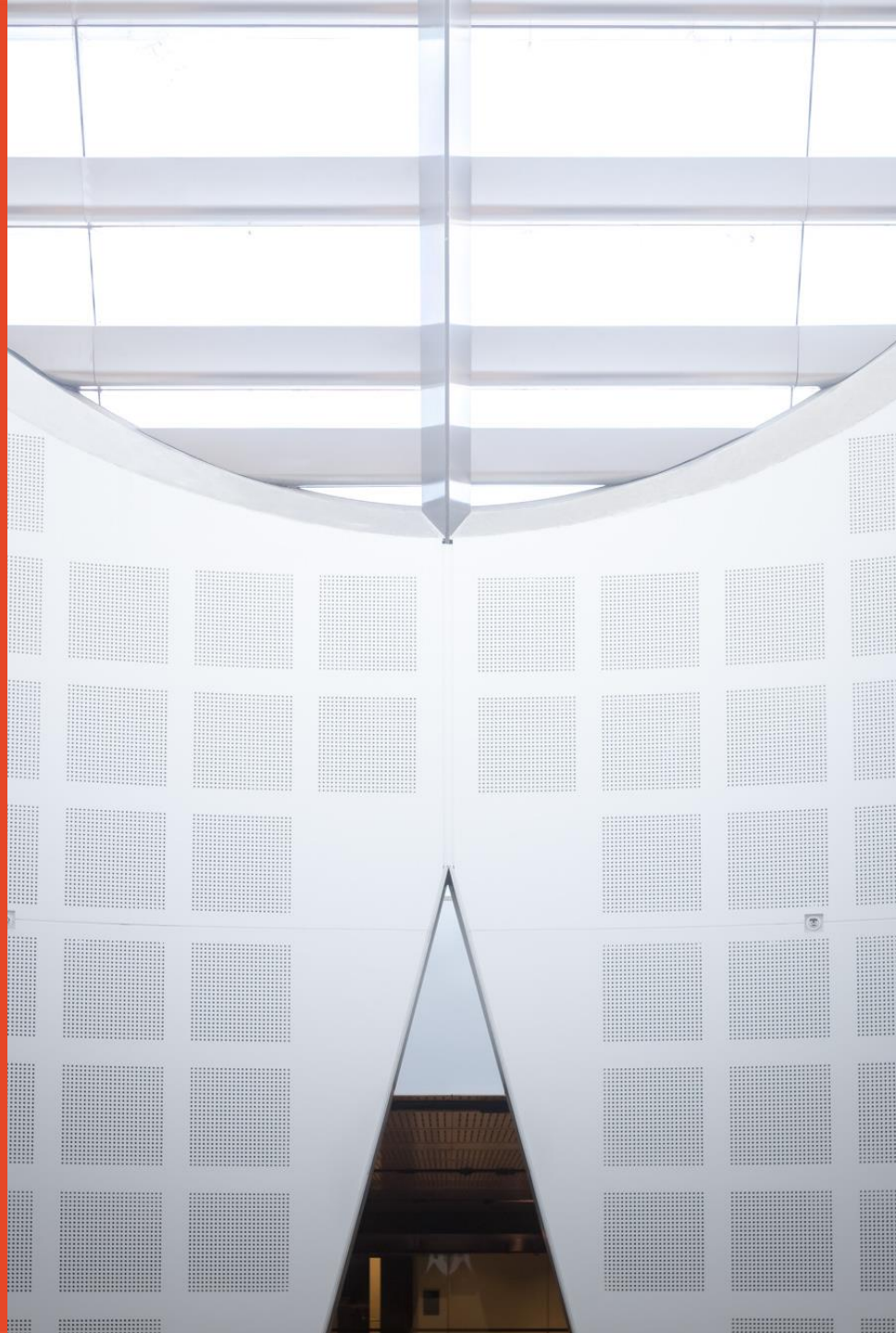
Topic 5:
SDN Controller

Presented by
Dong YUAN

School of Electrical and Computer Engineering

dong.yuan@sydney.edu.au

THE UNIVERSITY OF SYDNEY

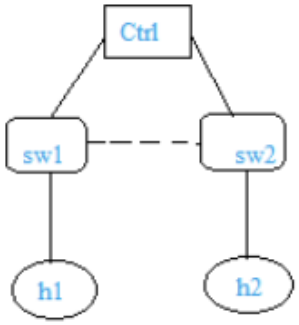


Contents

- Control plane

- Representative controllers

A few questions



- What should happen when h1 wants to visit h2 but no corresponding rules in sw1?

- How does a new switch connect to the Controller in an SDN?
  - Physically and Logically

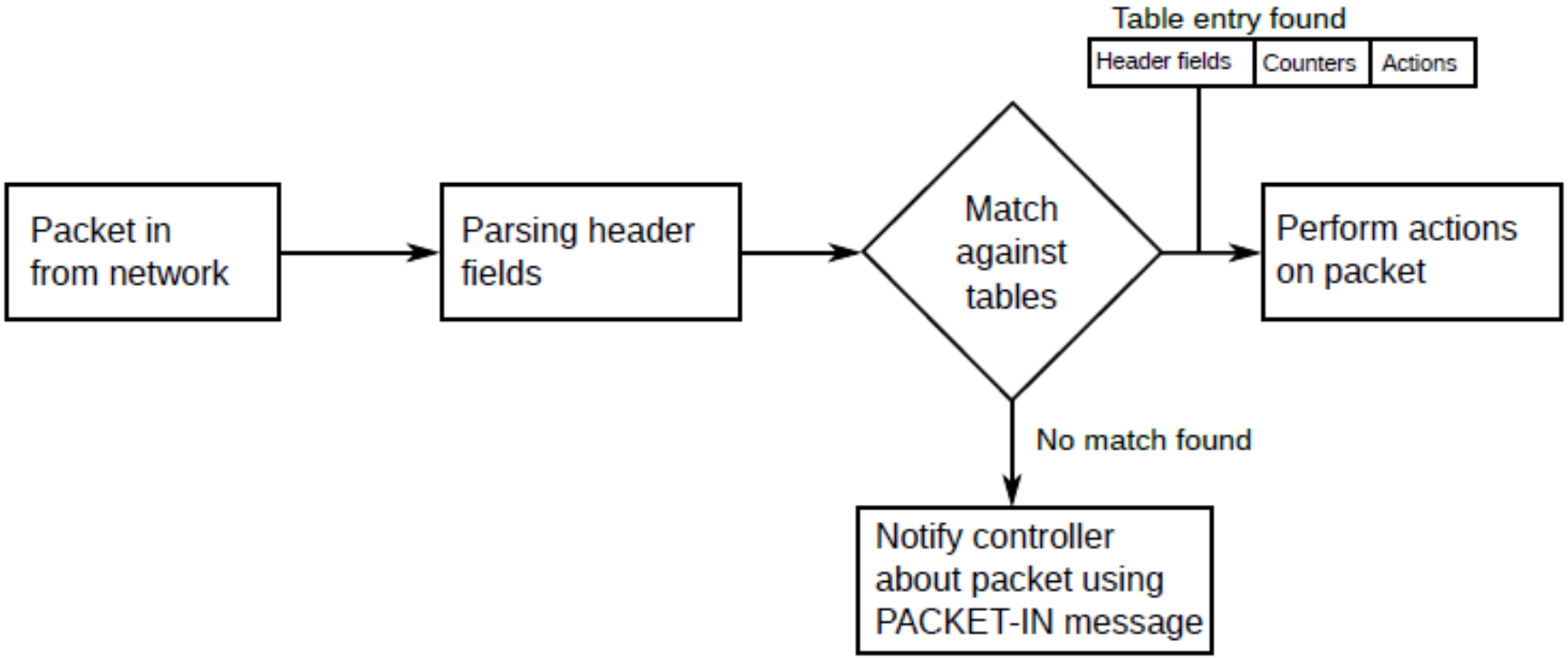
- How do data communicate among controllers?

