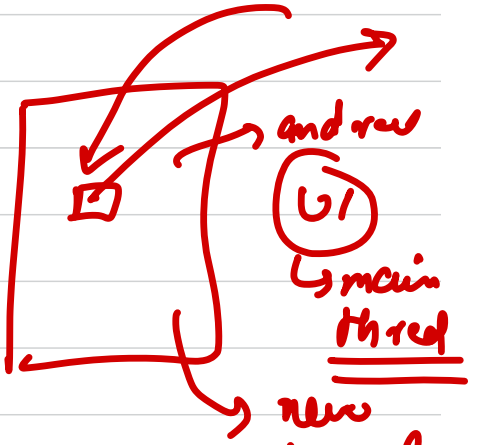
Imp features of JS

↳ JS is synchronous in nature (any piece of code understandable by JS will be executed line by line)

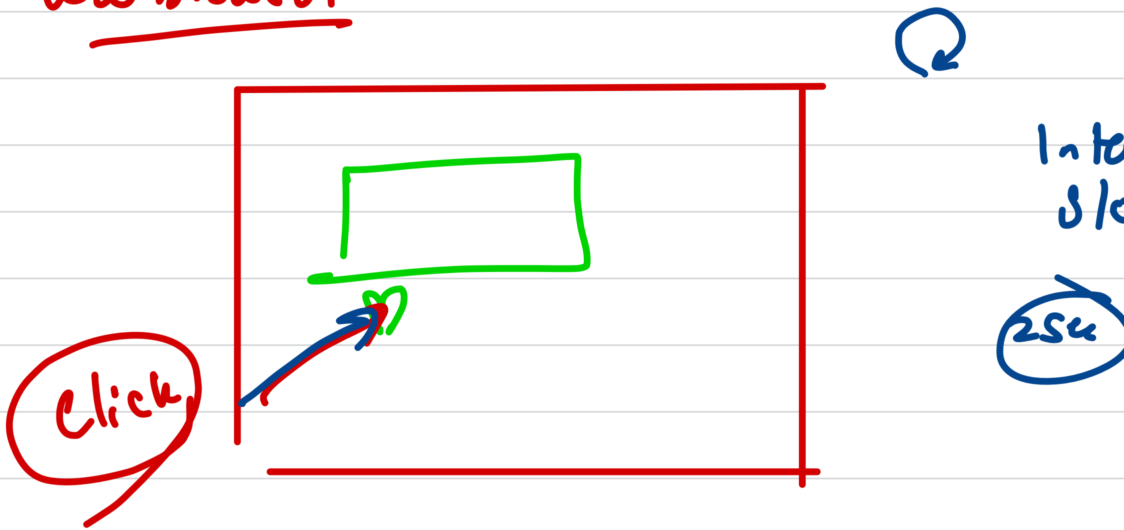
→
→
→
→



↳ JS is Single threaded.

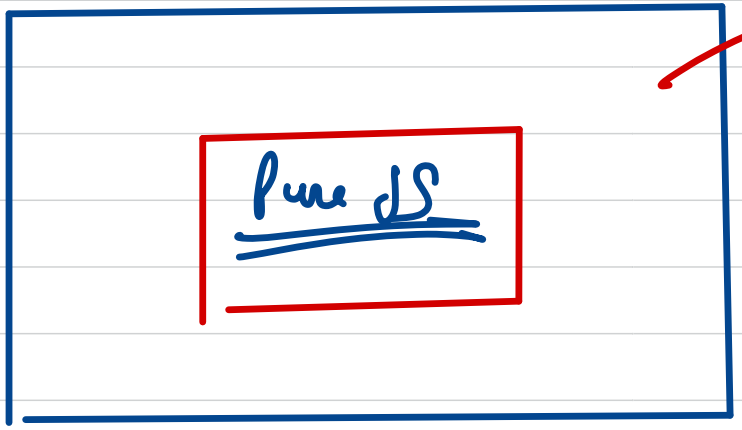
Still gives non blocky nature
↳ for features apart from pure JS

Web browser



Pure JS, doesn't know things like timers, counters, how to connect to internet, how render and control web pages etc etc

