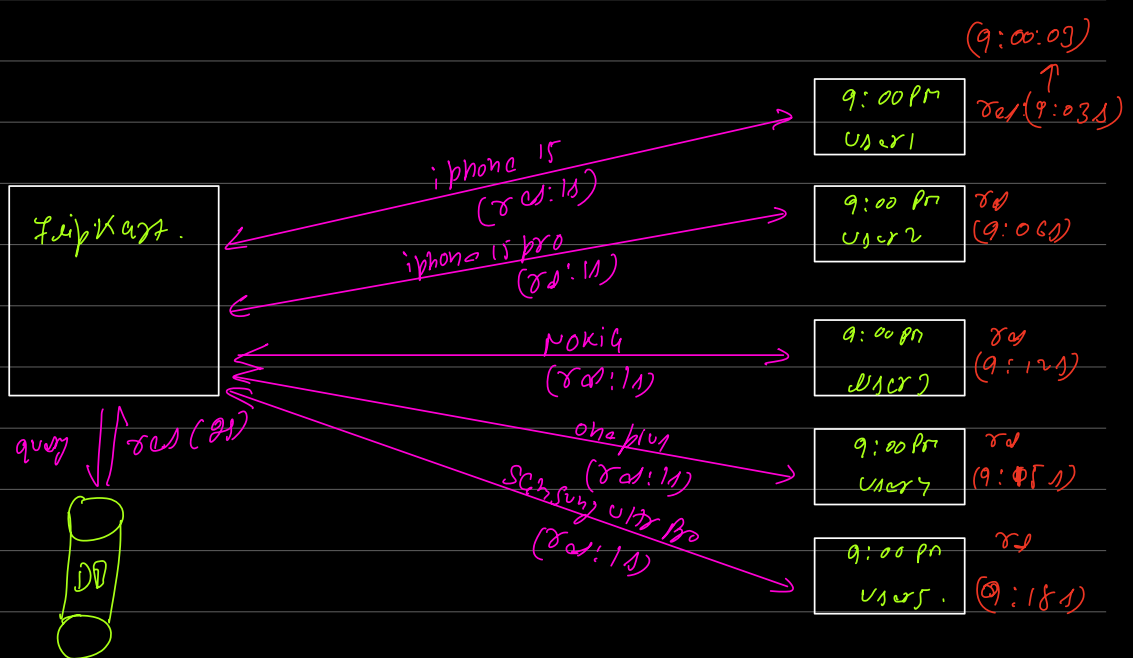
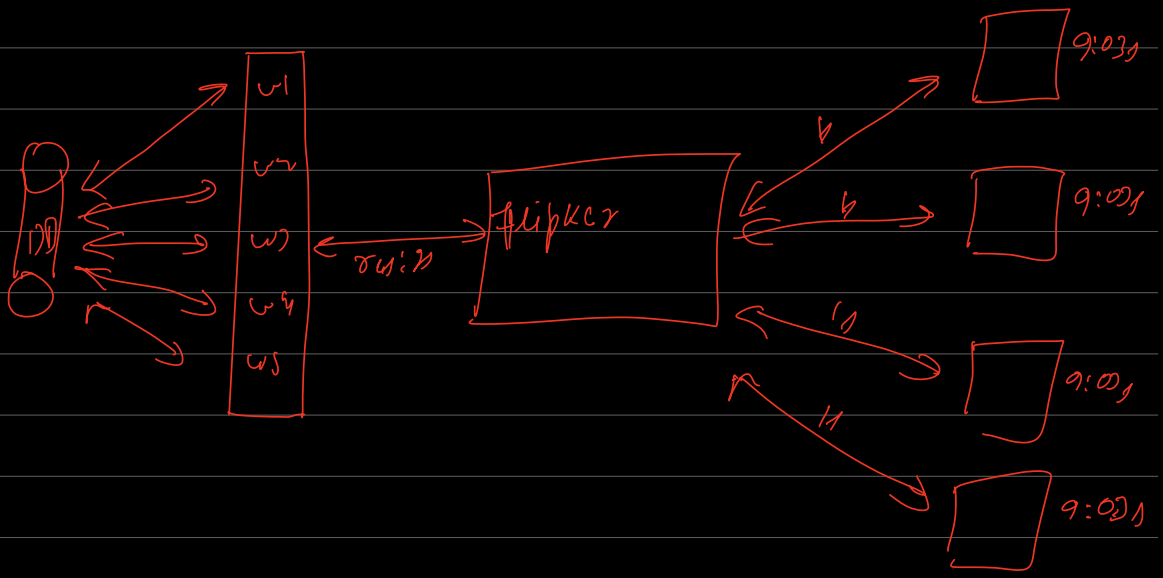
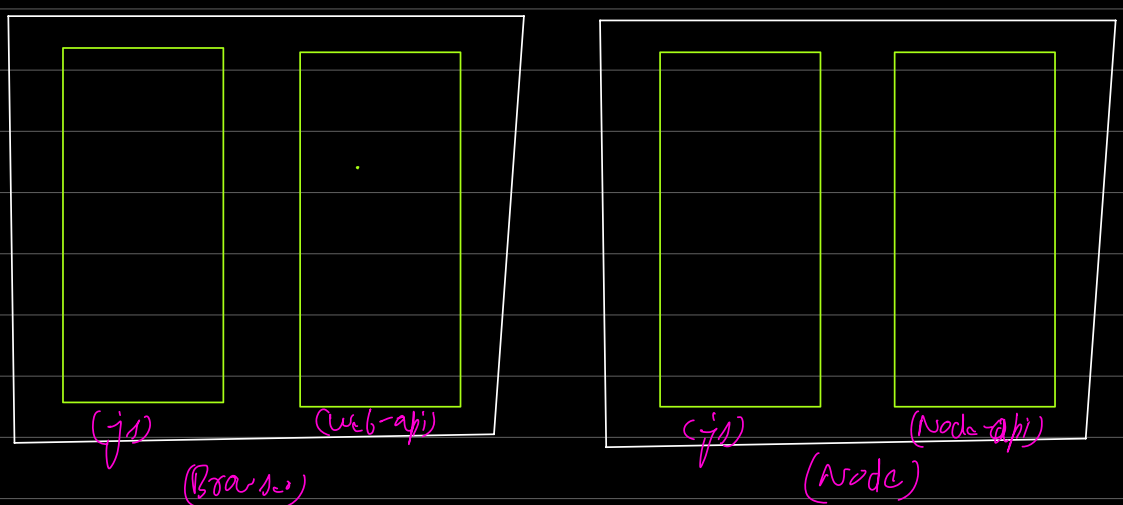
# Agenda

