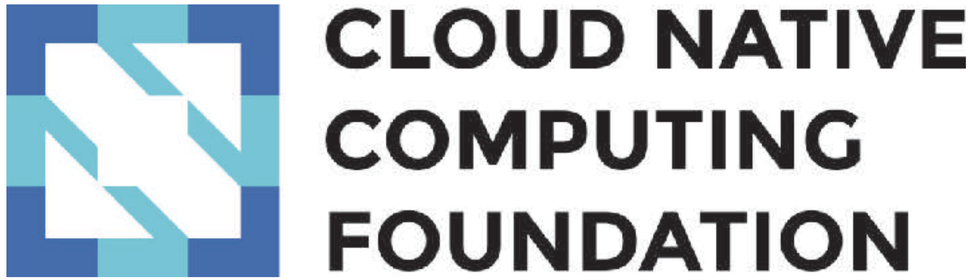
# linkerd

Buoyant

William Morgan ~ william@buoyant.io ~ @wm

Linkerd (“linker-dee”) is an open source *service mesh* for cloud-native applications




**linkerd.io**

**slack.linkerd.io**

**github.com/linkerd/linkerd**

Buoyant

# By the numbers

**13** months old

**600+** Slack channel members

**1600+** Github stars

**200k+** Docker Hub pulls

**30+** contributors

**20+** confirmed prod users

**100b+** production requests









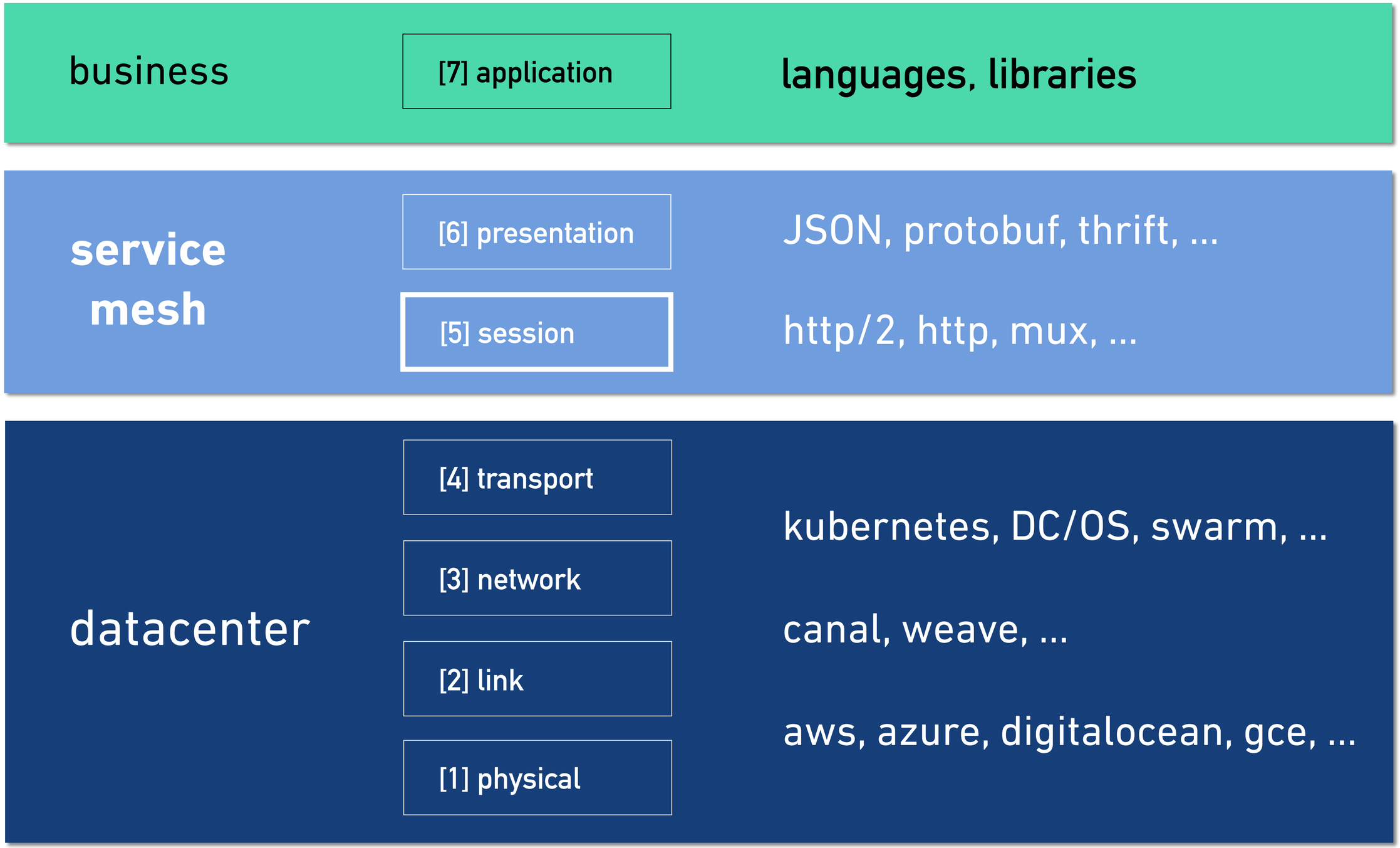



# What's a service mesh?

A dedicated infrastructure layer for **service-to-service** communication.

