APRESIA®

BDD + Python による
ネットワーク機器のテスト自動化

ネットワーク事業本部

第一技術部 ソフトウェア開発第一グループ

坪田 繁

(shigeru.tsubota.zv@apresiasystems.co.jp)

はじめに

- 自己紹介
  - 名前
    - 坪田　繁
  - 所属
    - ネットワーク事業本部 第一技術部 ソフトウェア開発第一グループ
  - 業務内容
    - 通信事業者向けイーサネットスイッチにおけるOS開発

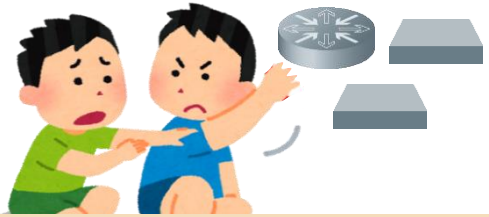
- 本日の内容
  - テスト自動化のモチベーション
  - BDD + Python に至った経緯
  - デモとまとめ

モチベーション（現状）

結合試験項目は数万件

実機試験の時期が重複

手動試験は大変



機材不足

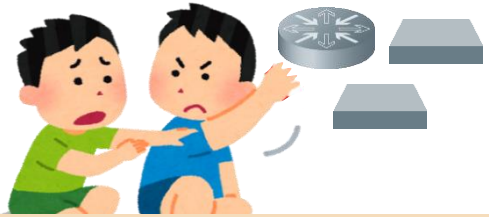
バグ、リリース毎の
機能追加に伴う
品質担保試験

モチベーション（今後）

結合試験項目は数万件

実機試験の時期が重複

手動試験は大変



機材不足

バグ、リリース毎の
機能追加に伴う
品質担保試験

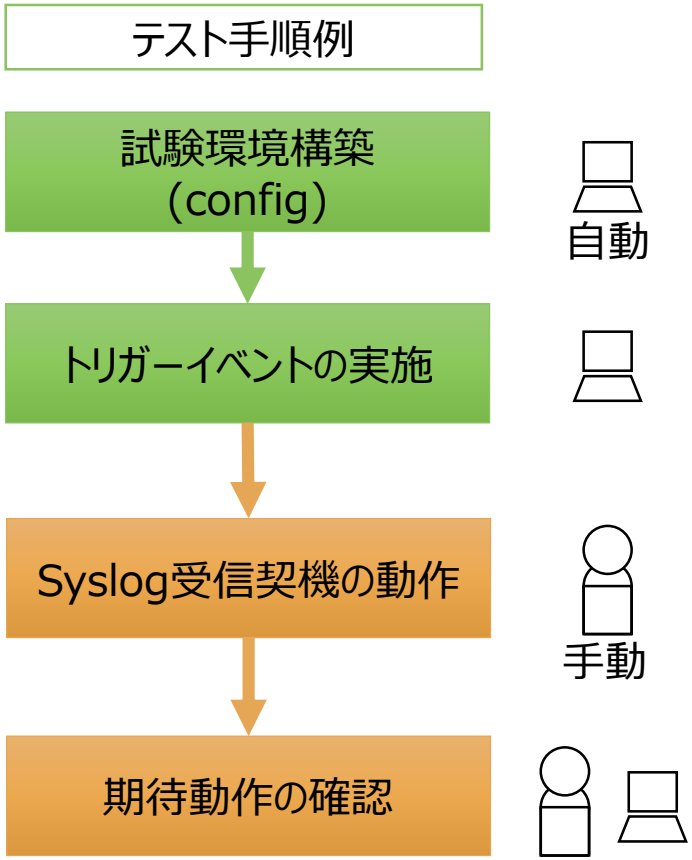
テスト自動化のメリット

夜間/休日も
マシン稼働

回帰試験コスト・
作業負荷の低減

APRESIA Systems におけるこれまでの問題点・課題

意欲的な開発担当者は**独自に**ツールを作成していたが、
自動化率は**約20%**に留まっていた



課題1

ヘルパーライブラリの機能拡充

APRESIA Systems におけるこれまでの問題点・課題

意欲的な開発担当者は**独自に**ツールを作成していたが、
自動化率は**約20%**に留まっていた

課題2

テストフレームワークの選定

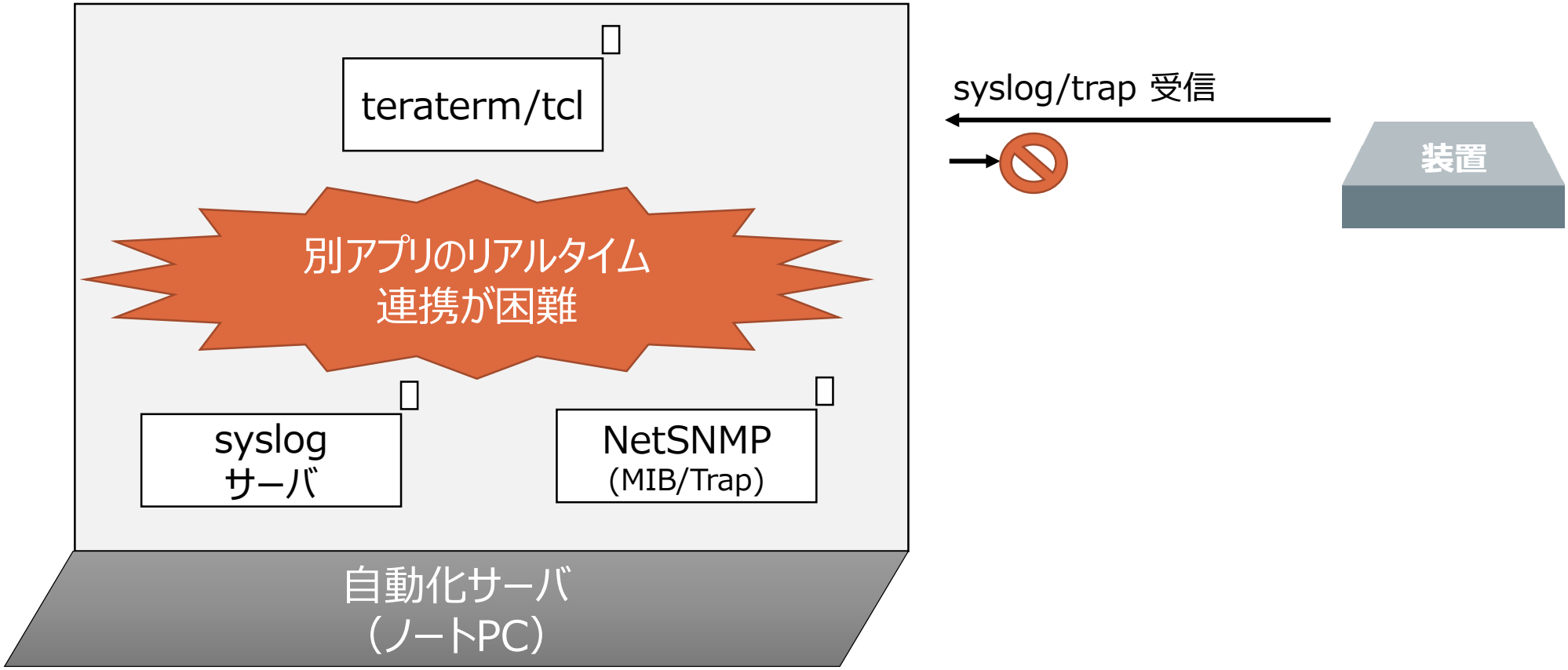
課題1

ヘルパーライブラリの機能拡充

APRESIA SYSTEMS

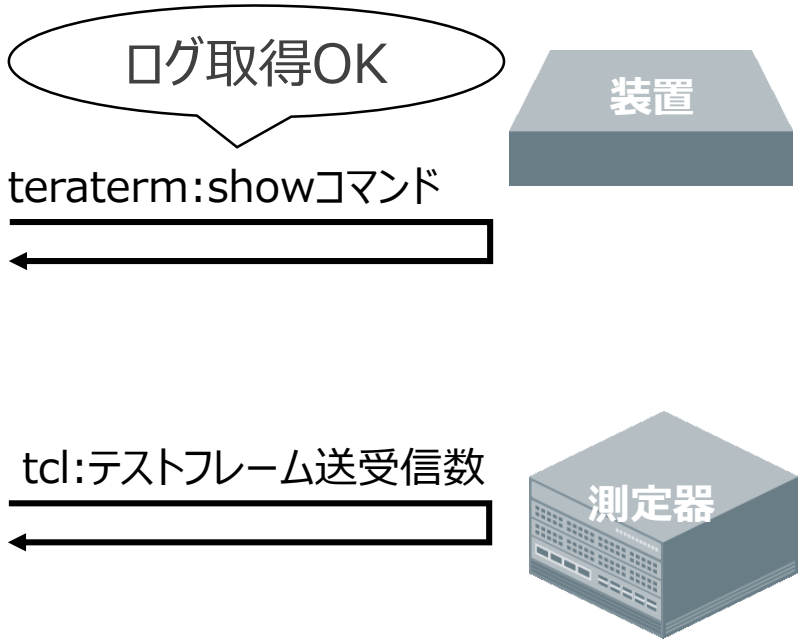
従来の自動化の問題点 (1/2)

◆装置から非同期受信するメッセージに連動　NG



従来の自動化の問題点 (2/2)

