

Go microservices

QCon SF · 10 November 2016

Perhaps better titled

How to do microservices

With Go and Go kit as an implementation detail

Outline

1. Microservice definitions and contextualization
2. Go and Go kit
3. A Go kit microservice from first principles

Follow along

- Install Go – <http://golang.org> – brew install go
- \$ export GOPATH=\$HOME/gocode # or somewhere else
- \$ go get github.com/peterbourgon/go-microservices

Who am I?

@peterbourgon · peter.bourgon.org

Who are you?

The microservices landscape

Toward a shared context

Size

A **single programmer** can design, implement,
deploy and maintain a microservice.

—Fred George

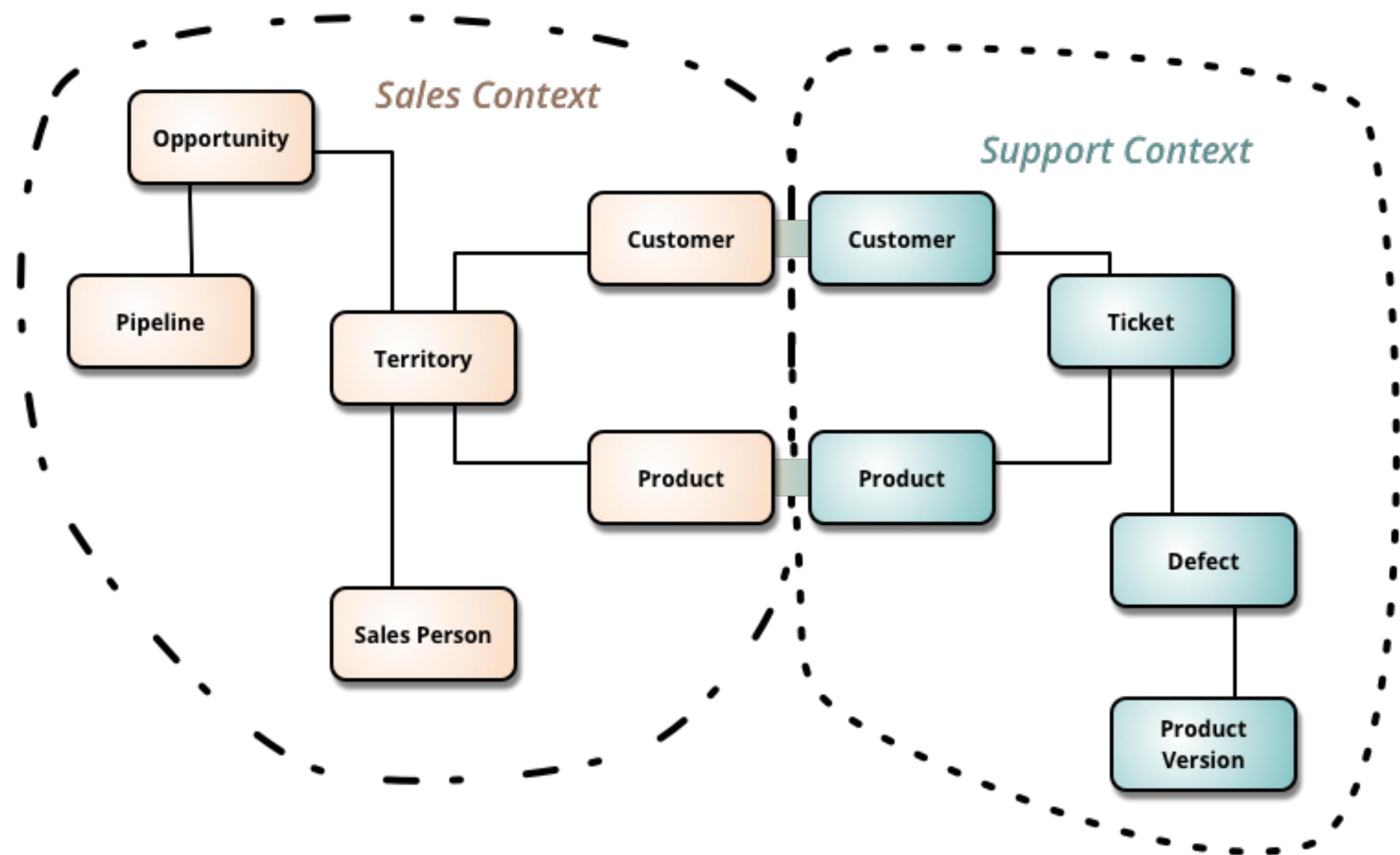
Software that **fits in your head**.

—Dan North

Data

A microservice implements a single
Bounded Context (from DDD)
—*Martin Fowler, Sam Newman*

A single logical **database per service**.
—*Chris Richardson*



Operation

Microservices **built & deployed independently.**

Stateless, with state as backing services.

— *12Factor.net*

Addressable through a **service discovery** system.

— *Chris Richardson*

Architecture

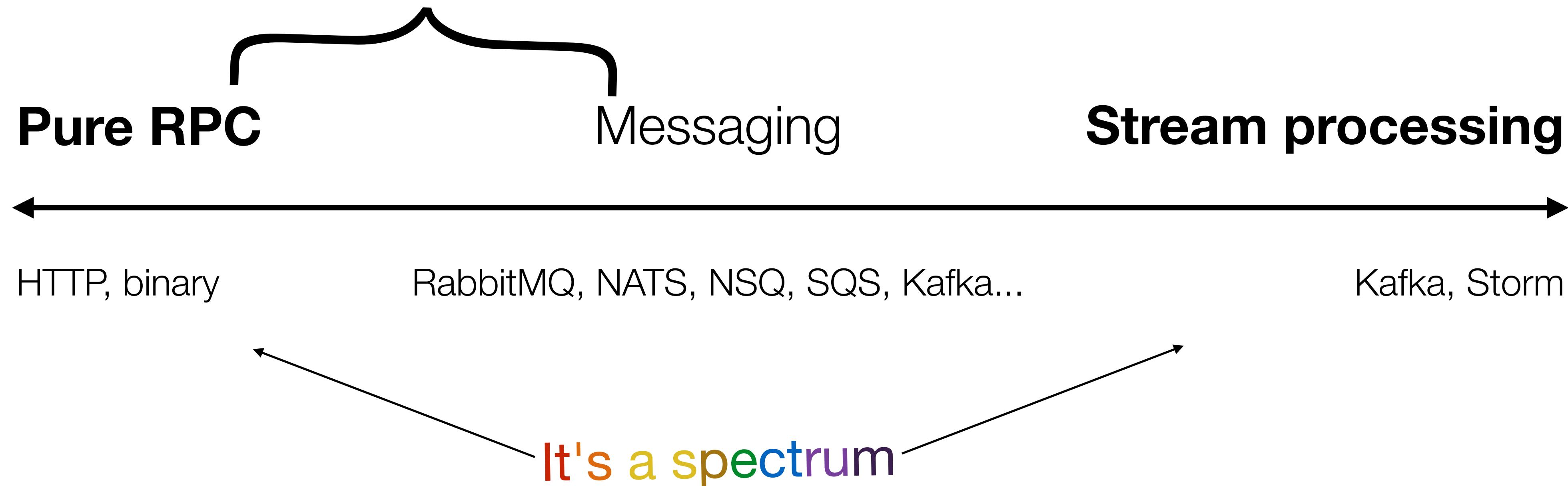
- CRUD-oriented
- Typically RPC, often HTTP
- Request processing
- Monolith → microservices
- Ruby on Rails; Tomcat/Jetty, Spring Boot; Akka, Play

Did you mean...

- Stream-oriented
- Event sourcing
- Message processing
- Materialized views
- SQS, Kinesis, Kafka, RabbitMQ, Storm...



Intermediate architectures



Microservices solve
organizational problems



Microservices cause
technical problems

Problems solved

- Team is too large to work effectively on shared codebase
- Teams are blocked on other teams – can't make progress
- Communication overhead too large
- Velocity stalled

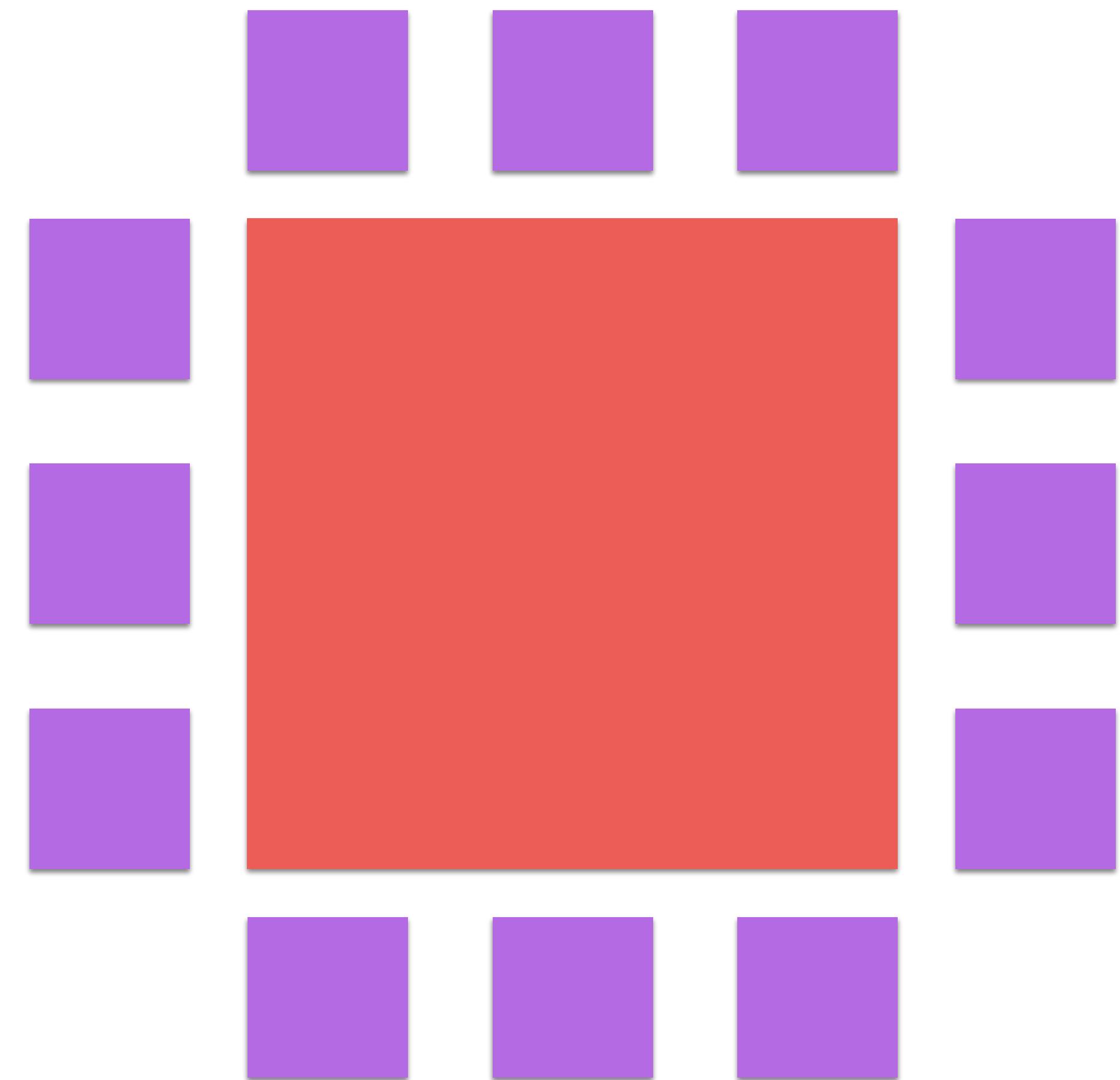
Problems caused

- Need well-defined business domains for stable APIs
- No more shared DB – distributed transactions?
- Testing becomes *really hard*
- Require dev/ops culture: devs deploy & operate their work
- Job (service) scheduling – manually works, for a while...

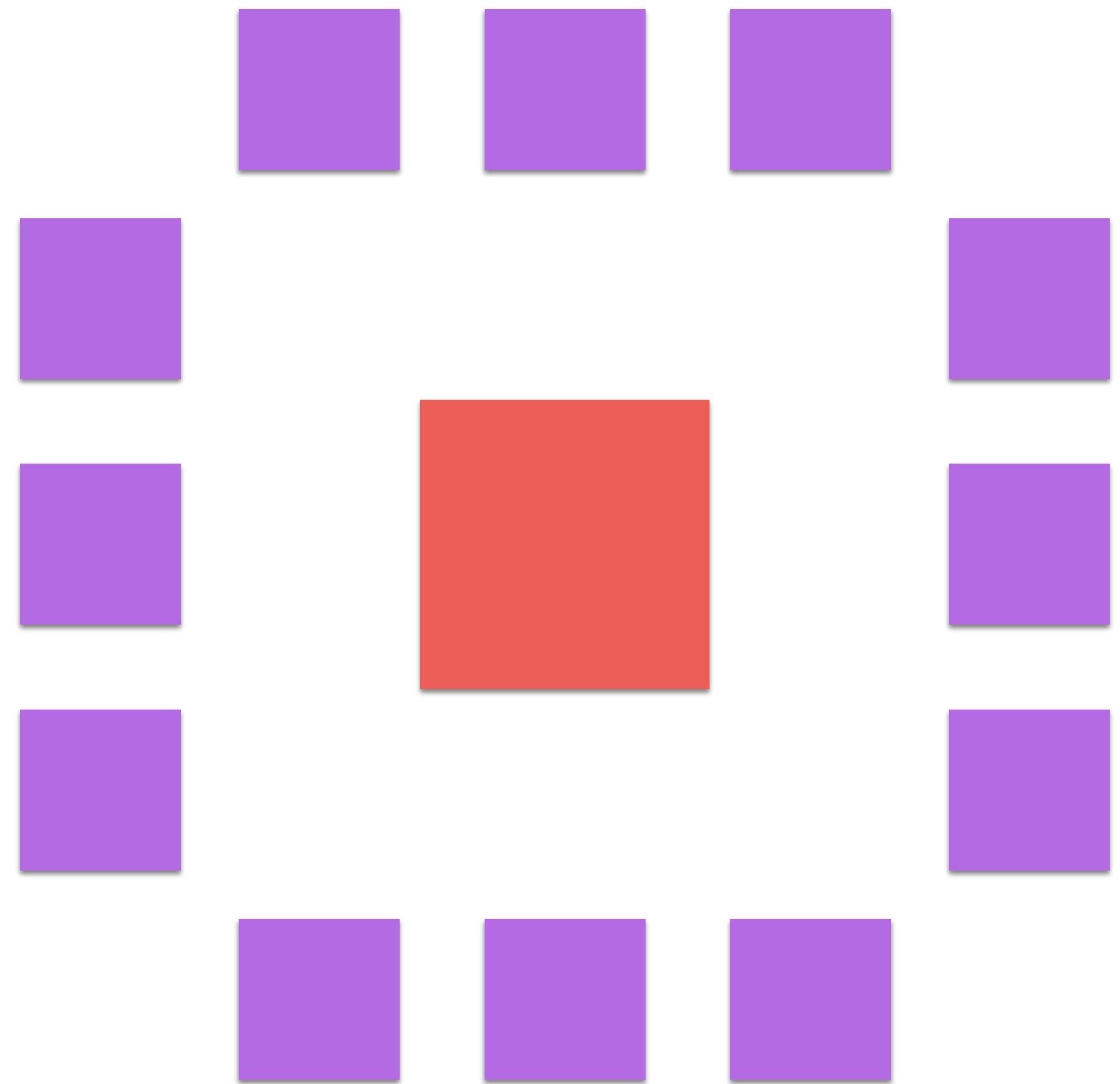
Problems caused

- Addressability i.e. service discovery
- Monitoring and instrumentation – tail -f? Nagios & New Relic? Ha!
- Distributed tracing?
- Build pipelines??
- Security???

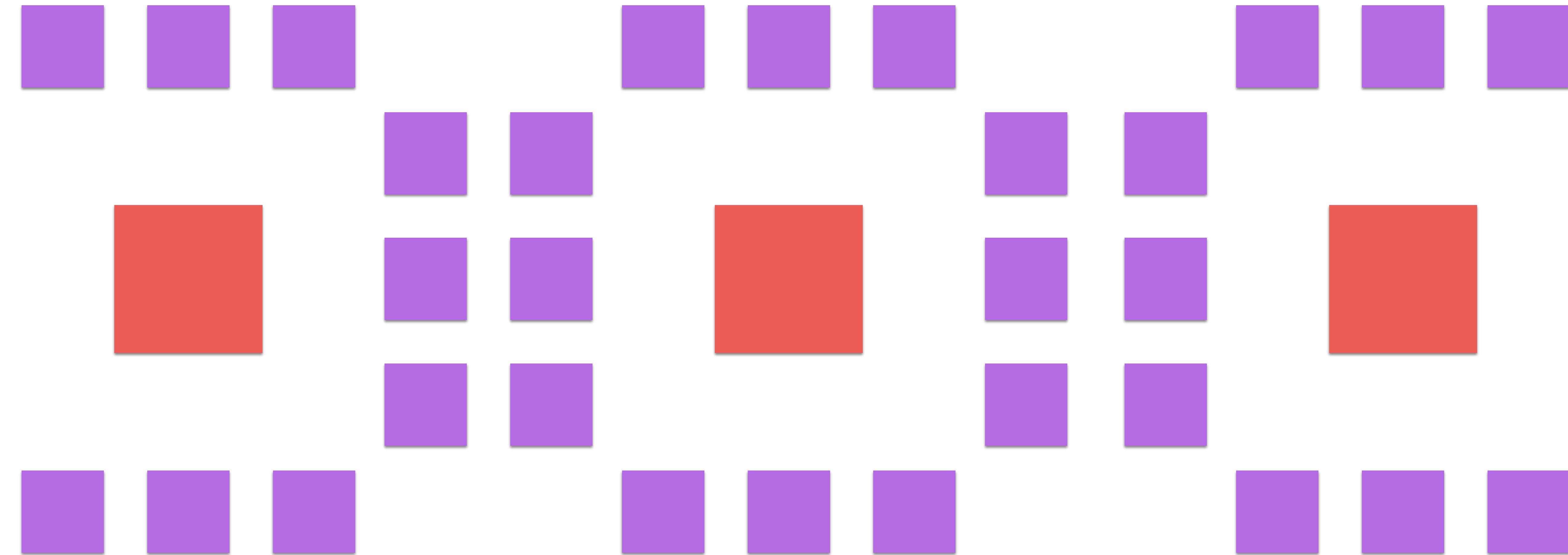
Monolith



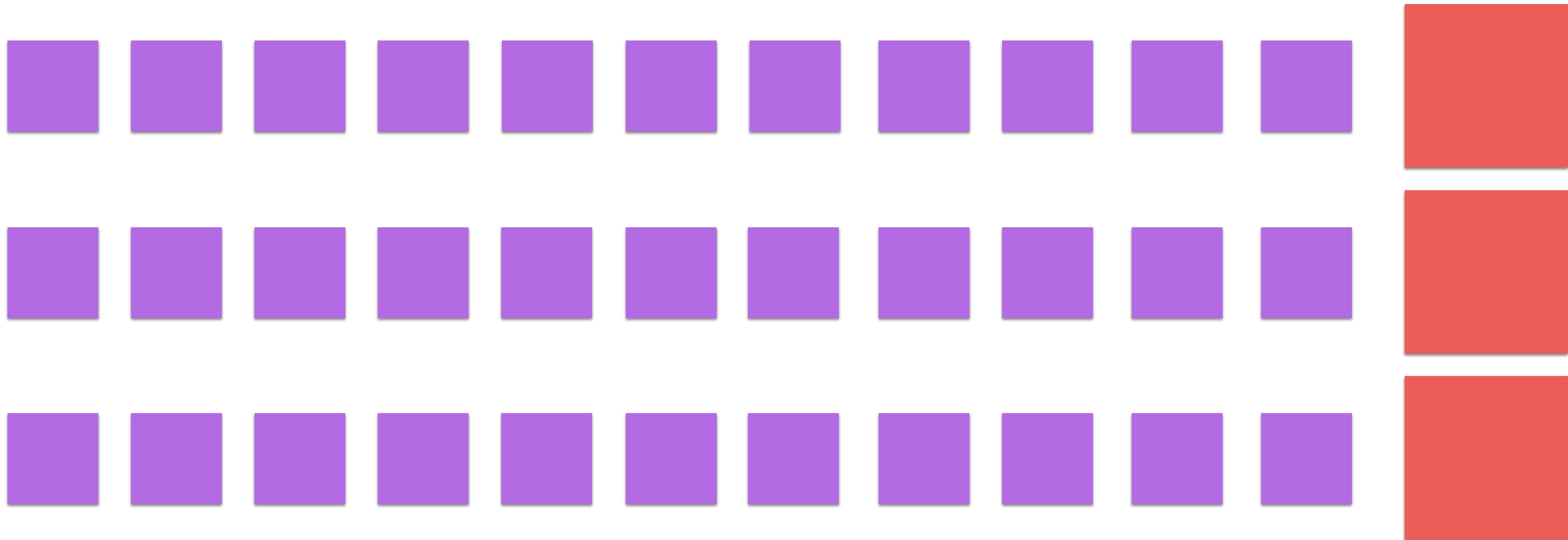
Microservice



Microservices



Microservices



From one to many

—

Service Name, Programming language(s), Programming paradigm(s), Architectural choices, Integration pattern(s), Transport protocols, Authentication, Authorization, Reporting, ETIs, Databases, Caching, Platform libraries, Service dependencies, CI Pipeline dependencies, 3rd party library dependencies, 3rd party service dependencies, Security threat model, License audit, Compliance audit, Capacity plan, Provisioning plan, Cost reporting plan, Monitoring plan, Maintenance process, Backup and Restore process, Secret management, Secret rotation, On-Call schedule, Configuration management, Workflow management, Alerts, Log aggregation, Unhandled failure aggregation, Operations and Incident response runbooks, API documentation, Source Code Repository, Humane Service Registry, Service Discovery Registry, Distributed Tracing Registry, Monitoring Dashboard Registry, Build Artifact Registry, CI pipeline(s): Build, Test, Publish, Integration tests, Contract tests, Canary, Deploy, Post deploy tests

Think twice

- Most [small] organizations don't need microservices
- 5 or fewer engineers? You *definitely* don't need microservices
- Building an AMI for an EC2 autoscaling group works *really really* well

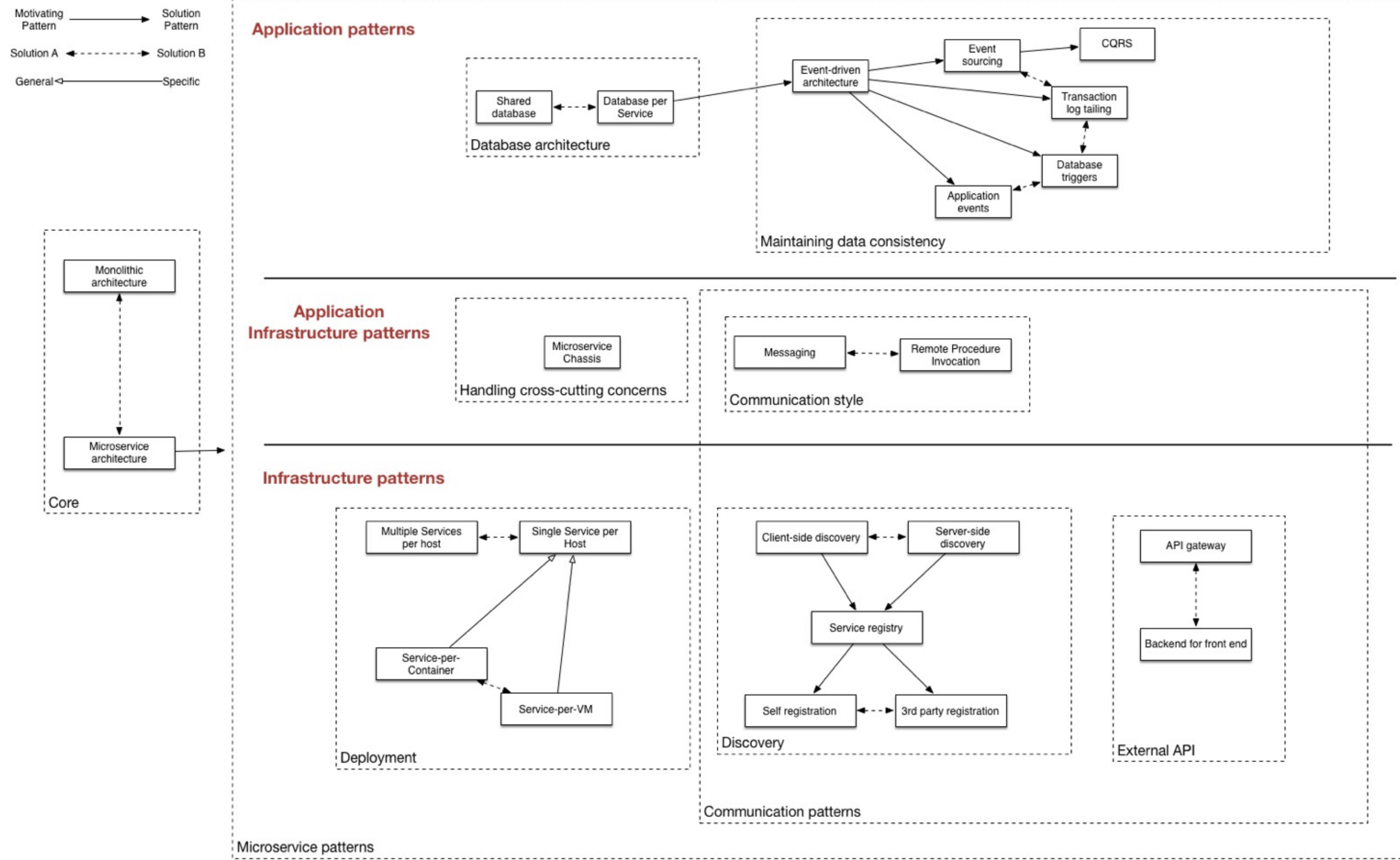
I ❤️ microservices

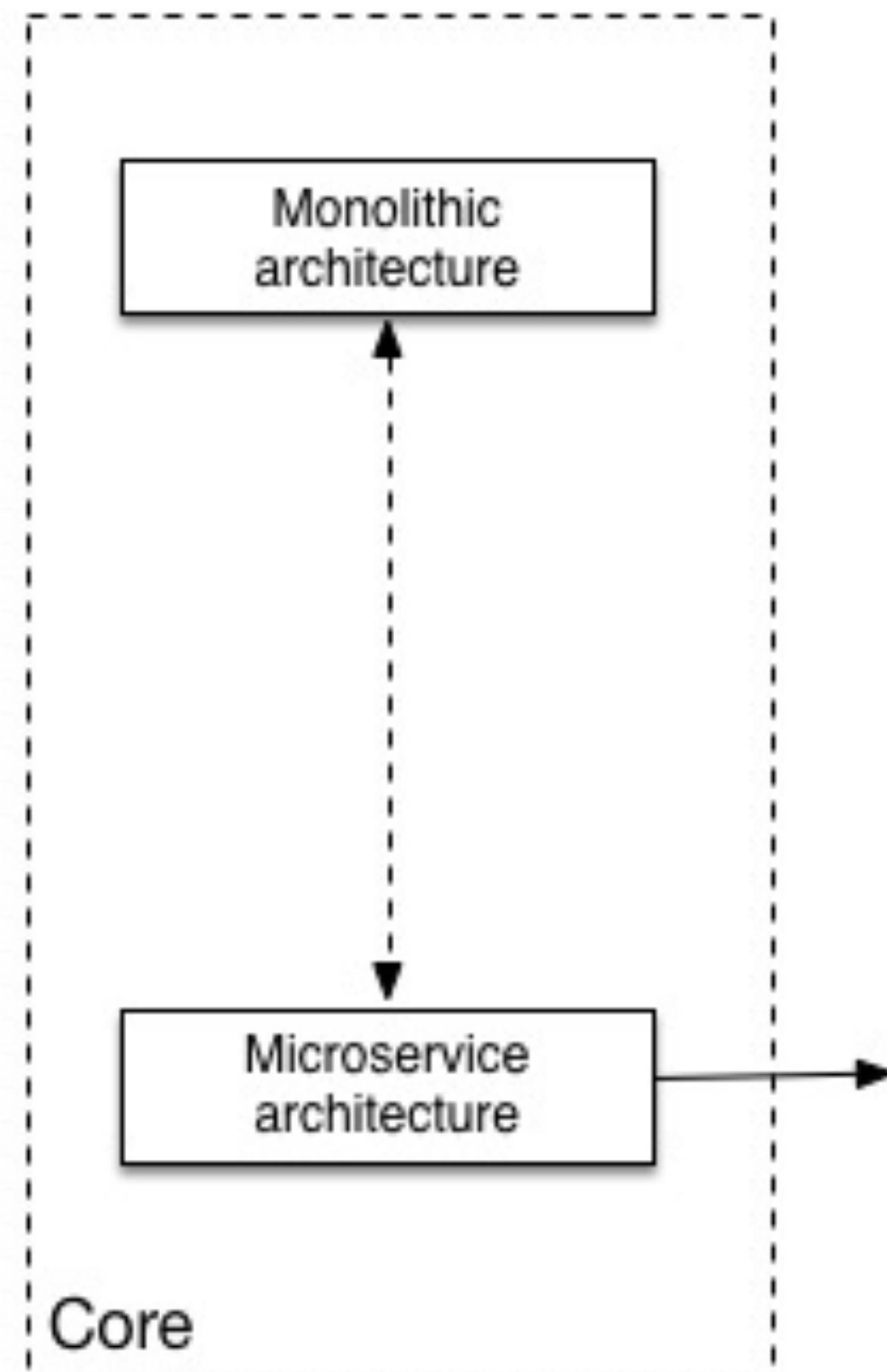
- Sense of ownership & responsibility → quality
- Confidence → Always Be Refactoring
- Align incentives with the business
- peter.bourgon.org/a-case-for-microservices

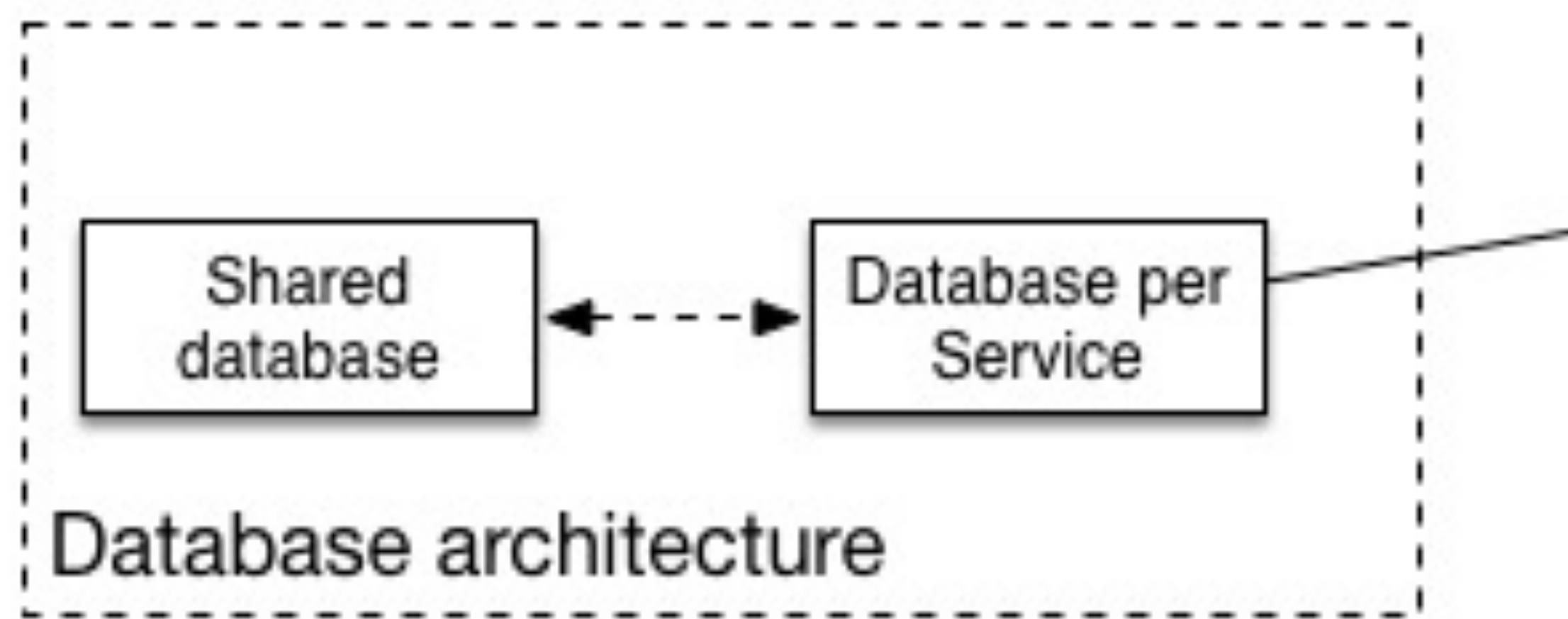
What's a pattern?

