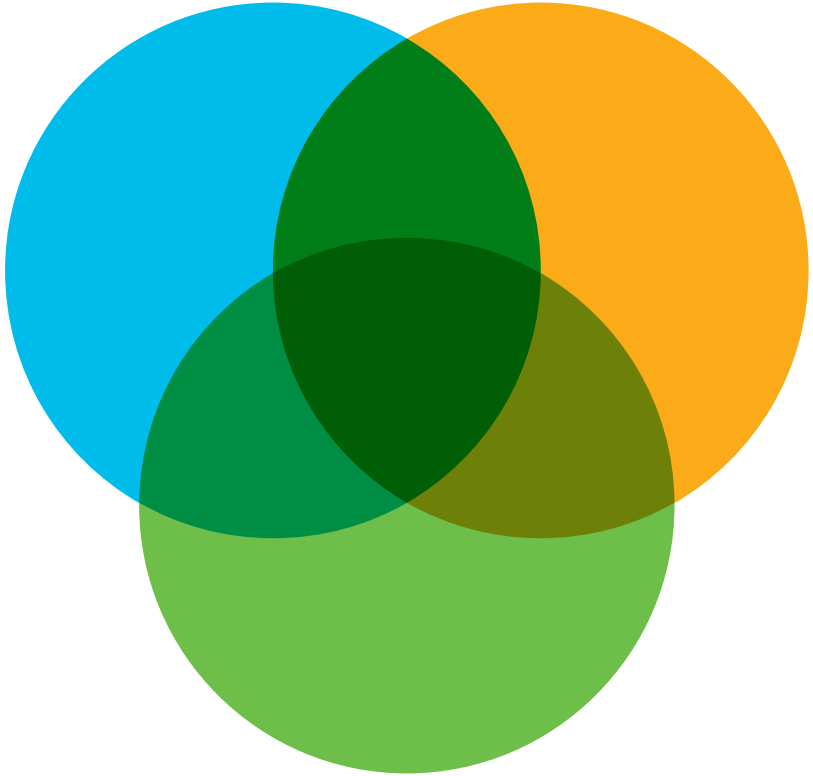
Cisco Network Services Orchestrator Workshop

Feb 2019

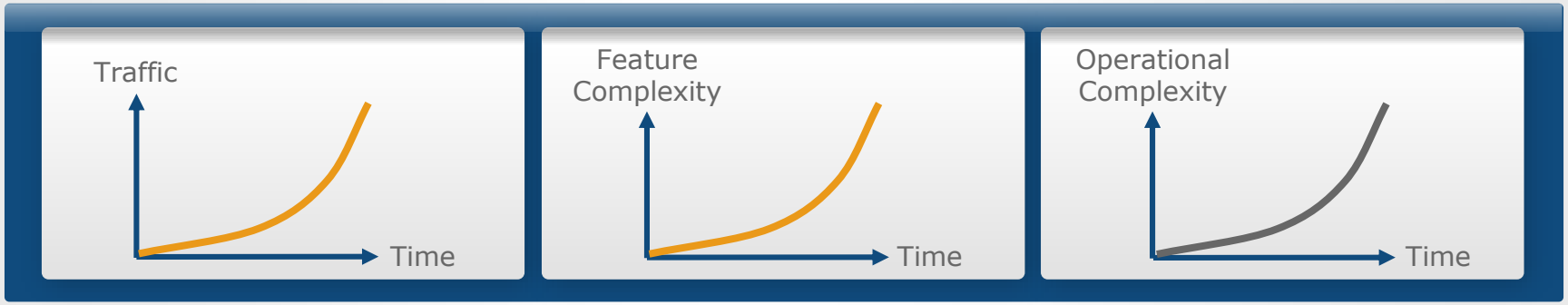
Topics

1. Traditional Provisioning Challenges
2. Cisco NSO Overview
3. NSO Demo

Traditional Provisioning Challenges



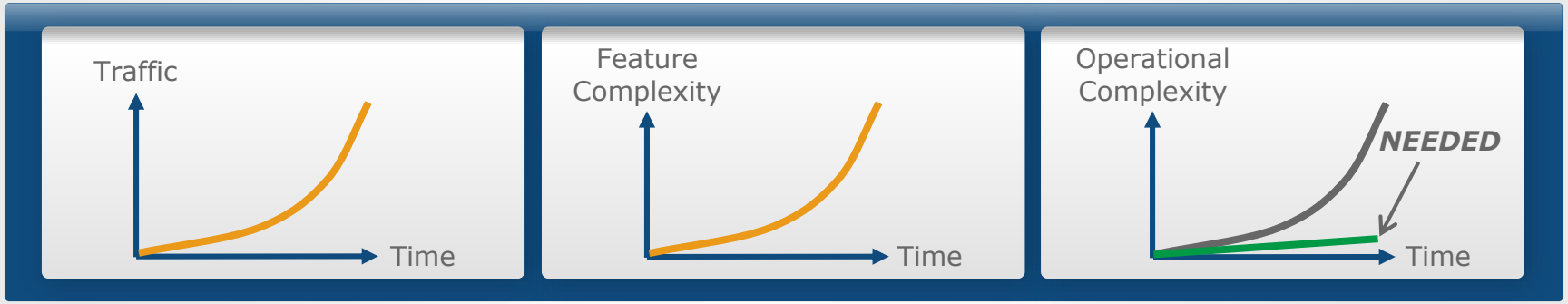
Operational Complexity Barrier



Why?

- Manual and error-prone processes
- Multi-vendor networks with stove-pipe solutions
- Closed OSS solutions result in vendor lock-in

Operational Complexity Barrier



How ?

- Give power tools (with built-in guard rails) to senior network engineers
- Make it easy to implement reliable automation services
- Make sure OSS solutions can handle new or updated services and network equipment from multiple vendors

Transition Towards Automation

Network Engineer "Automation"	Ops and Provisioning Team "Customer Experience"	Service Developers "Time-to-Market"
Day-to-day management of rapidly growing, complex networks	Provisions services and manages service quality in networks	Develops new network services on demand
Challenges • Error-prone manual tasks • Growing backlog • Virtualization is coming	Challenges • No service insight • Lack of automation • Quality issues in delivery	Challenges • Implementation time • Cost of change • Lack of tooling
Network API Utilize a single interface to all network devices 1	Service Abstraction Leverage one central API for all services 2	Transformation Develop your own services 3

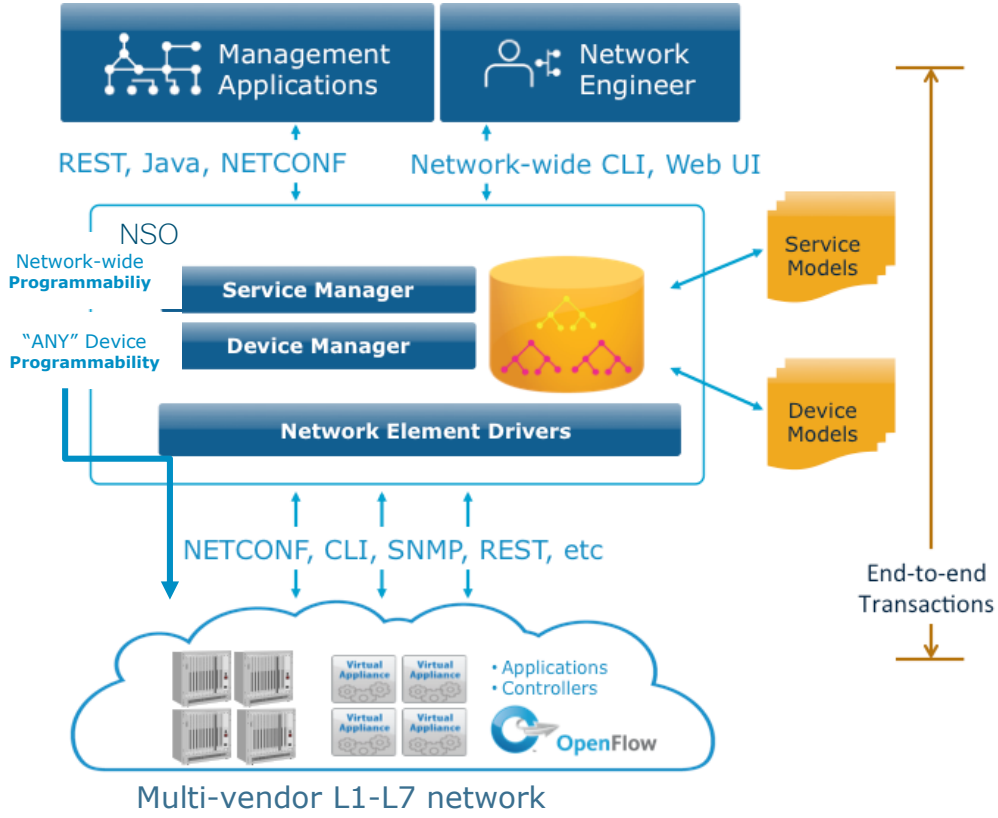
How Cisco NSO Solves these Challenges ?

Cisco Network Services Orchestrator Overview

Cisco Network Services Orchestrator

Multi-Vendor Service Orchestration & Network automation Solution for today's networks and NFV/SDN

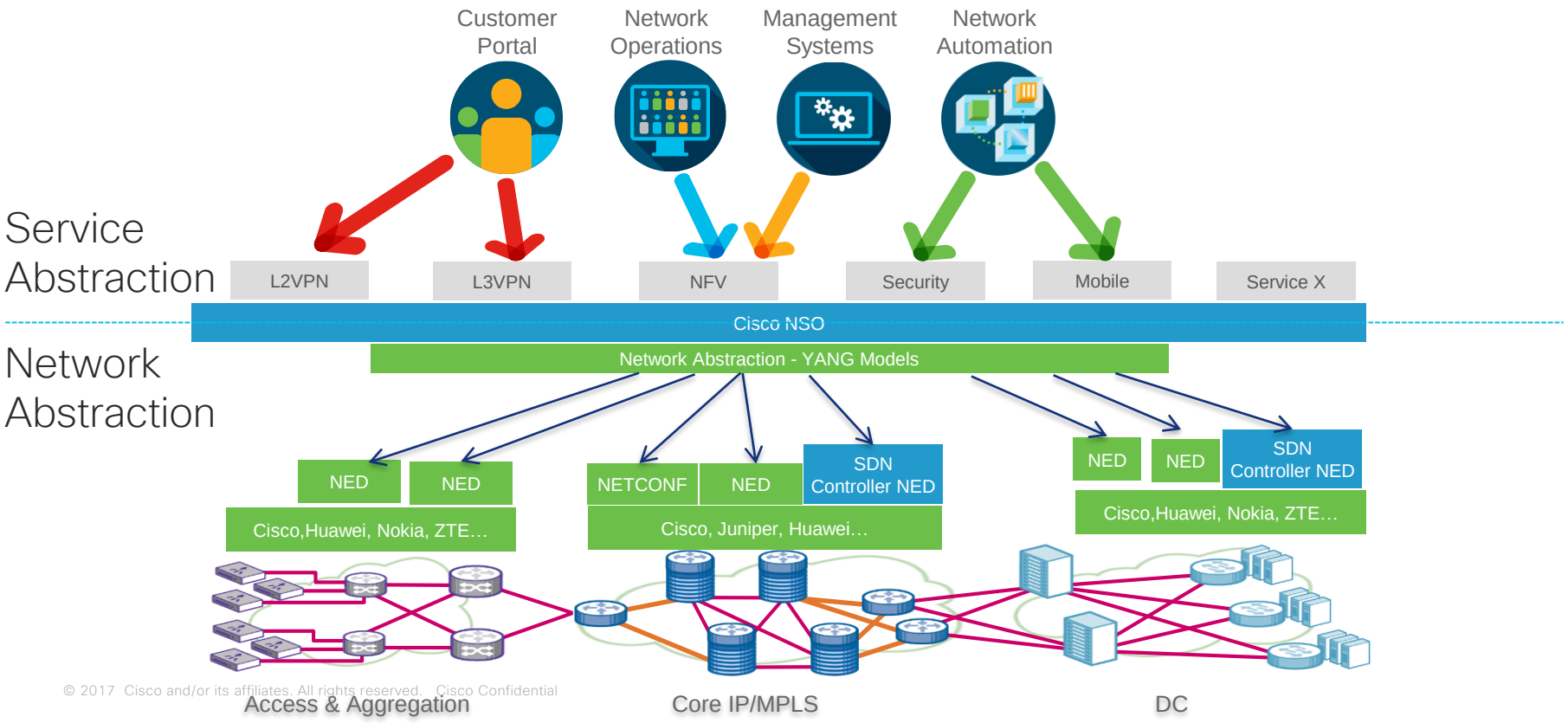
Cisco Network Services Orchestrator (NSO)



