OPEN SOURCE NETWORKING DAYS

YANG Centric Automation with OpenDaylight

OpenDaylightで実現するYANG中心の自動化

コア&トランスポート技術本部
C&I統括部 パケットネットワーク部
堀場 勝広

アウトライン

- 自己紹介
- 通信事業者におけるオペレーション自動化の課題
- ソリューションはYANGとOpenDaylight
- これからの自動化スクリプト開発の流れ
- まとめ

自己紹介

- 元大学研究者
  - 研究領域はNW仮想化、SDN、NFVなど
  - コンソーシアムなどで企業の皆さんとオープンソース技術を研究
- 2015年からソフトバンク株式会社
  - オープンソース技術を利用したSDNの研究開発に従事
  - OpenDaylightとの出会い
  - Lithium(3rd Release)から参戦
- 2017年から商用ネットワーク設計部門に異動
  - OpenDaylightを利用した商用ネットワークのSDN化に従事
  - SDNコントローラの上位システム(自動化システム)を含む

通信事業者におけるオペレーション自動化の課題

事故が怖い

↓

バージョンアップと
正規表現との戦い

引き継ぎが大変

↓

他人の書いたコードと
無限のツールとの戦い

レガシーの存在

↓

オーダーメイド装置と
非CLIな装置との戦い

事故が怖い…バージョンアップと正規表現との戦い

装置のバージョンアップなどに追従しながらメンテ続けるのは難しい

```
Value Required INTERFACE (¥S+)
Value LINK_STATUS (¥w+)
Value PROTOCOL_STATUS (.*)
Value HARDWARE_TYPE ([¥w ]+)
Value ADDRESS ([a-zA-Z0-9]+.[a-zA-Z0-9]+.[a-zA-Z0-9]+)
Value BIA ([a-zA-Z0-9]+.[a-zA-Z0-9]+.[a-zA-Z0-9]+)
Value DESCRIPTION (.*)
Value IP_ADDRESS (¥d+¥.¥d+¥.¥d+¥.¥d+¥/¥d+)
Value MTU (¥d+)
Value DUPLEX (.+?)
Value SPEED (.+?)
Value BANDWIDTH (¥d+¥s+¥w+)
Value DELAY (¥d+¥s+¥w+)
Value RELIABILITY (¥d+)
Value TXLOAD (¥d+)
Value RXLOAD (¥d+)
Value ENCAPSULATION (¥w+)
Value INPUT_DROPS (¥d+)
Value OUTPUT_DROPS (¥d+)
Value QUEUE_STRATEGY (.*)
Value INPUT_RATE (¥d+)
Value OUTPUT_RATE (¥d+)
Value INPUT_PACKETS (¥d+)
Value INPUT_BRAODCAST (¥d+)
```

```
Start
  ^${INTERFACE} is ${LINK_STATUS}.*protocol is ${PROTOCOL_STATUS}
  ^¥s+Hardware is ${HARDWARE_TYPE} -> Continue
  ^.*address is ${ADDRESS}.*bia ${BIA}
  ^¥s+Description: ${DESCRIPTION}
  ^¥s+Internet address is ${IP_ADDRESS}
  ^¥s+MTU ${MTU}.*BW ${BANDWIDTH}.*DLY ${DELAY}
  ^¥s+reliability ${RELIABILITY}.+txload ${TXLOAD}.+rxload ${RXLOAD}
  ^¥s+Encapsulation ${ENCAPSULATION}
  ^¥s+Input queue: ¥d+/¥d+/${INPUT_DROPS}/¥d+ -> Continue
  ^.*Total output drops: ${OUTPUT_DROPS}
  ^¥s+Queueing strategy: ${QUEUE_STRATEGY}
  ^¥s+${DUPLEX}, ${SPEED}, media type
  ^.*input rate ${INPUT_RATE}
  ^.*output rate ${OUTPUT_RATE}
  ^¥s+${INPUT_PACKETS} packets input
  ^¥s+Received ${INPUT_BRAODCAST} broadcasts
  ^¥s+${INPUT_ERRORS} input errors
  ^¥s+${OUTPUT_PACKETS} packets output
  ^¥s+${OUTPUT_ERRORS} output errors -> Continue
  ^.* ${INTERFACE_RESETS} interface resets -> Record
```

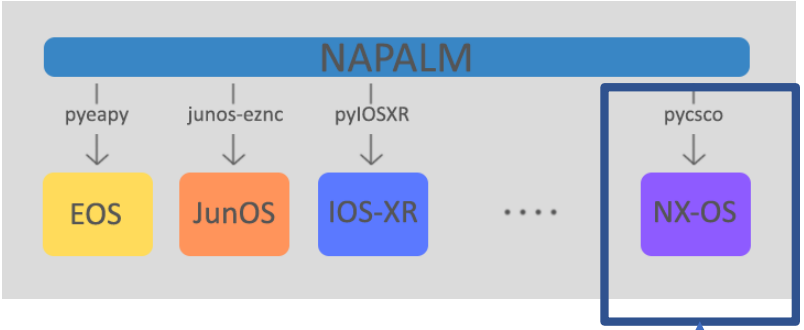
引用: **Cisco IOSのshowコマンドをk-meansでクラスタリング**
https://qiita.com/tokatsu/items/573f846f835f5c3d92c6

