NetTesterで
多拠点テストも自動化！

沖縄オープンラボラトリ
ネットワークテストシステム プロジェクト
（新日鉄住金ソリューションズ株式会社）田島 照久

JANOG41 ネットワーク運用自動化BoF
2018/01/24 @広島国際会議場

プロビジョニングを自動化して
5分で環境が作れるシステムを作った

半日かけて
サービスが問題なく動くか
手作業でテストをする

サービス提供の律速＝NWテスト

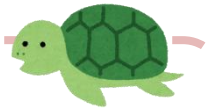
サービスプロバイダは素早くサービスを提供したい

サーバ/アプリケーション

- ■設定
  - ■各種プロビジョナ
- ■テスト
  - ■テストコード/○○spec
  - ■各種テストツール
  - ■CI/CDサービス

ネットワーク

- ■設定
  - ■各種プロビジョナ発展中
- ■テスト
  - ■出張
  - ■人海戦術
  - ■温かみのある手作業



NWテストの自動化によりボトルネック解消

NW(だけ)のテストは従来つまらなかった

OKINAWA OPEN LABORATORY

システム構築にNWのテストは大切

- 繋がること自体を保証しなければならない
- NWは自律分散システムなので挙動を確かめる必要あり

しかし現実は・・・

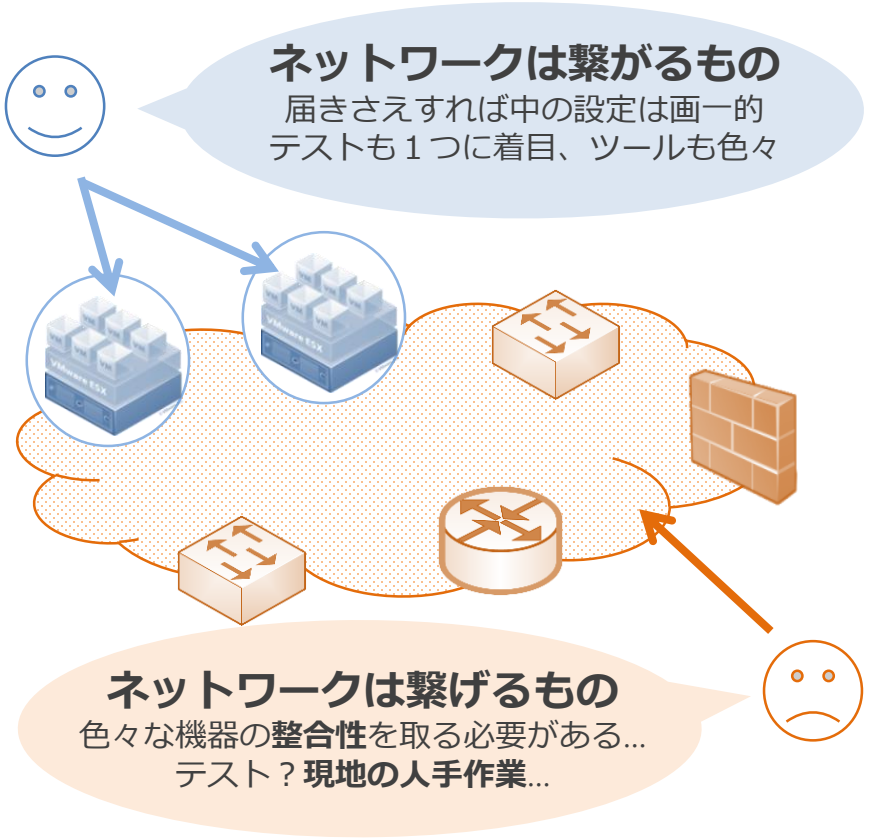
- 物理作業を伴う人力作業でスケールしない
- 系全体のテストが難しいので単体のテストしかできない
- 提供するサービスレイヤーと一致しない

NW全体の動作を自動テストできるように

なぜNWのテストは自動化が進まないのか

OKINAWA OPEN LABORATORY

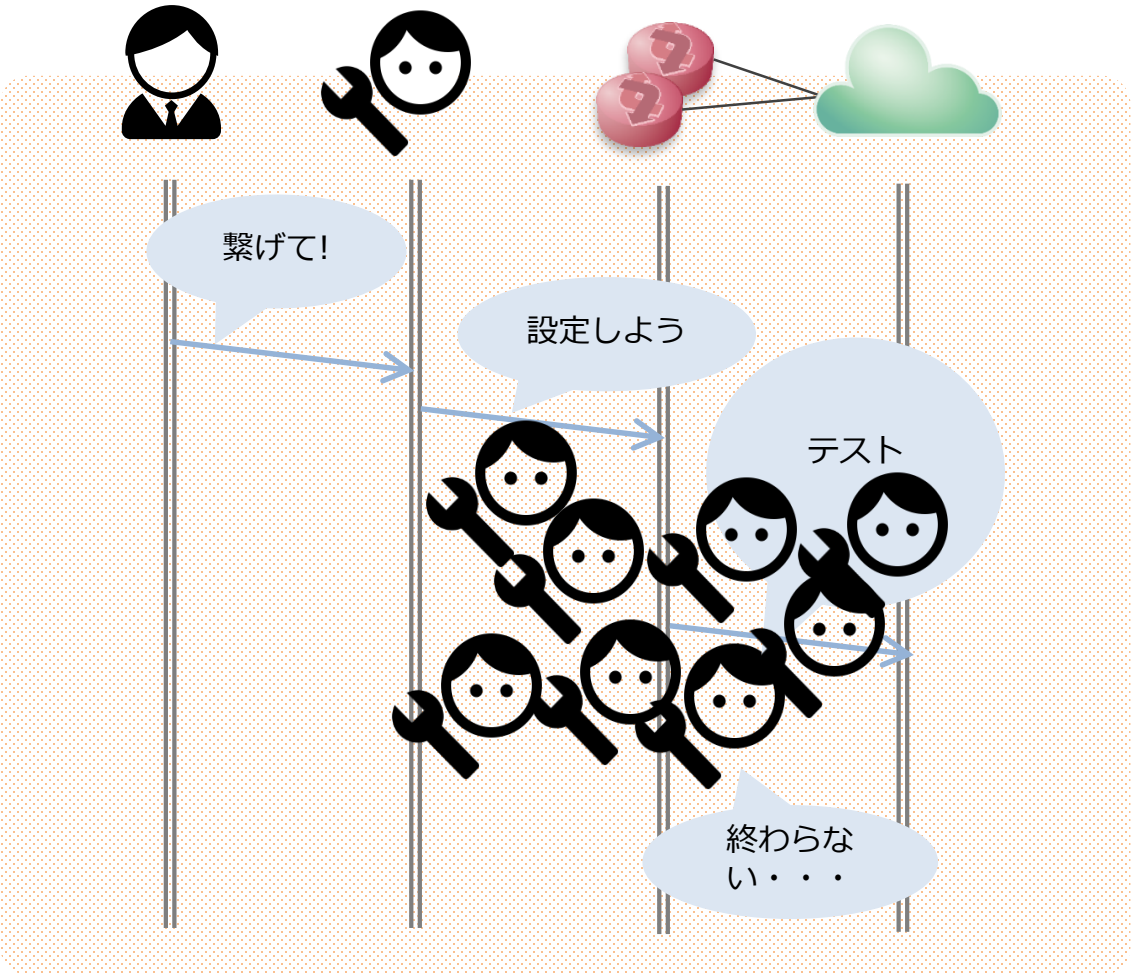
- 物理機器の各種作業が必要
  - 設置場所が遠隔地
  - 物理配線の変更
  - 複数機器にまたがる操作手順
- テスト対象の組み合わせ爆発



これらを解決し
ユースケースによる
テストをしよう
(ソフトウェア開発の知見を拝借)

NWテストにかかるコストの真相

テストのコストは見えにくい



■テストをしなかった場合

- ■暗黙的にユーザが負担
- ■問題発生時に信頼度の低下
- ■作業時のミス対策などのコスト増加

■人海戦術のテストをした場合

- ■カバレッジとテスト時間のバランス
- ■現地作業員のアサイン・出張
- ■テスト失敗したときの再実施コスト大

NWテスト自動化で解消

テスト自動化とこれまでの取り組み

OKINAWA OPEN LABORATORY

済 テスト対象
NW(トポロジ)の操作

物理トポロジのソフトウェアによる操作
障害模擬・物理経路の系切替試験実行

PJ外 テスト対象
NW機器の設定

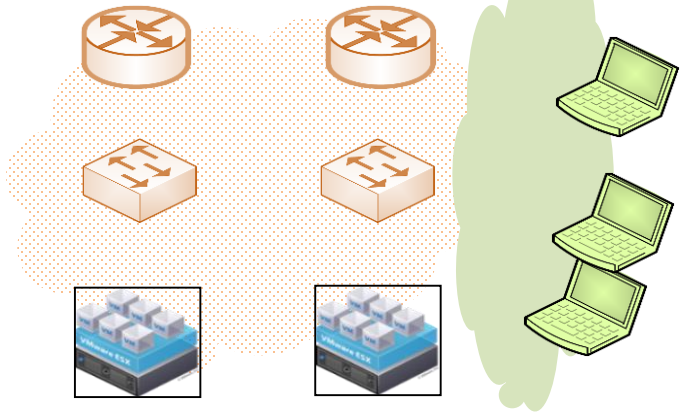
NW機器インタフェース(CLI/REST/NETCONF…)
による機器コンフィグレーション、機器状態取得