- Model based Multi-vendor , Multi-Domain Service Orchestration & Network Automation solution for existing & future (SDN/NFV) networks
- Single Pane of Glass for:
  - L1-L7 networking
  - Hardware Devices
  - Virtual Appliances
- NSO provides abstractions based on
  - Standard Data models (YANG RFC 6020) for devices & services
  - Transaction : ensures fail-safe operations & network configuration accuracy

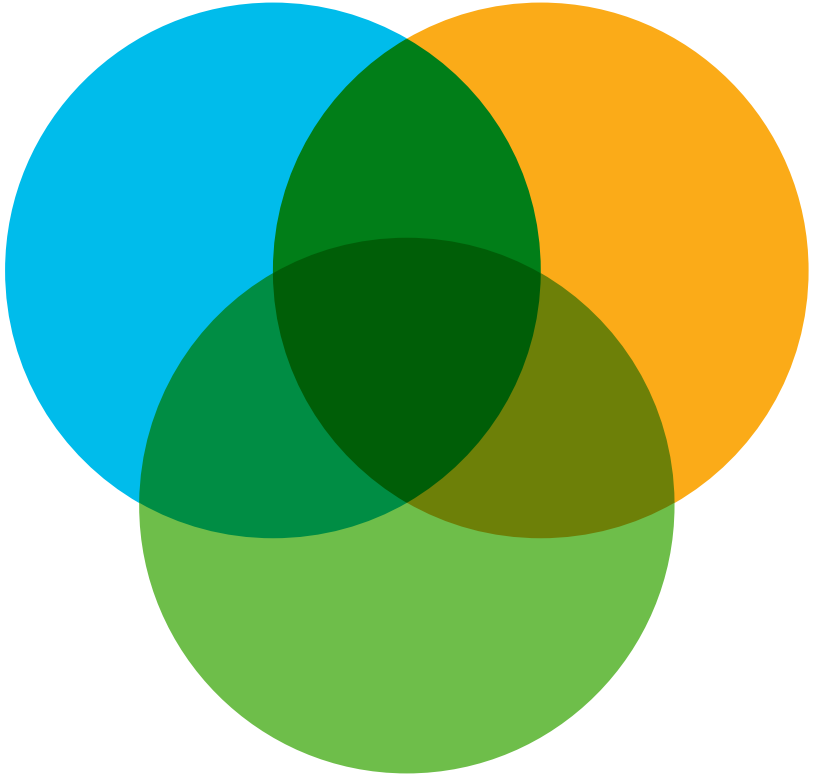
NSO Model Based Architecture

Cisco NSO Orchestration Platform Architecture



STEP 0

History of Network Programmability

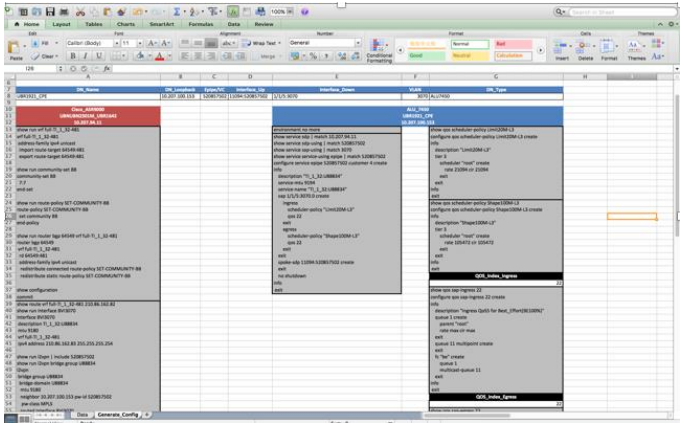


Today's Service Automation Solutions
CLI Script/Templates

Manual Provisioning

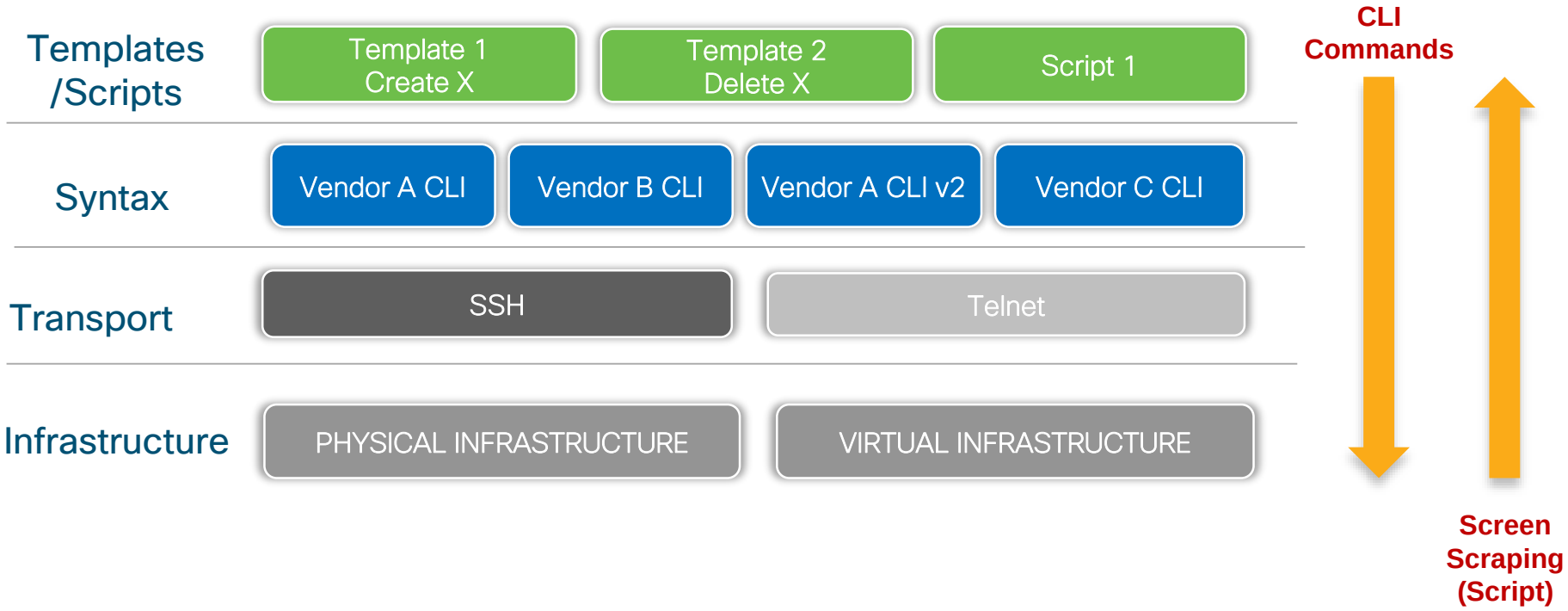
- Manual – Complex and Time consuming
- Error Prone
- Multi vendor skillsets
- Complexity barrier from handling failure scenarios
- No standardization

Template Based Traditional Provisioning



- Semi-Manual
- Limited Transactional Capability
- Complexity barrier from handling failure scenarios
- Service Change Complexities
- Multi vendor skillsets
- Human Errors

Command Line Interface (CLI) Scripts/Templates



Traditional CLI Provisioning Challenges

- What happens when one of the NE configuration fails ?
- What happens when you want to just change one or more service parameters ?
- What happens when you want to change the device with different vendor ?

Command Line Script Issues

- Lack a common data model (across platforms, even same vendor). Different CLIs
- No structured error management: Very difficult to cover all failure cases
- Lack of transaction management: Can leave network "half configured"
- Scripts fragile and costly to maintain: Each vendor, Each OS, Each OS version

"CLIs are designed to be used by humans and not an API for programmatic access."

History : SNMP

SNMP works "reasonably well for device monitoring"

RFC 3535: Overview of the 2002 IAB Network Management Workshop – 2003
https://tools.ietf.org/html/rfc3535

- Typical config: SNMPv2 read-only community strings
- Typical usage: interface statistics queries and traps
- Empirical Observation: SNMP is not used for configuration
  - Lack of Writeable MIBs
  - Security Concerns
  - Difficult to Replay/Rollback
  - Special Applications

What is Needed ?

- A programmatic interface for device configuration
- Separation of Configuration and State Data
- Ability to configure "services" NOT "devices"
- Integrated error checking and recovery

YANG / NETCONF Background

- In response to SNMP/SMI shortcomings for managing configuration eg., :
  - Lack of support simple backup and restore of configs.
  - No transactional capabilities (Single NE or multiple Nes)
  - Other inherent limitations (RFC 3535)
- NETCONF :
  - IETF configuration Management Protocol (Similar to SNMP)
  - RFC 4741/ RFC 6241
- YANG :
  - Human readable Data modelling language (Similar to MIB)
  - Defined in RFC 6020

YANG / NETCONF Vs SNMP

YANG / NETCONF

- Device Configuration (models). YANG
- NETCONF / RESTCONF verbs (commands)
  - Read device config
  - Write device config
  - Get events

<get-config>
<edit-config>
<delete-config>
<lock>
<unlock>
<close-session>
<kill-session>
<commit>
<discard-changes>

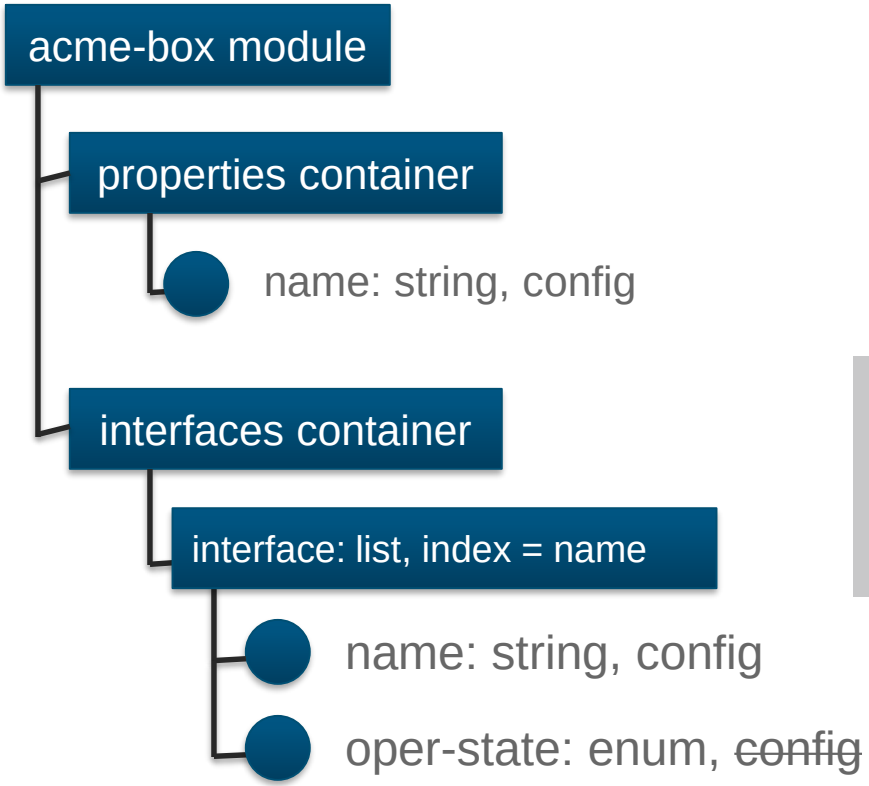
SNMP

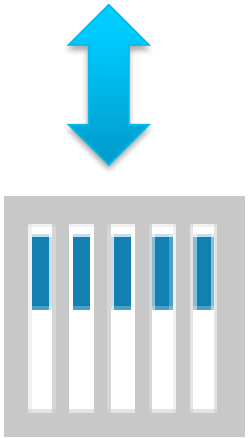
- Device Configuration (MIBs). SNMP-SMI
- SNMP verbs (commands).
  - Read device config (Get)
  - Write device config (Set) – weak
  - Get events (traps)

* Comparison for simpler understanding.

What is YANG ?