引継ぎが大変…他人の書いたコードと無限のツールとの戦い

他人が書いたコードを理解し、適切に拡張して行くのは難しい

プログラミング言語	Expect(Telnet CLI)	Netconf	制御抽象化	コマンド抽象化
perl	Net::Telnet::Expect	Net::Netconf		
python	pyexpect	ncclient	netmiko	napalm
go	goexpect	go-netconf		

様々なライブラリの組合せ → 当然、対応状況もまちまち



	EOS	FORTIOS	IOS	IOSXR	JUNOS	NXOS	PANOS	PLURIBUS	ROS	VYOS
get_arp_table	✓	✗	✓	✓	✓	✓	✓	✗	✗	✓
get_bgp_config	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗
get_bgp_neighbors	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
get_bgp_neighbors_detail	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗
get_config	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
get_environment	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓
get_facts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
get_firewall_policies	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
get_interfaces	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
get_interfaces_counters	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
get_interfaces_ip	✓	✗	✓	✓	✓	✓	✓	✗	✗	✓
get_lldp_neighbors	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗
get_lldp_neighbors_detail	✓	✗	✓	✓	☠	✓	✓	✗	✓	✗

また新しいライブラリを覚えねば・・・

レガシーの存在…オーダーメイド装置と非CLIな装置との戦い

ググって正規表現の雛形が出てくる装置ばかりではない

- オーダーメイド装置
  - 自分たち以外に利用者が見当たらない
  - ググっても誰も教えてくれない
  - 当然、世の中にはライブラリもない

- 非CLIな装置対応
  - NMSと装置の間での利用を想定
  - Telnet/Sshならまだマシな方で・・・
  - TL-1とかITU-T系の古いプロトコルとかの伝送装置系

