Good Morning!

Class starts 9:10 AM Sydney Time

(Today’s Topic: AVD!)

Morning Break

- Back at 11:50 AM Sydney Time

Quick Break!

- Back! 11:52

Lunch Break

- Back at 1:15 pm (13:15) Sydney Time

Afternoon Break

- Back at 2:32 pm (14:32) Sydney Time

Lab Time: Day 1

- https://l5.labs.sdn-pros.com/
  - Username: level5
  - Password: arista
- Labs 1, 2, 3
- If you need to sign off, and want to do labs later that’s OK
- Zoom is open to 17:00 (5PM) Sydney Time for proctored lab assistance
- Reconvene Tomorrow at 9:00 AM Sydney Time

Lab Time: Day 2

- https://l5.labs.sdn-pros.com/
  - Username: level5
  - Password: arista
- Previous day Labs 1, 2, 3
- Current Day Labs: 4, 5
- If you need to sign off, and want to do labs later that's OK
- Zoom is open to 17:00 (5PM) Sydney Time for proctored lab assistance
- Reconvene Tomorrow at 9:00 AM Sydney Time

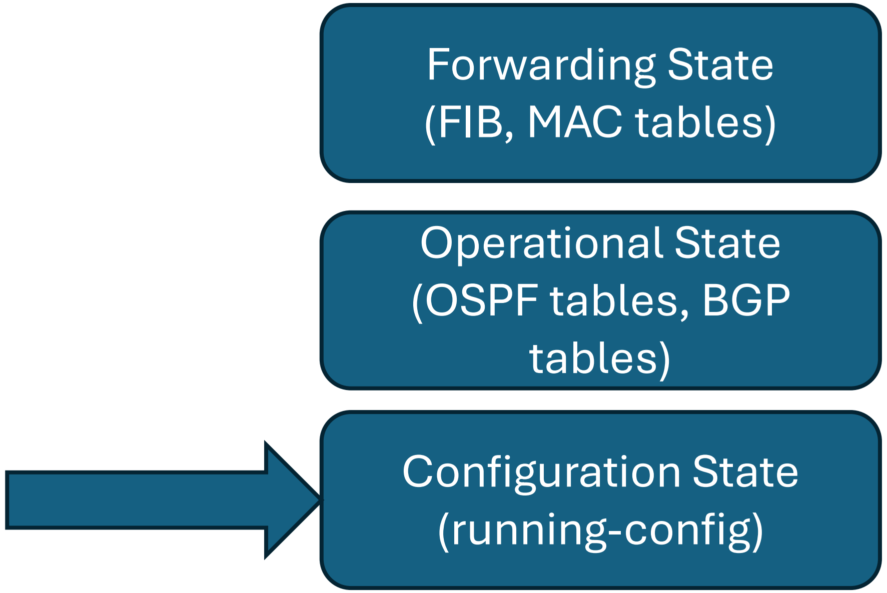
Lab Time: Day 3

- https://l5.labs.sdn-pros.com/
  - Username: level5
  - Password: arista
- Previous day Labs 1, 2, 3, 4, 5
- Current Day Labs: 6, 7, 8, 9
- If you need to sign off, and want to do labs later that's OK
- Zoom is open to 17:00 (5PM) Sydney Time for proctored lab assistance
- Reconvene Tomorrow at 9:00 AM Sydney Time

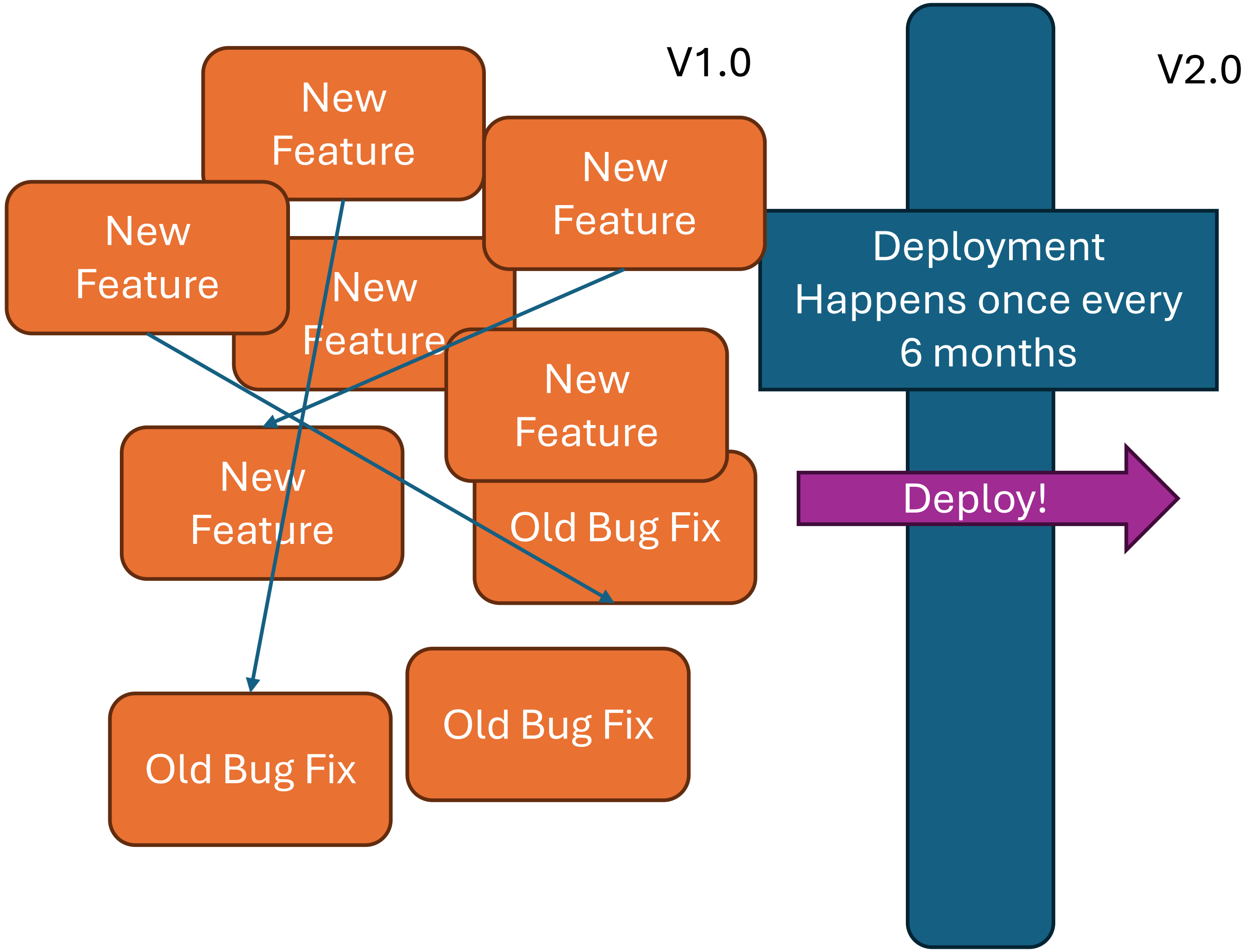
Lab Time: Day 4

- https://l5.labs.sdn-pros.com/
  - Username: level5
  - Password: arista
- Previous day Labs 1, 2, 3, 4, 5, 6, 7, 8, 9
- Today’s Labs 10, 11, 12, 13, 14, 15, 16
- Current Day Labs: If you need to sign off, and want to do labs later that’s OK
- Zoom is open to 17:00 (5PM) Sydney Time for proctored lab assistance
- Reconvene Tomorrow at 9:00 AM Sydney Time

Configuration States



“Waterfall” Development



V1.0

V2.0

New Feature

New Feature

New Feature

New Feature

New Feature

New Feature

Old Bug Fix

Old Bug Fix

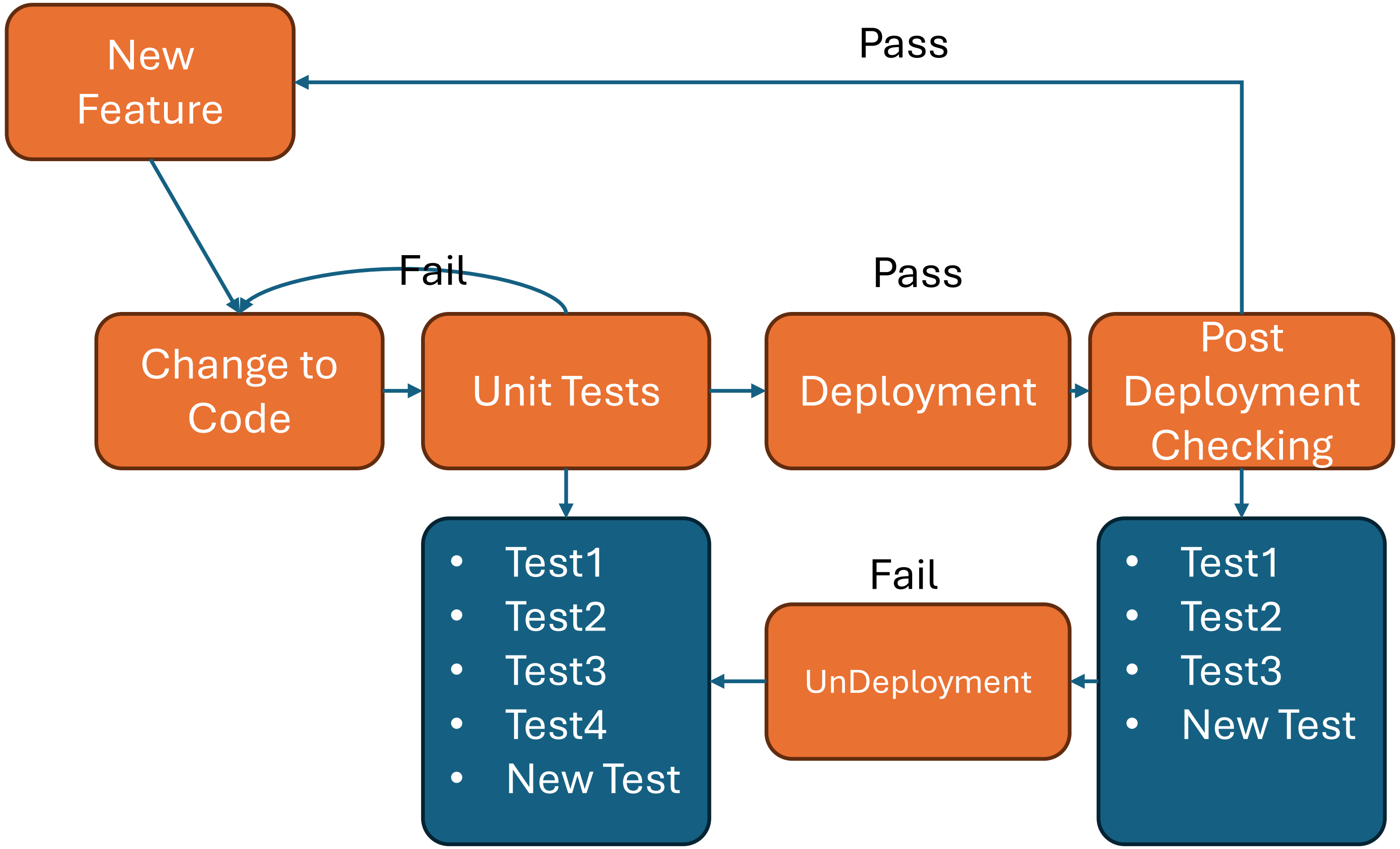
Old Bug Fix

Deployment Happens once every 6 months

Deploy!