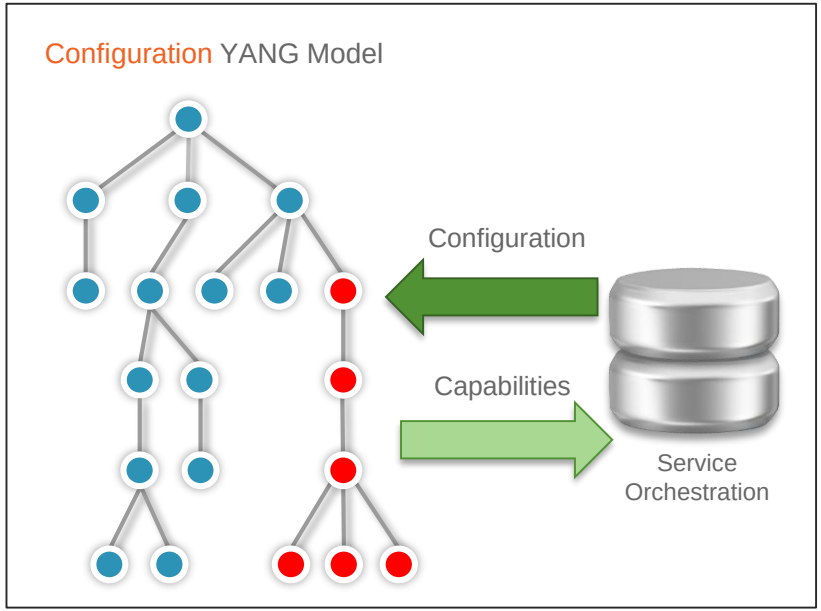
- Data modeling language
  - Configuration data
  - Operational data
- Tree structure
- Data and Types

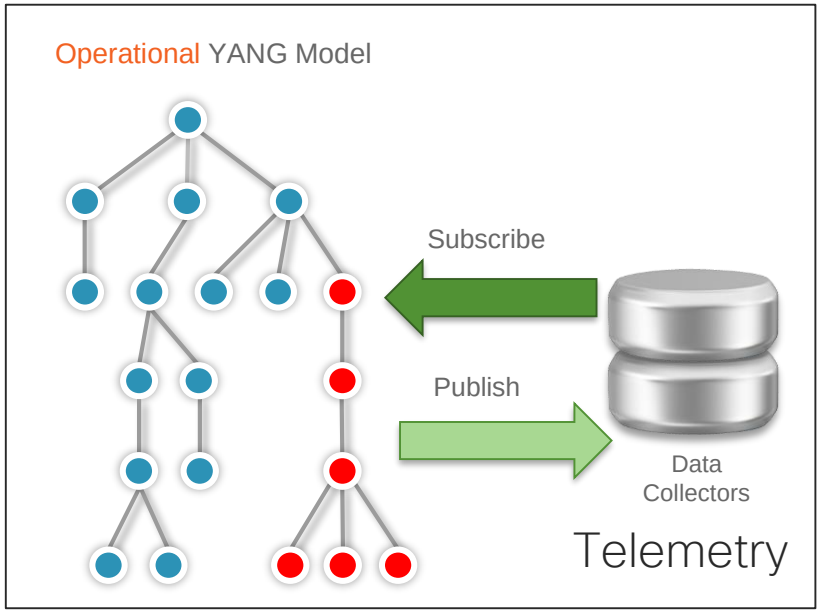




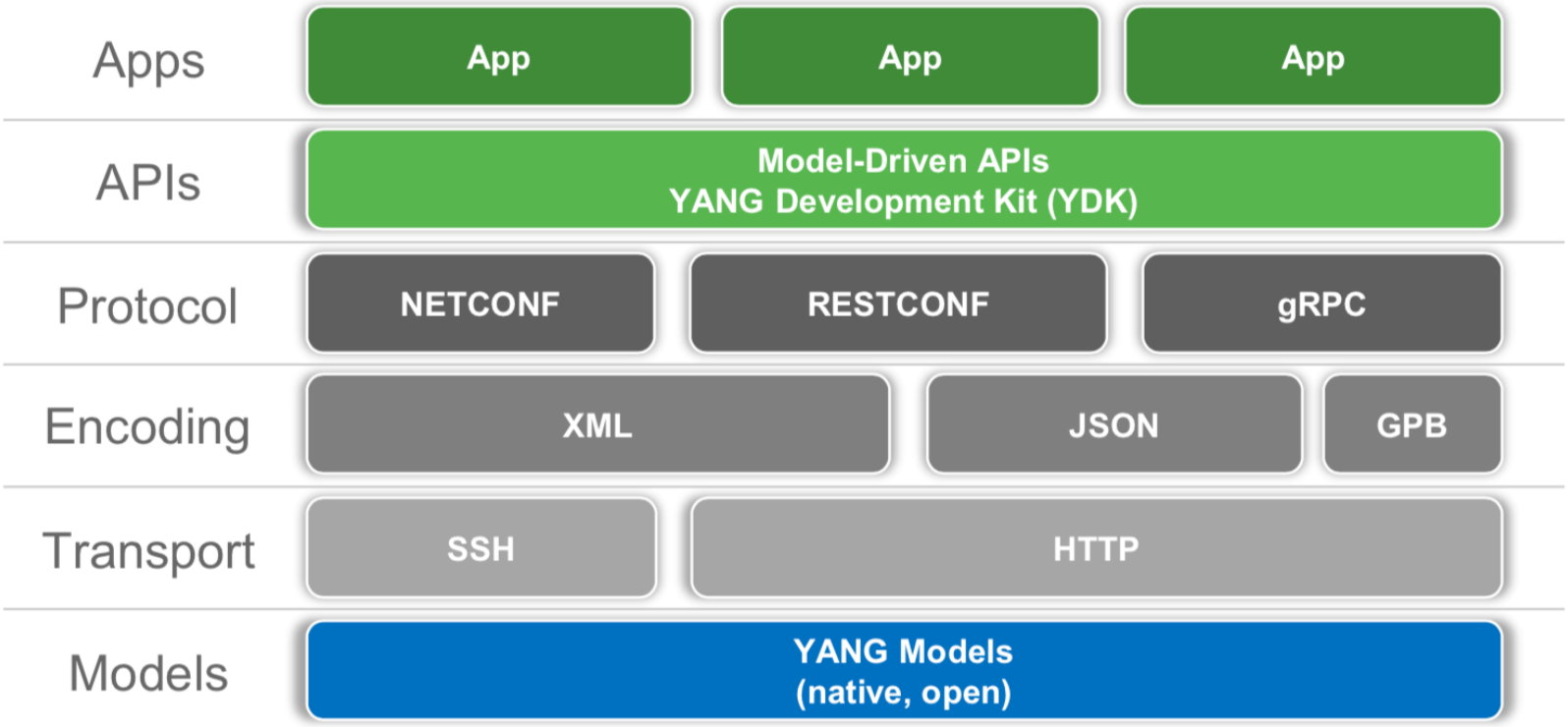
YANG Models for Networking

- “Yet Another Next Generation” modeling language: RFC 6020





Model Driven Protocol Stack



YANG Data Model vs Data Instances

Data Model (YANG)

```
container community-sets {
        description “Container for community sets";
        list community-set {
          key community-set-name;
          description "Definitions for community sets";
          leaf community-set-name {
            type string;
            description "name of the community set";
          }
          leaf-list community-member {
            type string {
              pattern '([0-9]+:[0-9]+)';
            }
            description "members of the community set";
          }
```

Defines

Defines

Data Model specifies the **Definition** of the Data Eg., Its structure , Type of data it can contain etc.,

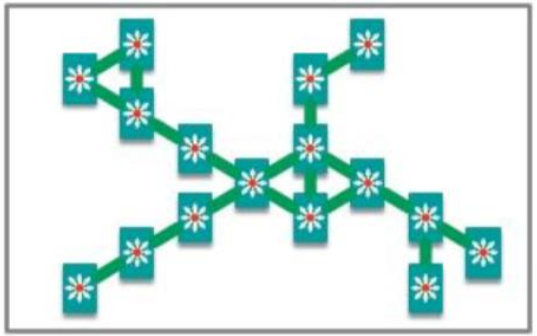
Data Instances : XML Format

```xml
<community-sets>
  <community-set>
    <community-set-name>C-SET1</community-set-name>
    <community-member>65172:1</community-member>
    <community-member>65172:2</community-member>
    <community-member>65172:3</community-member>
  </community-set>
  <community-set>
    <community-set-name>C-SET10</community-set-name>
    <community-member>65172:10</community-member>
    <community-member>65172:20</community-member>
    <community-member>65172:30</community-member>
  </community-set>
</community-sets>
```

Data Instances : JSON Format

```
{   "community-sets": {
        "community-set": [
            {   "community-set-name": "CSET1",
                "community-member": [
                    "65172:1",
                    "65172:2",
                    "65172:3" ]
            },
            {   "community-set-name": "CSET10",
                "community-member": [
                    "65172:10",
                    "65172:20",
                    "65172:30" ]
            }
        ]
    }
}
```

What can a YANG Model Describe ?



Device Data Models

- Interface
- VLAN
- ACL
- OSPF
- Etc.,

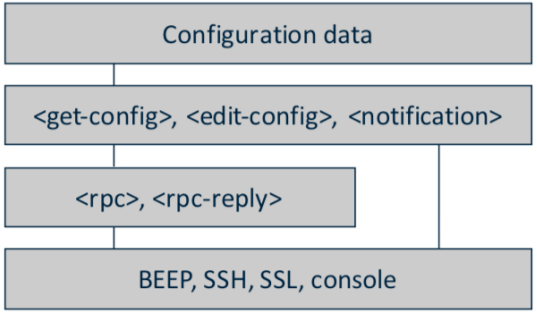
Vendor Provided/Cisco

Service Data Models

- L2 VPN - VLL
- L3VPN
- MP-BGP
- Security
- Etc.,

NETCONF

- NETCONF is an IETF network management protocol designed to support management of configuration, including:
  - Distinction between configuration and state data
  - Multiple configuration data stores (candidate, running, startup)
  - Configuration change validations
  - Configuration change transactions
  - Selective data retrieval with filtering
  - Streaming and playback of event notifications
  - Extensible remote procedure call mechanism
- NETCONF server runs on networking device and client runs as part the management application.

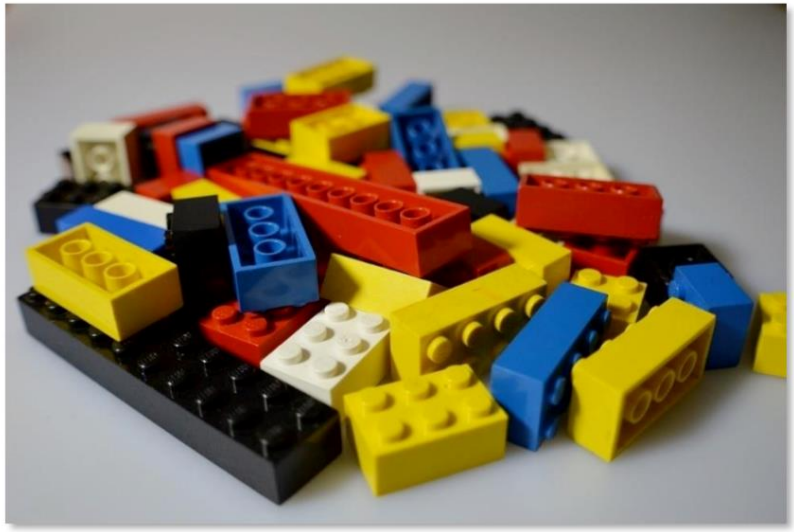


XML payload, modeled in YANG.

