Глубокое погружение в webpack

- Меня зовут Стас Курилов
- Работаю в JetBrains
- Последние годы занимаюсь разработкой инструментов для фронтенда

Есть опыт написания тулов для разных экосистем

banner-rotator-webpack-plugin

docpack

extract-svg-sprite-webpack-plugin

generator-kisenka

gulp-svg-mixer

postcss-aspect-ratio-from-bg-image

postcss-move-props-to-bg-image-query

postcss-svg-mixer

posthtml-rename-id

posthtml-transform

postsvg

node-sass-importer-builder

svg-mixer

svg-transform-loader

svg-sprite-loader

webpack-toolkit

Больше всего опыта с

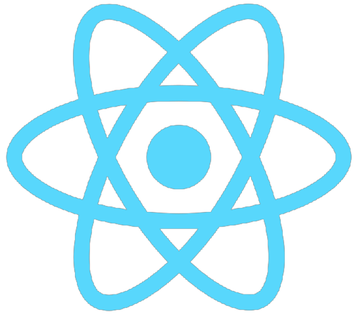
webpack

За что мы любим webpack?

За безграничные возможности!


BABEL


TS




Sass


less

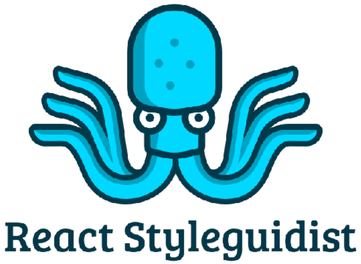

stylus


<%= EJS %>

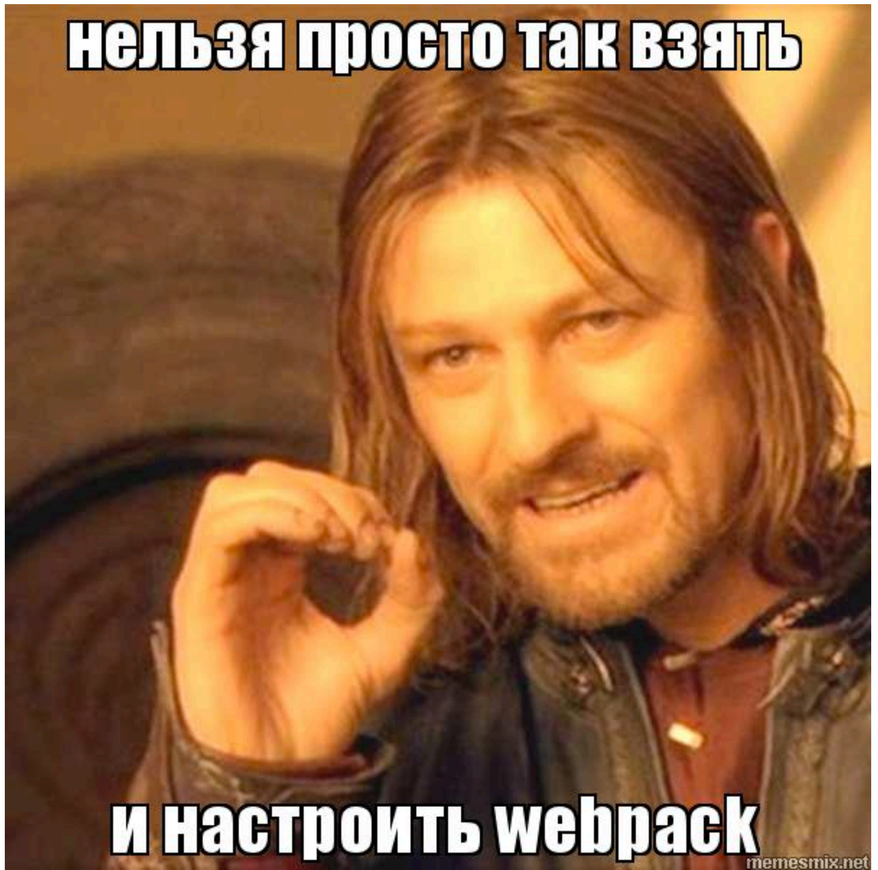

TWIG

Не просто сборщик

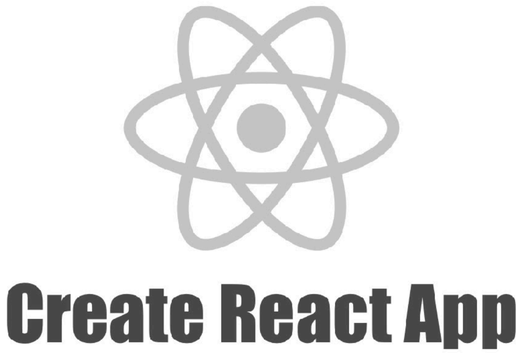
webpack – это платформа для написания инструментов

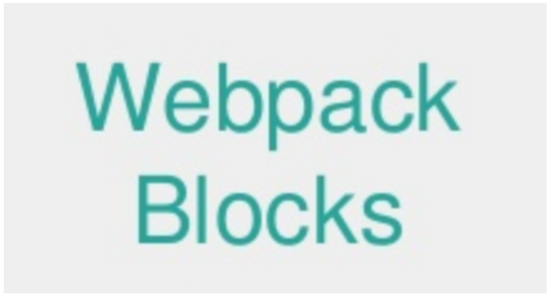


Но плата за безграничные возможности — сложность

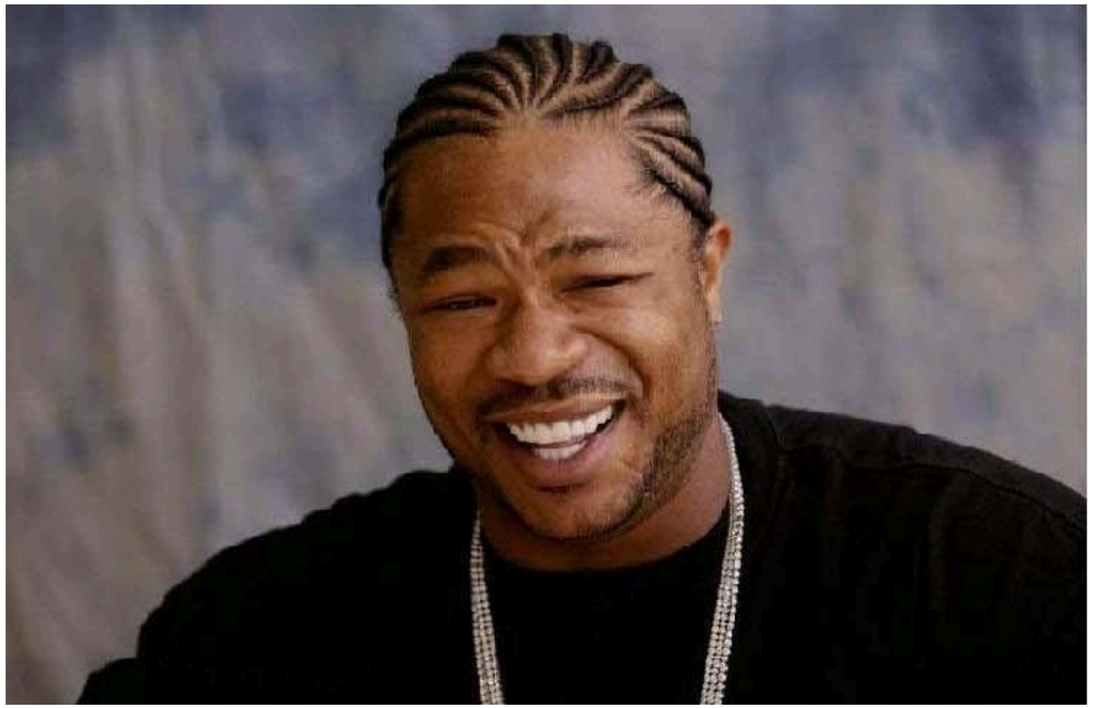


Чтобы настроить webpack есть отдельные инструменты





Чтобы настроить тул, ты ставишь другой тул, который настраивает первый



А всё потому что он сложный

Схема работы webpack от его автора Tobias Koppers

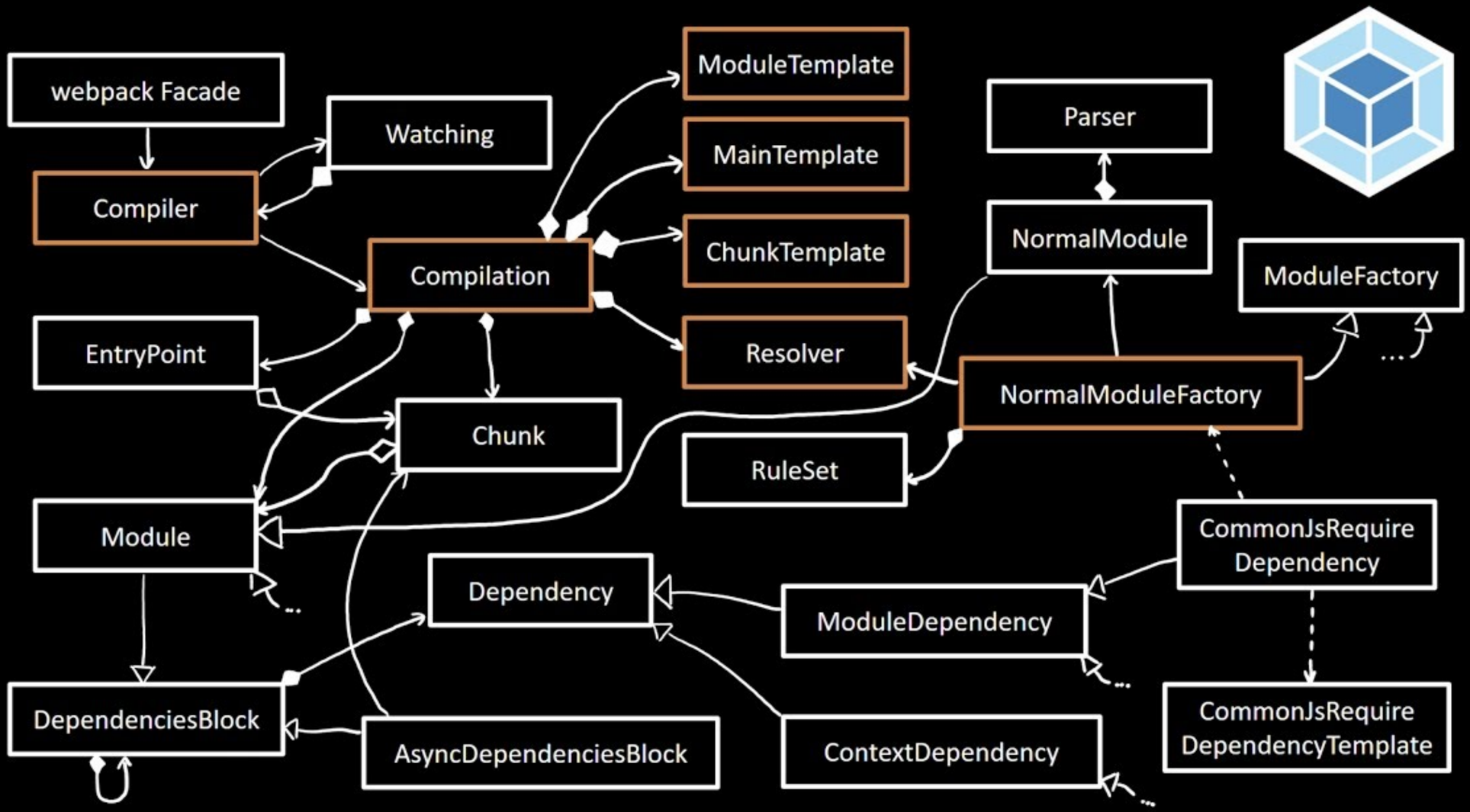
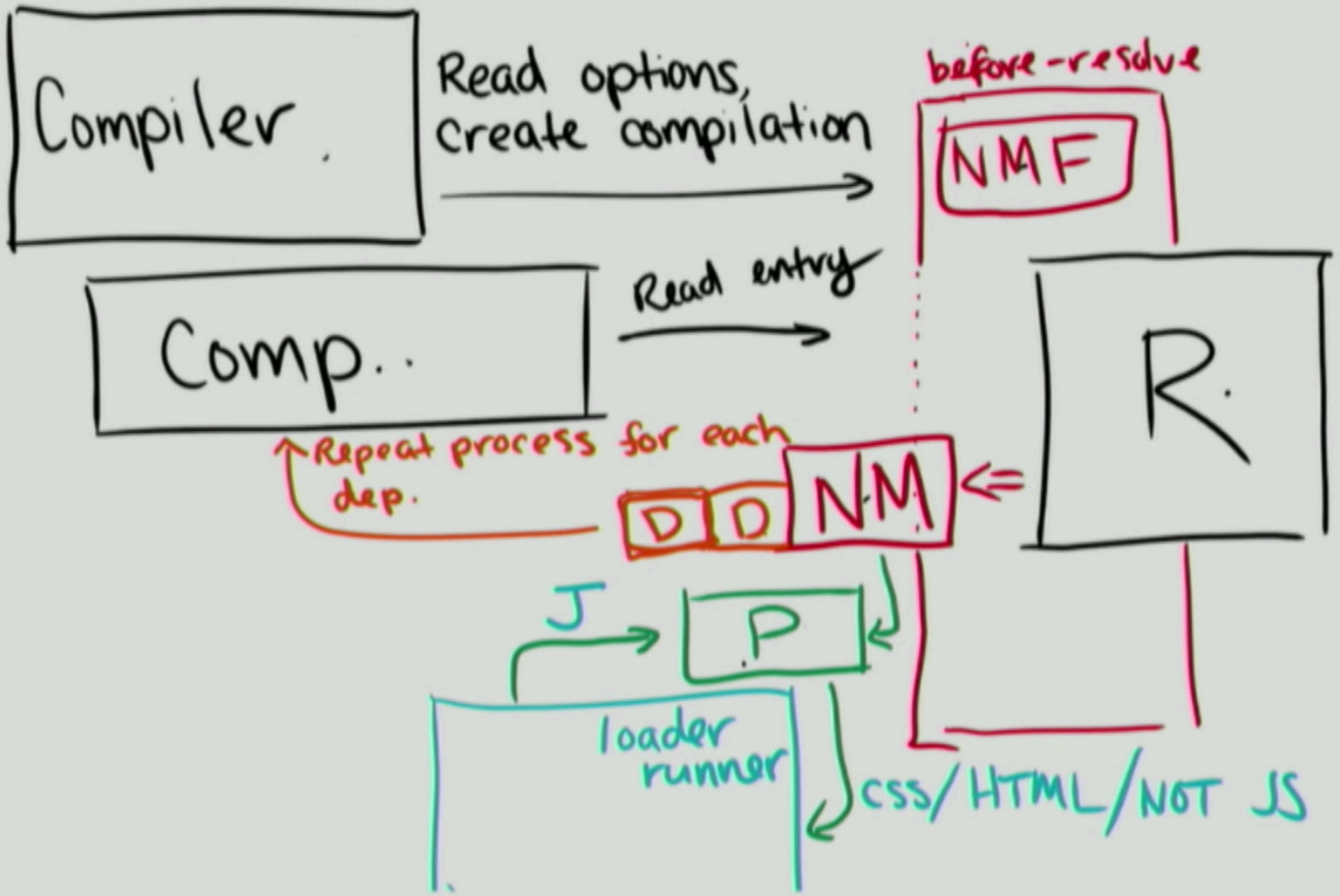


