Networkプログラマビリティとステート性・トランザクション性

Network Programmability and the statefulness/transactionality

9 November 2015

Miya Kohno (mkohno@cisco.com)

SDNの本質的価値

- MPLS Japan 2012 Panel Discussionより

- 本質的価値
  - C-Plane/D-Plane分離でなく, 集中 vs 分散でなく, Openflow protocol自体でなく
  - ソフトウェアのagility, 開発方法論, test, debug
  - 思考形態・文化、オープン化自体
  - Single point of failureとかscalabiiltyの議論とかではなく
  - OpenなInterface, Protocol開発とは違う自由さ。
  - 活かすも殺すもエンジニア次第　自由度、マルチキャスト得意。
  - ネットワークは、ユーザから見ると遠い。もう一度ネットワークを自分の手許に。
  - 個人をenpowerするもの
  - いつか来た道 - 残る： API, 12 tuple forwarding　あぶない：外注・乱立・
- Cloud Orchestration, 自動化, Programmability
  - PCEも　(Multilayer use case..)

現在のSDN喧噪騒ぎの後に残るものは何か ?!

あれから3年！

真に求められていたのは
プログラマビリティと,
それによるサービス迅速化
＋運用自動化だった！

「ネットワーク仮想化」に求められていたことも同様…

http://www.mpls.jp/2012/presentations/MK_SDN_Panel_MPLSJapan2012_rev3.pdf

では，どう「プログラム」するか？

…という訳で，

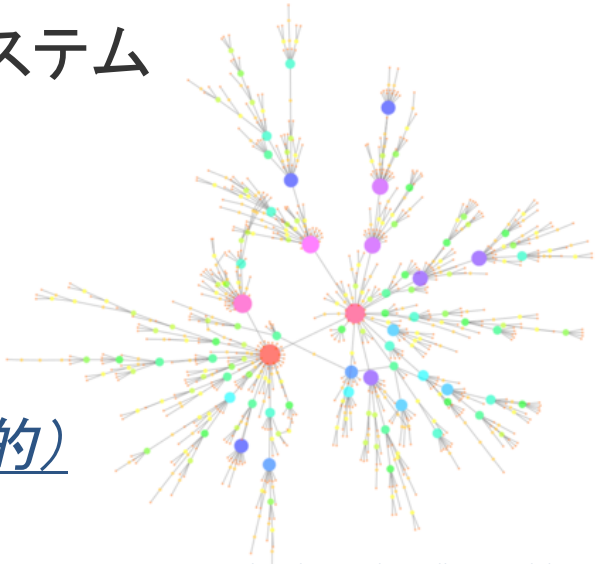
本日はNetworking領域におけるプログラミングスタイルを議論します

議論のポイント

- プログラミングスタイルの重要性
