

stencil

From Frameworks to Compilers

Manu Mtz.-Almeida



Manu Mtz.-Almeida

Senior Software Engineer on the
Ionic Framework Team

 @manucorporat

 @manucorporat

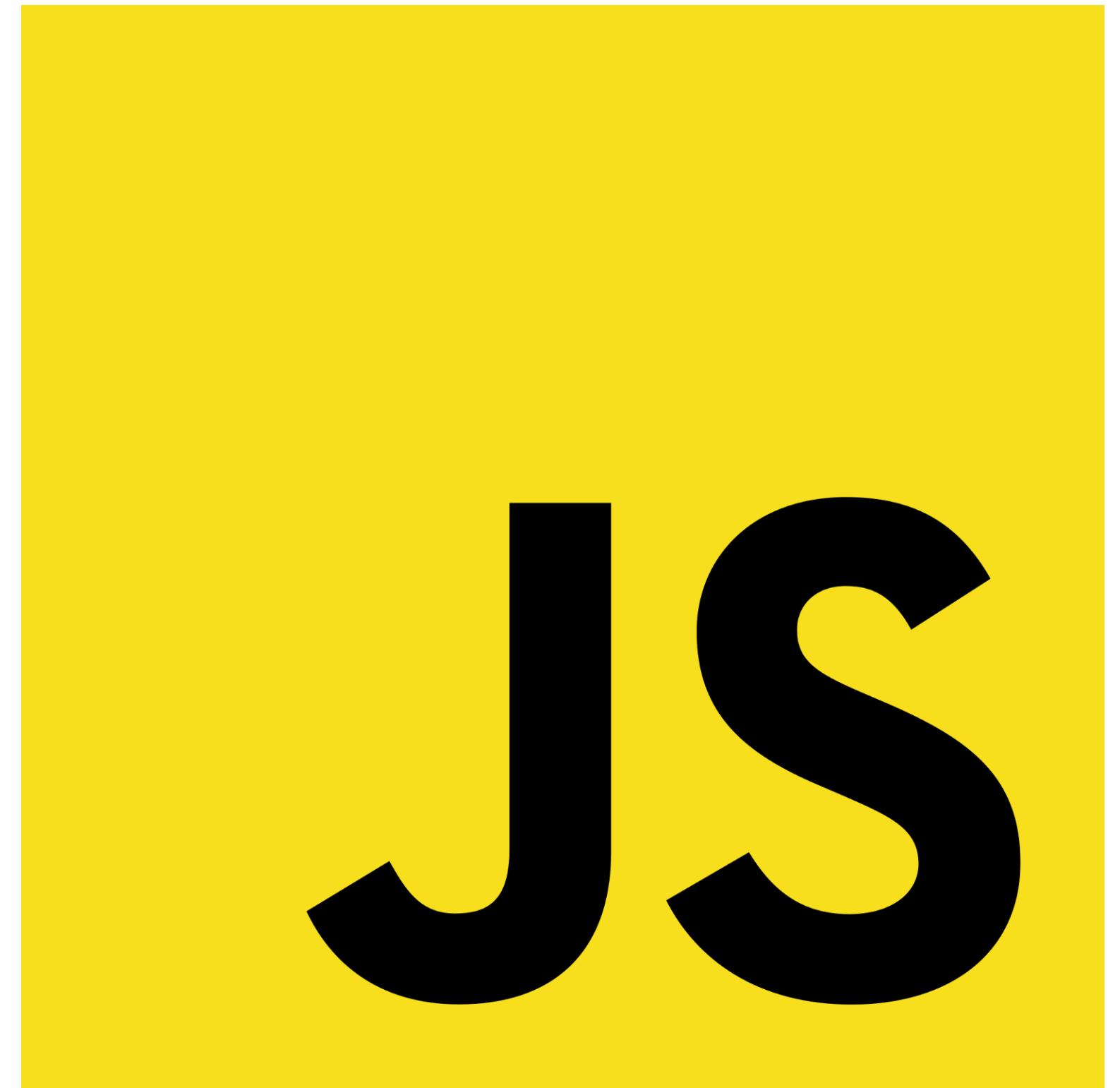
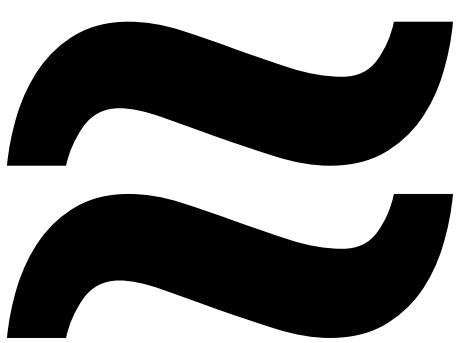


based in a true story

12 years ago



jQuery

The jQuery logo consists of the word "jQuery" in a dark navy blue, sans-serif font. Above the letter "j", there are four interlocking blue crescent shapes arranged in a curve.

JS

The JS logo is composed of the letters "JS" in a large, bold, black sans-serif font. It is set against a solid yellow rectangular background.



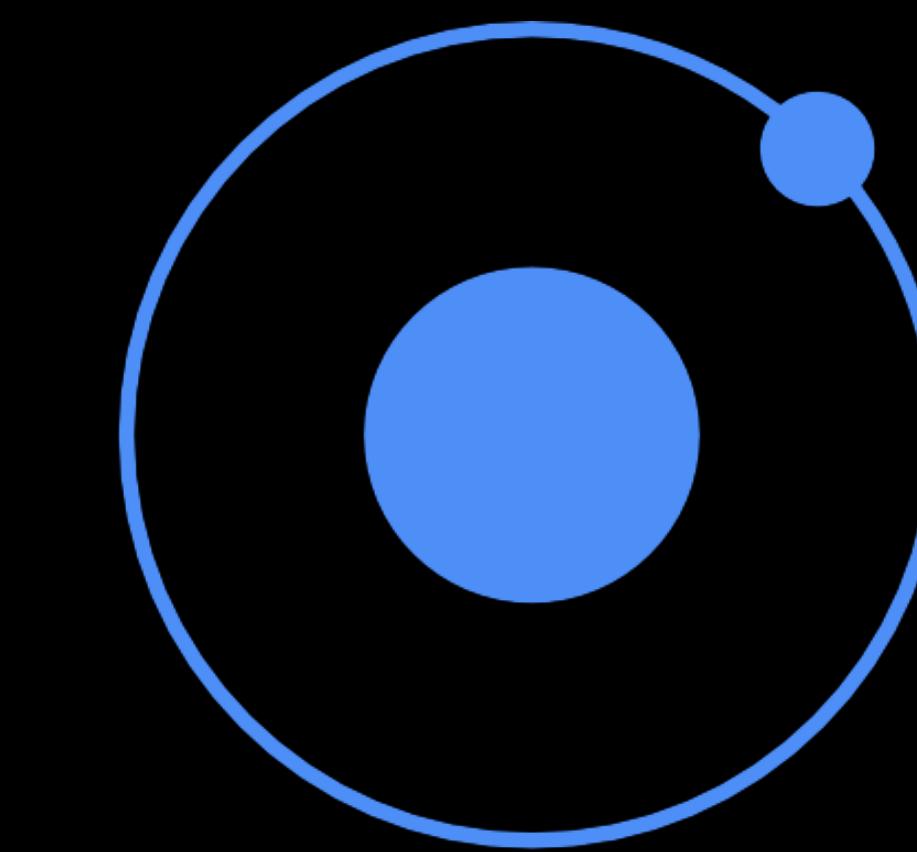
codiqa

some years later...

AngularJS!!!!

DIRECTIVES





ionic

A medium shot of a woman with long, wavy brown hair and dark-rimmed glasses. She is wearing a dark-colored blazer over a white collared shirt. Her gaze is directed towards the right side of the frame. The background is a solid, muted purple.

Angular 2!!



Angular 4, 5, 6, 7...

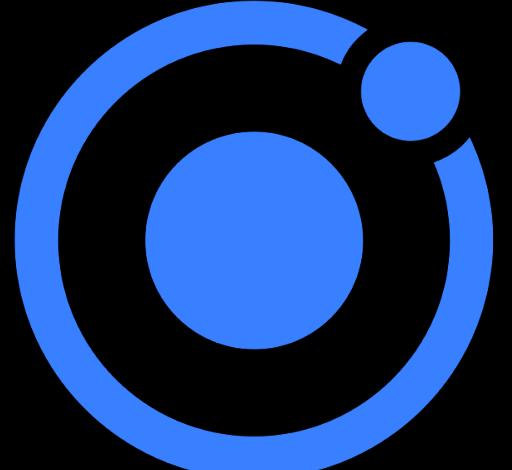
Ember

Vue

React

Preact

Polymer



ionic vue



ionic react



ionic



The end of

Framework Churn

Frameworks



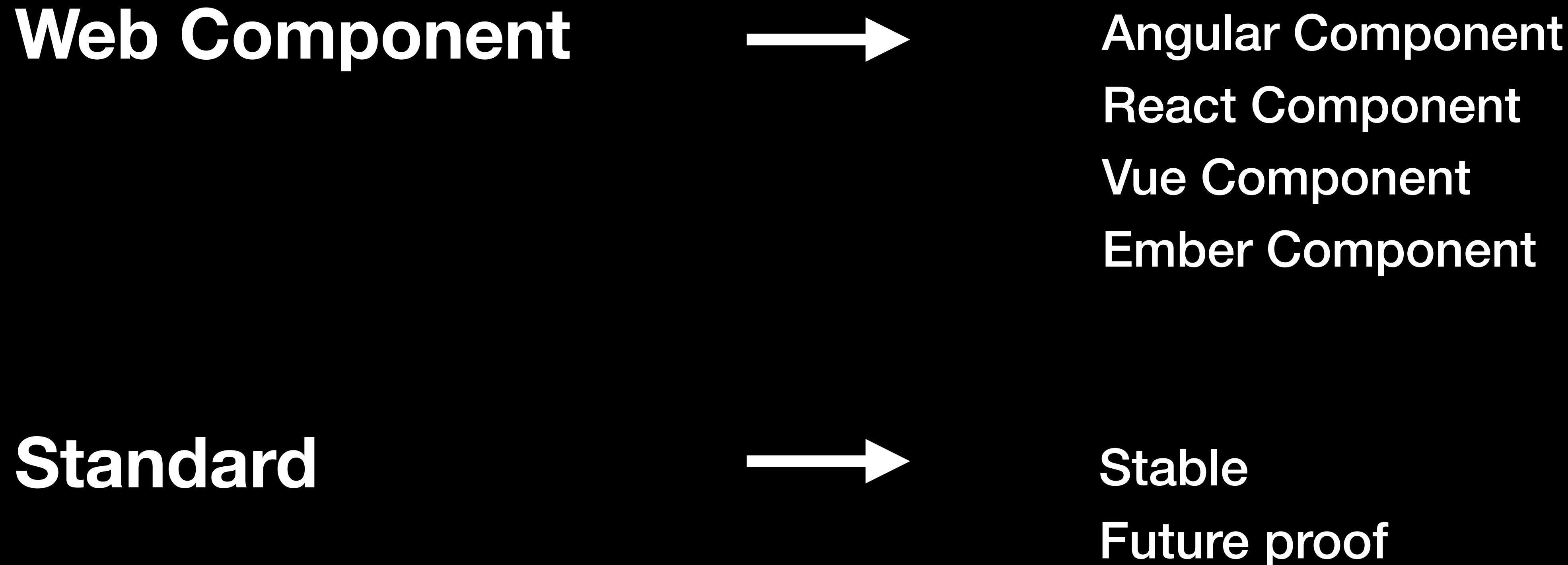
Great for final products



Terrible for reusable components

Web Components

Universal Model



				
HTML <template>	26	13	22	8
Shadow DOM	53	in development	63	10.1
Custom Elements	54	in development	63	10.1
ES Modules	61	16	60	10.1

Problem:
WC is a low level API

Solution:
create an abstraction

New Problem:
overhead

Unless...
build-time abstraction

The screenshot shows the Stencil.js homepage as it would appear in a web browser. The page has a dark header with the Stencil logo and navigation links for Docs, Demos, Resources, and GitHub. The main content features a large Stencil logo icon, followed by the tagline "The magical, reusable web component compiler". Below this, there's a section about Stencil's compatibility with popular frameworks and its unique features, each accompanied by a green checkmark. A prominent blue button at the bottom encourages users to start coding.

