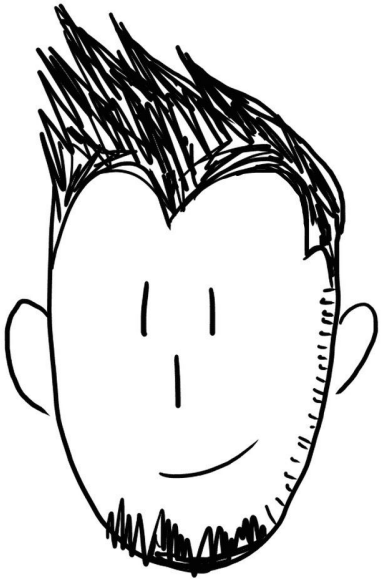
Architecture & Go kit

mercari.go #3

About me

- morikuni
- https://twitter.com/inukirom
- https://github.com/morikuni
- Mercari Microservices Development
- Go & Application Architecture



GopherCon 2018
Pre-Conference Workshop:

Architecture & Domain Modeling with Go Kit

https://www.gophercon.com/agenda/session/16822

Speaker

- Mr. Peter Bourgon
- https://twitter.com/peterbourgon
- Distributed System Engineer
- Fastly

Workshopの内容

- Microservicesについて
- Go kitの紹介
- https://github.com/peterbourgon/sympatico

  を使った演習

Microservicesについて

Microservicesは組織の課題を解決する

Microservicesは技術的な課題を生み出す

解決する課題

- 大きな組織が効率的に作業できるようになる
- 他のチームにブロックされなくなる
- コミュニケーションコストを下げる

生み出される課題

- サービス境界を上手く定義する必要がある
- 各サービスのモニタリング・分散トレーシング etc...

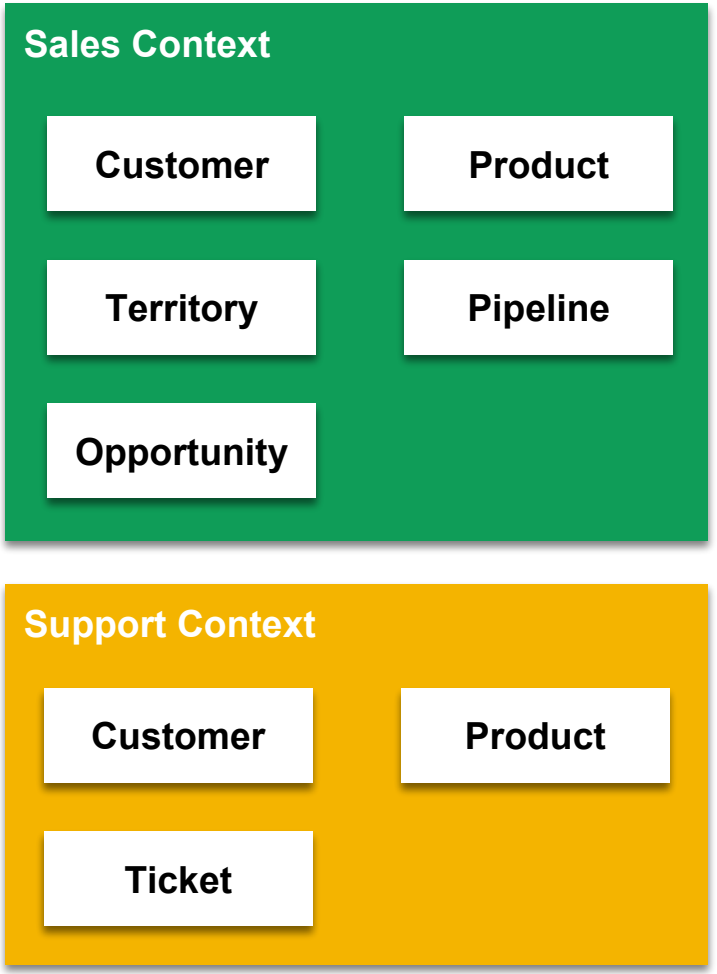
生み出される課題

- サービス境界を上手く定義する必要がある
  - DDD, Event Storming
- 各サービスのモニタリング・分散トレーシング etc…
  - Go kit

DDD (ドメイン駆動設計)

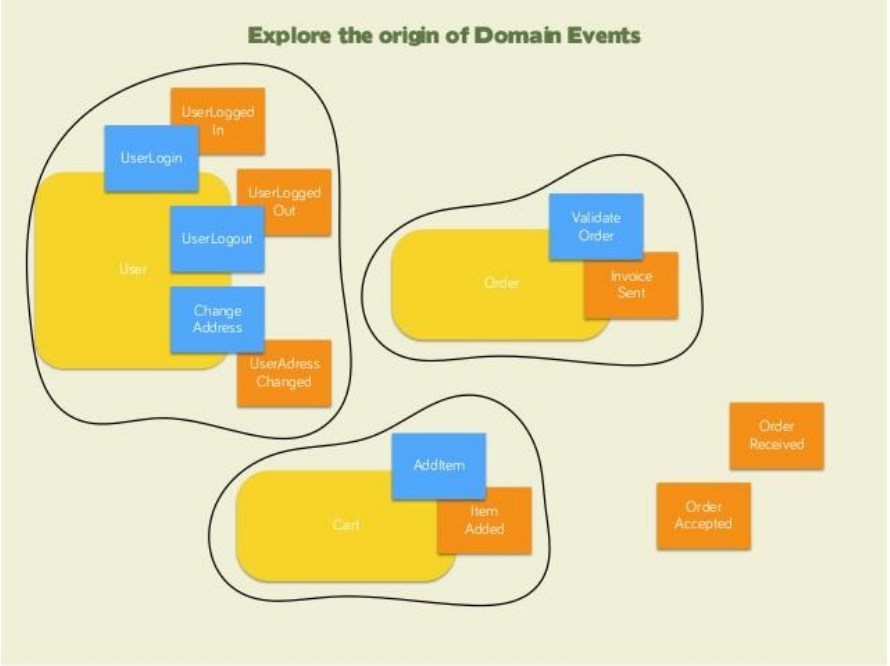
- データではなく**ドメインモデル**を中心とした設計
- **境界付けられたコンテキスト**に従ってサービスを分割する

境界付けられたコンテキスト: モデルが有効な範囲



Event Storming

- モデリング手法
- ビジネス内で発生するイベントを洗い出す
  - TicketCreated
  - ProductSold
- イベントの関連が理解しやすいようなモデルを構築する



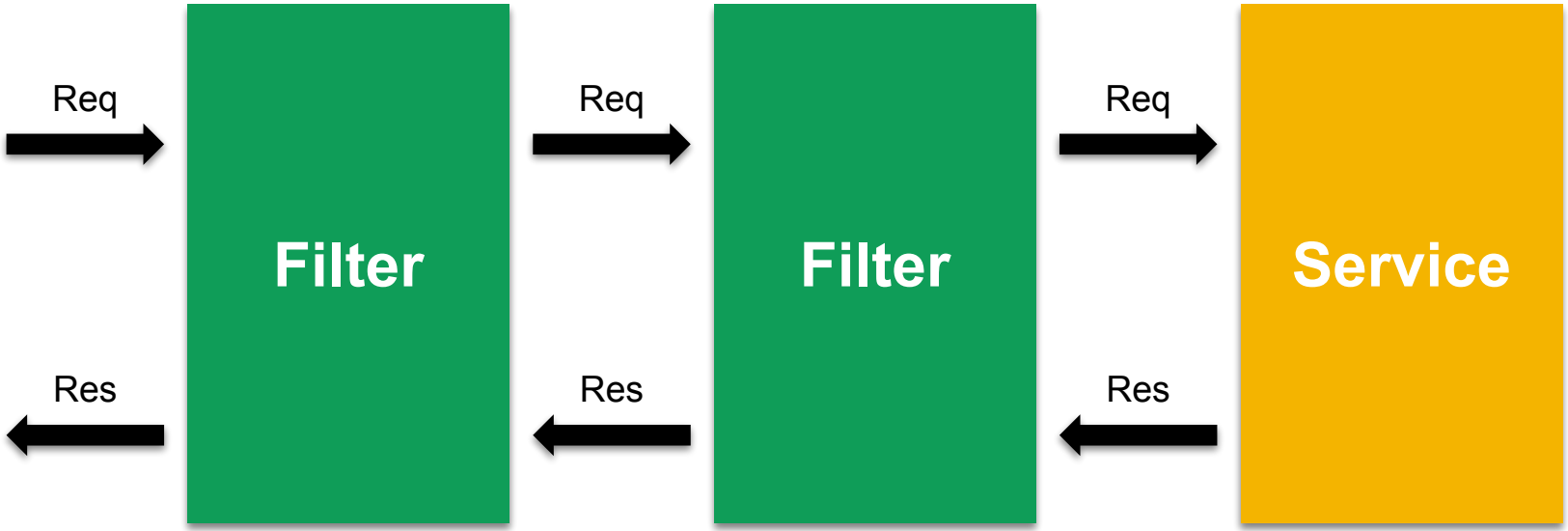
https://www.slideshare.net/aloyer/event-storming-notes

