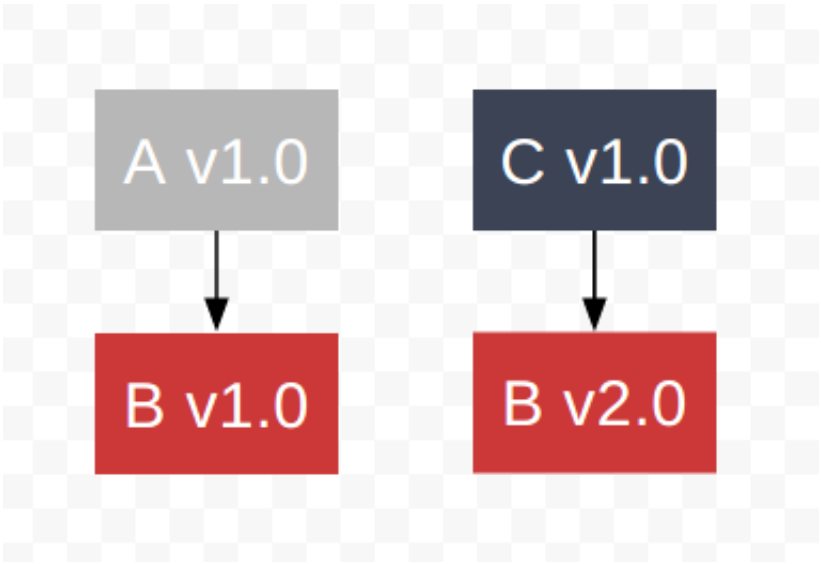
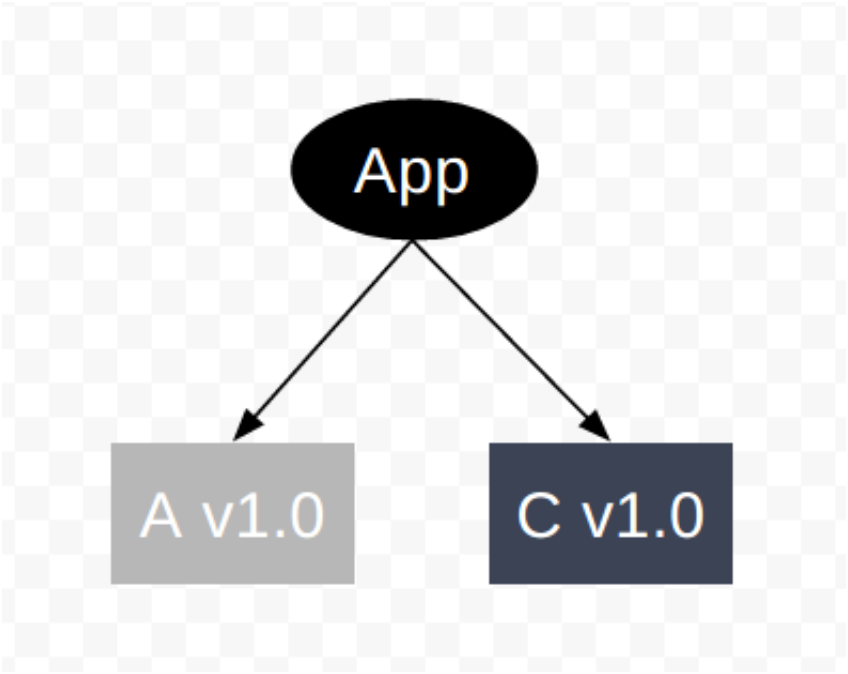
How npm2 Works

Imagine there are three modules: A, B, and C. A requires B at v1.0, and C also requires B, but at v2.0. We can visualize this like so:



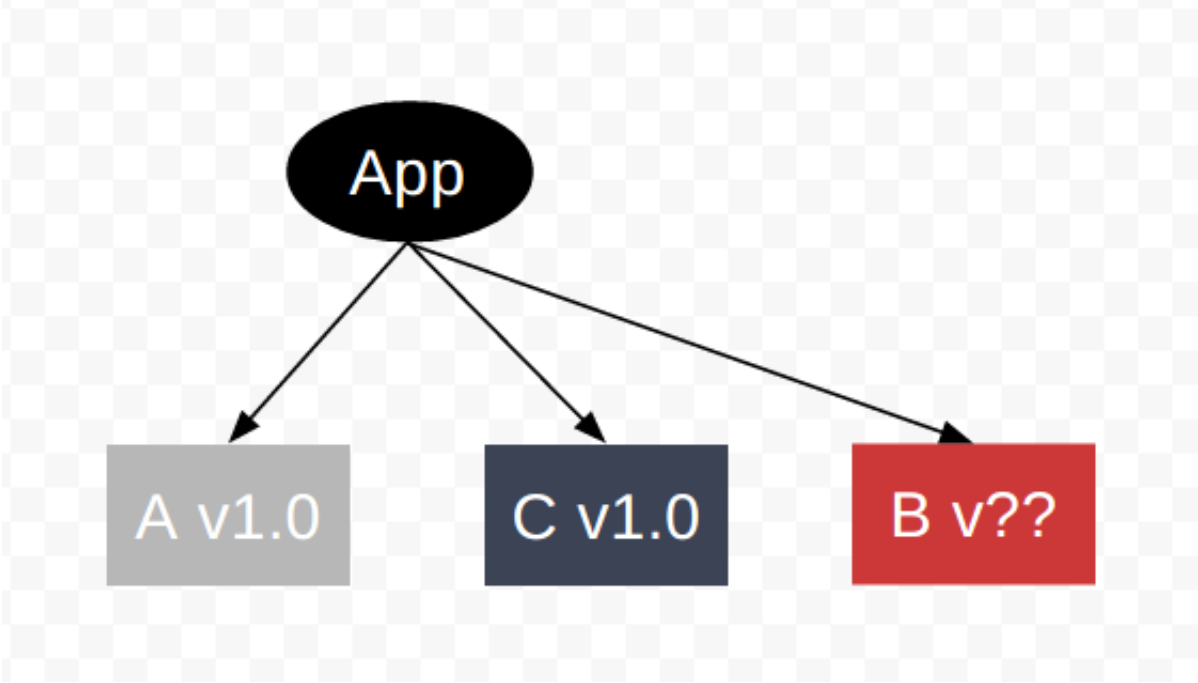
Now, let's create an application that requires both module A and module C.