<get>, <get-config>, <edit-config>, <copy-config>, <delete-config>, <lock>, <unlock>, <close-session>, <kill-session>, <notification>, <commit>…

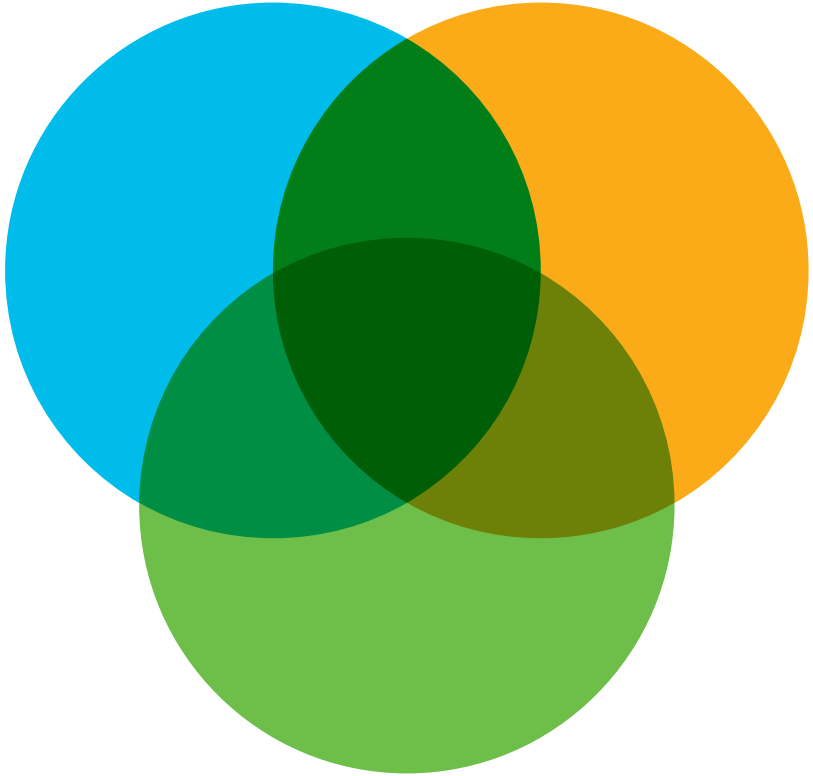
Benefits of Model Driven Programmability

- Model based, structured, computer and Human friendly
- Precise Data Definitions
- Models decoupled from transport, protocol and encoding
- Transactional Capability
- Model-driven APIs for abstraction and simplification



STEP 1

Network Abstraction



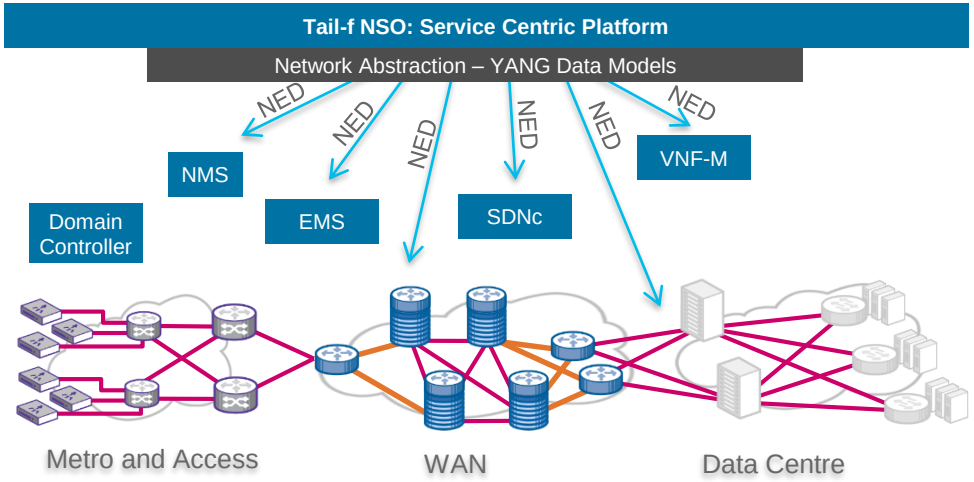
But …. Reality

Does All Network Vendors Support YANG / NETCONF today?

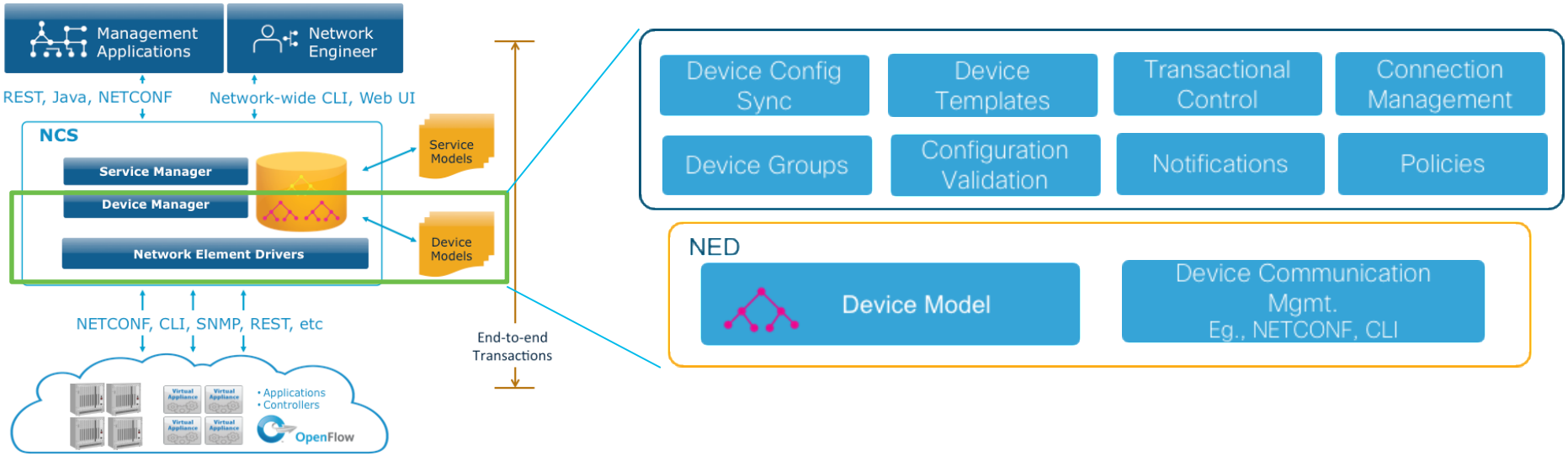
STEP1: Network Abstraction – Future proofed
Single Interface to all Network Devices

- **Precise data-model** for the entire network. YANG based device-models.
- Automatic CRUDs on network elements via NEDs - **normalised south-bound interfacing**
- Generic way of consuming the network **irrespective of technology vendor, platform, device**.

Network becomes YANG



How do we achieve Network Abstraction ?



Device Manager / Network Element Drivers (NED)

Device Manager

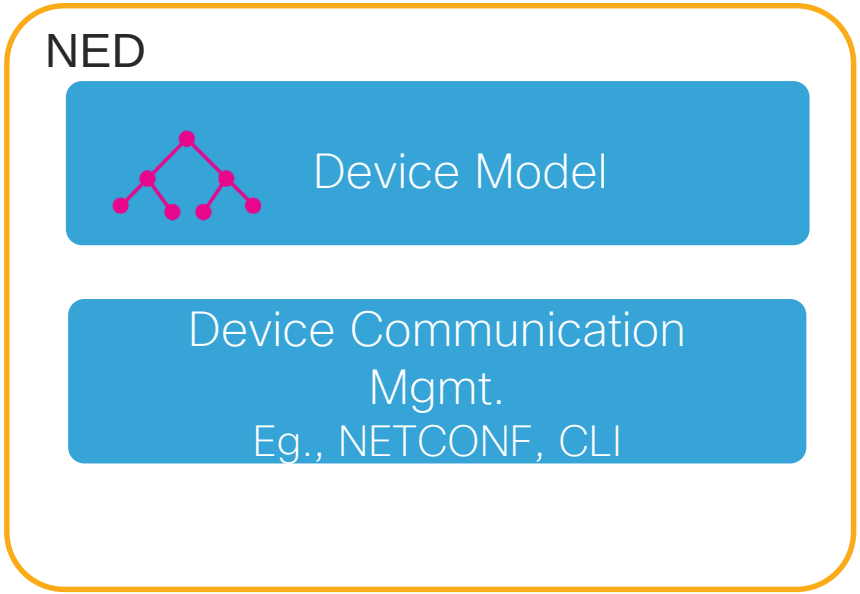
- Device Config Sync
- Device Templates
- Transactional Control
- Connection Management
- Device Groups
- Configuration Validation
- Notifications
- Policies

NED

- Device Model
- Device Communication Mgmt.
  Eg., NETCONF, CLI

Network Element Drivers(NED)

- Provided by Cisco for Model-driven device integration
- Code-free for YANG/NETCONF, SNMP, and other schema based interfaces
- Different types
  - NETCONF
  - SNMP
  - CLI
  - Generic
- Shipped as individual packages



Device Data Model

Precise YANG Definition describing the capabilities of a Device OS. It describes the Capability Structure and its content types.

Device Capabilities : VRF , ACL , QOS, VLAN etc.,

Structure : Interface can contain VLANs

Type: VLAN ID should be a number ranging 1 – 4096

Device Models (Cisco IOS)

policy-map[name]	list	
name	leaf	string
type	leaf	enumeration
protocol	leaf	enumeration
description	leaf	string
class[name]	list	
class-default	container	
class[name]	list	

Device Models (Vendor X)

qos-profile[name]	list	
name	leaf	string
car	container	
cir	leaf	uint32
pir	leaf	uint32
cbs	leaf	uint32
pbs	leaf	uint32
green	leaf	qos-car-packet
yellow	leaf	qos-car-packet
red	leaf	qos-car-packet
inbound	leaf	empty

~~Fragile Adapter~~ Network Element Driver (NED)

Declarative

- policy-map[name]
  - name
  - type
  - protocol
  - description
  - class[name]
  - class-default
    - class[name]
      - name
      - flow

YANG Model

Cisco IOS NED Engine

South Bound Protocol: CLI

Device Models (Cisco IOS)

- policy-map[name]
  - name
  - type
  - protocol
  - description
  - class[name]
  - class-default
    - class[name]
      - name
      - flow

IOS Device Command

```
! QOS Profile
policy-map {QOS_PROFILE_NAME}
 class class-default
  police {CIR_VALUE}
```

Device Models

- qos-profile[name]
  - name
  - car
    - cir
    - pir
    - cbs
    - pbs
    - green
    - yellow
    - red
    - inbound

Device Command

 # QOS Profile
qos-profile ***{QOS_PROFILE_NAME}***
car cir ***{CIR_VALUE}*** *pir* ***{PIR_VALUE}*** *cbs* ***{CBS_VALUE}*** *pbs* ***{PBS_VALUE}*** *green pass yellow pass red discard*

Multi-Vendor Support Spectrum (Sample)

Network Element Drivers

- Per Vendor OS Type – Can support multiple device types
- No NED license for full NETCONF/YANG devices
- Additional Command/API support can be added through Cisco TAC without any additional cost
- New Device NEDs in 2-6 weeks time

*NED development requires all its Pre-requisite are met

NED Summary

Provides Precise Device Data Models

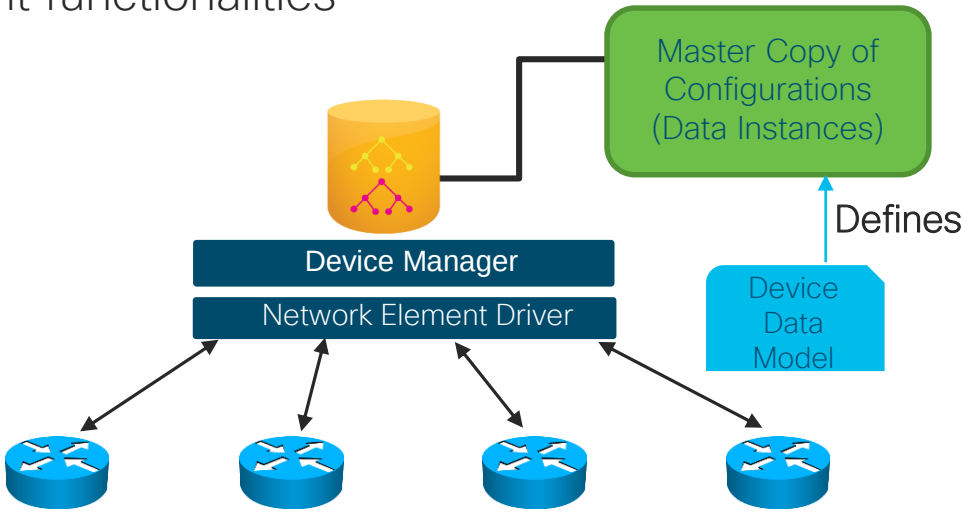
Abstracts Underlying Southbound Protocols

Cisco has Wide Spectrum of Multi–Vendor NEDs

Northbound APIs are **Auto** generated using Device Models

Device Manager and CDB

- Is the heart of NSO
- NSO keeps a master copy of configuration in CDB
- Supports various device management functionalities
  - Connection Management
  - Transaction Control
  - Device Groups
  - Device Syncing etc.,

