Async/Await

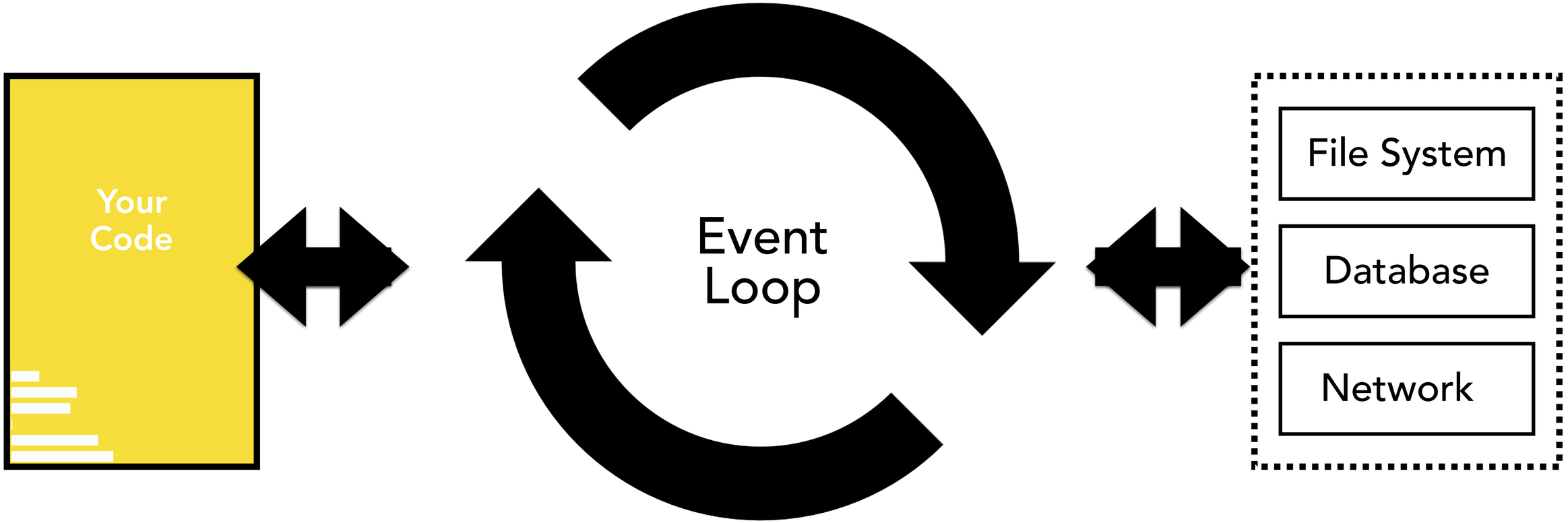
I Promise to await for async code...

Review: What is “asynchronous” code?

Asynchronous (aka *async*) literally means "happening at disconnected times."

Async code in JS will run at an arbitrary (unknown) future time, and *other JS code* can run in the *meantime*.

The Event Loop



Operations that involve Asynchronous Code

A few examples:

- Opening, reading, and closing files
- Making API calls such as HTTP requests from our frontend
- Accessing a database (read, write, and delete operations)

How to handle asynchronous code?

For a while, the only answer was…..

Callbacks!

Async with callbacks

```
console.log("Getting Configuration")
fs.readFile('/config.json', 'utf8', (err, data) => {
  console.log("Got configuration:", data)
});
console.log("Moving on…");
```

- BTW, In which order will the logs fire?

Problems with callbacks

```
const tryGetRich = () => {
  readFile('/luckyNumbers.txt', (err, fileContent) => {
    // Do something with lucky numbers
  })
}
```

Problems with callbacks

```
const tryGetRich = () => {
  readFile('/luckyNumbers.txt', (err, fileContent) => {
    nums = fileContent.split(",");
    nums.forEach(num => {
      bookmaker.getHorse(num, (err, horse) => {
        // Ok, this is getting a little confusing
      })
    })
  })
}
```

Problems with callbacks

```
const tryGetRich = () => {
  readFile('/luckyNumbers.txt', (err, fileContent) => {
    nums = fileContent.split(",");
    nums.forEach(num => {
      bookmaker.getHorse(num, (err, horse) => {
        bookmaker.bet(horse, (err, success) => {
          if(success) {
            // Help...
          }
        })
      })
      console.log('When will I run??')
    })
  })
}
```

CALLBACK HELL

How to better handle asyncronous code?

Because callbacks are difficult to work with we invented…

Promises!