これらの課題、YANGと共にOpenDaylightで解決します

事故が怖い

↓

バージョンアップと
正規表現との戦い

↓

正規表現からYANG
非構造化テキストは廃止

引き継ぎが大変

↓

他人の書いたコードと
無限のツールとの戦い

↓

手動開発から自動生成
コードはYANGから生成

レガシーの存在

↓

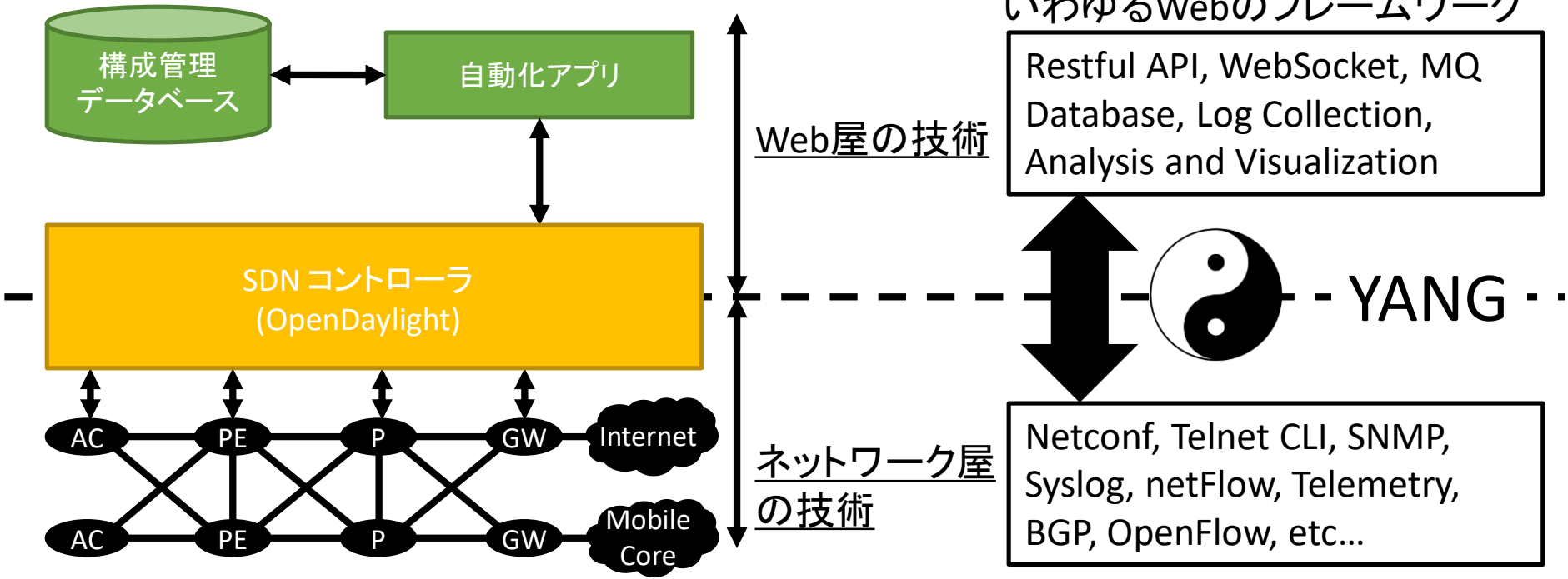
オーダーメイド装置と
非CLIな装置との戦い

↓

データ構造とSBIを分離
SBI非依存でRestで統一

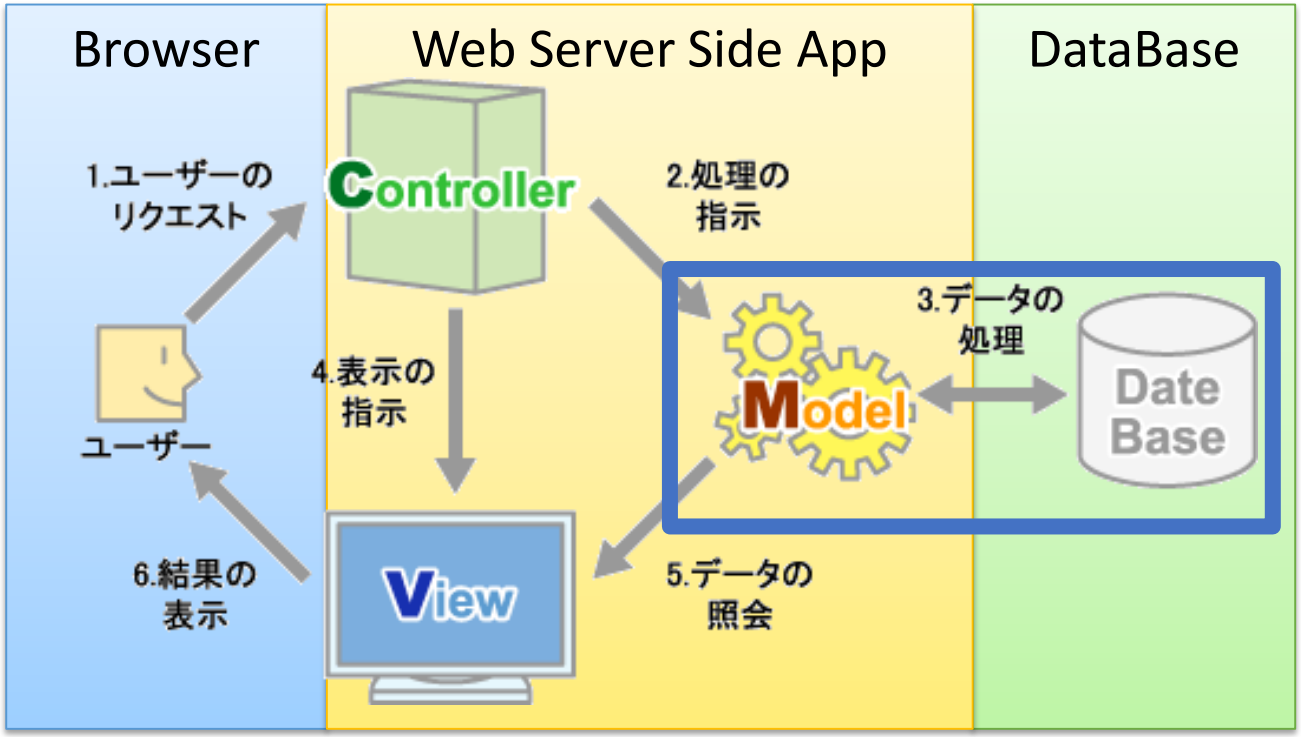
目指すのはWeb屋の技術によるネットワークの制御

