Event loop, callback queue, microtask queue. Working of browser.

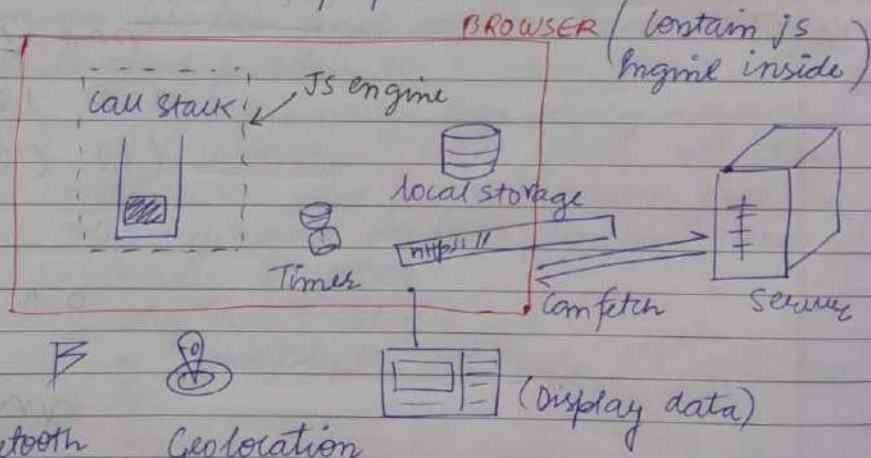
→ 10% developer understand js event loop.

→ Javascript engine executes our code by using call stacks.

→ As soon as call stack receive a function, it quickly execute it, without any delay.

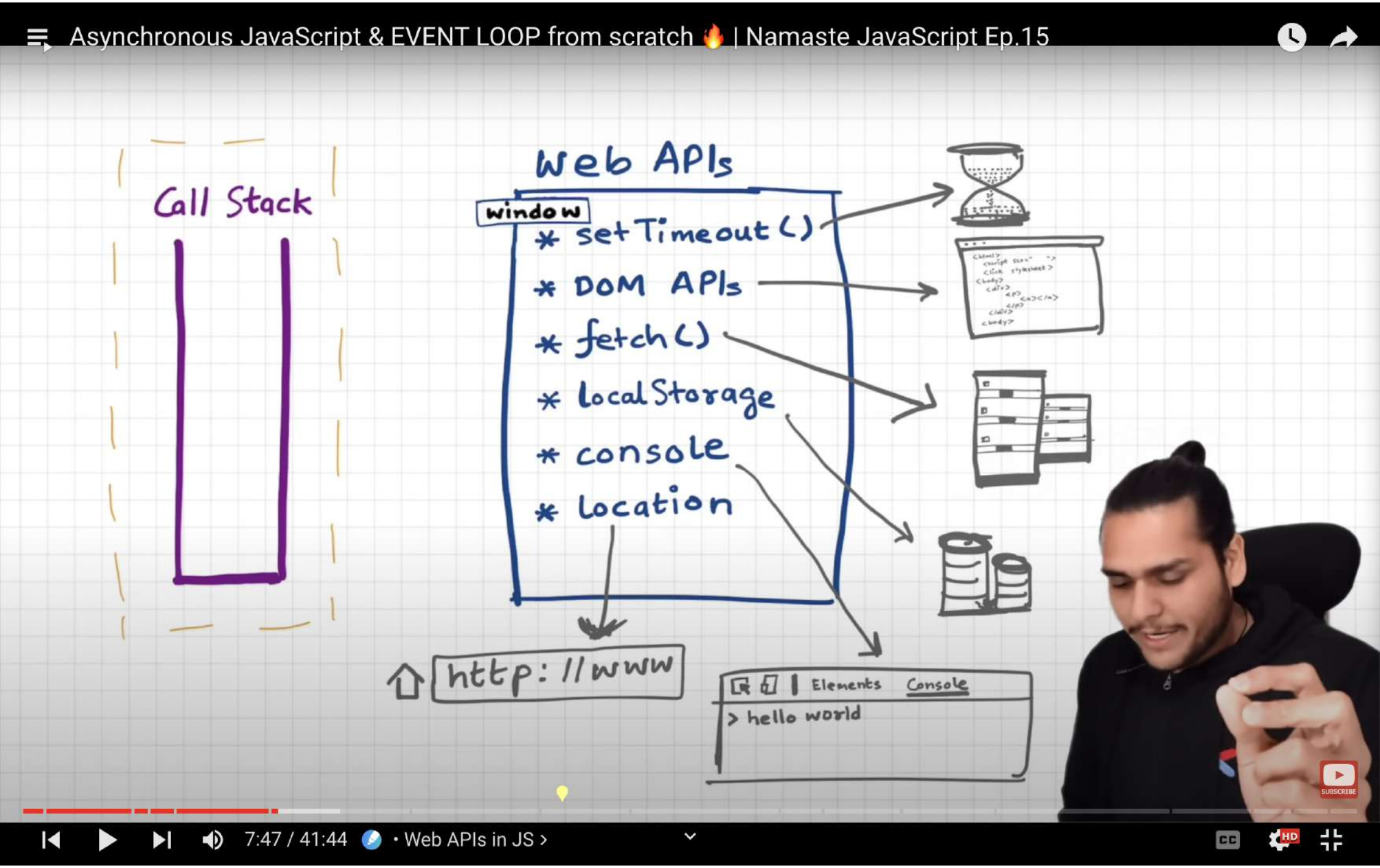
How to wait for sometime, like 5 seconds.
JS can't do that.

