Go microservices

QCon NYC · 16 June 2016

Who are you??

Go

Purpose-built for writing servers in big teams;
seemingly a **perfect match** for microservices.

Microservices

Solve organizational problems &
Cause technical problems

Definitions

Size

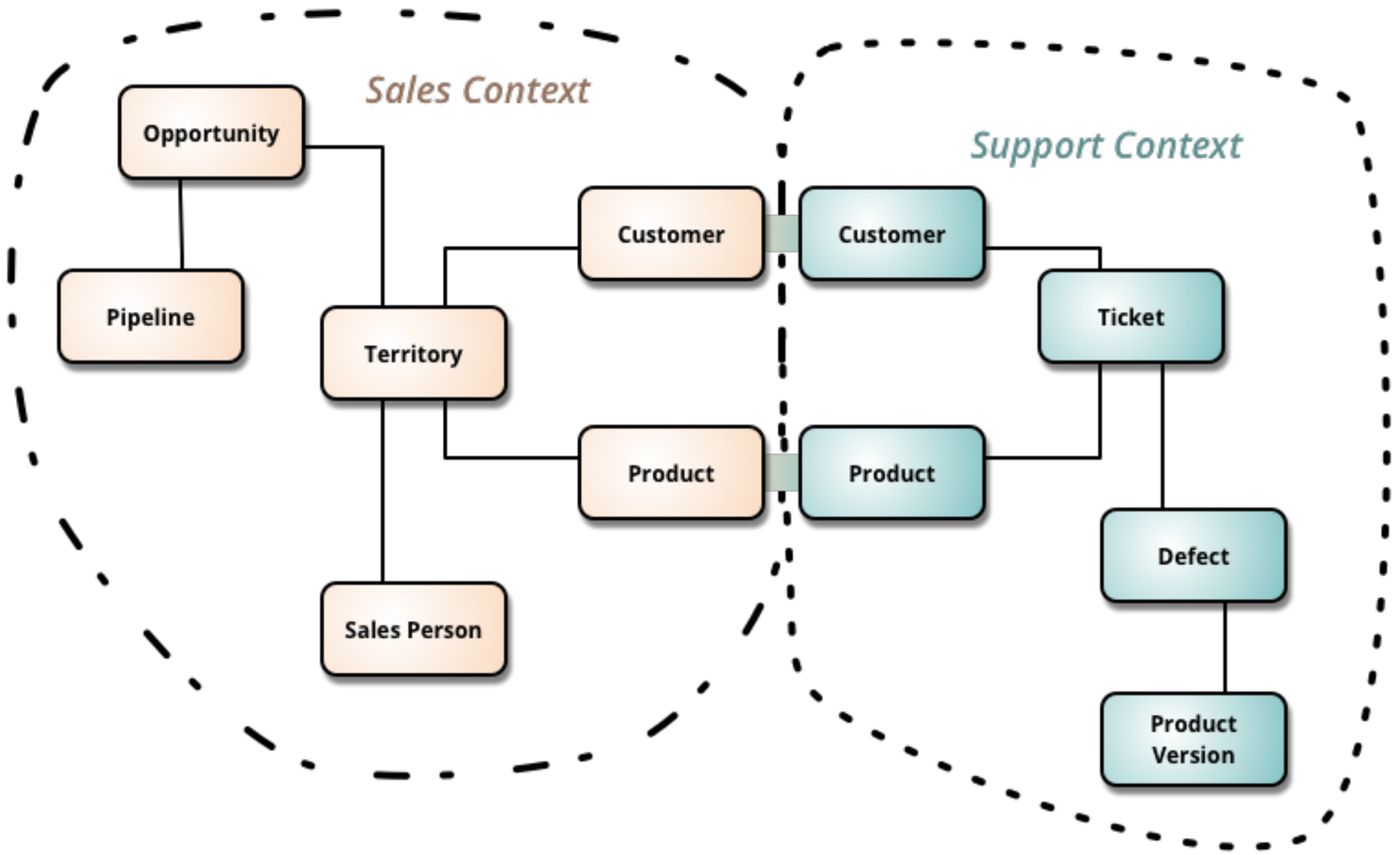
A **single programmer** can design, implement, deploy and maintain a microservice.
—Fred George

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*

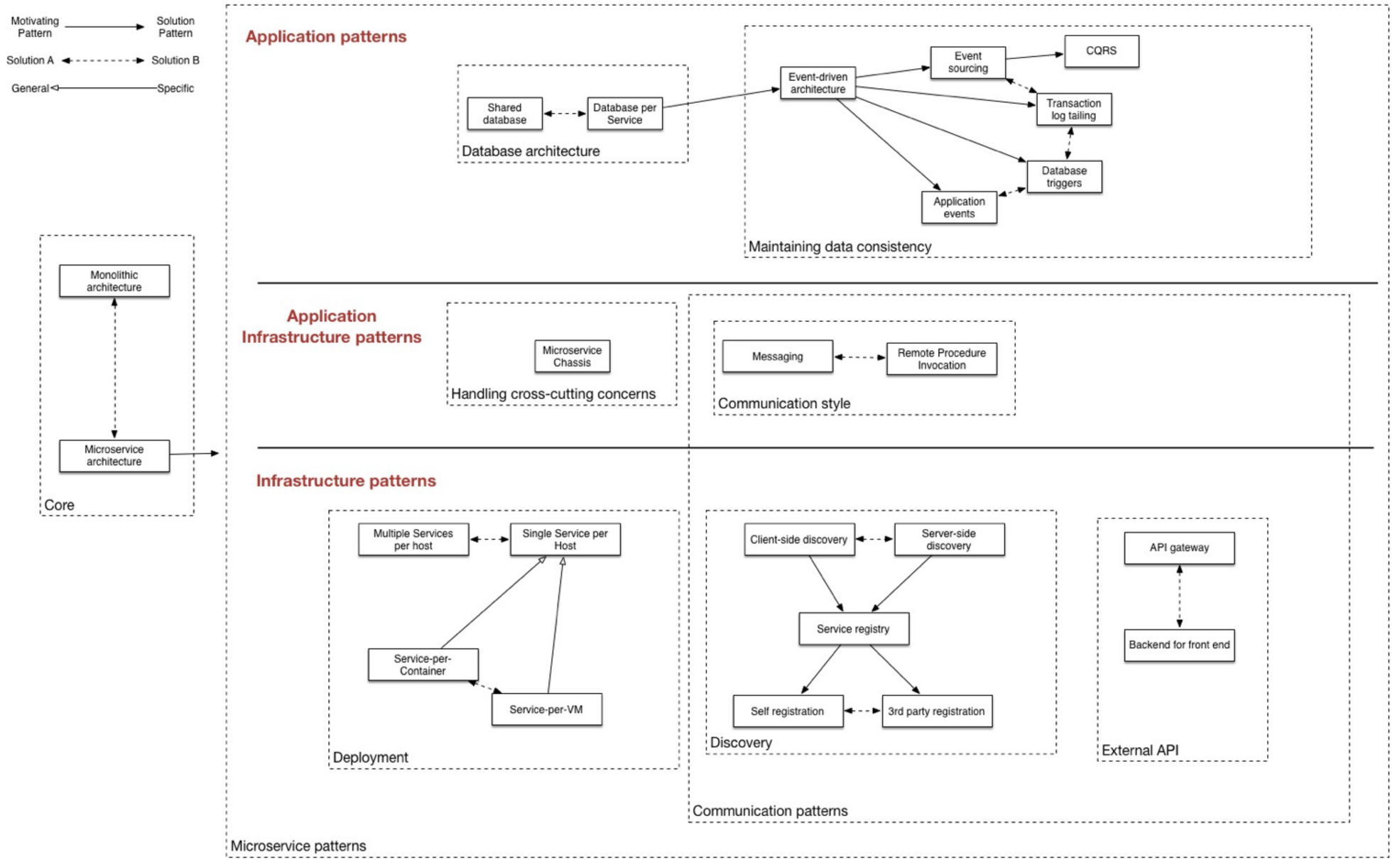
Addressable through a **service discovery** system.
—*Chris Richardson*

