

How

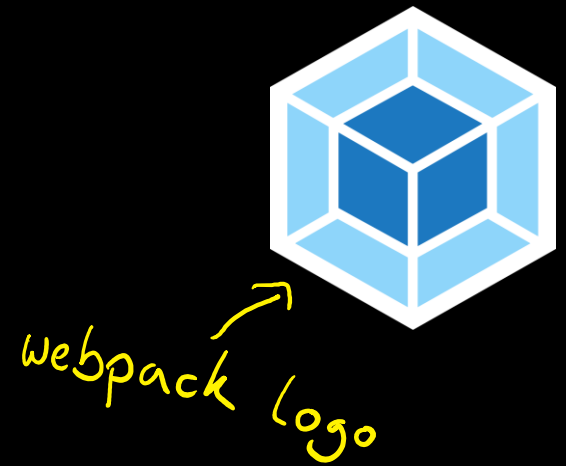
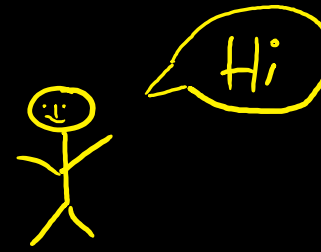


**webpack**

works

Tobias Koppers

# About me



- Tobias Koppers
  - M. Sc. Computer Science
  - Started with Open Source in 2012 as Hobby
  - Since 2017: Freelancer (Open Source + Consulting)
  - Father of a 2-months old daughter
- 
- Twitter: @wSokra
  - Github: @sokra



# Agenda

- Walkthrough webpack
  - You should get a rough understanding of the implementation of webpack.
  - You should know where to look into when customizing.
  - We will not go into details of individual optimization algorithms.
  - Some unimportant details are omitted for simplicity.
  - Some points for customization are highlighted.
- Customization
  - You will learn how to write plugins for webpack.
  - We won't list every possible customizable thing.



Plugins

```
class MyPlugin {  
  apply(compiler) {  
    compiler.plugin("done", (stats) => {  
      console.log(stats.toString());  
    })  
  }  
}
```

Handwritten annotations for the code above:

- Name**: points to `MyPlugin`
- Hook**: points to `"done"`
- Arguments**: points to `(stats)`
- Code**: points to the lambda function `(stats) => { console.log(stats.toString()); }`

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

**Usage**: points to the array `[ new MyPlugin() ]`



# Access to nested objects

```
class MyPlugin {  
  apply(compiler) {  
    compiler.plugin("compilation", compilation => {  
      compilation.plugin("optimize-modules", modules => {  
        modules.forEach(...);  
      }  
    }  
  }  
}
```

*Hook* (points to "compilation" in the first plugin call)

*Object* (points to "compilation" in the second plugin call)

*Tapable* (points to the "compilation" parameter in the first plugin call)



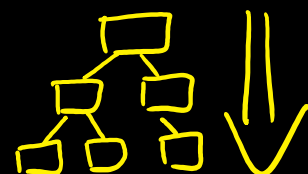
# Hooks

- There are different kind of hooks
  - sync
  - async (with callback argument)
    - sequential
    - parallel
    - waterfall (passing result to next plugin)
- Check details in code (not all hooks are documented)

• Don't forget to call the callback!  
Elsewise process will exit unexpectedly.



# Walkthrough webpack

- webpack is very object-orientated
- Design is probably not perfect
- We start our walkthrough from top down to bottom 
- Start: Invoking the CLI: `webpack entry.js bundle.js`



# Command Line Interface

- Get arguments from command line *npm yargs*
- Read webpack.config.js
- Invoke exported value if it's a function
- Convert command line arguments into configuration
- Call the webpack API with the configuration
- bin/webpack.js, bin/convert-argv.js
- Can't be customized