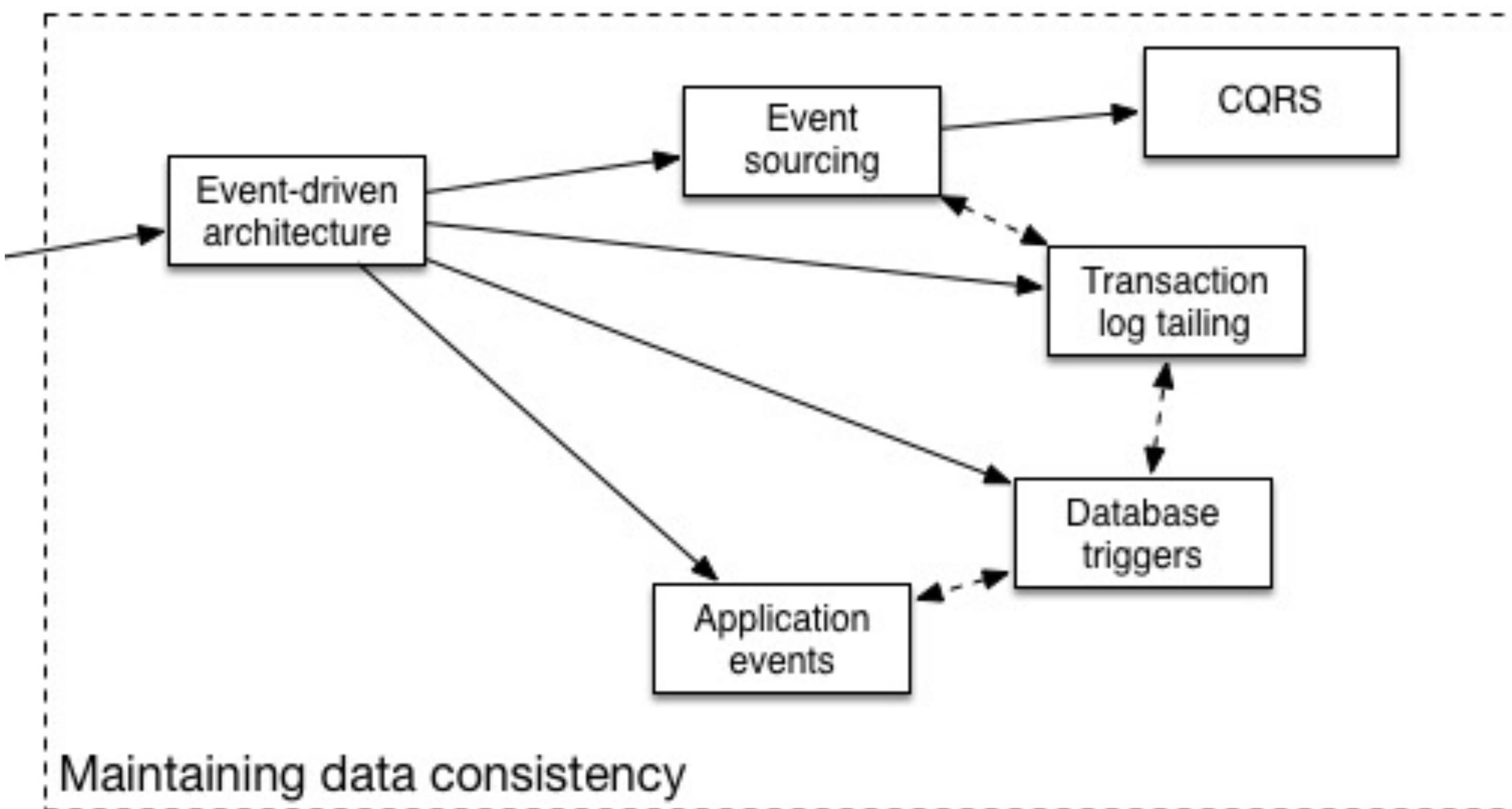


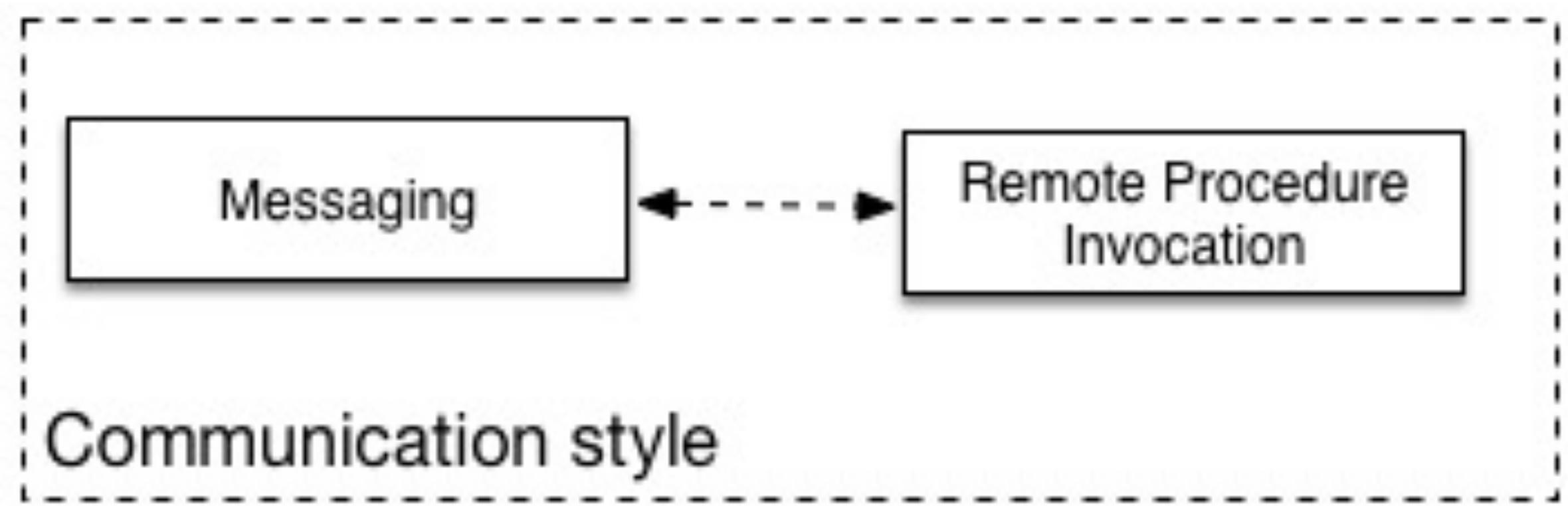
Reusable **solution**
to a **problem**
occurring
in a particular **context**

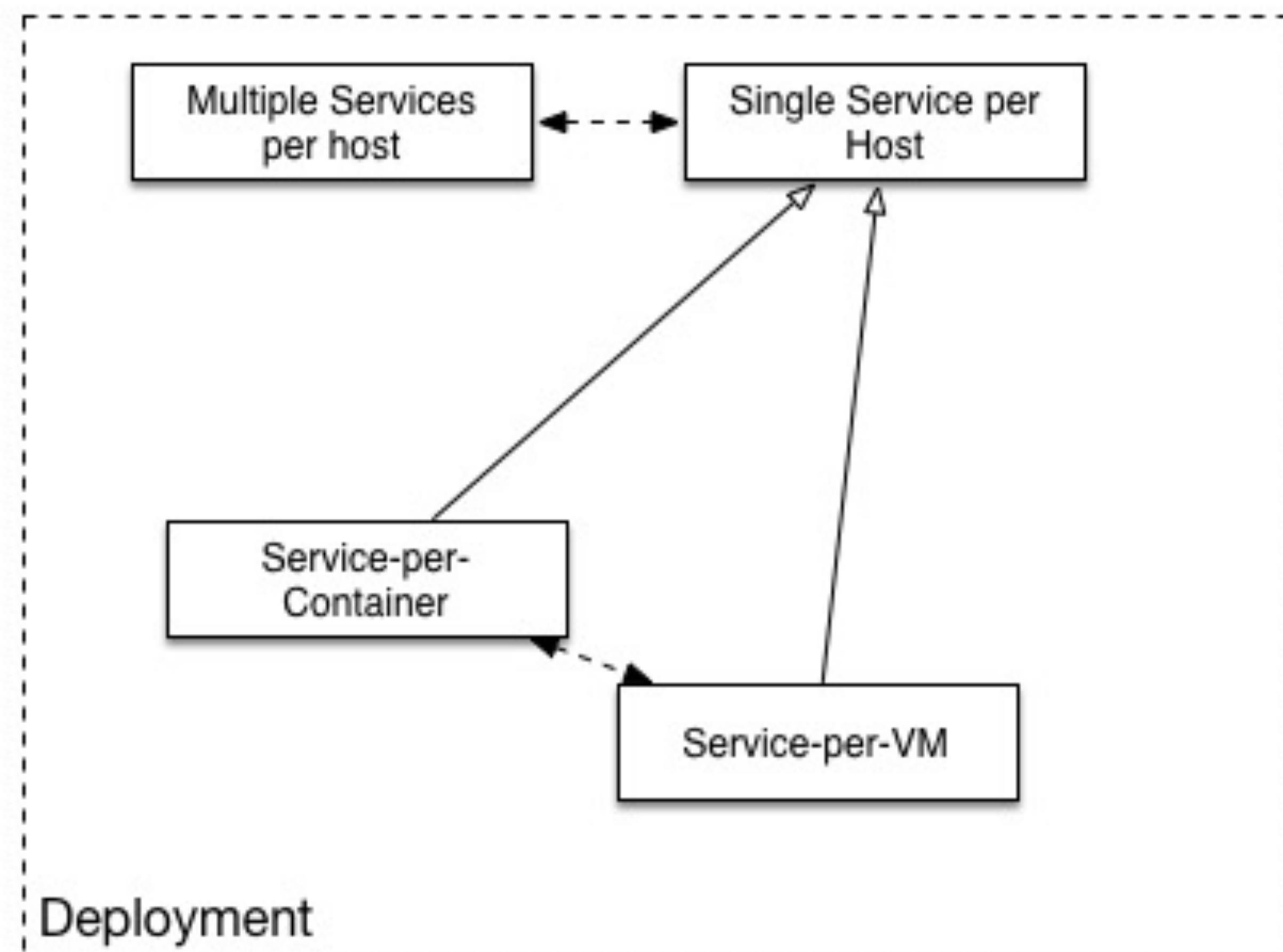


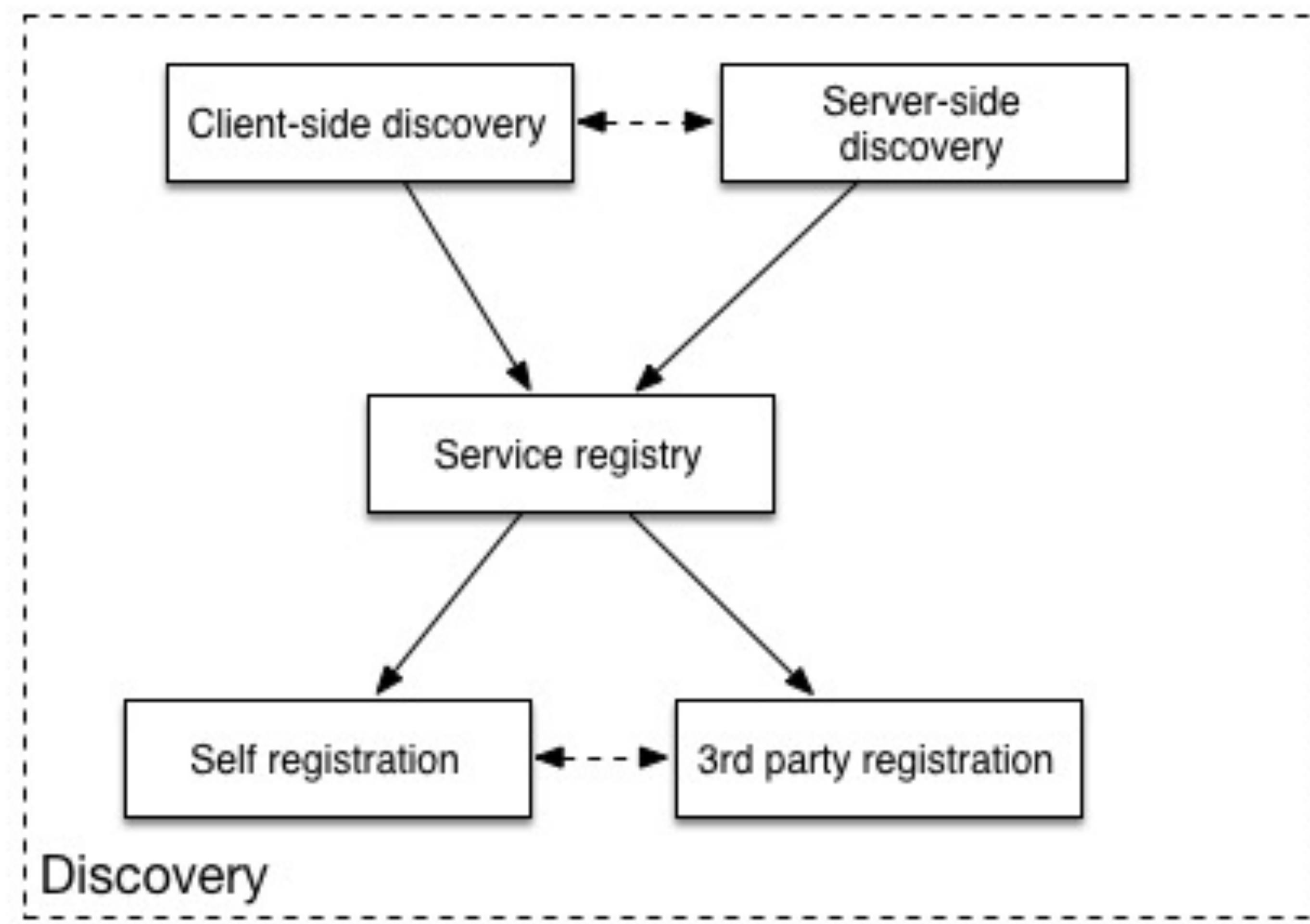


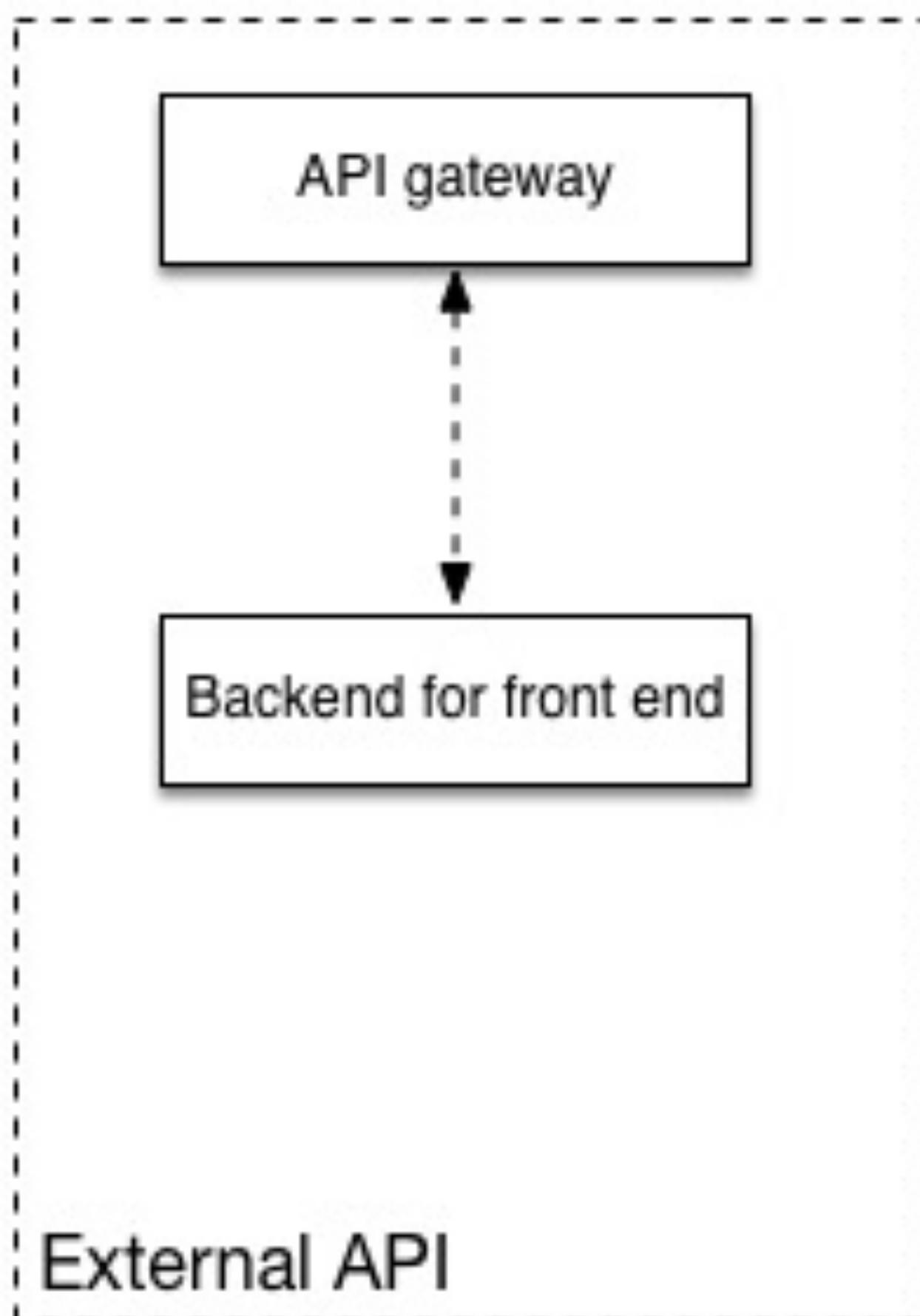


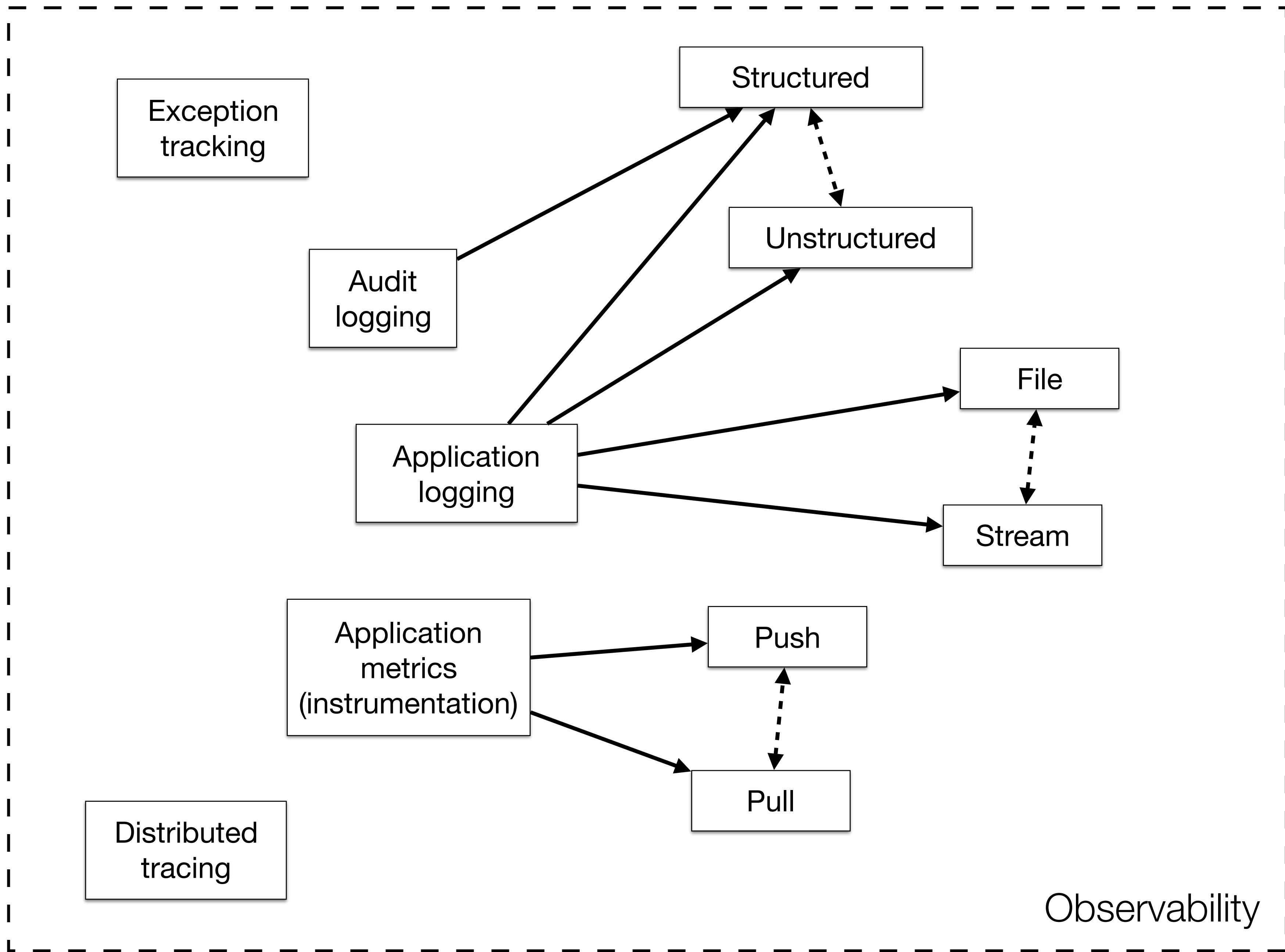












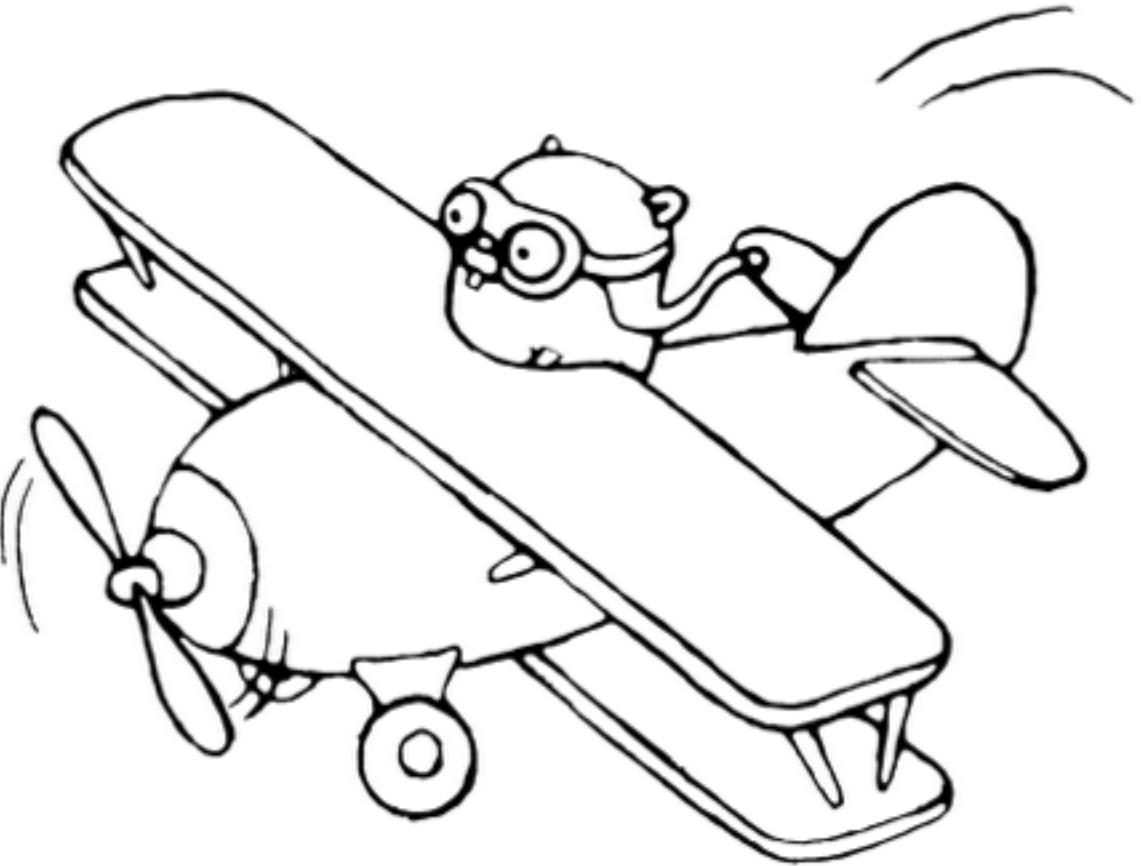
Why Go?

Facts about Go

- Statically typed
- Compiled – *fast!*
- Native binaries
- Garbage collected
- Looks like C
- Big standard library
- Baked-in concurrency
- Native networking

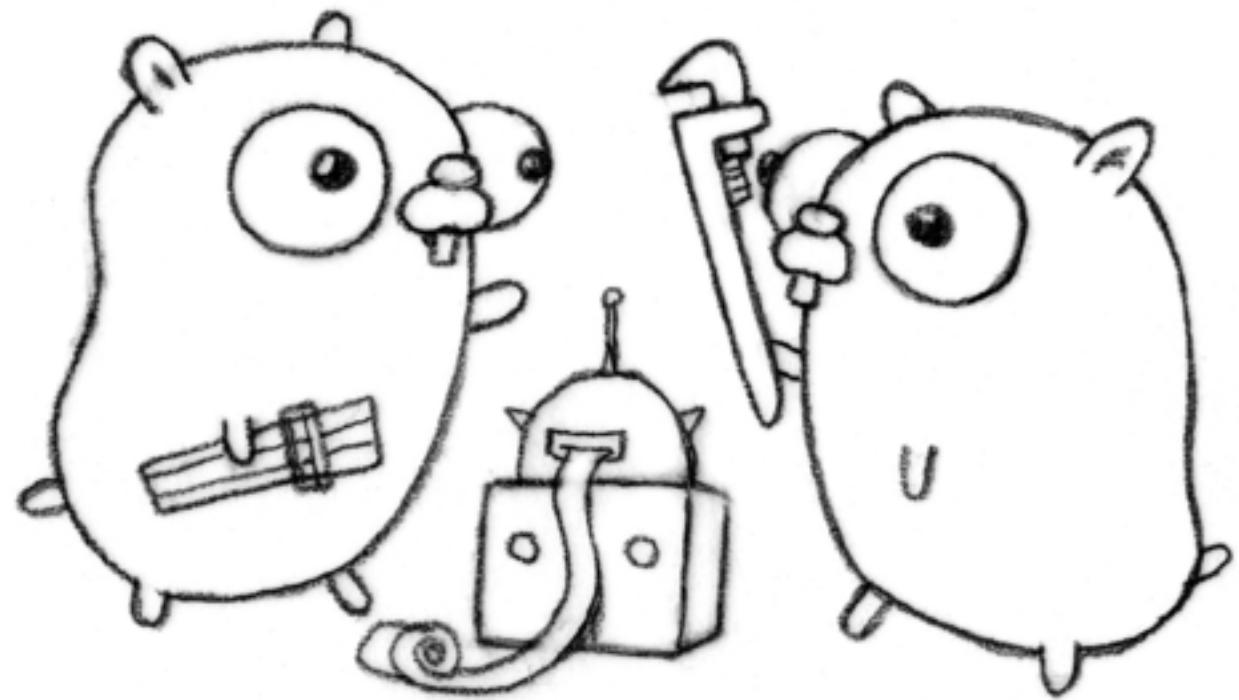
Go, IMO

- A breath of fresh air from "kitchen sink" languages (cf. Scala)
- Simple, orthogonal features that aren't surprising (cf. Node)
- Efficient by default (cf. Python, Ruby)
- Predictable runtime behavior, fast lifecycles (cf. all JVM languages)
- Familiar heritage, syntax, and paradigm (cf. Haskell, Elixir)



Exercise

Go language introduction/refresher



Go for servers

- Often the best language for writing servers
- Why not the best language for writing microservices?