PJ外 テスト対象サーバ
(サービス)の設定

テスト対象の物理・論理リソース、
サービスの設定・セットアップ

済 Test
Scenario
(Cucumber)

ユースケースから各種作業へ分解
テスト全体のオーケストレータ



済 テスト用ノードを
対象NWへ接続

物理・論理構成に対するテストパターンの網羅
必要なテストトラフィックを
必要なポイントで入出力させるための仕組み

済 テスト用ノードの
作成・テスト実行

テストのための多様なトラフィック生成、
トラフィックの送受信
テストの結果判定
複数ノードの同時制御(client/server etc)

実際の業務へ適用できる段階

参照

OKINAWA OPEN LABORATORY

■NetTesterリポジトリ

https://github.com/net-tester

- net-tester　：NetTester本体
- scenario-api　：シナリオ実行APIサーバ
- multisite-examples　：多拠点対応シナリオサンプル

■デモ動画

https://youtu.be/DhKutgqYdSw
https://youtu.be/C7z3aaWgsf4

■「nettester」スペースなしで検索

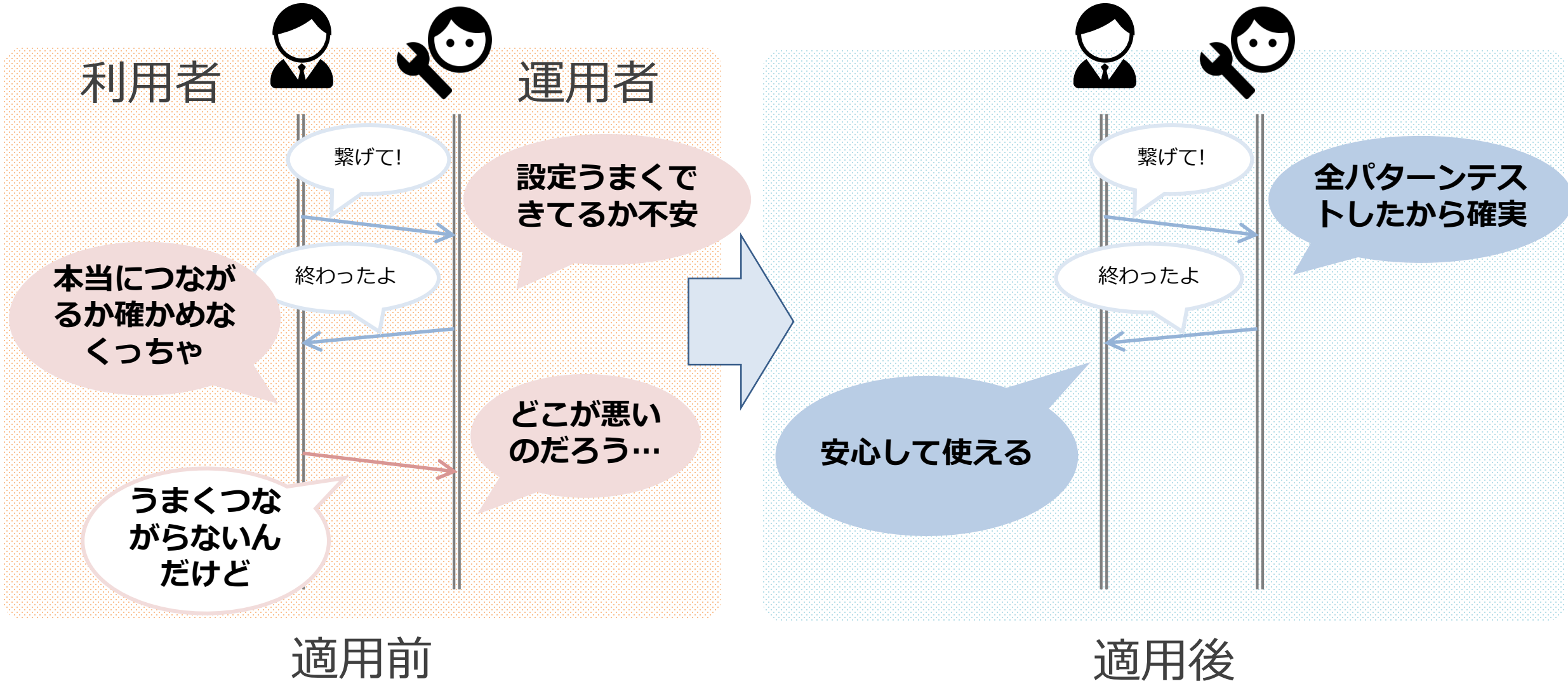
実適用事例

NTTCom様への実適用

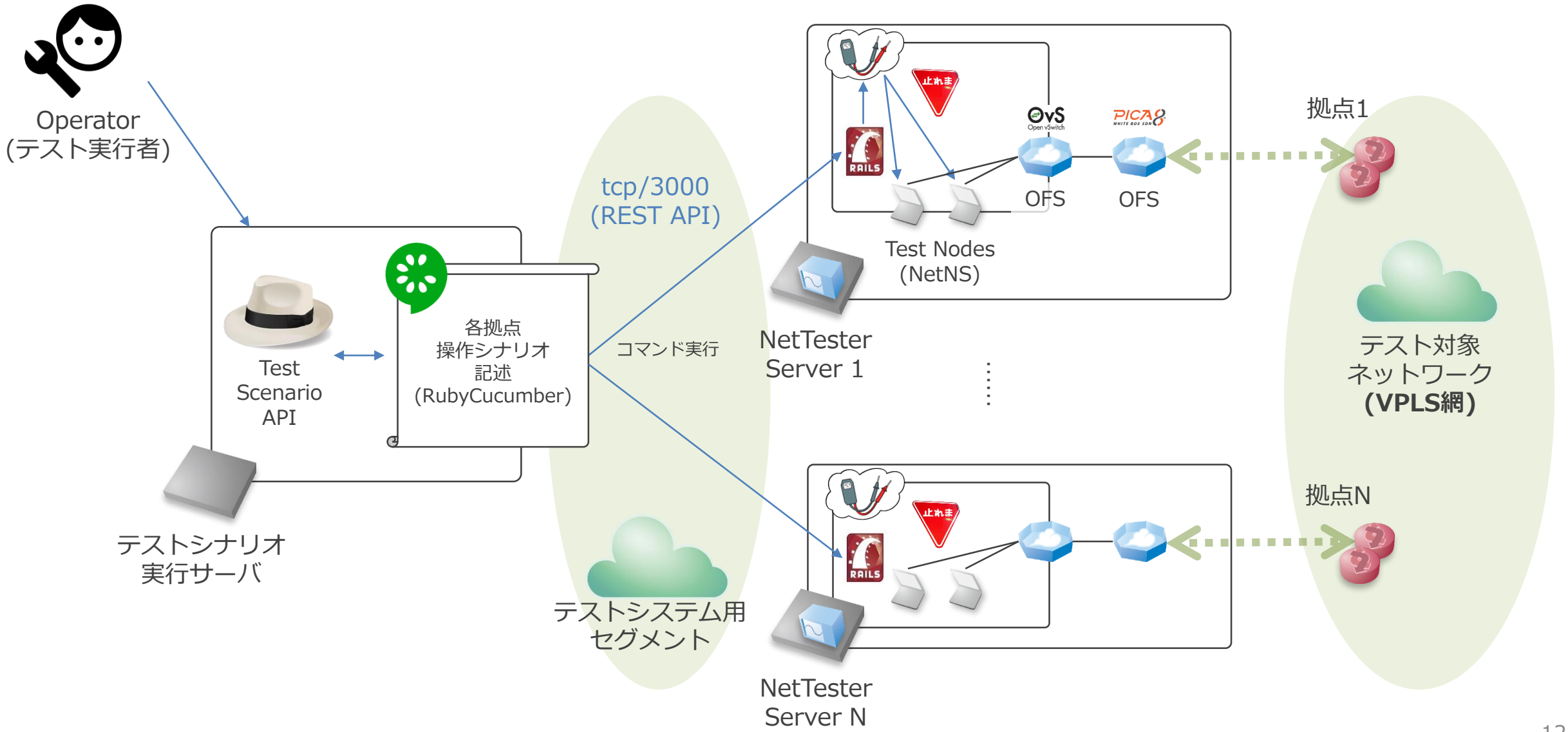
- 昨年の発表で興味を持っていただきました
- 検証用VPLS網の設定変更
  - 複数拠点、冗長化されたルータ(CE)で構成
  - VLAN単位での拠点間L2接続サービス
- 運用が抱える問題点
  - CEに投入する設定が手作業でレビューする余裕がない
  - ユーザからの指摘で設定ミスによる通信不可を発見
  - 運用管理者としては品質を安定したいが、テストが物理的に難しい

