Architecture and Domain Modeling

Workshop · GopherCon Denver 2018

@peterbourgon

Setup for today

1. Go installed
2. Gophers Slack #sympatico
3. Trello account (optional)

Introduction


fastly

Plan for today

- Microservice theory
- An example monolith with a good foundation
- Extending the monolith with a new "service"
- Extracting a microservice from the monolith
- Introducing Go kit

Concepts

Practical Microservices

Microservices ***solve***
organizational problems

$\sim$

Microservices ***cause***
technical problems

Problems *solved*

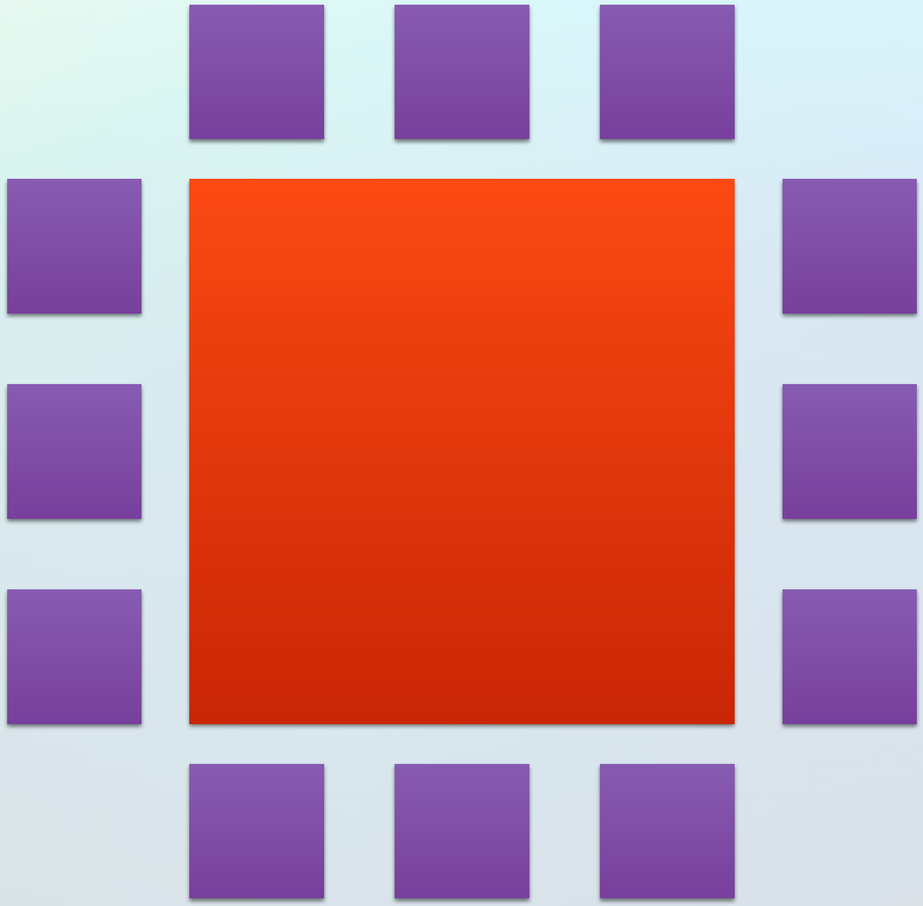
- Team is too large to work effectively on shared codebase
- Teams are blocked on other teams, can't make progress
- Communication overhead becomes gigantic
- Product velocity stalled

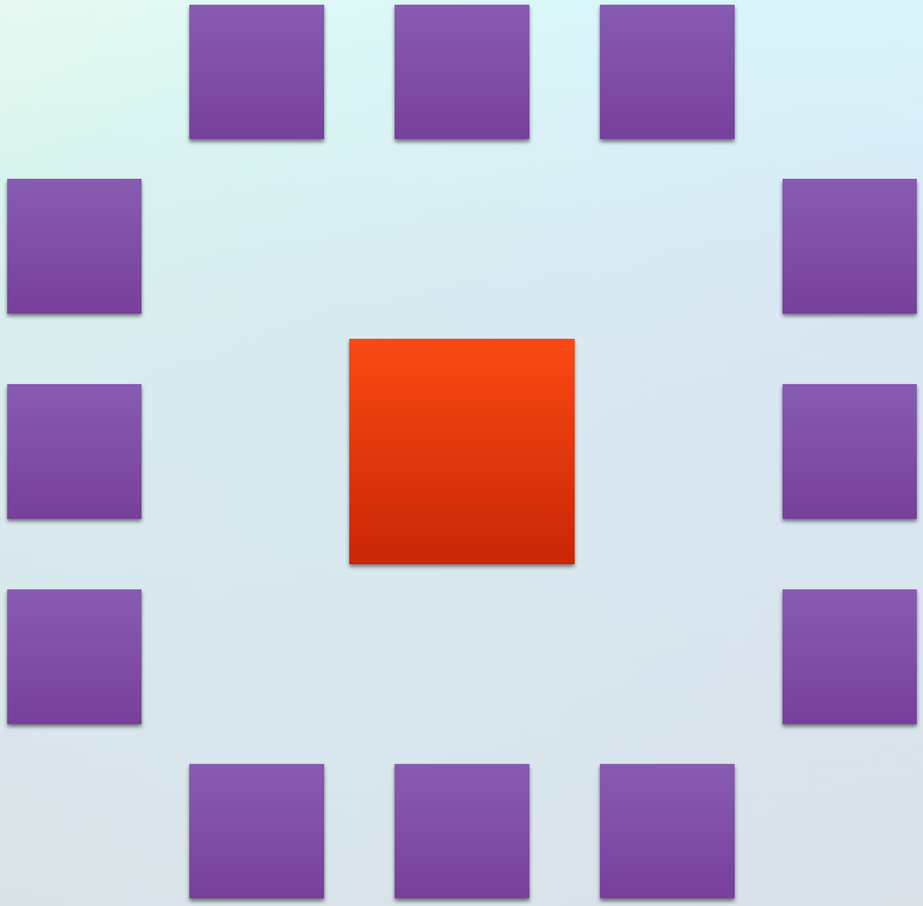
Problems *caused*

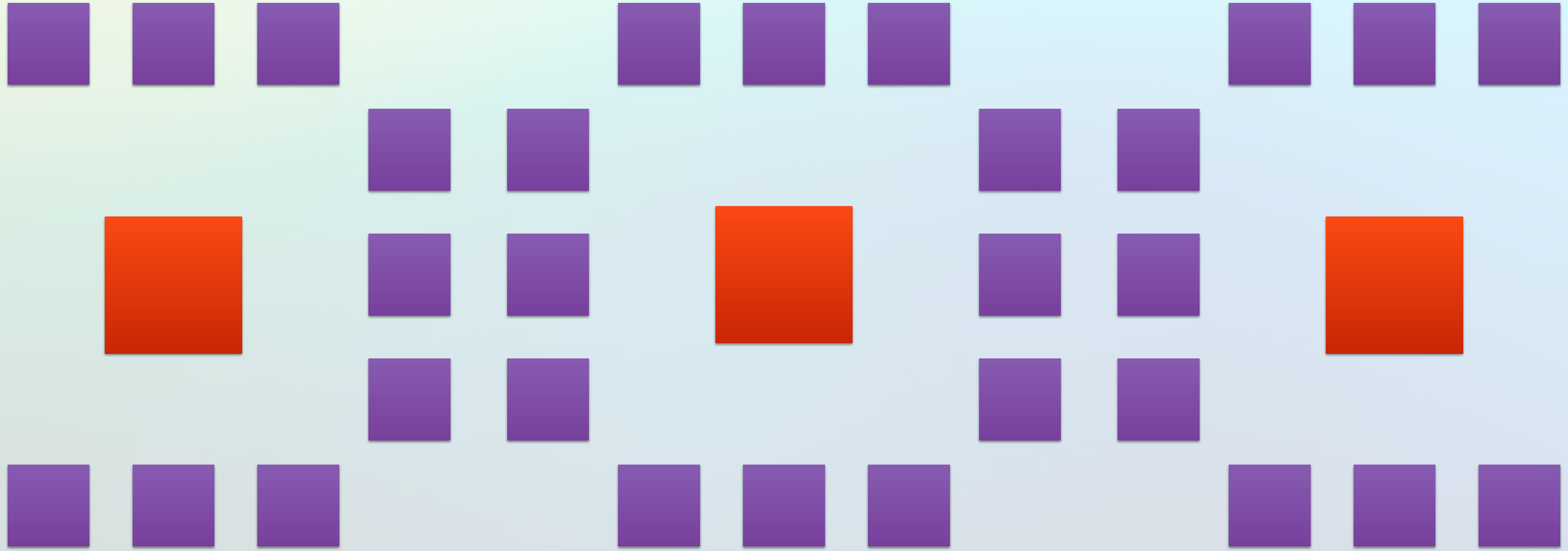
- Need well-defined business domains for stable APIs
- No more shared DB — distributed transactions?
- Testing becomes really hard
- Require dev/ops culture: devs deploy & operate their work
- Job (service) scheduling — manually works, for a while...

