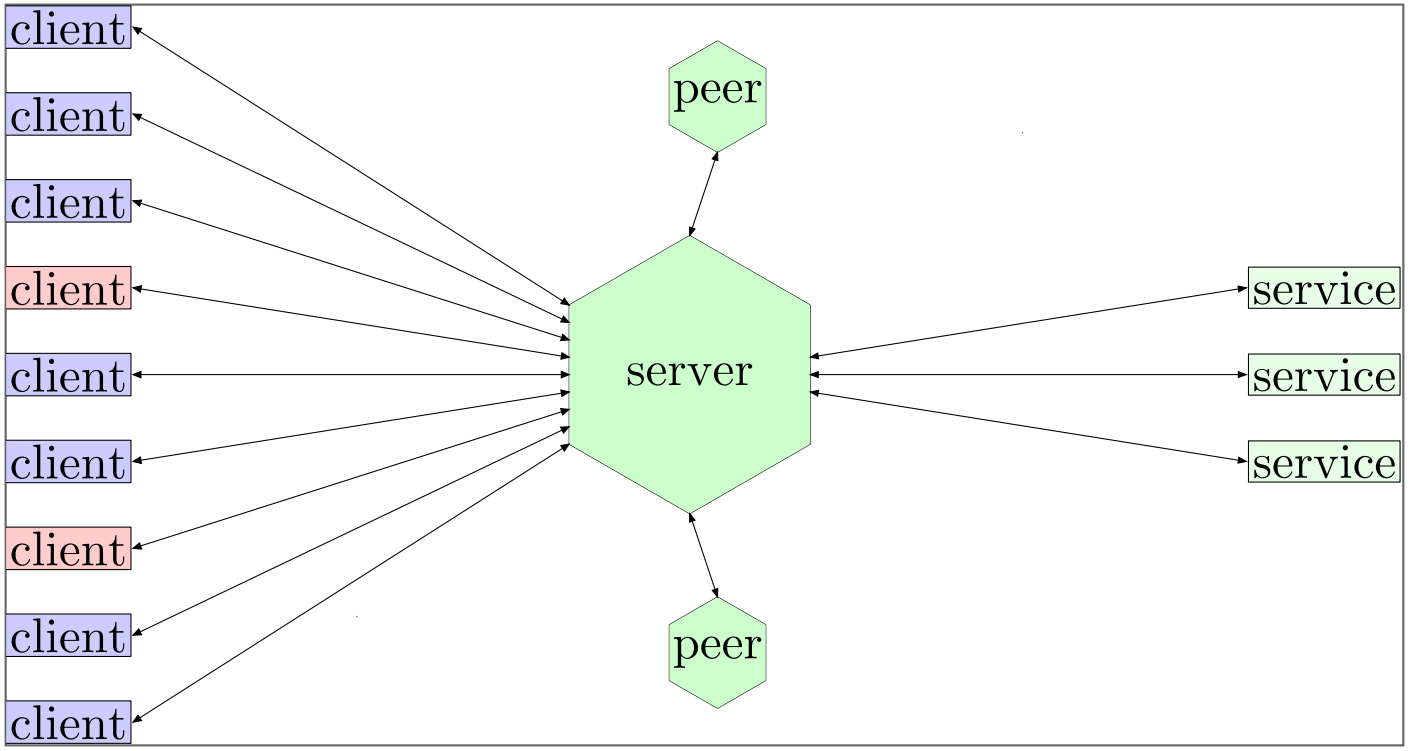
Node.js --- JavaScript on the server

- programming language
- communication
- choice
  - responsible trade-offs
  - meaningful options
  - low critical mass

Node.js --- JavaScript on the server

- free/open-source software, started 2009 by Ryan Dahl
- now developed by Joyent, StrongLoop (IBM), Google (V8)
- platform support: IDEs, debuggers...
- vibrant community; 200,000+ npm packages (Nov 2015)

Frictionless Communication



One process communicates with 2 - 1M other processes.

One process communicates with 2 - 1M other processes:

- web server: http, websockets
- IoT server: CoAP, MQTT
- web crawler
- proxy: load balancer, protocol translator
- peer-to-peer cluster; CDN
- (distributed) data base server
- process manager (e.g. D-bus)
- ...

familiar syntax and semantics

```
function factorial(n) {
  let p=1;
  for(let i=1; i<=n; ++i) { p*=i; }
  return p;
}
```

synchronous I/O (not in node.js)

sequential computation and sequential I/O:

```
print_body(http_get(url1));
print_body(http_get(url2));
print_body(http_get(url3));
console.log("done");
```

But how to fetch in parallel?

