コアネットワーク仮想化導入開始から10年。
完全仮想化完了までの歩み

NTT ドコモ
安藤寛史、宇田川雄貴

はじめに

- ドコモでは2016年にOpenStackベースの基盤へコアネットワークアプリケーションを導入し、2026年に全アプリケーションの完全仮想化を完了
- 通信事業として、通常運用時のみならずシステム障害/ハードウェア更改/ソフトウェア更改時でさえユーザ影響を出す事は許されず、その要件を満たすために技術的課題への挑戦
- 本報告では、ETSI ISG NFVにおけるVIM/NFVIに関する開発・運用のノウハウを共有

報道発表資料

Tweet

モバイルコアネットワークの完全仮想化を完了
-さらなるネットワーク信頼性の向上を実現-
<2026年4月2日>

株式会社NTTドコモ

株式会社NTTドコモ（以下、ドコモ）は、9,000万を超えるお客さまにご利用いただく、モバイル音声サービスおよびモバイルデータ通信サービスの提供基盤であるモバイルコアネットワーク（以下、コアネットワーク）について、2026年3月末までに実施したネットワーク設備の切り替えおよび第3世代移動通信方式（3G）のサービス終了に伴い、完全仮想化を完了※1いたしました。
これにより、ドコモのコアネットワークは、従来の専用ハードウェアに依存した構成から、汎用的なサーバ上でソフトウェアとして機能を実装する構成へと移行し、より柔軟で安定性が高く、かつ省コストの通信基盤へと進化します。

