



Asynchronicity: concurrency. A tale of

(a.k.a: the subtle art of making a PB&J sandwich)

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP

```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP

```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP

```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getfileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null),
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript



```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



@joel__lord
#confoo

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```

PHP



```
1 <?php
2
3 echo("Before reading file\n");
4 $fileContents = file_get_contents("sample.txt");
5 echo("After file read\n");
6
7 echo("Before HTTP request\n");
8 $ch = curl_init("https://swapi.co/api/planets/1/");
9 $data = curl_exec($ch);
10 curl_close($ch);
11 echo("After HTTP request\n");
```

Javascript

```
1 <script type="text/javascript">
2
3 console.log("Before reading file");
4 getFileContent("sample.txt", () => {
5   console.log("After file read");
6 });
7
8 echo("Before HTTP request");
9 const req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
12     console.log("After HTTP request")
13   }
14 };
15
16 req.open("GET", "https://swapi.co/api/planets/1/", true);
17 req.send(null);
18 </script>
```



Race Conditions



@joel__lord



joellord

About Me



Our Agenda

- Event Loop
- Callbacks
- Promises
- Generators
- Await/Async
- Events
- Reactive Events



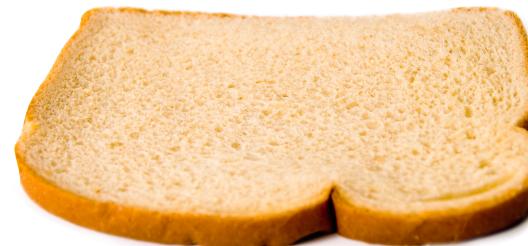


How to make a PB&J sandwich



PB&J Sandwich

- Get the ingredients
 - Get some bread
 - Get some PB
 - Get some J
- Prepare the sandwich
 - Spread PB
 - Spread J
- Close it
- Eat it!



A close-up, slightly blurred image of a dessert, possibly a tart or pie, with a dark, glossy filling and a crumbly crust.

The Event Loop

Classic Architecture (PHP, Java)



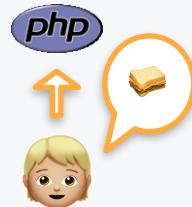
Classic Architecture (PHP, Java)

- I want a sandwich!



Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start



Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients



Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients



Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients



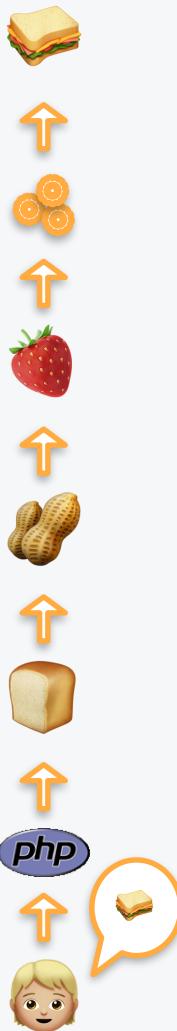
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
 - Prepares the sandwich



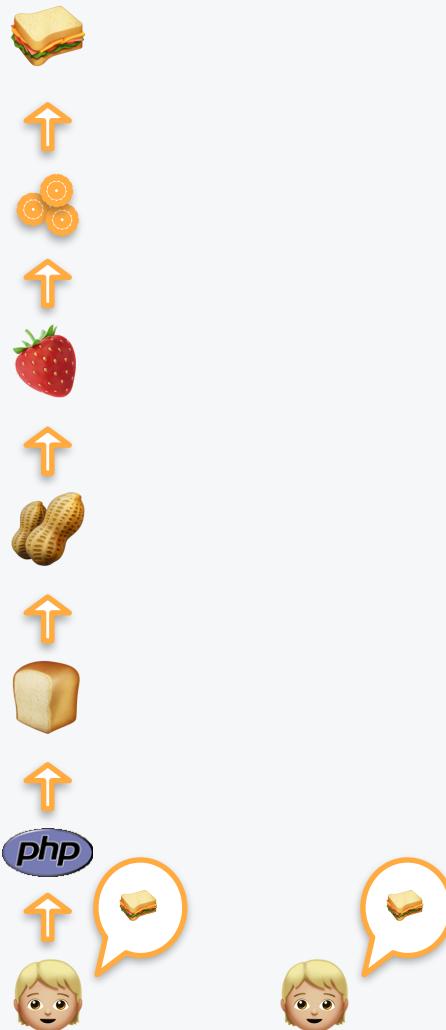
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



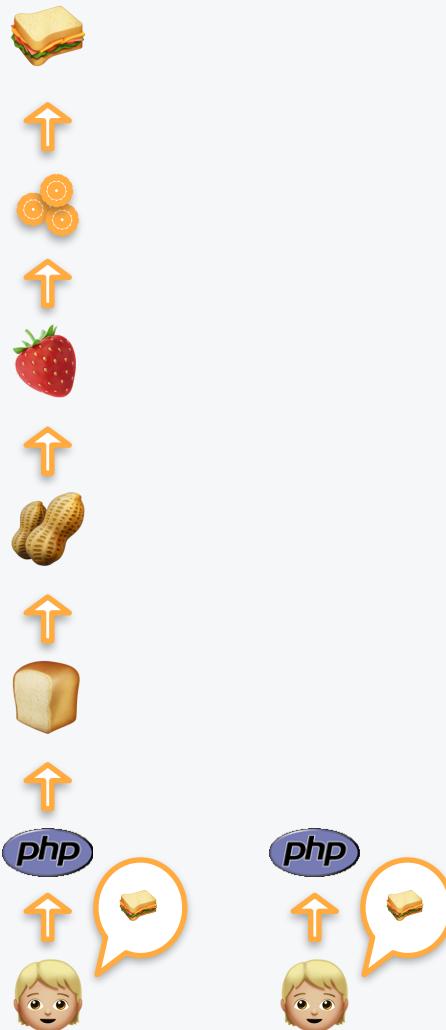
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!



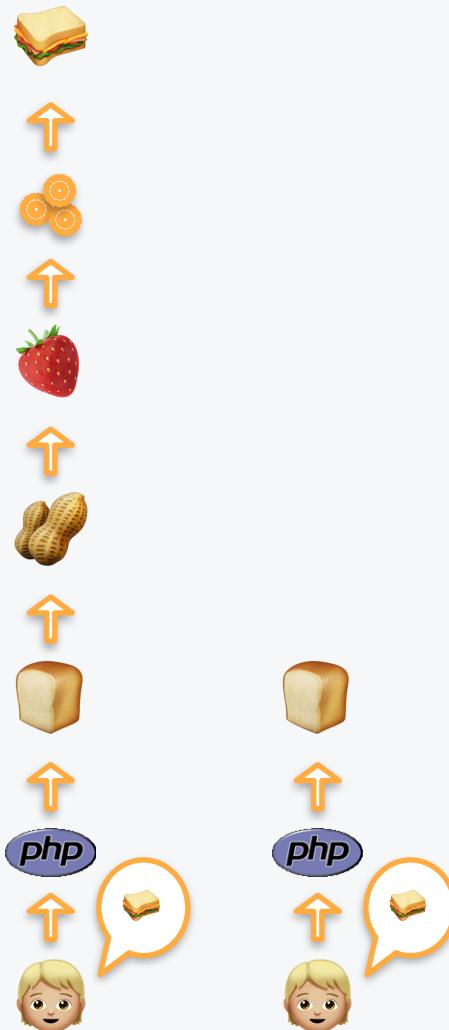
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts



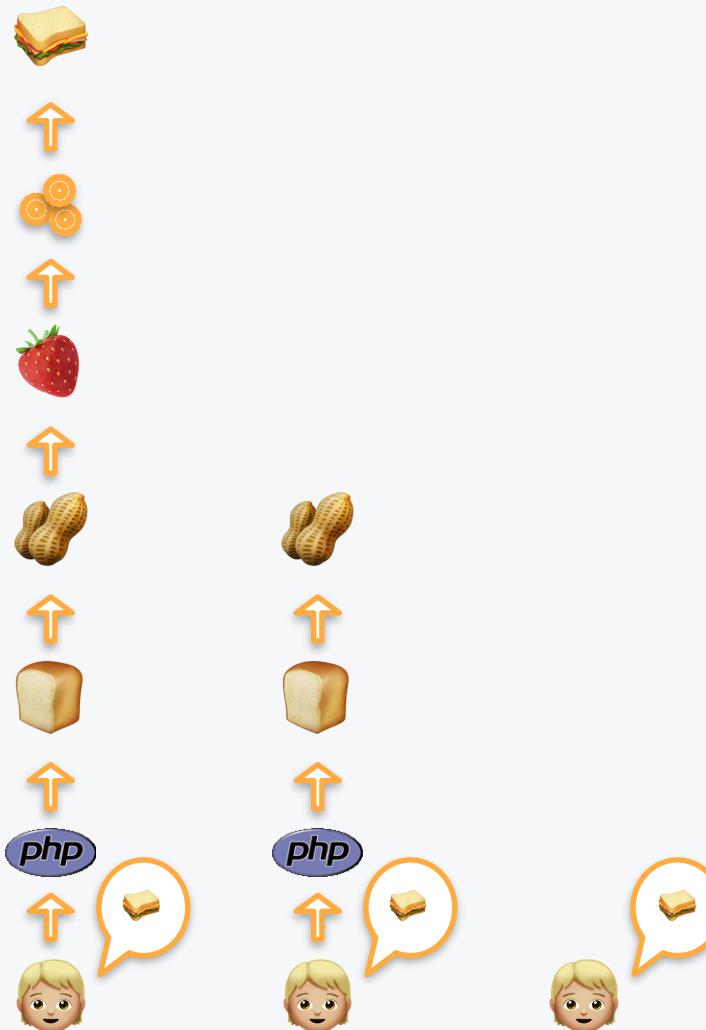
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients



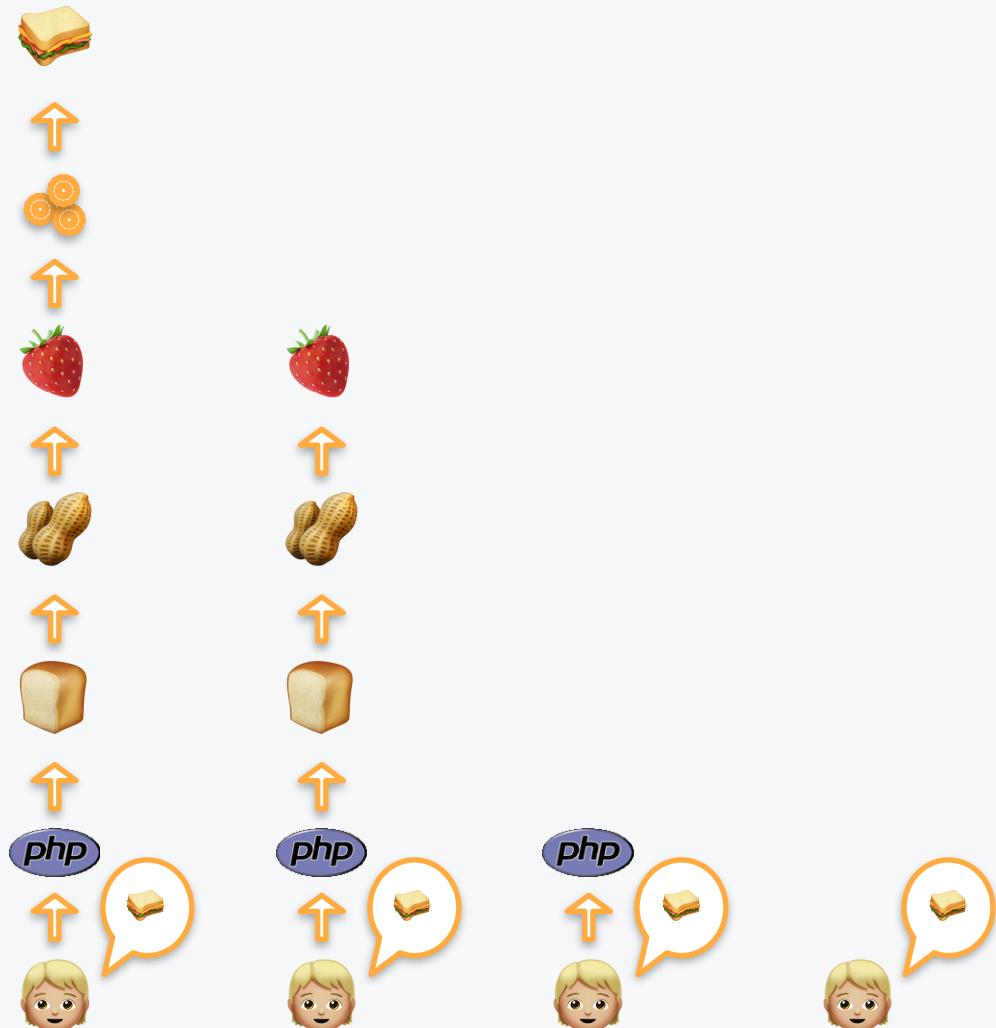
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich



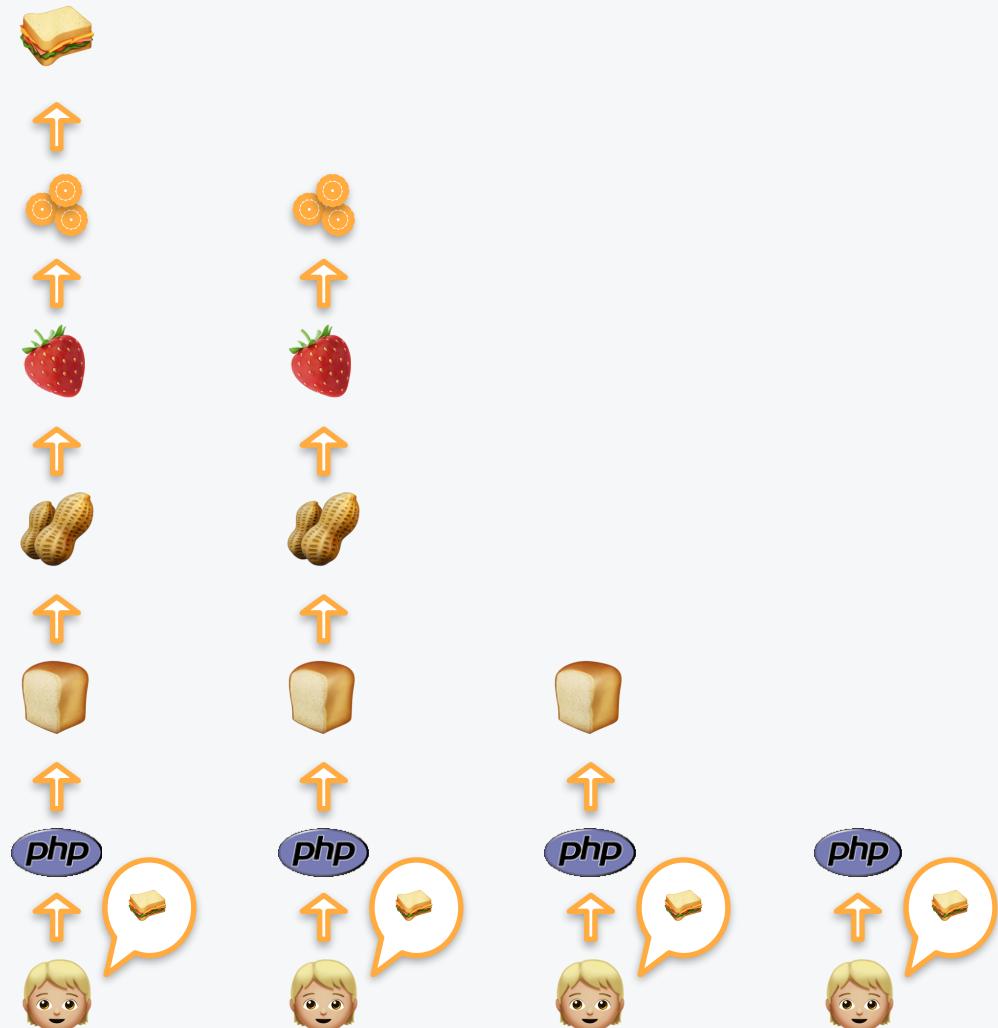
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



