

# Build-time Optimizations in Frontend Engineering

Evan You  
JSConf China 2017

Frontend used to have no build steps...



# Today



Compilation  
Infrastructure



Module  
Build Systems



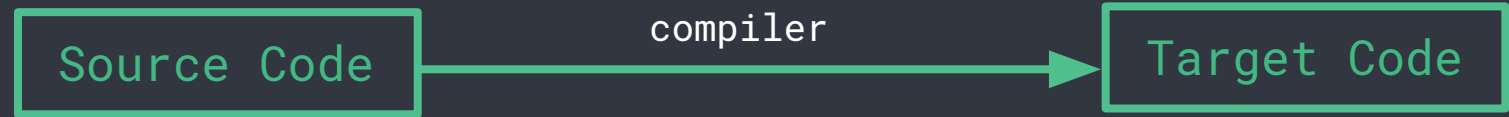
Compile-to-JS  
Languages

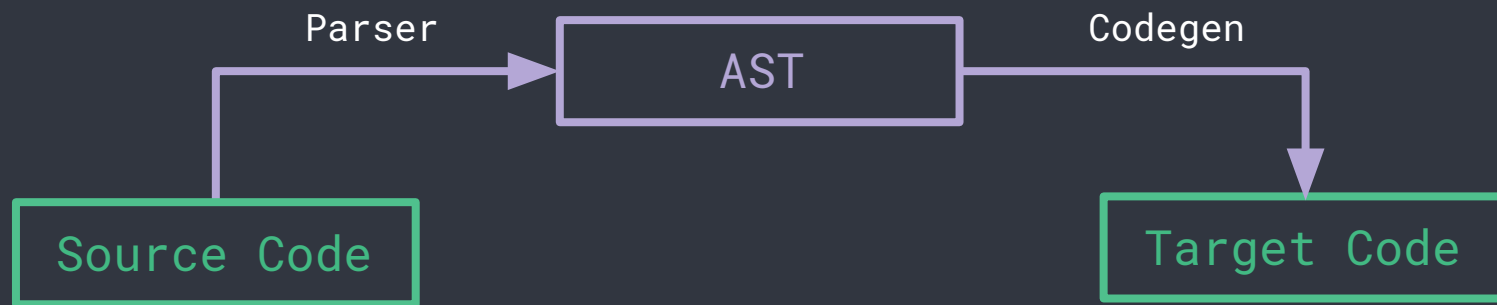


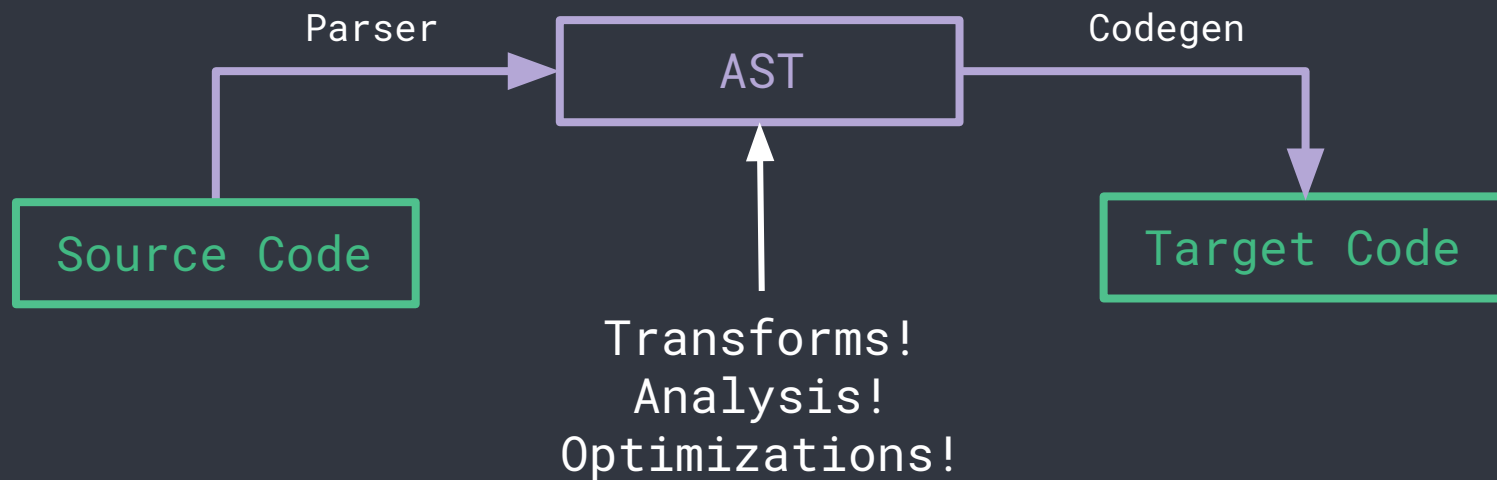
CSS Processors

Make Code Smaller

Minifiers: essentially Compilers









Closure Compiler

UglifyJS

Babili

Butternut

Says it's a compiler  
in its name



Closure Compiler

UglifyJS



Implements its own  
parser / AST /  
codegen

Built on top of Babel



Babili

Butternut



Similar architecture  
to Buble (a  
lightweight ES2015  
compiler)

Early days: concat + minify

Early days: concat + minify  
Problem: global scope sucks

Bundlers: let's use modules

Bundlers: let's use modules

Problem: modules make things harder to minify

Each module is wrapped inside a separate function scope

```
registerModules([  
  function (module, exports) {  
    // module 1  
  },  
  function (module, exports) {  
    // module 2  
  },  
  // ...  
)
```



Rollup: use ES modules!



# Module Scope Hoisting

main.js

```
import { foo } from './foo.js'  
import { bar } from './bar.js'  
  
foo(bar)
```

foo.js

```
export function foo () {  
  // ...  
}
```

bar.js

```
export const bar = 123
```

# Module Scope Hoisting

bundle.js

```
// foo.js
function foo () {
  // ...
}

// bar.js
const bar = 123

// main.js
foo(bar)
```

# Treeshaking

bundle.js

```
// foo.js
function foo () {
  // ...
}

// bar.js
const bar = 123

// main.js
// foo(bar)
```

# Treeshaking

bundle.js

```
// foo.js  
function foo () {  
  // ...  
}
```

← unused

```
// bar.js  
const bar = 123
```

← unused

```
// main.js  
// foo(bar)
```

# Treeshaking

bundle.js

```
// nothing left!
```



# Treeshaking



Now also in webpack 3.x via  
`webpack.optimize.ModuleConcatenationPlugin`

# Conditional Block Trick

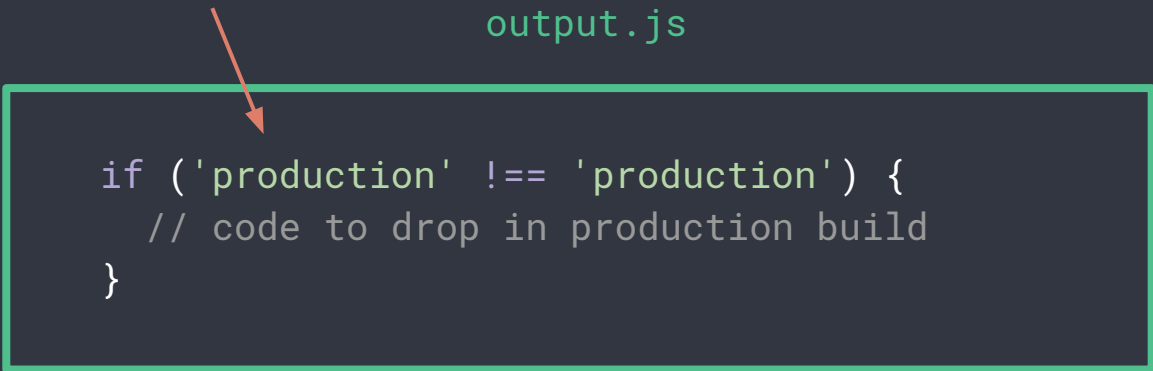
source.js

```
if (process.env.NODE_ENV !== 'production') {  
  // code to drop in production build  
}
```

# Conditional Block Trick

Replaced during build

output.js



```
if ('production' !== 'production') {  
  // code to drop in production build  
}
```



# Conditional Block Trick

output.js

```
if (false) {  
  // unreachable  
}
```

# Conditional Block Trick

output.min.js

```
// nothing left!
```

Make Code Faster

# AOT vs. JIT

Do more at build time

Do less at runtime

# Angular / Vue / Glimmer

Pre-compile templates to JavaScript  
to avoid runtime compilation cost

# React

Optimization via Babel plugins

<https://github.com/thejameskyle/babel-react-optimize>

## Hoisting Static Elements

input

```
class MyComponent extends React.Component {  
  render() {  
    return (  
      <div className={this.props.className}>  
        <span>Hello World</span>  
      </div>  
    );  
  }  
}
```



output

```
var _ref = <span>Hello World</span>;  
  
class MyComponent extends React.Component {  
  render() {  
    return (  
      <div className={this.props.className}>  
        {_ref}  
      </div>  
    );  
  }  
}
```

# Svelte

Compile everything to vanilla JS with no runtime lib



# Initial Render

template

```
<h1>Hello {{name}}!</h1>
```

output

```
// only showing initial render code  
h1 = createElement( 'h1' );  
text = createText(  
  text_value = state.msg  
);  
  
insertNode( h1, target, anchor );  
appendNode( text, h1 );
```

# Updates

template

```
<h1>Hello {{name}}!</h1>
```

output

```
// only showing update code  
if (text_value !== (text_value =  
state.msg)) {  
  text.data = text_value;  
}
```

# Relay Modern

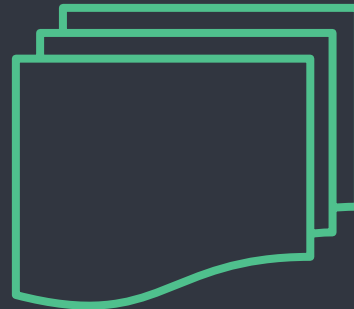
Pre-Compile GraphQL Queries & Schemas

Getting rid of expensive runtime query  
construction via static build step

GraphQL Query

```
graphql`  
  fragment MyComponent on Type {  
    field  
  }  
`
```

Runtime Artifacts & Types



# Prepack

Optimize performance via partial evaluation

## Partial Evaluation (moving more computation to build time)

input

```
(function () {  
  function fib(x) {  
    return x <= 1  
      ? x  
      : fib(x - 1) + fib(x - 2);  
  }  
  global.x = fib(23);  
})();
```



output

```
(function () {  
  x = 28657;  
})();
```

# Rakt

Application-level optimizations via compilation  
(proof of concept)

# Compile-time Optimizations in Vue



# Hoisting Static Trees

template

```
<div>
  <p class="foo">
    this is static
  </p>
</div>
```



output

```
function render() {
  return this._renderStatic(0)
}
```

# Skipping Static Bindings

template

```
<div>
  <p class="foo">
    {{ msg }}
  </p>
</div>
```



output

```
return h("div", [
  h(
    "p",
    { staticClass: "foo" },
    [...]
  )
])
```

# Skipping Children Array Normalization

template

```
<ul>
  <li v-for="i in 10">
    {{ i }}
  </li>
</ul>
```



output

```
return h("ul", [
  renderList(10, i => {
    return h("li", i)
  })
], 0) // ← optimization hint
```

# SSR: optimizing Virtual DOM render functions into string concat

output

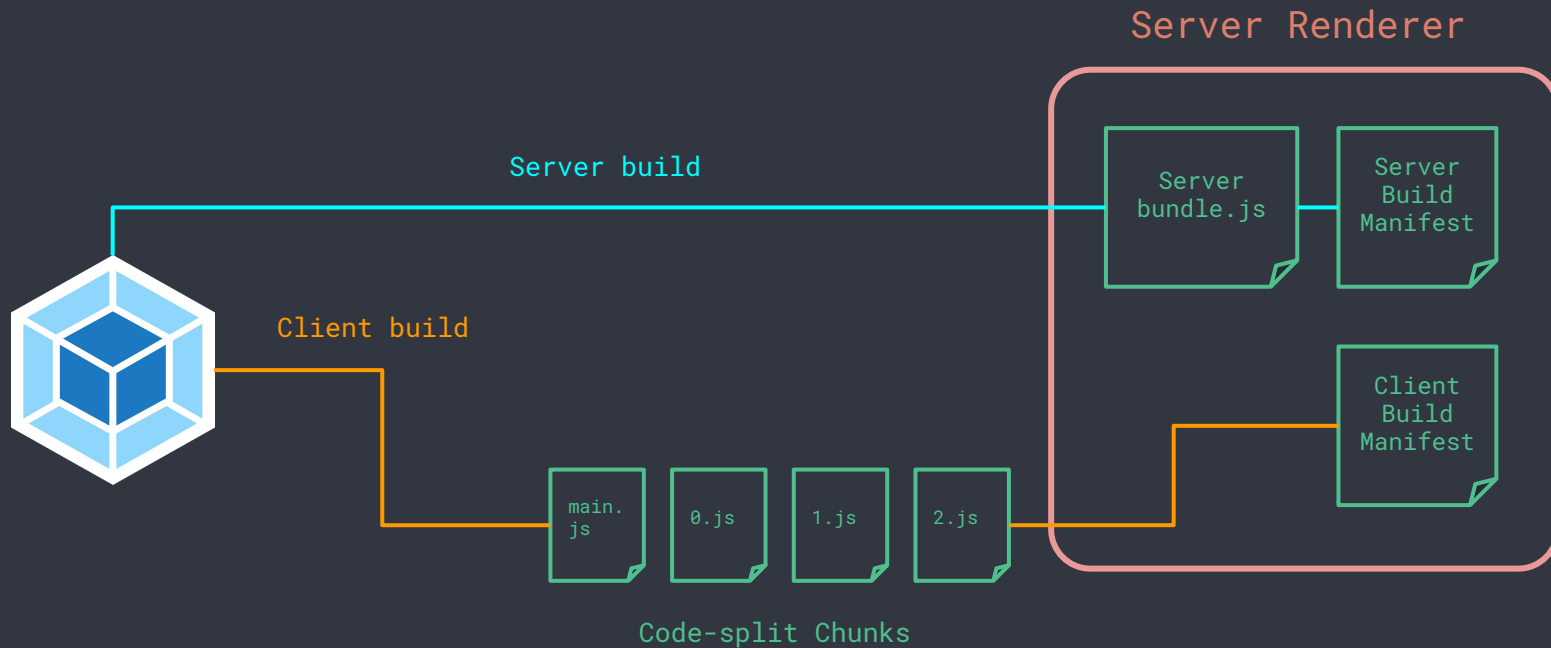
template

```
<div>
  <p class="foo">
    {{ msg }}
  </p>
</div>
```

