Software Engineering

Session 4

Today

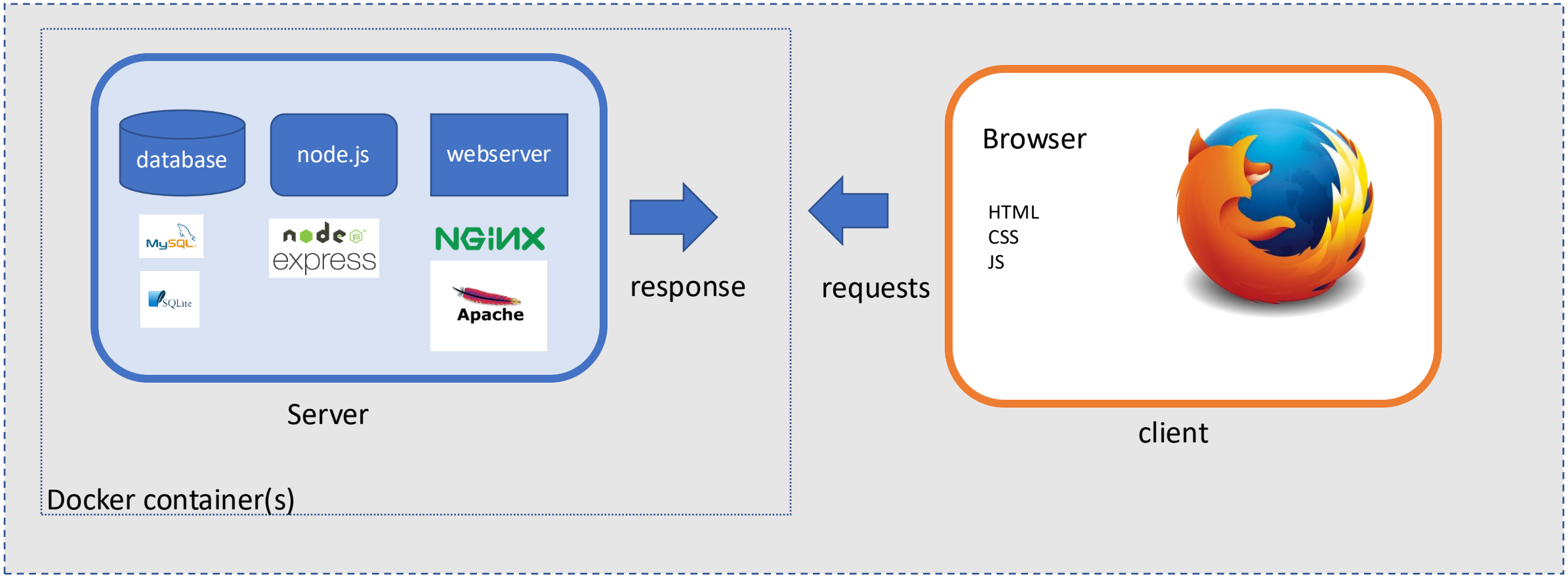
Lab

- Follow up from previous weeks – Docker, Express
- Import a database
- Write code that retrieves values from the database and uses the values to create simple HTML pages *(next week we will improve on this)*
- Use dynamic routes to get variable values from the URL, use these in your queries to create dynamic pages.
- **Understand asynchronous programming in JavaScript**

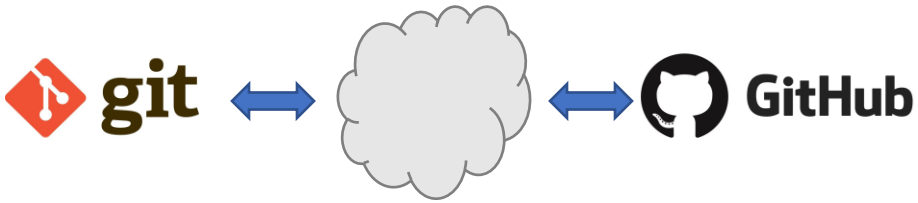
Seminar

- Sprint 1 review meetings

Development environments: When projects are in development, developers will have all the client and server software running on their own computer. Additionally they will use tools such as VS Code and Git to write and manage source code.



Visual studio code (IDE)



Recap - Express.js dynamic routes

Route parameters

Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the `req.params` object, with the name of the route parameter specified in the path as their respective keys.