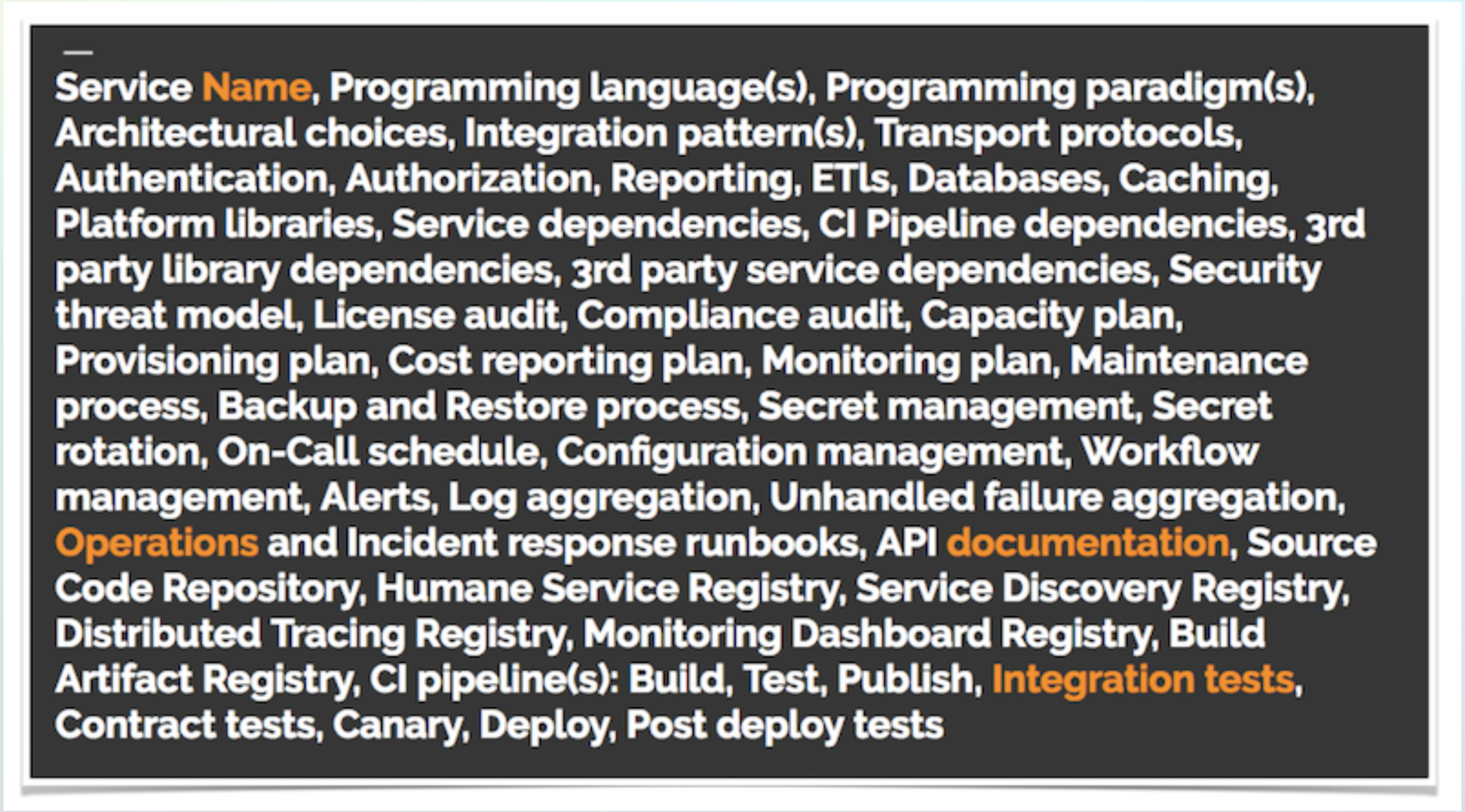
Problems *caused*

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









Concerns for a single service, Sean Treadway, SoundCloud

Go kit in theory

Initial goals

- A standard library for microservices
- Something like Finagle for Go
- Adapters, bindings, etc. to common infra components
- Play nice in your existing, heterogeneous infrastructure
- **Structure to tame the beast of incidental complexity**

Current goals

- Mostly the same
- Less about infrastructure, operational details, etc.
- More about **application architecture**

Non-goals

- Messaging patterns other than RPC
- Requiring specific bits of infrastructure or tooling to work
- Acting as an all-in service framework
- Re-implementing existing, good solutions to problems

Comparisons

- Micro *(Go)* — very opinionated, all-in, framework-ish
- Finagle *(Scala)* — original inspiration, lower-level than Go kit
- Spring Boot *(Java)* — similar abstractions, far more magical
- Tokio *(Rust)* — explicitly a clone of Finagle, lower-level than Go kit

I don’t care if you use Go kit, I care about

good architectural decisions 👉

in your service

Concepts

Domain Driven Design

Domain-driven design

From Wikipedia, the free encyclopedia

Domain-driven design (**DDD**) is an approach to software development for complex needs by connecting the implementation to an evolving model.[1] The premise of domain-driven design is the following:

- placing the project's primary focus on the core domain and domain logic;
- basing complex designs on a model of the domain;
- initiating a creative collaboration between technical and domain experts to iteratively refine a conceptual model that addresses particular domain problems.