1. Intution behind the async programming.
2. Async js with call back
3. all about event loop with call back queues.
4. series and parallel execution of Async code.
5. Timer: set Interval, set time out, clear time out.
6. polyfill of set interval and clear interval.





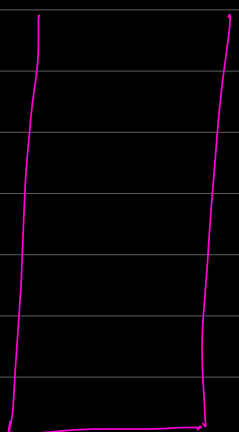

## Features

1. document
2. fetch
3. event listner
4. (I/O)

1. fs
2. http
3. child process

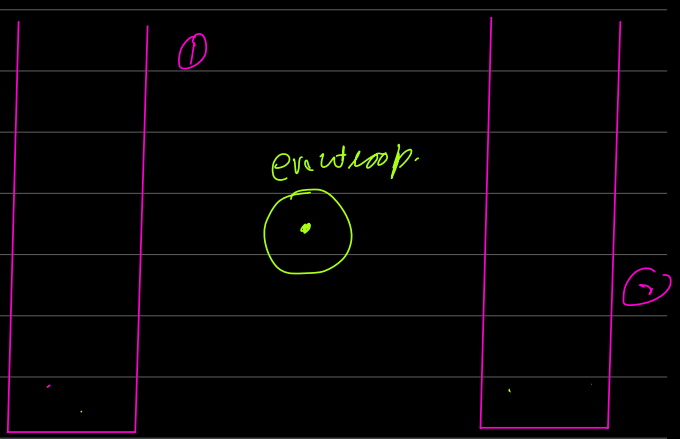
→ Javascript provide logic and features provide by environment.

→ (fyi): React native → mobile app.
Electron → desktop app.



