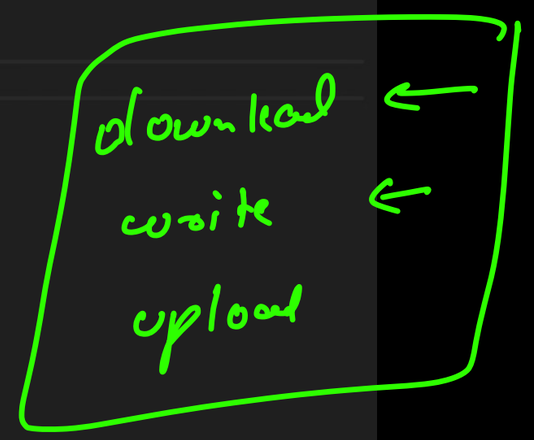
Tasks:

Write three dummy functions without using promises and only using normal callbacks,
All three functions are dummy, you dont need actual implementation.
These dummy functions are to just represent a delay.

- Write a function to download data from a url
- Write a function to save that downloaded data in a file and return the filename
- Write a function to upload the file written in previous step to a newurl

```
function downloader(url, cb) {
    // write a dummy impl using setTimeout to show a delay
}

function writeFile(data, cb) {
    // write a dummy impl using setTimeout to show a delay
}

function uploadFile(fileName, newUrl, cb) {
    // write a dummy impl using setTimeout to show a delay
}
```

- THe download should take say 4sec delay, filewrite should take 2sec delay, upload should take 3sec delay
- Write the above callbacks and use them in a manner that first download happens then writefile happens and then upload happens



Highly coupled

loosely coupled

fun^n download (——) {

writeFile ()

}

download

upload

(3)

Creation Of Promises

To create a new Promise :

new Promise ( )

- new → used with constructor func^n
- Promise → Promise constructor
- ( ) → argument → executor func^n

What is a constructor ? → it is a func^n capable of creating new objects :

what is an executor func^n

↳ this func^n contains the impl defining how the promise will fulfill or reject.

executor func^n

resolver func^n

→ new Promise ((res, rej) => {

rejector func^n

also that decides when the promise resolves

})

what is res and rej ??

res ⇒ resolver fnc. If you call res any time in the executor callback it will make the promise state go from Pendy to fulfilled. it takes one argument as well. that argument becomes value of promise after fulfillment.

```
new Promise ((res, rej) => {
  //
  res(10);
});
```

2) xy → xecutor funcⁿ. It is does the same thing as resolver just for xyet state

NOTE ⇒ when executor callback funcⁿ completes all of it's statements, the promise obj gets created -

```
1   function createAsyncPromise() {
2       return new Promise((res, rej) => {
3           // async algorithm
4           setTimeout(function timerCompleted() {
5               // lets change the state of promise
6               const value = Math.random();
7               if(value < 0.5) {
8                   // resolve the promise
9                   res(value);
10              } else {
11                  // reject the promise
12                  rej(value);
13              }
14          }, 10000);
15      });
16  }
17
```