Need some extra superpower.



→ We need access to these functionalities.
→ Need some way to acess these things.

* Can access these functionalities using web API's.


Asynchronous JavaScript & EVENT LOOP from scratch 🔥 | Namaste JavaScript Ep.15
Call Stack
Web APIs
window
* setTimeout()
* DOM APIs
* fetch()
* localStorage
* console
* location
http://www
Elements
Console
> hello world
SUBSCRIBE
7:47 / 41:44
• Web APIs in JS ›
HD
CC

Call Stack

cb

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

setTimeout(function cb() {
  console.log("Callback");
}, 5000);

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

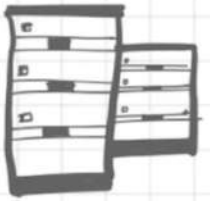
Web APIs

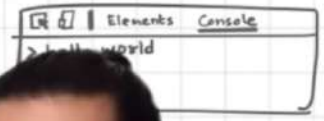
window

* setTimeout()
* DOM APIs
* fetch()
* console

cb

5000ms





EVENT LOOP

Callback Queue

cb

Start
End
Callback

→ SetTimeout is not a part of JS.
(document. (dom), fetch, local storage, console) (not a part of JS)

These are web API's (part of browser)
→ Use them to acess browsers functionality.
(can acess dom tree by dom API's)

→ All web API's are attached to global object.

Call Stack

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

document.getElementById("btn")
.addEventListener("click", function cb() {
  console.log("Callback");
});

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

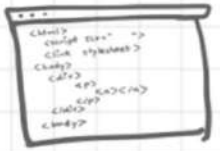
Web APIs

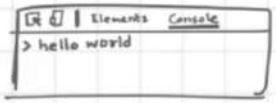
window

* setTimeout()
* DOM APIs
* fetch()
* console

click

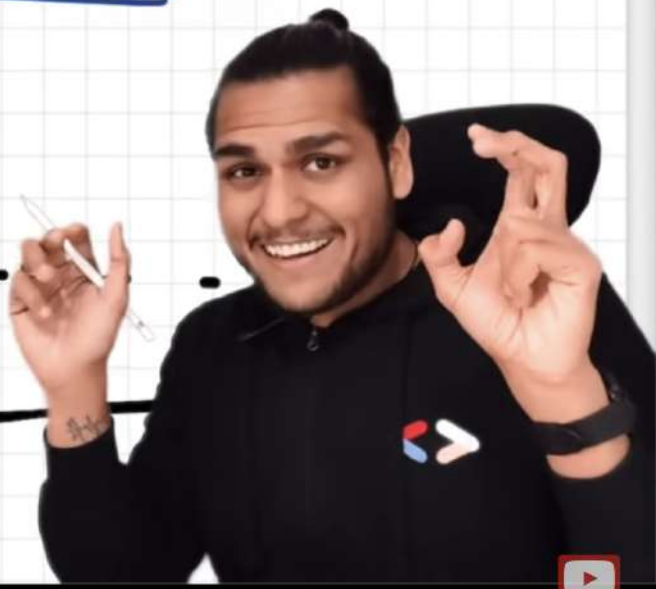
cb()





