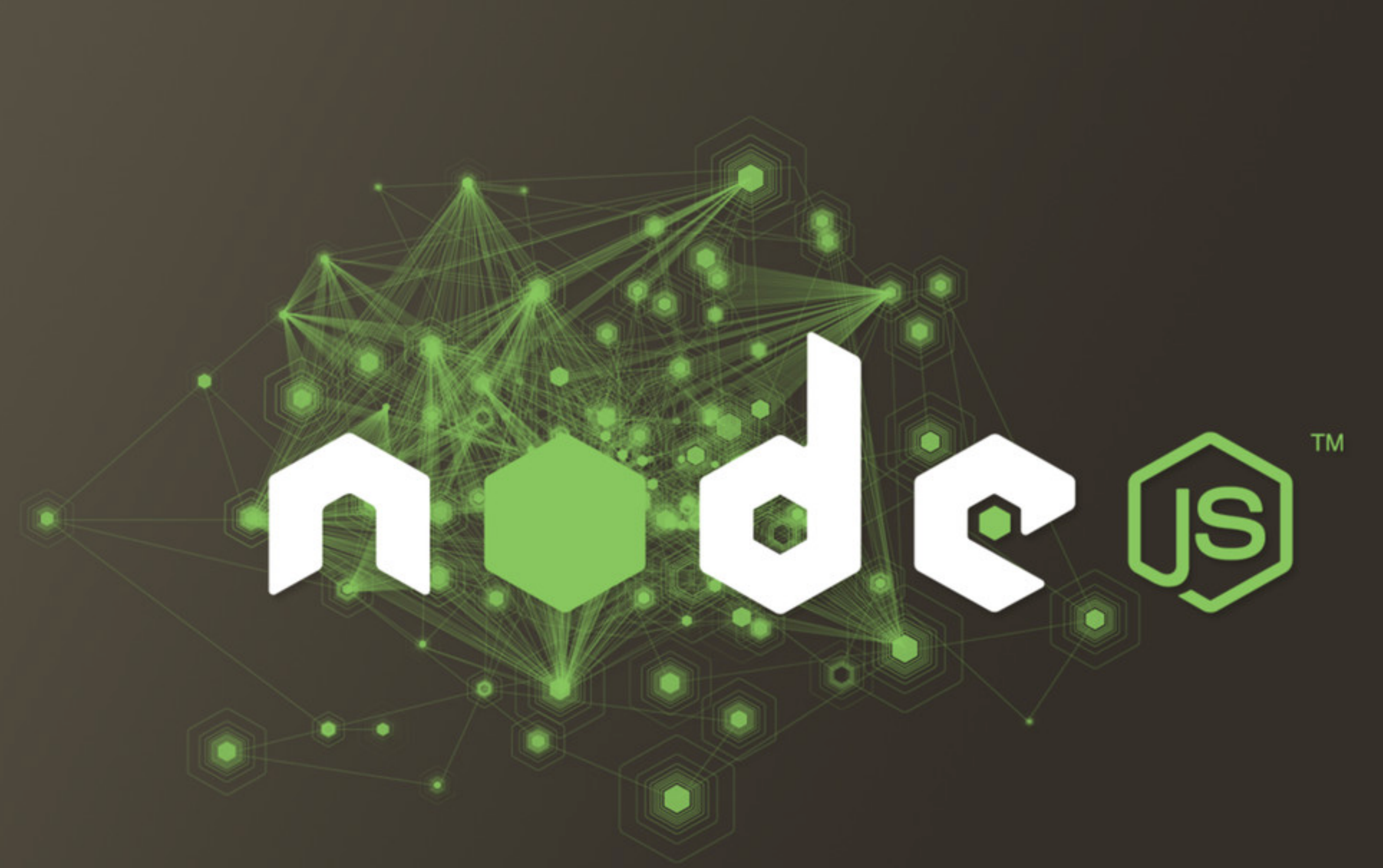

node
JS
TM

A thorough introduction to:

Node.js

“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.”

–NODEJS.ORG

V8 IS GOOGLE'S OPEN-SOURCE, HIGH-PERFORMANCE JAVASCRIPT ENGINE.

V8 IS WRITTEN IN C++ AND IS USED IN GOOGLE CHROME, THE OPEN SOURCE BROWSER FROM GOOGLE.

V8 IMPLEMENTS ECMASCRIPT AS SPECIFIED IN ECMA-262, 5TH EDITION, AND RUNS ON WINDOWS (XP OR NEWER), MAC OS X (10.5 OR NEWER), AND LINUX SYSTEMS THAT USE IA-32, X64, OR ARM PROCESSORS.

V8 CAN RUN STANDALONE, OR CAN BE EMBEDDED INTO ANY C++ APPLICATION.