# API / webpack facade

- Validates configuration according to schema
- Applies the node.js environment plugins
  - Input/Output/Watch FileSystem for node.js
- Calls WebpackOptionsApply to process configuration
- Creates the Compiler
- May call run or watch on the Compiler
- Exports all public plugins to use them in the configuration



# WebpackOptionsApply

- Converts all configuration options into plugins
- Applies default plugins
- Examples:
  - `output.library` → `LibraryTemplatePlugin`
  - `externals` → `ExternalsPlugin`
  - `devtool` → `EvalDevToolModulePlugin`, `SourceMapDevToolPlugin`, ...
  - `AMDPlugin`
  - `CommonJsPlugin`
  - `RemoveEmptyChunksPlugin`
  - `MergeDuplicateChunksPlugin`

Everything  
is  
a  
plugin



# Compiler

- run
  - compile
    - creates Compilation
  - emit
    - writes assets
- watch → Watching
  - run
  - watch dependencies → run again

- All plugins are attached to this instance

# Compiler hooks



process

- (before-)run
- (before-/after-)compile
- make
- (after-)emit
- done

- watch-run
- invalid
- watch-close

watch

- compilation  
→ Compilation
- normal-module-factory  
→ NormalModuleFactory
- context-module-factory  
→ ContextModuleFactory

nested



# Compilation

- Compiler calls this lifecycle methods of the Compilation:

make hook → • addEntry → addModuleChain

- finish
  - report errors from modules

- seal



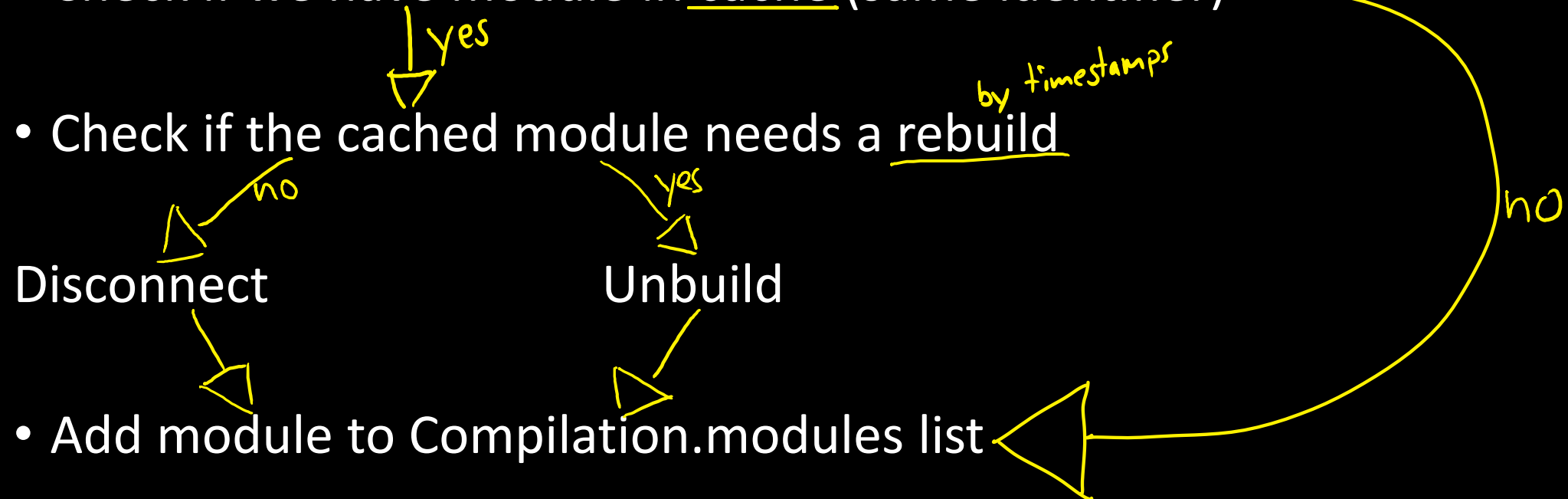
# Compilation.addModuleChain

- input is a Dependency
- get the ModuleFactory for the Dependency
- ModuleFactory.create → returns Module
- Compilation.addModule
- Compilation.buildModule
  - Calls Module.build
- Compilation.processModuleDependencies
  - foreach unique Dependency in Module



