Go kit: a toolkit for microservices

6 October 2016 · GOTO Copenhagen · @peterbourgon

Perhaps better titled

How to do microservices

With Go kit as an implementation detail

Prerequisites

- Go installed — http://golang.org — brew install go
- $ go version
- $ export GOPATH=$HOME/gocode # or somewhere else
- $ go get github.com/peterbourgon/go-microservices

Who am I?

Who are you?

The microservices landscape

Toward a shared context

Size

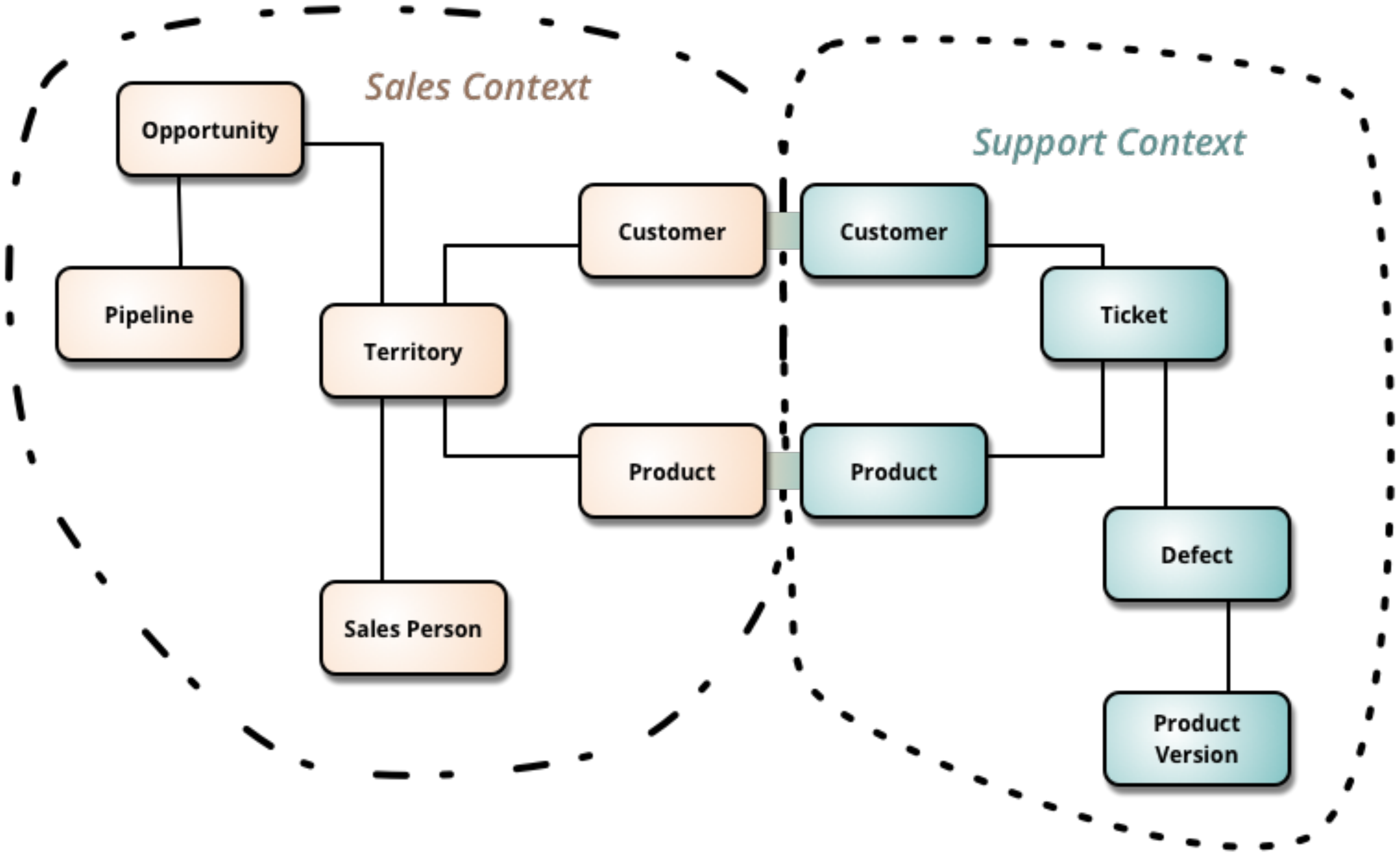
A **single programmer** can design, implement,
deploy and maintain a microservice.
—Fred George

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*



martinfowler.com/bliki/BoundedContext.html

Operation

Microservices **built & deployed independently**.
Stateless, with state as backing services.
—12Factor.net

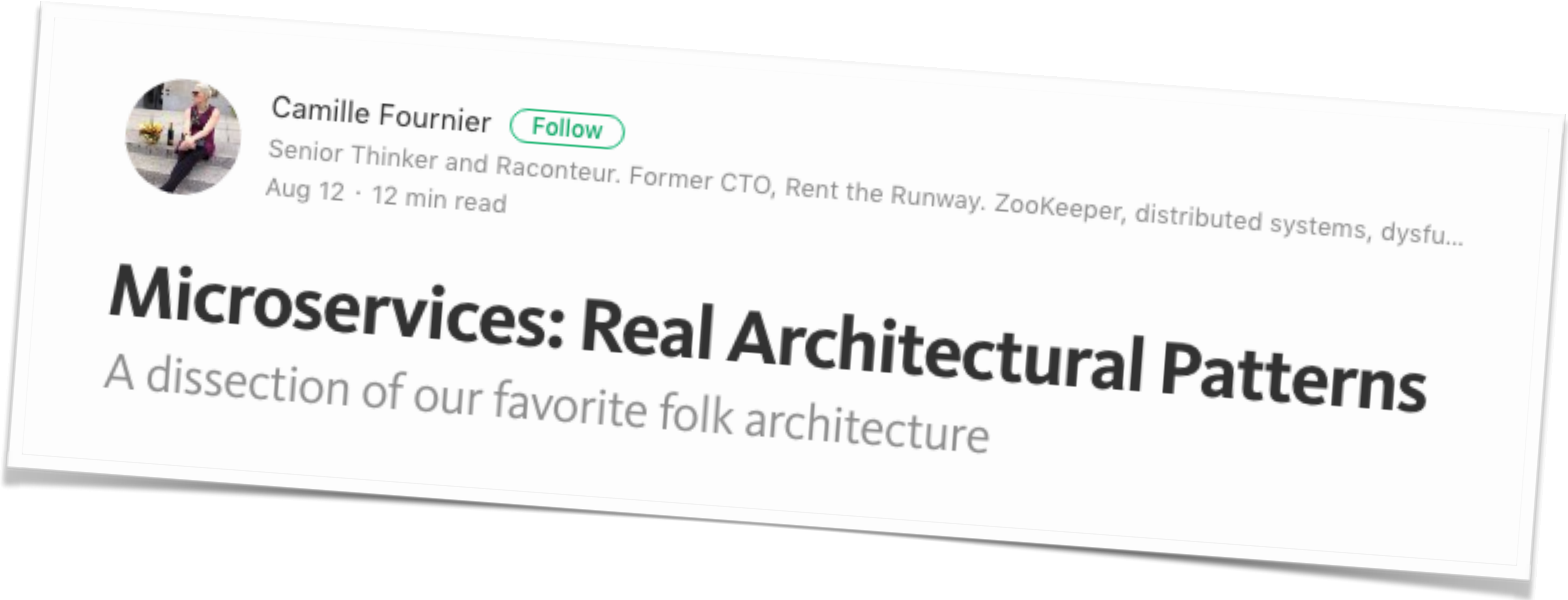
Addressable through a **service discovery** system.
—Chris Richardson

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...



Camille Fournier

Senior Thinker and Raconteur. Former CTO, Rent the Runway. ZooKeeper, distributed systems, dysfu...

Aug 12 · 12 min read

Microservices: Real Architectural Patterns

A dissection of our favorite folk architecture

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
- Product velocity stalled

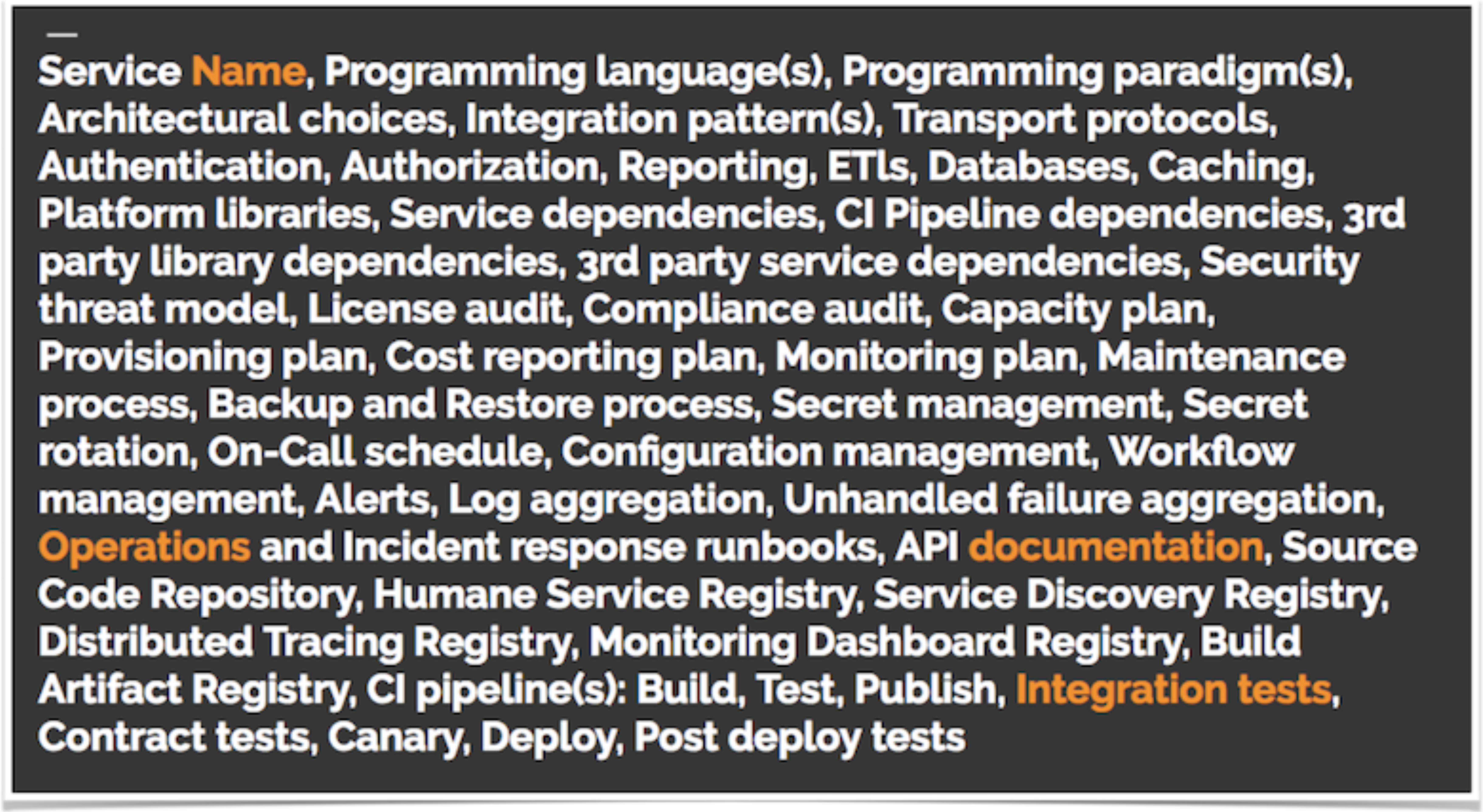
Problems caused

- 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???

From one to many

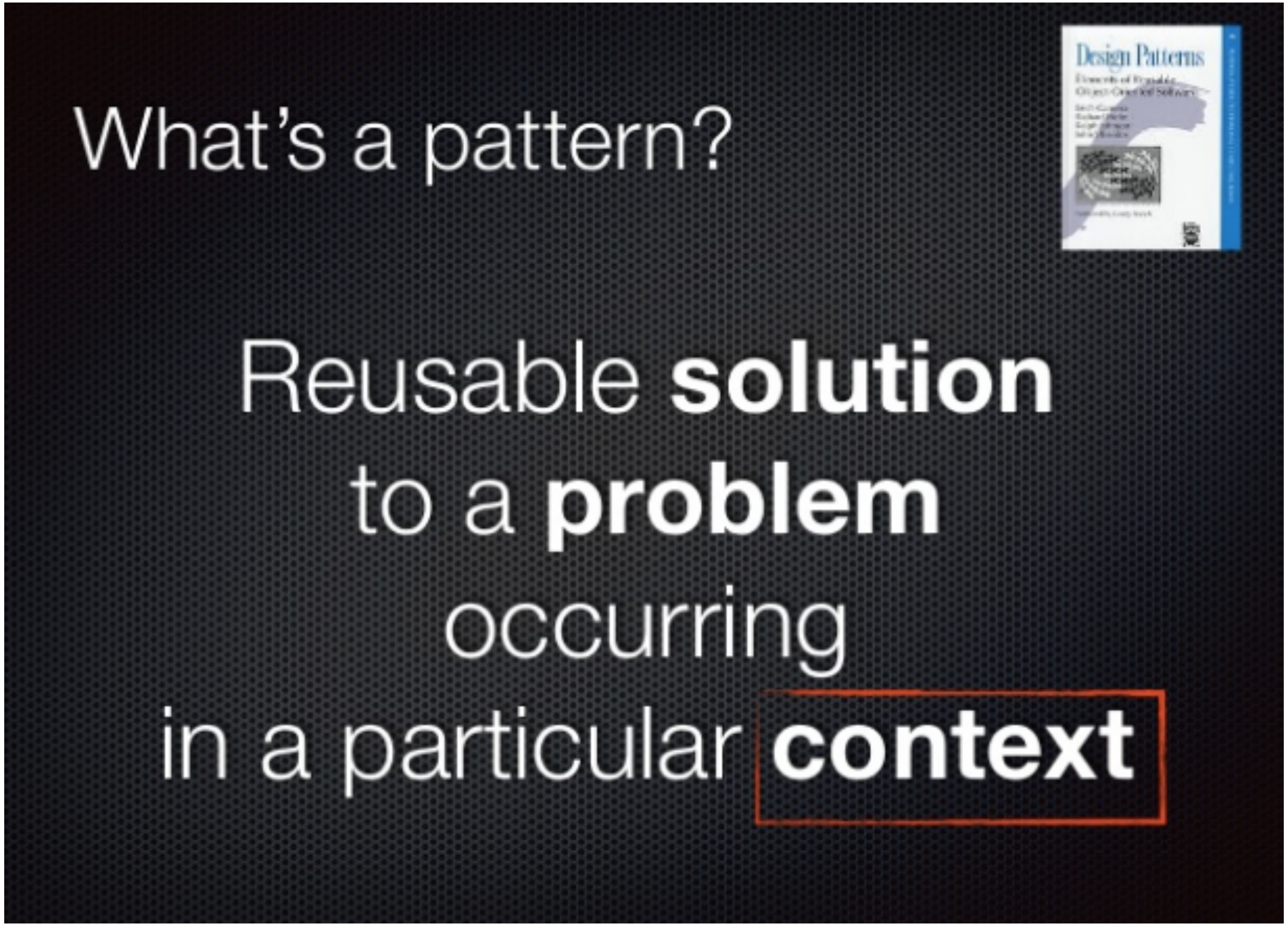


Service Name, Programming language(s), Programming paradigm(s), Architectural choices, Integration pattern(s), Transport protocols, Authentication, Authorization, Reporting, ETLs, Databases, Caching, Platform libraries, Service dependencies, CI Pipeline dependencies, 3rd party library dependencies, 3rd 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

Concerns for a single service, Sean Treadway, SoundCloud

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

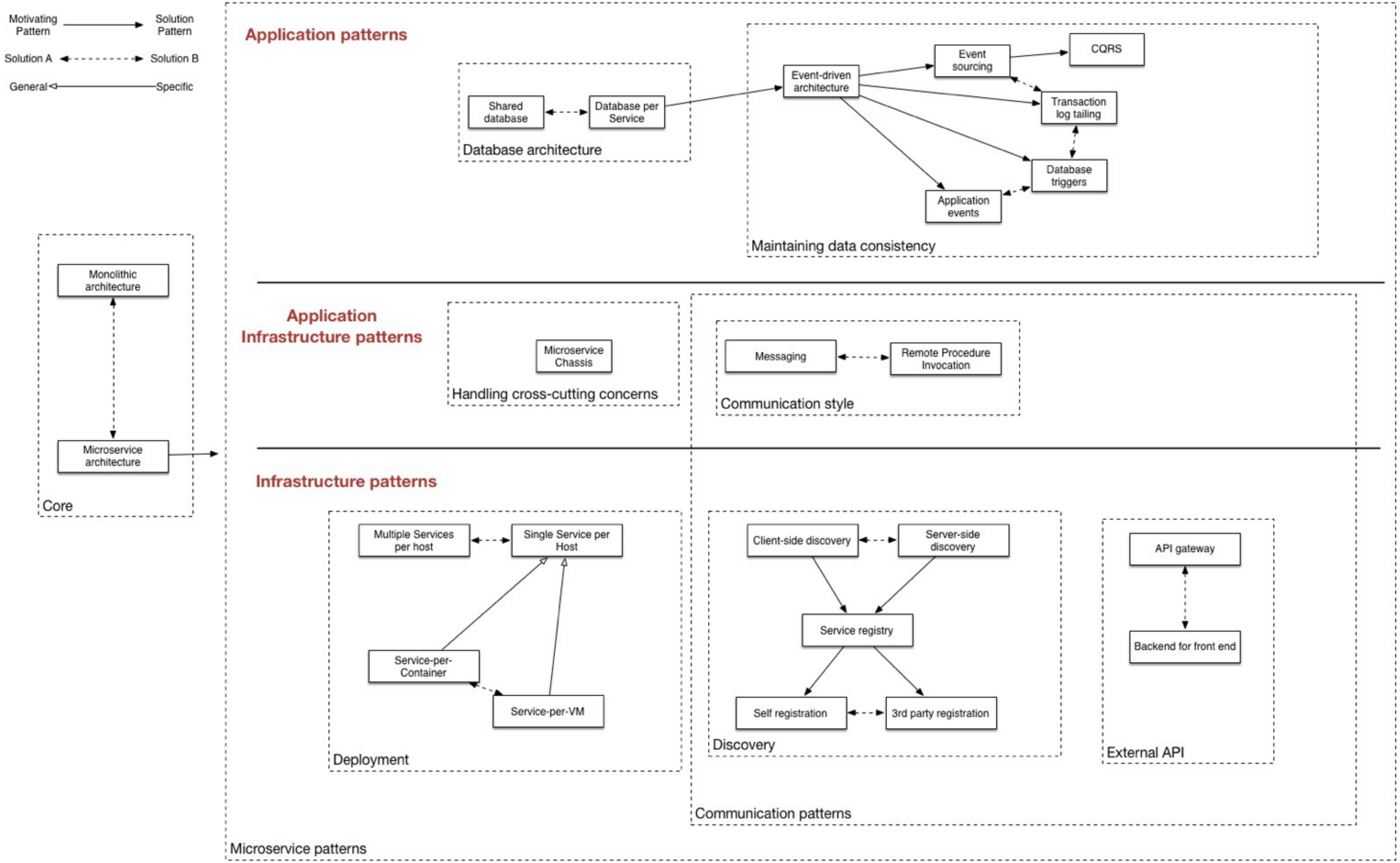


Design Patterns
Elements of Reusable Object-Oriented Software

What's a pattern?

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

slideshare.net/chris.e.richardson/microservices-pattern-language-microxchg-microxchg2016



