Human Crest

事例から学ぶAPIテスト

~観点別で考えてみよう~

株式会社ヒューマンクレスト

Profile

- 名前：マイ　マイ　クオン

- 経歴

  2011 来日

  2012 横浜国立大学入学

  2016 大学卒業、HC入社

  現在：RakAPIt PO

ダナン

ホイアン

会社紹介

株式会社ヒューマンクレスト

設立: 2002/12/16
本社: ランドマークタワー
社員数: 168名
認証資格: ISO27001
ISTQBパートナーシップ: Platinum

株式会社ヒューマンネクスト
（関内）

JapanQuality Co.,Ltd
（ベトナム　ダナン）

ぜひダナンに来てください。

ダナン

ホイアン

Human Crest

OUR SERVICES

当社のサービス

テスト自動化ソリューション

お客様の実態に即したテスト自動化と継続運用を実現させるテスト自動化ソリューション群です。
サブスクリプションモデルから納品型モデルまでご用意しております。

Lynx

RakAPIt

TEST AUTOMATION ENGINEERING

品質改善ソリューション

- QAコンサルティングサービス・マネジメントサービス — MORE DETAILS
- 品質エンジニアリング・ソフトウェアテスト — MORE DETAILS
- HCテストセンター（横浜／ベトナム・ダナン） — MORE DETAILS
- アジャイルテスト — MORE DETAILS
- セキュリティ脆弱性診断 — MORE DETAILS
- 負荷テスト・性能テスト — MORE DETAILS
- AWSマイグレーションテスト — MORE DETAILS
- QAチーム立ち上げ支援 — MORE DETAILS