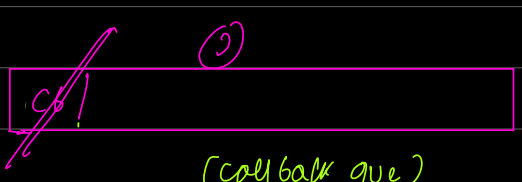
JS-stack (main stack)

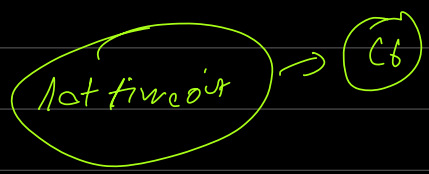
```
console.log("Before");

function fn(){
    console.log("I'm in function.");
}
setTimeout(fn, 2000);

console.log("After");
```

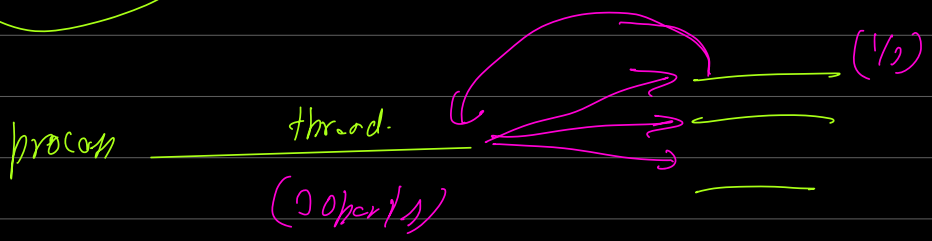



Lifo → Last in first out.

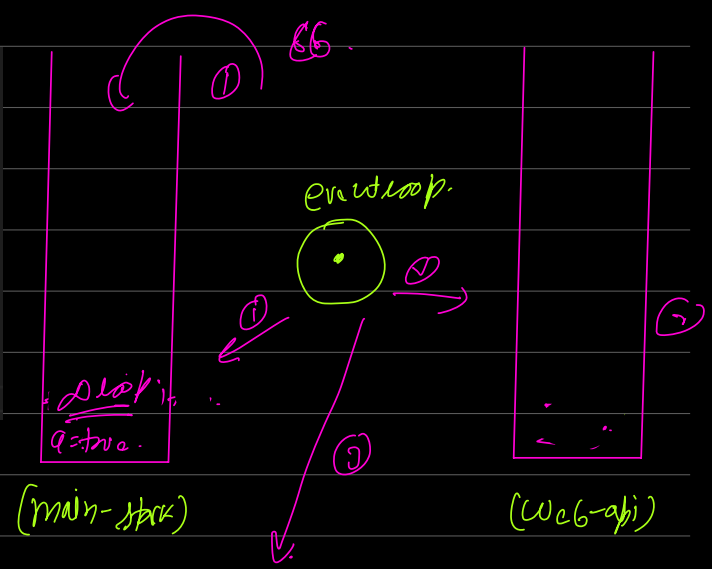
Before
After.
I'm in function.



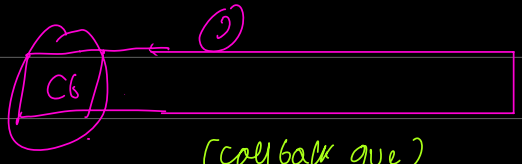


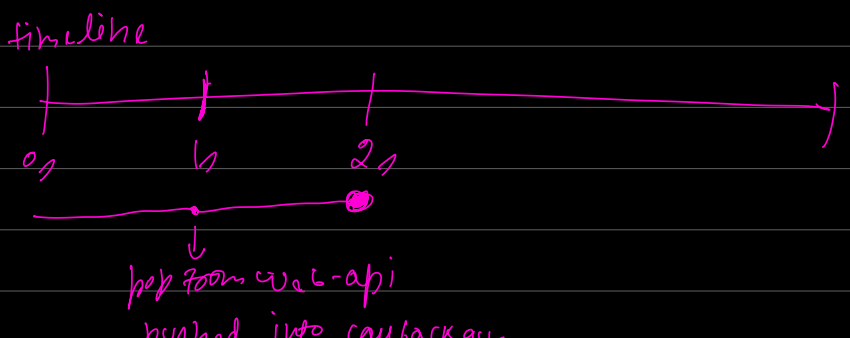




```
let a = true;
console.log("Before: ", a);

setTimeout(() => {
    a = false;
    console.log("I will broke the while loop: ", a);
}, 1000);

// wait for 2sec in series operation.
let timeFuture = Date.now() + 2000;
while(Date.now() < timeFuture){}
console.log("After: ", a);

while (a) { }
```