Runtime env provides extra features to JS



runtime env

engine

C++ & 2

Browser

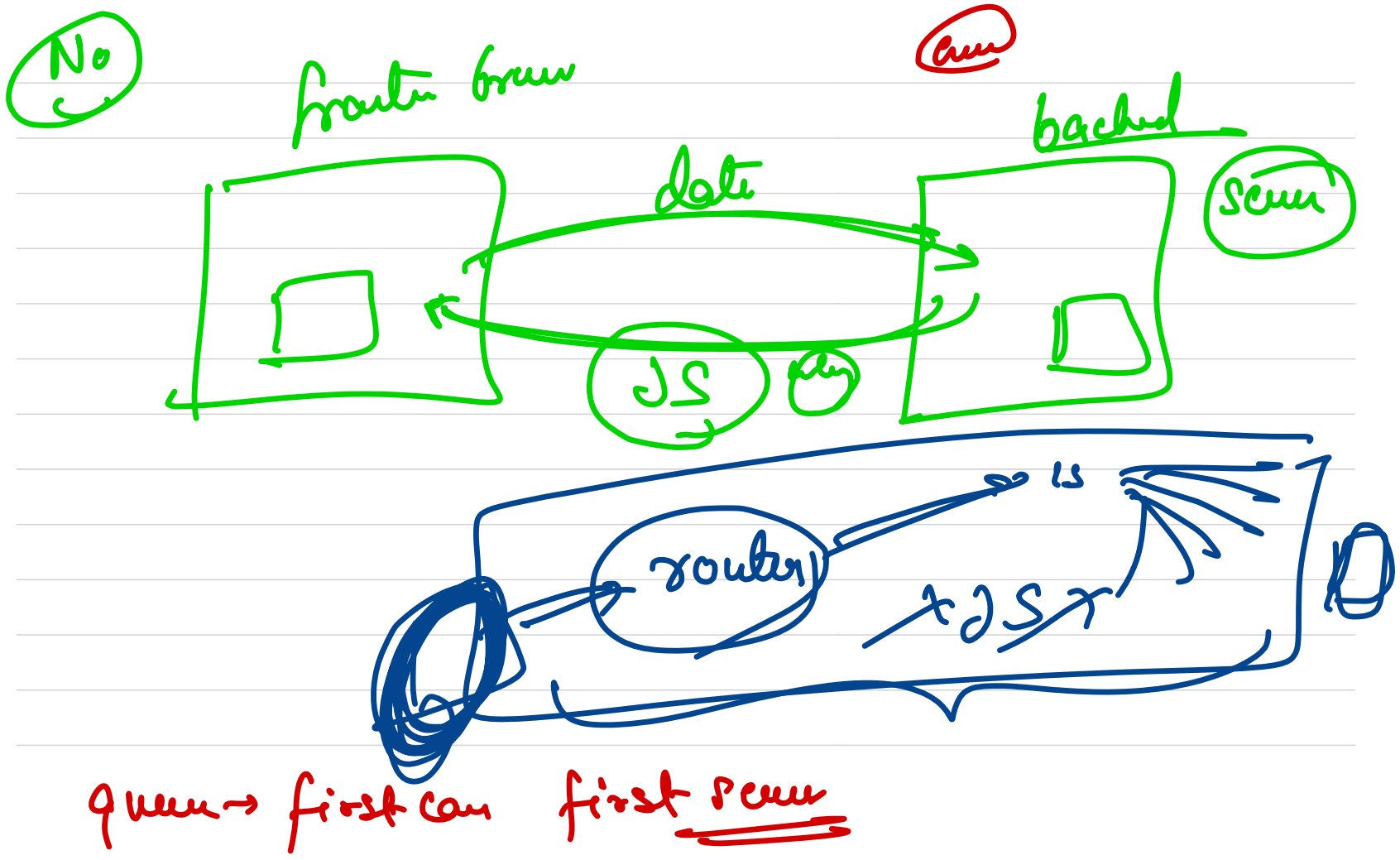
node

JS ⟷ browser

2009

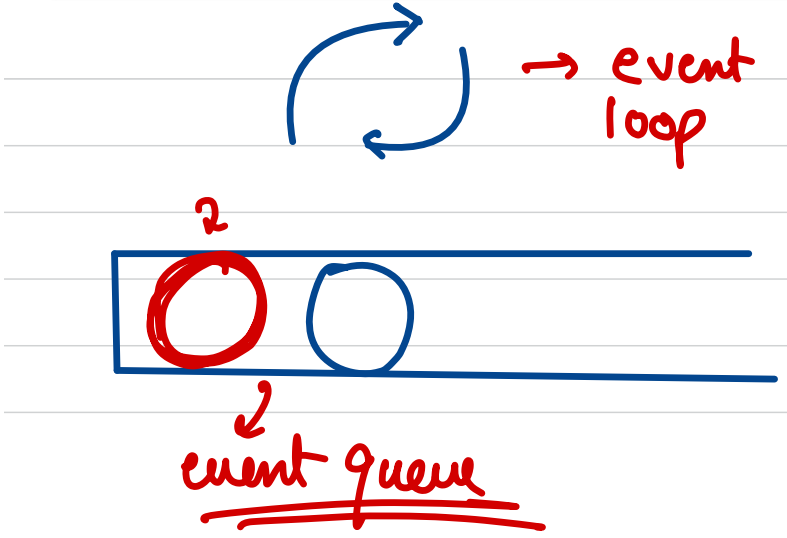
Ryan Dahl

node



queue → first come first serve

```
1   function printhello() {
2       console.log("hello");
3   }
4
5   function blockforfewsec() {
6       for(let i = 0; i < 1000000000; i++) {}
7       console.log("ending blocker");
8   }
9
10  setTimeout(printhello, 0);
11  blockforfewsec();
12  console.log("me first");
13  blockforfewsec();
```



JS will line by line execute the native code. The moment JS sees anything apart from native then it does the following -

→ JS just do one operation which is to send signal to the runtime, that a runtime feature is accessed.

→ Then it sends the function & arguments to the runtime and moves forward! It doesn't wait.

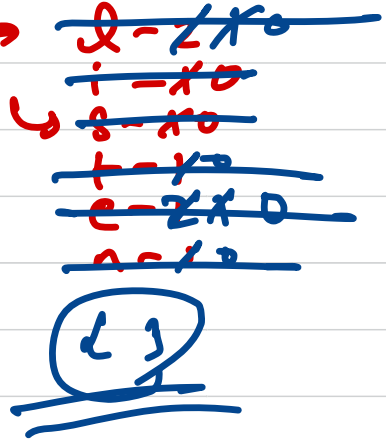
The moment runtime finishes it's func, we don't execute it then & there. We make it wait inside the event queue.

So JS will keep on execute the code & event queue will wait till the time our call stack / global code is yet to be completed.

Q) Given two strings, check if they are anagram of each other.

Ex: listen | silent

Ans → true



n → length of string

$n \leq 10^6$

Both the strings should have same occurrence of char

JS → obj

key-value
char-freq

O(n)

O(1)

