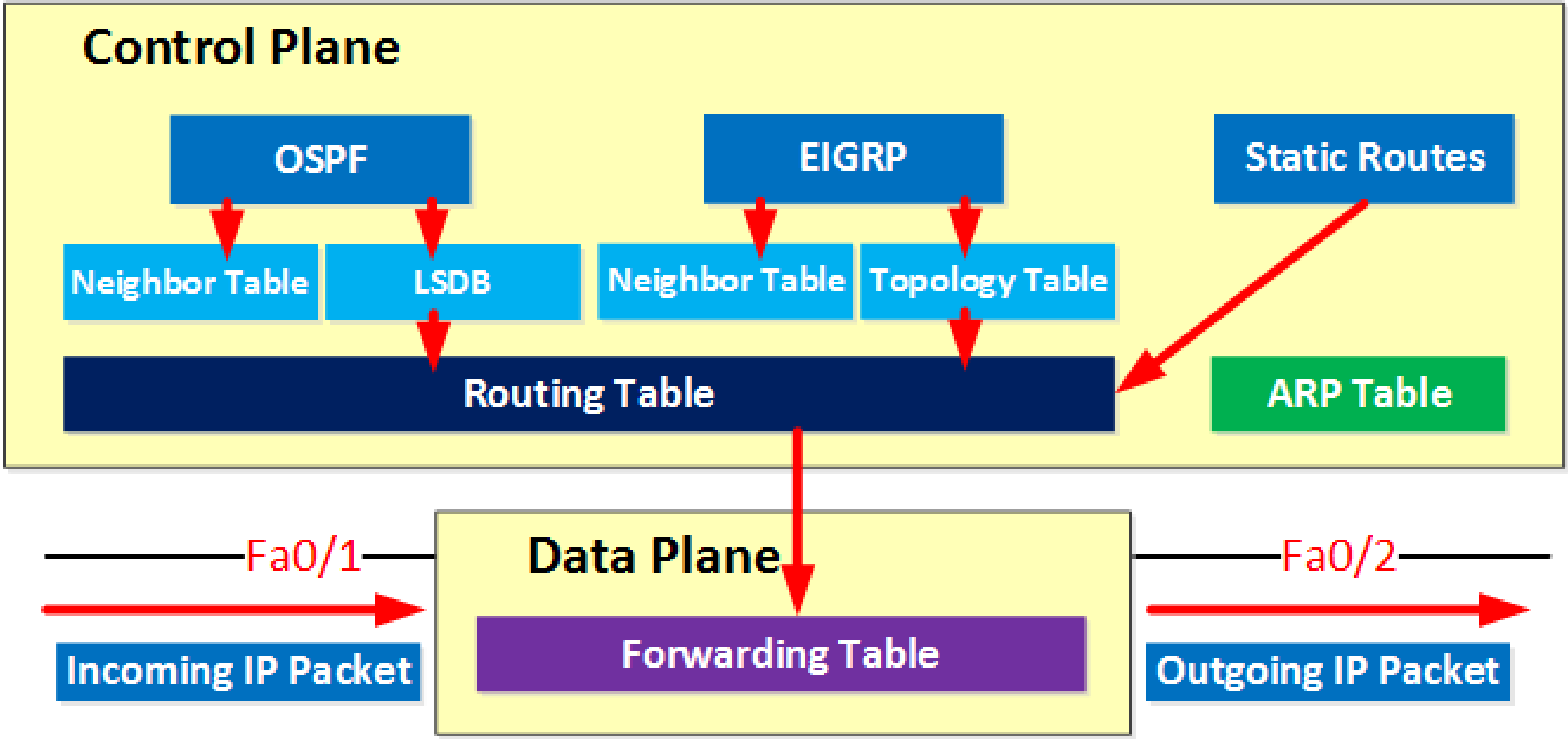
Software-defined networking (SDN)

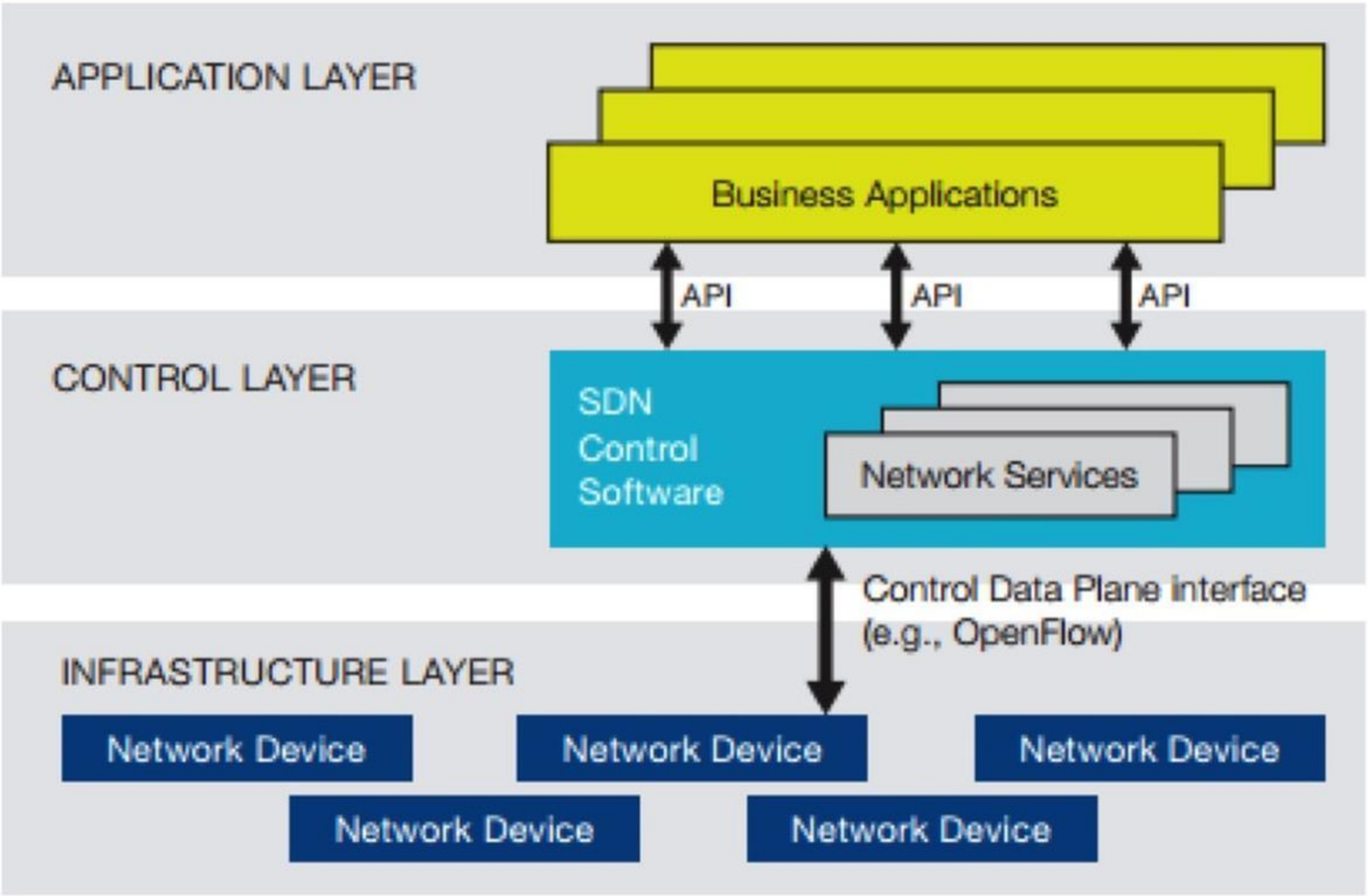
What is SDN?

- **SDN is defined as a separation of the control and data plane of a network device.**
- **A separated and centralized, software-based control plane allows network operators to administrate the network as a single unit, which lowers operational costs, and through open interfaces, makes integration to applications much easier.**
- **The SDN controller serves as the network control plane and the single management interface for the network.**

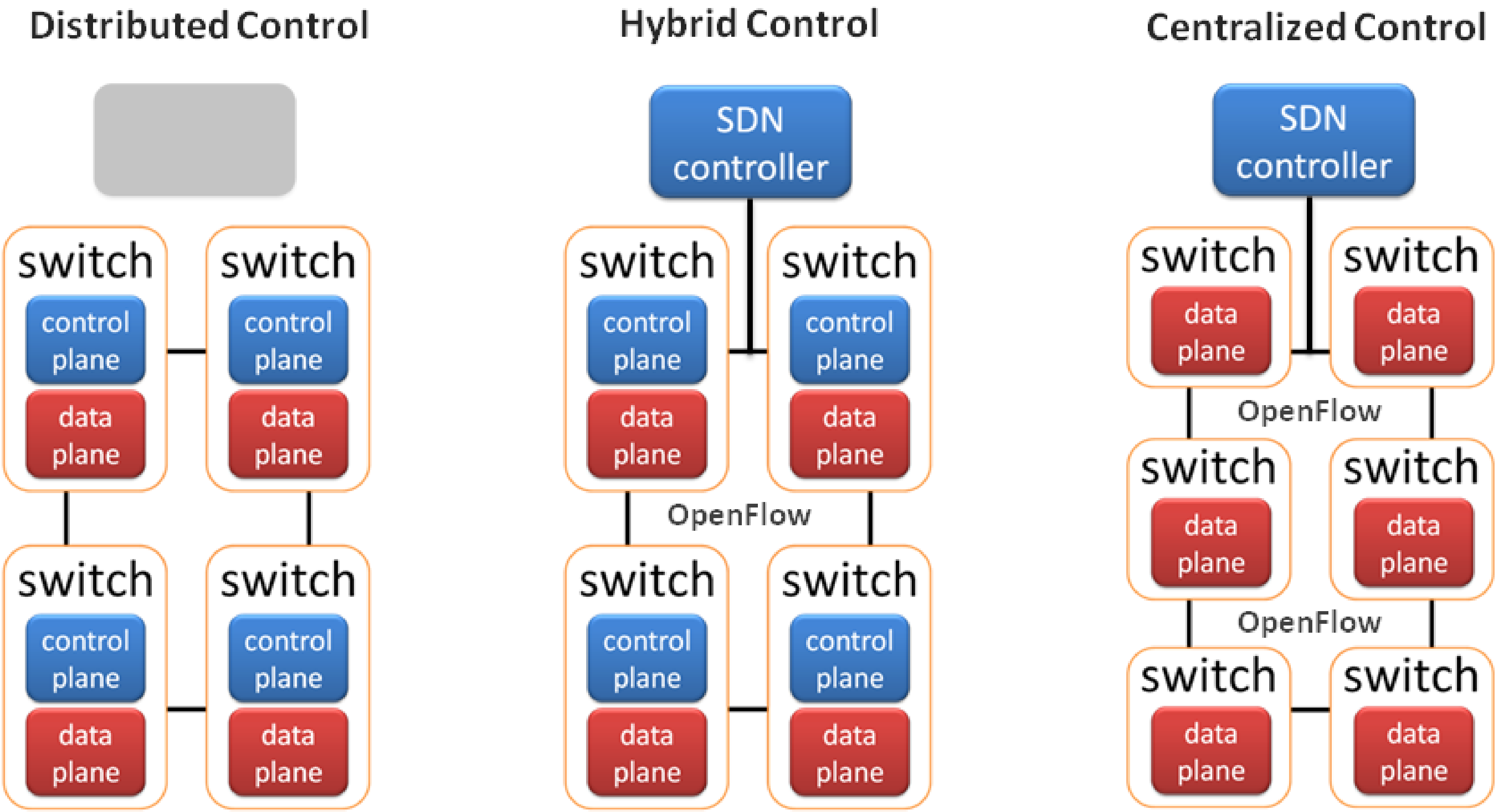
Control Plane vs Data Plane



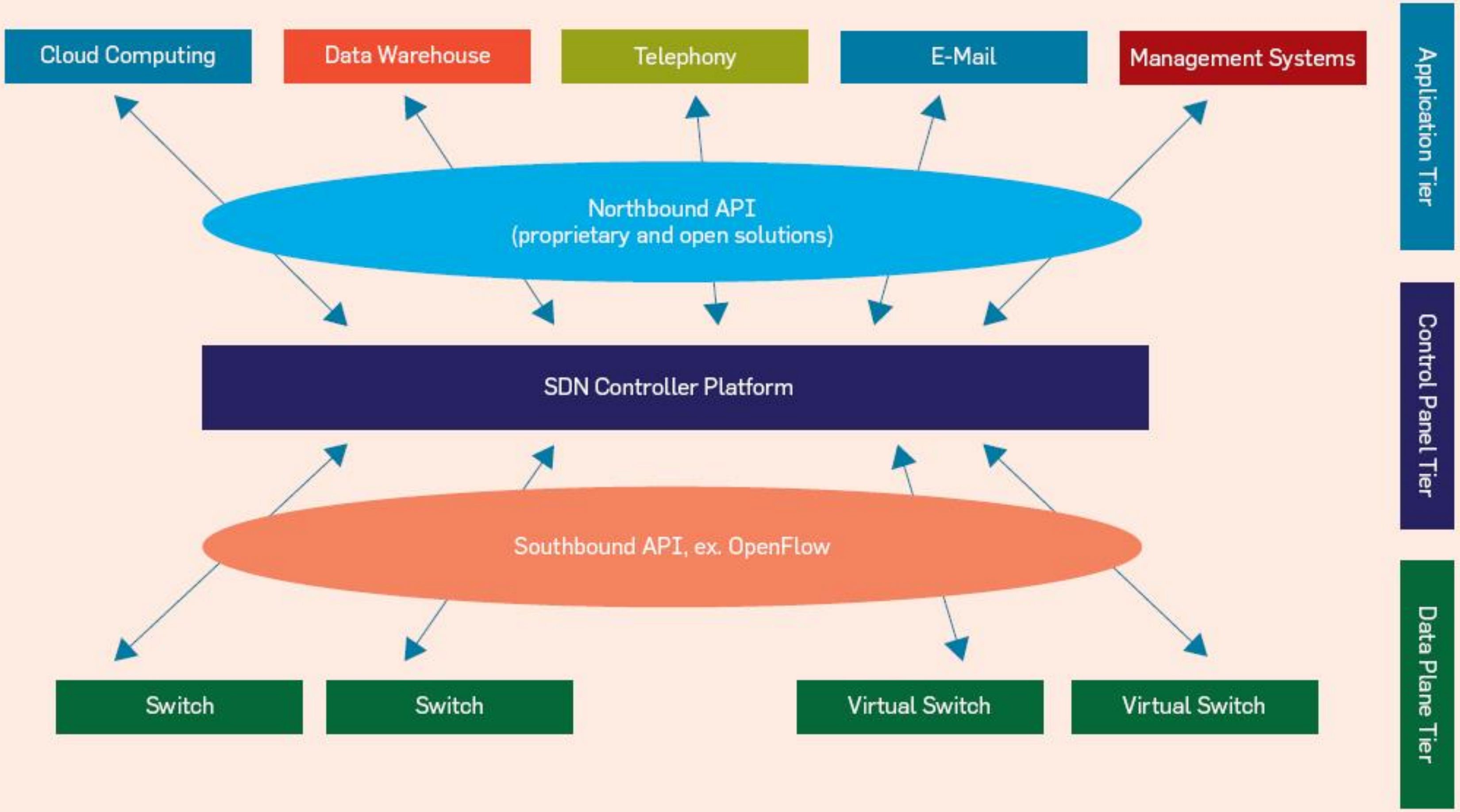
SDN Architecture



SDN Architecture (Cont)



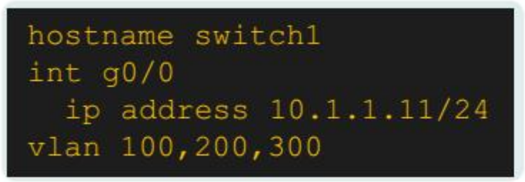
SDN Architecture (Cont)



Why Network Programmability

Why automation and programmability?

