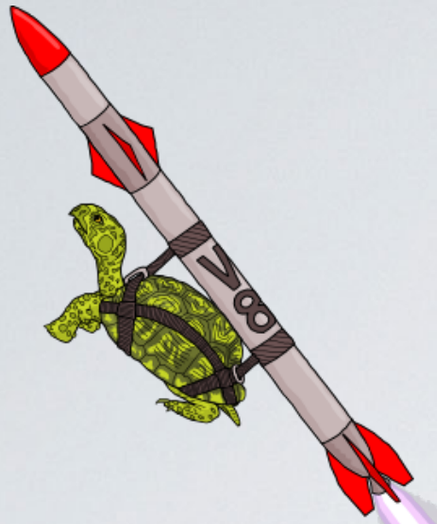


Scaling your NODE JS API like a boss

THE FRONTIERS MINI

http://htmlmag.com/etkinlik/the-frontiers-mini

Volkan Özçelik

http://volkan.io/

June, 20, 2015

Resources

Slides & Source Code

volkan.io

About Me

- JavaScript Lover & Performance Freak

- **Current:**
  - Technical Lead @ Cisco

- **Before:**
  - Mobile Frontend Engineer @ Jive Software
  - VP of Technology @ grou.ps
  - CTO @ cember.net (acquired by Xing)

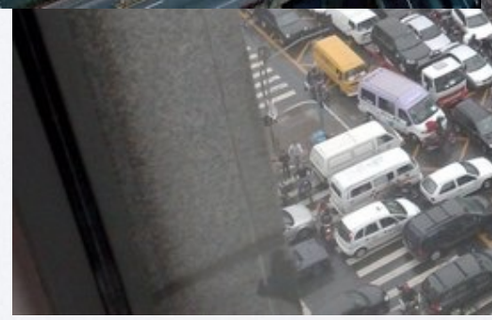
- **Chase me:**
  - @linkibol
  - volkan.io
  - `github.com/v0lkan`
  - speakerdeck.com/volkan
  - linkedin.com/in/volkanozcelik

In a Nutshell…

How do I Architect
a **Scalable** and **Consistent** API
with a Manageable **Complexity**?

Scalability & Consistency
(*in theory*)

Scalability & Consistency
(*in practice*)



Agenda

Environment

- Why APIs?
- Complexity
- Consistency
- Microservices
- Know Your Platform
- Know Your Tools
- Limitations
  - V8 Limitations
  - OS Limitations

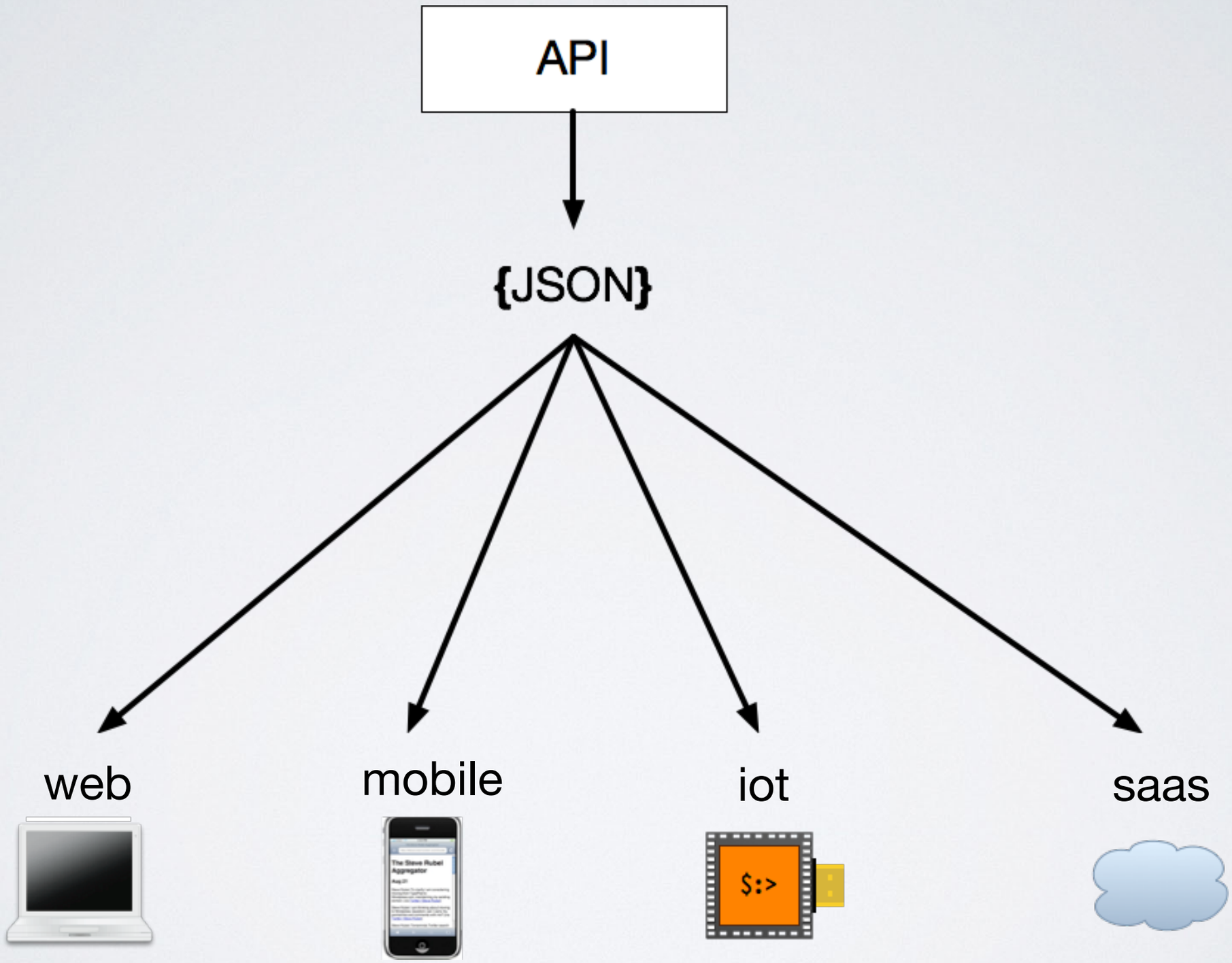
Operations

- Monitoring
  - Throughput
  - Concurrency
  - Event Loop
- Debugging
- Automation
- Logging
- Configuration
- Security

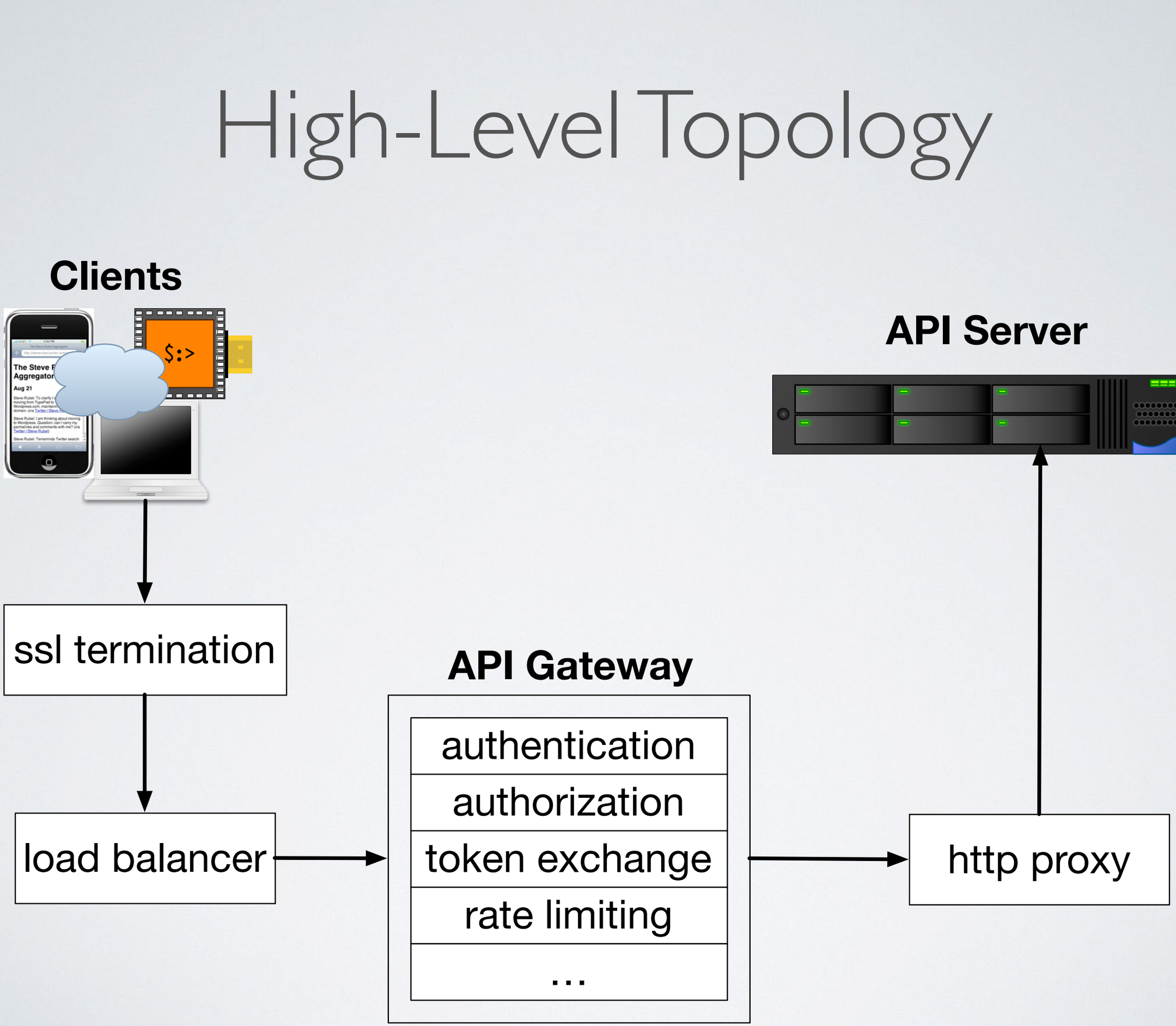
Scaling Up

- Perf Before Scale
  - Memory Leaks
  - IO Optimization
  - Hot Code Paths
- Scaling Your API

Why APIS?



High-Level Topology



Complexity

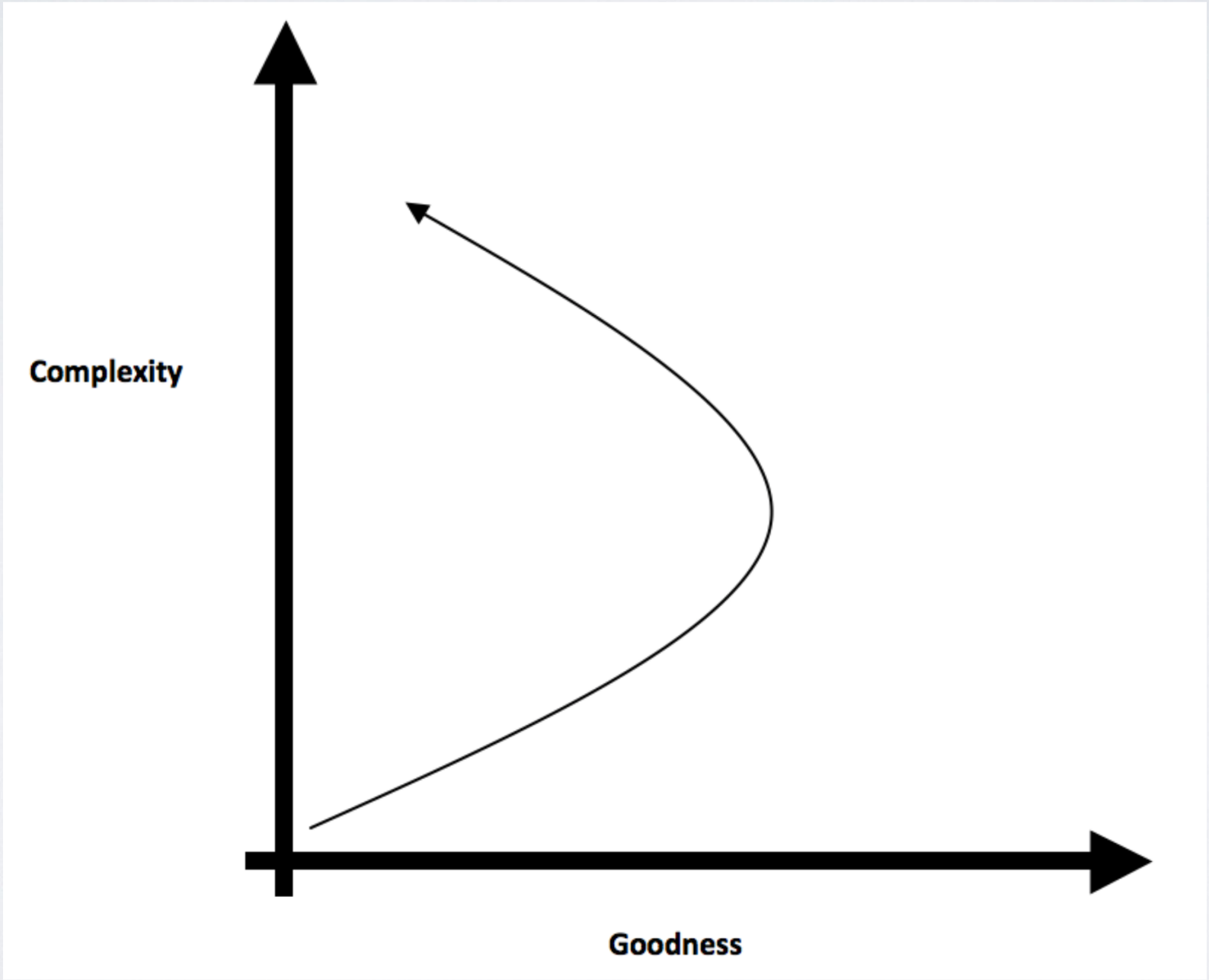
"Through carelessness, inattention, or miscalculation, we may inadvertently overfill [the design] to the detriment of the whole.

Additions once led to improvement.

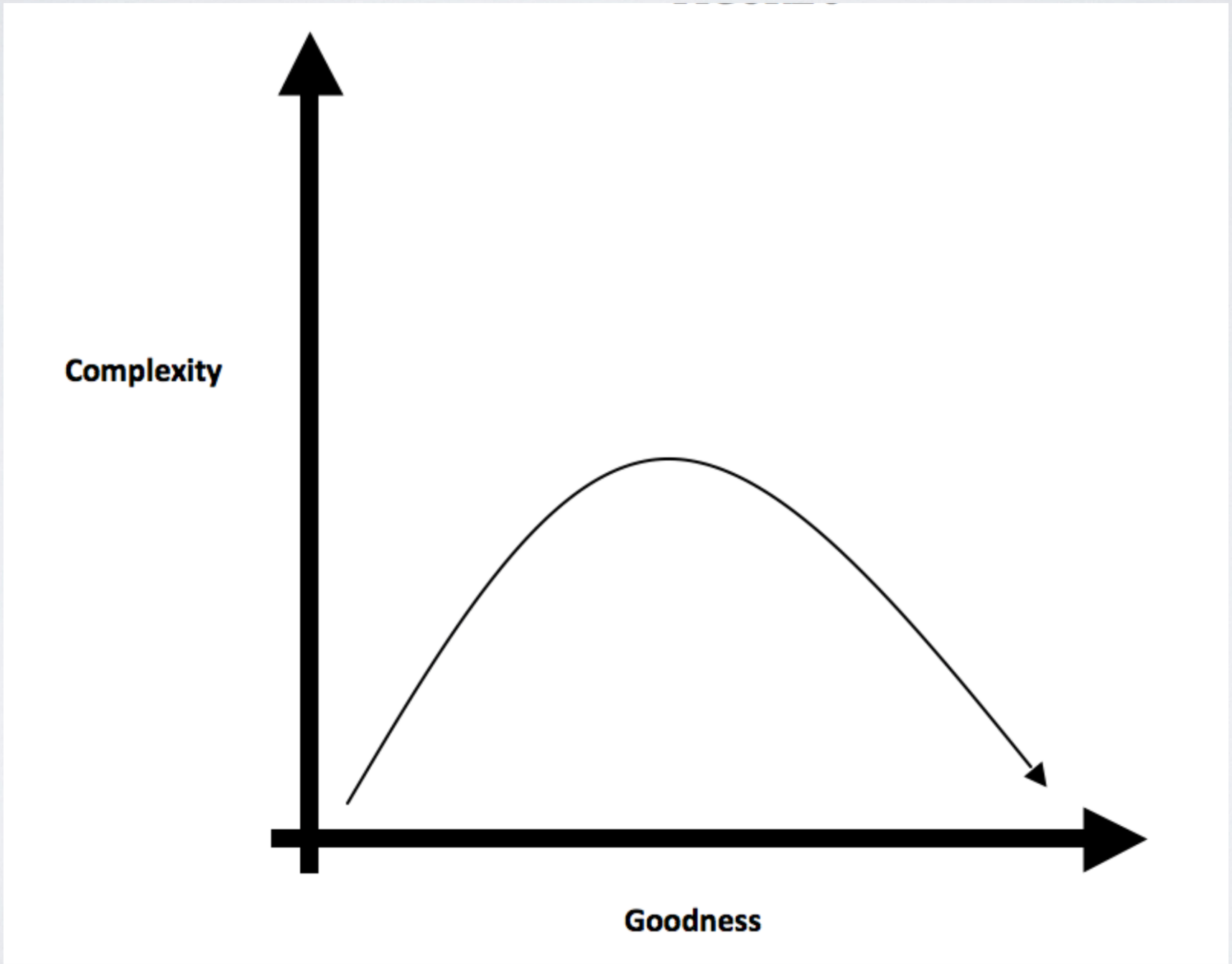
Beyond a certain point, that is no longer the case. **Additions begin to make things worse**."

– Dan Ward

Keep It **Simple**



Keep It **Simple**r



Find a Balance

Consistency

Eventual Consistency

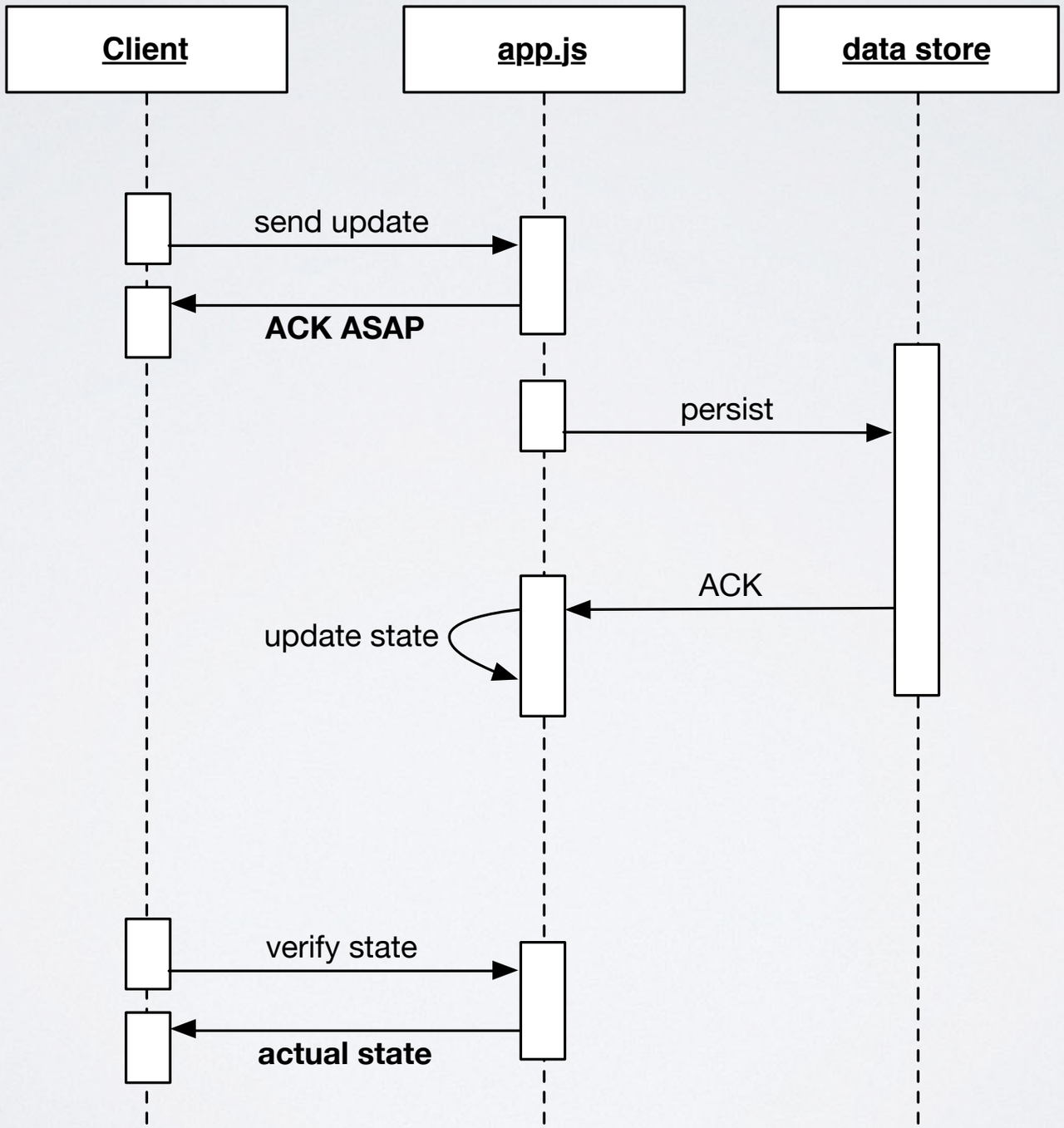
Eventual consistency is a consistency model used in distributed computing to achieve **high availability** that guarantees that, if no new updates are made to a given data item, **eventually** all accesses to that item will return the last updated value.

see also: "Event Sourcing"
(http://martinfowler.com/eaaDev/EventSourcing.html)

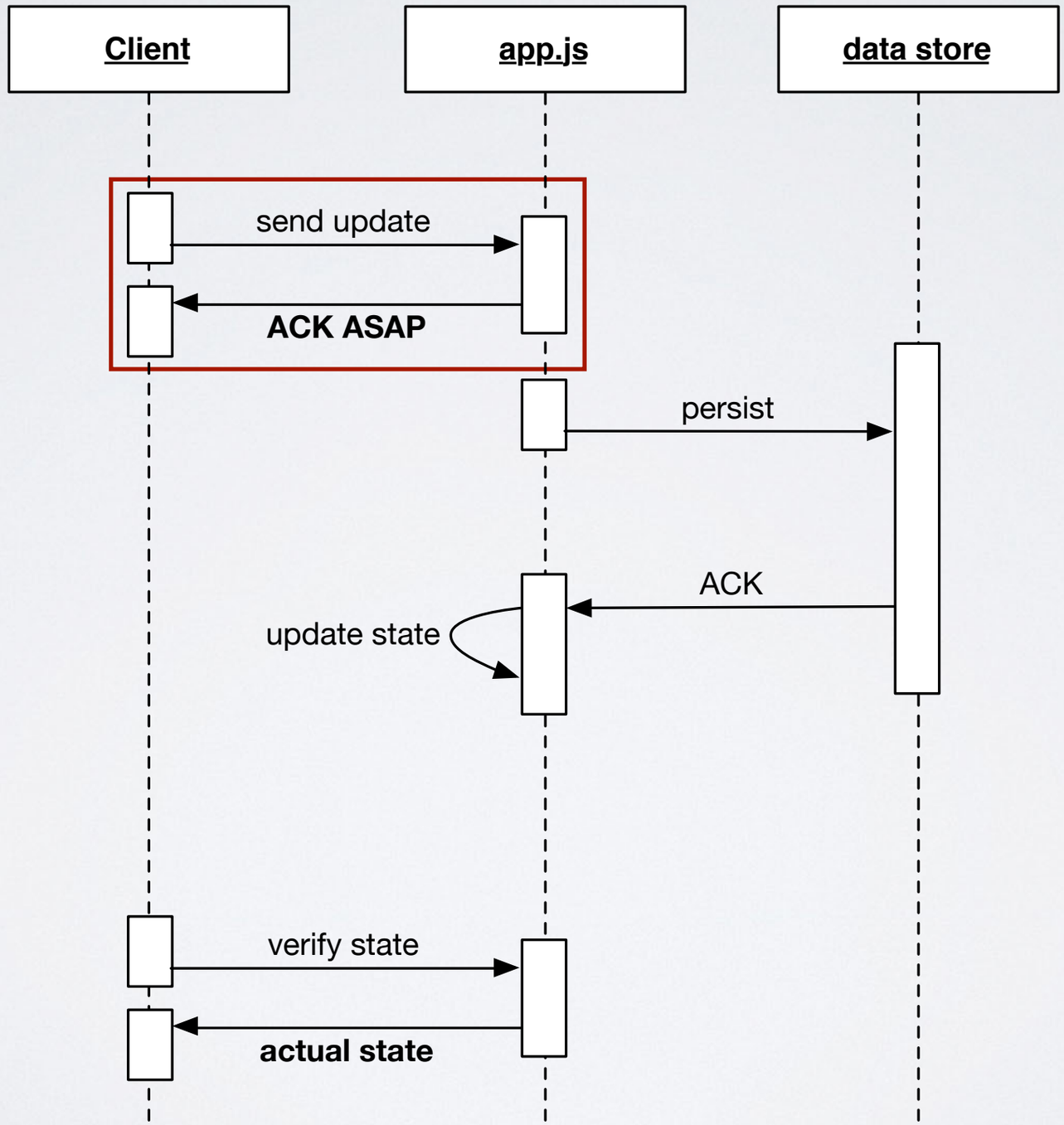
99% of the Time Eventual Consistency is Good Enough

Transactional Consistency in a distributed system
is hard to achieve and expensive to implement.

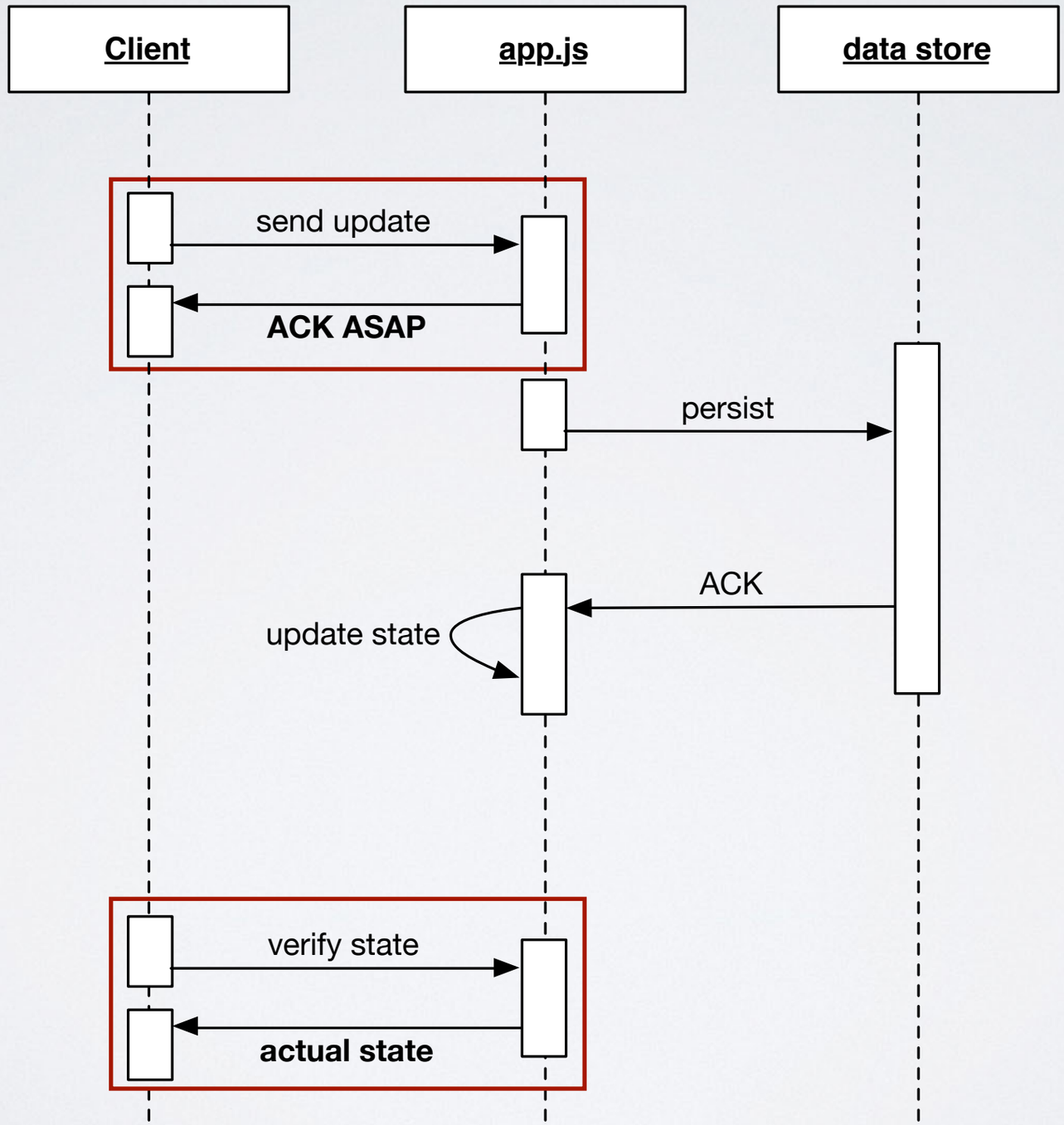
Be Optimistic



Be Optimistic

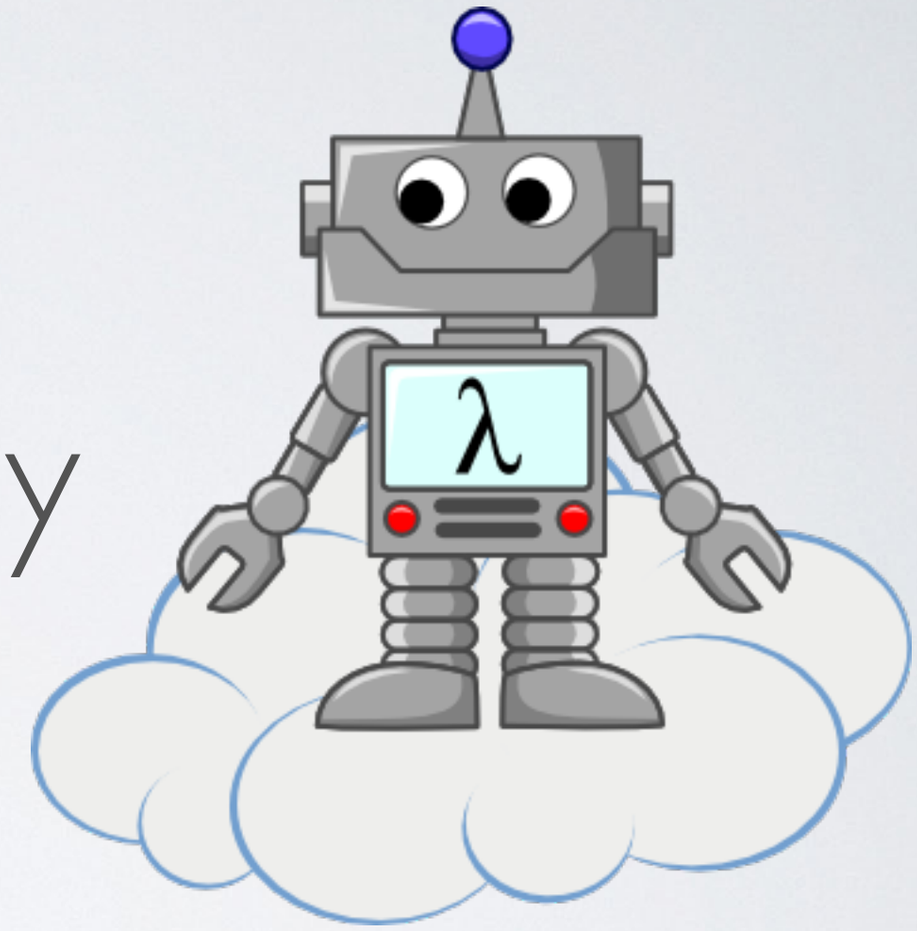


Be Optimistic



Before Everything

Think Functionally



volkan.io

@linkibol

Think Functionally

- Delegate; Don't Return
- Avoid Side Effects
- **Avoid State**
  - Easier to Scale Horizontally
  - Easier to Swap/Reboot Instances
  - Session Affinity is Not A Problem
  - Better Performance

Think Functionally

- **Understand Control Flow Patterns**
- Understand **Promise** Patterns (*and anti-patterns*)
- Avoid `this` and avoid `new` as much as you can.*
  - You'll thank me later.
- Think in **Streams**
  - **Translate**, **Transform**, **Reduce**

* https://medium.com/javascript-scene/the-two-pillars-of-javascript-ee6f3281e7f3

Think Functionally

1. Cook ground beef, onion, and garlic over medium heat until well browned.
2. Stir in crushed tomatoes, tomato paste, tomato sauce, and water.
3. Season with sugar, basil, fennel seeds, Italian seasoning, stirring occasionally.
4. Bring a large pot of lightly salted water to a boil.
5. Cook noodles in boiling water for 8 to 10 minutes.
6. Drain noodles, and rinse with cold water.
7. In a mixing bowl, combine ricotta cheese with egg, remaining parsley.
8. Arrange 6 noodles lengthwise over meat sauce.
9. Spread with one half of the ricotta cheese mixture.
10. Top with a third of mozzarella cheese slices.
11. Spoon 2 cups meat sauce over mozzarella.
12. Repeat layers, and top with remaining mozzarella and Parmesan cheese.
13. Bake in preheated oven for 25 minutes.
14. Remove foil, and bake an additional 25 minutes.
15. Cool for 15 minutes before serving.

