

The background is a vibrant blue abstract composition. It features a central globe showing the Americas, overlaid with a white grid. Radiating from the globe are numerous thin, white lines that resemble a network or data flow. In the upper left, there's a small, complex diagram of interconnected nodes. The overall effect is one of high-tech connectivity and global communication.

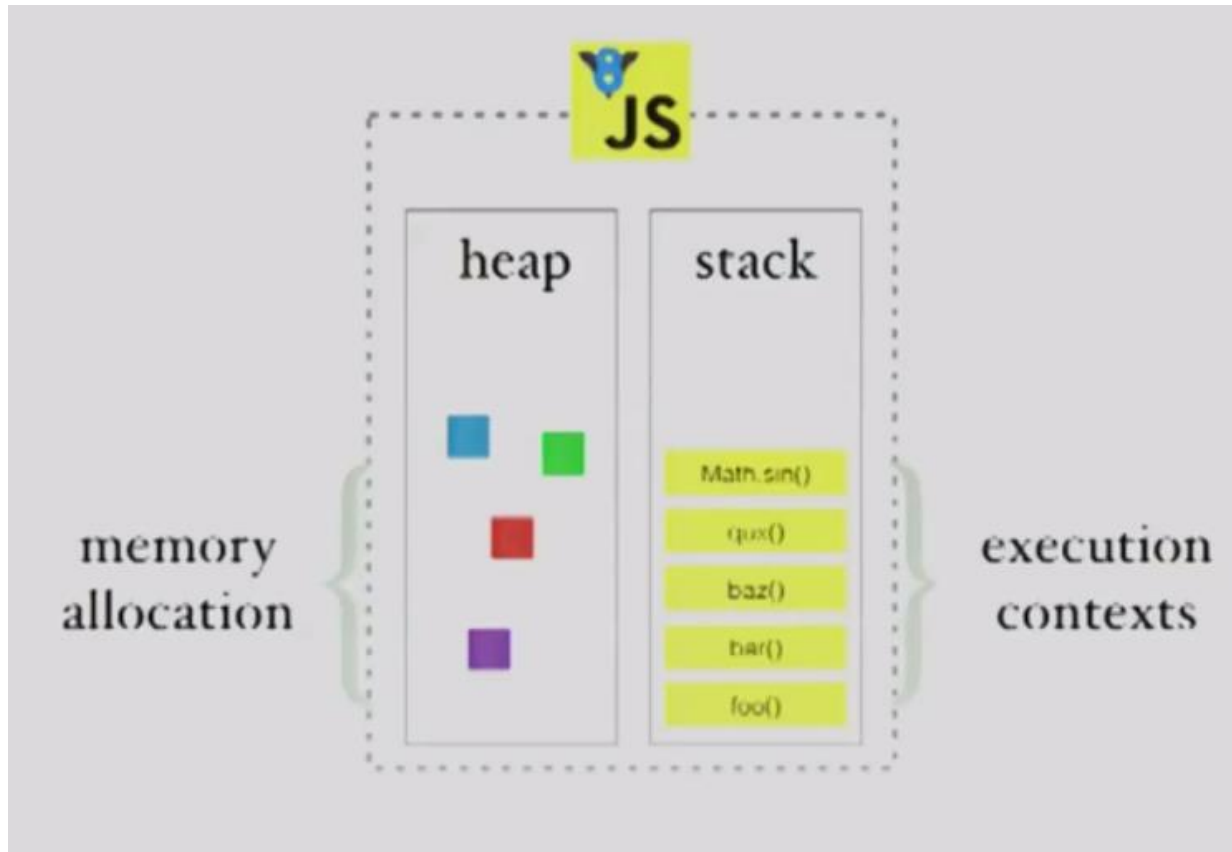
Advanced Topics in Internet Application Development

Class 3 – Javascript Internals

Agenda

- JS Event Loop
- JS as opposed to Browser APIs

JS V8 Engine



- But where's **AJAX**??? **SetTimeout()**??? **DOM**??

The JS Call Stack

One thread == One call stack == One thing at a time

How the Call Stack works?