# Compilation.addModule

- input is a Module from ModuleFactory
- Return if Module is already in the Compilation (same identifier)
- Check if we have module in cache (same identifier)





# Compilation.seal

chunk  
graph

- foreach entrypoint
  - Compilation.addChunk
  - Chunk.addModule
  - Compilation.processDepBlock-ForChunk

- Optimize
  - Modules
  - Chunks
  - Tree
  - Chunk Modules

- Sort

plugin  
hooks

- Ids
  - Modules
  - Chunks
- Record
- Hash

- Assets
  - Chunk assets
  - Additional Assets
  - Optimize Assets

assets





# Compilation Hooks

## modules

- build-module
- failed-module
- succeed-module

- finish-modules

- chunk-hash

hashing

- module-asset
- chunk-asset

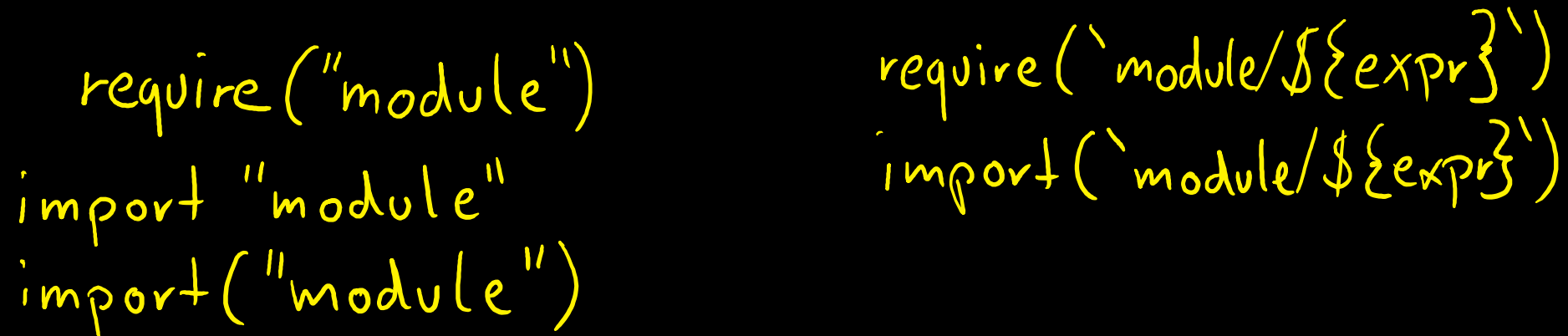
asset  
generation

## seal & optimize

- (after-)seal
- optimize
- optimize-modules (-basic/advanced)
- after-optimize-modules
- optimize-chunks (-basic/advanced)
- after-optimize-chunks
- (after-)optimize-tree
- optimize-chunk-modules (-basic/advanced)
- after-optimize-chunk-modules
- optimize-module/chunk-order
- before-module/chunk-ids
- (after-)optimize-module/chunk-ids
- before/after-hash
- additional-(chunk-)assets
- (after-)optimize-(chunk-)assets

...

