Microservices with Go kit

October/November 2016 · Groupon

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?

Who are you?

- Team & team's responsibilities
- Personal responsibilities
- Technical background: languages, past projects
- Why microservices?

The microservices landscape

Toward a shared context

Size

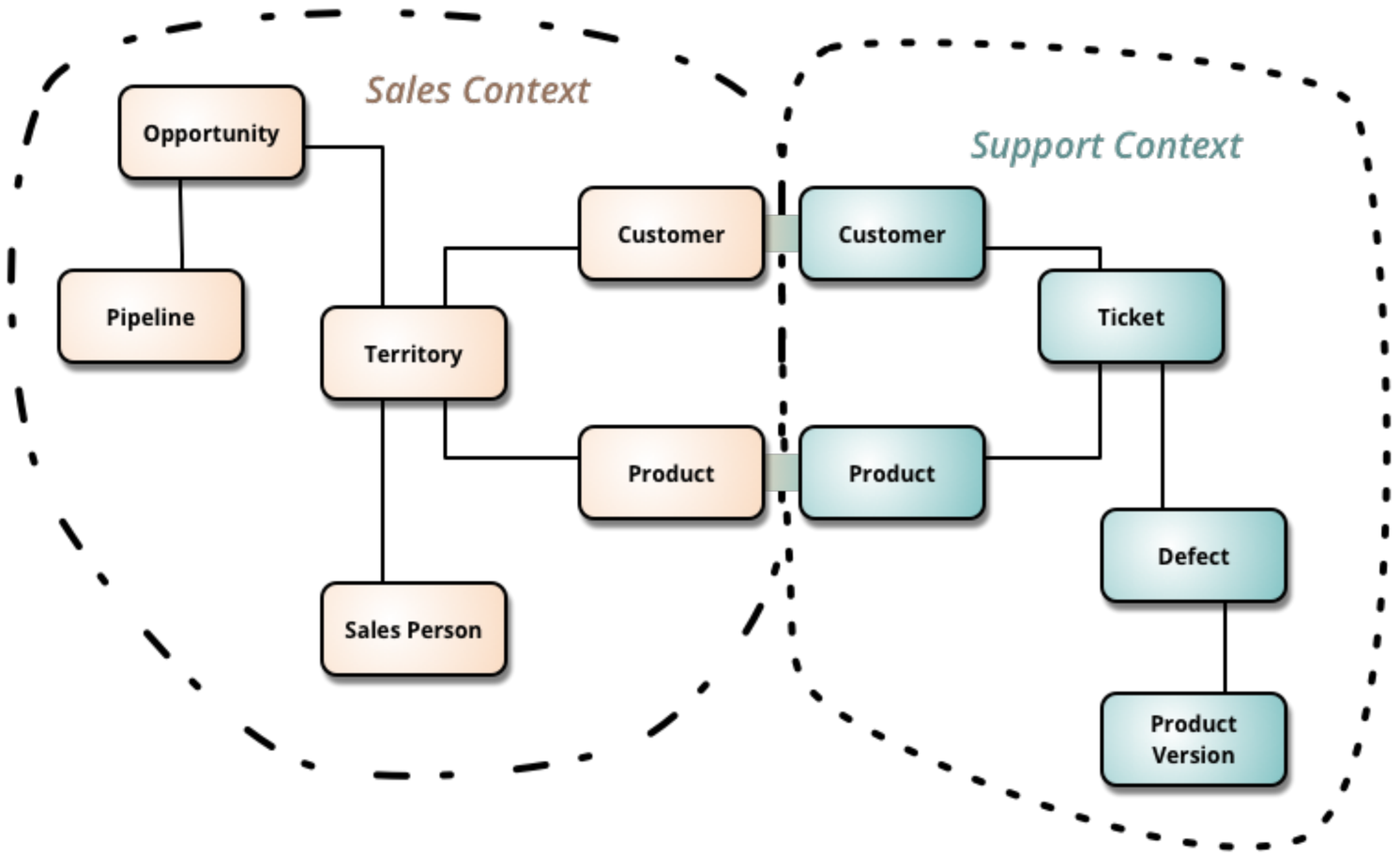
A **single programmer** can design, implement, deploy and maintain a microservice.
—Fred George

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*

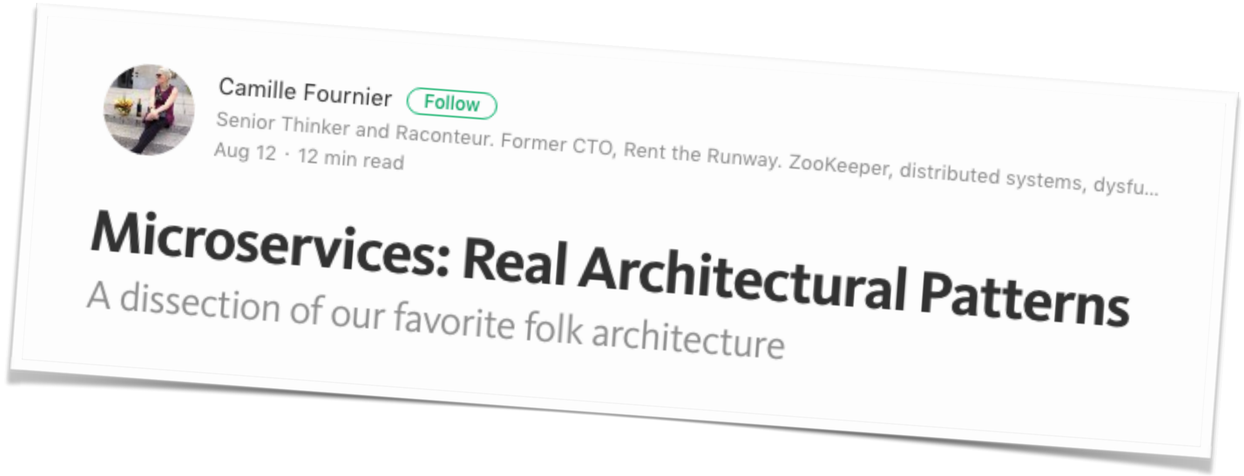
Addressable through a **service discovery** system.
—*Chris Richardson*

Architecture

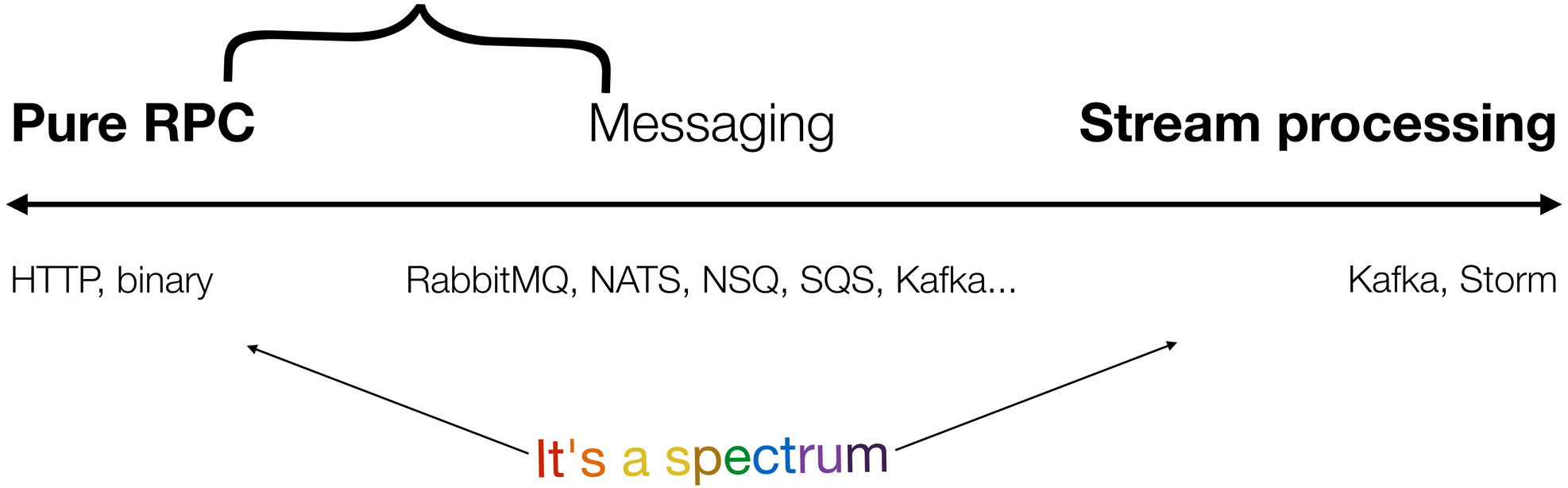
- CRUD-oriented
- Typically RPC, often HTTP
- Request processing
- Monolith → microservices
- Ruby on Rails; Tomcat/Jetty, Spring Boot; Akka, Play

Did you mean...

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



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

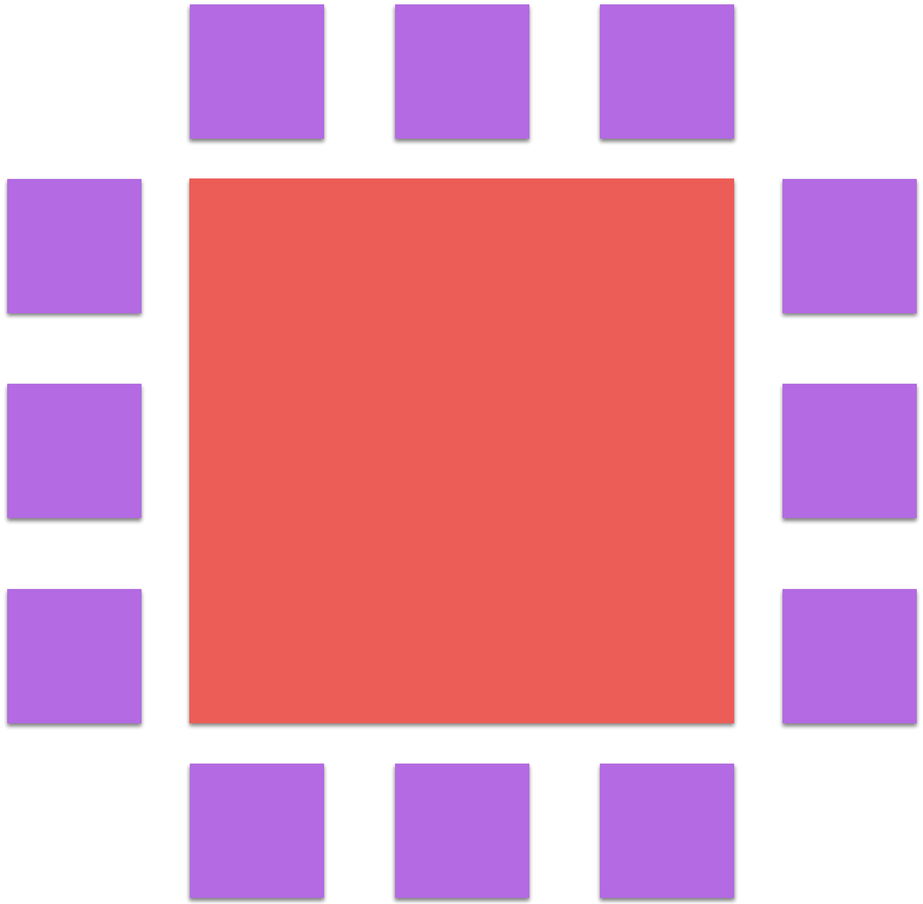
Problems caused

- 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...

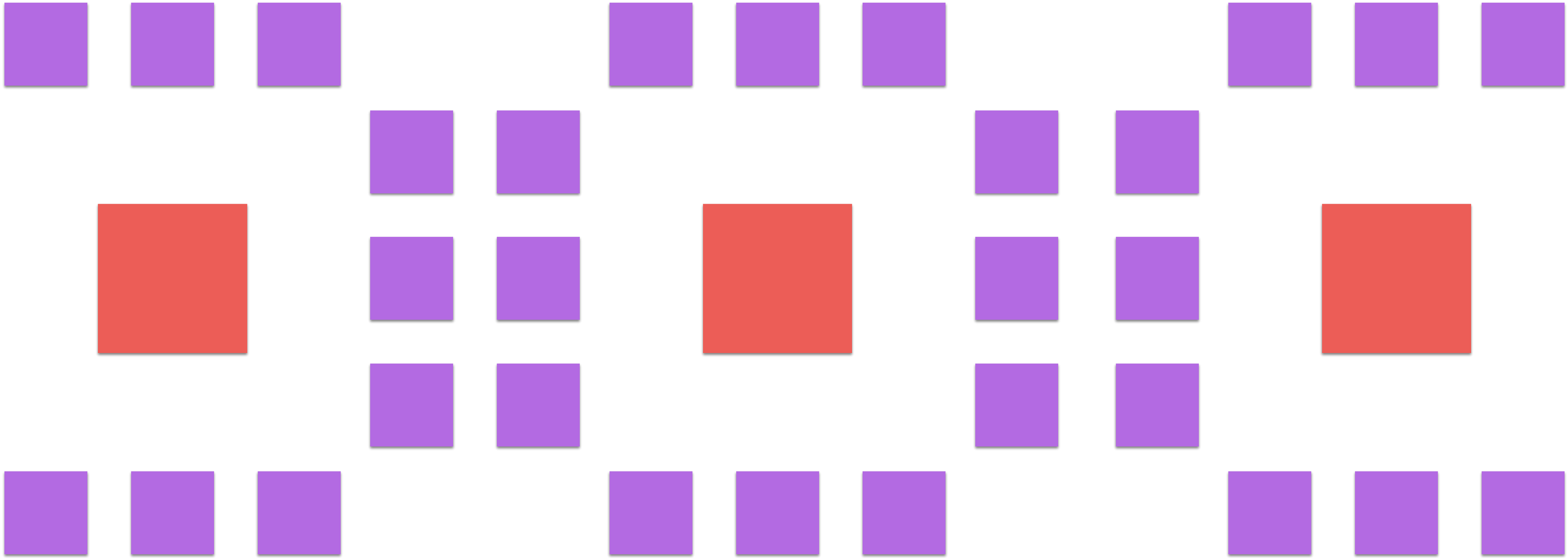
Problems caused

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

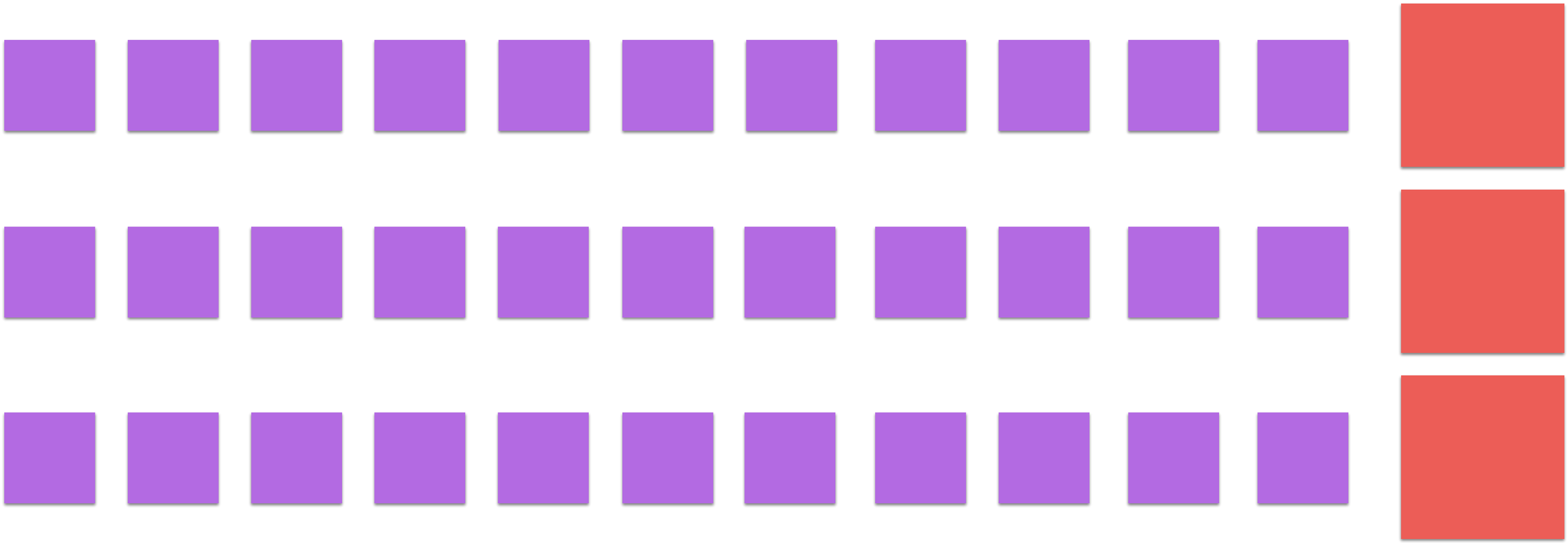
Monolith



Microservices



Microservices



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

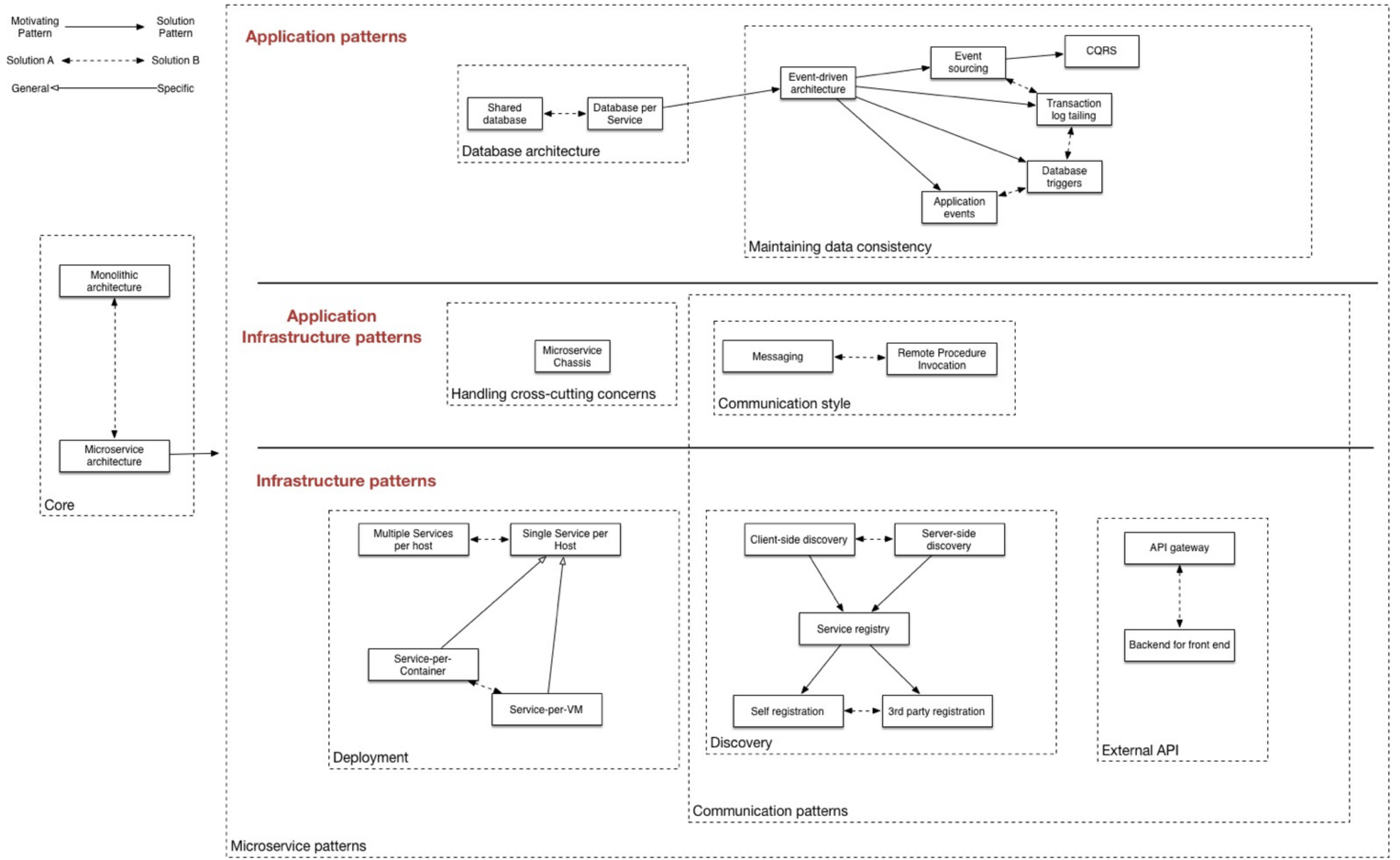
I ❤ microservices

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

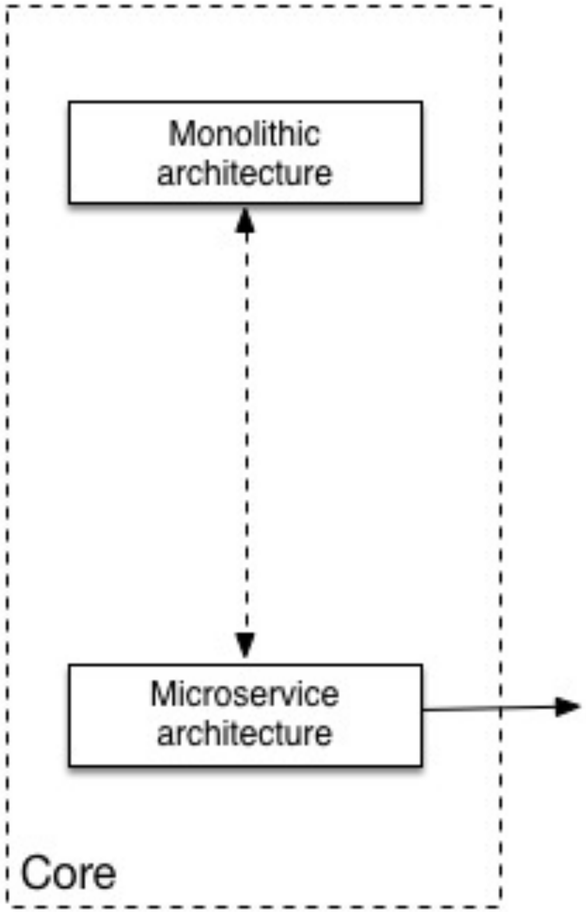
What's a pattern?

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

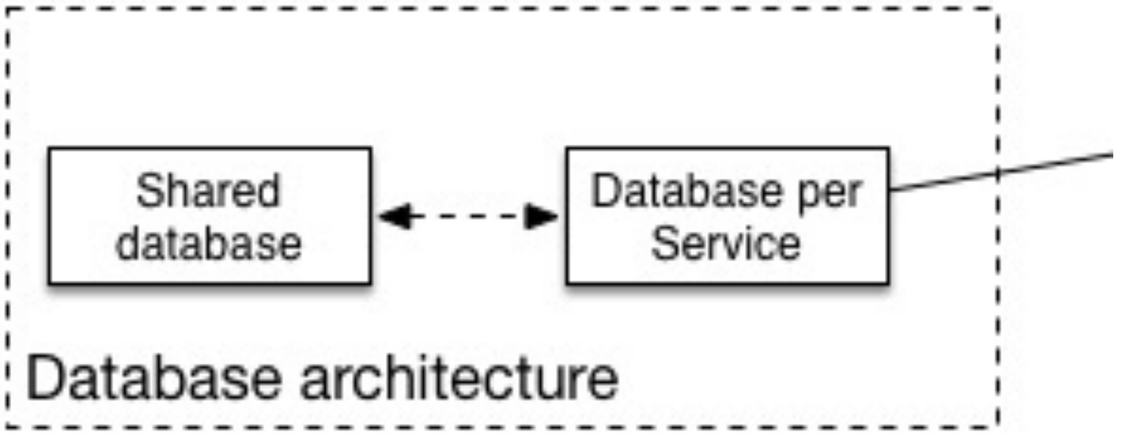
slideshare.net/chris.e.richardson/microservices-pattern-language-microxchg-microxchg2016