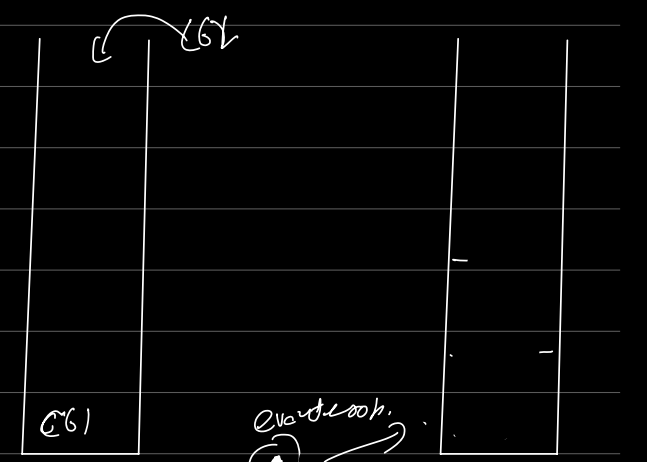



Before : true
After : true
I will broke.... : false.



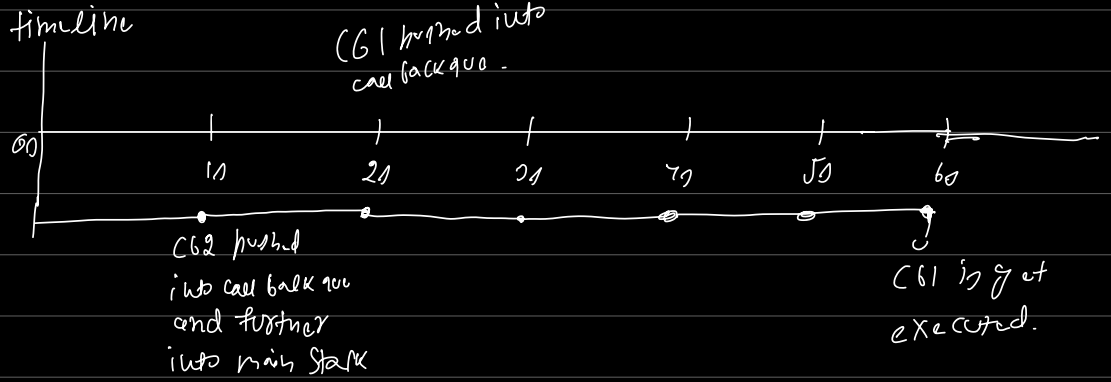



```
console.log("Before");
const cb2 = () => {
    console.log("set timeout 1");
    let timeInfuture = Date.now() + 5000;

    console.log("Before while loop: ", Date.now());
    while (Date.now() < timeInfuture) {
    }
    console.log("After while loop: ", Date.now());

}
const cb1 = () => console.log("hello");
setTimeout(cb2, 1000)

setTimeout(cb1, 2000)

console.log("After");
```




log
Before
After
set timeout
Before while loop: ----
After while loop: ----

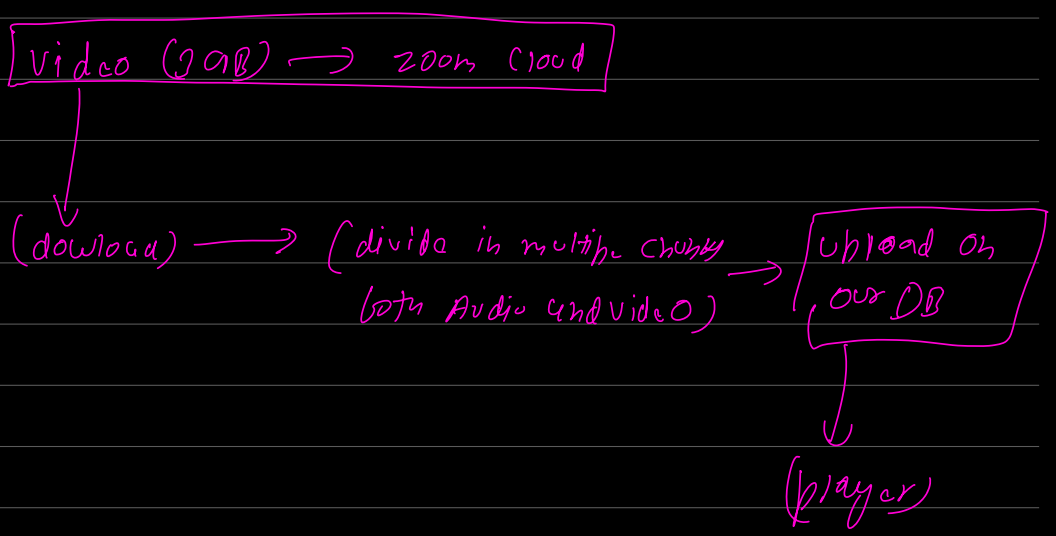
(callback que)