Think Functionally

REDUCTION

Lasagna is grated cheese on cheese sauce on flat pasta on cheese sauce on bolognese on flat pasta on cheese sauce on bolognese on flat pasta on cheese sauce baked for 45 minutes.

TRANSFORMATION

bolognese is onion and oil fried until golden mixed with ground beef mixed with tomato simmered for 20 minutes.

TRANSFORMATION

cheese sauce is milk and cheese added progressively to roux while frying it until the sauce thickens.

TRANSFORMATION

roux is flour and butter fried briefly.

TRANSLATION

baked is put in an oven dish in a hot oven.

TRANSLATION

fried is put in a pan on high and mixed frequently.

TRANSLATION

simmered is put in a pan on low and mixed infrequently.

Know Your Platform

JavaScript

Perf Advice Is Addictive

Learn JavaScript Before the Deep Dive

- Understand EcmaScript 5
  - scope, hoisting, closures, promises, `this`, all that jazz.
- Keep an Eye on EcmaScript 6
  - Consider Using EcmaScript 6 for new Projects

Fact Check

- "JavaScript is **like Java, but easier.**"
- "Node.JS is slow."
- "I can't let the app restart; it will take too long."
- "Node.JS is Insecure." (*no! **people are**.*)
- "Node.JS does not support feature **x**."

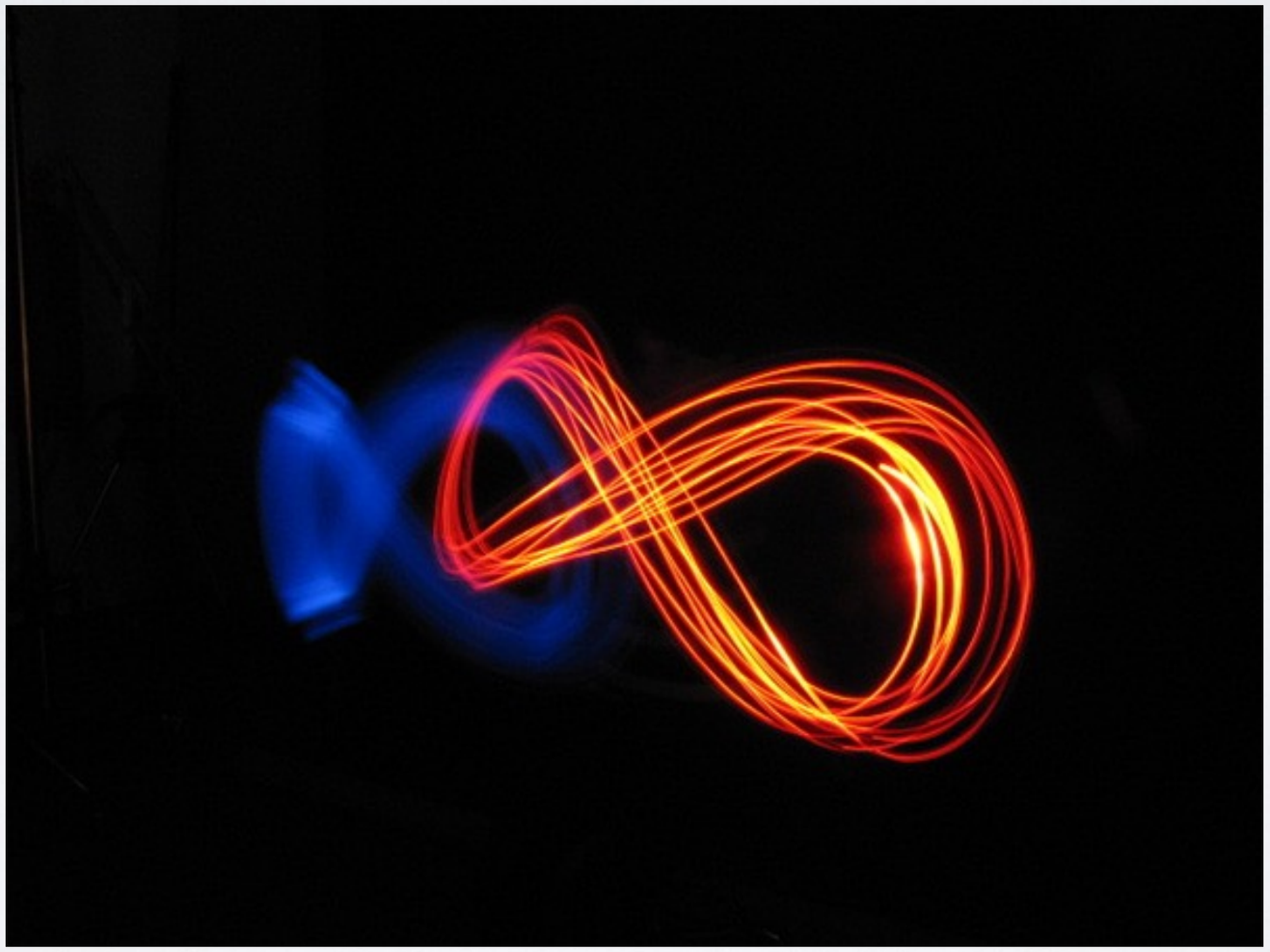
Node.JS Is **Perfect** For…

- IO-Heavy Applications
- Data-Intensive Realtime Apps
- RESTful / API-Driven (Micro)services
- Streams
  - Queued (Lazy) Writes
  - Processing data on-the-fly

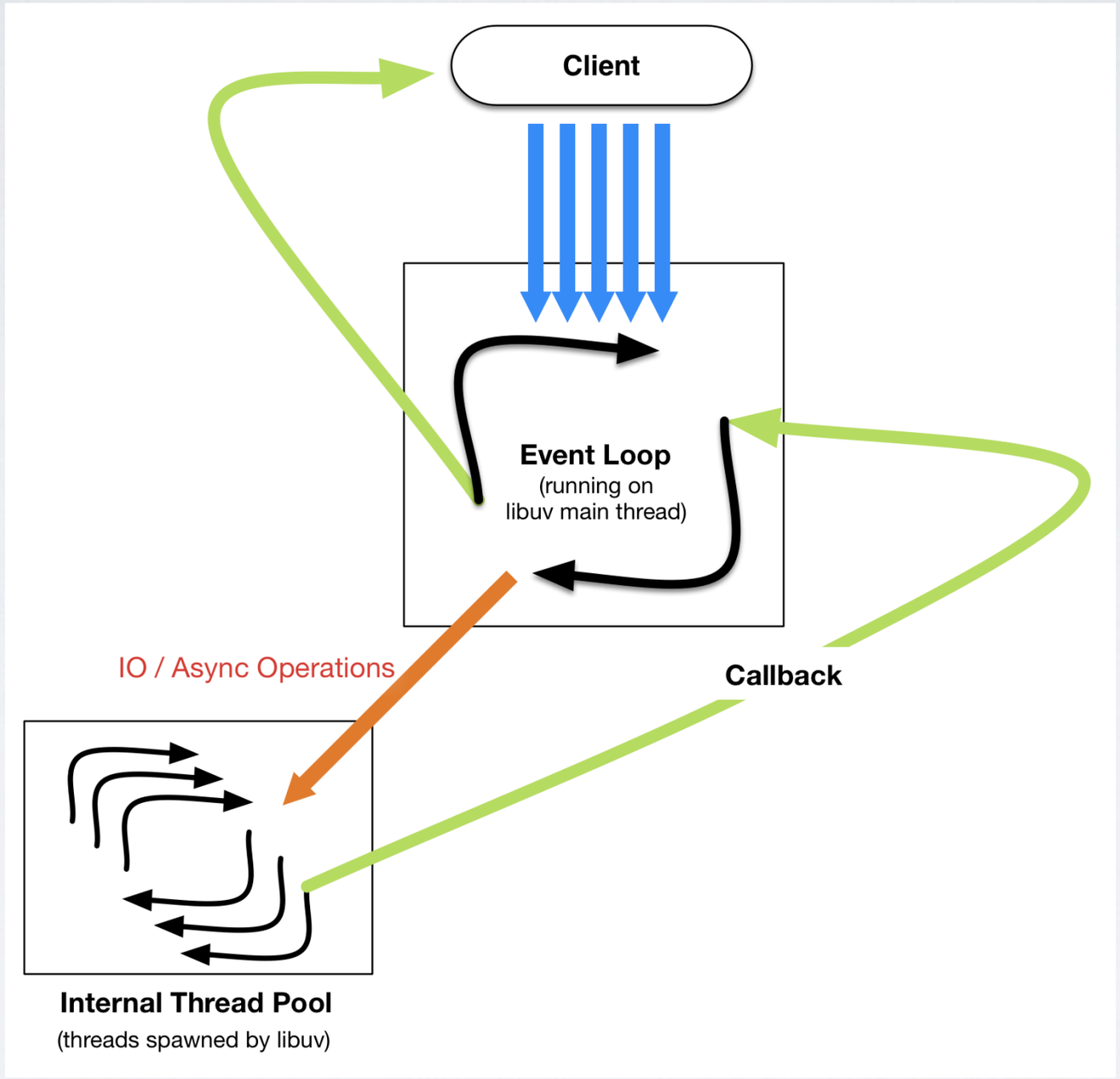
Node.JS Is **not** For…

- Serving Static Files
- CPU-bound Applications
- Creating a Monolithic Infrastructure

The Event Loop



Node.JS Event Model



* http://nikhilm.github.io/uvbook
* https://strongloop.com/strongblog/node-js-event-loop/

Never Block The Event Loop

- First rule of **Node.JS** Programming:
  - **Never Block the Event Loop**

- Second rule of **Node.JS** Programming:
  - **Never Block the Freaking Event Loop!**

- Corollary:
  - Delegate long (>10ms) Tasks to a Worker
    - child_process (*)
    - native add-ons (**)

* https://nodejs.org/api/child_process.html
** https://nodejs.org/api/addons.html

Know the Ecosystem

- Do Not Ignore The Ecosystem
- Follow Community News and Updates
- Attend to Conferences (*like this one*)

Know the Ecosystem

- **npm** (as of June 20, 2015)
  - **155,880** (and increasing) total packages
  - 70,624,534 downloads yesterday
  - 389,190,331 downloads in the last week
  - **1,609,312,413 downloads in the last month**

Embrace Open Source

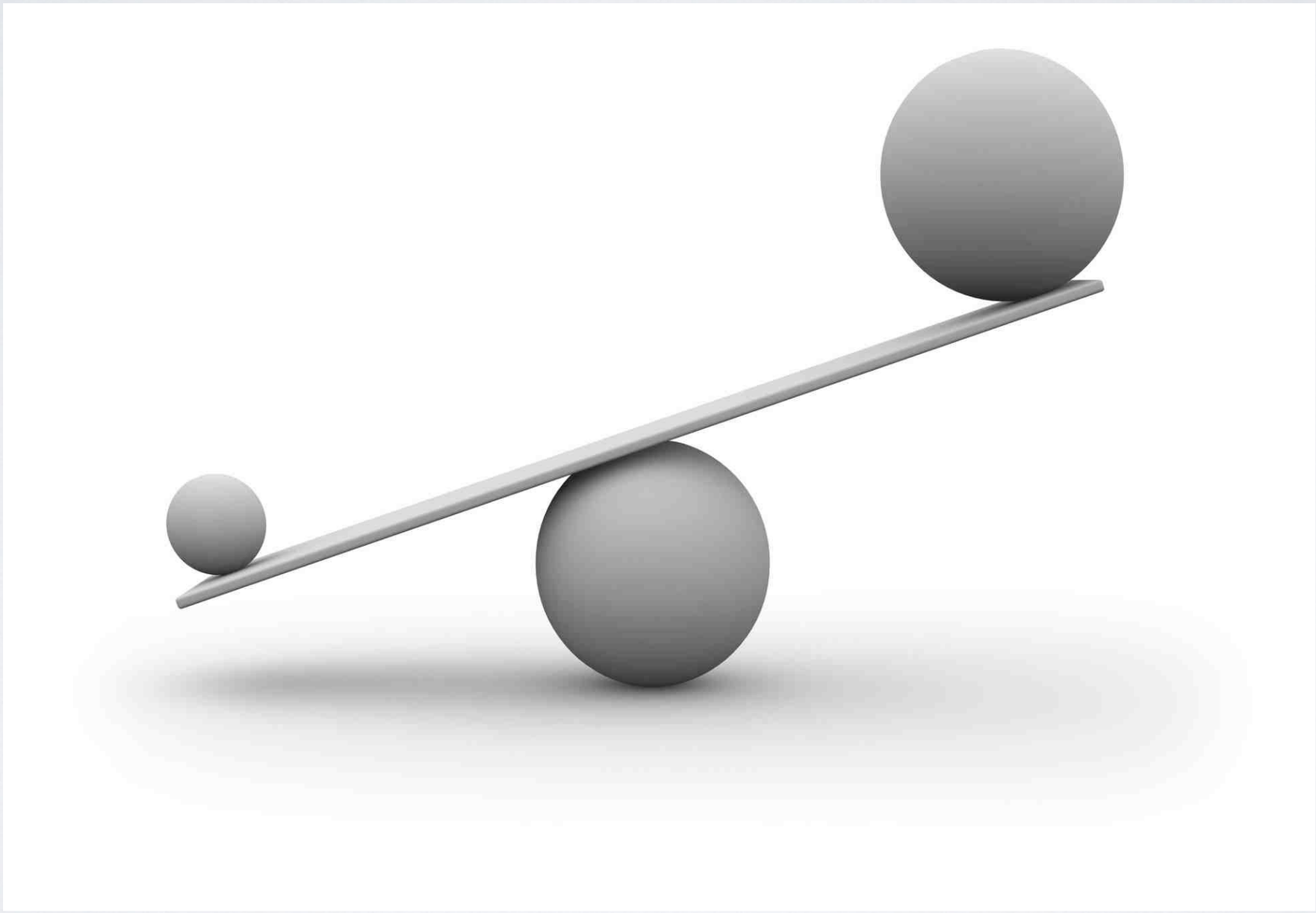
and contribute back

Know Your Tools

Node.JS is **not** a Swiss army knife

- Load Balancing ➡ **haproxy** (http://www.haproxy.org/)
- Web Server ➡ **nginx** (http://nginx.org/)
- SSL Termination ➡ **stud** (https://github.com/bumptech/stud)
- GZIP Compression ➡ **nginx** / haproxy
- Static Assets ➡ CDN / **Varnish** (https://www.varnish-cache.org/)

Limitations



v8 Limitations

volkan.io

@linkibol

v8 Limitations

- **~2gb ➡** heap limit (--max-old-space-size)
  - will be more than enough
  - spawn more processes for more (child_process)
- **~1gb ➡** max size of a Buffer

Show Love to Streams

- Lazily produce or consume data in buffered chunks.
- Evented and non-blocking.
- Low memory footprint.
- Automatically handle back-pressure.
- Buffers allow you to **work around the v8 heap memory limit**.
- Most core node.js content sources/sinks are streams already.

OS Limitations

volkan.io

@linkibol

Open File Limits

- "Error: EMFILE, Too many open files"
  - ***ulimit -n 65535;***

Open File Limits

```
vim gedit /etc/security/limits.conf;

    web  soft  nofile 9000
    web  hard  nofile 65000
    root soft  nofile 9000
    root hard  nofile 65000

vim /etc/pam.d/common-session;

    session required pam_limits.so
```

Configuring the Load Balancer

```
1   user web;
2
3   worker_processes 8;
4   worker_rlimit_nofile 65535;
5
6   pid /run/nginx.pid;
7
8   events {
9       worker_connections 65535;
10      multi_accept on;
11      use epoll;
12  }
13
14  http {
15      client_body_buffer_size 10K;
16      client_header_buffer_size 1k;
17      client_max_body_size 8m;
18      large_client_header_buffers 2 1k;
19
20      client_body_timeout   32;
21      client_header_timeout 32;
22
23      ...
```

see https://bit.ly/nginx-rocks

Configuring the Load Balancer

```
1   user web;
2
3   worker_processes 8;
4   worker_rlimit_nofile 65535;
5
6   pid /run/nginx.pid;
7
8   events {
9       worker_connections 65535;
10      multi_accept on;
11      use epoll;
12  }
13
14  http {
15      client_body_buffer_size 10K;
16      client_header_buffer_size 1k;
17      client_max_body_size 8m;
18      large_client_header_buffers 2 1k;
19
20      client_body_timeout   32;
21      client_header_timeout 32;
22
23      ...
```

see https://bit.ly/nginx-rocks

Additional Tweaks

http.globalAgent.maxSockets = SOME_BIG_NUMBER;

(if you have many outgoing requests)

Additional Tweaks

sysctl -w net.core.somaxconn=1024;

(default is 128)

Even More Tweaks

```
vim /etc/sysctl.conf

    fs.file-max = 1000000
    fs.nr_open = 1000000
    net.ipv4.netfilter.ip_conntrack_max = 1048576
    net.nf_conntrack_max = 1048576
    net.core.rmem_max = 33554432
    net.core.wmem_max = 33554432
    net.ipv4.tcp_rmem = 4096 16384 33554432
    net.ipv4.tcp_wmem = 4096 16384 33554432
    net.ipv4.tcp_mem = 786432 1048576 26777216
    net.ipv4.tcp_max_tw_buckets = 360000
    net.core.netdev_max_backlog = 2500
    net.ipv4.ip_local_port_range = 1024 65535
```

Monitoring

You Can't Optimize What You Don't Measure

volkan.io

@linkibol

Things to Watch Out For

- Your API Service **may** Become CPU-Bound
- External API Calls Can Be a Bottleneck
- **Always Keep an Eye on the Event Loop**
- Implement Sanity Checks
  - Circuit Breaker
  - **Have an Upper Bound for Concurrency**

Things to Watch Out For

- Is app running and functional?
- Is app overloaded?
- How many errors have been raised so far?
- How many times do forks restored?
- Are all forks alive and okay?
- Is app performant (*throughput, memory utilization, concurrency*)?

Which Will (*most of the time*) Boil Down to…

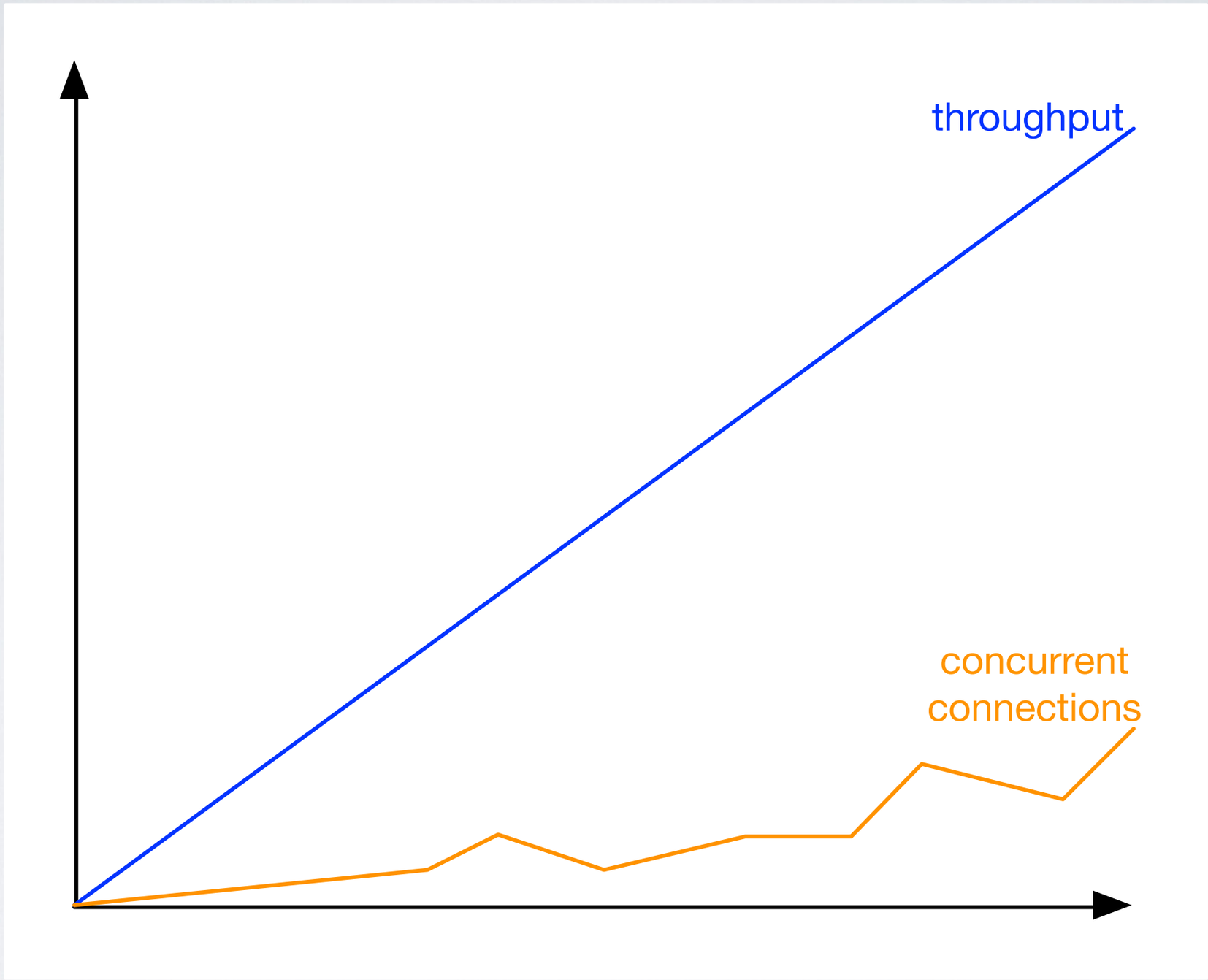
- Watching **Response Times**
- Watching **CPU Utilization** + General Sys Resource Usage
- Watching Number of **Concurrent Connections**

