

## Promises

Recap of IOC

↳ Readability Enhancers

↳ they can solve the problem of IOC.

↳ In JS, promises are special type of objects that get returned immediately when we call them.

↳ Promises acts as a placeholder for the data we hope to get back sometime in future.

↳ In these promise objects we can attach the functionality we want to execute once the future task is done.

once the future task is done, promises will automatically  
exercise the attached functionality.

function

`{ } x = fetch("http://www.xyz.com")`

Assume fetch is written using promises then, it will immediately return a promise obj which will act as a placeholder for the result.

The diagram illustrates the execution of the `fetch` function. It starts with a function call `fetch("http://www.xyz.com")`. An arrow points from the `fetch` function to a promise object, represented by a green circle with horizontal lines inside. Another arrow points from the promise object to the variable `x`, which is also represented by a green circle with horizontal lines inside. A third arrow points from the variable `x` to the `result` variable, which is also represented by a green circle with horizontal lines inside. The `result` variable is underlined, indicating it is the final output.

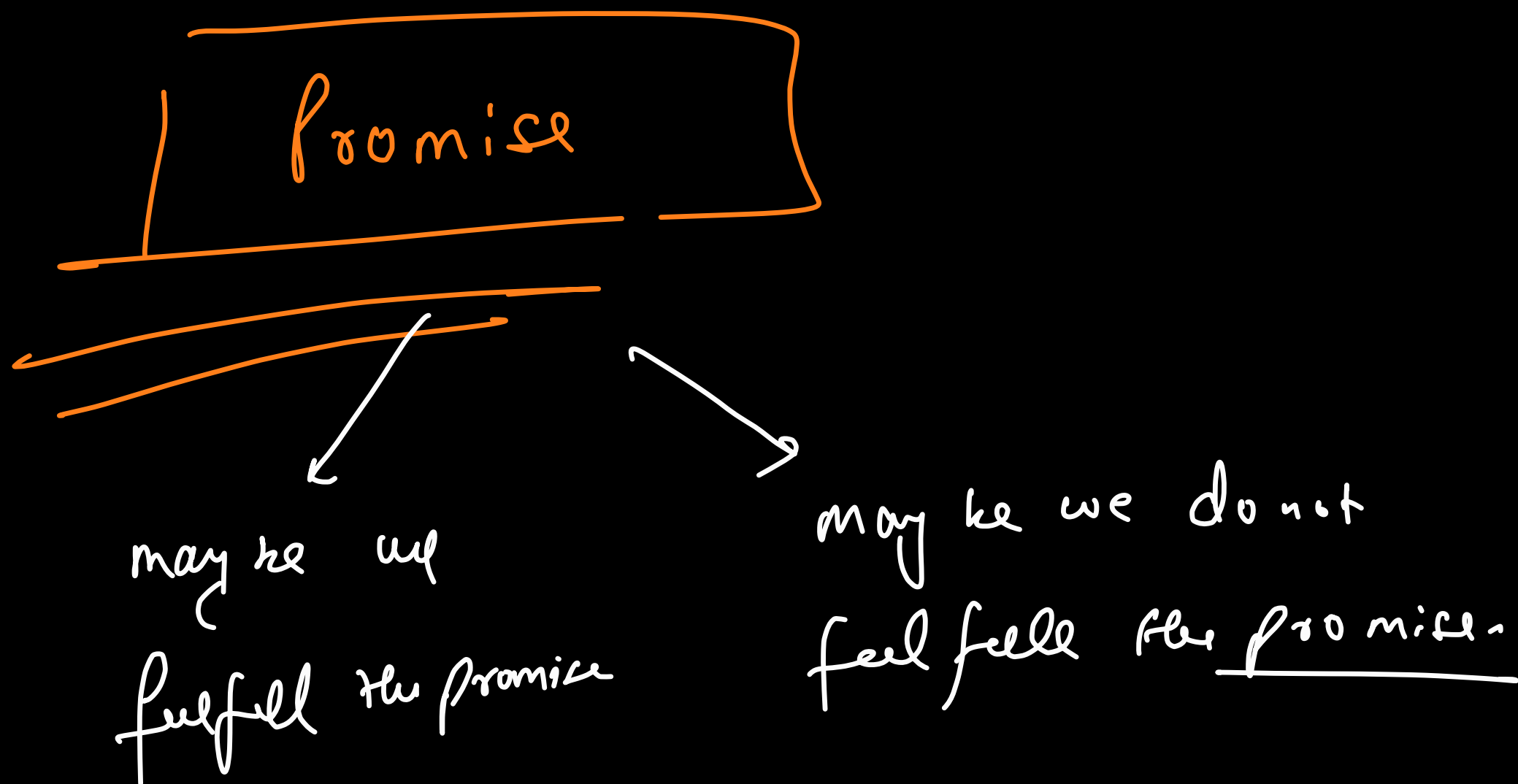
```
function fun(x, cb) {  
  for (i=0; i<x; i++) { // - }  
    cb(); cb();  
  }  
}
```

→ fun(10, function nec() {  
 console.log("done");  
});

```
fetch("http://www.xyz.com", function exec() {  
    console.log("done");  
})
```

⇒

- 1) How we can create a promise ??
- 2) How can we consume a promise ???

