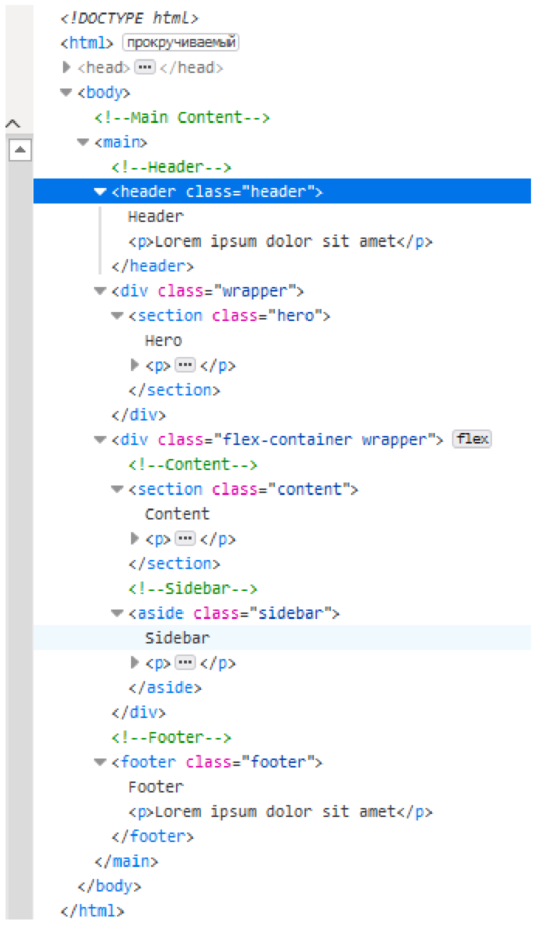
Кросс-платформенная разработка

Lecture 6

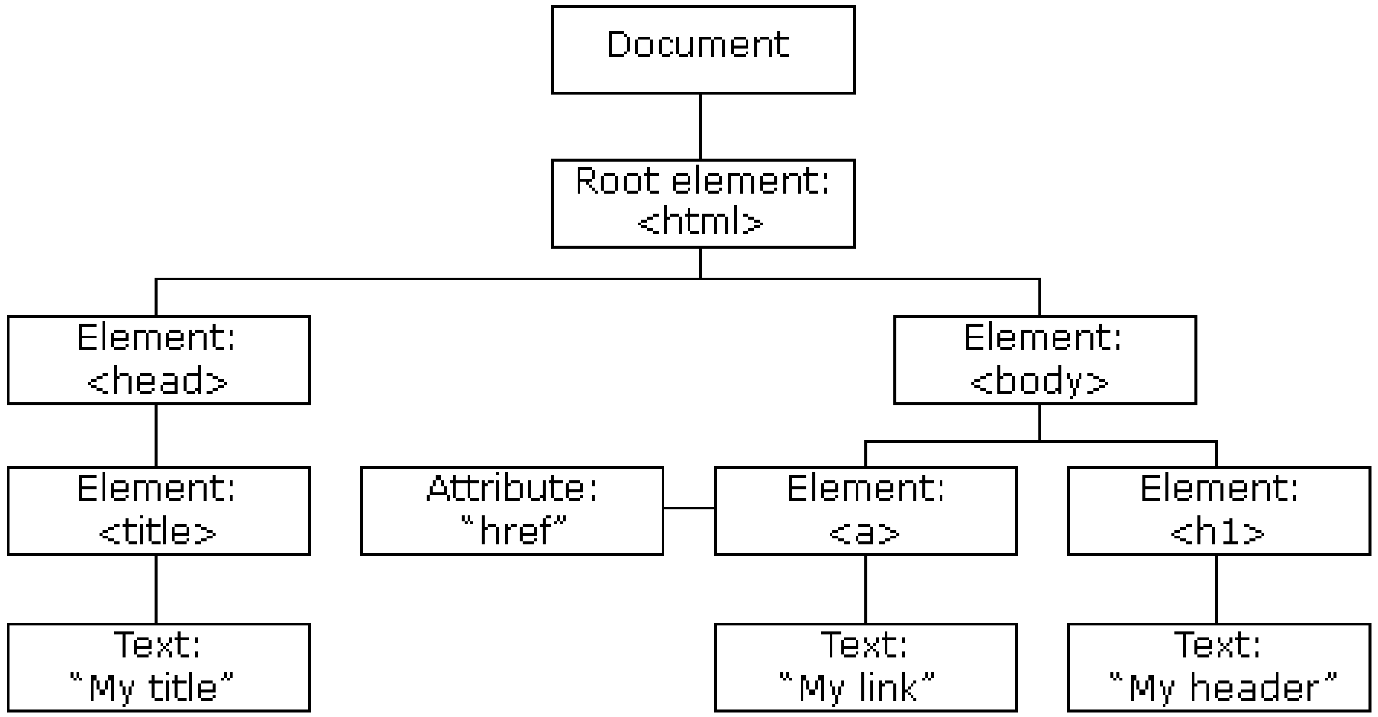
Topics

- DOM
- Modules
- Сборка
- Promise

DOM



```
<!DOCTYPE html>
<html> прокручиваемый
▶ <head> ··· </head>
▼ <body>
    <!--Main Content-->
  ▼ <main>
      <!--Header-->
    ▼ <header class="header">
        Header
        <p>Lorem ipsum dolor sit amet</p>
      </header>
    ▼ <div class="wrapper">
      ▼ <section class="hero">
          Hero
        ▶ <p> ··· </p>
        </section>
      </div>
    ▼ <div class="flex-container wrapper"> flex
        <!--Content-->
      ▼ <section class="content">
          Content
        ▶ <p> ··· </p>
        </section>
        <!--Sidebar-->
      ▼ <aside class="sidebar">
          Sidebar
        ▶ <p> ··· </p>
        </aside>
      </div>
      <!--Footer-->
    ▼ <footer class="footer">
        Footer
        <p>Lorem ipsum dolor sit amet</p>
      </footer>
    </main>
  </body>
</html>
```