Enter Go kit

Concerns + patterns

Toward some kind of software engineering

Transport

Service registration

Load balancing

Business logic

Metrics

Circuit breaking

Service discovery

Rate limiting

Logging

Distributed tracing

Transport
Rate limiting
Circuit breaking

Business logic

Metrics
Logging
Distributed tracing

Service registration
Service discovery
Load balancing

- Transport

- Rate limiting

- Circuit breaking

Business logic

- Metrics

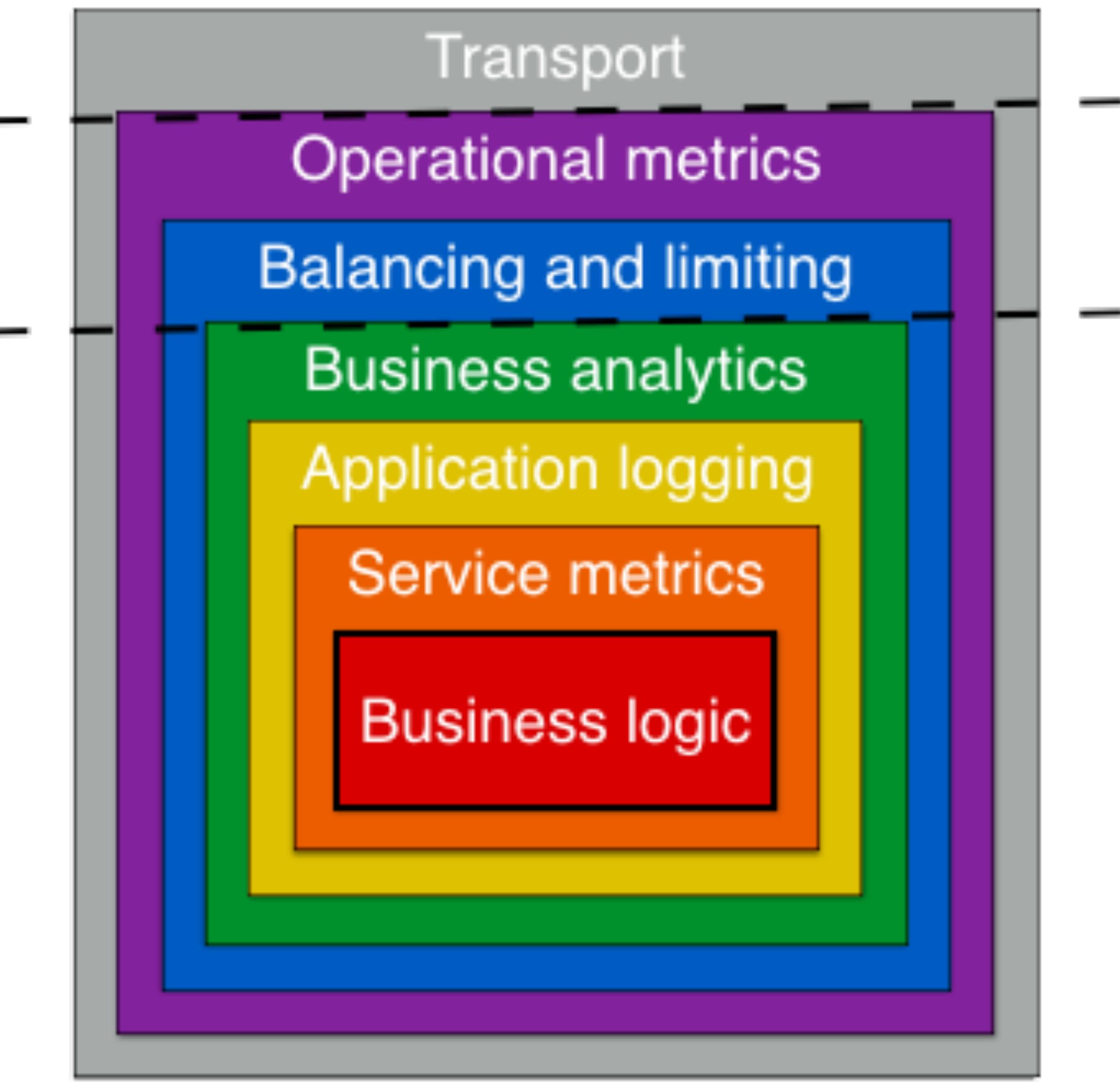
- Logging

- Distributed tracing

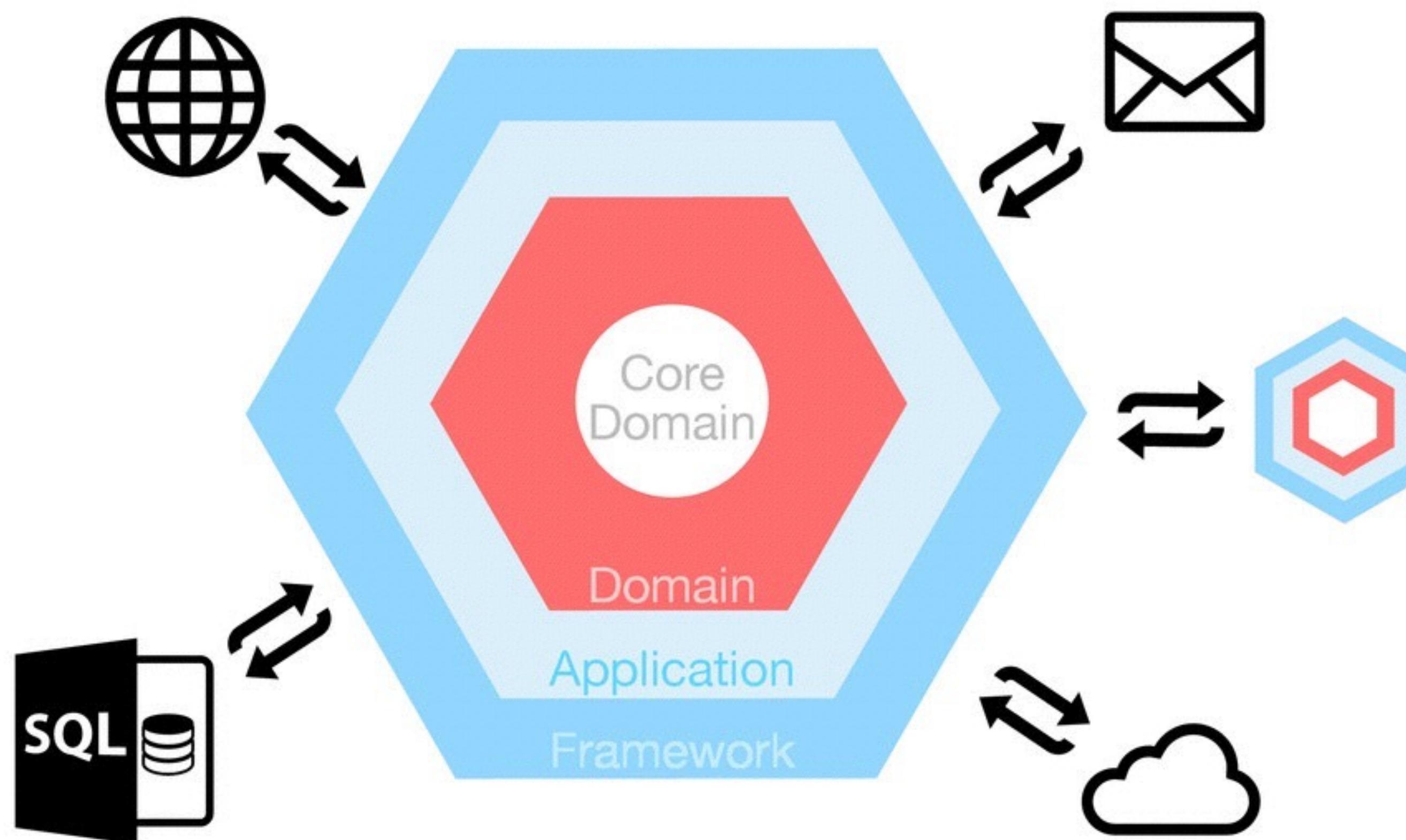
- Service registration

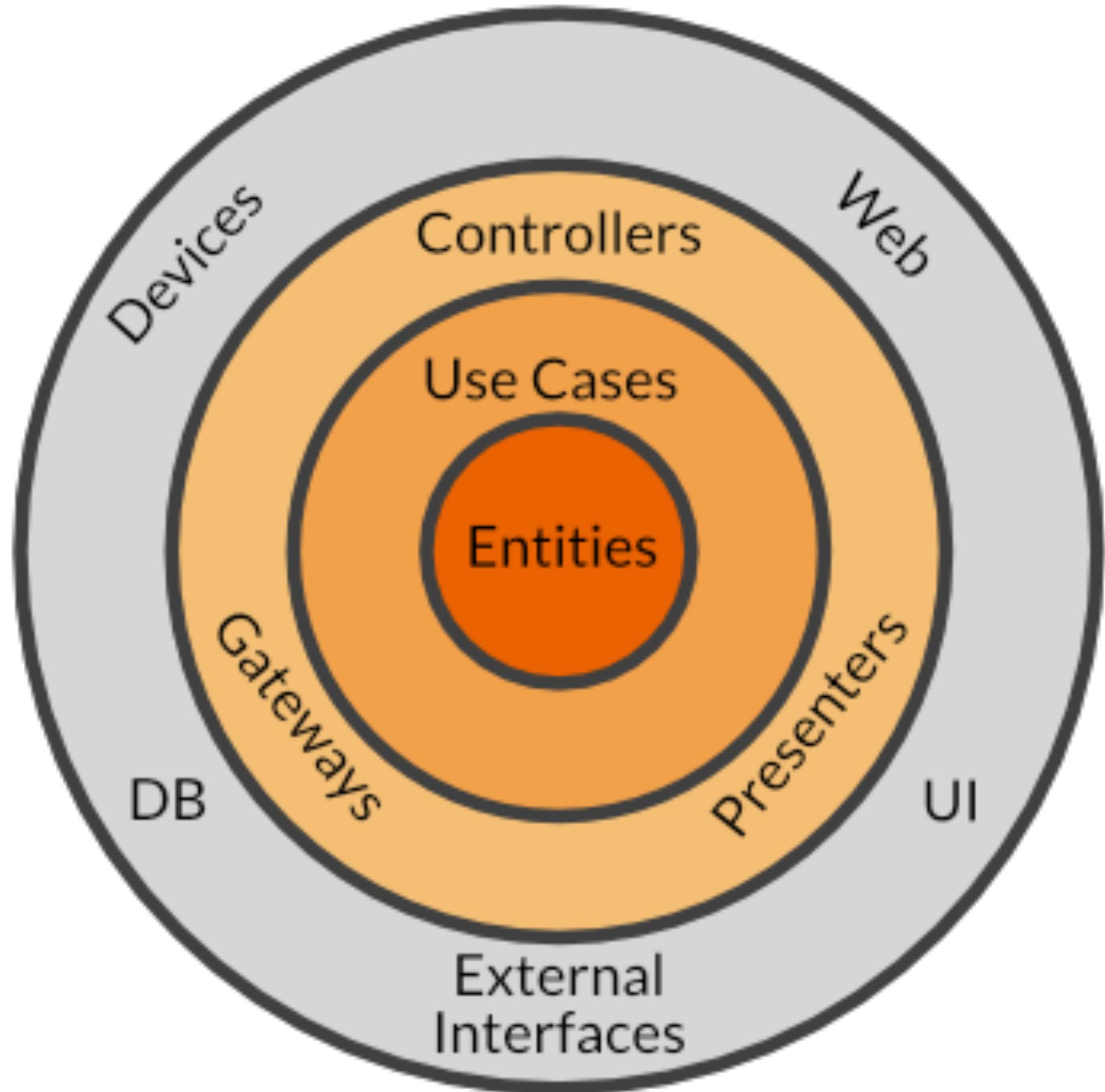
- Service discovery

- Load balancing



The Hexagon





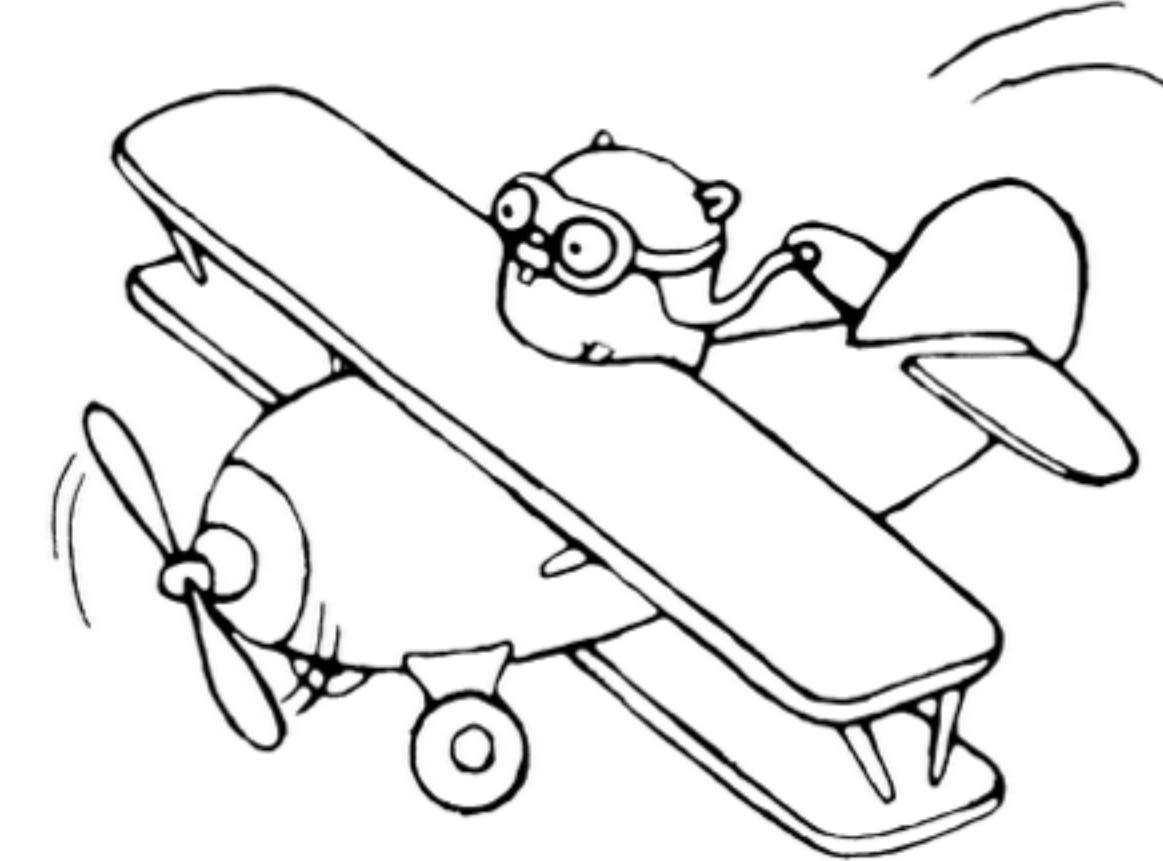
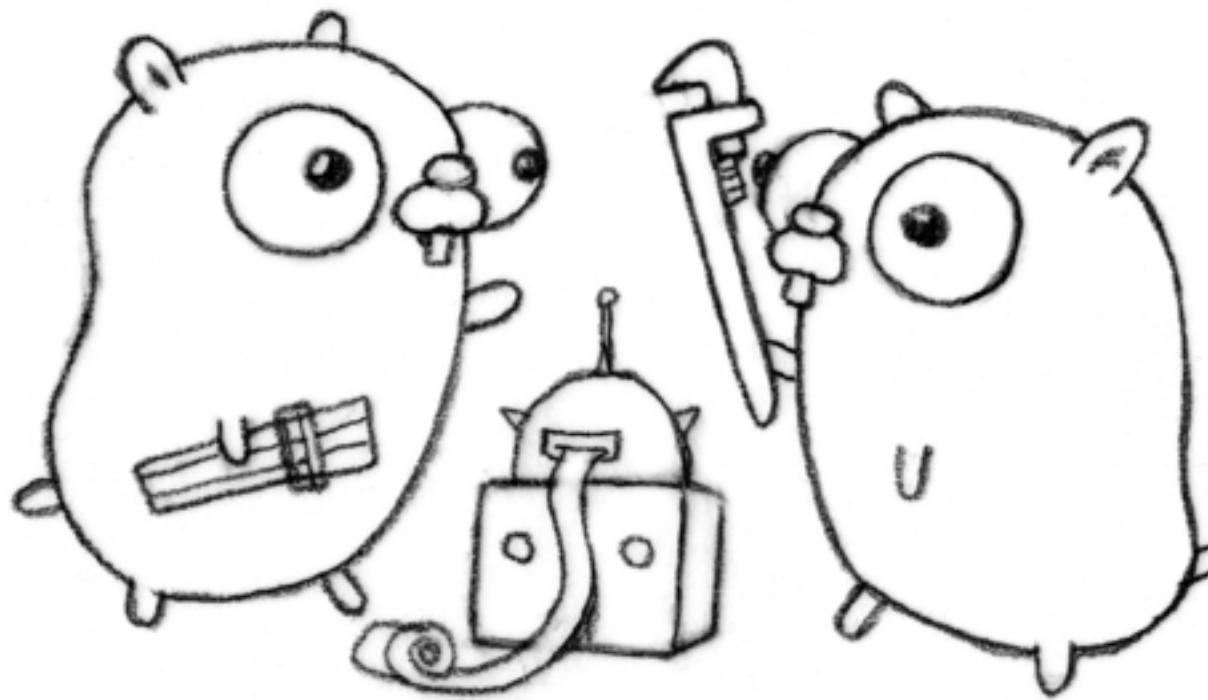
The central rule of The Clean Architecture is
the Dependency Rule, which says

**Source code dependencies
can only point inwards.**

Exercise

Given billingsvc, place all of these concerns in our model:

- Customer ID
- HTTP request decoding
- Max 1 QPS per origin IP
- Log each account credit
- Log each account debit
- Log each error
- Track overall credit rate
- Track overall debit rate
- Make service reachable by frontend
- Spread load over different services
- Stop cascading failure with circuit breaker
- Trace requests with correlation ID
- Monitor request durations
- Monitor error rates
- Dispute ID



Go kit: the pitch

- Make microservice concerns tractable
- Make Go attractive to your organization
- Play nicely with others

Go kit: not a framework

- Not like other Go projects: Revel, Beego, Kite, Micro, H2, gocircuit...
- More like Gorilla
- Use what you need
- Progressive enhancement

Go kit: compare to...

- Finagle (Scala) – initial inspiration
- Netflix OSS: Eureka, Hystrix, Zuul, etc. (JVM) – similar goals
- Spring Boot (Java) – similar goals, *radically* different approach
- Nameko (Python) – similar goals
- Others?

Go kit: philosophy

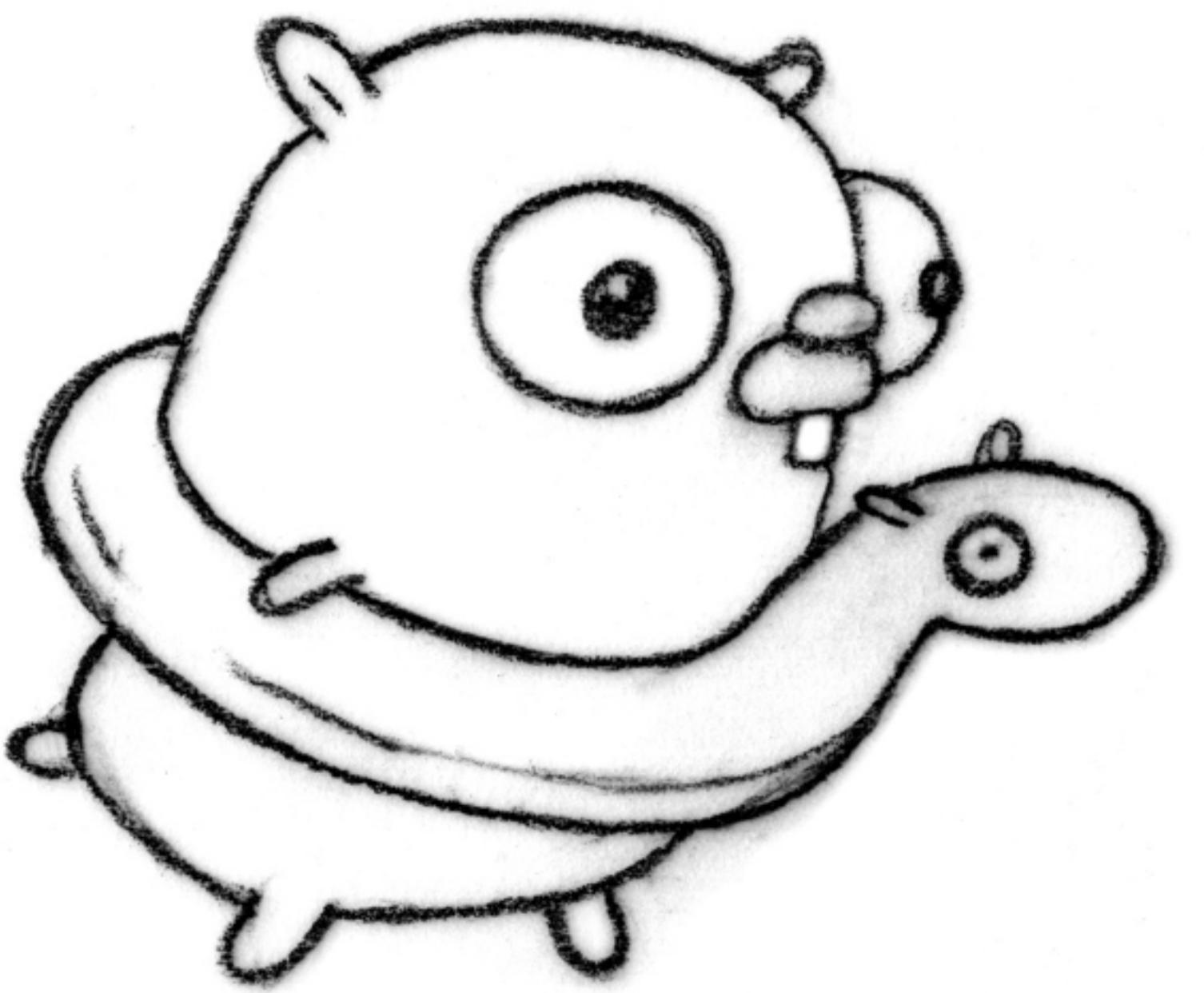
- Exemplify Go best practices
 - No global state
 - Declarative composition
 - Explicit dependencies
 - Interfaces as contracts
- Toward a software engineering
 - SOLID Design
 - Domain Driven Design
 - The Clean Architecture
 - Hexagonal Architecture

A photograph of a traditional Japanese rock garden. In the foreground, a large, light-colored gravel area is raked into fine, parallel lines. A paved walkway leads from the bottom left towards a white-walled building with a dark tiled roof in the background. To the left, there's a low wall made of stacked stones, and behind it, several large, rounded green bushes. On the right, a large, prominent grey rock stands next to a smaller, curved green bush. The overall scene is peaceful and minimalist.

My little zen rock garden

addsvc

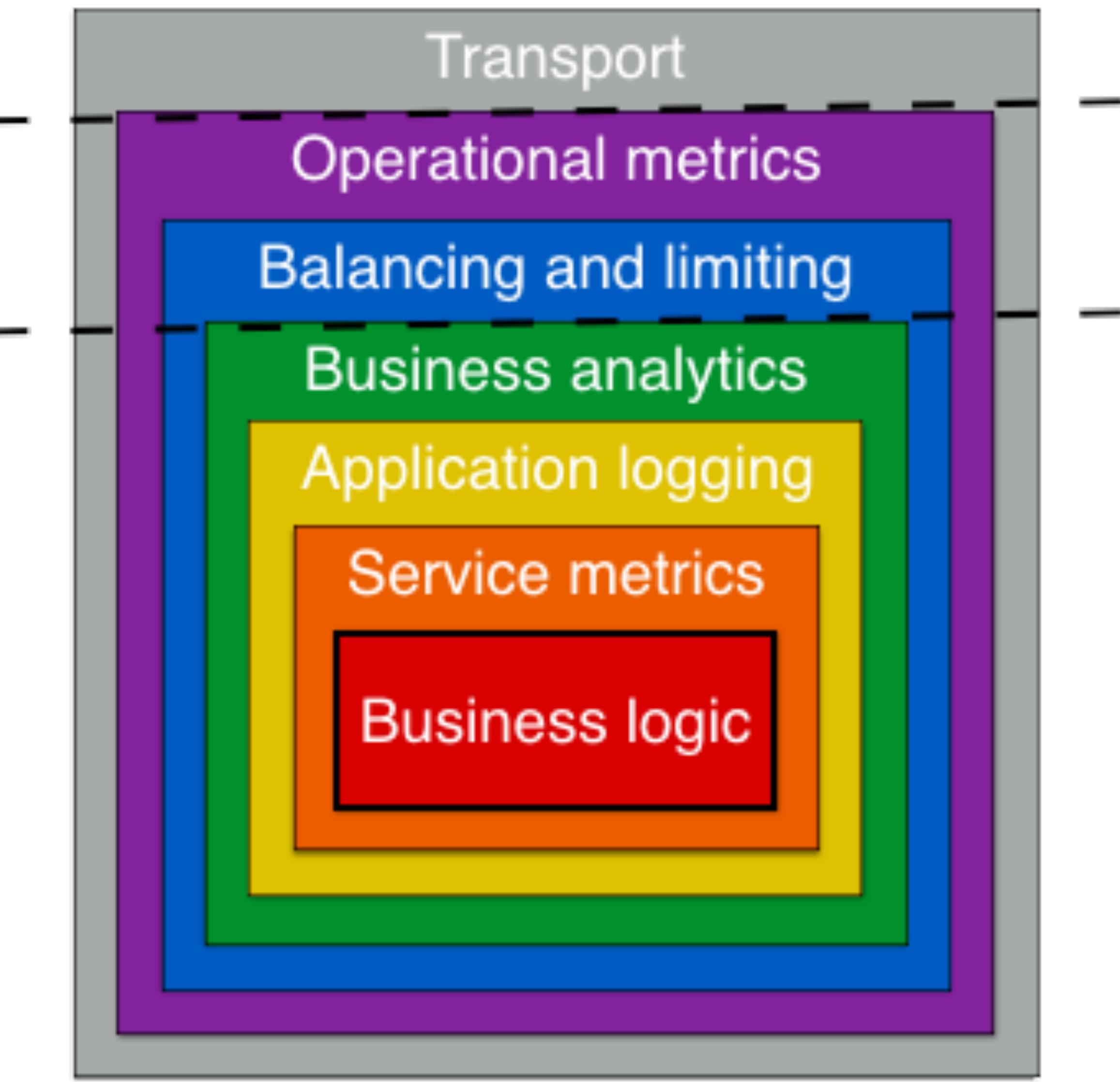
The naïve implementation



Code
01

addsvc

Structure & scaffolding



Decorators

```
func foo(...) {  
    // business logic  
}
```

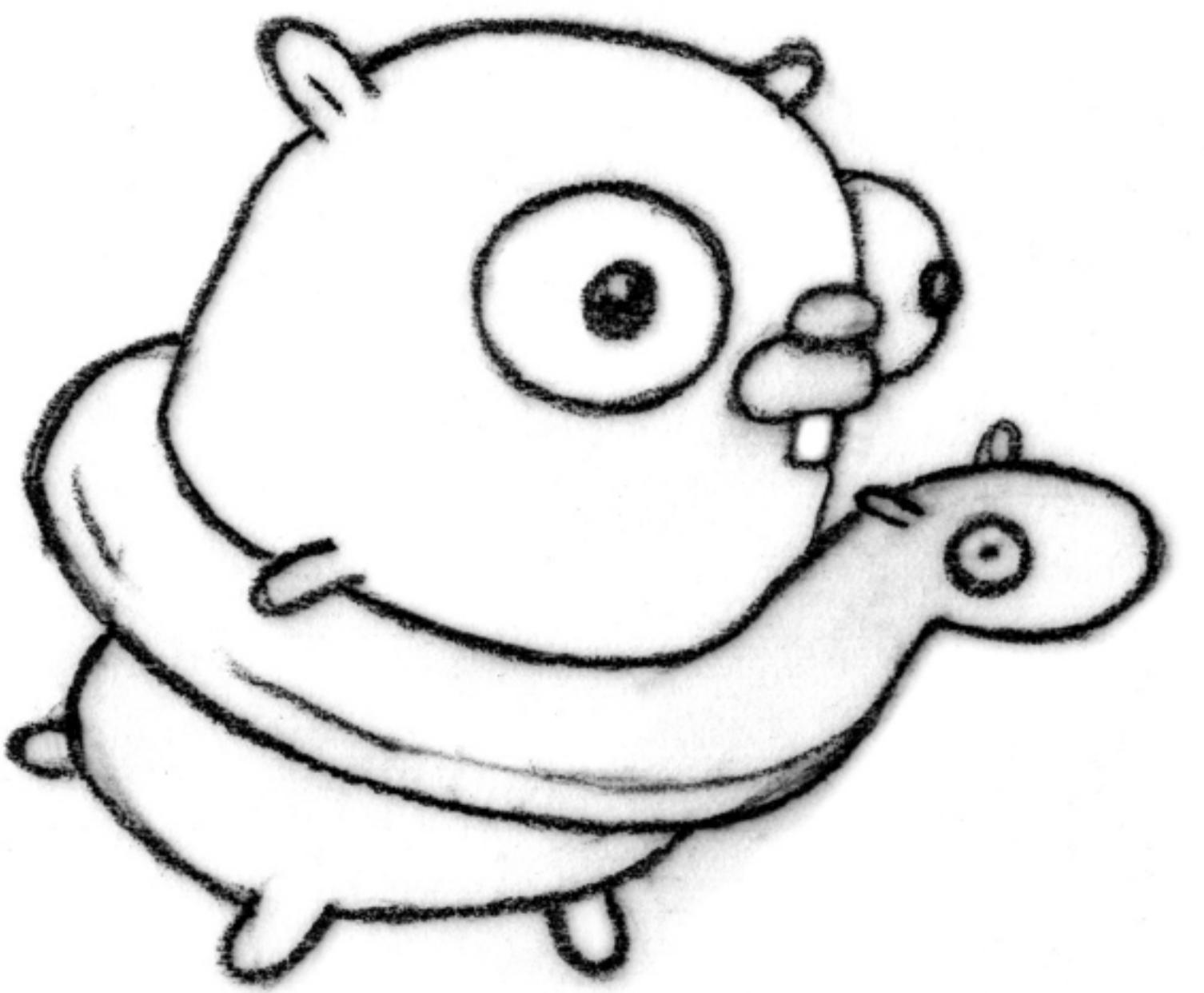
```
func log(...) {  
    // proceed as normal  
    log.Printf("...")  
}
```

```
func instrument(...) {  
    // proceed as normal  
    m.With(method, code).Observe(t)  
}
```

```
func rateLimit(...) {  
    if aboveThreshold {  
        error  
    }  
    // proceed as normal  
}
```

Endpoint

- Generalize each operation as RPC: request, response
 - type Endpoint func(request) response
- Accommodate failure and request-scoped information
 - type Endpoint func(ctx context.Context, request interface{}) (response interface{}, err error)
- Empty interface? Empty interface :(

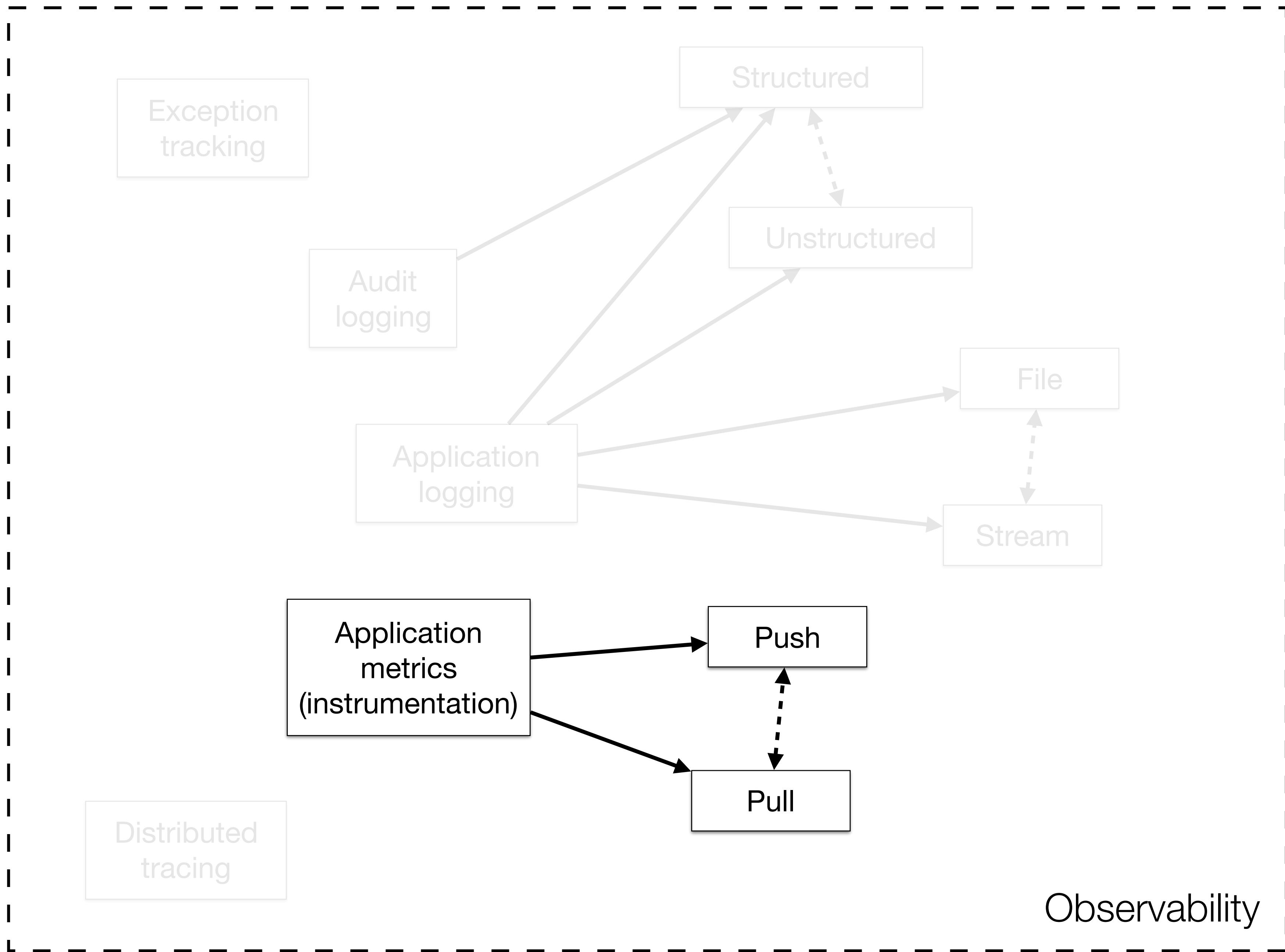


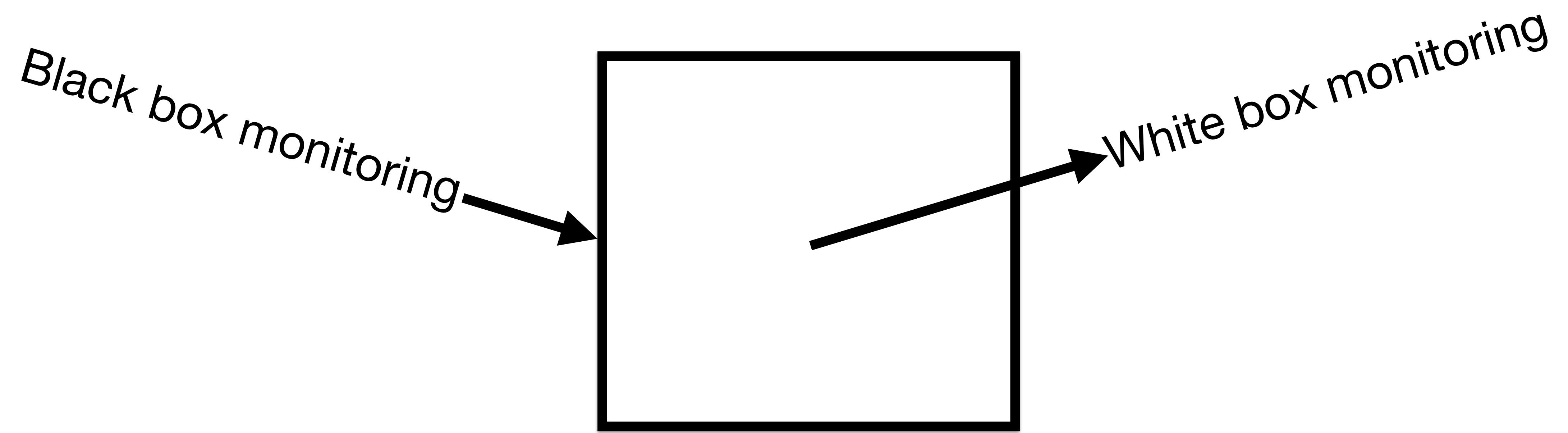
Code
02

addsvc

Decorators & middlewares

Instrumenting





circonus	Package circonus provides a Circonus backend for metrics.
discard	Package discard provides a no-op metrics backend.
dogstatsd	Package dogstatsd provides a DogStatsD backend for package metrics.
expvar	Package expvar provides expvar backends for metrics.
generic	Package generic implements generic versions of each of the metric types.
graphite	Package graphite provides a Graphite backend for metrics.
influx	Package influx provides an InfluxDB implementation for metrics.
internal/lv	
internal/ratemap	Package ratemap implements a goroutine-safe map of string to float64.
multi	Package multi provides adapters that send observations to multiple metrics simultaneously.
prometheus	Package prometheus provides Prometheus implementations for metrics.
provider	Package provider provides a factory-like abstraction for metrics backends.
statsd	Package statsd provides a StatsD backend for package metrics.

```
package metrics

// Counter describes a metric that accumulates values monotonically.
// An example of a counter is the number of received HTTP requests.
type Counter interface {
    With(labelValues ...string) Counter
    Add(delta float64)
}

// Gauge describes a metric that takes specific values over time.
// An example of a gauge is the current depth of a job queue.
type Gauge interface {
    With(labelValues ...string) Gauge
    Set(value float64)
}

// Histogram describes a metric that takes repeated observations of the same
// kind of thing, and produces a statistical summary of those observations,
// typically expressed as quantiles or buckets. An example of a histogram is
// HTTP request latencies.
type Histogram interface {
    With(labelValues ...string) Histogram
    Observe(value float64)
}
```

USE method

Brendan Gregg

Utilization

Saturation

Error count (rate)

For resources e.g. queues

RED method

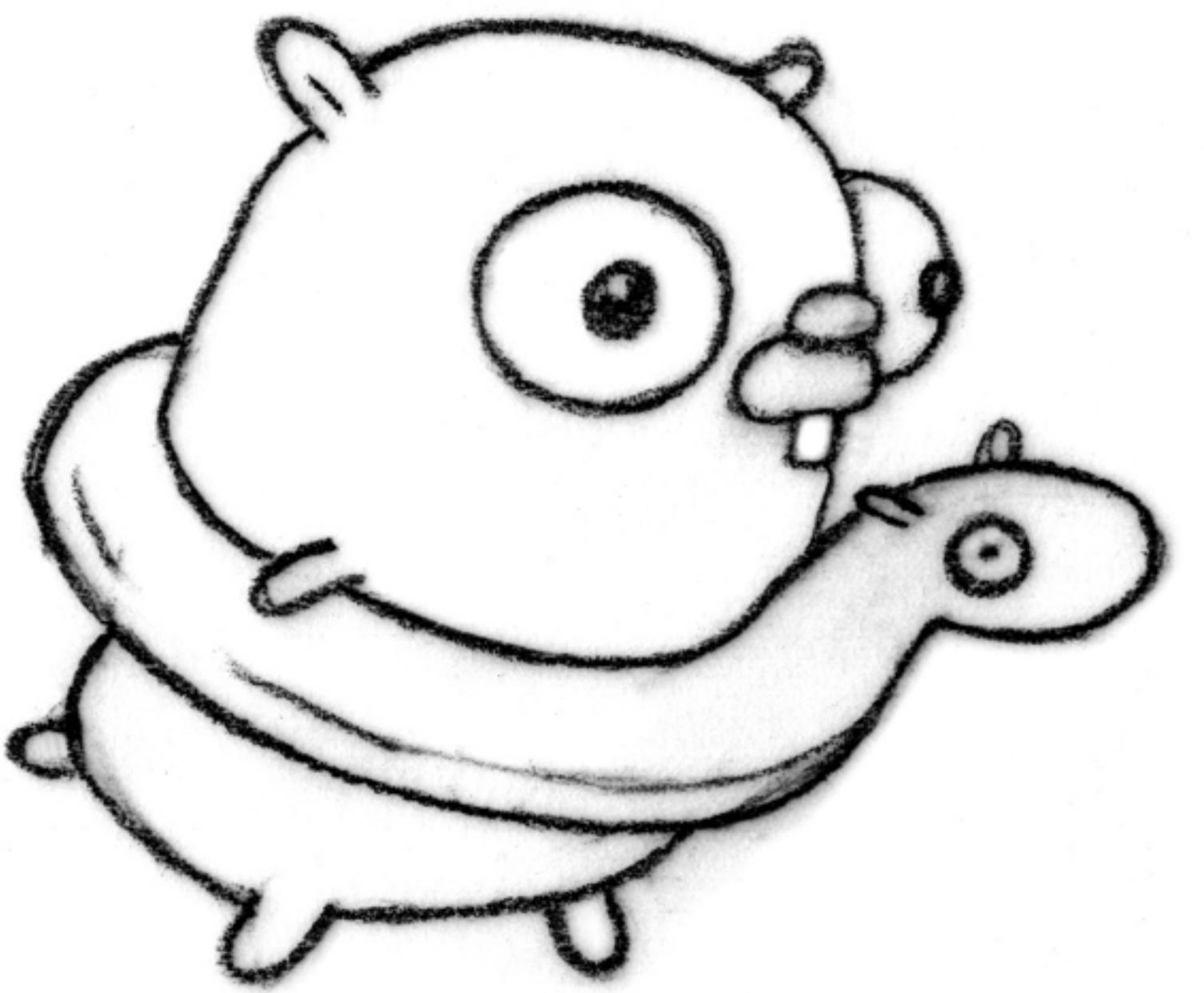
Tom Wilkie

Request count (rate)

Error count (rate)