モダンなWeb屋の道具を活用した自動化アプリ開発の促進
YANGはネットワーク屋の技術とWeb屋の技術の橋渡しを担う



モダンなWebアプリのフレームワーク

Model View Controller(MVC)パターンで成り立っている

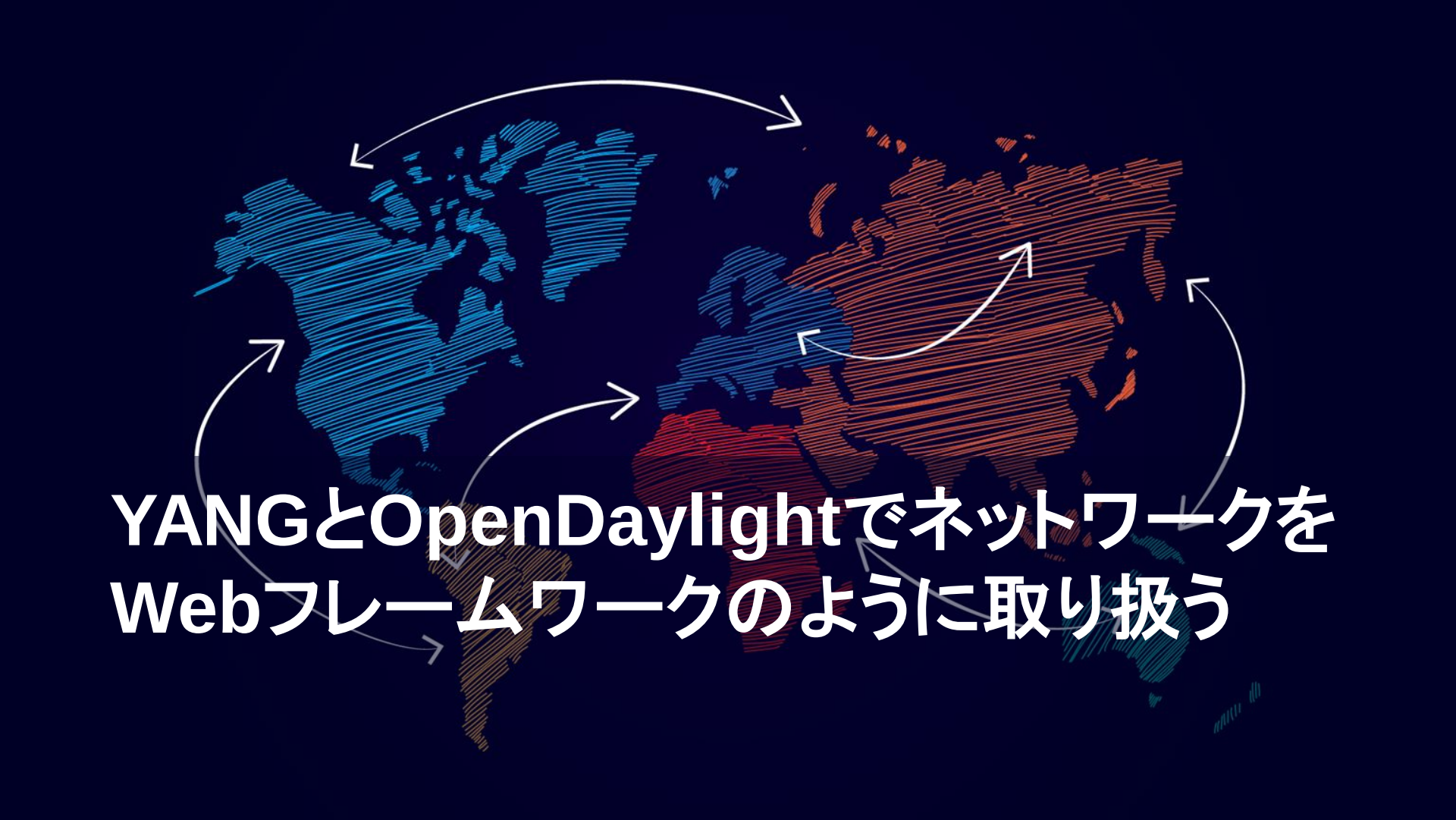


データの読み書きで
DBには直接アクセスしない

必ずモデルを介して
アクセスする

何故なのか？

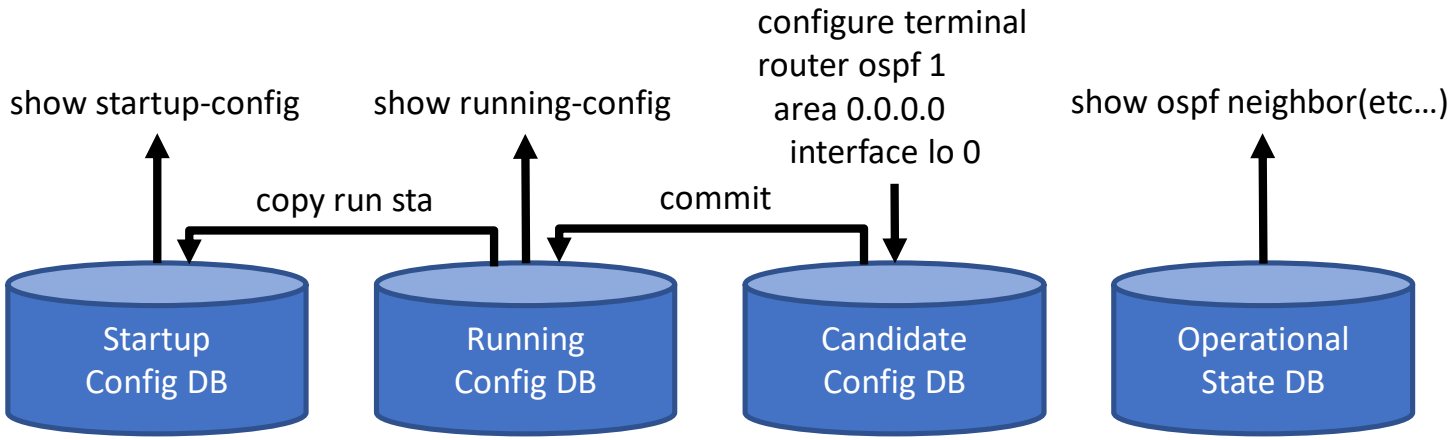
軽量なMVCフレームワークの自作
https://codezine.jp/article/detail/104



