Mechanics of Promises (2)

Understanding JavaScript Promise Generation & Behavior

Topics

- Async functions
- Async IIFE
- Sequential vs Parallel

Async functions

Async Functions

- An async function can contain an await expression, that pauses the execution of the async function and waits for the passed Promise's resolution.
- If the promise fulfills, you get the value back. If the promise rejects, the rejected value is thrown

```
async function myAsyncFunction() {
  try {
    const fulfilledValue = await promise;
  }
  catch (rejectedValue) {
    // …
  }
}
```

Async Functions return promises

- Async functions always return a promise, whether you use await or not.
- That promise resolves with whatever the async function returns, or rejects with whatever the async function throws.

Q: What does this function return?

```
async function getLuckyNumber() {
  let v;
  try {
    v = await readFileAsync('num.txt');
  } catch(e) {
   // Return Fallback Data
    v = 42;
  }
  return v;
}
```

Q: What does this function return?

```
async function getLuckyNumber() {
  let v;
  try {
    v = await readFileAsync('num.txt');
  } catch(e) {
   // Return Fallback Data
    v = 42;
  }
  return v;
}
```

A: a promise

Q: What does this function return?

```
async function getLuckyNumber() {
  let v;
  try {
    v = await readFileAsync('num.txt');
  } catch(e) {
   // Return Fallback Data
    v = 42;
  }
  return v;
}
```

```
// Somewhere else in your code...
console.log(getLuckyNumber());
```

Q: What does this function return?

```
async function getLuckyNumber() {
  let v;
  try {
    v = await readFileAsync('num.txt');
  } catch(e) {
   // Return Fallback Data
    v = 42;
  }
  return v;
}
```

```
// Somewhere else in your code...
console.log(getLuckyNumber());
```

✗

A: a promise

Q: What does this function return?

```
async function getLuckyNumber() {
  let v;
  try {
    v = await readFileAsync('num.txt');
  } catch(e) {
   // Return Fallback Data
    v = 42;
  }
  return v;
}
```

```
// Somewhere else in your code...
console.log(await getLuckyNumber());
```

Awaiting at Top-Level

Awaiting at top level

- As you know, the `await` keyword can only be used inside an `async` function.
- This presents a challenge: How to structure code that needs to `await` at top level?

Awaiting at top level

```
const express = require("express")
const models = require("./models")

const app = express();


await models.User.sync()
await models.Page.sync()
app.listen(PORT, () => {
  console.log(`Server is listening on port ${PORT}!`)
})
```

Awaiting at top level

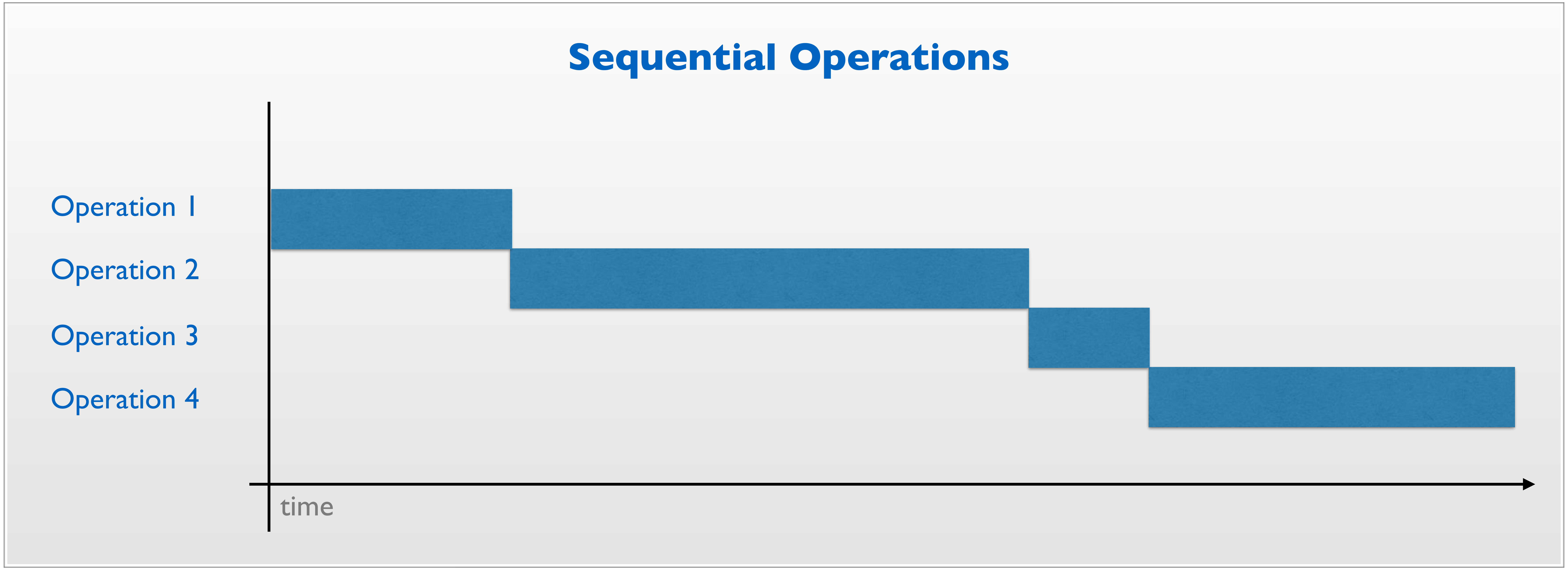
```
const express = require("express")
const models = require("./models")

const app = express();

const init = async () => {

  await models.User.sync()
  await models.Page.sync()
  app.listen(PORT, () => {
    console.log(`Server is listening on port ${PORT}!`)
  })

}

init();
```

