Node.js Enterprise Middleware

Behrad Zari
behradz@gmail.com

June 2014

=Server-Side Javascript

Context

- Soft Real-Time Push-Based Applications
  - twitter, chat, sport bets, mobile, ...
- SPA
  - many req/sec with low response times, sharing things like validation code between the client and server, FAT client
- Cloud
  - SaaS, PaaS, DBaaS

Context (cont.)

- Queued Inputs
  - High concurrency, DB bottleneck, Brokers (MOM)
- JSON-based REST-full Interfaces
  - Service mediation & federation (SOA)
- Polyglot-Database Applications
  - NoSQL + Relational + Distributed Caches

in a rapid agile development fashion

Motivation: grow

Performance + Scalability

Rapid Change

Concurrency

Cloud

Mobile

IoT

Data Volume

Enterprise Service Farms

Integration

CGI (1993)

```
#!/usr/bin/perl

=head1 DESCRIPTION

printenv — a CGI program that just prints its environment

=cut
print "Content-type: text/plain\r\n\r\n";

for my $var ( sort keys %ENV ) {
 printf "%s = \"%s\"\r\n", $var, $ENV{$var};
}
```

a process/request

PHP (1994)

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP Test</title>
    </head>
    <body>
        <?php echo '<p>Hello World</p>'; ?>
    </body>
</html>
```

a thread/request

No Pooling

Java Servlet (1995)

```java
import java.io.IOException;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class ServletLifeCycleExample extends HttpServlet {

    private int count;

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        getServletContext().log("init() called");
        count = 0;
    }

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        getServletContext().log("service() called");
        count++;
        response.getWriter().write("Incrementing the count: count = " + count);
    }

    @Override
    public void destroy() {
        getServletContext().log("destroy() called");
    }

}
```

a thread/request

with pooling in container = pre-allocation

Problem 1?

- Context switching is not free
- Execution stacks take up memory

  A quick calculation: assuming that each thread potentially has an accompanying 2 MB of memory with it, running on a system with 8 GB of RAM puts us at a theoretical maximum of 4000 concurrent connections, plus the cost of context-switching between threads.

For massive concurrency,

cannot use an OS thread for each connection

Problem 2?

We always do I/O

```
result = db.query("select somthing from T");

// use result
```

which is blocking

I/O latency

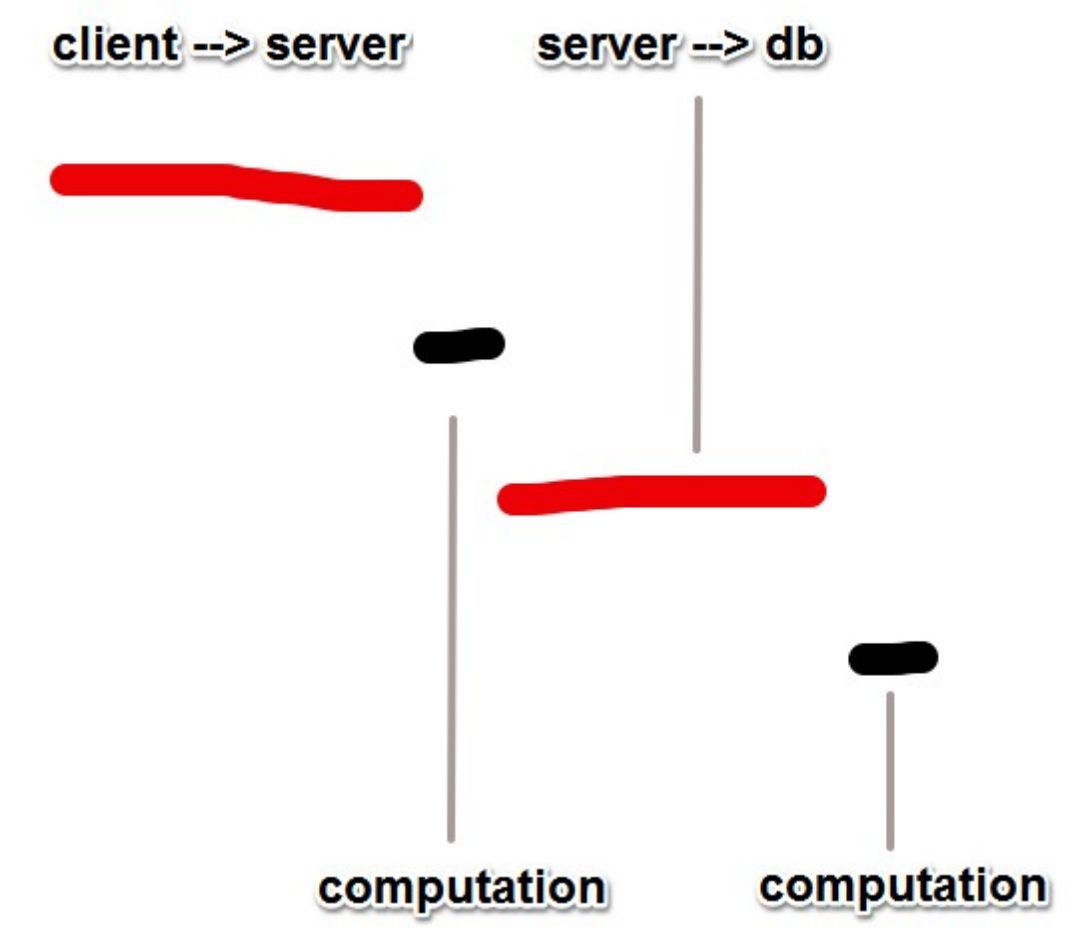
L1: 3 cycles

L2: 14 cycles

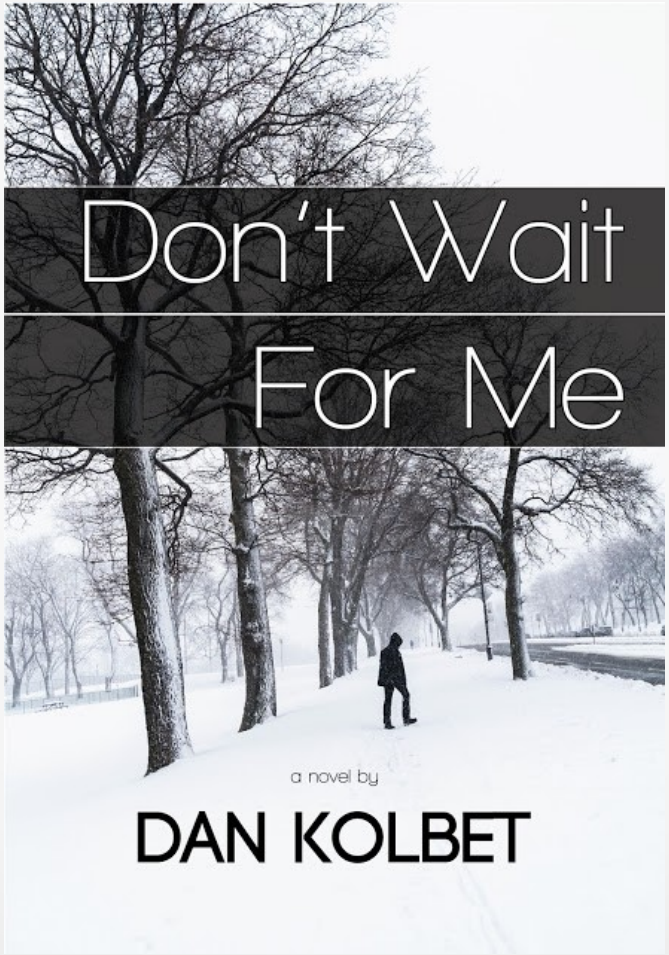
RAM: 250 cycles

DISK: 41,000,000 cycles

NETWORK: 240,000,000



Non-Blocking IO



Non-Blocking IO

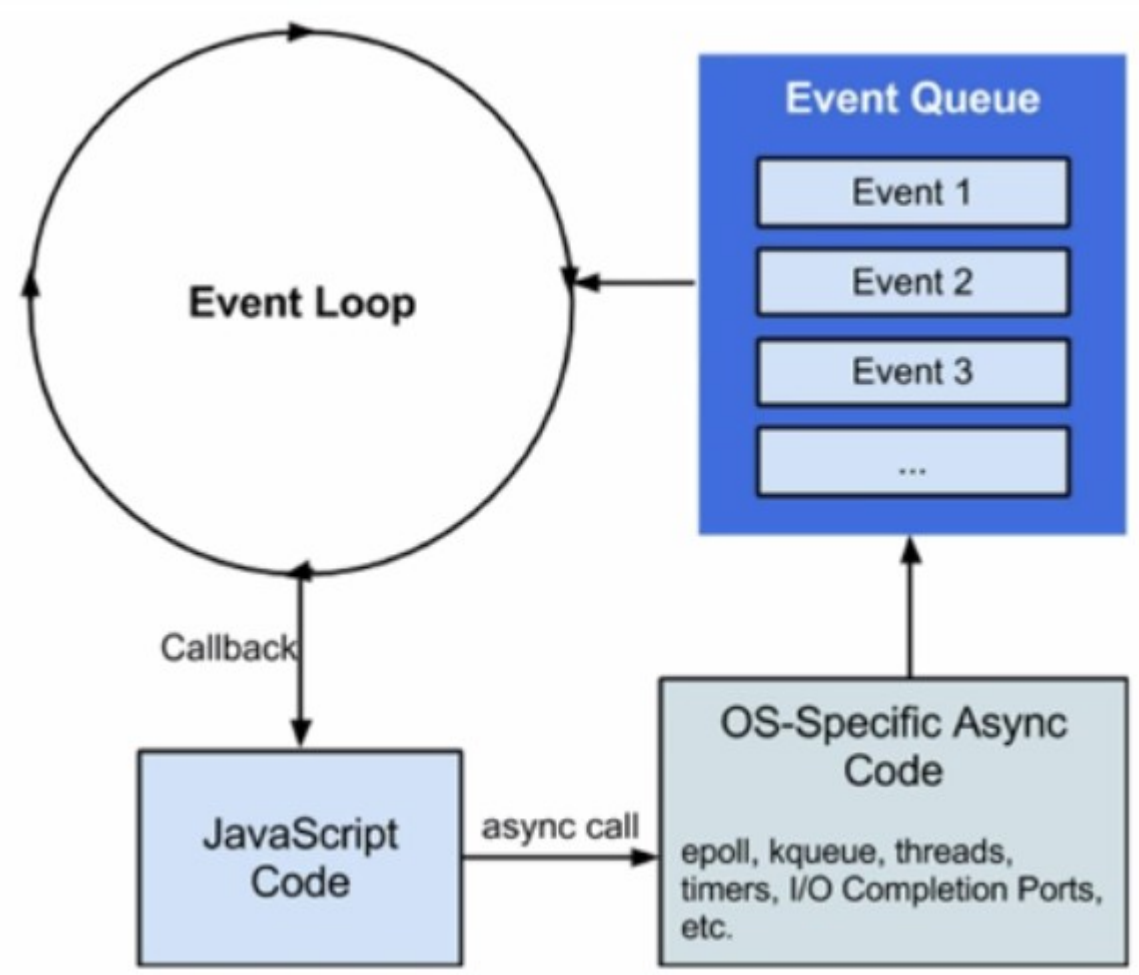
Callbacks + Polling

Epoll, POSIX AIO, ...

Event-Driven

