XCCELERATE

D24 - Promise

•••

World of JS

Single-Thread

```
console.log('Ready to write file');
fs.writeFileSync('abc.txt', 'file-content-here');
console.log('Writing done');
```

World of JS

Single-Thread

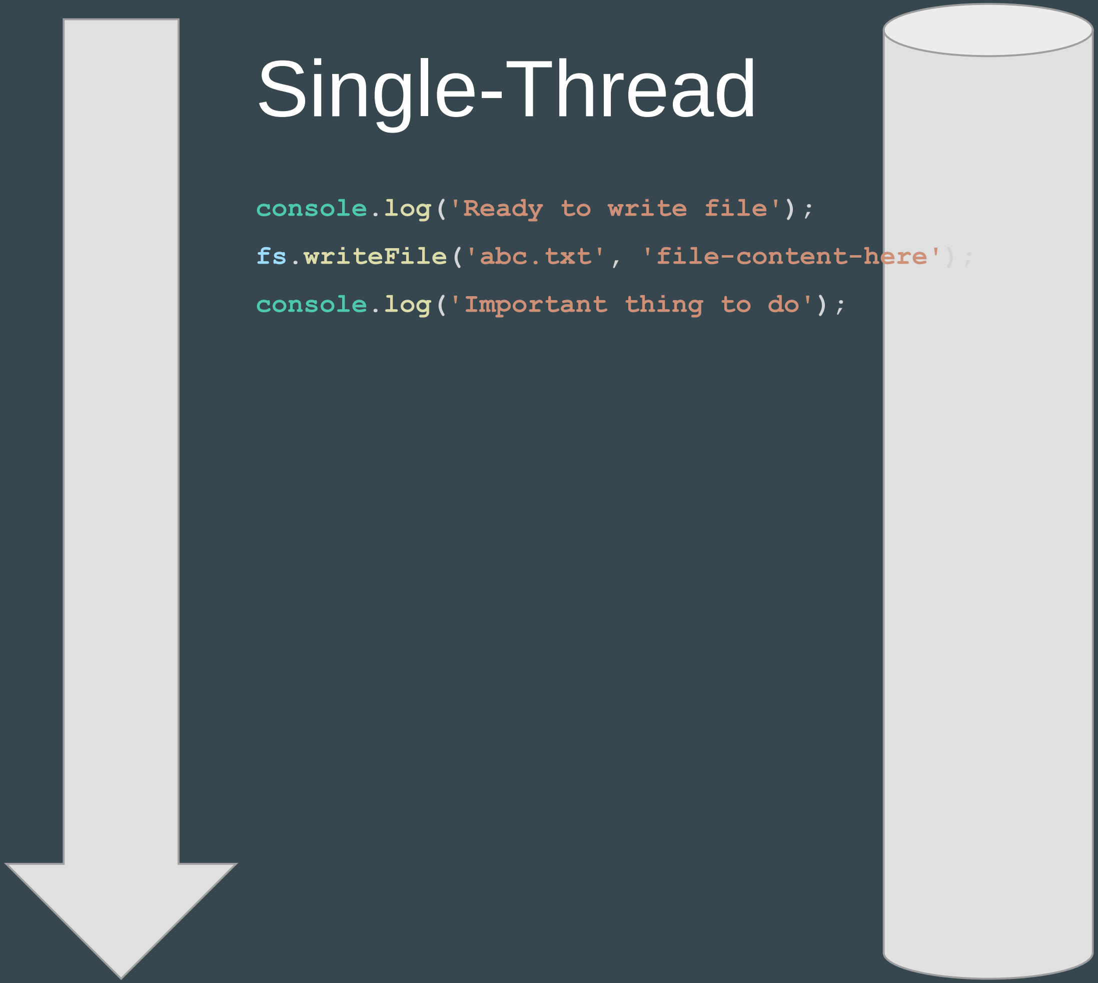
```
console.log('Ready to write file');  1ms
fs.writeFileSync('abc.txt', 'file-content-here');  33ms
console.log('Writing done');  1ms
```

World of JS

Single-Thread

```
console.log('Ready to write file');              1ms
fs.writeFileSync('abc.txt', 'file-content-here');    3300ms
console.log('Important thing to do');              1ms
```

World of Node.JS



Single-Thread

```
console.log('Ready to write file');
fs.writeFile('abc.txt', 'file-content-here');
console.log('Important thing to do');
```

Event Queue

```
Writing File in File System
```

Motivation

Problem: Node.js program only has one thread

```
console.log('Ready to write file');
fs.writeFile('abc.txt', 'file-content-here');
console.log('Writing done');
console.log(fs.readFile('abc.txt')); // Would you expect
`file-content-here`?
```

✗

Motivation

Problem: Node.js program only has one thread

Solution: Make most functions asynchronous (non-blocking)

```
console.log('Ready to write file');
fs.writeFile('abc.txt', 'file-content-here', function() {
 console.log('Writing done');
 console.log(fs.readFile('abc.txt')); // Would you expect
`file-content-here`?
});
```

Motivation

Problem: Node.js program only has one thread

Solution: Make most functions asynchronous (non-blocking)

