

---

---

## Lecture 3

# JavaScript Ecosystem

— Frontend Web Development —

---

---

Recap

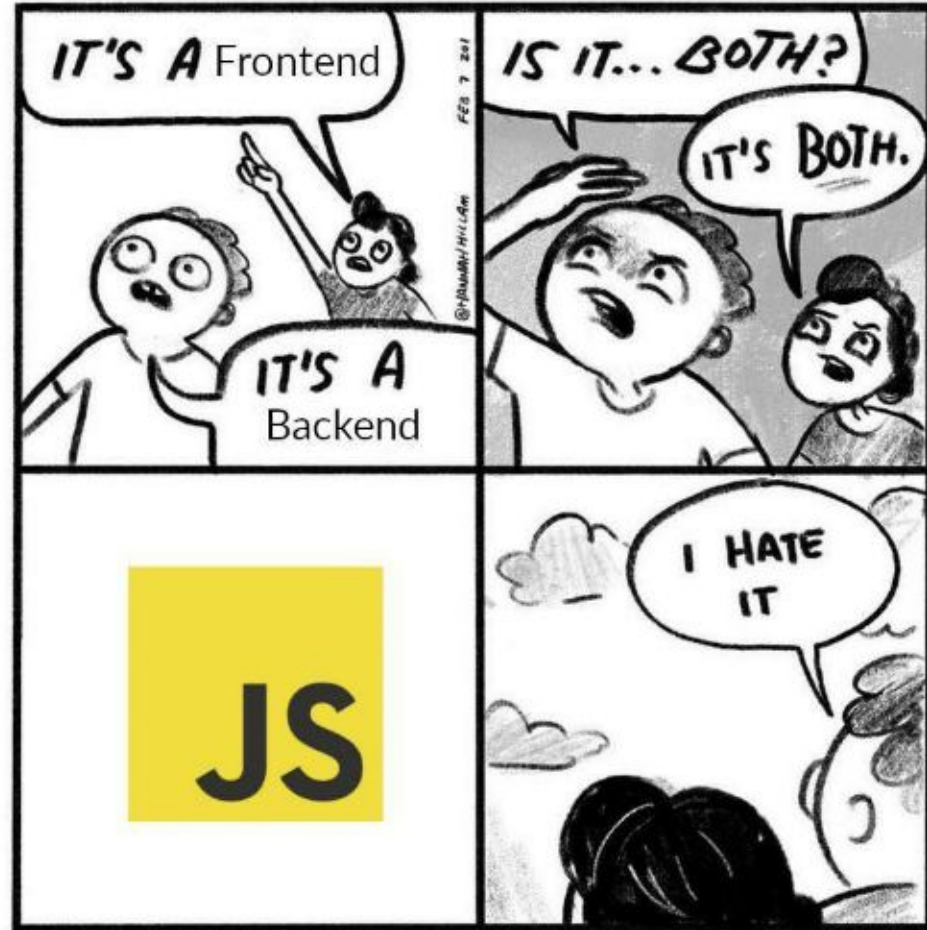
JS  
101



# Today's plan

- NodeJS and npm
- Transpilation & Babel
- Bundlers (Webpack, Rollup, Snowpack, Vite)
- Tree Shaking
- Minification
- Code Splitting

# NodeJS



# System APIs

command line utils

HTTP files net

path cluster

OS

UDP process

crypto timers

URL TLS/SSL console

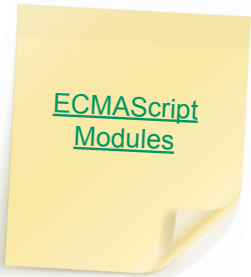
HTTPS



# ECMAScript Modules (.mjs files)

```
// circle.js
const { PI } = Math;
export const area = (r) => PI * r ** 2;
export const circumference = (r) => 2 * PI * r;
```

```
// index.js
import { area } from './circle.js'
console.log(`The area of a circle of radius 4 is ${area(4)}`);
```



ECMAScript  
Modules

# CommonJS Modules (.js files)

```
// circle.js
const { PI } = Math;
module.exports = {
  area: (r) => PI * r ** 2,
  circumference: (r) => 2 * PI * r
};
```

```
// index.js
const circle = require('./circle.js');
console.log(`The area of a circle of radius 4 is ${circle.area(4)}`);
```



# Historical sidenote: Why “*ECMAScript*”?

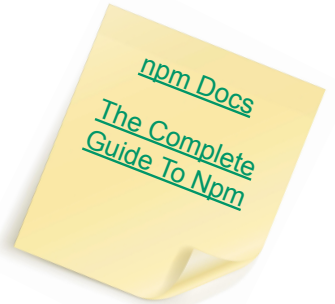
- **E**uropean **C**omputer **M**anufacturers **A**ssociation
- Purpose: Standardization of ICT systems (like ISO)
- JS naming history: Mocha → LiveScript → JavaScript → ECMAScript
- JS = ES + DOM + BOM
- Very irrelevant. Just use **JavaScript** :)