Interest in Node.js over time

Topics

Subscribe

node.js
Search term

ruby on rails
Search term

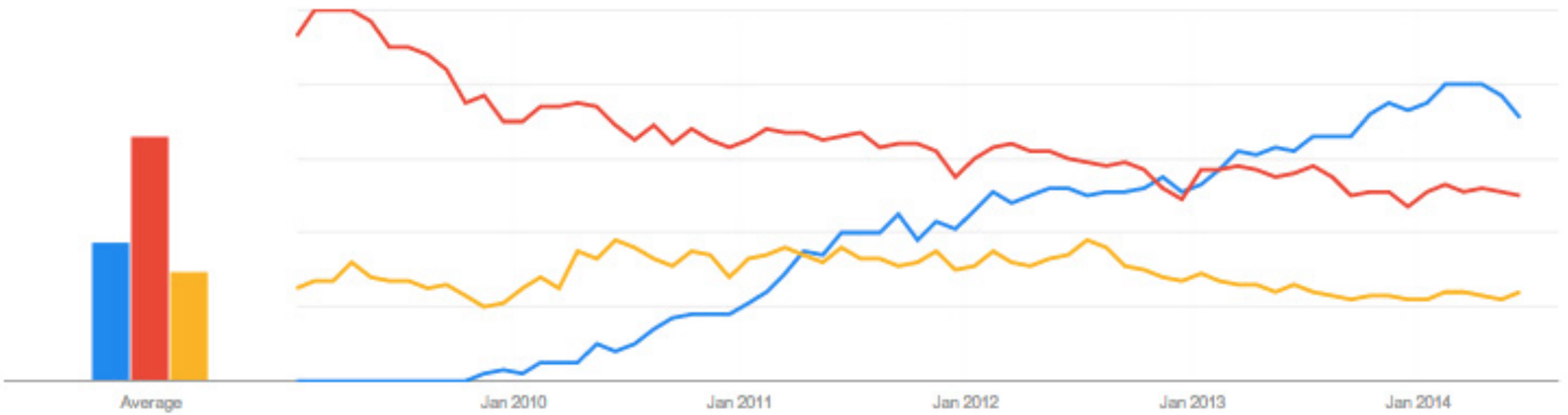
objective-c
Search term

+Add term

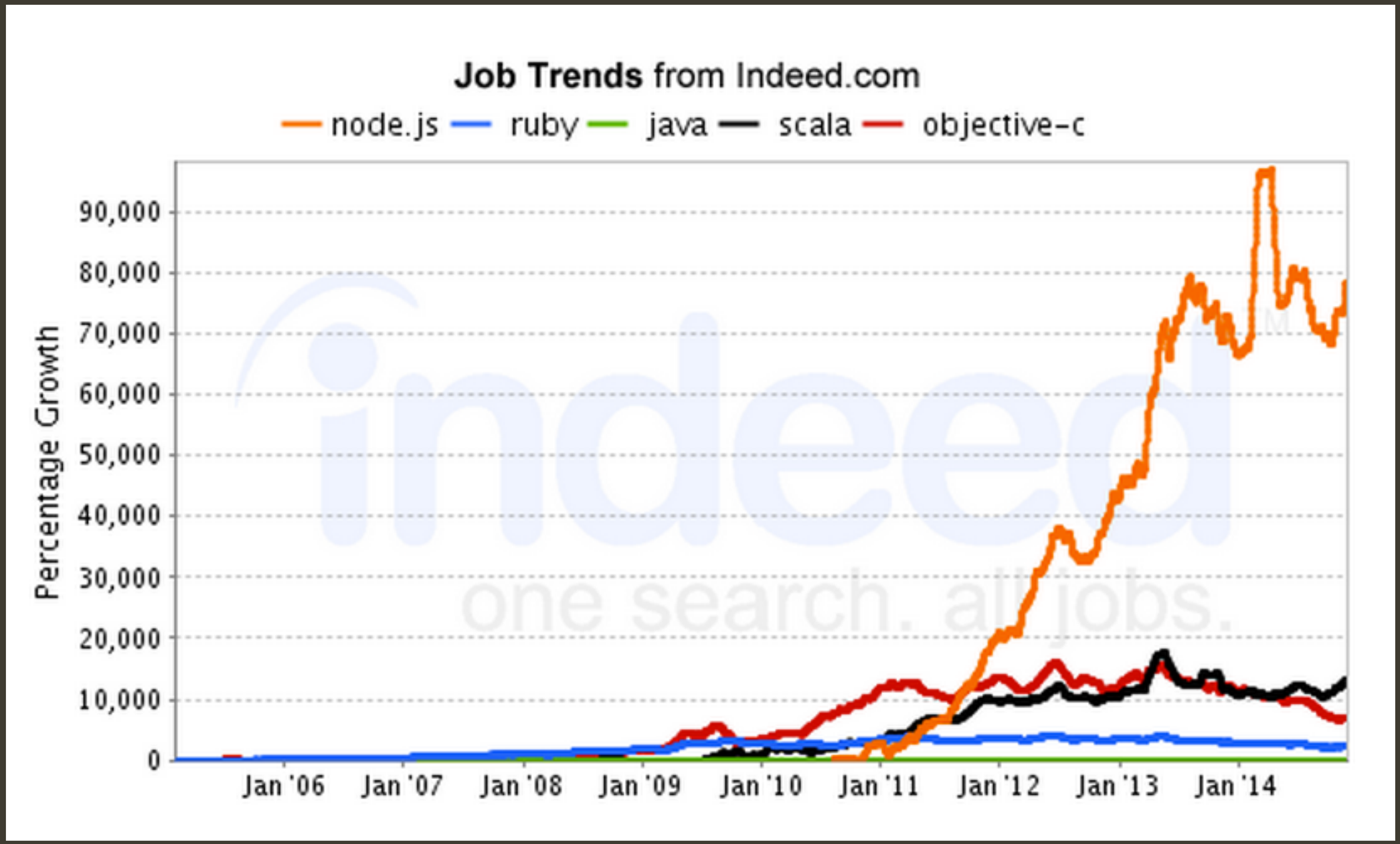
Interest over time

News headlines

Forecast



Also great for getting a job



Node.js in the Wild

Companies that use node.js right now

LinkedIn

UBER

Storify

non-blocking, evented I/O makes node

event-driven

non-blocking, evented I/O makes node

asynchronous

non-blocking, evented I/O makes node

fast

UNDERSTANDING I/O