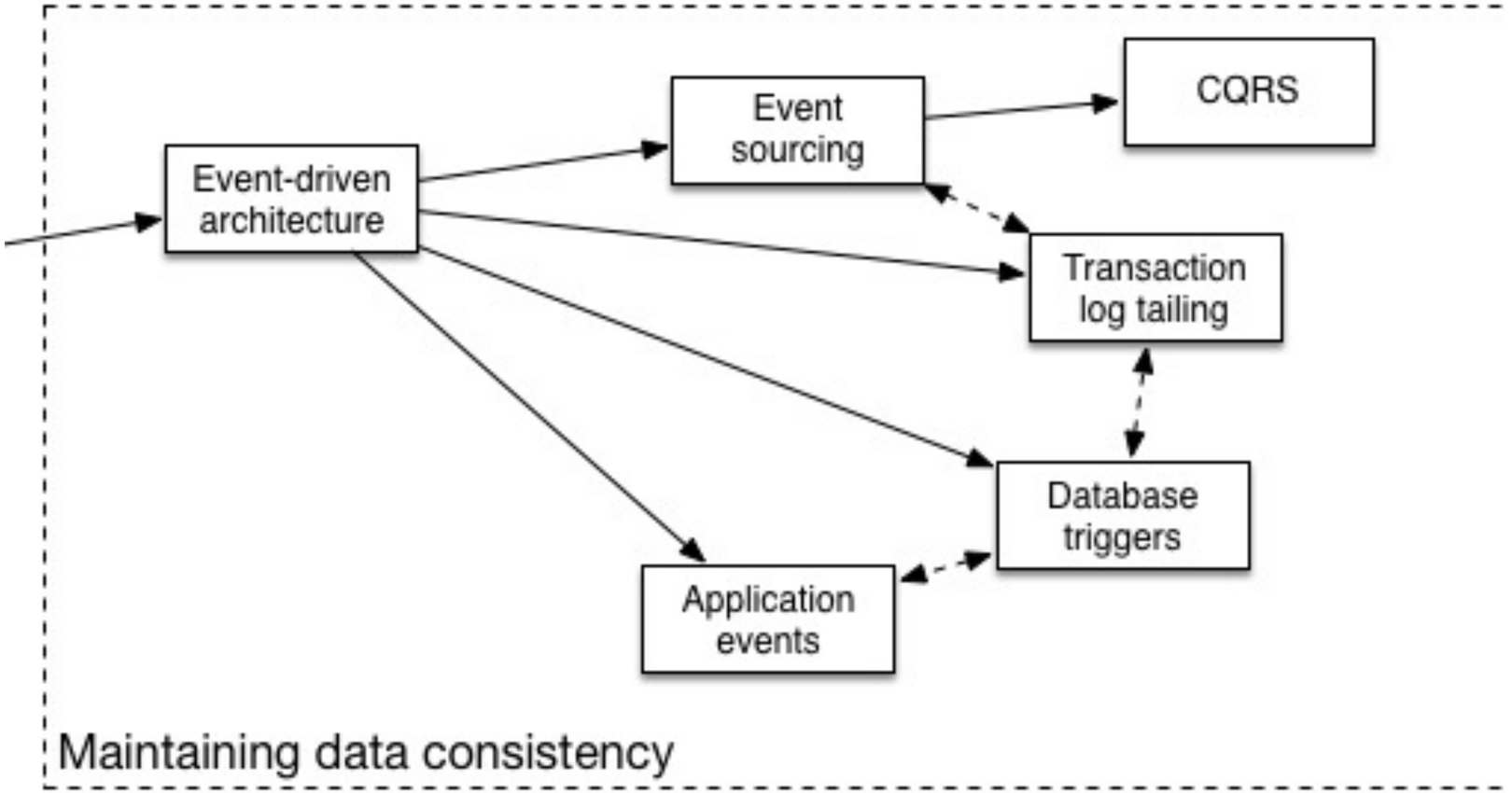
microservices.io/patterns/index.html



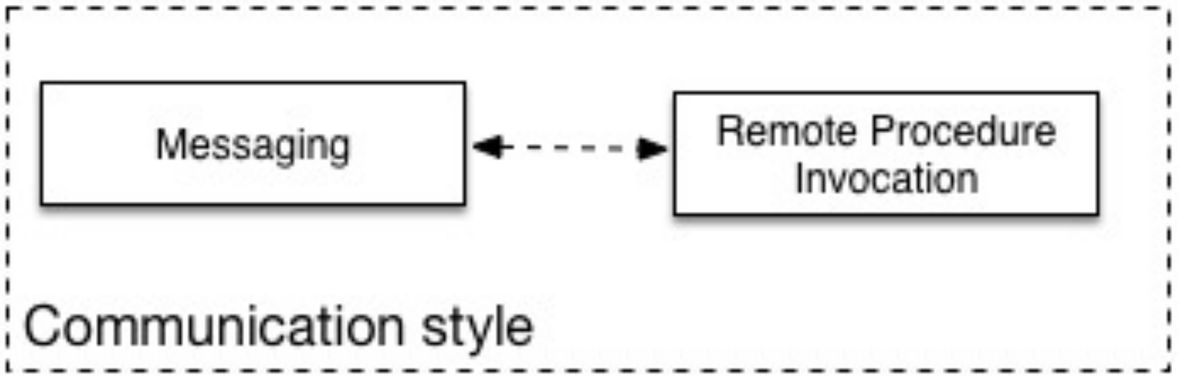
microservices.io/patterns/index.html



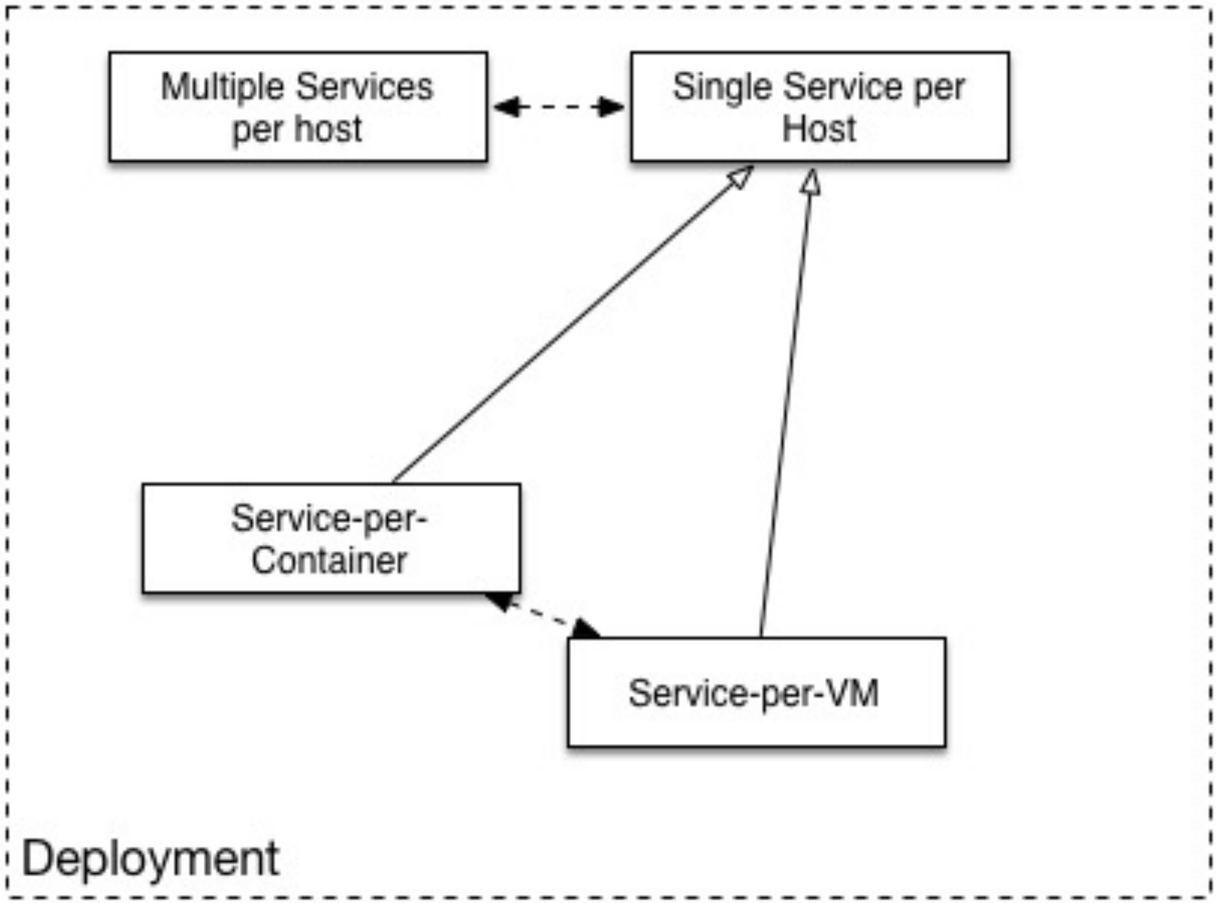
microservices.io/patterns/index.html



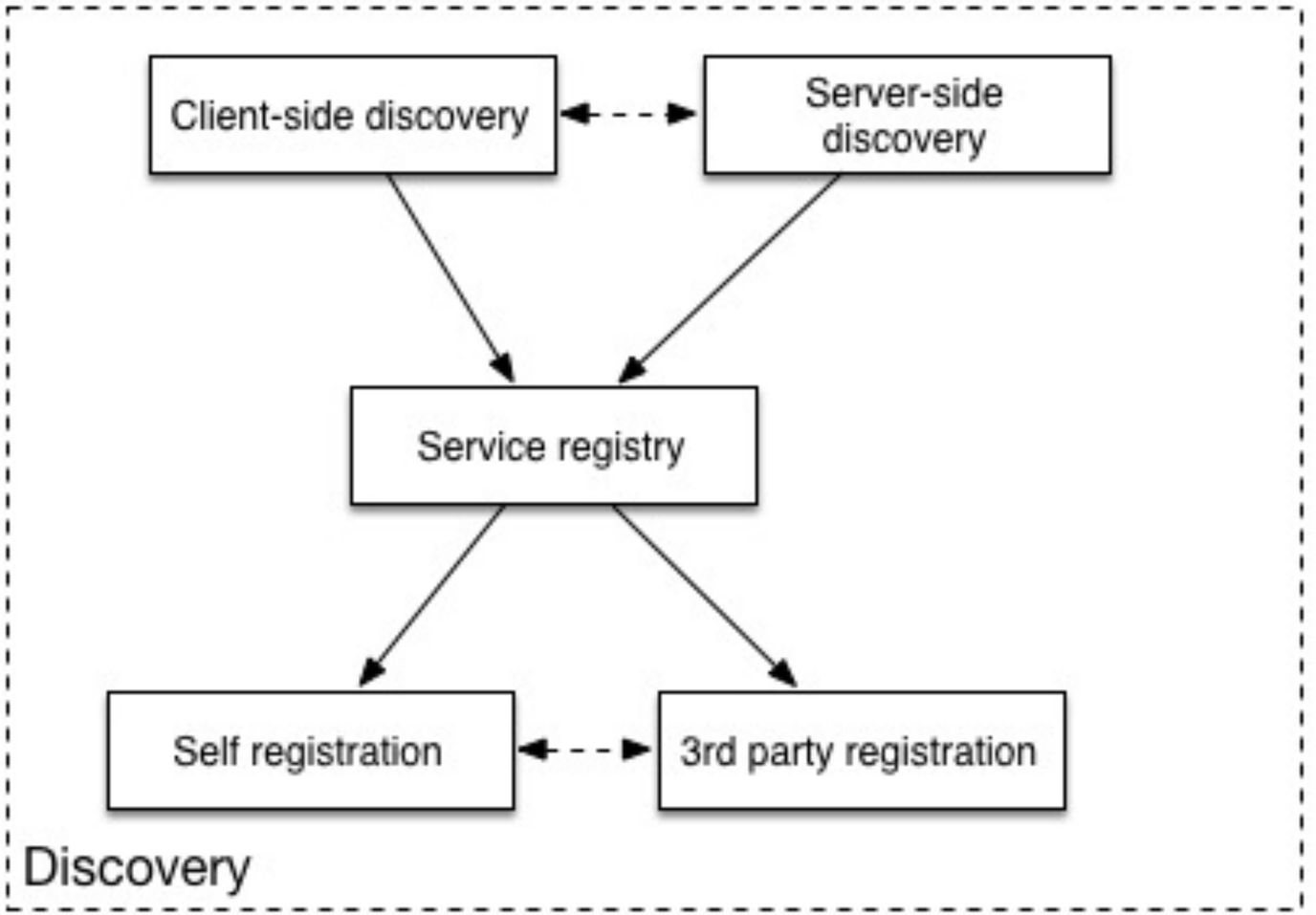
microservices.io/patterns/index.html



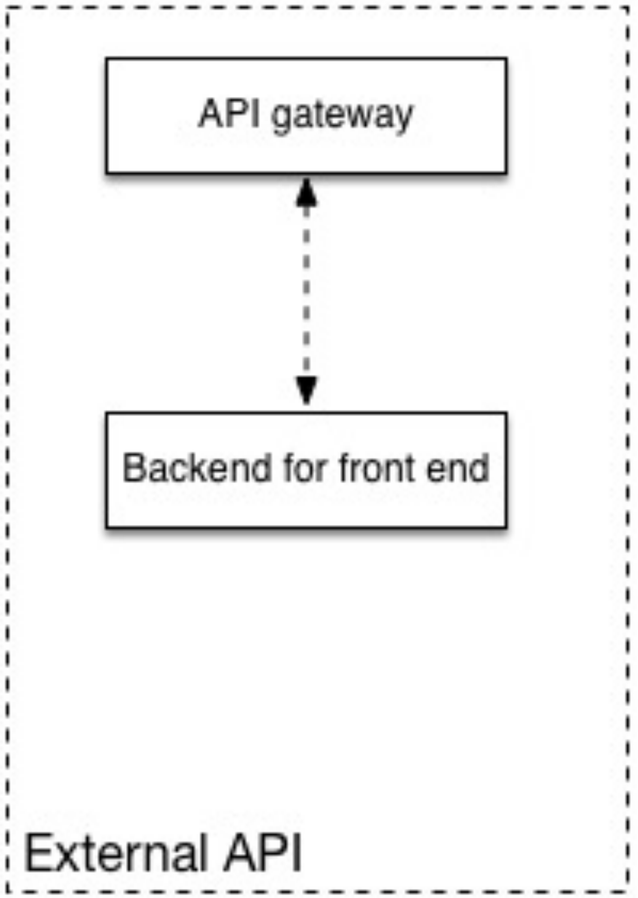
microservices.io/patterns/index.html



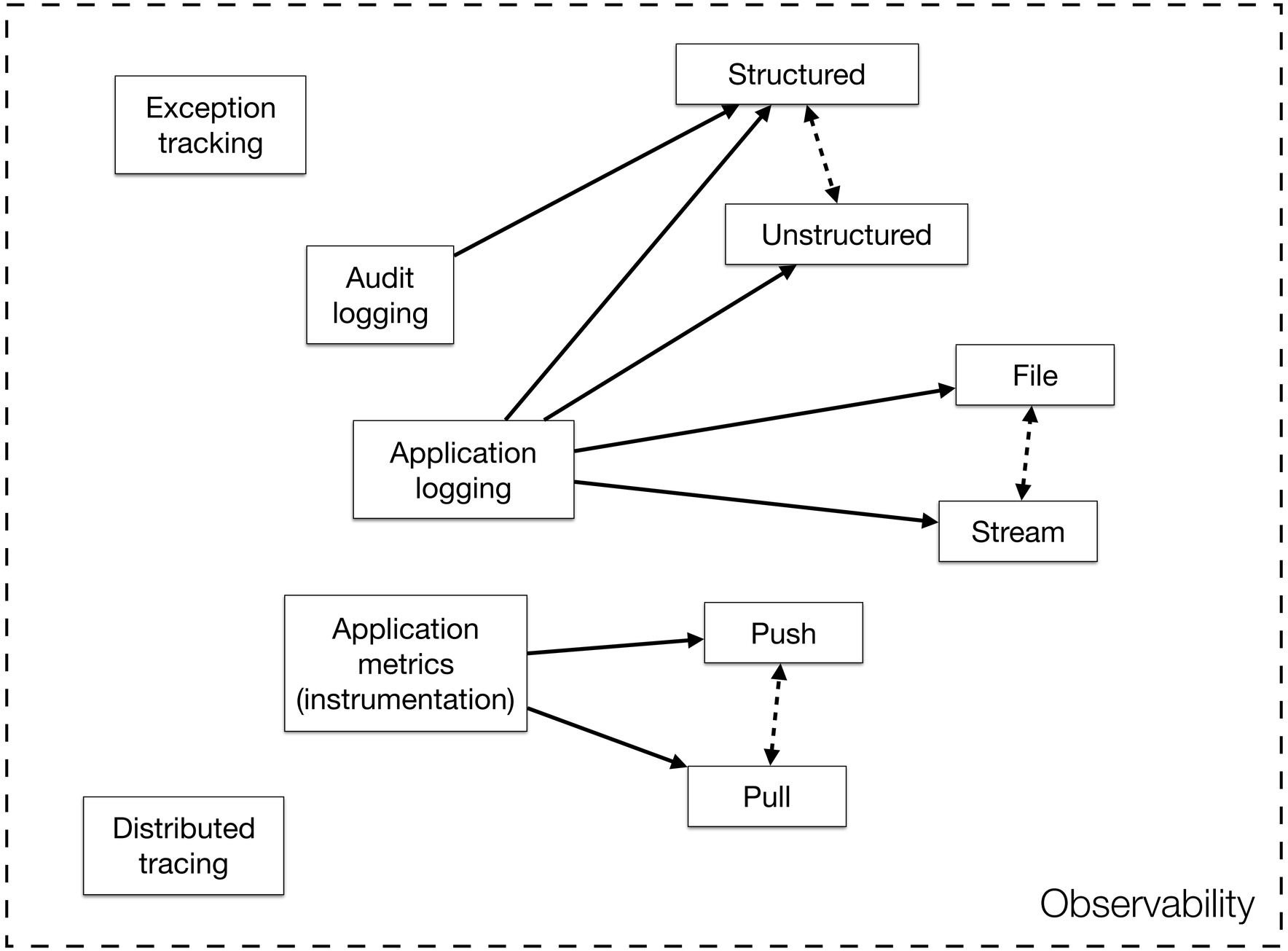
microservices.io/patterns/index.html



microservices.io/patterns/index.html



microservices.io/patterns/index.html


Exception tracking
Structured
Unstructured
Audit logging
Application logging
File
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

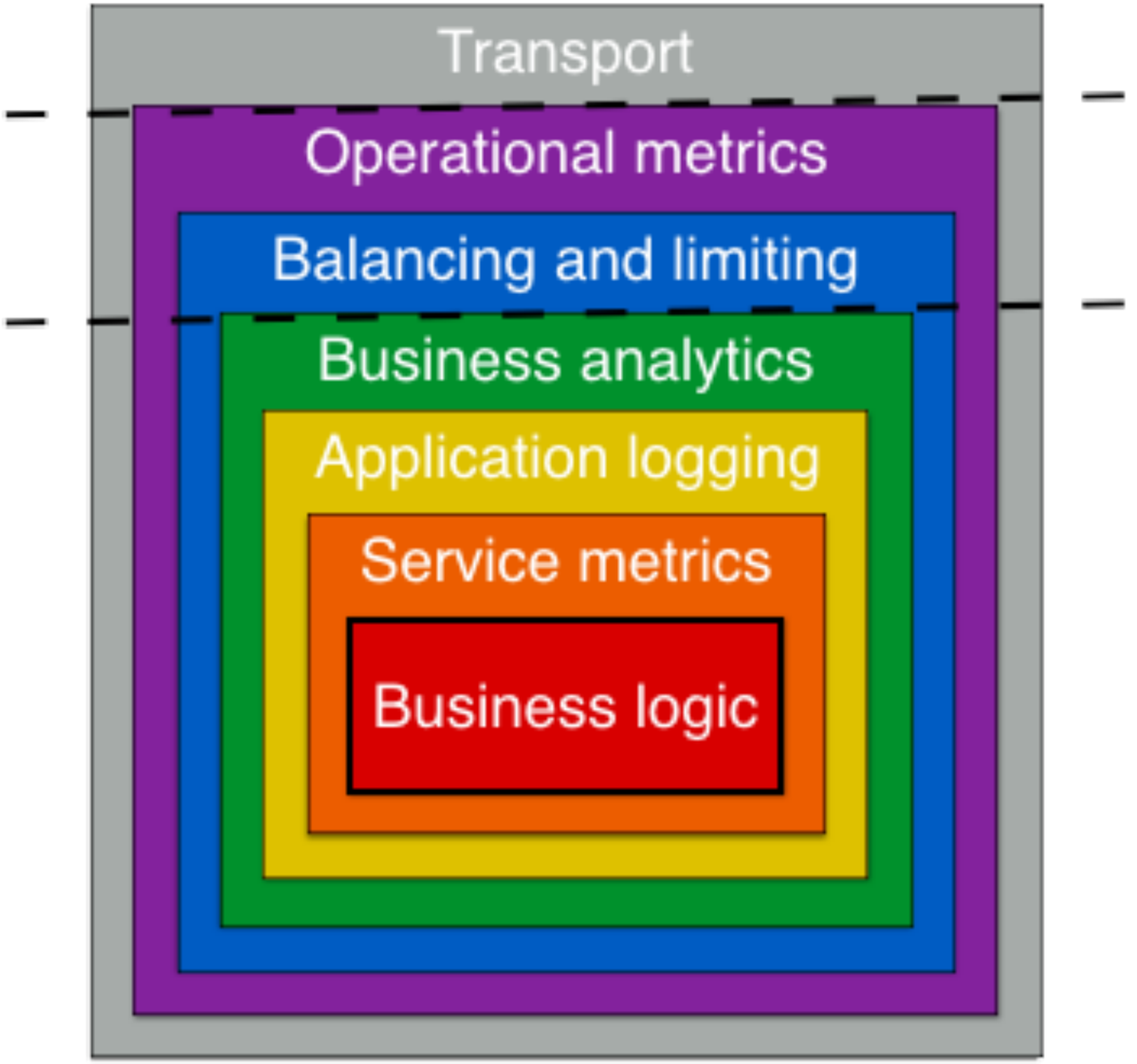
Metrics
Logging
Distributed tracing

- Transport
- Rate limiting
- Circuit breaking

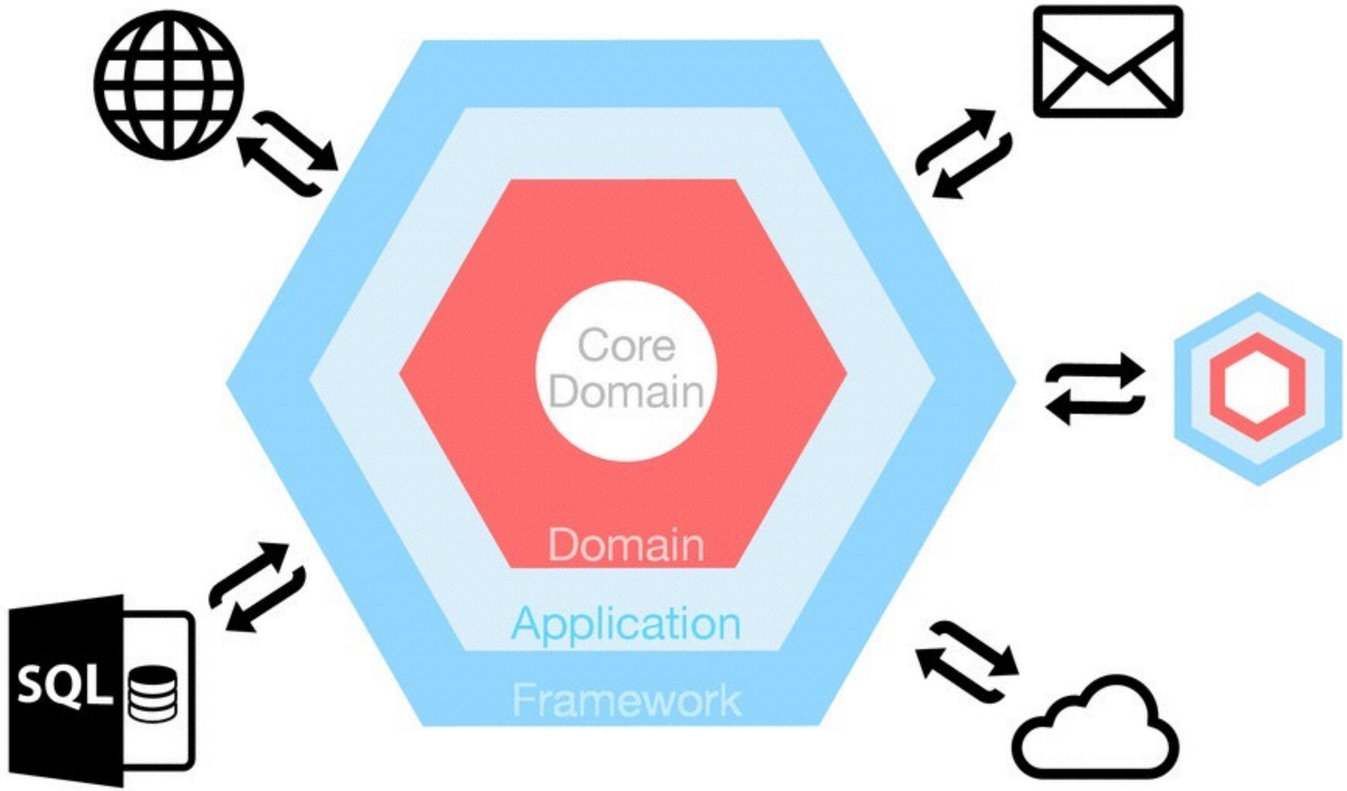
Business logic

- Service registration
- Service discovery
- Load balancing

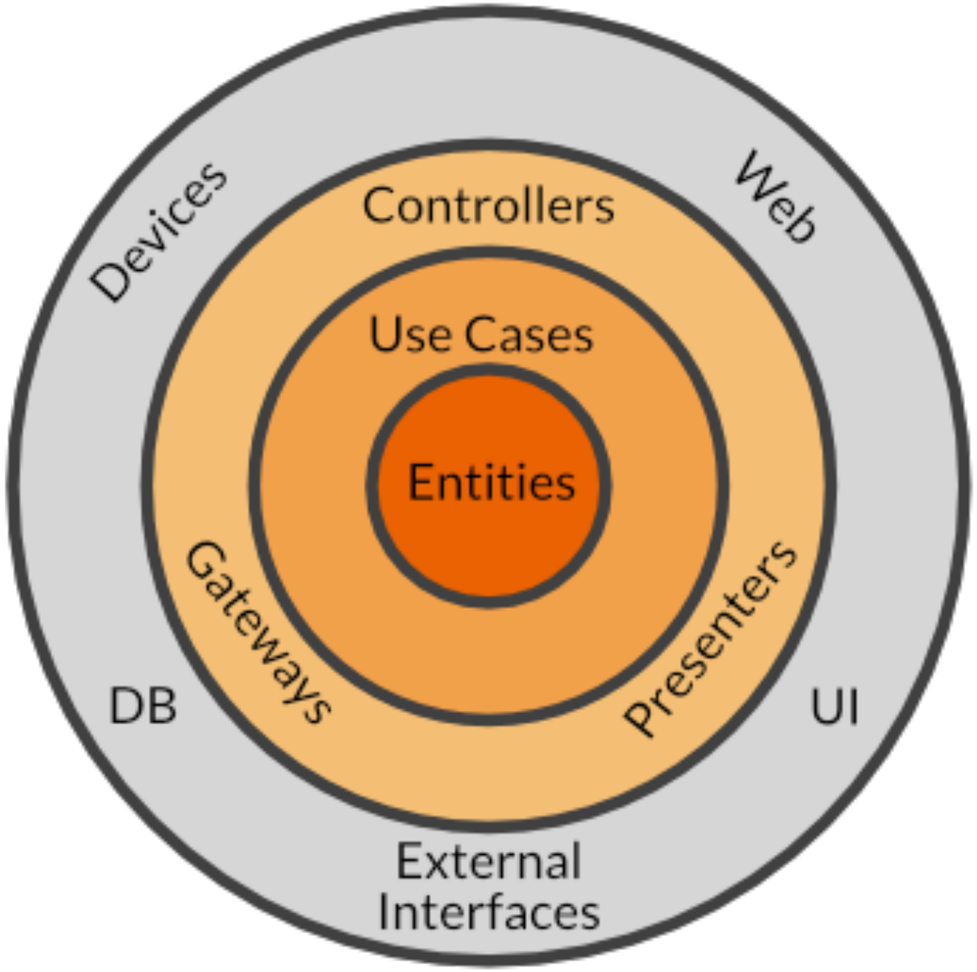
- Metrics
- Logging
- Distributed tracing


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

The Hexagon



http://fideloper.com/hexagonal-architecture



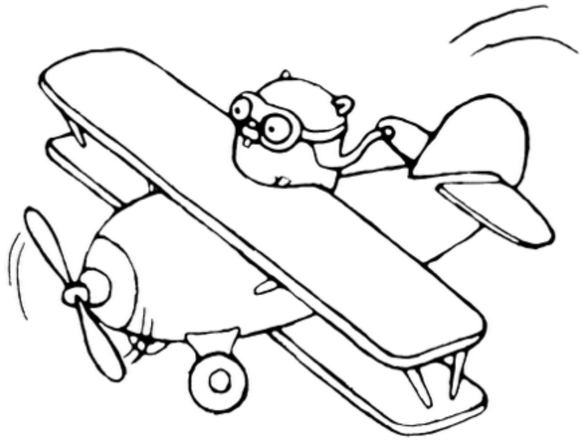
The central rule of The Clean Architecture is the Dependency Rule, which says

Source code dependencies can only point inwards.

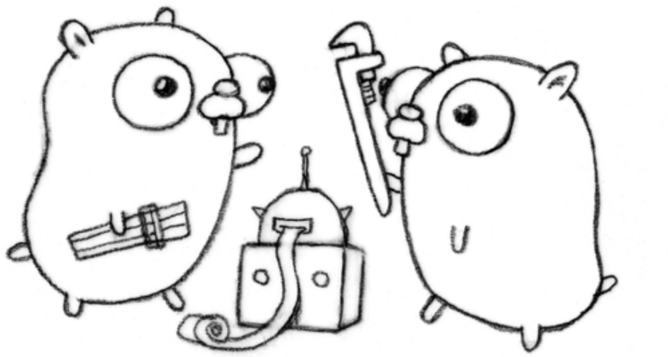
https://appliedgo.net/di/

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
- Transactional RDBMS
- 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