The landscape

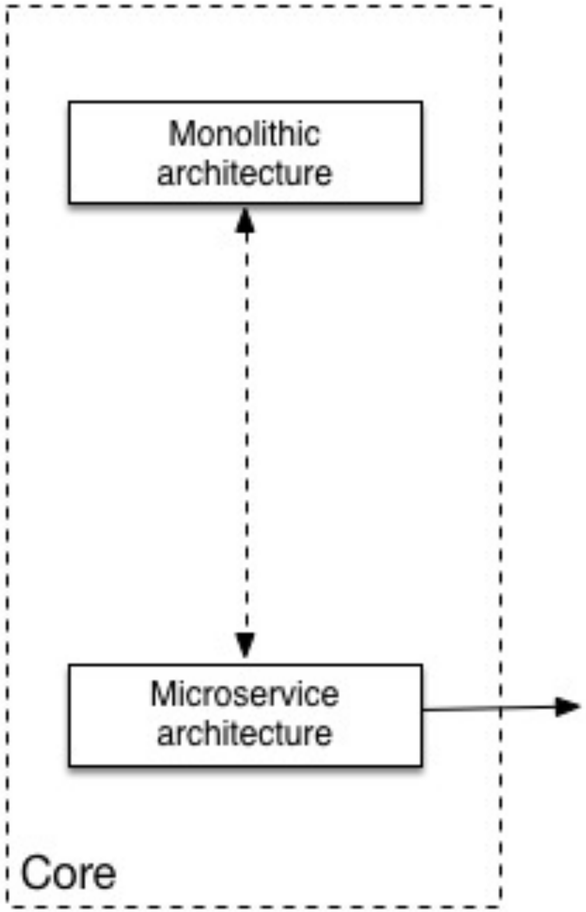
What's a pattern?

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

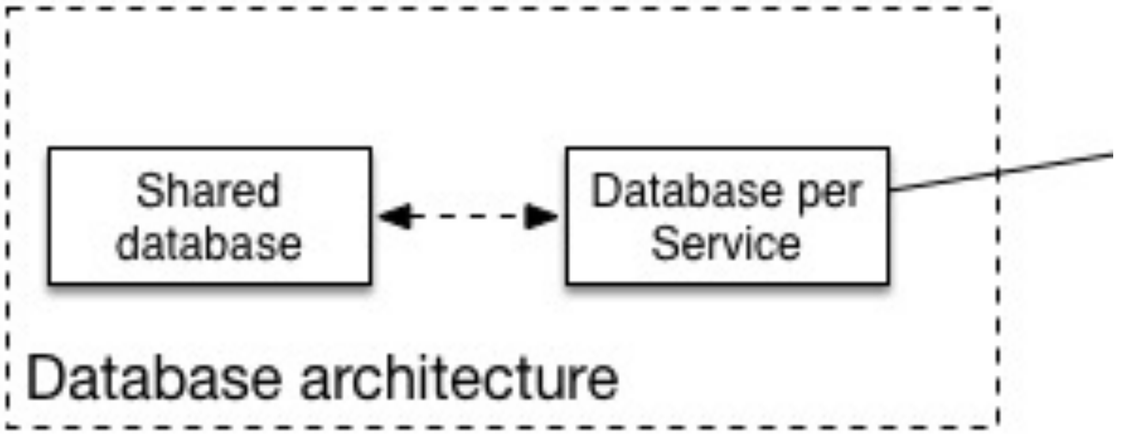
slideshare.net/chris.e.richardson/microservices-pattern-language-microxchg-microxchg2016



