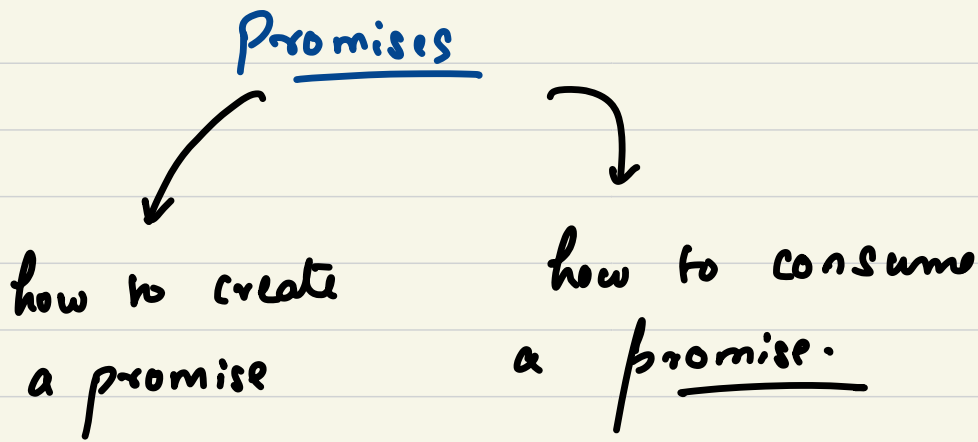
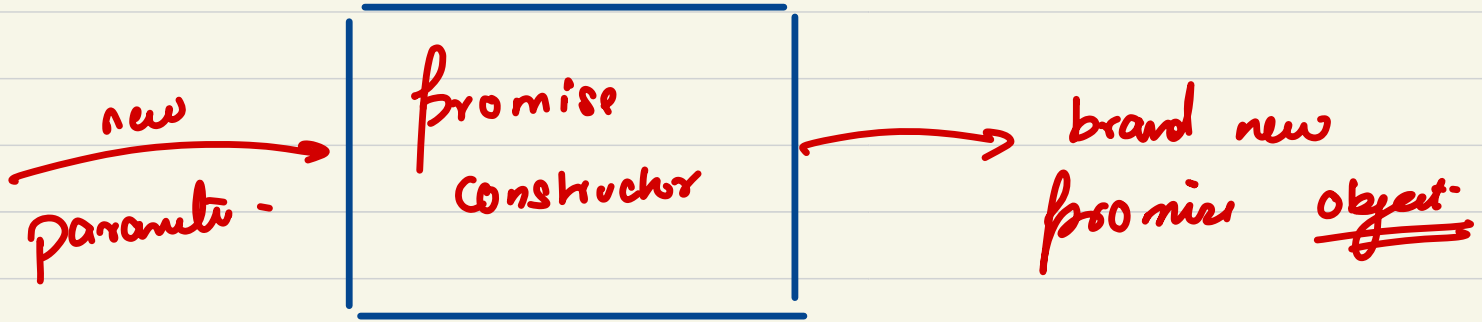
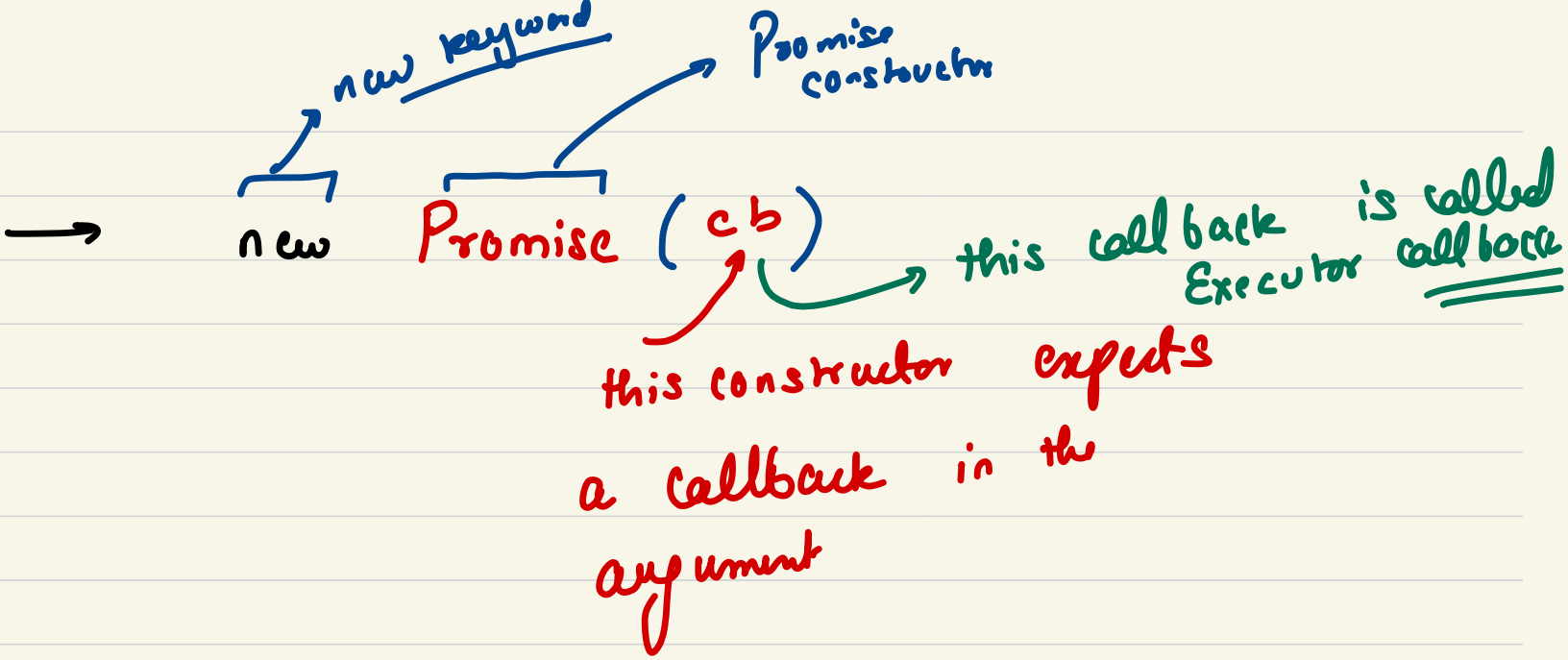

