### 1. Synchronous JavaScript (Sync)

- **Executes line by line.**
- One operation must complete before the next starts.
- If one task takes a long time, everything else has to wait.

**Example:**

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

function syncTask() {
  console.log("Doing a sync task...");
}

syncTask();
console.log("End");
```

**Output:**

```
Start
Doing a sync task...
End
```

**Explanation:** The code runs top to bottom. Nothing jumps ahead.

---

### 2. Asynchronous JavaScript (Async)

- **Doesn't block** the rest of the code.

- Often used for tasks like **API calls**, **setTimeout**, **reading files**, etc.
- JavaScript uses the **event loop** to manage async tasks.

**Example using** `setTimeout`**:**

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

setTimeout( ( ) => {
  console.log("Async task finished");
}, 2000);

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

**Output:**

```
Start
End
Async task finished
```

**Explanation:** `setTimeout` schedules the function to run after 2 seconds, but the rest of the code continues immediately.

---

**Async Patterns in JavaScript**

**1. Callbacks**

```
function getData(callback) {
  setTimeout(() => {
    callback("Here is your data");
  }, 1000);
}
```

```
getData((data) => {
  console.log(data);
});
```

**2. Promises**

```
function getData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Here is your data");
    }, 1000);
  });
}

getData().then(data => console.log(data));
```

**3. async/await**

```
async function fetchData() {
  let data = await getData();
  console.log(data);
}

fetchData();
```

---

**Summary**

| Type | Blocks Code? | Use Case Examples |
|---|---|---|
| Synchronous | Yes | Basic calculations |
| Asynchronous | No | API calls, timers, etc. |

---

**Example: A Restaurant Kitchen**

Imagine JavaScript is a **chef** in a small kitchen.

**Synchronous Cooking (Sync)**

- The chef does **one task at a time**.
- If someone orders pasta, the chef:
    1. Boils water.
    2. Waits until it's boiled.
    3. Cooks the pasta.
    4. Serves it.
- **Only after finishing one dish** can the chef start the next.

**Problem?** If boiling water takes 10 minutes, all other orders must wait.

**Asynchronous Cooking (Async)**

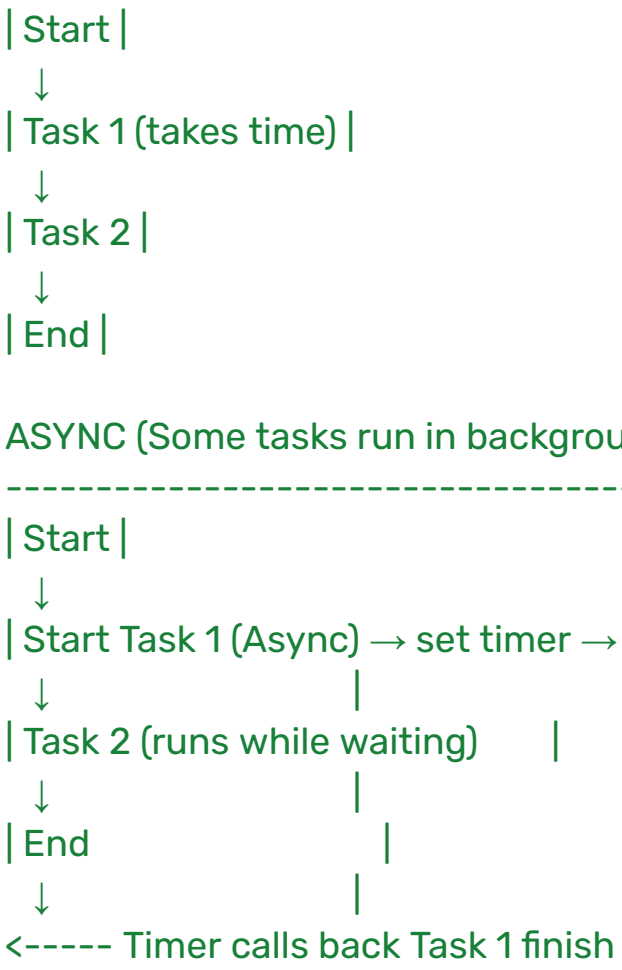
- The chef uses **a timer**.
- When boiling water, they **set a timer** and move to the next task.
- When the water is ready, the timer **notifies the chef** to return and continue.
- This way, **multiple dishes progress in parallel**.

---

**Visual Diagram (Conceptual)**

SYNCHRONOUS (One step at a time):

---




---

**Real-Life Code Version:**

```
// Sync
console.log("Wash vegetables");
console.log("Boil pasta"); // wait for boiling
console.log("Serve pasta");
```

**Sync Output:**

*Wash vegetables*
*Boil pasta*
*Serve pasta*

```
// Async
console.log("Wash vegetables");
setTimeout(() => {
  console.log("Boil pasta done");
}, 2000); // doesn't block
console.log("Serve pasta");
```

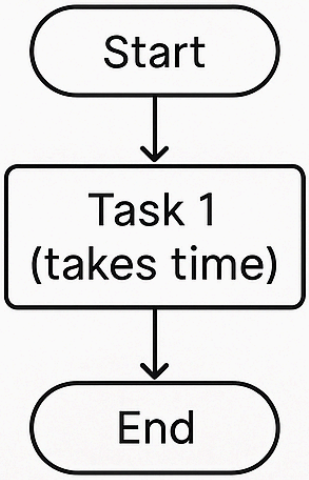
**Async Output:**

*Wash vegetables*
*Serve pasta*
*Boil pasta done*

**User List API Call**

- **async**:
  - The fetchUserData function is marked as async, which allows the use of await inside it.
- **await**:
  - The await keyword pauses the execution of the fetchUserData function until the promise returned by fetch() is resolved. It ensures that the response.json() is called only after the response is fetched.
- **Error Handling**:
  - The try/catch block is used to handle errors. If there's an issue with the network request (e.g., if the user is offline or the API is down), the catch block will handle that error gracefully.

## SYNCHRONOUS




---

## ASYNC