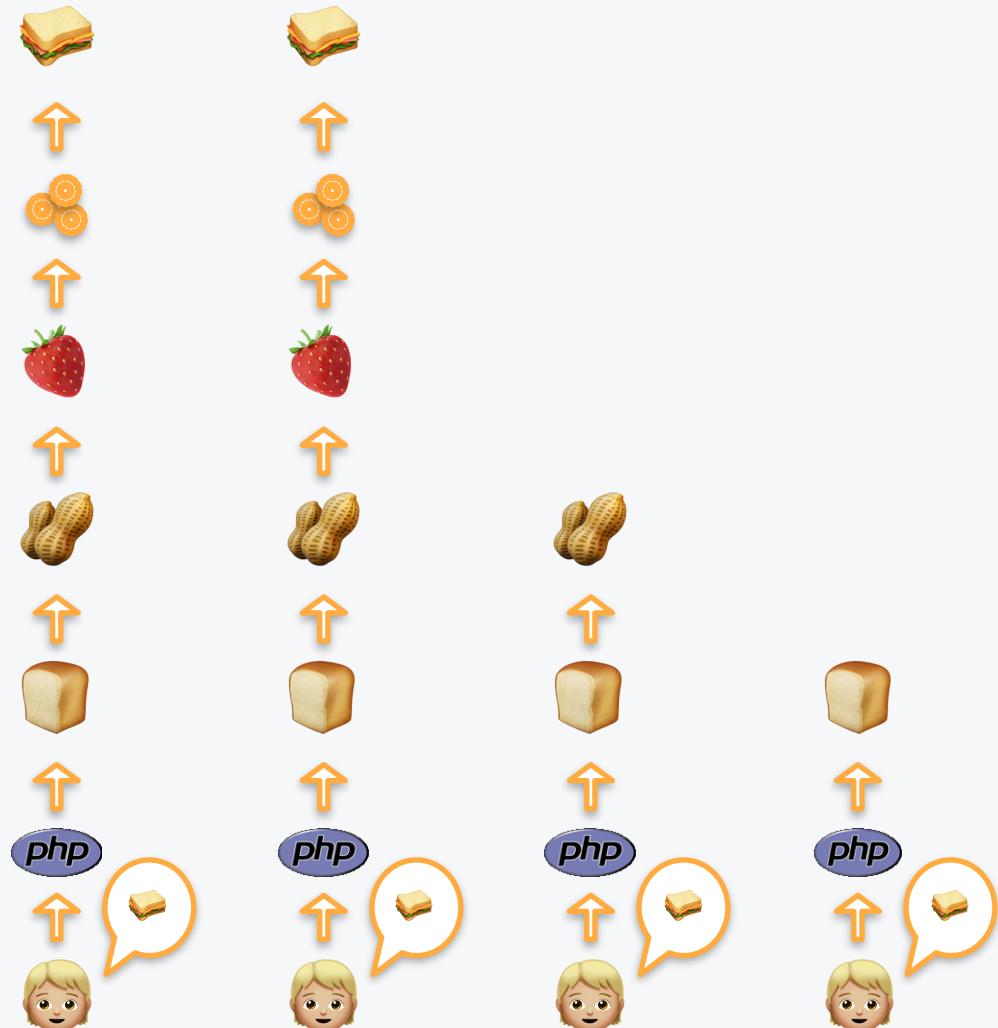
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



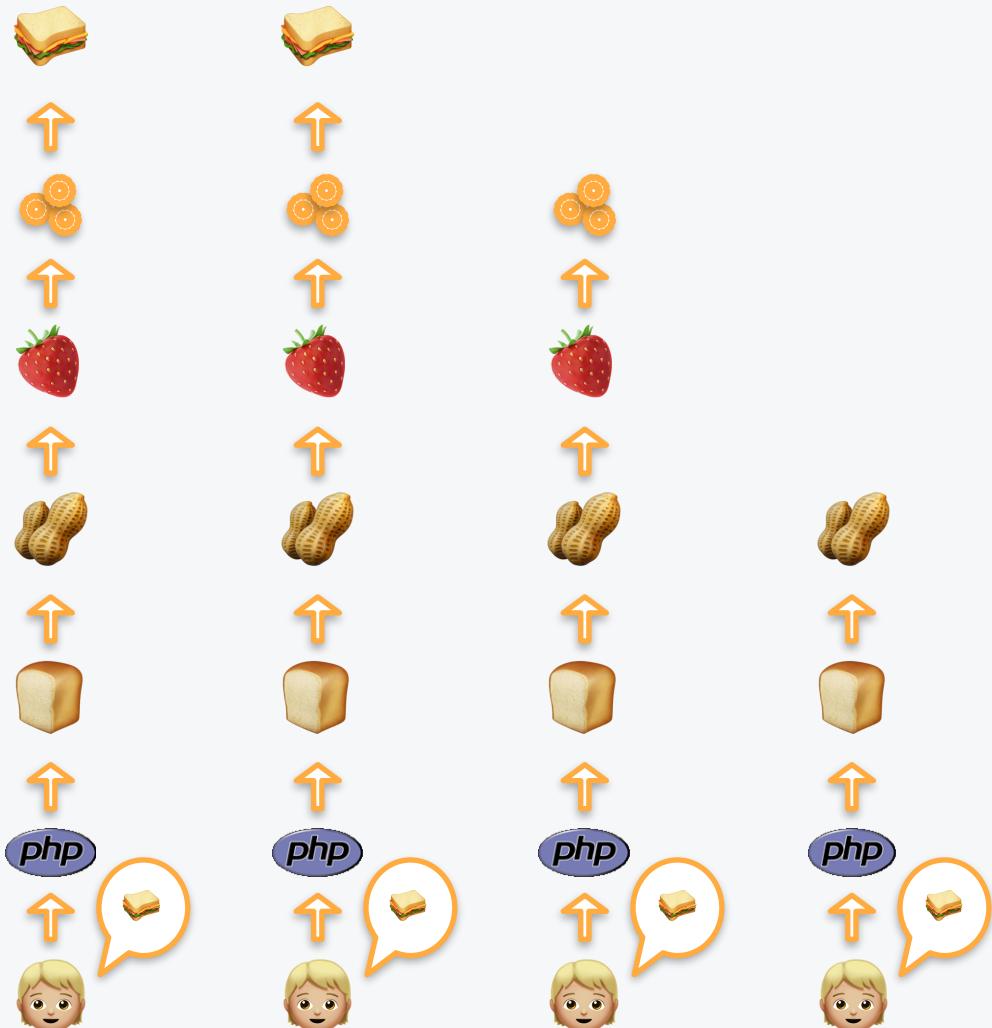
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



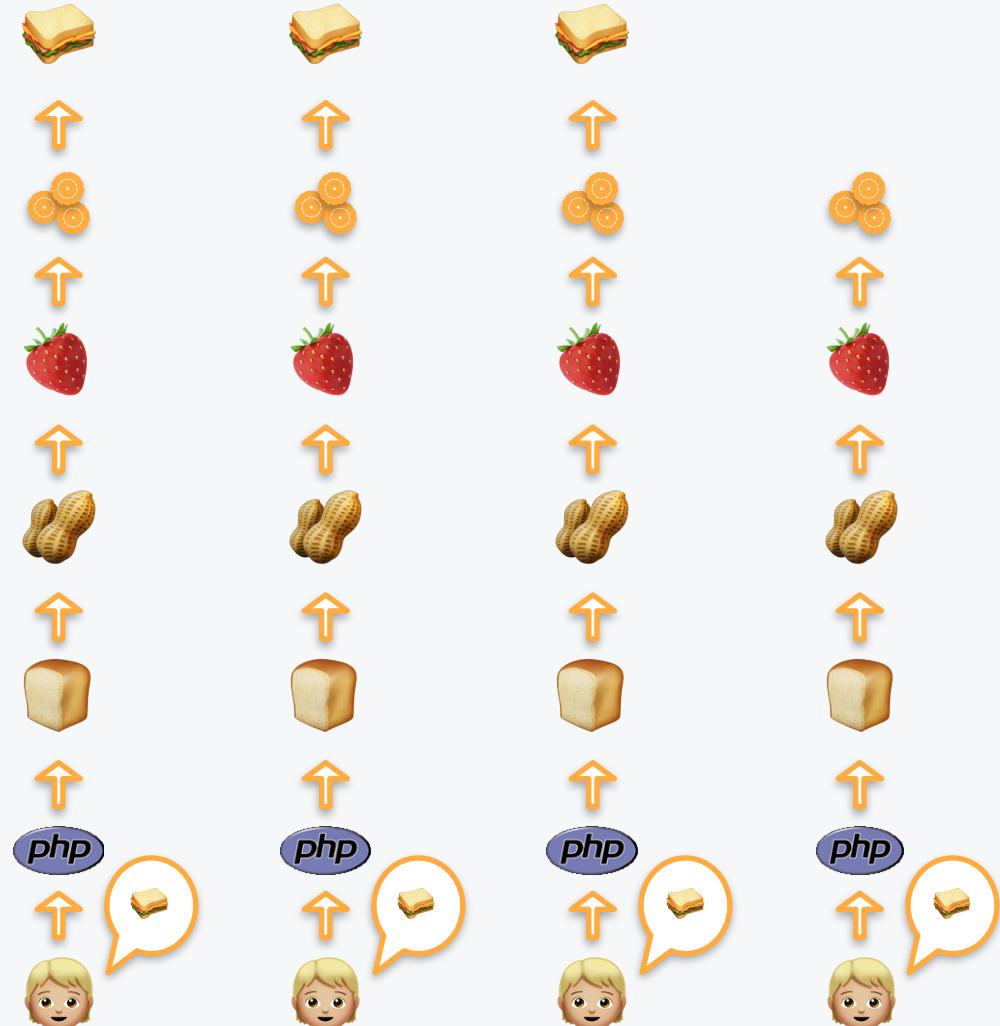
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



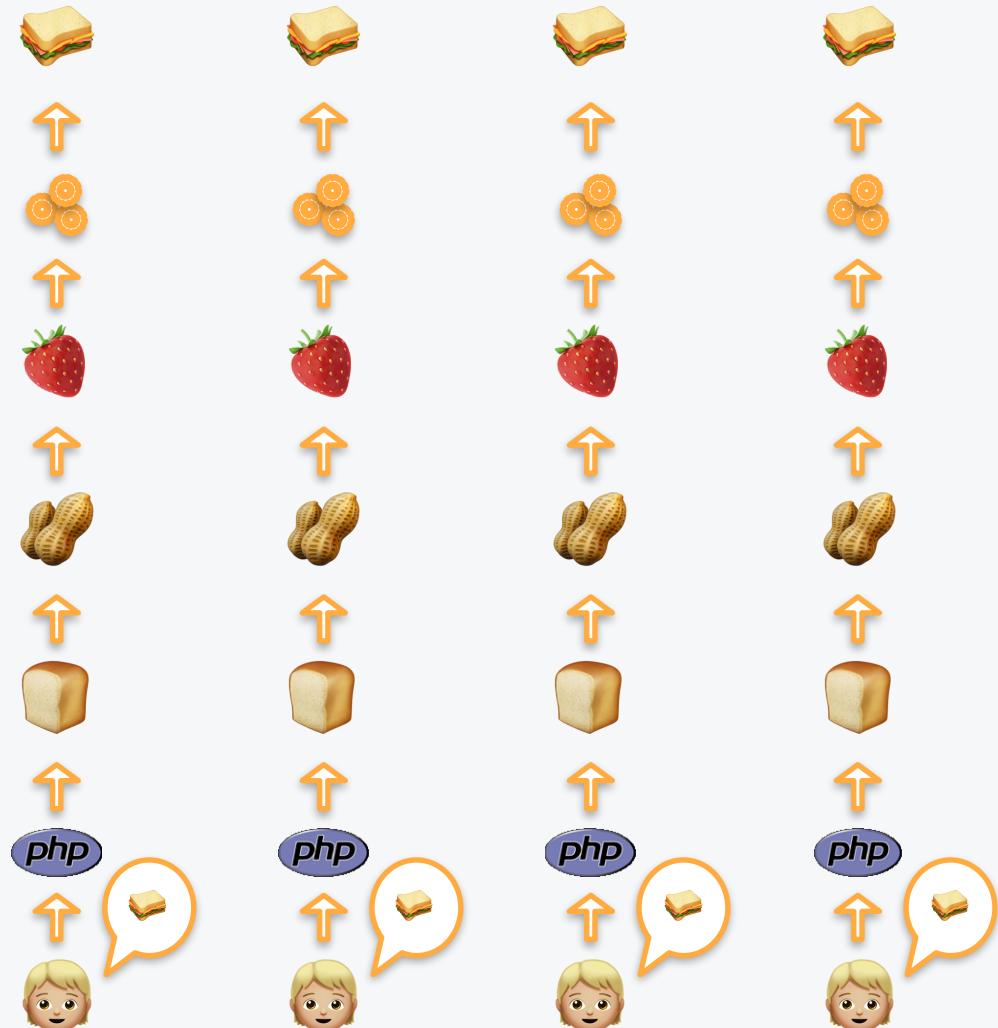
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



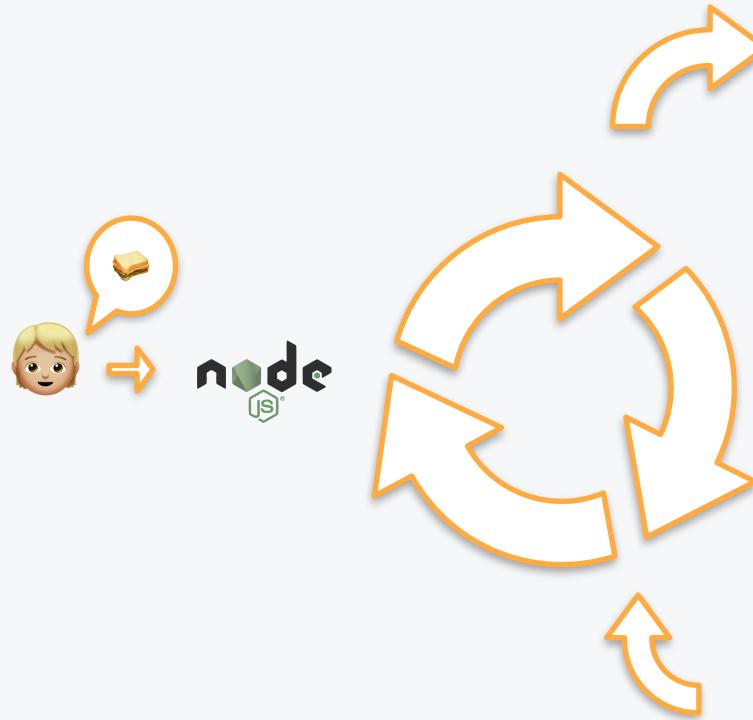
Classic Architecture (PHP, Java)

- I want a sandwich!
- Process start
 - Fetches the ingredients
- A new customer arrives:
I want a sandwich!
- Another process starts
 - Fetches the ingredients
 - Prepares the sandwich
 - Here's your sandwich



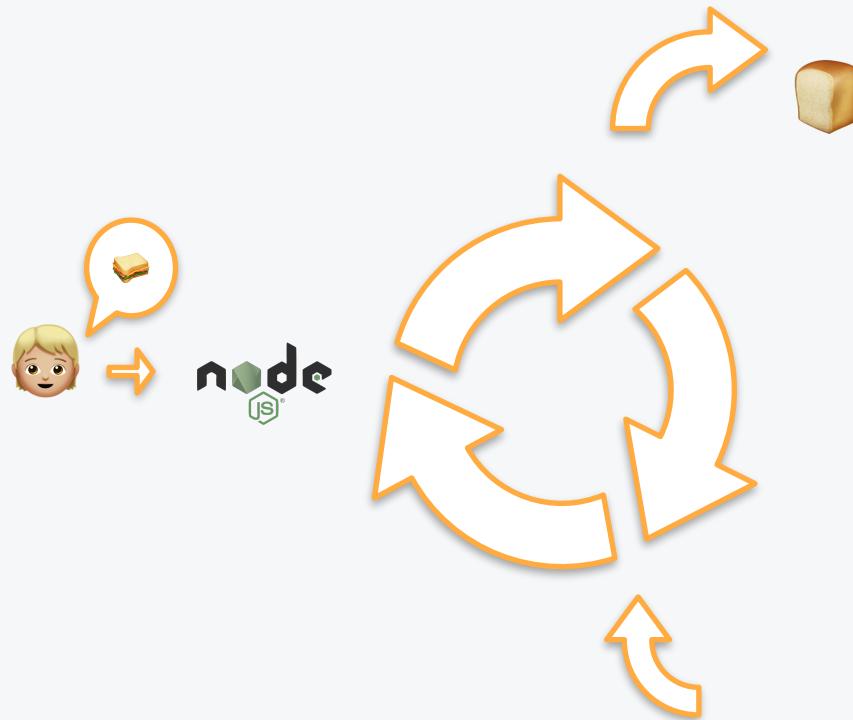
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