<https://stackoverflow.com/a/30113184>



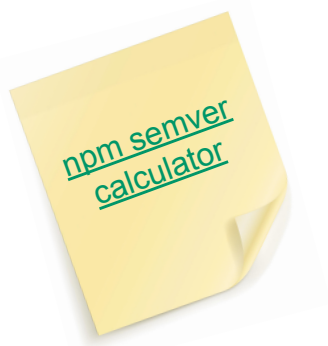
# npm

1. Node Package Manager (or **npm**) is the standard package manager of the Node.js ecosystem and a command-line interface tool used by developers to manage their Node.js projects
2. npm registry is the most extensive online package repository, containing over one-million packages



# Semantic Versioning (semver)

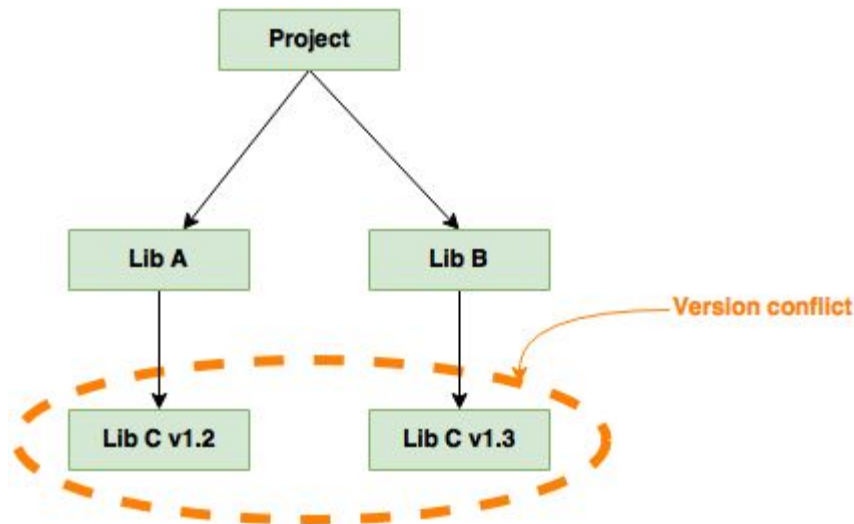
- A “pinky swear” among developers about updates to the package
- Consists of 3 numbers:
  - Major version: breaking changes
  - Minor version: adding new features
  - Patch version: fixing a bug
- Separated by dots: X.Y.Z
- Example: `lodash: 4.17.21`



# Dependency management

- Traditional package managers (e.g.: C++, Java, pip):
  - Only one version of a package at a time
  - Problematic for transitive dependencies with different versions
- Node:
  - Nesting (isolating) dependencies

```
loopback-boot@2.8.1
├── async@0.9.2
├── commondir@0.0.1
├── debug@2.2.0
│   └── ms@0.7.1
├── lodash@3.9.3
├── semver@4.3.6
├── toposort@0.2.10
└── loopback-datasource-juggler@2.18.1
    ├── async@1.2.1
    ├── debug@2.2.0
    │   └── ms@0.7.1
    └── depd@1.0.1
```



# npm commands examples

*# Initialize a project*

```
npm init
```

*# Install a package*

```
npm install <package>
```

*# Remove a package*

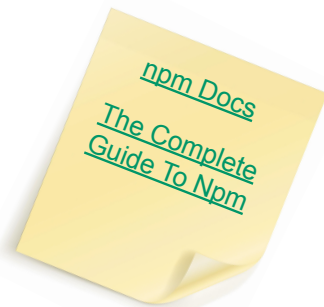
```
npm uninstall <package>
```

*# Update all packages*

```
npm update
```

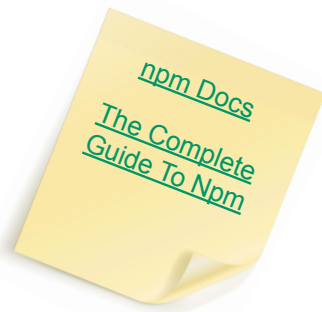
*# Run any script*

```
npm run <script>
```