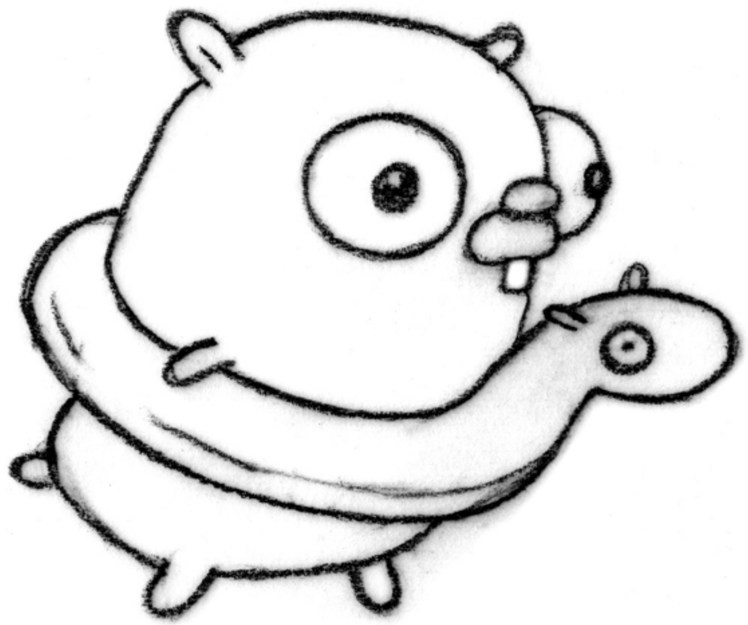
My little zen rock garden

Go refresher

Via go tool present

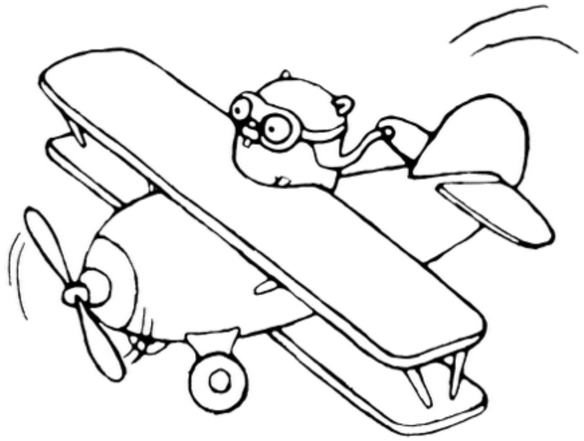
addsvc

Basic implementation



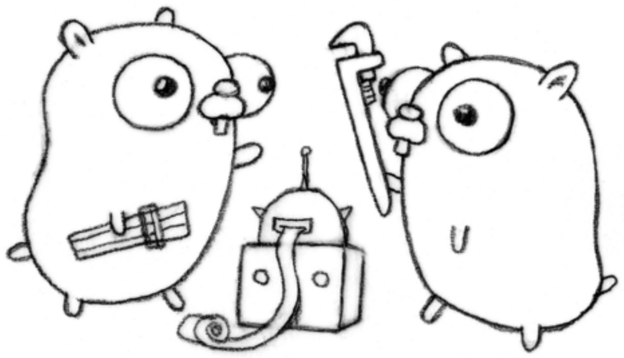
git apply

01, 02, 03



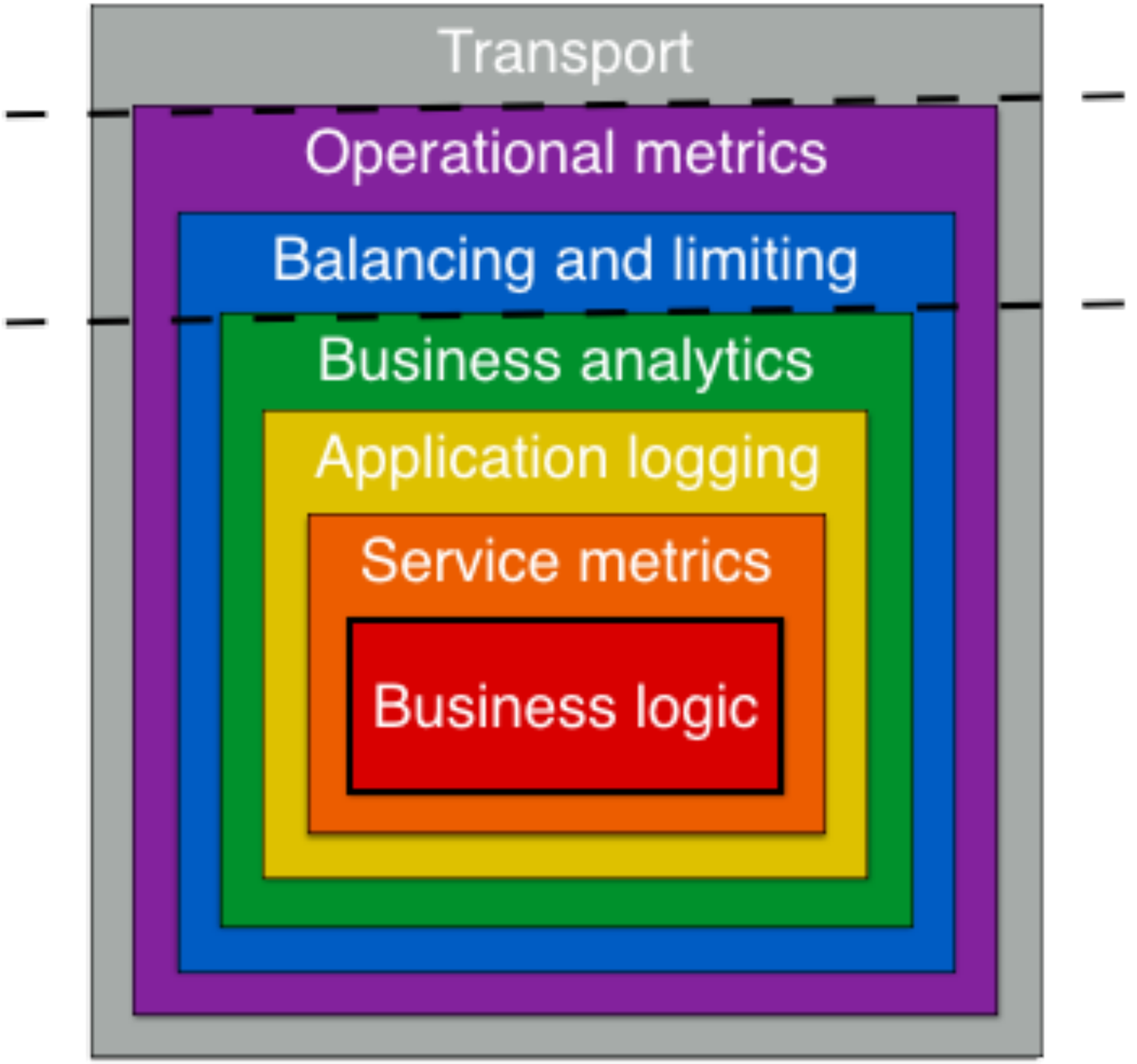
Challenge

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
}
```

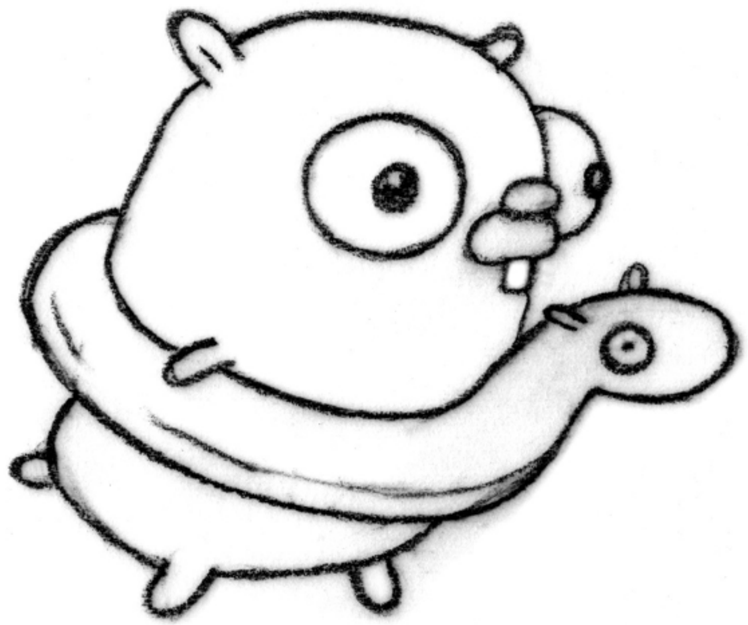
```
func log(...) {
    // proceed as normal
    log.Printf("...")
}
```

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

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

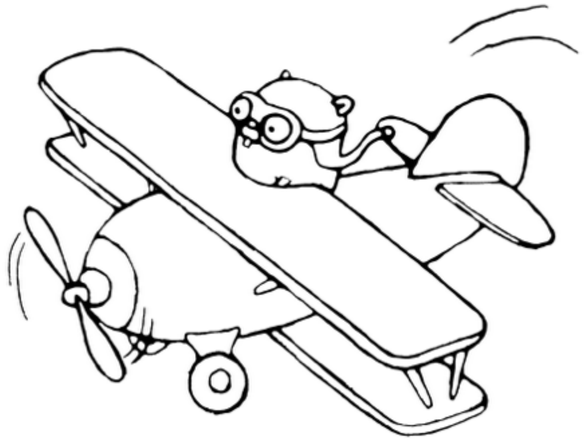
Endpoint

- 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 :(



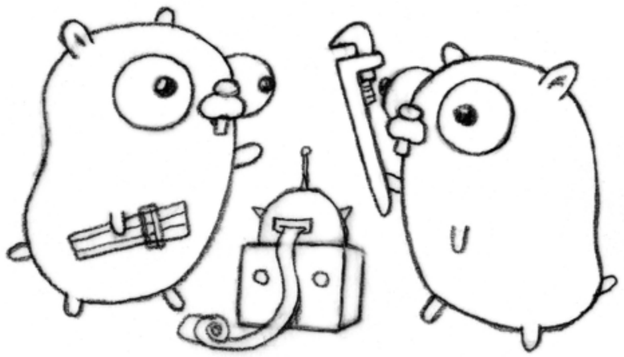
git apply

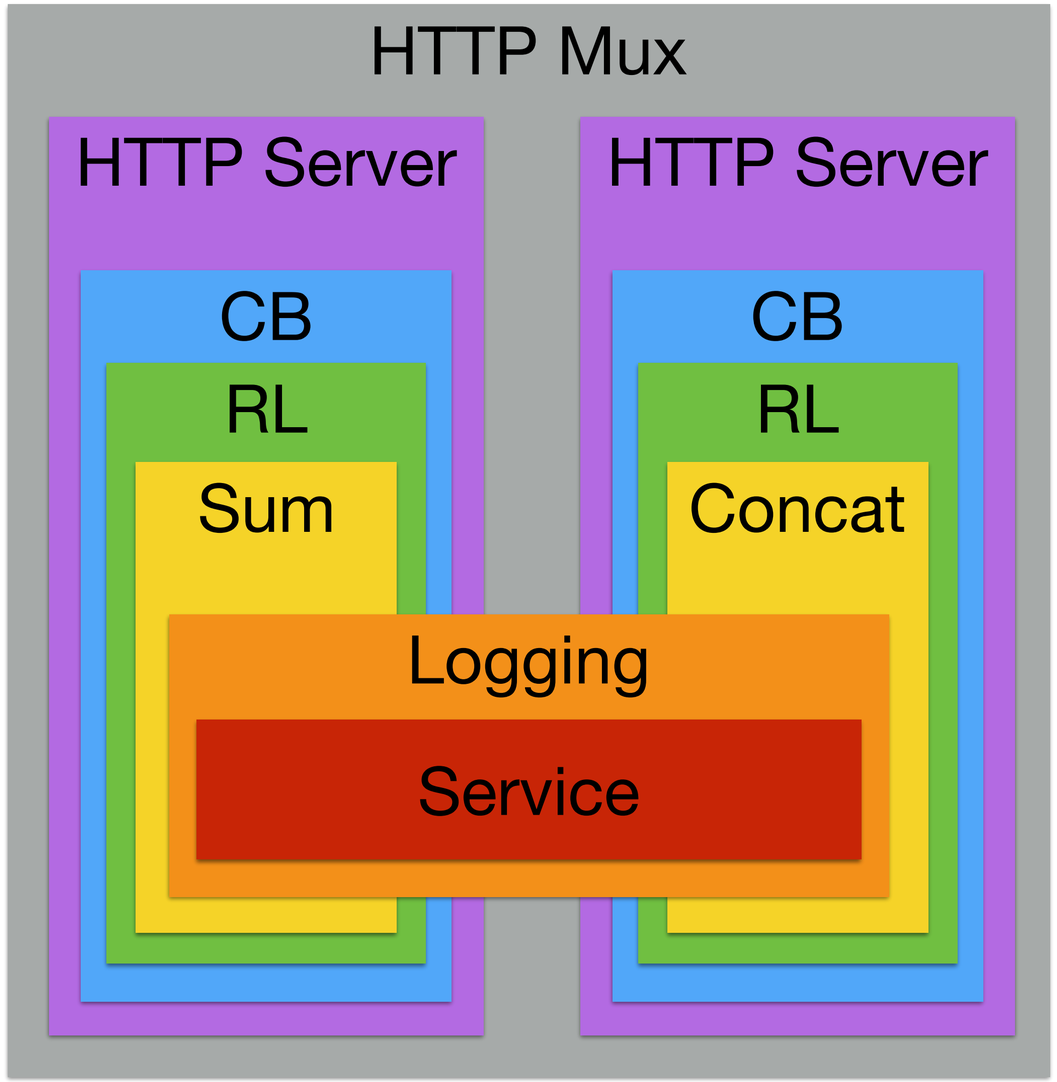
04, 05

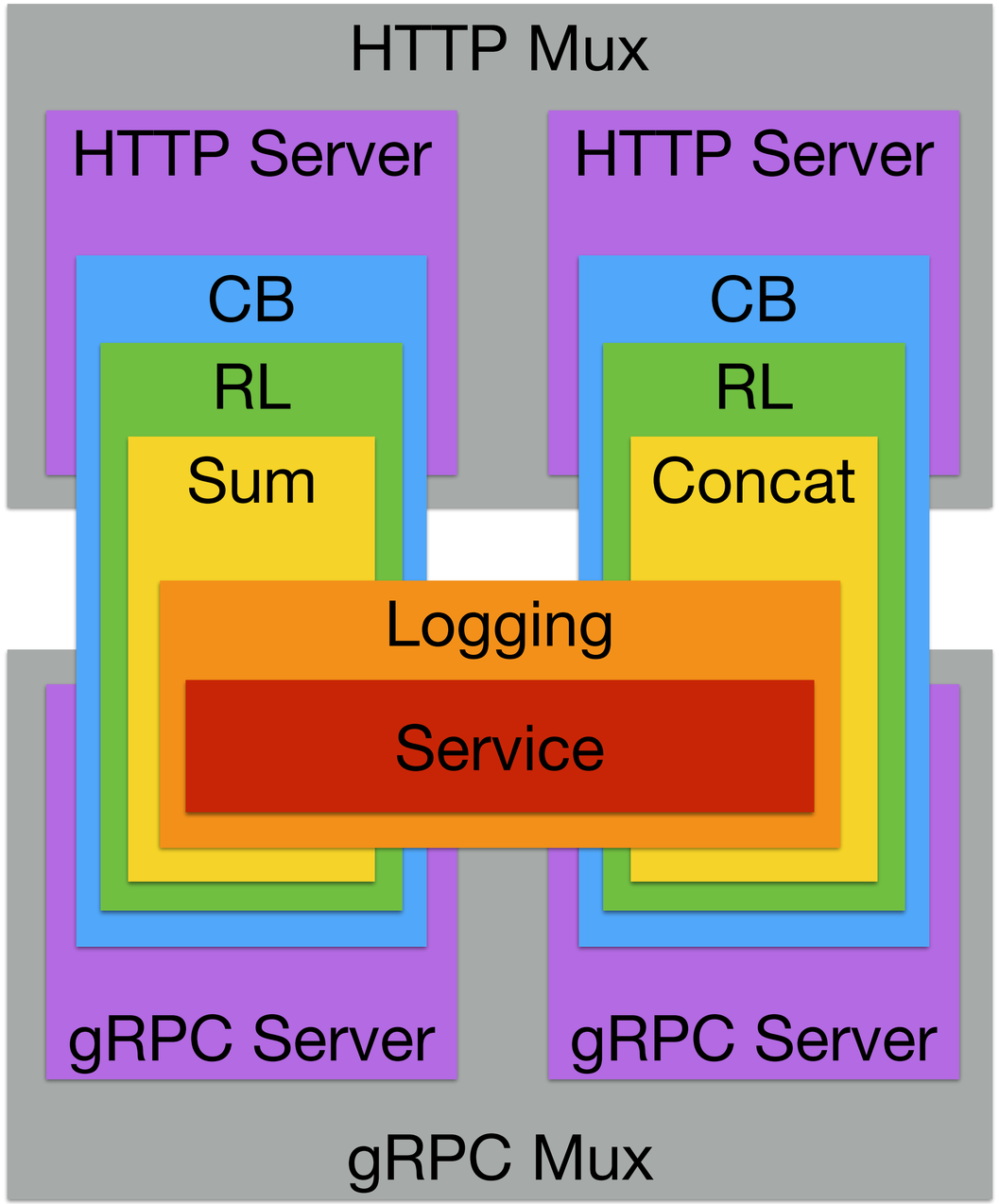


Challenge

A Service middleware that uppercases the result of Concat
(*Hint: strings.ToUpper)*

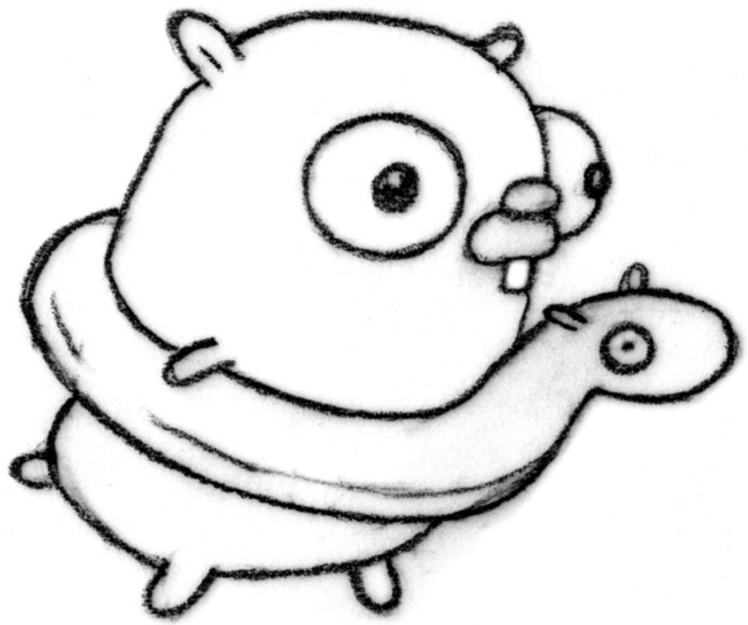



HTTP Mux
HTTP Server
HTTP Server
CB
CB
RL
RL
Sum
Concat
Logging
Service


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

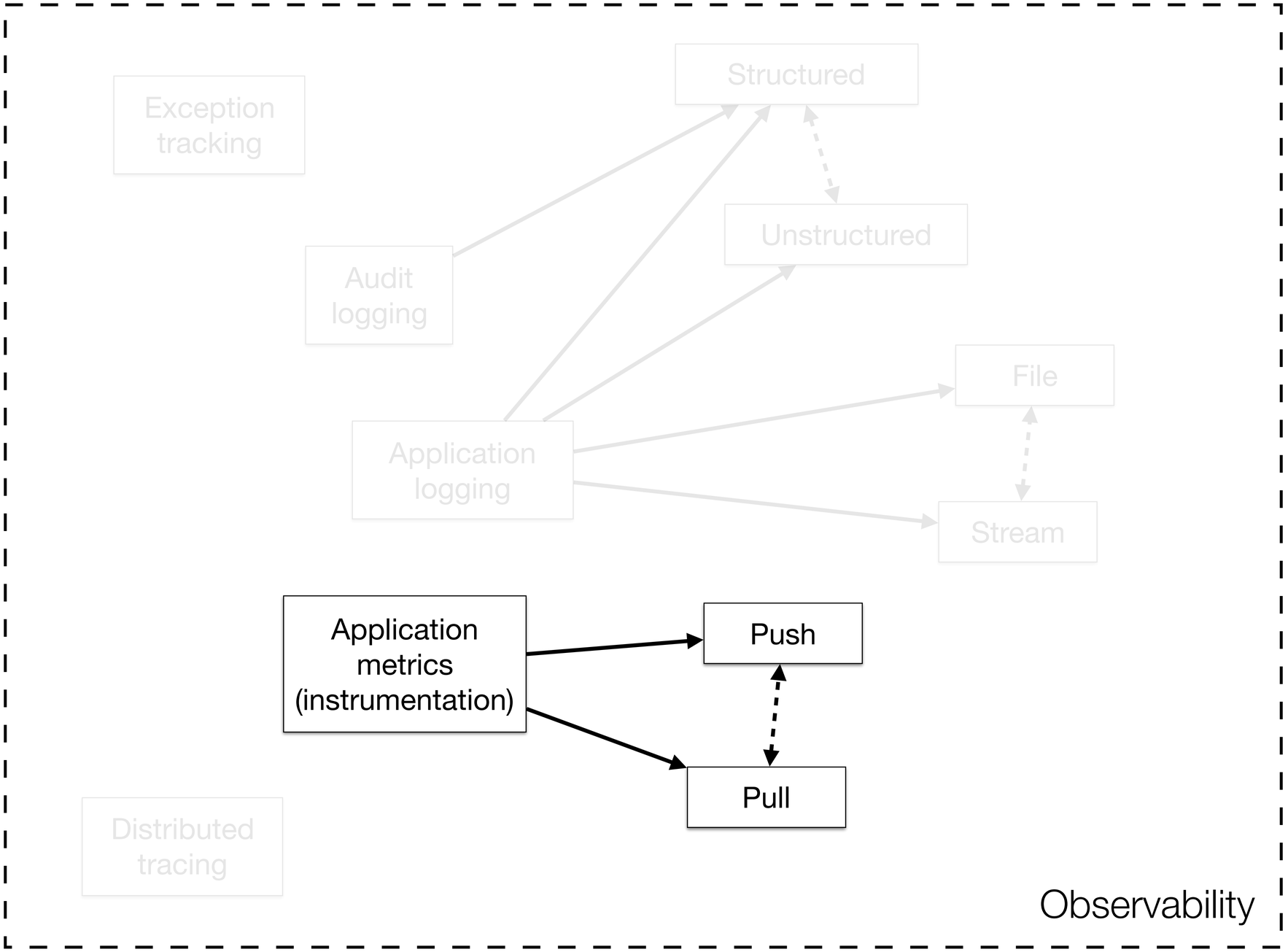


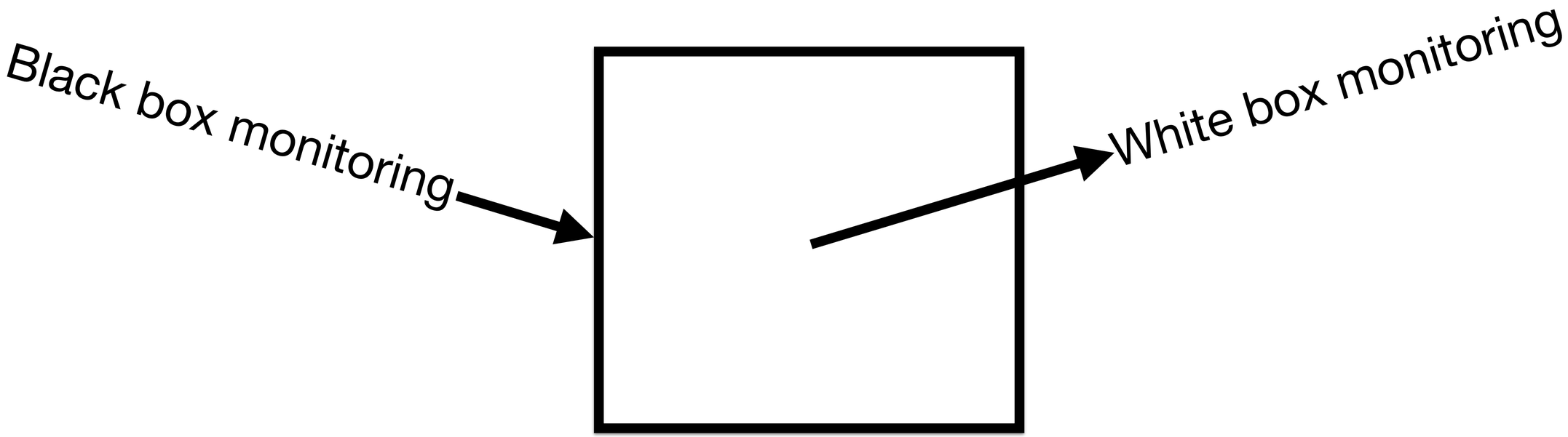
git apply

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)

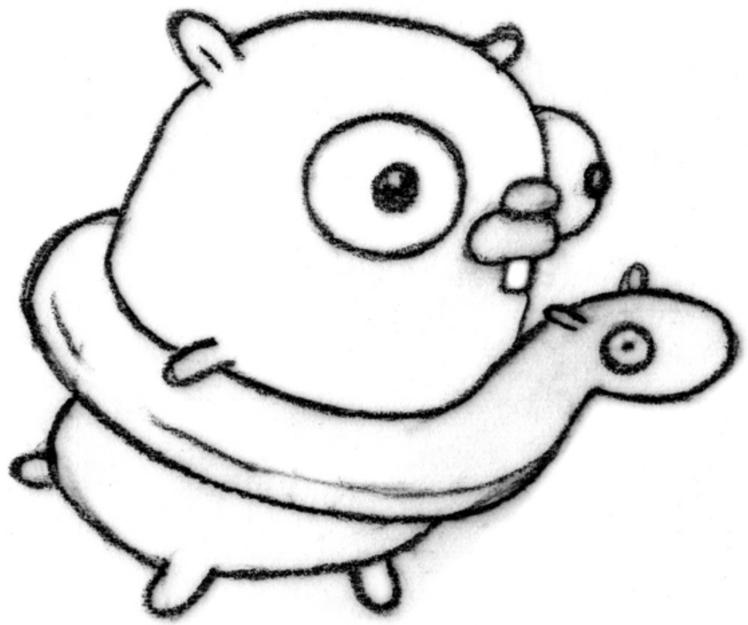
For resources e.g. queues

RED method

Tom Wilkie

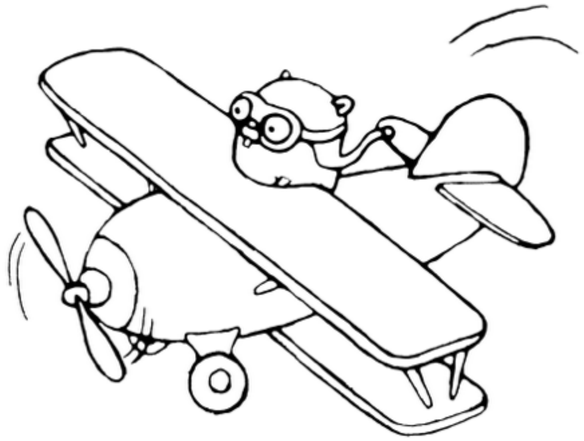
Request count (rate)
Error count (rate)
Duration

For e.g. endpoints



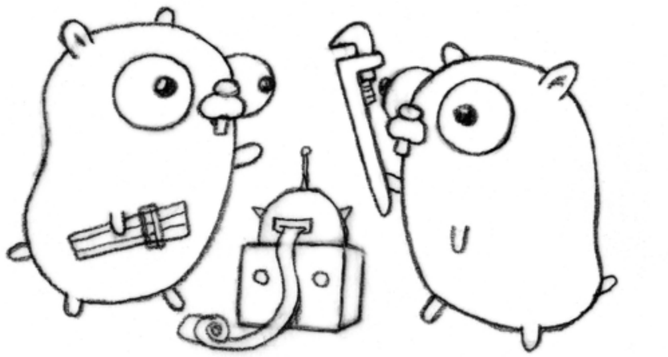
git apply

07 + Demo!



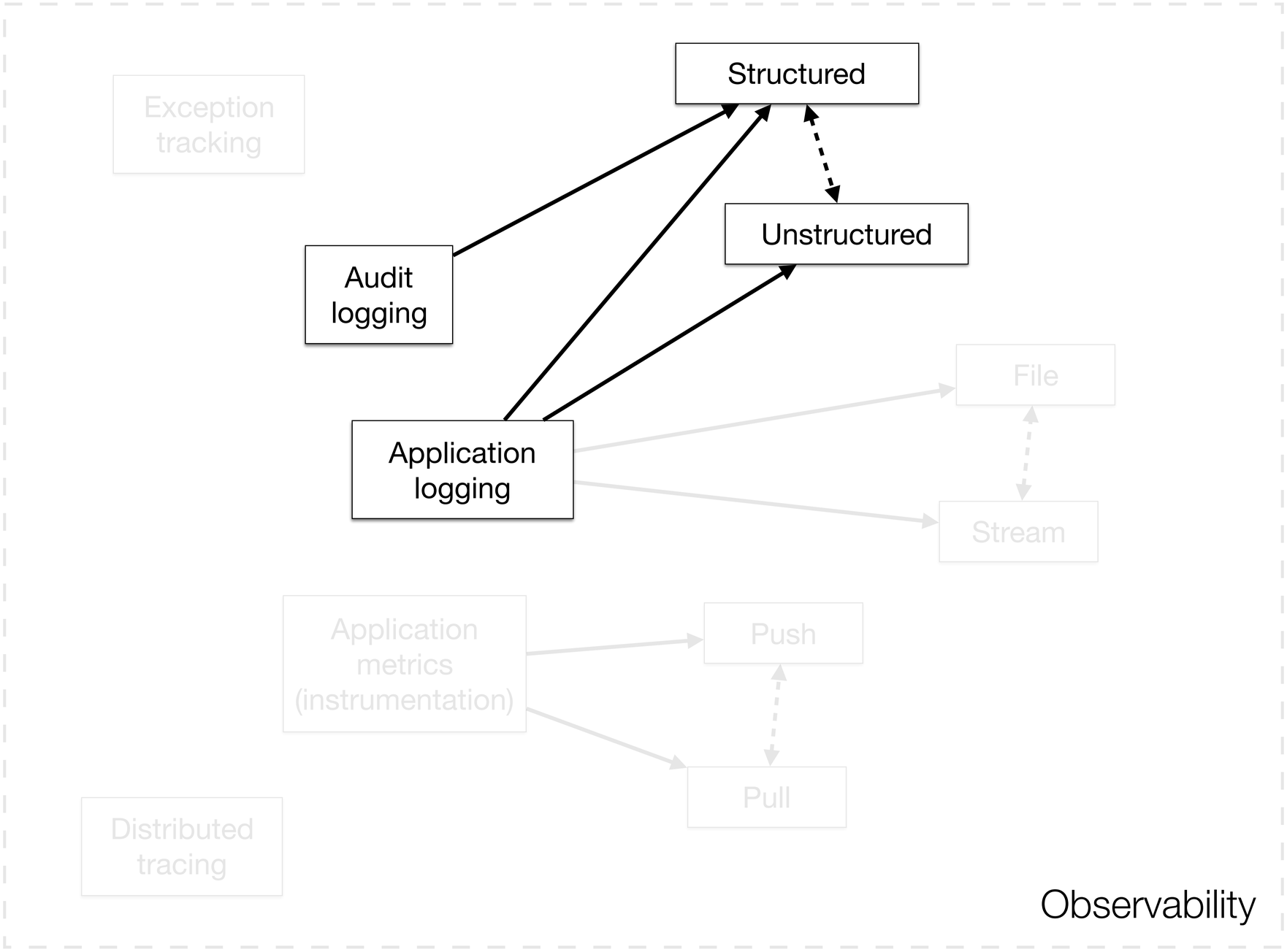
Challenge