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```

stack

Let's run this code and see how the stack behaves

Call Stack



```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```

stack

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}
```

```
function square(n) {  
    return multiply(n, n);  
}
```

→

```
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}
```

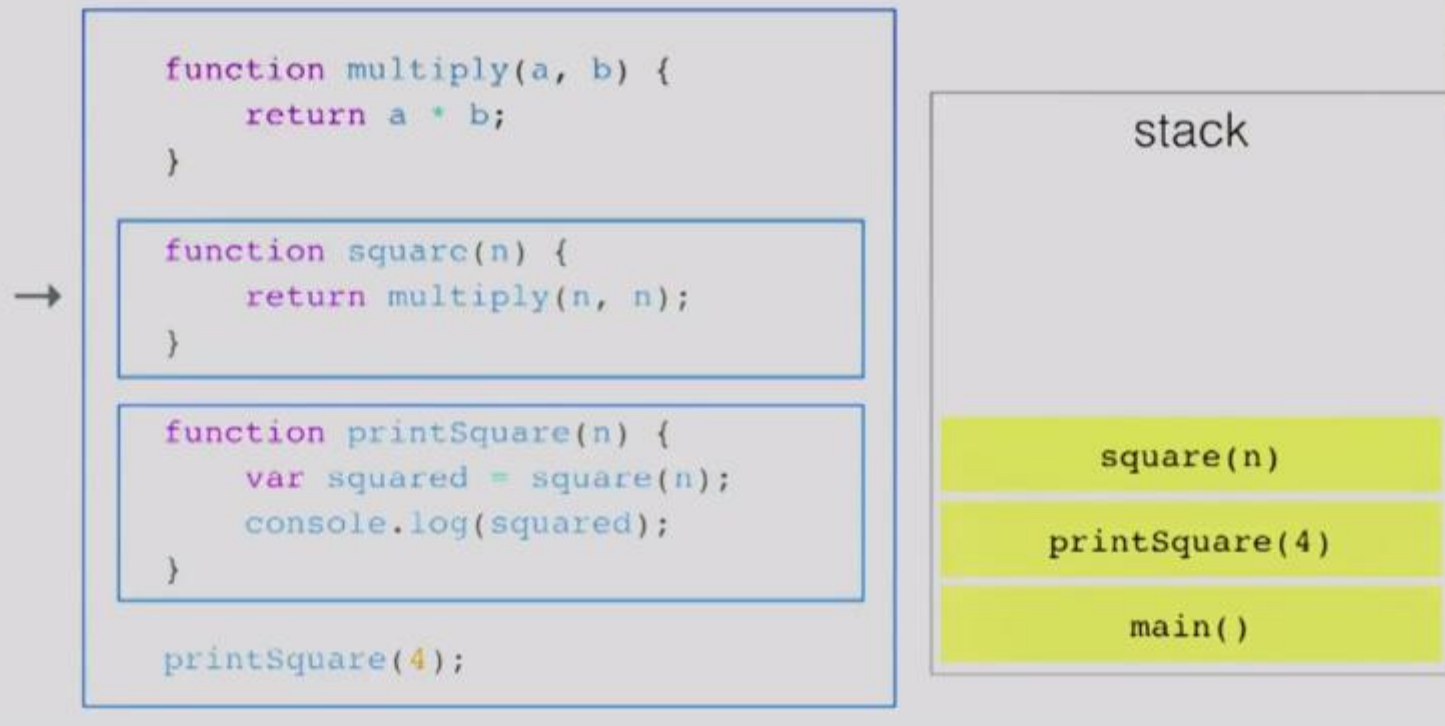
```
printSquare(4);
```

stack

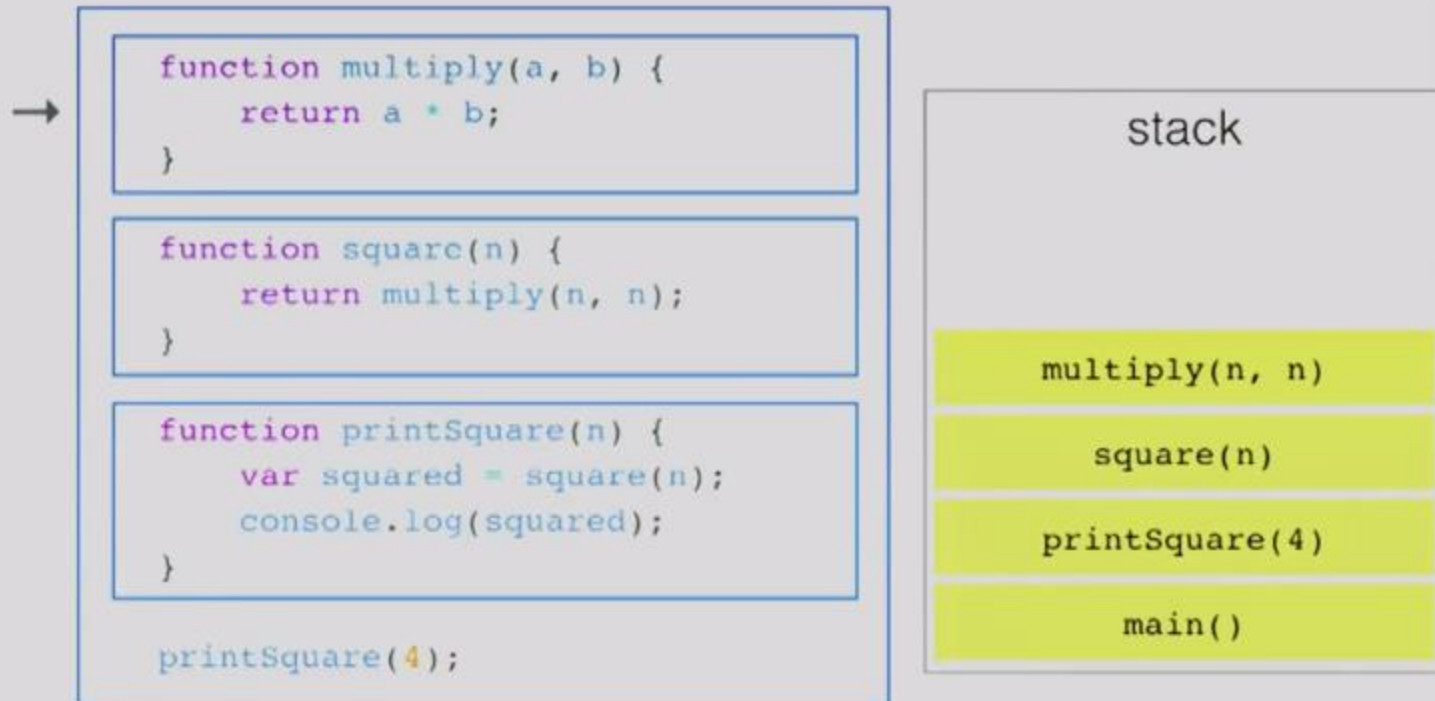
printSquare(4)

main()