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

To define routes with route parameters, simply specify the route parameters in the path of the route as shown below.

```
app.get('/users/:userId/books/:bookId', (req, res) => {
  res.send(req.params)
})
```

- Route parameters allow you to build fully dynamic web applications

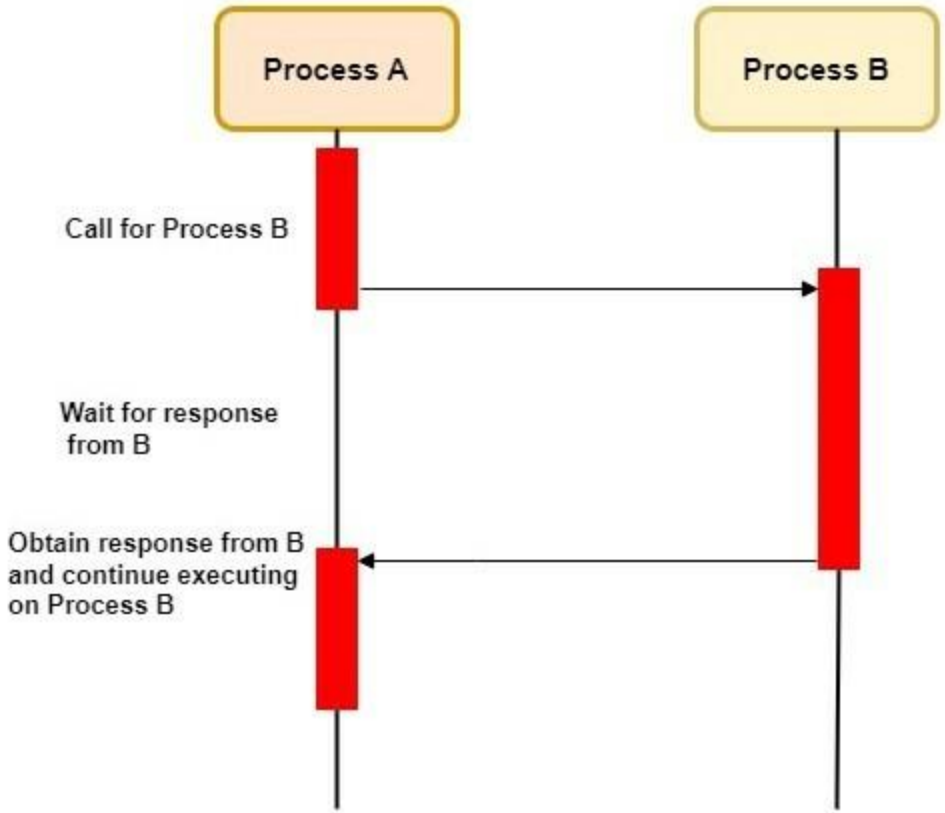
Lab today

- Make sure you can run Docker from the scaffolding files
- Make sure you can access routes that you create in Express
- Import the database from the lab sheet
- Write code that queries the database and uses data from the database to fulfill the user's request.
- *Understand the 'async' and 'await' keywords and then() blocks*
- Use dynamic routes to filter SQL queries