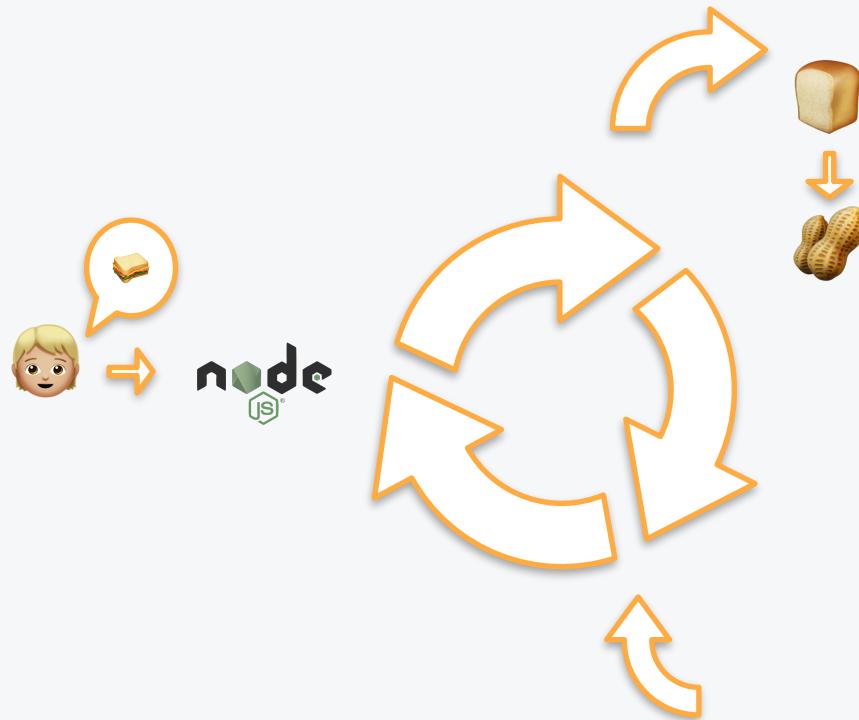
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



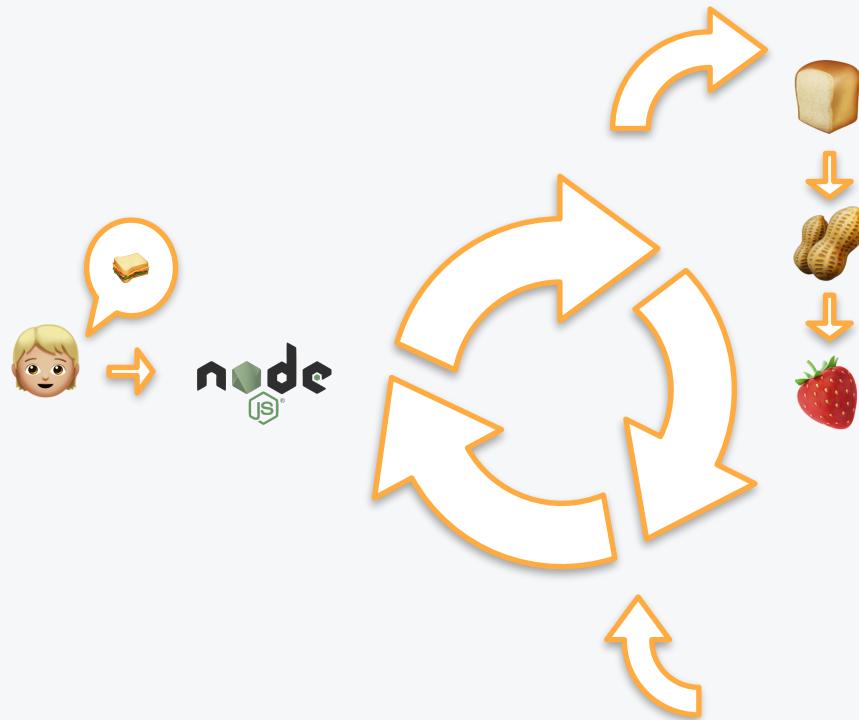
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



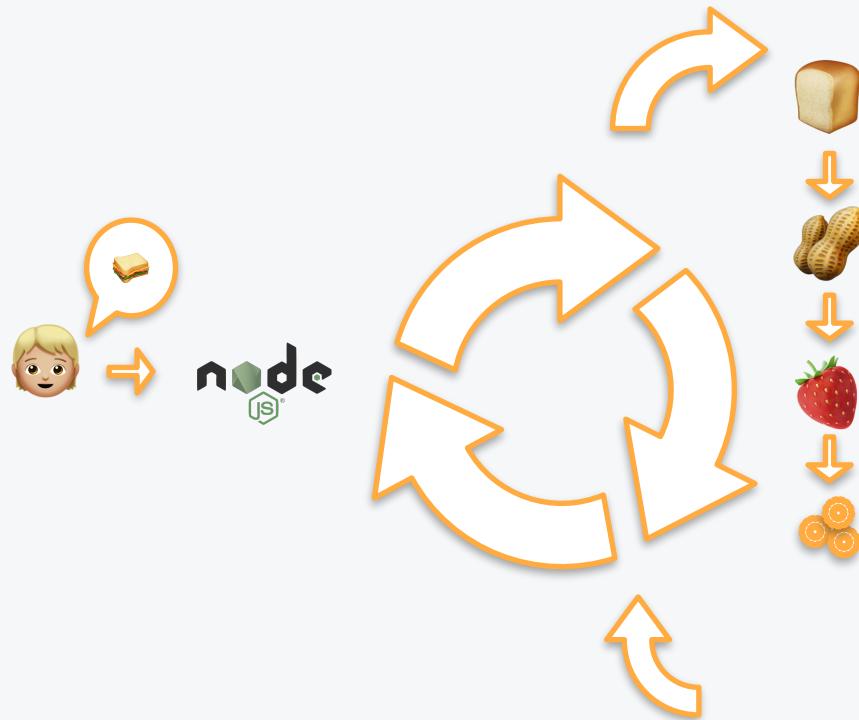
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



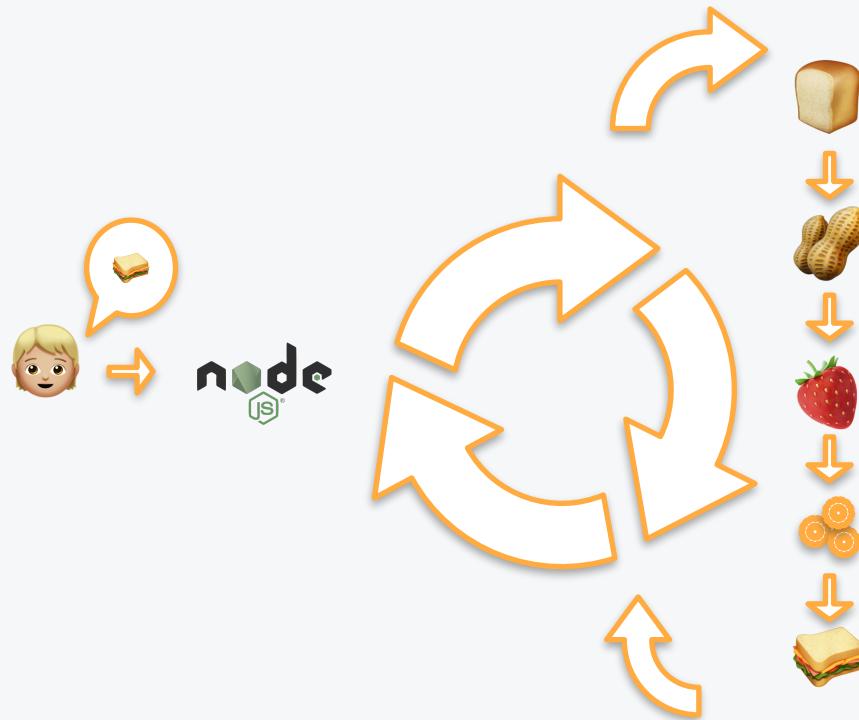
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



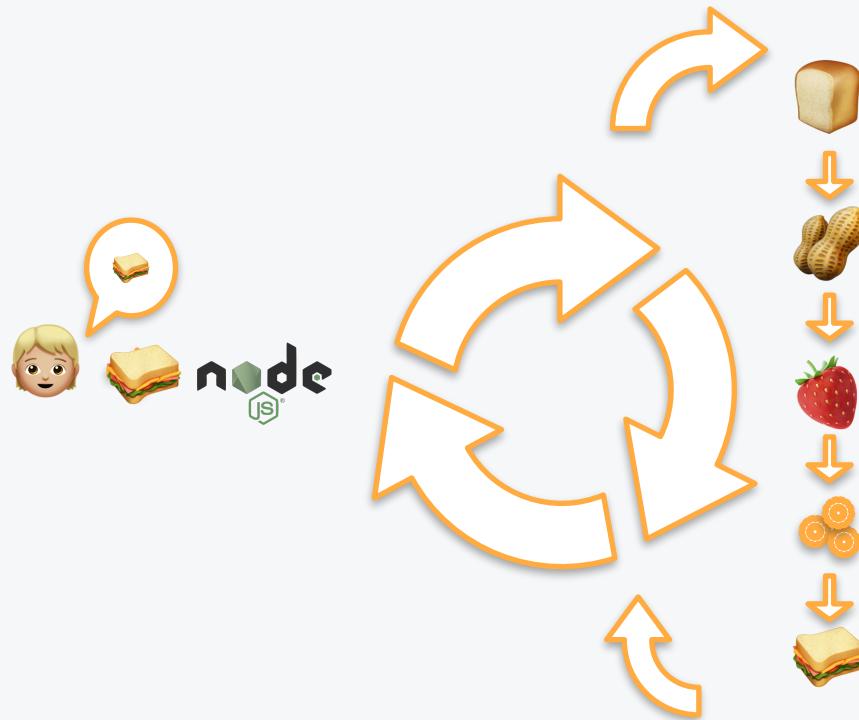
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



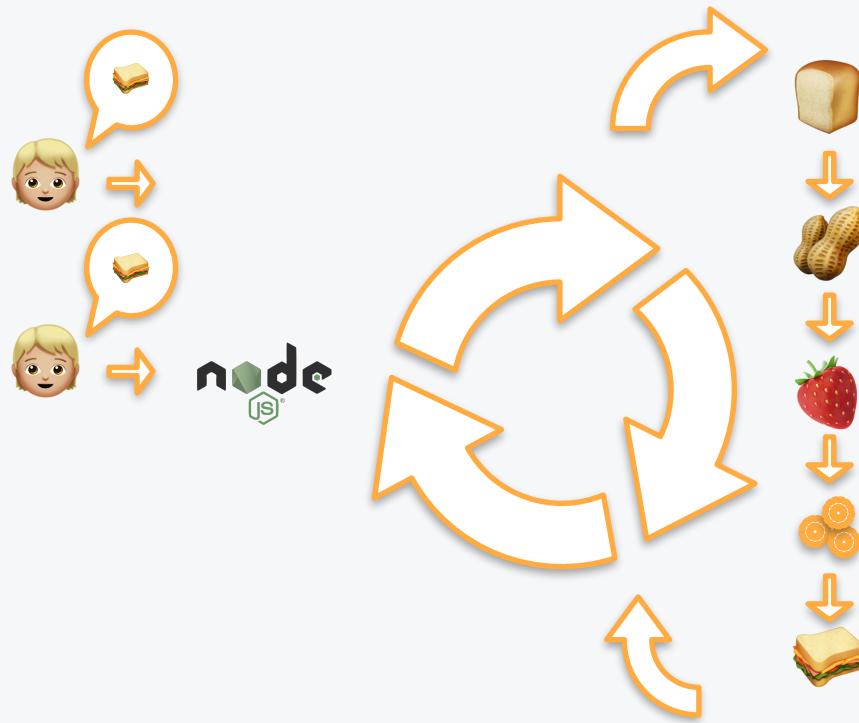
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



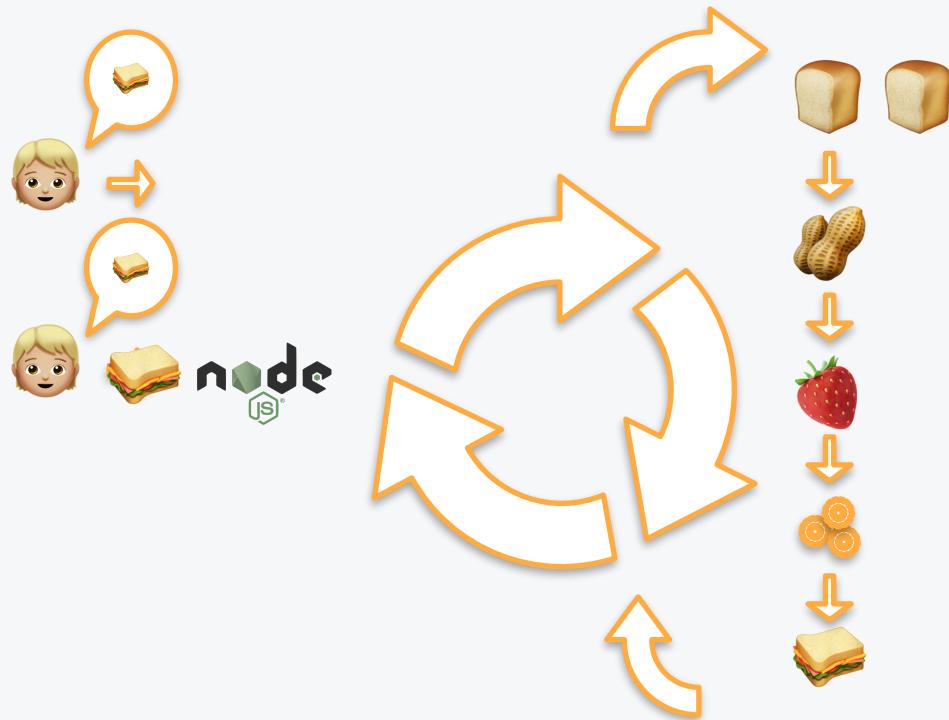
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



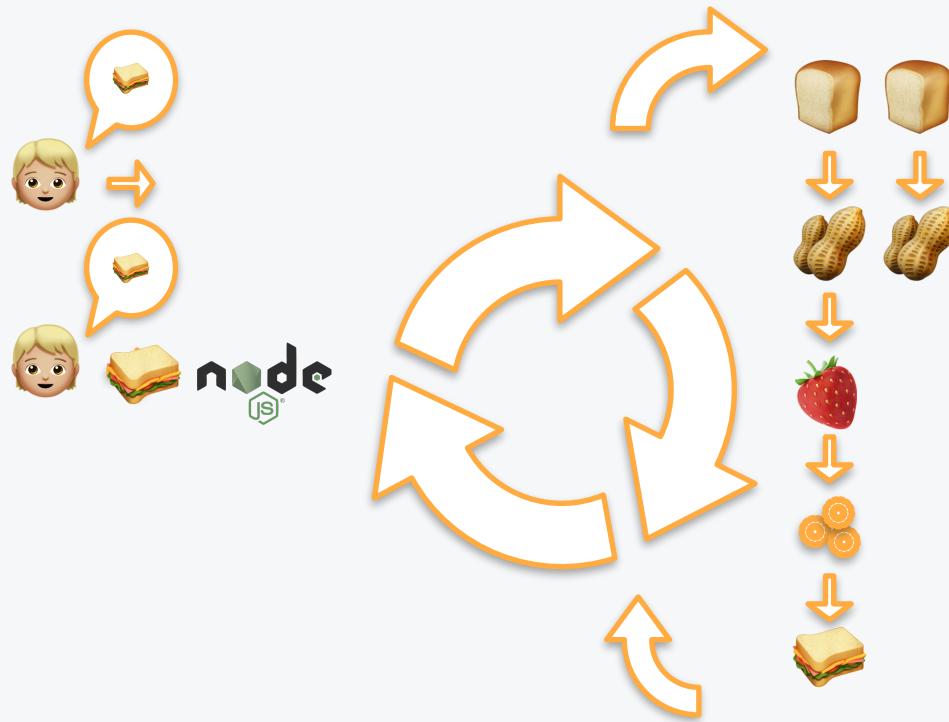
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



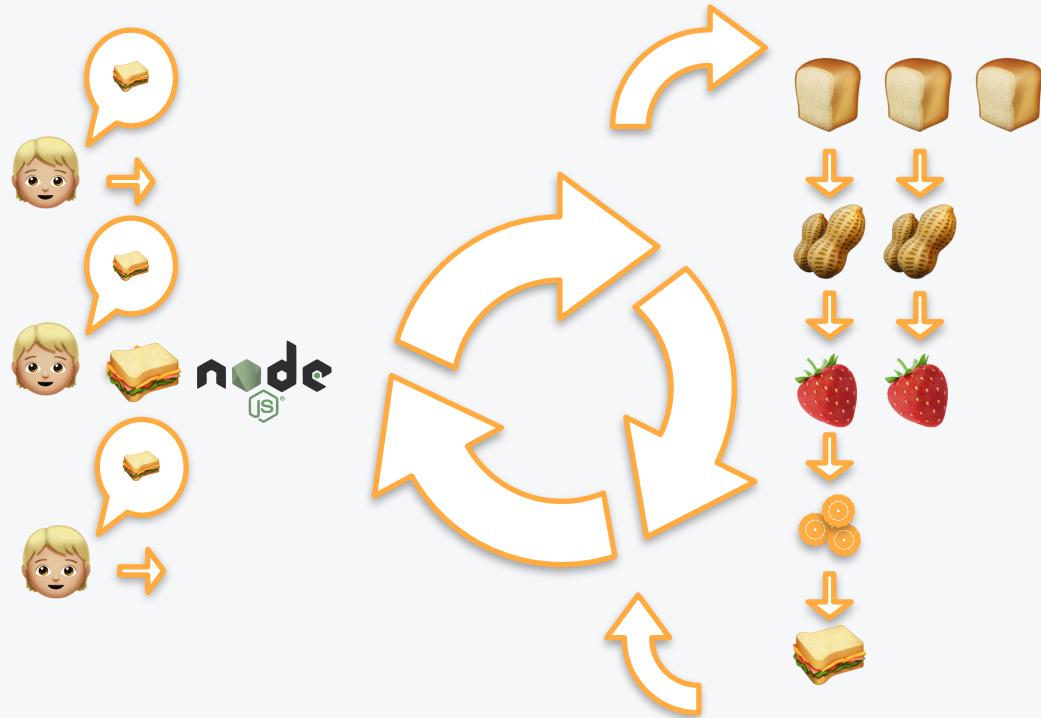
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



