

# Frontend Oops

Frontend devs used to just need HTML, CSS, and JS! Now apparently they need to learn about node, npm, grunt, gulp, webpack, babel...wait up...

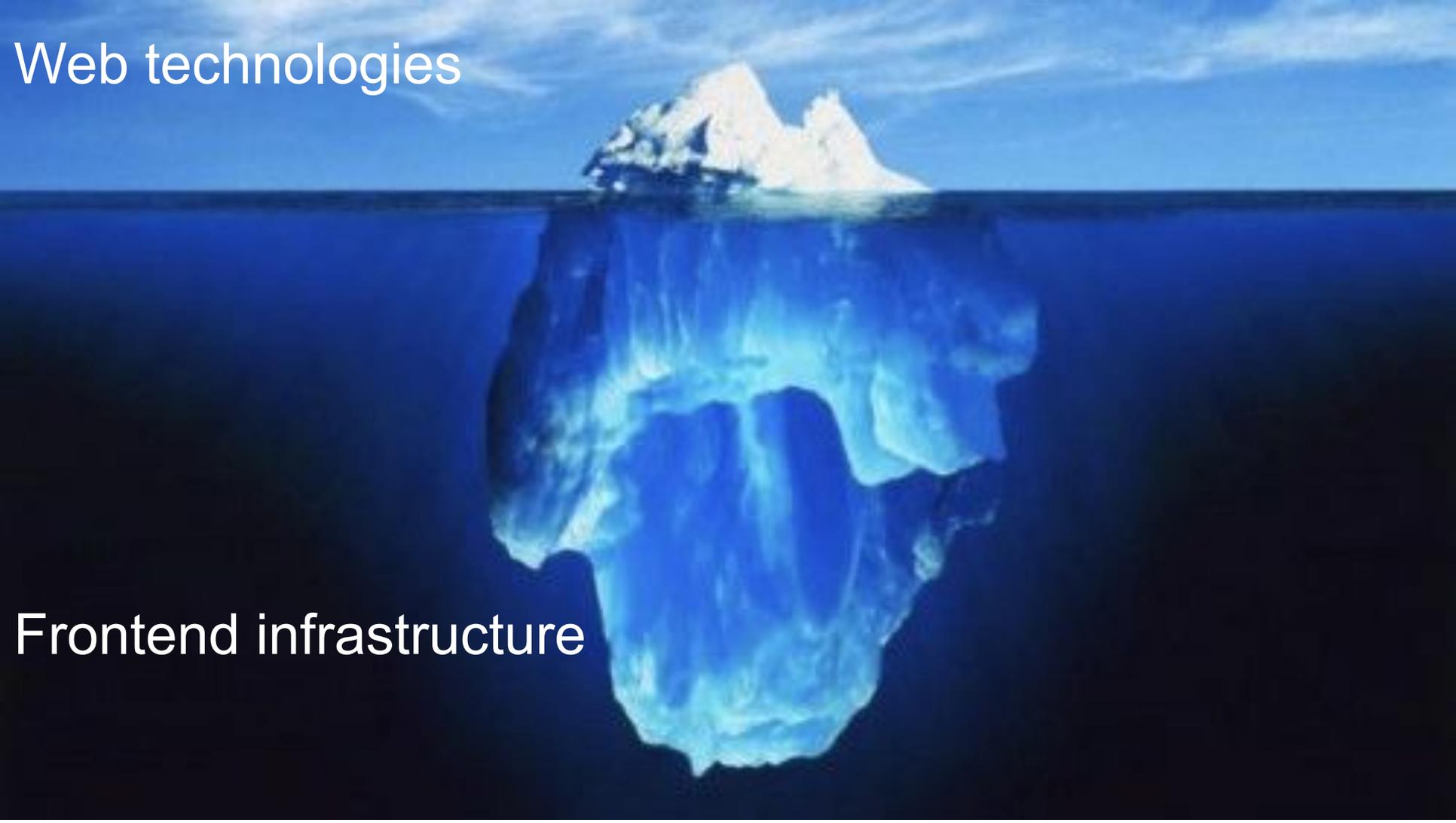


what the heck do I need node for on the frontend?