```
While there are events to process
  e = get the next event
  perform the action requested by e in a thread
  if e’s thread is complete and e has an associated callback
    call the callback
```

Event Loop


Event Loop
Event Queue
Event 1
Event 2
Event 3
...
Callback
JavaScript Code
async call
OS-Specific Async Code
epoll, kqueue, threads, timers, I/O Completion Ports, etc.

Now…

```
var result = null;
db.query("select..", function (passedResult) {
  // use passed result…
  result = passedResult;
});
console.log( result ); // what’s result now?
```

Concurrency Models

Thread-based

- Locks and Shared State

vs

Event-driven Concurrency

- I/O parallelism without requiring CPU parallelism
- no synchronization is required
- maximize resource efficiency

Where were you!?

Why everyone isn’t using event loops, callbacks, non-blocking IO?

- Cultural
- Infrastructural

Cultural

This is How we're taught I/O:

```
puts("Enter your name: "
var name = gets();
puts("Name: " + name);
```

Cultural (2)

Rejected as too complicated

```
puts("Enter your name: ");
gets(function (name) {
   puts("Name: " + name);
});
```

Missing Infrastructure

Single threaded event loops require I/O to be non-blocking

Most libraries are not.

Too Much Infrastructure

EventMachine, Twisted, AnyEvent, Apache Mina, Java nio, ...

users are confused how to combine with other available libraries

Why Javascript?

Javascript designed specifically to be used with an event loop purely:

- Anonymous functions, closures
- Only one callback at a time
- I/O through DOM event callbacks
- culture of Javascript is already geared towards evented programming

What is Node.js?

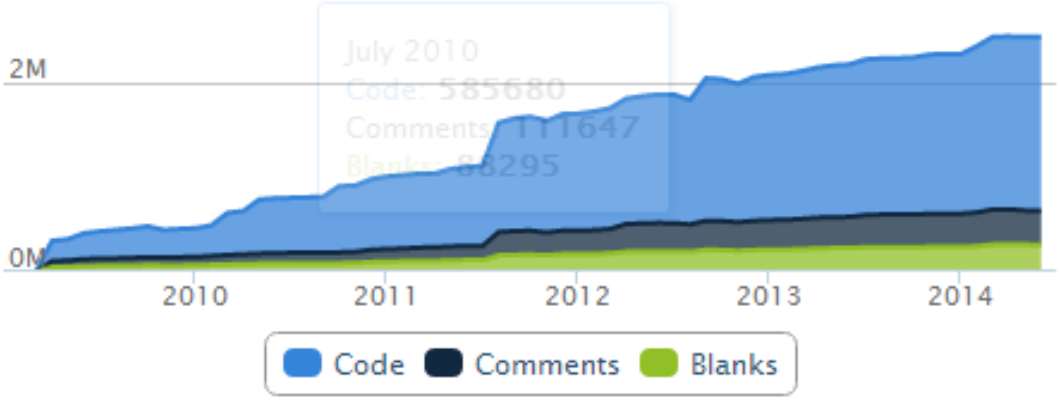
Node.js emerged from a project at cloud service provider Joyent in 2009

Node.js Project on www.ohloh.net

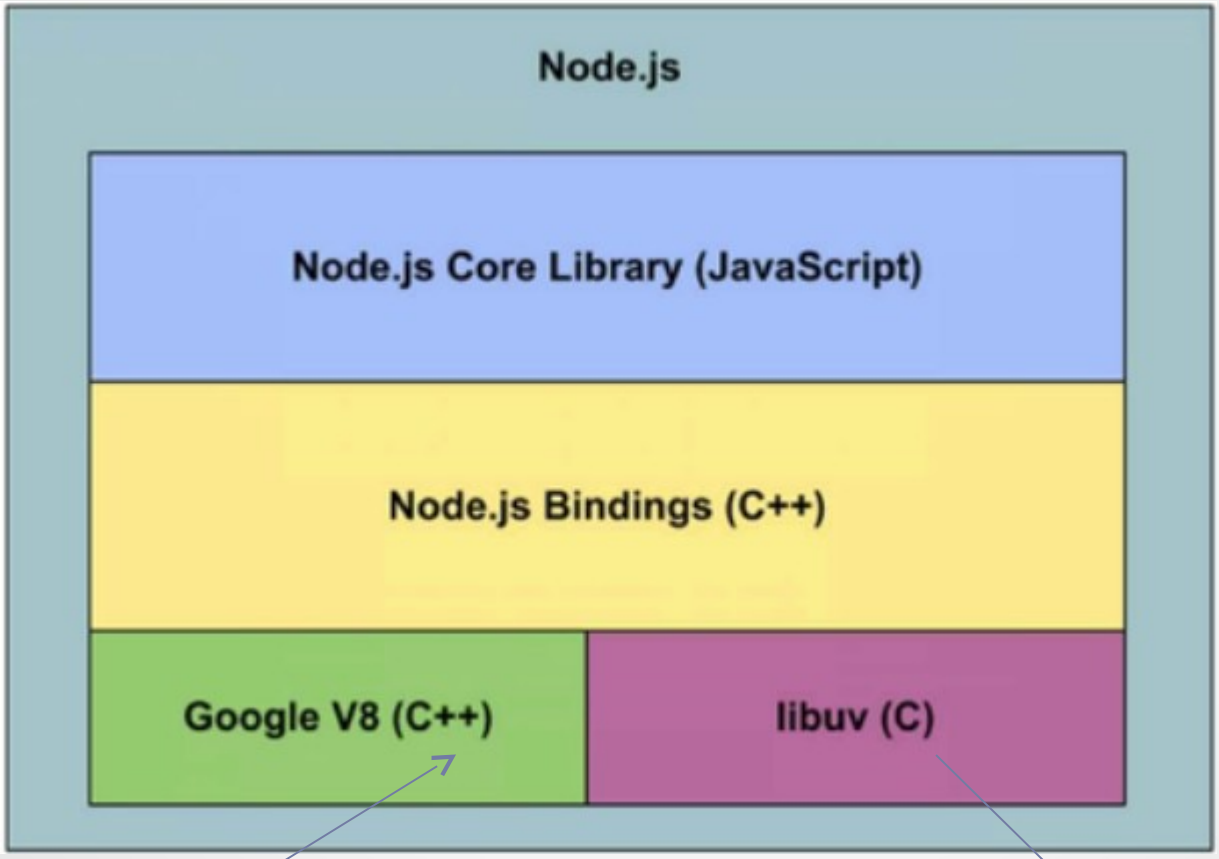
Languages

C++	39%	C	25%
JavaScript	18%	21 Other	18%

Lines of Code



Node.js Architecture



Open source JavaScript engine, Optimizer, JIT, inline caching, GC

Cross-platform asychronous I/O

Node.js Server (2009)

```
1  var http = require('http');
2  var count = 0;
3  http.createServer(function (req, res) {
4      res.writeHead(200, {'Content-Type': 'text/plain'});
5      res.end((++count)+'\n');
6  }).listen(1337);
7  console.log("Server running at http://127.0.0.1:1337");
```

```
% node example.js
Server running at http://127.0.0.1:1337/
```

Node.js Ecosystem

Modules (CommonJS Securable Module)

- Think Modular, not Monolithic
- Fine grained pieces of logic

innovation through modularity

npm

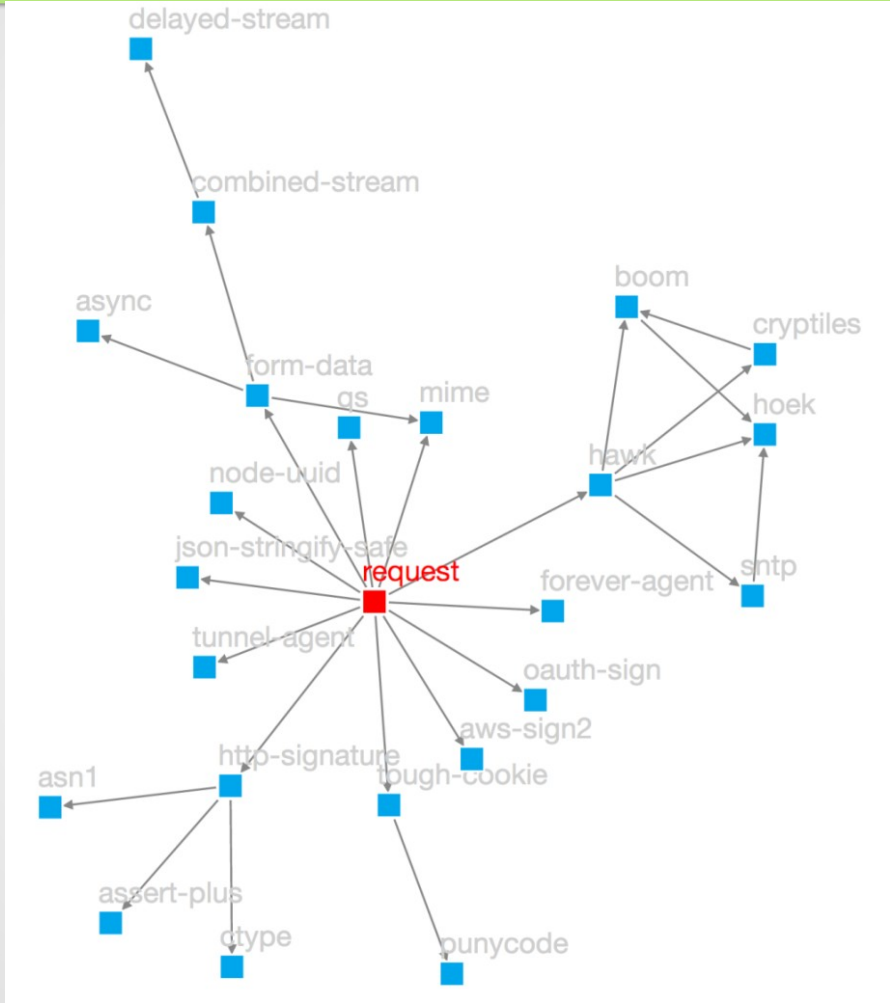
module (package) manager, taking care of recursive dependencies and resolving collisions, ...

Publishing a node module fulfills satisfaction for developers much greater than contributing to a larger code base

Module’s package.json