◆期待動作の確認 NG



ヘルパーライブラリ向け言語 or ツールの選定

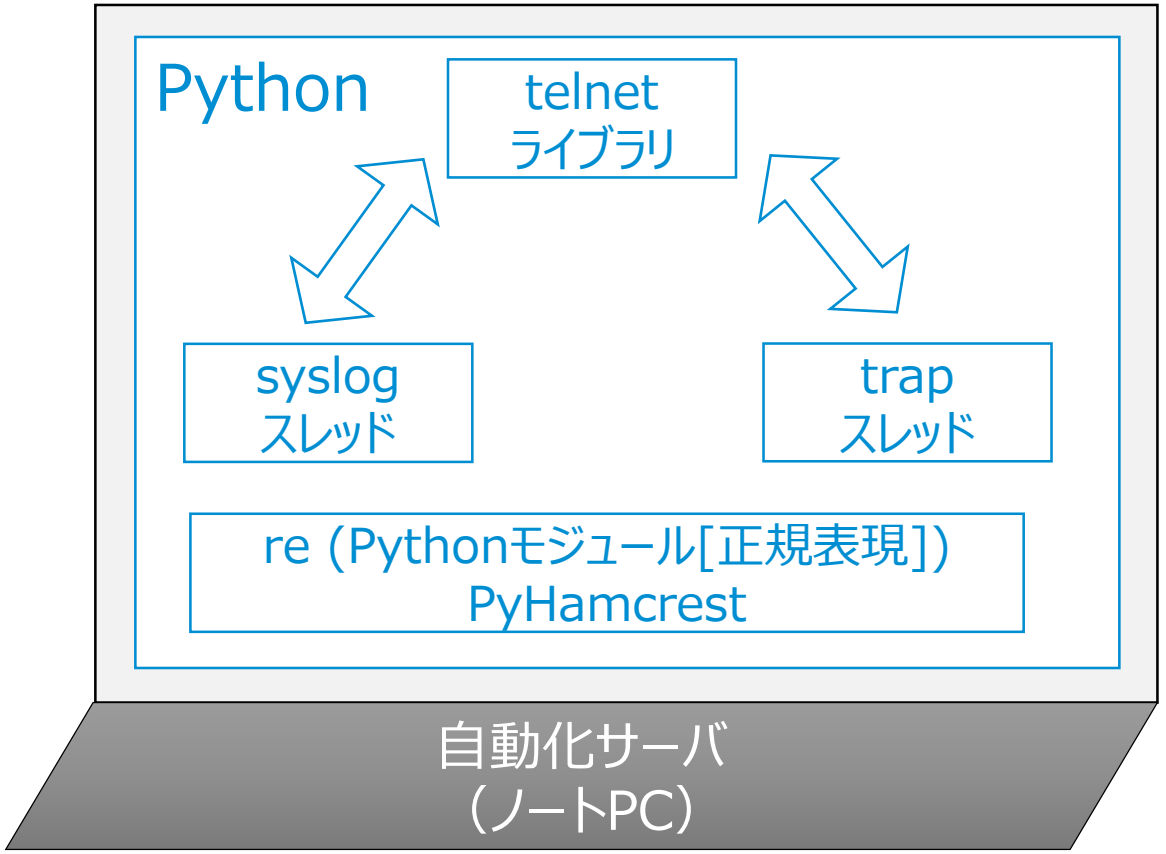
言語 or ツール	config設定	syslog/trap 受信契機の動作	期待動作の確認
tk+teraterm+tcl	○	×	△（実装量大）
tk+teraterm+tcl+ C（サーバ実装）	○	△（実装量大）	△（実装量大）
ansible	○	×	×
Python	**○**	**○**	**○ (re, PyHamcrest)**

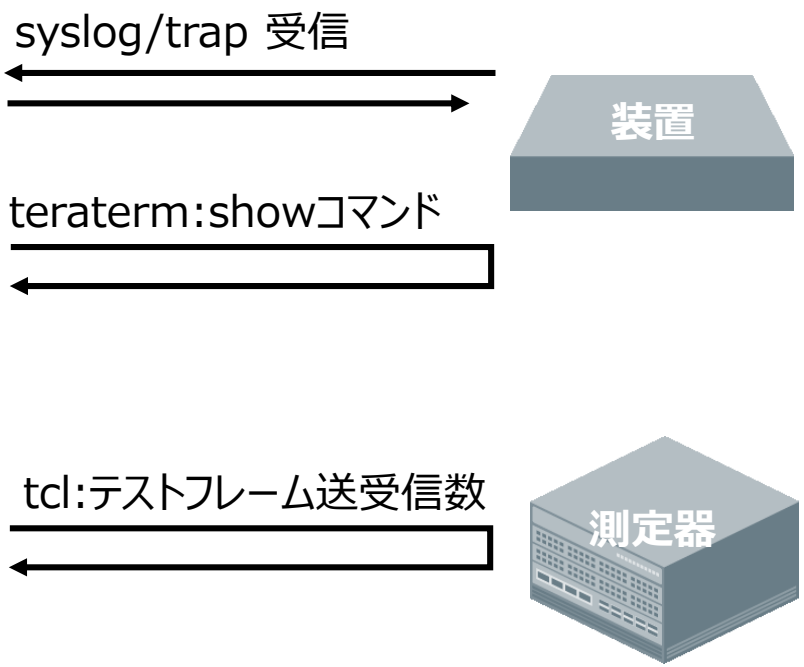
苦労した箇所（例）：
ベンダ独自MIBの認識処理
（pysnmpのドキュメントでは不明で、ソースコードを解析）

Python を採用

パッケージの豊富さ、扱いやすさ、Ver.間の互換性良

Python によるヘルパーライブラリ





※ライブラリの構成は予備スライド参照

課題 2

テストフレームワークの選定

APRESIA SYSTEMS

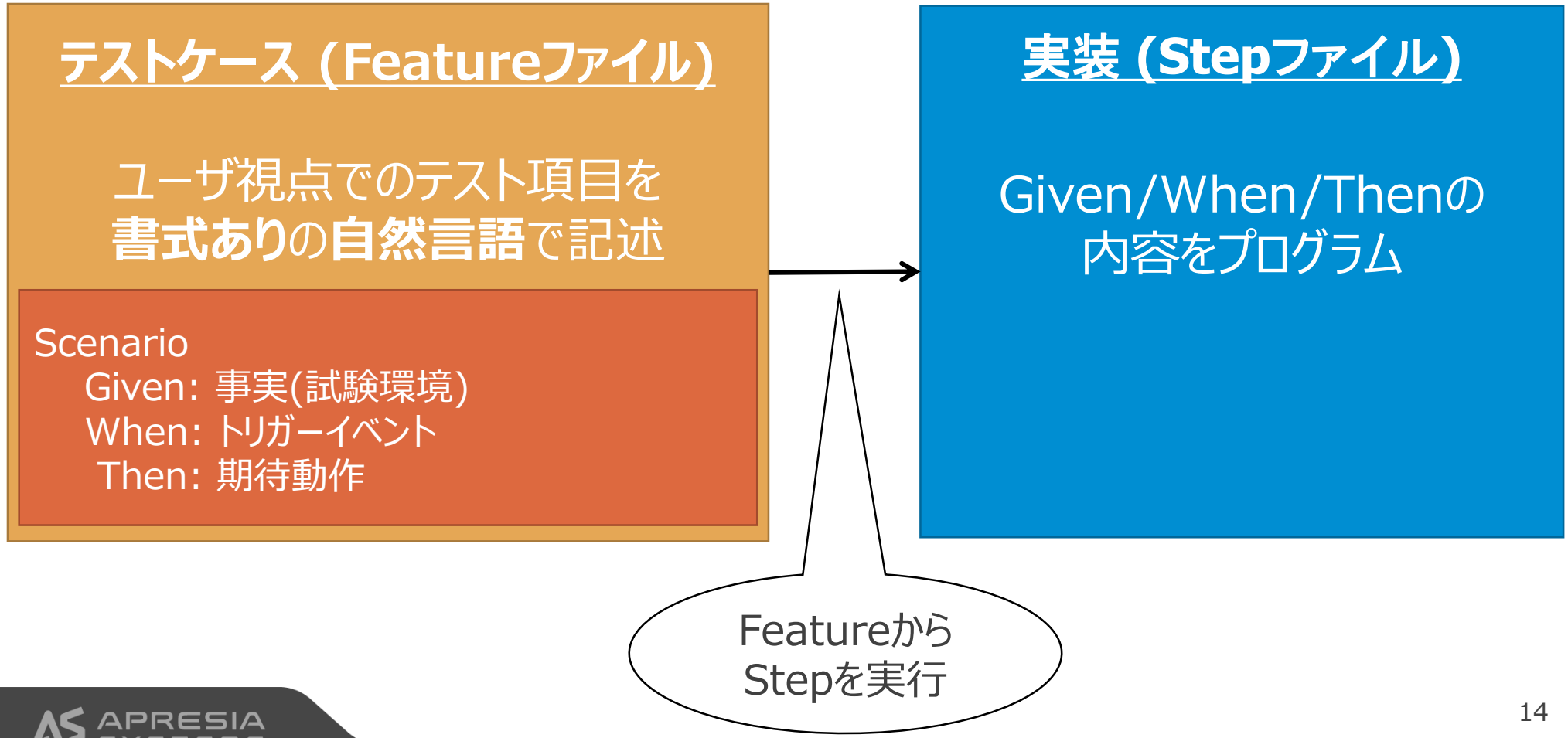
テストフレームワークの検討

ツール		言語	メンテナンス	実績(主観)(2016年2月)
自作		任意	×(コスト大)	ー
xUnit	pytest	Python	○	○
BDD	cucumber	Ruby	○	○
	freshen	Python	×(開発停止?)	ー
	lettuce		×(開発停止?)	ー
	pytest-bdd		○	△
	behave		○	○

