分散アーキテクチャにおける
統一的なデータバリデーションを実現するミドルウェア
Open-VE

SecHack 365 開発駆動コース 仲山ゼミ 渋谷和樹

shibukazu/open-ve

@s_k_526

京都大学情報学研究科 知能情報学専攻
Openly 合同会社
医療介護共創基盤株式会社

I’m Kazuki Shibutani

Education Background

東北大学工学部電気情報物理工学科
量子アニーリングを用いた離散数理最適化アルゴリズムの研究
京都大学情報学研究科知能情報学専攻
深層学習を用いた音声認識モデルにおける効率的なファインチューニング

Entrepreneurship

Openly 合同会社
音声認識分野における研究成果を用いたアプリケーション開発
医療介護共創基盤 株式会社
大学病院をはじめとした大規模病院向けシフト管理アプリ GauDiの開発運用
京都大学病院を中心として京都・大阪の大規模病院へ導入

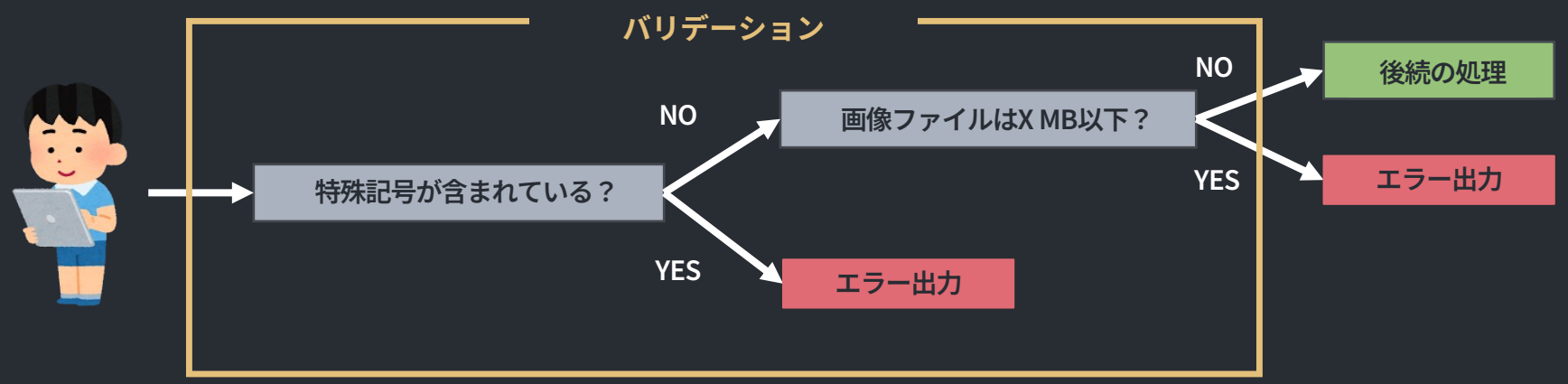
Job (2025/4/1 ~)