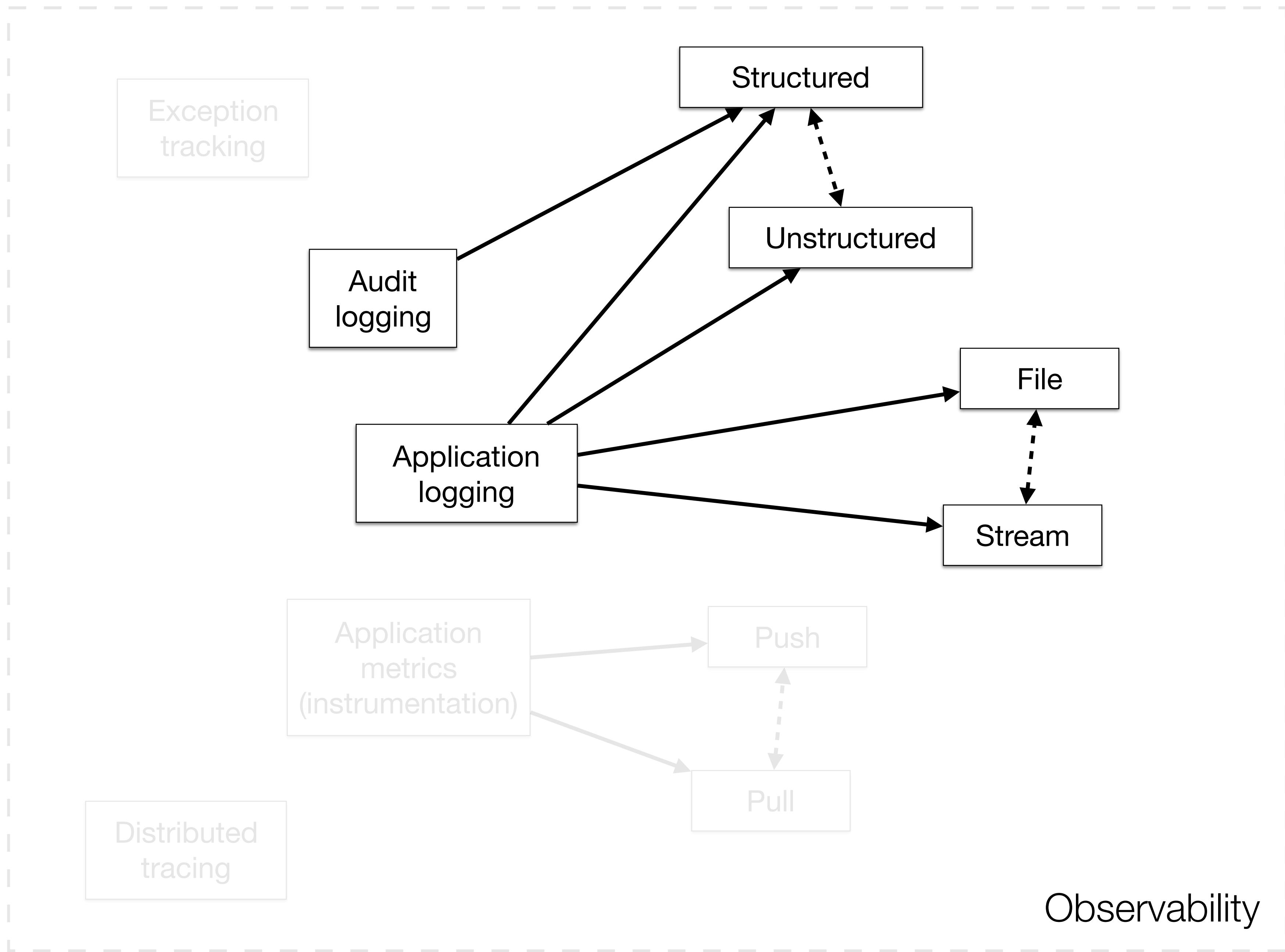
Duration

For e.g. endpoints



Code
03

Logging



Compare/contrast

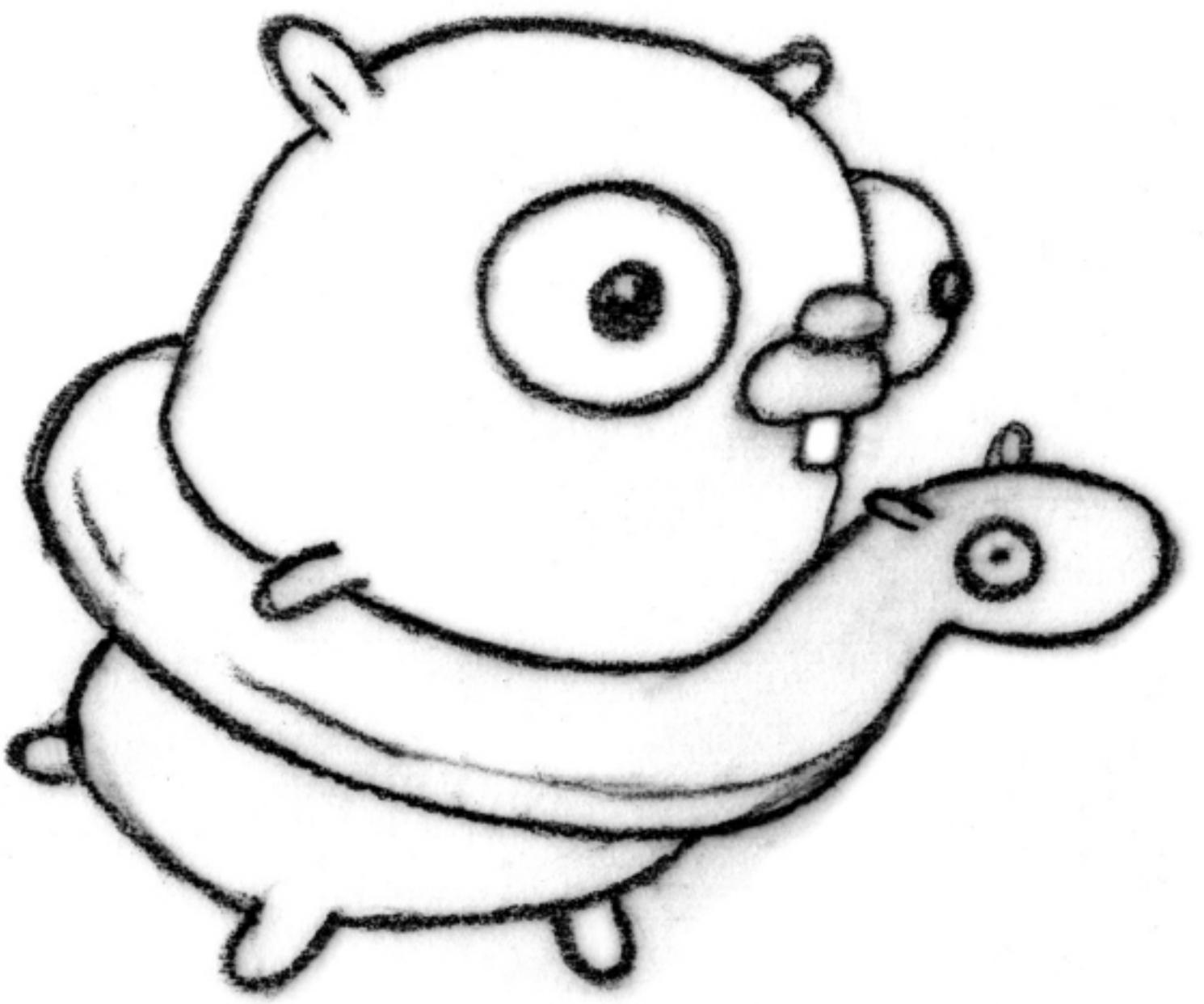
- `log.Infof("HTTP server listening on %s", httpAddr)`
 - 2016-10-31 11:43:06 [INFO] main.go:78: HTTP server listening on :8080
- `level.Info(logger).Log("transport", "HTTP", "addr", httpAddr)`
 - ts="2016-10-31 11:43:06" caller=main.go:78 level=info transport=HTTP addr=:8080
 - { "ts": "2016-10-31 11:43:06", "caller": "main.go:78", "level": "info", "transport": "HTTP", "addr": ":8080" }

Benefits

- Machine parseable
- Unit testable
- No less readable, especially logfmt
- Chris Hines – The Hunt for a Logger Interface

Logs are streams, not files

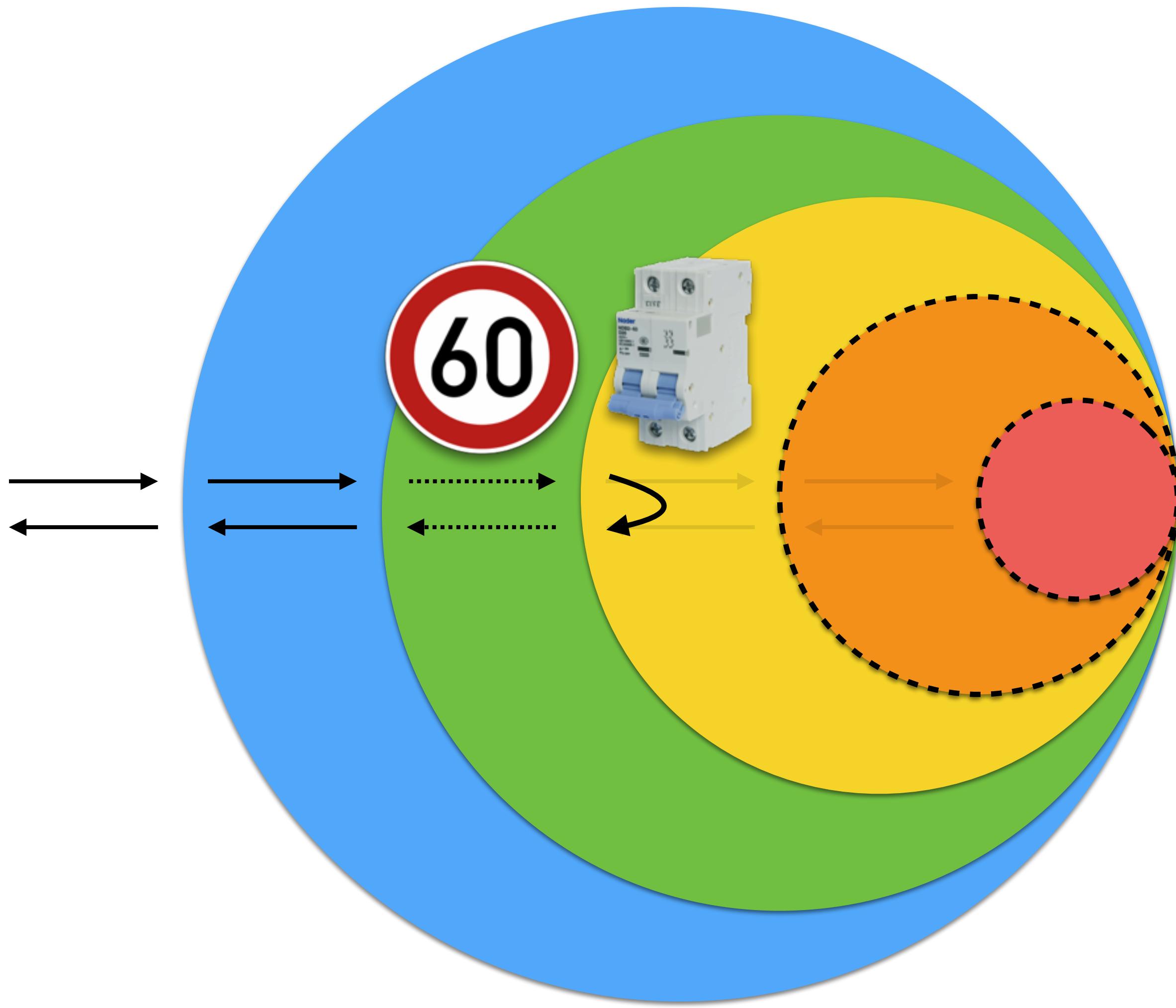
- You may not have a filesystem, write privileges, etc.
- You probably want aggregation – implies platform involvement
- stdout/stderr is a much better contract
- 12factor.net