Needs to configure

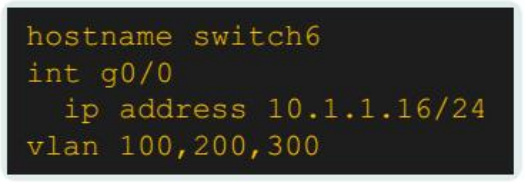


```
hostname switch1
int g0/0
  ip address 10.1.1.11/24
vlan 100,200,300
```

.

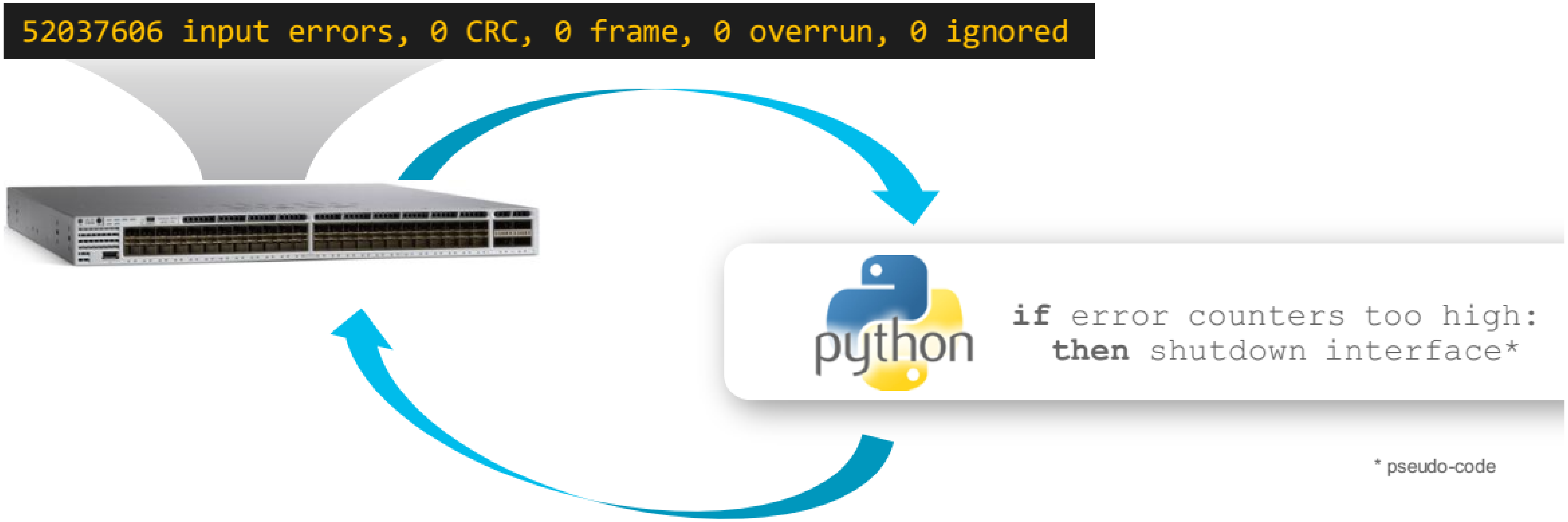
.

.



```
hostname switch6
int g0/0
  ip address 10.1.1.16/24
vlan 100,200,300
```

Automation and Programmability



```
52037606 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
```

```
if error counters too high:
  then shutdown interface*
```

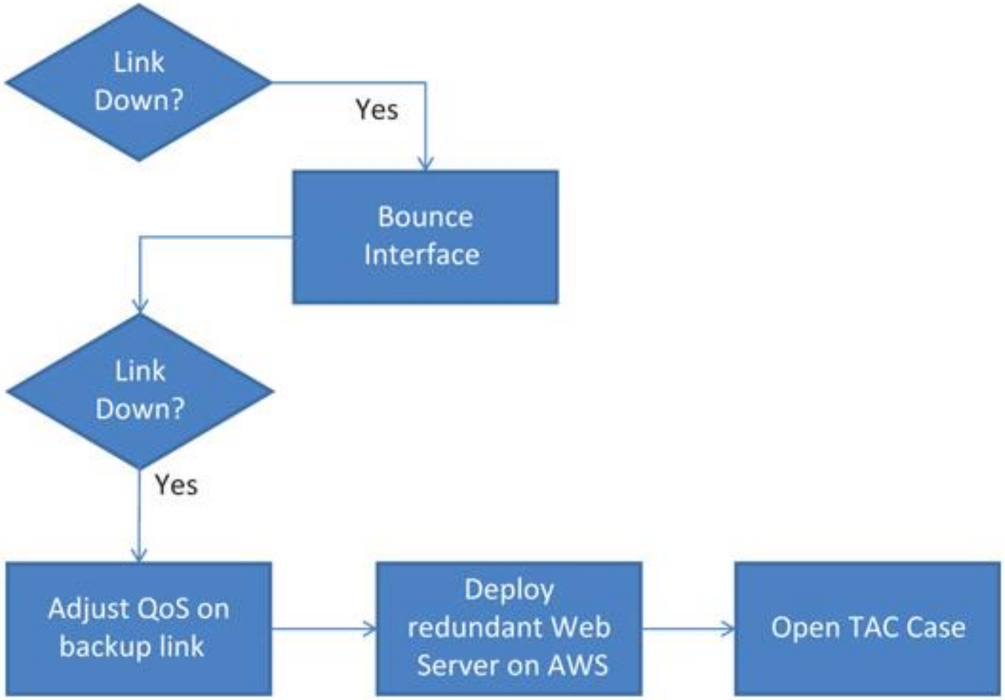
* pseudo-code

Programmatic Control of network devices

- **Network programmability changes the role of network administrators from operationally focused to innovation focused.**
- **Network programmability frees a network administrator of operational burdens and allows them to focus on utilizing the network as a source of innovation.**

Automation and Programmability

- **Network automation is the process of automating the configuration, management, testing, deployment, and operations of physical and virtual devices within a network.**
- **Like programmability, automation reduces cost and complexity, but it does so with an *if* statement to automatically invoke actions or changes, based on events.**

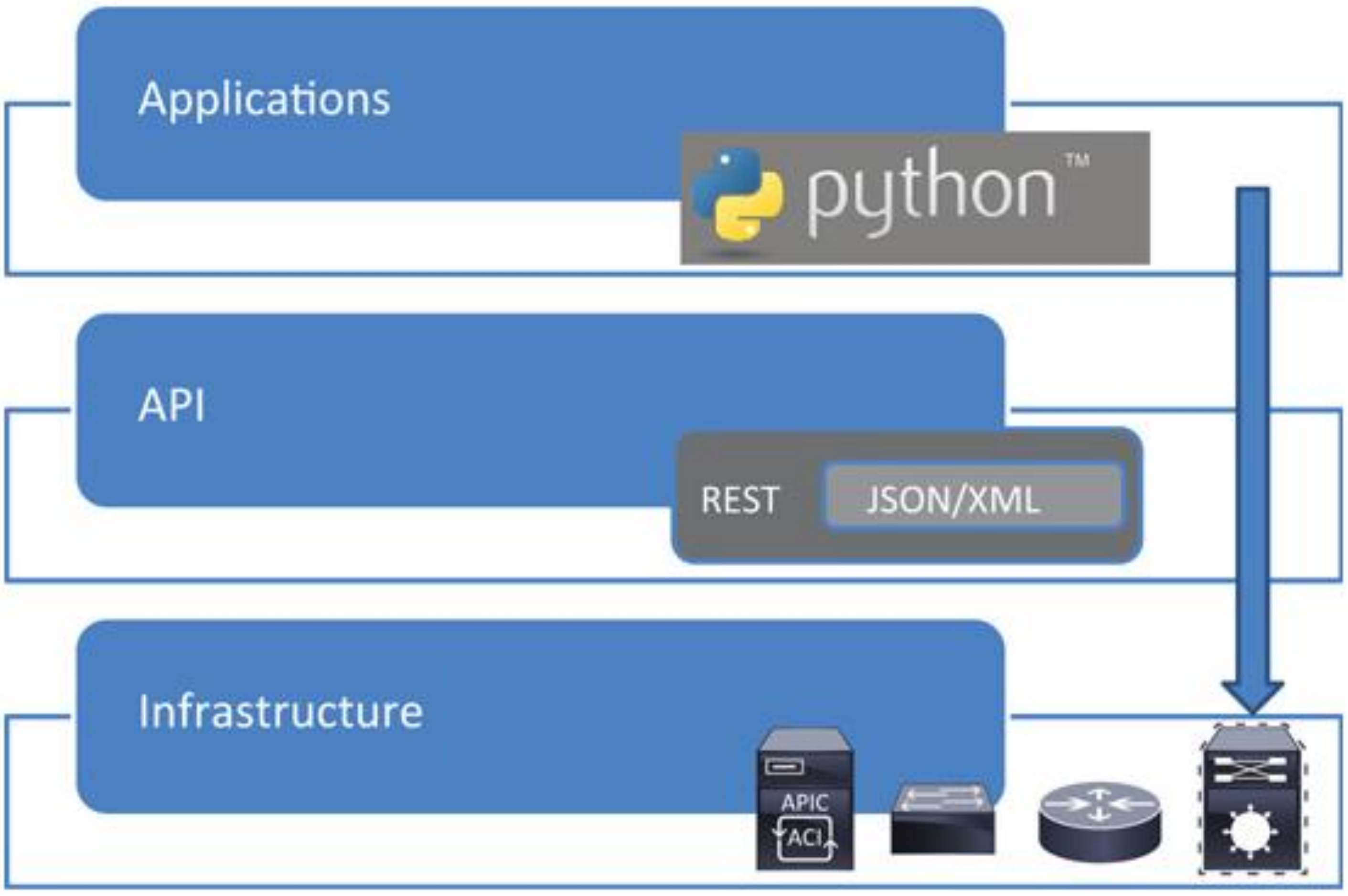


Network Programmability and Automation Benefits

Some of the benefits of network programmability & automation include:

- **Time and money cost savings**
- **Customization**
- **Reduction of human error**
- **Innovation**

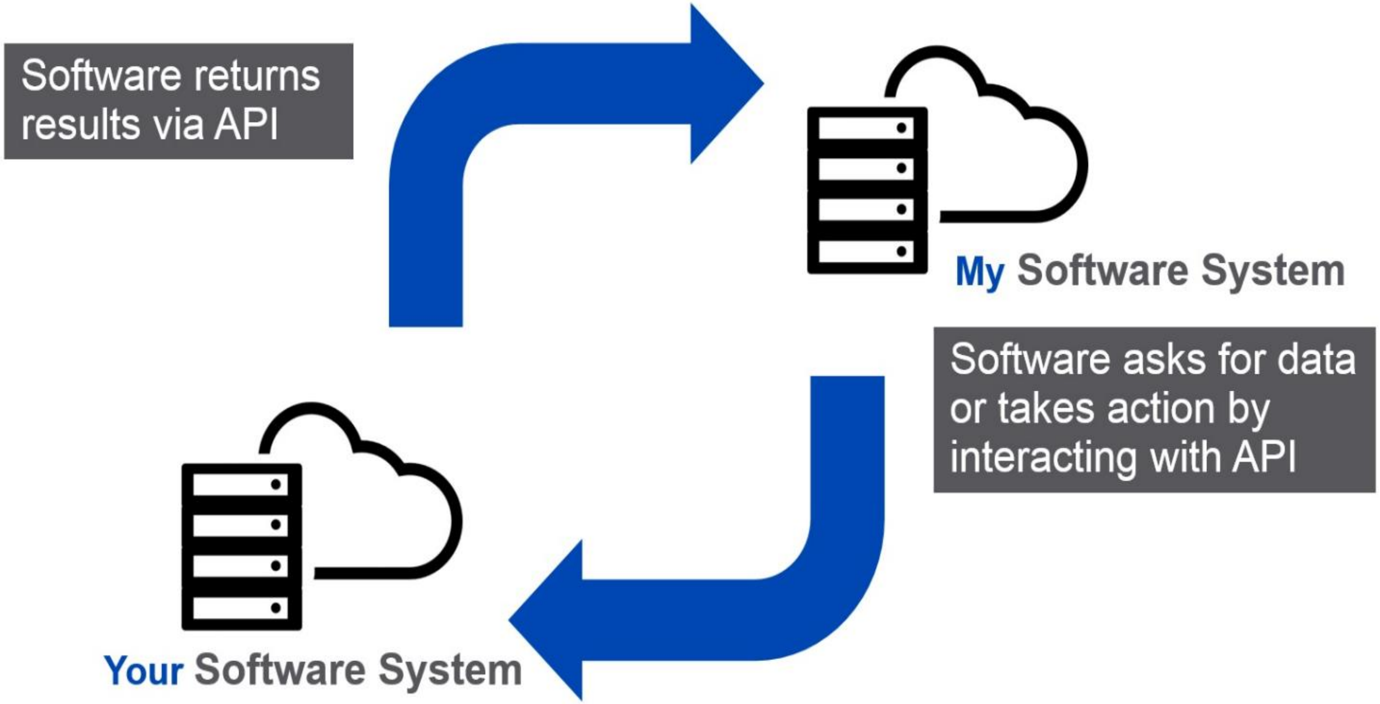
What is network programmability?



Overview of APIs

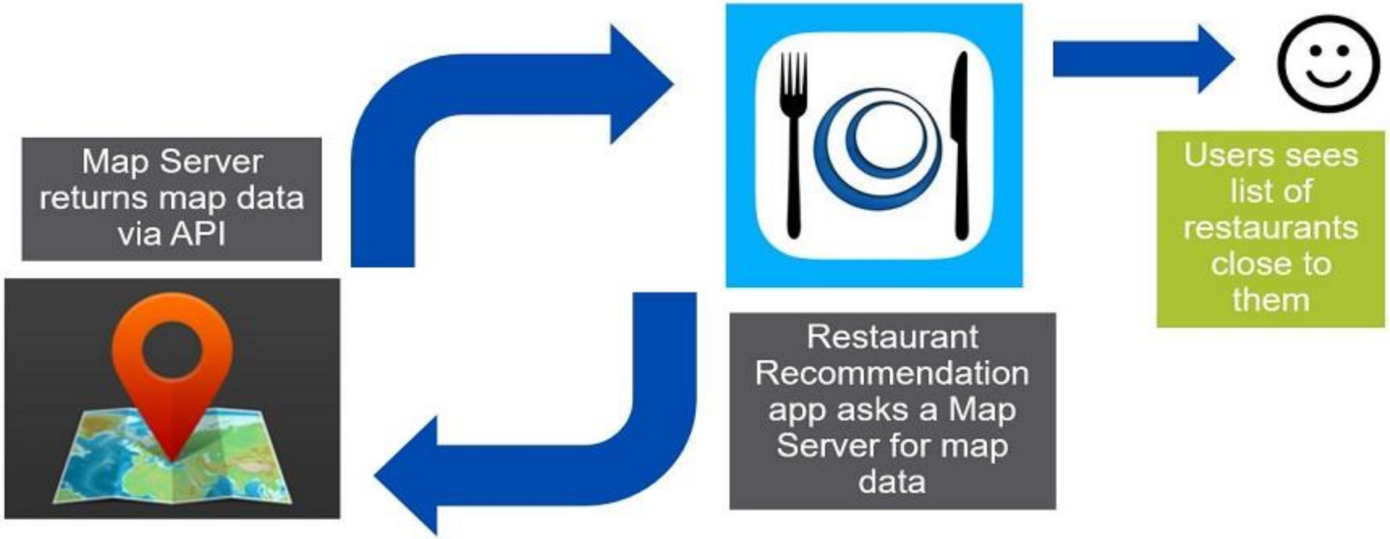
An API (Application Programming Interface) enables two pieces of software to communicate with each other.