検討当初は、xUnit系でいくか、BDDでいくか悩んでいました

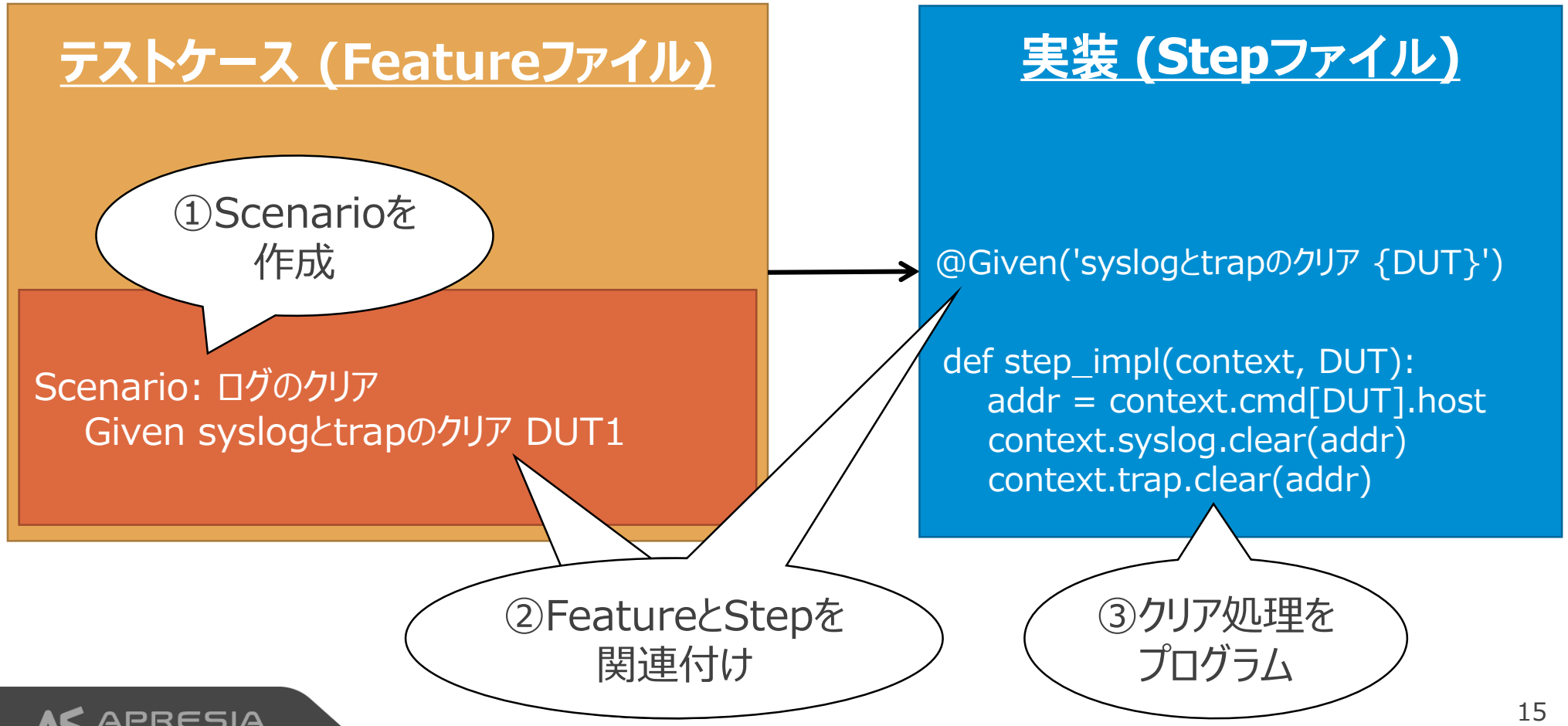
BDD (Behavior Driven Development) のイメージ

- **テストケース**と具体的な**実装**を分けるための仕組み
  - Webの受け入れテストで、よく使われるフレームワーク



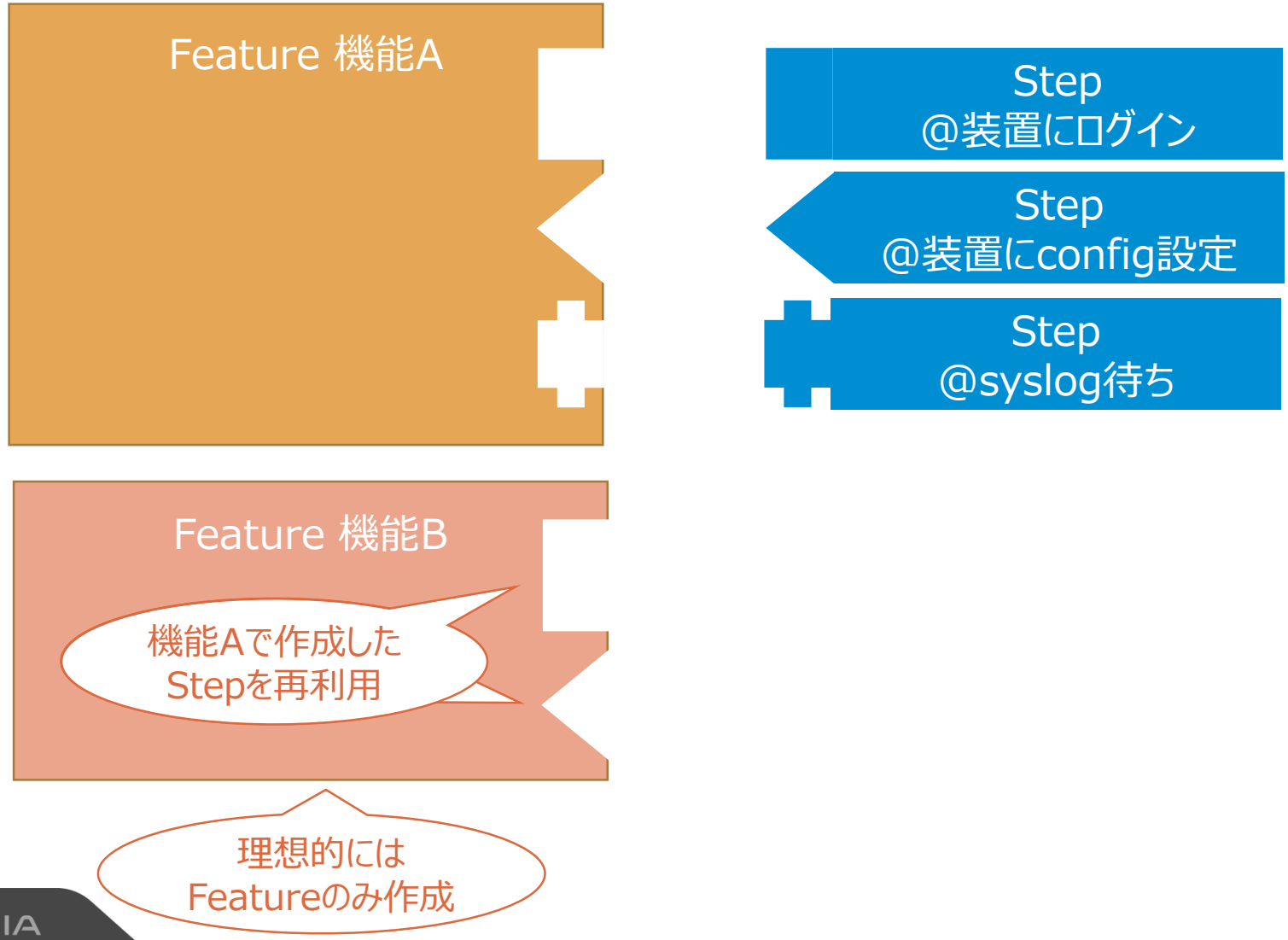
BDD (Behavior Driven Development) のイメージ

◆例：syslogとtrapのクリア



BDD 調査中に感じた可能性（1/2）

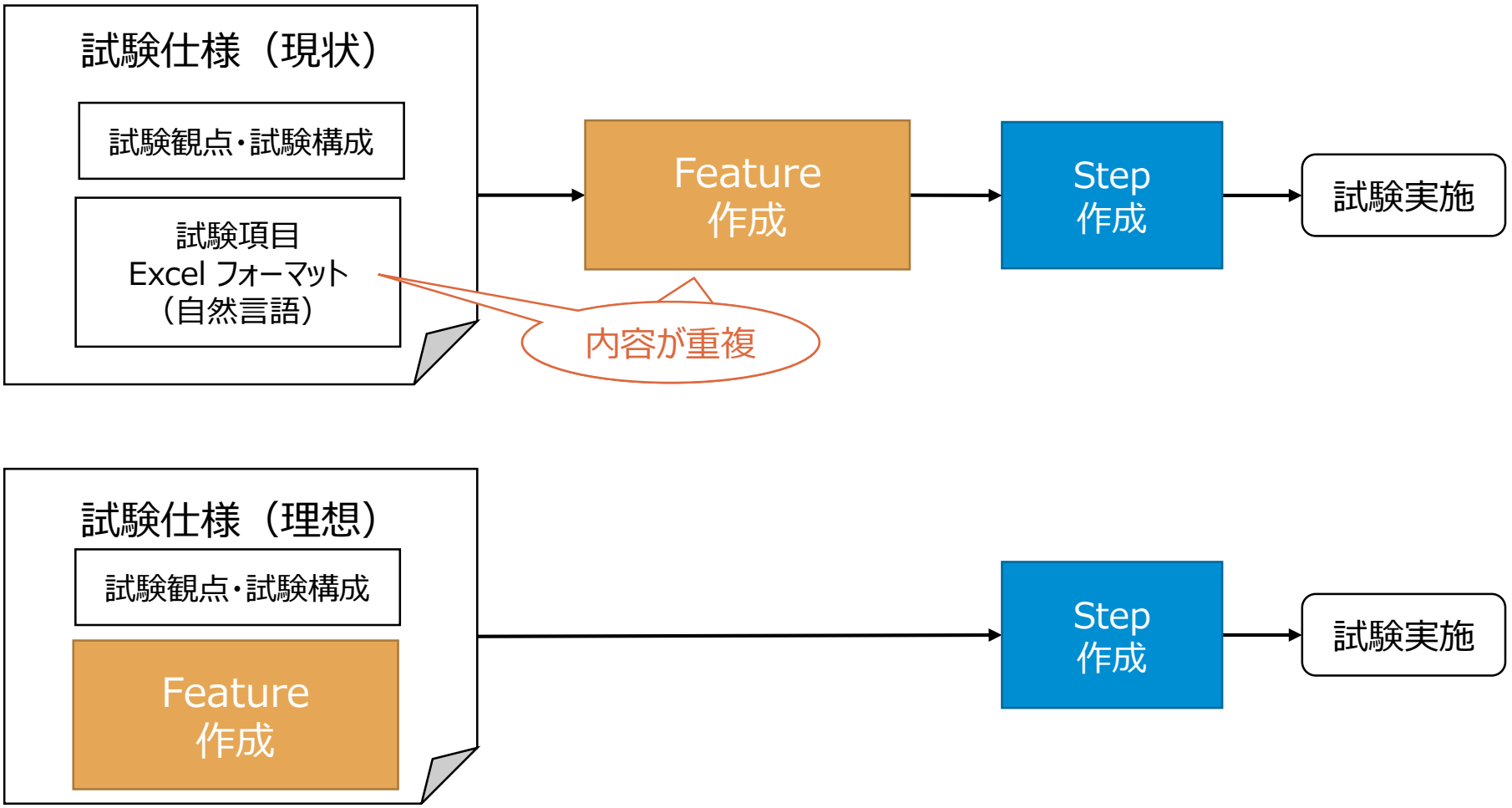
◆**Step** は**再利用**し、テストケース**のみ**追加でいけるのでは？



