How JS handles sync and async tasks ??

→ JS prog. lang. is Single threaded.

→ console.log ("Start")

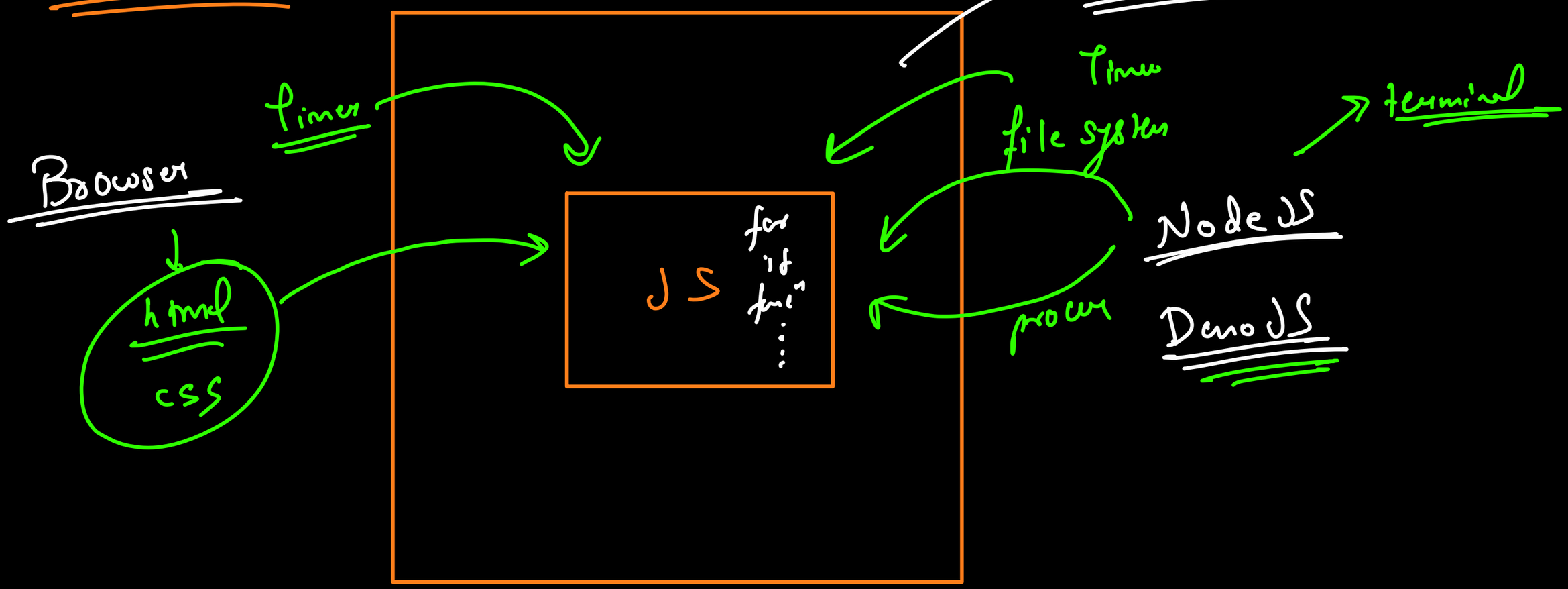
for (i=0; i<10^10 ; i++) {} ← blocking

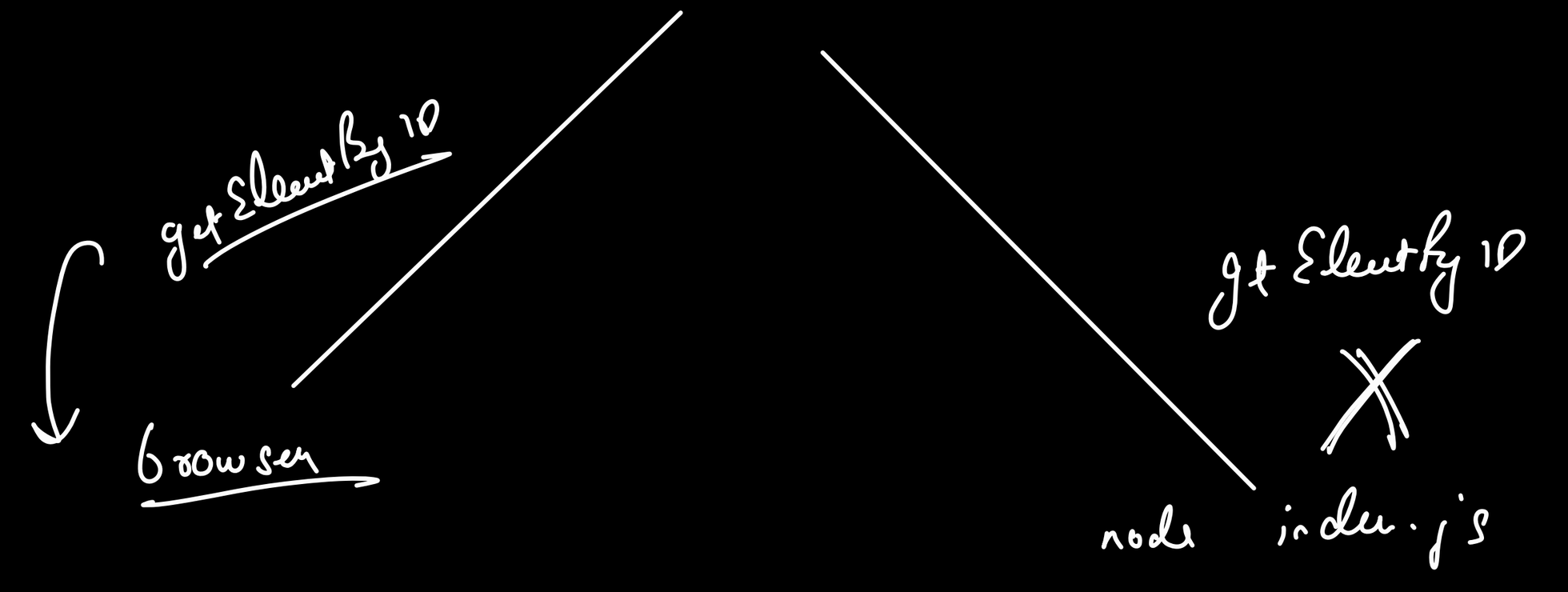
↳ console.log ("end")

setTimeout (cb, ms)