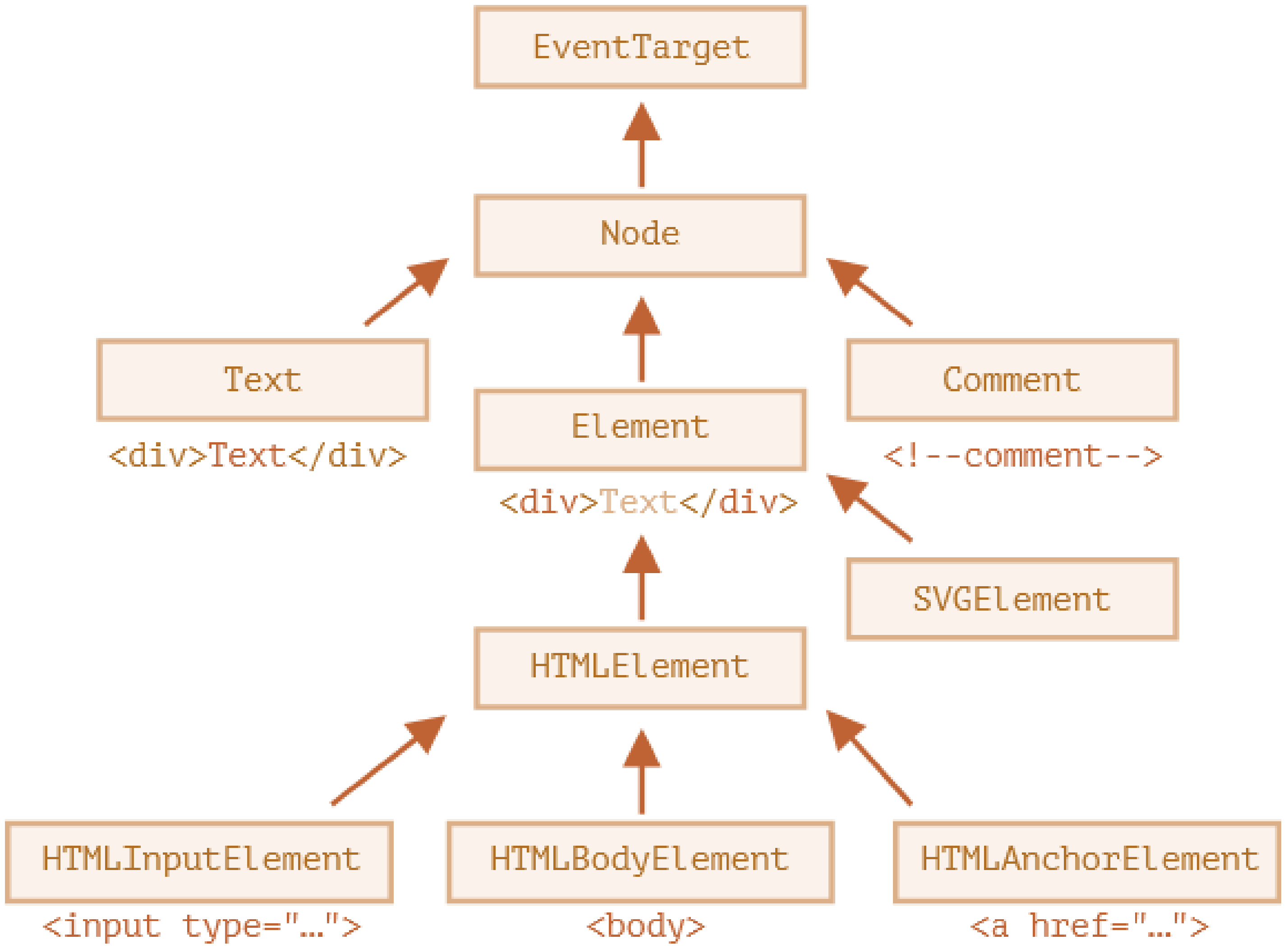


Accessing DOM with JS

```
const myElement = document.querySelector('#foo > div.bar’)

myElement.matches('div.bar') === true

const myElements = document.querySelectorAll('.bar’)

const myChildElemet = myElement.querySelector('input[type="submit"]')
```

The difference

```
const elements1 = document.querySelectorAll('div’)

const elements2 = document.getElementsByTagName('div’)

const newElement = document.createElement('div')

document.body.appendChild(newElement)

elements1.length === elements2.length // false
```


EventTarget
Node
Text
<div>Text</div>
Element
<div>Text</div>
Comment
<!--comment-->
SVGElement
HTMLElement
HTMLInputElement
<input type="…">
HTMLBodyElement
<body>
HTMLAnchorElement
<a href="…">

Element

myElement.children

myElement.firstElementChild

myElement.lastElementChild

myElement.previousElementSibling

myElement.nextElementSibling

Node

myElement.childNodes
myElement.firstChild
myElement.lastChild
myElement.previousSibling
myElement.nextSibling
myElement.parentNode
myElement.parentElement

Classes

```
myElement.classList.add('foo')
myElement.classList.remove('bar')
myElement.classList.toggle('baz')
```

Adding elements

```
// Append element1 as the last child of element2
element1.appendChild(element2)

// Insert element2 as child of element 1, right before element3
element1.insertBefore(element2, element3)
```

A lot of code and operations

- Manually accessing elements by selectors
- Editing DOM leads to re-renders
- Manual Data-View Syncs
- Shared Code for UI and Logic