- ステート性 – ステートをどう扱うか
- トランザクション性 – 分散システムにおける整合性問題

Network領域におけるプログラミングスタイル

ネットワークは不確定性の高い，並列分散システム



Wikipedia: Barabasi-Albert Model

- Waterfallよりも *Agile*

  ---- フィードバックと継続的改善

- Imperative（命令的）よりも *Declarative（宣言的）*

  ---- Howを指示するのでなく，Whatを合意

- Procedureよりも *Model driven*

  ---- 逐次命令するのではなく，あるべき姿をモデルで示す

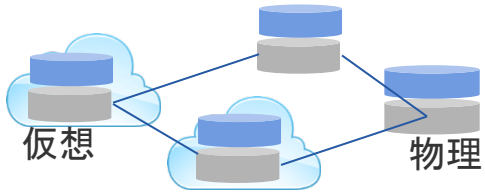
“Declarative” “Model-driven” 性 の強み

“How”を”記述”する 場合

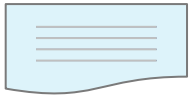
サービス・アプリケーション

オーケストラ・コントローラ

デバイス・インフラ



- Script
- Workflow



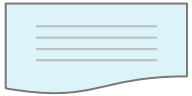
← Procedure

状態変化に対応困難

部分的変更，削除が困難

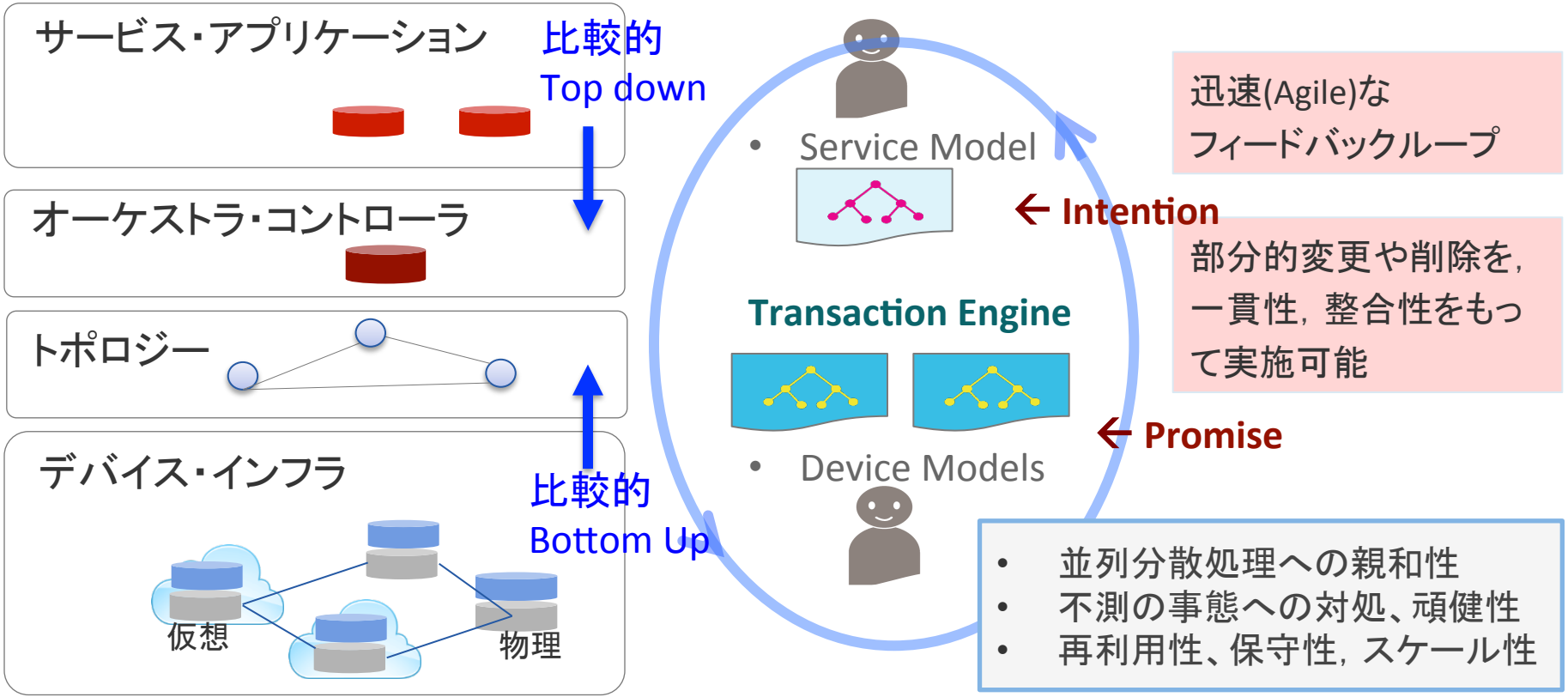
- CLI
- Openflow protocol

← Command



“Declarative” “Model-driven” 性 の強み

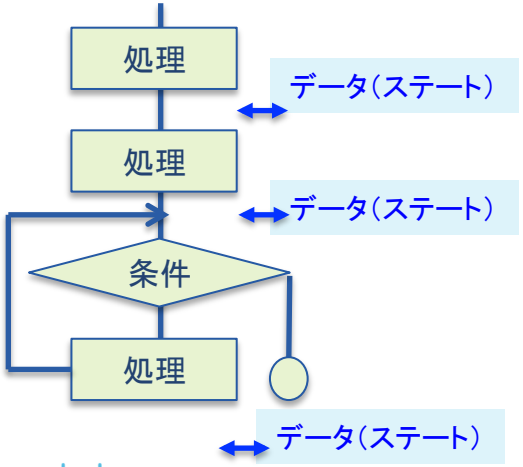
“What”を”合意”する 場合

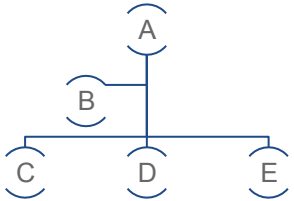


Computing領域におけるプログラミングスタイルの変遷

手続き型
Procedural

構造化
Structuralized





オブジェクト指向
OOP

オブジェクトが
データ(ステート)
を内包する

ステートの明示化

REST
(Representational State Transfer)