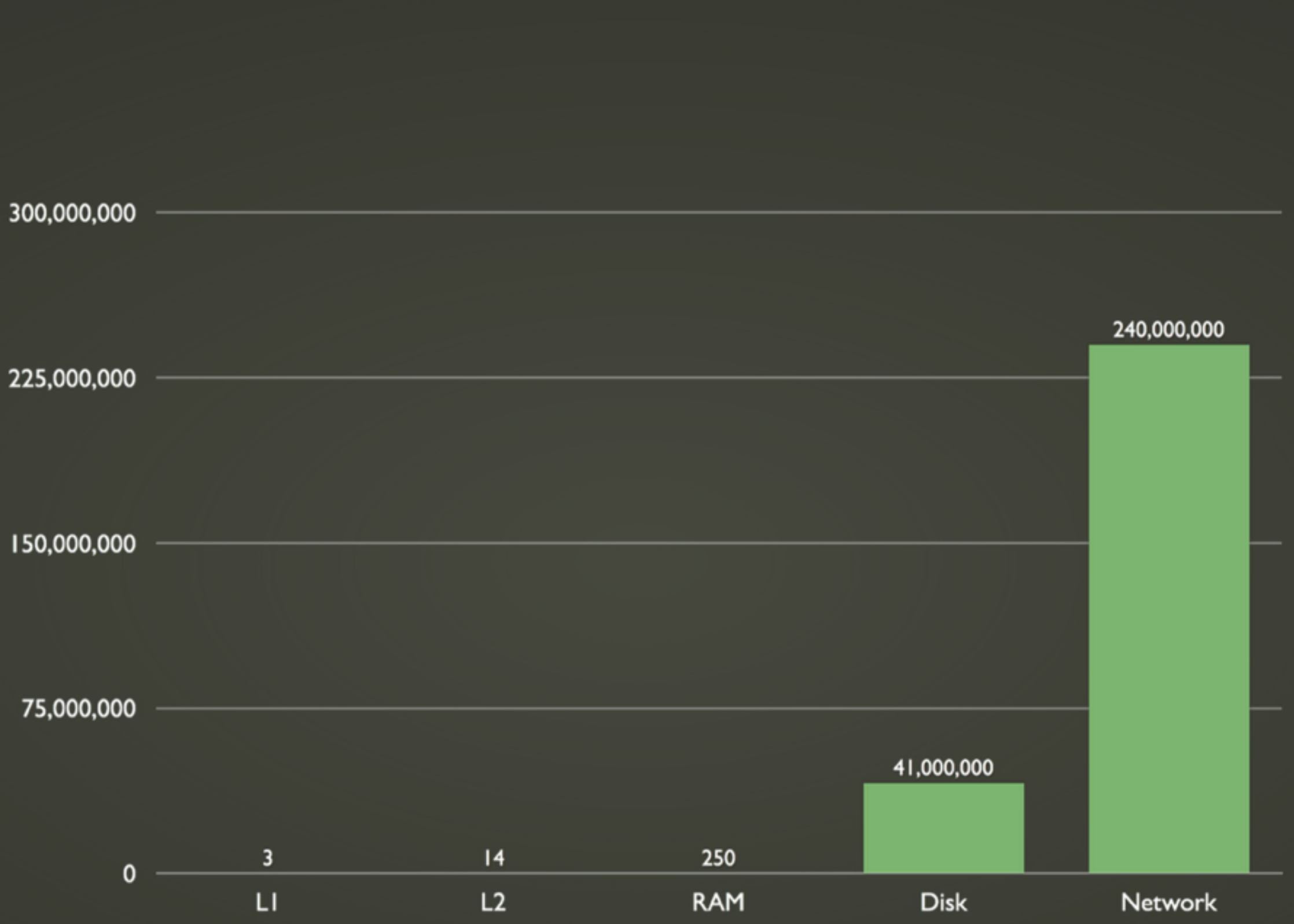
Memory Hierarchy

	1 ns	1x
	10 ns	10x
	100 ns	100x
	100 ms	100,000,000x
	10 s	1e+10x

I/O Latency

Source: Ryan Dahl's 2008.11.08 node.js presentation

L1	3 cycles
L2	14 cycles
RAM	250 cycles
Disk	41,000,000
Network	240,000,000

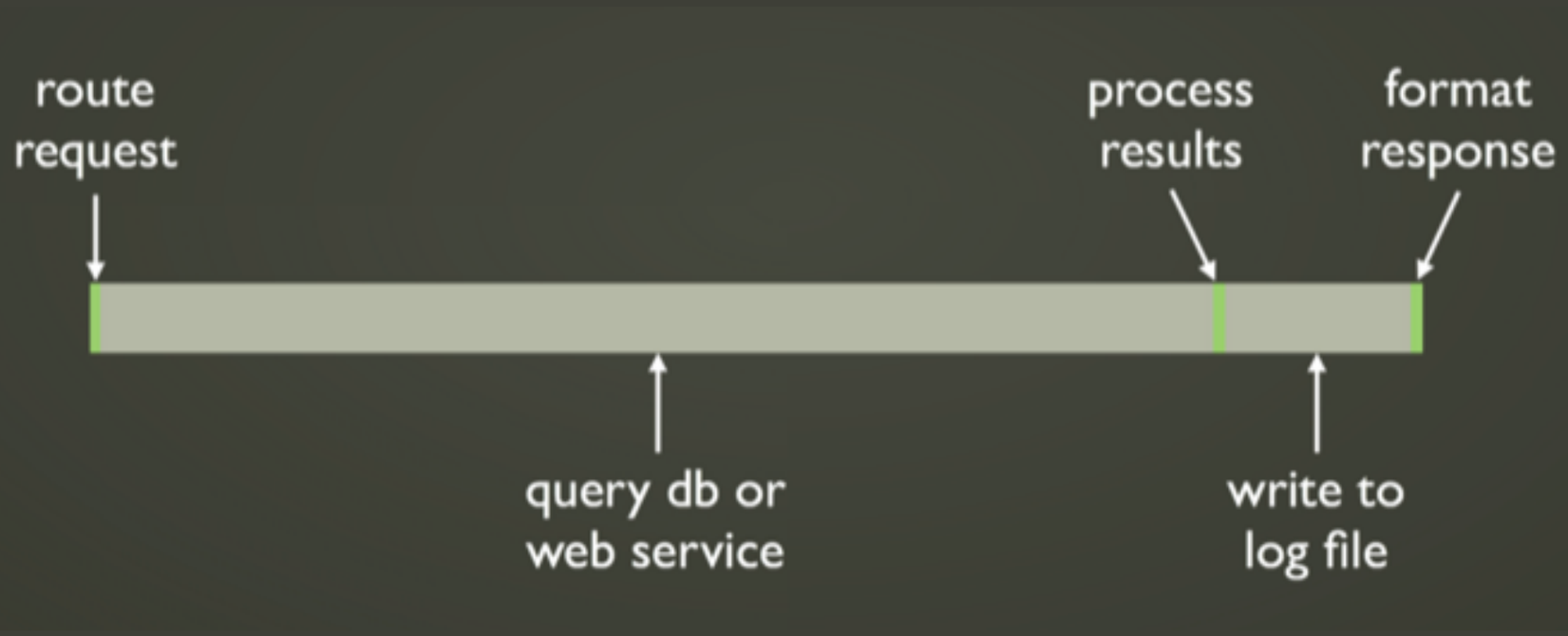

300,000,000
225,000,000
150,000,000
75,000,000
0
3
14
250
41,000,000
240,000,000
L1
L2
RAM
Disk
Network

I/O Latency

	L2	RAM	Disk	Network
L1	5	83	13,666,666	80,000,000
L2		18	2,928,571	17,142,857
RAM			164,000	960,000
Disk				6