Controller

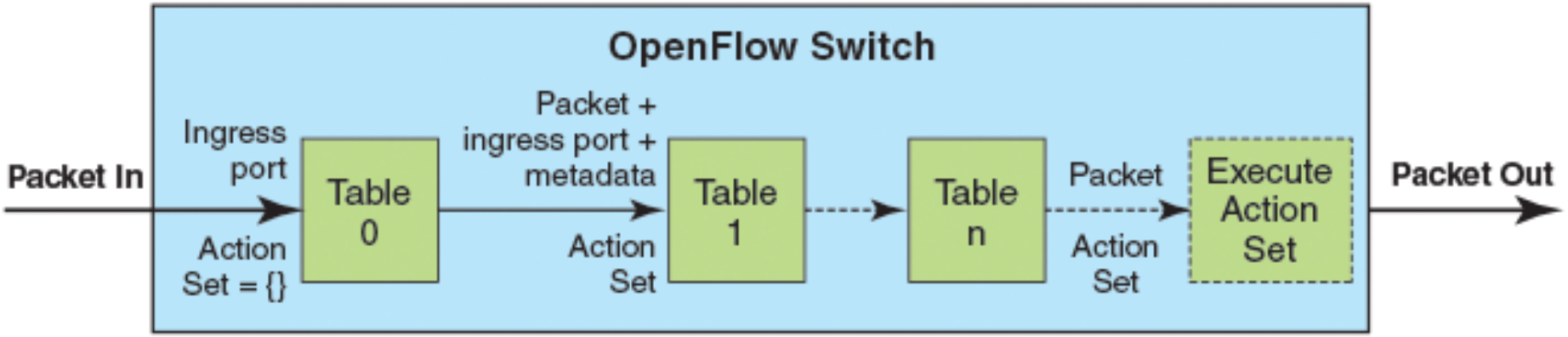
- In openflow, controller communicates with switch over a secure channel
  - Openflow protocol defines message format
  - Purpose of control channel: update flow table
  - Logic is executed at controller
  - Communication with external controllers

- Flow Table: Packet switching
  - All packets compare to flow table for match
    - Packet header fields matched against one of the N tables
  - Actions depend on match being found
    - Forward, Drop, modify, enqueue
  - If no match, traffic is sent to controller

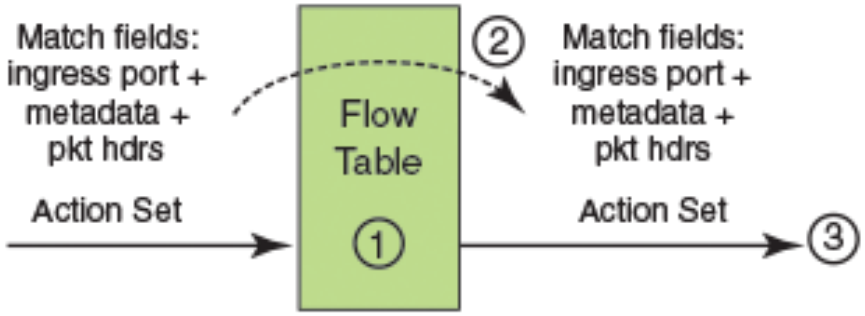
Basic OpenFlow Matching



Advanced Matching



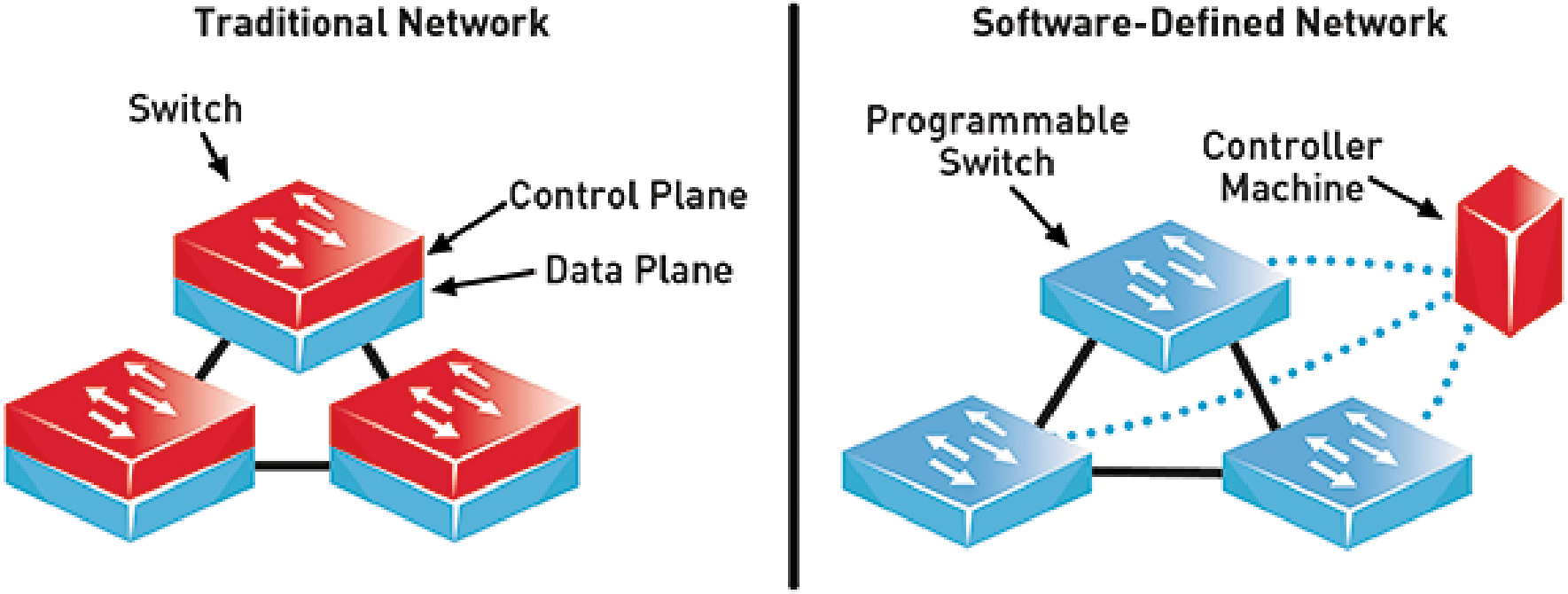
{a} Packets are matched against multiple tables in the pipeline



1. **Find highest - priority matching flow entry**
2. **Apply instructions:**
   i. Modify packet & update match fields (apply actions instruction)
   ii. Update action set (clear actions and/or write actions instructions)
   iii. Update metadata
3. **Send match data and action set to next table**

