• github: divyaranjan10

DATE

CHAP 17: Trust Issue with setTimeout ()

• Problem with setTimeout() : It doesn't neces- -sarily mean that the setTimeout will run after the given 'n' seconds, it totally depends on the call stack.

• For ex:

```
console.log ('Start"),
setTimeout(function cb(){
   console.log ("Callback");
}, 5000)·
console.log (" End");
:
:
~~Million of lines of code~~ (10000 ms) (17.1)
:
:
```

• (17.1) For example, after the setTimeout, our code has millions of lines of code that takes, let's say 10s to finish up but by the time " setTimeout 5s timer" would have finis-

-ad before and would be waiting in the 'Callback Queue' and waiting for the 'million of lines of code that would take 10s' to finish so that call stack would be empty and then the 'cb' function can get moved into the call stack. That's why we have trust issue with setTimeout() that after stating the '5000ms' timer, it may or may not ~~finish~~ ~~up~~ execute after 5s.

• Code example:

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

• Output:

```
Start
End
While Expires....
Callback
```

```
// Suppose it's that million lines of code
let startDate = new Date().getTime();
let endDate = startDate;
```

DATE

```
while (endDate < startDate +10000) {
    endDate = new Date().getTime();
}
console.log ("while expires...");
```

=> So, this example is similarly to the prev-
-ious example, read the theory in the
previous page.

• Concurrency model in JS : JS runtime is
single threaded which means that it can
execute one piece of code at a time.

• setTimeout (0) :

```
console.log ("Start");
setTimeout (function cb(){
    console.log ("Callback");
}, 0);
console.log ("End");
```

• Output :

```
Start
End
Callback
```

DATE________

- So, even it has "timer = 0", it still has to wait, before whole program is executed except it. So, the setTimeout will have to register and attach a timer of 0 to itself and place in a separate space and have to wait in the callback queue ~~before the~~ till the call stack gets empty.