SERVING REQUESTS

lots of waiting time
lots of processing time



but on the web, response time is

What actually matters

a scalable server

handles requests fast

+

handles lots of requests

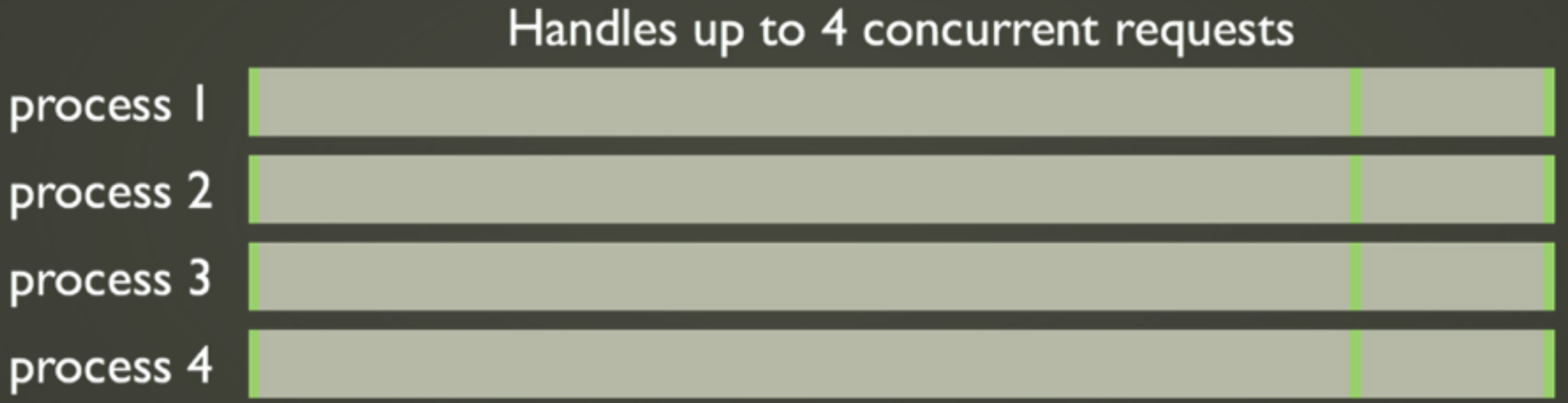
Scale with Threads

Context switching overhead

Execution stacks take up memory

Complicates concurrency

Scale with Processes

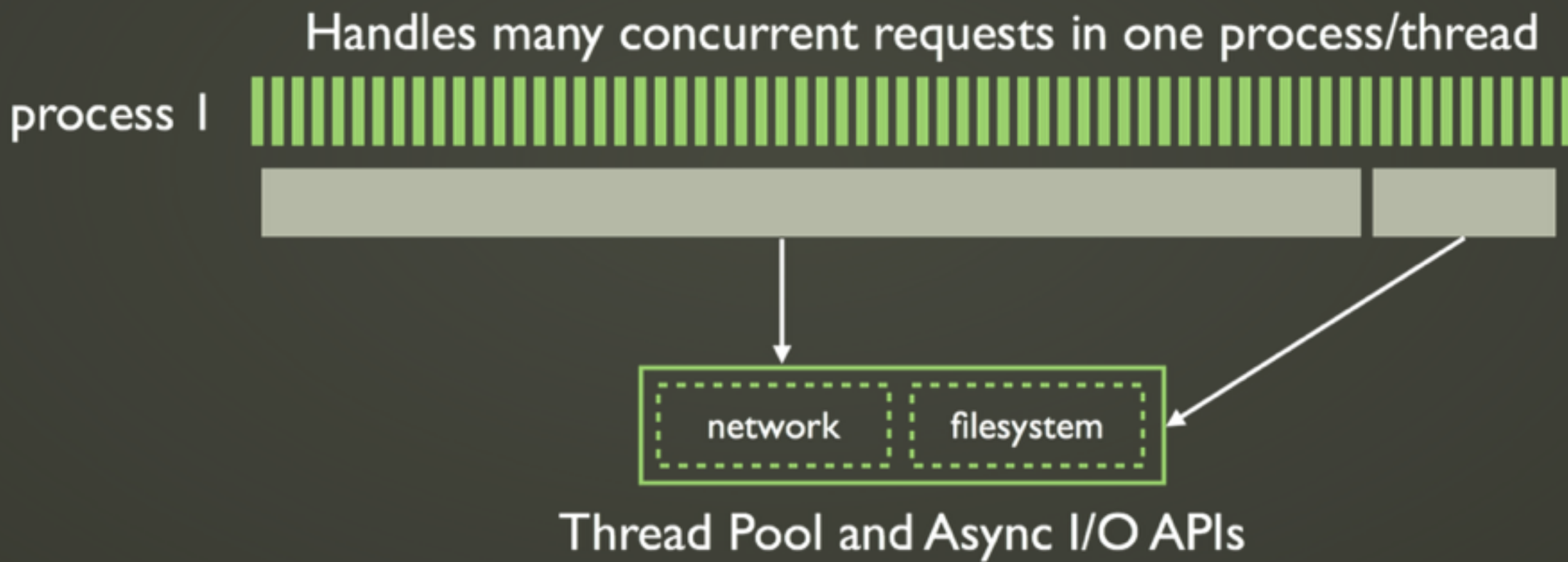


High memory usage

Process scheduling overhead

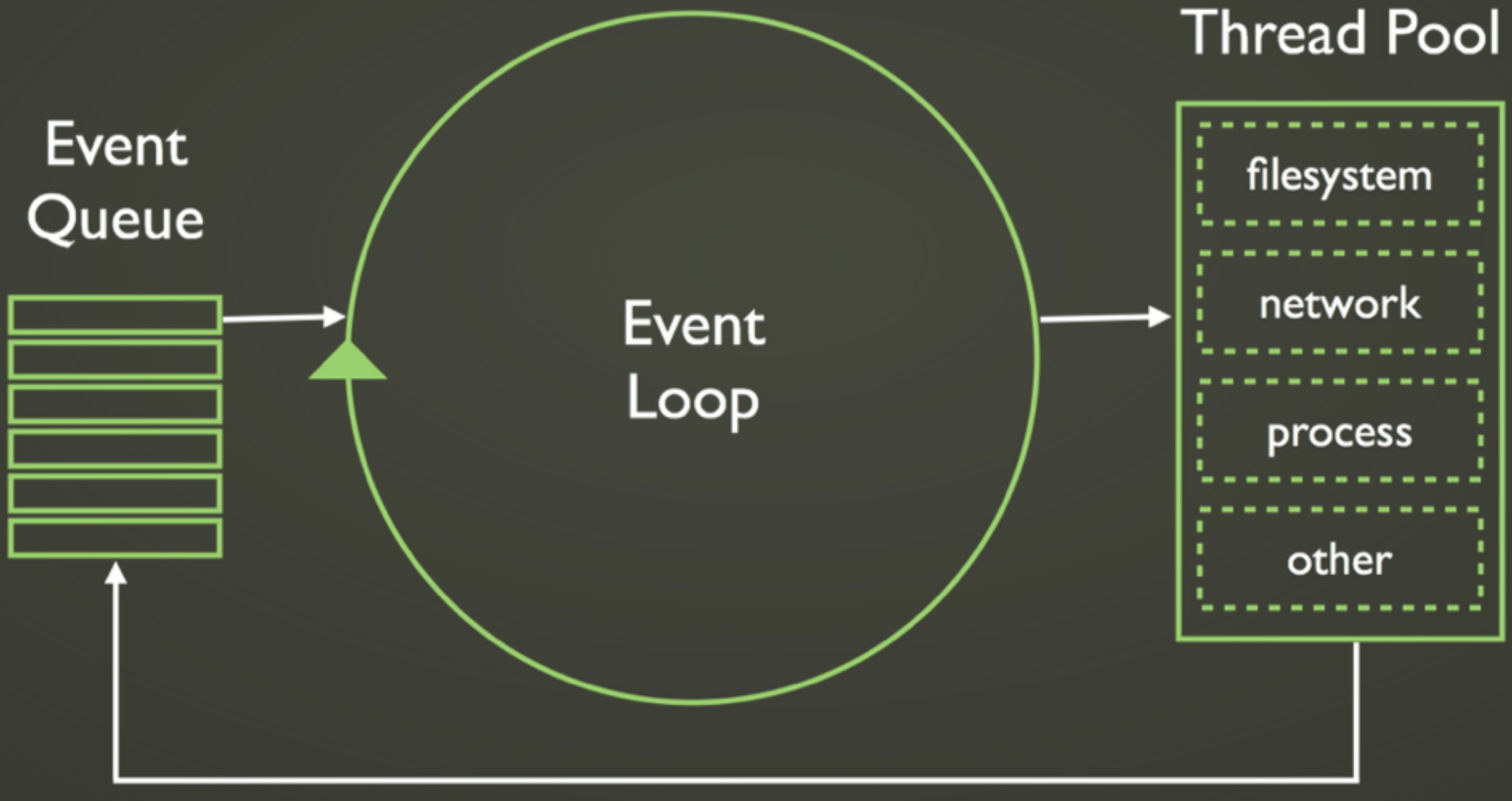
or

Scale with Event Loop



THE EVENT LOOP

THIS IS THE EVENT LOOP



NODE IS SINGLE-THREADED

single thread

many threads

