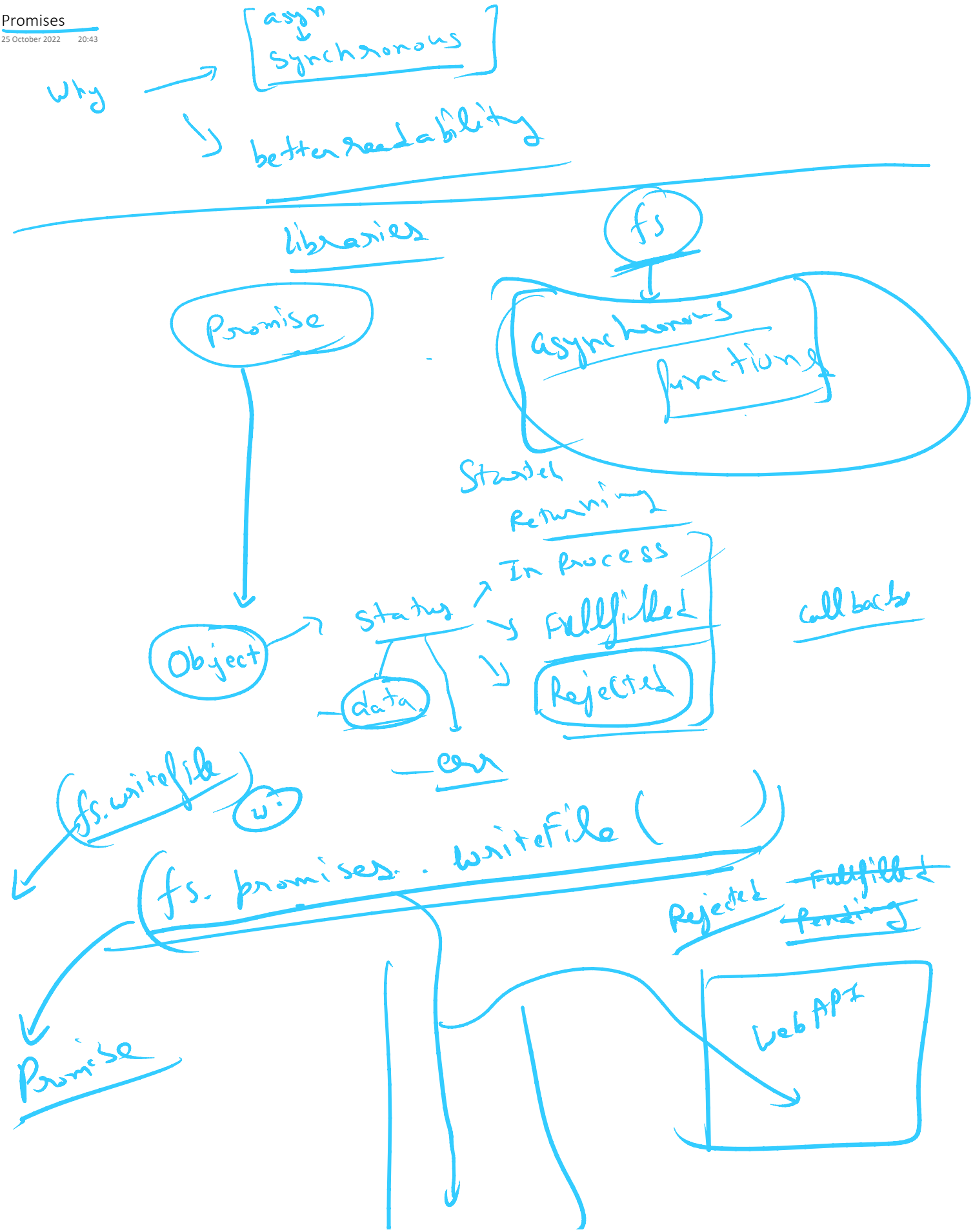