Problem: Asynchronous functions cannot return result directly

```
console.log('Ready to write file');
fs.writeFile('abc.txt', 'file-content-here', function() {
 console.log('Writing done');
 console.log(fs.readFile('abc.txt')); // Would you expect
`file-content-here`?
});
```

✕

Motivation

Problem: Node.js program only has one thread

Solution: Make most functions asynchronous (non-blocking)

Problem: Asynchronous functions cannot return result directly

Solution: Use callbacks

```
console.log('Ready to write file');
fs.writeFile('abc.txt', 'file-content-here', function() {
 console.log('Writing done');
 fs.readFile('abc.txt', function(err, content) {
   console.log(content); // Finally...
 });
});
```

✓

Motivation

Problem: Node.js program only has one thread

Solution: Make most functions asynchronous (non-blocking)

Problem: Asynchronous functions cannot return result directly

Solution: Use callbacks

Problem: Callback hell

Solution: **Promise**

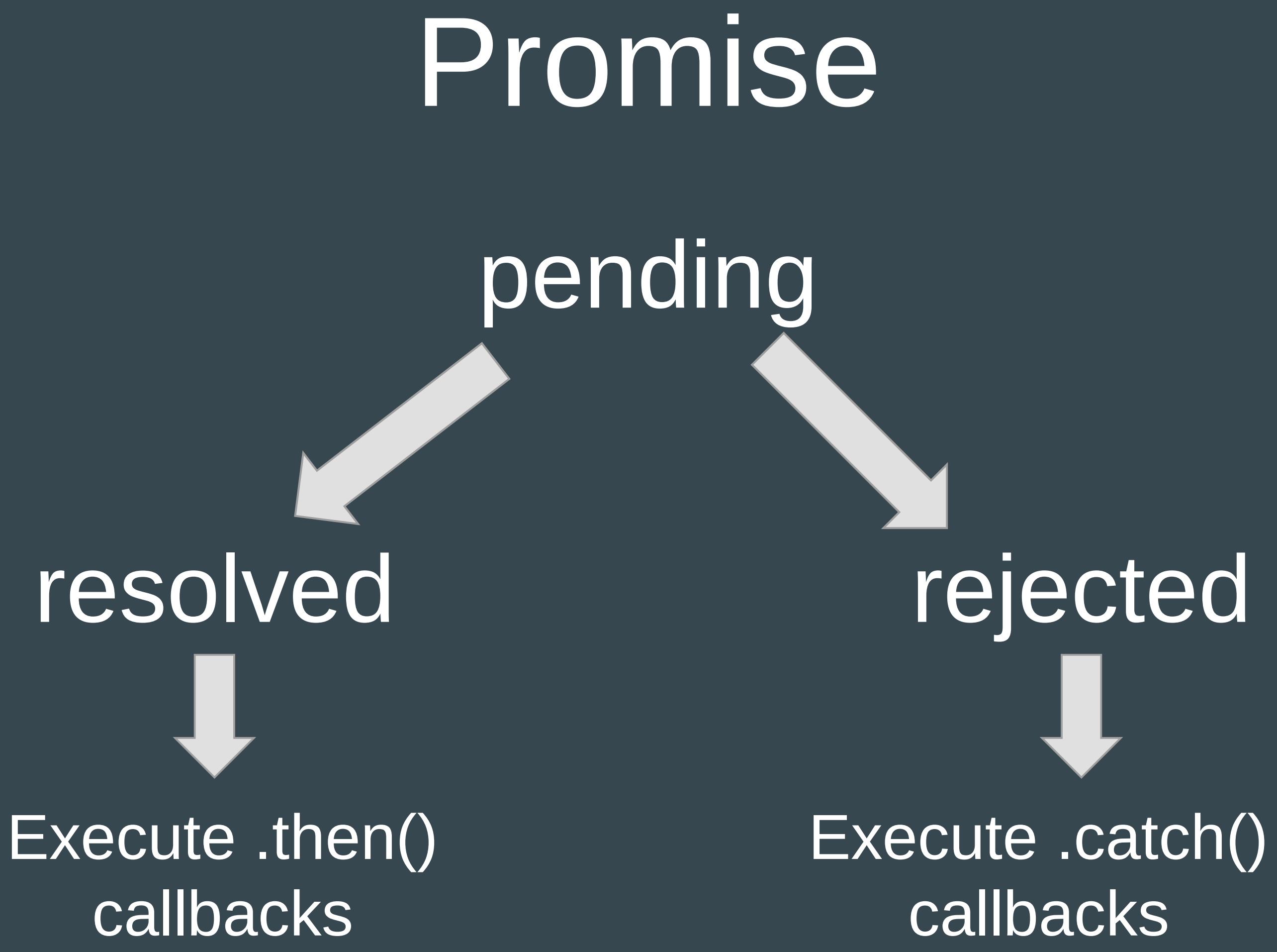
Prerequisite

- Distinguish blocking and non-blocking functions
- Use of callbacks

callback

```
fs.readFile('abc.txt', function(err, content) {
 console.log(content); // Finally...
});
```

Concept


Promise
pending
resolved
rejected
Execute .then() callbacks
Execute .catch() callbacks

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    nugget.eat()
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    nugget.eat()
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

new Promise(...)