株式会社 LayerX
ソフトウェアエンジニア（プラットフォームエンジニア）

バリデーションの重要性

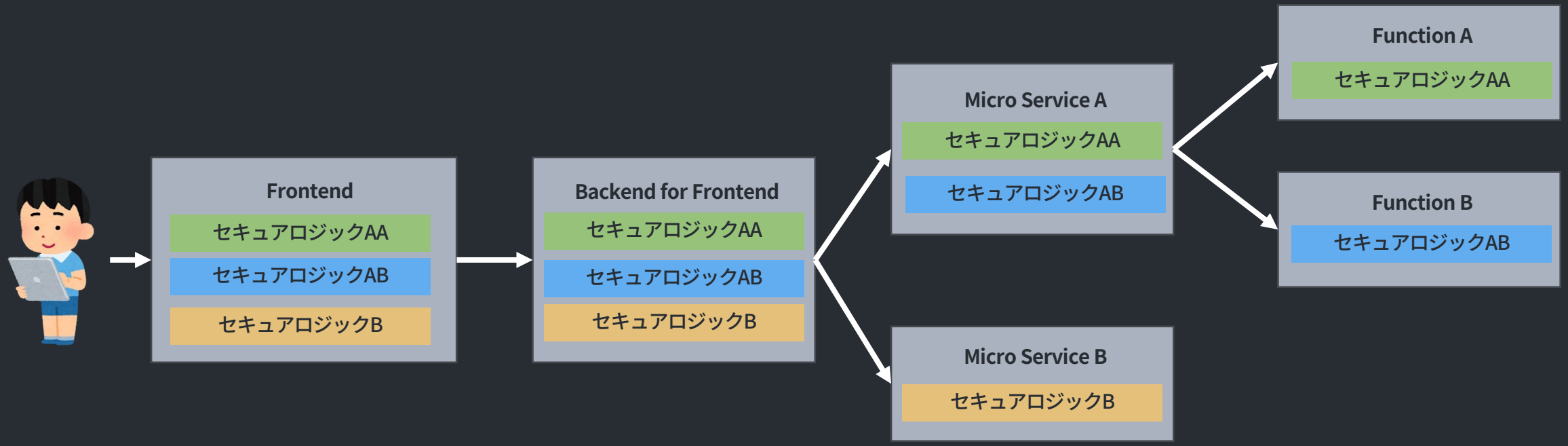
- バリデーションとは
  ユーザーの入力を検証する処理の総称
  (例) ユーザー名の入力において特殊記号を弾く
  　　SQLインジェクションやXSSなどの原因となる入力を防ぐ
  (例) プロフィール画像の入力において巨大な画像ファイルを防ぐ
  　　ファイルストレージが一部のユーザーによって圧迫されることを防ぐ

安全なウェブアプリを開発する上では適切なバリデーションが不可欠



ウェブアプリ開発における分散化と中央集権的アプローチの台頭

- マイクロサービスやサーバレスをはじめとしてウェブアプリが分散化、多層化
- 認証認可やバリデーションなどセキュリティ上重要なロジックが各システムに分散することで実装漏れや重複実装などの問題が顕在化
- 認証認可において中央集権的なミドルウェアとしてGoogleのZainzibarが一定の成果を挙げる



中央集権型データバリデーションエンジン「Open-VE」の概要

バリデーションロジックの中央集権管理

・各システムごとに実装していたバリデーションロジックを中央集権的に管理
・システム間の実装ミスマッチや重複実装を防ぐことで効率的な実装をサポート

言語非依存のロジック実装・管理

・CELを用いたロジック記述により言語やフレームワークに依存しないロジック実装をサポート
・言語の差異によるバリデーションのミスマッチを防ぐ
(例) Goにおいては文字数カウントにruneを使わなくてはいけない