{b} Per-table packet processing

The control plane



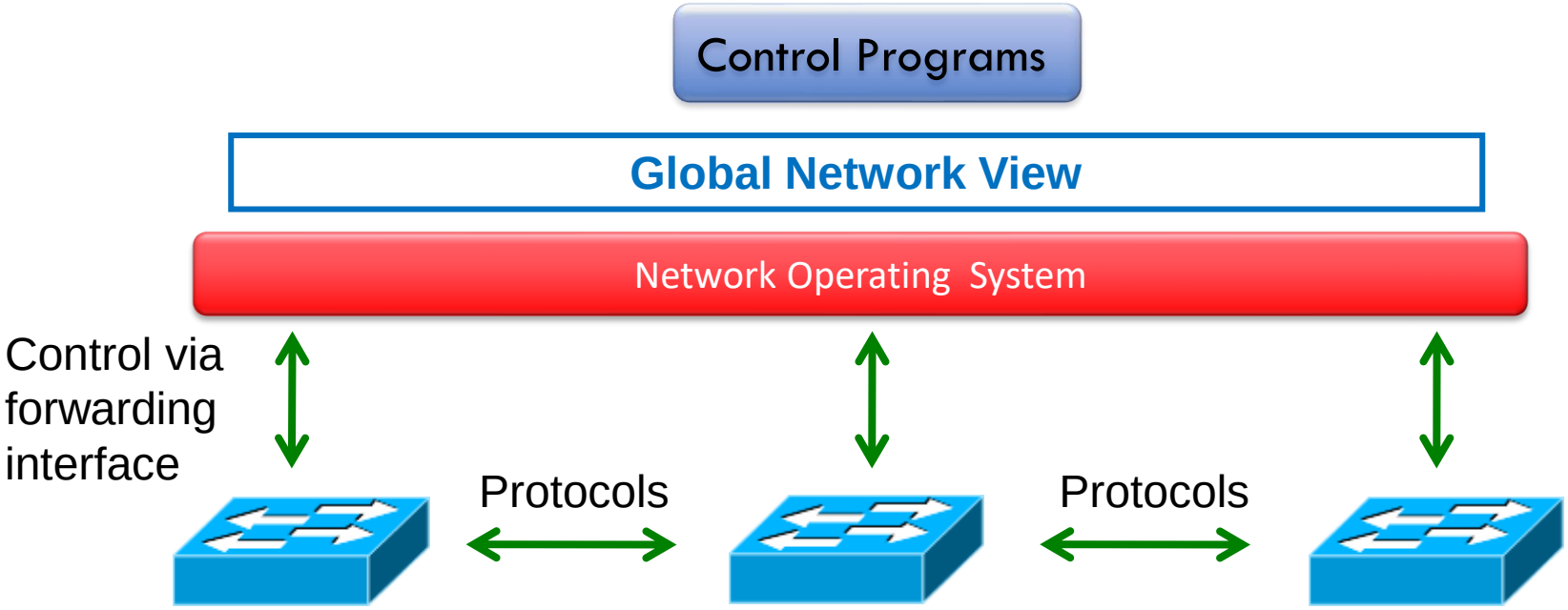
– *Separation of data plane and control plane*

Deriving the Control Plane

- What does it take to control a network?
  - Learn the network state (e.g., topologies, etc.)
  - Decide how to configure it (routing, iosolation, traffic management, etc.)
  - Push configuration to the network element
- What processes are re-usable?
  - Build the network topology
  - Push configuration to network elements

Idea: An OS for Networks

Software-Defined Networking (SDN)



Network OS

- Maintain an up-to-date view of the network state (e.g., topology, etc.)
  - Next page
- Configure network elements
  - A.k.a “south-bound” interface or API
- Provide a graph abstraction to the applications on the top
  - A.k.a “north-bound” interface or API

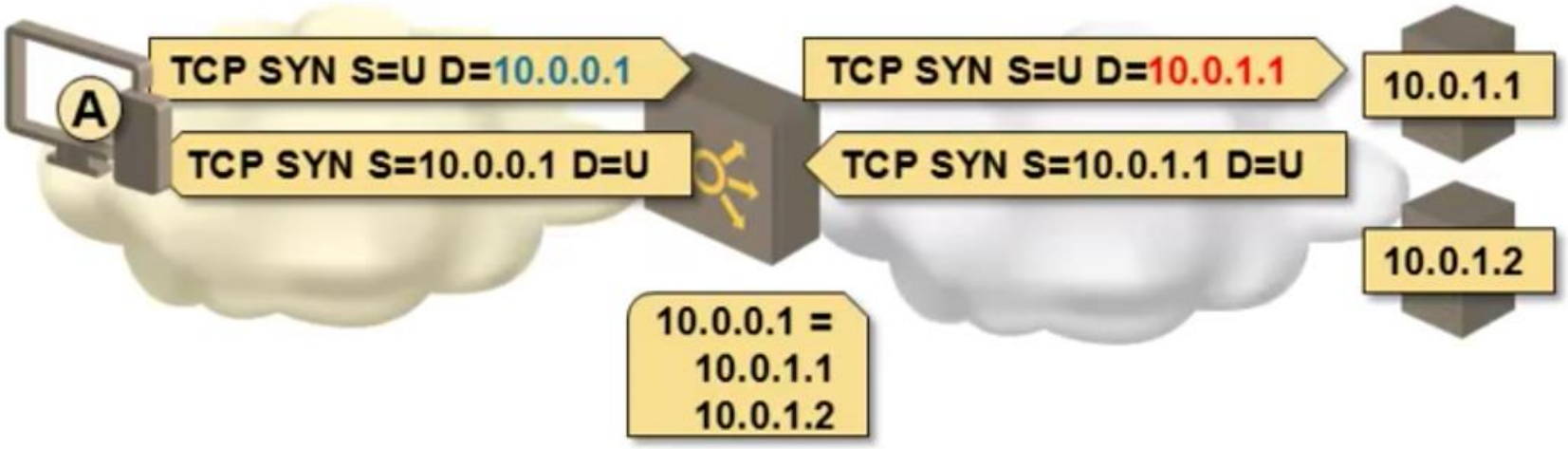
Network View (graph abstraction)

- In the network view:
  - Switch-level topology
  - Location of host, middlebox and other network elements
  - Location of users
  - Namespace: bindings between names and addresses
- Not in the network view:
  - Some states of network, e.g., traffic.

- Network Information Base (NIB)
  - Graph and abstraction

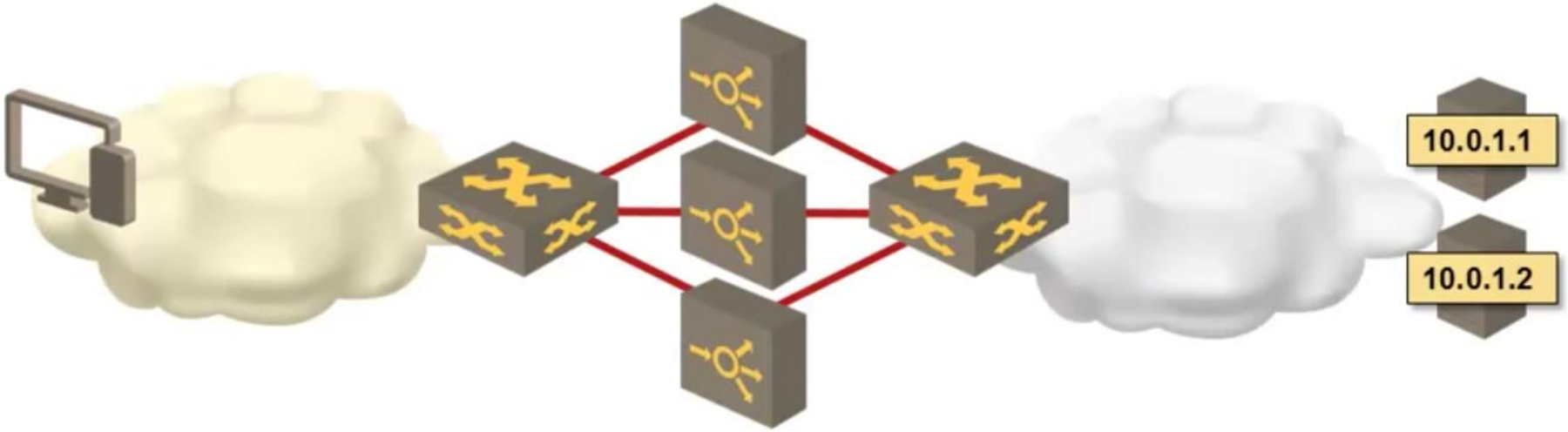
Example: Scale-out load balancers

- Map incoming sessions to a single virtual IP address into sessions distributed across a server farm
- Challenge
  - NAT translation record for every active session
  - Large amount of states