Back to callback

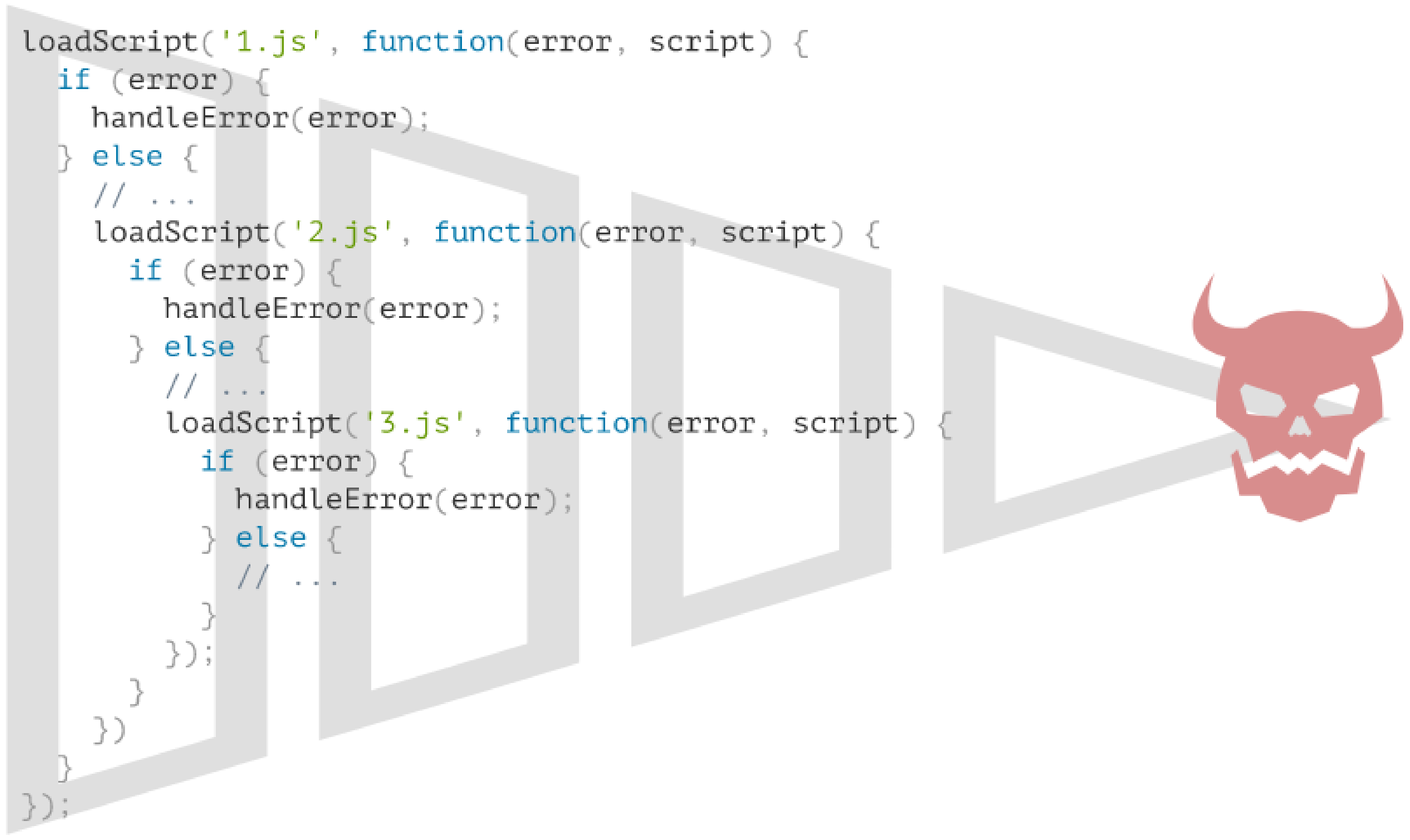
```
function loadScript(src) {
  let script = document.createElement('script');
  script.src = src;
  document.head.append(script);
}
```

```
loadScript('/my/script.js');

newFunction();
```