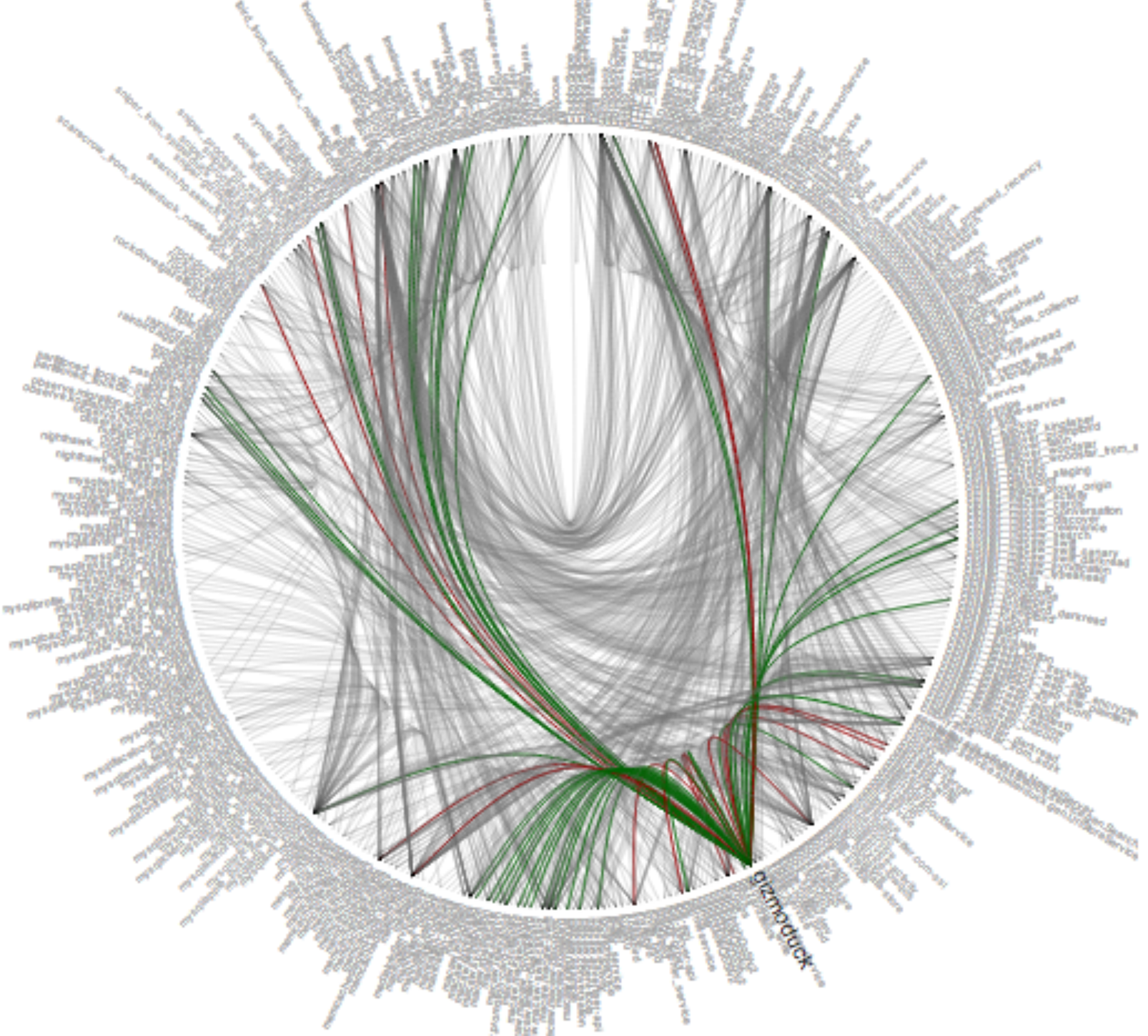
Decoupled from the application.

Focused on *services* and *requests*.


business
[7] application
languages, libraries
service mesh
[6] presentation
JSON, protobuf, thrift, ...
[5] session
http/2, http, mux, ...
datacenter
[4] transport
kubernetes, DC/OS, swarm, ...
[3] network
canal, weave, ...
[2] link
aws, azure, digitalocean, gce, ...
[1] physical

# Why do I need a service mesh?

Because service-to-service (“east-west”) communication needs to be **monitored**, **managed**, and **controlled**.