Motivating Pattern → Solution Pattern

Solution A ↔ Solution B

General ◁— Specific

Core

Monolithic architecture

Microservice architecture

Microservice patterns

Application patterns

Database architecture

Shared database

Database per Service

Maintaining data consistency

Event-driven architecture

Event sourcing

CQRS

Transaction log tailing

Database triggers

Application events

Application Infrastructure patterns

Handling cross-cutting concerns

Microservice Chassis

Communication patterns

Communication style

Messaging

Remote Procedure Invocation

Infrastructure patterns

Deployment

Multiple Services per host

Single Service per Host

Service-per-Container

Service-per-VM

Discovery

Client-side discovery

Server-side discovery

Service registry

Self registration

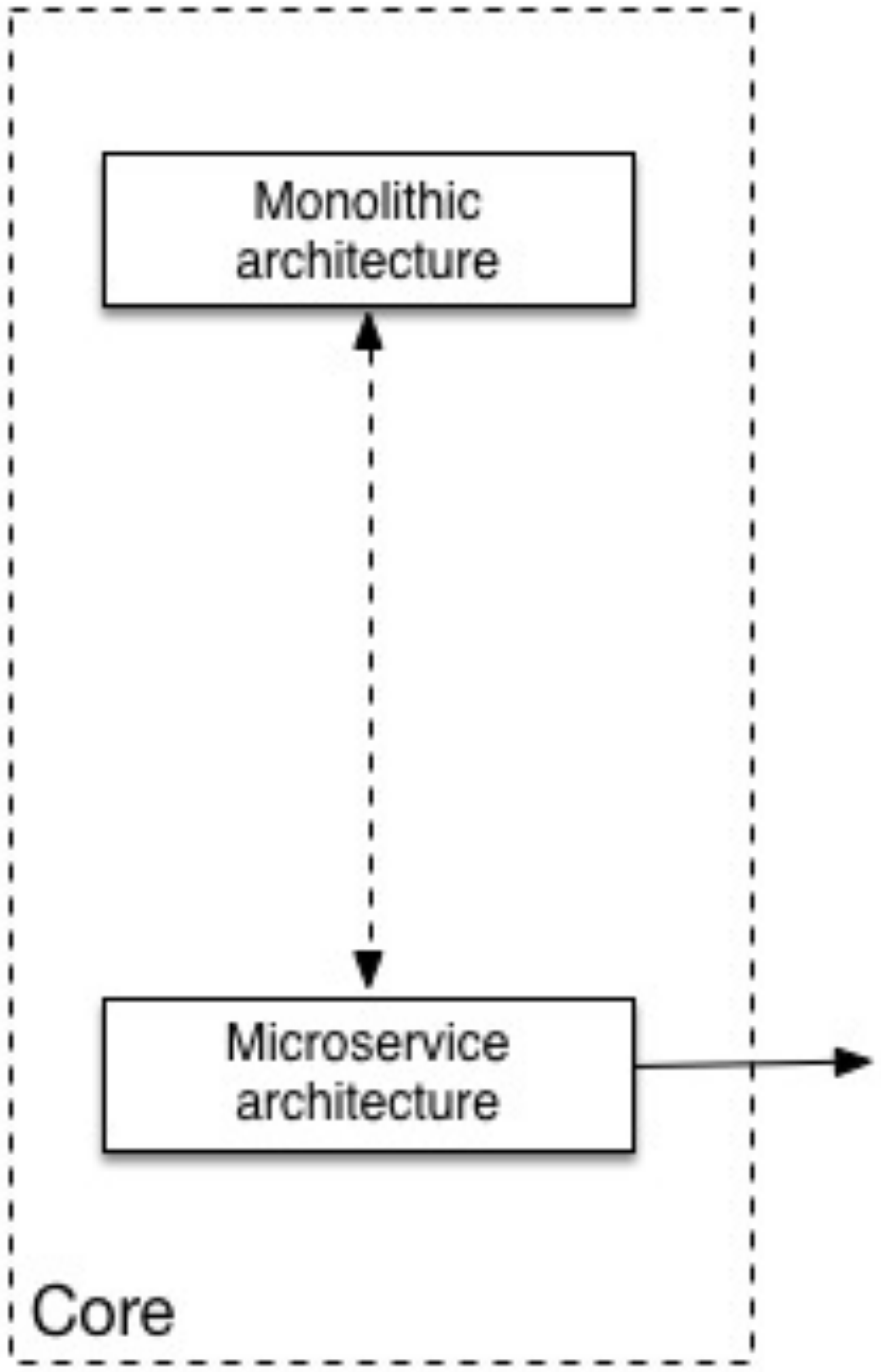
3rd party registration

External API

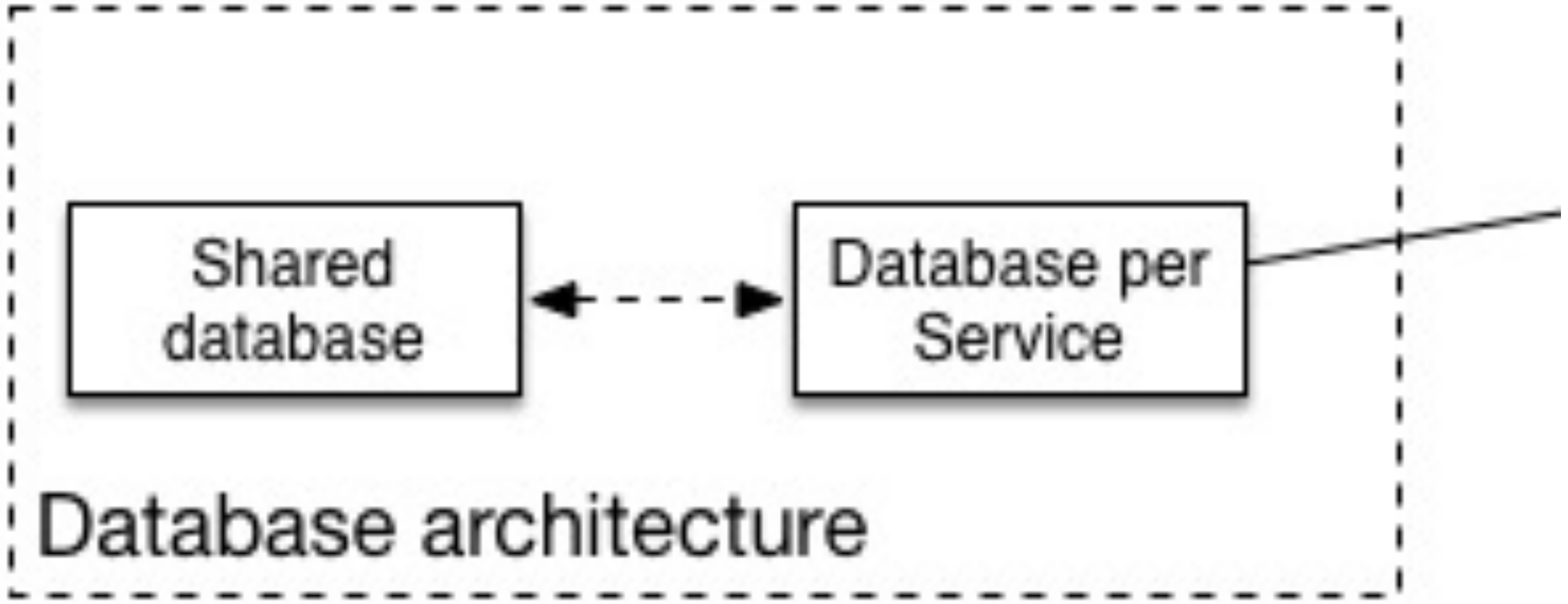
API gateway

Backend for front end

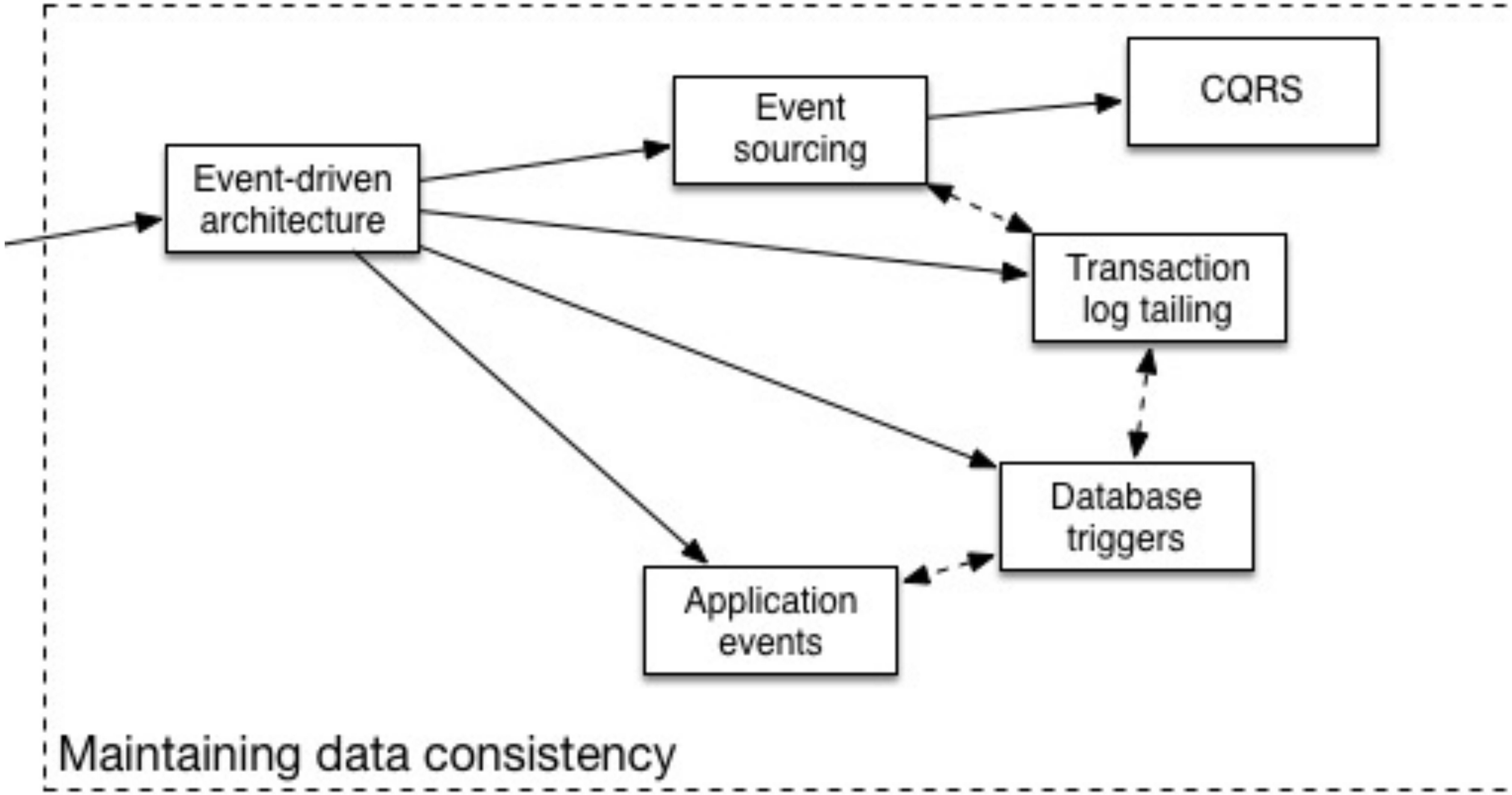
microservices.io/patterns/index.html



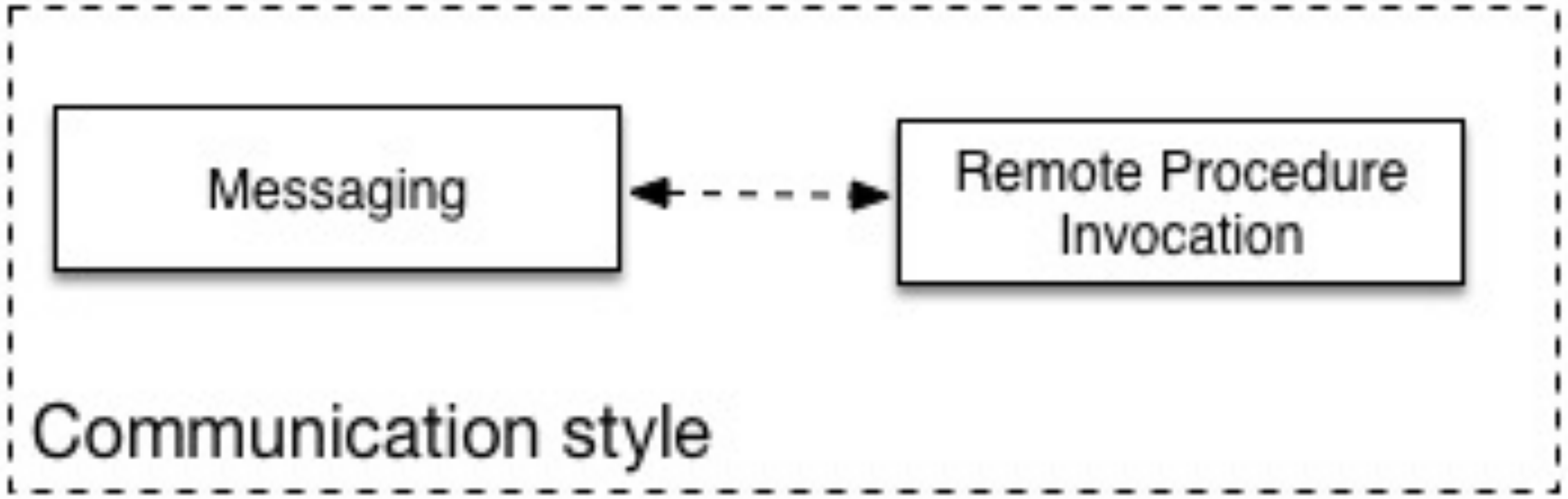
microservices.io/patterns/index.html



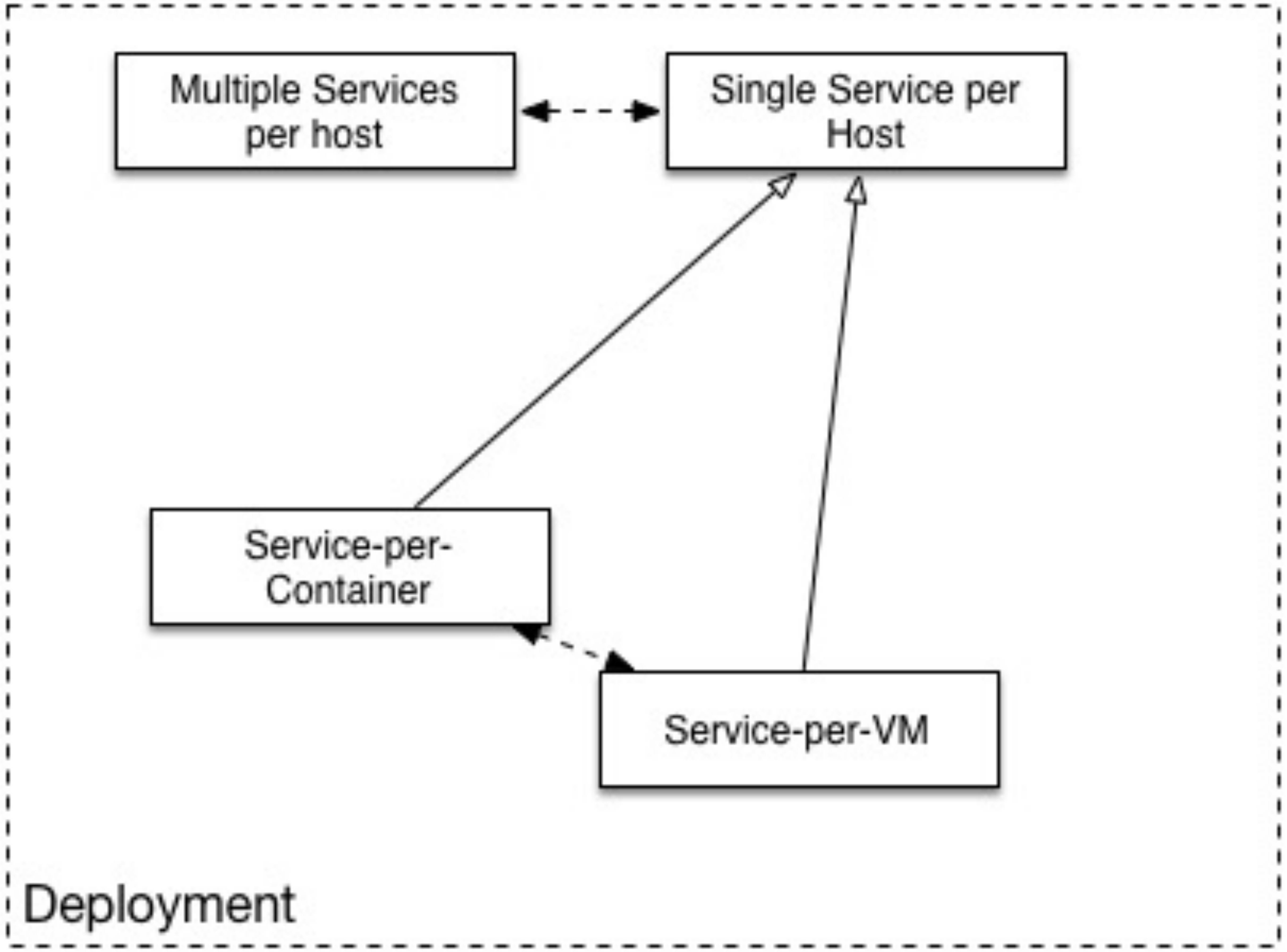
microservices.io/patterns/index.html



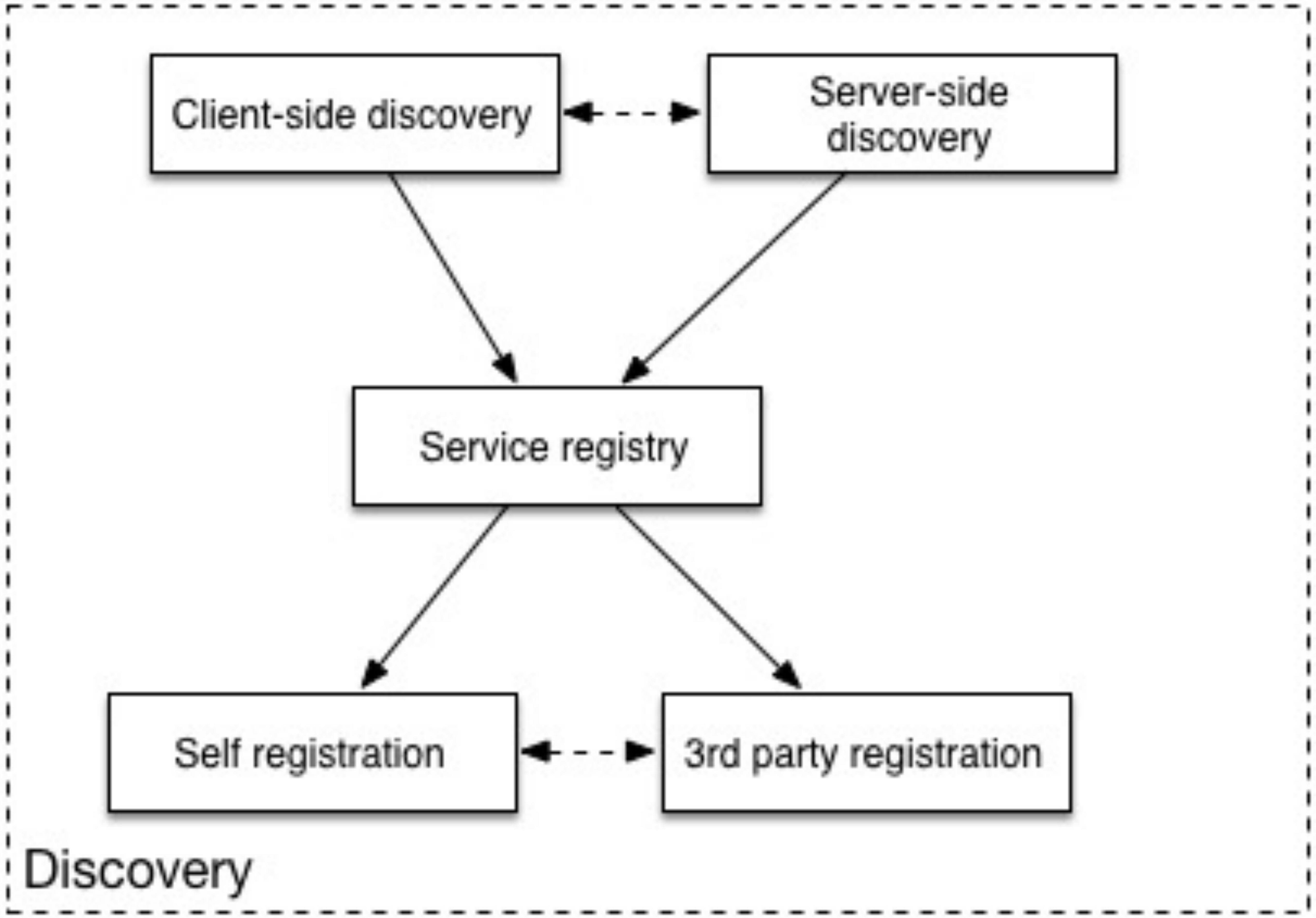
microservices.io/patterns/index.html



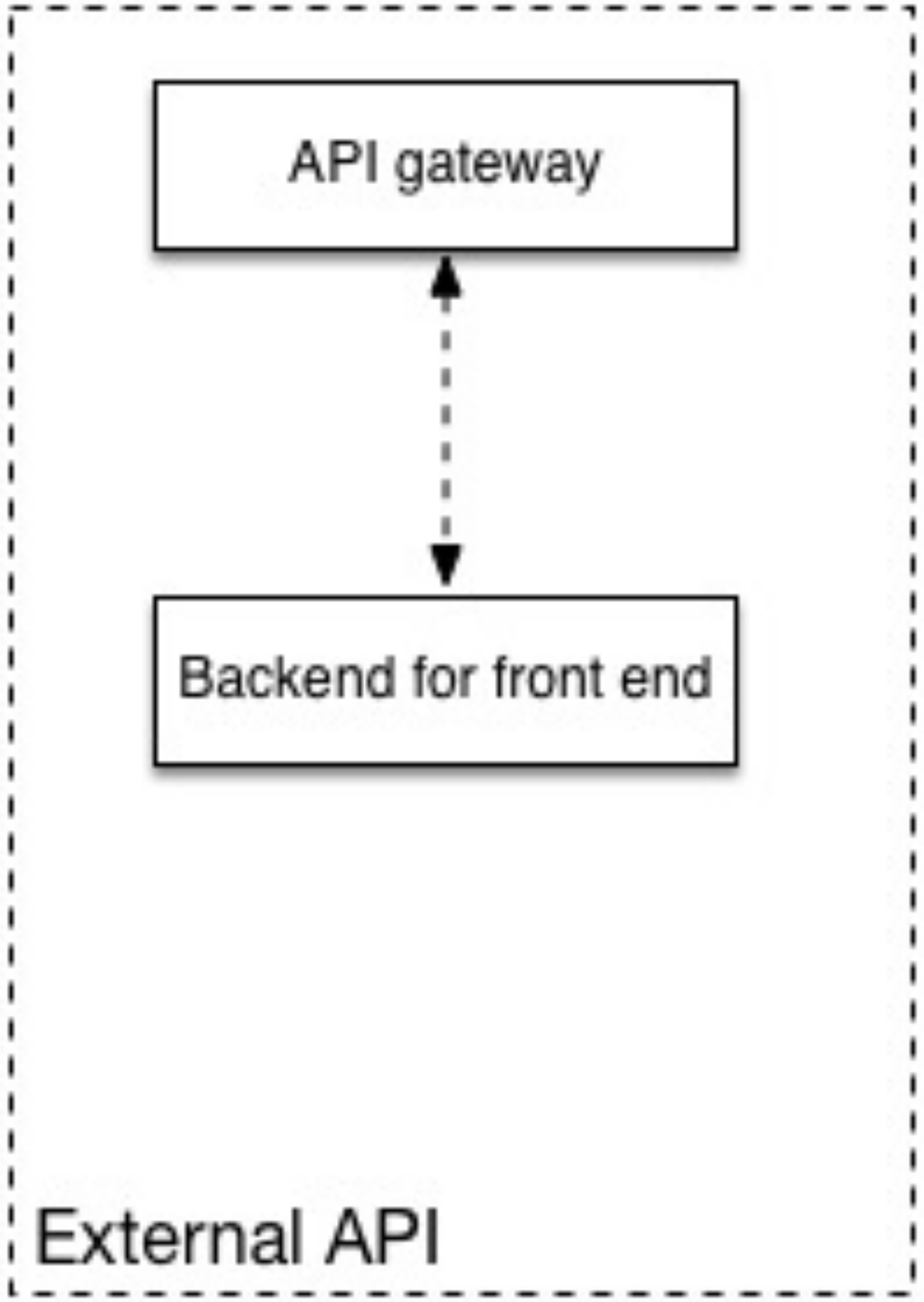
microservices.io/patterns/index.html



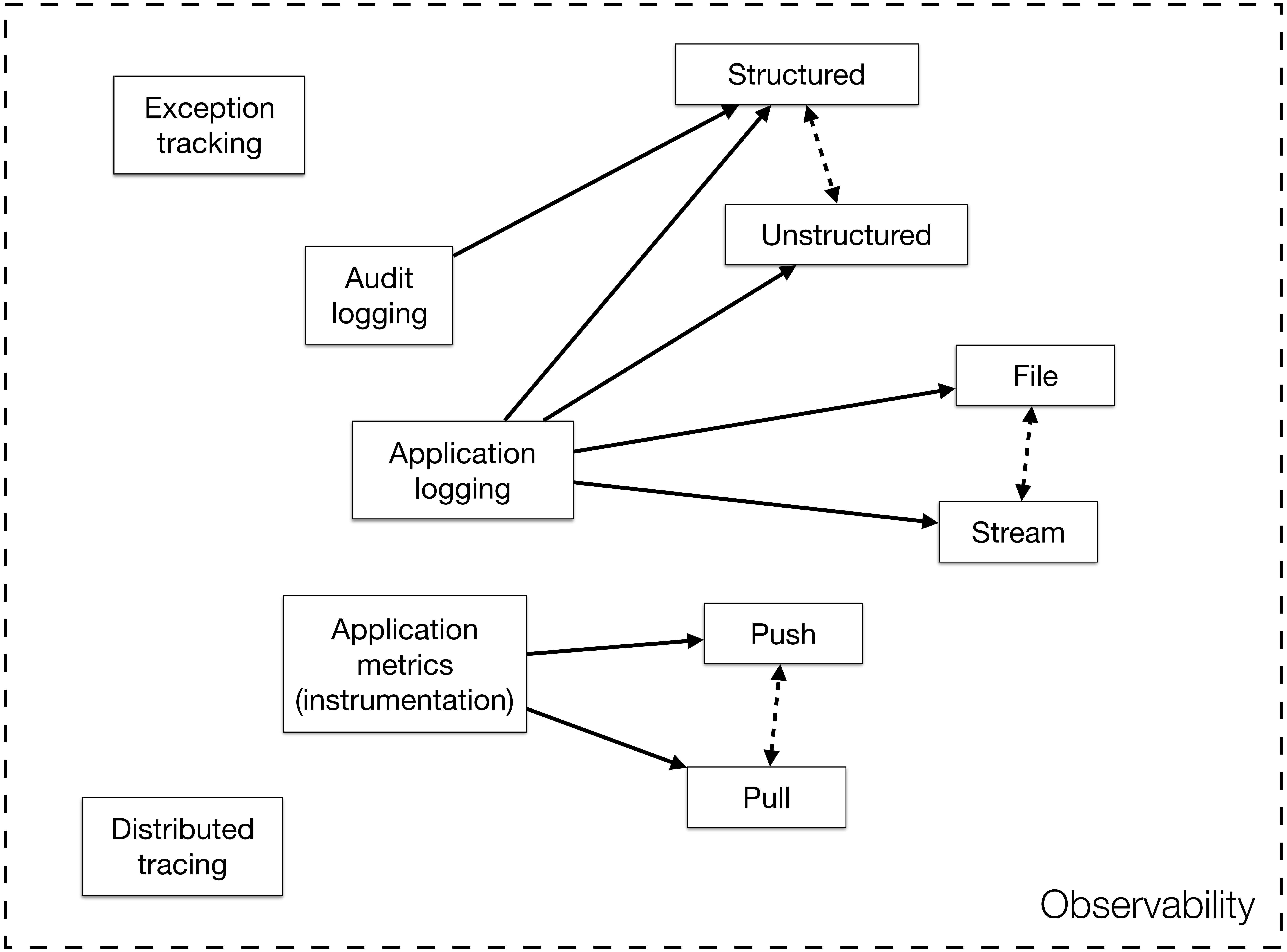
microservices.io/patterns/index.html



microservices.io/patterns/index.html



microservices.io/patterns/index.html


Structured
Exception tracking
Unstructured
Audit logging
File
Application logging
Stream
Application metrics (instrumentation)
Push
Pull
Distributed tracing
Observability

What is 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

Service registration

Business logic

Service discovery
Load balancing

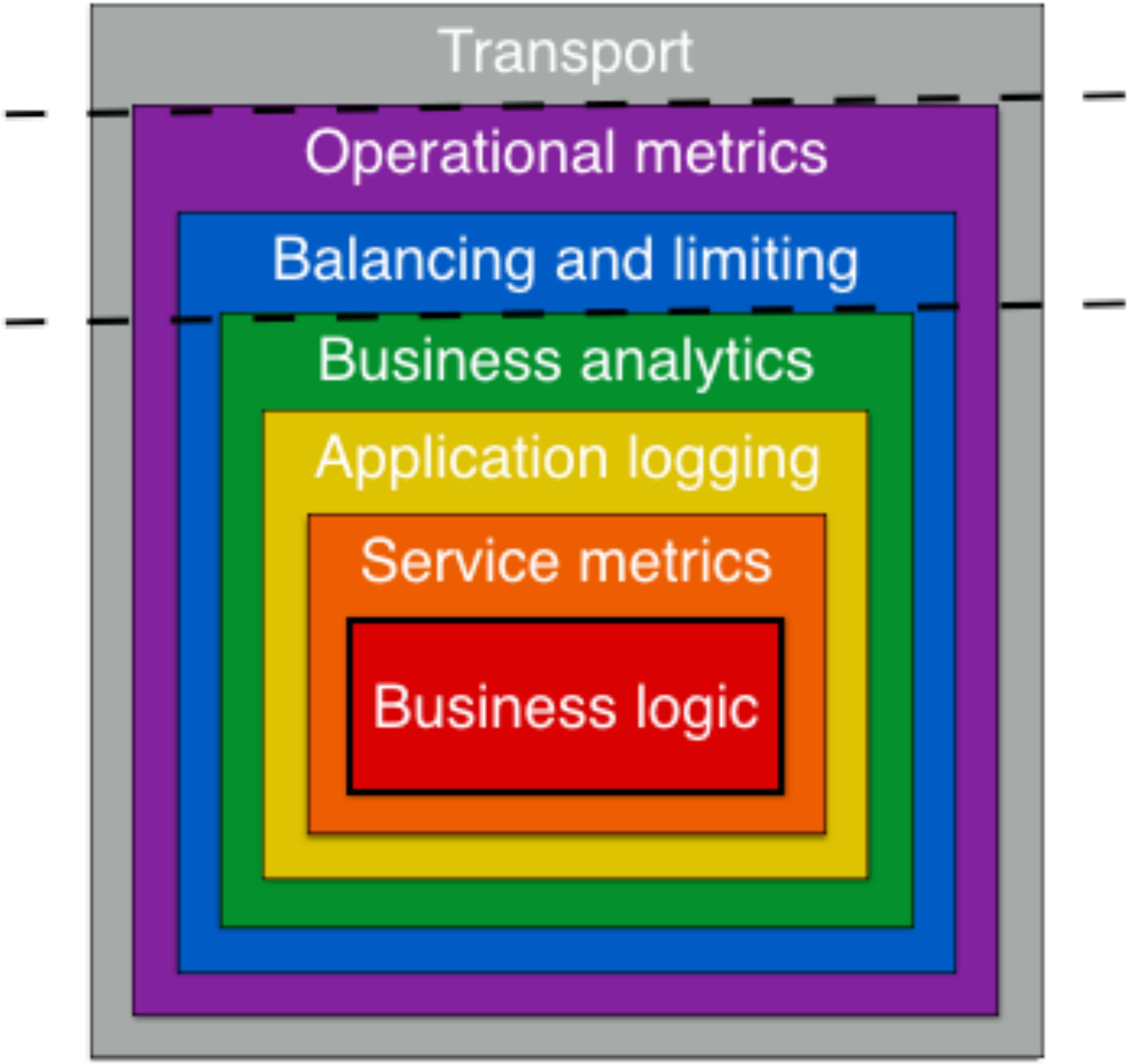
Metrics
Logging
Distributed tracing

- Transport
- Rate limiting
- Circuit breaking

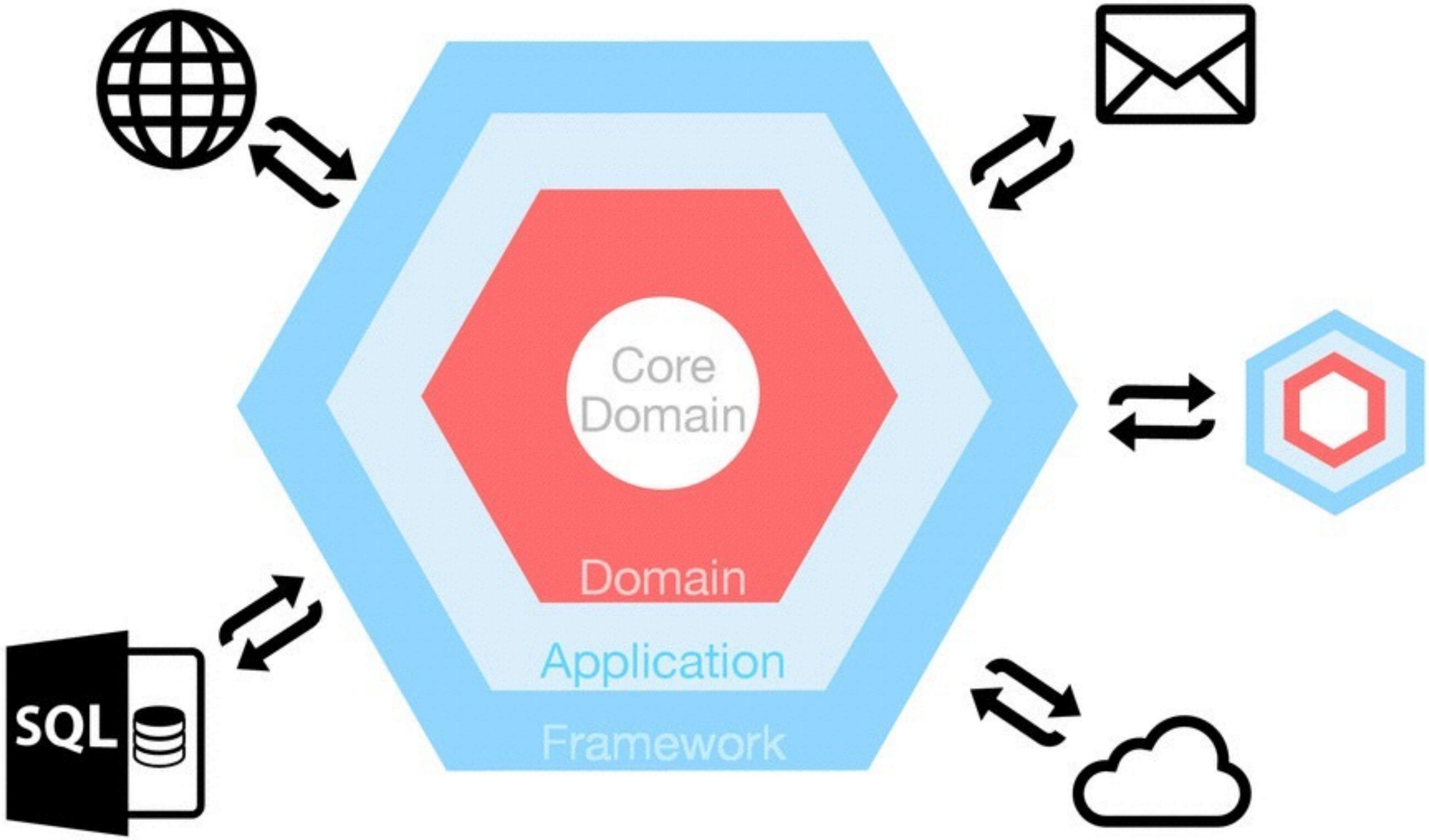
Business logic

- Service registration
- Service discovery
- Load balancing

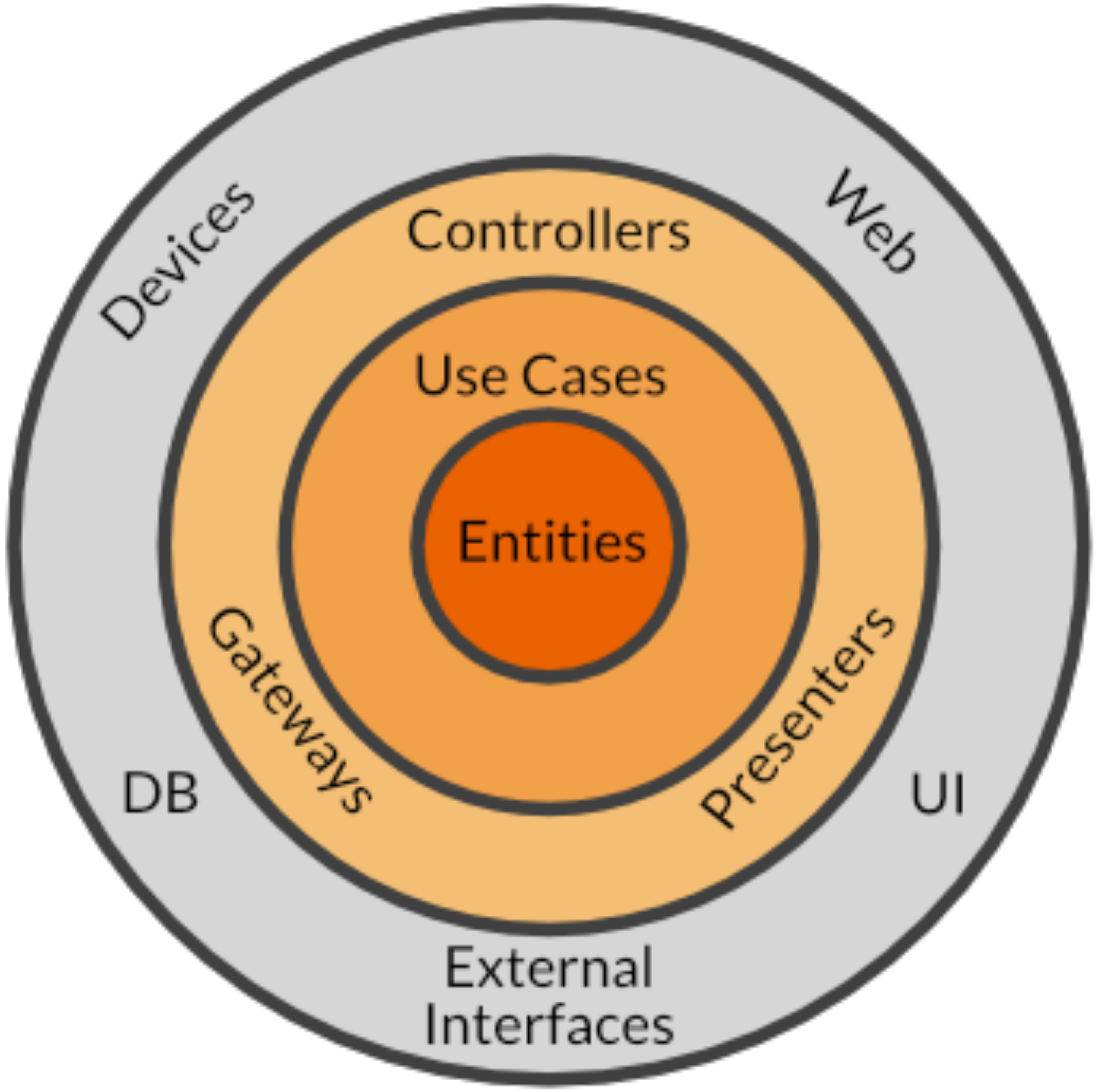
- Metrics
- Logging
- Distributed tracing


Transport
Operational metrics
Balancing and limiting
Business analytics
Application logging
Service metrics
Business logic

The Hexagon



http://fideloper.com/hexagonal-architecture



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

Source code dependencies can only point inwards.

