



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>
```



@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>
```



@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>
```



@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>
```



@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 let req = new XMLHttpRequest();
10 req.onreadystatechange = () => {
11   if (req.readyState === XMLHttpRequest.DONE && req.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>
```



@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 netFileContent("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



Race Conditions

About Me



@joel__lord



joellord

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!



@joel__lord
#confoo



The Event Loop

Classic Architecture (PHP, Java)



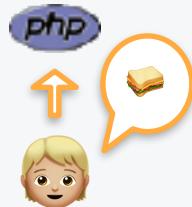
Classic Architecture (PHP, Java)

- 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



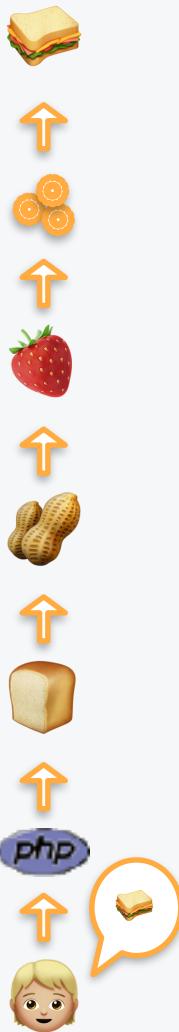
Classic Architecture (PHP, Java)

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



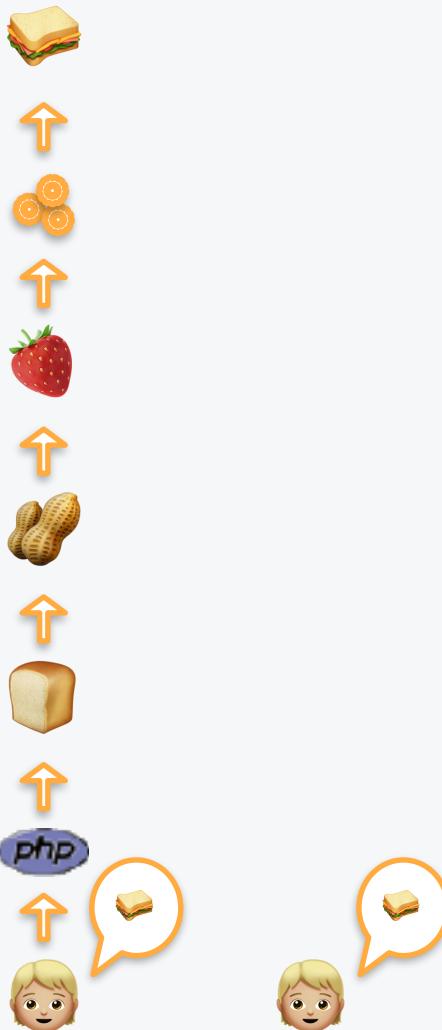
Classic Architecture (PHP, Java)

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



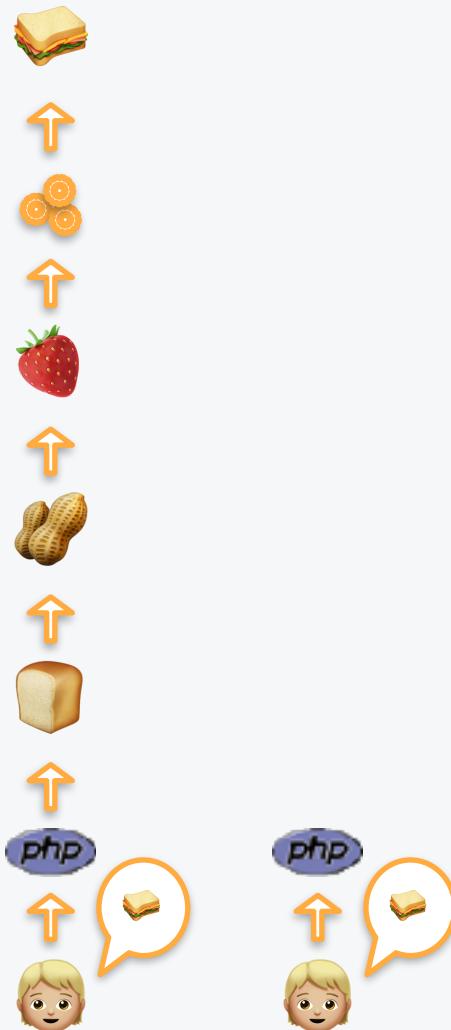