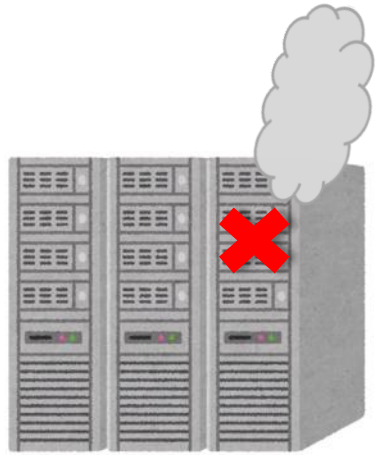
本日の結論

■ 仮想化の成否は「2つの運用設計」で決まる。ポイントは「やりすぎないこと」

運用設計の中で重要だったのは2つ

ライフサイクル（更改）に関する設計

障害発生時の設計



ドコモのネットワーク仮想化基盤の規模

Data Centers **34**

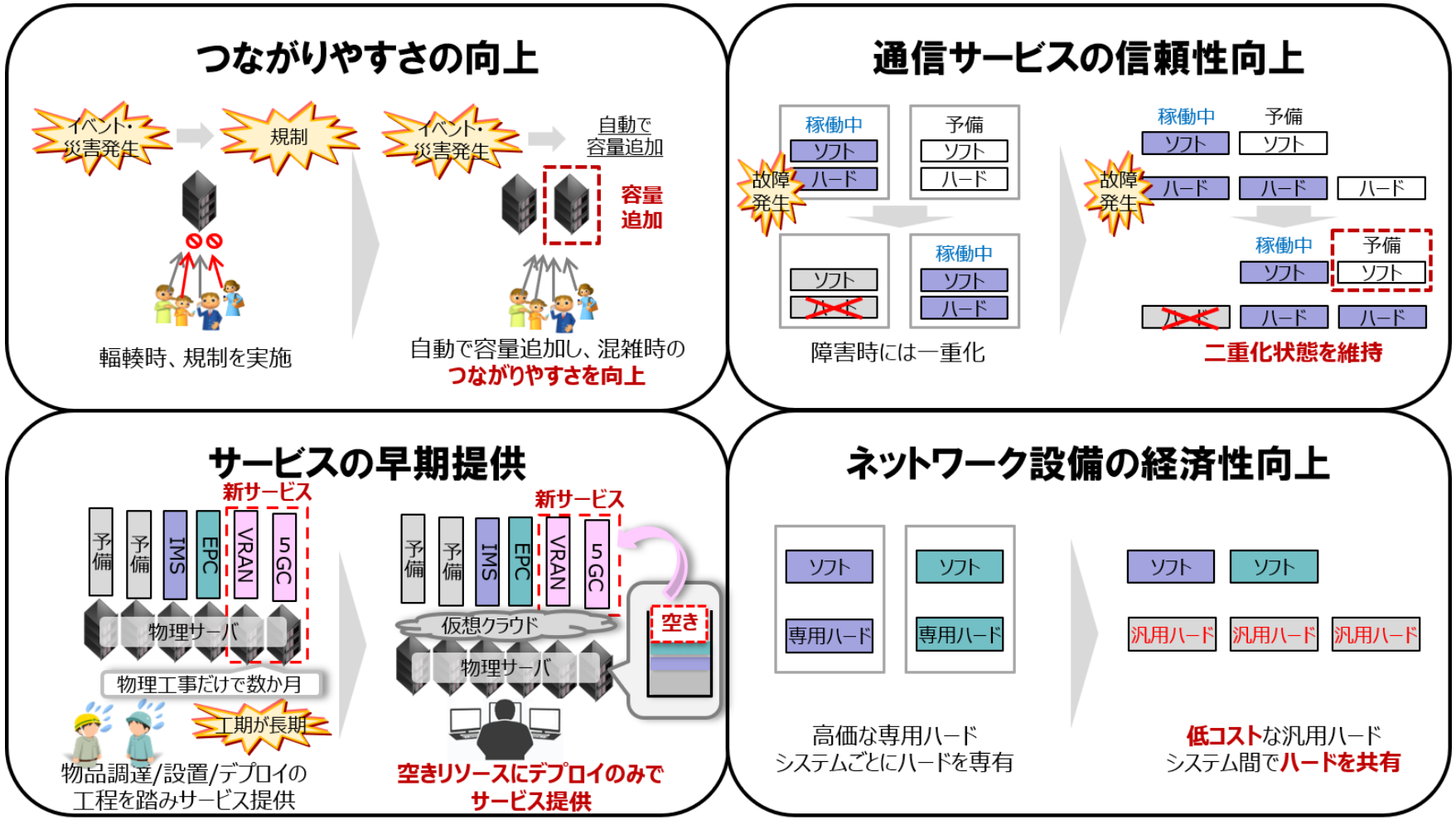
Virtual Network Functions **70+**

Virtual Machines **425,000+**

Servers **14000+**

そもそも、なぜドコモはネットワーク仮想化基盤を導入したのか

- ドコモの使命は、**いかなる時も通信サービスを中断させない**こと
- 仮想化の導入により、通信品質・信頼性のさらなる向上とコストの効率化を目指した



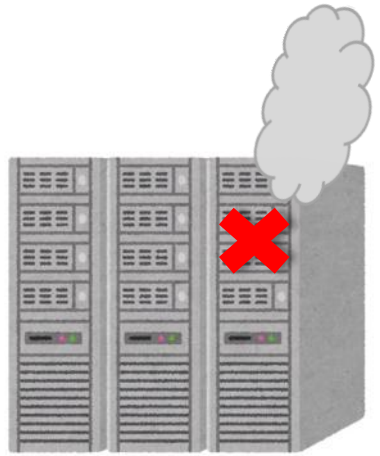