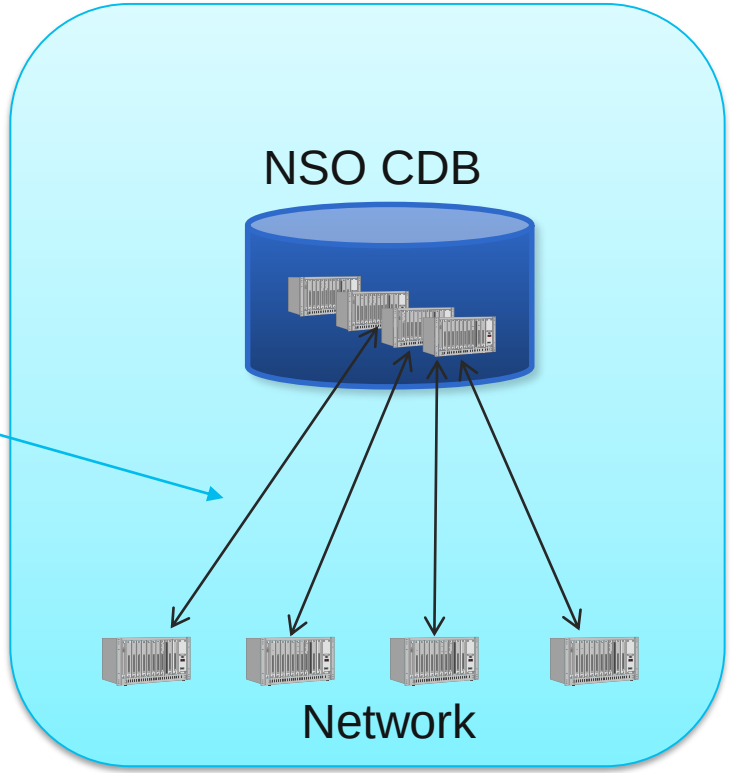


Two-way Device Config Synchronization

Accurate Provisioning is enabled in NSO through:

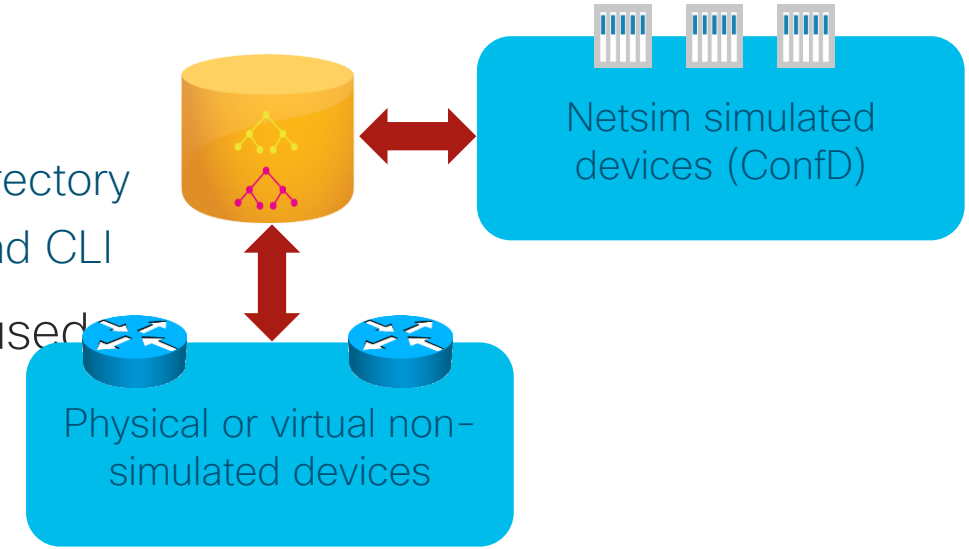
- Device-Level Data-Syncing

- sync-to
- sync-from
- check-sync
- compare-config



Netsim Overview

- **ncs-netsim** is a network devices simulation tool
- Used to test NSO with simulated devices
- Uses NED device packages
  - A NED package contains netsim directory
  - Represents device configuration and CLI
- The same YANG for models are used for simulated and real devices

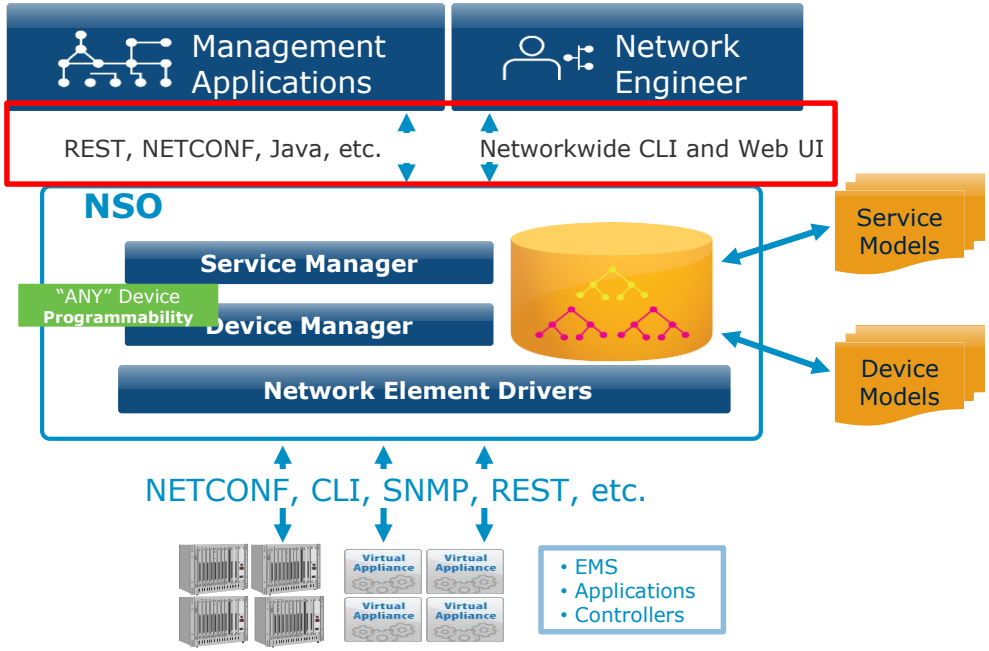


Demo 1: Device Management

- NETSIM
- NEDs
- Device Configuration
- Device Synchronization
- Rollbacks

NSO Northbound APIs – Model Driven
Any Device Programmability

- **Auto generated** based on YANG Models
- Unified CLI across **Multi-Vendor Network**
- **Auto generated Web UI**
- **Multiple types of NBI APIs** Eg., NETCONF, REST , RESTCONF, JSON RPC , Java , Python



Clients

- REST : POSTMAN
- NETCONF : MGSOFT NETCONF Browser
- NSO Auto Generated UI
- NSO Unified CLI

How does this help?

- **Precise YANG Data Models** for the entire Network. Even if the vendor does not support YANG Models. Makes the existing **network Programmable**.
- **Single interface** to all Multi-Vendor Network Devices. **Eases Automation**.
- Full Transactional Capability

Templates & Policies

Typical Usage

- Templates usually used for to generate compliance reports in NSO. Not so much for service configuration or provisioning. Provisioning best done via design and build of “Service” constructs.
- Note that Template do not have FASTMAP pattern
- Policy Rules is used to enforce organizational policies. As configuration may be correct but might violate some policy.

Device Templates

- Multi vendor device configuration
- Network (Adhoc / Non-service) Configuration Changes
- Variable substitutions
- Applied to
  - Individual devices
  - Device Groups
- Can be used in Compliance Reporting

Device Templates (New Template Creation)

```
devices template snmp-community-template
 config
  cisco-ios-xr:snmp-server community {$COMMUNITY}
   RO
  !
  junos:configuration snmp community {$COMMUNITY}
   authorization read-only
  !
  ios:snmp-server community {$COMMUNITY}
   RO
  !
 !
!
```

Compliance Reporting

- Who has done what?
- Is the network correctly configured?

* Simplified for illustration

Compliance Reporting

- Current / Historic check sync details (System wide / selective)
  - Device Sync
  - Service Sync
- Compare Templates

* Simplified for illustration

Policies (Pro-Active)

- Network wide configuration constraints
  - Warning (Overridden)
  - Errors (Can not be overridden)

* Simplified for illustration

Demo 2: Device Management

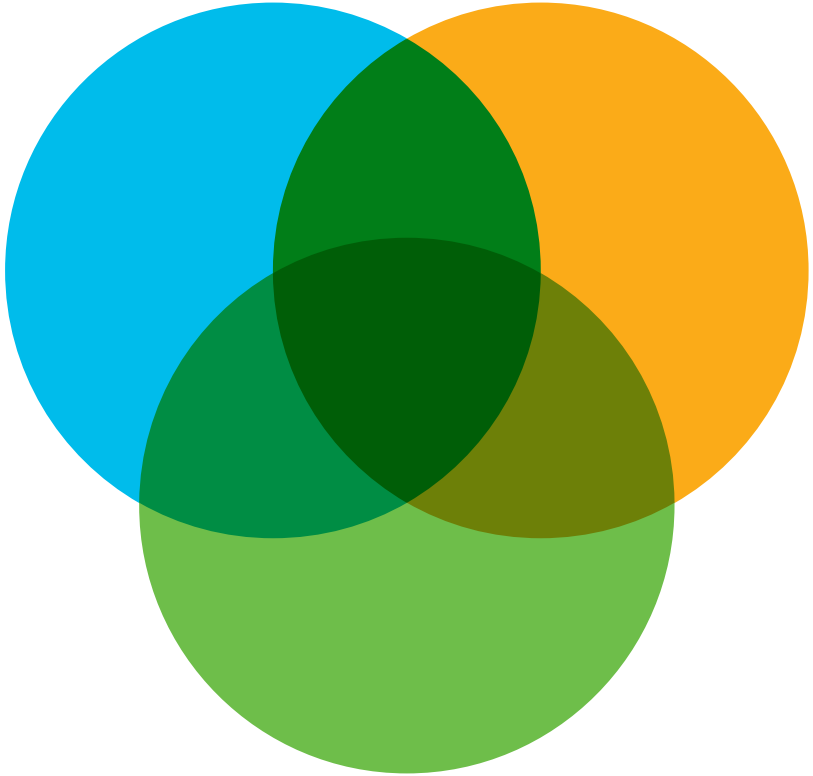
- Device Templates
- Policies
- Compliance Report
- Any Device Programmability

How does this help ?

- **Enforce network policies**
- Keep a **check on Network Consistency** state using Compliance Reports.
- **Single interface** to all Multi-Vendor Network Devices. **Eases Automation**.

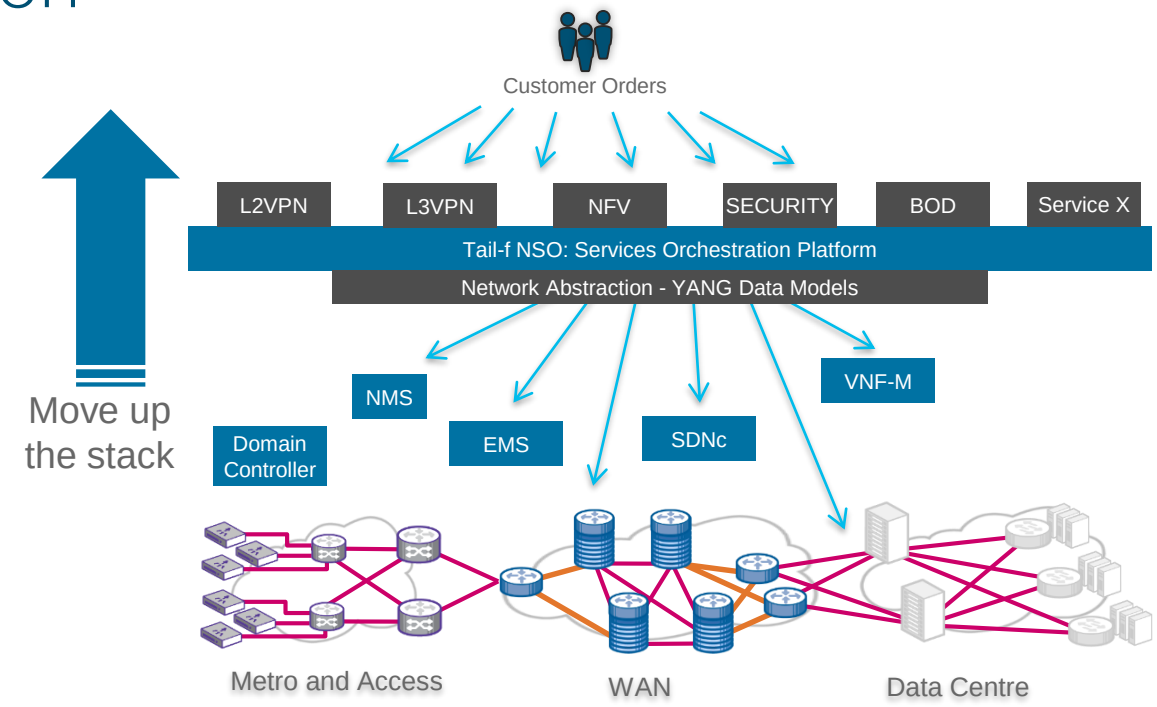
STEP 2

Service Abstraction



STEP 2: Build Decoupled Services
Service Abstraction

- Services defined in YANG.
- Services are Customer's Intellectual Property
- Loosely coupled, precise "Mapping" from Service Yang → Device[s] Yang



Definitions

Service Types:

- L2VPN, L3VPN, FW-rules, Security-policy, my-cool-service, etc.