Throughput / Concurrency

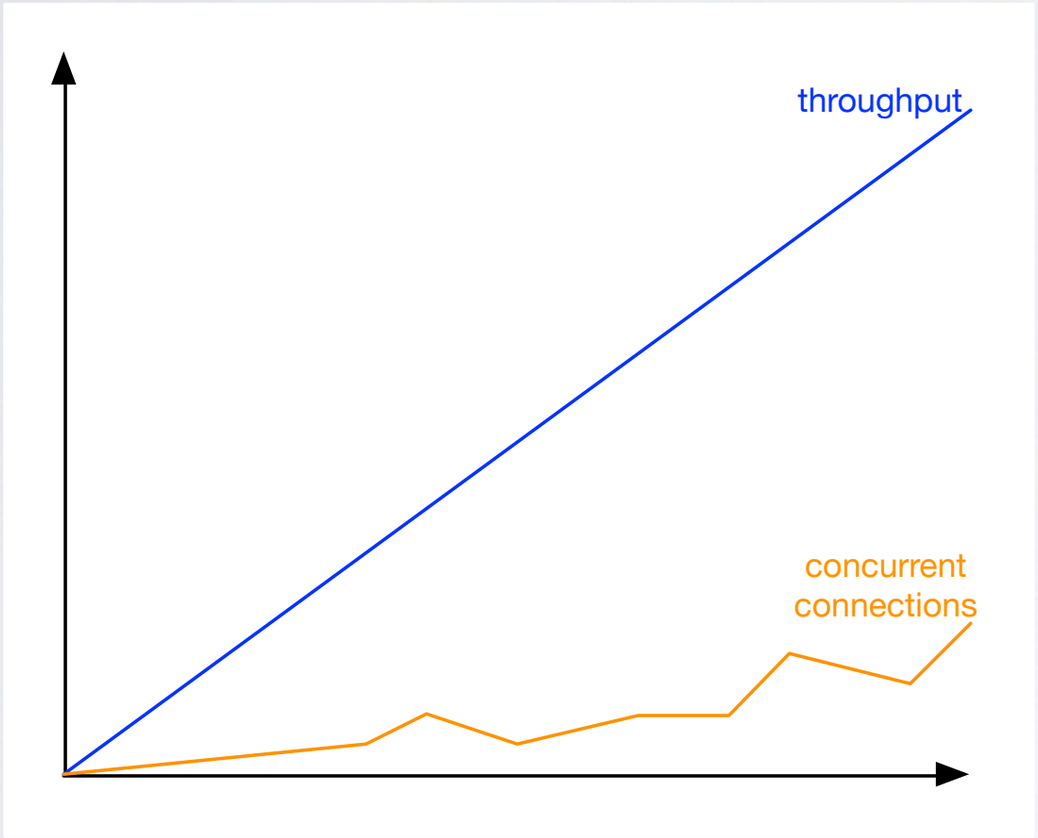
volkan.io

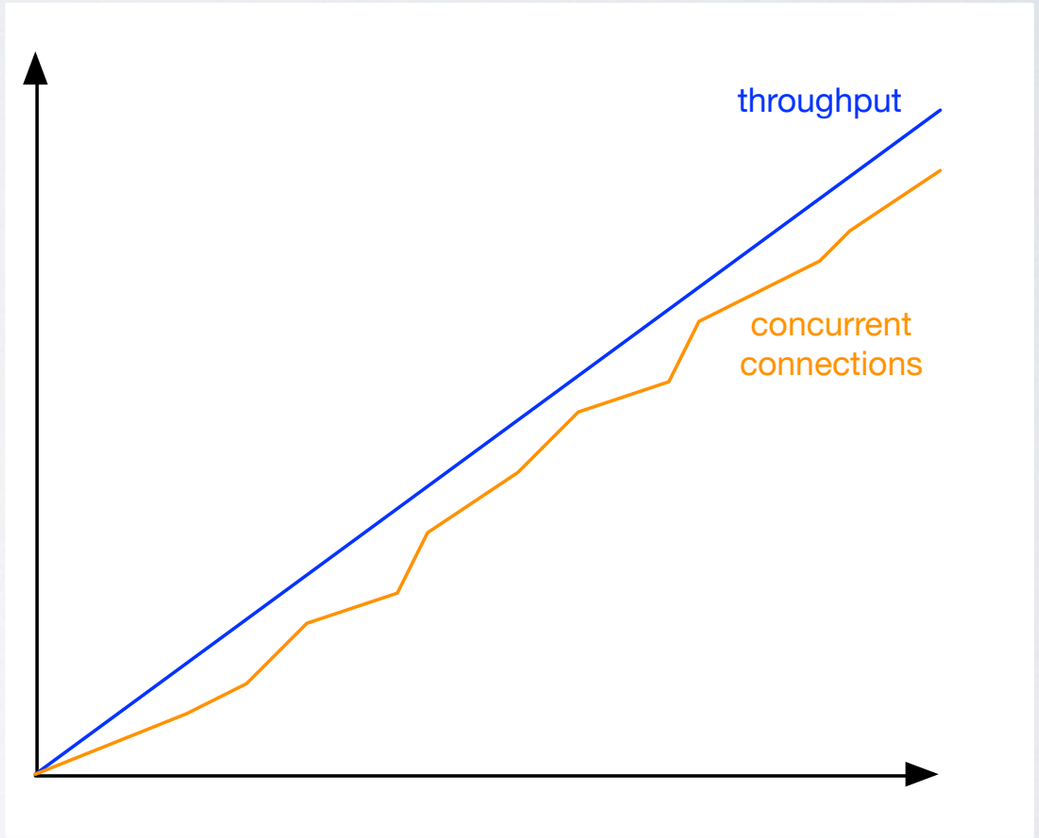
@linkibol

Throughput vs Concurrency

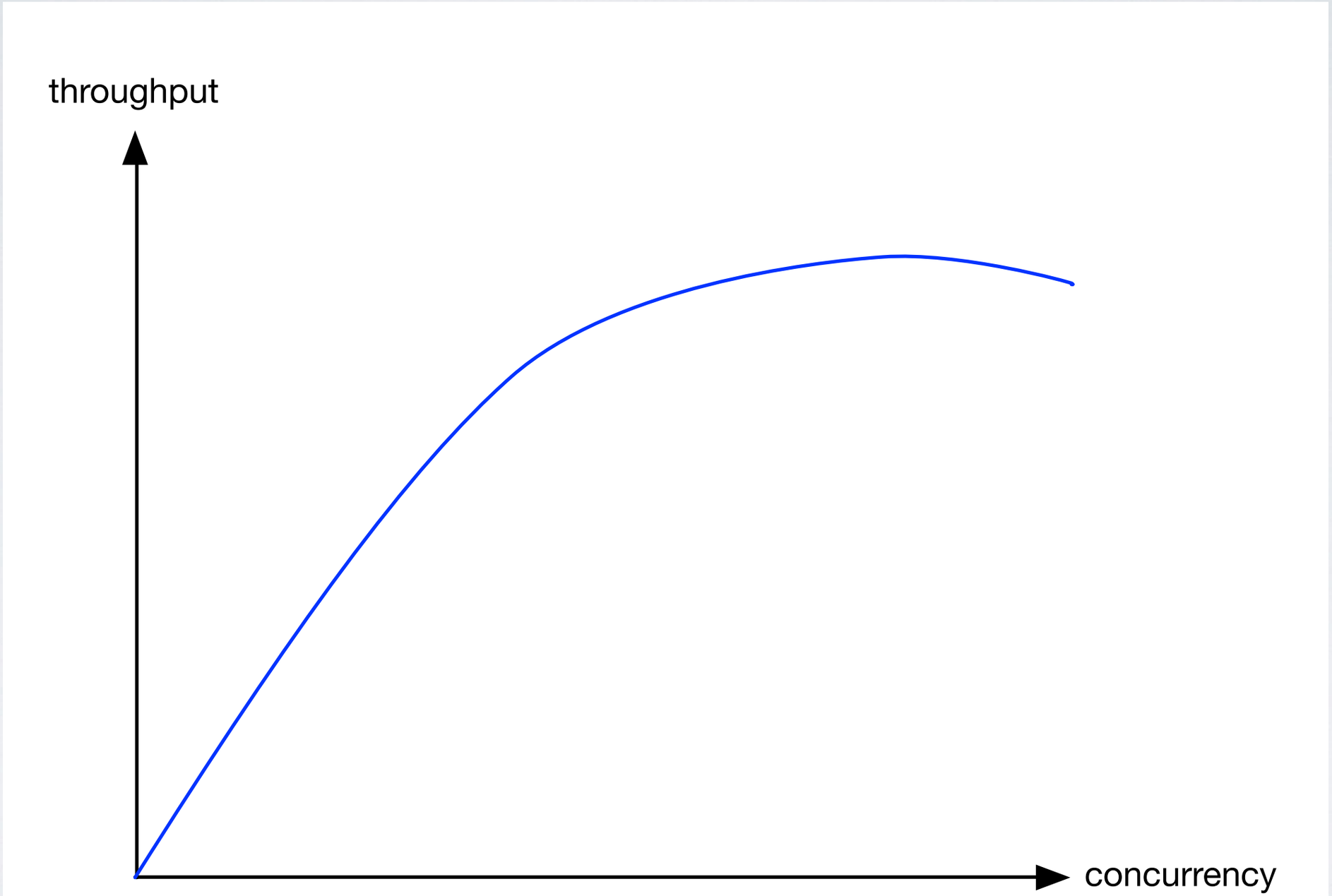


Throughput vs Concurrency

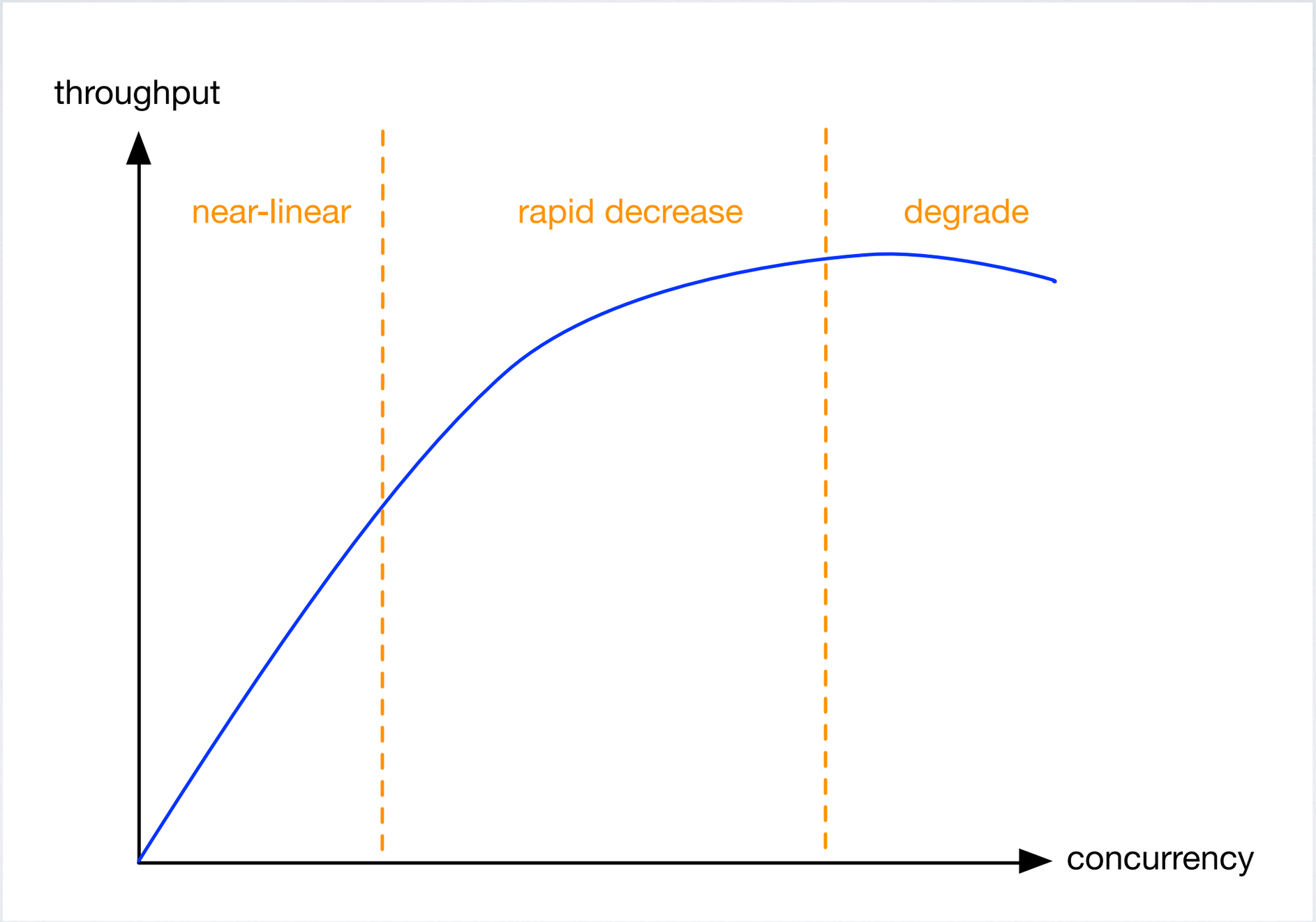




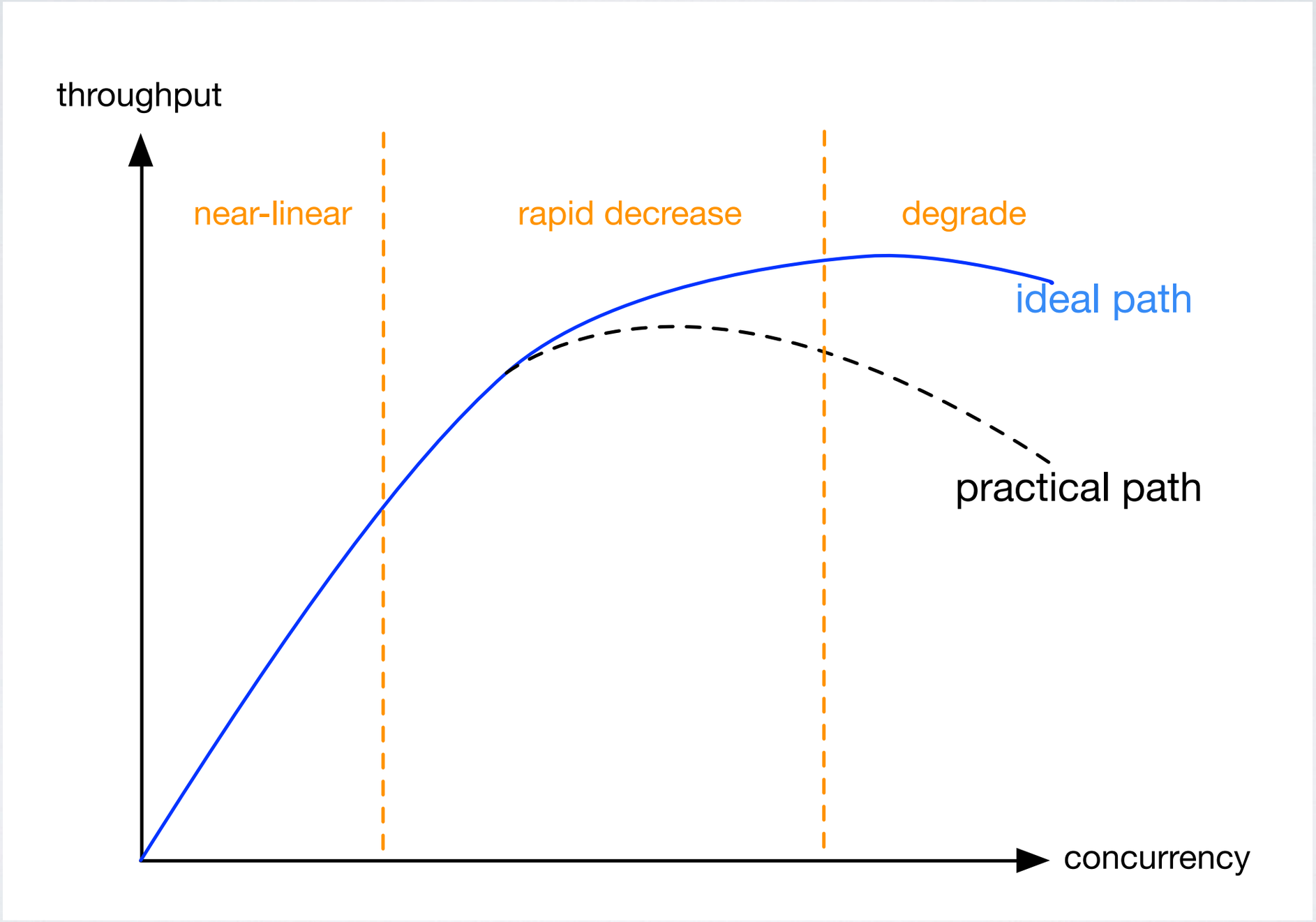
Throughput vs Concurrency



Throughput vs Concurrency



Throughput vs Concurrency



Event Loop Delay

Track the Event Loop

```
1   // ./track.js
2
3   'use strict';
4
5   var config = {
6       INTERVAL: 100,
7       THRESHOLD: 10
8   };
9
10  module.exports = function track(report) {
11      var start = process.hrtime();
12
13      setInterval(function() {
14          var delta = process.hrtime(start),
15              deltaMs = ((delta[0] * 1e9 + delta[1]) / 1e6) - INTERVAL;
16
17              if (deltaMs > config.THRESHOLD) {
18                  report(Math.round(deltaMs));
19              }
20
21              start = process.hrtime();
22      }, config.INTERVAL).unref();
23  };
```

Track the Event Loop

```
1   // usage:
2
3   require('./track')(function(blockedTimeInMs) {
4       log.warn('Event loop is blocked for "' + blockedTimeInMs + '" ms.');
5   });
```

Track the Event Loop

```
'use strict';

var overloaded = false;

require('./track')(function(blockedTimeInMs) {
    log.warn('Event loop is blocked for "' + blockedTimeInMs + '" ms.');

    if (!overloaded && blockedTimeInMs > 100) {
        overloaded = true;
    }
});

function handle(req, res) {
    if (overloaded) {
        res.send(503, 'Server is busy, come back later!');

        (setTimeout(function() {
            overloaded = false;
        }, 30000)).unref();

        return;
    }

    // Other routing stuff…
}
```

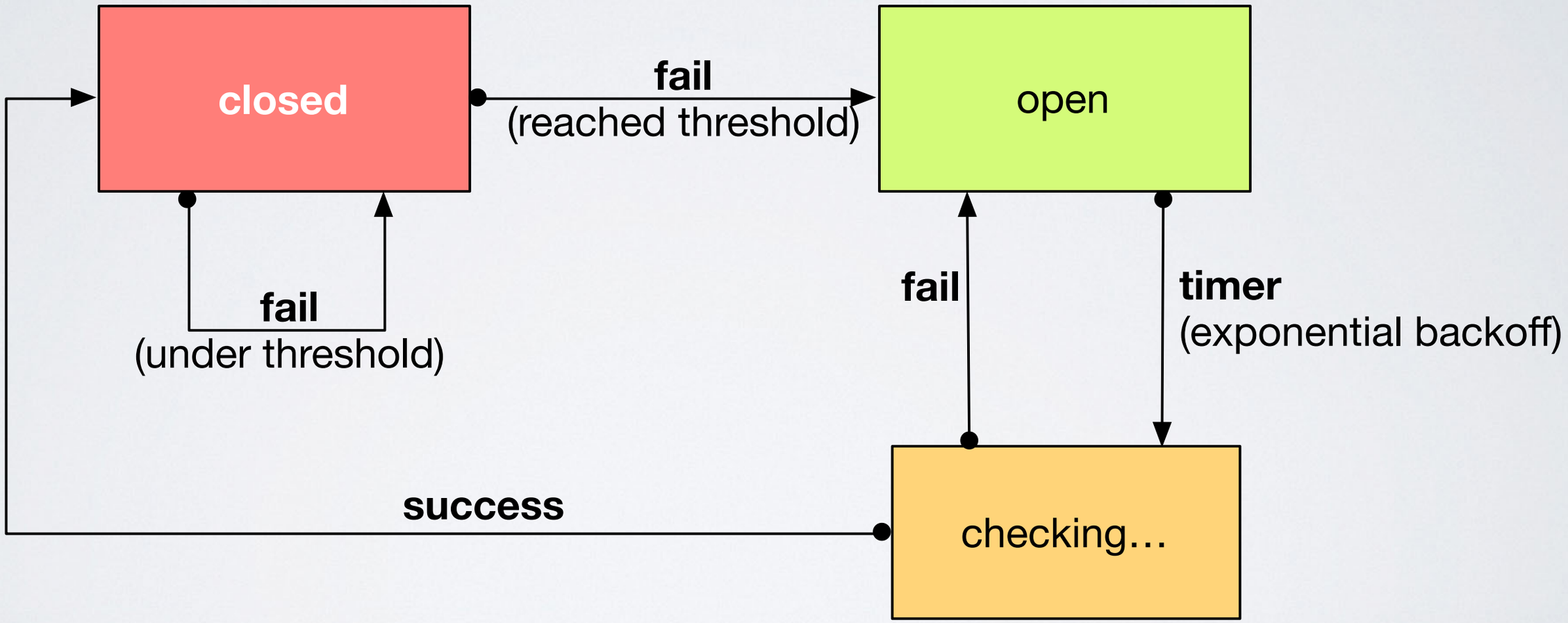
Track the Event Loop

```
1  'use strict';
2
3  var overloaded = false;
4
5  require('./track')(function(blockedTimeInMs) {
6      log.warn('Event loop is blocked for "' + blockedTimeInMs + '" ms.');
7
8      if (!overloaded && blockedTimeInMs > 100) {
9          overloaded = true;
10     }
11 });
12
13 function handle(req, res) {
14     if (overloaded) {
15         res.send(503, 'Server is busy, come back later!');
16
17         (setTimeout(function() {
18             overloaded = false;
19         }, 30000)).unref();
20
21         return;
22     }
23
24     // Other routing stuff…
25 }
```

Track the Event Loop

```
1  'use strict';
2
3  var overloaded = false;
4
5  require('./track')(function(blockedTimeInMs) {
6      log.warn('Event loop is blocked for "' + blockedTimeInMs + '" ms.');
7
8      if (!overloaded && blockedTimeInMs > 100) {
9          overloaded = true;
10     }
11 });
12
13 function handle(req, res) {
14     if (overloaded) {
15         res.send(503, 'Server is busy, come back later!');
16
17         (setTimeout(function() {
18             overloaded = false;
19         }, 30000)).unref();
20
21         return;
22     }
23
24     // Other routing stuff…
25 }
```

Circuit Breaker



Circuit Breaker

- Can be used with any kind of metric.
  - Event-loop tracking is just a specific example.
- Useful when you depend on other APIs that might fail.

Monitoring Tools

volkan.io

@linkibol

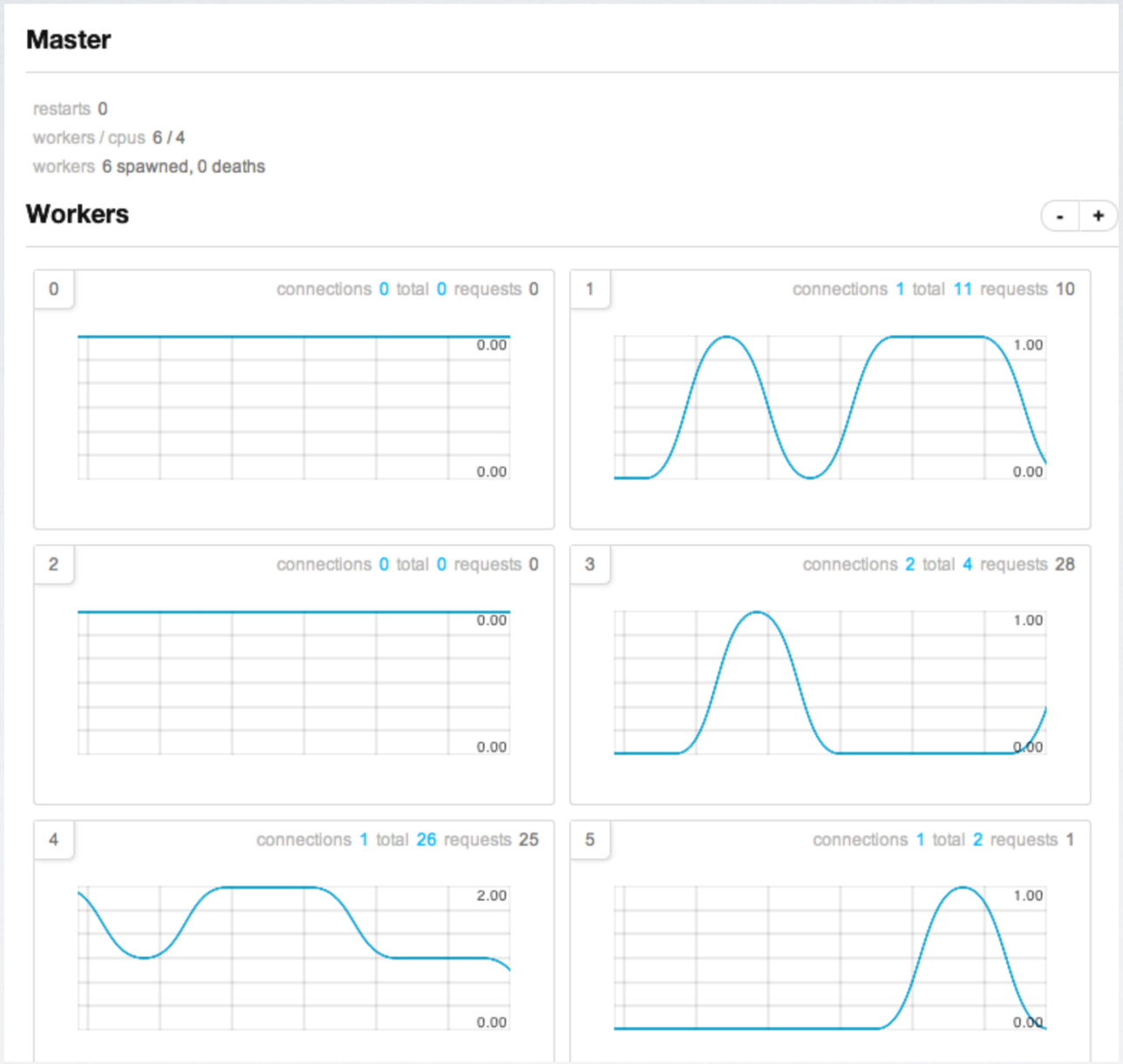
Monitoring as a Service

- **nodetime**
  - https://nodetime.com/
- **newrelic**
  - http://newrelic.com/nodejs
- **strongloop**
  - https://strongloop.com/node-js/performance-monitoring/

Memwatch

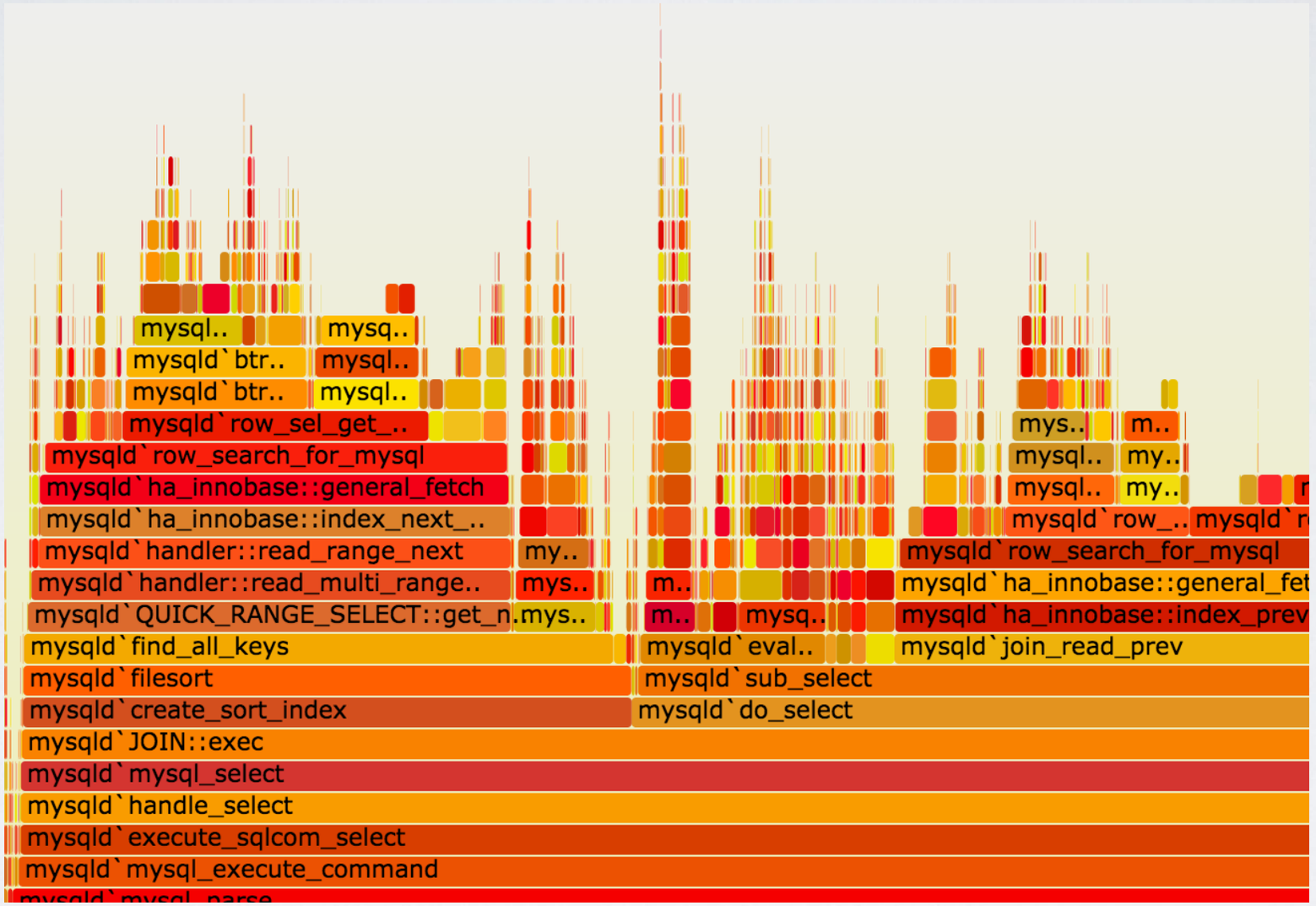
- https://github.com/lloyd/node-memwatch
  - heap stats
  - heap diffing
  - leak detection (subscribe to "*leak*" events)

cluster-live



https://github.com/tj/cluster-live

Flame Graphs



http://www.brendangregg.com/flamegraphs.html

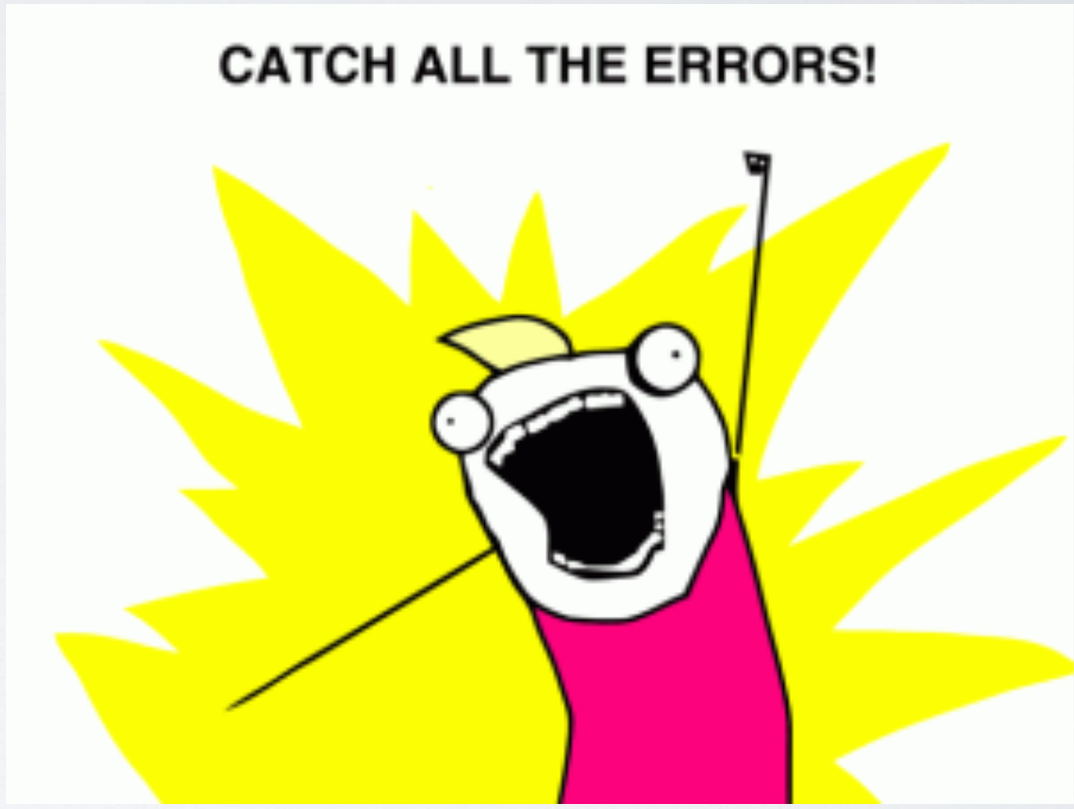
Flame Graphs

- Can Be Created Post-Mortem (after a core dump)
- Can Be Created at Runtime (using **gcore** *)
- You Can Use **dtrace** + **stackvis** to generate them **

* http://man7.org/linux/man-pages/man1/gcore.1.html

** http://blog.nodejs.org/2012/04/25/profiling-node-js/

Error Handling



Error Handling

- **Use an error object**; not a String (*or better, use an* ***error*** *event*)
  - https://github.com/davepacheco/node-verror
- Error handling for Async Code Is Hard
  - Throwing does not make sense (scope and stack trace is lost)
  - Consider using domains (https://nodejs.org/api/domain.html)

Error Handling

- Throwing is for programmer errors.
- Consider using an error event instead of throwing.

```
1   res.on('error', function(er) {
2       // Handle error.
3   });
```

Handle Errors Gracefully

- Know how exceptions and errors propagate in Node.JS.
- Raise ***error*** events to throwing exceptions.
- Restart on uncaught exceptions.
- Utilize **domains**.

https://www.joyent.com/developers/node/design/errors

Using Domains

```
1   var d = domain.create();
2
3   function readFile(fileName, callback) {
4       fs.readFile(fileName, d.bind(function(err, data) {
5           return callback(err, data ? JSON.parse(data) : null);
6       }));
7   }
8
9   d.on('error', function(err) {
10      auditLog(err);
11
12      releaseResources();
13      notifyOthers();
14
15      throw err;
16  });
```

Processes Die

Accept it

Processes Die

Accept it

```
#!/bin/bash

while:
do
        node api.js;
        echo "Server crashed!";
        sleep 1;
done
```

Processes Die

Accept it

```
1  process.on('uncaughtException', function(err) {
2      log.error(err.stack);
3
4      process.exit(1);
5  });
```

Processes Die

Accept it

No system is %100 resilient.

Keep things as simple as possible.