Creating APIs enables the development of rich applications with a wide range of functionality.


Software returns results via API
My Software System
Software asks for data or takes action by interacting with API
Your Software System

Overview of APIs (Cont)

APIs help developers create apps that benefit the end user.



Map Server returns map data via API

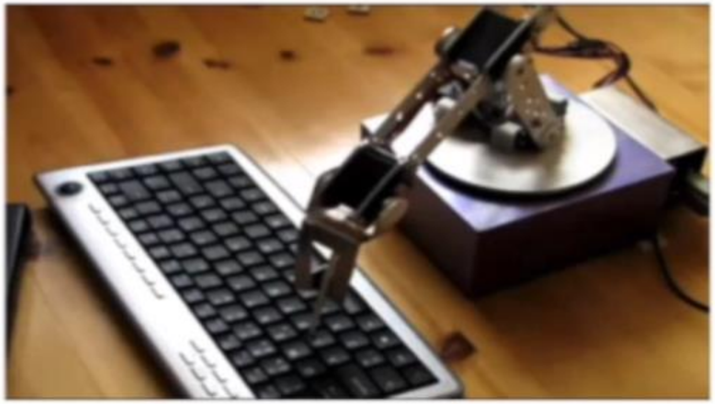
Restaurant Recommendation app asks a Map Server for map data

Users sees list of restaurants close to them

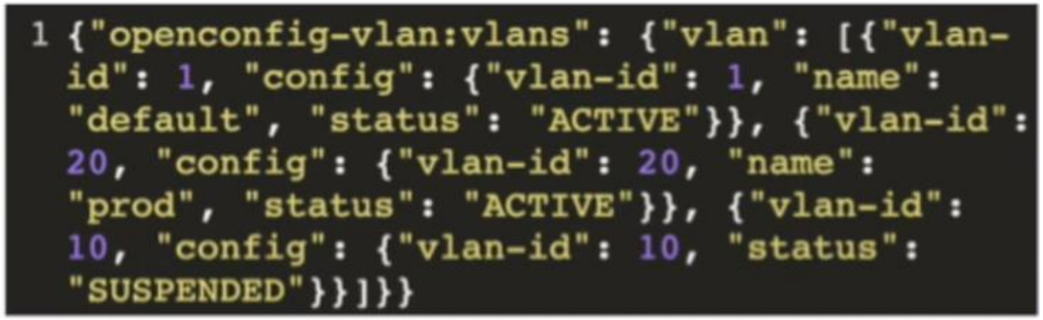
- **For example, suppose you create a restaurant recommendation app. When users search for a restaurant, you want the app to return a list of relevant restaurants including a map of where they are.**

CLI vs API

CLI: Easy to Read



API: Hard to Read

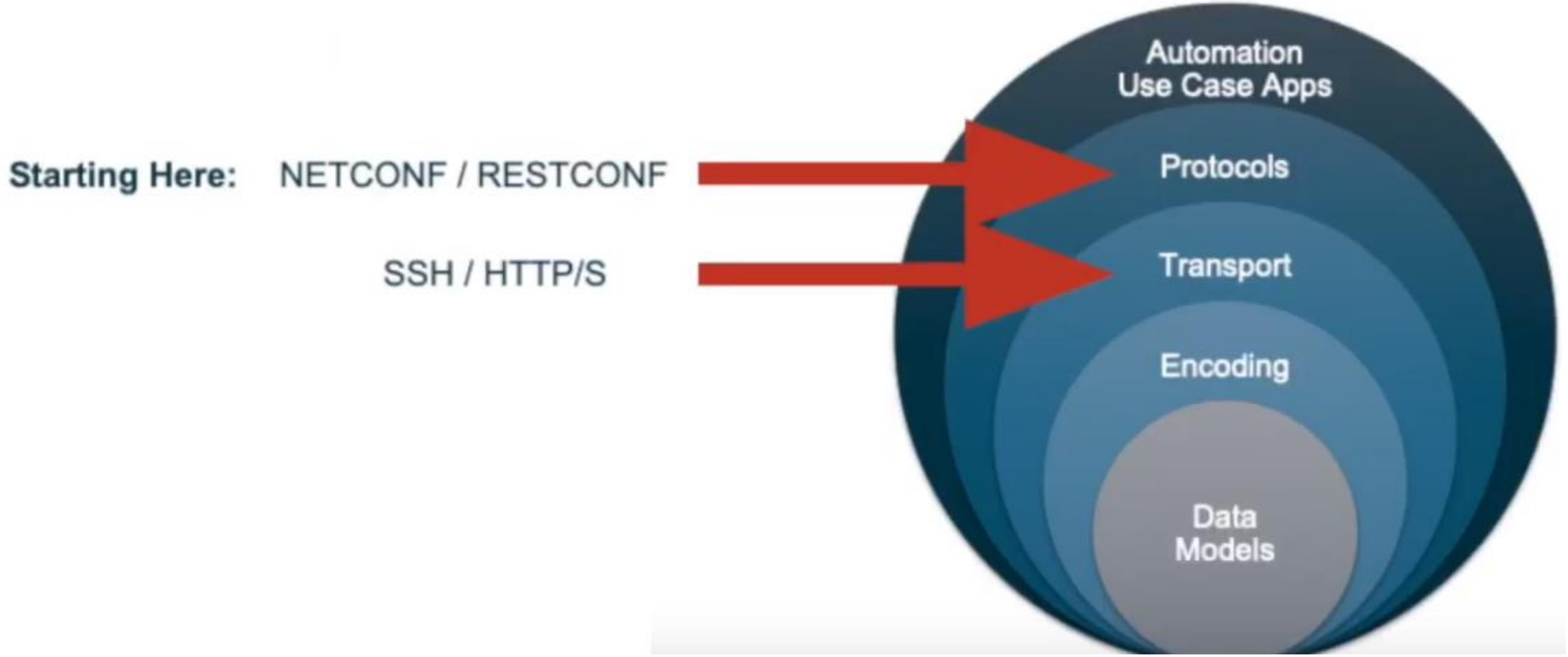


```
{"openconfig-vlan:vlans": {"vlan": [{"vlan-
id": 1, "config": {"vlan-id": 1, "name":
"default", "status": "ACTIVE"}}, {"vlan-id":
20, "config": {"vlan-id": 20, "name":
"prod", "status": "ACTIVE"}}, {"vlan-id":
10, "config": {"vlan-id": 10, "status":
"SUSPENDED"}}]}}
```

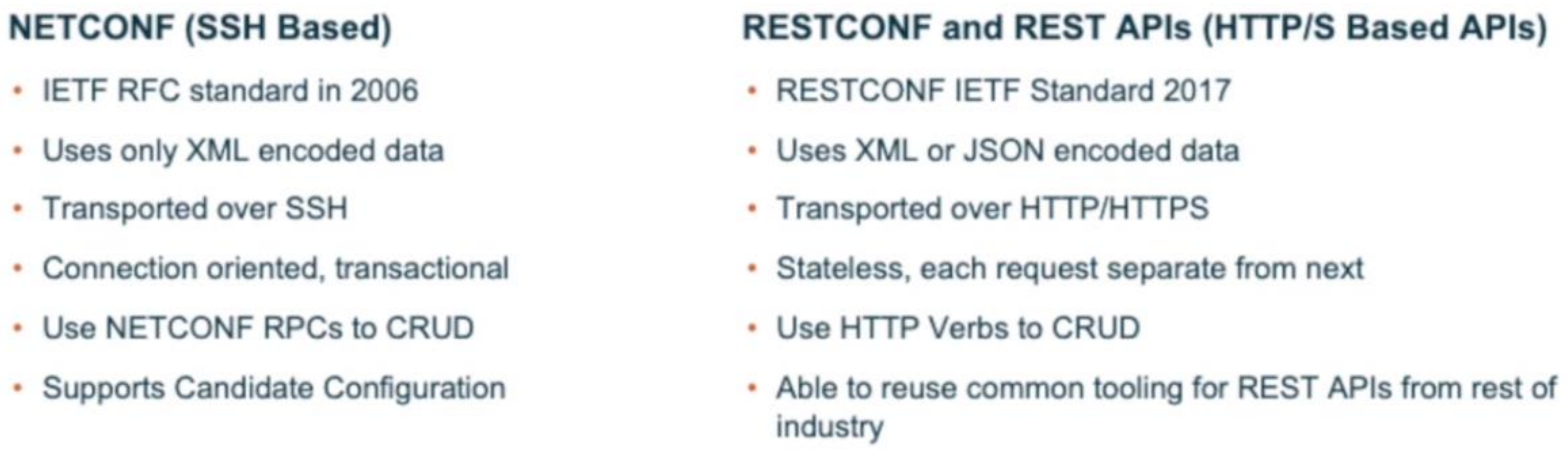
Built for Humans

Built for Machines

API Protocols and Transport



Two Most Common Network API Protocols



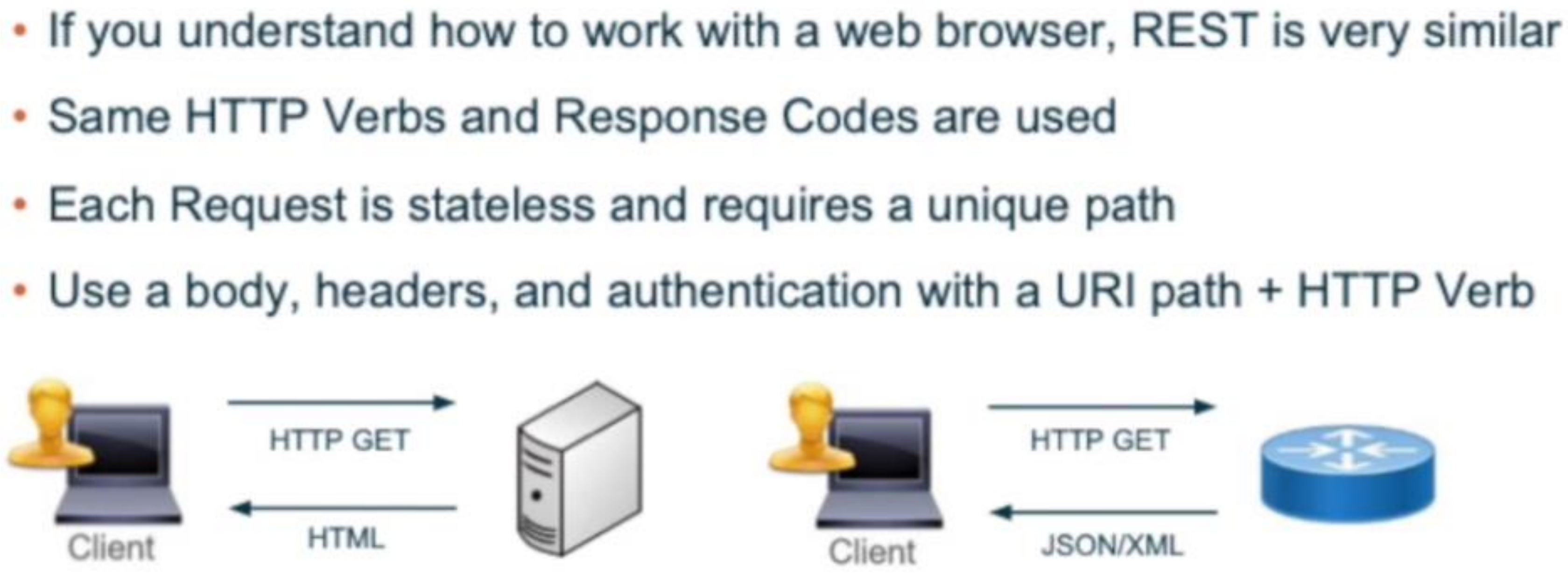
NETCONF (SSH Based)

- IETF RFC standard in 2006
- Uses only XML encoded data
- Transported over SSH
- Connection oriented, transactional
- Use NETCONF RPCs to CRUD
- Supports Candidate Configuration

RESTCONF and REST APIs (HTTP/S Based APIs)

- RESTCONF IETF Standard 2017
- Uses XML or JSON encoded data
- Transported over HTTP/HTTPS
- Stateless, each request separate from next
- Use HTTP Verbs to CRUD
- Able to reuse common tooling for REST APIs from rest of industry

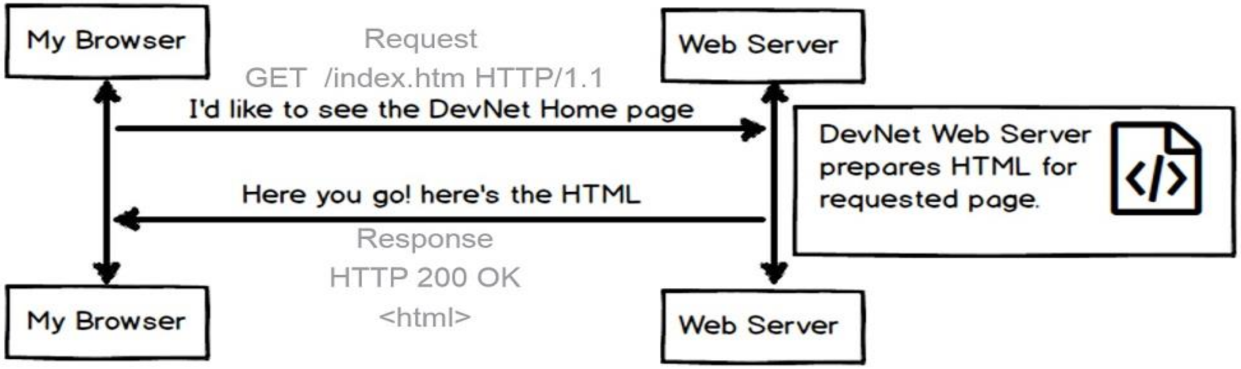
What is REST?