Go kit

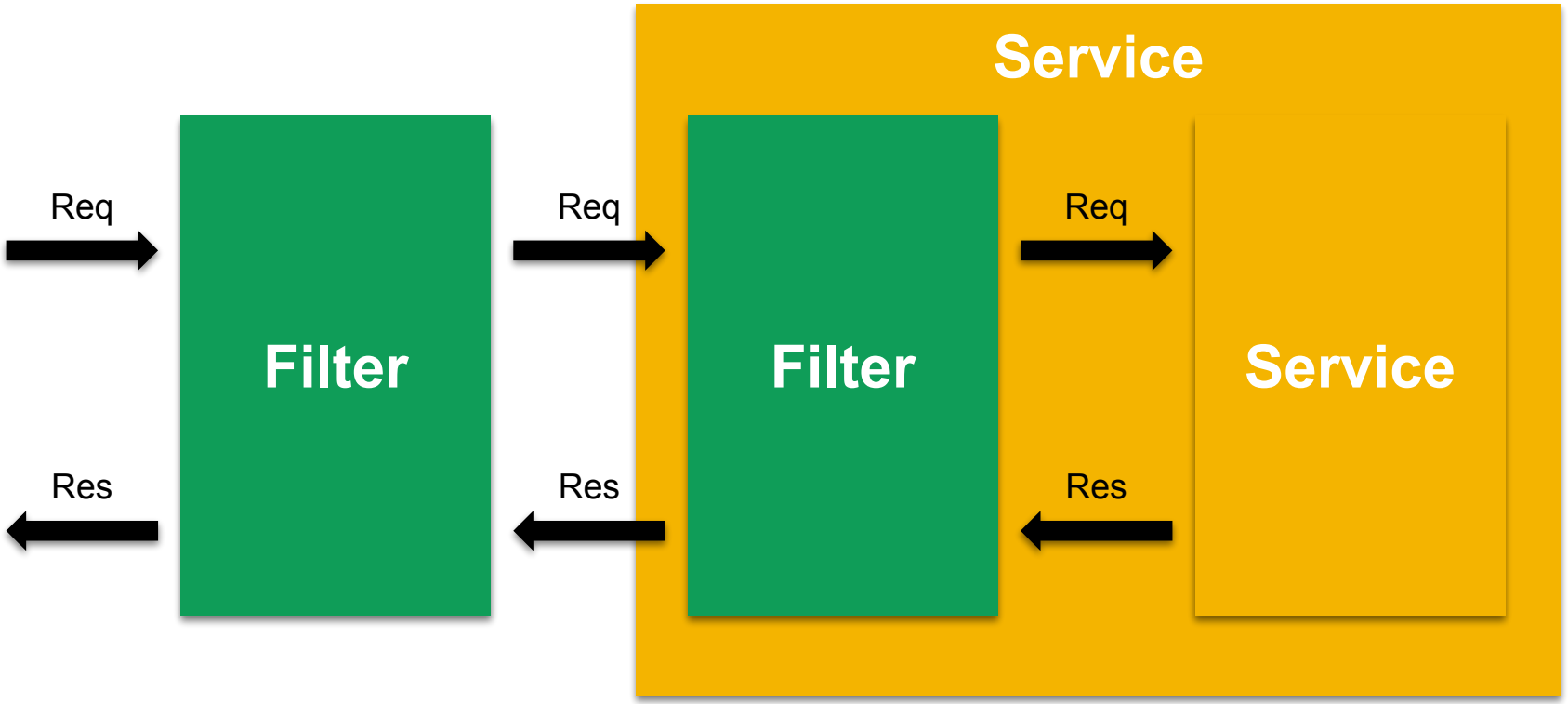
Go kit

- https://github.com/go-kit/kit
- マイクロサービスのためのツールキット
- いくつものコンポーネントとのアダプタを提供
  - Open Census, Zipkin, Prometheus, Graphite etc…
- ScalaのFinagleに近い思想

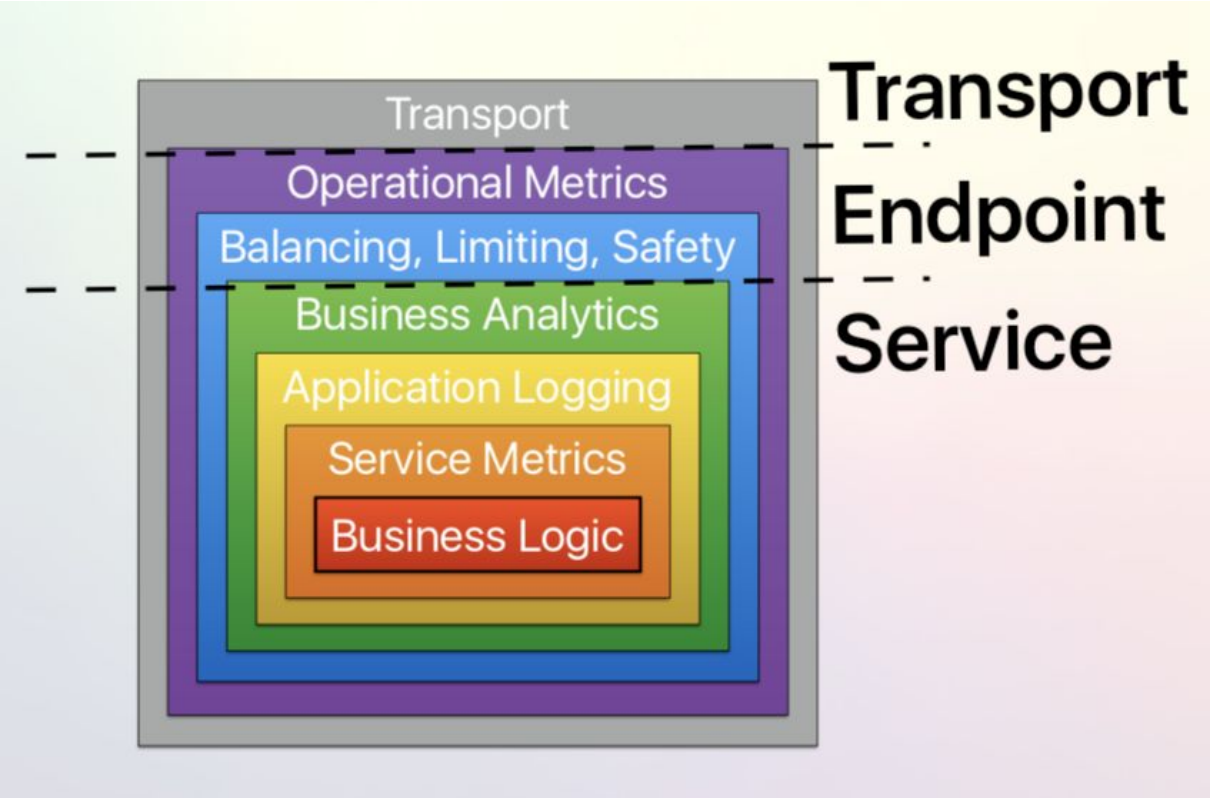
Finagleの思想



Finagleの思想



Go kitの思想


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

Go kitの思想

- Service
  - Business logicを書くところ
- Endpoint
  - Serviceを抽象化するところ
- Transport
  - HTTPやgRPCからEndpointを呼び出すところ