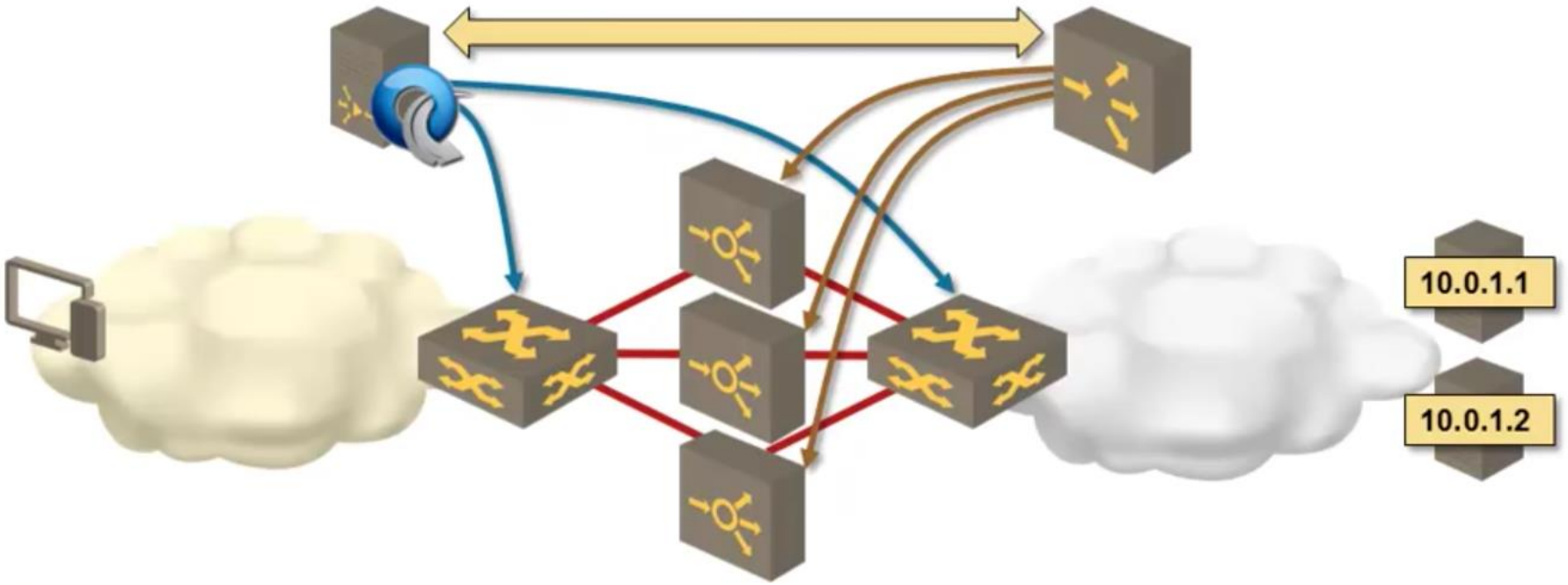
Example: Scale-out load balancers

- Stateful services are hard to scale out
  - Forward and reverse traffic flows might not match
  - Traffic distribution might change with adding or removing of devices



OpenFlow-assisted scale-out load balancer

- OpenFlow switches use coarse-grained flows, based on source/dest IP address ranges
- Load Balancing appliances perform fine-grained load balancing
- Load Balancing controller modifies OpenFlow flows and traffic distribution based on node availability



Challenges in building an NOS

- Scalability
- Reliability

- Good performance (fast, etc.)
- Generality and Simplicity of the “north-bound” API.
  - Easily to be used by applications

Controller's Diversification

- Different SDN controllers
  - NOX/POX
  - ONIX
  - Ryu
  - Beacon
  - Floodlight

  - Pyretic, Frenetic, Procera, etc.

- The Most Popular Two Controllers: ONOS and OpenDayLight

How to choose a controller

- Programming language
  - Can affect performance
- Focus
  - Southbound API
  - Northbound API
  - Support for Openstack/Kubernetes
  - Research or production
- Learning curve
- User base and community support

NOX

- First-generation OpenFlow controller
  - Open source, stable, widely used
  - Fast, well mantained
  - C++, Openflow 1.0
    - Some fork versions support higher OpenFlow version
- Programming model
  - Controller registers for events
  - Programmer writes event handlers
  - This is a very common SDN programming model

When to use NOX

- You know C++
- You are willing to use low-level facilities and semantics of OpenFlow
- You need good performance

POX

- NOX in Python
  - Also only support OpenFlow 1.0
- Advantages:
  - Widely used, well maintained and supported.
  - Relatively easy to read and write code
- Disadvantages:
  - Performance
- When to use POX
  - You know python and do not care about performance
  - Rapid prototyping and experimentation
    - For research, demonstration and learning concept

ONIX

- A distributed controller
- Closed source
- Production quality (used by google)

- Distributed Design
  - Scalability
  - Robustness

Ryu (pronounced "ree-yooh")

- Open source Python controller
  - Support OpenFlow 1.0-1.4
  - Works with openstack/kubernetes
- Aims to be an “Operating System” for SDN
  - Provide well-defined North-Bound APIs.

- Advantages
  - Openstack integration
  - Support newest OpenFlow
  - Easy to use, and often used in research for prototyping
- Disadvantages
  - Performance

Beacon

- From Stanford
- First (?) Java based controller
  - Fast
  - Cross platforms

Floodlight

- Open source Java controller
  - Support OpenFlow 1.0
  - For from Beacon
  - Maintained by Big Switch Networks
- Advantage
  - Good documentation
  - Integration with REST API, OpenStack
  - Production-level performance
- Disadvantagte
  - Deep learning curve

Today's Two Popular controllers

- ONOS and OpenDayLight (ODL)
- ONOS (2014)
  - From Open Networking Foundation
  - Previously ON.LAB funded by Stanford and Berkeley
- ODL (2013)
  - From Linux Foundation

- Both ONOS and ODL are written in Java and designed for modular use with a customizable infrastructure
- Both support OpenStack
- Every ONOS partner is also an ODL member