Classic Architecture (PHP, Java)

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



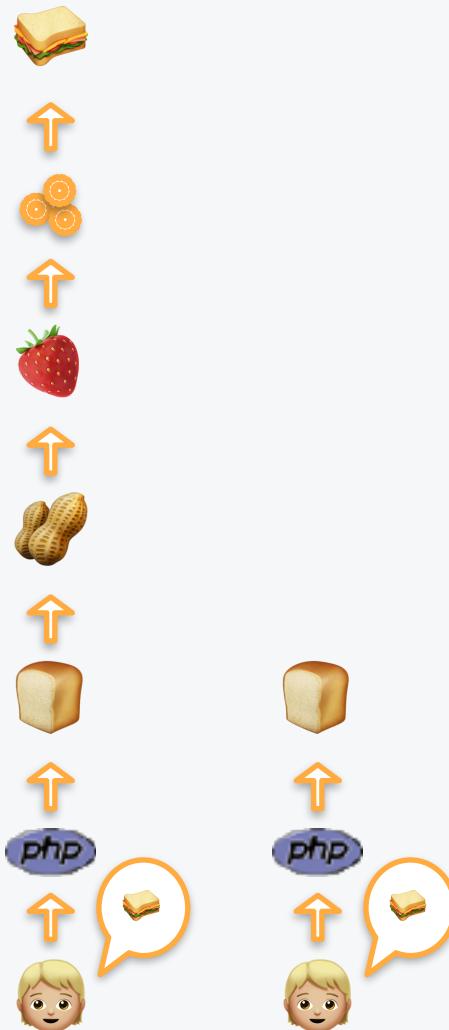
Classic Architecture (PHP, Java)

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



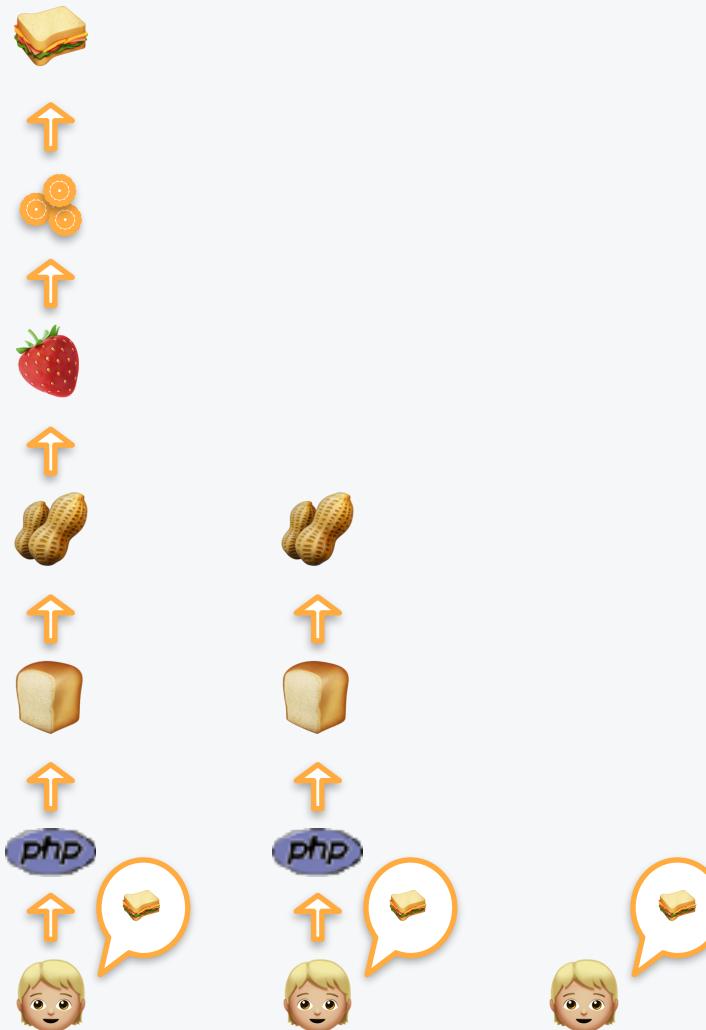
Classic Architecture (PHP, Java)

- 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



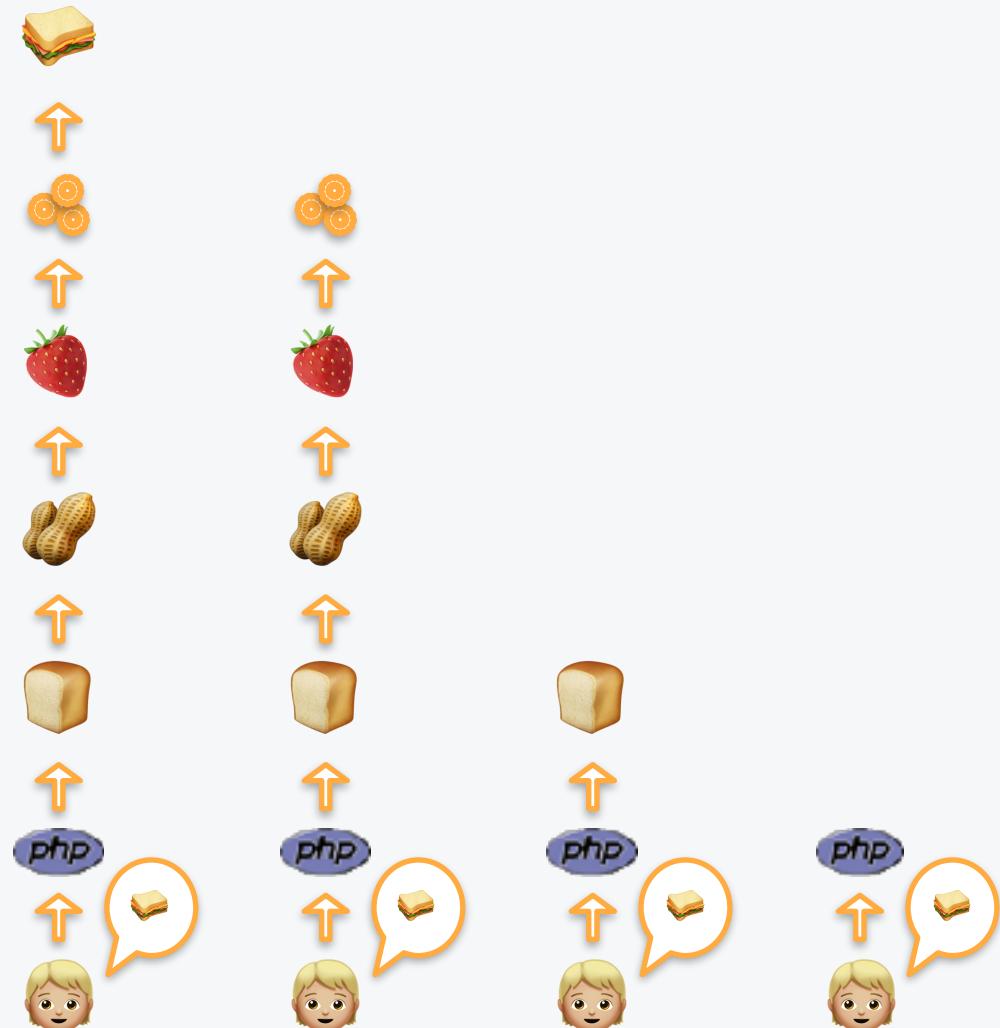
Classic Architecture (PHP, Java)

- 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



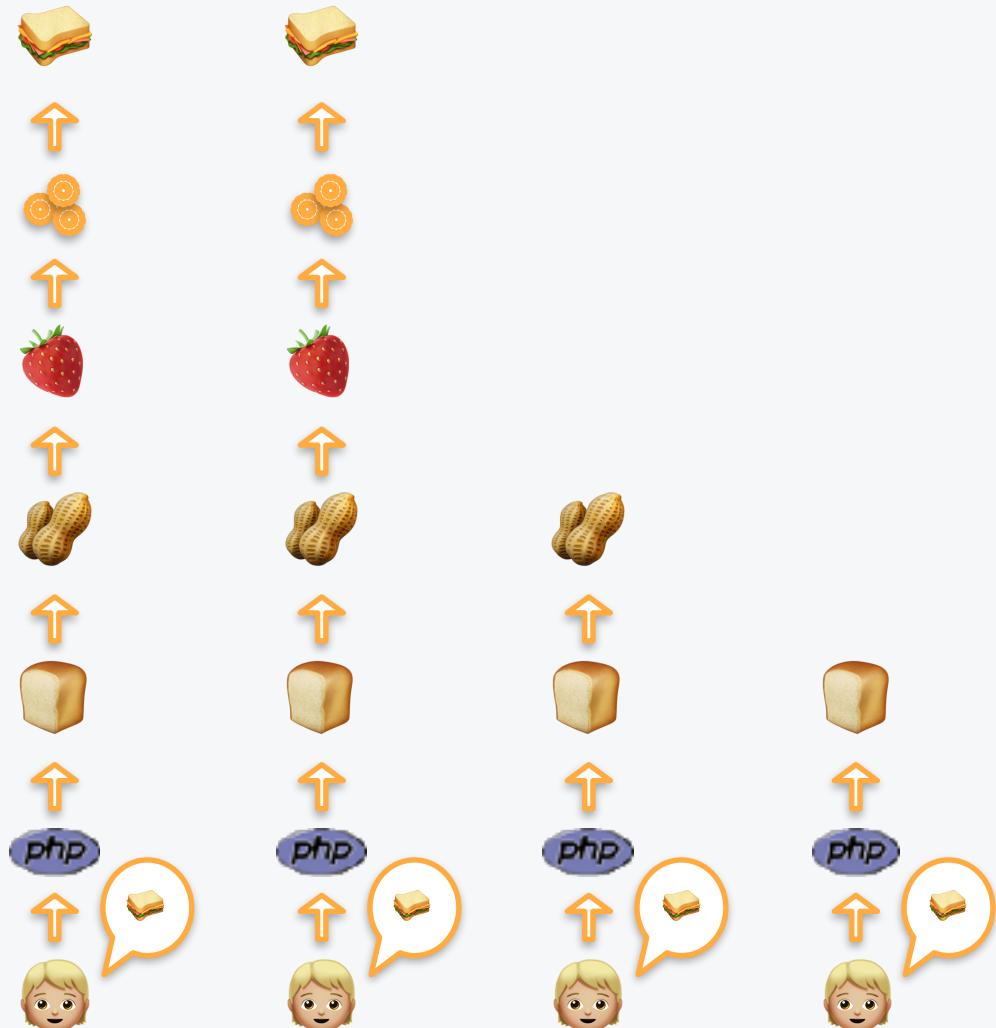
Classic Architecture (PHP, Java)

- 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



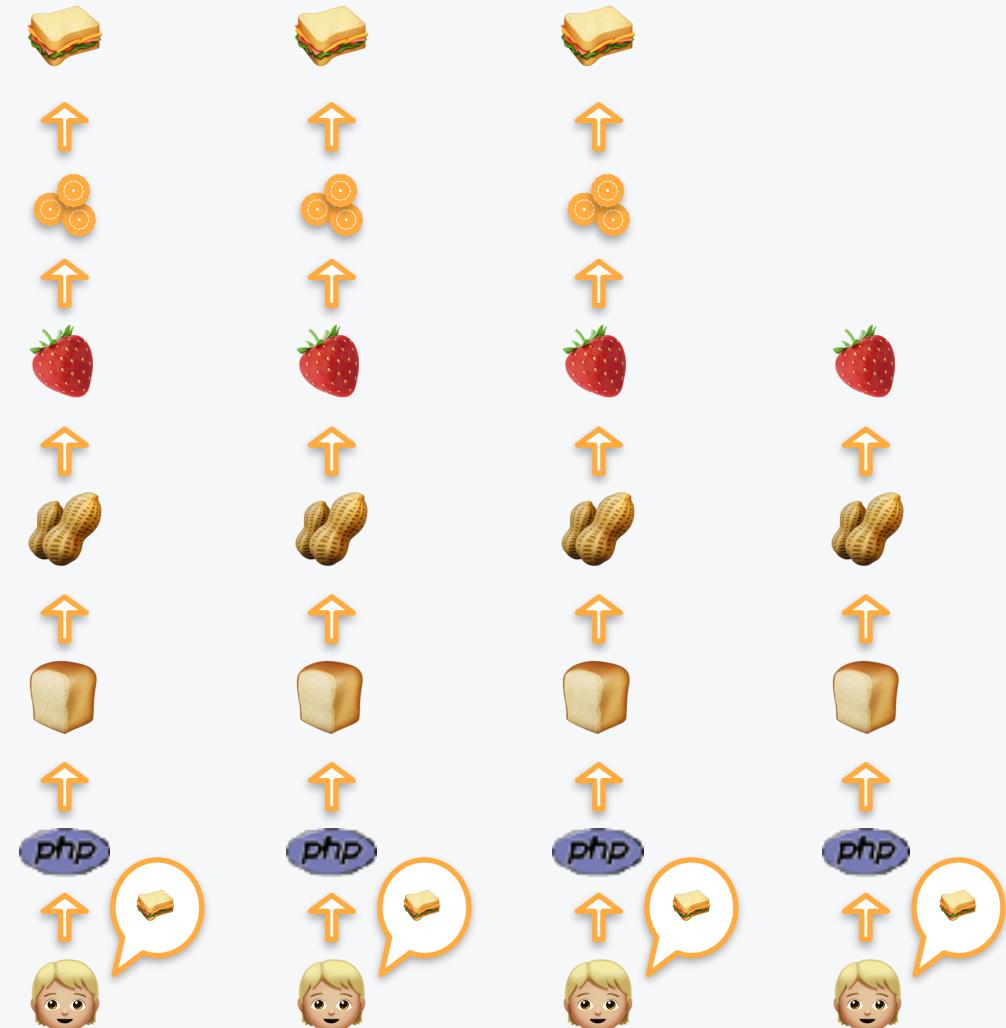
Classic Architecture (PHP, Java)

- 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



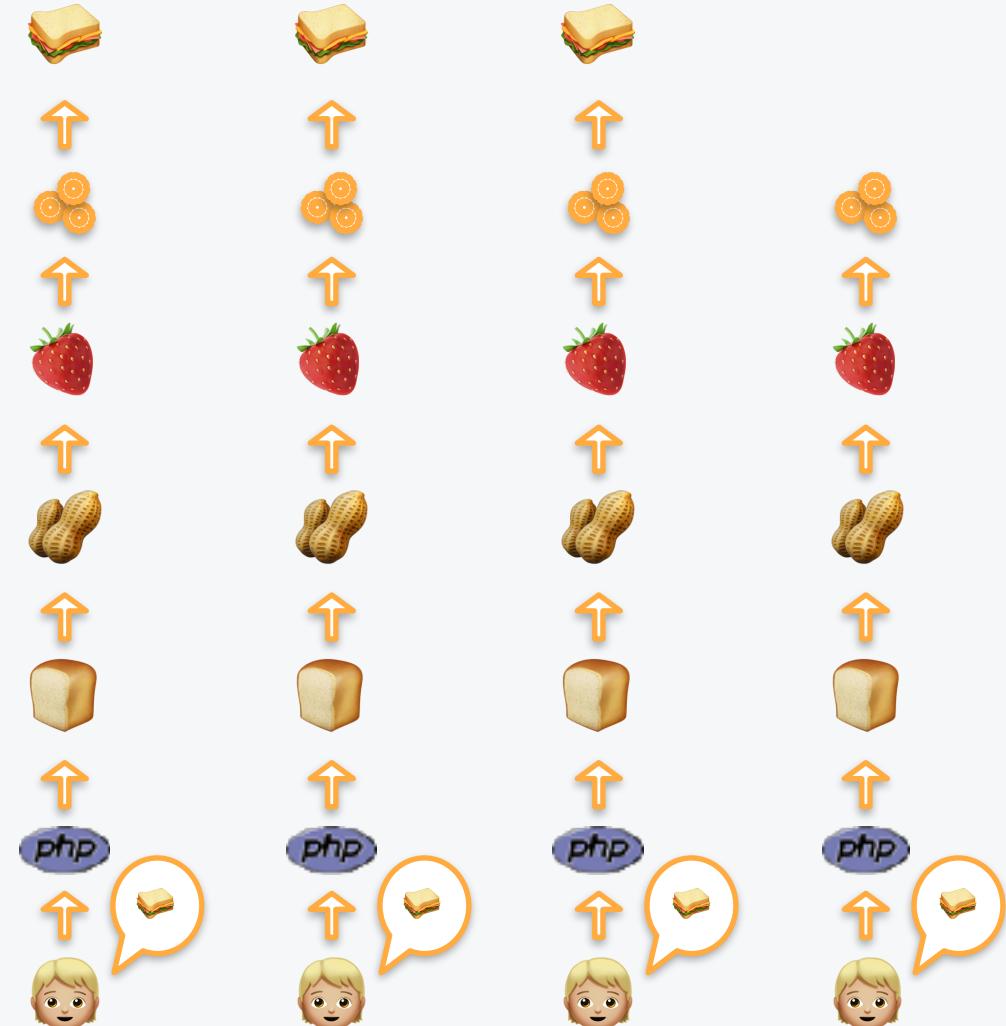
Classic Architecture (PHP, Java)

- 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



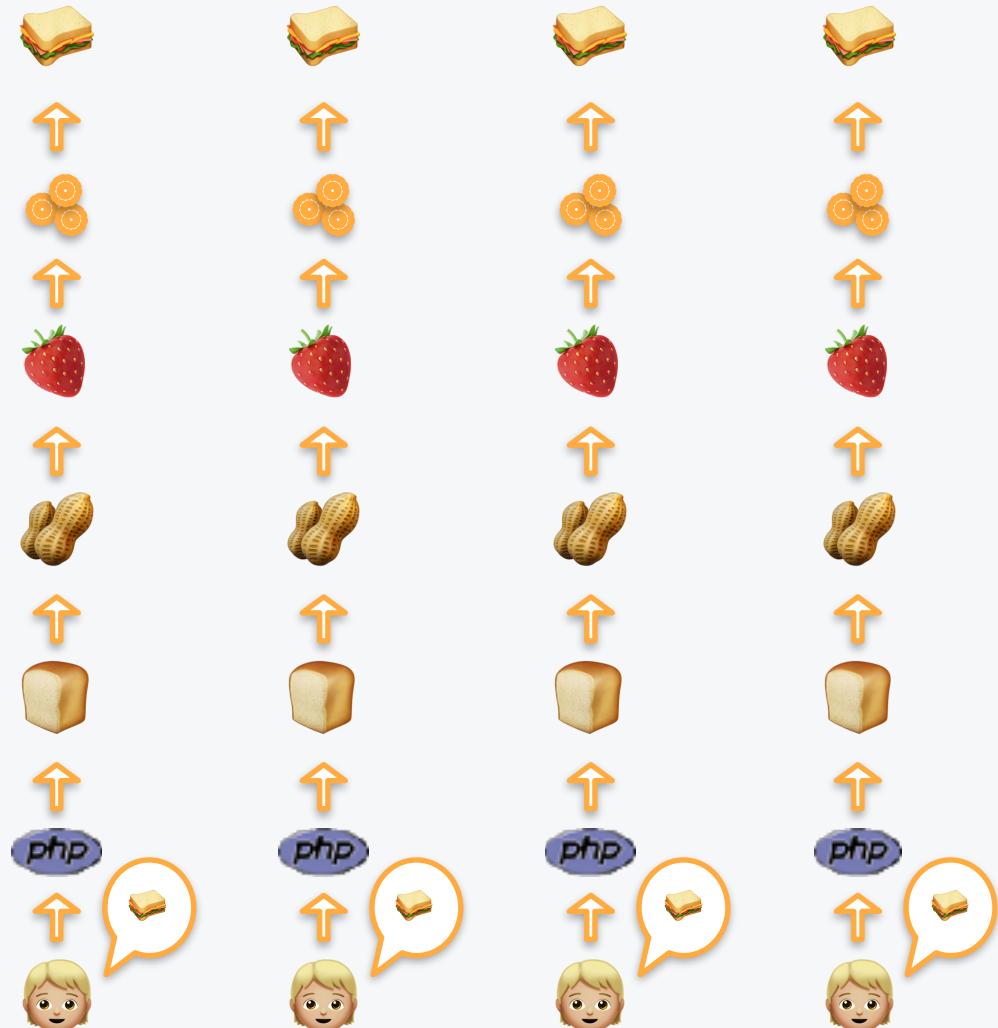
Classic Architecture (PHP, Java)

- 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



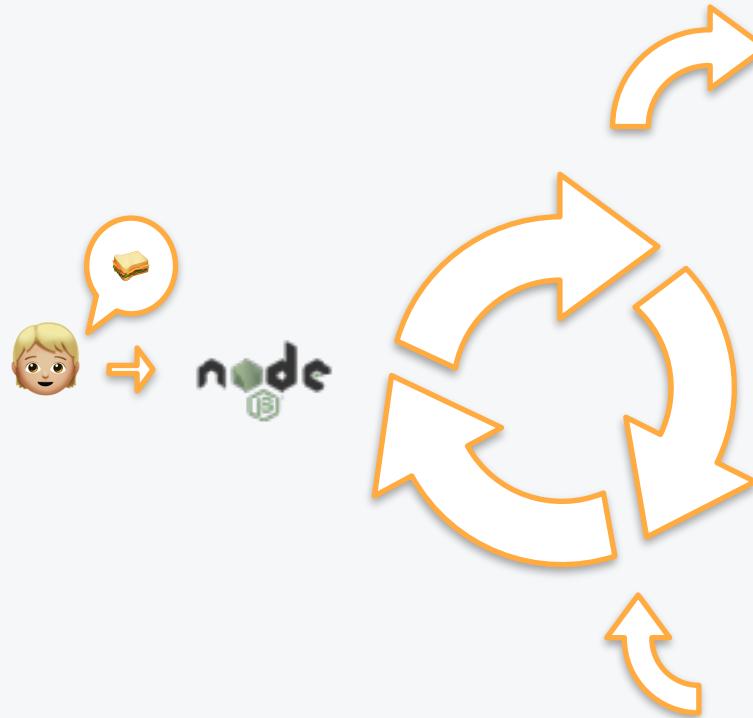
Classic Architecture (PHP, Java)

- 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



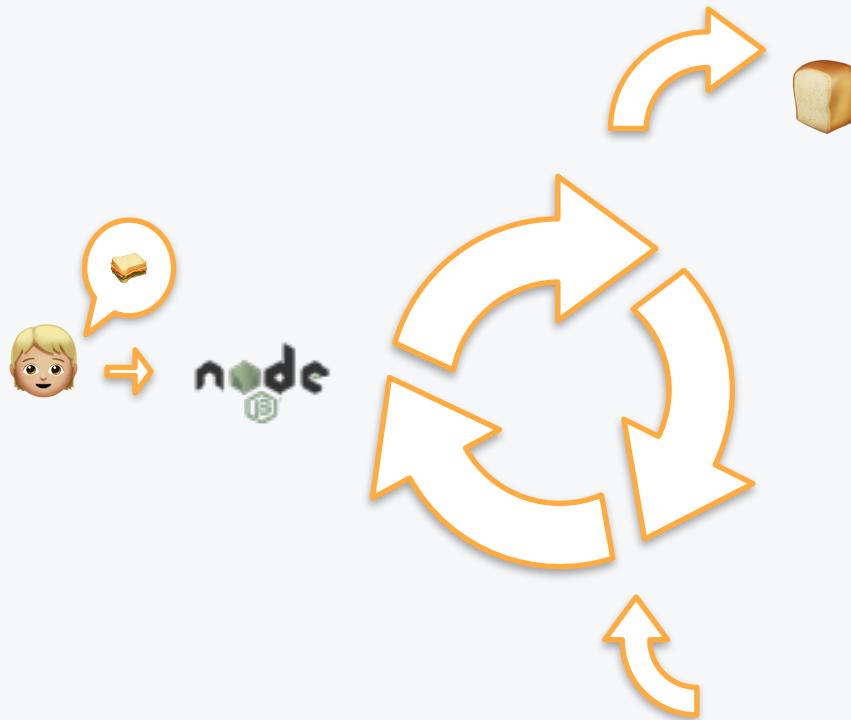
Event Loop (NodeJS)