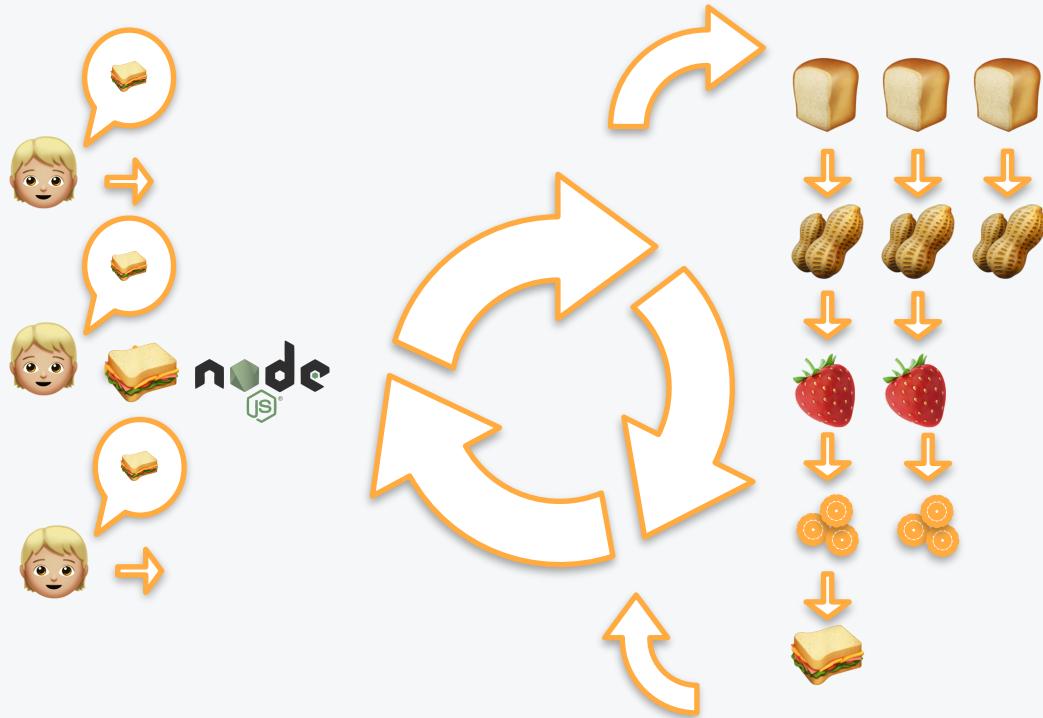
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



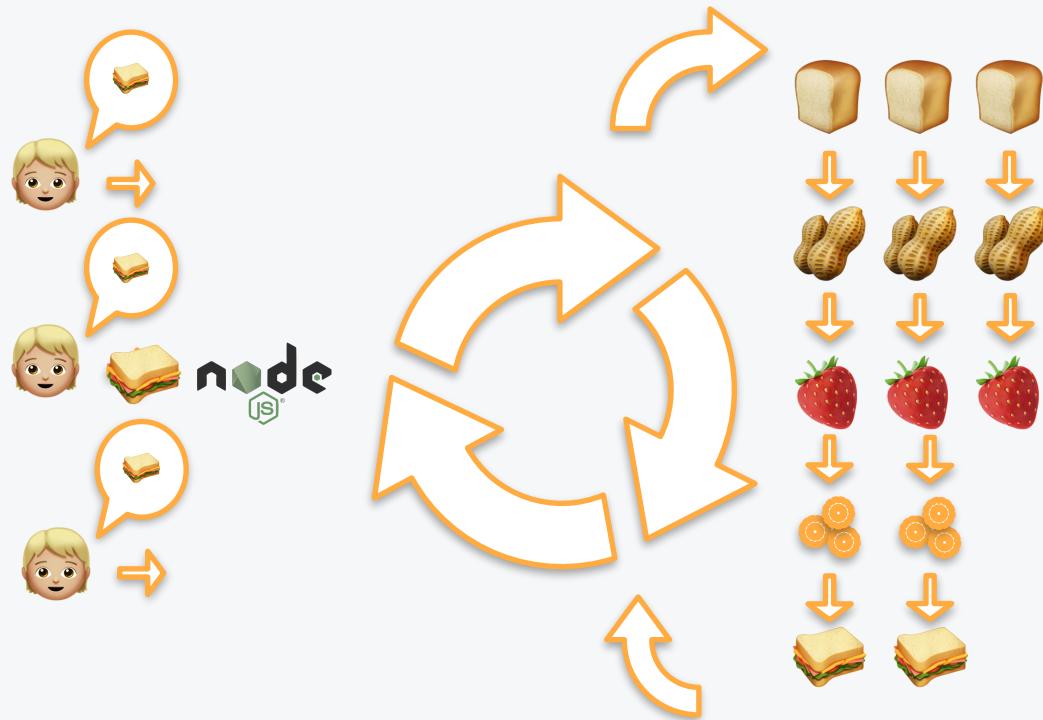
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



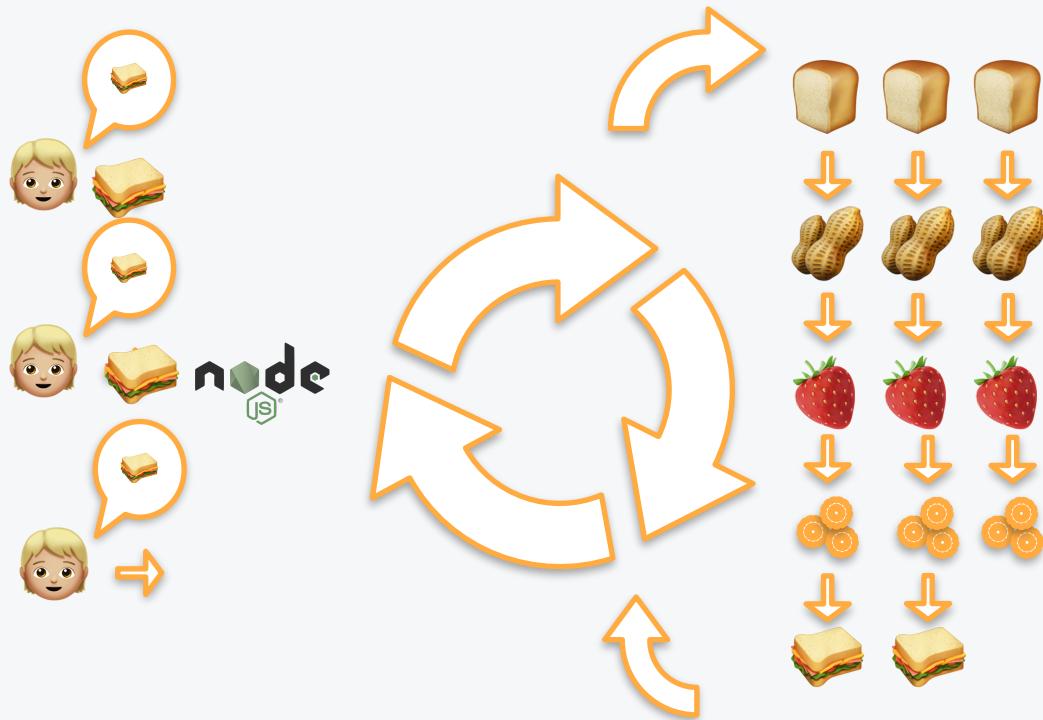
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



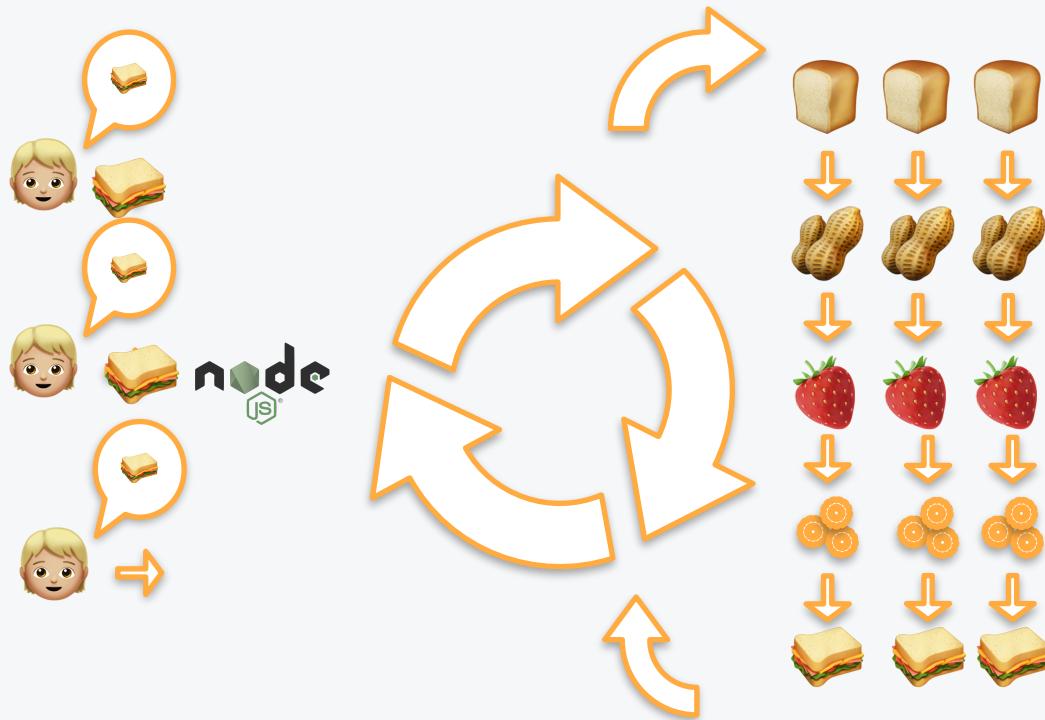
Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



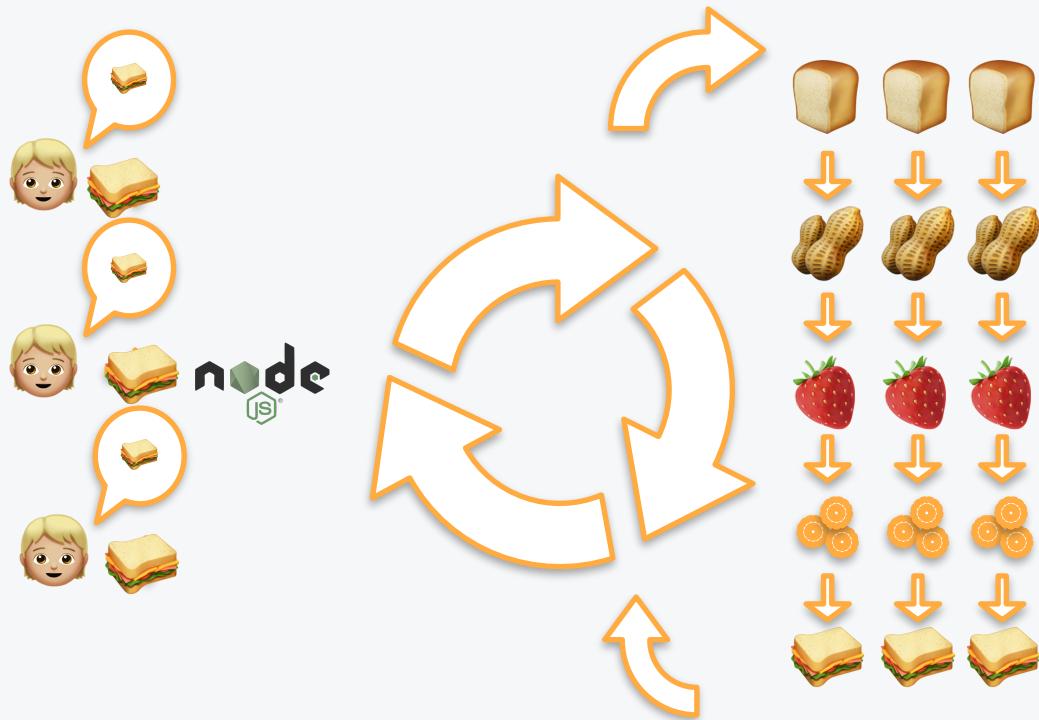
Event Loop (NodeJS)

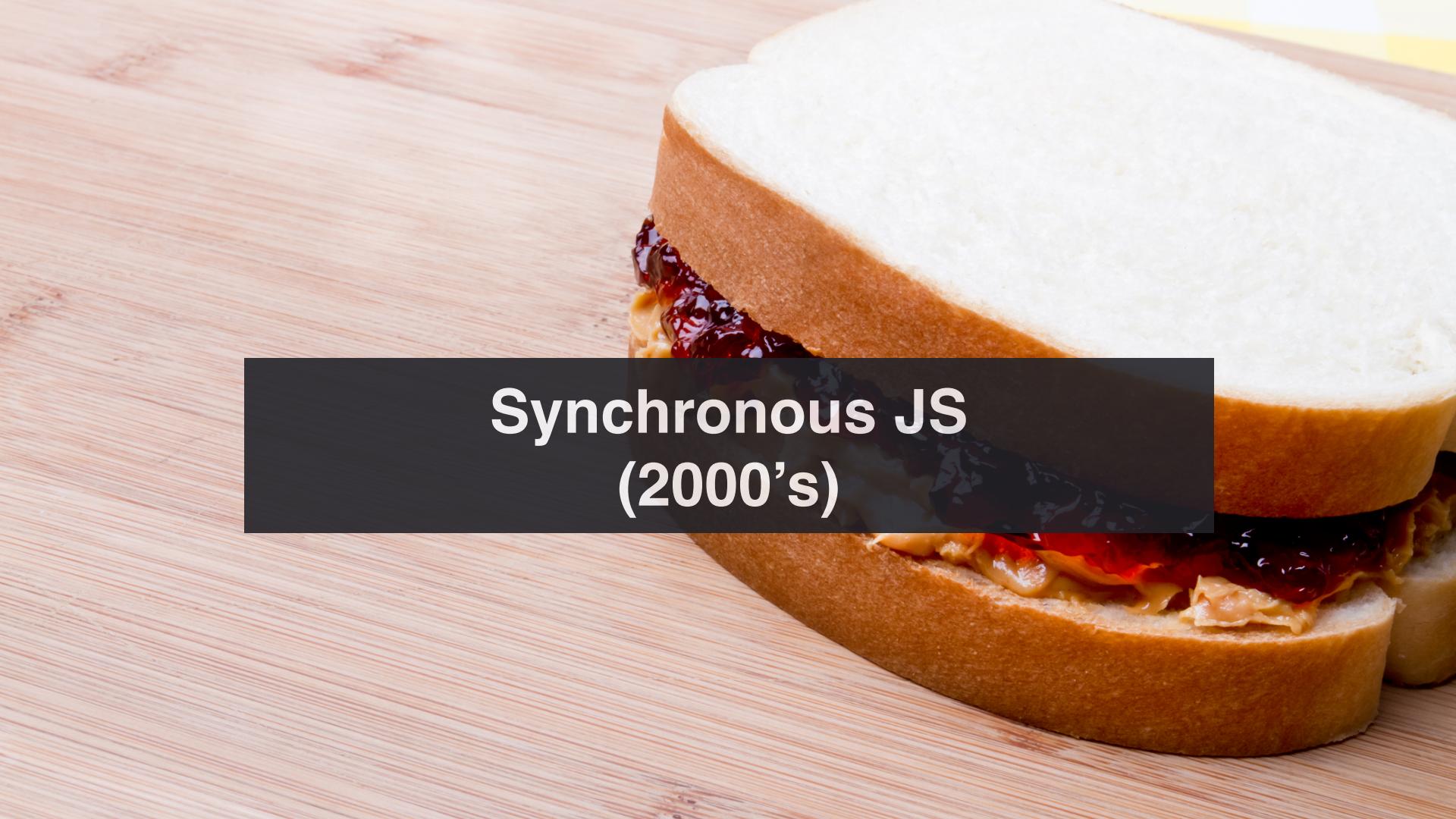
- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



Event Loop (NodeJS)

- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



A close-up photograph of a sandwich made with white bread. The sandwich is filled with peanut butter and a dark, chunky jam or jelly. It is cut in half diagonally and placed on a light-colored wooden cutting board. The lighting highlights the texture of the bread and the spread.

