Building a smarter application stack

Tomas Doran
@bobtfish
2014-06-10

Docker is the future

- Preaching to the converted here ;)

- Game changing technology

- No silver bullets (ever)
  - Introduces it's own set of problems and issues
  - Dependency discovery / wiring
  - Scheduling

Smartstack

- One possible solution to discovery problems

- This talk:
  - Application architecture
  - Problem(s) we’re solving
  - Why this solution works well for us
  - Alternate solutions

Microservices - also the future!

- The same as SOA
- But one API per service.
- Own data store!

- Lots of services (dozens, maybe 100s)

- All individually very simple
  - Easy to reason about.
  - Easy to replace

Don’t break the site - ever!!!

- Microservices are individually deployable!
- When we say “Don’t break the site”
  - We mean
  - Don’t break **all of** the site!

Don’t break the site - ever!!!

- If you have graceful degradation…
  - You can ignore MTBF in the backend services!
  - You only care about MTTR.

“I’ll just break this out into it’s own application, as it’ll be easier to maintain in 10 years time”

- Pre seed funding nobody, ever!

Monolith - the reality

- Everyone has one of these :)
- If you’re far enough down the path, you call this ‘`The presentation layer`’.

- Still poses a challenge
  - need async requests
  - need graceful degradation

Monolith - the reality

- Most popular service
- Most dependencies
  - Call into 10s or 100s of other services in a request!
- Needs circuit breakers + dynamic configuration

No silver bullet = No one solution

- You should **always** have 2.
  - Nagios / Sensu
  - RRDs + Ganglia / Graphite + Diamond
  - YAML files / Zookeeper

No silver bullet = No one solution

- ‘Top down’ architecture sucks.
- Instead, broad goals + ’Bottom up’ architecture
  - Internal competition!
  - Replacing the incumbent solution happens organically
  - If your thing works better, people will **want** to move!

- Not perfect! Better than top-down!

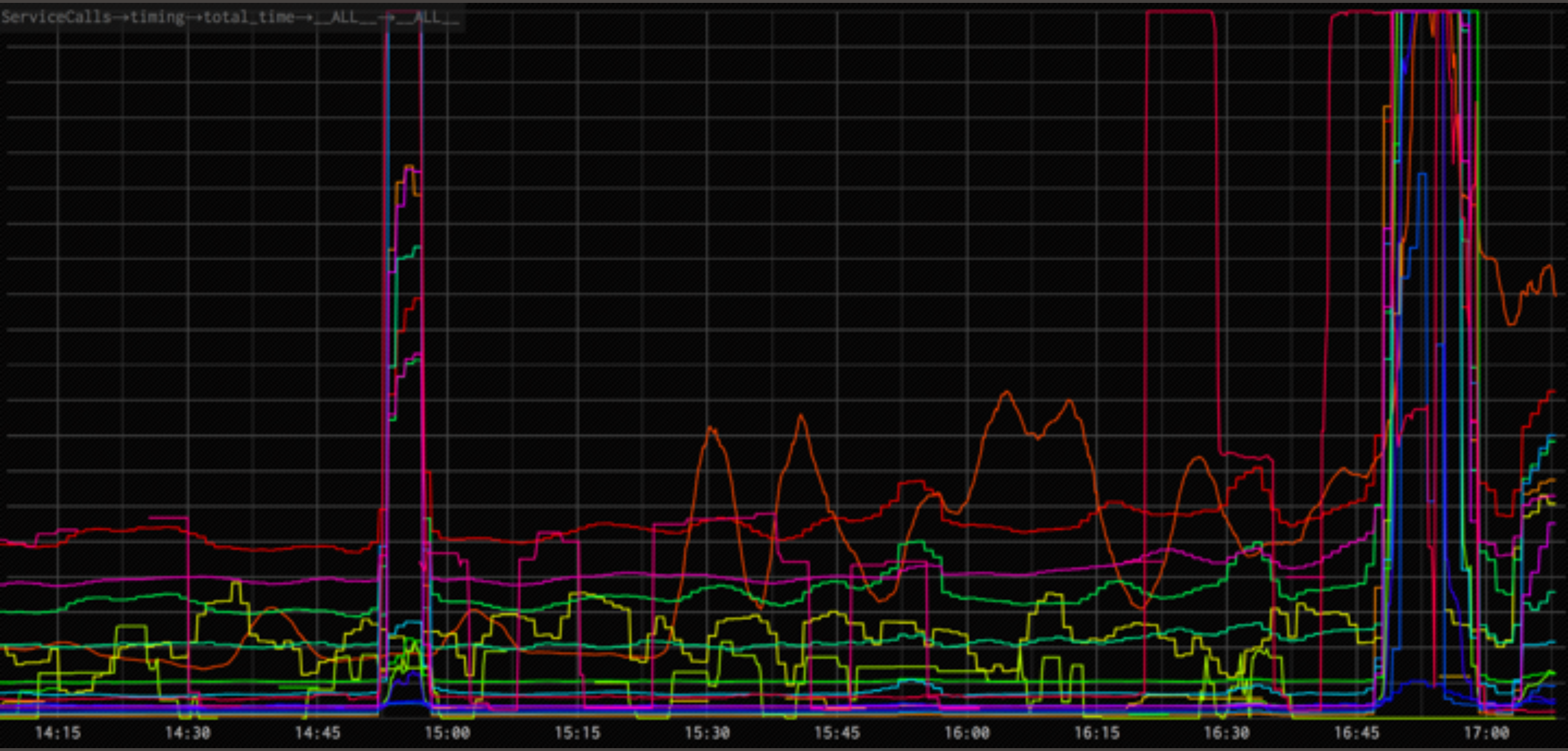
“Humans are bad at predicting the performance of complex systems […]. Our ability to create large and complex systems fools us into believing that we’re also entitled to understand them”