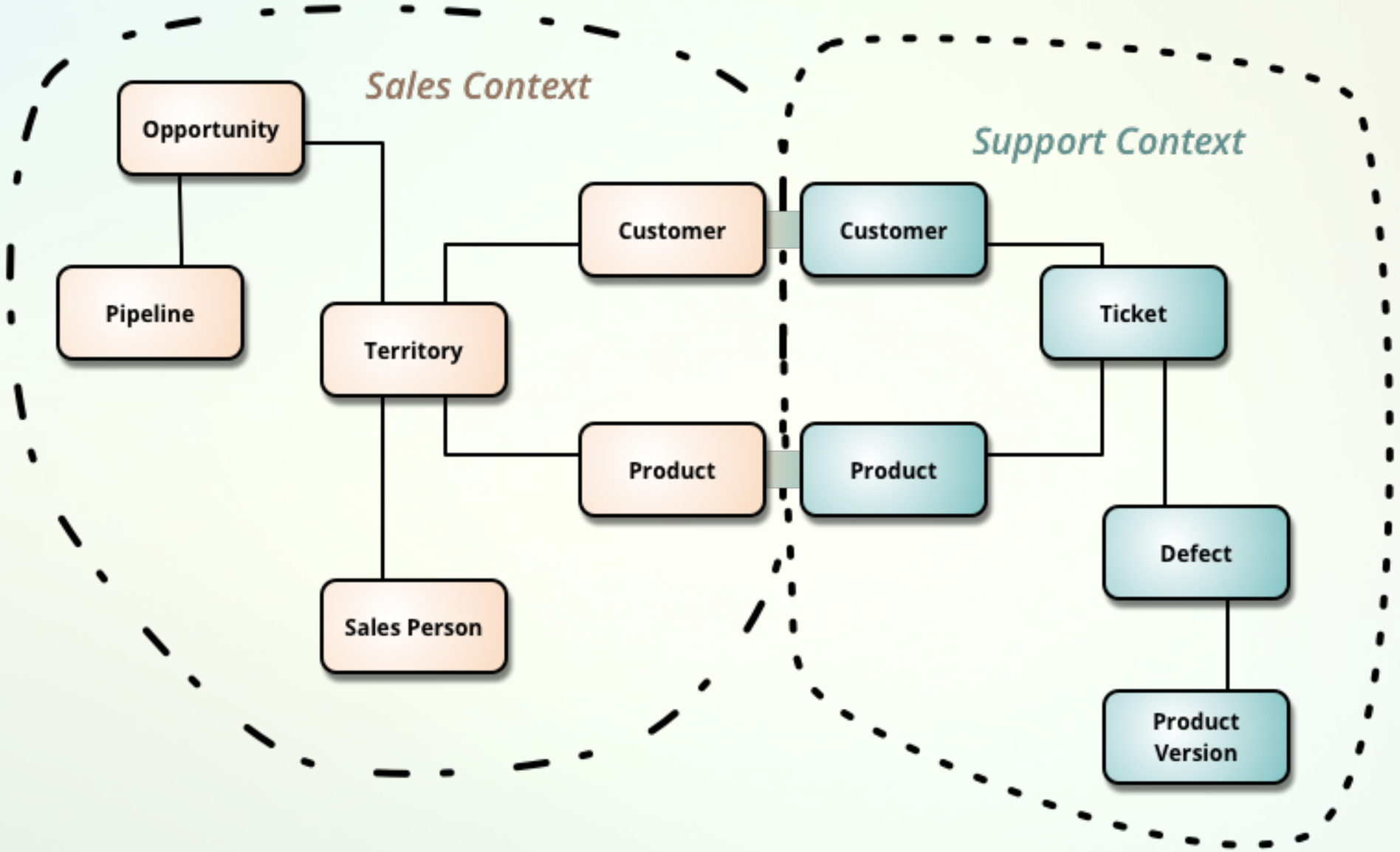
The term was coined by Eric Evans in his book of the same title.[2]

https://en.wikipedia.org/wiki/Domain-driven_design

Disadvantages [edit]

In order to help maintain the model as a pure and helpful language construct, the team must typically implement a great deal of isolation and encapsulation within the domain model. Consequently, a system based on domain-driven design can come at a relatively high cost. While domain-driven design provides many technical benefits, such as maintainability, Microsoft recommends that it be applied only to complex domains where the model and the linguistic processes provide clear benefits in the communication of complex information, and in the formulation of a common understanding of the domain.[3]

https://en.wikipedia.org/wiki/Domain-driven_design



https://www.martinfowler.com/bliki//BoundedContext.html

Example service

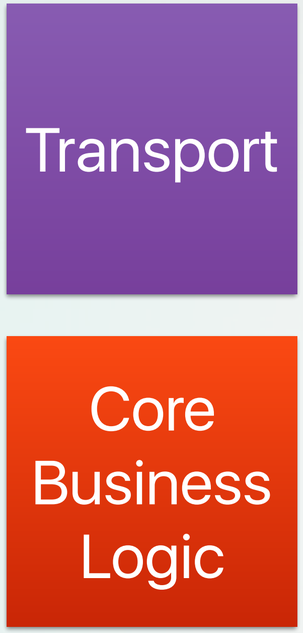
```
type AddService interface {
    Sum(a, b int) int
    Concat(a, b string) string
}
```

```
type AddService interface {
    Sum(a, b int) (int, error)
    Concat(a, b string) (string, error)
}
```

```
type basicService struct{}

func (s basicService) Sum(a, b int) (int, error) {
    return a + b, nil
}

func (s basicService) Concat(a, b string) (string, error) {
    return a + b, nil
}
```


Transport
Core
Business
Logic

```
func (s basicService) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    switch r.URL.Path {
    case "/sum":
```

```
func (s basicService) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    switch r.URL.Path {
    case "/sum":
        var req struct {
            A int `json:"a"`
            B int `json:"b"`
        }
        if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
            http.Error(w, err.Error(), http.StatusBadRequest)
            return
        }
```

```
func (s basicService) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    switch r.URL.Path {
    case "/sum":
        var req struct {
            A int `json:"a"`
            B int `json:"b"`
        }
        if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
            http.Error(w, err.Error(), http.StatusBadRequest)
            return
        }
        v, err := s.Sum(req.A, req.B)
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }
```

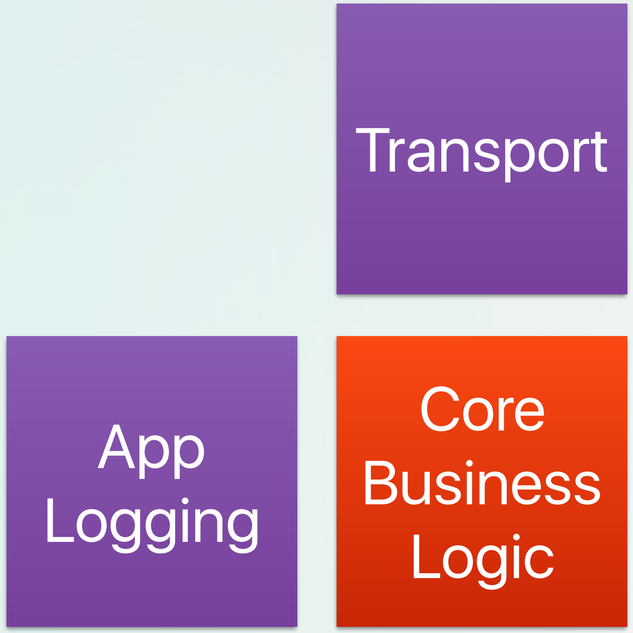
```
func (s basicService) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    switch r.URL.Path {
    case "/sum":
        var req struct {
            A int `json:"a"`
            B int `json:"b"`
        }
        if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
            http.Error(w, err.Error(), http.StatusBadRequest)
            return
        }
        v, err := s.Sum(req.A, req.B)
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }
        w.Header().Set("Content-Type", "application/json; charset=utf-8")
        json.NewEncoder(w).Encode(map[string]int{"v": v})
```

```
case "/concat":
```

```
case "/concat":
    var req struct {
        A string `json:"a"`
        B string `json:"b"`
    }
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
```

```
case "/concat":
    var req struct {
        A string `json:"a"`
        B string `json:"b"`
    }
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    v, err := s.Concat(req.A, req.B)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
```

```
case "/concat":
    var req struct {
        A string `json:"a"`
        B string `json:"b"`
    }
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    v, err := s.Concat(req.A, req.B)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    w.Header().Set("Content-Type", "application/json; charset=utf-8")
    json.NewEncoder(w).Encode(map[string]string{"v": v})
```

