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.

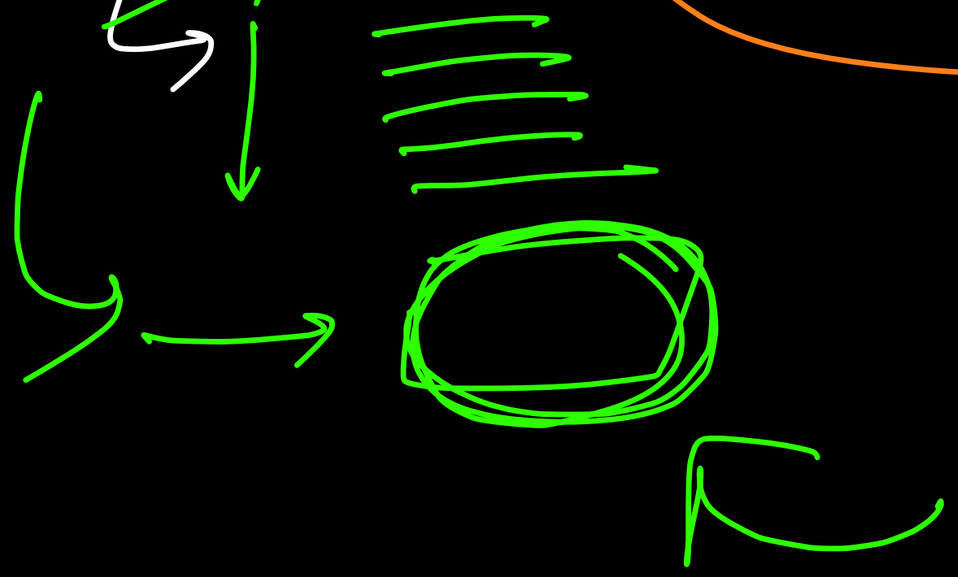
↳ Promise acts as a placeholder for the data we hope to get back sometime in future.

↳ 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 exec the attached functionality.

runtime

```
{ }
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

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

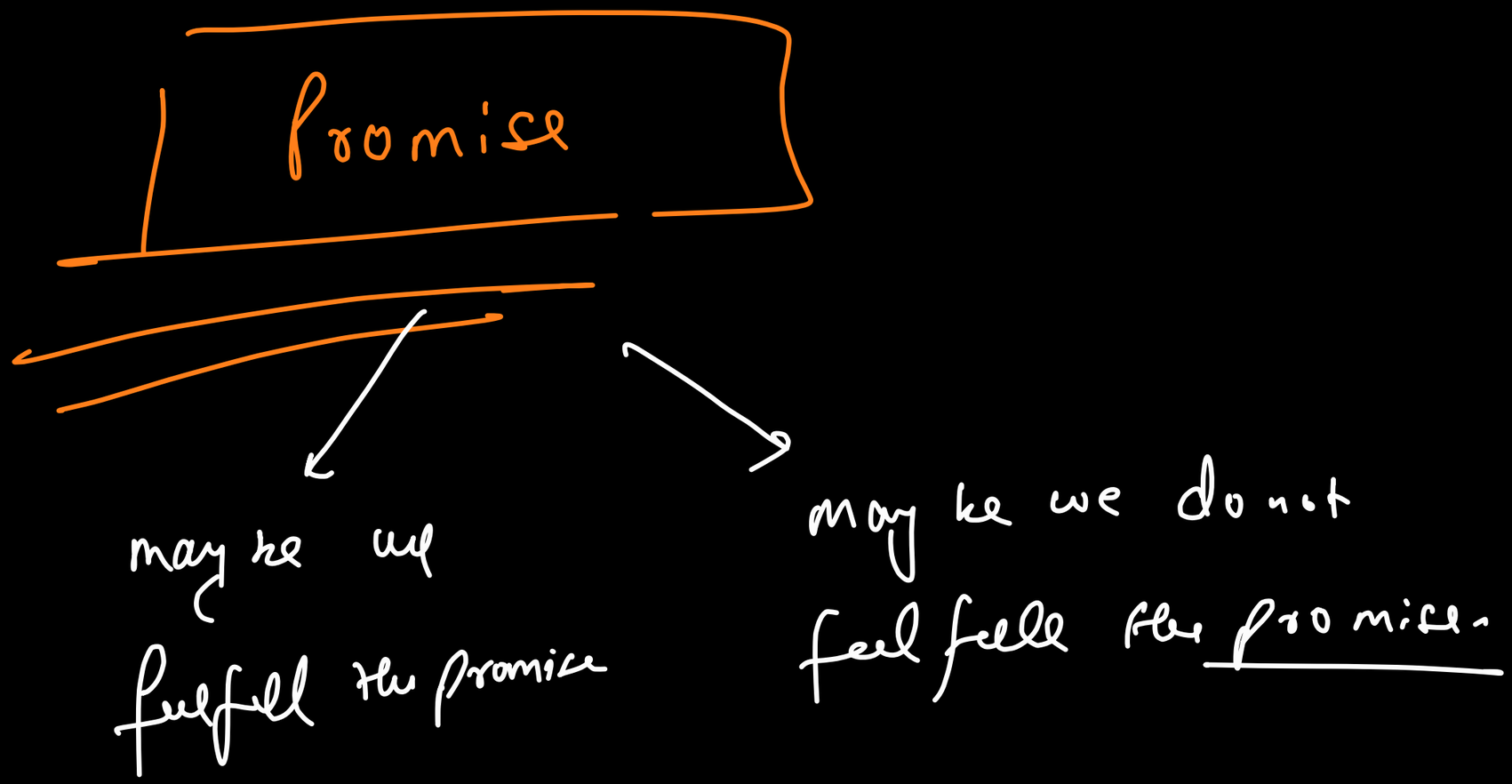
↳