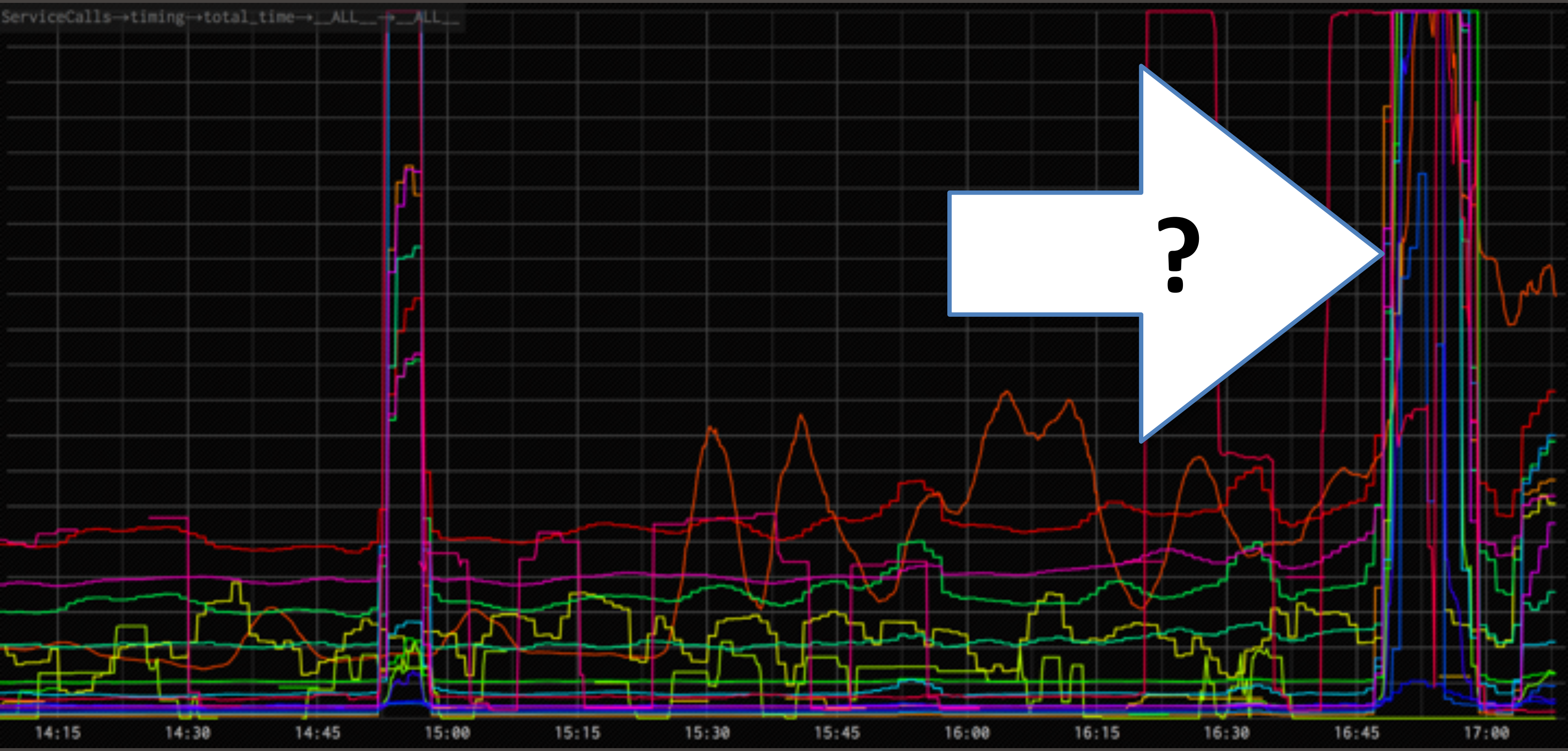
- Carlos Bueno “Mature optimization handbook”

Distributed complexity

- Distributed systems introduce their own set of complexity
  - Reasoning about the whole system challenging
  - Timing/profiling/performance analysis non-trivial
  - Resource scheduling also non-trivial
  - 2nd order effects