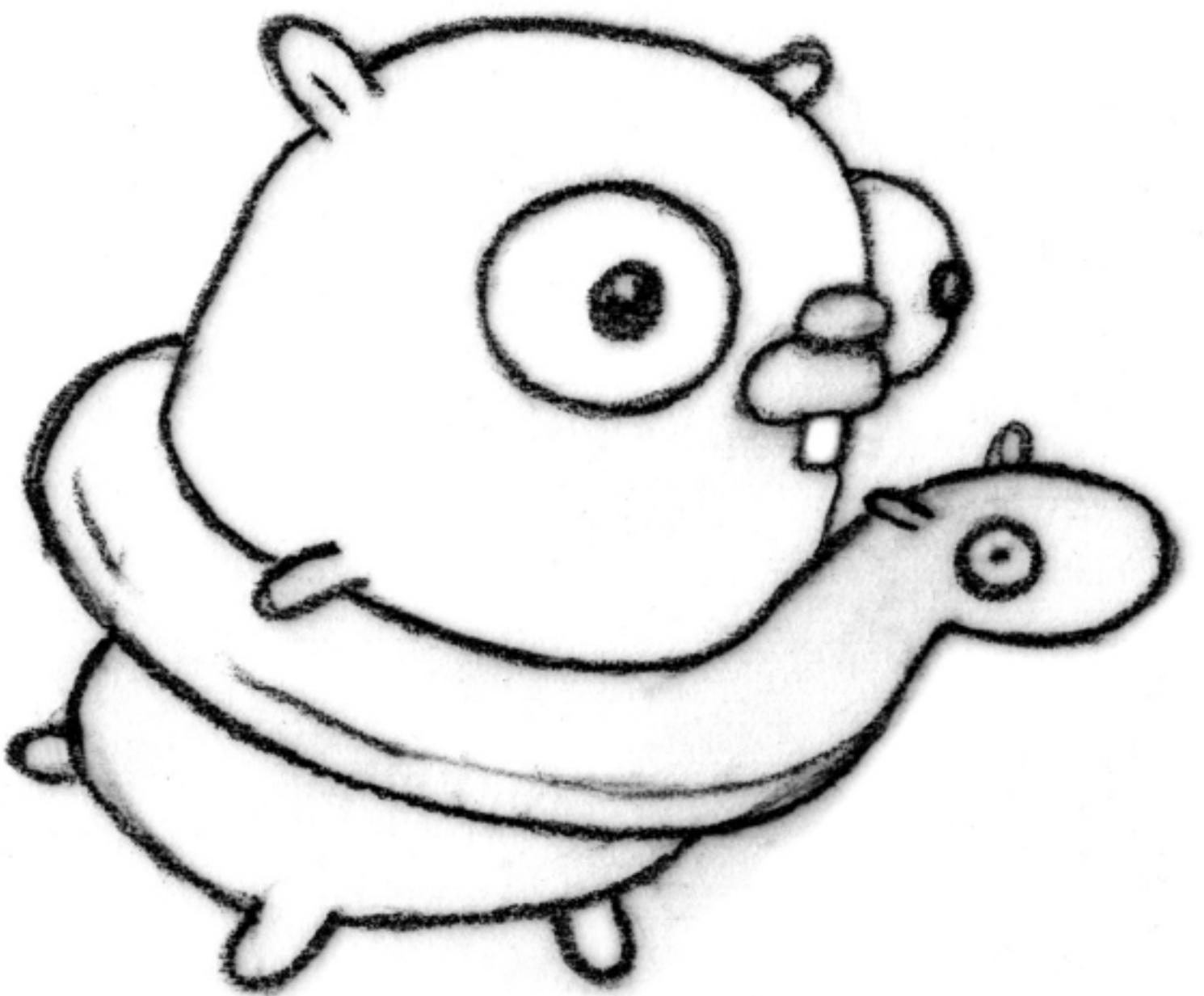


Code

04 – main.go, endpoints, service

Safety

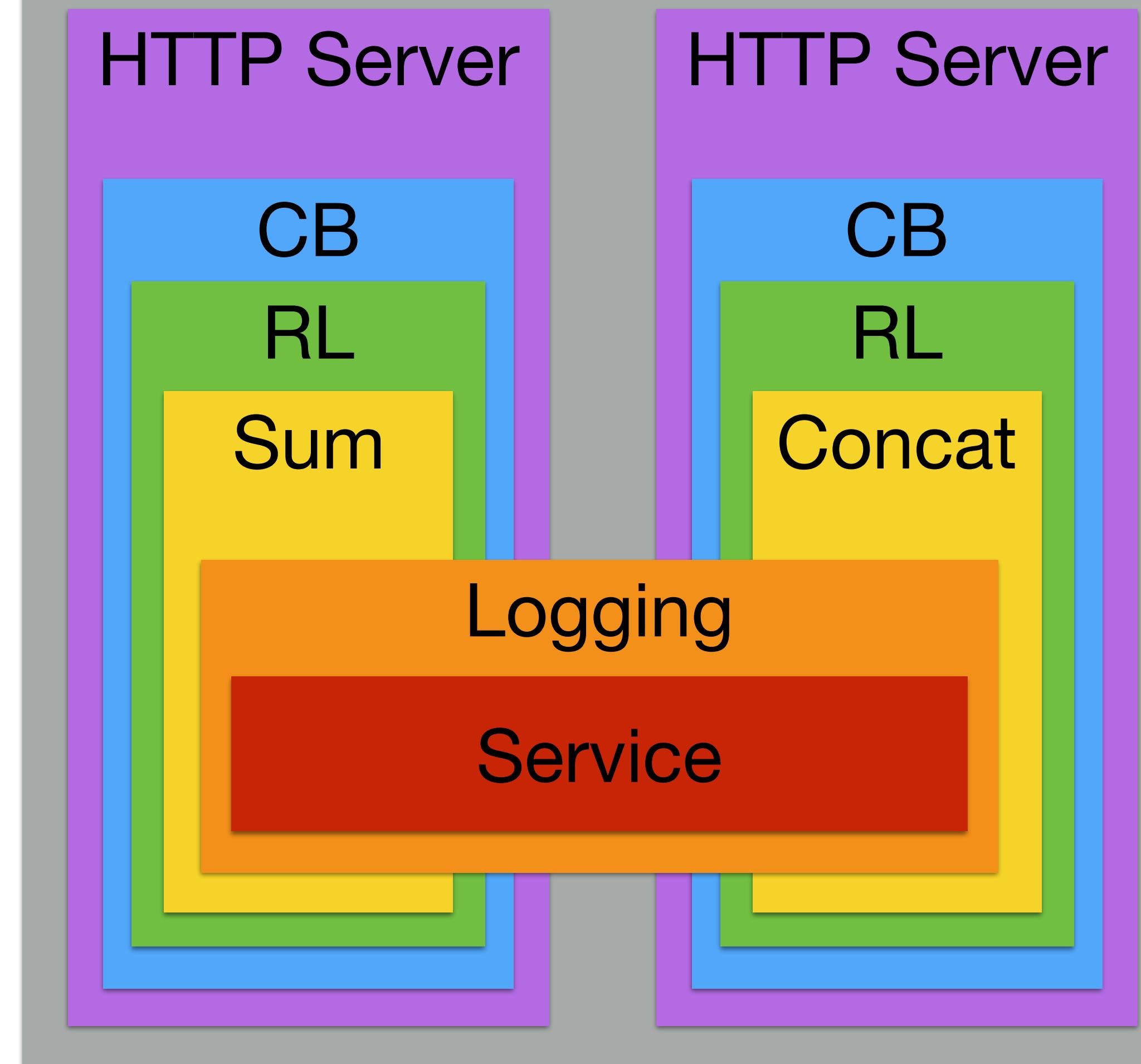


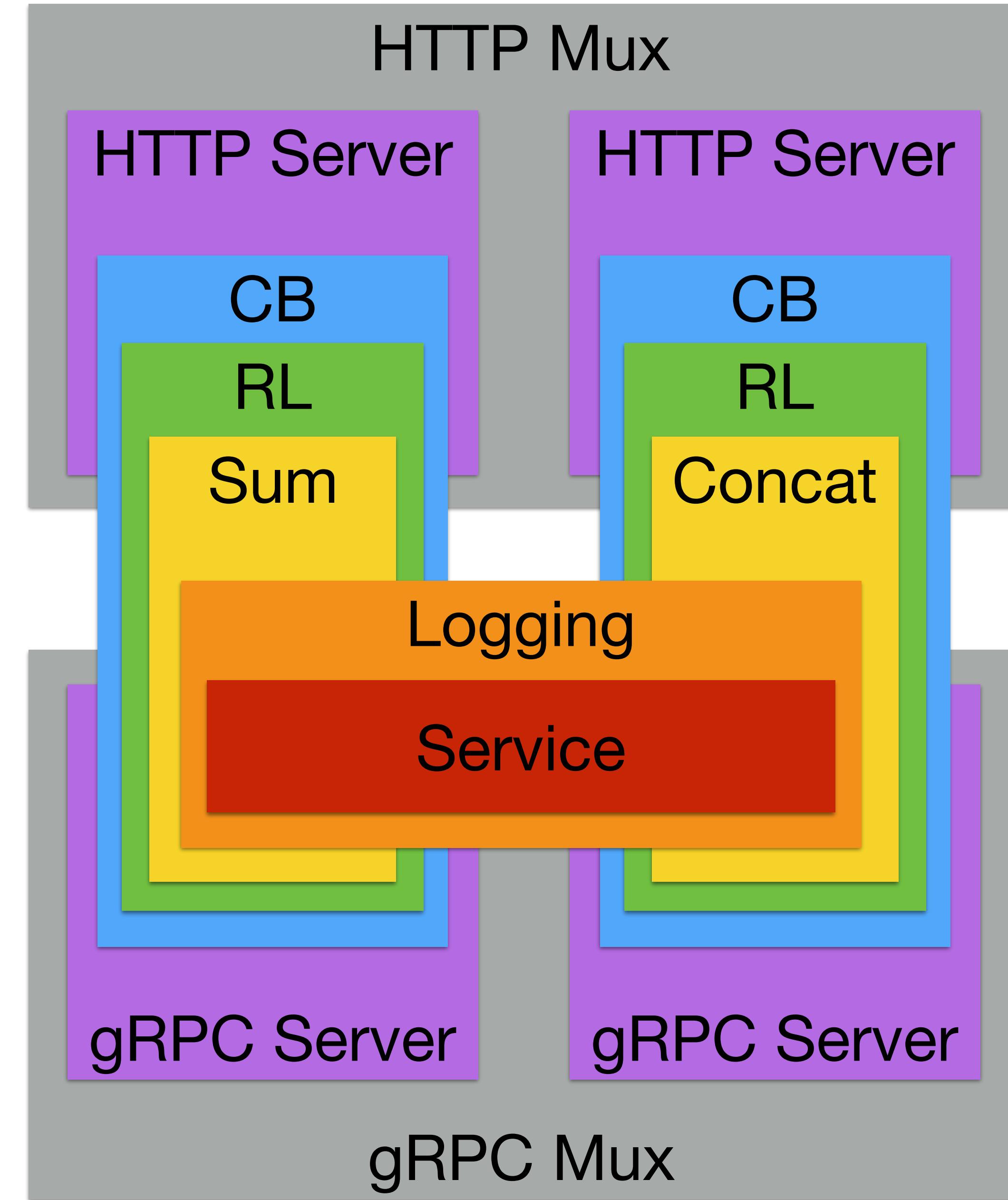


Code

05 – Endpoints middleware

HTTP Mux

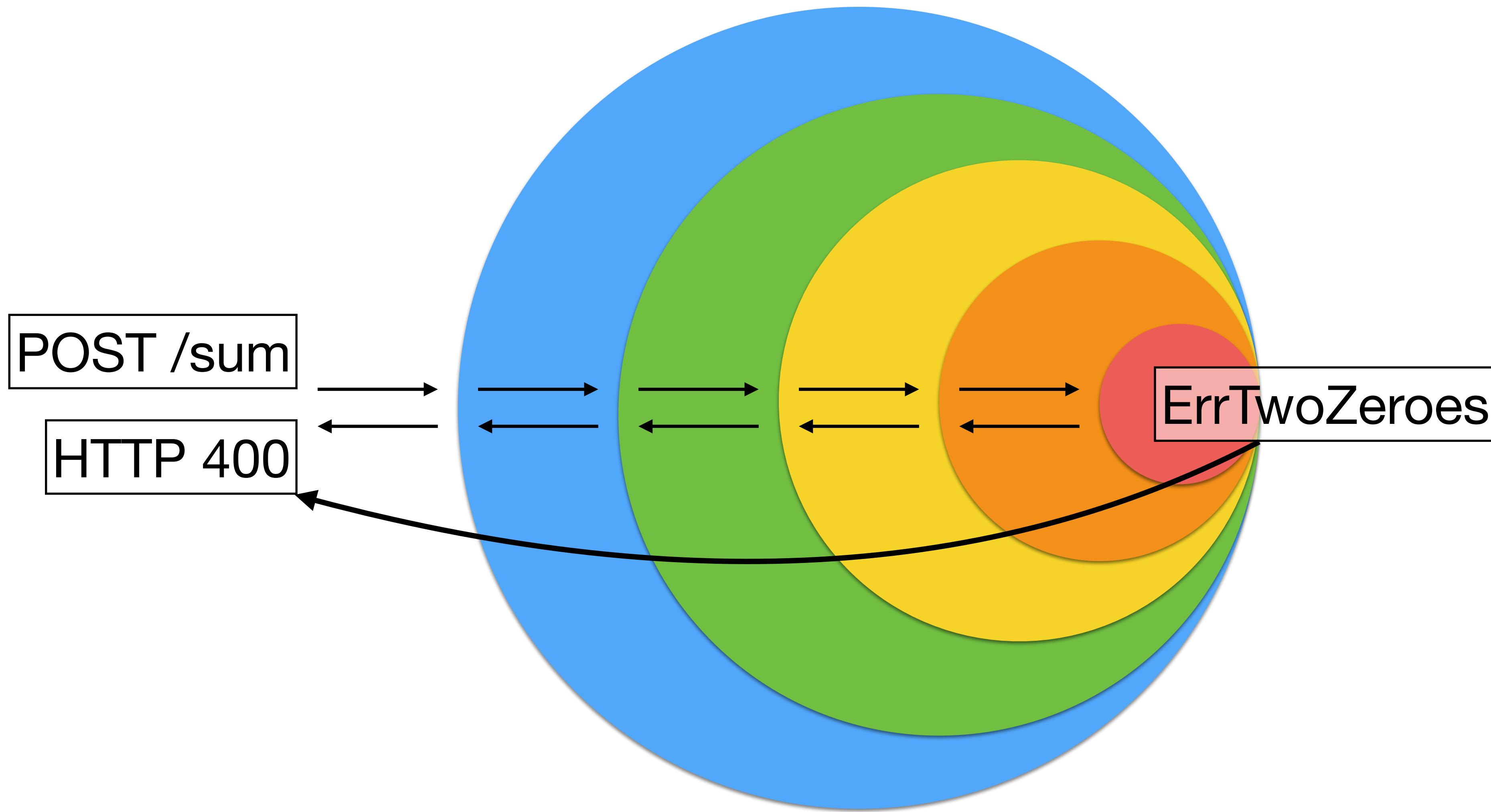


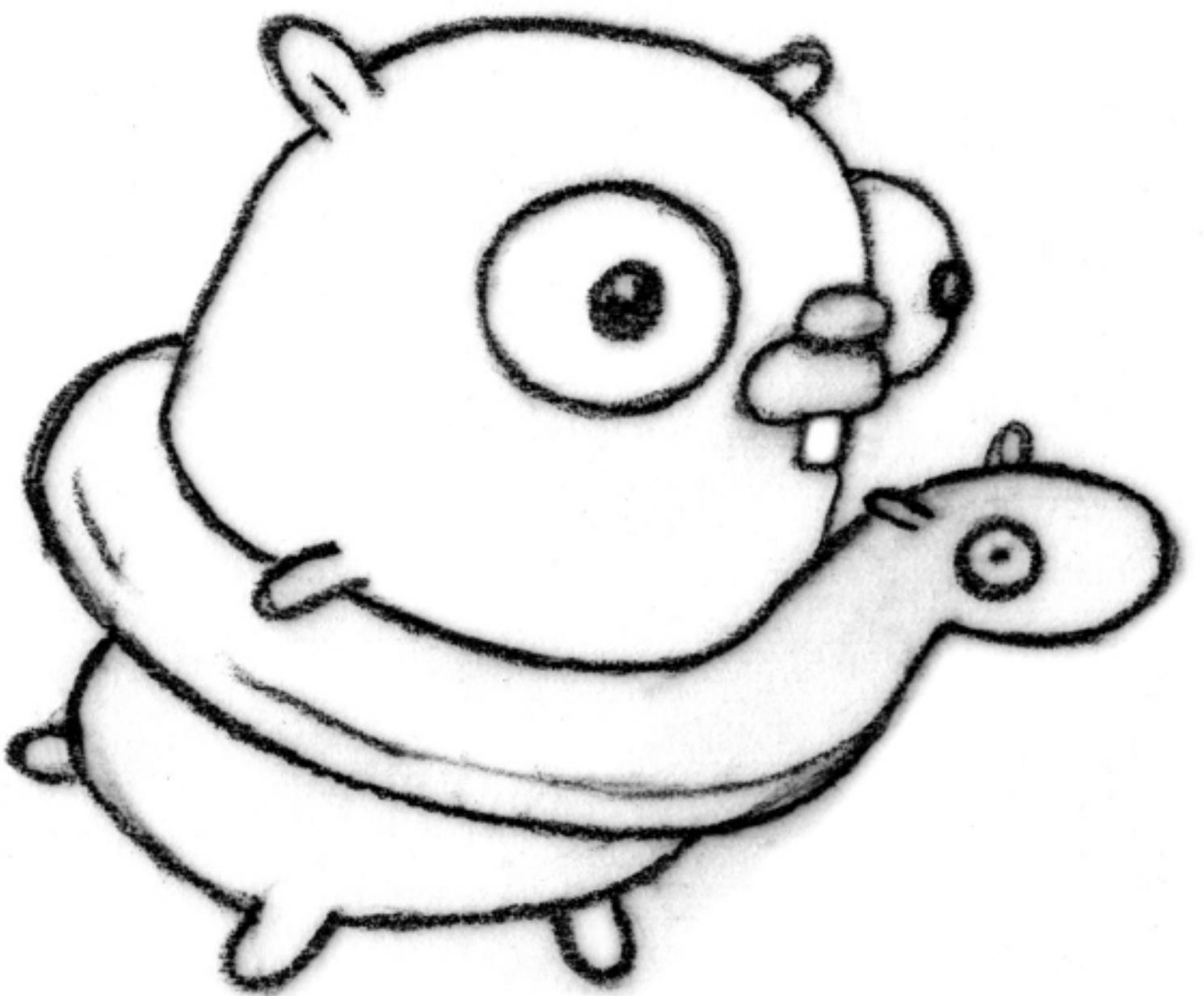


addsvc

Cross-cutting concerns

Error mapping

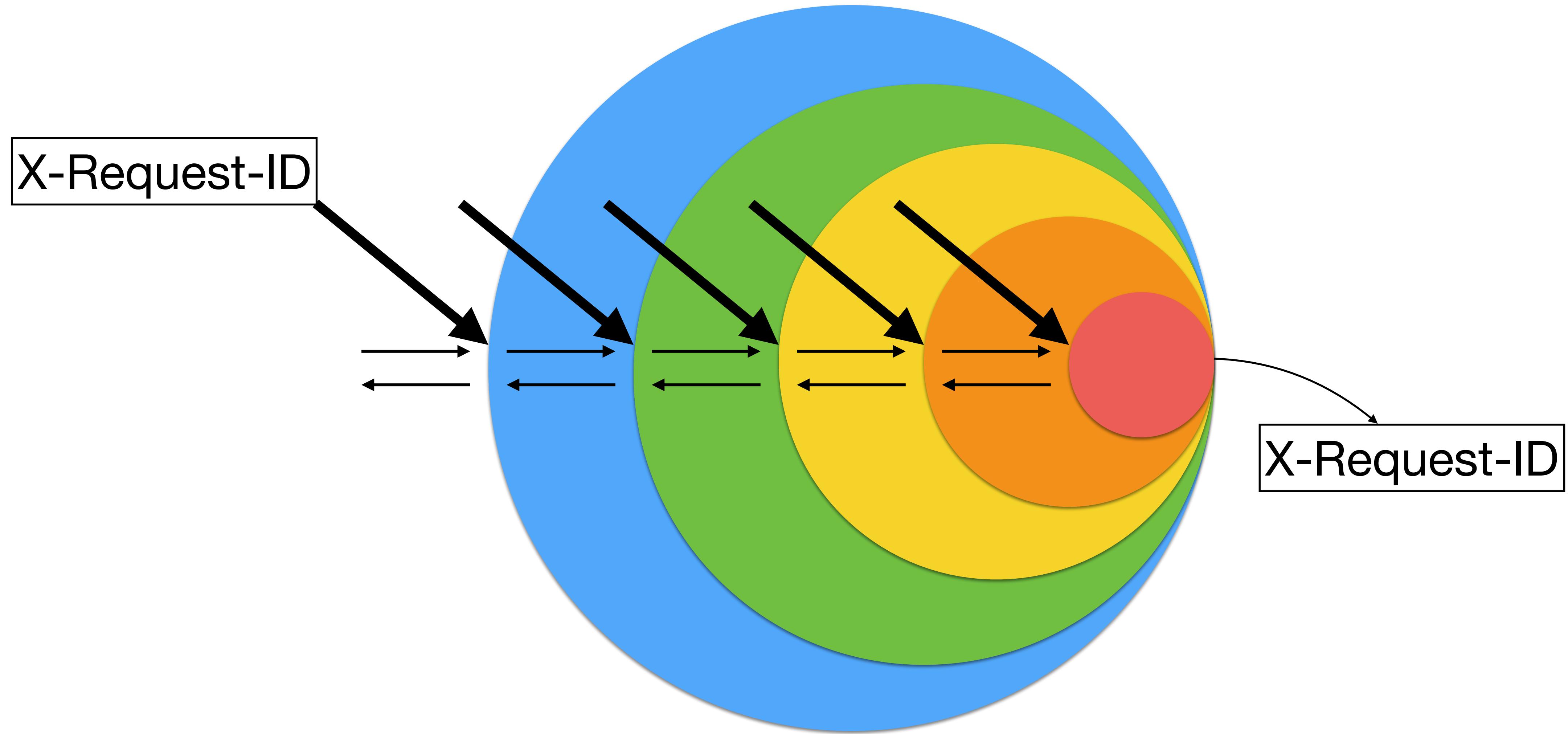


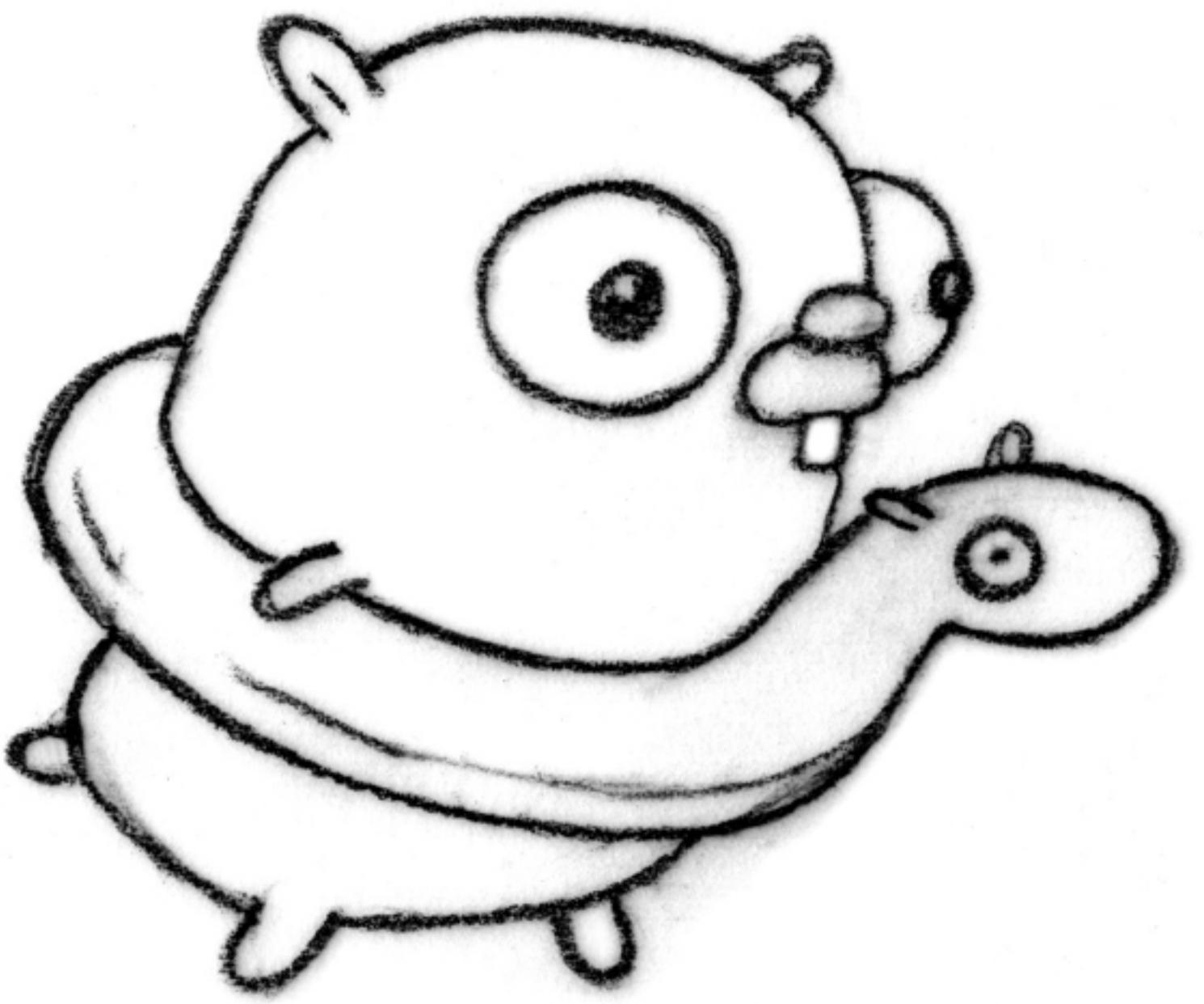


Code

06 – Service, HTTP

Context

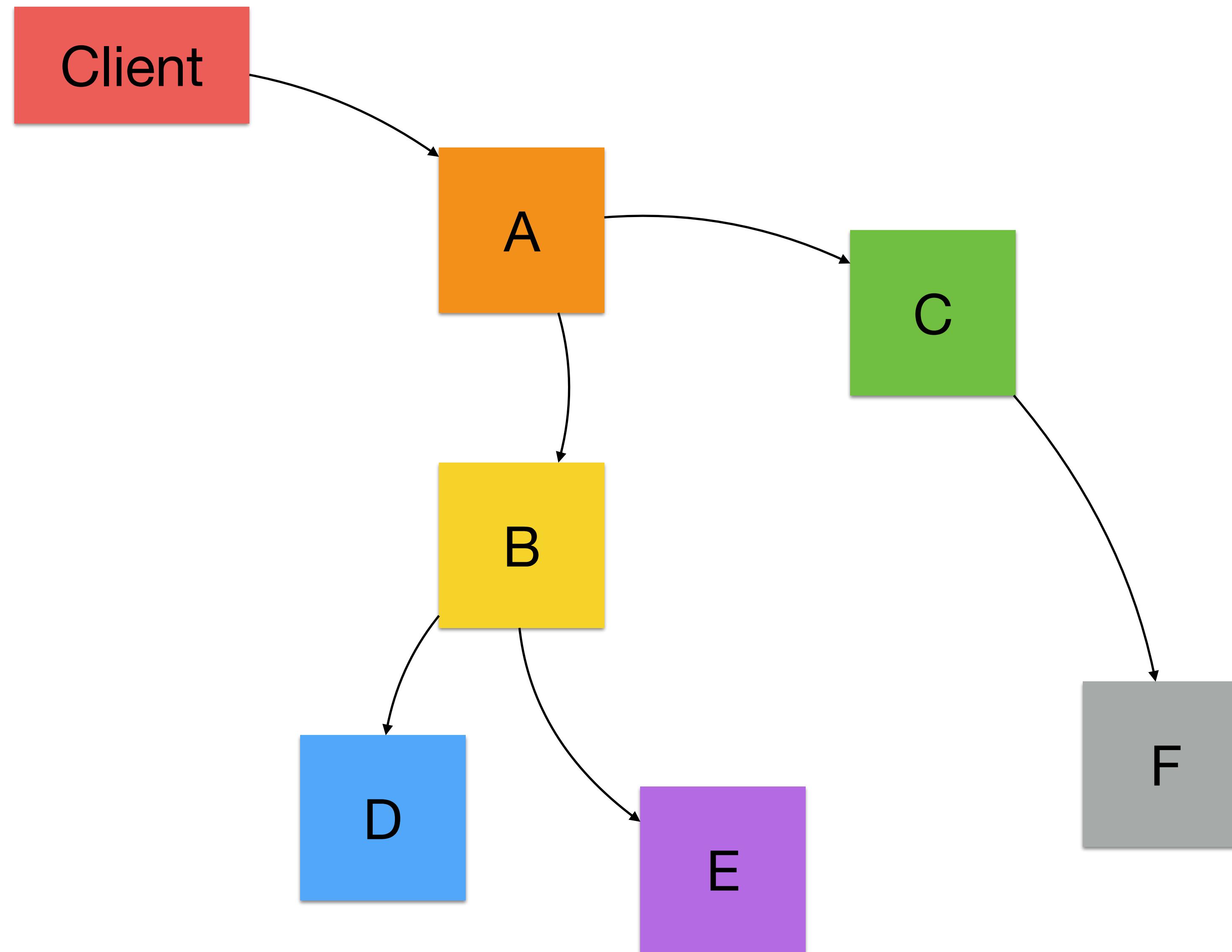


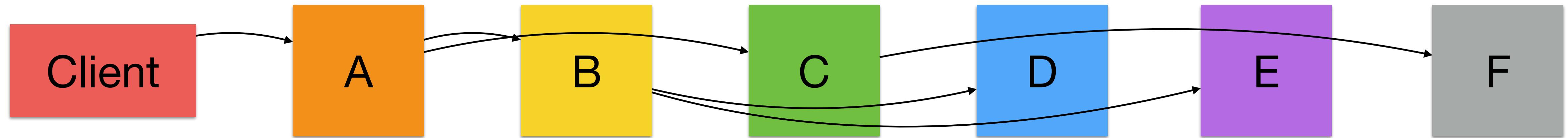


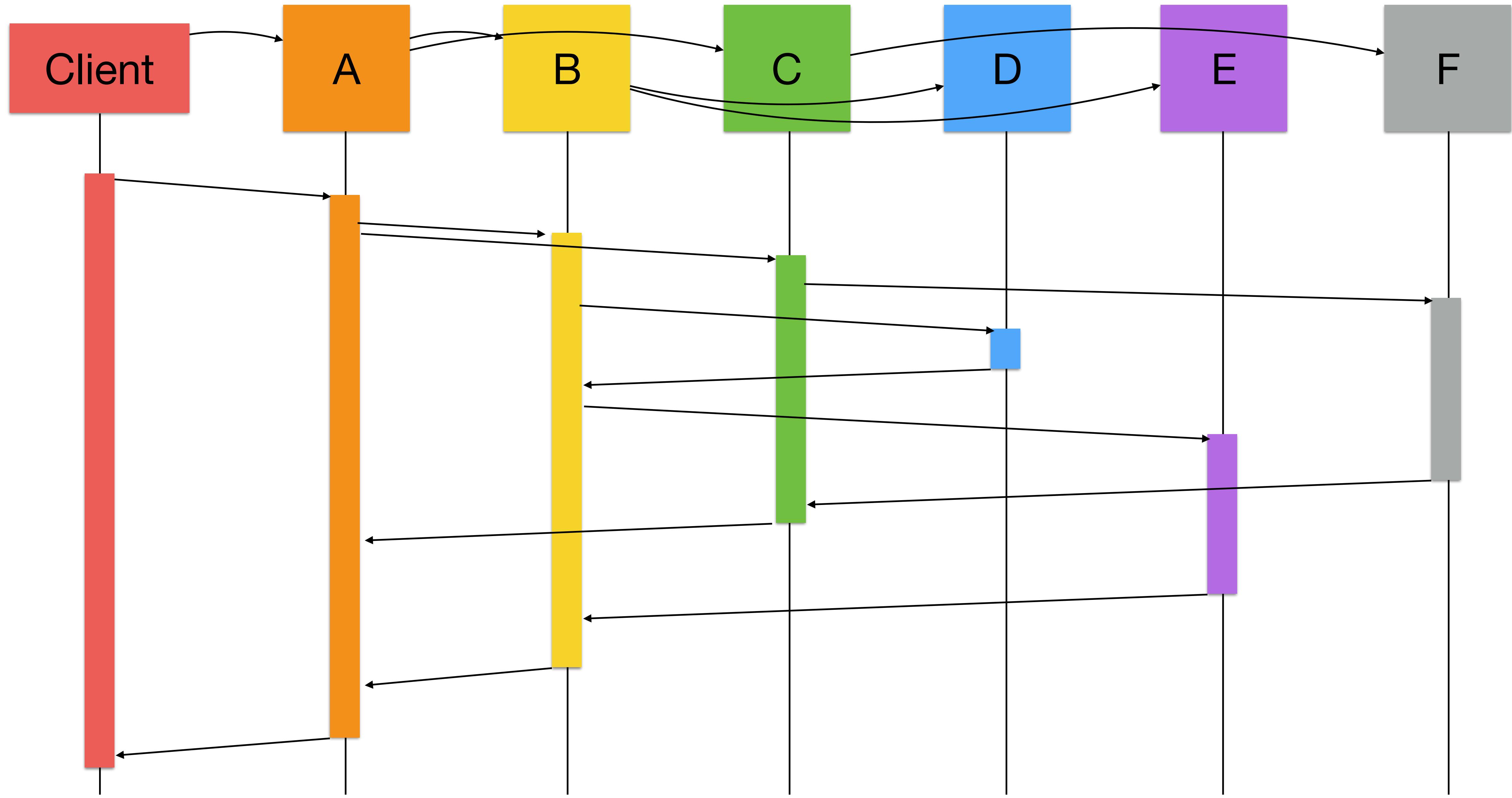
Code

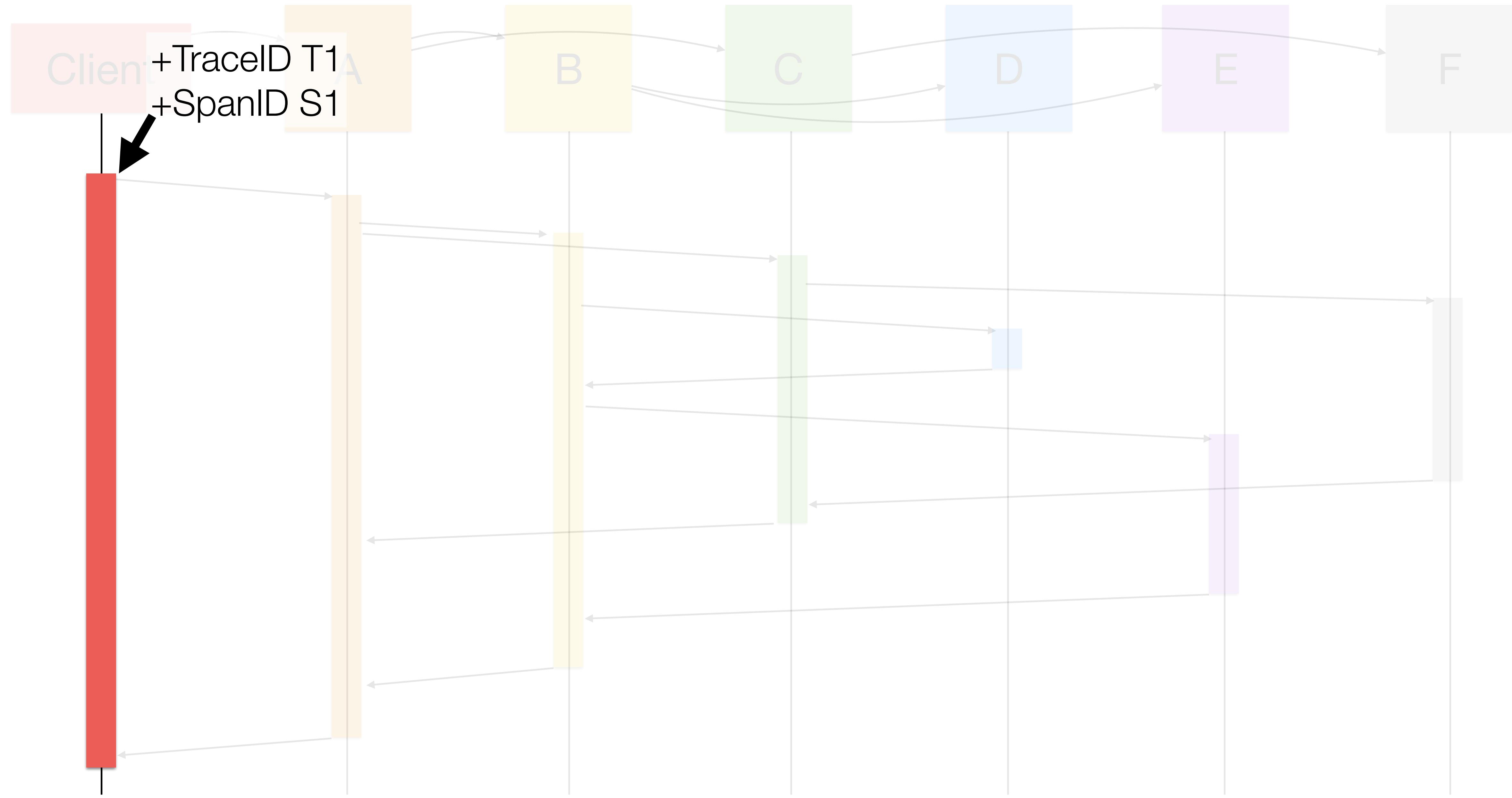
07 – Service, endpoints

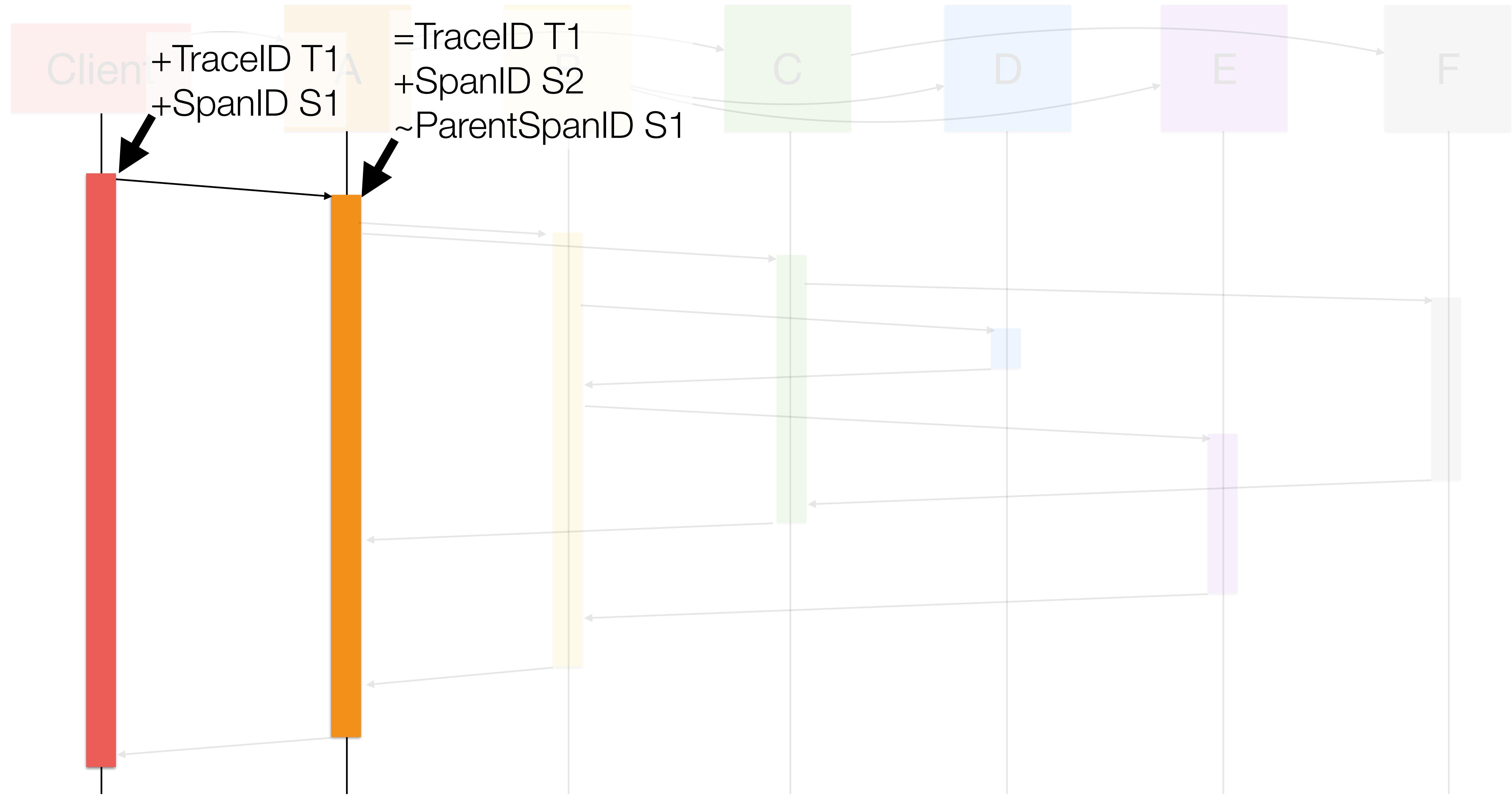
Distributed tracing

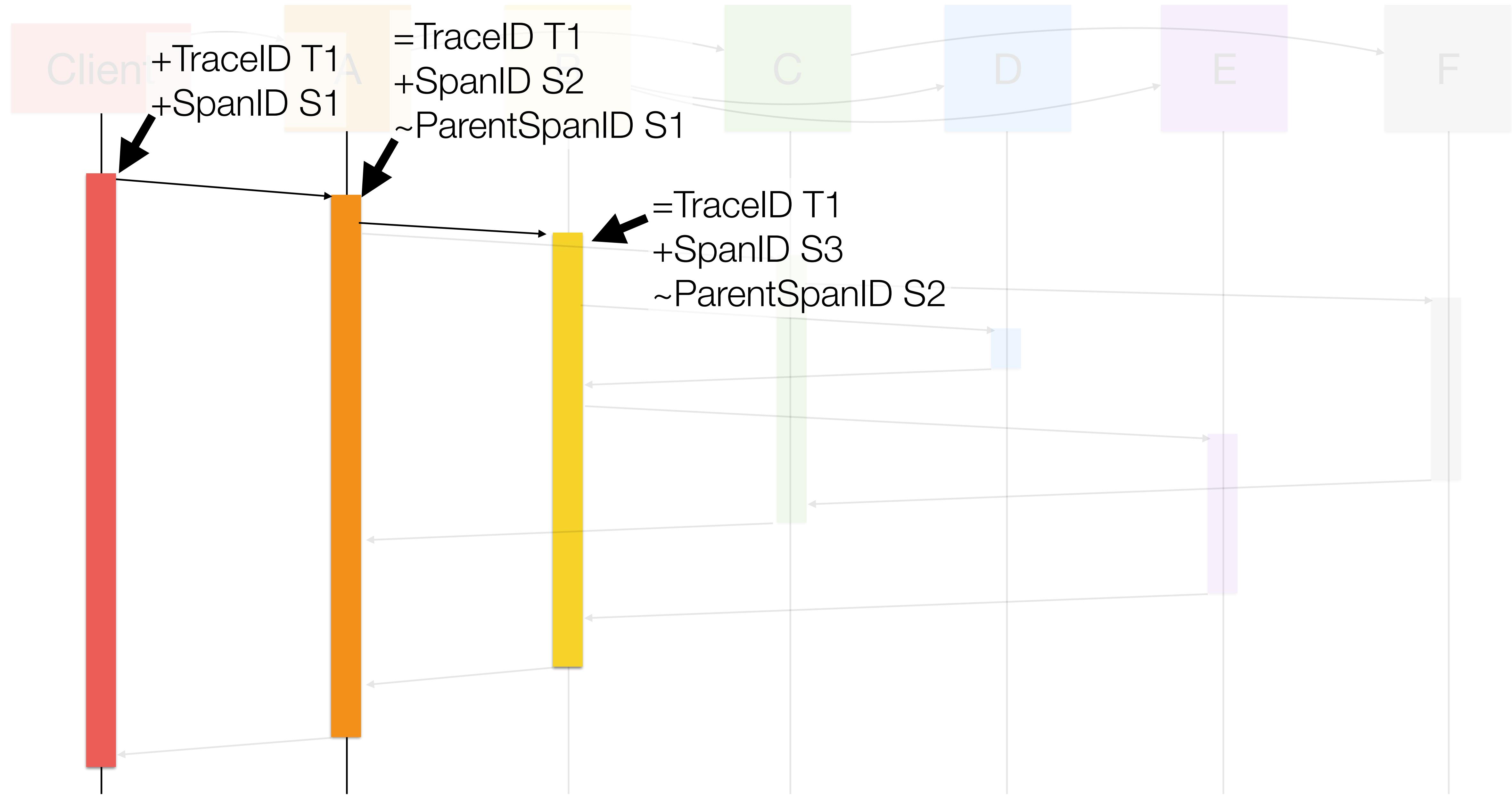