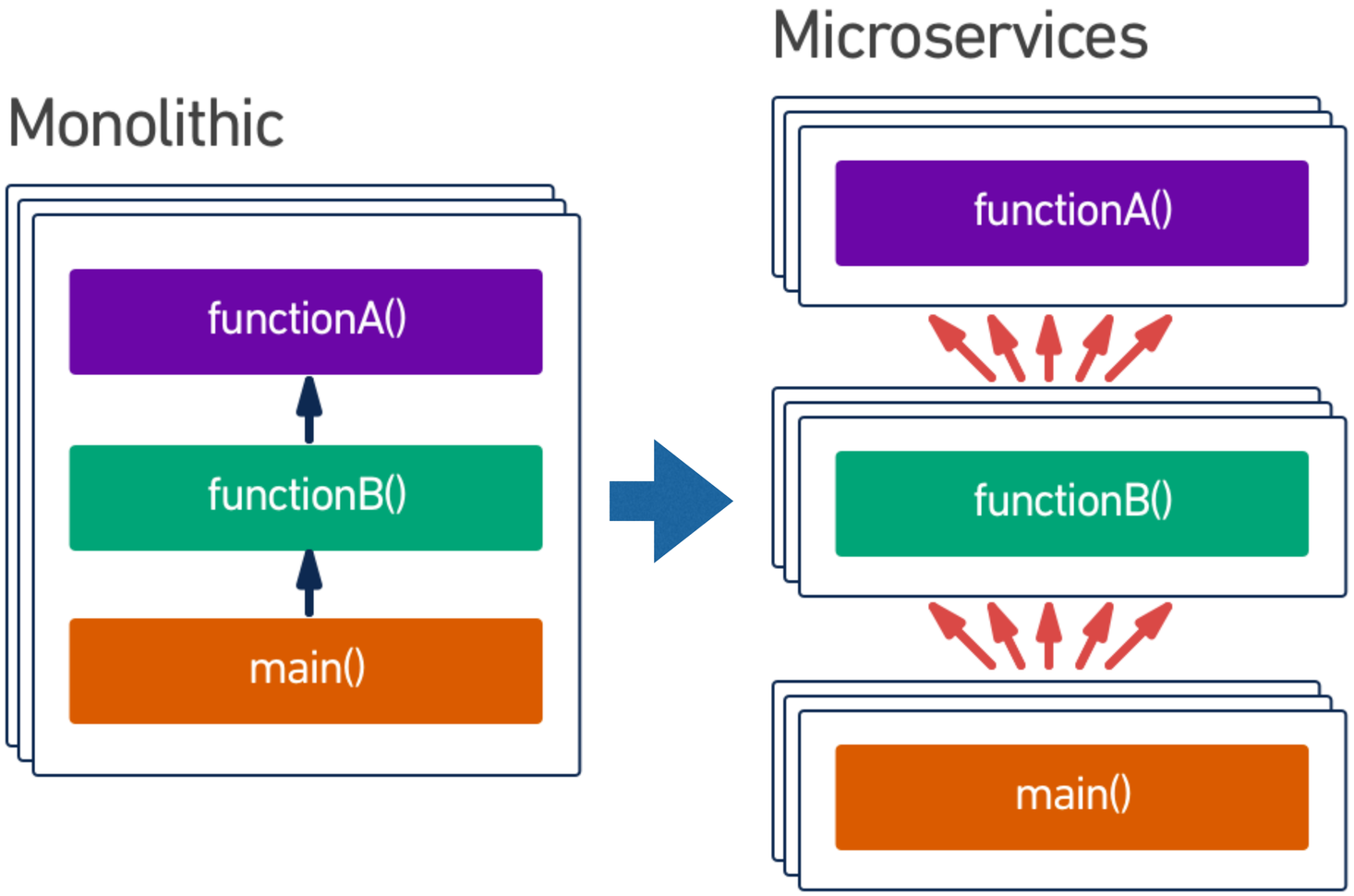



Buoyant

# But I never needed this before!

You weren't running containerized microservices in an orchestrated environment before.

Buoyant

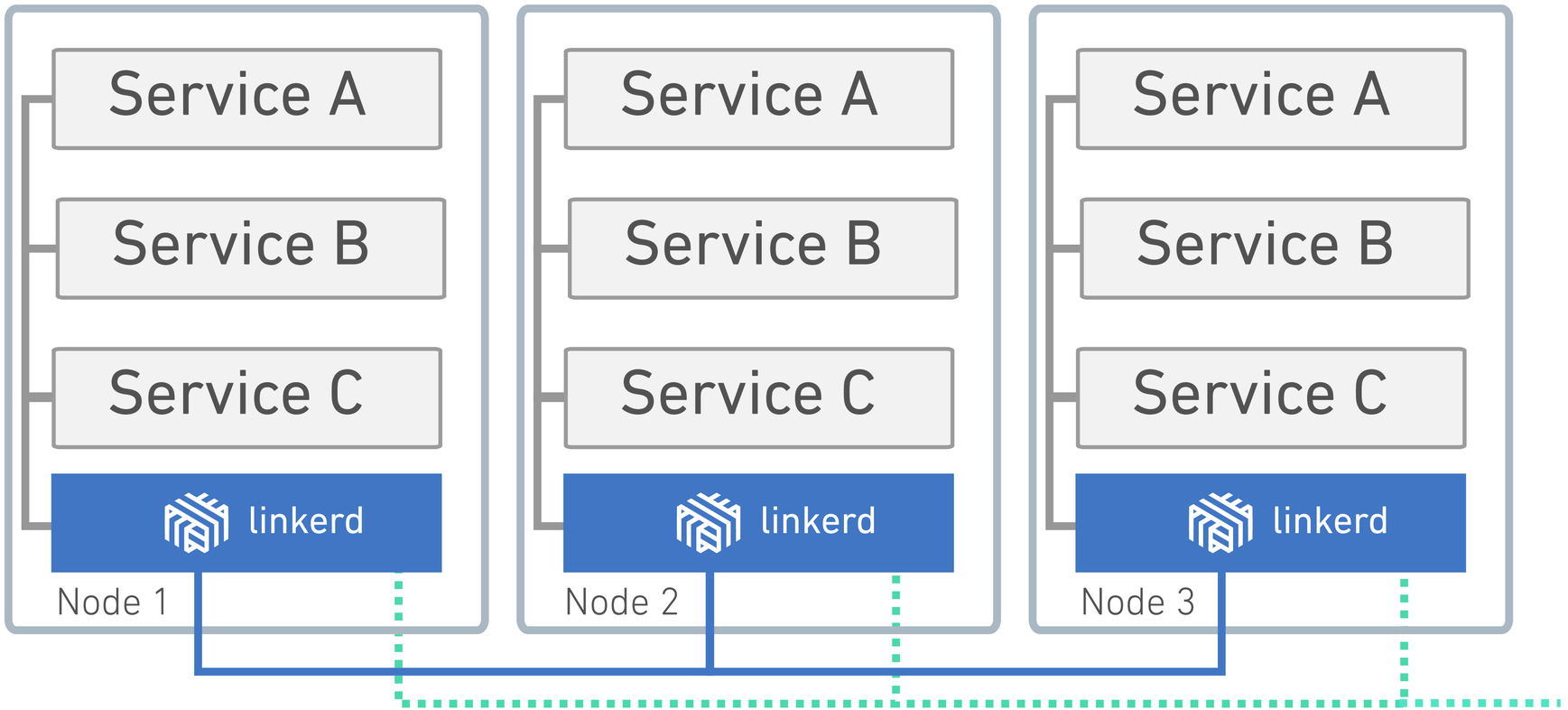

Monolithic
functionA()
functionB()
main()
Microservices
functionA()
functionB()
main()

Buoyant

# How does it work?

1. Linkerd is deployed per-host or per-pod.
2. It acts as a transparent proxy + reverse proxy for internal requests.
3. Applications send their HTTP/gRPC/... calls through their local Linkerd instance
4. That's it!