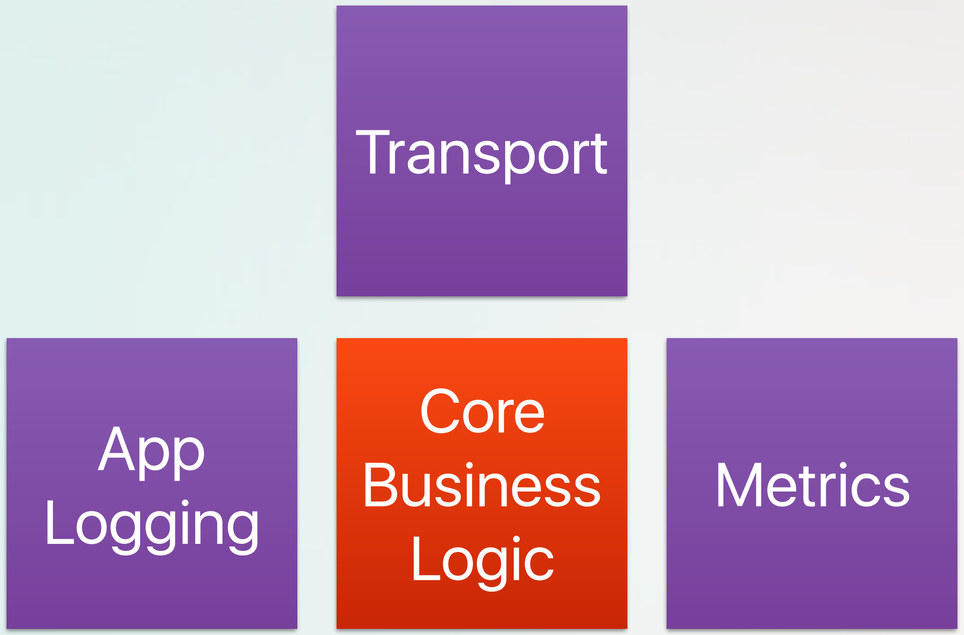

Transport
App
Logging
Core
Business
Logic

```
if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
    code := http.StatusBadRequest
    log.Printf("%s: %s: %d", r.RemoteAddr, r.URL, code)
    http.Error(w, err.Error(), code)
    return
}
v, err := s.Sum(req.A, req.B)
if err != nil {
    code := http.StatusInternalServerError
    log.Printf("%s: %s: %d", r.RemoteAddr, r.URL, code)
    http.Error(w, err.Error(), code)
    return
}
w.Header().Set("Content-Type", "application/json; charset=utf-8")
json.NewEncoder(w).Encode(map[string]int{"v": v})
log.Printf("%s: %s: %d", r.RemoteAddr, r.URL, 200)
```

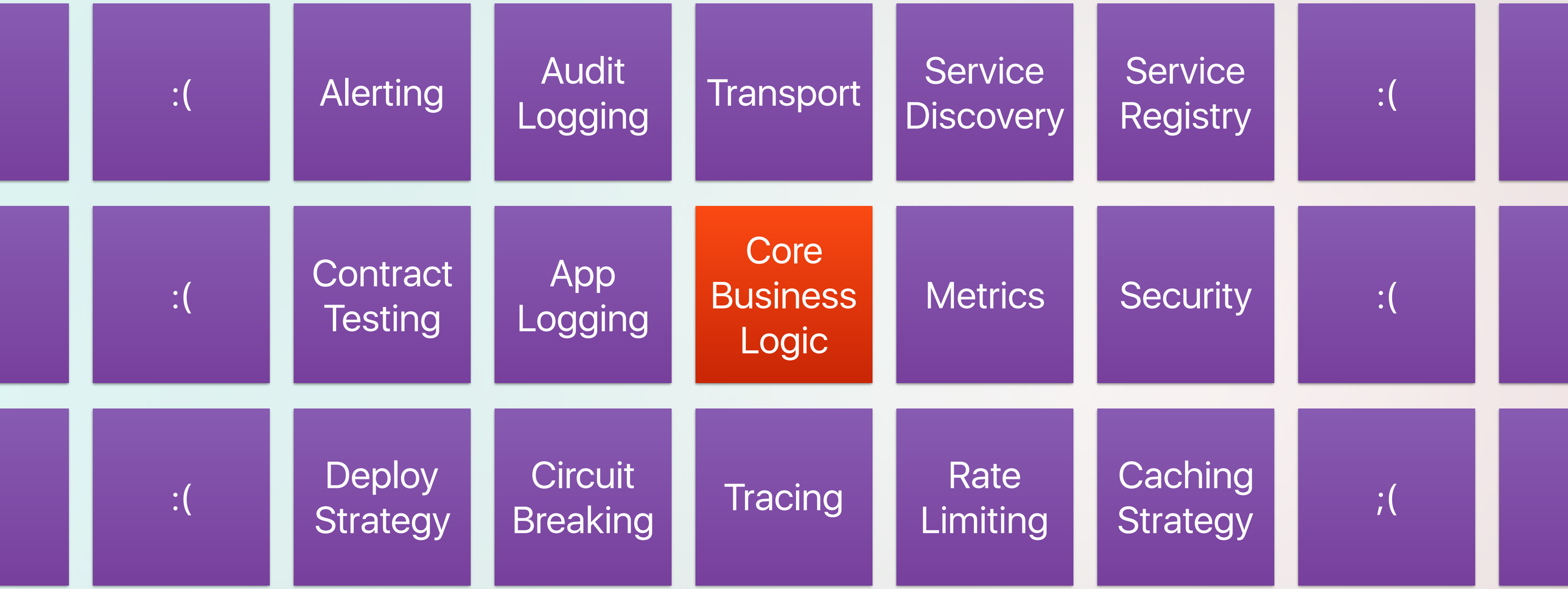
```
type basicService struct{}

func (s basicService) Sum(a, b int) (v int, err error) {
    defer func() {
        log.Printf("Sum(%d,%d)=(%d,%v)", a, b, v, err)
    }()
    return a + b, nil
}

func (s basicService) Concat(a, b string) (string, error) {
    defer func() {
        log.Printf("Concat(%q,%q)=(%q,%v)", a, b, v, err)
    }()
    return a + b, nil
}
```