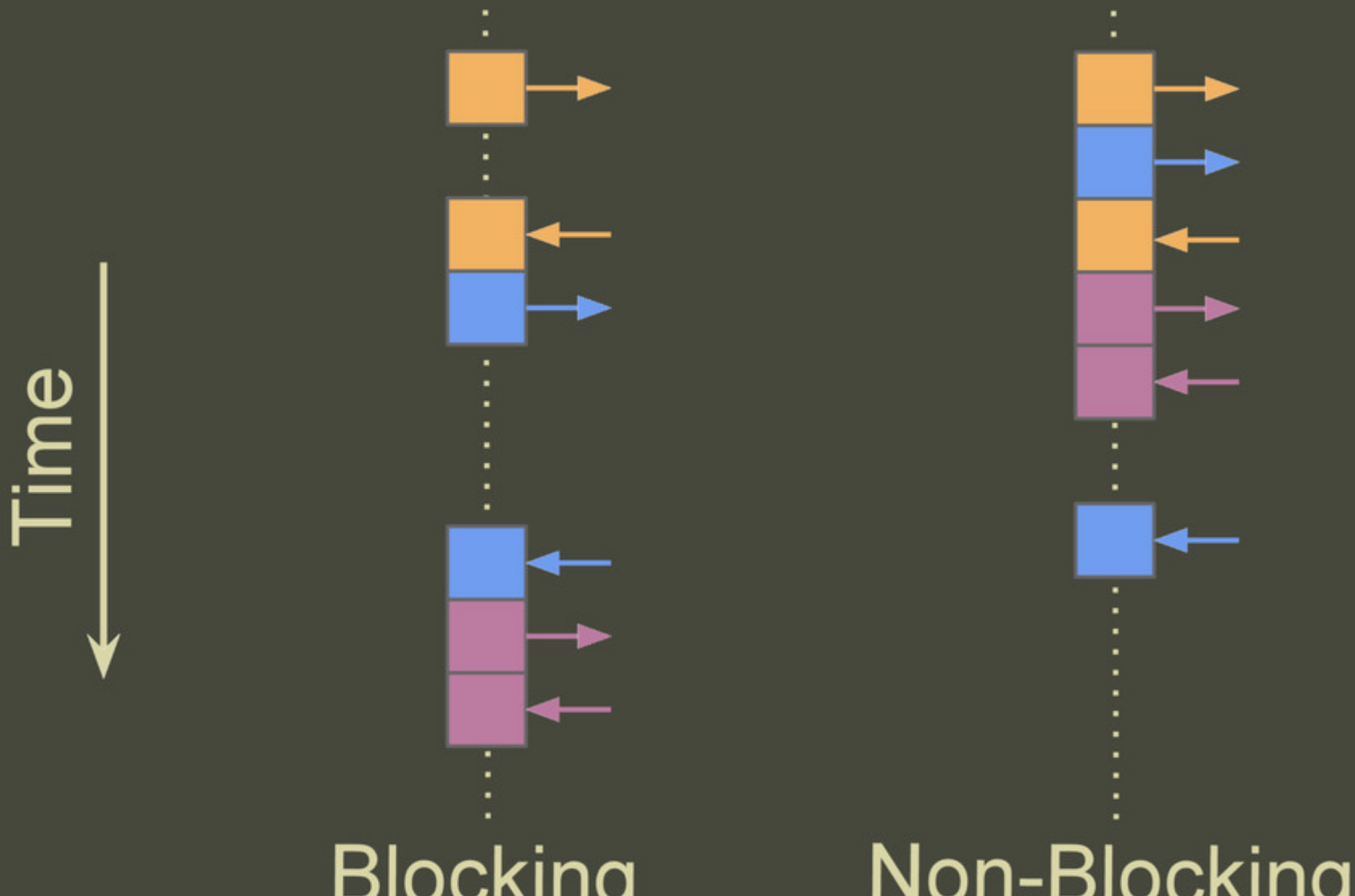
BLOCKING I/O

From the longest traffic jam in history:
http://www.autoevolution.com/news/the-longest-traffic-jam-in-history-12-days-62-mile-long-47237.html

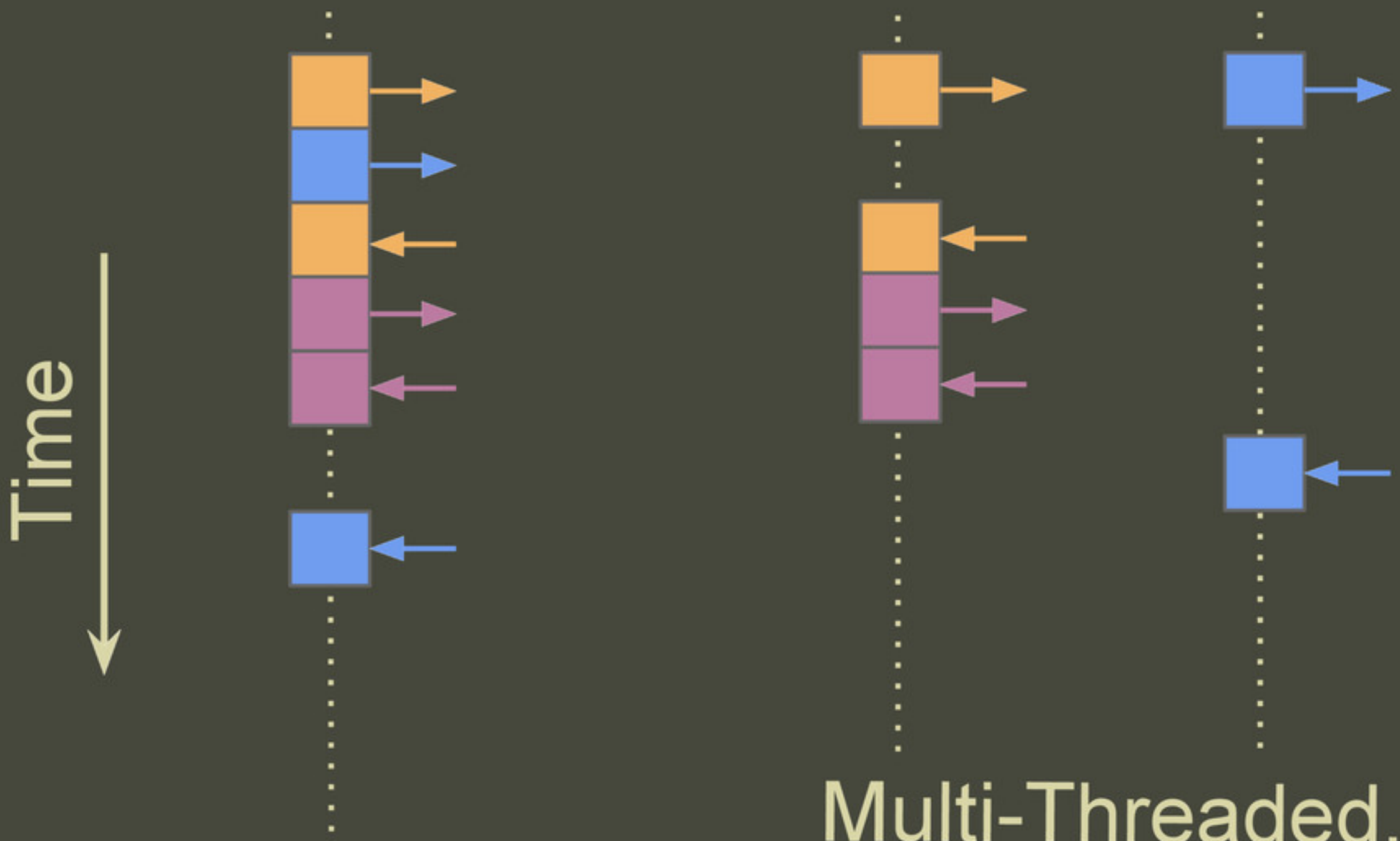
BLOCKING I/O

BLOCKING I/O IS BAD

Single Thread, Non-Blocking



Single Thread, Non-Blocking



Doing I/O:

```php
$contents = file_get_contents("file.txt");
```

PHP

```ruby
contents = File.read "file.txt"
```

Ruby

```csharp
var contents = File.ReadAllText("file.txt");
```

C#

```nodejs
fs.readFile('file.txt', function (err, contents) {
  //do something useful with contents
});
```