https://appliedgo.net/di/

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

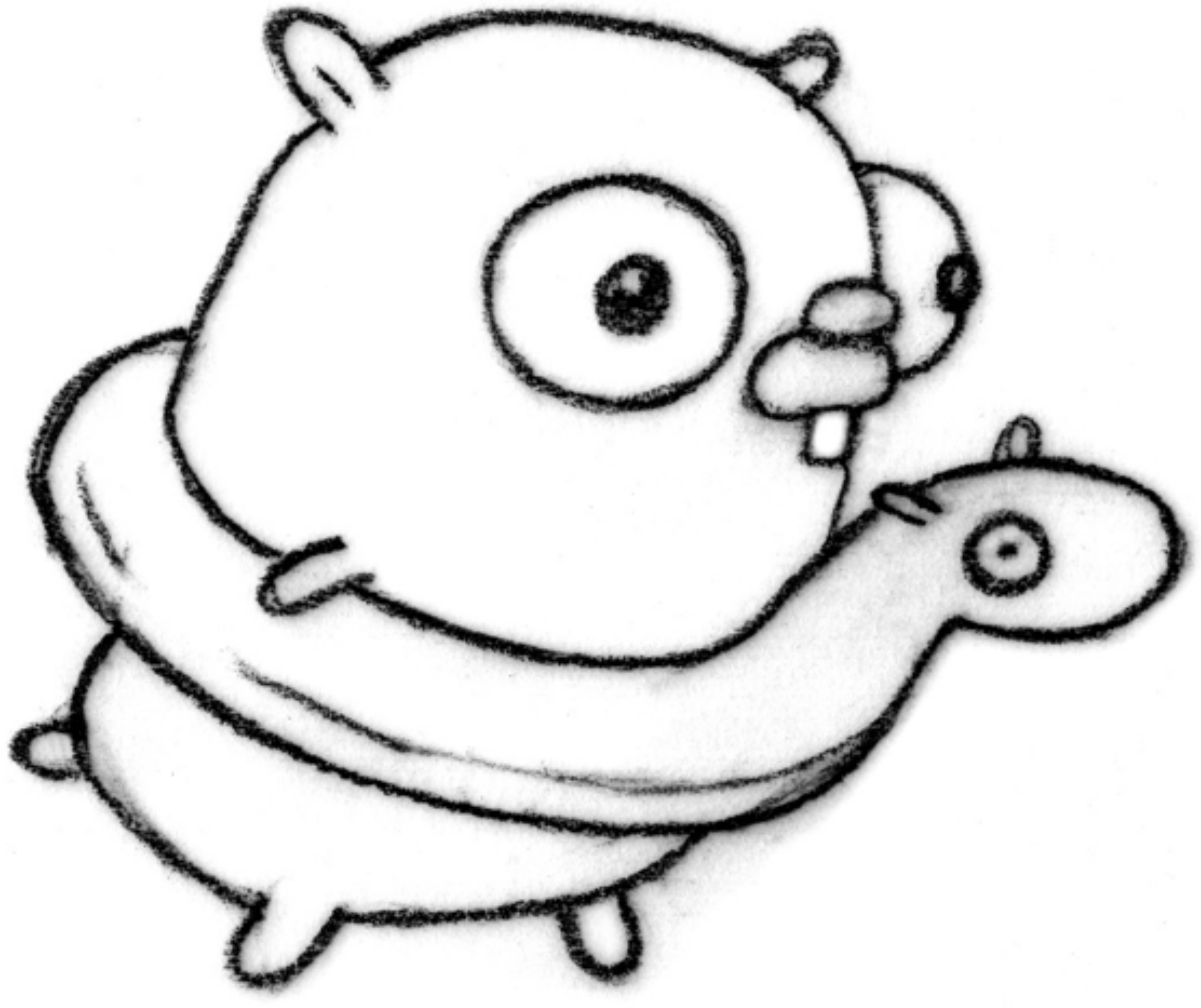
- Toward a software engineering
  - SOLID Design
  - Domain Driven Design
  - The Clean Architecture
  - Hexagonal Architecture

Go refresher

Via go tool present

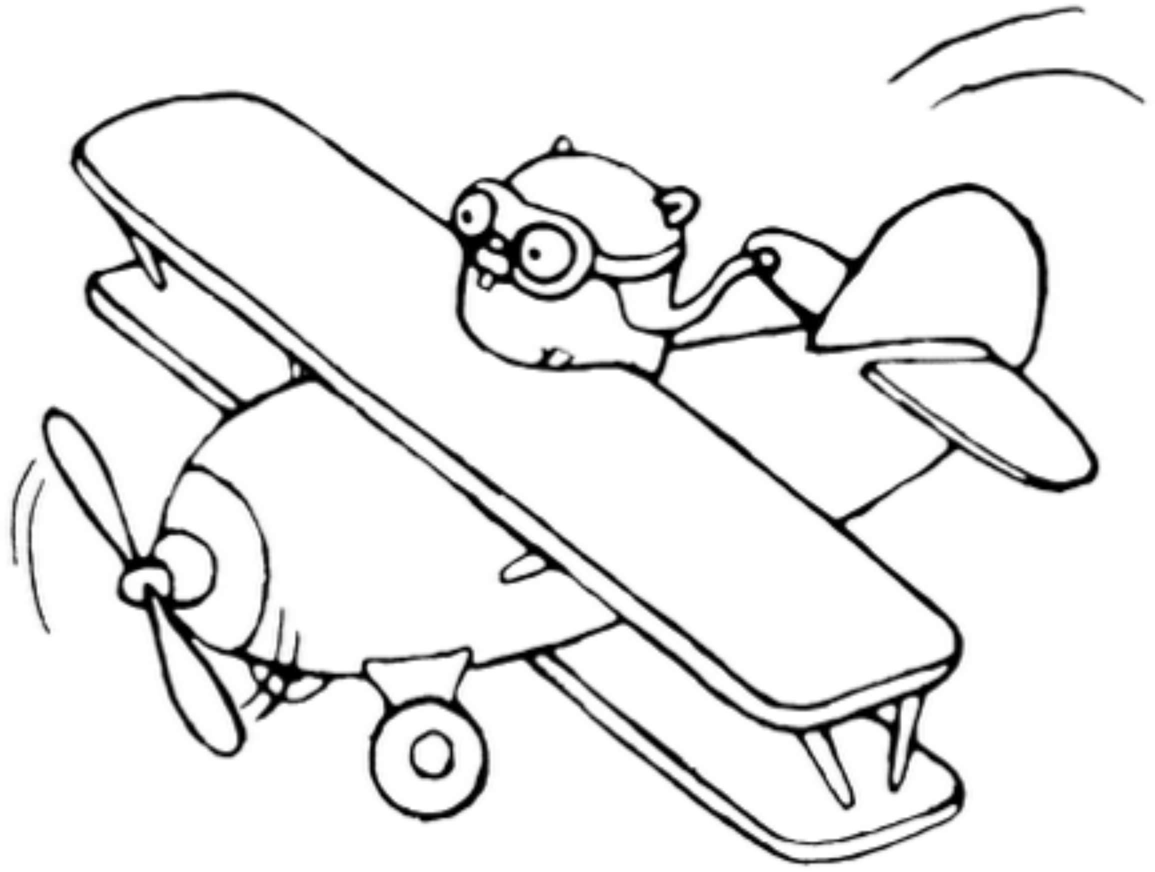
addsvc

Basic implementation



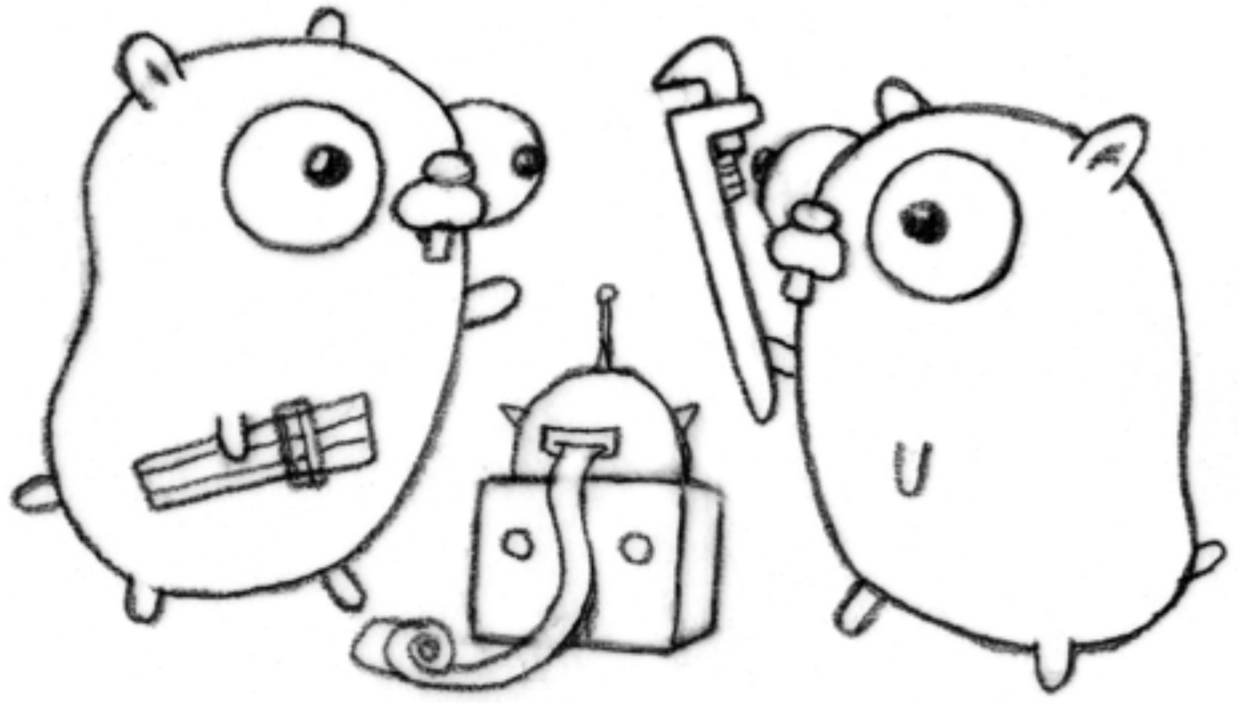
git apply

01, 02, 03



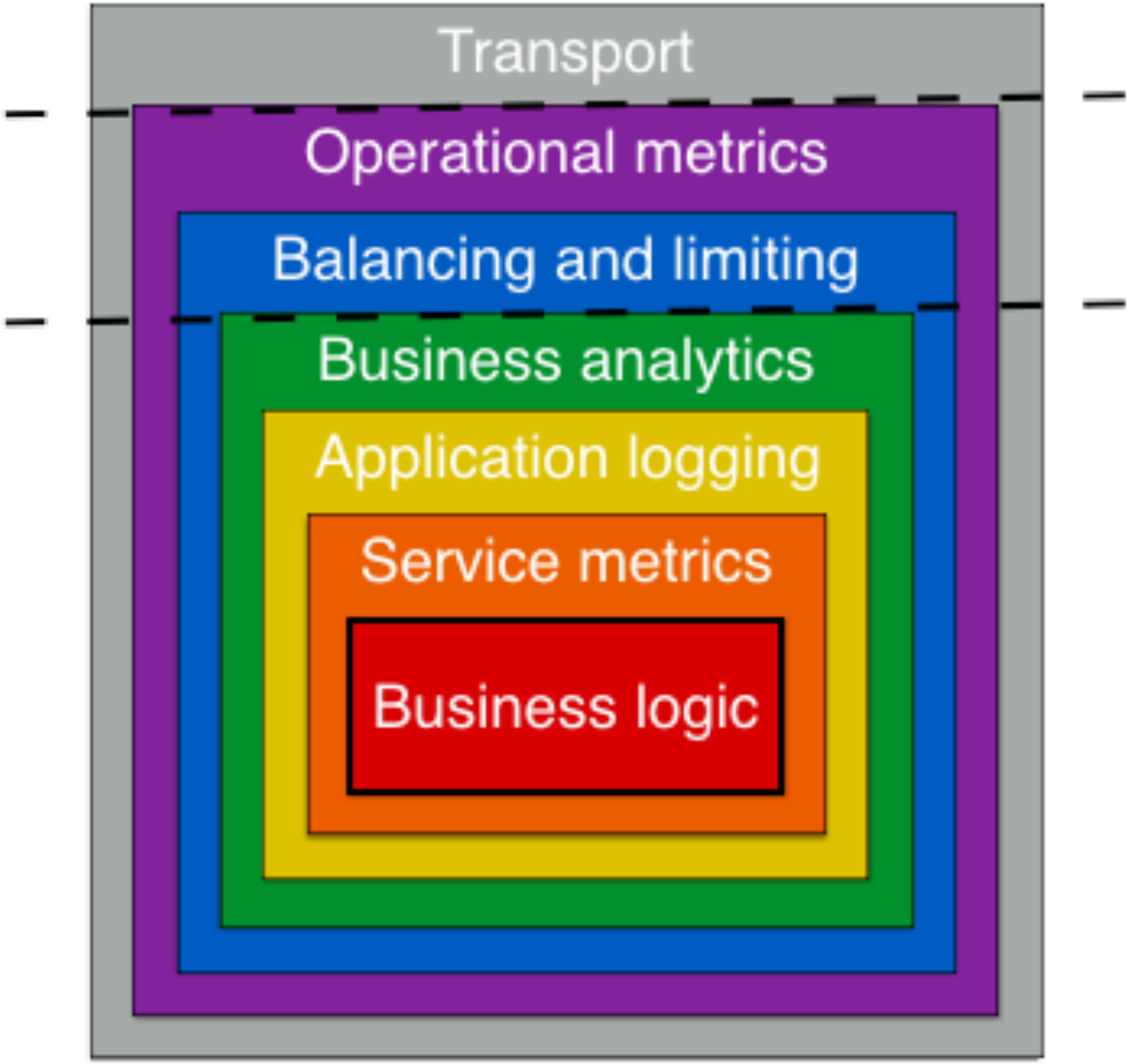
Challenge

Add a method Mult(a, b float64) (float64, error)



addsvc

Structure: endpoints, middlewares, transports



Transport

Operational metrics

Balancing and limiting

Business analytics

Application logging

Service metrics

Business logic

Middleware

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

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

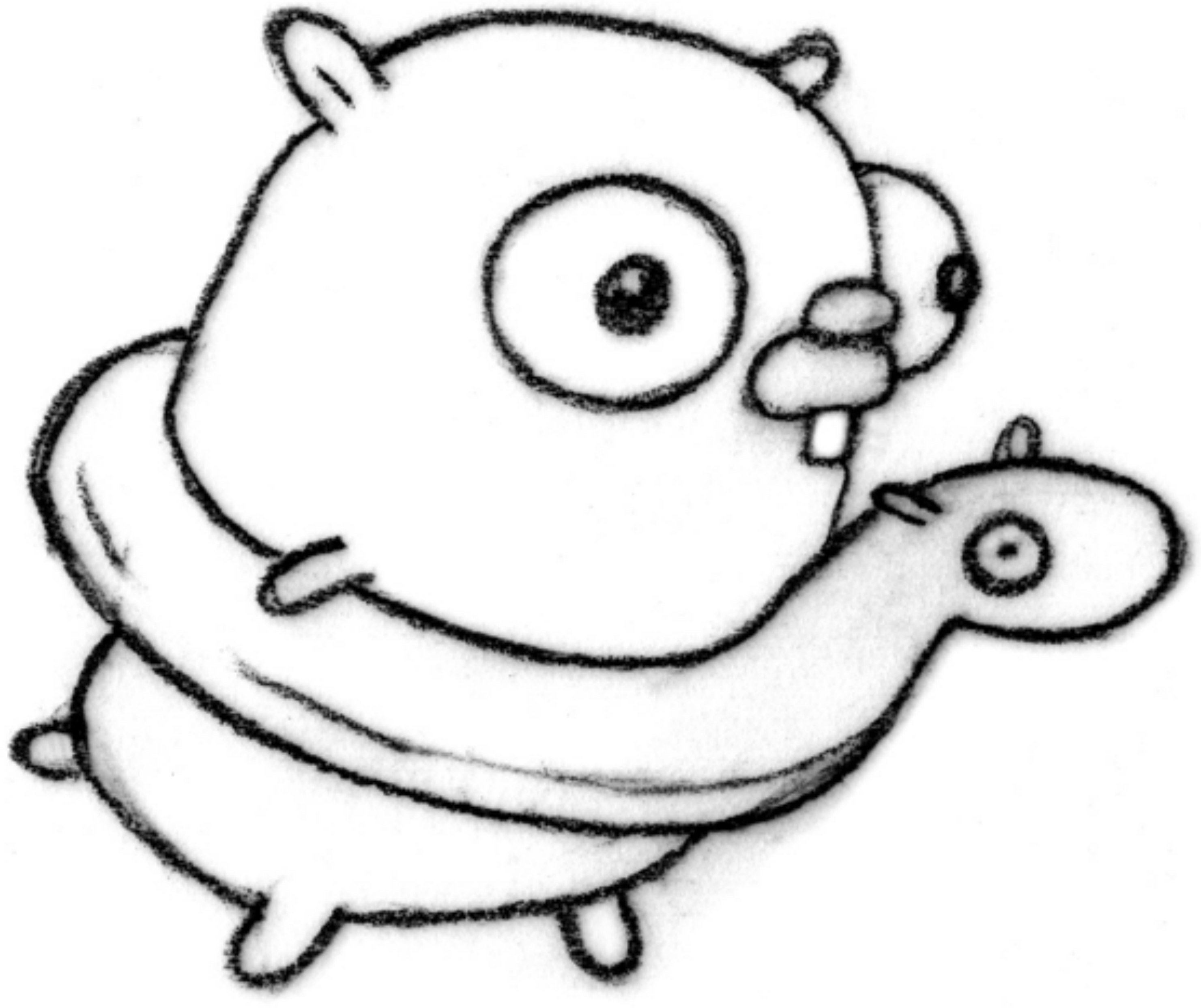
```
func instrument(...) {
    // proceed as normal
    m.With(method, code).Observe(t)
}
```

