

PV247

Vilém Jeniš

Asynchronous operations

Vilém Jeniš

JAVASCRIPT:

I'm a:

- single threaded
- concurrent language

I have

- a call stack
- an event loop
- a callback queue
- and some other APIs and stuff.

JAVASCRIPT:

I'm a:

- **single threaded**
- concurrent language

I have

- a call stack
- an event loop
- a callback queue
- and some other APIs and stuff.

JAVASCRIPT:

I'm a:

- **single threaded**
- **concurrent language**

I have

- a call stack
- an event loop
- a callback queue
- and some other APIs and stuff.

JAVASCRIPT:

I'm a:

- single threaded
- concurrent language

I have

- **a call stack**
- an event loop
- a callback queue
- and some other APIs and stuff.

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(n) {  
  return multiply(n, n);  
}  
  
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}  
  
printSquare(4);
```



stack

Call Stack



```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```

stack

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}  
  
function square(n) {  
  return multiply(n, n);  
}  
  
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}  
  
printSquare(4);
```

stack

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

square(n)

printSquare(4)

main()

Call Stack



```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

multiply(n, n)

square(n)

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

square(n)

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

```
console.log(squared)
```

```
printSquare(4)
```

```
main()
```

Call Stack

```
function multiply(a, b) {  
  return a * b;  
}
```

```
function square(n) {  
  return multiply(n, n);  
}
```

```
function printSquare(n) {  
  var squared = square(n);  
  console.log(squared);  
}
```

```
printSquare(4);
```

stack

printSquare(4)

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
→ printSquare(4);
```

stack

main()

Call Stack

```
function multiply(a, b) {  
    return a * b;  
}  
  
function square(n) {  
    return multiply(n, n);  
}  
  
function printSquare(n) {  
    var squared = square(n);  
    console.log(squared);  
}  
  
printSquare(4);
```



stack

Here's another one...

Async Callbacks & The Call Stack?

```
console.log('hi');  
  
setTimeout(function () {  
  console.log('there');  
}, 5000);  
  
console.log('JSConfEU');
```

stack

```
console.log('hi')
```

```
main()
```

Async Callbacks & The Call Stack?

```
console.log('hi');
```

```
setTimeout(function () {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

stack

```
setTimeout(cb, 5000)
```

```
main()
```

Async Callbacks & The Call Stack?

```
console.log('hi');  
  
setTimeout(function () {  
  console.log('there');  
}, 5000);  
  
console.log('JSConfEU');
```

stack

main()

Async Callbacks & The Call Stack?

```
console.log('hi');  
  
setTimeout(function () {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

stack

```
console.log('JSConfEU')
```

```
main()
```

Async Callbacks & The Call Stack?

```
console.log('hi');  
  
setTimeout(function () {  
  console.log('there');  
}, 5000);  
  
console.log('JSConfEU');
```



stack

Async Callbacks & The Call Stack?

```
console.log('hi');  
  
setTimeout(function () {  
  console.log('there');  
}, 5000);  
  