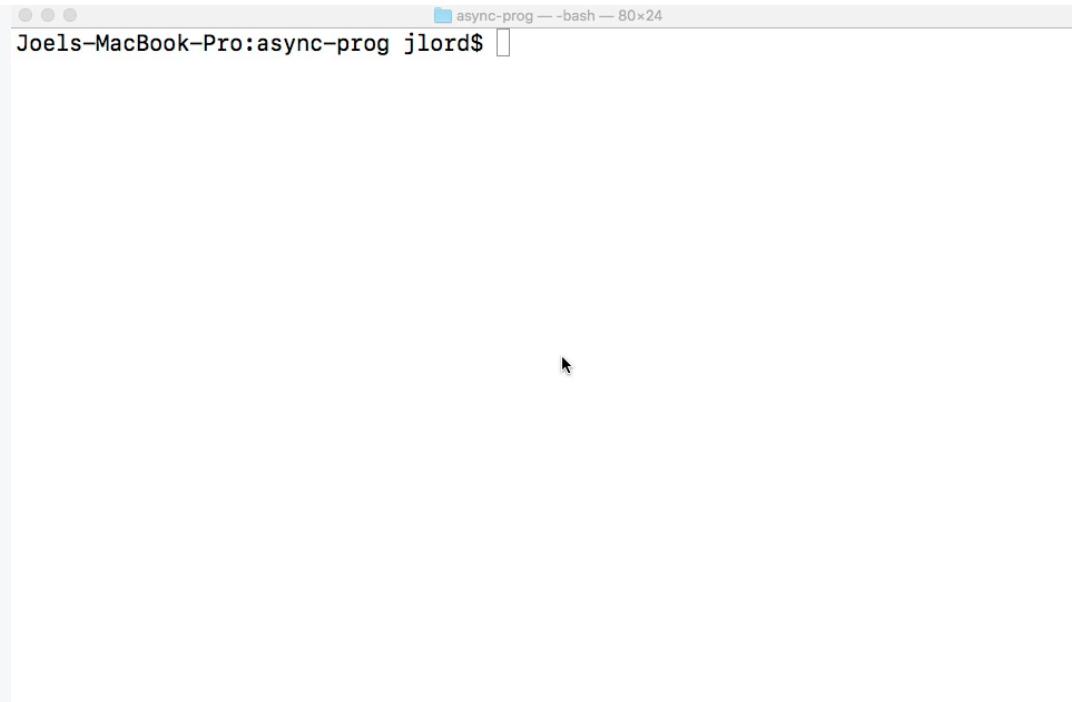
Synchronous JS (2000's)

Code

```
let PBnJMaker = require("./pbnjmaker");

PBnJMaker.fetchBread();
PBnJMaker.fetchPeanutButter();
PBnJMaker.fetchJam();
PBnJMaker.spreadPeanutButter();
PBnJMaker.spreadJam();
let result = PBnJMaker.closeSandwich();
console.log("Eat that " + result);
```

Result



async-prog — bash — 80x24
Joels-MacBook-Pro:async-prog jlord\$



A top-down photograph of a sandwich on a light-colored wooden surface. The sandwich is cut in half diagonally, showing two layers of white bread with a thick spread of peanut butter and a layer of red jelly or jam on the top slice.

Callbacks (2010's)

What is a callback

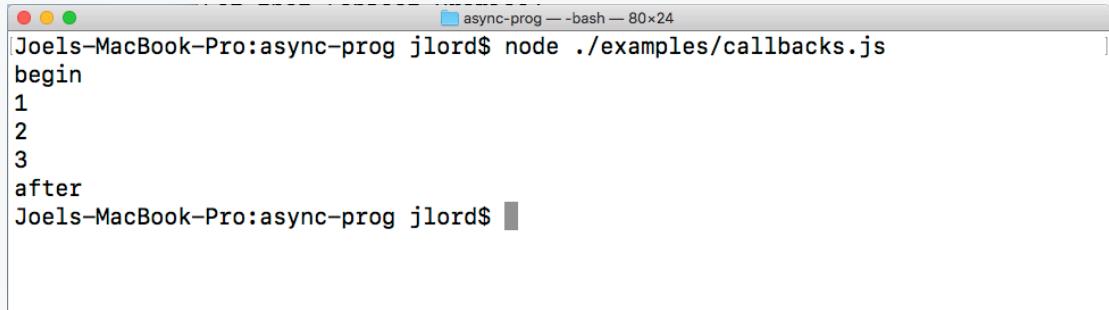
- Simply a function as an argument to a function
- Could be synchronous or asynchronous



```
1 //Synchronous
2 let array = [1,2,3];
3 console.log("begin");
4 array.map((i) => {console.log(i)});
5 console.log("after");
```

What is a callback

- Simply a function as an argument to a function
- Could be synchronous or asynchronous



```
async-prog — bash — 80x24
Joels-MacBook-Pro:async-prog jlord$ node ./examples/callbacks.js
begin
1
2
3
after
Joels-MacBook-Pro:async-prog jlord$
```

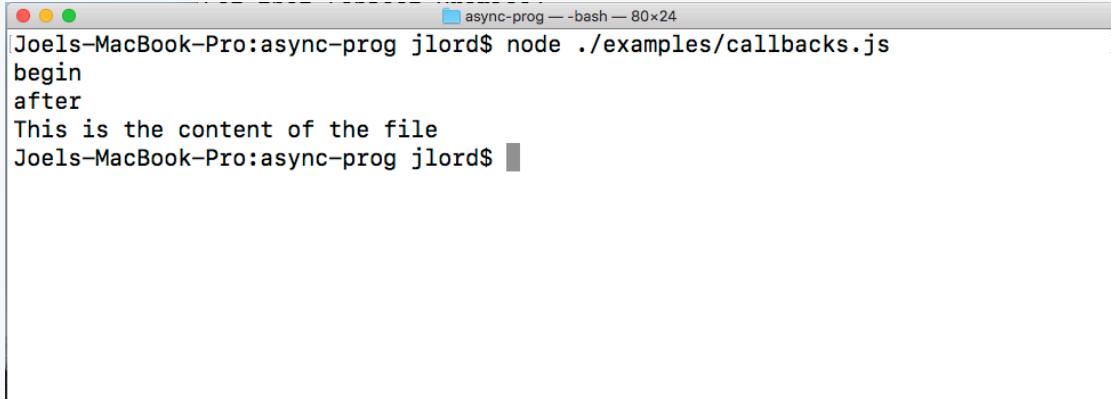
What is a callback

- Simply a function as an argument to a function
- Could be synchronous or asynchronous

```
1 //Asynchronous
2 console.log("begin");
3 fs.readFile("./cb.txt", (err, data) => {
4     console.log(data);
5 });
6 console.log("after");
```

What is a callback

- Simply a function as an argument to a function
- Could be synchronous or asynchronous



The screenshot shows a terminal window titled "async-prog — bash — 80x24". The command entered is "node ./examples/callbacks.js". The output consists of three lines of text: "begin", "after", and "This is the content of the file". The word "after" is colored orange, while the other two lines are black.

```
Joels-MacBook-Pro:async-prog jlord$ node ./examples/callbacks.js
begin
after
This is the content of the file
Joels-MacBook-Pro:async-prog jlord$
```

What is a callback

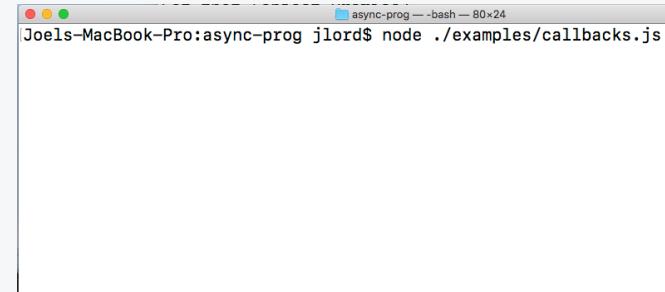
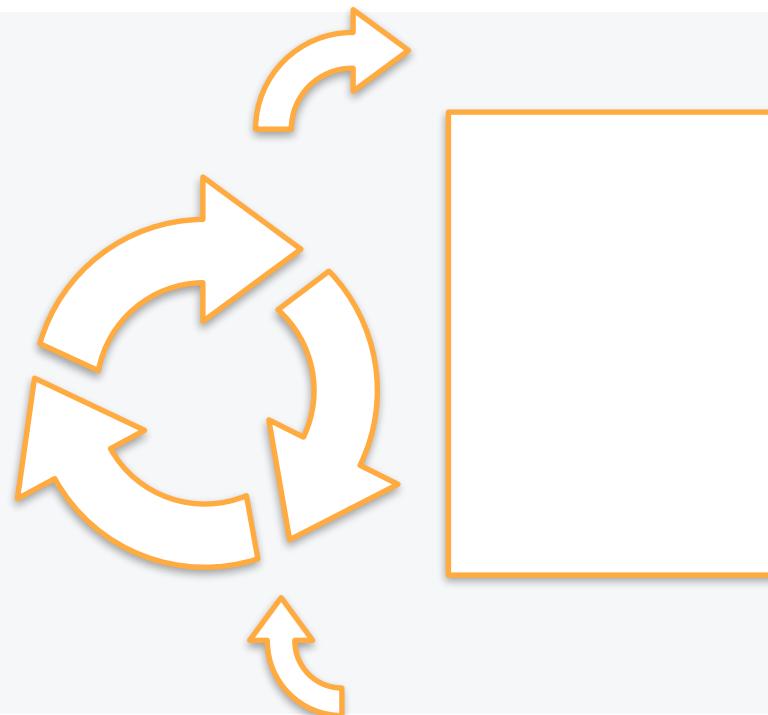
- Simply a function as an argument to a function
- Could be synchronous or asynchronous

```
1 //Asynchronous
2 .....
3 ..... (err, data) => {
4     console.log(data);
5 }..
6 .....
```

What is a callback

Asynchronous callback and the event loop

```
console.log("begin");
fs.readFile("./cb.txt",
  (err, data) => {
  console.log(data);
});
console.log("after");
```



What is a callback

Asynchronous callback and the event loop

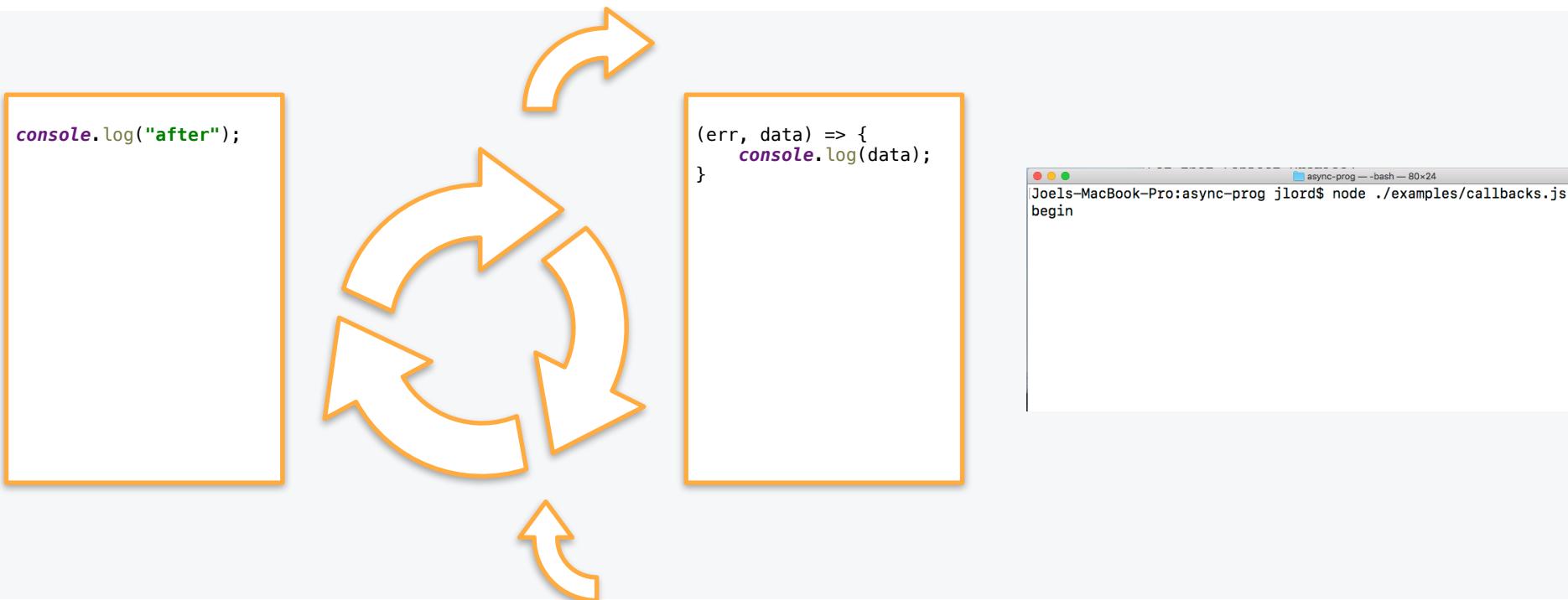
```
fs.readFile("./cb.txt",  
  (err, data) => {  
    console.log(data);  
  };  
  
console.log("after");
```

A screenshot of a terminal window titled "async-prog — bash — 80x24". The command "node ./examples/callbacks.js" is run, and the output "begin" is displayed.

@joel__lord
#confoo

What is a callback

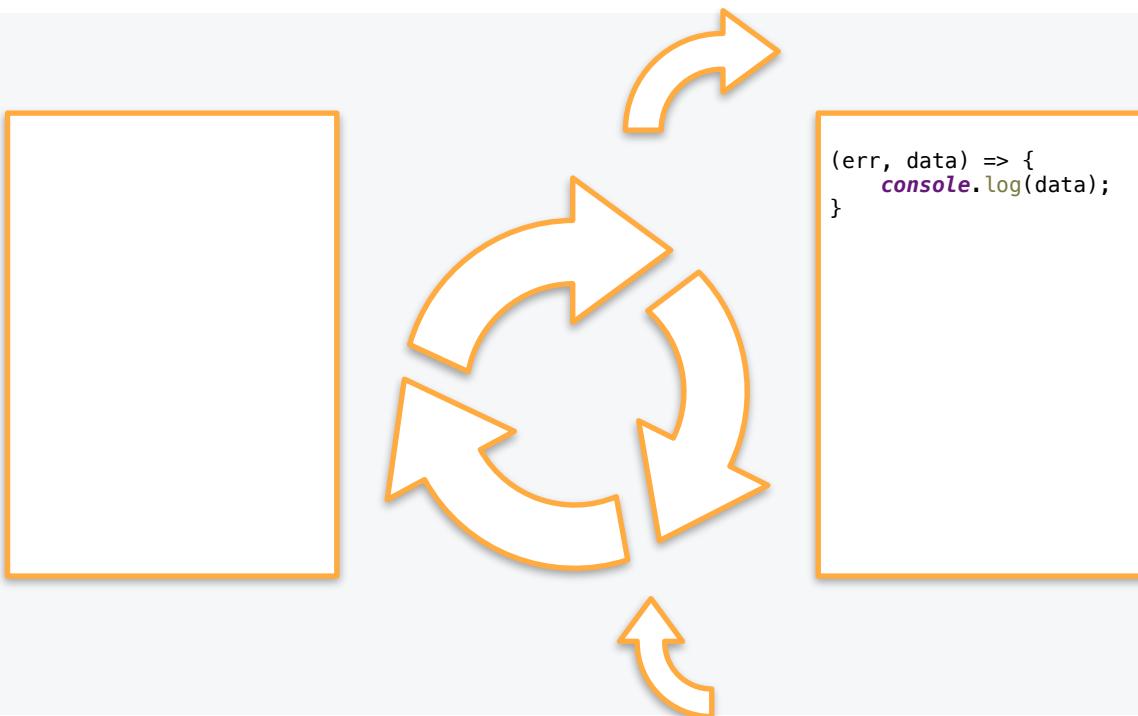
Asynchronous callback and the event loop



A screenshot of a terminal window titled "async-prog — bash — 80x24". The window shows the command "Joels-MacBook-Pro:async-prog jlord\$ node ./examples/callbacks.js" followed by the output "begin".

What is a callback

Asynchronous callback and the event loop



A screenshot of a terminal window titled "async-prog — bash — 80x24". The window shows the command "Joels-MacBook-Pro:async-prog jlord\$ node ./examples/callbacks.js" followed by the output "begin".