Service Model:

- A service type's precise schema represented in YANG

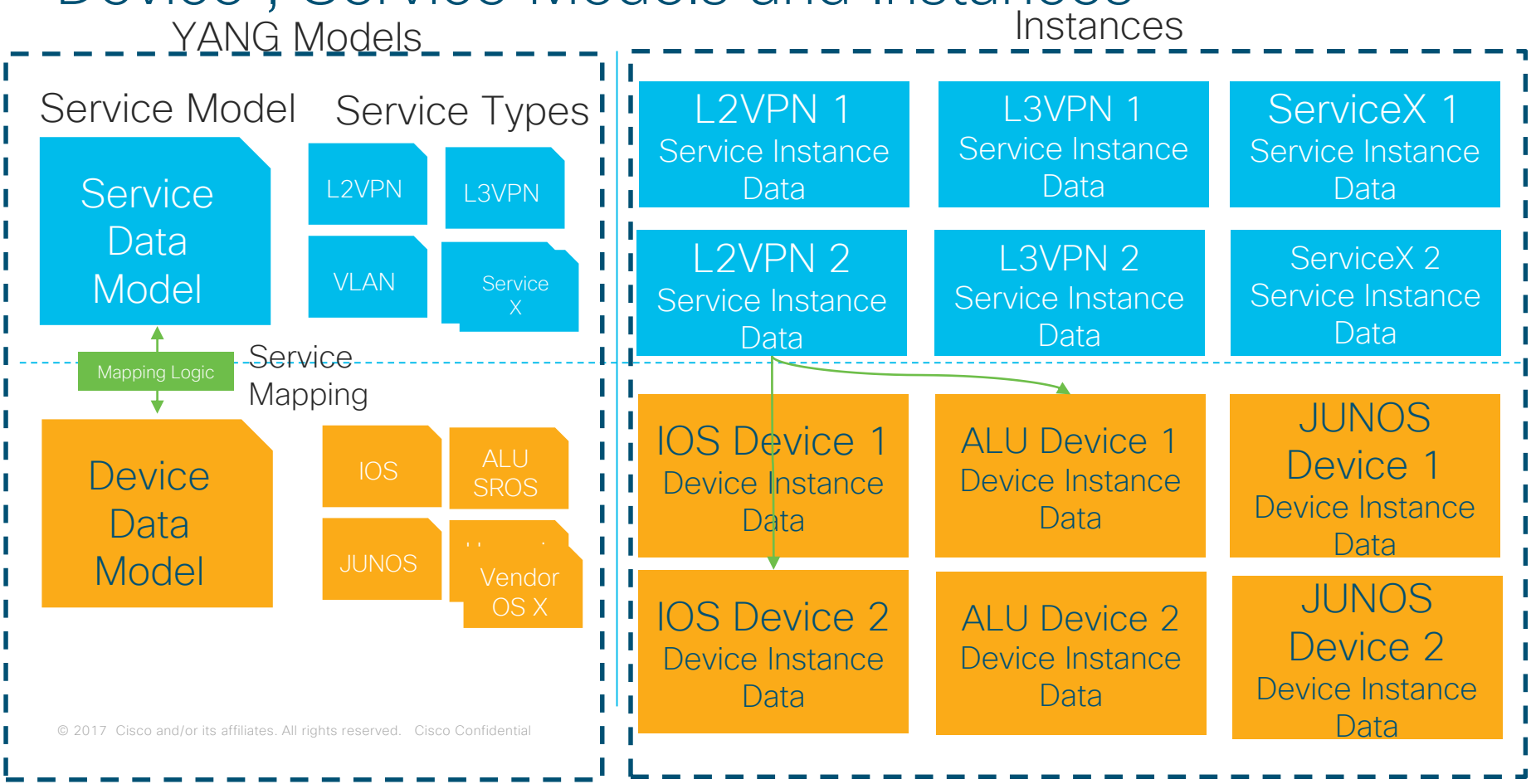
Service Creation Mapping:

- Mapping of the service-model to the network layer (i.e. device model[s])
- NSO patented FASTMAP design pattern

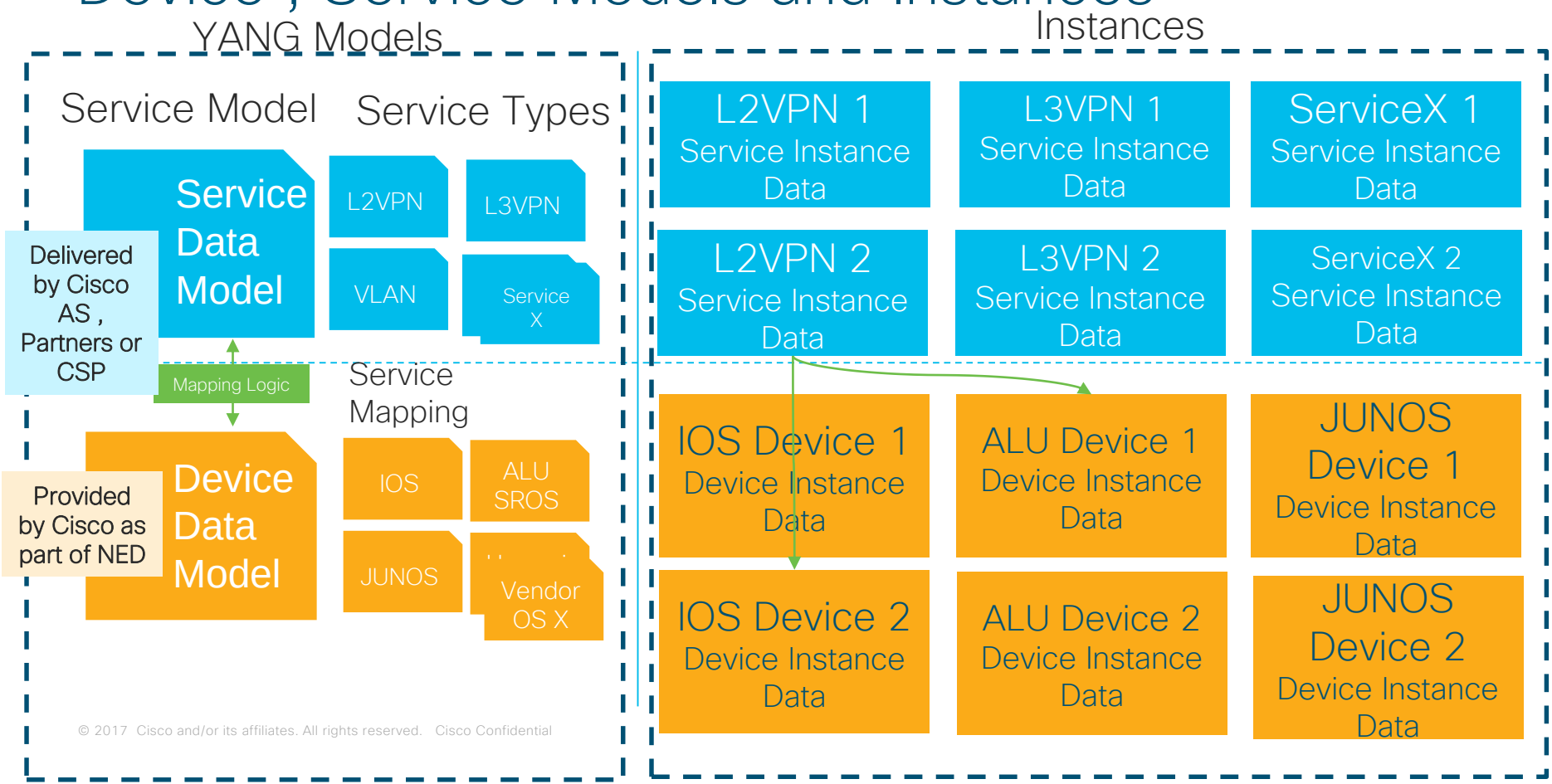
Service Instance:

- Use a service-model to create (rubber stamping) instances
- CRUD are done on service instances.
- L2VPN instances for Countrywide Hospital, Bank, etc.
- L3VPN instances for Countrywide Bank, Toyota, etc.