Node.JS

What's different about Node?

node's philosophy

node's philosophy

“Do One Thing, and Do It Well”

UNIX®

Celebrating 40 years uptime

Ken Thompson.

He’s the one who said this.

The UNIX Philosophy

(i) Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.

(ii) Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

(iii) Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.

(iv) Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

- Doug McIlroy

The Node Philosophy

1. Write modules that do one thing well. Write a new module rather than complicate an old one.

2. Write modules that encourage composition rather than extension.

3. Write modules that handle data Streams, because that is the universal interface.

4. Write modules that are agnostic about the source of their input or the destination of their output.

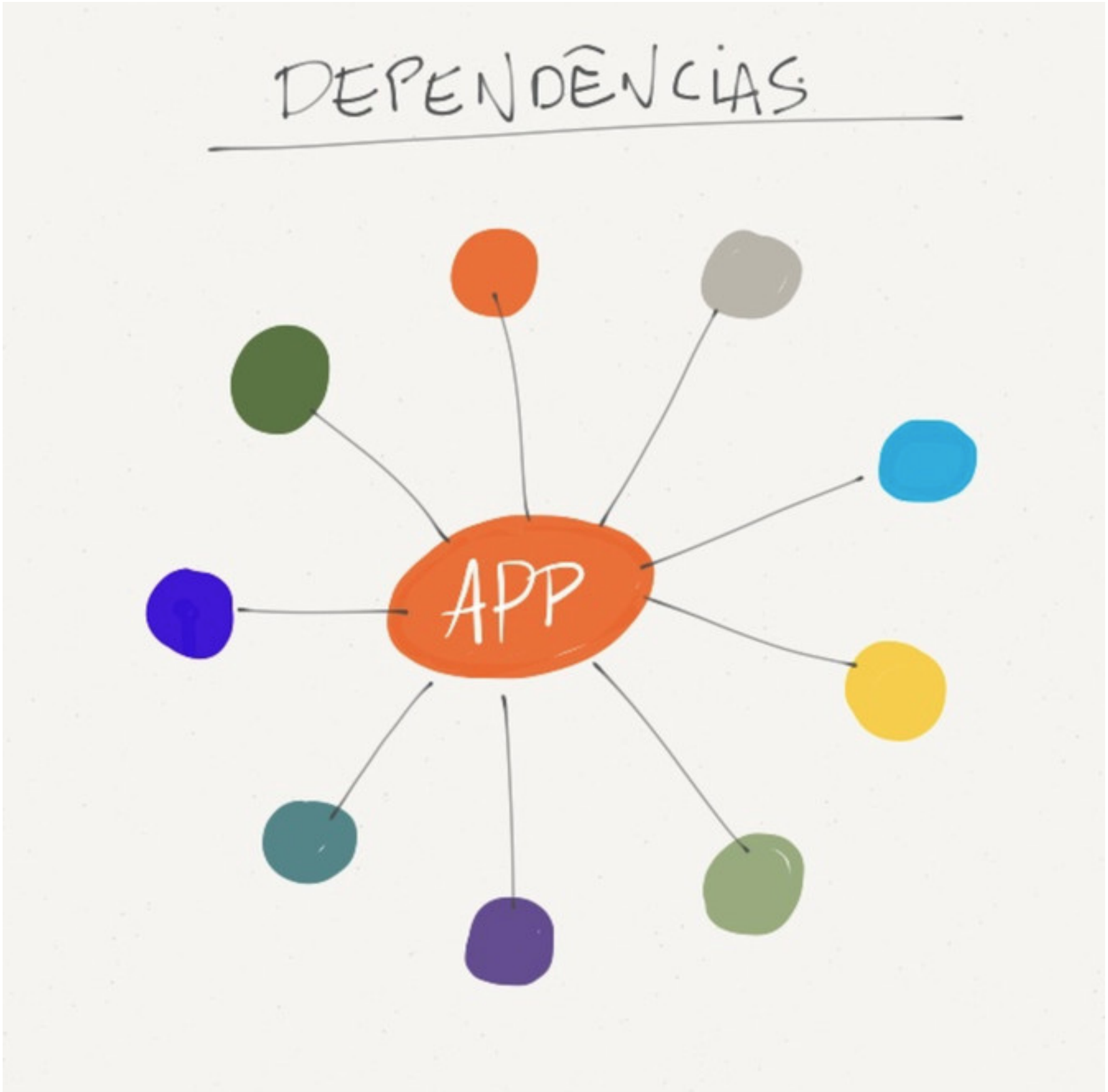
5. Write modules that solve a problem you know, so you can learn about the ones you don't.

6. Write modules that are small. Iterate quickly. Refactor ruthlessly. Rewrite bravely.

7. Write modules quickly, to meet your needs, with just a few tests for compliance. Avoid extensive specifications. Add a test for each bug you fix.

