$$\begin{tikzcd}
\text{Callbacks} \arrow[d] \\
\text{Callback Hell} \arrow[d] \\
\text{Promises} \arrow[d] \\
\text{Promises Chaining} \arrow[d] \\
\text{Async Await}
\end{tikzcd}$$

Synchronous v/s Asynchronous:

- Synchronous code runs line by line. Each operation must complete before the next one starts.
- Asynchronous code can start a task and move on without waiting for it to finish.
- Asynchronous code execution allows to execute next instructions (code) immediately and doesn’t block the flow.

```
console.log("task 1");
console.log("task 2");
console.log("task 3");
```

```
console.log("Start");
setTimeout(() => {
    console.log("Async Task Done");
}, 2000);
console.log("End");
```

Don’t block the other tasks due to a single lengthy/long task.

```javascript
console.log("Hey guys..!! Do You Want Coffee??")

console.log("Muskan servers coffee");

setTimeout(() => {
    for (let i = 1; i <= 400000; i++) {
        console.log("Person", i, "Comes")
    }
}, 100);

console.log("Muskan is learning dance..!!")
```

Feature	Synchronous	Asynchronous
Execution Flow	Line by line	Skips long tasks, comes back
Blocking	Yes	No
Use Cases	Simple tasks, calculations	API calls, DB queries, timers

Why Do We Get a Promise Instead of Data? 🤔

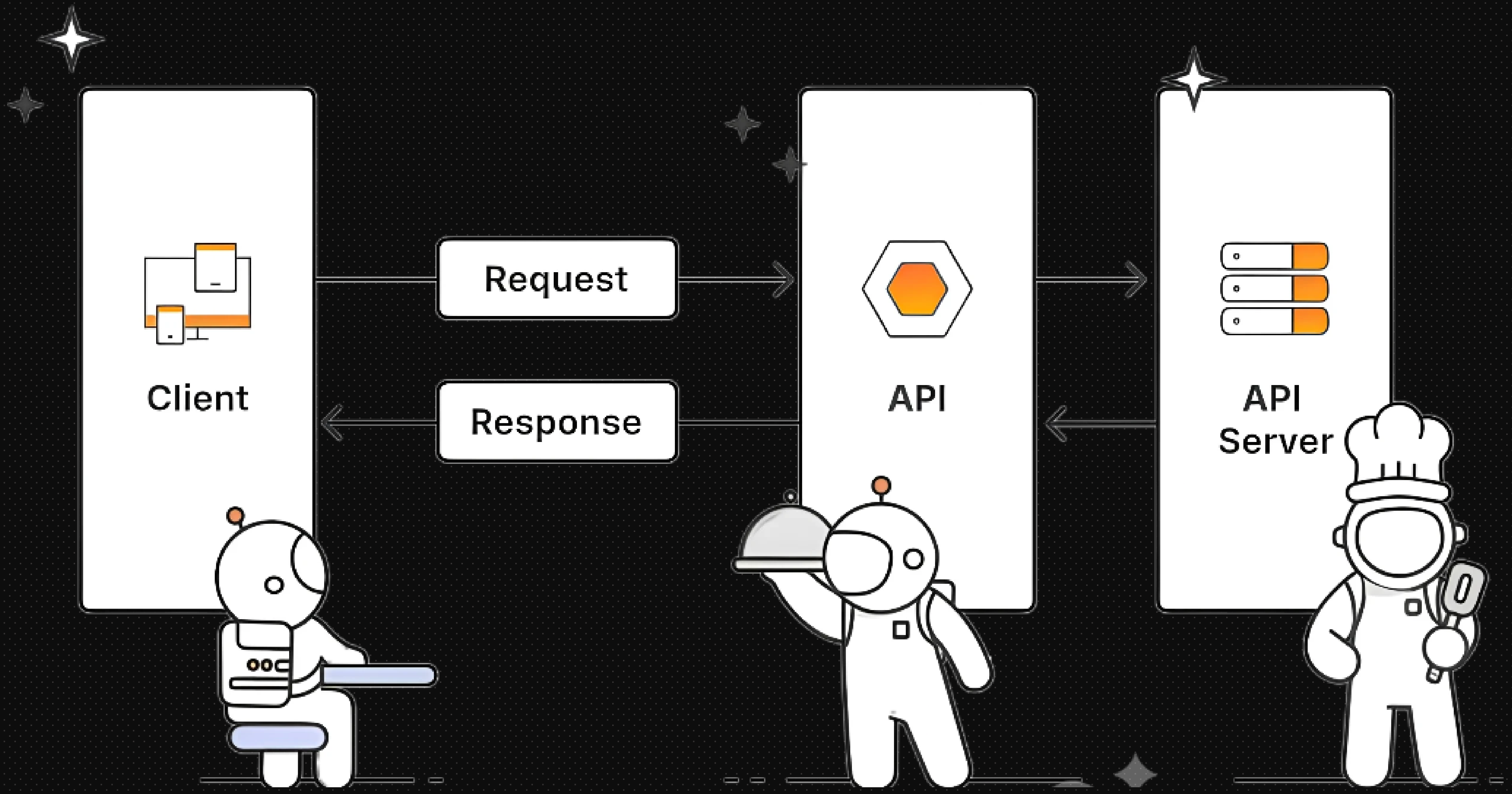
```
let data = fetch("https://jsonplaceholder.typicode.com/users");
console.log(data); // 👉 It logs a Promise, not actual data
```

You get a Promise — not the real data — because the data isn't ready yet.