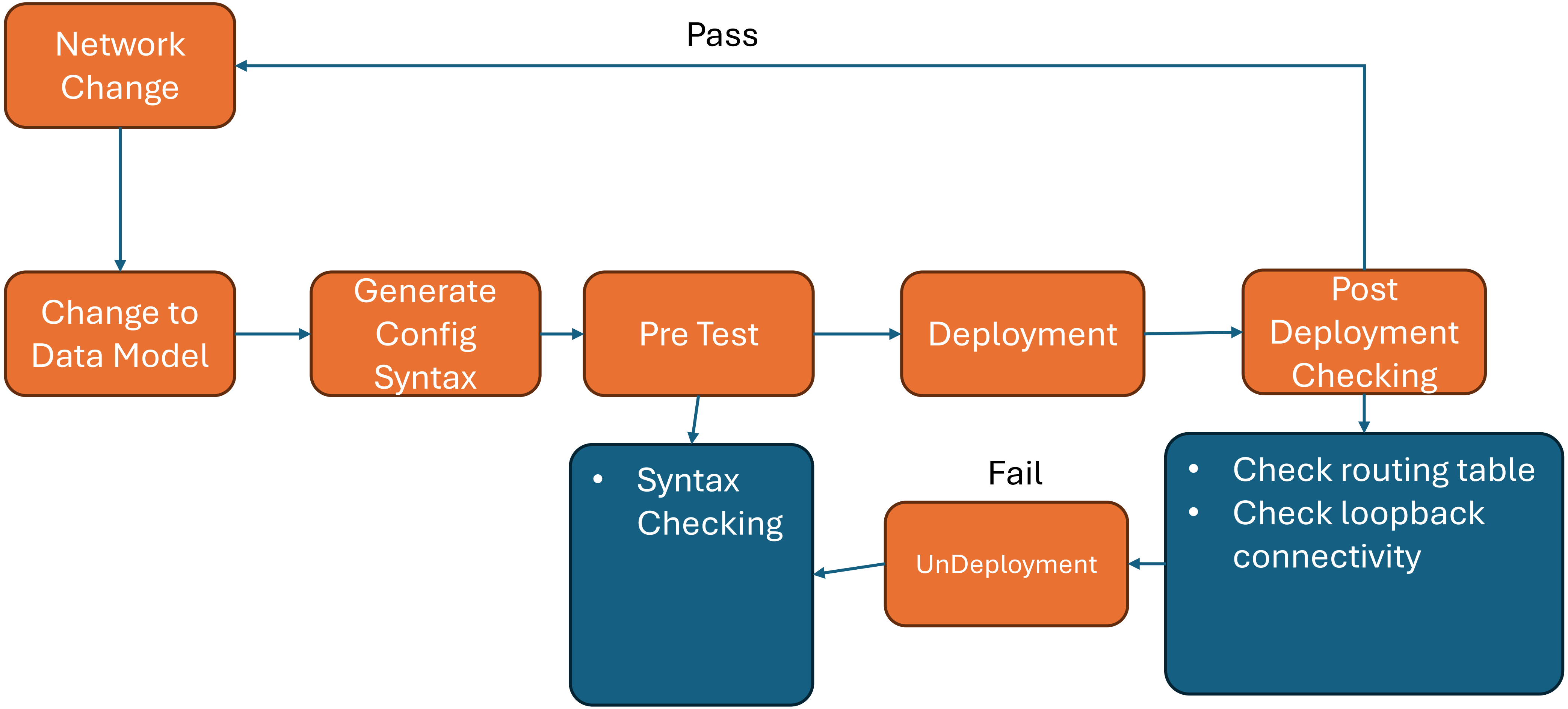
CI/CD 101

Continuous **I**ntegration/[**C**ontinuous **D**elivery/**D**eployment]



Continuous **I**ntegration/[**C**ontinuous **D**elivery/**D**eployment]

CI/CD Networking 101



Network Change

Pass

Change to Data Model

Generate Config Syntax

Pre Test

Deployment

Post Deployment Checking

- Syntax Checking

Fail

UnDeployment

- Check routing table
- Check loopback connectivity

Historical Automation

- No automation (wired campus, DC)
    - Some automation SP
    - All wireless has been automated
- On-box automation (rudimentary)

Server Automation Evolution

- No automation
- Perl scripts (1990s)
  - On-box
  - Remote Perl scripts over SSH (ssh keys)
- Automation Frameworks (mid 2000s)
  - Puppet Labs
  - Chef
  - Saltstack

Why No Automation?

- We lacked tools
  - Didn’t have good software frameworks (Ansible, Nornir, CloudVision)
  - Didn’t have remote configuration options (no APIs)
  - We didn’t have good methods

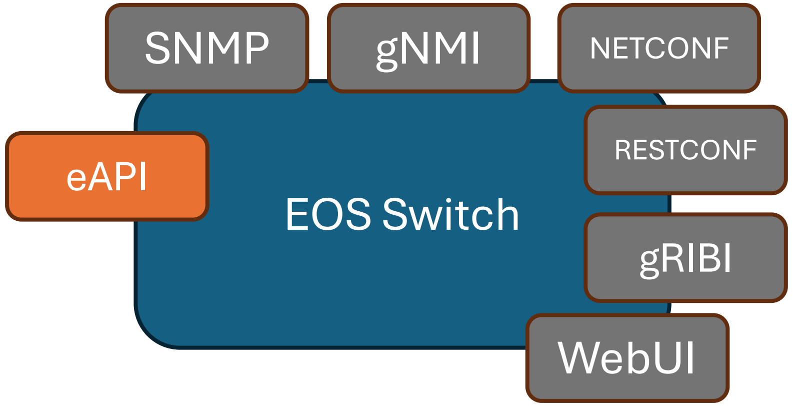
Structured Data Formats

- YAML
  - To represent desired state
  - Abstracted network state
  - Ansible file format for playbooks
  - AVD data model format
  - Jinja template data model format
- JSON
  - To interact with a device
  - Give a device new information
  - Query a device for existing information

APIs

- An API is a programmatic interface (meant for machines)
  - Query a device’s state
  - Set a device’s state
  - CRUD (Create/Read/Update/Delete)
- REST API (Representative State Transfer)
- Vendor specific APIs (**eAPI for Arista**, NX-API for NX-OS/Cisco)
- Vendor neutral APIs
  - **REST API**
  - gNMI (OpenConfig)
  - NETCONF/RESTCONF

APIs on Network Devices



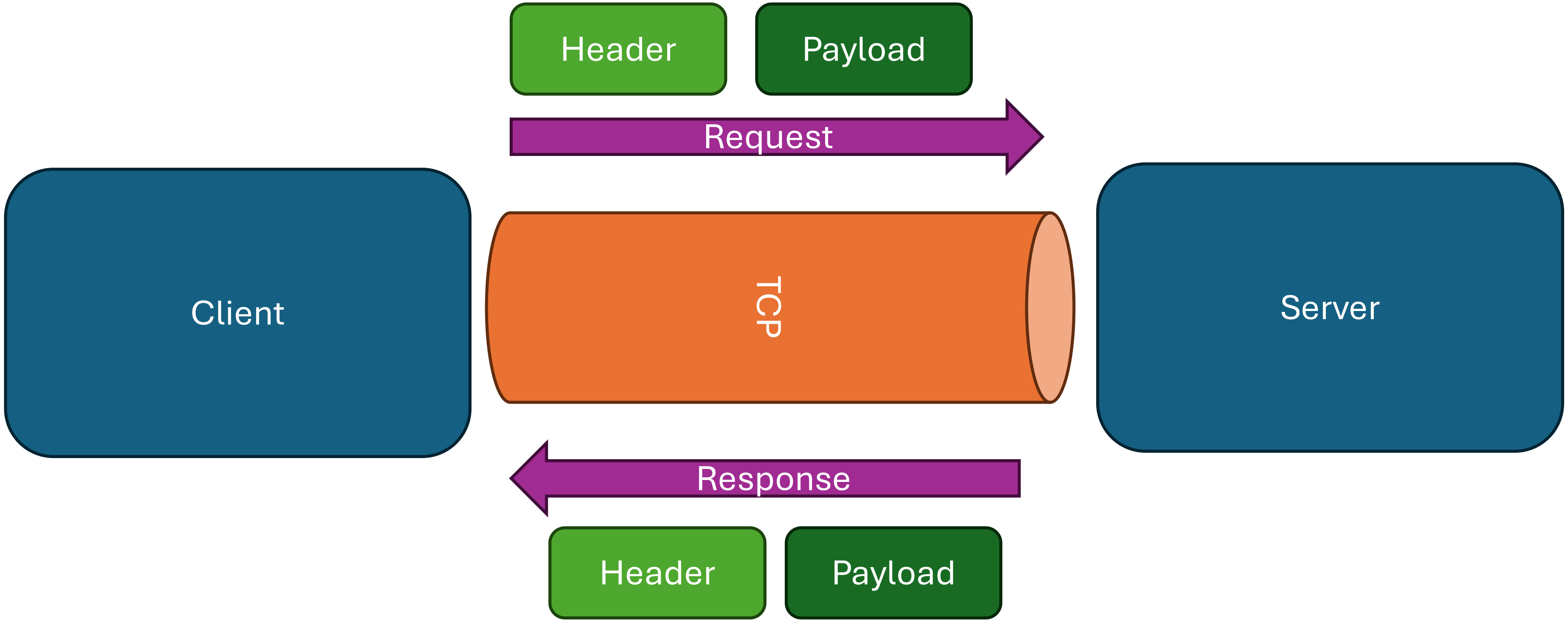
Types of APIs

- REST API (CloudVision Portal)
- eAPI (JSON-RPC)
- gNMI (gRPC)
- TerminAttr (gRPC, Arista specific)
- NX-API (XML-RPC)

HTTP 1.0/1.1

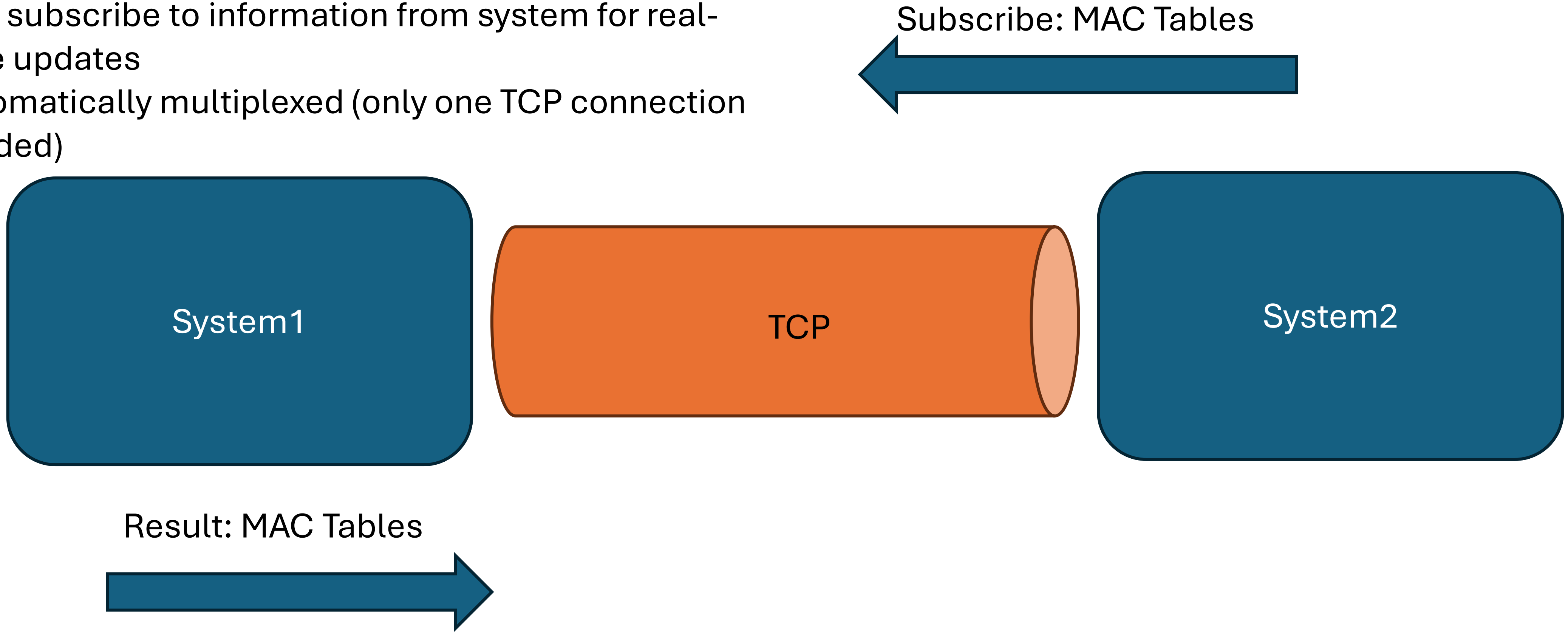
HTTP PDU: HTTP Message

- Request (Header/Payload)
- Response (Header/Payload)