→ functions like setTimeout, setInterval, getElementById etc are not native part of JS.

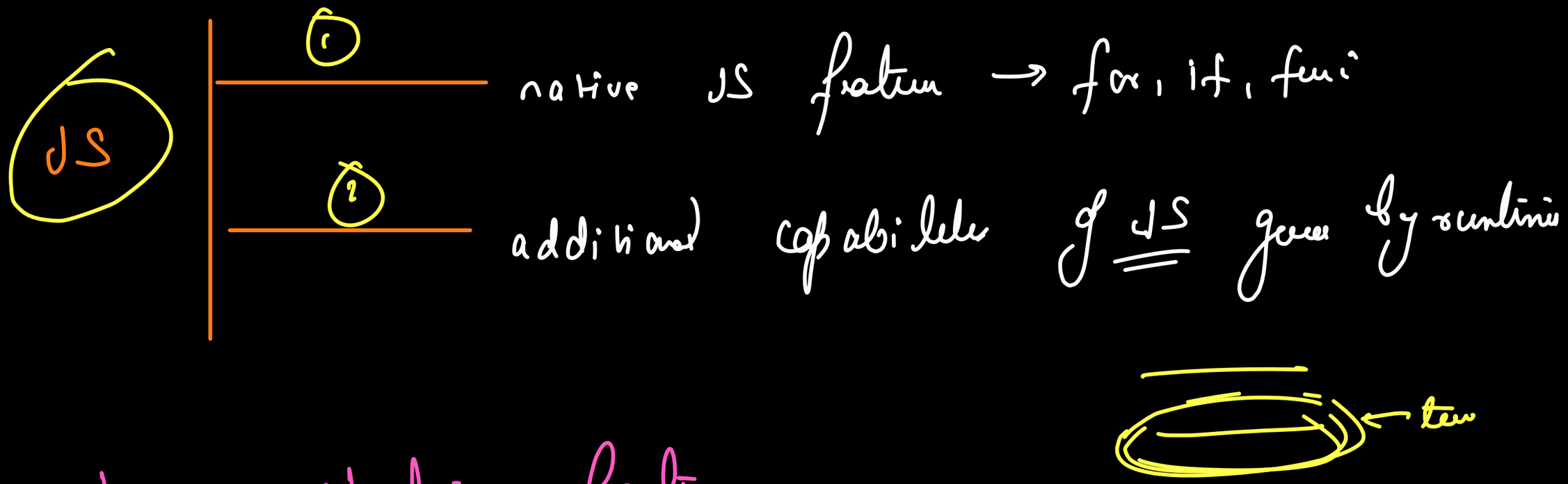
Runtime Env





Browser → document.getElementById
document.getElementsByClassName
⋮
setTimeout
setInterval
XMLHttpRequest