Stencil combines the best concepts of the most popular frontend frameworks and generates 100% standards-based Web Components that run in any modern browser.

- ✓ Typescript support
- ✓ A tiny virtual DOM layer
- ✓ JSX support
- ✓ Asynchronous rendering pipeline
- ✓ One-way data binding
- ✓ Simple component lazy-loading

Start coding with Stencil in seconds

npm init stencil

stencil?

It's a build-time abstraction

Ionic Requirements

stencil

Requirements

- Productivity
- Raw Web Components with minimal overhead
- Stability
- Scalability
- Performance
- Lazy loading

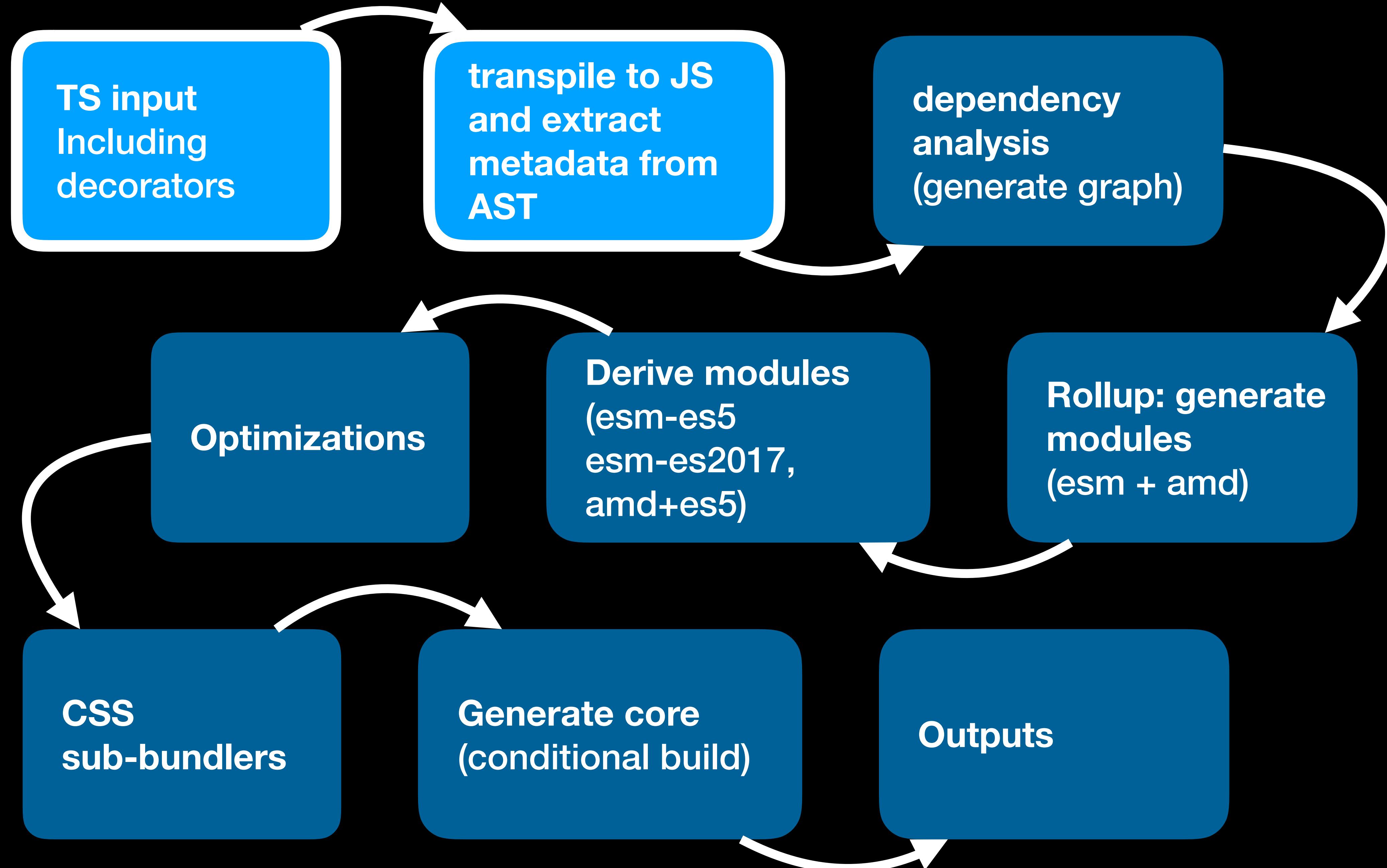
Deprecations → AST transforms

Small API / features → Stability

Standard features → Stability

Compiler-time feature > Run-time feature

Deep Dive



```
import { Component, Prop } from '@stencil/core';

export type Color = 'primary' | 'secondary' | 'dark';

@Component({
  tag: 'app-home',
  styleUrl: 'app-home.css',
  shadow: true
})
export class AppHome {

  /**
   * El color theme del componente.
   */
  @Prop() color: Color = 'primary';

  /**
   * Anchura del componente
   */
  @Prop() width?: number;

  /**
   * Un objeto que no es un atributo html.
   */
  @Prop() obj?: Object;

  render() {
    return (
      <div class={{
        'contenedor': true,
        [ `color-${this.color}` ]: true
      }}>
        <slot></slot>
      </div>
    );
  }
}
```

```
import { Component, Prop } from '@stencil/core';

export type Color = 'primary' | 'secondary' | 'dark';

@Component({
  tag: 'app-home',
  styleUrl: 'app-home.css',
  shadow: true
})
export class AppHome {

  /**
   * El color theme del componente.
   */
  @Prop() color?: Color = 'primary';

  /**
   * Anchura del componente
   */
  @Prop() width = 200;
}
```

Tag-name: “app-home”
Style: “app-home.css”
Shadow: true

Property: “color”
Type: Color => union type
Primitive: string | undefined
Docs

Property: “width”
Type: width => number
Primitive: number
Docs

render() used
class object used
Slot used

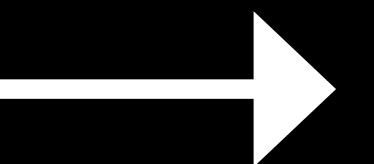
```
 * Anchura del componente
 */
@Prop() width ?: number;

/**
 * Un objeto que no es un atributo html.
 */
@Prop() obj ?: Object;

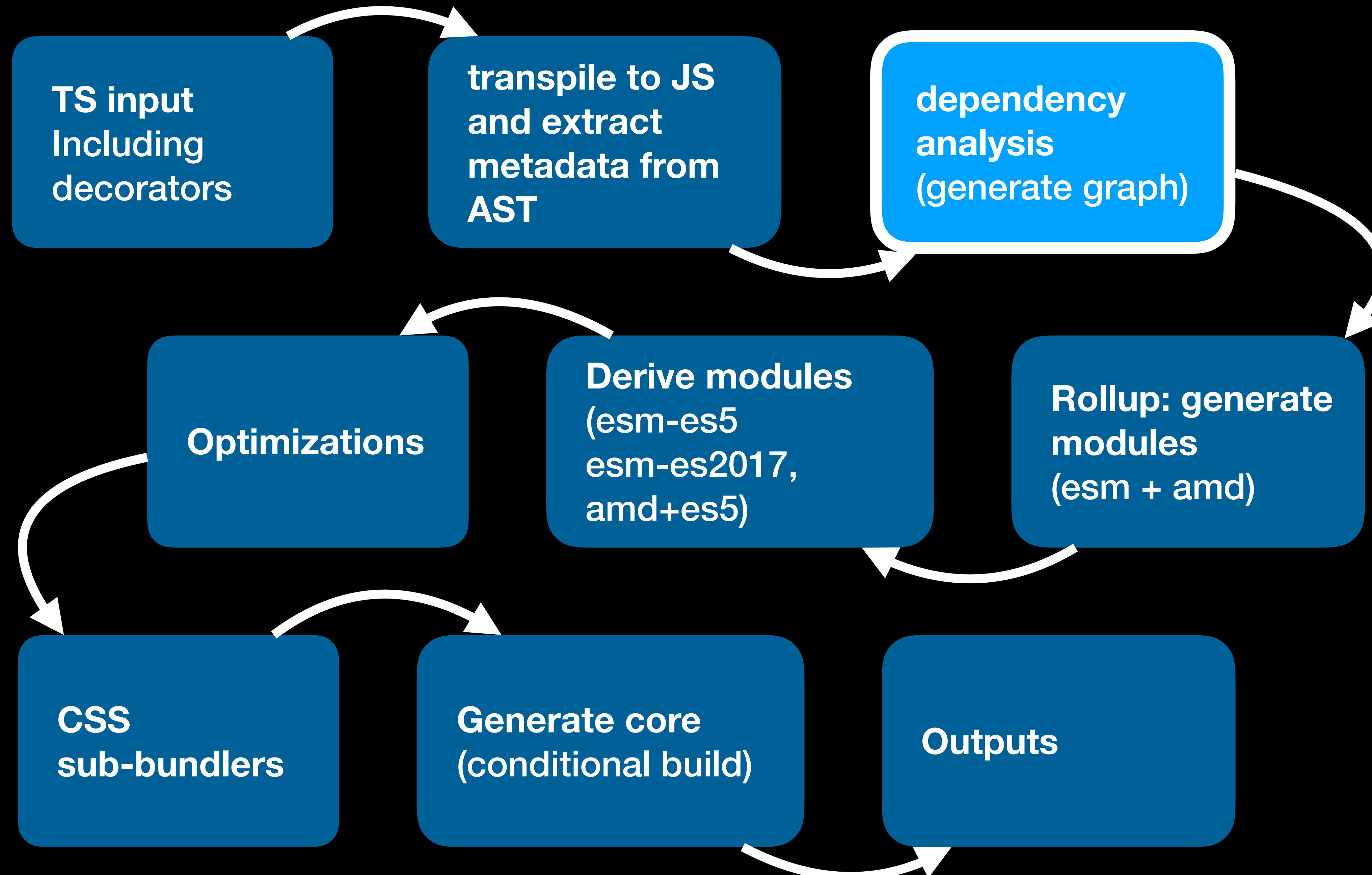
render() {
  return (
    <div class={{
      'contenedor': true,
      [`color-${this.color}`]: true
    }}>
      <slot></slot>
    </div>
  );
}
```

lowering features

```
/**  
 * When using a router, it specifies the transition direction when navigating to  
 * another page using `href`.  
 */  
@Prop() routerDirection?: RouterDirection;  
  
/**  
 * Contains a URL or a URL fragment that the hyperlink points to.  
 * If this property is set, an anchor tag will be rendered.  
 */  
@Prop() href?: string;  
  
/**  
 * The button shape.  
 * Possible values are: `^"round"`.  
 */  
@Prop({ reflectToAttr: true }) shape?: 'round';  
  
/**  
 * The button size.  
 * Possible values are: `^"small"`, `^"default"`, `^"large"`.  
 */  
@Prop({ reflectToAttr: true }) size?: 'small' | 'default' | 'large';  
  
/**  
 * If `true`, activates a button with a heavier font weight.  
 */
```