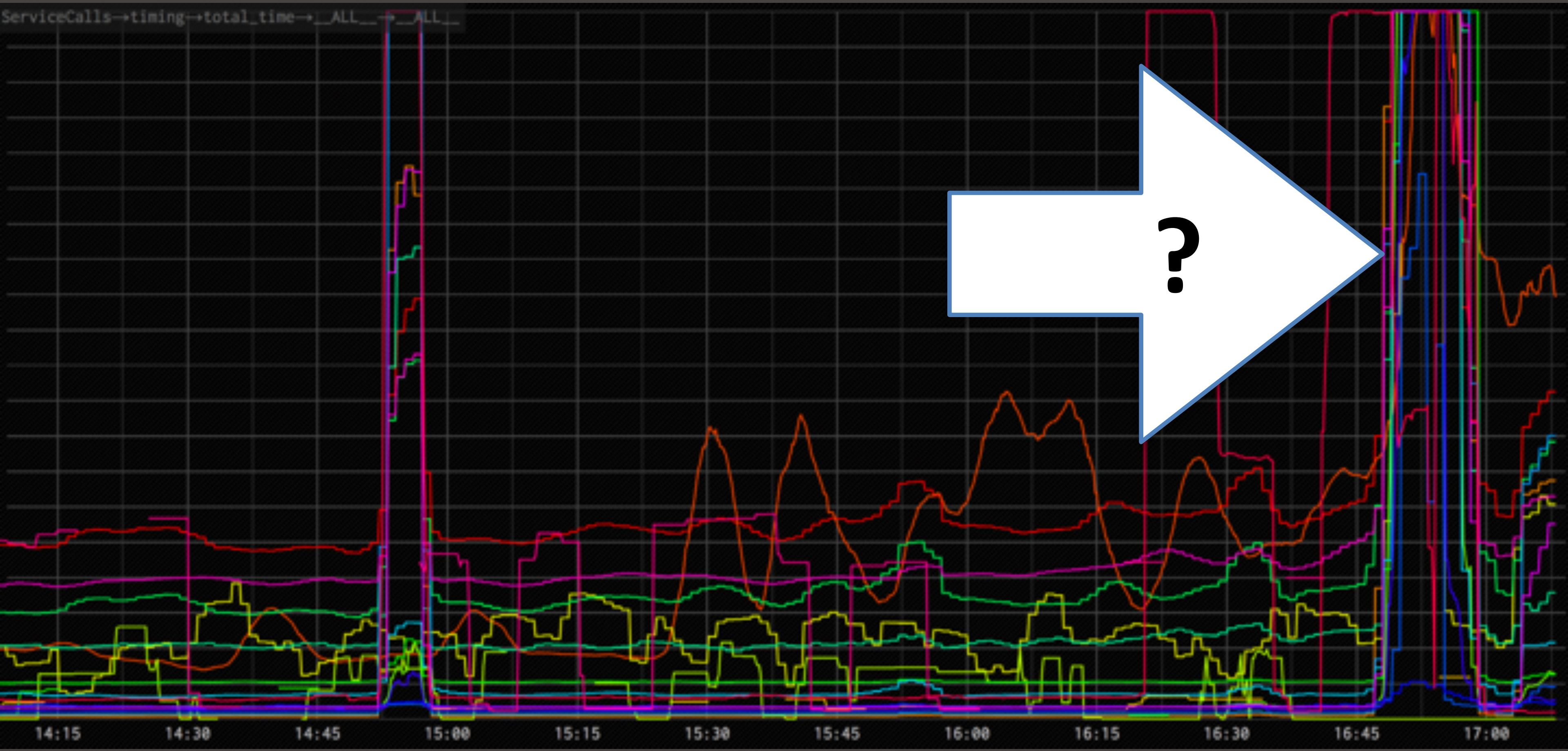
- **Can't** reason about emergent behavior


ServiceCalls→timing→total_time→__ALL__→__ALL__
14:15
14:30
14:45
15:00
15:15
15:30
15:45
16:00
16:15
16:30
16:45
17:00