# The Linkerd service mesh



# What does it do?

**Adds reliability**: latency-aware load balancing, circuit breaking, retry budgets, deadlines

**Decouples transport protocol from app protocol**: transparent TLS, HTTP/1.1 -> HTTP/2, ...

**Sanitized naming**: decouples architectural names (the "users" service") from deployment names ("DC1/prod/users/v4")

# What does it do? (Part II)

**Adds logical routing and traffic shifting**: routing rules give runtime control over logical -> concrete mapping

**Glues worlds**: multiple SDs, e.g. merge K8s and non-K8s service namespaces!

**Failover and hybrid cloud**: unified routing layer

**Consistent, global metrics!** Provides distributed traces and top-line metrics like success rates and latencies

# But Kubernetes already has load balancing / service discovery / ...

## A Service Mesh for Kubernetes, Part I: Top-line service metrics

Alex Leong | 4 October 2016

What is a service mesh, and how is it used by cloud native apps—apps designed for the cloud? In this article, we'll show you how to use linkerd as a service mesh on Kubernetes, and how it can capture and report top-level service metrics such as success rates, request volumes, and latencies without requiring changes to application code.

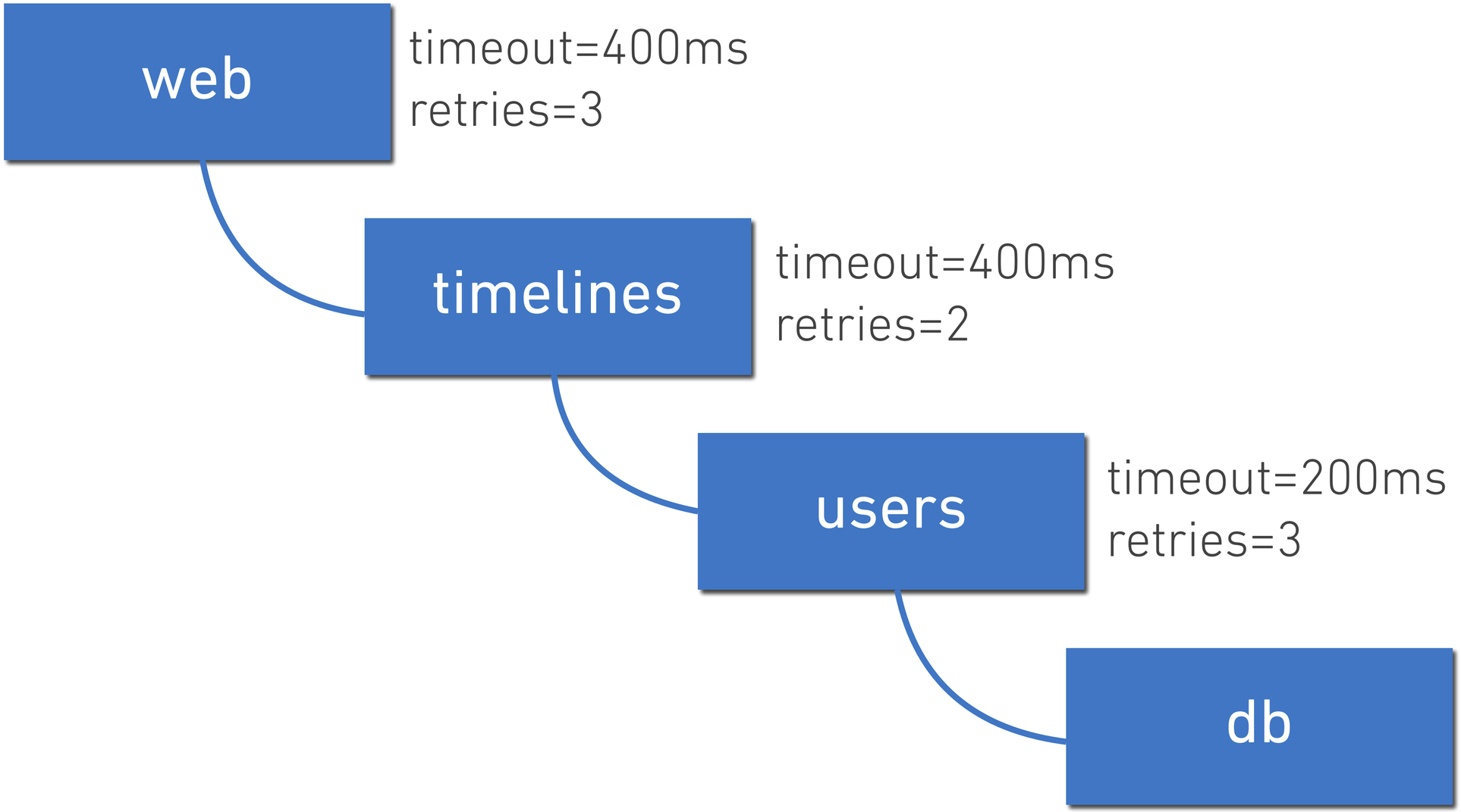
Note: this is Part I of a series of articles about linkerd and cloud native applications. In upcoming weeks, we'll cover:

1. Top-line service metrics (this article)
2. Pods are great, until they're not
3. Encrypting all the things
4. Continuous deployment via traffic shifting
5. Dogfood environments, ingress, and edge routing
6. Staging microservices without the tears
7. Distributed tracing made easy