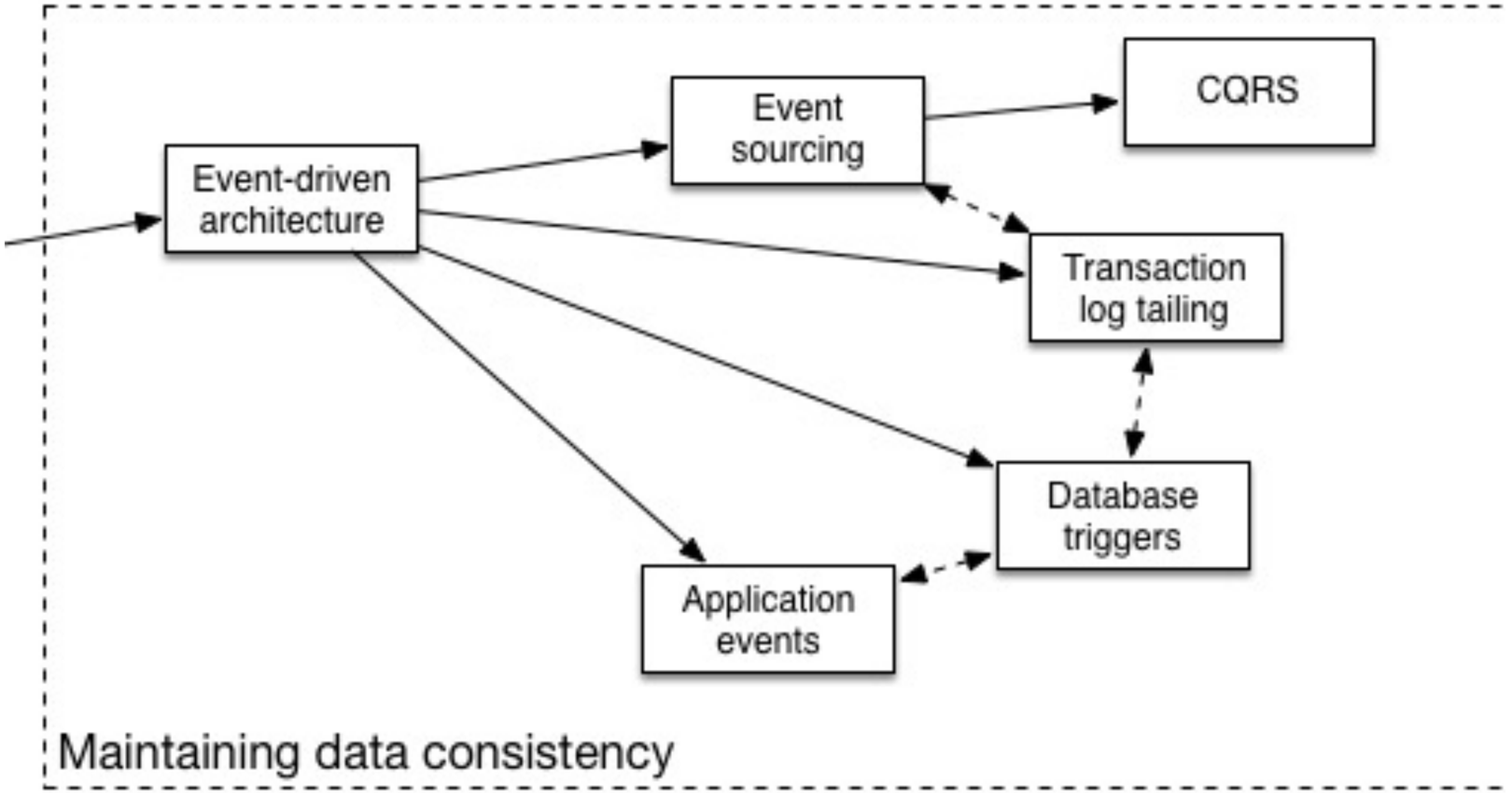
microservices.io/patterns/index.html



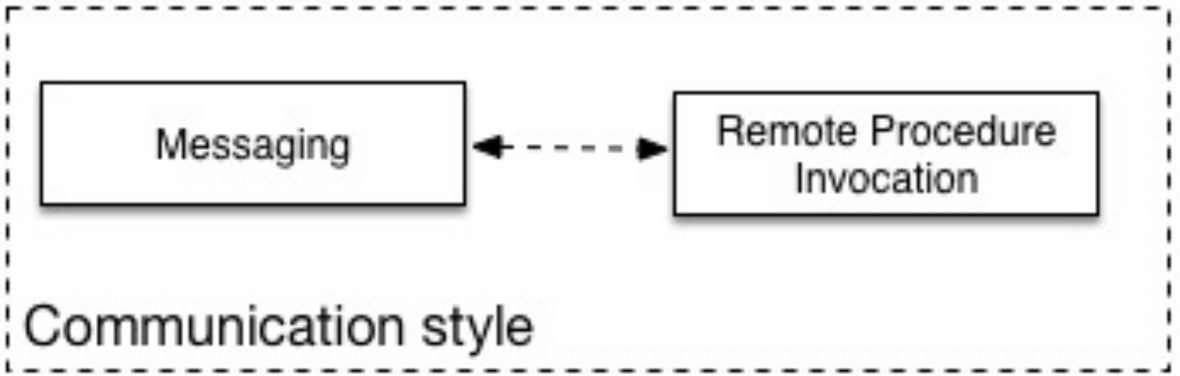
microservices.io/patterns/index.html



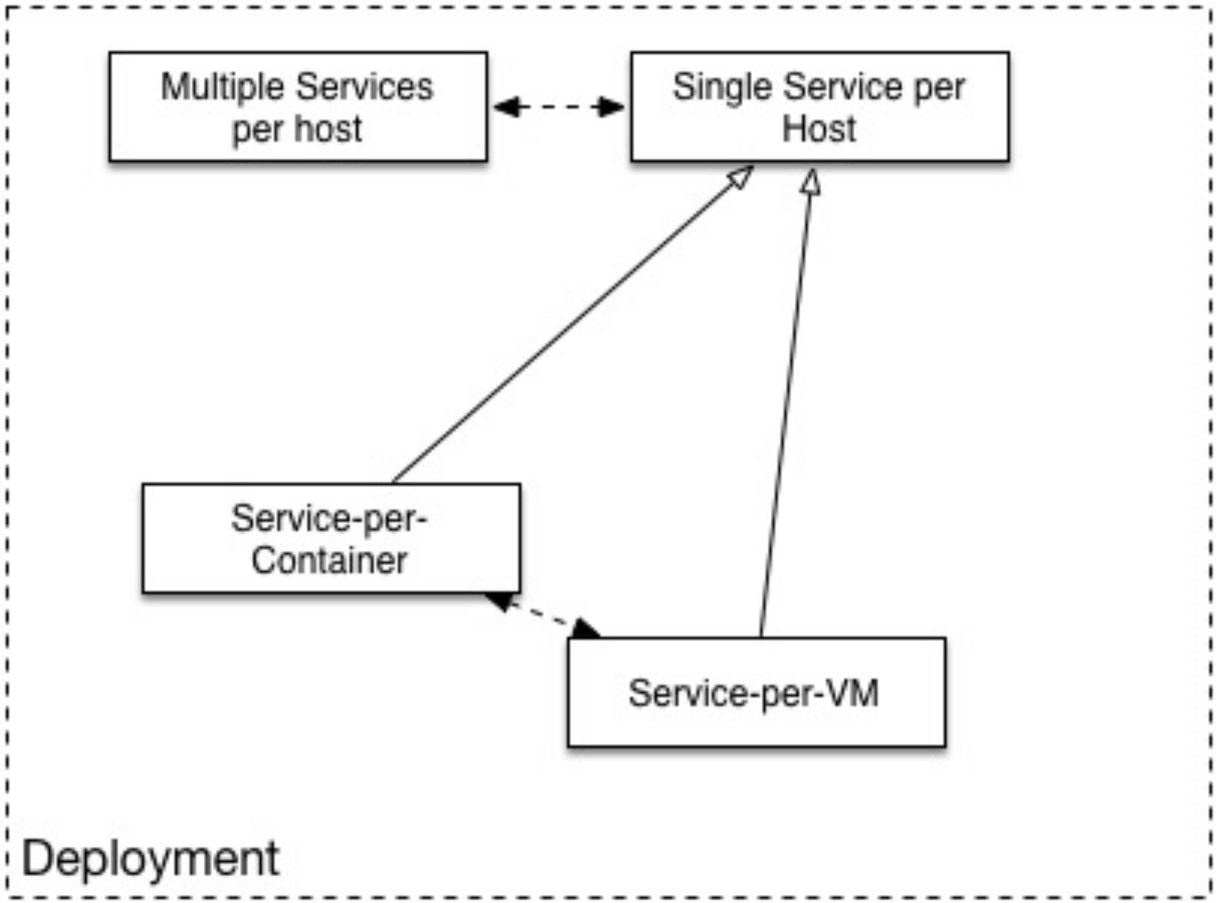
microservices.io/patterns/index.html



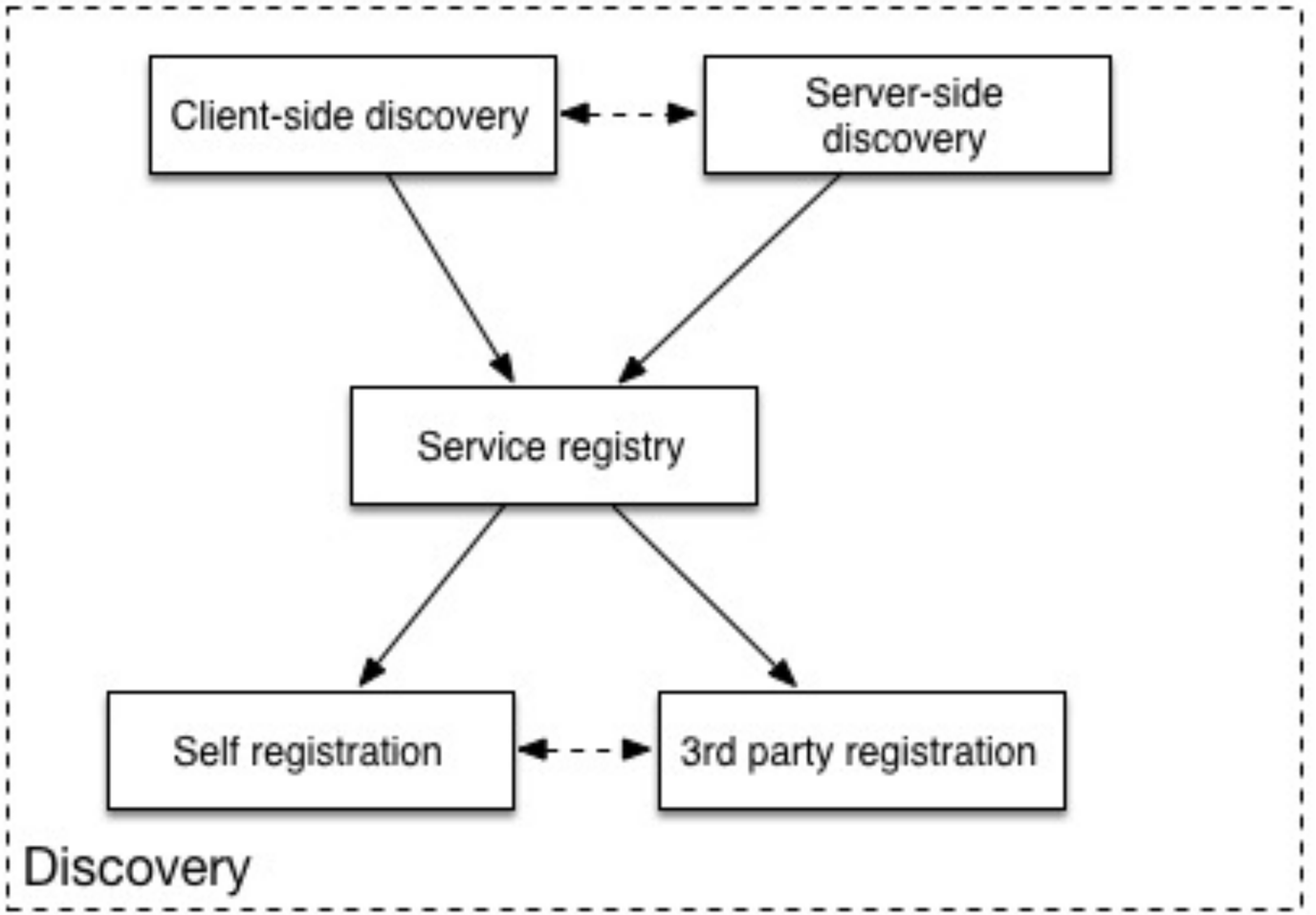
microservices.io/patterns/index.html



