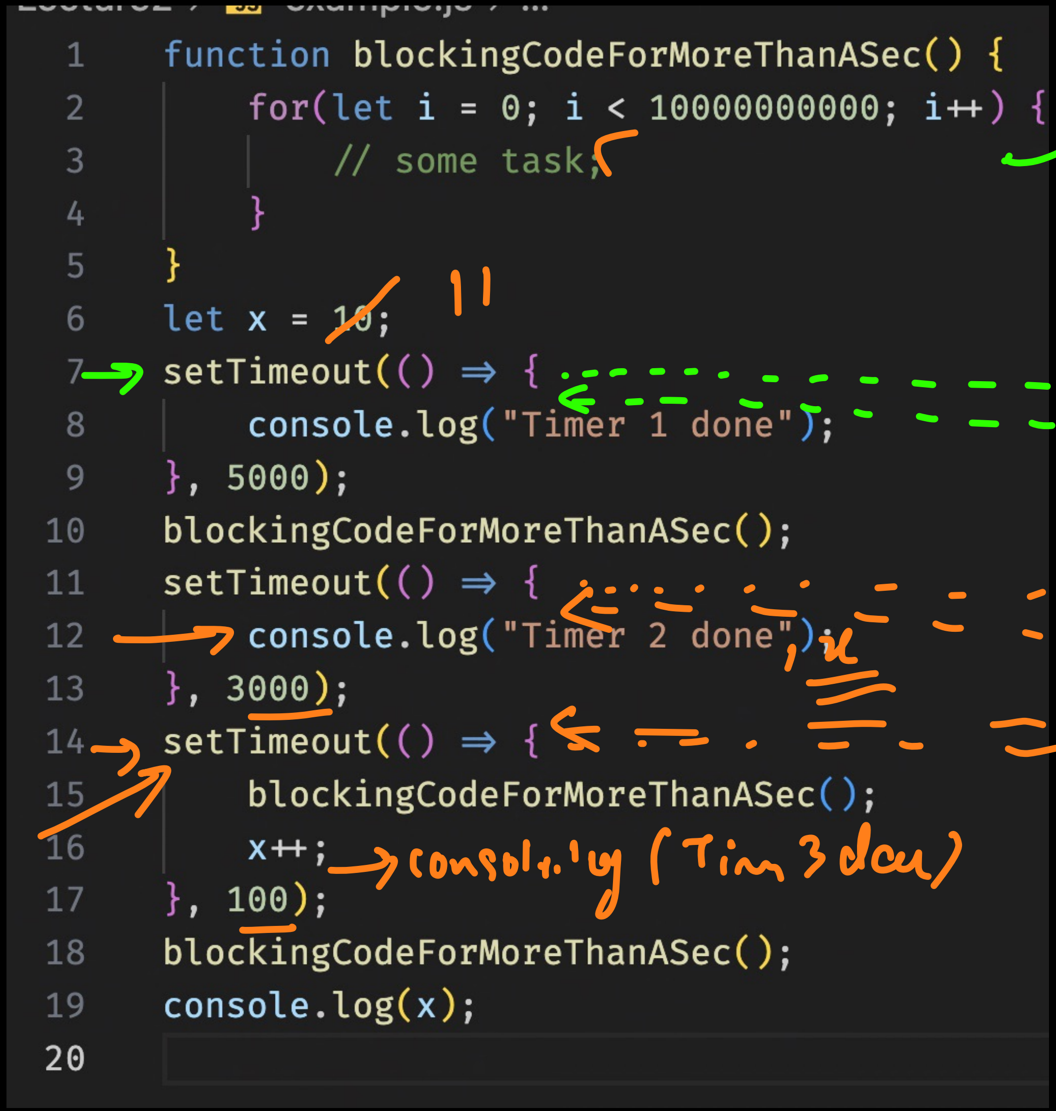
→ example to revise

→ what the issues with current async Behaviour

→ Resolve issues → Promises

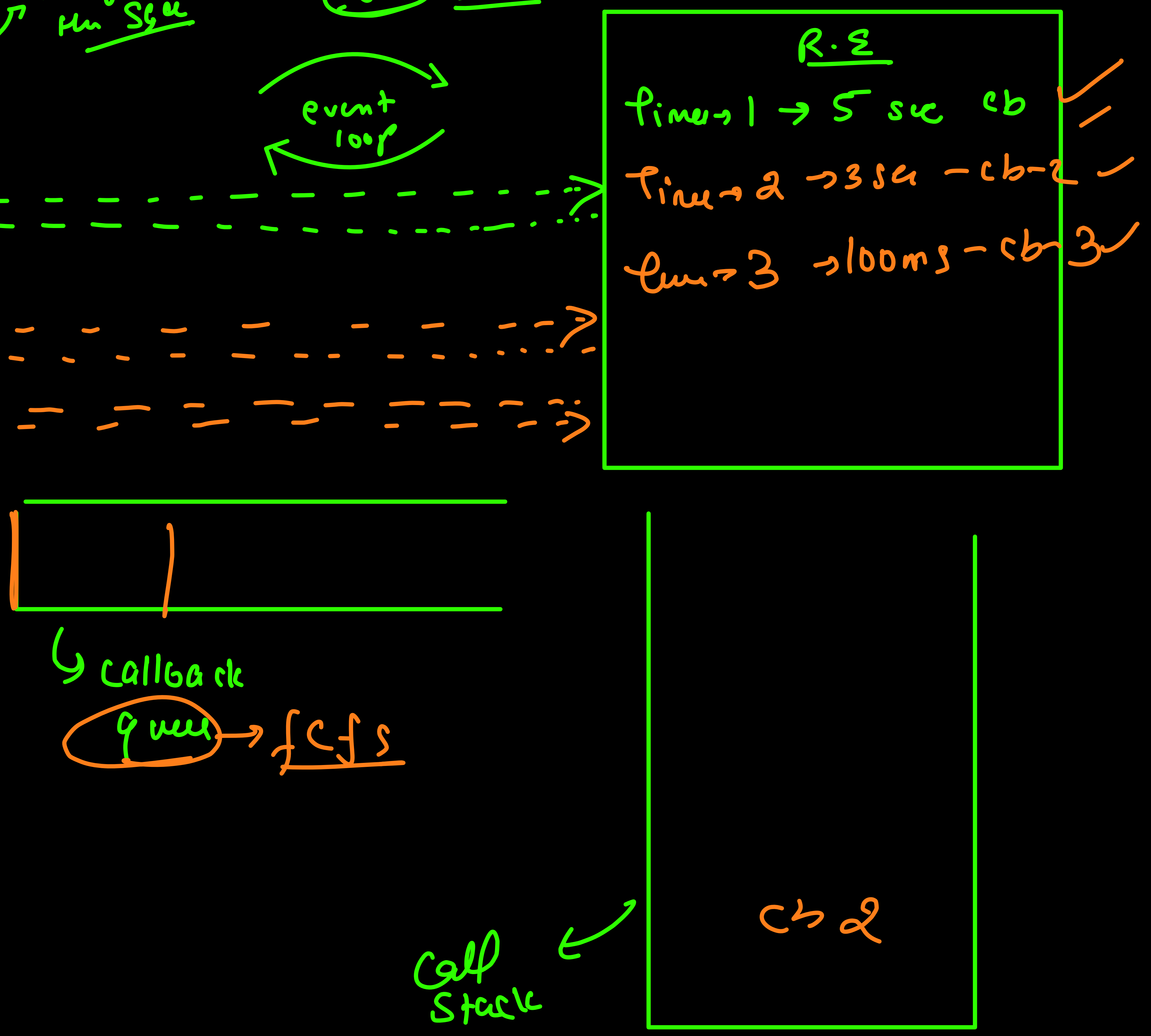


```
function blockingCodeForMoreThanASec() {
    for(let i = 0; i < 10000000000; i++) {
        // some task;
    }
}
let x = 10;
setTimeout(() => {
    console.log("Timer 1 done");
}, 5000);
blockingCodeForMoreThanASec();
setTimeout(() => {
    console.log("Timer 2 done");
}, 3000);
setTimeout(() => {
    blockingCodeForMoreThanASec();
    x++;
}, 100);
blockingCodeForMoreThanASec();
console.log(x);
```

