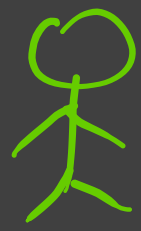
Callback Hell :-



Ramu friends

⇒ Planning fun day!

⇒ ① Toys - clean it up, then we will go to park

⇒ ② After park, we will eat Ice cream!

⇒ ③ After Ice Cream, we buy toys

⇒ ④ After toy store, go home

① Toys clean

↓

② PARK

↓

③ Eat Ice Cream

→ ④ Buy toys

→ ⑤ Go home

```
function aa(cb1) {
  cb1(cb2);
}

function cb1(cb2) {
  console.log("Hi from cb1");
  cb2();
}

aa(cb1);

function cb2() {
  console.log("Hi from cb2");
}
```

Problem:

⇒ Hard to read and understand

⇒ Difficult to Maintain

⇒ Error handling is tricky

1. GetDressed
2. EatBF
3. Go Out

```
function goout()
{
Console.log("—");
}
```

```
function getDressed(callback){
  Console.log("—");
  setTimeOut(()=>{
    callback();
  },1000);
}
```

```
getDressed(()=>{
  eatBreakfast(()=>{
    goout();
  });
});
```

```
function eatBreakfast(callback){
  Console.log("—");
  setTimeOut(()=>{
    callback();
  },1000);
}
```

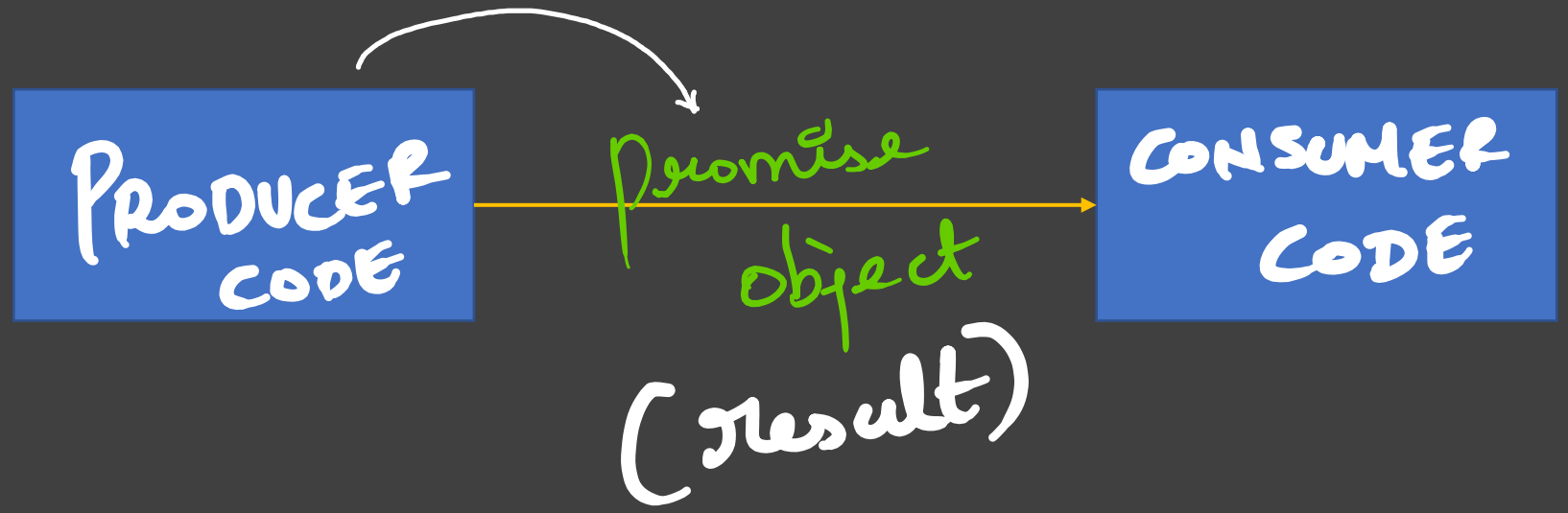
PROMISES :- (Asynchronous Programming)

There are 2 types of code in Asynchronous programming

* Producer Code : The code that produces some result
* Consumer Code : The code that consumes the result produced by the producer code.

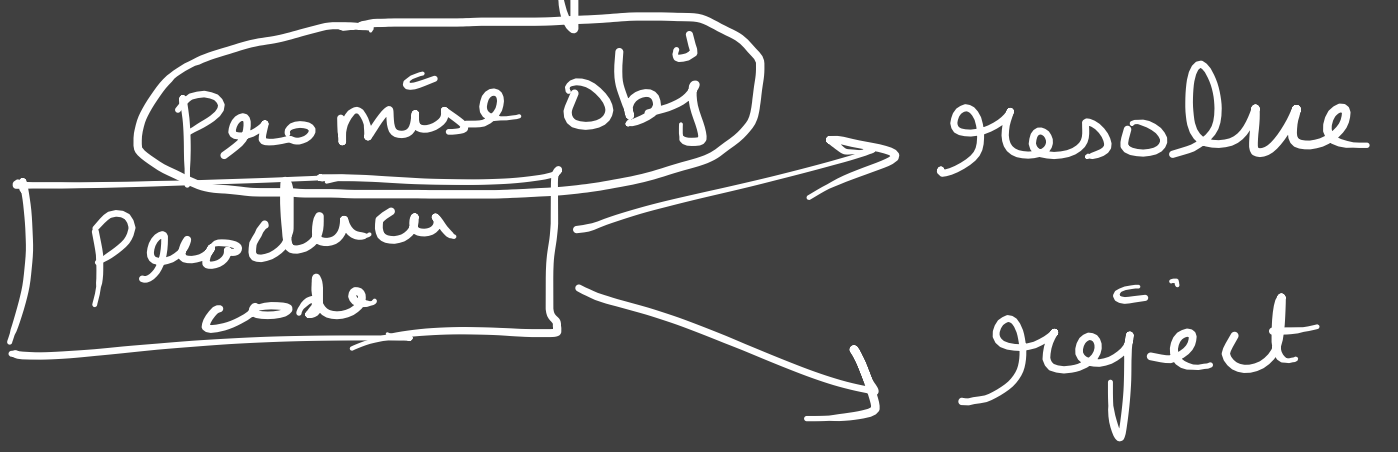
"A promise is an object which makes the result produced by the producer code available to the consumer code."

"Thus linking them together"



⇒ The producer code is contained inside the promise object.

⇒ The producer code, which is inside the promise object, contains resolve & reject callbacks.



```
let promiseObj = new Promise((resolve, reject) =>
{
    // producer code
});
```

=> The producer code is executed as soon as the promise Object is Created. You do not need to explicitly call the producer code.