console.log('JSConfEU');
```

stack

```
console.log('there')
```

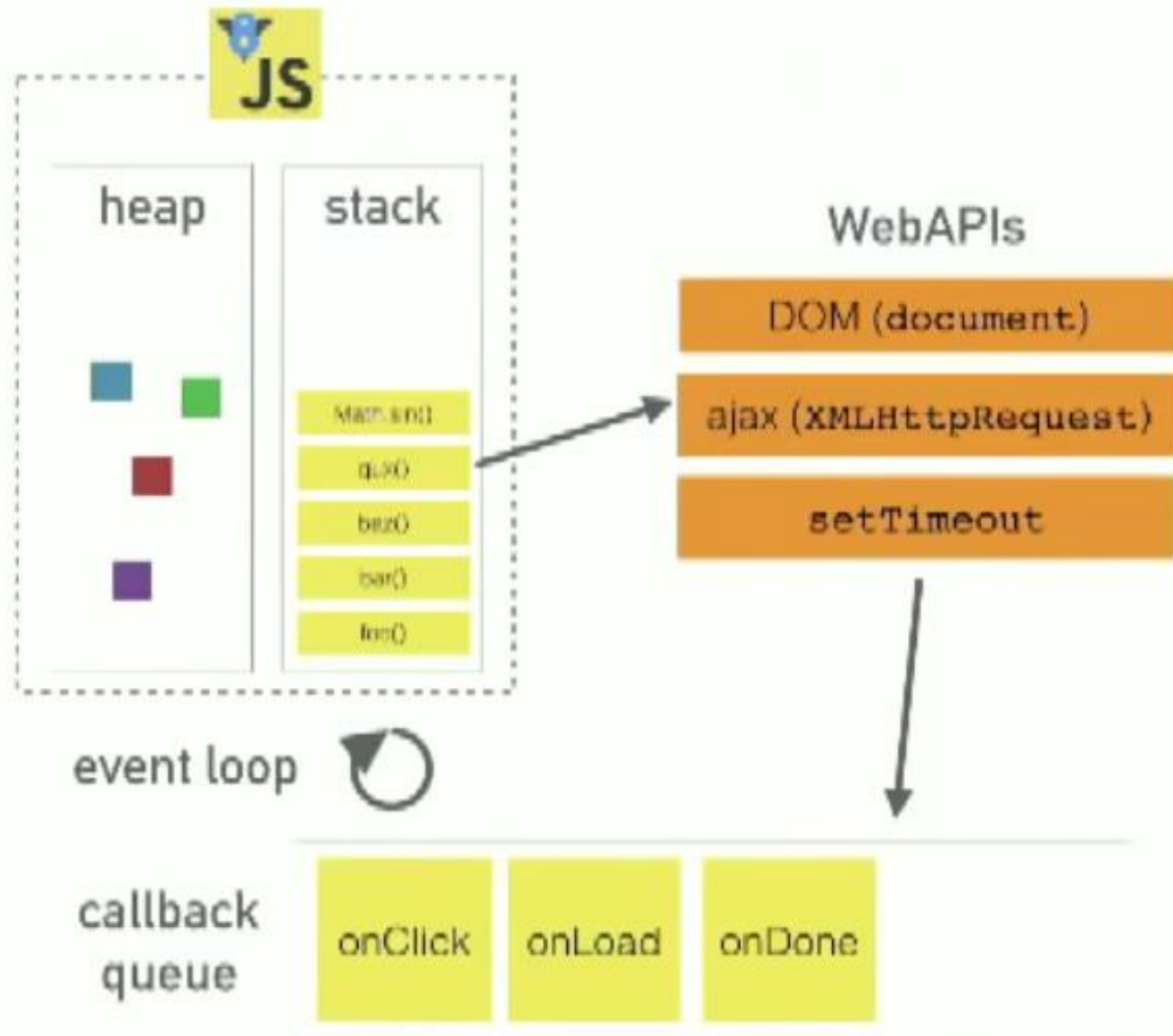
JAVASCRIPT:

I'm a:

- single threaded
- concurrent language

I have

- a call stack
- **an event loop**
- **a callback queue**
- and some other APIs and stuff.



```
JS console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

Console

Hi

stack

log('Hi')

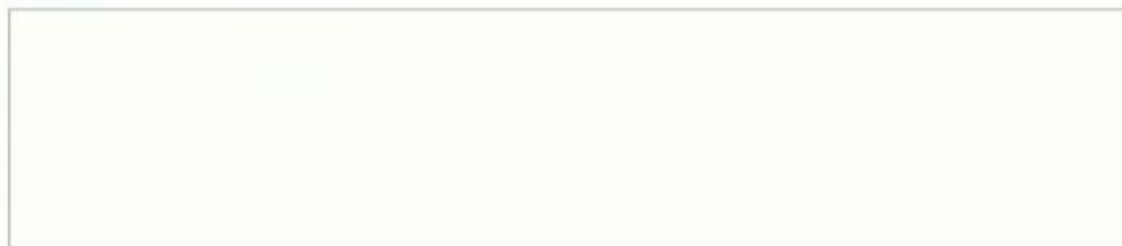
main()

webapis

event loop



task
queue



```
JS console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

Console

Hi

stack

setTimeout cb

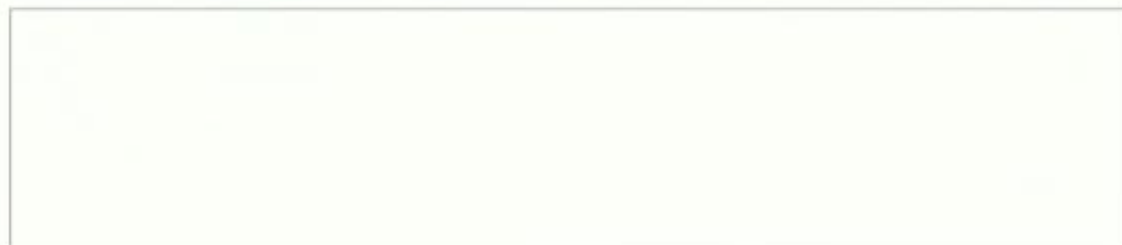
main()

webapis

event loop



task
queue



JS