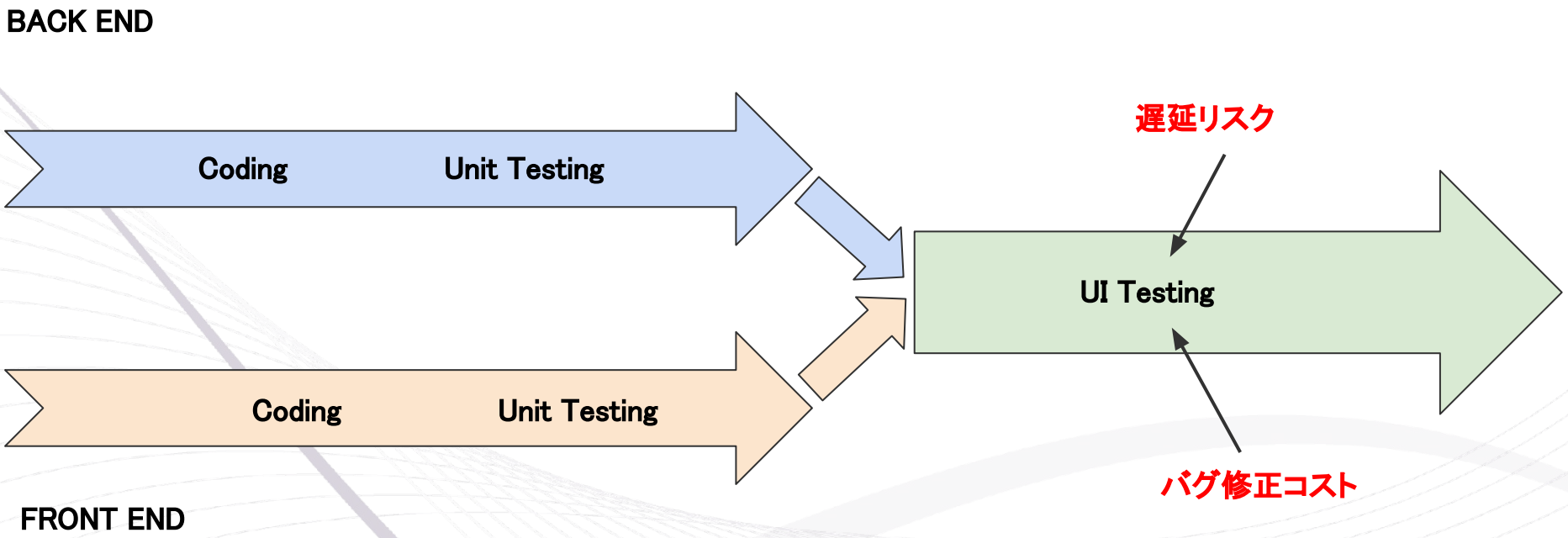
Agenda

- APIテストのメリット
- 観点別で考えるAPIテスト
  - バリデーション観点
  - 機能観点
  - セキュリティ観点
- リグレッションテスト

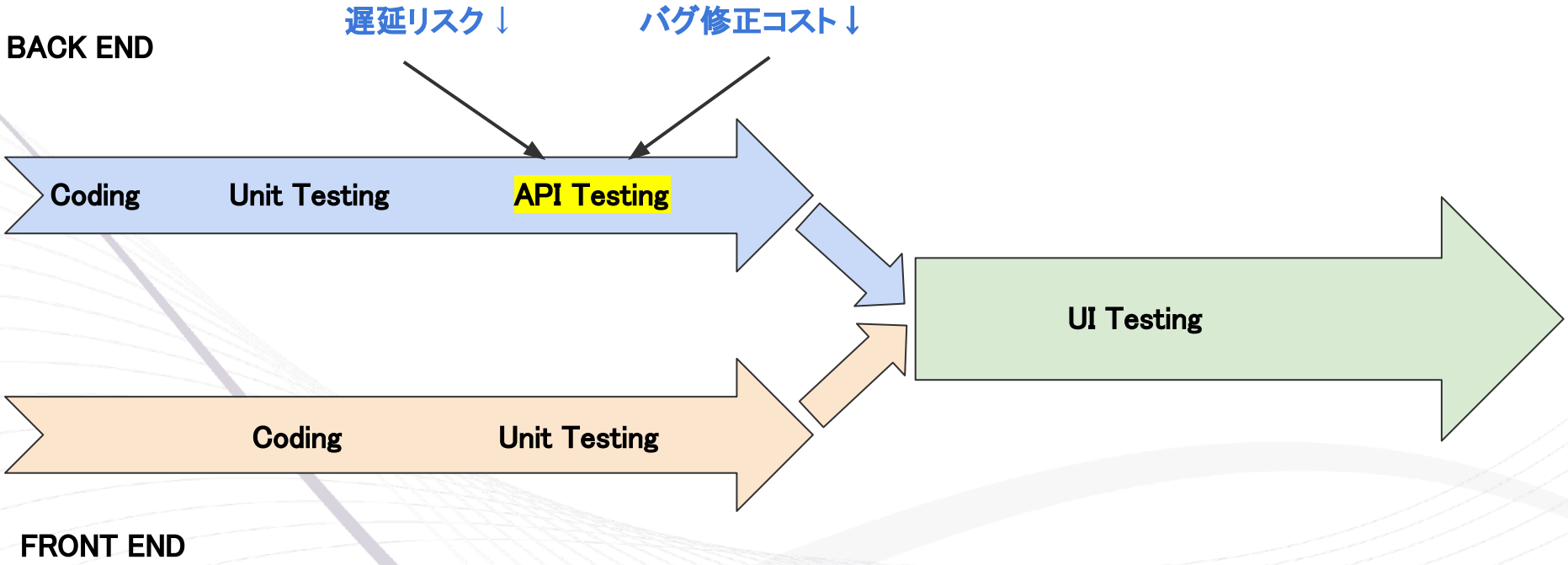
Agenda

- APIテストのメリット
- 観点別で考えるAPIテスト
  - バリデーション観点
  - 機能観点
  - セキュリティ観点
- リグレッションテスト

なぜAPIテストを行うのか？

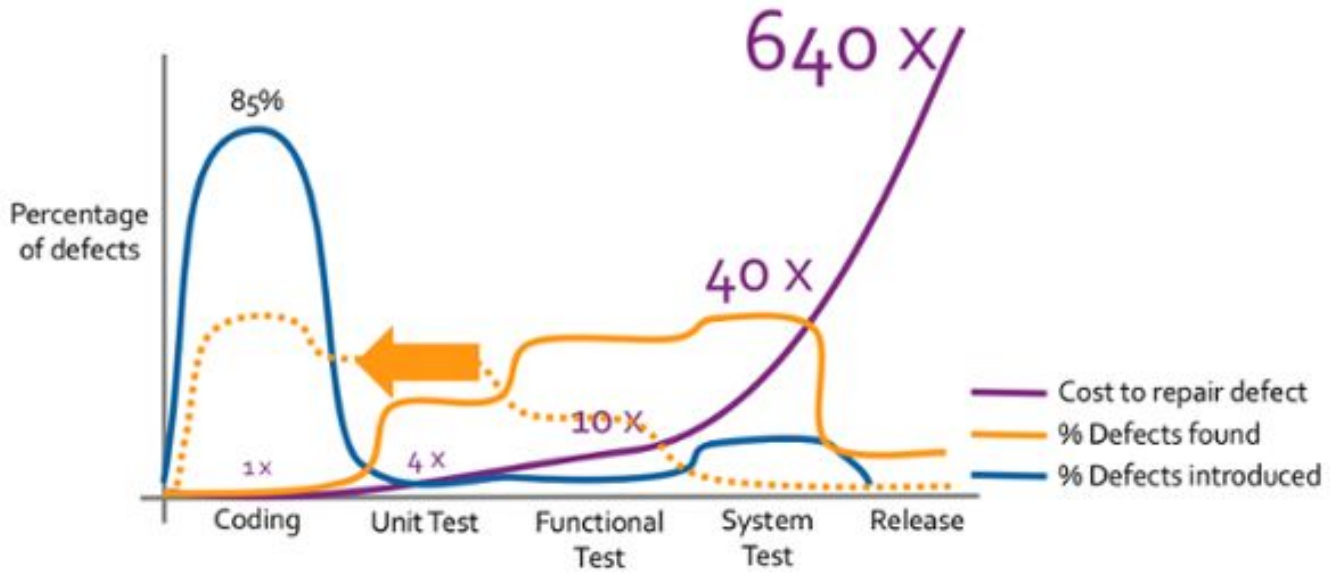


なぜAPIテストを行うのか？



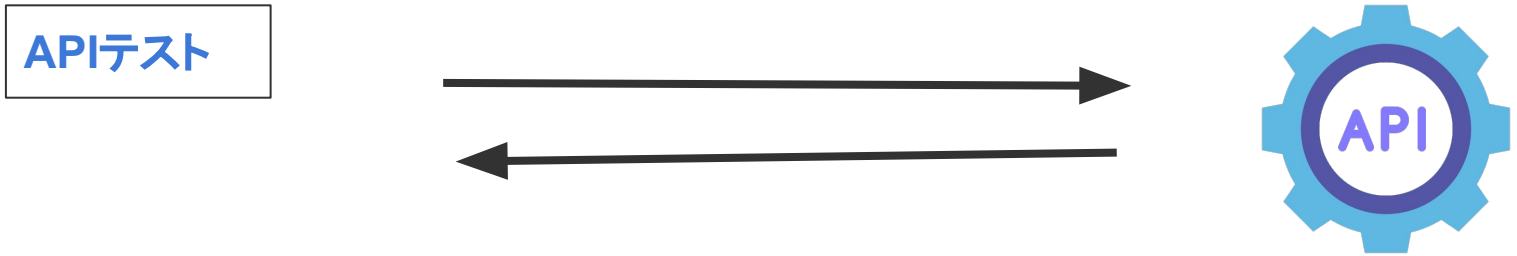
なぜAPIテストを行うのか？

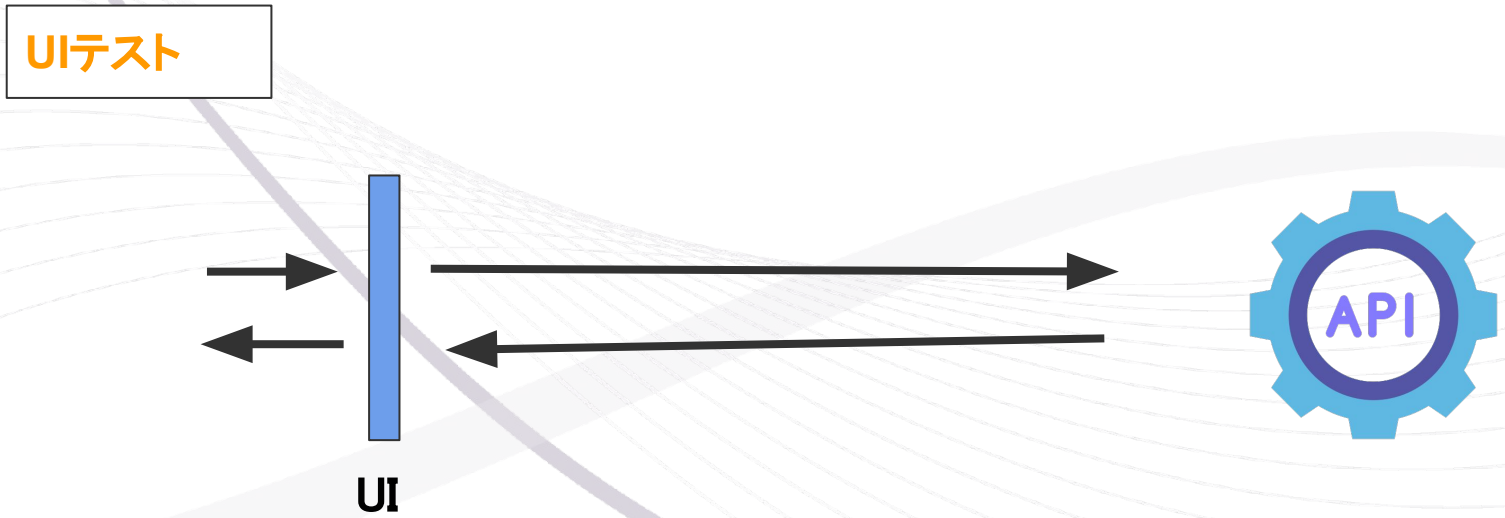
メリット①　早期にテストを開始できる



出典:https://www.stickyminds.com/article/shift-left-approach-software-testing

なぜAPIテストを行うのか？



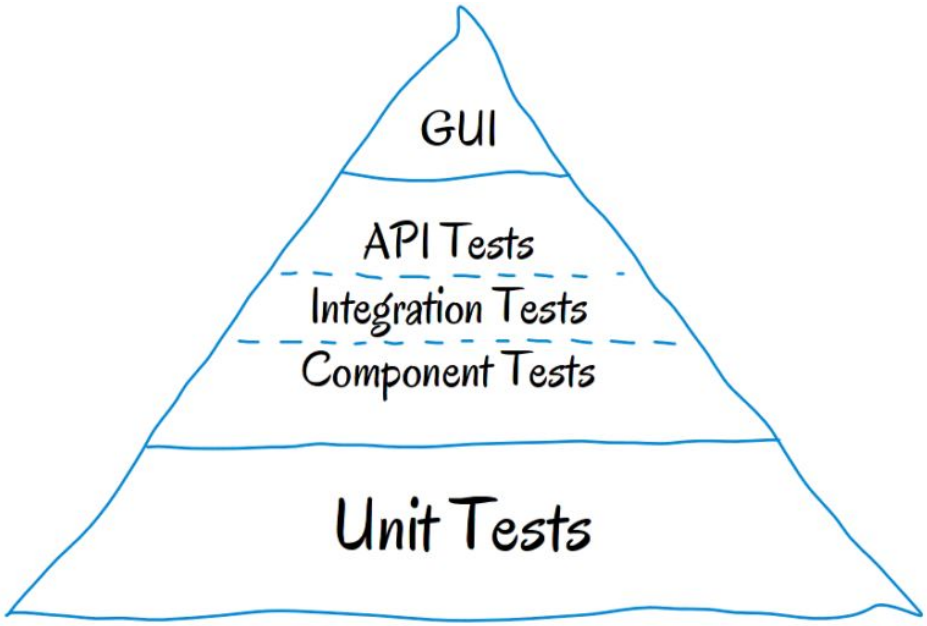


なぜAPIテストを行うのか？

メリット②

UIテストで実施できないケースでも実施できる

なぜAPIテストを行うのか？



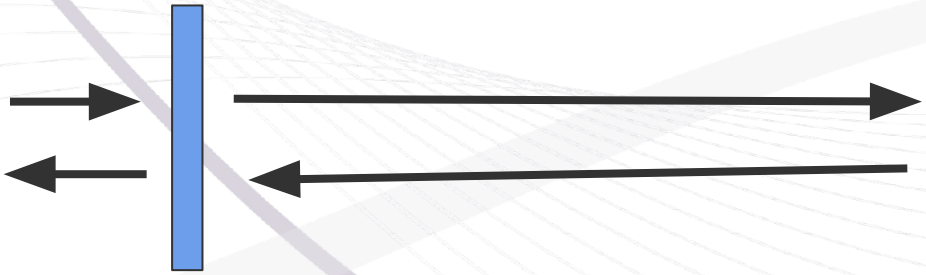
出典:https://www.mostafableu.com/blog/test-automation/

なぜAPIテストを行うのか？

APIテスト

- 実行時間が短い
- 実装がシンプル
- 安定性が高い

UIテスト



UI

なぜAPIテストを行うのか？

メリット③

自動化しやすい

APIテストの現状

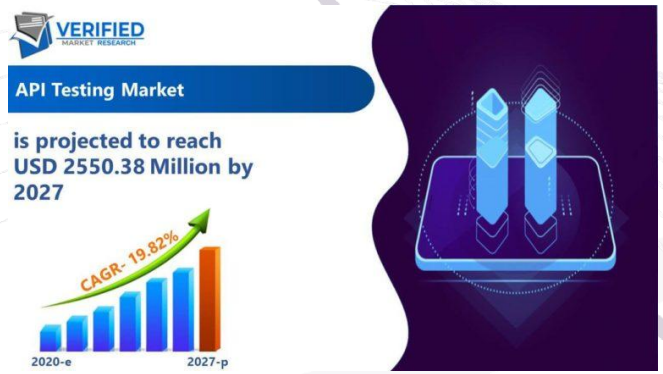
API Testing

2019

USD 661.69 million

2020~2026

伸び率19.82%



出典：https://www.verifiedmarketresearch.com

1.7%

~3.5%
(2026)

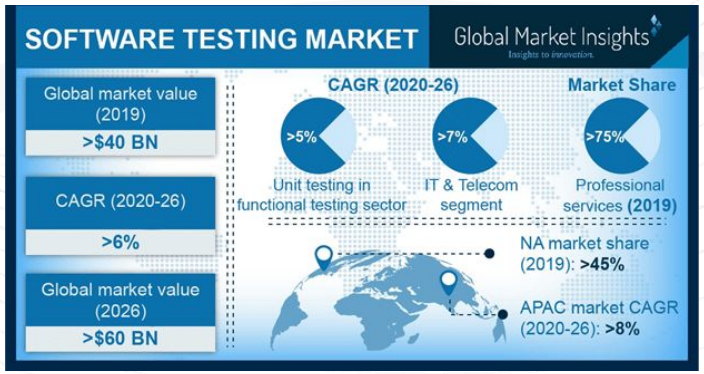
Software Testing

2019

USD 40 billion

2020~2026

伸び率6%



出典：https://www.gminsights.com/

APIテストの課題

- どんなテストをするか?
- どうテストすれば良いか?