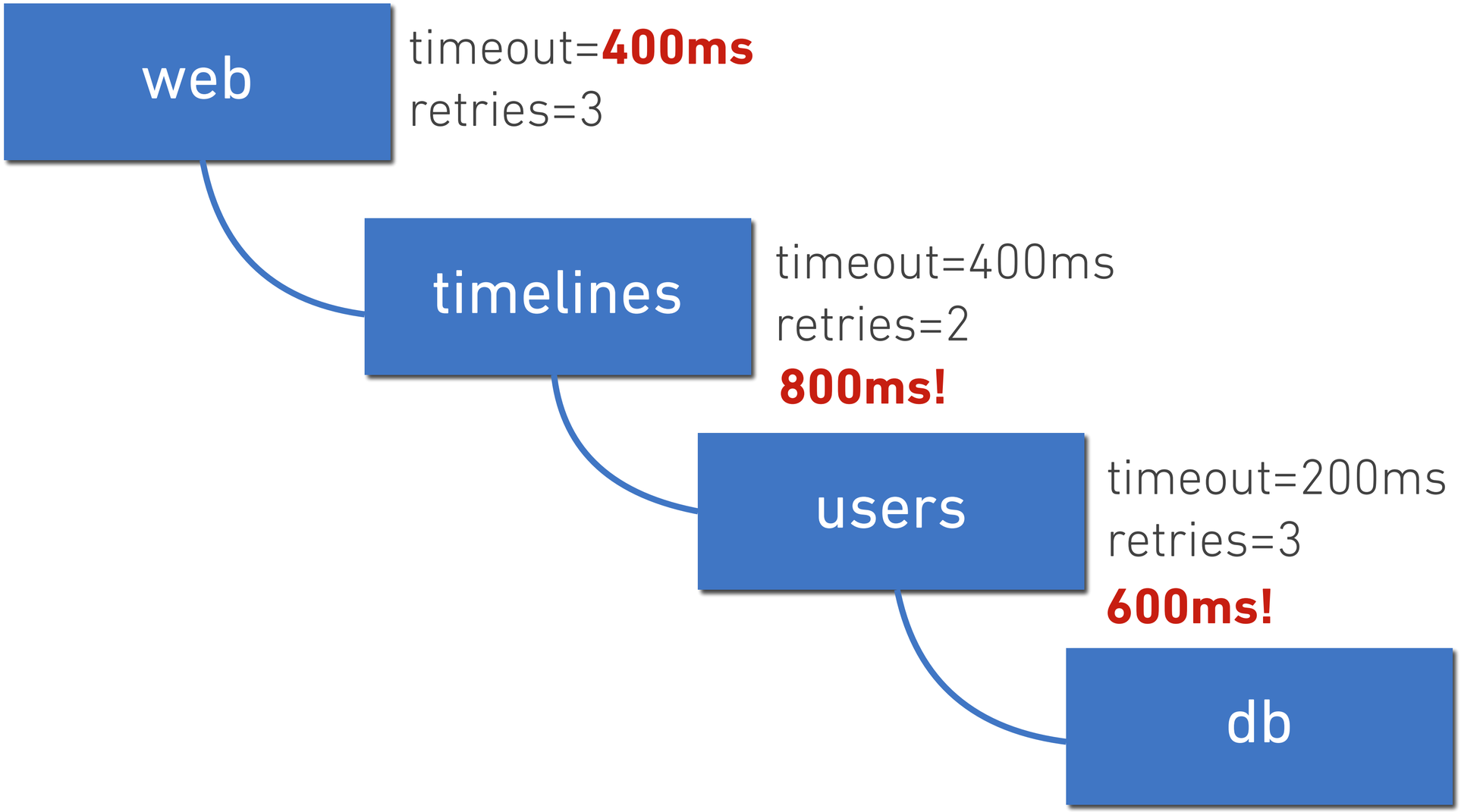
Buoyant

# Some examples

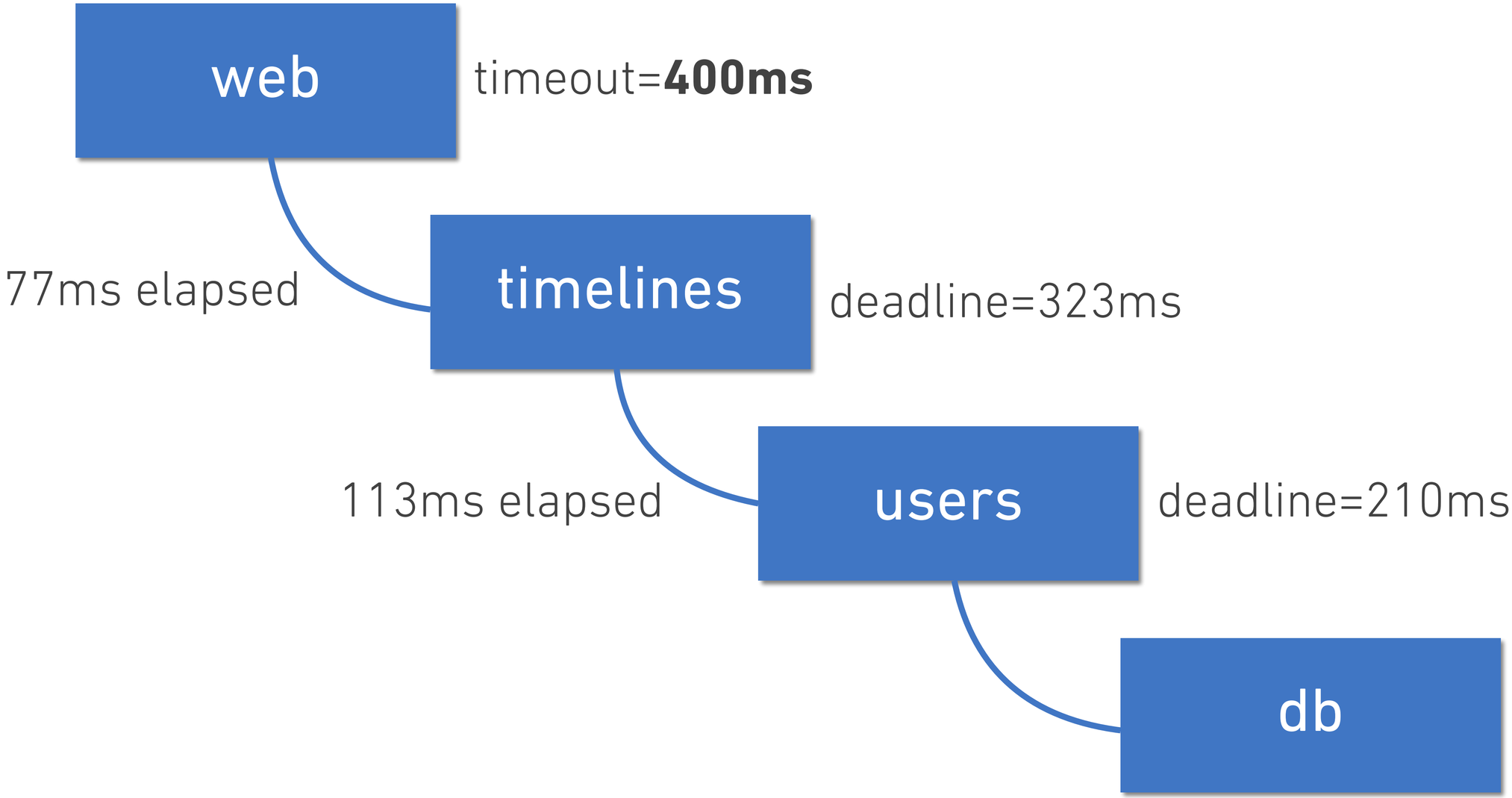
# Timeouts


web
timeout=400ms
retries=3
timelines
timeout=400ms
retries=2
users
timeout=200ms
retries=3
db

# Timeouts


web
timeout=400ms
retries=3
timelines
timeout=400ms
retries=2
800ms!
users
timeout=200ms
retries=3