Call Stack

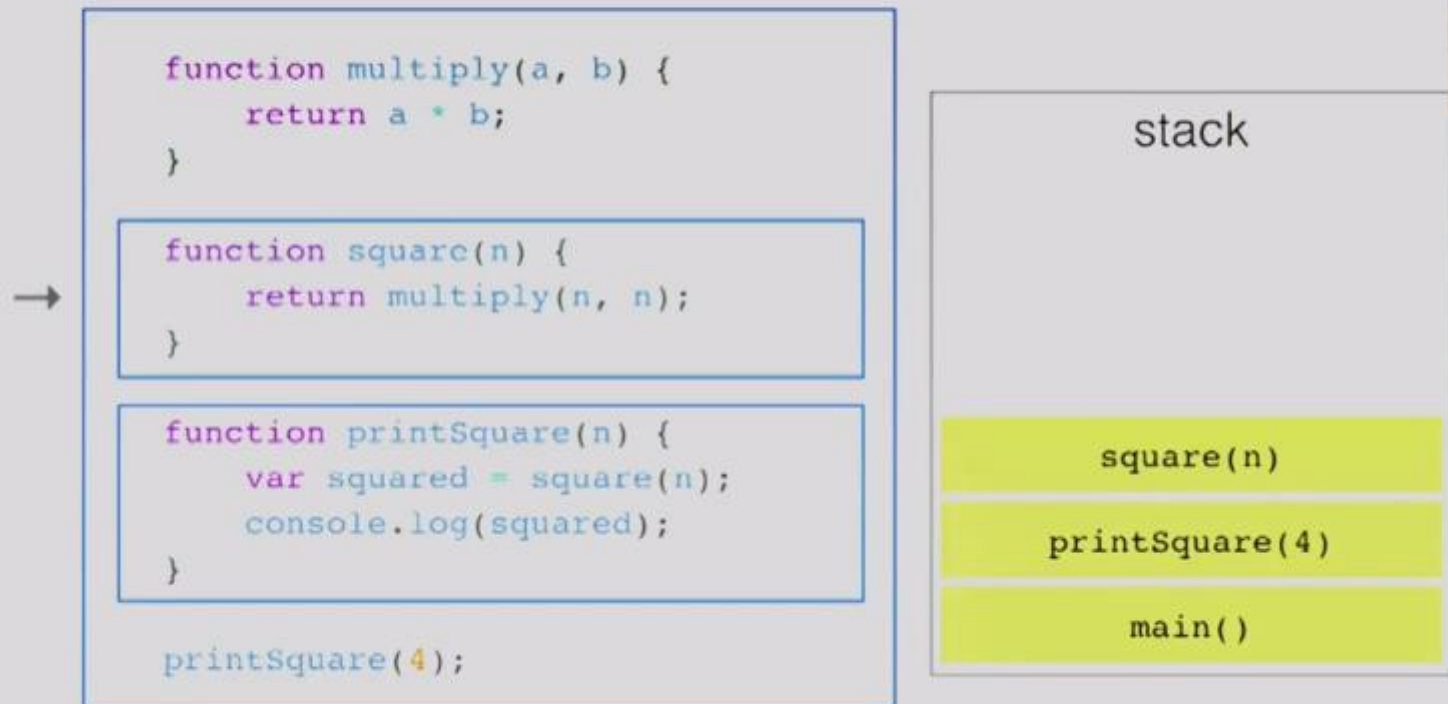


Call Stack



And now for the way back...

Call Stack



Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

→

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```



stack

```
console.log(squared)
```

```
printSquare(4)
```

```
main()
```

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```



stack

`printSquare(4)`

`main()`

Call Stack


```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
→ printSquare(4);
```

stack

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(n) {  
  return multiply(n, n);  
}  
  
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}  
  
printSquare(4);
```



stack

Finished!

How does this one behave?

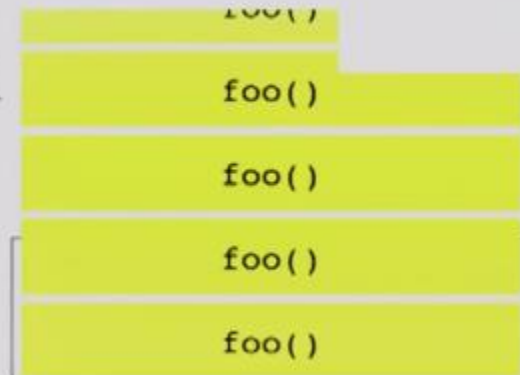
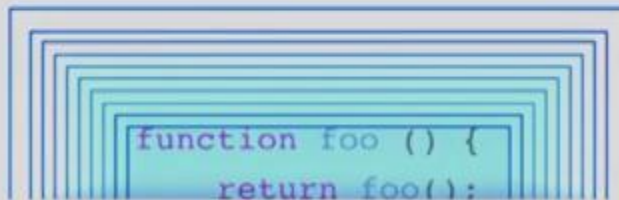
Call Stack

```
function foo () {  
    return foo();  
}
```

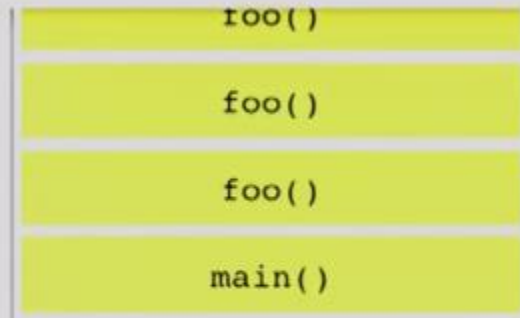
```
foo();
```

stack

Call Stack



X **RangeError: Maximum call stack size exceeded**



What happens on slow calls?

- Calls that are **Blocking** ! For example

Call Stack