} =



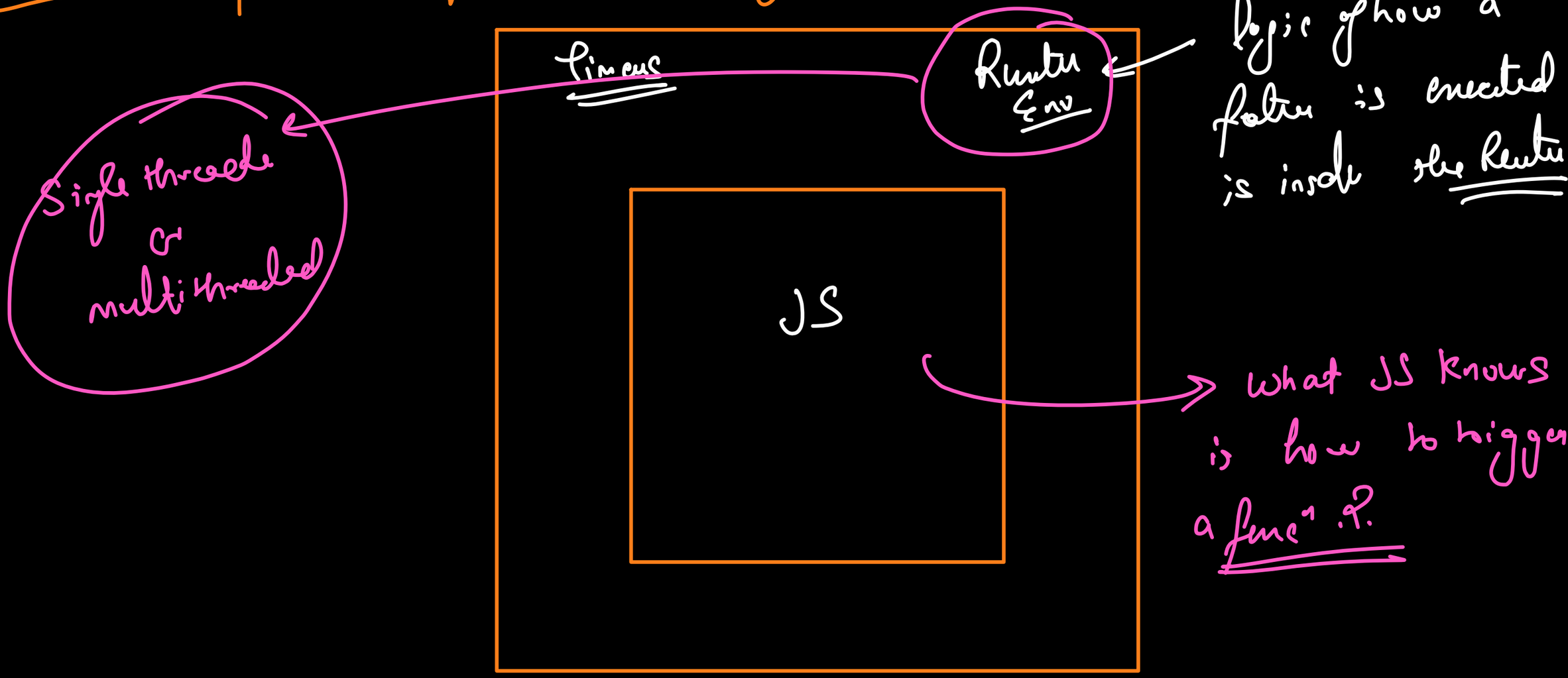
Case1 → Native feature

Because JS is single threaded, every native piece of code will always execute on the main thread and it will be always Sync in nature i.e. if you have a time