600ms!
db

# Deadlines


web
timeout=**400ms**
77ms elapsed
timelines
deadline=323ms
113ms elapsed
users
deadline=210ms
db

# Retries

Typical:

retries=3

# Retries

Typical:

retries=3

worst-case: 300% more load!!!

# Budgets

Typical:

retries=3

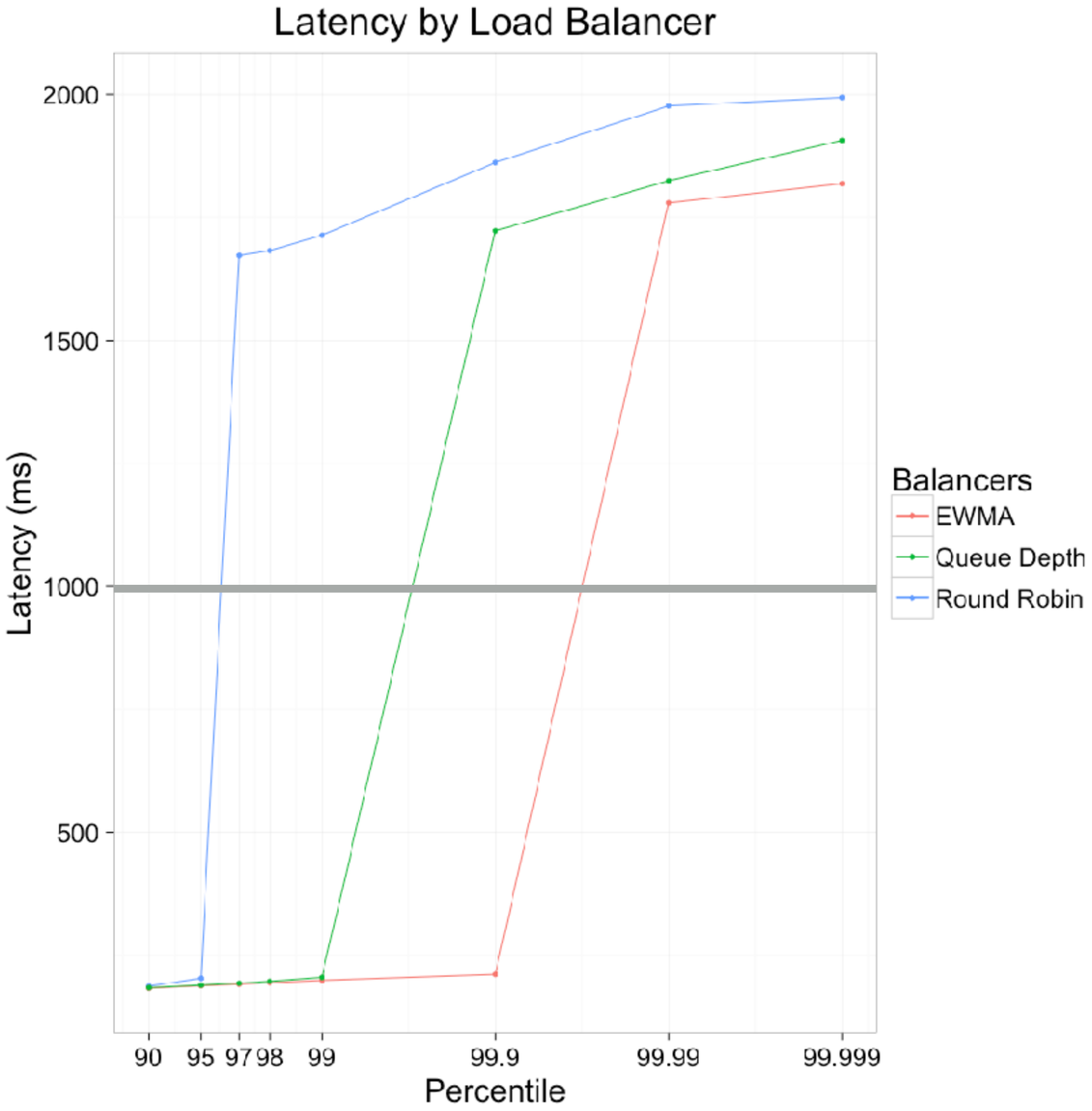
worst-case: 300% more load!!!