Promises
how to create a promise
how to consume a promise.

# How to create a Promise ?

↳ In JS we have a promise constructor






# why is it called executor callback ??

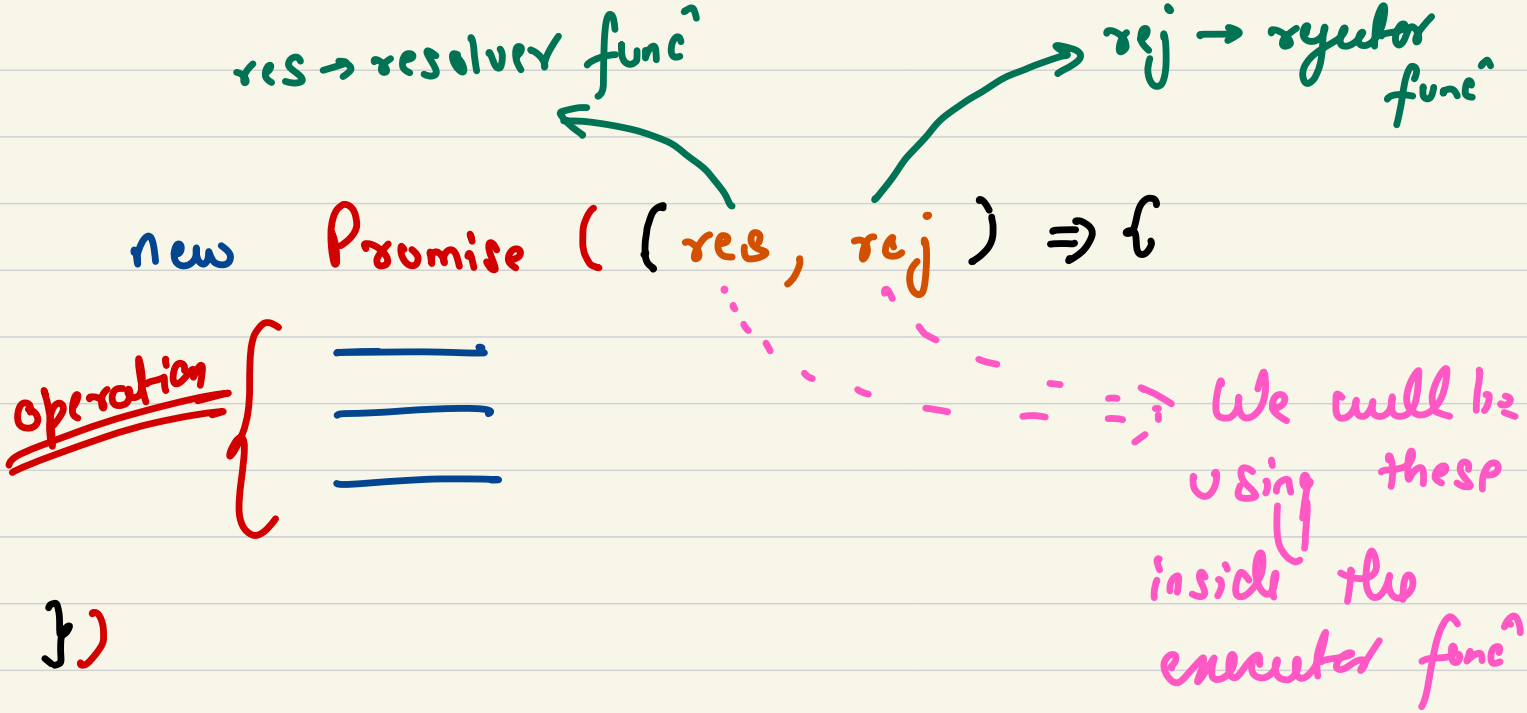
→ Because when we create a promise object at that point of time, our constructor will

