Practical Promises

As opposed to impractical promises

what is asynchronous code?

Asynchronous **(aka *async*) just means:**
“takes some time” or

“happens in the future, not right now”…

…and JavaScript won't wait for it.

what is asynchronous code?

```
console.log("One")
setTimeout(() => console.log("Two"), 10)
console.log("Three")
```

- **In which order will the logs fire?**

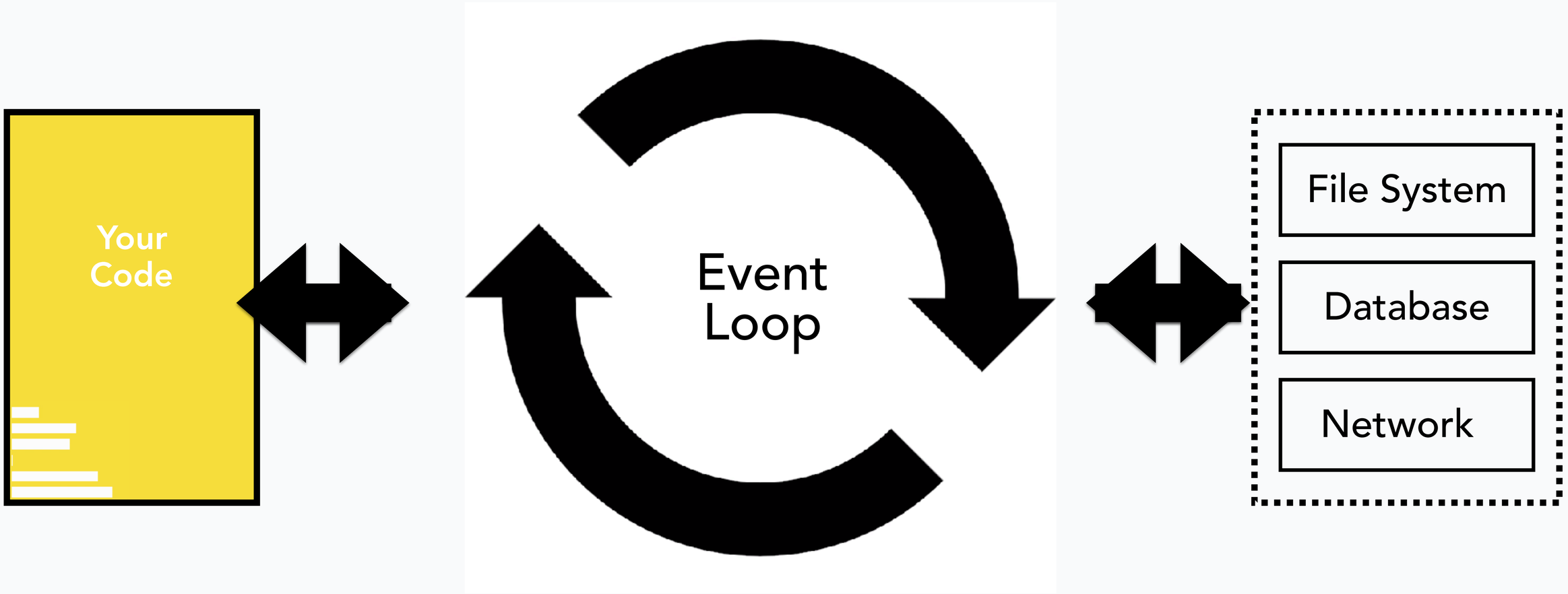
what is a asynchronous code?

```
console.log("One")
setTimeout(() => console.log("Two"), 10)
console.log("Three")
```

- **In which order will the logs fire?**

  One
  Three
  Two

what is asynchronous code?



How do we handle asynchronous code?

1. Callbacks

what is a callback?

what is a callback?