通信品質の向上とコストの効率化を実現するための設計検討

■ 仮想化基盤を導入した後も通信サービスを中断させないために
「ライフサイクル（更改）」と「障害発生時の設計」を重要課題として導入の検討を実施

仮想化基盤導入後もサービス無中断を実現するための重要課題

ライフサイクル（更改）に関する設計

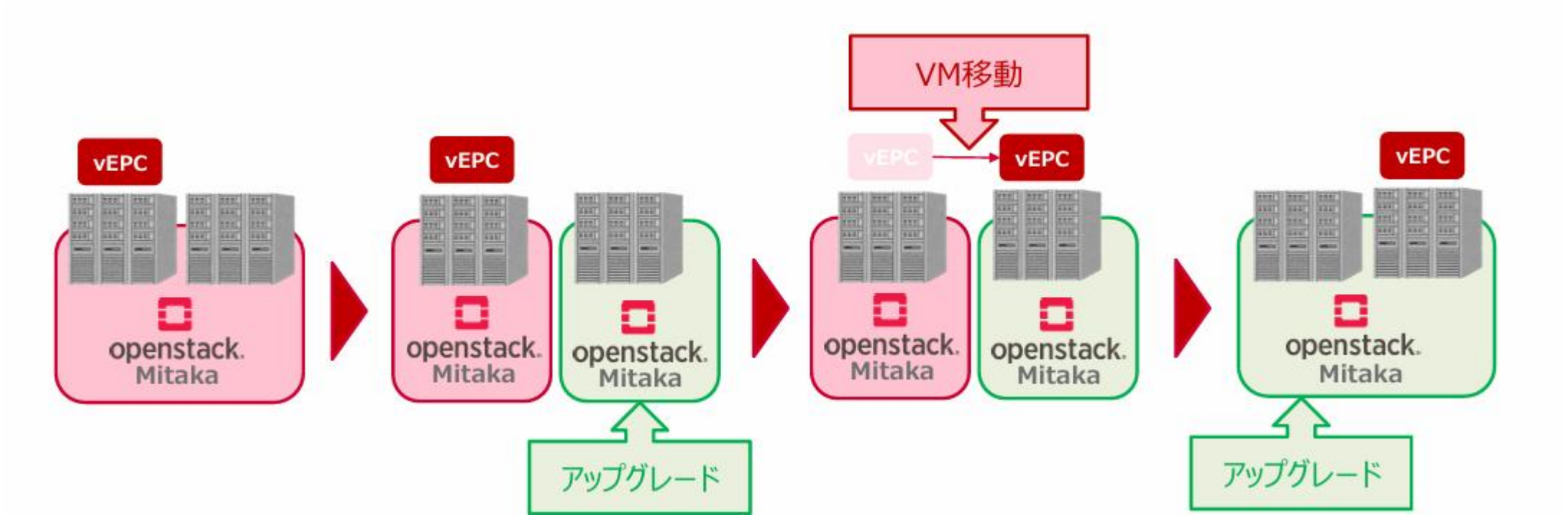
障害発生時の設計



実際に運用してみたら設計者と運用者で**ギャップ**があることが発覚

本題に入る前に前提条件の共有

■ ネットワーク仮想化基盤のアップデート方式は複数存在する中、
「最小限の設備で更改」、「サービスの無中断」を実現するため　ローリングアップデートを採用

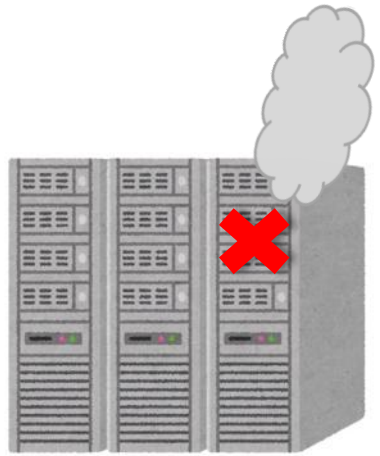


通信品質の向上とコストの効率化を実現するための設計検討

■ ライフサイクル（更改）に関する設計と運用のギャップについて説明します