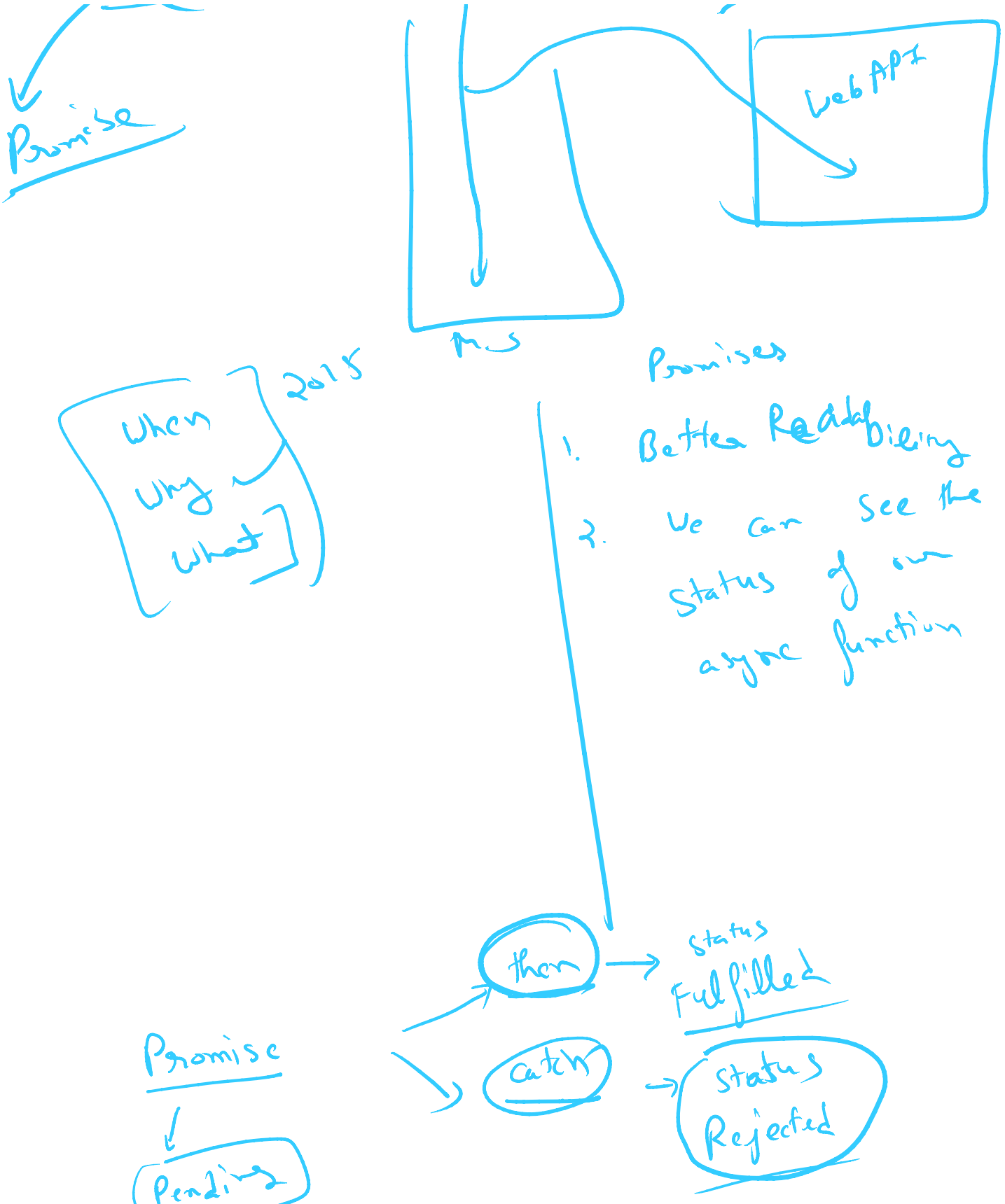
Promises