Awaiting at top level

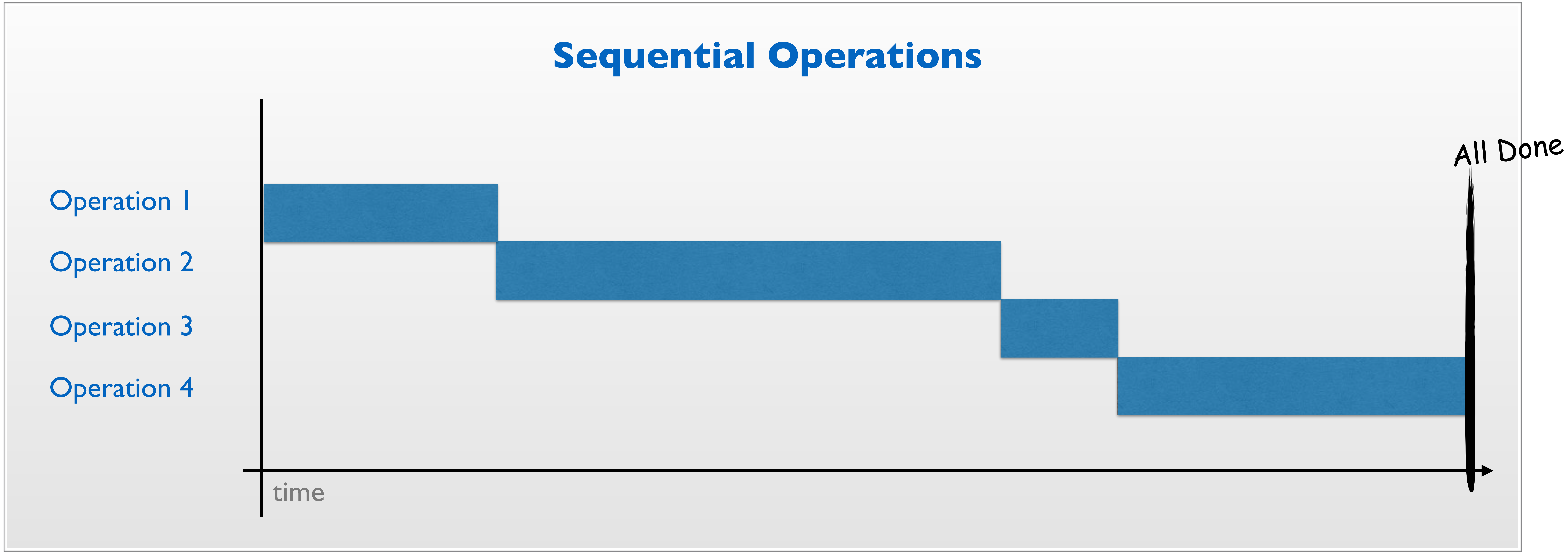
```
const express = require("express")
const models = require("./models")

const app = express();

(async () => {

  await models.User.sync()
  await models.Page.sync()
  app.listen(PORT, () => {
    console.log(`Server is listening on port ${PORT}!`)
  })

})();
```