Agenda

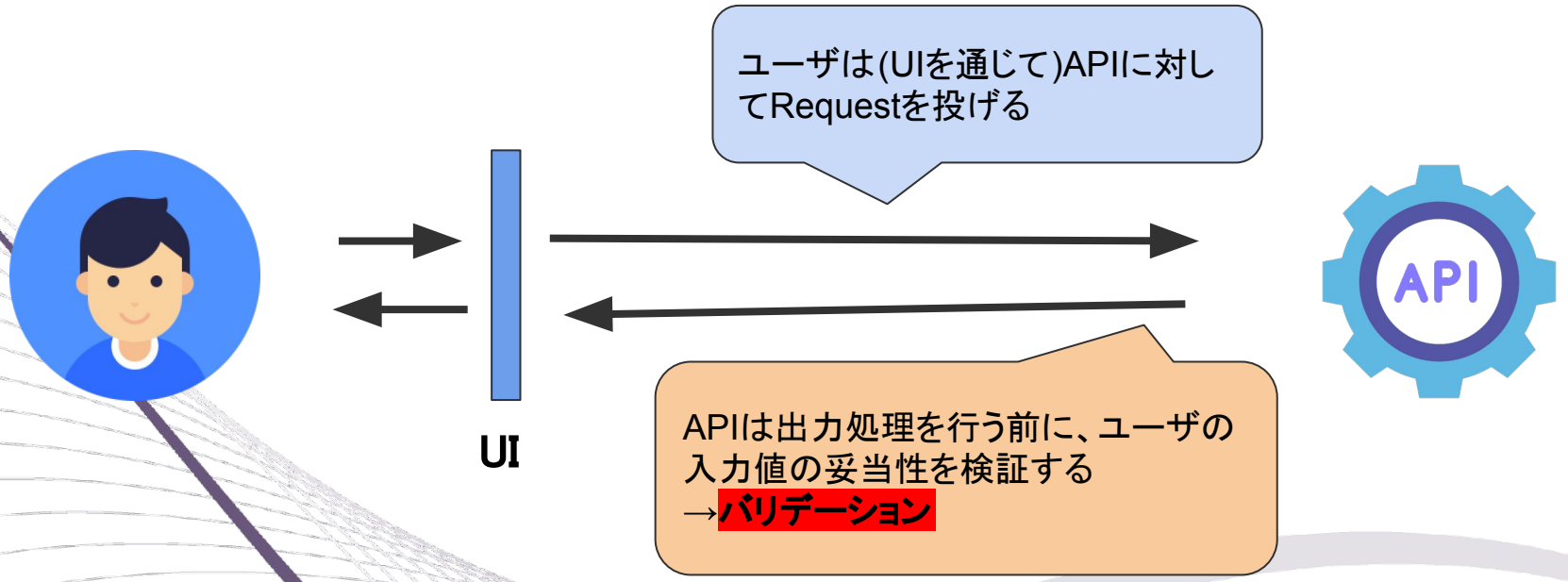
- APIテストのメリット

- 観点別で考えるAPIテスト
  - **バリデーション観点**
  - 機能観点
  - セキュリティ観点

- リグレッションテスト

バリデーション観点

~ 何をみるか ~

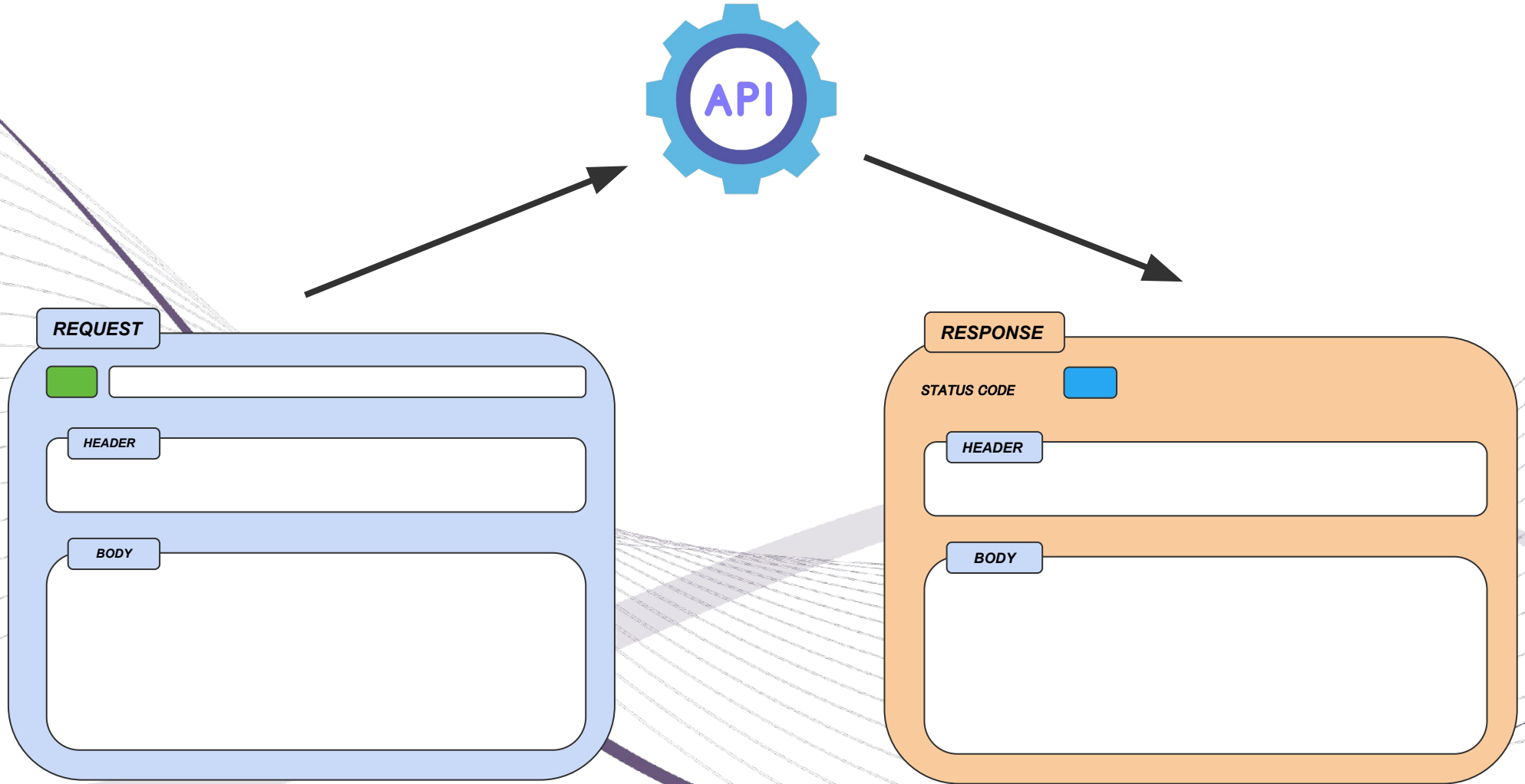


正常なデータ：受け付けて仕様通りに処理

異常なデータ：適切なエラーメッセージを返す

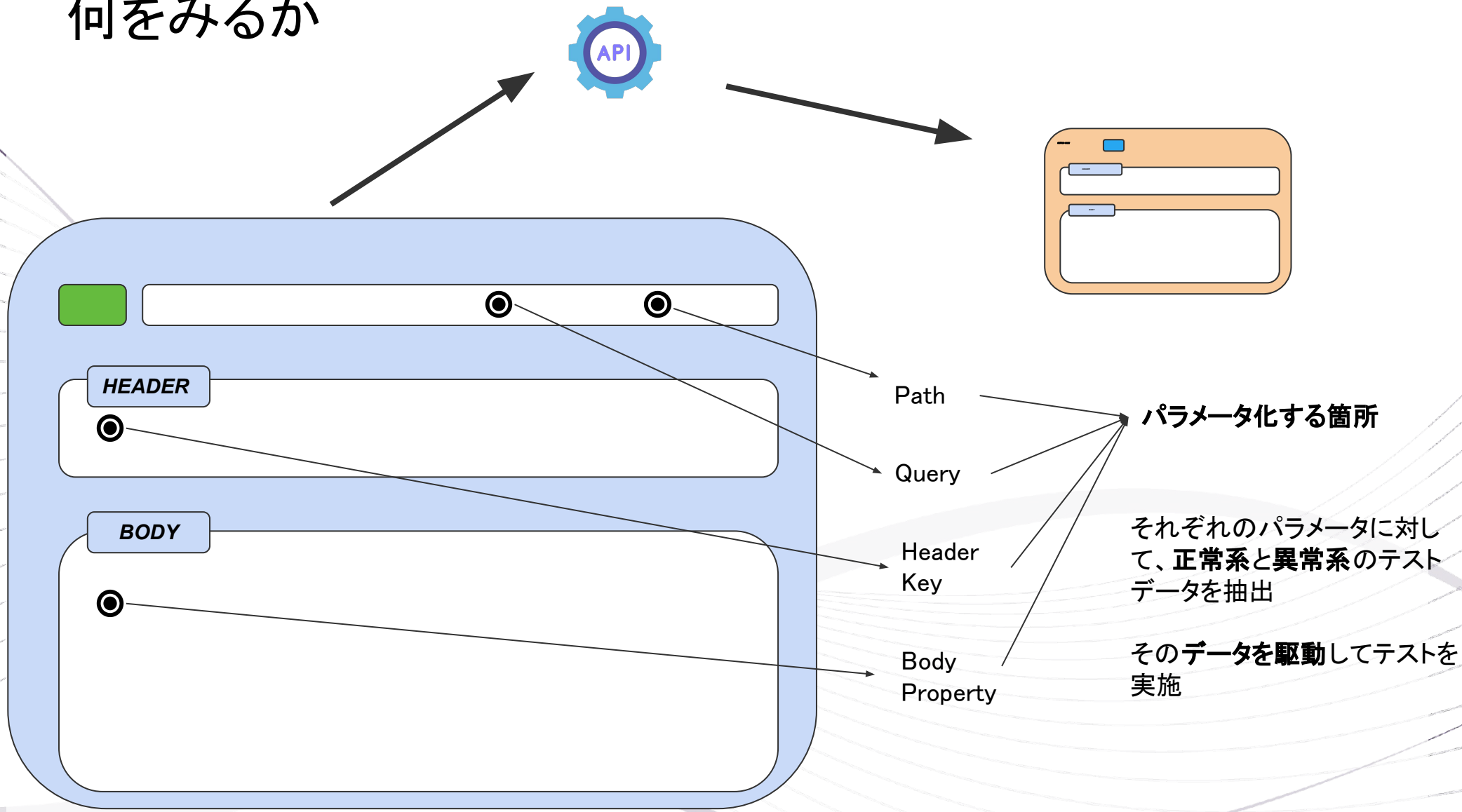
バリデーション観点

~ 何をみるか ~



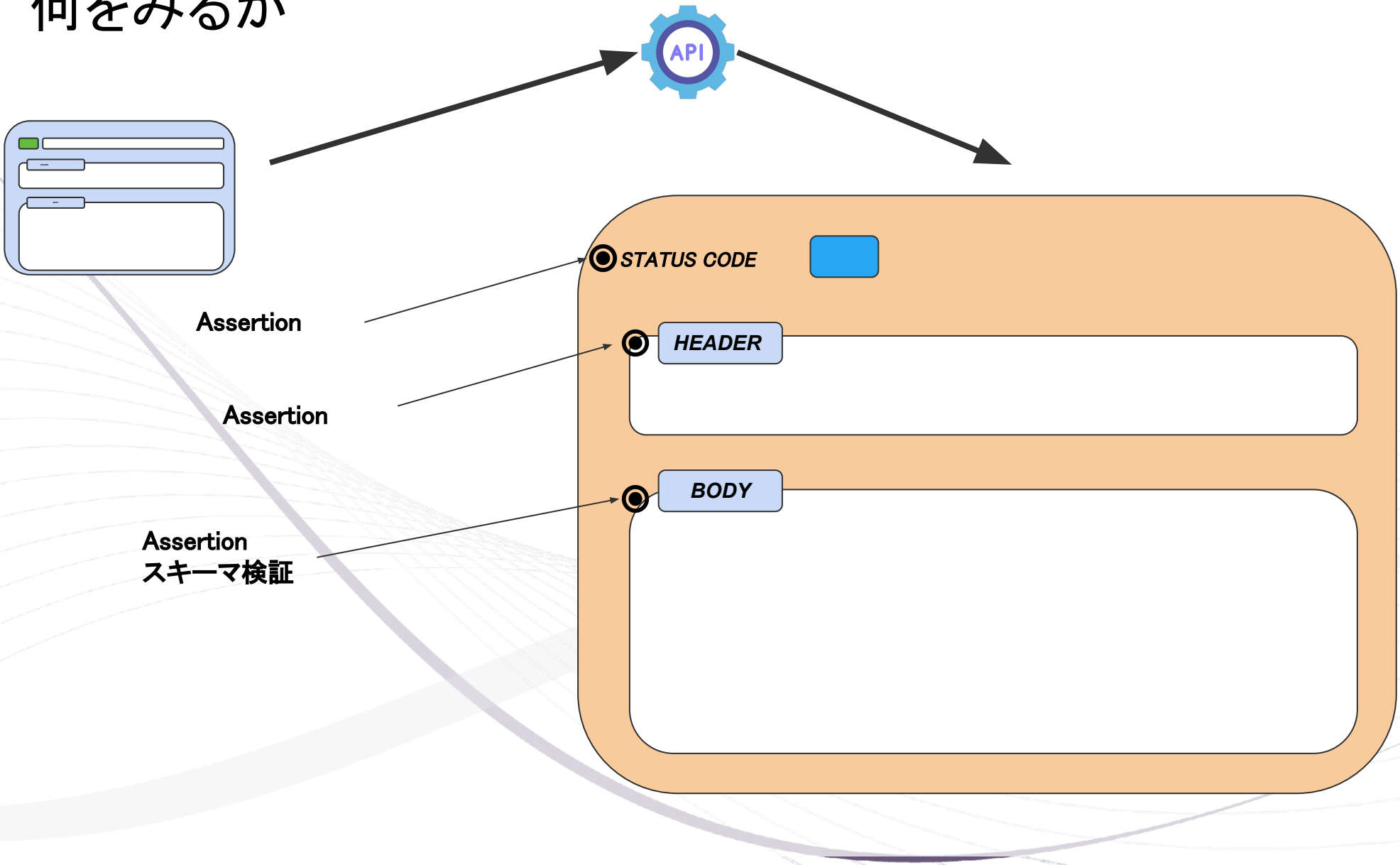
バリデーション観点

~ 何をみるか ~



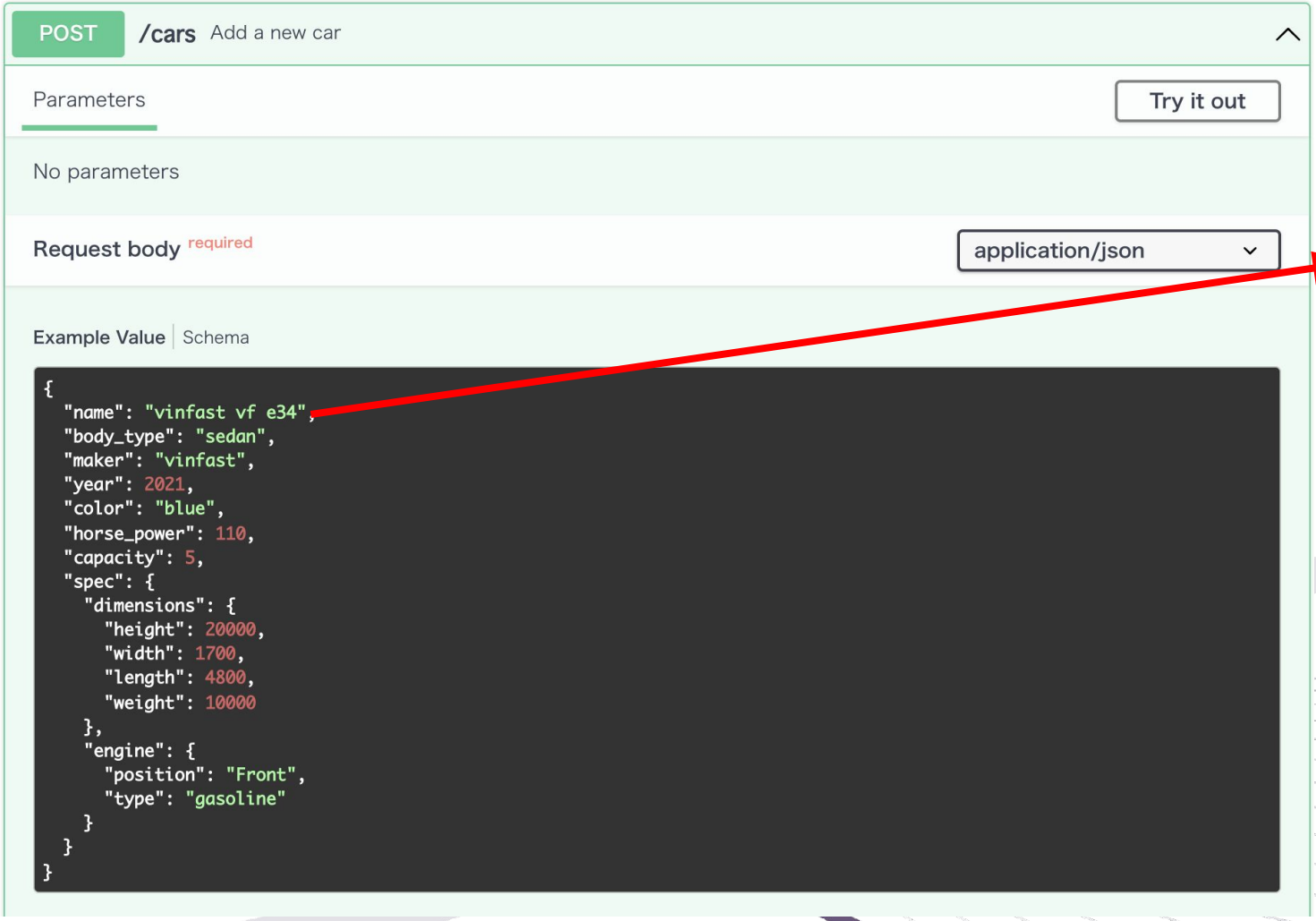
バリデーション観点

~ 何をみるか ~



バリデーション観点

~ 例 ~




```
"Name": {
  "description": "Car name",
  "type": "string",
  "minLength": 1,
  "maxLength": 100,
  "example": "vinfast vf e34"
},
```

テスト対象として考える

Human Crest

RAKU API TESTING

RakAPIt

バリデーション観点

~ どう実施しているか ~

テストデータ → テストケース → アサーション設定 → テスト実施

```
"Name": {
  "description": "Car name",
  "type": "string",
  "minLength": 1,
  "maxLength": 100,
  "example": "vinfast vf e34"
},
```

正常データ

- **サンプルデータ** *vinfast vf e34*
- **文字列長最小** *A*
- **文字列長最長** *vinf … e34* (100文字)
- **2バイト文字** *トヨタ*

異常データ

- **キーが存在しない**
- **空、null**
- **文字列長 最長の境界値** *vinf … e344* (101文字)
- **typeチェック** *(int)12, (boolean)true*

バリデーションチェック観点

~ どう実施しているか ~

テストデータ → テストケース → アサーション設定 → テスト実施

RakAPIt

Title	Headers	
	Content-Type	name
【name】【正常系】サンプルデータ	application/json	vinfast vf e34
【name】【正常系】文字列長最小	application/json	A
【name】【正常系】文字列長最長	application/json	vinfast vf e34 vinfast v
【name】【正常系】２バイト文字	application/json	トヨタ

バリデーションチェック観点

~ どう実施しているか ~

テストデータ → テストケース → アサーション設定 → テスト実施

Status Code アサーション

Header アサーション

Bodyアサーション スキーマ検証

RakAPIt

Title	Response					
	Status Code		Headers		Body	
	Expected	Result	content-type		Json / Schema	
			Expected	Result	Expected	Result
【name】【正常系】サンプルデータ	200		Exist		Match	
【name】【正常系】文字列長最小	200		Exist		Match	
【name】【正常系】文字列長最長	200		Exist		Match	
【name】【正常系】２バイト文字	200		Exist		Match	

JSON Schema

JSON Schema is a vocabulary that allows you to **annotate** and **validate** JSON documents.