冪等性
(Idempotence)

参照透過性
(Referential
Transparency)

関数型
Functional Programming

ステートレス化

[補足] 冪等性(Idempotence)，参照透過性(Referential Transparency)

冪等性(Idempotence)

ある操作を一度行っても複数回行っても結果が変わらないことを表す概念
（例えば、n++;は冪等でない．冪等性が保証できないと、例えば timeout -> retry の場合に結果が変わってしまう可能性がある）

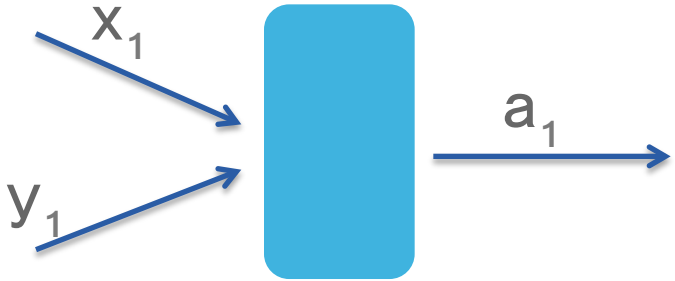
参照透過性(Referential Transparency)

文脈に依らず、式の値はその構成要素だけによって決まる。
同じ条件を与えると、必ず同じ値が返る。
（外部変数・グローバル変数とかを使ってはいけない。）

→ 並列分散処理，ネットワークプログラミングにも重要な概念

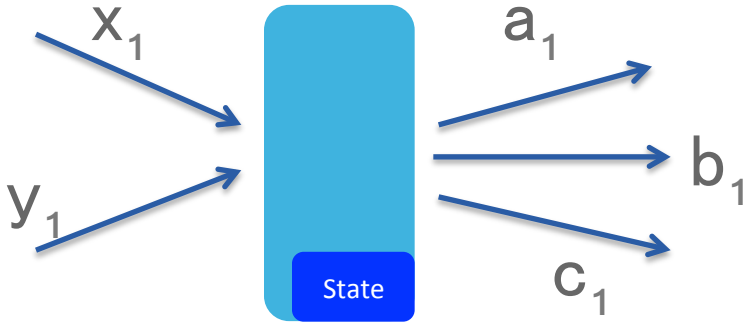
Stateの問題

* Stateless



e.g. a = f(x + y)

