JAPAN | JUNE 21, 2024

aws SUMMIT

DOL-18

今、あらためて考える API デザイン

Paul

Solutions Architect
Amazon Web Services Japan

どのような API を
利用/開発されていますか？

REST API

Async API

gRPC

GraphQL

SOAP

その API は何故
そのデザインなのでしょう？

このセッションのゴールやスコープ

ゴール

API デザインを考える上で必要な観点、フローで考えるのかを知る

お話すること

- API デザインのアプローチの解説
- API デザインを行う上での考慮ポイント

お話しないこと

- API 方式の基本的な考え方や用語解説
- AWS サービスの基本的な機能解説

自己紹介

ポール

𝕏 @bulbulpaul

アマゾン ウェブ サービス ジャパン合同会社
ソリューションアーキテクト

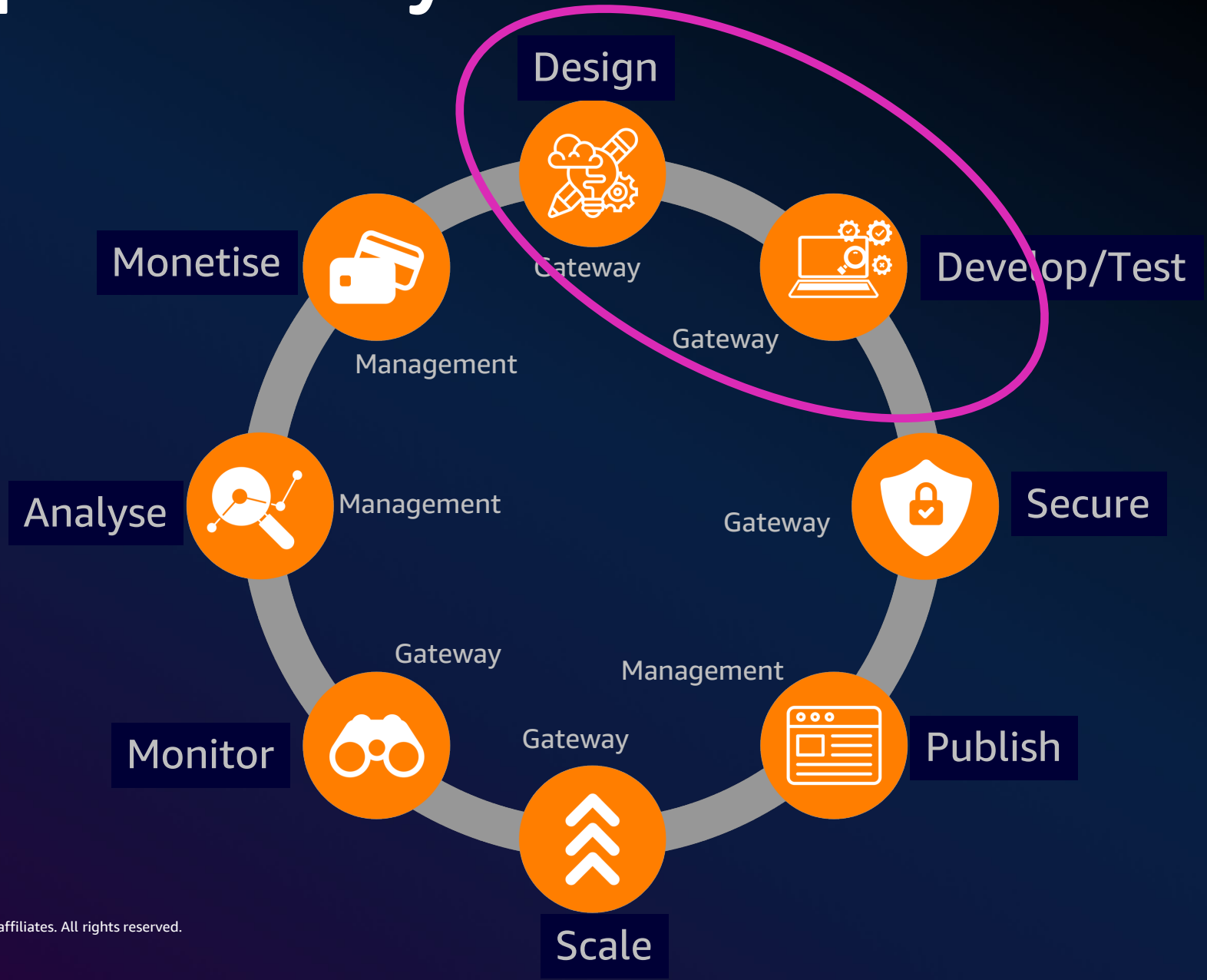
[関心のある技術]
Java, Kotlin, Python, Serverless, 釣り

Agenda

- API ライフサイクルにおける API デザインの重要性
- API デザインで考慮すべきこと
- まとめ

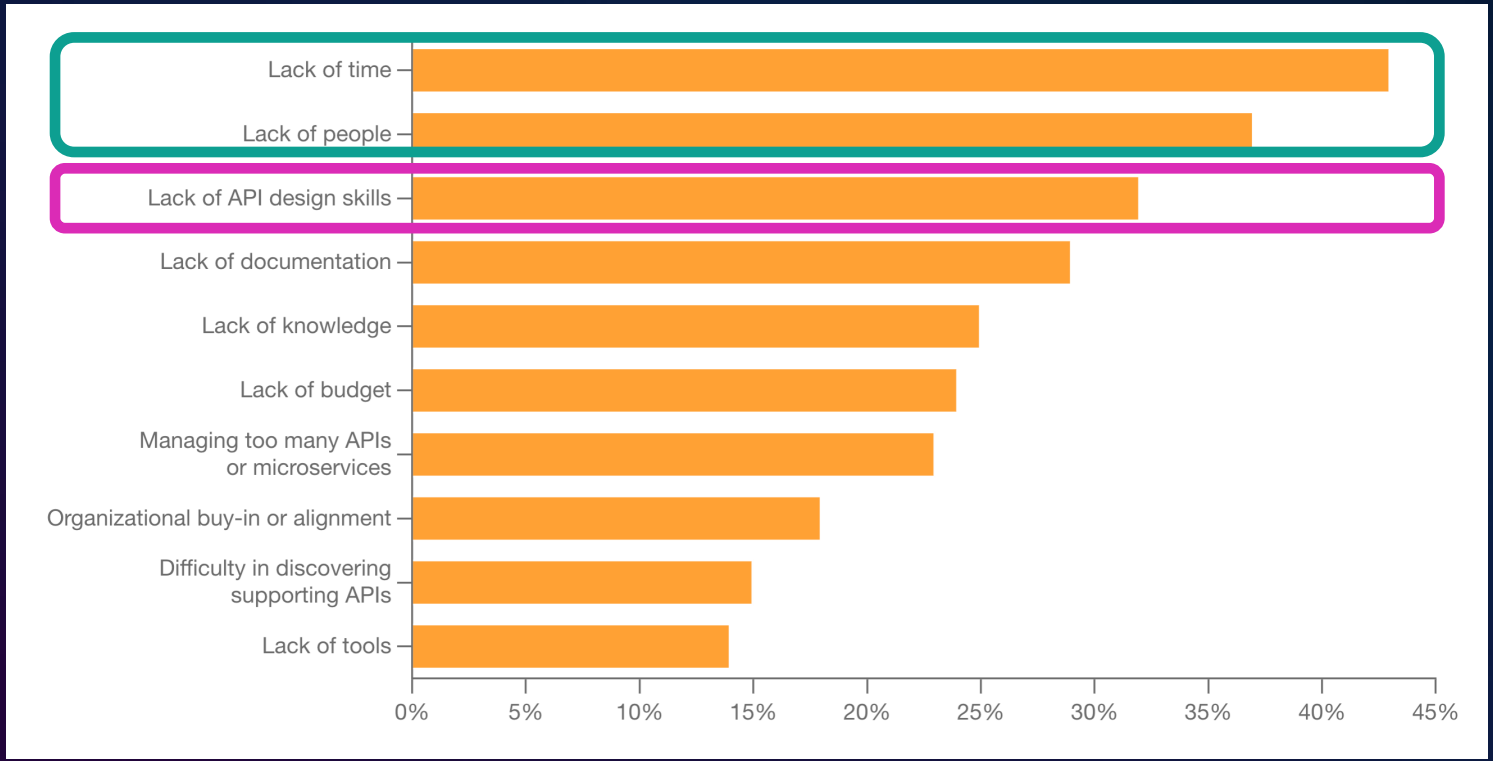
API ライフサイクルにおける API デザインの重要性

API development lifecycle



API デザイン の重要性

API 開発を行う際の障壁となるもの



人・時間の欠如

API デザインスキルの欠如

一方、
開発者の多い企業では

API デザインスキルの欠如が一位に

引用: State of the API Report

API は単なるインタフェースではない

さまざまな業態、行政機関のシステムでも API を利用した
サービス提供や、業務の効率化等を行っている

→ API はソフトウェアやビジネスの重要な構成要素

API をソフトウェアの重要な構成要素として扱う

API を優先した設計アプローチが重要に

API ファーストでのアプローチ

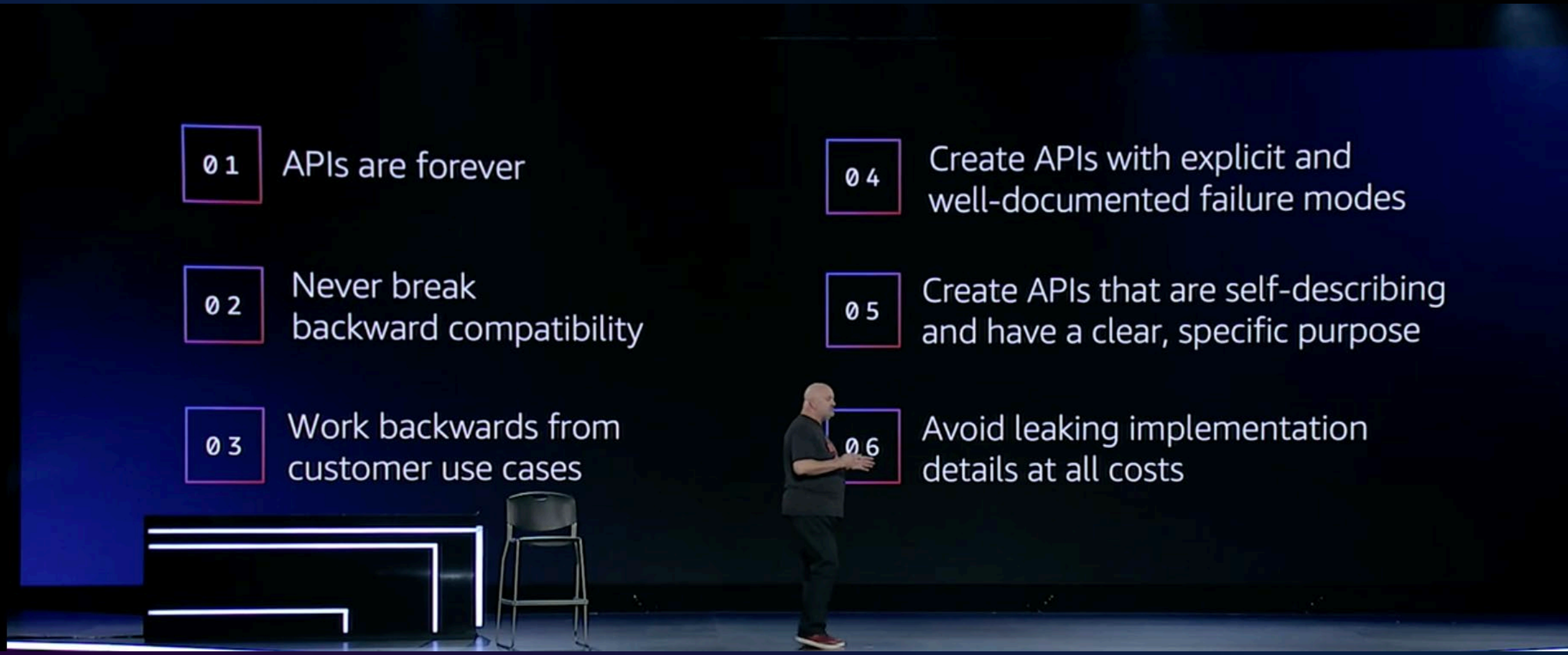
ソフトウェア開発プロセスの開始時に API を優先して設計し
API を重要な構成要素として位置付けて進める

メリット
- 要件・設計などの問題の早期発見や不確実性に対する柔軟性
- クライアントとサーバーサイドのロジックの並行開発が可能
- 市場投入までの時間の短縮

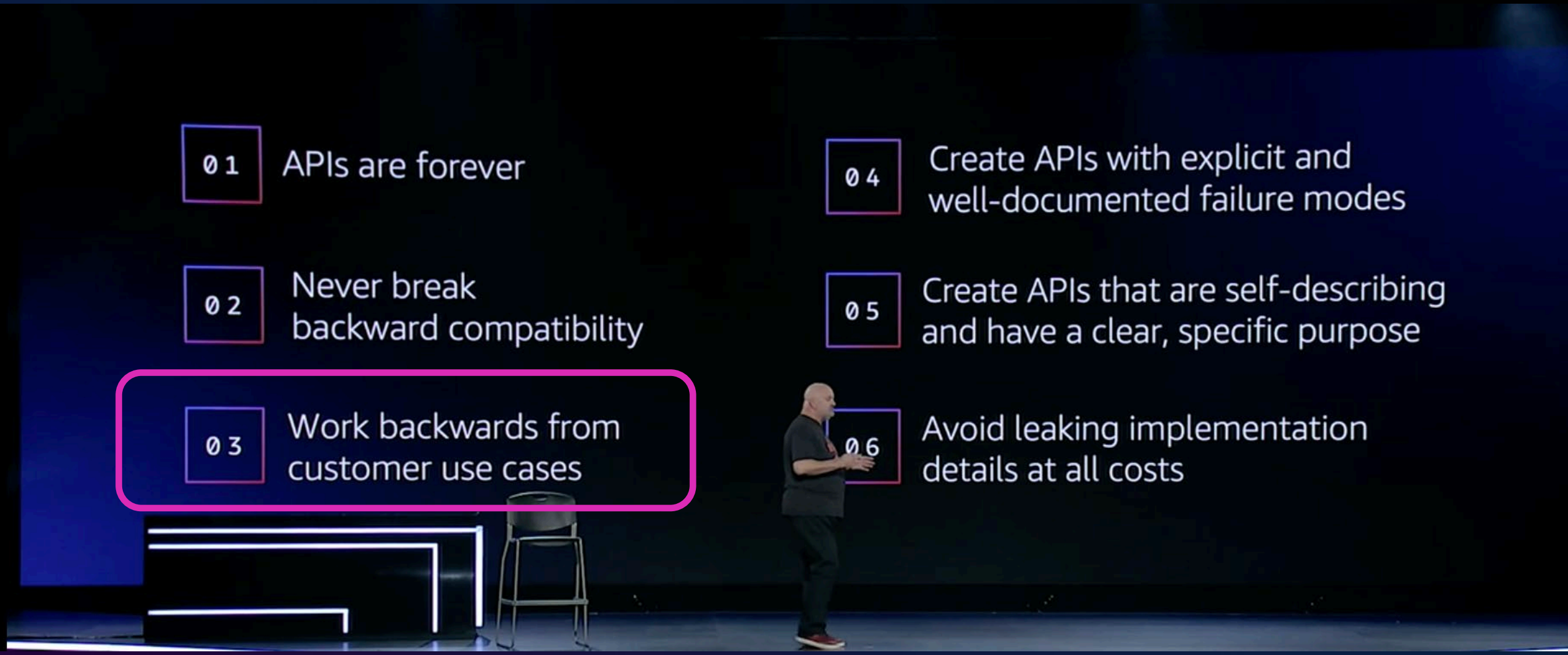
デメリット
- 設計フェーズに時間を要する

API デザインで考慮すべきこと

API advice from Werner Vogels (CTO, Amazon)



API advice from Werner Vogels (CTO, Amazon)



カスタマーユースケースから逆算で考える

コードやリソースからの設計

```
// 1. Blog サービスの疑似コード
class Posts {
  createComments() { ... }
  addLikes() { ... }
}
```

```
// 2. メソッドとAPI のマッピング
POST  /posts/{post_id}
{"type": "comment", "data": ...}

POST  /posts/{post_id}
{"type": "likes", "data": ...}
```

ユースケースからの設計

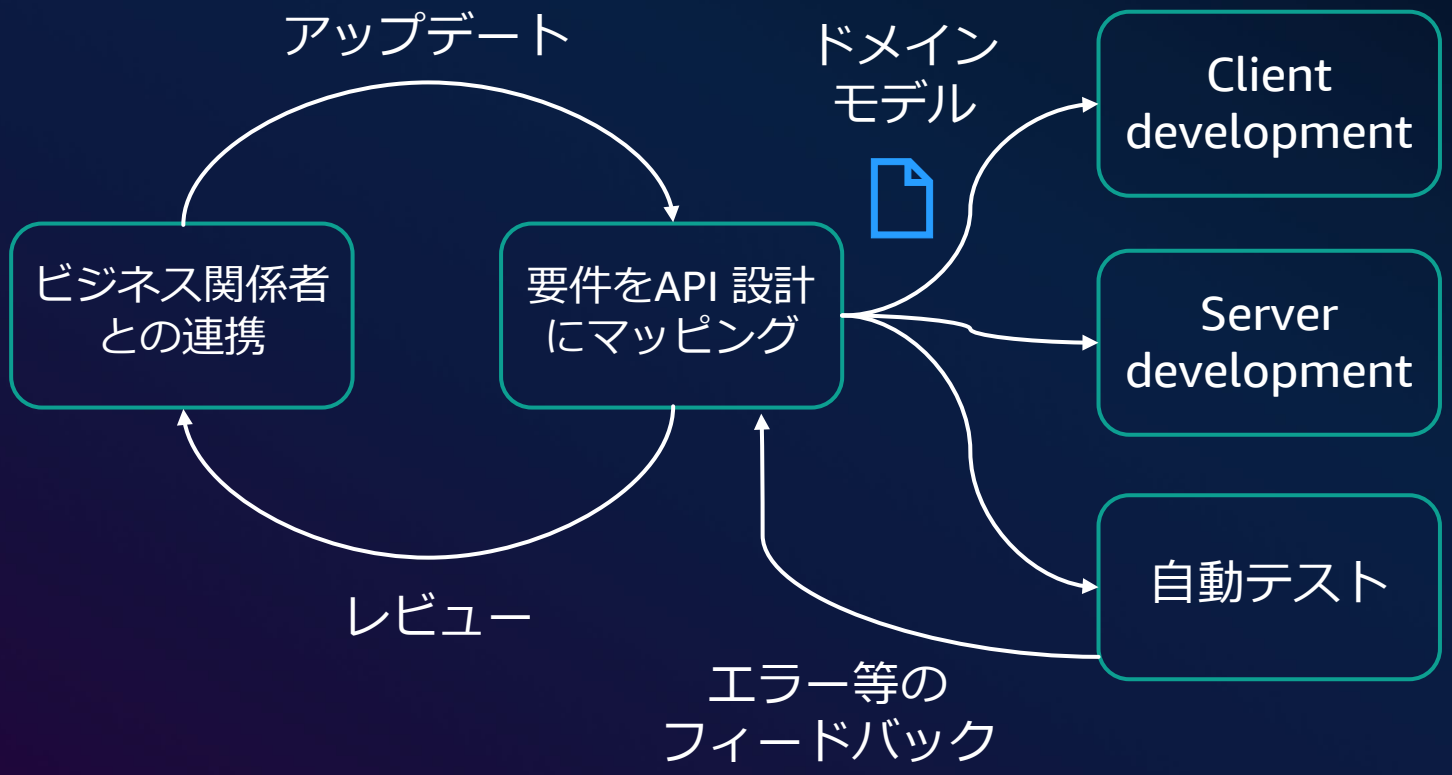
```
// 1-1. ある投稿にコメントをしたい
POST /posts/{post_id}/comments

// 1-2. ある投稿にLikeをしたい
POST /posts/{post_id}/likes
```

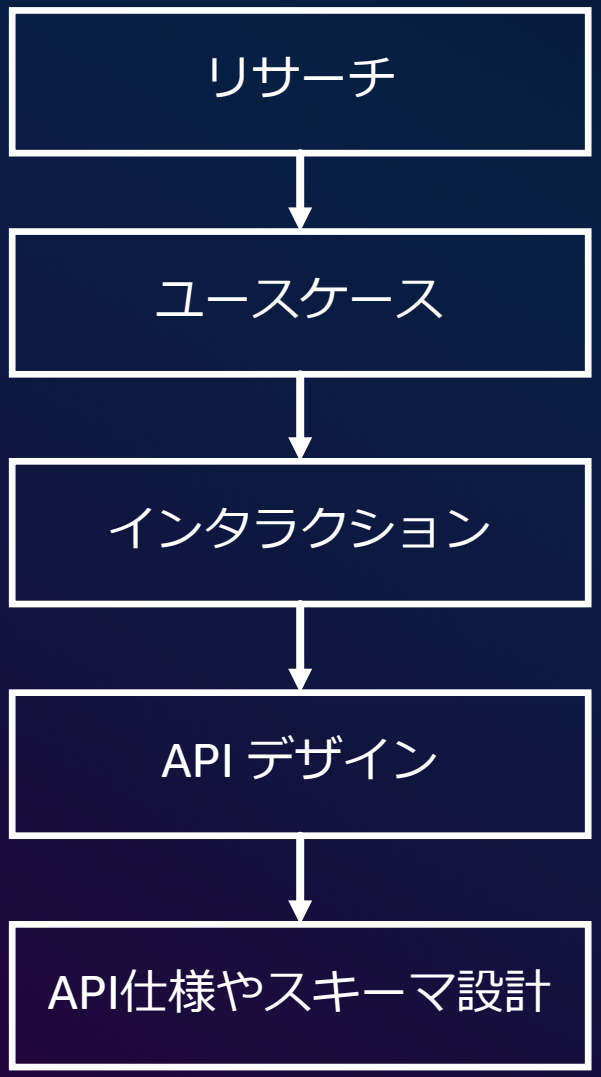
```
// 2. Blog サービスの疑似コードの設計
createCommentOnPost(postId) { ... }
likePost(postId) { ... }
```


https://www.thoughtworks.com/insights/blog/rest-api-design-resource-modeling

ユースケース起点での開発フロー



ユースケース起点でのデザインフロー



ユーザー調査（ユーザーは誰？

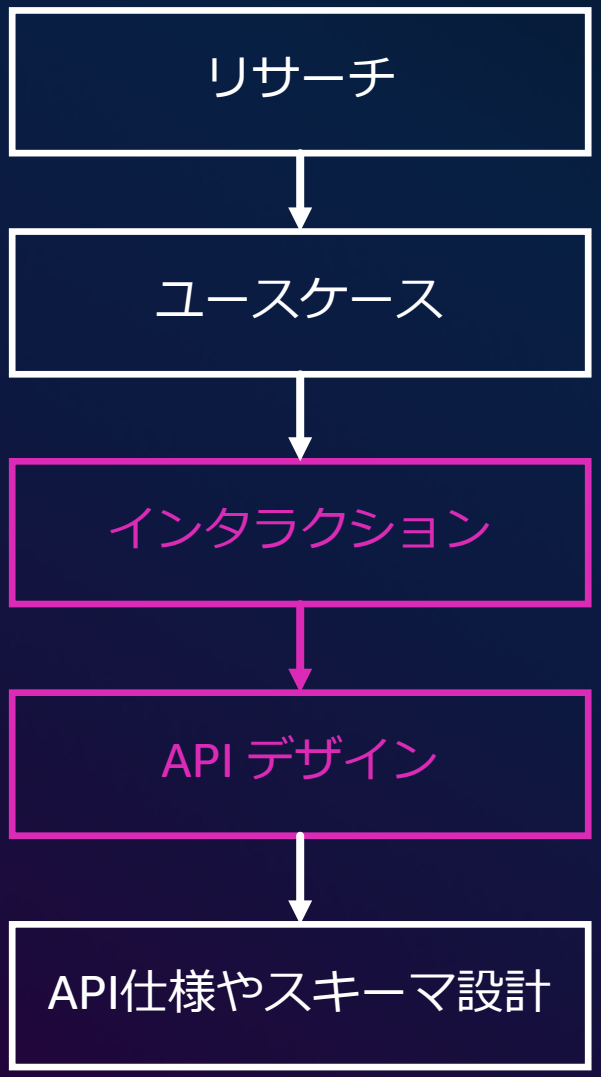
ユースケースの整理
（どのような環境やシーンで利用するのか

ユースケースに適合したAPIインタラクション
(e.g. Protocol, Sync /Async)

ユースケースを起点とした設計

仕様やスキーマの設計
(e.g. JSON Schema, OpenAPI)

ユースケース起点でのデザインフロー



ユーザー調査（ユーザーは誰？

ユースケースの整理
（どのような環境やシーンで利用するのか

ユースケースに適合したAPIインタラクション
(e.g. Protocol, Sync /Async)

ユースケースを起点とした設計

仕様やスキーマの設計
(e.g. JSON Schema, OpenAPI)

API Design : インタラクション

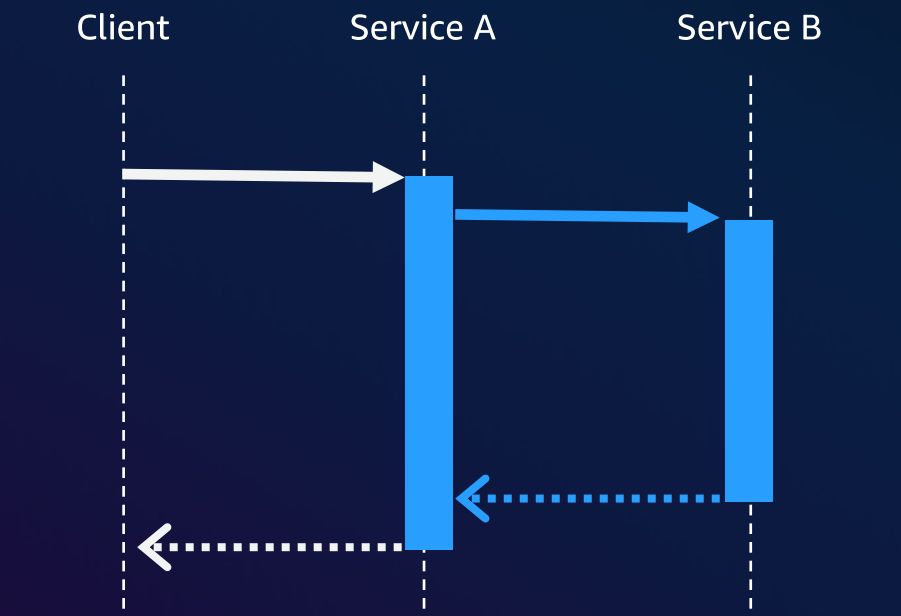
RESTful: HTTP APIs
& REST APIs

WebSocket APIs

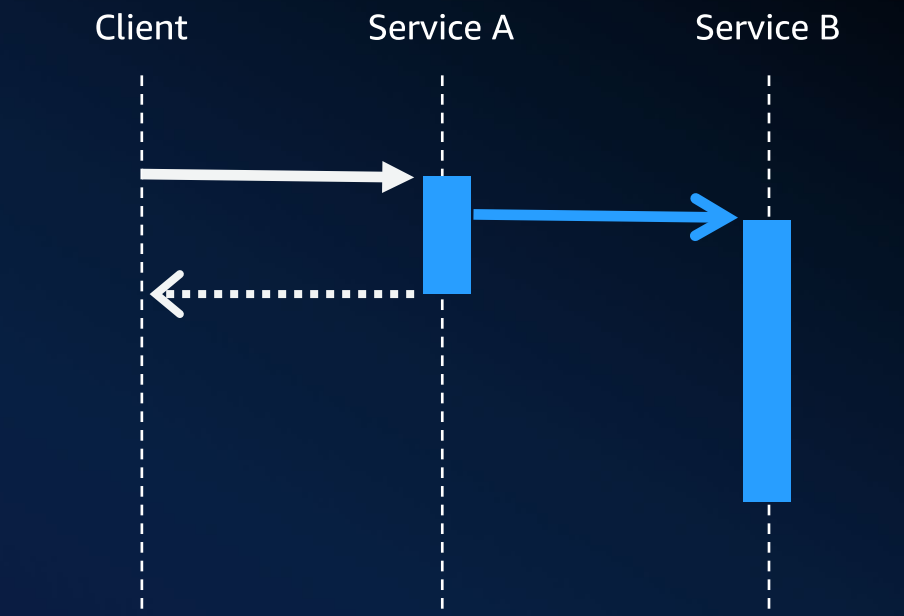
- リクエスト / レスポンス
- 単方向のコミュニケーション
- Short-lived コミュニケーション
- ステートレス

- 永続的な接続
- 双方向のコミュニケーション
- Long-lived コミュニケーション
- ステートフル

同期・非同期処理でのインタラクション

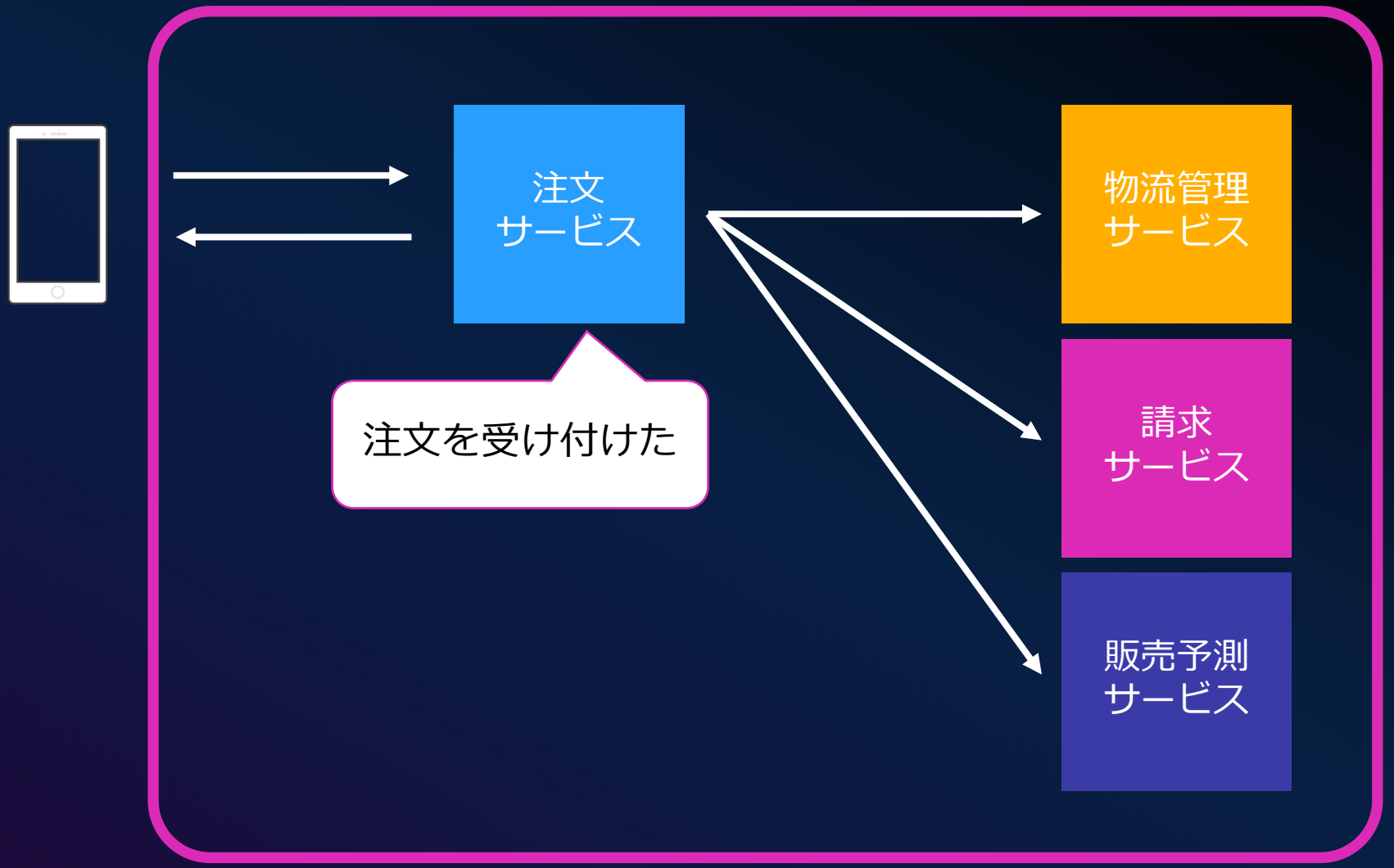


同期的 Command



非同期的 Event

商品購入のユースケース（同期 API）



商品購入のユースケース（同期 API）

注文
サービス

物流管理
サービス

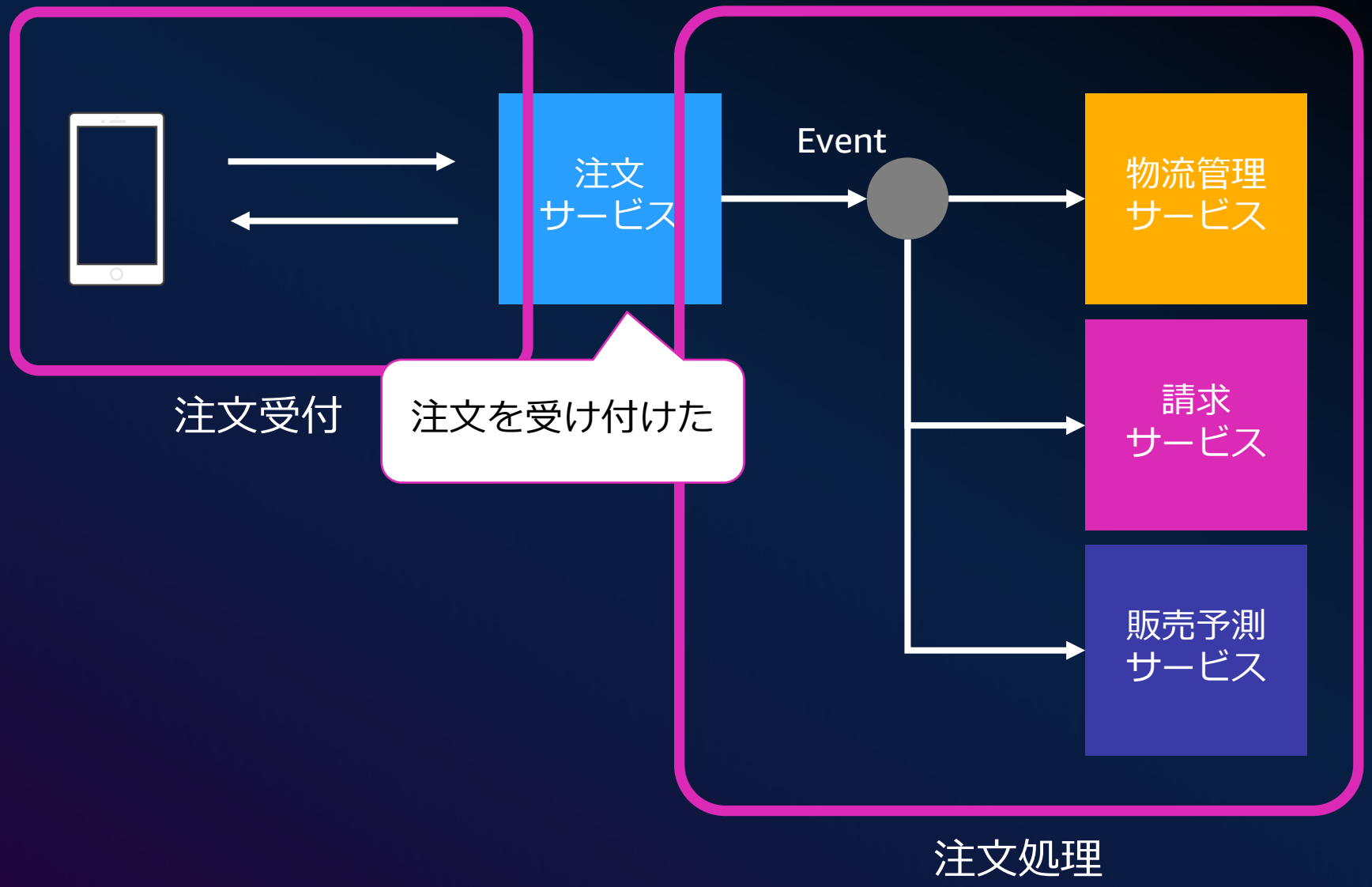
請求
サービス

販売予測
サービス

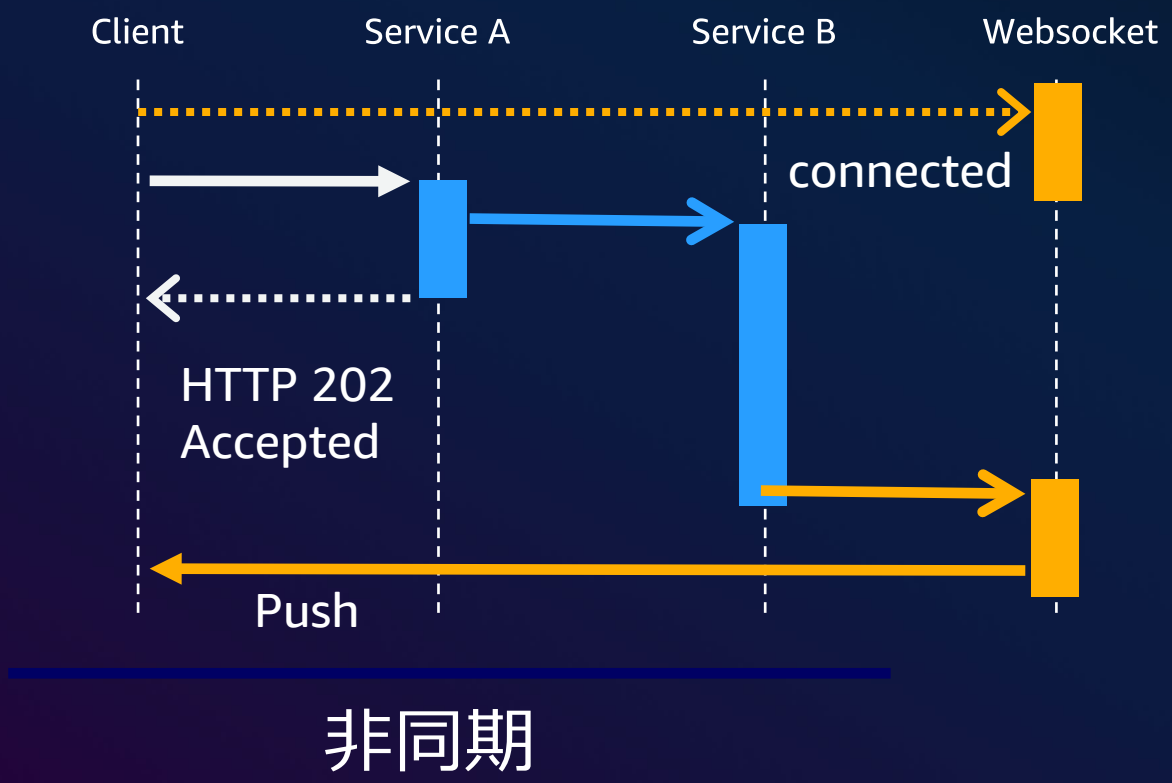
「注文の受付」が担保できるとよいのでは？
「全ての注文関連処理の完了」はすべて同期的である必要性はあるのか？