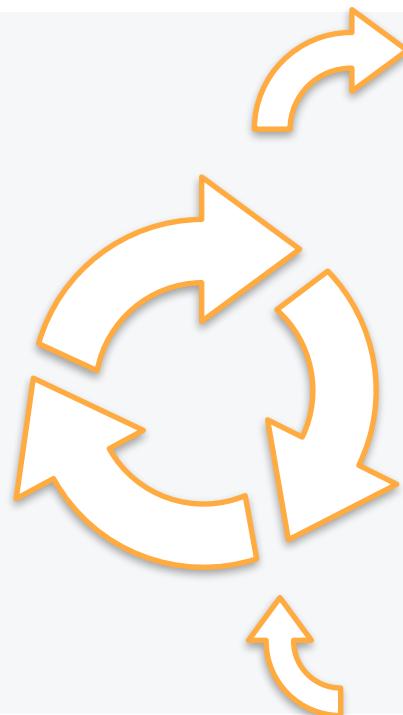


@joel__lord
#confoo

What is a callback

Asynchronous callback and the event loop

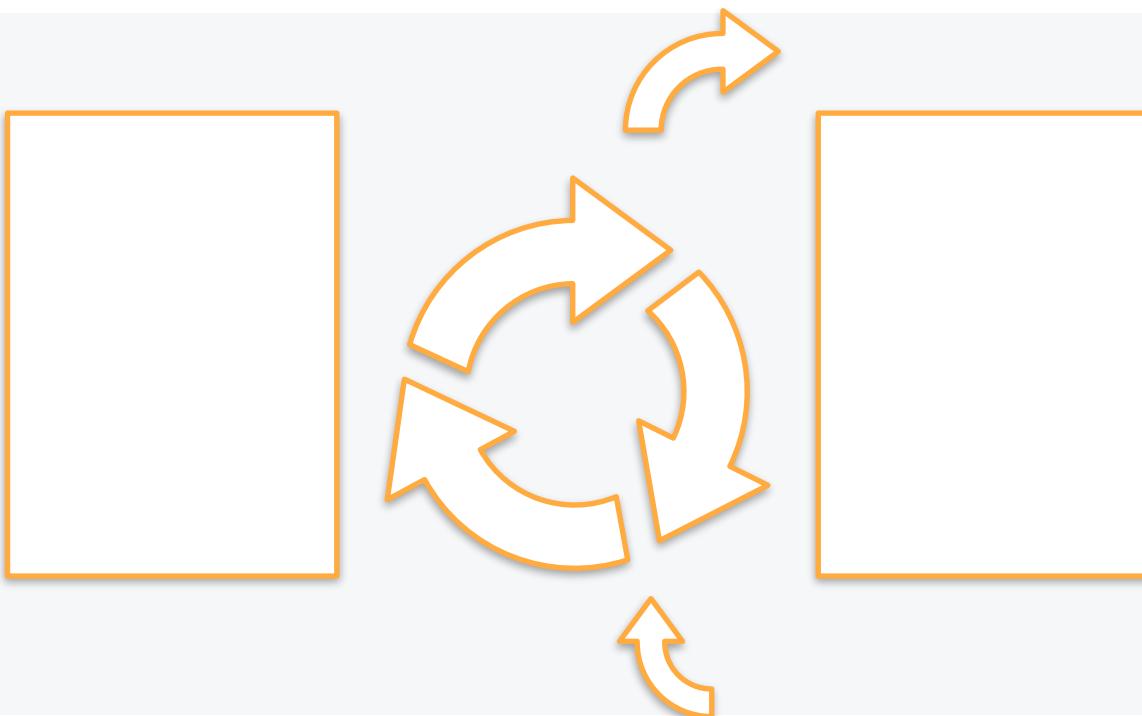
```
(err, data) => {  
  console.log(data);  
}
```



```
Joels-MacBook-Pro:async-prog jlord$ node ./examples/callbacks.js  
begin  
after
```

What is a callback

Asynchronous callback and the event loop



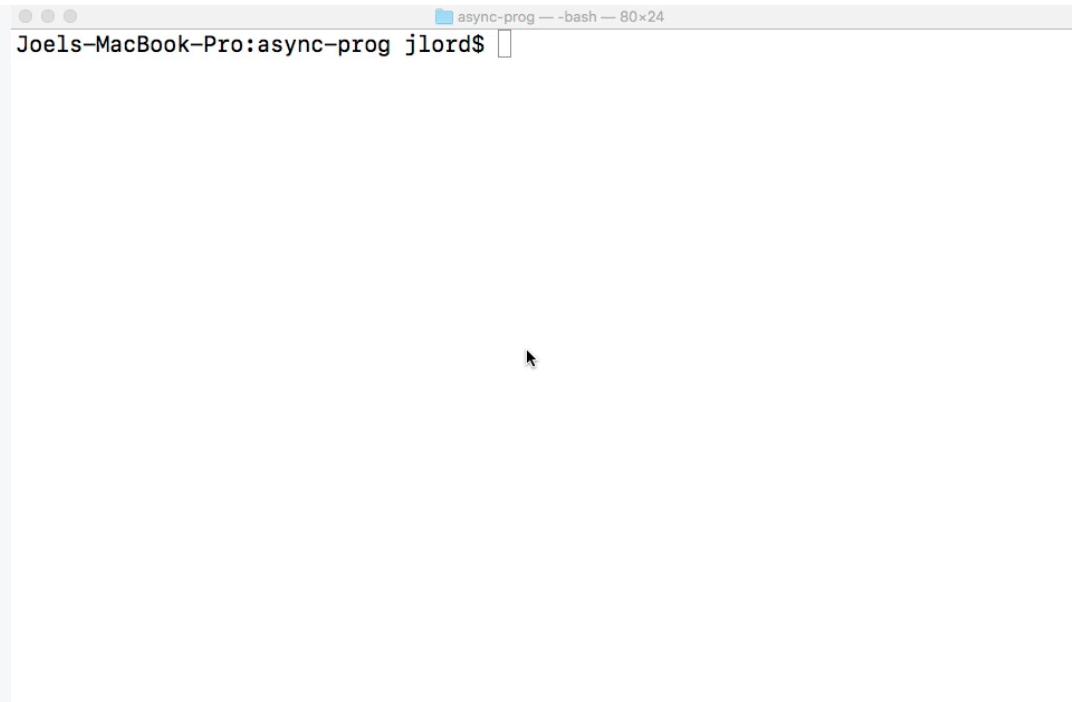
```
Joels-MacBook-Pro:async-prog jlord$ node ./examples/callbacks.js
begin
after
This is the content of the file
Joels-MacBook-Pro:async-prog jlord$
```

Code

```
let PBnJMaker = require("../pbnjmaker");

PBnJMaker.fetchBread(() => {
  PBnJMaker.fetchPeanutButter(() => {
    PBnJMaker.fetchJam(() => {
      PBnJMaker.spreadPeanutButter(() => {
        PBnJMaker.spreadJam(() => {
          PBnJMaker.closeSandwich((err, output) => {
            console.log("Eat that " + output);
          });
        });
      });
    });
  });
});
```

Result



```
async-prog — bash — 80x24
Joels-MacBook-Pro:async-prog jlord$
```

Callbacks

- It works!
- but...

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.simulateError();
4
5 PBnJMaker.fetchBread((err) => {
6   if (err) {
7     console.log("An error occurred while fetching the ingredients");
8   } else {
9     PBnJMaker.fetchPeanutButter((err) => {
10       if (err) {
11         console.log("An error occurred while fetching the ingredients");
12       } else {
13         PBnJMaker.fetchJam((err) => {
14           if (err) {
15             console.log("An error occurred while fetching the ingredients");
16           } else {
17             PBnJMaker.spreadPeanutButter((err) => {
18               if (err) {
19                 console.log("An error occurred while preparing the sandwich");
20               } else {
21                 PBnJMaker.spreadJam((err) => {
22                   if (err) {
23                     console.log("An error occurred while preparing the sandwich");
24                   } else {
25                     PBnJMaker.closeSandwich((err, result) => {
26                       if (err) {
27                         console.log("An error occurred");
28                       } else {
29                         console.log("Eat that " + result);
30                       }
31                     });
32                   });
33                 });
34               });
35             });
36           });
37         });
38       });
39     });
40   });
41 });

});
```

Callback Hell

```
let PBnJMaker = require("../pbnjmaker");

PBnJMaker.simulateError();

PBnJMaker.fetchBread((err) => {
  if (err) {
    console.log("An error occurred while fetching the ingredients");
  } else {
    PBnJMaker.fetchPeanutButter((err) => {
      if (err) {
        console.log("An error occurred while fetching the ingredients");
      } else {
        PBnJMaker.fetchJam((err) => {
          if (err) {
            console.log("An error occurred while fetching the ingredients");
          } else {
            PBnJMaker.spreadPeanutButter((err) => {
              if (err) {
                console.log("An error occurred while preparing the sandwich");
              } else {
                PBnJMaker.spreadJam((err) => {
```

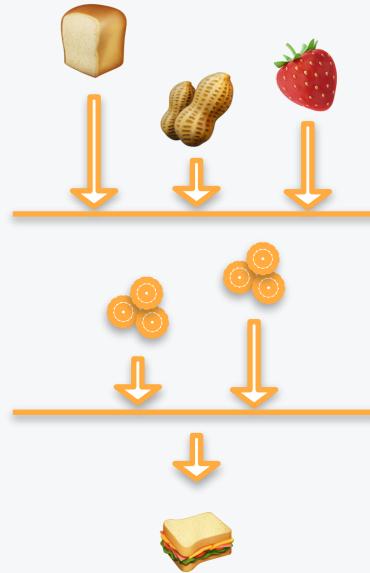
Parallel events



Step by step



Parallel events



Code

```
1 let PBnJMaker = require("./pbnjmaker");
2 let ingredients = 0;
3 let steps = 0;
4 // PBnJMaker.simulateError();
5
6 function fetchIngredients() {
7     PBnJMaker.fetchBread(handleIngredients);
8     PBnJMaker.fetchPeanutButter(handleIngredients);
9     PBnJMaker.fetchJam(handleIngredients);
10 }
11
12 function handleIngredients(err) {
13     if (err) return console.log("An error occurred while fetching the ingredients");
14     ingredients++;
15     if (ingredients === 3) {
16         prepareSandwich();
17     }
18 }
19
20 function prepareSandwich() {
21     PBnJMaker.spreadPeanutButter(preparationStepCompleted);
22     PBnJMaker.spreadJam(preparationStepCompleted);
23 }
24
25 function preparationStepCompleted(err) {
26     if (err) return console.log("An error occurred while preparing the sandwich");
27     steps++;
28     if (steps === 2) {
29         completeSandwich();
30     }
31 }
32
33 function completeSandwich(err) {
34     PBnJMaker.closeSandwich((err, result) => {
35         if (err) {
36             console.log("An error occurred");
37         } else {
38             console.log("Eat that " + result);
39         }
40     });
41 }
42
43 fetchIngredients();
```

Result

```
4 PBnJMaker — ./pbnjmaker
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
```

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2 let ingredients = 0;
3 let steps = 0;
4 PBnJMaker.fetchBread(handleBread());
5
6 function fetchIngredients() {
7   PBnJMaker.fetchBread(handleIngredients);
8   PBnJMaker.fetchPeanutButter(handleIngredients);
9   PBnJMaker.fetchJam(handleIngredients);
10 }
11
12 function handleIngredients(err) {
13   if (err) return console.log("An error occurred while fetching the ingredients");
14   ingredients++;
15   if (ingredients === 3) {
16     prepareSandwich();
17   }
18 }
19
20 function prepareSandwich() {
21   PBnJMaker.spreadPeanutButter(preparationStepCompleted);
22   PBnJMaker.spreadJam(preparationStepCompleted);
23 }
24
25 function preparationStepCompleted(err) {
26   if (err) return console.log("An error occurred while preparing the sandwich");
27   steps++;
28   if (steps === 2) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(err) {
34   PBnJMaker.closeSandwich((err, result) => {
35     if (err) {
36       console.log("An error occurred");
37     } else {
38       console.log("Eat that " + result);
39     }
40   });
41 }
42
43 fetchIngredients();
```

Result

```
Joels-MacBook-Pro:async-prog jlord$
```

Code

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2 let ingredients = 0;  
3 let steps = 0;  
4 // PBnJMaker.simulateError();  
5 function fetchIngredients() {  
6     PBnJMaker.fetchBread(handleIngredients);  
7     PBnJMaker.fetchPeanutButter(handleIngredients);  
8     PBnJMaker.fetchJam(handleIngredients);  
9 }  
10 }  
11  
12 function handleIngredients(err) {  
13     if (err) return console.log("An error occurred while fetching the ingredients");  
14     ingredients++;  
15     if (ingredients === 3) {  
16         prepareSandwich();  
17     }  
18 }  
19  
20 function prepareSandwich() {  
21     PBnJMaker.spreadPeanutButter(preparationStepCompleted);  
22     PBnJMaker.spreadJam(preparationStepCompleted);  
23 }  
24  
25 function preparationStepCompleted(err) {  
26     if (err) return console.log("An error occurred while preparing the sandwich");  
27     steps++;  
28     if (steps === 2) {  
29         completeSandwich();  
30     }  
31 }  
32  
33 function completeSandwich(err) {  
34     PBnJMaker.closeSandwich((err, result) => {  
35         if (err) {  
36             console.log("An error occurred");  
37         } else {  
38             console.log("Eat that " + result);  
39         }  
40     });  
41 }  
42  
43 fetchIngredients();
```

Result

```
● ● ●  
Joels-MacBook-Pro:async-prog jlord$
```

Code

```
● ● ●
1 let PBnJMaker = require("./pbnjmaker");
2 let ingredients = 0;
3 let steps = 0;
4 // PBnJMaker.simulateError();
5
6 function fetchIngredients() {
7     PBnJMaker.fetchBread(handleIngredients);
8     PBnJMaker.fetchPeanutButter(handleIngredients);
9     PBnJMaker.fetchJam(handleIngredients);
10 }
11
12 function handleIngredients(err) {
13     if (err) return console.log("An error occurred while fetching the ingredients");
14     ingredients++;
15     if (ingredients === 3) {
16         prepareSandwich();
17     }
18 }
19
20 function prepareSandwich() {
21     PBnJMaker.spreadPeanutButter(preparationStepCompleted);
22     PBnJMaker.spreadJam(preparationStepCompleted);
23 }
24
25 function preparationStepCompleted(err) {
26     if (err) return console.log("An error occurred while preparing the sandwich");
27     steps++;
28     if (steps === 2) {
29         completeSandwich();
30     }
31 }
32
33 function completeSandwich(err) {
34     PBnJMaker.closeSandwich((err, result) => {
35         if (err) {
36             console.log("An error occurred");
37         } else {
38             console.log("Eat that " + result);
39         }
40     });
41 }
42
43 fetchIngredients();
```

Result

```
● ● ●
async-prog — bash — 80x24
Joels-MacBook-Pro:async-prog jlord$
```

Code

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2 let ingredients = 0;  
3 let steps = 0;  
4 // PBnJMaker.simulateError();  
5  
6 function fetchIngredients() {  
7   PBnJMaker.fetchBread(handleIngredients);  
8   PBnJMaker.fetchPeanutButter(handleIngredients);  
9   PBnJMaker.fetchJam(handleIngredients);  
10 }  
11  
12 function handleIngredients(err) {  
13   if (err) return console.log("An error occurred while fetching the ingredients");  
14   ingredients++;  
15   if (ingredients === 3) {  
16     prepareSandwich();  
17   }  
18 }  
19  
20 function prepareSandwich() {  
21   PBnJMaker.spreadPeanutButter(preparationStepCompleted);  
22   PBnJMaker.spreadJam(preparationStepCompleted);  
23 }  
24  
25 function preparationStepCompleted(err) {  
26   if (err) return console.log("An error occurred while preparing the sandwich");  
27   steps++;  
28   if (steps === 2) {  
29     completeSandwich();  
30   }  
31 }  
32  
33 function completeSandwich(err) {  
34   PBnJMaker.closeSandwich((err, result) => {  
35     if (err) {  
36       console.log("An error occurred");  
37     } else {  
38       console.log("Eat that " + result);  
39     }  
40   });  
41 }  
42  
43 fetchIngredients();
```

Result

```
● ● ●  
4 PBnJMaker — bash — 80x24  
Joels-MacBook-Pro:async-prog jlord$
```

Code

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2 let ingredients = 0;  
3 let steps = 0;  
4 // PBnJMaker.simulateError();  
5  
6 function fetchIngredients() {  
7     PBnJMaker.fetchBread(handleIngredients);  
8     PBnJMaker.fetchPeanutButter(handleIngredients);  
9     PBnJMaker.fetchJam(handleIngredients);  
10 }  
11  
12 function handleIngredients(err) {  
13     if (err) return console.log("An error occurred while fetching the ingredients");  
14     ingredients++;  
15     if (ingredients === 3) {  
16         prepareSandwich();  
17     }  
18 }  
19 function prepareSandwich() {  
20     PBnJMaker.spreadPeanutButter(preparationStepCompleted);  
21     PBnJMaker.spreadJam(preparationStepCompleted);  
22 }  
23  
24 function preparationStepCompleted(err) {  
25     if (err) return console.log("An error occurred while preparing the sandwich");  
26     steps++;  
27     if (steps === 2) {  
28         completeSandwich();  
29     }  
30 }  
31  
32  
33 function completeSandwich(err) {  
34     PBnJMaker.closeSandwich((err, result) => {  
35         if (err) {  
36             console.log("An error occurred");  
37         } else {  
38             console.log("Eat that " + result);  
39         }  
40     });  
41 }  
42  
43 fetchIngredients();
```

Result