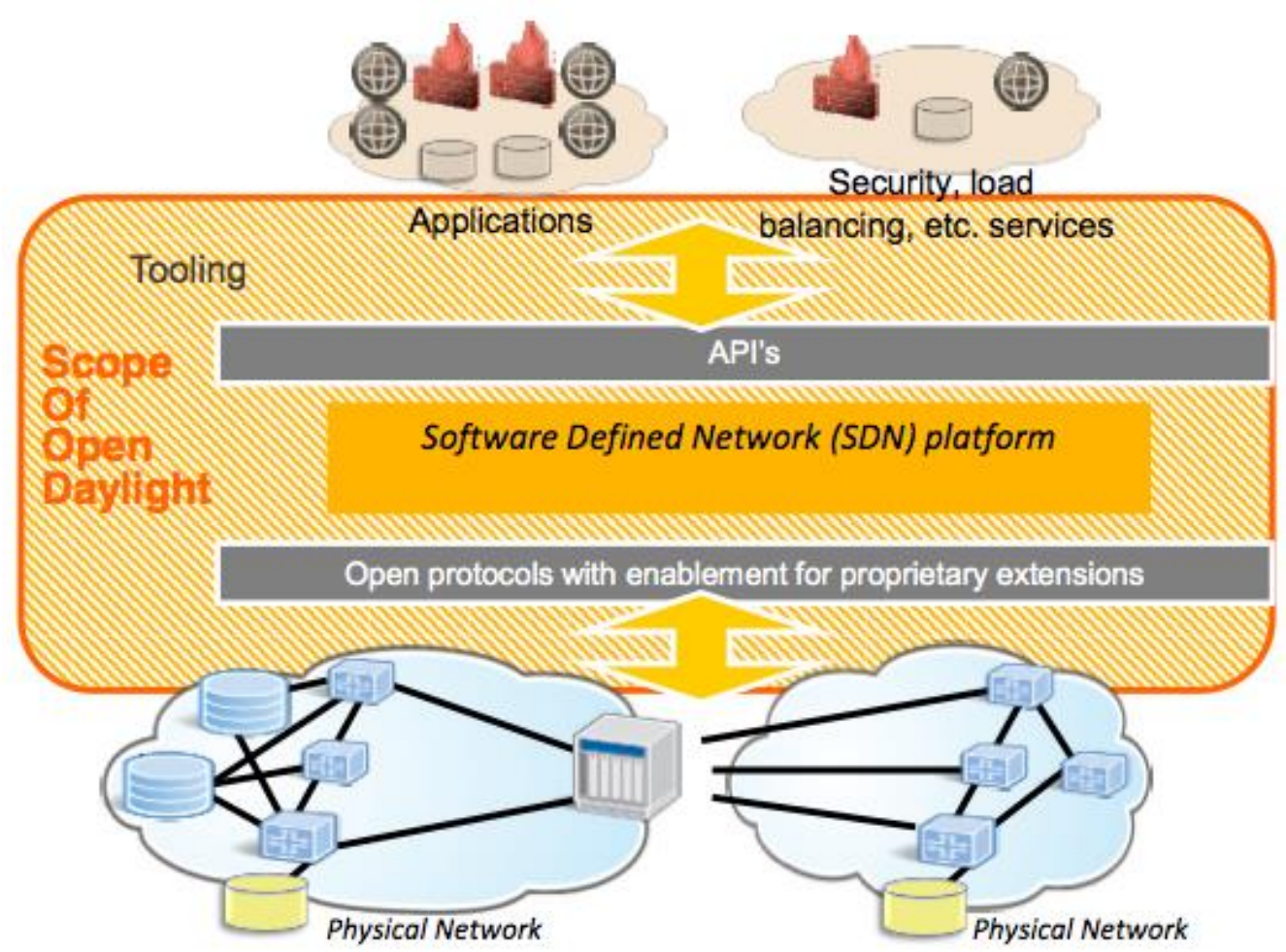
Differences

- ONOS vs. ODL
  - Carrier-grade networks vs. Cloud provider
  - Pure SDN vs. Legacy
  - Academic initiated vs. Corporate initiated