* Stateful

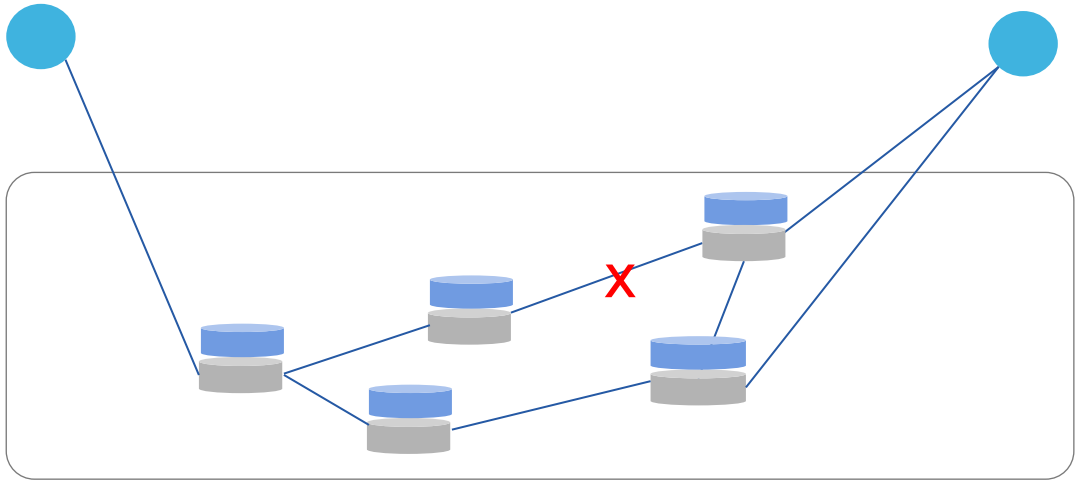


*e.g. a = f(x + y * N_{state})*

Stateによって結果が異なる
→並列分散処理に適しにくい

- 並列分散するエンティティにおけるStateの一貫性が必要
- High Availabilityを実現する際に，Stateの複製が必要

Networkは Statefulな並列分散システム



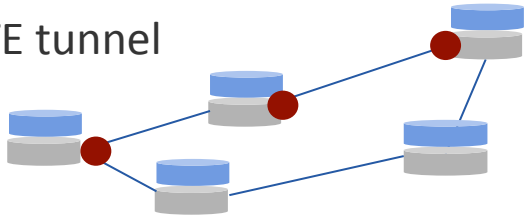
State Mismatch ← Topology Convergence, RIB(CP)-FIB(DP) inconsistency

↓

Routing Loop, Black-holing

Stateの度合いはさまざま

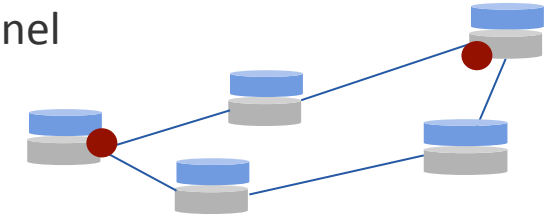
RSVP TE tunnel



Route情報
＋
Tunnel Endpoint情報
＋
RSVP state

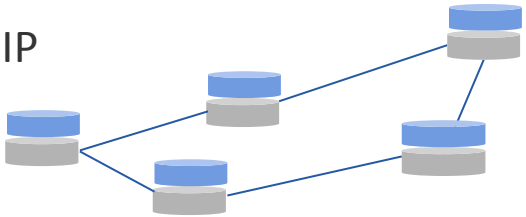
その他
PPP/L2TPとか
IPSec/SSLとか
SIPとかMobileとか..