```
fun(10, function exec(){
    console.log("done");
});
```

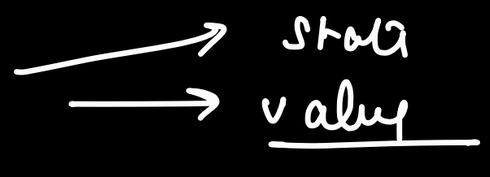
```
fetch ("http://www.xyz.com", function exec() {
        console.log ("done");
});
```

1) How we can create a promise ??

2) How can we consume a promise ??



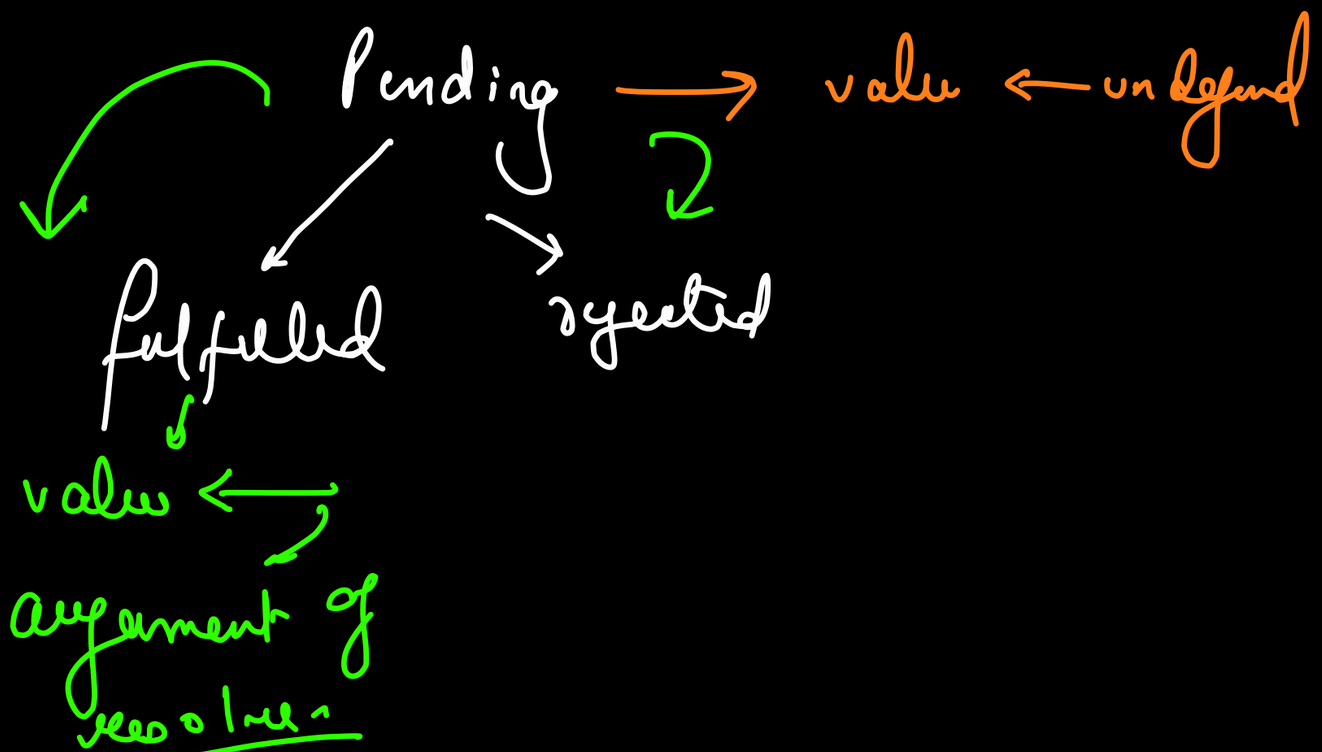
How to create a promise ??



↳ Creation of a promise object is sync in nature.

States

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 → ?? . executor function

these are func^n

new Promise (function (resolve, reject) {

// inside the function we can write our

// time consuming task.

resolve (x);

})

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

if you call reject func, it goes to a rejected state

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

⇒ with whatever argument we call resolve or reject with gets assigned to the value property.