- 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



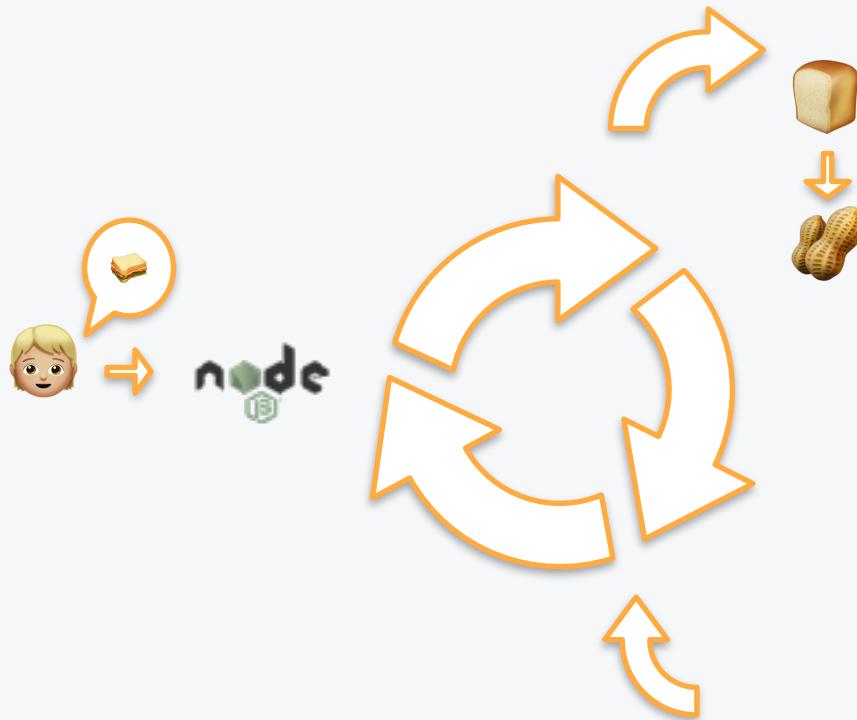
Event Loop (NodeJS)

- 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



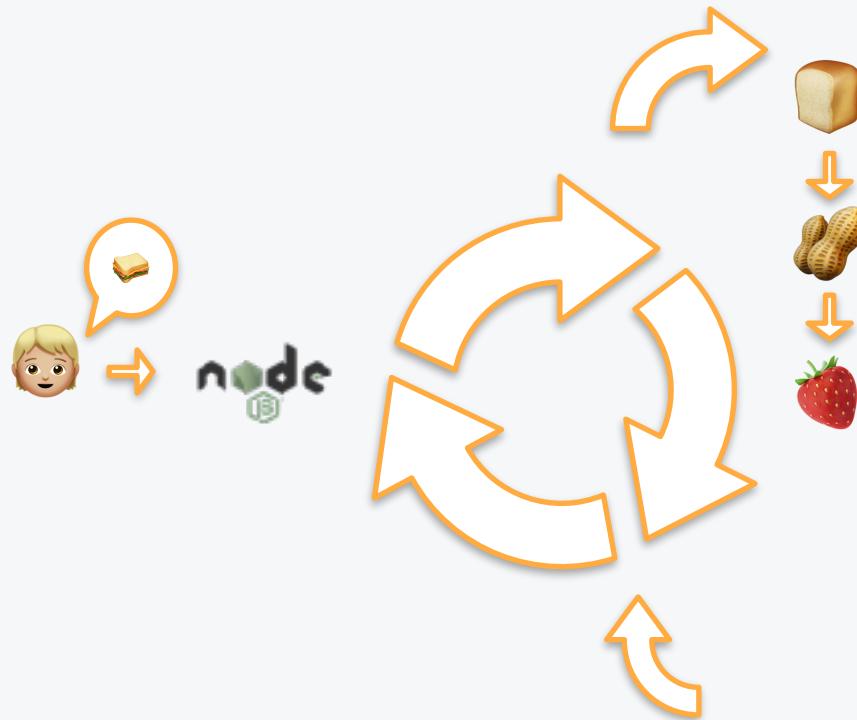
Event Loop (NodeJS)

- 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



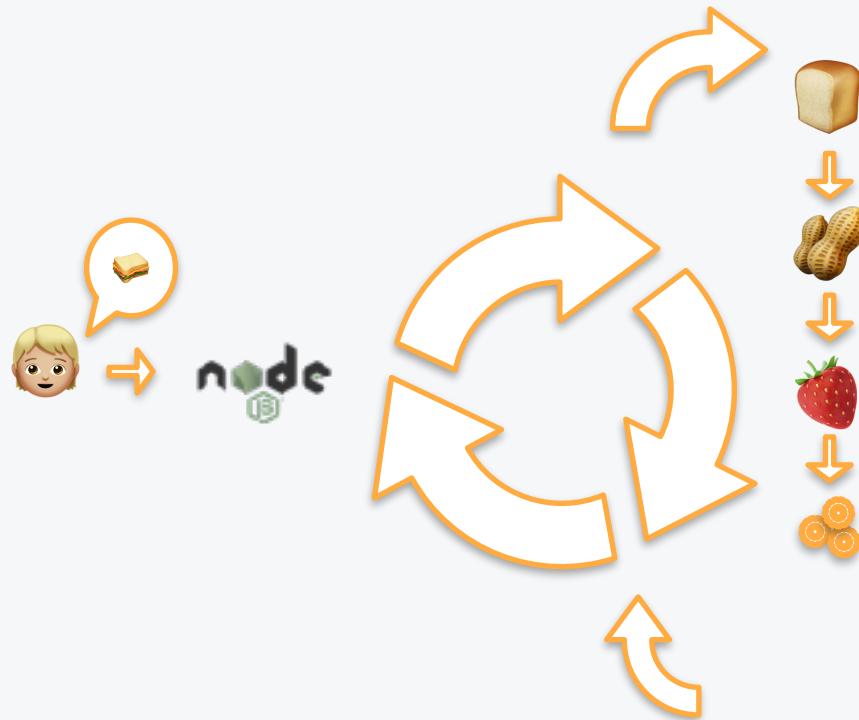
Event Loop (NodeJS)

- 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



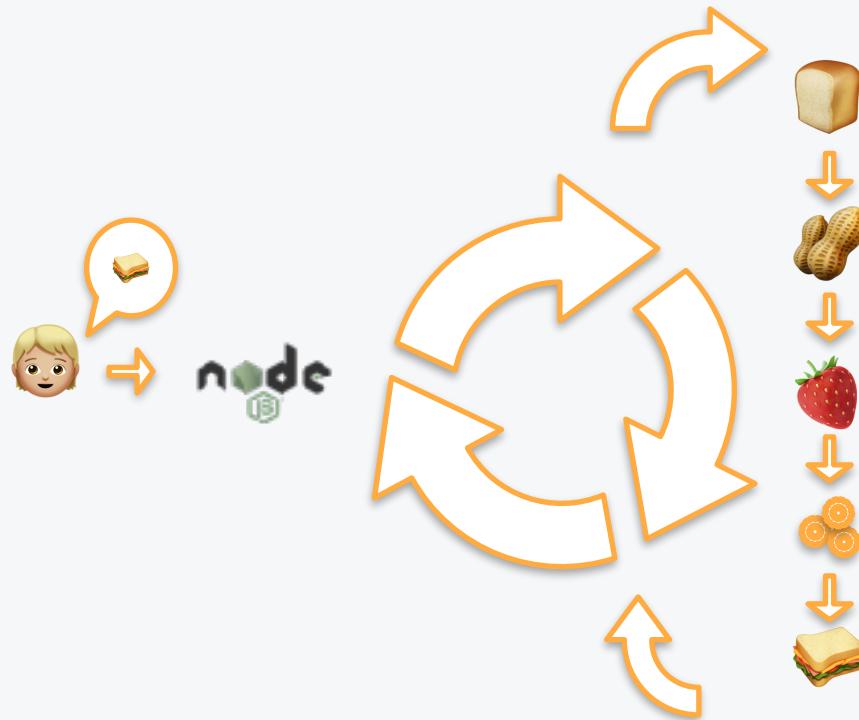
Event Loop (NodeJS)

- 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



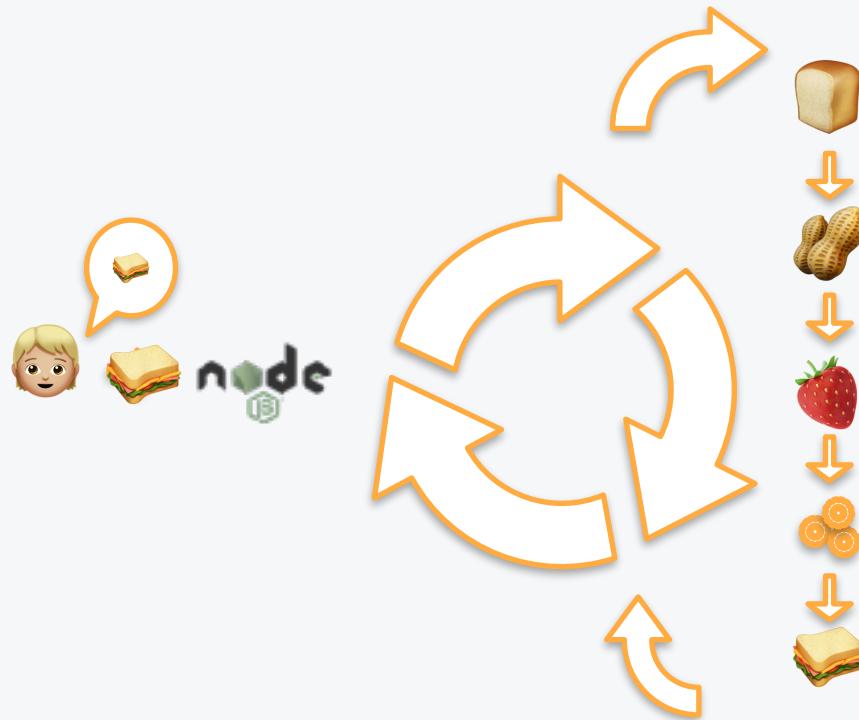
Event Loop (NodeJS)

- 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



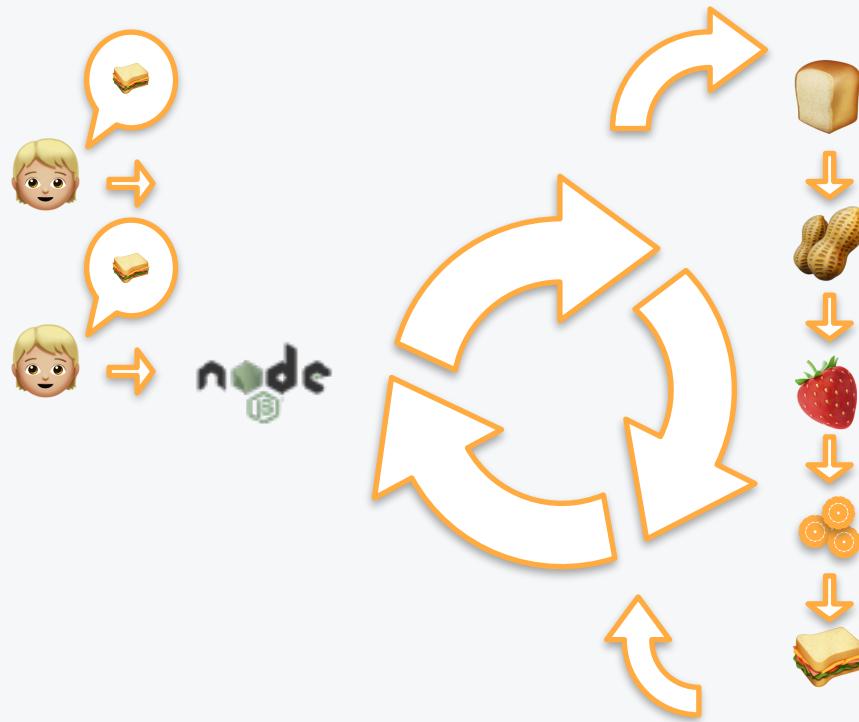
Event Loop (NodeJS)

- 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



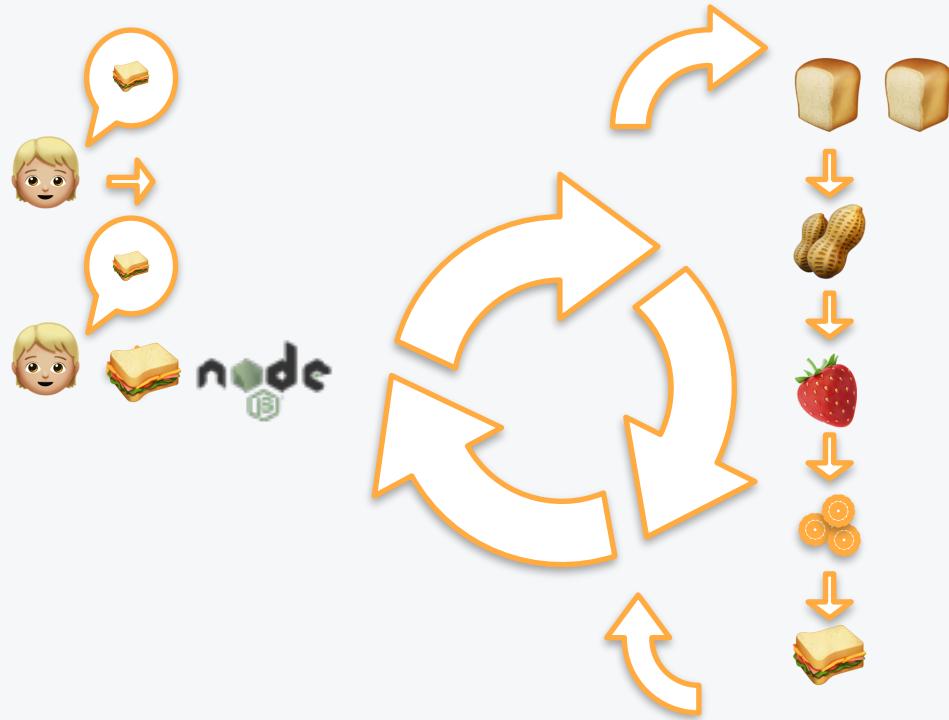
Event Loop (NodeJS)

- 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



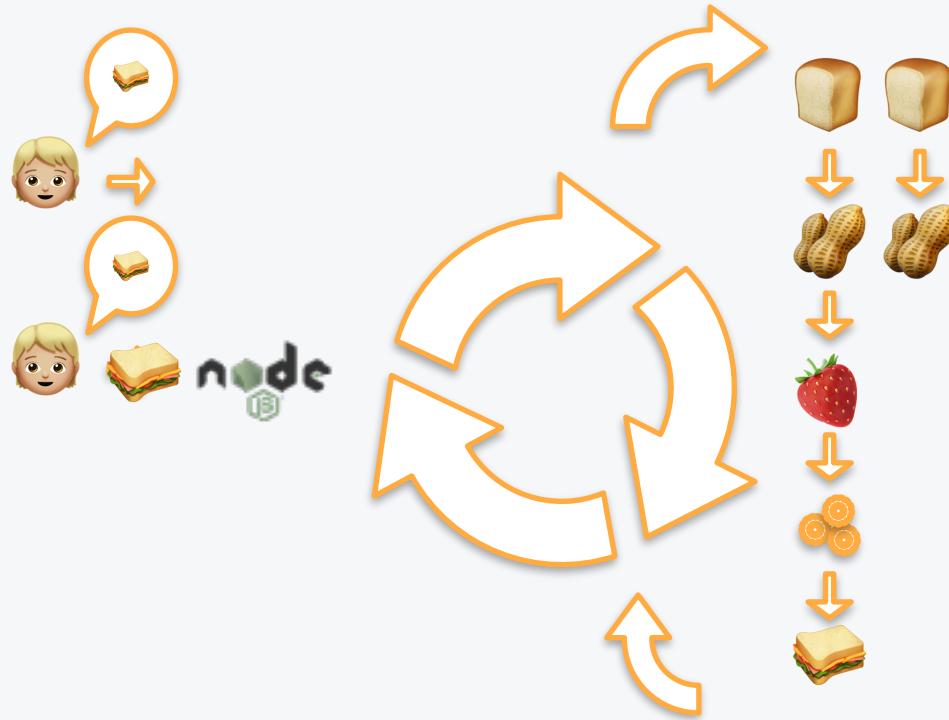
Event Loop (NodeJS)

- 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



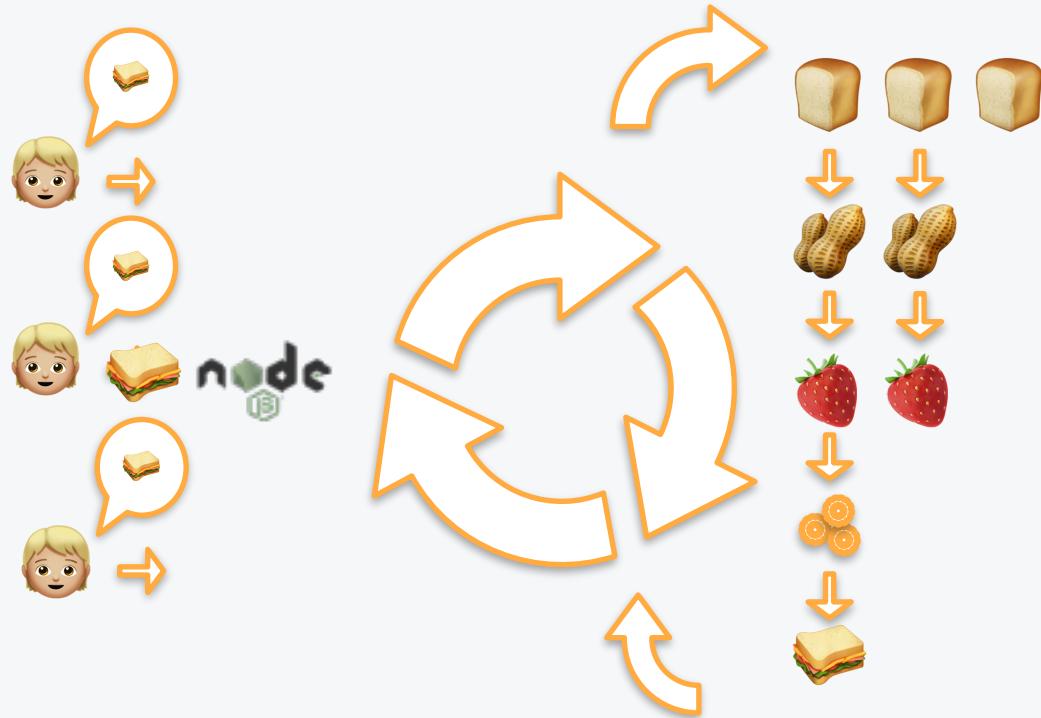
Event Loop (NodeJS)

- 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



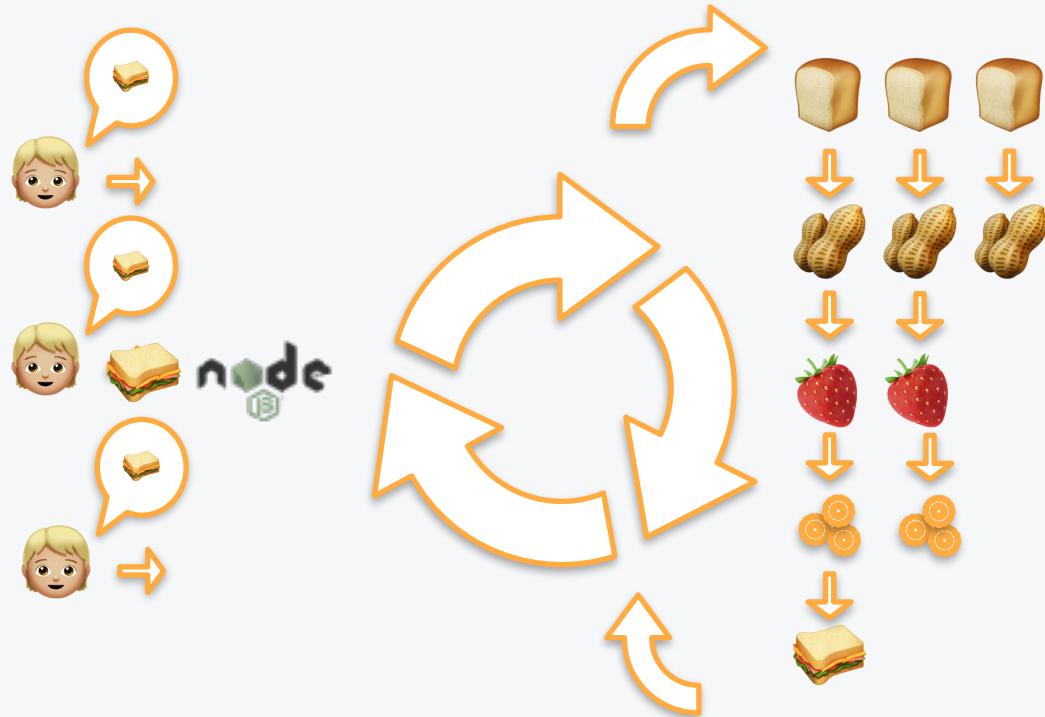
Event Loop (NodeJS)

- 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



