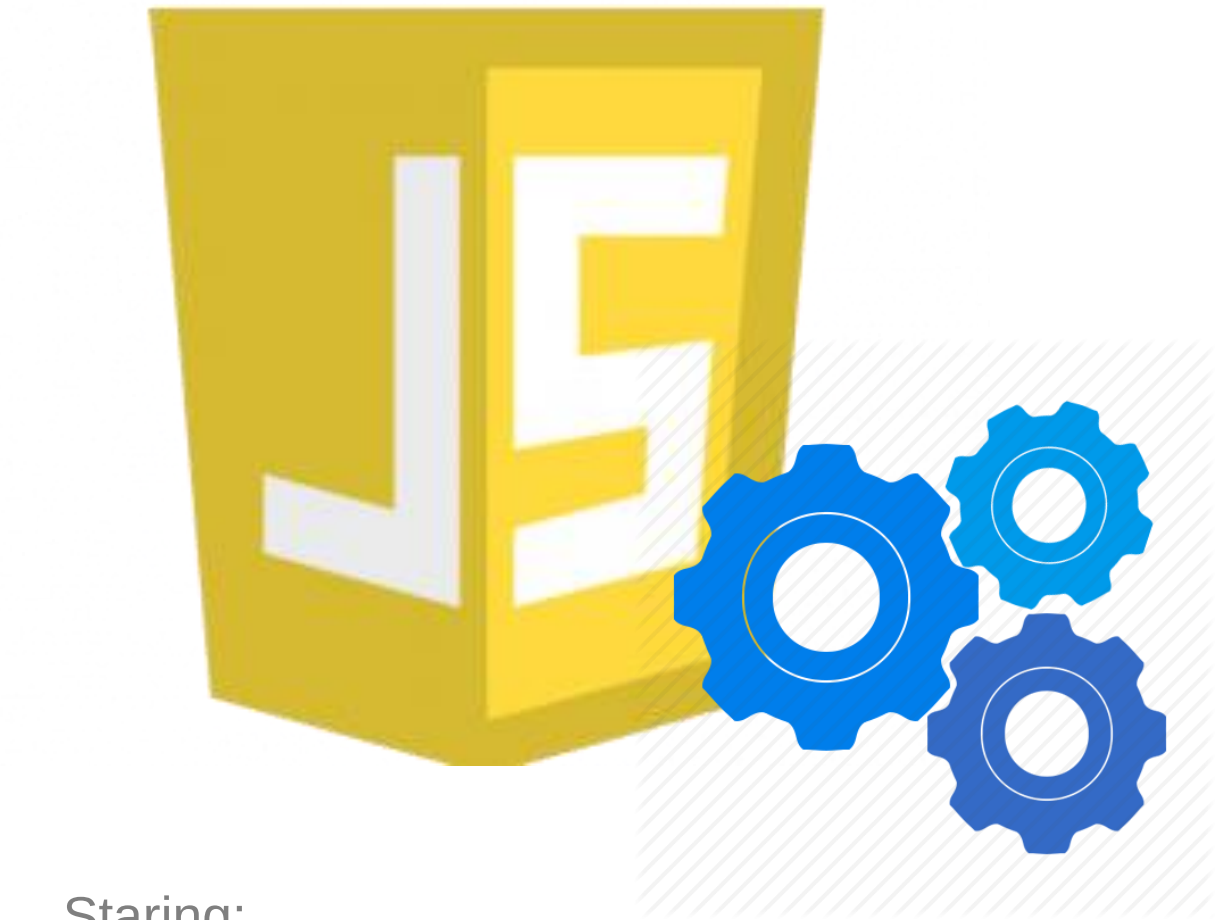
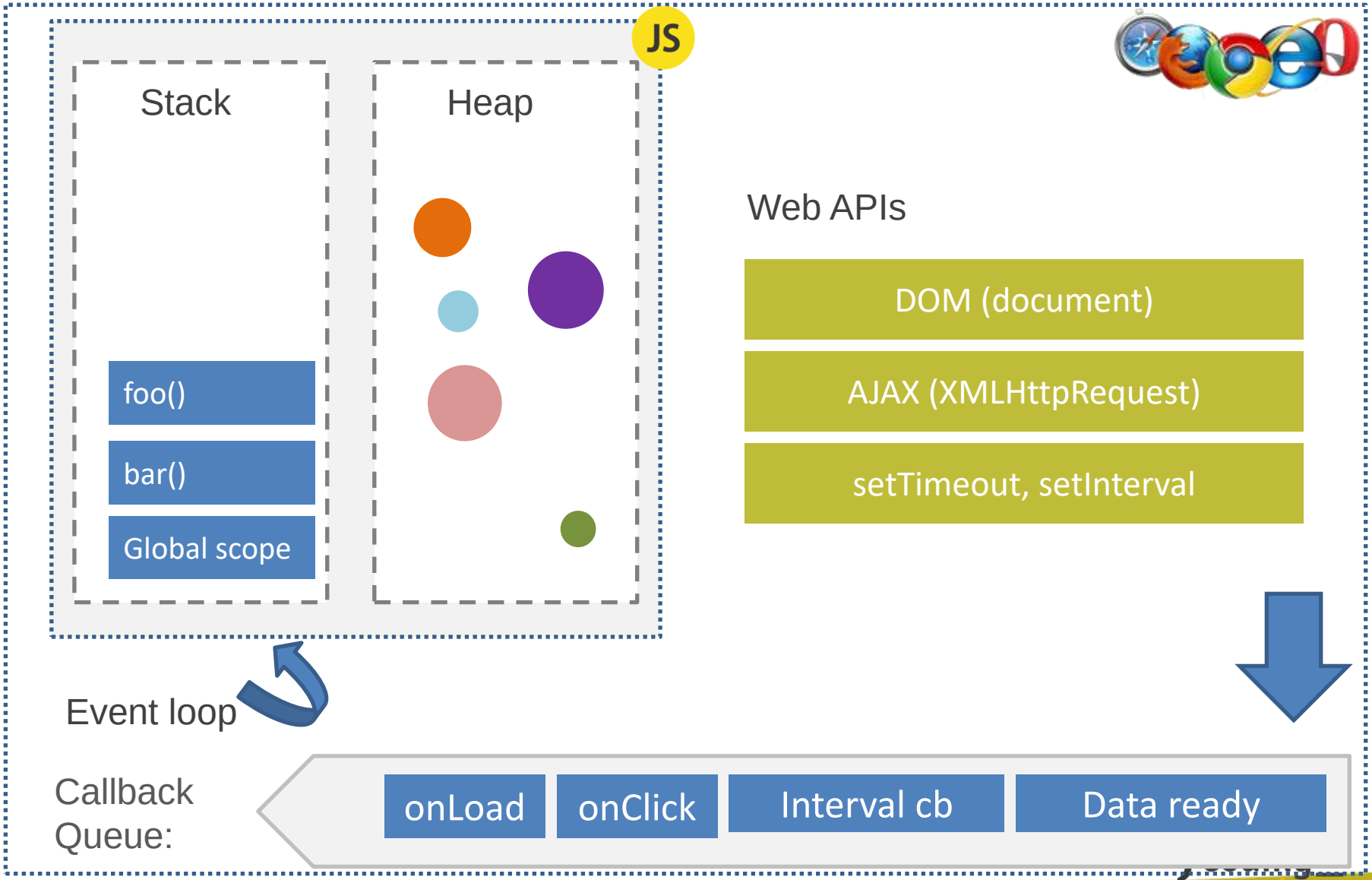
# Javascript Execution Model