Multithreading with shared data??

- Parallel processing can corrupt shared data.
- Race conditions --- appear only under load, in production.
- Clocks can create deadlocks.
- ...

asynchronous I/O (node.js)

sequential computation and parallel communication:

```
http.get(url1, function(res){ print_body(res); });
http.get(url2, function(res){ print_body(res); });
http.get(url3, function(res){ print_body(res); });
console.log("done");
```

sequential input/output in node.js

```
http.get(url1, function(res1) {
  print_body(res1);

  http.get(url2, function(res2) {
    print_body(res2);

    http.get(url3, function(res3) {
      print_body(res2);

    });
  });
});
```

syntactic sugar for sequential I/O

```
print_body(await http_get(url1));
print_body(await http_get(url2));
print_body(await http_get(url3));
console.log("done.");
```

JSON --- simple, reliable data transfer

```
{ "request":"update-user-address",
  "user-id":564399,
  "user-address": {"street":"HaRav Kehaneman 14/10", city: "Jerusalem"}
}
```

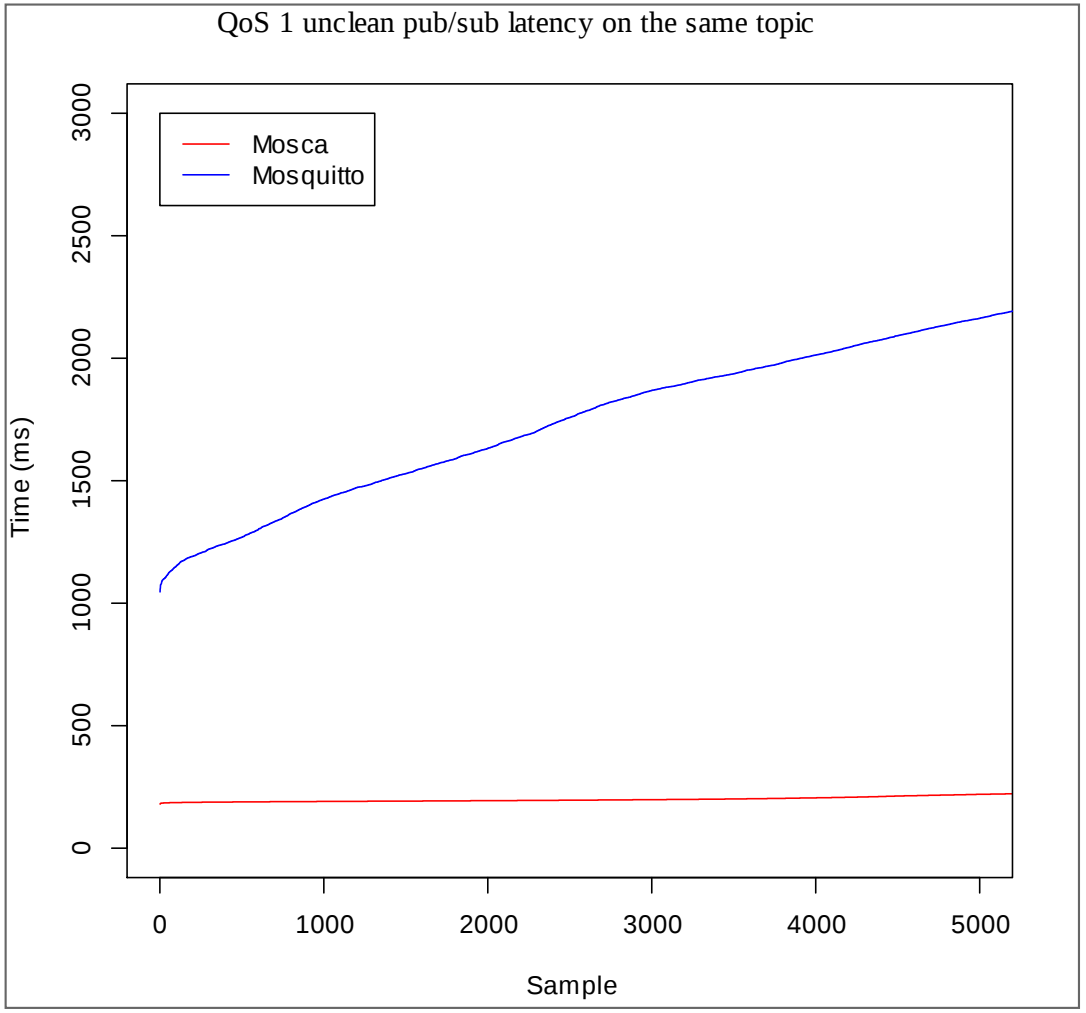
=>

```
{ "response": true }
```

or:

```
{ "error": "unknown_user",
  "error-data": {"user-id": 564399},
  "error-message": "This user is not known in the system."
}
```

Performance: Node.js can compete with C++



Trade-off: fast development vs. fast execution

Dynamic typing/objects and garbage collection:

- flexible APIs, flexible app structure
- generic functions

Faster development, slower execution.

- Save dev time on 80% of your code.
- Better optimize the critical 20%.

Optimize only what is important

- Algorithms: async is hard in C++ but easy in node.js.
- Profiling: Look at real data.

Optimize hot functions

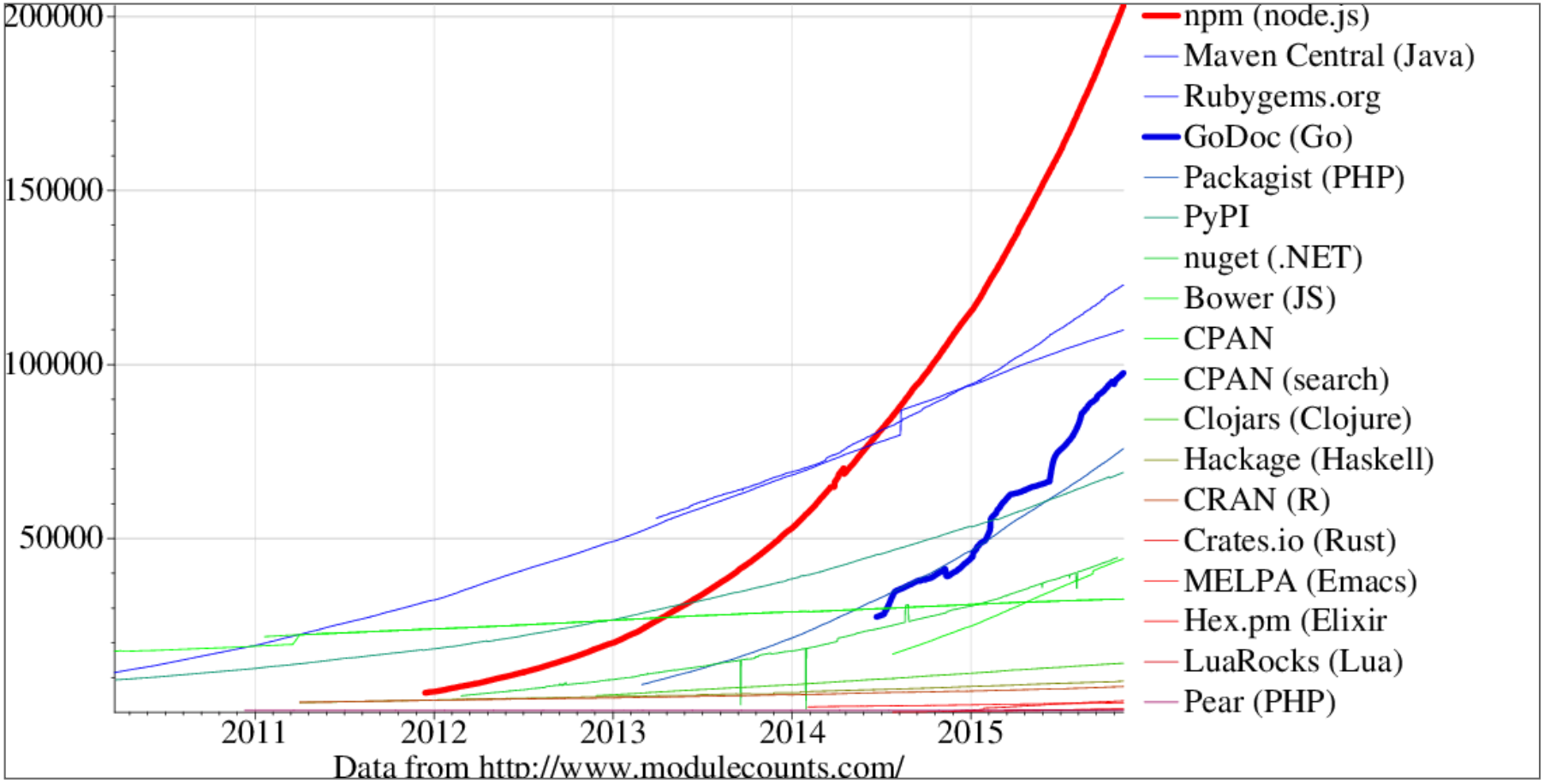
V8 can optimize your code close to C++ speed when you:

- Use objects with identical structure.
- Use arrays with values of the same type.
- Call functions with consistent argument types.

Check --trace-opt, --trace-bailout and --trace-deopt.

Last resort: write a C++ addon package.

Community


200000
150000
100000
50000
2011
2012
2013
2014
2015
Data from http://www.modulecounts.com/
npm (node.js)
Maven Central (Java)
Rubygems.org
GoDoc (Go)
Packagist (PHP)
PyPI
nuget (.NET)
Bower (JS)
CPAN
CPAN (search)
Clojars (Clojure)
Hackage (Haskell)
CRAN (R)
Crates.io (Rust)
MELPA (Emacs)
Hex.pm (Elixir
LuaRocks (Lua)
Pear (PHP)

npm

- It's easy to **use** packages.
- It's easy to **publish** packages.
- **It Just Works...**
  - even when packages change.

Your app depends on the package foo

It works well with foo v1.0.0.

But foo v2.0.0 is not backwards compatible.

Will your application break?

The npm solution:

Your app v1.0.0 depends on package foo v1.0.0.

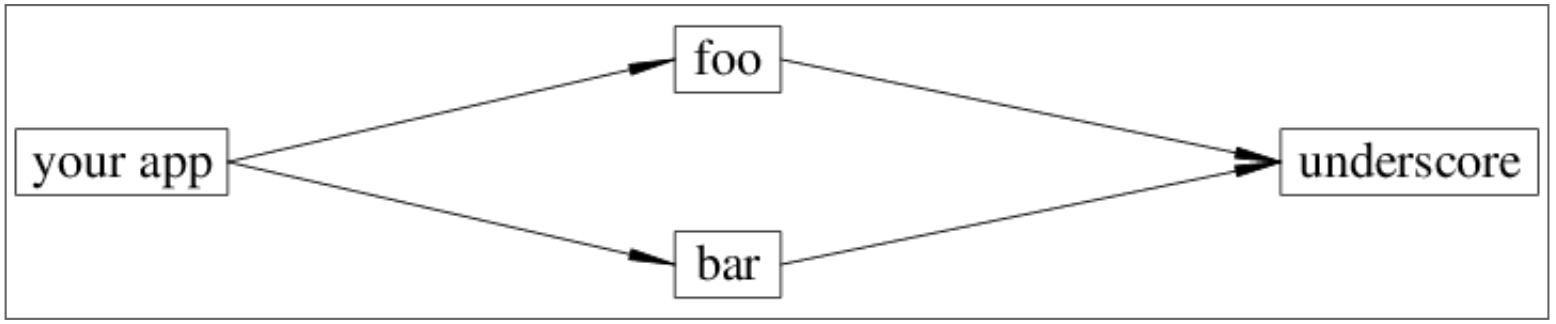
It will not install foo v2.0.0.

When foo v2.0.0 is released...