出典：https://json-schema.org/

JSON Schema

JSON Schema

```
{
    "type": "object",
    "properties": {
      "email": {
        "type": "string"
      },
      "password": {
        "type": "string"
      }
    },
    "required": [
      "email",
      "password"
    ]
  }
```

JSON Schema

JSON

```
{
    "email": "abc@mail.com",
    "password": "12345678"
}
```

Pass

JSON Schema

```
{
    "type": "object",
    "properties": {
      "email": {
        "type": "string"
      },
      "password": {
        "type": "string"
      }
    },
    "required": [
      "email",
      "password"
    ]
  }
```

JSON Schema

JSON

```
{
    "email": "abc@mail.com",
    "password": 12345678
}
```

Fail

JSON Schema

```
{
    "type": "object",
    "properties": {
      "email": {
        "type": "string"
      },
      "password": {
        "type": "string"
      }
    },
    "required": [
      "email",
      "password"
    ]
  }
```

JSON Schema

JSON

```
{
    "email": "abc@mail.com"
}
```

Fail

JSON Schema

```
{
    "type": "object",
    "properties": {
      "email": {
        "type": "string"
      },
      "password": {
        "type": "string"
      }
    },
    "required": [
      "email",
      "password"
    ]
  }
```

バリデーションチェック観点

~ どう実施しているか ~

テストデータ → テストケース → アサーション設定 → テスト実施

Status Code アサーション

Header アサーション

Bodyアサーション スキーマ検証

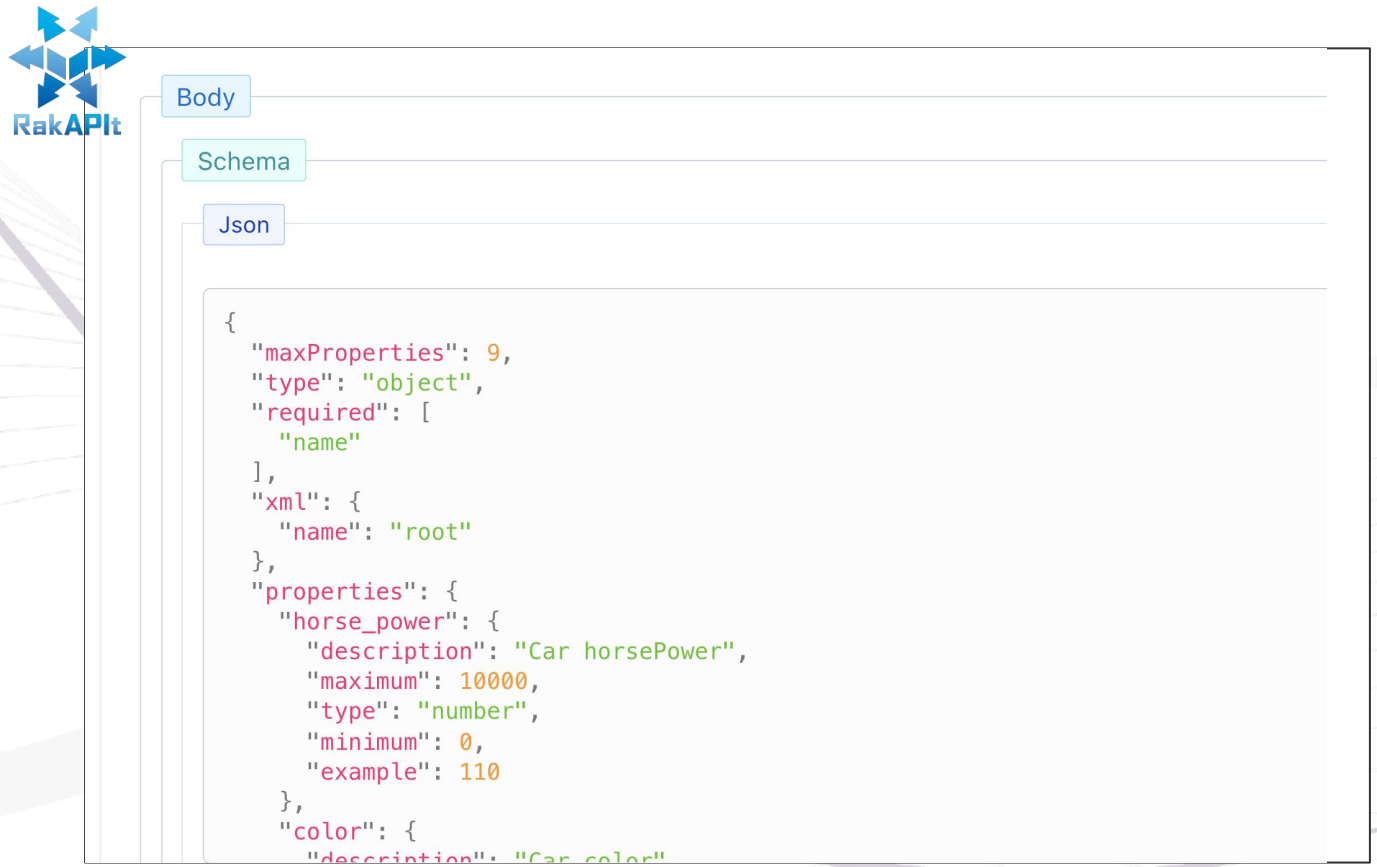
RakAPIt

Title	Response					
	Status Code		Headers		Body	
	Expected	Result	content-type		Json / Schema	
			Expected	Result	Expected	Result
【name】【正常系】サンプルデータ	200		Exist		Match	
【name】【正常系】文字列長最小	200		Exist		Match	
【name】【正常系】文字列長最長	200		Exist		Match	
【name】【正常系】２バイト文字	200		Exist		Match	