```
var foo = $.getSync('//foo.com');  
var bar = $.getSync('//bar.com');  
var qux = $.getSync('//qux.com');  
  
console.log(foo);  
console.log(bar);  
console.log(qux);
```

stack

Let's run and see...

Call Stack

```
var foo = $.getSync('//foo.com');  
var bar = $.getSync('//bar.com');  
var qux = $.getSync('//qux.com');  
  
console.log(foo);  
console.log(bar);  
console.log(qux);
```

stack

main()

Call Stack

```
var foo = $.getSync('//foo.com');  
var bar = $.getSync('//bar.com');  
var qux = $.getSync('//qux.com');  
  
console.log(foo);  
console.log(bar);  
console.log(qux);
```

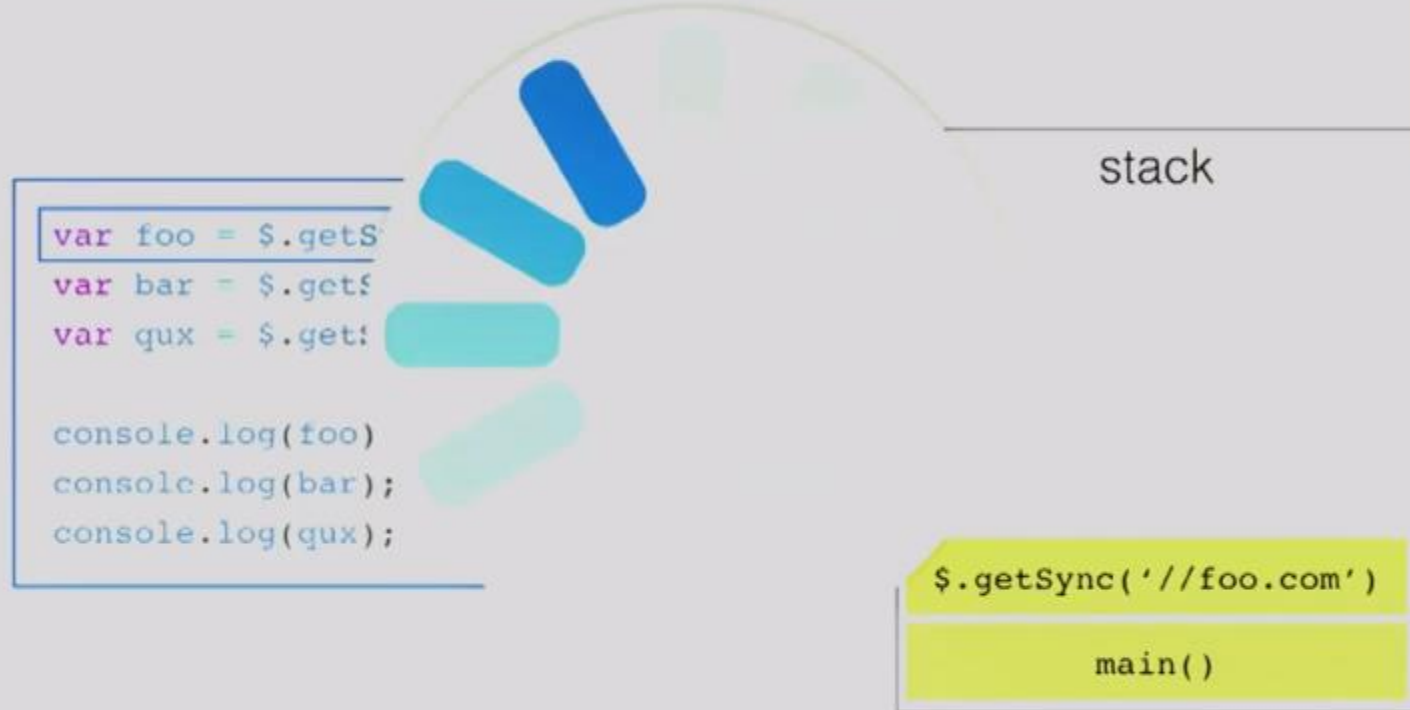
stack