```
{
    "name": "my-program",
    "version": "1.0.0",
    "description": "program description",
    "homepage": "http://programs-website.com",
    "keywords": [
        "one",
        "two",
        "three"
    ],
    "author": "Bevry Pty Ltd <us@bevry.me> (http://bevry.me)",
    "maintainers": [
        "Benjamin Lupton <b@lupton.cc> (http://balupton.com)"
    ],
    "contributors": [
        "Benjamin Lupton <b@lupton.cc> (http://balupton.com)"
    ],
    "bugs": {
        "url": "http://programs-issue-tracker.com"
    },
    "repository" : {
        "type": "git",
        "url": "http://programs-repository.git"
    },
    "engines" : {
        "node": ">=0.6.0",
        "npm": ">=1.1.0"
    },
    "dependencies": {
        "connect": "2.6.x",
        "express": "3.0.x",
        "socket.io": "0.9.x"
    },
    "devDependencies": {
        "coffee-script": "1.4.x"
    },
    "directories": {
        "lib": "./lib"
    },
    "bin": {
        "program": "./bin/program"
    },
    "scripts": {
        "test": "node ./test/everything.test.js"
    },
    "main": "./main.js"
}
```

Dependencies *are* your friends



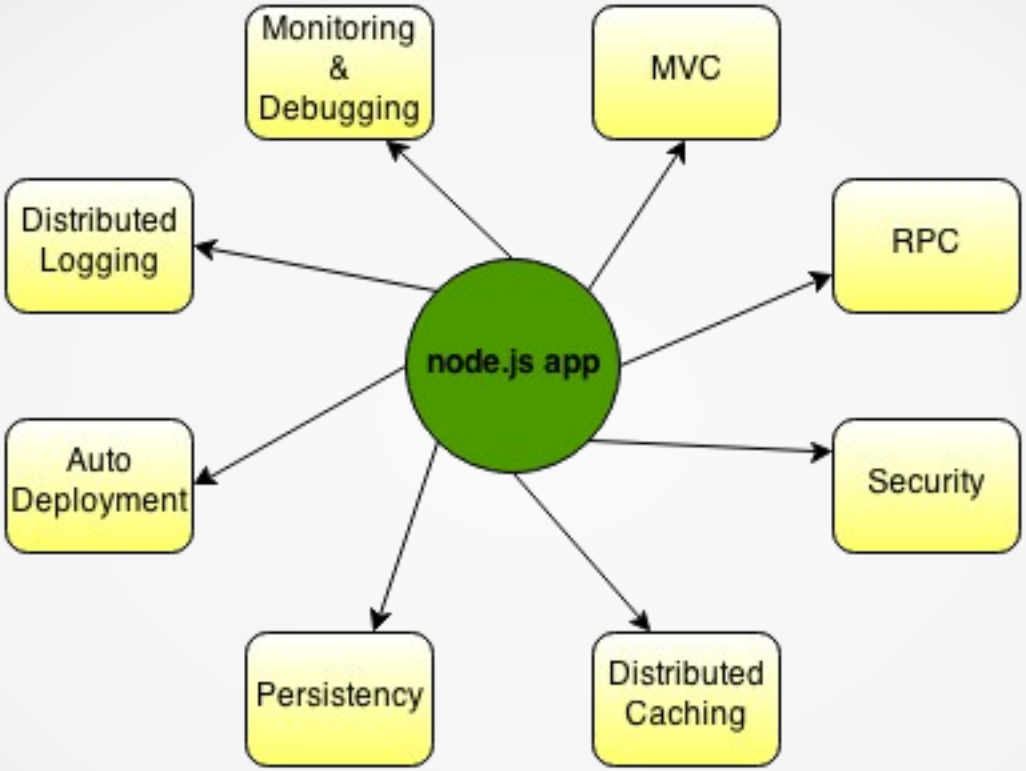
lets you avoid dependency hell by modifying the require.paths at runtime

Do not force everyone to agree on the same version of a module

Node.js Ecosystem

A single Node.js application normally **depends on more than a dozen npm modules.**

Node.js Ecosystem



event loop and event handlers yields
an IoC:
why we needed IoC?
how is real world?

Node.js TCP Server Socket

```
var net = require('net');

var HOST = '127.0.0.1';
var PORT = 6969;

// Create a server instance, and chain the listen function to it
// The function passed to net.createServer() becomes the event handler for the 'connection' event
// The sock object the callback function receives UNIQUE for each connection
net.createServer(function(sock) {

    // We have a connection - a socket object is assigned to the connection automatically
    console.log('CONNECTED: ' + sock.remoteAddress +':'+ sock.remotePort);

    // Add a 'data' event handler to this instance of socket
    sock.on('data', function(data) {

        console.log('DATA ' + sock.remoteAddress + ': ' + data);
        // Write the data back to the socket, the client will receive it as data from the server
        sock.write('You said "' + data + '"');

    });

    // Add a 'close' event handler to this instance of socket
    sock.on('close', function(data) {
        console.log('CLOSED: ' + sock.remoteAddress +' '+ sock.remotePort);
    });

}).listen(PORT, HOST);

console.log('Server listening on ' + HOST +':'+ PORT);
```

RPC (RMI) in node.js

Dnode Server

```
// server:
var DNode = require('dnode');

var server = DNode({
    timesTen : function (n,f) { f(n * 10) },
}).listen(6060);
```

Dnode Client

```
// client:
var DNode = require('dnode');
var sys = require('sys');