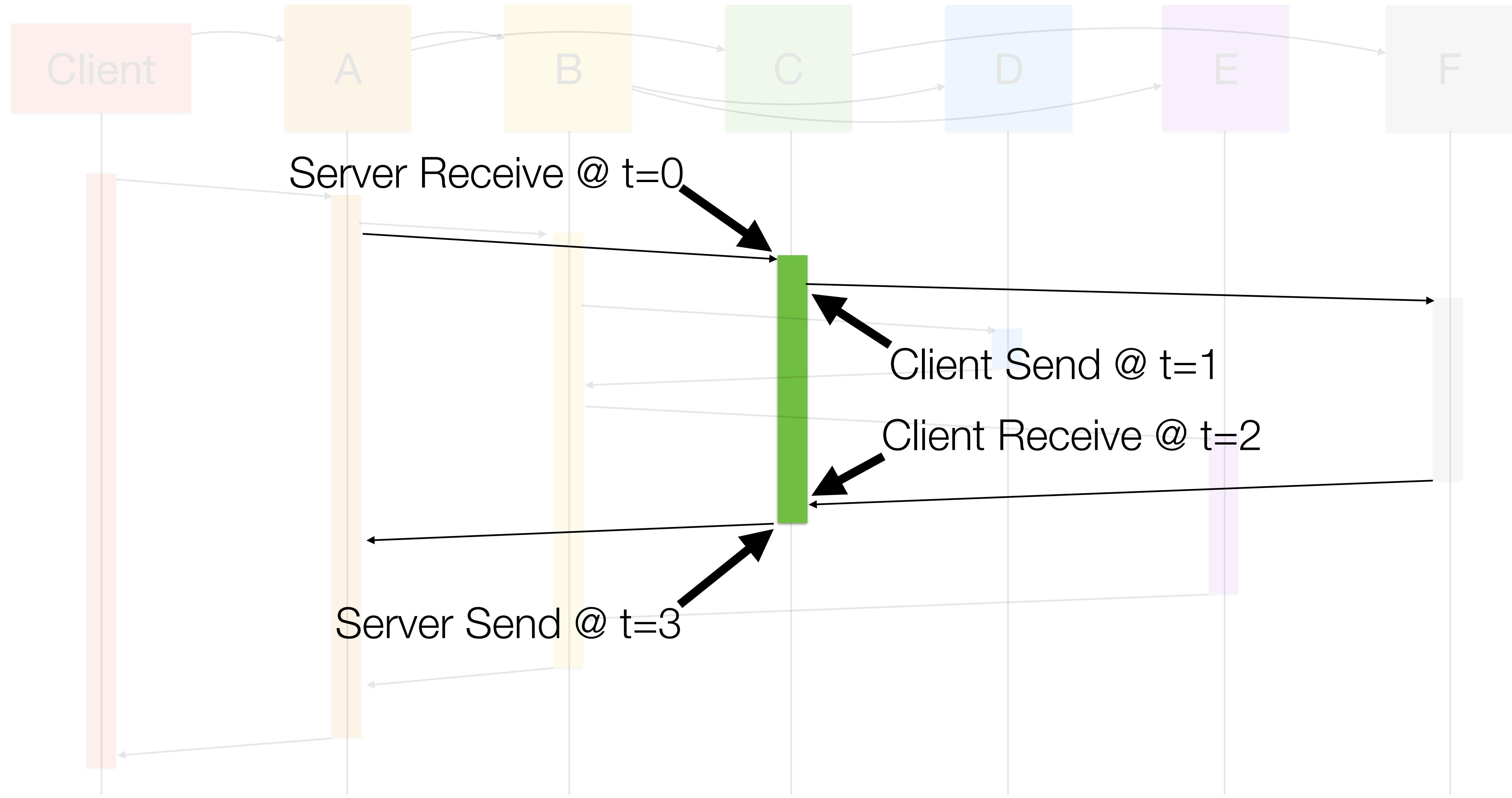


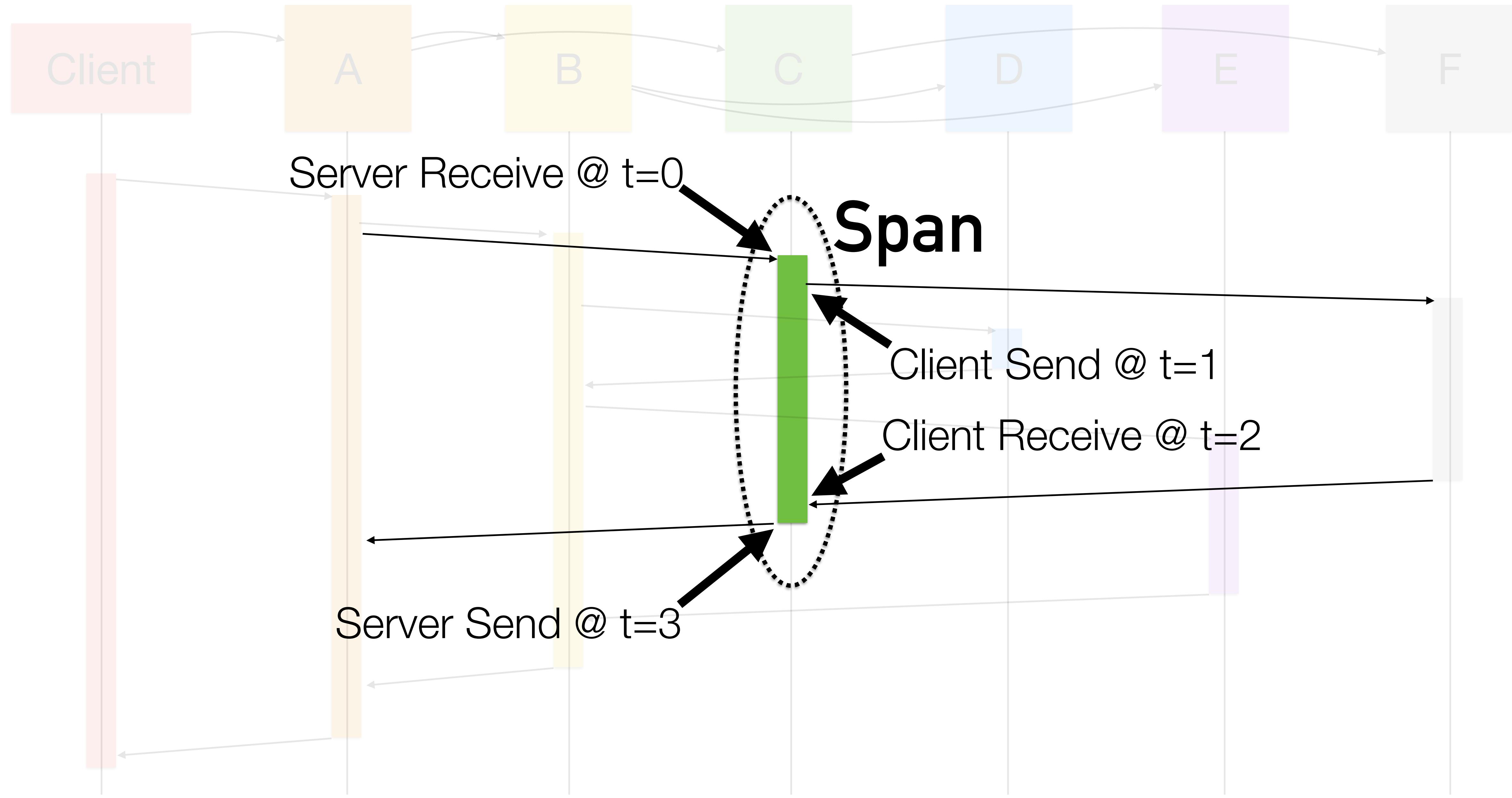


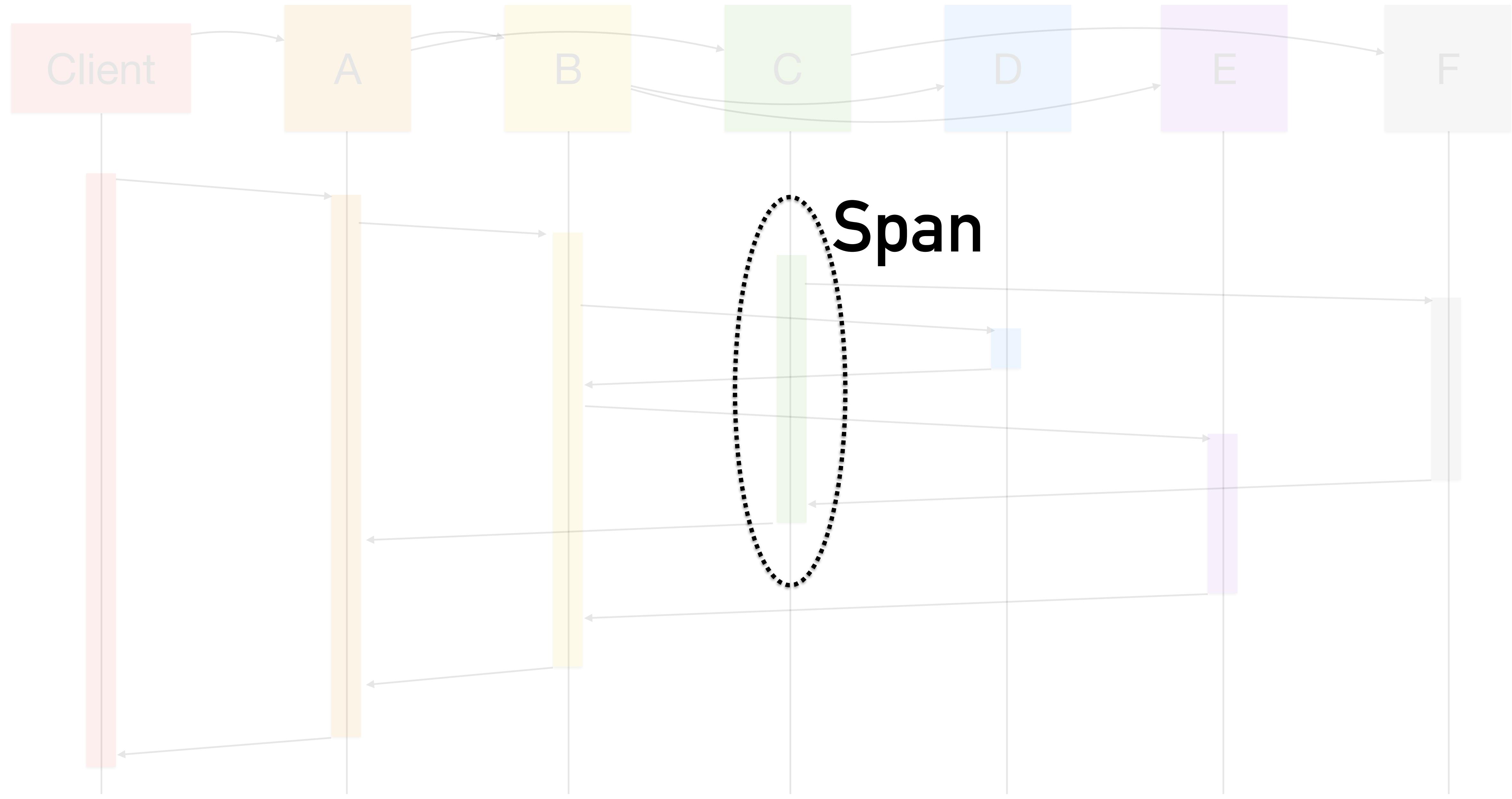


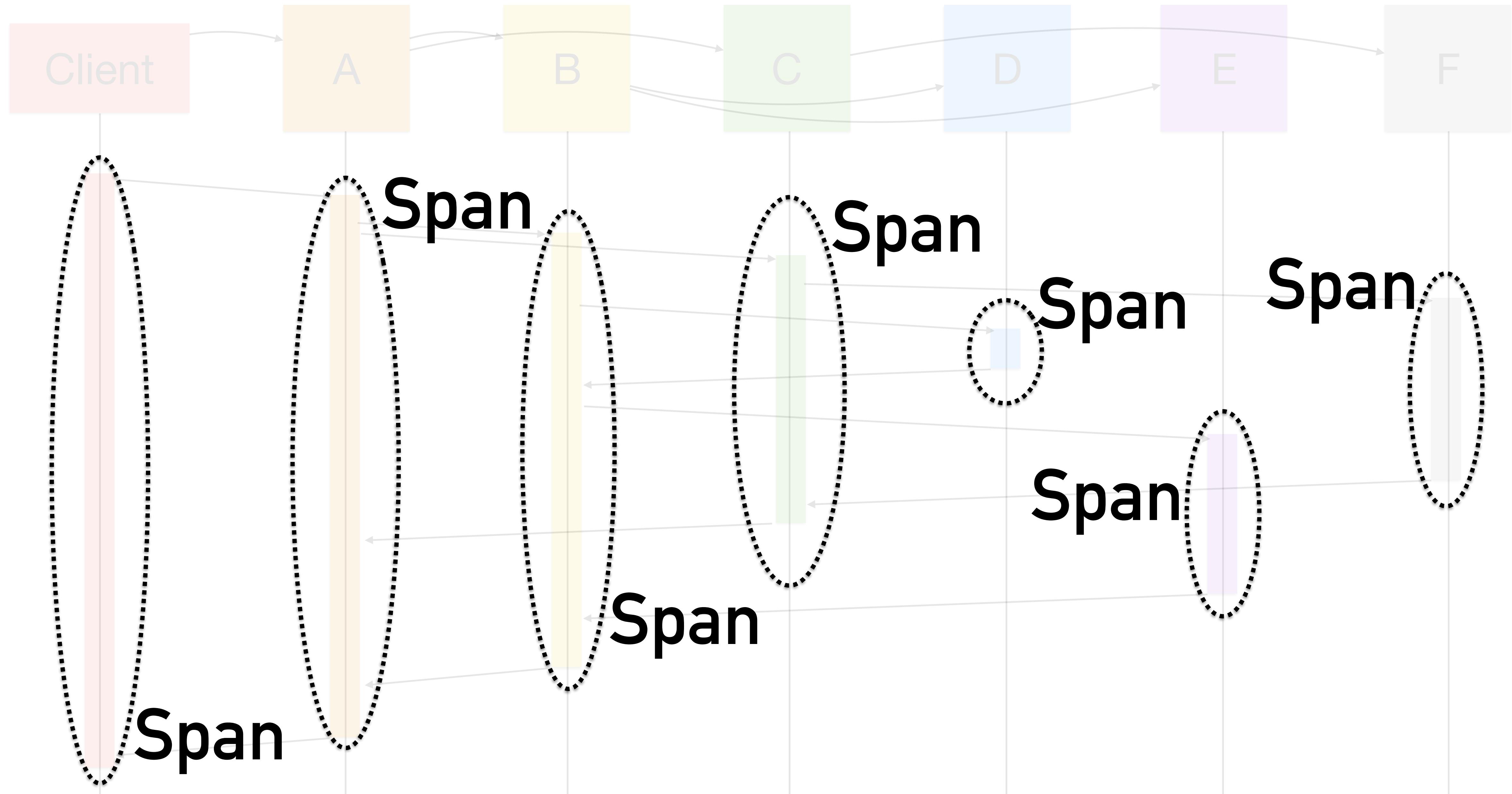


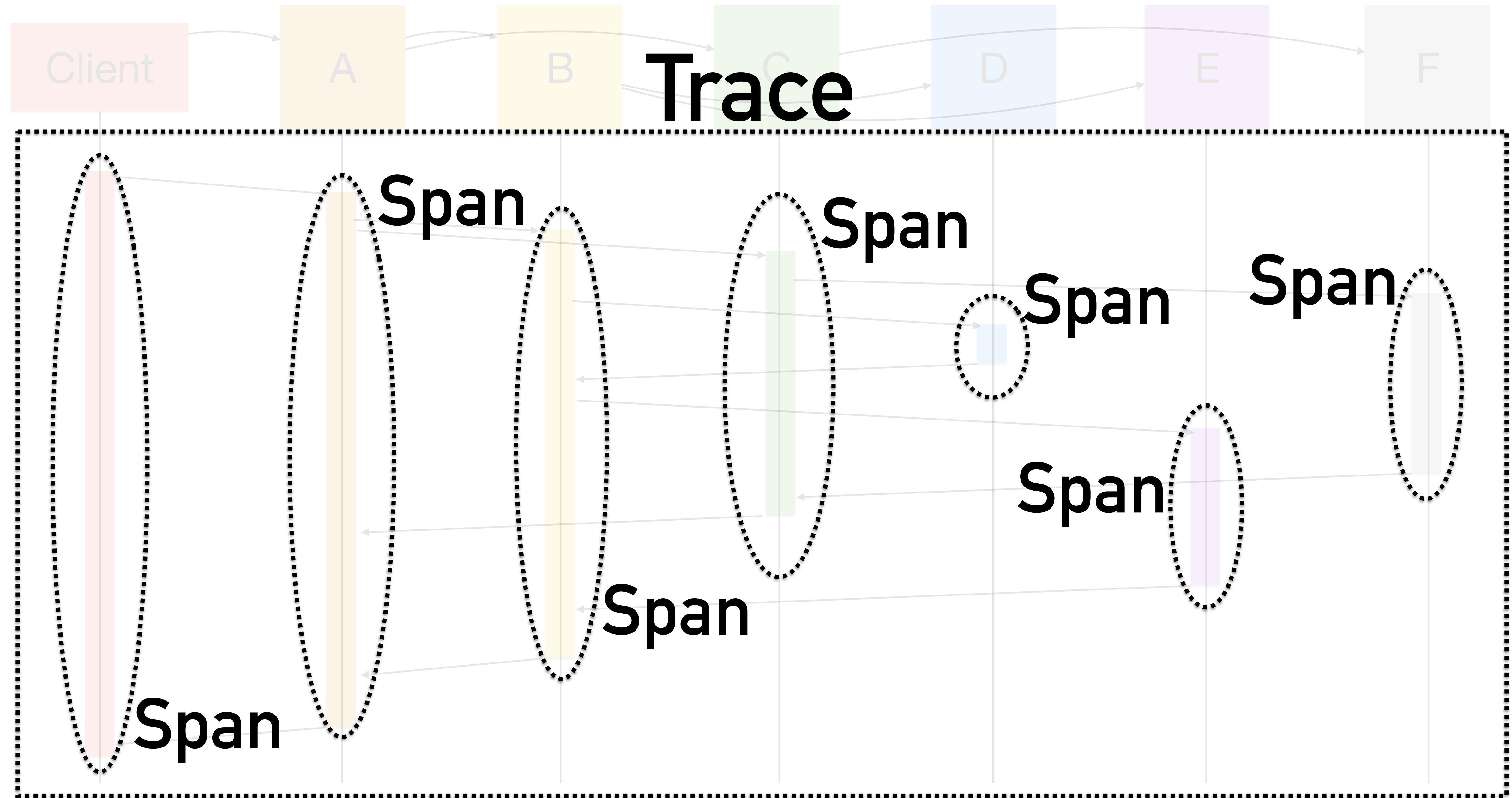


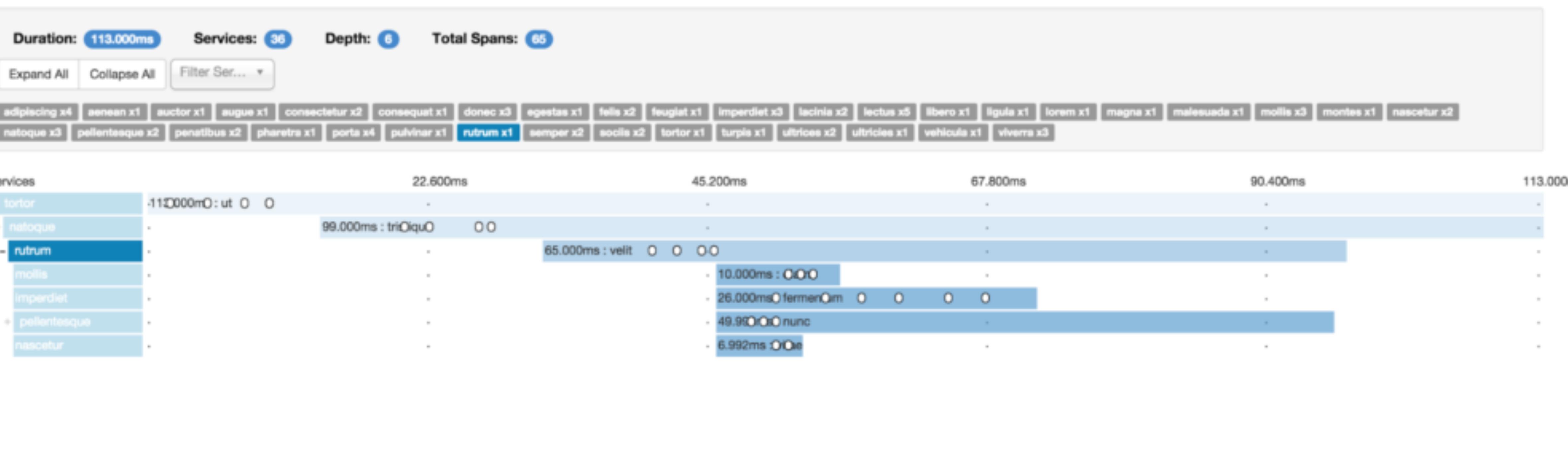


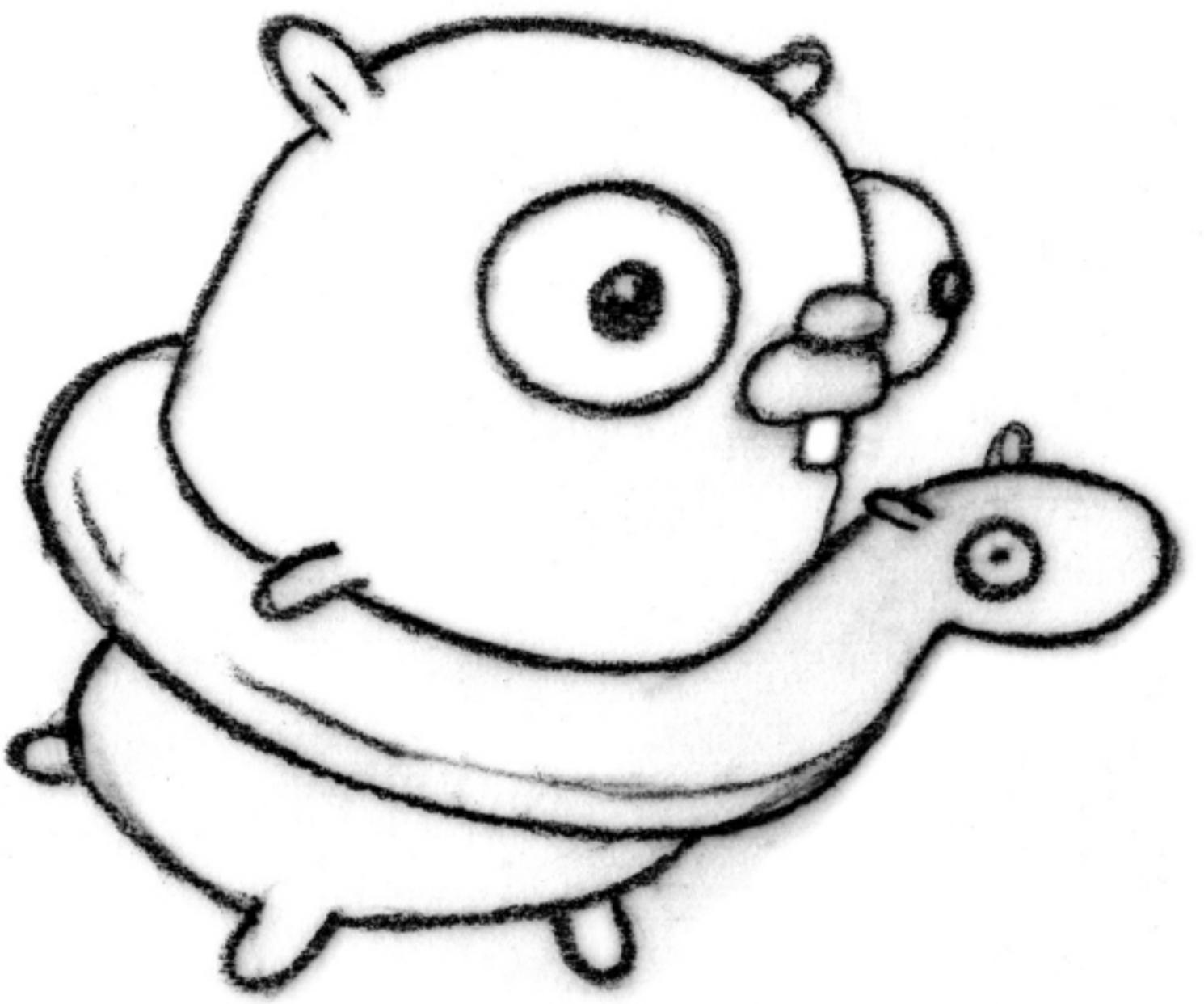










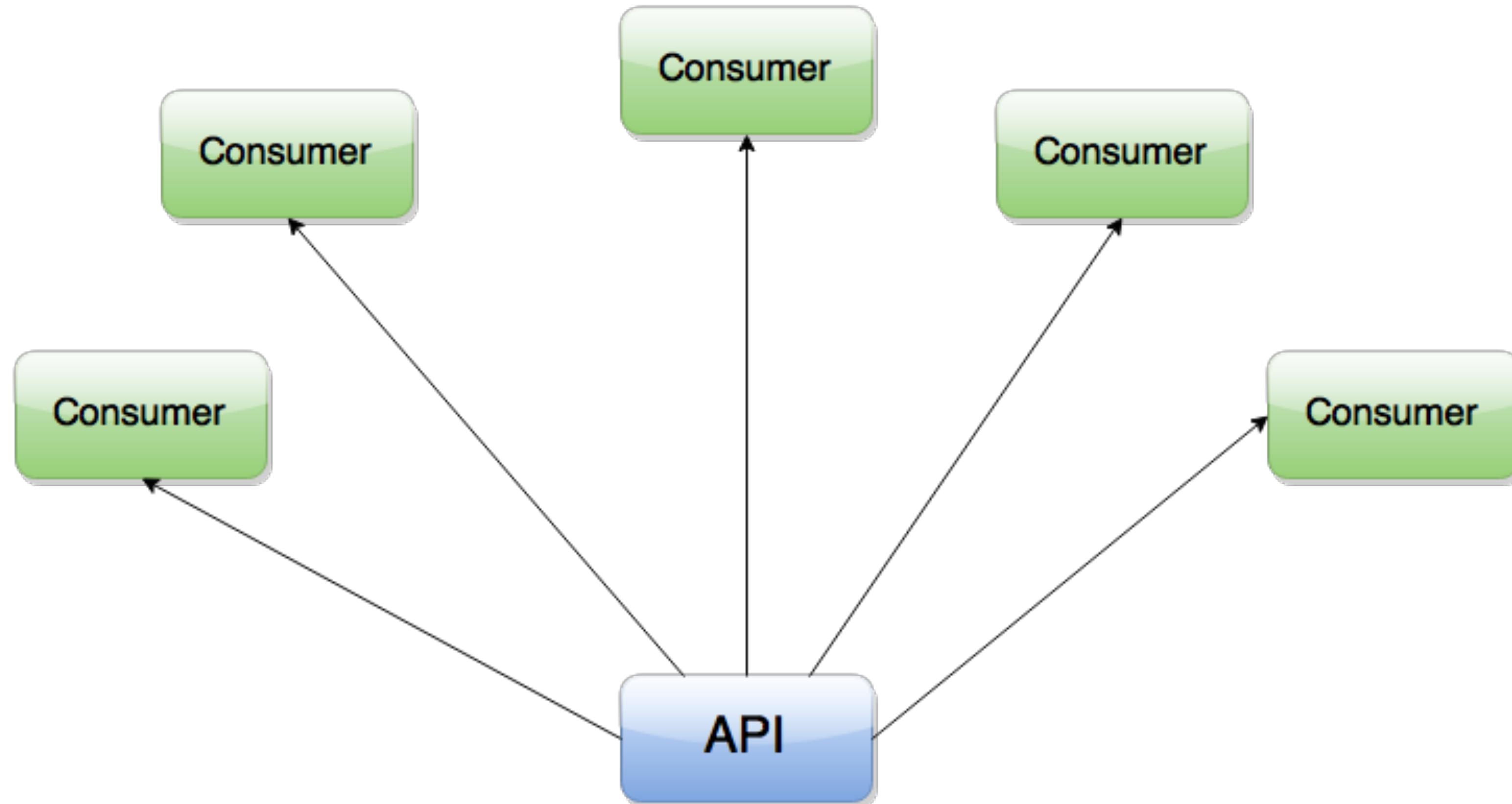


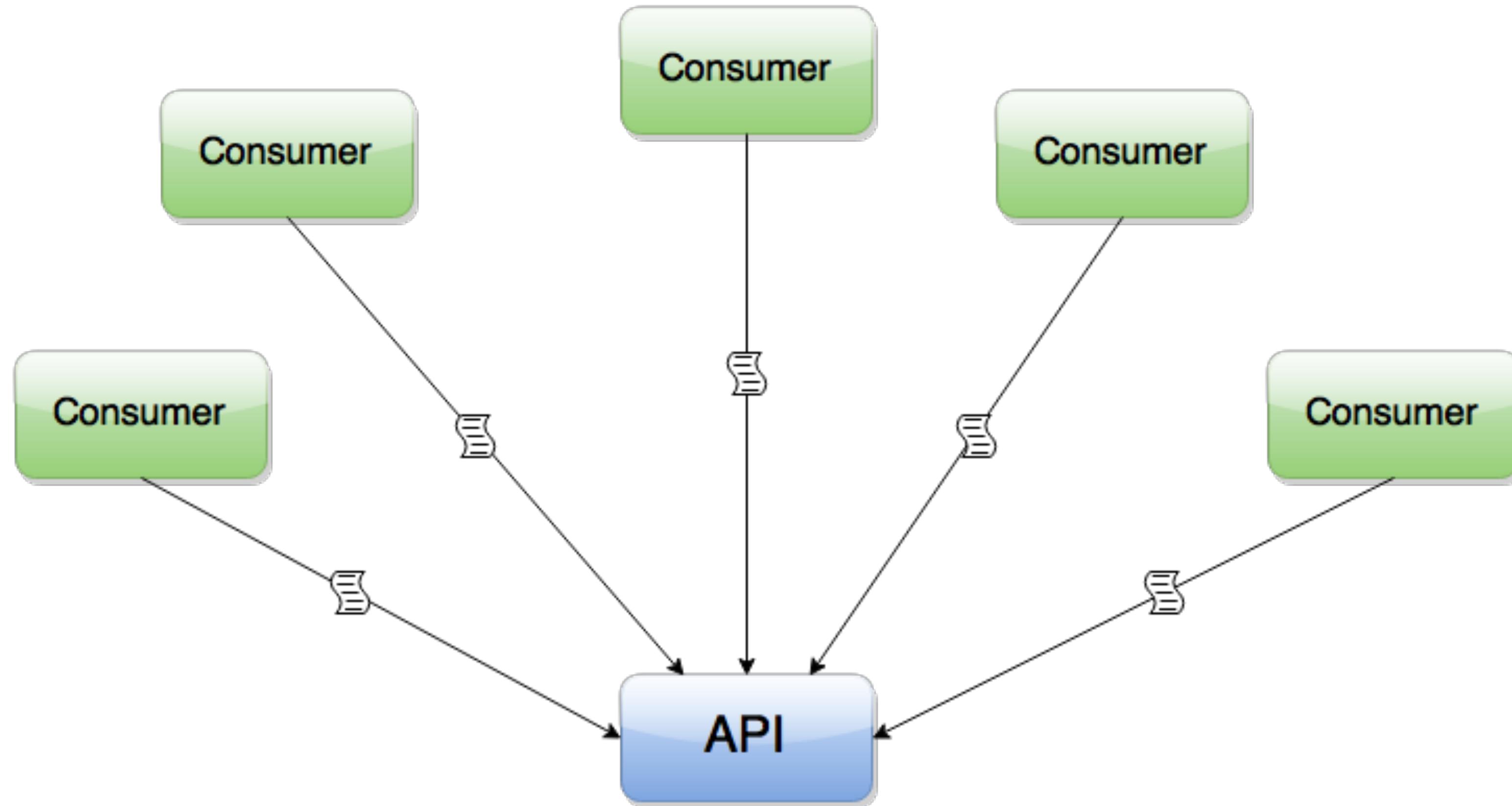
Code

08 – main.go, HTTP, endpoints

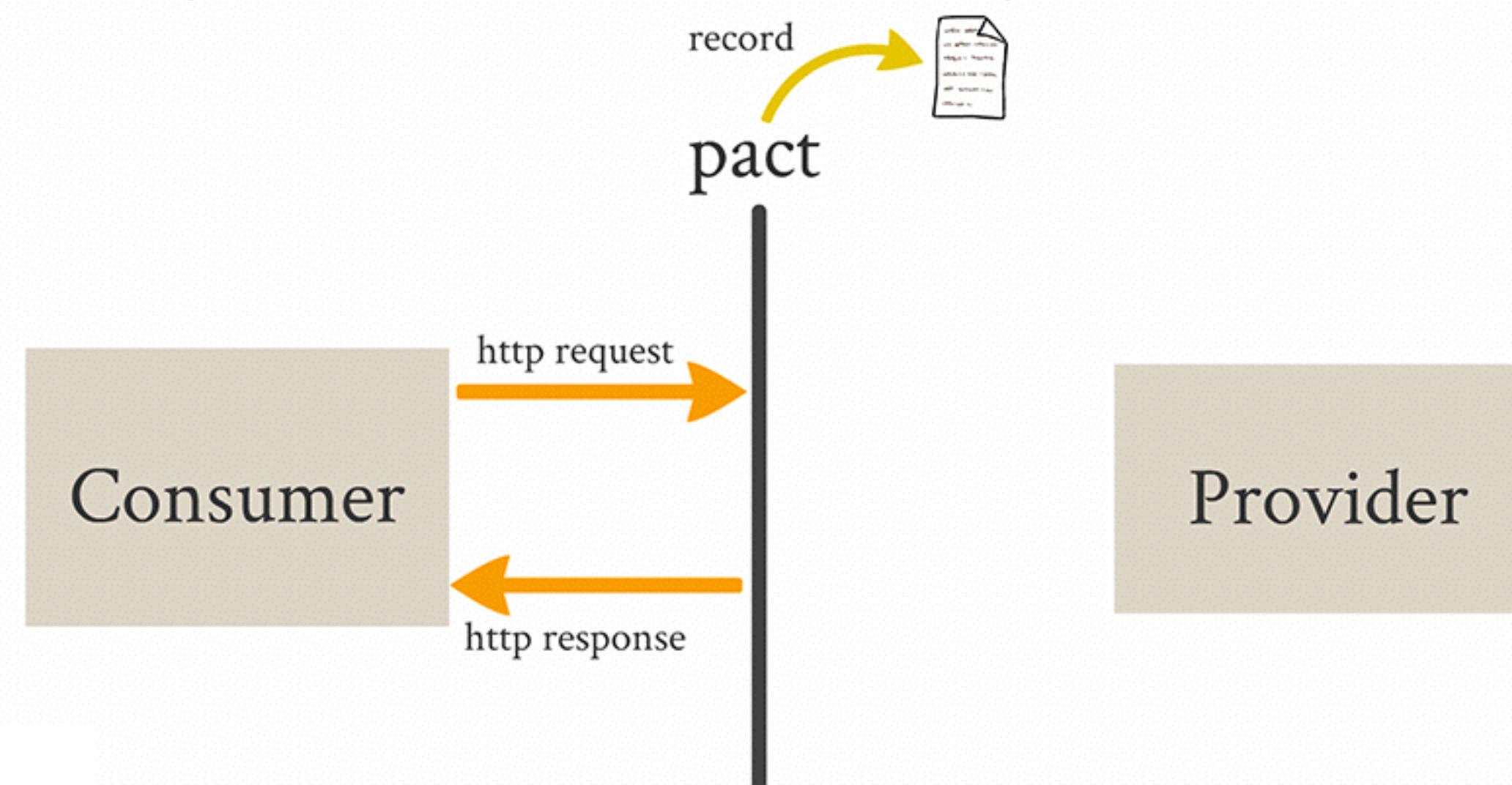
addsvc

Testing

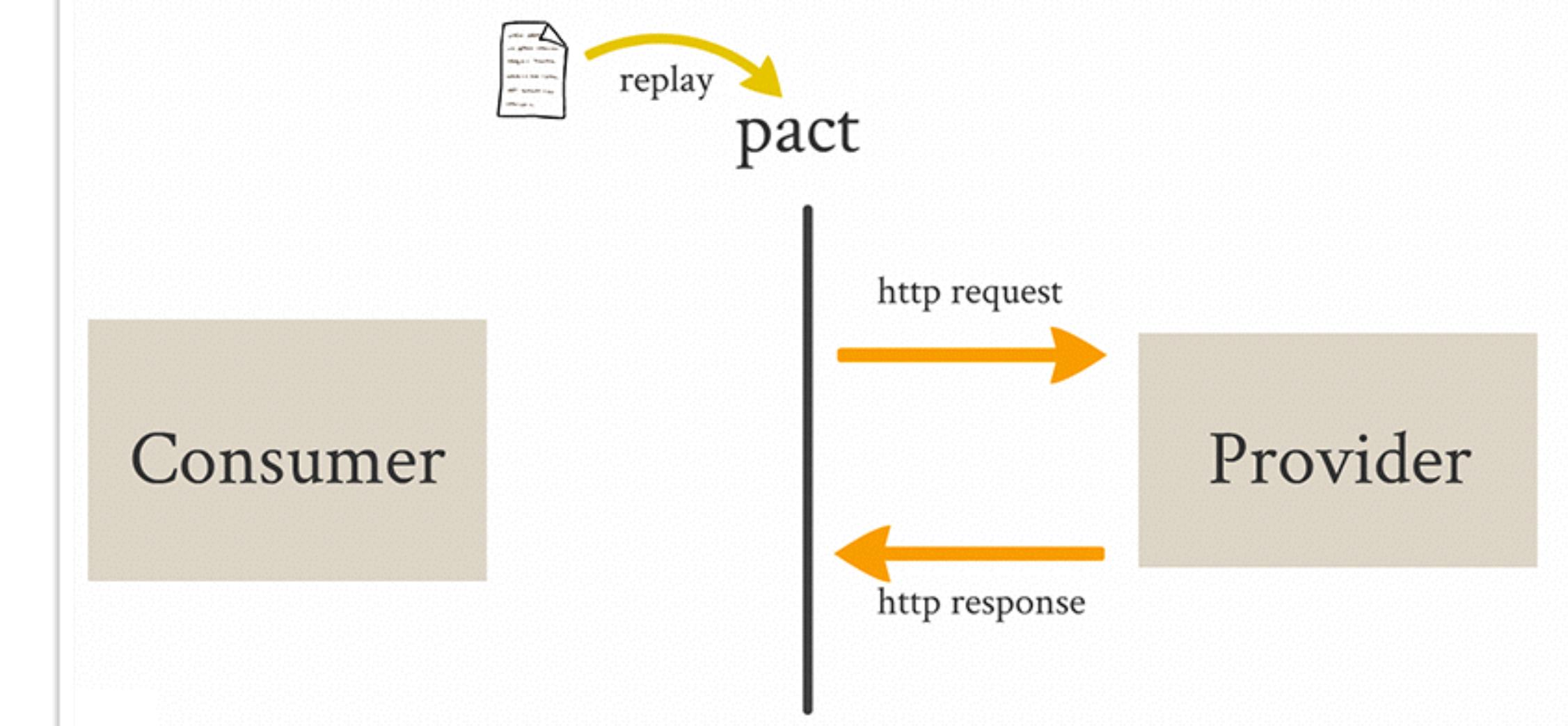


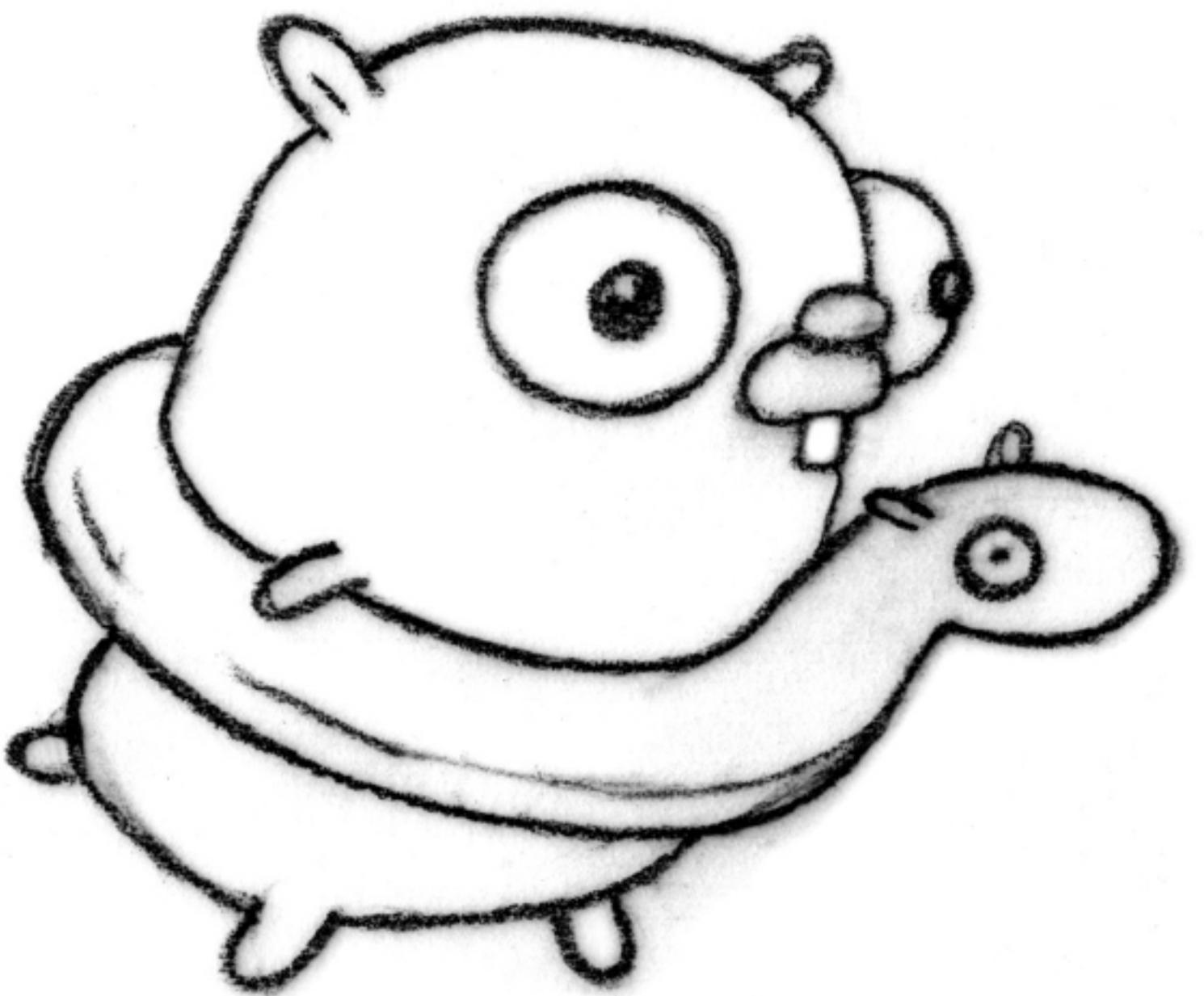


Step 1 - Define Consumer expectations



Step 2 - Verify expectations on Provider





Code

09 – stringsvc and Pact

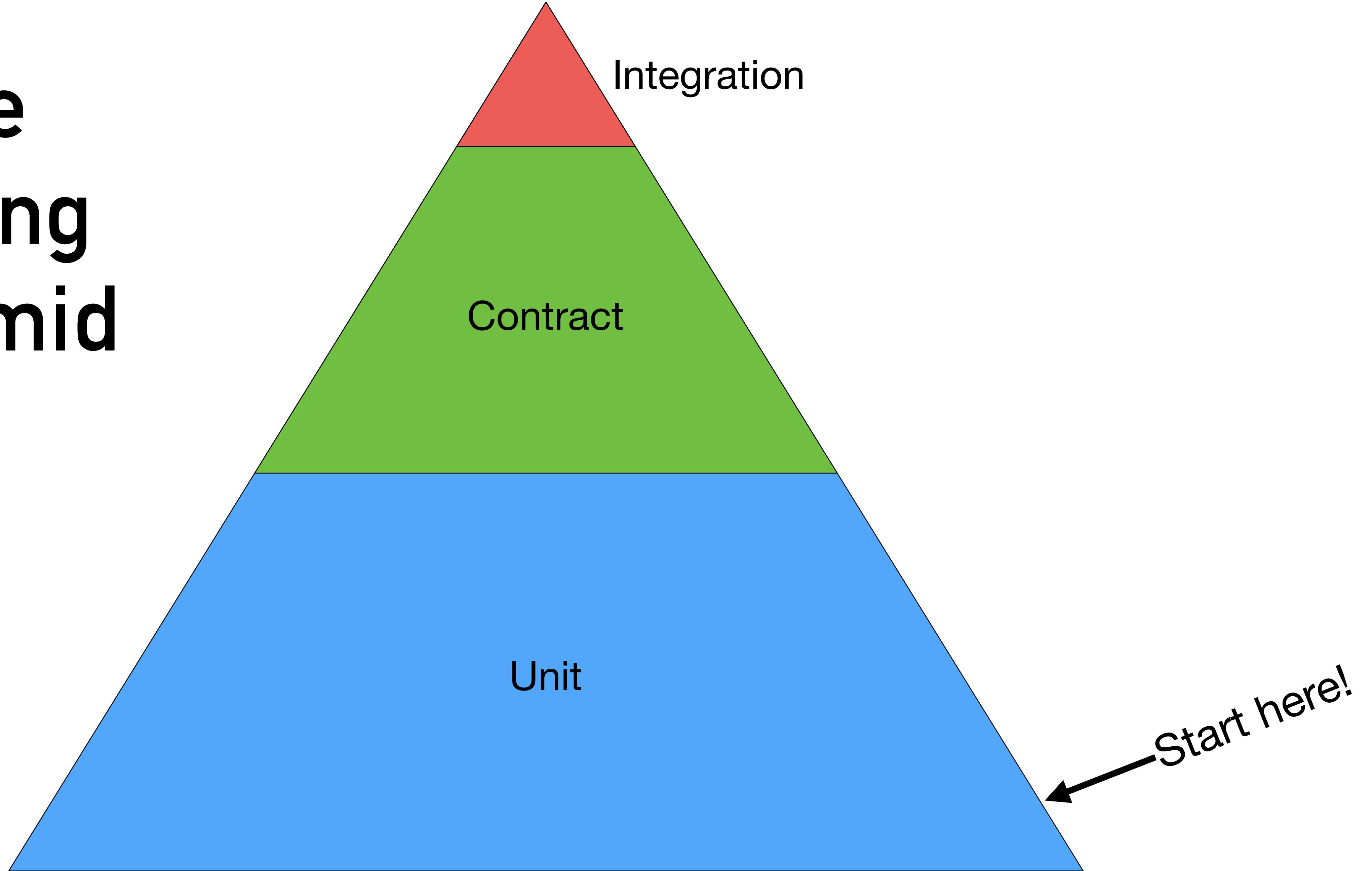
What about unit tests?

