Event loop → Continuous process

In Javascript, enables handling of Asynchronous operations and events within a single threaded execution model.

Responsive, Non Blocking JS code.

Web APIs → Provided by → Runtime Environment
→ Include Asynchronous Functions
→ setTimeout
→ Fetch API
→ DOM Events

→ When function → called → Initiate → Asynchronous ops
→ immediately → return → allowing → main thread → continue → execute other code.

Call Stack → keeps track of function are currently executing.
→ When function called → added to → Call Stack
→ When function completed → removed from → Call stack
→ Follows LIFO Principle.

CallBack queque / Task / MacroTask queque.

When → Asynchronous ops complete / Event occur → Callback associated with ops → placed in → Task queque.

MicroTask queque
→ Seperate queque
→ Handles Smaller & High Priority Asynchronous Tasks.
→ Tasks related to → Promises
→ JS APIs
→ executed with high priority than tasks from callback queque.

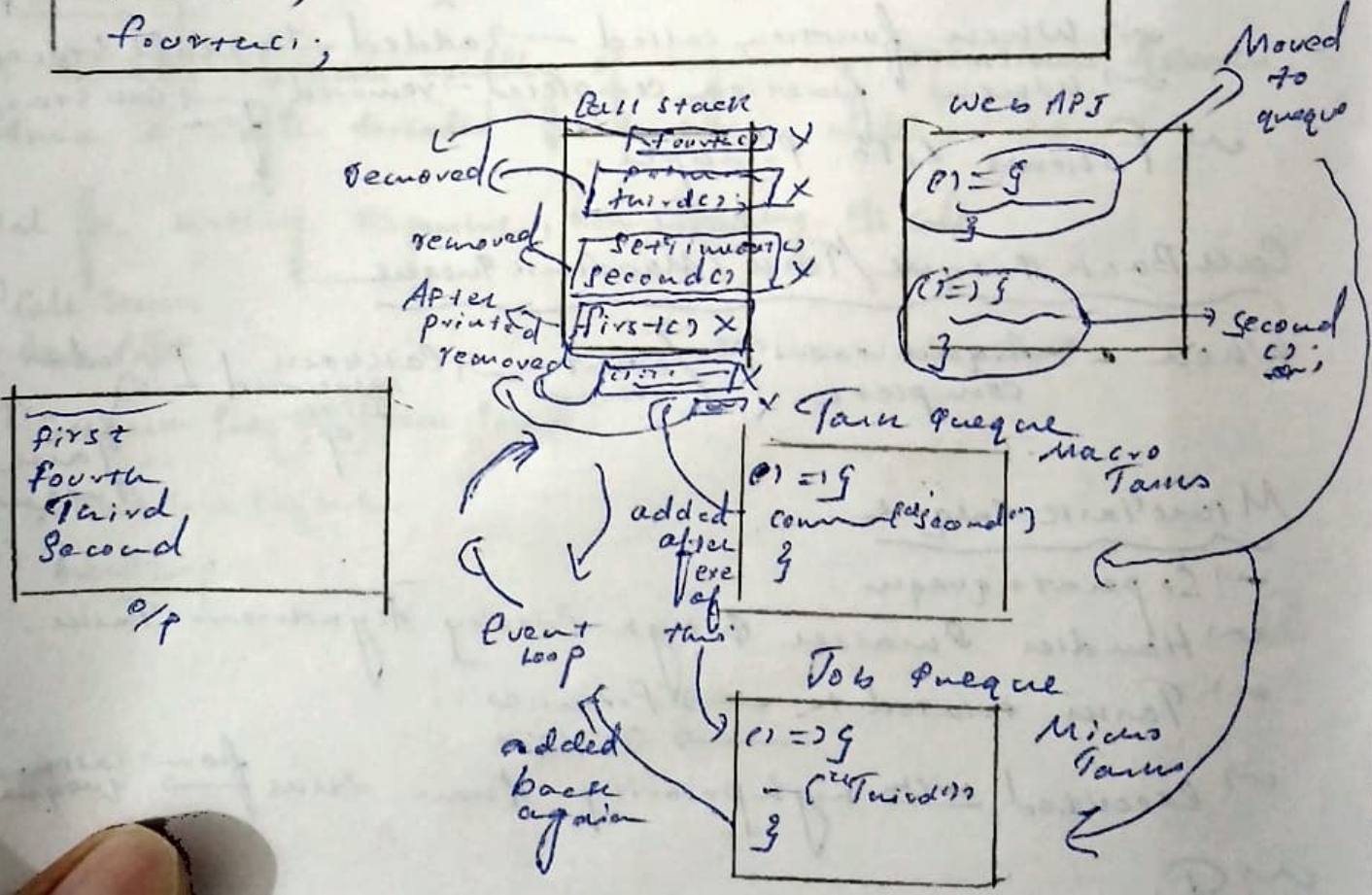
→ Tasks
→ Process Microtasks
→ Check call stack
→ Process MacroTasks