IP tunnel



Route情報
＋
Tunnel Endpoint情報

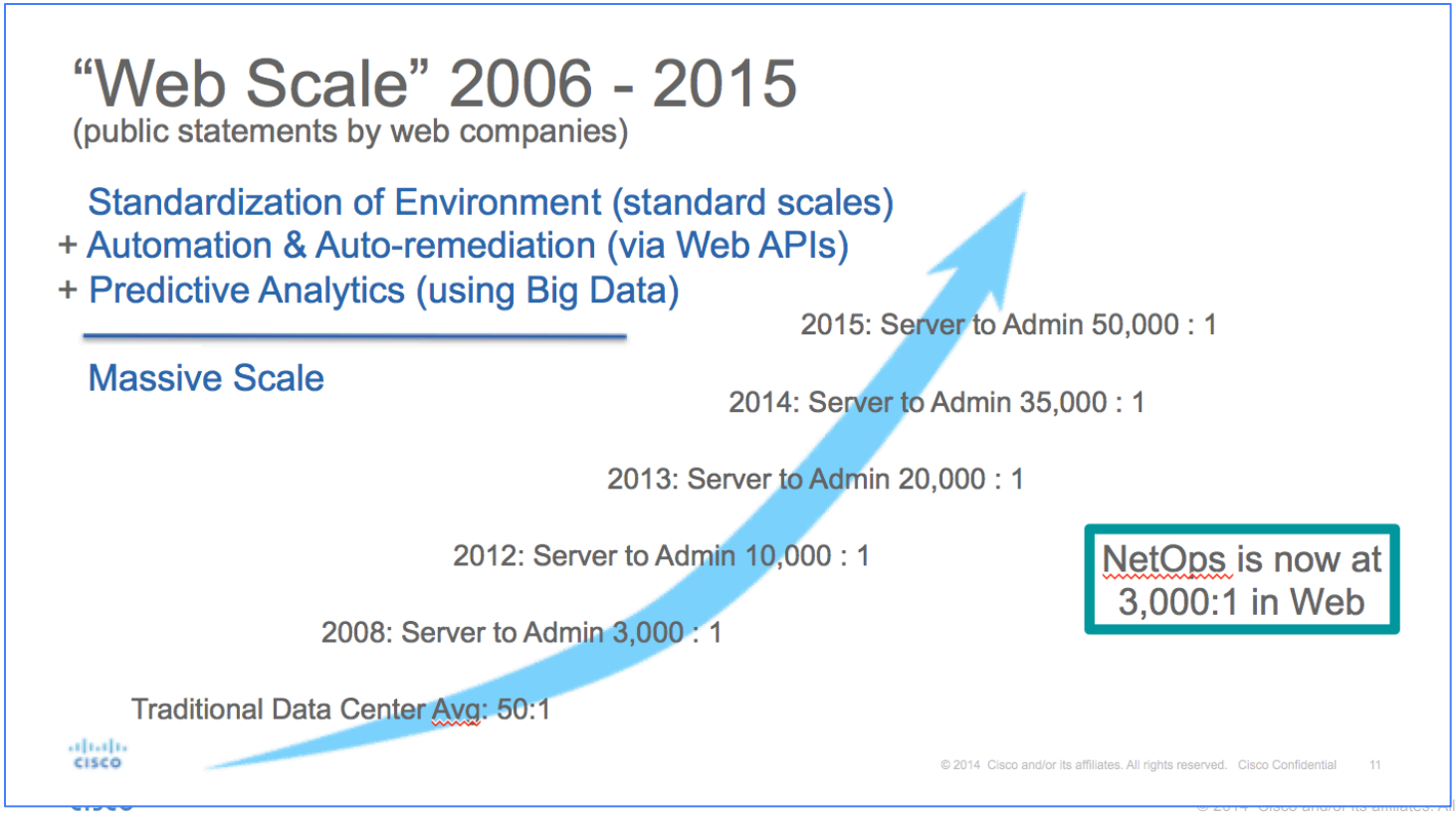
Native IP



Route情報のみ

- ステートとスケールはTrade-off
- ステートフルの場合，N:1冗長などもやりにくい

Computing SystemにおけるMassive Scaling

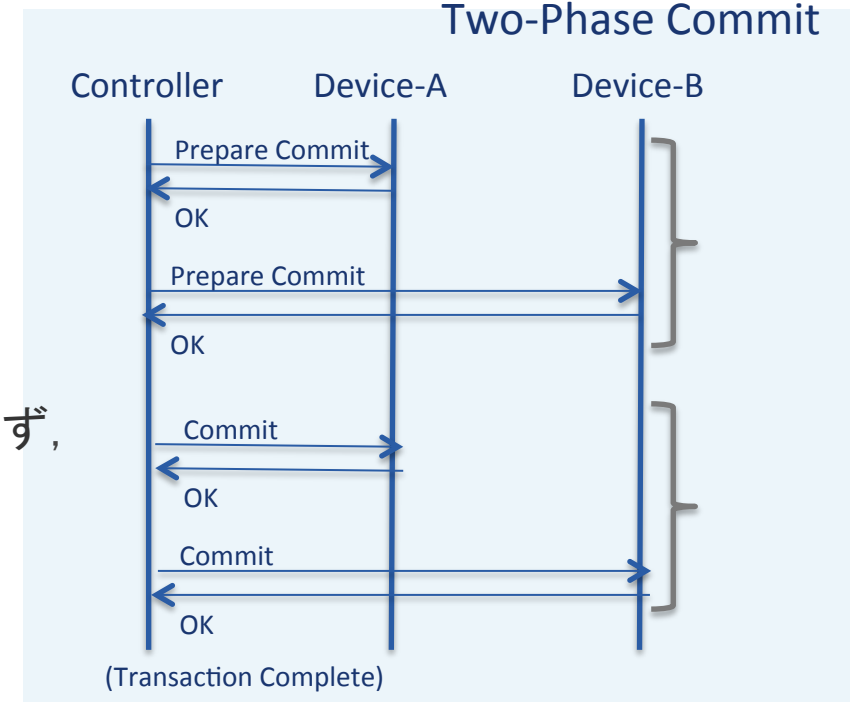


- 超並列分散化
- ACID -> BASE

Source:
Colin Kincade@Cisco

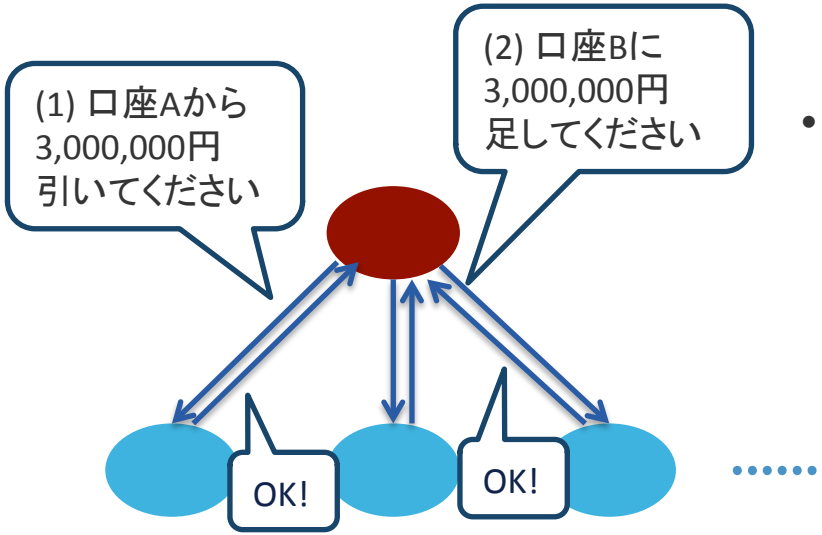
ACID – 分散システムの整合性問題

- Atomicity
  原始性 – 操作は，完了するか，一切なにもしないかのどちらかである
- Consistentncy
  一貫性 – 整合性条件が一貫して保持される
- Isolation
  独立性 – 処理中の中間状態は隠蔽される
- Durablity
  永続性 – 完了したトランザクションは取消されず，システムは継続する



ACID – 分散システムの整合性問題

（例）口座AからBへの振込処理

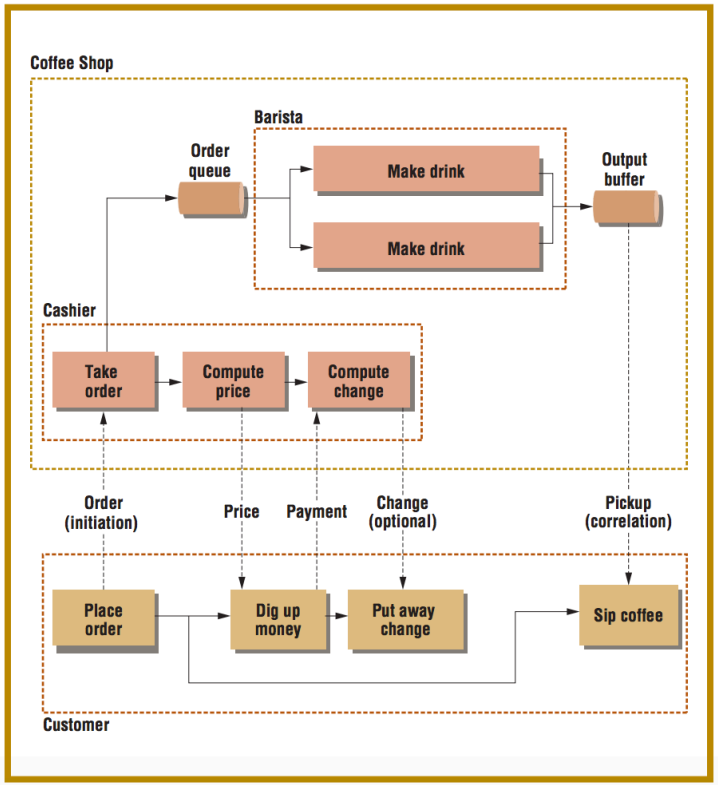


- (1)と(2)の両方の処理が成功しなければ，操作は完了しない
- どちらかが失敗した場合は，全て元に戻す

このためには，2 phase commit ＋
操作完了するまでのデータベースロック
が必要

→ スケールしない！！！！

Your Coffeshop doesn’t use two phase commit!



[東京のStarbucksでの体験からの考察]

顧客から注文を受けたらすぐに作り始める

注文の言い間違い・聞き間違いなどで,
欲しいものと違うものを作ってしまったら?
→ 謝って作りなおす(やりなおし)

顧客が料金を払わなかったら?
→ コーヒーは廃棄(無かったコトにする)

→ スケールする！！！！

http://www.enterpriseintegrationpatterns.com/docs/IEEE_Software_Design_2PC.pdf

BASE – ACIDの代替手段として

- Basically Available
  基本的に動作している
- Soft-state
  ソフトステート – あまり厳密なステートの一貫性を求めない
  楽観的，Best effort, Simple, Weak Consistency…
- Evetually consistent
  結果整合 – 一時的に不整合が起こりえるが，
  結果的には整合性が取れる

Base: An Acid Alternative

In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability.

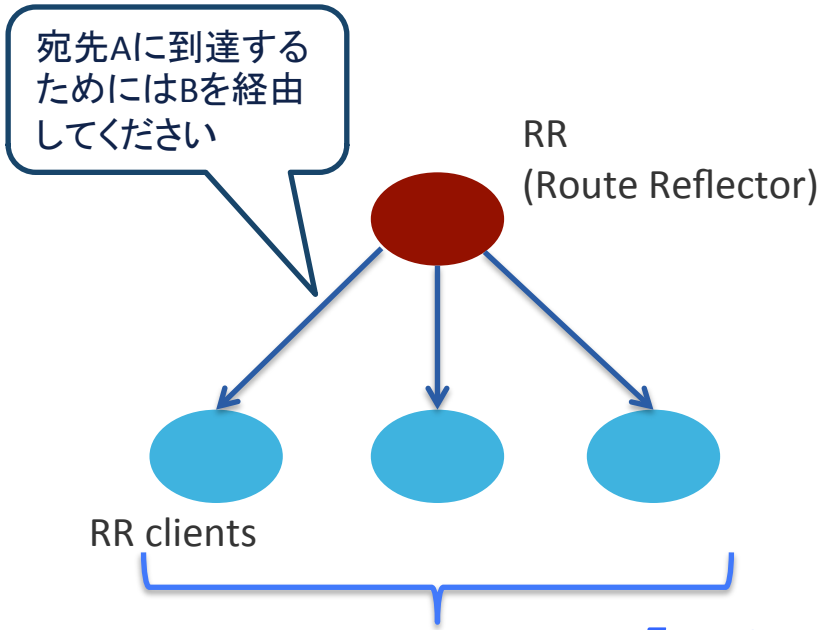
Dan Pritchett, Ebay

Web applications have grown in popularity over the past decade. Whether you are building an application for end users or application developers (i.e., services), your hope is most likely that your application will find broad adoption—and with broad adoption will come transactional growth. If your application relies upon persistence, then data storage will probably become your bottleneck.

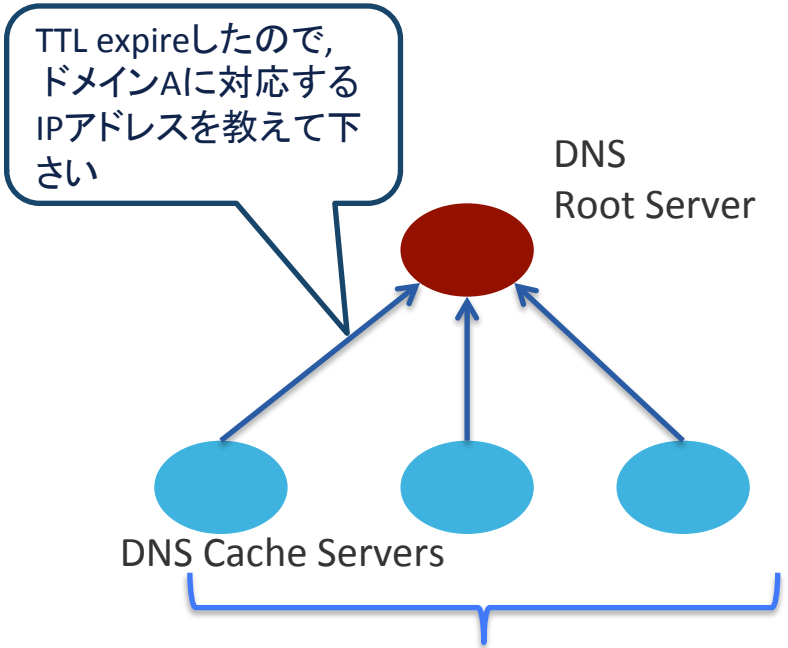
http://queue.acm.org/detail.cfm?id=1394128

Network Programmingは元々BASE！

Routing Protocols (e.g. BGP)