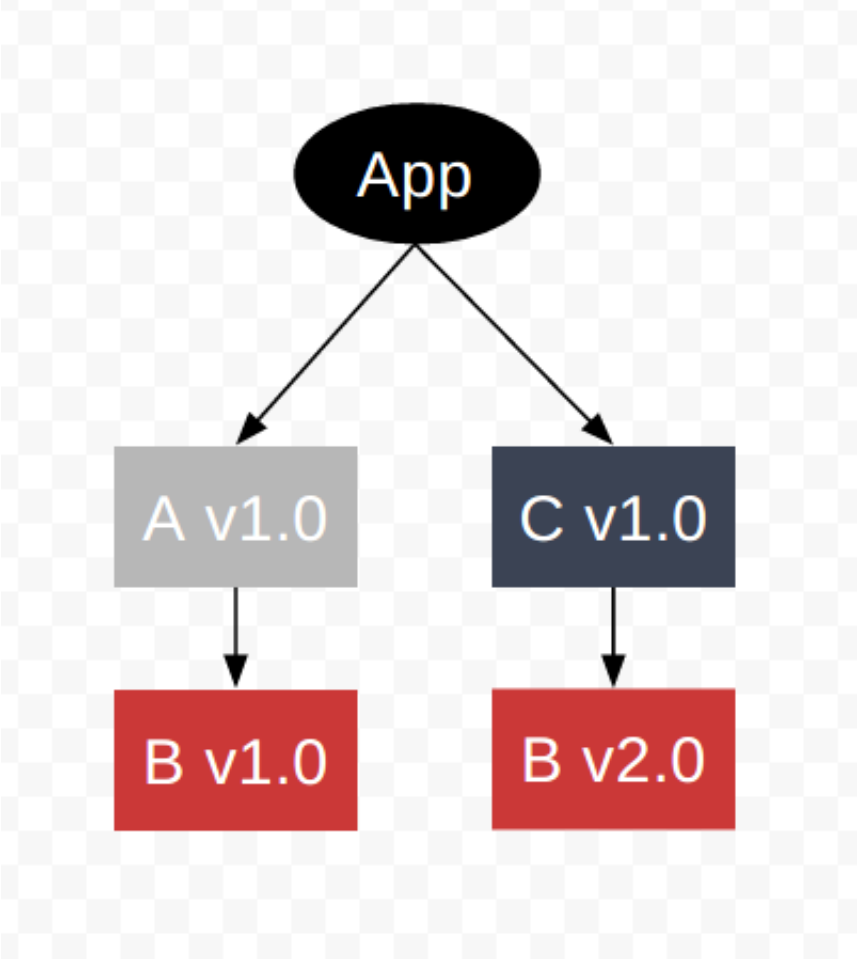
Dependency Hell

A package manager would need to provide a version of module B. In all

other runtimes prior to Node.js, this is what a package manager would try to do. This is dependency hell:



Instead of attempting to resolve module B to a single version, npm puts both versions of module B into the tree, each version nested under the module that requires it.



In the terminal, this looks like this:

```
(jessie)ag_dubs@localhost:~/Projects/npm-sandbox/npm2/example1$ npm install mod-a --save
npm WARN package.json example1@1.0.0 No repository field.
mod-a@1.0.0 node_modules/mod-a
└── mod-b@1.0.0
(jessie)ag_dubs@localhost:~/Projects/npm-sandbox/npm2/example1$ npm install mod-c --save
npm WARN package.json example1@1.0.0 No repository field.
mod-c@1.0.0 node_modules/mod-c
└── mod-b@2.0.0
(jessie)ag_dubs@localhost:~/Projects/npm-sandbox/npm2/example1$ tree -d node_modules/
node_modules/
├── mod-a
│   └── node_modules
│       └── mod-b
└── mod-c
    └── node_modules
        └── mod-b
```

You can list the dependencies and still see their relationships using `npm ls`:

```
(jessie)ag_dubs@localhost:~/Projects/npm-sandbox/npm2/example1$ npm ls
example1@1.0.0 /home/ag_dubs/Projects/npm-sandbox/npm2/example1
├─┬ mod-a@1.0.0
│ └── mod-b@1.0.0
└─┬ mod-c@1.0.0
  └── mod-b@2.0.0
```

If you want to just see your primary dependencies, you can use:

```
npm ls --depth=0
```

```
(jessie)ag_dubs@localhost:~/Projects/npm-sandbox/npm2/example1$ npm ls --depth=0
example1@1.0.0 /home/ag_dubs/Projects/npm-sandbox/npm2/example1
├── mod-a@1.0.0
└── mod-c@1.0.0
```

npm and the Node.js Module Loader

However, npm doing this is *not enough*. Despite the fact that their nested locations allow for the coexistence of two versions of the same module, most module loaders are unable to load two different versions of the same module into memory. Luckily, the Node.js module loader is written for exactly this situation, and can easily load both versions of the module in a way that they do not conflict with each other.

How is it that npm and the node module loader are so wonderfully symbiotic? They were both written in large part by the same person, npm, Inc. CEO, Isaac Z. Schlueter. Like 2 sides of the same piece of paper, npm and the Node.js module loader are what make Node.js a uniquely well-suited runtime for dependency management.