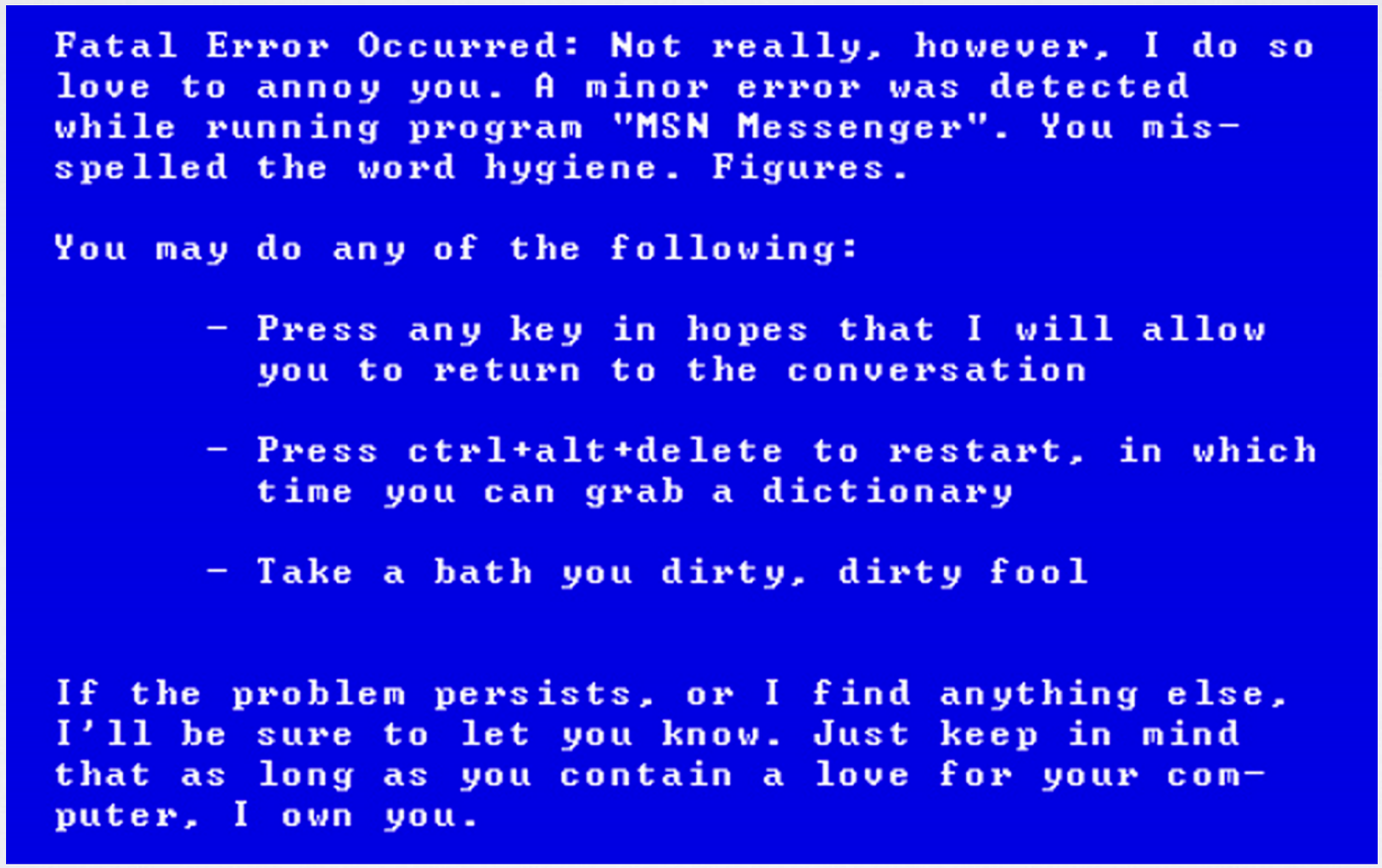
Build something that's **good enough** for your purpose.

Solve for the problems that are **actually** on your plate.

Keep It Running

- **forever** (https://github.com/foreverjs/forever)
- pm2 (https://github.com/Unitech/pm2)
- upstart (http://upstart.ubuntu.com/)
- systemd (https://www.wikiwand.com/en/Systemd)

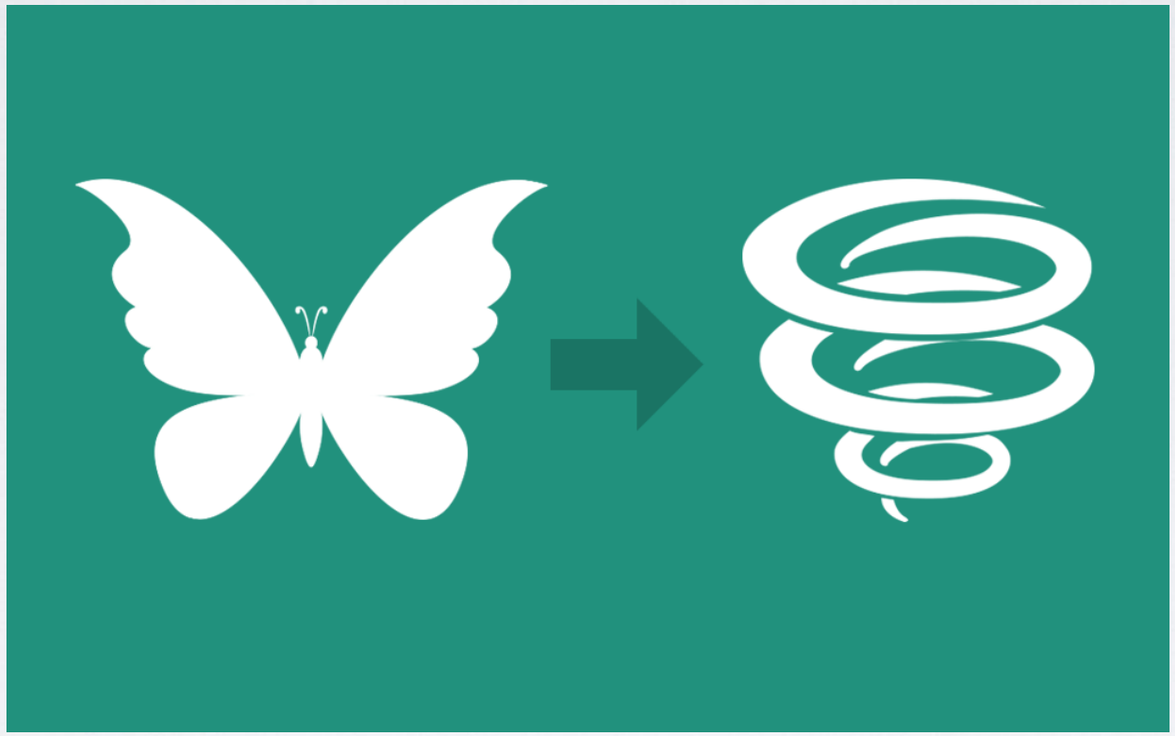
Debugging


Fatal Error Occurred: Not really, however, I do so love to annoy you. A minor error was detected while running program "MSN Messenger". You mis-spelled the word hygiene. Figures.
You may do any of the following:
- Press any key in hopes that I will allow you to return to the conversation
- Press ctrl+alt+delete to restart, in which time you can grab a dictionary
- Take a bath you dirty, dirty fool
If the problem persists, or I find anything else, I'll be sure to let you know. Just keep in mind that as long as you contain a love for your com-puter, I own you.

Debugging

- Live Debugging
- Post-Mortem Debugging

Live Debugging

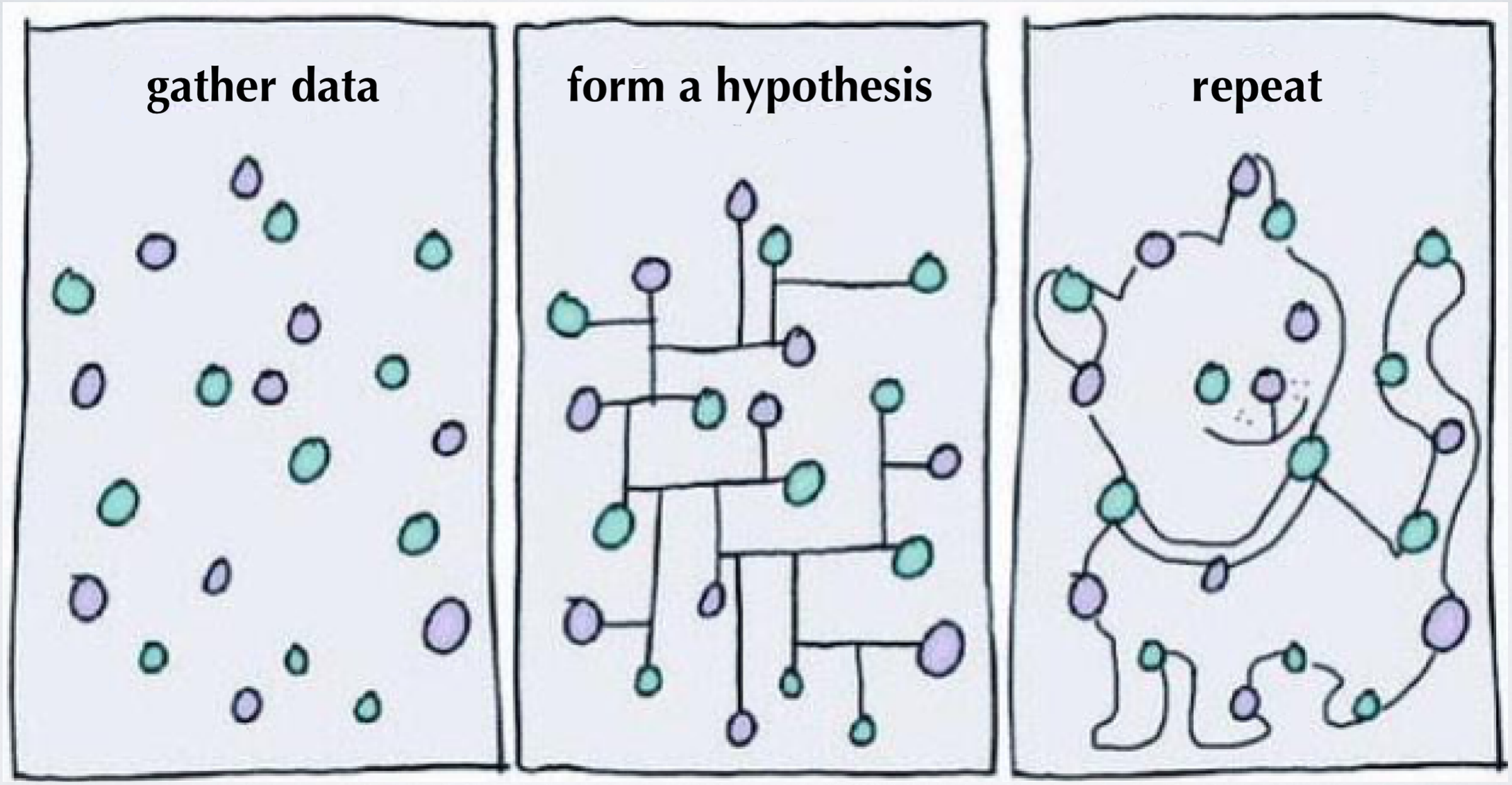


Given the Tornado, Where's the Butterfly?

volkan.io

@linkibol

Post-Mortem Debugging



Node.JS Debugging Myths

- Debugging and Profiling in Node.JS is Hard
- Debugging and Profiling in Node.JS is Immature
- You Cannot Debug or Profile a Live Production Node.JS App

Node.JS Debugging Tips

- Attaching a debugger to prod is not practical.
- Make as much state as possible **observable**.
- You can take a **core dump**, and analyze it later.

Debugging Node.JS

- You Can Expose Internal State via an API and/or a CLI/REPL
  - https://github.com/davepacheco/kang
  - https://nodejs.org/api/repl.html
- Expose Additional Logging Info at Runtime (in systems that support it)
  - bunyan -p (https://github.com/trentm/node-bunyan)

Debugging Node.JS

- Interactive Debugging
  - Using node-debug
    - https://nodejs.org/api/debugger.html
  - Using Node Inspector
    - https://github.com/node-inspector/node-inspector
  - Using Cloud9 IDE
    - https://c9.io/

Debugging Node.JS

- Log Libraries Specialized for Dumping Debug Info
- Caterpillar (https://github.com/bevry/caterpillar)
- Tracer (https://github.com/baryon/tracer)

Logging

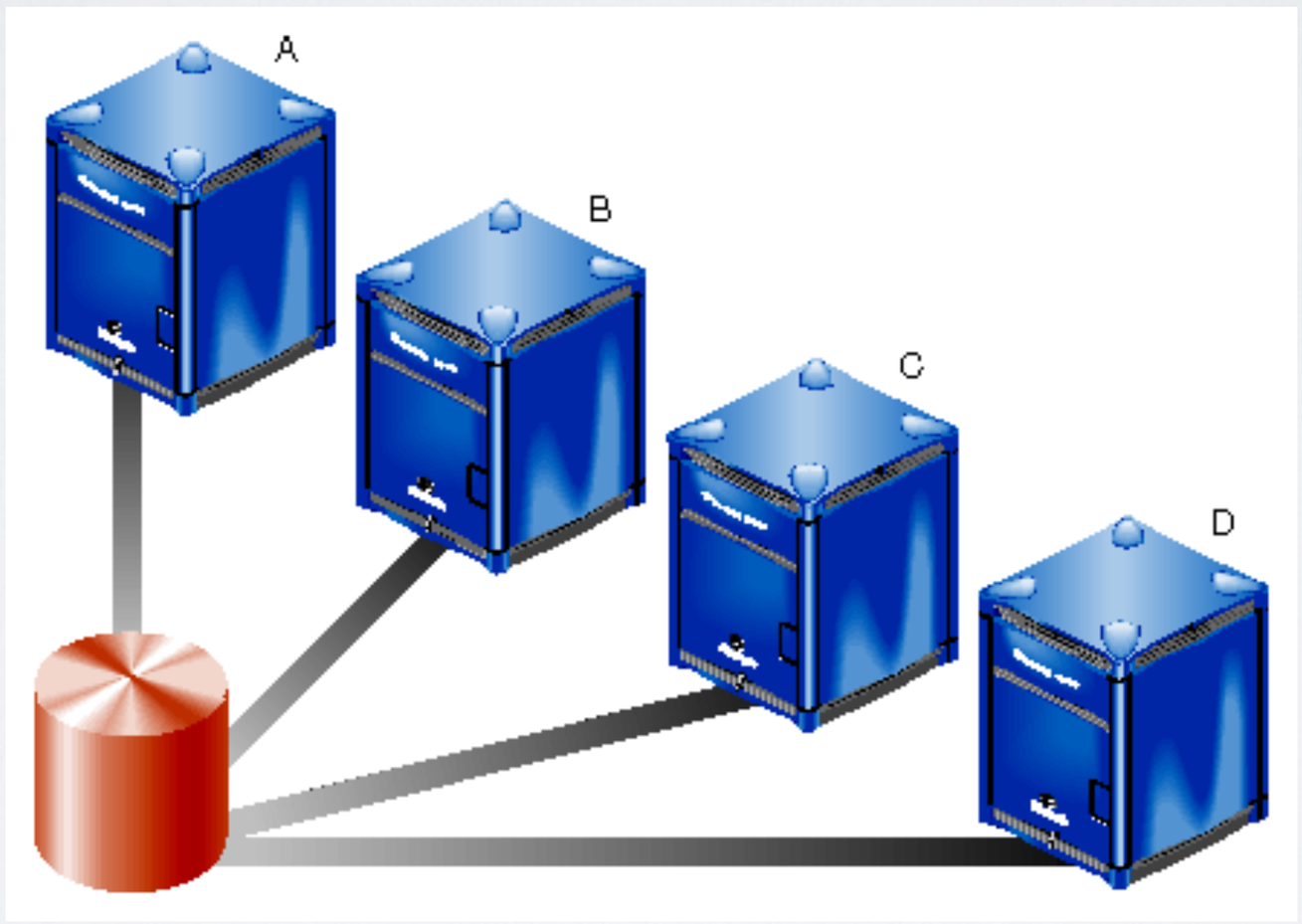
What to Log

- Authentication & Authorization
- Session Management
- Method Entry Points
- Erros and Weird Events
- Specific Events (startup, shutdown, slowdown etc.)
- High-Risk Functionalities (payments, privileges, admins etc)

Centralized Logging

- Send logs to a log aggregation server;
- Using a library that supports it:
  - bunyan (https://github.com/trentm/node-bunyan);
  - winston (https://github.com/winstonjs/winston);
  - custom (using Streams).

Configuration



Configuration

- No Hard-Coded IP Addresses in Config Files
  - Let DNS do What it Does Best
- Update Configuration From a Central Service
  - **saltstack** (http://saltstack.com/)
  - chef (https://www.chef.io/)
  - puppet (https://puppetlabs.com/)

Automation

Have a CI/CD Pipeline

- go (http://www.go.cd/)
- gradle (http://gradle.org/)
- jenkins (https://jenkins-ci.org/)

Code Quality

Code Quality

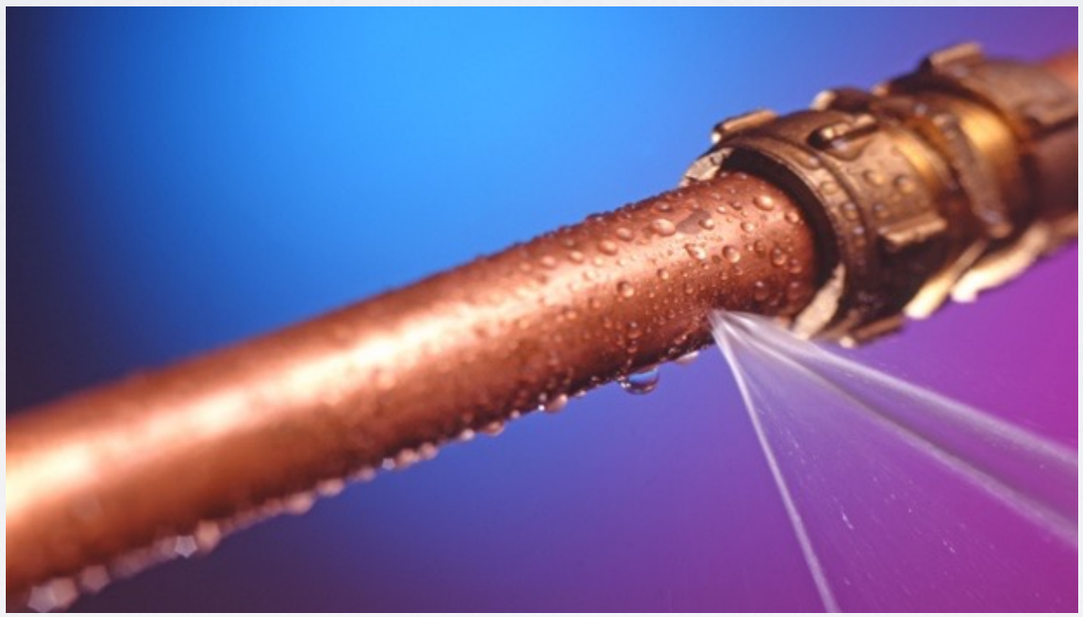
- Group Common Logic Into Reusable Modules
- Modules Should Do One Thing, and Do One Thing Well
- Use Static File Analyzers
  - jshint (https://github.com/gruntjs/grunt-contrib-jshint)
  - jscs (https://github.com/jscs-dev/grunt-jscs)
  - grunt-complexity (https://github.com/vigetlabs/grunt-complexity)

Perf Before Scale

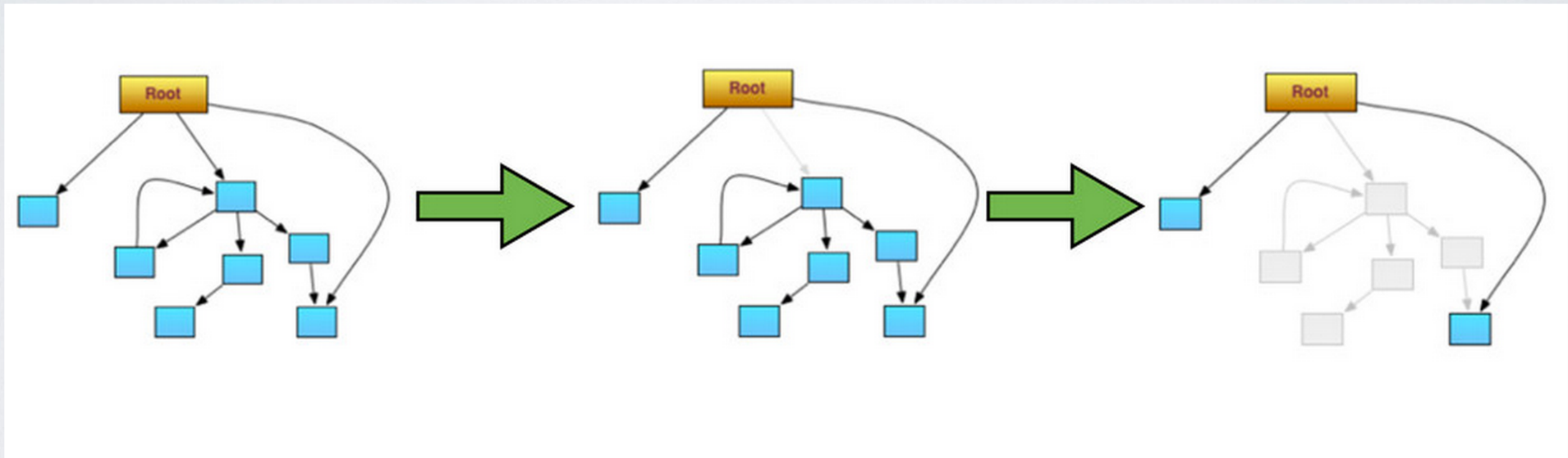
volkan.io

@linkibol

Memory Leaks



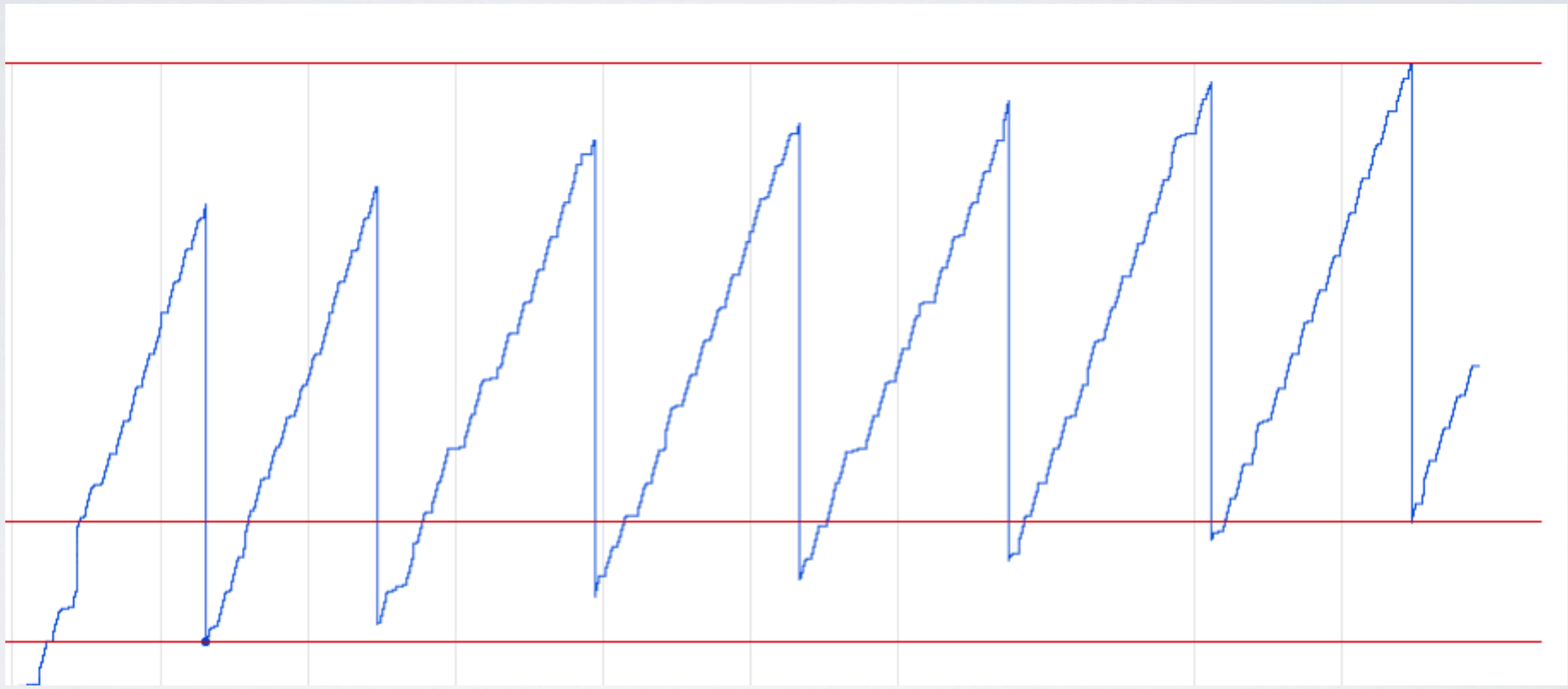
Memory Leaks



volkan.io

@linkibol

Memory Leaks



Perf Before Scale

- Configure Your System for High Performance
- Cache at Every Layer
  - The fastest API response is no response at all.
- Delegate Long-Running(*) Operations

Perf Before Scale

- CPU-Intensive Computations?
- child_process + External Libraries
- Native Extensions

IO Optimization

volkan.io

@linkibol

IO Optimization

- Don't Immediately Write Small Packets
- Reduce the Number of Outgoing Requests
- Use Lesser Abstractions for Maximum Throughput

Know Your Bottlenecks

- 99% of the time **you will be IO-bound**.
    - You'll scale horizontally before hitting 100% CPU.
    - You'll have to try really hard to have a CPU or memory bottleneck.
- Node.JS serves really well as a highly concurrent **networking** app.
- Node.JS is very sensitive to **memory leaks** and **blocking code**.

Torture Your System

- Try Chaos Monkey
  - https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey
- Randomly send `kill -9` to Processes
- Randomly Knock a Server Offline
- Intentionally Run Out of Disk Space
- Take an entire data center down

Going Bare Bones

volkan.io

@linkibol

express.js vs Bare-Bones Node.JS

express.js

```
1  'use strict';
2
3  var express = require('express'),
4      app = express();
5
6  app.get('/', function(req, res) {
7      res.writeHead(200, {'Content-Type': 'text/plain'});
8      res.end('hello world');
9  });
10
11 app.listen(process.env.PORT);
```

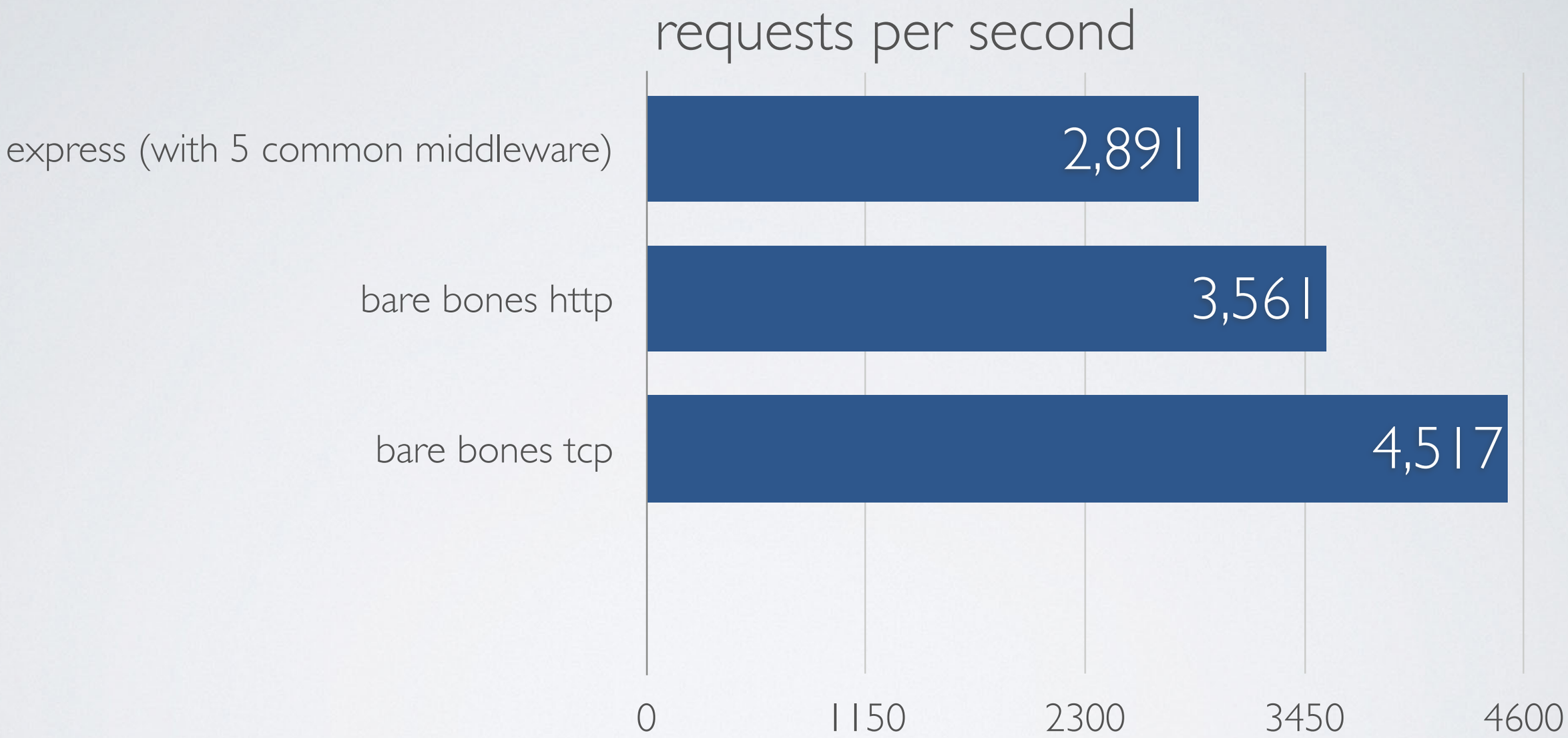
http

```
1  'use strict';
2
3  var http = require('http');
4      server;
5
6  server = http.createServer(function(req, res) {
7      res.writeHead(200, {'Content-Type': 'text/plain'});
8      res.end('Hello World\n');
9  });
10
11 server.listen(process.env.PORT);
```

Diving Deeper

```
1  'use strict';
2
3  var net = require('net'), server;
4
5  function handleData(chunk) {
6      // `this` refers to connection.
7
8      // Parse the raw response.
9  }
10
11 function handleError(error) {
12     // `this` refers to connection.
13 }
14
15 function handleEnd() {
16     // `this` refers to connection.
17
18     // Close socket.
19     // Do cleanup.
20 }
21
22 function onConnection(connection) {
23
24     // Disable Nagle algorithm.
25     connection.setNoDelay();
26
27     // Keep open if expecting subsequent requests.
28     connection.keepAlive(true);
29
30     connection.on('data', handleData);
31     connection.on('error', handleError);
32     connection.on('end', handleEnd);
33 }
34
35 server = net.createServer(onConnection).listen(PORT);
```

Going Bare Bones



ab -c 100 -n 1000

Tested on MacBook Pro, 2.4 GHz Intel Core i5, 16 GB 1600 MHz DDR3, single core, w/o keep-alive

Is It Worth It?