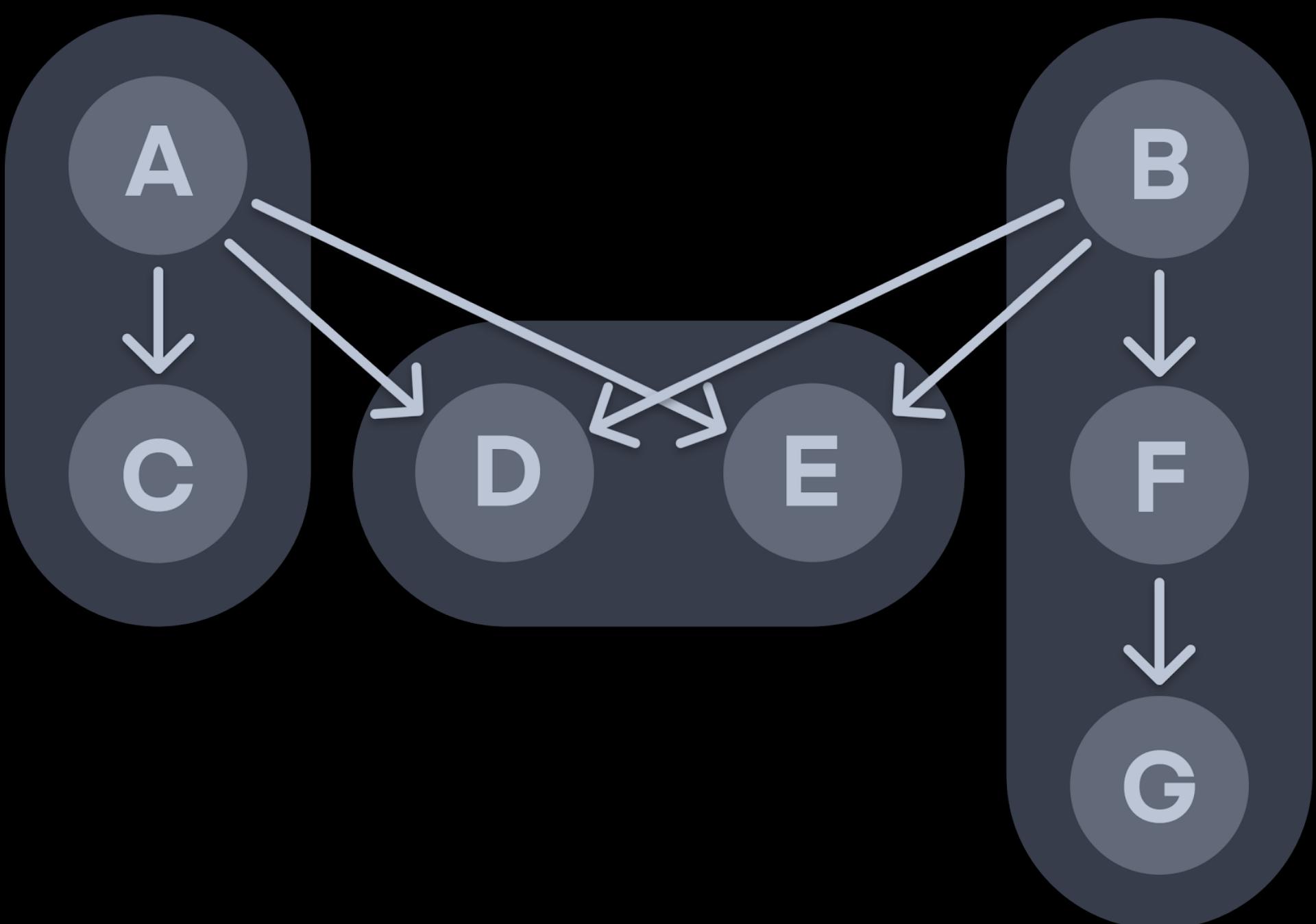


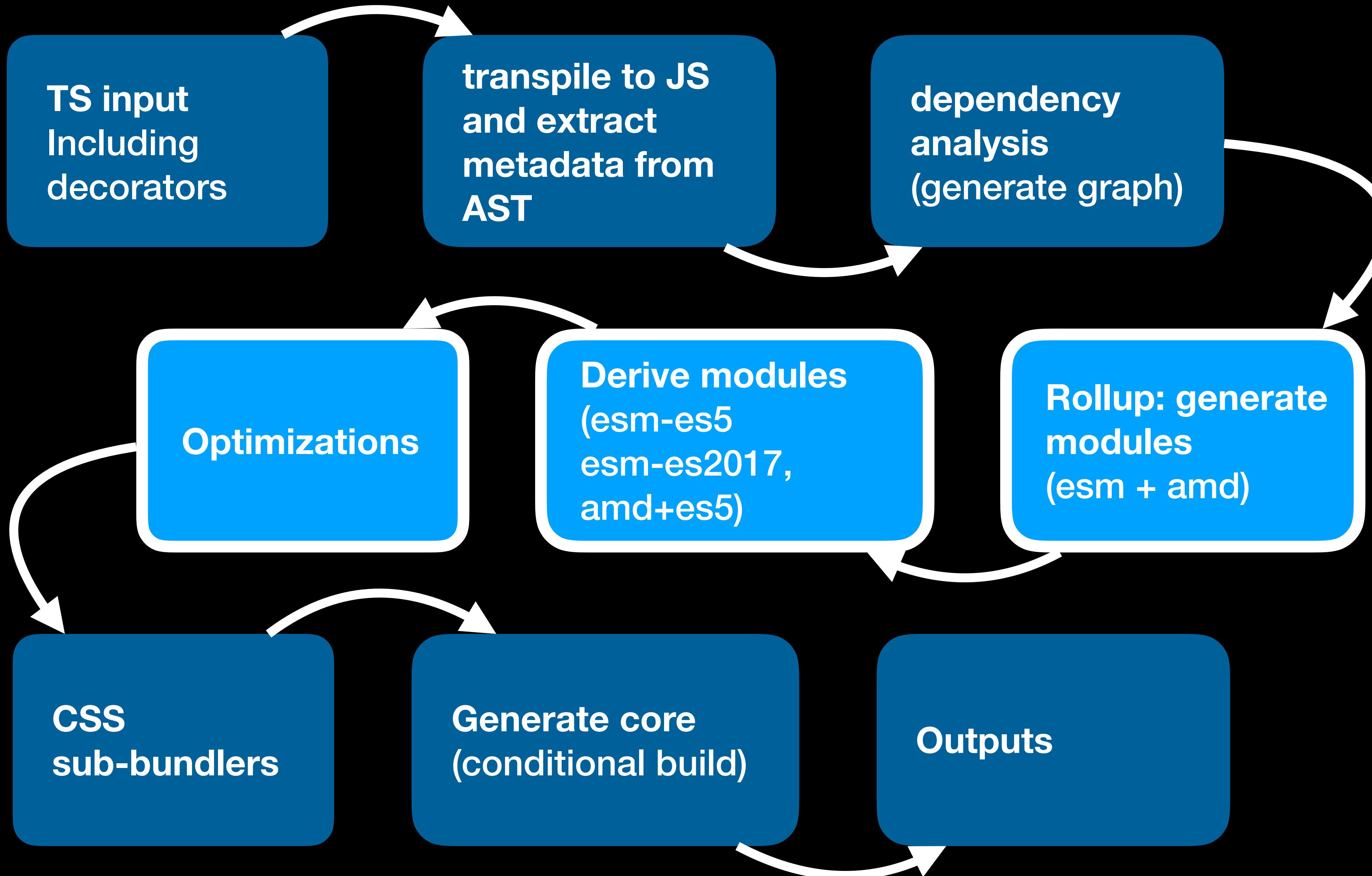
```
    "f": 193  
},  
"href": {  
    "f": 1  
},  
"keyFocus": {  
    "f": 16  
},  
"mode": {  
    "f": 1  
},  
"routerDirection": {  
    "attr": "router-direction",  
    "f": 1  
},  
"shape": {  
    "f": 129  
},  
"size": {  
    "f": 129  
},  
"strong": {  
    "f": 4  
},  
"type": {  
    "f": 1
```



graph analysis

- The compiler generates a dependency graph based in a static analysis
- Components are optimized further by groups them optimally
- HTTP network requests are reduced