```
● ● ●  
Joels-MacBook-Pro:async-prog jlord$  
I
```

Code

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2 let ingredients = 0;  
3 let steps = 0;  
4 // PBnJMaker.simulateError();  
5  
6 function fetchIngredients() {  
7     PBnJMaker.fetchBread(handleIngredients);  
8     PBnJMaker.fetchPeanutButter(handleIngredients);  
9     PBnJMaker.fetchJam(handleIngredients);  
10 }  
11  
12 function handleIngredients(err) {  
13     if (err) return console.log("An error occurred while fetching the ingredients");  
14     ingredients++;  
15     if (ingredients === 3) {  
16         prepareSandwich();  
17     }  
18 }  
19  
20 function prepareSandwich() {  
21     PBnJMaker.spreadPeanutButter(preparationStepCompleted);  
22     PBnJMaker.spreadJam(preparationStepCompleted);  
23 }  
24  
25 function preparationStepCompleted(err) {  
26     if (err) return console.log("An error occurred while preparing the sandwich");  
27     steps++;  
28     if (steps === 2) {  
29         completeSandwich();  
30     }  
31 }  
32  
33 function completeSandwich(err) {  
34     PBnJMaker.closeSandwich((err, result) => {  
35         if (err) {  
36             console.log("An error occurred");  
37         } else {  
38             console.log("Eat that " + result);  
39         }  
40     });  
41 }  
42  
43 fetchIngredients();
```

Result

```
● ● ●  
Joels-MacBook-Pro:async-prog jlord$  
I
```

Code

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2 let ingredients = 0;  
3 let steps = 0;  
4 // PBnJMaker.simulateError();  
5  
6 function fetchIngredients() {  
7   PBnJMaker.fetchBread(handleIngredients);  
8   PBnJMaker.fetchPeanutButter(handleIngredients);  
9   PBnJMaker.fetchJam(handleIngredients);  
10 }  
11  
12 function handleIngredients(err) {  
13   if (err) return console.log("An error occurred while fetching the ingredients");  
14   ingredients++;  
15   if (ingredients === 3) {  
16     prepareSandwich();  
17   }  
18 }  
19  
20 function prepareSandwich() {  
21   PBnJMaker.spreadPeanutButter(preparationStepCompleted);  
22   PBnJMaker.spreadJam(preparationStepCompleted);  
23 }  
24  
25 function preparationStepCompleted(err) {  
26   if (err) return console.log("An error occurred while preparing the sandwich");  
27   steps++;  
28   if (steps === 2) {  
29     completeSandwich();  
30   }  
31 }  
32  
33 function completeSandwich(err) {  
34   PBnJMaker.closeSandwich((err, result) => {  
35     if (err) {  
36       console.log("An error occurred");  
37     } else {  
38       console.log("Eat that " + result);  
39     }  
40   });  
41 }  
42  
43 fetchIngredients();
```

Result

```
● ● ●  
Joels-MacBook-Pro:async-prog jlord$  
I
```

Code

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2 let ingredients = 0;  
3 let steps = 0;  
4 // PBnJMaker.simulateError();  
5  
6 function fetchIngredients() {  
7     PBnJMaker.fetchBread(handleIngredients);  
8     PBnJMaker.fetchPeanutButter(handleIngredients);  
9     PBnJMaker.fetchJam(handleIngredients);  
10 }  
11  
12 function handleIngredients(err) {  
13     if (err) return console.log("An error occurred while fetching the ingredients");  
14     ingredients++;  
15     if (ingredients === 3) {  
16         prepareSandwich();  
17     }  
18 }  
19  
20 function prepareSandwich() {  
21     PBnJMaker.spreadPeanutButter(preparationStepCompleted);  
22     PBnJMaker.spreadJam(preparationStepCompleted);  
23 }  
24  
25 function preparationStepCompleted(err) {  
26     if (err) return console.log("An error occurred while preparing the sandwich");  
27     steps++;  
28     if (steps === 2) {  
29         completeSandwich();  
30     }  
31 }  
32  
33 function completeSandwich(err) {  
34     PBnJMaker.closeSandwich((err, result) => {  
35         if (err) {  
36             console.log("An error occurred");  
37         } else {  
38             console.log("Eat that " + result);  
39         }  
40     });  
41 }  
42 fetchIngredients();  
43
```

Result

```
● ● ●  
Joels-MacBook-Pro:async-prog jlord$  
I
```



Promises
(2014)

What is a promise?

- A representation of a callback
- Returns a `.then()` method
- Error handling simplified



```
1 //Asynchronous  
2 console.log("begin");  
3 asyncPromise().then(onSuccess, onFailure);  
4 console.log("after");
```

What is a promise?

- A representation of a callback
- Returns a .then() method
- Error handling simplified



```
1 //Asynchronous
2 console.log("begin");
3 fs.readFile("./cb.txt")
4   .then((data) => {
5     console.log("This is the content of the file");
6   }, (err) => {
7     console.log("An error occurred");
8 });
9 console.log("after");
```

What is a promise?

- A representation of a callback
- Returns a .then() method
- Error handling simplified

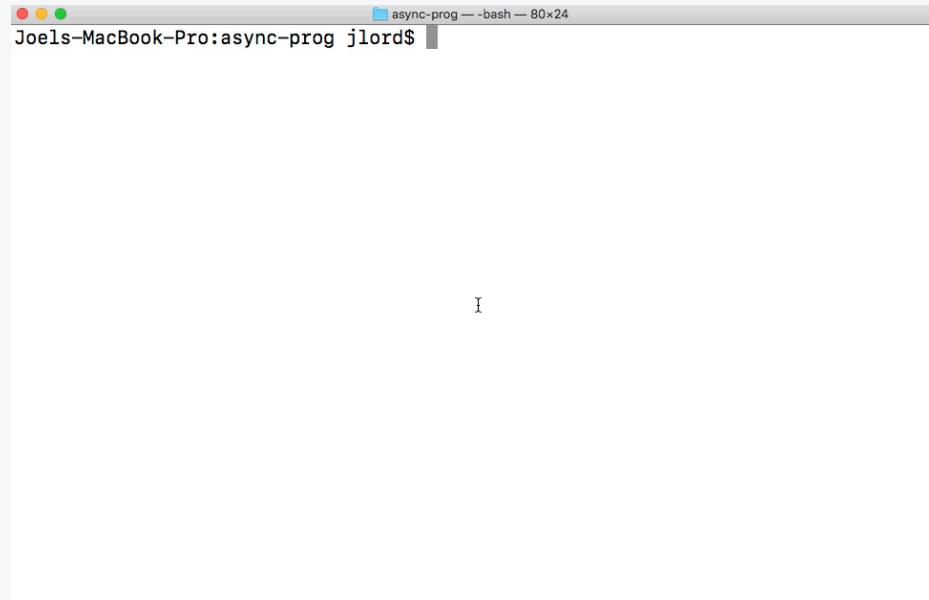


```
1 //Asynchronous
2 console.log("begin");
3 fs.readFile("./cb.txt");
4   .then((data) => {
5     console.log("This is the content of the file");
6   })
7   .catch((err) => {
8     console.log("An error occurred");
9   });
10 console.log("after");
```

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window shows the command "Joels-MacBook-Pro:async-prog jlord\$". Below the title bar, there are three colored window control buttons (red, yellow, green). The terminal itself is empty, indicating that the code has been run but no output is visible.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window shows the command "Joels-MacBook-Pro:async-prog jlord\$". Below the title bar, there are three colored window control buttons (red, yellow, green). The terminal itself is empty, indicating that the code has been run but no output is visible.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a macOS terminal window titled "async-prog — bash — 80x24". The window is located on a MacBook Pro, as indicated by the title bar. The command "jlord\$" is visible at the bottom of the terminal. The terminal is currently empty, showing only the window title and prompt.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window shows the command "Joels-MacBook-Pro:async-prog jlord\$". Below the title bar, there are three colored window control buttons (red, yellow, green). The terminal itself is empty, indicating that the code has been run but no output is visible.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window shows the command "Joels-MacBook-Pro:async-prog jlord\$". Below the title bar, there are three colored window control buttons (red, yellow, green). The terminal itself is empty, indicating that the code has been run but no output is visible.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window shows the command "Joels-MacBook-Pro:async-prog jlord\$". Below the title bar, there are three colored window control buttons (red, yellow, green). The terminal itself is empty, indicating that the code has been run but no output is visible.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a macOS terminal window titled "async-prog — bash — 80x24". The window is located on a MacBook Pro, as indicated by the title bar. The command "jlord\$" is visible at the bottom. The terminal is currently empty, showing only the window title and prompt.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14    console.log("Eat that " + result);
15 });
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window shows the command "Joels-MacBook-Pro:async-prog jlord\$". Below the title bar, there are three colored dots (red, yellow, green) and a small icon. The terminal itself is empty, displaying only the prompt.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread().then(() => {
4     return PBnJMaker.fetchPeanutButter();
5 }).then(() => {
6     return PBnJMaker.fetchJam();
7 }).then(() => {
8     return PBnJMaker.spreadPeanutButter();
9 }).then(() => {
10    return PBnJMaker.spreadJam();
11 }).then(() => {
12    return PBnJMaker.closeSandwich();
13 }).then((result) => {
14     console.log("Eat that " + result);
15 });
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window shows the command "Joels-MacBook-Pro:async-prog jlord\$". Below the title bar, there are three colored window control buttons (red, yellow, green). The terminal itself is empty, indicating that the code has been run but no output is visible.

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.simulateError();
4
5 PBnJMaker.fetchBread().then(() => {
6   return PBnJMaker.fetchPeanutButter();
7 }).then(() => {
8   return PBnJMaker.fetchJam();
9 }).catch((err) => {
10   console.log("Error fetching the ingredients");
11 }).then(() => {
12   return PBnJMaker.spreadPeanutButter();
13 }).then(() => {
14   return PBnJMaker.spreadJam();
15 }).catch((err) => {
16   console.log("An error occurred while preparing the sandwich");
17 }).then(() => {
18   return PBnJMaker.closeSandwich();
19 }).then((result) => {
20   console.log("Eat that " + result);
21 }).catch((err) => {
22   console.log("An error occurred");
23 });
```

Result

```
Joels-MacBook-Pro:async-prog jlord$
```

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.simulateError();
4
5 PBnJMaker.fetchBread().then(() => {
6   return PBnJMaker.fetchPeanutButter();
7 }).then(() => {
8   return PBnJMaker.fetchJam();
9 }).catch((err) => {
10   console.log("Error fetching the ingredients");
11 }).then(() => {
12   return PBnJMaker.spreadPeanutButter();
13 }).then(() => {
14   return PBnJMaker.spreadJam();
15 }).catch((err) => {
16   console.log("An error occurred while preparing the sandwich");
17 }).then(() => {
18   return PBnJMaker.closeSandwich();
19 }).then((result) => {
20   console.log("Eat that " + result);
21 }).catch((err) => {
22   console.log("An error occurred");
23 });
```

Result

```
Joels-MacBook-Pro:async-prog jlord$
```

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.simulateError();
4
5 PBnJMaker.fetchBread().then(() => {
6   return PBnJMaker.fetchPeanutButter();
7 }).then(() => {
8   return PBnJMaker.fetchJam();
9 }).catch((err) => {
10   console.log("Error fetching the ingredients");
11 }).then(() => {
12   return PBnJMaker.spreadPeanutButter();
13 }).then(() => {
14   return PBnJMaker.spreadJam();
15 }).catch((err) => {
16   console.log("An error occurred while preparing the sandwich");
17 }).then(() => {
18   return PBnJMaker.closeSandwich();
19 }).then((result) => {
20   console.log("Eat that " + result);
21 }).catch((err) => {
22   console.log("An error occurred");
23});
```

Result

```
Joels-MacBook-Pro:async-prog jlord$
```

Aggregate functions

- `.all()`
- `.race()`

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2  
3 Promise.all([PBnJMaker.fetchBread(), PBnJMaker.fetchPeanutButter(), PBnJMaker.fetchJam()])  
4   .then(() => {  
5     return Promise.all([PBnJMaker.spreadPeanutButter(), PBnJMaker.spreadJam()]);  
6   }).then(() => {  
7     return PBnJMaker.closeSandwich();  
8   }).then((result) => {  
9     console.log("Eat that " + result);  
10  });  
11
```

Aggregate functions

- `.all()`
- `.race()`

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2  
3 Promise.all([PBnJMaker.fetchBread(), PBnJMaker.fetchPeanutButter(), PBnJMaker.fetchJam()])  
4   .then(() => {  
5     return Promise.all([PBnJMaker.spreadPeanutButter(), PBnJMaker.spreadJam()]);  
6   }).then(() => {  
7     return PBnJMaker.closeSandwich();  
8   }).then((result) => {  
9     console.log("Eat that " + result);  
10  });  
11
```

Aggregate functions

- `.all()`
- `.race()`

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 Promise.all([PBnJMaker.fetchBread(), PBnJMaker.fetchPeanutButter(), PBnJMaker.fetchJam()])
4   .then(() => {
5     return Promise.all([PBnJMaker.spreadPeanutButter(), PBnJMaker.spreadJam()]);
6   }).then(() => {
7     return PBnJMaker.closeSandwich();
8   }).then((result) => {
9     console.log("Eat that " + result);
10  });
11
```

Aggregate functions

- `.all()`
- `.race()`

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2  
3 Promise.all([PBnJMaker.fetchBread(), PBnJMaker.fetchPeanutButter(), PBnJMaker.fetchJam()])  
4   .then(() => {  
5     return Promise.all([PBnJMaker.spreadPeanutButter(), PBnJMaker.spreadJam()]);  
6   }).then(() => {  
7     return PBnJMaker.closeSandwich();  
8   }).then((result) => {  
9     console.log("Eat that " + result);  
10  });  
11
```

Generators (2015)



What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 let iteration = iterable();  
9 console.log(iteration.next()); // 3  
10 //Do stuff  
11 console.log(iteration.next()); // 2  
12 console.log(iteration.next()); // 1  
13 console.log(iteration.next()); // 0  
14 //More stuff  
15 console.log(iteration.next()); // undefined
```

What is a generator?

- An iterable function
- “Pauses” the execution of a function

```
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 for (let iteration of iterable()) {  
9     console.log(iteration);  
10 }  
11  
12 // 3  
13 // 2  
14 // 1  
15 // 0
```

What is a generator?

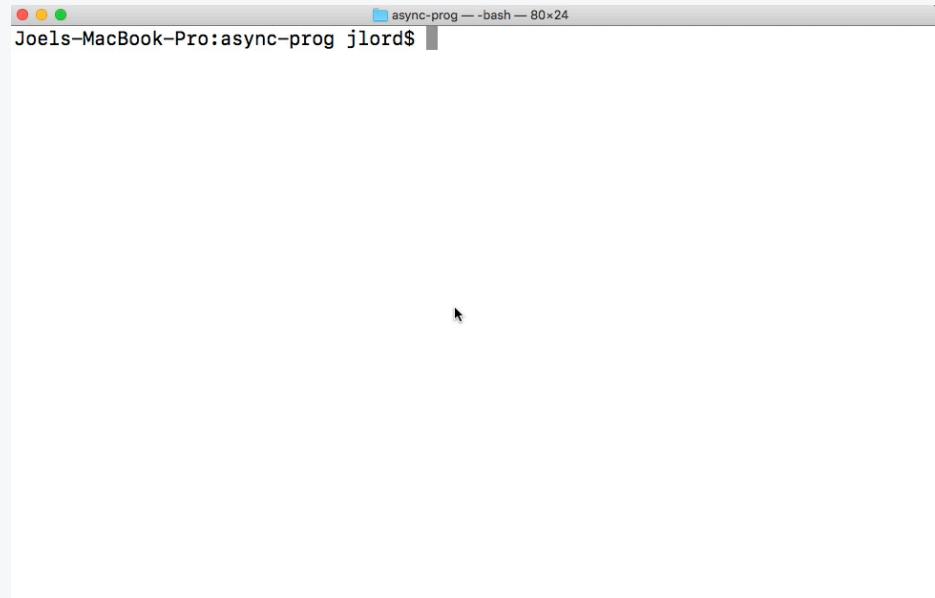
- An iterable function
- “Pauses” the execution of a function

```
  ● ○ ●  
1 function* iterable() {  
2     let i = 4;  
3     while(i--) {  
4         yield i;  
5     }  
6 }  
7  
8 for (let iteration of iterable()) {  
9     console.log(iteration);  
10 }  
11  
12 // 3  
13 // 2  
14 // 1  
15 // 0
```

Code

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.verbose();
4
5 function* makePBnJ() {
6     yield PBnJMaker.fetchBread;
7     yield PBnJMaker.fetchPeanutButter;
8     yield PBnJMaker.fetchJam;
9     yield PBnJMaker.spreadPeanutButter;
10    yield PBnJMaker.spreadJam;
11    yield PBnJMaker.closeSandwich;
12 }
13
14 for (let step of makePBnJ()) {
15     step();
16 }
17
18 console.log("Eat that sandwich");
```

Result



A screenshot of a terminal window titled "async-prog — bash — 80x24" running on a MacBook Pro. The window title bar shows "Joels-MacBook-Pro:async-prog jlord\$". The terminal is currently empty, with a cursor visible at the top right.

Async Generators

- Combining the power of promises and generators

```
1 module.exports = (makeGenerator) => {
2     let generator = makeGenerator.apply(this, arguments);
3
4     function handle(result) {
5         // A generator returns an object
6         // {done: [Boolean], value: [Any]}
7         if (result.done) return Promise.resolve(result.value);
8         return Promise.resolve(result.value).then(function(res) {
9             return handle(generator.next(res));
10        }, function(err) {
11            return handle(generator.throw(err));
12        });
13    }
14
15    try {
16        return handle(generator.next());
17    } catch(e) {
18        return Promise.reject(e);
19    }
20};
```

Code



```
1 let PBnJMaker = require("./pbnjmaker");
2 let async = require("./async");
3
4 function* makePBnJ() {
5   yield PBnJMaker.fetchBread();
6   yield PBnJMaker.fetchPeanutButter();
7   yield PBnJMaker.fetchJam();
8   yield PBnJMaker.spreadPeanutButter();
9   yield PBnJMaker.spreadJam();
10  return PBnJMaker.closeSandwich();
11 }
12
13 async(makePBnJ).then((result) => console.log("Eat that " + result));
```

Result

```
Joels-MacBook-Pro:async-prog jlord$
```

Code



```
1 let PBnJMaker = require("./pbnjmaker");
2 let async = require("./async");
3
4 function* makePBnJ() {
5     yield PBnJMaker.fetchBread();
6     yield PBnJMaker.fetchPeanutButter();
7     yield PBnJMaker.fetchJam();
8     yield PBnJMaker.spreadPeanutButter();
9     yield PBnJMaker.spreadJam();
10    return PBnJMaker.closeSandwich();
11 }
12
13 async(makePBnJ).then((result) => console.log("Eat that " + result));
```

Result

```
Joels-MacBook-Pro:async-prog jlord$
```



Async / Await (2017)

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4   await PBnJMaker.fetchBread();
5   await PBnJMaker.fetchPeanutButter();
6   await PBnJMaker.fetchJam();
7   await PBnJMaker.spreadPeanutButter();
8   await PBnJMaker.spreadJam();
9   return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Newly introduced in ES8 (ES2017)
- Natural evolution of generators and promises

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function makePBnJ() {
4     await PBnJMaker.fetchBread();
5     await PBnJMaker.fetchPeanutButter();
6     await PBnJMaker.fetchJam();
7     await PBnJMaker.spreadPeanutButter();
8     await PBnJMaker.spreadJam();
9     return PBnJMaker.closeSandwich();
10 }
11
12 makePBnJ().then((result) => console.log("Eat that " + result));
```

What is async/await?