```
console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

Console


Hi

stack

setTimeout(cb)

main()

webapis

timer()

cb

event loop



task
queue



```
JS console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

Console

Hi

JSConfEU

stack

log('Jsc')

main()

webapis

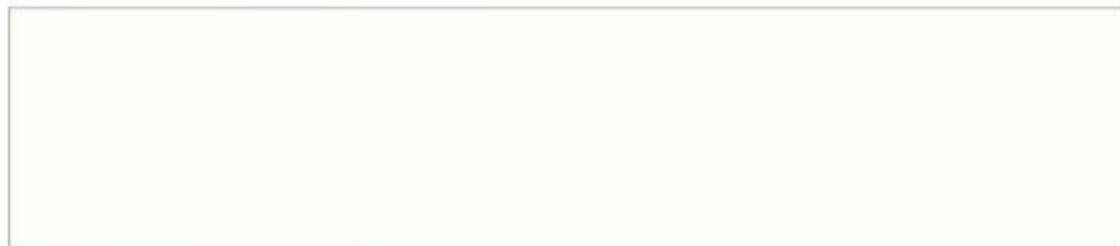
timer(🕒)

cb

event loop



task
queue





```
console.log('Hi');
```

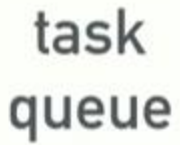
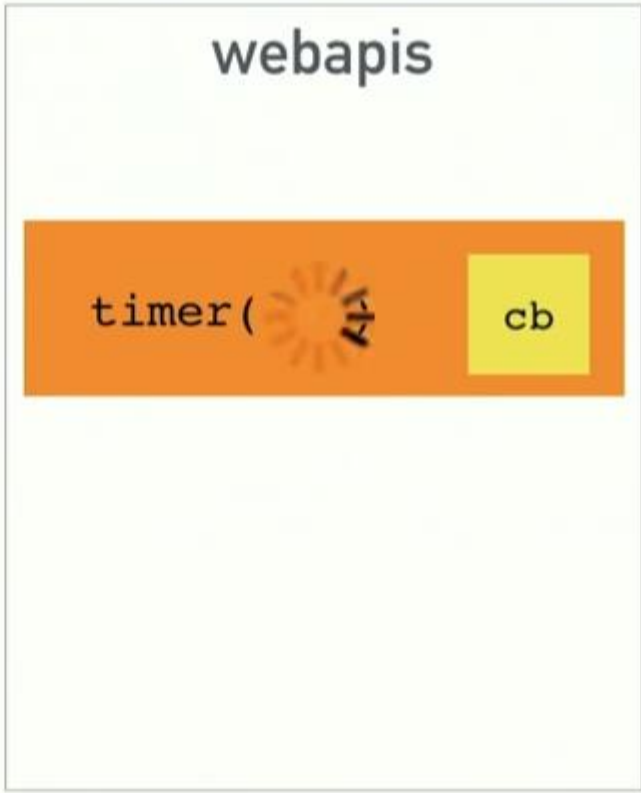
```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

Console

Hi

JSConfEU



JS

```
console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

Console

Hi

JSConfEU

stack

webapis

event loop



task
queue

cb

JS

```
console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```


Console

Hi

JSConfEU

stack

webapis

event loop 

task
queue

cb

JS

```
console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 5000);
```

```
console.log('JSConfEU');
```