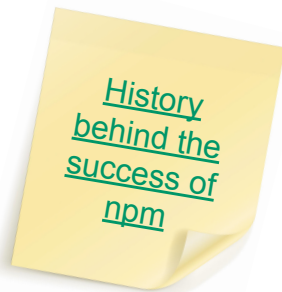
# package.json

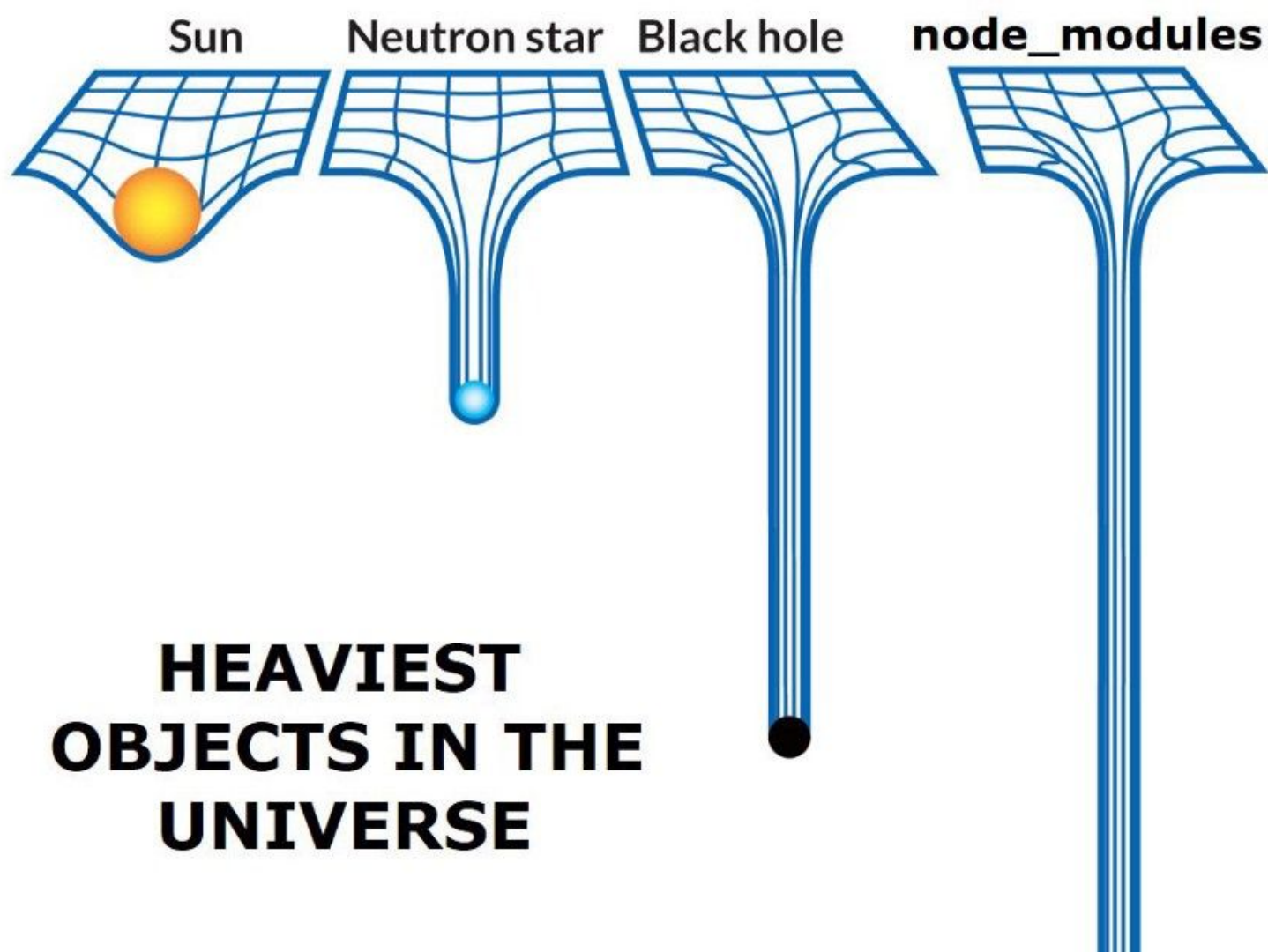
```
{  
  "name": "react",  
  "description": "React is a JavaScript library for building user interfaces.",  
  "version": "17.0.3",  
  "homepage": "https://reactjs.org/",  
  "license": "MIT",  
  "main": "index.js",  
  "dependencies": { ... },  
  "devDependencies": { ... }  
}
```



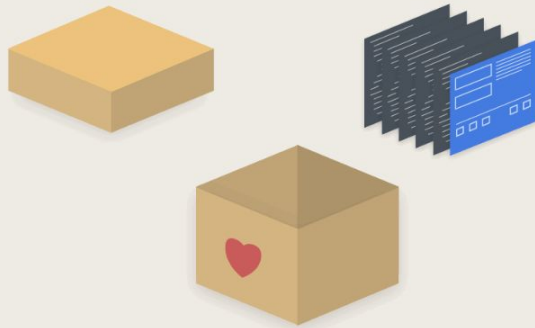
# Node/npm on the frontend

- Same technology can be useful in frontend development as well
  - Publishing/importing reusable code
- Browsers don't support it? No problem!
  - We're in control over the development process
  - Can simply generate stuff the browser understands
- Everything is already JavaScript!





# Bundlers

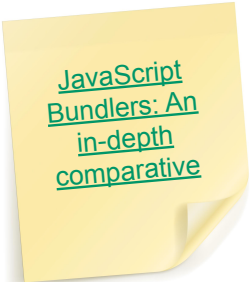




# Bundler

A bundler is a development tool that combines many JavaScript code files into a single one that is production-ready loadable in the browser.

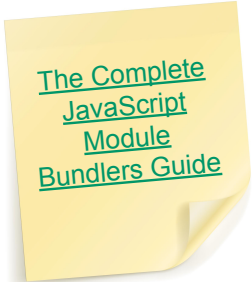
It generates a dependency graph as it traverses your first code files, and thus it keeps track of both your source files' dependencies and third-party dependencies.

