what fn can do is decided during process array function call

parameters → expecting these values during func^n call

```
13  function processArray(array, fn) {
14
15  }
16
17
18  processArray([2,3,4,1,5], function cube(x) { return x*x*x; }); // passing function as an argument
```

fn can give you access to call cube

function body

argument passed for "array" parameter

argument passed for "fn" parameter

internally fn is that we are passing function as an argument.

```
function cube (x) {
    return x * x * x;
}
```

anyone, anywhere calls cube func^n with a number will get cube of the no as return.

[1, 2, 3]

i = 0

array [i] = 1

i = 1

array [i] = 2

fn (array [i])

cube (2) → 8

fn (array [i])

cube (1) → 1

```
13  function processArray(array, fn) {
14
15  }
16
17
18  processArray([2,3,4,1,5], function cube(x) { return x*x*x; }); // passing function as an argument
```

This a HOF

this is a callback

HOf → It is a function that takes one or more functions as arguments.

Higher Order Function

Callback → Call-back is a function, which is an actual argument to a Hof and is passed as an argement to a HOf

One Imp use case of callback →

→ Sometimes our functions might not complete immediately and complete in future -

↳ Ex → a func^n that downloads a file -

In these cases when the execution of func^n completes in future ; what should we do can be controlled by callbacks.

Inbuilt fun^{n}

JS gives us access to some already built in function.

