

# Shorter Syntax

1

## Arrow Functions

() => {}

```
// ES5
```

```
var add = function (num1, num2) {  
    return num1 + num2;  
}
```

```
// ES6
```

```
var add = (num1, num2) => num1 + num2
```

# Lexical This & (Arrow) => {Functions}

2

## Arrow functions =>

`this` is *undefined*

```
this.x = 'yes';  
var y = function(){  
  "use strict";  
  console.log(this.x);  
};  
y(); // TypeError...
```

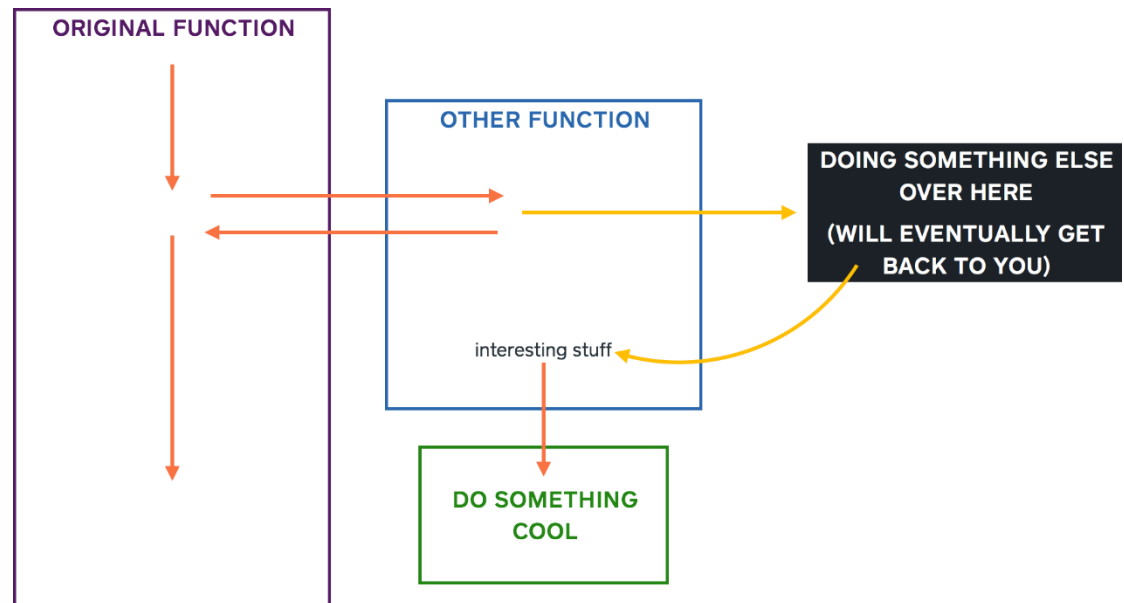
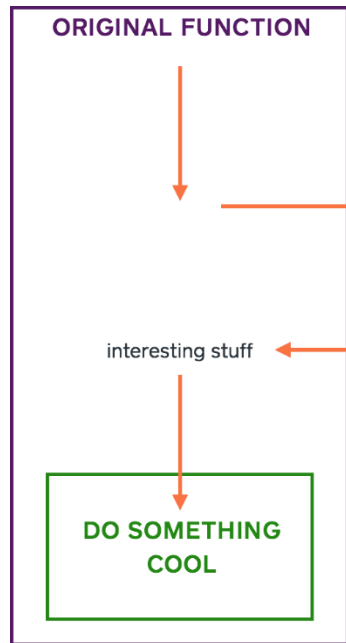
`this` from outer scope

```
this.x = 'yes';  
let y = () => {  
  "use strict";  
  console.log(this.x);  
};  
y(); // 'yes'
```