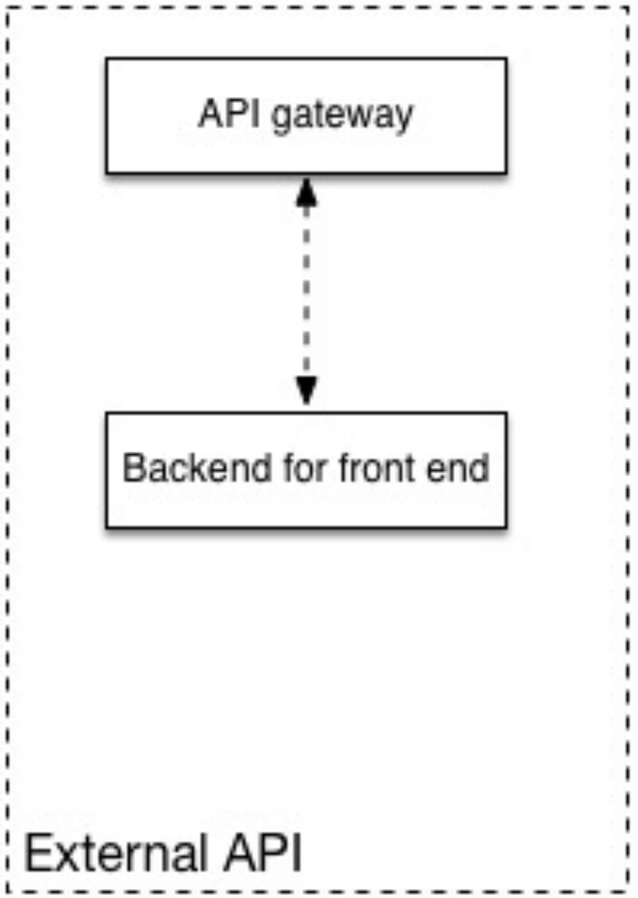
microservices.io/patterns/index.html



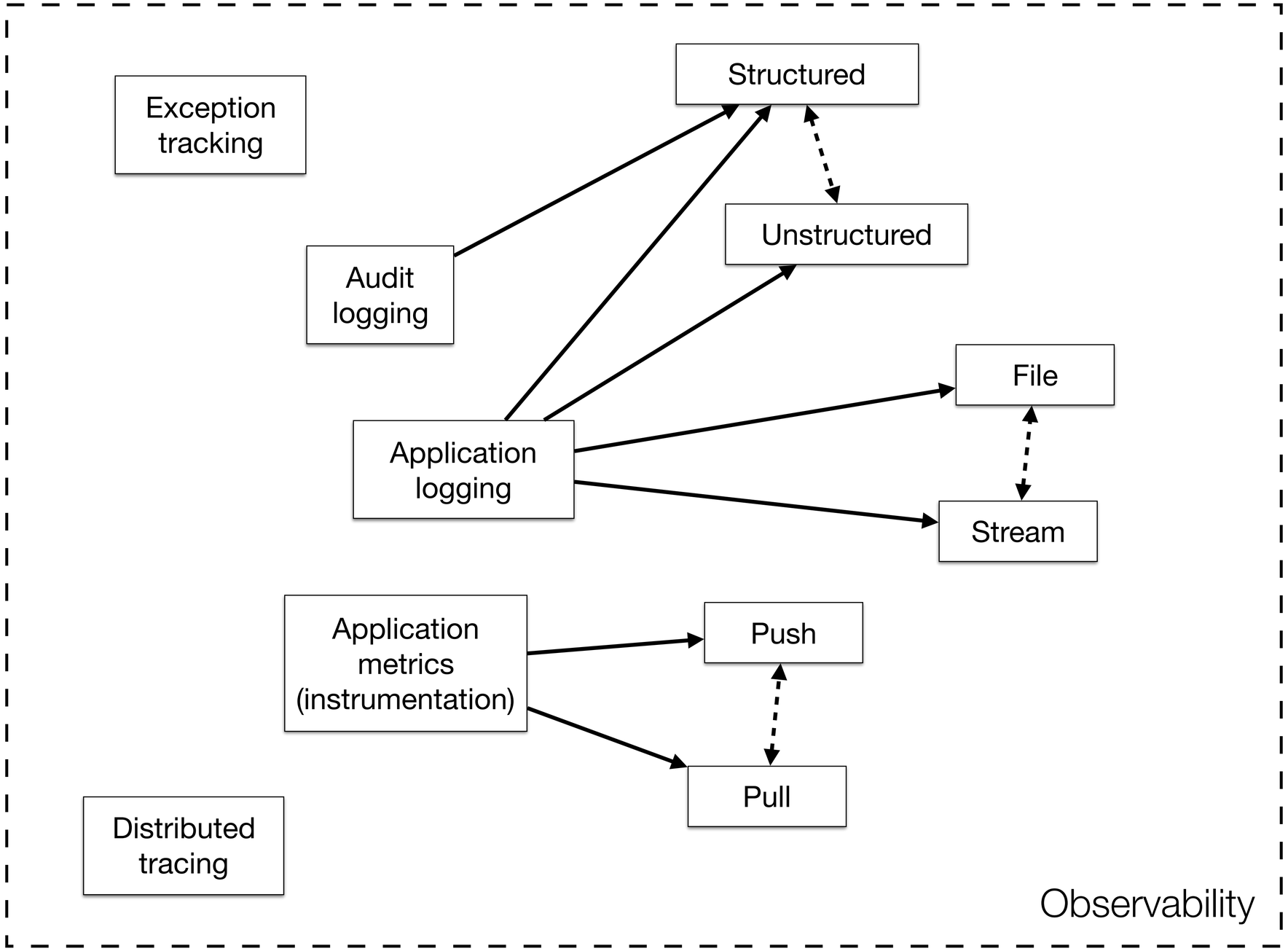
microservices.io/patterns/index.html



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

Resilience

Fallacies of distributed computing

From Wikipedia, the free encyclopedia

The fallacies [edit]