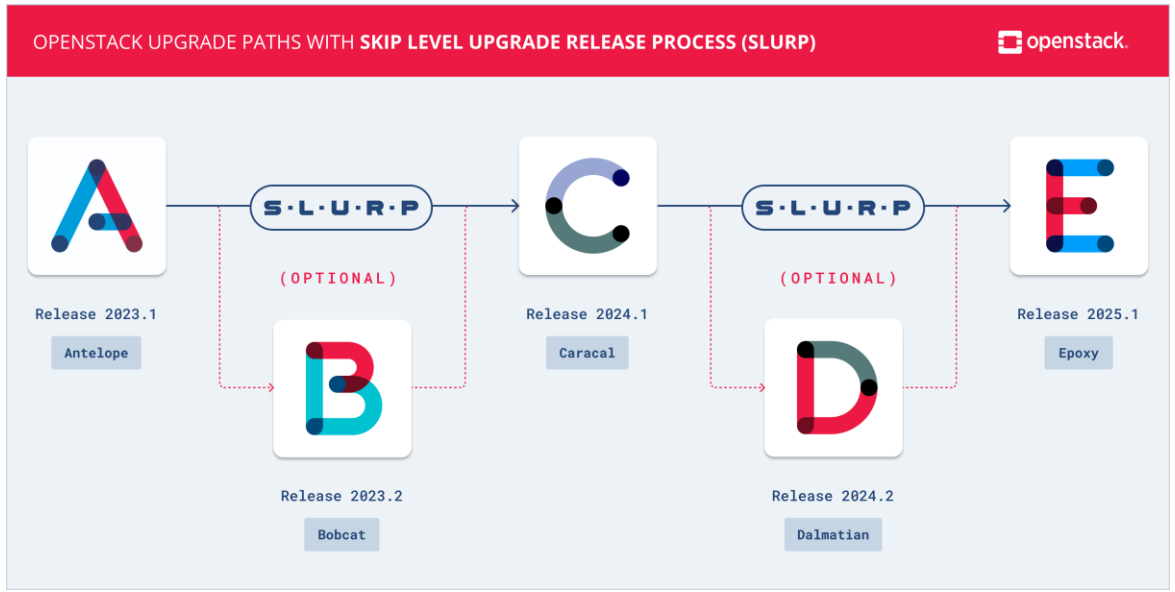
ライフサイクル（更改）
に関する設計

障害発生時の設計

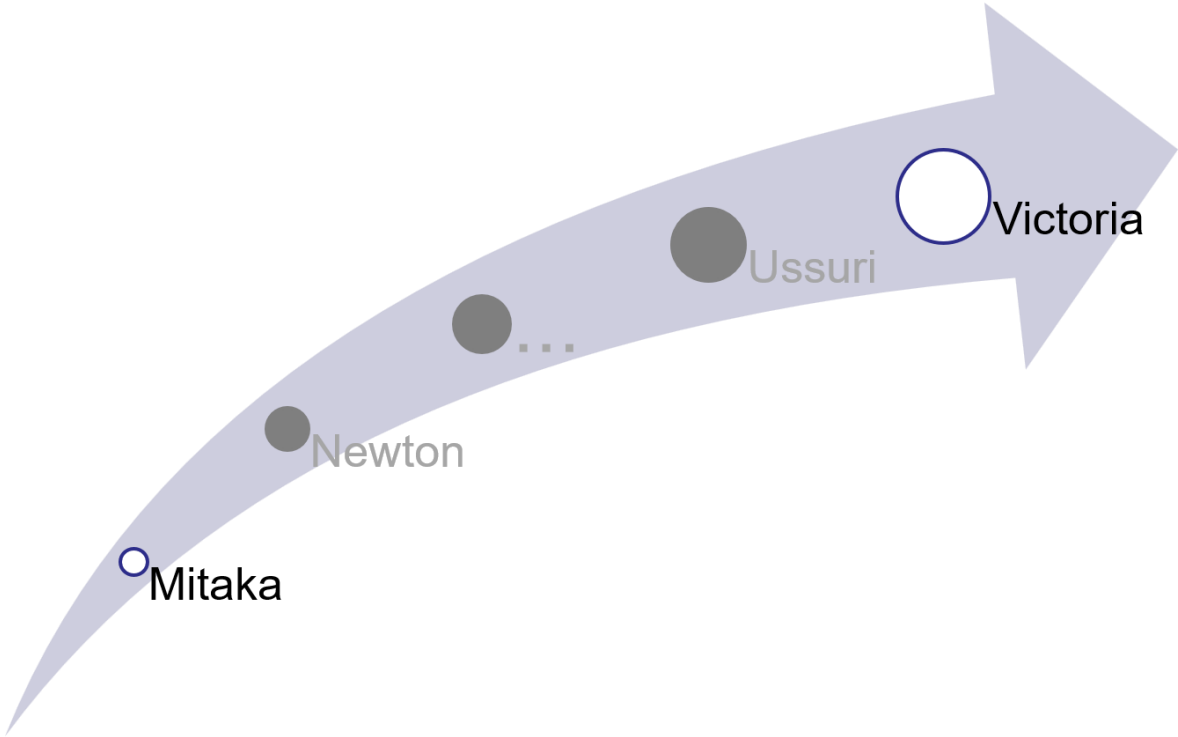


設計と運用のギャップ−OpenStackのジャンピングアップデート−

理想はN+1 or N+2のアップデート



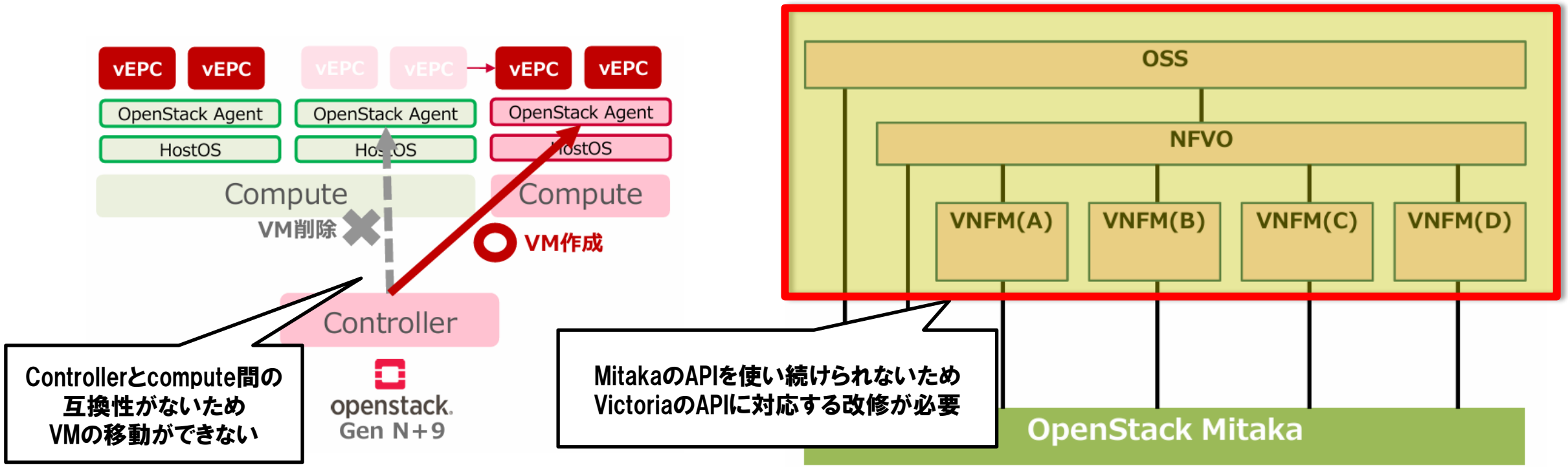
現実はイレギュラーの9世代のジャンピング



設計と運用のギャップ−OpenStackのジャンピングアップデートの影響−

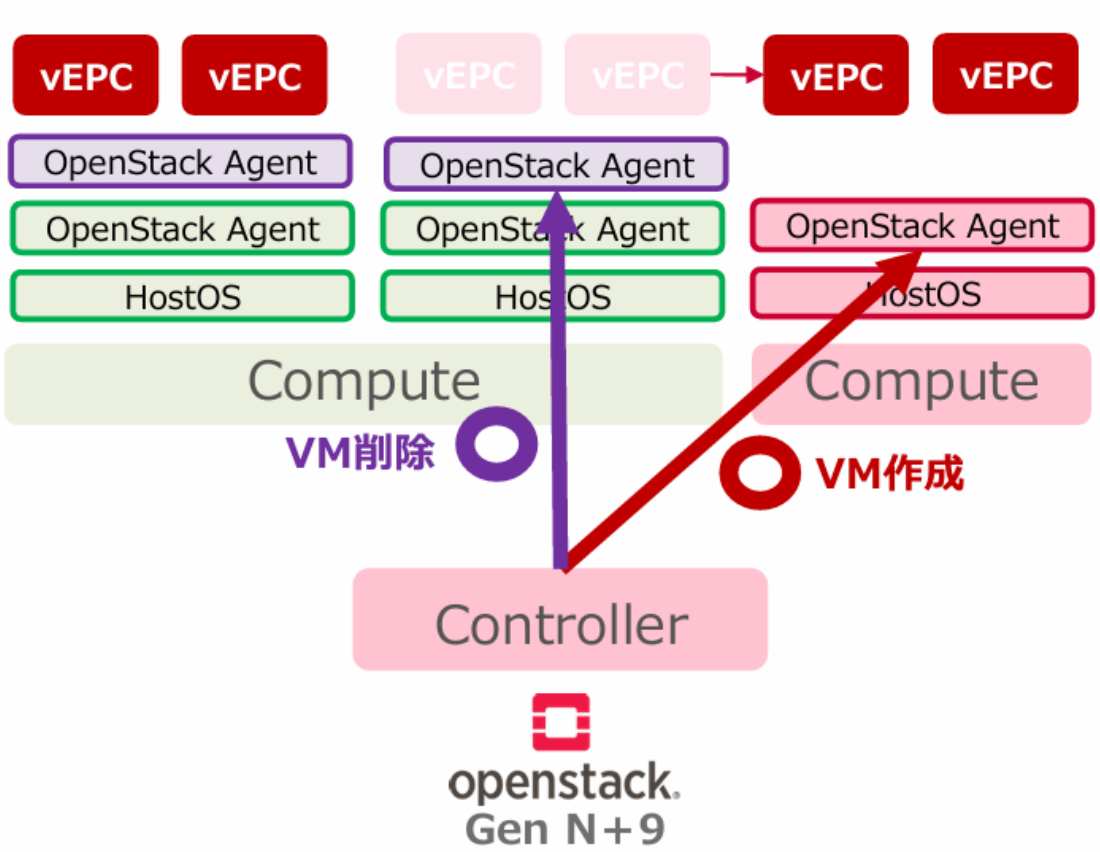
■ ジャンピングアップデートを実現するためには2つの障害
- Controller (victoria) がcompute (mitaka) で動くVMの操作が不可になりローリングアップデート不可
- APIの差分も発生し、外部装置も改修が必要になってしまう