API Calls Are Asynchronous

- ❑ Fetching data takes time (maybe 500ms, 2s, or more).
- ❑ JavaScript doesn't want to stop everything and wait (it’s single-threaded).
- ❑ So instead, it gives you a Promise, saying:

  “I'll give you the data later, once it arrives.”


Client
Request
Response
API
API Server

Let’s Build A Project:

fetch

- ❑ fetch is a built-in JavaScript function used to make HTTP requests (like GET, POST) to a server or API. (It is like “Hey server, please give me some data!”)

What is CRUD?

- ❑ CRUD stands for:
    - Create
    - Read
    - Update
    - Delete
- ❑ These are the 4 basic operations we perform on data.

CRUD Operation	HTTP Method	Purpose
Create	POST	Add new data
Read	GET	Get/fetch existing data
Update	PUT / PATCH	Modify existing data
Delete	DELETE	Remove existing data

- JavaScript is single-threaded. That means it does one thing at a time.
- Suppose you want to fetch user data from a server. It takes 2 seconds. If we wait normally, the whole app freezes. Users can’t click or scroll.

$$\begin{tikzcd}
\textbf{Callbacks} \arrow[d] \\
\textbf{Promises} \arrow[d] \\
\textbf{Async Await}
\end{tikzcd}$$

Callbacks:

- ❑ A Callback is a function passed as an argument to another function

```javascript
console.log("1. Start fetching data...");

function fetchData(callback) {
    setTimeout(() => {
        console.log("2. Data fetched from server");
        callback(); // run the callback after data is fetched
    }, 3000);
}

function processData() {
    console.log("3. Now processing the data...");
}

fetchData(processData);

console.log("4. Do other things while waiting...");
```

- ❑ Callbacks help us deal with tasks that take time, like loading data from a server, without blocking other code from running.

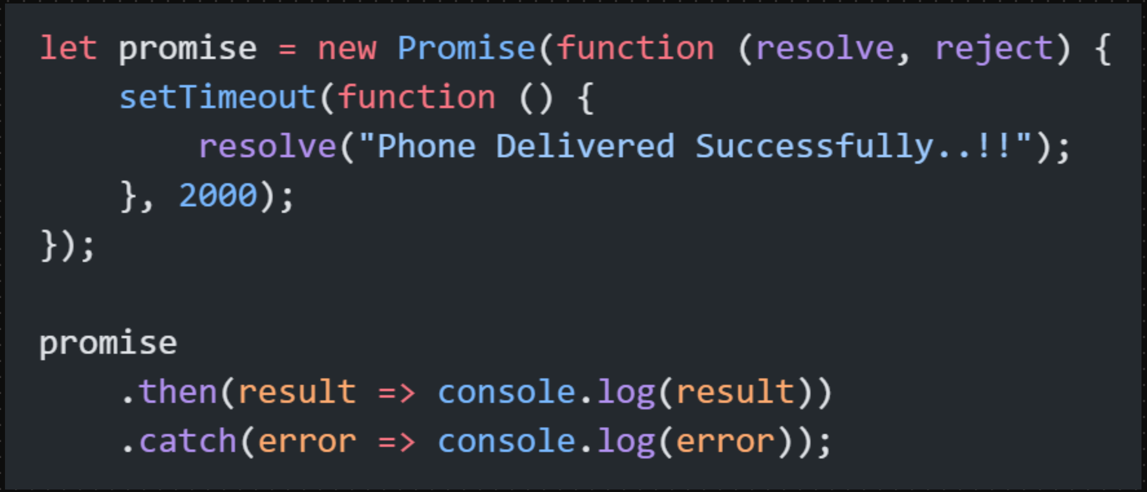
Callback Hell (Pyramid Of Doom):

❑ Callback Hell happens when you have many nested callbacks — one inside another — usually in asynchronous code.

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

setTimeout(() => {
    console.log("1. Getting user from database...");

    setTimeout(() => {
        console.log("2. Getting user's orders...");

        setTimeout(() => {
            console.log("3. Processing payment...");

            setTimeout(() => {
                console.log("4. Sending confirmation email...");
            }, 1000);

        }, 1000);

    }, 1000);

}, 1000);
```

Promises:

- ❑ A Promise is a special object in JavaScript that represents a task that will finish in the future.



```
let promise = new Promise(function (resolve, reject) {
    setTimeout(function () {
        resolve("Phone Delivered Successfully..!!");
    }, 2000);
});

promise
    .then(result => console.log(result))
    .catch(error => console.log(error));
```

- ❑ resolve and reject are callbacks provided by JavaScript.

- ❑ A promise has 3 states:
  - Pending – still waiting
  - Resolved (fulfilled) – task completed
  - Rejected – something went wrong

async await:

- async / await helps you write asynchronous code in a cleaner, more readable way — almost like it's synchronous.

```
async function getData() {
    try {
        const response = await fetch('https://api.example.com/data');
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.log(error);
    }
}
getData();
```

- Code outside the async function continues immediately.
- Code inside the async function pauses at await.

- async : Makes a function always return a Promise.
- await : Pauses inside an `async` function until the Promise is resolved.