The fallacies are:[1]

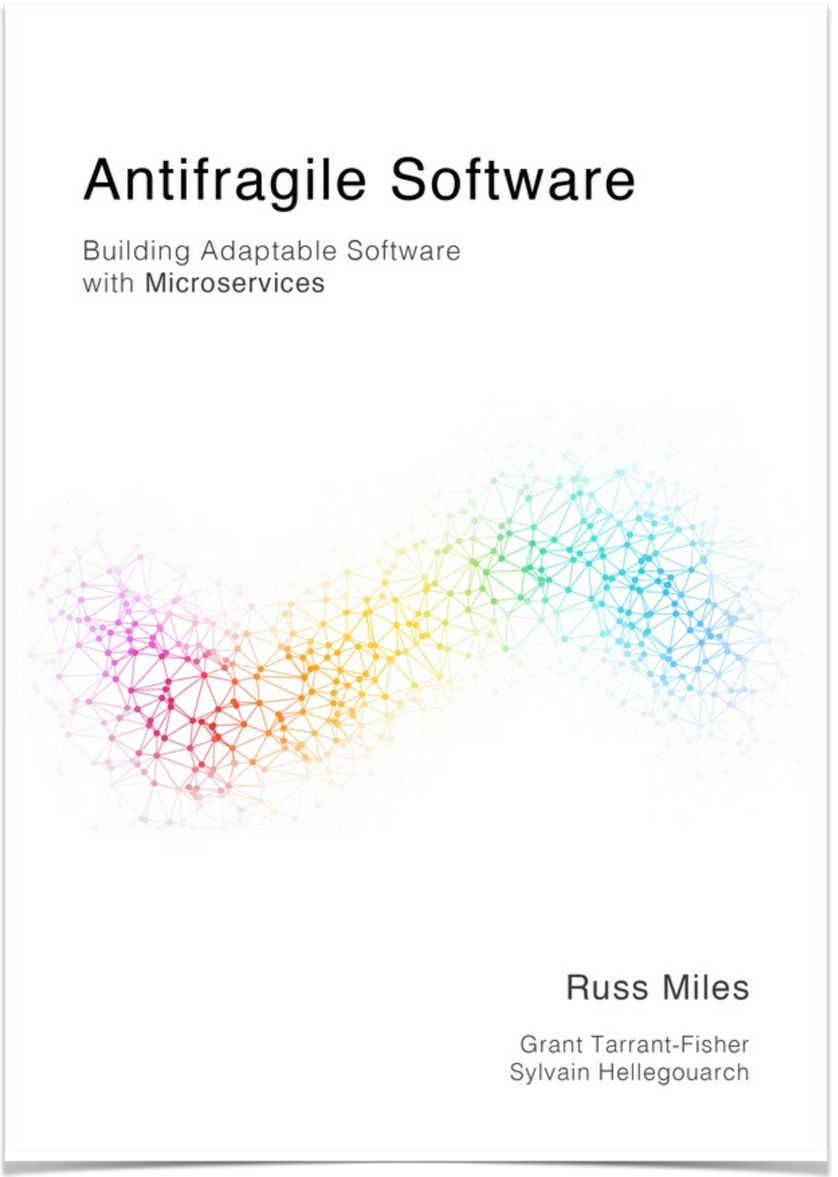
1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

en.wikipedia.org/wiki/Fallacies_of_distributed_computing

Concerns

- Monitoring: metrics, instrumentation
- Logging: application logging, event sourcing, audit trail
- Circuit breaking
- Rate limiting (ingress and egress)
- Timeouts and retry strategies (e.g. budget)
- Security: auth, crypto (in-motion and at-rest)


NETFLIX



leanpub.com/antifragilesoftware

Go microservices

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

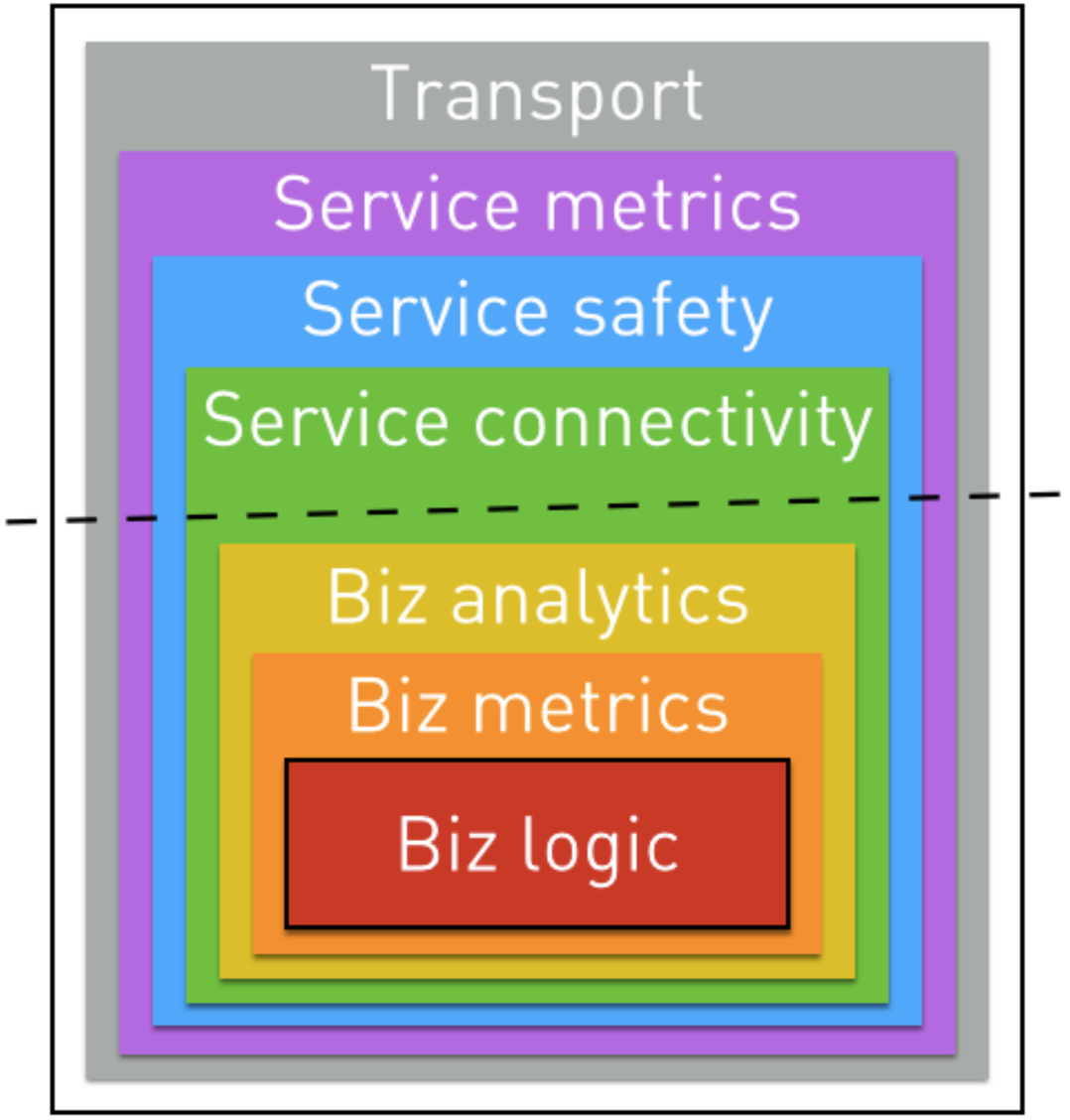
Metrics
Logging
Distributed tracing

- Transport
- Rate limiting
- Circuit breaking

Business logic

- Service registration
- Service discovery
- Load balancing

- Metrics
- Logging
- Distributed tracing


Transport
Service metrics
Service safety
Service connectivity
Biz analytics
Biz metrics
Biz logic