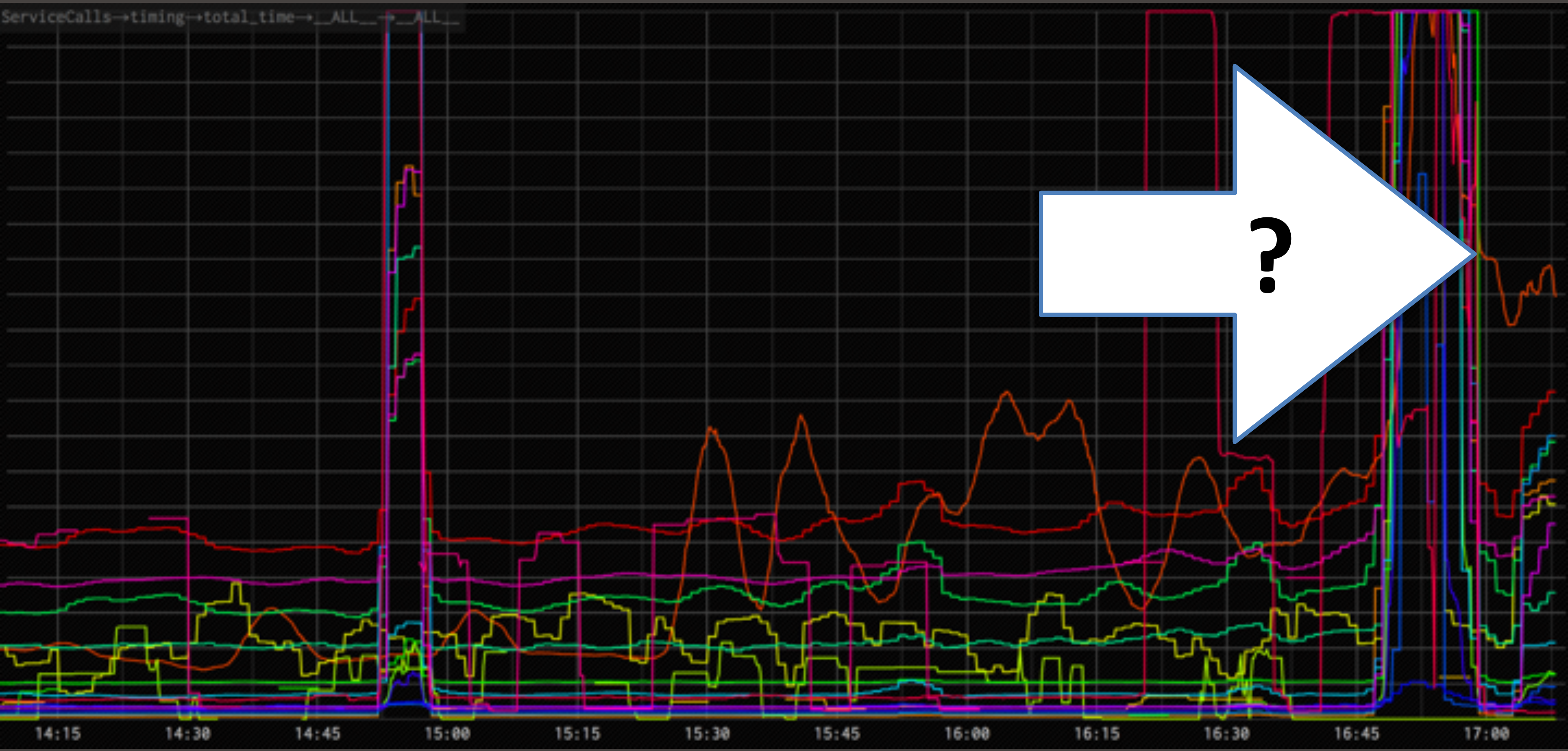

ServiceCalls→timing→total_time→__ALL__→__ALL__
14:15
14:30
14:45
15:00
15:15
15:30
15:45
16:00
16:15
16:30
16:45
17:00

?

What the heck happened at 16:46?



And why did it stop at 17:00?



Dynamic architecture

- Cattle not pets
- AWS and VMs in ‘the cloud’ started this
  - Docker takes it a step further

- Explicitly manage persistent state
- Explicit regular recycling
- **All** updates are redeploys

Dependency nightmares

- Almost everything has **some** dependencies
    - Simple example, web app with a mysql DB
    - App config in a YAML file

- Mysql container address changes when you restart mysql!
    - Oops, app can't find mysql!

- Do I need to restart every application using mysql?
    - Sucks!
- Do I need to rebuild application containers using mysql?
    - To edit the config YAML file!
    - Super super sucks!

Runtime wiring

- mysql failovers - the simple case!

- Presentation layer talking to service REST layers
  - Different deployment schedules
  - No downtime

- Only possible solution: wiring dependencies at runtime
  - A challenge
  - Also an opportunity

- DNS is workable in some cases

Dynamic discovery

- Discovery becomes a core problem
- DNS re-resolving not generally trustworthy
  - You need to test everything for this
- DNS balancing (internally) is awful
  - Failed node + multiple connections/requests
    - DNS round robin
    - Everything sees failure
  - Slow to shift traffic
  - Round robin is crappy for load balancing

Externalized wiring

- Remove a lot of complexity from the application domain
- Run a load balancer (haproxy) on each machine
- Applications always connect to load balancer on fixed host/port
  - localhost on traditional metal/VMs
  - supplied by —link or environment variables in Docker
- Applications become wiring agnostic!

you had one job

‘Client side load balancing’

- Lots of projects use this approach:
  - Project Atomic
  - Marathon + Mesos-Docker
  - vulcand (https://github.com/mailgun/vulcand)
  - Frontrunner (https://github.com/Wizcorp/frontrunner)
  - Consul

- Smartstack

Legacy infrastructure

- Physical machines
- Application images in AMIs
- kvm

- Can’t just use container links or a Docker only solution

- Want to use the same (uniform) solution everywhere.