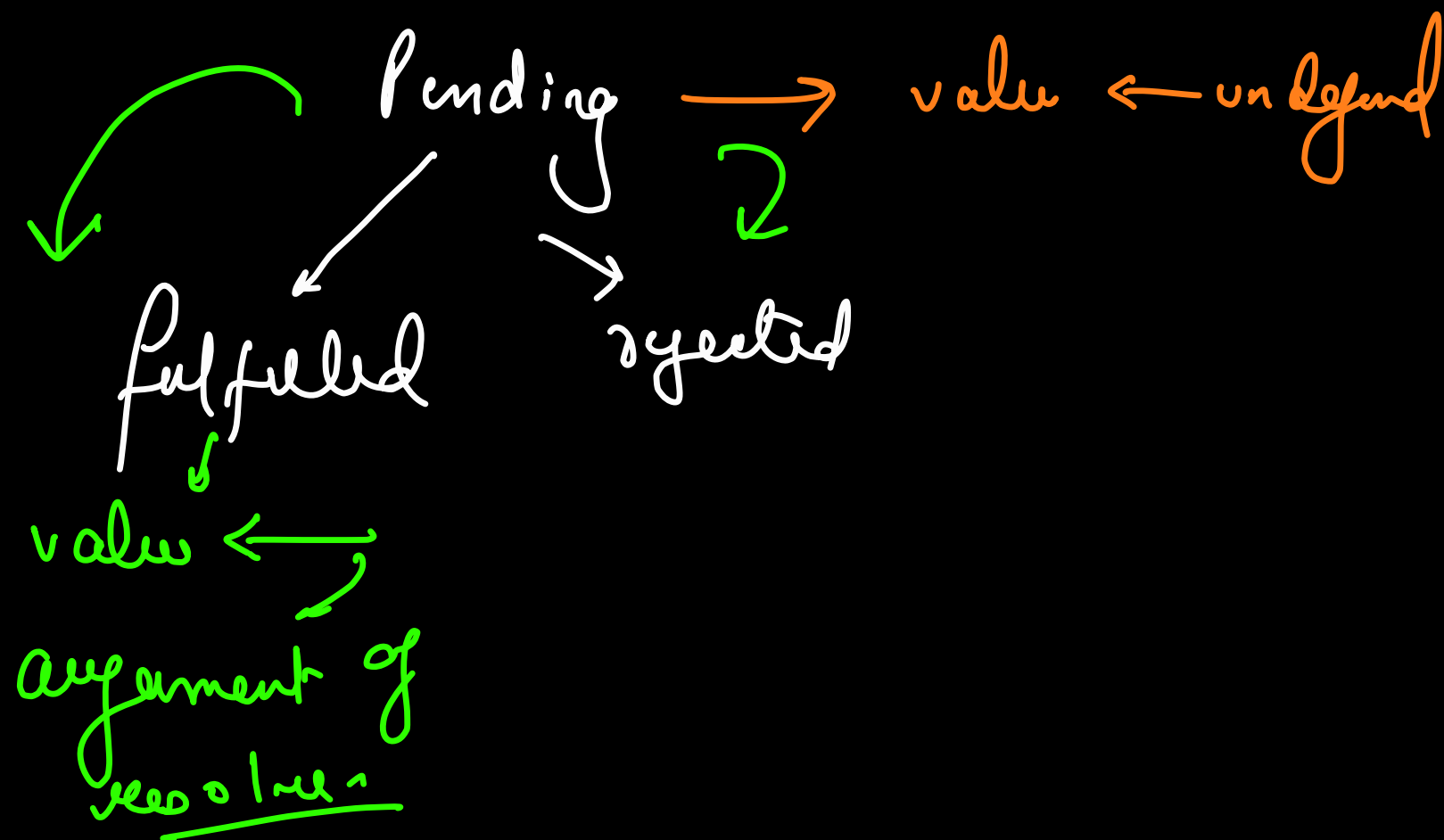


How to create a promise ?? → state  
→ value

↳ Creation of a promise object is sync in nature.

State

- 1) pending → when we create a new promise object this is the default state it represents work in progress.
- 2) fulfilled → if the operation is completed successfully
- 3) rejected → if op was not successful



keyword new Promise (f) this constructor expects a callback  
↓  
constructor  
k, ?? . executor function

these are func's

new Promise (function (resolve, reject) {

// inside this function we can write our  
// time consuming task.  
resolve(x);

})



→ whenever in the impl of executor callback you call the resolve function, the promise goes to a fulfilled state.

if you call reject func<sup>n</sup>, it goes to a rejected state

& if you don't call anything, promise remains in pending state.

→ with whatever argument we call resolve or reject with gets assigned to the value properly -