Console

Hi

JSConfEU

stack

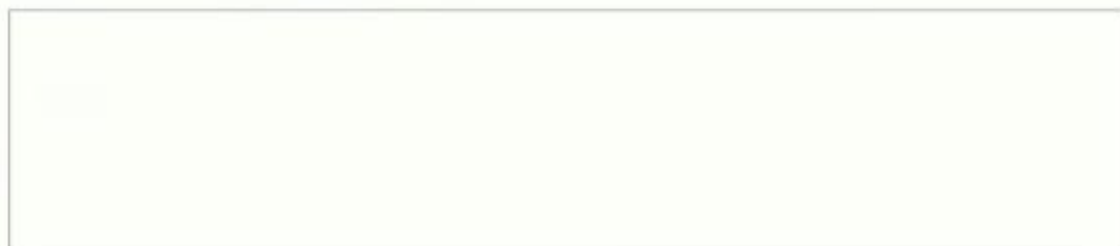
cb

webapis

event loop



task
queue





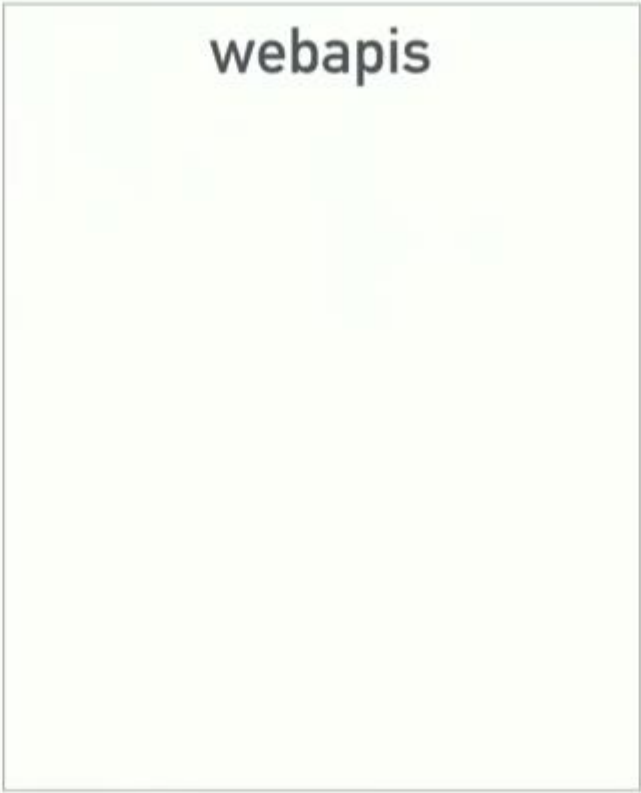
```
console.log('Hi');
```

```
setTimeout(function cb() {
  console.log('there');
}, 5000);
```

```
console.log('JSConfEU');
```

Console

Hi
JSConfEU
there



event loop



JS

```
console.log('Hi');
```

```
setTimeout(function cb() {  
  console.log('there');  
}, 0);
```

```
console.log('JSConfEU');
```

