another new promise obj

promise obj

Promises

new Promise ((res, rej) => {
setTimeout (() => res(10), 10000)

promise

.then (— , —)

.then (— , —)

.then (— , —)

new promise obj

3)

another new promise obj

.then chain

```
pr.then(r1, s1)
pr.then(r2, s2)
pr.then(r3, s3)
```

⇓

Here we are registering [r1, r2, r3] to the onfulfilled array of pr object & [s1 s2 s3] in the onRejected array of pr.

```
pr
.then(r1, s1)
.then(r2, s2)
.then(r3, s3)
```

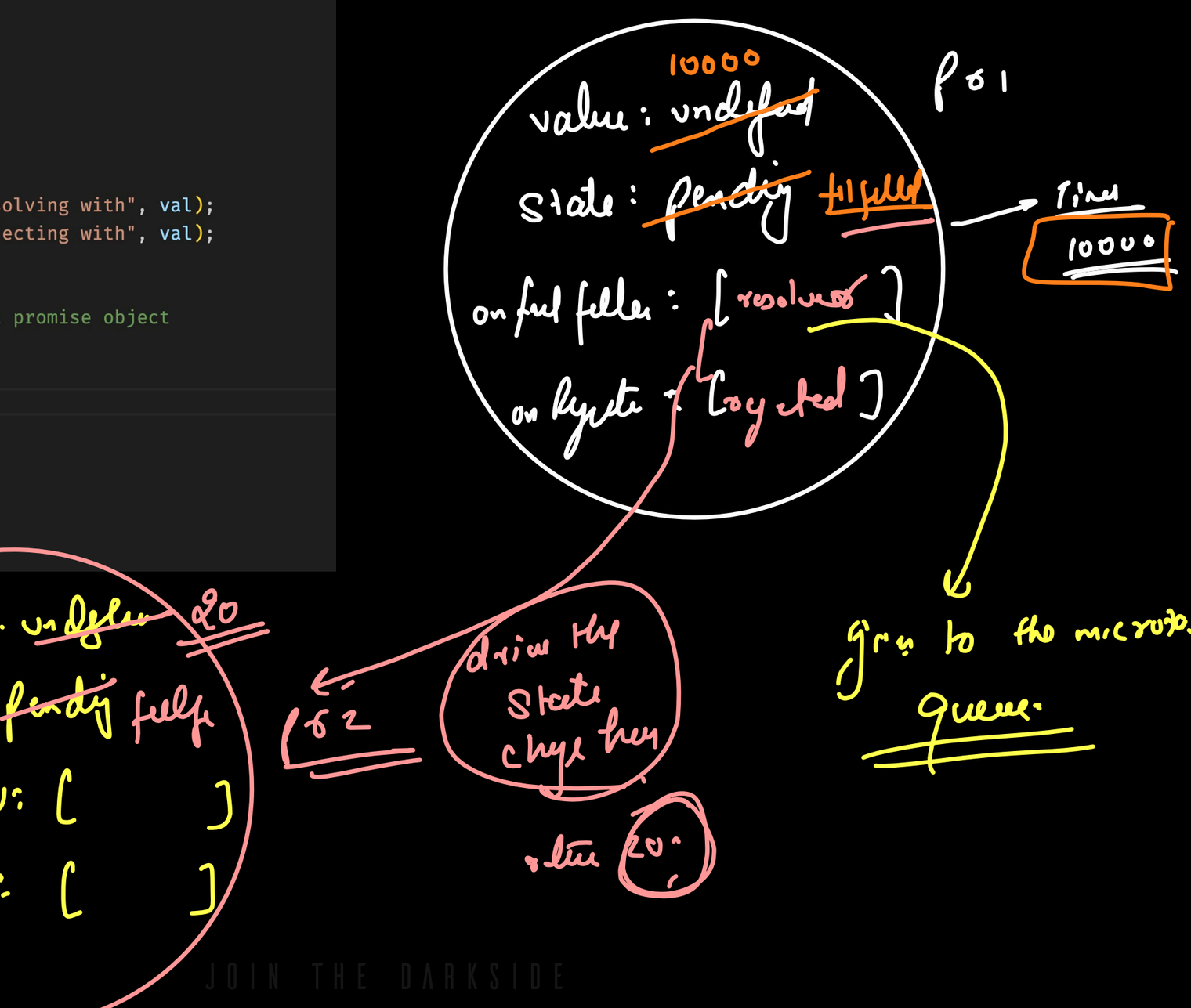
Here we register r1 to pr's onfulfilled & s1 to pr's onRejected. Then the promise returned by first pr.then gets r2 & s2

```js
function createPromise(time) {
    return new Promise((res, rej) => {
        setTimeout(() => {
            res(time);
        }, time);
    });
}

const resolver = (val) => console.log("Resolving with", val);
const rejector = (val) => console.log("Rejecting with", val);


const pr1 = createPromise(10000); // first promise object
console.log("PR1", pr1);

const pr2 = pr1.then(resolver, rejector);
console.log("PR2", pr2);
```



