Mechanics of Promises *(2)*

Understanding JavaScript Promise Generation & Behavior

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 getProcessedData(url) {
  let v;
  try {
    v = await downloadData(url);
  } catch(e) {
   // Return Fallback Data
    v = 42;
  }
  return v;
}
```

Q: What does this function return?

```
async function getProcessedData(url) {
  let v;
  try {
    v = await downloadData(url);
  } catch(e) {
   // Return Fallback Data
    v = 42;
  }
  return v;
}
```

A: A promise

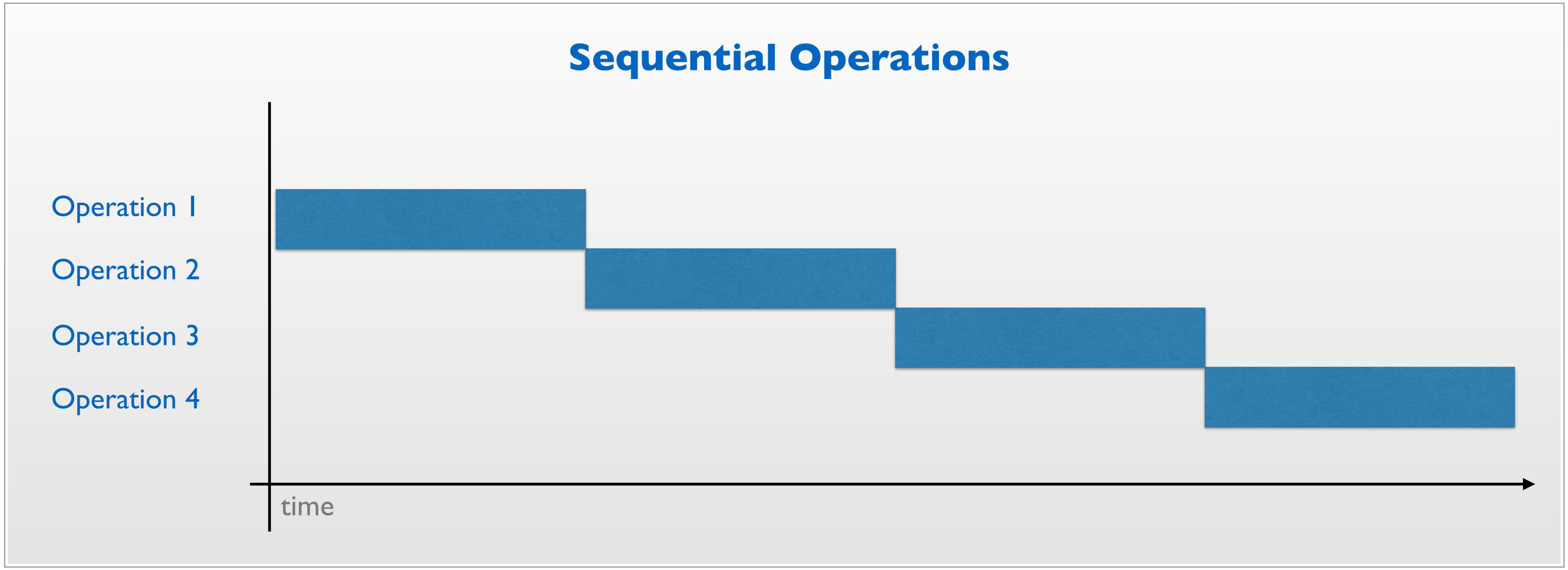
Async Functions return promises