API経由でのバリデーション管理および実行

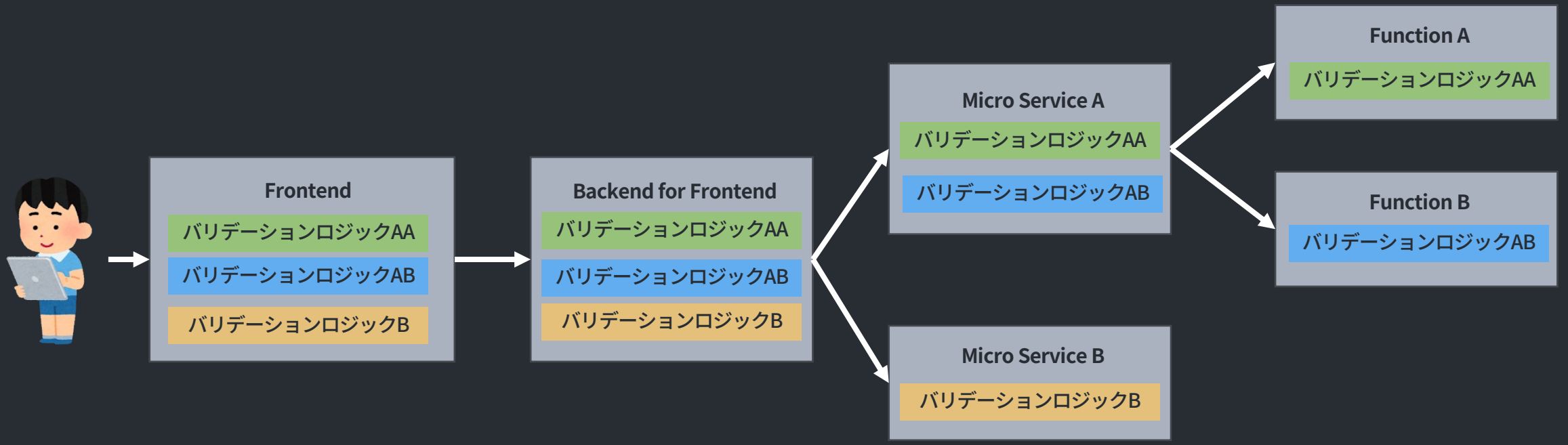
・HTTP APIおよびgRPC経由でバリデーション関連の操作を実行可能
・言語に依存せず同一のロジックで実行されることが保証される

VS. コード生成アプローチ (protovalidate)
・CELから各言語ごとのバリデーションコードを生成するコード生成アプローチが存在
・言語ごとに生成をサポートする必要があり、汎用性が低い
・バリデーションごとにネットワークリクエストが不要というメリット → 高いパフォーマンスで対処

Key1 : 中央集権型のバリデーションロジック管理

これまでのバリデーションロジック管理

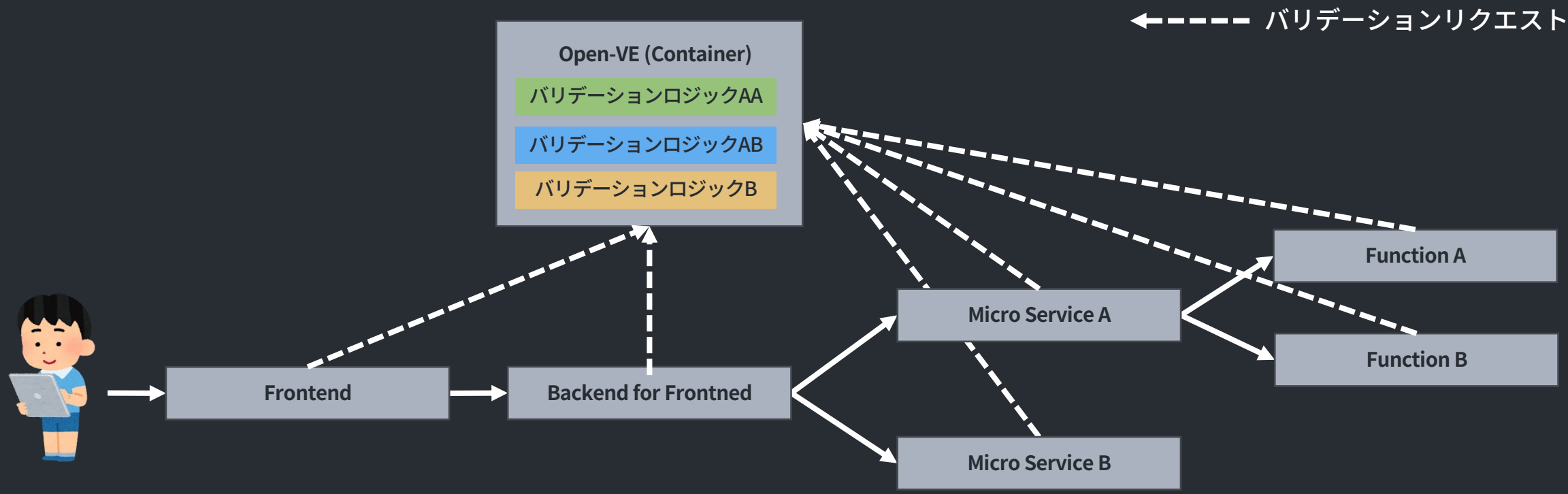
分散システムでは同一のデータが複数のシステムを流れる
そのため、各システムごとに類似のバリデーションロジックを個別に実装・管理する必要