will go more than sec → $\boxed{11sec}$ assum

~~10~~ 11

→ console.log(Timer 3 done)



printed → $\boxed{10}$

Timer 1 done
Timer 3 done
Timer 2 done 11

every async code we discussed is mainly based out of callbacks

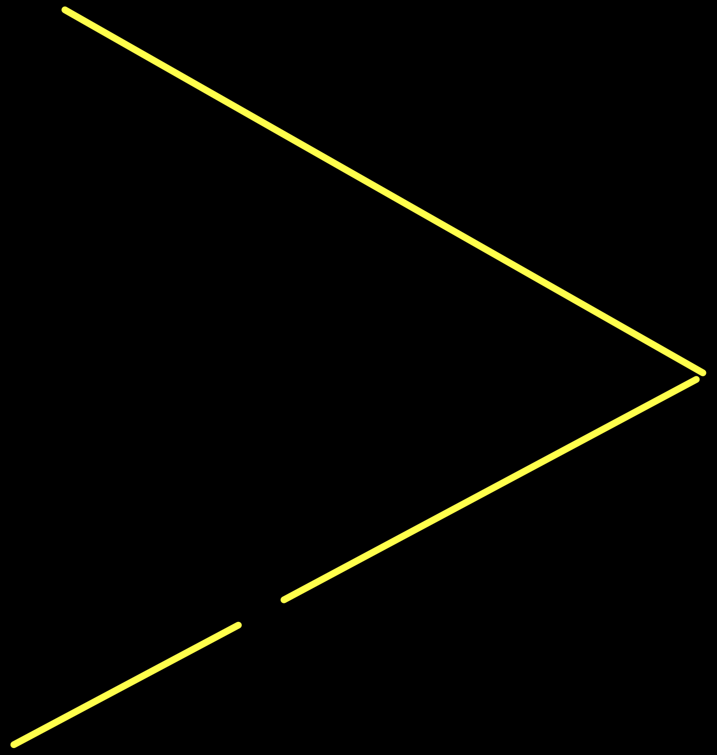
But callbacks can be a bit problematic.