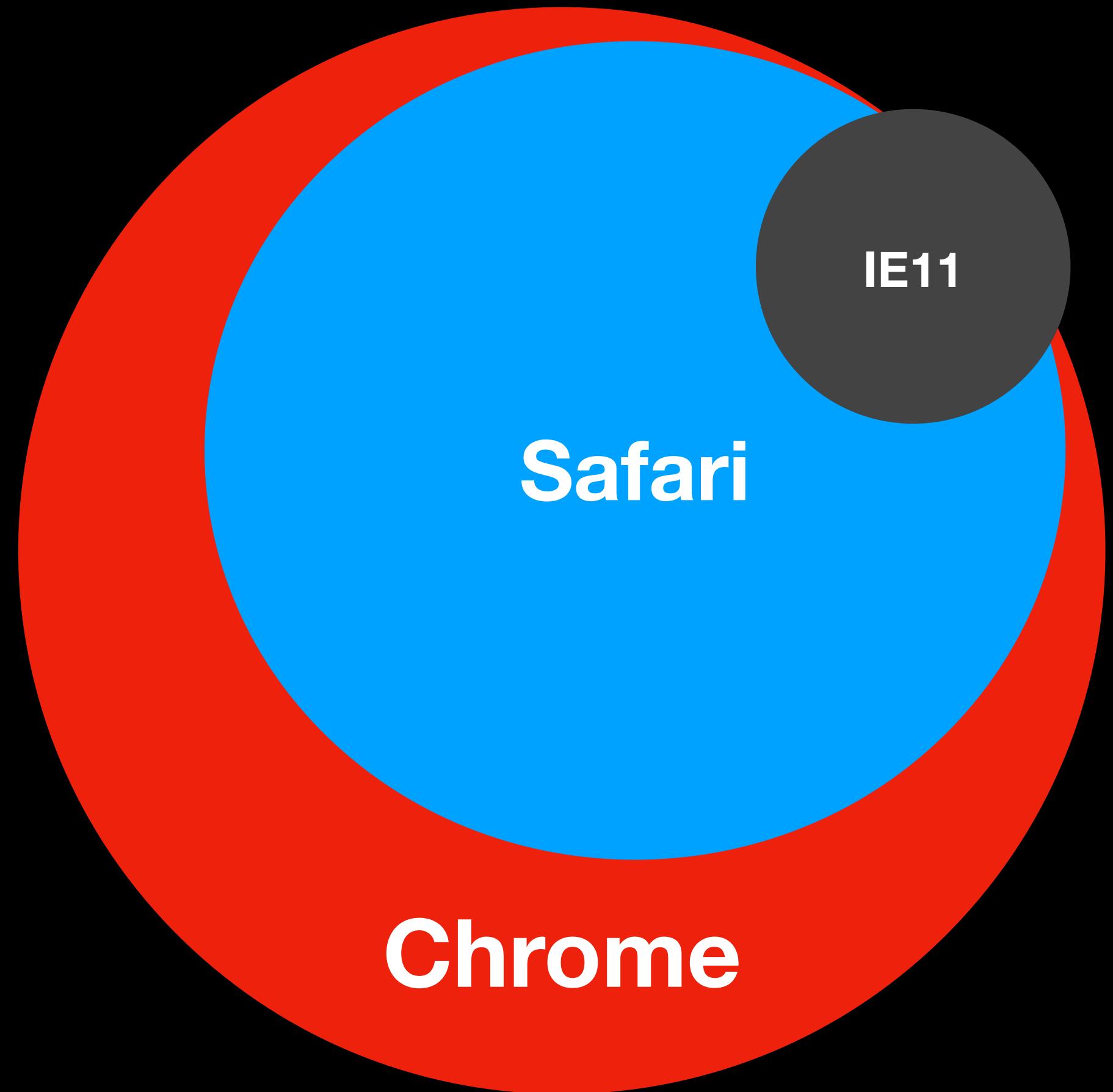




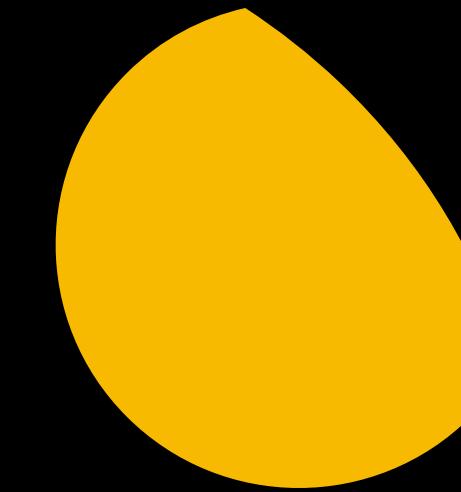
multiple targets

- Modern browsers
 - ES Modules nativos
 - No polyfills
- Legacy
 - AMD
 - Polyfills



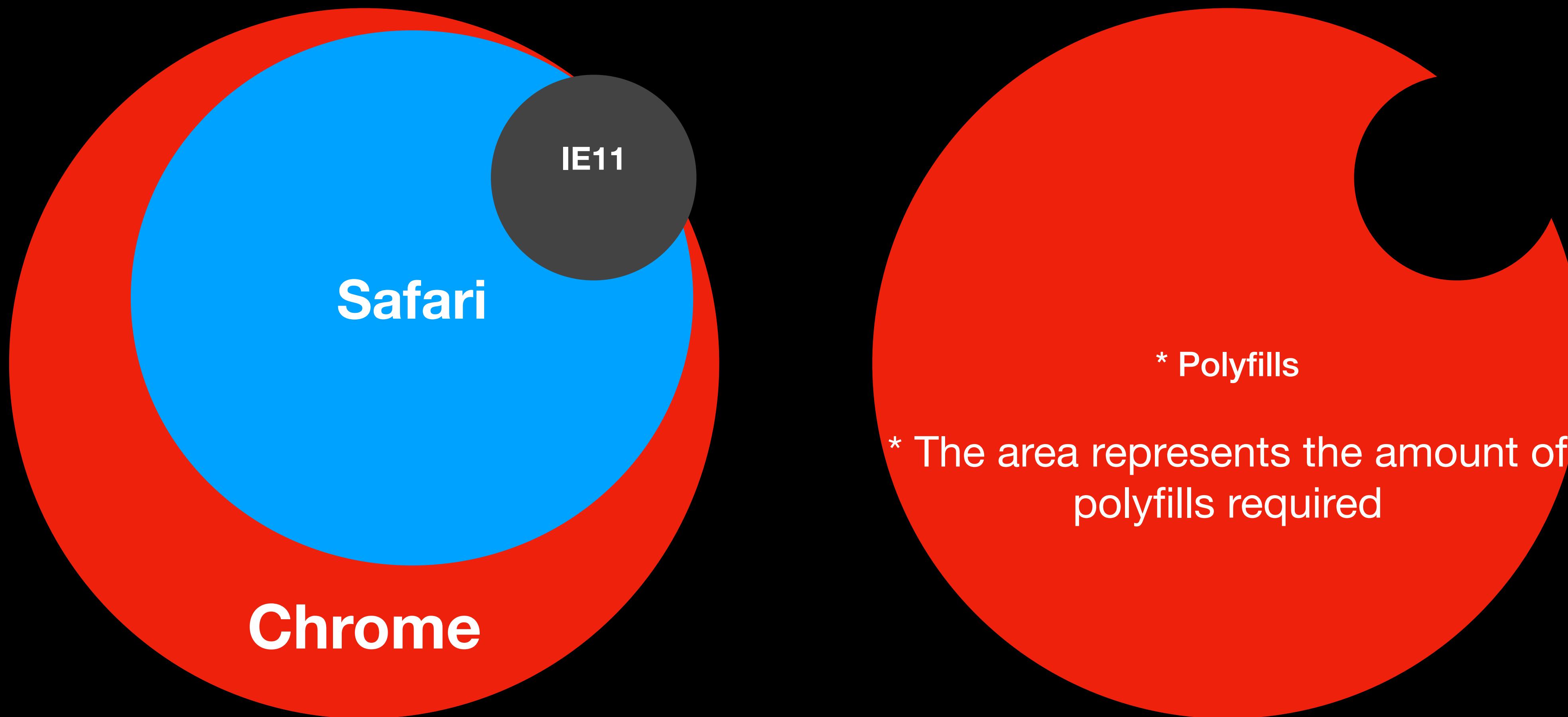


APIs developers can use



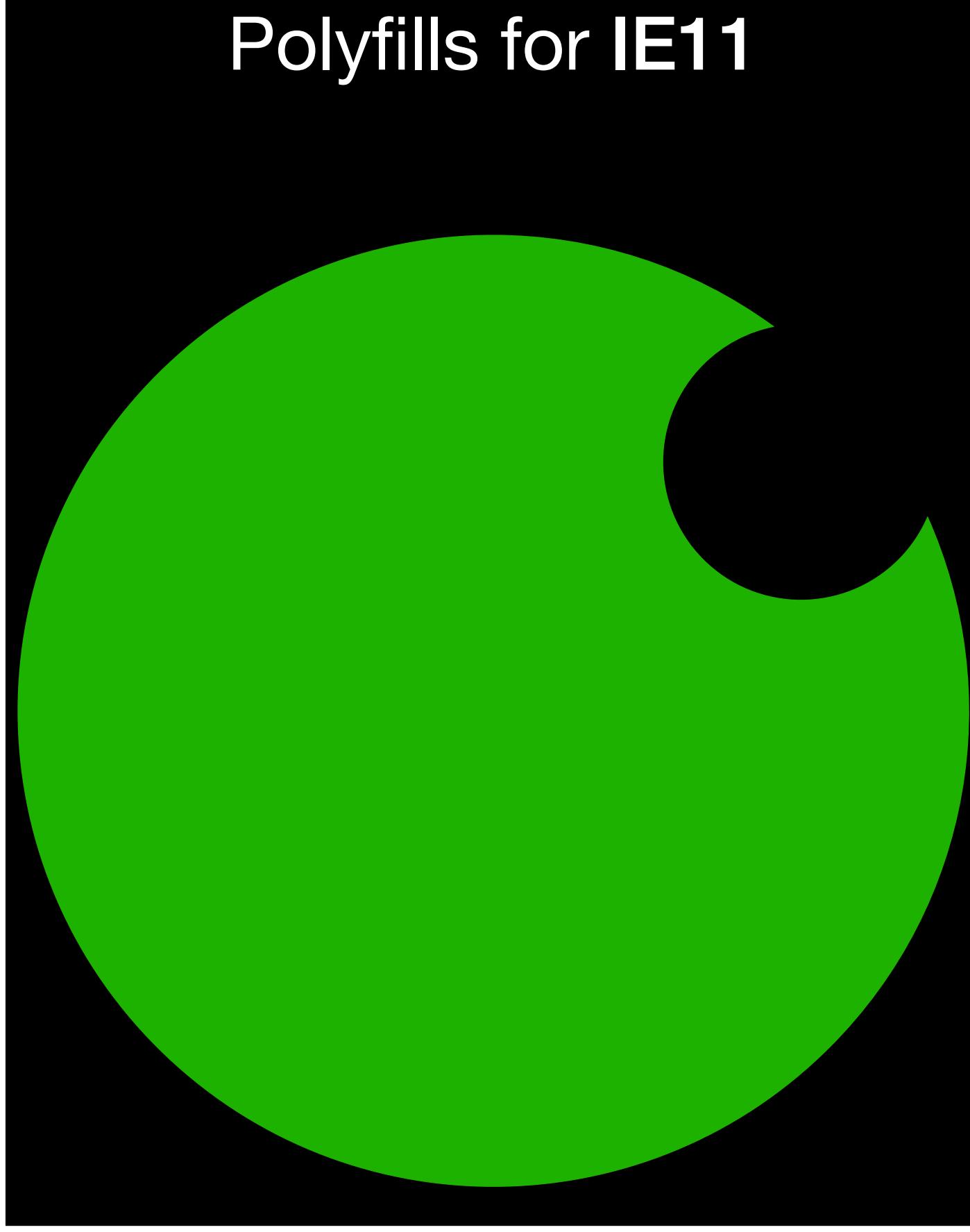
* The area represents the amount of APIs available for each browsers