DNode.connect(6060, function (remote) {
    remote.timesTen(5, function (result) {
        sys.puts(result); // 5 * 10 == 50
    });
});
```

How to scale our single-threaded high performance node.js service?

with node.js built-in Cluster module

Cluster Module

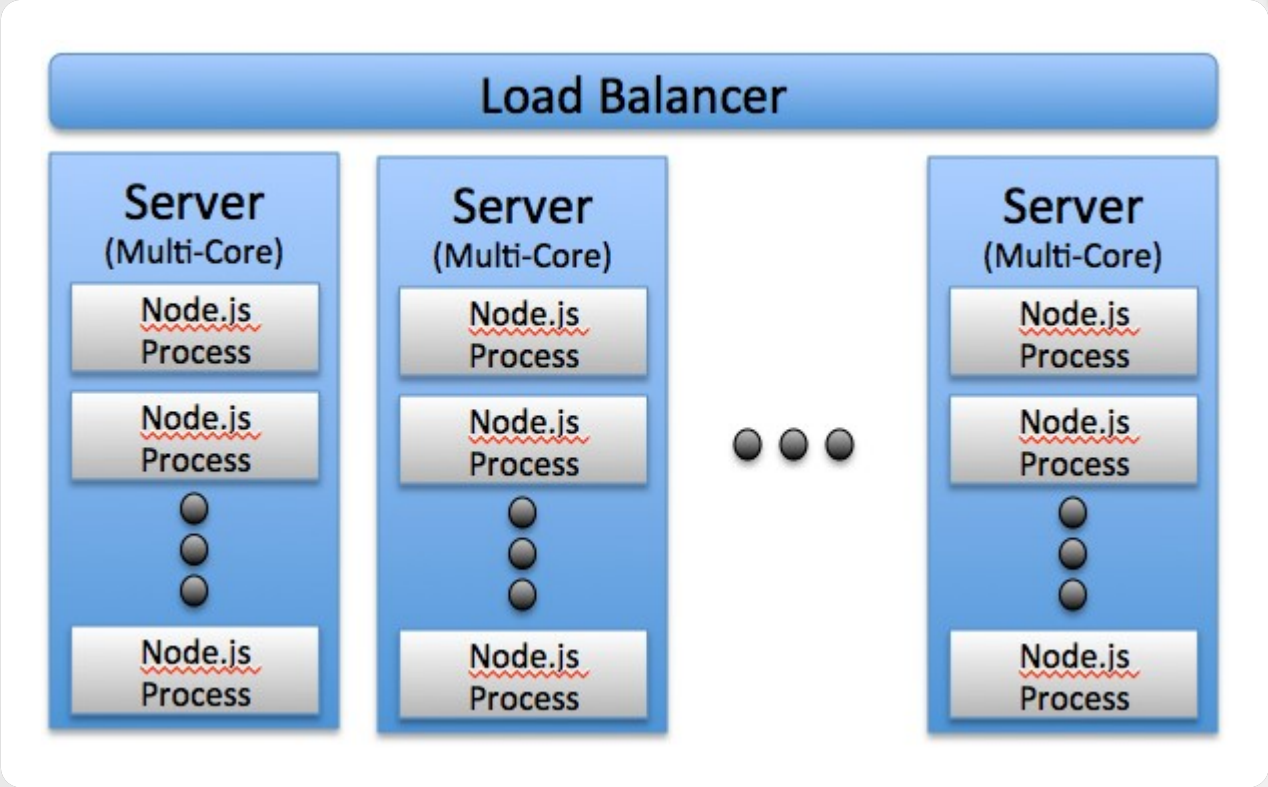
```
var cluster = require('cluster');
var http = require('http');
var numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  // Fork workers.
  for (var i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', function(worker, code, signal) {
    console.log('worker ' + worker.process.pid + ' died');
  });
} else {
  // Workers can share any TCP connection
  // In this case its a HTTP server
  http.createServer(function(req, res) {
    res.writeHead(200);
    res.end("hello world\n");
  }).listen(8000);
}
```

N-Copy,
Shared Socket,
Round-Robin

Node.js achieves scalability levels of over 1Million concurrent connections

Node.js Scalability Model