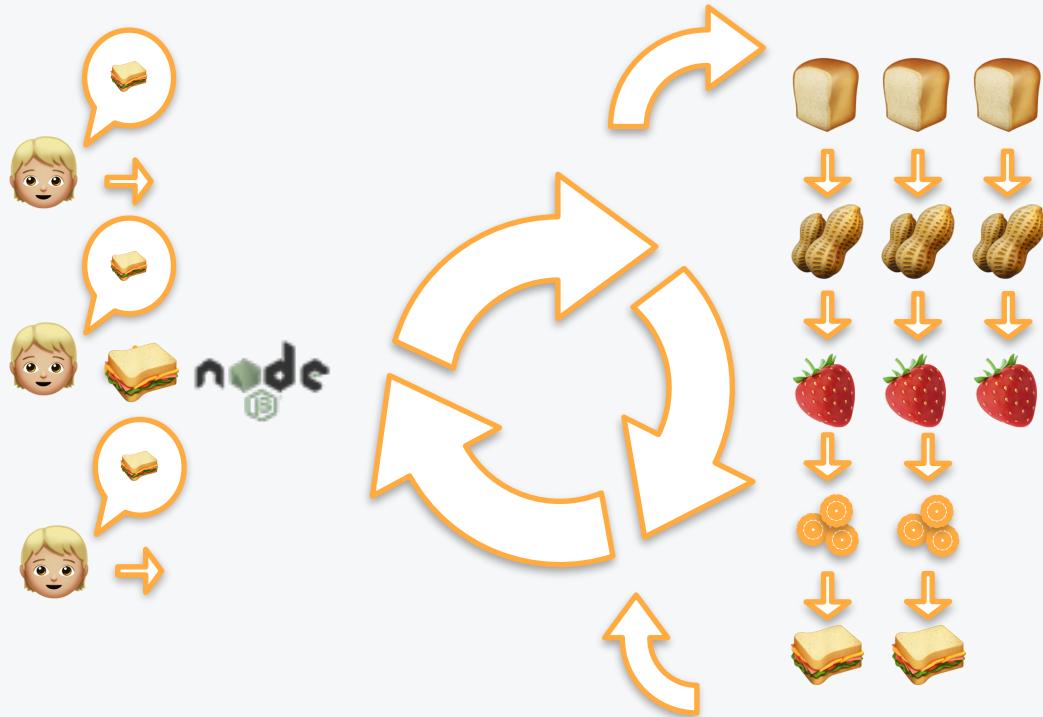
Event Loop (NodeJS)

- 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



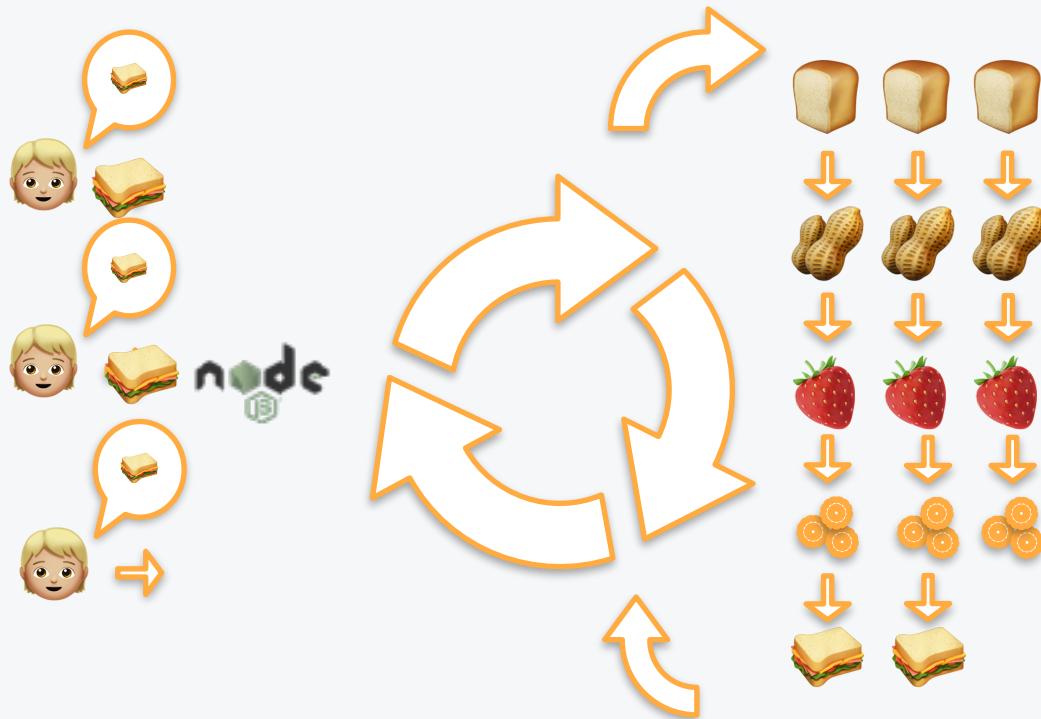
Event Loop (NodeJS)

- 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



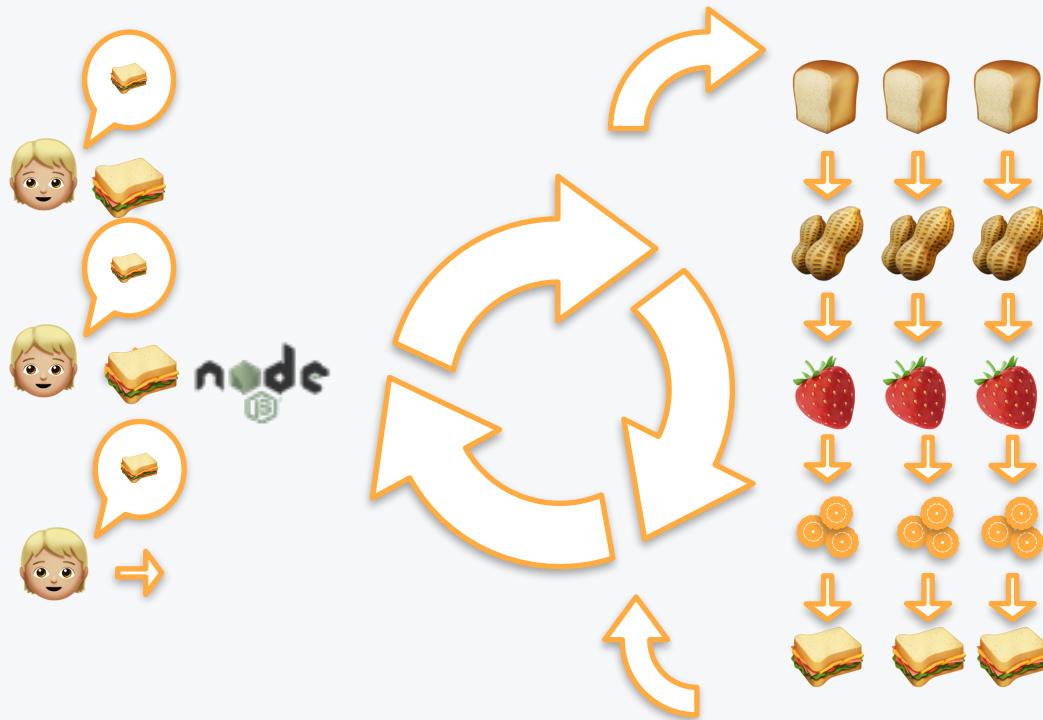
Event Loop (NodeJS)

- 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



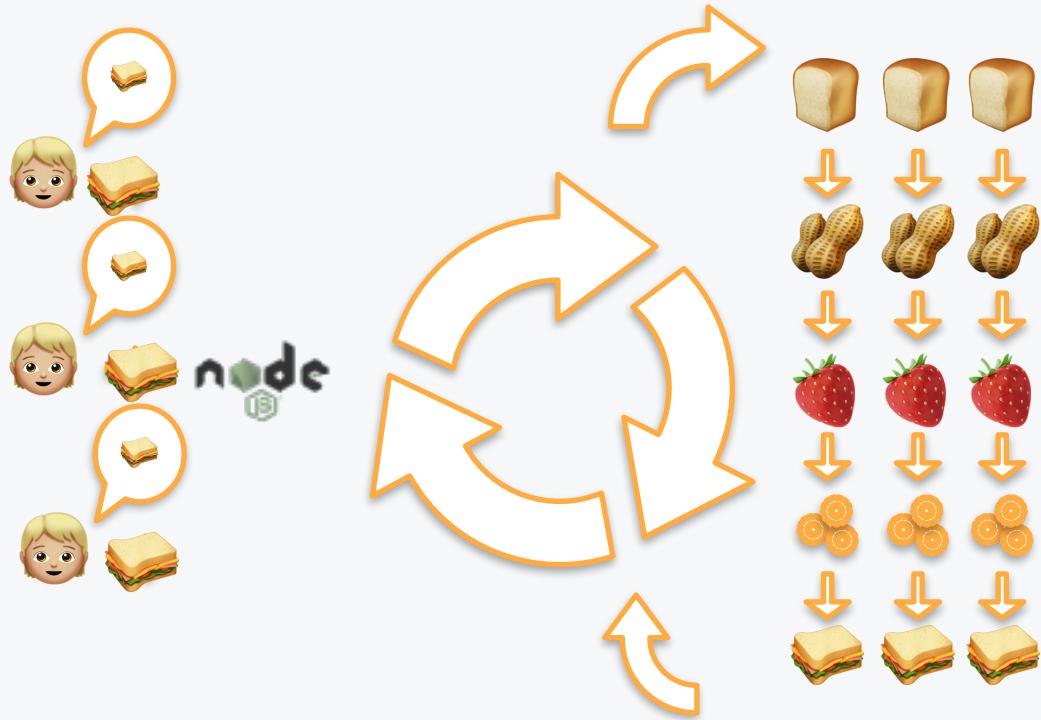
Event Loop (NodeJS)

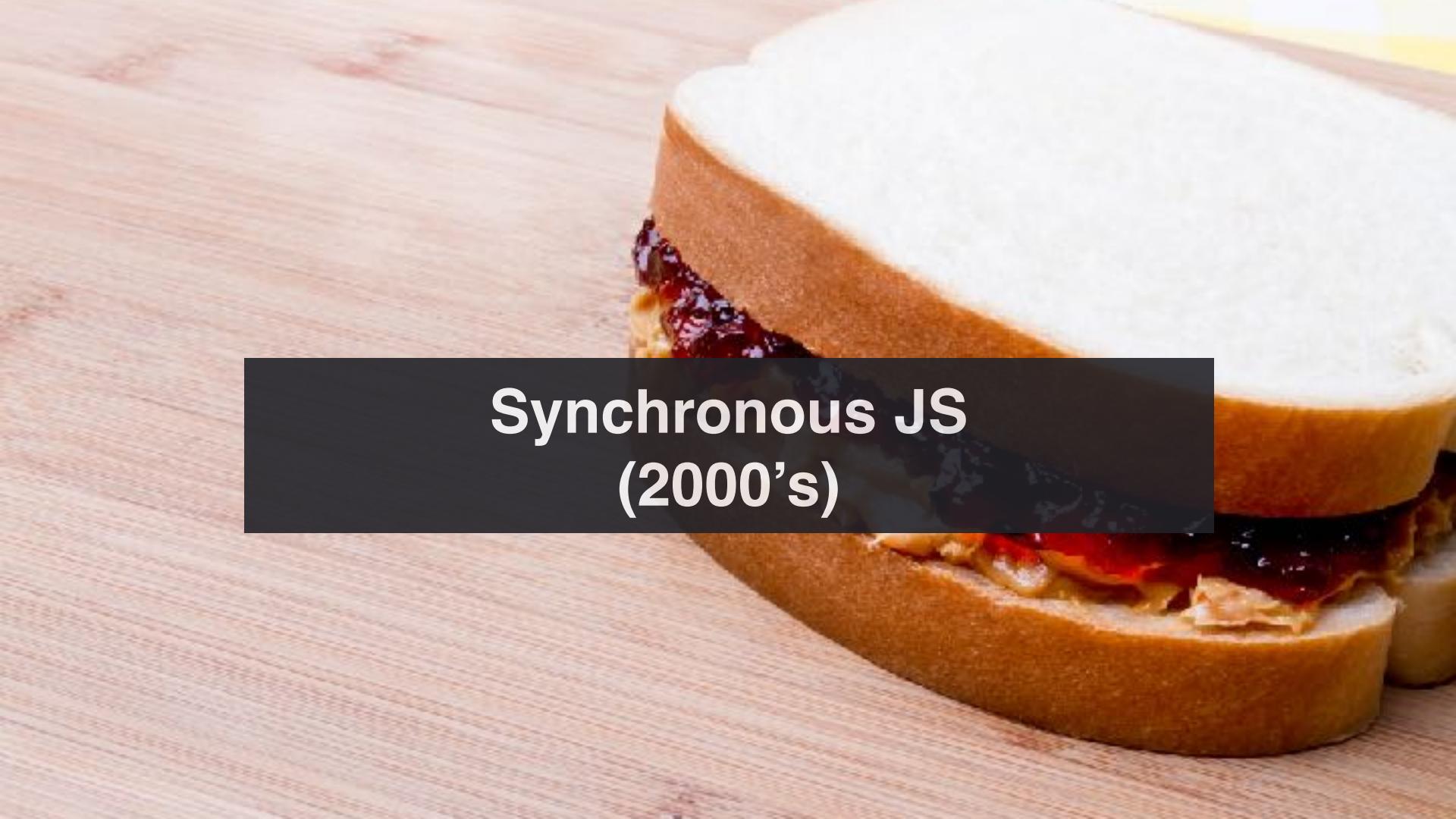
- I want a sandwich!
- Sure
 - “Worker, Make a sandwich”
- A new customer arrives:
I want a sandwich!
- Sure
 - “Other worker,
make a sandwich”
- Here is your sandwich
customer 1



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, filled with peanut butter and jelly. The sandwich is cut in half diagonally, with one half resting on top of the other. It is placed on a light-colored wooden surface with a visible grain.

Synchronous JS (2000's)

Code

Result

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 PBnJMaker.fetchBread();
4 PBnJMaker.fetchPeanutButter();
5 PBnJMaker.fetchJam();
6 PBnJMaker.spreadPeanutButter();
7 PBnJMaker.spreadJam();
8 let result = PBnJMaker.closeSandwich();
9 console.log("Eat that " + result);
```