#1: Polyfill for everyone

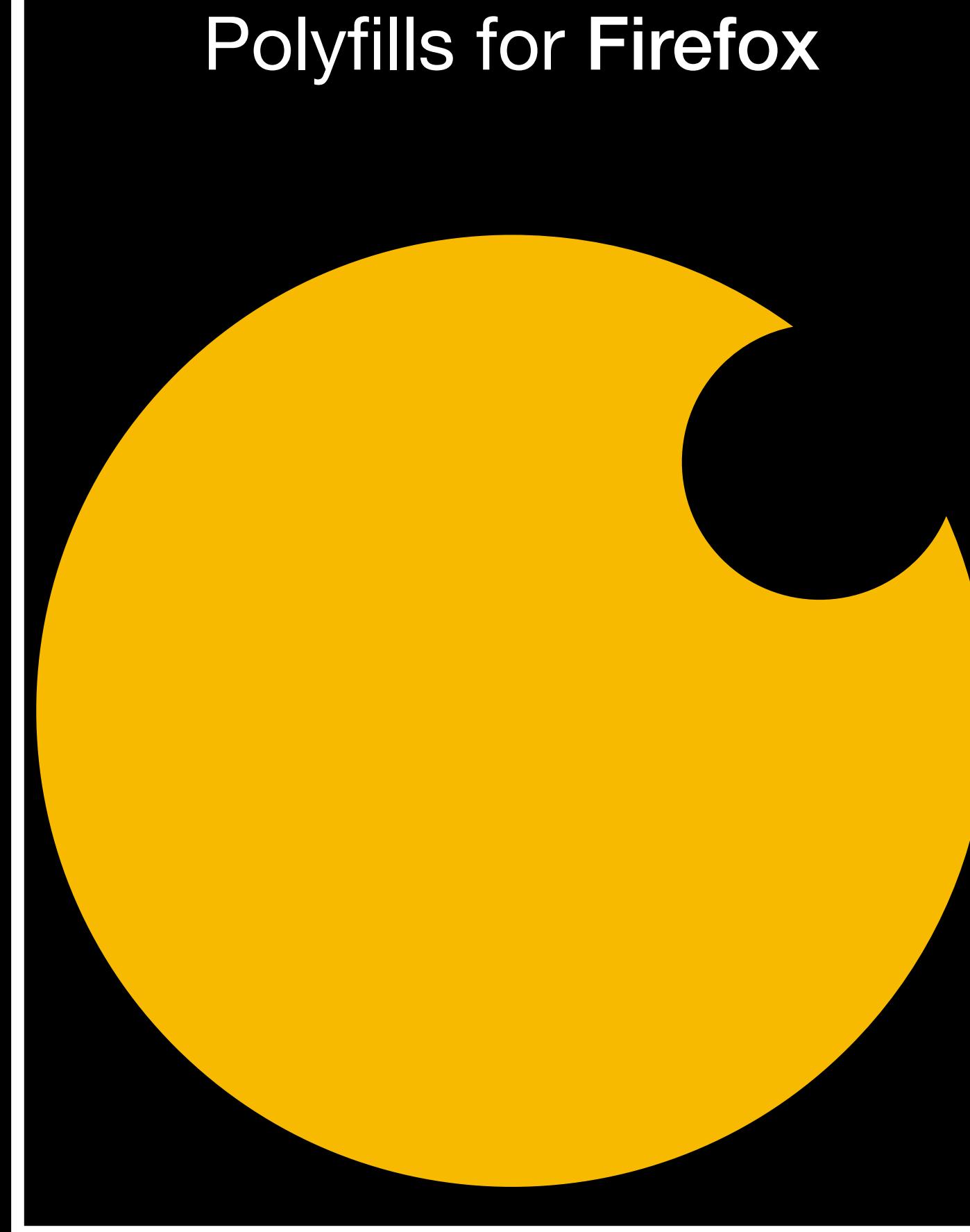


#1: Polyfill for everyone

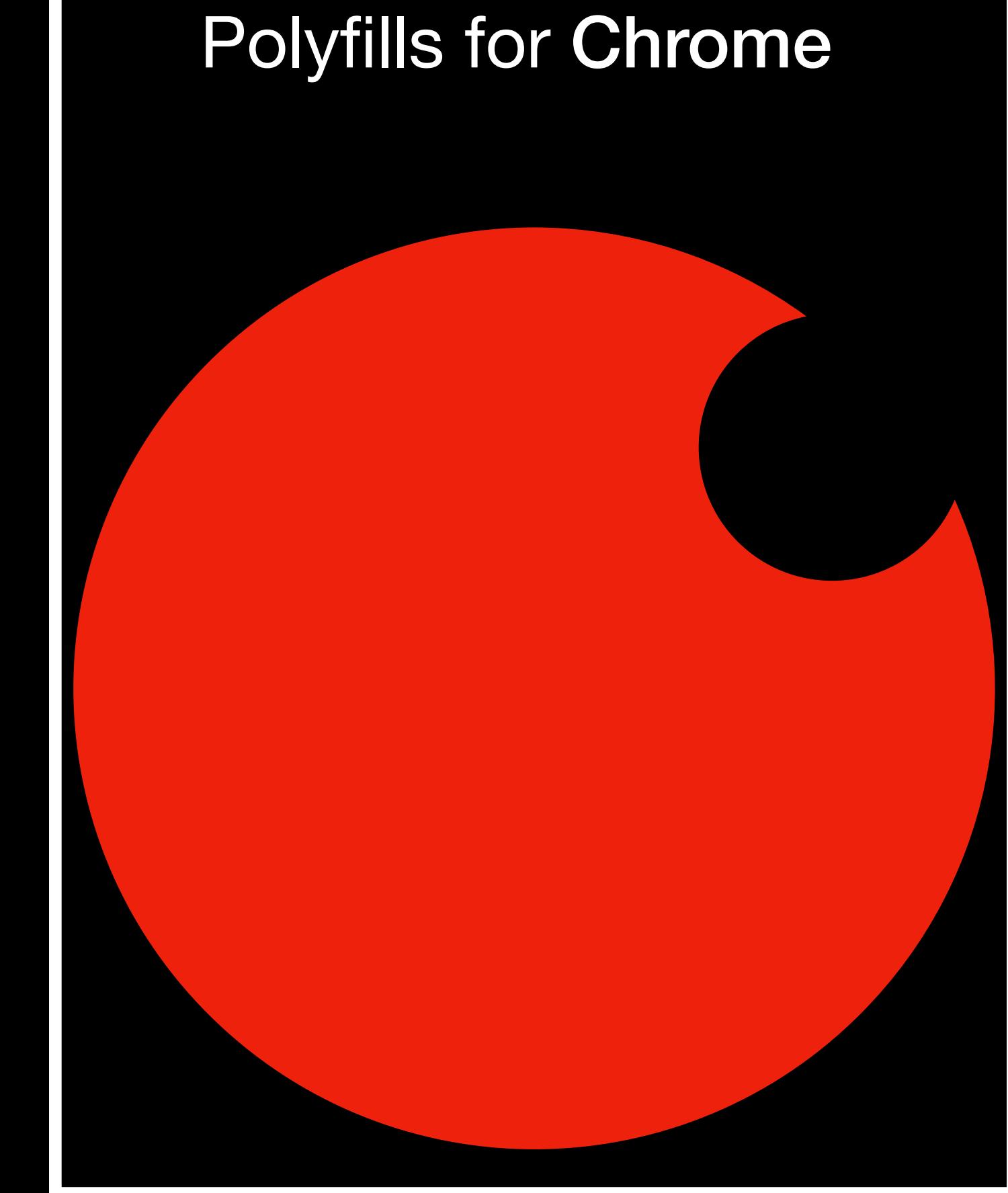
Polyfills for IE11



Polyfills for Firefox

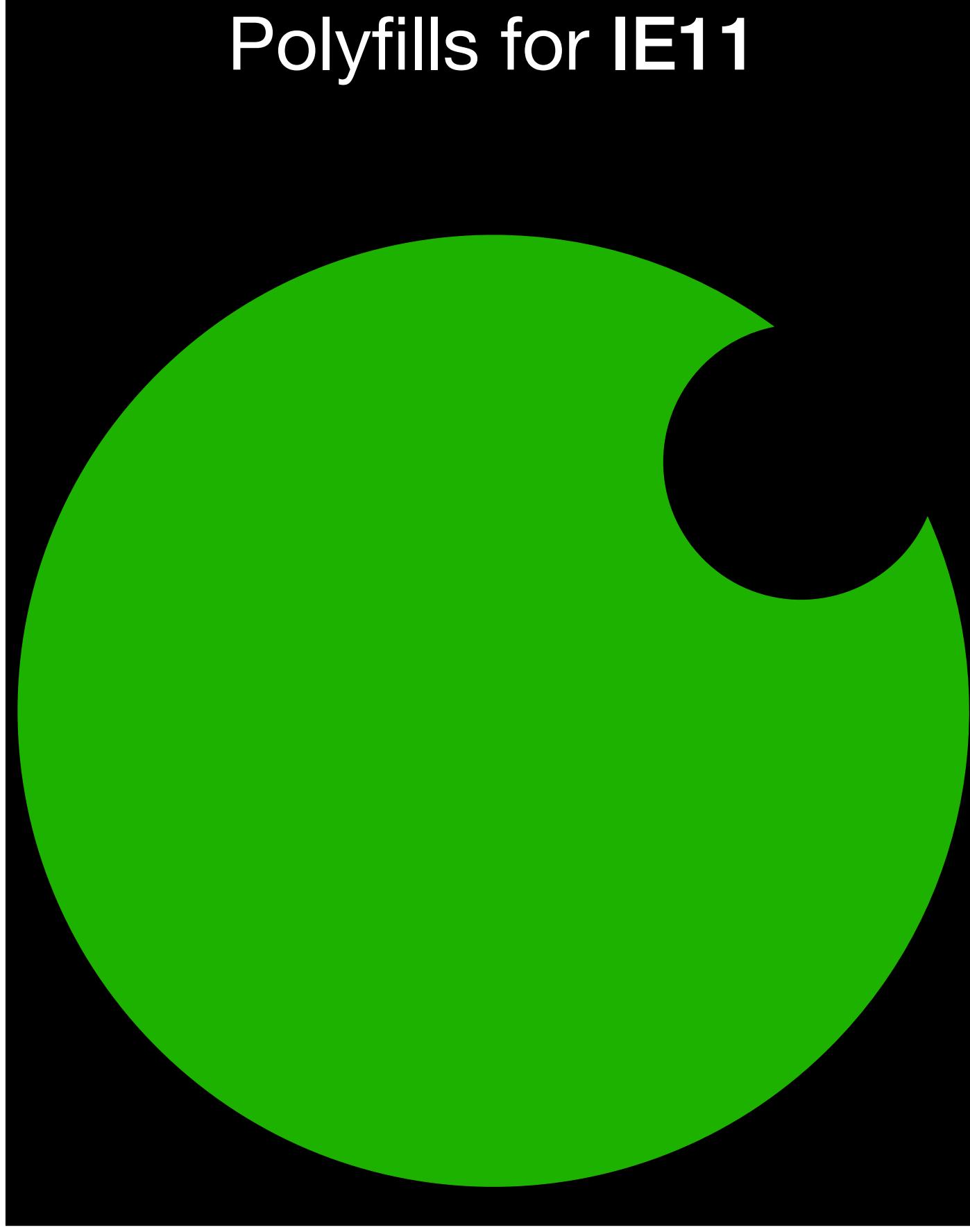


Polyfills for Chrome

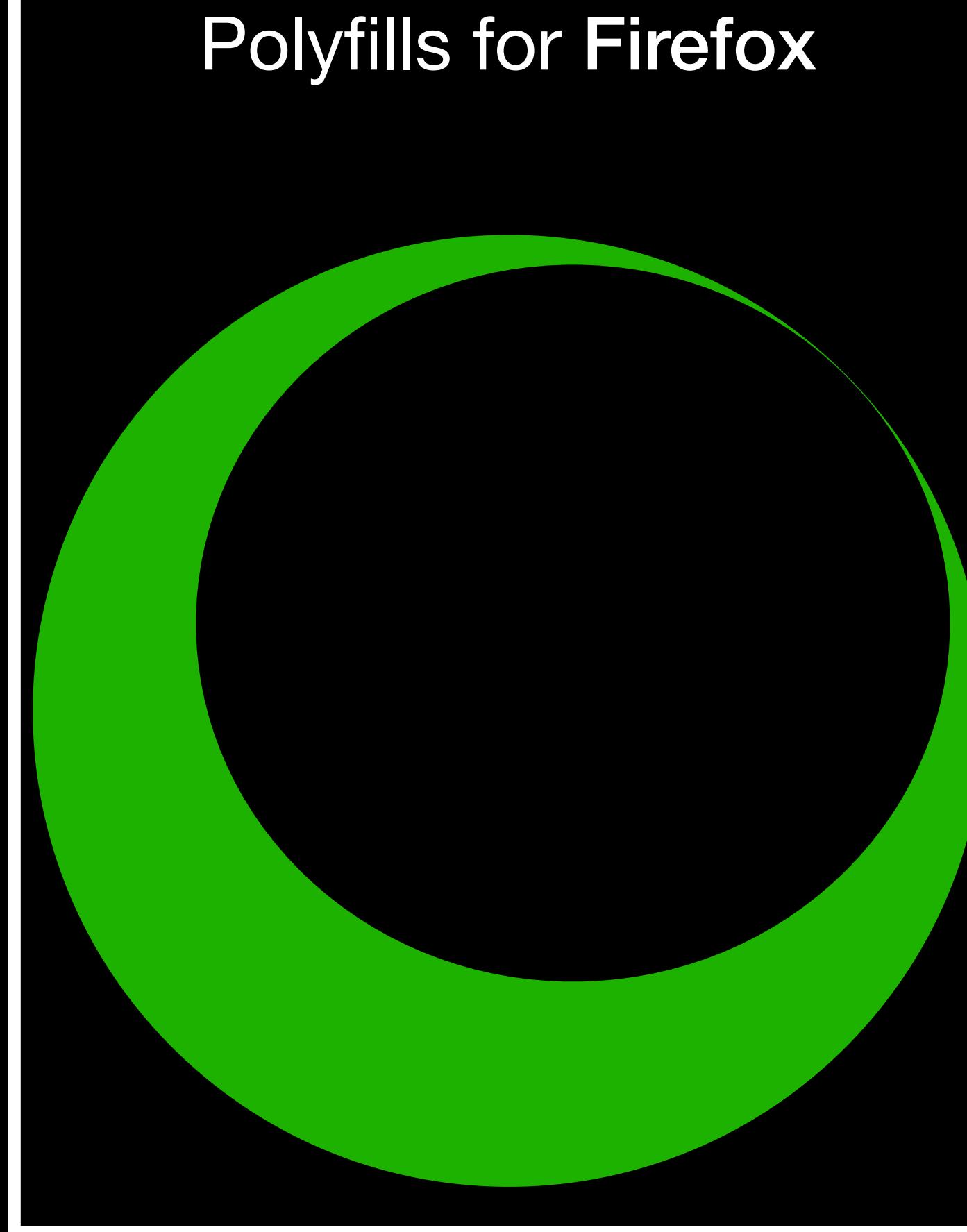


#2: Polyfills on demand

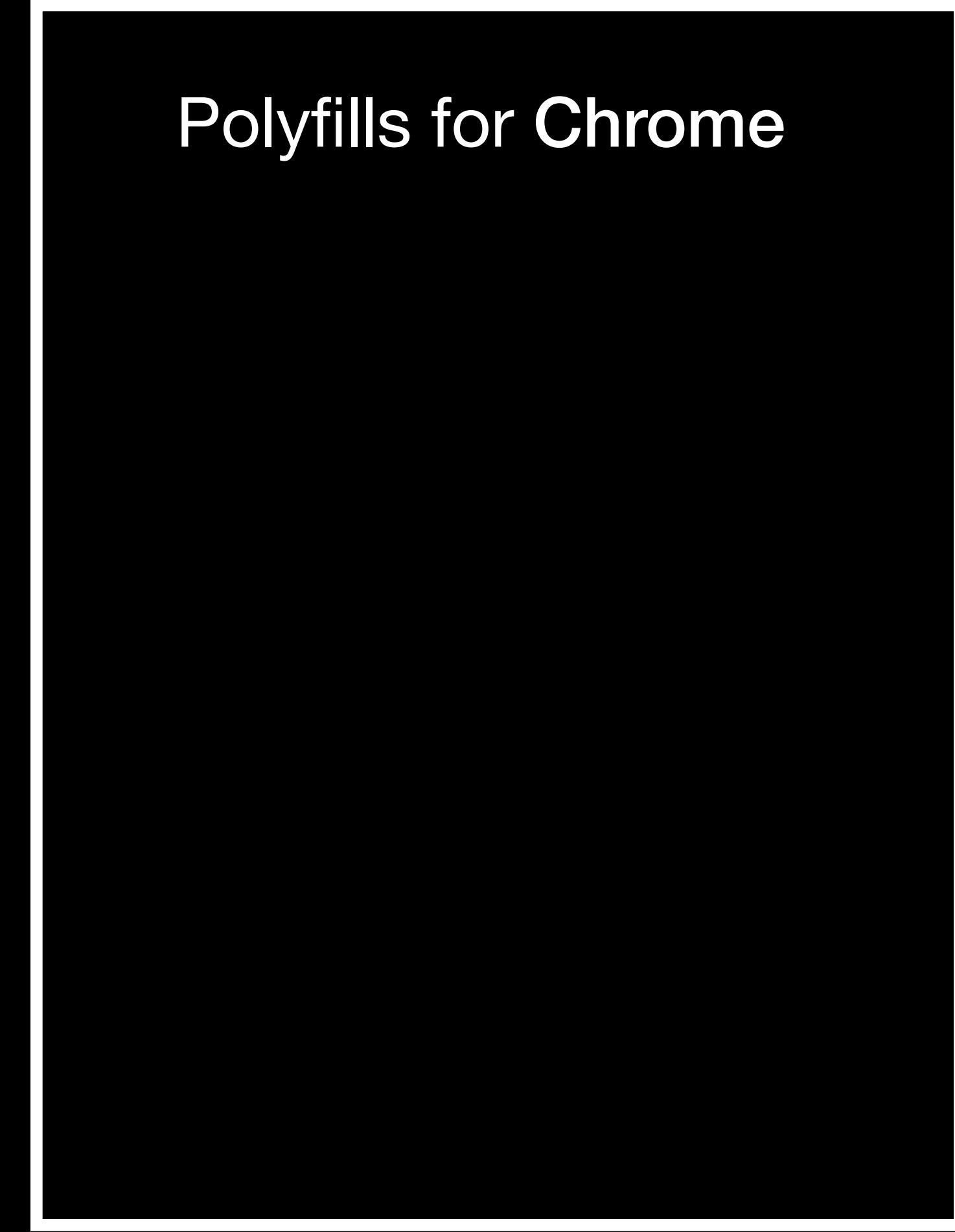
Polyfills for IE11



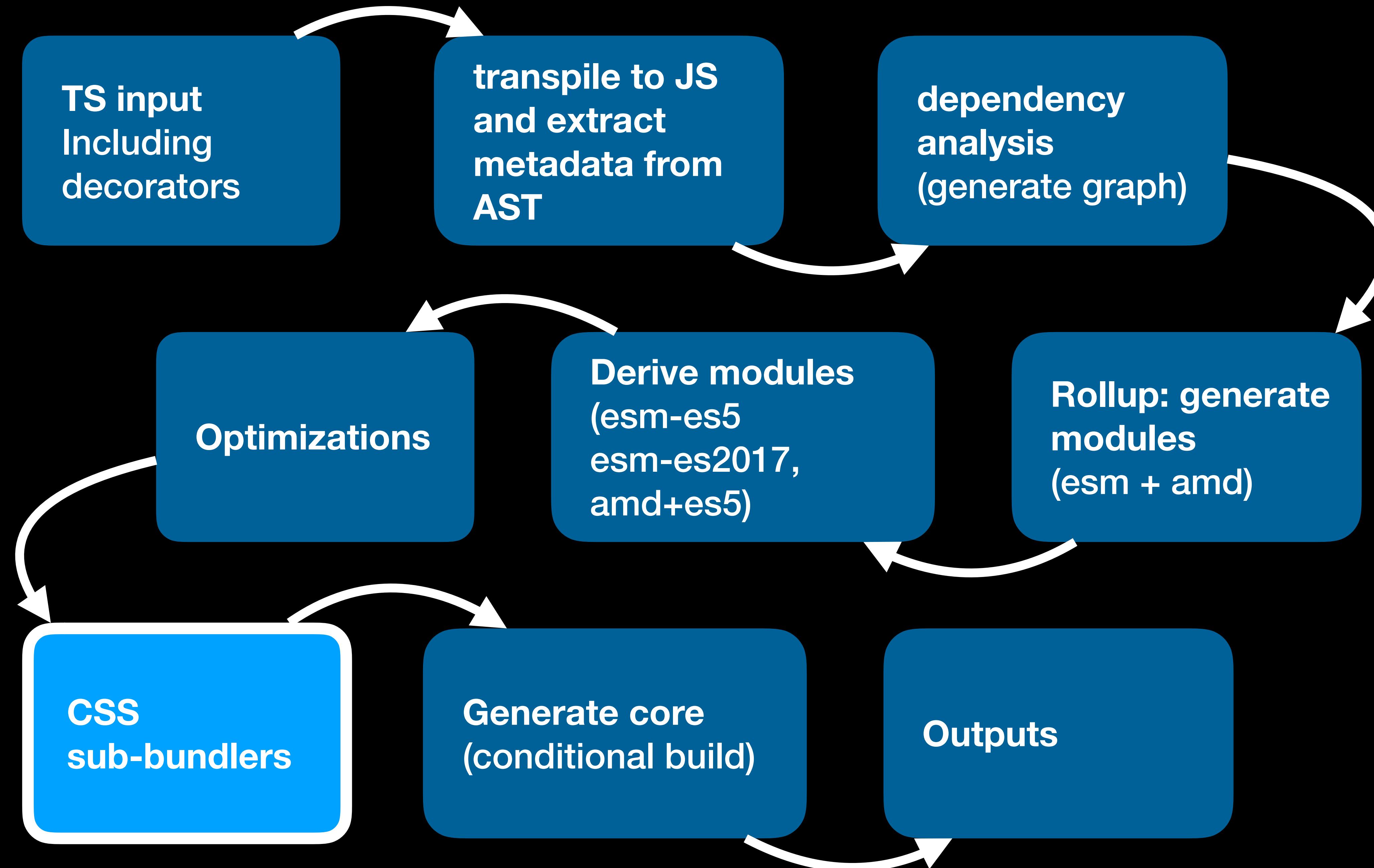
Polyfills for Firefox



Polyfills for Chrome



DEMO



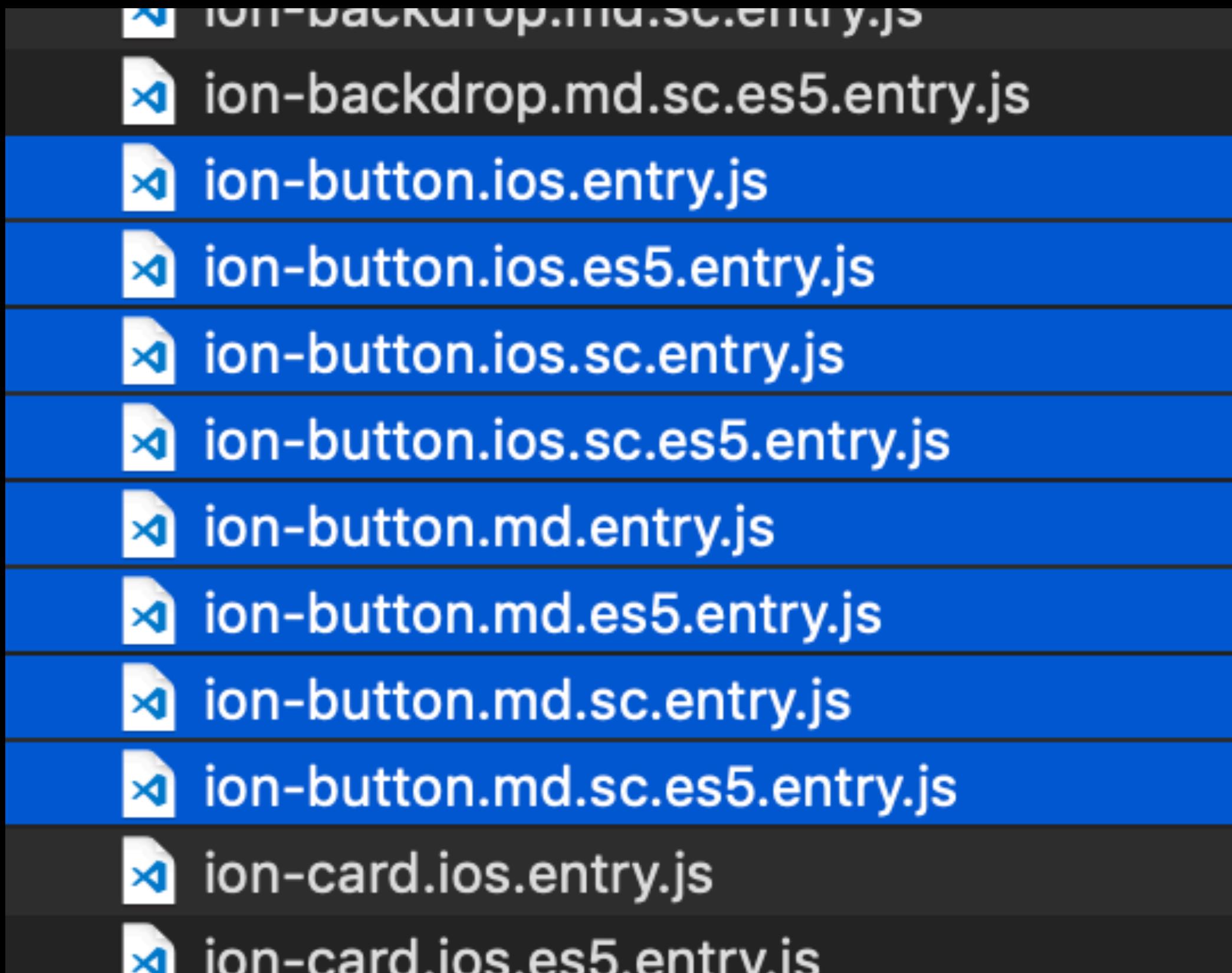
CSS submodules

```
] @Component({  
  tag: 'ion-button',  
  styleUrls: {  
    ios: 'button.ios.scss',  
    md: 'button.md.scss'  
  },  
  shadow: true,  
)  
export class Button implements
```

Different styles for
the same component



```
<ion-button mode="md">Material Design Button</ion-button>  
<ion-button mode="ios">iOS Button</ion-button>
```



- Build with iOS and MD styles
- Build in ESM (modern)
- Build in ES5 (legacy, IE11)
- Build SC: browser that doesn't support native shadow-dom

CSS autoprefixing

Original CSS

```
● ● ●  
.example {  
  display: flex;  
  align-items: centered;  
  user-select: none;  
}
```

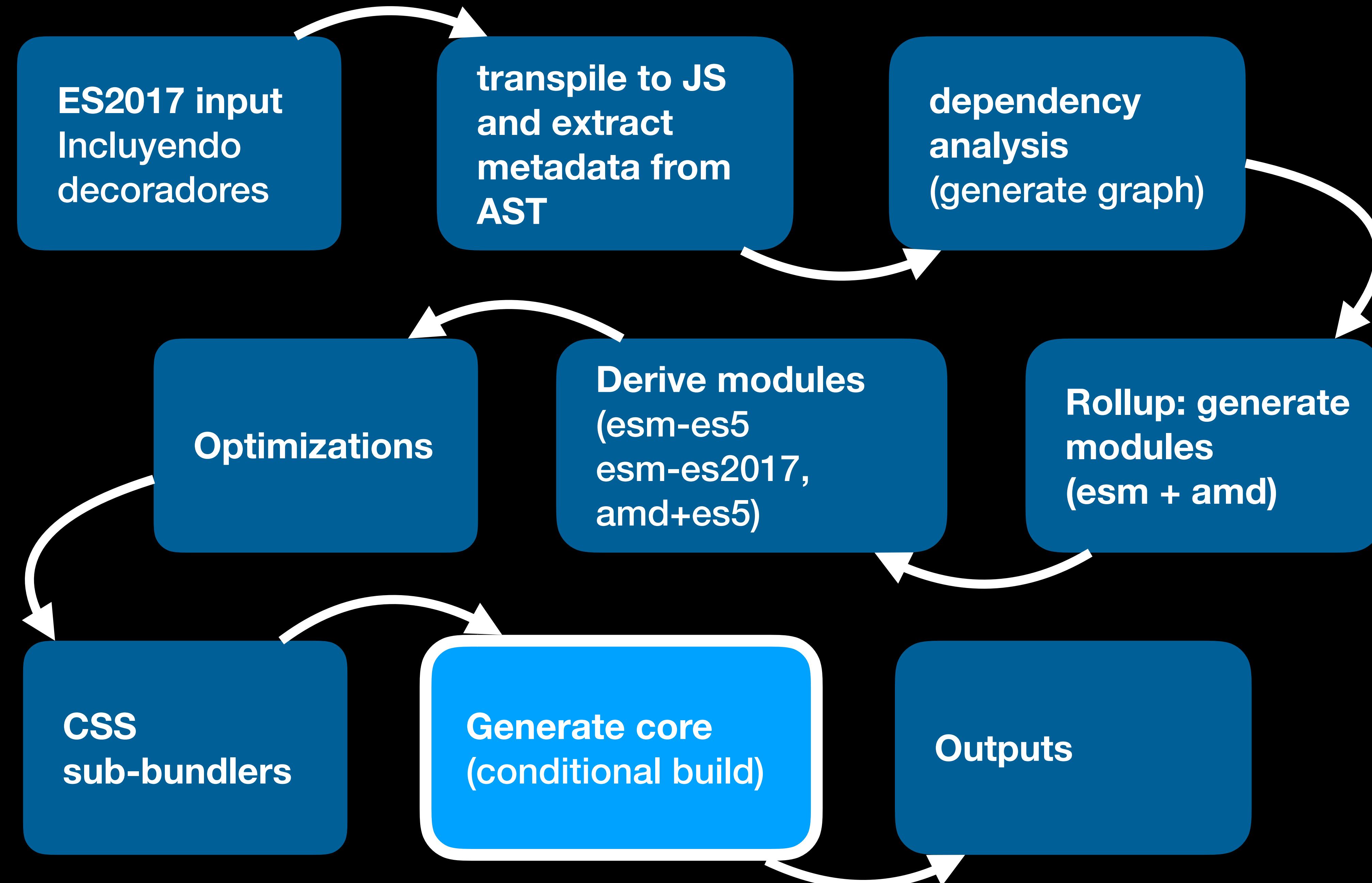
ES5 - legacy browsers

```
● ● ●  
.example {  
  display: -webkit-box;  
  display: -webkit-flex;  
  display: -moz-box;  
  display: -ms-flexbox;  
  display: flex;  
  -webkit-box-align: centered;  
  -webkit-align-items: centered;  
  -moz-box-align: centered;  
  -ms-flex-align: centered;  
  align-items: centered;  
  -webkit-user-select: none;  
  -moz-user-select: none;  
  -ms-user-select: none;  
  user-select: none;  
}
```

ESM - modern browsers

Firefox, Chrome, Safari

```
● ● ●  
.example {  
  display: flex;  
  align-items: centered;  
  user-select: none;  
}
```