Now with callback

```
function loadScript(src, callback) {
    let script = document.createElement('script');
    script.src = src;
    script.onload = () => callback(script);
    document.head.append(script);
}

loadScript('https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js', script => {
    alert(`Cool, the script ${script.src} is loaded`);
    alert( _ );
});
```

Too many levels



```
loadScript('1.js', function(error, script) {
  if (error) {
    handleError(error);
  } else {
    // ...
    loadScript('2.js', function(error, script) {
      if (error) {
        handleError(error);
      } else {
        // ...
        loadScript('3.js', function(error, script) {
          if (error) {
            handleError(error);
          } else {
            // ...
          }
        });
      }
    })
  }
});
```

Modules

```
export function sayHi(user) {
  alert(`Hello, ${user}!`);
}
```

```
import {sayHi} from './sayHi.js';

alert(sayHi); // function...
sayHi('John'); // Hello, John!
```

Modules Loading

```
// 🗀 alert.js
alert("Module is evaluated!");

// Import the same module from different files

// 🗀 1.js
import `./alert.js`; // Module is evaluated!

// 🗀 2.js
import `./alert.js`; // (shows nothing)
```

Only via HTTP(S)

- No modules will be loaded from local files
- So, we need a server!

In-line modules

```
<script async type="module">
  import {counter} from './analytics.js';

  counter.count();
</script>
```

Node.js + NPM

- *Node.js*® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

- *Node package manager*, the *npm* Registry, and *npm* CLI

Why Node.js?

- We want to run some JS on server side
- We want to run it in the same way as in browser (Chrome)

Why Node.js?

- Use external packages
- Check for updates, check for vulnerabilities
- Select proper versions

Bundlers

- We need to “compile” js
- Large files are better for performance (for now?)
- We want some checks/minification/specific builds

Bundlers

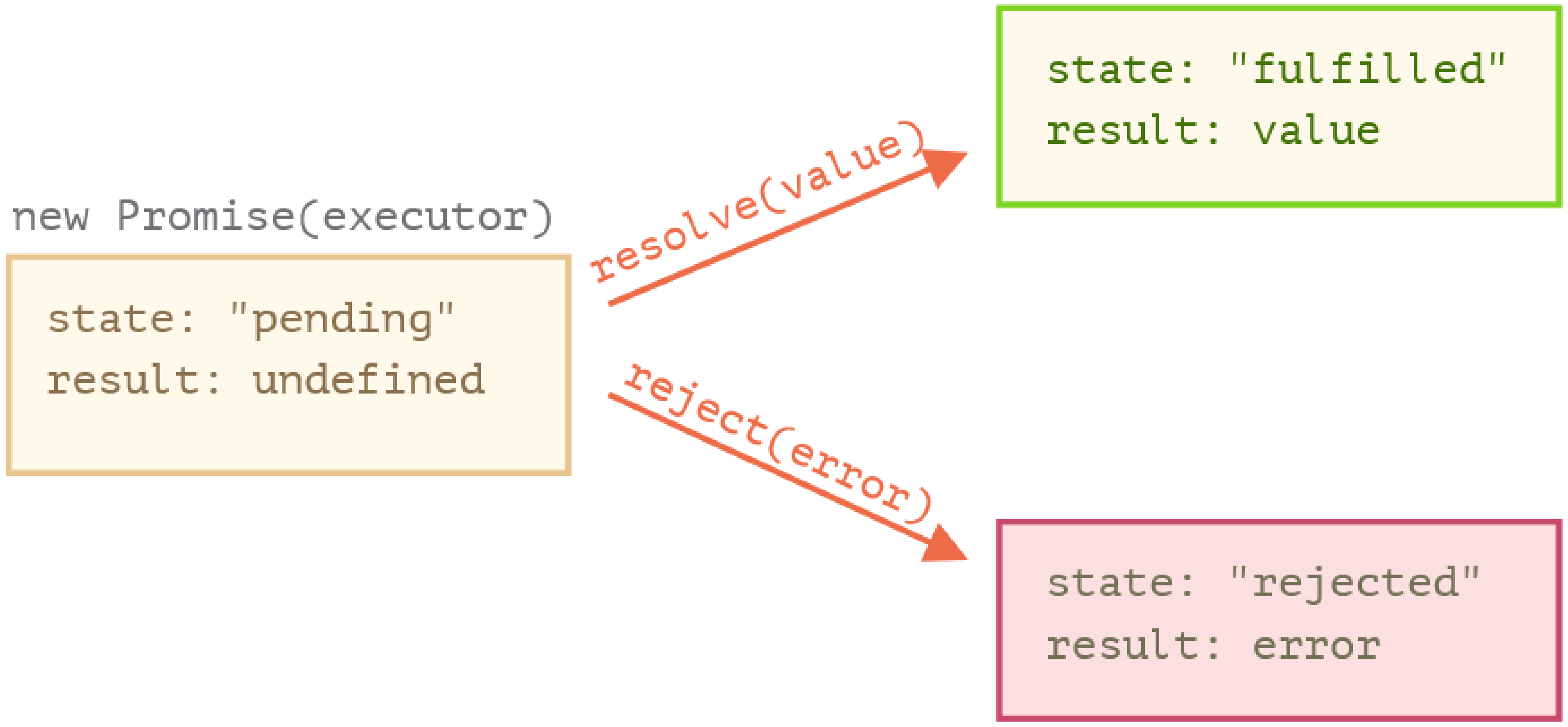
- Webpack
- Rollup
- Google Closure Compiler
- Parcel
- Browserify
- FuseBox