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



package log

Structured, stream-oriented logging


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

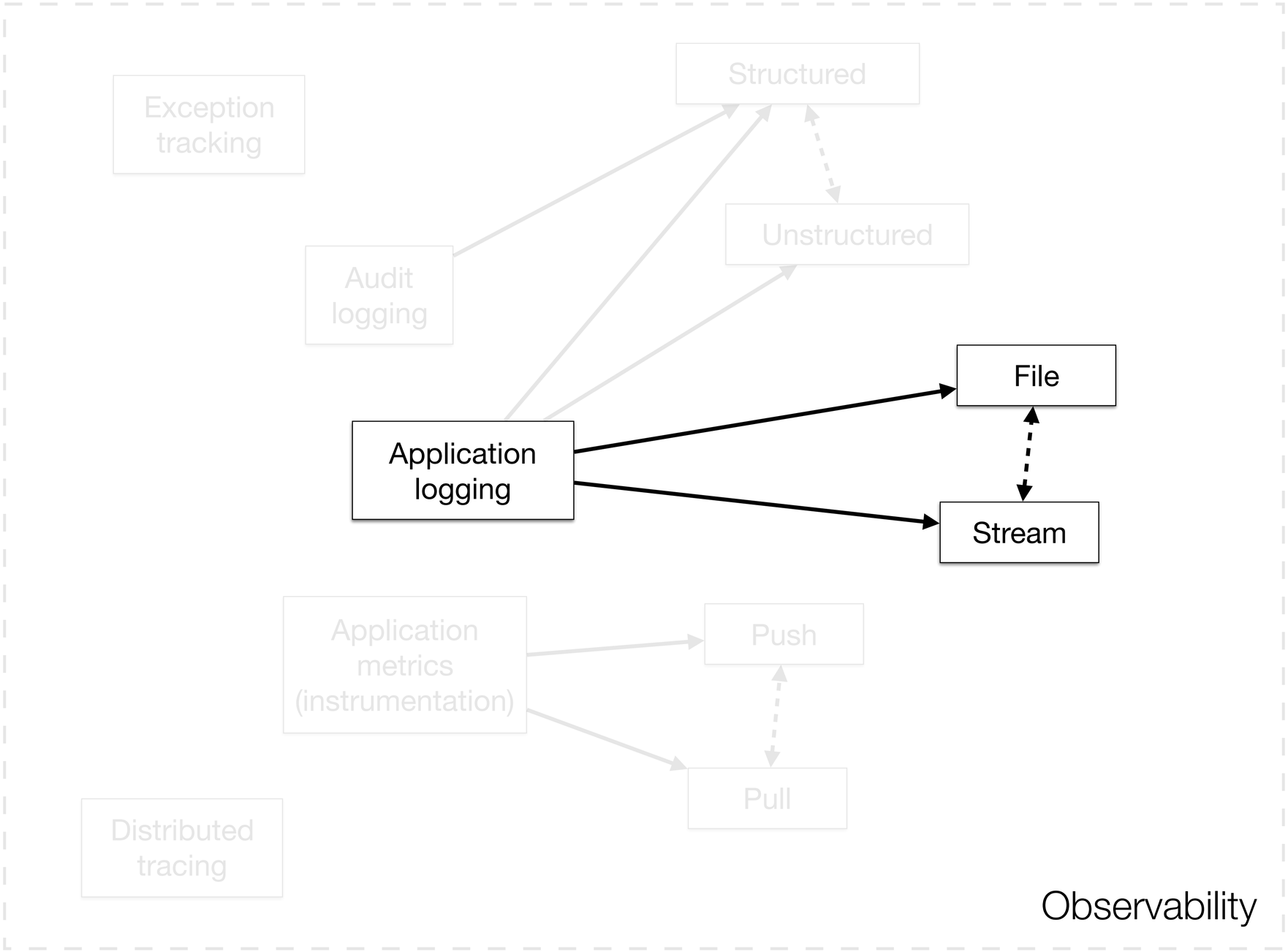
Compare/contrast

- `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" }`

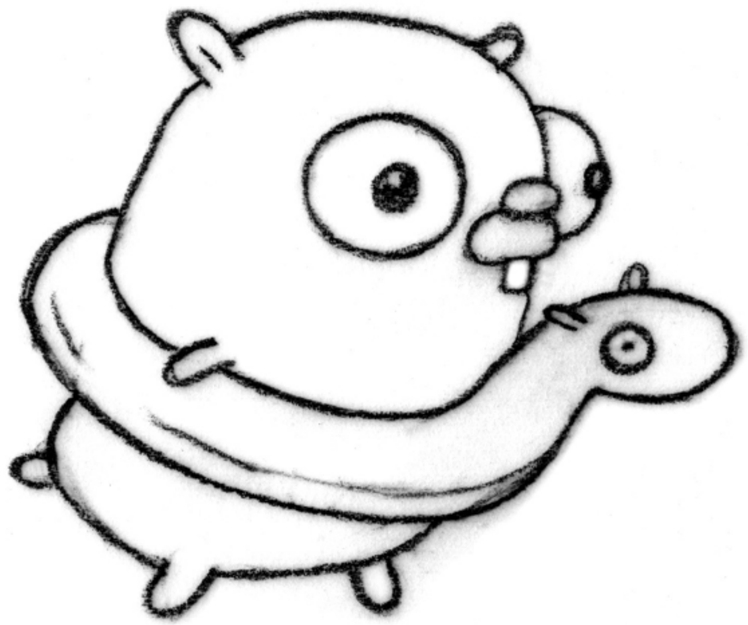
Benefits

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


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

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

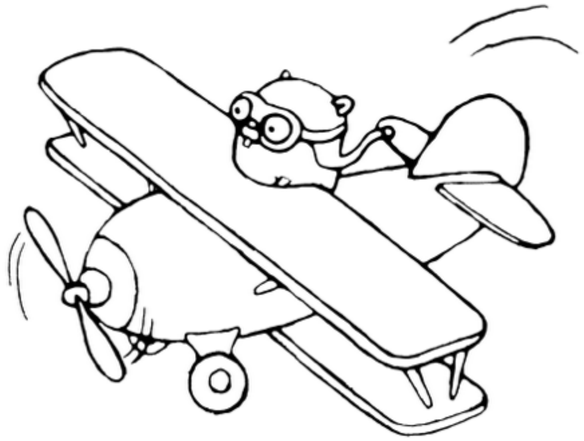


git apply

08, 09, 10

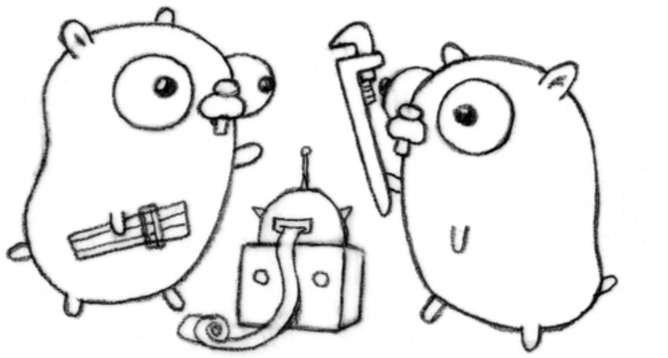
Logging v. instrumentation

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



Challenge

Thread a structured logger through to the transport

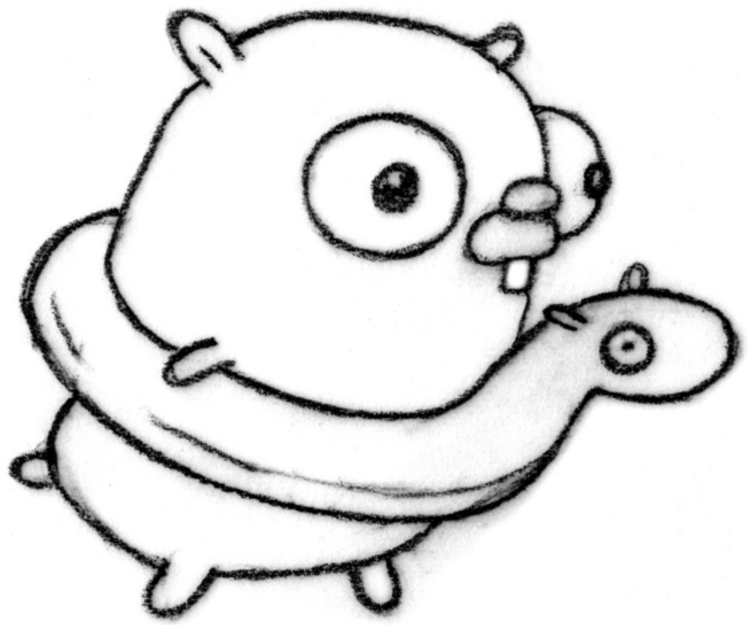


addsvc

Coda: context, error handling

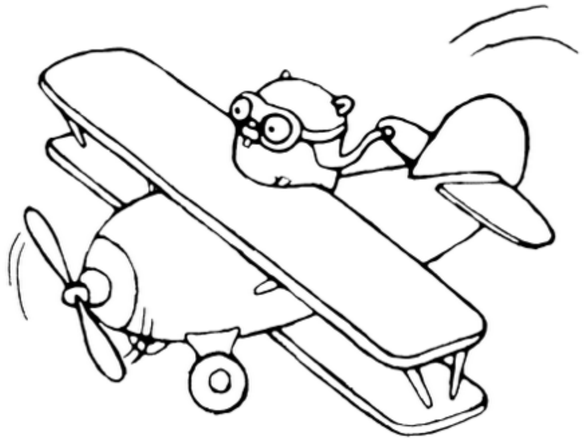
Context

Request-scoped data



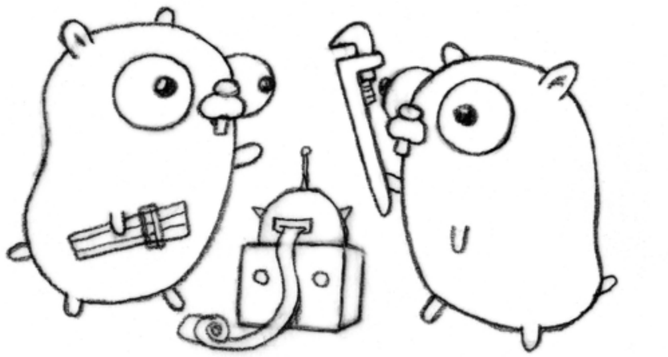
git apply

11



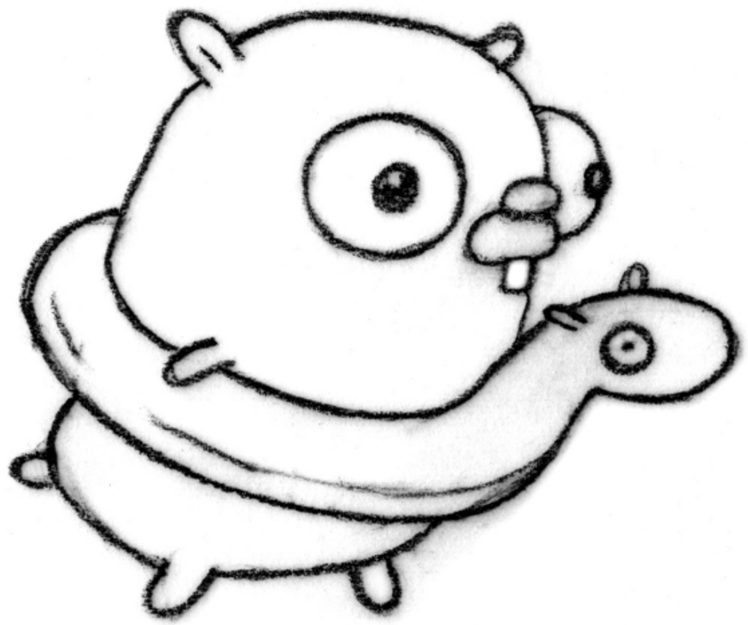
Challenge

Take X-Request-ID from the HTTP request, add it to each log line.
(If it doesn't exist, create it. Hint: use kit/transport/http.ServerBefore)



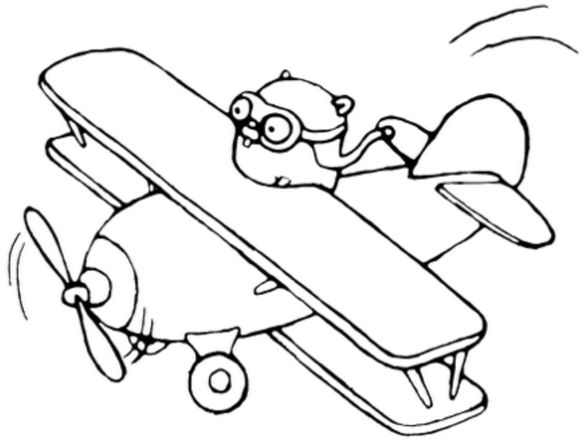
Error handling

Mapping business to transport domain



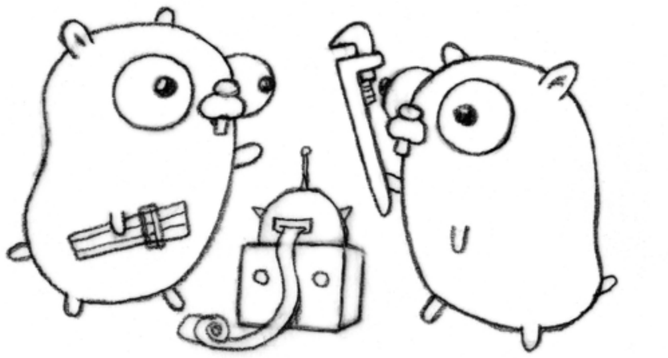
git apply

12, 13, 14



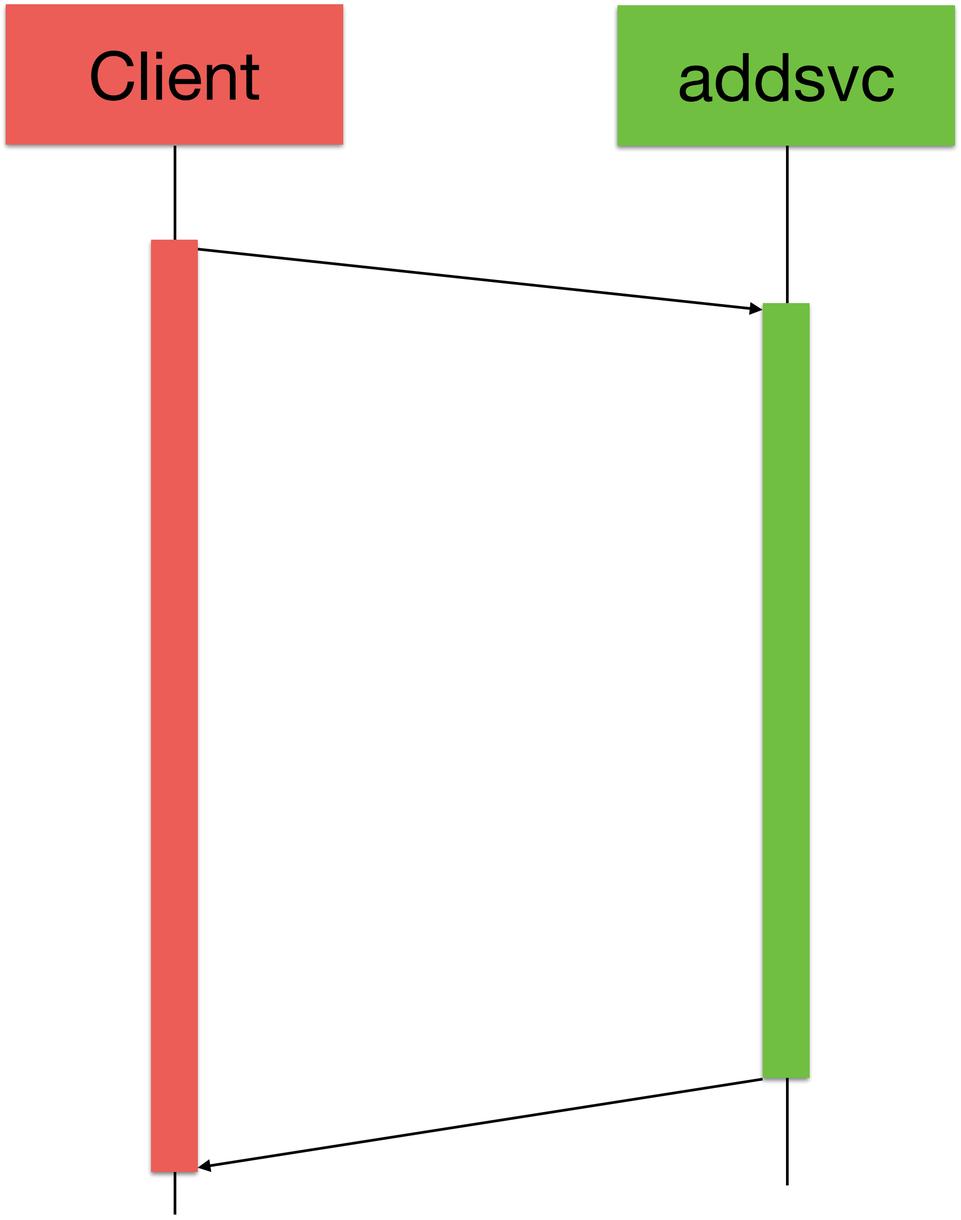
Challenge

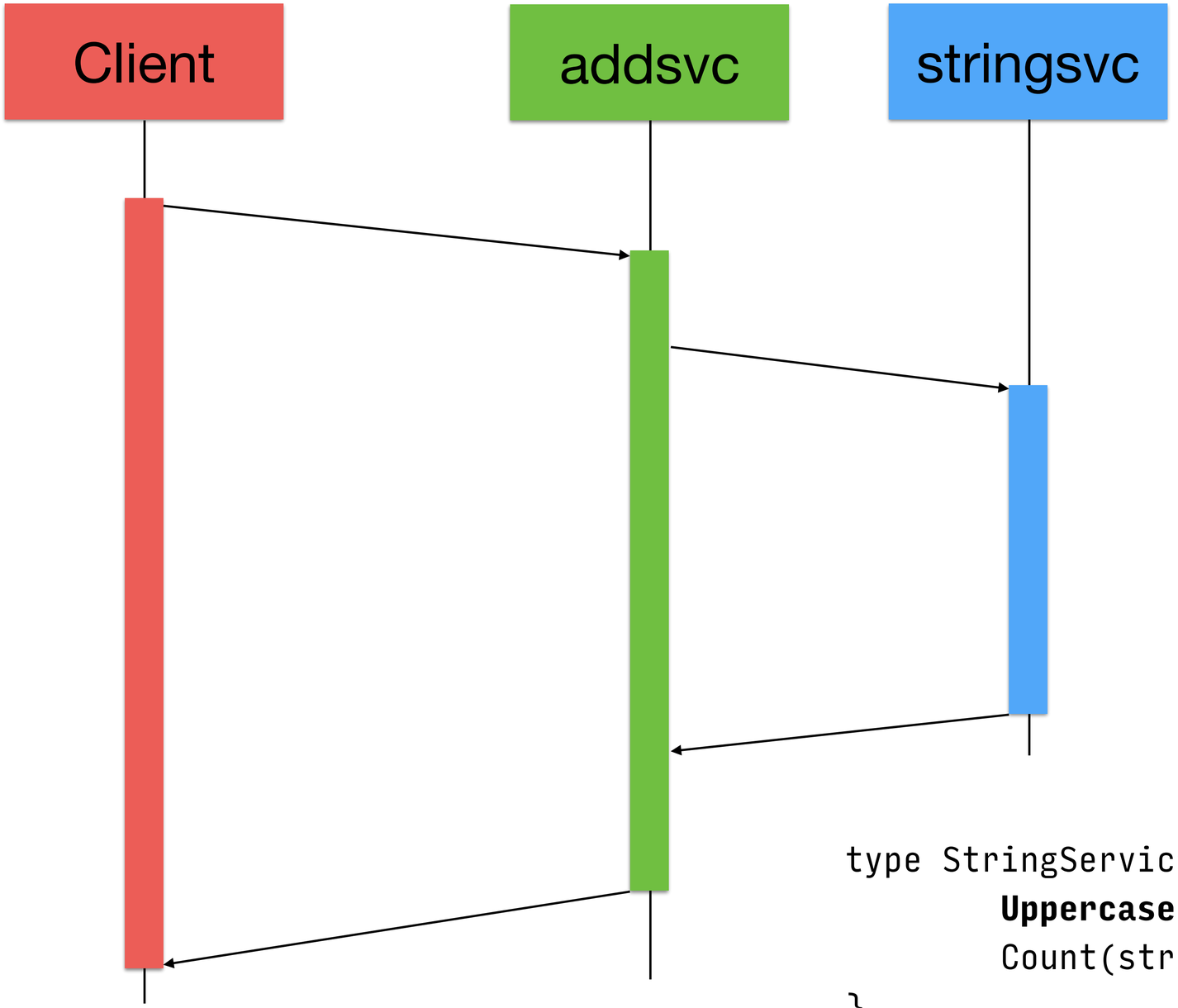
Create a new business logic error that will return HTTP 418



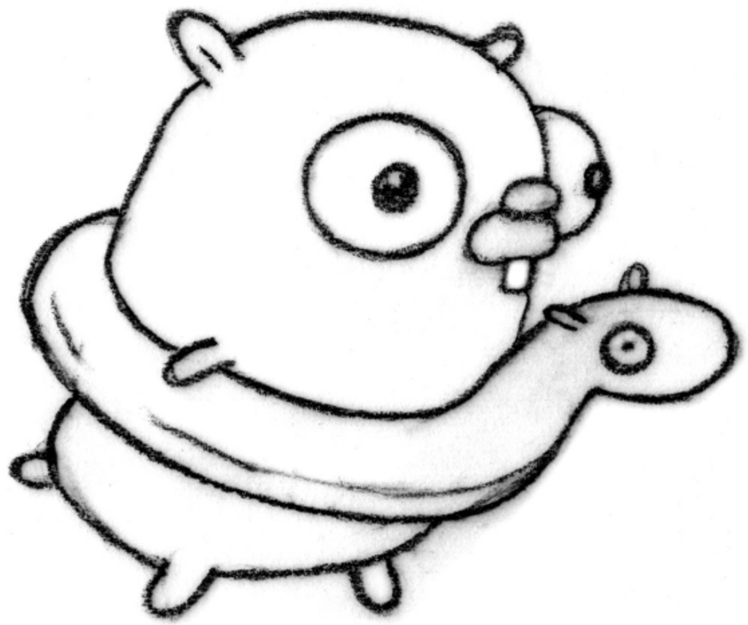
addsvc

Dependency on stringsvc


Client
addsvc

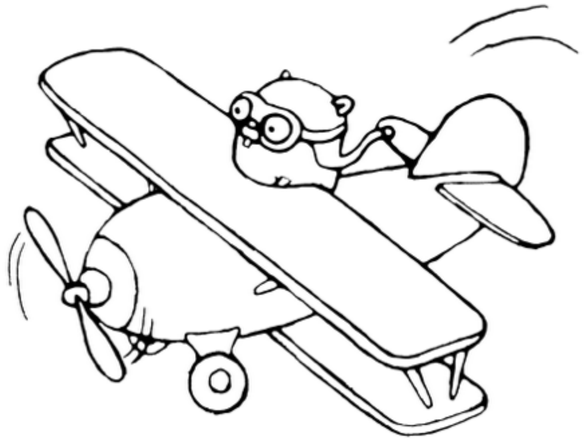