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

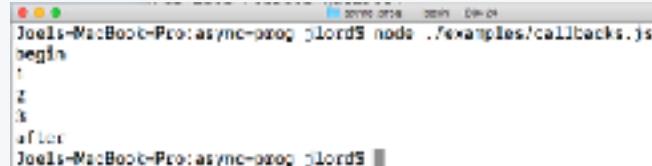


Callbacks (2010's)

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");
```



```
Joel's-MacBook-Pro:async-prog jlord$ node ./examples/callbacks.js
begin
1
2
3
after
Joel's-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");
```

```
Deals-MacBook-Pro:asycn-prog jlores$ node ./examples/callbacks.js
begin
after
This is the content of the file
Deals-MacBook-Pro:asycn-prog jlores$
```

What is a callback

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

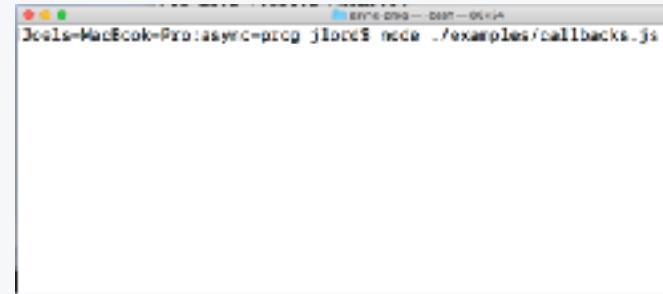
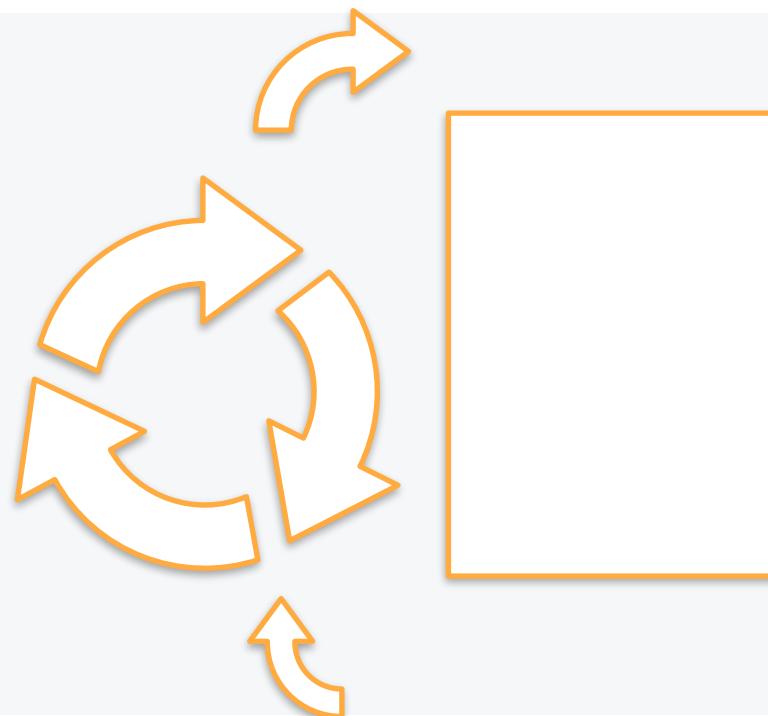
```
1 //Asynchronous
2 .....
3 .....(err, data) => {
4     console.log(data);
5 }..
6 .....
```

```
Deals-MacBook-Pro:asycn-prog jlores$ node ./examples/callbacks.js
begin
atnc
This is the content of the file
Deals-MacBook-Pro:asycn-prog jlores$
```

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");
```

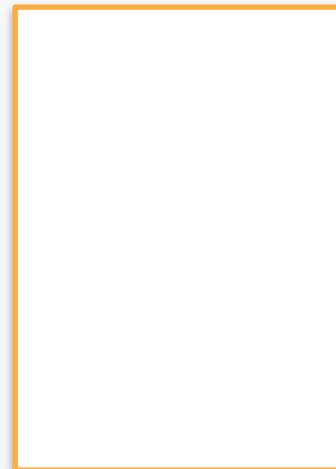
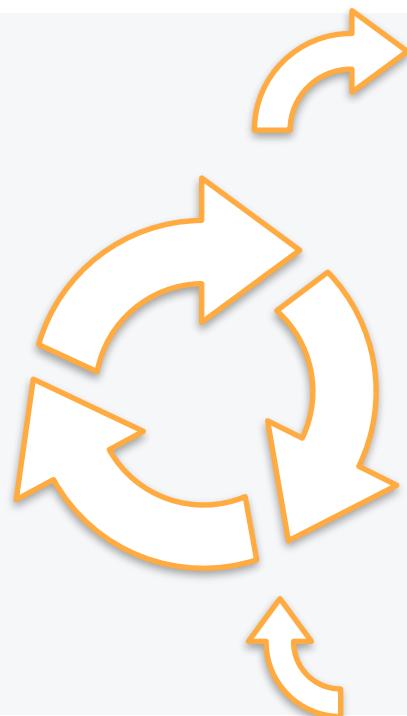


@joel__lord
#confoo

What is a callback

Asynchronous callback and the event loop

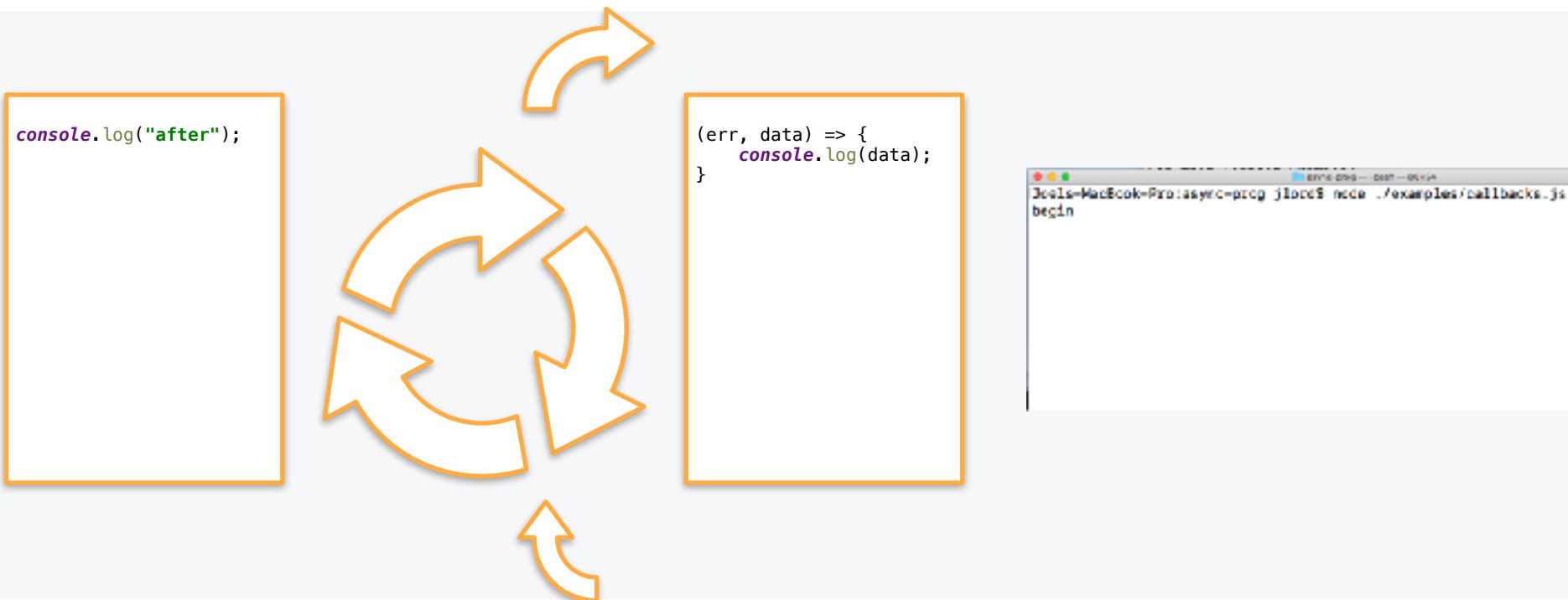
```
fs.readFile("./cb.txt",  
  (err, data) => {  
    console.log(data);  
  };  
  
console.log("after");
```

A screenshot of a terminal window titled "Terminal - Oct 24". The command "node ./examples/callbacks.js" is entered, followed by the output "begin". This demonstrates the execution of an asynchronous callback function.