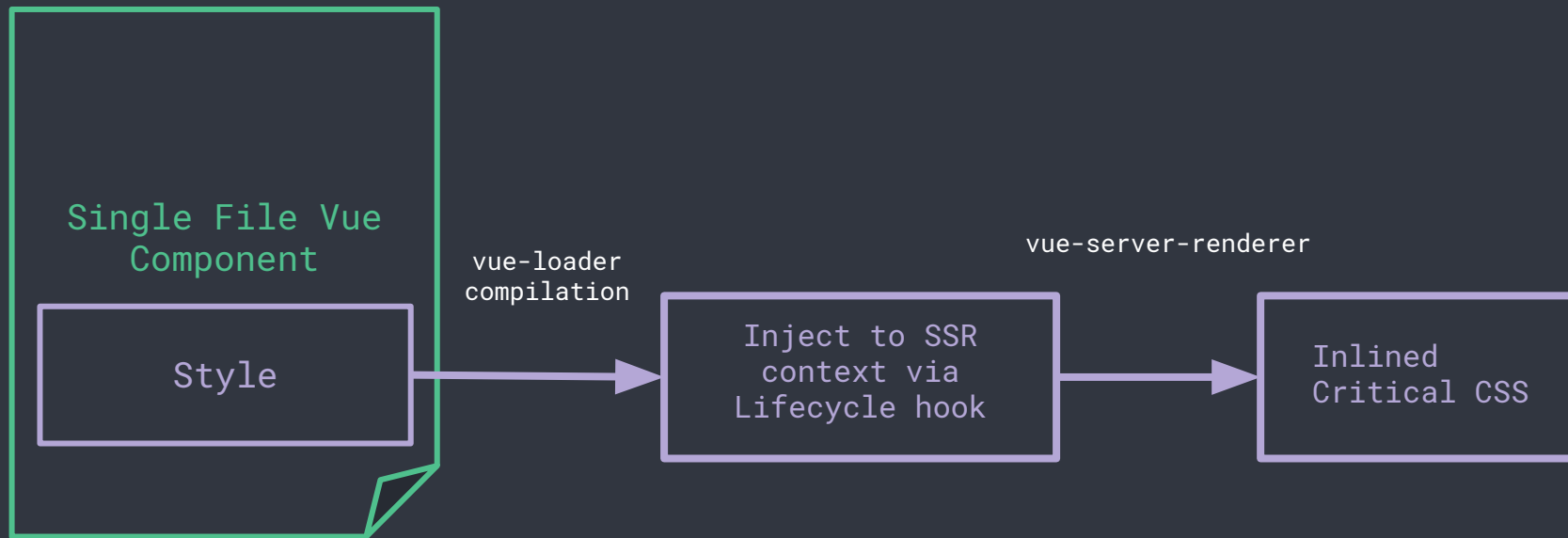


```
function render() {
  return h("div", [
    this._ssrString(
      "<p class=\"foo\">" +
      this.msg +
      "</p>"
    ),
    h("comp") // mix w/ vdom
  ])
}
```

# SSR: inferring async chunks



## SSR: inlining Critical CSS

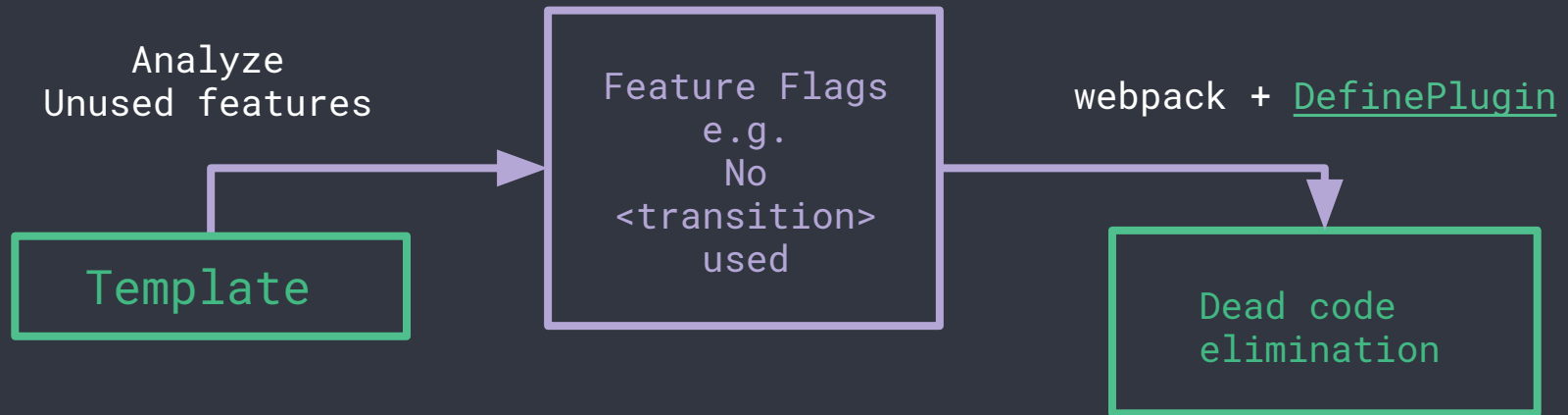


```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Vue HN 2.0 | Top</title>
    <meta charset="utf-8">
    <meta name="mobile-web-app-capable" content="yes">
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-
scalable=no, minimal-ui">
    <link rel="shortcut icon" sizes="48x48" href="/public/logo-48.png">
    <meta name="theme-color" content="#f60">
    <link rel="manifest" href="/manifest.json">
    <link rel="preload" href="/dist/manifest.1e547b25b2af27b910d8.js" as="script">
    <link rel="preload" href="/dist/vendor.e4d69e40768f3a87479f.js" as="script">
    <link rel="preload" href="/dist/app.dd8b3eb63d43f4647423.js" as="script">
    <link rel="preload" href="/dist/common.dd8b3eb63d43f4647423.css" as="style">
    <link rel="preload" href="/dist/1.482045dce837afbcc821.js" as="script"> Async chunks used in SSR
    <link rel="prefetch" href="/dist/0.f1fe6a270f9d53dc7cb2.js" as="script"> Unused async chunks
    <link rel="prefetch" href="/dist/2.560d8fe30d673faffdaa.js" as="script">
    <link rel="stylesheet" href="/dist/common.dd8b3eb63d43f4647423.css"> Extracted common CSS
    <style data-vue-ssr-id="2cf0ef4f:0 2f664187:0">...</style> Critical SSR CSS (inlined)
  </head>
  <body>
    <div id="app">...</div>
    <script>...</script>
    <script src="/dist/manifest.1e547b25b2af27b910d8.js"></script>
    ... <script src="/dist/1.482045dce837afbcc821.js"></script> == $0 Async chunks used in SSR
    <script src="/dist/vendor.e4d69e40768f3a87479f.js"></script>
    <script src="/dist/app.dd8b3eb63d43f4647423.js"></script>
    <div data-v-b9f0df9e class="progress" style="width: 0%; height: 2px; background-color: rgb(255,
202, 43); opacity: 0;"></div>
  </body>
</html>