Disadvantages Of Callback Based Codes

#opinion

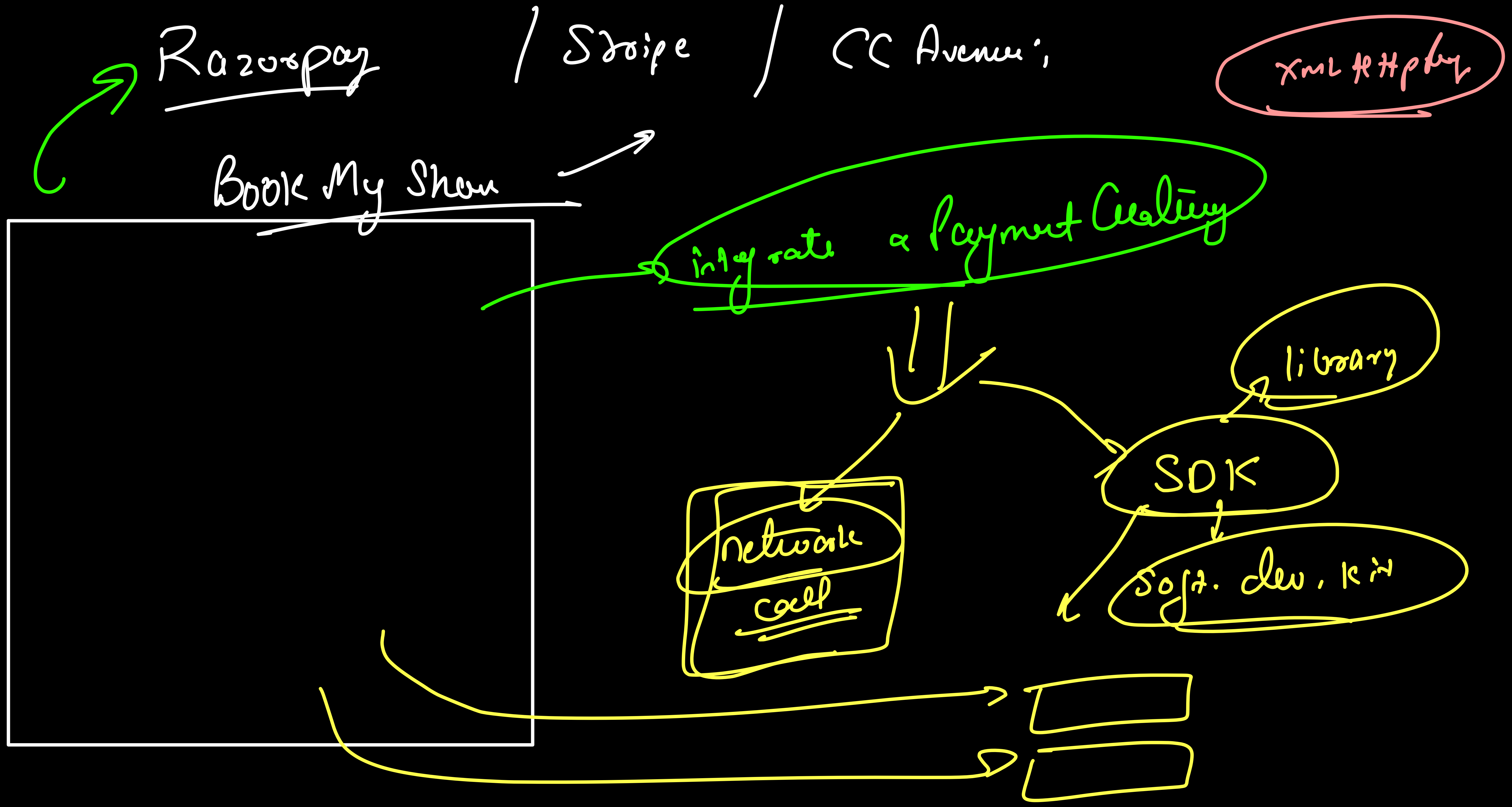
→ Callback hell → Code readability issue.

↳ callback insided a callback inside a callback



(IoC)

② Inversion Of Control : you are giving control of your code execution to somebody else.


Razorpay / Stripe / CC Avenue;
XML HttpReq
Book My Show
integrate a Payment Gateway
library
SDK
network call
soft. dev. kit