# ModuleFactory



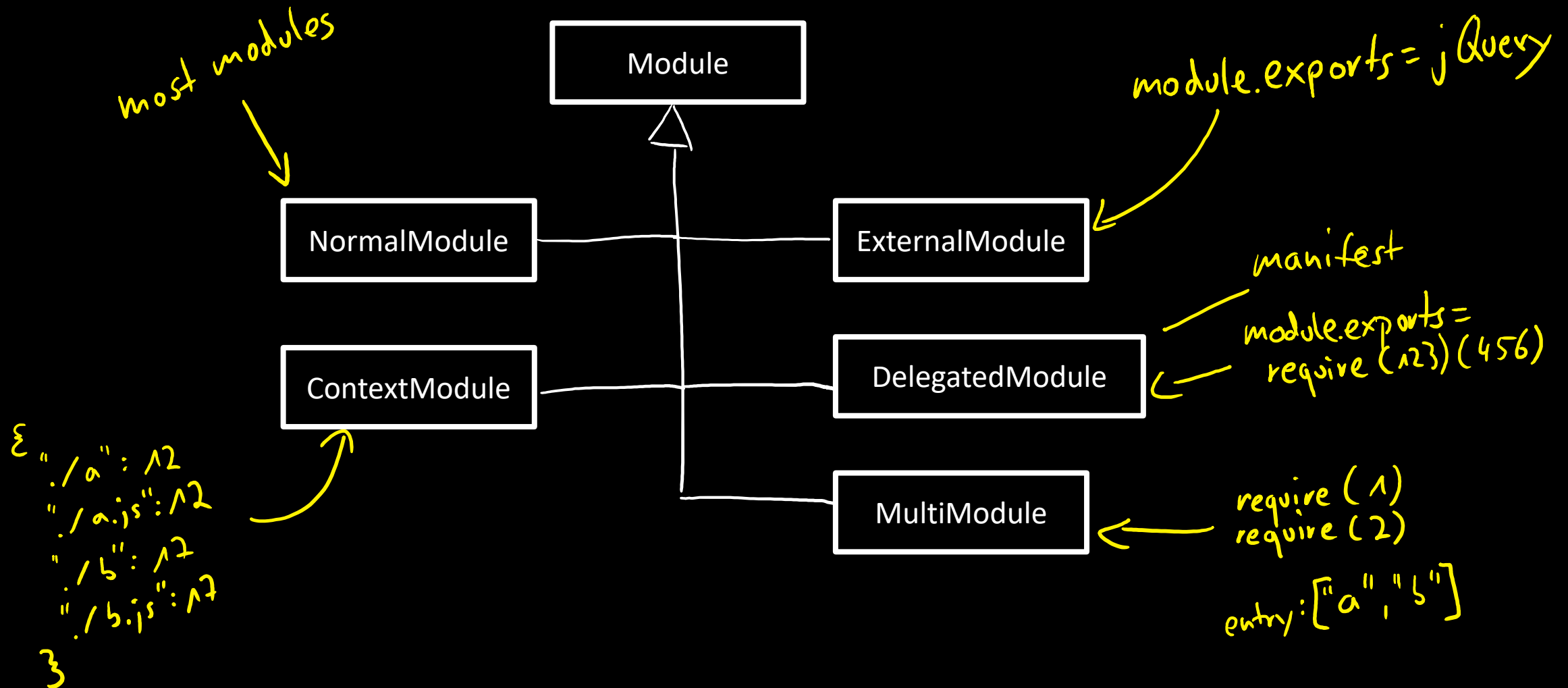


# NormalModuleFactory

npm

- Resolve request (using enhanced-resolve)
- Process rules from RuleSet (*module.rules*)
- Resolve remaining requests i. e. loaders
- Instantiate NormalModule
- Plugins may alter this process
  - Returning other kind of Modules (i. e. externals, dll, ignore)
  - Overriding requests

# Module





# NormalModule

- Build
  - Run loaders using loader-runner
  - Store returned Code
  - Parse using Parser
  - ParserPlugins add dependencies to the DependencyBlock/Module
- Get Source (Code Generation)
  - Create a ReplaceSource on top of the stored Code
  - Get DependencyTemplate for each Dependency
  - Apply templates for Dependency on ReplaceSource
    - Templates do string replacements/insertings