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    nugget.eat()
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    nugget.eat()
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Promise

Example

```
axios.get("/users")
  .then(res => {
    console.log(res)
  })
  .catch(err => {
    console.log(err)
  });
```

Example

```
var promise = new Promise((resolve, reject)
=> {
    resolve();
});

promise.then(() => {
    console.log('Promise resolved.');
});

promise.catch(() => {
    console.log('An error occurred');
});
```

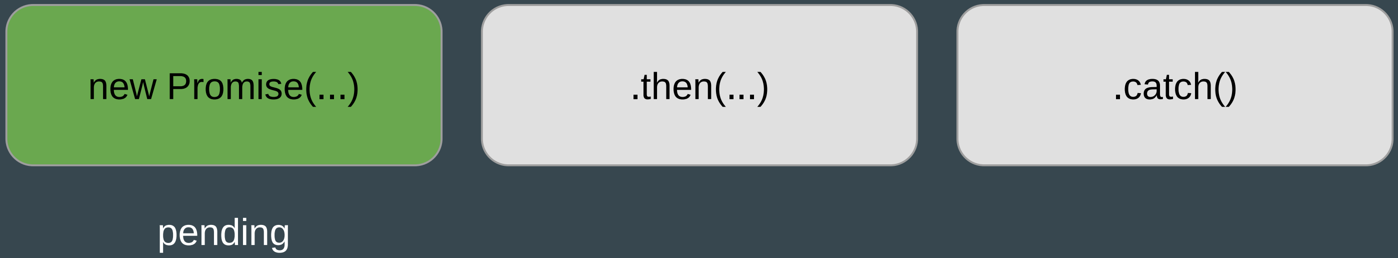
Concept

new Promise(...)

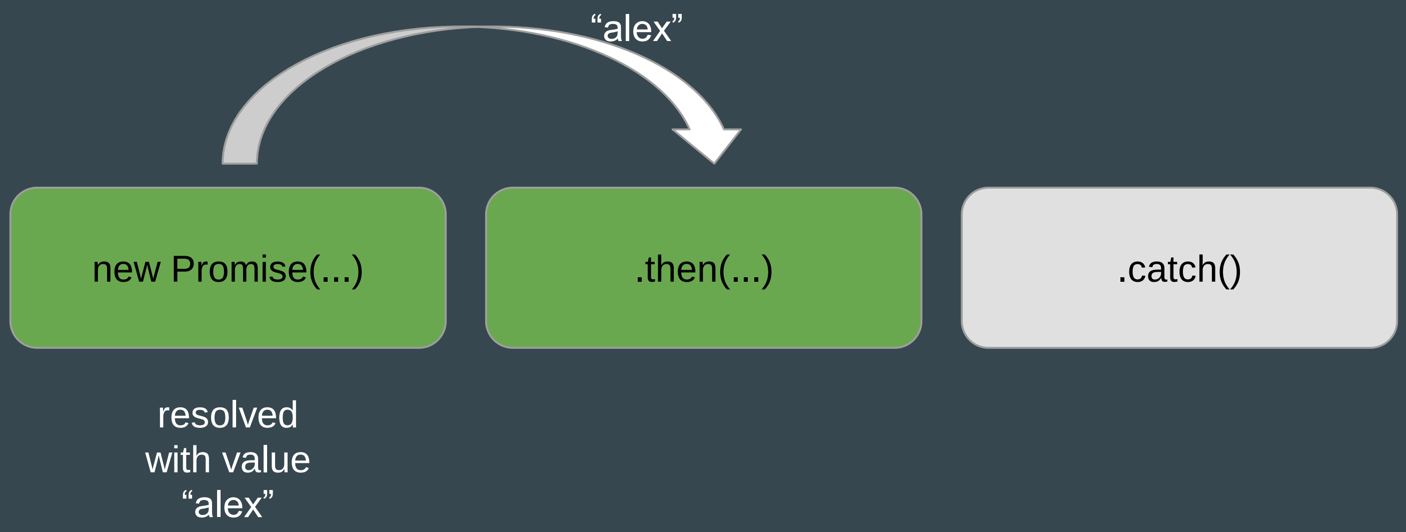
.then(...)

.catch()