BDD 調査中に感じた可能性（2/2）

◆いずれ試験項目書を、既存のExcel から **Featureファイルに移行**できる？

※試験観点、試験構成除く



テストフレームワークについて

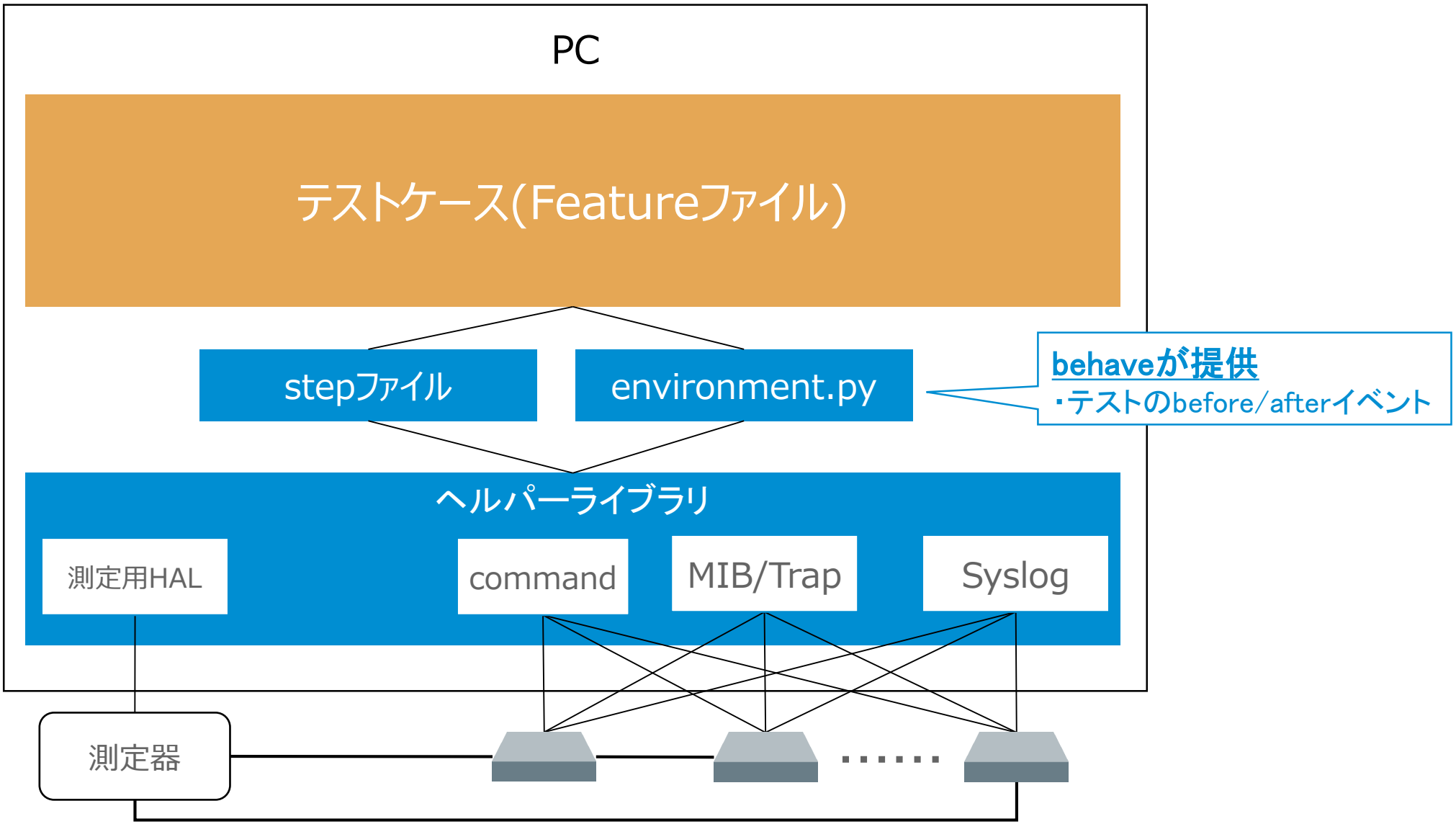
ツール		言語	メンテナンス	実績(主観) (2016年2月)	将来性 (実機試験全体の効率化)
自作		任意	×(初期コスト大)	ー	ー
xUnit	pytest	Python	○	○	×（プログラム寄り）
BDD	cucumber	Ruby	○	○	○
	Freshen	**Python** （ヘルパーライブラリに利用）	×(開発停止?)	ー	○
	lettuce		×(開発停止?)	ー	○
	pytest-bdd		○	△	○
	behave		**○**	**○**	**○**

behave(http://pythonhosted.org/behave/) を採用

当時、総合的に判断してよさそうだった

※現在ならpytest-bdd や、本発表向けに調べ直した中で知った radish を試してみても良いかも

behave + Python によるテストフレームワーク



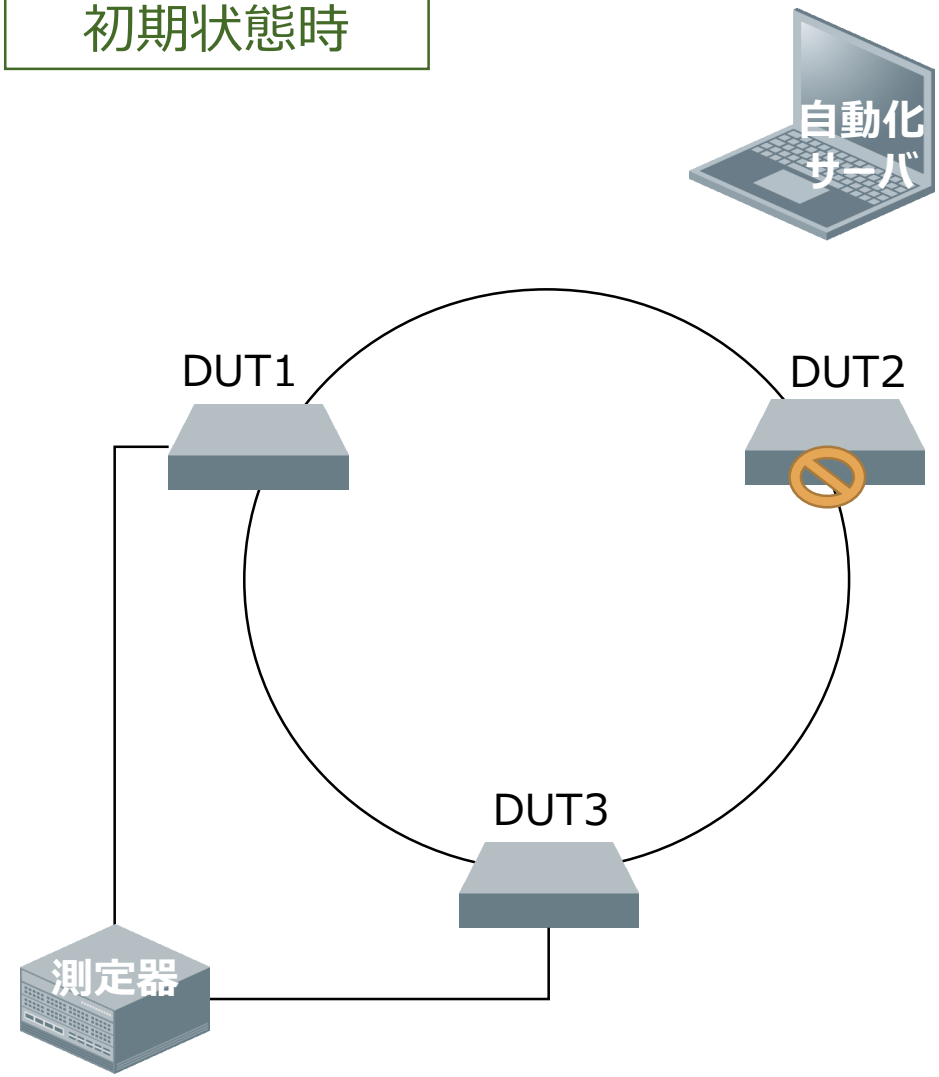
デモ

リング冗長機能の切り替わり時間の
計測自動化

APRESIA SYSTEMS

テストの流れ

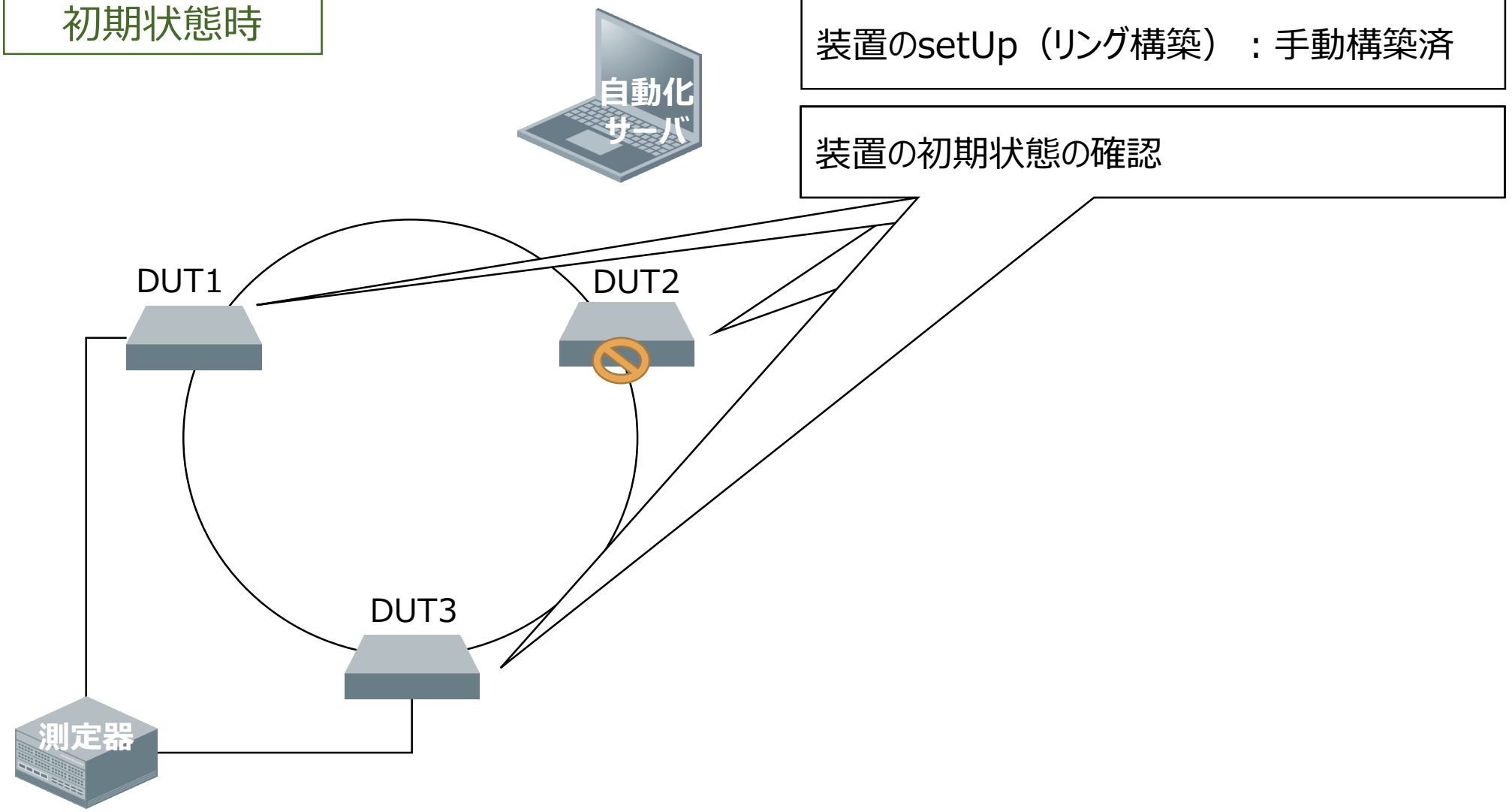
初期状態時



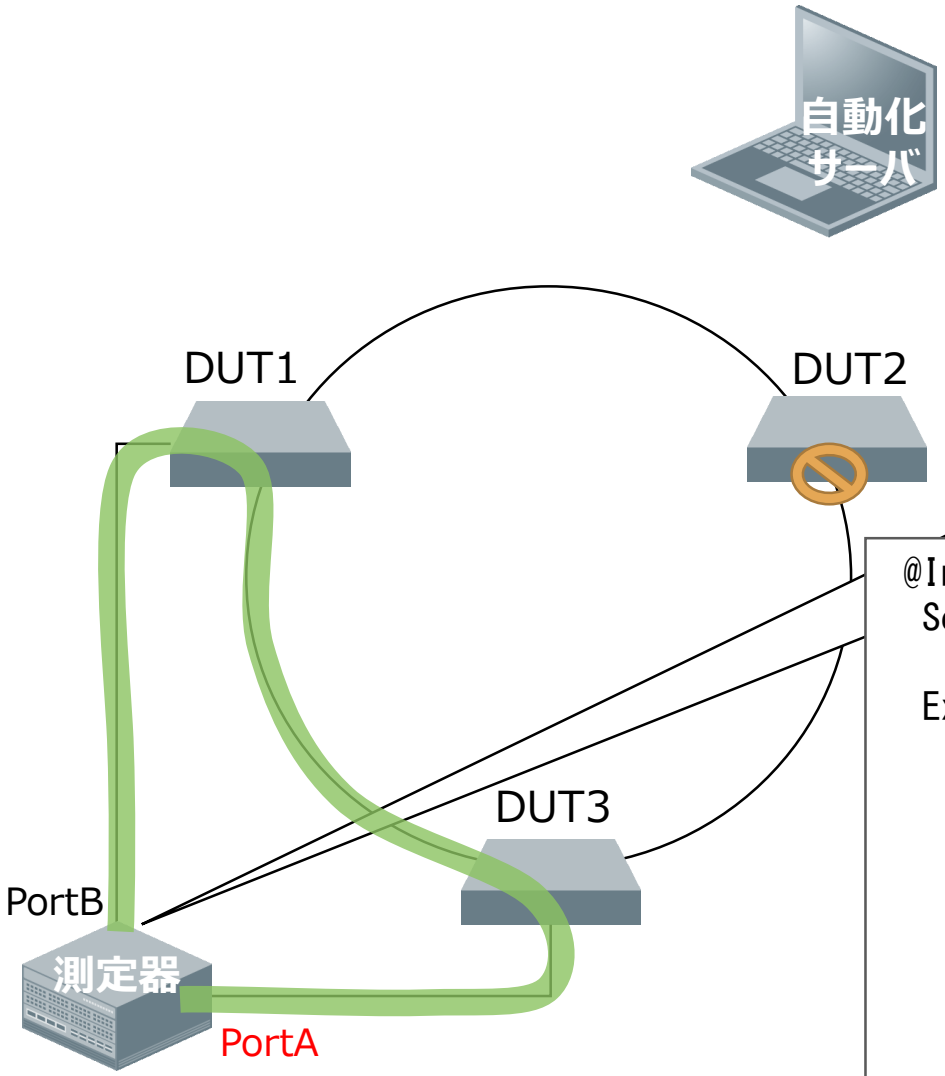
装置のsetUp（リング構築）：手動構築済

テストの流れ

初期状態時



テストの流れ



装置のsetUp（リング構築）：手動構築済

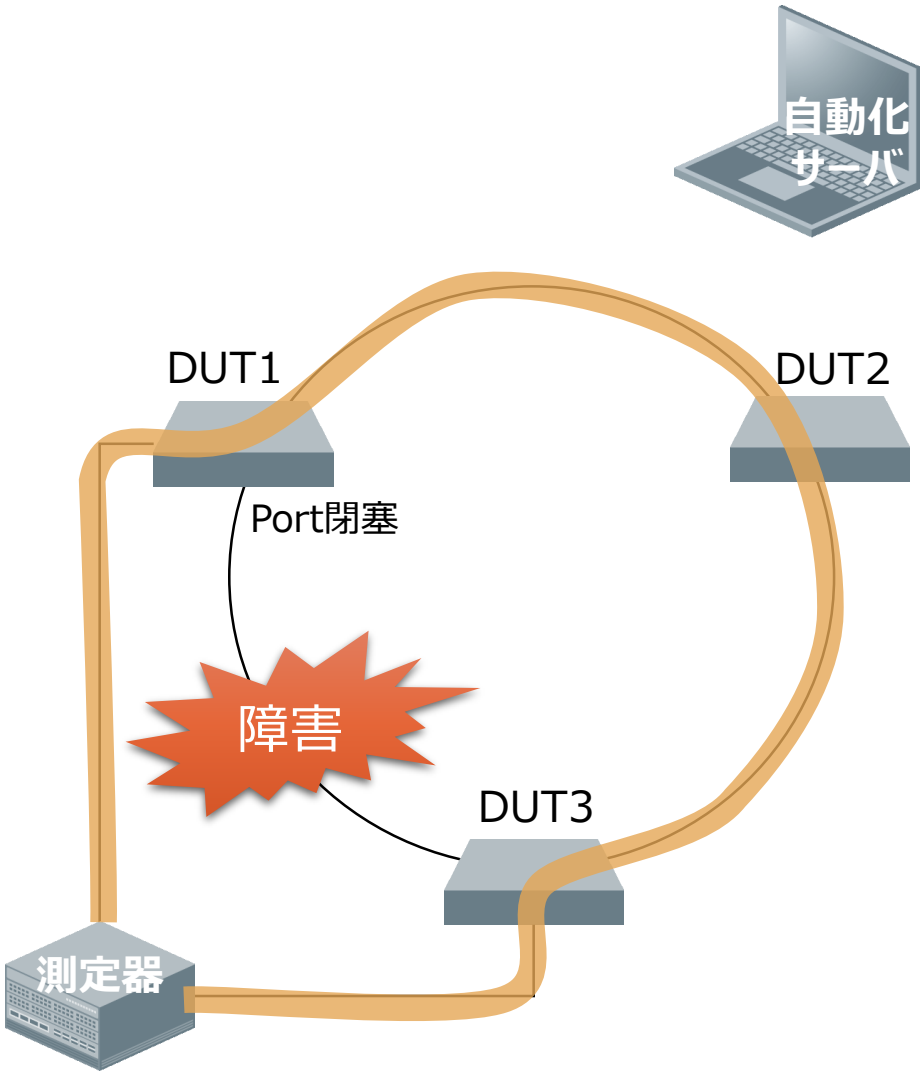
装置の初期状態の確認

テストフレームの登録、送信開始

1001：経路切替向け
1002：経路切替とは独立

```
@Initial
 Scenario Outline: 測定器のsetUp
   Given フレーム登録 <Port> "<DA>" "<SA>" <PID> <VLAN>
 Examples:
    |Port  |DA                |SA                |PID  |VLAN|
    |PortA |00 10 01 02 00 00|00 10 01 01 00 00|1    |1001|
    |PortA |01 00 5E 00 00 00|00 10 01 01 00 00|2    |1001|
    |PortA |00 10 02 02 00 00|00 10 02 01 00 00|3    |1002|
    |PortA |01 00 5E 00 00 00|00 10 02 01 00 00|4    |1002|
    |PortB |00 10 01 01 00 00|00 10 01 02 00 00|5    |1001|
    |PortB |01 00 5E 00 00 00|00 10 01 02 00 00|6    |1001|
    |PortB |00 10 02 01 00 00|00 10 02 02 00 00|7    |1002|
    |PortB |01 00 5E 00 00 00|00 10 02 02 00 00|8    |1002|
```