YANGとOpenDaylightでネットワークをWebフレームワークのように取り扱う

モダンなNW装置はデータベースである

装置の設定、状態などはデータベースに格納されている



起動時に参照される
設定の格納場所

動作中設定の格納場所
show runで参照される

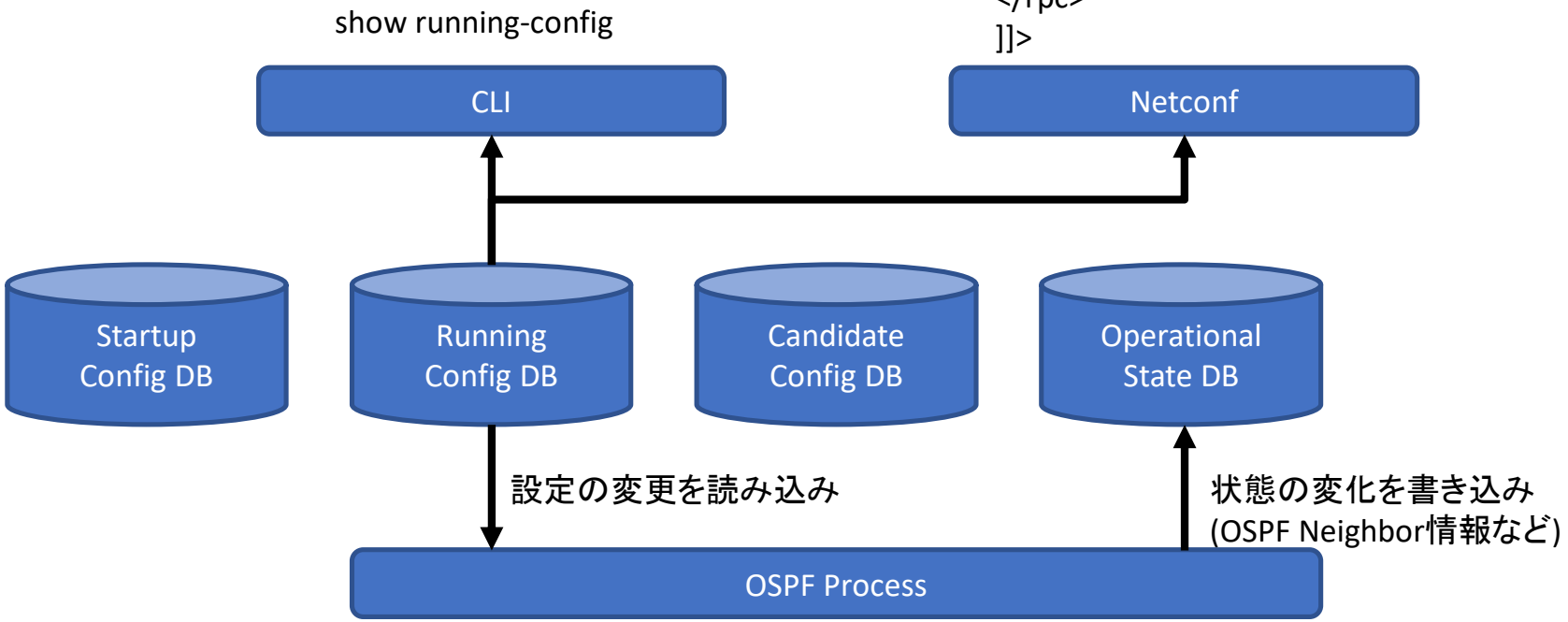
Commitする前の
設定の格納場所

show xxxで参照される
ネットワークの状態

モダンなNW装置はデータベースである(contd.)

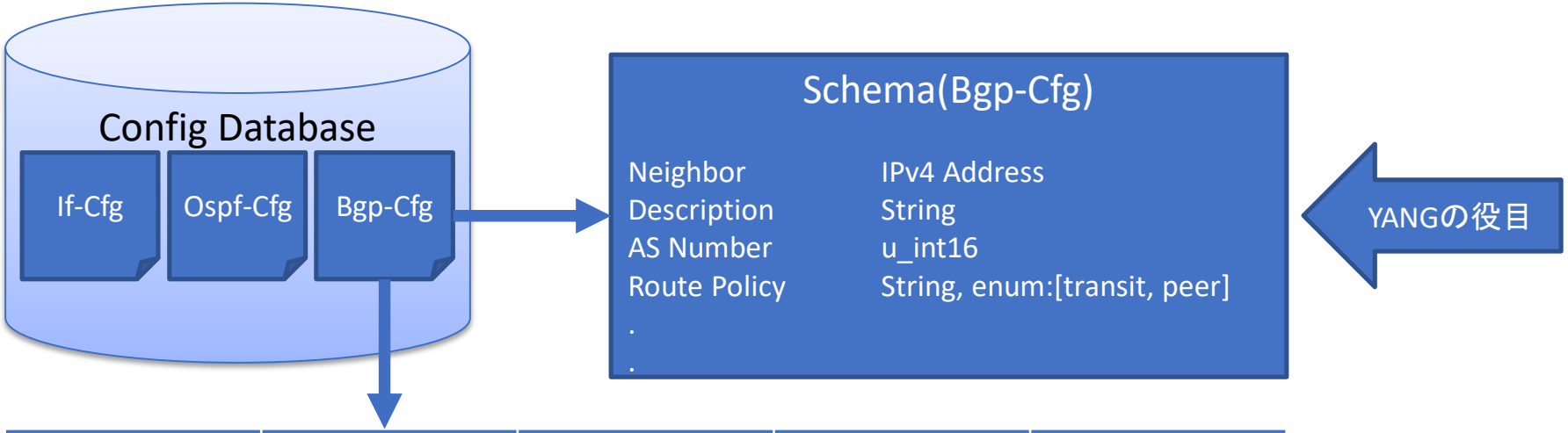
NetconfもCLIも同じDBを読み書きしている

```
<rpc>
  <get-config>
    <source><running/></source>
  </get-config>
</rpc>
]]>
```