- Consider Going Bare-Bones for Maximum Throughput
- Tradeoff:
  - harder to maintain
  - more complex code
  - error prone
  - lots of edge cases
  - harder to use additional tooling (*i.e. no dtrace support*)

v8 Optimizations

volkan.io

@linkibol

Taking Control of the Garbage Controller

- Warning: You probably would **not** want to do that!

- --expose-gc

- --nouse-idle-notification

Types of Compilers in v8

- Generic Compiler
- Optimizing Compiler
    - Can Be Two or More Orders of Magnitude Faster

Optimize Only

Hot Code Paths

Unless You Have Solid Evidence to Do Otherwise

Tooling

```
node --trace_opt
     --trace_deopt
     --allow-natives-syntax test.js;
```

- `console.log(%HasFastProperties(obj))`
- `console.log(%GetOptimizationStatus(fn))`

https://github.com/Nathanaela/v8-natives

v8 Optimization Killers

- These will be bailed out (likely, forever):
  - Using ***debugger*** anywhere within the function
  - Using ***eval*** anywhere within the function
  - Using **with** anywhere within the function

https://github.com/petkaantonov/bluebird/wiki/Optimization-killers

v8 Optimization Killers

- These will be bailed out (for now):
  - **Generators**
  - Functions that contain a **for-of** statement
  - Functions that contain **try-catch** or **try-finally**
  - **let** assignments
  - **const** assignments
  - functions that contain object literals with **__proto__**, **get**, or **set** declarations.

Infinite Loops
With Unclear Logic

- while(true) { … }
- for(; ;) { … }

Objects

```
Object.prototype.baz = function() {};
```

- Do not define enumerable properties in the prototype chain.
- Use **Object.defineProperty** to create non-enumerable properties.

Objects

```
function willBeDeoptimized() {
    var arr = [1,2,3], key;

    for (key in arr) {
        ;
    }
}

function willBeDeoptimizedToo() {
    var arr = {'1': 'a', '2': 'b'}, key;

    for (key in arr) {
        ;
    }
}

function willBeDeoptimizedAsWell() {
    var arr = {'finally': true, 'delete': false}, key;

    for (key in arr) {
        ;
    }
}

function mayBeDeoptimized() {
    var arr = {'foo': 11, 'baz': 12}, key;

    delete arr.foo;

    arr.boo = 44;

    for (key in arr) {
        ;
    }
}
```

for/each

```
1   'use strict';
2
3   function cannotReferenceKeyFromClosure() {
4       var obj = {},
5           key;
6
7       for (key in obj) {
8           ;
9       }
10
11      return function() {
12          return key;
13      }
14  }
15
16  var key;
17  function cannotUseKeyFromUpperScope() {
18      var obj = {};
19
20      for (key in obj) {
21          ;
22      }
23  }
```

Isolate try/catch

```
1   'use strict';
2
3   function hotCode() {
4       try {
5           if (Math.random() > 0.9) {throw new Error('kaboom!');}
6
7           doOtherStuff();
8
9           return 0;
10      } catch (ex) {
11          log.error(ex);
12
13          return -1;
14      }
15  }
16
17  hotCode();
```

Isolate try/catch

```
1   'use strict';
2
3   var globalError = {value: null};
4
5   function tryCatch(delegate, context, args) {
6       try {
7           return delegate.apply(context, args);
8       } catch (e) {
9           globalError.value = e;
10
11          return globalError;
12      }
13  }
14
15  function mightThrow() {
16      if (Math.random() > 0.9) {throw new Error('kaboom!');}
17  }
18
19  function hotCode() {
20      if (tryCatch(mightThrow, null, [1, 2, 3]) === globalError) {
21          log.error(globalError.value);
22
23          return -1;
24      }
25
26      doOtherStuff();
27
28      return 0;
29  }
30
31  hotCode();
```

Isolate try/catch

```
1   'use strict';
2
3   var globalError = {value: null};
4
5   function tryCatch(delegate, context, args) {
6       try {
7           return delegate.apply(context, args);
8       } catch (e) {
9           globalError.value = e;
10
11          return globalError;
12      }
13  }
14
15  function mightThrow() {
16      if (Math.random() > 0.9) {throw new Error('kaboom!');}
17  }
18
19  function hotCode() {
20      if (tryCatch(mightThrow, null, [1, 2, 3]) === globalError) {
21          log.error(globalError.value);
22
23          return -1;
24      }
25
26      doOtherStuff();
27
28      return 0;
29  }
30
31  hotCode();
```

Isolate try/catch

```
1   'use strict';
2
3   var globalError = {value: null};
4
5   function tryCatch(delegate, context, args) {
6       try {
7           return delegate.apply(context, args);
8       } catch (e) {
9           globalError.value = e;
10
11          return globalError;
12      }
13  }
14
15  function mightThrow() {
16      if (Math.random() > 0.9) {throw new Error('kaboom!');}
17  }
18
19  function hotCode() {
20      if (tryCatch(mightThrow, null, [1, 2, 3]) === globalError) {
21          log.error(globalError.value);
22
23          return -1;
24      }
25
26      doOtherStuff();
27
28      return 0;
29  }
30
31  hotCode();
```

arguments Object

```
1   'use strict';
2
3   // `arguments` object must not be passed, or leaked anywhere.
4
5   function badSlice() {
6       return Array.prototype.slice.call(arguments);
7   }
8
9   function goodSlice() {
10      var ar = new Array(arguments.length),
11          i, len;
12
13      for (i = 0, len = arguments.length; i++) {
14          ar[i] = arguments[i];
15      }
16  }
```

Iteration Tips

- Rule of thumb: Avoid for/in loops in hot code paths.
- Always use **Object.keys** to iterate an object.
- **Rethink your architecture**
  if you need to iterate the parent prototype's keys.

Inherited keys

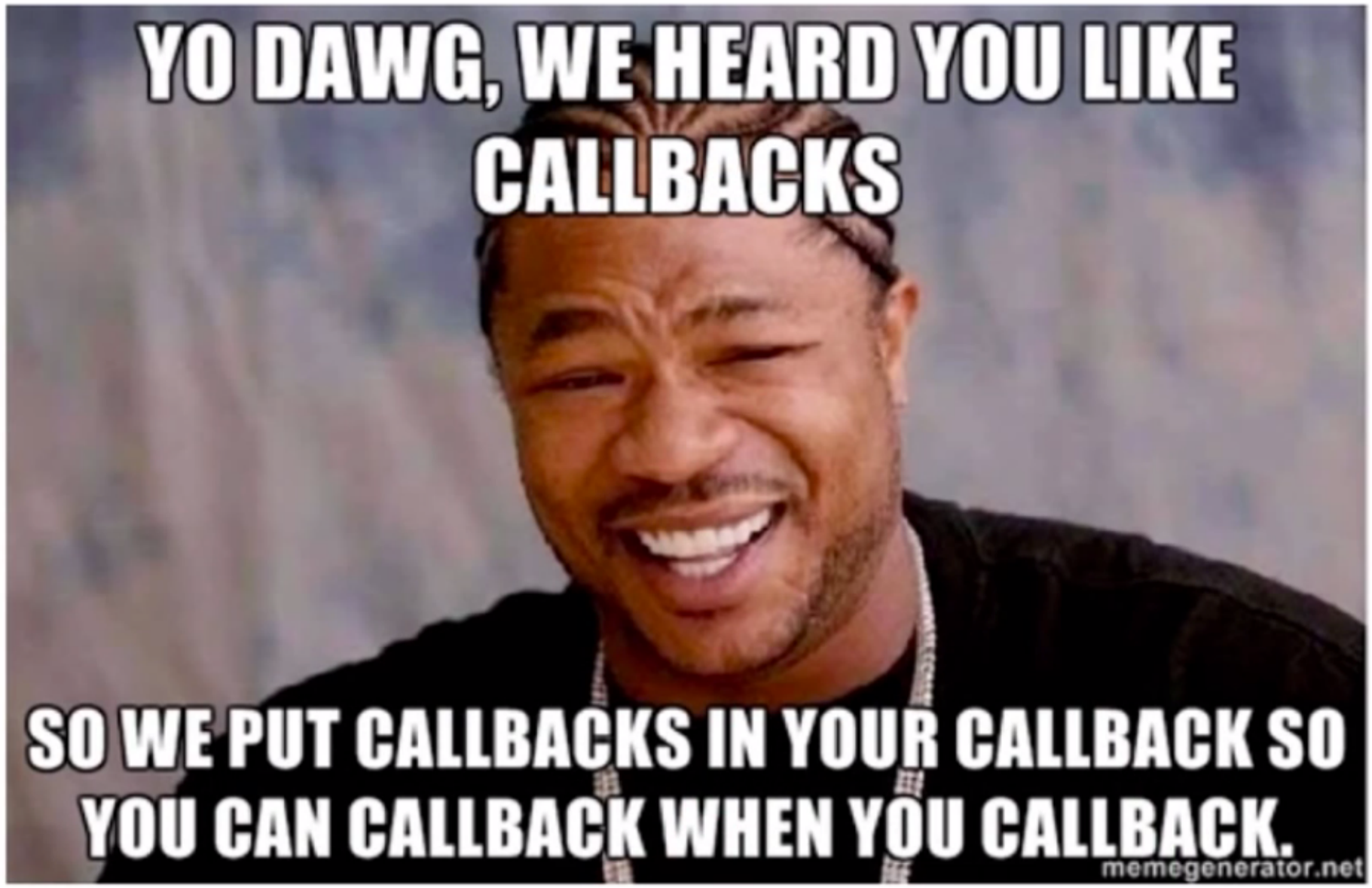
```
1  function getInheritedKeys(obj) {
2      var key, ar = [];
3
4      for (key in obj) {
5          ar.push(key);
6      }
7
8      return ar;
9  }
```

Promises

volkan.io

@linkibol

Callback Hell Is Real



Promises

Promises are not as slow as they once were.

https://github.com/petkaantonov/bluebird

Promises

- Use promises — ***ahem*** BlueBird — liberally.
- Consider using continuation passing style
  (i.e. callbacks) for very hot code paths.
  - Callbacks are **always** faster, and more memory-efficient.
- Do not optimize prematurely; **measure things first**!

Security

Do Not Run Node.JS As Root

```
useradd -mrU web;
mkdir /opt/web-app;
chown web /opt/web-app;
cd /opt/web-app;
su web;
node app.js;
firewall-cmd --permanent --zone=public --add-port=3000/tcp
```

Security

- https://www.owasp.org/index.php/OWASP_Node_js_Goat_Project
- http://nodesecurity.io/
- https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Common Threats

- XSS / CSRF
- Input Validation Attack
- DoS / ReDoS
- Request Size

* not different from any other web app.

Your API at Scale

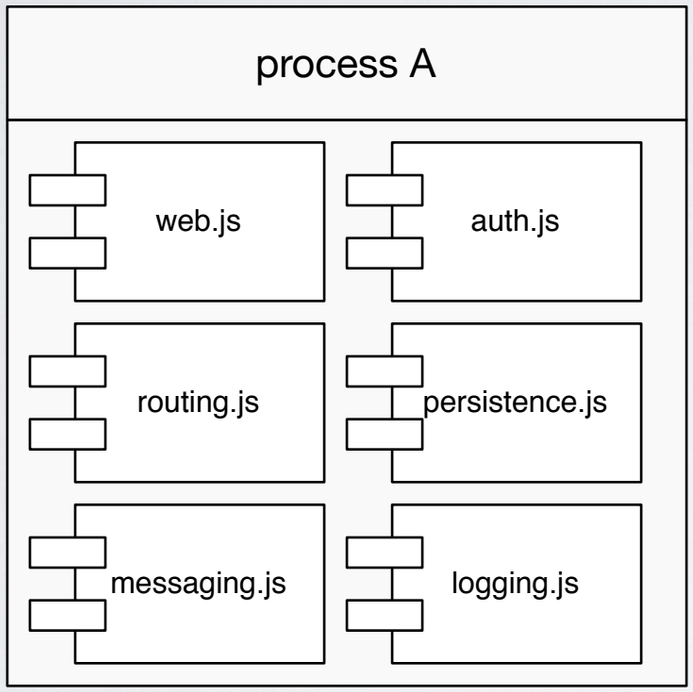
Goals

- **Minimize** Client Response Time
- **Maximize** Resource Efficiency on the Server

Hint: Leave 50% of the memory unused
(for taking core dumps)

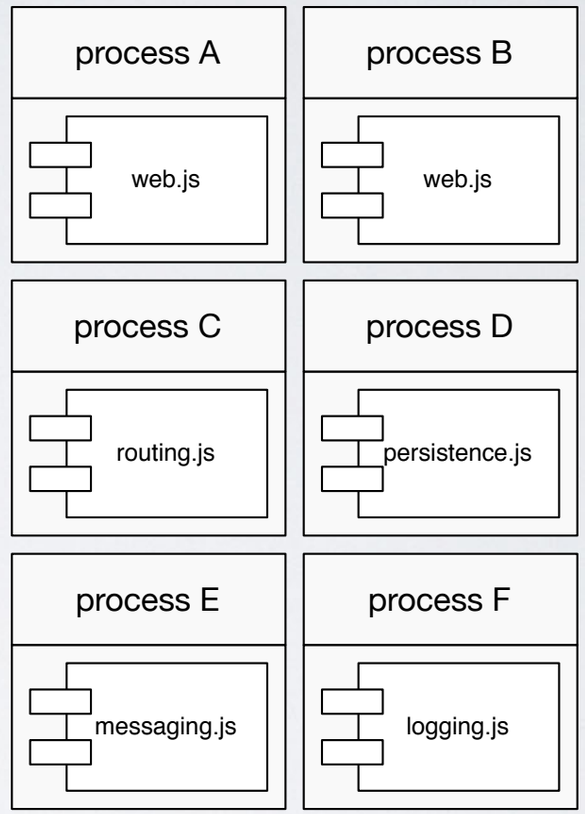
Microservices to the rescue

Microservices



Monolith

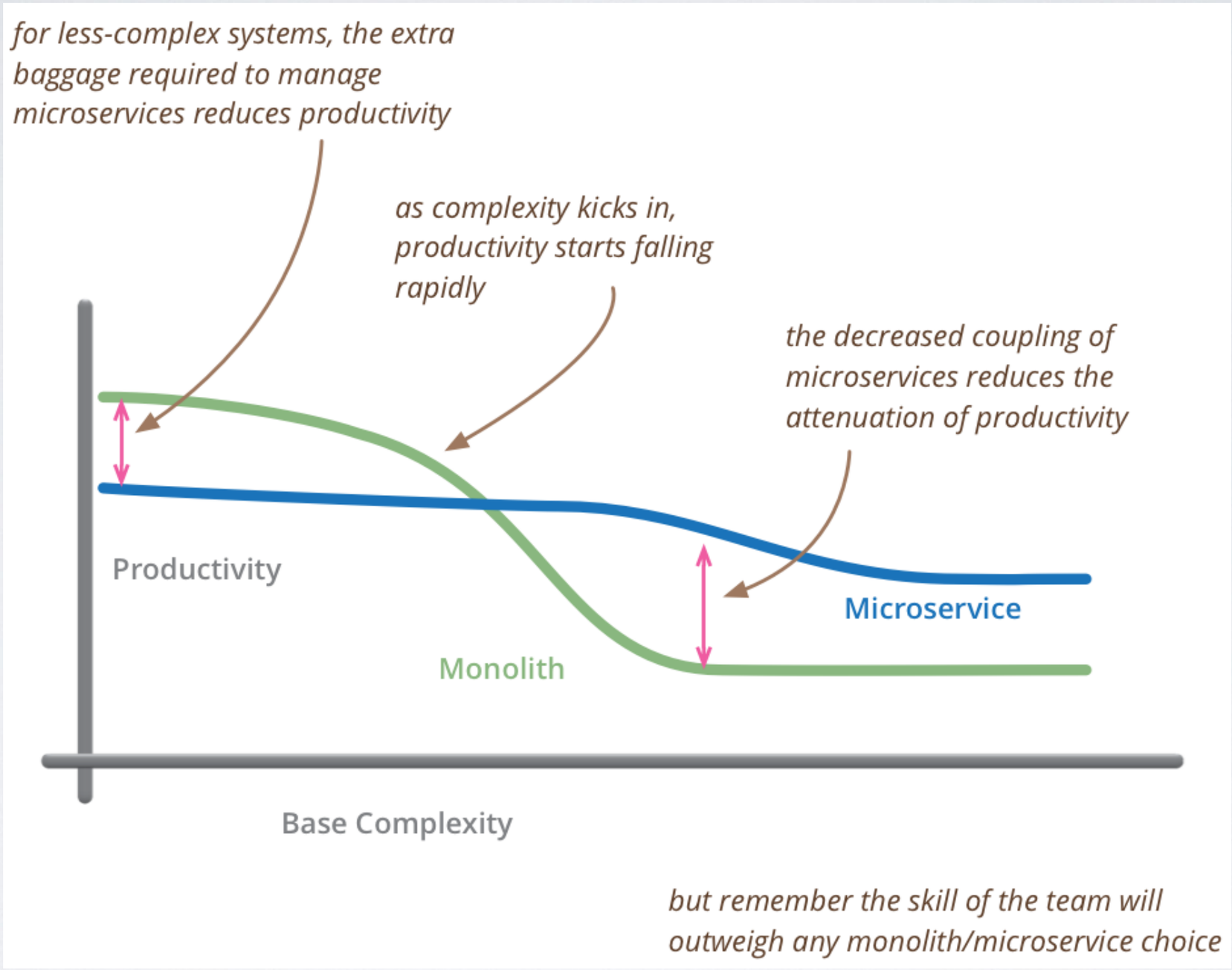
multiple modules
single process



Microservice

multiple modules
multiple processes

Think Twice



http://martinfowler.com/bliki/MicroservicePremium.html

Favor Microservices Over Monoliths

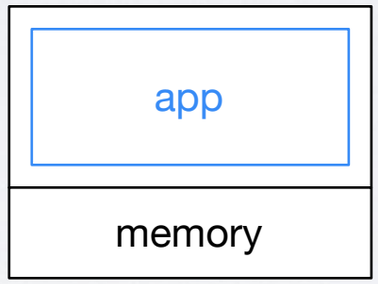
- **Prefer Composition over Inheritance**
- **Publish Modules** Instead of Inlining Functionality
  - Use an Internal npm

Scaling Your API

volkan.io

@linkibol

Scaling Your API

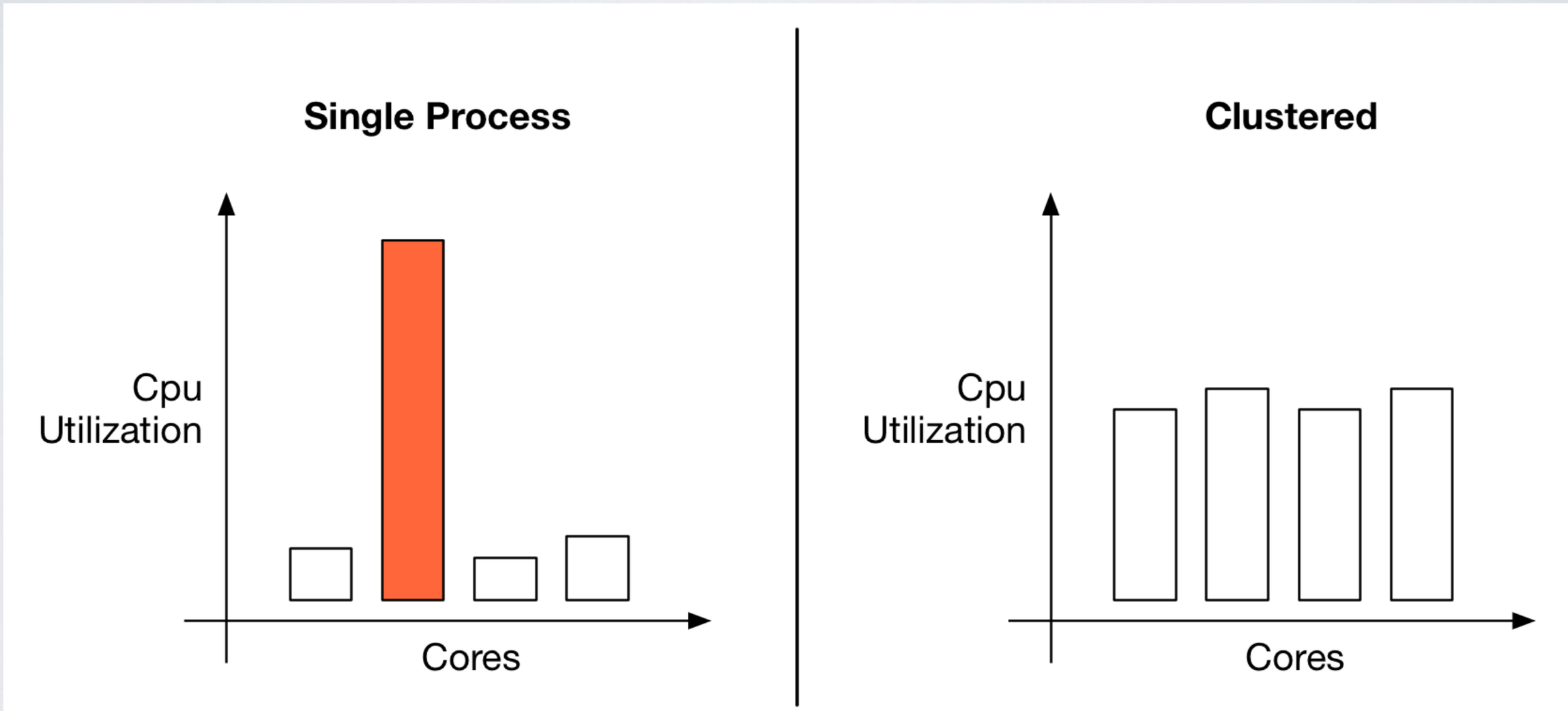


Cluster

volkan.io

@linkibol

Cluster


Single Process
Cpu Utilization
Cores
Clustered
Cpu Utilization
Cores

Cluster

```
1   'use strict';
2
3   var cluster = require('cluster'),
4       http = require('http'),
5       PORT = +process.env.PORT || 1337,
6       server;
7
8   if (cluster.isMaster) {
9        cluster.fork();
10       cluster.fork();
11
12       cluster.on('disconnect', function(/*worker*/) {
13            log.error('disconnect!');
14
15            cluster.fork();
16       });
17  } else {
18       // The worker (i.e. the place we put our bugs)
19       // Normally this should live in a separate module.
20
21       var server = http.createServer(function(req, res) {
22          doAsyncStuff(req, res);
23
24          res.end('Hello world.');
25       });
26
27       server.listen(PORT);
28  }
```

Cluster

```
1   'use strict';
2
3   var cluster = require('cluster'),
4       http = require('http'),
5       PORT = +process.env.PORT || 1337,
6       server;
7
8   if (cluster.isMaster) {
9       cluster.fork();
10      cluster.fork();
11
12      cluster.on('disconnect', function(/*worker*/) {
13          log.error('disconnect!');
14
15          cluster.fork();
16      });
17  } else {
18      // The worker (i.e. the place we put our bugs)
19      // Normally this should live in a separate module.
20
21      var server = http.createServer(function(req, res) {
22         doAsyncStuff(req, res);
23
24         res.end('Hello world.');
25      });
26
27      server.listen(PORT);
28  }
```

Cluster

```
1  'use strict';
2
3  var cluster = require('cluster'),
4      http = require('http'),
5      PORT = +process.env.PORT || 1337,
6      server;
7
8  if (cluster.isMaster) {
9      cluster.fork();
10     cluster.fork();
11
12     cluster.on('disconnect', function(/*worker*/) {
13         log.error('disconnect!');
14
15         cluster.fork();
16     });
17 } else {
18     // The worker (i.e. the place we put our bugs)
19     // Normally this should live in a separate module.
20
21     var server = http.createServer(function(req, res) {
22        doAsyncStuff(req, res);
23
24        res.end('Hello world.');
25     });
26
27     server.listen(PORT);
28 }
```

Cluster

+

Domains

Cluster + Domains

```
10  if (cluster.isMaster) {
11      /* ... */
12  } else {
13       var server = http.createServer(function(req, res) {
14          var d = domain.create();
15          d.on('error', function(er) {
16             log.error('error', er.stack);
17             try {
18                  (setTimeout(function() {
19                       // Exit and generate a core file.
20                       process.abort();
21                  }, KILL_TIMEOUT)).unref();
22
23                  // Stop taking new requests.
24                  server.close();
25                  // Let the master know we're dead.
26                  cluster.worker.disconnect();
27
28                  res.statusCode = 500;
29                  res.end('Oops, I did it again!\n');
30             } catch (exception) {
31                  log.error('Error sending 500!', exception.stack);
32             }
33          });
34
35          d.add(req);
36          d.add(res);
37
38          d.run(function() {
39              doAsyncStuff(req, res);
40
41              res.end('Hello world.');
42          });
43      });
44
45      server.listen(PORT);
46  }
```

Cluster + Domains

```
10  if (cluster.isMaster) {
11      /* ... */
12  } else {
13       var server = http.createServer(function(req, res) {
14          var d = domain.create();
15          d.on('error', function(er) {
16             log.error('error', er.stack);
17             try {
18                  (setTimeout(function() {
19                       // Exit and generate a core file.
20                       process.abort();
21                  }, KILL_TIMEOUT)).unref();
22
23                  // Stop taking new requests.
24                  server.close();
25                  // Let the master know we're dead.
26                  cluster.worker.disconnect();
27
28                  res.statusCode = 500;
29                  res.end('Oops, I did it again!\n');
30             } catch (exception) {
31                  log.error('Error sending 500!', exception.stack);
32             }
33          });
34
35          d.add(req);
36          d.add(res);
37
38          d.run(function() {
39              doAsyncStuff(req, res);
40
41              res.end('Hello world.');
42          });
43      });
44
45      server.listen(PORT);
46  }
```

Cluster + Domains

```
10  if (cluster.isMaster) {
11      /* ... */
12  } else {
13       var server = http.createServer(function(req, res) {
14          var d = domain.create();
15          d.on('error', function(er) {
16             log.error('error', er.stack);
17             try {
18                  (setTimeout(function() {
19                       // Exit and generate a core file.
20                       process.abort();
21                  }, KILL_TIMEOUT)).unref();
22
23                  // Stop taking new requests.
24                  server.close();
25                  // Let the master know we're dead.
26                  cluster.worker.disconnect();
27
28                  res.statusCode = 500;
29                  res.end('Oops, I did it again!\n');
30             } catch (exception) {
31                  log.error('Error sending 500!', exception.stack);
32             }
33          });
34
35          d.add(req);
36          d.add(res);
37
38          d.run(function() {
39              doAsyncStuff(req, res);
40
41              res.end('Hello world.');
42          });
43      });
44
45      server.listen(PORT);
46  }
```

Cluster + Domains

```
10  if (cluster.isMaster) {
11      /* ... */
12  } else {
13       var server = http.createServer(function(req, res) {
14          var d = domain.create();
15          d.on('error', function(er) {
16             log.error('error', er.stack);
17             try {
18                  (setTimeout(function() {
19                       // Exit and generate a core file.
20                       process.abort();
21                  }, KILL_TIMEOUT)).unref();
22
23                  // Stop taking new requests.
24                  server.close();
25                  // Let the master know we're dead.
26                  cluster.worker.disconnect();
27
28                  res.statusCode = 500;
29                  res.end('Oops, I did it again!\n');
30              } catch (exception) {
31                  log.error('Error sending 500!', exception.stack);
32              }
33          });
34
35          d.add(req);
36          d.add(res);
37
38          d.run(function() {
39              doAsyncStuff(req, res);
40
41              res.end('Hello world.');
42          });
43      });
44
45      server.listen(PORT);
46  }
```

Cluster + Domains

```
10  if (cluster.isMaster) {
11      /* ... */
12  } else {
13       var server = http.createServer(function(req, res) {
14          var d = domain.create();
15          d.on('error', function(er) {
16             log.error('error', er.stack);
17             try {
18                  (setTimeout(function() {
19                       // Exit and generate a core file.
20                       process.abort();
21                  }, KILL_TIMEOUT)).unref();
22
23                  // Stop taking new requests.
24                  server.close();
25                  // Let the master know we're dead.
26                  cluster.worker.disconnect();
27
28                  res.statusCode = 500;
29                  res.end('Oops, I did it again!\n');
30             } catch (exception) {
31                  log.error('Error sending 500!', exception.stack);
32             }
33          });
34
35          d.add(req);
36          d.add(res);
37
38          d.run(function() {
39              doAsyncStuff(req, res);
40
41              res.end('Hello world.');
42          });
43      });
44
45      server.listen(PORT);
46  }
```