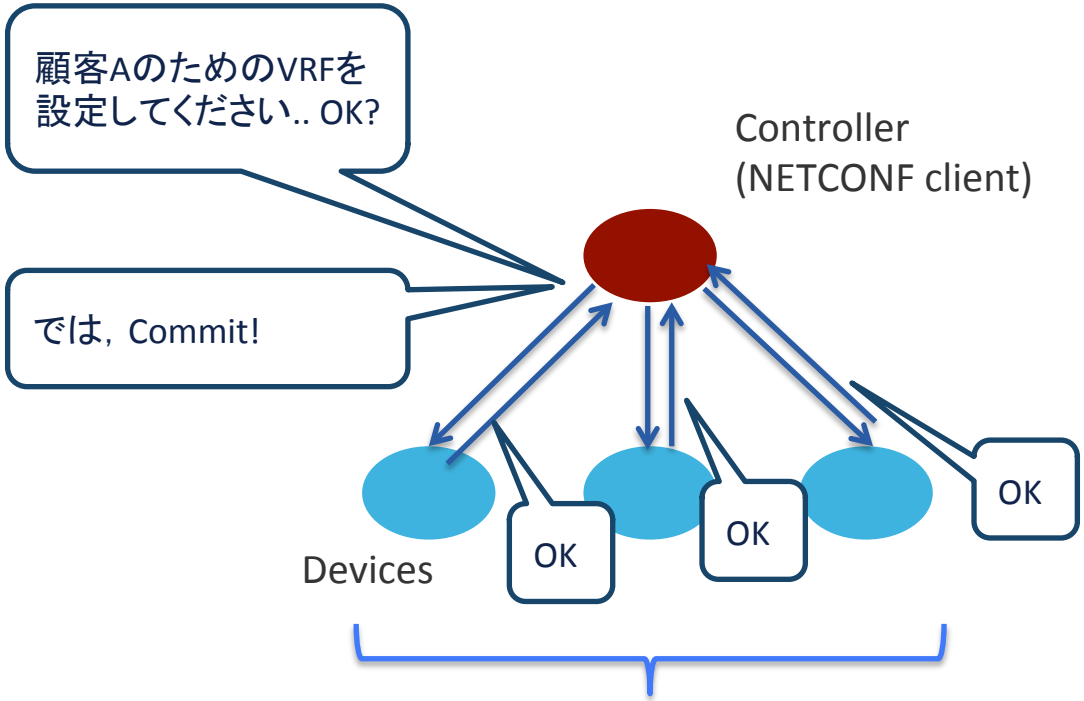


DNS



Eventually consistent !

NETCONFは..?!

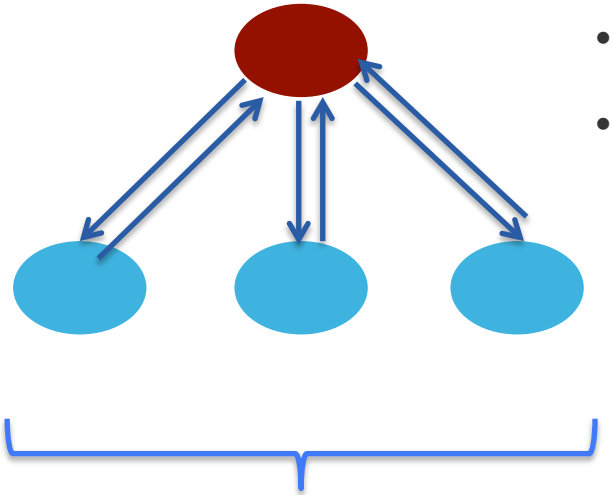


ACID可能！

RFC6241 ：

The NETCONF protocol contains sufficient primitives upon which transaction-oriented operations can be built. Providing complete transactional semantics across multiple devices is prohibitively expensive, but the size and number of windows for failure scenarios can be reduced.

ACIDを適用したほうが良いかもしれないところ



ACID可能！

スケール制御可能な範囲であれば，効用が大きい

- ASドメイン内 Controller – Device
- Cluster Controller – DC Fabric/Cluster

Routing loopやパケットロス，
ブラックホールなどの
いかなる不整合を防ぐことが
できる！

Network Programmability – まとめ

ネットワークは，不確定性の高い並列分散システム

- Model-driven, Declarative なプログラミングスタイルが適する
- ステートはできるだけ最小化，局所化する
- 必要性，適用性に応じて，トランザクション一貫性を実装する

TOMORROW starts here.