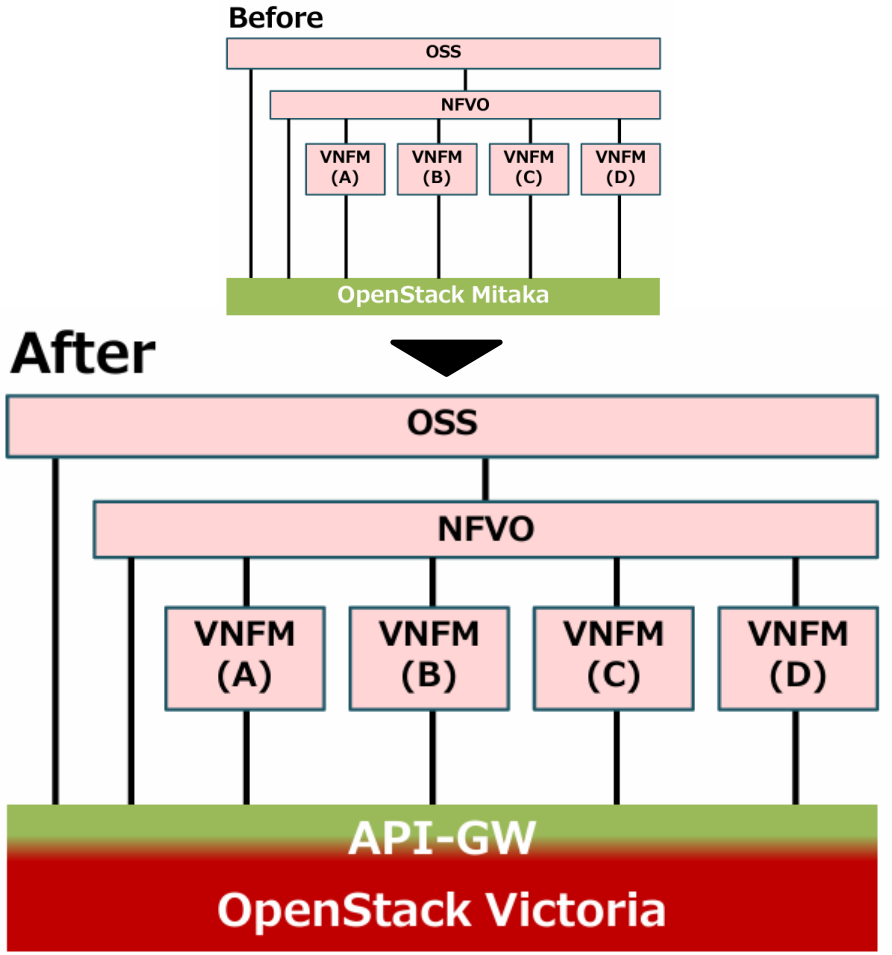
このままでは、サービスは止めざるを得ない、改修の費用も生まれる・・・



設計と運用のギャップ－OpenStackのジャンピングアップデートの対策－

- MitakaとVictoria間の差分を吸収する中間レイヤを独自に開発・実装
  - ジャンピングアップデートでもサービスは無中断かつ最小のコストでのアップデートを実現





設計と運用のギャップ－OpenStackのジャンピングアップデートの教訓－

教訓

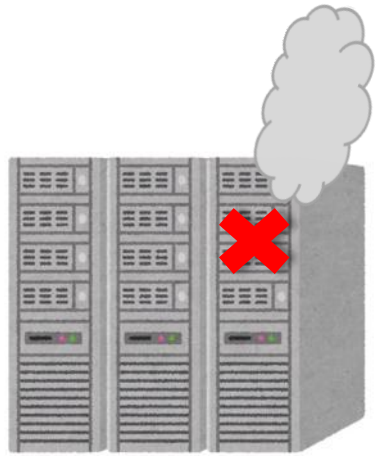
- 大規模かつサービス無中断を前提にすると、
  OpenStackの全世代アップデートは非現実的
- アップデートは途中世代を踏めない前提で更改の設計をすべき