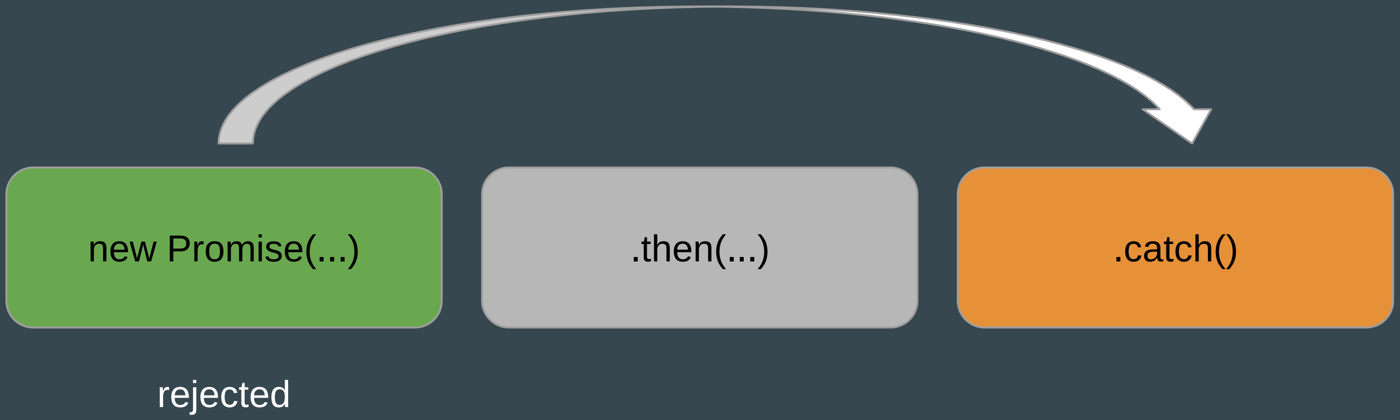
Concept



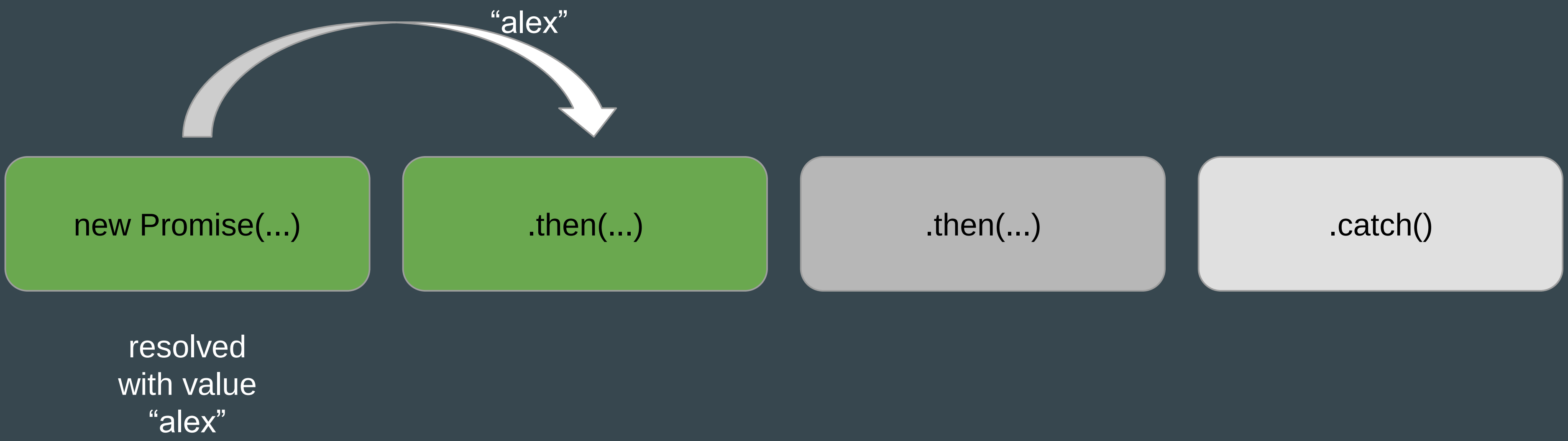
Concept (e1)



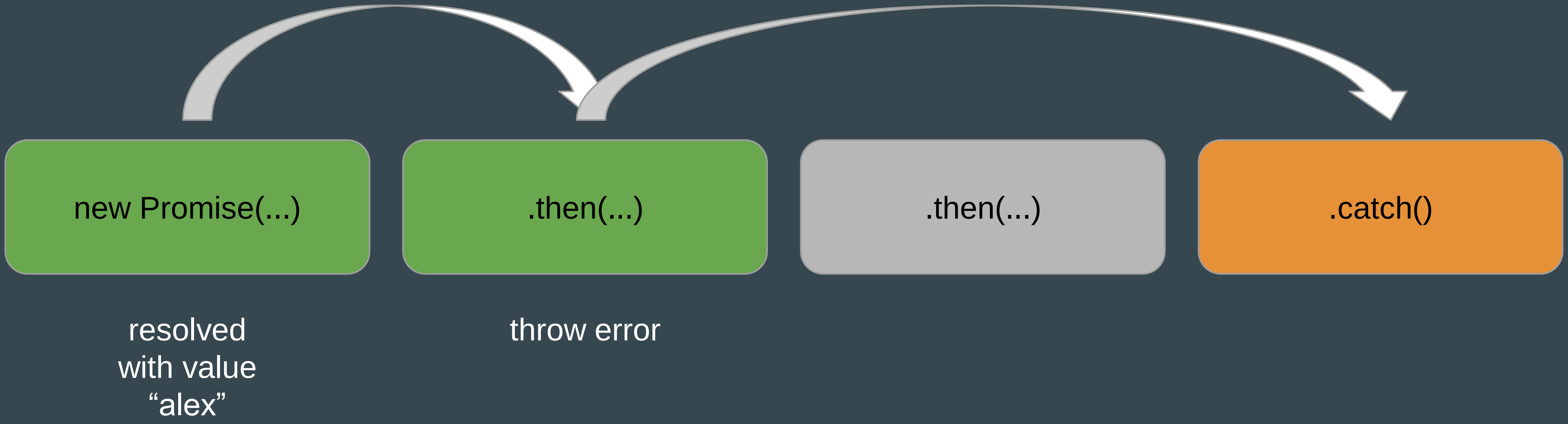
Concept (e2)