No need for var that = this, `bind` with  
arrow functions

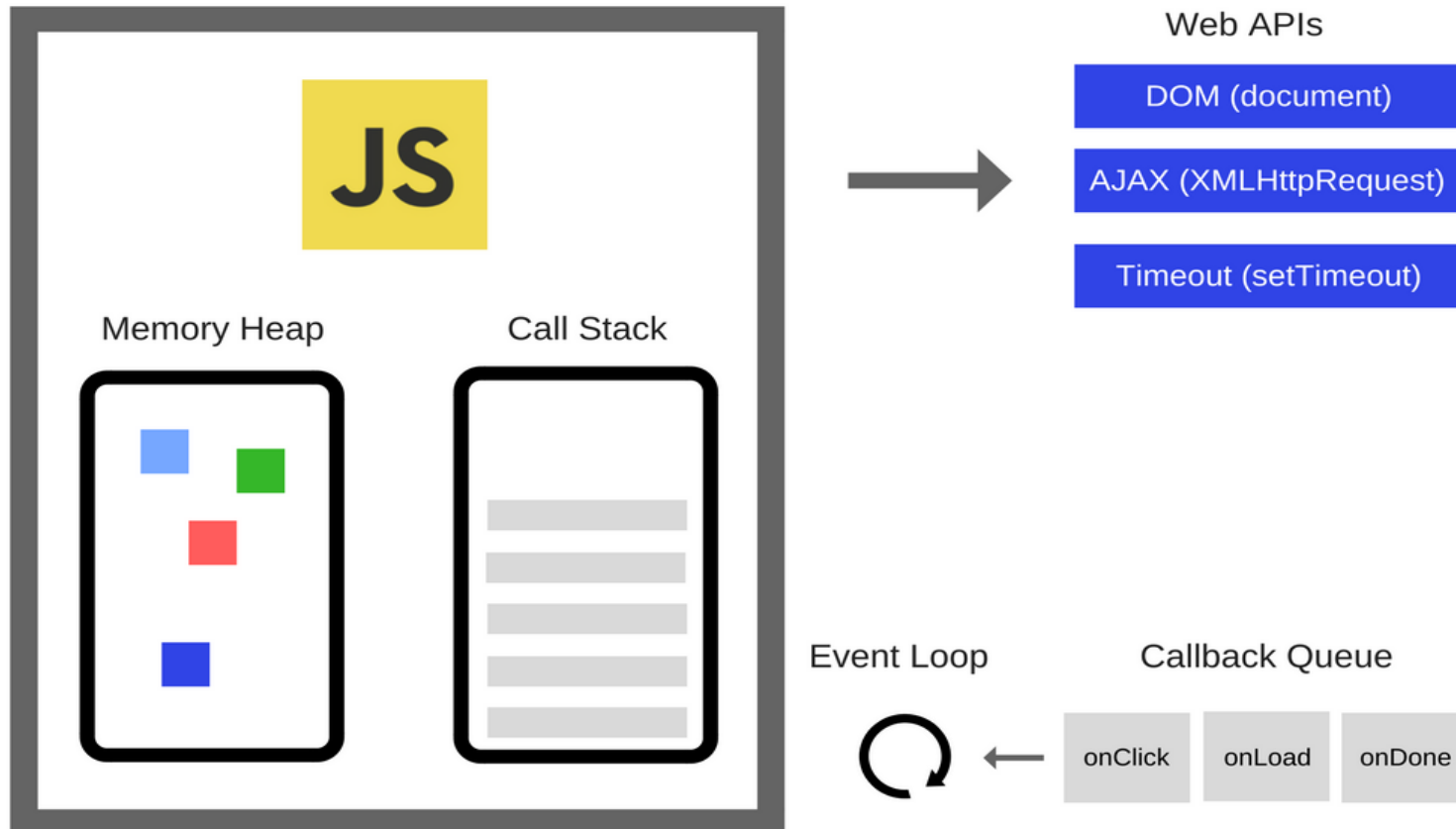
# DEMO

# Asynchronous functions



# Callbacks everywhere!

5



# DEMO

# The Promise — of a — BURGER PARTY

written by @kasamari