Staring:

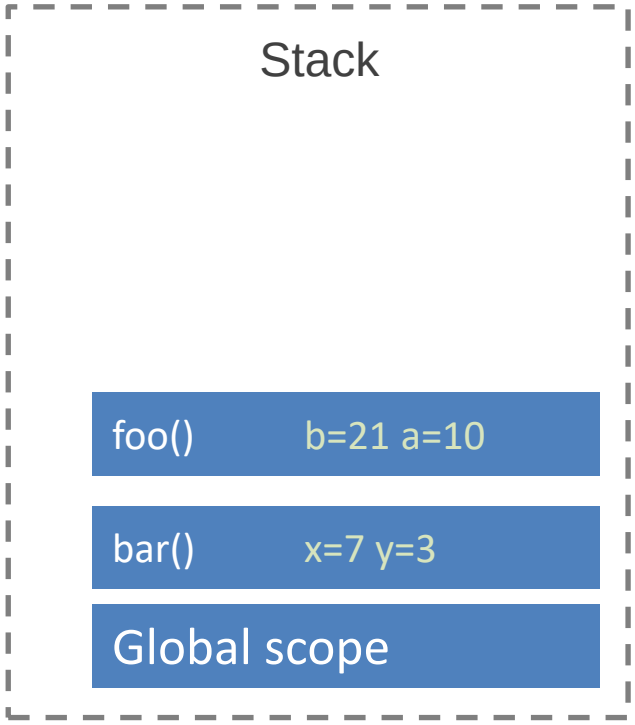
- the main (UI) thread
- Asynchronous actions