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

Endpoint

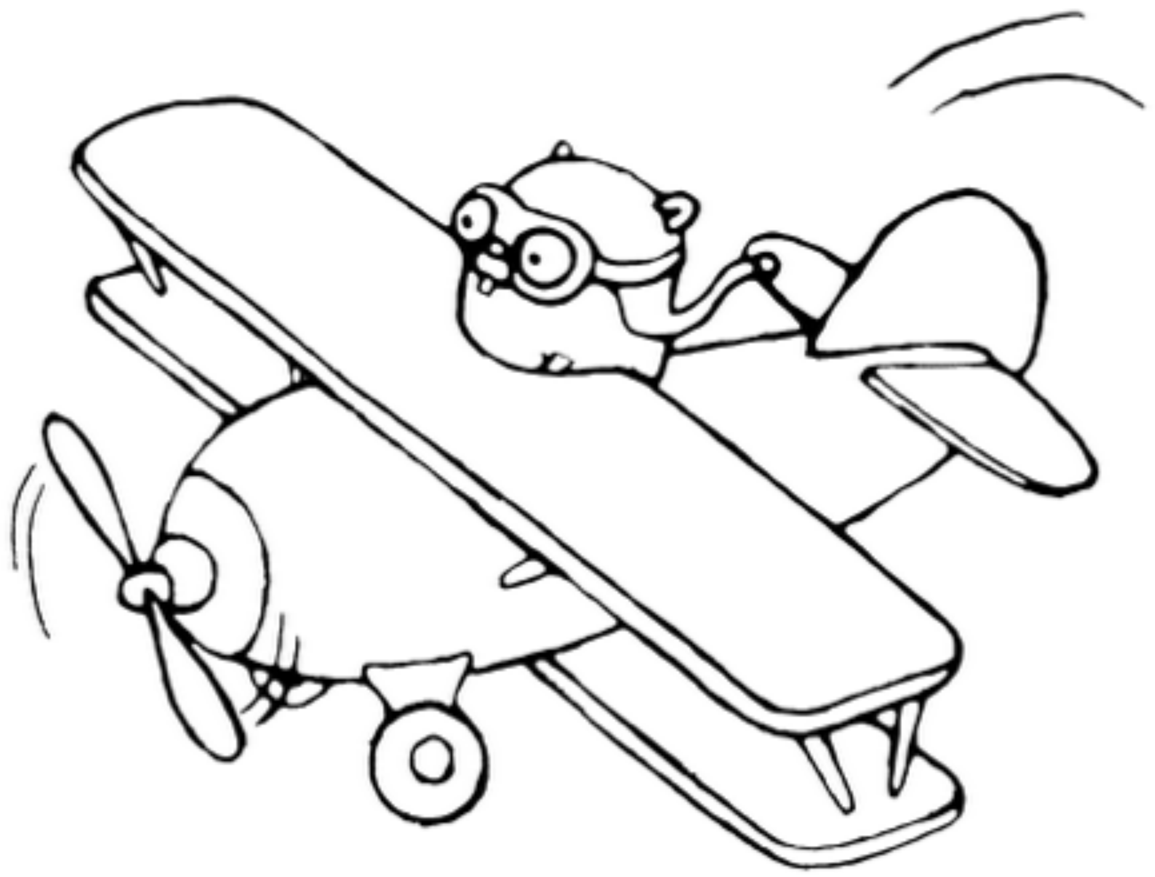
- Generalize each operation as RPC: request, response
  - `type Endpoint func(request) response`
- Accommodate failure and request-scoped information

  -
    ```
    type Endpoint func(ctx context.Context, request interface{})
        (response interface{}, err error)
    ```
- Empty interface? Empty interface :(



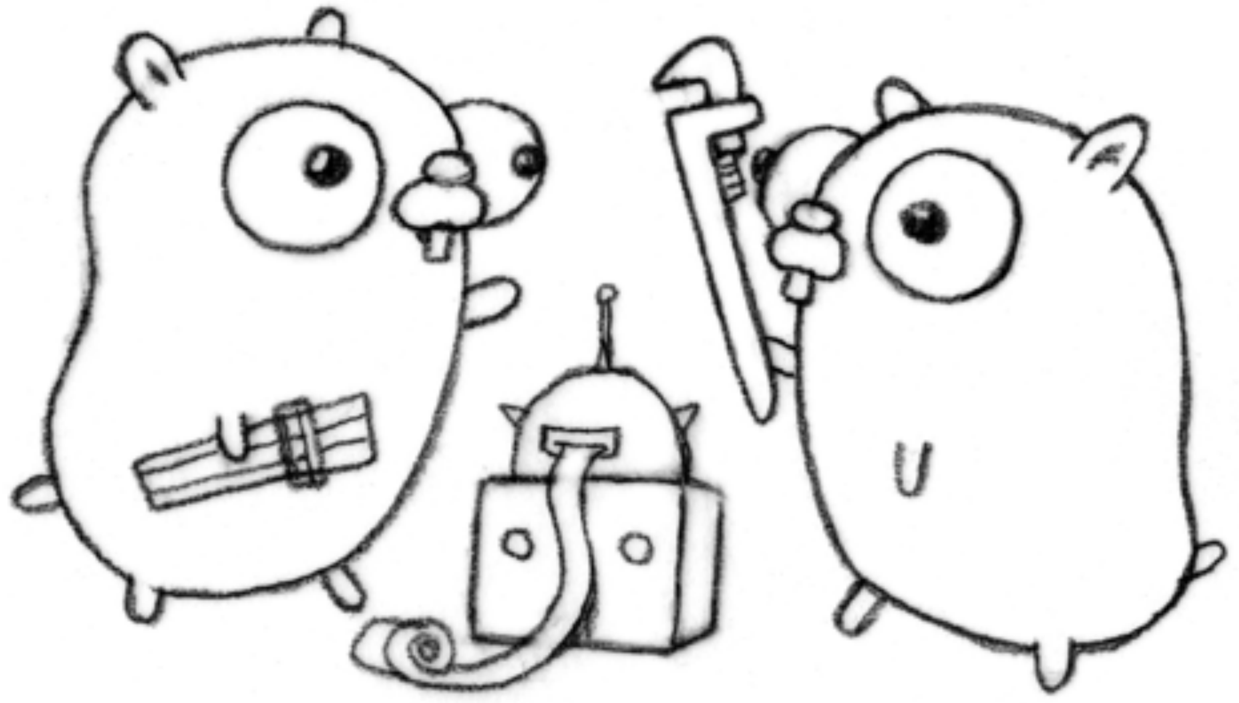
git apply

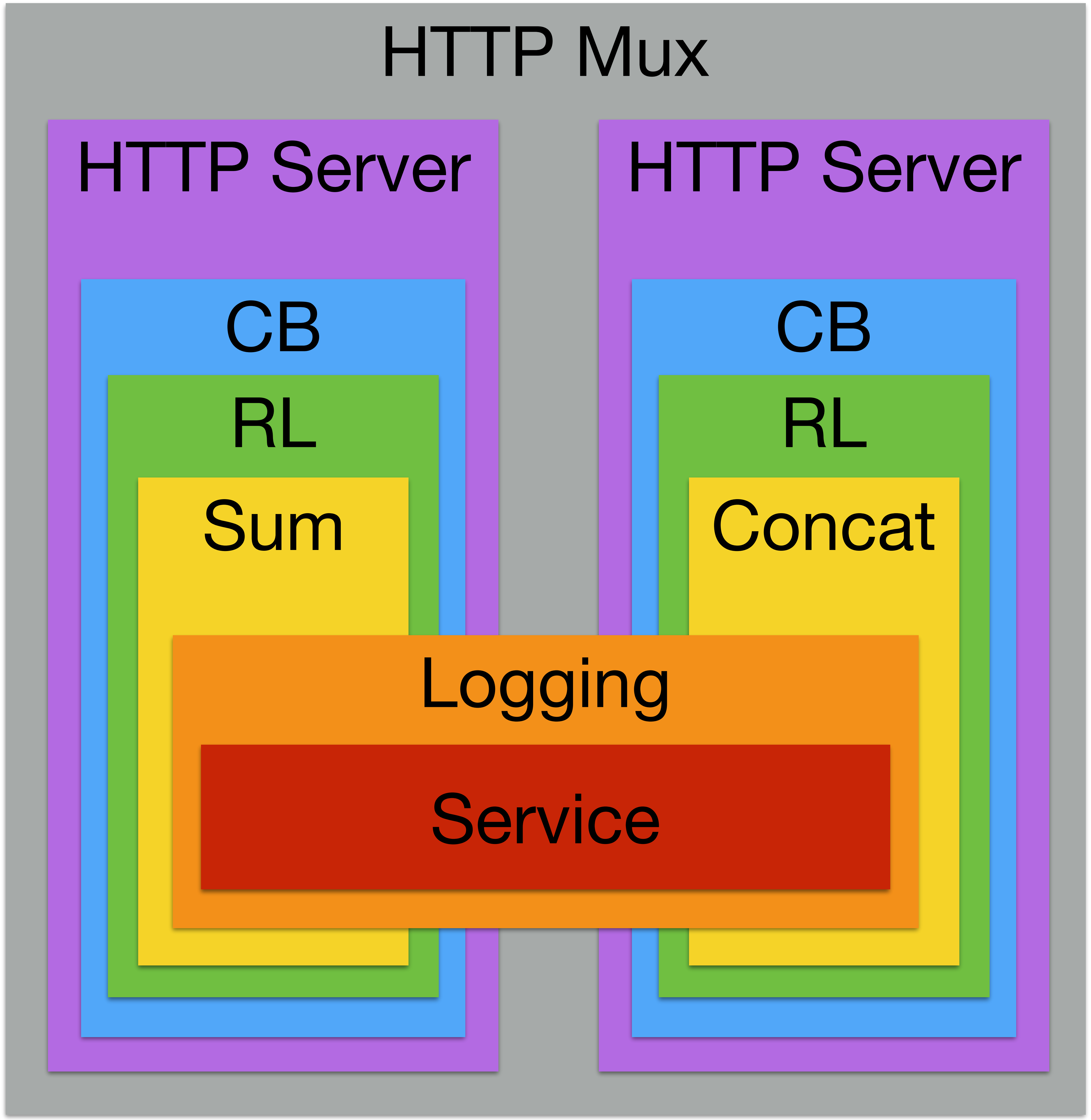
04, 05

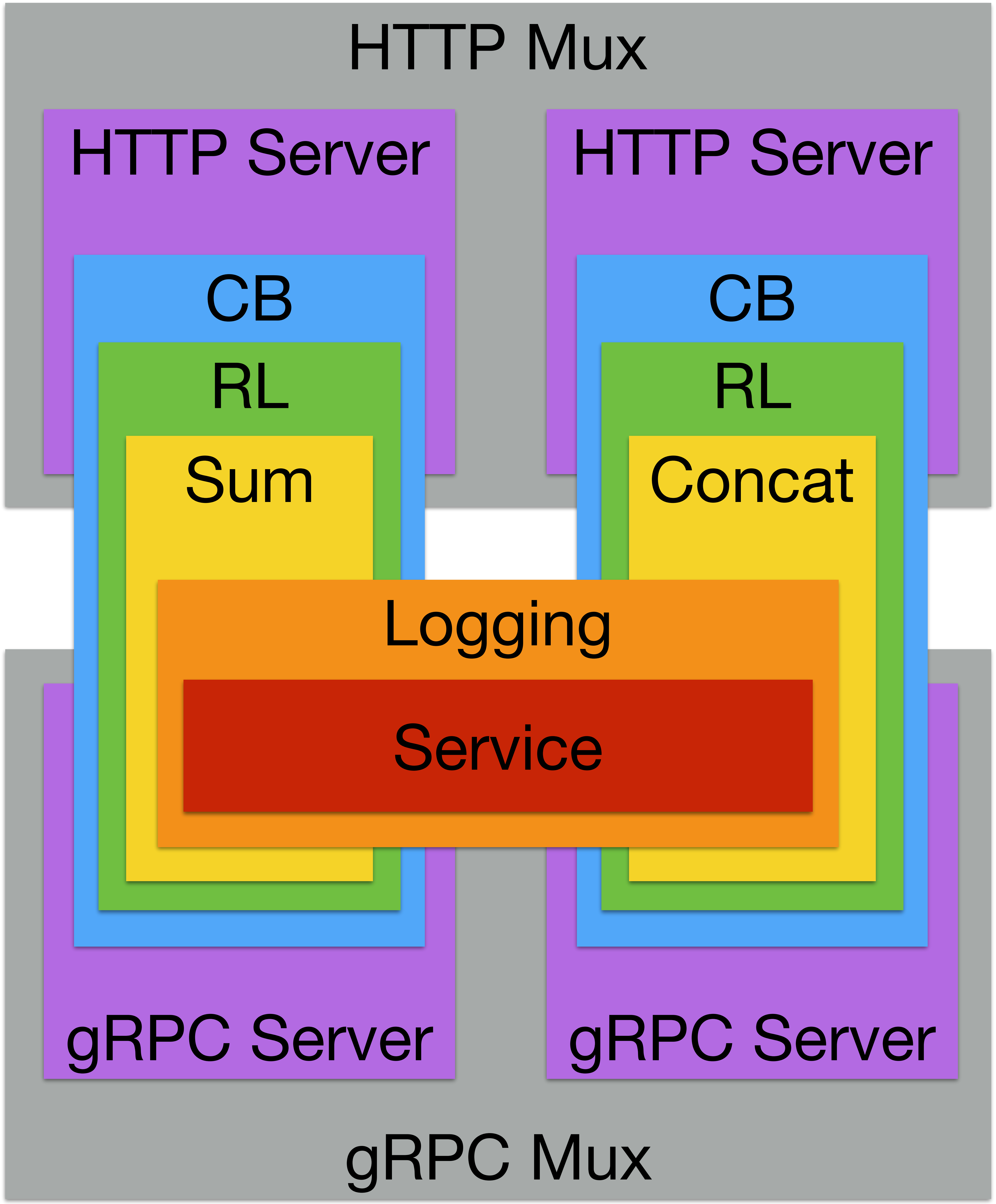


Challenge

A Service middleware that uppercases the result of Concat




HTTP Mux
HTTP Server
HTTP Server
CB
CB
RL
RL
Sum
Concat
Logging
Service



HTTP Mux

HTTP Server

HTTP Server

CB

CB

RL

RL

Sum

Concat

Logging

Service

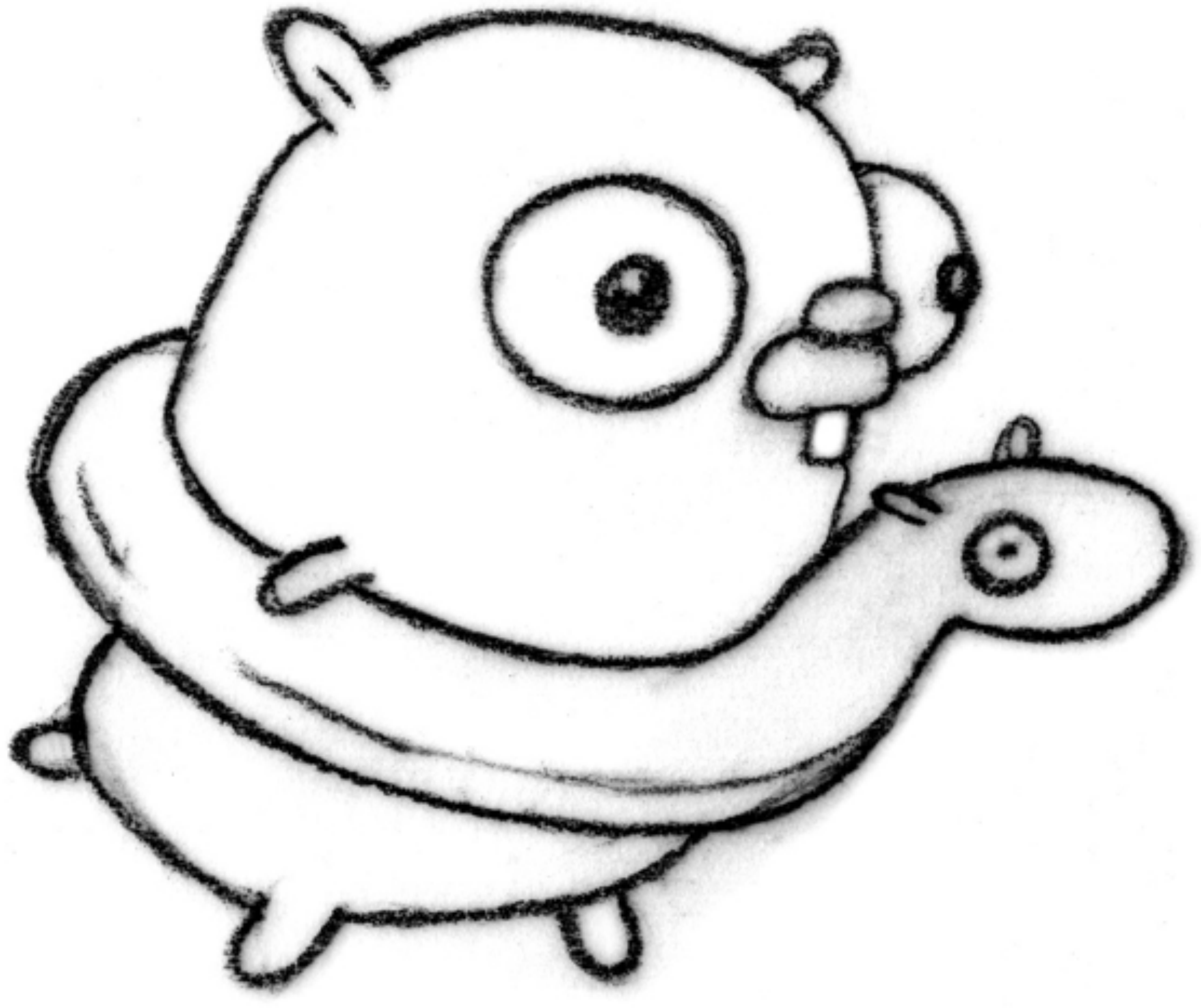
gRPC Server

gRPC Server

gRPC Mux

Repo organization

github.com/thockin/go-build-template

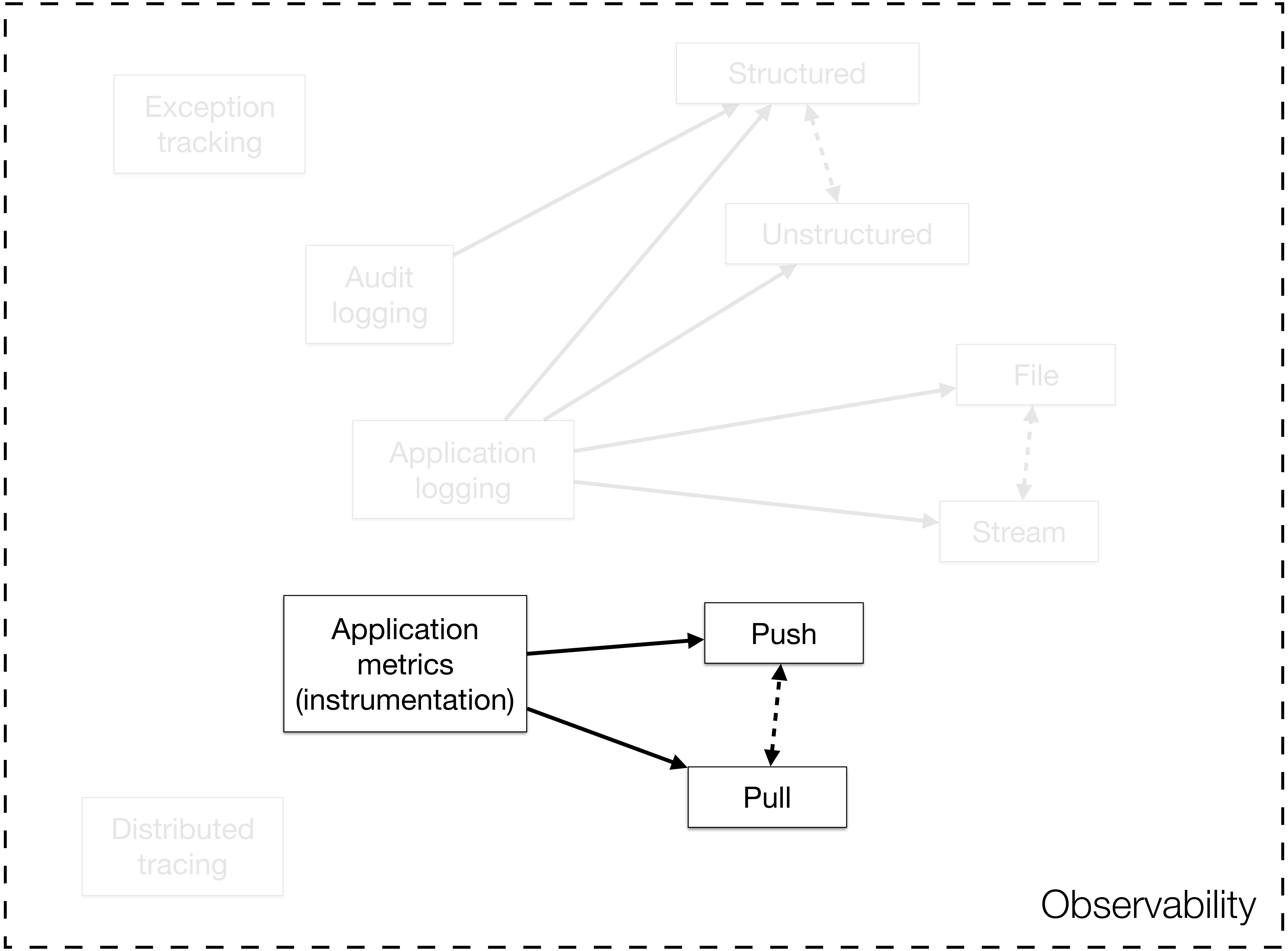


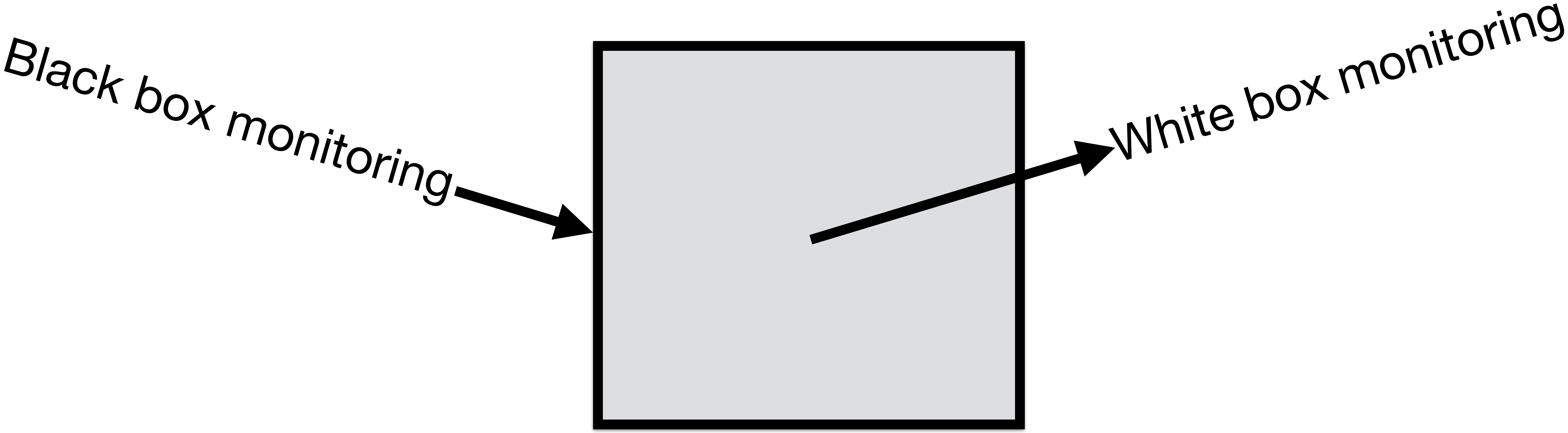
git apply

06

addsvc

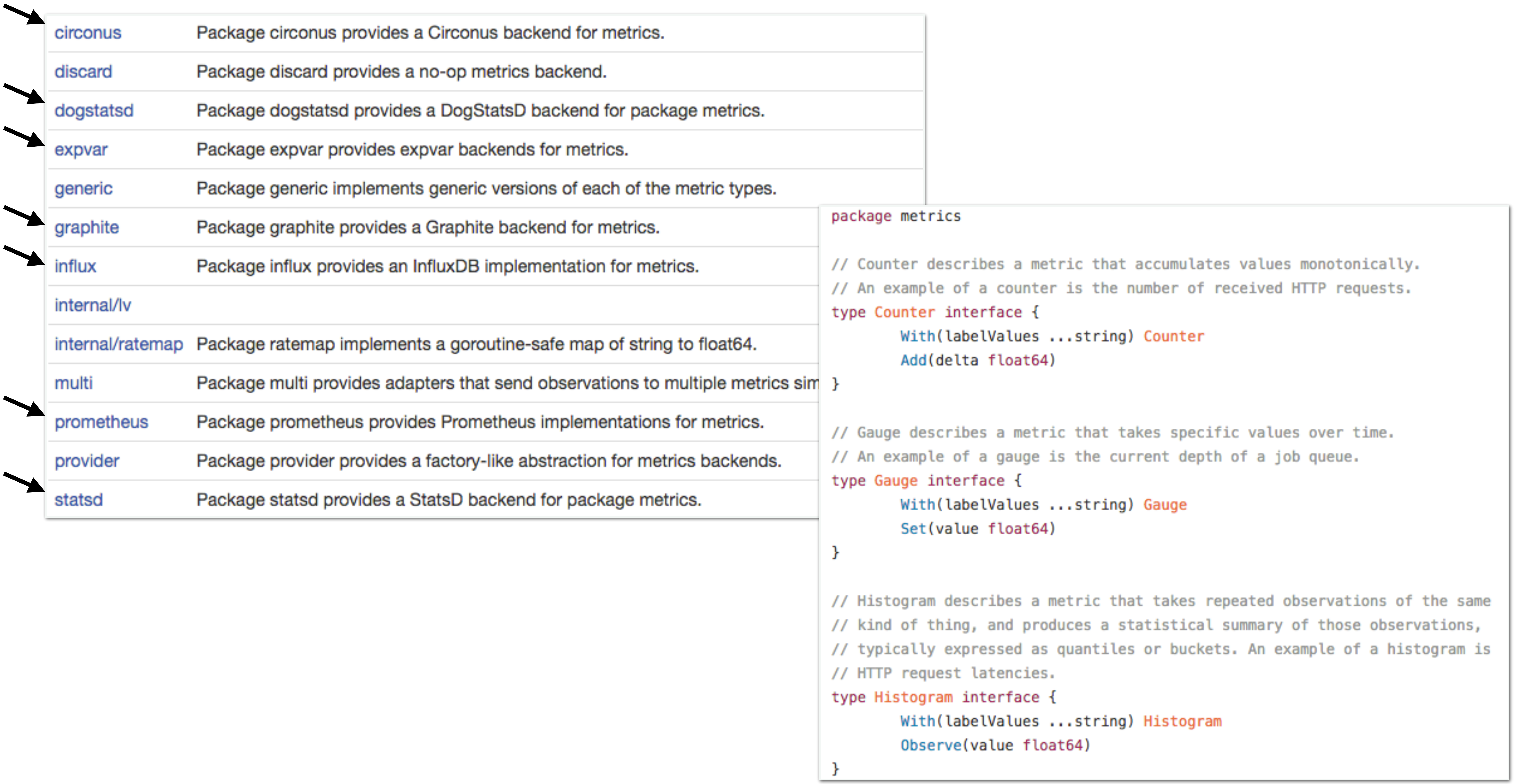
Instrumentation with Prometheus


Exception tracking
Structured
Unstructured
Audit logging
File
Application logging
Stream
Application metrics (instrumentation)
Push
Pull
Distributed tracing
Observability



Black box monitoring

White box monitoring



