upGrad ENTERPRISE

DOM Manipulation & Asynchronous Programming

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

Part 1: Selecting and Modifying the DOM

1.1 What is the DOM?

- **DOM (Document Object Model)** represents the structure of an HTML document as a tree of nodes.
- JavaScript can **access**, **traverse**, and **manipulate** these nodes to change how the page looks and behaves.

1.2 Selecting Elements

Method	Description	Exam
`document.getElementById('id')`	Selects one element by ID	`const el = document.getElementBy`
`document.getElementsByClassName('class')`	Returns a live HTMLCollection	`const items = document.getElementsBy`
`document.getElementsByTagName('tag')`	Returns elements by tag name	`const paras = document.getElementsBy`

Method	Description	Example
`document.querySelector('selector')`	Returns the **first** matching element	`const button = document.querySelector('.btn`
`document.querySelectorAll('selector')`	Returns **all** matching elements as a NodeList	`const links = document.querySelectorAll('a`

1.3 Modifying Elements

```
const title = document.querySelector('#title');
title.textContent = 'Hello, JavaScript!';
title.style.color = 'blue';
title.classList.add('highlight');
```

Common Properties/Methods:

- `.textContent`, `.innerHTML`, `.value`
- `.setAttribute()`, `.getAttribute()`
- `.classList.add()`, `.classList.remove()`
- `.appendChild()`, `.removeChild()`

Part 2: Event Listeners and Delegation

2.1 Adding Event Listeners

```
const btn = document.querySelector('#clickMe');
btn.addEventListener('click', () => {
  alert('Button clicked!');
});
```

- Syntax: `element.addEventListener(eventType, callback)`
- Common events: `click`, `input`, `submit`, `mouseover`, `keydown`

2.2 Removing Event Listeners

```
function sayHi() {
  console.log('Hi!');
}
btn.addEventListener('click', sayHi);
btn.removeEventListener('click', sayHi);
```