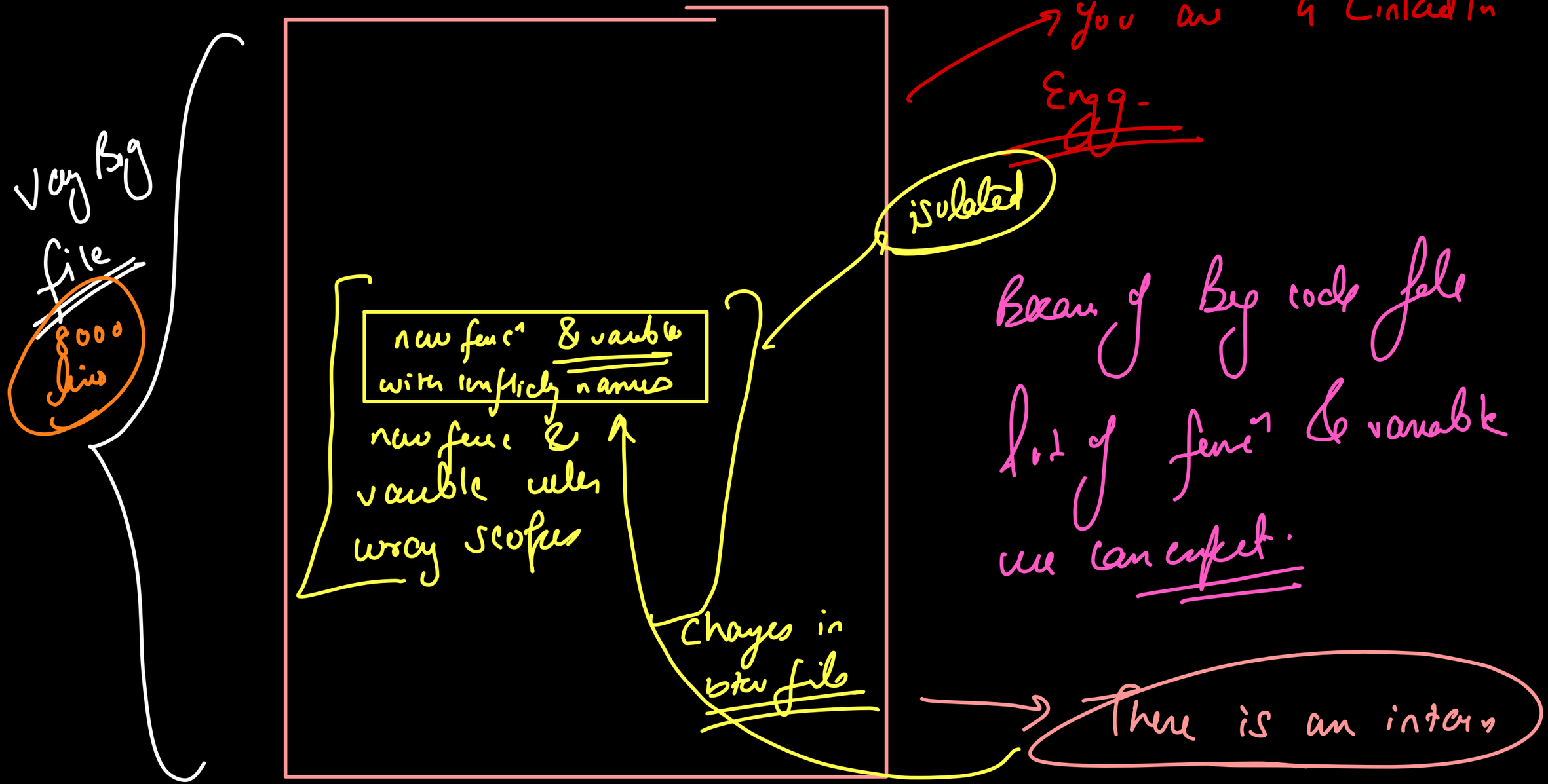
Ex → Math.sqrt , forEach , map etc

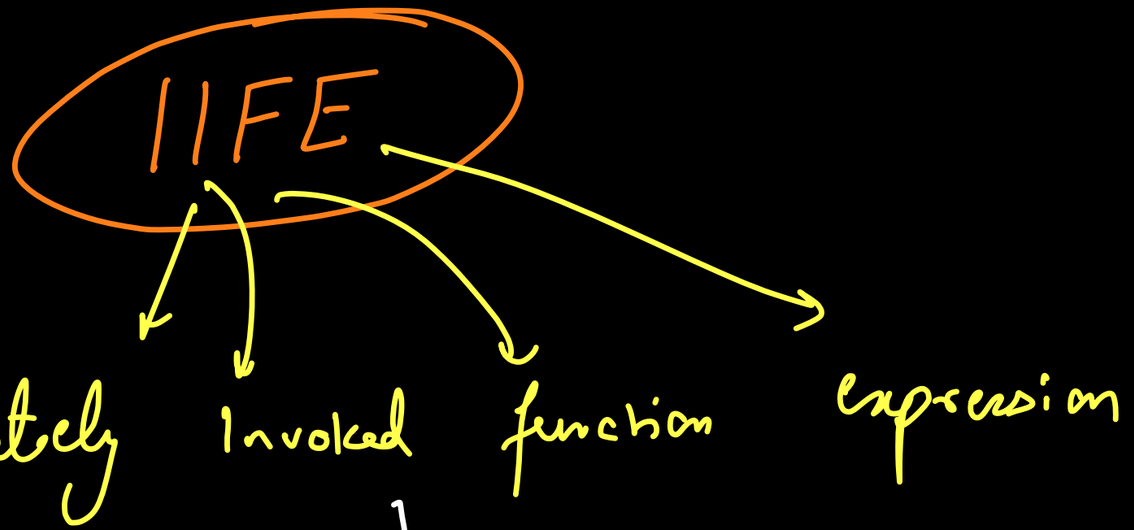
Because func^{n} are first class citizens, we can return a function also.

```
1   function fun() {
2
3       console.log("Called the function fun");
4
5       return function cleanUp() {
6           console.log("Cleaning up the resources");
7       }
8
9   }
10
11
12  const result = fun();
13  result();
```

from fun we are returning cleanUp

result = function cleanUp() {
....
}





meany of immediately invoked is that the moment you define the fen^n expression it is immediately executed then & cannot be accessed anywhere else.

```
function downloadScriptFromIdeoneAndPopulateDatabase() {
    // this was an old function

    console.log("Old function implementation");
}


// more code  8000 lines
downloadScriptFromIdeoneAndPopulateDatabase();
// somewhere between



// intern by mistake makes the same function
// To solve this, we can wrap                    de that can break, into an IIFE
// Docs: https://developer.mozilla.org/en-US/docs/Glossary/IIFE
(function () {
    function downloadScriptFromIdeoneAndPopulateDatabase() {
        console.log("New function created");
    }
    console.log("Calling intern code")
    downloadScriptFromIdeoneAndPopulateDatabase();
})();
```