consuming piece of code written by nativ JS feature then it will be blocking

Summary → Every native JS code is synchronously executed

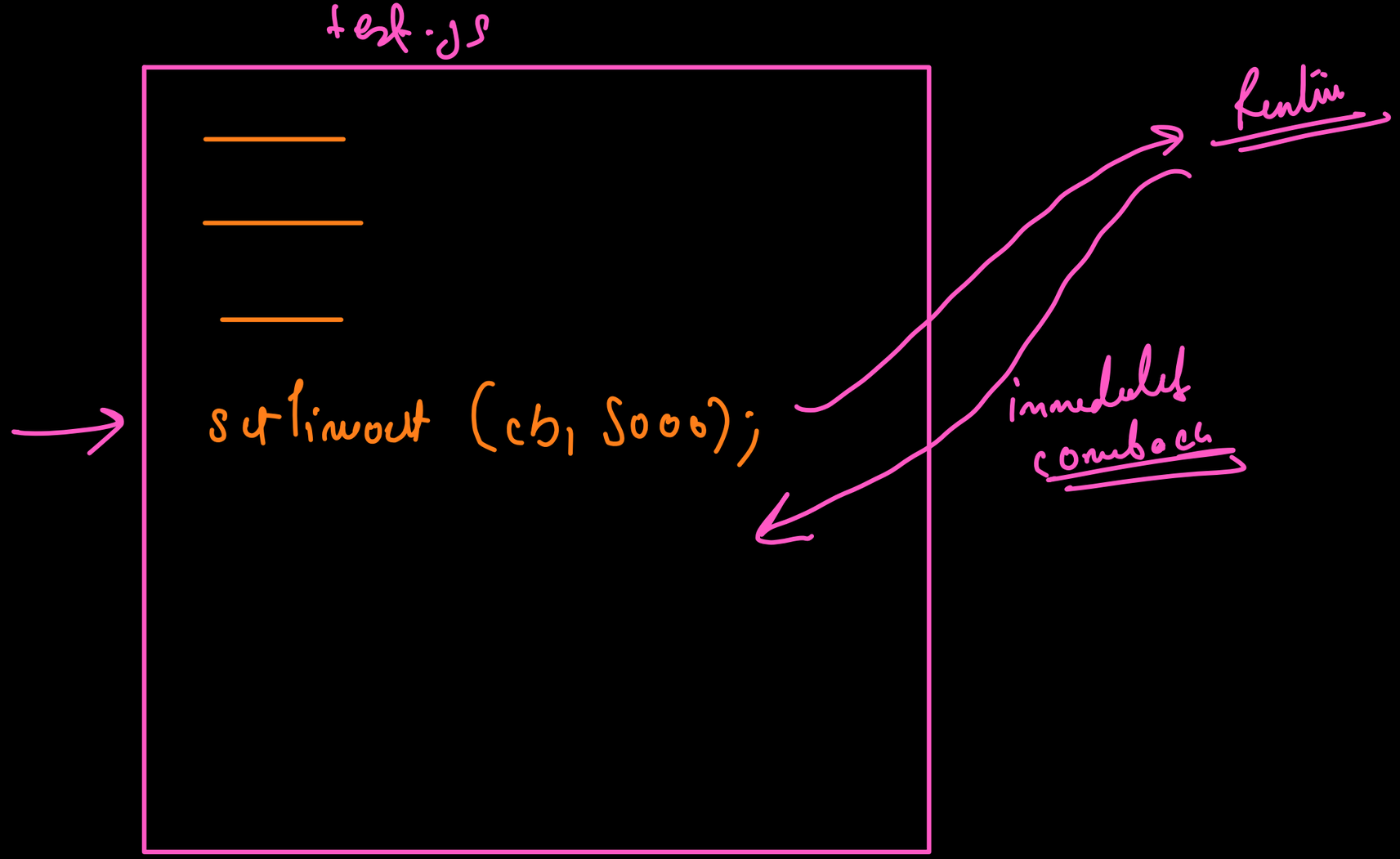
Case-2 features provided by Runtime Env



whenever we execute a Runtime feature from JS

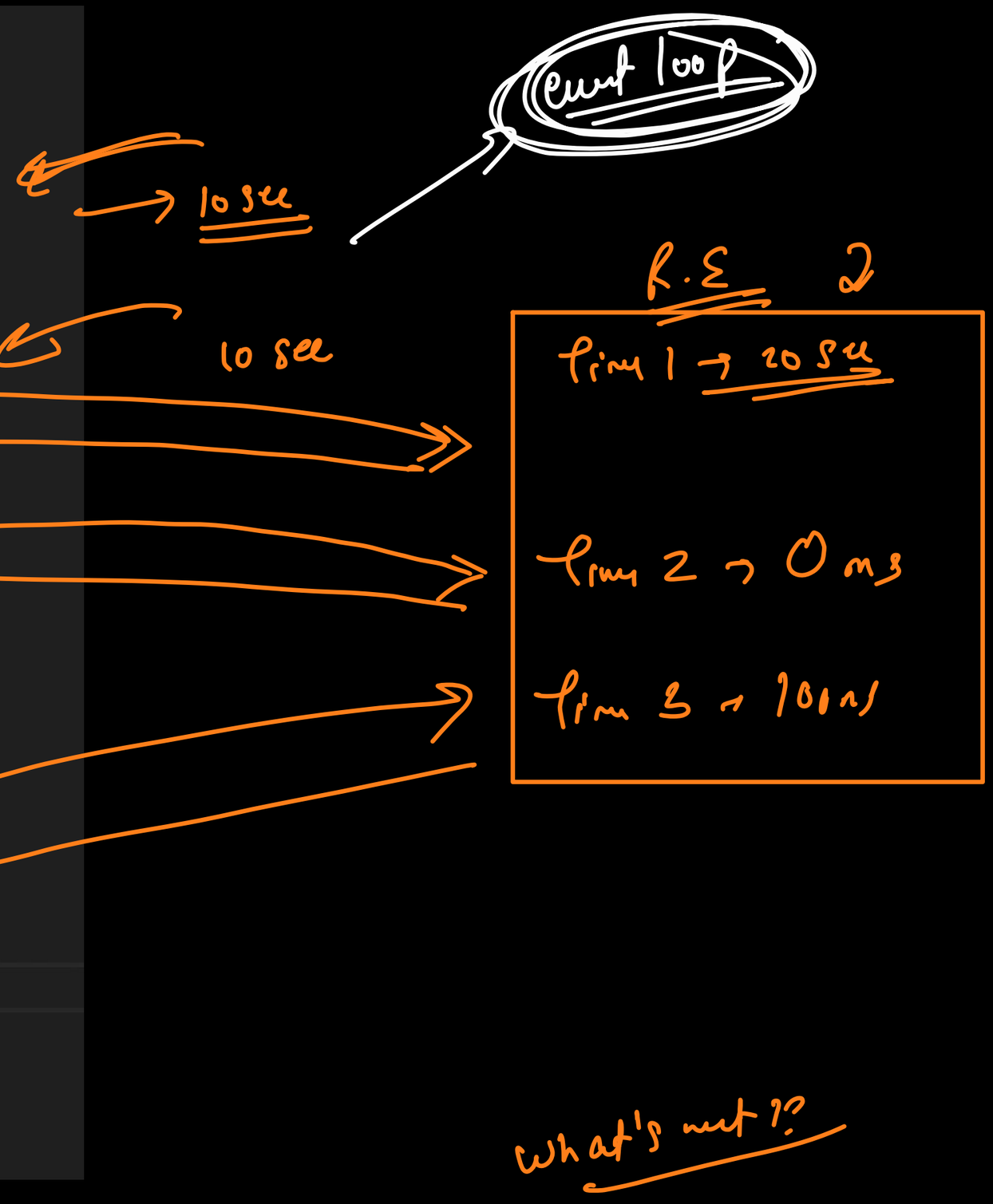
following happens :-

(1) The moment JS code encounters a functionality which is a Runtime feature, all it does is trigger the request to Runtime and do not wait for the execution, so immediately comes back.



OKAY!! JS just triggers the runtime feature, then what happens after that ?? Bcaus JS is back on the main thread, & let's say timer completes what happens then ::

```
1   // starting here
2
3   for(let i = 0; i < 10000000000; i++) {
4       // some work
5   }
6   for(let i = 0; i < 10000000000; i++) {
7       // some more work
8   }
9   setTimeout(() => {
10      console.log("timer 1 done");
11  }, 20000);
12  setTimeout(() => {
13      console.log("timer 2 done")
14  }, 0);
15  for(let i = 0; i < 20; i++) {
16      // some more work
17  }
18  setTimeout(() => {
19      console.log("timer 3 done");
20  }, 100);
21
22  // end here
```



Any point of time if any code is left to be enabled/triggered by JS on the MAIN THREAD the response of Runtime feature has to wait, they cannot interrupt the main thread.