Concept



Concept (e3)


new Promise(...)
.then(...)
.then(...)
.catch()
resolved
with value
"alex"
throw error

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    nugget.eat()
    return "delicious"
  })
  .then(taste => {
    console.log('Alex said it is ' + taste);
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Example

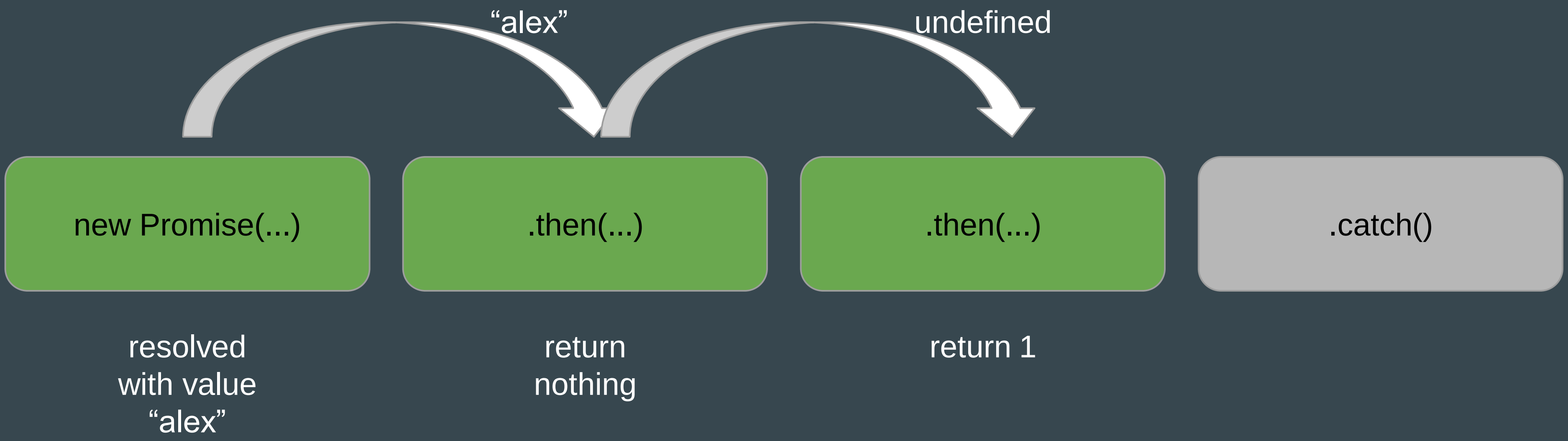
```
mcdonalds.get("mcnugget")
  .then(nugget => {
    nugget.eat()
    return "delicious"
  })
  .then(taste => {
    console.log('Alex said it is ' + taste);
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Example

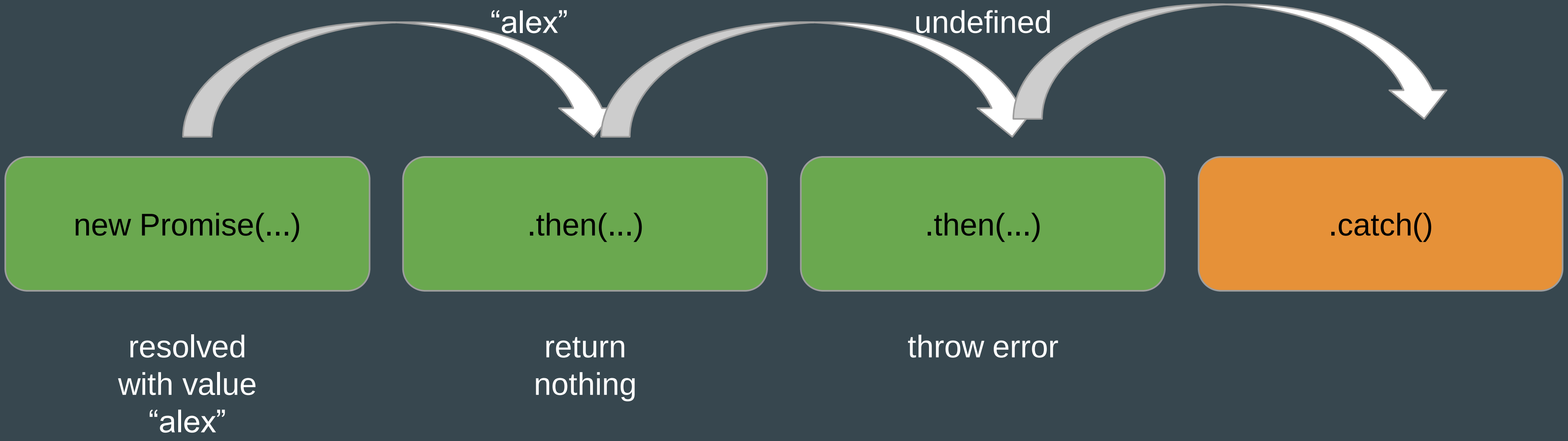
```
mcdonalds.get("mcnugget")
  .then(nugget => {
    nugget.eat()
    return "delicious"
  })
  .then(taste => {
    console.log('Alex said it is ' + taste);
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Run immediately