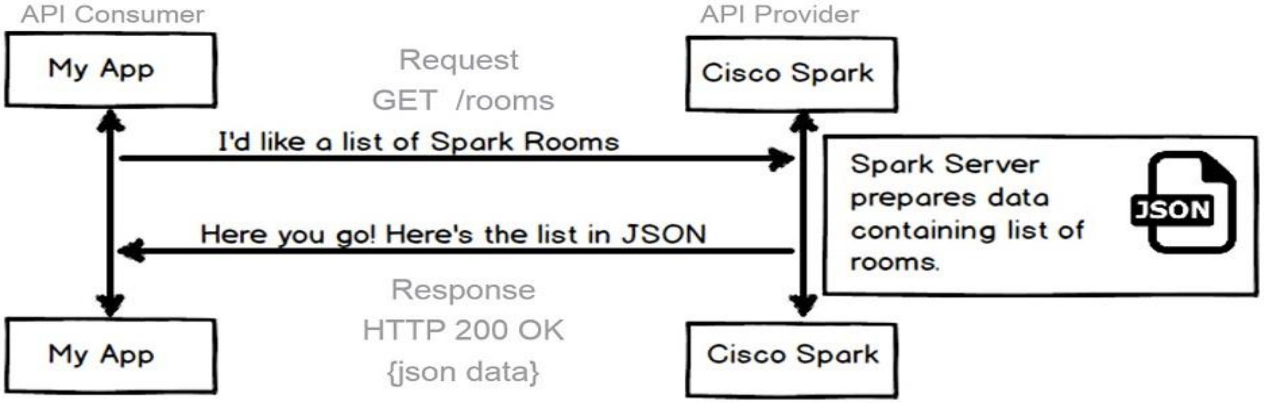
- If you understand how to work with a web browser, REST is very similar
- Same HTTP Verbs and Response Codes are used
- Each Request is stateless and requires a unique path
- Use a body, headers, and authentication with a URI path + HTTP Verb

What is REST? (Cont)

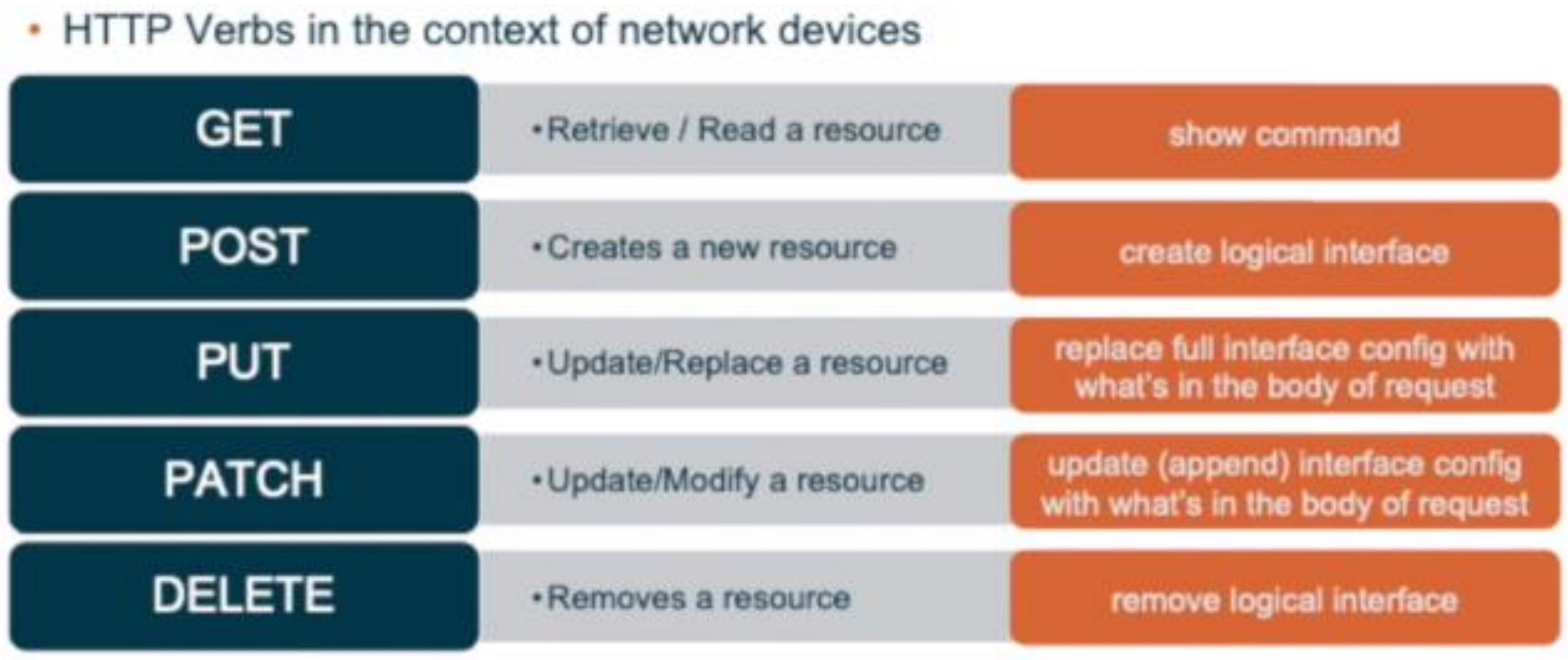
HTTP(S) uses CRUD (Create, Read, Update, Delete) operations on the wire to request data.


My Browser
Web Server
Request
GET /index.htm HTTP/1.1
I'd like to see the DevNet Home page
DevNet Web Server prepares HTML for requested page.
</>
Here you go! here's the HTML
Response
HTTP 200 OK
<html>
My Browser
Web Server

Browsers are replaced by software to interface with the RESTful service.


API Consumer
API Provider
My App
Cisco Spark
Request
GET /rooms
I'd like a list of Spark Rooms
Spark Server prepares data containing list of rooms.
JSON
Here you go! Here's the list in JSON
Response
HTTP 200 OK
{json data}
My App
Cisco Spark

REST (Cont)



- HTTP Verbs in the context of network devices

GET	•Retrieve / Read a resource	show command
POST	•Creates a new resource	create logical interface
PUT	•Update/Replace a resource	replace full interface config with what's in the body of request
PATCH	•Update/Modify a resource	update (append) interface config with what's in the body of request
DELETE	•Removes a resource	remove logical interface

The NETCONF Protocol

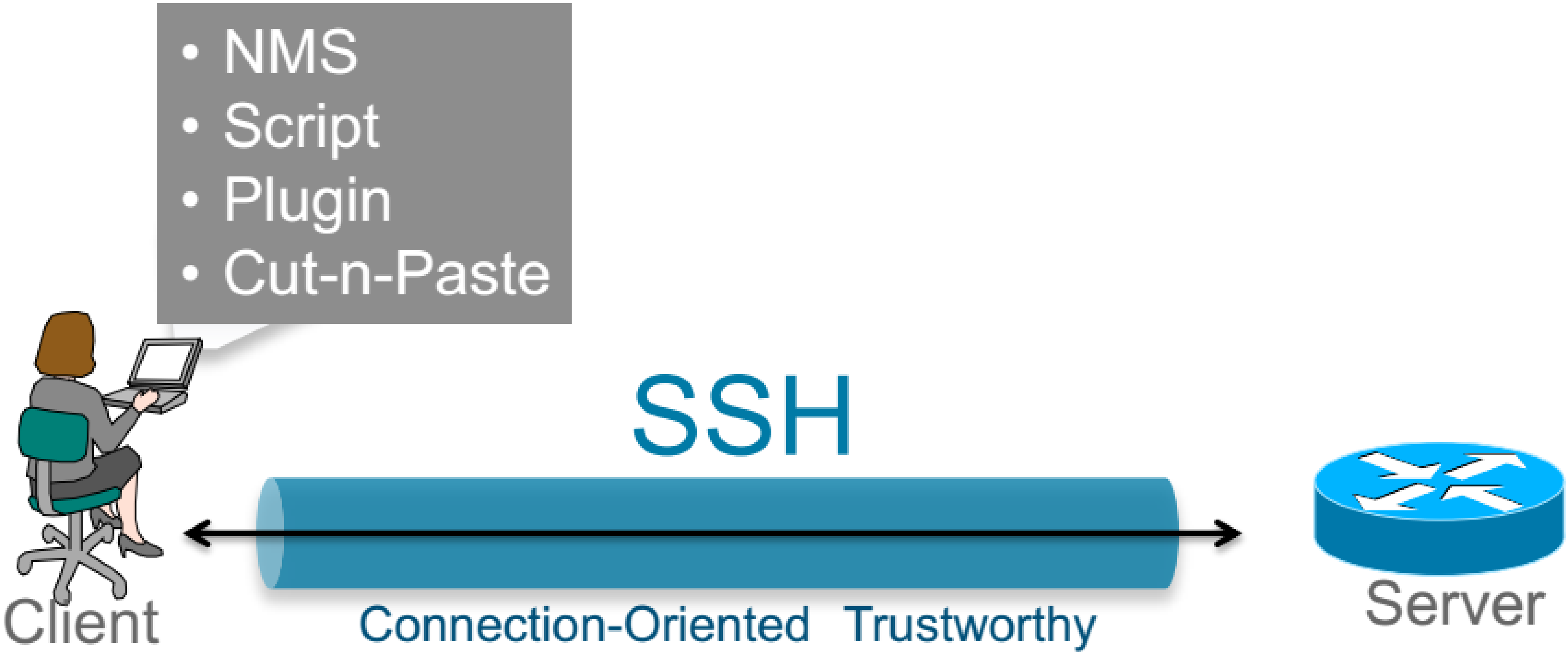
Protocol to Manipulate Networks

- IETF network management protocol
- Distinction between configuration and state data
- Multiple configuration data stores (candidate, running, startup)
- Configuration change validations and transactions
- Selective data retrieval with filtering
- Streaming and playback of event notifications

Why you should care:

NETCONF provides fundamental programming features for comfortable and robust automation of network services

NETCONF Uses a Client-Server Model



- NMS
- Script
- Plugin
- Cut-n-Paste

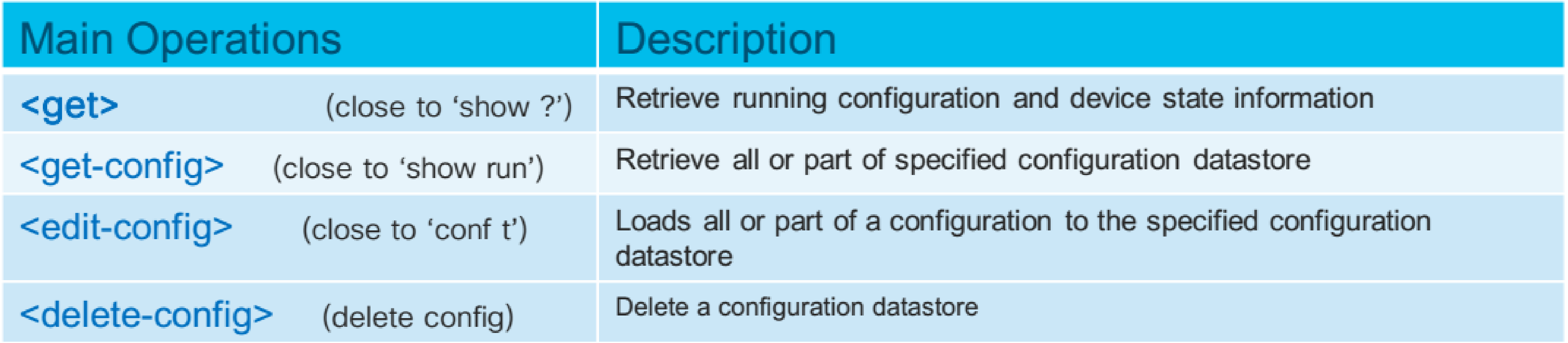
SSH

Client

Server

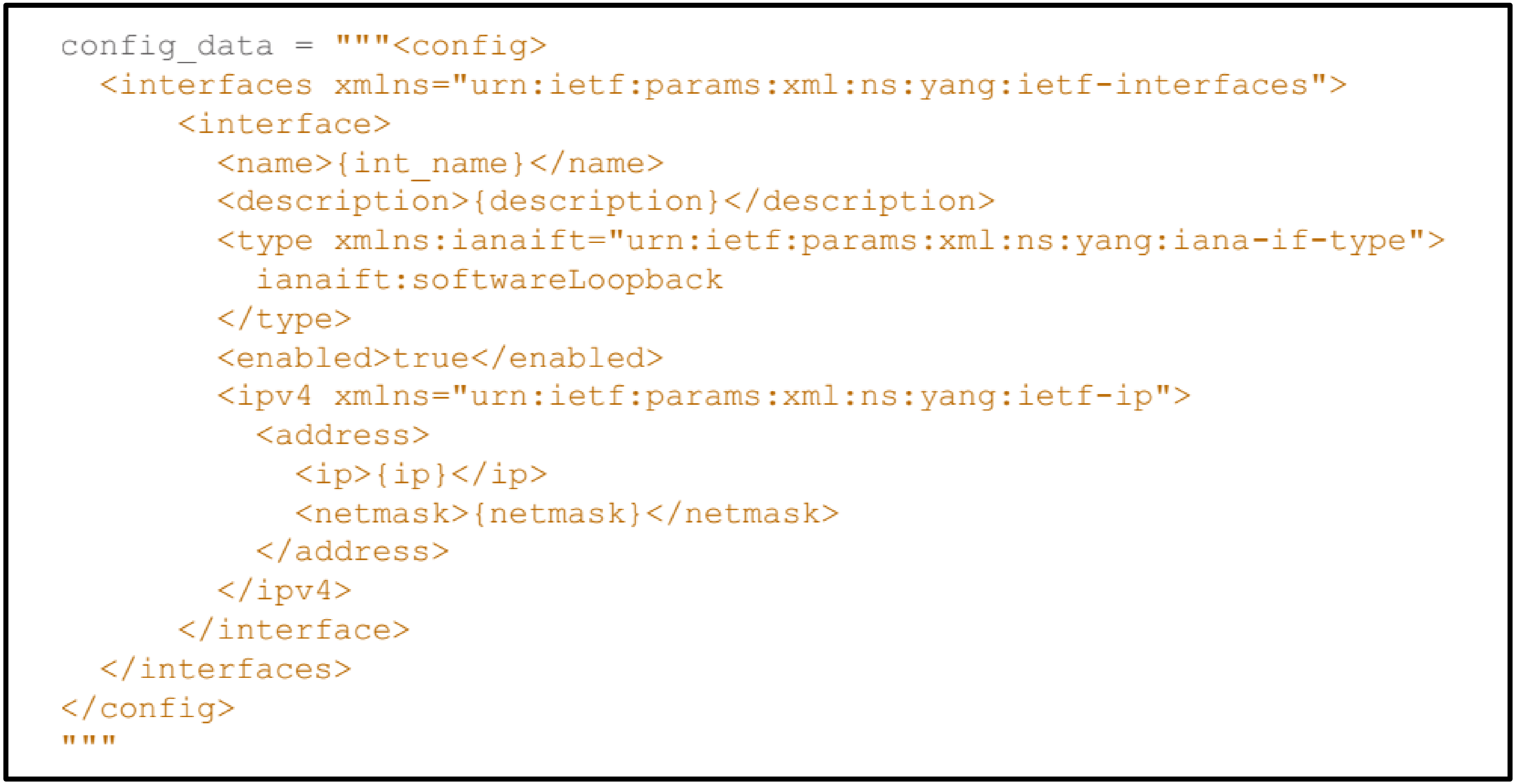
Connection-Oriented	Trustworthy
Authenticated	Secure
Reliable	