```
$.getSync('//foo.com')
```

```
main()
```

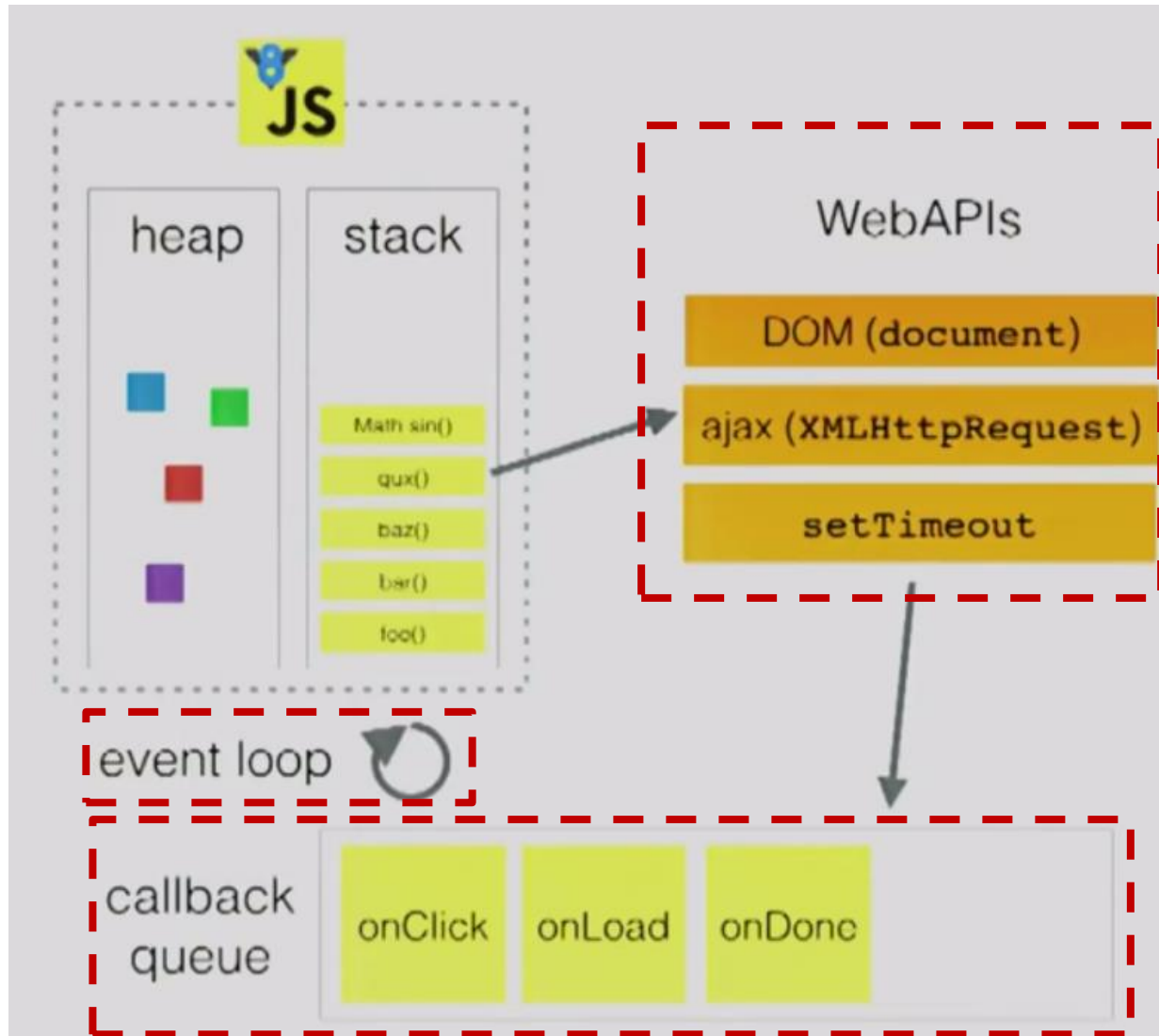
It's a synchronous call, so we have to wait...

Call Stack



And wait...

How to solve it?



An easy example

```
console.log('Hi');  
  
setTimeout(function cb() {  
    console.log('there');  
}, 5000);  
  
console.log('almost Bye');
```

Let's try to see how this code works ([link](#))

Notice! – the Event Loop will **pull** from the Queue **only if the call stack is empty**

An example with key events + timeout

[link](#)

Loupe

- Event Loop visualizer for JS code:

➤ <http://latentflip.com/loupe/>