HTTP 2.0

Any side can initiate TCP connection
Any side can request the other side (and get a response)
Can subscribe to information from system for real-time updates
Automatically multiplexed (only one TCP connection needed)



Encodings

- XML
- JSON
- YAML
- Google Protocol Buffers (GPB)
  - Binary
  - Efficient (CPU, network utilization)

HTTP Request

Header:
Request type (GET, PUT, UPDATE, DELETE, POST)
Agent: Chrome

Payload:
Some JSON data
Some XML data
Some YAML

HTTP Request

Header:
Request type (GET, **PUT**, UPDATE, DELETE, POST)
Agent: Chrome

Payload:

```
{ fname: “James”,
  lname: “Kirk”,
  mi:    “T”,
  rank:  “Captain”,
  shirt: “Gold”
}
```

HTTP Response

Header:

Response Code: 200

Payload:

```
{ result: ‘succeeded’
}
```

Automation Tools

- APIs: They allow us to reliably push configuration changes to a device
- Structured data formats
  - YAML, XML, JSON
- Automation frameworks
  - Ansible
- Git (version control)
  - Keep track of versions of files
  - Allows for multiple people to coordinate on files
- IDE (Integrated Development Environment)
  - VS Code

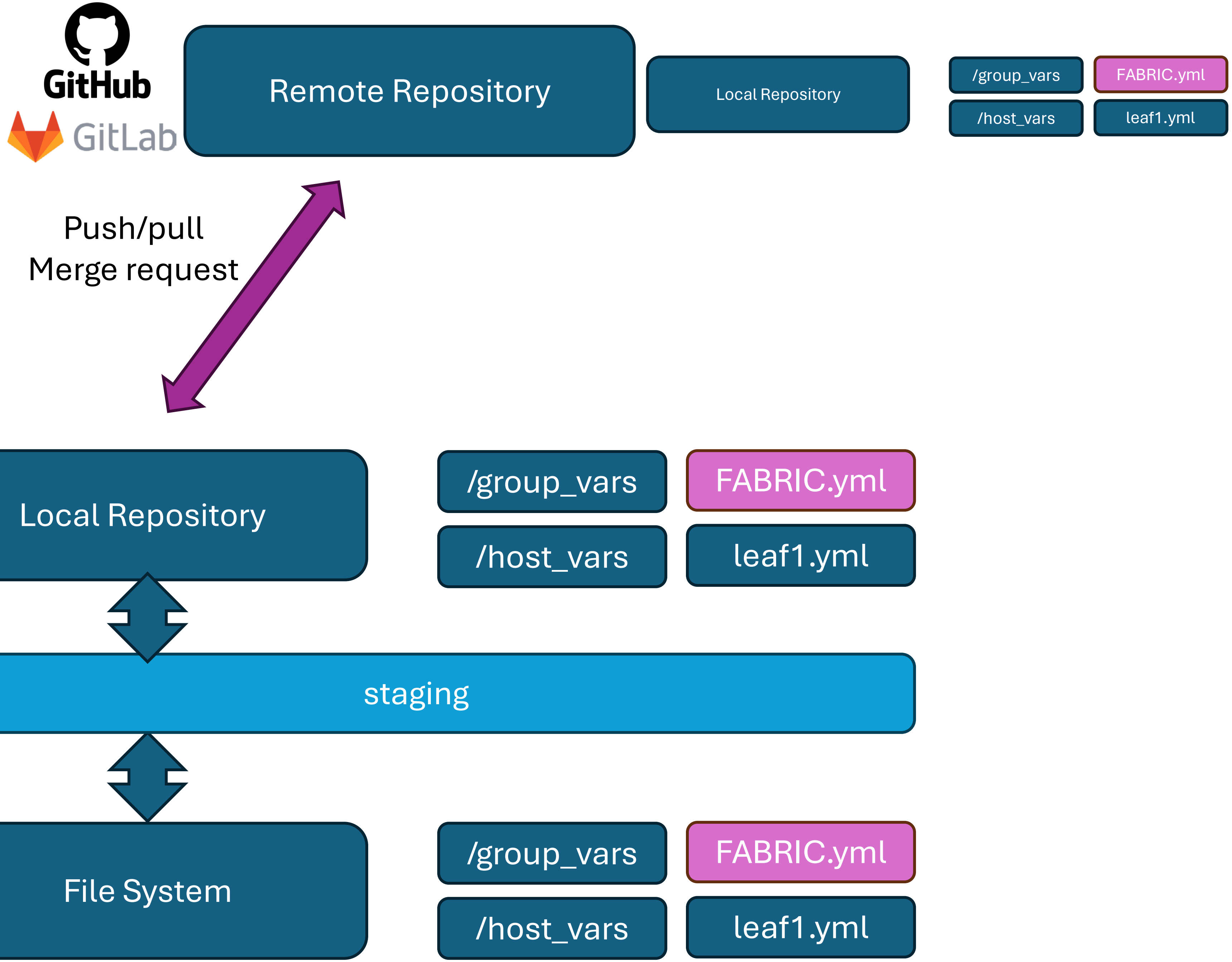
What is Git?

- Git is a version control system that allows multiple authors to collaborate on a code/files
- Git was created by Linux Torvalds (create of the Linux kernel)
  - Created 2005
  - Born of frustration with available tools (CVS, BitKeeper)

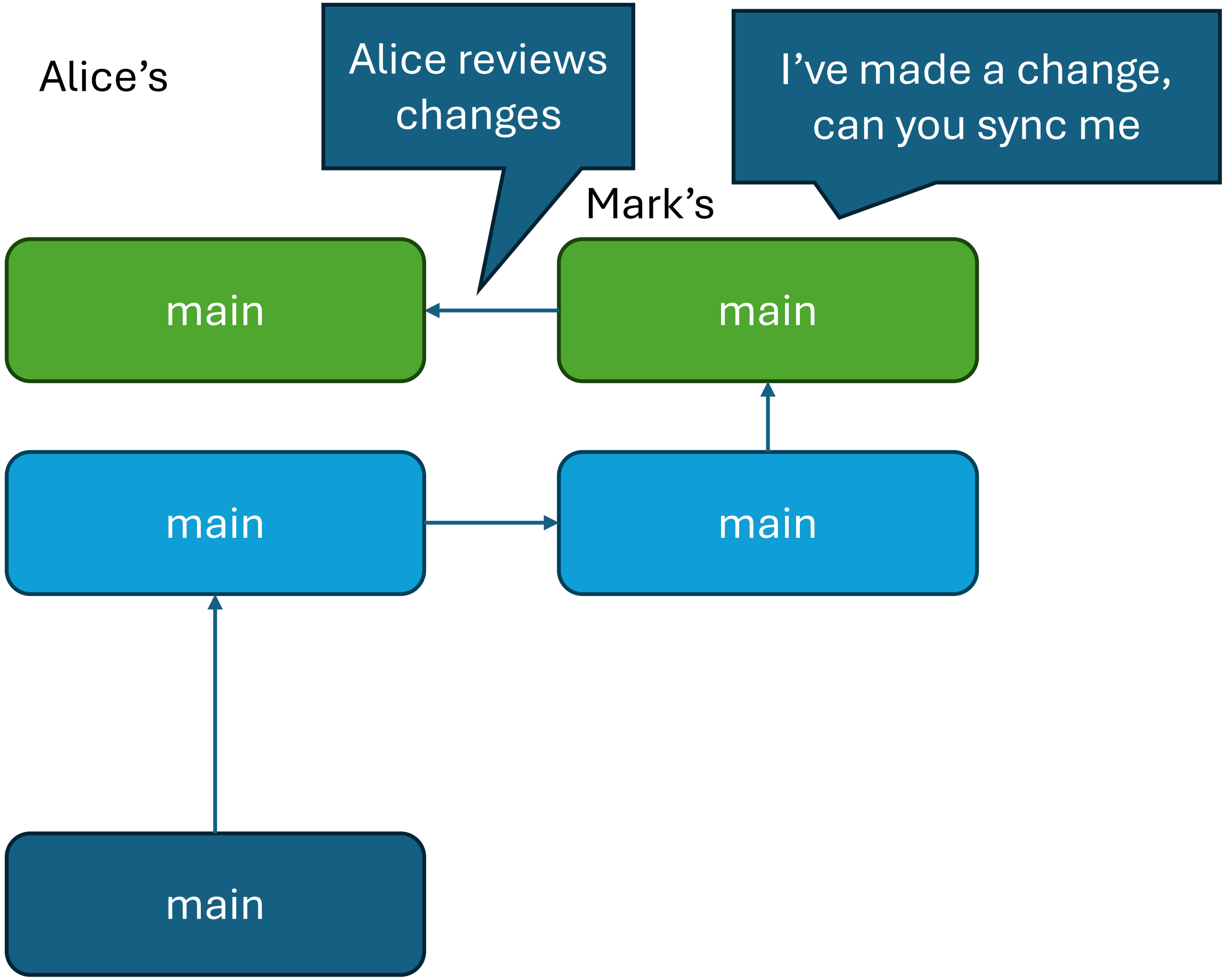
What Is It Used For?

- File tracking
- Collaboration
- Resolving conflicts (between version of files)
- Keeps you from having FABRIC_FINAL_FINALv2_FINAL_REALFINAL.yml
- Single Source of Truth