```
type StringService interface {
        Uppercase(string) (string, error)
        Count(string) int
}
```



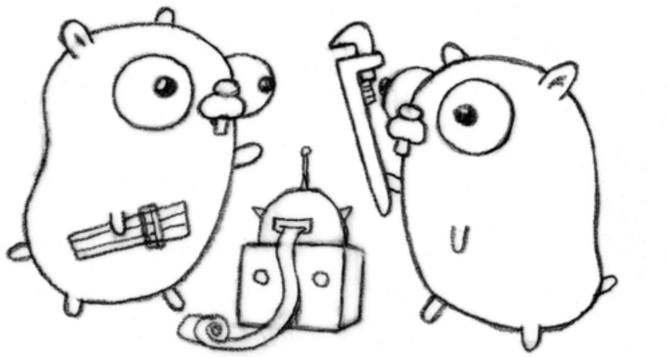
git apply

15, 16, 17



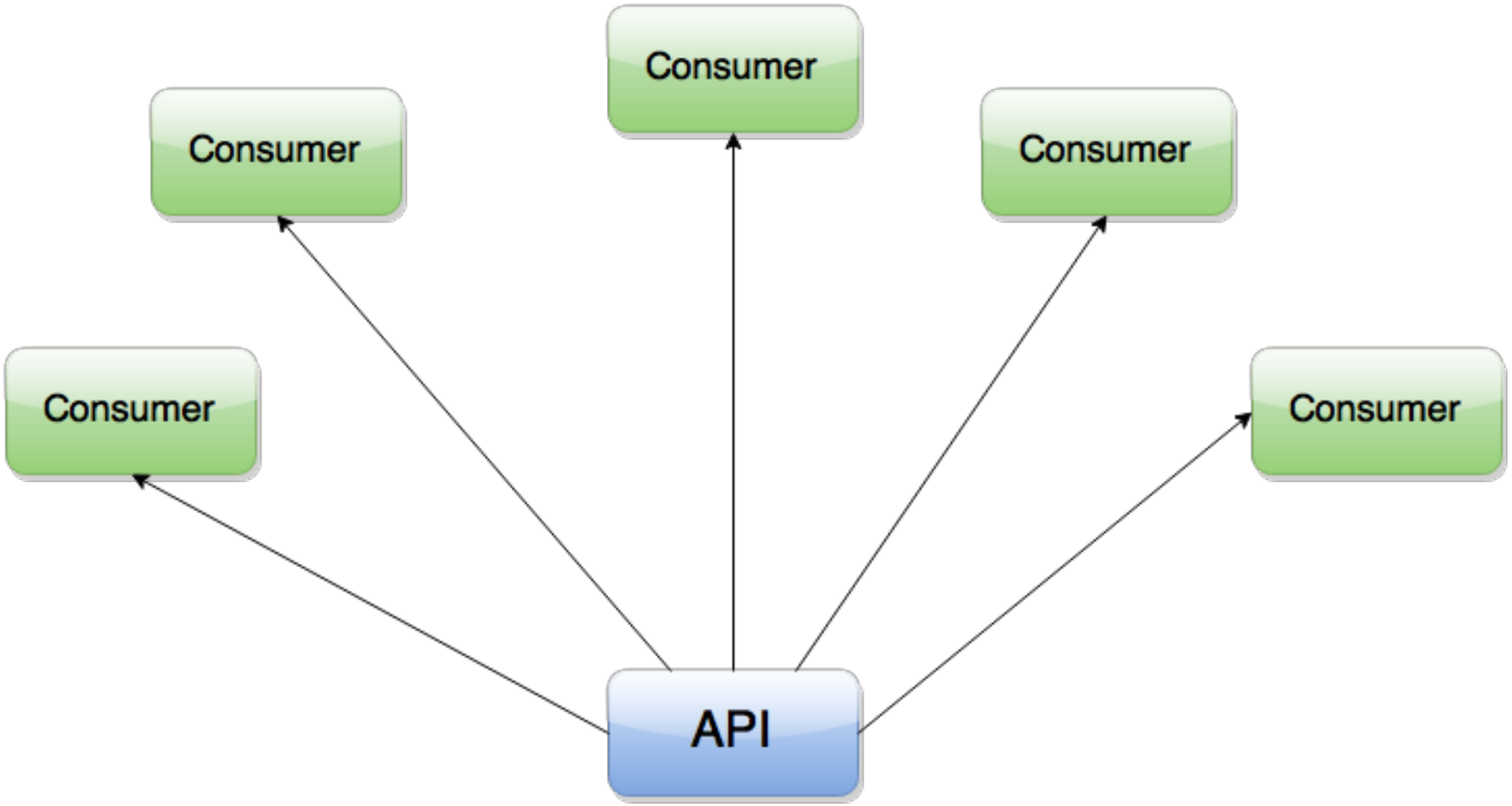
Challenge

Make UPPERCASE via stringsvc optional

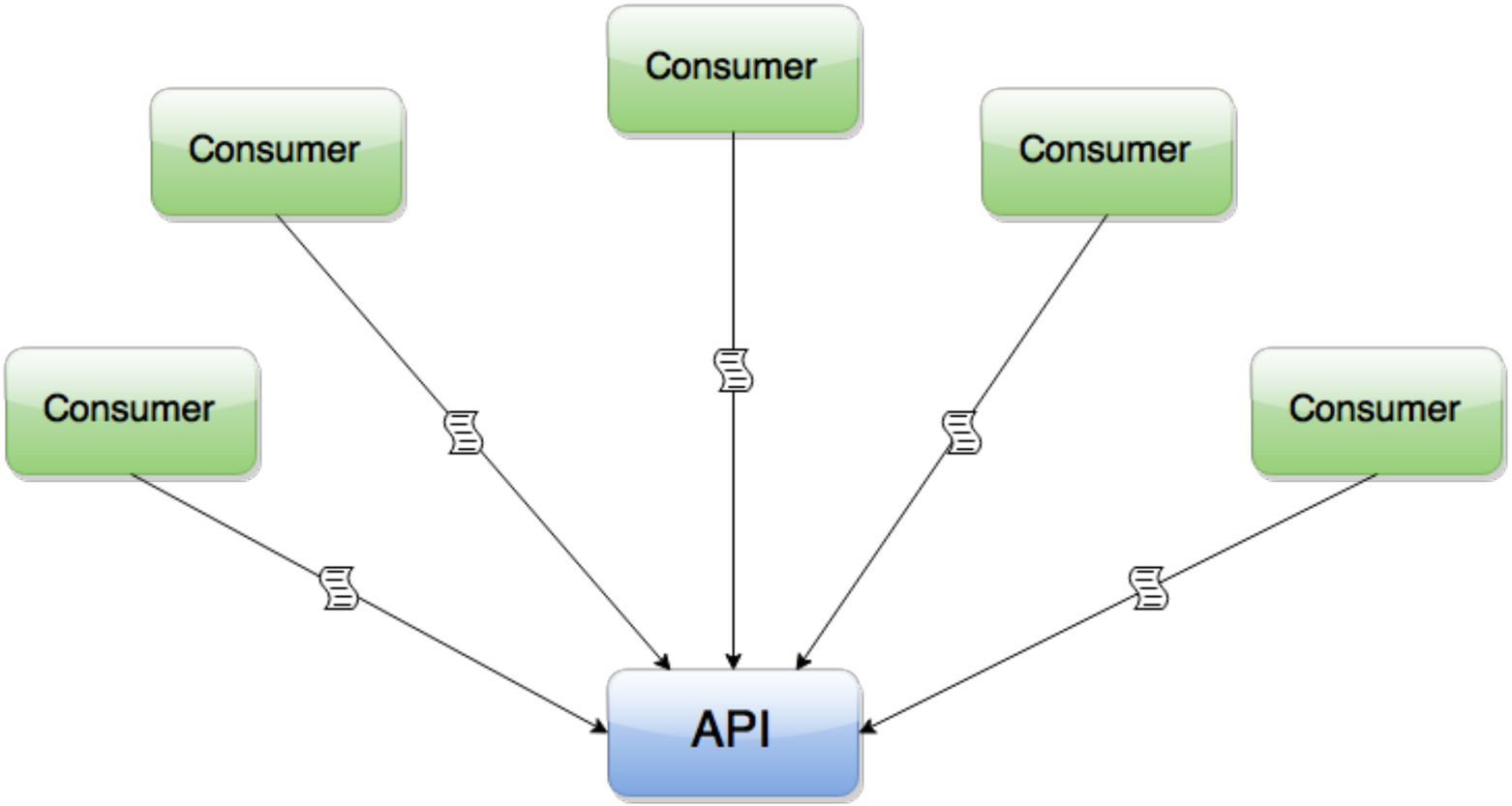


addsvc

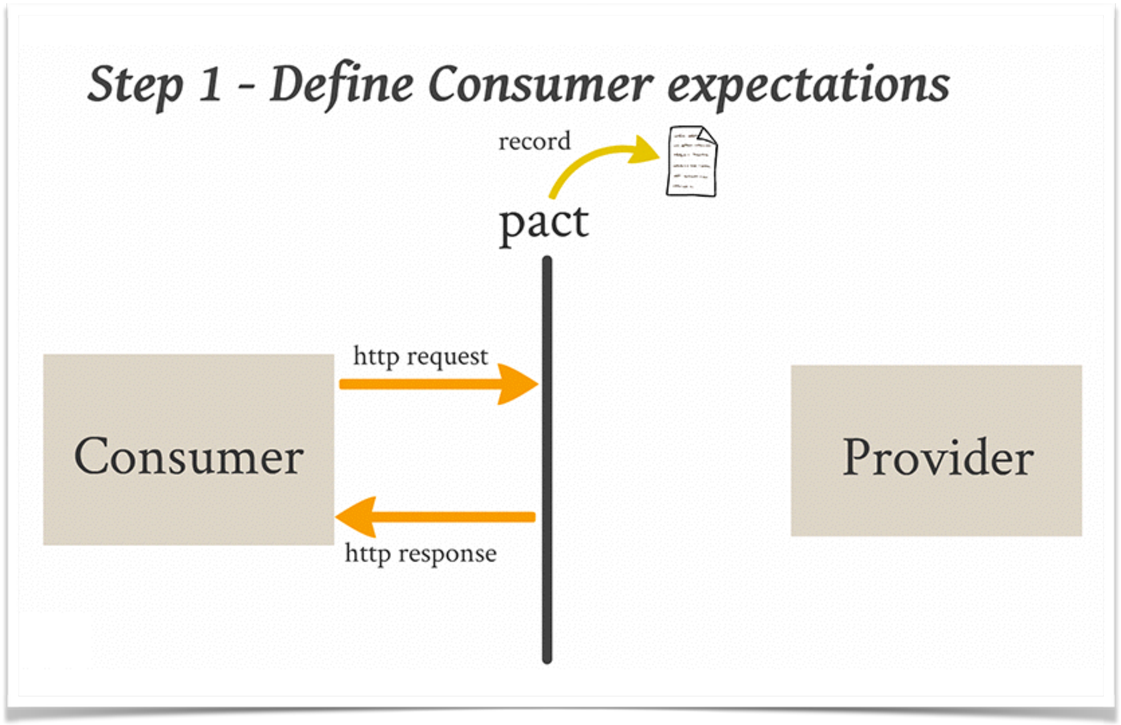
Contract testing with Pact

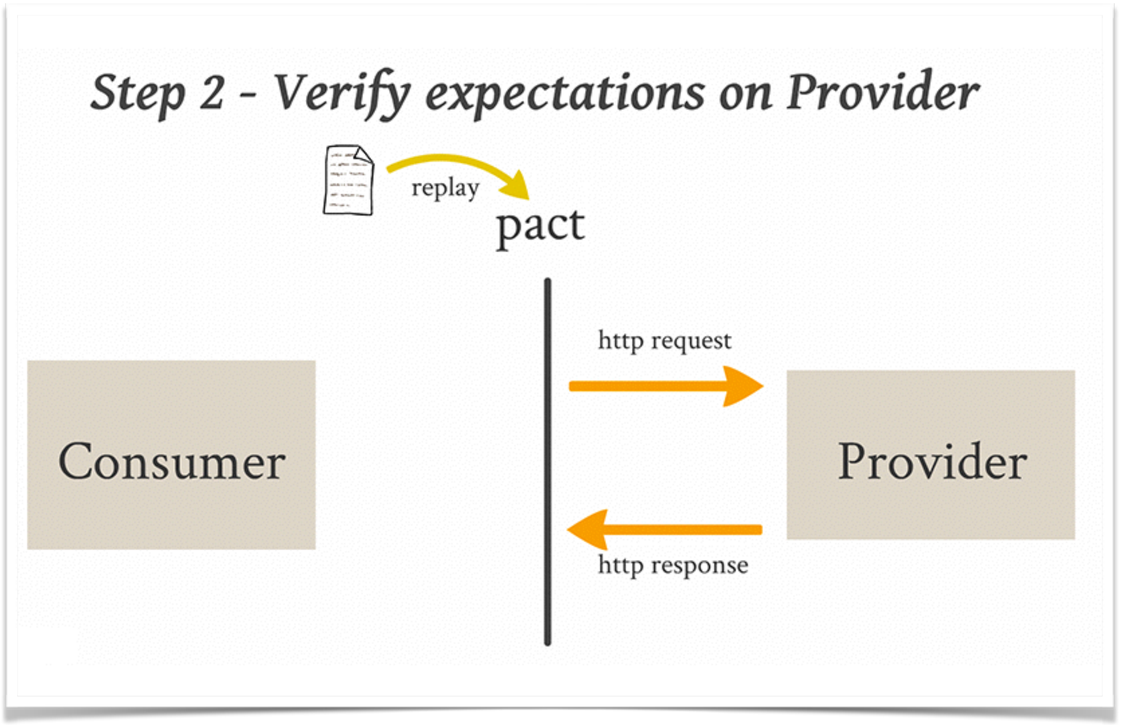


http://dius.com.au/2016/02/03/microservices-pact/

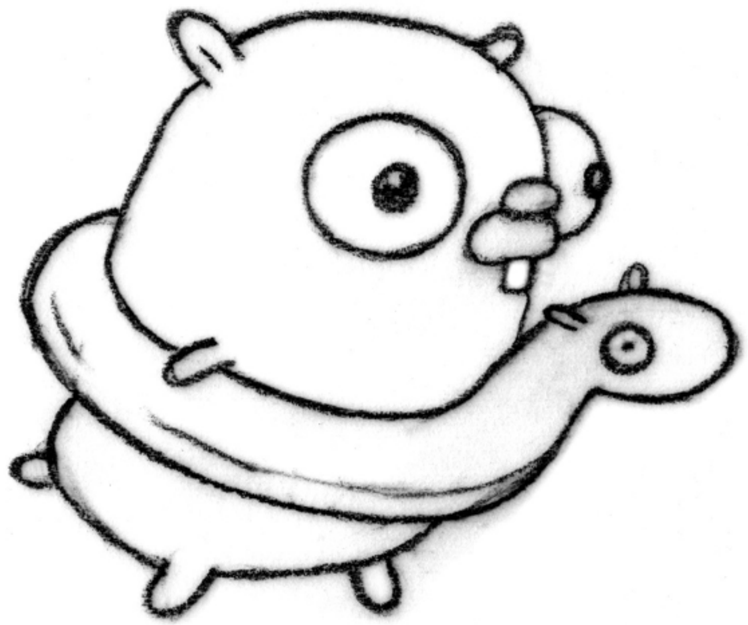


http://dius.com.au/2016/02/03/microservices-pact/

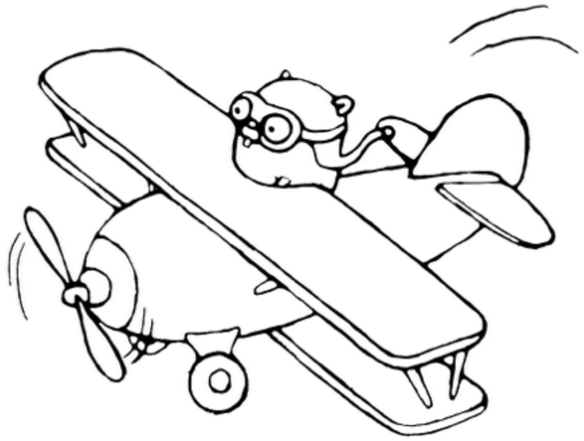




http://dius.com.au/2016/02/03/microservices-pact/

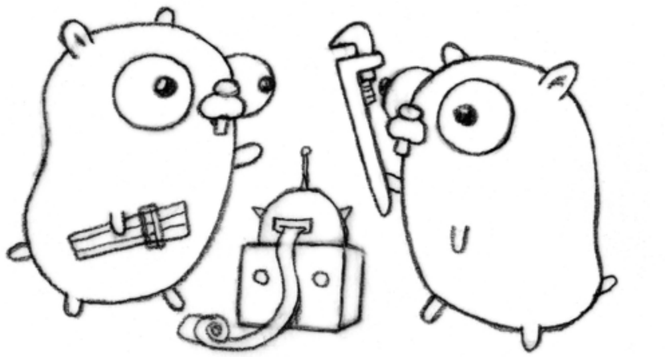


git apply



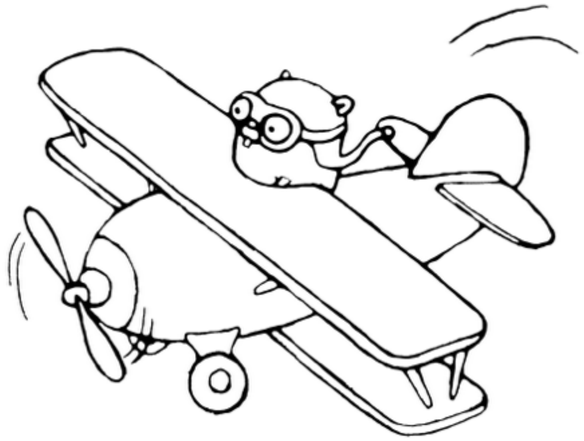
Challenge

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



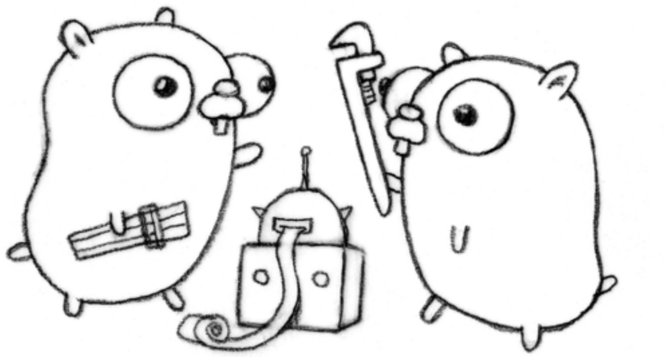
What about unit tests?

- **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



Challenge

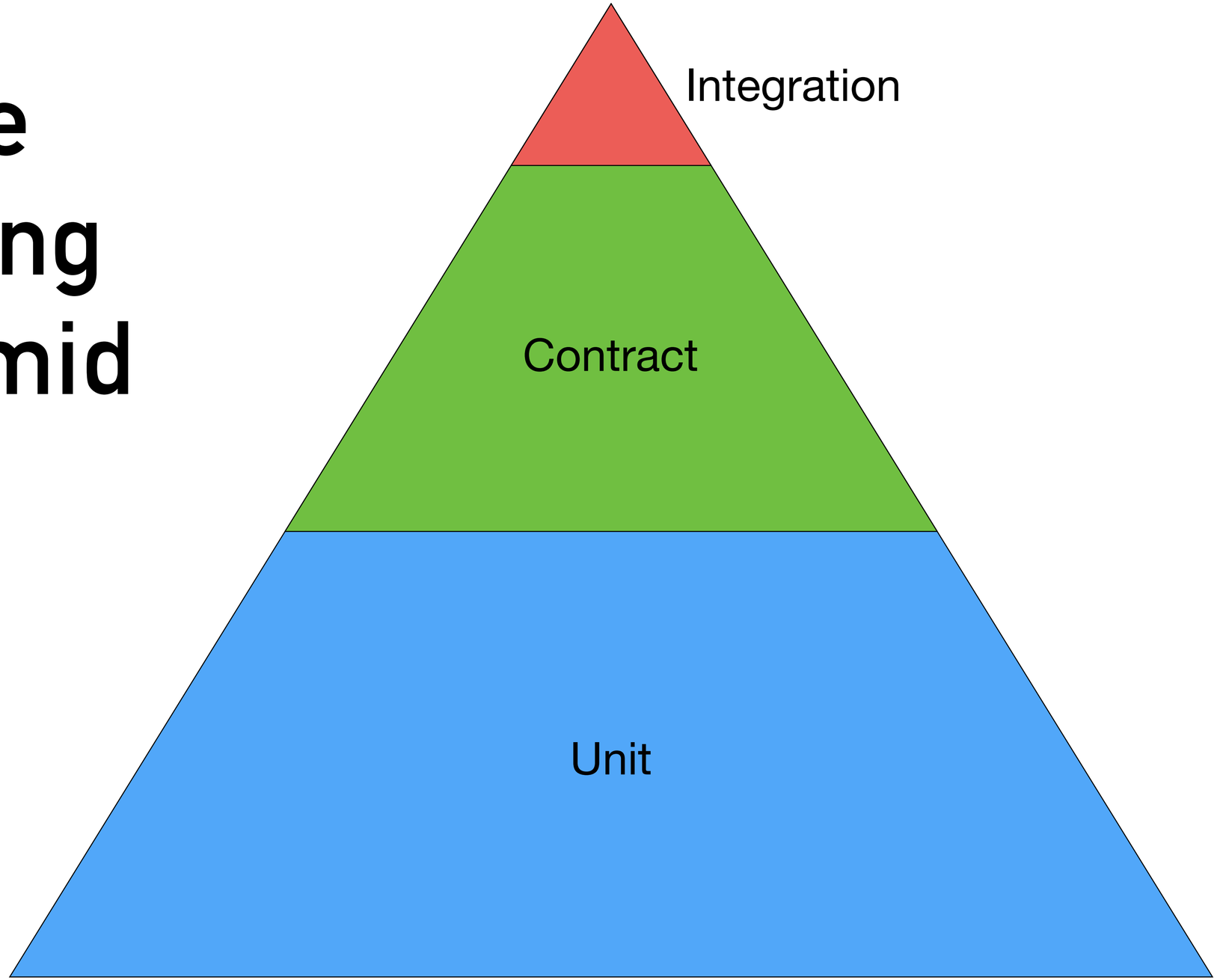
Brainstorm good unit tests;
implement a few on a component



What about integration tests?

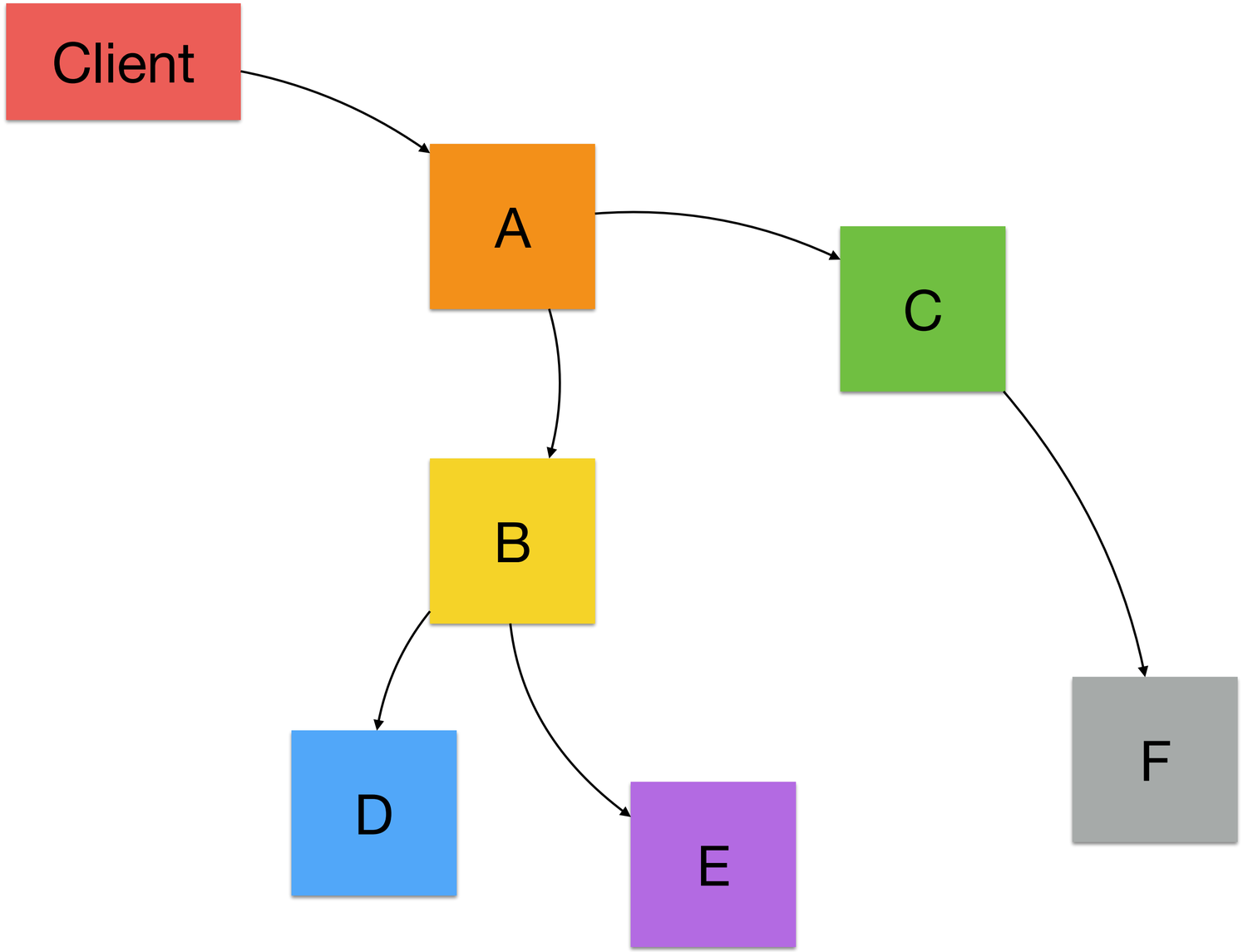
- **Integration tests** can mean different things
  - For a single service: test DB, etc. — use Docker
  - For many services: test interactions — already covered by contracts!
- With microservices, there is no steady state