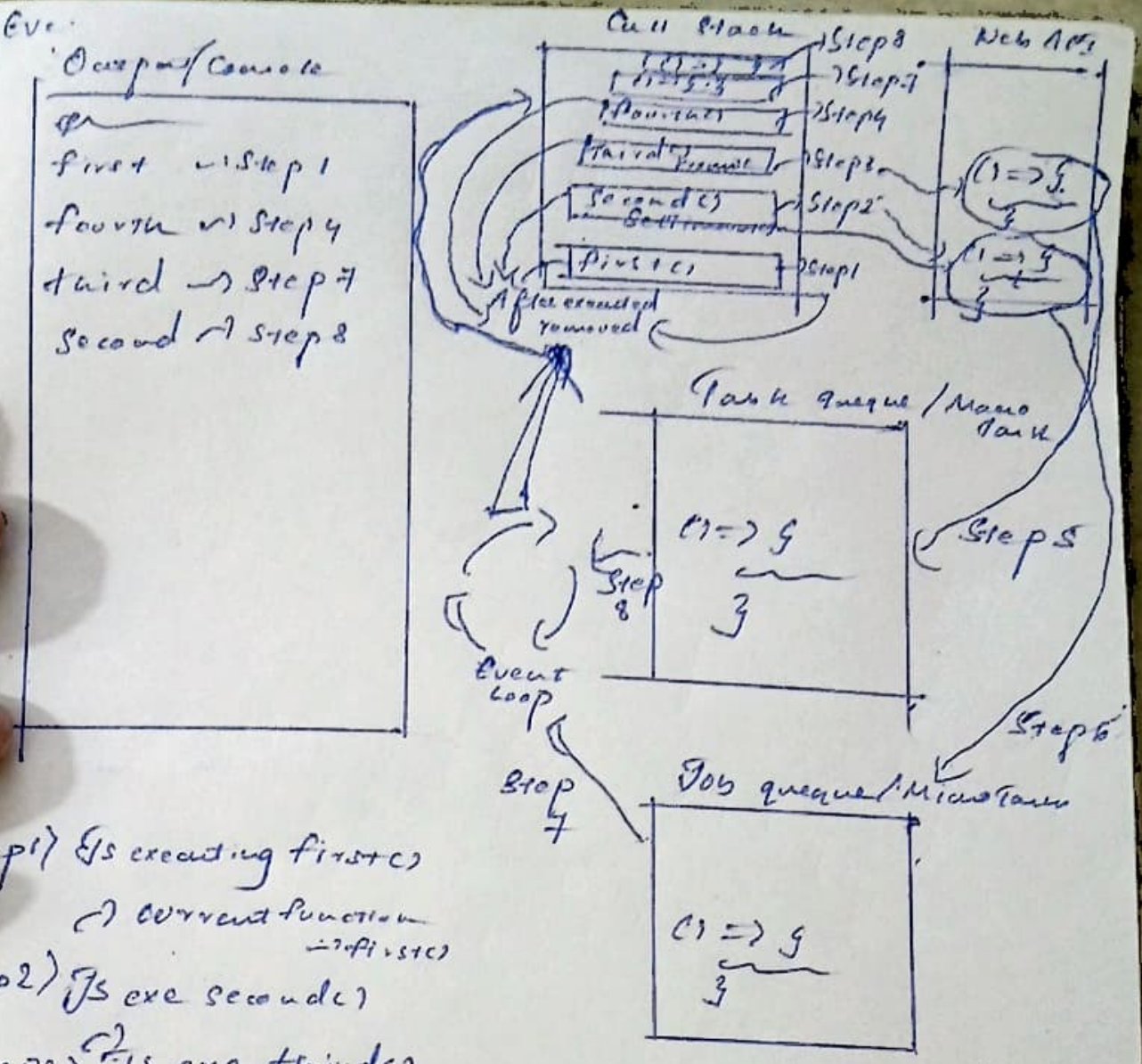
Eg.

```
function first(){
    console.log("First");
}

function second(){
    setTimeout(() => {
        console.log("second");
    }, 0);
}

function third(){
    Promise.resolve().then(() => {
        console.log("third");
    })
}

function fourth(){
    console.log("fourth")
}

first();
second();
third();
fourth();
```

1) Normal function

2) Anonymous with Asynchronous function





Step1) JS creating first()
→ current function → first()

Step2) JS exe second()

Step3) JS exe third()

Step4) JS exe fourth()

After all Call Stack is empty

Step5) Function second() in Web APIs is moved to Task queue.

Step6) Function third() in Web APIs is moved to Job queue (Promise)

Web APIs all empty

Step 7) third() is moved to call Stack executed & removed

Step 8) same for second()