Basics of Git

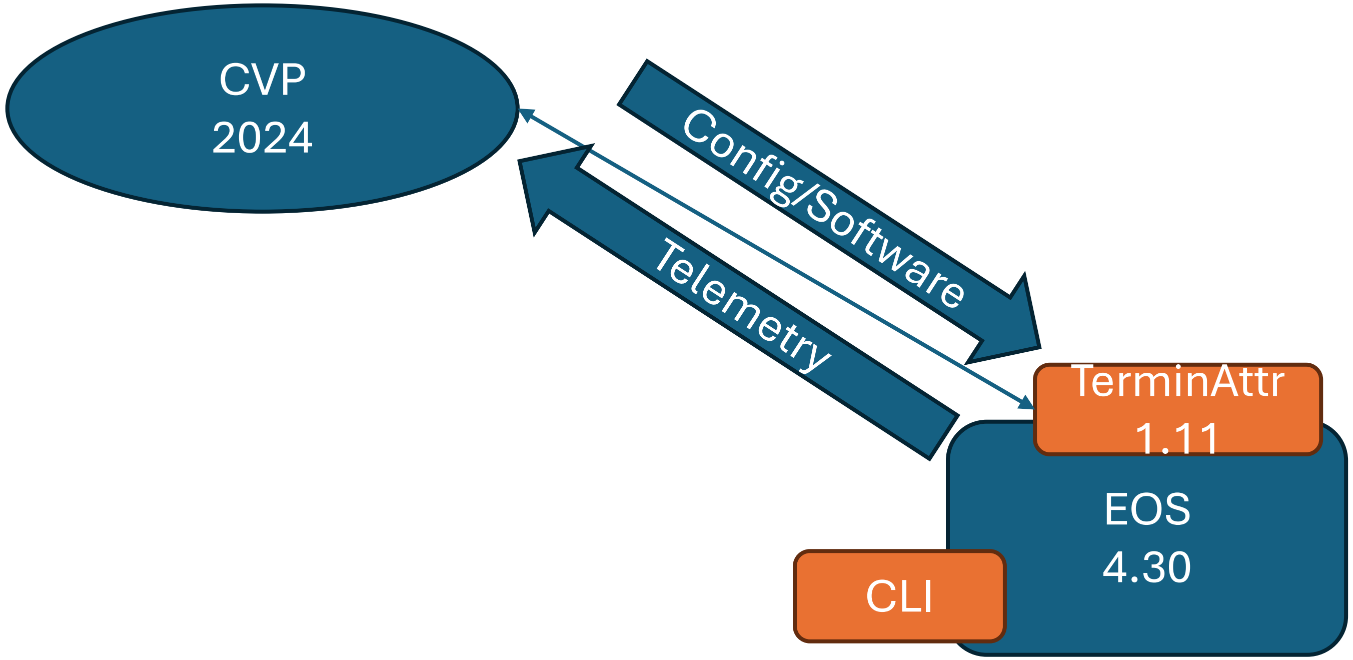


Forking

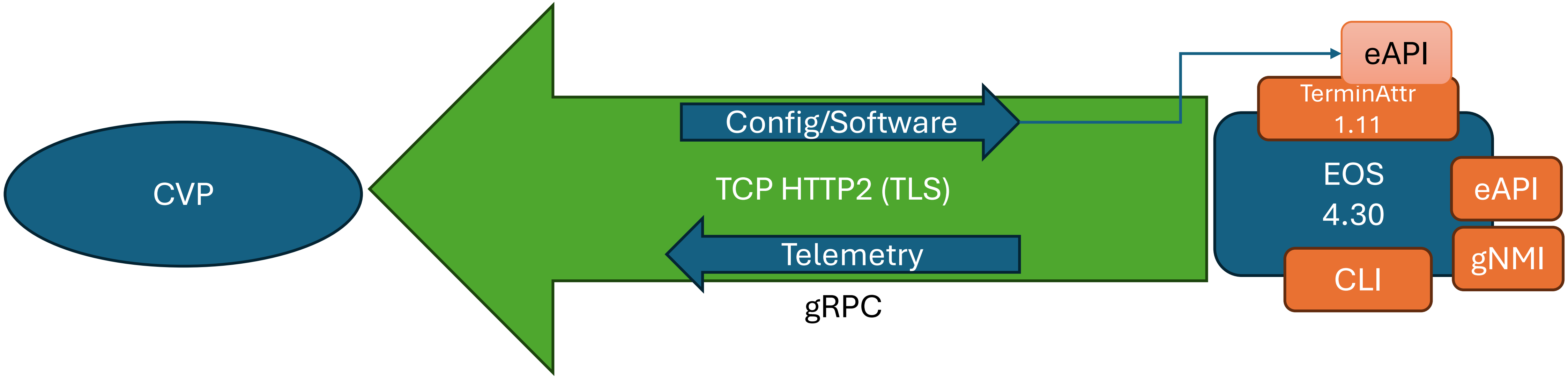


Alice's
Alice reviews changes
I've made a change, can you sync me
Mark's
main
main
main
main
main

What is TerminAttr?



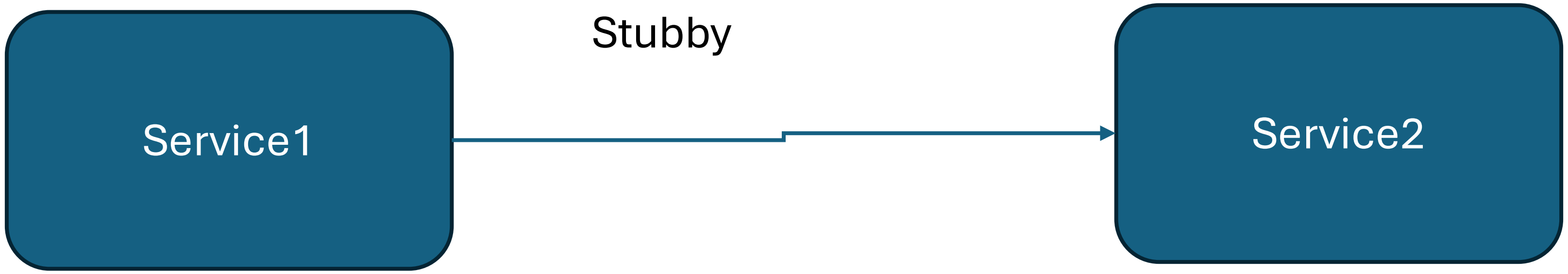
What is TerminAttr?



Google Protocol Buffers (protobuffs)

Using protobuffs
gNMI
gRIBI

~~RESTAPI~~
~~SOAP~~
~~JSON-RPC~~
~~XML-RPC~~
Stubby



HTTP2 (binary)
Protocol Buffers (binary)
gRPC

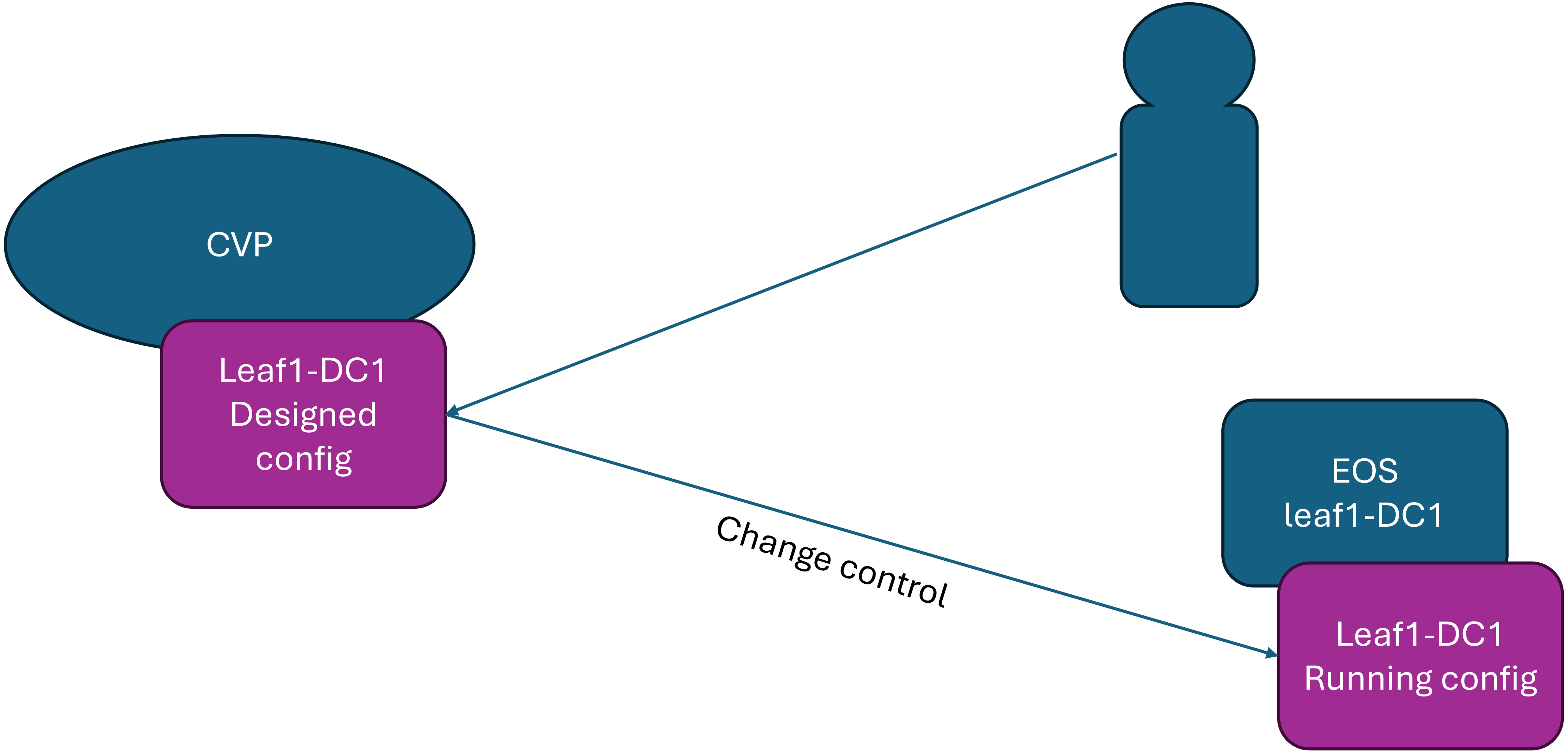
```
{ ship: “Enterprise”,
  registry: “NCC-1701”,
  captain: “Kirk” }

 1010101011010101010
```

What is a CVP Configlet?

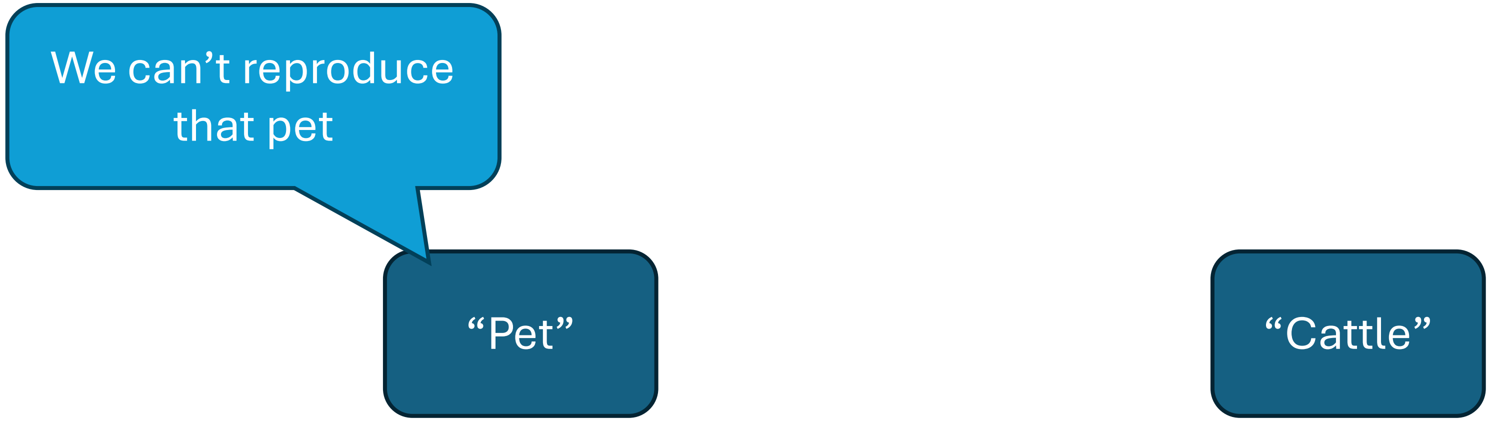
- A configlet is a piece of EOS syntax
- Configlets are applied to devices (switches)
- A single configlet can comprise the entire config of a device, but generally only represent a portion of the config
  - Normally multiple configlets are combined to produce a device's config
- When one or more configlets are combined to apply to a device, it's known as the **Designed Config**

CVP Config Management



Pets versus Cattle

“Designed Config"
OS: Ubuntu 22.04
Package list:
sshd_4.2
….