Let’s build an app

- Module(s)
- Tests
- Updating the page from JS

Promise

```
let promise = new Promise(function(resolve, reject) {
  // executor (the producing code)
});
```



Resolve/Reject

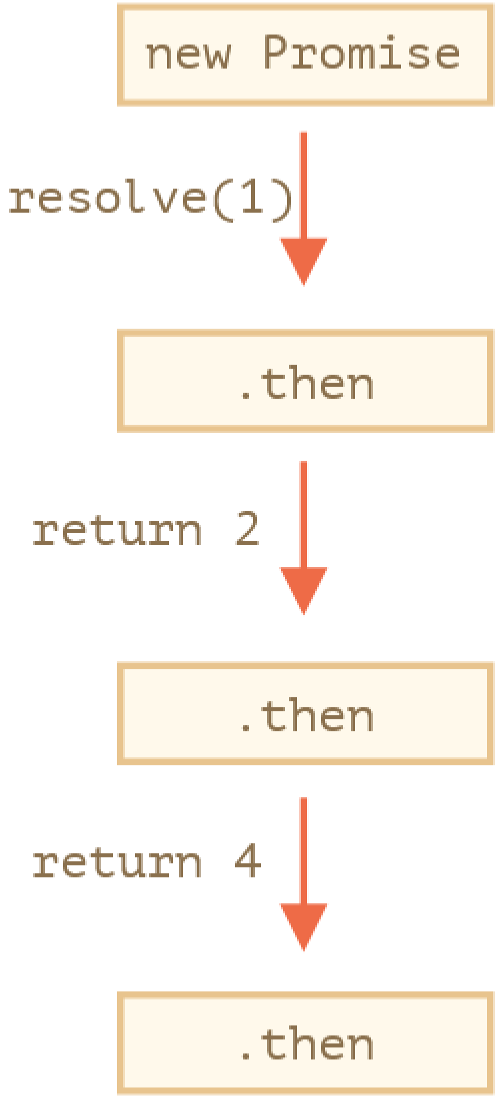
```
let promise = new Promise(function(resolve, reject) {
  // after 1 second signal that the job is finished with an error
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```

then/catch/finally