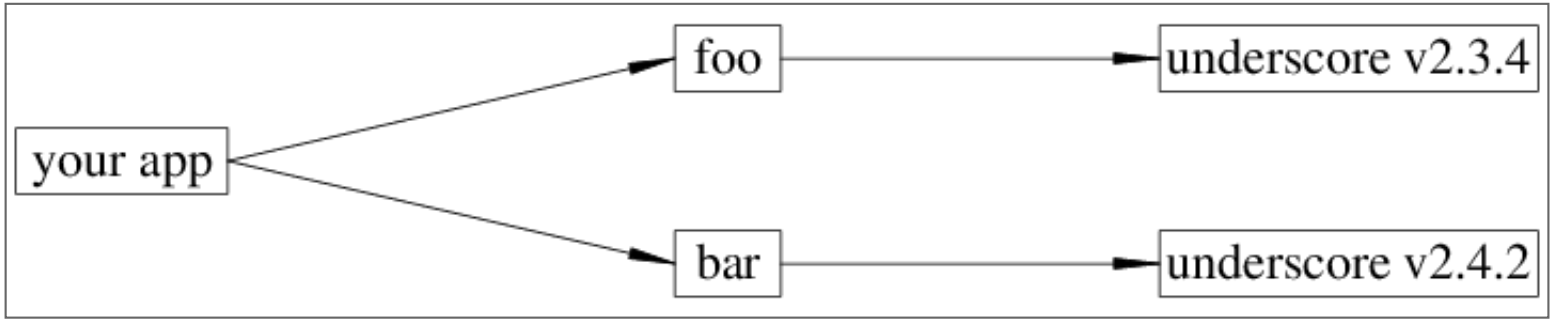
You test your app with it.

and make app v1.0.1 depend on foo v2.0.0.

Your app always uses the correct dependencies.

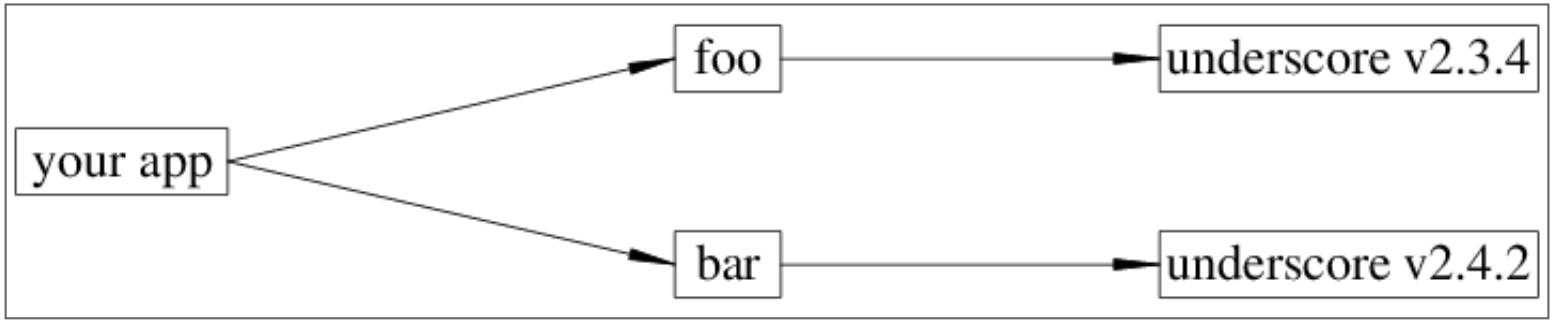


- foo v1.0.0 depends on underscore v2.3.4
- bar v1.0.0 depends on underscore v2.4.2



Package tree with two different underscore versions!

Disk space wasted?



Trade-off: This is the smaller problem.

When we have compatible dependencies,

then the space problem can be solved.

Otherwise, we prefer to trade space for reliability.

The future...

happens right now.

Languages compile to JavaScript

- babel (future JavaScript versions): ES6 and ES7
- syntax: CoffeeScript &co
- functional, static typing, security
- Java, Ruby, Python, Haskell, Erlang...
- emscripten, asm.js

Four Security mindsets

Passwords, firewalls, encryption, Alice, Bob and Charlie

- Security by disconnection --- but node.js connects!
- Individually protect sensitive data with separate containers and communication.

Buffer overflows, code injection and security patches

- Security aware community.
- JavaScript has no *"undefined behaviour/full compromize"*.
- JSON obsoletes data/code injection.
- Build specialized-bare bones servers (behind proxy).
- Reliable dependencies.

Social engineering

- Avoid confusion: follow established rules.
- Share code and concerns with the community.

Copy --- paste

- Run a server as root?!
- Parse JSON with *eval* or inject *data* into SQL?!
- Can exceptions crash and DOS your server?!

Avoid such failures with auditing, good examples:

Education and training.

Node.js training in Israel

- Dr. Yaakov Belch
- **node-js@yaakovnet.net**
- 050-8589070