Cluster
+
Zero Downtime
Rolling Deployments

volkan.io

@linkibol

Zero Downtime

kill --USR2 <pid>

Zero Downtime

```
var GRACE_PERIOD = process.env.NODE_ENV === 'production' ? 600000 : 30000;

...

if (cluster.isMaster) {
    cluster.on('death', function(worker) {
        cluster.fork();
    });
} else {
    ...
}

...

process.on('SIGUSR2', function() {
    respawn();
});

function respawn() {
    server.close();

    (setTimeout(function() {
        process.exit(0);
    }, GRACE_PERIOD).unref();
}

server.listen(3000);
```

see also: https://github.com/doxout/recluster

Zero Downtime

```
2  var GRACE_PERIOD = process.env.NODE_ENV === 'production' ? 600000 : 30000;
3
4  ...
5
6  if (cluster.isMaster) {
7      cluster.on('death', function(worker) {
8          cluster.fork();
9      });
10 } else {
11     ...
12 }
13
14 ...
15
16 process.on('SIGUSR2', function() {
17     respawn();
18 });
19
20 function respawn() {
21     server.close();
22
23     (setTimeout(function() {
24         process.exit(0);
25     }, GRACE_PERIOD).unref();
26 }
27
28 server.listen(3000);
```

see also: https://github.com/doxout/recluster

Zero Downtime

```
2  var GRACE_PERIOD = process.env.NODE_ENV === 'production' ? 600000 : 30000;
3
4  ...
5
6  if (cluster.isMaster) {
7      cluster.on('death', function(worker) {
8          cluster.fork();
9      });
10 } else {
11     ...
12 }
13
14 ...
15
16 process.on('SIGUSR2', function() {
17     respawn();
18 });
19
20 function respawn() {
21     server.close();
22
23     (setTimeout(function() {
24         process.exit(0);
25     }, GRACE_PERIOD).unref();
26 }
27
28 server.listen(3000);
```

see also: https://github.com/doxout/recluster

Build Redundancy Everywhere

volkan.io

@linkibol

Build Redundancy Everywhere

```
1  fs.watchFile('package.json', function(curr, prev) {
2      logger.info('Package.json has changed, reloading cluster…');
3
4      reloadCluster();
5  });
6
7  process.on('SIGUSR2', function() {
8      logger.info('Got SIGUSR2, reloading cluster…');
9
10     reloadCluster();
11 });
```

Build Redundancy Everywhere

```
setInterval(function() {
    var request = http.get('http://localhost:' + port, function(res) {
        request.setTimeout(0);

        if ([200,302].indexOf(res.statusCode) === -1) {
            log.error('[heartbeat] : FAIL with code ' + res.statusCode);

            reloadCluster();

            return;
        }

        log.info(' [heartbeat]:  OK [' + res.statusCode + ']');
    }).on('error',function(err) {
        log.error('[heartbeat] : FAIL with error ' + err.message);

        reloadCluster();
    });

    request.setTimeout(MAX_WAIT_TIMEOUT, function() {
        log.error('[heartbeat] : FAIL with timeout');

        reloadCluster();
    });
}, HEARTBEAT_INTERVAL);
```

Build Redundancy Everywhere

```
1  setInterval(function() {
2      var request = http.get('http://localhost:' + port, function(res) {
3          request.setTimeout(0);
4
5          if ([200,302].indexOf(res.statusCode) === -1) {
6              log.error('[heartbeat] : FAIL with code ' + res.statusCode);
7
8              reloadCluster();
9
10             return;
11         }
12
13         log.info(' [heartbeat]:  OK [' + res.statusCode + ']');
14     }).on('error',function(err) {
15         log.error('[heartbeat] : FAIL with error ' + err.message);
16
17         reloadCluster();
18     });
19
20     request.setTimeout(MAX_WAIT_TIMEOUT, function() {
21         log.error('[heartbeat] : FAIL with timeout');
22
23         reloadCluster();
24     });
25 }, HEARTBEAT_INTERVAL);
```

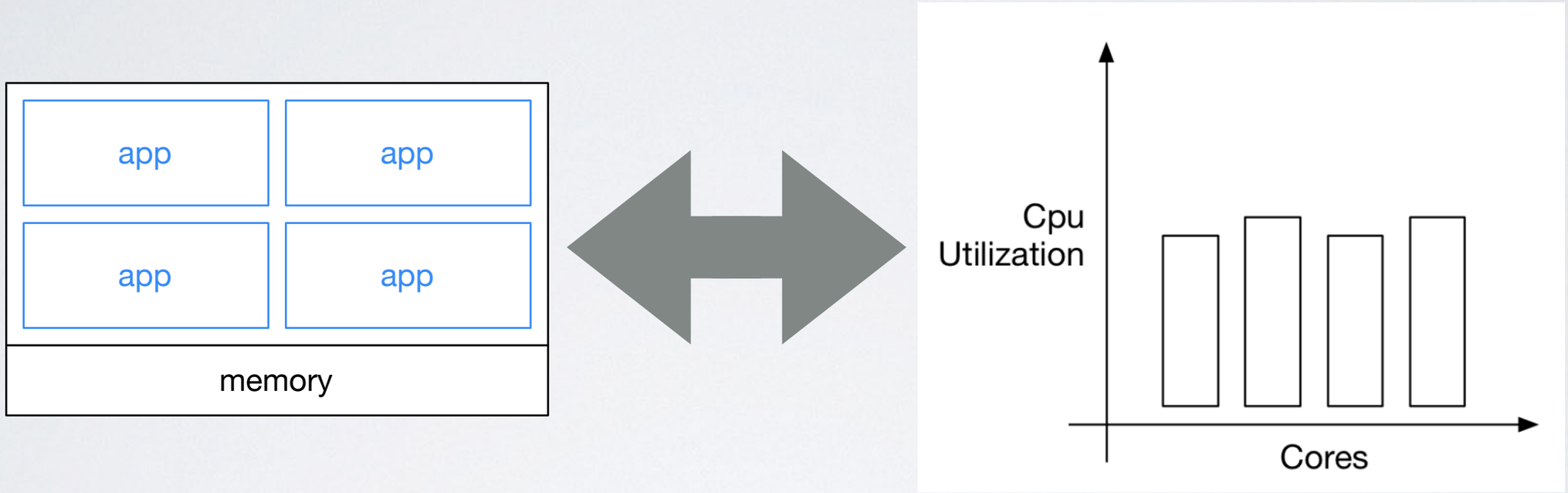
Build Redundancy Everywhere

```
1  setInterval(function() {
2      var request = http.get('http://localhost:' + port, function(res) {
3          request.setTimeout(0);
4
5          if ([200,302].indexOf(res.statusCode) === -1) {
6              log.error('[heartbeat] : FAIL with code ' + res.statusCode);
7
8              reloadCluster();
9
10             return;
11         }
12
13         log.info(' [heartbeat]:  OK [' + res.statusCode + ']');
14     }).on('error',function(err) {
15         log.error('[heartbeat] : FAIL with error ' + err.message);
16
17         reloadCluster();
18     });
19
20     request.setTimeout(MAX_WAIT_TIMEOUT, function() {
21         log.error('[heartbeat] : FAIL with timeout');
22
23         reloadCluster();
24     });
25 }, HEARTBEAT_INTERVAL);
```

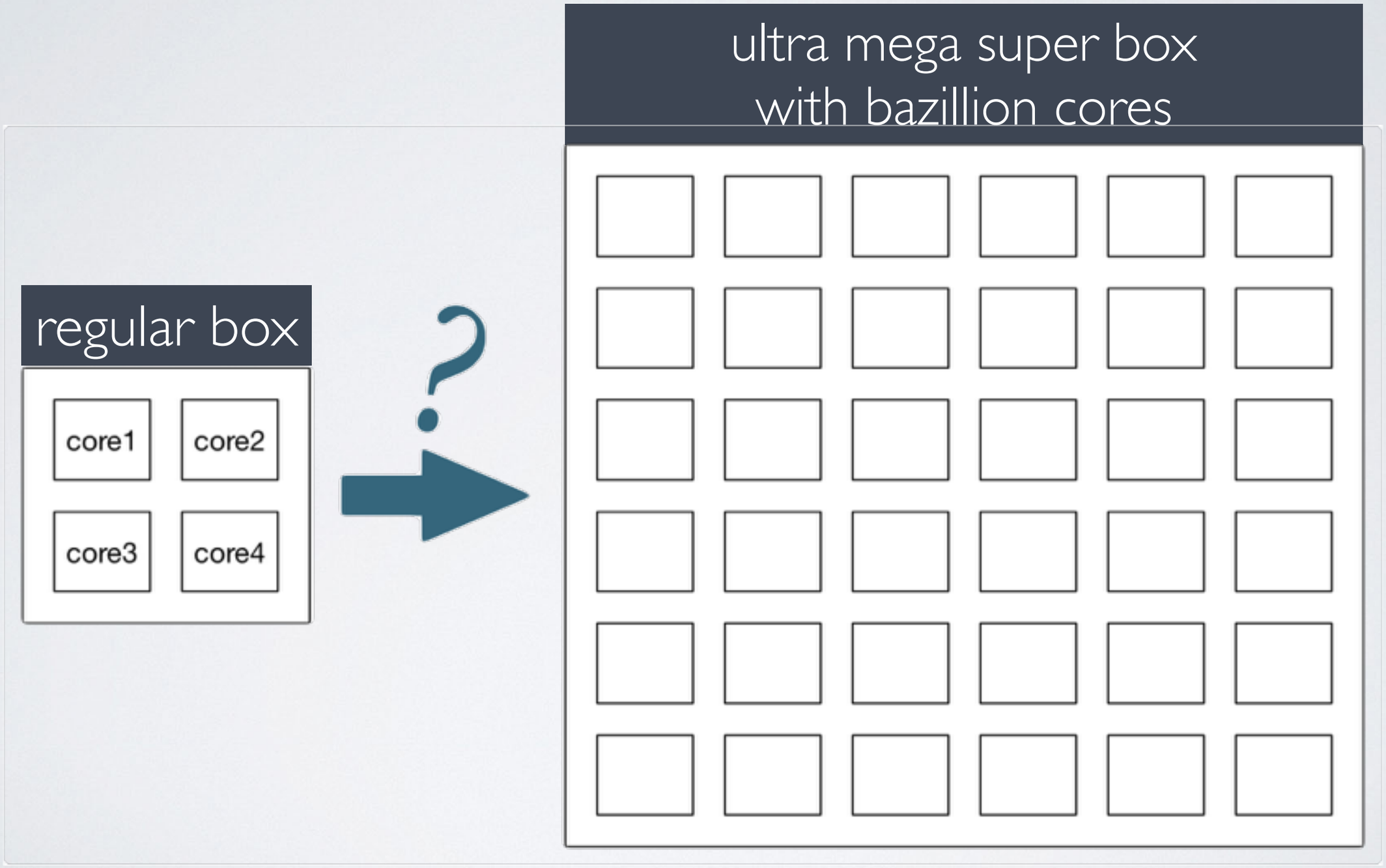
Build Redundancy Everywhere

```
1  setInterval(function() {
2      var request = http.get('http://localhost:' + port, function(res) {
3          request.setTimeout(0);
4
5          if ([200,302].indexOf(res.statusCode) === -1) {
6              log.error('[heartbeat] : FAIL with code ' + res.statusCode);
7
8              reloadCluster();
9
10             return;
11         }
12
13         log.info(' [heartbeat]:  OK [' + res.statusCode + ']');
14     }).on('error',function(err) {
15         log.error('[heartbeat] : FAIL with error ' + err.message);
16
17         reloadCluster();
18     });
19
20     request.setTimeout(MAX_WAIT_TIMEOUT, function() {
21         log.error('[heartbeat] : FAIL with timeout');
22
23         reloadCluster();
24     });
25 }, HEARTBEAT_INTERVAL);
```

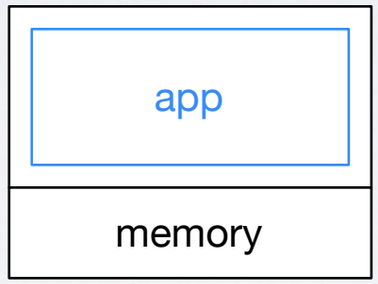
Cluster



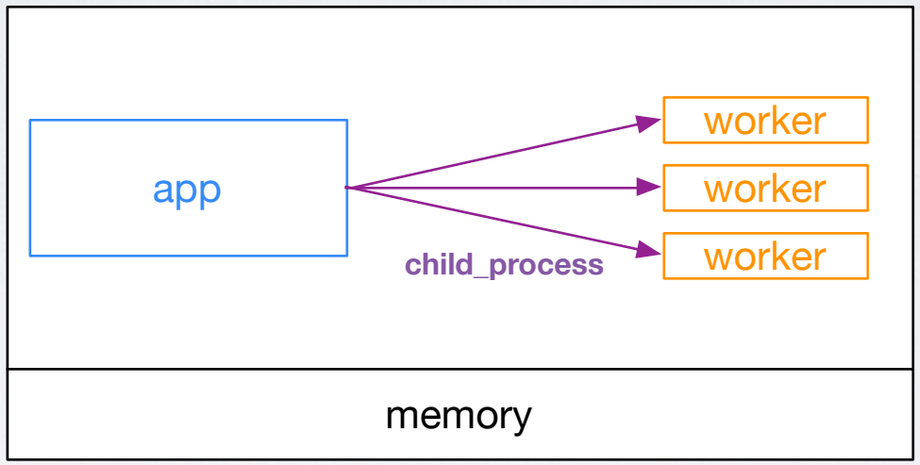
IS BIGGER ALWAYS BETTER?



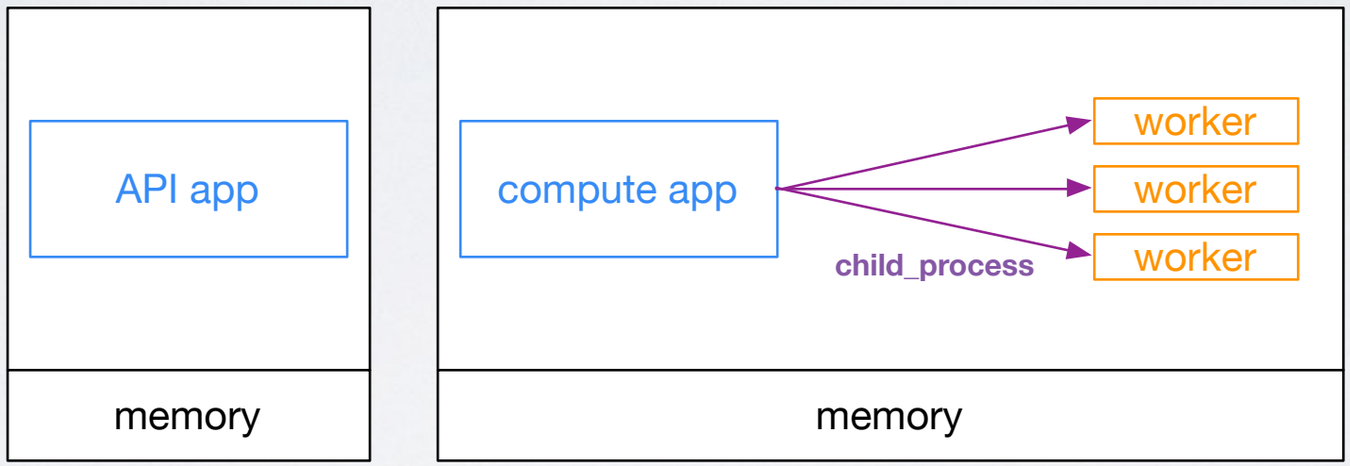
Let's Step Back



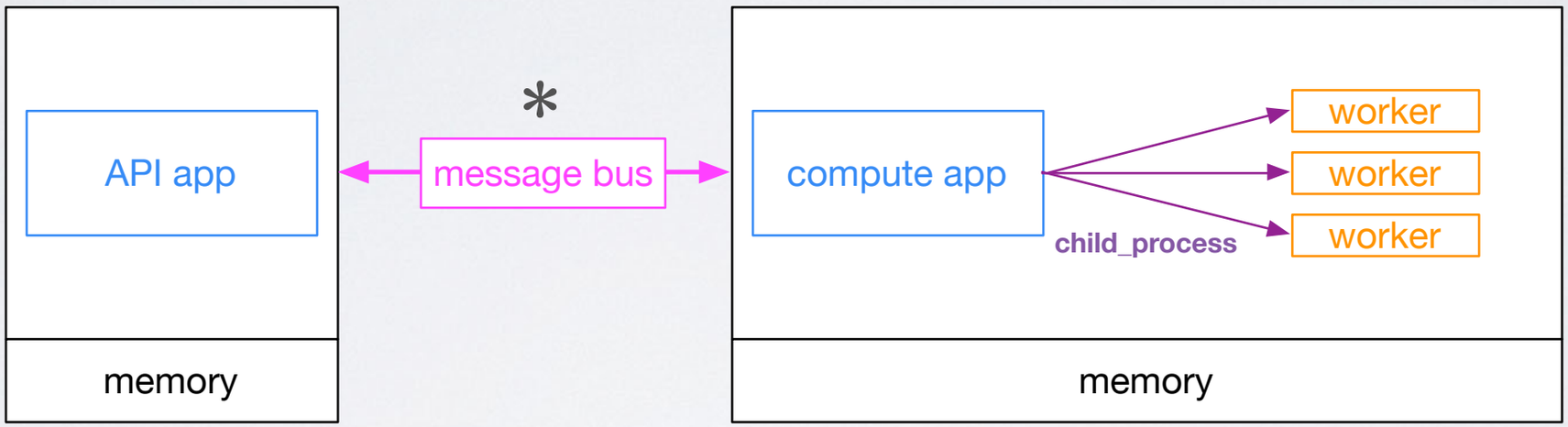
Delegate Heavy Computation to Workers



Split Logical Components into Distinct Services



IPC With a Message Bus

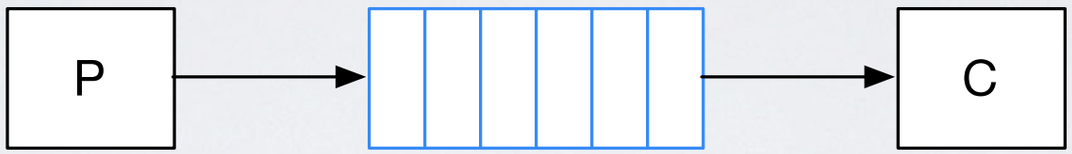


* rabbitmq, zeromq, resque etc.

see also http://queues.io/

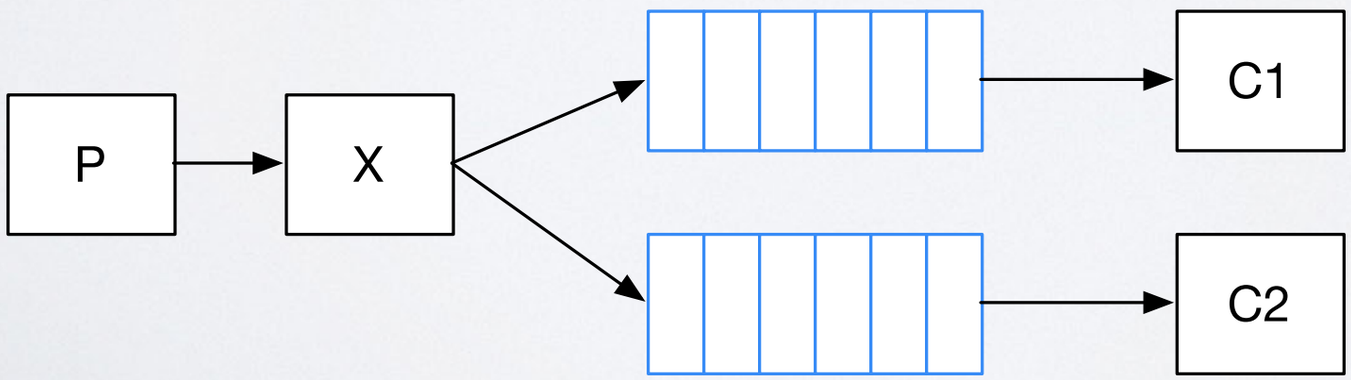
Message Bus Topologies

Send/Listen

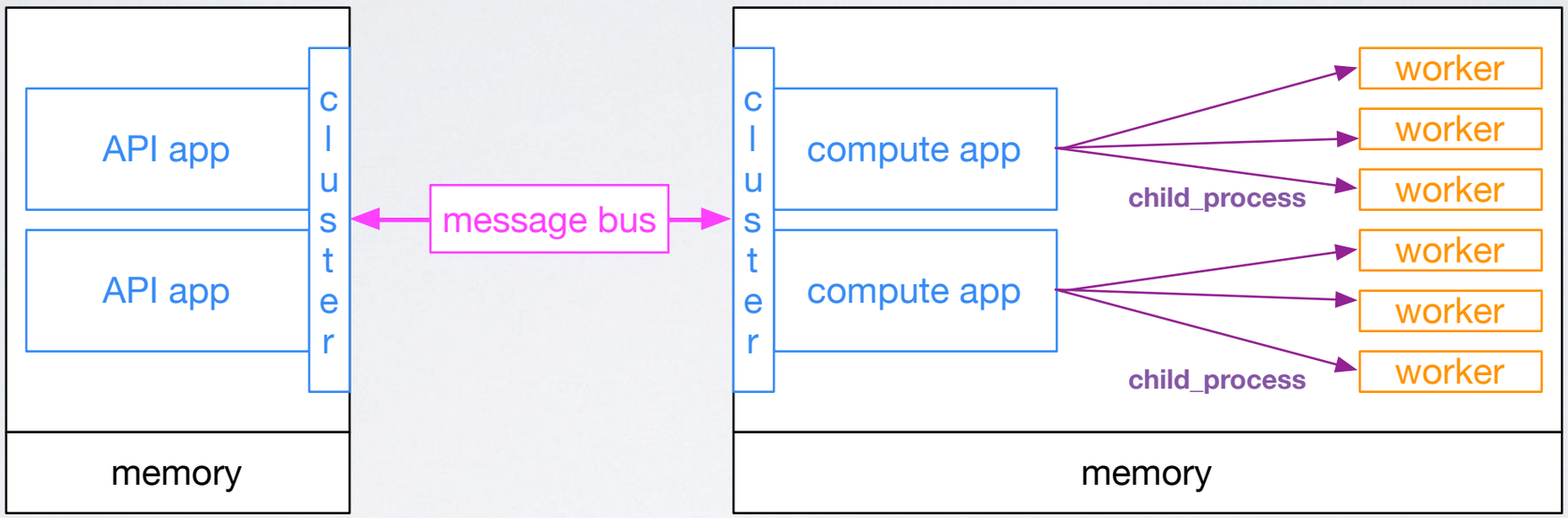


Worker Queue

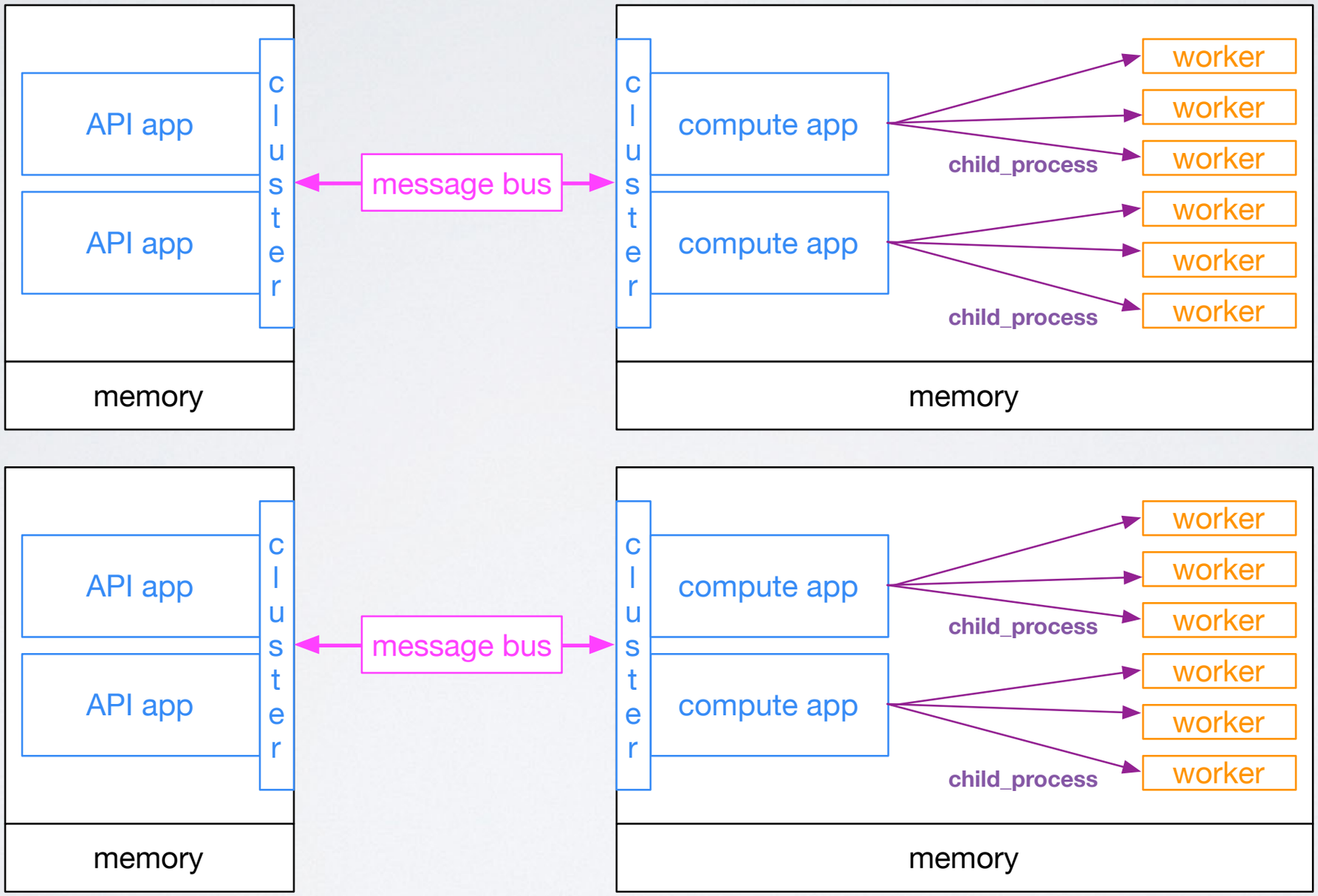
PubSub



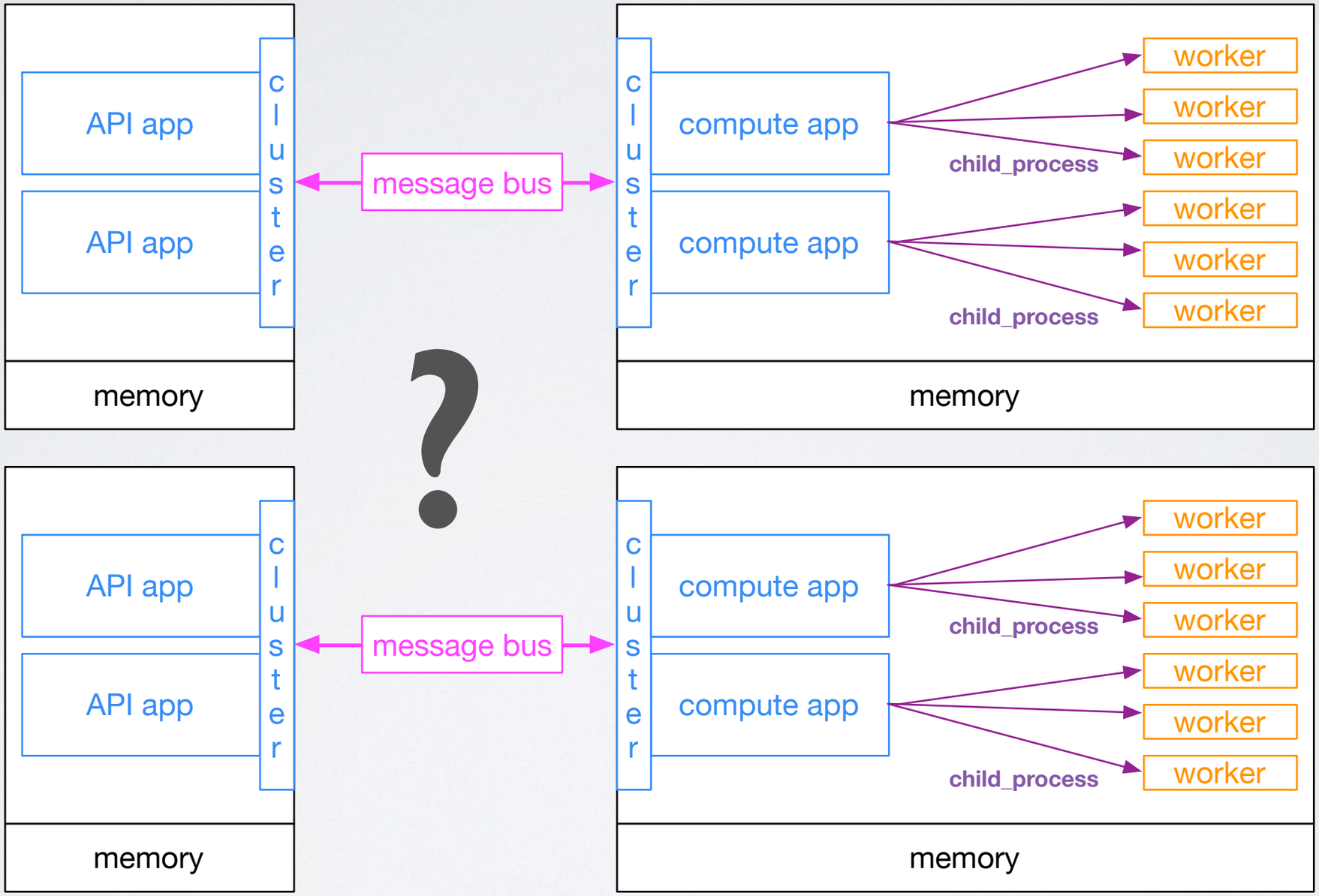
Cluster Processes



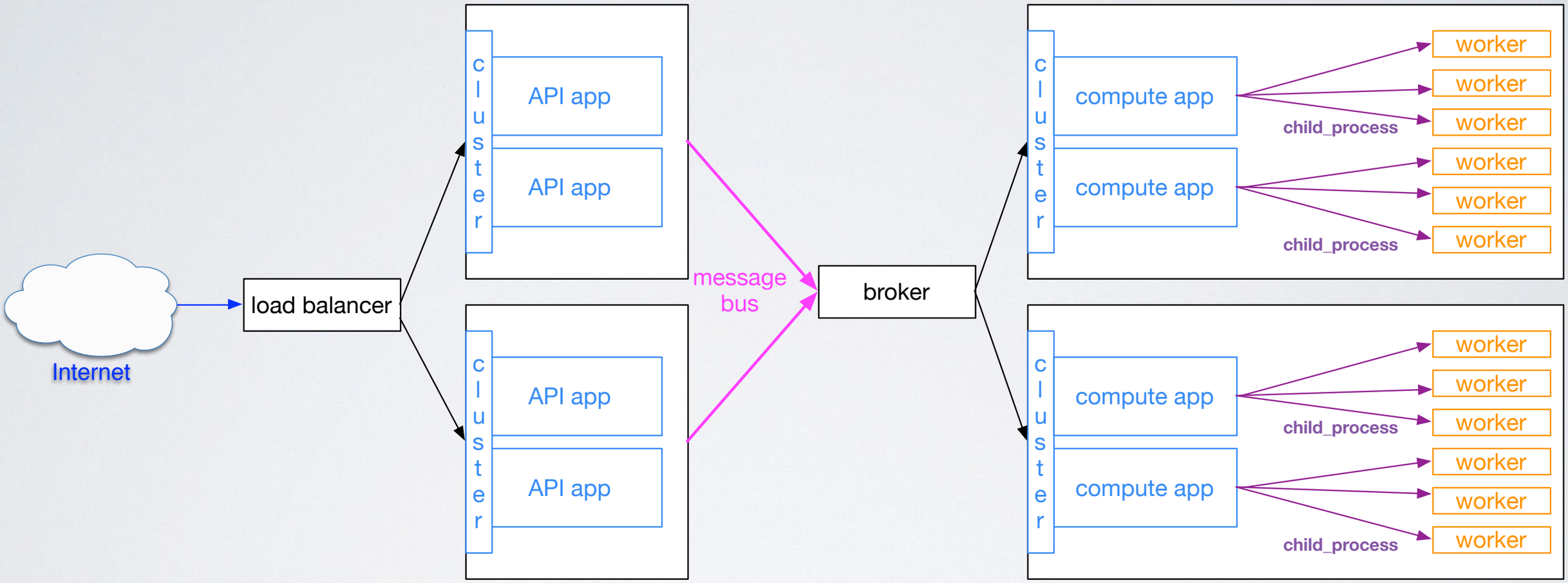
How Do We Multiplex?