Entropy reduction

- You can’t change everything at once!

- Everything will tend towards chaos
  - ‘Old infrastructure’
  - ‘New infrastructure’
  - ‘New new infrastructure’

- Solution specifically chosen so it could be generic.

SmartStack

- 2 parts
  - Synapse
  - Nerve

- Conceptually simple
- Very flexible
- Easy to hack on
- Plays well on traditional machines
- Plays well in docker

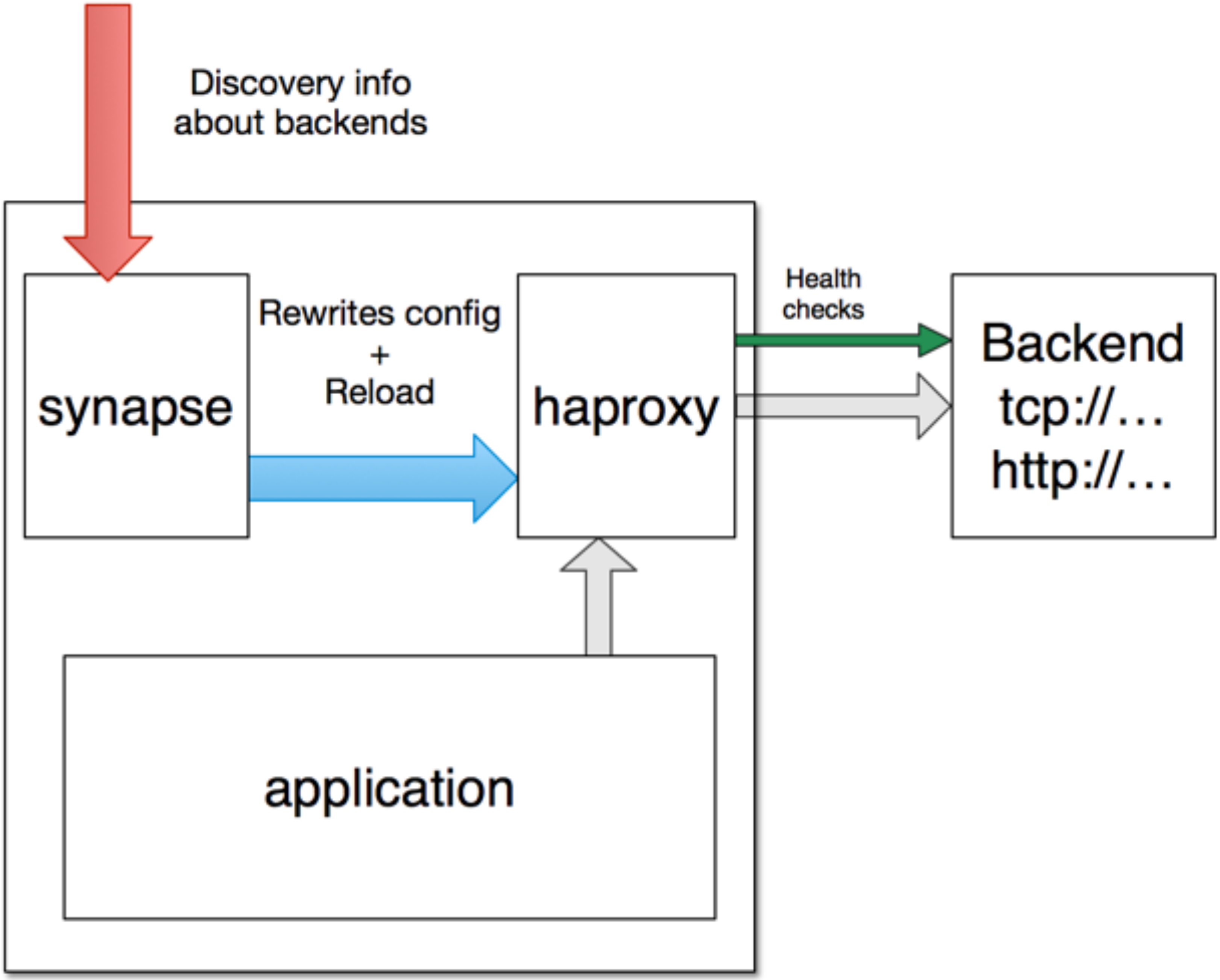
Synapse

- Does discovery against a pluggable backend
- Writes out a haproxy configuration

- Assign a well known port to all services
  - Application connects to that port
  - haproxy forwards to an available backend

- Your application doesn’t need to know about discovery!