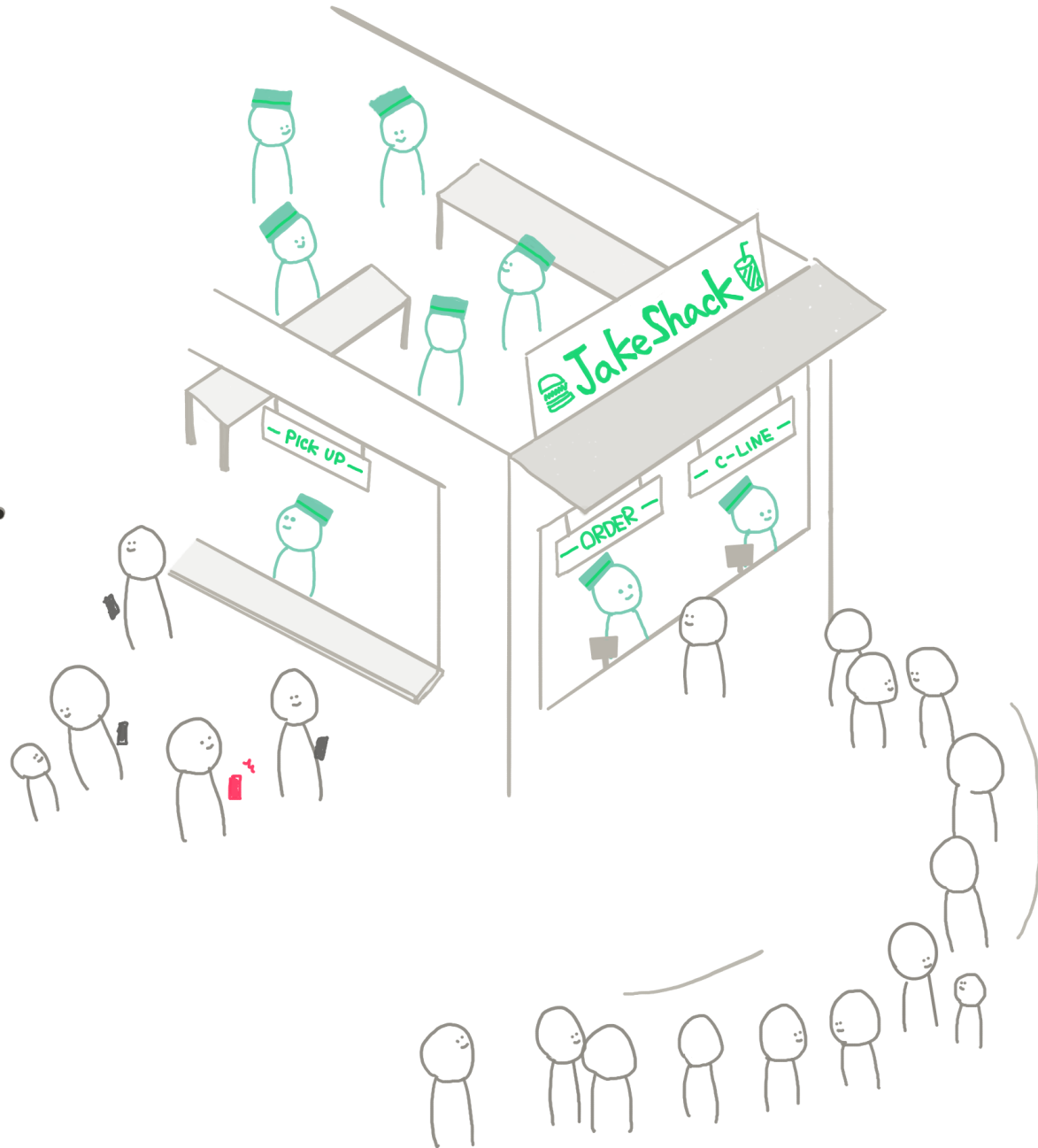
---

---

A quest to understanding  
JavaScript Promise

---

---



# Promise

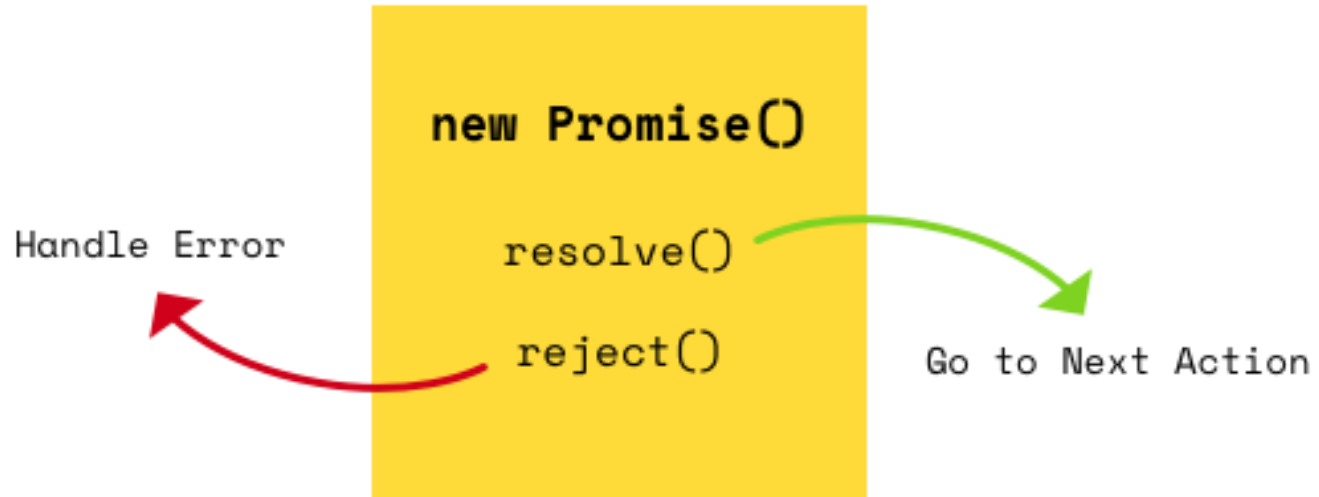
## What is Promise?