NWテスト自動化の導入

実適用の成果


利用者
運用者
繋げて!
設定うまくできてるか不安
本当につながるか確かめなくっちゃ
終わったよ
どこが悪いのだろう…
うまくつながらないんだけど
適用前
繋げて!
全パターンテストしたから確実
終わったよ
安心して使える
適用後

導入したシステム構成

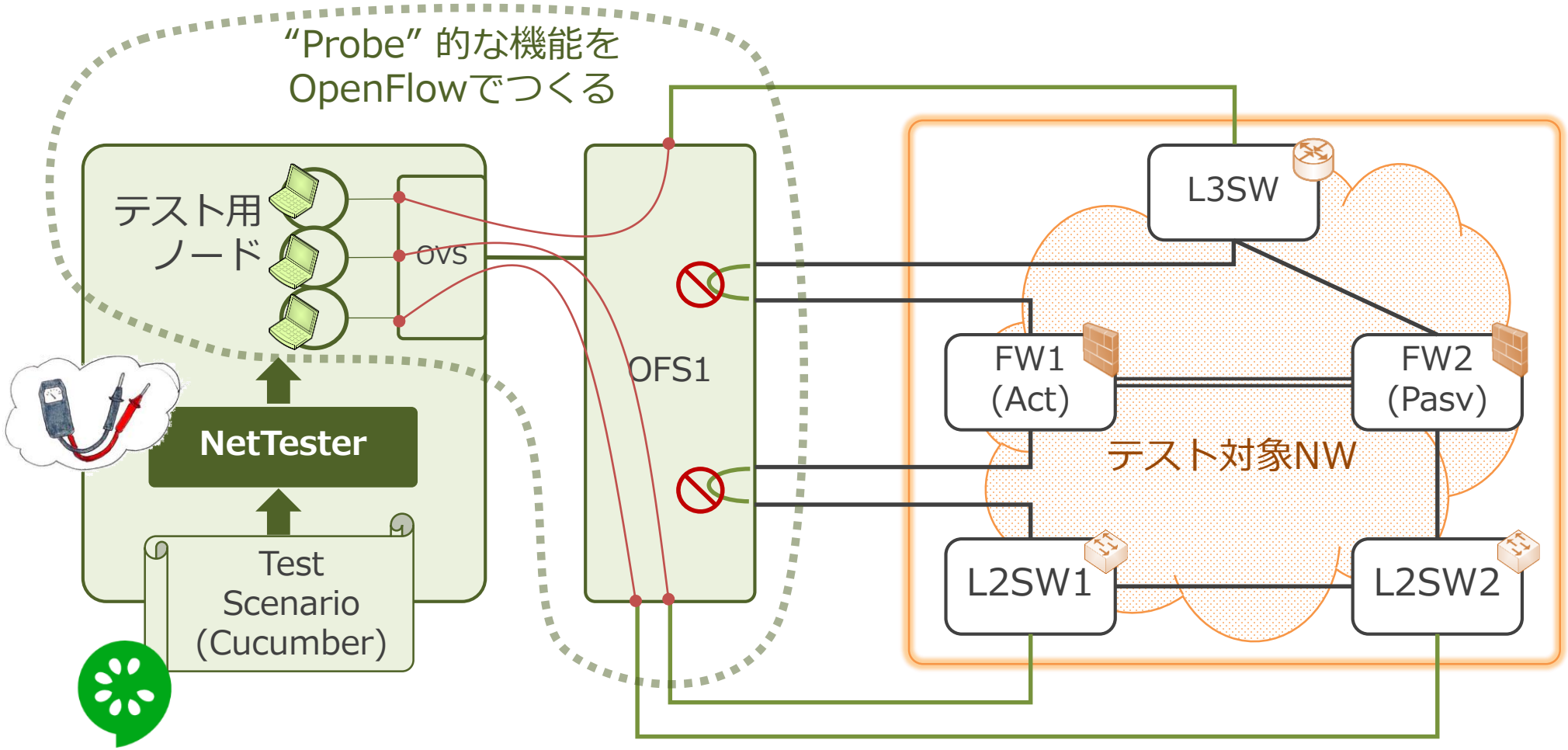

Operator
(テスト実行者)
tcp/3000
(REST API)
Test
Scenario
API
各拠点
操作シナリオ
記述
(RubyCucumber)
コマンド実行
テストシナリオ
実行サーバ
テストシステム用
セグメント
NetTester
Server 1
NetTester
Server N
Test Nodes
(NetNS)
OFS
OFS
拠点1
拠点N
テスト対象
ネットワーク
(VPLS網)

実適用の効果まとめ

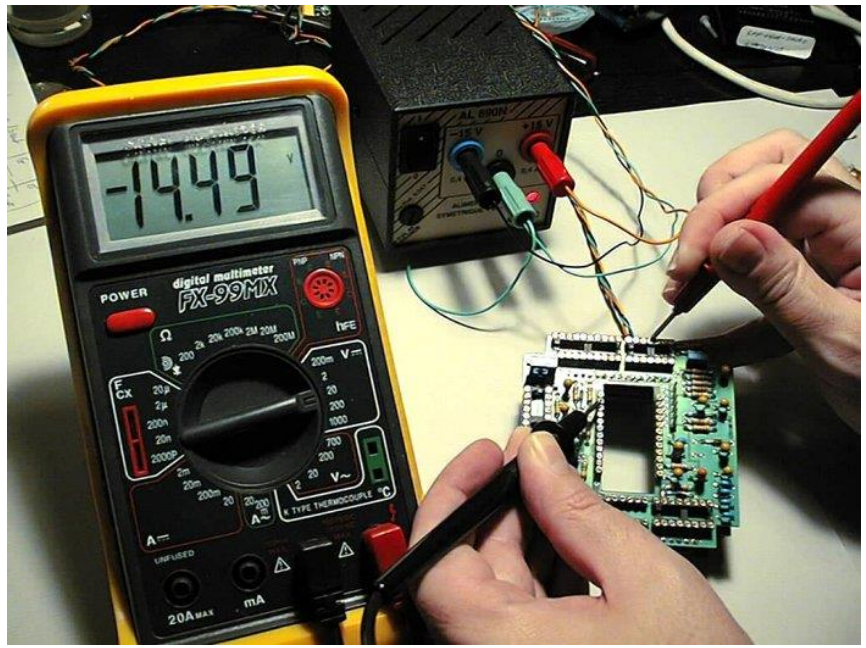
- テストにより設定ミスを発見できた
- 作業者のレベルによらない品質保証を実現
- 変更内容が機械的にチェックされる安心感UP
- 各フェーズが自動化できワークフローの見直しにつながる

補足資料
テスト自動化の仕組み

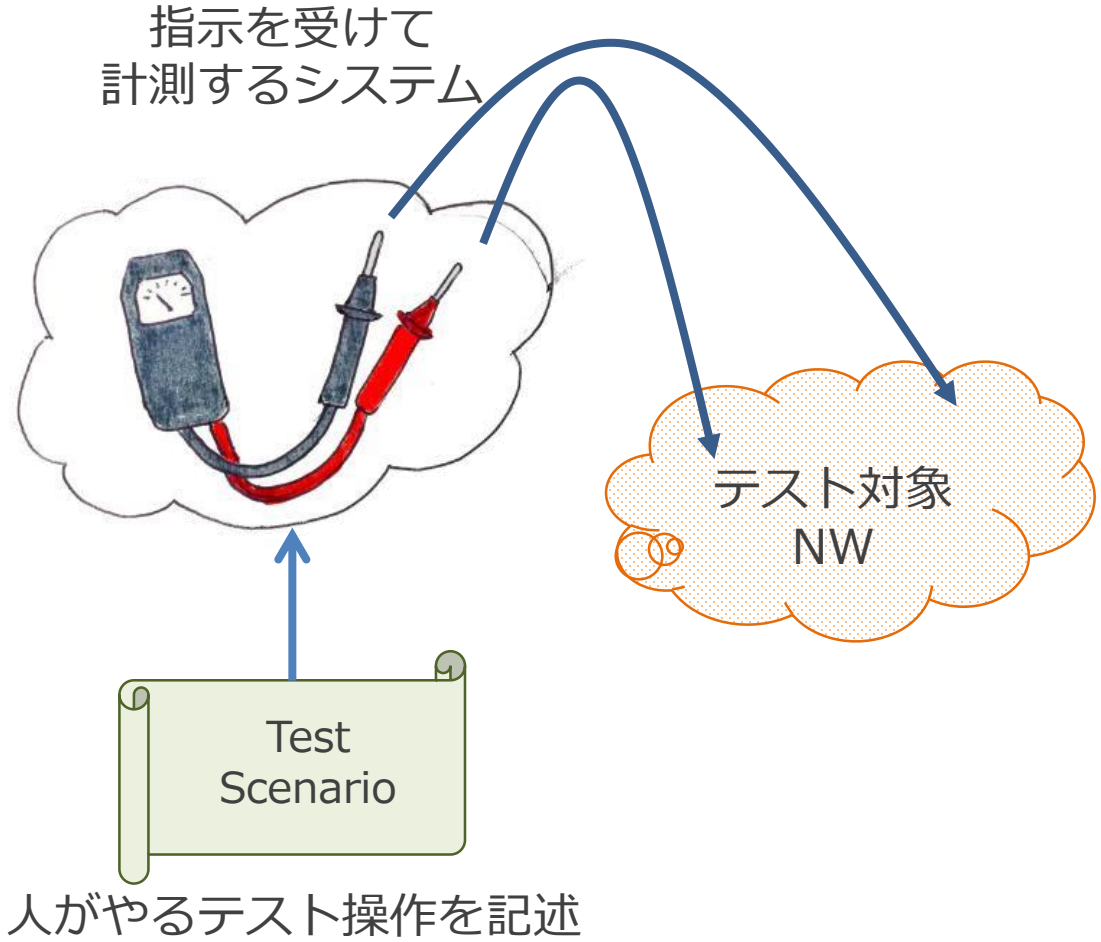
テスト自動化のしくみ(TesterSet)



ふるまいのテスト?



Electronic boards tests
http://www.astrosurf.com/audine/English/cisup2.htm



ふるまい＝テストシナリオ例

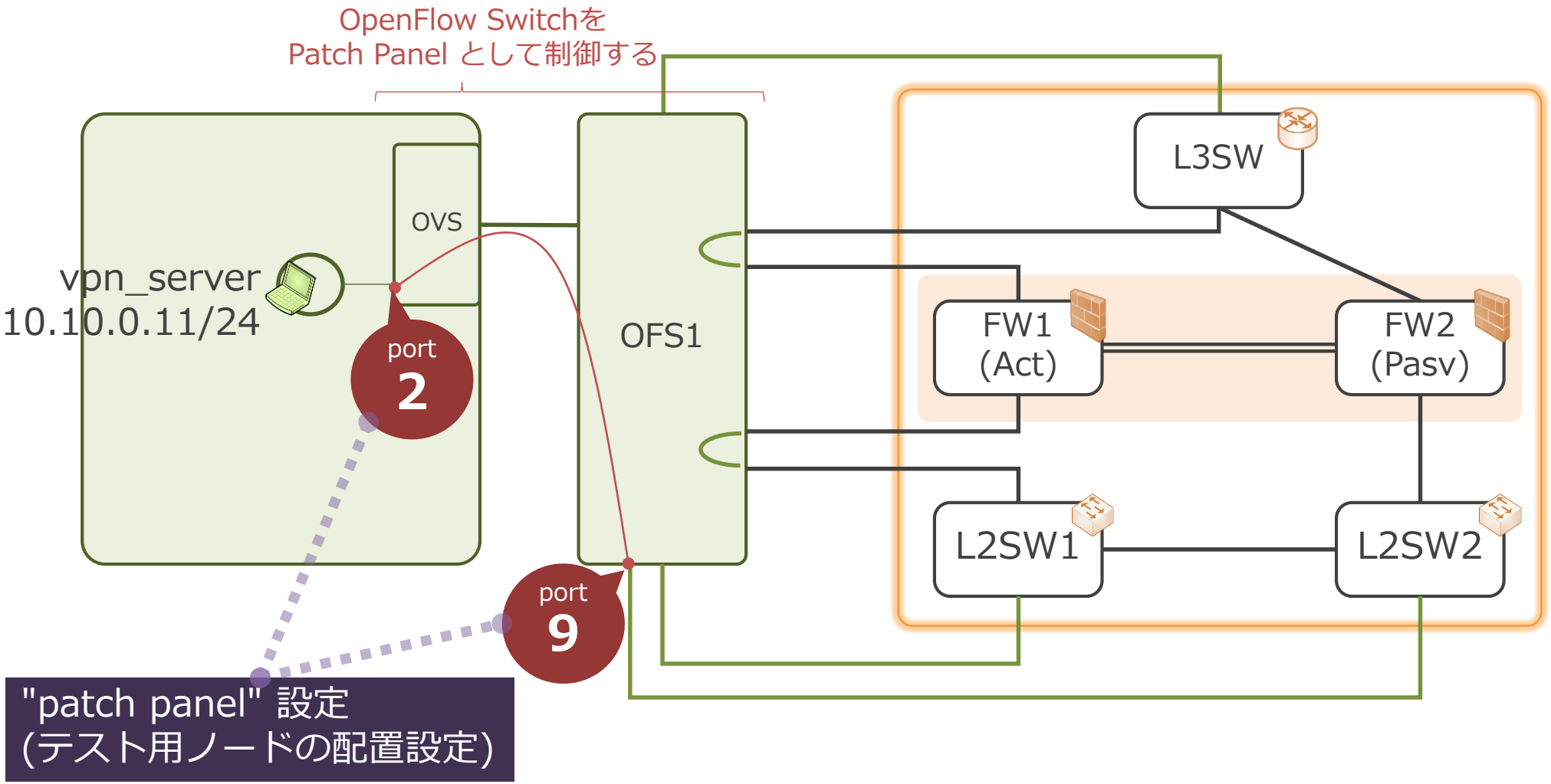
OKINAWA OPEN LABORATORY

```
Feature: リモート開発環境への安定したアクセス
  タジマックス通信工業社の社員として
  ヨーヨーダイン社内の開発環境に安定してアクセスしたい
  なぜならリモート作業を日常的に行うから

  Scenario: リンク障害発生でもリモート接続が切れない
    Given ヨーヨーダイン社のDMZ内部のVPNサーバ
    And タジマックス工業のPCをVPNクライアントに
    And ヨーヨーダイン社のサーバにVPN経由でリモートアクセスして作業
    When “FW1” と “L2SW1” 間にリンク障害が発生
    Then リモート接続が切れていない
```

https://github.com/net-tester/examples/blob/feature/ood_demo/features/tcp_fw1_l2sw1_linkdown.feature

テスト用ノード生成と配置



テスト用ノード生成と配置

```
Given(/^ヨーヨーダイン社のDMZ内部のVPNサーバ$/) do
  @vpn_server = Netns.new(attributes_for(:vpn_server))
end
```

テスト用ノード

https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/virtual_host.rb

```
sequence :virtual_port_number, 2

factory :vpn_server, class: NetTester::Netns do
  name 'vpn_server'
  dmz_network
  ip_address '10.10.0.11'
  physical_port_number 9
  mac_address {Faker::Internet.mac_address('00')}
end
```

"patch panel" 設定
(テスト用ノードの配置設定)

```
trait :dmz_network do
  netmask '255.255.255.0'
  gateway '10.10.0.1'
  virtual_port_number
end
```

テスト用ノードのパラメタ設定

NW(セグメント)のパラメタ設定

https://github.com/net-tester/examples/blob/feature/ood_demo/features/factories.rb

テスト用ノード操作(テスト実行)

```
Given(/^ヨーヨーダイン社のサーバにVPN経由でリモートアクセスして作業$/) do
  step %(ヨーヨーダイン社のDMZ内部のVPNサーバにタジマックス工業のPCからpingを連続実行)
  step %(ヨーヨーダイン社のDMZ内部のVPNサーバにタジマックス工業のPCからTCP接続を開始)
end
```

https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/remotework_linkdown_steps.rb

```
When(/^ヨーヨーダイン社のDMZ内部のVPNサーバにタジマックス工業のPCからTCP接続を開始$/) do
  cd('.') do
    @echo_server = AsyncExecutor.new(host: @vpn_server, result_file: 'log/tcp_server.log')
    @echo_server.exec("../../features/support/echo_server.pl 80")
    @echo_client = AsyncExecutor.new(host: @tajimax_pc, result_file: 'log/tcp_a.log')
    @echo_client.exec("../../features/support/echo_client.pl 203.0.113.5 80 30")
  end
end
```

tcp echo server/client をテスト用ノード上で実行してログを取得("tcp ping")

https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/continuous_tcp_steps.rb

テスト用ノード操作(テスト結果判定)

```
Then(/^リモート接続が切れていない$/) do
  step %(ヨーヨーダイン社のDMZ内部のVPNサーバにタジマックス工業のPCからのpingによる疎通が 10 秒以内に復帰)
  step %(ヨーヨーダイン社のDMZ内部のVPNサーバにタジマックス工業のPCからのTCP接続が維持されている)
  step %(FWの主系が Passive 、予備系が Active になっていること)
end
```