```
async function myAsyncFunction() {
  return new Promise(/*...*/);
}
```

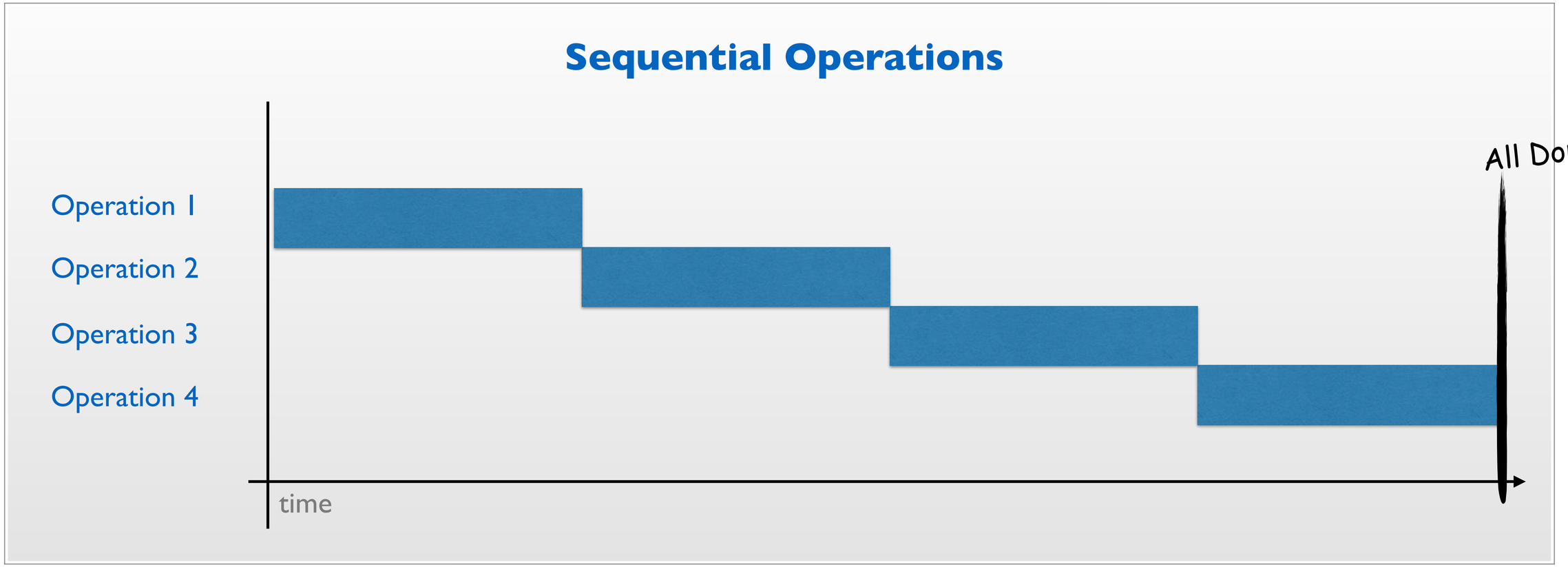
- Returning a Promise from an async function means that the returned Promise now mirrors the state of that Promise.

Sequential vs Parallel

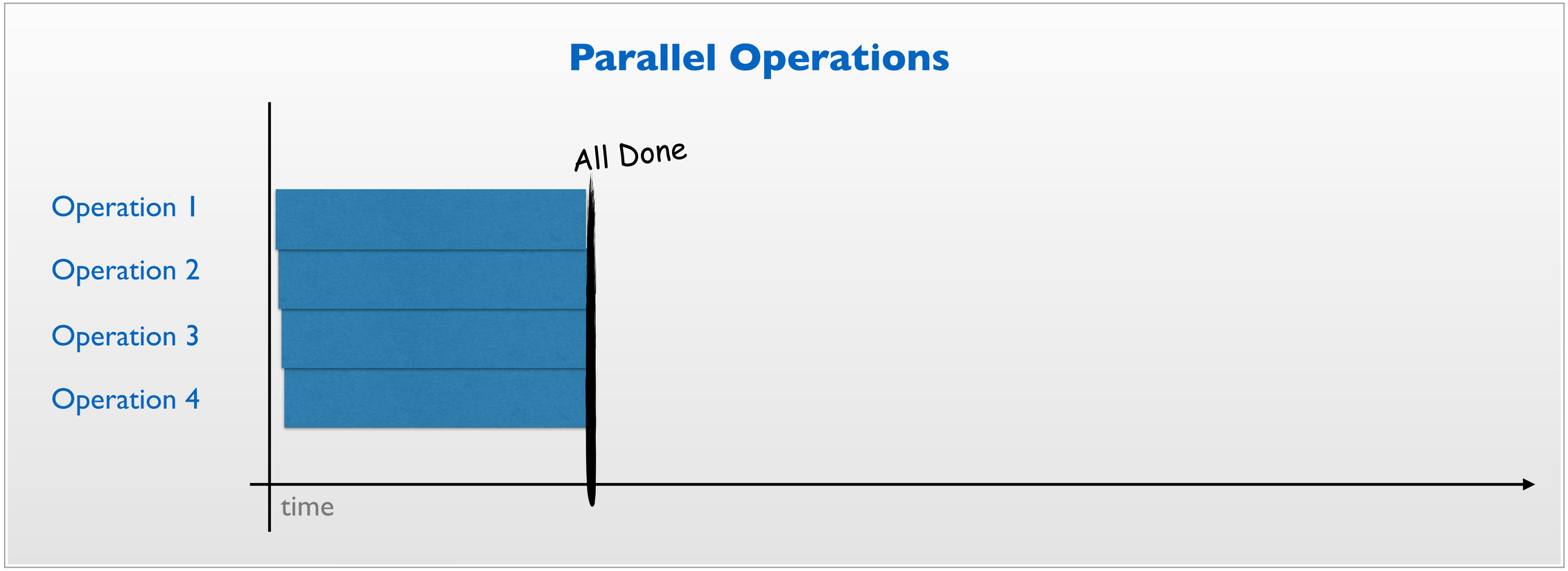
Sequential vs Parallel



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);
}
```

Promises are eager

- A promise will start doing whatever task you give it as soon as the promise constructor is invoked.
- In other words, the task will run whether you `await` for 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);
}
```

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);
}
```

Promise.all([promises])

- **Returns a single promise that resolves when all of the promises in the argument have resolved.**
- **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 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);
}
```

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);
}
```


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!")
 }

}
```