8. Write modules for publication, even if you only use them privately. You will appreciate documentation in the future.

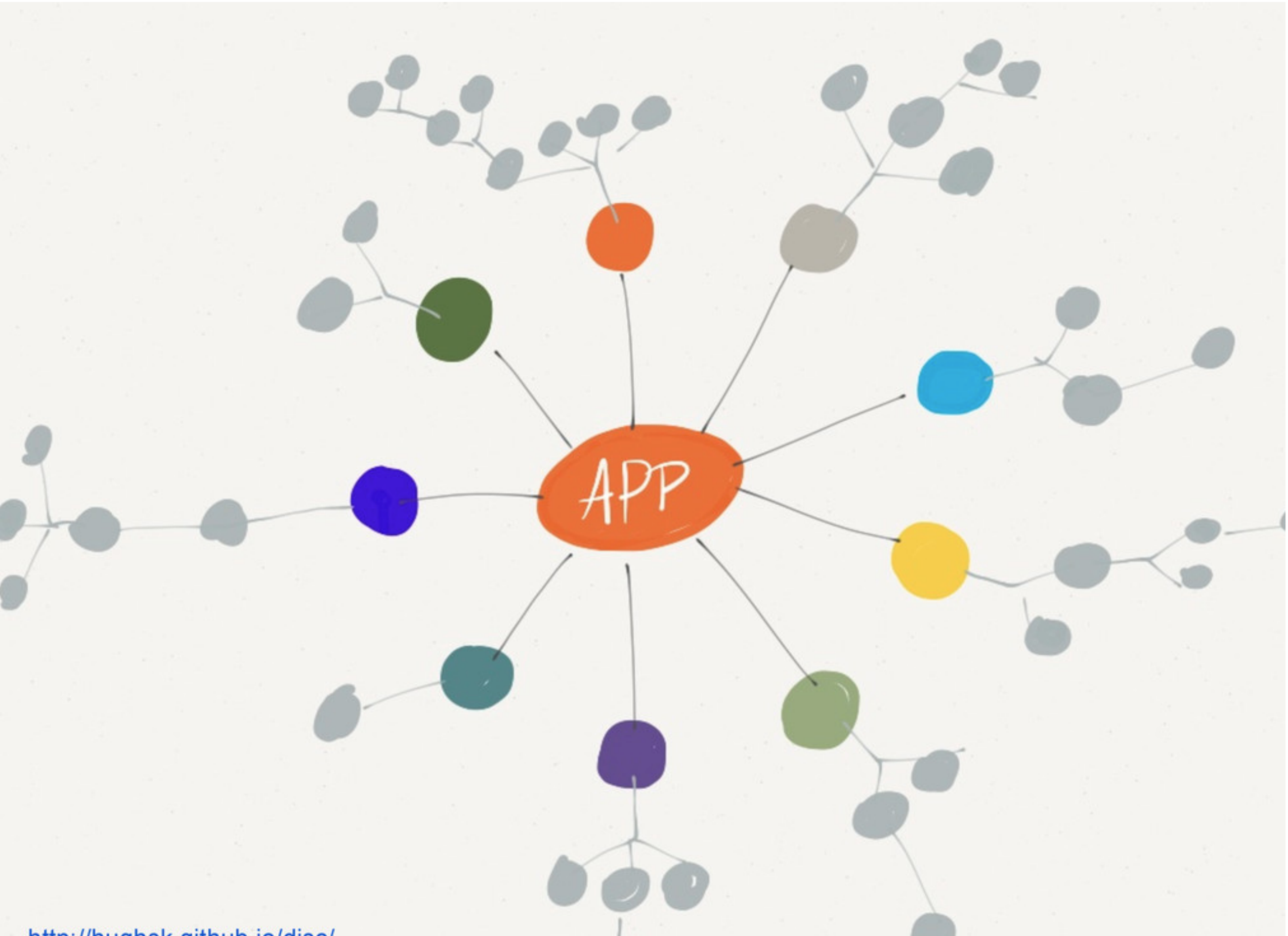
from http://blog.izs.me/post/48281998870/unix-philosophy-and-node-js