YANG: NW装置が持つDBのスキーマ

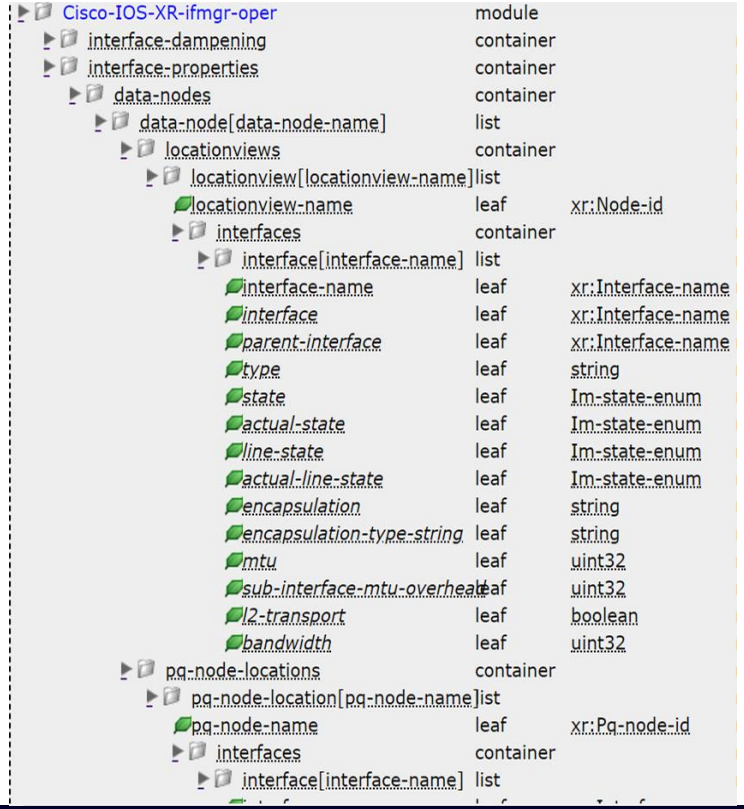
ネットワーク装置の設定文法は、YANGスキーマに書いてある



Neighbor	Description	AS Number	Route Policy	…
1.2.3.4	Tier-1 AS-1	1234	transit	…
9.8.7.6	Peer AS-1	9876	peer	…
…	…	…	…	…

YANG = ネットワーク装置のスキーマ記述言語

装置ベンダーさんがサポートしてくれる「安全な」装置のスキーマ



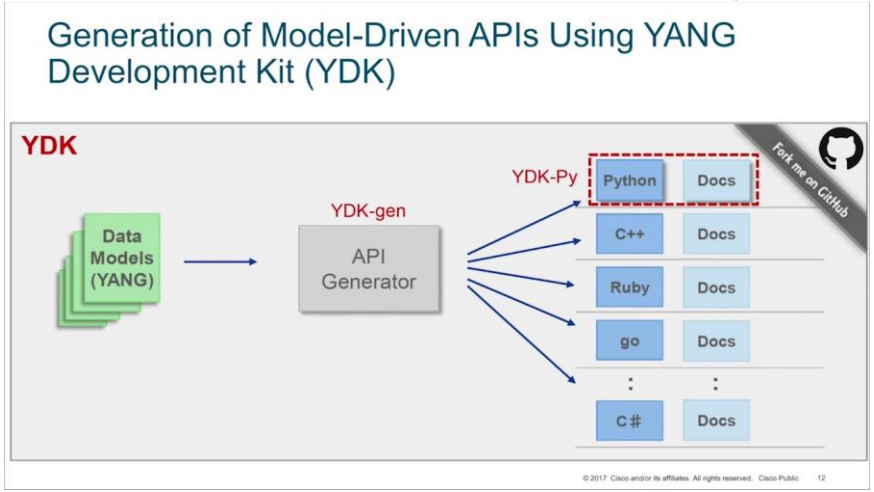
Cisco-IOS-XR-ifmgr-oper	module	
interface-dampening	container	
interface-properties	container	
data-nodes	container	
data-node[data-node-name]	list	
locationviews	container	
locationview[locationview-name]	list	
locationview-name	leaf	xr:Node-id
interfaces	container	
interface[interface-name]	list	
interface-name	leaf	xr:Interface-name
interface	leaf	xr:Interface-name
parent-interface	leaf	xr:Interface-name
type	leaf	string
state	leaf	Im-state-enum
actual-state	leaf	Im-state-enum
line-state	leaf	Im-state-enum
actual-line-state	leaf	Im-state-enum
encapsulation	leaf	string
encapsulation-type-string	leaf	string
mtu	leaf	uint32
sub-interface-mtu-overhead	leaf	uint32
l2-transport	leaf	boolean
bandwidth	leaf	uint32
pg-node-locations	container	
pg-node-location[pg-node-name]	list	
pg-node-name	leaf	xr:Pg-node-id
interfaces	container	
interface[interface-name]	list	

- スキーマがある・・・型チェックが入る
- スキーマに合致しない値は事前に落とせる
  - 例えばBGPのAS番号
  - 2バイトで入れる？4バイトで入れる？
  - 記載方法はASDOT？ASPLAIN？
    - ASDOT: **1.34464**
    - ASPLAIN **100000**
- こう言うエラーチェックの有無が安全性を左右する・・・属人化したら負け

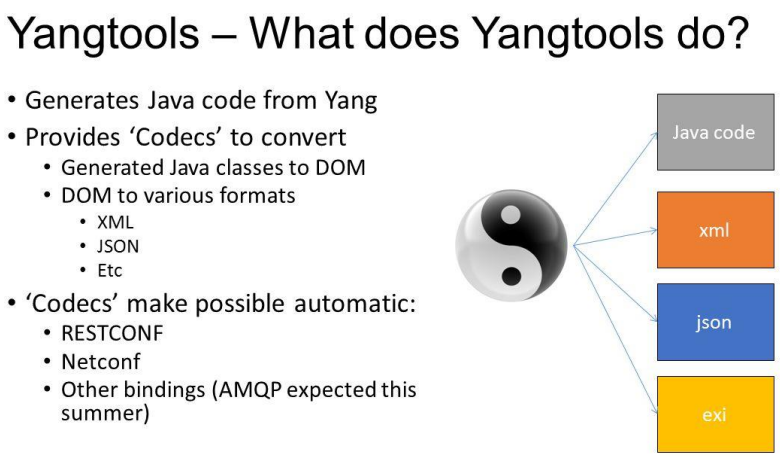
ネットワーク装置のORMはYANGから自動生成可能

変換エンジンが正しければ)バグの混入なし、引き継ぎ不要データ構造、エンコード、トランスポートのコードを分離可能

- 好きなエンコードに変換可能(JSON、XML、etc…)
- 好きなトランスポートで転送する(SSH、HTTP、Google Protocol Buffer)