A yellow sticky note with a folded bottom-right corner, containing the text: 

JavaScript  
Bundlers: An  
in-depth  
comparative

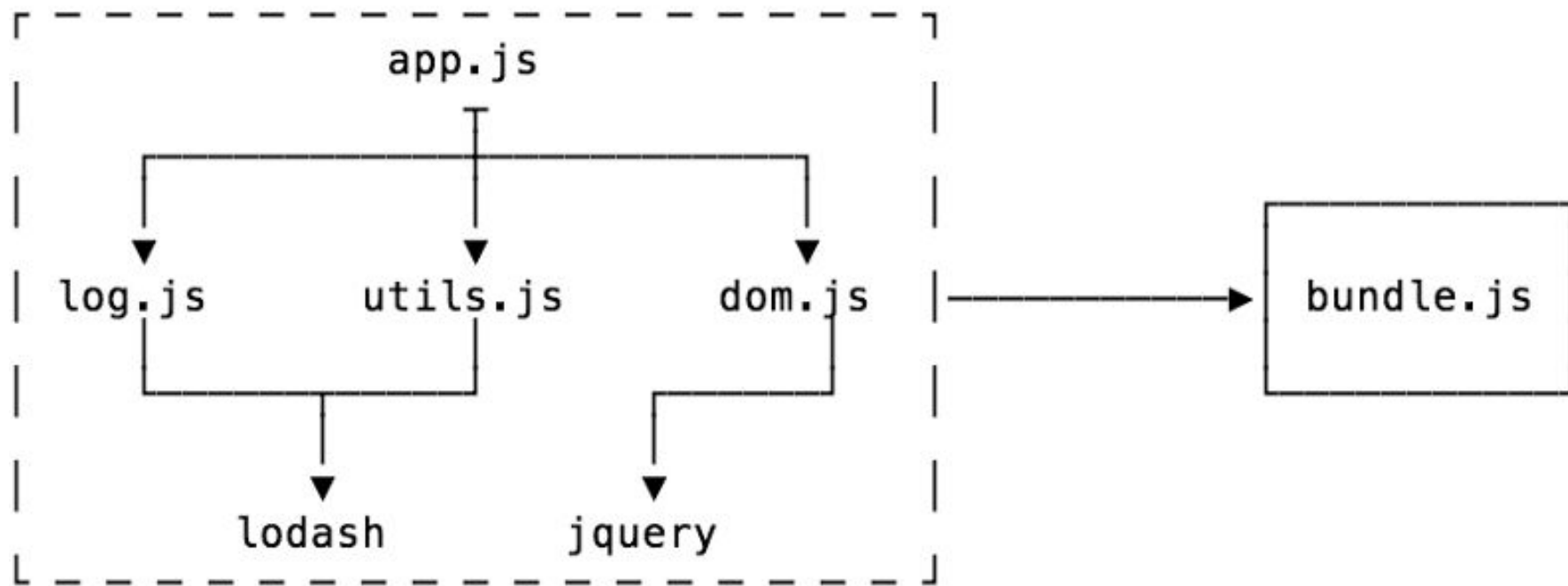
A yellow sticky note with a folded bottom-right corner, containing the text: 

JavaScript  
Bundlers: An  
in-depth  
comparative

A yellow sticky note with a folded bottom-right corner, containing the text: 

The Complete  
JavaScript  
Module  
Bundlers Guide

# Bundler



# Why do we need it?

- Combine modules together in one production-ready file
- Use imports to manage dependencies
  - rather than order of `<script>s`
- Create more complicated build pipelines
  - e.g.: replace some string in the code with an environment variable
- Import other types of files (images, CSS, ...)
- Code splitting
- Output code in multiple different formats (ESM, CJS, ...)
  - Or standards (ES5, ES6, ES2022, ...)

# Some bundlers

Top 5  
JavaScript  
Module  
Bundlers for  
2021



webpack

browserify



Snowpack



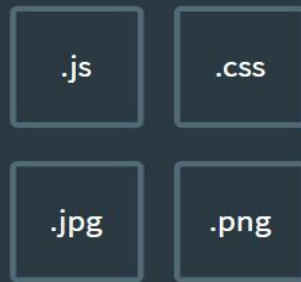
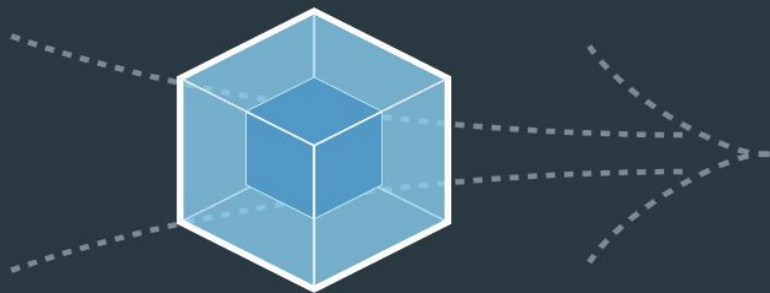
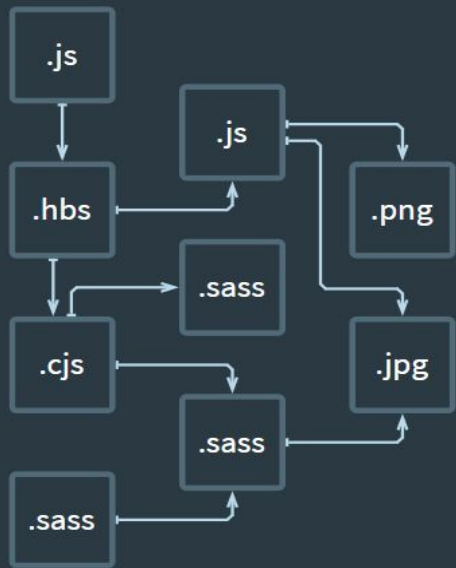
rollup.js



