Bhai 😎 chalo ab **Chapter 3: Event Loop** ko ekdum **desi style** me samajhte hain.

**1. Concept – Event Loop kya hota hai?**

Socho tum ek **dhaba (Node.js)** chalate ho 🍲.

* Ek waiter (single-thread) hai jo **order leta hai aur serve karta hai**.
* Waiter ek time par ek hi order serve kar sakta hai, par uske paas ek **register (Event Loop)** hai jisme sare pending orders likhe jaate hain.

⚡ Matlab:

* Node.js ek **single-threaded** system hai.
* Lekin thanks to **Event Loop**, ye ek sath bohot saare kaam handle kar leta hai bina atke.

**2. Kaise kaam karta hai?**

Event Loop ka process roughly aisa hota hai:

1. **Call Stack** → Code line by line execute hota hai (synchronous code).
2. **Node APIs** → Agar koi async kaam (setTimeout, fs.readFile, HTTP request) hota hai toh wo APIs handle karti hain.
3. **Callback Queue** → Jab async kaam complete ho jata hai, uska callback queue me store ho jata hai.
4. **Event Loop** → Call Stack khali hota hai toh Event Loop queue se callback uthata hai aur execute kar deta hai.

👉 Isi jugad se Node.js lagta hai jaise multi-tasking kar raha hai, lekin asal me ek hi thread hai.

**3. Code Example**

console.log("Start");

setTimeout(() => {

console.log("Bhai, 2 second baad ye aaya ⏱️");

}, 2000);

console.log("End");

**Output:**

Start

End

Bhai, 2 second baad ye aaya ⏱️

👉 Kyun?

* setTimeout async hai, isliye pehle **Node API** le gayi.
* Event Loop ne wait kiya.
* Jaise hi 2 sec pura hua, callback queue me gaya.
* Event Loop ne jab stack free dekha, tab print kara.

**4. Important Baatein**

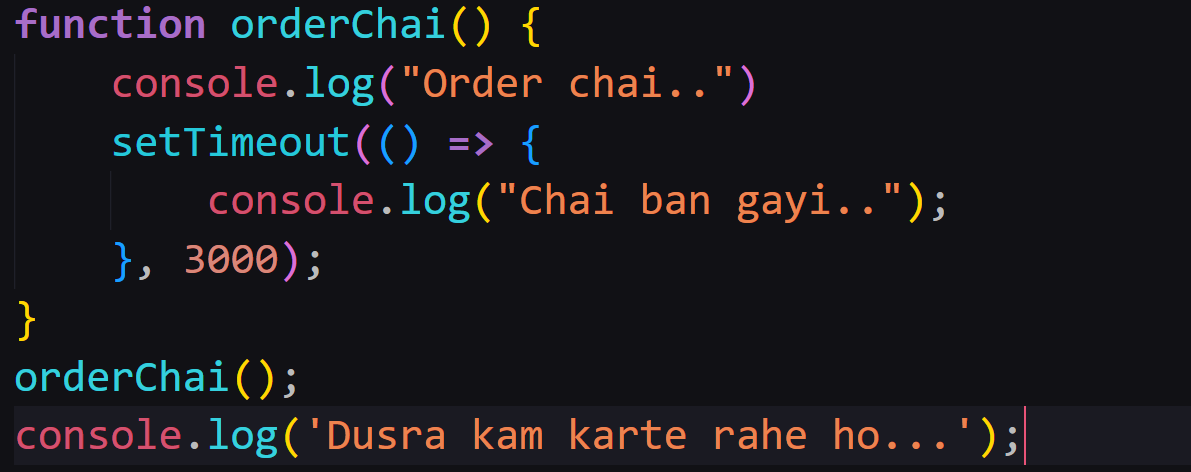
* Node.js = **Non-blocking I/O model** → matlab jab tak ek kaam chal raha hai, dusra block nahi hota.
* Event Loop hi reason hai ki Node.js fast aur scalable hai 🚀.

**5. Exercise Time 📝**

**Exercise 1:**

Ek program likho jisme:

* “Order chai” print ho
* 3 sec baad print ho: “Chai ban gayi ☕”
* Aur bina wait kare turant print ho: “Dusra kaam karte raho...”



**Exercise 2:**

Output predict karo:

console.log("1");

setTimeout(() => {

console.log("2");

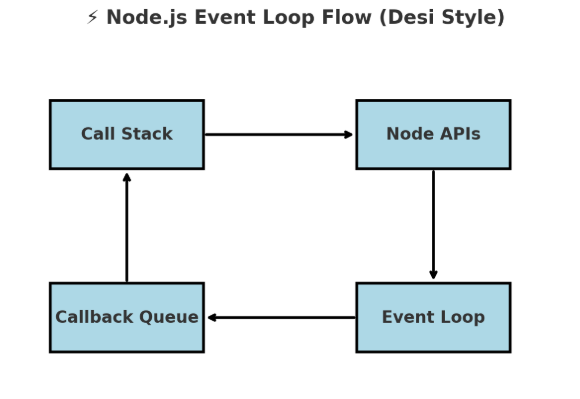
}, 0);

console.log("3");

**🎯 Desi Summary**

* **Event Loop** = Node.js ka waiter + register
* Call Stack → line by line execution
* Node APIs → async tasks ko sambhalna
* Callback Queue → pending work ready for execution
* Event Loop → bridge jo stack khali hote hi queue ke tasks chala deta hai

👉 Bhai, chahte ho main tumhare liye ek **diagram** bana kar Event Loop ka flow step-by-step samjhau (Call Stack → APIs → Queue → Event Loop)?



Bhai 😎 ye dekh — ye diagram hai **Event Loop ka asli flow**:

1. **Call Stack** → sabse pehle yaha code execute hota hai line by line.
2. Agar async kaam mila (setTimeout, fs, HTTP) → wo chala jata hai **Node APIs** me.
3. Kaam complete hote hi callback jata hai **Callback Queue** me.
4. **Event Loop** check karta hai → agar stack khali hai, toh callback uthake wapas **Call Stack** me daal deta hai execute karne ke liye.

👉 Isi jugad se Node.js **ek waiter se hi pura restaurant sambhal leta hai** 🚀

NB: Node APIs => setTimeout, fs, HTTP ye tino Node APIs hain.