通信品質の向上とコストの効率化を実現するための設計検討

■ 障害発生時の対応に関する設計と運用のギャップについて説明します

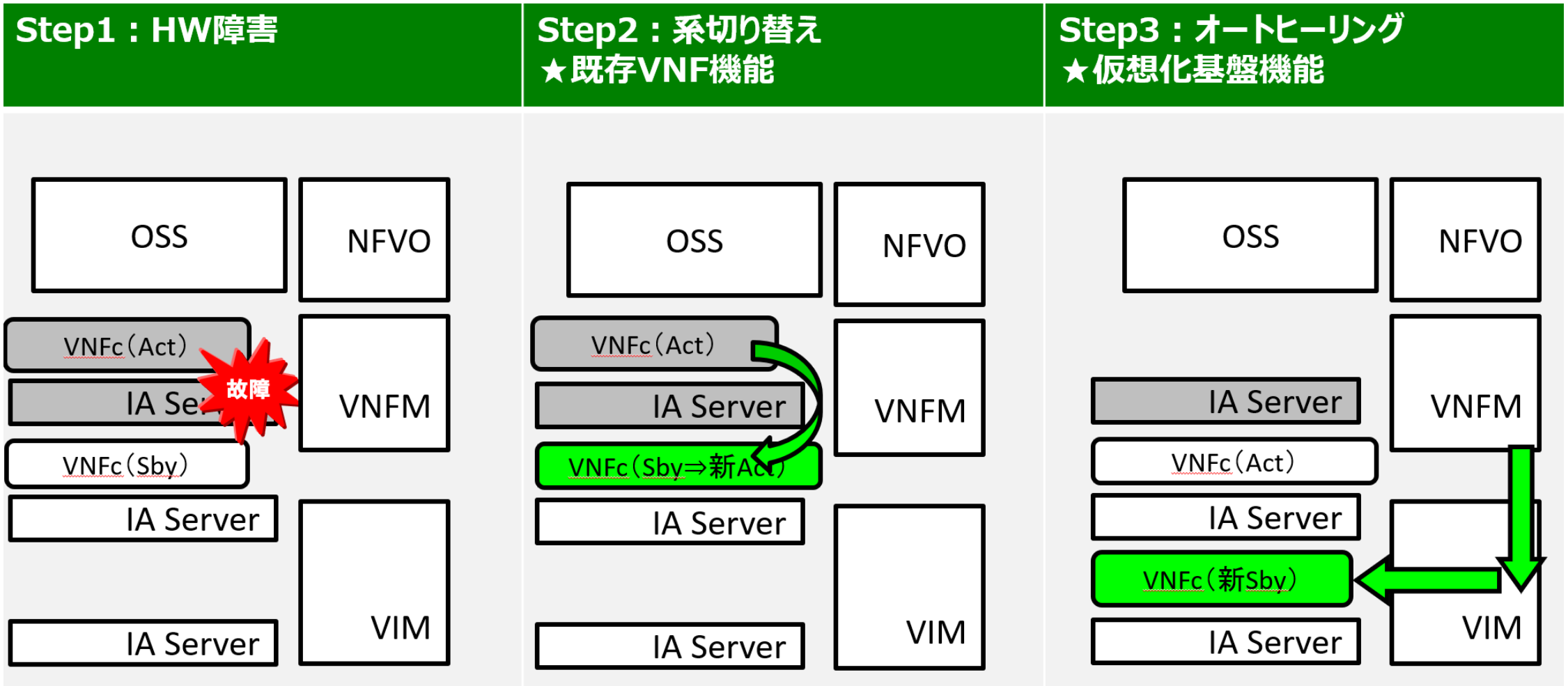
ライフサイクル（更改）に関する設計

障害発生時の設計



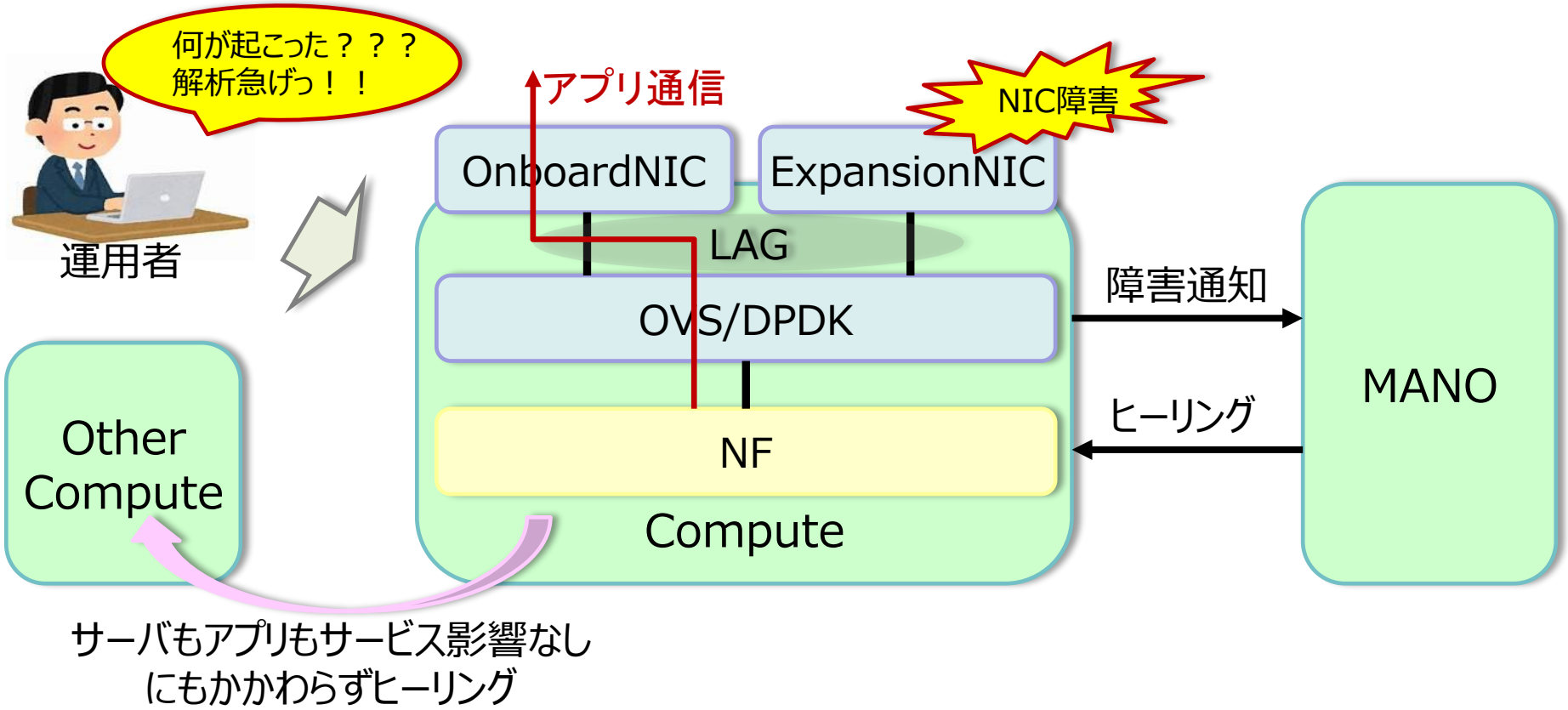
設計と運用のギャップ－自動ヒーリングの実装－

- HWの障害時にVMを別の正常なサーバへ退避する機能（ヒーリング）により、信頼性を向上
- 何かしらの障害があるサーバは使わない方が良いと考え、すべての障害を検知するように設計