- A promise is an object that represents the return value or the thrown exception that the function may eventually provide.
- In other words, a promise represents a value that is not yet known.  
*A promise is an **asynchronous value**.*
- The core idea behind promises is that a promise represents the result of an asynchronous operation.
- A Promise has 3 possible states
  - Pending
  - Fulfilled
  - Rejected



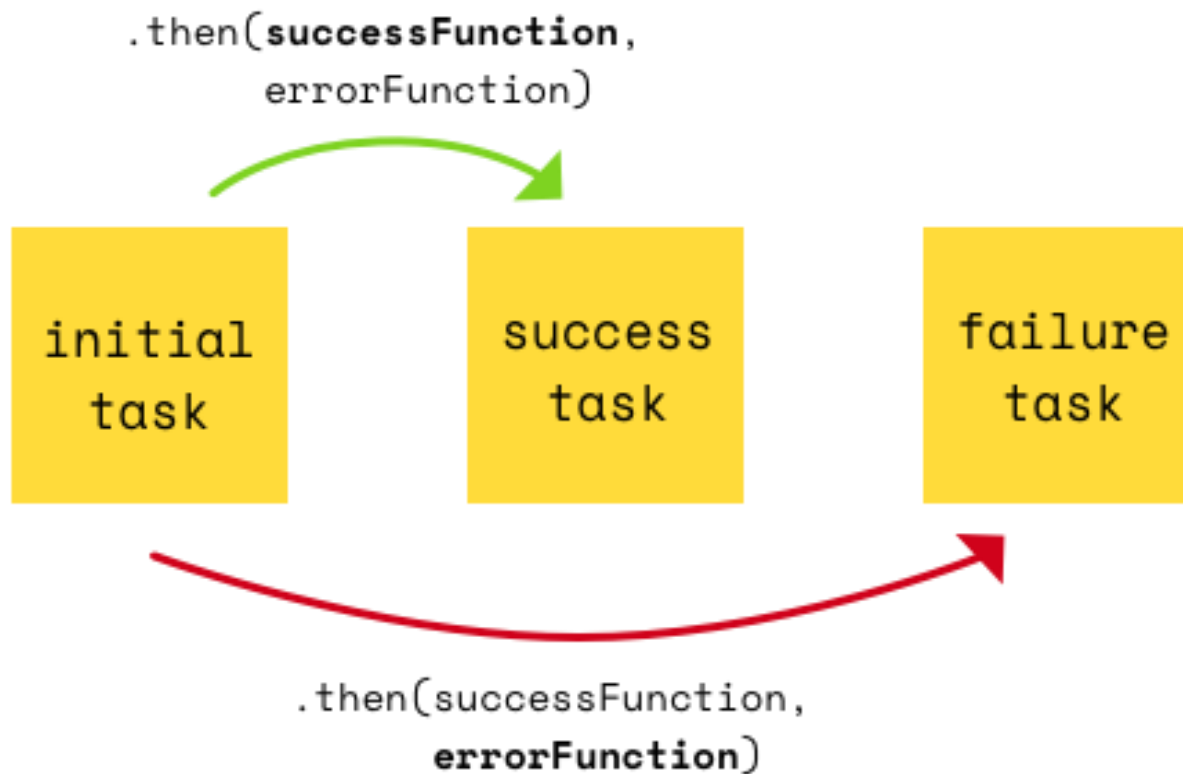
# Promise

9



# Promise:then

10



# Promise

11

```
var p = new Promise(  
  function(resolve, reject){  
    ...  
    if(something)  
      resolve({});  
    else{  
      reject(new Error());  
    }  
  })  
p.then(  
  function(data){  
    ...  
  },  
  function(err){  
    ...  
  }  
);
```