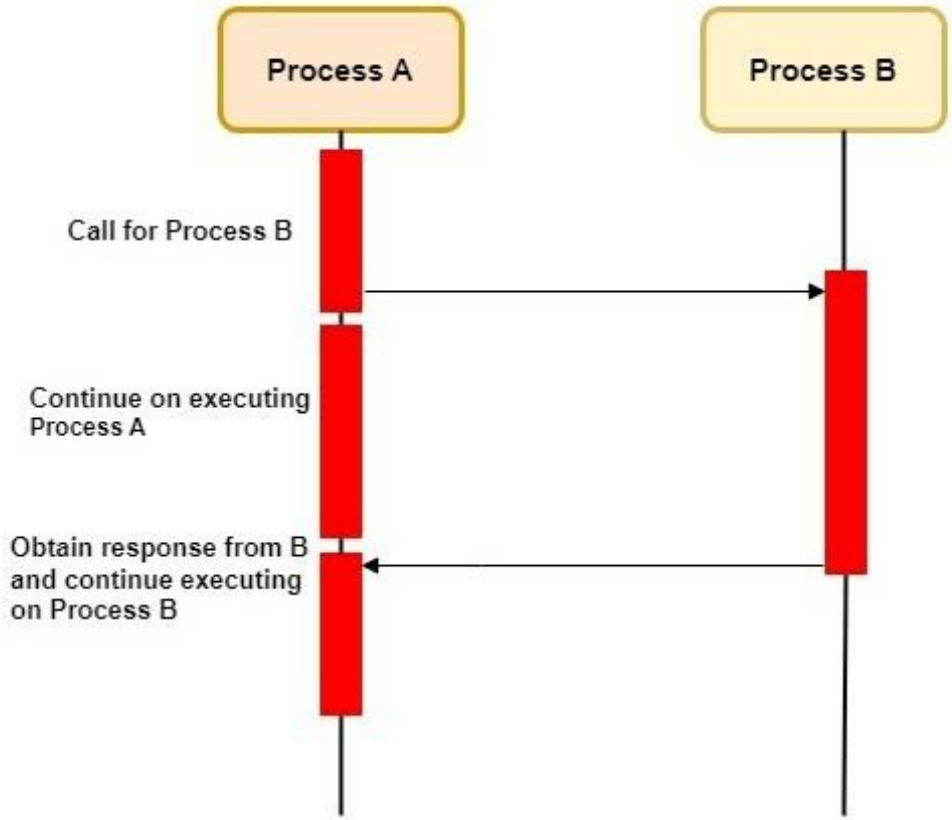
Asynchronous Programming

- **Asynchronous programming** is a programming paradigm that allows for the execution of multiple tasks or processes at the same time without waiting for each other to complete.
- In traditional synchronous programming, tasks are executed sequentially, one after the other, and each task must be completed before the next one starts. In contrast, asynchronous programming enables tasks to be initiated and executed independently of the main program flow.
- The key idea behind asynchronous programming is to handle tasks that may take some time to complete (***such as I/O operations, network requests, or file operations***) without blocking the execution of the entire program.

Synchronous Programming



Asynchronous Programming



Synchronous programming

Prons:

- Easier to implement
- Easier to debug
- More intuitive code flow

Cons:

- Blocking calls
- Lower resource utilization
- Slower performance

Asynchronous programming

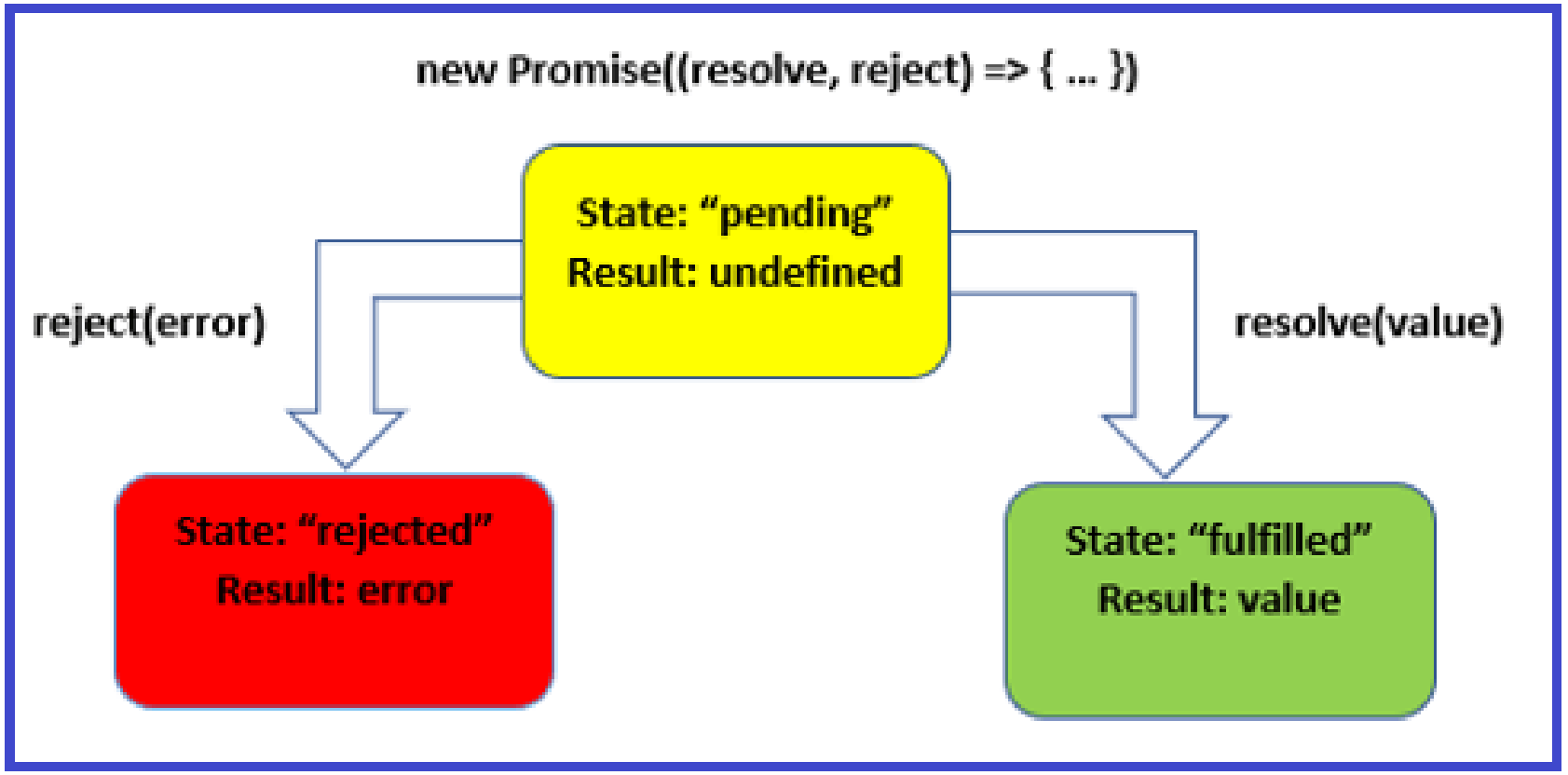
Prons:

- Improved Performance
- Better Scalability
- More Responsive UI

Cons:

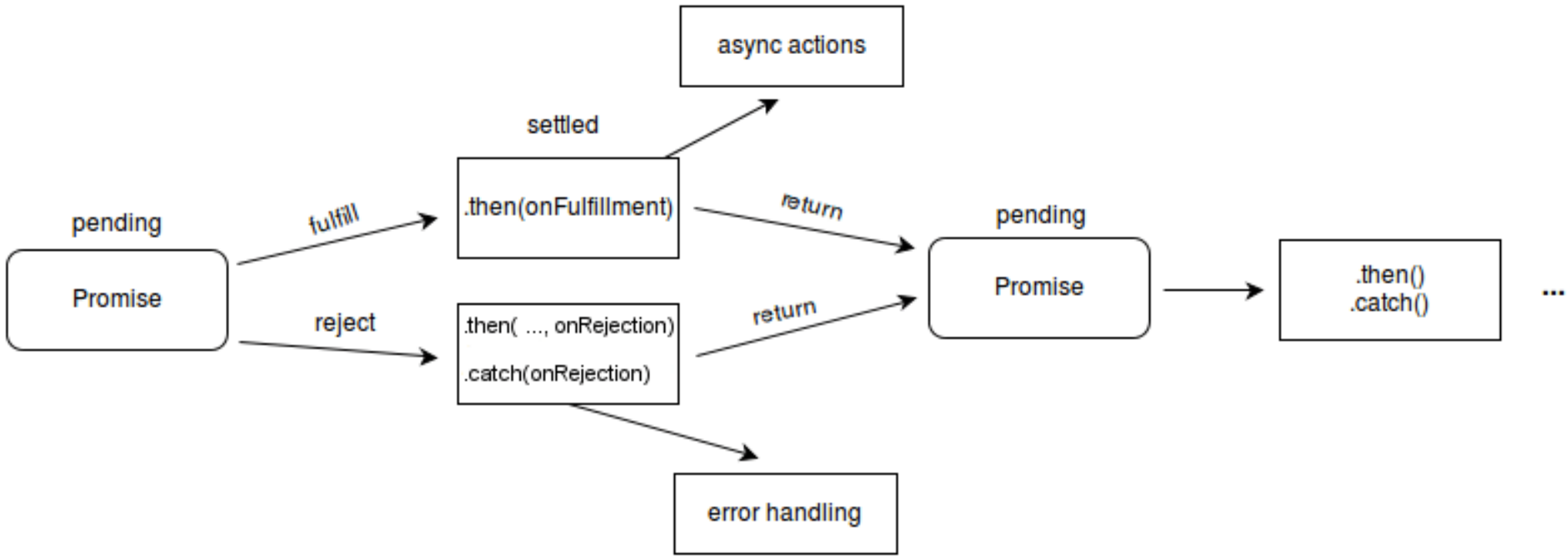
- Complex code flow
- Harder to debug
- Increased callbacks

JavaScript 'promises'



A Promise is an object representing the eventual completion or failure of an asynchronous operation.


promise
.then(() => { /* result */ })
.catch(() => { /* error */ })
.finally(() => { /* cleanup */ })


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

Our example: Querying the database

```
// Create a route for testing the db
app.get("/db_test", function(req, res) {
    // Assumes a table called test_table exists in your database
    var sql = 'select * from test_table';
    // As we are not inside an async function we cannot use await
    // So we use .then syntax to ensure that we wait until the
    // promise returned by the async function is resolved before we proceed
    db.query(sql).then(results => {
        console.log(results);
        res.json(results)
    });
});
```

Now lets look at the lab sheet and code: