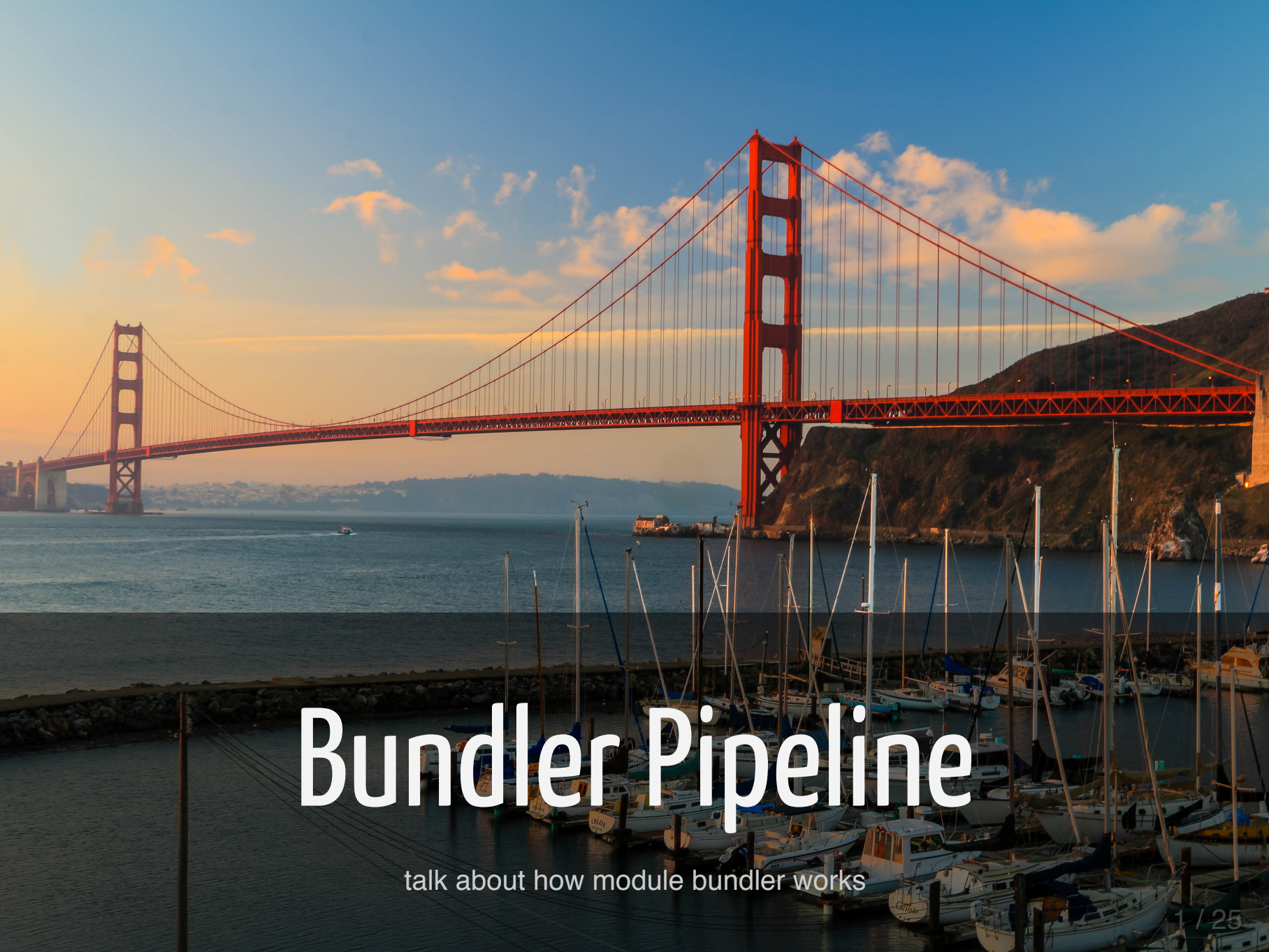


Bundler Pipeline

talk about how module bundler works

What is module bundling

On a high level, module bundling is simply the process of stitching together a group of modules (and their dependencies) into a single file (or group of files) in the correct order.

Why

Why

Problems:

- hard to balance between modularity and http requests
- plenty of tools in src-dist lifecycle
- low productivity because of many repeated works

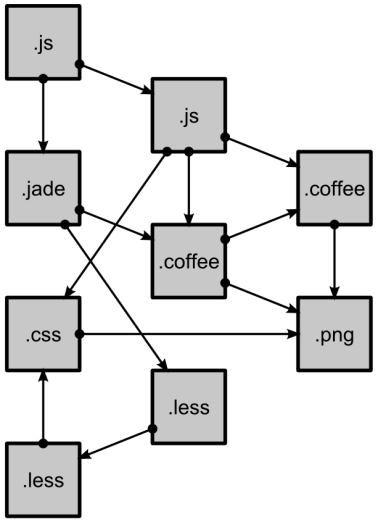
What we want

- Modularity & Performance, mainly for separation of concerns
- Integration, aggregate all dirty things
- Automation, DRY
- Fast

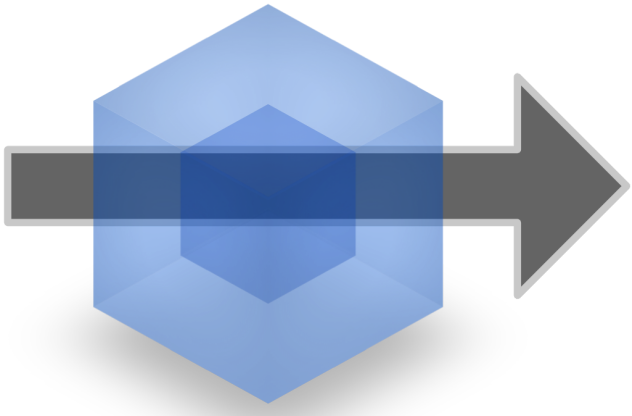
Bundler Solutions

- Webpack
- Browserify
- JSpm
- FIS
- ...

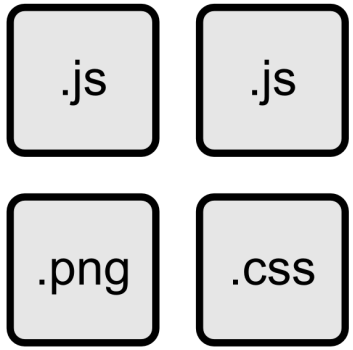
Webpack



modules
with dependencies



webpack
MODULE BUNDLER



static
assets

Design

- based on a plugin system
- use loader to compile source code
- all in one
- configuration driven, like Grunt


CASE IH

Pipeline

```
["run-async", "watch-run-async", "compilation",
  "normal-module-factory", "context-module-factory",
  "compile", "make-parallel", "after-compile-async",
  "emit-async", "after-emit-async", "done", "failed",
  "invalid", "after-plugins", "after-resolvers",
  "normal-module-loader", "seal", "optimize",
  "optimize-tree-async", "optimize-modules",
  "after-optimize-modules", "optimize-chunks",
  "after-optimize-chunks", "revive-modules",
  "optimize-module-order", "optimize-module-ids",
  "after-optimize-module-ids", "record-modules",
  "revive-chunks", "optimize-chunk-order", "optimize-chunk-ids",
  "after-optimize-chunk-ids", "record-chunks", "before-hash",
  "after-hash", "before-chunk-assets", "additional-chunk-assets",
  "record", "optimize-chunk-assets-async", "after-optimize-chunk-assets"
  "optimize-assets-async", "after-optimize-assets", "build-module",
  "succeed-module", "failed-module", "module-asset", "chunk-asset"]
```

Pipeline

```
["run-async", "watch-run-async", "compilation",
  "normal-module-factory", "context-module-factory",
  "compile", "make-parallel", "after-compile-async",
  "emit-async", "after-emit-async", "done", "failed",
  "invalid", "after-plugins", "after-resolvers",
  "normal-module-loader", "seal", "optimize",
  "optimize-tree-async", "optimize-modules",
  "after-optimize-modules", "optimize-chunks",
  "after-optimize-chunks", "revive-modules",
  "optimize-module-order", "optimize-module-ids",
  "after-optimize-module-ids", "record-modules",
  "revive-chunks", "optimize-chunk-order", "optimize-chunk-ids",
  "after-optimize-chunk-ids", "record-chunks", "before-hash",
  "after-hash", "before-chunk-assets", "additional-chunk-assets",
  "record", "optimize-chunk-assets-async", "after-optimize-chunk-assets"
  "optimize-assets-async", "after-optimize-assets", "build-module",
  "succeed-module", "failed-module", "module-asset", "chunk-asset"]
```

hard to understand

Pipeline

- load
- seal
- optimize
- chunk
- hash
- restore

Loader

Loaders are transformations that are applied on a resource file of your app. They are functions (running in node.js) that take the source of a resource file as the parameter and return the new source.

Loader

Loaders are transformations that are applied on a resource file of your app. They are functions (running in node.js) that take the source of a resource file as the parameter and return the new source.

```
require('jade!./template.jade');

// multi loaders, transformed from right to left
require('!style!css!less!bootstrap/less/bootstrap.less');
```

Plugin

Plugins expose the full potential of the Webpack engine to third-party developers. Using staged build callbacks, developers can introduce their own behaviors into the Webpack build process.

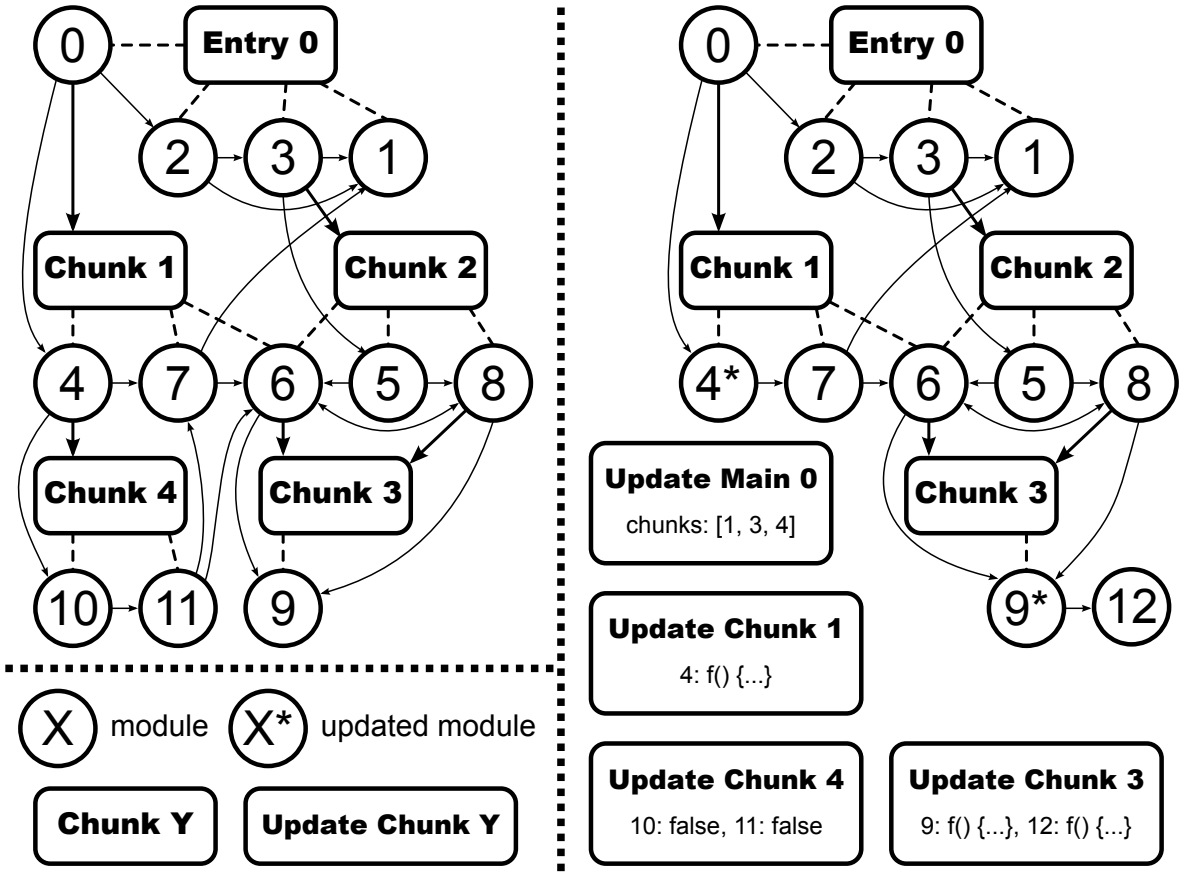
Plugin

Plugins expose the full potential of the Webpack engine to third-party developers. Using staged build callbacks, developers can introduce their own behaviors into the Webpack build process.

```
function WebpackMd5Hash () {}

WebpackMd5Hash.prototype.apply = (compiler) => {
  compiler.plugin("compilation", (compilation) => {
    compilation.plugin("chunk-hash", (chunk, chunkHash) => {
      const source = chunk.modules.sort(compareModules)
        .map(getModuleSource)
        .reduce(concatenateSource, '');
      const chunk_hash = md5(source);
      chunkHash.digest = () => chunk_hash;
    });
  });
};
```

HMR


0
Entry 0
2
3
1
Chunk 1
Chunk 2
4
7
6
5
8
Chunk 4
Chunk 3
10
11
9
X
module
X*
updated module
Chunk Y
Update Chunk Y
0
Entry 0
2
3
1
Chunk 1
Chunk 2
4*
7
6
5
8
Chunk 3
9*
12
Update Main 0
chunks: [1, 3, 4]
Update Chunk 1
4: f() {...}
Update Chunk 4
10: false, 11: false
Update Chunk 3
9: f() {...}, 12: f() {...}

Code Splitting

Code Splitting is not just about extracting common code into a shared chunk. The more notable feature is that Code Splitting can be used to split code into an on demand loaded chunk.

```
var a = require("a");
var b = require("b");
require.ensure(["c"], function(require) {
  require("b").xyz();
  var d = require("d");
});
```

Problem

so COMPLEX to find the right configuration

Browserify

Design

- based on Stream
- Unix on browser
- code driven, like Gulp

Pipeline

```
var pipeline = splicer.obj([
  'record', [ this._recorder() ],
  'deps', [ this._mdeps ],
  'json', [ this._json() ],
  'unbom', [ this._unbom() ],
  'unshebang', [ this._unshebang() ],
  'syntax', [ this._syntax() ],
  'sort', [ depsSort(dopts) ],
  'dedupe', [ this._dedupe() ],
  'label', [ this._label(opts) ],
  'emit-deps', [ this._emitDeps() ],
  'debug', [ this._debug(opts) ],
  'pack', [ this._bpack ],
  'wrap', []
]);
```

With Gulp

```
var stream = browserify.bundle()
  .pipe(addsrc(standaloneFiles, { base: clientDir }))
  .pipe(dedupe())
  .pipe(hold())
  .pipe(gulp.dest(staticDir))
  // stamp version
  .pipe(imagemin())
  .pipe(stampImageVersion)
  .pipe(hold())
  .pipe(rewriteStyleUrl)
  .pipe(stampOtherVersion)
  .pipe(gulp.dest(staticDir))
  // compress
  .pipe(stampMin)
  .pipe(cssmin())
  .pipe(filterScript)
  .pipe(uglify())
  .pipe(filterScript.restore)
  .pipe(gulp.dest(staticDir));
```

References

- webpack
- Exploring WebPack
- browserify
- Choosing the correct packaging tool for React JS

Thanks