PARCEL

# Webpack

bundle your assets



STATIC ASSETS


# Tree Shaking



# Dead Code Elimination

Removal of code that's never going to run no matter what you do.

```
function answer() {  
    return 42;  
    console.log('Found it!');  
}  
if (false) { // for debugging  
    console.log('Finished calculation');  
}
```

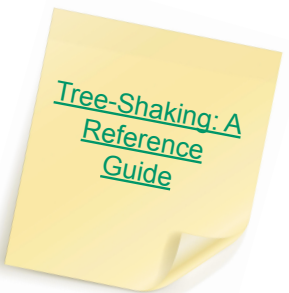


newline tree  
shaking

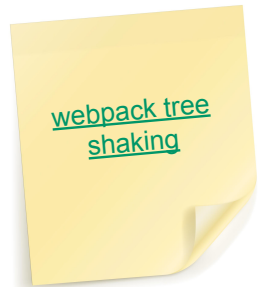
# Tree Shaking

Same as dead code elimination, but based on modules and their usages.


A module can export multiple features, but only ones that are imported (or used transitively) will remain in the bundle.




[Tree-Shaking: A Reference Guide](#)



[webpack tree shaking](#)



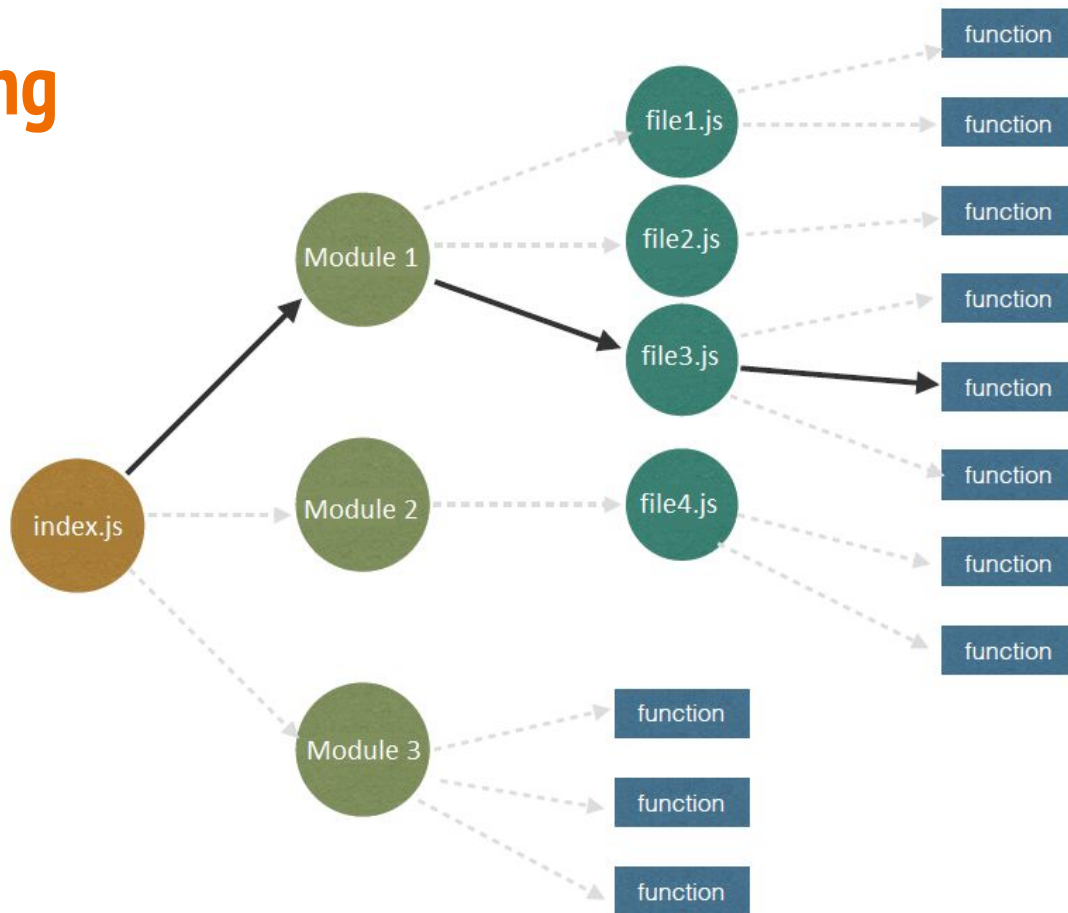
[How To Make Tree Shakeable Libraries](#)