Technically: a function passed to another function

two flavors…

- Blocking
- Non-blocking

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

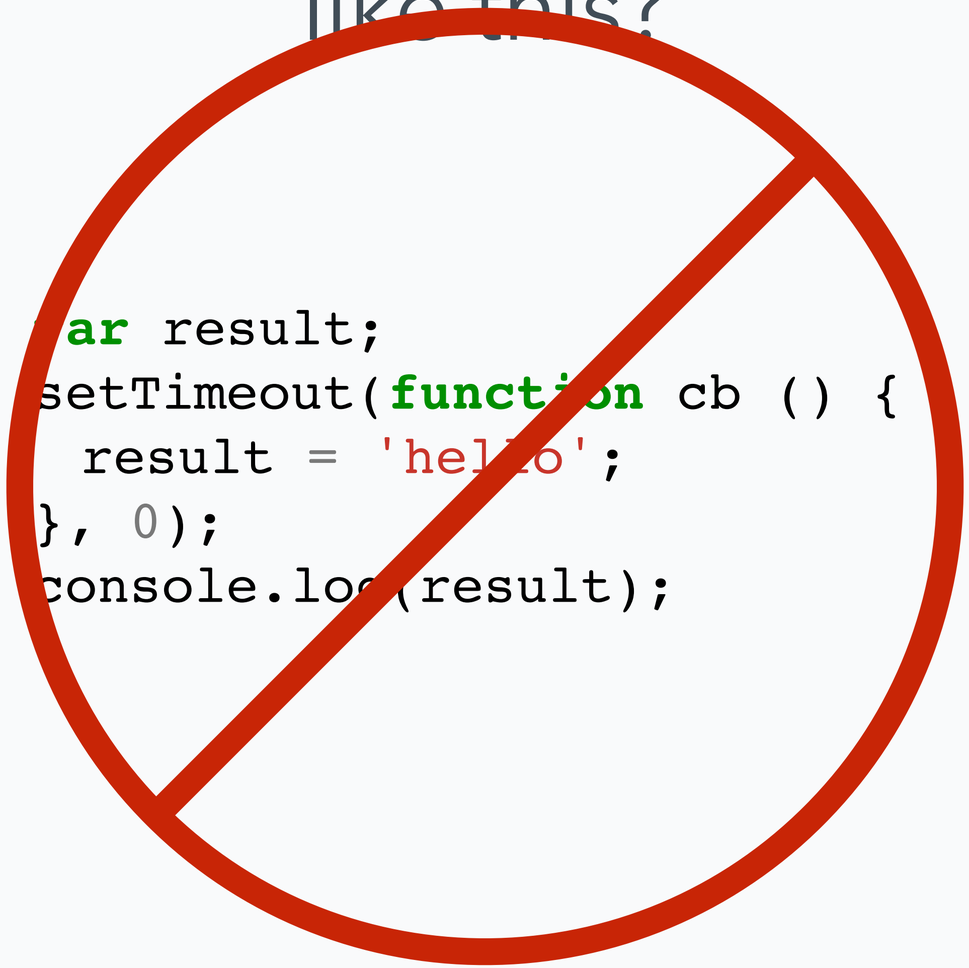
```
const tryGetRich = () => {
  readFile('/luckyNumbers.txt', (err, fileContent) => {
    nums = fileContent.split(",");
    nums.forEach(num => {
      bookmaker.getHorse(num, (err, horse) => {
        // Ok, this is getting a little confusing
      })
    })
  })
}
```

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

like this?

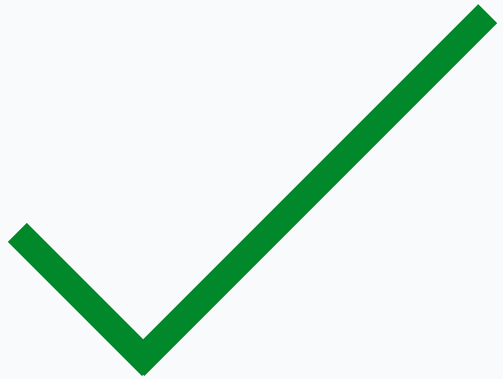

ar result;
setTimeout(function cb () {
result = 'hello';
}, 0);
console.log(result);

like this?

```
var result = setTimeout(function () {
  return 'hello';
}, 0);
console.log(result);
```

like this?

```
setTimeout(function cb () {
  var result = 'hello';
  console.log(result);
}, 0);
```



How do we handle asynchronous code?

1. Callbacks
2. **Promises**

promise

"A promise represents the eventual result of an asynchronous operation."