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 sim
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)

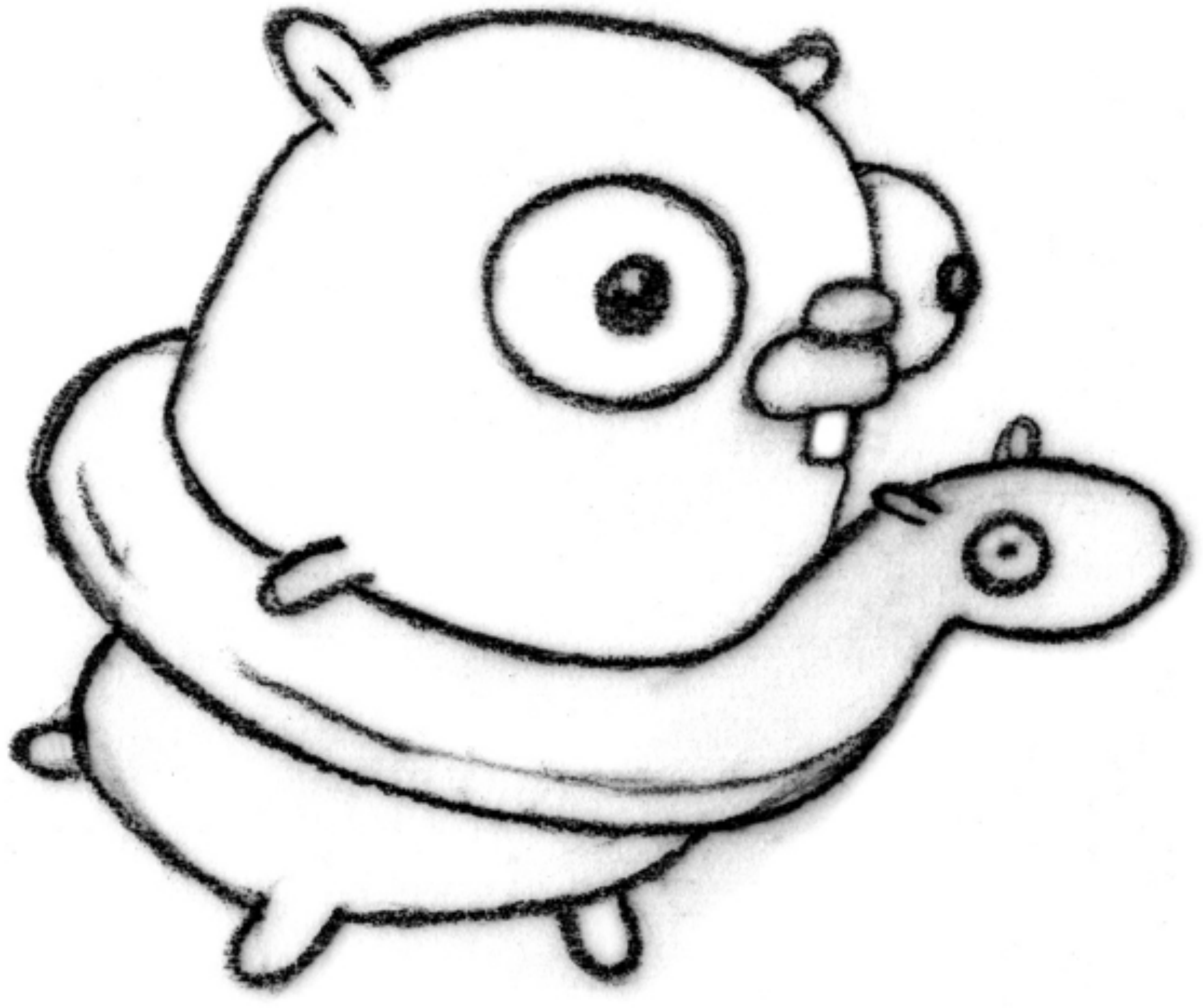
For resources e.g. queues

RED method

Tom Wilkie

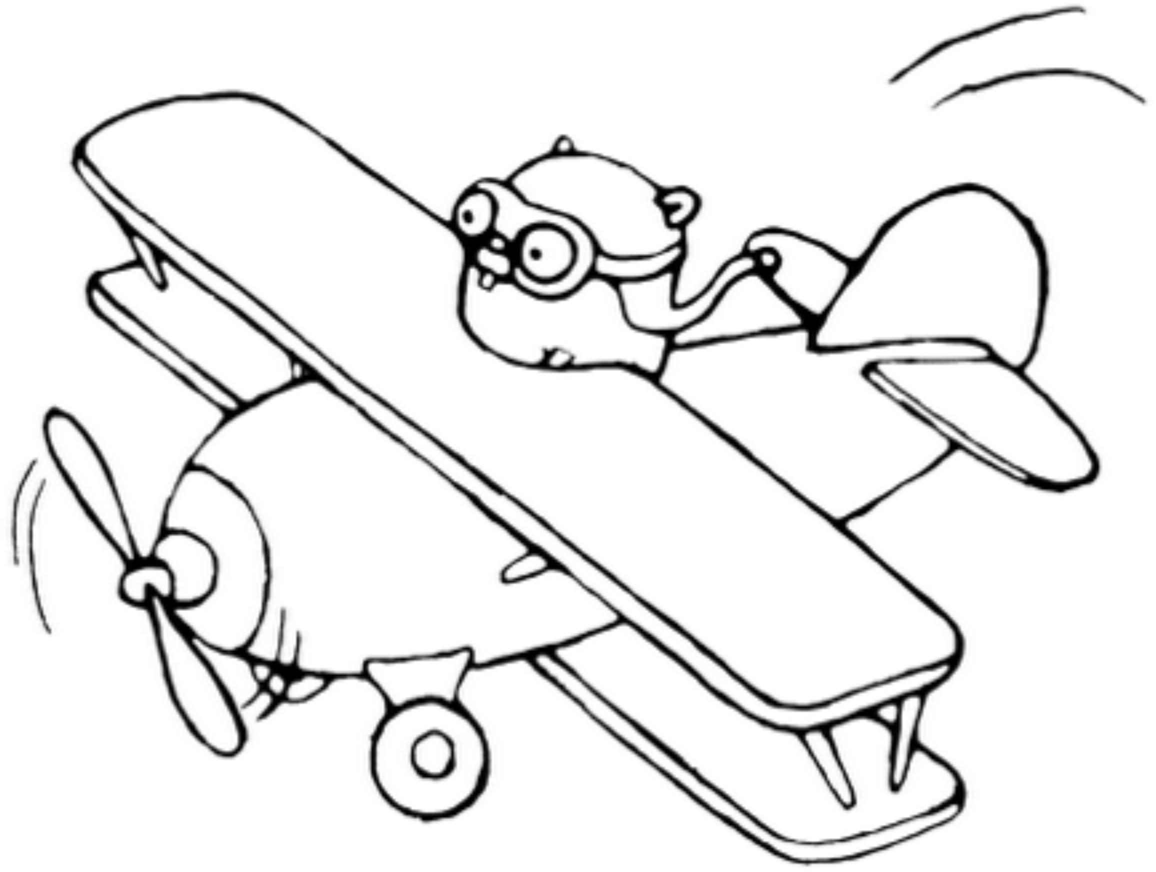
Request count (rate)
Error count (rate)
Duration

For e.g. endpoints



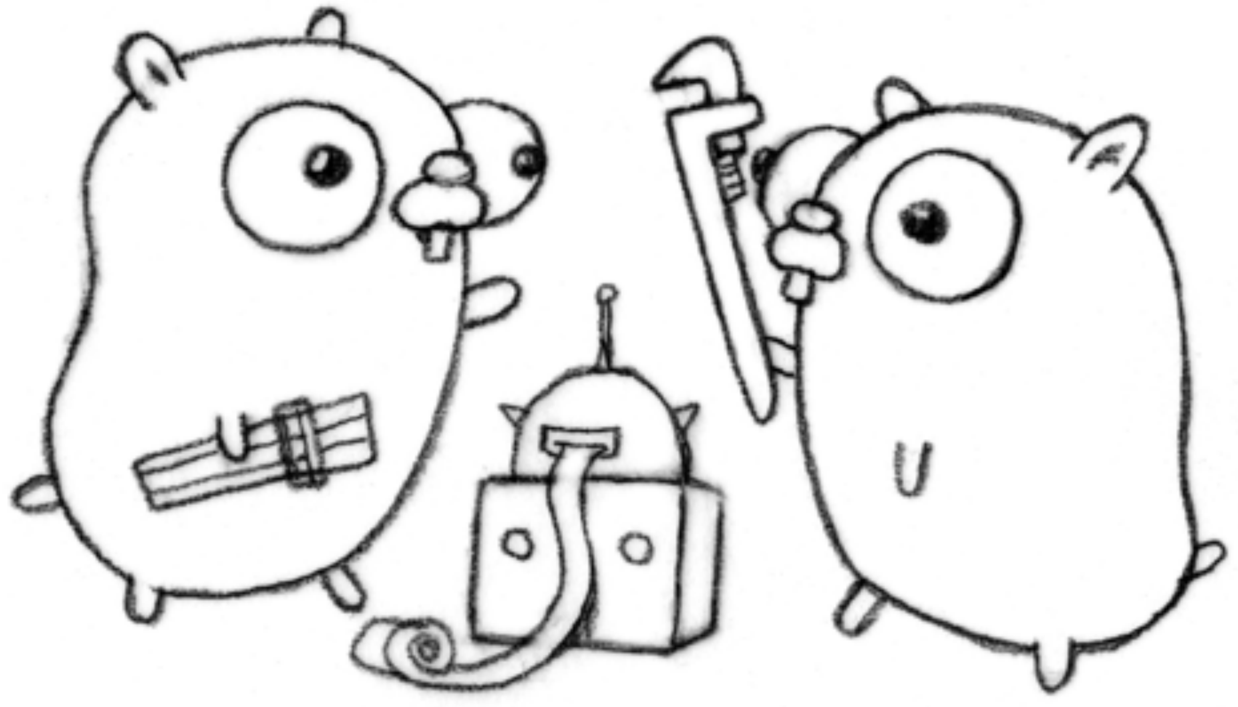
git apply

07 + Demo!



Challenge

Add a Gauge instrumenting in-flight requests –*or*–
Add a Histogram instrumenting **service** durations & compare

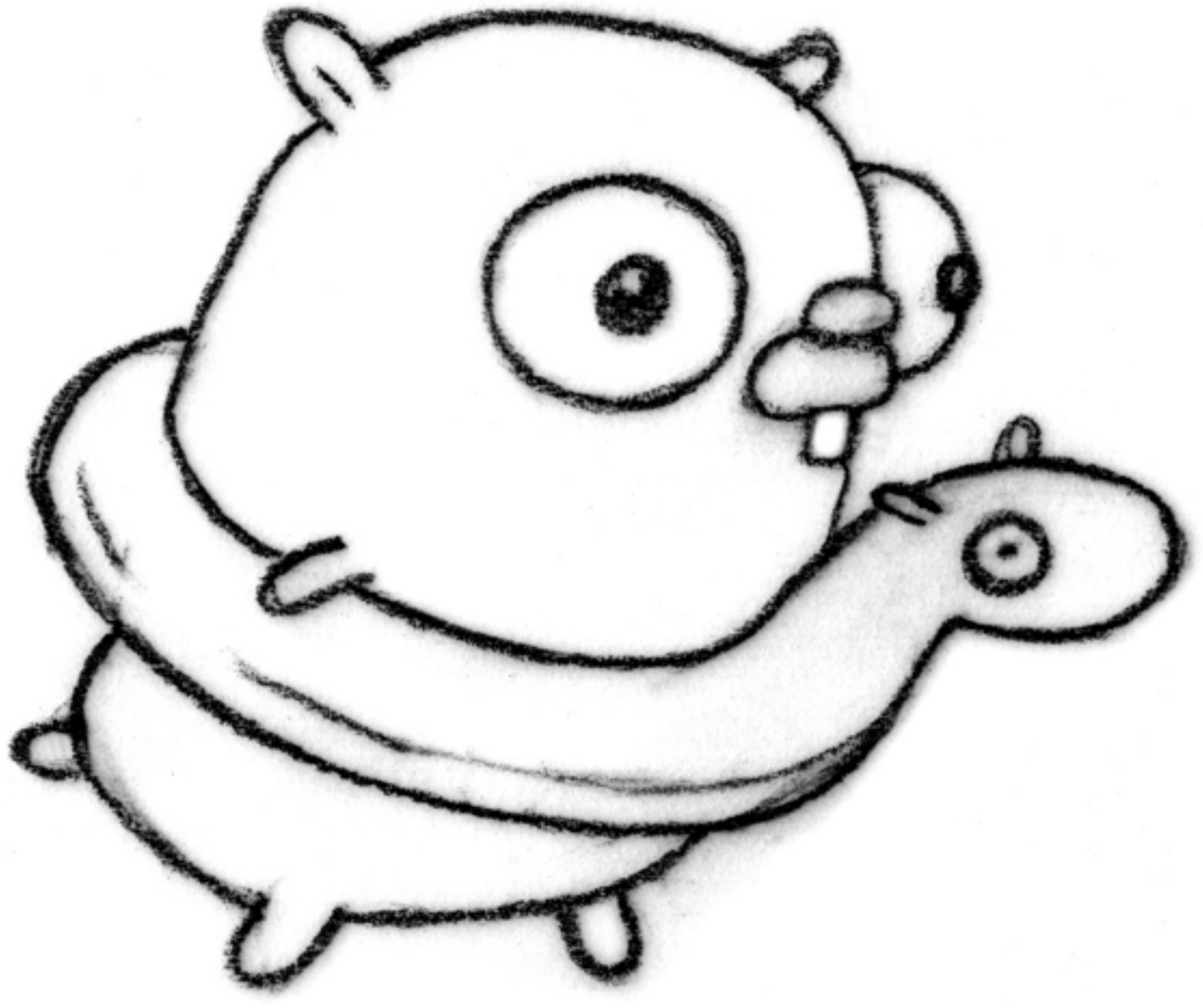


addsvc

Coda: structured logging, context, error handling

Structured logging

A good idea



git apply

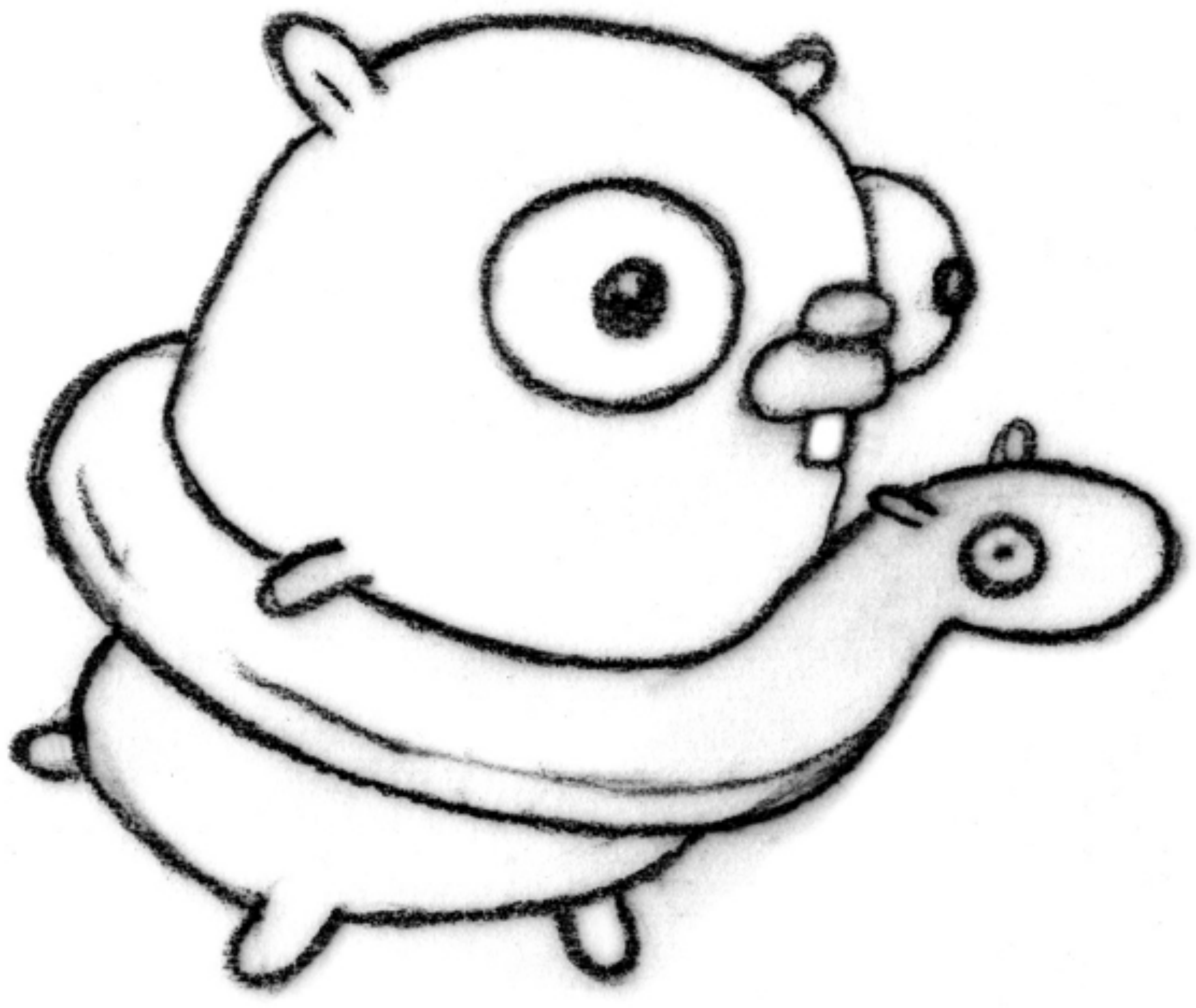
08, 09, 10

Logging v. instrumentation

peter.bourgon.org/blog/2016/02/07/logging-v-instrumentation.html

Context

Request-scoped data

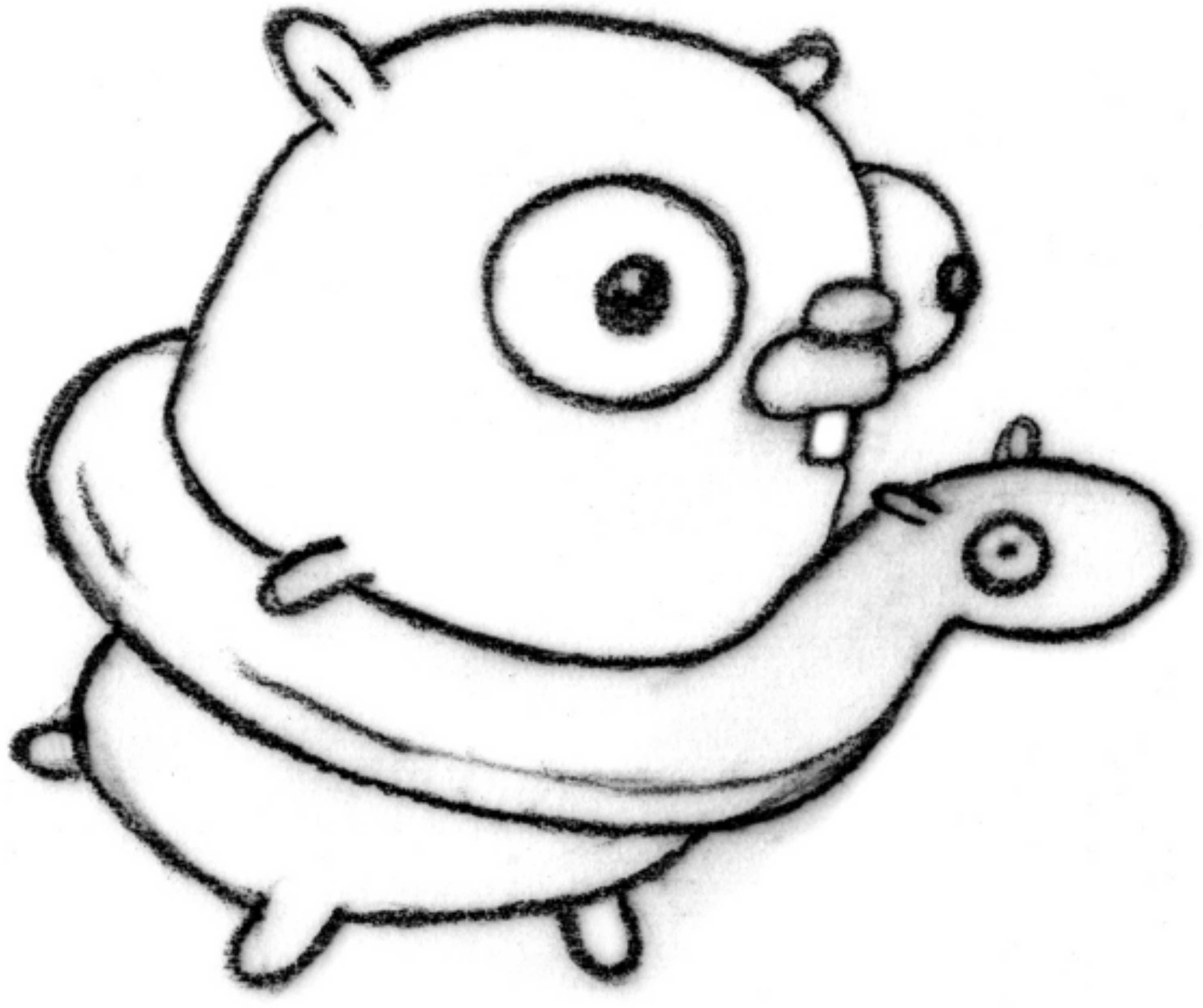


git apply

11

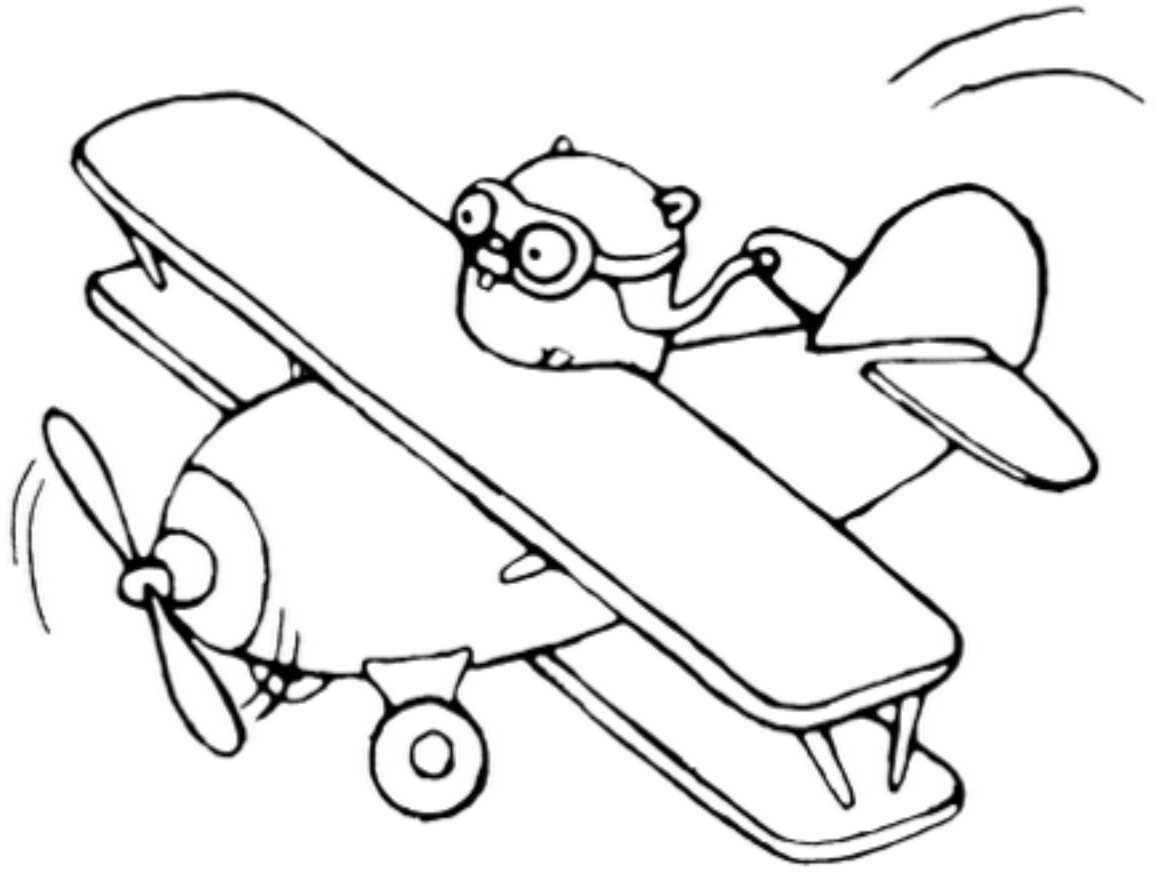
Error handling

Mapping business to transport domain



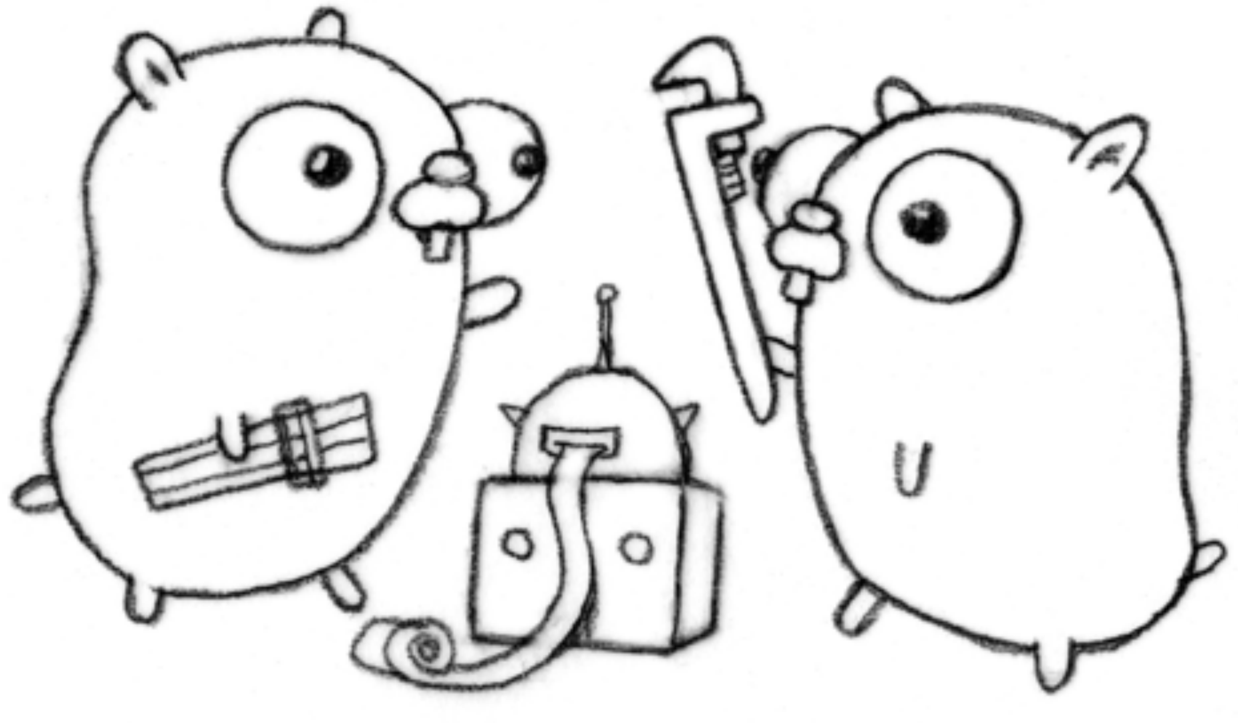
git apply

12, 13, 14



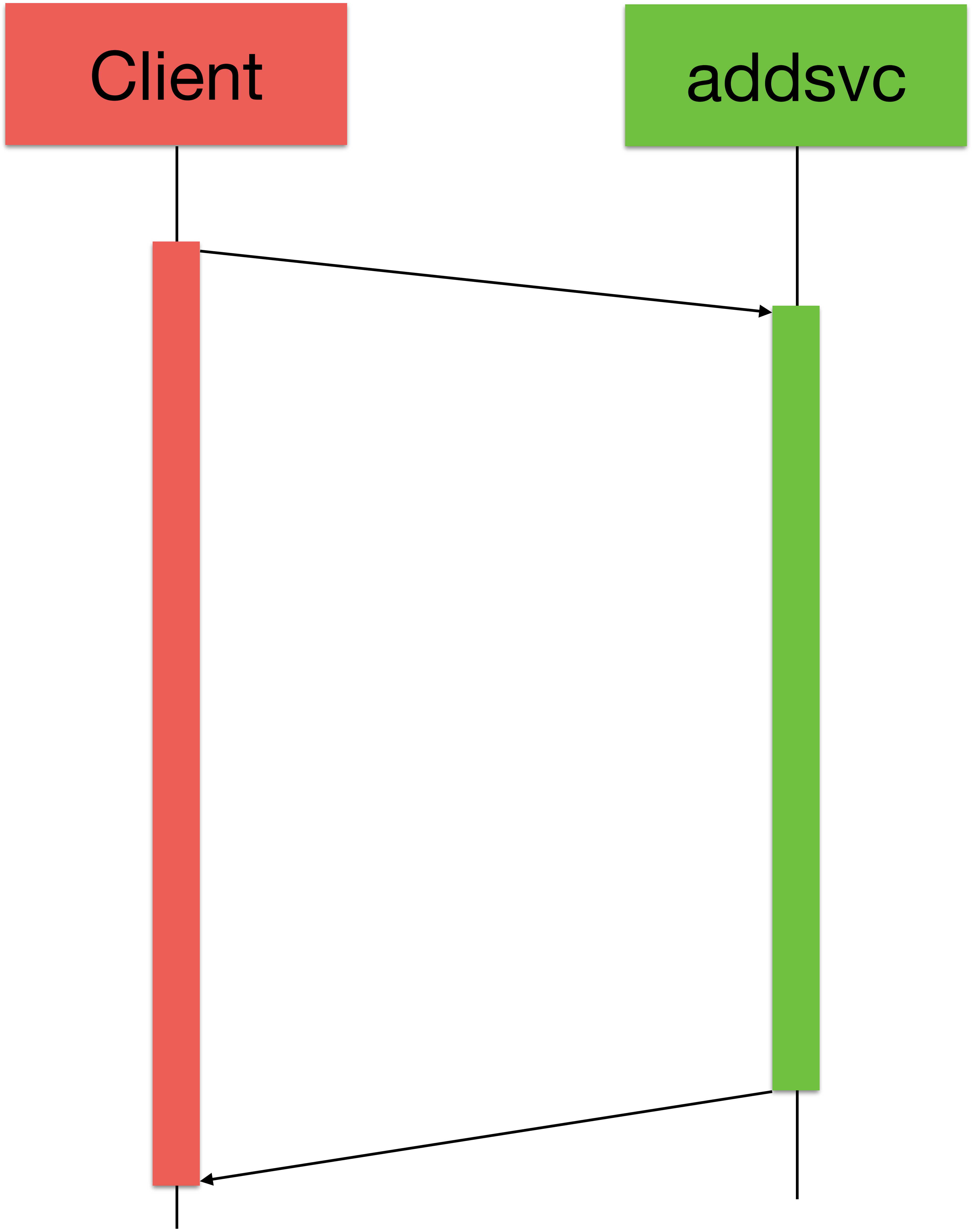
Challenge

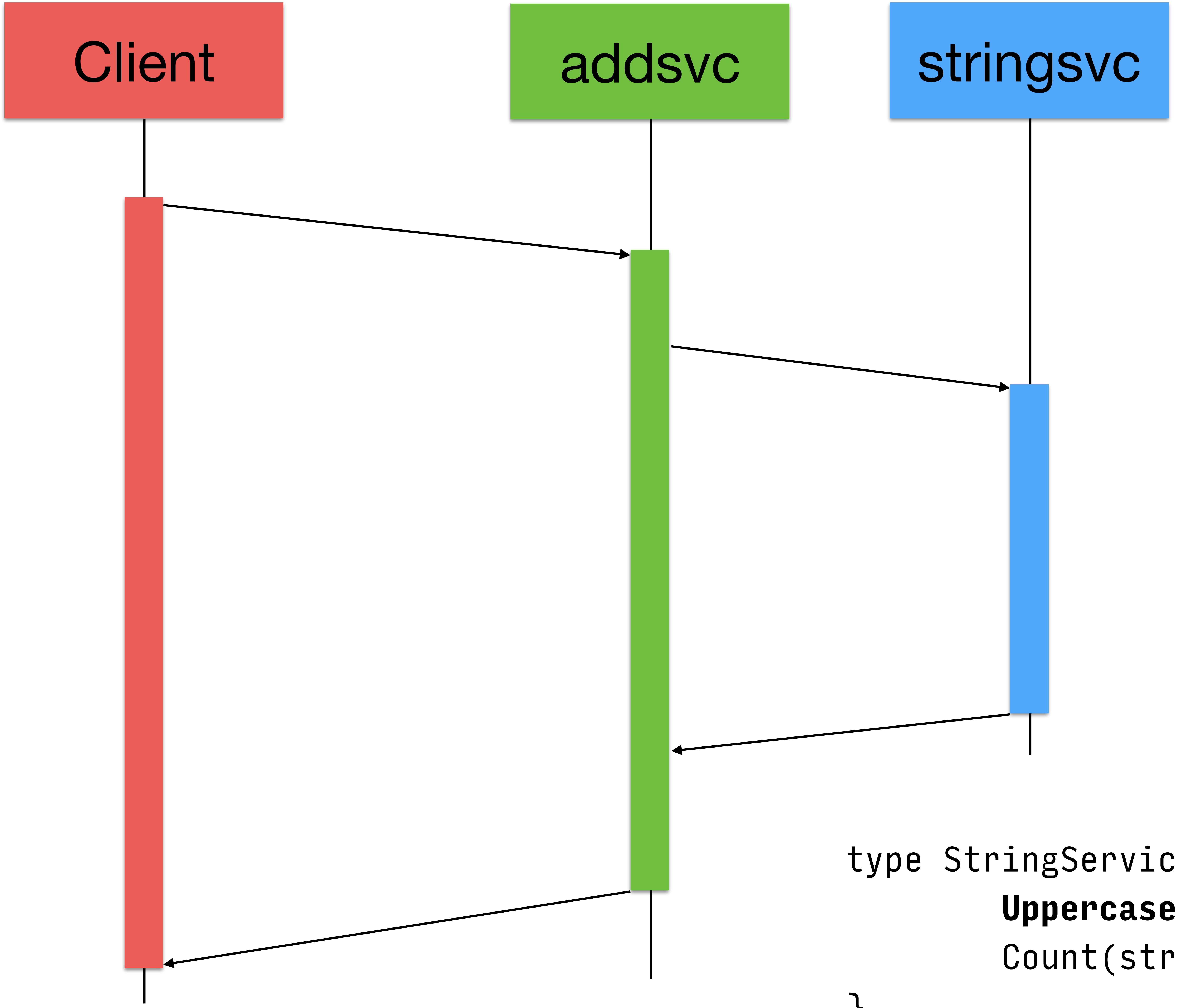
Create a new business logic error that will return HTTP 418



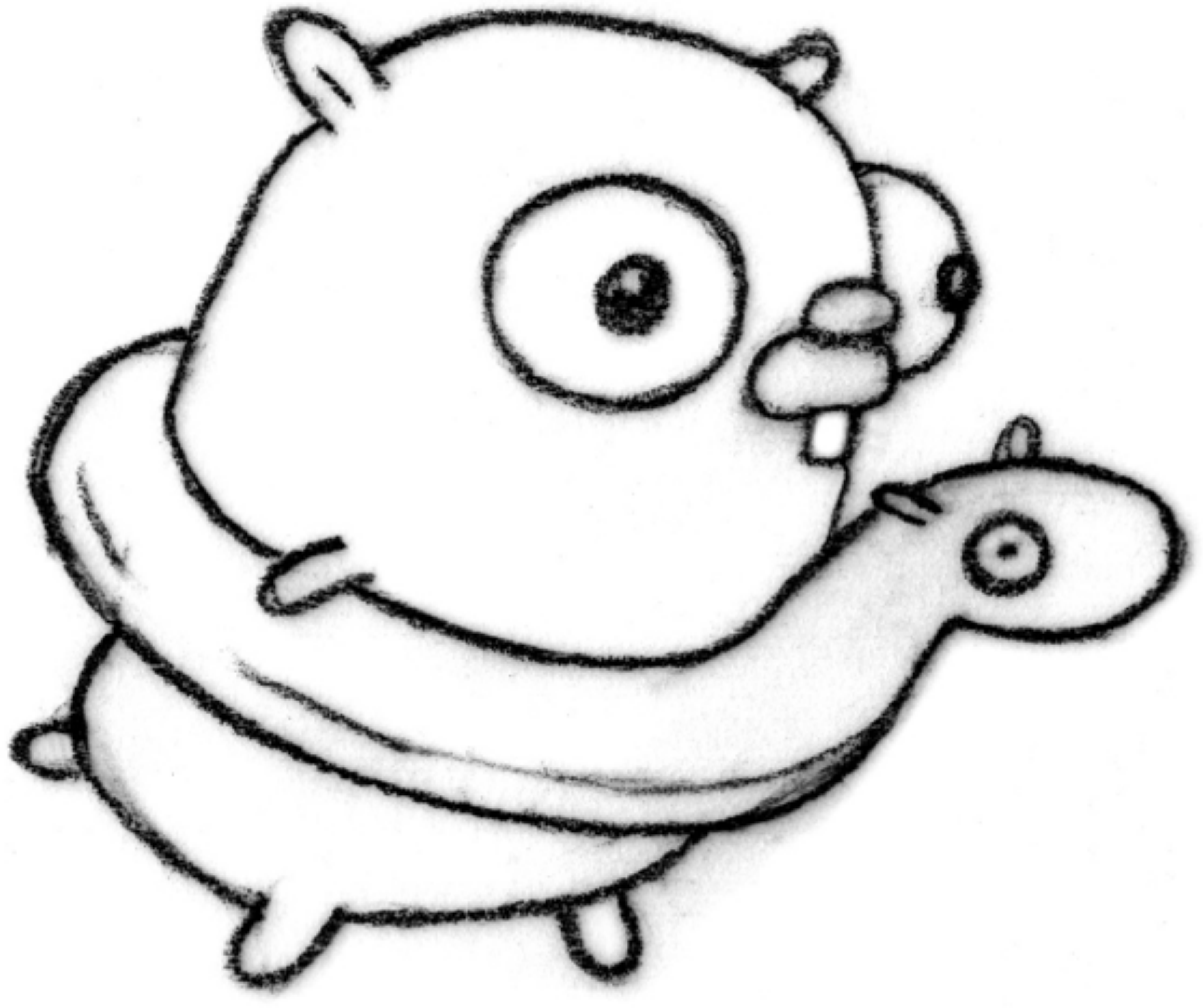
addsvc

Dependency on stringsvc


Client
addsvc

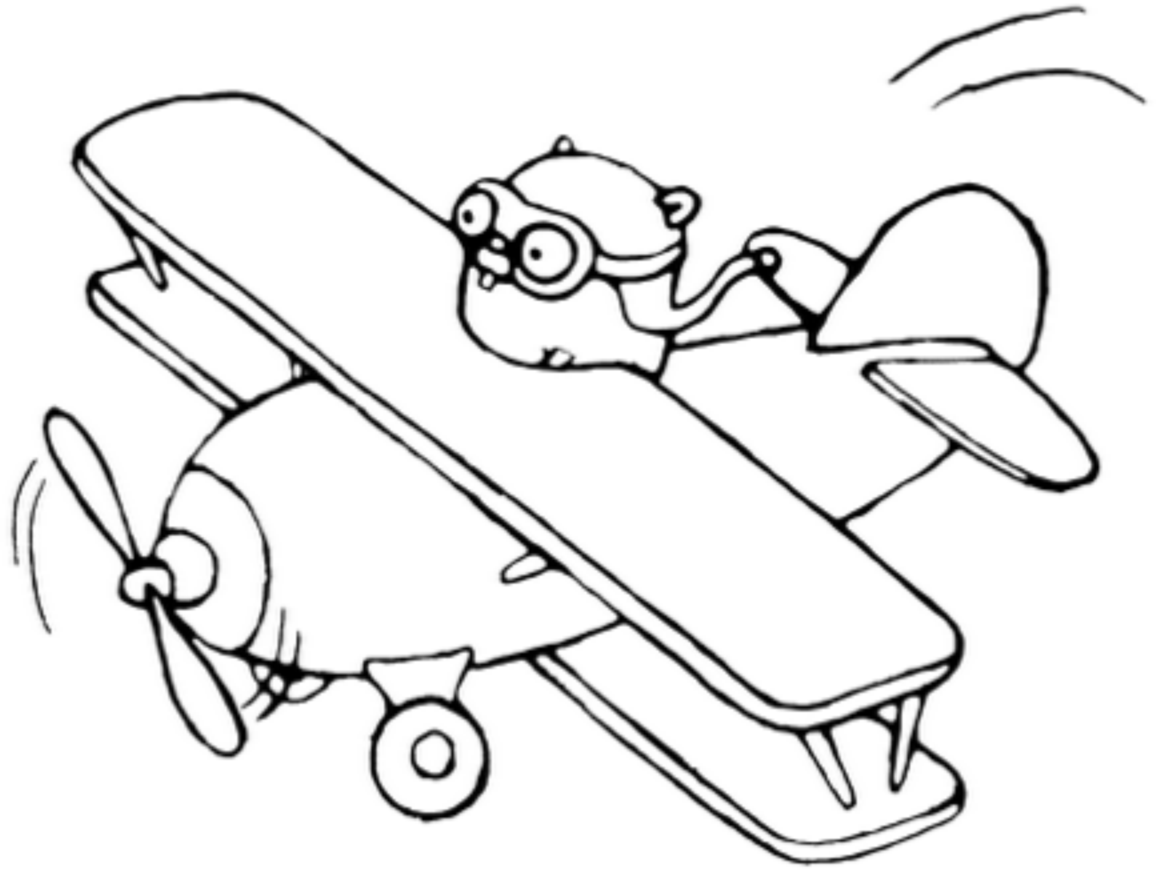