state : pendij

value : undefined

onfulfilled [f1, f2, f3]

onRejected [r1, r2, r3]

pr1

a)

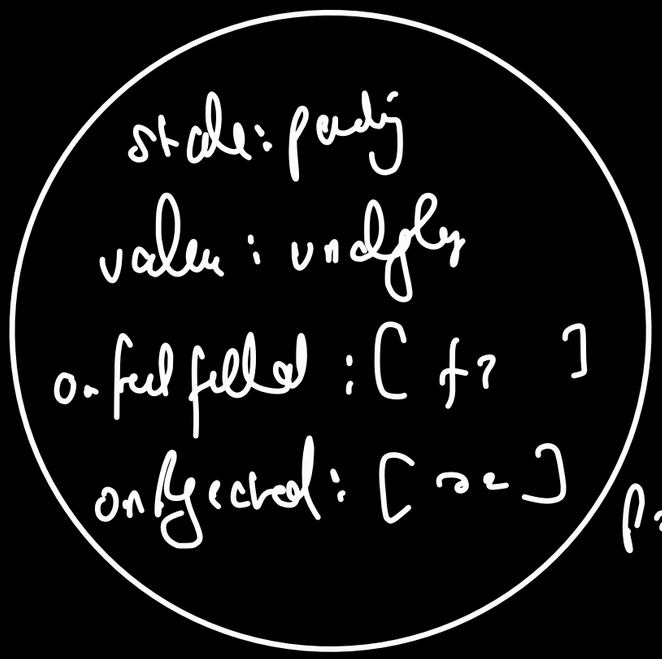
pr2 ← pr1.then(f1, r1)

pr3 ← pr1. then (f2, r2)

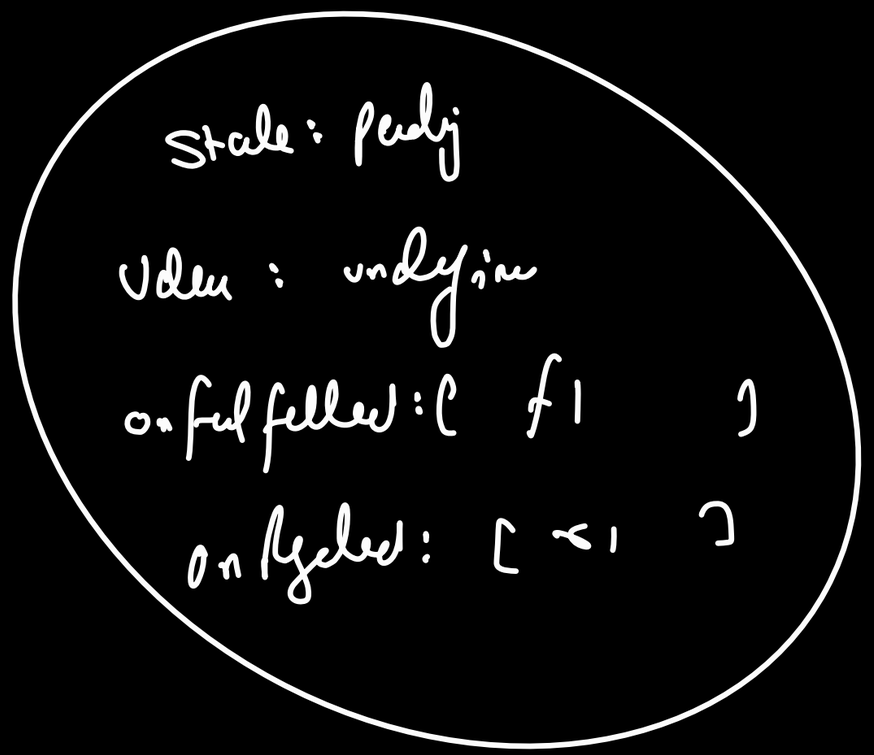
pr4 ← pr1. the (f3, r3)