Programming time

Logging

Harder (more expensive) than it seems

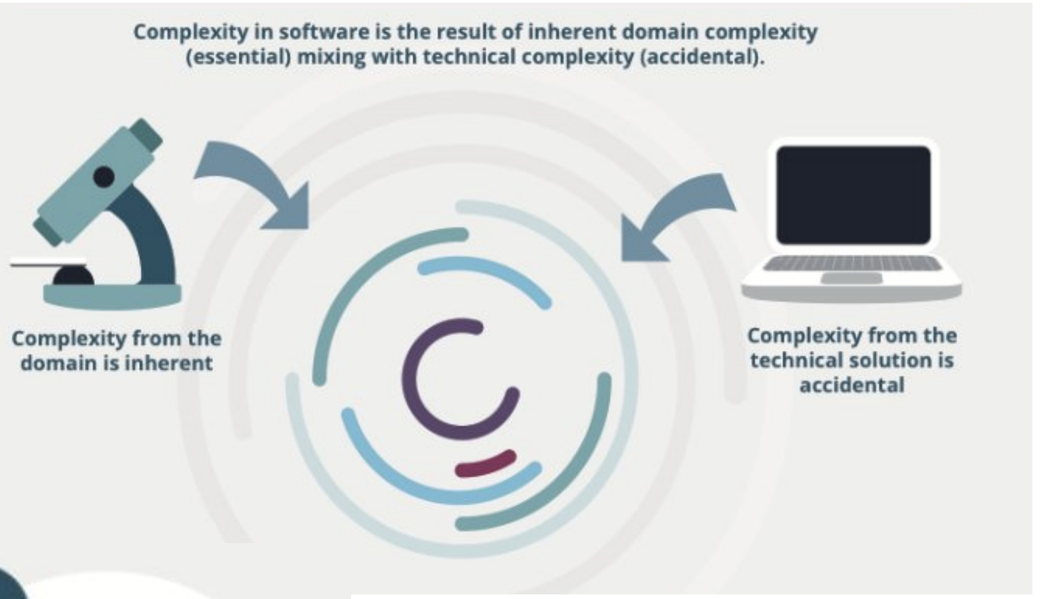
Instrumentation

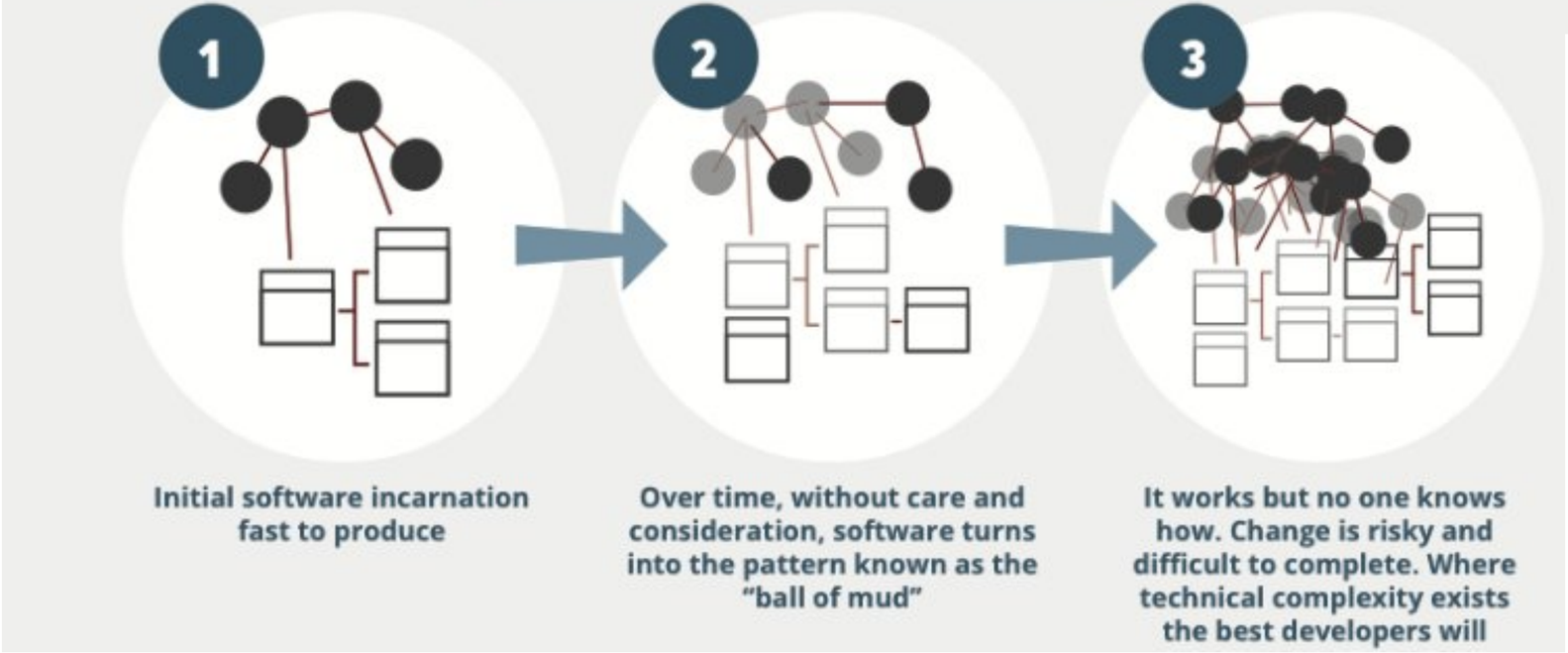
Easier (cheaper) than it seems

Scaffolding

THE ANATOMY OF DOMAIN-DRIVEN DESIGN