execute this callback.

That means this callback is executed by the Promise constructor immediately

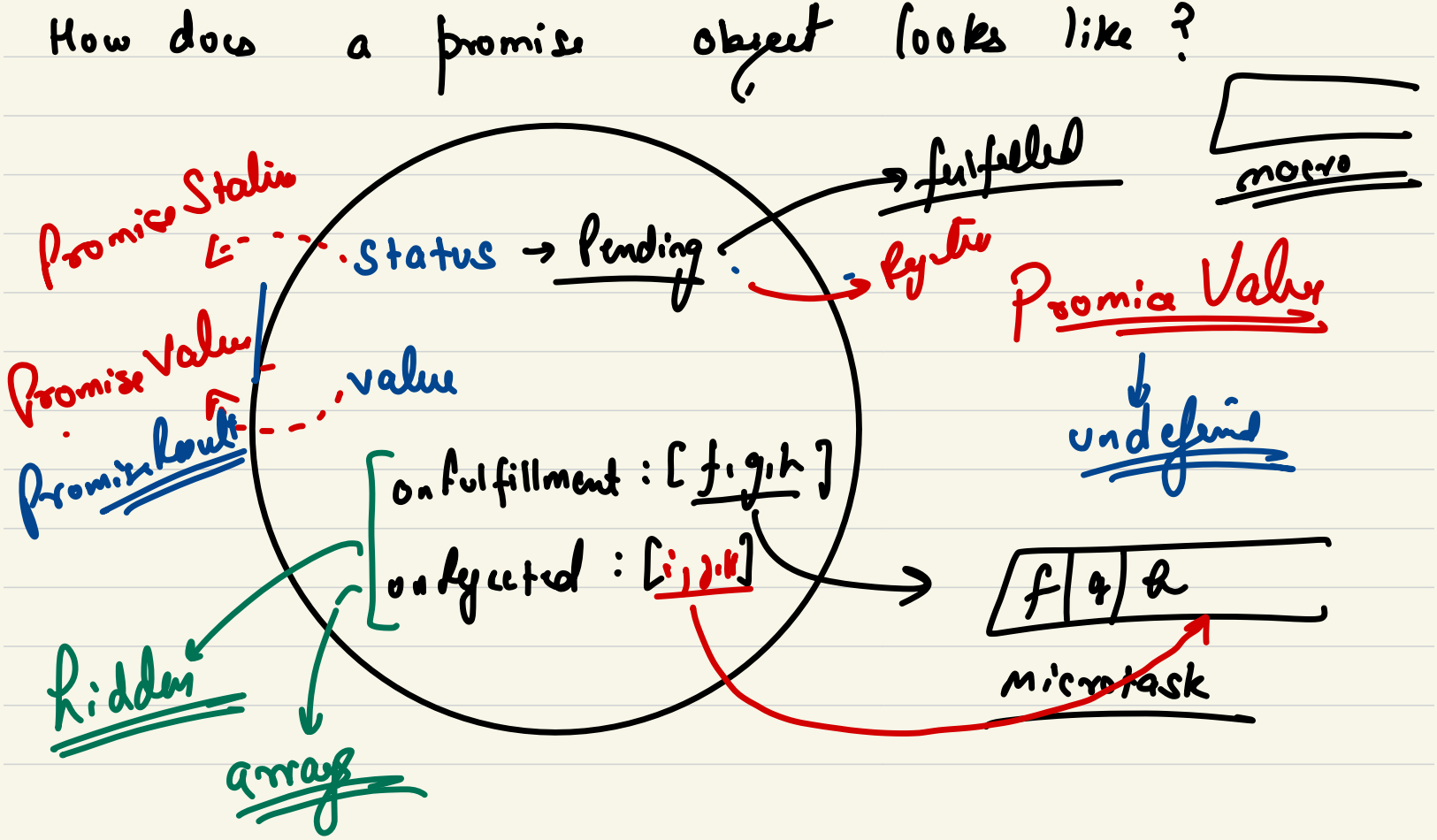
→ We are only responsible for giving the definition of the callback.