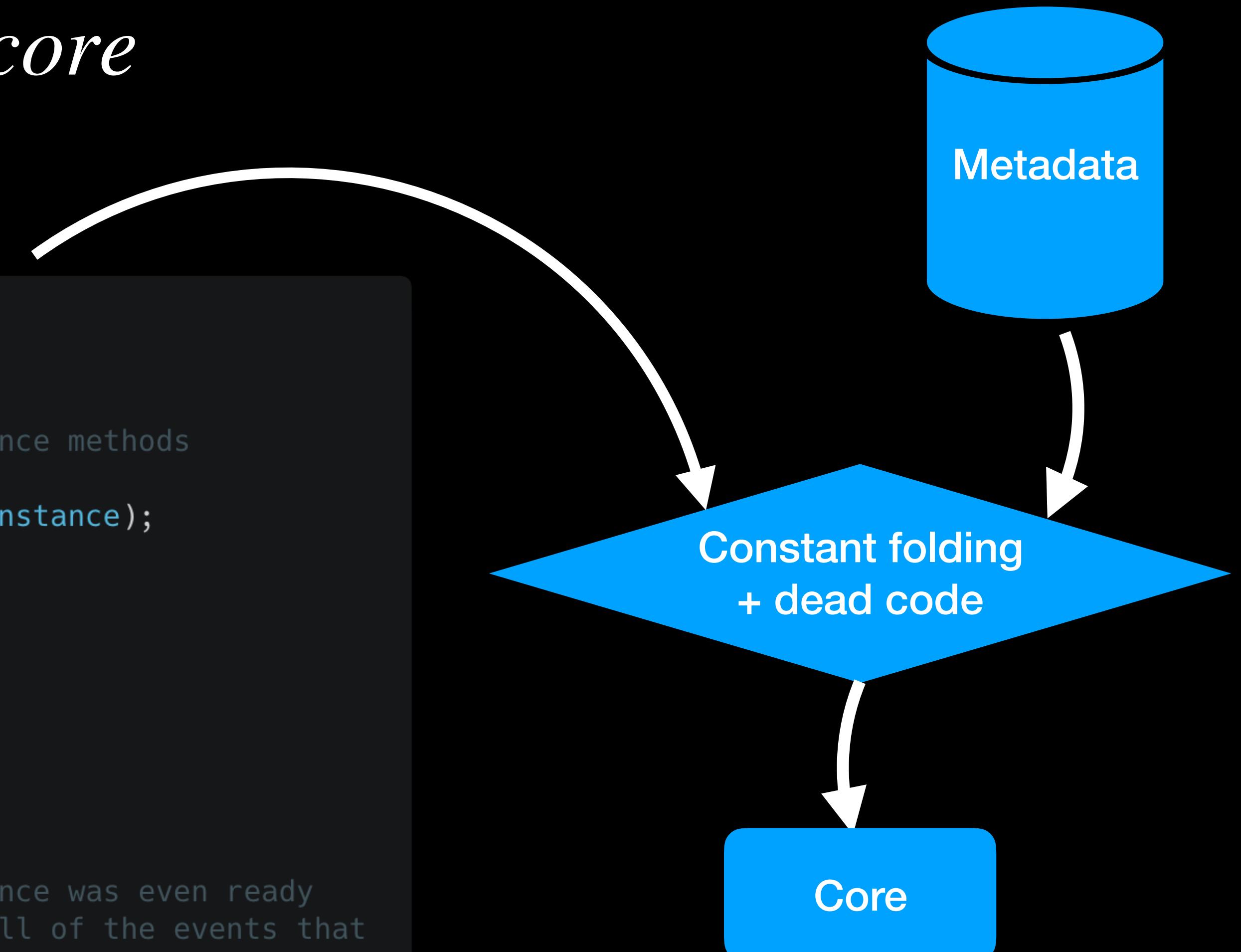
conditional builds

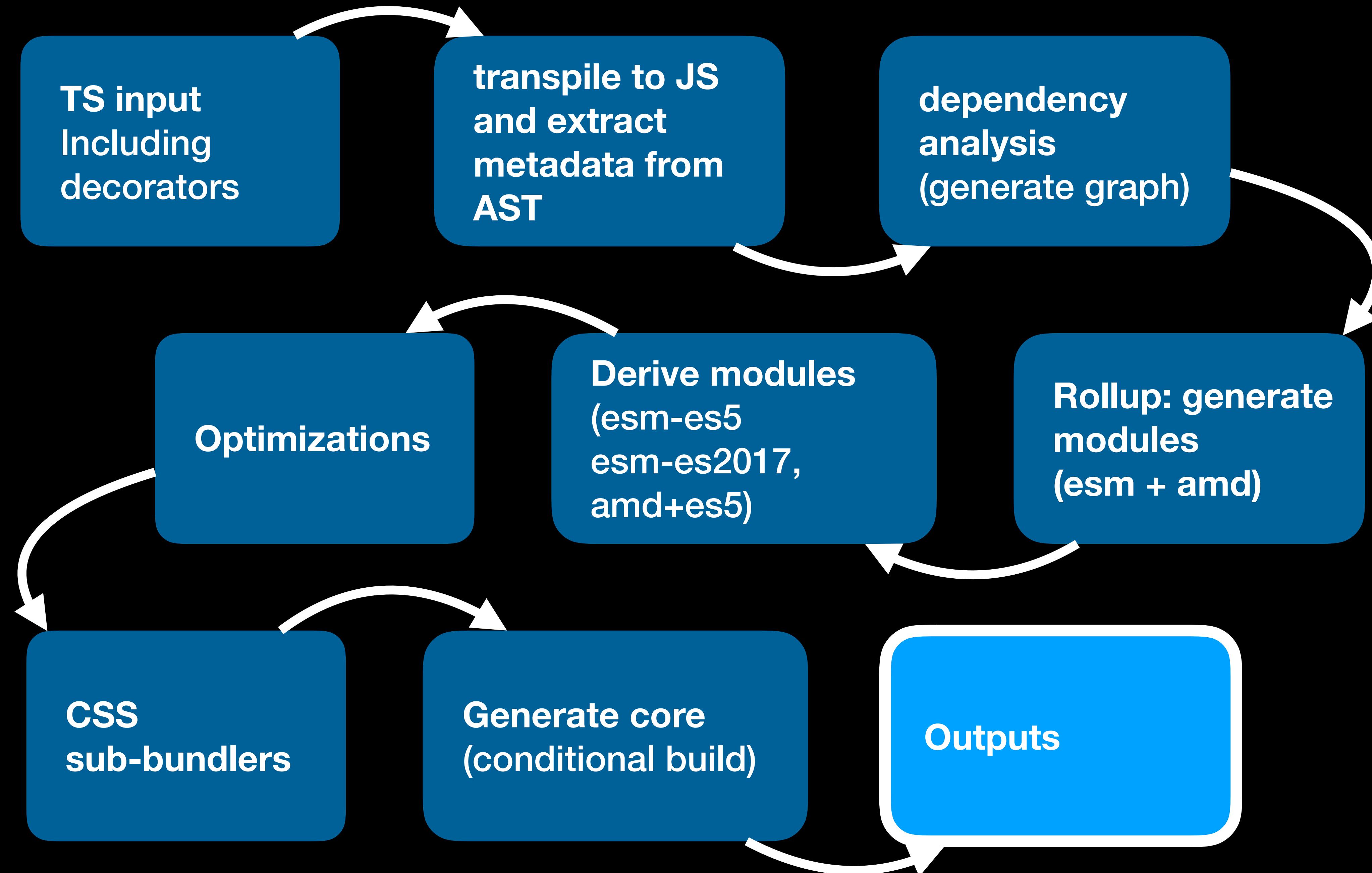
$f(components, jsV, type) \Rightarrow core$



A screenshot of a code editor showing a file with conditional build logic. The code uses the `__BUILD_CONDITIONALS__` object to conditionally execute blocks of code. Two specific blocks are highlighted with white boxes:

```
if (__BUILD_CONDITIONALS__.event) {  
    // add each of the event emitters which wire up instance methods  
    // to fire off dom events from the host element  
    initEventEmitters(plt, componentConstructor.events, instance);  
}  
  
if (__BUILD_CONDITIONALS__.listener) {  
    try {  
        // replay any event listeners on the instance that  
        // were queued up between the time the element was  
        // connected and before the instance was ready  
        queuedEvents = plt.queuedEvents.get(elm);  
        if (queuedEvents) {  
            // events may have already fired before the instance was even ready  
            // now that the instance is ready, let's replay all of the events that  
            // we queued up earlier that were originally meant for the instance  
            for (i = 0; i < queuedEvents.length; i += 2) {  
                // data was added in sets of two  
                // first item the eventMethodName
```





framework bindings



```
export const config = {
  outputTargets: [
    {
      type: 'angular',
      componentCorePackage: '@ionic/core',
      directivesProxyFile: '../angular/src/directives/proxies.ts',
    }
  ]
};
```

framework bindings

```
import { Component, ViewChild } from '@angular/core';
import { Slides } from '@ionic/angular';

@Component({
  selector: 'page-tutorial',
  templateUrl: 'tutorial.html',
})
export class TutorialPage {

  @ViewChild(Sli
    (property) Components.IonSlides['slideNext']: (speed?: number, runCallbacks?: boolean) => Promise<void>
  slideNext() {
    Transition to the next slide.
    this.slides.slideNext();
  }
}
```

Docs generation

The screenshot shows a dark-themed application window titled "Ionic Docs". Inside, there's a code editor with the following configuration:

```
export const config = {  
  outputTargets: [  
    {  
      type: 'docs',  
    }  
  ]  
};
```

Below the code editor, the word "ion-radio" is highlighted in bold. The page content includes:

- CONTENTS**
 - Usage
 - Properties
 - Events
 - CSS Custom Properties
- Usage**

Radios are generally used as a set of related options inside of a group, but they can also be used alone. Pressing on a radio will check it. They can also be checked programmatically by setting the `checked` property.

An `ion-radio-group` can be used to group a set of radios. When radios are inside of a `radio group`, only one radio in the group will be checked at any time. Pressing a radio will check it and uncheck the previously selected radio, if there is one. If a radio is not in a group with another radio, then both radios will have the ability to be checked at the same time.
- Properties**

A screenshot of a mobile application interface titled "Radio" is shown. It displays two groups of radio buttons:
 - Fruits (Group w/ values)**: Options include "Apple" (selected), "Grape, checked, disabled" (disabled), and "Cherry".
 - Extra Pizza Topping (Group w/ no values)**: Options include "Pepperoni" (selected), "Sausage", and "Veggies (Group w/ allow empty)".At the bottom, there are buttons for "Checked", "Disabled", "Print", and "Color". Below the interface, the "Values:" section lists:

```
fruitRadio: grape  
pizzaRadio: ion-rb-14  
veggiesRadio: broccoli
```

The screenshot shows a documentation page for the "ion-radio" component. At the top, the file name "readme.md" is visible. The main content starts with the heading "ion-radio".

ion-radio

Radios are generally used as a set of related options inside of a group, but they can also be used alone. Pressing on a radio will check it. They can also be checked programmatically by setting the `checked` property.

An `ion-radio-group` can be used to group a set of radios. When radios are inside of a `radio group`, only one radio in the group will be checked at any time. Pressing a radio will check it and uncheck the previously selected radio, if there is one. If a radio is not in a group with another radio, then both radios will have the ability to be checked at the same time.

Properties

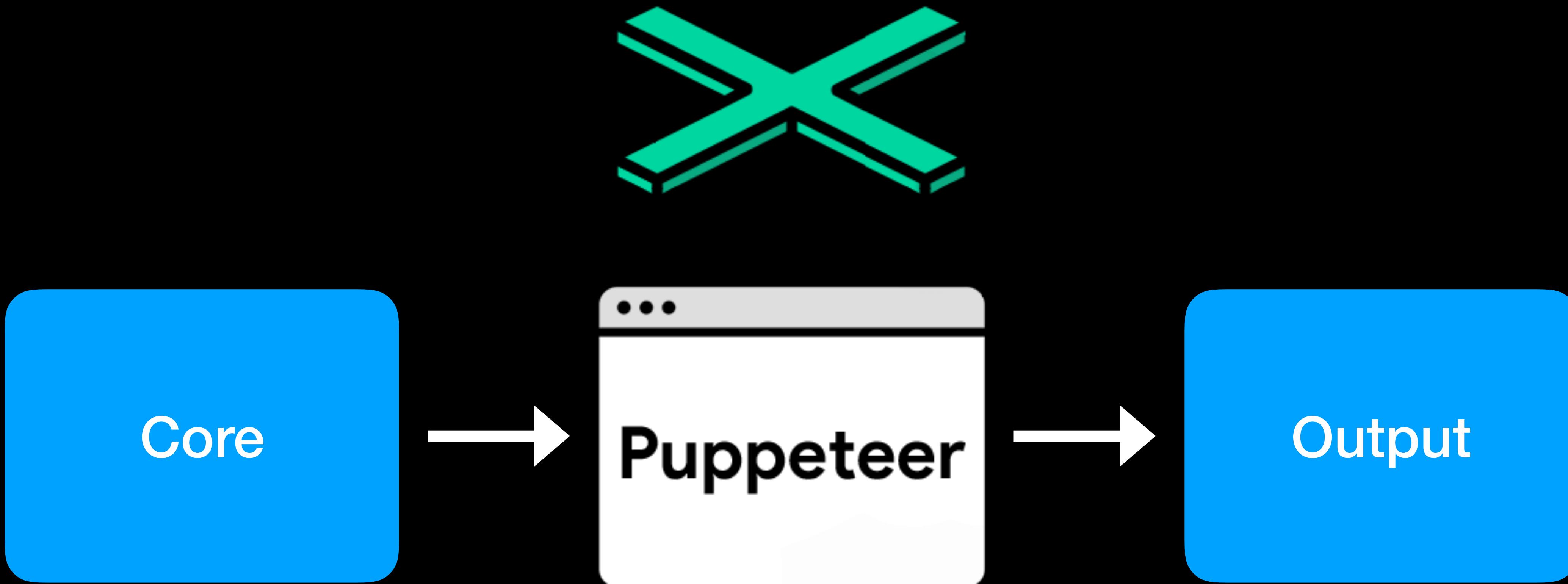
Property	Attribute	Description
<code>checked</code>	<code>checked</code>	If <code>true</code> , the radio is selected. Defaults to <code>false</code> .
<code>color</code>	<code>color</code>	The color to use from your application's color palette. Default options are: "primary", "secondary", "tertiary", "success", "warning", "danger", "light", "medium", and "dark". For more information on colors, see theming .
<code>disabled</code>	<code>disabled</code>	If <code>true</code> , the user cannot interact with the radio. Defaults to <code>false</code> .
<code>mode</code>	<code>mode</code>	The mode determines which platform styles to use. Possible values are: "ios" or "md".
<code>name</code>	<code>name</code>	The name of the control, which is submitted with the form data.
<code>value</code>	--	the value of the radio.

Events

Bonus: CSS variables

Name	Description
--background	Background of the button
--background-activated	Background of the button when activated
--background-focused	Background of the button when focused
--border-color	Border color of the button
--border-radius	Border radius of the button
--border-style	Border style of the button
--border-width	Border width of the button
--box-shadow	Box shadow of the button
--color	Text color of the button

SSR and Prerendering



DEMO

Thank You

@manucorporat