- Google Testing Blog: Just Say No to more End-to-End Tests

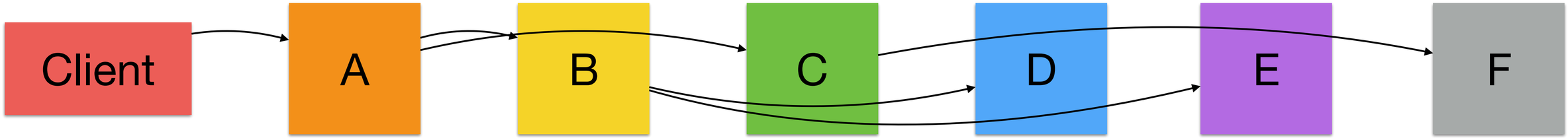
The Testing Pyramid

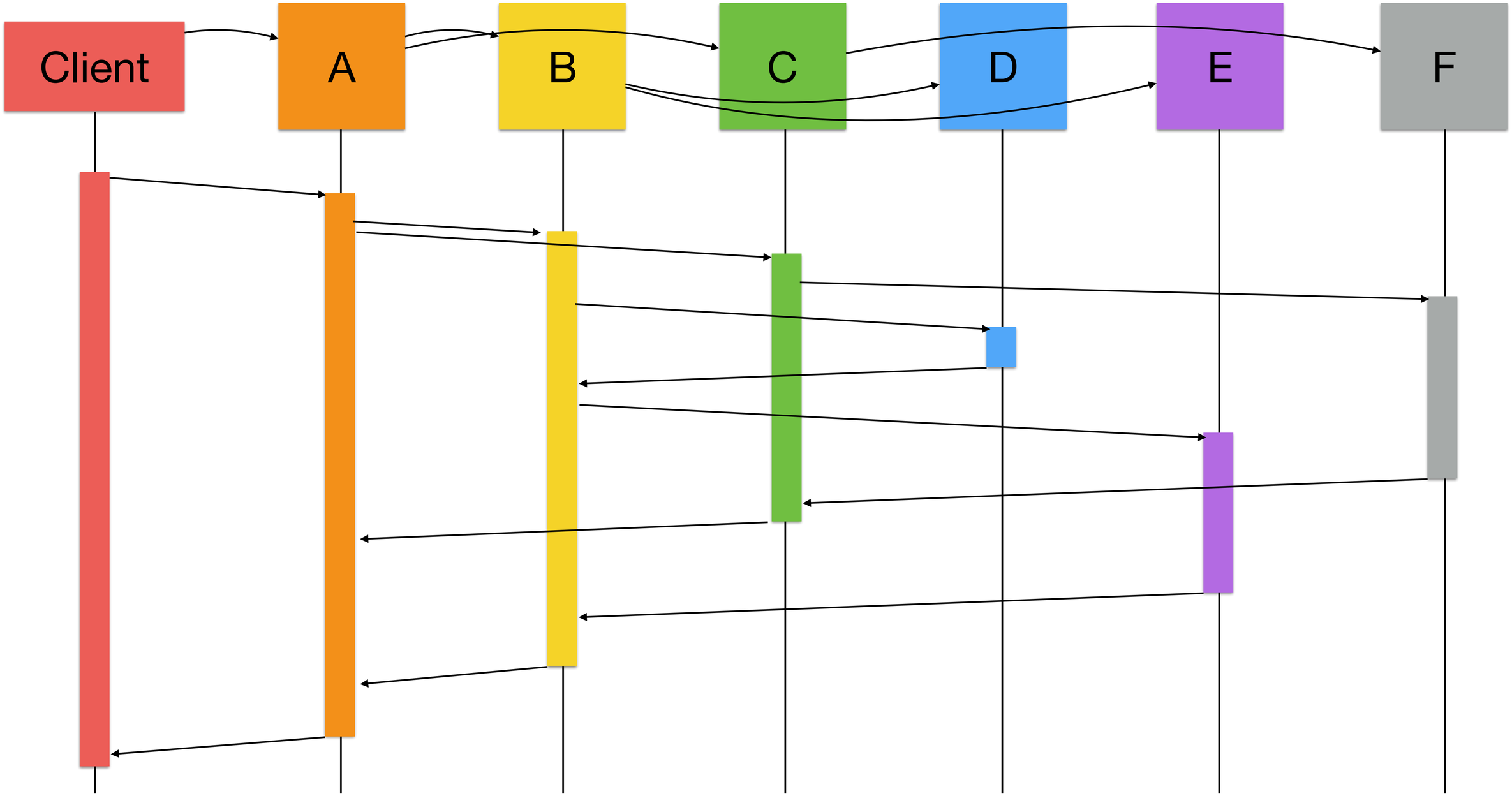


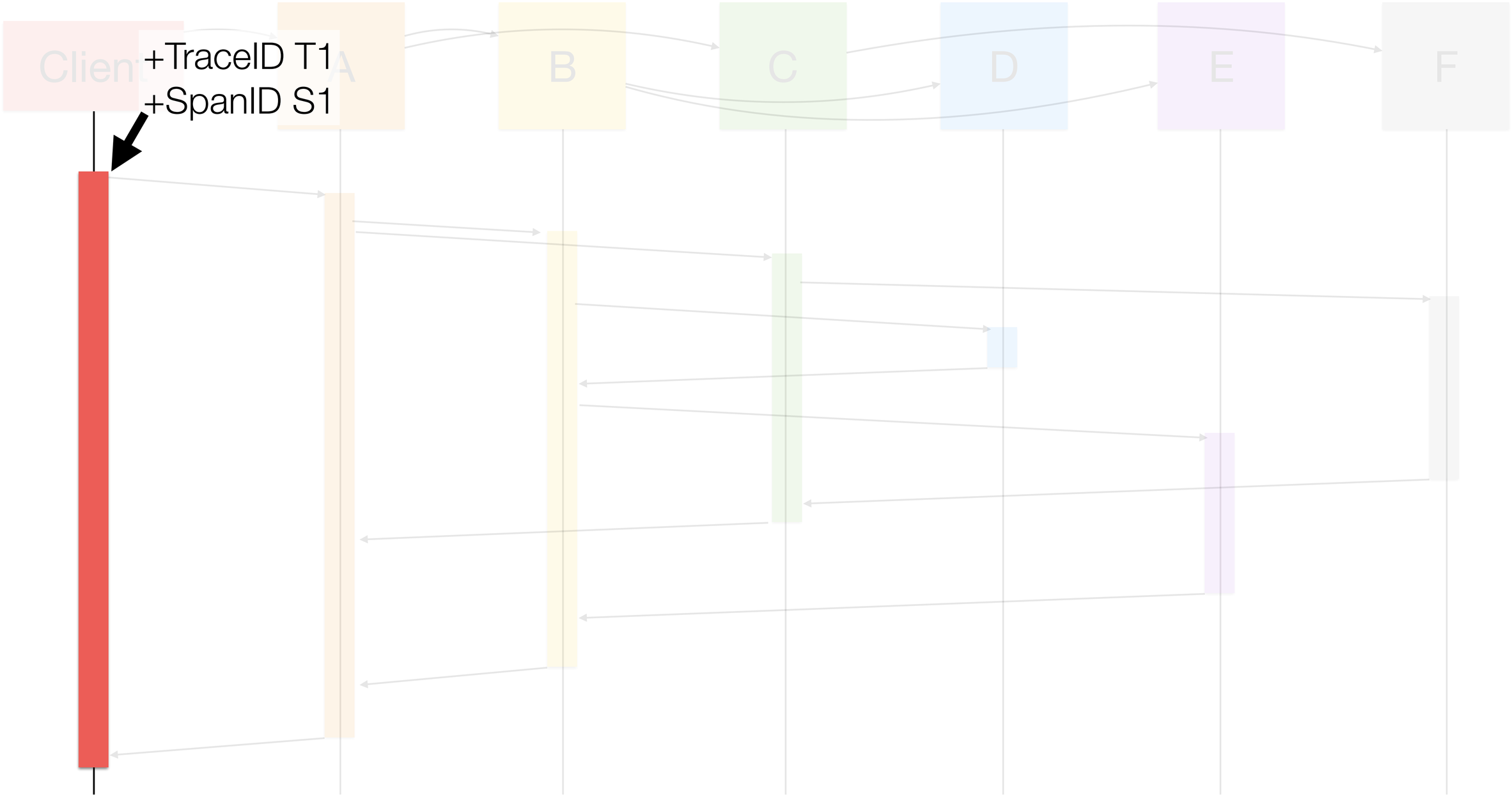
addsvc

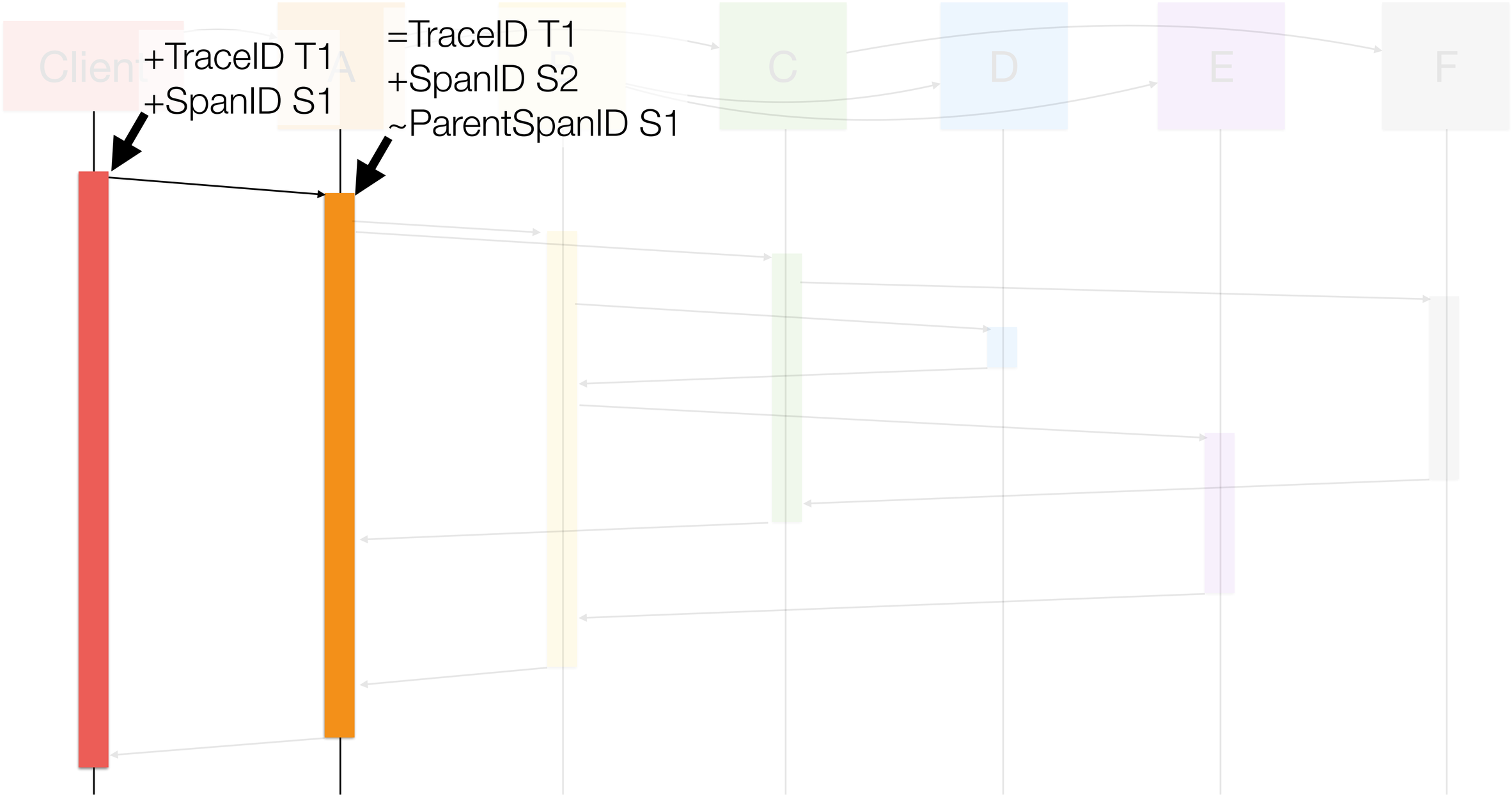
Distributed tracing with Tracer

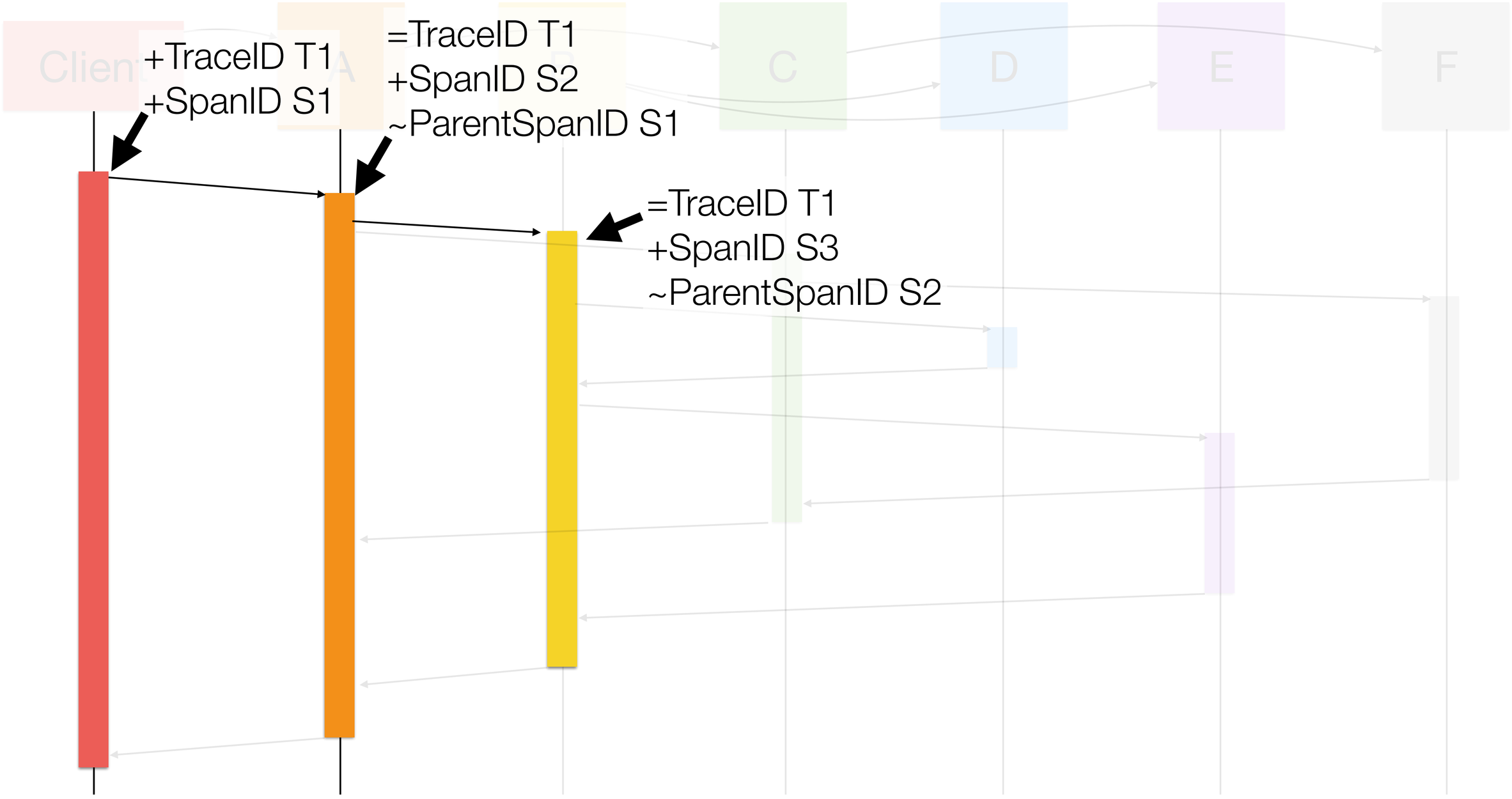

Client
A
C
B
D
E
F

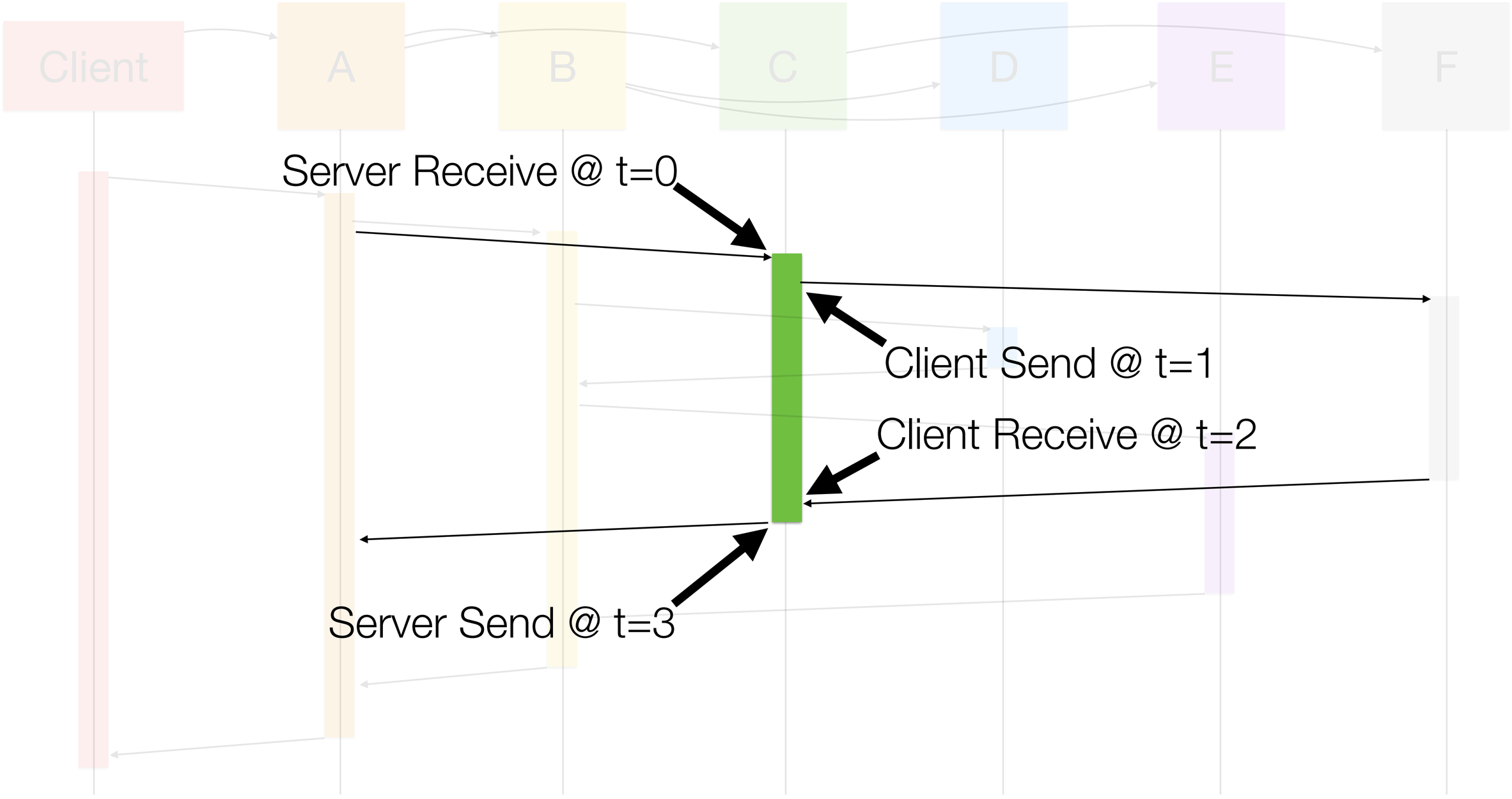

Client
A
B
C
D
E
F

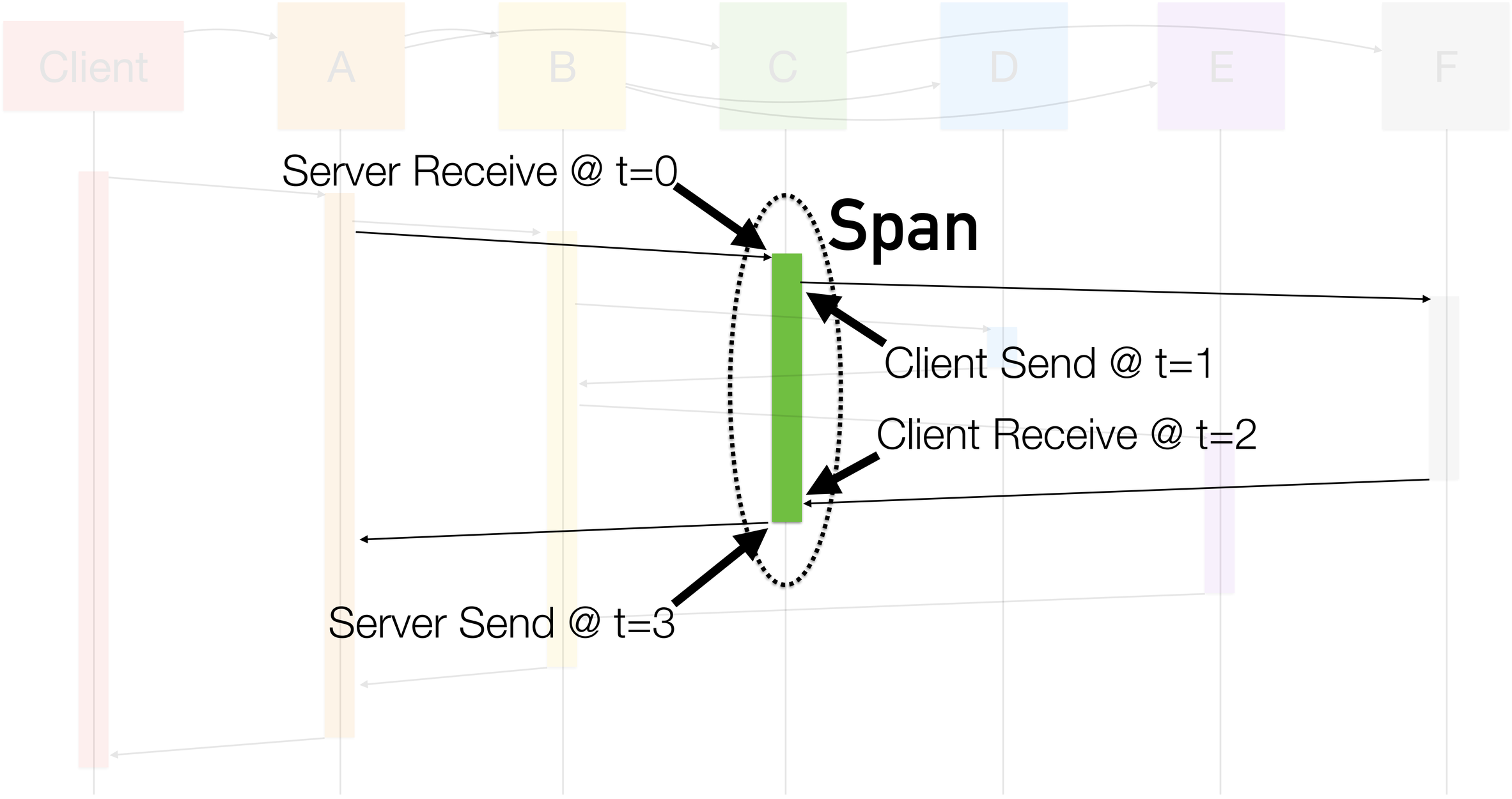

Client
A
B
C
D
E
F

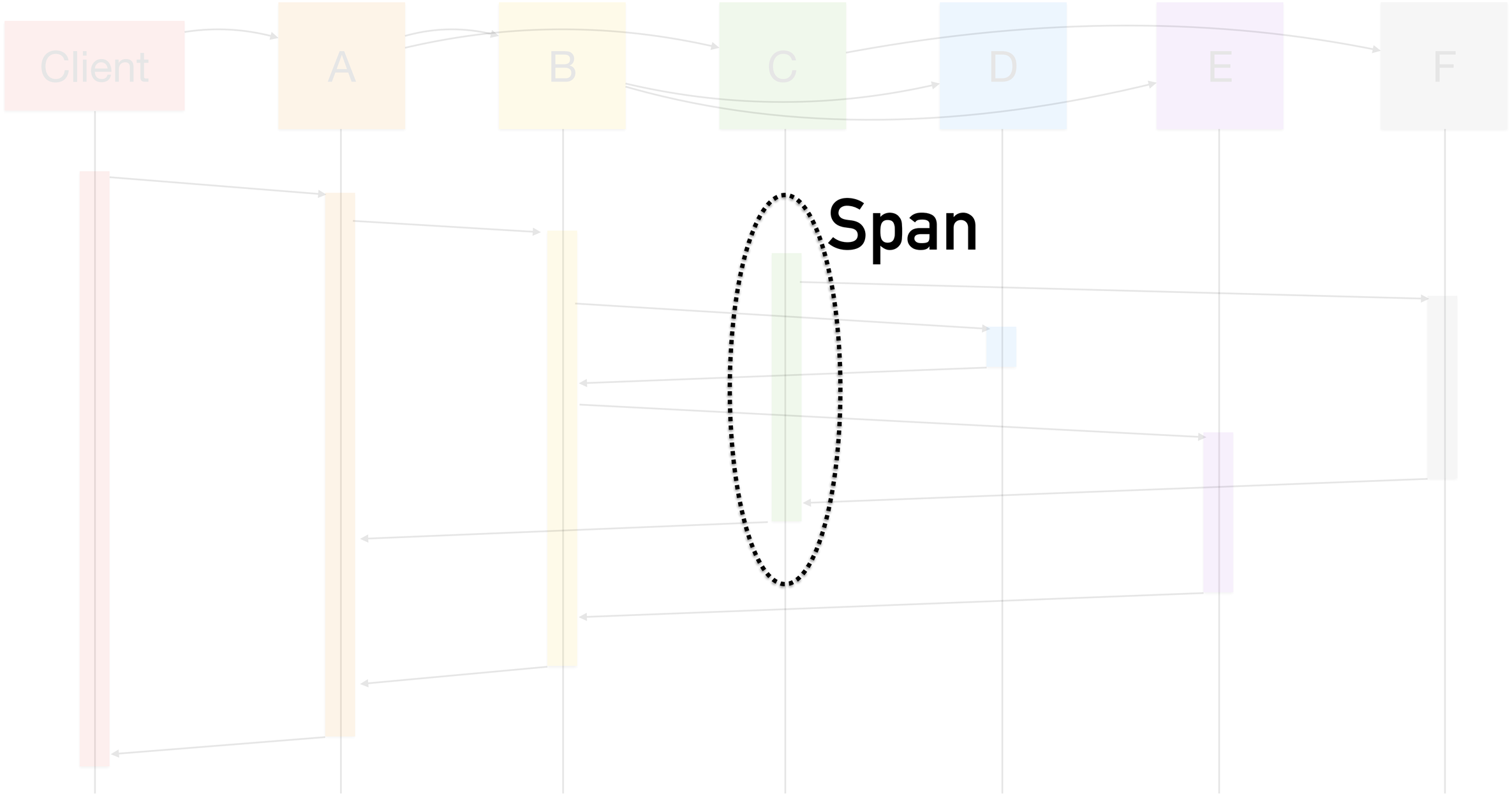

Client
A
B
C
D
E
F
+TraceID T1
+SpanID S1

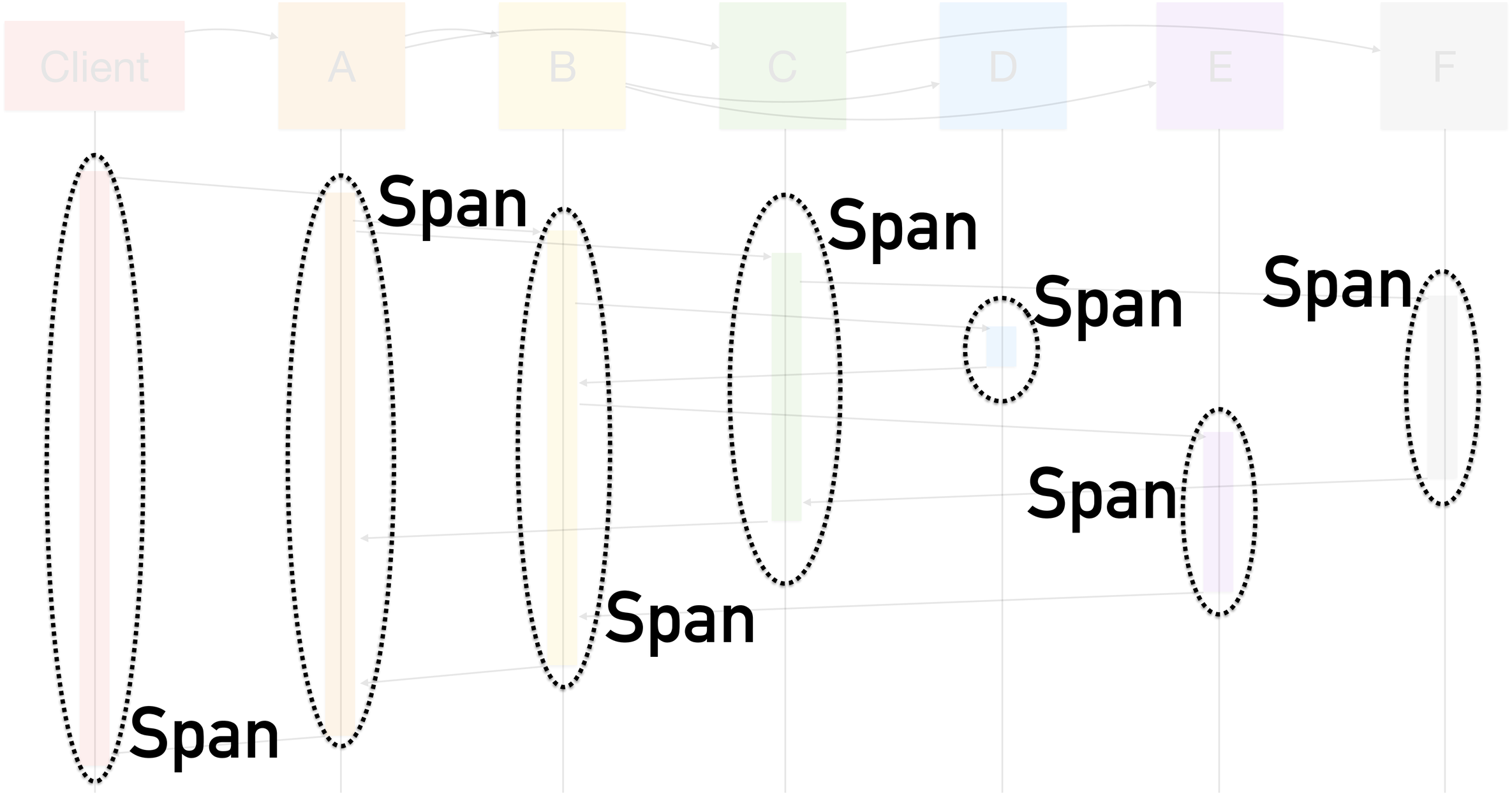

+TraceID T1
+SpanID S1
=TraceID T1
+SpanID S2
~ParentSpanID S1
C
D
E
F

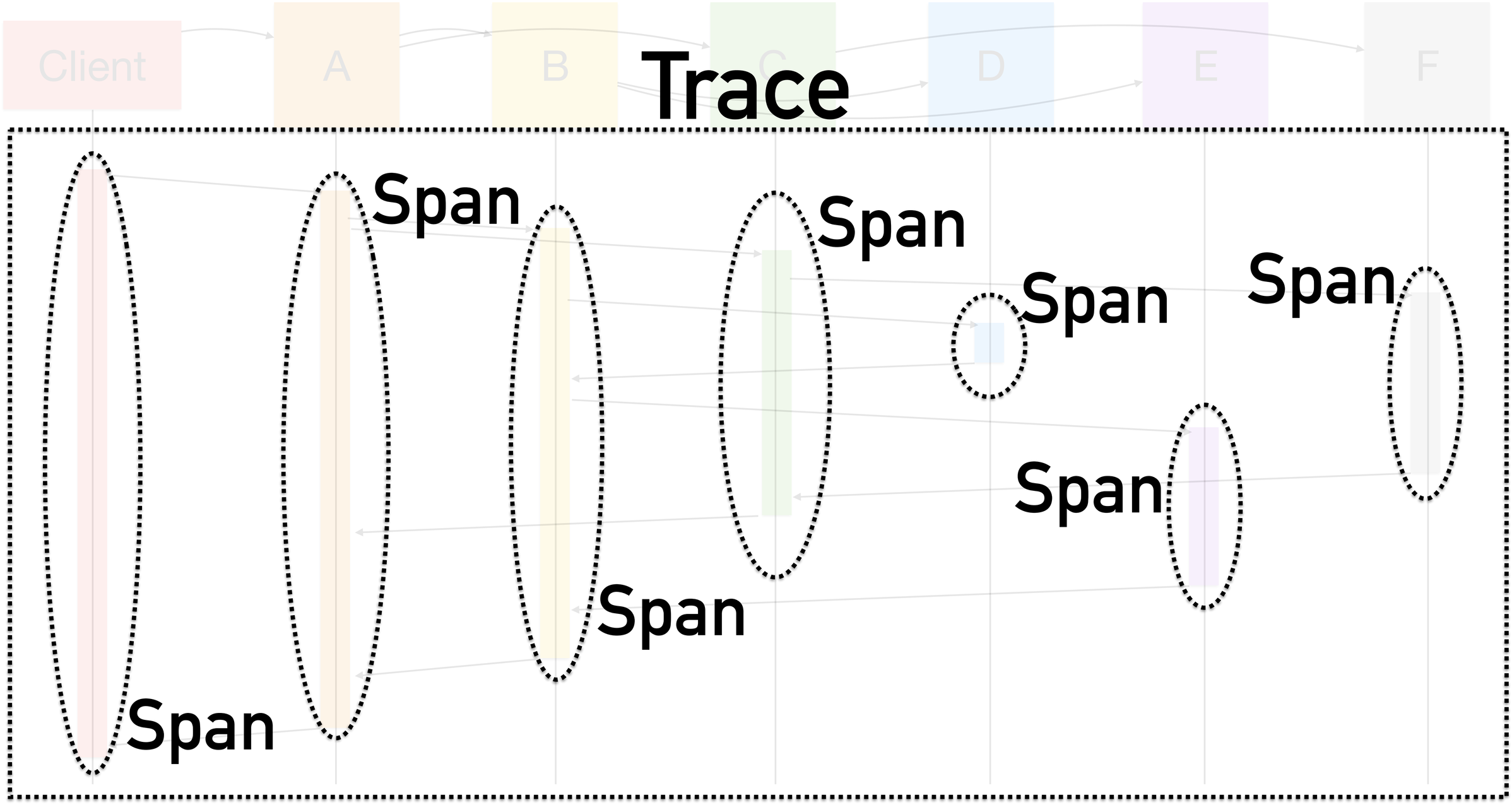

Client
A
B
C
D
E
F
+TraceID T1
+SpanID S1
=TraceID T1
+SpanID S2
~ParentSpanID S1
=TraceID T1
+SpanID S3
~ParentSpanID S2

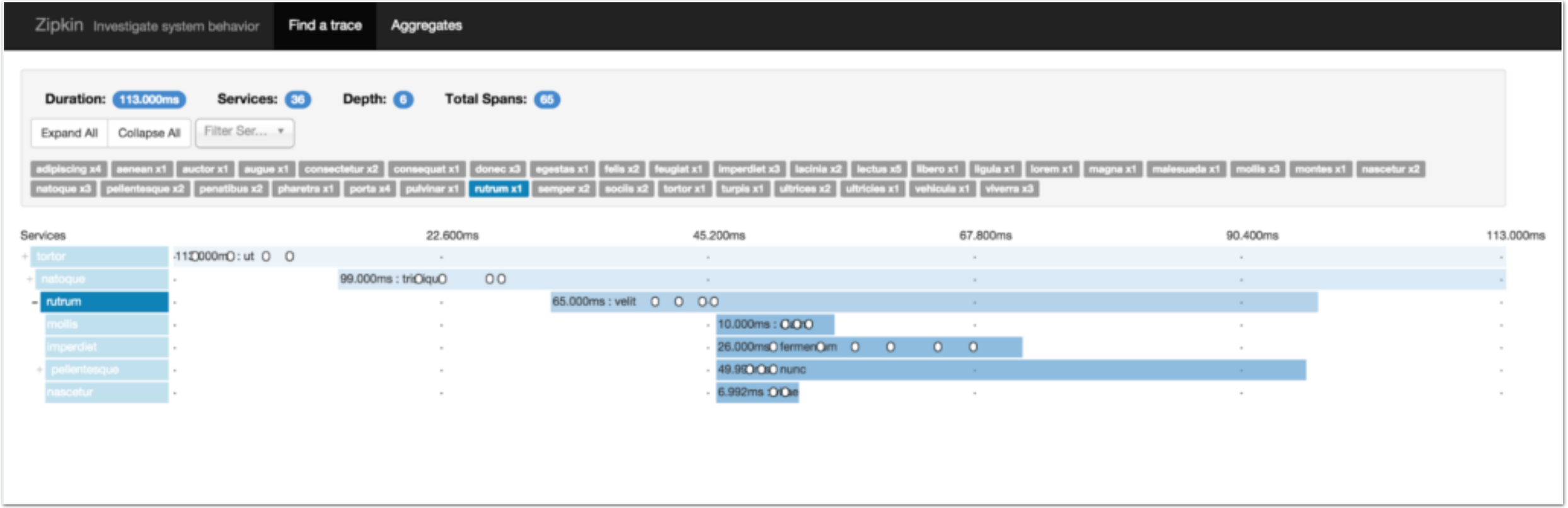

Client
A
B
C
D
E
F
Server Receive @ t=0
Client Send @ t=1
Client Receive @ t=2
Server Send @ t=3

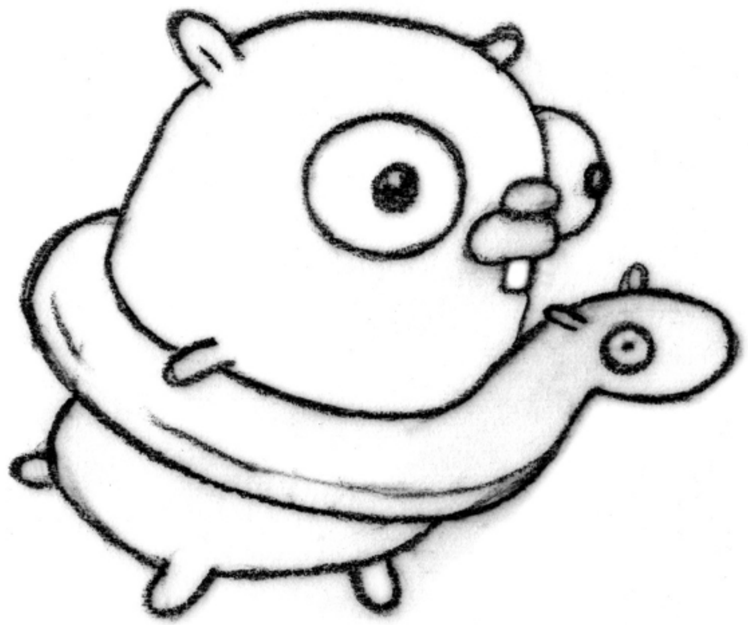

Client
A
B
C
D
E
F
Server Receive @ t=0
Span
Client Send @ t=1
Client Receive @ t=2
Server Send @ t=3


Client
A
B
C
D
E
F
Span

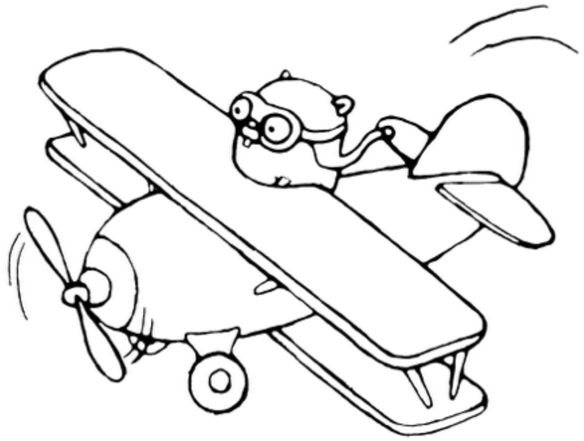

Client
A
B
C
D
E
F
Span
Span
Span
Span
Span
Span
Span


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


Zipkin
Investigate system behavior
Find a trace
Aggregates
Duration: 113.000ms
Services: 36
