Modules

JS

Task Runner Tools

Package Management

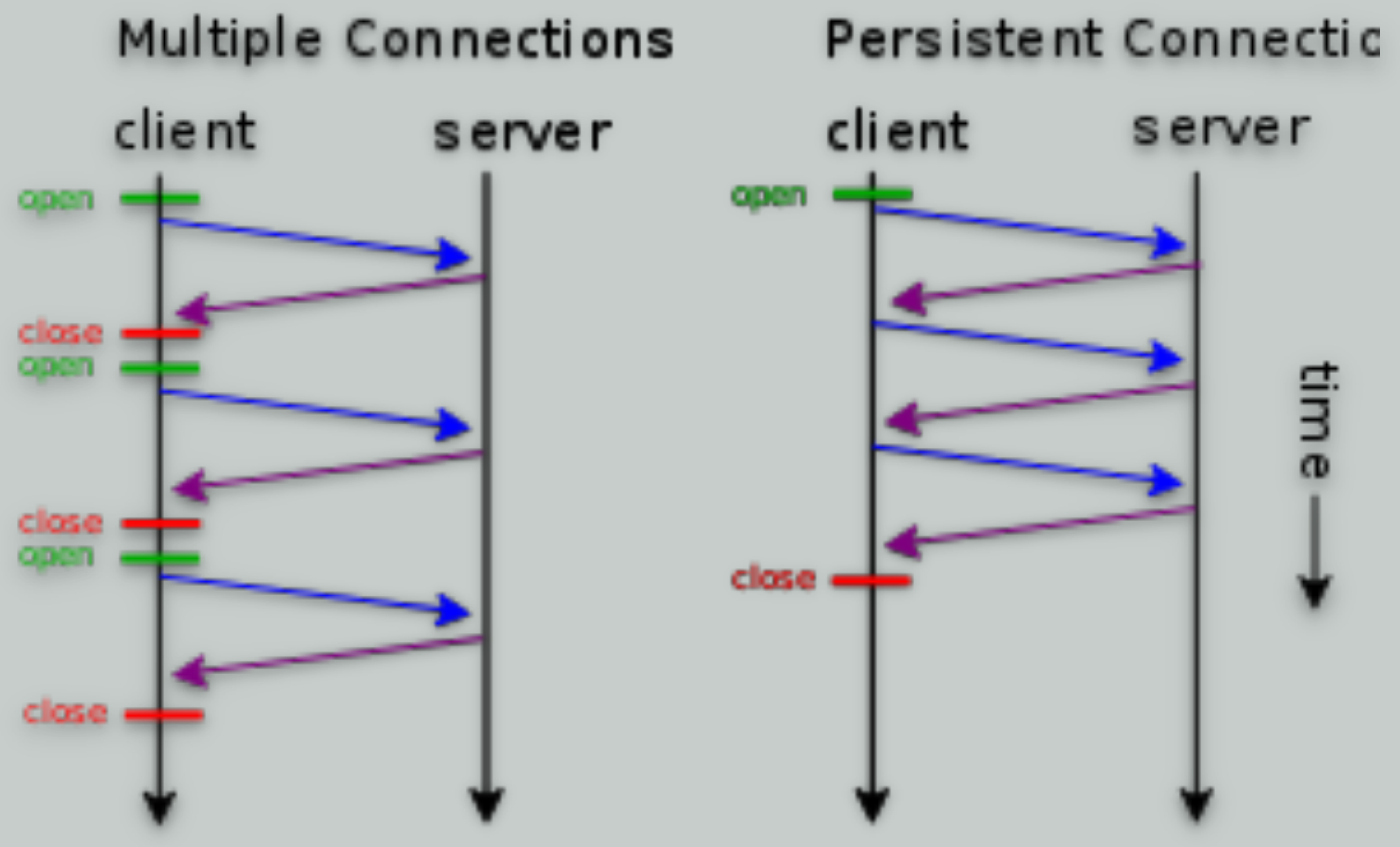
Developing JavaScript With Traditional Techniques

1	Inline Style	Complex and crowded pages
2	Global Scope	Variable problem named with the same name
3	Namespace	Difficult to manage dependencies
4	Traditional Module Pattern	No way to programmatically import modules (except by using eval). Dependencies need to be handled manually. Asynchronous loading of modules is not possible. Circular dependencies can be troublesome. Hard to analyze for static code analyzers.