The diagram illustrates the relationship between Promise methods and their corresponding handlers. Two curved arrows originate from the left: one from the `resolve({})` call inside the Promise constructor, pointing to the `function(data)` success handler in the `p.then()` call; and another from the `reject(new Error())` call, pointing to the `function(err)` failure handler. This visualizes how the internal state of the Promise (resolved or rejected) determines which of the two functions in the `then` chain is executed.

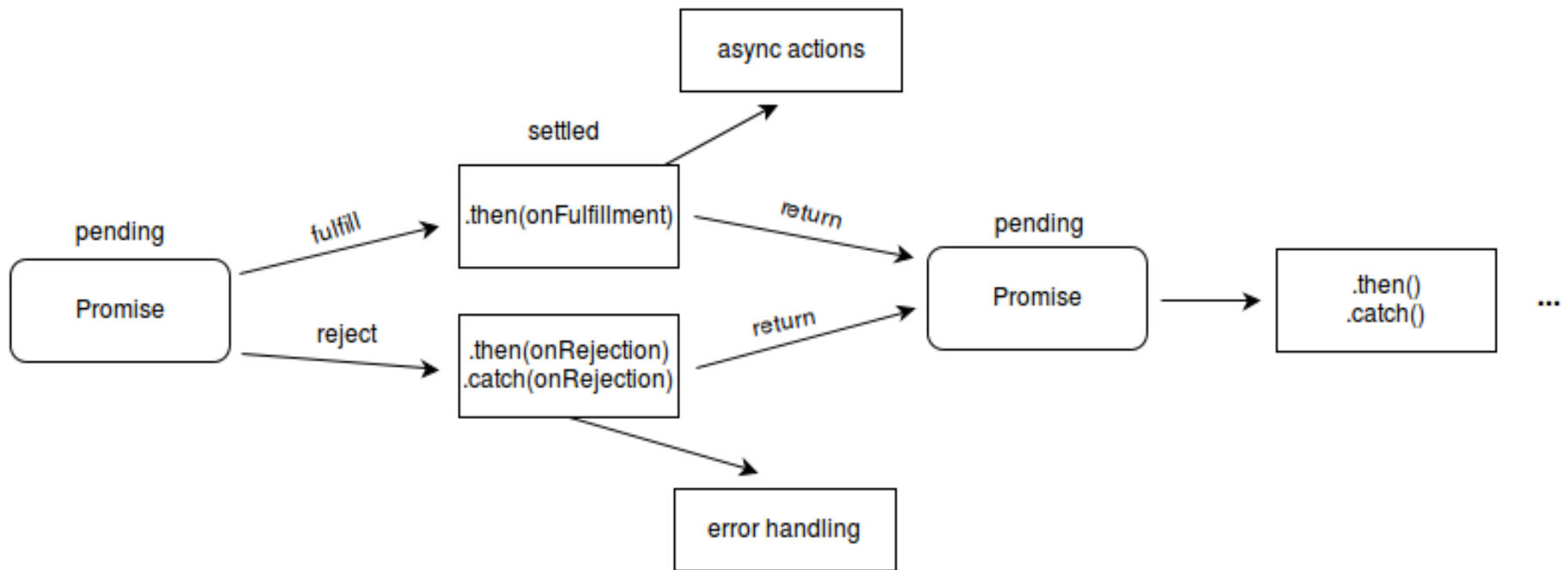
# DEMO

# Callback Hell

```
func1(param, function(err, res) {  
  func2(param, function(err, res) {  
    func3(param, function(err, res) {  
      func4(param, function(err, res) {  
        func5(param, function(err, res) {  
          func6(param, function(err, res) {  
            func7(param, function(err, res) {  
              func8(param, function(err, res) {  
                func9(param, function(err, res) {  
                  // Do something...  
                });  
              });  
            });  
          });  
        });  
      });  
    });  
  });  
});
```

# Chaining the promises

14



# DEMO