テストの流れ



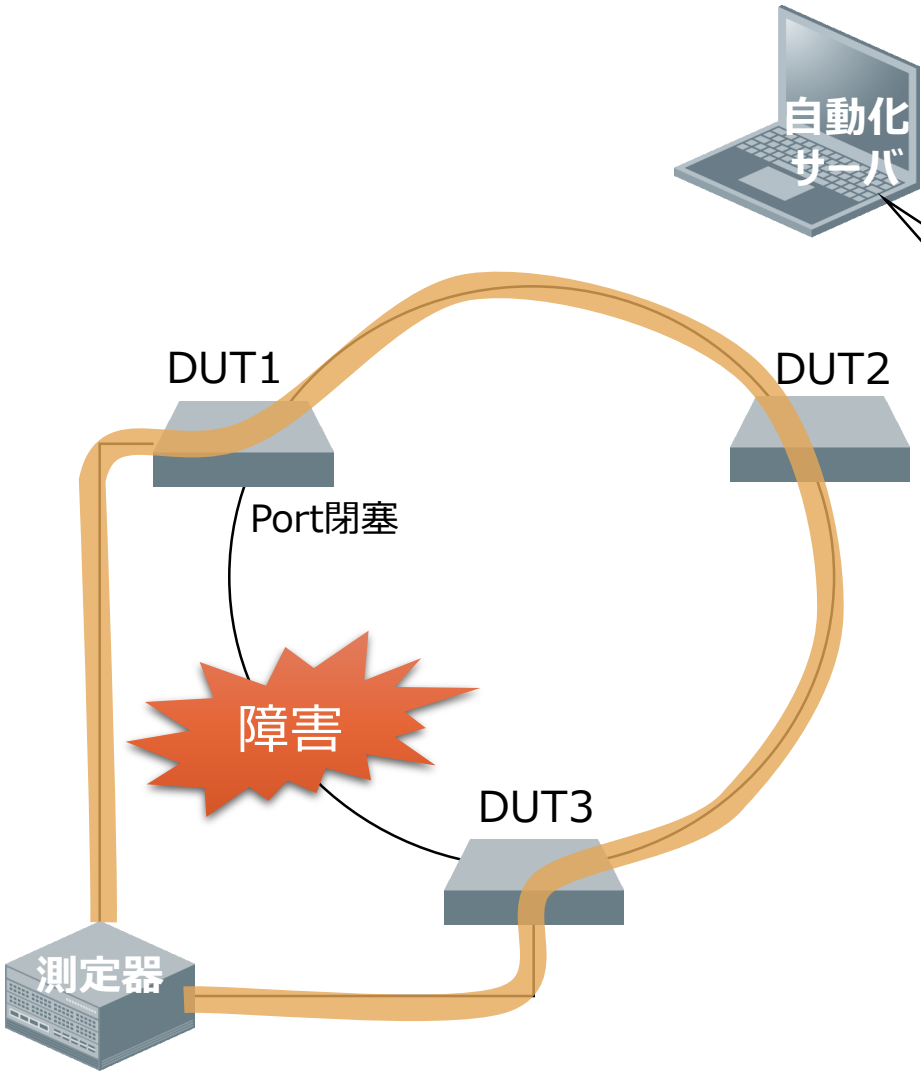
装置のsetUp（リング構築）：手動構築済

装置の初期状態の確認

テストフレームの登録、送信開始

ポート閉塞イベントの発生

テストの流れ



装置のsetUp（リング構築）：手動構築済

装置の初期状態の確認

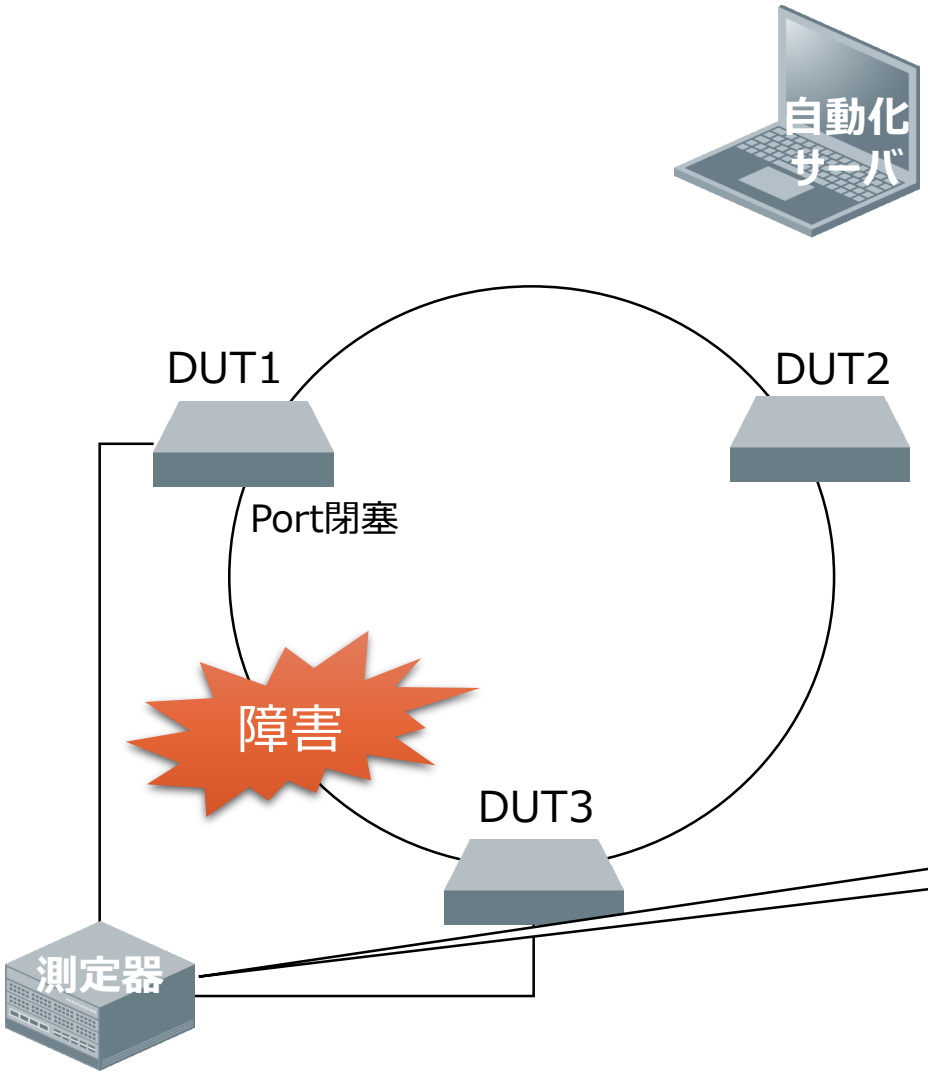
テストフレームの登録、送信開始

ポー　　ベントの発生

syslog/trapの受信待ち

10.249.26.42: <131>Jan 9 08:53:39 (1)
<mmrp:err> Ring 1 ARENA1: Master LAG 1 goes failure.