Web technologies

Frontend infrastructure





Немного философии



# Модули в программировании

**import**

*// code*

**export**

Reusability (переиспользуемость)

Composability (компоновка)

Leverage (делегирование)

Isolation (инкапсуляция)

Leverage

# Олды помнят

---



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="https://cdn/dependency1.js"></script>
  <script src="https://cdn/dependency2.js"></script>
  <script src="main.js"></script>
</head>
<body>
<h1>Hello from HTML!</h1>
</body>
</html>
```

# Олды помнят

---



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="/vendors/dependency1.js"></script>
  <script src="/vendors/dependency2.js"></script>
  <script src="main.js"></script>
</head>
<body>
<h1>Hello from HTML!</h1>
</body>
</html>
```

Packages!

# Package managers



Bower



NPM

# Пакеты? Зачем?

- Не нужно изобретать велосипед. Если вам что-то понадобилось, стоит изучить, возможно другие программисты уже решили вашу задачу. В реестре NPM находится крупнейшая в мире коллекция пакетов (точнее, более 1 000 000). Скорее всего, если вам нужен конкретный пакет, он есть у NPM.
- Возможно шарить собственные библиотеки между своими проектами.

Используем пакеты в  
браузере!

# Попробуем так:

---

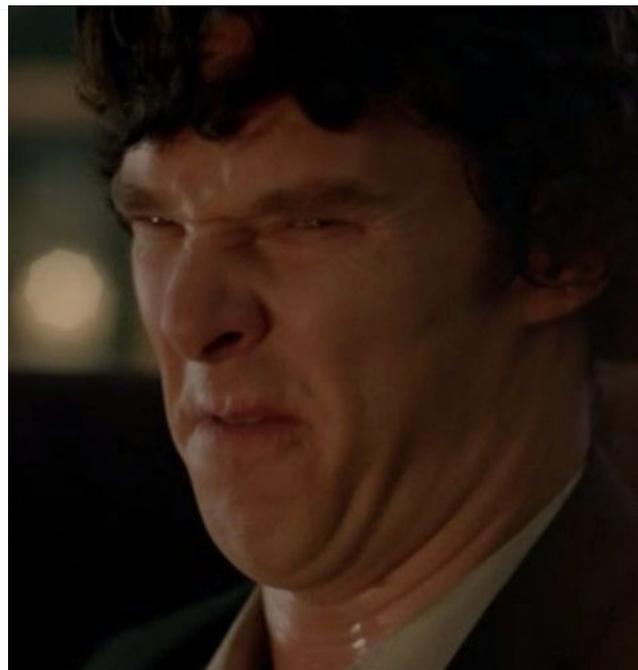


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="/node_modules/dependency1.js"></script>
  <script src="/node_modules/dependency2.js"></script>
  <script src="main.js"></script>
</head>
<body>
<h1>Hello from HTML!</h1>
</body>
</html>
```

# Ну такое...



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="/node_modules/dependency1.js"></script>
  <script src="/node_modules/dependency2.js"></script>
  <script src="main.js"></script>
</head>
<body>
<h1>Hello from HTML!</h1>
</body>
</html>
```



# CommonJs modules

## Простой пример:

```
// module.js
```

```
const one = 1;
```

```
exports.two = 2;
```

```
module.exports = { one };
```

```
// othermodule.js
```

```
const m = require('module');
```

```
console.log(m.one);
```

```
console.log(m.two); // ошибка
```

## Пара НО:

- Браузеры ничего про не знают ни про какой `require`
- Все вызовы `require` — синхронные

# Module Loaders

# Async Module Loader and AMD — requireJS

AMD *Asynchronous Module Definition* — встречается в большом количестве "взрослых" проектов



```
define('mail/User', [  
    'RPCModel',  
    'mail/Phone',  
    'mail/User/TOTPCode'  
], function (  
    RPCModel,  
    Phone,  
    TOTPCode  
) {  
    const User = '...'; // объявляем модуль, используем зависимости  
  
    return User;  
})
```



```
// где-то в коде приложения  
require(['mail/User', 'otherModule'], function(User, otherModule) {  
    // some stuff  
});
```