Bundling And Minification

Http 1.0

Http 1.1 (Connection Keep Alive)



Bundling And Minification

The time required to render a web page mainly depends on four factors

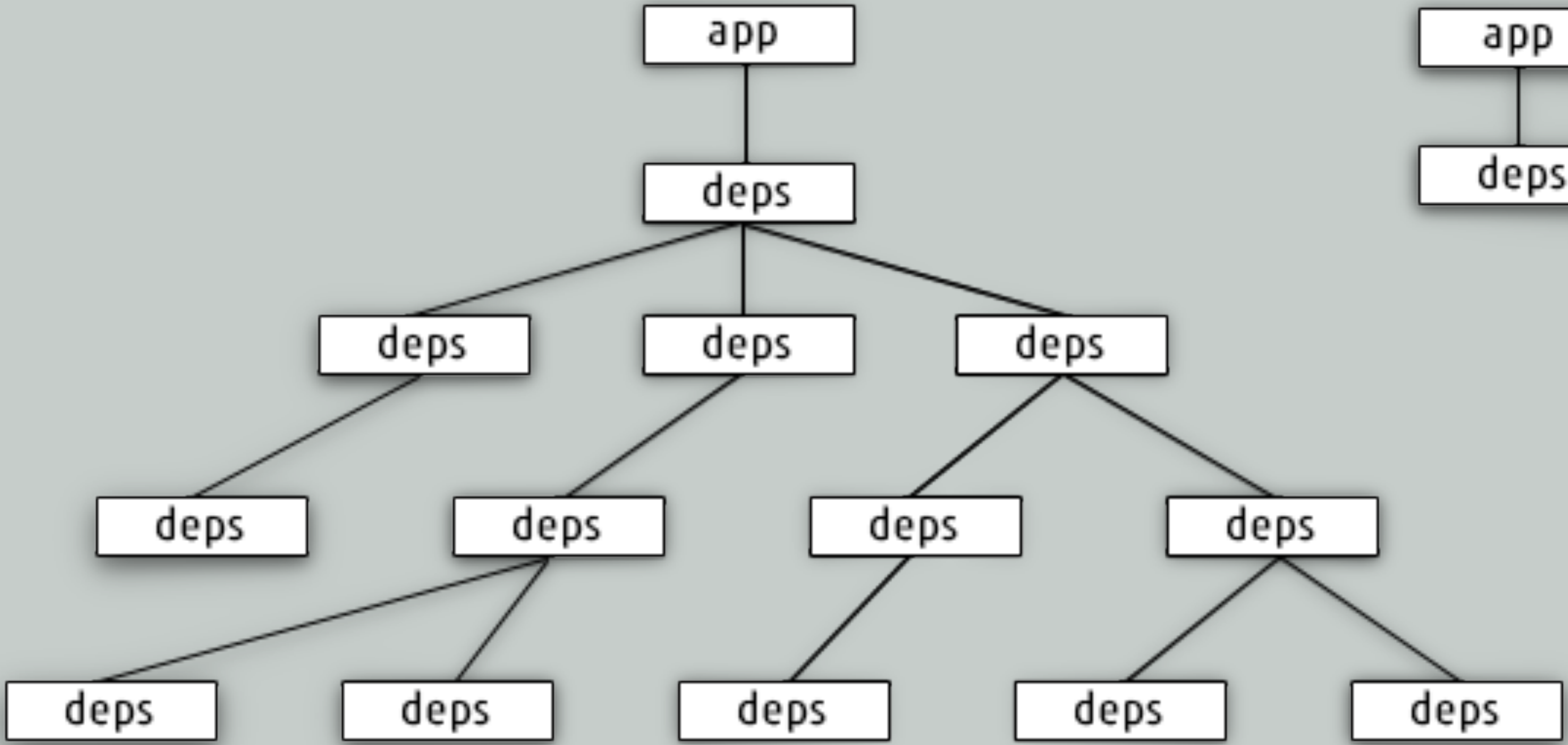
Network latency
Available bandwidth
Number of HTTP requests
Size of HTTP requests