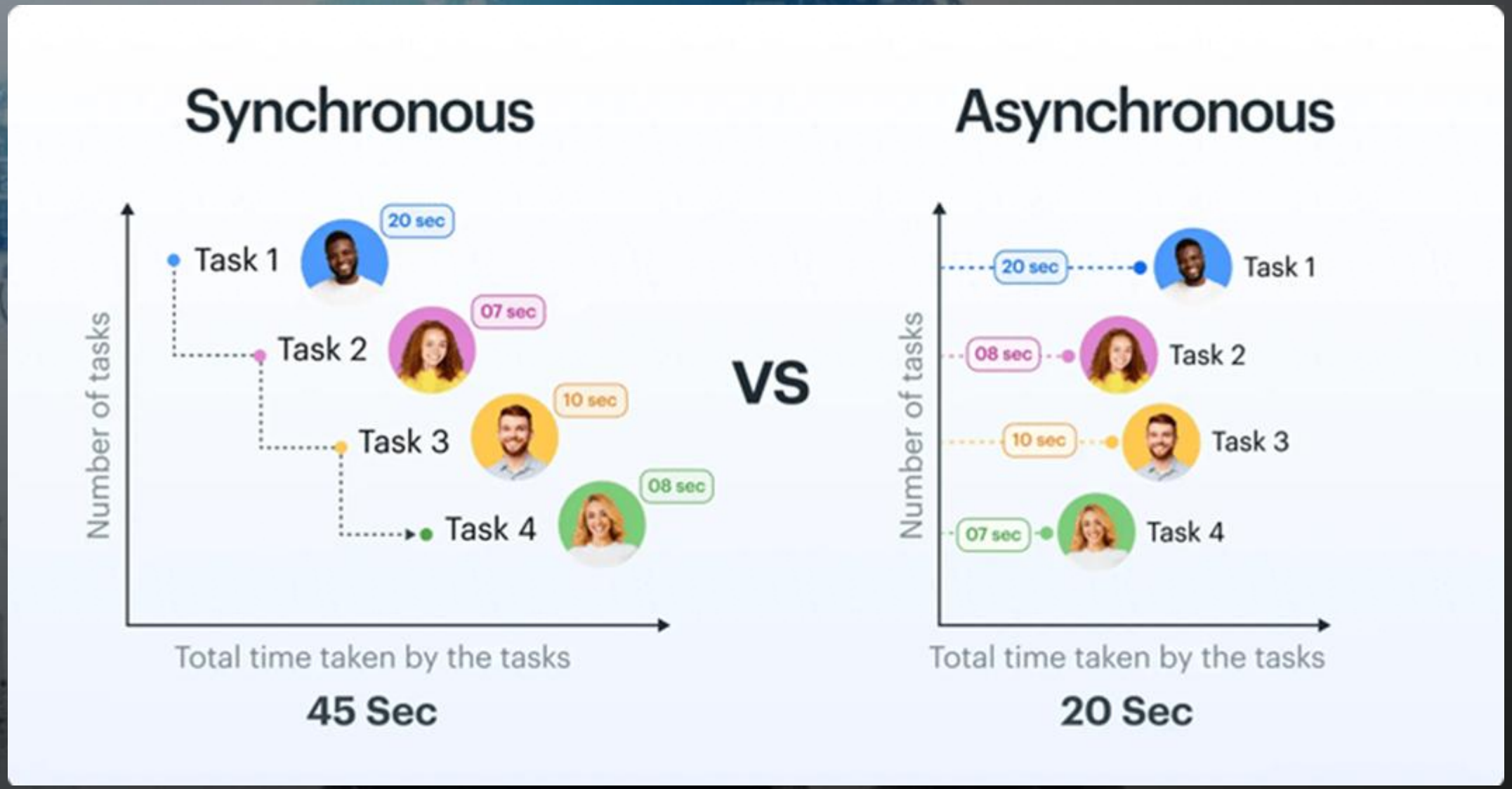
2.3 Event Delegation

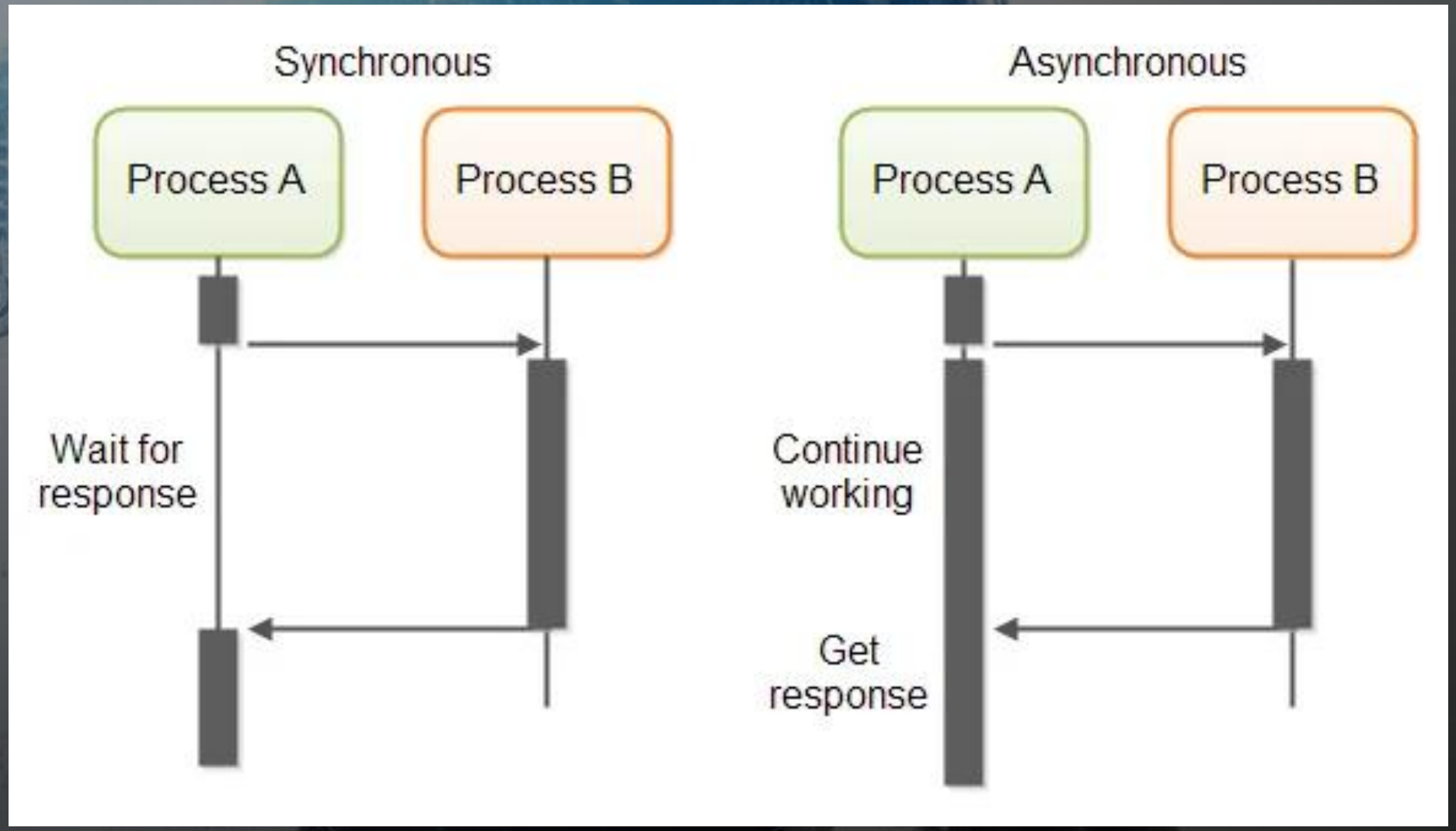
Instead of adding listeners to multiple elements, add one to a parent and use **event bubbling** to handle child actions.

```
document.querySelector('#list').addEventListener('click', (e) => {
  if (e.target.matches('li')) {
    e.target.classList.toggle('done');
  }
});
```

upGrad ENTERPRISE

Asynchronous Programming


Synchronous
Task 1
20 sec
Task 2
07 sec
Task 3
10 sec
Task 4
08 sec
Number of tasks
Total time taken by the tasks
45 Sec
VS
Asynchronous
20 sec
Task 1
08 sec
Task 2
10 sec
Task 3
07 sec
Task 4
Number of tasks
Total time taken by the tasks
20 Sec


Synchronous
Process A
Process B
Wait for response
Asynchronous
Process A
Process B
Continue working
Get response

Part 3: Asynchronous JavaScript

3.1 The Problem: Blocking Code

JavaScript is single-threaded — long operations (like network requests) block others.
To fix this, JS uses **callbacks**, **promises**, and **async/await**.

Part 4: Callbacks

4.1 What are Callbacks?

A **callback** is a function passed as an argument to another function, executed later.

```
function greet(name, callback) {
  console.log(`Hello, ${name}`);
  callback();
}

function afterGreet() {
  console.log('Welcome to the site!');
}

greet('Alice', afterGreet);
```

Part 5: Promises

5.1 What is a Promise?

A **Promise** represents a value that will be available *now, later, or never*.

```
const promise = new Promise((resolve, reject) => {
  const success = true;
  success ? resolve('Done!') : reject('Error!');
});

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

Part 6: Async/Await

6.1 What Is It?

Syntactic sugar for Promises — makes async code look synchronous.

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

6.2 Rules

- Must be inside an `async` function.
- `await` pauses execution until the Promise resolves or rejects.

Part 7: Fetch API

7.1 Fetch Basics

The **Fetch API** is used for HTTP requests — returns a Promise.

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

7.2 With Async/Await

```
async function getPosts() {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts');
  const posts = await res.json();
  console.log(posts);
}
getPosts();
```

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com