設計と運用のギャップ－自動ヒーリングの罠－

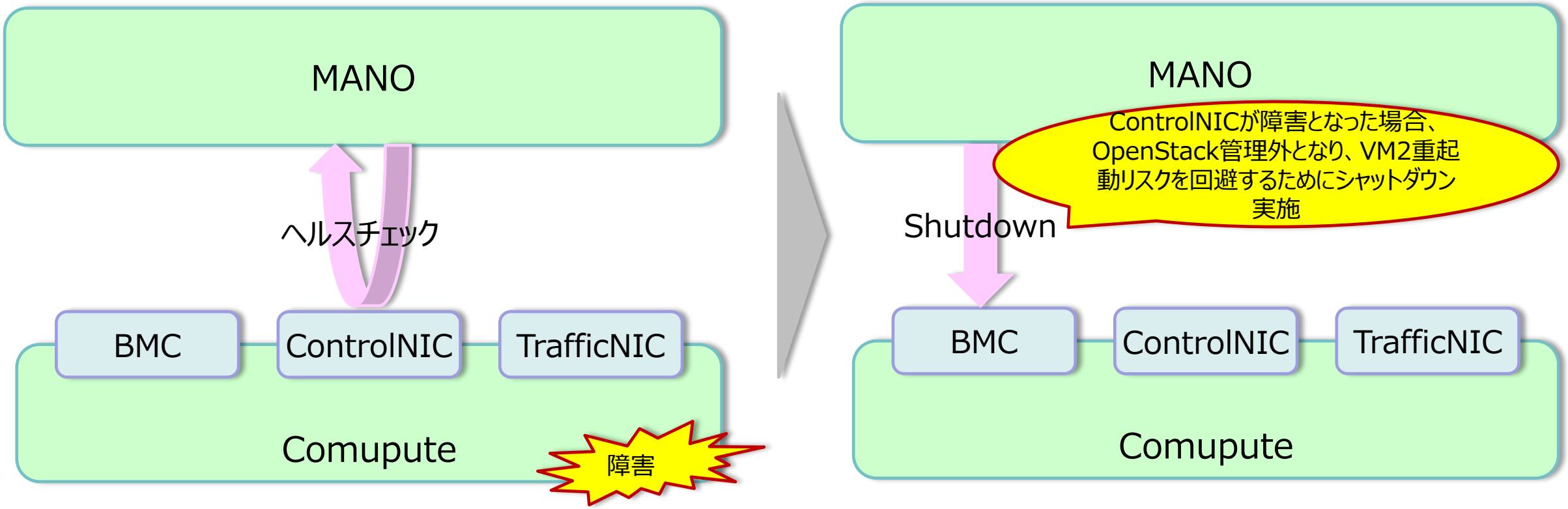
- 運用者は自動ヒーリングが動くと「なぜ動いたのか」、「システムに異常がないか」を都度チェックするプロセスが存在
- HDDやNICなどサービス影響に直結しない障害も自動ヒーリングが動き、チェックプロセスが動く
  - 問題ないにもかかわらず、問題が起こったように見え運用者のチェック稼働が増加



設計と運用のギャップ－自動ヒーリングの対策－

■ 障害をサーバの完全故障と部分故障に分類し、完全故障のみを検知する実装に変更

➢ 真にサービス影響がでる障害のみを救うことで、通信の信頼性を維持しつつ運用者の負荷を削減



設計と運用のギャップ－自動ヒーリングの教訓－

教訓

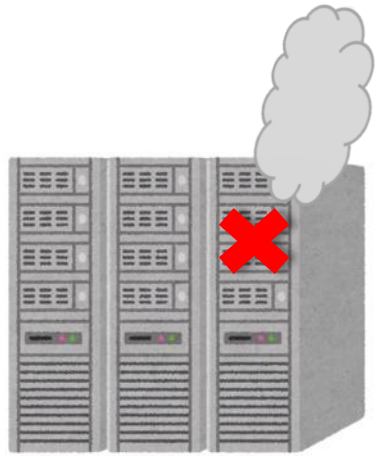
パーツレベルの障害でVMの退避はやりすぎ
VMのサービスが正常に動作できるのかを主軸に設計すべき