Follow link (cmd + click)

invocation of the func^ expression

if we donot use the 2nd pair of parenthesis, it is a dead code-

x = 10

print(x)

print(x)

x = 20

print(x)

print(x)

fn → old

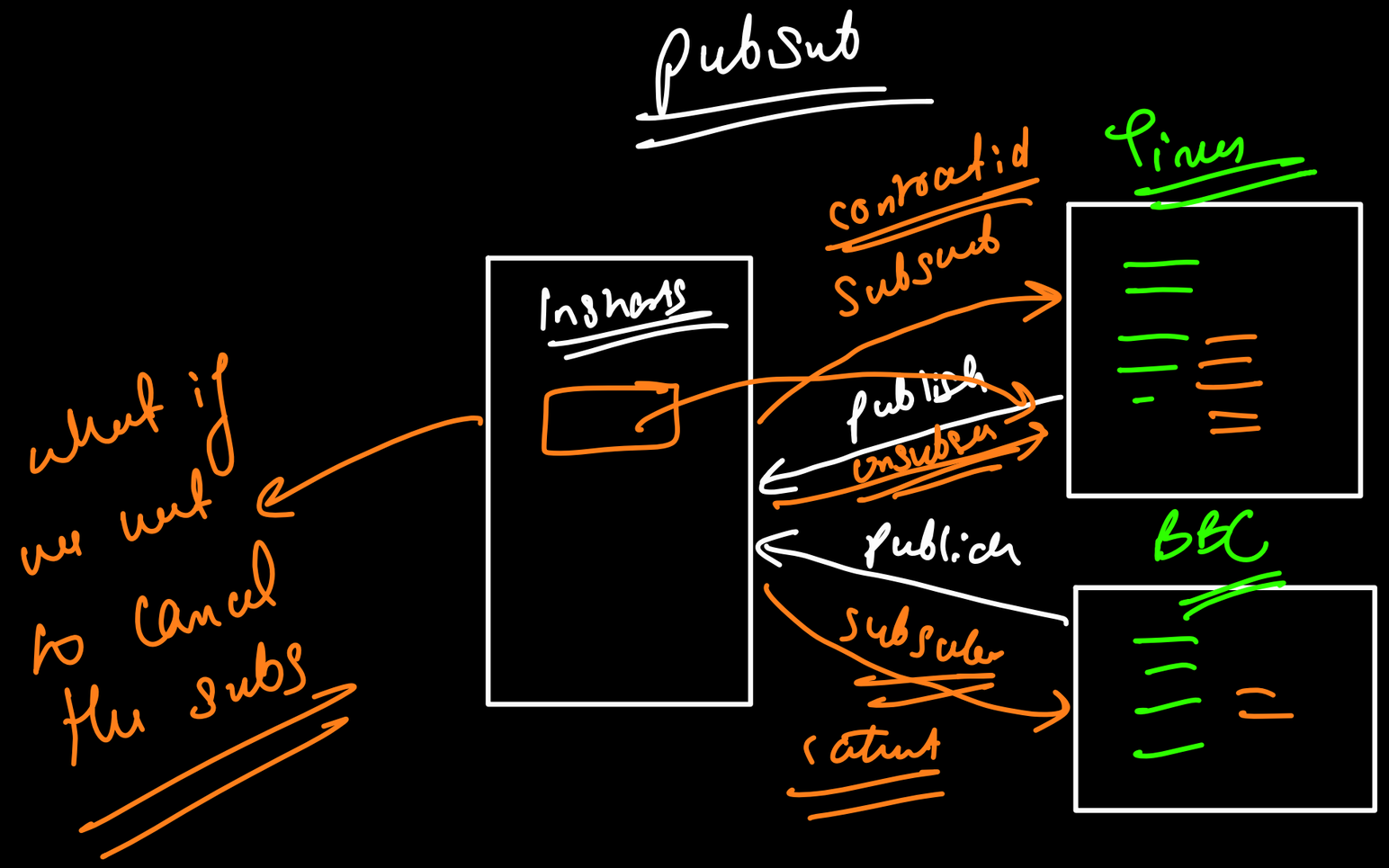
fn()