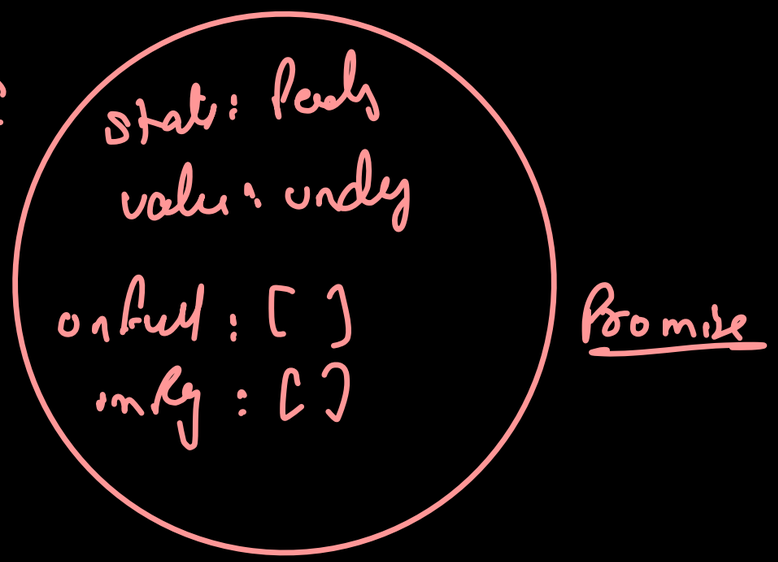
executor cb

R.E

Timer → 10sec → timerComp

timerComplete

cb queue



Promise

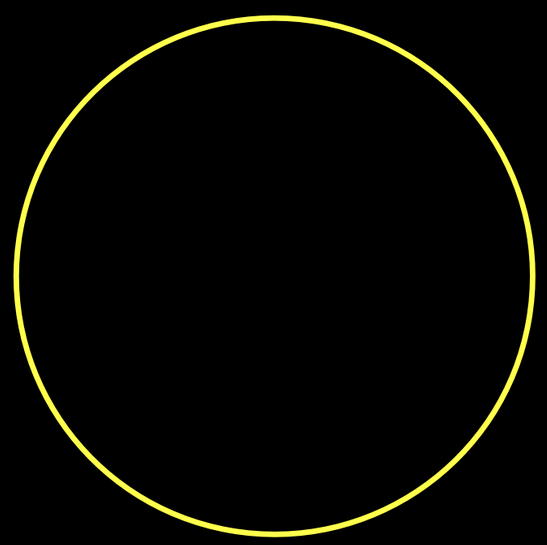
new Promise ((res, rej) => {

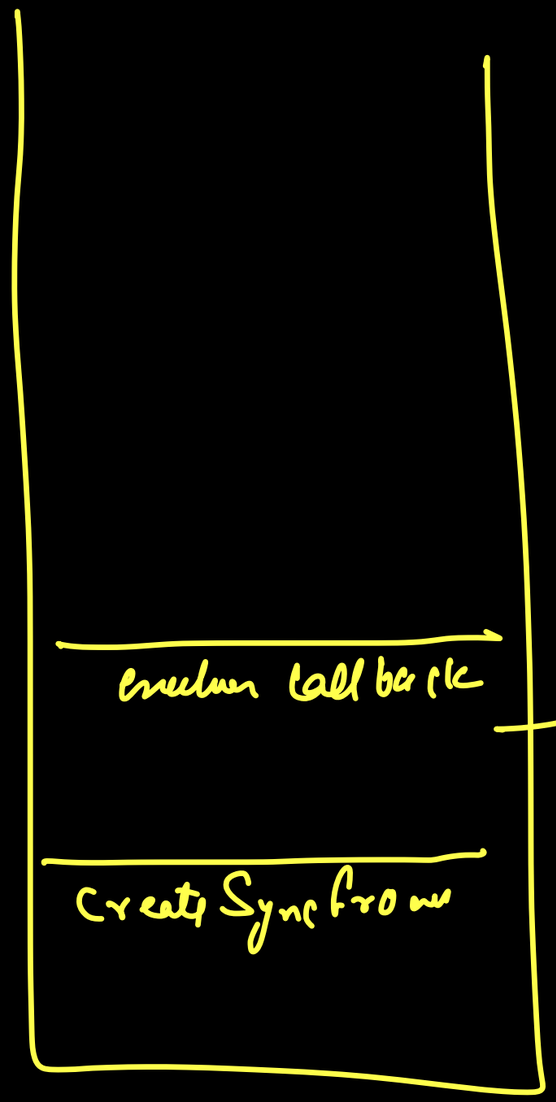
});