# Пример

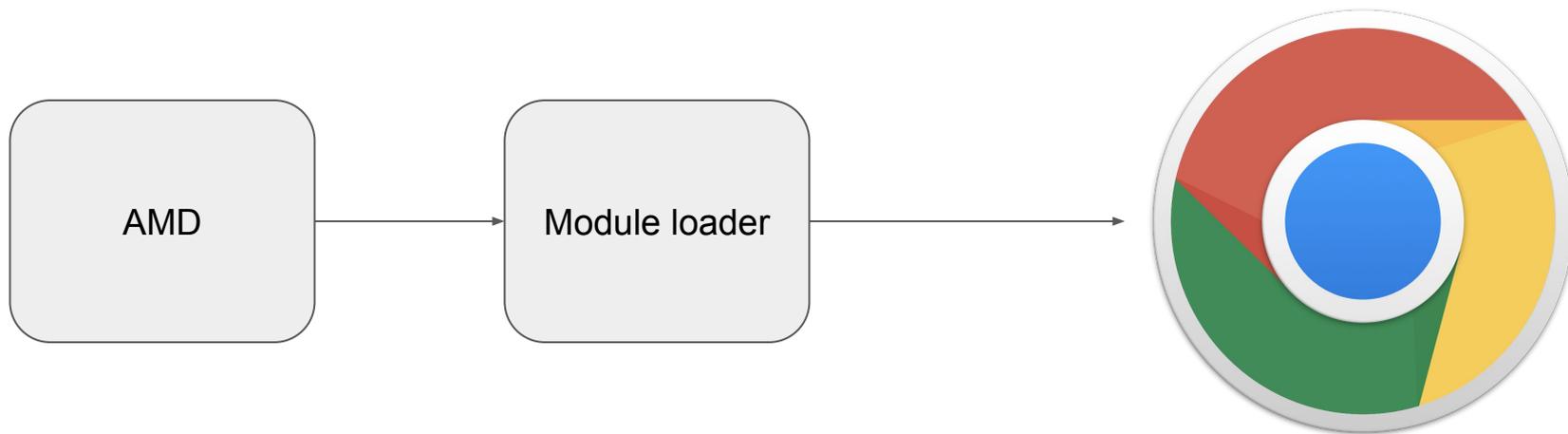


```
define('mail/User', [  
  'RPCModel',  
  'mail/Phone',  
  'mail/User/TOTPCode'  
], function (  
  RPCModel,  
  Phone,  
  TOTPCode  
) {  
  const User = '...'; // объявляем модуль, используем зависимости  
  
  return User;  
})
```



```
// где-то в коде приложения  
require(['mail/User', 'otherModule'], function(User, otherModule) {  
  // some stuff  
});
```

# Схема работы



А если я хочу  
переиспользовать код на  
сервере и в браузере?

У нас есть проблема с разрозненными модулями?

Сделаем ещё одну систему модулей!