# how does this cb looks like ??




Because Promise constructor is responsible for

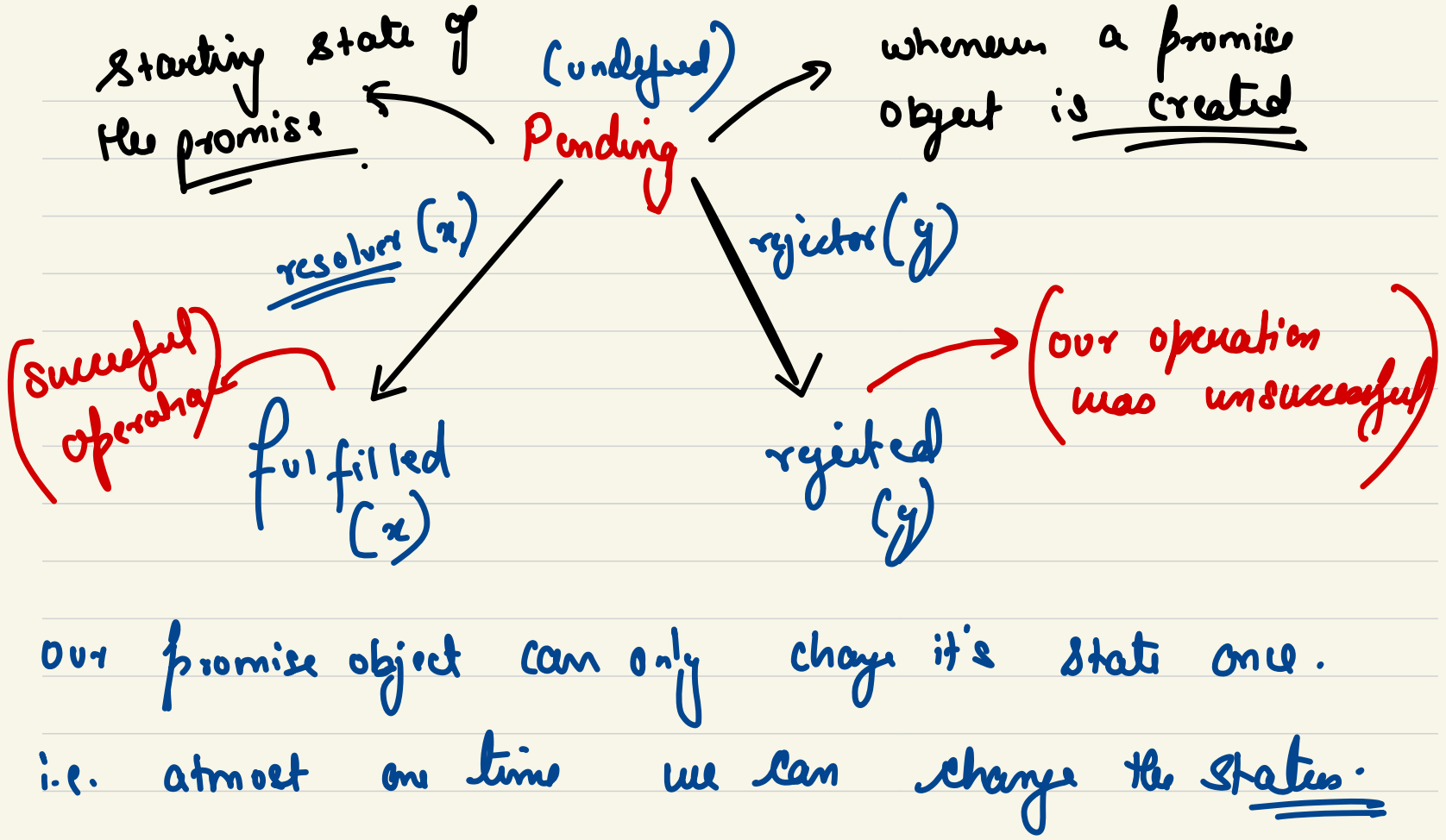
executing the executor callback, it becomes the role of promise constructor to give the definition of resolver & rejector fun^n.