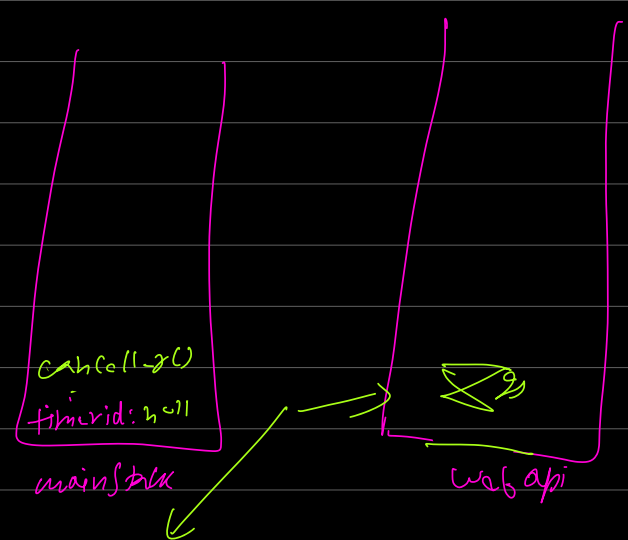



total time : (6+.0) s

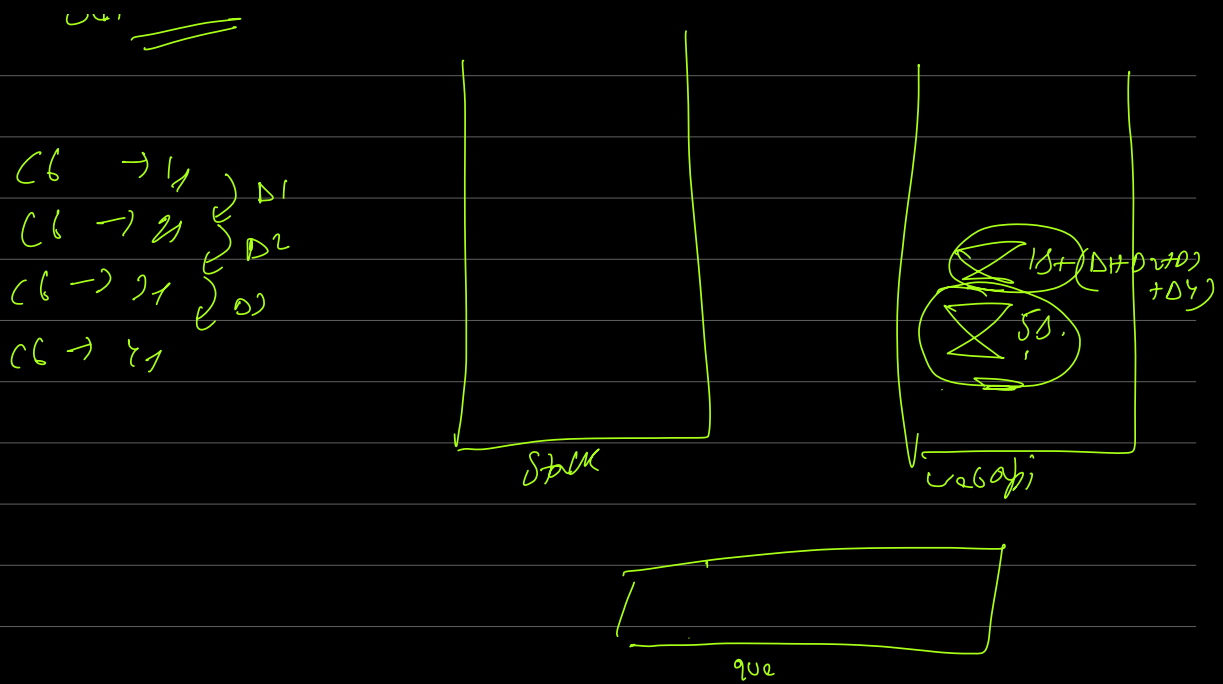
(Serial & parallel task)

①




Set interval


Stack
que