通信品質の向上とコストの効率化を実現するための設計検討

■ まだあります

ライフサイクル（更改）
に関する設計

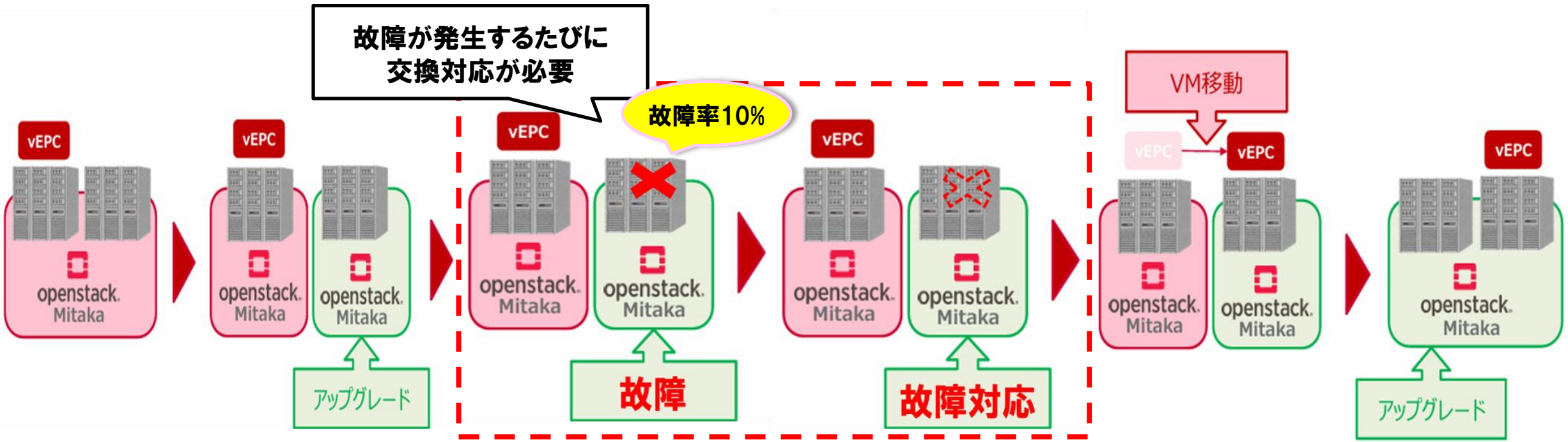
障害発生時の設計



設計と運用のギャップ−「直して進める」設計−

- 更改作業中の故障を考慮し、工数とサーバ種別から300パターン以上の復旧手順を整理
- いつどこで故障が発生しても直して工程を進められるように設計

OLUの工程48項目 × サーバの更改ステータス7種類 = 336パターン



設計と運用のギャップ–「直して進める」設計思想の罠–

■ 仮想化基盤の規模拡大に伴い、故障との遭遇率も上昇。都度故障交換の対応も耐えれない

更改前

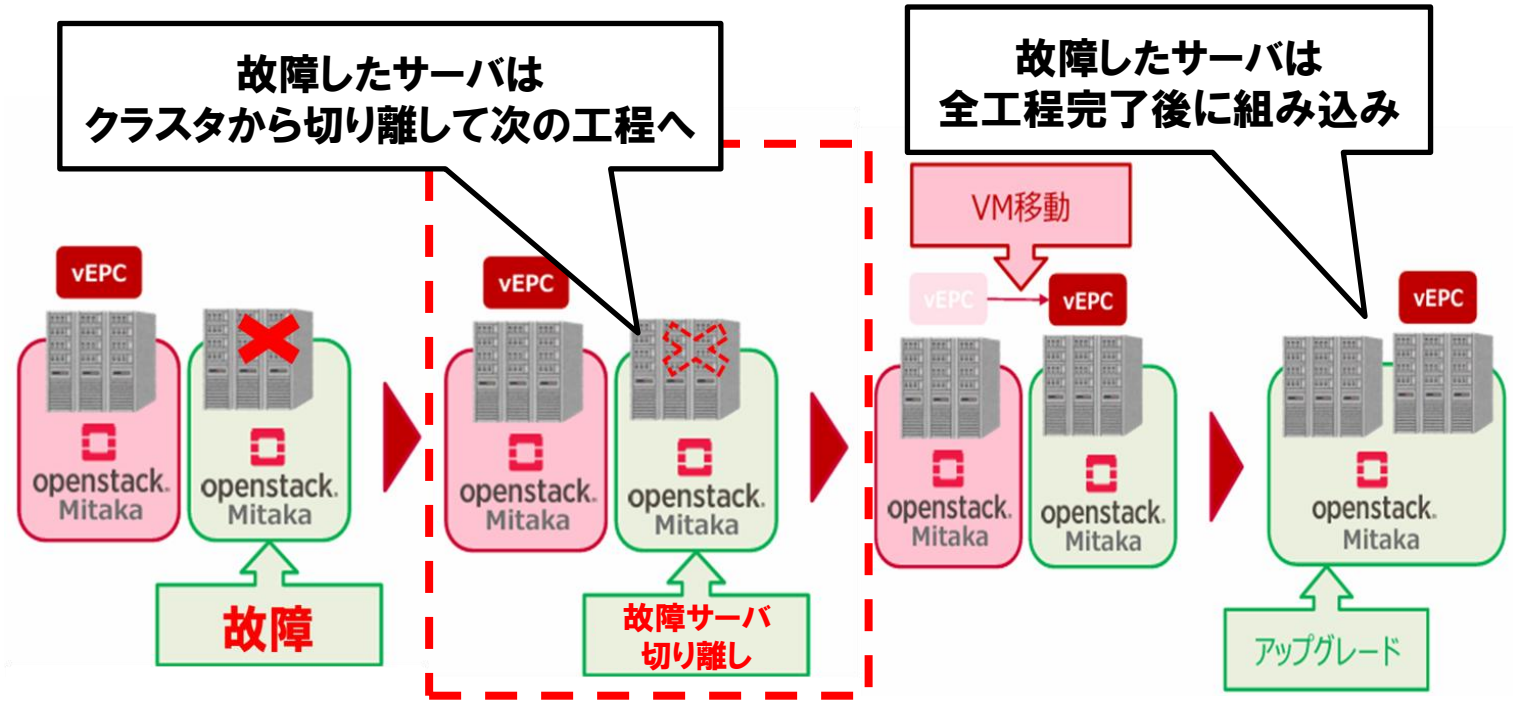
年間　9000台中300+件

更改中

年間　12000台中1200+件

設計と運用のギャップ−「直して進める」設計思想の転換−

- 大量のサーバを扱う仮想化基盤において、「正常系に戻して再開」の考え方は通用しない
  - そもそも仮想化基盤は、数台の故障サーバがいても運用できるのがコンセプトの一つ
- 故障は無視できるケースがあると展開し、HW故障を無視する例外処理を実装



設計と運用のギャップ−「直して進める」設計思想の転換で得た教訓−

教訓

大規模なシステムだからこそ、故障0台前提の運用は非現実的
仮想化だからこそサービス影響がない故障は工程優先できる
設計にすべき

まとめ

- ローリングアップデートそのものは、ドコモとして最良の判断
  - HWのみ、SWのみ、両方すべての更改パターンを網羅
  - 最小限の設備で更改ができたので、昨今のメモリ高騰にも耐えうる方式

- 仮想化の成否は「運用設計」で決まり、特に重要だったなのは
  ①ライフサイクル（更改）設計
  ②障害発生時の設計
- ポイントは「やりすぎないこと」