How does a promise object looks like ?
Promise State
Promise Value
PromiseResult
Status → Pending
value
onfulfillment : [ f, g, h ]
onRejected : [ i, j, k ]
fulfilled
Rejected
Promise Value
undefined
macro
f
g
h
microtask
hidden
array

# Promise States

at any point of time, a promise object will be in 1 of the 3 states:

- → Pending
- → fulfilled
- → Rejected




our promise object can only change it's state once.

i.e. atmost one time we can change the states.

operation refers to the algorithm written inside the executor callback.

if inside the executor callback we called the resolver func^ then : Pending → Fulfilled

if inside the executor callback we called the rejector func^ then : Pending → Rejected.

```js
const pr = new Promise(function exec(res, rej) {
    console.log("Executor callback triggered by Promise constructor");
    setTimeout(() => {
        const randomNumber = Math.floor(Math.random()*100);
        if( randomNumber % 2 === 0 ) {
            // random number is even
            res();
        } else {
            // random number is odd
            rej();
        }
    }, 5000);
});

console.log("Created the promise object");
console.log(pr);
```

2

33

R.E

Timer

↳ 5 sec

↳ cb - line 3

Queue

Static :

Pending

rejected

event loop

cb

3

1

# onfulfillment & onRejected arrays

→ These arrays are internally managed by Promise & initially are empty.

→ We can store some func^ in both of these arrays. How ?? using the .then() / .catch() func^ available on a promise object.

[f,g,h] onfulfillment
[i,j,k] onRejected

→ these func^ remain in their respective arrays till the time promise is pending.

→ if the promise moves to folfilled state, all the func^ store in the onfolfillment arrays is queued in a new queue in the memory called as MicroTask queue & onRejected array does nothing.

if promise goes to Rejected state then all the func" in the onRejected array goes to Microtask queue & onfulfillment does nothing.

# How to consume a promise??

↳ Promises acts as a placeholder object for something that will come in future.

↳ Once the future execution is done, then we might want to execute some algorithm. based on if the future were a fail or success.

To achieve this on the promise object we have a .then function.

```
const pr = new Promise (exec);
pr.then(onfulfilled, onRejected)
```

onRejected → optional

onfulfilled, onRejected → functions

onfulfilled → this will go to the onfulfillment array

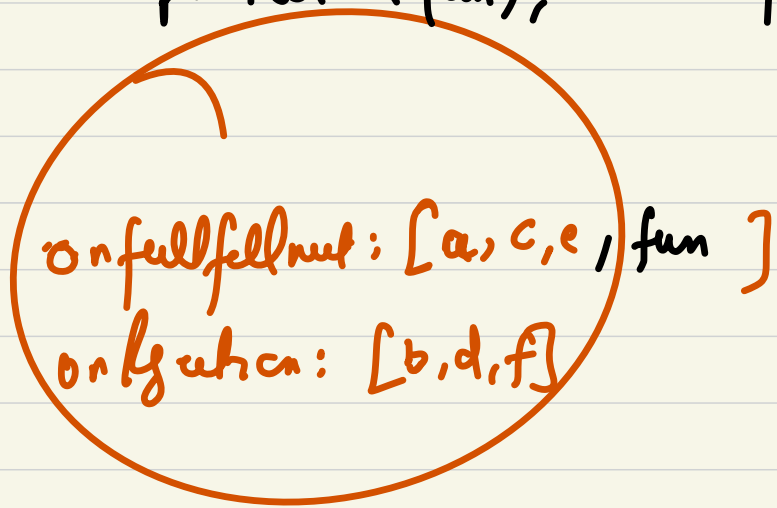
onRejected → this will go to the onRejection array