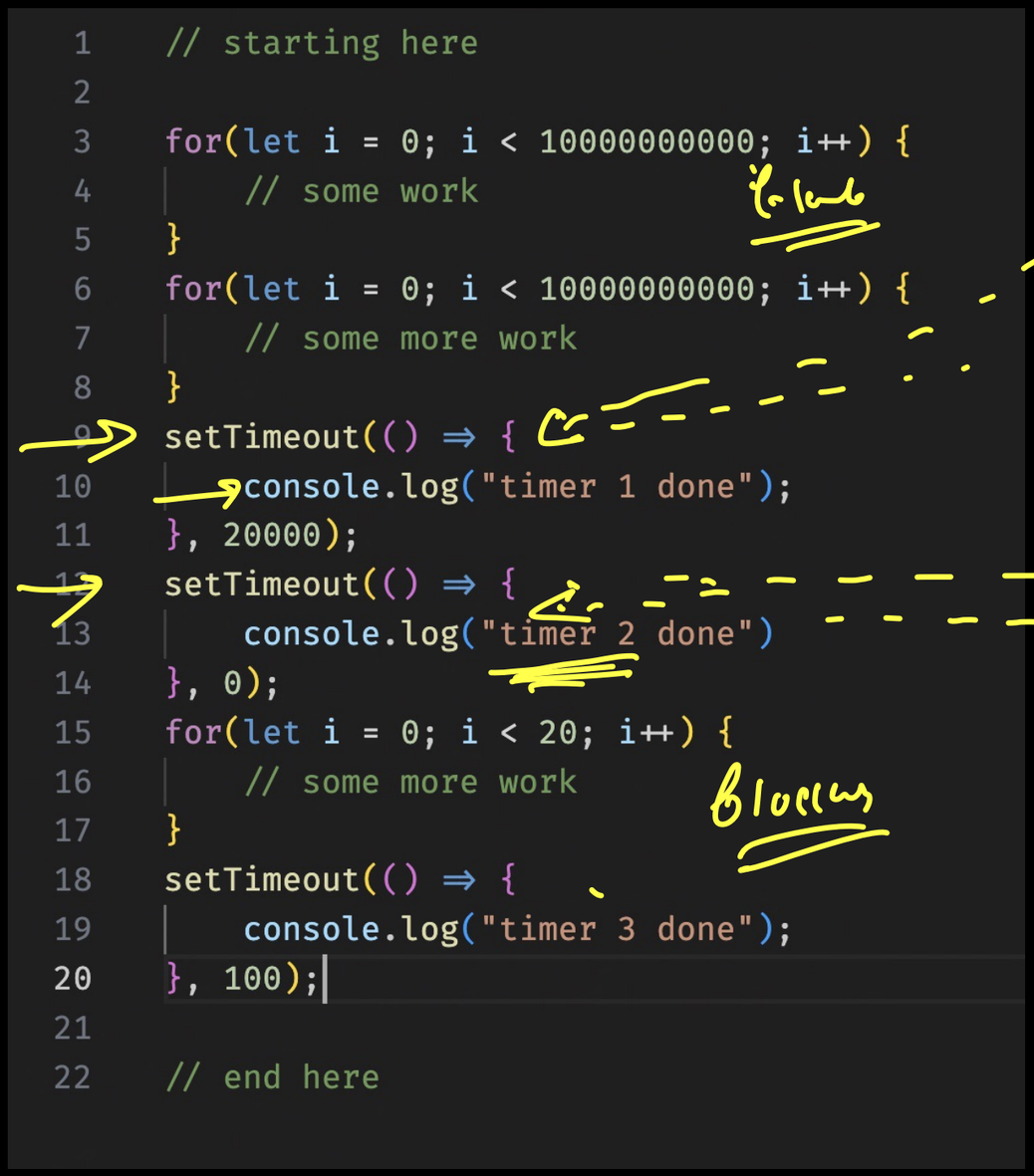
Sole purpose of event loop is to continuously check is the main thread free or not.