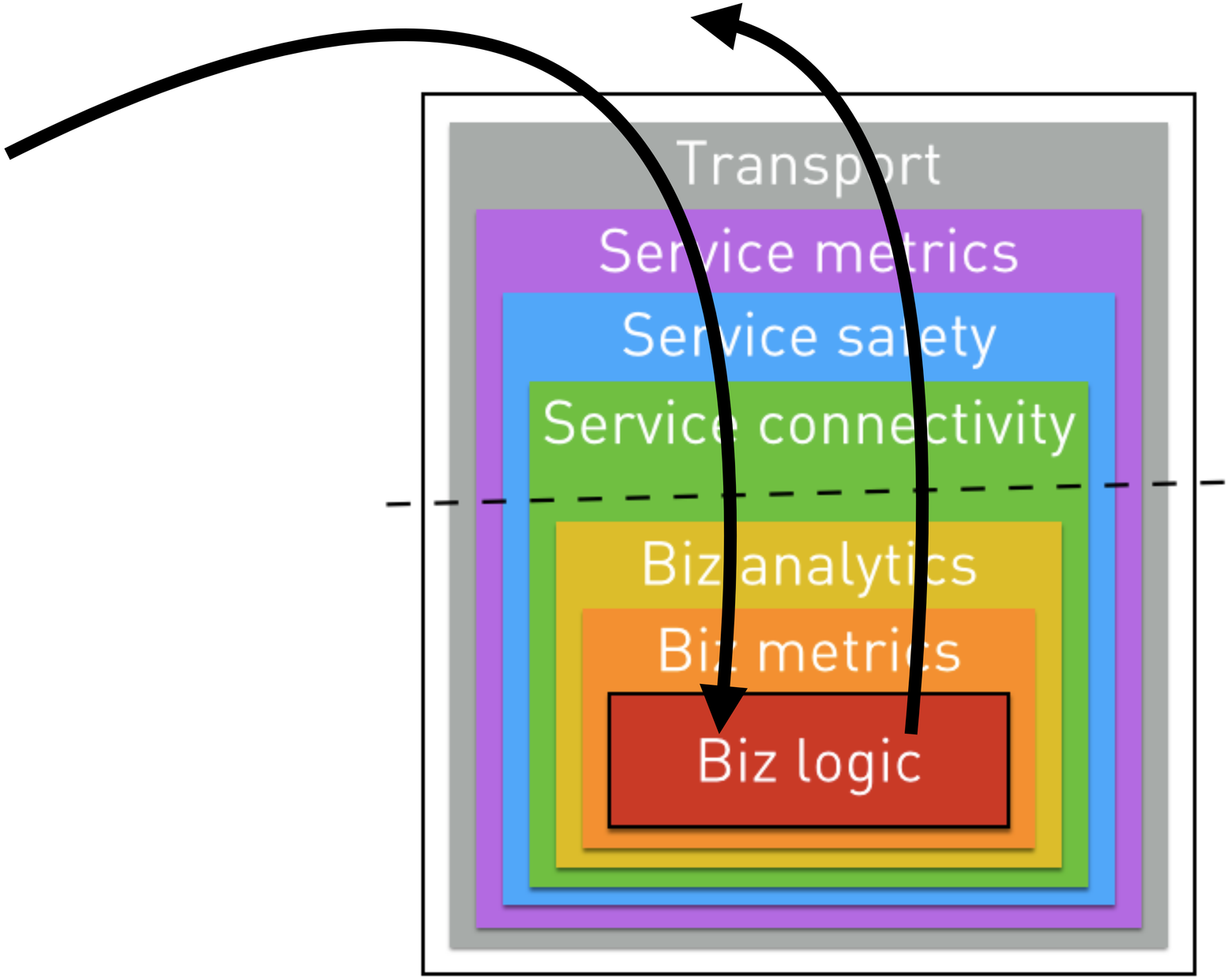
BY **SCOTT MILLETT**



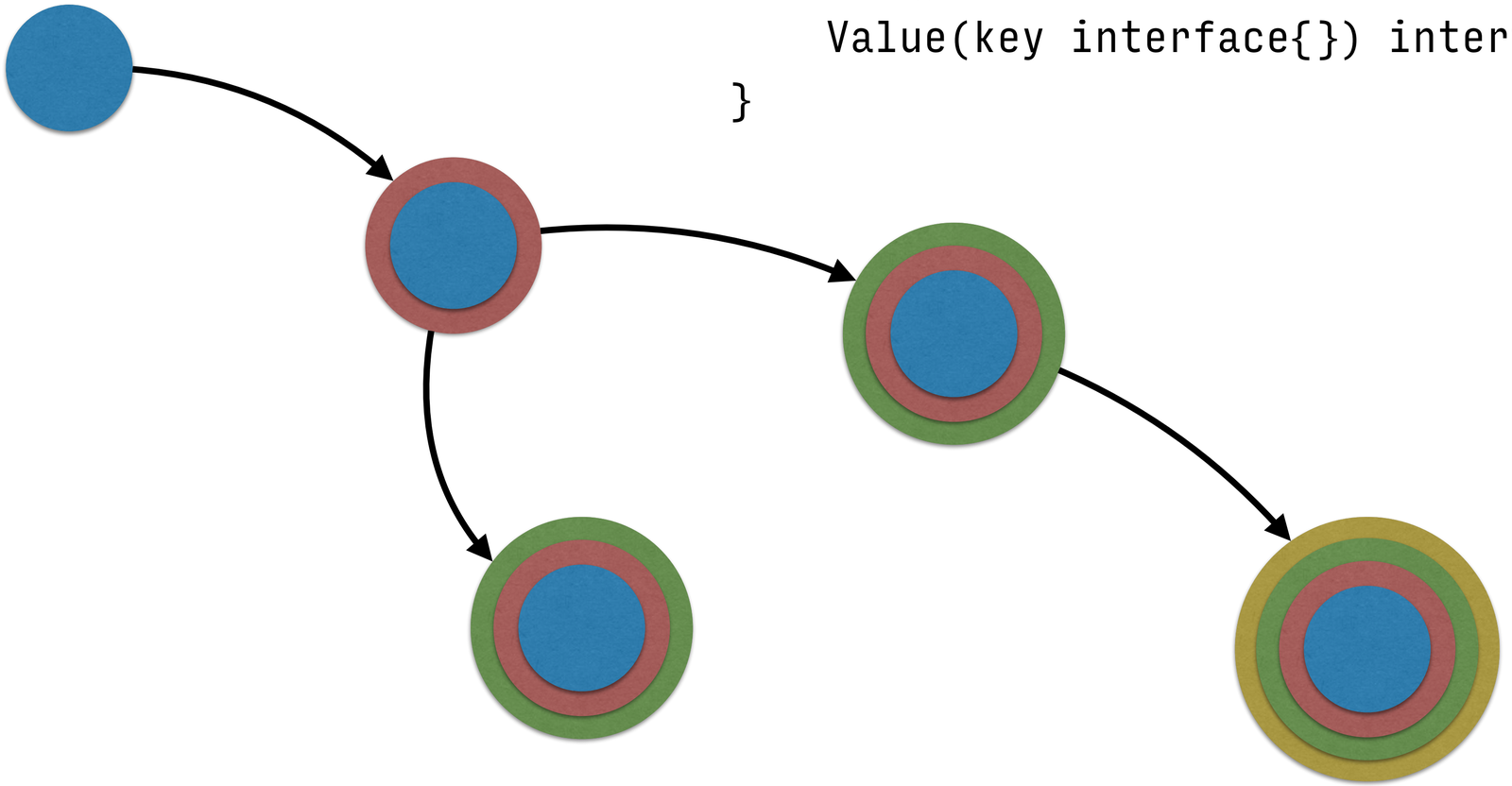


leanpub.com/theanatomyofdomain-drivendesign

context.Context


Transport
Service metrics
Service safety
Service connectivity
Biz analytics
Biz metrics
Biz logic