– The Promises/A+ Spec

vanilla async callback

```
fs.readFile('file.txt',
  function callback (err, data) {…}
);
```

async promise

```
fs.readFileAsync('file.txt')
.then(
  function onSuccess (data) {…},
  function onError (err) {…}
);
```

promise

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

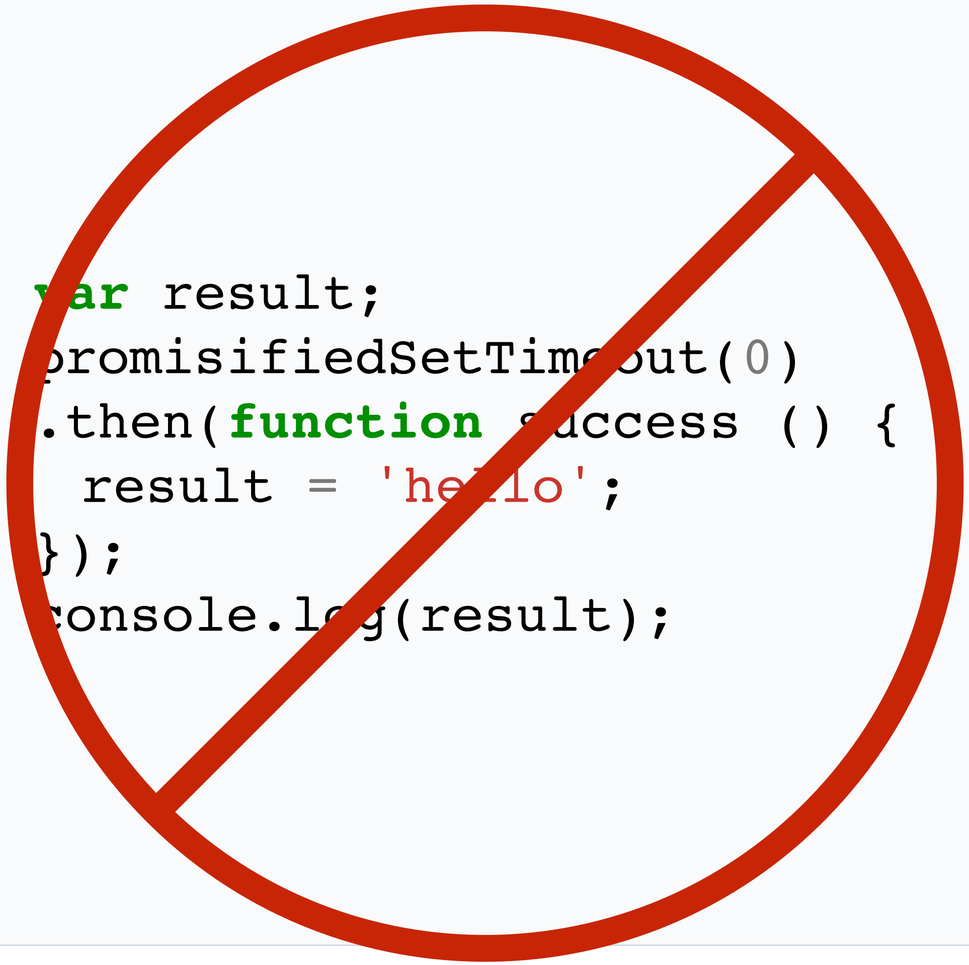
```
{
  [[PromiseValue]]: undefined,
  [[PromiseStatus]]: "pending"
}
```