Sequential vs Parallel

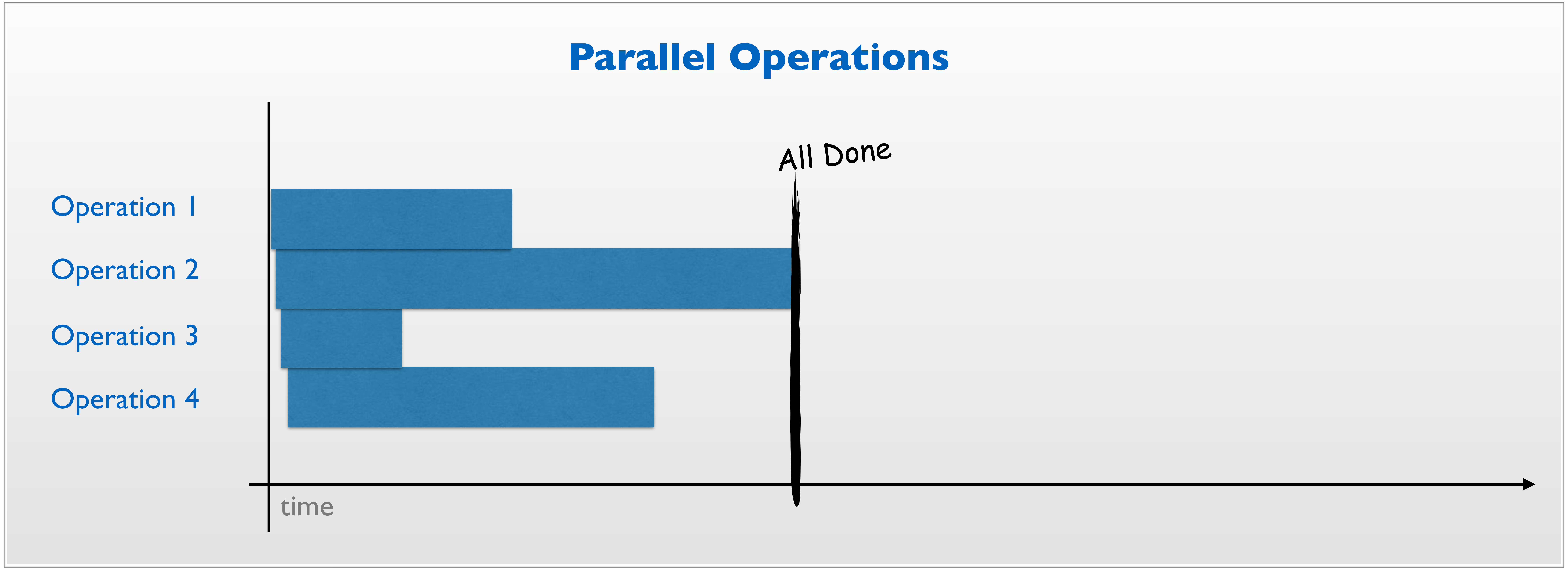
Sequential vs Parallel



Sequential vs Parallel



Sequential vs Parallel



Sequential vs Parallel

```
try {
  const number = await readFileAsync('/luckyNumber.txt');
  const charm = await readFileAsync('/luckyCharm.txt');
  const color = await readFileAsync('/luckyColor.txt');
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const number = await readFileAsync('/luckyNumber.txt');
  const charm = await readFileAsync('/luckyCharm.txt');
  const color = await readFileAsync('/luckyColor.txt');
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const number = await readFileAsync('/luckyNumber.txt');
  const charm = await readFileAsync('/luckyCharm.txt');
  const color = await readFileAsync('/luckyColor.txt');
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const number = await readFileAsync('/luckyNumber.txt');
  const charm = await readFileAsync('/luckyCharm.txt');
  const color = await readFileAsync('/luckyColor.txt');
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const number = await readFileAsync('/luckyNumber.txt');
  const charm = await readFileAsync('/luckyCharm.txt');
  const color = await readFileAsync('/luckyColor.txt');
} catch (error) {
  console.error(error);
}
```

Sequential

Promises are eager

- A promise will start doing whatever executor you give it as soon as the promise constructor is invoked.
- In other words, the task is already running whether you `await` the promise or not.

