Presentation #4

ASYNCHRONOUS PROGRAMMING IN NODE JS

Mastering Callbacks, Promises, and Async/Await

Github Organization

CREATED BY

KETI ELIZBARASHVILI

Agenda

. Introduction
- Objective: Set the stage for the presentation and introduce the topic of asynchronous programming in Node.js.
- Key Points:
  - Briefly introduce Node.js and its non-blocking nature.
  - Outline the importance of understanding asynchronous patterns.

2. Understanding Callbacks
- Objective: Introduce and explain the concept of callbacks in Node.js.
- Key Points:
  - Define what callbacks are and their role in asynchronous operations.
  - Discuss the pros and cons of using callbacks, including the concept of "Callback Hell."

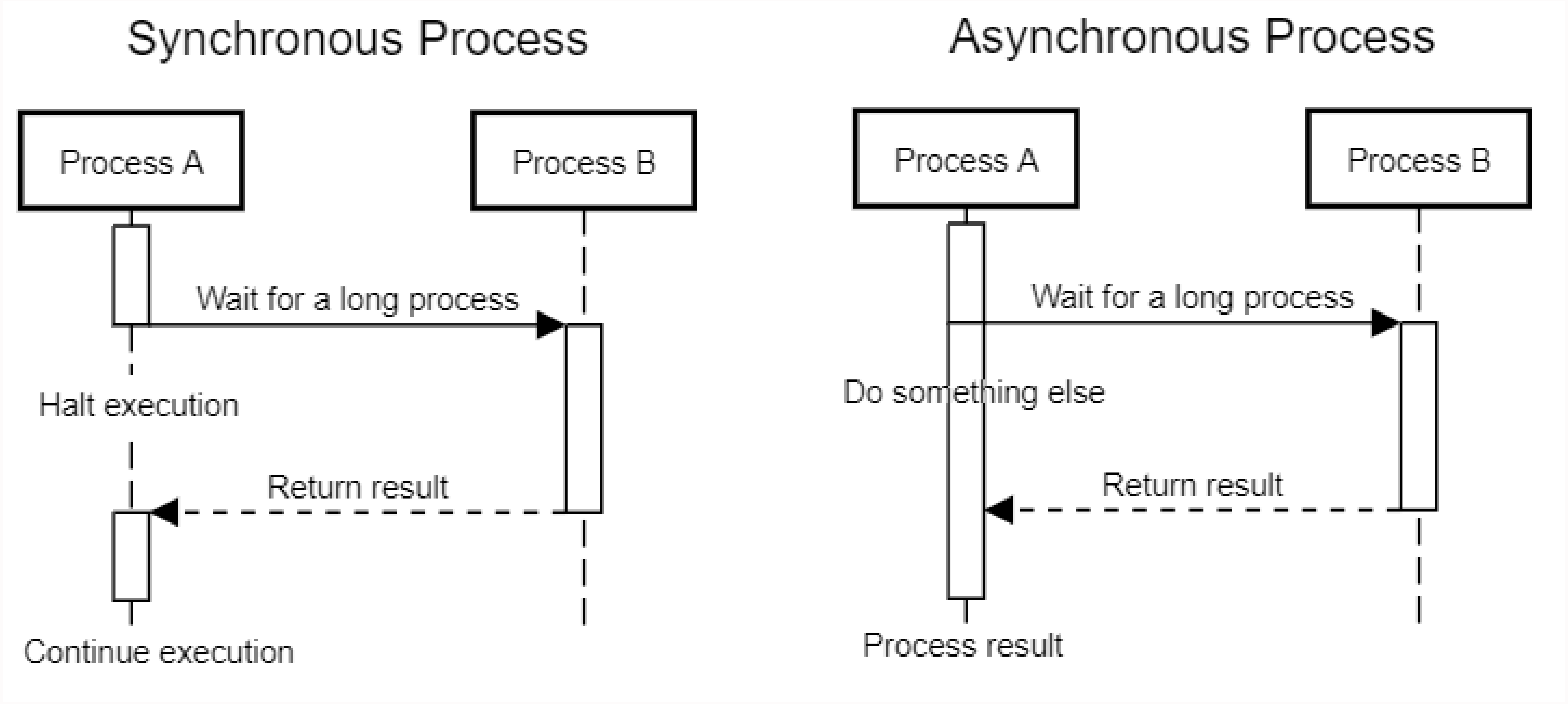
3. Exploring Promises
- Objective: Dive into the world of Promises and how they improve handling asynchronous operations.
- Key Points:
  - Define Promises and their advantages over callbacks.
  - Walk through a code example converting a callback pattern to Promises.

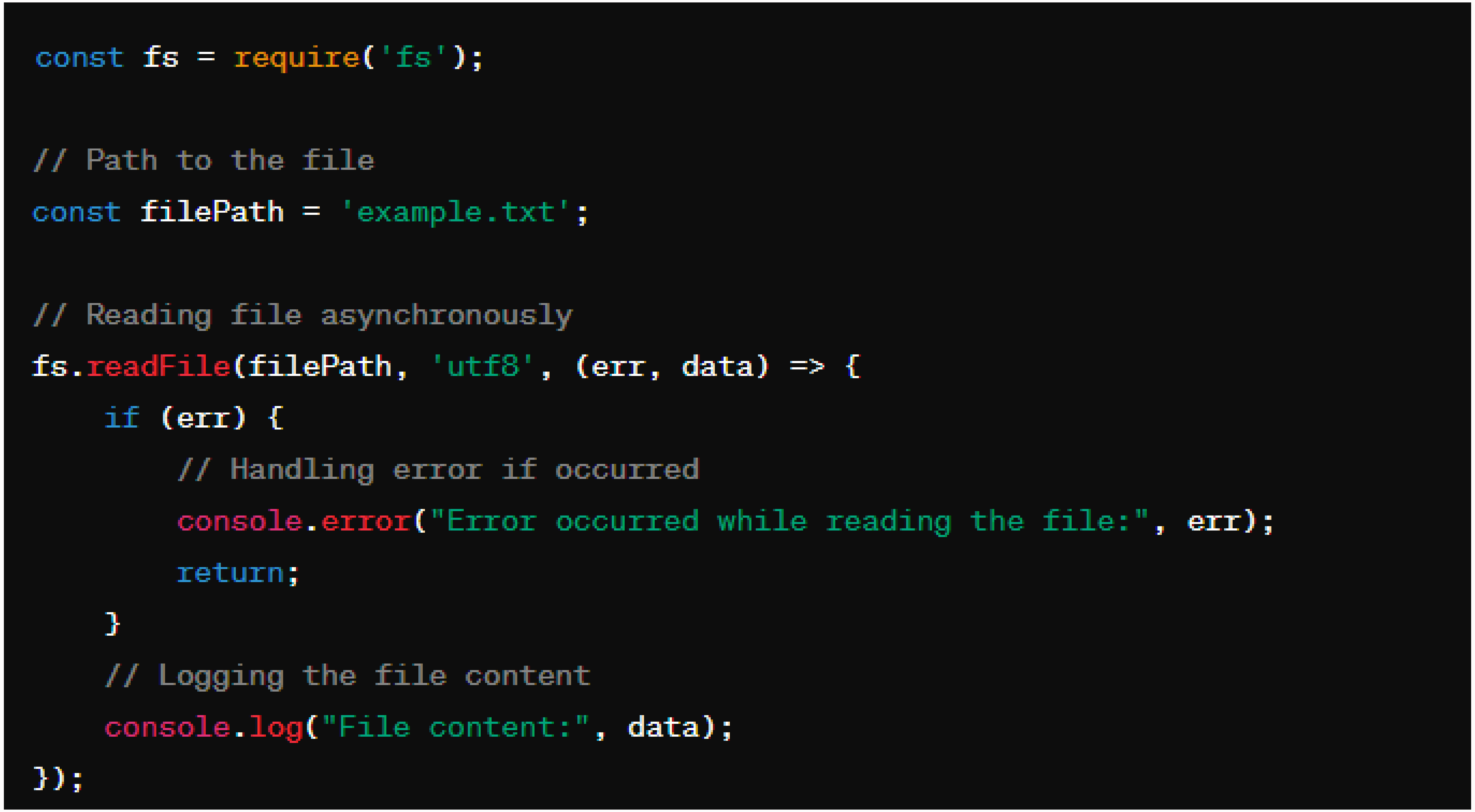
4. Simplifying with Async/Await
- Objective: Introduce async/await as a cleaner syntax for working with Promises.
- Key Points:
  - Explain how async/await simplifies asynchronous code.
  - Showcase a code example refactoring a Promise-based solution to use async/await.