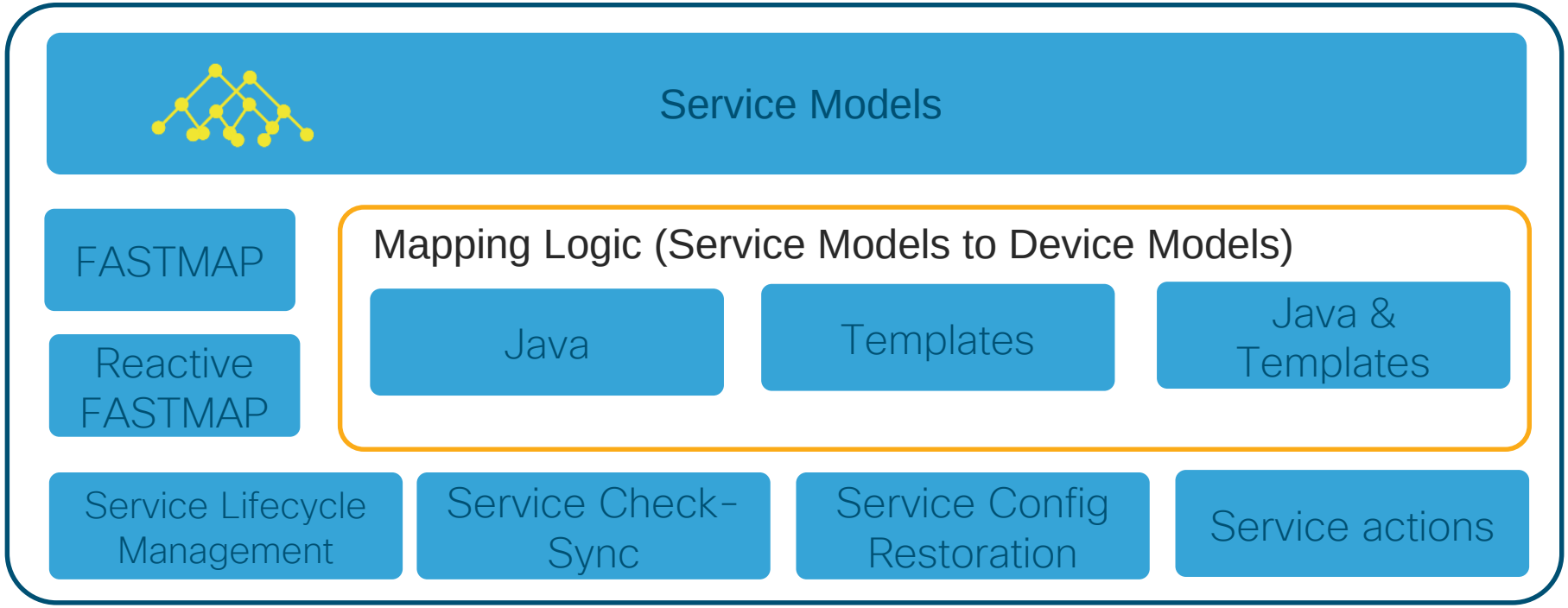
Device , Service Models and Instances



Device , Service Models and Instances

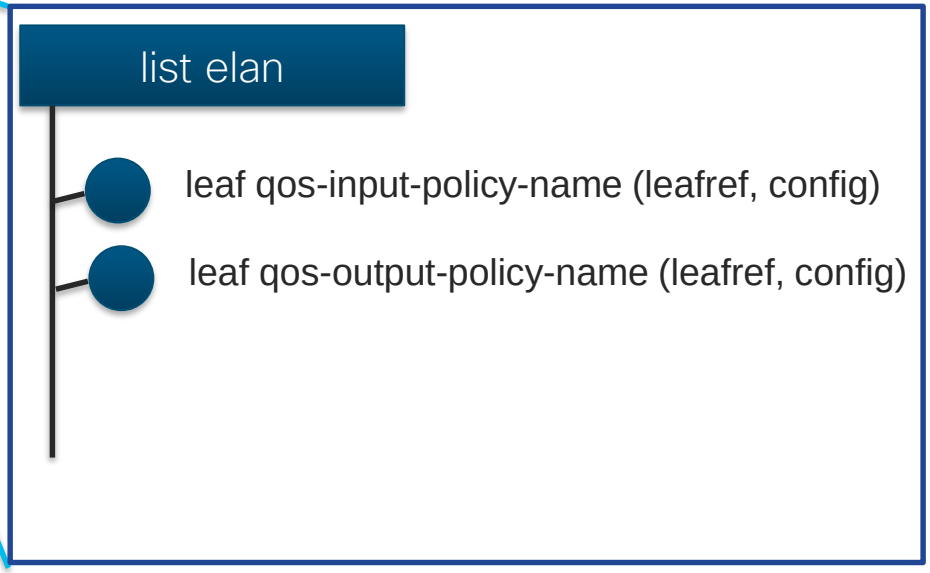


Service Manager

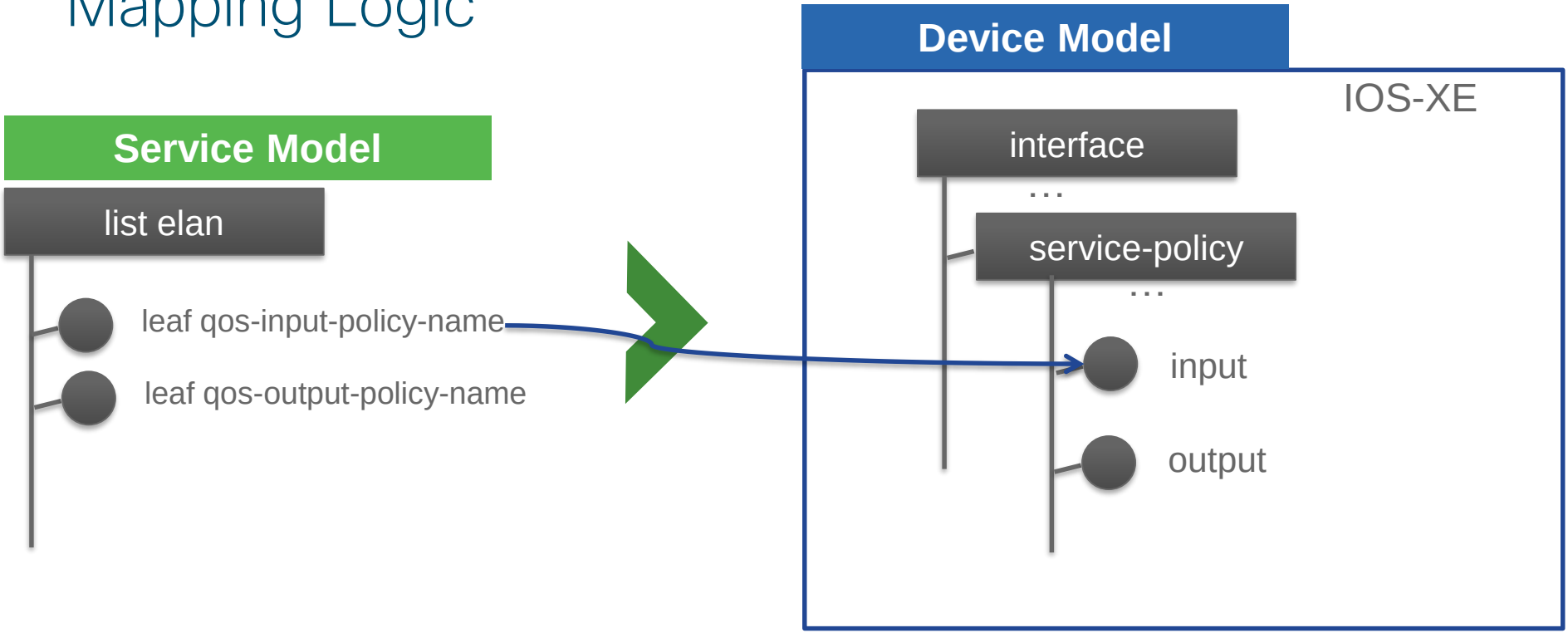


Service Model

```
      leaf qos-input-policy-name {
        tailf:info "Ingress QOS policy name ";
      type leafref {
        path "/ncs:services/qospolicy:qospolicy/qospolicy:name";
        }
      }

      leaf qos-output-policy-name {
        tailf:info "Egress QOS policy name ";
      type leafref {
        path "/ncs:services/qospolicy:qospolicy/qospolicy:name";
        }
    }

  } // End of site list

  // services / elan * / security
container security {

  // services / elan * / security / storm-control
  container storm-control {
    leaf unicast-pps {
      tailf:info "Unicast storm control suppression level in packets per second";
      type string {
          tailf:info "<0.0 - 10000000000.0>[k|m|g]";
        }
    }
    leaf broadcast-pps {
      tailf:info "Broadcast storm control suppression level in packets per second";
      type string {
          tailf:info "<0.0 - 10000000000.0>[k|m|g]";
        }
    }
    leaf multicast-pps {
```