- Very easy to parallelize processes

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 async function getIngredients() {
4     let p1 = PBnJMaker.fetchBread();
5     let p2 = PBnJMaker.fetchPeanutButter();
6     let p3 = PBnJMaker.fetchJam();
7     return await p1 + await p2 + await p3;
8 }
9
10 async function prepareSandwich() {
11     let p1 = PBnJMaker.spreadPeanutButter();
12     let p2 = PBnJMaker.spreadJam();
13     return await p1 + await p2;
14 }
15
16 async function makePBnJ() {
17     await getIngredients();
18     await prepareSandwich();
19     return PBnJMaker.closeSandwich();
20 }
21 makePBnJ().then((result) => console.log("Eat that " + result));
22
```

What is async/await?

- Very easy to parallelize processes

```
1 let PBnJMaker = require( "./pbnjmaker" );
2
3 async function getIngredients() {
4     let p1 = PBnJMaker.fetchBread();
5     let p2 = PBnJMaker.fetchPeanutButter();
6     let p3 = PBnJMaker.fetchJam();
7     return await p1 + await p2 + await p3;
8 }
9
10 async function prepareSandwich() {
11     let p1 = PBnJMaker.spreadPeanutButter();
12     let p2 = PBnJMaker.spreadJam();
13     return await p1 + await p2;
14 }
15
16 async function makePBnJ() {
17     await getIngredients();
18     await prepareSandwich();
19     return PBnJMaker.closeSandwich();
20 }
21 makePBnJ().then((result) => console.log("Eat that " + result));
22
```

What is async/await?

- Very easy to parallelize processes

```
● ● ●  
1 let PBnJMaker = require( "./pbnjmaker" );
2
3 async function getIngredients() {
4     let p1 = PBnJMaker.fetchBread();
5     let p2 = PBnJMaker.fetchPeanutButter();
6     let p3 = PBnJMaker.fetchJam();
7     return await p1 + await p2 + await p3;
8 }
9
10 async function prepareSandwich() {
11     let p1 = PBnJMaker.spreadPeanutButter();
12     let p2 = PBnJMaker.spreadJam();
13     return await p1 + await p2;
14 }
15
16 async function makePBnJ() {
17     await getIngredients();
18     await prepareSandwich();
19     return PBnJMaker.closeSandwich();
20 }
21 makePBnJ().then((result) => console.log("Eat that " + result));
22
```

What is async/await?

- Very easy to parallelize processes

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2  
3 async function getIngredients() {  
4     let p1 = PBnJMaker.fetchBread();  
5     let p2 = PBnJMaker.fetchPeanutButter();  
6     let p3 = PBnJMaker.fetchJam();  
7     return await p1 + await p2 + await p3;  
8 }  
9  
10 async function prepareSandwich() {  
11     let p1 = PBnJMaker.spreadPeanutButter();  
12     let p2 = PBnJMaker.spreadJam();  
13     return await p1 + await p2;  
14 }  
15  
16 async function makePBnJ() {  
17     await getIngredients();  
18     await prepareSandwich();  
19     return PBnJMaker.closeSandwich();  
20 }  
21 makePBnJ().then((result) => console.log("Eat that " + result));  
22
```

What is async/await?

- Very easy to parallelize processes



```
1 let PBnJMaker = require( "./pbnjmaker" );
2
3 async function getIngredients() {
4     let p1 = PBnJMaker.fetchBread();
5     let p2 = PBnJMaker.fetchPeanutButter();
6     let p3 = PBnJMaker.fetchJam();
7     return await p1 + await p2 + await p3;
8 }
9
10 async function prepareSandwich() {
11     let p1 = PBnJMaker.spreadPeanutButter();
12     let p2 = PBnJMaker.spreadJam();
13     return await p1 + await p2;
14 }
15
16 async function makePBnJ() {
17     await getIngredients();
18     await prepareSandwich();
19     return PBnJMaker.closeSandwich();
20 }
21 makePBnJ().then((result) => console.log("Eat that " + result));
22
```

What is async/await?

- Very easy to parallelize processes

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2  
3 async function getIngredients() {  
4     let p1 = PBnJMaker.fetchBread();  
5     let p2 = PBnJMaker.fetchPeanutButter();  
6     let p3 = PBnJMaker.fetchJam();  
7     return await p1 + await p2 + await p3;  
8 }  
9  
10 async function prepareSandwich() {  
11     let p1 = PBnJMaker.spreadPeanutButter();  
12     let p2 = PBnJMaker.spreadJam();  
13     return await p1 + await p2;  
14 }  
15  
16 async function makePBnJ() {  
17     await getIngredients();  
18     await prepareSandwich();  
19     return PBnJMaker.closeSandwich();  
20 }  
21 makePBnJ().then((result) => console.log("Eat that " + result));  
22
```

What is async/await?

- Very easy to parallelize processes

```
1 let PBnJMaker = require( "./pbnjmaker" );
2
3 async function getIngredients() {
4     let p1 = PBnJMaker.fetchBread();
5     let p2 = PBnJMaker.fetchPeanutButter();
6     let p3 = PBnJMaker.fetchJam();
7     return await p1 + await p2 + await p3;
8 }
9
10 async function prepareSandwich() {
11     let p1 = PBnJMaker.spreadPeanutButter();
12     let p2 = PBnJMaker.spreadJam();
13     return await p1 + await p2;
14 }
15
16 async function makePBnJ() {
17     await getIngredients();
18     await prepareSandwich();
19     return PBnJMaker.closeSandwich();
20 }
21 makePBnJ().then((result) => console.log("Eat that " + result));
22
```

Async / Await

Programmatically define your steps

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 let ingredients = ["Bread", "PeanutButter", "Jam"];
4 let steps = ingredients.map((i) => `fetch${i}`);
5 steps.push(...["spreadPeanutButter", "spreadJam"]);
6
7 async function makePBnJ(todo) {
8     for (var i = 0; i < todo.length; i++) {
9         await PBnJMaker[todo[i]]();
10    }
11    return PBnJMaker.closeSandwich();
12 }
13
14 makePBnJ(steps).then((result) => console.log("Eat that " + result));
```



Events (1990's)

Events in the browser

- At the core of the original Javascript
- window.onload = ...
- button.onclick = ...

```
1 <html>
2 <body>
3   <button id="btn1">Click me</button>
4 </body>
5
6 <script>
7   let btn = document.querySelector("#btn1");
8   btn.onclick = () => alert("Clicked");
9 </script>
10 </html>
```

Node Event Emitter

- In node, you can use the event emitter
- Create your own events for a library

```
1 const EventEmitter = require('events');
2 class MyEmitter extends EventEmitter {}
3 const eventEmitter = new MyEmitter();
4
5 let fetchBread = function(options) {
6   return axios.get(SERVER + "/fetchBread").then((response) => {
7     eventEmitter.emit("breadFetched", response);
8   });
9 };
10
11 exports.Events = eventEmitter;
```

Node Event Emitter

- In node, you can use the event emitter
- Create your own events for a library

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.Events.on("breadFetched", () => {
4     PBnJMaker.fetchPeanutButter();
5 });
6 PBnJMaker.Events.on("peanutButterFetched", () => PBnJMaker.fetchJam());
7 PBnJMaker.Events.on("jamFetched", () => PBnJMaker.spreadPeanutButter());
8 PBnJMaker.Events.on("peanutButterSpreaded", () => PBnJMaker.spreadJam());
9 PBnJMaker.Events.on("jamSpreaded", () => PBnJMaker.closeSandwich());
10 PBnJMaker.Events.on("sandwichClosed", (result) => console.log("Eat that " + result));
11
12 PBnJMaker.fetchBread();
13
```

Code Demo

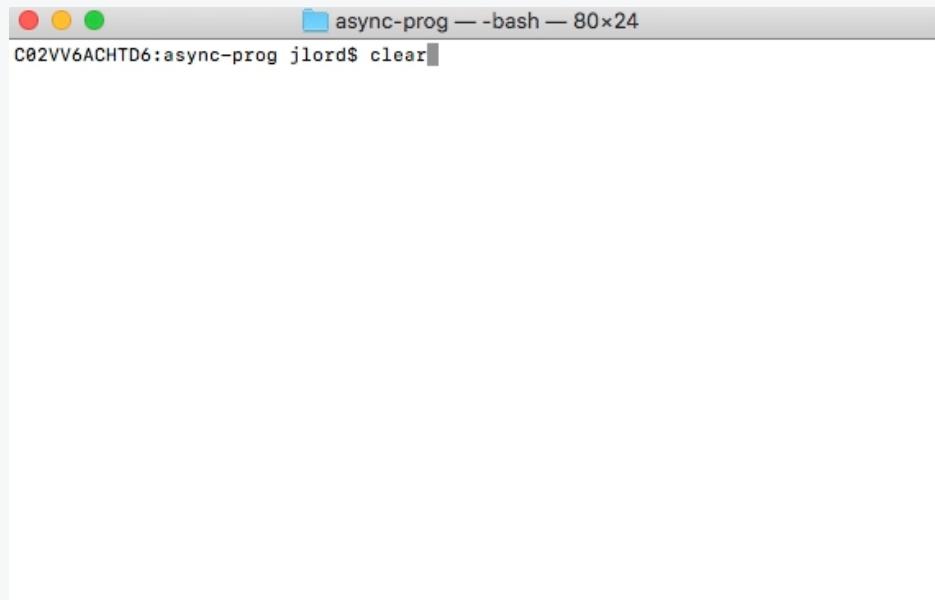
```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.Events.on("breadFetched", () => {
4   PBnJMaker.fetchPeanutButter();
5 });
6 PBnJMaker.Events.on("peanutButterFetched", () => PBnJMaker.fetchJam());
7 PBnJMaker.Events.on("jamFetched", () => PBnJMaker.spreadPeanutButter());
8 PBnJMaker.Events.on("peanutButterSpreading", () => PBnJMaker.spreadJam());
9 PBnJMaker.Events.on("jamSpreading", () => PBnJMaker.closeSandwich());
10 PBnJMaker.Events.on("sandwichClosed", (result) => console.log("Eat that " + result));
11
12 PBnJMaker.fetchBread();
13
```



Code

```
● ● ●  
1 let PBnJMaker = require("./pbnjmaker");  
2  
3 let ingredientsFetched = 0;  
4 let ingredientsSpreaded = 0;  
5  
6 let ingredientFetched = () => {  
7   ingredientsFetched++;  
8   if (ingredientsFetched === 3) {  
9     PBnJMaker.Events.emit("ingredientsFetched");  
10 }  
11 };  
12  
13 let spreaded = () => {  
14   ingredientsSpreaded++;  
15   if (ingredientsSpreaded === 2) {  
16     PBnJMaker.Events.emit("ingredientsSpreaded");  
17   }  
18 };  
19  
20 PBnJMaker.Events.on("breadFetched", ingredientFetched);  
21 PBnJMaker.Events.on("peanutButterFetched", ingredientFetched);  
22 PBnJMaker.Events.on("jamFetched", ingredientFetched);  
23  
24 PBnJMaker.Events.on("ingredientsFetched", () => {  
25   PBnJMaker.spreadPeanutButter();  
26   PBnJMaker.spreadJam();  
27 });  
28  
29 PBnJMaker.Events.on("peanutButterSpreaded", spreaded);  
30 PBnJMaker.Events.on("jamSpreaded", spreaded);  
31  
32 PBnJMaker.Events.on("ingredientsSpreaded", () => {  
33   PBnJMaker.closeSandwich();  
34 });  
35  
36 PBnJMaker.Events.on("sandwichClosed", (result) => console.log("Eat that " + result));  
37  
38 PBnJMaker.fetchBread();  
39 PBnJMaker.fetchPeanutButter();  
40 PBnJMaker.fetchJam();  
41
```

Demo



A screenshot of a terminal window titled "async-prog — bash — 80x24". The window has red, yellow, and green window control buttons at the top left. The title bar shows the path "C02VV6ACHTD6:async-prog jlord\$". The main area of the terminal is empty, showing only the command "clear" entered at the prompt.

A close-up photograph of a peanut butter and jelly sandwich. The sandwich is made with white bread and filled with a generous amount of peanut butter and jelly. It is cut in half diagonally and placed on a white, round plate. The background is a dark, wooden surface.

Reactive Stream (2017)

RxJS

- Manipulate data, synchronous or asynchronous as streams
- Manipulate streams like arrays

```
1 getDataFromLocalMemory( )
2   .filter (s => s != null)
3   .map(s => `${s} transformed`)
4   .forEach(s => console.log(`next => ${s}`))
```

```
1 getDataFromNetwork( )
2   .filter (s => s != null)
3   .map(s => `${s} transformed`)
4   .subscribe(s => console.log(`next => ${s}`))
```

Code Demo

```
● ● ●  
1 const Rx = require("rxjs");  
2 let PBnJMaker = require("../pbnjmaker");  
3  
4 const ingredients = ["bread", "peanutButter", "jam"];  
5  
6 const ingredientEvents = ingredients.map((ingredient) => {  
7   return Rx.Observable.fromEvent(PBnJMaker.Events, `${ingredient}Fetched`);  
8 });  
9  
10 const ingredientEvents$ = Rx.Observable.merge(...ingredientEvents);  
11  
12 const subscription = ingredientEvents$.subscribe(() => {  
13   console.log("Just fetched an ingredient");  
14 });
```



The terminal window shows the command `clear` being typed at the prompt. The window title is `rx -- -bash -- 80x24`. The prompt is `C02VV6ACHTD6:rx jlord$`.

RxJS

- RTFM

Build Reactive Web Sites with RxJS

Master Observables and Wrangle Events



A close-up photograph of a person's hands holding a sandwich in half. The sandwich is made with white bread and filled with a generous amount of red jam. The hands are positioned to show the filling, and the background is blurred, focusing on the sandwich.

In Conclusion
(c'est presque la faim)

Summary

Asynchronous patterns

- The Event Loop
- Callbacks
- Promises
- Generators
- Async/Await
- Events
- Reactive Stream

```
getData(a => {
    getMoreData(a, b => {
        getMoreData(b, c => {
            getMoreData(c, d => {
                getMoreData(d, e => {
                    console.log(e);
                });
            });
        });
    });
});
```



@joel__lord
#confoo

Summary

Asynchronous patterns

- The Event Loop
- Callbacks
- Promises
- Generators
- Async/Await
- Events
- Reactive Stream
- How to make a PBnJ sandwich!

```
getData(a => {
    getMoreData(a, b => {
        getMoreData(b, c => {
            getMoreData(c, d => {
                getMoreData(d, e => {
                    console.log(e);
                });
            });
        });
    });
});
```



@joel__lord
#confoo



Asynchronicity: concurrency. A tale of

Confoo - Montreal, Canada

March 7, 2018



@joel__lord



joellord