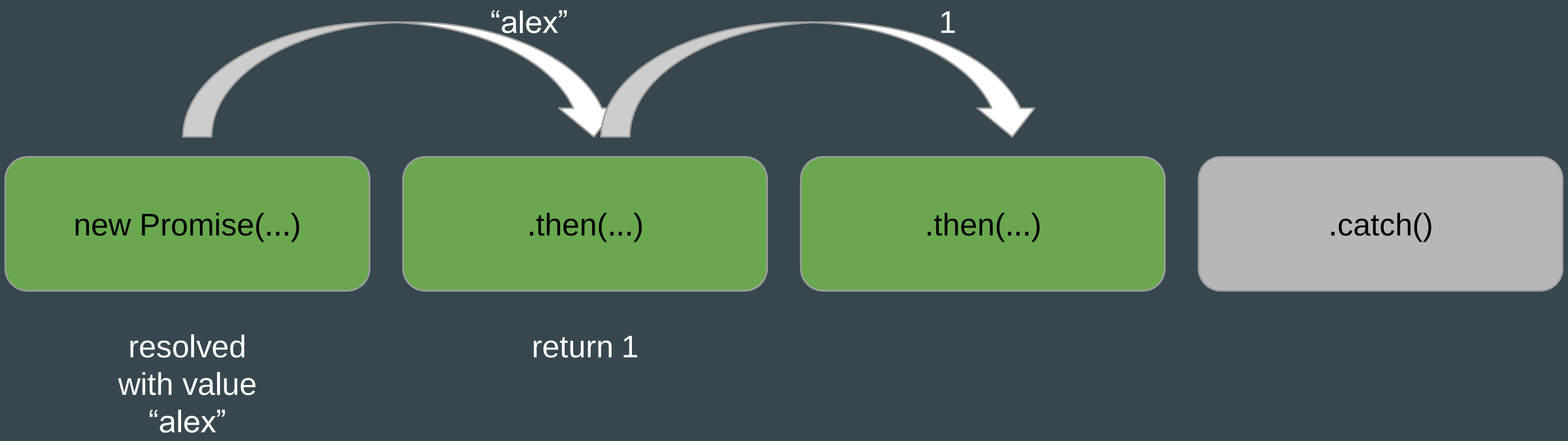
Concept


“alex”
undefined
new Promise(...)
.then(...)
.then(...)
.catch()
resolved with value “alex”
return nothing
return 1

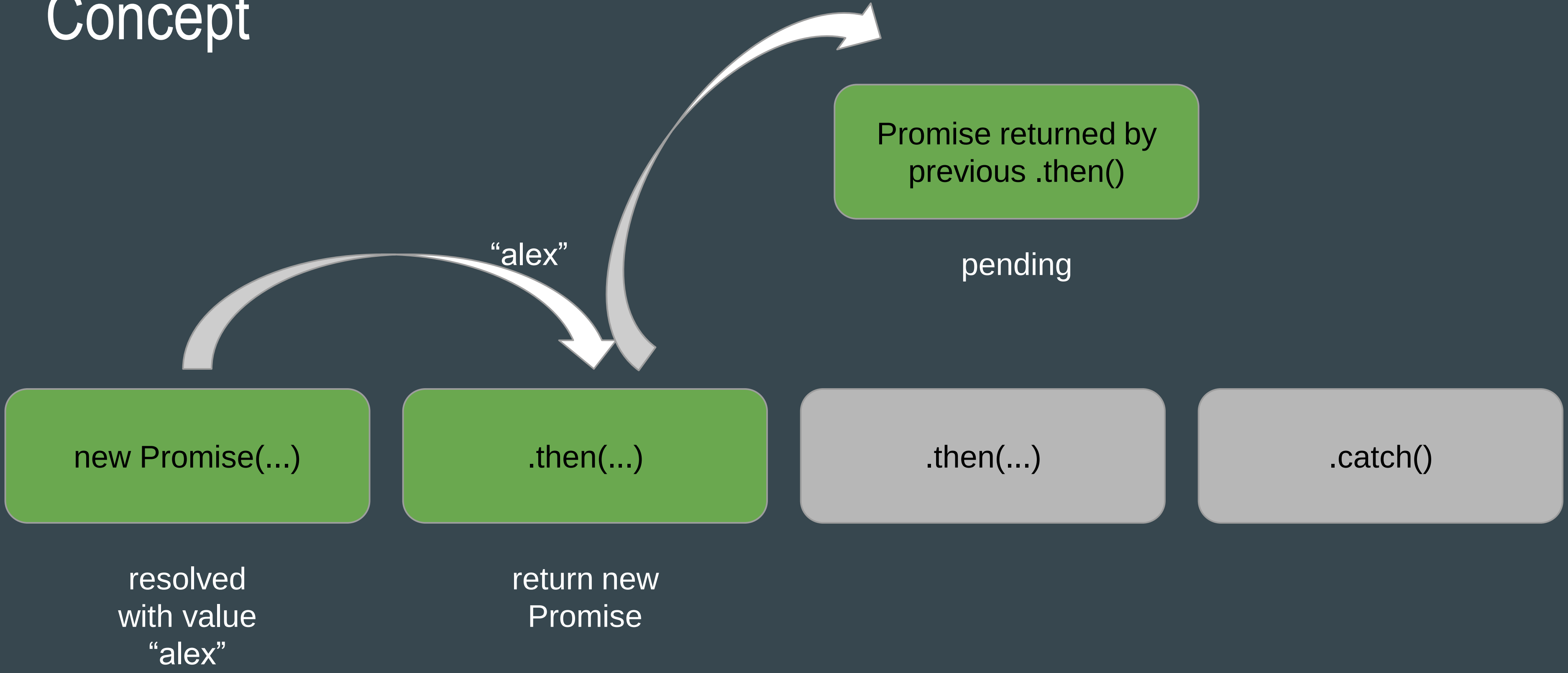
Concept (e5)