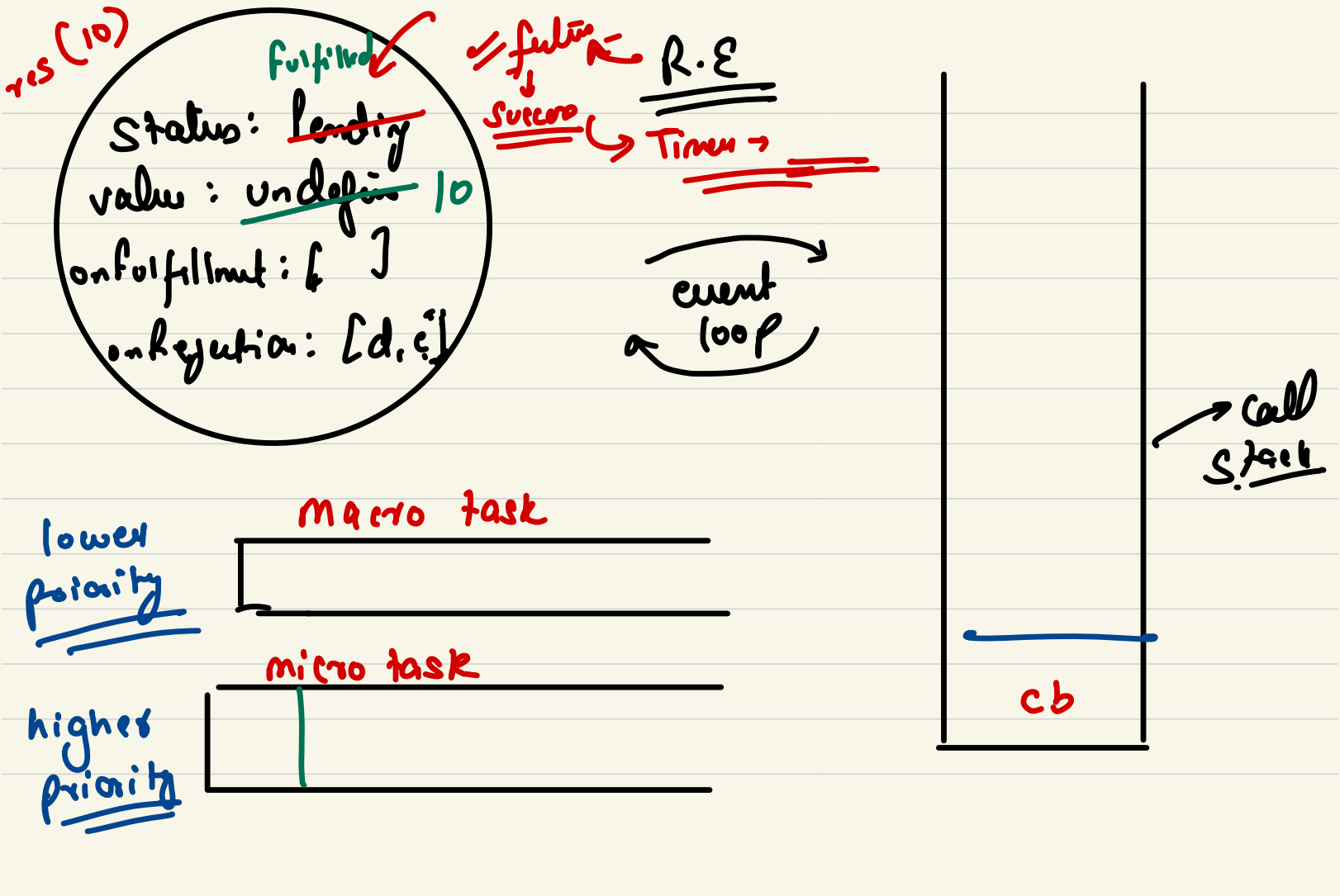
→ pr. then (a, b);

pr. then (c, d);

→ pr. then (e, f);

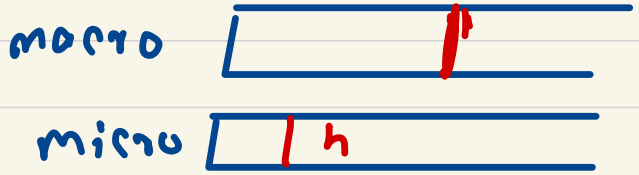
pr. then (fun); → if we just pass a single func it will go for onfulfilled.