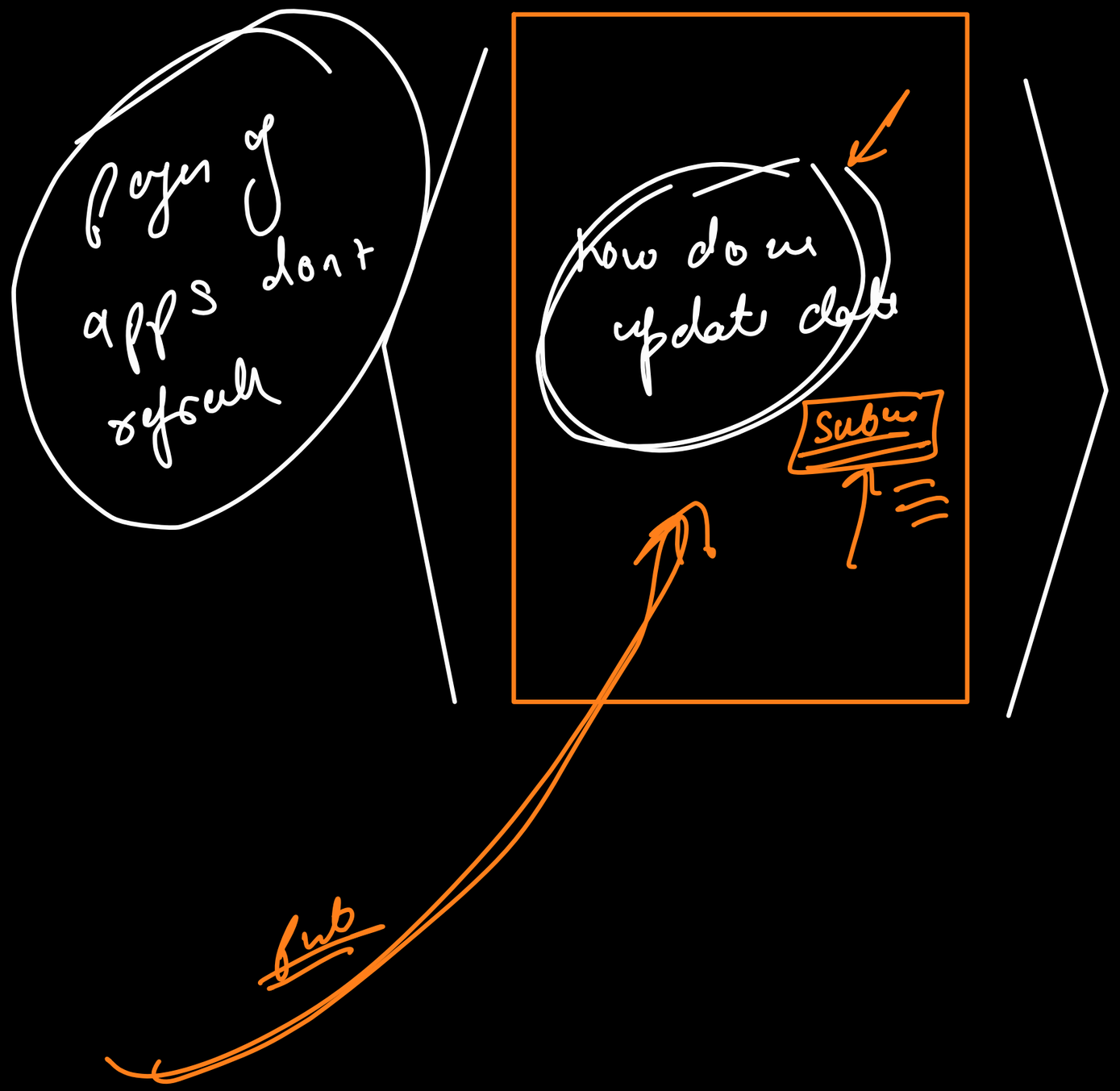
fn()

fn()

fn → new

fn()




Reyes of
apps don't
refresh
How do we
update data
Subs
fwb

Chaining Of functions

Use Case

Promises

obj.func1(x).func2(y).fun3(z).....

.then()
.then()
.then()

obj.func1(x)

JS object

```
{
  func1: function(x){...}
}
```

code → Todo app

→ low pay easy inter
→ good pay easy inter
→ high pay hard inter → google, codecademy, sprinklr

frontend — JS

2019 → off campus

interview

JS + dsa
DSA
maths + DSA
design
hm

2 tips → it is ok to say no about interview
fear

→ HTML + CSS learning