promise

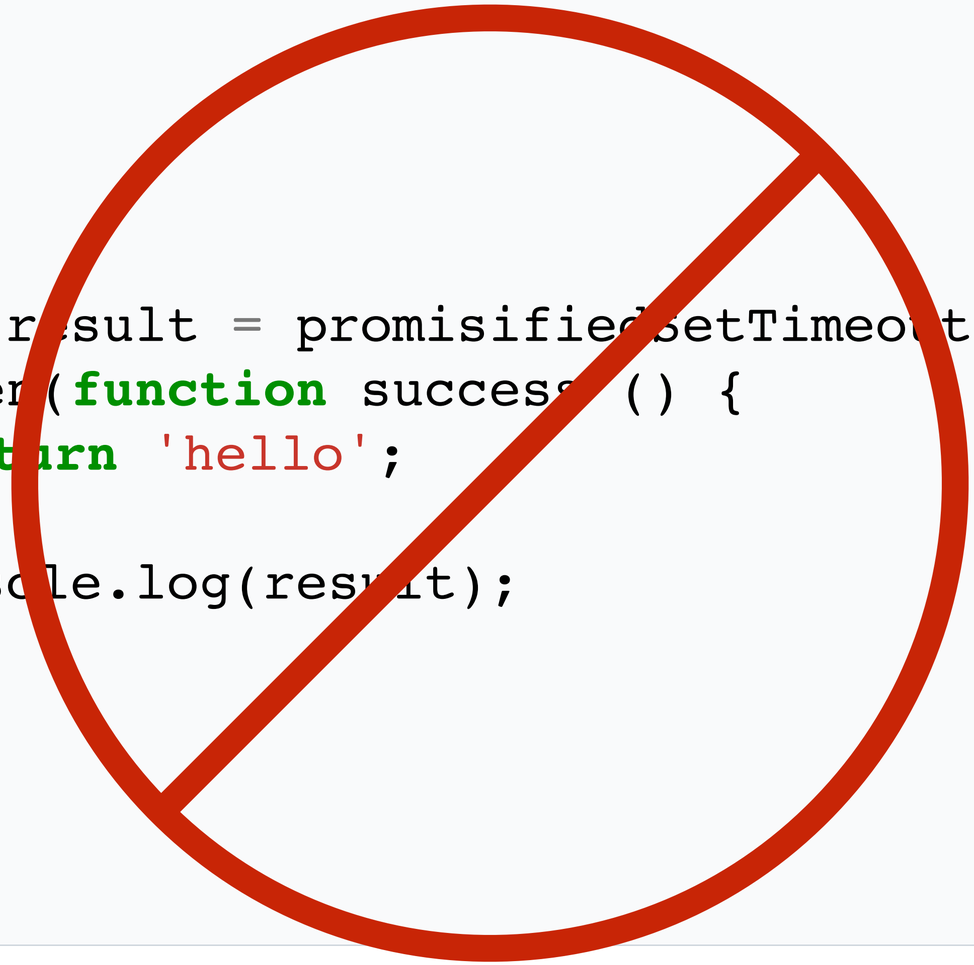
```
readFileAsync(‘/luckyNumber.txt’)
```

```
{
  [[PromiseValue]]: "42",
  [[PromiseStatus]]: "fulfilled"
}
```

like this?


var result;
promisifiedSetTimeout(0)
.then(function success () {
result = 'hello';
});
console.log(result);

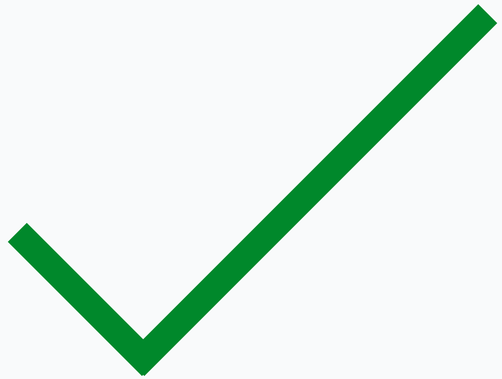
like this?



```
var result = promisifiedSetTimeout(0)
.then(function success () {
  return 'hello';
});
console.log(result);
```

like this?

```
promisifiedSetTimeout(0)
.then(function success () {
  var result = 'hello';
  console.log(result);
});
```



reading a file

synchronous

```
var path = ‘demo-poem.txt';

console.log('- I am first -');

try {
  var buff = fs.readFileSync(path);
  console.log(buff.toString());
} catch (err) {
  console.error(err);
}

console.log('- I am last -');
```

async (callbacks)

```
var path = 'demo-poem.txt';

fs.readFile(path, function (err, buff) {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});

console.log('- I am first -');
```

async

```
var path = 'demo-poem.txt';

promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString());
}, function (err) {
  console.error(err);
})
.then(function () {
  console.log('- I am last -');
});

console.log('- I am first -');
```

error handling

fine

```
var path = 'demo-poem.txt';

promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString());
}, function (err) {
  console.error(err);
});
```

better

```
var path = ‘demo-poem.txt';

promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString());
})
.then(null, function (err) {
  console.error(err);
});
```

best*

```
var path = ‘demo-poem.txt';

promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString());
})
.catch(function (err) {
  console.error(err);
});
```

error handling continued

```
var path = 'demo-poem.txt';
var path = 'demo-poem-2.txt';

promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString());
 return promisifiedReadFile(path2);
})
.then(function (buff) {
  console.log(buff.toString());
})
.catch(function (err) {
  console.error(err);
});
```

promise advantages

- Portable
- Multiple handlers
- “Linear”
- Unified error handling