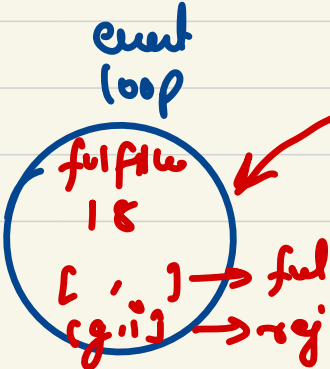


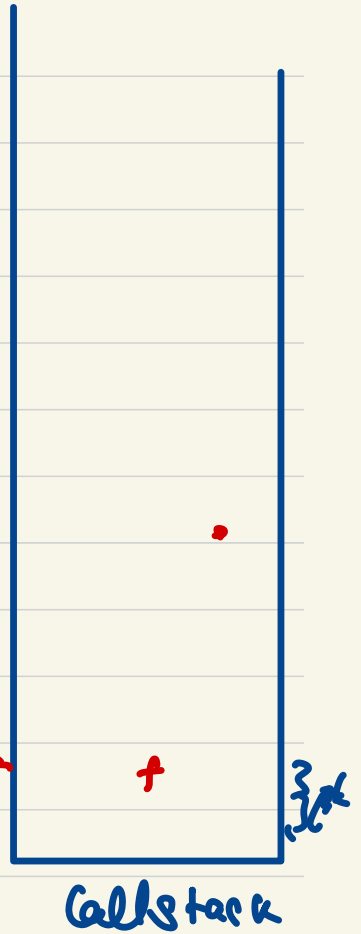

res (10)
fulfilled
Status: pending
value: undefined 10
onfulfillmt: [ ]
onRejection: [d, c]
fulfilled
success
R.E
Timer →
event loop
call Stack
cb
macro task
lower priority
micro task
higher priority

```
1  console.log("start");
2  setTimeout(function timerCB() {
3      console.log("timer 1 done");
4  }, 1000); // timer of 1 sec
5  const pr = new Promise(function exec(res, rej) {
6      console.log("Executor callback triggered by Promise constructor");
7      setTimeout(function prCB() {
8          const randomNumber = Math.floor(Math.random()*100);
9          console.log(randomNumber);
10         if( randomNumber % 2 == 0 ) {
11             // random number is even
12             res(randomNumber);
13         } else {
14             // random number is odd
15             rej(randomNumber);
16         }
17     }, 2000);
18 });
19 pr.then(function f() {console.log("executing f") }, function g(){ console.log("executing g") });
20 pr.then(function h() { console.log("execution h") }, function i() { console.log("executing i") });
21 for(let i = 0; i < 10000000000; i++) {}
22 for(let i = 0; i < 10000000000; i++) {}
23 console.log("end");
```

R.E → Timer → 1sec →
Timer → 2sec →



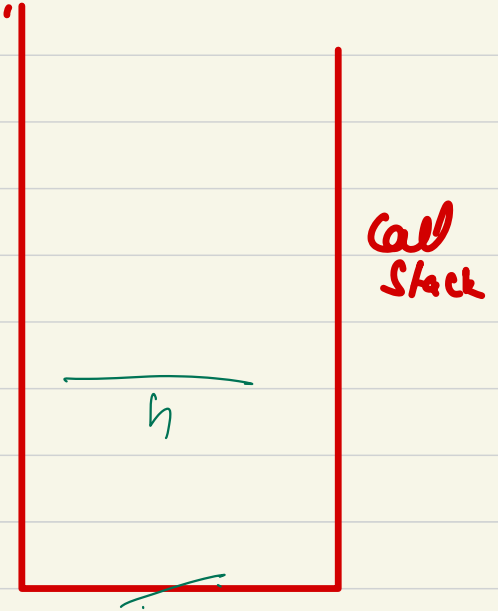



```
const p1 = new Promise((res, rej) => {
    console.log("Executor callback triggered by Promise constructor: p1");
    setTimeout(() => {
        console.log("Timer of p1 done");
        res(100);
    }, 500);
});
p1.then(function a() {console.log("a")}, function b() {console.log("b")} );
setTimeout(function timerCB() {
    console.log("timer 1 done");
}, 3000); // timer of 1 sec
const pr = new Promise(function exec(res, rej) {
    console.log("Executor callback triggered by Promise constructor");
    setTimeout(function prCB() {
        const randomNumber = Math.floor(Math.random()*100);
        console.log(randomNumber);
        if( randomNumber % 2 == 0 ) {
            // random number is even
            res(randomNumber);
        } else {
            // random number is odd
            rej(randomNumber);
        }
    }, 4000);
});
pr.then(function f() {console.log("executing f") }, function g(){ console.log("executing g") });
pr.then(function h() { console.log("execution h") }, function i() { console.log("executing i") });
for(let i = 0; i < 10000000000; i++) {}
```