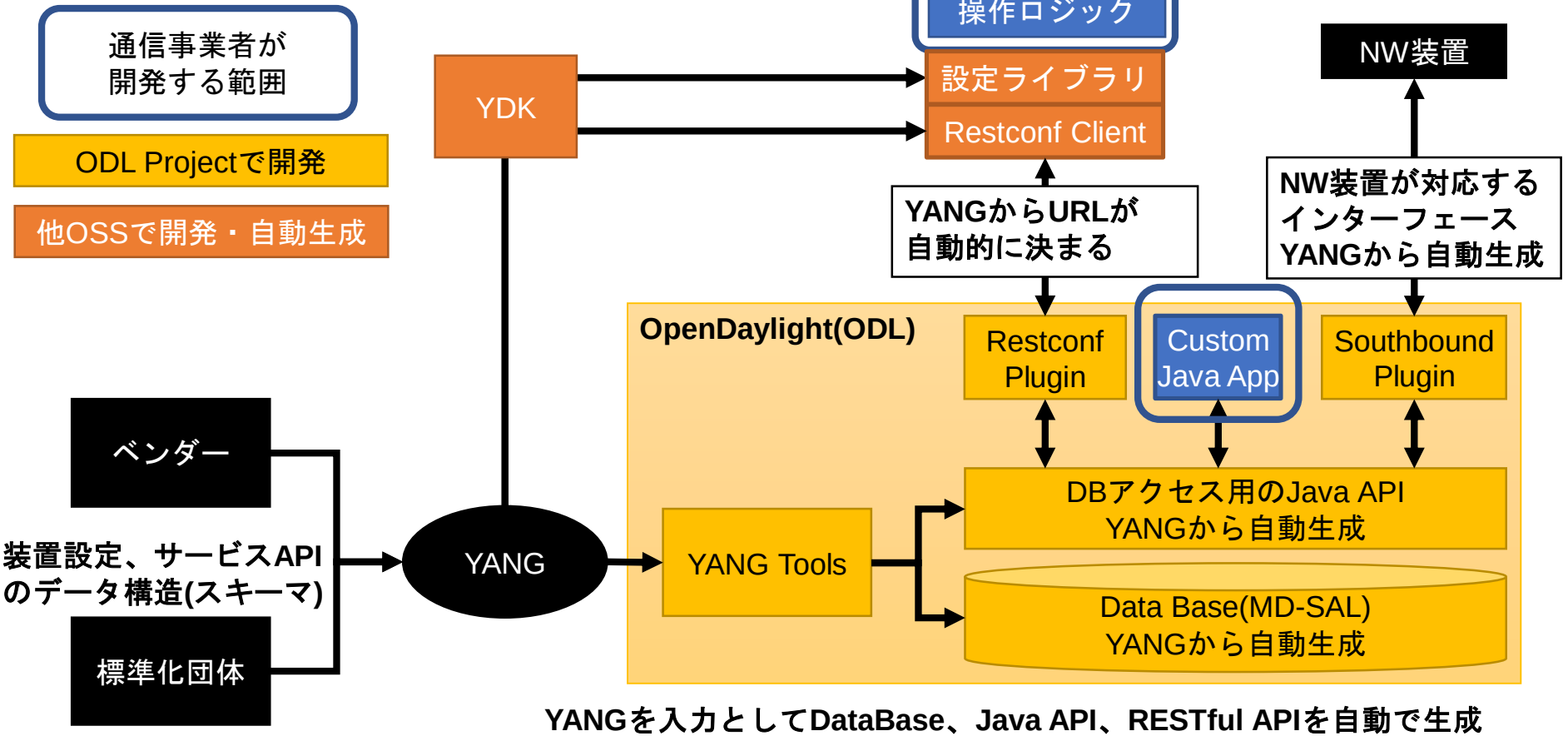


Infrastructure as a Code Using YANG, OpenConfig and YDK
https://www.youtube.com/watch?v=G1b6vJW1R5w