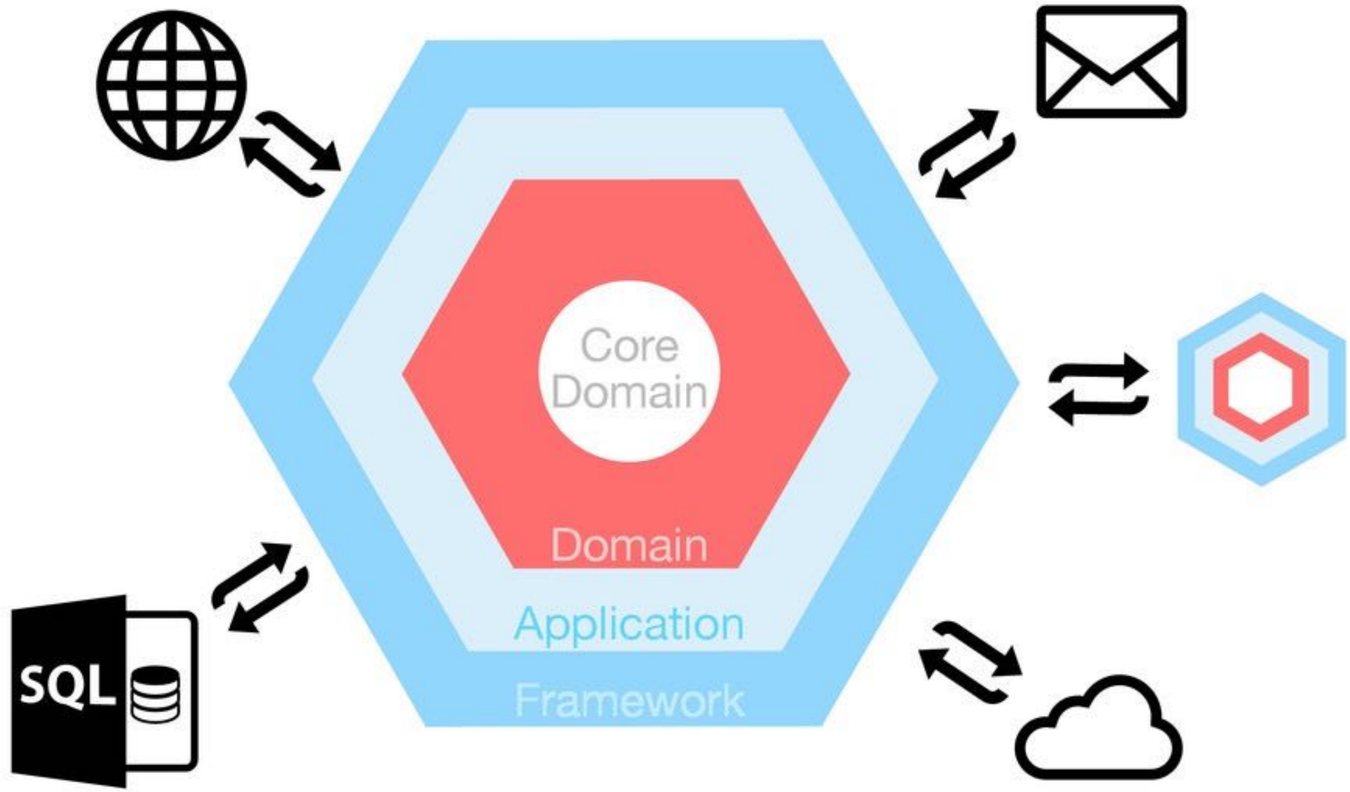

Transport
App
Logging
Core
Business
Logic
Metrics

```
if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
    code := http.StatusBadRequest
    log.Printf("%s: %s: %d", r.RemoteAddr, r.URL, code)
    errorCount.Add(1)
    http.Error(w, err.Error(), code)
    return
}
v, err := s.Sum(req.A, req.B)
if err != nil {
    code := http.StatusInternalServerError
    log.Printf("%s: %s: %d", r.RemoteAddr, r.URL, code)
    errorCount.Add(1)
    http.Error(w, err.Error(), code)
    return
}
w.Header().Set("Content-Type", "application/json; charset=utf-8")
json.NewEncoder(w).Encode(map[string]int{"v": v})
log.Printf("%s: %s: %d", r.RemoteAddr, r.URL, 200)
successCount.Add(1)
```

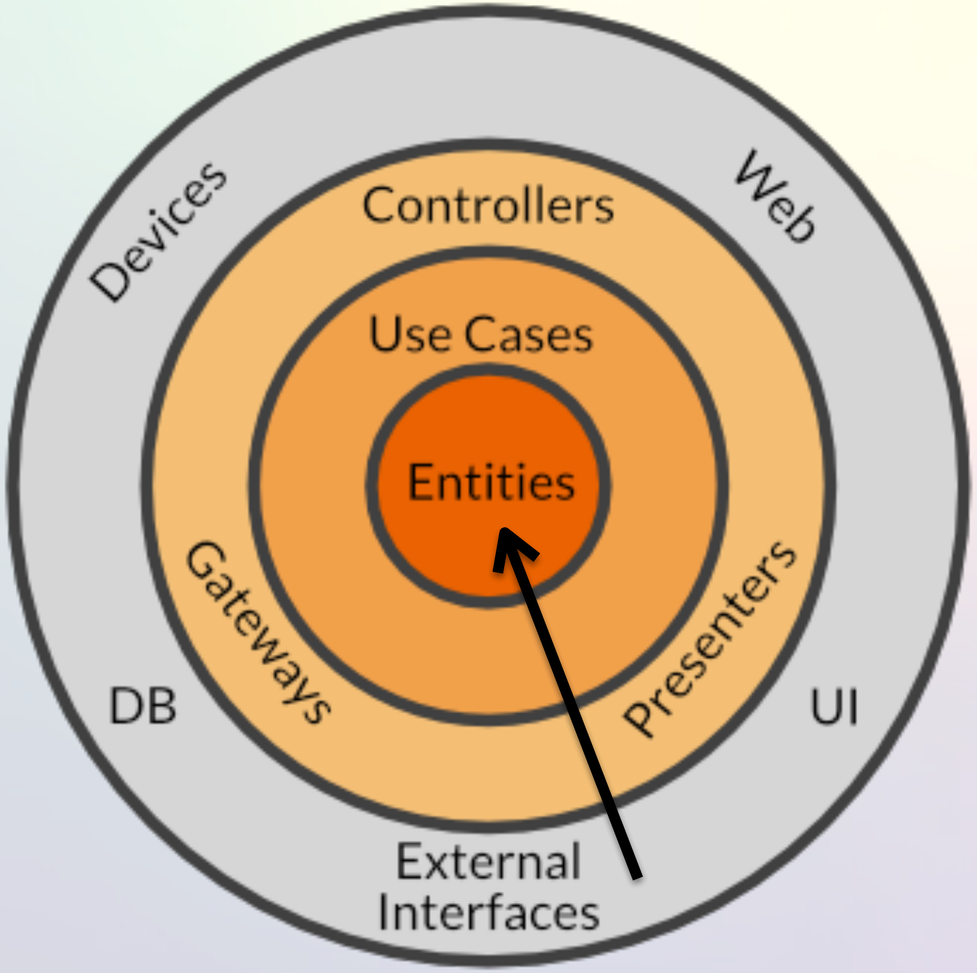

:(
Alerting
Audit Logging
Transport
Service Discovery
Service Registry
:(
:(
Contract Testing
App Logging
Core Business Logic
Metrics
Security
:(
:(
Deploy Strategy
Circuit Breaking
Tracing
Rate Limiting
Caching Strategy
;(