b)

pr1. then (f1, r1)
. then (f2, r2)
. then (f3, r3)



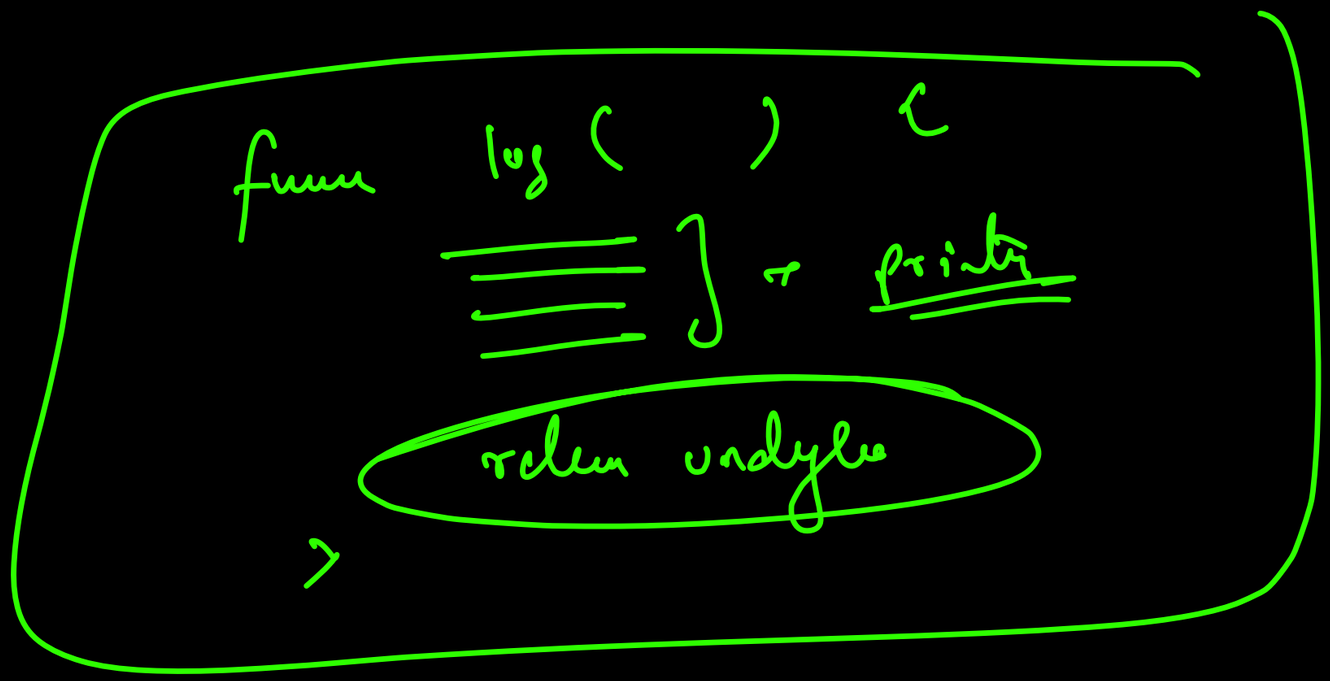
p2



p1

. then (—— , ——)

↳ optional




```
() => console.log(—)
```

```
console.log()
```

when can a promise Reject ??