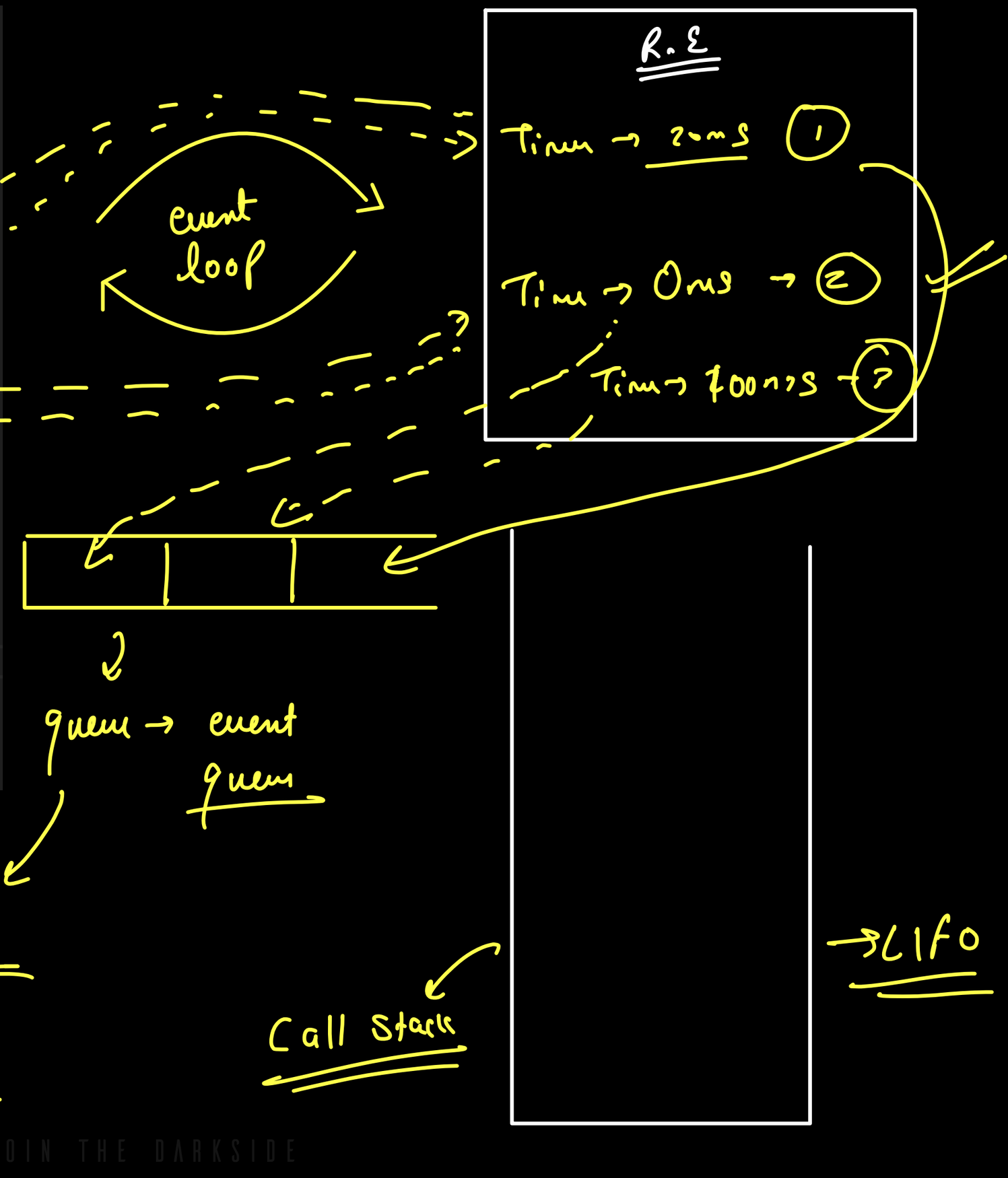
When R.E completes the task, it will immediately push to the Cb func^ in the queue.

```
// starting here

for(let i = 0; i < 10000000000; i++) {
    // some work
}
for(let i = 0; i < 10000000000; i++) {
    // some more work
}
setTimeout(() => {
    console.log("timer 1 done");
}, 20000);
setTimeout(() => {
    console.log("timer 2 done")
}, 0);
for(let i = 0; i < 20; i++) {
    // some more work
}
setTimeout(() => {
    console.log("timer 3 done");
}, 100);

// end here
```