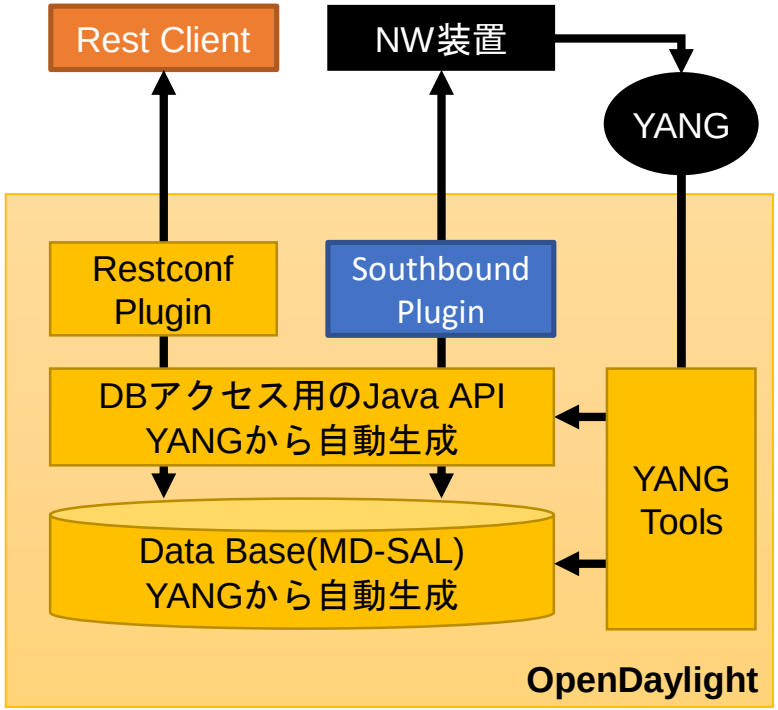
OpenDaylight Architecture
https://slideplayer.com/slide/5876657/

整理すると・・・


通信事業者が
開発する範囲
ODL Projectで開発
他OSSで開発・自動生成
YDK
操作ロジック
設定ライブラリ
Restconf Client
NW装置
YANGからURLが
自動的に決まる
NW装置が対応する
インターフェース
YANGから自動生成
OpenDaylight(ODL)
Restconf
Plugin
Custom
Java App
Southbound
Plugin
ベンダー
装置設定、サービスAPI
のデータ構造(スキーマ)
標準化団体
YANG
YANG Tools
DBアクセス用のJava API
YANGから自動生成
Data Base(MD-SAL)
YANGから自動生成
YANGを入力としてDataBase、Java API、RESTful APIを自動で生成

レガシー装置も簡単

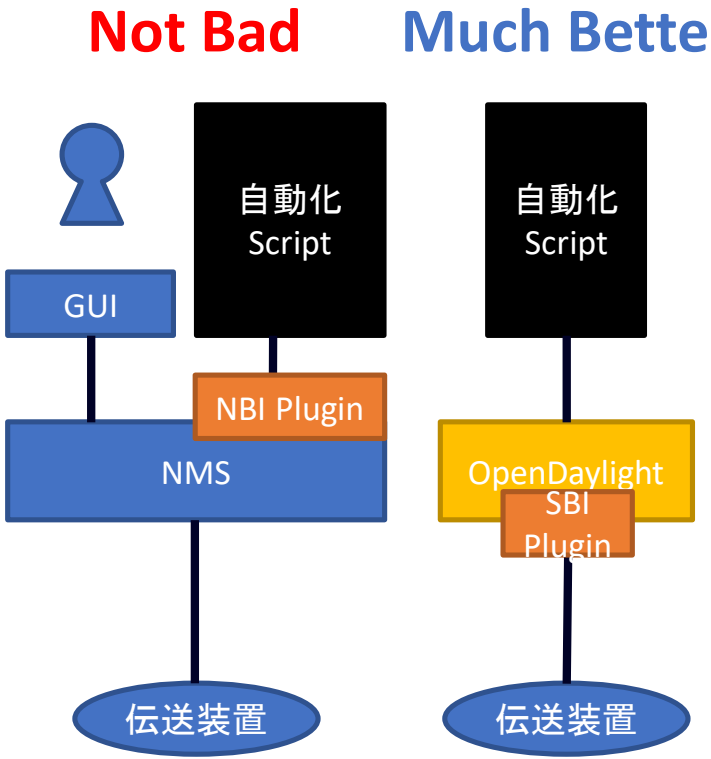
CLIだけ対応装置、非CLI対応装置もYANGで違いを吸収



- まずは後付でYANGを定義する
  - 値を入れる器を用意してやる
  - 当然、肩の定義は必要
  - OpenConfigなんかで代用もあり
- 定義したYANGスキーマをインポート
  - スキーマに合致したDBとアクセス用APIがYANGから自動生成
  - この時点でRestconfでNBIアクセス可能に
- Southbound Pluginを実装する
  - トランスポートをプロトコルに合わせて実装する
  - 構造化されていないテキストからYANGスキーマに合致したデータ構造に変換を実装する
  - 正規表現の処理をココに局所化する

レガシー装置対応の例

諦め掛けていた伝送系のレガシー装置も対応可能



- とある伝送装置の話・・・
  - NMSとセットの製品
  - SBIのトランスポートはITU-T勧告ベース
  - 装置の設定データベースはSQL実装
- あれ、思ったより簡単だった・・・
  - SQLスキーマからYANGスキーマに変換
  - YANGをOpenDaylightにインポート
  - トランスポートを実装
  - 以上、終了・・・
- NMSにNorthbound APIを実装より楽！

まとめ

OpenDaylightは実際の商用現場でも使える！

- 通信事業者における自動化の課題と対策
  - 課題は「事故の防止」、「コードの引継ぎ」、「レガシー対応」
  - ソリューションはWebフレームワークの考え方
- コアになるオープンソース技術
  - YANG
  - OpenDaylight
  - YDK
- 実際にやってみて
  - 人がコードを書く範囲は最早あまり多くない
  - YANG対応のネットワーク装置は最高に便利

ご静聴ありがとうございました