NETCONF



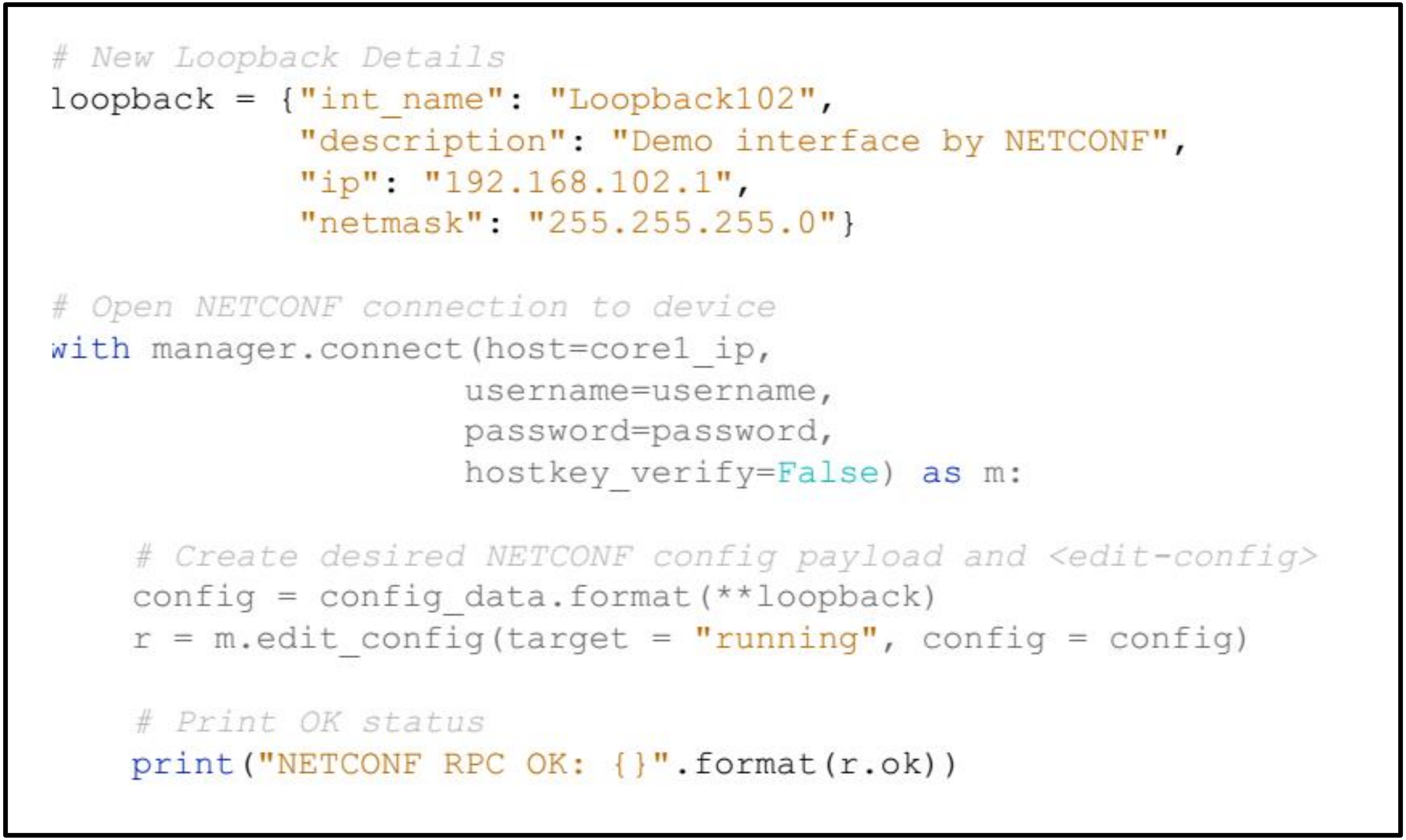
Main Operations	Description
<get> (close to 'show ?')	Retrieve running configuration and device state information
<get-config> (close to 'show run')	Retrieve all or part of specified configuration datastore
<edit-config> (close to 'conf t')	Loads all or part of a configuration to the specified configuration datastore
<delete-config> (delete config)	Delete a configuration datastore

NETCONF: Creating a New Loopback 1/2



```
config_data = """<config>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>{int_name}</name>
        <description>{description}</description>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
          ianaift:softwareLoopback
        </type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
          <address>
            <ip>{ip}</ip>
            <netmask>{netmask}</netmask>
          </address>
        </ipv4>
      </interface>
  </interfaces>
</config>
"""
```

NETCONF: Creating a New Loopback 2/2



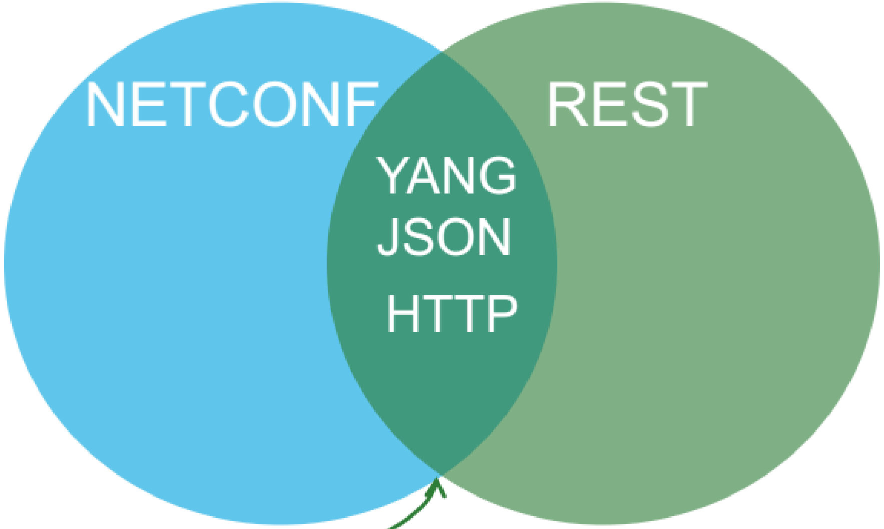
```python
# New Loopback Details
loopback = {"int_name": "Loopback102",
            "description": "Demo interface by NETCONF",
            "ip": "192.168.102.1",
            "netmask": "255.255.255.0"}

# Open NETCONF connection to device
with manager.connect(host=core1_ip,
                     username=username,
                     password=password,
                     hostkey_verify=False) as m:

    # Create desired NETCONF config payload and <edit-config>
    config = config_data.format(**loopback)
    r = m.edit_config(target = "running", config = config)

    # Print OK status
    print("NETCONF RPC OK: {}".format(r.ok))
```

What is RESTCONF?

“the simplicity of the HTTP protocol with the predictability and automation potential of a schema-driven API”

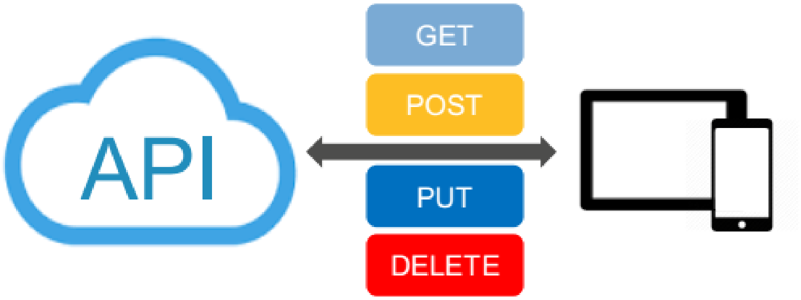
NETCONF/YANG
– SSH
– XML
+ JSON
+ HTTP(S)

RESTCONF



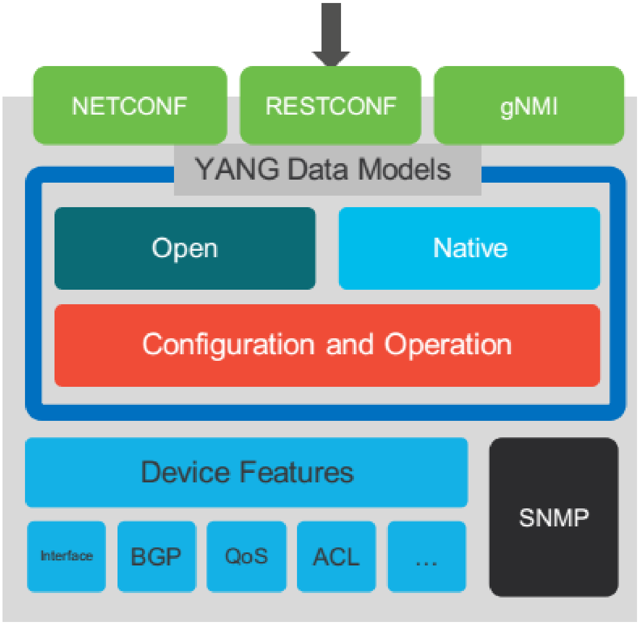
REST And RESTCONF

REST



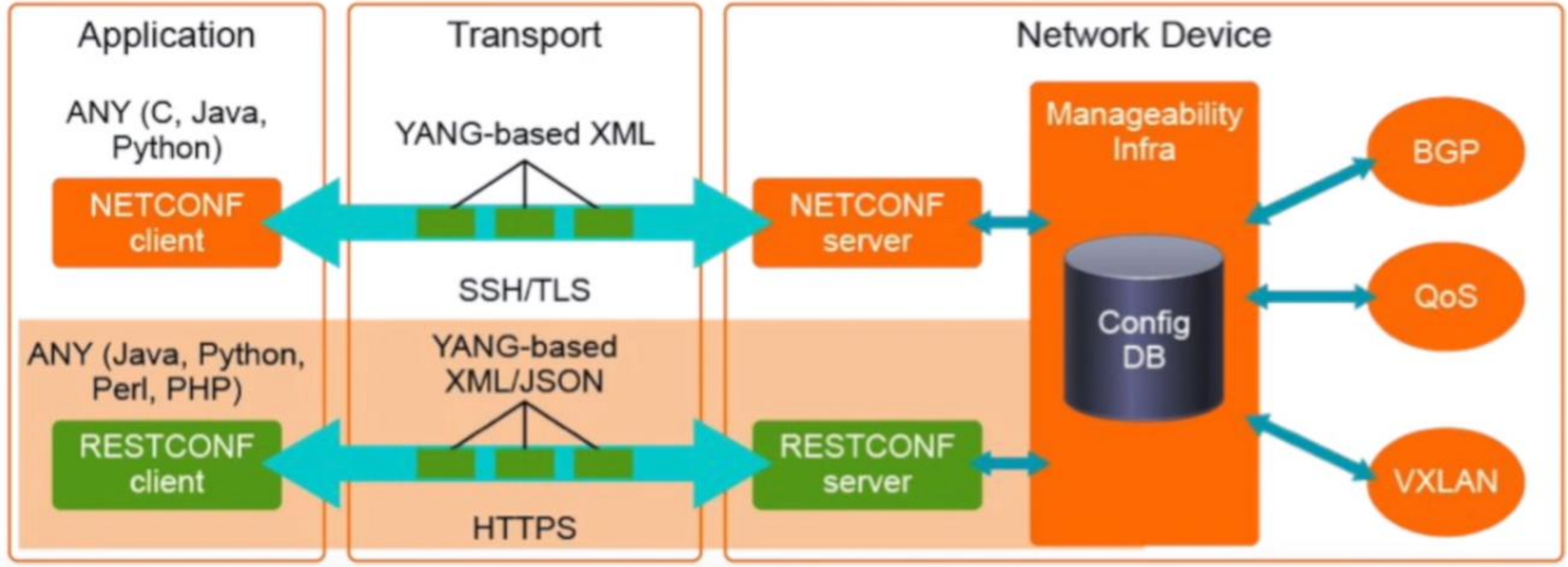
“A framework for client-server communications”

RESTCONF



“REST-like protocol for accessing YANG models”

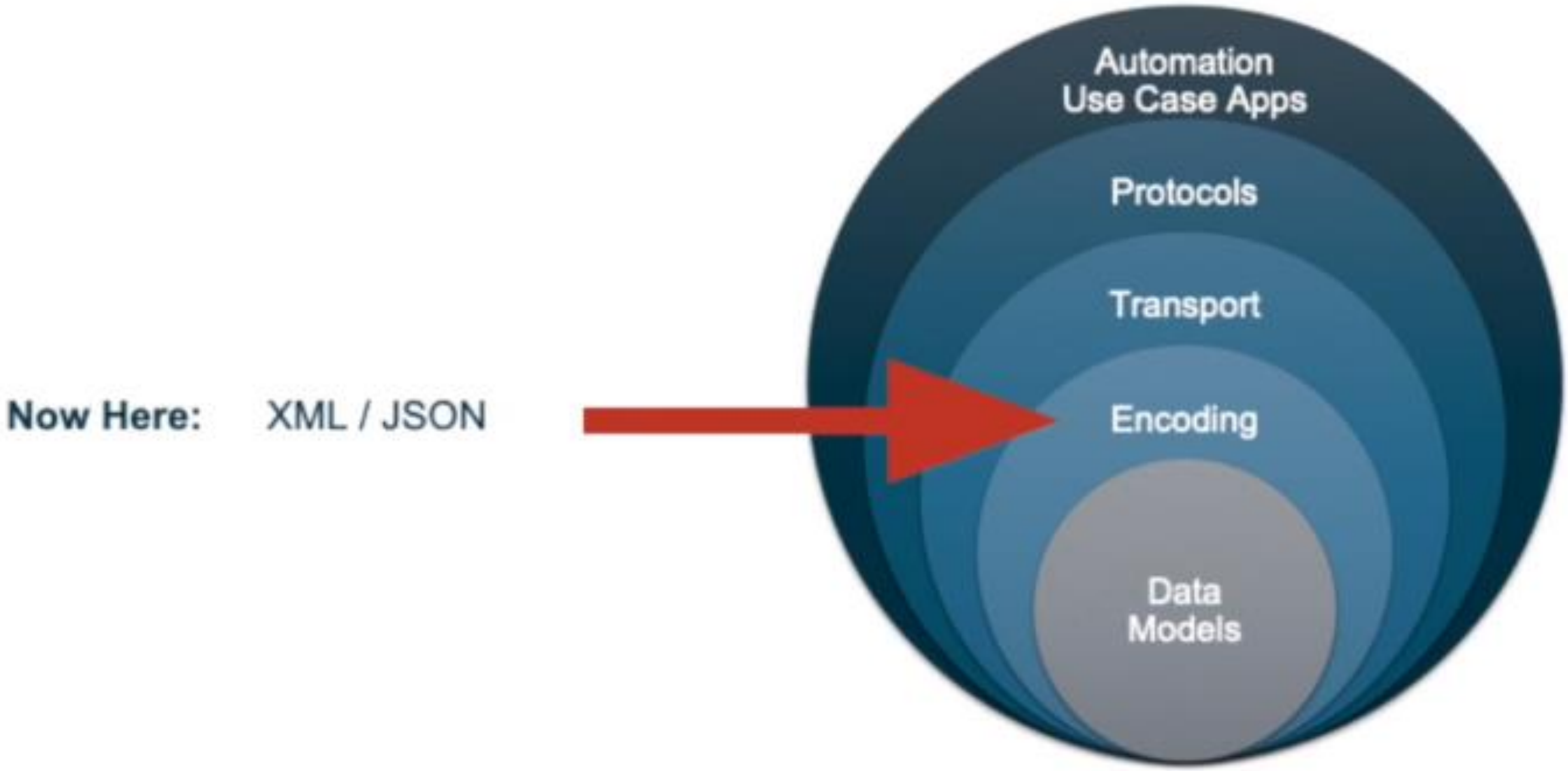
NETCONF And RESTCONF



NETCONF versus RESTCONF

- RESTCONF:
  - The HTTP POST, PUT, PATCH, and DELETE methods are used to edit data resources represented by YANG data models
  - RESTCONF is rest-like
  - Encoding: XML or JSON
- NETCONF
  - XML only