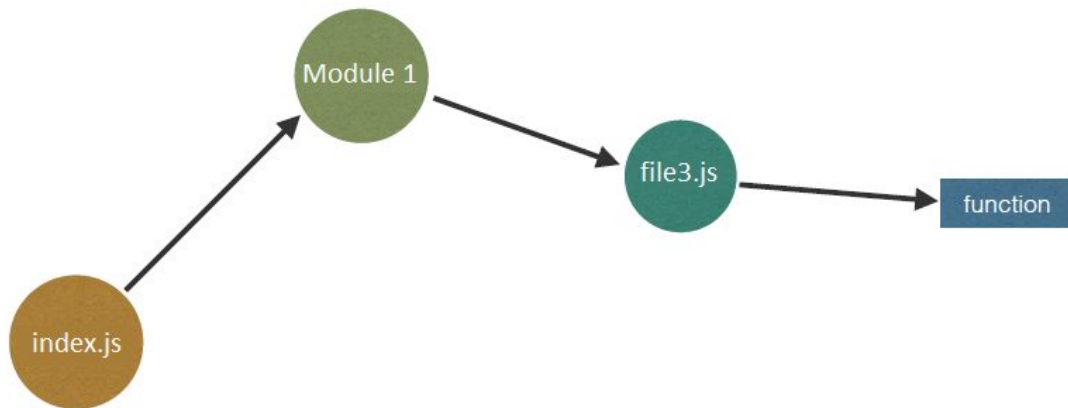
[MDN Tree shaking](#)



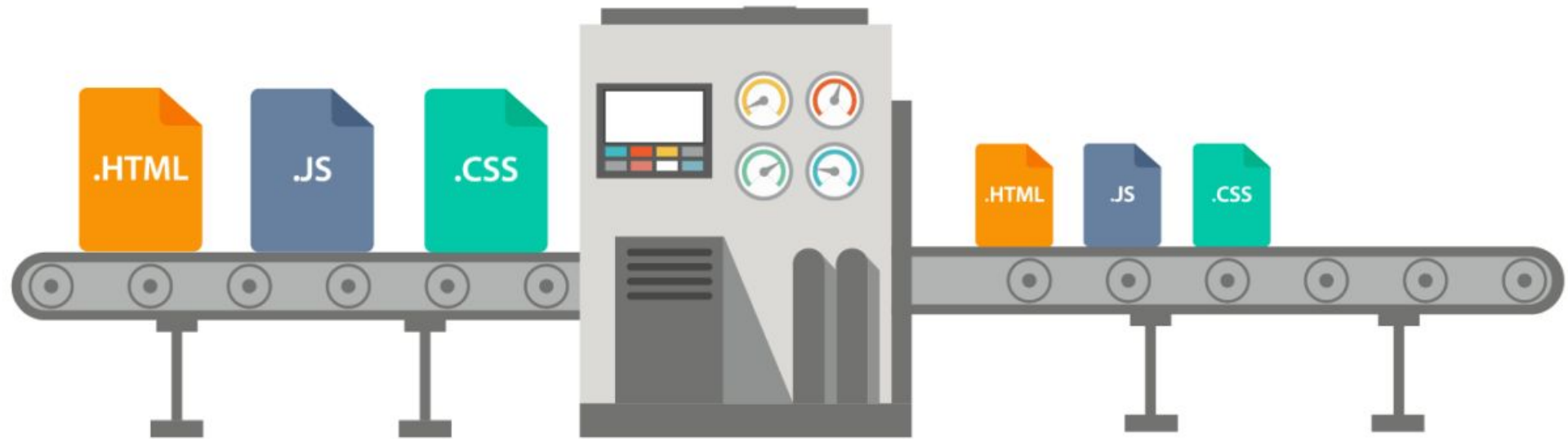
# Tree Shaking



# Tree Shaking



# Minification



# Minification

What is  
Minification?

Minimizes the amount of code shipped as much as possible by:

- Removing redundant whitespace
- Shortening variable/function names
- Removing comments
- Replacing code constructs with other functionally-equivalent but shorter ones

```
// This is a comment that will be removed  
const array = [];  
for (var i = 0; i < 20; i++) {  
    array[i] = i;  
}
```

```
for(var a=[i=0];i<20;a[i]=i++);
```

# What can be minified?

- HTML
- CSS
- JavaScript
- JSON
- Images and other media files\*

\* By decreasing quality (more like optimization than “minification”)

# Benefits of minification & tree-shaking

1. Faster load times
2. Lower business costs
  - a. less data is transmitted over the network
  - b. lower resource usage since less data needs to be processed for each request
  - c. generated once, use for an unlimited number of requests.
3. Better development experience