```
type StringService interface {
        Uppercase(string) (string, error)
        Count(string) int
}
```



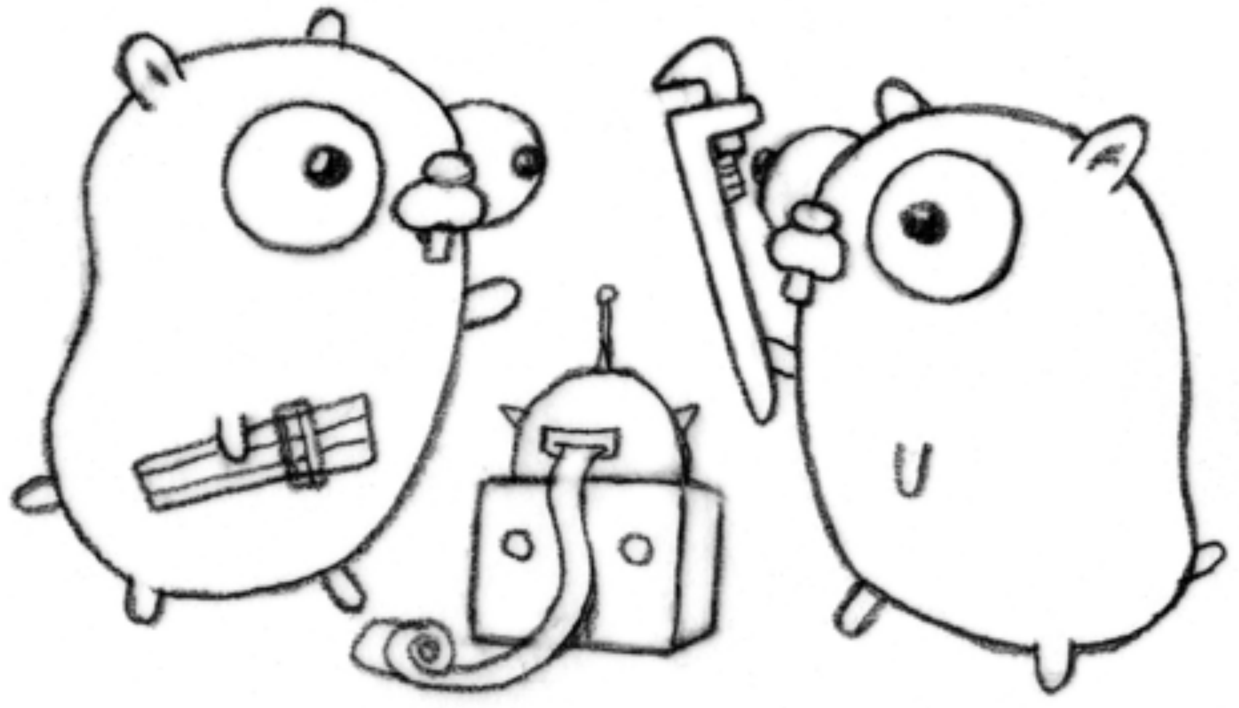
git apply

15, 16, 17



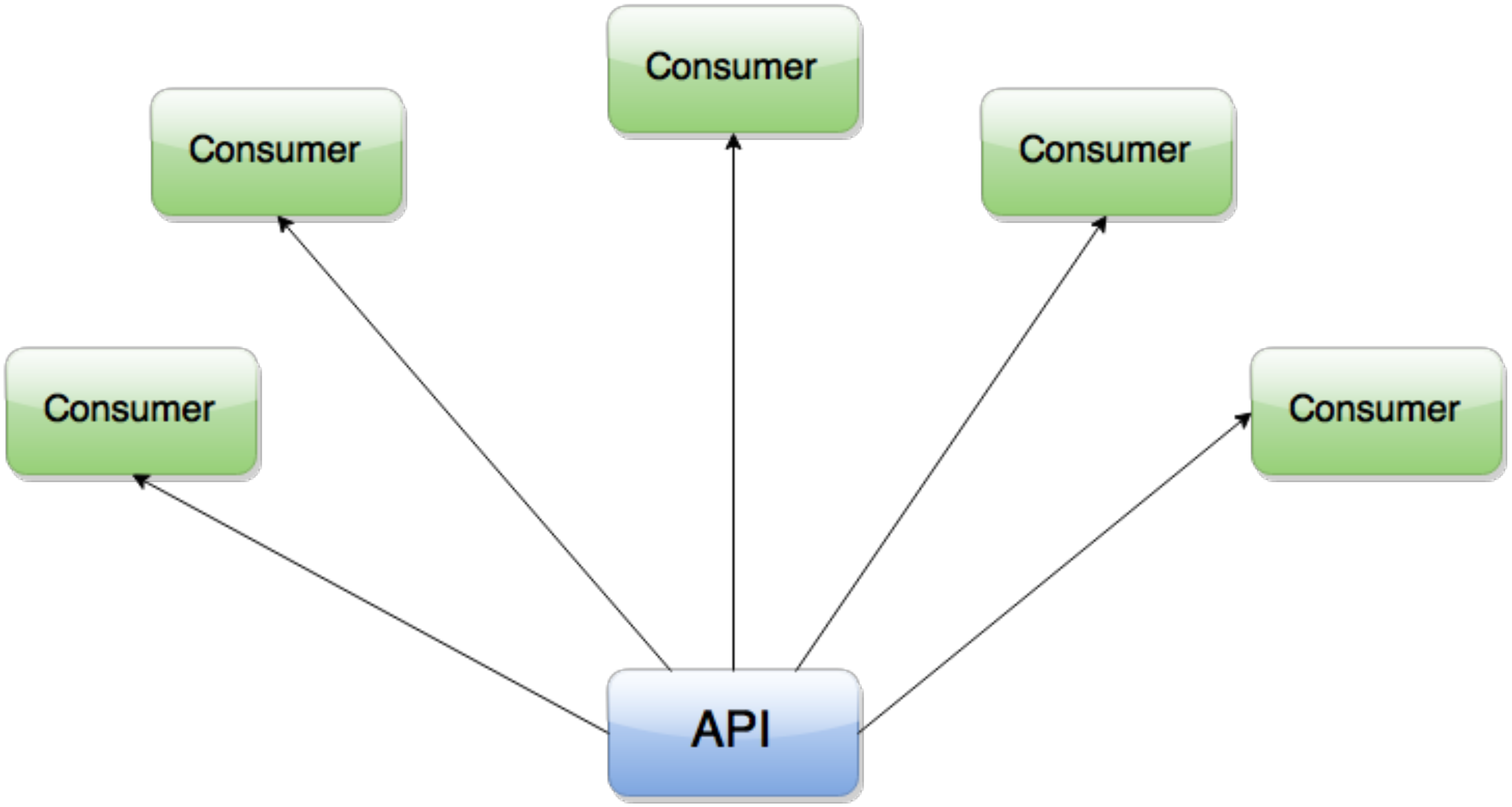
Challenge

Make UPPERCASE via stringsvc optional (18)

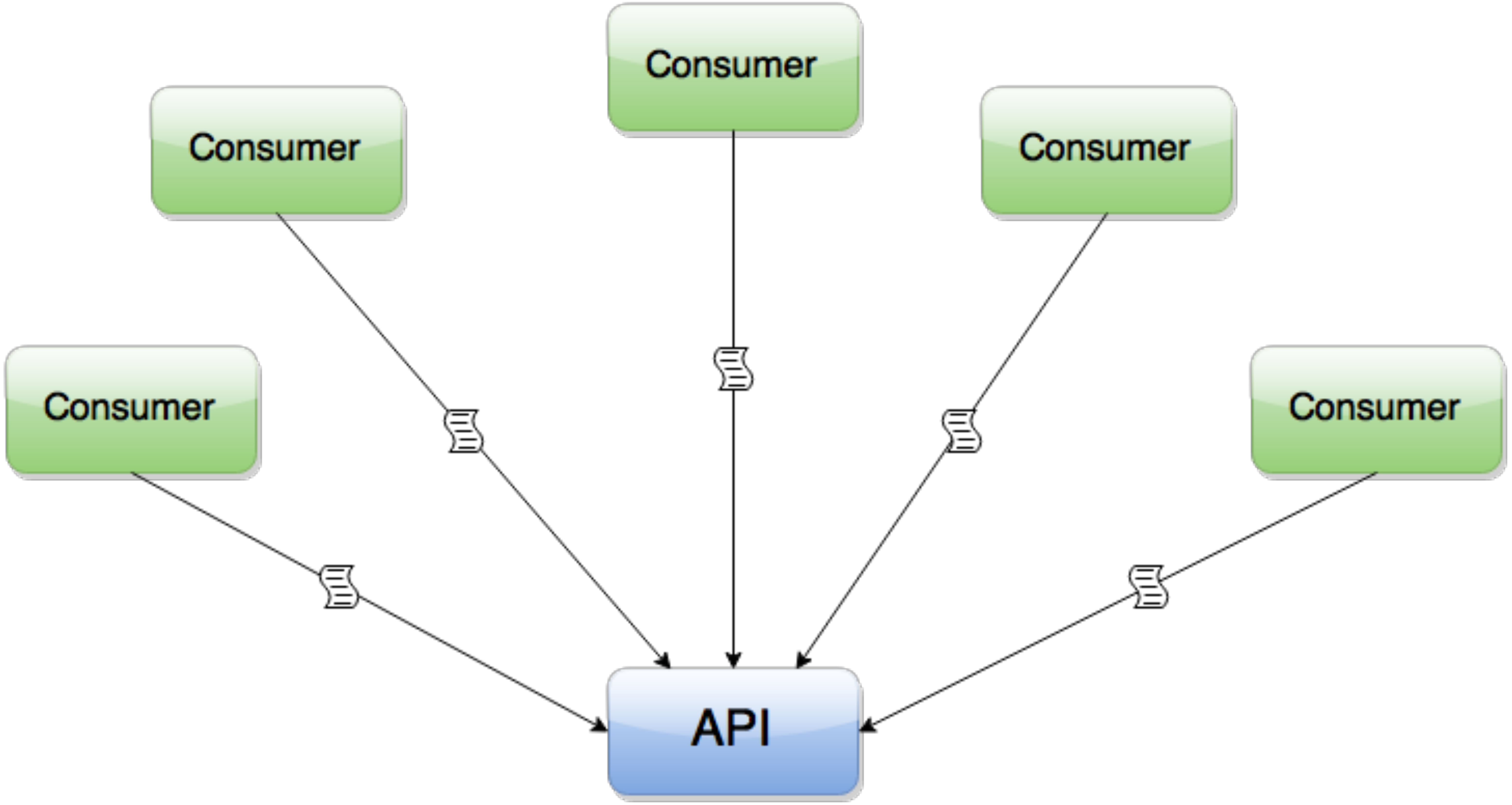


addsvc

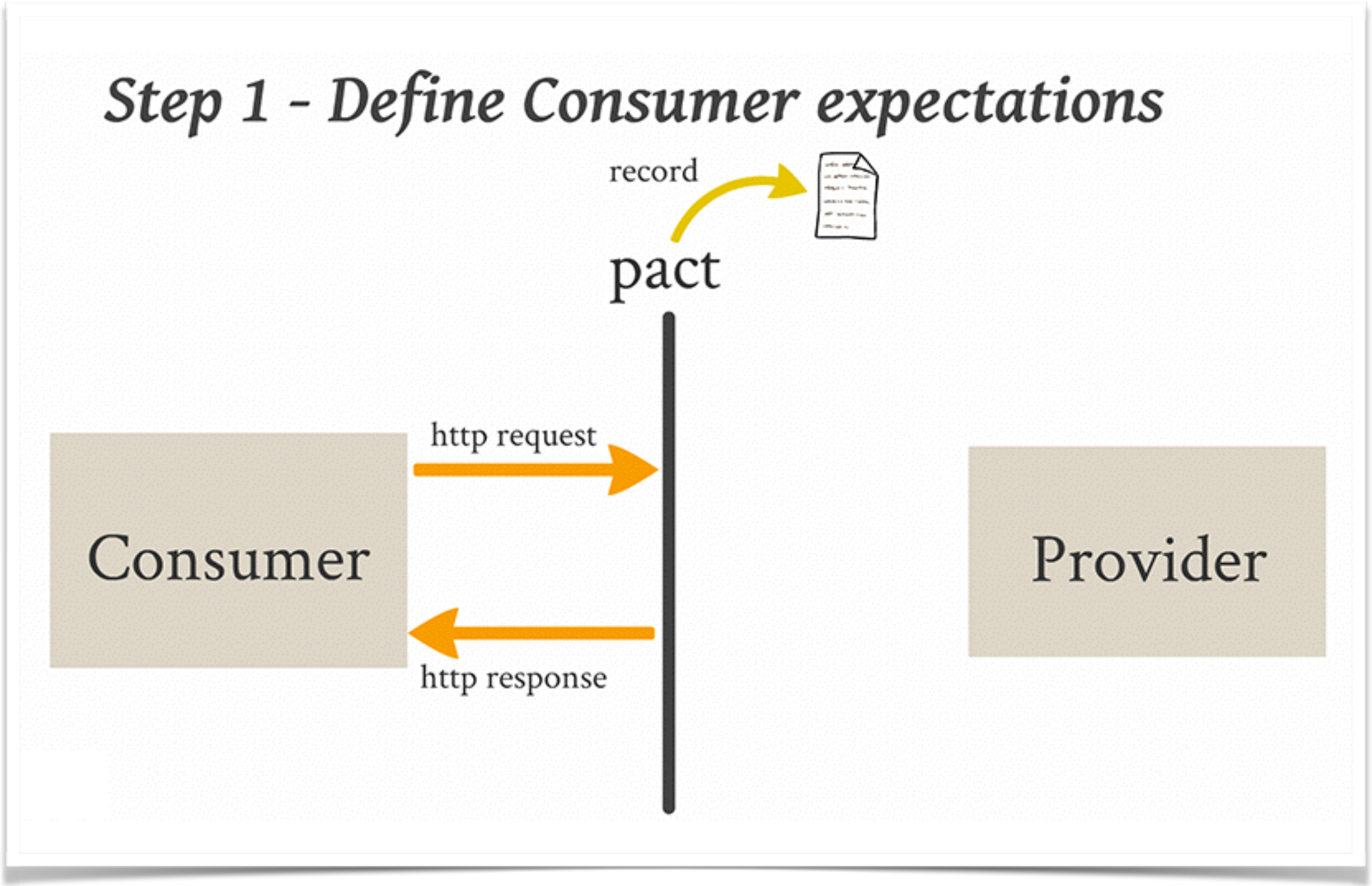
Contract testing with Pact

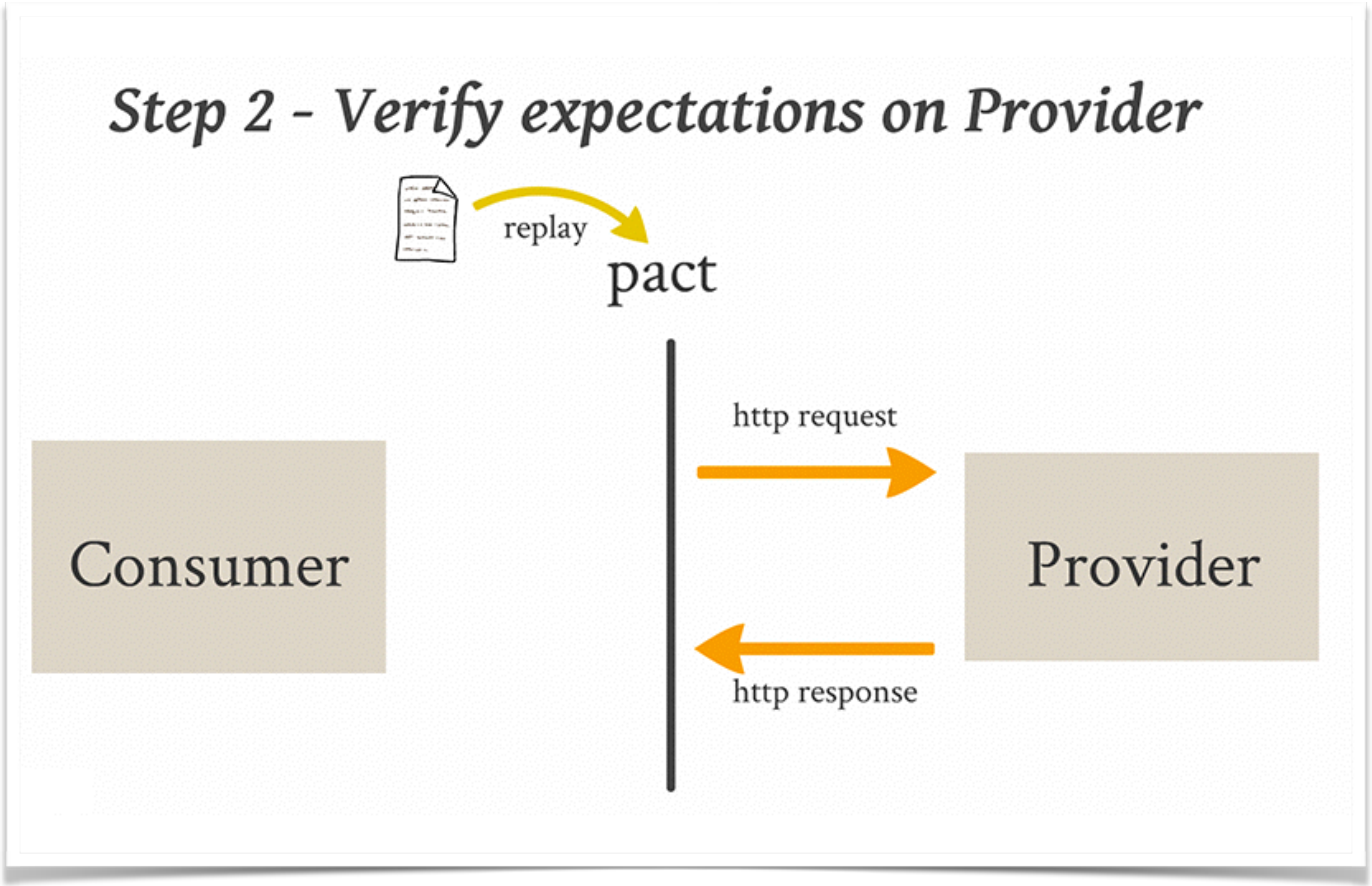


http://dius.com.au/2016/02/03/microservices-pact/

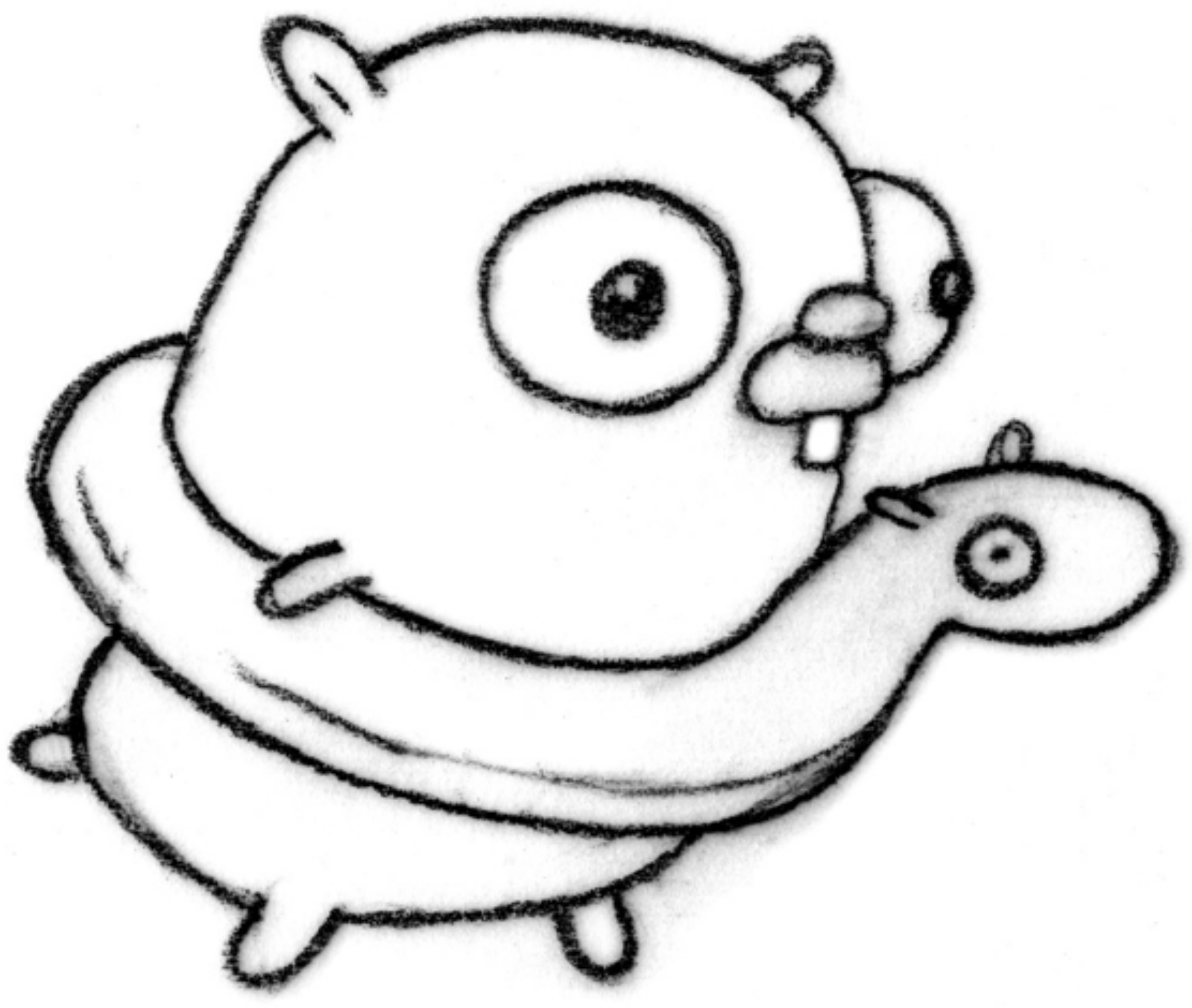


http://dius.com.au/2016/02/03/microservices-pact/



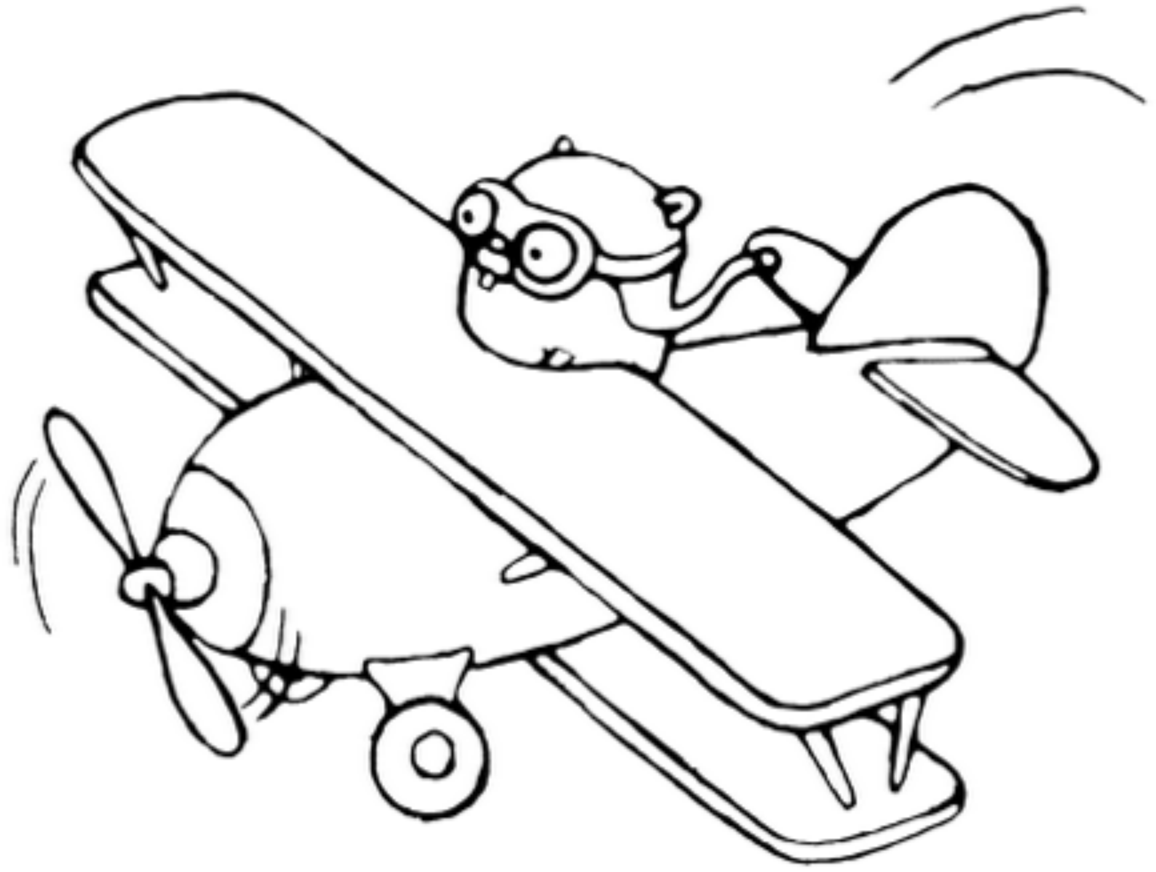


http://dius.com.au/2016/02/03/microservices-pact/



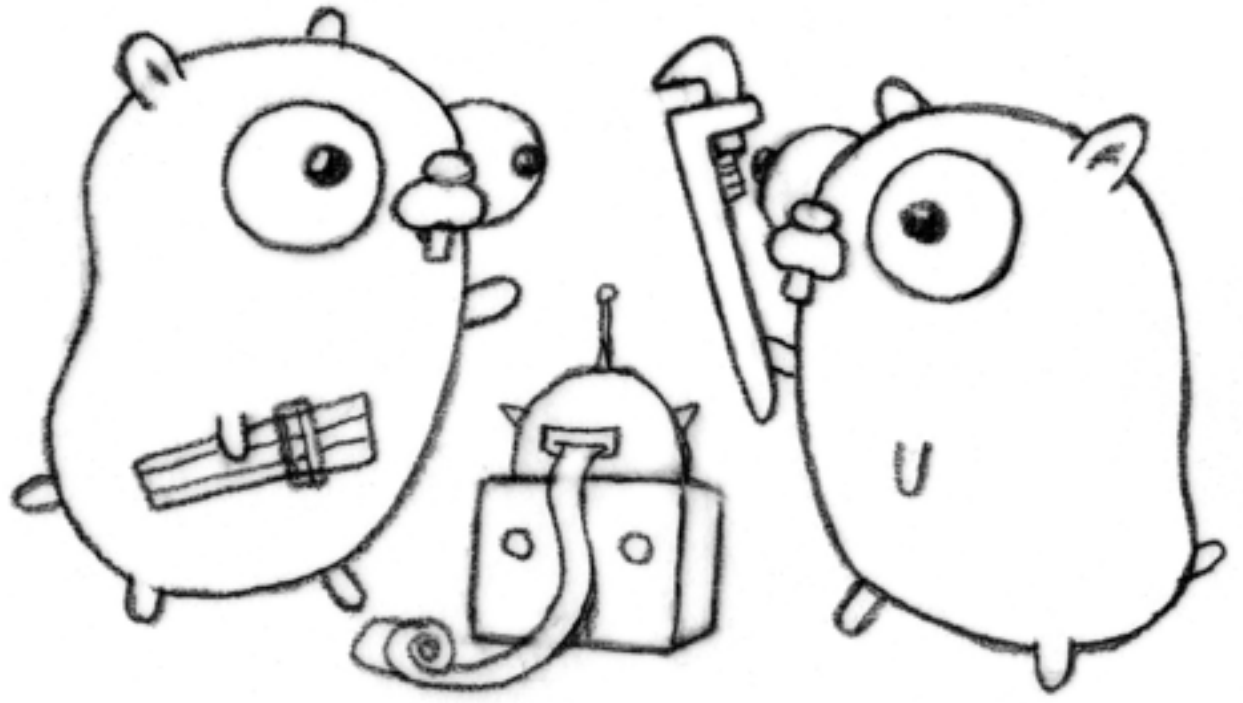
git apply

19



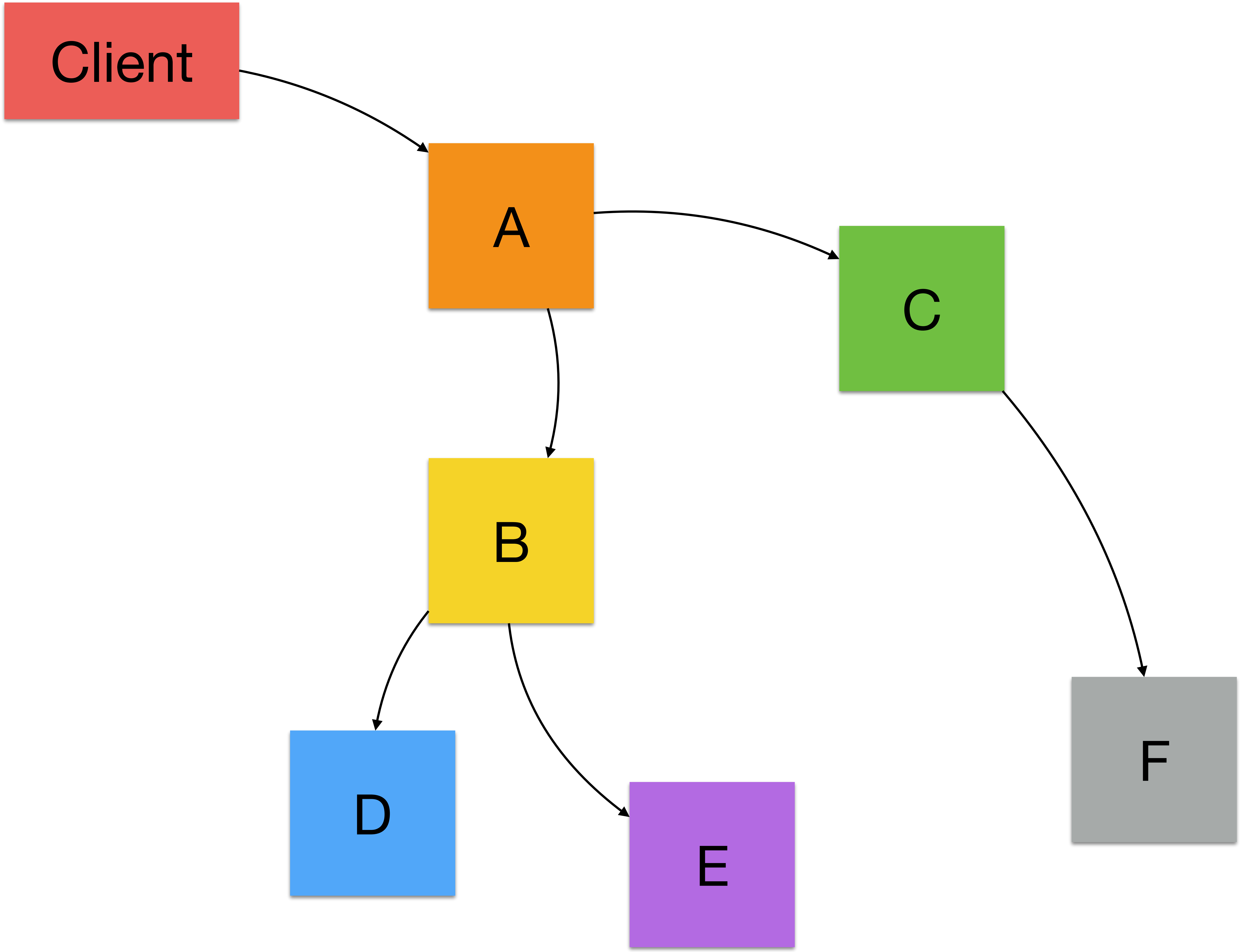
Challenge

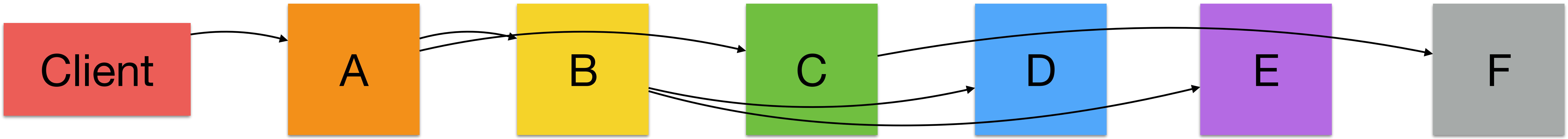
Add a new Pact contract, with a more sophisticated test case.

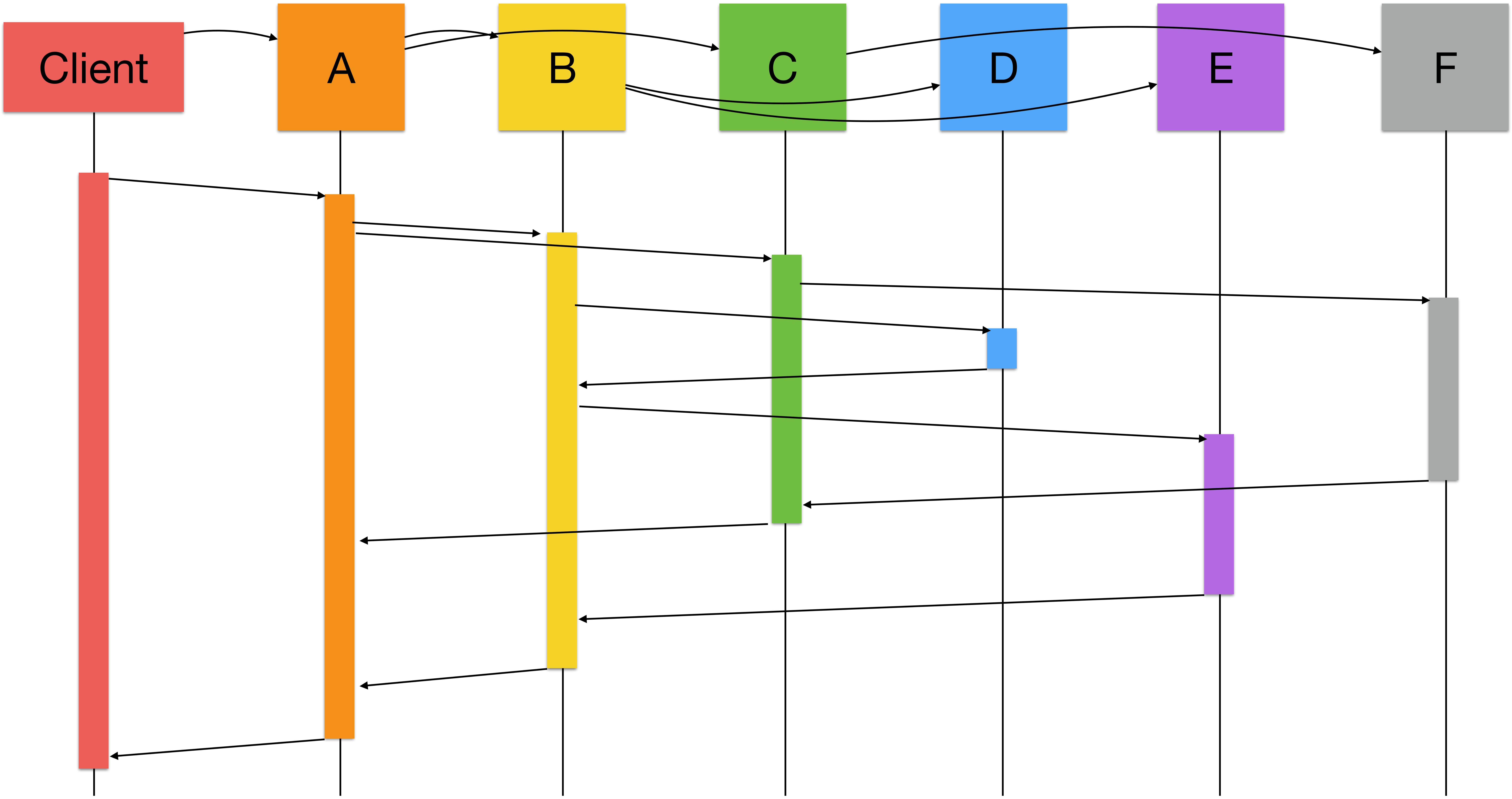


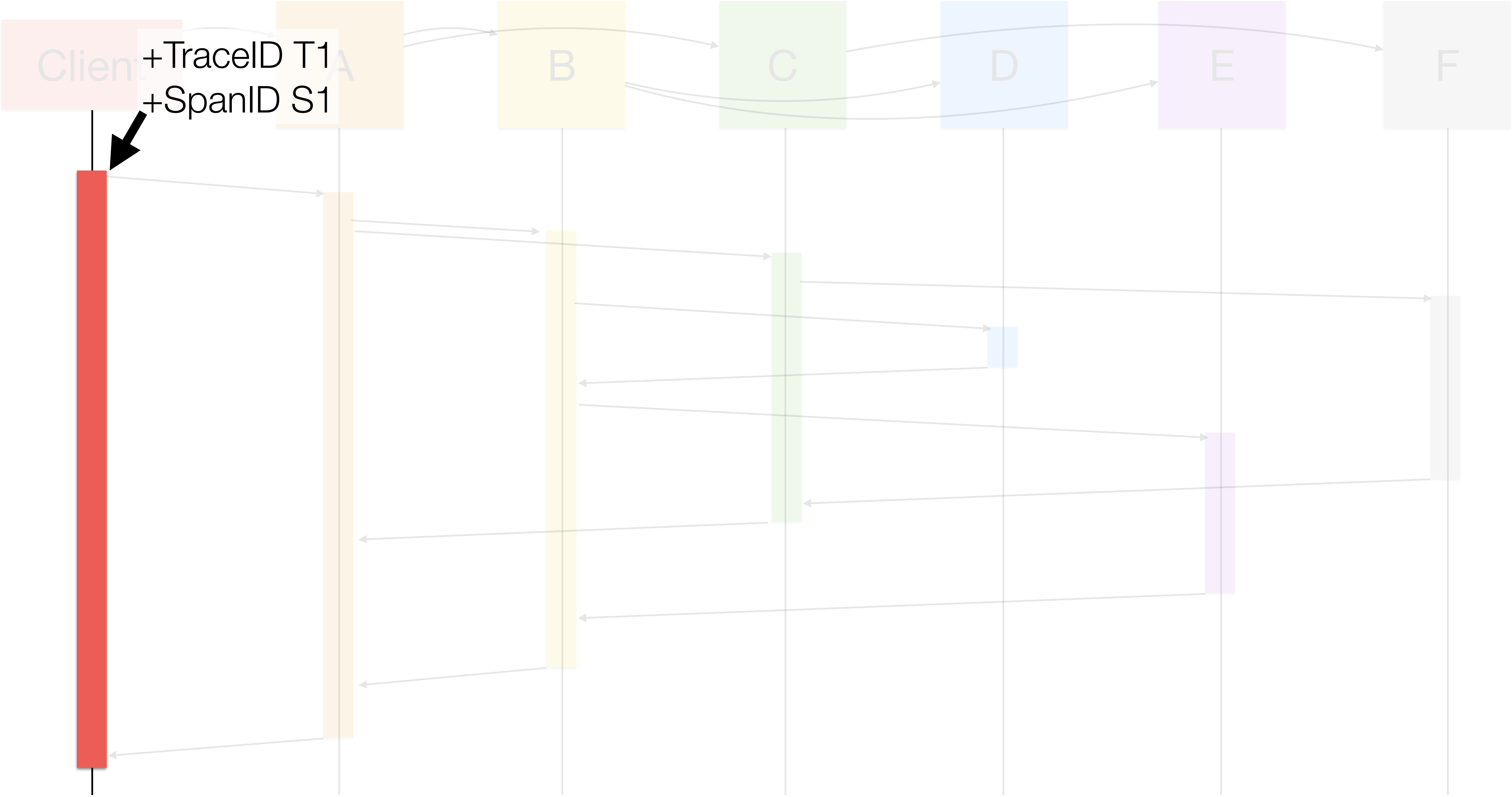
addsvc

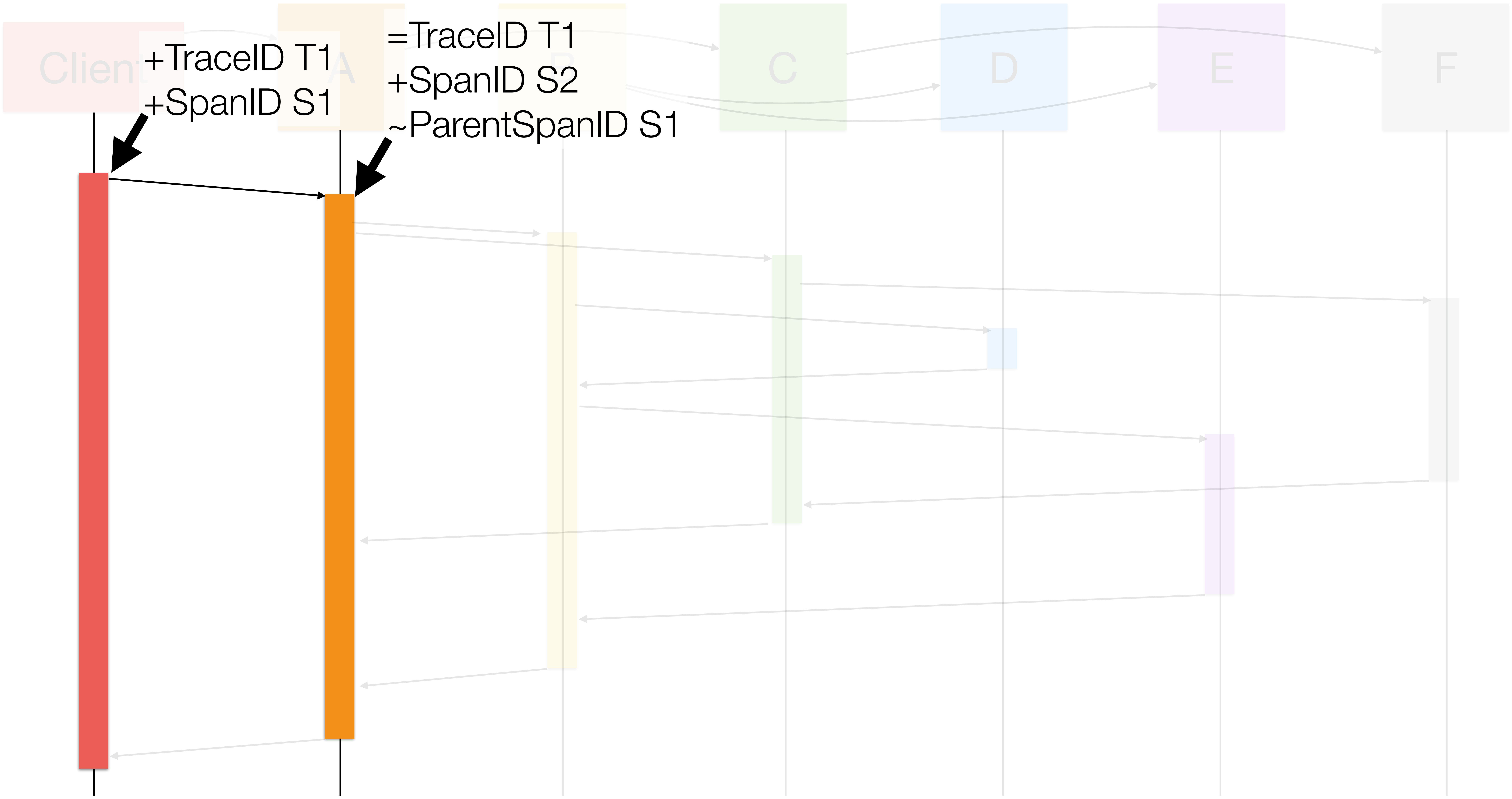