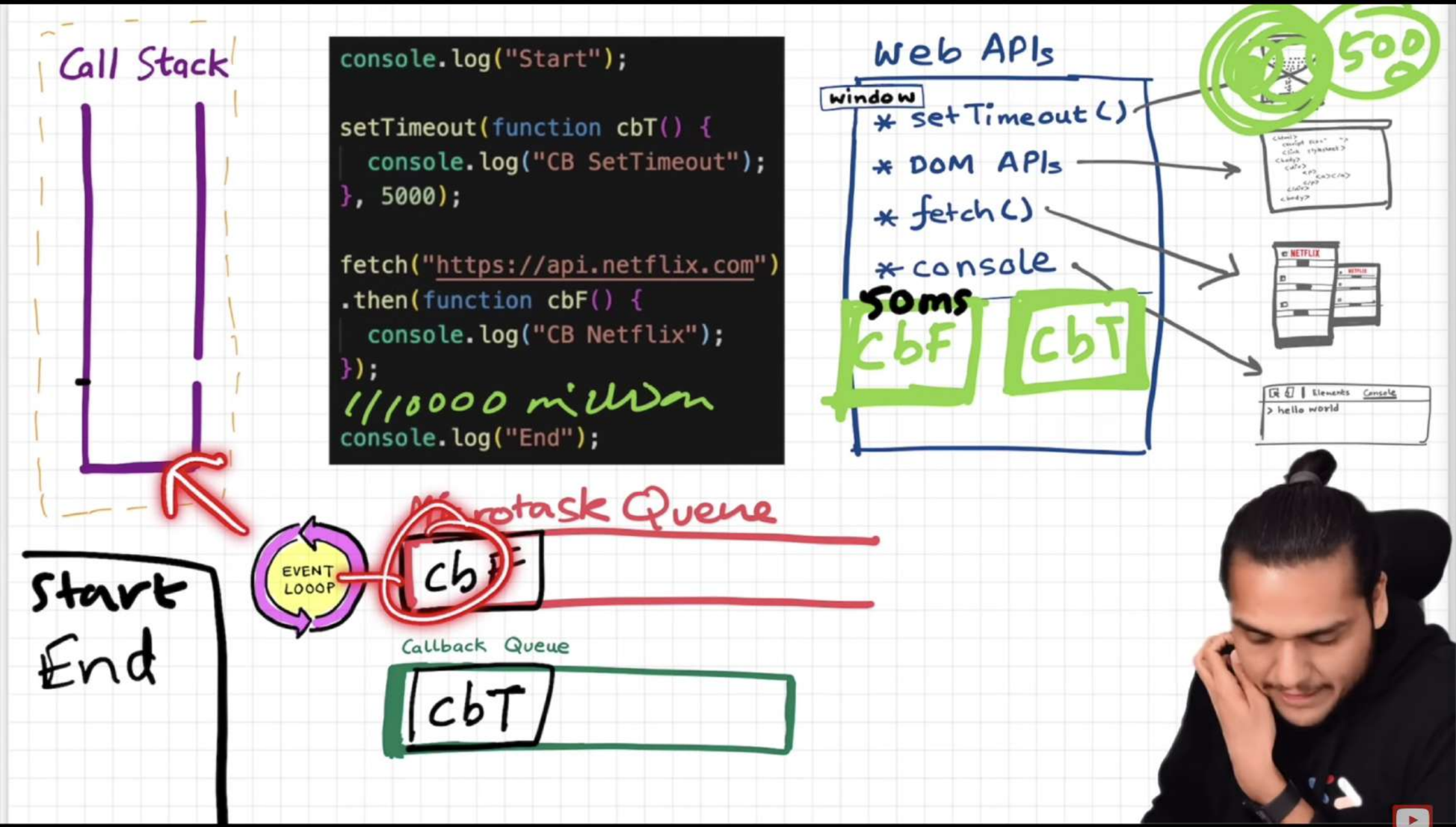
Start
End
Callback

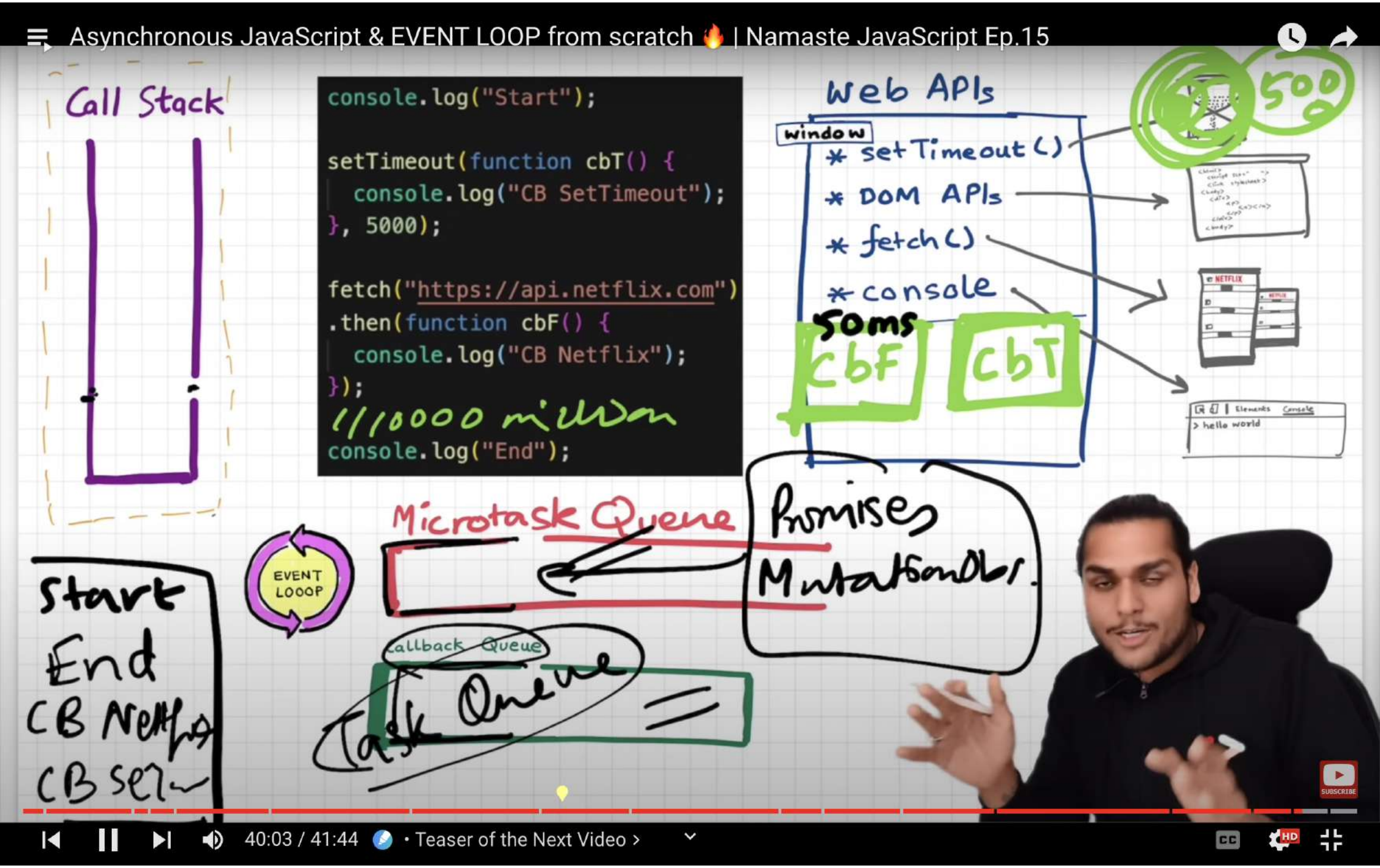
Callback Queue



After the timer expires, the callback func is pushed to the callback queue.

The work of the event loop is to check, if any time our call stack is empty & callback queue have some remaining func, if yes, then it simply push that callback to call stack.

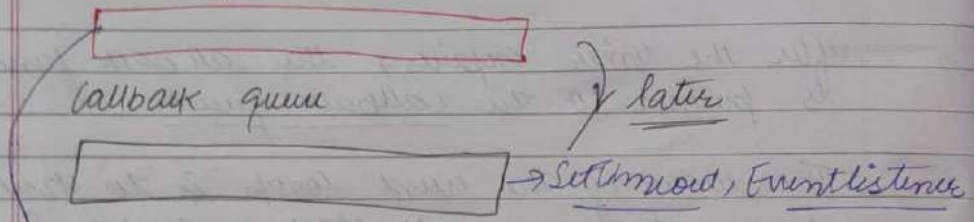

Call Stack
console.log("Start");
setTimeout(function cbT() {
console.log("CB SetTimeout");
}, 5000);
fetch("https://api.netflix.com")
.then(function cbF() {
console.log("CB Netflix");
});
//10000 million
console.log("End");
Web APIs
window
* setTimeout()
* DOM APIs
* fetch()
* console
5oms
cbF
cbT
500
NETFLIX
Elements
Console
> hello world
Microtask Queue
EVENT LOOP
cbF
Callback Queue
cbT
Start
End
SUBSCRIBE


Asynchronous JavaScript & EVENT LOOP from scratch 🔥 | Namaste JavaScript Ep.15
Call Stack
console.log("Start");
setTimeout(function cbT() {
console.log("CB SetTimeout");
}, 5000);
fetch("https://api.netflix.com")
.then(function cbF() {
console.log("CB Netflix");
});
//10000 million
console.log("End");
Web APIs
window
* setTimeout()
* DOM APIs
* fetch()
* console
5oms
cbF
cbT
Microtask Queue
Promises
MutationObserver
EVENT LOOP
Callback Queue
Task Queue
Start
End
CB Netflix
CB setTimeout
> hello world
40:03 / 41:44
• Teaser of the Next Video ›

Fetch will go and request an api call and it will return a promise, & we have to pass a call back function which will be executed when the promise is resolved.

Even if the call back is ready, it will wait in callback queue until the call stack got empty.

Microtask queue (HIGHER PRIORITY)



1 → In case of promises, call back is sent to microtask queue.

2 → Mutation Observer keeps on checking if there is some mutation in DOM true or not? If yes, it can execute some callback function

read about both (HW)

Callback queue / task queue.

* There could be condition of starving if a microtask create another microtask. Then the callback queue will never get a chance to execute.

Read more (HW)