↳ manually

↳ if your code throws an exception

p.then(f1, r1)

pr1

rejects

do u have rej handler

NO

do u have .a .catch

NO

propagate from pr1

YES

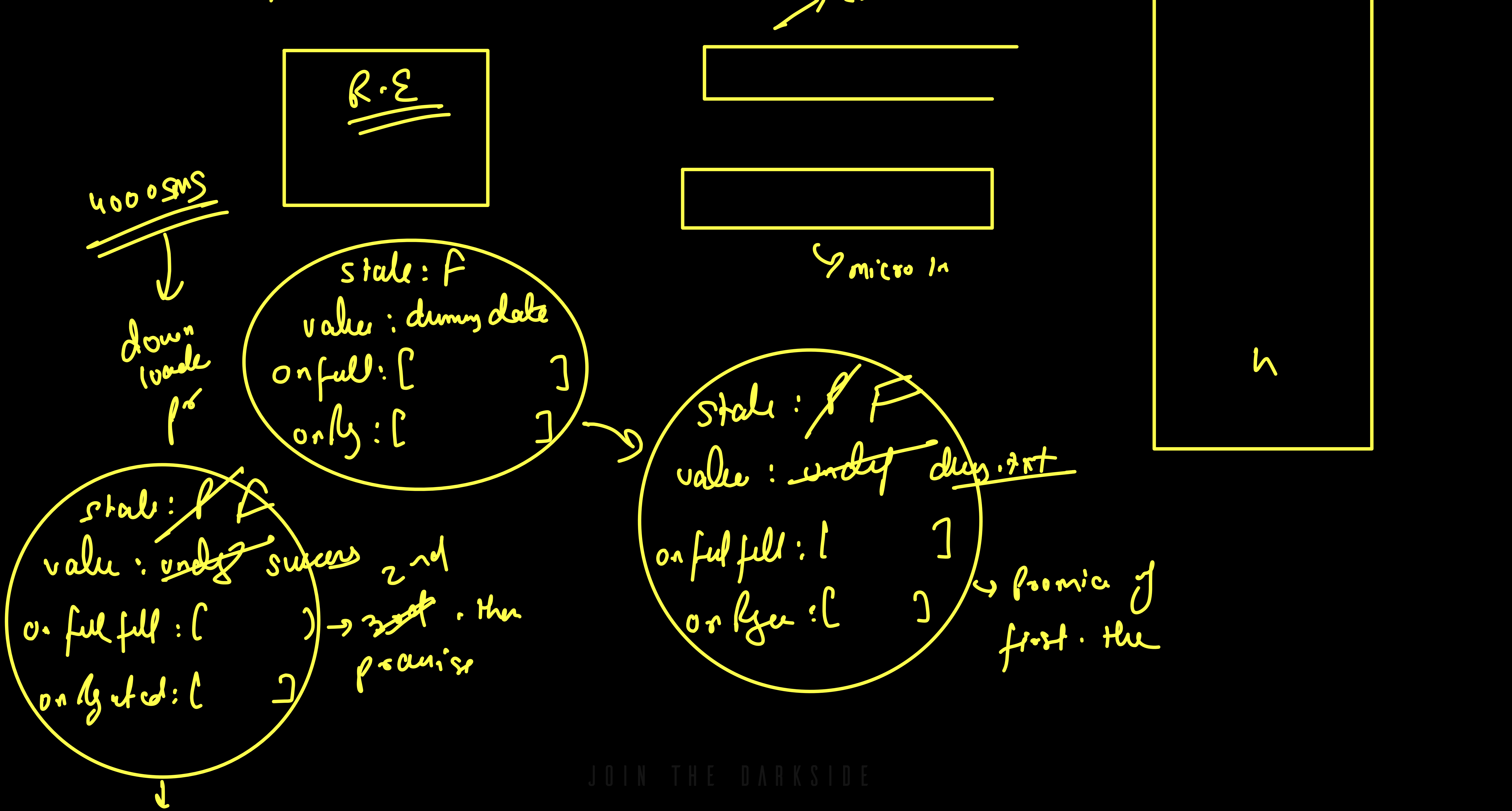
pr2 → fulfills

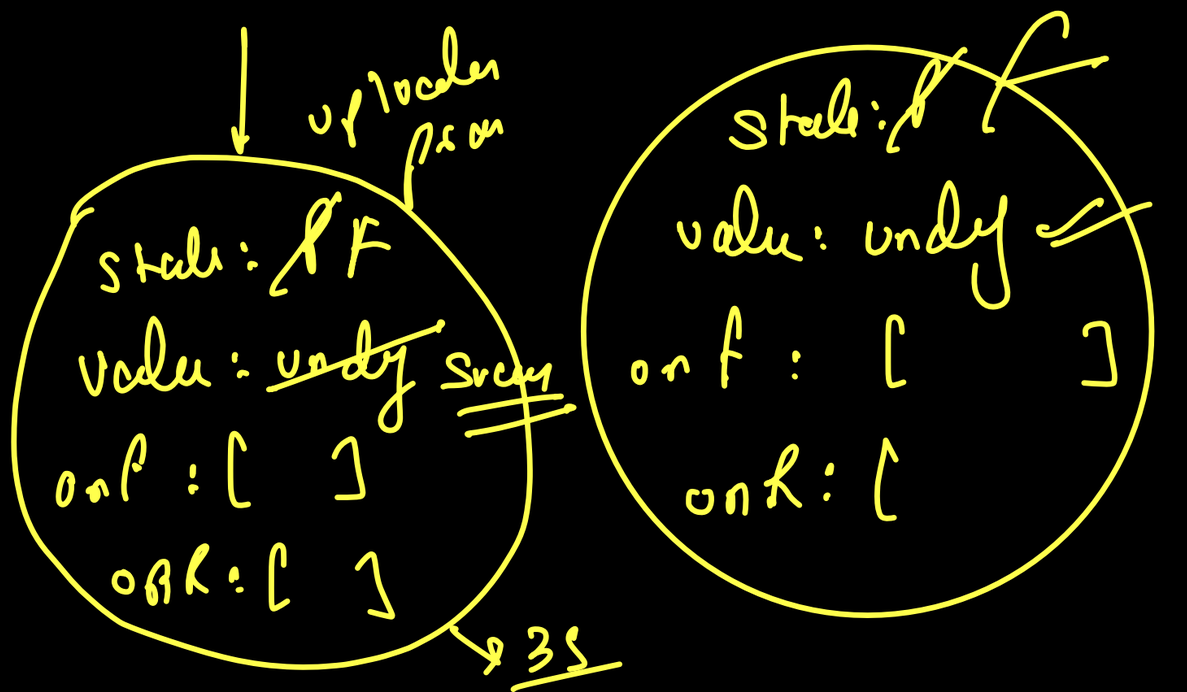
YES

handle

To handle case of promise chaining when rej handler is missing we can use .catch fun()

```javascript
downloader("www.google.com")
    .then(downloadedData => {
        return writeFile(downloadedData).then()
    })
    .then(fileName => {
        console.log("Started uplaoding from .then");
        return uploadFile(fileName, "www.x.com")
    })
    .then(value => console.log(value));
```





- returns a value: `p` gets fulfilled with the returned value as its value.
- doesn't return anything: `p` gets fulfilled with `undefined` as its value.
- throws an error: `p` gets rejected with the thrown error as its value.
- returns an already fulfilled promise: `p` gets fulfilled with that promise's value as its value.
- returns an already rejected promise: `p` gets rejected with that promise's value as its value.
- returns another pending promise: `p` is pending and becomes fulfilled/rejected with that promise's value as its value immediately after that promise becomes fulfilled/rejected.

for cb in .then