Patterns *a.k.a.*
Cages for complexity

The Hexagon

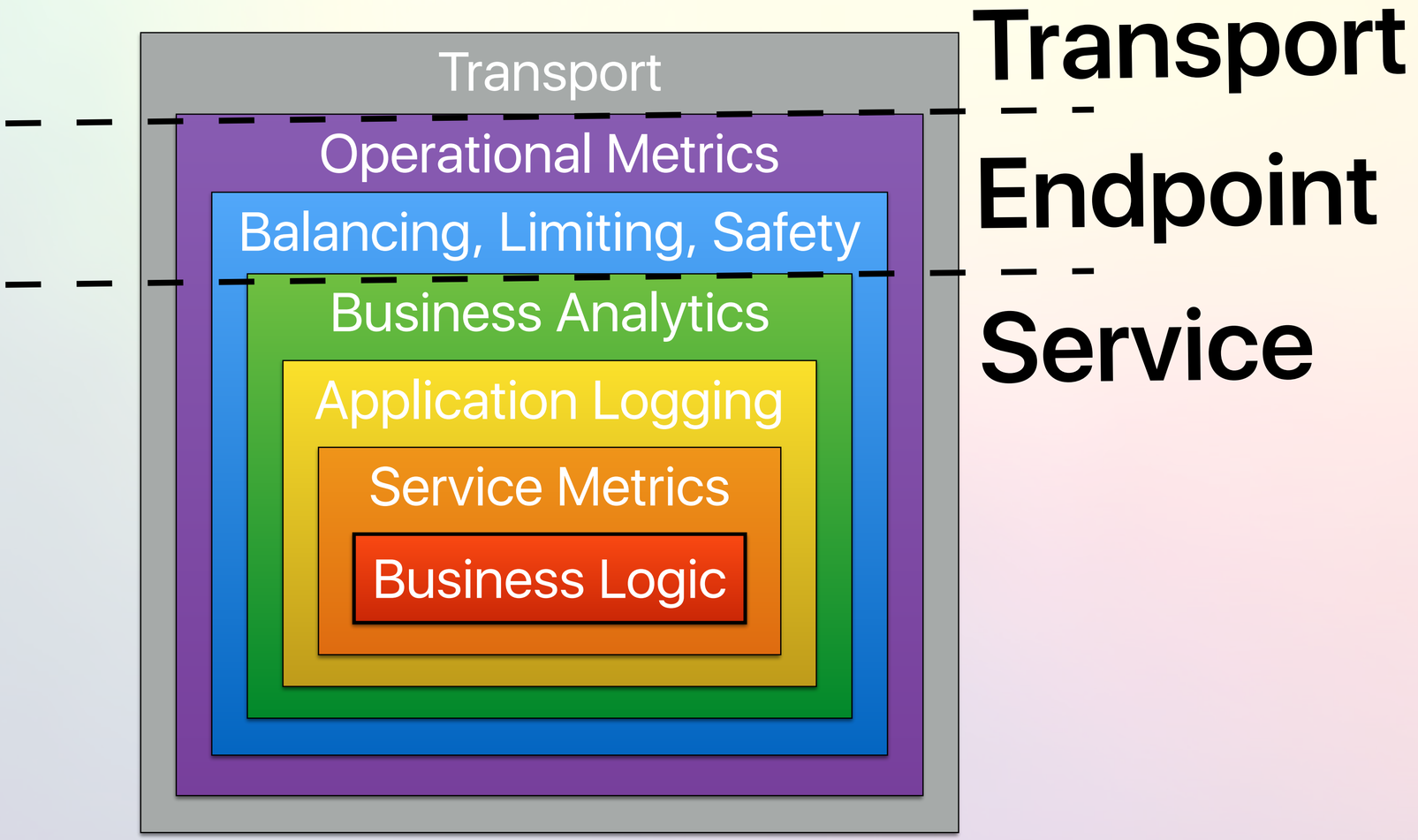


http://fideloper.com/hexagonal-architecture



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

source code dependencies can only point inwards


Transport
Operational Metrics
Balancing, Limiting, Safety
Business Analytics
Application Logging
Service Metrics
Business Logic
Transport
Endpoint
Service

Sympatico v1

Code walk

Sympatico v1

We’re missing something...

Individual exercise

1. Validate DNA sequence — *easy*
2. Add structured logging — *medium*
3. Instrument with Prometheus — *hard*

Group exercise

- Formally separate transport layer
- Apply same pattern to both services

```
func auth.NewTransport(s *auth.Service) http.Handler
func dna.NewTransport(s *dna.Service) http.Handler
```

Event Storming

Monetization strategy

Individual exercise

Implement your version of the monetization service as a peer to the dnasvc and authsvc.

Code review

Balkanization

Monolith → microservices

Your monolith will never completely go away

Accept this inviolable truth
with a serene and calm mind

When to extract a service?

- **Service has well-defined bounded context**
  - Minimal/no coördination with other services
- At least one of these is true:
  - Radically different runtime reqs (i.e. QPS)
  - Test/deploy/etc. on much different schedule

Individual exercise

Pick a service, extract it to a separate binary, and remove it from the monolith.
Hint: create a new directory in cmd/.

Code review

Go kit in practice

Actually, maybe don't

- Go kit enforces a lot of structure and boilerplate
- The benefit really comes in large and fluid orgs
- Good architecture already gives us 80% of the value
- Ask your doctor if Go kit is right for your team

Service layer

Business logic and related concerns

```
type AddService interface {
    Sum(a, b int) (int, error)
    Concat(a, b string) (string, error)
}
```

```
type Service interface {
    Sum(a, b int) (int, error)
    Concat(a, b string) (string, error)
}
```

```
type Service interface {
    Sum(ctx context.Context, a, b int) (int, error)
    Concat(ctx context.Context, a, b string) (string, error)
}
```

```
type Service interface {
    Sum(ctx context.Context, a, b int) (int, error)
    Concat(ctx context.Context, a, b string) (string, error)
}

type basicService struct{}
```

```
type Service interface {
    Sum(ctx context.Context, a, b int) (int, error)
    Concat(ctx context.Context, a, b string) (string, error)
}

type basicService struct{}

func (s basicService) Sum(_ context.Context, a, b int) (int, error) {
    if a == 0 && b == 0 {
        return 0, ErrTwoZeroes
    }
    if (b > 0 && a > (intMax-b)) || (b < 0 && a < (intMin-b)) {
        return 0, ErrIntOverflow
    }
    return a + b, nil
}
```

```
type Service interface {
    Sum(ctx context.Context, a, b int) (int, error)
    Concat(ctx context.Context, a, b string) (string, error)
}

type basicService struct{}

func (s basicService) Sum(_ context.Context, a, b int) (int, error) {
    if a == 0 && b == 0 {
        return 0, ErrTwoZeroes
    }
    if (b > 0 && a > (intMax-b)) || (b < 0 && a < (intMin-b)) {
        return 0, ErrIntOverflow
    }
    return a + b, nil
}

func (s basicService) Concat(_ context.Context, a, b string) (string, error) {
    if len(a)+len(b) > maxLen {
        return "", ErrMaxSizeExceeded
    }
    return a+b, nil
}
```

```
type Middleware func(Service) Service
```

```
type Middleware func(Service) Service

type loggingMiddleware struct {
    logger log.Logger
    next   Service
}
```

```
type Middleware func(Service) Service

type loggingMiddleware struct {
    logger log.Logger
    next   Service
}

func NewLoggingMiddleware(logger log.Logger) Middleware {
    return func(next Service) Service {
        return loggingMiddleware{logger, next}
    }
}
```

```
type Middleware func(Service) Service

type loggingMiddleware struct {
    logger log.Logger
    next   Service
}

func NewLoggingMiddleware(logger log.Logger) Middleware {
    return func(next Service) Service {
        return loggingMiddleware{logger, next}
    }
}

func (mw loggingMiddleware) Sum(ctx context.Context, a, b int) (v int, err error) {
    defer func() {
        mw.logger.Log("method", "Sum", "a", a, "b", b, "v", v, "err", err)
    }()
    return mw.next.Sum(ctx, a, b)
}

// Concat is the same
```

```
type Middleware func(Service) Service
```

```
type Middleware func(Service) Service

type instrumentingMiddleware struct {
    ints  metrics.Counter
    chars metrics.Counter
    next  Service
}
```

```
type Middleware func(Service) Service

type instrumentingMiddleware struct {
    ints  metrics.Counter
    chars metrics.Counter
    next  Service
}

func NewInstrumentingMiddleware(ints, chars metrics.Counter) Middleware {
    return func(next Service) Service {
        return instrumentingMiddleware{
            ints:  ints,
            chars: chars,
            next:  next,
        }
    }
}
```

```
type Middleware func(Service) Service

type instrumentingMiddleware struct {
    ints  metrics.Counter
    chars metrics.Counter
    next  Service
}

func NewInstrumentingMiddleware(ints, chars metrics.Counter) Middleware {
    return func(next Service) Service {
        return instrumentingMiddleware{
            ints:  ints,
            chars: chars,
            next:  next,
        }
    }
}

func (mw instrumentingMiddleware) Sum(ctx context.Context, a, b int) (int, error) {
    v, err := mw.next.Sum(ctx, a, b)
    mw.ints.Add(v)
    return v, err
}
```

And a lot more...!

