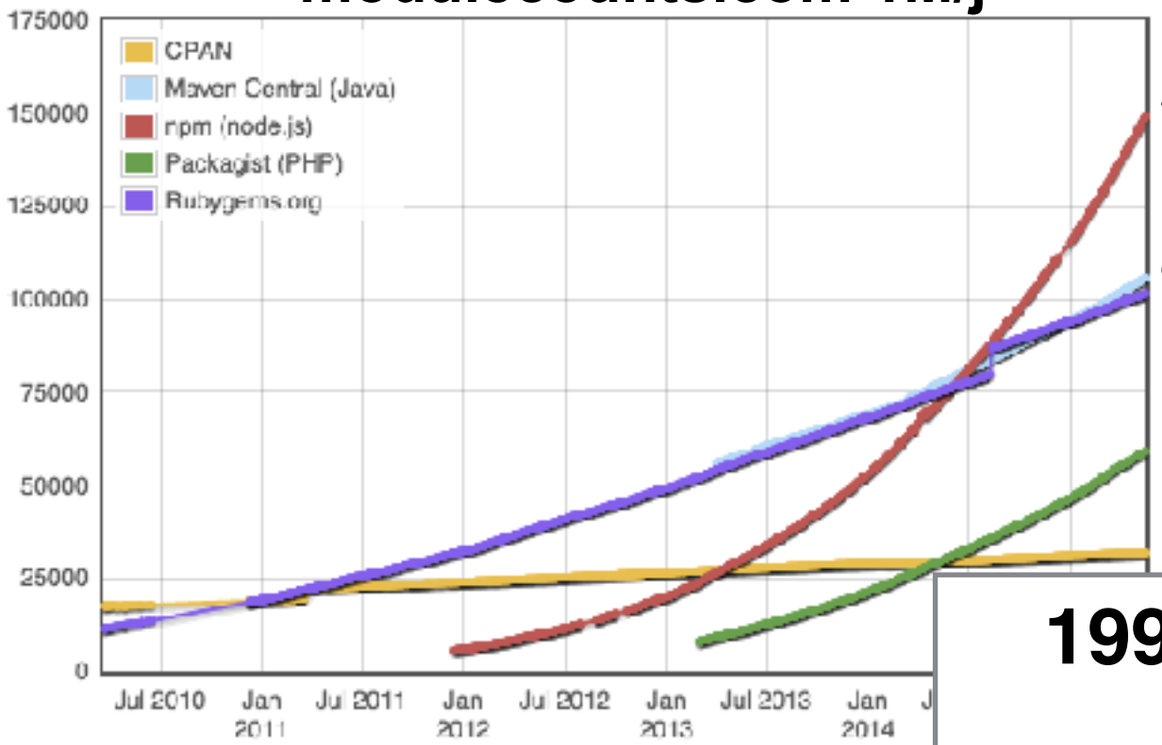


NodeJs
network event programming system, '*turn based*' giving queue priority

```
Promise = require 'bluebird'
mongoose = require 'mongoose'

database = mongoose.createConnection 'mongodb://localhost/jumplyn'
Document = database.model 'Document', require '../core/models/Document'
DocumentFile = Document.discriminator 'DocumentFile', require '../core/models/
DocumentFile'

connect = new Promise (resolve, reject) ->
  database.on 'connected', () ->
    console.log 'Connected to the new database "jumplyn"'
    resolve()

connect
.then ->
  Promise.map oseos, (oseo) ->
    DocumentFile
    .update(
      _id: oseo._id
    ,
      $set:
        extract:
          note: parseInt oseo.note
          abstract: oseo.abstract
          extracts:
            marche: oseo.extracts['marché']
            social: oseo.extracts['social']
    )
    .exec()
.catch (err) ->
  console.log 'ERROR'
  console.log err.stack
.then ->
  console.log 'Closing connections.'
  database.close()
```



modulecounts.com 1M/j



250/j

90/j

1995 javascript

2008 V8

2009 nodejs

jQuery — write less, do more.

Bootstrap

ember

BACKBONE.JS

ANGULARJS by Google

polymer

React

material design

CoffeeScript

TypeScript

Haskell

Dart

Scala

Walmart

ebay

PayPal

Microsoft

NETFLIX

Google

UBER

YAHOO!

LinkedIn

http://www.quora.com/What-are-the-biggest-websites-built-with-Node-js-on-the-server-side

Merci !

Stéphane Frénot, CITI, INSA, Ixxi

google -> @sfrenot

Thanks to the help of Damien Remeirt

Javascript (quelques pointeurs, non exhaustif)

Douglas Crockford https://www.youtube.com/watch?v=dkZFtimgAcM

Eloquent javascript, le livre à lire sur javascript. http://eloquentjavascript.net/

https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

https://www.destroyallsoftware.com/talks/the-birth-and-death-of-javascript

Philip Roberts https://www.youtube.com/watch?v=8aGhZQkoFbQ

Angularjs, reactjs, Polymer, nodejs/iojs

Exemples d'applications javascript

http://setosa.io/bus/

http://ww2.kqed.org/lowdown/2013/11/12/traffic-waves/

http://setosa.io/blog/2014/09/02/gridlock/

Rupture numérique (quelques pointeurs, non exhaustif)

http://www.economie.gouv.fr/files/rapport-fiscalite-du-numerique_2013.pdf

http://www.economie.gouv.fr/files/files/PDF/rapport_TNEF.pdf

http://pezziardi.net/2014/09/25/startup-detat-mefiez-vous-des-contrefacons/

Peter Thiel, Competition is for looser, https://www.youtube.com/watch?v=5_0dVHMpJlo

Philosophie magazine #73, octobre 2013

http://www.cyberstrategie.org/?q=fr/etude-prospective-strategique-balkanisation-du-web-chance-risque-europe

http://www.ecfr.eu/publications/summary/chinas_expanding_cyberspace311

http://www.marketplatforms.com/wp-content/uploads/Downloads/Platform-Economics-Essays-on-Multi-Sided-Businesses.pdf

http://platformed.info/

google js / mdn

https://medium.com/javascript-scene/10-must-see-web-apps-games-36ab9ca60754#.l619vsip3

https://www.destroyallsoftware.com/talks/wat

http://scriptcraftjs.org/