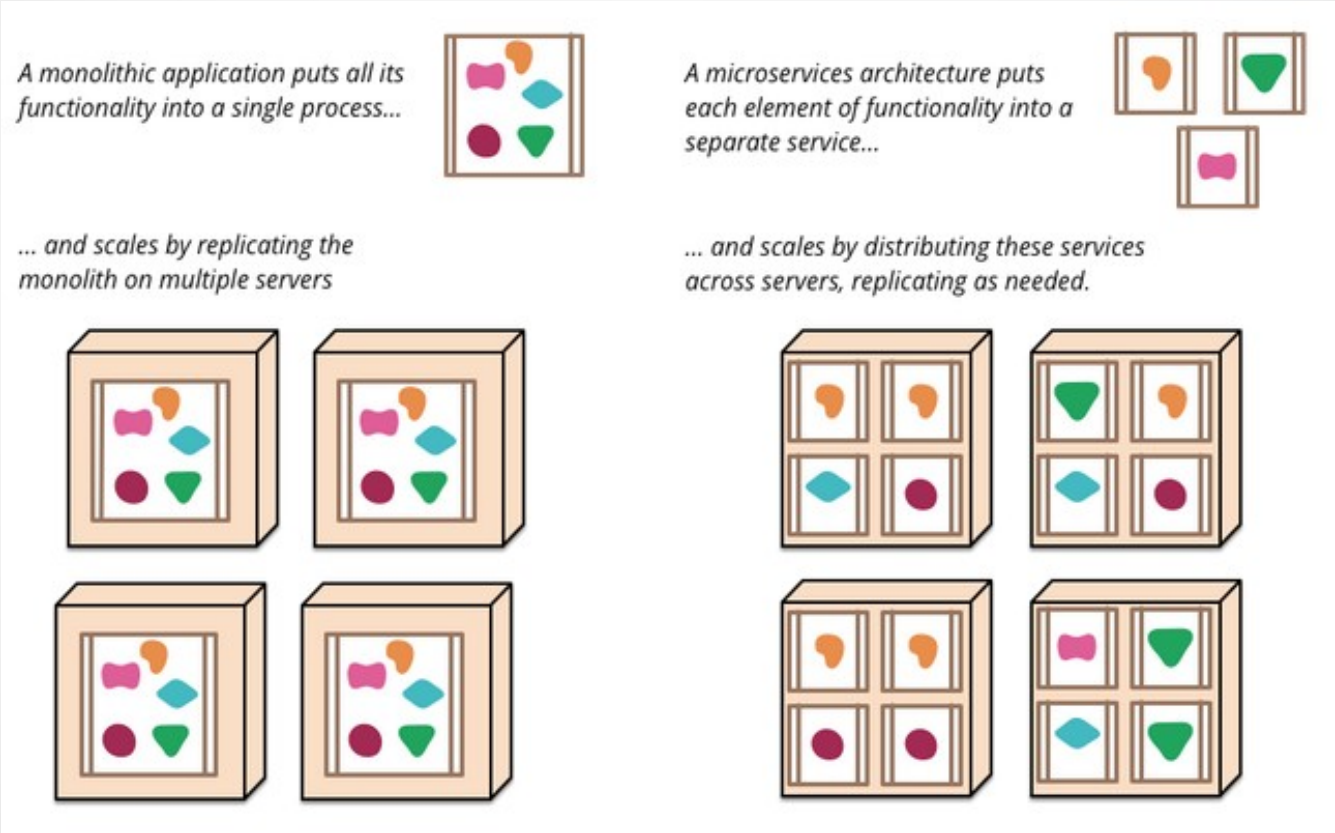
Horizontal, share-nothing



independently scale the subsystems

a JavaEE 3-tier app is actually not written with Horizontal clustering in mind

Microservice Architecture



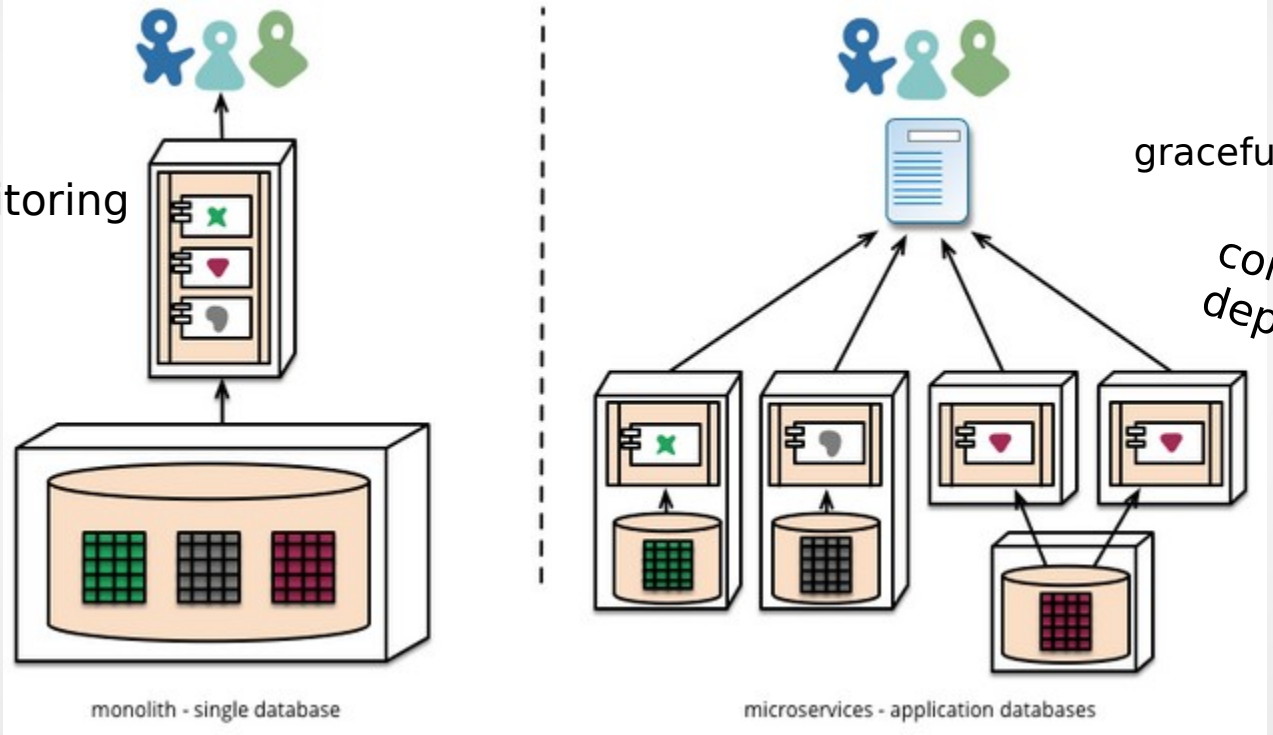
http://martinfowler.com/articles/microservices.html

Decentralized Responsibility & Data

Independently deployable services

Easy monitoring

SPoF



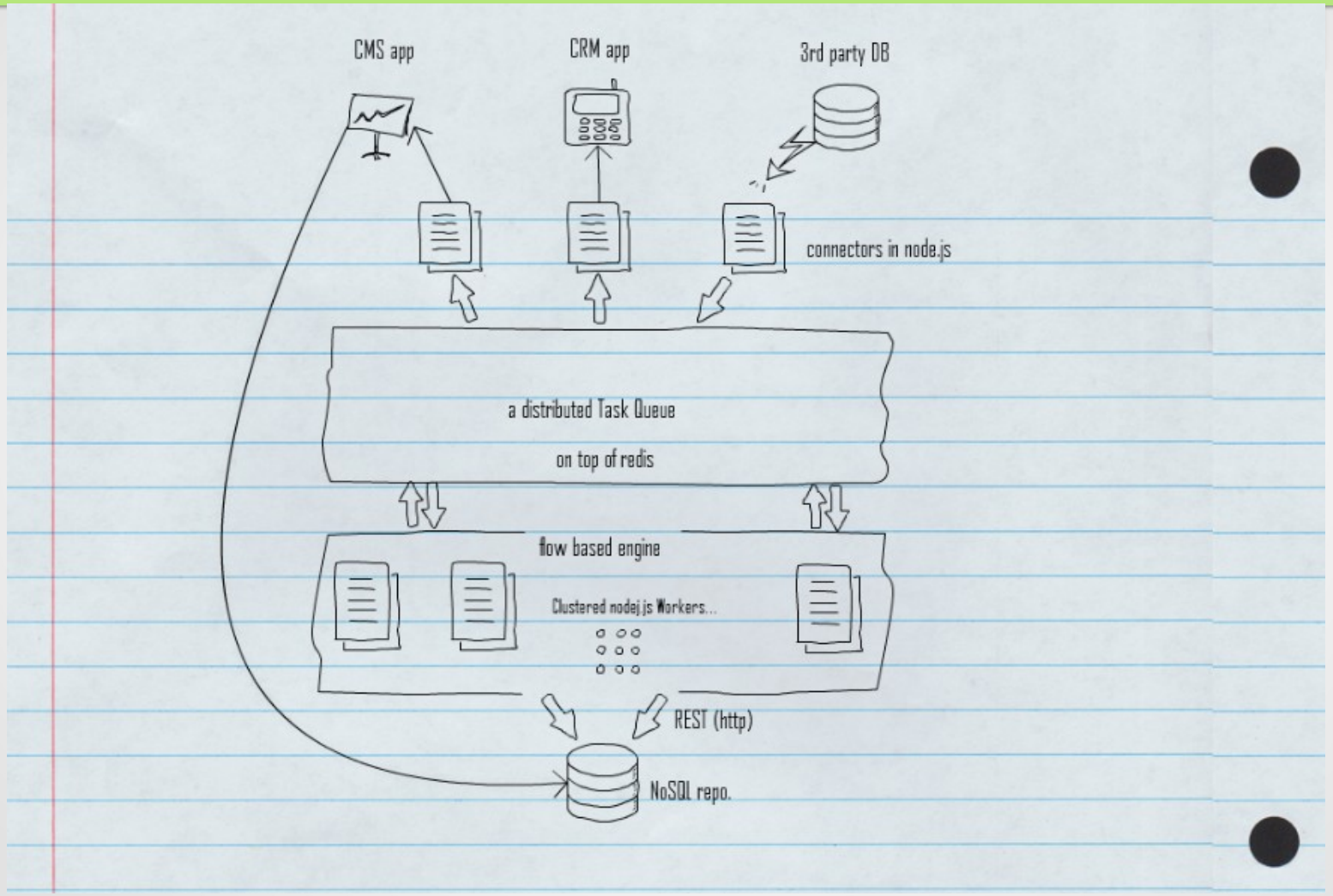
gracefully fail

continuous deployment

Polyglot Persistence

http://martinfowler.com/articles/microservices.html

Node.js Enterprise Middleware



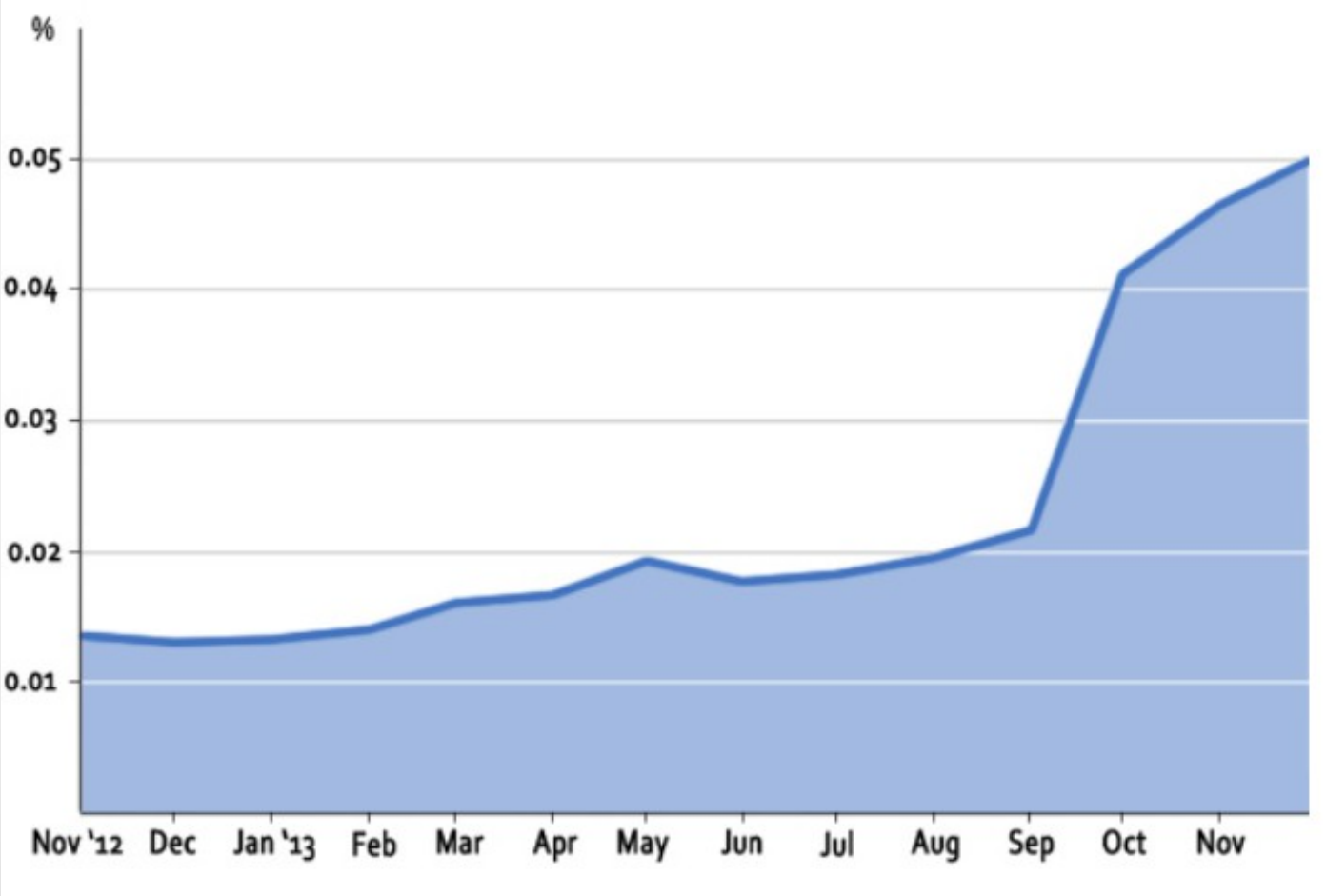
Node.js Middleware Benefits

- Performance & Scalability
  - single threaded event loop + async IO (built-in)
  - easy high performance middleware stack (as a primary goal)
  - hard to write slow programs
- Agile and lightweight development
  - Javascript
  - modularity
  - TDD, BDD, automated tests(mocha,...)
  - avoid build step (live coding)