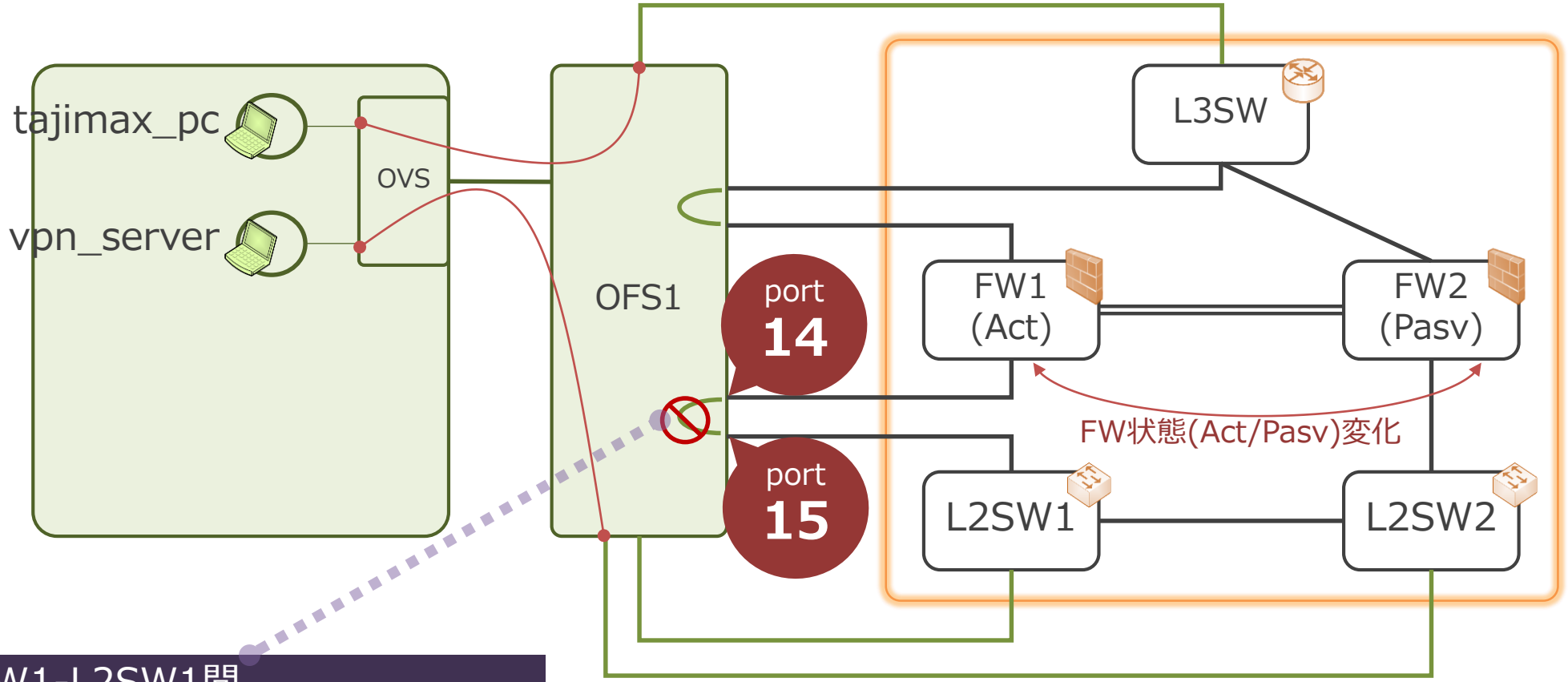
https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/remotework_linkdown_steps.rb

```
Then(/^ヨーヨーダイン社のDMZ内部のVPNサーバにタジマックス工業のPCからのTCP接続が維持されている$/) do
  @echo_client.join
  cd('.') do
    line_count, _ = check_connection('log/tcp_a.log')
    expect(line_count).to be == 30
  end
end
```

ログから、一定時間以上の通信切断が発生しなかったことを確認。

https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/continuous_tcp_steps.rb

テスト対象NWの物理構成操作



FW1-L2SW1間
リンクダウン/リンクアップ操作

テスト対象NWの物理構成操作

```
When(/^“FW1” と “L2SW1” 間にリンク障害が発生$/) do
  step %(10 秒待つ)
  step %(FW1-L2SW1間リンク障害発生)
end
```

https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/remotework_linkdown_steps.rb

```
When(/^FW1-L2SW1間リンク障害発生$/) do
  cd('.') do
    make_port_down(14)
    make_port_down(15)
  end
end
```

https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/fw_fault_steps.rb

OpenFlow Switch (Pica8)にログインしてポートダウン コマンドを実行

```
def make_port_down(port)
  thrower = Expectacle::Thrower.new(base_dir: __dir__ + '/../support/expectacle', logger: :syslog, verbose: false)
  pica8_hosts = YAML.load_file("#{thrower.hosts_dir}/pica8_hosts.yml")
  pica8_commands = YAML.load_file("#{thrower.commands_dir}/pica8_port_#{port}_down.yml")
  thrower.run_command_for_all_hosts(pica8_hosts, pica8_commands)
end
```

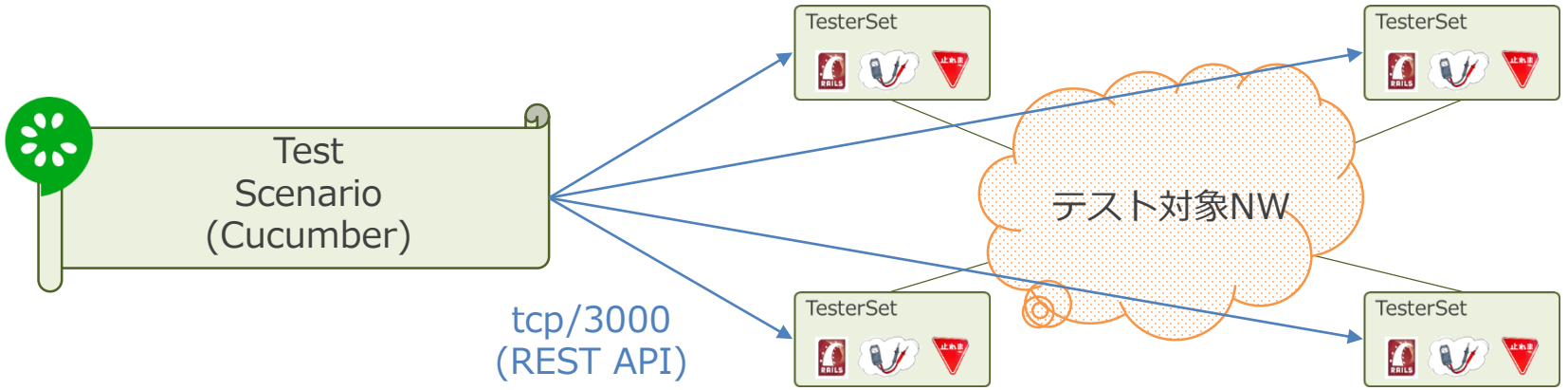
https://github.com/net-tester/examples/blob/feature/ood_demo/features/step_definitions/util.rb

```
- "ovs-ofctl mod-port br0 14 down"
```

https://github.com/net-tester/examples/blob/feature/ood_demo/features/support/expectacle/commands/pica8_port_14_down.yml

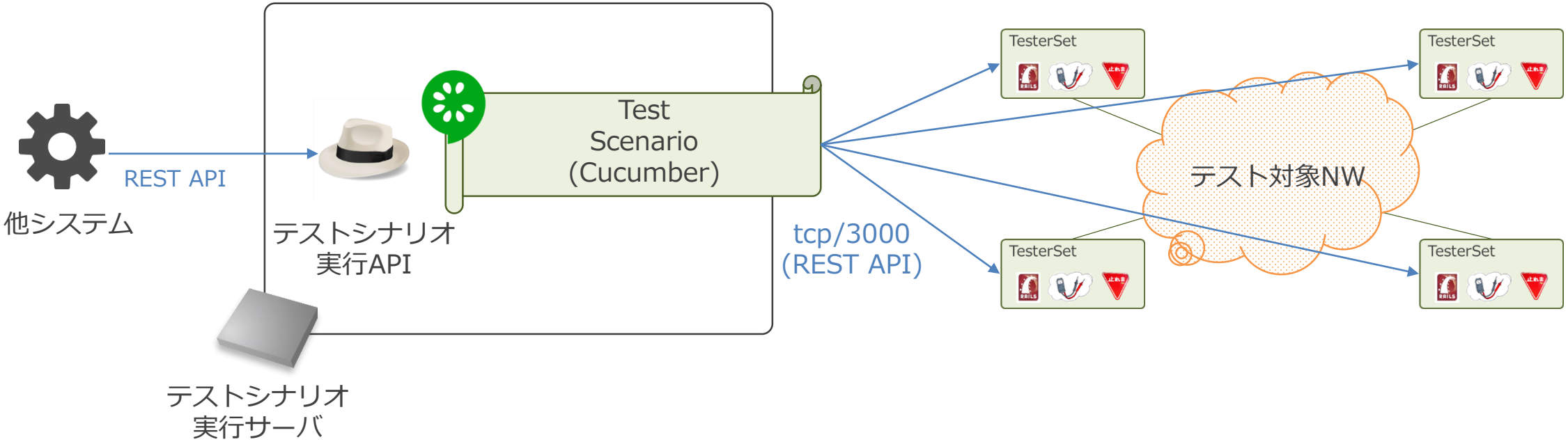
多拠点展開

- 多拠点間テスト対応時のコンポーネント疎結合化と役割分担
  - テストシナリオと環境制御機能の分離
    - 各拠点を担当するテストシステムの単位 “TesterSet (NetTester+OFS)”を定義
    - テストの主体＝テストシナリオが全体のテスト状況を制御する
      - ✓ 必要に応じTesterSetに制御を指示
    - NetTesterは設計前提から部品であり具体的なテストと実行内容を知る必要はない
      - ✓ NetTesterはステートレスな単発処理(ホスト起動、繋ぎ込み、コマンド実行)に専念
      - ✓ 単拠点の機能を遠隔から要求に応じて実行可能にする形へ
    - SSHでの指示は鍵管理が複雑 → RESTで指示を定義