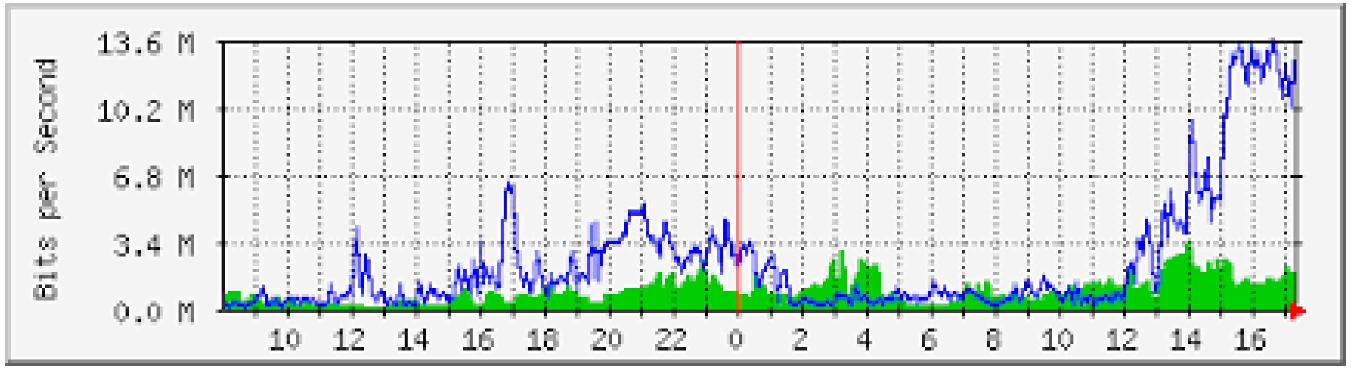
We can’t reproduce that pet

“Pet”

“Cattle”

Bandwidth Graphs SNMP

10:00 AM: 10000
10:05 AM: 100000

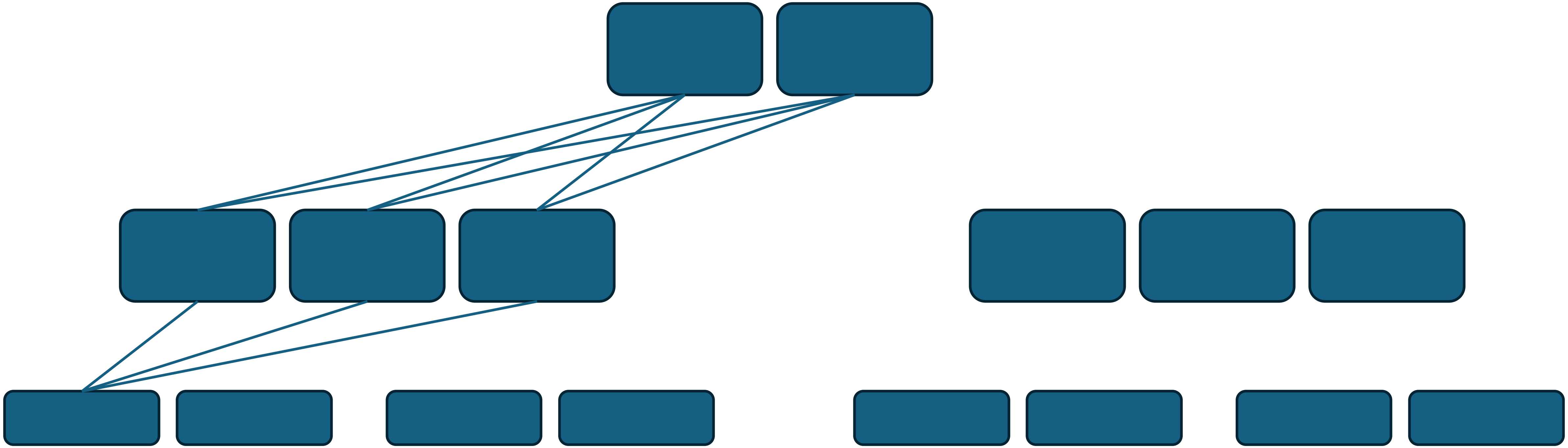


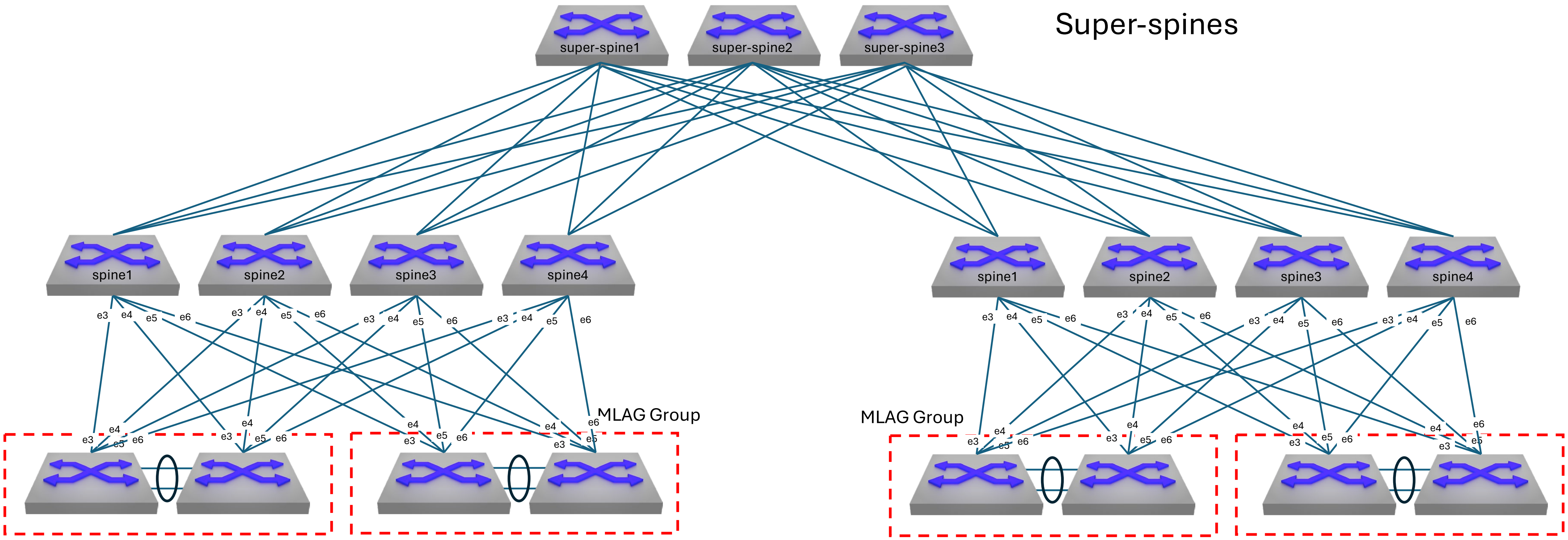
eth1

In: 10000
Out: 10000

90000 / 300 = 2.4 kilobits/second

DC Terminology



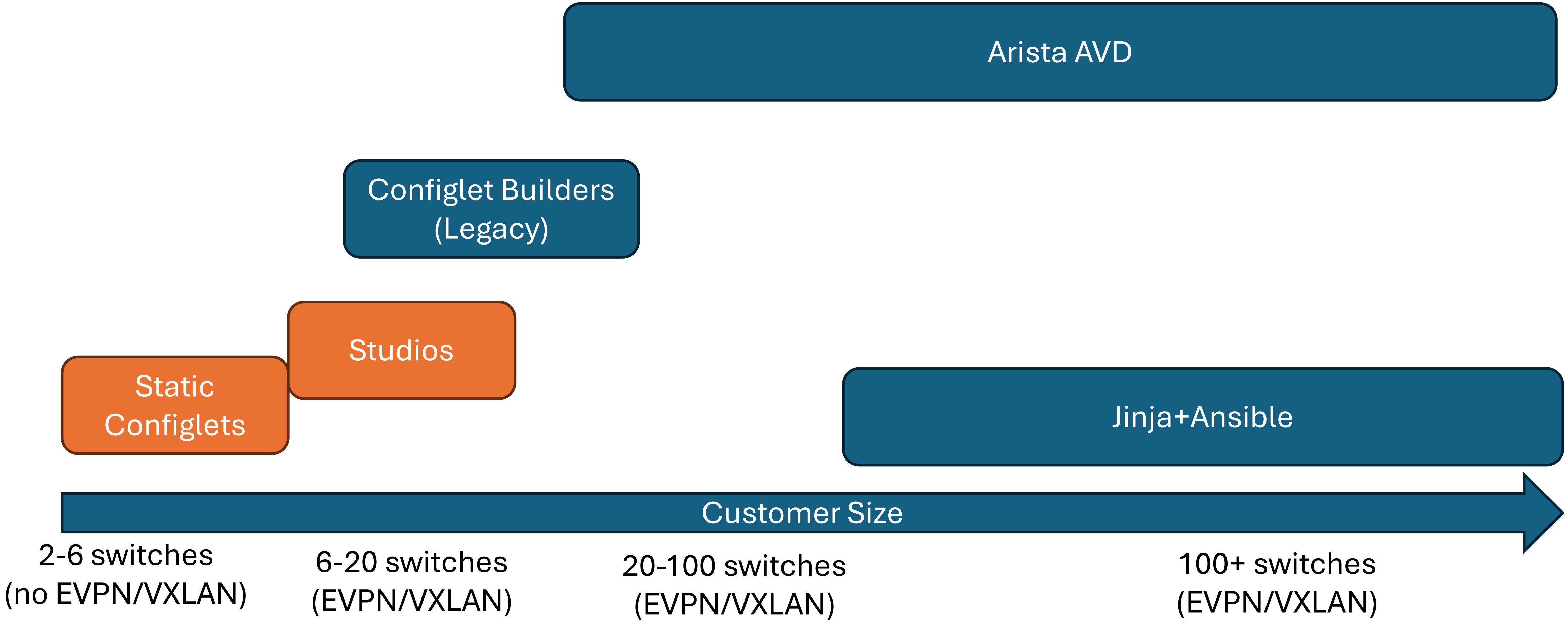

Super-spines
super-spine1
super-spine2
super-spine3
spine1
spine2
spine3
spine4
spine1
spine2
spine3
spine4
e3
e4
e5
e6
MLAG Group
MLAG Group

pod1

pod1

Use Cases

*These are general guidelines and not hard and fast rules


Arista AVD
Configlet Builders (Legacy)
Studios
Static Configlets
Jinja+Ansible
Customer Size
2-6 switches (no EVPN/VXLAN)
6-20 switches (EVPN/VXLAN)
20-100 switches (EVPN/VXLAN)
100+ switches (EVPN/VXLAN)

What Is A Templating System?

- Templates are documents with some static text and some dynamic text
- The dynamic text uses logic (if, then, loops) to fill in the blanks with various variables

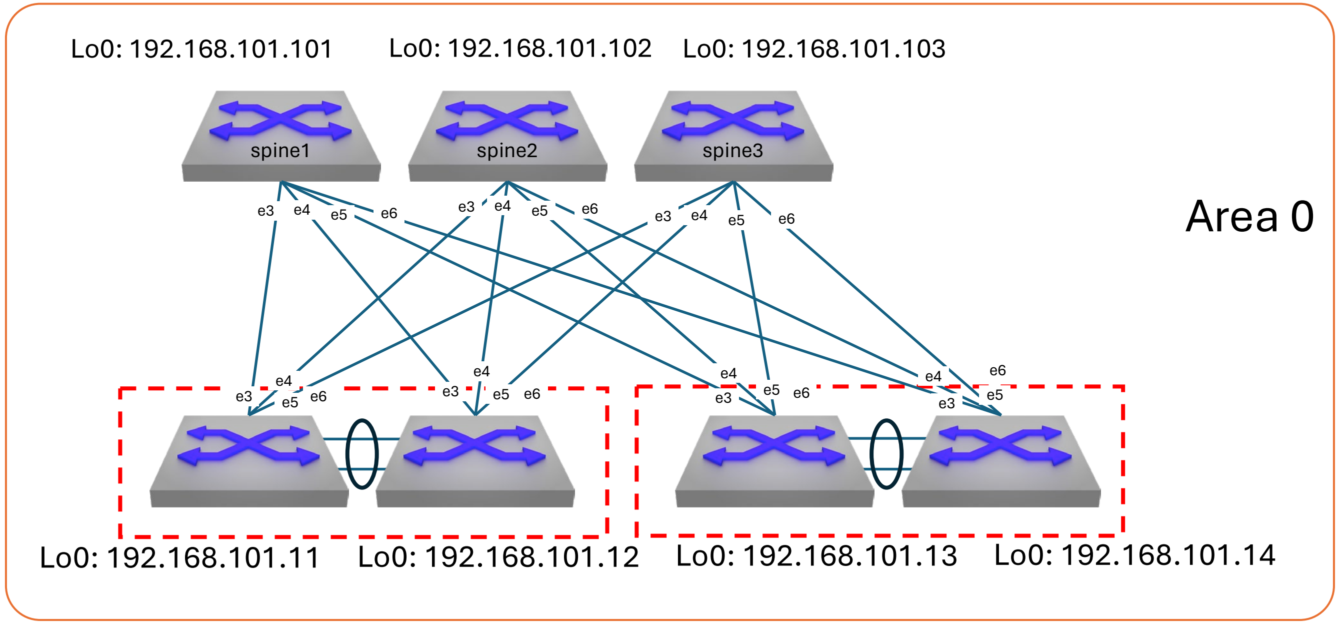
```
fave_colour: [‘Red’, ‘Blue’]
```

```
% for fave in fave_colour
% if fave == “Red” skip
% endif
One of my favorite colours is {fave}
% endfor
```

```
One of my favorite colours is Blue
```

Hesitation To Learn Programming Languages

- Automation will take away my job
- “I’m not a programmer”
- I’m too old to learn something new



Lo0: 192.168.101.101

Lo0: 192.168.101.102

Lo0: 192.168.101.103

Area 0

Lo0: 192.168.101.11

Lo0: 192.168.101.12

Lo0: 192.168.101.13

Lo0: 192.168.101.14

Python Skills

- Simple variables (integers, strings, Booleans)
- Complex variables (lists, dictionaries)
- For loops (iterating through lists or dictionaries)
- Working with strings
- Conditionals

Python Resources

- Learn Python the Hard Way (Zed Shaw)
  - https://learnpythonthehardway.org/
- Kirk Byers Python Course
  - https://pynet.twb-tech.com/

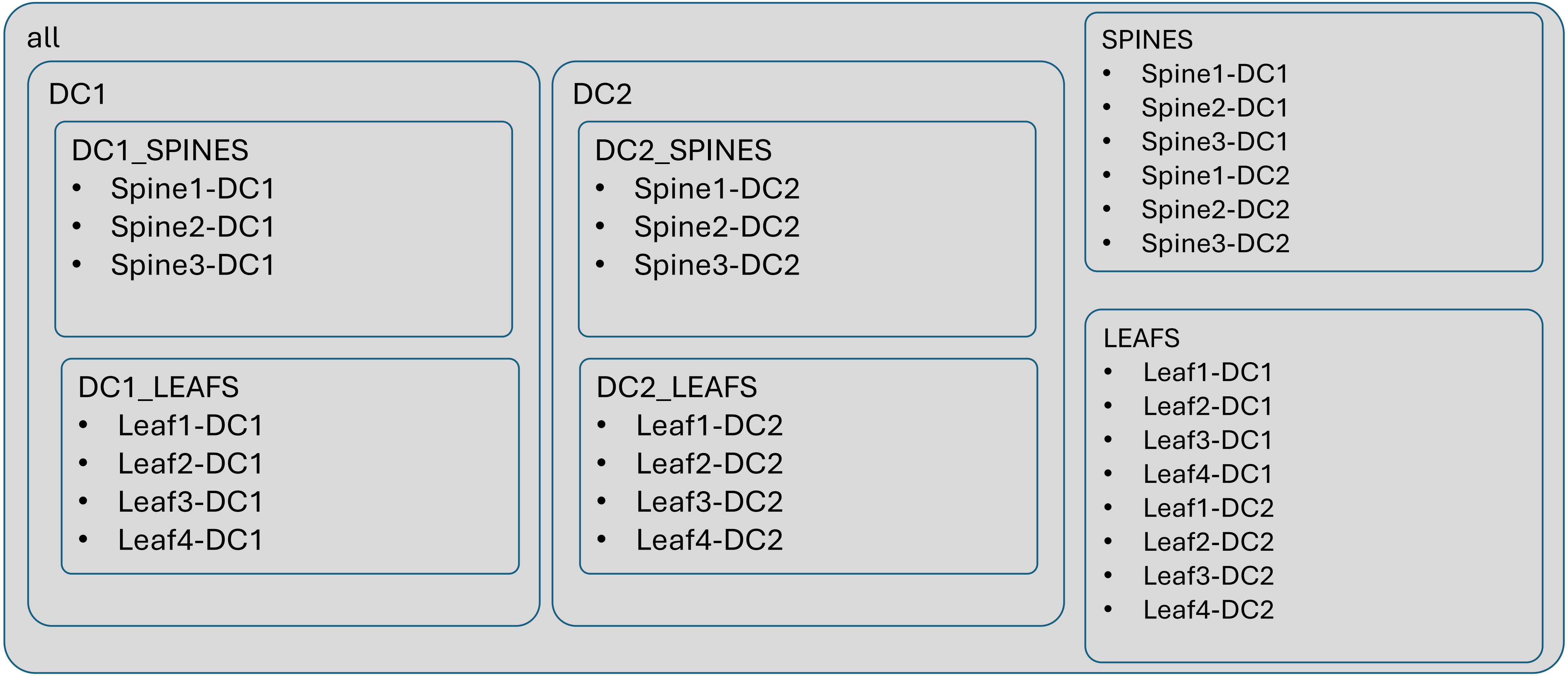
Ansible and Modules (Module Collections)

	Jinja	Arista.eos	Arista.cvp	Arista.avd
Maintainer	Red Hat	Red Hat	Arista	Arista
Found	Built-In	Built-in	Galaxy	Galaxy
Purpose	Create files from templates and data models	Configure EOS devices directly	Perform functions on CVP	Build, document, deploy, and test fabric configurations (L2LS, L3LS+EVPN, MPLS)
		Supplementing manual configuration		

Inventory and Groups with Ansible

CVP_Cluster

- Cvp1



all

DC1

DC1_SPINES

- Spine1-DC1
- Spine2-DC1
- Spine3-DC1

DC1_LEAFS

- Leaf1-DC1
- Leaf2-DC1
- Leaf3-DC1
- Leaf4-DC1

DC2

DC2_SPINES

- Spine1-DC2
- Spine2-DC2
- Spine3-DC2

DC2_LEAFS

- Leaf1-DC2
- Leaf2-DC2
- Leaf3-DC2
- Leaf4-DC2

SPINES

- Spine1-DC1
- Spine2-DC1
- Spine3-DC1
- Spine1-DC2
- Spine2-DC2
- Spine3-DC2

LEAFS

- Leaf1-DC1
- Leaf2-DC1
- Leaf3-DC1
- Leaf4-DC1
- Leaf1-DC2
- Leaf2-DC2
- Leaf3-DC2
- Leaf4-DC2

Arista CVP Collection

- A collection of Ansible modules to interact with CVP
- Upload configlets
- Create containers
- Assign devices to containers
- Assign configlets to devices and/or containers
- Run change controls

Don’t Mix and Match

- Either CloudVision (and arista.cvp/avd) for configuration
- Or
- Use CLI (and arista.eos) for configuration
- But do not do both

Ansible Map

Inventory.yml

Ansible.cfg

Playbooks

- Upload_configlets.yml
- Apply_configlets.yml

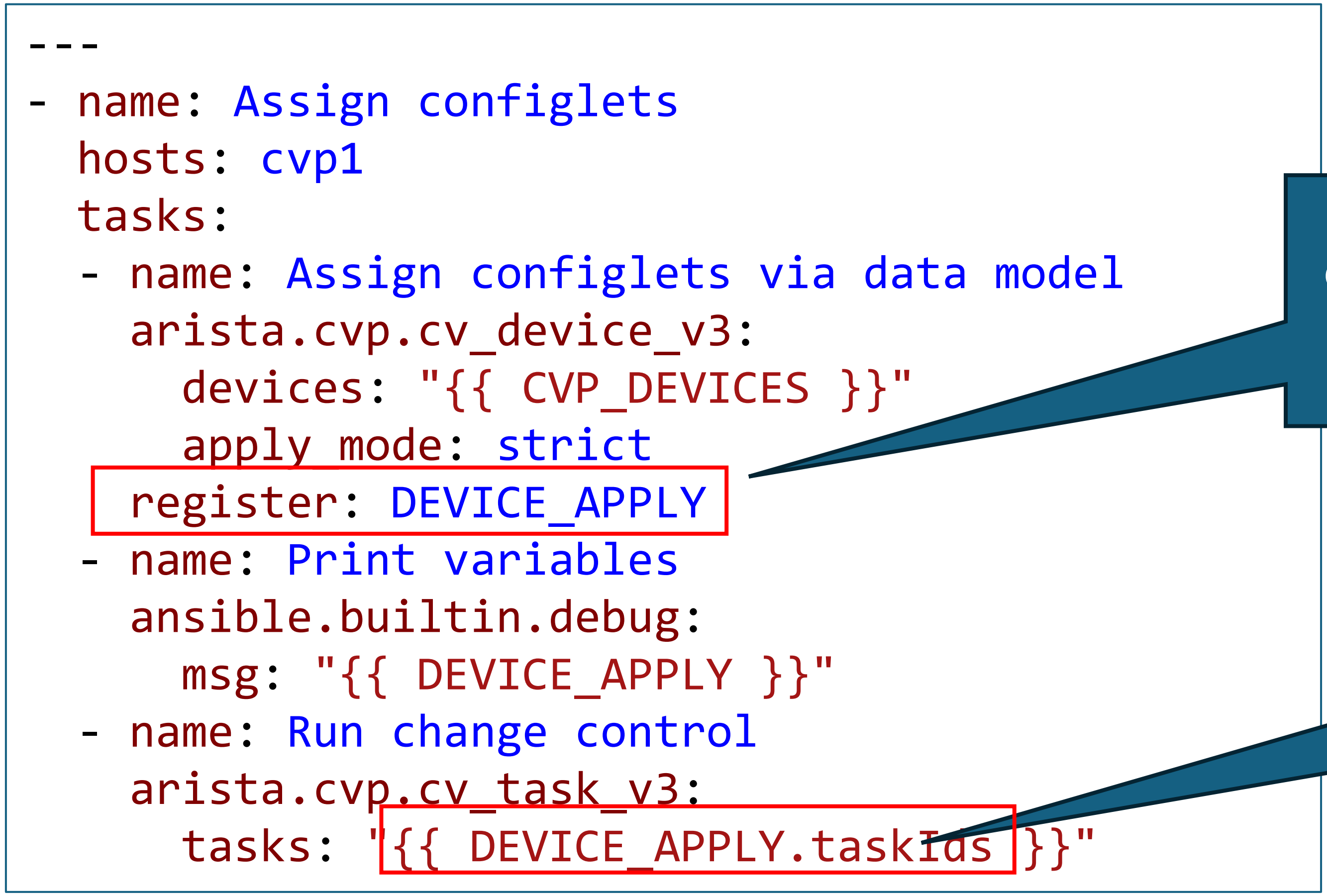
group_vars/
(Groups in inventory)

- all.yml
- CVP_cluster.yml

host_vars/
(hosts are in inventory)