Matrix traversal

Quick Sort

more problem on recursion

Hashing

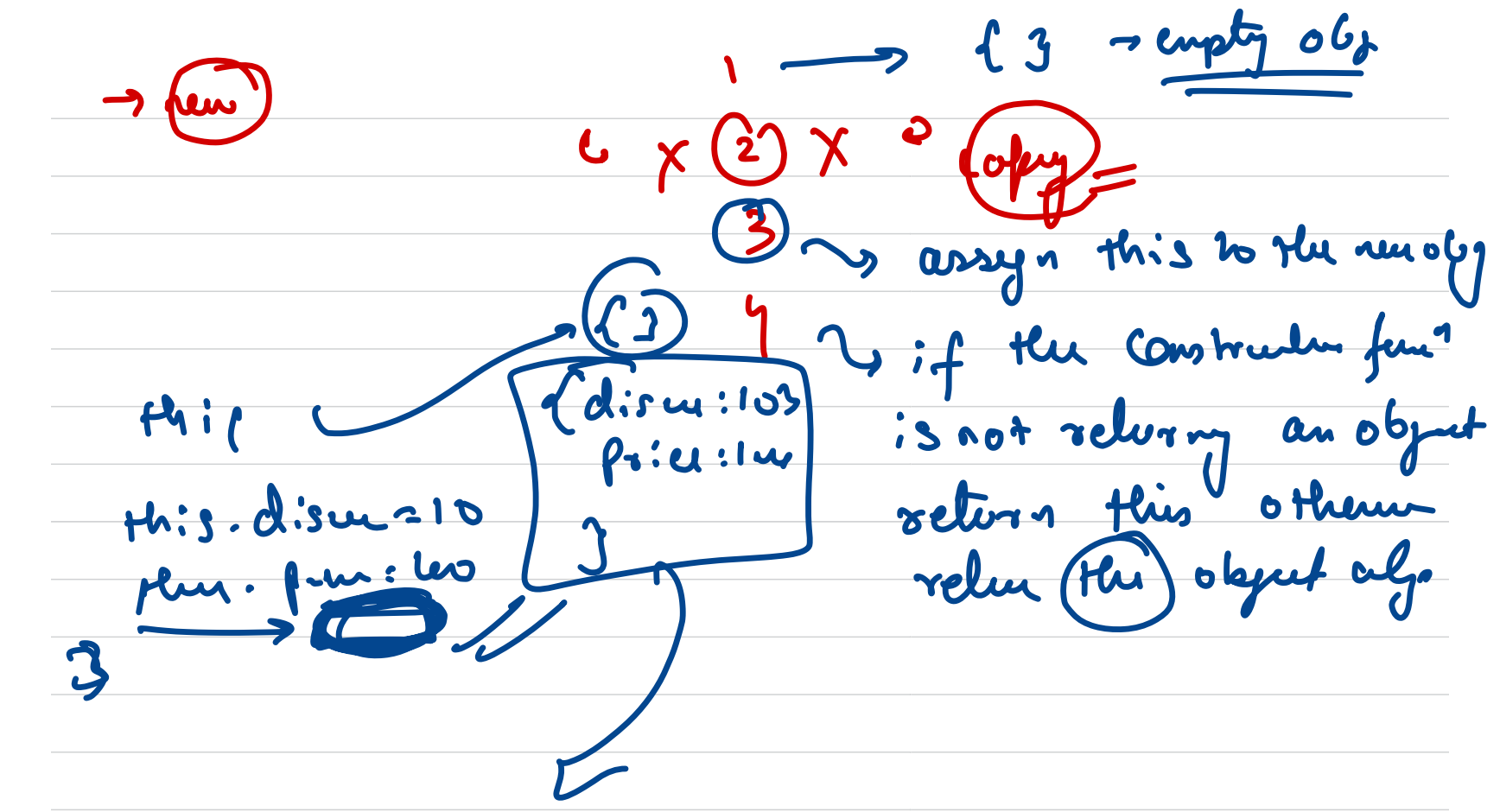
Code of quicksort

this & new → karan Gupta

this → archit

event loop & gun → Arnab

Private → Tirth


new
1
{ } → empty obj
c x 2 x o
copy
3
assign this to the new obj
if the constructor fun^n
is not returning an object
return this otherwise
return this object obj
{ }
this
{dis:10
pri:100
this.dis=10
new.pri=100
}

What are callbacks ??

↳ a lot of time cb, async code

```
1   // whatever functions we will write will not hamper the main js thread
2   // Write a dummy program that can download a file, then compress it and then upload it
3
4   function download(url, callback) {
5       // somehow it will download the file from the given url
6       console.log("Started downloading from ", url);
7       setTimeout(function() {
8           console.log("Download done for the file image.jpg");
9           callback("image.jpg");
10      }, 2000);
11  }
12
13  function compress(filePath, callback) {
14      // somehow it will compress the file present on the file path
15      console.log("Started compressing the file ", filePath);
16      setTimeout(function() {
17          console.log("Compression done for the file", filePath, "and saved it as image.zip");
18          callback("https://uploader.com", "image.zip");
19      }, 4000);
20  }
21
22  function upload(url, file) {
23      // uploads the file on the given url
24      console.log("Started uploading", file, "to", url);
25      setTimeout(function() {
26          console.log("Successfully uploaded the file");
27      }, 3000)
28  }
29
30
31  download("https://downloader.com/image.jpg", function(file) {
32      compress(file, function(url, file) {
33          upload(url, file);
34      })
35  }
36
```

Start downloading
Download done
Start compressing
Compress in done
Start uploading
Successfully uploaded

Promise

Promises are readability enhancers. (they enhance the code a bit).

These are objects but special ones. They get returned immediately when we make a runtime env func' call

```
function display(data){
    console.log(data);
}

let fd = fetch("http://www.abc.com");
fd.then(display);
```

variable

placeholder for promise object

async

runtime env func

fd variable will be storing a promise object

So promise does 2 things, one inside JS & one outside JS i.e runtime.

1) It will give the func^n call to the runtime env.