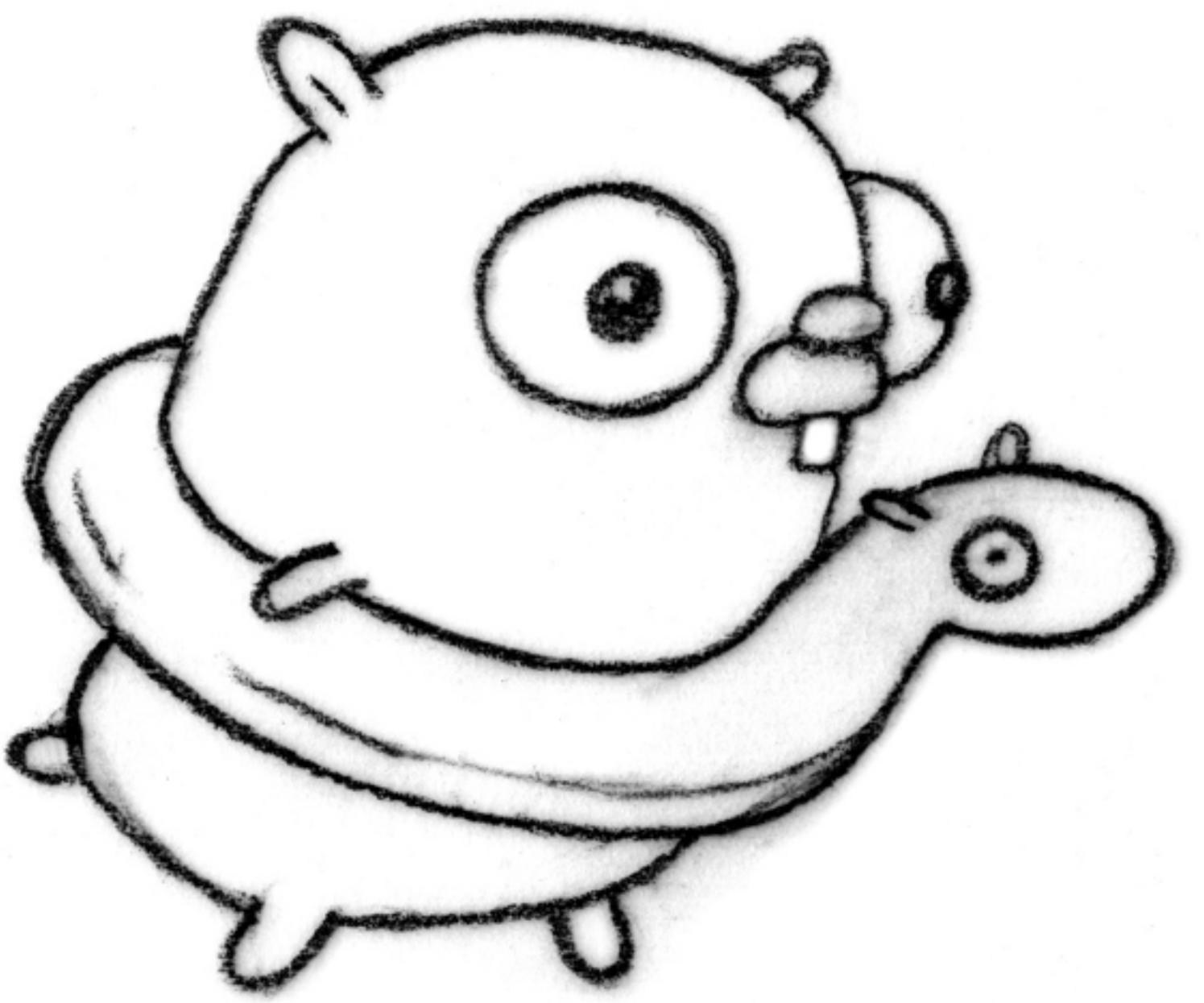
- **Contract tests** give us the freedom to refactor as necessary
- **Unit tests** make that process faster
- Test the thing being tested, and nothing more
- [Mitchell Hashimoto – Advanced Testing with Go](#)
- peter.bourgon.org/go-best-practices-2016/#testing

What about integration tests?

- **Integration tests** can mean different things
 - For a single service: test DB, etc. — use Docker
 - For many services: test interactions — already covered by contracts!
- With microservices, there is no steady state
- Google Testing Blog: Just say no to more end-to-end tests