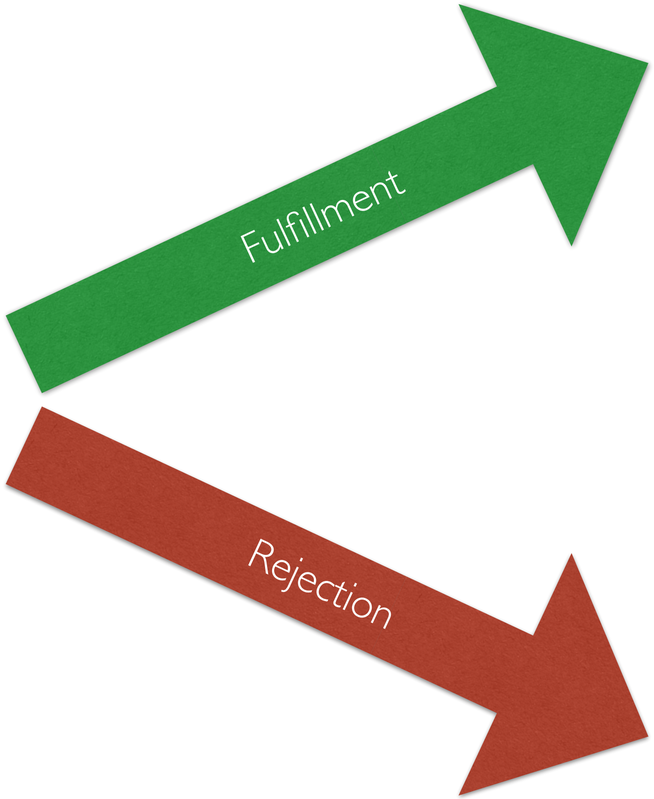
What is a Promise?

- A promise is a JavaScript object that represents the eventual result of an asynchronous operation.
- It it an object containing a *value* and *status*.

Promise Outcomes

```
readFileAsync('/luckyNumber.txt')
```

```
{
  [[PromiseValue]]: undefined,
  [[PromiseState]]: "pending"
}
```




```
{
  [[PromiseValue]]: "42",
  [[PromiseState]]: "fullfilled"
}
```

```
{
  [[PromiseValue]]: "Error: Something
                     went wrong!”,
  [[PromiseState]]: "rejected"
}
```

What is async/await?

- **Built on top of promises, async/await syntax allows us to handle asynchronous code in a simpler way**
- **The “async” keyword labels functions as having asynchronous code**
  - This also forces the function itself to return a promise!
- **The “await” keyword is ONLY used inside of an “async” function and forces the code inside of your function to stop running until that operation is finished**

async/await

```
const num = await readFileAsync('/luckyNumber.txt')
```

async/await

```
async function getNumber() {
  const num = await readFileAsync('/luckyNumber.txt')
}
getNumber()
```

async/await

```
const getNumber = async () => {
  const num = await readFileAsync('/luckyNumber.txt')
}
getNumber()
```

What about error handling?

Try/Catch

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

  try {
    let num = await readFileAsync(‘/luckyNumber.txt')
    let success = await bookmaker.bet(num)
  } catch (error) {
    console.error(error.message)
  }

}

getNumber()
```

You may also come across...

Promise.all()

- A method that takes in multiple promises but returns a single promise representing their collective status
- Useful if you don't care what order those promises resolve and only want to know when they are all completed

```
const values = await Promise.all( [ promise1, promise2, promise3 ] )
```

.then

- Also built on top of promises, it's an older alternative to async/await
- Accepts up to two arguments (both technically optional)
  - "Success" callback
  - "Error" callback
- If the promise resolves (succeeds)
  - "Success" callback is invoked with the value
- If the promise rejects (fails)
  - "Error" callback is invoked with the value

Async/await vs .then

```
const result = await myDB.query('SELECT ...')
console.log(result)
```

```
myDB.query('SELECT ...')
.then(result => {
  console.log(result)
})
```

Async/await vs .then

```
try {
  const result = await myDB.query('SELECT ...')
  console.log(result)
} catch (err) {
  console.log(err)
}
```

```
myDB.query('SELECT ...')
.then(result => {
  console.log(result)
})
```

Async/await vs .then

```
try {
  const result = await myDB.query('SELECT ...')
  console.log(result)
} catch (err) {
  console.log(err)
}
```

```
myDB.query('SELECT ...')
.then(result => {
  console.log(result) // success
})
.catch(err => {
  console.log(err) // err
})
```

Demo