DEPENDÊNCIAS
APP

Party with npm

modules : node :: gems : ruby

lots of small, modular packages that can be added to programs via require



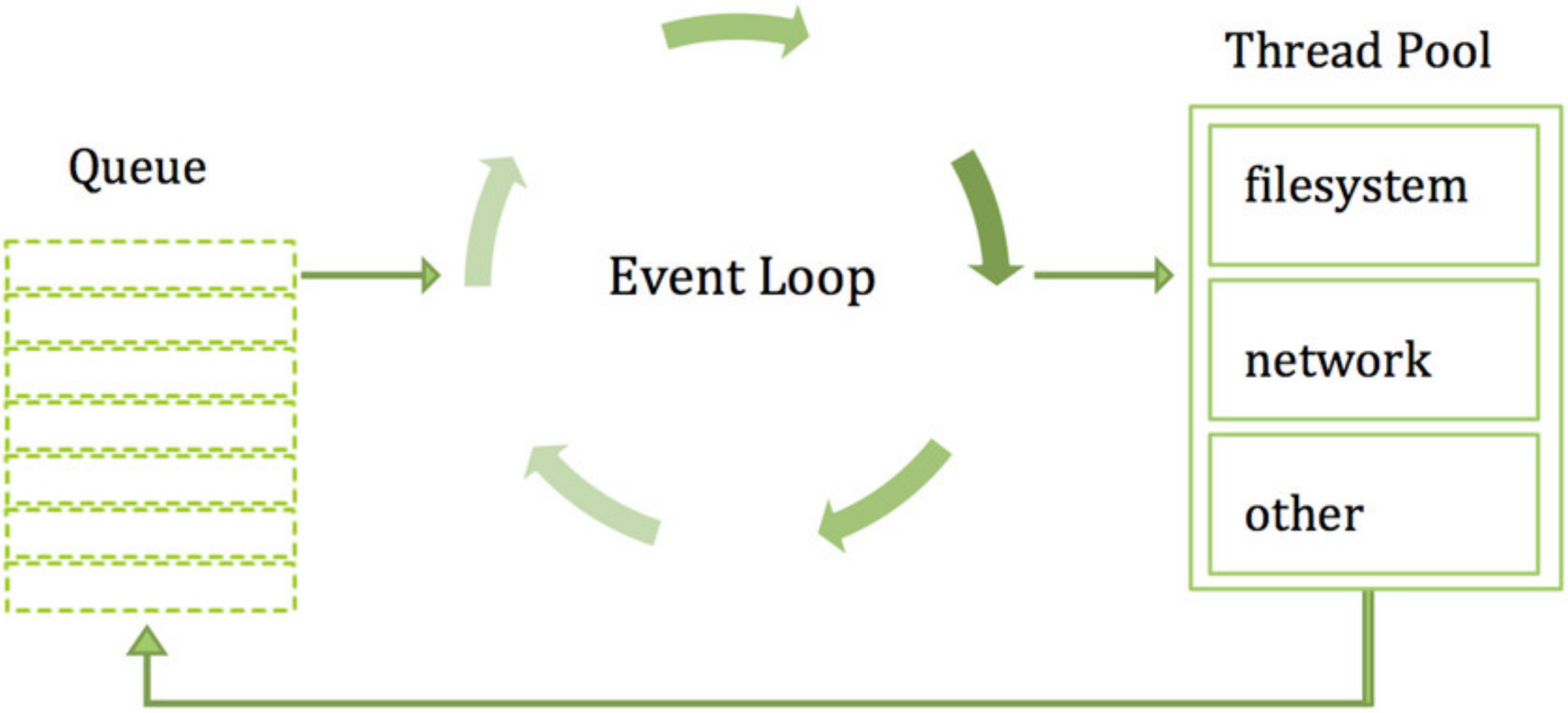
http://hughsk.github.io/diag/

package management

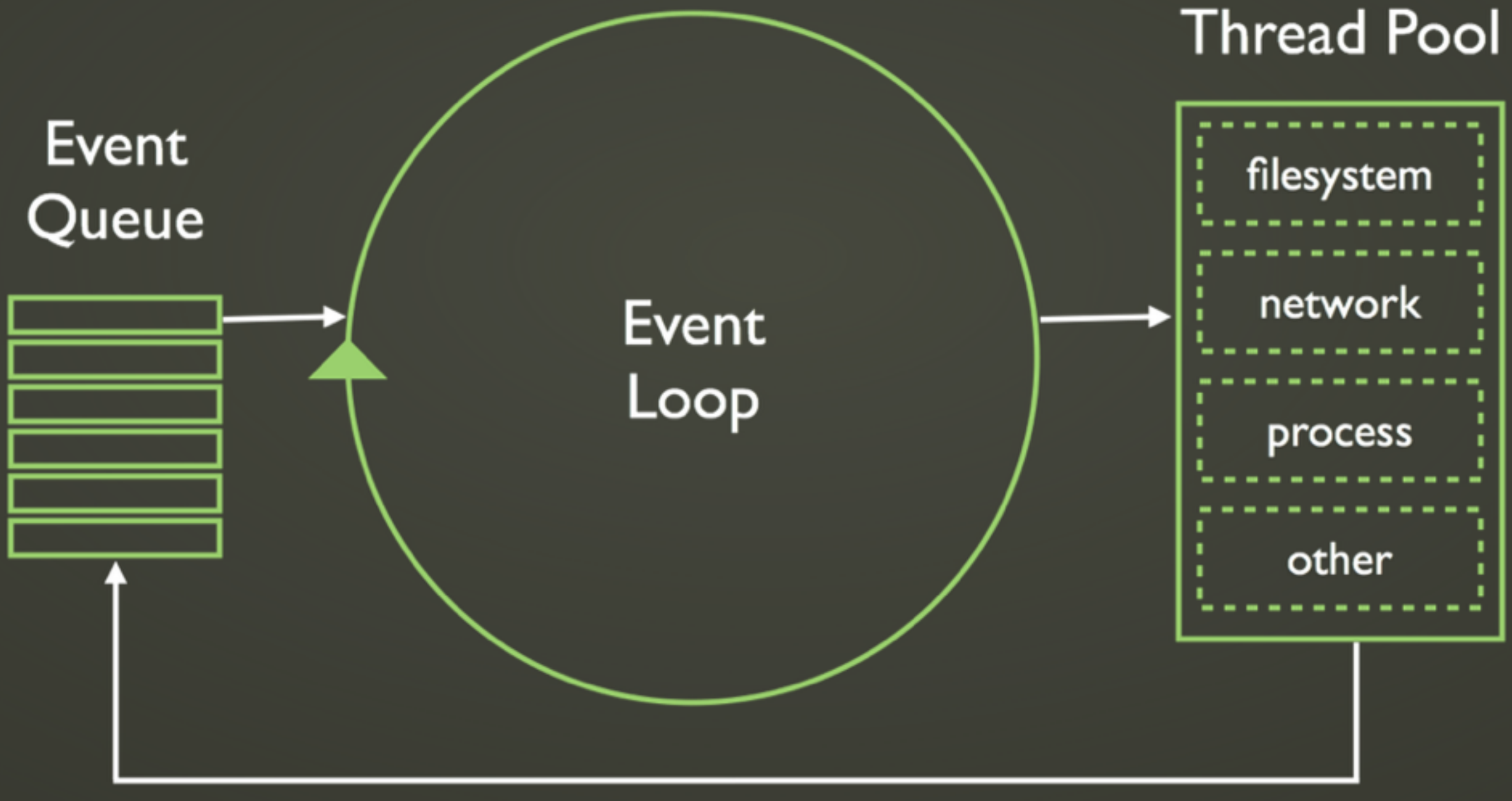
npmjs.org

package.json

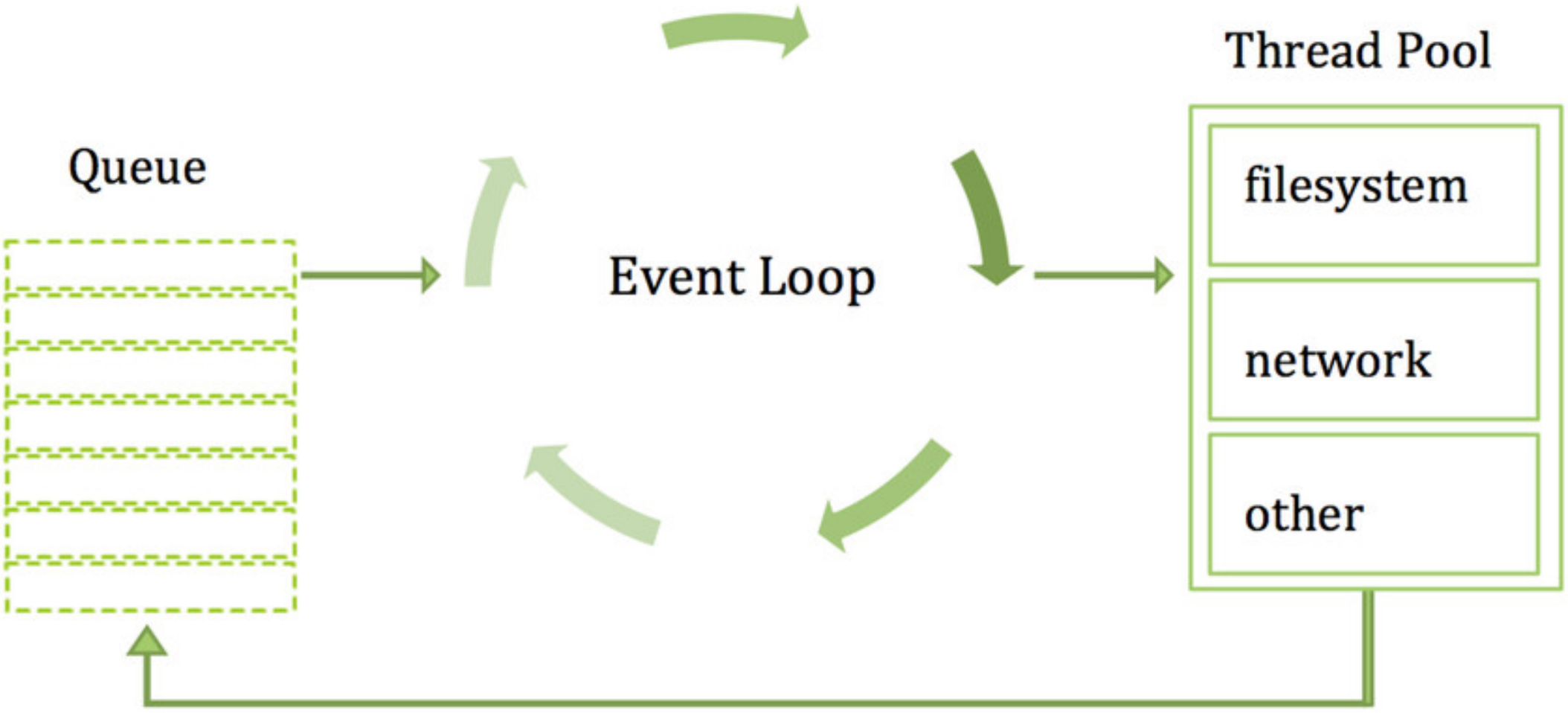
REMEMBER THE EVENT LOOP



THIS IS THE EVENT LOOP



REMEMBER THE EVENT LOOP



THIS IS THE EVENT LOOP