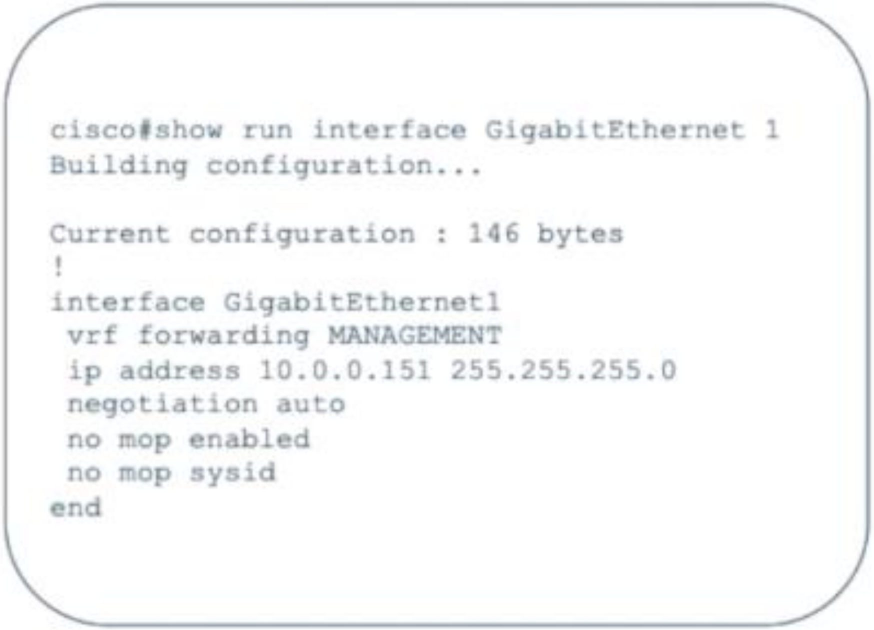
Data Encoding



Now Here: XML / JSON

The Need for Data Encoding

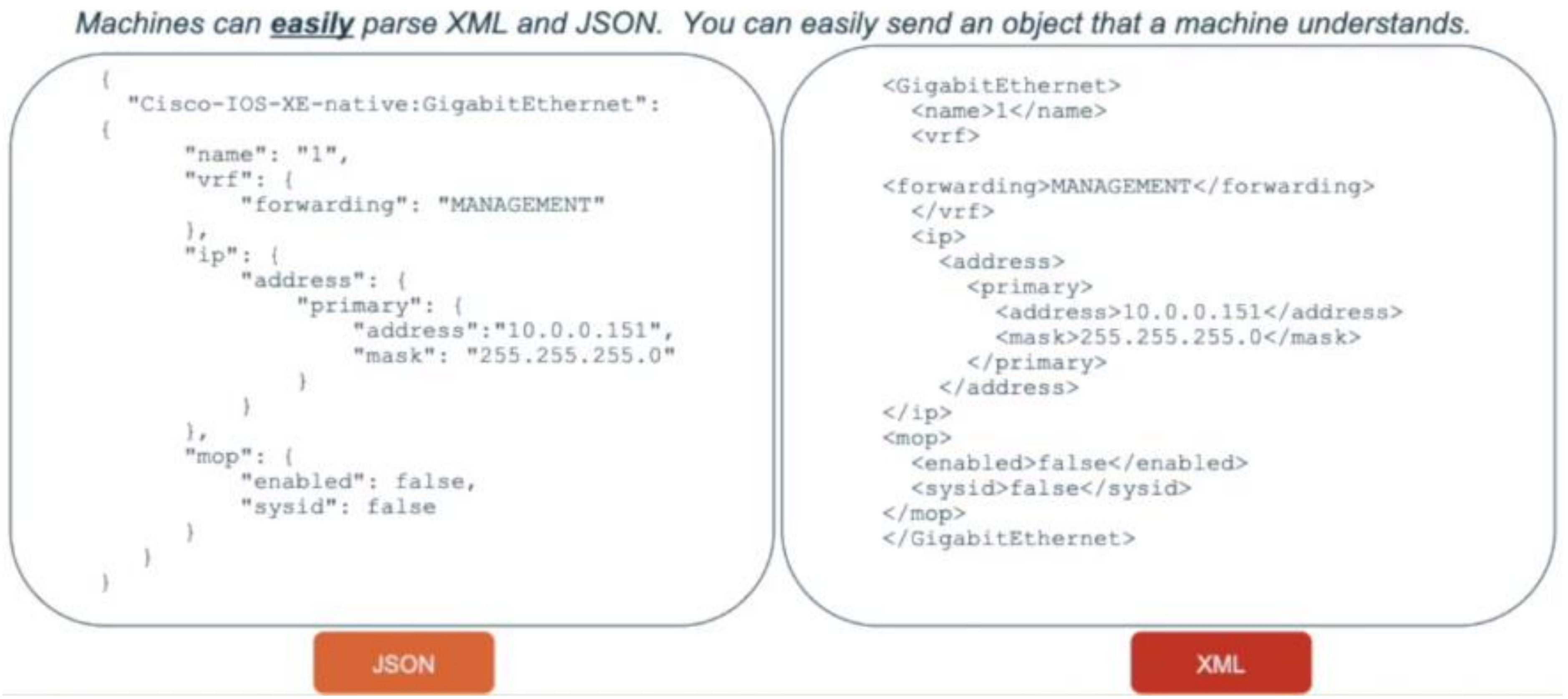
There needs to be structure behind what is communicated between **systems**



```
cisco#show run interface GigabitEthernet 1
Building configuration...

Current configuration : 146 bytes
!
interface GigabitEthernet1
 vrf forwarding MANAGEMENT
 ip address 10.0.0.151 255.255.255.0
 negotiation auto
 no mop enabled
 no mop sysid
end
```

This is formatted text, not structured data

Data Encoding Formats: JSON and XML



*Machines can **easily** parse XML and JSON. You can easily send an object that a machine understands.*

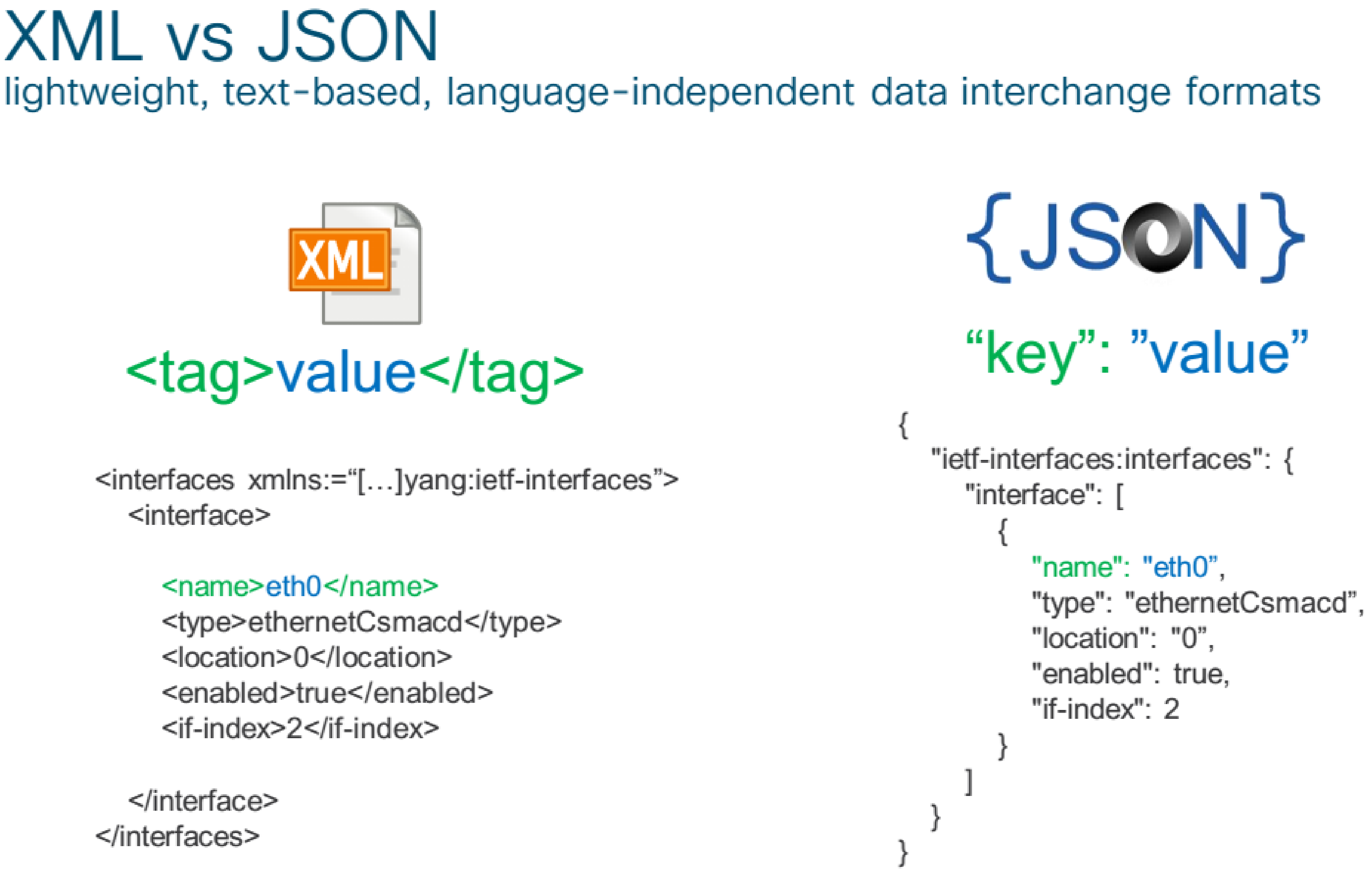
```
{
  "Cisco-IOS-XE-native:GigabitEthernet":
{
        "name": "1",
        "vrf": {
            "forwarding": "MANAGEMENT"
        },
        "ip": {
            "address": {
                "primary": {
                    "address":"10.0.0.151",
                    "mask": "255.255.255.0"
                }
            }
        },
        "mop": {
            "enabled": false,
            "sysid": false
        }
    }
}
```

JSON

```
<GigabitEthernet>
  <name>1</name>
  <vrf>

<forwarding>MANAGEMENT</forwarding>
  </vrf>
  <ip>
    <address>
      <primary>
        <address>10.0.0.151</address>
        <mask>255.255.255.0</mask>
      </primary>
    </address>
</ip>
<mop>
  <enabled>false</enabled>
  <sysid>false</sysid>
</mop>
</GigabitEthernet>
```

XML

Encoding Formats



XML vs JSON

lightweight, text-based, language-independent data interchange formats

<tag>value</tag>

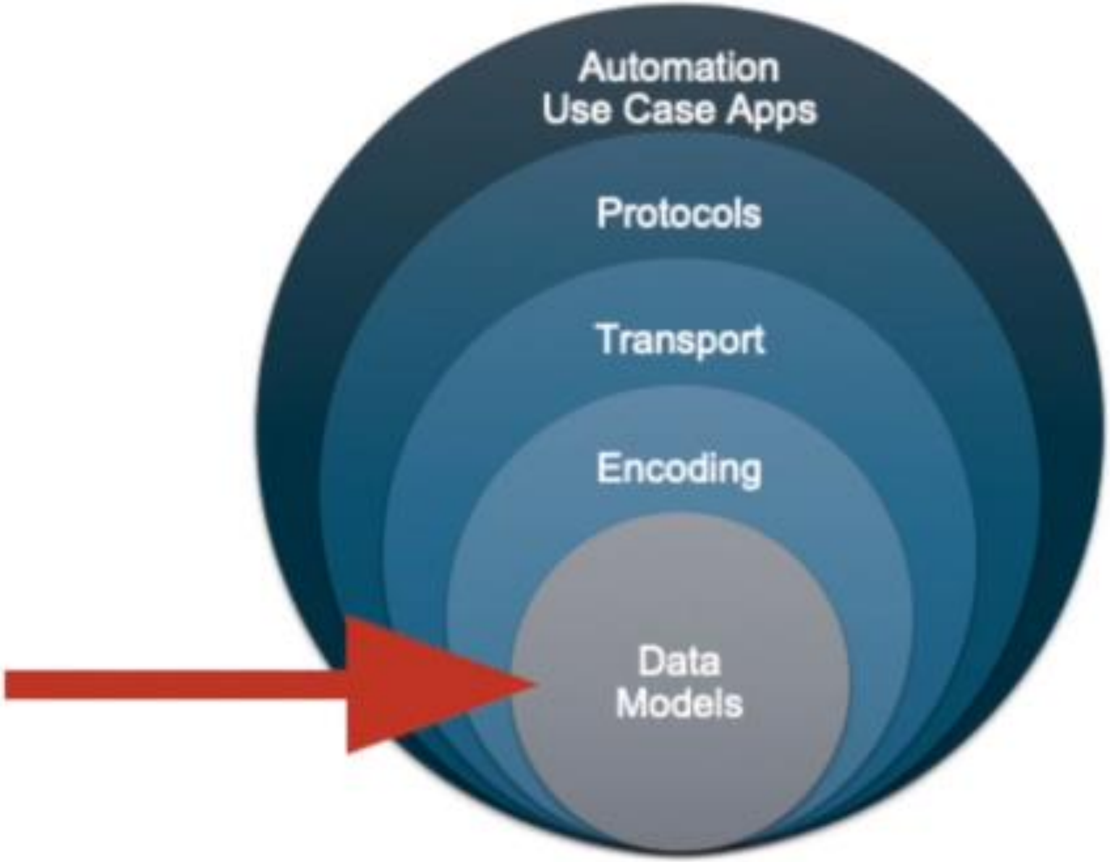
```
<interfaces xmlns:=“[…]yang:ietf-interfaces”>
   <interface>

      <name>eth0</name>
      <type>ethernetCsmacd</type>
      <location>0</location>
      <enabled>true</enabled>
      <if-index>2</if-index>

   </interface>
</interfaces>
```

“key”: ”value”

```
{
   "ietf-interfaces:interfaces": {
      "interface": [
         {
            "name": "eth0”,
            "type": "ethernetCsmacd”,
            "location": "0”,
            "enabled": true,
            "if-index": 2
         }
      ]
   }
}
```