バリデーションチェック観点

~ どう実施しているか ~

テストデータ → テストケース → アサーション設定 → テスト実施



バリデーション観点

~ どう実施しているか ~

テストデータ → テストケース → アサーション設定 → テスト実施

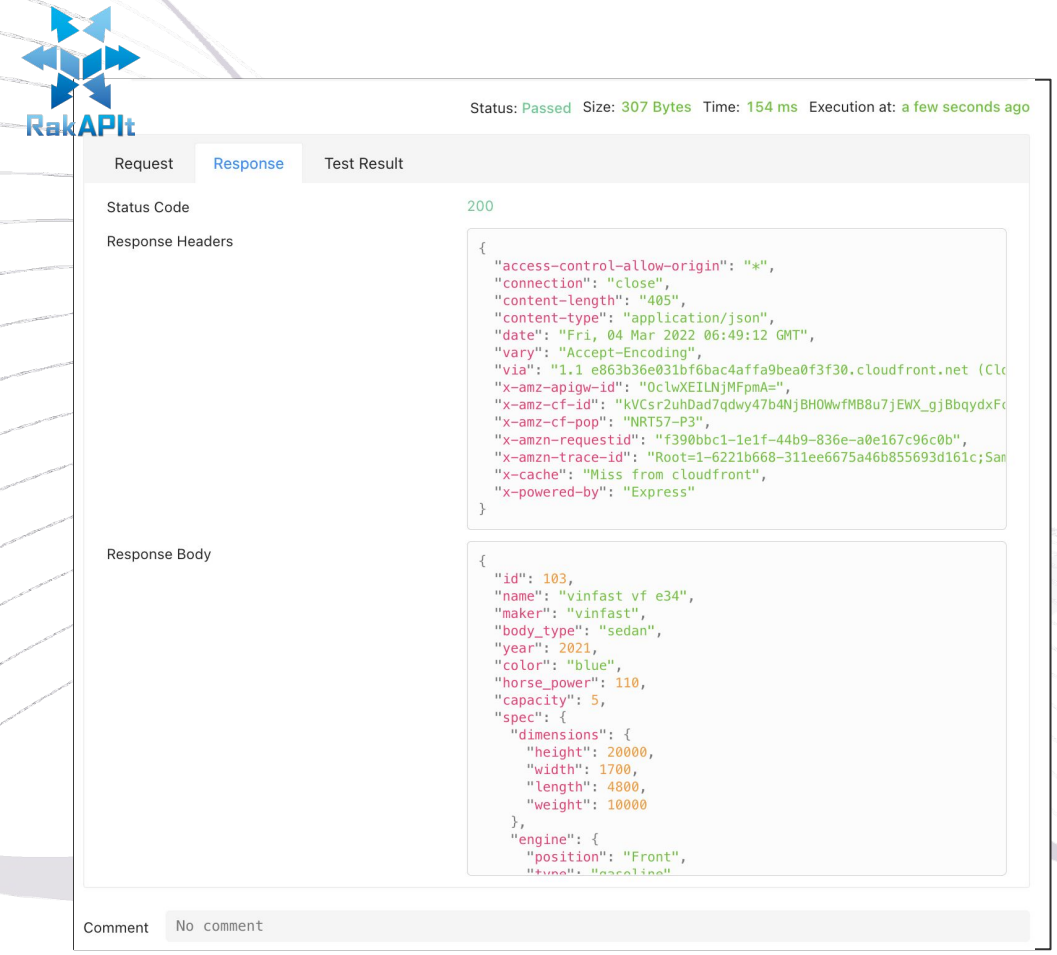
RakAPIt

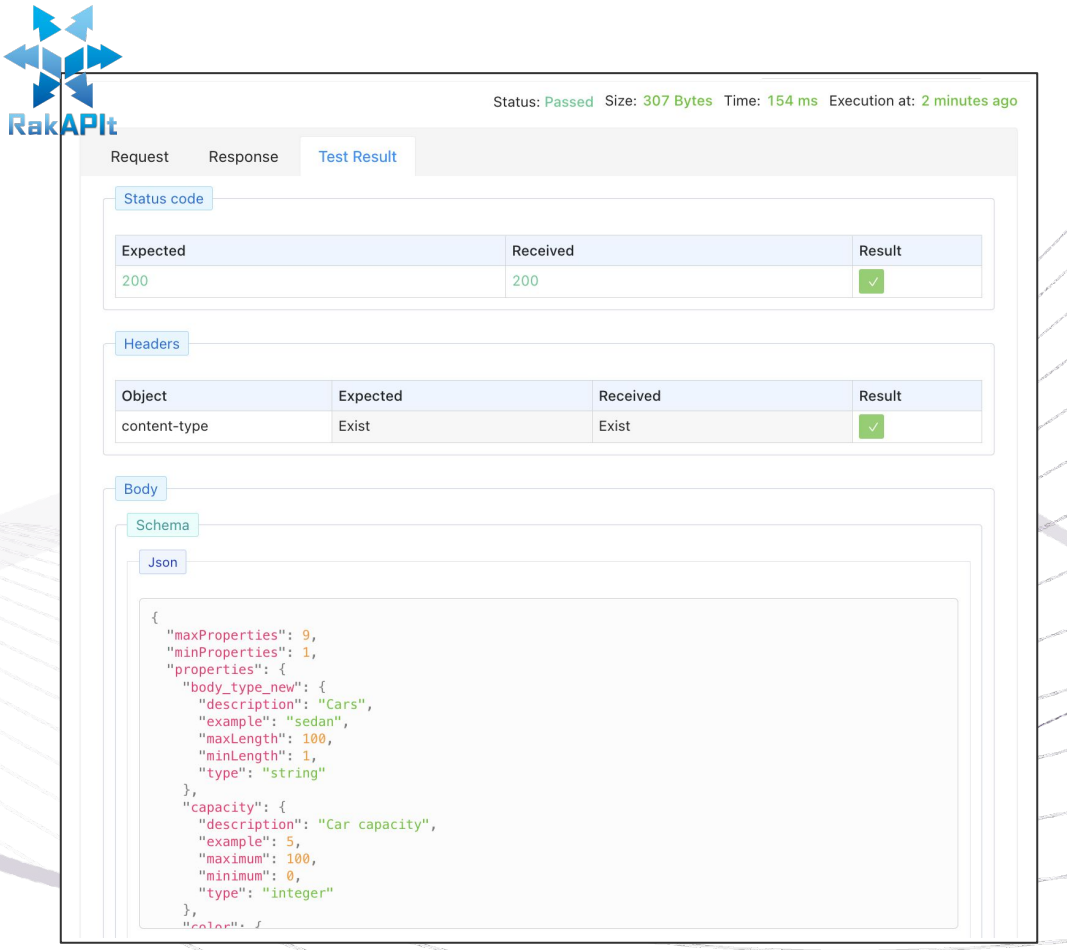
Title	Response					
	Status Code		Headers		Body	
	Expected	Result	content-type		Json / Schema	
			Expected	Result	Expected	Result
【name】【正常系】サンプルデータ	200	✓	Exist	✓	Match	✓
【name】【正常系】文字列長最小	200	✓	Exist	✓	Match	✓
【name】【正常系】文字列長最長	200	✓	Exist	✓	Match	✓
【name】【正常系】２バイト文字	200	✓	Exist	✓	Match	✓