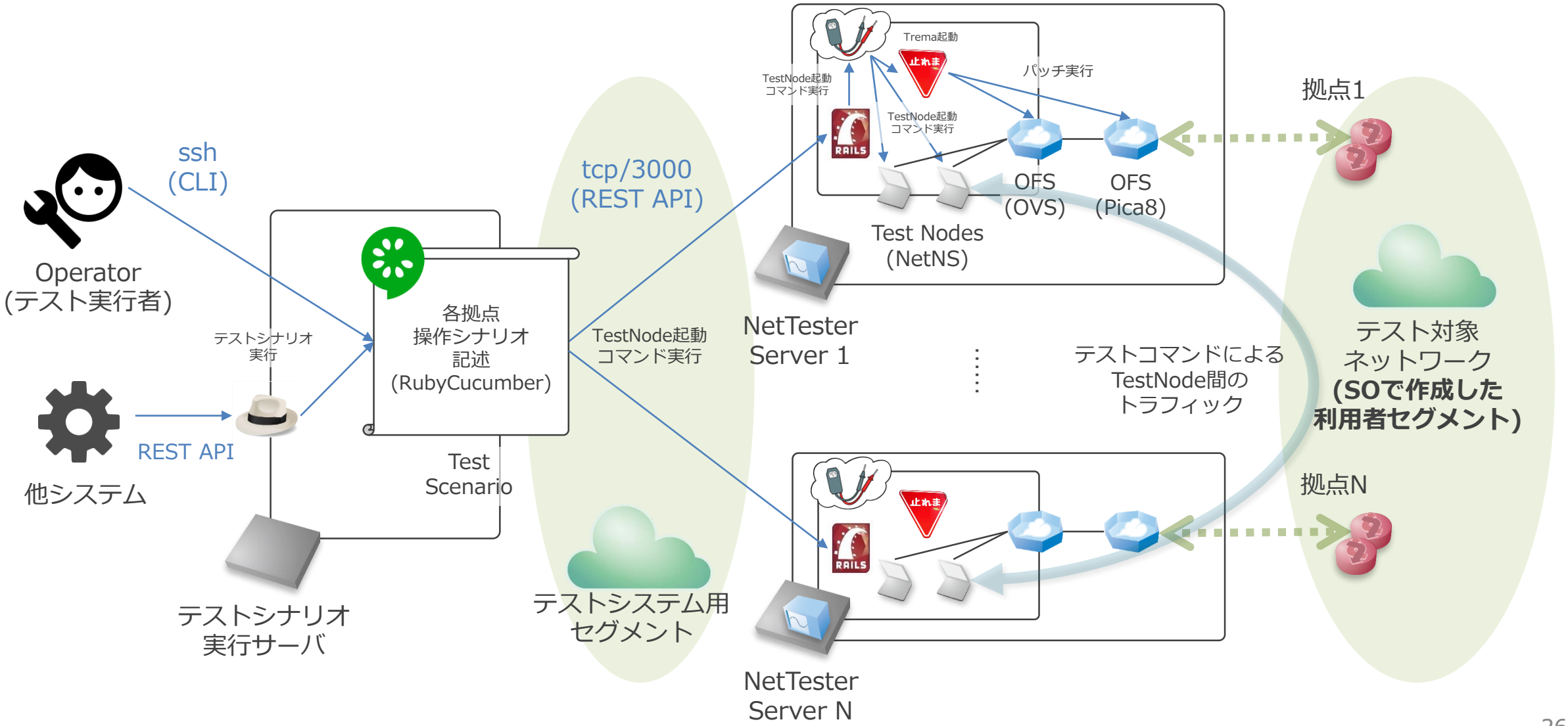


他システムとの連携

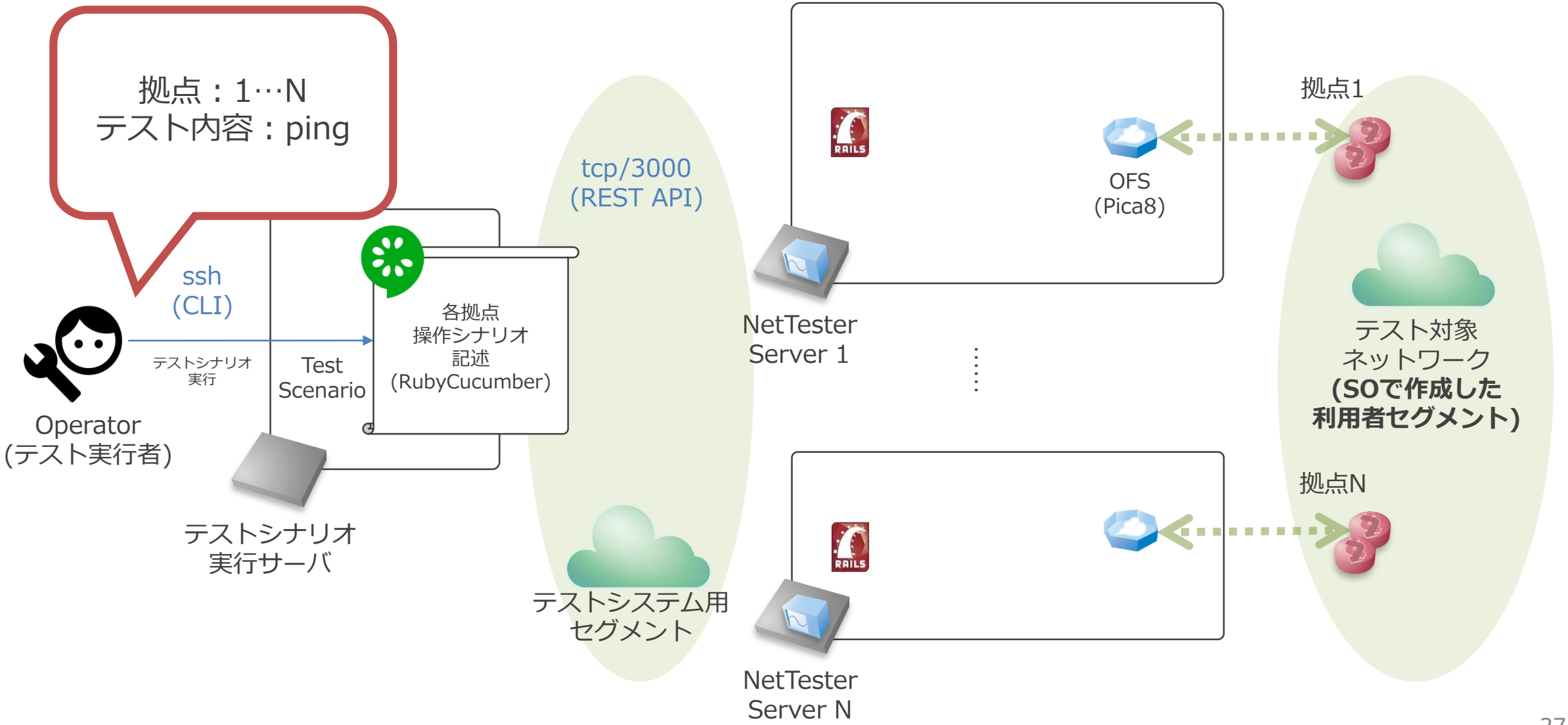
- NorthBoundとしてのテストシナリオ実行のサービス化
  - テストシナリオを実行するAPI層を提供
  - テスト自体を提供するサービスとして疎結合化