Depth: 6
Total Spans: 65
Expand All
Collapse All
Filter Ser...
adipiscing x4
aenean x1
auctor x1
augue x1
consectetur x2
consequat x1
donec x3
egestas x1
felis x2
feugiat x1
imperdiet x3
lacinia x2
lectus x5
libero x1
ligula x1
lorem x1
magna x1
malesuada x1
mollis x3
montes x1
nascetur x2
natoque x3
pellentesque x2
penatibus x2
pharetra x1
porta x4
pulvinar x1
rutrum x1
semper x2
sociis x2
tortor x1
turpis x1
ultrices x2
ultricies x1
vehicula x1
viverra x3
Services
22.600ms
45.200ms
67.800ms
90.400ms
113.000ms
tortor
natoque
rutrum
mollis
imperdiet
pellentesque
nascetur
99.000ms : tri
65.000ms : velit
10.000ms :
26.000ms
49.9
nunc
6.992ms :

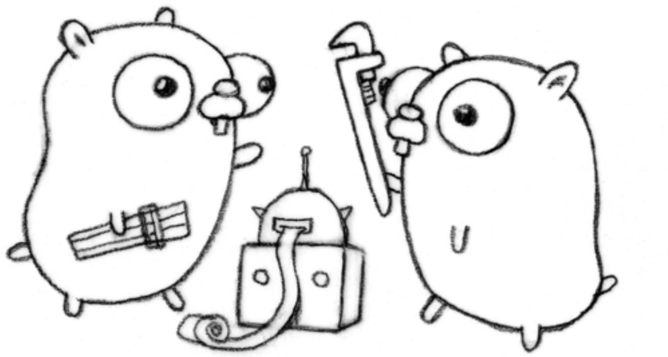


git apply



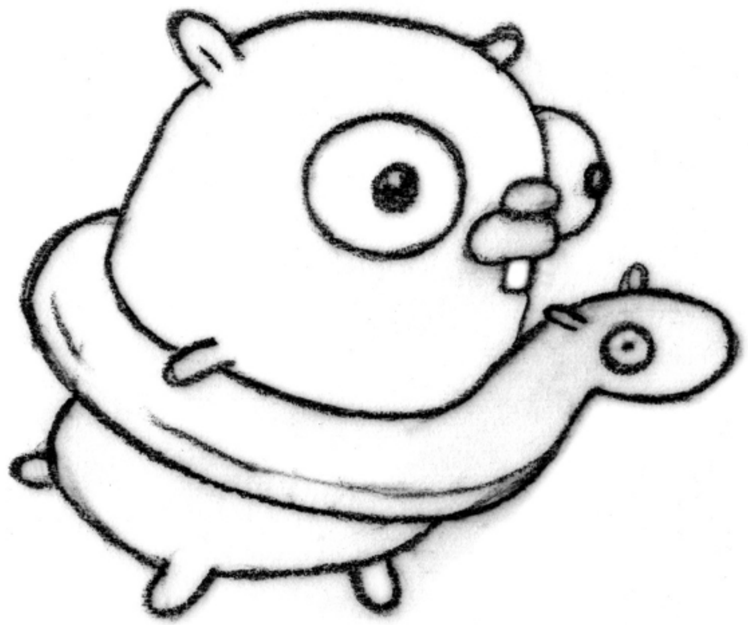
Challenge

Actually get it working :(



addsvc

Continuous integration with CircleCI



git apply

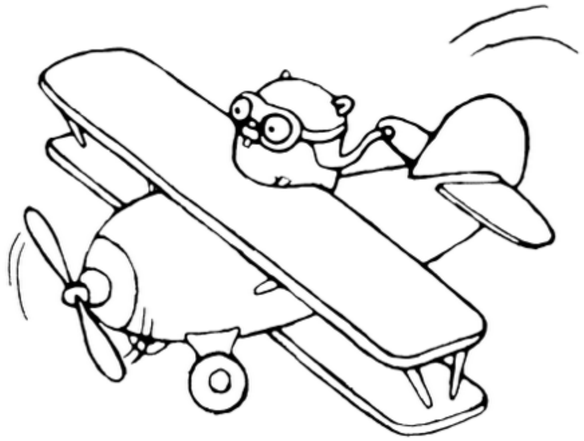
21, 22

CircleCI.com

Project settings

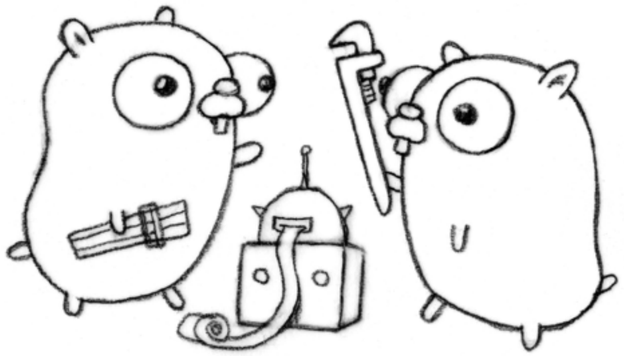
circle.yml

and Makefile and Dockerfile



Challenge

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



addsvc

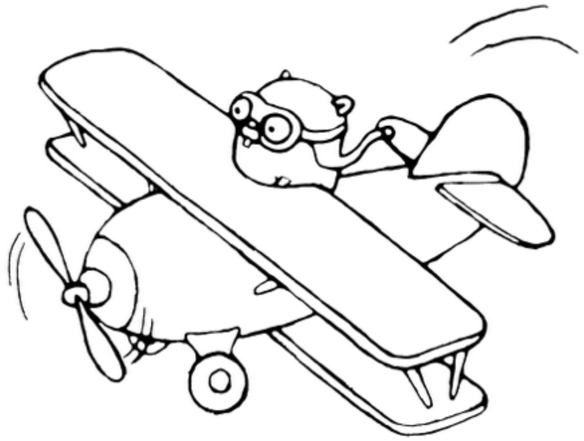
Cloud-native deployment with Kubernetes

Kubernetes architecture

And minikube local cluster

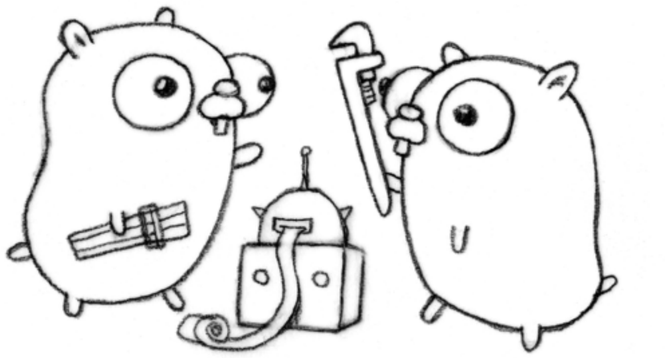
addsvc-*.yaml

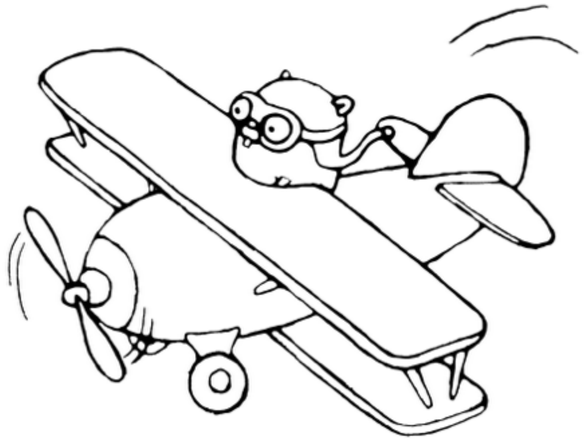
Resource definition files



Challenge