バリデーション観点

~ どう実施しているか ~

テストデータ → テストケース → アサーション設定 → テスト実施



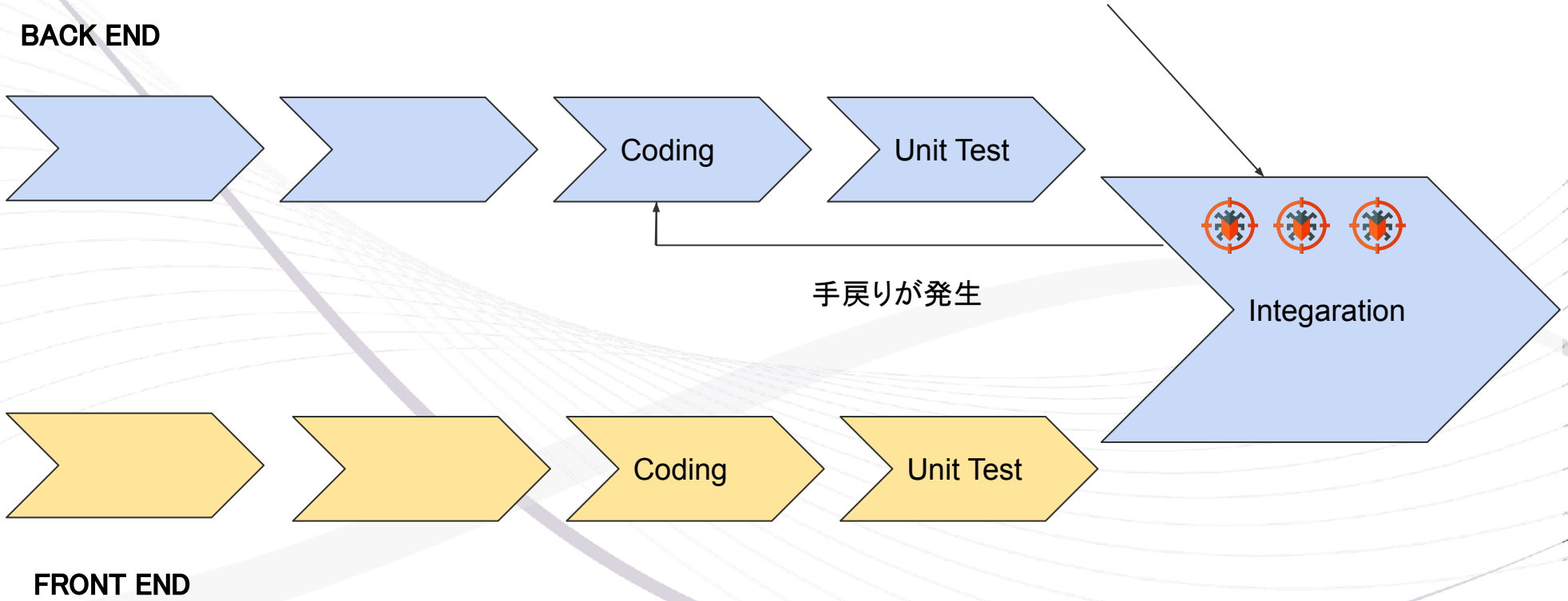


バリデーション観点

事例

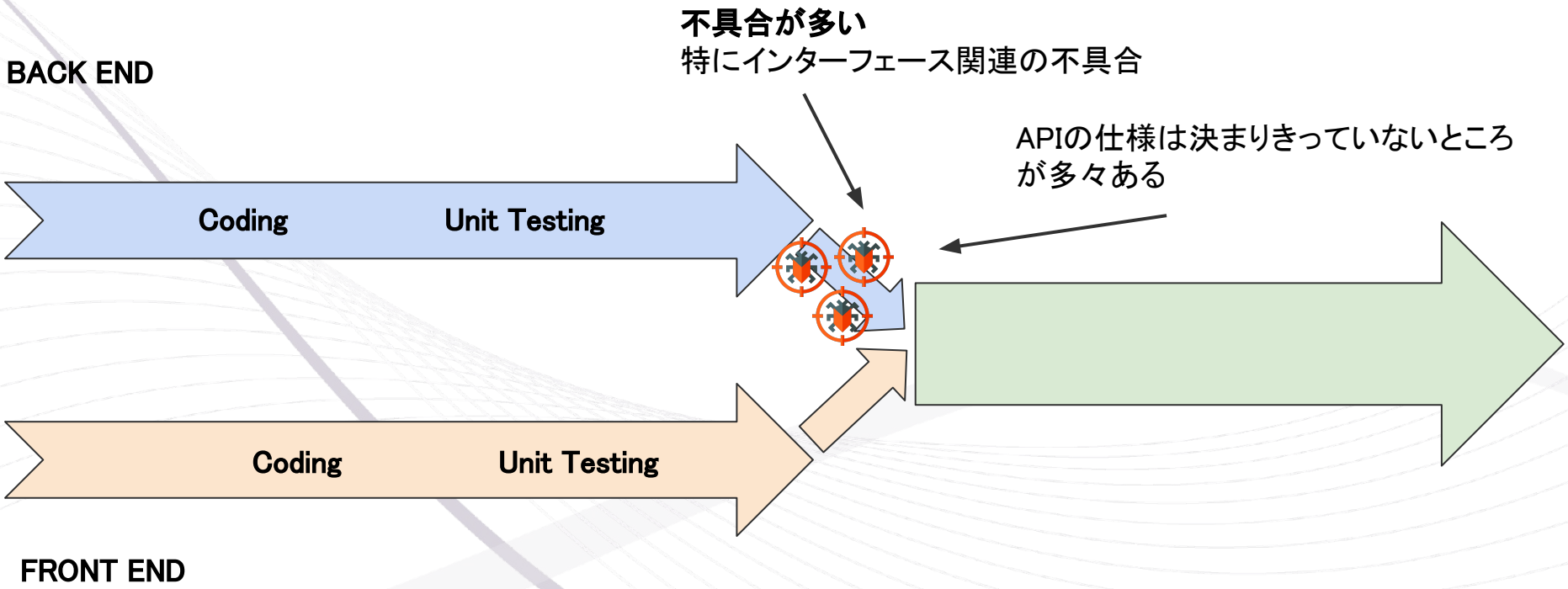
問題

結合する際のAPI側の問題:
・不具合が多い。特にインターフェース関連の不具合。
・ APIの仕様は決まりきっていないところが多々ある



事例

問題



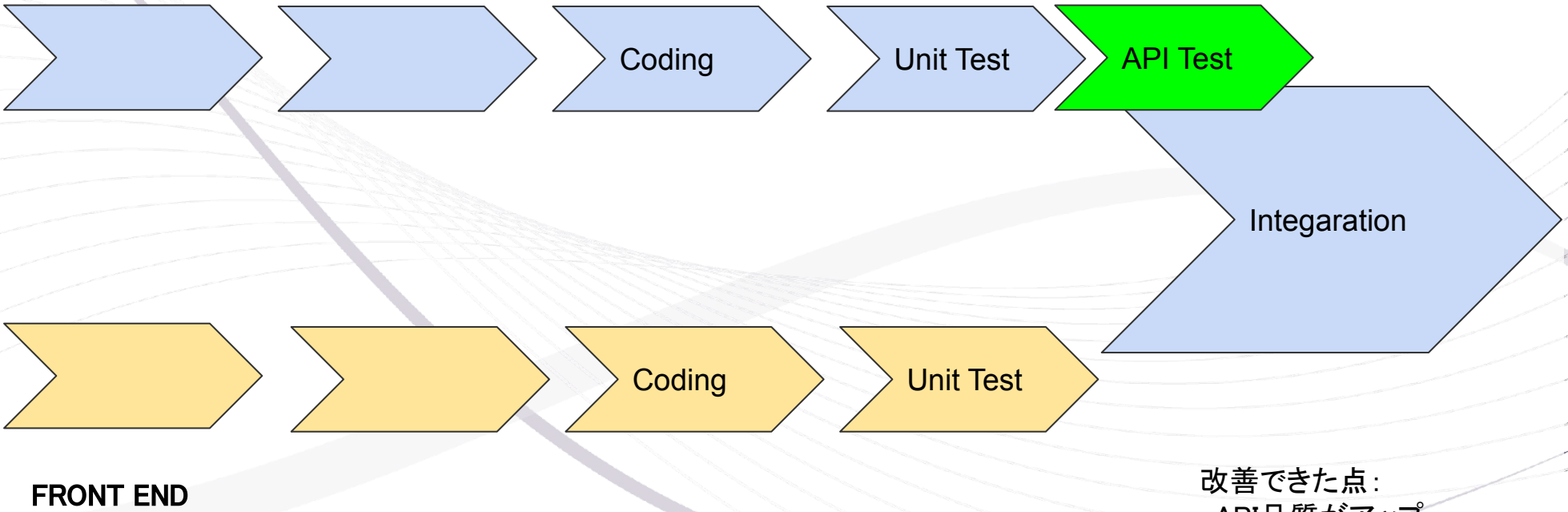
バリデーション観点

事例

対策

Swagger作成して仕様を整理
・**仕様の不足分**を洗い出す
・**API間でのパラメータの統一性**を調査
→仕様を修正・追加

Swaggerに基づいて
追加APIテストを行う



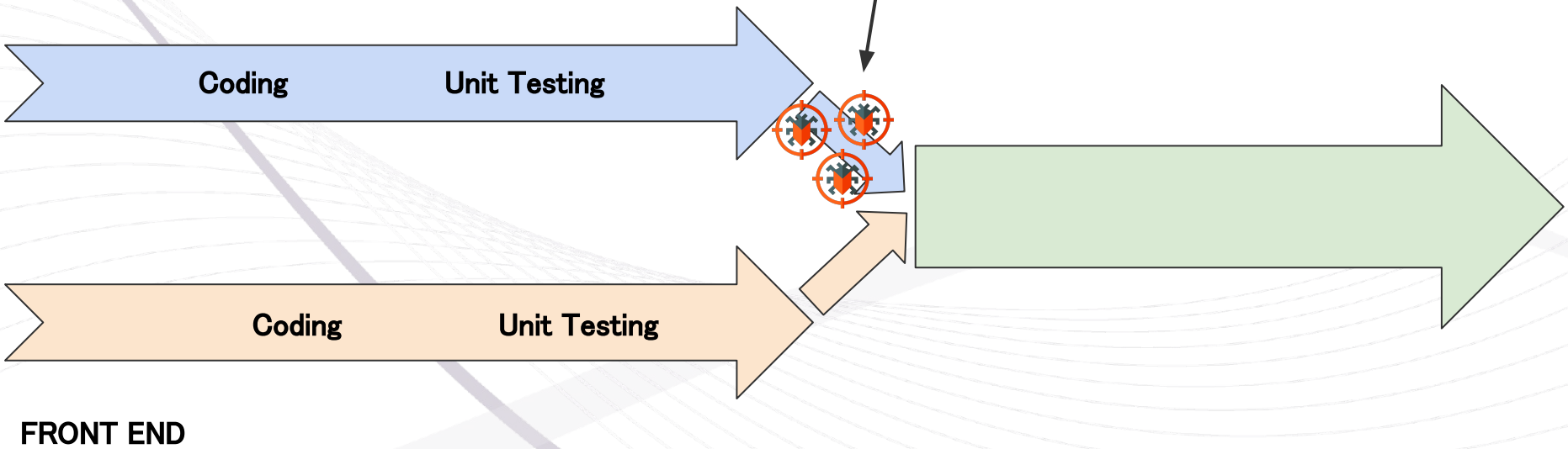
改善できた点：
・API品質がアップ
・スムーズにFEと結合できた

事例

対策

仕様を整理(OpenAPI作成)
・**仕様の不足分**を洗い出す
・**API間でのパラメータの統一性** を調査・指摘

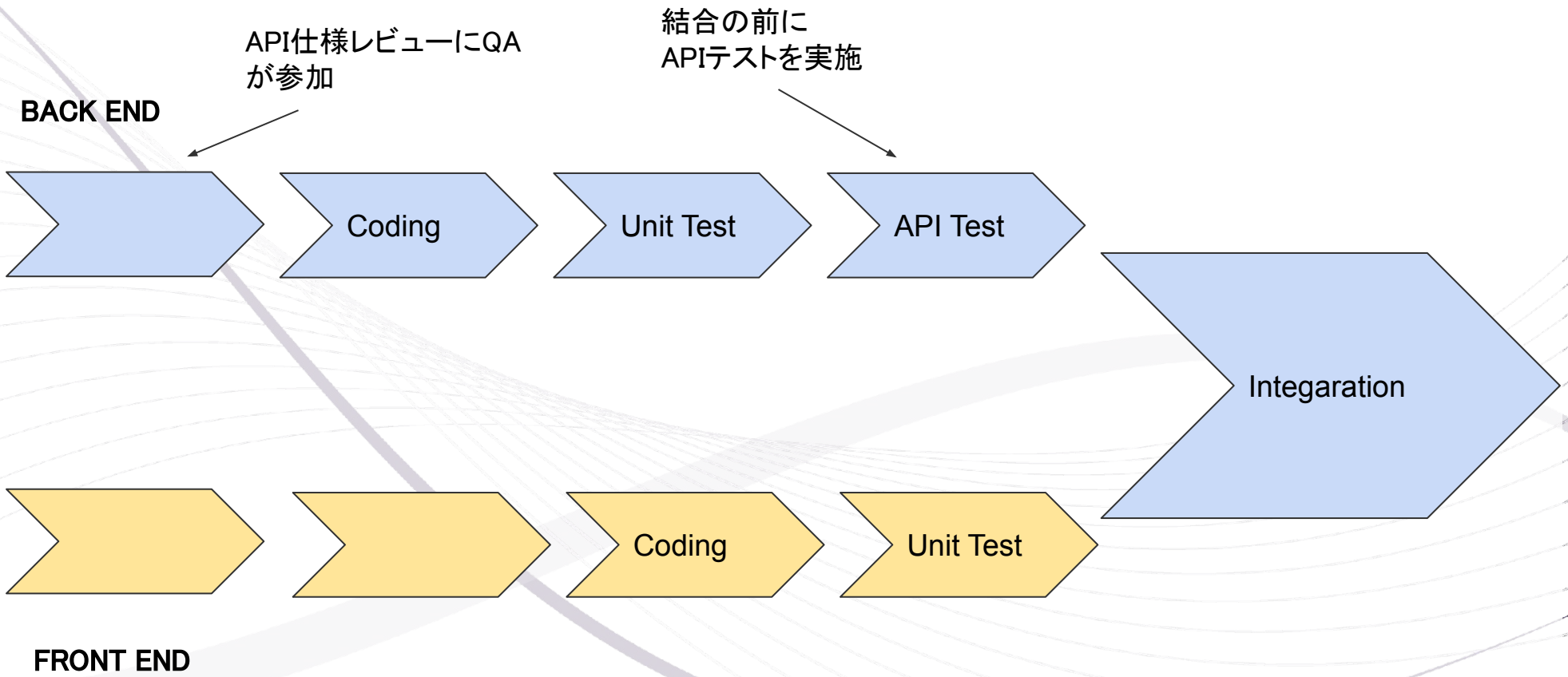
OpenAPIに基づいて
追加APIテスト を行う



バリデーション観点

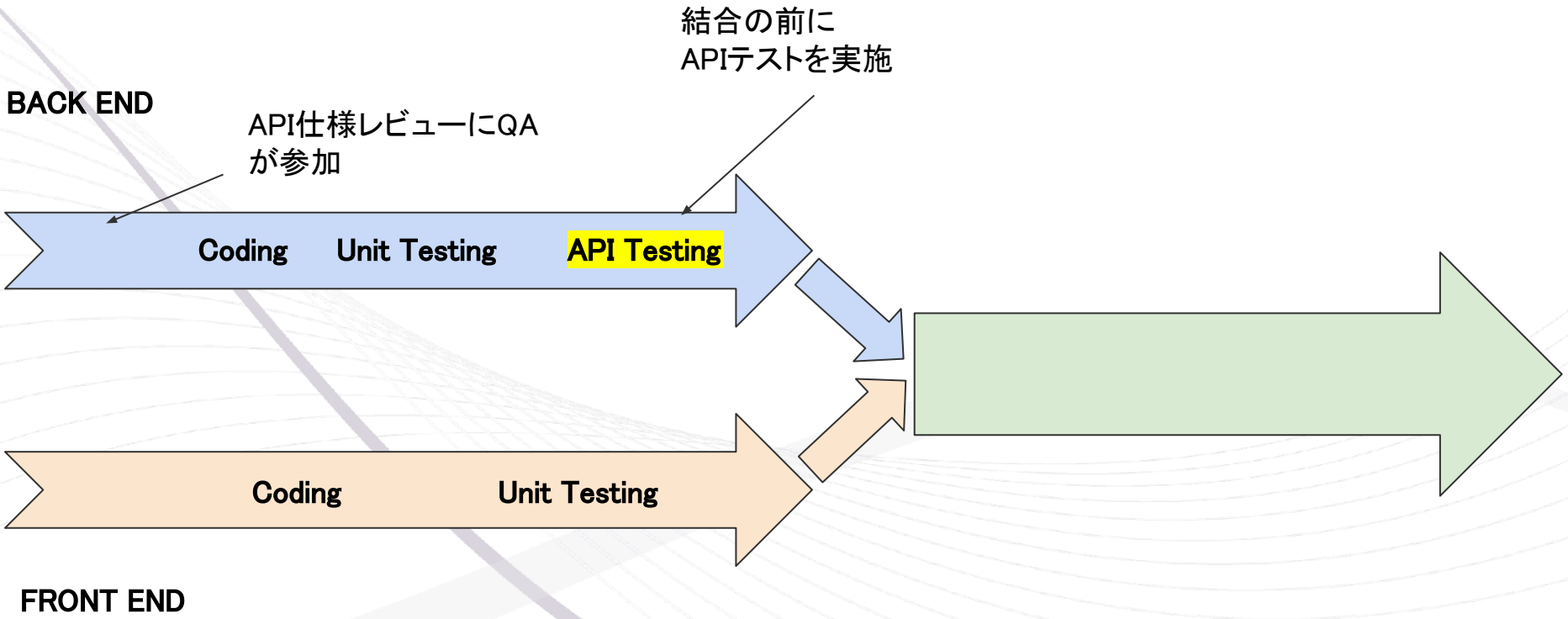
事例

次フェーズの取り組み



事例

次フェーズの取り組み



観点別で考えてみよう

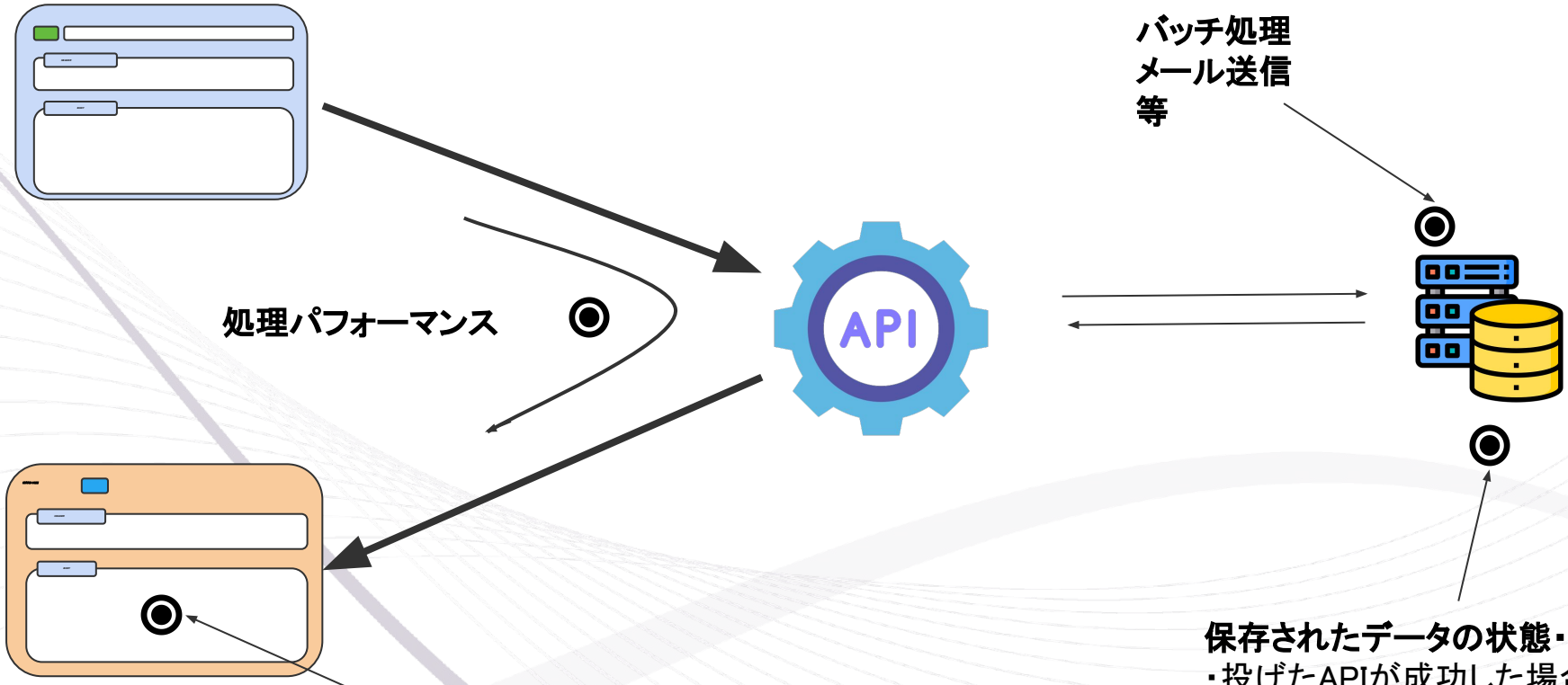
- バリデーション観点
- 機能観点 （★）
- セキュリティ観点

Agenda

- APIテストのメリット
- 観点別で考えるAPIテスト
  - バリデーション観点
  - **機能観点**
  - セキュリティ観点
- リグレッションテスト

機能観点

~ 何をみるか ~



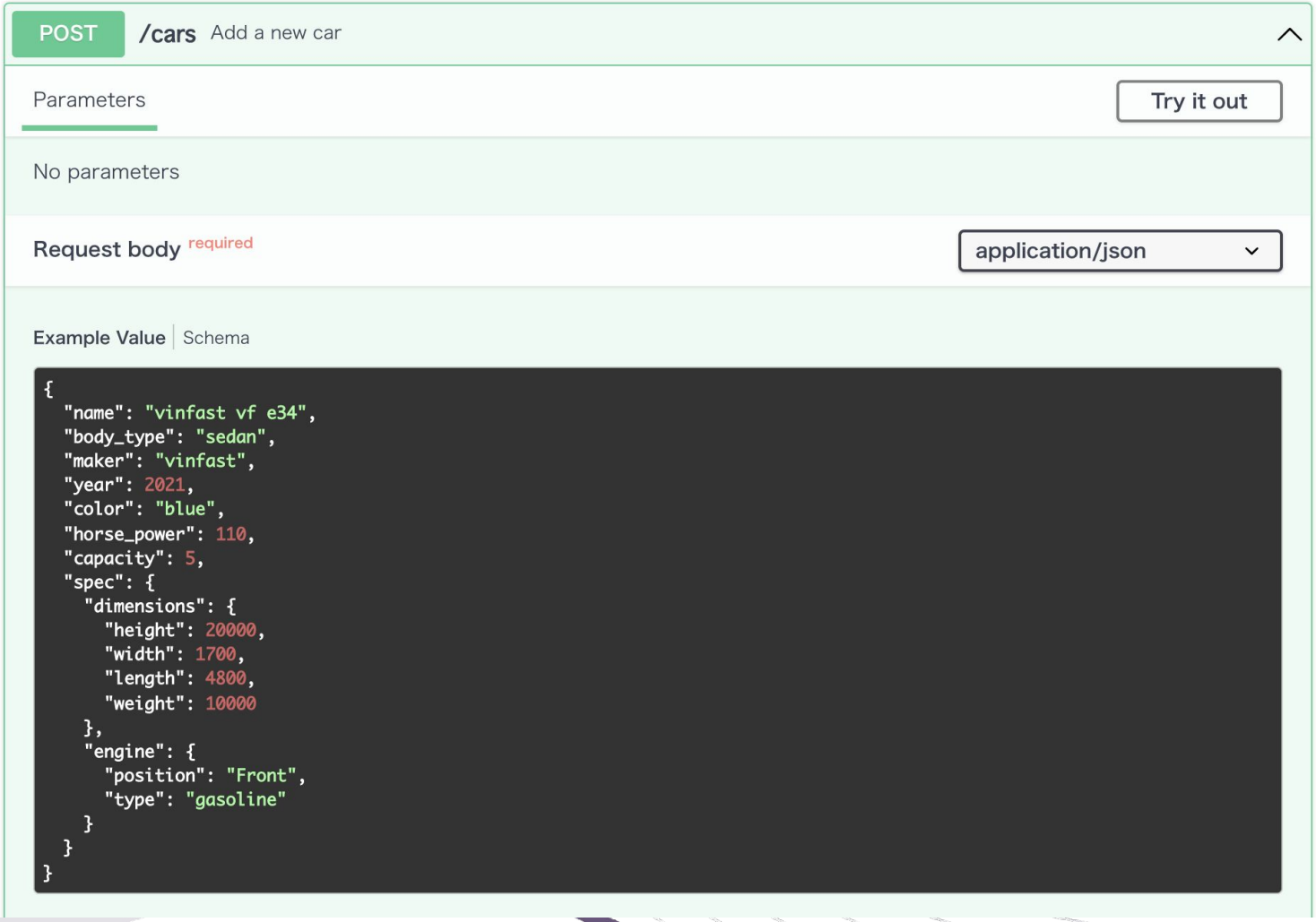
機能観点

UIテストで実施した場合との比較

	UIテスト	APIテスト
難易度	直観的でやりやすい	学習コストがかかる
実施できるタイミング	UIが完成しないと実施できない	APIの実装が完了したらすぐできる
テスタビリティ	UIによる制限がある	UIテストで実施できないケースも実施できる
実施コスト	実行時間が長い	実行時間が短い
安定性	不安定	安定
自動化	自動化しにくいところがある	自動化しやすい

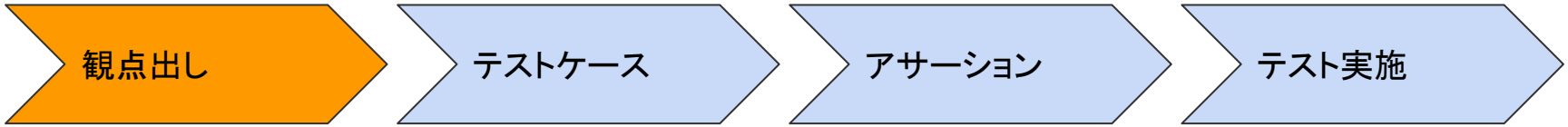
機能観点

~ 例 ~



機能観点

~ どう実施しているか ~



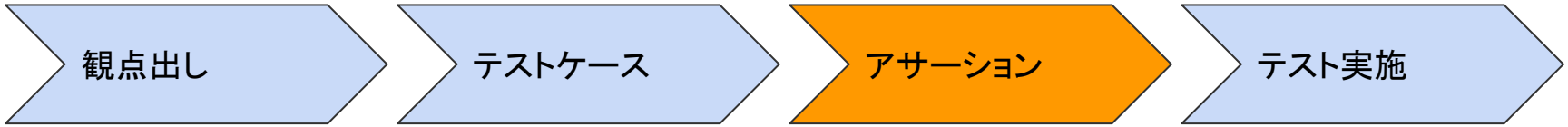
- Response確認
  - 新規idが付与されるか。idはユニークか。
  - 返却されたデータは送ったデータと一致するか

- データ確認
  - 成功した場合、データが反映されているか
  - 失敗した場合、データが作成されていないか

- 処理パフォーマンス
  - レスポンスタイム
  - 容量が大きいデータを送った場合のパフォーマンス

機能観点

~ どう実施しているか ~



- Response Body
- レスポンスタイム
- データ確認

  POST実施した後、GETメソッドでデータ取得・確認

機能観点

~ バグ例 ~

削除済みのデータを変更できてしまう

機能観点

~ バグ例 ~

削除APIが成功したが、データは削除されない

Agenda

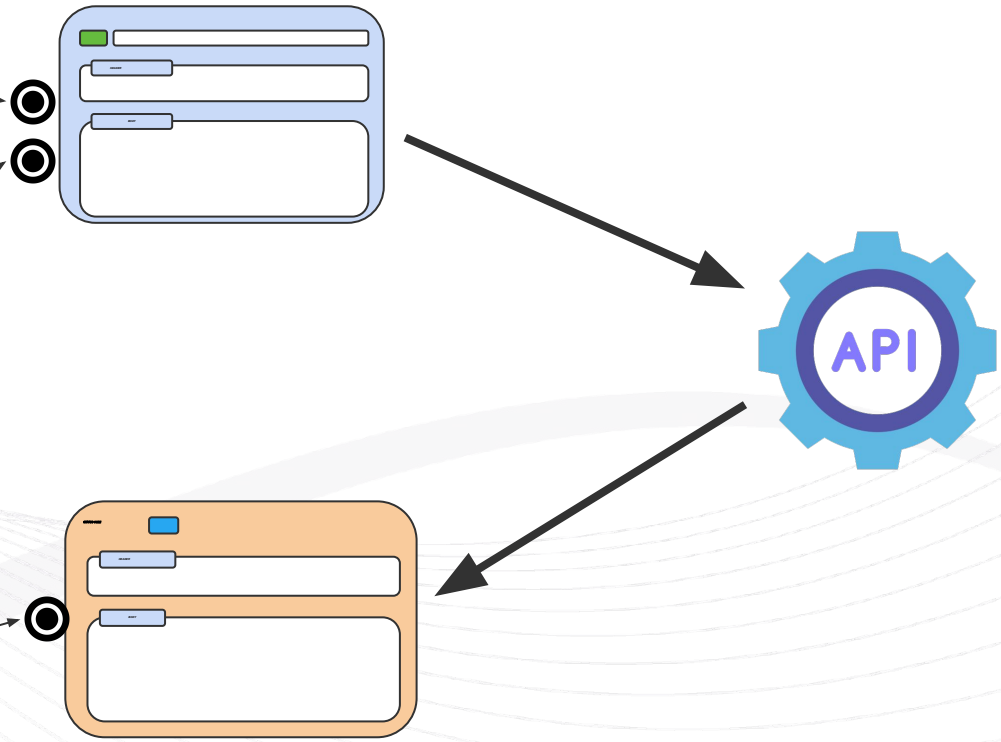
- APIテストのメリット
- 観点別で考えるAPIテスト
  - バリデーション観点
  - 機能観点
  - **セキュリティ観点**
- リグレッションテスト

セキュリティ観点

~ 何をみるか ~

認証・認可
・トークン切れ
・他ユーザの情報へのアクセス

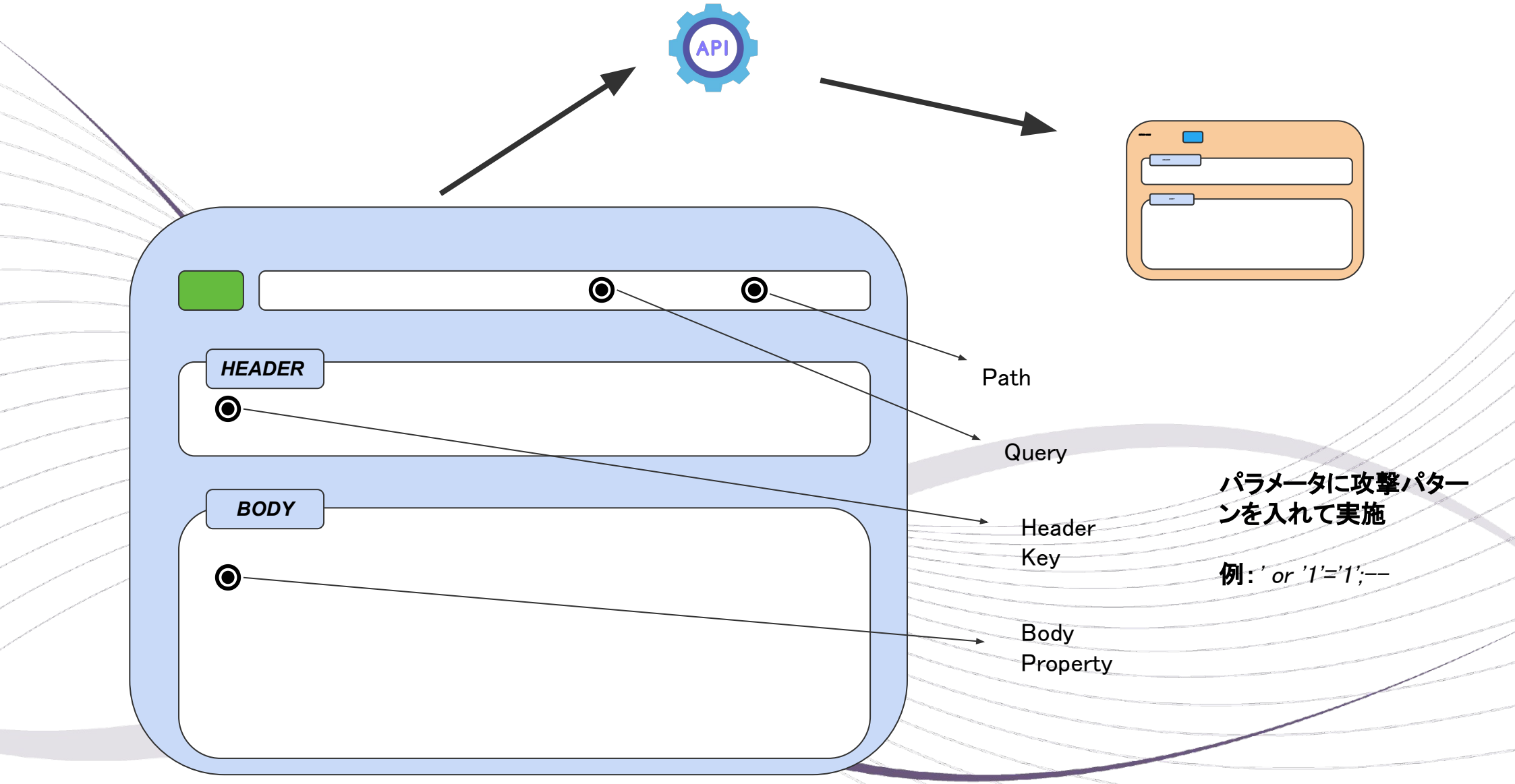
既知の脆弱性パターンに陥っていないか
・SQLインジェクション
・XSS
・XML外部エンティティ



重要情報の不用意な出力があるか

セキュリティ観点

~ 例：SQLインジェクション ~



セキュリティ観点

・例：SQLインジェクション

Body

```
{
  "name": "' or '1'='1';-- ",
  "body_type": [
    1,
    2,
    3
  ],
  "maker": "vinfast",
  "year": 0,
  "color": "u",
```

パラメータに攻撃パターンを入れて実施

例：*' or '1'='1';--*

Title	Headers				
	Content-Type	name	body_type	maker	year
【name】SQLインジェクション	application/json	' or '1'='1';--	[1,2,3]	vinfast	0
【body_type】SQLインジェクション	application/json	vinfast	' or '1'='1';--	j	0
【maker】SQLインジェクション	application/json	vinfast	[1,2,3]	' or '1'='1';--	0
【year】SQLインジェクション	application/json	vinfast	[1,2,3]	vinfast	' or '1'='1';--