- all.yml
- cpv1.yml

Using Output from One Task as Input on Another

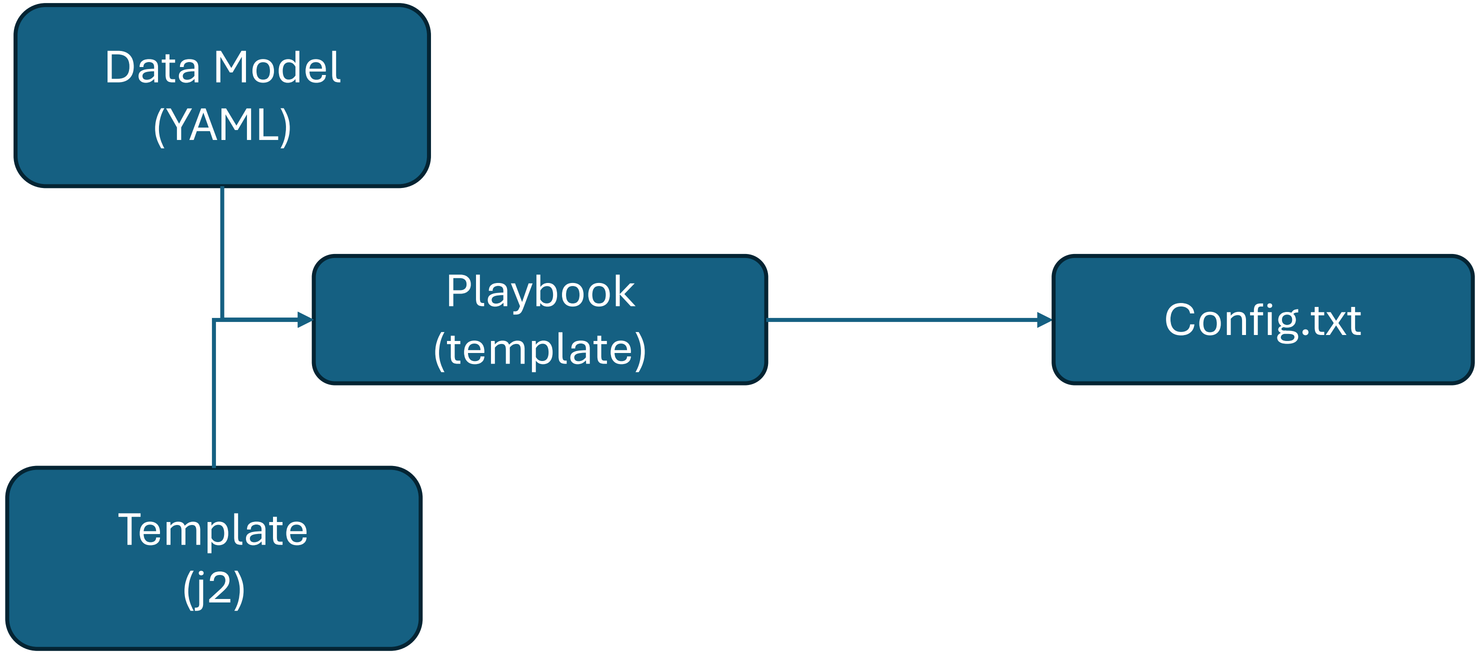


```
---
- name: Assign configlets
  hosts: cvp1
  tasks:
  - name: Assign configlets via data model
    arista.cvp.cv_device_v3:
      devices: "{{ CVP_DEVICES }}"
      apply_mode: strict
    register: DEVICE_APPLY
  - name: Print variables
    ansible.builtin.debug:
      msg: "{{ DEVICE_APPLY }}"
  - name: Run change control
    arista.cvp.cv_task_v3:
      tasks: "{{ DEVICE_APPLY.taskIds }}"
```

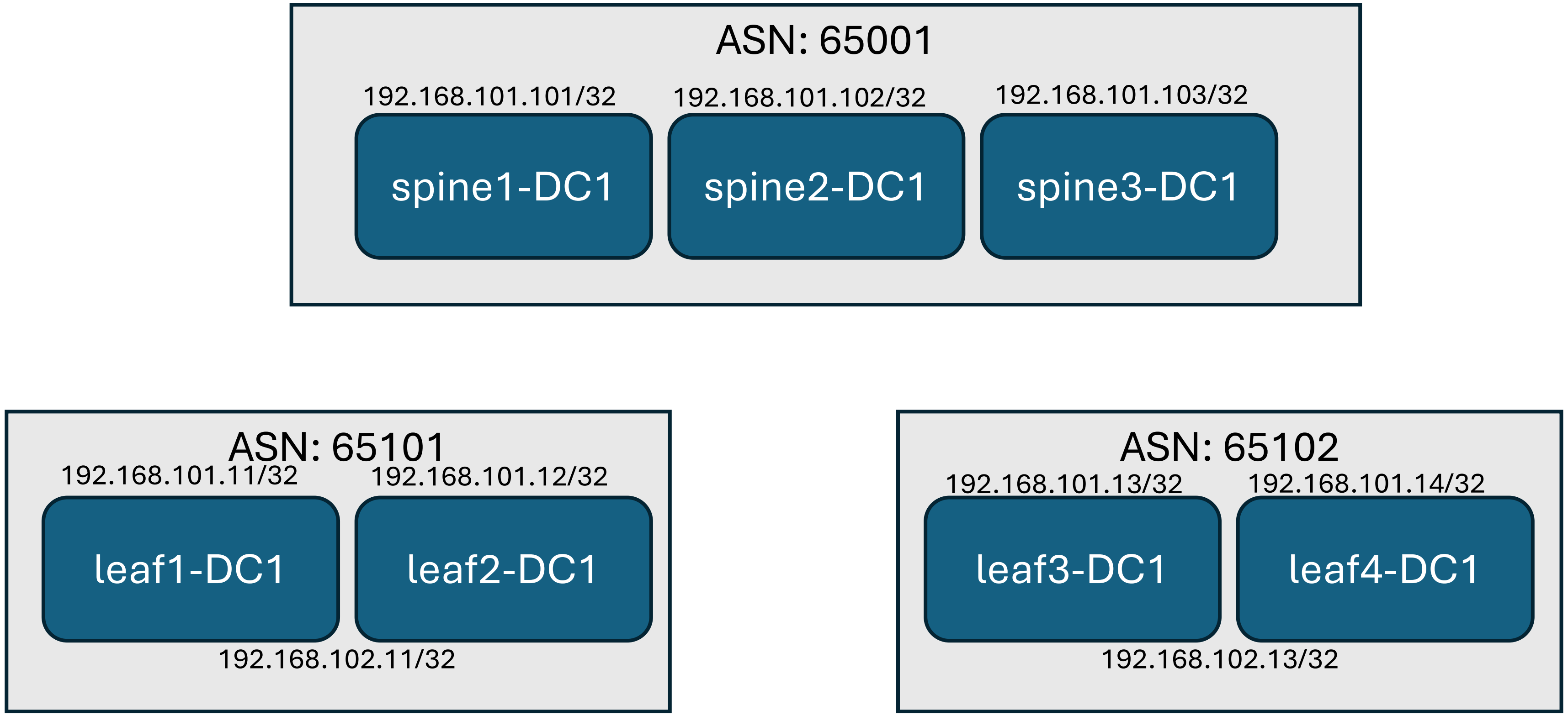
Takes output of cv_device_v3 module an puts it into a new dictionary

Runs a change control with the taskIds that were created from the previous task

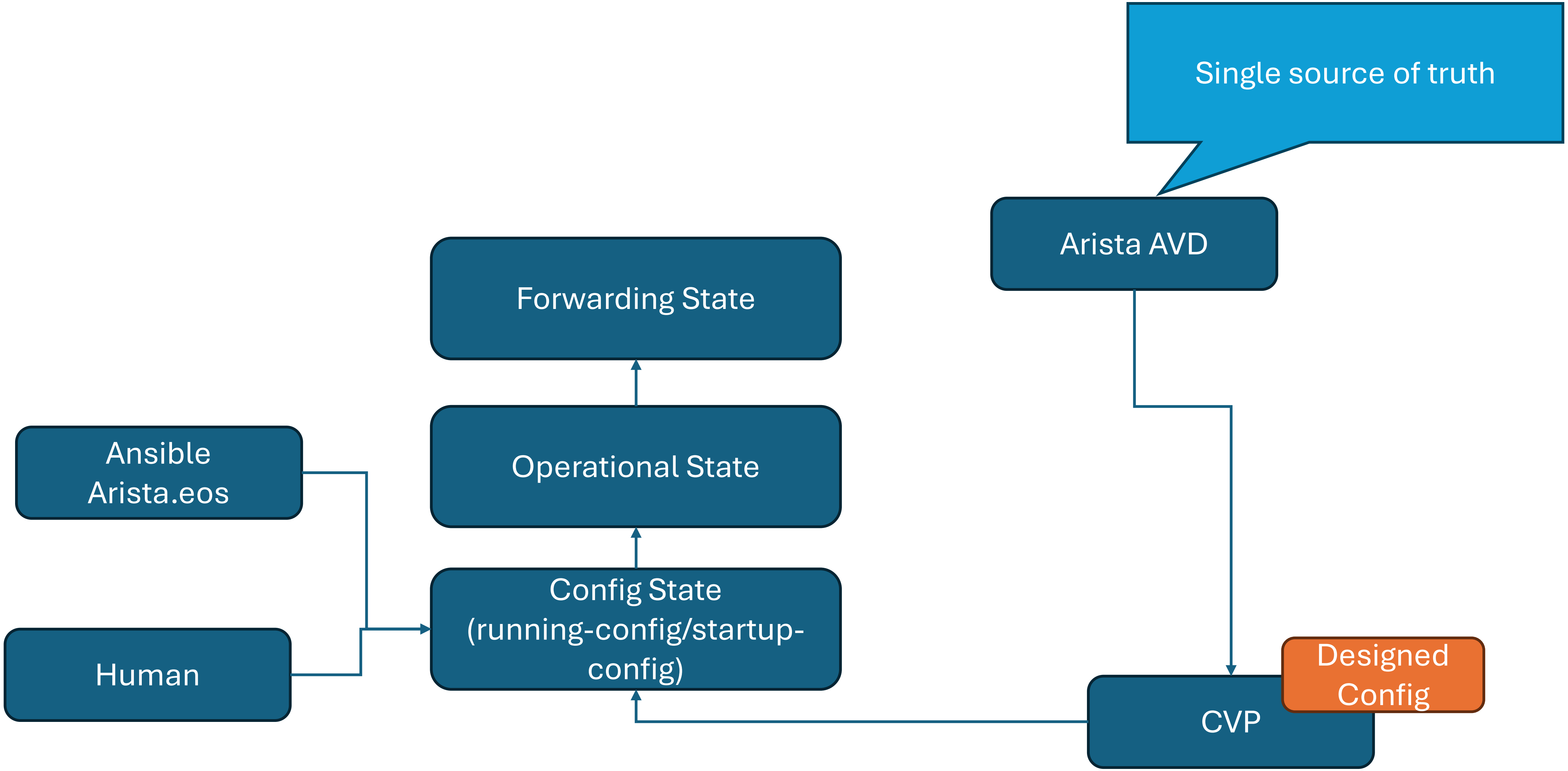
Jinja Template Maps



eBGP VXLAN


ASN: 65001
192.168.101.101/32
192.168.101.102/32
192.168.101.103/32
spine1-DC1
spine2-DC1
spine3-DC1
ASN: 65101
192.168.101.11/32
192.168.101.12/32
leaf1-DC1
leaf2-DC1
192.168.102.11/32
ASN: 65102
192.168.101.13/32
192.168.101.14/32
leaf3-DC1
leaf4-DC1
192.168.102.13/32

Three States of a Network Device



The Grand Table of Arista Automation

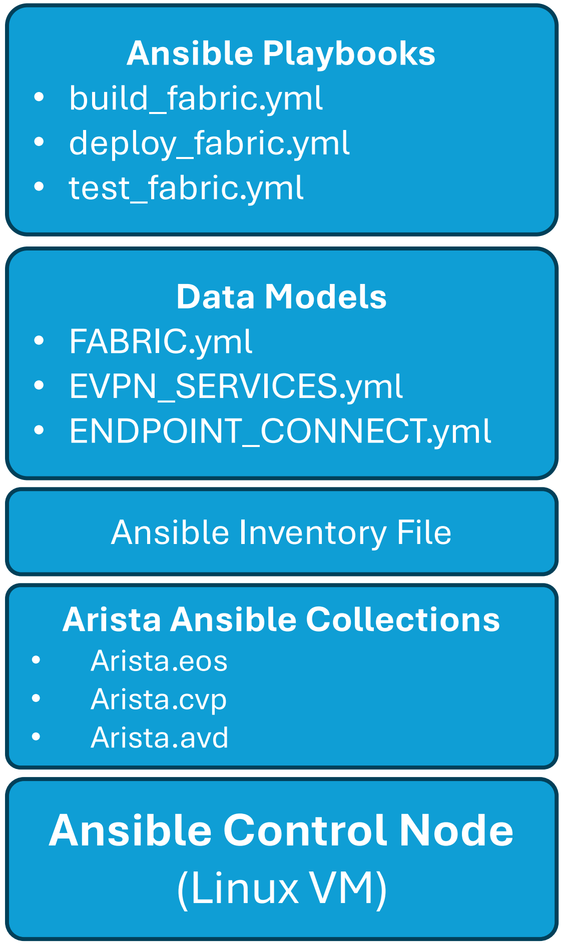
Method	Works Well With	Not Great For	Learning Curve
Manual	L2LS	EVPN/VXLAN	N/A
Ansible arista.eos	Manual config supplementing	EVPN/VXLAN Total automation	Low
Studios	Small-medium shops EVPN/VXLAN	Integration with external tools Customization	Extremely Low
Configlets Builders	Legacy customers	Modern automation	High
Ansible arista.cvp	Medium-large CVP installations Jinja templating		Medium
Jinja+Ansible	Highly customized situations Multi-vendor	When more simplicity is needed	High
AVD	Complex EOS configs (EVPN/VXLAN, MPLS)	Simple environments, multivendor	Medium

What Does AVD Do?