テストの流れ



装置のsetUp（リング構築）：手動構築済

装置の初期状態の確認

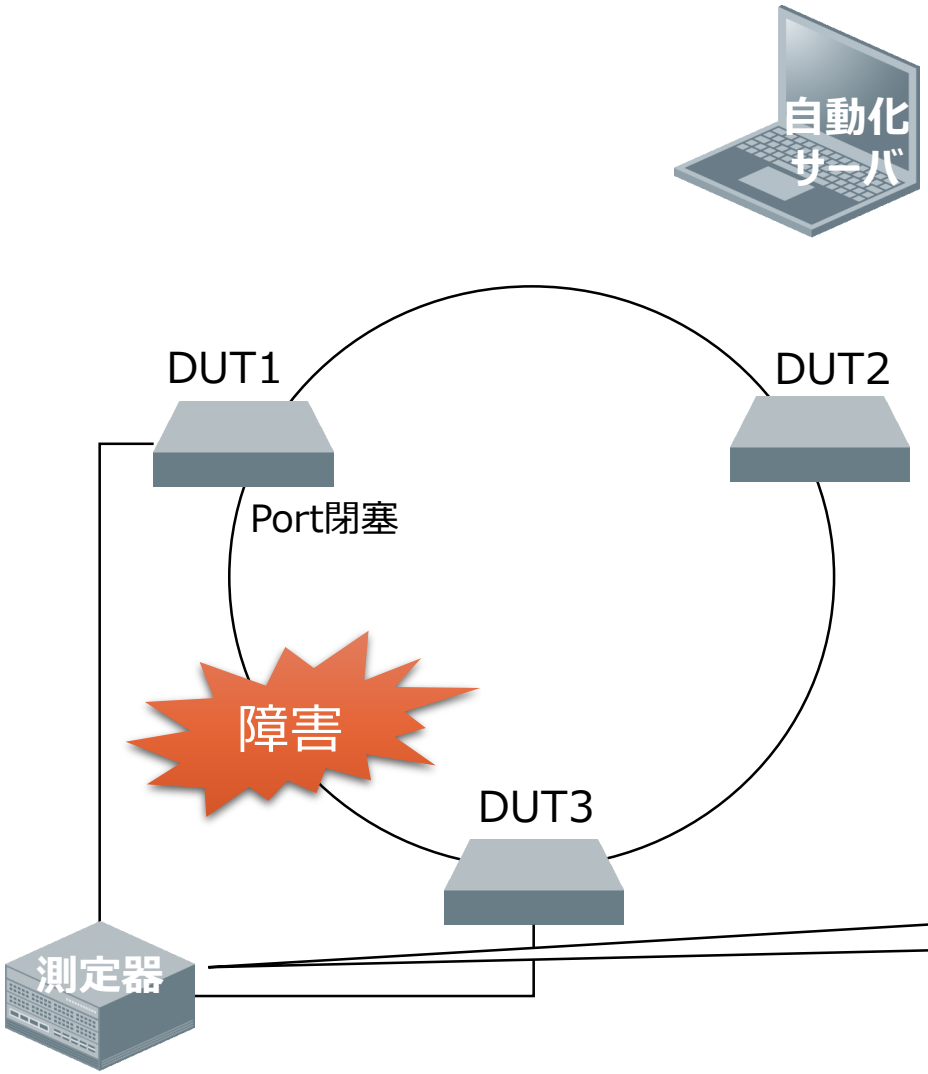
テストフレームの登録、送信開始

ポート閉塞イベントの発生

syslog/trapの受信待ち

テストフレームの送信停止

テストの流れ



装置のsetUp（リング構築）：手動構築済

装置の初期状態の確認

テストフレームの登録、送信開始

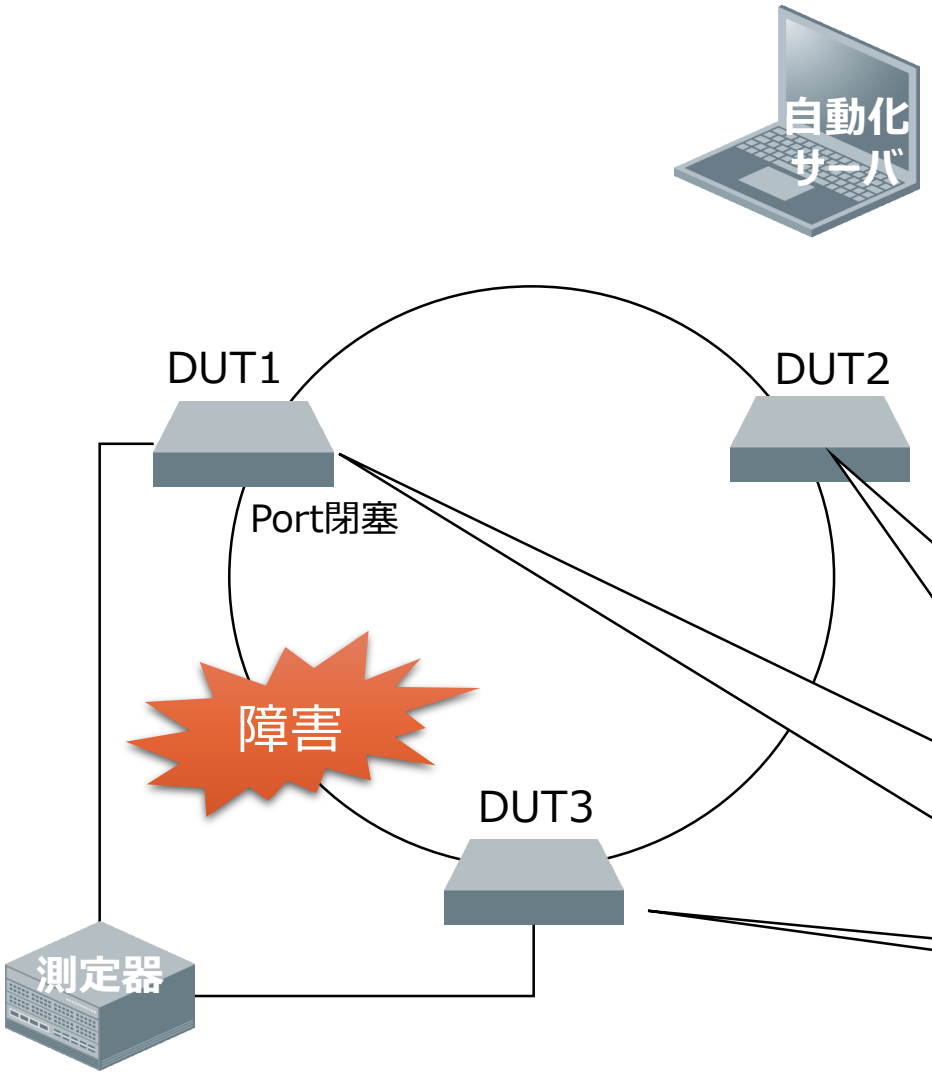
ポート閉塞イベントの発生

syslog/trapの受信待ち

テストフレームの送信停止

経路切替が1秒以内を確認

テストの流れ



装置のsetUp（リング構築）：手動構築済

装置の初期状態の確認

テストフレームの登録、送信開始

ポート閉塞イベントの発生

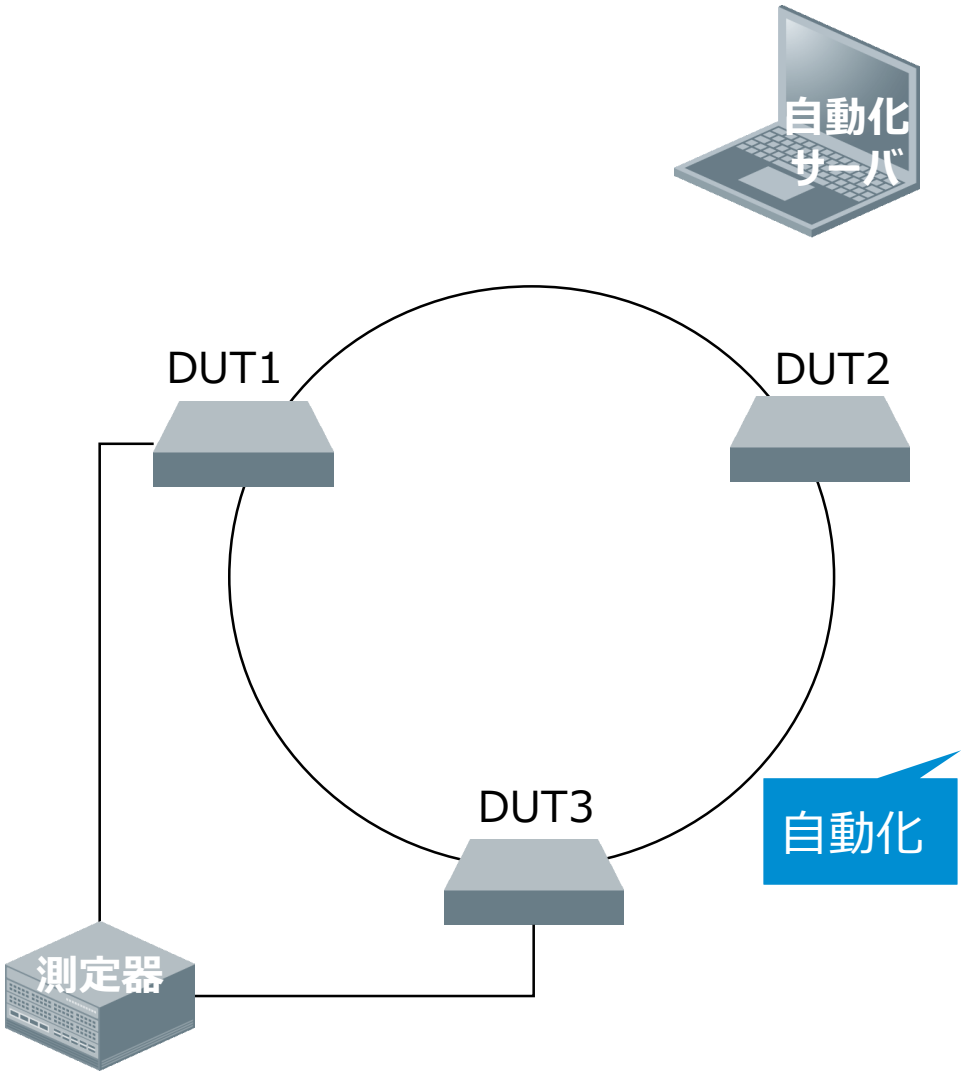
syslog/trapの受信待ち

フレームの送信停止

秒以内を確認

リング状態の確認

テストの流れ



装置のsetUp（リング構築）：手動構築済

自動化

- 装置の初期状態の確認
- テストフレームの登録、送信開始
- ポート閉塞イベントの発生
- syslog/trapの受信待ち
- テストフレームの送信停止
- 経路切替が1秒以内を確認
- リング状態の確認

デモ画面

考察と

今後の課題

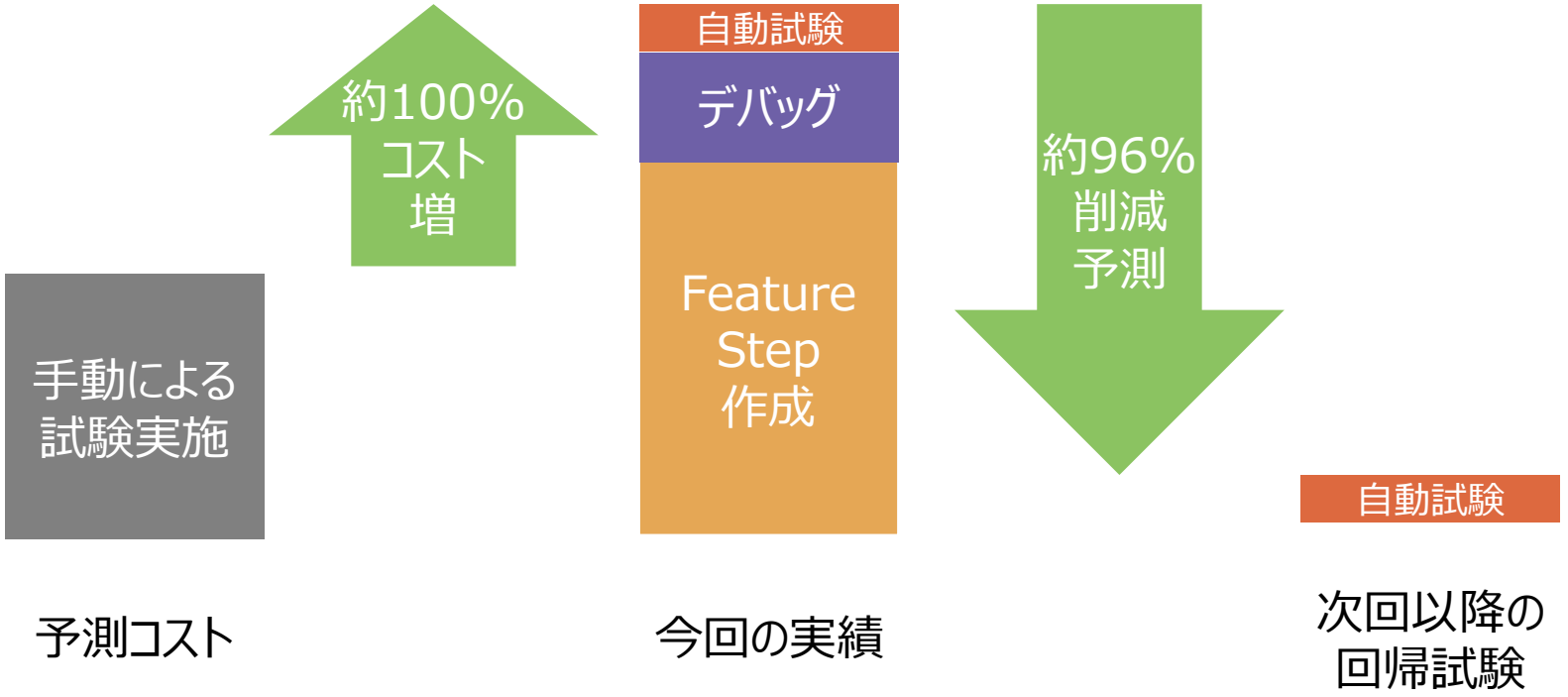
APRESIA SYSTEMS

テスト自動化してみてわかったこと

- 試験項目書のレビュー時には気づけなかった間違いに気づく
- テスト項目の粒度が機能毎に異なると再認識
  - テストケース(Feature)記述にも影響
- 手動テストがパスしているテストの自動テスト失敗時に、テストコードとテスト対象のどちらの問題かを切り分るのが難しいケース有
  - 慣れてくるとテストコードは正しいことが意外と多い