5. Comparative Analysis
- Objective: Compare and contrast callbacks, promises, and async/await.
- Key Points:
  - Present a side-by-side comparison of handling an asynchronous task with each method.
  - Discuss when to use each approach effectively.

6. Best Practices and Common Pitfalls
- Objective: Share best practices and caution against common mistakes in asynchronous programming.
- Key Points:
  - Highlight effective practices for writing clean, maintainable asynchronous code.
  - Point out common pitfalls and how to avoid them.

7. Conclusion and Q&A
- Objective: Summarize the presentation and engage with the audience through questions.
- Key Points:
  - Recap the main takeaways about asynchronous programming in Node.js.
  - Open the floor for questions and further discussion.

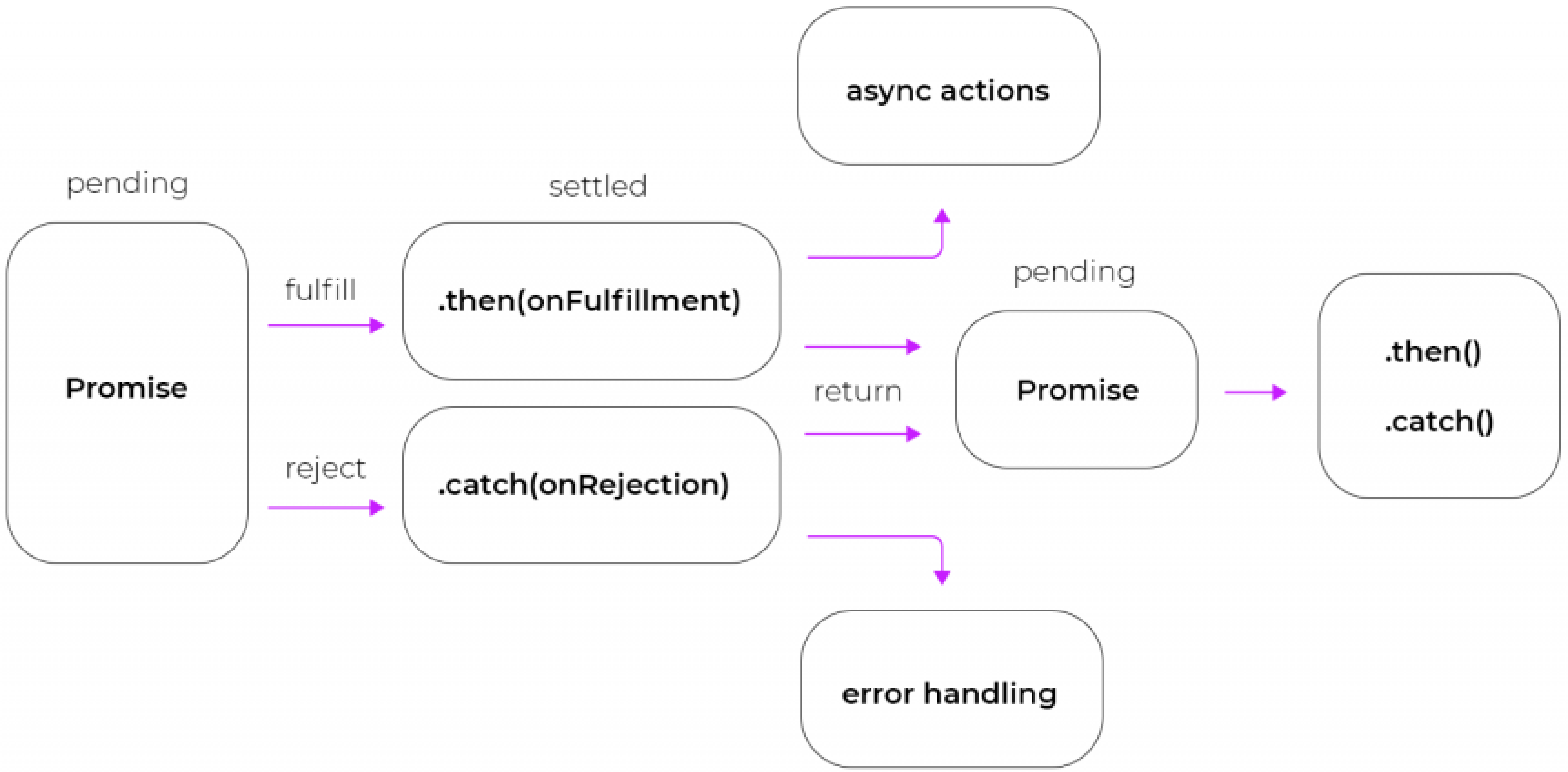

Synchronous Process
Process A
Process B
Wait for a long process
Halt execution
Return result
Continue execution
Asynchronous Process