p.then((val) => {

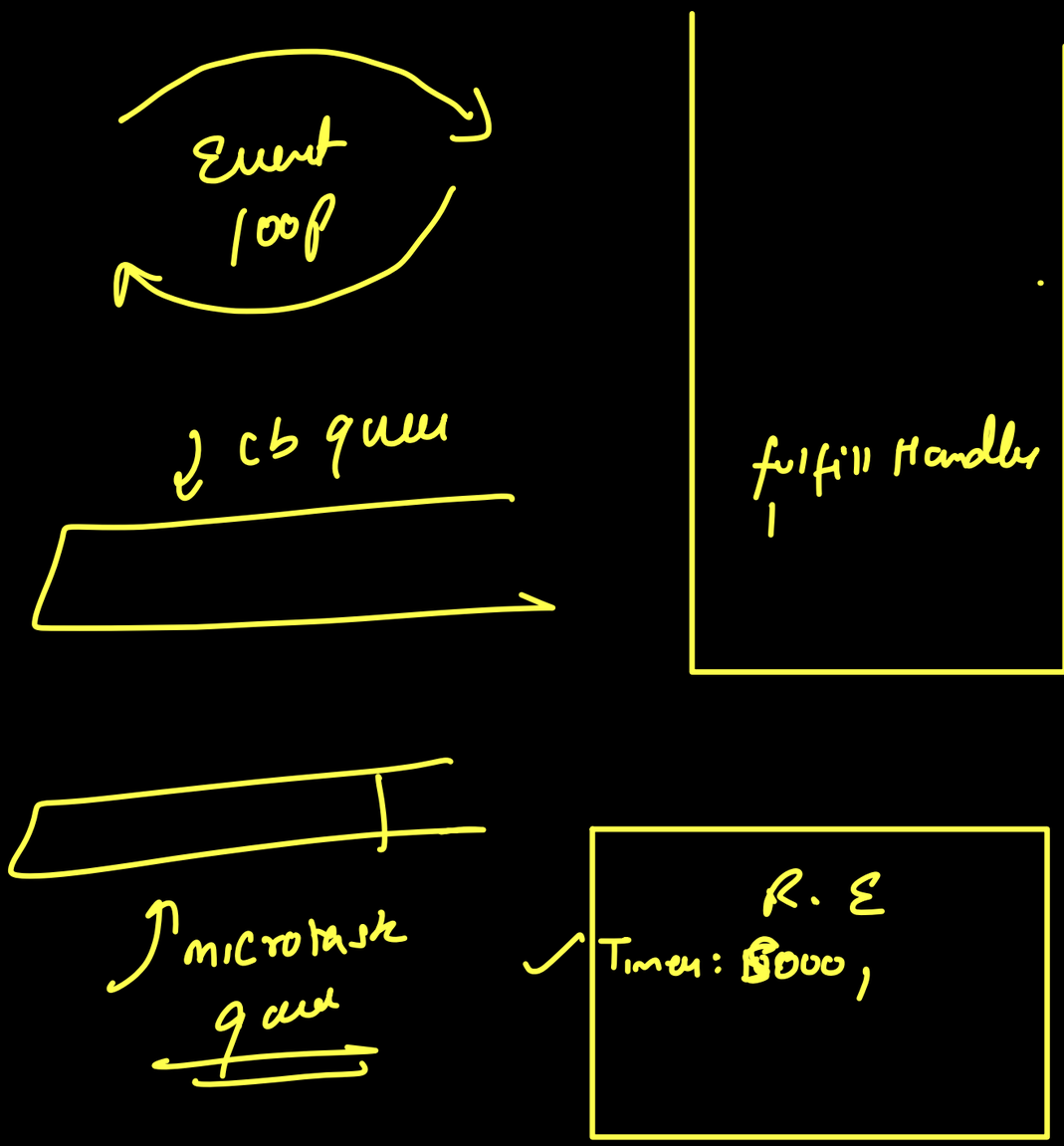
});

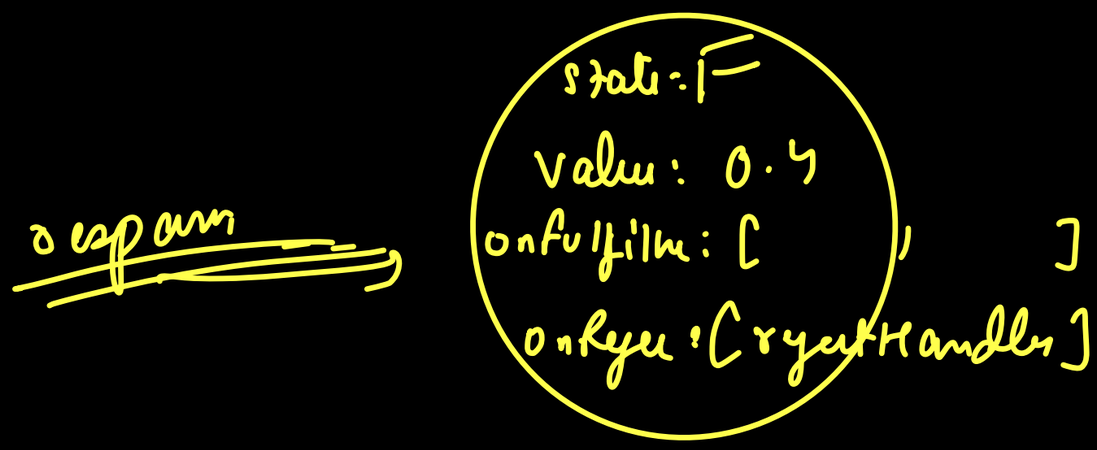
```js
function createSyncPromise() {
    return new Promise((res, rej) => {
        // async algorithm
        for(let i = 0; i < 10000000000; i++) {
            // blocking code
        }

        res(100);
    });
}

const response = createSyncPromise();
```