Distributed tracing with Tracer


Client
A
C
B
D
E
F


Client
A
B
C
D
E
F

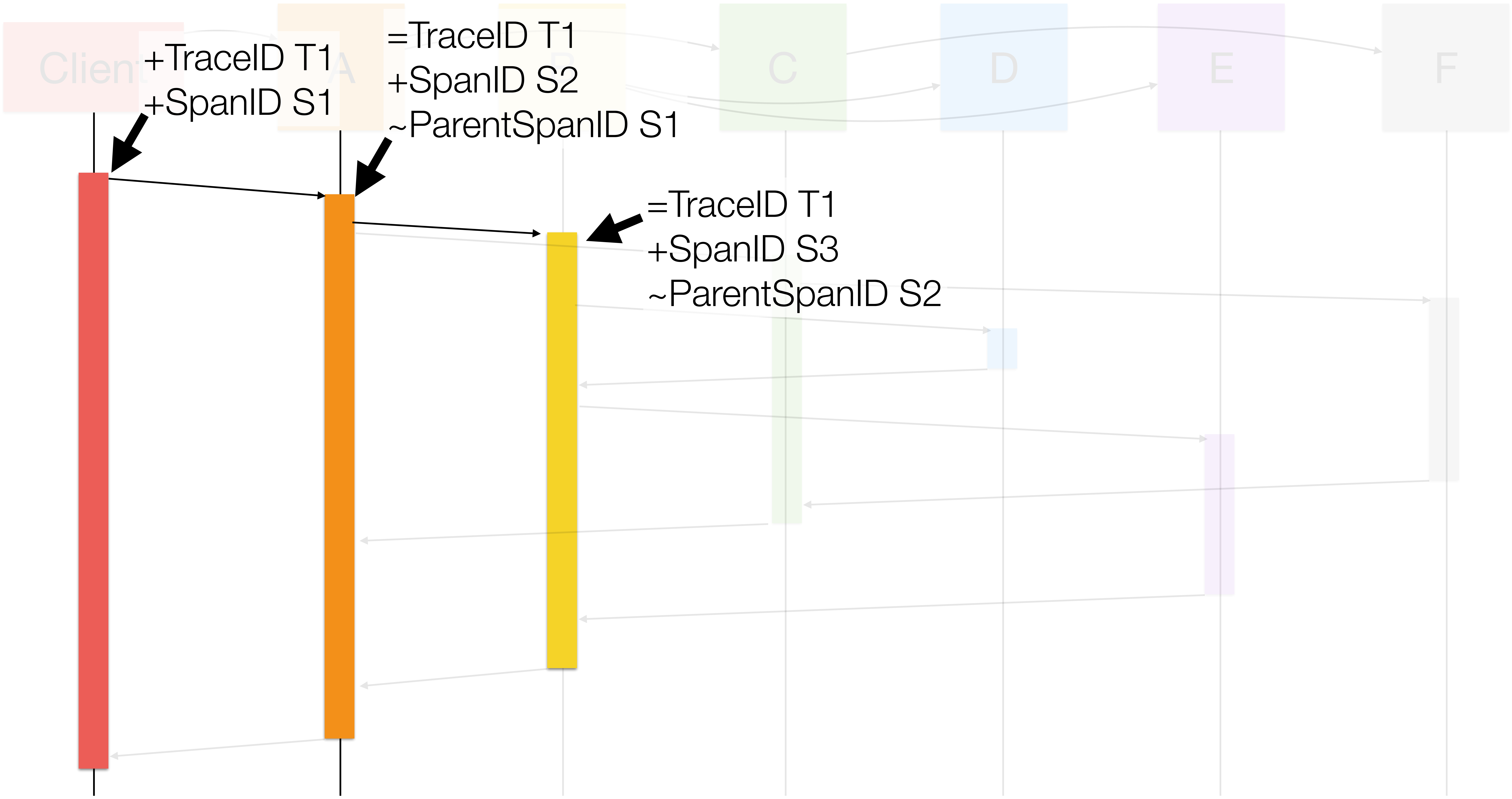

Client
A
B
C
D
E
F


Client
A
B
C
D
E
F
+TraceID T1
+SpanID S1



+TraceID T1
+SpanID S1

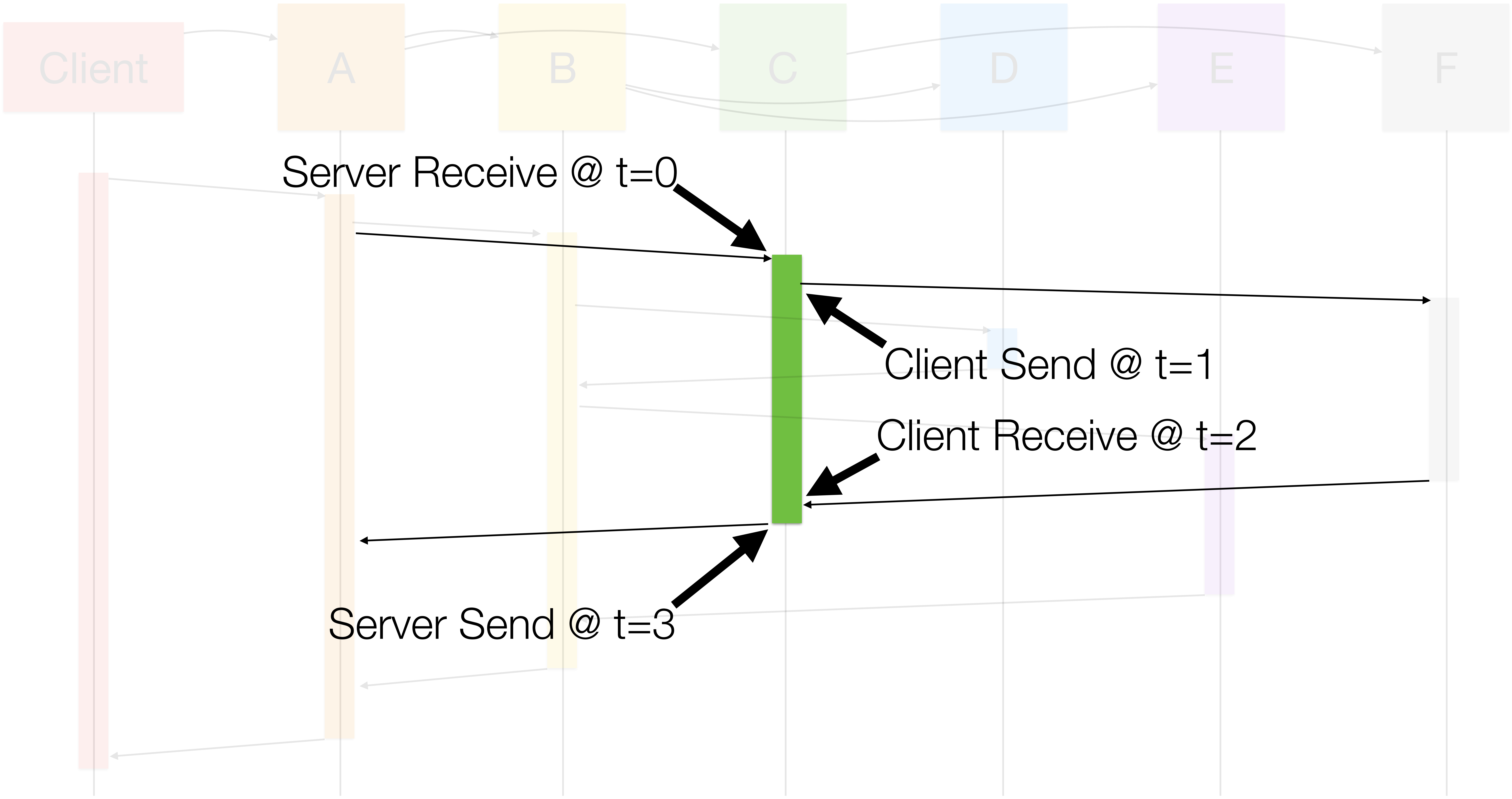
=TraceID T1
+SpanID S2
~ParentSpanID S1



+TraceID T1
+SpanID S1

=TraceID T1
+SpanID S2
~ParentSpanID S1

=TraceID T1
+SpanID S3
~ParentSpanID S2

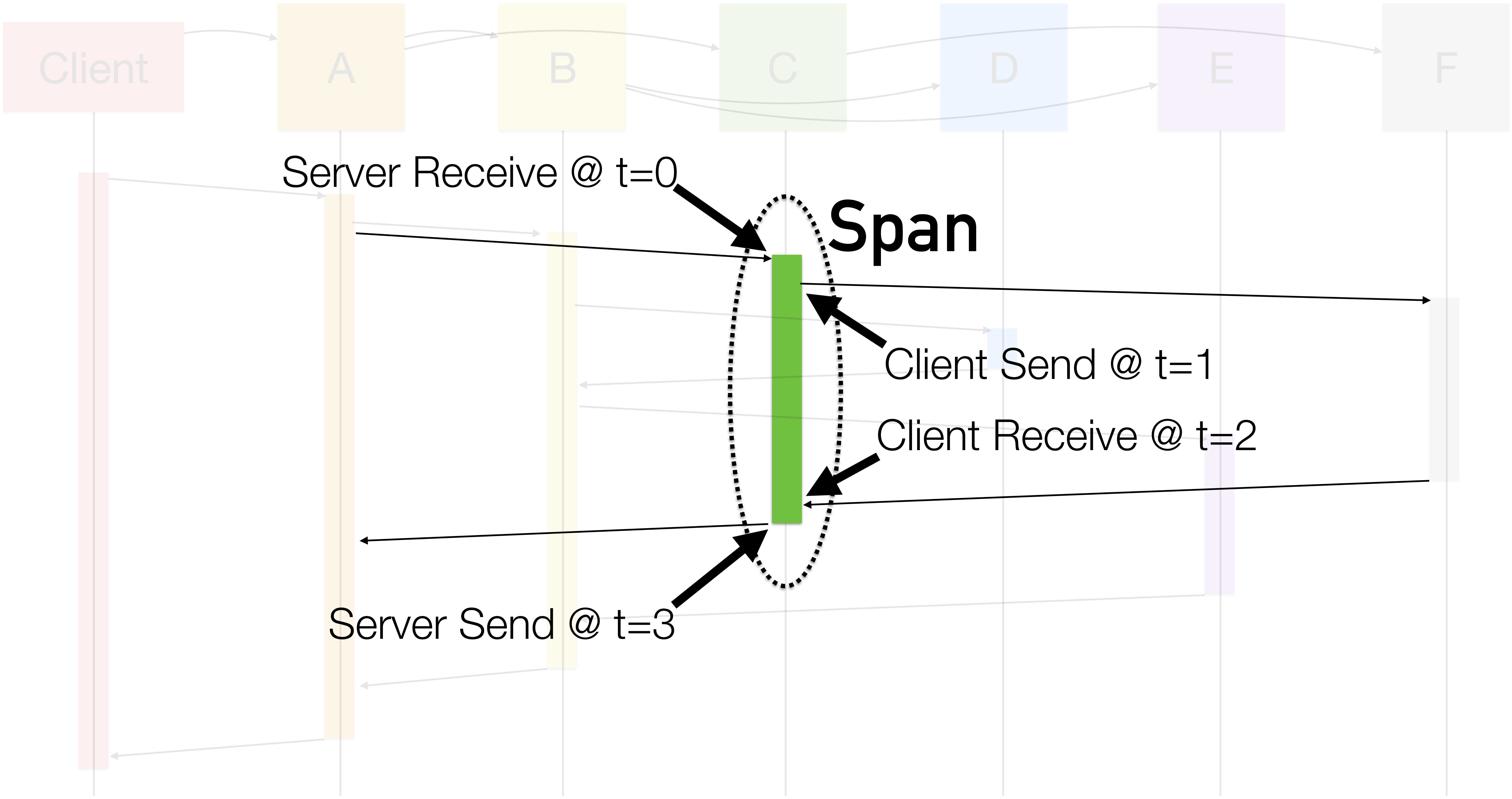


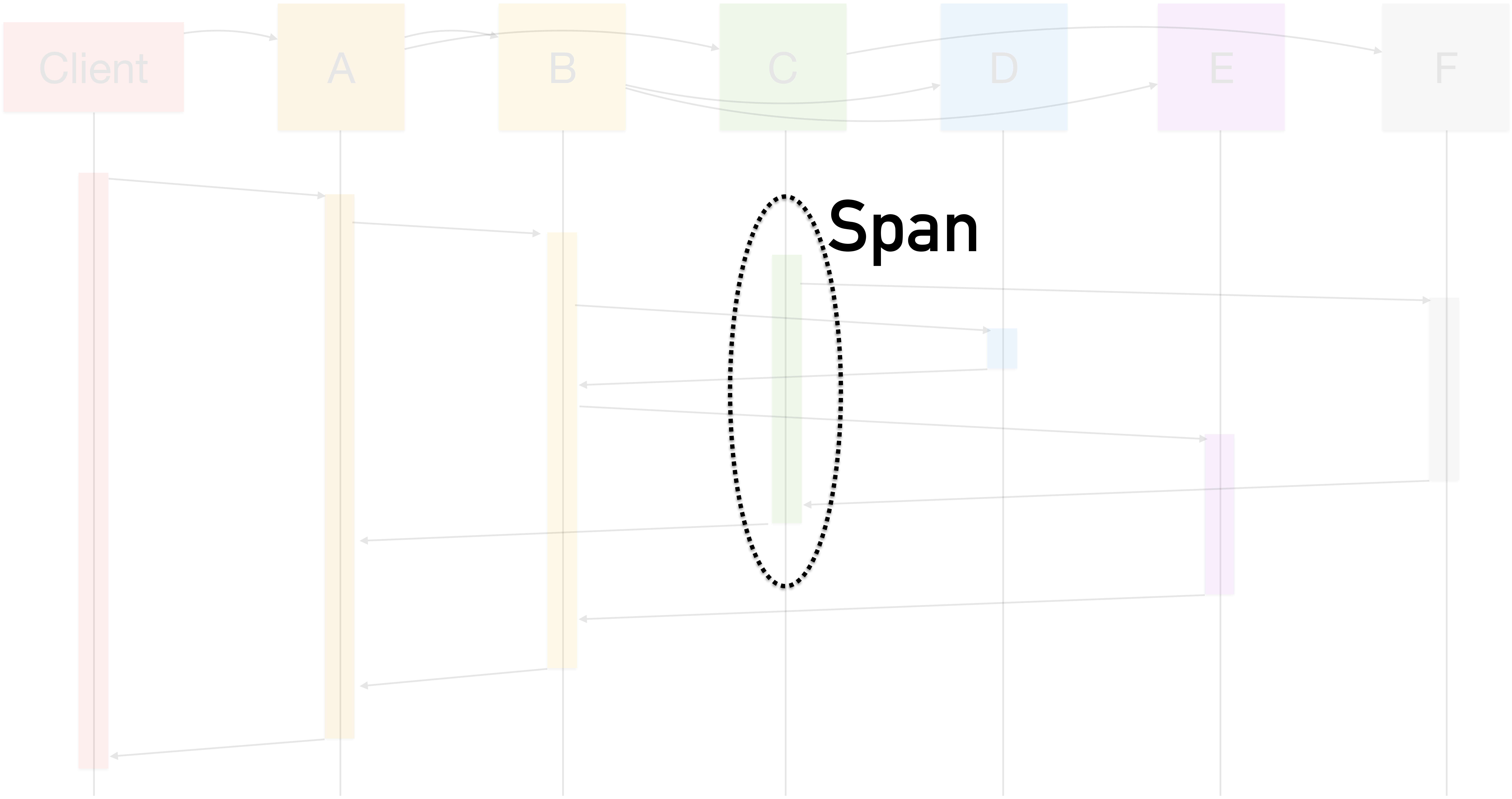
Server Receive @ t=0

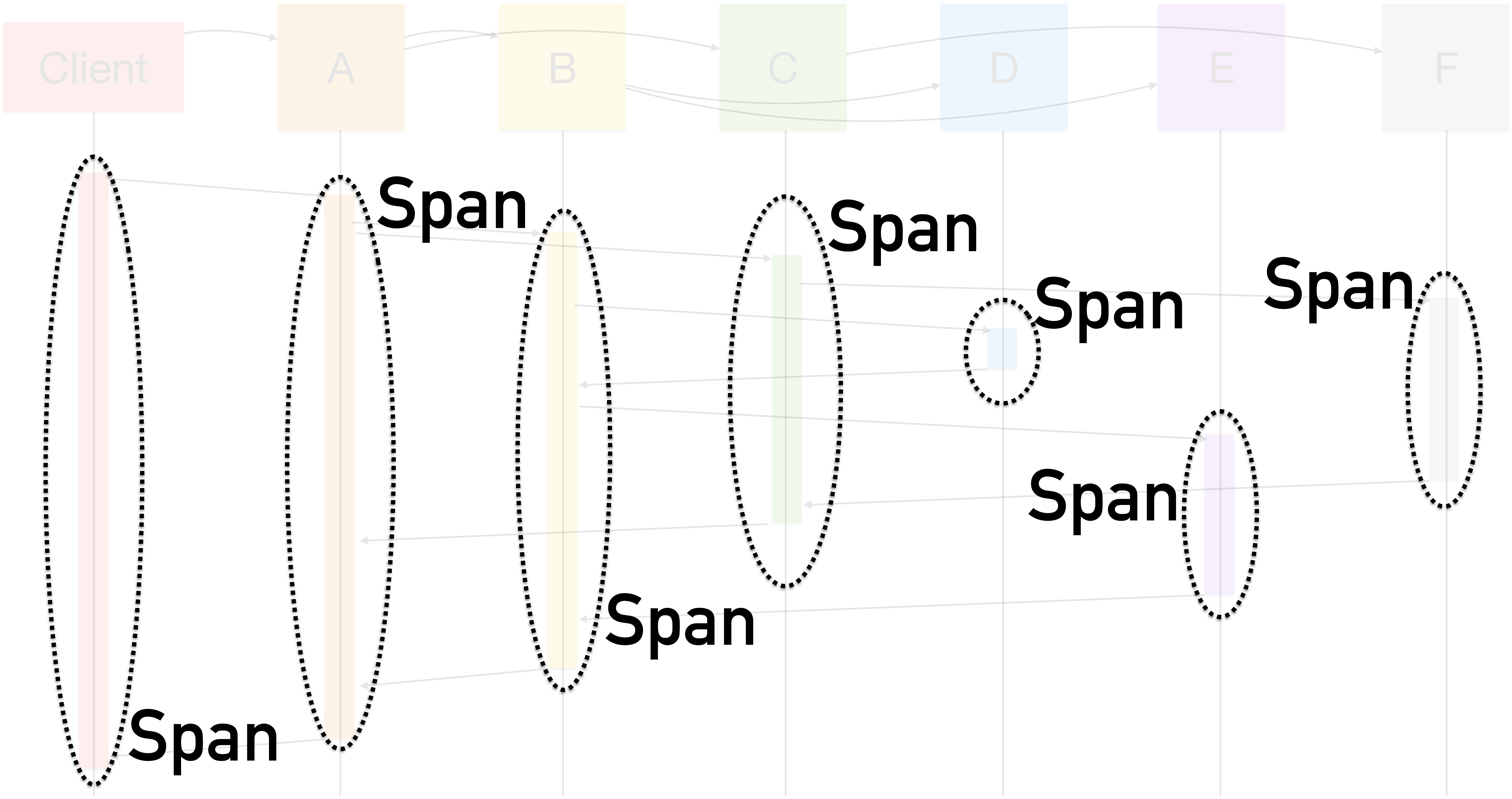
Client Send @ t=1

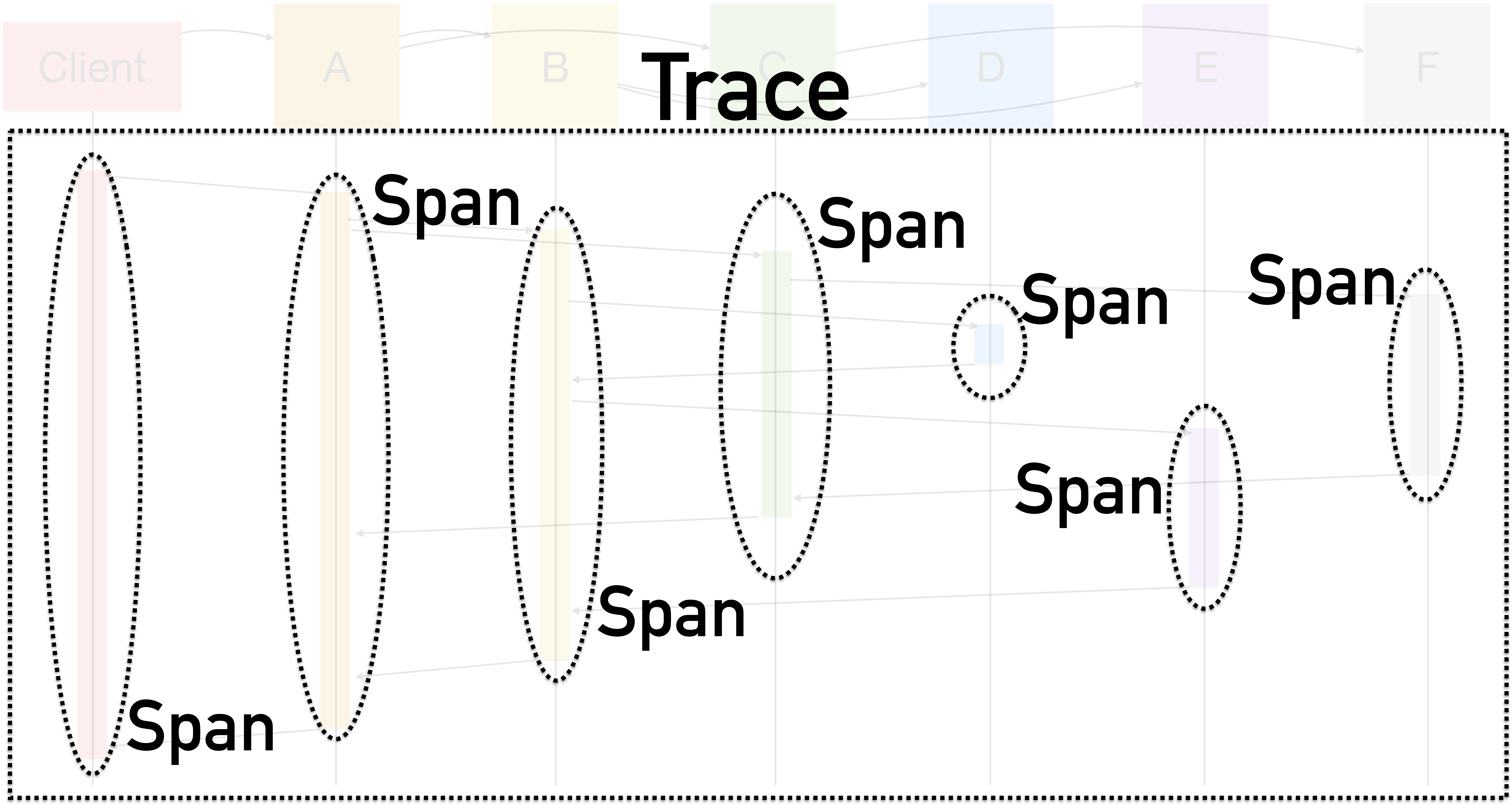
Client Receive @ t=2

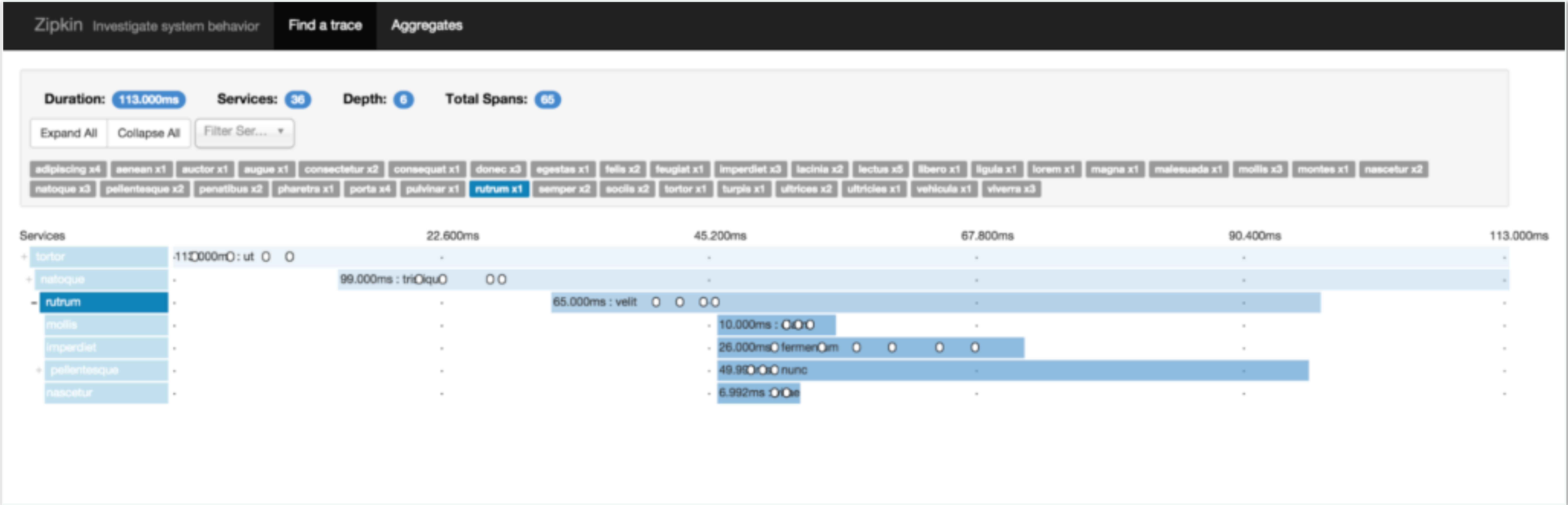
Server Send @ t=3


Client
A
B
C
D
E
F
Server Receive @ t=0
Span
Client Send @ t=1
Client Receive @ t=2
Server Send @ t=3

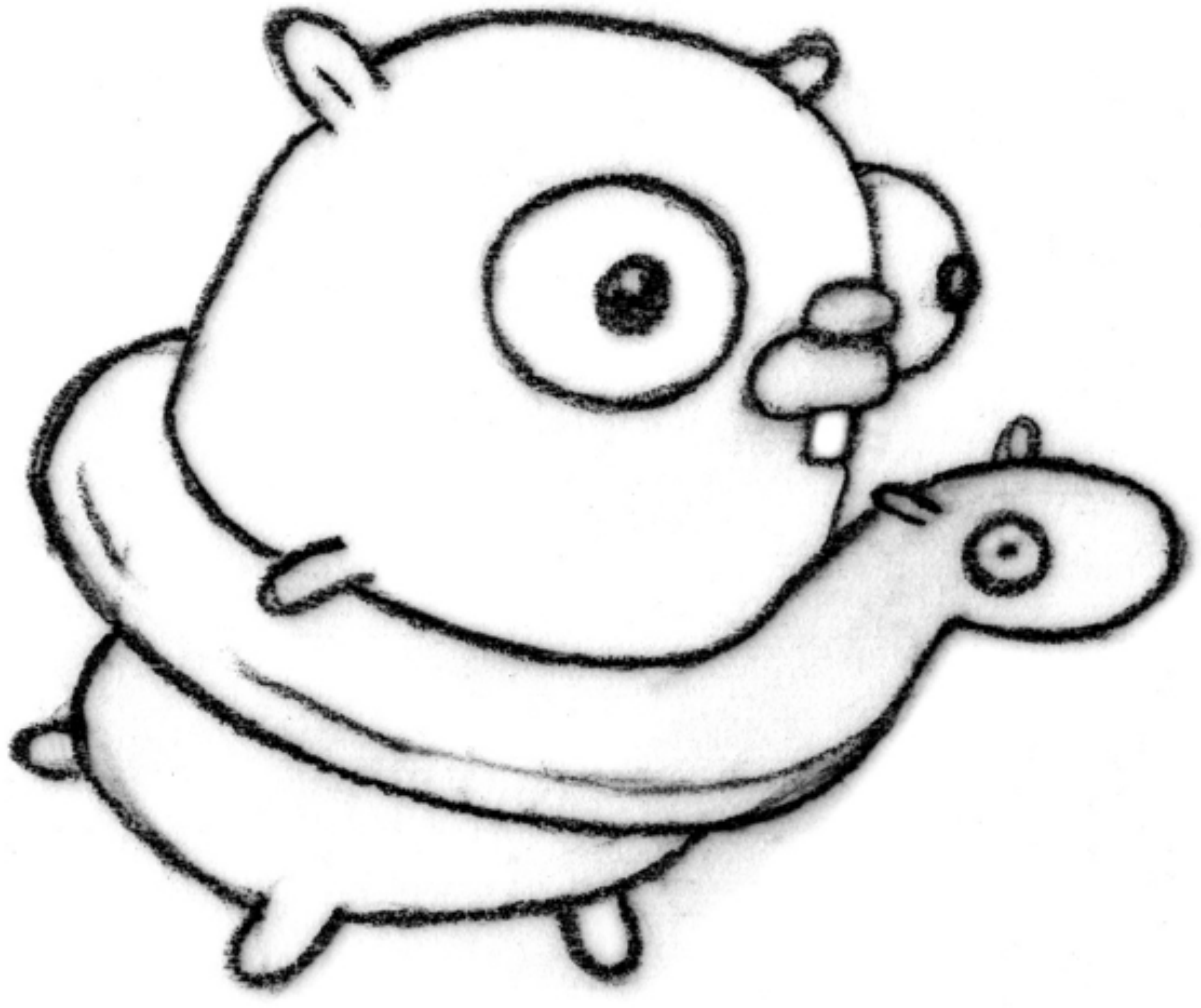

Client
A
B
C
D
E
F
Span


Client
A
B
C
D
E
F
Span
Span
Span
Span
Span
Span
Span
Span


Client
A
B
C
D
E
F
Trace
Span
Span
Span
Span
Span
Span
Span
Span