Key1 : 中央集権型のバリデーションロジック管理

これからのバリデーションロジック管理

すべてのバリデーションロジックをOpen-VEで一括管理
各システムは自身の責務の実装に集中できる



Key2 : APIでのバリデーション問い合わせ

3種類のAPIを提供

- POST /v1/dsl バリデーションロジックの登録
- GET /v1/dsl 登録済みバリデーションロジックの確認
- POST /v1/check バリデーションロジックの実行

バリデーションロジックの登録（一部省略）

```
curl --request POST \
  --url http://localhost:8080/v1/dsl \
  --data '{
  "validations": [
    {
      "id": "item",
      "cels": [
        "price > 0",
        "size(image) < 360"
      ]
    }
  ]
}'
```

商品価格は正

商品画像は360バイト未満

バリデーション実行（リクエスト）

```
curl --request POST \
  --url http://localhost:8080/v1/check \
  --data '{
  "validations": [
    {
      "id": "item",
      "variables": {
        "price": -100,
        "image": "iVB0Rw0kGgoAAASasaGYGfdGdH"
      }
    }
  ]
}'
```

負の商品価格

359バイトの画像

バリデーション実行（レスポンス）

```
{
"results": [
  {
   "id": "item",
   "isValid": false,
   "message": "failed validations: price > 0"
  }
 ]
}
```

バリデーション結果

商品価格に関するバリデーションに違反

Key3 : 高いレベルの開発者サポート

Open-VEスキーマの自動生成

Open APIスキーマまたは Protobuf から Open-VE スキーマの一部を自動生成
リクエストパラメータやスキーマの多重管理を防ぐ

Open APIスキーマ（一部省略）

```
{
"paths": {
  "/item": {
    "parameters": [
      {
       "id": { "type": "integer", "format": "int32" },
       "name": { "type": "string" },
       "price": { "type": "number", "format": "float"}
      }
    ]
  }
}
```

Open-VE スキーマ

```
validations:
- id: SampleObject
  cels: []
  variables:
  - name: SampleObject.id
    type: int
  - name: SampleObject.name
    type: string
  - name: SampleObject.price
    type: double
  testCases: []
```

Open-VEスキーマの自動テスト

Open-VE CLIを用いてOpen-VEスキーマの記述の正しさ、およびバリデーションの正確性をテスト可能
テスト駆動的なスキーマ記述が可能に

```yaml
validations:
- id: "price"
  cels:
  - "number > 0"
  variables:
  - name: "number"
    type: "int”
  testCases:
  - name: "invalid price"
    variables:
    - name: "number"
      value: 0
    expected: false
```

🧪 testing open-ve schema filePath=dsl.yml
✅ PASS : price
❌ FAILED : name
📊 Results: 1 passed, 1 failed, 0 not found

Key3 : 高いレベルの開発者サポート

GoとTypeScript向けの簡易SDKの提供

わずか数行のコードでバリデーション問い合わせが可能に

クライアントの定義

バリデーションリクエストの実行

TypeScript

```typescript
import { Client } from "open-ve-typescript-sdk/client";

const client = new Client({
url: "http://localhost:8081",
version: "v1",
});

const App = () => {
  const [name, setName] = useState("");
  const [isNameValid, setIsNameValid] = useState(false);
  const handleNameChange = async (e) => {
    const res = await client.check({
    validations: [
      {
        id: "product",
        variables: {
          name: e.target.value,
        },
      },
    ],
  }
  if (res.results[0].isValid) {
    setName(e.target.value);
    setIsNameValid(true);
  } else {
    setIsNameValid(false);
  }
};
```