25 October 2022 20:43




Promise
Web API
JS
When
Why
What
2015
Promises
1. Better Readability
2. We can see the status of our async function
then
status
Fulfilled
Promise
catch
Status
Rejected
Pending
Promise . then ((data) => {
on (data)

Promise . then (
console.log (data)
}) . catch (err (1) => {
console.log (err)
})

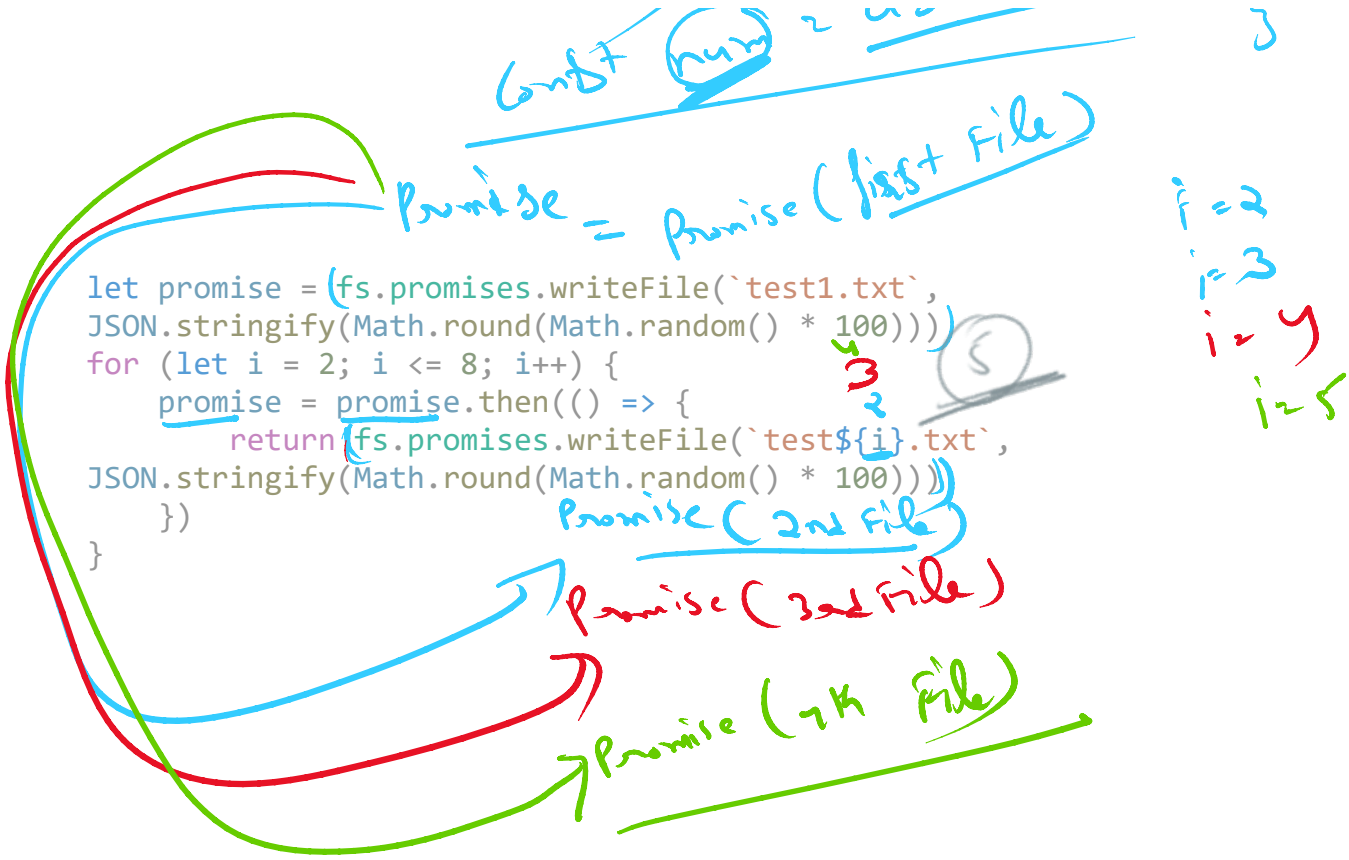
Chaining of Promises

Promise1 . then (() => {
return Promise2
}) . then (( ) => {

Promise1 . then (( ) => {
return Promise2
}) . then (( ) => {
})

const num = abc()

function abc()
{
return 2
}



There are few asynchronous JS methods / functions which get executed in the web API.

To wait for the completion of these functions callbacks were introduced first.