Process A
Process B
Wait for a long process
Do something else
Return result
Process result

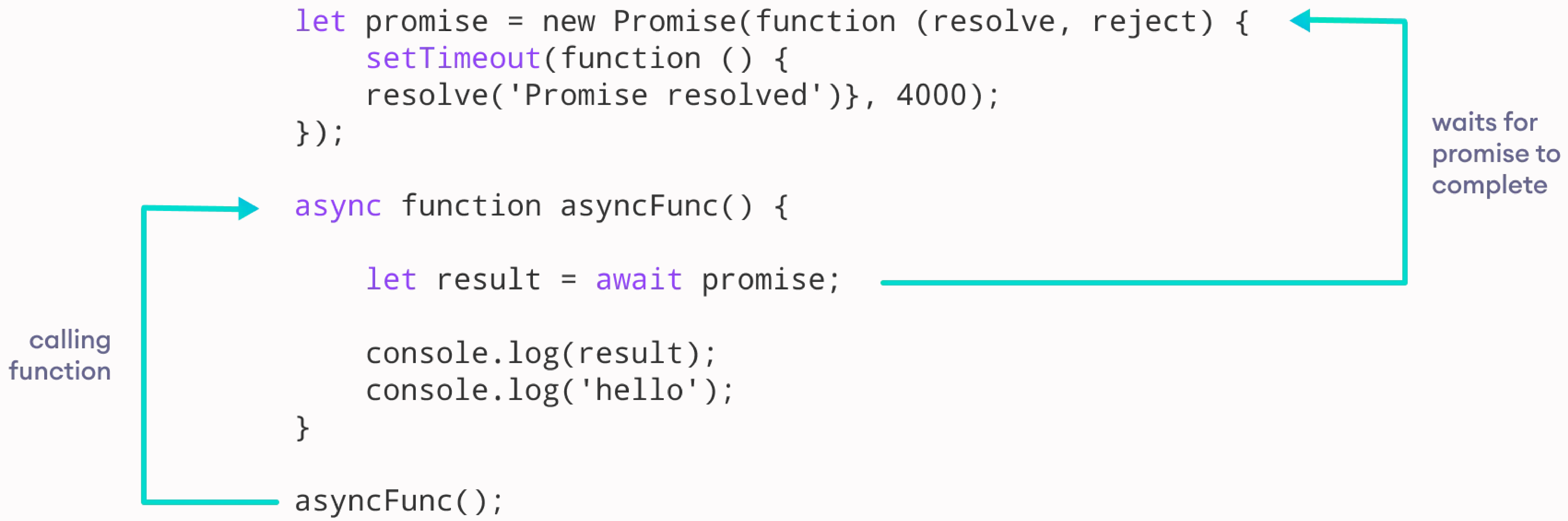


```javascript
const fs = require('fs');

// Path to the file
const filePath = 'example.txt';

// Reading file asynchronously
fs.readFile(filePath, 'utf8', (err, data) => {
    if (err) {
        // Handling error if occurred
        console.error("Error occurred while reading the file:", err);
        return;
    }
    // Logging the file content
    console.log("File content:", data);
});
```


async actions
pending
settled
Promise
fulfill
.then(onFulfillment)
reject
.catch(onRejection)
pending
return
Promise
.then()
.catch()
error handling



```
let promise = new Promise(function (resolve, reject) {
    setTimeout(function () {
    resolve('Promise resolved')}, 4000);
});

async function asyncFunc() {

    let result = await promise;

    console.log(result);
    console.log('hello');
}

asyncFunc();
```

Practical Work