Service

- 通常のGoのinterfaceとして定義する

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

Endpoint

- Go kitが提供しているinterface
- RequestとResponseを interface{} で抽象化している

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

type Middleware func(Endpoint) Endpoint
```

Transport

- HTTP, gRPC, JSON-RPCなどが提供されている
- DecoderとEncoderを実装するとEndpointを呼び出してくれる

Transport (HTTP)

```
type DecodeRequestFunc func(context.Context, *http.Request)
                                    (request interface{}, err error)

type EncodeResponseFunc func(context.Context, http.ResponseWriter, interface{}) error

func NewService(
        e endpoint.Endpoint,
        dec DecodeRequestFunc,
        enc EncodeResponseFunc,
        options ...ServerOption,
) *Server // implements http.Handler
```

Transport (gRPC)

```
type DecodeRequestFunc func(context.Context, interface{})
                                    (request interface{}, err error)

type EncodeResponseFunc func(context.Context, interface{})
                                     (response interface{}, err error)

func NewService(
        e endpoint.Endpoint,
        dec DecodeRequestFunc,
        enc EncodeResponseFunc,
        options ...ServerOption,
) *Server
```

Example

https://github.com/go-kit/kit/tree/master/examples/addsvc/pkg

Workshop Exercise:

Sympatico

https://github.com/peterbourgon/sympatico

Sympaticoを使った演習

- SympaticoはDNAを管理するサービス
- 演習の解答が1 commitずつpushしてある
  - DNAのバリデーションを追加せよ
  - Prometheusを導入せよ etc...
- 1プロセスを2プロセスのマイクロサービスに分割
- Go kitを導入

Sympaticoの感想

- 2パッケージを1プロセスから2プロセスにした感じなのであまりマイクロサービス化の参考にはならない
- Go kitを使うためのサンプルとしてならよさそう
- Goの書き方として面白い部分があったので紹介

パッケージ構成

- main以外をinternalパッケージに突っ込んでいる
  - 別のプロジェクトからは参照不可
  - IDEなどの補完にも出ない
- Flat package
  - 境界づけられたコンテキスト
  - authにserviceからtransportまで入っている

- cmd
- internal
  - auth
    - service.go
    - endpoints.go
    - transport.go
  - dna
  - ctxlog
  - usage

ブロックで初期化

- 初期化したい変数を定義
- ブロックを作って初期化

- 一時変数を閉じ込める
  - ブロック ⇔ 関数

```
var authrepo *auth.SQLiteRepository
{
	var err error
	authrepo, err = auth.NewSQLiteRepository(*authURN)
	if err != nil {
		logger.Log("during", "auth.NewSQLiteRepository", "err", err)
		os.Exit(1)
	}
}

var authsvc *auth.Service
{
	authsvc = auth.NewService(authrepo, authEventsTotal)
}

var api http.Handler
{
	r := mux.NewRouter()
	r.PathPrefix("/auth/").Handler(http.StripPrefix("/auth",
auth.NewHTTPTransport(authsvc)))
	api = ctxlog.NewHTTPMiddleware(r, logger)
}
```

まとめ

- マイクロサービス化にあたって
  - DDD, Event Stormingでモデリング
  - Go kitで実装
- 個人的には interface{} はできるだけ避けたいので
  Go kitは使わなさそう...

参考資料

- Architecture & Domain Modeling with Go Kit
  - https://gophers.slack.com/archives/CCF41LFDW/p1535410195000100
- Go kit
  - https://github.com/go-kit/kit
- Go kit - Architecture and design
  - https://gokit.io/faq/#design-mdash-how-is-a-go-kit-microservice-modeled
- Sympatico
  - https://github.com/peterbourgon/sympatico