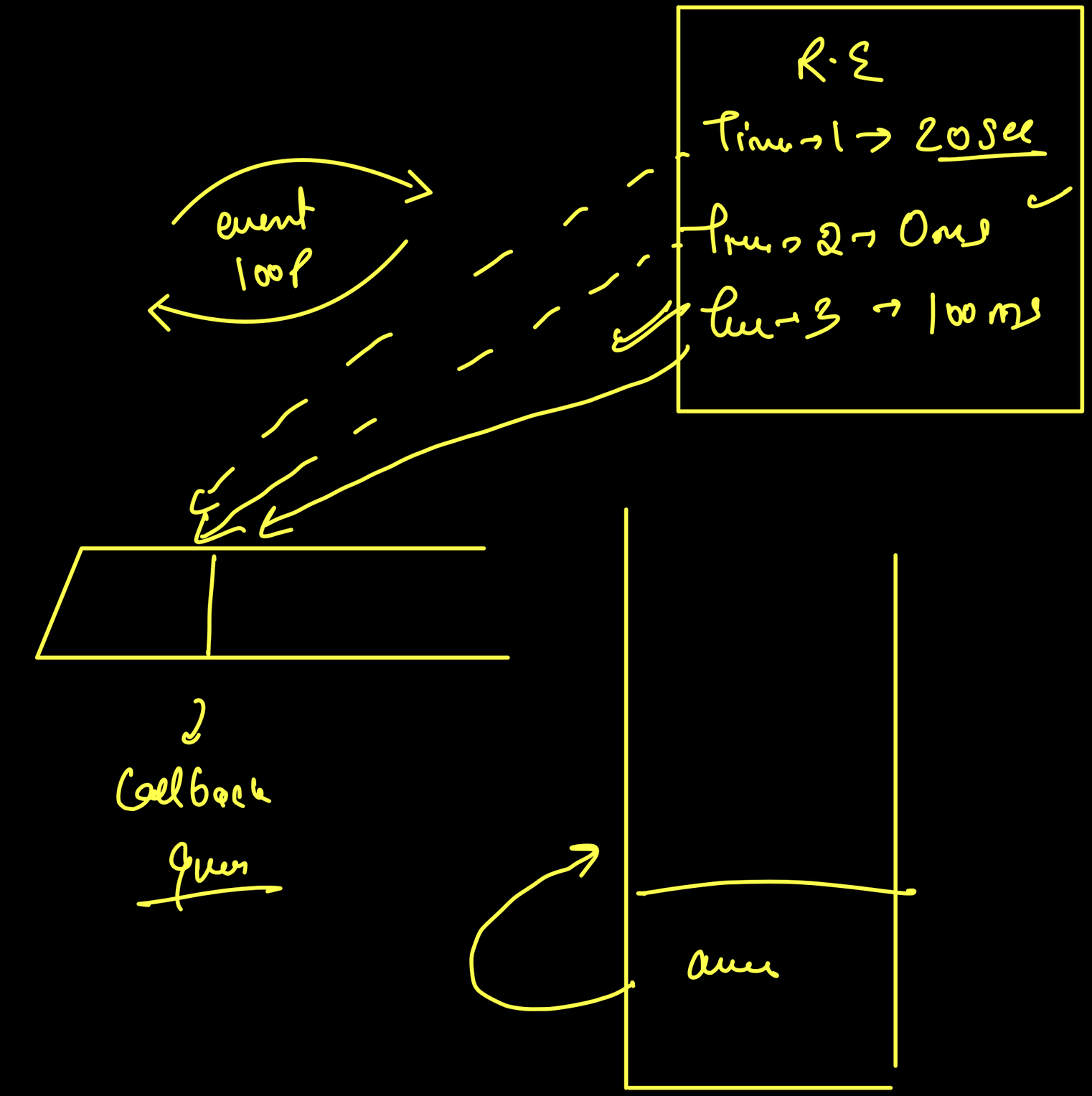




```
// starting here

for(let i = 0; i < 10000000000; i++) {
    // some work
}
for(let i = 0; i < 10000000000; i++) {
    // some more work
}
setTimeout(() => {
    console.log("timer 1 done");
}, 20000);
setTimeout(() => {
    console.log("timer 2 done")
}, 0);
for(let i = 0; i < 20; i++) {
    // some more work
}
setTimeout(() => {
    console.log("timer 3 done");
}, 100);

// end here
```



FCFS

JOIN THE DARKSIDE