後から非同期でのメール通知でも良いのでは？

商品購入のユースケース（同期 API）

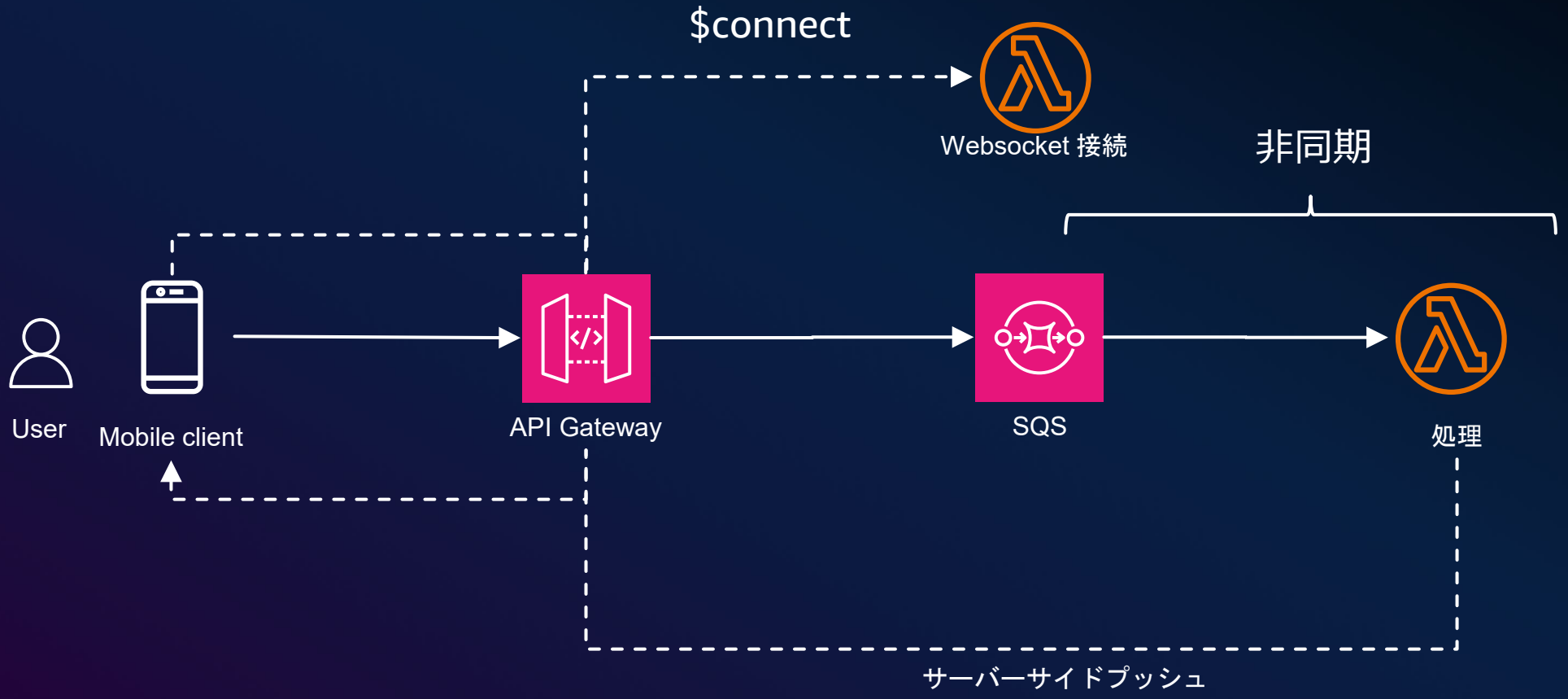


API x Websocket による応答性の向上



- Websocket によるサーバーサイド Push
- 処理受付と完了を非同期とすることで、クライアントへの応答性を向上させる

API x Websocket による応答性の向上

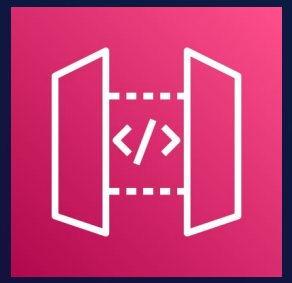


API 方式の検討（一例

	REST	GraphQL	gRPC	Async API*
About	Architecture pattern for W3	Data query and manipulation language	Remote Procedure Call (RPC) framework	Event-driven architecture (EDA)
Protocol	HTTP(s)	HTTP(s), WebSocket	HTTP/2	WebSocket, MQTT, Kafka, etc.
Interaction	Sync	Sync and async	Sync and async*	Async batch and real-time
Dev experience	SDK or HTTP library	Generated code	Protobuf	SDK based on protocol
Documentation	API contract	Schema	Service contract	Event schema

** Uses http streaming*

API 構築におけるAWSサービス（一部）



Amazon
API Gateway

AWS AppSync

Application
Load Balancer

Amazon VPC
Lattice

AWS Lambda
(Function URL)

API Design : 必要な要素のマッピング

コンシューマーの種類

Mobile client

Websites

Services

Lambda 関数

Any other AWS service

Endpoints in VPC

Endpoint type?

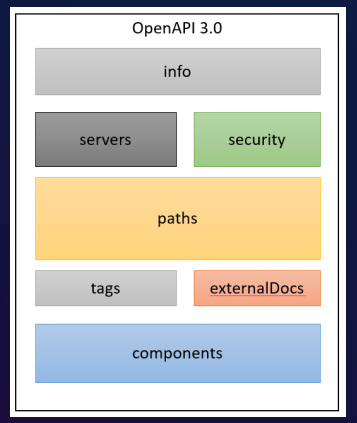
Enable Cache?

Protocol option?

Request & Response interface? (i.e Open API Spec)

Amazon API Gateway

Interface spec?