"alex"
undefined
new Promise(...)
.then(...)
.then(...)
.catch()
resolved with value "alex"
return nothing
throw error

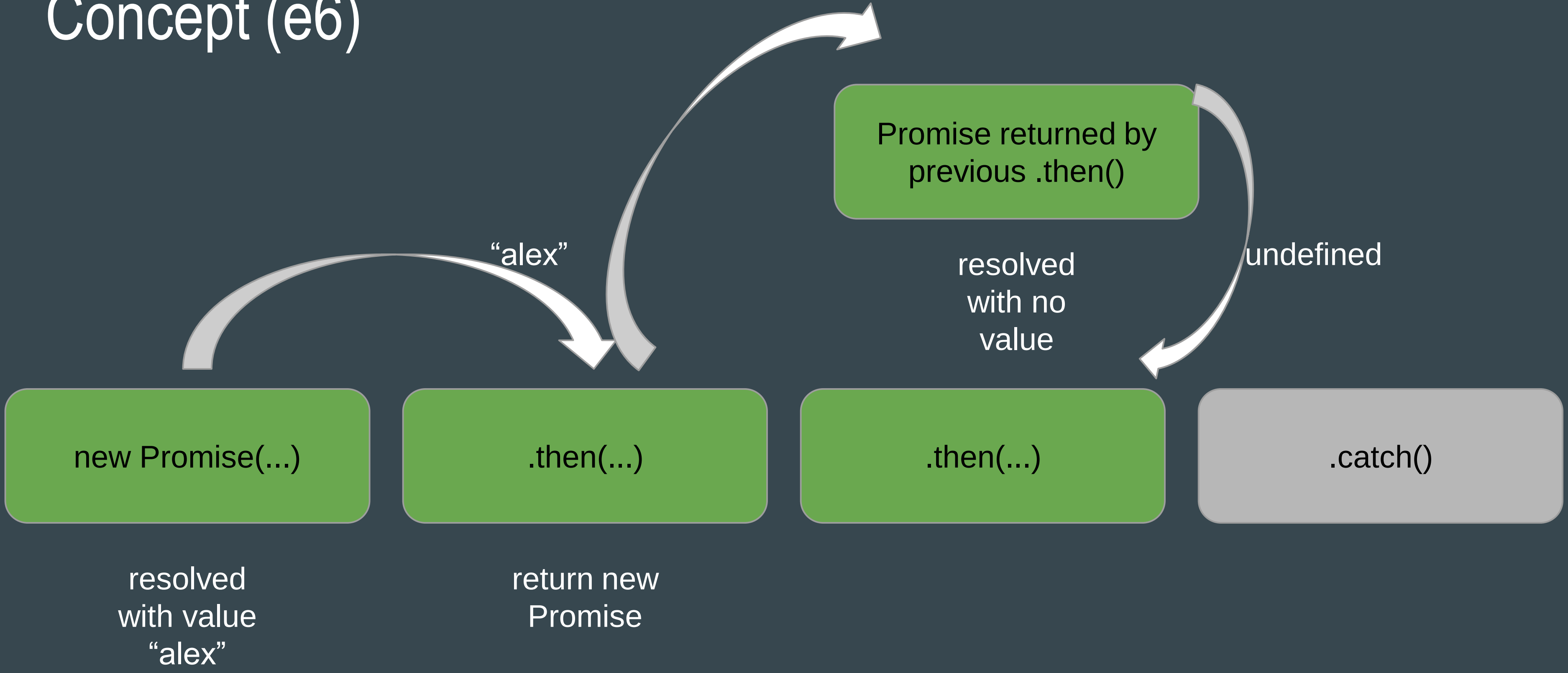
Concept


"alex"
1
new Promise(...)
.then(...)
.then(...)
.catch()
resolved with value "alex"
return 1

Concept


Promise returned by previous .then()
pending
"alex"
new Promise(...)
.then(...)
.then(...)
.catch()
resolved with value "alex"
return new Promise

Concept (e6)


Promise returned by previous .then()
"alex"
resolved with no value
undefined
new Promise(...)
.then(...)
.then(...)
.catch()
resolved with value "alex"
return new Promise

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    return nugget.eat()
  })
  .then(taste => {
    console.log('Alex said it is ' + taste);
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    return nugget.eat()
  })
  .then(taste => {
    console.log('Alex said it is ' + taste);
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Promise