- Auto Restart and Continuous Deployment
  - process monitoring (forever, pm2,...)
  - fail fast: just fail & let some other process do error recovery

Node.js Business Benefits

- Motivation
  - high risk
  - fast prototyping
  - continuous delivery
- Productivity
  - an enterprise scale web server in 5 lines
  - >80K modules on npm (growing ecosystem)
  - Comet, Pervasive Computing, REST, MOM, Integration?
- Developer Joy
  - more happy, works better, more developers join
- Cost Savings
  - fewer developers
  - smaller iterations + more velocity => half-time
  - Less hardware

Market Share

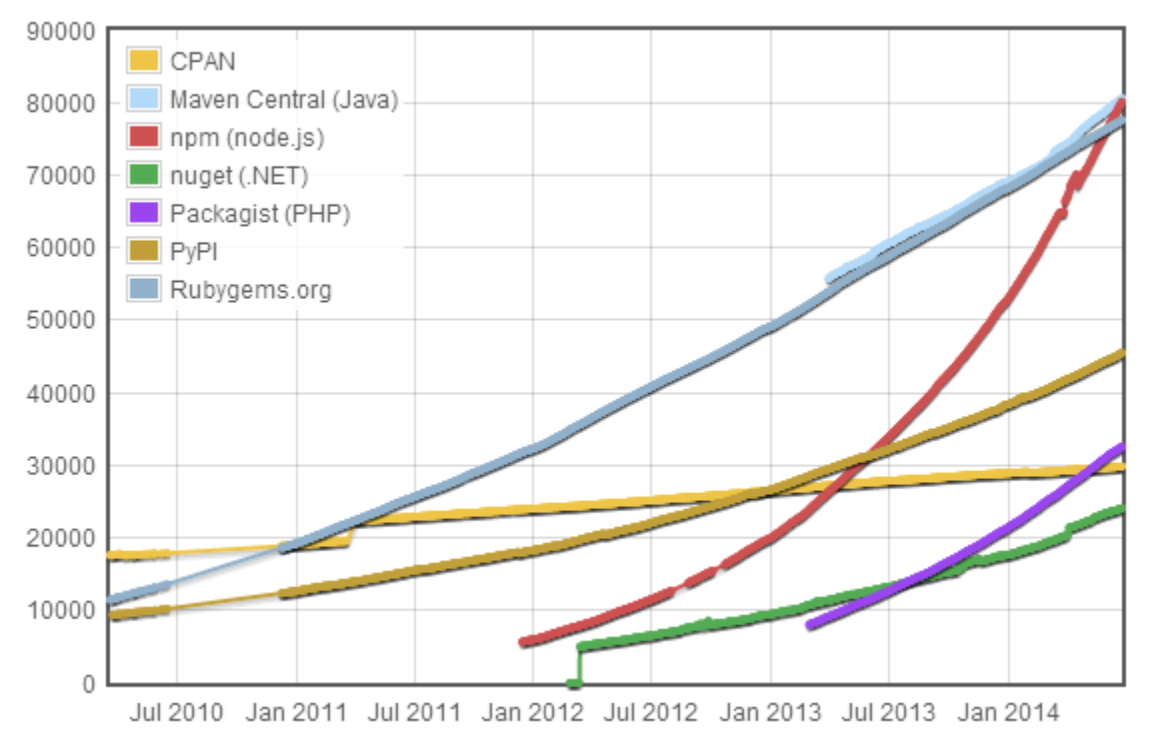


whalepath.com report

Success Stories

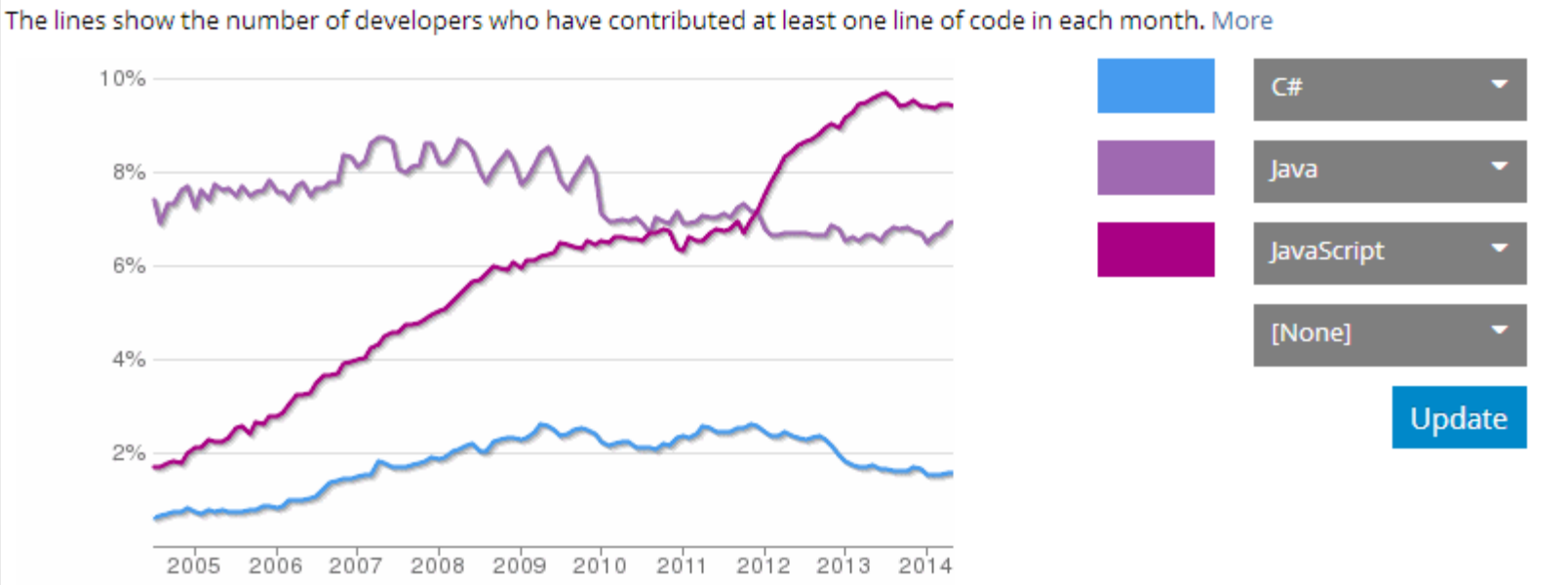
Name	Usage
Paypal	$3.5 billion IN 2011, In Java Spring: unfriendly environment for front-end engineers With Java, when deployed to a staging server was a few thousandths of a second wait in Node.js, Bill Scott , 33-50% fewer lines of code to get a job done PayPal has redone 22 of its customer applications: PayPal Funding Source, PayPal Account Details, PayPal LogOn, and Wallet
eBay	released ql.io, an event-driven aggregation tool for API consumption
LinkedIn	massive performance gains with Node
Yahoo!	Mojito, 200 developers, 500 private, 800 extra modules, 2,000,000 req/min
Walmart	mobile development, no server down time!
Oracle	Nashorn: a JVM-based JavaScript engine

Repository Module Count



www.modulecounts.com

Monthly Contributors



Node.js Project on Ohloh (2014)

node.js (NodeJs)

Compare

Analyzed 1 day ago

This is the NodeJs project: Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

1.84M	lines of code
151	current contributors
7 days	since last commit
220	users on Ohloh

Very High Activity

0 Reviews

Mostly written in C++

Licenses: Apache-2.0

evented | node | c++ | cplusplus | v8 | server | framework | javascript | asynchronous | nodejs

8000 lines of C/C++, 2000 lines of Javascript, 14 contributors @ 2009

.end()