@joel__lord
#confoo

What is a callback

Asynchronous callback and the event loop



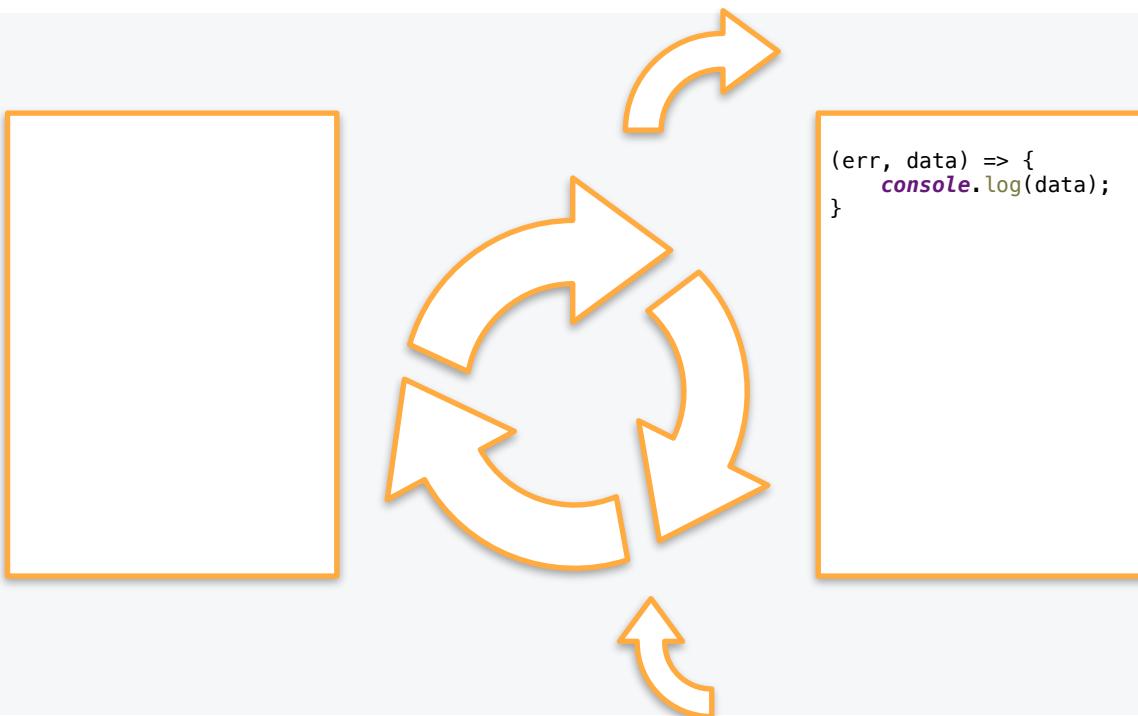
```
3cells-MacBook-Pro:asyncc=prog jlord$ node ./examples/callbacks.js
begin
```



@joel__lord
#confoo

What is a callback

Asynchronous callback and the event loop



A screenshot of a terminal window titled "Terminal - Oct 24". The command entered is "node ./examples/callbacks.js begin". The output shows the word "begin" printed to the console.



@joel__lord
#confoo

What is a callback

Asynchronous callback and the event loop

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



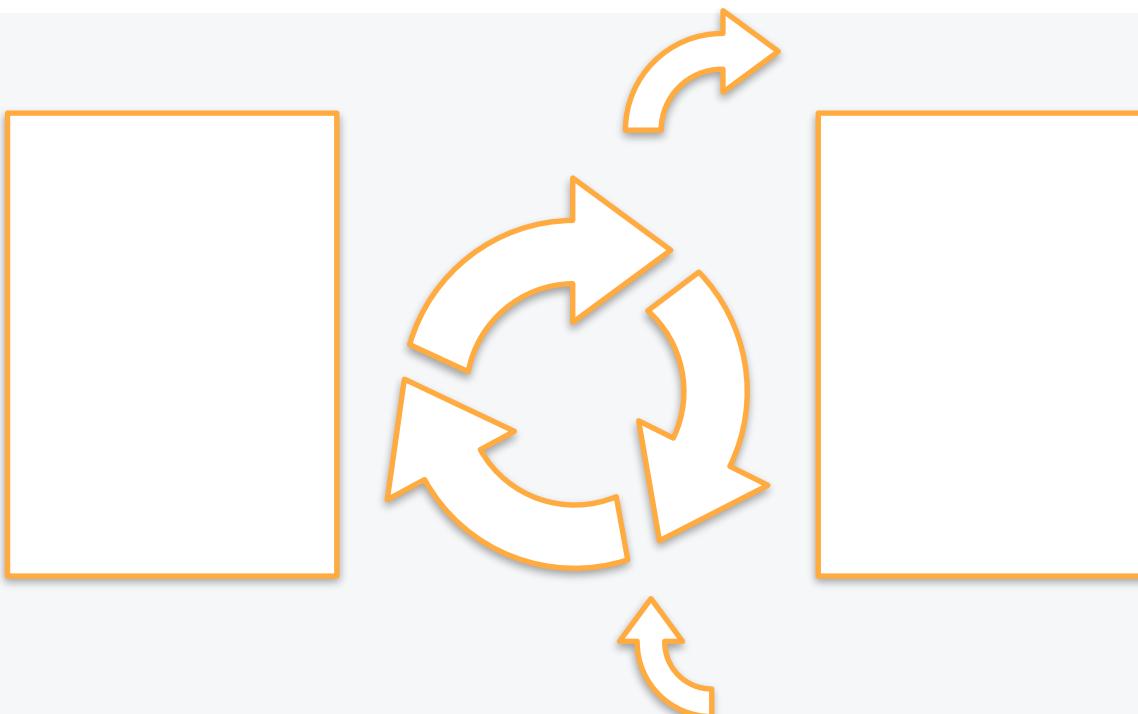
```
Joels-MacBook-Pro:asynrc=prog jlord$ node ./examples/callbacks.js  
begin  
a+tic
```



@joel__lord
#confoo

What is a callback

Asynchronous callback and the event loop



```
Joel's-MacBook-Pro:asycn-prog jlord$ node ./examples/callbacks.js
begin
after
This is the content of the file
Joel's-MacBook-Pro:asycn-prog jlord$
```



@joel__lord
#confoo

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

```
Joel's-MacBook-Pro:async-prog jlord$
```

Callbacks

- It works!
- but...

```
1 let PBnJMaker = require("./pbnjmaker");
2
3 FBnJMaker.simulateError();
4
5 FBnJMaker.fetchIngredients((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.spreadJam((err) => {
18            if (err) {
19              console.log("An error occurred while preparing the sandwich");
20            } else {
21              PBnJMaker.closeSandwich((err, result) => {
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});
```

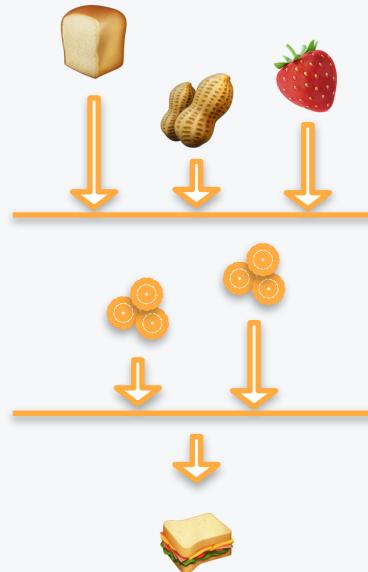
Parallel events



Step by step



Parallel events



Code

```
1 let PnBmaker = require("./pnbmaker");
2
3 let steps = 0;
4 // PBnBmaker simulatesError();
5
6 function fetchIngredients() {
7   PBnBmaker.fetchBread(handleIngredients);
8   PnBmaker.fetchPeanutButter(handleIngredients);
9   PnBmaker.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   PBnBmaker.spreadPeanutButter(preparationStepCompleted);
22   PnBmaker.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 === 4) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(result) {
34   PnBmaker.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();
```

Result

```
Joel-MacBook-Pro:sync-prog jlord$
```

1

Code

```
1 let ingredients = 0;
2 let steps = 0;

3 function fetchIngredients() {
4     PBnMakr.fetchBread(handleIngredients);
5     PnDmaker.fetchPeanutButter(handleIngredients);
6     PBnDmaker.fetchJam(handleIngredients);
7 }

8 function handleIngredients(error) {
9     if (error) return console.log("An error occurred while fetching the ingredients");
10    ingredients++;
11    if (ingredients === 3) {
12        prepareSandwich();
13    }
14}

15 function prepareSandwich() {
16    PBnMakr.spreadBreadOnButter(preparationStepCompleted);
17    PnDmaker.spreadJam(preparationStepCompleted);
18}

19 function preparationStepCompleted(error) {
20    if (error) return console.log("An error occurred while preparing the sandwich");
21    steps++;
22    if (steps === 4) {
23        completeSandwich();
24    }
25}

26 function completeSandwich(result) {
27    PBnDmaker.closeSandwich(error, result) => {
28        if (error) {
29            console.log("An error occurred");
30        } else {
31            console.log("Eat that " + result);
32        }
33    });
34}

35 function fetchIngredients() {
36    PBnMakr.fetchBread(handleIngredients);
37    PnDmaker.fetchPeanutButter(handleIngredients);
38    PBnDmaker.fetchJam(handleIngredients);
39}

40 fetchIngredients();
```

Result

```
Joel's-MacBook-Pro:sync-prog jlord$
```

1

Code

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

Result

```
Joel-MacBook-Pro:sync-prog jlord$
```

1

Code

```
1 let PnBmaker = require("./pnbmaker");
2 let ingredients = {};
3 let steps = 0;
4 // PBnBmaker.simulatesError();
5
6 function fetchIngredients() {
7   PnBmaker.fetchBread(handleIngredients);
8   PnBmaker.fetchPeanutButter(handleIngredients);
9   PnBmaker.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   PnBmaker.spreadPeanutButter(preparationStepCompleted);
22   PnBmaker.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 === 4) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(result) {
34   PnBmaker.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();
```

Result

```
Joel-MacBook-Pro:sync-prog jlord$
```

1

Code

```
1 let PBnMakr = require("./pbnmaker");
2 let ingredients = {};
3 let steps = 0;
4 // PBnMakr simulates errors();
5
6 function fetchIngredients() {
7   PBnMakr.fetchBread(handleIngredients);
8   PBnMakr.fetchPeanutButter(handleIngredients);
9   PBnMakr.fetchJam(handleIngredients);
10 }
11
12 function handleIngredients(error) {
13   if (error) 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   PBnMakr.spreadPeanutButter(preparationStepCompleted);
22   PBnMakr.spreadJam(preparationStepCompleted);
23 }
24
25 function preparationStepCompleted(error) {
26   if (error) return console.log("An error occurred while preparing the sandwich");
27   steps++;
28   if (steps === 4) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(result) {
34   PBnMakr.closeSandwich(result, result) === {
35     if (result) {
36       console.log("An error occurred");
37     } else {
38       console.log("Eat that " + result);
39     }
40   });
41 }
42 fetchIngredients();
```

Result

```
Joel-MacBook-Pro:sync-prog jlord$
```

1

Code

```
1 let PBnMakr = require("./pbnmaker");
2 let ingredients = 0;
3 let steps = 0;
4 // PBnMakr simulates error();
5
6 function fetchIngredients() {
7   PBnMakr.fetchBread(handleIngredients);
8   PBnMakr.fetchPeanutButter(handleIngredients);
9   PBnMakr.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   PBnMakr.spreadPeanutButter(preparationStepCompleted);
22   PBnMakr.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 === 4) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(result) {
34   PBnMakr.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();
```

Result

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

1

Code

```
1 let PBnMakr = require("./pbnmaker");
2 let ingredients = {};
3 let steps = 0;
4 // PBnMakr simulates error();
5
6 function fetchIngredients() {
7   PBnMakr.fetchBread(handleIngredients);
8   PBnMakr.fetchPeanutButter(handleIngredients);
9   PBnMakr.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   PBnMakr.spreadPeanutButter(preparationStepCompleted);
22   PBnMakr.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 === 4) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(result) {
34   PBnMakr.closeSandwich(result) === {
35     if (result) {
36       console.log("An error occurred");
37     } else {
38       console.log("Eat that " + result);
39     }
40   });
41 }
42 fetchIngredients()
```

Result

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

1

Code

```
1 let PBnMakr = require("./pbnmaker");
2 let ingredients = {};
3 let steps = 0;
4 // PBnMakr simulates error();
5
6 function fetchIngredients() {
7   PBnMakr.fetchBread(handleIngredients);
8   PBnMakr.fetchPeanutButter(handleIngredients);
9   PBnMakr.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   PBnMakr.spreadPeanutButter(preparationStepCompleted);
22   PBnMakr.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 === 4) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(result) {
34   PBnMakr.closeSandwich(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

```
Jools-MacBook-Pro:sync-prog jlord$
```

1

Code

```
1 let PBnMakr = require("./pbnmaker");
2 let ingredients = {};
3 let steps = 0;
4 // PBnMakr simulates error();
5
6 function fetchIngredients() {
7   PBnMakr.fetchBread(handleIngredients);
8   PBnMakr.fetchPeanutButter(handleIngredients);
9   PBnMakr.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   PBnMakr.spreadPeanutButter(preparationStepCompleted);
22   PBnMakr.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 === 4) {
29     completeSandwich();
30   }
31 }
32
33 function completeSandwich(result) {
34   PBnMakr.closeSandwich(err, result) => {
35     if (err) {
36       console.log("An error occurred");
37     } else {
38       console.log("Eat that " + result);
39     }
40   };
41 }
```

Result