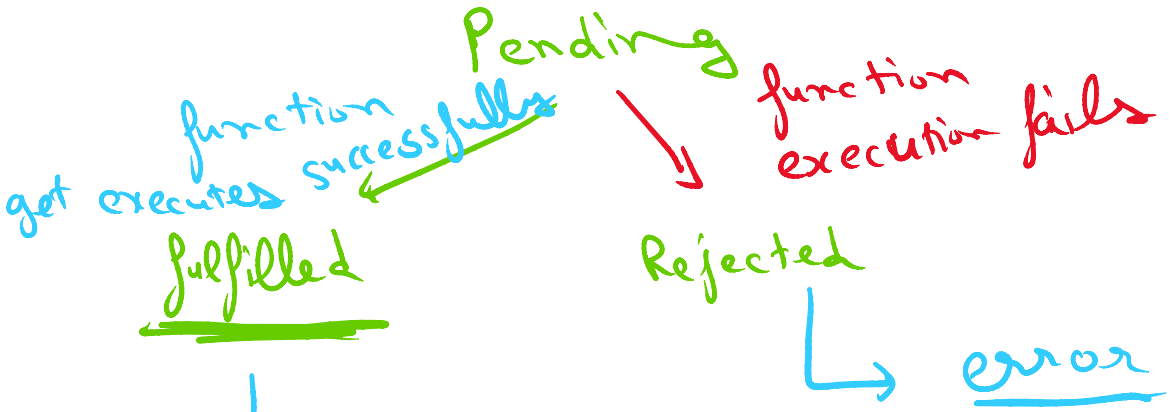
- ① callback hell → nesting of callback
  Readability & complexity of the code was very high.
- ② We had no way to check for the status of asynchronous function

2015 → Promises were introduced to solve this problem.

After 2015 → JS libraries e.g fs which had async functions with callback functionality were slowly changed to Promises concept

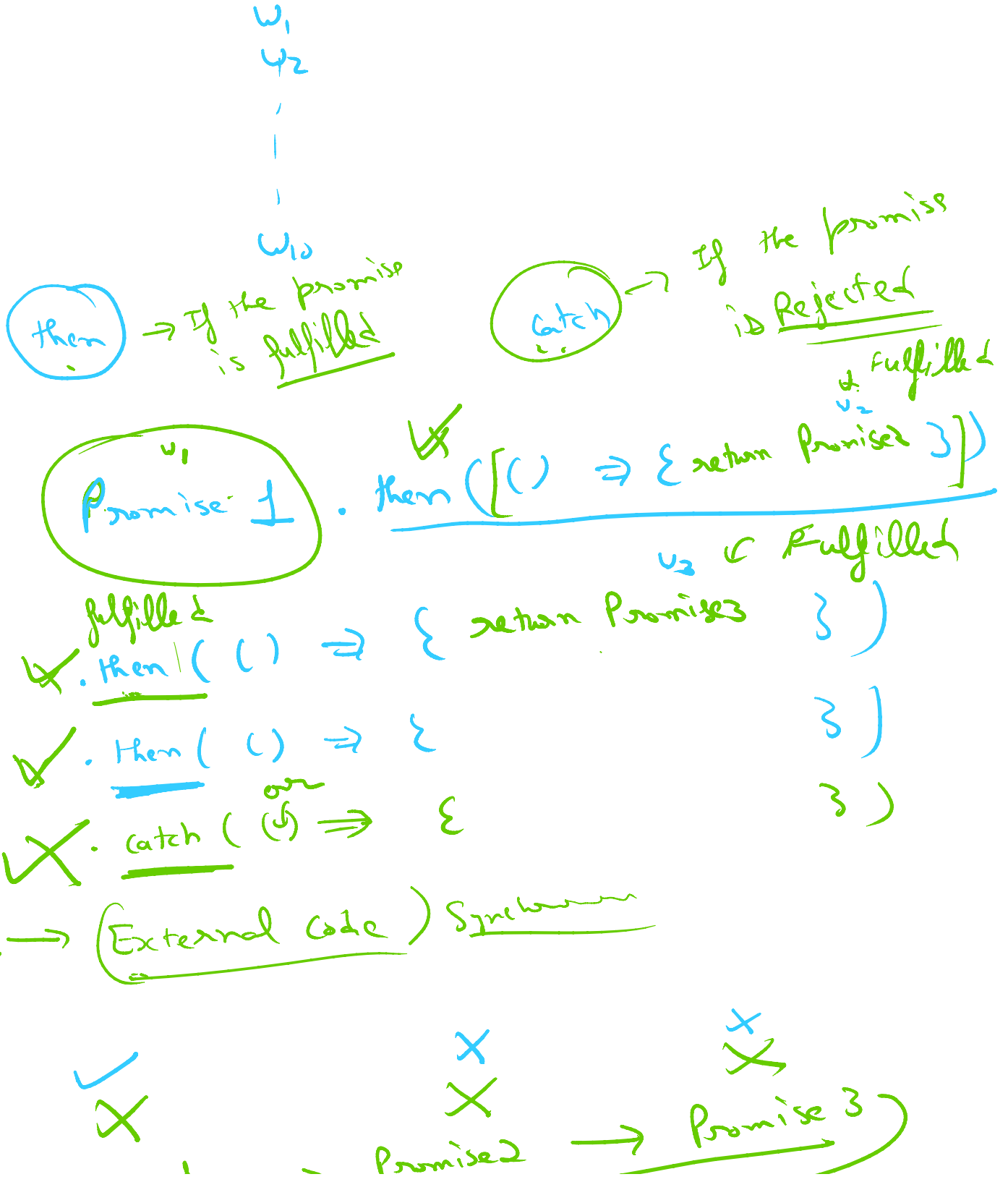
1) Every asynchronous function will return an object instantly, and the type of this object is Promise.