Better:

retryBudget=20%

worst-case: 20% more load

# *Request-level* load balancing



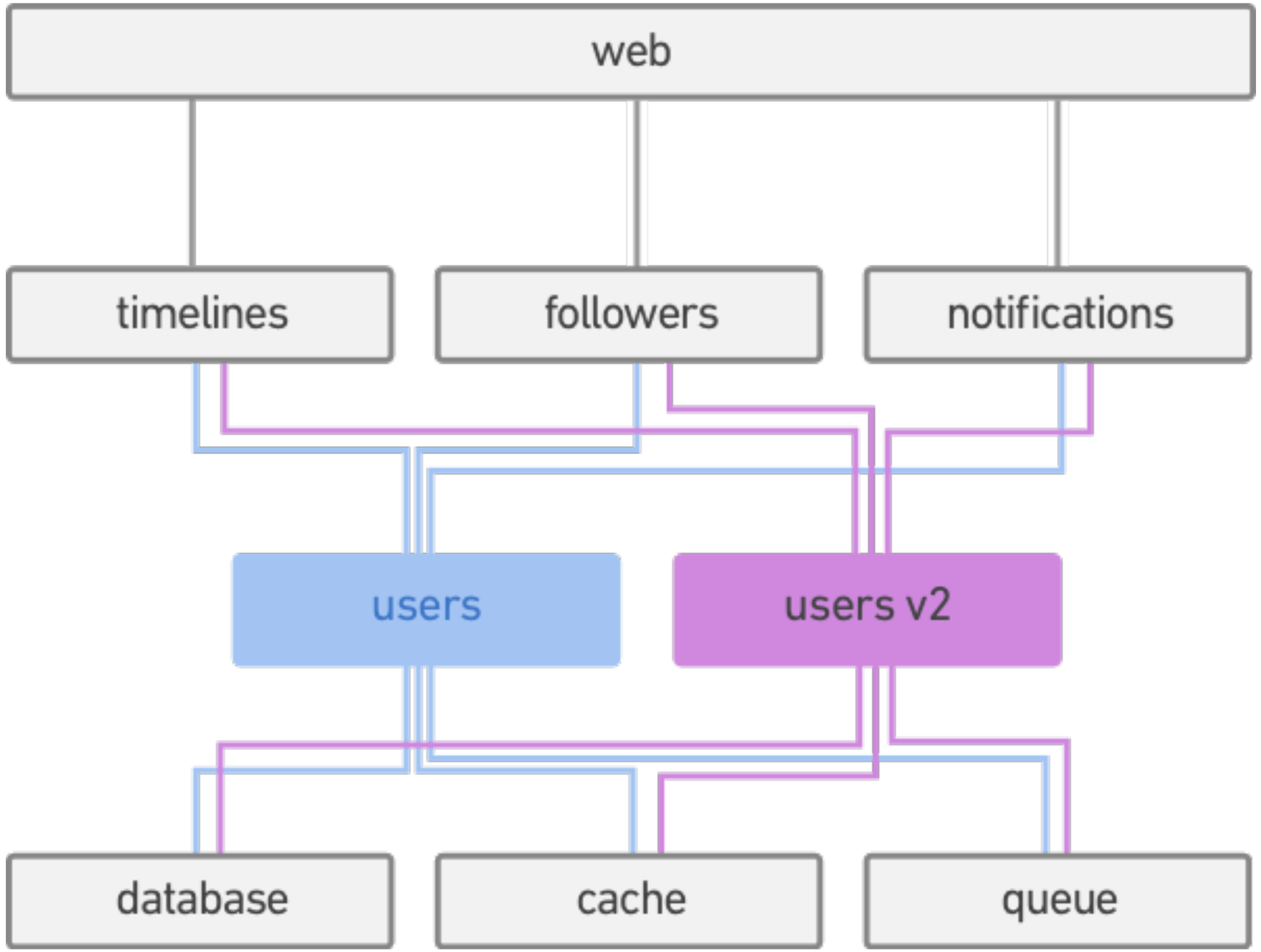

lb algorithms:

- ~~round-robin~~
- ~~fewest connections~~
- queue depth
- exponentially-weighted moving average (EWMA)
- aperture

## 1.1 What Do Linkers and Loaders Do?

The basic job of any linker or loader is simple: It binds more abstract names to more concrete names, which permits programmers to write code using the more abstract names. For example, it takes a name written by a programmer such as `getline` and binds it to "the location 612 bytes from the beginning of the executable code in module `iosys`." Or it may take a more abstract numeric address such as "the location 450 bytes beyond the beginning of the static data for this module" and bind it to a numeric address.

# A linker for your datacenter



# Logical naming

applications refer to *logical names*

```
/svc/users
```

requests are bound to *concrete names*

```
/#/io.l5d.k8s/prod/users
/#/io.l5d.k8s/staging/users
```