OpenDayLight Controller

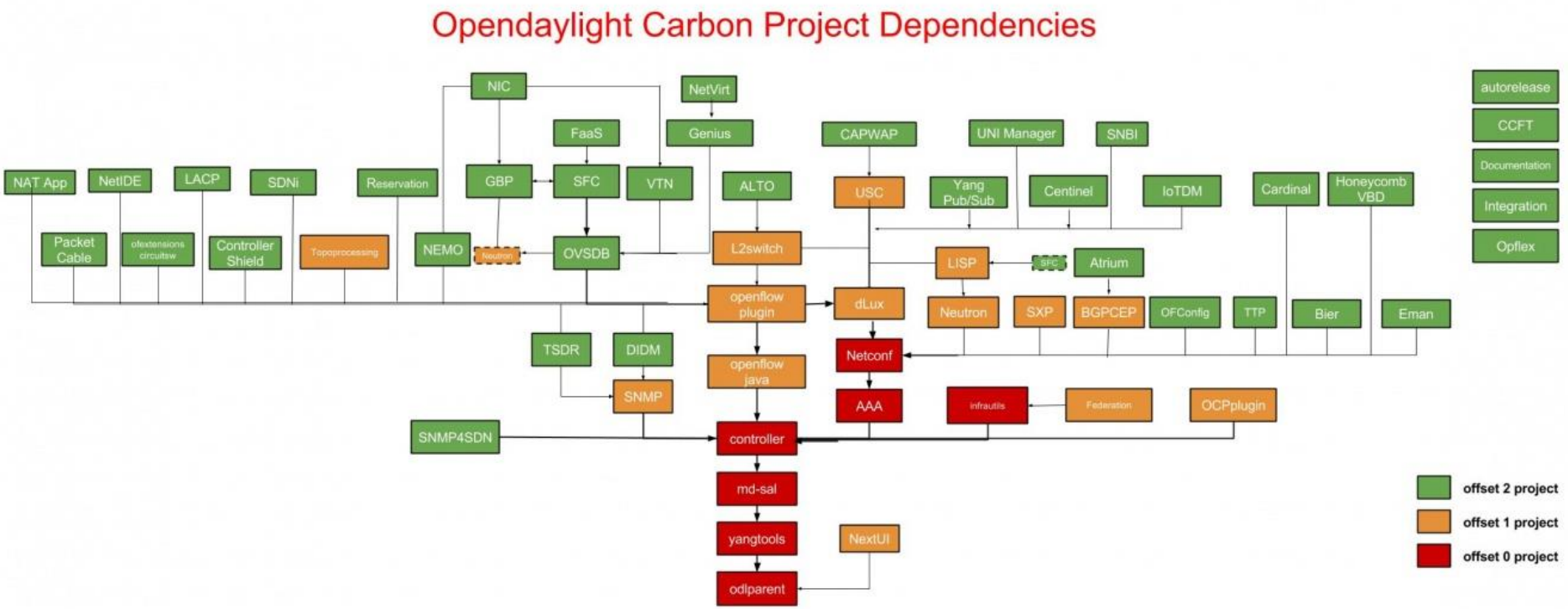
OpenDaylight Scope

- OpenDaylight Scope



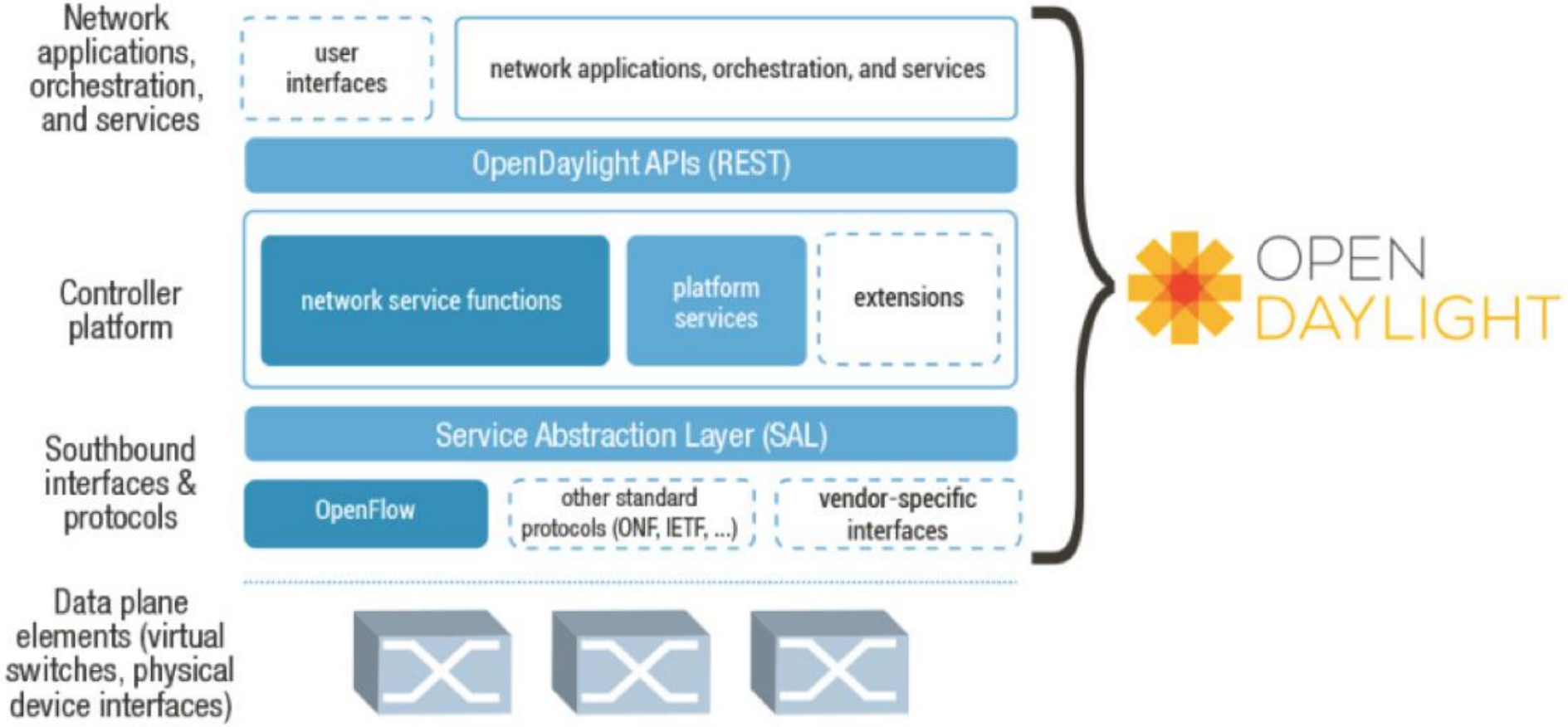
OpenDaylight Projects

- OpenDaylight Projects
  - Classification: kernel projects, protocol projects, app projects, service projects, support projects, etc.
  - More projects indicate the OpenDayLight community is active



OpenDaylight Framework

- Project Framework



Key Technologies in ODL

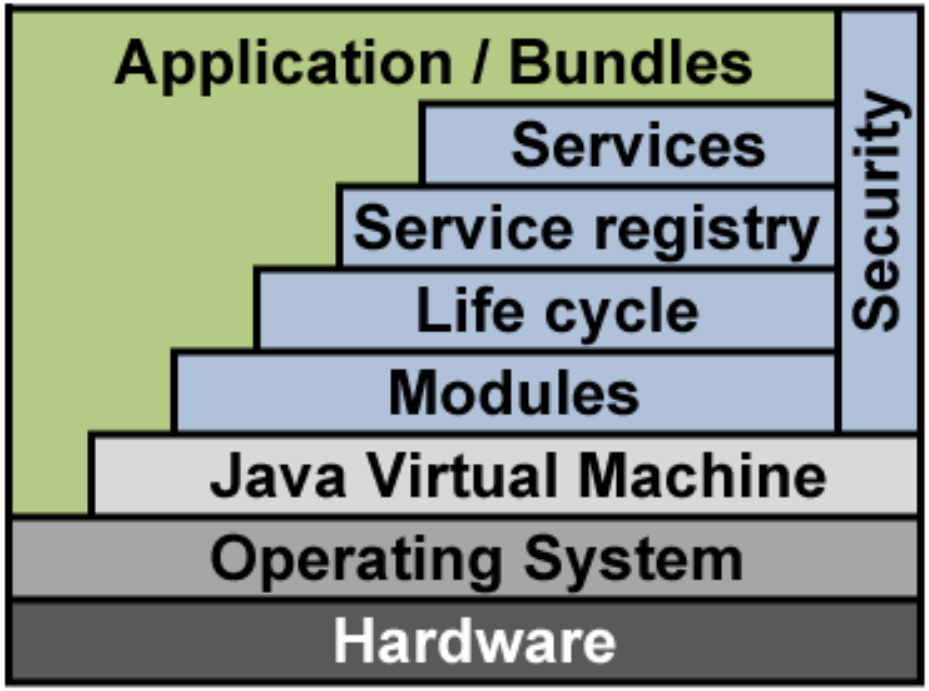
- **OSGI** - This framework is the back-end of OpenDaylight as it allows dynamically loading of bundles and packages JAR files, and binding bundles together for exchanging information.

- **Karaf** - Application container built on top of OSGI, which simplifies operational aspects of packaging and installing applications. Runtime environment for well designed code modules, aka, “modulith runtime”.

- **YANG** - a data modeling language used to model configuration and state data manipulated by the applications, remote procedure calls, and notifications.

Java OSGi Framework

- OSGi (Open Service Gateway Initiative) is a Java framework for developing and deploying modular software programs and libraries.
- Applications or components, coming in the form of bundles for deployment, can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot.