Схема работы webpack от евангелиста Sean Larkin

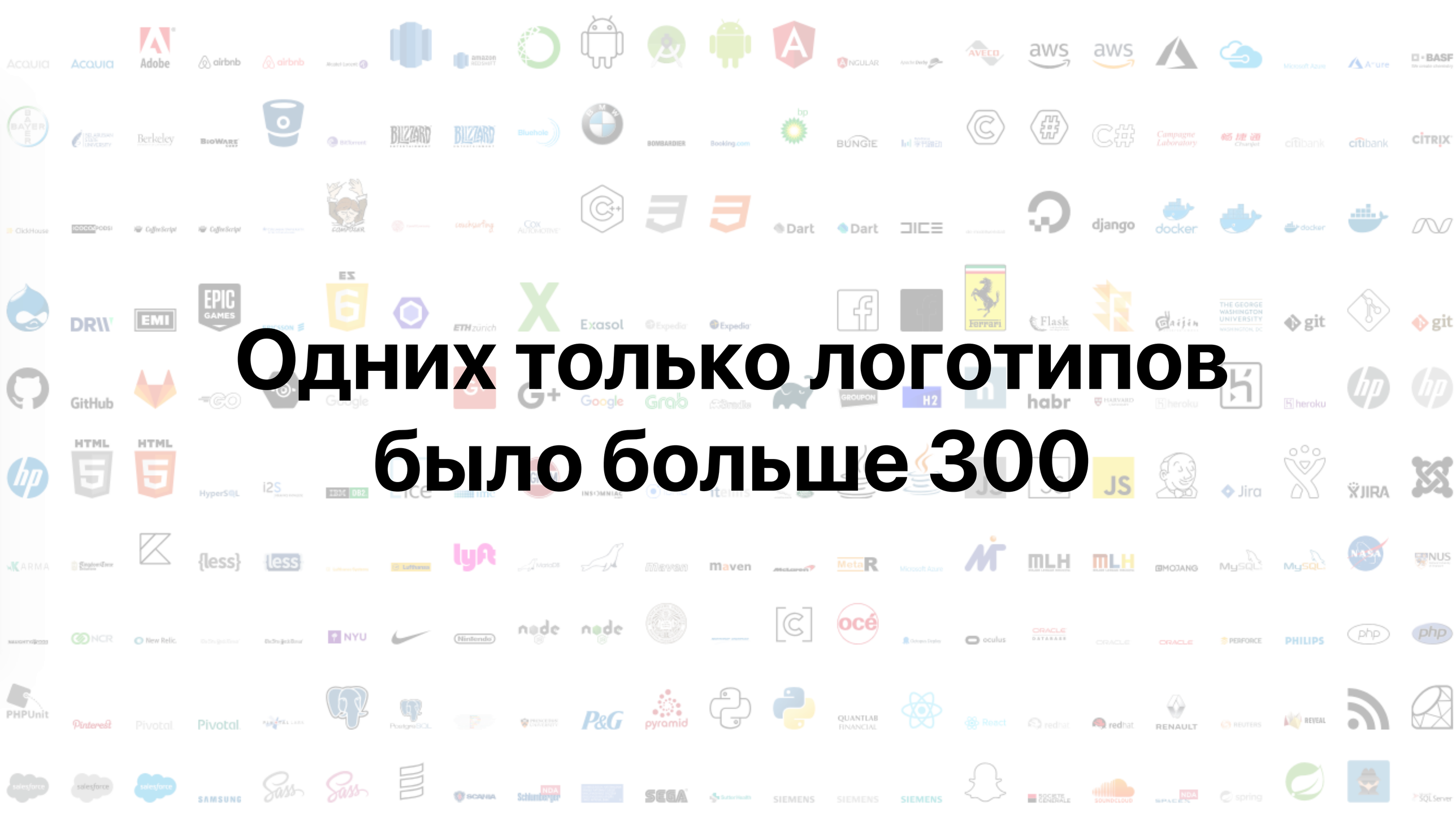


Disclaimer

Как я познакомился с webpack

Всё началось с редизайна сайта JetBrains в 2015 году

Требование: вся графика, кроме скриншотов, должна быть в SVG



Одних только логотипов было больше 300

Вызов принят!

Как можно вставить SVG в страницу?

```
<img src="image.svg">

<div class="image"><svg>...</svg></div>

.image {background: url(data:image/svg+xml;...)}

.image {background: url(sprite.svg) no-repeat 0 -33%}

<symbol id="image">...</symbol>

<img src="sprite.svg#image">
```

Что такое символ в SVG?

```
<svg style="display: none">
    <symbol id="icon-1">...</symbol>
    <symbol id="icon-2">...</symbol>
    <symbol id="icon-3">...</symbol>
</svg>

...

<body>
    <svg><use xlink:href="#icon-1"></use></svg>
    <svg><use xlink:href="#icon-2"></use></svg>
    <svg><use xlink:href="#icon-2" fill="blue"></use></svg>
</body>
```

Сначала решили реализовать на бекенде

Реализуем на фронтенде!

Готовых решений для работы с SVG-спрайтами в webpack не было

Пишем своё решение!

Как расширить возможности вебпак?

1. Лоадеры

- Используются чтобы "научить" webpack новым форматам файлов: CSS, HTML и т.д.
- Простая копценция – трансформер одного формата в другой
- Подробно описаны в документации

2. Плагины

- Более мощный инструмент, которым можно вмешаться в любую стадию билда
- Сложное API – надо понимать как webpack работает изнутри чтобы что-то сделать
- Документация очень скудна

Compilation Hooks

EDIT DOCUMENT

The `Compilation` module is used by the `Compiler` to create new compilations (or builds). A `compilation` instance has access to all modules and their dependencies (most of which are circular references). It is the literal compilation of all the modules in the dependency graph of an application. During the compilation phase, modules are loaded, sealed, optimized, chunked, hashed and restored.

The `Compilation` class also extends `Tapable` and provides the following lifecycle hooks. They can be tapped the same way as compiler hooks:

```
compilation.hooks.someHook.tap(/* ... */);
```

As with the `compiler`, `tapAsync` and `tapPromise` may also be available depending on the type of hook.

`buildModule`

`SyncHook`

Triggered before a module build has started.

Parameters: `module`

`rebuildModule`

`SyncHook`

Fired before rebuilding a module.

Parameters: `module`

`failedModule`

`SyncHook`

Run when a module build has failed.

Parameters: `module` `error`

`succeedModule`

`SyncHook`

Executed when a module has been built successfully.

Parameters: `module`

`finishModules`

`SyncHook`

Что же выбрать?

- Лоадер или плагин?
- Нет опыта
- Надо быстро!

Пишем свой лоадер!

Что такое лоадер?

- Это функция, которая трансформирует исходный код
- Работает с одним файлом
- Настраивается на файлы по маске:

```
rules: [{
  test: /\.css$/,
  loader: 'css-loader',
  options: {
    ...
  }
}]
```

Пример лоадера

```
function loader(source) {
  const newContent = `/* Hello, world! */ ${source}`;

  return `module.exports = ${newContent}`;
}
```

Что надо сделать с SVG?

```
<svg>...</svg>
```

```
<symbol>...</symbol>
```

Sprite

```
<svg>
  <symbol>...</symbol>
  <symbol>...</symbol>
</svg>
```

Смотрим аналоги: style-loader

```
import './styles.css';
```

```
var css = document.createTextNode('.a {...}');
var style = document.createElement('style')

style.appendChild(css);
```

Адаптируем под себя

```
import './image.svg';
```

```
const symbol = '<symbol id="image">...</symbol>';
const sprite = require('svg-sprite-loader/sprite');
sprite.add(symbol);
```

Создаём спрайт и вставляем в страницу

(где-то в app.js)

```
const sprite = require('svg-sprite-loader/sprite');
// ^ на этот момент все символы уже в объекте `sprite`


window.addEventListener('DOMContentLoaded', () => {
  sprite.render(document.body);
});
```

Код лоадера выглядит примерно так

```
function svgSpriteLoader(svg) {
   const symbol = transformToSymbol(svg);

   return `
     var sprite = require('svg-sprite-loader/sprite');
     sprite.add(${symbol.id}, ${symbol.content});
     module.exports = ${symbol.id};
   `;
}
```

Ура, получилось?

PROBLEM?

Стали вылезать лютые баги

- Сломанные градиенты при использовании History API
- Сломанные градиенты и маски в FireFox если они внутри <symbol>
- Встроенные в SVG стили иногда не работали в IE/Egde
- Но самое главное...

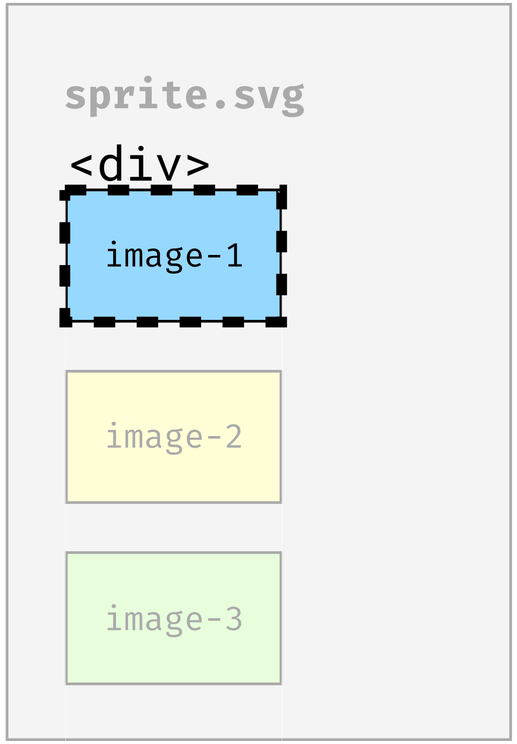
Это не работало без JS

Нужно решение без JS и
чтобы работало во всех браузерах

Смотрим ещё раз

```
<img src="image.svg">

<div class="image"><svg>...</svg></div>

.image {background: url(data:image/svg+xml;...)}

.image {background: url(sprite.svg) no-repeat 0 -33%}

<symbol id="image">...</symbol>

<img src="sprite.svg#image">
```

Классический спрайт




```
.image-2 {
   background: url('sprite.svg') no-repeat 0 -33%;
   background-size: 100% 100%;
}
```

Хотелось автоматически

```
.img {
  background-image: url(image.svg);
}
```

```
.img {
  background: url('sprite.svg') no-repeat 0 -33%;
  background-size: 100% 100%;
}
```

Как это сделать?

```
.img {
  background-image: url(image.svg);
}
```

1. svg -> symbol

```
.img {
  background: url('sprite.svg') no-repeat 0 -33%;
  background-size: 100% 100%;
}
```

3. transform CSS

2. create sprite file

Лоадером тут не обойтись

Смотрим про плагины

- Плагины могут всё что не могут лоадеры
- Документация описывает только простейшие случаи
- **"Будьте готовы читать исходники"!**

Ок, идём курить сорцы!

AMDMainTemplatePlugin.js
APIPlugin.js
AsyncDependenciesBlock.js
AsyncDependencyToInitialChunkError.js
AutomaticPrefetchPlugin.js
BannerPlugin.js
BasicEvaluatedExpression.js
CachePlugin.js
CaseSensitiveModulesWarning.js
Chunk.js
ChunkGroup.js
ChunkRenderError.js
ChunkTemplate.js
CommentCompilationWarning.js
compareLocations.js
CompatibilityPlugin.js
Compilation.js
Compiler.js
ConcurrentCompilationError.js
ConstPlugin.js
ContextExclusionPlugin.js
ContextModule.js
ContextModuleFactory.js
ContextReplacementPlugin.js
DefinePlugin.js
DelegatedModule.js
DelegatedModuleFactoryPlugin.js
DelegatedPlugin.js
DependenciesBlock.js
DependenciesBlockVariable.js
Dependency.js
DllEntryPlugin.js
DllModule.js
DllModuleFactory.js
DllPlugin.js
DllReferencePlugin.js
DynamicEntryPlugin.js
EntryModuleNotFoundError.js
EntryOptionPlugin.js
Entrypoint.js
EnvironmentPlugin.js
ErrorHelpers.js
EvalDevToolModulePlugin.js
EvalDevToolModuleTemplatePlugin.js
EvalSourceMapDevToolModuleTemplatePlugin.js
EvalSourceMapDevToolPlugin.js
ExportPropertyMainTemplatePlugin.js
ExtendedAPIPlugin.js
ExternalModule.js
ExternalModuleFactoryPlugin.js
ExternalsPlugin.js
FlagDependencyExportsPlugin.js
FlagDependencyUsagePlugin.js
FlagInitialModulesAsUsedPlugin.js
formatLocation.js
FunctionModulePlugin.js
FunctionModuleTemplatePlugin.js
Generator.js
GraphHelpers.js
HarmonyLinkingError.js
HashedModuleIdsPlugin.js
HotModuleReplacement.runtime.js
HotModuleReplacementPlugin.js
HotUpdateChunk.js
HotUpdateChunkTemplate.js
IgnorePlugin.js
JavascriptGenerator.js
JavascriptModulesPlugin.js
JsonGenerator.js
JsonModulesPlugin.js
JsonParser.js
LibManifestPlugin.js
LibraryTemplatePlugin.js
LoaderOptionsPlugin.js
LoaderTargetPlugin.js
MainTemplate.js
MemoryOutputFileSystem.js
Module.js
ModuleBuildError.js
ModuleDependencyError.js
ModuleDependencyWarning.js
ModuleError.js
ModuleFilenameHelpers.js
ModuleNotFoundError.js
ModuleParseError.js
ModuleReason.js
ModuleTemplate.js
ModuleWarning.js
MultiCompiler.js
MultiEntryPlugin.js
MultiModule.js
MultiModuleFactory.js
MultiStats.js
MultiWatching.js
NamedChunksPlugin.js
NamedModulesPlugin.js
NodeStuffPlugin.js
NoEmitOnErrorsPlugin.js
NoModeWarning.js
NormalModule.js
NormalModuleFactory.js
NormalModuleReplacementPlugin.js
NullFactory.js
OptionsApply.js
OptionsDefaulter.js
Parser.js
ParserHelpers.js
PrefetchPlugin.js
ProgressPlugin.js
ProvidePlugin.js
RawModule.js
RecordIdsPlugin.js
RemovedPluginError.js
RequestShortener.js
RequireJsStuffPlugin.js
ResolverFactory.js
RuleSet.js
RuntimeTemplate.js
SetVarMainTemplatePlugin.js
SingleEntryPlugin.js
NormalModuleFactory.js
NormalModuleReplacementPlugin.js
NullFactory.js
OptionsApply.js
OptionsDefaulter.js
Parser.js
ParserHelpers.js
PrefetchPlugin.js
ProgressPlugin.js
ProvidePlugin.js
RawModule.js
RecordIdsPlugin.js
RemovedPluginError.js
RequestShortener.js
RequireJsStuffPlugin.js
ResolverFactory.js
RuleSet.js
RuntimeTemplate.js
SetVarMainTemplatePlugin.js
SingleEntryPlugin.js
SizeFormatHelpers.js
SourceMapDevToolModuleOptionsPlugin.js
SourceMapDevToolPlugin.js
Stats.js
Template.js
TemplatedPathPlugin.js
UmdMainTemplatePlugin.js
UnsupportedFeatureWarning.js
UseStrictPlugin.js
validateSchema.js
WarnCaseSensitiveModulesPlugin.js
WarnNoModeSetPlugin.js
WatchIgnorePlugin.js
Watching.js
webpack.js
webpack.web.js
WebpackError.js
WebpackOptionsApply.js
WebpackOptionsDefaulter.js
WebpackOptionsValidationError.js

Главный модуль

```
function webpack(options, callback) {
  const options = new WebpackOptionsDefaulter().process(options);

  const compiler = new Compiler(options.context);
  compiler.options = options;

  compiler.hooks.environment.call();
  compiler.hooks.afterEnvironment.call();

  compiler.run(callback);
}
```

Compiler.run

```
...

this.hooks.beforeRun.callAsync(this, err => {
  if (err) return finalCallback(err);

  this.hooks.run.callAsync(this, err => {
    if (err) return finalCallback(err);

    this.readRecords(err => {
      if (err) return finalCallback(err);

      this.compile(onCompiled);
    });
  });
});
```

Compiler.compile

```
const params = this.newCompilationParams();

this.hooks.beforeCompile.callAsync(params, err => {
  this.hooks.compile.call(params);

  const compilation = this.newCompilation(params);

  this.hooks.make.callAsync(compilation, err => {
    compilation.finish();

    compilation.seal(err => {
      this.hooks.afterCompile.callAsync(compilation, err => {
        return callback(null, compilation);
      });
    });
  });
});
```

Что за хуки такие?

Большинство объектов в webpack наследуюются от пакета Tapable

Это такая pub/sub система, которая регистрирует свои события (хуки) и запускает обработчики разными способами - sync, async, promise, etc.

Пример плагина из доков webpack

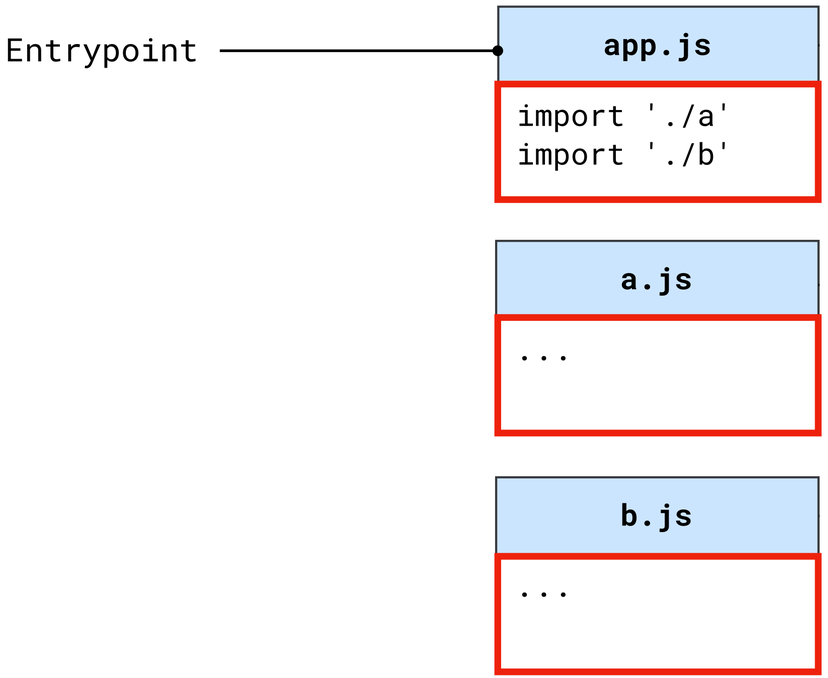
```
class HelloWorldPlugin {
  apply(compiler) {
    compiler.hooks.done.tap('HelloWorldPlugin', (stats) => {
      console.log(stats.hasErrors());
    });
  }
}
```

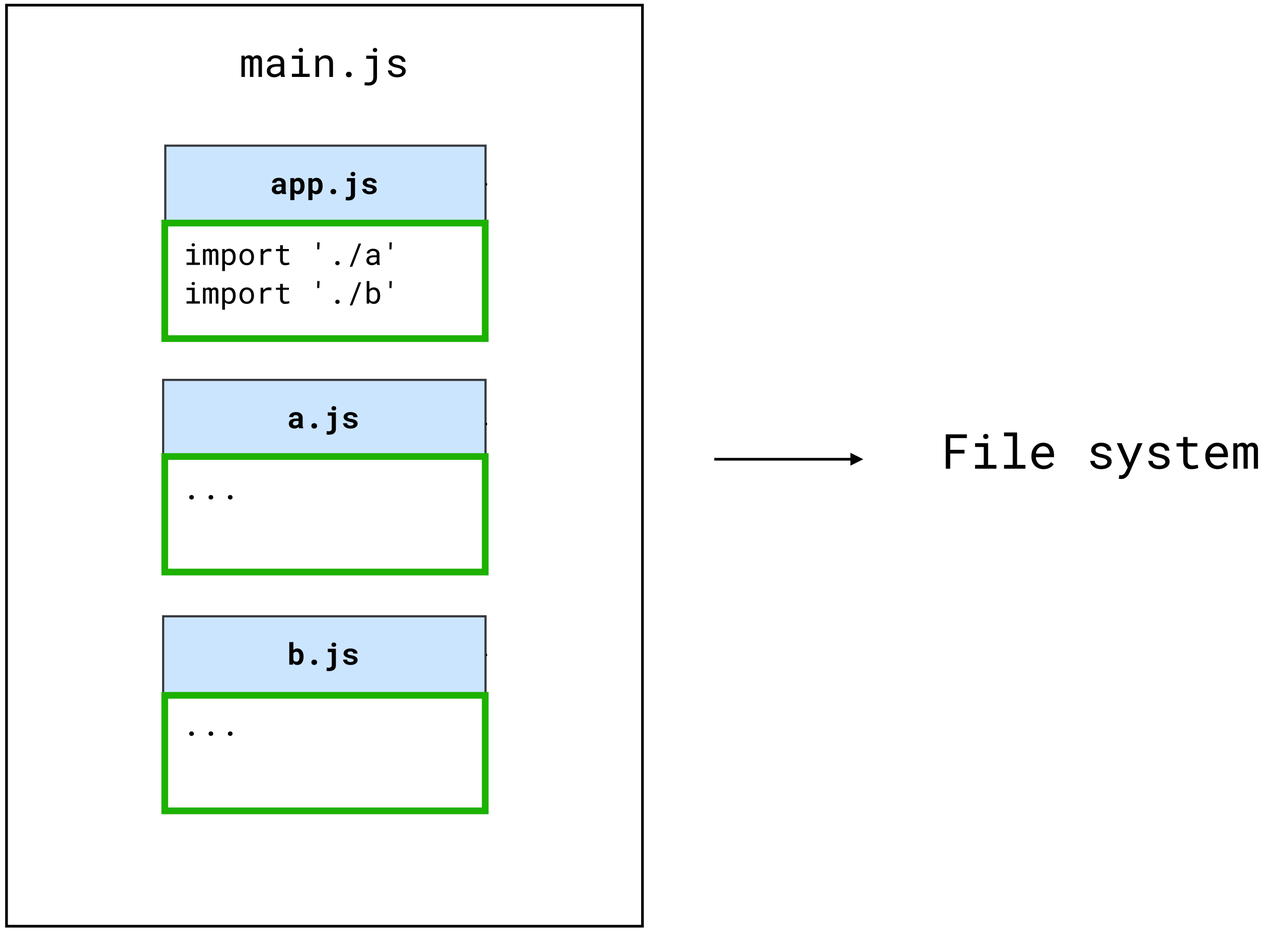
Инициализация

Подписка

webpack весь состоит из таких плагинов, они запускаются в разные стадии сборки и влияют на процесс

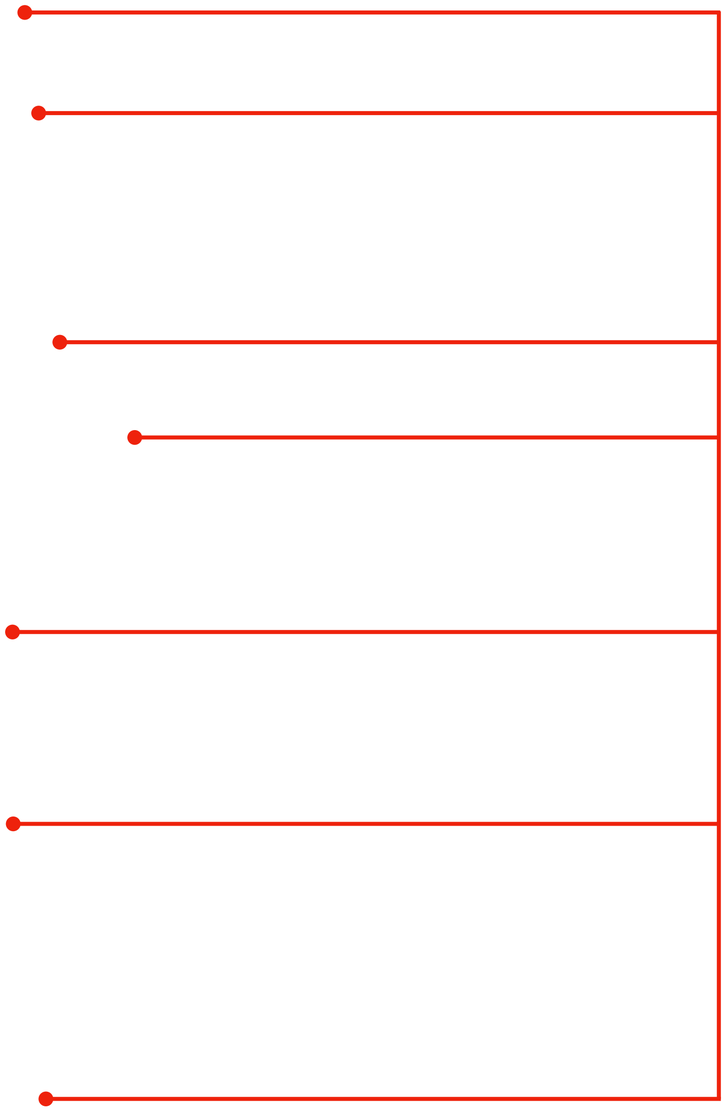
Как собрать бандл?


Entrypoint
app.js
import './a'
import './b'
a.js
...
b.js
...


main.js
app.js
import './a'
import './b'
a.js
...
b.js
...
File system

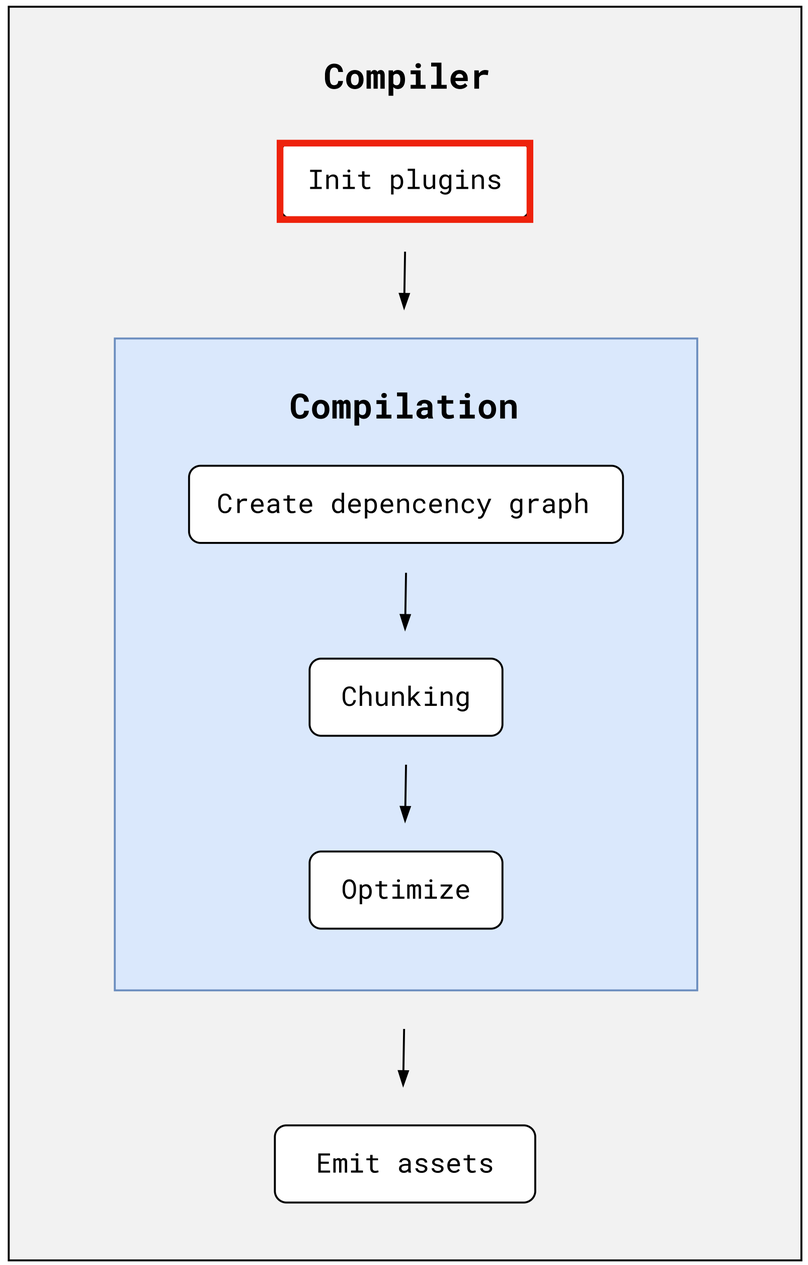
Как это делает webpack?

```
webpack.config.js
entry: "app.js"
```



Pluggable

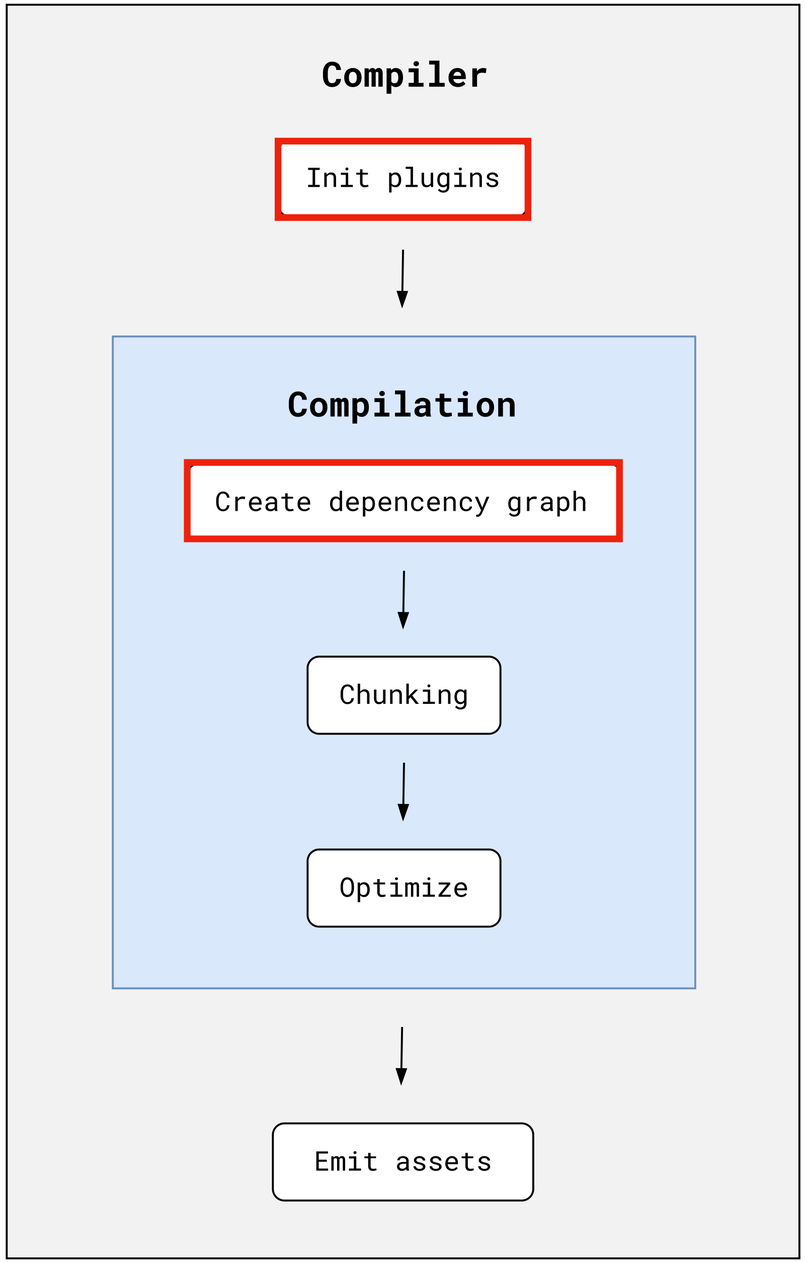
webpack.config.js
entry: "app.js"



Compiler вызывает метод `apply` и передаёт себя как аргумент

```
class HelloWorldPlugin {
  apply(compiler) {
    compiler.hooks.done.tap('HelloWorldPlugin', (stats) => {
      console.log(stats.hasErrors());
    });
  }
}
```

webpack.config.js
entry: "app.js"



Что такое зависимость в webpack?

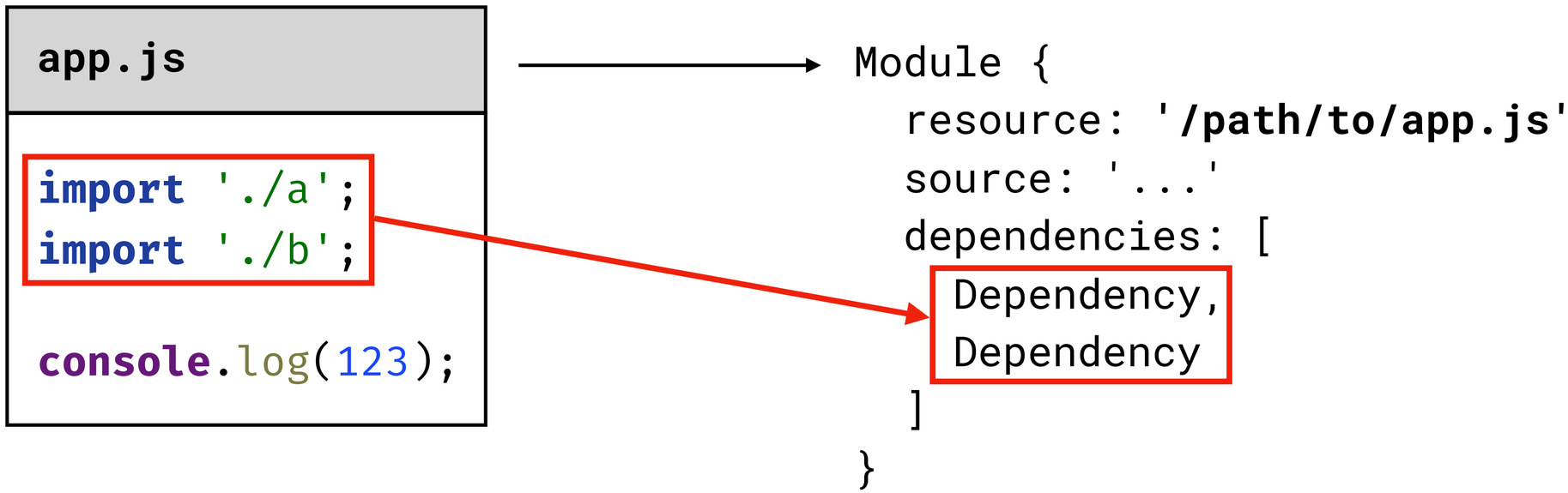
Это объект, отражающий зависимость одного модуля от другого

app.js

```
import './a';
import './b';

console.log(123);
```

Что такое модуль?

Минимальная структурная единица приложения в виде файла



Как строится граф зависимостей?

Module factory

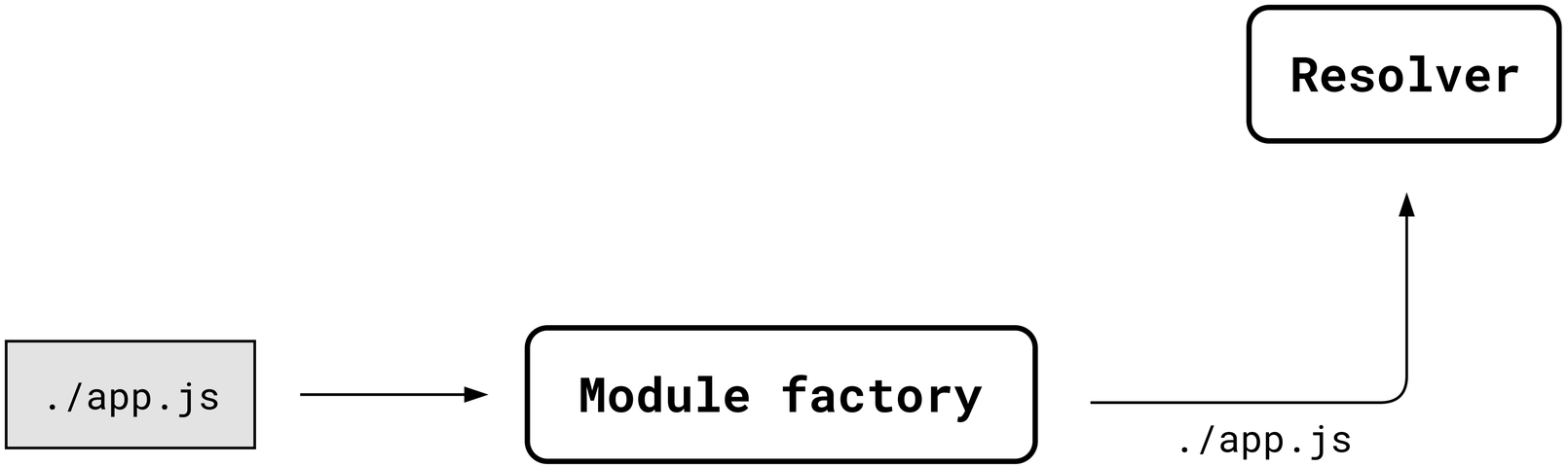
Resolver ищет файлы

- Алиасы

```
import parseSvg from 'Utils/parseSvg';
```

- Entrypoint пакета в package.json

```
{
  "main": "lib/index.js",
  "module": "lib/index.es6.js",
  "main:src": "src/index.js"
}
```


Resolver
./app.js
Module factory
./app.js

app.js

```
import './a'
import './b'
```

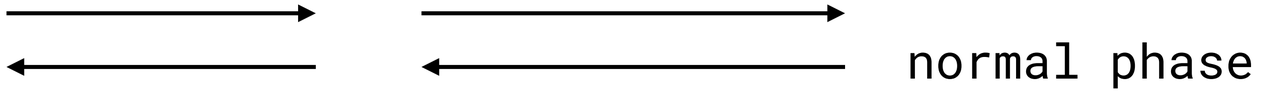
Loader runner запускает лоадеры

Справа-налево

```
{
  test: /\.css$/,
  use: ['css-loader', 'postcss-loader', 'sass-loader']
}
```

Pitch phase

```
{
  test: /\.css$/,
  use: ['css-loader', 'postcss-loader', 'sass-loader']
}
```




```
module.exports.pitch = function(remainingRequest, precedingRequest) {
  return `module.exports = require("new-path")`;
}
```

Pitch phase

Собирает модуль последовательно применяя лоадеры из конфига справа-налево

```
{
  test: /\.css$/,
  use: ['css-loader', 'postcss-loader', 'sass-loader']
}
```

Работает только с одним файлом

Доступа к другим модулям нет, так как граф ещё
не построен окончательно

Работает со строкой или буфером

Бинарные данные – буфер, остальное строка

Может передать мета-информацию следующему лоадеру

```
// postcss-loader
module.exports = function (content, sourcemap, meta) {
  const ast = meta.ast;

  ast.root.walkDecls(() => {
    ...
  });
}
```

Не может хранить состояние

Перед каждым запуском его контекст сбрасывается

```
module.exports = function (content) {
  if (!this.data) {
    this.data = [];
  }

  this.data.push('foo');
  // this.data пустой при каждом запуске лоадера
}
```

Должен всегда возвращать предсказуемый результат

Поскольку результат его работы используется кешом webpack

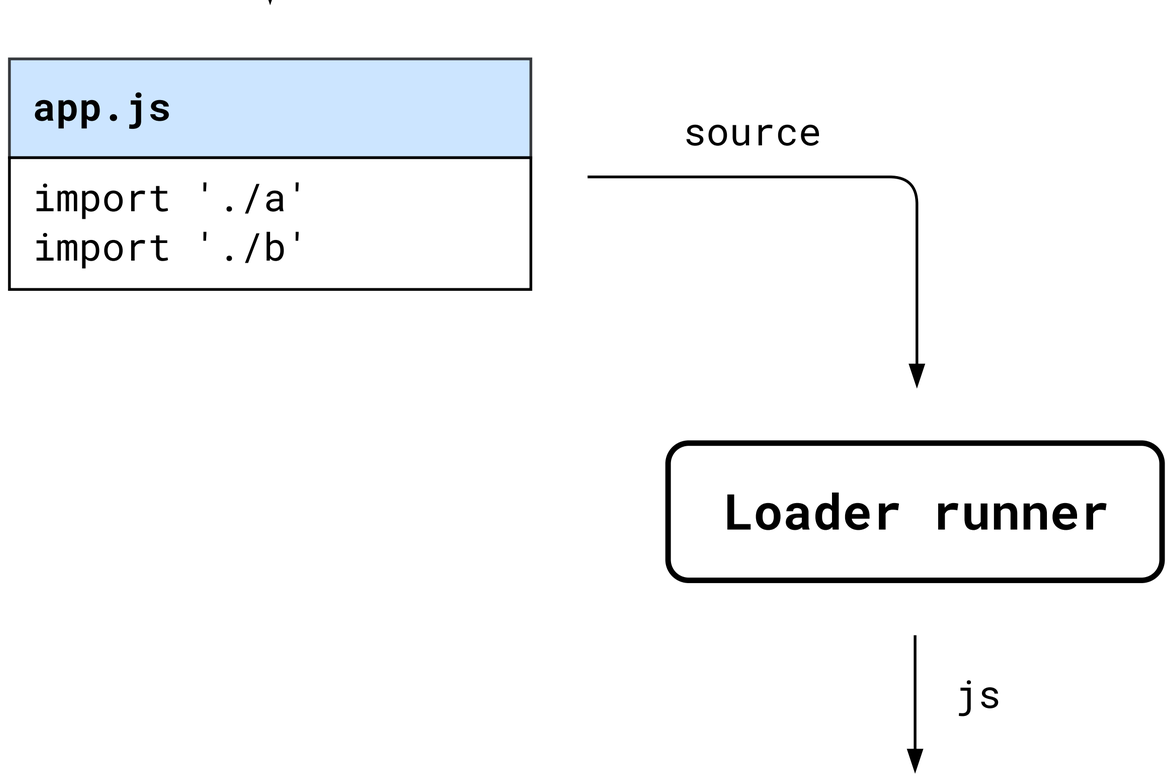
И он должен всегда возвращать JavaScript

CSS

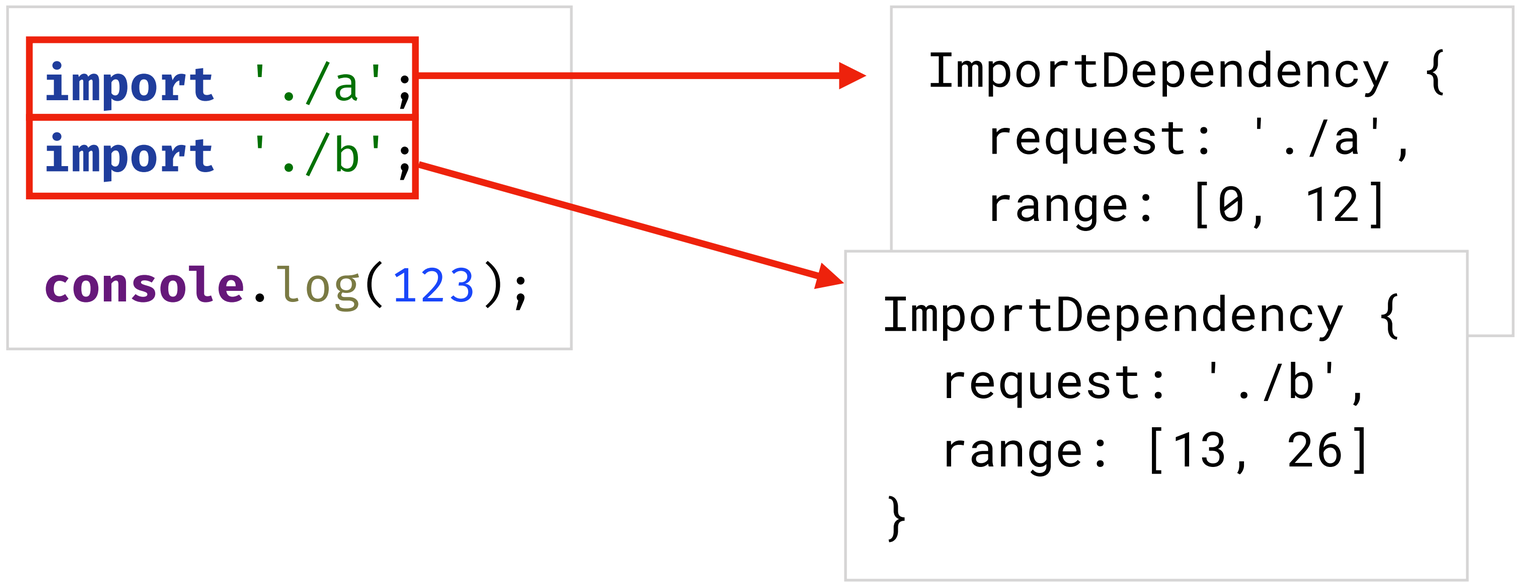
```css
.img {
  background-image: url('image.png');
}
```

JS

```js
module.exports = '.img {\nbackground-image: url(\'' +
  require('./image.png') +
  '\');\n}';
```

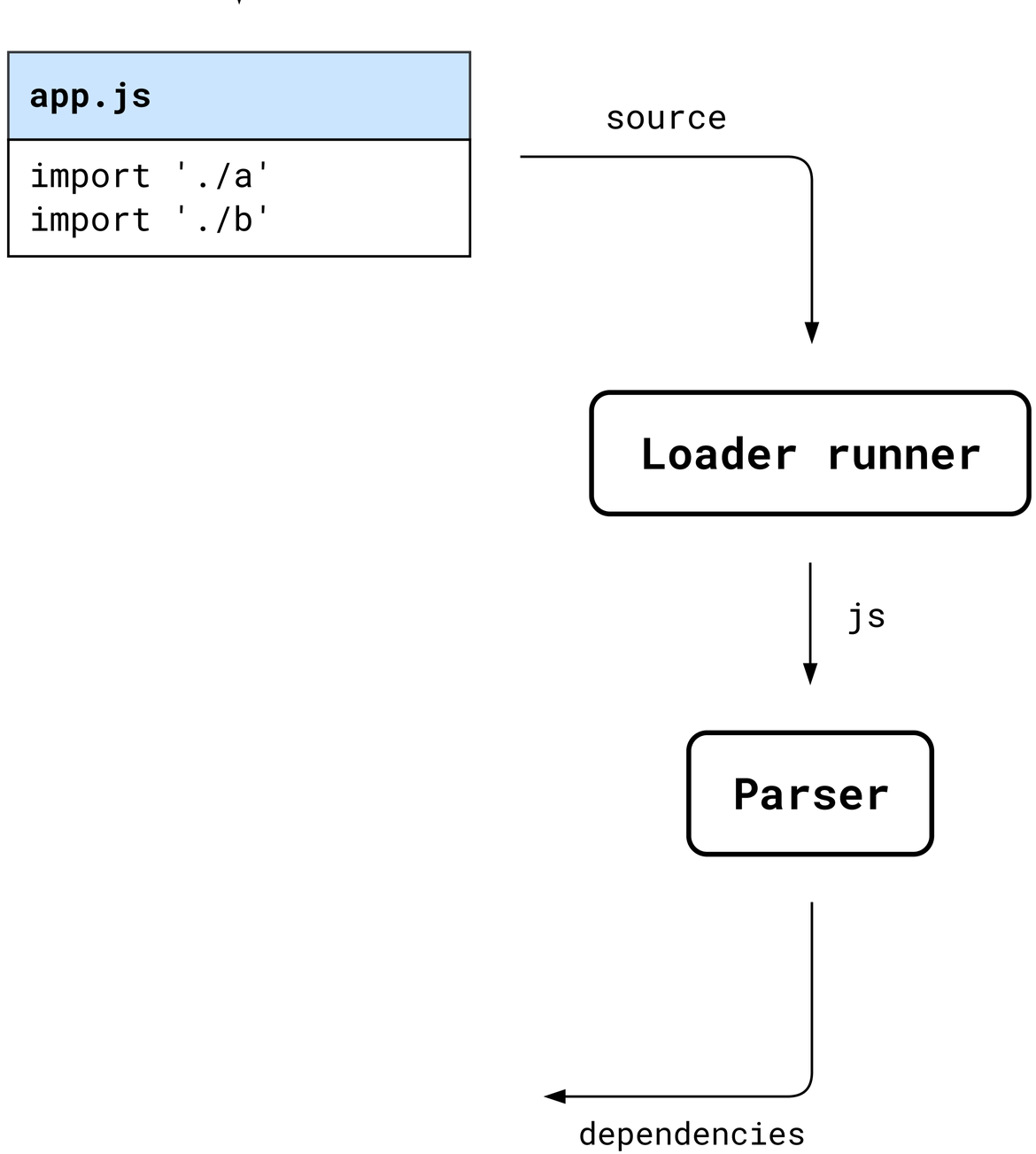

app.js
import './a'
import './b'
source
Loader runner
js

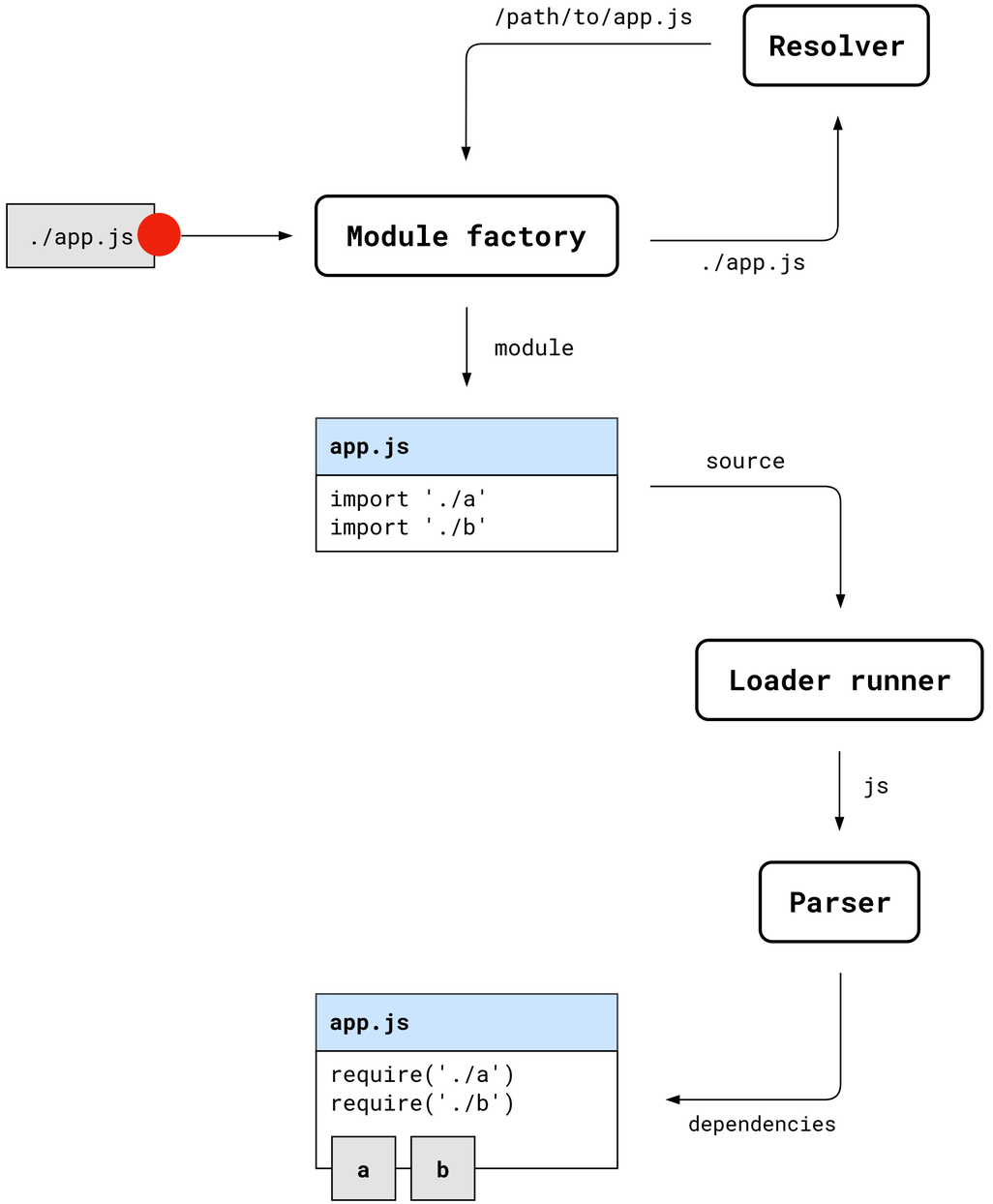
Parser извлекает зависимости

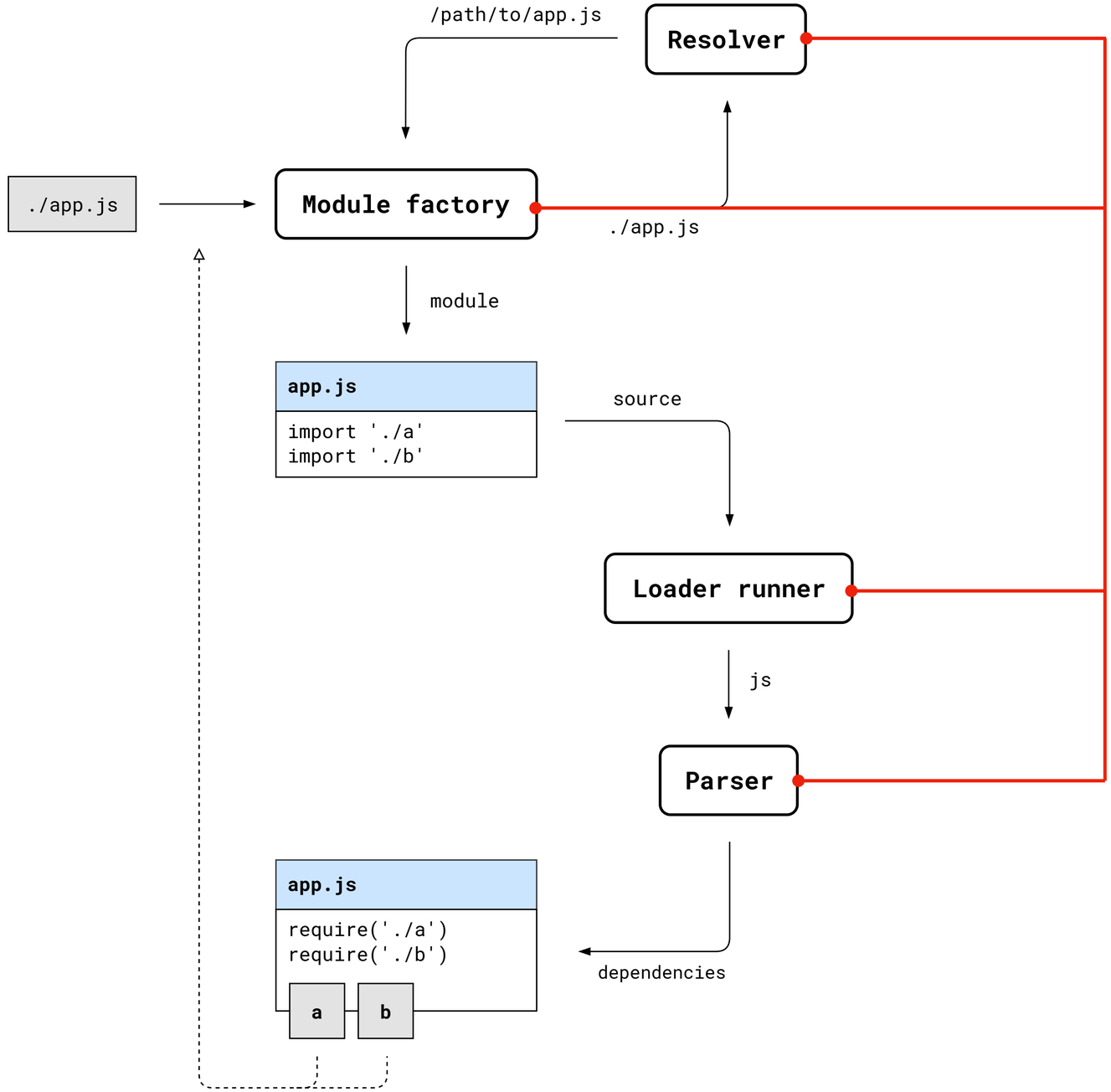


Parser понимает огромное количество форматов зависимостей

```
const foo = require('bar');

import foo from 'bar';

require.context('./dir', /\.svg$/, true);
```

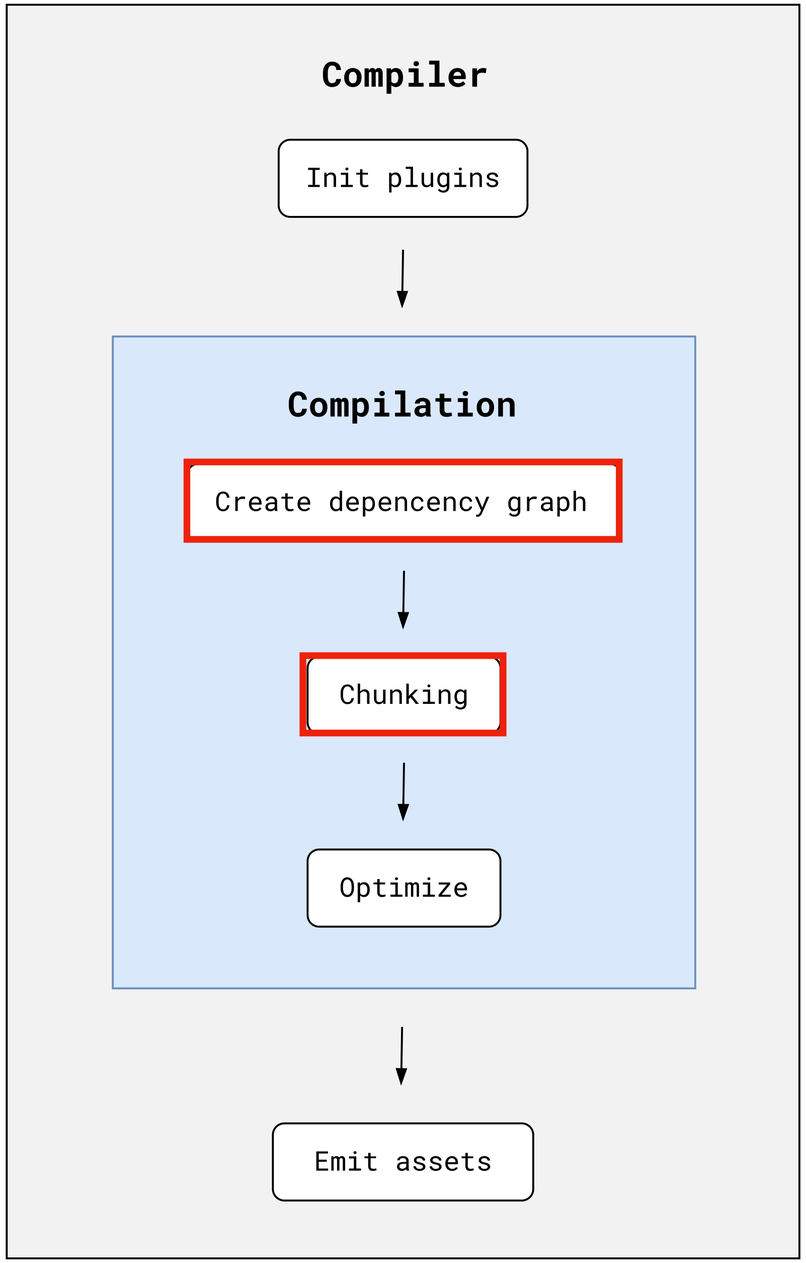

app.js
import './a'
import './b'
source
Loader runner
js
Parser
dependencies


/path/to/app.js
Resolver
./app.js
Module factory
./app.js
module
app.js
import './a'
import './b'
source
Loader runner
js
Parser
app.js
require('./a')
require('./b')
a
b
dependencies


/path/to/app.js
Resolver
./app.js
Module factory
./app.js
module
app.js
import './a'
import './b'
source
Loader runner
js
Parser
app.js
require('./a')
require('./b')
a
b
dependencies

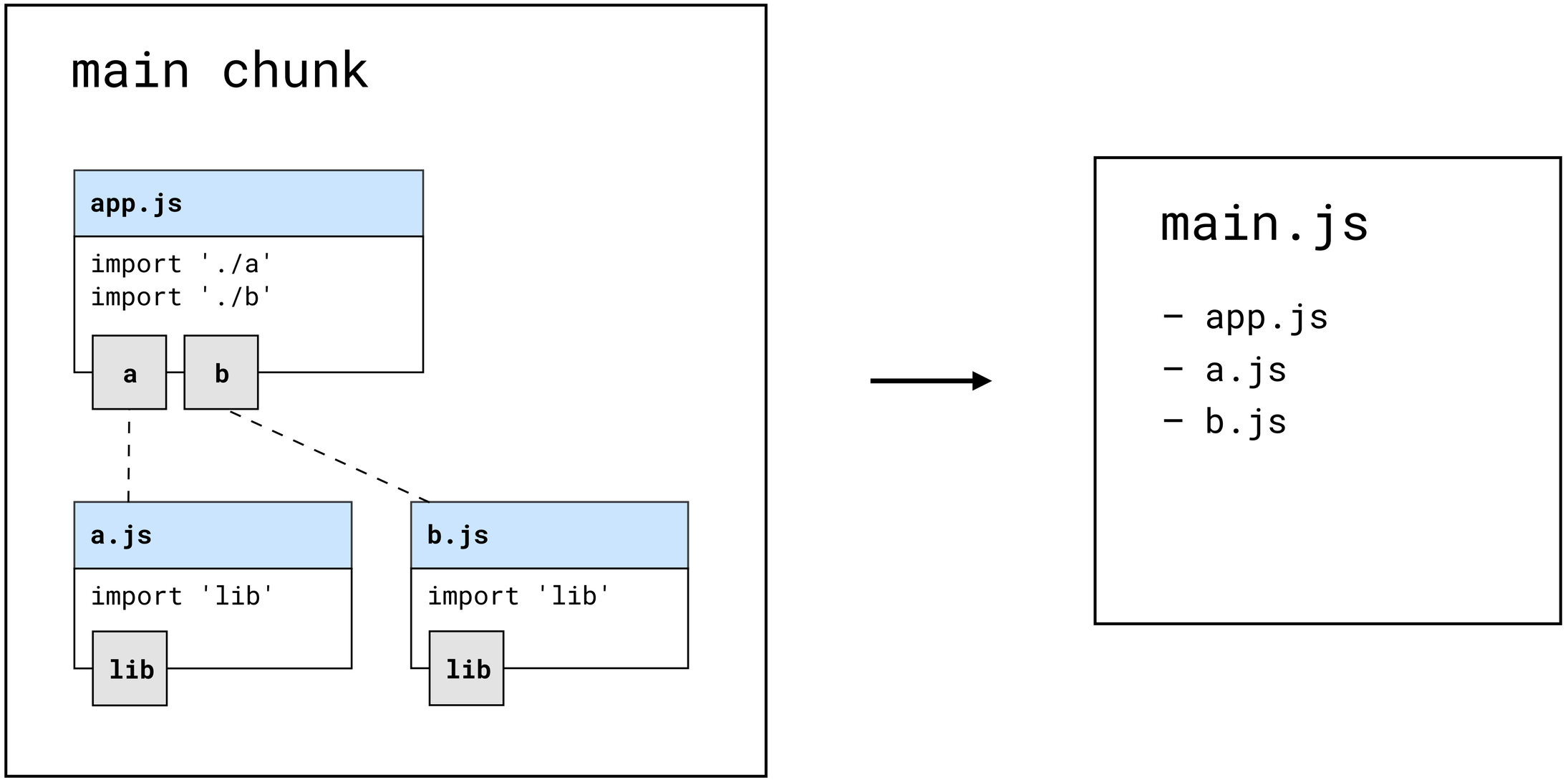
Pluggable

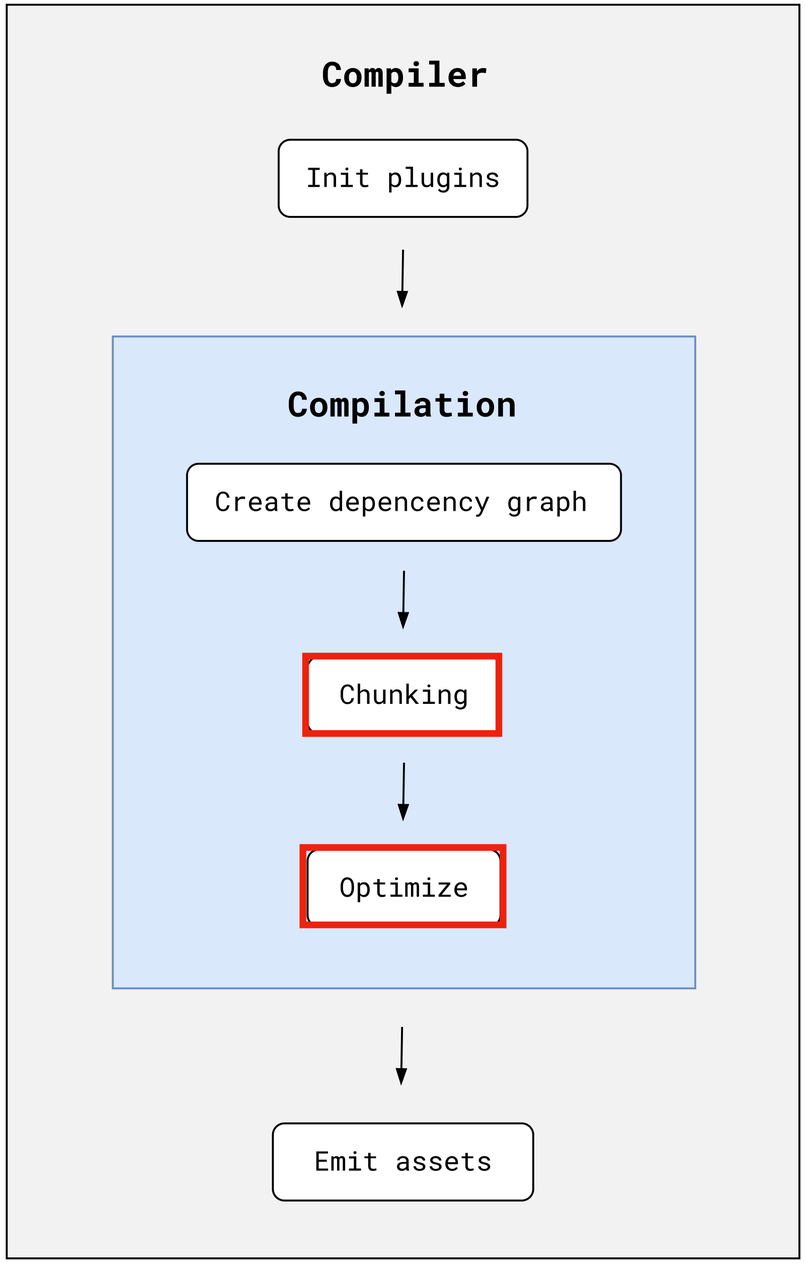
Вернёмся к общей схеме


Compiler
Init plugins
Compilation
Create depencency graph
Chunking
Optimize
Emit assets

Что такое чанк?

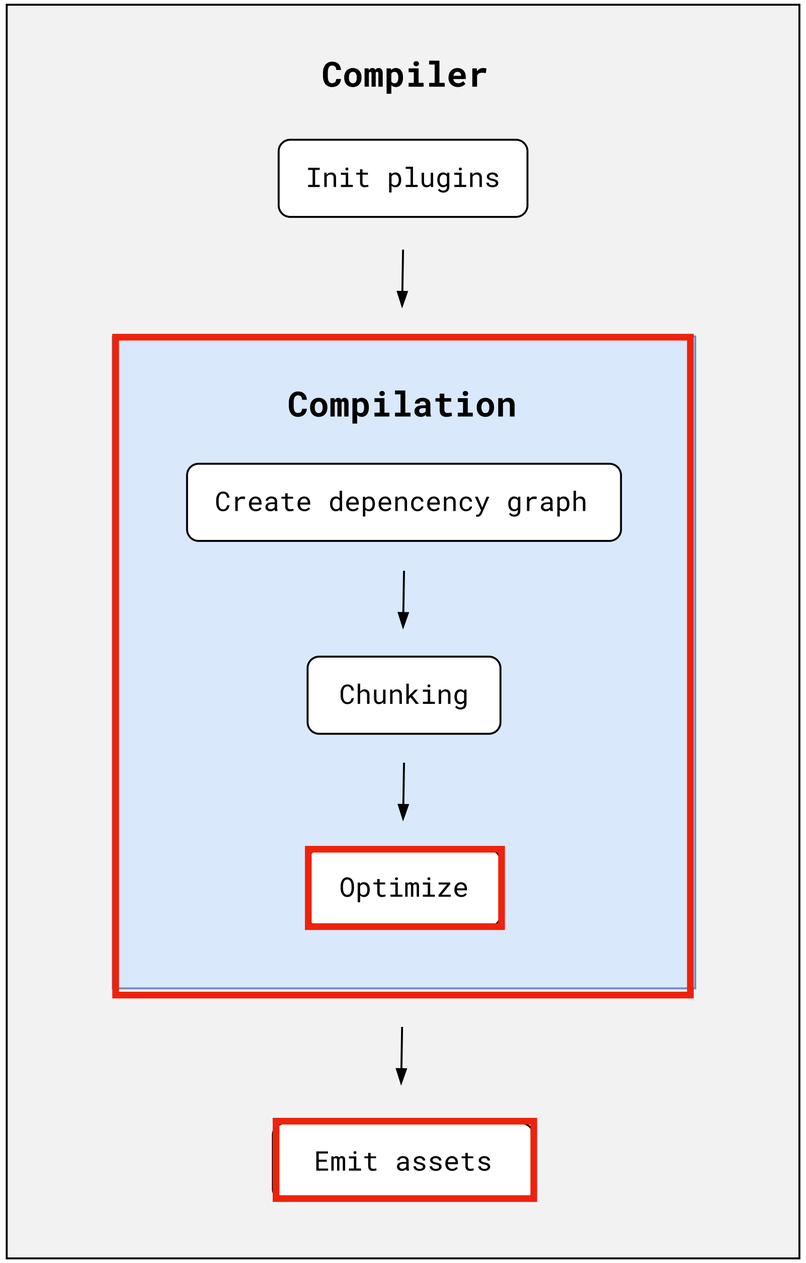
Это кусочек графа зависимостей, который будет собран в отдельный файл после сборки


main chunk
app.js
import './a'
import './b'
a
b
a.js
import 'lib'
lib
b.js
import 'lib'
lib
main.js
- app.js
- a.js
- b.js

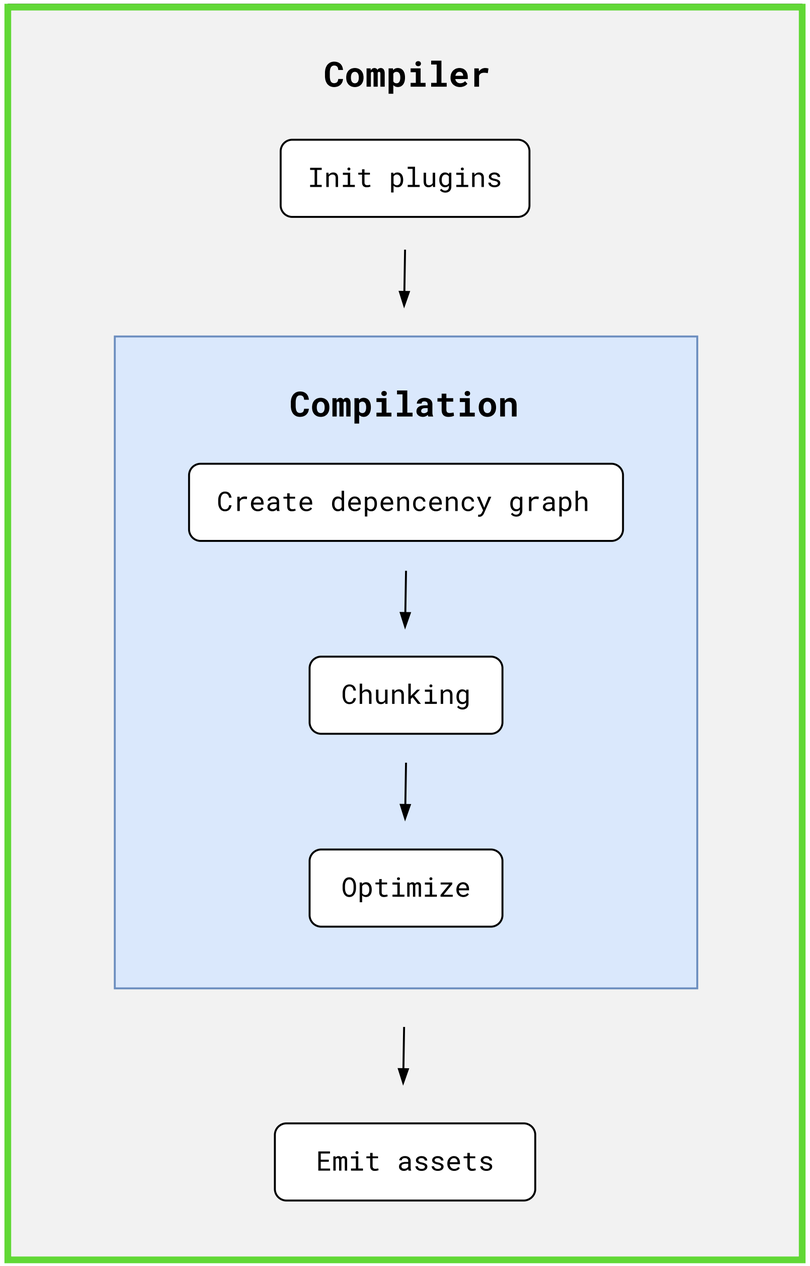

Compiler
Init plugins
Compilation
Create depencency graph
Chunking
Optimize
Emit assets

Оптимизация

- Минификация, tree shaking в production режиме
- Сортировка и удаление дубликатов
- Генерация id
- И ещё куча всего что настраивается опцией optimization в конфиге


Compiler
Init plugins
Compilation
Create depencency graph
Chunking
Optimize
Emit assets

Input и output FS


Compiler
Init plugins
Compilation
Create depencency graph
Chunking
Optimize
Emit assets

Но вернёмся к SVG-спрайтам

Хотелось чего-то такого

```
.img {
   background-image: url(image.svg);
}
```

```
.img {
   background: url('sprite.svg') no-repeat 0 0;
   background-size: 100% 104.50%;
}
```

Как сделать SVG спрайт в webpack?

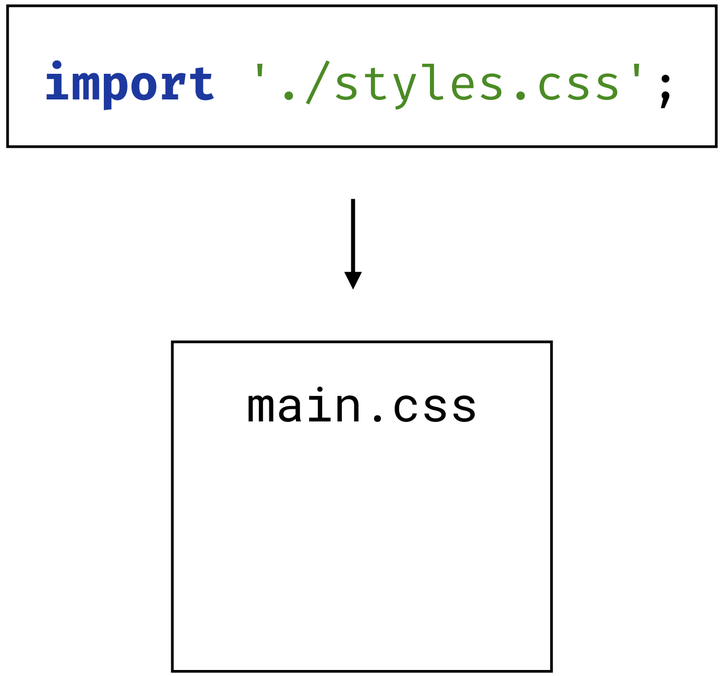
1. Лоадером перегнать <svg> в <symbol> и
   сохранить в какое-нибудь хранилище

Но как сохранить данные в лоадере если он не может хранить состояние?

Курим сорцы дальше: extract-text-webpack-plugin

extract-text-webpack-plugin

Извлекает CSS в отдельный файл



extract-text-webpack-plugin: настройка

```
{
  test: /\.css$/,
  use: ExtractPlugin.extract({use: [
    'css-loader',
    'postcss-loader',
    'sass-loader'
  ]})
}

...

new ExtractPlugin()
```

extract-text-webpack-plugin: настройка

```
{
  test: /\.css$/,
  use: [
    'extract-text-webpack-plugin/loader',
    'css-loader',
    'postcss-loader',
    'sass-loader'
  ]
}

...

new ExtractPlugin()
```

extract-text-webpack-plugin: лоадер

```
const result = cssLoaderResult.toString();

someStorage(file, result);

return '// text was extracted by extract-plugin';
```

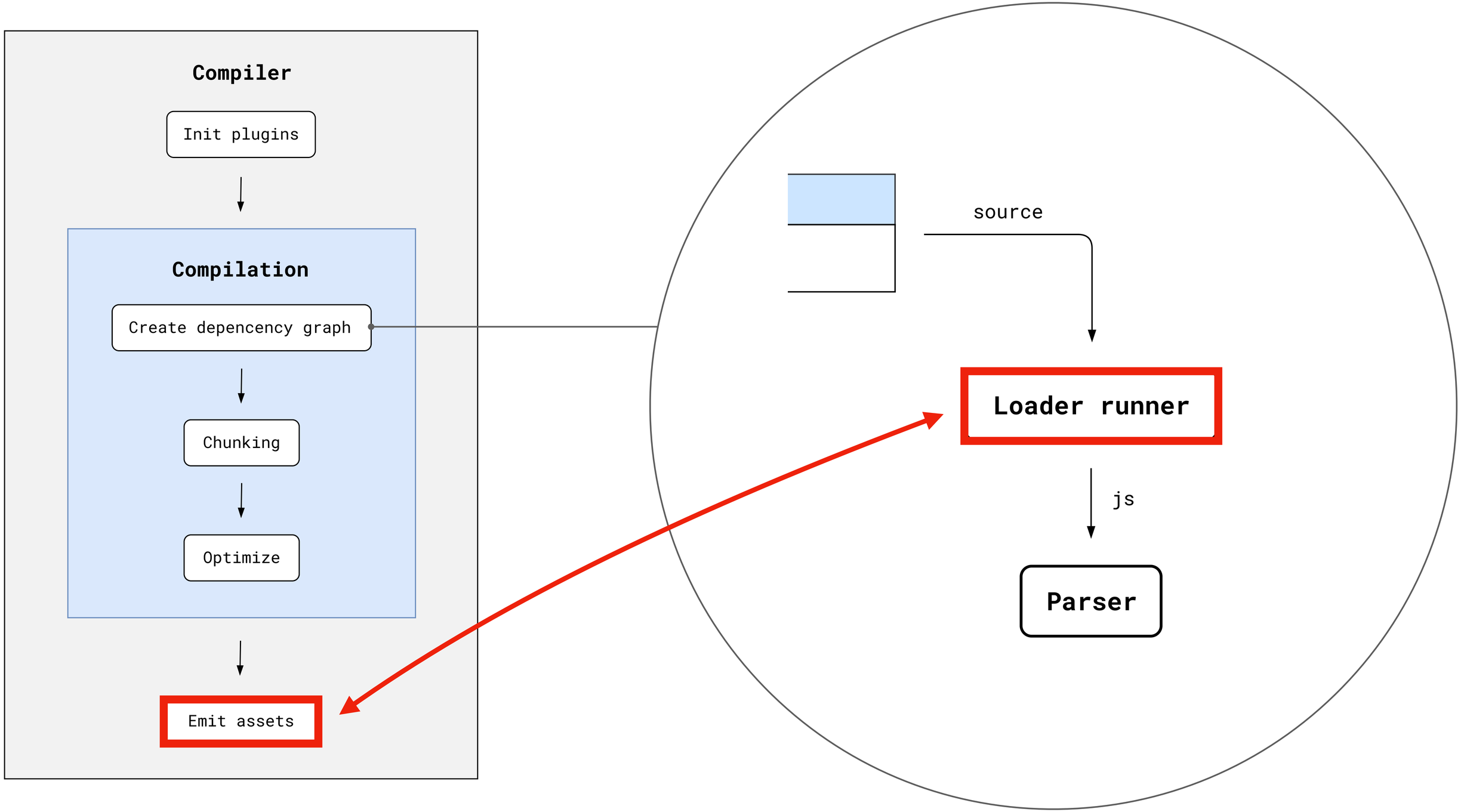
Лайфхак

Плагин может расшарить с лоадером произвольные данные используя хук normalModuleLoader

```
class Plugin {
   apply(compiler) {
      const myData = [];

      compiler.hooks.compilation.tap('MyPlugin', compilation => {
         compilation.hooks.normalModuleLoader.tap('MyPlugin', loaderCtx => {
            loaderCtx._myData = myData;
         });
      });
   }
}
```

Использование в лоадере

```
function loader(content) {

   ...

   const data = this._myData
   data.push(result);
}
```


Compiler
Init plugins
Compilation
Create depencency graph
Chunking
Optimize
Emit assets
source
Loader runner
js
Parser

Как сделать SVG спрайт в webpack?