```
behrad.thankYou( function( err, question, next ) {
   question.tryAnswering( function(err, ack) {
      if( err && !ack ) {
         return question.explainMore( next );
      }
      next();
  });
});
```

References

- It was a huge list, please surf the web as me, or contact me :)

Let me know if you have...

- High Throughput & Scalable backend
- NoSQL, Data Engineering, BI
- Enterprise Integration

problems.

Supplementary Content...

- Js anti-pattern 1: Callback hell

JS Callback hell

```
doAsync1(function () {
  doAsync2(function () {
    doAsync3(function () {
      doAsync4(function () {
    })
  })
})
```

```
var fs = require('fs')
var path = require('path')

module.exports = function (dir, cb) {
  fs.readdir(dir, function (er, files) { // [1]
    if (er) return cb(er)
    var counter = files.length
    var errored = false
    var stats = []

    files.forEach(function (file, index) {
      fs.stat(path.join(dir,file), function (er, stat) { // [2]
        if (errored) return
        if (er) {
          errored = true
          return cb(er)
        }
        stats[index] = stat // [3]

        if (--counter == 0) { // [4]
          var largest = stats
            .filter(function (stat) { return stat.isFile() }) // [5]
            .reduce(function (prev, next) { // [6]
              if (prev.size > next.size) return prev
              return next
            })
          cb(null, files[stats.indexOf(largest)]) // [7]
        }
      })
    })
  })
}
```

Solution 1: async module

```
var fs = require('fs')
var async = require('async')
var path = require('path')

module.exports = function (dir, cb) {
  async.waterfall([ // [1]
    function (next) {
      fs.readdir(dir, next)
    },
    function (files, next) {
      var paths =
       files.map(function (file) { return path.join(dir,file) })
      async.map(paths, fs.stat, function (er, stats) { // [2]
        next(er, files, stats)
      })
    },
    function (files, stats, next) {
      var largest = stats
        .filter(function (stat) { return stat.isFile() })
        .reduce(function (prev, next) {
        if (prev.size > next.size) return prev
          return next
        })
        next(null, files[stats.indexOf(largest)])
    }
  ], cb) // [3]
}
```

Solution 2: promises

```
var fs = require('fs')
var path = require('path')
var Q = require('q')
var fs_readdir = Q.denodeify(fs.readdir) // [1]
var fs_stat = Q.denodeify(fs.stat)

module.exports = function (dir) {
  return fs_readdir(dir)
    .then(function (files) {
      var promises = files.map(function (file) {
        return fs_stat(path.join(dir,file))
      })
      return Q.all(promises).then(function (stats) { // [2]
        return [files, stats] // [3]
      })
    })
    .then(function (data) { // [4]
      var files = data[0]
      var stats = data[1]
      var largest = stats
        .filter(function (stat) { return stat.isFile() })
        .reduce(function (prev, next) {
        if (prev.size > next.size) return prev
          return next
        })
      return files[stats.indexOf(largest)]
    })
}
```

Solution 3: generators

```
var co = require('co')
var thunkify = require('thunkify')
var fs = require('fs')
var path = require('path')
var readdir = thunkify(fs.readdir) <strong>[1]</strong>
var stat = thunkify(fs.stat)

module.exports = co(function* (dir) { // [2]
  var files = yield readdir(dir) // [3]
  var stats = yield files.map(function (file) { // [4]
    return stat(path.join(dir,file))
  })
  var largest = stats
    .filter(function (stat) { return stat.isFile() })
    .reduce(function (prev, next) {
      if (prev.size > next.size) return prev
      return next
    })
  return files[stats.indexOf(largest)] // [5]
})
```