# Execution model - Visual Representation



# JS uses the call-stack to manage the execution

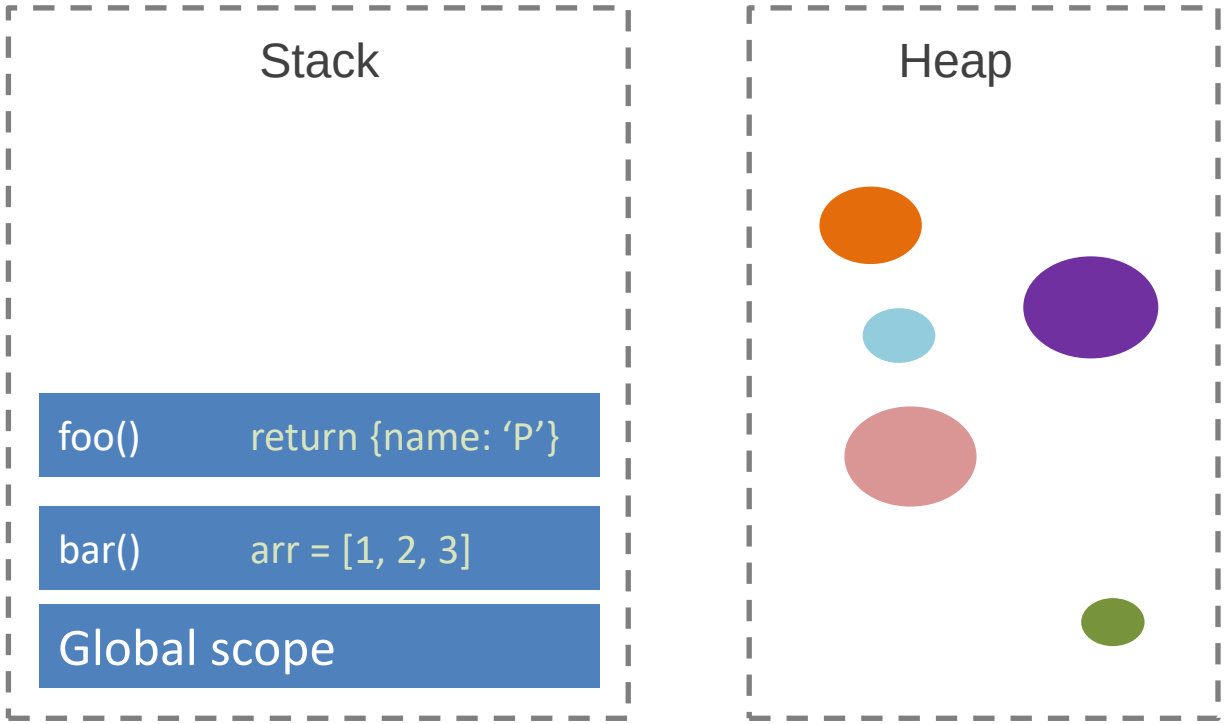
```
function foo(b) {
    var a = 10
    return a + b + 11
}

function bar(x) {
    var y = 3
    return foo(x * y)
}

console.log(bar(7))
```



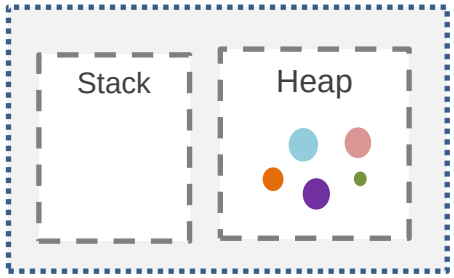
# The Heap

Objects are allocated in the heap

which is large mostly unstructured region of memory.



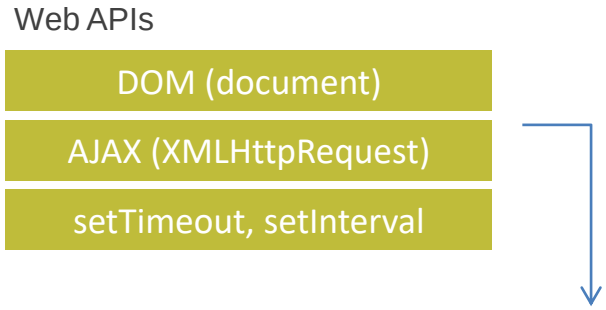
# Messages Queue / Callback Queue

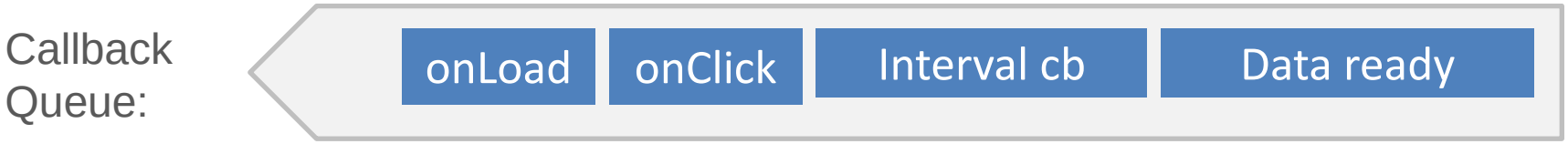



The JS engine contains a message queue, which is a list of messages to be processed.
A function is associated with each message.

When the stack is empty, a message is taken out of the queue and the function is processed.

The processing ends when the stack becomes empty again.

# Run to completion & Never blocking

**Run to completion**
Each message is processed completely before any other message is processed.

- This is the single threaded nature of javascript
- This means our processing should be kept short and sweet

**Never Blocking**
Javascript *never* blocks!

Besides:
1. the native popups: *alert, prompt, confirm*
2. There is a synchronous option (hardly used) for I/O such as in XMLHttpRequest

# Javascript Execution Model