```
Joel-MacBook-Pro:pbnsync-prog jlord$
```

1



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?

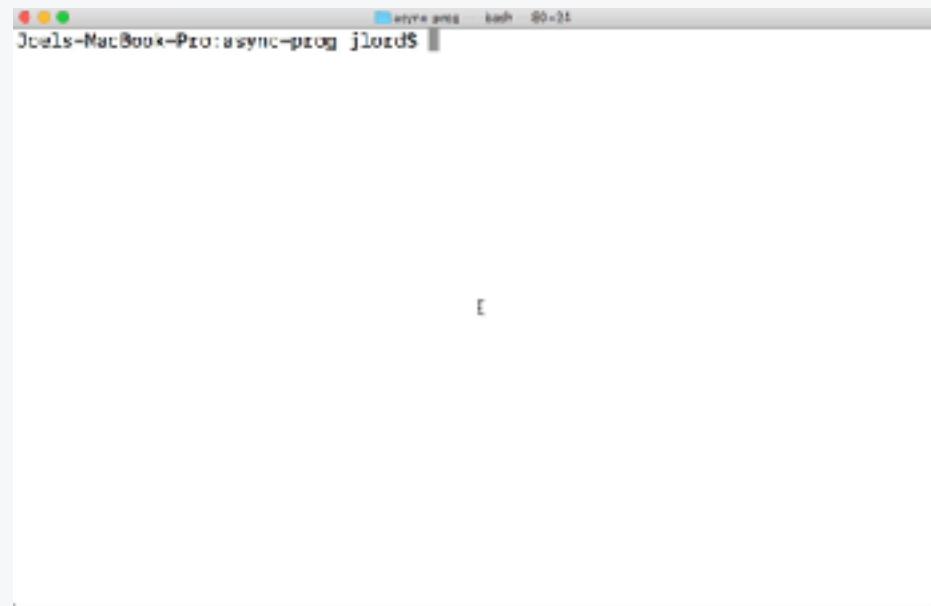
- A representation of a callback
- Returns a .then() method
- Error handling simplified

```
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



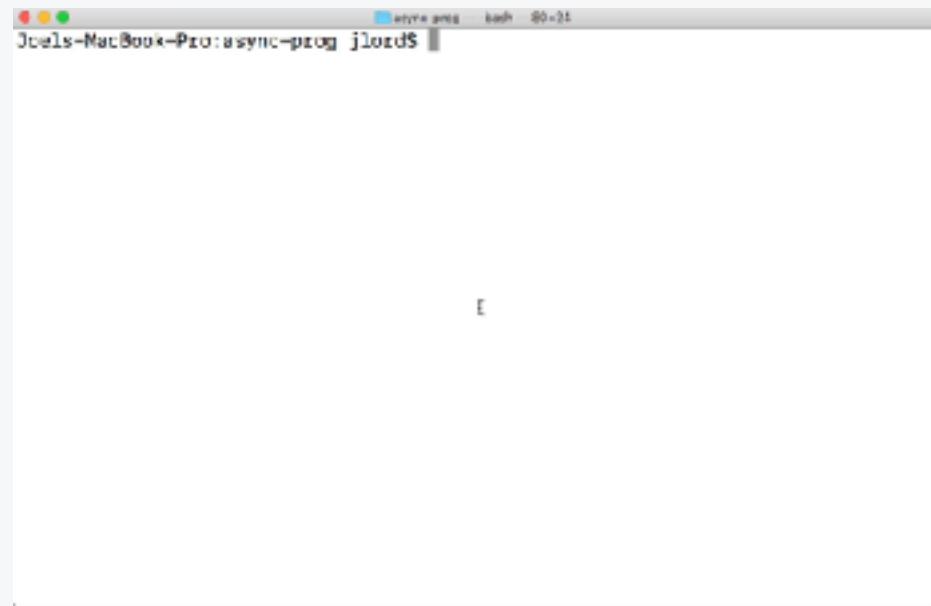
A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command-line interface for running the asynchronous JavaScript code. The code itself is identical to the one shown in the "Code" section, but it has been executed, resulting in the following output:

```
[1]
```

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

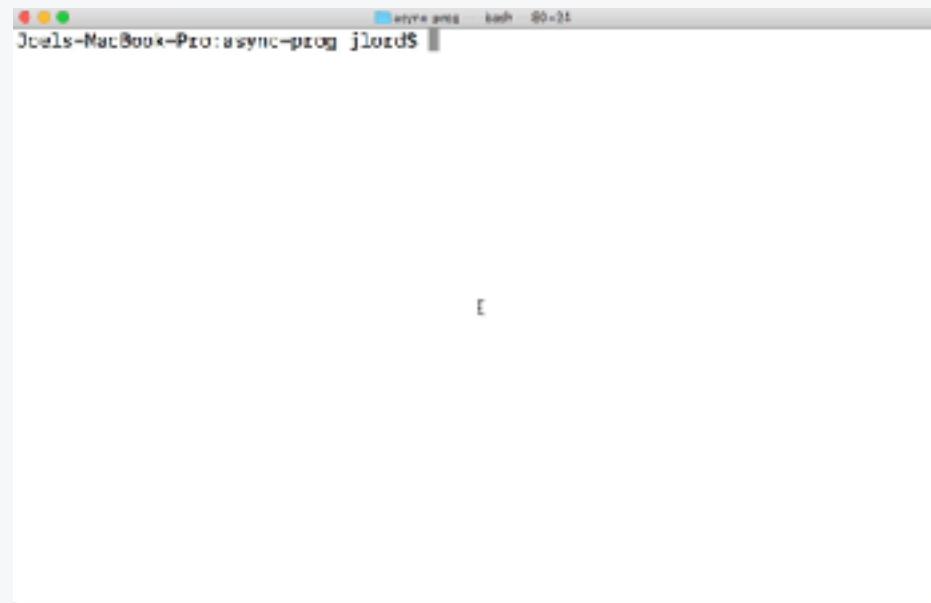


A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command-line interface for running the code. The output of the code execution is visible at the bottom of the terminal window.

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

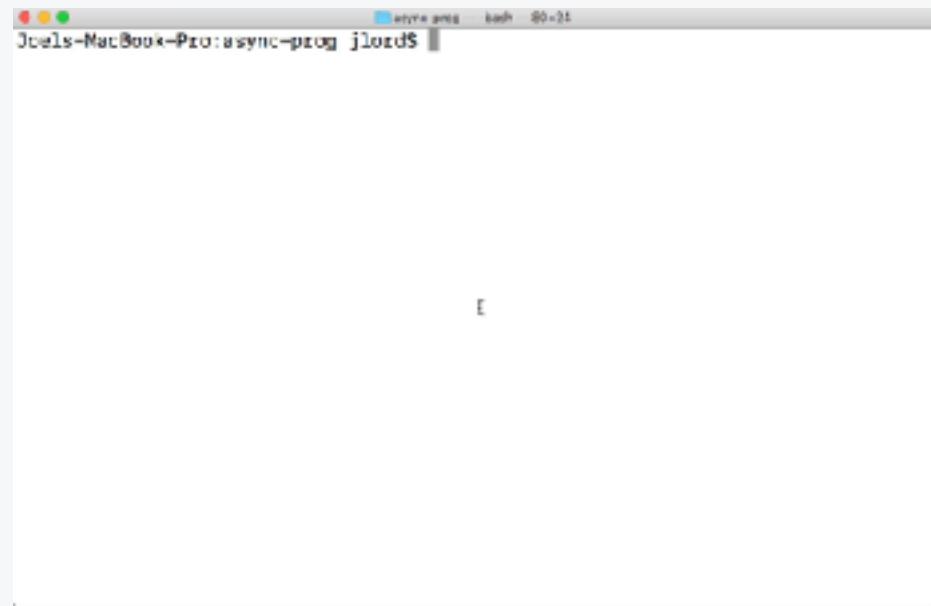


A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command-line interface for running the code. The output of the code execution is visible at the bottom of the terminal window.

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

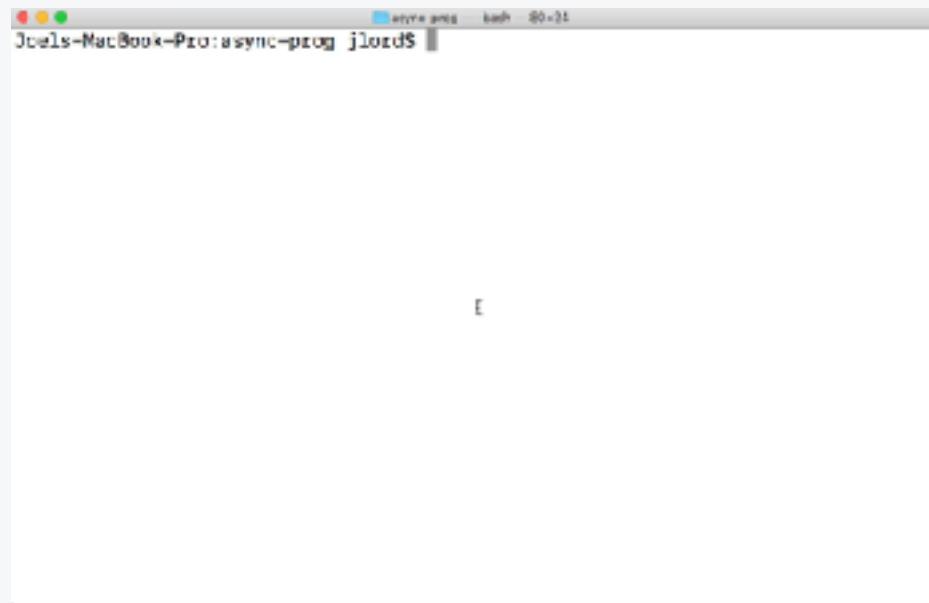


A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command-line interface for running the code. The output of the code execution is visible at the bottom of the terminal window.

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

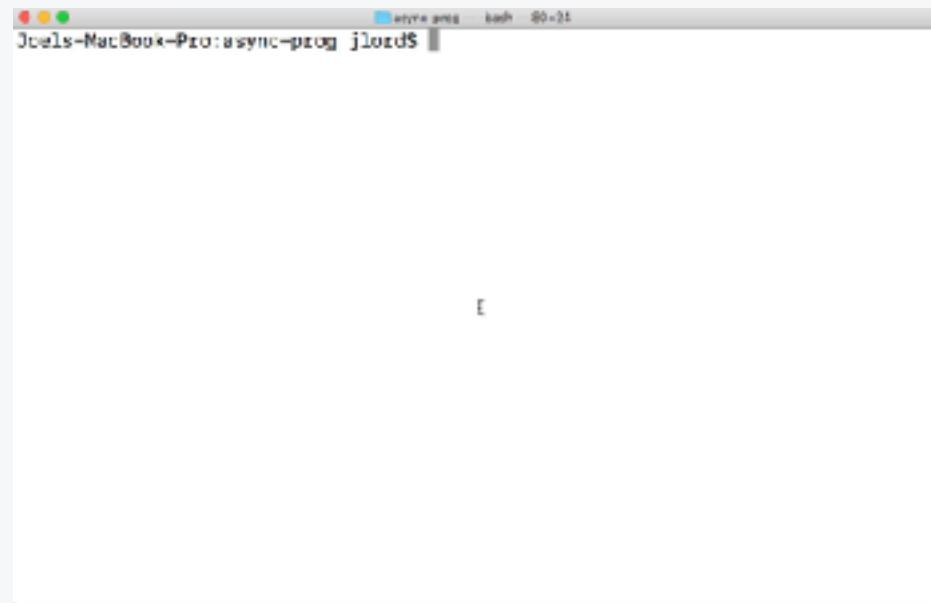


A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command-line interface for running the code. The code itself is identical to the one shown in the "Code" section, but the output is empty, indicating no visible output was produced.

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

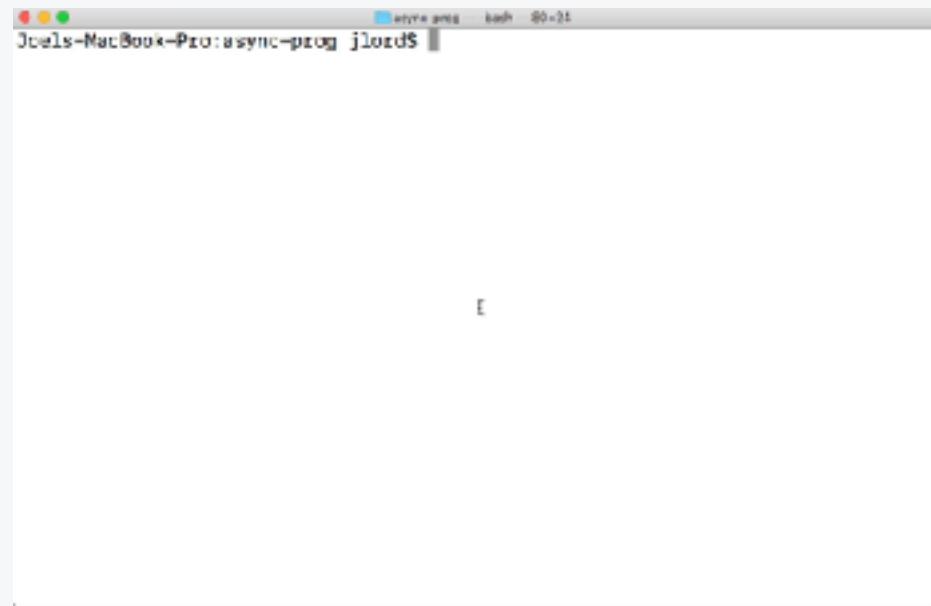


A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command-line interface for running the code. The code itself is identical to the one shown in the "Code" section, but the output is empty, indicating no visible output was produced.

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

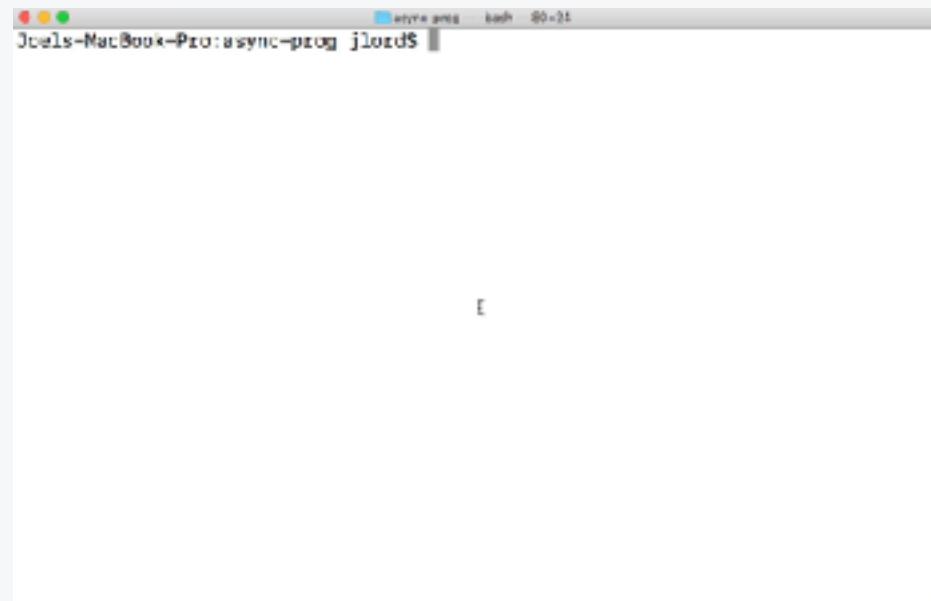


A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command being run and a blank white space where the output would normally appear.