How Do We Multiplex?



Introduce a **LB** and a **Broker**



volkan.io

@linkibol

Load Balancing Options

- Elastic Load Balancing as a Service (*amazon, rackspace*…)
- Hardware Load Balancer (*Cisco ACE, Barracuda, etc*…)
- Software Load Balancer
  - nginx
  - haproxy
  - home grown

Load Balancer

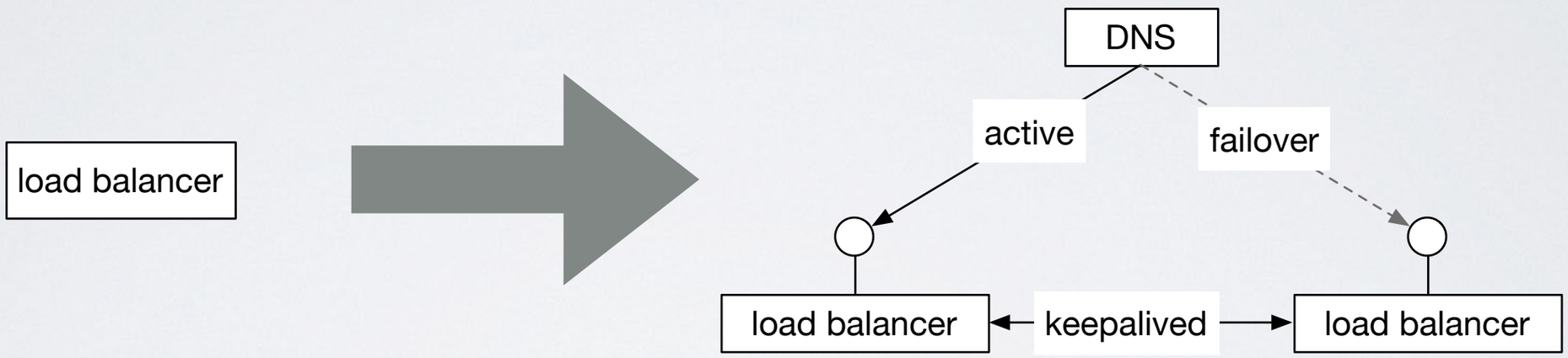
```
1   'use strict';
2
3   var hosts = [
4           'host1.foobar.bazdomain.com',
5           'host2.foobaz.batdomain.com'
6       ],
7       PORT = 8080,
8       index = 0;
9
10  require('bouncy')(function, req, res, bounce) {
11      bounce(hosts[
12          (index = (index + 1) % hosts.length)
13      ], PORT);
14  }).listen(PORT);
```

Making the Load Balancer **Highly Available**

- round-robin DNS
  - https://www.wikiwand.com/en/Round-robin_DNS
- heartbeat
  - https://www.wikiwand.com/en/Heartbeat_(computing)
- **keepalived**
  - http://keepalived.org/

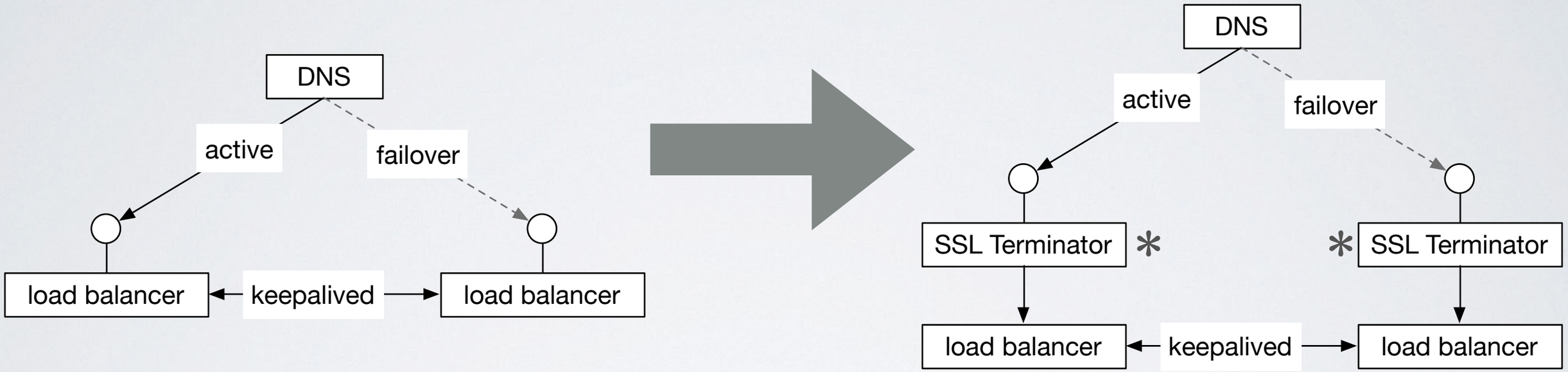
* You can use these tools to make **any** component HA.

Making the Load Balancer **HA**



* see also: https://www.wikiwand.com/en/Virtual_Router_Redundancy_Protocol

SSL Termination

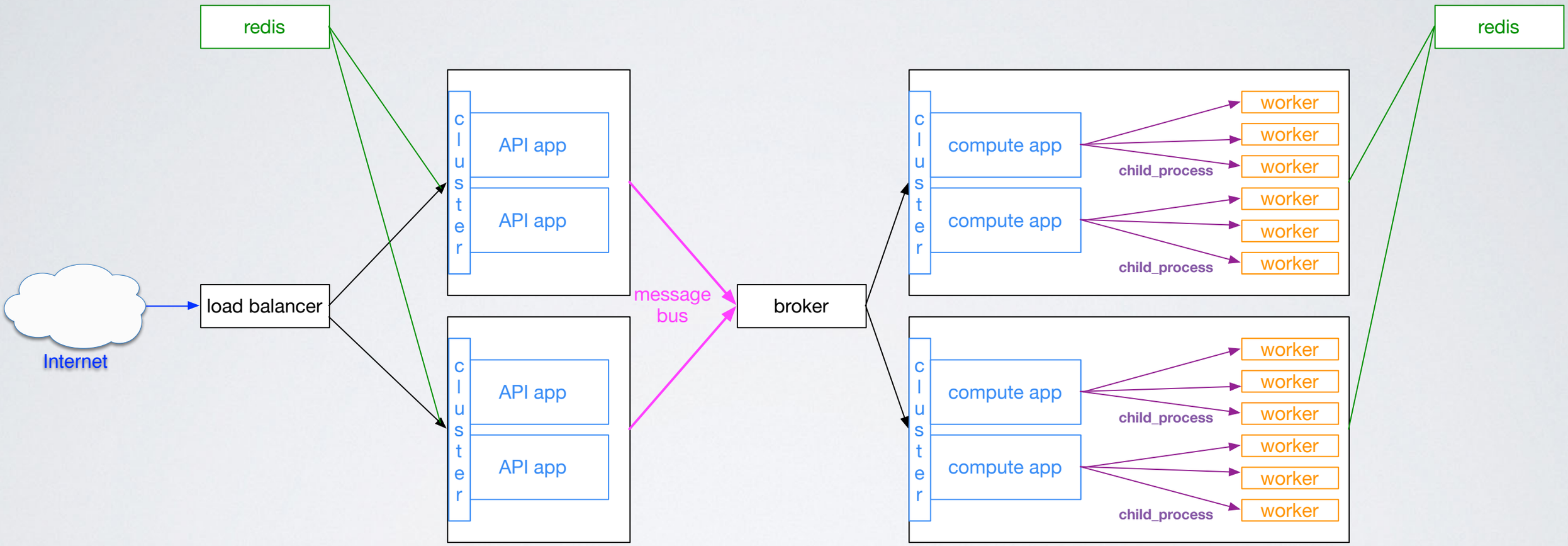


* https://github.com/bumptech/stud

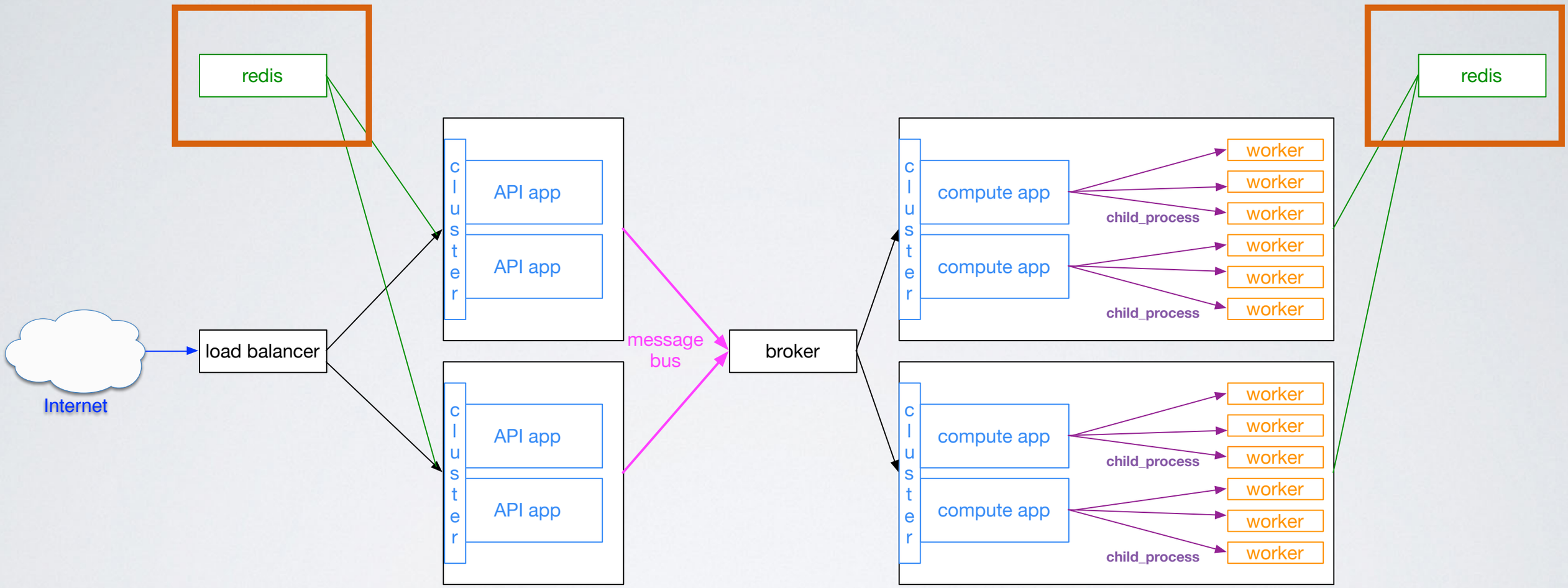
Elephant in the Room

How Do We Manage State?

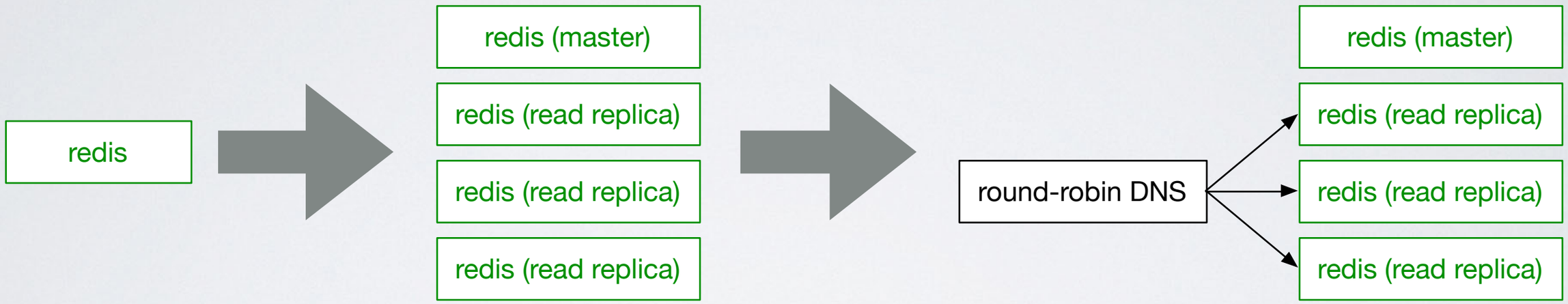
Share State With Redis



Share State With Redis



Make Redis Redundant

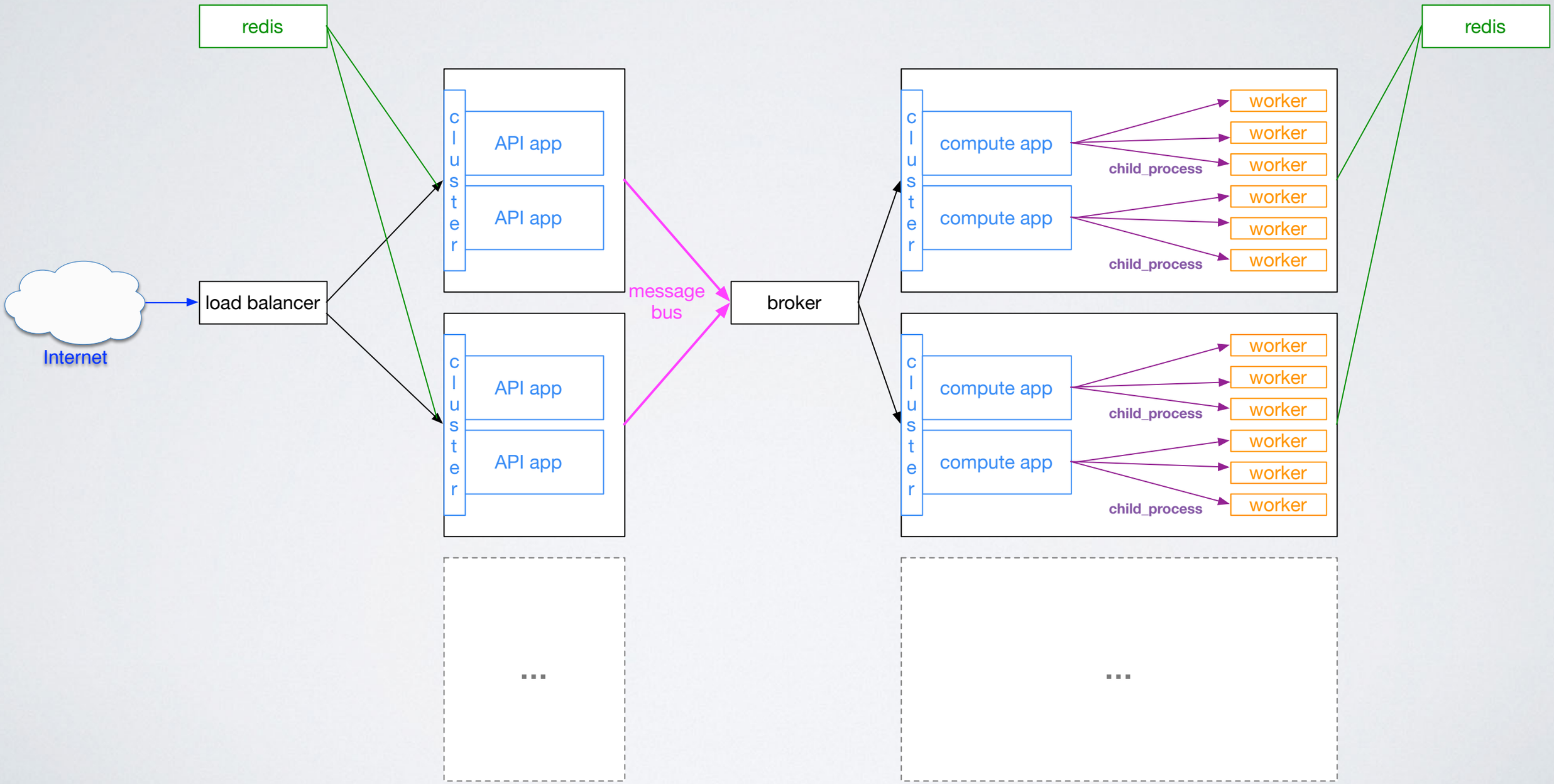


This will also increase throughput as a side benefit.

See http://redis.io/topics/replication and http://redis.io/topics/cluster-tutorial

You can also use a managed "memory as a service" solution.

We Can Add More…

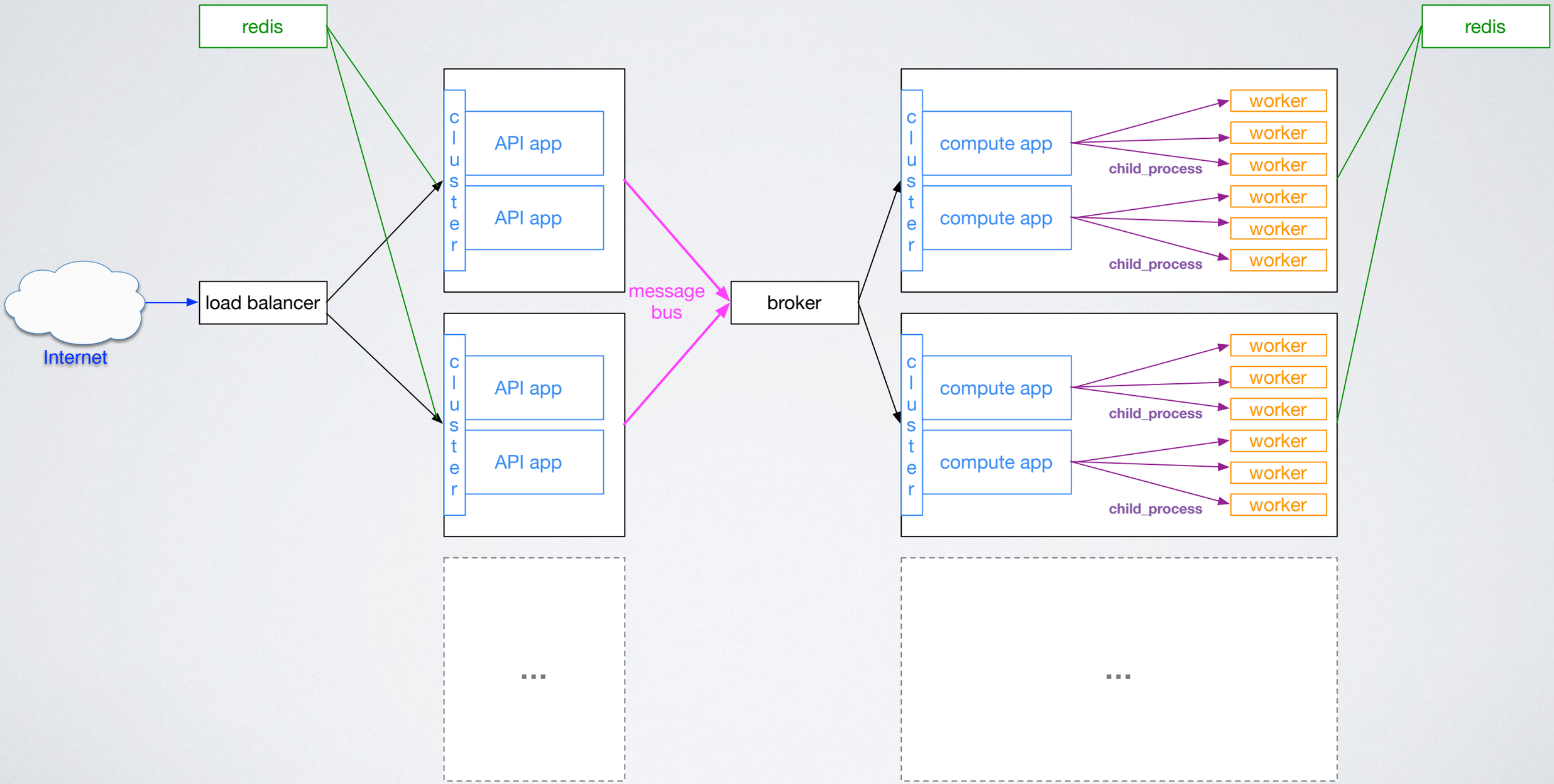


we are actually using

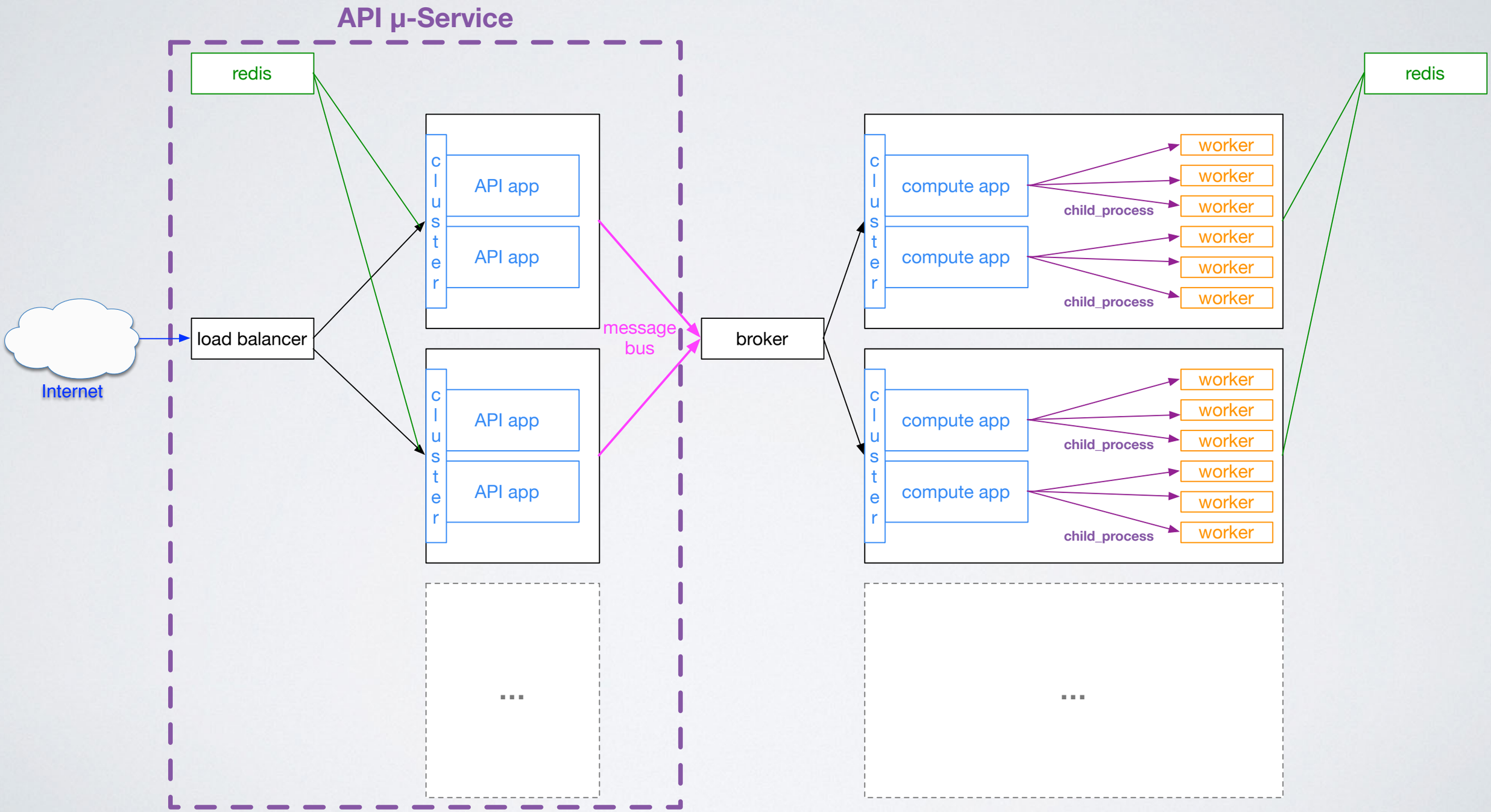
Microservices

here

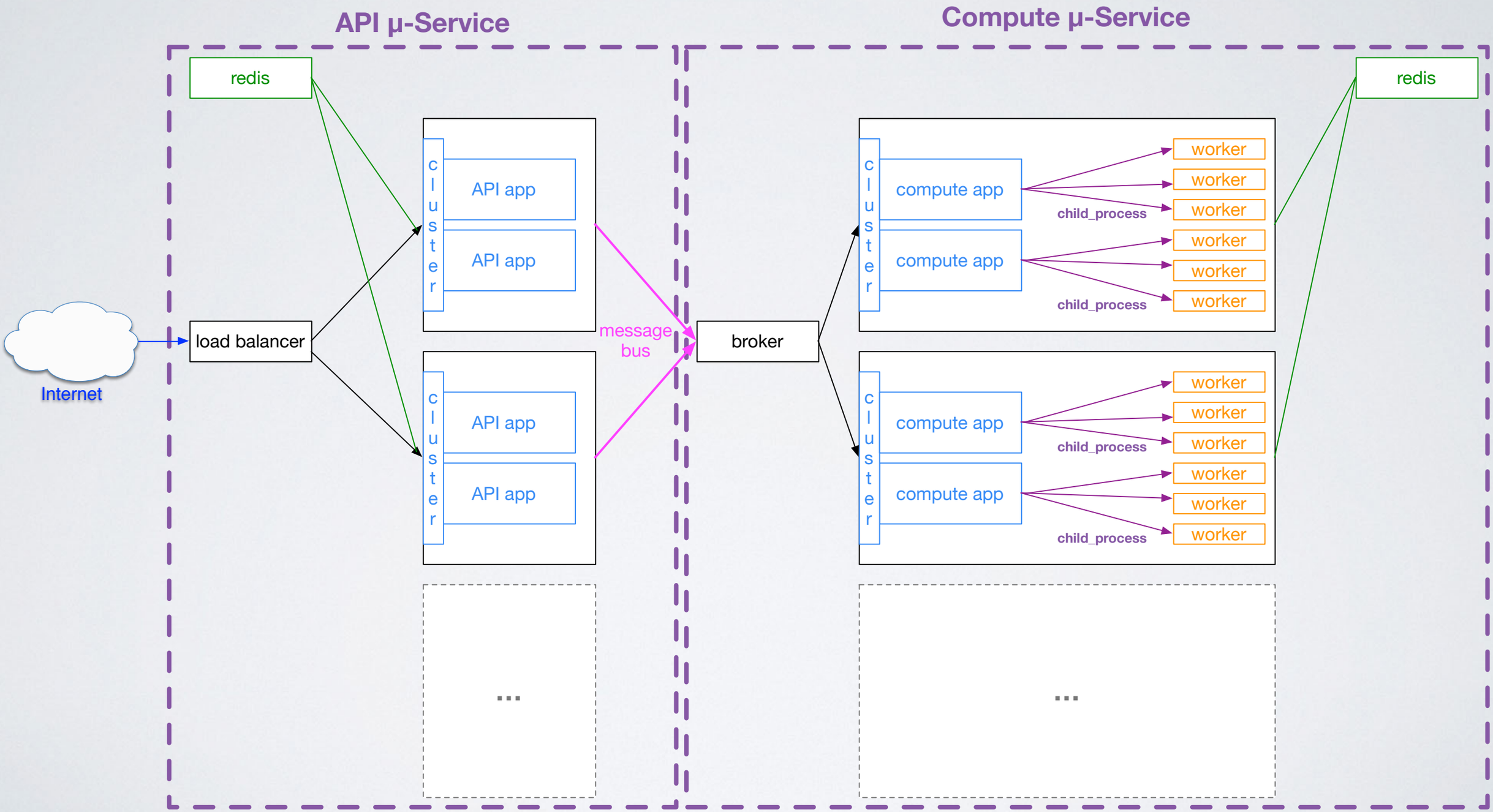
Microservices



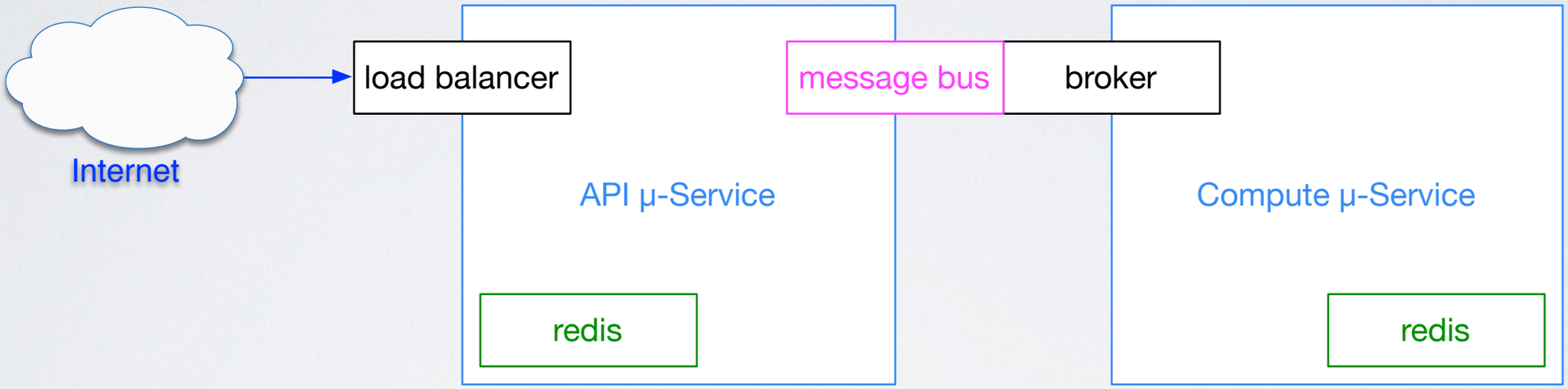
Microservices



Microservices



Microservices

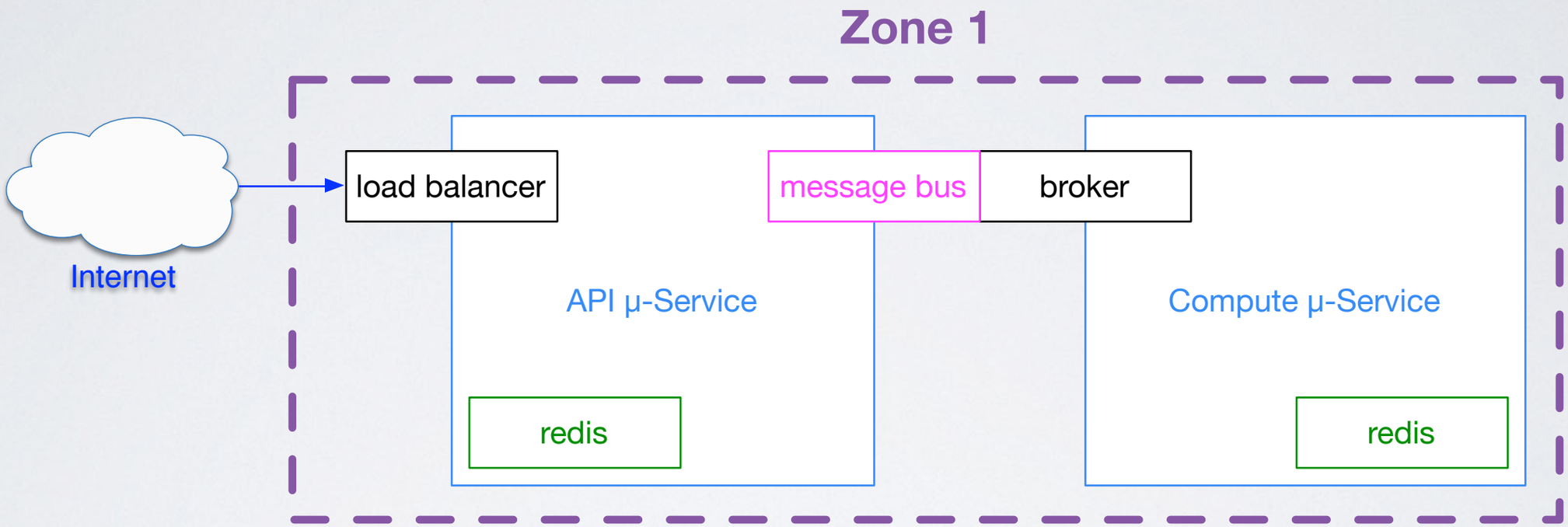


What If We Exhaust All the Bandwidth?