The Testing Pyramid



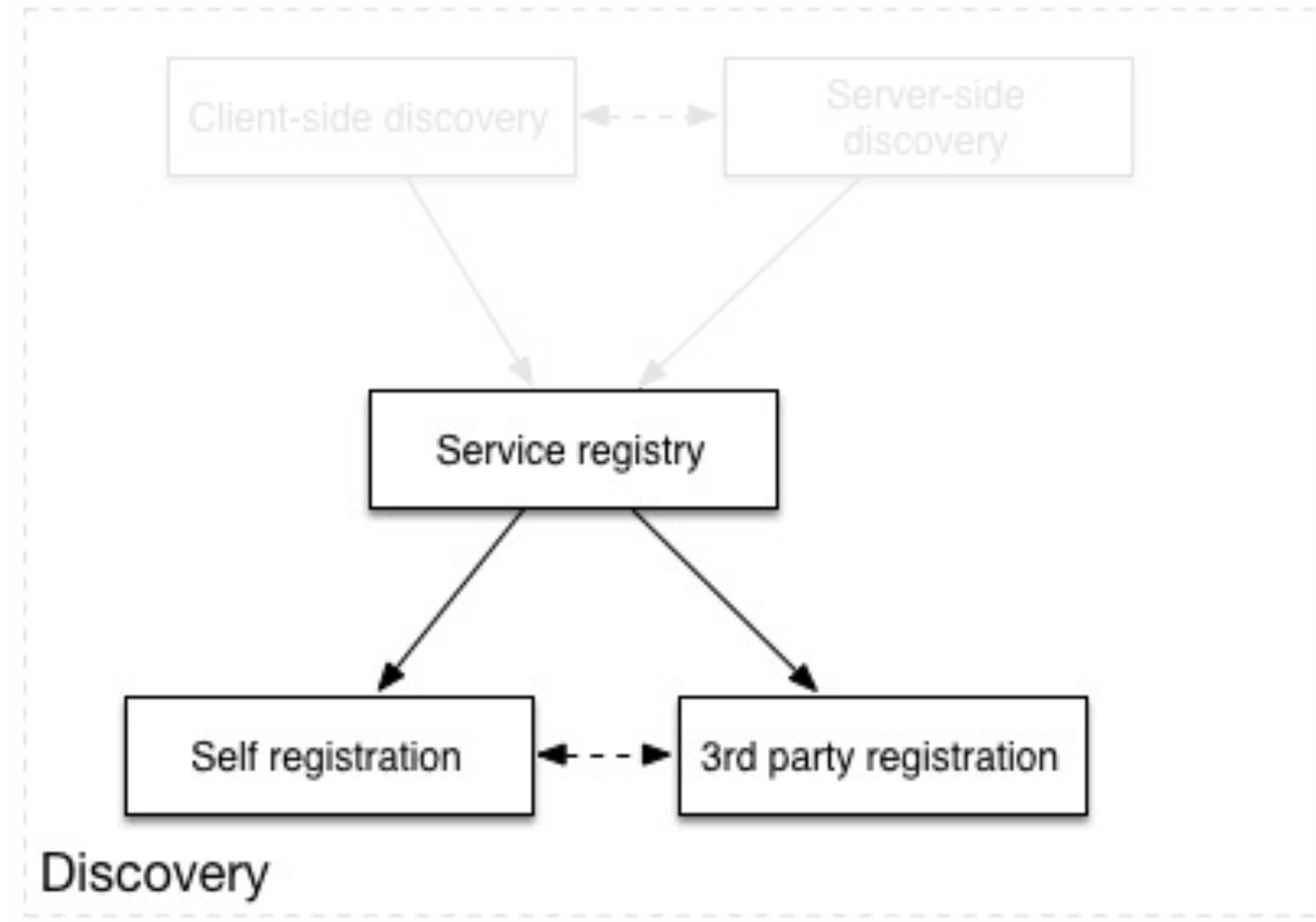


Code

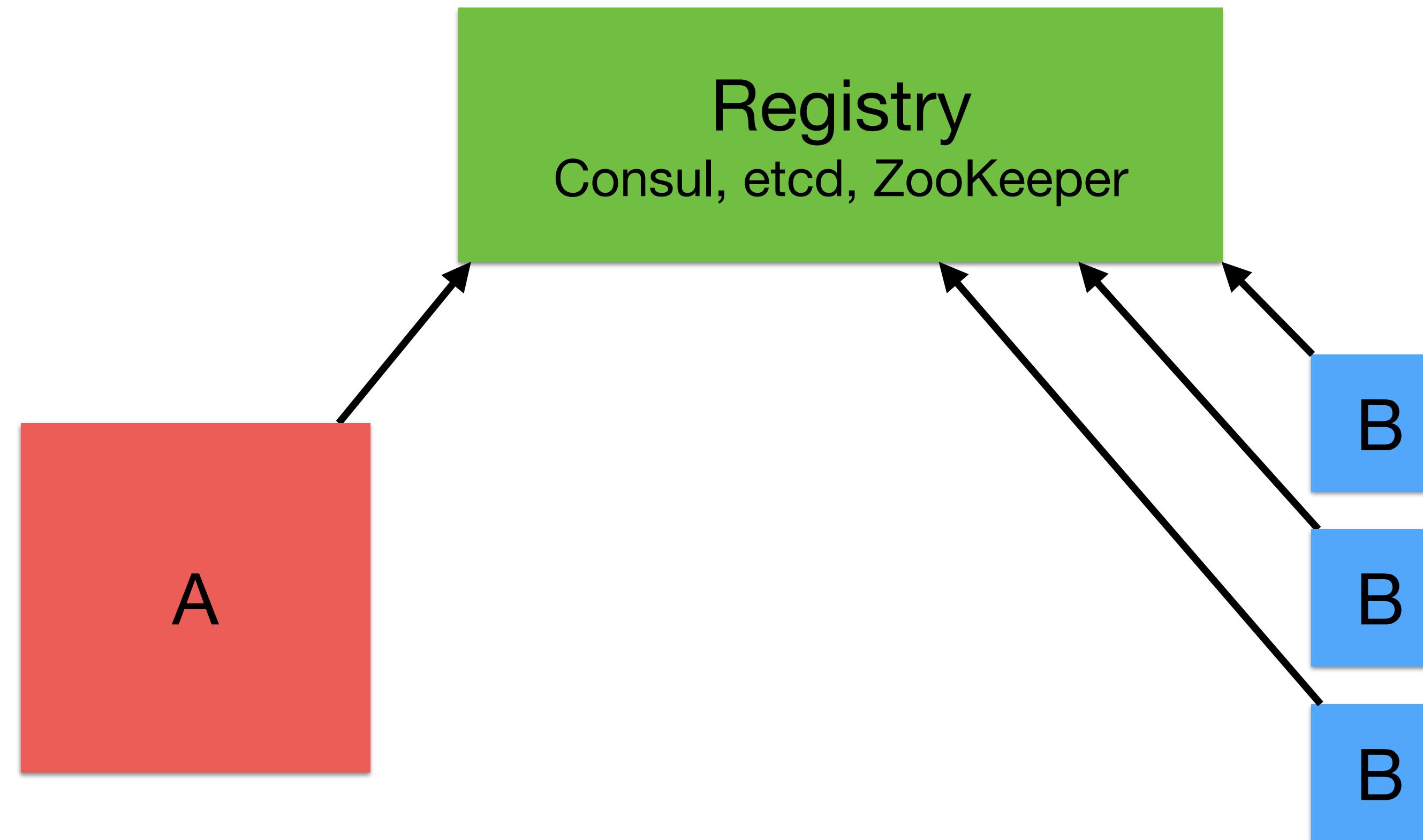
10 – Wiring test

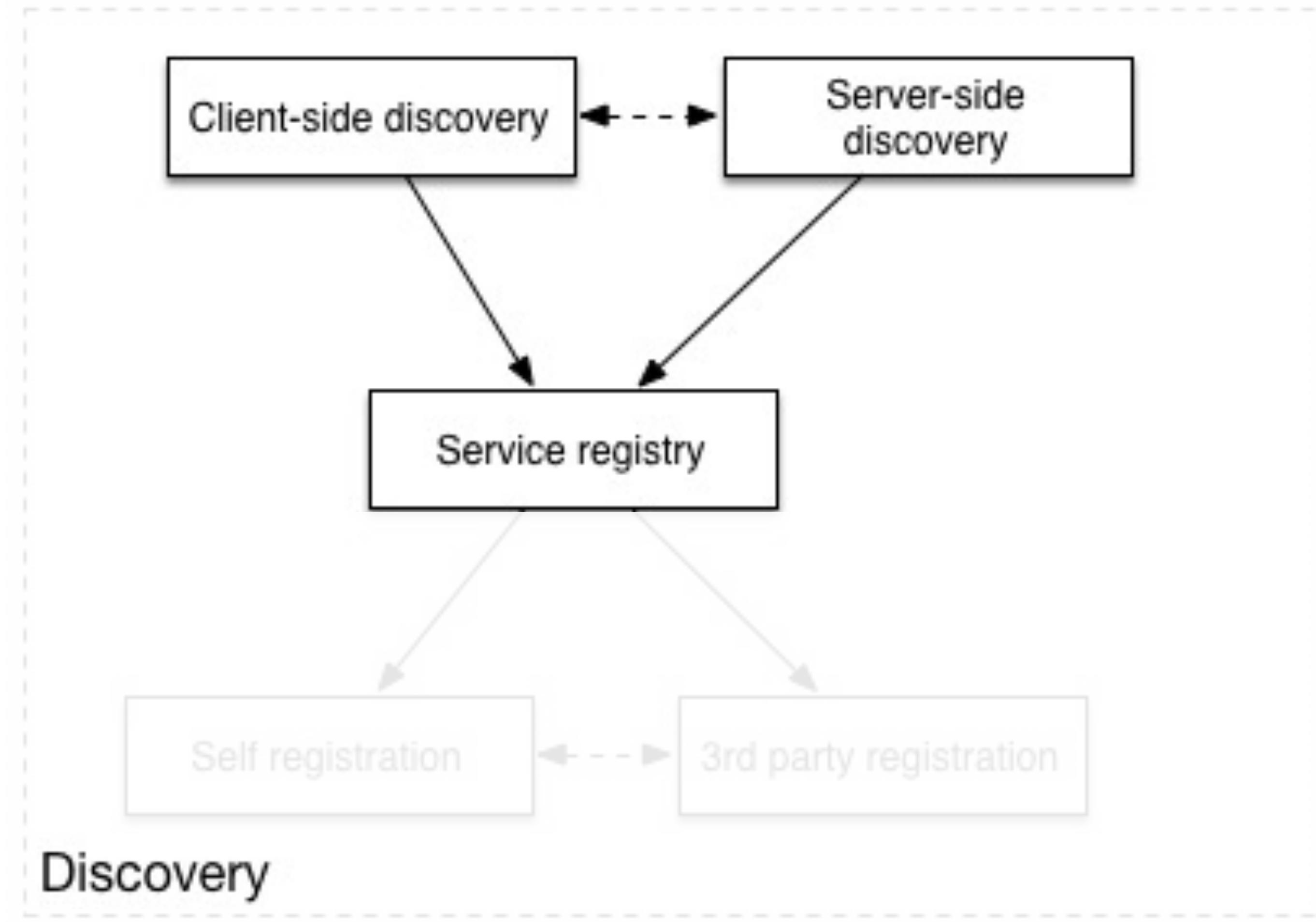
Bonus: service discovery

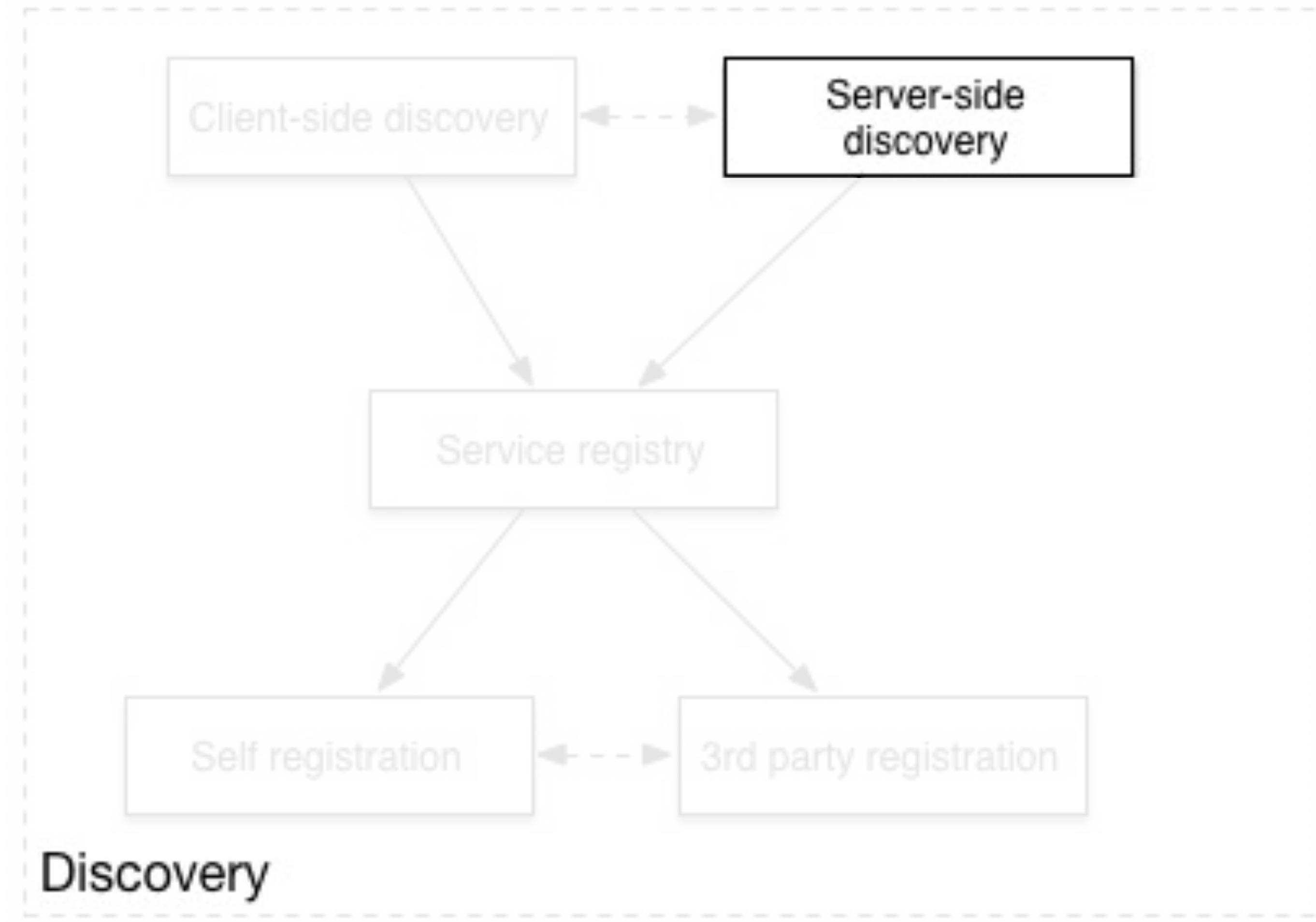
Go kit's package sd



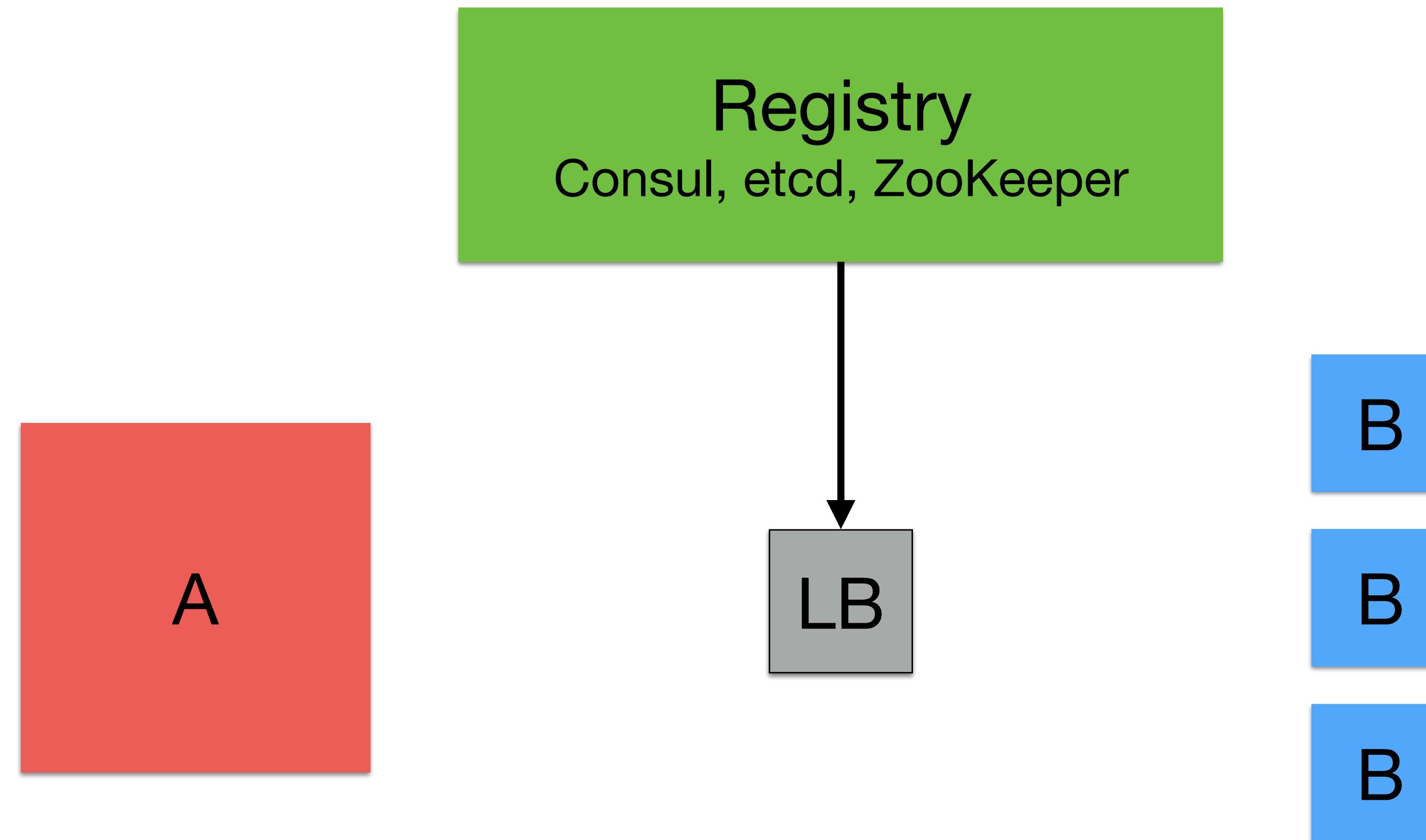
Registration



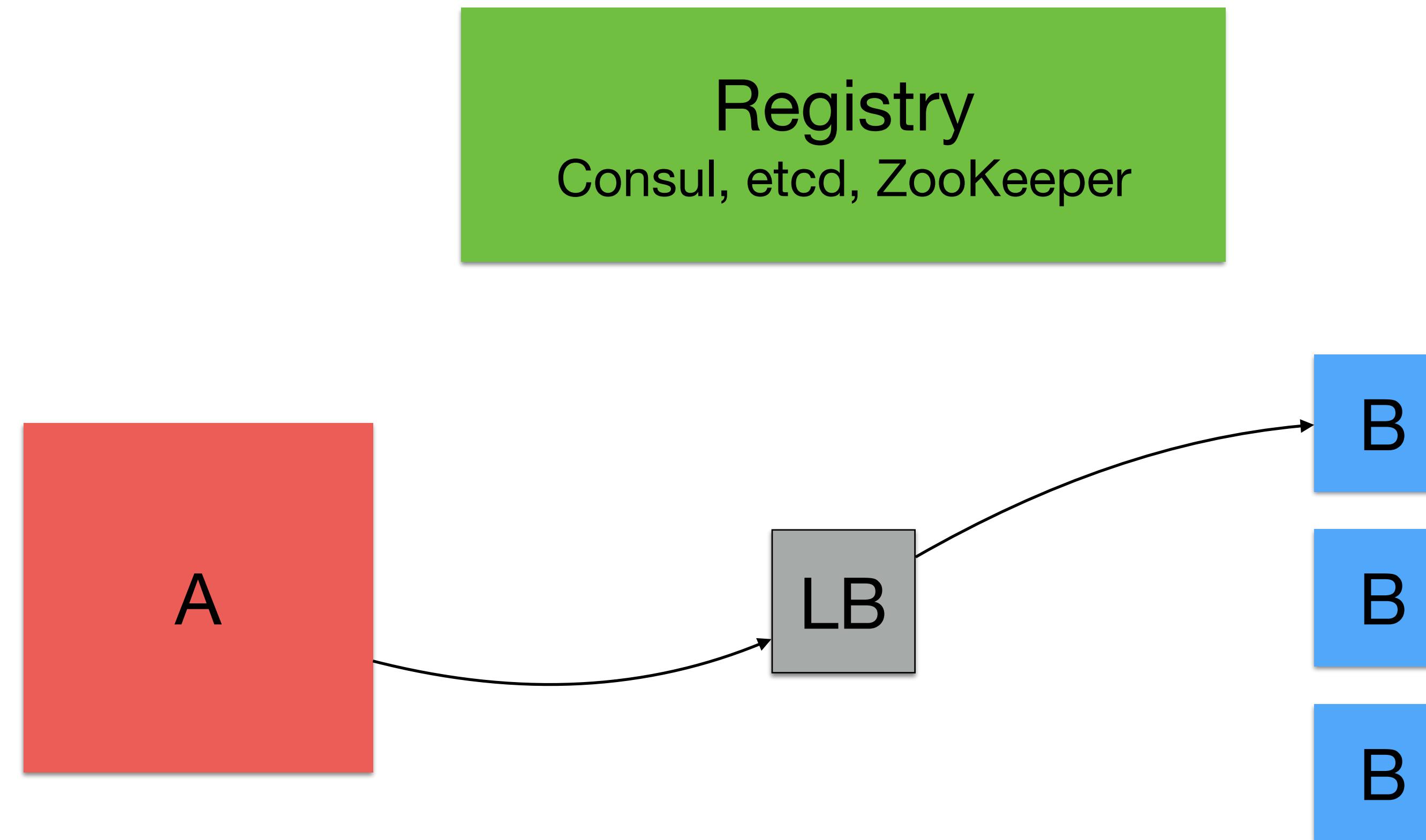




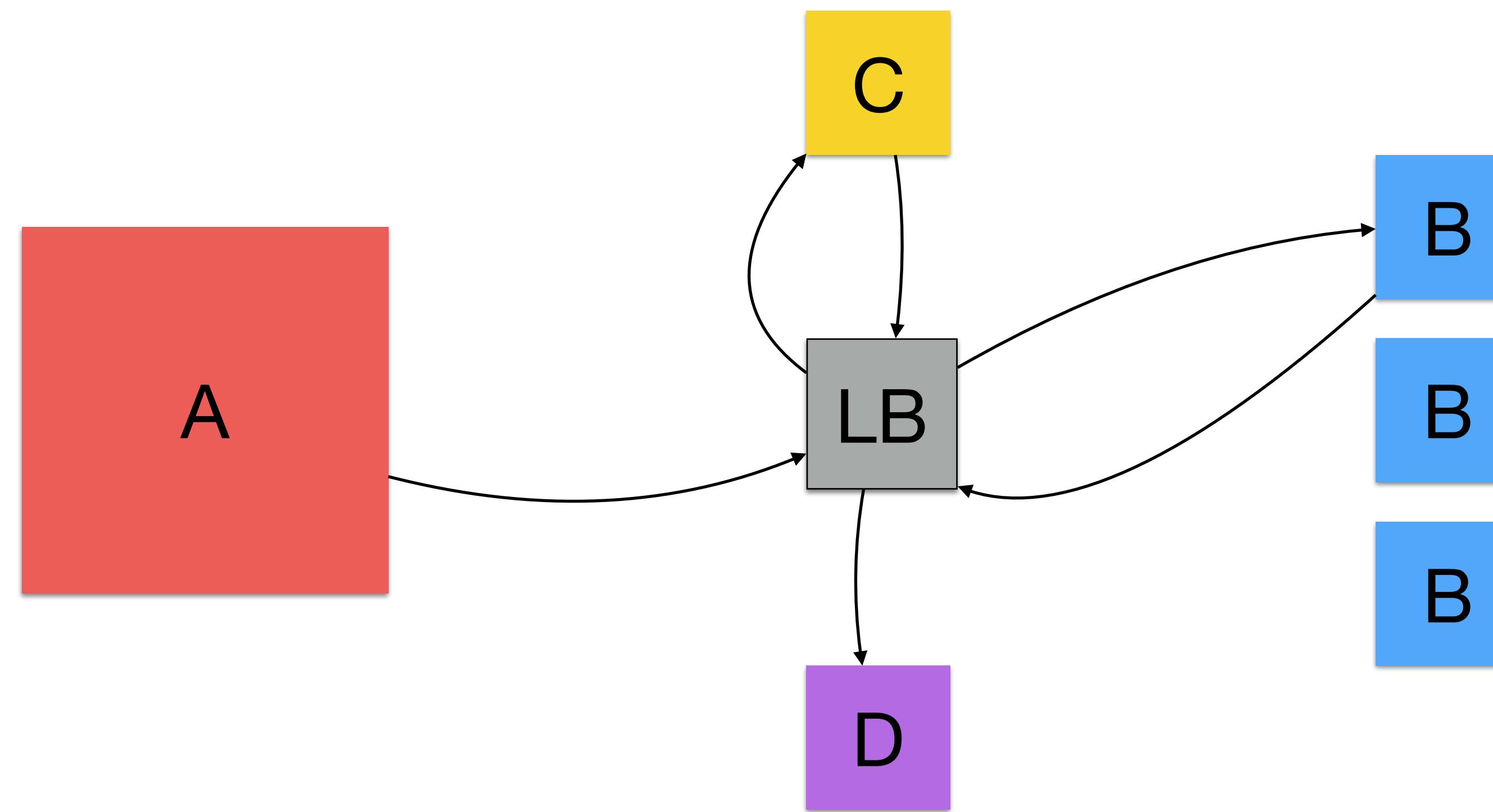
Central load balancer



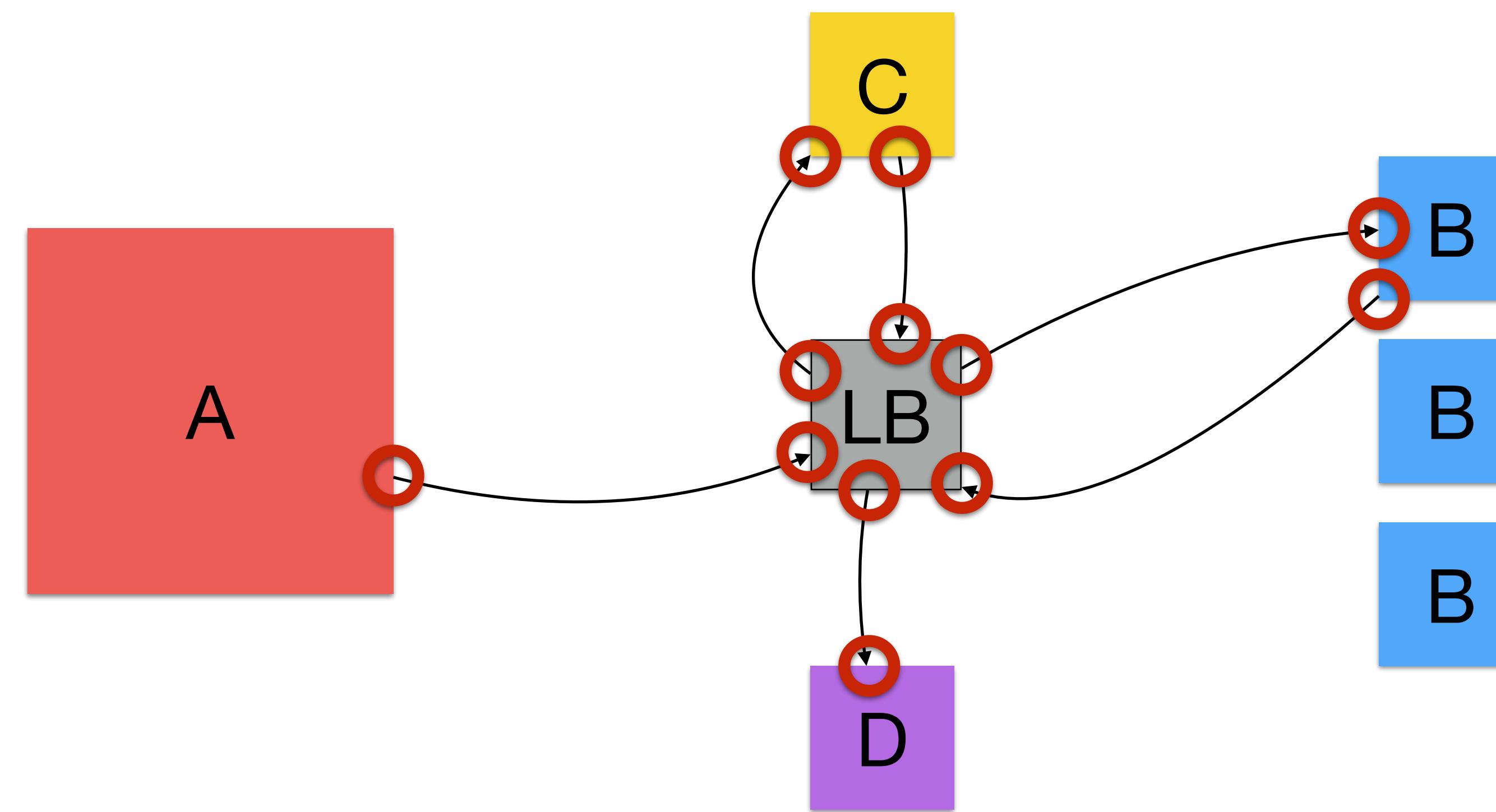
Central load balancer



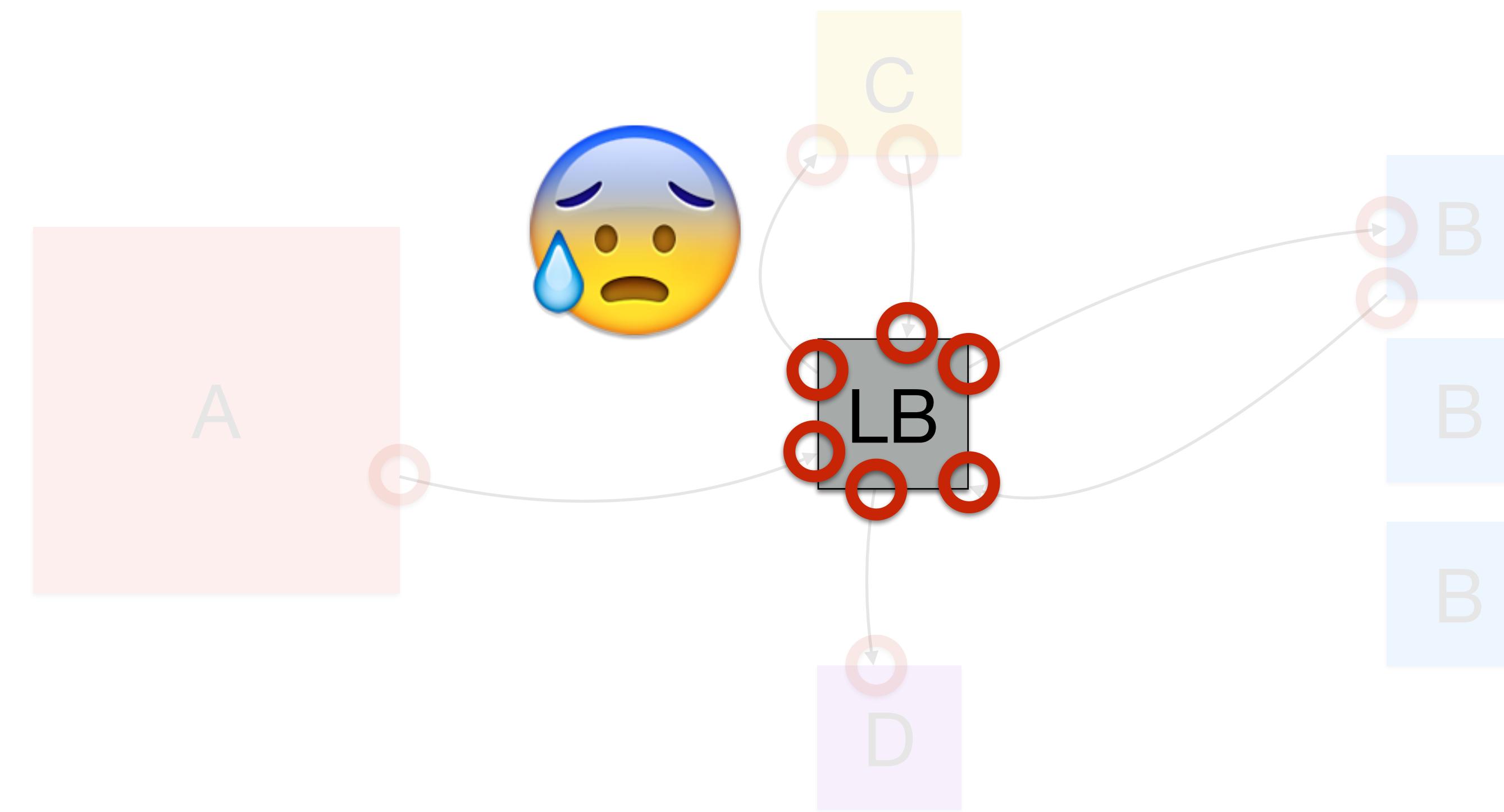
Central load balancer

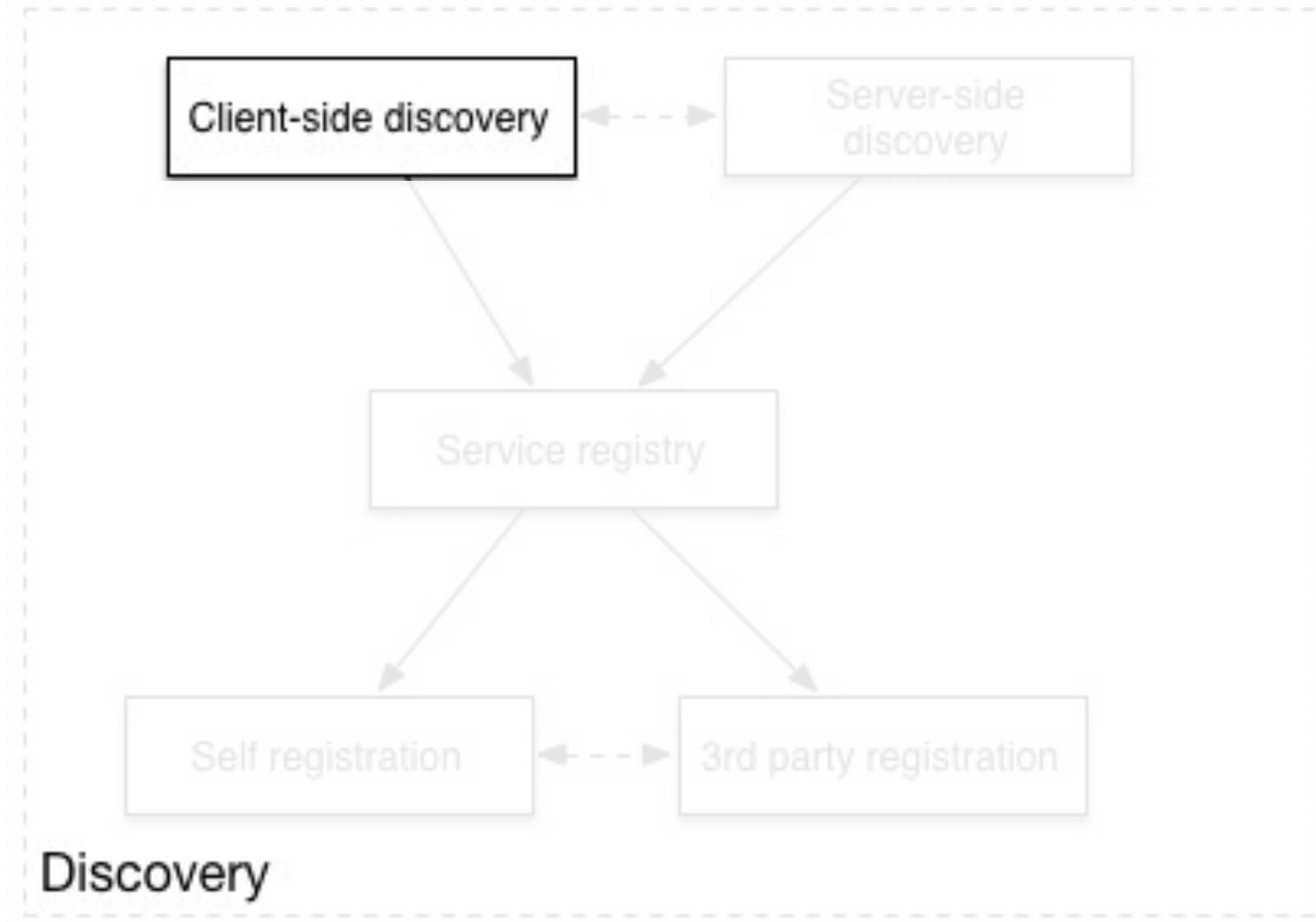


Central load balancer

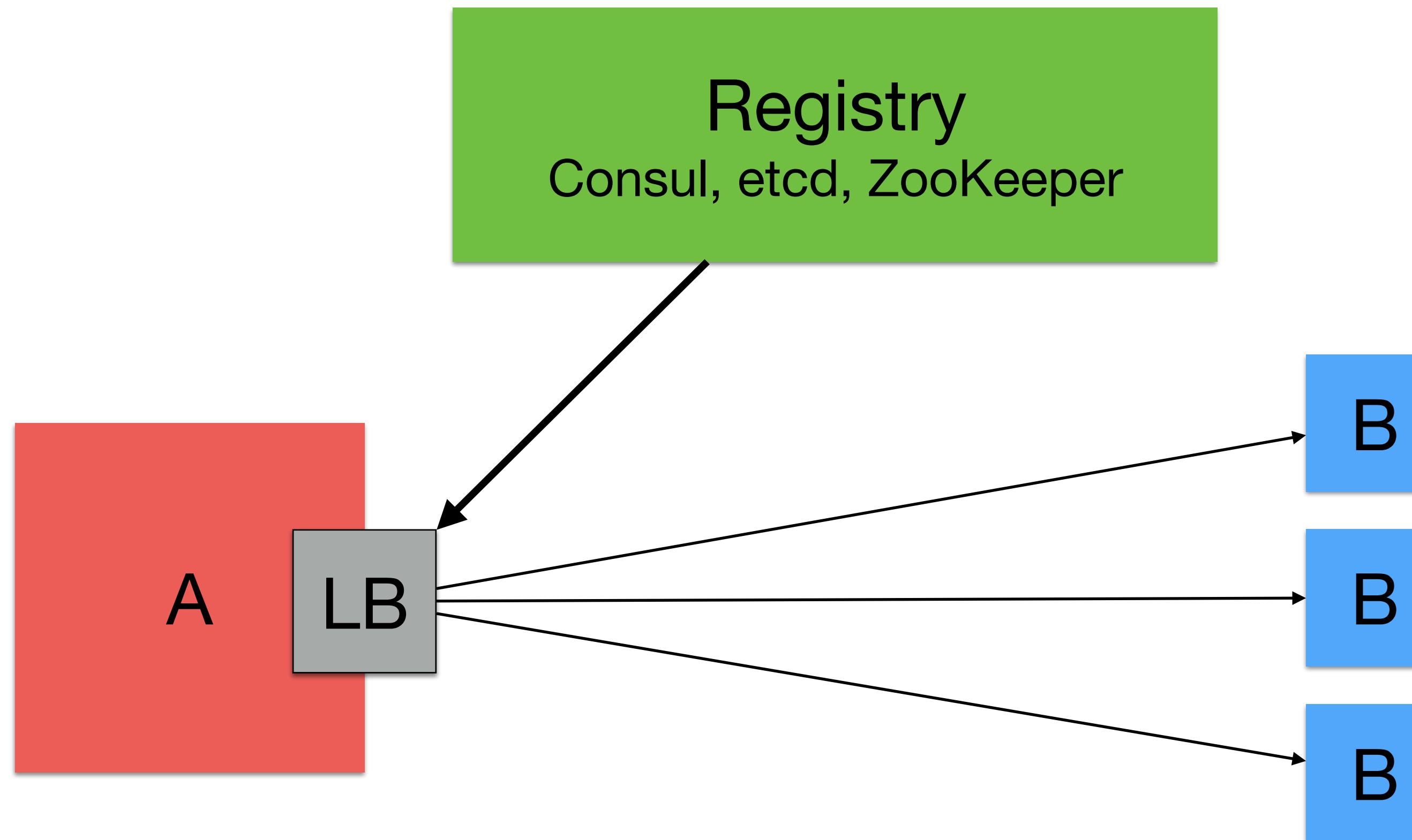


Central load balancer

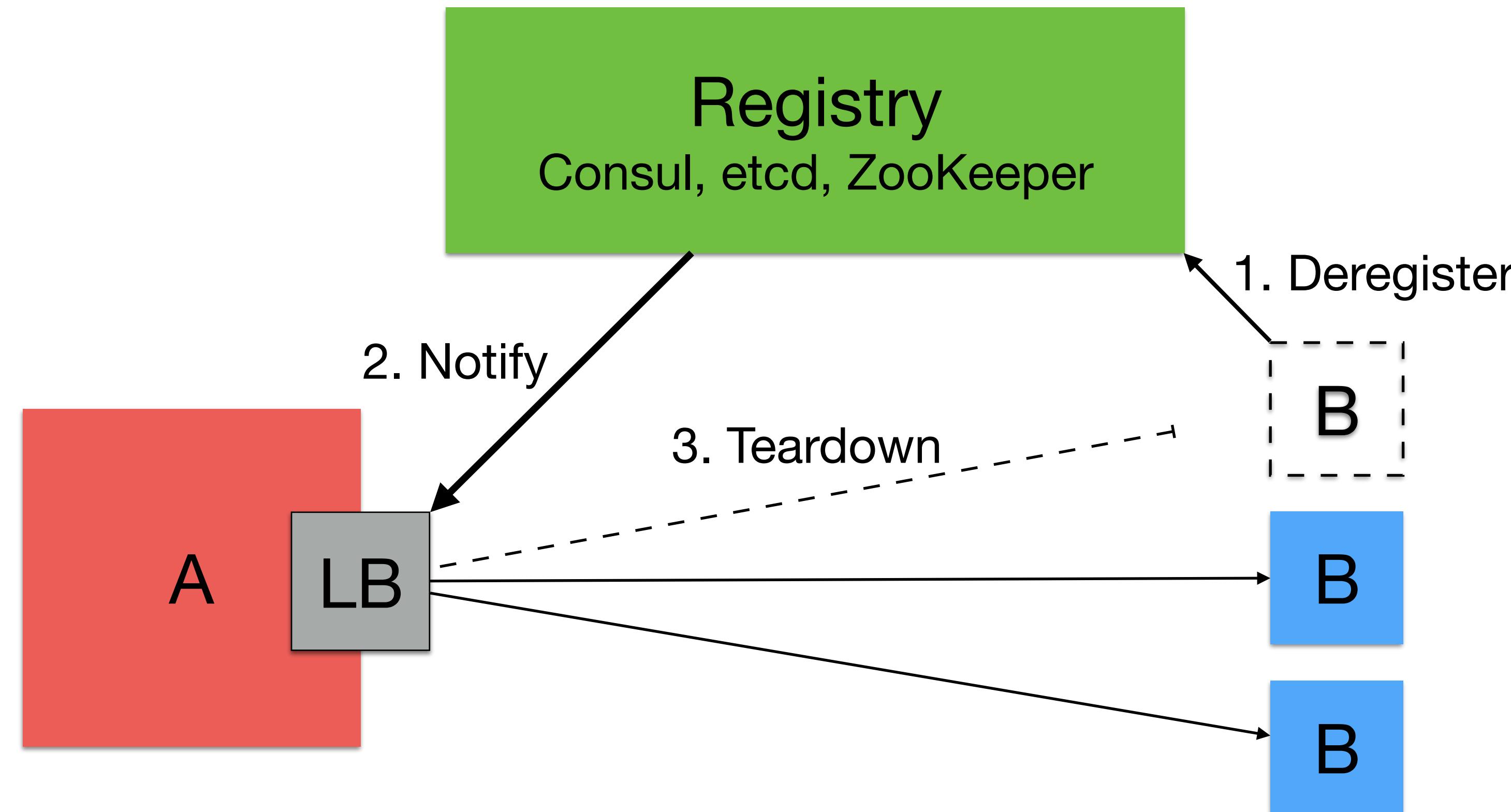


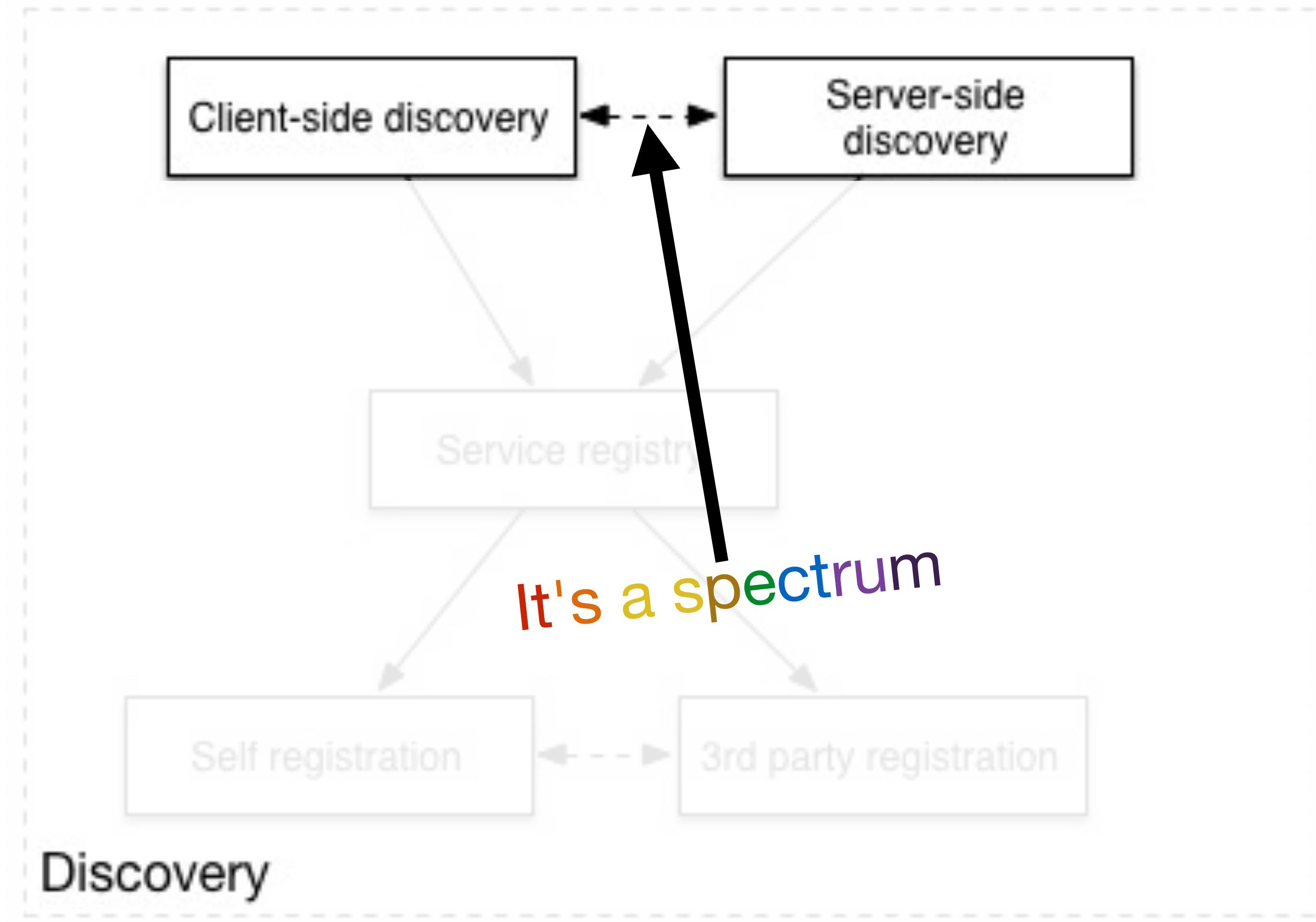


Client-side load balancer

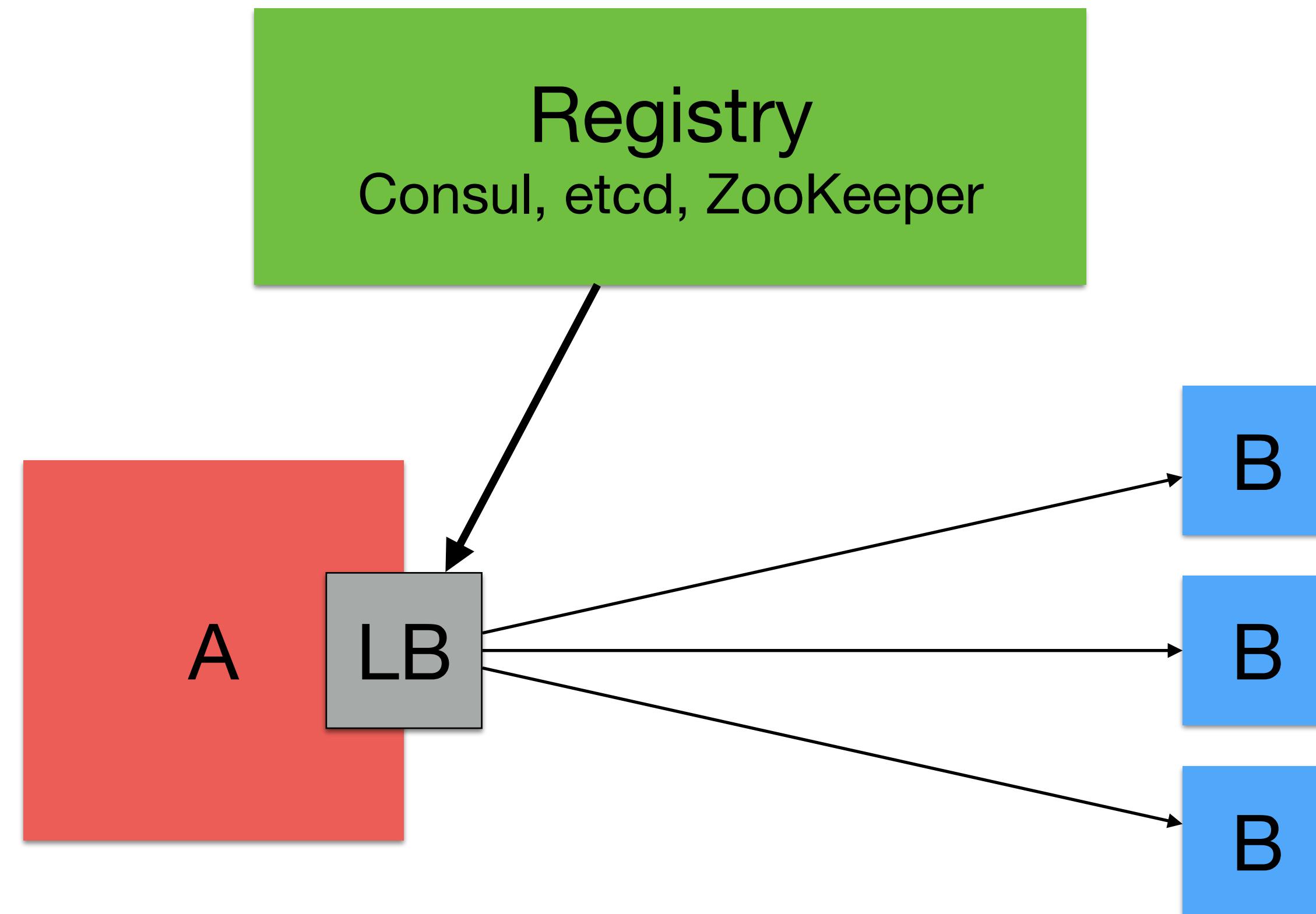


Deregistration

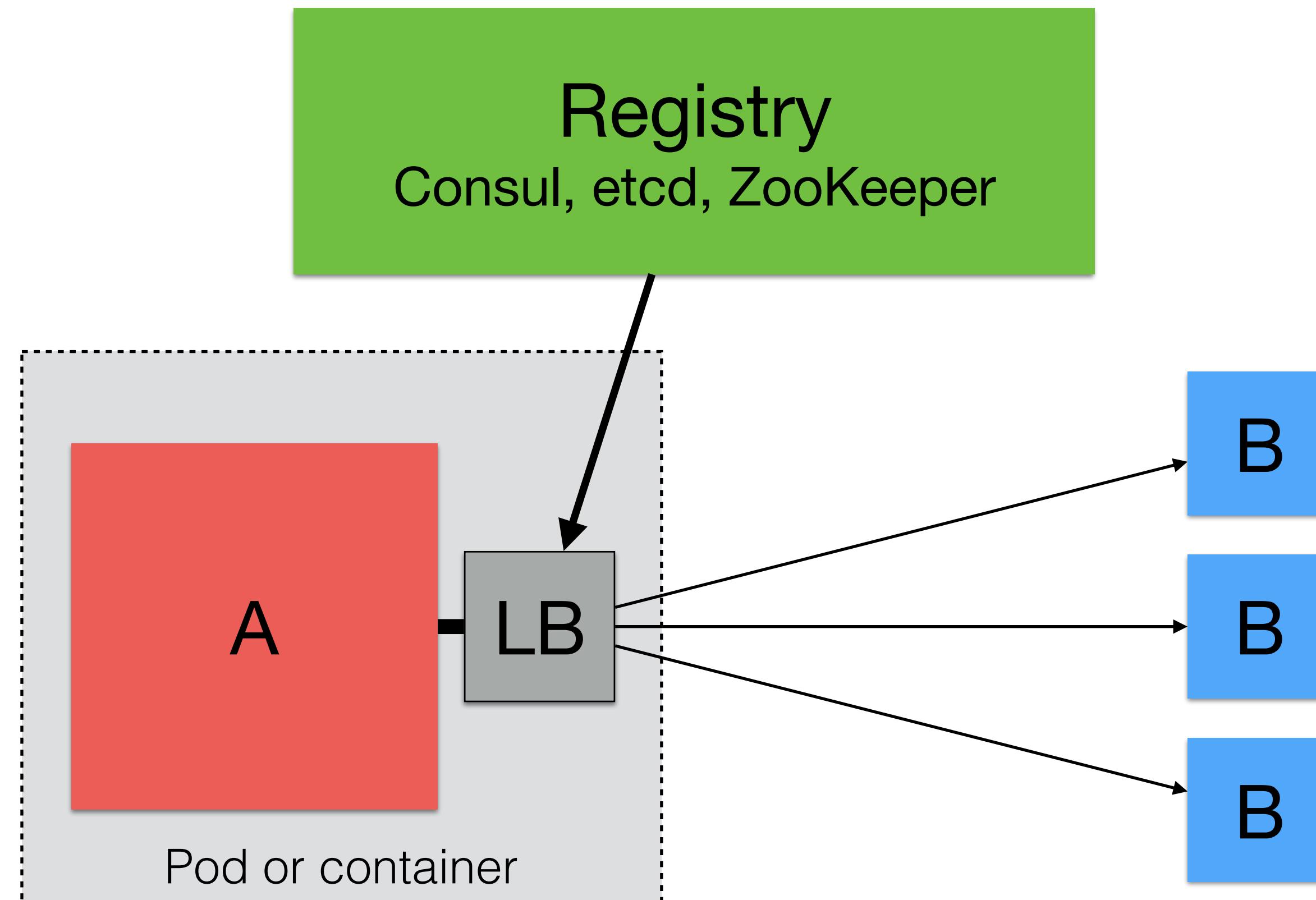




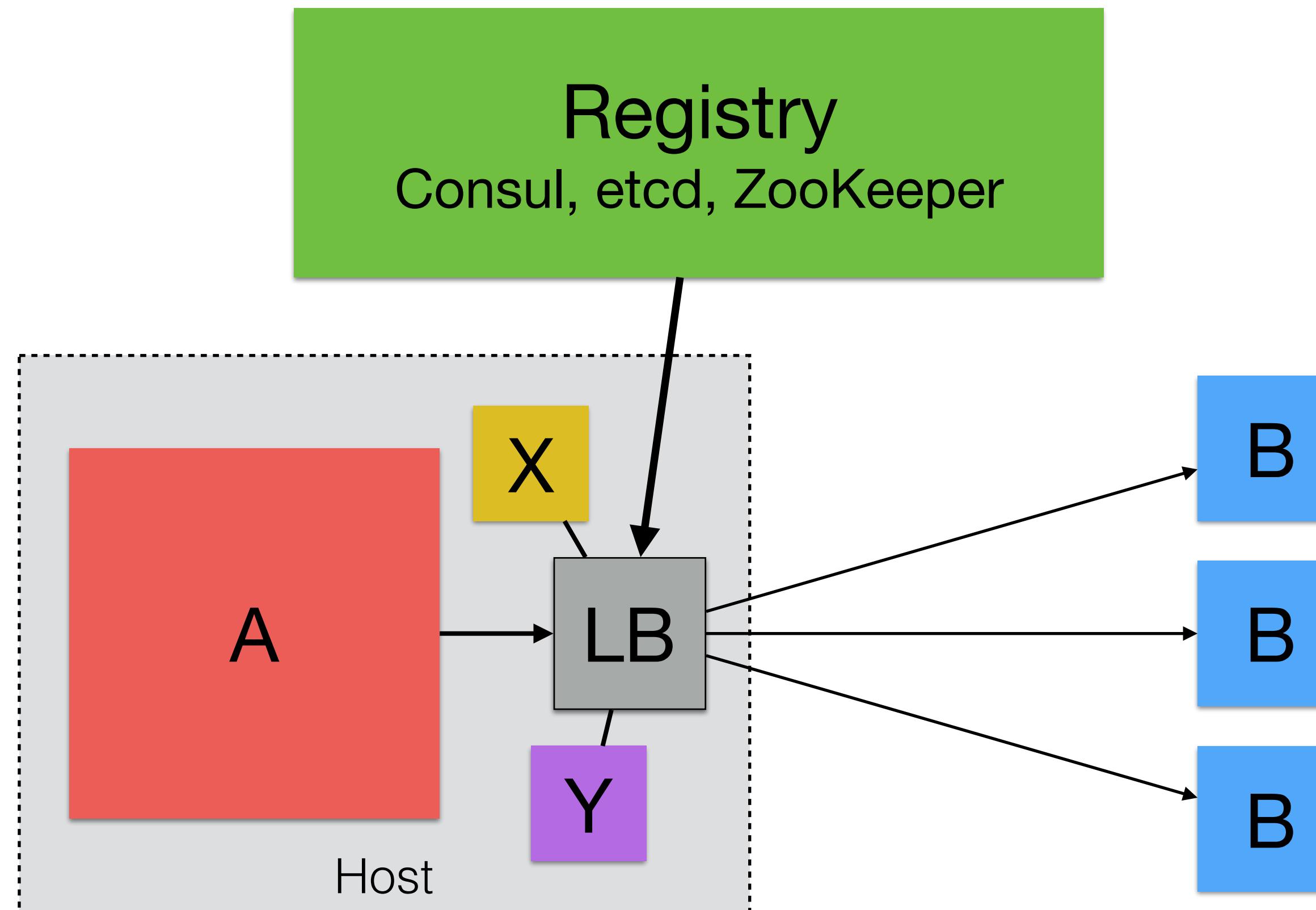
Client-side



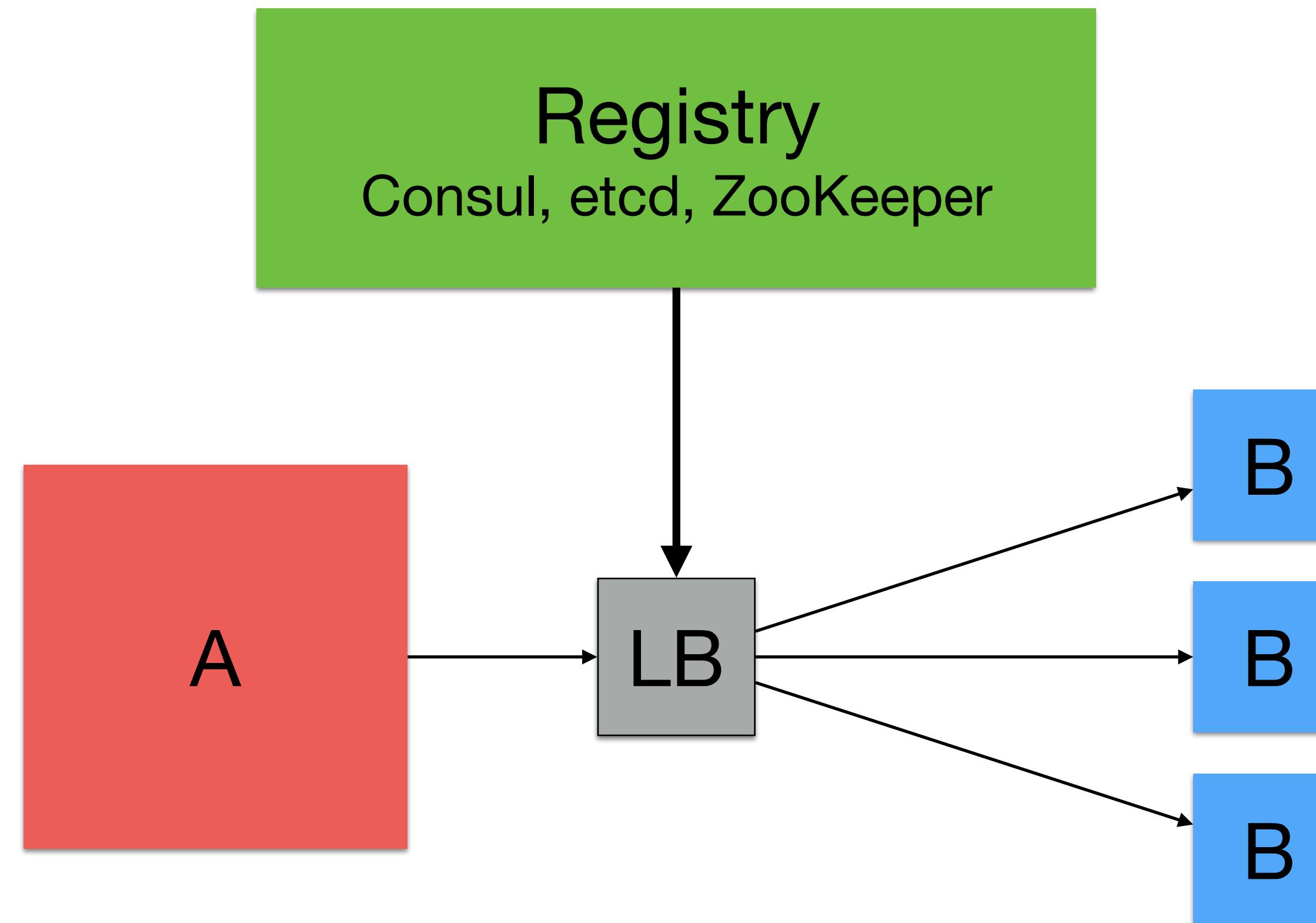
Linkerd sidecar



Per-host load balancers



Central loadbalancer



Speaker Deck Published on Nov 8, 2016

μCon 2016 / London / [@alexramirez](#)

A SERVICE DISCOVERY PRIMER

@alexramirez #mucon #servicediscovery

◀ ▶ share

A Service Discovery Primer / μCon 2016 by Alex Ramírez
Published November 8, 2016 in Technology

A Service Discovery Primer

Alex Ramírez



Service discovery at Stripe

Julia Evans on October 31, 2016 in Engineering

With so many new technologies coming out every year (like [Kubernetes](#) or [Habitat](#)), it's easy to become so entangled in our excitement about the future that we forget to pay homage to the tools that have been quietly supporting our production environments. One such tool we've been using at Stripe for several years now is [Consul](#). Consul helps discover services (that is, it helps us navigate the thousands of servers we run with various services running on them and tells us which ones are up and available for use). This effective and practical architectural choice wasn't flashy or entirely novel, but has served us dutifully in our continued mission to provide reliable service to our users around the world.

We're going to talk about:

- What service discovery and Consul are,
- How we managed the risks of deploying a critical piece of software,

Service Discovery at Stripe

Julia Evans



Thanks! Hooray!

QCon SF · 10 November 2016