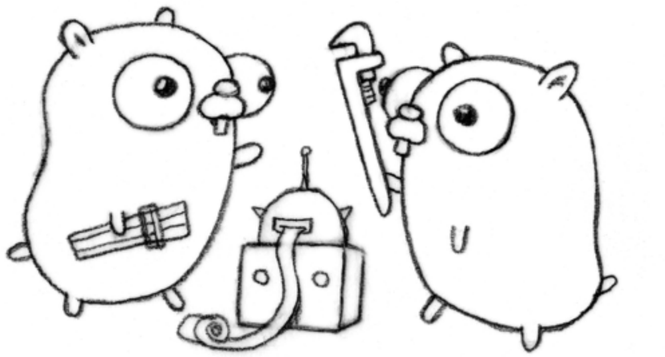
Deploy a Kubernetes cluster, and your own addsvc





Challenge

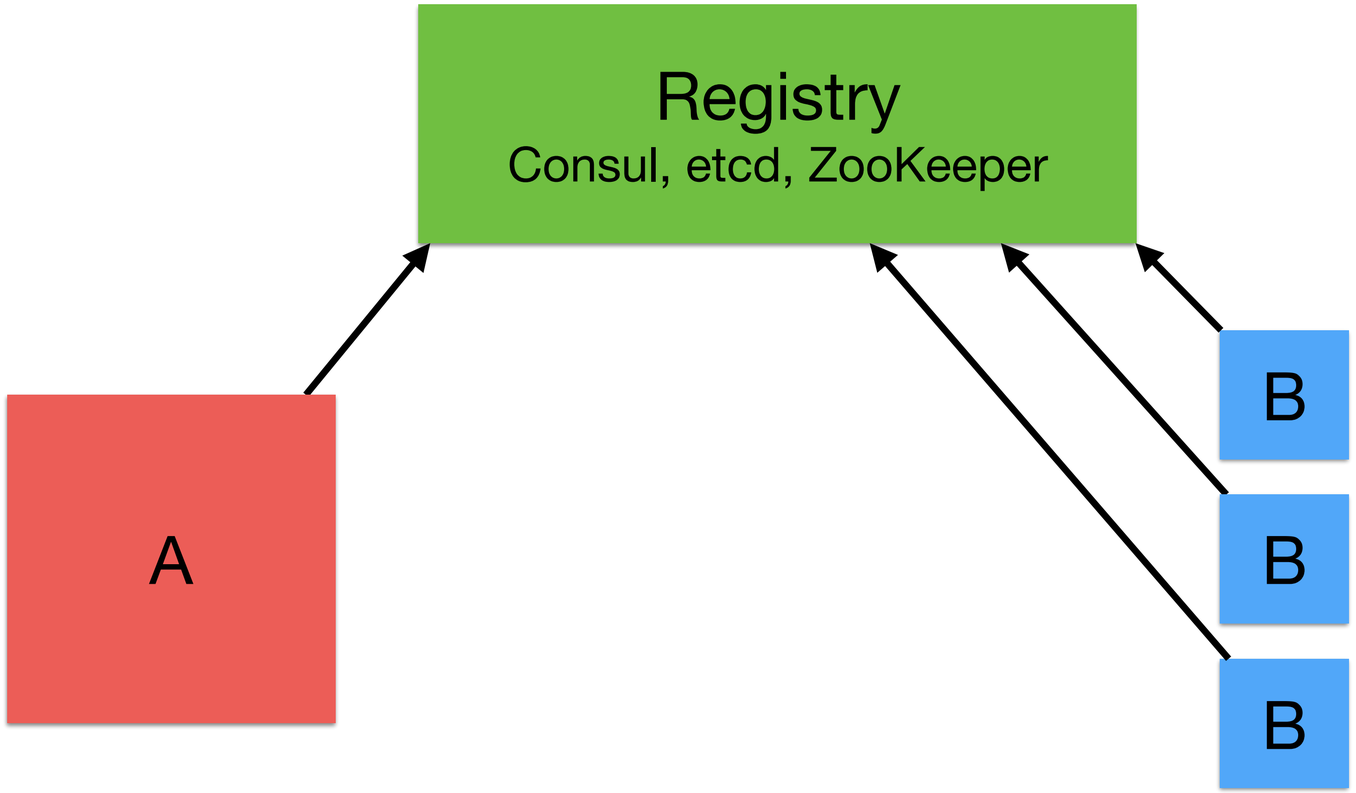
Deploy stringsvc, and have addsvc call it

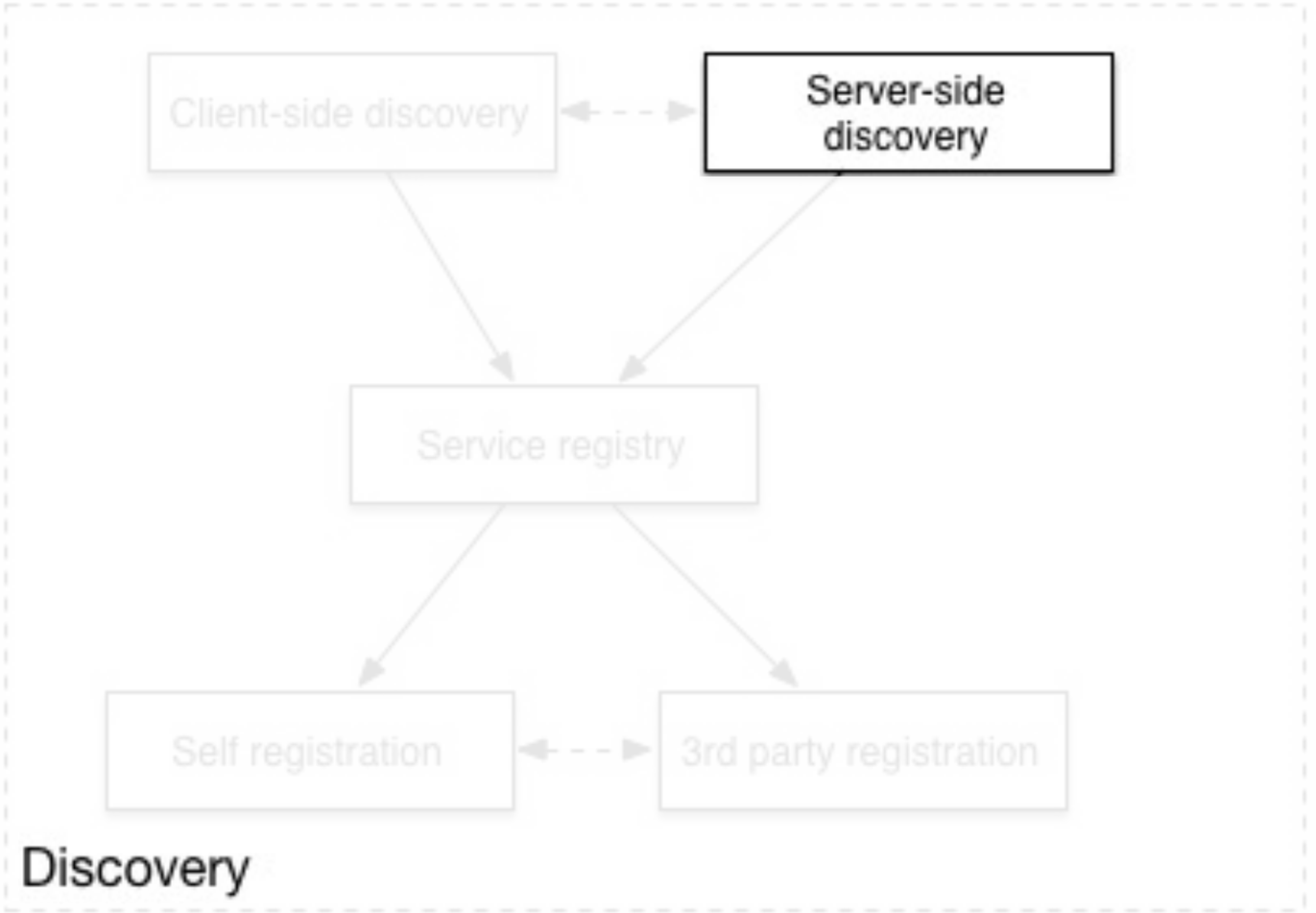


package sd

Service discovery and client-side load balancing

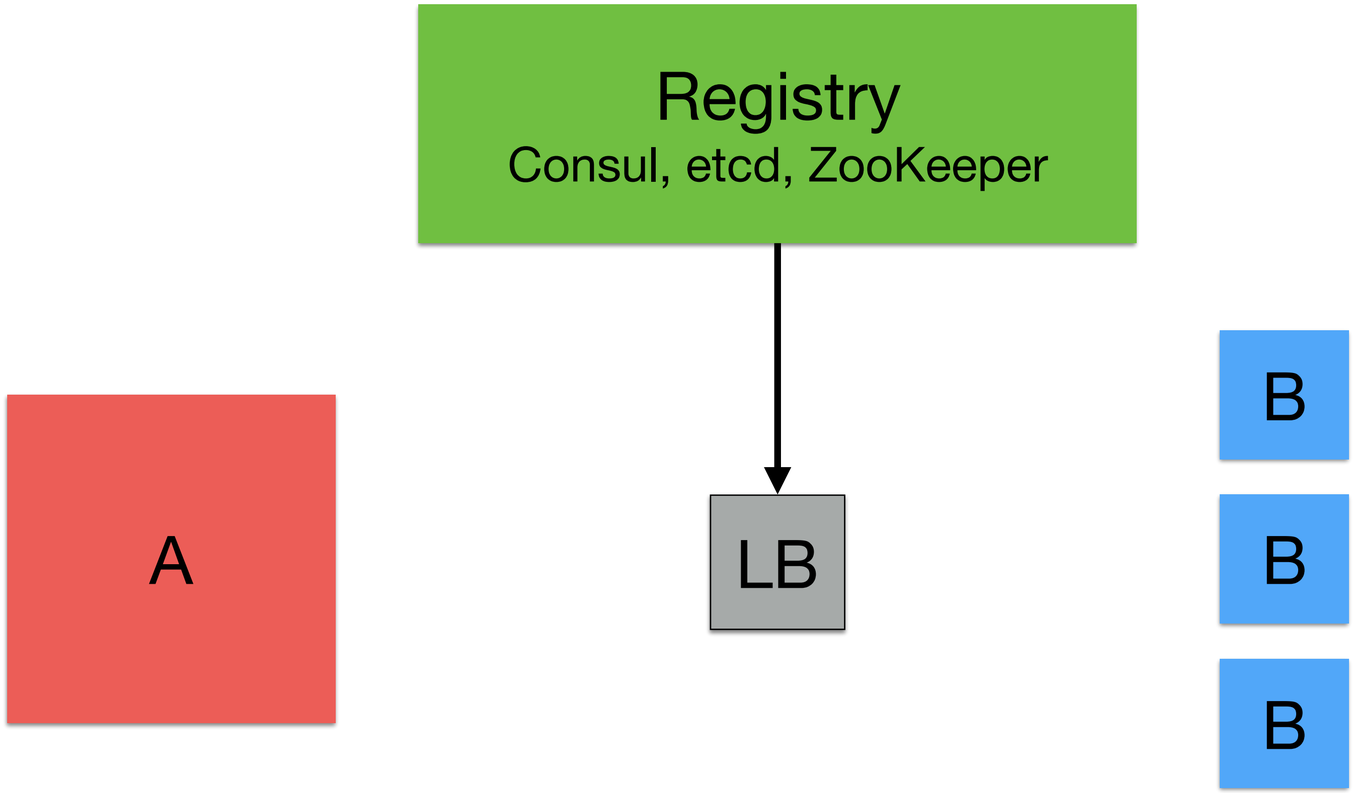
Registration



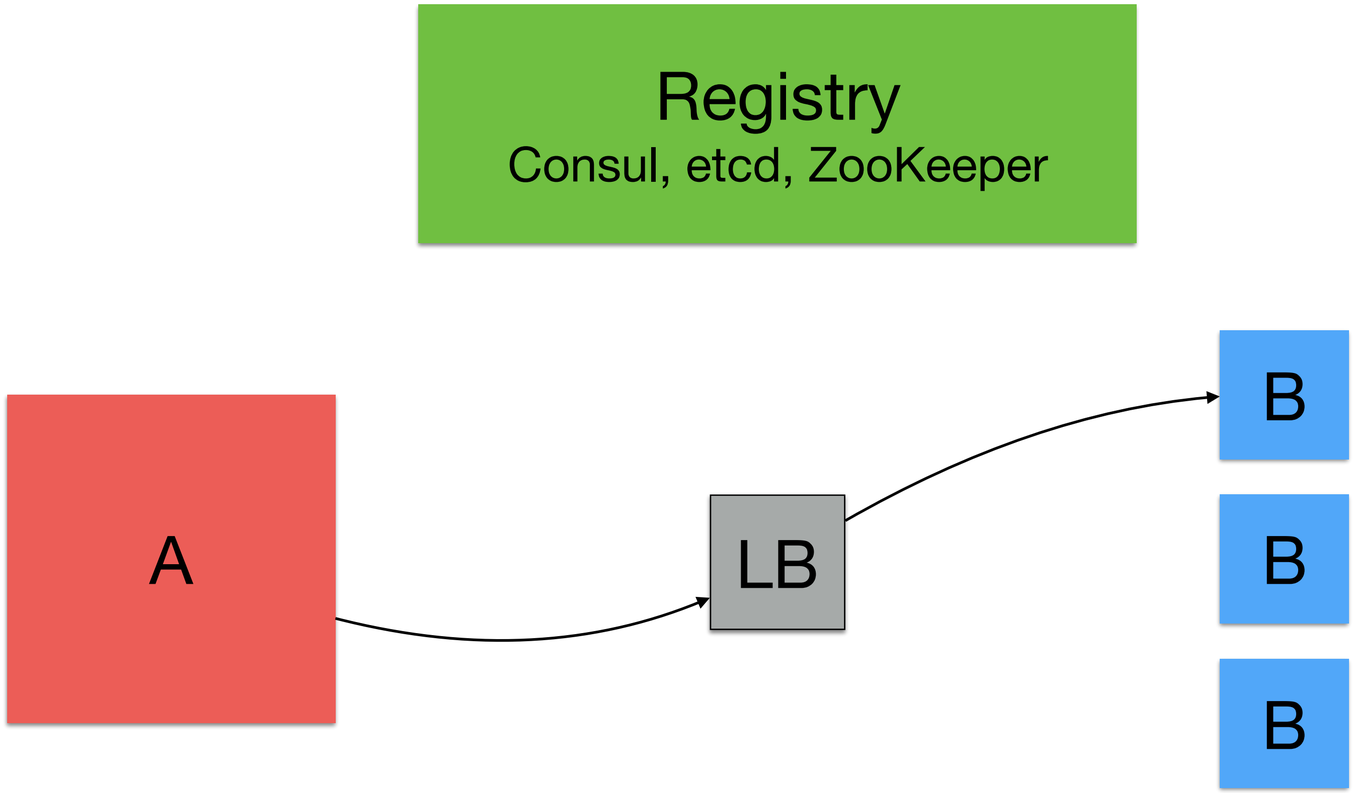


microservices.io/patterns/index.html

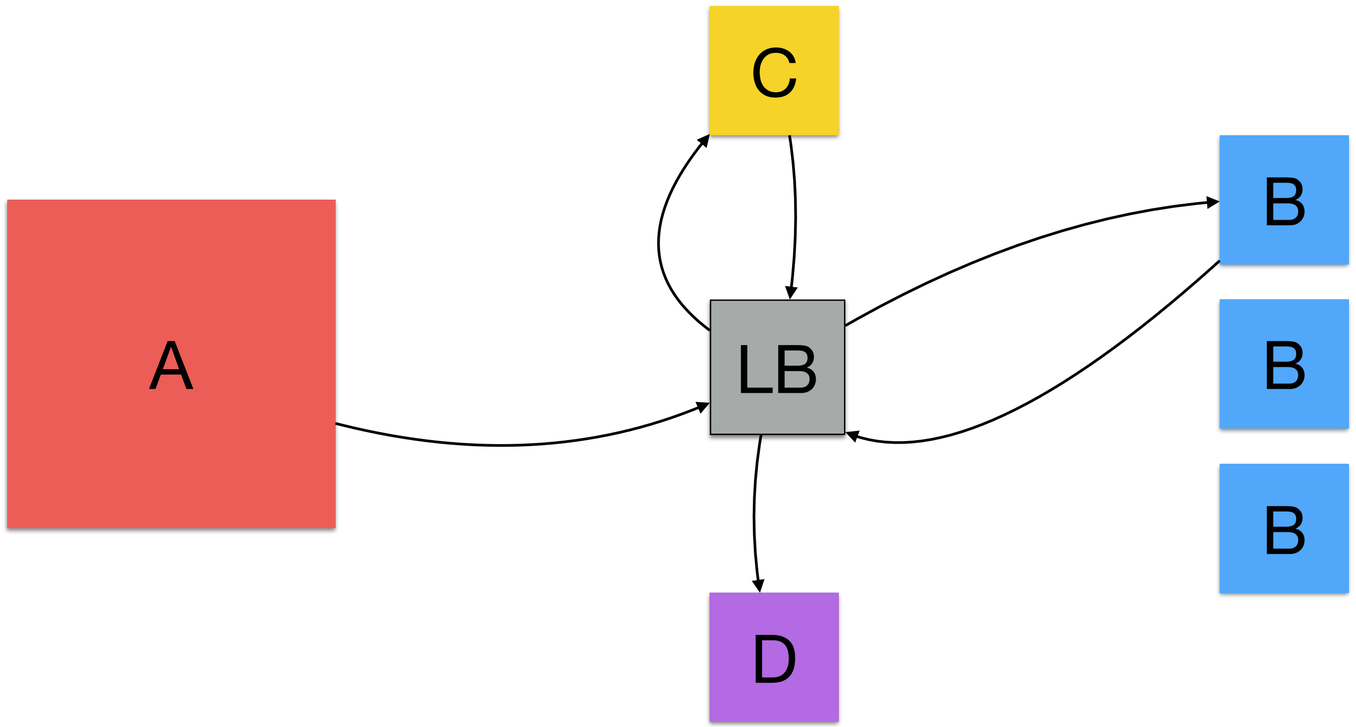
Central load balancer



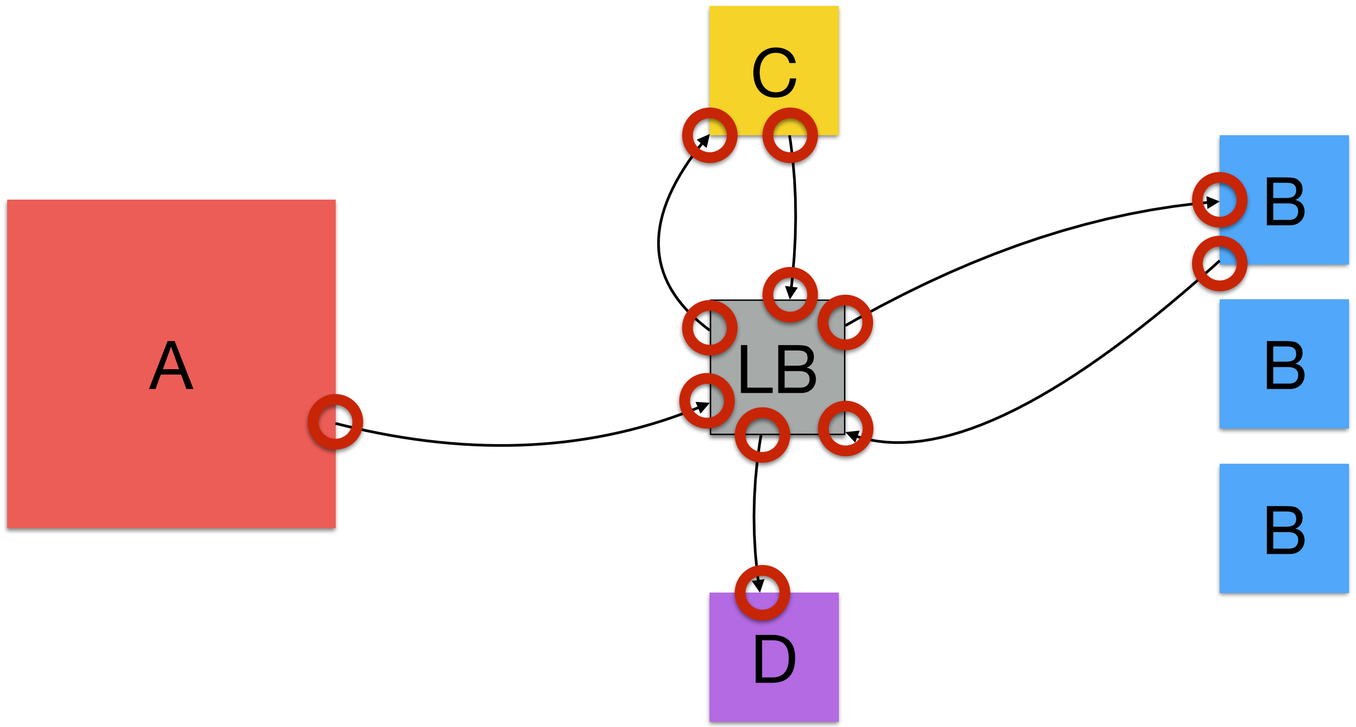
Central load balancer



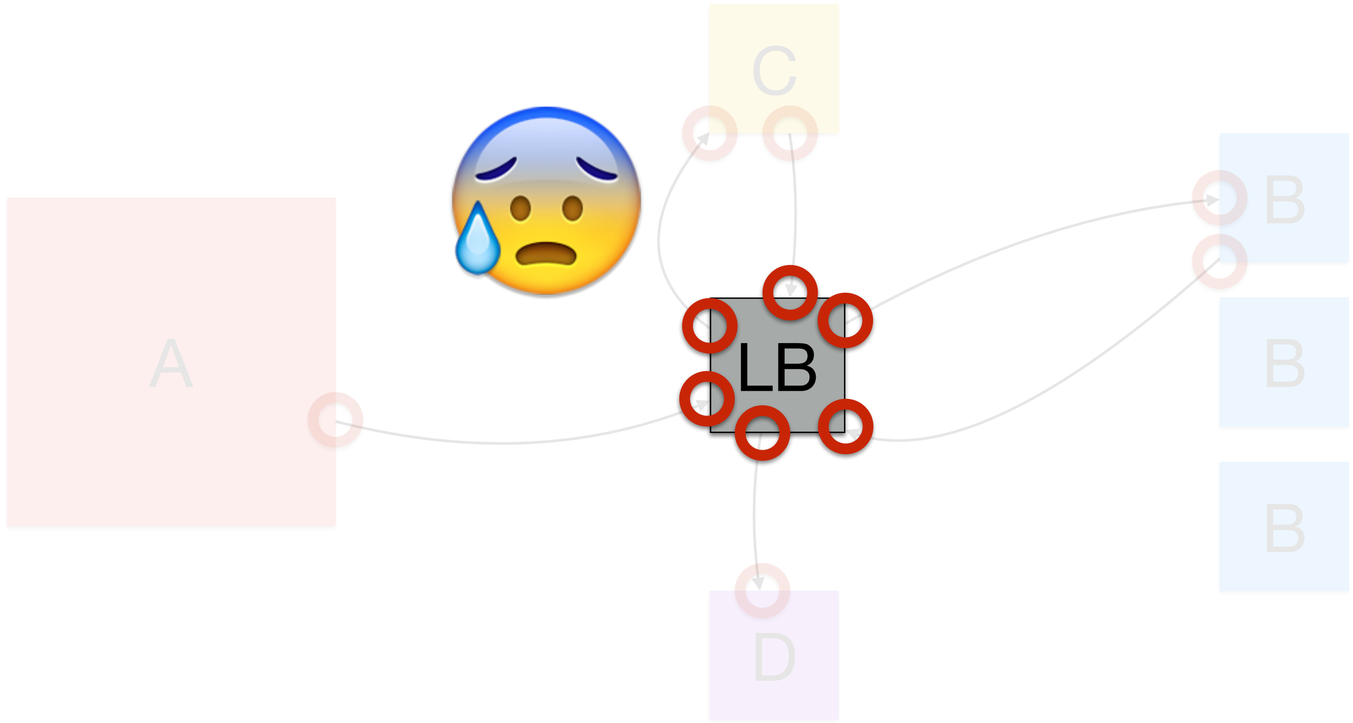
Central load balancer

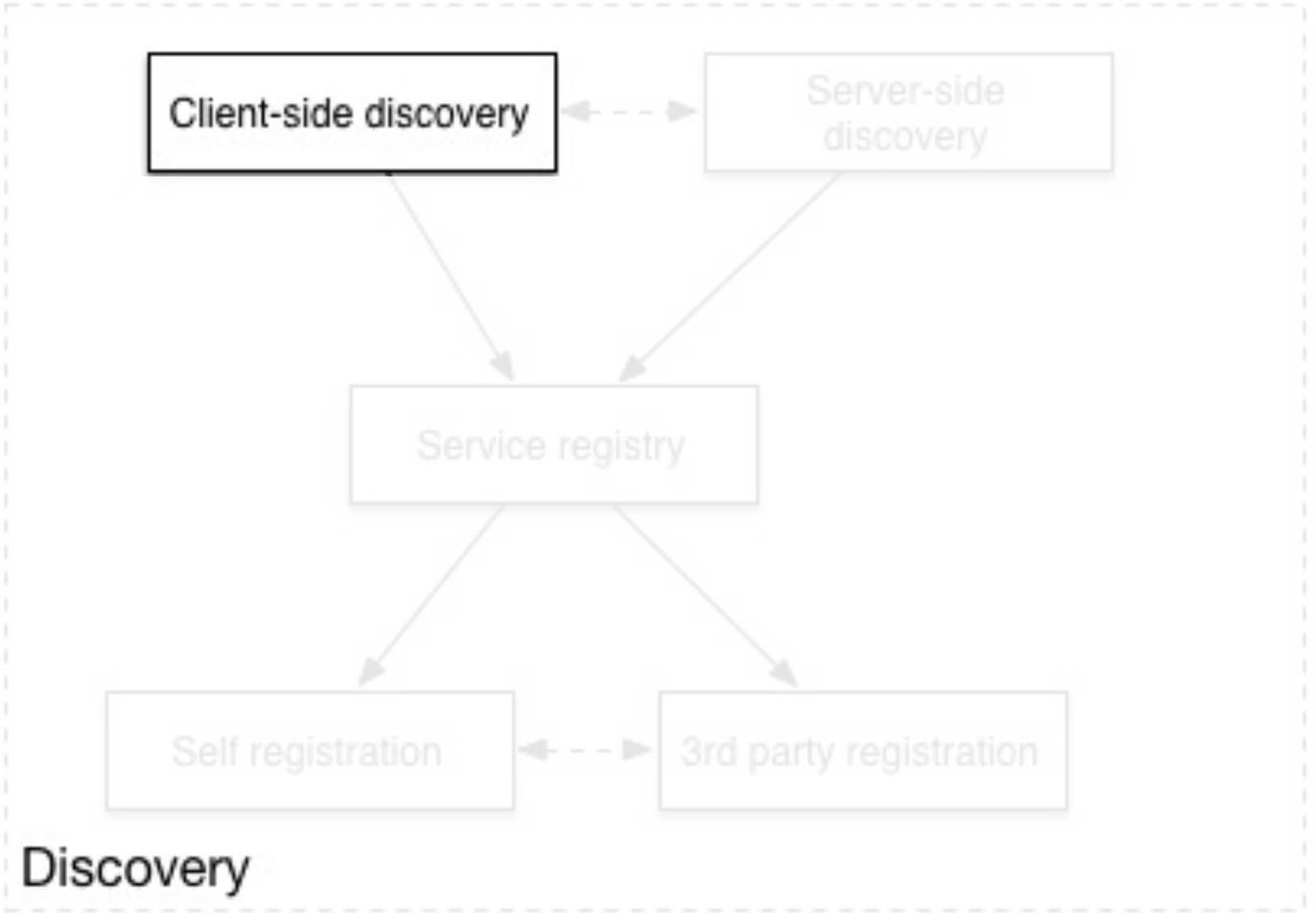


Central load balancer



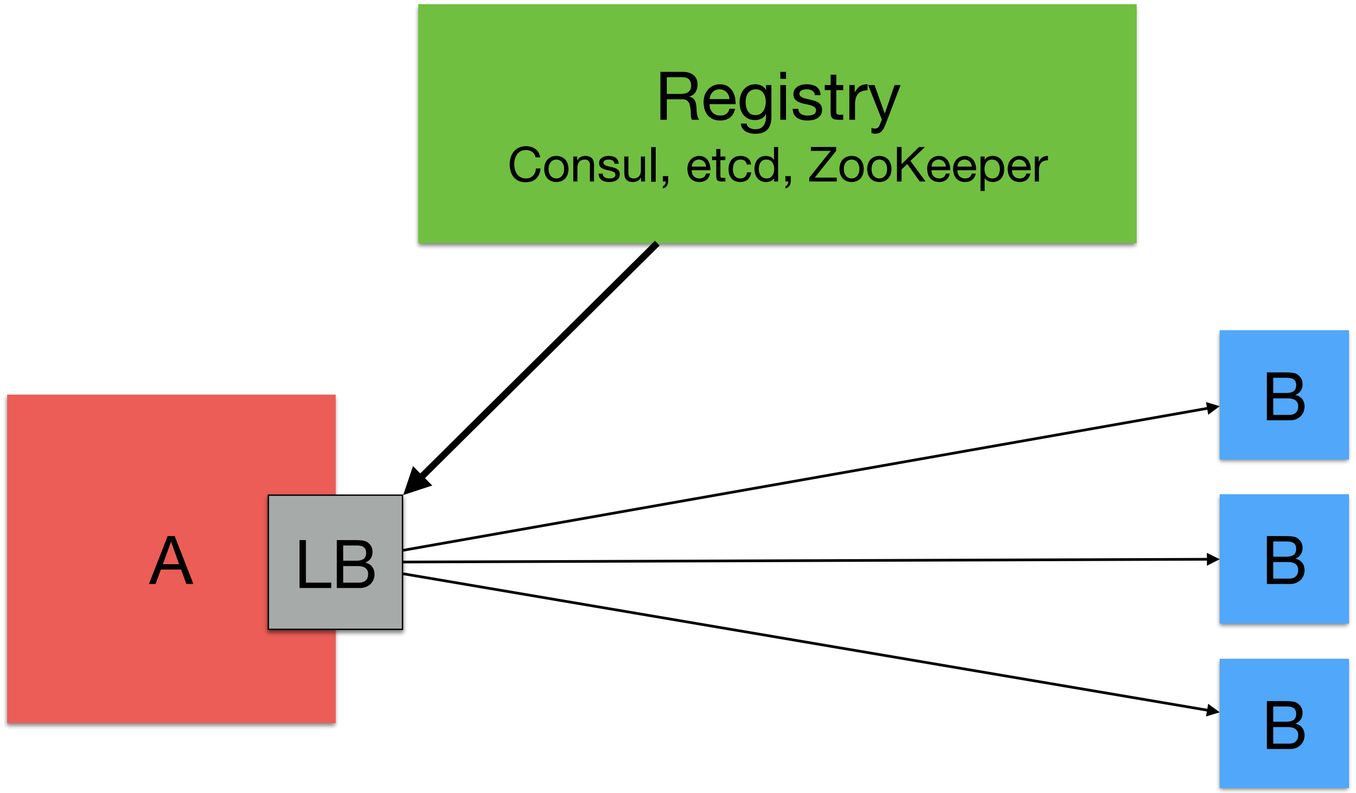
Central load balancer



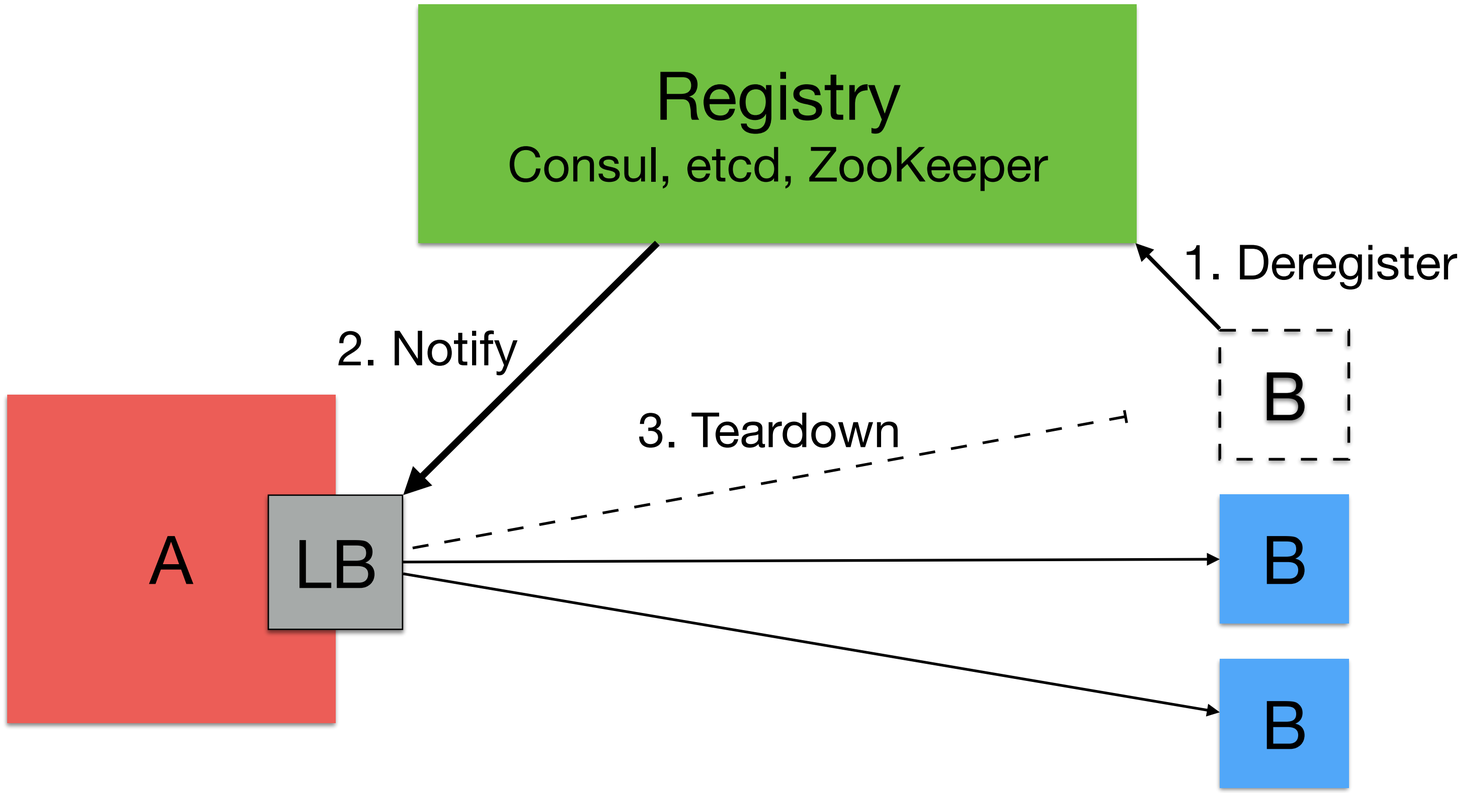


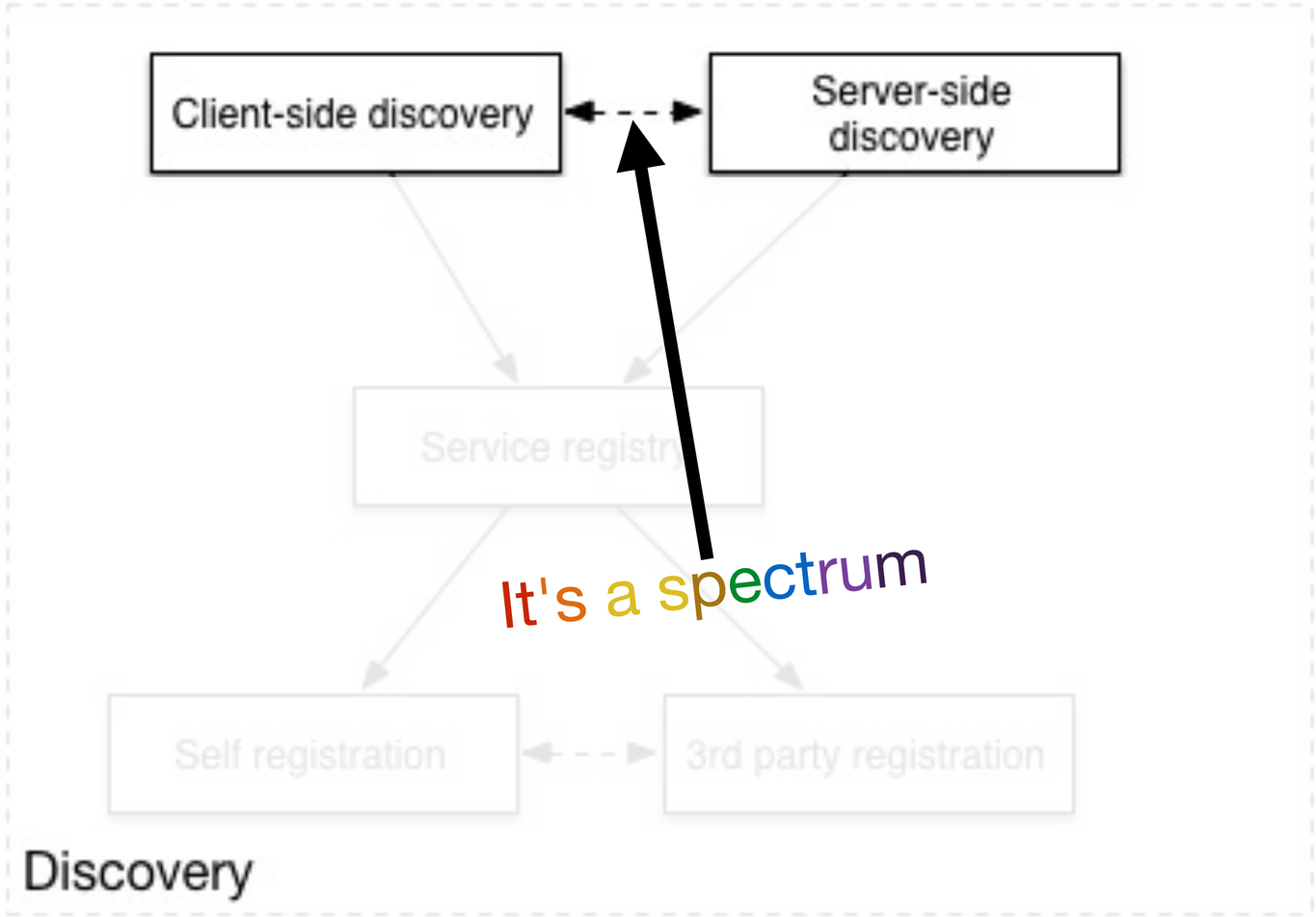
microservices.io/patterns/index.html

Client-side load balancer



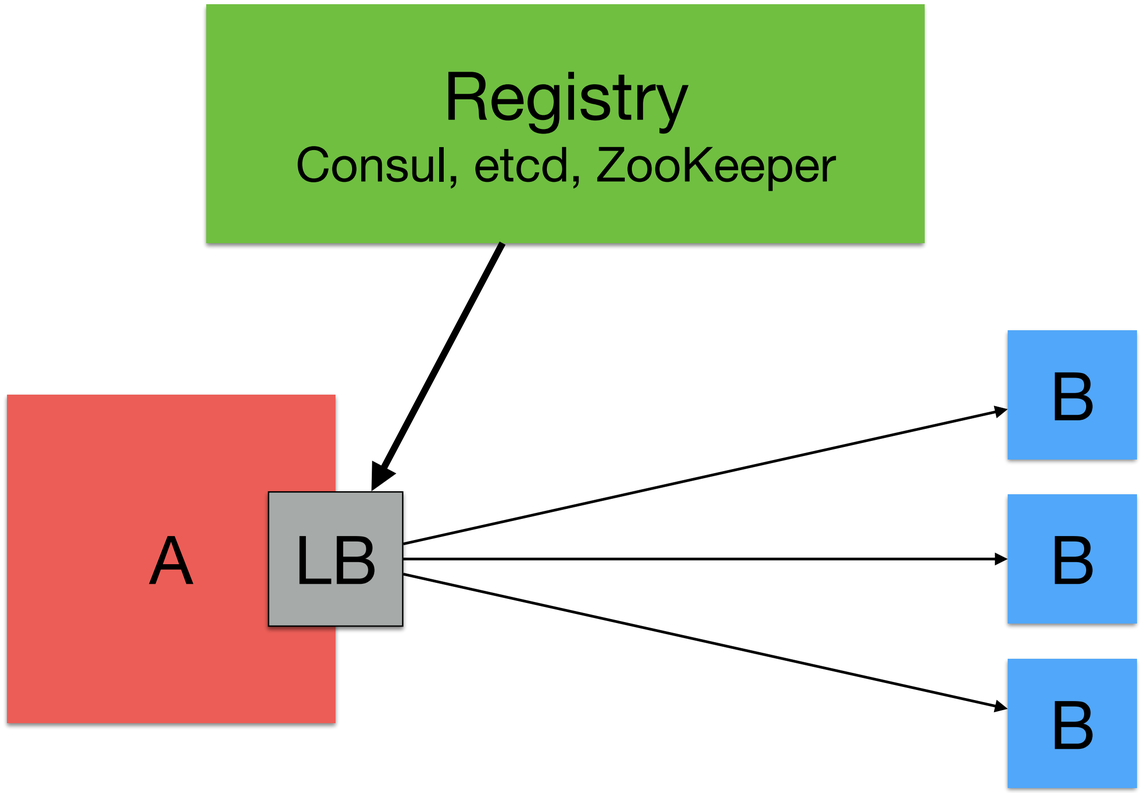
Deregistration



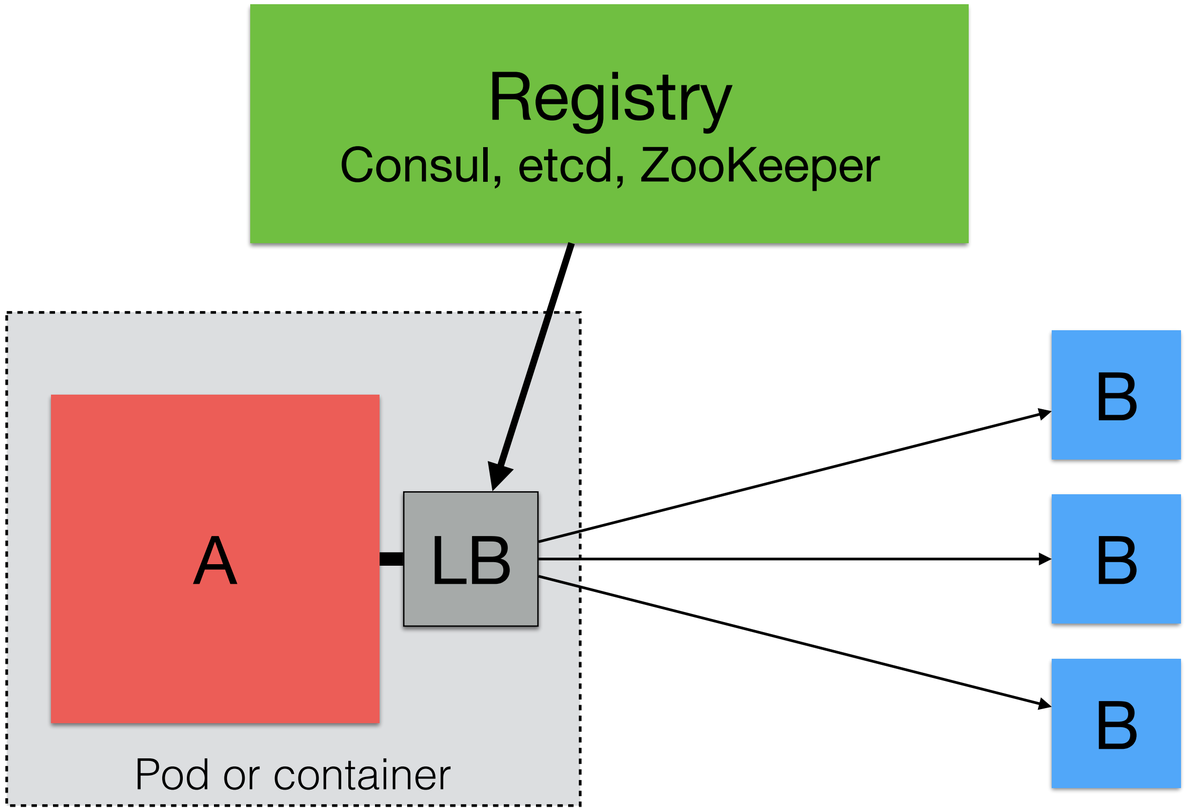


microservices.io/patterns/index.html

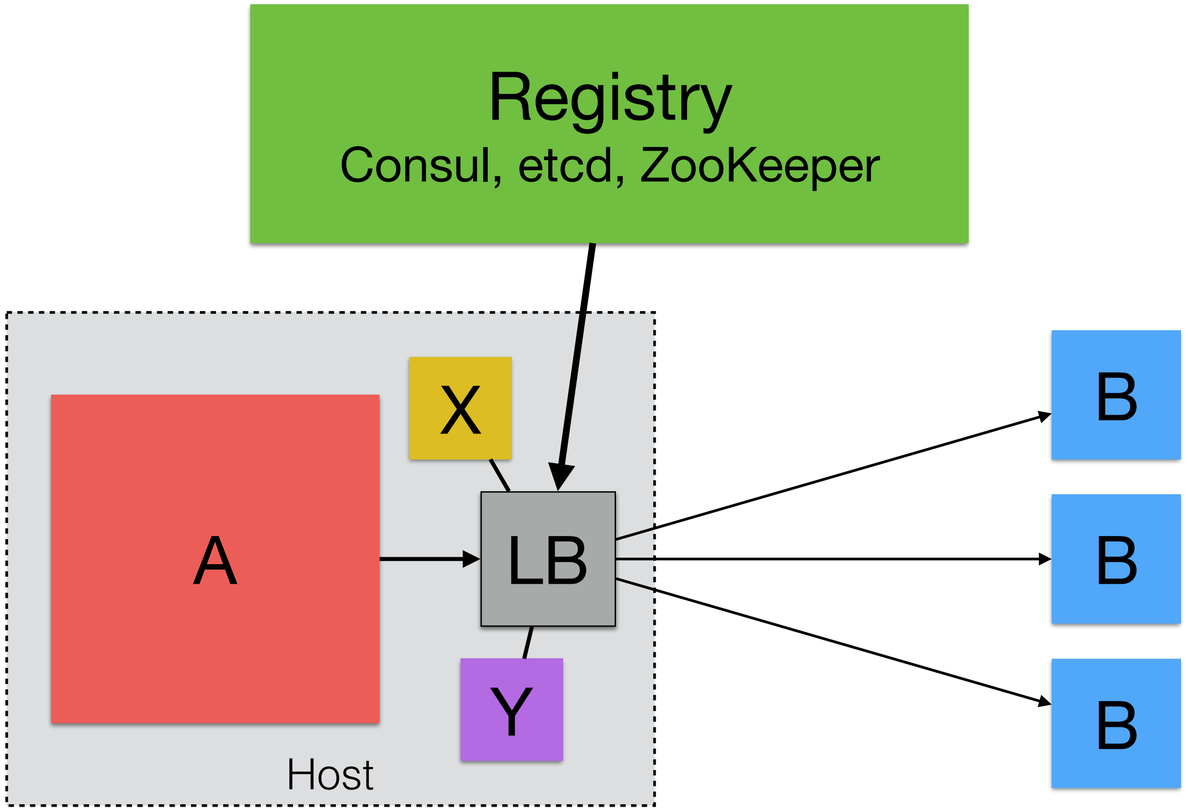
Client-side



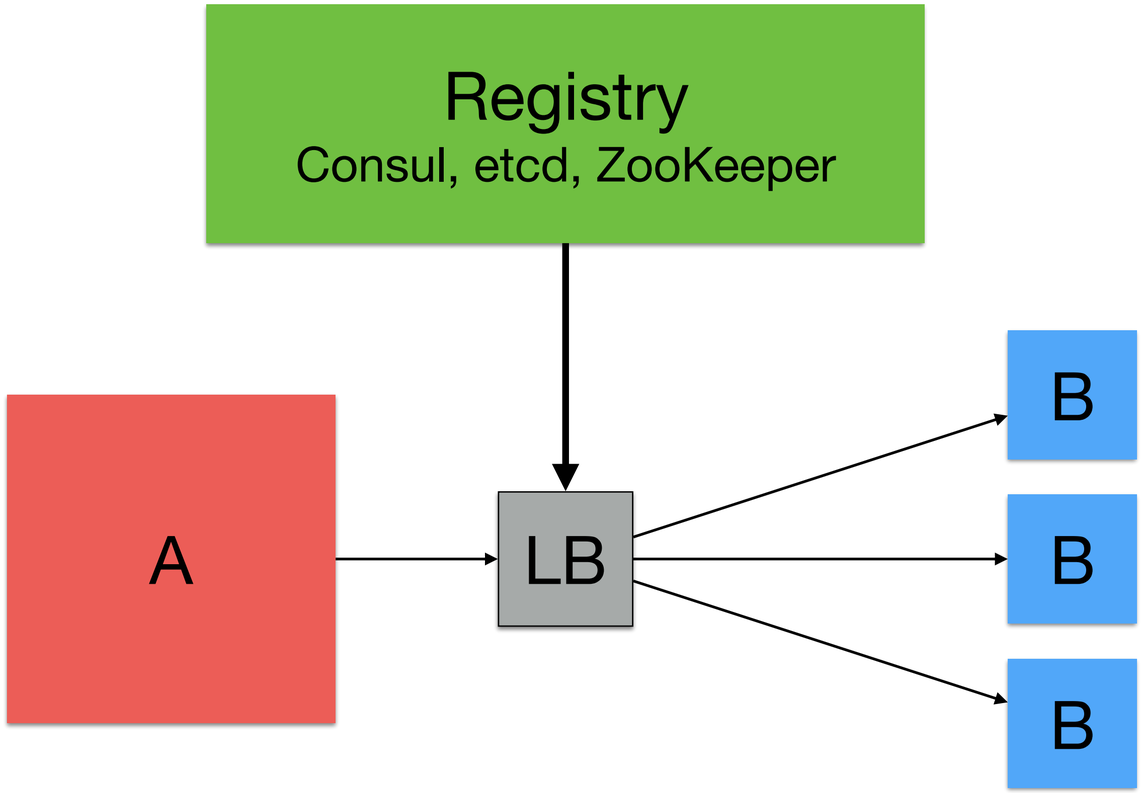
Linkerd sidecar

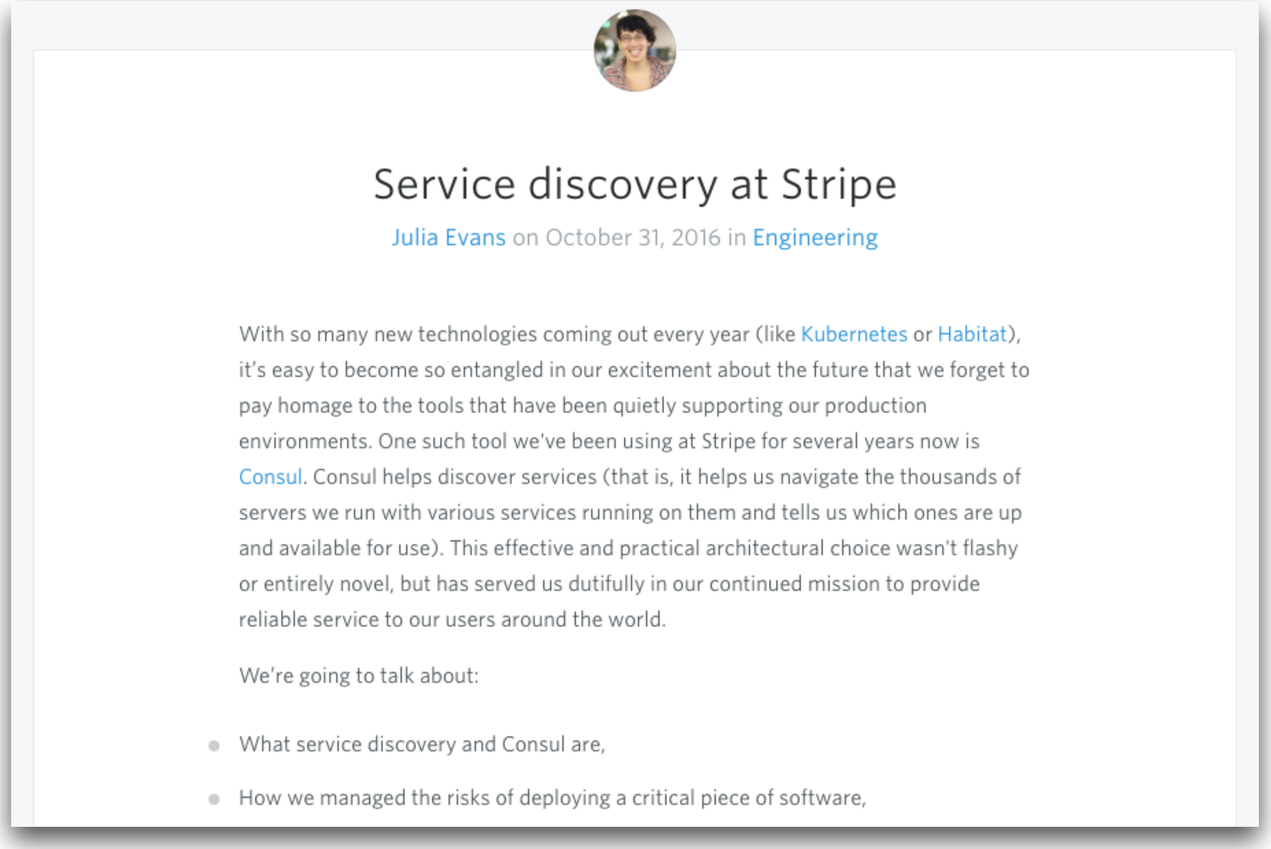


Per-host load balancers



Central loadbalancer





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

Your microservices

Custom projects

Thanks! Hooray!

October/November 2016 · Groupon