セキュリティ観点

~ バグ例 ~

他ユーザの情報をアクセスできる

※同じ組織に所属している場合

セキュリティ観点

~ バグ例 ~

「%」が含まれたものを取得しようとすると、全件データが返ってくる

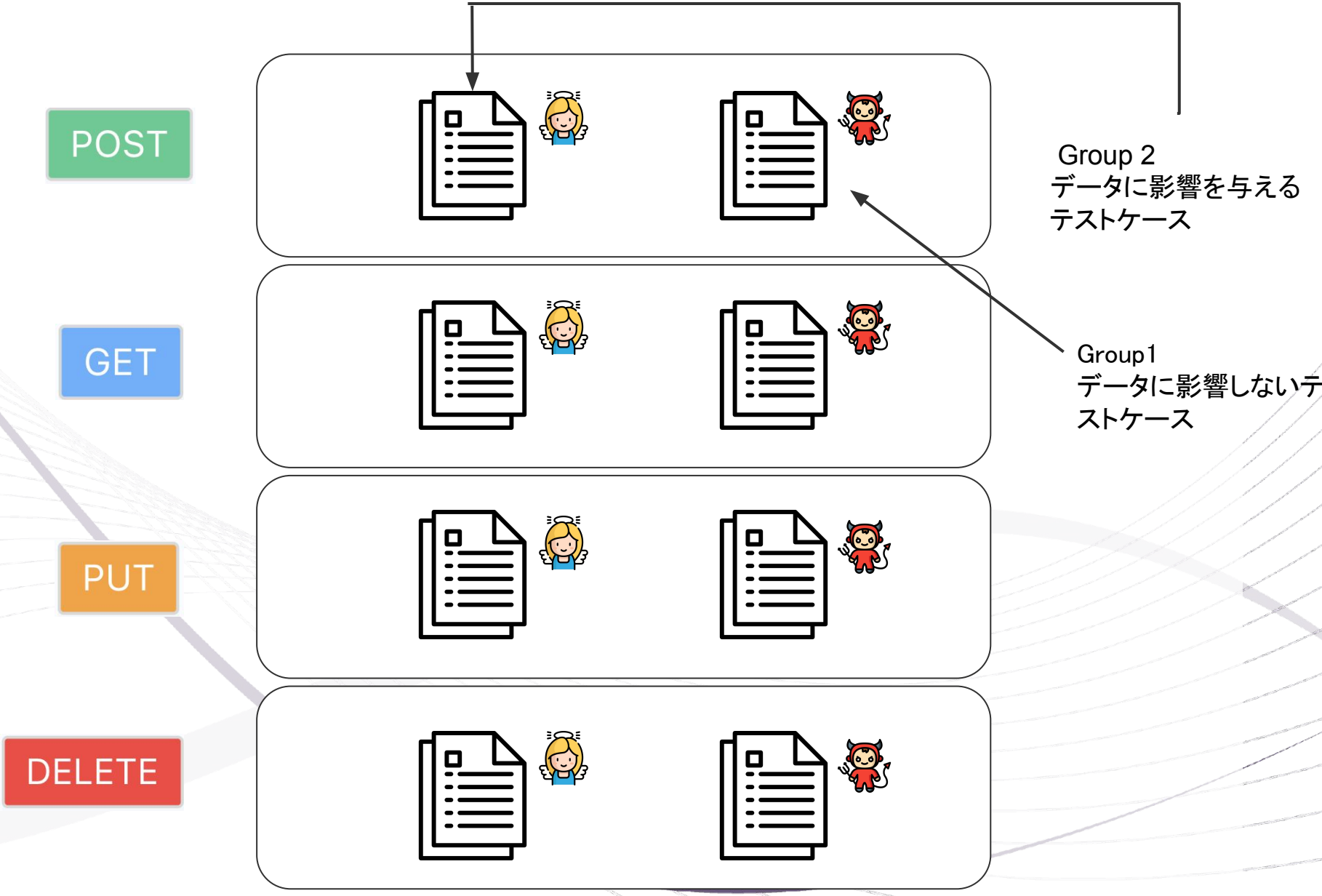
Agenda

- APIテストのメリット
- 観点別で考えるAPIテスト
  - バリデーション観点
  - 機能観点
  - セキュリティ観点
- リグレッションテスト

リグレッションテスト

ほかの箇所に意図しない影響が発生していないを確認する必要がある

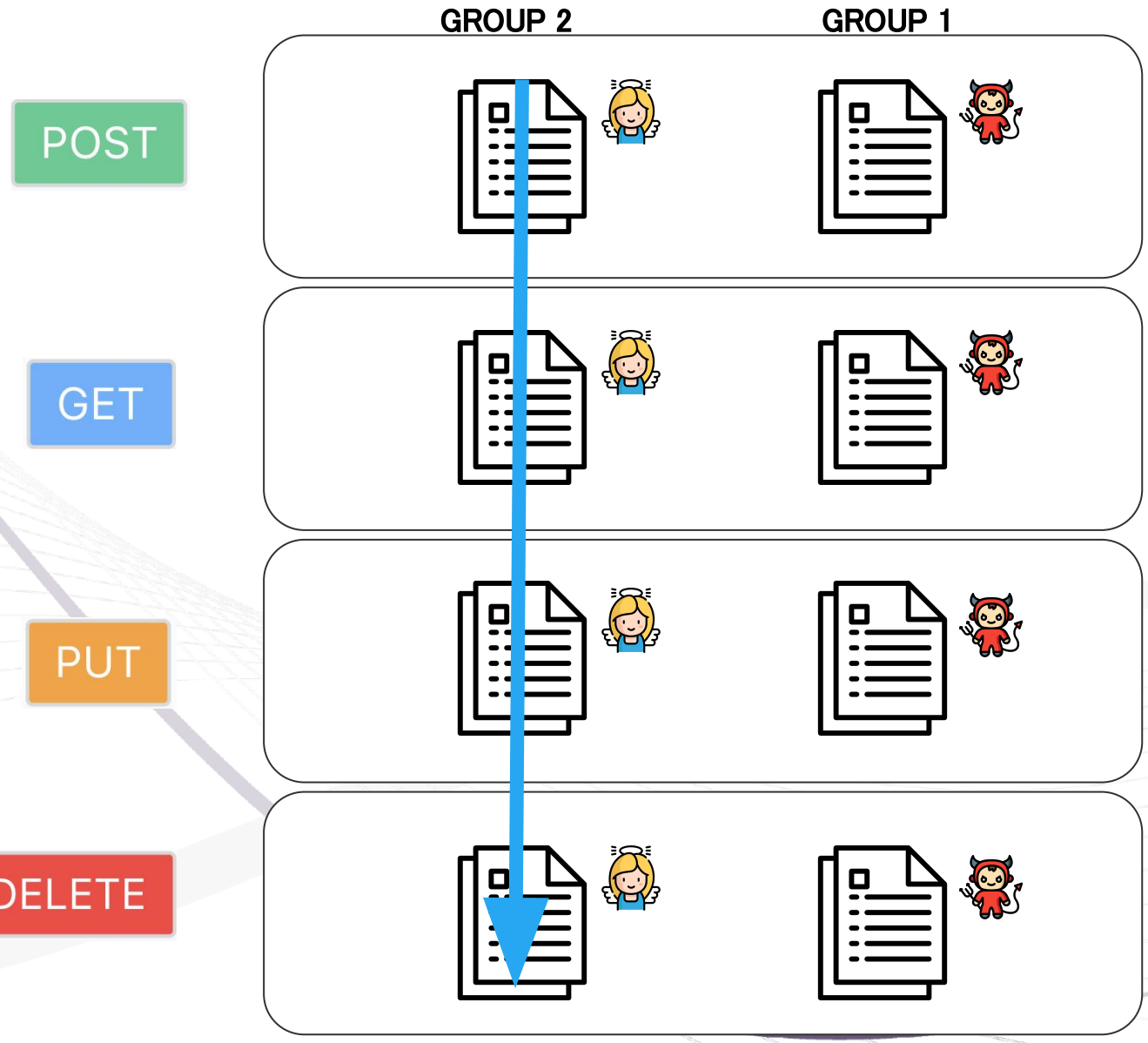
リグレッションテスト



リグレッションテスト

重要パスのシナリオを作成する

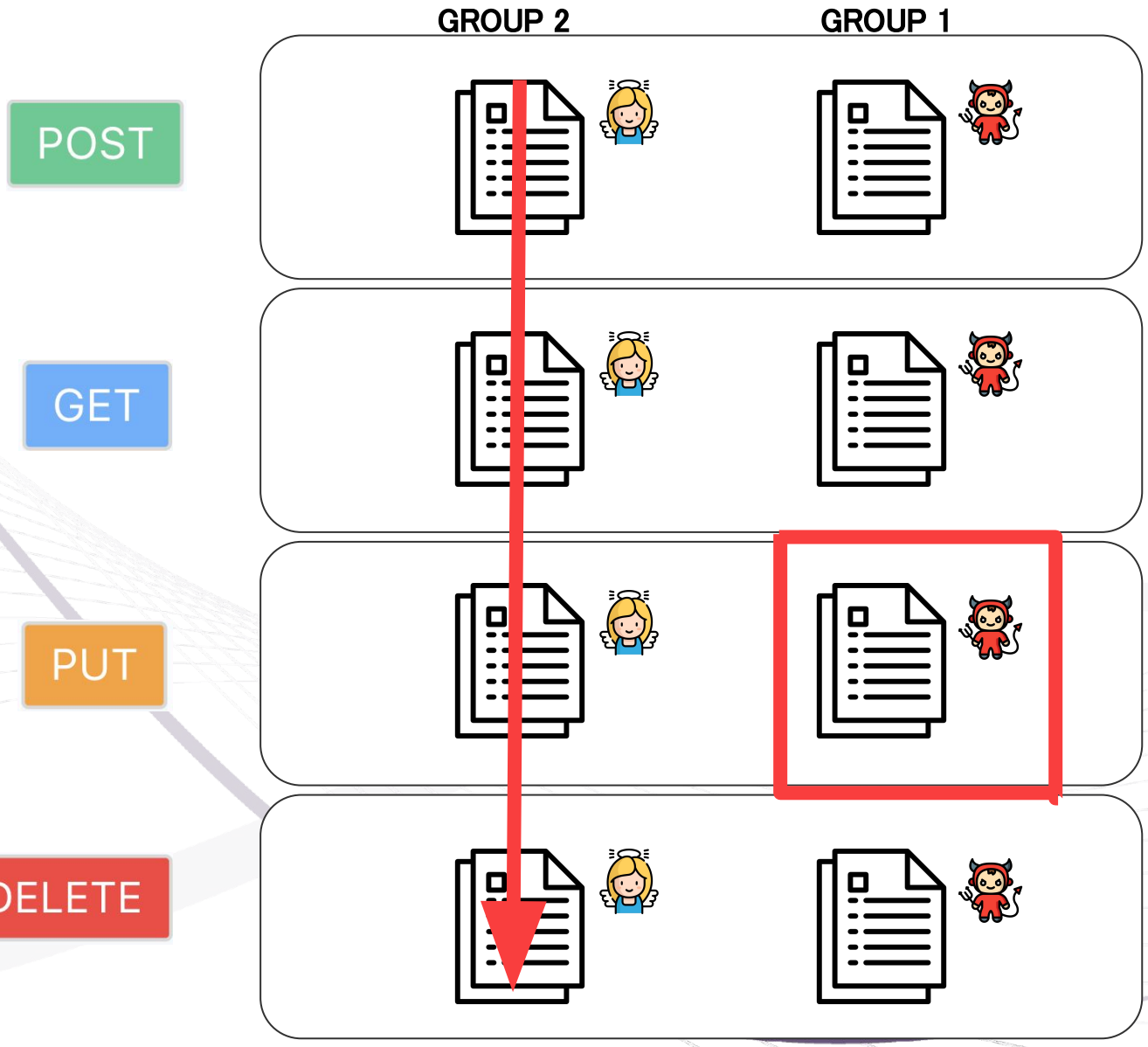
データ作成からデータ削除までのシナリオは理想的



リグレッションテスト

例えば、PUTに影響の可能性がある場合

・Group1：全て実施
・Group2：重要パスのシナリオを実施



まとめ

- APIテストのメリット
- 観点別で考えるAPIテスト
  - バリデーション観点
  - 機能観点
  - セキュリティ観点
- リグレッションテスト

UIテストで実施した場合との比較

	UIテスト	APIテスト
難易度	直観的でやりやすい	学習コストがかかる
実施できるタイミング	UIが完成しないと実施できない	APIの実装が完了したらすぐできる
テスタビリティ	UIによる制限がある	UIテストで実施できないケースも実施できる
実施コスト	実行時間が長い	実行時間が短い
安定性	不安定	安定
自動化	自動化しにくいところがある	自動化しやすい

Human Crest

RAKU API TESTING

RakAPIt

ぜひダナンに来てください。

ダナン

ホイアン

ご清聴
ありがとうございました！