- Technology agnostic - works the same on metal/VMs/Docker

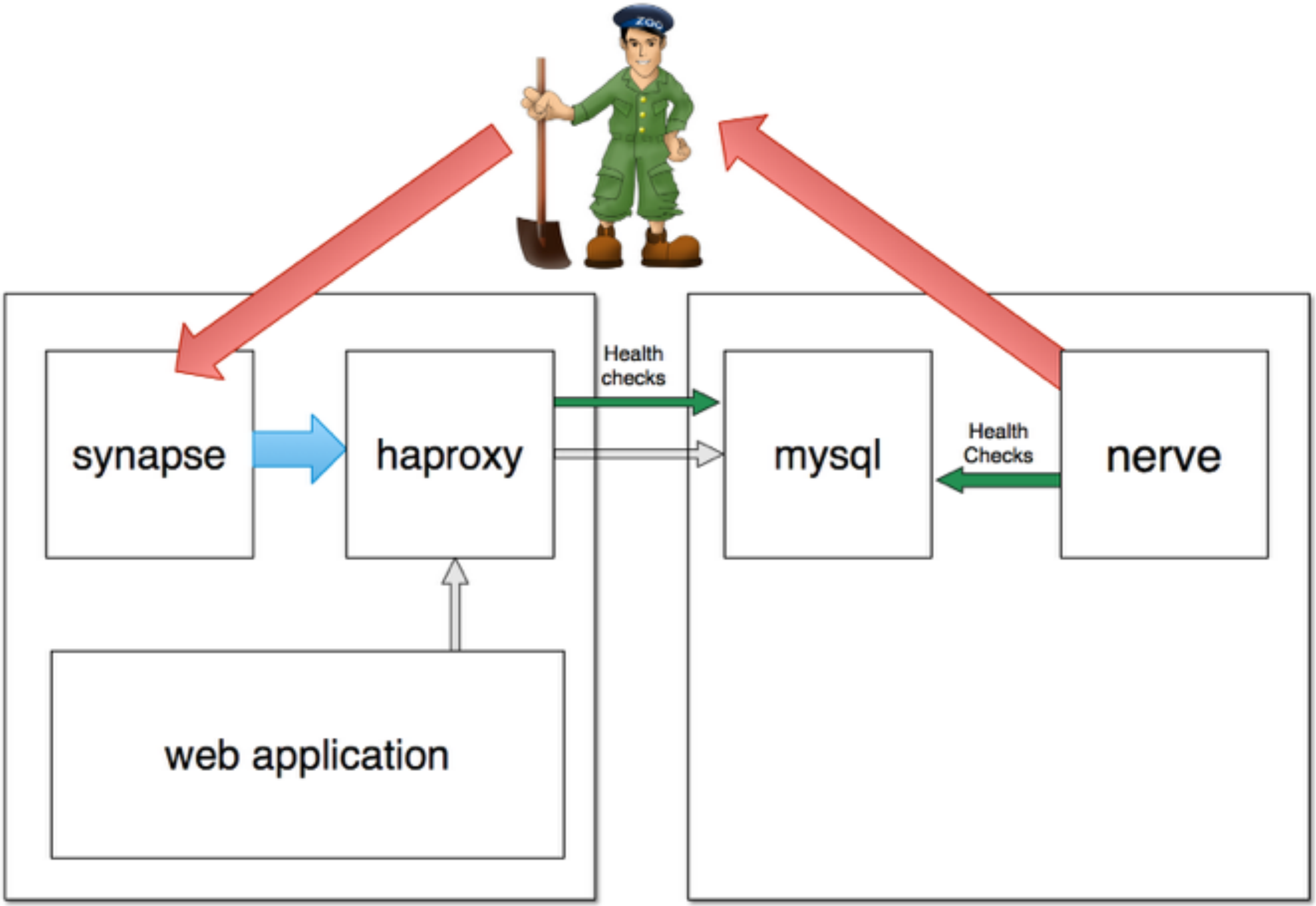

Discovery info about backends
synapse
Rewrites config + Reload
haproxy
Health checks
Backend
tcp://…
http://…
application

Why synapse?

- haproxy is a well known solution
- ruby - easy to modify
- simple (has one job)
- Pluggable - discovery with multiple methods:
  - JSON config (static)
  - zookeeper
  - etcd
  - docker API
  - ec2 tags
- Flexible
  - Deploy one per instance
  - Or pairs as dedicated lbs

Nerve

- Health checks services
  - Health checks are pluggable.
  - HTTP (flexible) + mysql come out the box
- Registers service information to backend
  - zookeeper
  - etcd (beta)