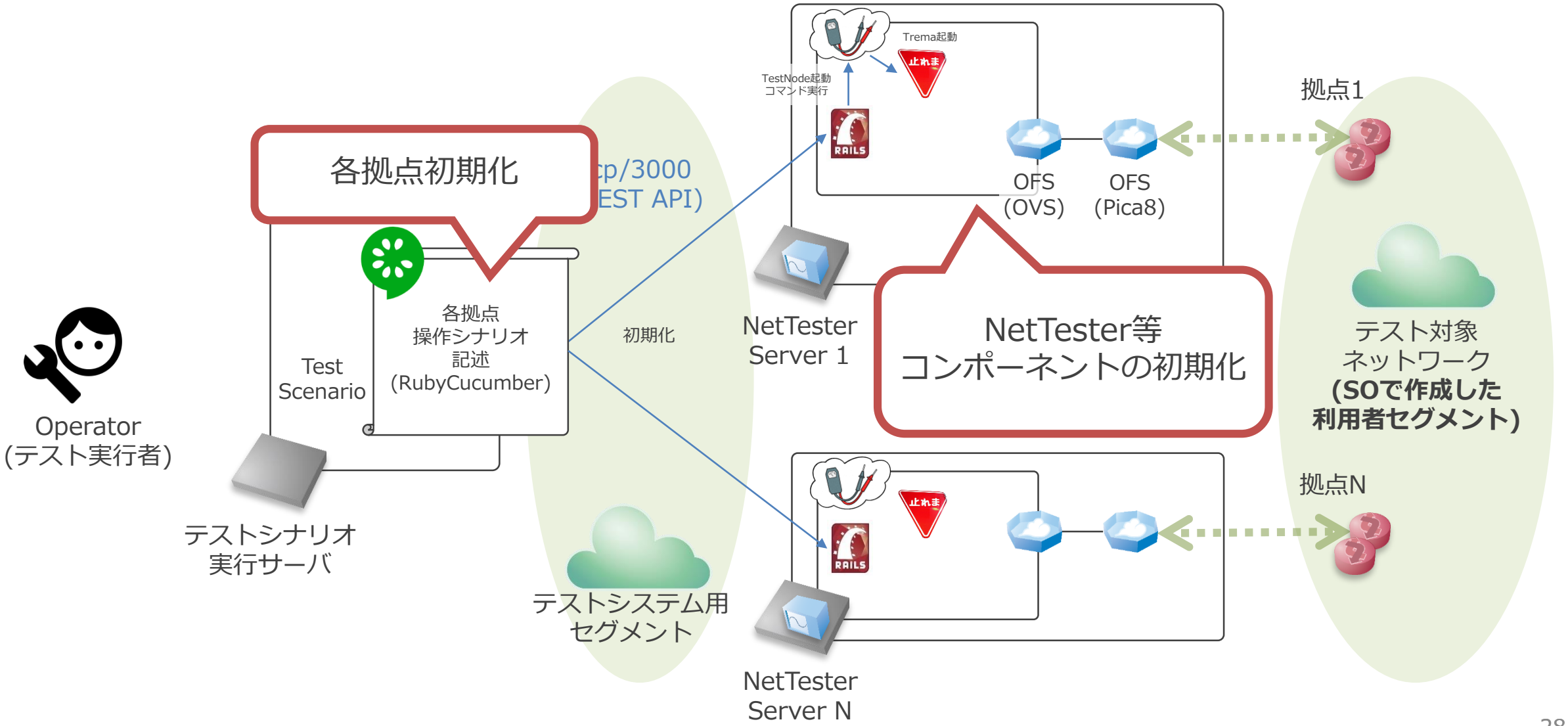
動作の全体像


ssh
(CLI)
Operator
(テスト実行者)
テストシナリオ
実行
REST API
他システム
各拠点
操作シナリオ
記述
(RubyCucumber)
Test
Scenario
テストシナリオ
実行サーバ
tcp/3000
(REST API)
TestNode起動
コマンド実行
テストシステム用
セグメント
Trema起動
TestNode起動
コマンド実行
パッチ実行
TestNode起動
コマンド実行
OFS
(OVS)
OFS
(Pica8)
Test Nodes
(NetNS)
NetTester
Server 1
NetTester
Server N
テストコマンドによる
TestNode間の
トラフィック
拠点1
拠点N
テスト対象
ネットワーク
(SOで作成した
利用者セグメント)