統合先の種類

Lambda Functions

Public HTTP Endpoints

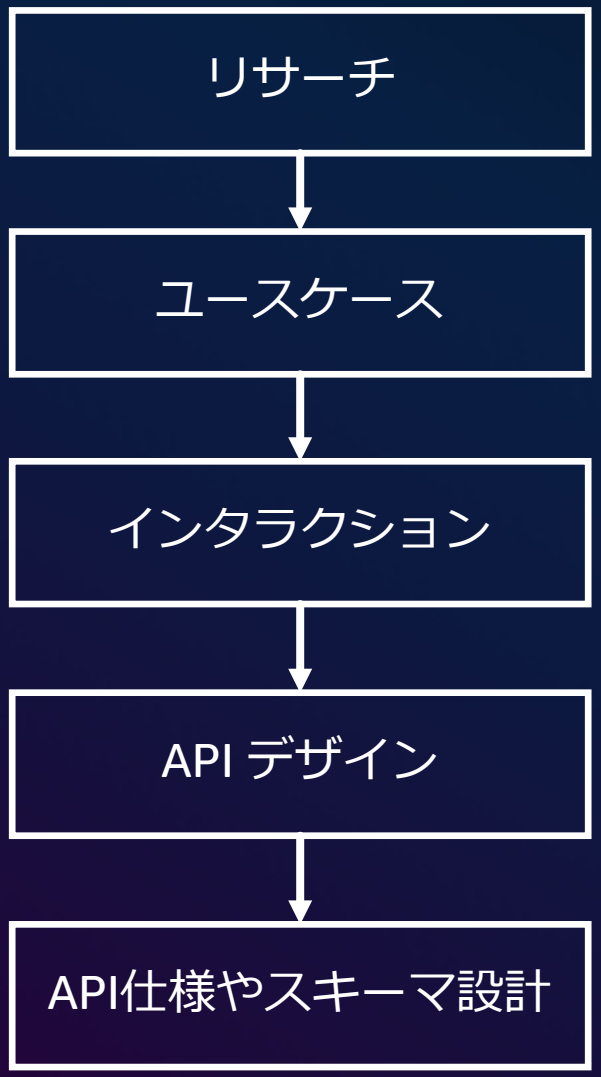
Any other AWS service

Endpoints in VPC

AWS Direct Connect

On-premises

ユースケース起点でのデザインフロー



ユーザー調査（ユーザーは誰？

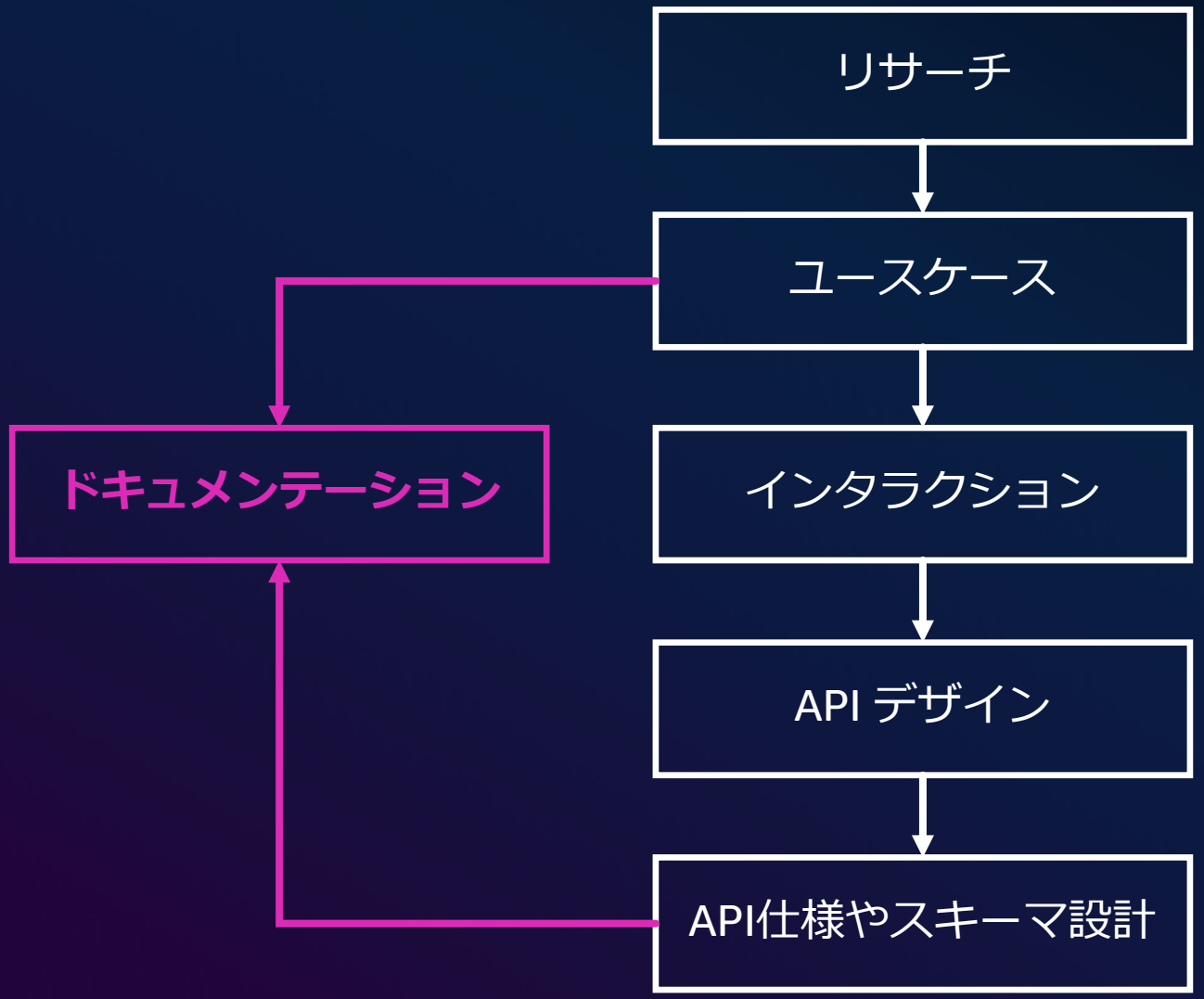
ユースケースの整理
（どのような環境やシーンで利用するのか

ユースケースに適合したAPIインタラクション
(e.g. Protocol, Sync /Async)

ユースケースを起点とした設計

仕様やスキーマの設計
(e.g. JSON Schema, OpenAPI)

ユースケース起点でのデザインフロー



ユーザー調査（ユーザーは誰？

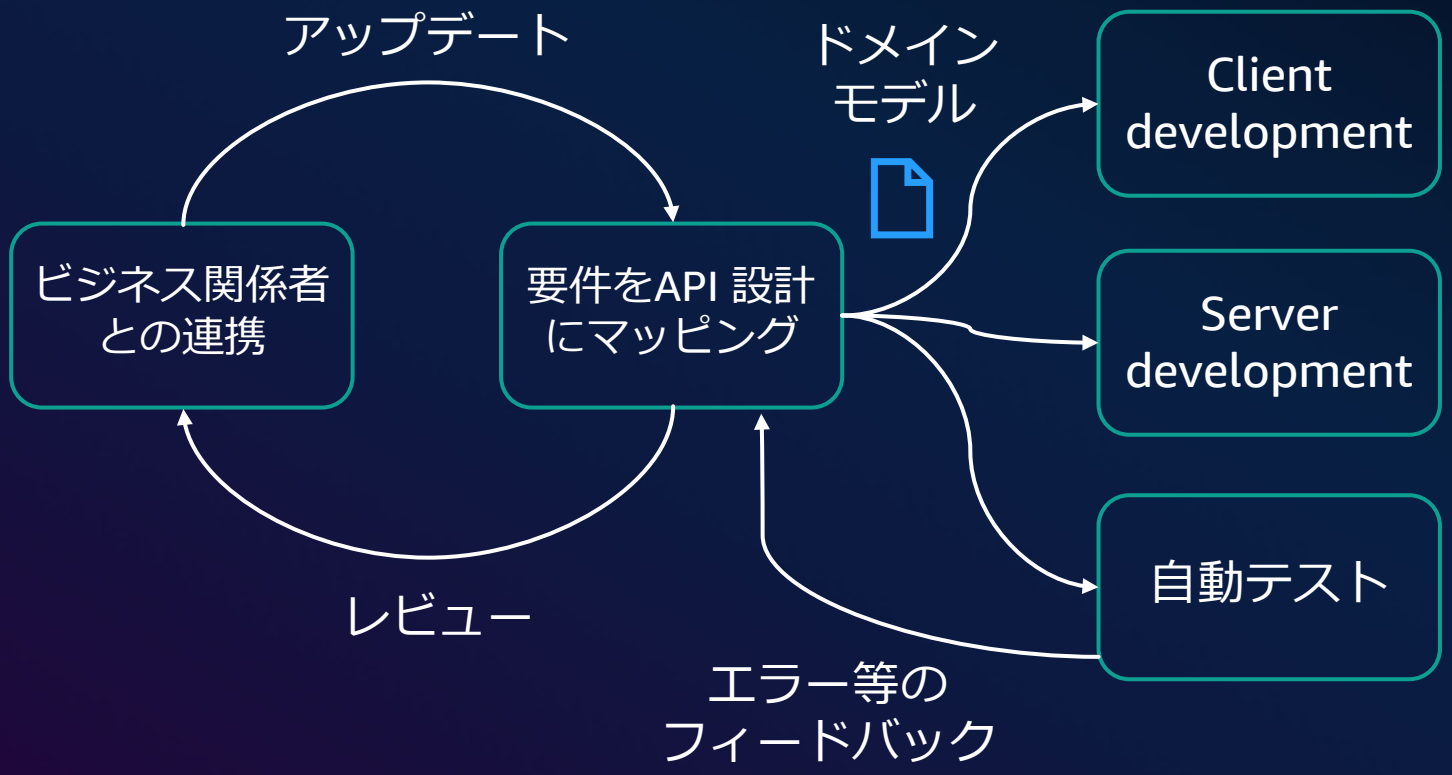
ユースケースの整理
（どのような環境やシーンで利用するのか

ユースケースに適合したAPIインタラクション
(e.g. Protocol, Sync /Async)