- **Builds** configurations (fabric-wide)
- **Document** the configurations
- **Deploy** the configurations
  - Directly to EOS
  - Via CVP
- **Test** those configurations

How Does AVD Work?

- AVD runs on Ansible (now a pyAVD Python module to run outside of Ansible)
- You build the Ansible inventory file, data models, and create simple playbooks



Ansible Playbooks

- build_fabric.yml
- deploy_fabric.yml
- test_fabric.yml

Data Models

- FABRIC.yml
- EVPN_SERVICES.yml
- ENDPOINT_CONNECT.yml

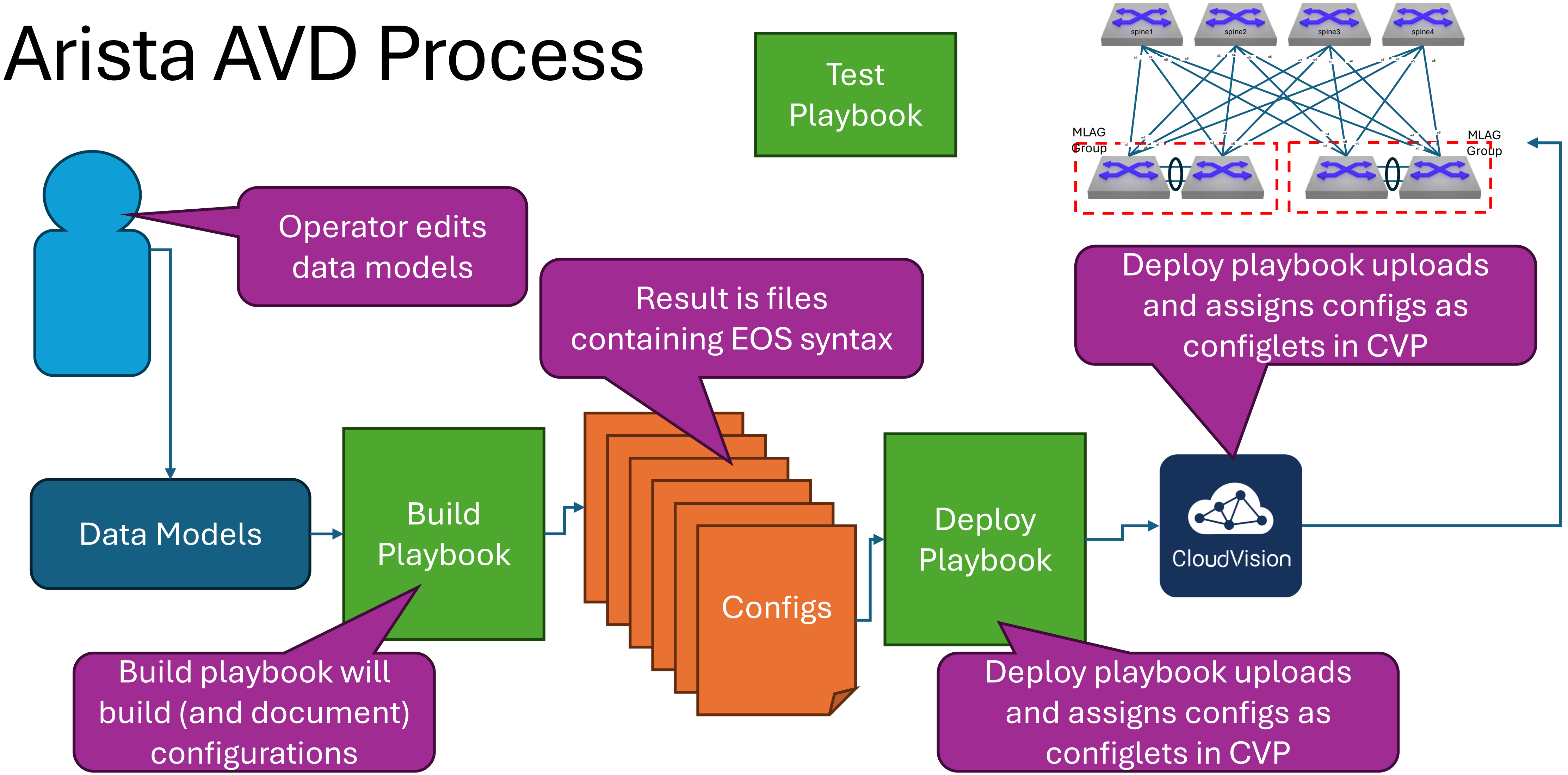
Ansible Inventory File

Arista Ansible Collections

- Arista.eos
- Arista.cvp
- Arista.avd

Ansible Control Node
(Linux VM)

Arista AVD Process



Test Playbook

Operator edits data models

Result is files containing EOS syntax

Deploy playbook uploads and assigns configs as configlets in CVP

Data Models

Build Playbook

Configs

Deploy Playbook

CloudVision

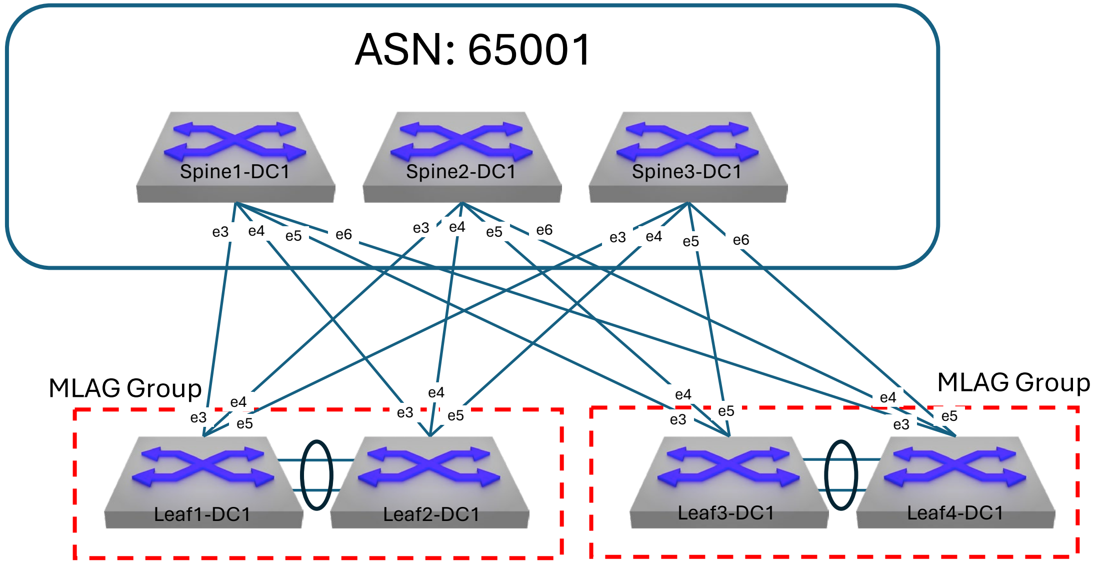
Build playbook will build (and document) configurations

Deploy playbook uploads and assigns configs as configlets in CVP

Operational Model

- Day 0
  - Getting requirements, ordering hardware, racking/stacking
  - Build our data models
- Day 1
  - Deploy configs created from data models
  - Test the environment (acceptance testing)
- Day 2+
  - Change data models, build configs, deploy configs, test
  - 20 GOTO 10

DC1 Buildout

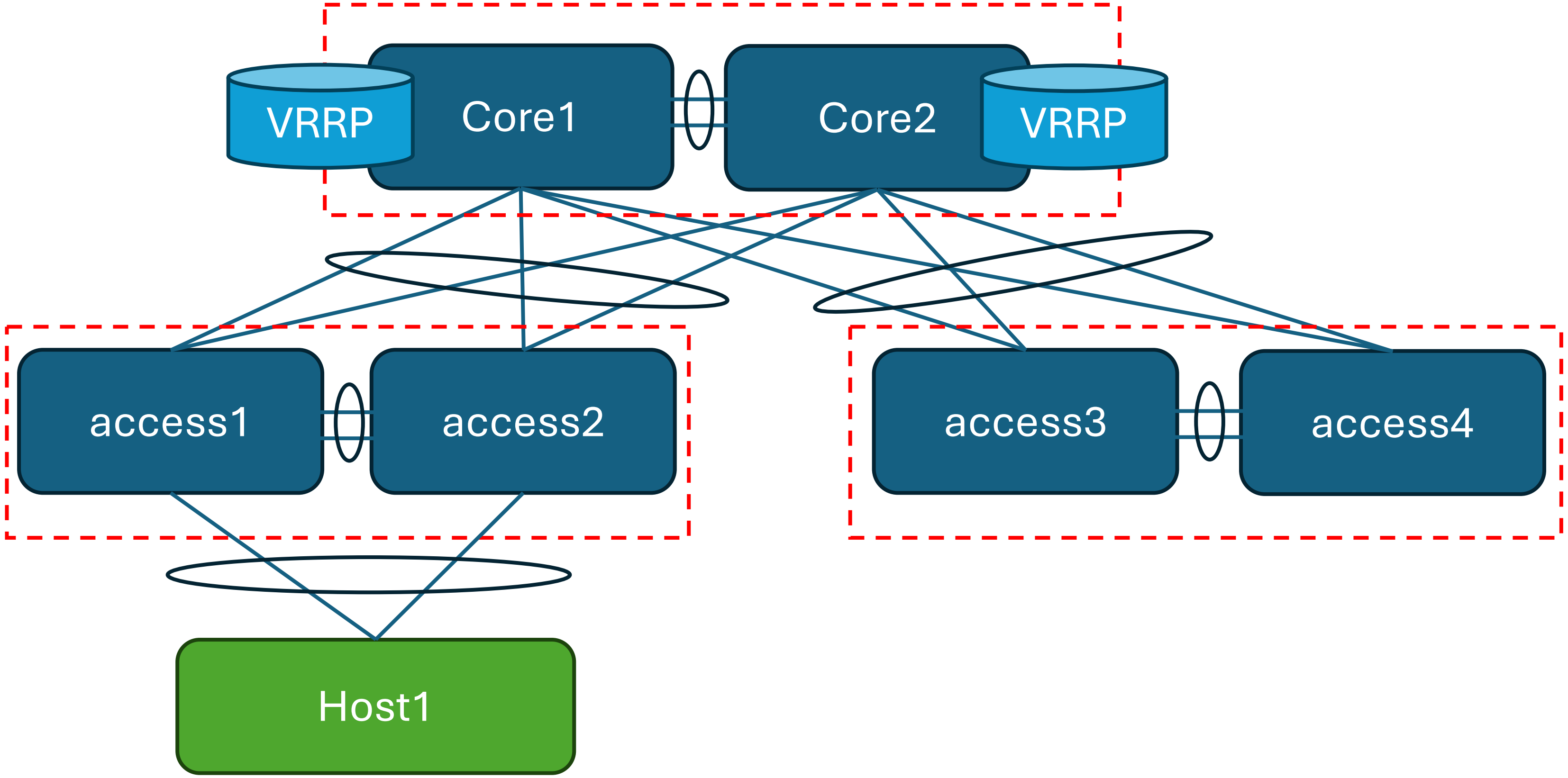


Auto: 65100-65199

AVD Process

- We need an Ansible Control Node
  - Linux VM
  - Give it lots of cores (8 or more)
  - Integrate with Git
  - Install Ansible, Python modules, AVD/EOS/CVP collections
- Build ansible.cfg
- Build inventory file

L2LS


VRRP
Core1
Core2
VRRP
access1
access2
access3
access4
Host1