Console

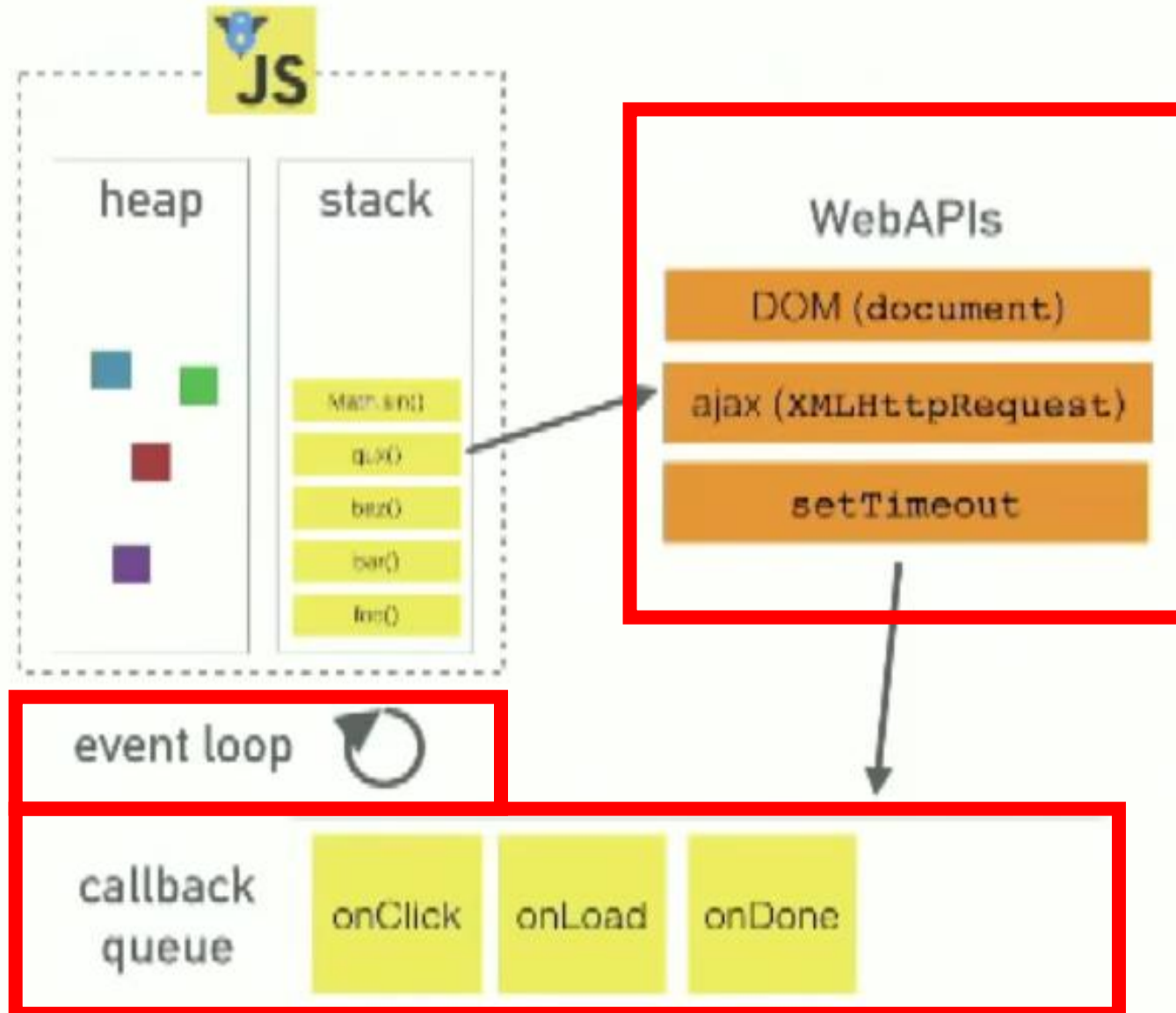
stack

webapis

event loop



task
queue



Event loop

Previous slides borrowed from: <https://www.youtube.com/watch?v=8aGhZQkoFbQ>

More here: <http://latentflip.com/loupe/>

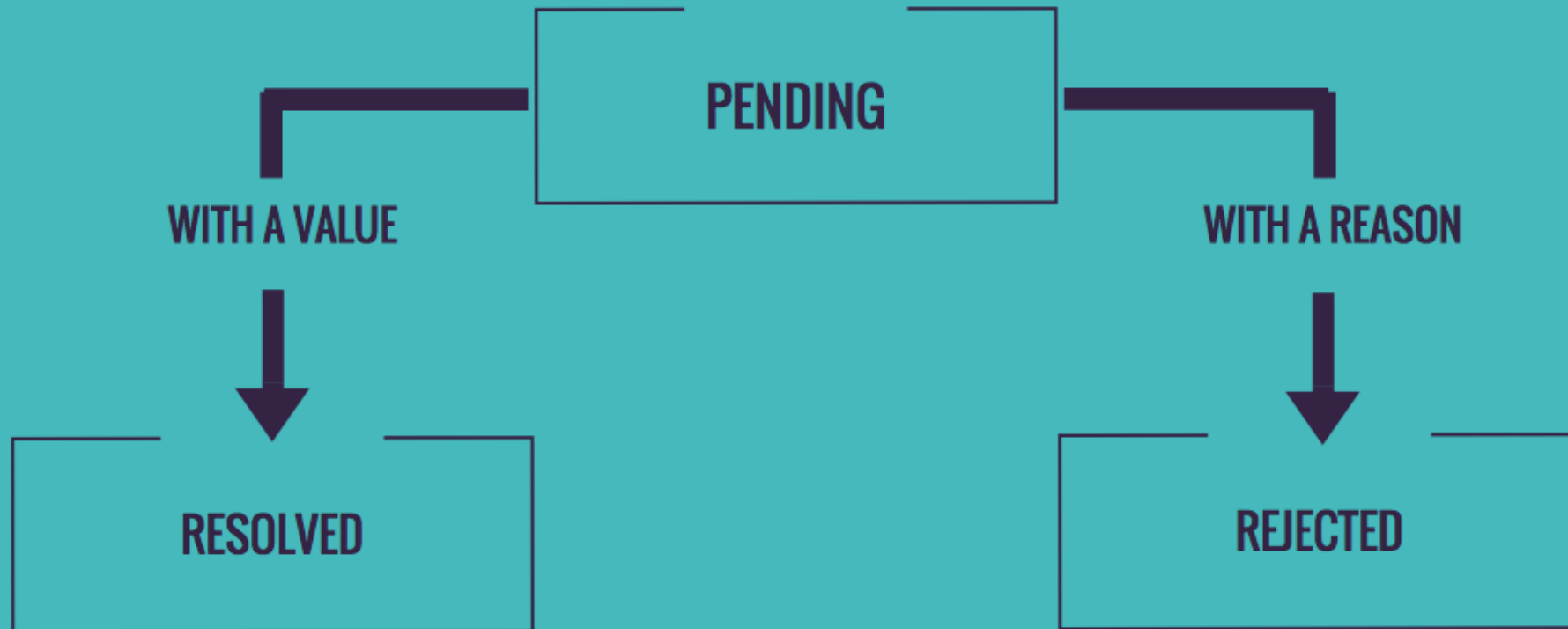
Question time! (Remind me if I forget...)


```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  };
26 }
```