工数削減効果

◆シャーシ型装置の立ち上げプロジェクト(2016年度)の結合試験に適用



※フレームワークの検討とOnboardingのコスト除く

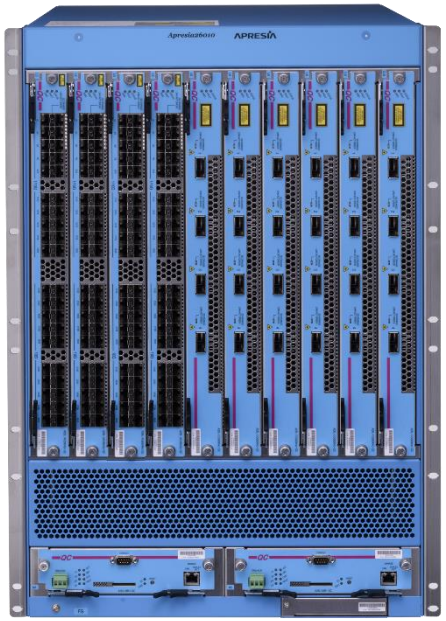
2回目以降の回帰試験で
コスト回収が可能

テスト自動化の網羅率

◆結合テストの**67%**を自動化

◆未対応箇所は、主に人（手や目）が必要な物理操作

ラインカード/ケーブル等
の挿抜



装置電源ON/OFF

LEDの
点灯・点滅確認

疑似障害試験
（専用基板のDIPSW）

今後の課題

- APIの拡充
  - L1スイッチ対応
  - 他ベンダとの相互接続対応

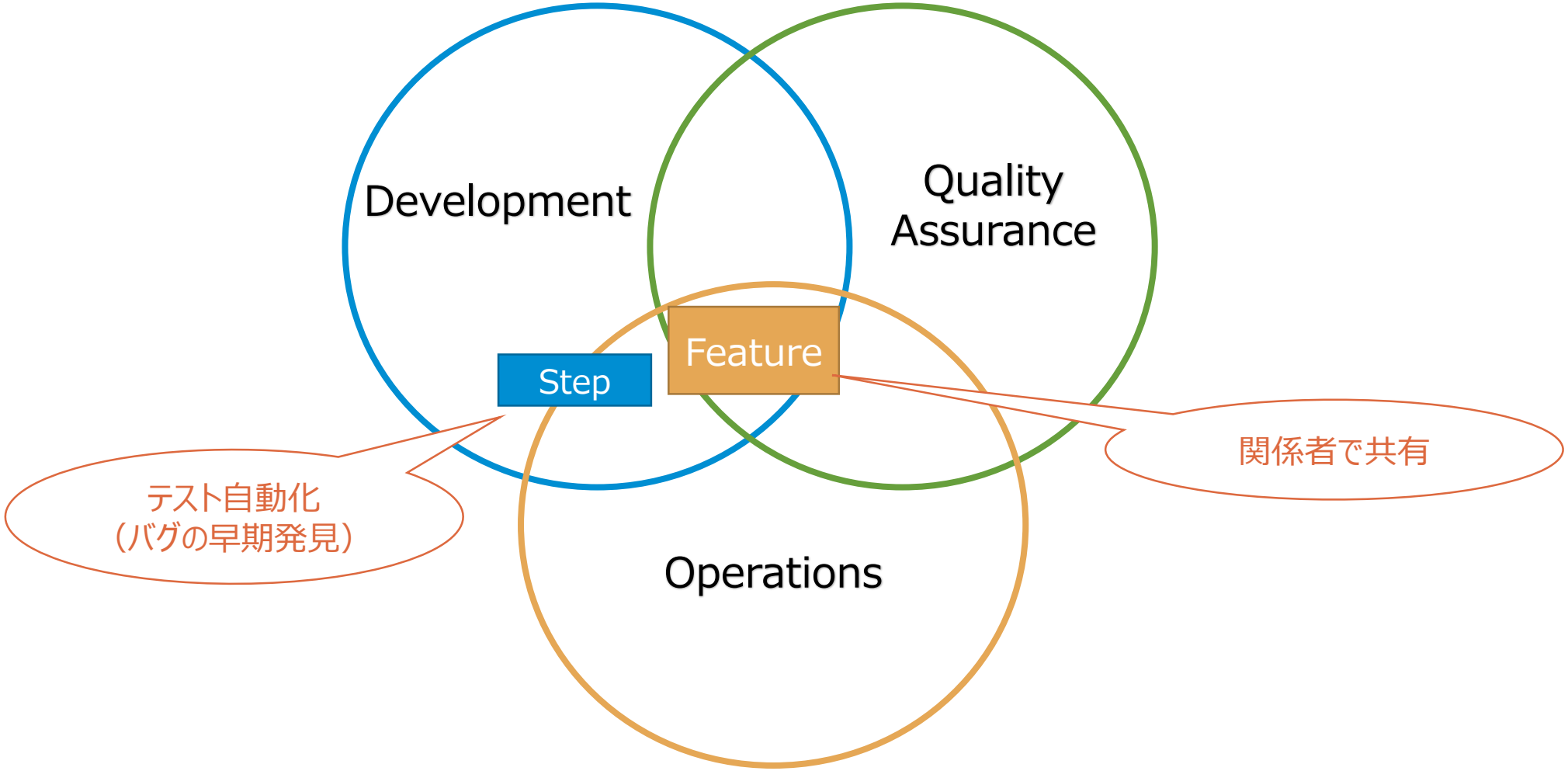
- テスト項目書とテストケース（Featureファイル）の二重管理の解消

以後、予備スライド

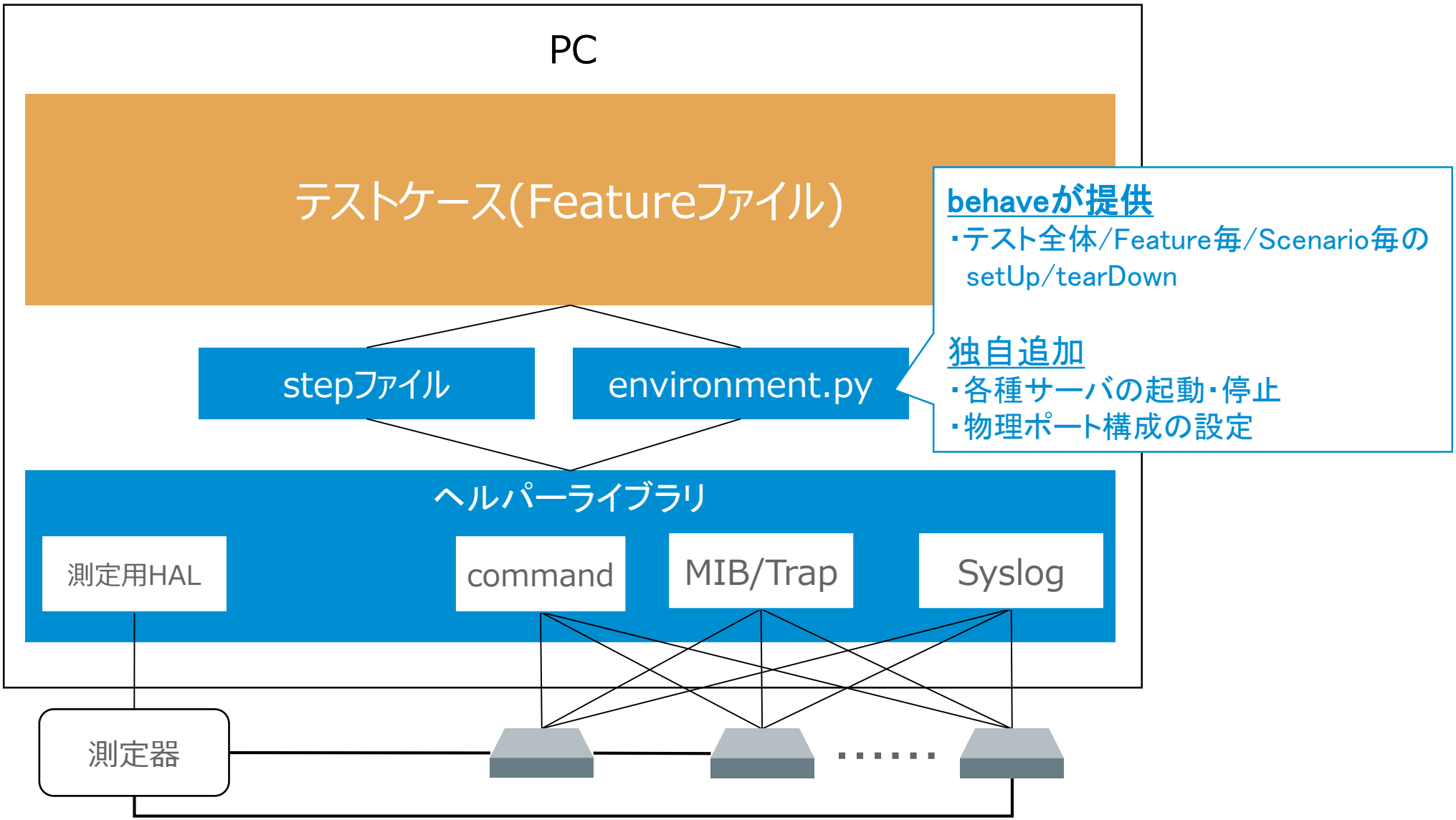
APRESIA®

BDD 調査中に感じた可能性

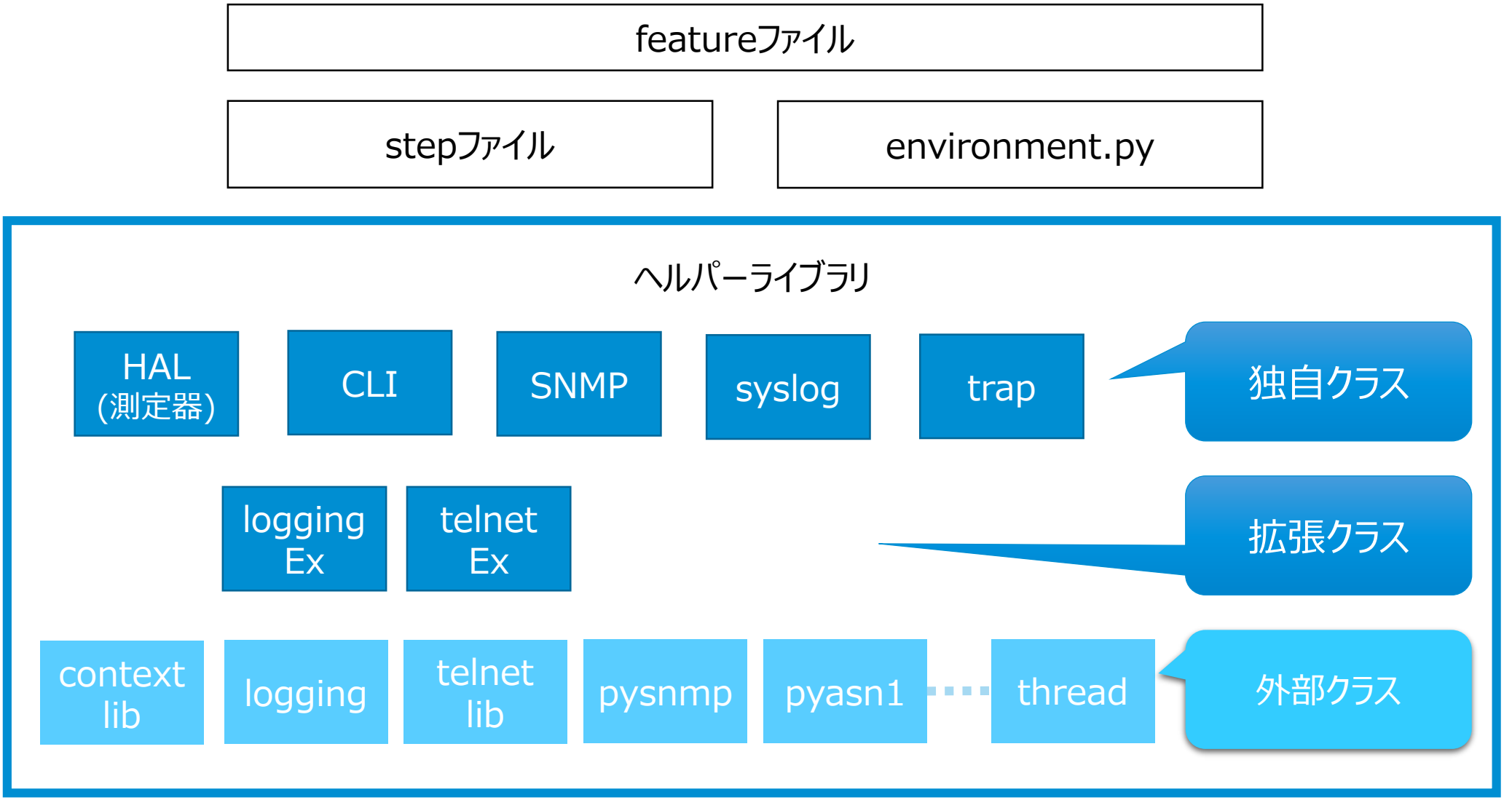
◆将来、**DevOps を支援**できないか？



behave + Python によるテストフレームワーク



ヘルパーライブラリの構成について



テスト１フレームロス時間の確認

行単位で上から順に実行

```
 @01
  Scenario Outline: TEST 01 (loss time check)
    Then フレームロス時間確認 <TxPort> <MinStrmID> <MaxStrmID> <RxPort> <MinPGID> <MaxPGID>
<ExpectLossTime>
  Examples:
    |TxPort|MinStrmID|MaxStrmID|RxPort|MinPGID|MaxPGID|ExpectLossTime|
    |PortA |1        |2        |PortB |1      |2      |1.0           |
    |PortA |3        |4        |PortB |3      |4      |0.0           |
    |PortB |1        |2        |PortA |5      |6      |1.0           |
    |PortB |3        |4        |PortA |7      |8      |0.0           |
```

切替に関係のないTrafficはロスレス

1秒以内切替

テスト１リング状態の確認

```
  @01
  Scenario Outline: TEST 01 (リングの状態遷移確認)
    Then MMRPの状態確認 <DUT> "<ring_ID>" "<Name>" "<Port1>" "<Mode1>" "<Status1>" "<Port2>"
"<Mode2>" "<Status2>"
  Examples:
    |DUT|ring_ID|Name   |Port1|Mode1 |Status1          |Port2|Mode2|Status2          |
    |Ap1|1      |ARENA1|LAG1 |Master|failure(d)  [U]|-    |-    |-                |
    |Ap2|1      |ARENA1|-    |-     |-                |LAG1 |Slave|Forwarding  [U]|
    |Ap3|1      |ARENA1|LAG1 |Aware |failure(d)  [U]|LAG3 |Aware|Forwarding  [N]|
```

AP1とAP3のリンクを落としたため、状態がfailure(d)に遷移していることを確認
（dはdownの略）

AP2はTrafficを迂回するため、
Forwarding状態に遷移していることを確認