SDK of razorpay

callback based

Razorpay Checkout

```
RazorpayCheckout({ ..... },
  () => {

  })
```

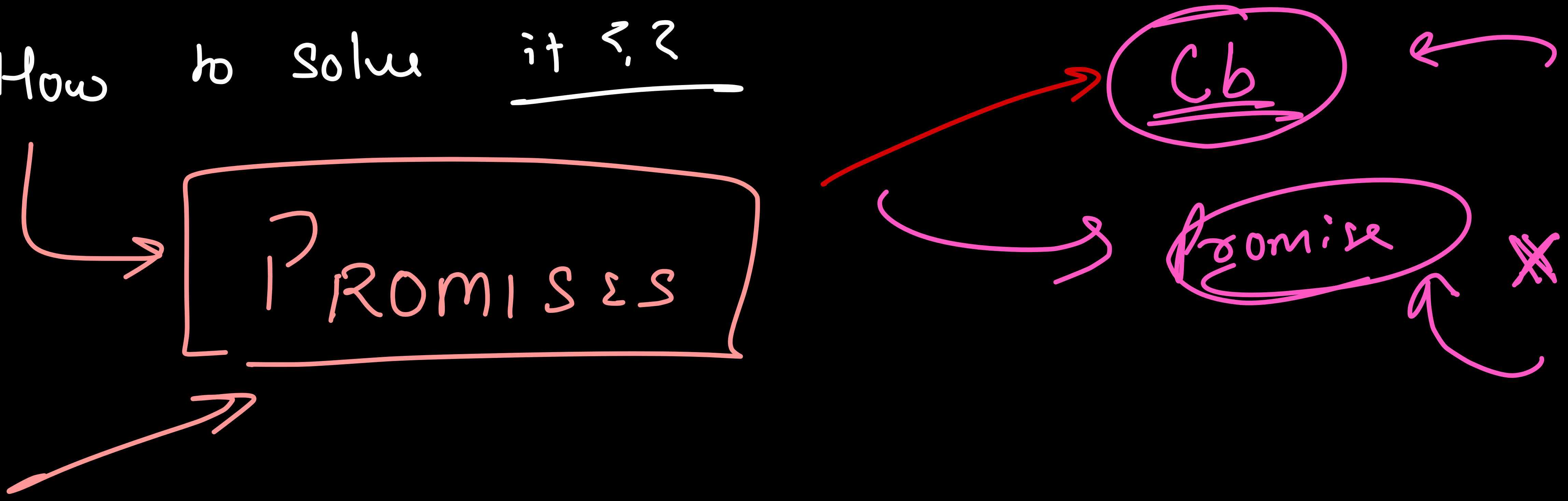
sign

SDK

```
RazorpayCheckout (credentials, checkoutCb) {
    verifies (credentials);
      eckout Cb ( );
   → checkout Cb ( );
}
```

deduct money

OK!! → How to solve it ??



Promises

#→ It will be a long road before we understand how Promises solve IoC.

→ What is a Promise in JS??

↳ It is a special JS object.

↳ It is a part of native JS lang. (i.e. it is a feature of JS not the Runtime).

↳ Promises are considered Readability enhancers as well. (it is slighty more readable than cb)

↳ Just like cb, Promises can also be used with Sync or async code.

↳ Promises are also considered placeholders for future tasks.

To understand Promises →

② ├── how to create a Promise ?

① └── how to use a Promise ?

Assume for some time, you don't need to care about how promise is created.

How to Consume a Promise ??

Properties of promise object :

Note: there are other func" also apart from these properties

→

- status → every promise object can have one of the 3 status: Pending, Fulfilled, Rejected
- value
- onfullfillment
- onRejection

State of promise :

↳ Possible states : Pending , Rejected , fulfilled

↳ the moment we create the promise immediately the Status is "Pending" always.

$$\begin{tikzcd}
 & & \text{Fulfilled} \\
\text{Pending} \arrow[loop above] \arrow[r] & {} \arrow[ur] \arrow[dr] & \text{or} \\
 & & \text{Rejected}
\end{tikzcd}$$

from pending you can go to fulfilled or rejected <u>status</u>.

once the promise is either fulfilled OR rejected state, the state cannot change again

→ When will the status change & to what it will change is programmed when promise is created. Consumer of promise donot decide when & how state changes.

→ A promise can be in forever pending state also.

Value of a promise : $\longrightarrow$ PromiseResult

↳ initially when Promise is created, the state is pending & the value property is undefined.

↳ when the state of a promise changes to fulfilled or rejected, then the value property MIGHT-CHANGE.

↳ Value of a promise cannot change without state change. i.e. if promise pending forever, value will be undefined forever.

Q:- if the value of a promise has changed once, can it change again ?? NO.

onfulfillment : [f1, f2, f3]

↳ It is an array.

↳ It holds all the funcⁿ which we want to execute once promise state goes from PENDING to FULFILLED. It has nothing to do with rejection state.

↳ Who writes these funcⁿ & registers them in the array?

↳ The consumer of the promise writes the methods & manually register / store them in this array.

↳ the array remains empty until or unless you registry / store the first func^n. That means, if state of state change doesnot control, when the array is empty.

→ when the func^ns stored in this array when it will be executed is controlled by state change.

→ How to register func^n ??

↳ we will discuss in sometime.

onRejected : $\rightarrow$ [f_1, f_2, f_3]

↳ It is everything you learned in onfulfillment just the state change we target is

PENDING $\rightarrow$ Rejected.