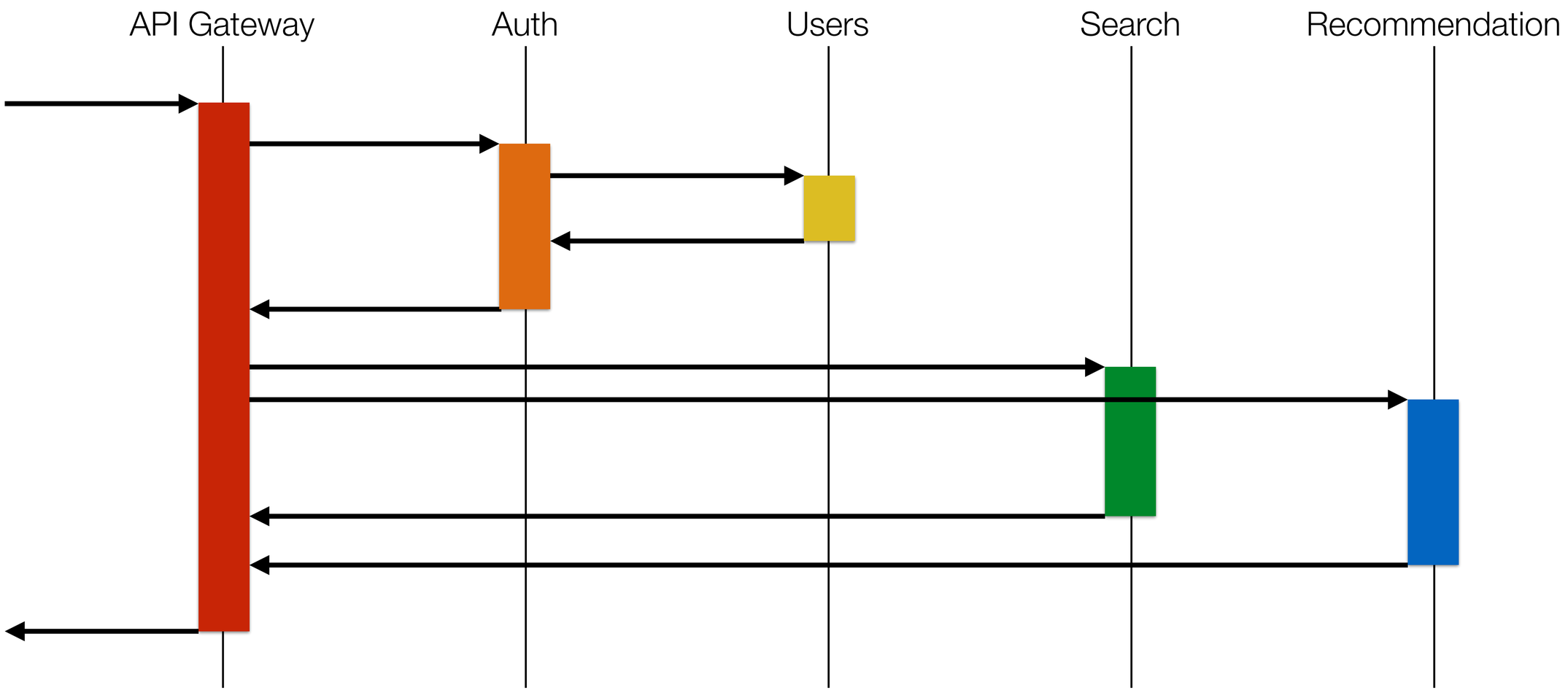
```
type Context interface {
    Deadline() (deadline time.Time, ok bool)
    Done() <-chan struct{}
    Err() error
    Value(key interface{}) interface{}
}
```

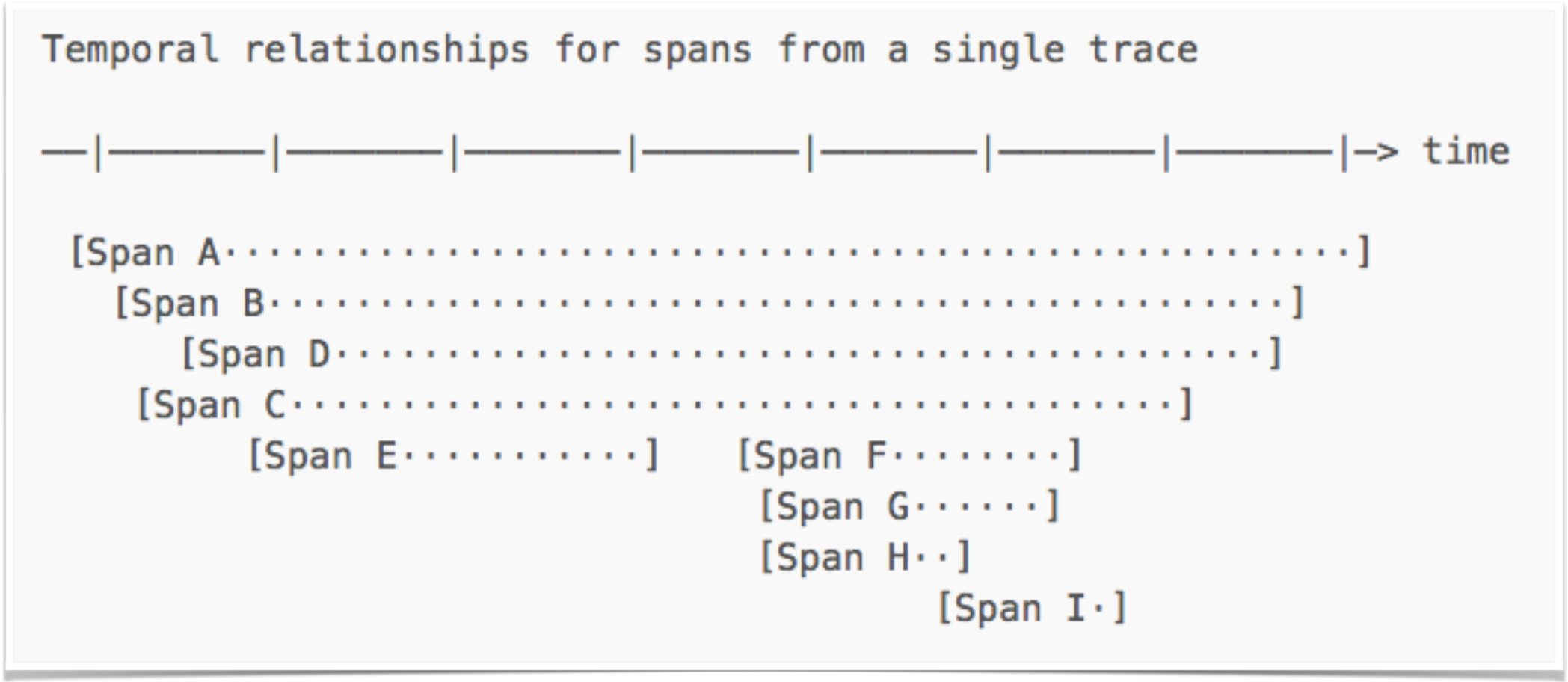


Distributed tracing

A **Trace** represents the potentially distributed, potentially concurrent data/execution path in a (potentially distributed, potentially concurrent) system. A Trace can be thought of as a tree of Spans. (See the ASCII diagrams above)

A **Span** represents a logical unit of work in the system that has a start time and a duration. Spans may be nested and ordered to model parent-child and causal relationships. Each span has an **operation name**, a presumably human-readable string which concisely names the work done by the span (e.g., an RPC method name, a function name, or the name of a subtask within a larger computation).


API Gateway
Auth
Users
Search
Recommendation


Temporal relationships for spans from a single trace
-|-------|-------|-------|-------|-------|-------|-------|-> time
[Span A·····]
[Span B·····]
[Span D·····]
[Span C·····]
[Span E·····]
[Span F·····]
[Span G·····]
[Span H·····]
[Span I·]

Conclusion

Microservices

Solve organizational problems &
Cause technical problems

Patterns + structure

Middleware pattern +
Separation of concerns

Go microservices

@peterbourgon · gokit.io