1 cb 3 4 5 2 6 7 8

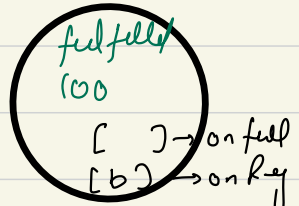
← only registration

18




R.E → T1 → 500 ÷
↳ T2 → 3 sec →
T3 → 4 sec →

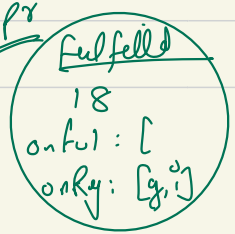
event loop

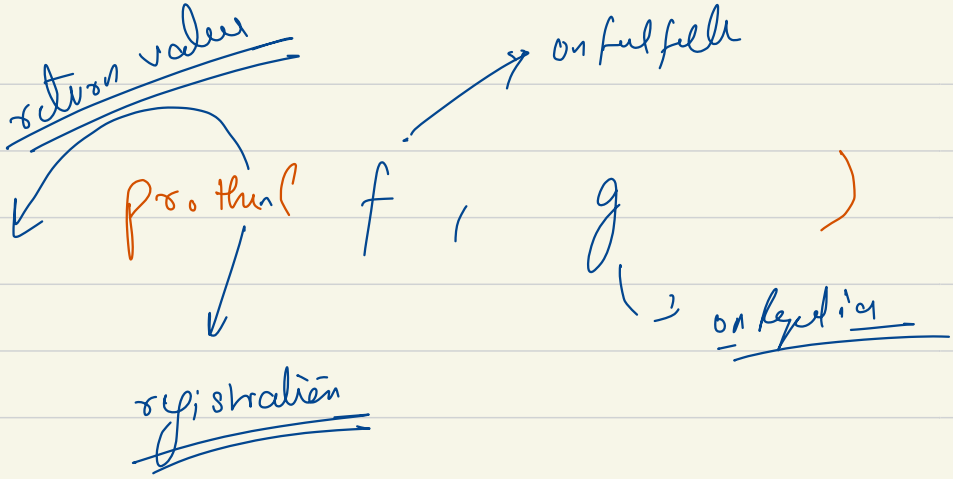



macro

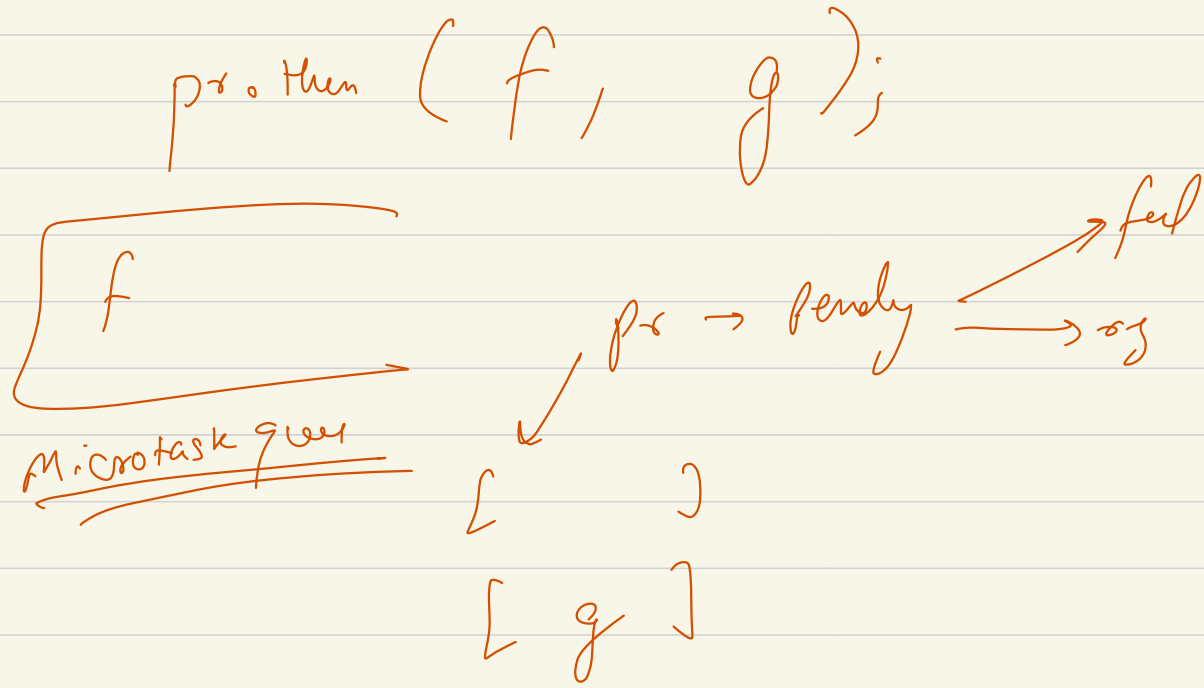
micro


pr




return value
on fulfill
pro.then( f , g )
on rejection
registration

- catch ( )


pr.then ( f , g );
f
Microtask queue
pr → Pendy
fal
sg
[ ]
[ g ]