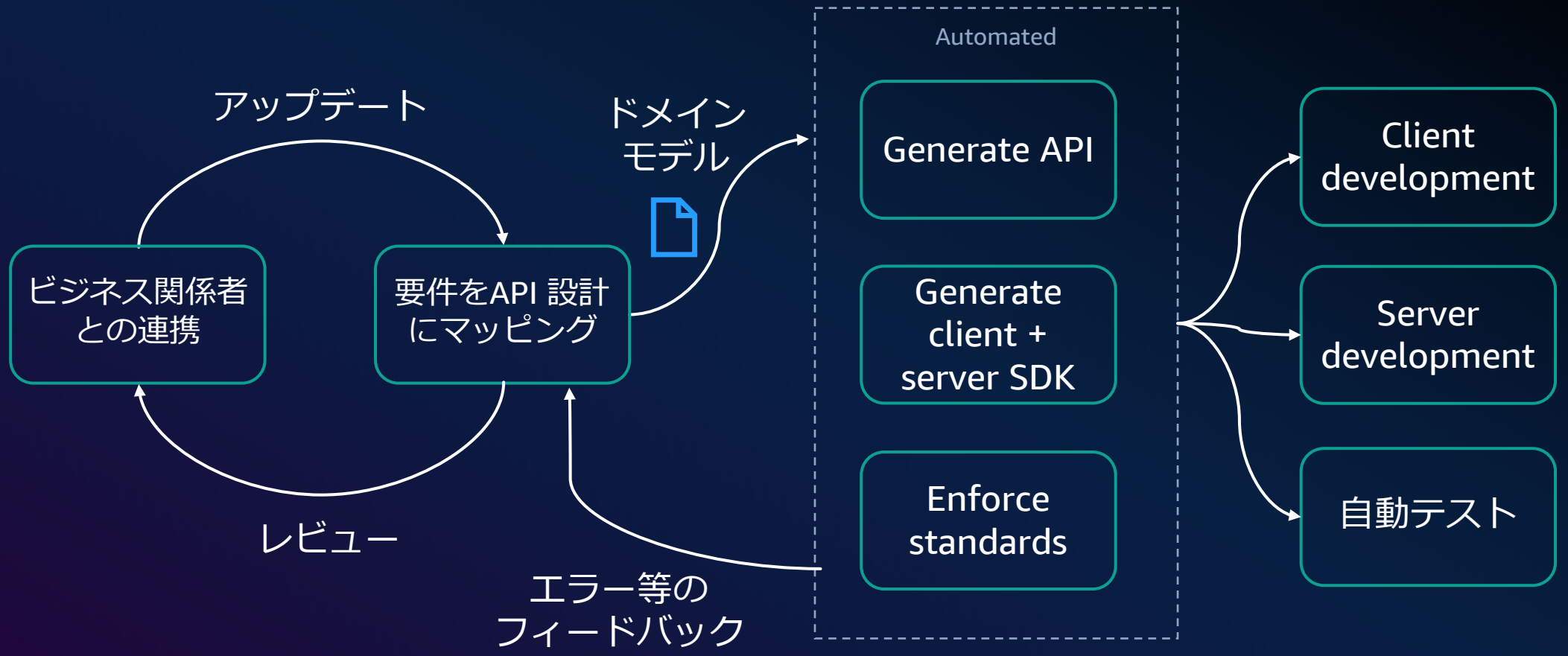
ユースケースを起点とした設計

仕様やスキーマの設計
(e.g. JSON Schema, OpenAPI)

ユースケース起点での開発フロー



ユースケース起点での開発フロー



他にも API デザインで考慮すべきこと（一例）

API スタンダード

- 標準仕様への準拠
- スタイルガイド等

持続可能性

- 下位互換性
- バージョニング

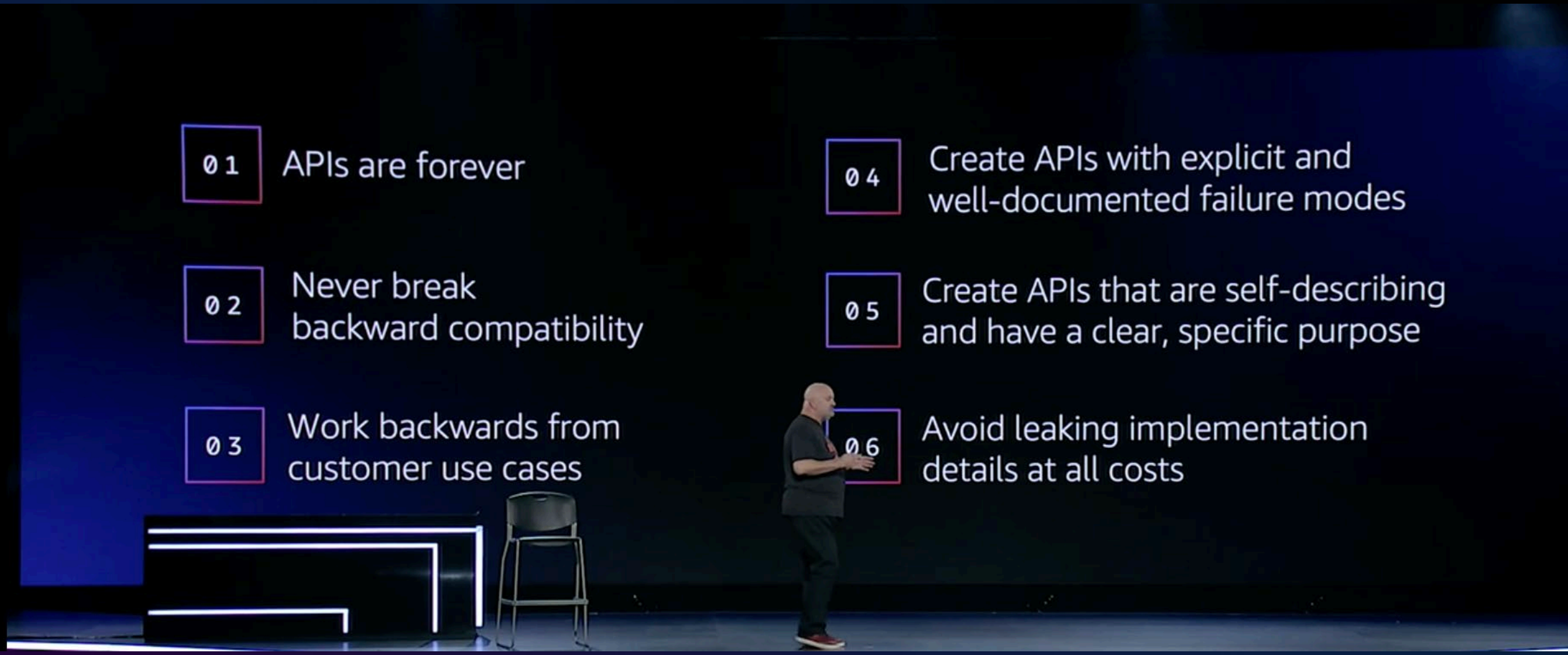
非機能要件

- セキュリティ
- 性能, 拡張性

他にも技術選定に影響を及ぼす要素

- Developer experience
- チームのスキルセット
- 組織のバックグラウンド, etc...

API advice from Werner Vogels (CTO, Amazon)



まとめ

まとめ

- API ファーストでデザインや仕様に焦点をあてて考える
- まずはユースケースから API デザインを考える
- 必要なインタラクションや技術選定、仕様への落とし込み
- ユースケースを意識した仕様、ドキュメンテーションも重要

Thank you!

Paul
@bulbulpaul