UMD



# ES6 Modules

Directed by  
ROBERT B. WEIDE

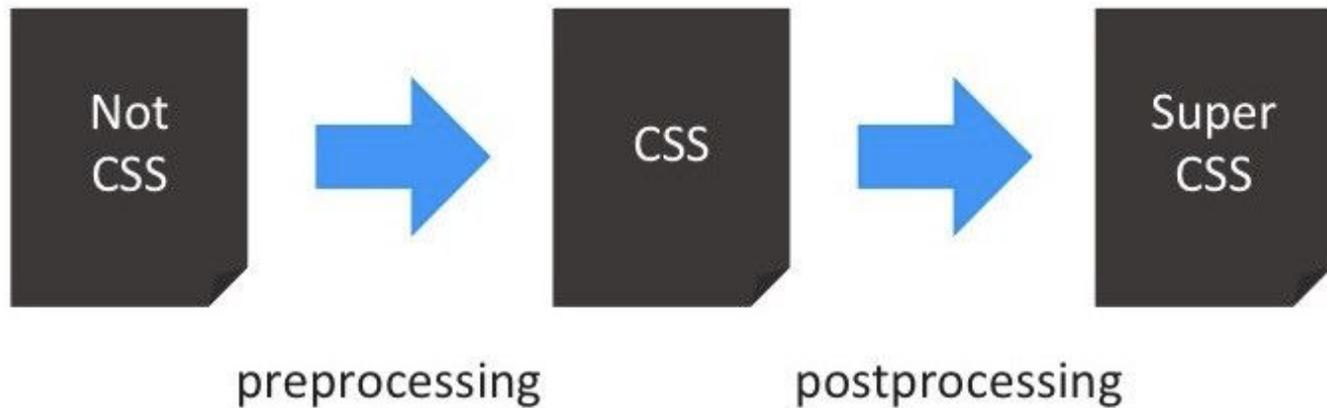
# Asset managers

Это что ещё за твари?



# Task runners

# Вспомним прошлую лекцию



# Сборка — это pipeline

*И через этот pipeline пропускаются все наши исходники пока из них не получится то, что браузер сможет “скушать”*

# Bundlers

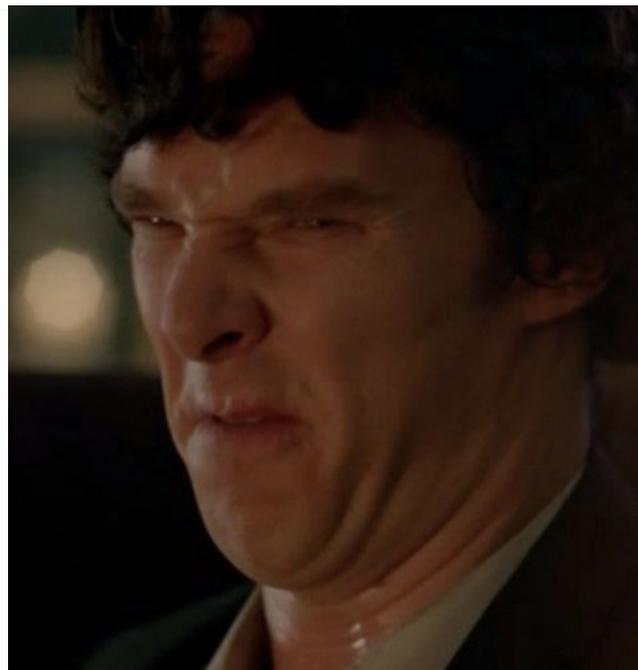
# А это ещё что такое?

This means it takes modules with dependencies and emits static assets representing those modules.

# Ну такое...



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="/node_modules/dependency1.js"></script>
  <script src="/node_modules/dependency2.js"></script>
  <script src="main.js"></script>
</head>
<body>
<h1>Hello from HTML!</h1>
</body>
</html>
```



# А вот это интересно...

---



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="my app with all dependencies.js"></script>
</head>
<body>
<h1>Hello from HTML!</h1>
</body>
</html>
```