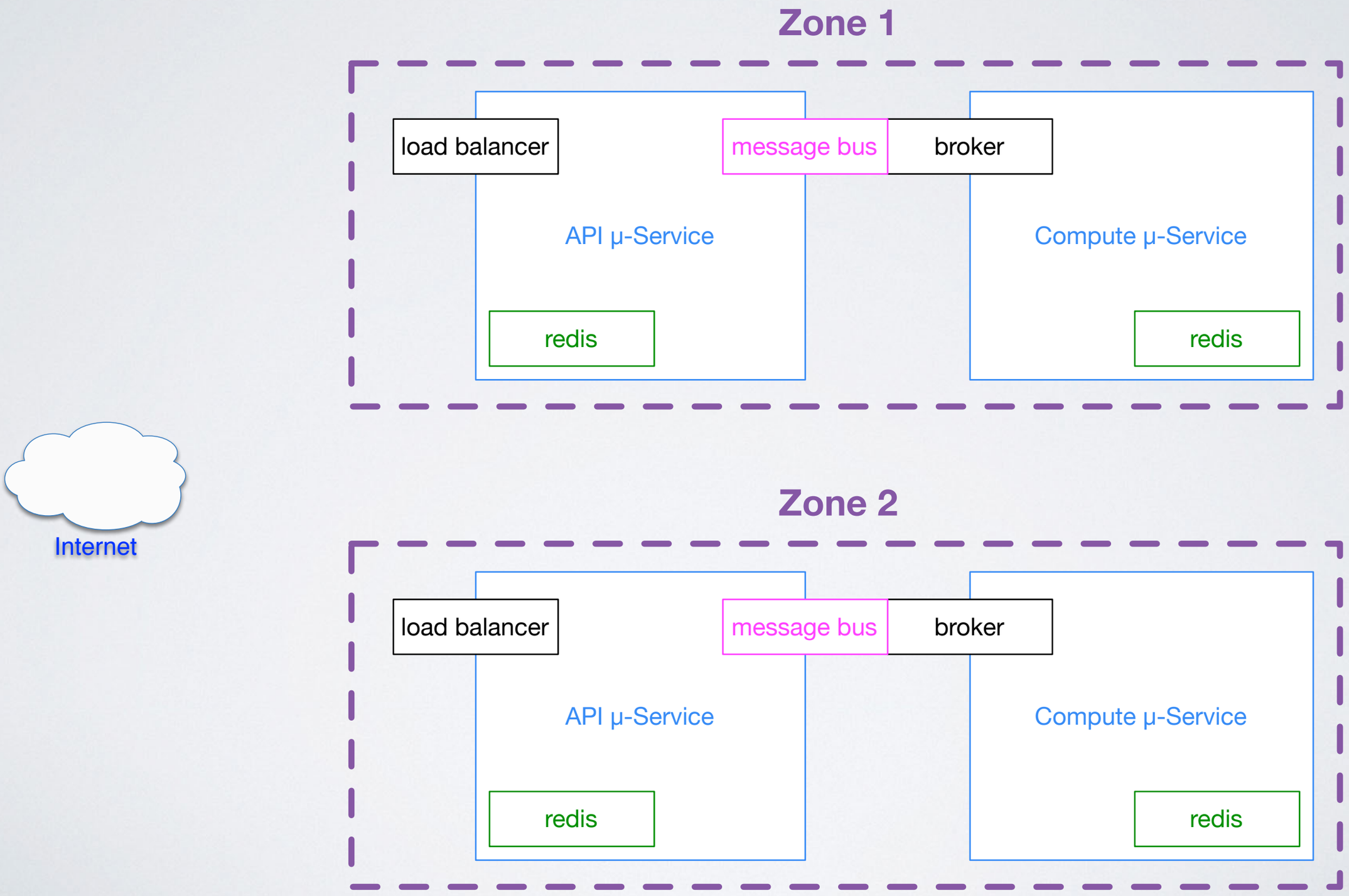
That Means You've Become Famous

Scalability Will Be the Least of Your Concerns

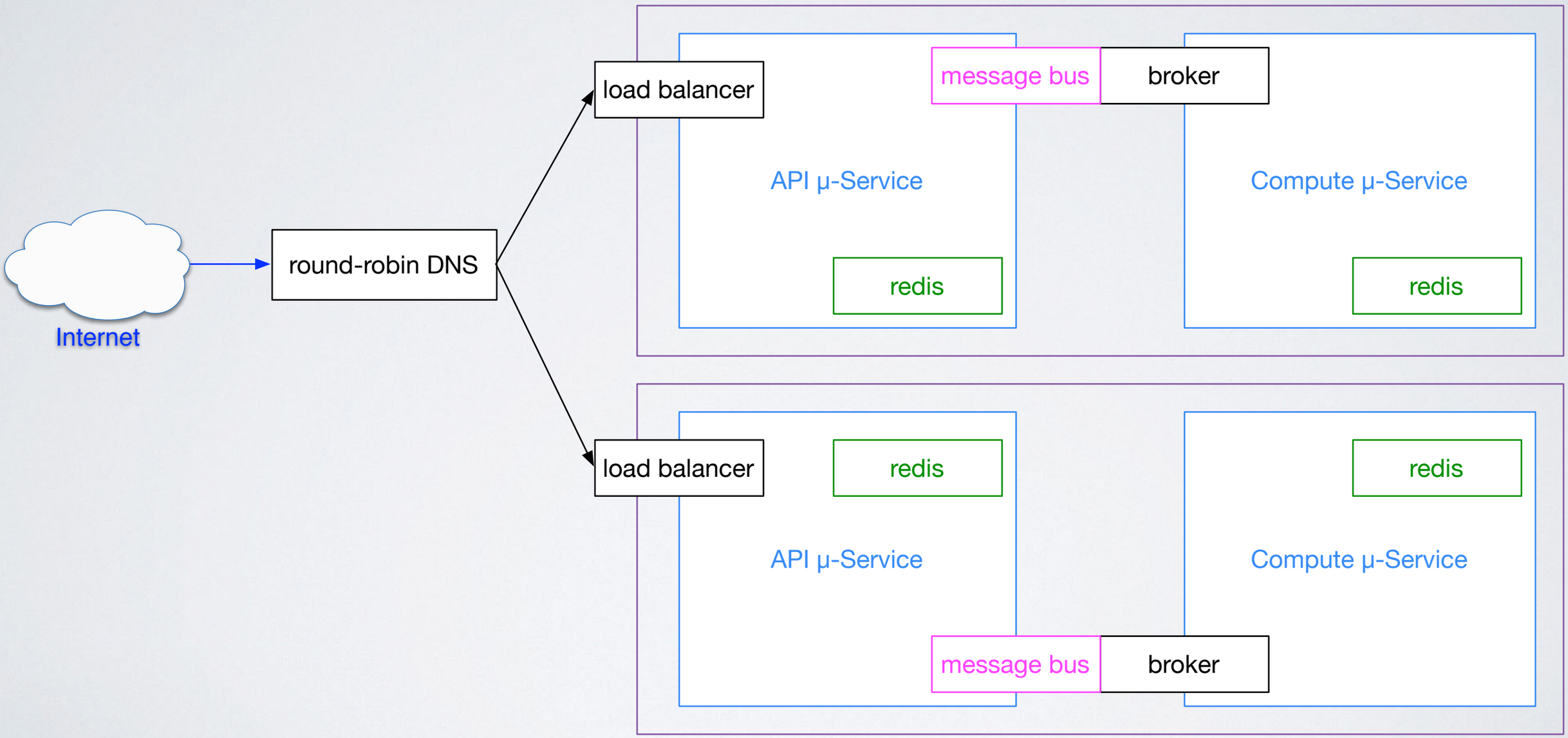
Scaling Into Multiple Zones



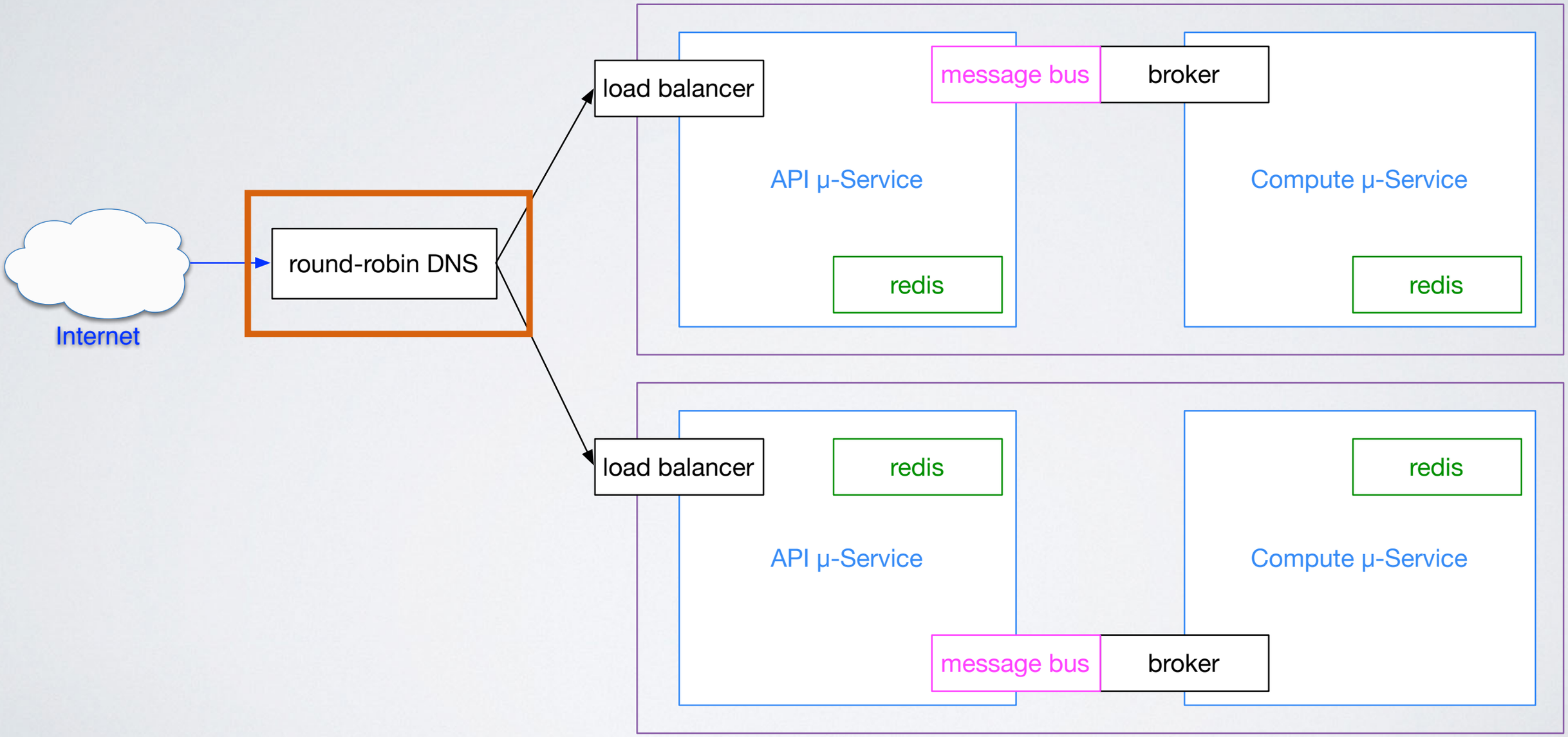
Scaling Into Multiple Zones



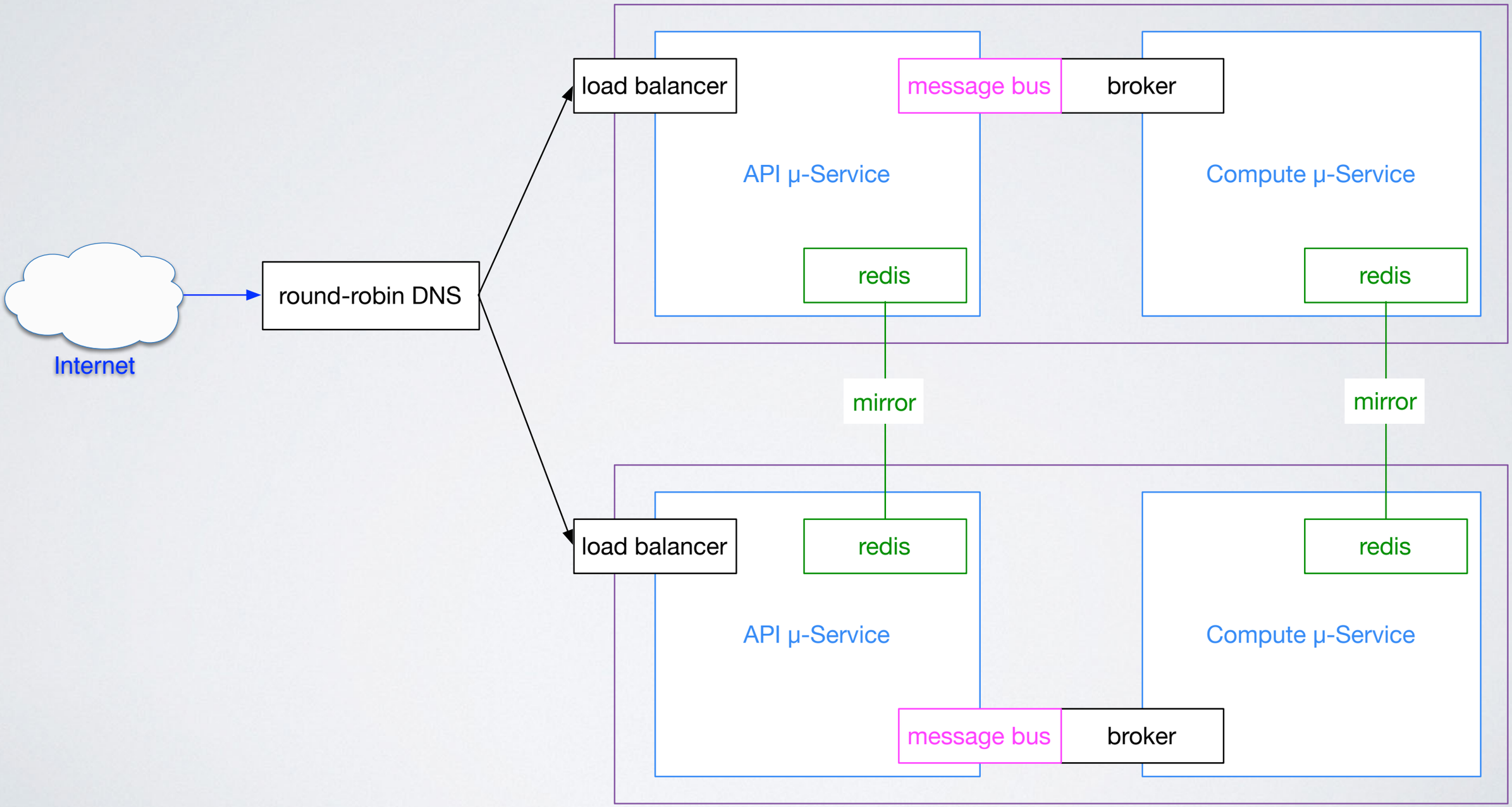
Scaling Into Multiple Zones



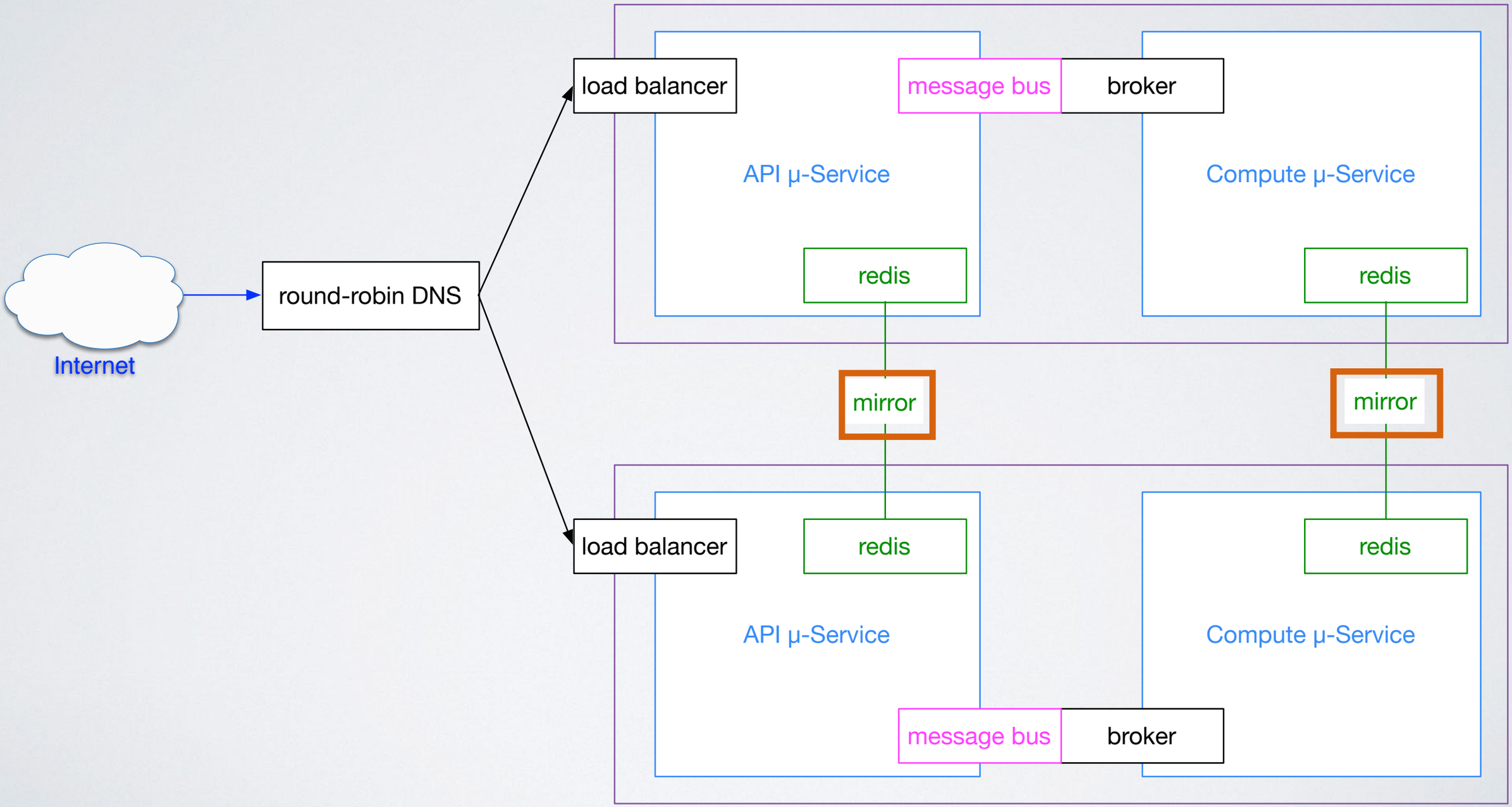
Scaling Into Multiple Zones



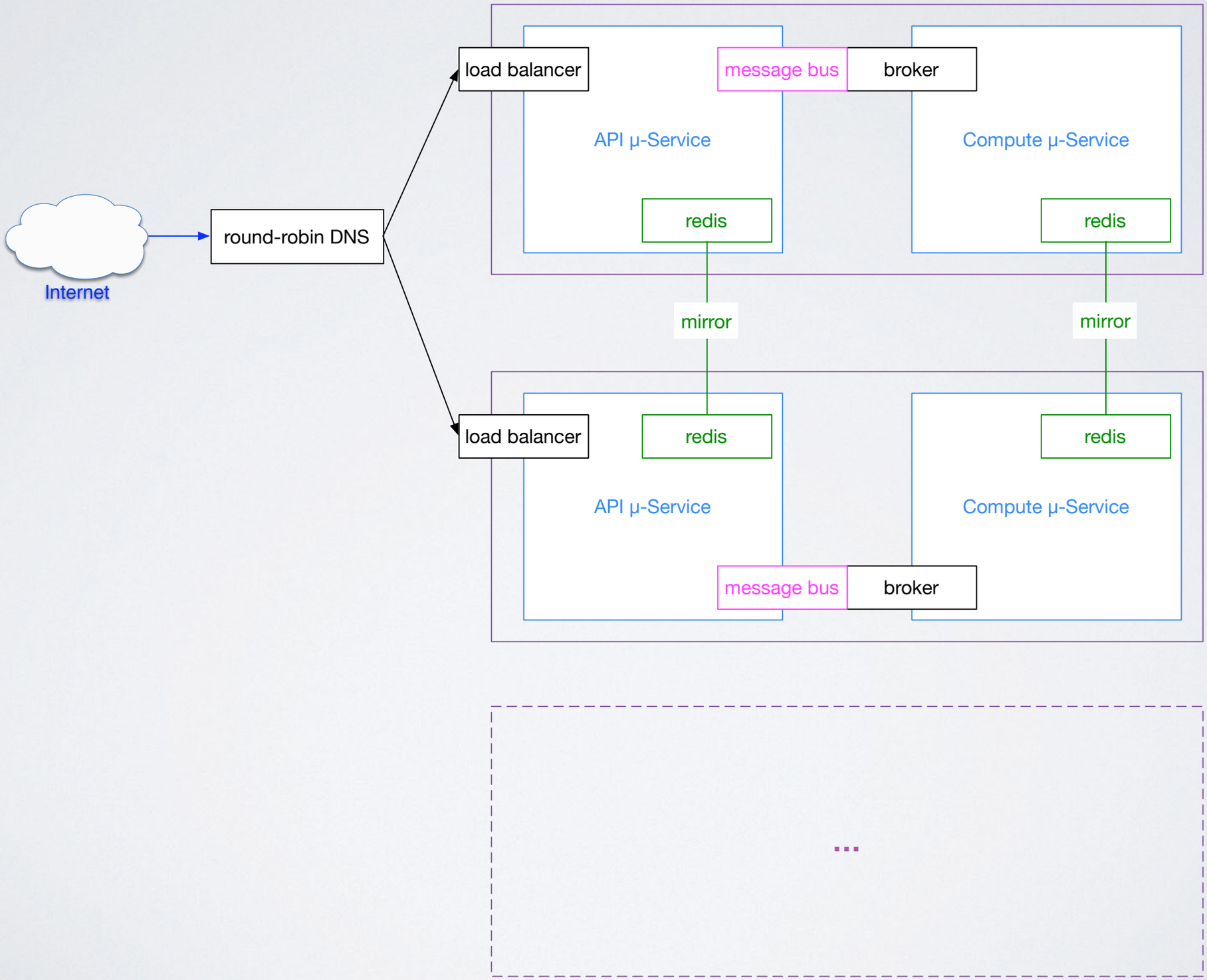
Mirroring the Data Stores



Mirroring the Data Stores



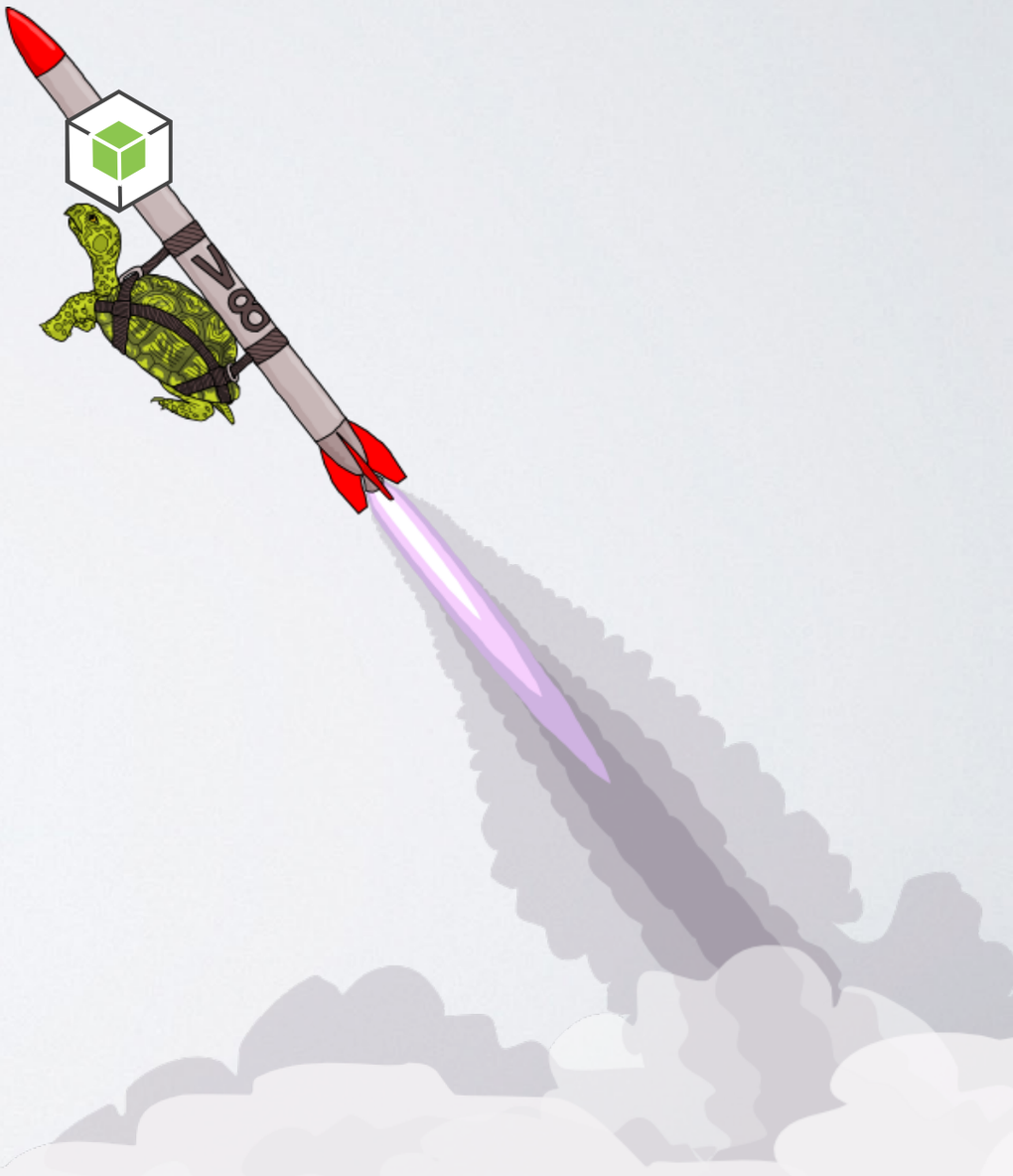
We Can Add More…



scale 2 ∞ & ⇒

Thank You

Questions?



@linkibol