```

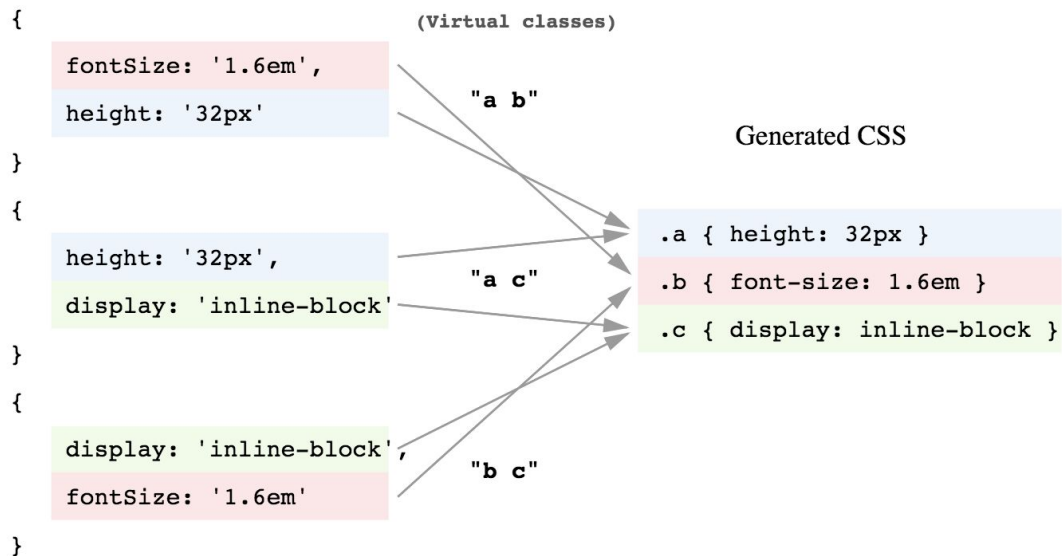
IDEA: compile away parts of Vue that's not used in your app





# IDEA: Styletron-style Atomic CSS generation at build time

Source style objects



The build step affords  
many more possibilities!

We've only scratched the surface, let's keep experimenting.