# Bundle splitting

# Vendors

- vendors.js
- app.js

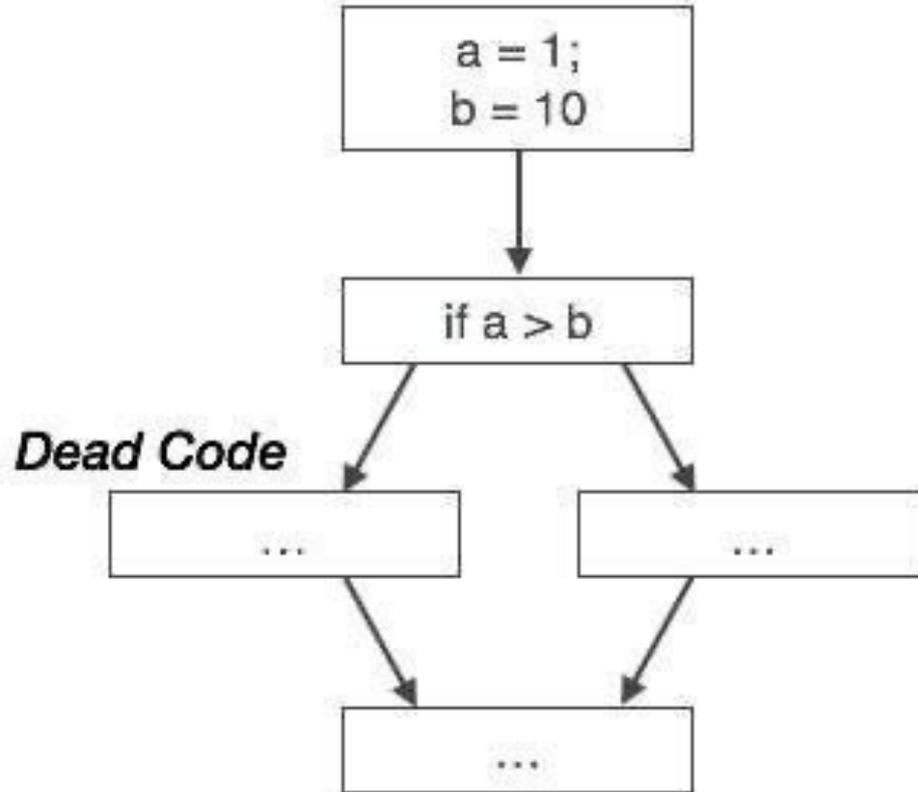
# chunk splitting

- vendors.js
- core.js
  - signup.js
  - login.js
  - ...js

# Tree Shaking



# Dead code elimination



Это что ещё за твари?



# npm scripts

# npm scripts

---



```
// package.json
{
  "scripts": {
    "start": "devServer start",
    "build": "webpack ...",
    "test": "npm run lint && ...",
    "deploy": "..."
  }
}
```

Окей, новый  
Frontend супер, Js  
вообще — 



Microsoft®

**Internet  
Explorer® 6**

Что делать?

# Полифиллы

# Полифиллы

*Полифилл* — это библиотека, которая добавляет в старые браузеры поддержку возможностей, которые в современных браузерах являются встроенными

```
if (!Object.is) {  
  
    Object.is = function(x, y) {  
  
        if (x === y) { return x !== 0 || 1 / x === 1 / y; }  
  
        else { return x !== x && y !== y; }  
  
    }  
}
```

# Транспайлинг

# Транспайлинг

*Транспайлинг — это конвертация кода программы, написанной на одном языке программирования в другой язык программирования*

# Babel ([babeljs.io](https://babeljs.io))

**Babel** – многофункциональный транспайлер, позволяет транспиллировать *ES5, ES6, ES2016, ES2017, ES2018, ES2019, ES2020, ES.Next, JSX* и *Flow*

[Babel · The compiler for next generation JavaScript](https://babeljs.io)

# Как работает Vabel?

- Парсит исходный код и строит AST
- Последовательно вызывает набор функций, которые каким-то образом трансформируют AST программы
- В процессе трансформации части AST, относящиеся к современному синтаксису, заменяются на эквивалентные, но более общеупотребительные фрагменты
- Преобразует модифицированное AST в новый транспилированный код

Что не затронули?

# Что не затронули?

- Unit тесты <https://jestjs.io/ru/>
- prettier и power linting <https://prettier.io/> + хорошая статья про то как писать код аккуратно <https://habr.com/ru/company/ruvds/blog/428173/>
- i18n (след. слайд)
- Hot-reloading как сделать в webpack <https://webpack.js.org/concepts/hot-module-replacement/>

# Про i18n (всё что есть про i18n в Web на 2020)

"Трудности перевода или i18n в крупных проектах"

*доклад с митапа*

*SmartMail Meetup: Frontend*

Всем спасибо!