Data Models

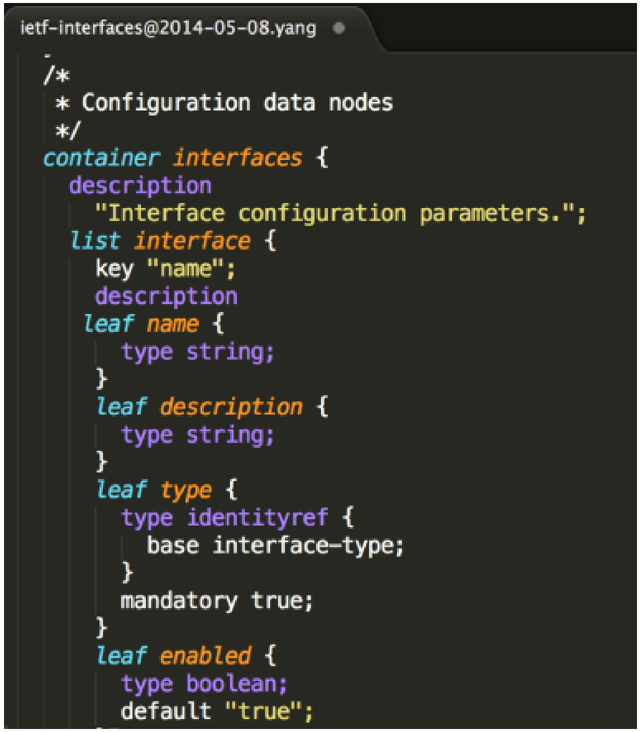
Now Here: Yang



Data Models

"A Data-Model Explicitly and precisely defines Data Structure, Syntax and Semantics"

Interface Model definition

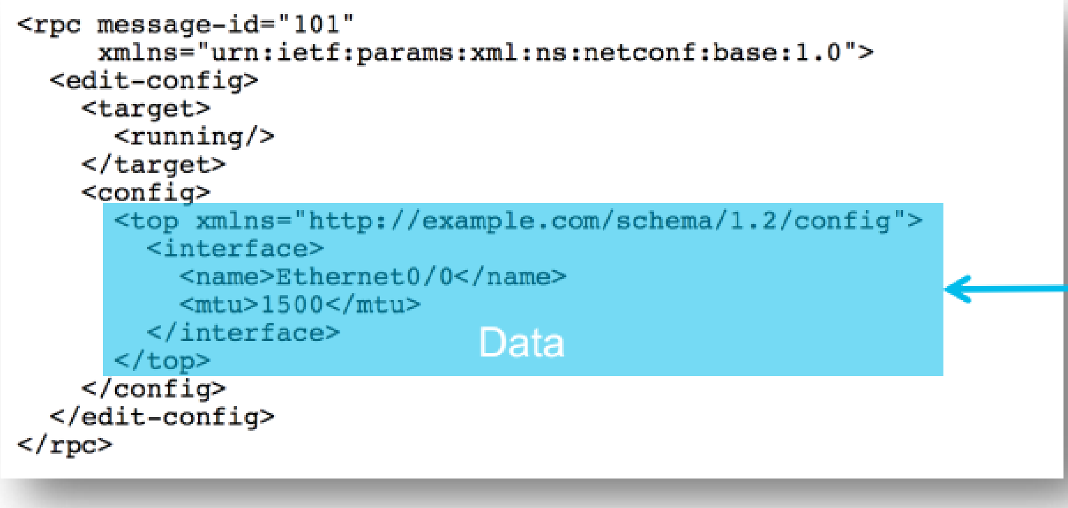


ietf-interfaces@2014-05-08.yang

```
/*
 * Configuration data nodes
 */
container interfaces {
  description
    "Interface configuration parameters.";
  list interface {
    key "name";
    description
   leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf type {
      type identityref {
        base interface-type;
      }
      mandatory true;
    }
    leaf enabled {
      type boolean;
      default "true";
```

YANG definition

"YANG - A Data Modeling Language for NETCONF"



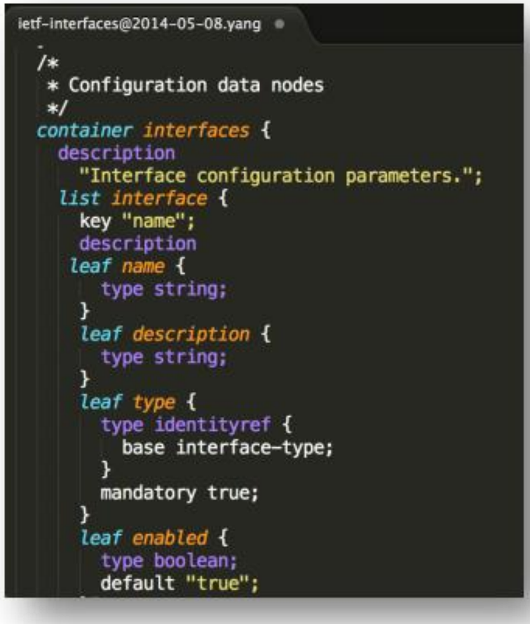
```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <top xmlns="http://example.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>
```

The Data is NOT defined by NETCONF!

- YANG describes how to structure the Data to send/receive
- Standard defined in RFC 6020

YANG Models Example

YANG Models



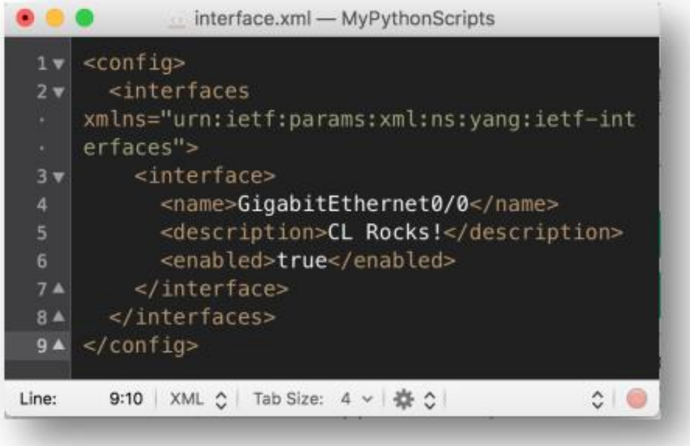
ietf-interfaces@2014-05-08.yang

```
/*
 * Configuration data nodes
 */
container interfaces {
  description
    "Interface configuration parameters.";
  list interface {
    key "name";
    description
   leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf type {
      type identityref {
        base interface-type;
      }
      mandatory true;
    }
    leaf enabled {
      type boolean;
      default "true";
```

Data

```
Gig 1/0/1
“CL rocks!”
enabled
```

XML Payload

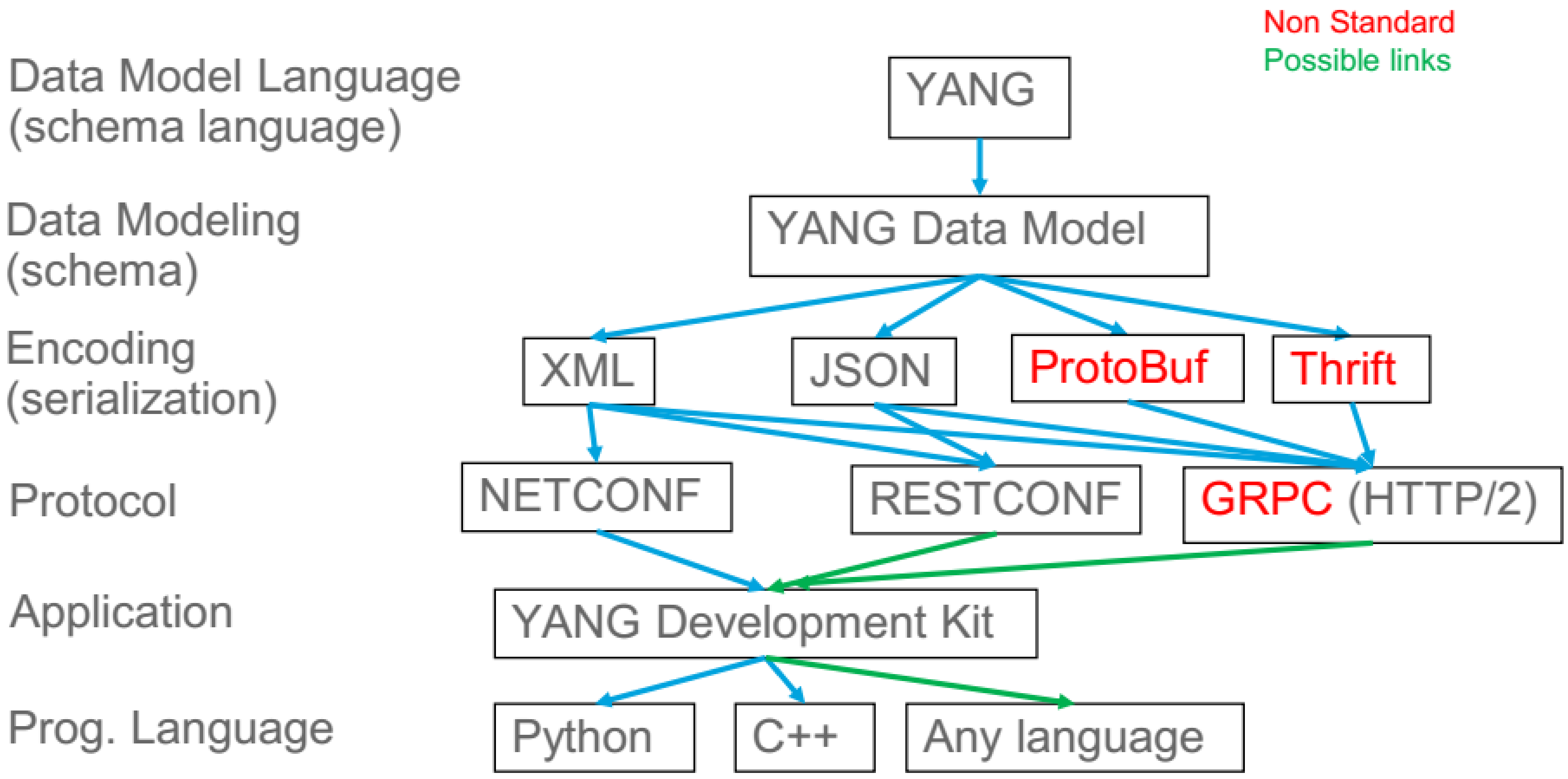


interface.xml — MyPythonScripts

```
<config>
  <interfaces
xmlns="urn:ietf:params:xml:ns:yang:ietf-int
erfaces">
    <interface>
      <name>GigabitEthernet0/0</name>
      <description>CL Rocks!</description>
      <enabled>true</enabled>
    </interface>
  </interfaces>
</config>
```

YANG Models → Data Models defined using the YANG language

Data Model-Driven Management



Non Standard

Possible links

Data Model Language (schema language): YANG

Data Modeling (schema): YANG Data Model

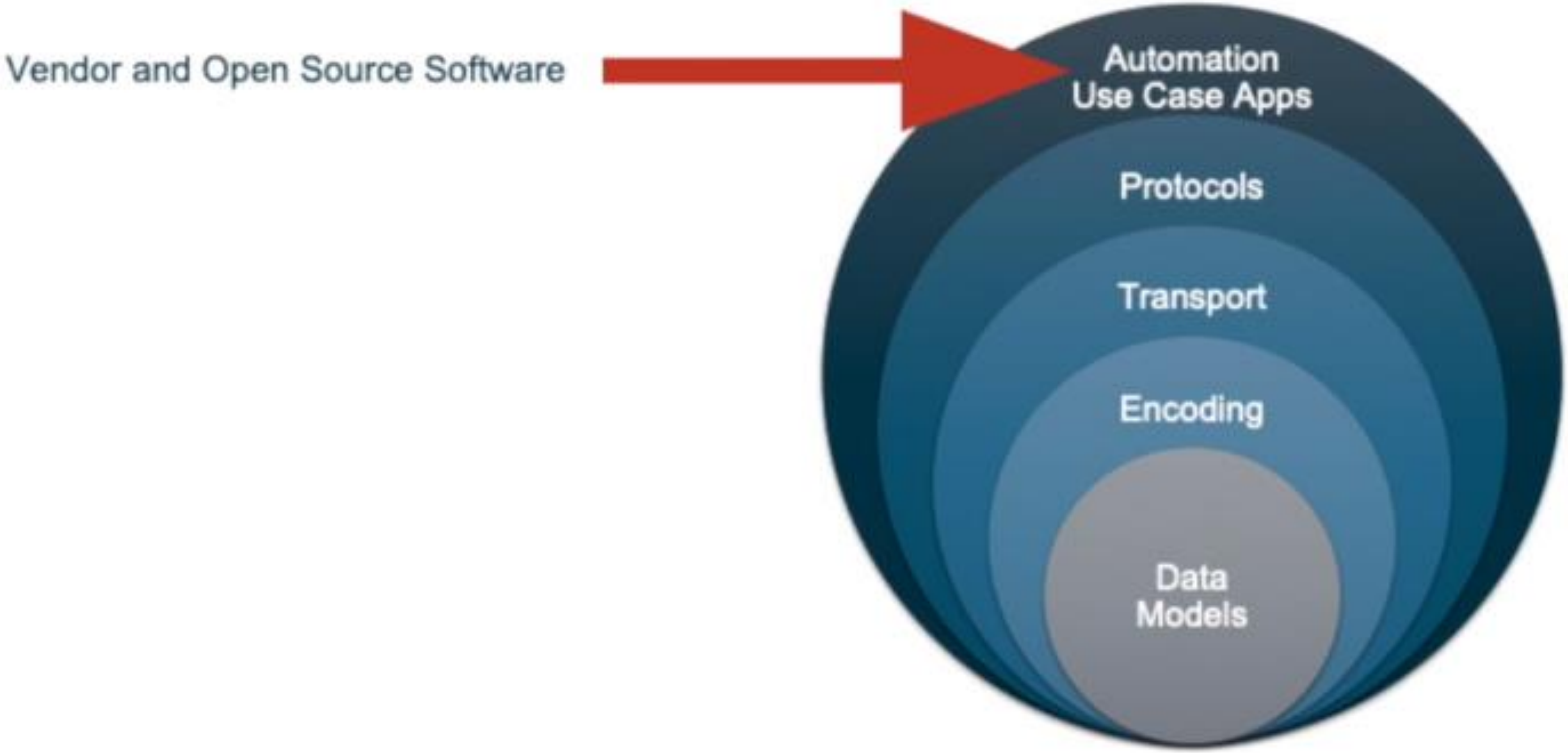
Encoding (serialization): XML, JSON, ProtoBuf, Thrift

Protocol: NETCONF, RESTCONF, GRPC (HTTP/2)