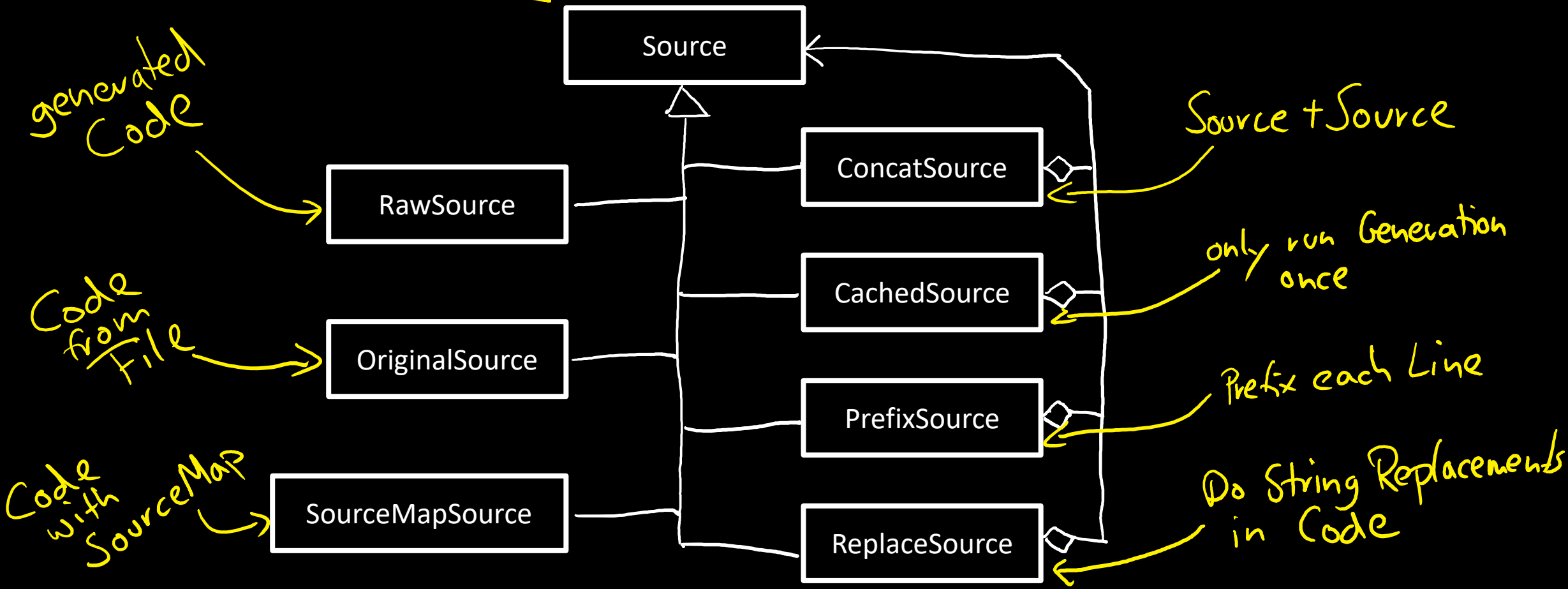
*npm*

npm  
webpack-sources

# Source



get { code  
SourceMap } with column mapping  
without column mapping



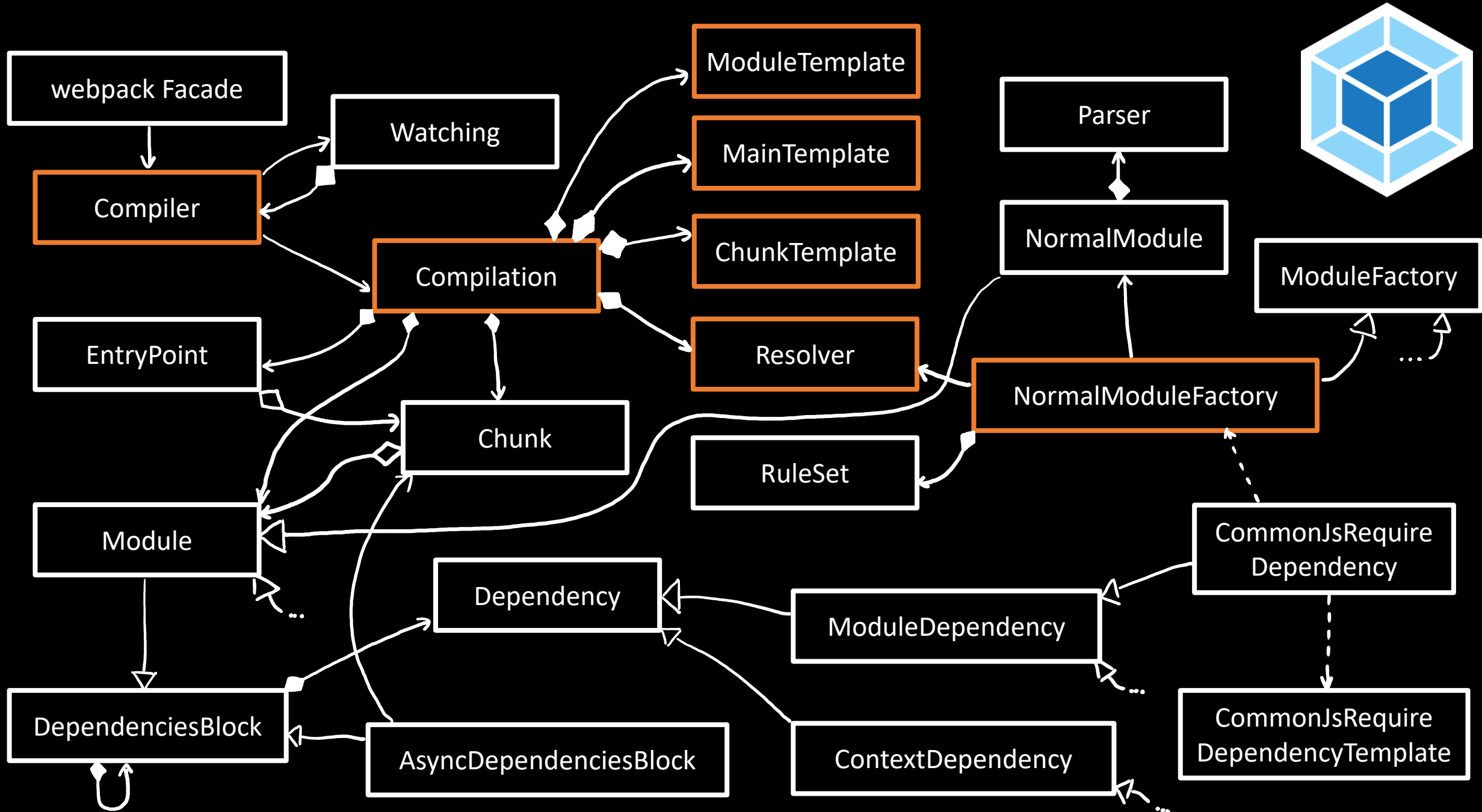


# Compilation.createAssets

- MainTemplate
  - ChunkTemplate
  - ModuleTemplate
- yes*  
*no* entry chunk?
- foreach Module*

*plugins modify behavior*  
*i.e. JsonpChunkTemplatePlugin*  
*SetVarMainTemplatePlugin*

*→ Sources*







# More Info

- Source Code
  - [github.com/webpack/webpack](https://github.com/webpack/webpack)
  - [github.com/webpack/webpack-sources](https://github.com/webpack/webpack-sources)
  - [github.com/webpack/enhanced-resolve](https://github.com/webpack/enhanced-resolve)
  - [github.com/webpack/loader-runner](https://github.com/webpack/loader-runner)
- Documentation
  - [webpack.js.org](https://webpack.js.org)

Read  
the  
Source!



# Funding

- webpack needs your help. We are funded by donations. Sponsor?



≥ \$10,000

≥ \$1,000

\$100  
to get into  
the list