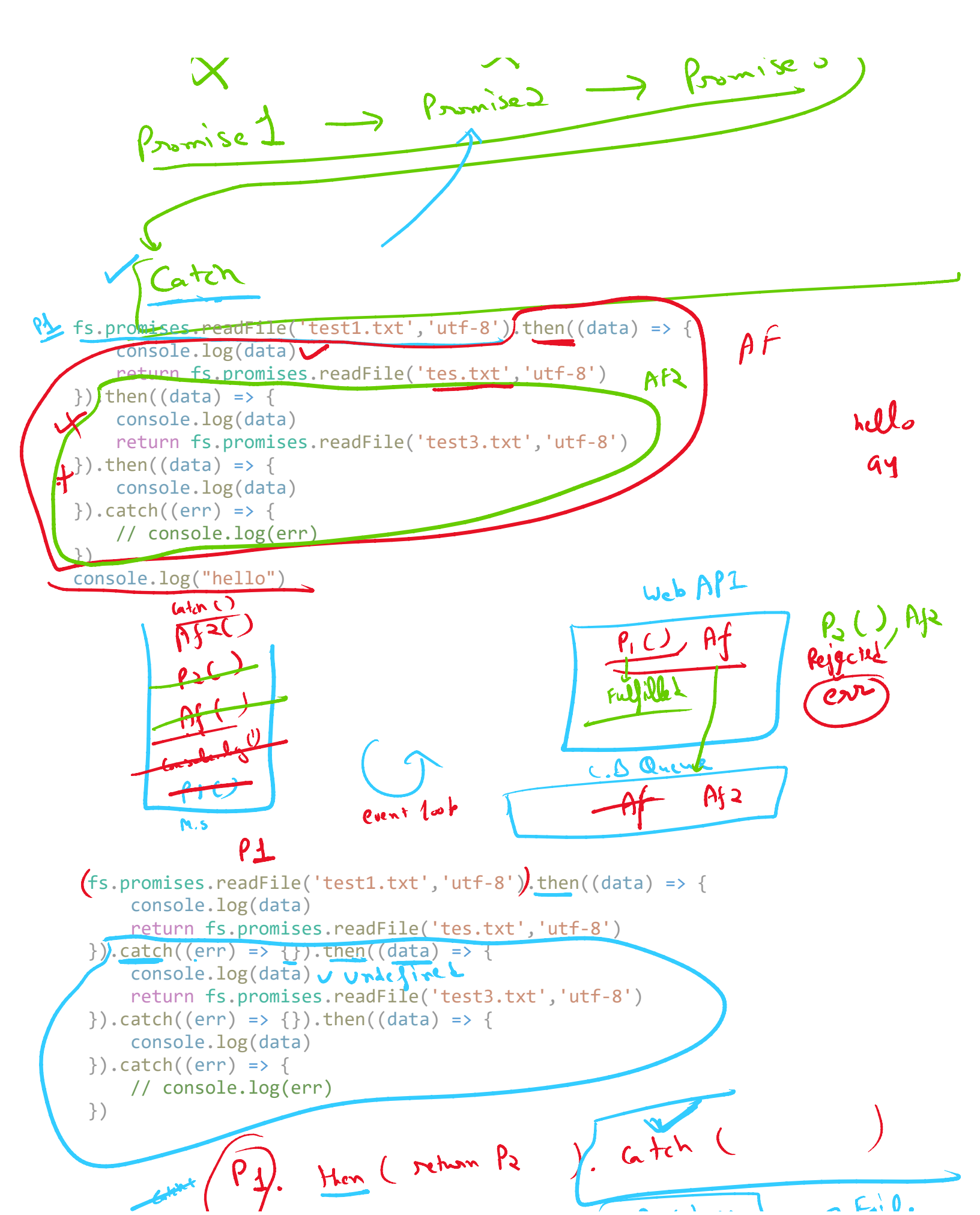
2) This object contains the status of async function