PROMISES

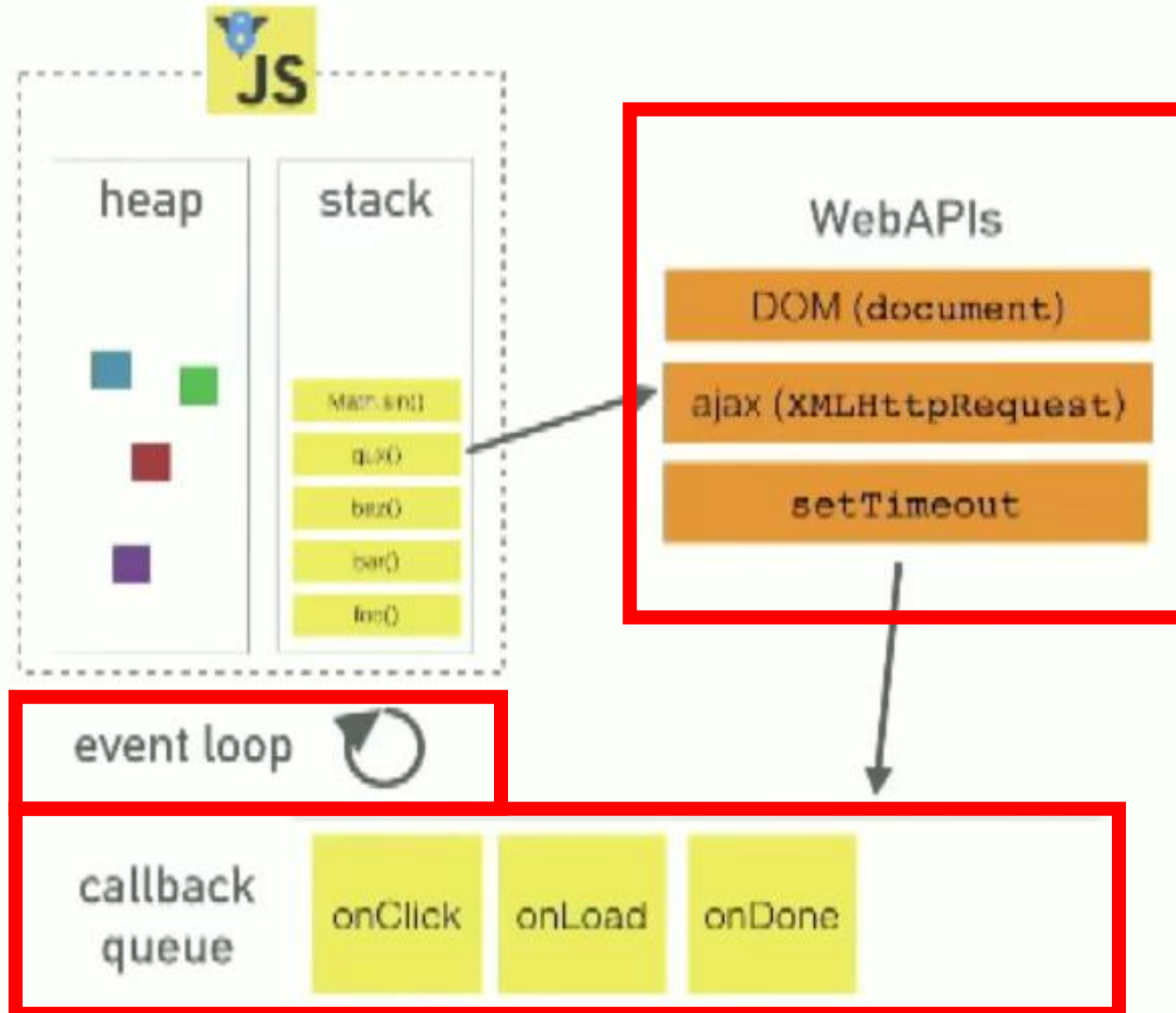
JAVASCRIPT

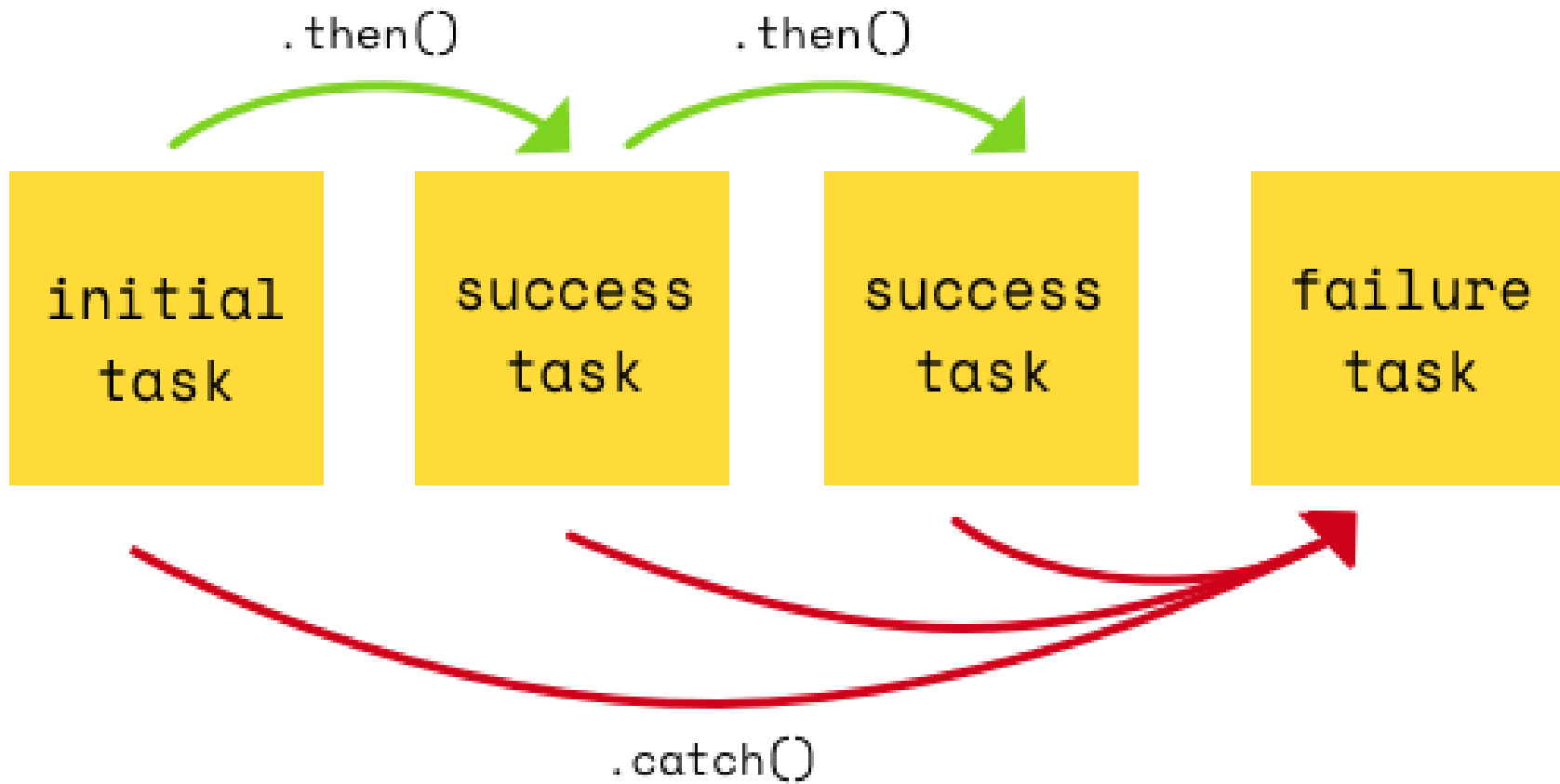


What is a Promise?

A promise is an object that may produce a single value some time in the future: either a resolved value, or a reason that it's not resolved (e.g., a network error occurred). A promise may be in one of 3 possible states: fulfilled, rejected, or pending. Promise users can attach callbacks to handle the fulfilled value or the reason for rejection.

Promises are eager, meaning that a promise will start doing whatever task you give it as soon as the promise constructor is invoked.