Sequential vs Parallel

```
try {
  const number = await readFileAsync('/luckyNumber.txt');
  const charm = await readFileAsync('/luckyCharm.txt');
  const color = await readFileAsync('/luckyColor.txt');
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const numberP = readFileAsync('/luckyNumber.txt');
  const charmP = readFileAsync('/luckyCharm.txt');
  const colorP = readFileAsync('/luckyColor.txt');
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const numberP = readFileAsync('/luckyNumber.txt');
  const charmP = readFileAsync('/luckyCharm.txt');
  const colorP = readFileAsync('/luckyColor.txt');
  const number = await numberP;
  const charm = await charmP;
  const color = await colorP;
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const numberP = readFileAsync('/luckyNumber.txt');
  const charmP = readFileAsync('/luckyCharm.txt');
  const colorP = readFileAsync('/luckyColor.txt');
  const number = await numberP;
  const charm = await charmP;
  const color = await colorP;
} catch (error) {
  console.error(error);
}
```

Parallel

Sequential vs Parallel

```
try {
  const numberP = readFileAsync('/luckyNumber.txt');
  const charmP = readFileAsync('/luckyCharm.txt');
  const colorP = readFileAsync('/luckyColor.txt');
  const number = await numberP;
  const charm = await charmP;
  const color = await colorP;
} catch (error) {
  console.error(error);
}
```

Parallel

But cumbersome...

`Promise.all([promises])`

- **Returns a single promise that resolves when all of the promises in the argument have resolved.**
  - Resolves with an array of results, in the <u>same order</u> as the input promises.
- **Rejects if any of the passed promises are rejected.**
  - If any of the passed-in promises reject, Promise.all rejects with the value of the <u>earliest</u> promise that rejected.

Sequential vs Parallel

```
try {
  const numberP = readFileAsync('/luckyNumber.txt');
  const charmP = readFileAsync('/luckyCharm.txt');
  const colorP = readFileAsync('/luckyColor.txt');
  const number = await numberP;
  const charm = await charmP;
  const color = await colorP;
} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const numberP = readFileAsync('/luckyNumber.txt');
  const charmP = readFileAsync('/luckyCharm.txt');
  const colorP = readFileAsync('/luckyColor.txt');

  const values = await Promise.all([numberP, charmP, colorP])
  console.log(values); // Array [42, "Four-leaf clover", "Red"]

} catch (error) {
  console.error(error);
}
```

Sequential vs Parallel

```
try {
  const numberP = readFileAsync('/luckyNumber.txt');
  const charmP = readFileAsync('/luckyCharm.txt');
  const colorP = readFileAsync('/luckyColor.txt');

  const values = await Promise.all([numberP, charmP, colorP])
  console.log(values); // Array [42, "Four-leaf clover", "Red"]

} catch (error) {
  console.error(error);
}
```

Parallel

Sequential vs Parallel

```
const numberP = readFileAsync('/luckyNumber.txt');
const charmP = readFileAsync('/luckyCharm.txt');
const colorP = readFileAsync('/luckyColor.txt');

try {
  const values = await Promise.all([numberP, charmP, colorP])
  console.log(values); // Array [42, "Four-leaf clover", "Red"]

} catch (error) {
  console.error(error);
}
```

Parallel

PROCEED
WITH
CAUTION

Sequential vs Parallel

- A given asynchronous operation may depend on the result of a previous one.

```
const tryGetRich = async () => {

 let num = await readFileAsync('/luckyNumber.txt')
 let success = await bookmaker.bet(num)

 if(success) {
   console.log("I'm rich!")
 }

}
```