# Before Minification

```
<html>
<head>
  <style>
    #myContent { font-family: Arial }
    #myContent { font-size: 90% }
  </style>
</head>
<body>
  <!-- start myContent -->
  <div id="myContent">
    <p>Hello world!</p>
  </div>
  <!-- end myContent -->
</body>
</html>
```

Total: 250 bytes

## After Minification

```
<style>#d{font-family:Arial;font-size:90%}</style><div  
id=d><p>Hello world!</div>
```

Total: 81 bytes




# Code Splitting

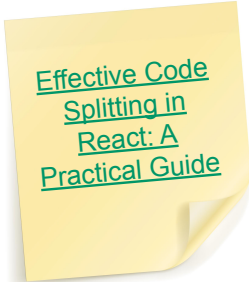
# Code Splitting

Code splitting allows you to split your code into multiple bundles which can then be loaded on demand or in parallel.


It can be used to achieve smaller bundles and control resource load prioritization which can have a major impact on load time.



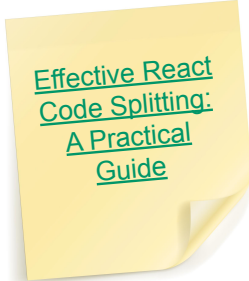
Webpack Code Splitting



Effective Code Splitting in React: A Practical Guide






Reduce JavaScript payloads with code splitting









Effective React Code Splitting: A Practical Guide

# Code Splitting: tradeoff

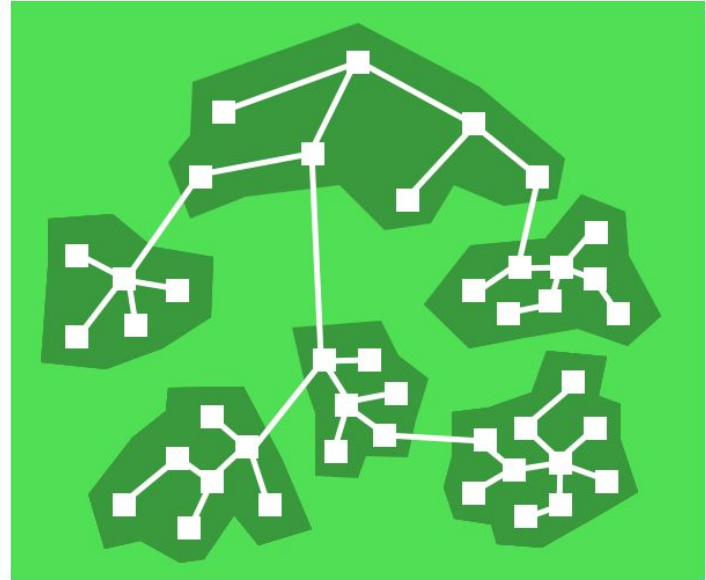
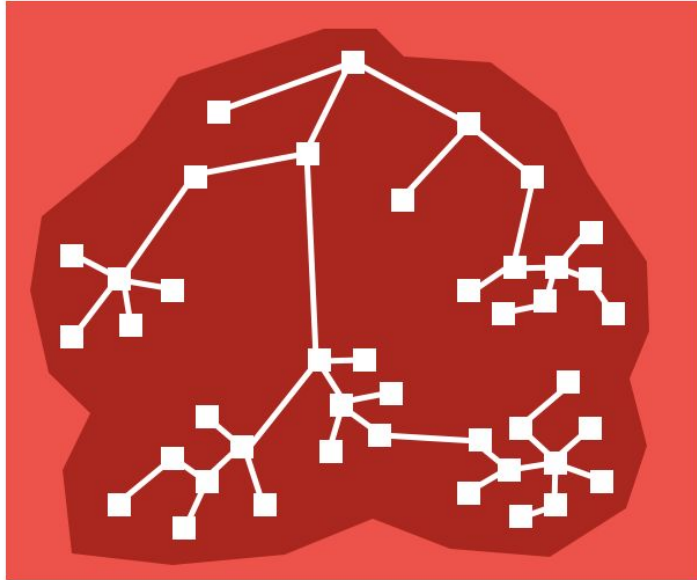
## Single request:

- Reduced latency 
- Larger bundle 
- Larger time to interactive 

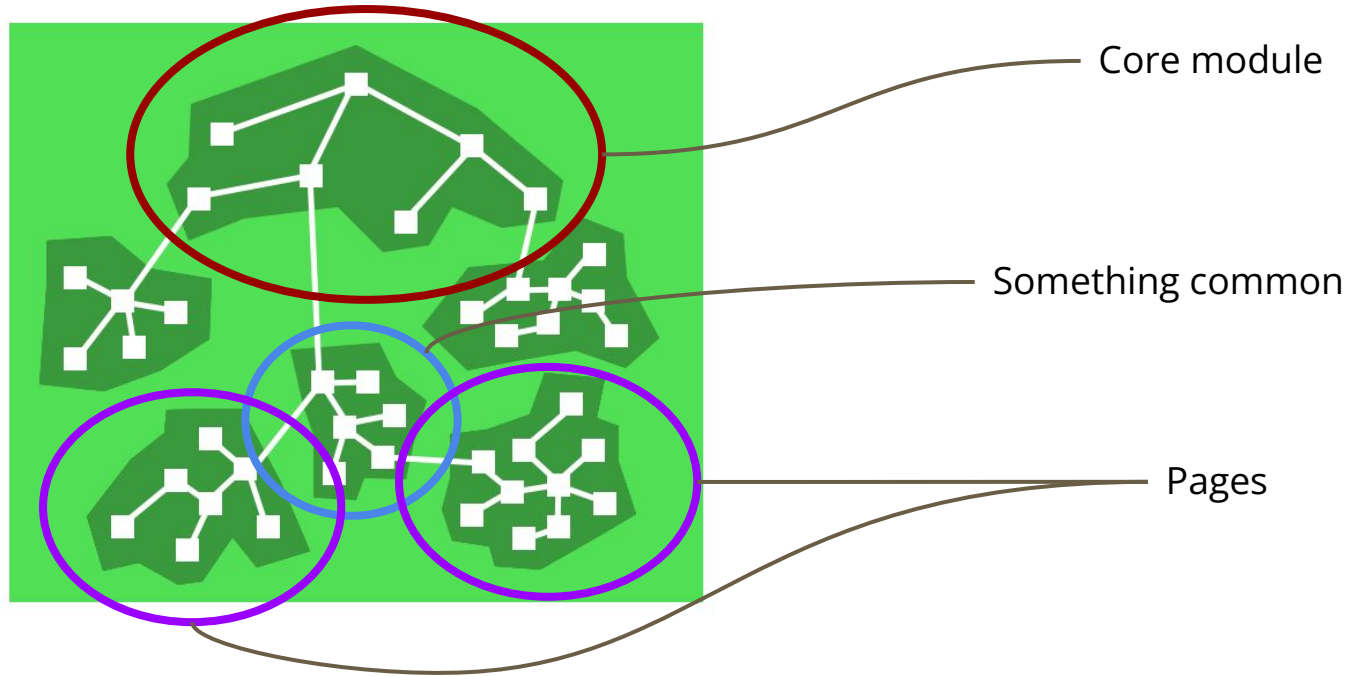
## Multiple requests:

- Fetch only what is needed for this page 
- Smaller individual bundles 
- Reduced time to interactive 
- Multiple requests 
- Increased latency 
- Increased complexity 


# Low Coupling and High Cohesion






# Code Splitting



# Code Splitting

Name	Status	...	I...	Size
 main.bundle.js	200	...	(i...	15.2 KB



Name	Status	...	I...	Size
 main.bundle.js	200	...	(i...	2.7 KB
 1.bundle.js	200	...	...	13.9 KB


# Transpilation & Babel



*BABEL*

# Transpilation

Transpilation is the process of transforming the program written in language **X** into the equivalent program in language **Y**. In contrast to compilation, languages **X** and **Y** have roughly the same level of abstraction.



Why Do We  
Need  
Transpilation  
Into JavaScript?



# Why do we need it?

1. Migration between different versions of the same language
2. Translation from one programming language into another based on the runtime system requirements and/or developers' wishes

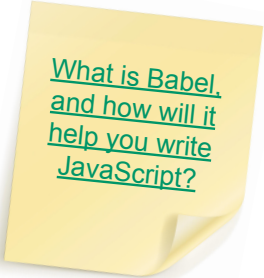
# Babel

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. Here are the main things Babel can do for you:


- Transform syntax
- Polyfill features that are missing in your target environment (through a third-party polyfill such as core-js)
- Source code transformations (codemods)




Why you don't  
need Babel



What is Babel,  
and how will it  
help you write  
JavaScript?



Do we need  
Babel in every  
project



What is babel?

## Example: JS

*// Babel Input: ES2015 (ES6) arrow function*  
`[1, 2, 3].map(n => n + 1);`

*// Babel Output: ES5 equivalent*  
`[1, 2, 3].map(function(n) {  
 return n + 1;  
}));`

# Example: React

*// JSX Syntax*

```
const Component = () => {  
  return (  
    <div style={{color: '#fff'}}>  
      Convert JSX to JS  
    </div>  
  )  
};
```

*// Pure JS*

```
const Component = () => {  
  return React.createElement('div', {  
    style: {color: '#fff'}  
  }, 'Convert JSX to JS')  
};
```



JSX In Depth

# Polyfills

“A piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.” - MDN

```
// check if the method `startsWith` exists on the standard built-in `String`  
if (!String.prototype.startsWith) {  
    // if not we add our own version of the native method newer browsers provide  
    String.prototype.startsWith = function (searchString, position) {  
        position = position || 0;  
        return this.substr(position, searchString.length) === searchString;  
    };  
}
```

Introduction To  
Polyfills & Their  
Usage

Polyfills and  
transpilers

**Parents: No this won't affect our kid.**



**Meanwhile kid:**



**Toby, Age 3  
Alcoholic**

That's all for today!