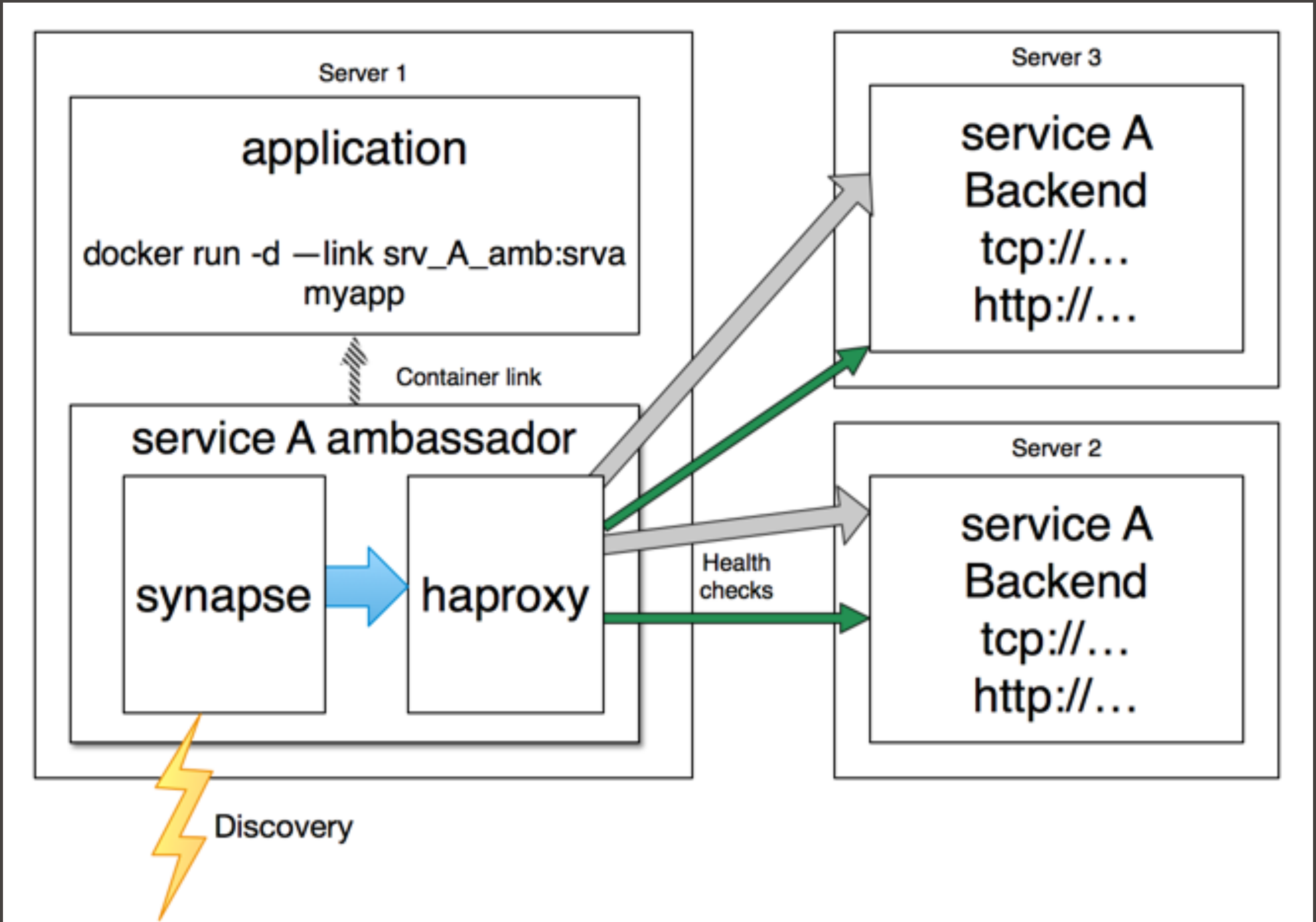

ZOO
synapse
haproxy
Health checks
mysql
Health Checks
nerve
web application

Connector agnostic containers

- On ‘real servers’ or VMs, running a synapse instance per box is fine.
- In docker, we want to abstract more than that
  - Make containers connector agnostic!
  - They don’t need to know or care
  - Upgrade independently.

Synapse <3 ambassador containers

- ‘Ambassador pattern’
  - Run a synapse ‘ambassador’ container on each host for each service
  - Link each application to the ambassador for each of it’s dependencies
  - Environment variables to each service’s haproxy
  - Separates synapse management (i.e. changing the wiring) from application management (i.e. upgrading the app version).


Server 1
application
docker run -d —link srv_A_amb:srva
myapp
Container link
service A ambassador
synapse
haproxy
Discovery
Health
checks
Server 3
service A
Backend
tcp://…
http://…
Server 2
service A
Backend
tcp://…
http://…

Container links

- Ambassador for service A presents:
    - port 8000 for HTTP REST service
    - port 8443 for HTTPS REST service

- Container linking to ambassador sees:
    - SRVA_PORT_8000_TCP=tcp://172.17.0.8:6379
    - SRVA_PORT_8000_TCP_PROTO=tcp
    - SRVA_PORT_8000_TCP_ADDR=172.17.0.8
    - SRVA_PORT_8000_TCP_PORT=6379
    - SRVA_PORT_8443_TCP=tcp://172.17.0.8:6380
    - SRVA_PORT_8443_TCP_PROTO=tcp
    - SRVA_PORT_8443_TCP_ADDR=172.17.0.8
    - SRVA_PORT_8443_TCP_PORT=6380