Go

```go
import "github.com/shibukazu/open-ve-go-sdk/client"

var openveClient = client.NewClient(
                    client.NewConfig("http://localhost:8081"))
func main() {
  r.POST("/upload", func(c *gin.Context) {
    name, _ := c.GetPostForm("name")
    res, _ := openveClient.Check(&client.CheckRequest{
      Validations: []client.CheckRequestValidation{
        {
          ID: "product",
          Variables: map[string]interface{}{
            "name": name,
          },
        },
      },
    })
    if res.Results[0].IsValid {
      c.JSON(200, gin.H{
      "message": "Valid",
      })
    } else {
    c.JSON(400, gin.H{
      "message": res.Results[0].Message,
      })
    }
  })
```

Key4 : セキュリティとパフォーマンス

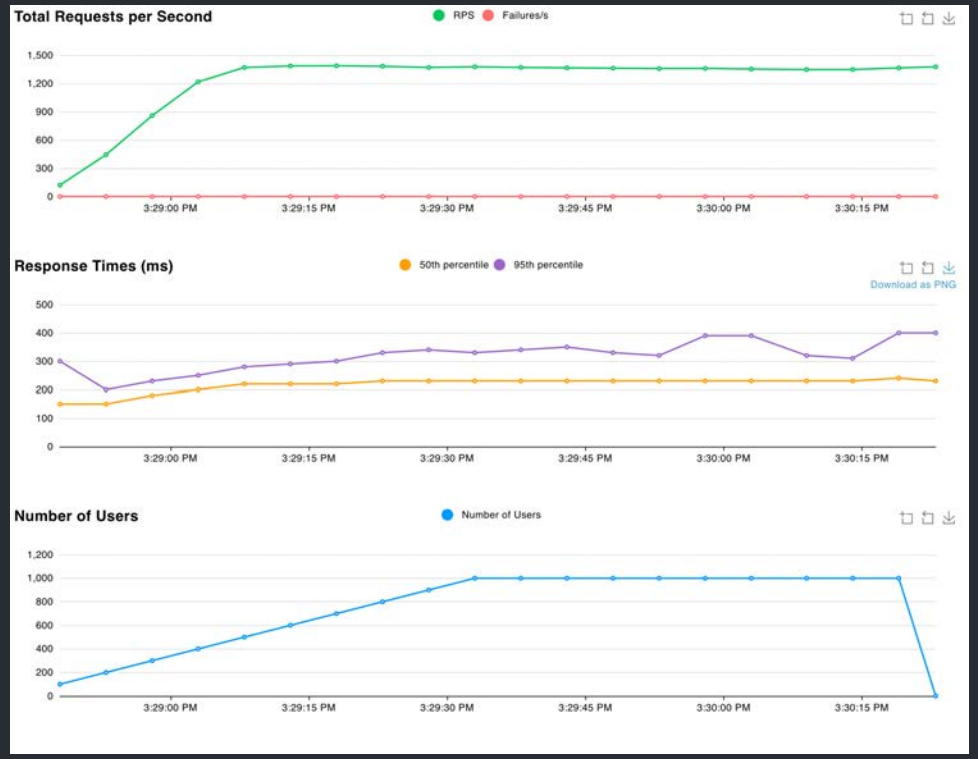
プロダクトセキュリティ

SSL通信のサポート
機密性の高いバリデーションロジックの登録や、バリデーションリクエストの盗聴を防ぐ

Open-VEへのアクセス認証
バリデーションロジックの不正な書き換えを防ぐために事前共有キーを用いたアクセス認証をサポート

パフォーマンステストの実施

実際の利用環境を想定した環境においてLOCUSTを用いた
E2Eパフォーマンステストを実施
OktaFGAと並ぶ高いパフォーマンスを発揮



検証環境
　経路　　: GitHub Actions ->AWS (ap-northeast-1)
　インスタンス: 2v CPU, 4096 MBメモリ

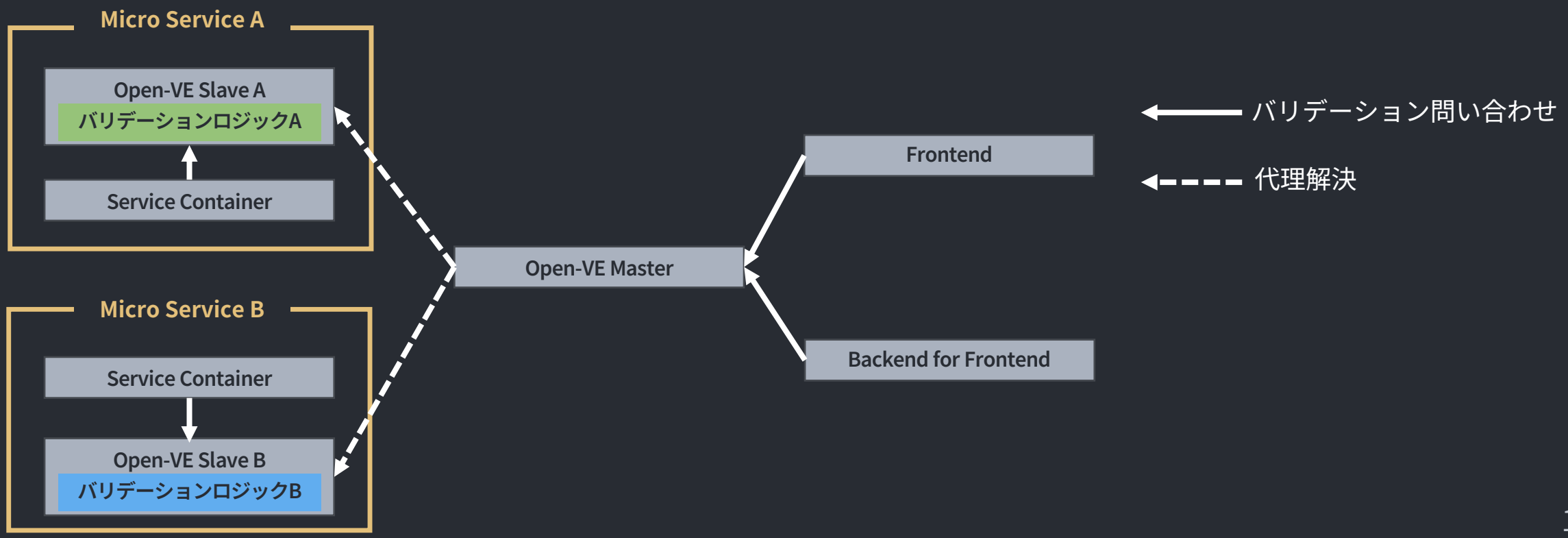
結果
　中央値　　: 200ms (1000ユーザー同時アクセス時)

Key4 : セキュリティとパフォーマンス

マスタースレーブモードのサポート

マイクロサービスにおける「責務の分離」との相性向上

・複数のOpen-VEノードにバリデーションロジックを分散して保存
・マイクロサービスごとのロジックは専用ノードに保存、問い合わせ
・フロントエンドやBFFなどサービス横断なシステムはマスターノードにバリデーション問い合わせ
・マスターノードがバリデーション問い合わせをフォワードし、代理解決



まとめ

データバリデーションの中央集権管理を実現するOpen-VEを提案

CELおよびAPI経由でのアクセスにより言語非依存のバリデーションを提供

開発者支援やパフォーマンステストの実施により開発現場での利用をサポート