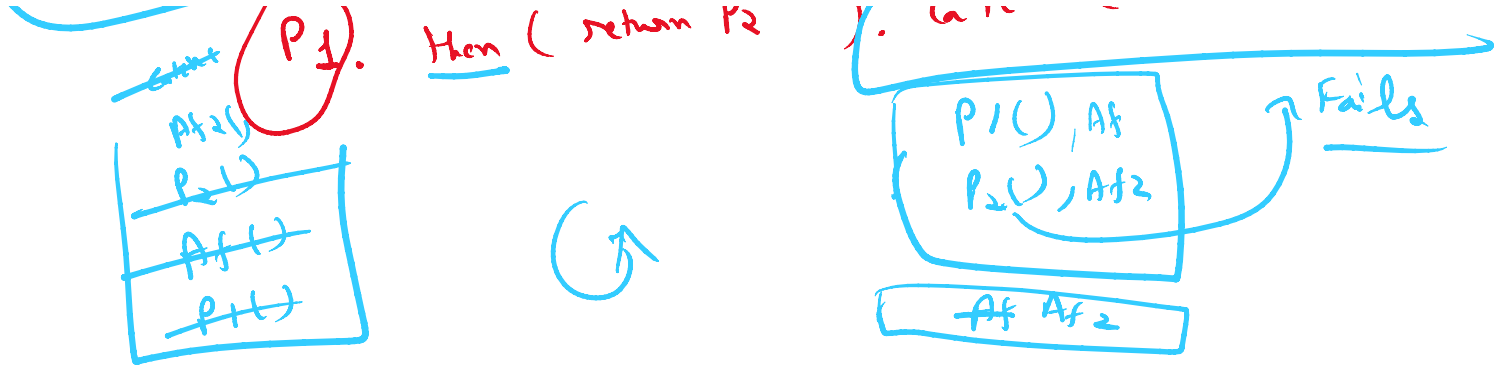


Chaining of Promises

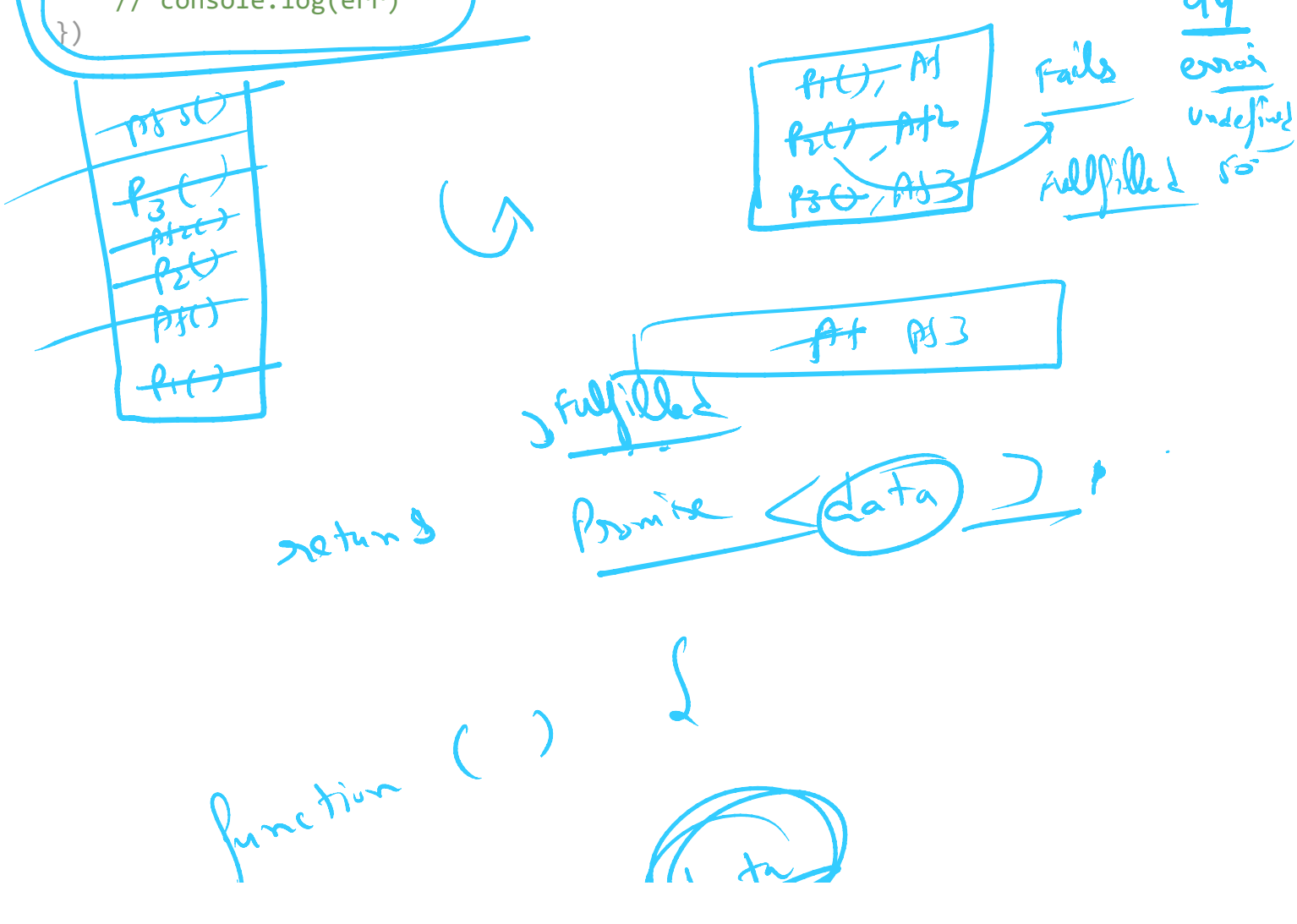
① Nesting was compulsory in case of callbacks




Promise 1
Promise 2
Promise 3
Catch
P1
fs.promises.readFile('test1.txt','utf-8').then((data) => {
console.log(data)
return fs.promises.readFile('tes.txt','utf-8')
}).then((data) => {
console.log(data)
return fs.promises.readFile('test3.txt','utf-8')
}).then((data) => {
console.log(data)
}).catch((err) => {
// console.log(err)
})
console.log("hello")
AF
AF2
hello
ay
Web API
P1()
Af
Fulfilled
P2(), Af2
Rejected
err
C.B Queue
Af
Af2
event loop
P1
fs.promises.readFile('test1.txt','utf-8').then((data) => {
console.log(data)
return fs.promises.readFile('tes.txt','utf-8')
}).catch((err) => {}).then((data) => {
console.log(data) undefined
return fs.promises.readFile('test3.txt','utf-8')
}).catch((err) => {}).then((data) => {
console.log(data)
}).catch((err) => {
// console.log(err)
})
P1. Then (return P2). Catch ( )




```
fs.promises.readFile('test1.txt', 'utf-8').then((data) => {
    console.log(data)
    return fs.promises.readFile('tes.txt', 'utf-8')
}).catch((err) => { console.log("error") }).then((data) => {
    console.log(data)
    return fs.promises.readFile('test3.txt', 'utf-8')
}).catch((err) => { console.log("error") }).then((data) => {
    console.log(data)
}).catch((err) => {
    console.log("error")
    // console.log(err)
})
```



function

return data

}

1) If Promise fulfilled, it will look for nearest .then, inbetween .catch's will be skipped.

2) If Promise Rejected, it will look for nearest .catch, inbetween .then will be skipped

3) .then, .catch always returns a fulfilled Promise with (return value) wrapped inside it.

↑ should not be a promise