```
promise.then(
  function(result) { /* handle a successful result */ },
  function(error) { /* handle an error */ }
);

 promise.catch(alert);

 promise.finally(() => {{ /* general finalization*/ });
```

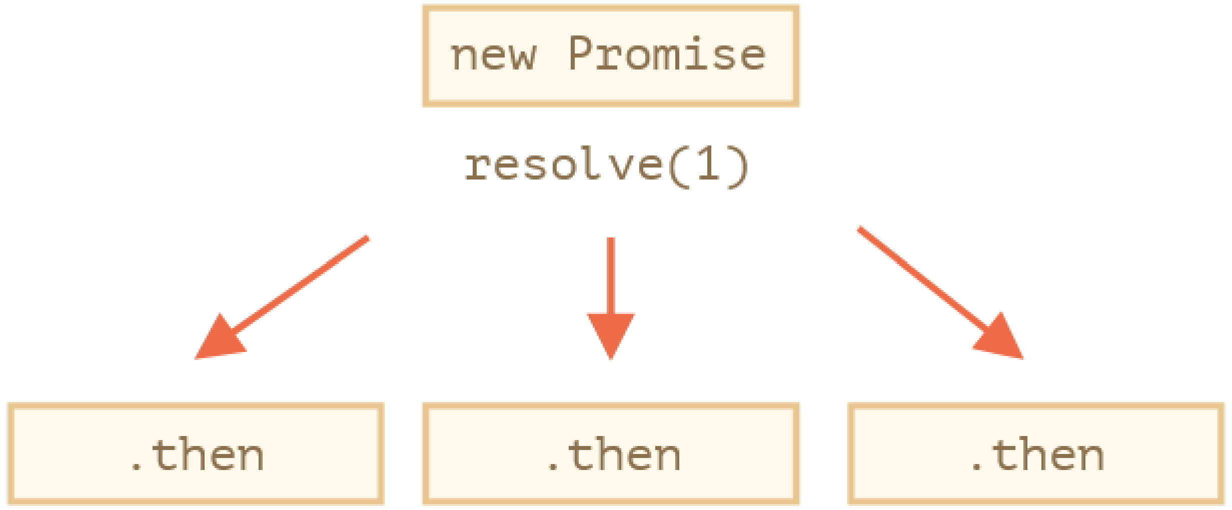
Chaining

```
new Promise(function(resolve, reject) {

  setTimeout(() => resolve(1), 1000); // (*)
}).then(function(result) { // (**)

  alert(result); // 1
  return result * 2;
}).then(function(result) { // (***)

  alert(result); // 2
  return result * 2;
});
```



Check the difference

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => resolve(1), 1000);
});

promise.then(function(result) {
  alert(result); // 1
  return result * 2;
});
promise.then(function(result) {
  alert(result); // 1
  return result * 2;
});
```