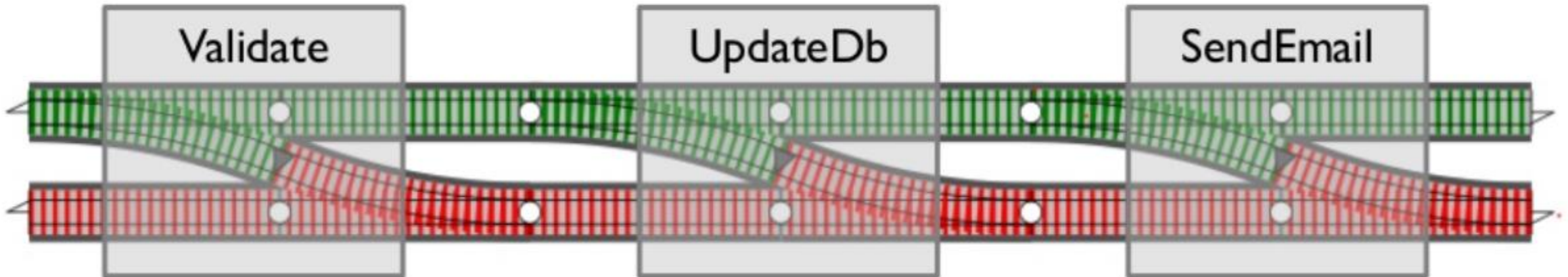




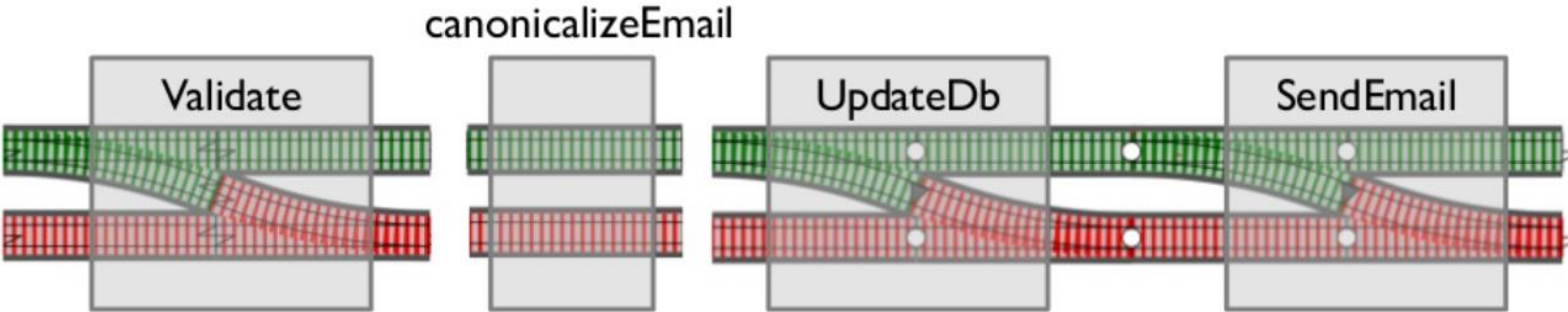
```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  };
26 }
```



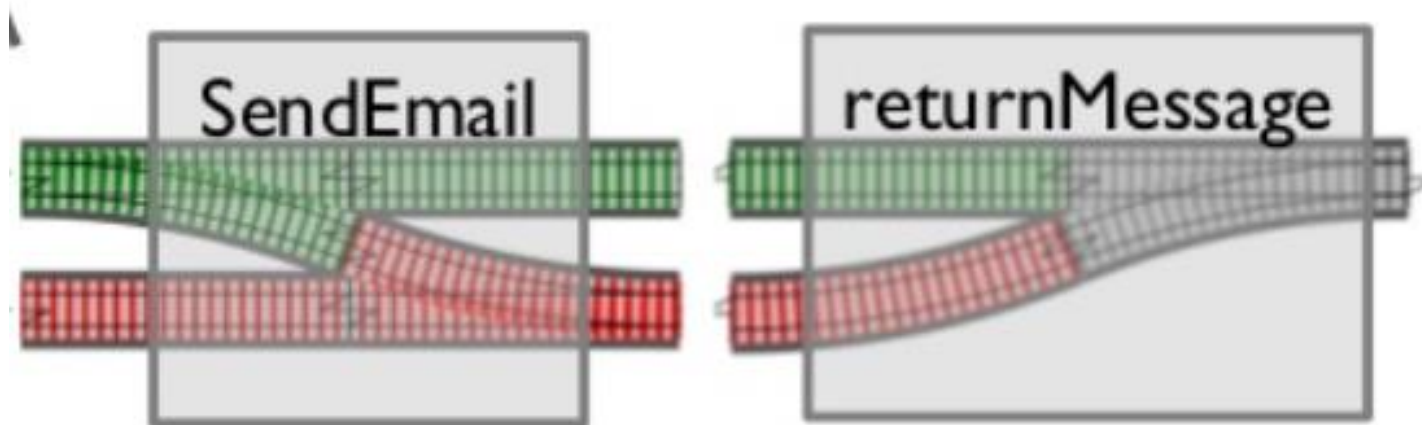
Promises as a Railway



What if nothing can go wrong?



How to stop an error I've dealt with?



Source: <https://fsharpforfunandprofit.com/rop/> <- Seriously... Look at the presentation at least once!

It's all the same.

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