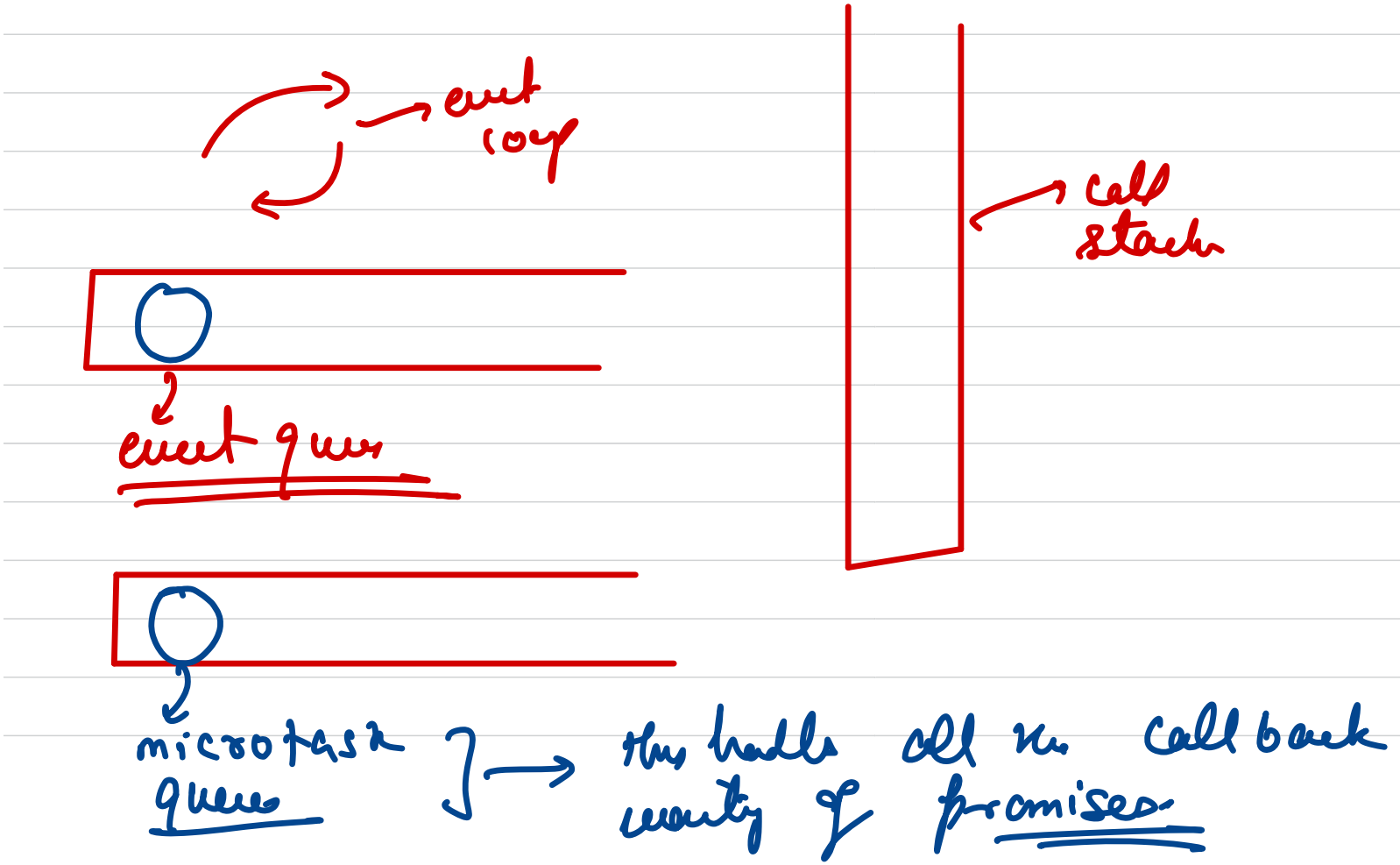
2) It gives a placeholder to the variable in the form promise object

Inside the placeholder promise object we have few properties

① value : Initially when the fetch call is not completed, value will be undefined but as soon fetch is completed the result of the func^ call will be stored in this value property.

② onfulfillment: This is an array of fcn^n. Now as soon as fetch is completed & value property gets the exact response, we one by one start executing the functions present in this array with value property as argument.

Who fills this array?? So this promise object has a "then" fcn^n also, whatever fcn^n we pass inside then gets added to onfulfillment


event loop
call stack
event queue
microtask queue
this handles all the callback
events of promises

microtask queue has higher priority