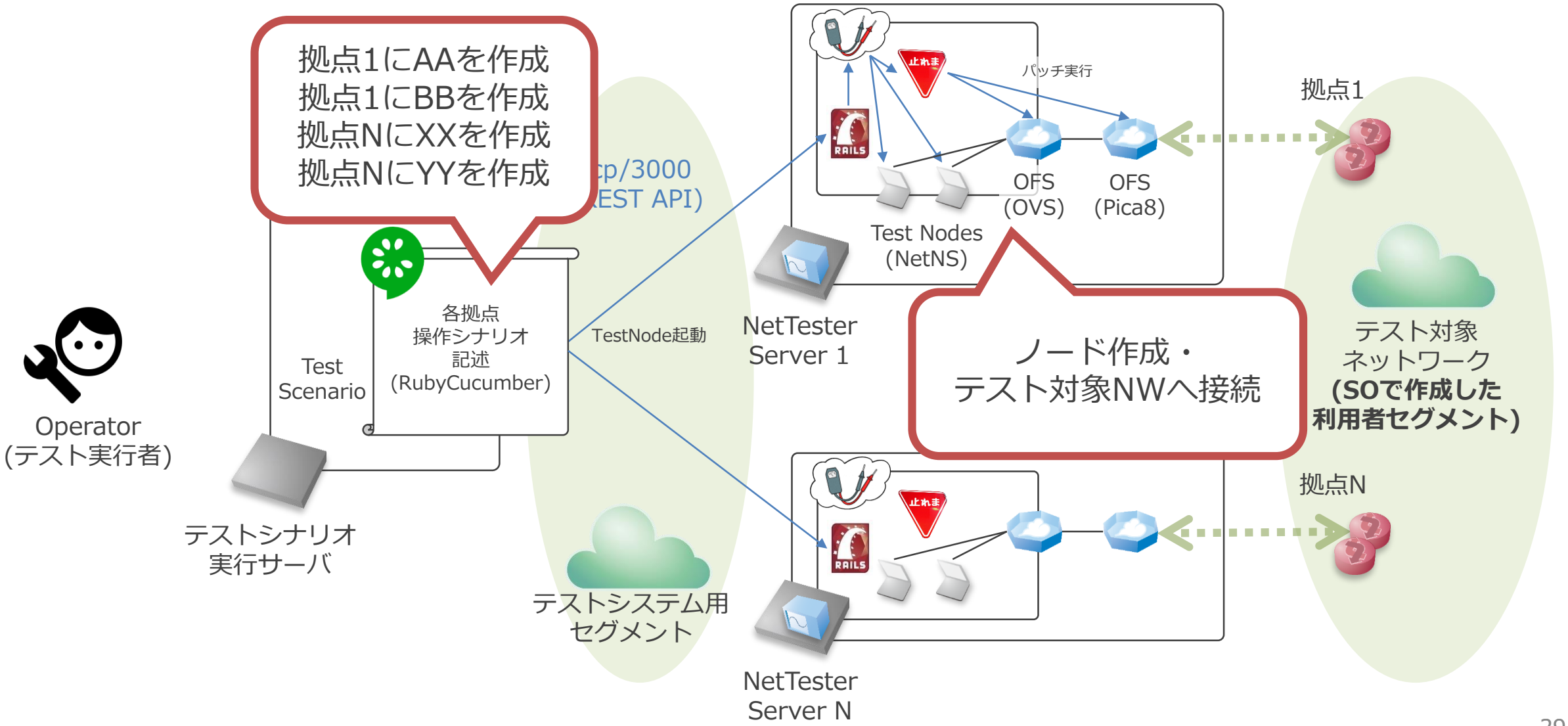
1.テストシナリオ実行要求



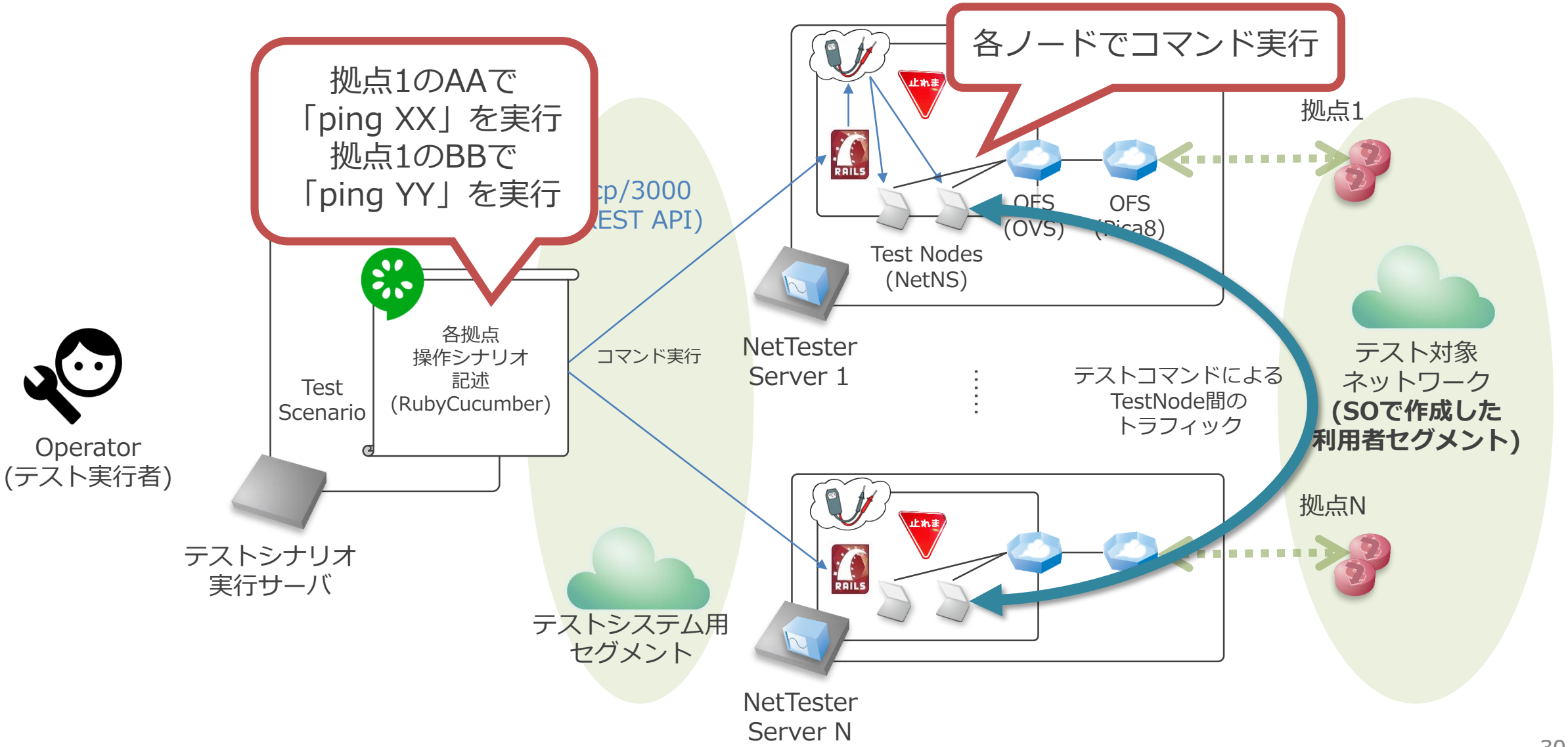
2.シナリオに従い、各拠点を初期化



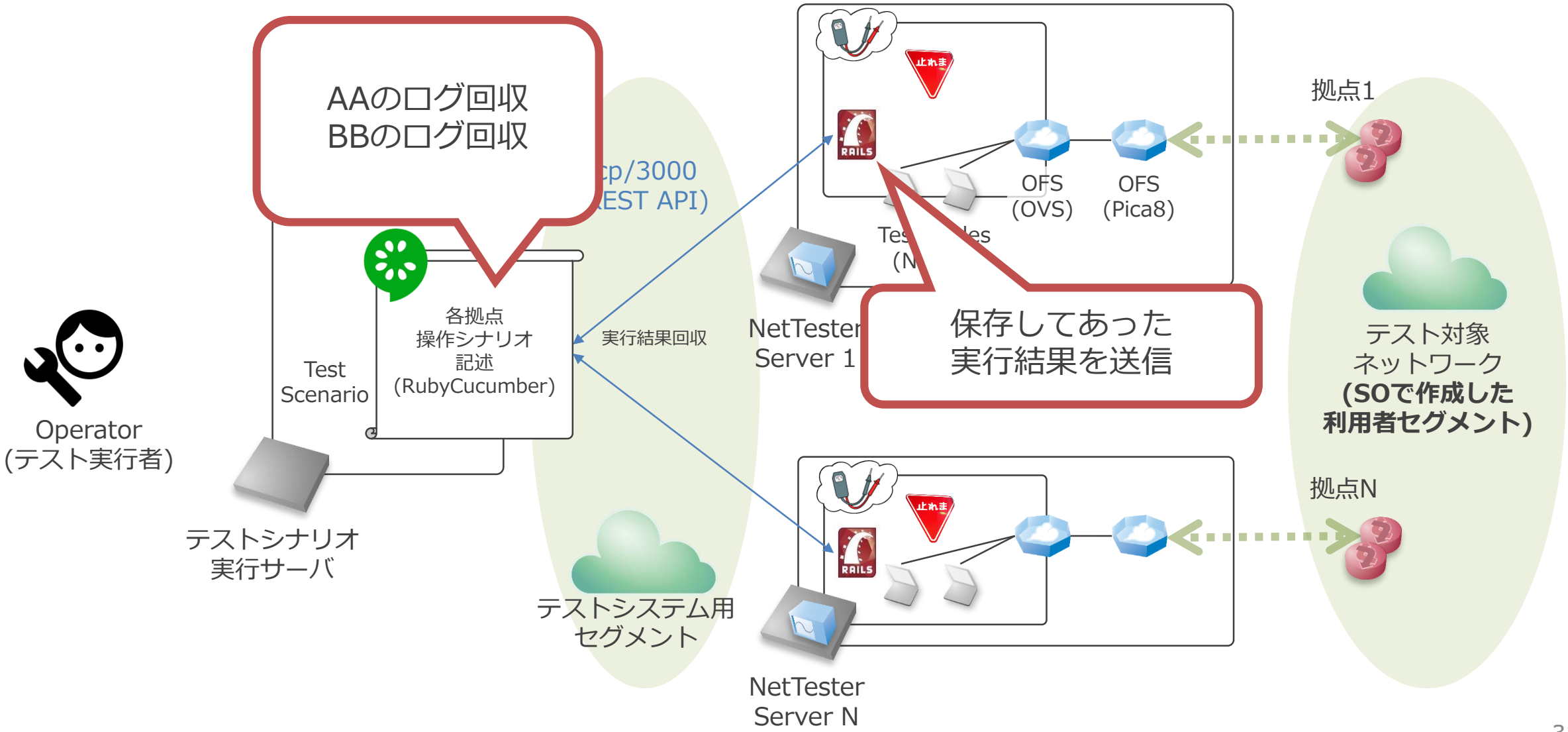
3.各拠点にノード作成


拠点1にAAを作成
拠点1にBBを作成
拠点NにXXを作成
拠点NにYYを作成
cp/3000
EST API)
Operator
(テスト実行者)
Test
Scenario
各拠点
操作シナリオ
記述
(RubyCucumber)
テストシナリオ
実行サーバ
TestNode起動
テストシステム用
セグメント
NetTester
Server 1
NetTester
Server N
止まれ
RAILS
パッチ実行
OFS
(OVS)
OFS
(Pica8)
Test Nodes
(NetNS)
ノード作成・
テスト対象NWへ接続
拠点1
拠点N
テスト対象
ネットワーク
(SOで作成した
利用者セグメント)

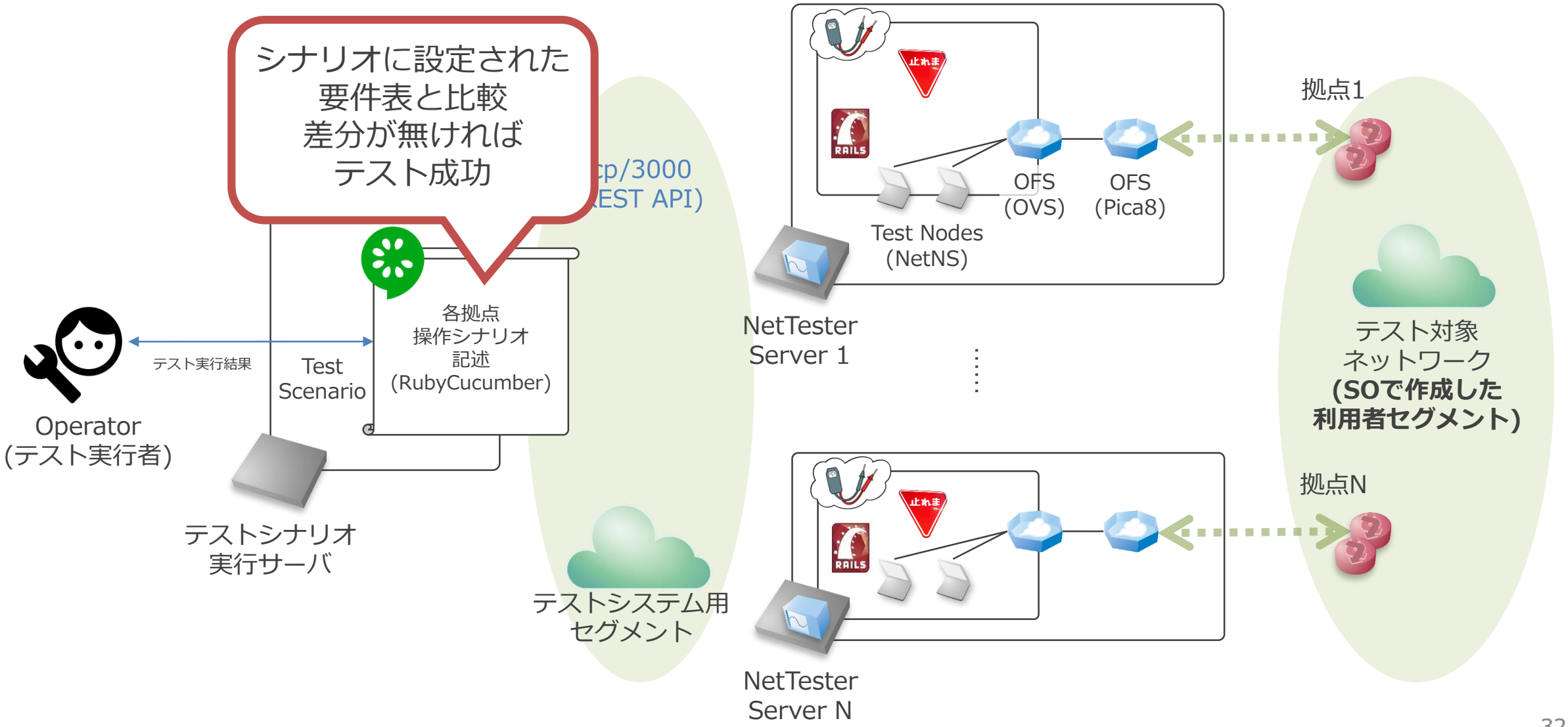
4.各拠点でコマンド実行



5.各拠点からコマンド実行結果を回収



6.シナリオと比較し、テスト成否を判断



7.テスト結果をもとに続行/中止を判断