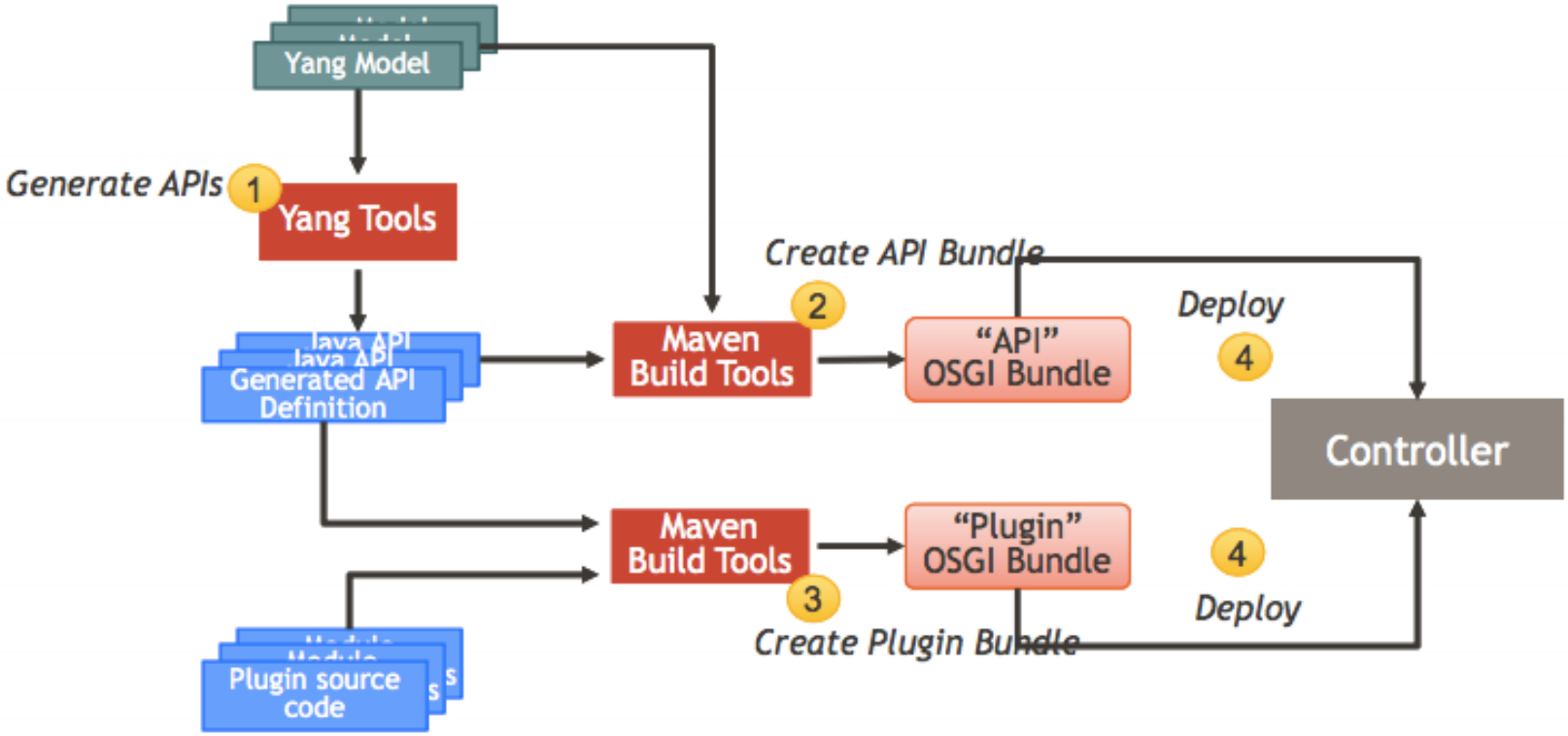


OpenDaylight Controller

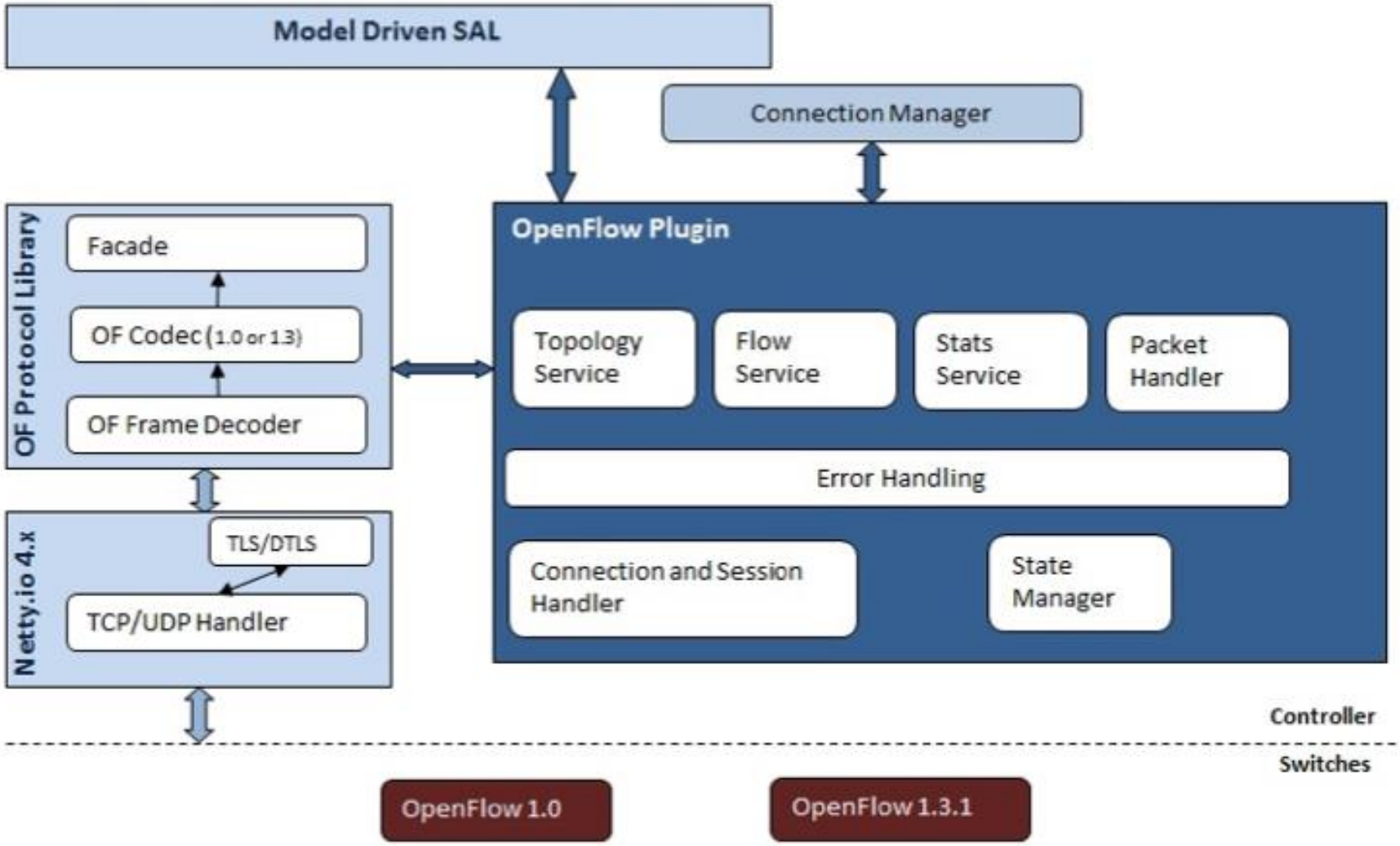
- Features
  - Built using Java OSGi Framework
    - Provides Modularity & Extensibility
    - Bundle Life-cycle management
    - In-Service-Software Upgrade (ISSU) & multi-version support
  - Service Abstraction Layer (SAL)
    - Provides Multi-Protocol Southbound support
    - Abstracts/hides southbound protocol specifics from the applications
  - High availability & horizontal scaling using clustering
  - Version evolve: Hydrogen, Helium, Lithium, Beryllium, Boron, Carbon, Nitrogen, Oxygen, Fluorine, …