Example

```
mcdonalds.get("mcnugget")
 .then(nugget => {
   return axios.get('/eat/nugget')
 })
 .then(taste => {
   console.log('Alex said it is ' + taste);
 })
 .catch(err => {
   alex.complain(mcdonalds.managers[0])
 });
```

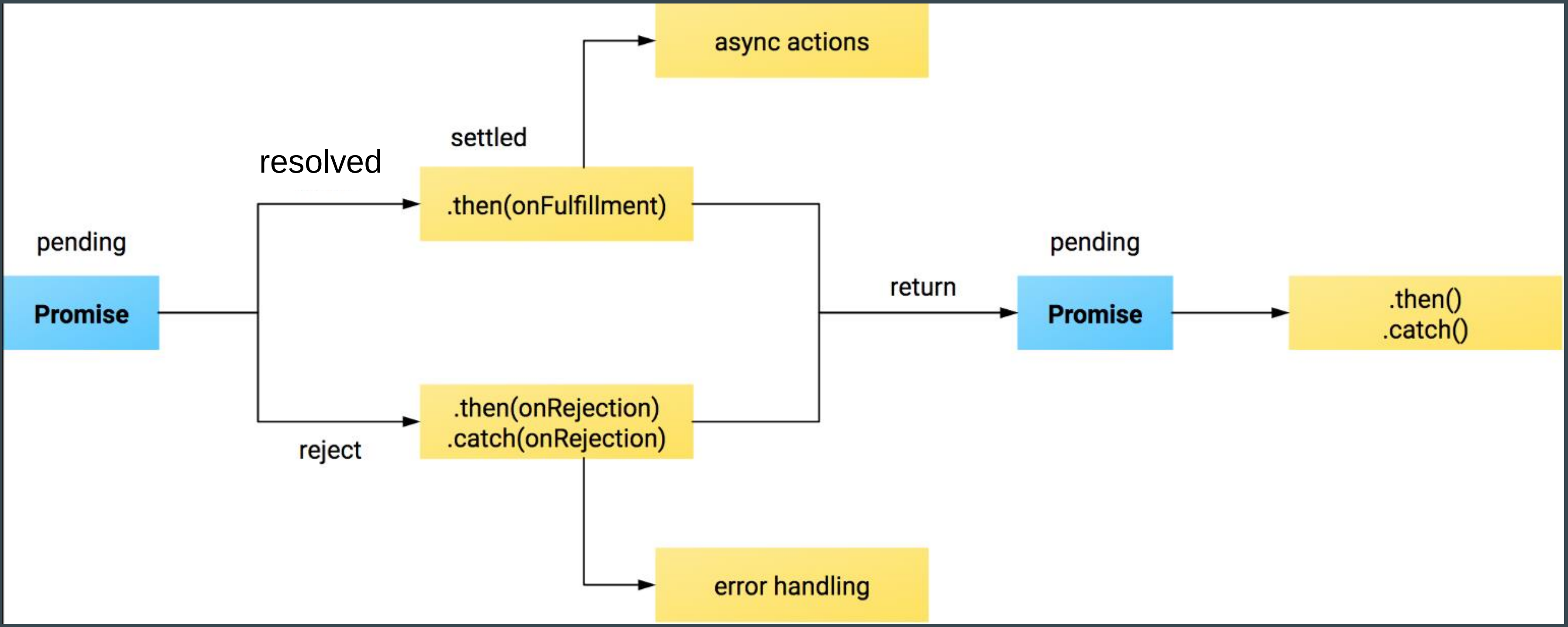
Promise

Example

```
mcdonalds.get("mcnugget")
  .then(nugget => {
    return nugget.eat()
  })
  .then(taste => {
    console.log('Alex said it is ' + taste);
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```

Only run after eating is done

Concept (Detailed)



Syntax

To create a promise:

```
const myFirstPromise = new Promise((resolve, reject) => {
  // do something asynchronous which eventually calls either:
  //
  //   resolve(someValue); // fulfilled
  // or
  //   reject("failure reason"); // rejected
});
```

Syntax

To use a promise:

```
let myFirstPromise = new Promise((resolve, reject) => {
  // We call resolve(...) when what we were doing asynchronously was successful, and
reject(...) when it failed.
  // In this example, we use setTimeout(...) to simulate async code.
  // In reality, you will probably be using something like XHR or an HTML5 API.
  setTimeout(function(){
    resolve("Success!"); // Yay! Everything went well!
  }, 250);
});

myFirstPromise.then((successMessage) => {
  // successMessage is whatever we passed in the resolve(...) function above.
  // It doesn't have to be a string, but if it is only a succeed message, it probably will
be.
  console.log("Yay! " + successMessage);
});
```

Even more complicated...

```
mcdonalds.queueUp()
  .then(() => {
    return mcdonalds.order("mcnugget")
  })
  .then(receipt => {
    return receipt.wait()
  })
  .catch(err => {
    mcdonalds.leave()
  })
  .then(nugget => {
    return nugget.eat().catch(() => {
      return 'bad';
    });
  })
  .then(taste => {
    if (taste === 'bad') {
      throw new Error('The food is horrible');
    }
  })
  .catch(err => {
    alex.complain(mcdonalds.managers[0])
  });
```