Application: YANG Development Kit

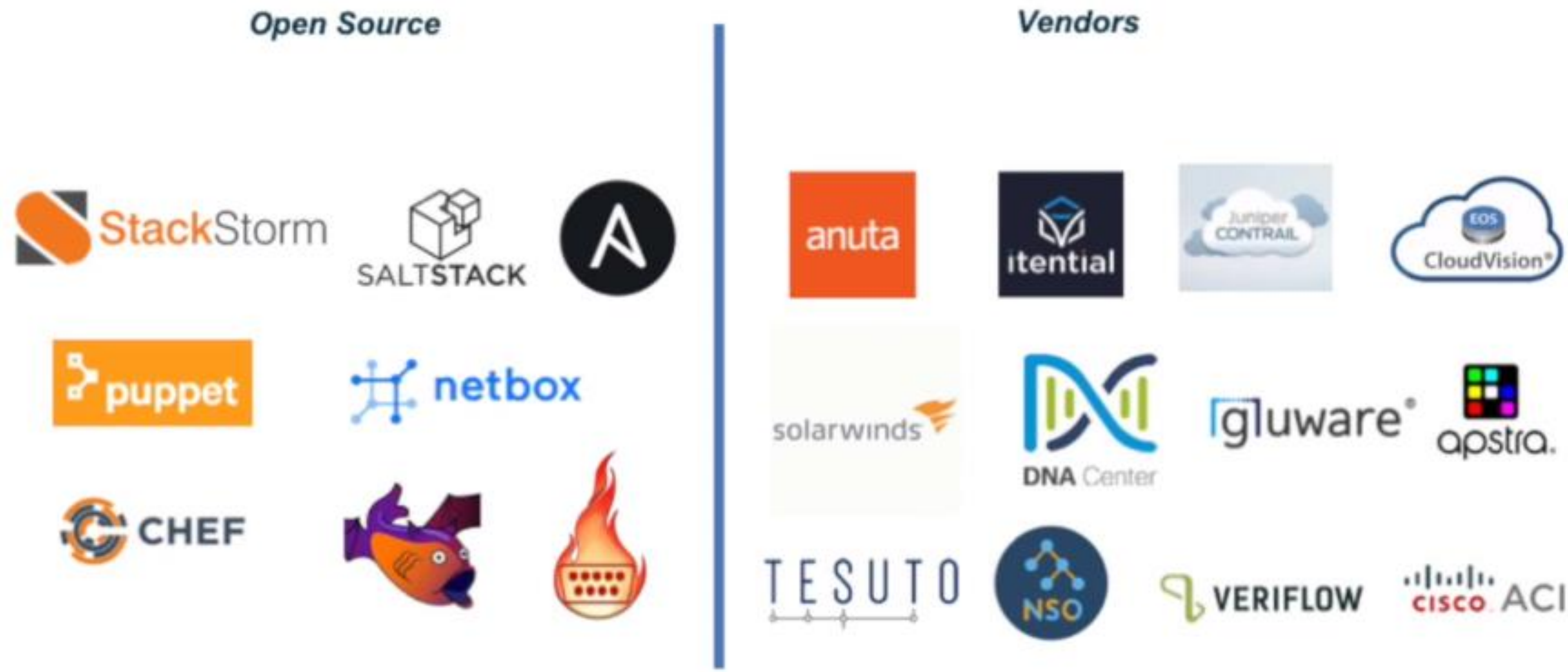
Prog. Language: Python, C++, Any language

Automation Application



Vendor and Open Source Software

Putting it All Together: Vendor Sorfware

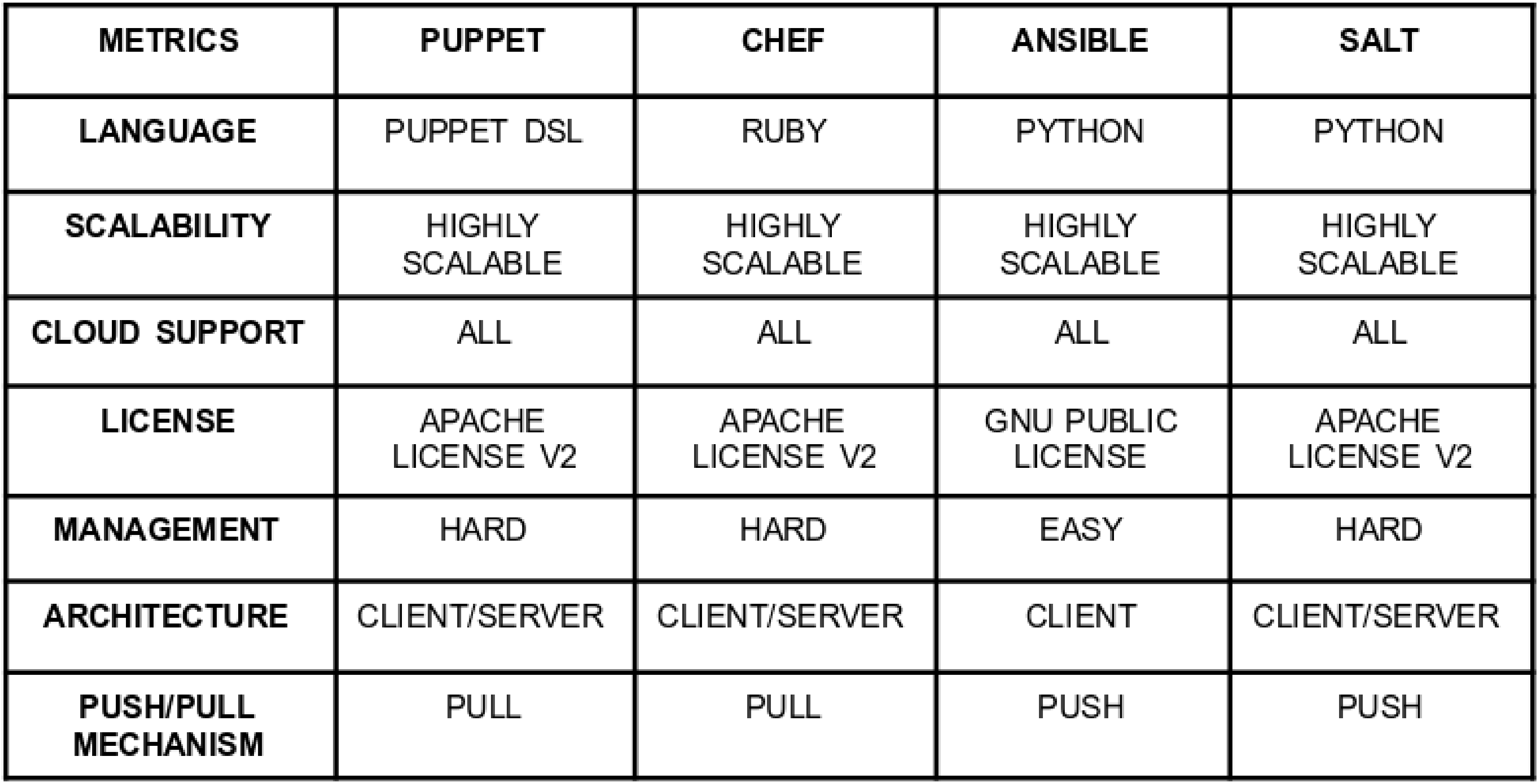


Configuration Management

Overview of Configuration Management

- Configuration management tools help manage infrastructure at scale. Consider the challenges of managing large data centers:
  - Updating packages on thousands of virtual machines.
  - Changing configuration files on hundreds of servers.
  - Orchestrating a workflow such as the deployment of a new application to production across different data centers.
  - Running multiple CLI commands on dozens of servers to retrieve operational data.
- All of this is prone to various human error. As such, configuration management tools provide features that are useful to solve problems such as these.

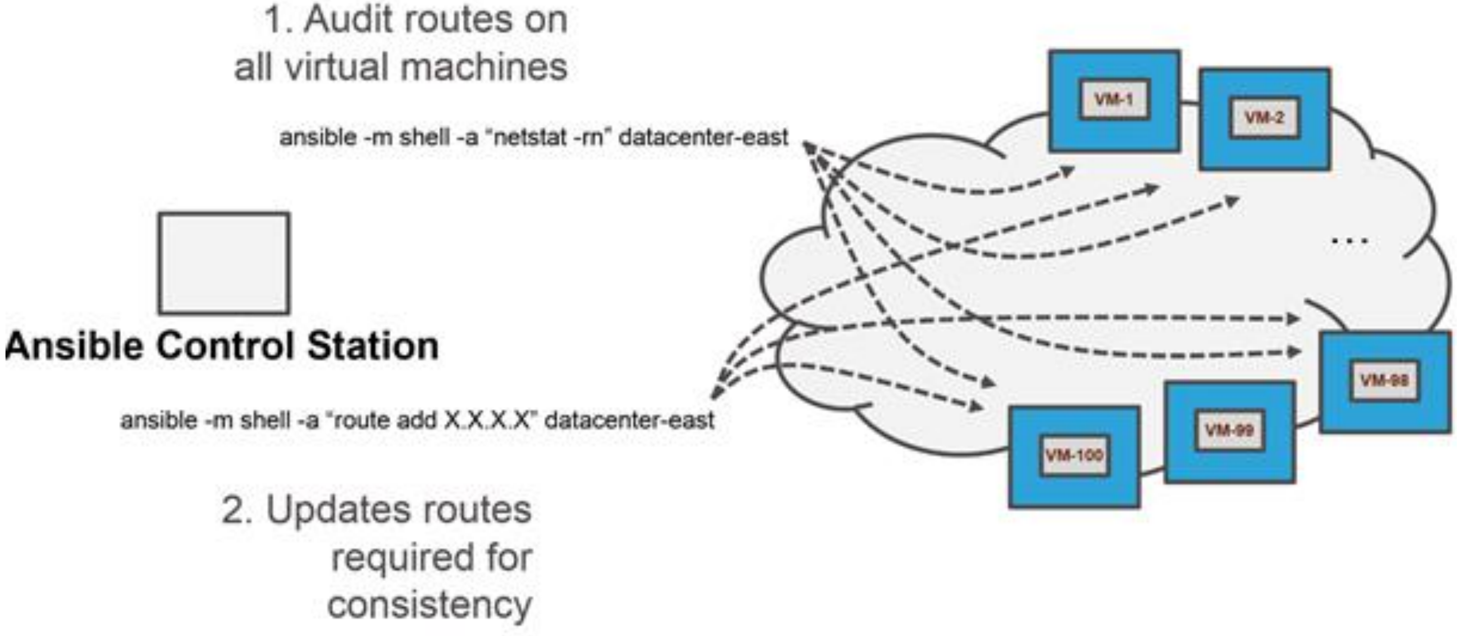
Comparison of Configuration Management Tools



METRICS	PUPPET	CHEF	ANSIBLE	SALT
LANGUAGE	PUPPET DSL	RUBY	PYTHON	PYTHON
SCALABILITY	HIGHLY SCALABLE	HIGHLY SCALABLE	HIGHLY SCALABLE	HIGHLY SCALABLE
CLOUD SUPPORT	ALL	ALL	ALL	ALL
LICENSE	APACHE LICENSE V2	APACHE LICENSE V2	GNU PUBLIC LICENSE	APACHE LICENSE V2
MANAGEMENT	HARD	HARD	EASY	HARD
ARCHITECTURE	CLIENT/SERVER	CLIENT/SERVER	CLIENT	CLIENT/SERVER
PUSH/PULL MECHANISM	PULL	PULL	PUSH	PUSH

What is Ansible?

- **Ansible is open source software that automates software provisioning, configuration management, and application deployment, based upon an agentless architecture.**
- **Hosts are managed by an Ansible control machine via SSH.**



1. Audit routes on all virtual machines

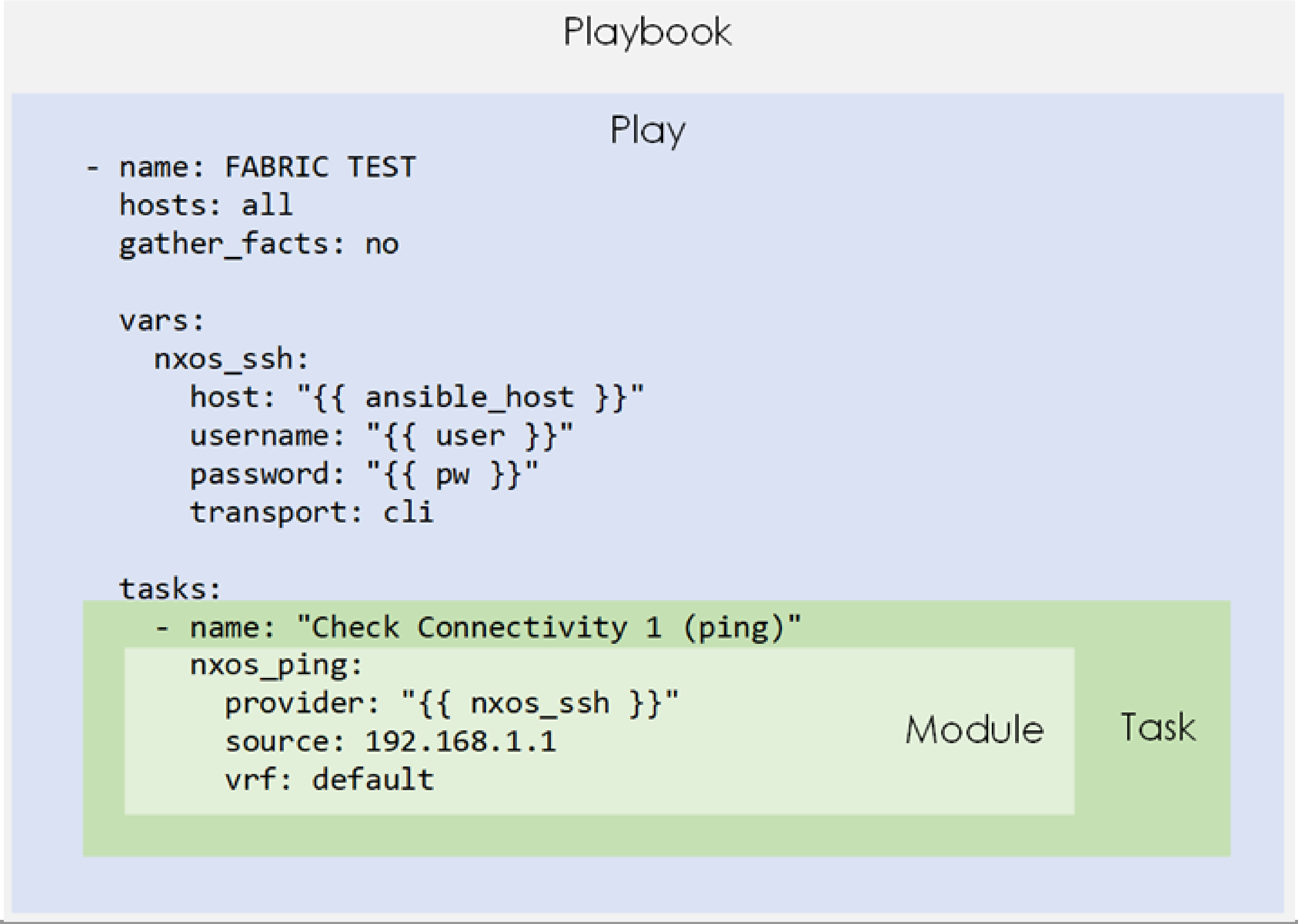
```
ansible -m shell -a "netstat -rn" datacenter-east
```

Ansible Control Station

```
ansible -m shell -a "route add X.X.X.X" datacenter-east
```

2. Updates routes required for consistency

Ansible Components



```
- name: FABRIC TEST
  hosts: all
  gather_facts: no

  vars:
    nxos_ssh:
      host: "{{ ansible_host }}"
      username: "{{ user }}"
      password: "{{ pw }}"
      transport: cli

  tasks:
    - name: "Check Connectivity 1 (ping)"
      nxos_ping:
        provider: "{{ nxos_ssh }}"
        source: 192.168.1.1
        vrf: default
```

cisco