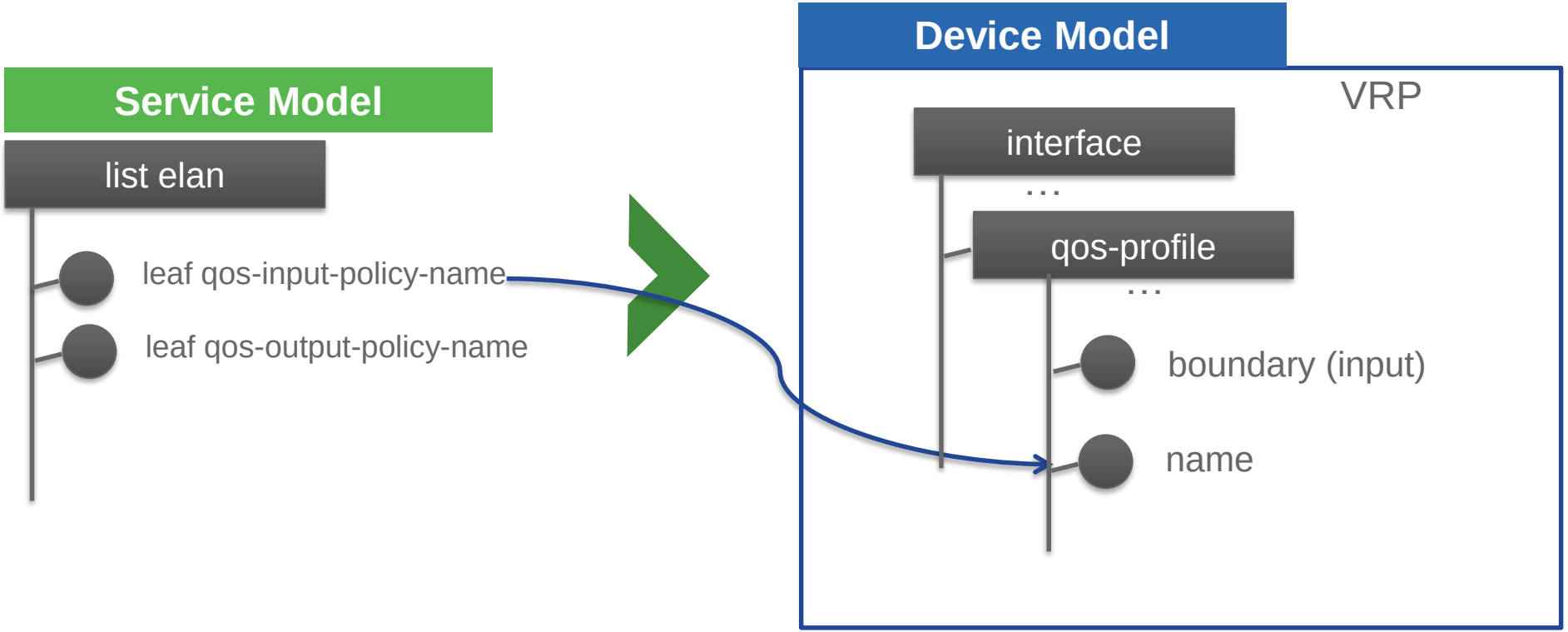


Mapping Logic

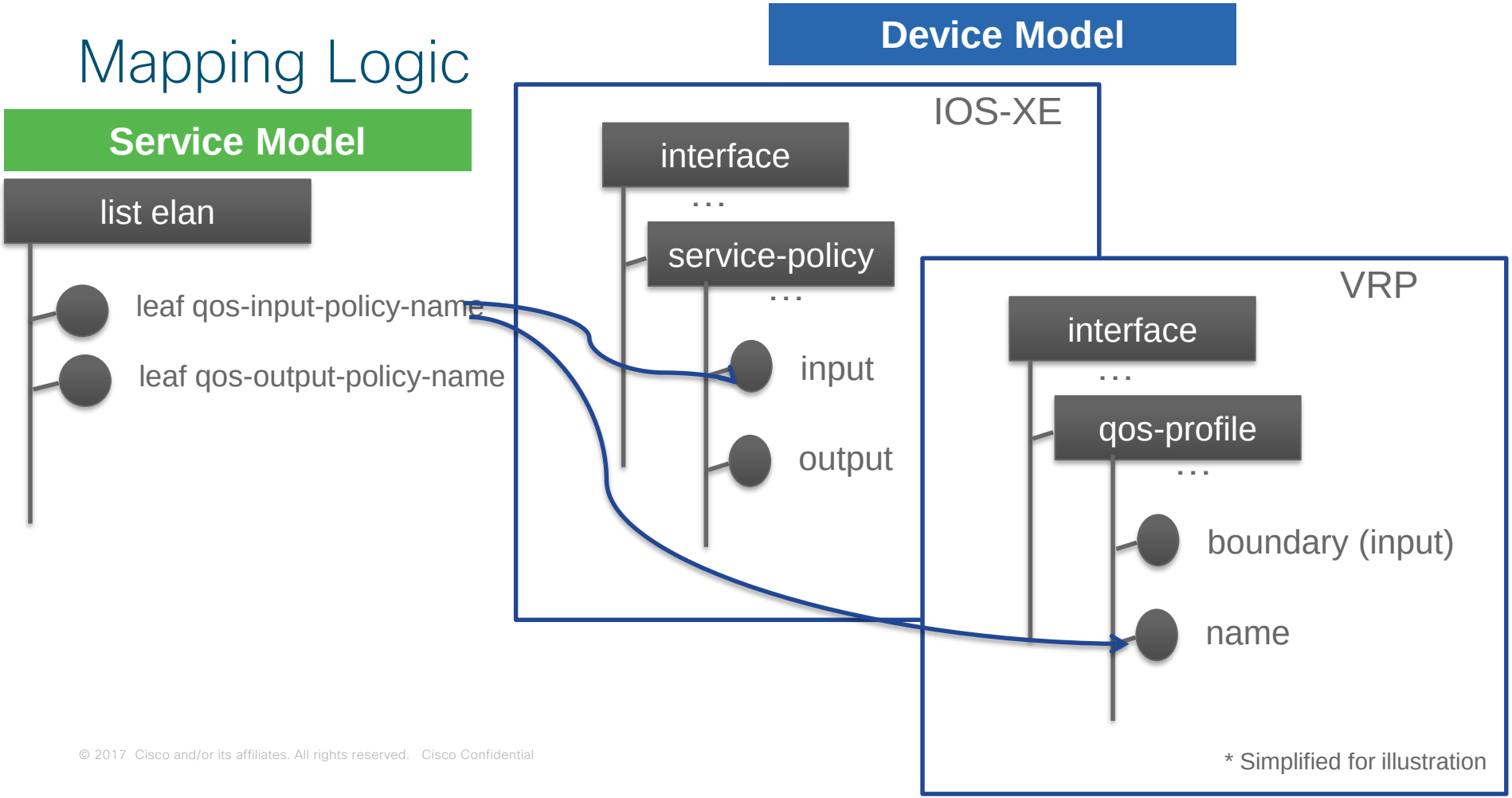


* Simplified for illustration

Mapping Logic

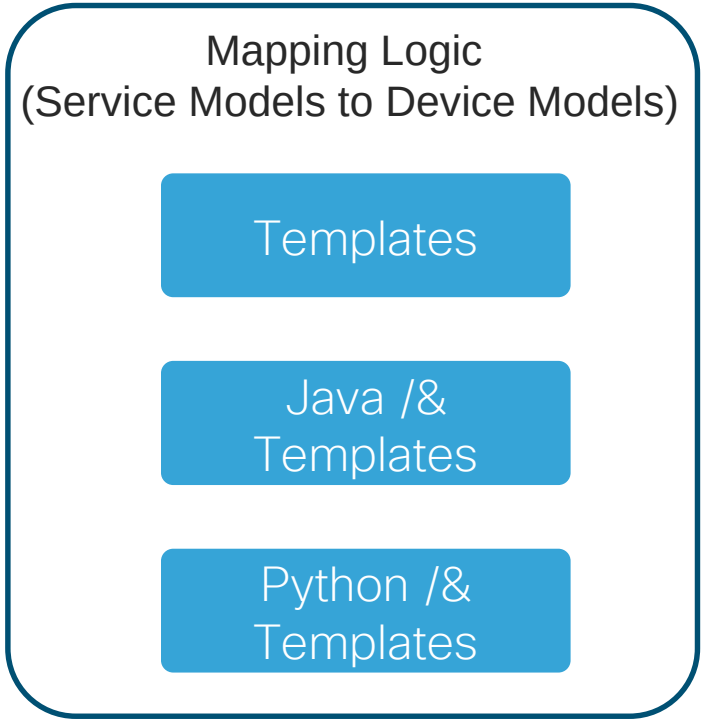


Mapping Logic

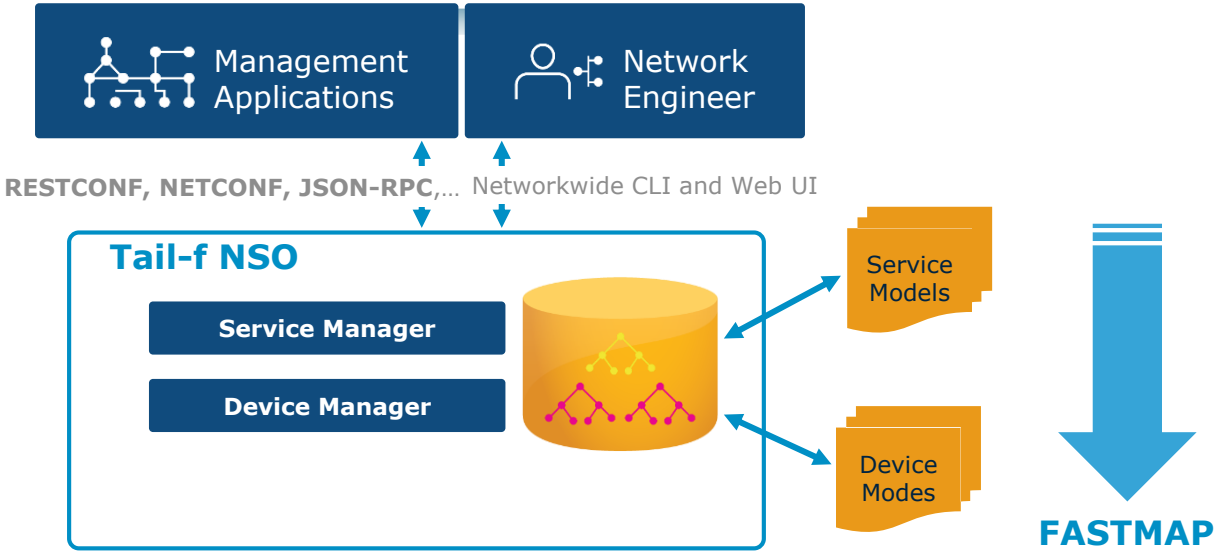


Mapping Logic

- Data-model driven model mapping
- Templates
  - Simple Parameter mappings
- Java / Python
  - Complex Logic
  - External call-outs
- Java/ Python and Templates
  - Use both technologies

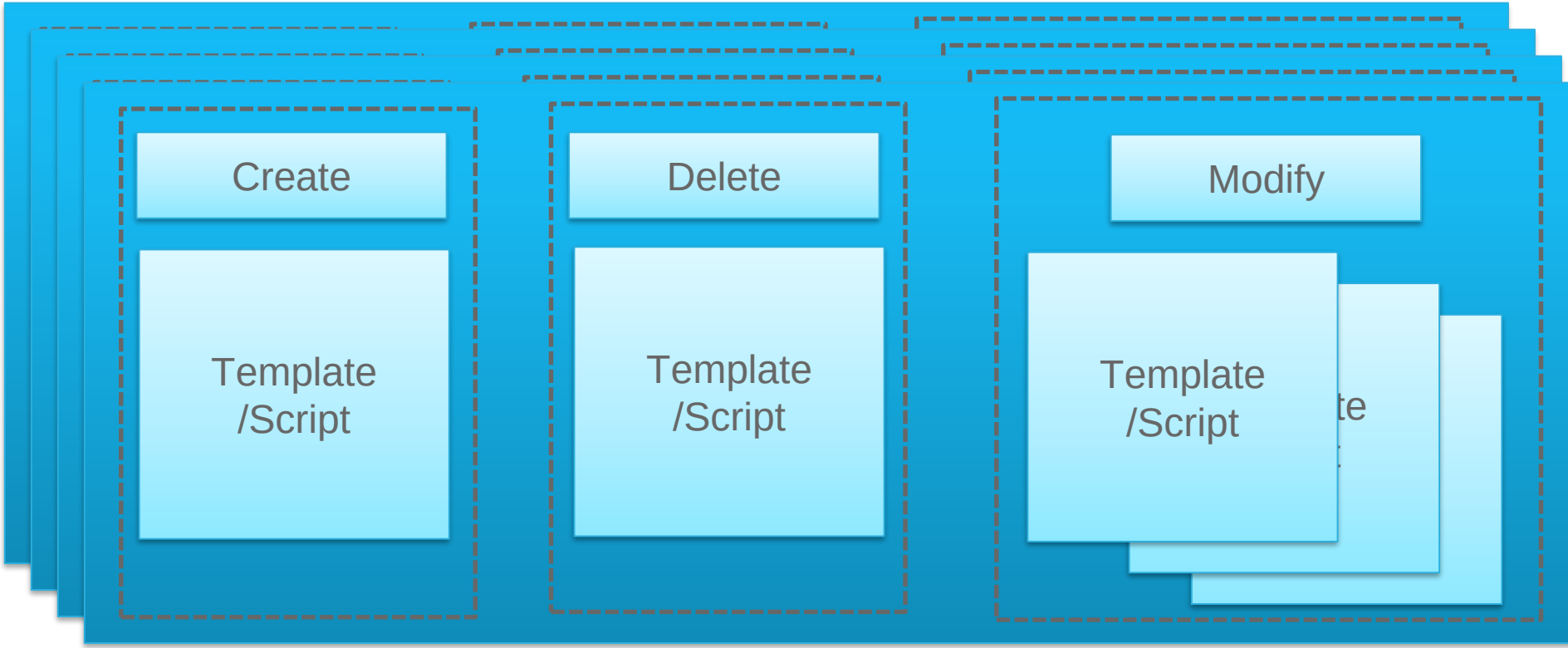


Model driven Auto-Generation : FASTMAP



NSO vs Traditional : FastMap* Algorithm

Traditional



NSO vs Traditional : FastMap* Algorithm

NSO

Reduces service implementation code/efforts by two orders of magnitude – Faster Time to Market

Create

Model Mapping / Prog.Logic

Delete

Modify

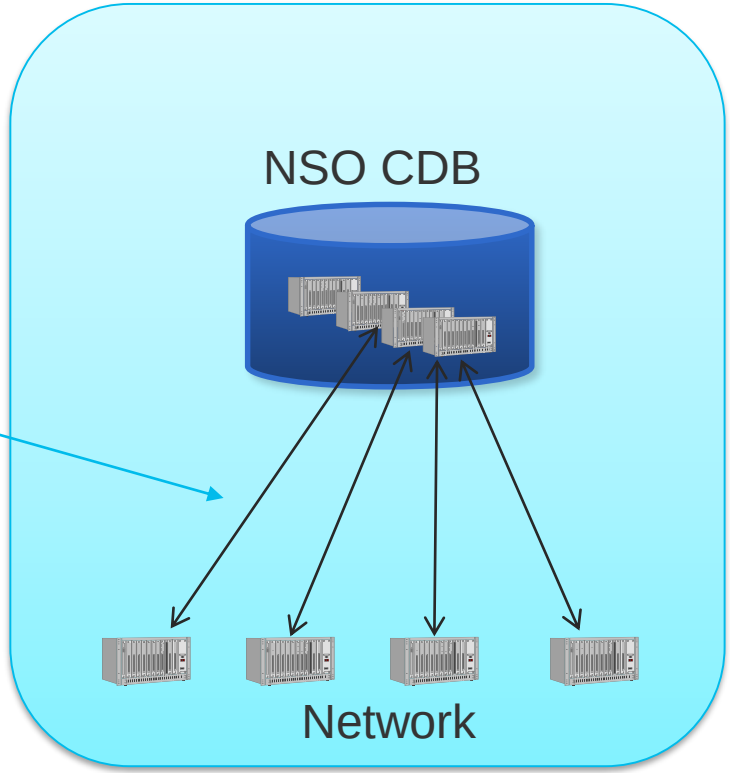
Auto Calculated (Minimal Change) using FastMap Algorithm

Service Config Synchronization

Accurate Provisioning is enabled in NSO through:

- Service-Level Data-Syncing

- sync-to
- sync-from
- check-sync
- compare-config



Summary

Day 3 – Recap

- Traditional Network management methods like CLI / SNMP has issues and can not be used for making the network programmable.
- YANG (RFC6020) was created to solve the inherent problems with Network management like SNMP eg., Transactional capability , Precise definitions, Separating transport and encoding of the data etc.,
  - YANG : Modeling Language
  - NETCONF / RESTCONF : Protocols to manage the YANG based data
- Not all devices support YANG / NETCONF
- NSO's Model driven architecture allows to abstract the network – any vendor , any domain to be consumed via YANG Model.
  - This is achieved using Cisco NEDs.
  - NEDs provide Device YANG Models and the engine to translate the model data to device specific Command / API
- NSO can help managing the multi-vendor network using NEDs with transactional capability. It provides device templates, compliance reports and policy rules to configure and validate the network

Day 3 – Recap

- Once the network is abstracted using YANG then the next step is to move to Service Layer.
- This is where the innovation starts where you can define your own services.
- Service model captures the service intent and needs to be kept as simplified as possible.
- Service Model will then need to be mapped to Device Models. This can be done via multiple mechanisms – Template , Java /Python & Templates.

Demo 3: Service Management

- Service Models
- Service Lifecycle Management

How does this help ?

- **Faster Time to Market :** Introduce New Services quickly
- Patented **FASTMAP Algorithm** helps to **reduce time and efforts** for introducing New Services.
- Single interface to all upstream Systems Eg., OSS/BSS, Portal etc.,
- Full E2E Transactional Capability

Key Takeaways

- NSO's standard based YANG/NETCONF **Model-driven approach helps**
  - **Abstracting Network and Service Layers:** This helps to focus on providing differentiated services to customers than focusing on network CLI/API complexities of multi-vendor network
  - Surgical precision configuration **reduces activation failure & human errors**
  - **Improve Customer experience** by quickly identifying the service config discrepancies and rectify quickly
  - NED support for new capabilities **without any additional change request costs**
- Fastmap Algorithm **reduces service introduction time** significantly
- NSO has **broadest Multi-vendor support** helps in supporting existing and future network vendors
- Industry **proven ETSI MANO compliant Automation** Platform **supports multi-domain physical and virtual networks** future proofing investment

CISCO