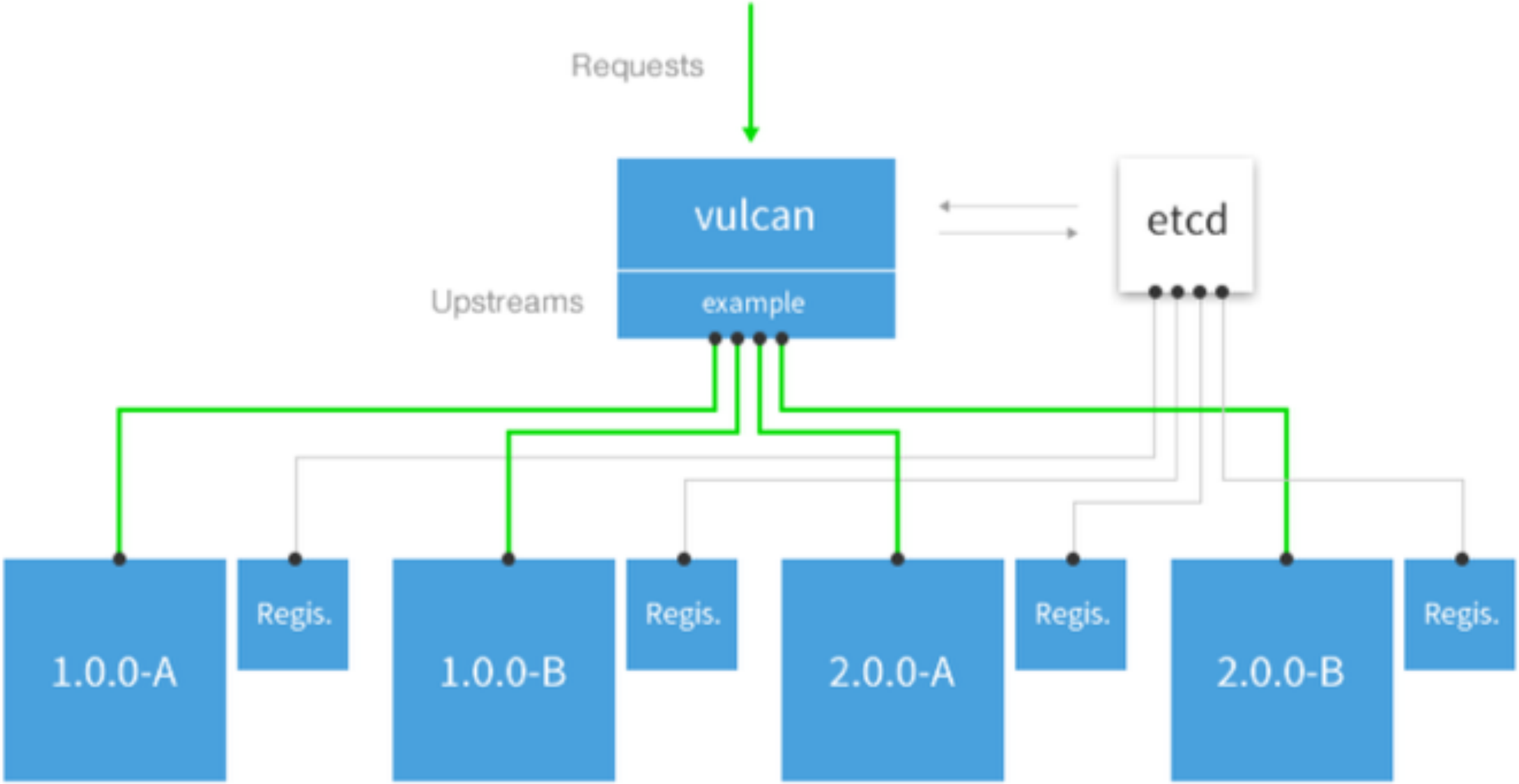
Nerve registration container

- Each app container gets a Nerve instance
- Nerve registers its 1 app
- Nerve instance can be generic
  - Make services all have a standard /health endpoint
  - Healthchecks standard
  - Only need one nerve container image!

Alternate options

- Just register the contents of the docker API into etcd
  - http://coreos.com/blog/docker-dynamic-ambassador-powered-by-etcd/
  - No health checks
  - Docker only
- confd
- Consul
- frontrunner - discovery from Marathon
  - Uses haproxy too
  - Less health checking options

Vulcand


Requests
vulcan
etcd
Upstreams
example
Regis.
Regis.
Regis.
Regis.
1.0.0-A
1.0.0-B
2.0.0-A
2.0.0-B

Issues

- If you have lots of machines + services, you have a lot of Synapses
  - haproxy health checks can become expensive on end user apps
  - Nerve helps with this

- Lots of small load balancers is harder to reason about than a few big ones

Live demo?

Thanks

- Slides will be online
  http://slideshare.net/bobtfish

- Official Smartstack site:
  http://nerds.airbnb.com/smartstack-service-discovery-cloud/

- Pre-built containers to play with + blog post
  http://engineeringblog.yelp.com/
  https://index.docker.io/u/bobtfish/synapse-etcd-amb/
  https://index.docker.io/u/bobtfish/nerve-etcd/

- Questions?