PACKAGE MANAGEMENT TOOLS

	npm	Bower
What for	Commonly used for NodeJs Modules	Front end asset package management
	Nested dependency tree	Flat dependency tree
When to use	Great for server, space is not a concern	Great for front end, optimal size
	No dependency conflict	Can have dependency conflict
	Open Source (2009) (+/- 500.000 Package)	Twitter (2012) (+/- 100.000 Package)
Initialize	> npm init	> bower init
Search	> npm search <%package_name%>	> bower search <%package_name%>
File	package.json	bower.json
Command	npm install <package_name>	bower install <package_name>

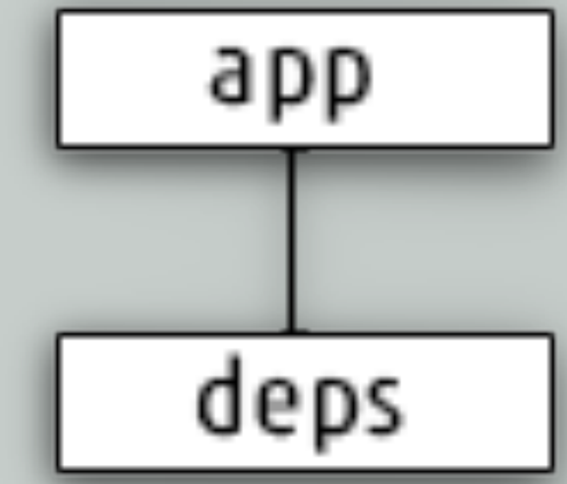
nested dependencies

npm (node_modules)



flat dependencies

(bower, gem, pip, jspm, etc)



JAVASCRIPT TASK RUNNER TOOLS

Gulp

Grunt

Brunch

As JavaScript development gets more and more complex, dependency management can get cumbersome.

Good modules, however, are highly self-contained with distinct functionality, allowing them to be shuffled, removed, or added as necessary, without disrupting the system as a whole.

Good authors divide their books into chapters and sections; good programmers divide their programs into modules.

Where did the modules come from

C#

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization;
using System.Text;
```

Java

```
import java.util.ArrayList;
import java.util.List;
import javax.ws.rs.core.Link;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
```

python™

```
import inspect
import textwrap
import typing
import coreapi
import coreschema
import uritemplate

from apistar import Route, Settings, exceptions, typesystem
from apistar.core import flatten_routes
from apistar.interfaces import Router, Schema
from apistar.types import RouteConfig
```

There is no standart way to implement packages lib vs like these in JavaScript

WHY USE MODULES ?

1	Maintainability	Updating a single module is much easier when the module is decoupled from other pieces of code.
2	Namespacing	Sharing global variables between unrelated code is a big fault in development.
3	Reusability	For example, let’s imagine you copied some utility methods you wrote from a previous project to your current project.That’s all well and good, but if you find a better way to write some part of that code you’d have to go back and remember to update it everywhere else you wrote it.
4	Decoupling	A well-designed module will provide an interface for external code to use. As the module gets updated with bug fixes and new functionality, the existing interface stays the same (it is stable) so that other modules can use the new, improved version without any changes to themselves.
5	Using With Module Loaders	Do not load unnecessary resources in vain

JAVASCRIPT MODULES

UMD

CommonJS	AMD	ES6
The project was started by Mozilla engineer Kevin Dangoor in January 2009 and initially named **ServerJS.** **The project was renamed CommonJS**	The Asynchronous Module Definition (AMD) API specifies a mechanism for defining modules such that the module and its dependencies can be asynchronously loaded. This is particularly well suited for the browser environment where synchronous loading of modules incurs performance, usability, debugging, and cross-domain access problems.	Browser support for ES2015 is still incomplete.
Node.js developers originally intended to follow the CommonJS specification but later decided against it. When it comes to modules, Node.js's implementation is very influenced by it	AMD started as a spinoff of the CommonJS Transport format and evolved into its own module definition API. Hence the similarities between the two. The new feature in AMD is the define() function that allows the module to declare its dependencies before being loaded.	However, ES6 code can be transpiled into ES5 code, which has more consistent support across browsers.

?

UMD

JAVASCRIPT MODULE LOADERS

webpack

Browserify

RequireJS