Duck Typing

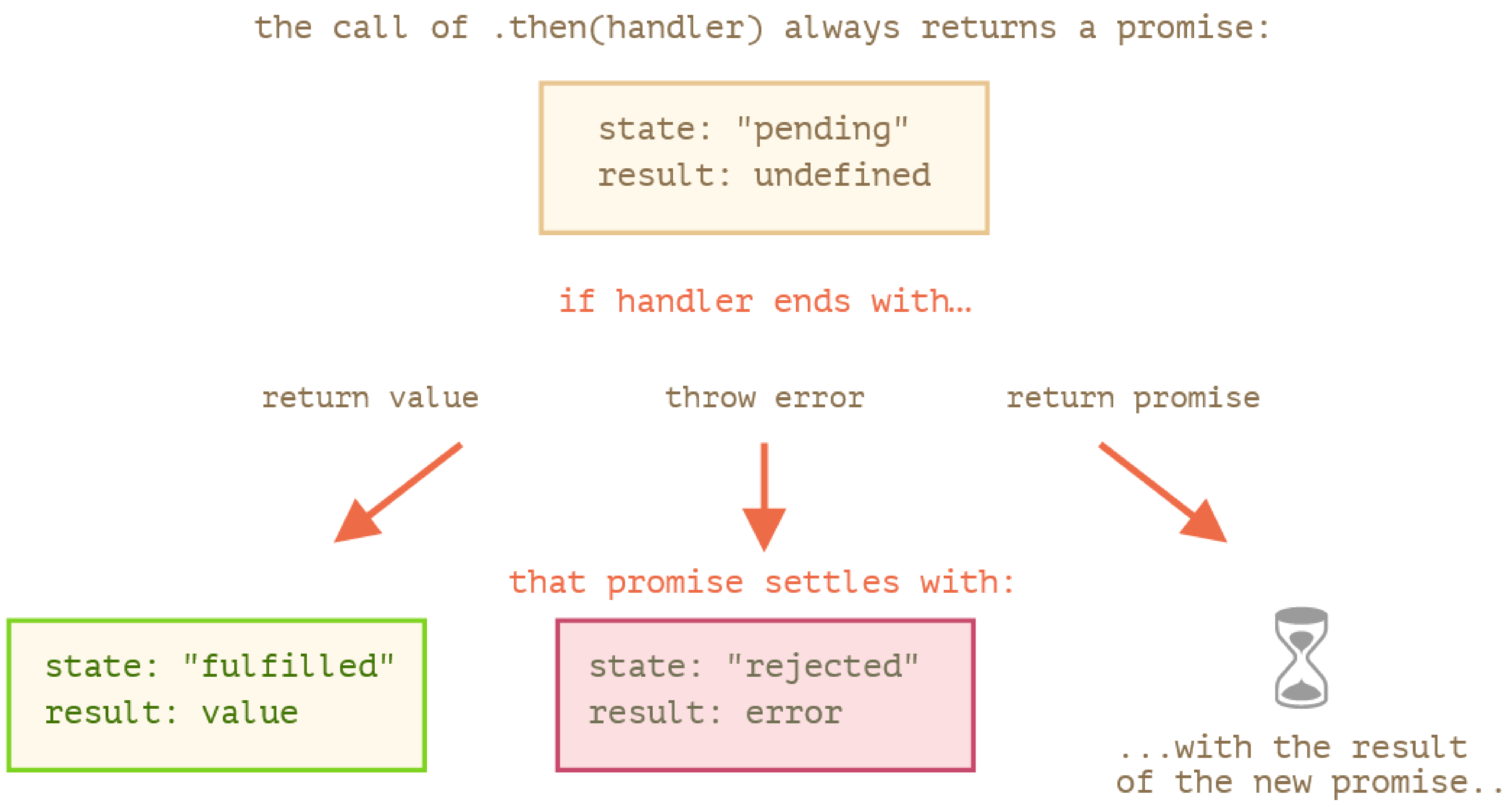
```
class Thenable {
  constructor(num) {
    this.num = num;
  }
  then(resolve, reject) {
    alert(resolve); // function() { native code }
    // resolve with this.num*2 after the 1 second
    setTimeout(() => resolve(this.num * 2), 1000); // (**)
  }
}
```

fetch

```
fetch('/article/promise-chaining/user.json')
  // .then below runs when the remote server responds
  .then(function(response) {
    // response.text() returns a new promise that resolves with the full response text
    // when it loads
    return response.text();
  })
  .then(function(text) {
    // ...and here's the content of the remote file
    alert(text);
});
```

=>

```
fetch('/article/promise-chaining/user.json')
    .then(response => response.json())
    .then(user => alert(user.name));
```


the call of .then(handler) always returns a promise:
state: "pending"
result: undefined
if handler ends with…
return value
throw error
return promise
that promise settles with:
state: "fulfilled"
result: value
state: "rejected"
result: error
...with the result
of the new promise..

References

- https://javascript.info/
- https://developer.mozilla.org/
- https://www.w3schools.com