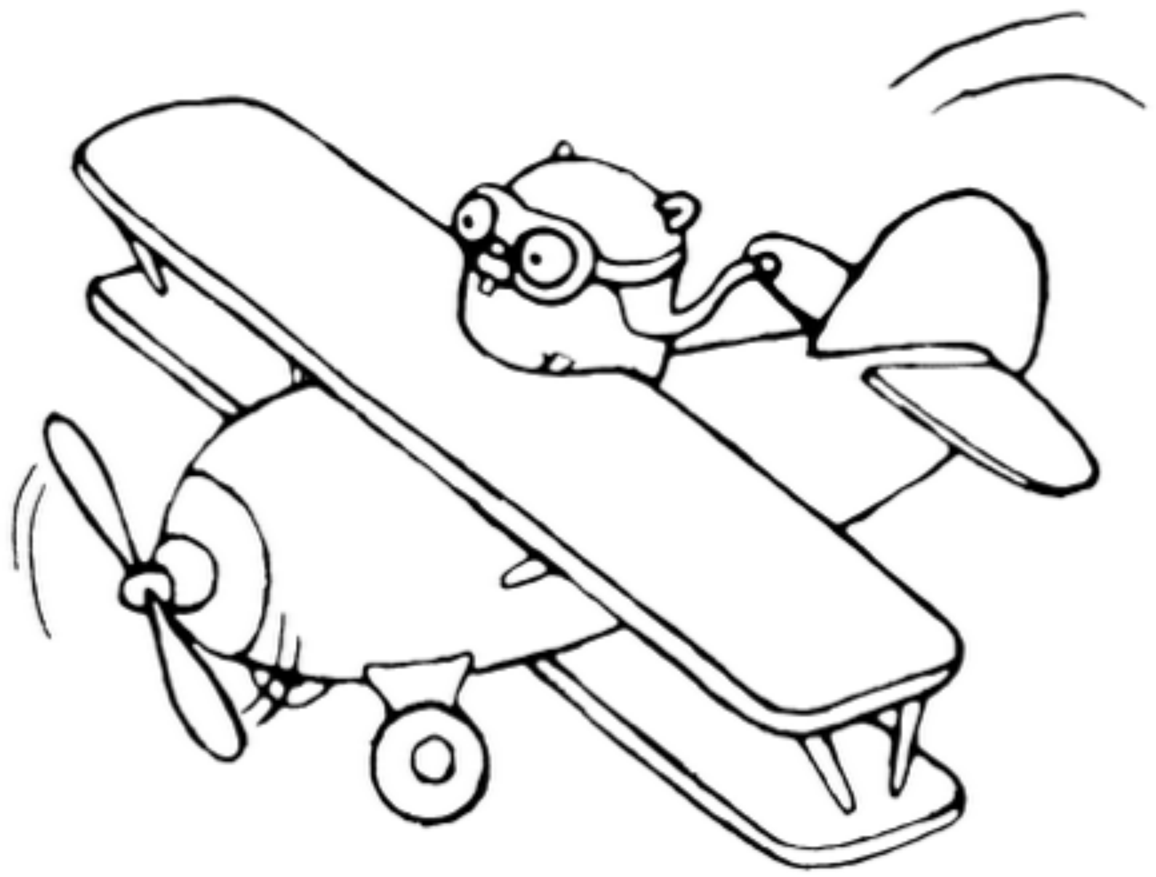


Duration: 113.000ms Services: 36 Depth: 6 Total Spans: 65



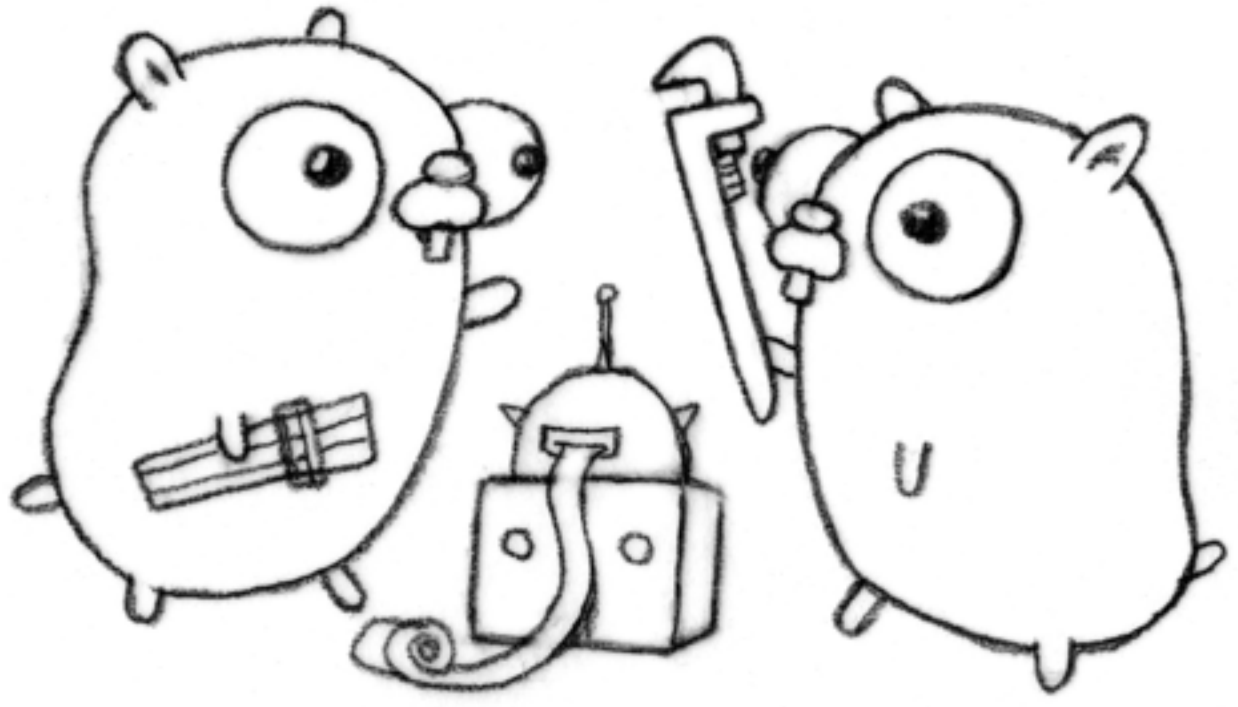
git apply

20



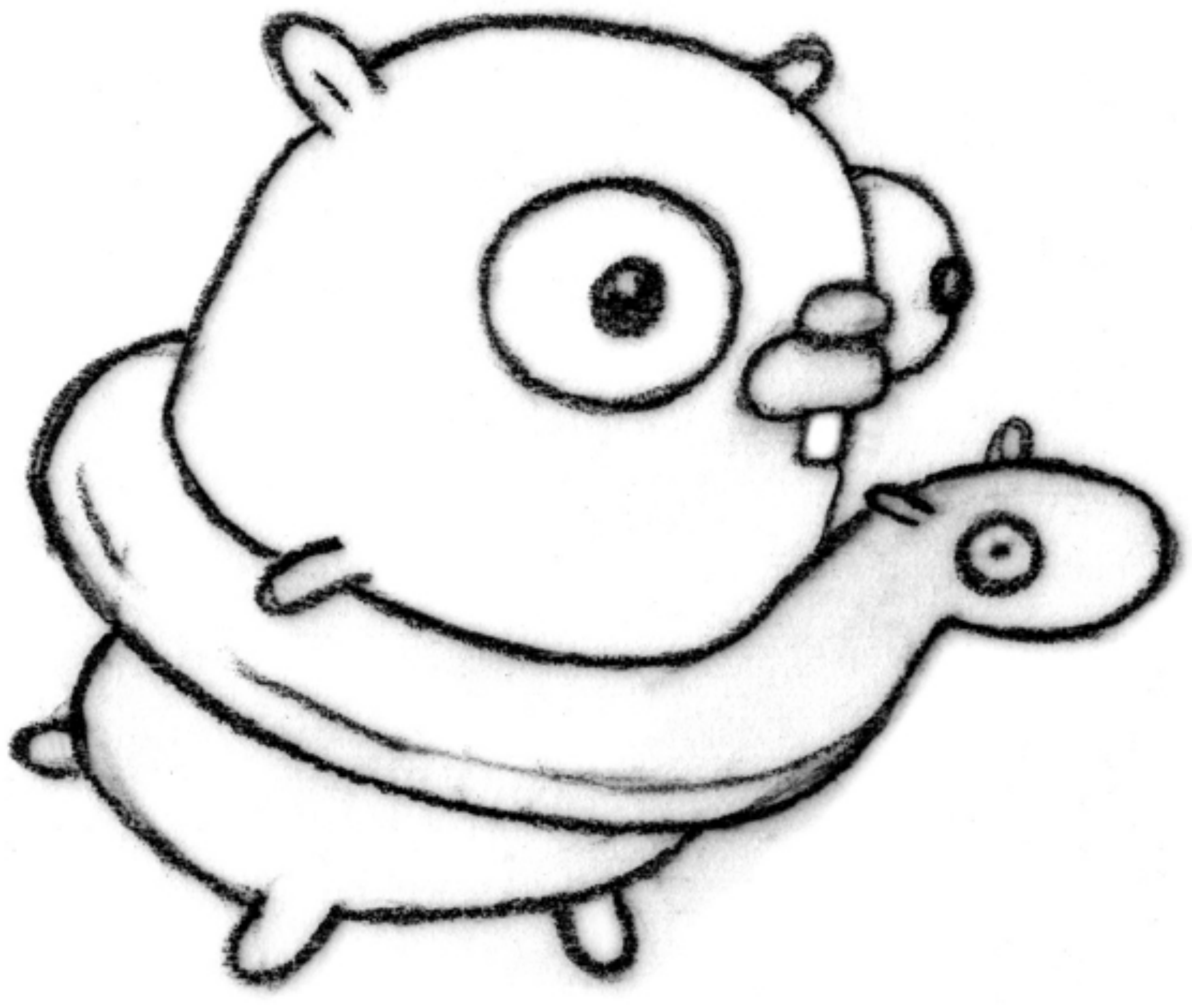
Challenge

Actually get it working :(



addsvc

Continuous integration with CircleCI



git apply

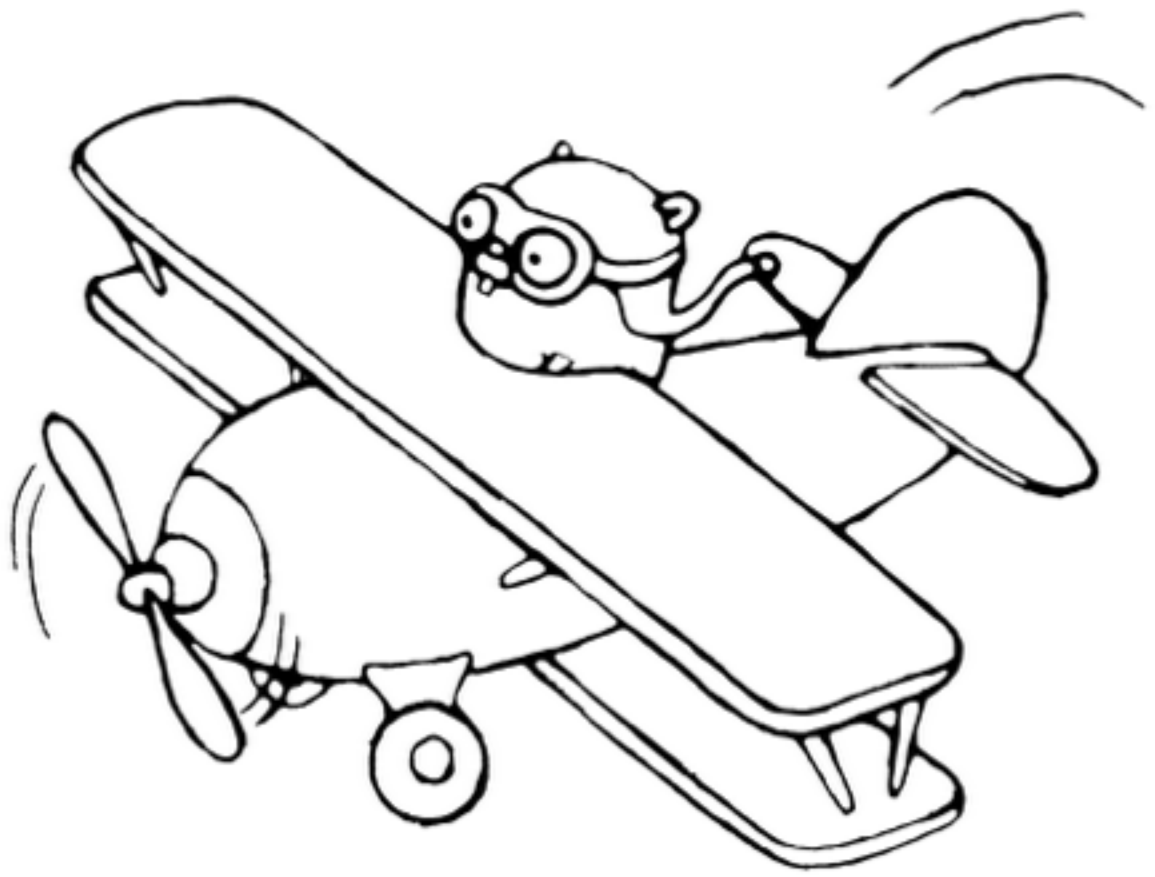
21, 22

CircleCI.com

Project settings

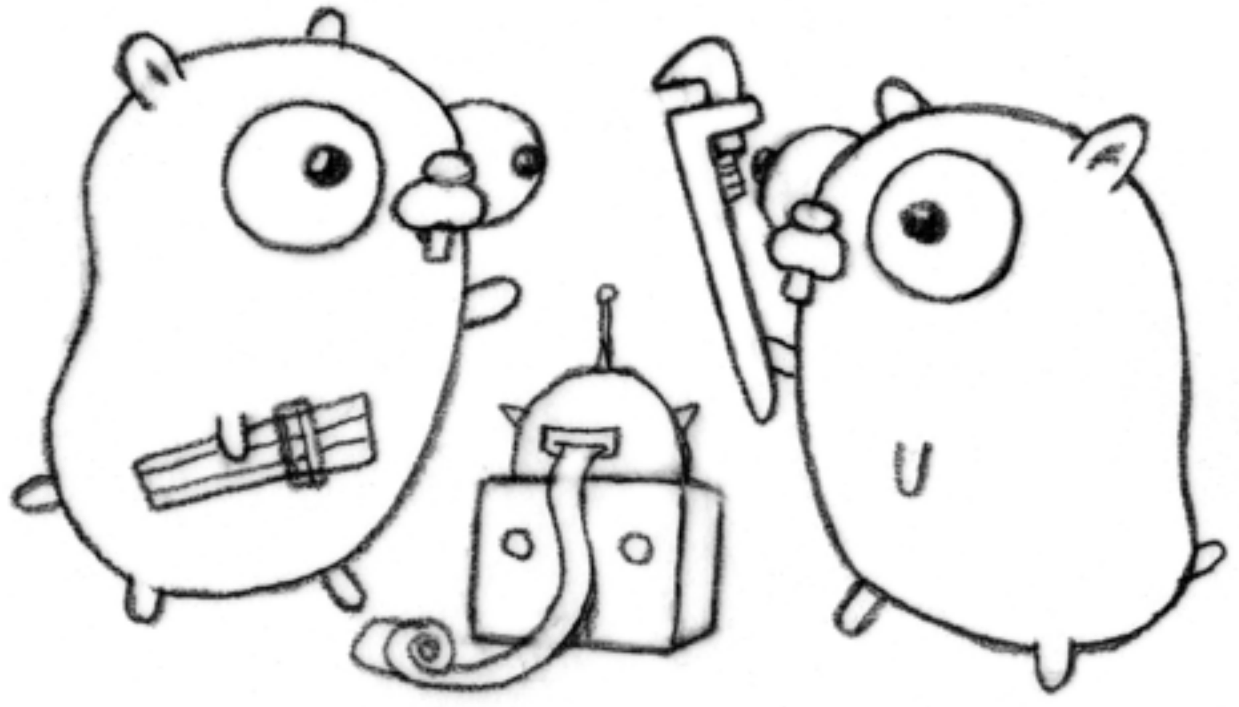
circle.yml

and Makefile and Dockerfile



Challenge

Create your own repo, and test it with CircleCI!
(*Hint: you'll need to change import paths...)*



addsvc

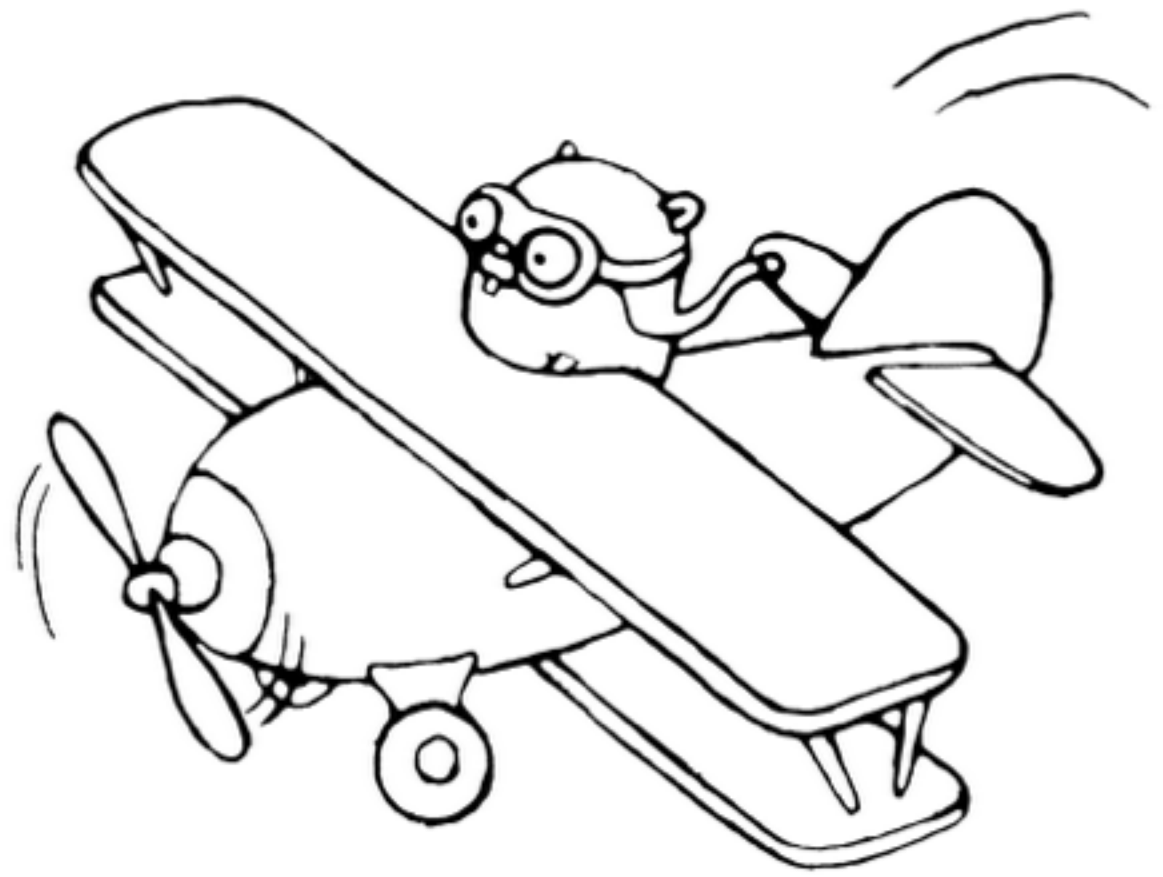
Cloud-native deployment with Kubernetes

Kubernetes architecture

And minikube local cluster

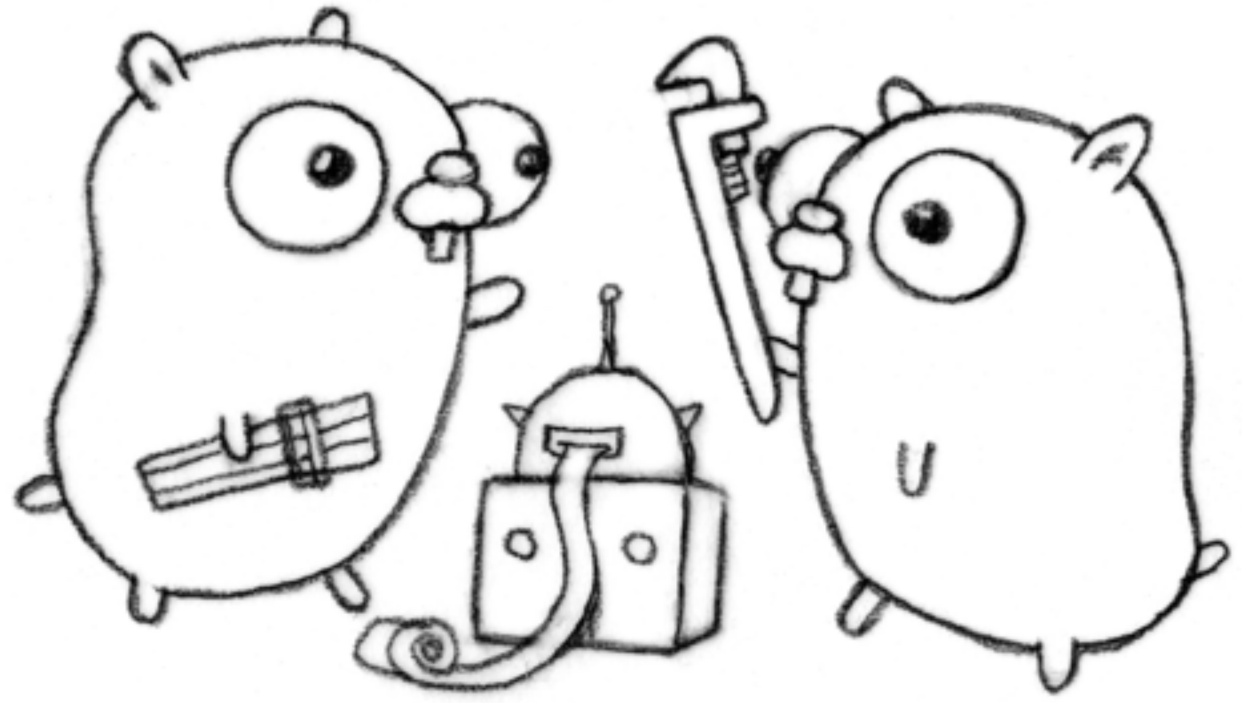
addsvc-*.yaml

Resource definition files



Challenge

Deploy a Kubernetes cluster, and your own addsvc



package sd

Service discovery and client-side load balancing

package log

Structured logging

Thanks! Hooray!

6 October 2016 · GOTO Copenhagen · @peterbourgon