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

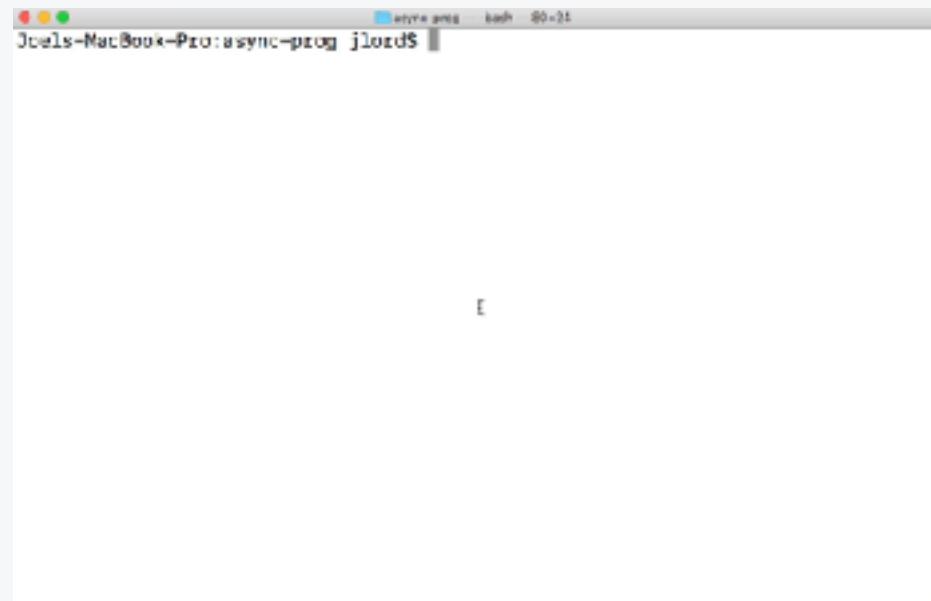


A screenshot of a terminal window titled "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command being run and the resulting output. The output consists of a single character, likely a carriage return or a blank line, indicating that the script has completed its execution.

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 "Joels-MacBook-Pro:async-prog jlord\$". The window shows the command being run and the resulting output. The output consists of a single character, likely a carriage return or a blank line, indicating that the script has completed its execution.

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

```
[Joel's-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

```
Joel's-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

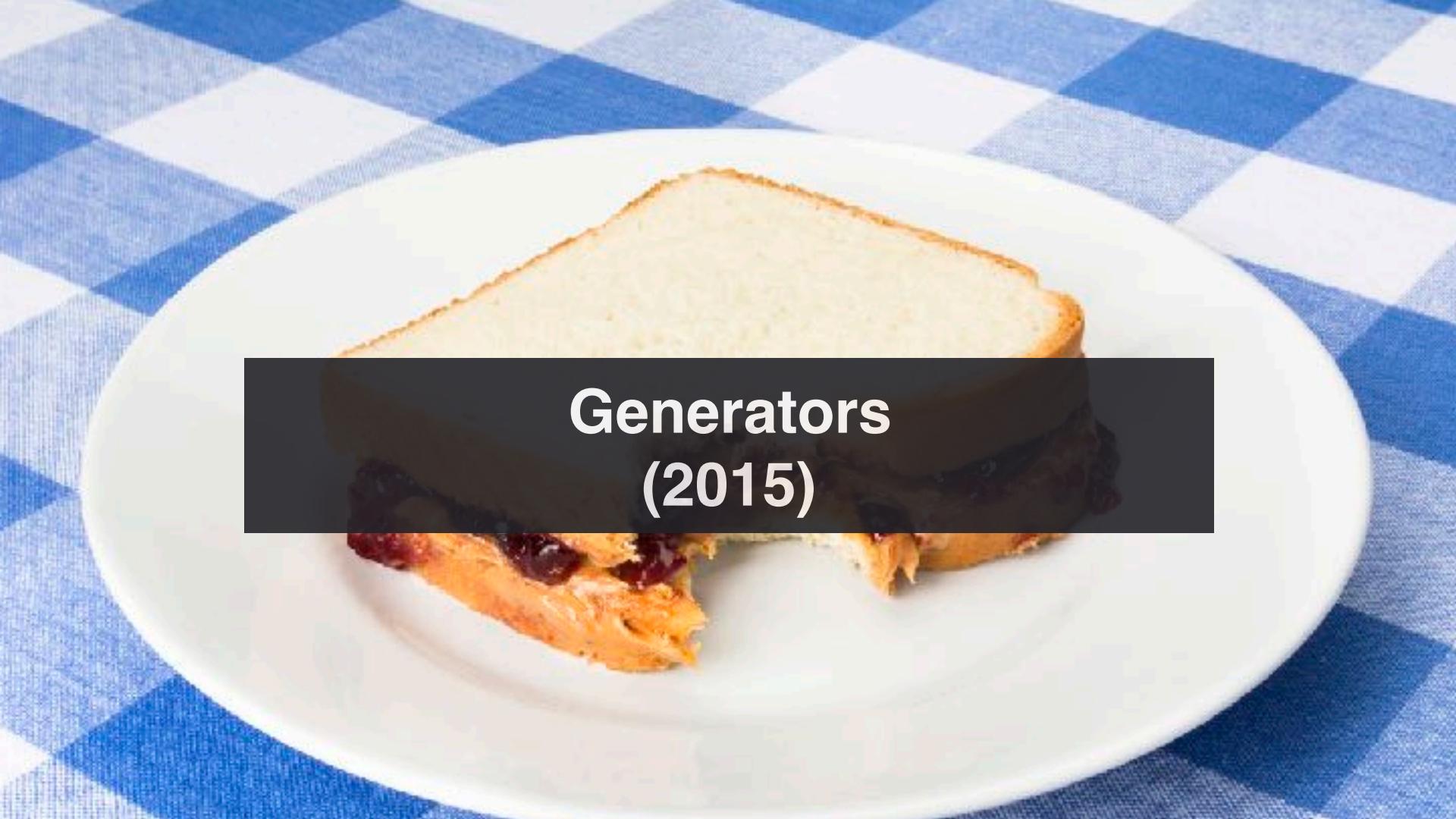
- `.all()`
- `.race()`

```
● ● ●  
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
```

A slice of blueberry pie is served on a white plate. The pie has a golden-brown lattice crust and is filled with blueberries. A dark rectangular overlay covers the middle portion of the pie. Inside the overlay, the words "Generators (2015)" are written in a white, sans-serif font.

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--){ 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>=1) {
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>=1) {
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>=1) {
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

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

Async Generators

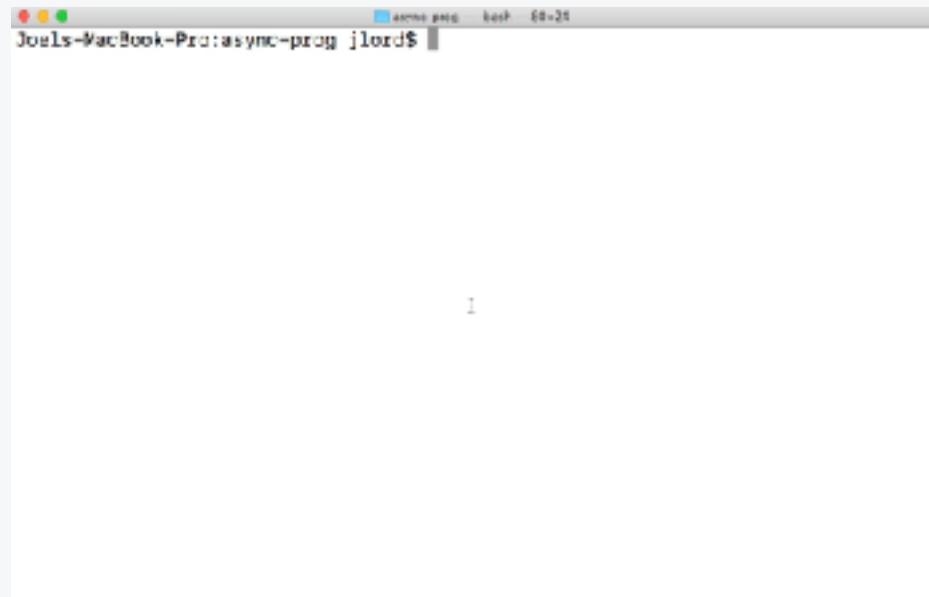
- Combining the power of promises and generators

```
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

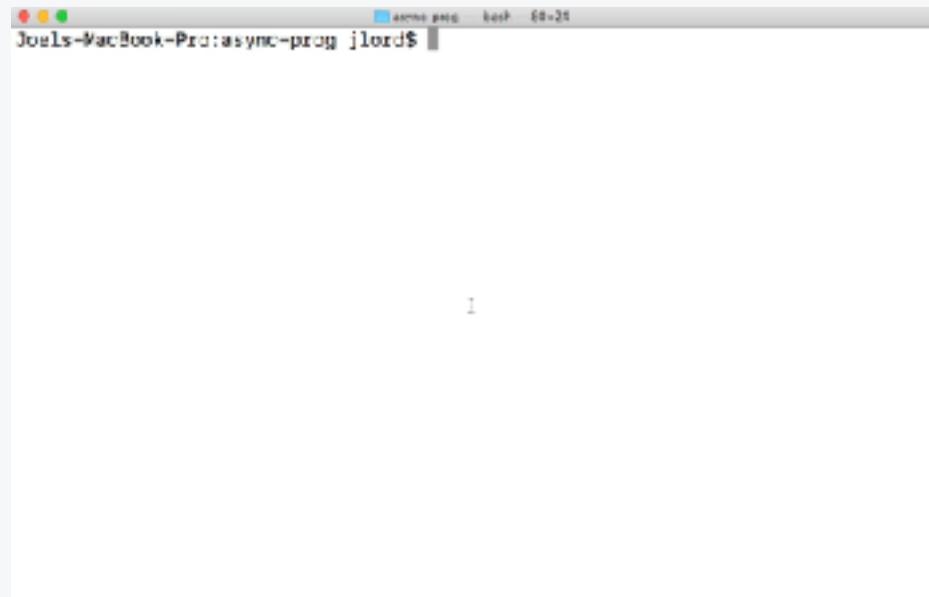


The screenshot shows a terminal window titled 'Joels-MacBook-Pro:async-prog jlord\$'. The window contains a single character: a small blue square with a white 'I' inside, indicating an input cursor.

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



The screenshot shows a terminal window titled 'Joels-MacBook-Pro:async-prog jlord\$'. The window contains a single line of text: 'I'. This indicates that the asynchronous code has been executed and is waiting for its final output.

A photograph of a peanut butter and jelly sandwich on a white plate. The sandwich is made with whole grain bread, peanut butter, and jelly. In the background, there is a jar of red jam, a glass of iced coffee with a spoon, and some peanuts and pomegranate seeds scattered around.

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));
```

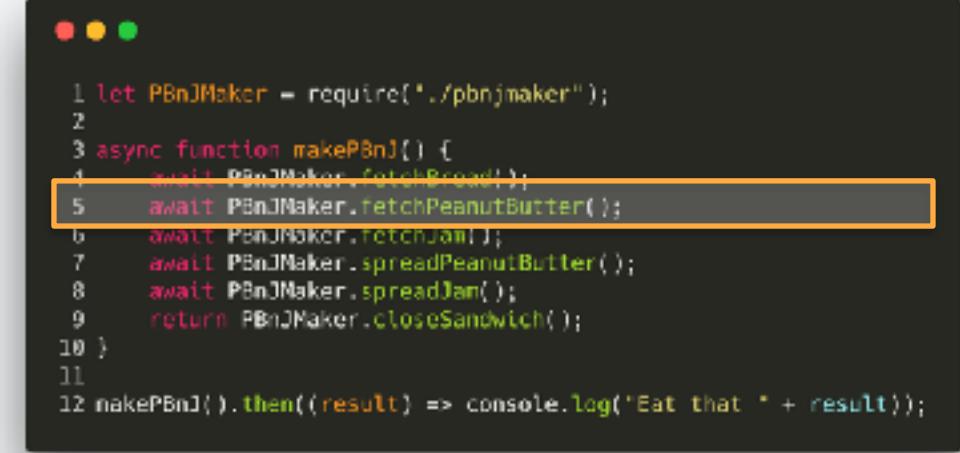
What is async/await?

- 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("jarFetched", () => 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("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 let ingredientFetched = 0;
4 let ingredientsSpreaded = 0;
5
6 let ingredientFetched = () => {
7   ingredientFetched++;
8   if (ingredientFetched === 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("breadPatched", (ingredientFetched));
21 PBnJMaker.Events.on("peanutButterFetched", (ingredientFetched));
22 PBnJMaker.Events.on("jamFetched", (ingredientFetched));
23
24 PBnJMaker.Events.on("ingredientFetched", () => {
25   PBnJMaker.spreadPBnutButter();
26   PBnJMaker.screamLast();
27 });
28
29 PBnJMaker.Events.on("peanutButtonSpreaded", (spreaded));
30 PBnJMaker.Events.on("jamspreaded", (spreaded));
31
32 PBnJMaker.Events.on("ingredentsSpreaded", () => {
33   PBnJMaker.closeSandwich();
34 });
35
36 PBnJMaker.Events.on("conditionClosed", (result) => console.log("Eat that " + result));
37
38 PBnJMaker.fetchBread();
39 PBnJMaker.fetchPBnutButter();
40 PBnJMaker.fetchJam();
41
42
```

```
● ○ ■
async-prog — bash — 80x24
GR2VVVAAHTD$async-prog.jl$cd$clear

```

A close-up photograph of a peanut butter and jelly sandwich. The sandwich is cut diagonally, showing the layers of bread, peanut butter, and jelly. It sits 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 });
```



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. The sandwich is cut in half and filled with a generous amount of red jam, likely strawberry or raspberry. The bread appears to be white or light-colored. The background is blurred, showing a wooden surface and a white plate.

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);
                });
            });
        });
    });
});
```



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