mapping from logical to concrete is *routing*

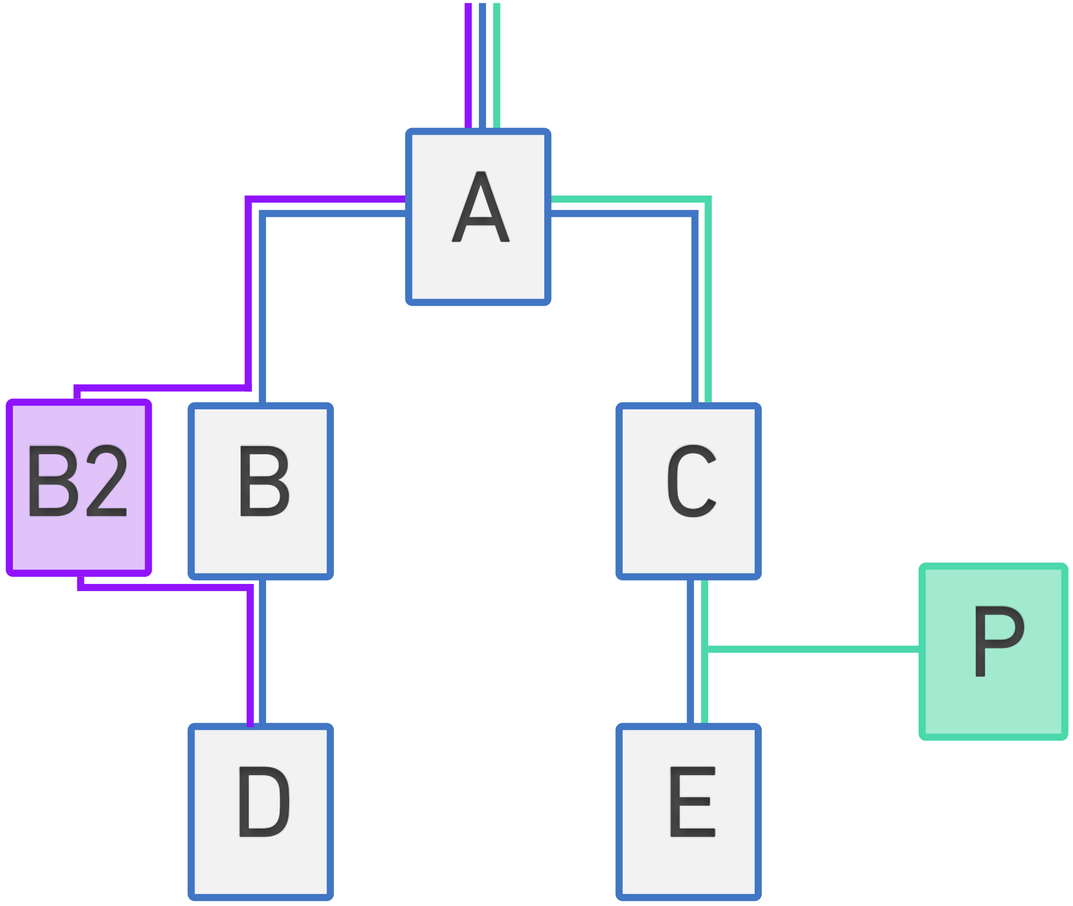
```
/svc => /#/io.l5d.k8s/prod
```

# Per-request routing: staging

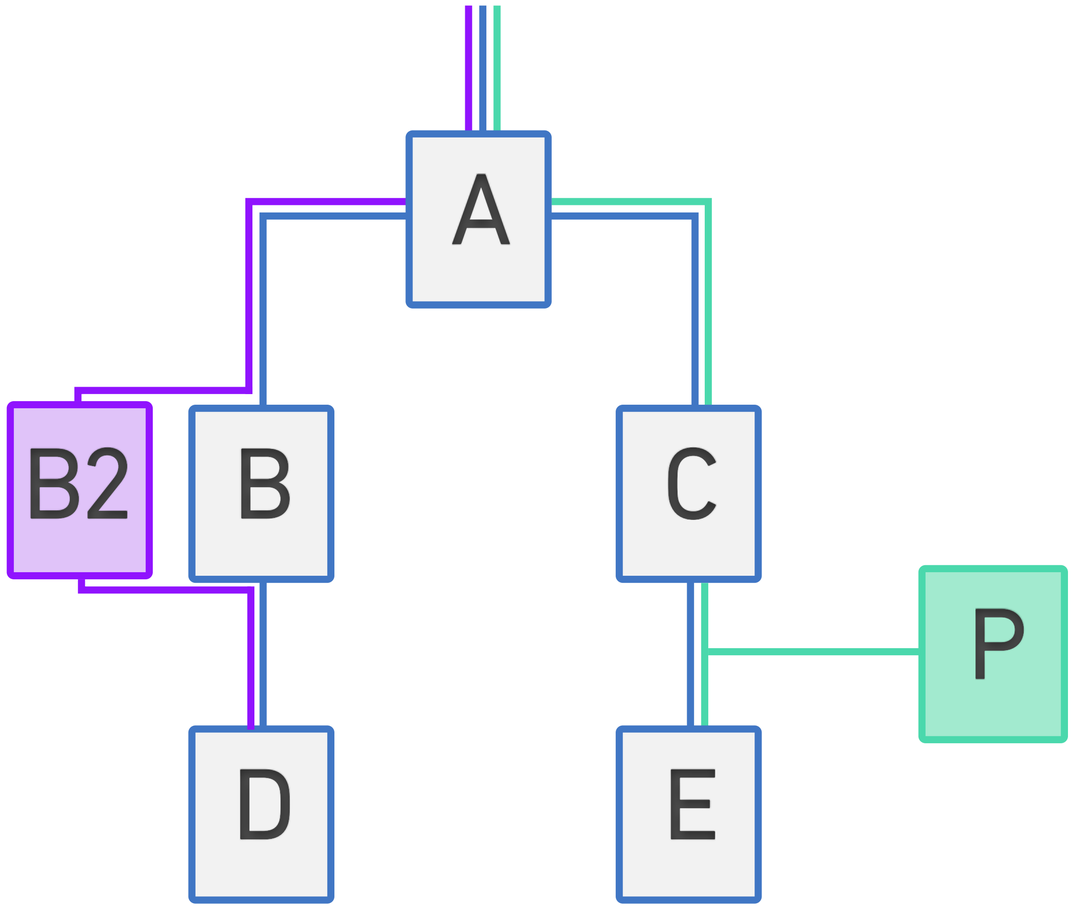
GET / HTTP/1.1

Host: mysite.com

l5d-dtab: /svc/B => /svc/B2



# Per-request routing: debug proxy




GET / HTTP/1.1

Host: mysite.com

l5d-dtab: /svc/E => /svc/P/svc/E

linkerd

# Thank you!

## Demo time

William Morgan ~ william@buoyant.io