Endpoint layer

Transport adapter & generic middleware

```
type Endpoint func(req) resp
```

```
type Endpoint func(req interface{}) (resp interface{})
```

```
type Endpoint func(ctx context.Context, req interface{}) (resp interface{}, err error)
```

```
type Endpoint func(ctx context.Context, req interface{}) (resp interface{}, err error)

func MakeSumEndpoint(s Service) Endpoint {
    return func(ctx context.Context, req interface{}) (resp interface{}, err error) {
        request := req.(SumRequest)
        v, err := s.Sum(ctx, request.A, request.B)
        return SumResponse{V: v, Err: err}, nil
    }
}
```

```
type Endpoint func(ctx context.Context, req interface{}) (resp interface{}, err error)

func MakeSumEndpoint(s Service) Endpoint {
    return func(ctx context.Context, req interface{}) (resp interface{}, err error) {
        request := req.(SumRequest)
        v, err := s.Sum(ctx, request.A, request.B)
        return SumResponse{V: v, Err: err}, nil
    }
}

type SumRequest struct {
    A, B int
}

type SumResponse struct {
    V   int
    Err error
}
```

```
type Middleware func(Endpoint) Endpoint
```

```
type Middleware func(Endpoint) Endpoint

func NewCircuitBreakingMiddleware(cb *gobreaker.CircuitBreaker) Middleware {
    return func(next Endpoint) Endpoint {
        return func(ctx context.Context, req interface{}) (interface{}, error) {
            return cb.Execute(func() (interface{}, error) { return next(ctx, req) })
        }
    }
}
```

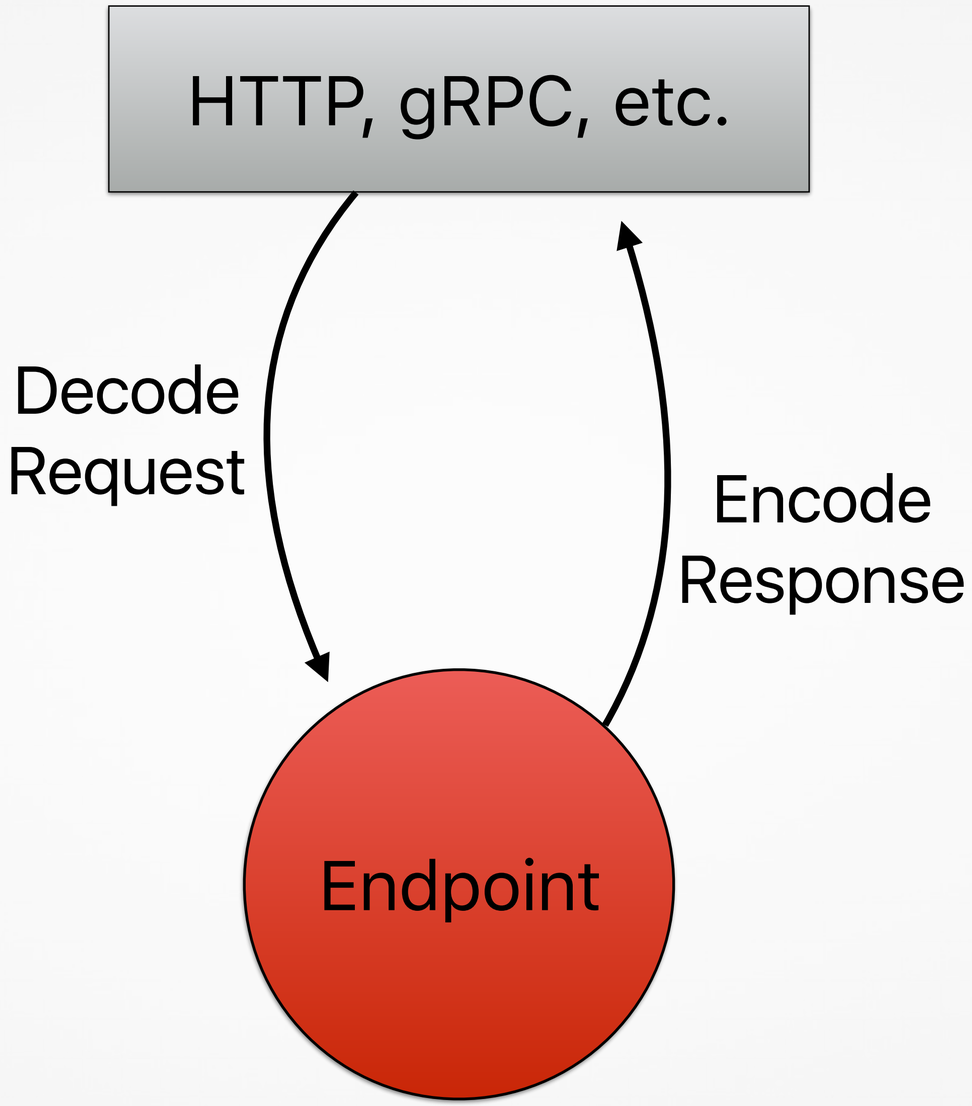
```
type Middleware func(Endpoint) Endpoint

func NewCircuitBreakingMiddleware(cb *gobreaker.CircuitBreaker) Middleware {
    return func(next Endpoint) Endpoint {
        return func(ctx context.Context, req interface{}) (interface{}, error) {
            return cb.Execute(func() (interface{}, error) { return next(ctx, req) })
        }
    }
}

func NewThrottlingMiddleware(b *ratelimit.Bucket) Middleware {
    return func(next Endpoint) Endpoint {
        return func(ctx context.Context, req interface{}) (interface{}, error) {
            time.Sleep(tb.Take(1))
            return next(ctx, request)
        }
    }
}
```

Transport layer

{Clean, Hexagonal, Onion} Architecture


HTTP, gRPC, etc.
Decode
Request
Encode
Response
Endpoint

HTTP

```
func NewServer(

)

}
```

```
func (s Server) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    request, err := s.dec(ctx, r)
    if err != nil {
        return
    }

    response, err := s.e(ctx, request)
    if err != nil {
        return
    }

    if err := s.enc(ctx, w, response); err != nil {
        return
    }
}
```