1. Лоадером перегнать <svg> в <symbol> и сохранить в **плагин** ✓
2. Плагином собрать все символы и создать спрайт

Плагин для SVG

```
const symbols = [];

// шарим данные с лоадером
compiler.hooks.compilation.tap('MyPlugin', compilation => {
  compilation.hooks.normalModuleLoader.tap(NAMESPACE, loaderCtx => {
    loaderCtx._pluginData = symbols;
  });
});

// создаём файл спрайта
compiler.hooks.emit.tap('MyPlugin', compilation => {
  const sprite = `<svg>${symbols.join('')}</svg>`;
  compilation.assets['sprite.svg'] = sprite;
});
```

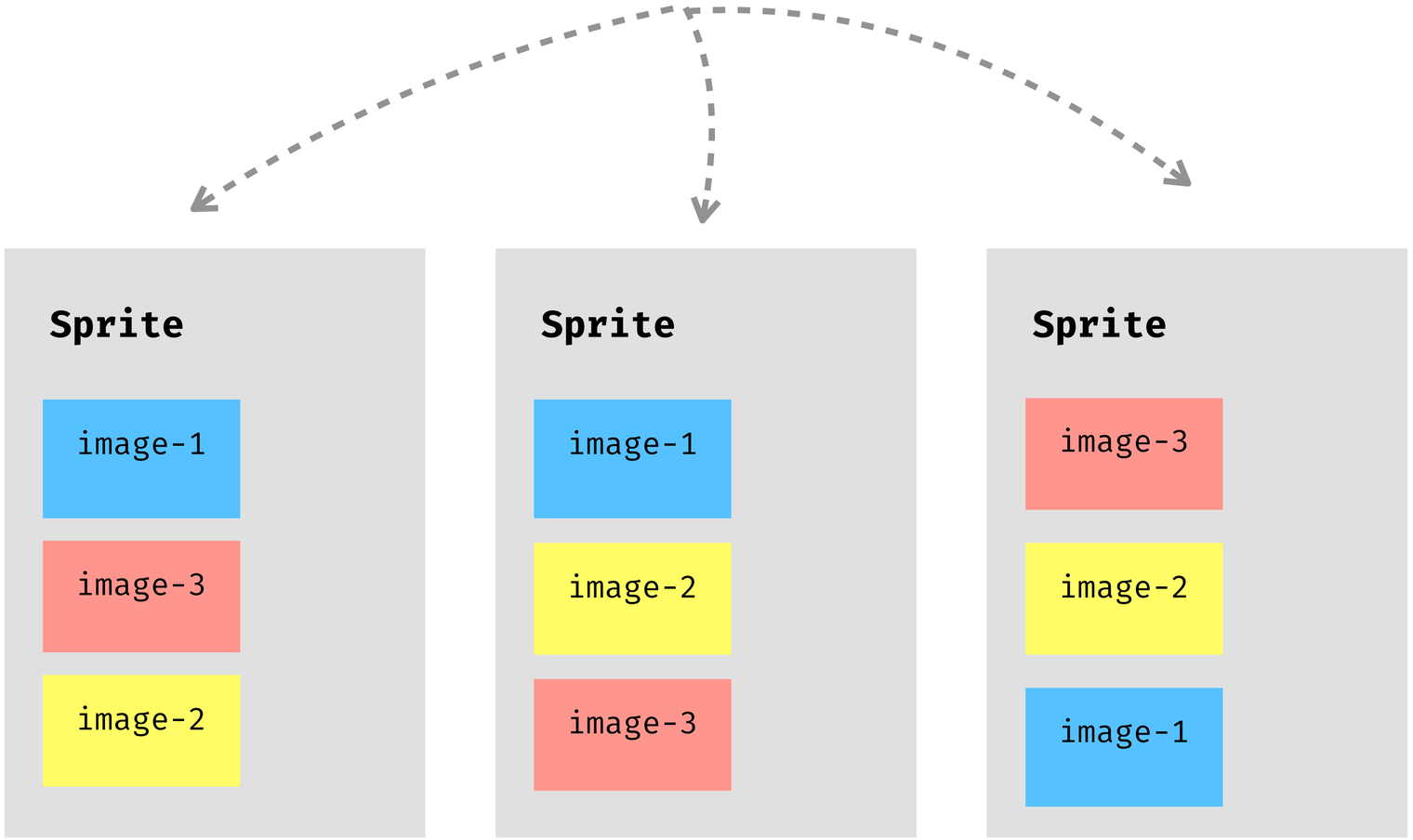
Ура, получилось?

1. Лоадером перегнать <svg> в <symbol> и сохранить в **плагин** ✓

2. Плагином собрать все символы и создать спрайт ✓

При каждом билде порядок символов в спрайте разный

```
.img1 {background-image: url(image-1.svg);}
.img2 {background-image: url(image-2.svg);}
.img3 {background-image: url(image-3.svg);}
```



webpack запускает лоадеры асинхронно, поэтому порядок символов не гарантирован

Решение

Перед созданием спрайта нужно
отсортировать символы по какому-нибудь
постоянному признаку, например по алфавиту

Ура, получилось?

1. Лоадером перегнать <svg> в <symbol> и сохранить в **плагин** ✓

2. Плагином собрать все символы и создать спрайт ✓

3. Обработать CSS для правильного позиционирования символа

Лоадер для CSS

```
.img {
   background-image: url(image.svg);
}
```

```
.img {
   background: url('sprite.svg') no-repeat 0 0;
   background-size: 100% 104.50%;
}
```

Пишем ещё один лоадер!

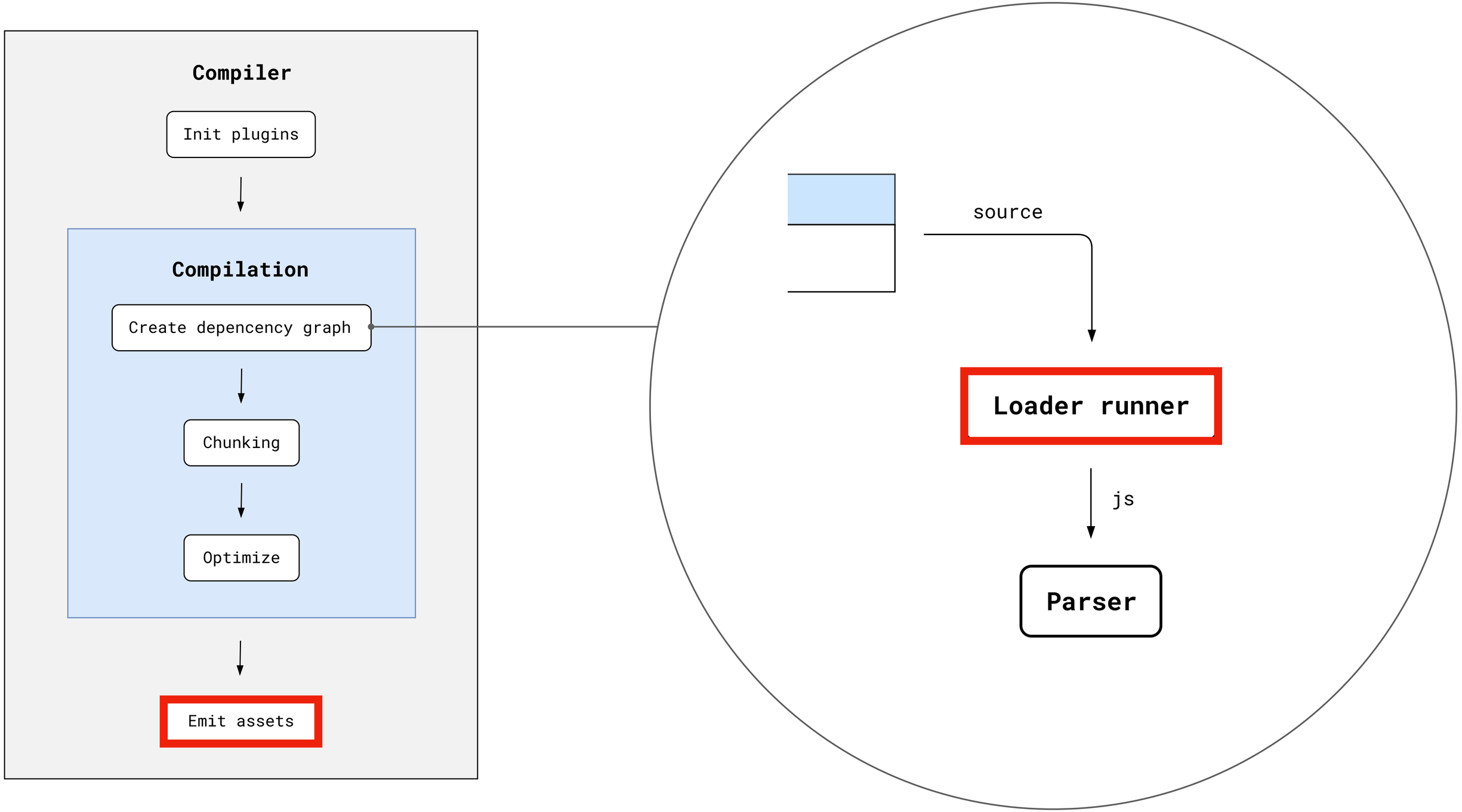
css-loader, который использует PostCSS

```
function cssLoader(css, sourcemap, meta) {

  const ast = meta && meta.ast && meta.ast.type === 'postcss'
    ? meta.ast.root
    : postcss.parse(content, { from });

  const result = transformCss(ast);

  return `module.exports = ${result}`;
}
```

Проблема

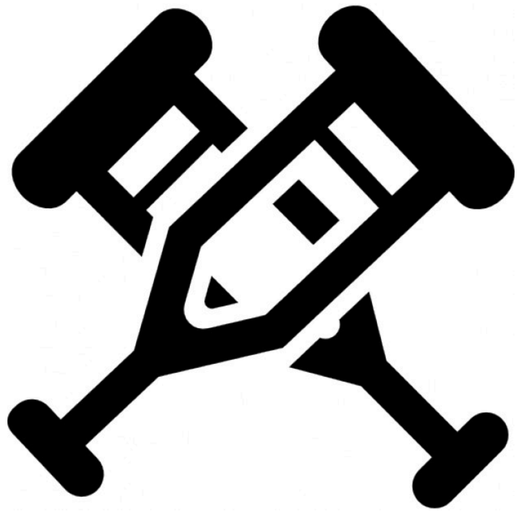
```
.img {
   background: url('sprite.svg') no-repeat 0 0;
   background-size: 100% 104.50%;
}
```

Узнать координаты символа можно только при создания спрайта (т.е. в плагине), когда этот лоадер уже отработал


Compiler
Init plugins
Compilation
Create depencency graph
Chunking
Optimize
Emit assets
source
Loader runner
js
Parser

И что делать?

Творческий подход!



Вместо настоящих значений лоадер вставляет метки

```
.img {
  background-image: url(image.svg);
}
```

```
.img {
  background: url('sprite.svg') no-repeat _d27f9_ _fb139_;
  background-size: _8a3e5_ _c0367_;
}
```

А плагин заменяет метки на реальные значения

```
.img {
  background: url('sprite.svg') no-repeat _d27f9_ _fb139_;
  background-size: _8a3e5_ _c0367_;
}
```

```
.img {
  background: url('sprite.svg') no-repeat 0 0;
  background-size: 100% 104.50%;
}
```

Ура, получилось?

А что с перекрашиванием SVG?

```
.img {
  background-image: url(image.svg);
}
```

```
<svg><use xlink:href="#icon" fill="blue"></use></svg>
<svg><use xlink:href="#icon" fill="red"></use></svg>
<svg><use xlink:href="#icon" fill="#ddf"></use></svg>
```

Ещё один лоадер для SVG!

Параметризация импортов

В webpack есть малоиспользуемая концепция resource query (аналог query string из URL), которая позволяет передать параметры в лоадер при обработке импортированного файла

```
import './foo?param=value';
```

```
.image {
  background: url(image.png?param=value)
}
```

Resource query

```
.image {
  background: url(image.png?fill=red)
}
```

```
function loader(svg) {
  const params = parseQuery(this.resourceQuery);
  transformSvg(svg, params.fill);
}
```

SVG стали перекрашиваться!

Но хотелось поудобнее

```
.image {
  background: url(image.svg);
  -svg-fill: red;
}
```

PostCSS!

```
.image {
  background: url(image.png?fill=red)
}
```

Ну уж теперь-то всё?

Итого

```
{
  test: /\.svg$/,
  loader: SpritePlugin.loader
}
```

```
plugins: [
   new SpritePlugin()
]
```

```
{
  test: /\.css$/,
  use: [
    'css-loader',
    SpritePlugin.cssLoader
  ]
}
```

```
{
  test: /\.svg$/,
  loader: 'svg-transform-loader'
}
```

Контакты и ссылки

Стас Курилов – github.com/kisenka, qtuzov@gmail.com

Лоадеры и плагины – github.com/JetBrains/svg-mixer

Доклад Ларкина https://youtu.be/4tQiJaFzuJ8

How webpack works https://youtu.be/CA-upQKYjYc

Спасибо за внимание!