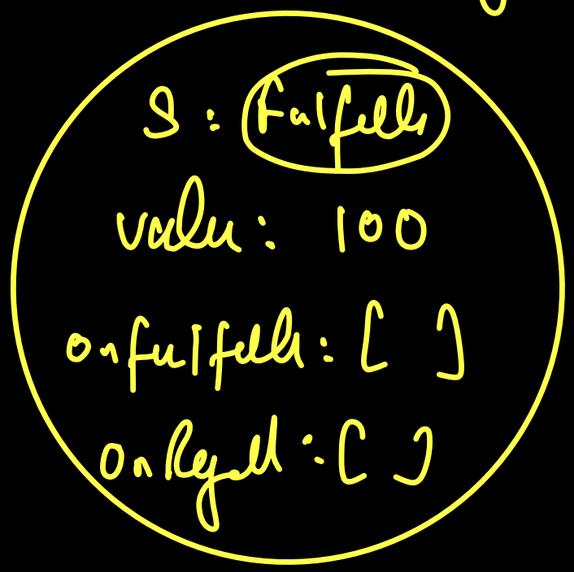


```js
function createAsyncPromise() {
    return new Promise((res, rej) => {
        // async algorithm
        setTimeout(function timerCompleted() {
            // lets change the state of promise
            const value = Math.random();
            if(value < 0.5) {
                // resolve the promise
                res(value);
            } else {
                // reject the promise
                rej(value);
            }
        }, 5000);
    });
}
const response = createAsyncPromise();
response.then(function fulfillHandler(value) {
    console.log("promise filfilled with", value);
}, function rejectHandler(value) {
    console.log("promise rejected with", value);
})
for(let i = 0; i < 10000000000; i++) {
    // blocking code
}
```



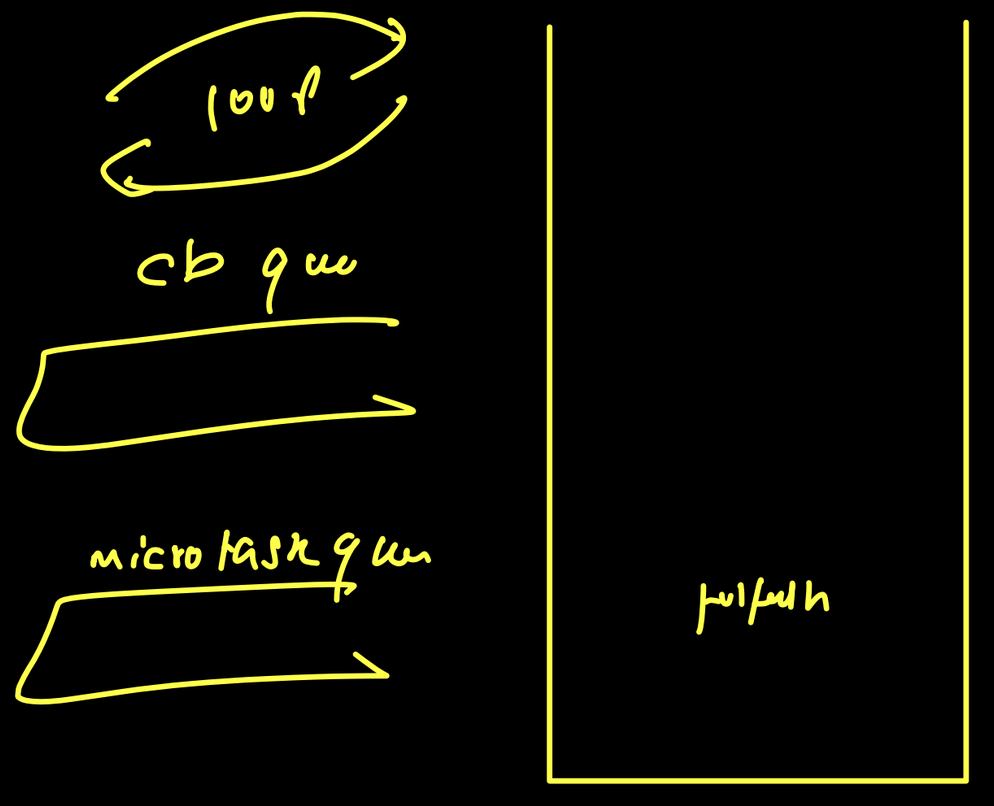


State changes before the registration handlers



p8. then (fh, rh)

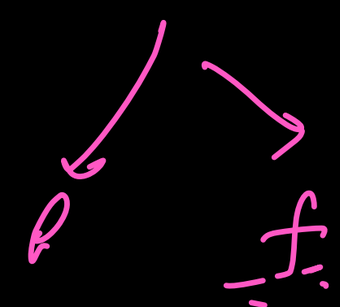
```
function createAsyncPromise() {
    return new Promise((res, rej) => {
        // some logic
        res(10);
    })
}
const response = createAsyncPromise();
// we have a fulfilled promise
for(let i = 0; i < 10000000000; i++) {
    // something
}
response.then(function fulfillHandler(value) {
    console.log("1. promise filfilled with", value);
}, function rejectHandler(value) {
    console.log("1. promise rejected with", value);
});
for(let i = 0; i < 10000000000; i++) {
    // something
}
```



response

stack : f
value : 10
onfulfill : [ ]
onreject : [ryuth]

R.E



how promise solve IoC ??

Razorpay (credentials , cb

Razorpay (credential) {
  new Promise(___)
}

F
R

response. then (fh, gf)

```
p = fs.rename("old.txt", "new.txt")
p.then(fh, rh)
```