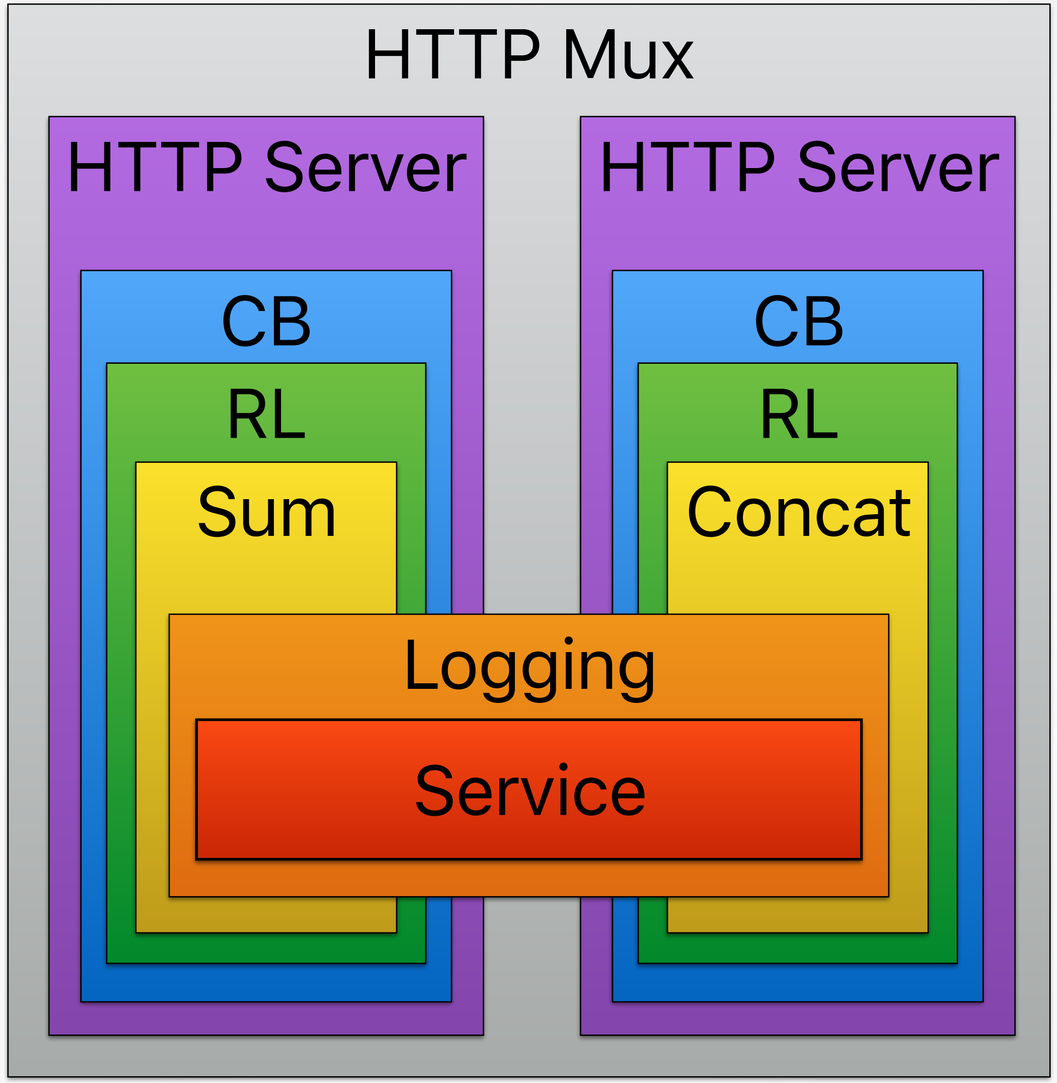
gRPC

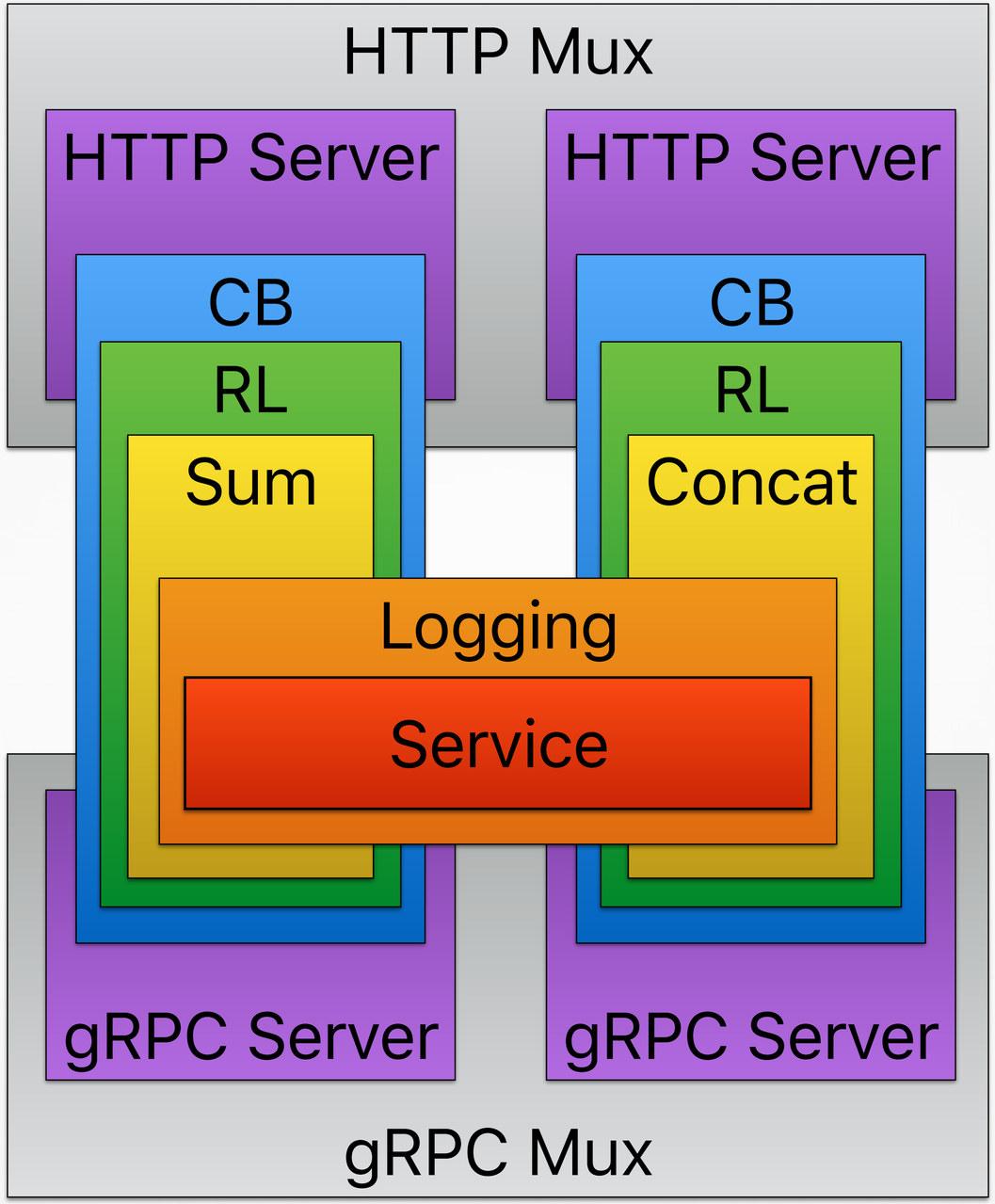
```
func NewServer(
)
}
```

```
func (s Server) ServeGRPC(ctx Context, req interface{}) (Context, interface{},
    request, err := s.dec(ctx, req)
    if err != nil {
        s.logger.Log("err", err)
        return ctx, nil, err
    }

    response, err := s.e(ctx, request)
    if err != nil {
        return ctx, nil, err
    }

    grpcResp, err := s.enc(ctx, response)
    if err != nil {
        return ctx, nil, err
    }
    return ctx, grpcResp, nil
}
```


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

We embrace and extend Go’s philosophy, to be

Simple* and non-magical*

in service to software engineering in the large

Optimize for

Maintainability

above all else

Individual exercise

Add an endpoint layer to your extracted service, and use Go kit's package transport/http to serve HTTP.

Hint: for each service method, you'll need request and response types, decodeRequest and encodeResponse funcs, and a makeEndpoint constructor.

Individual exercise

```
// NewHTTPHandler returns an http.Handler with routes for each endpoint.
// It uses the Go kit style endpoints, and Go kit http.Servers.
func NewHTTPHandler(service *Service) http.Handler {
    r := mux.NewRouter()
    {
        r.Methods("POST").Path("/signup").Handler(kithttp.NewServer(
            makeSignupEndpoint(service),
            decodeSignupRequest,
            encodeSignupResponse,
        ))
        // ...
```

Code review

Go kit value-add

- Transport layer: HTTP, gRPC, Thrift, Twirp...
- Endpoint layer: circuit breaking, rate limiting, etc.
- Service layer: your domain, but consistent patterns
- Cross-layer: distributed tracing, error mapping
- Adapters for service discovery and metrics systems

Go kit in your org

- Maybe (probably?) don’t use Go kit at all...*!*
- Create mycorpkit, wrapping Go kit with sane defaults
- Investigate scaffolding builders/code generators
- Goal = reliably produce *maintainable* services

Architecture and Domain Modeling

Workshop · GopherCon Denver 2018

@peterbourgon