OpenDaylight Controller

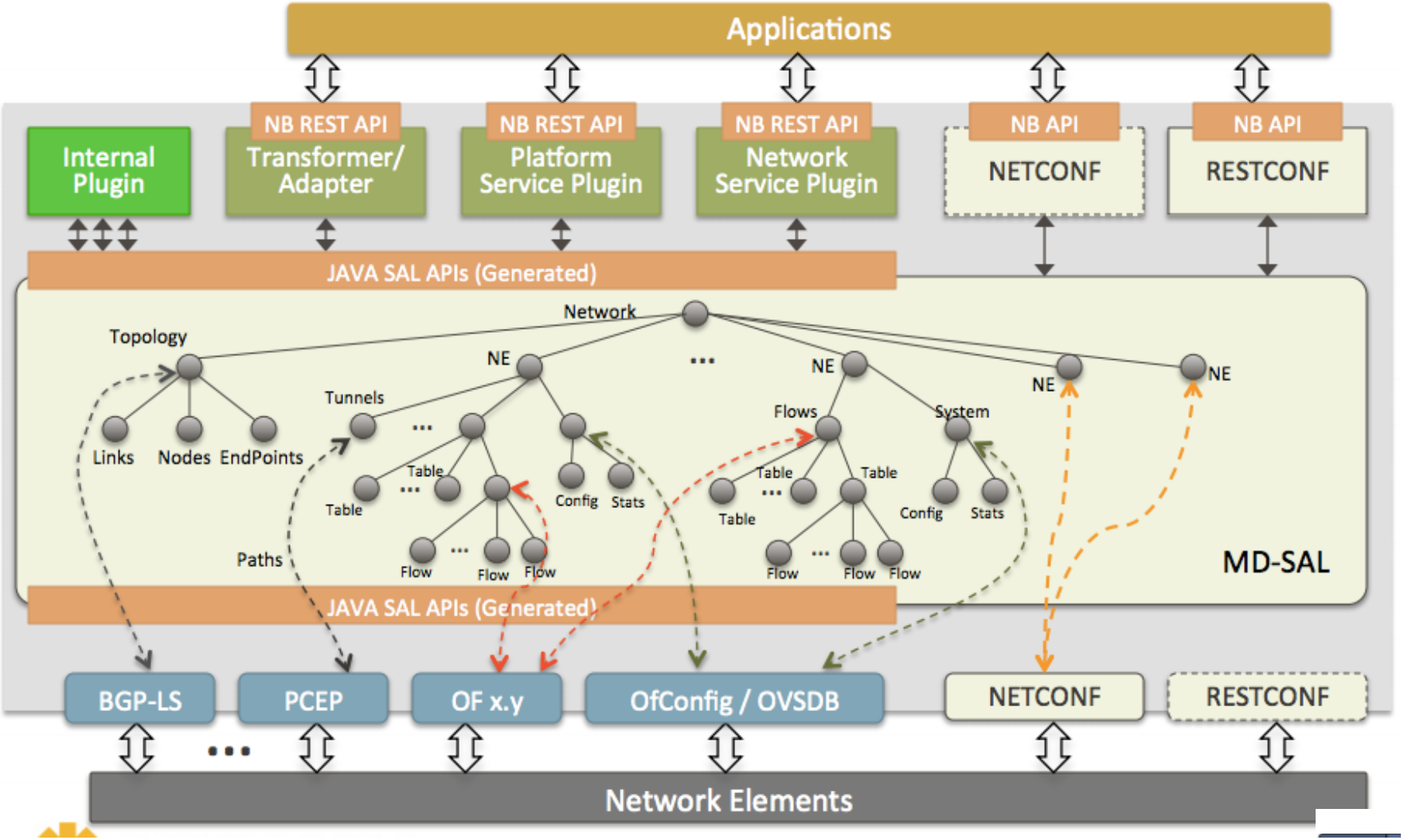
– Plugin Build Process



OpenDaylight Controller

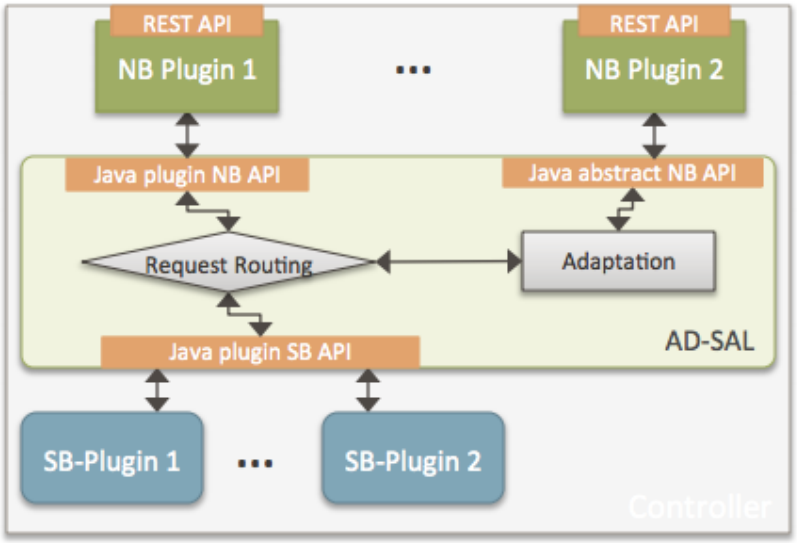


Model-Driven SAL (1/2)

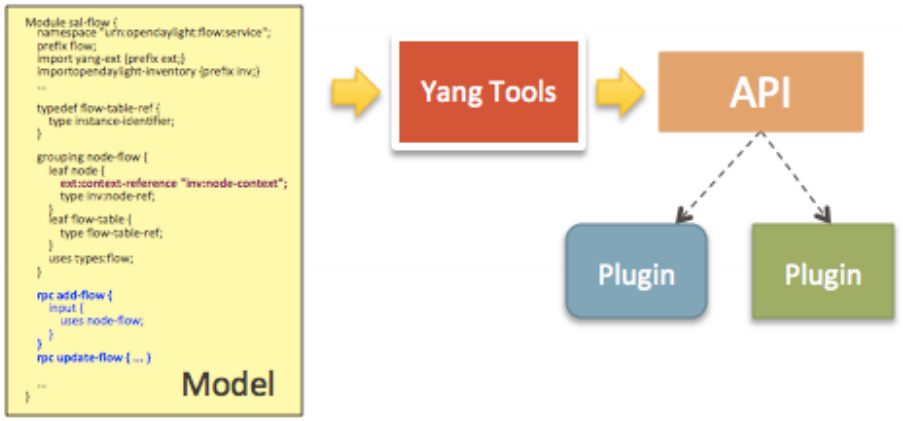


Model-Driven SAL (2/2)

- Model-Driven Service Abstraction Layer (SAL)
  - Yang tools – supporting for model-driven SAL
  - Provides tooling to build Java bindings in yang from yang models

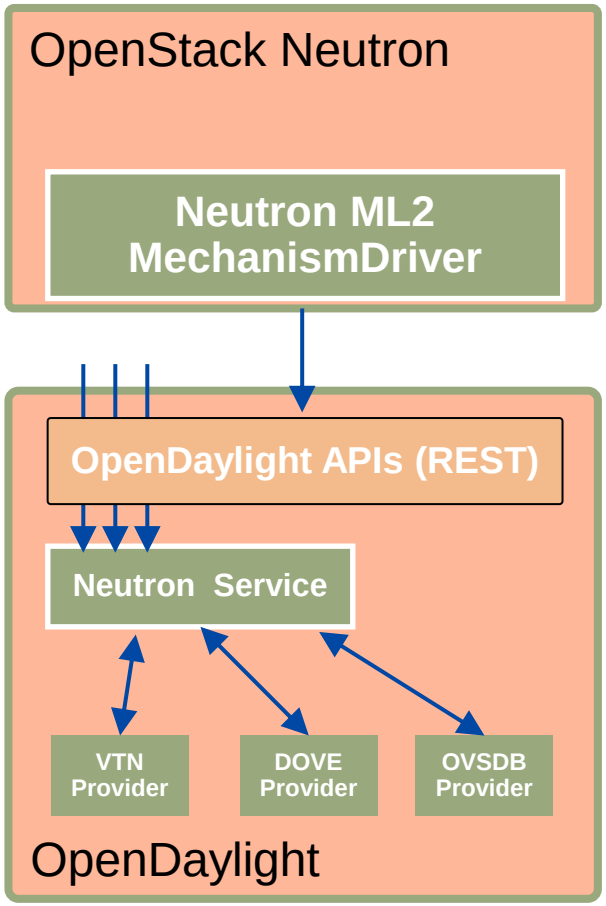


API-Driven SAL



Model-Driven SAL

OpenStack Integration



- OpenDaylight exposes a single common OpenStack Service Northbound
  - API exposed matches Neutron API precisely
  - multiple implementations of Neutron networks in OpenDaylight
  - The Modular Layer 2 (ml2) plugin is a framework allowing OpenStack Networking to simultaneously utilize the variety of layer 2 networking technologies found in complex real-world data centers.
- OpenDaylight OpenStack Neutron Plugin simply passes through
  - simplifies OpenStack plugin
  - pushes complexity to OpenDaylight

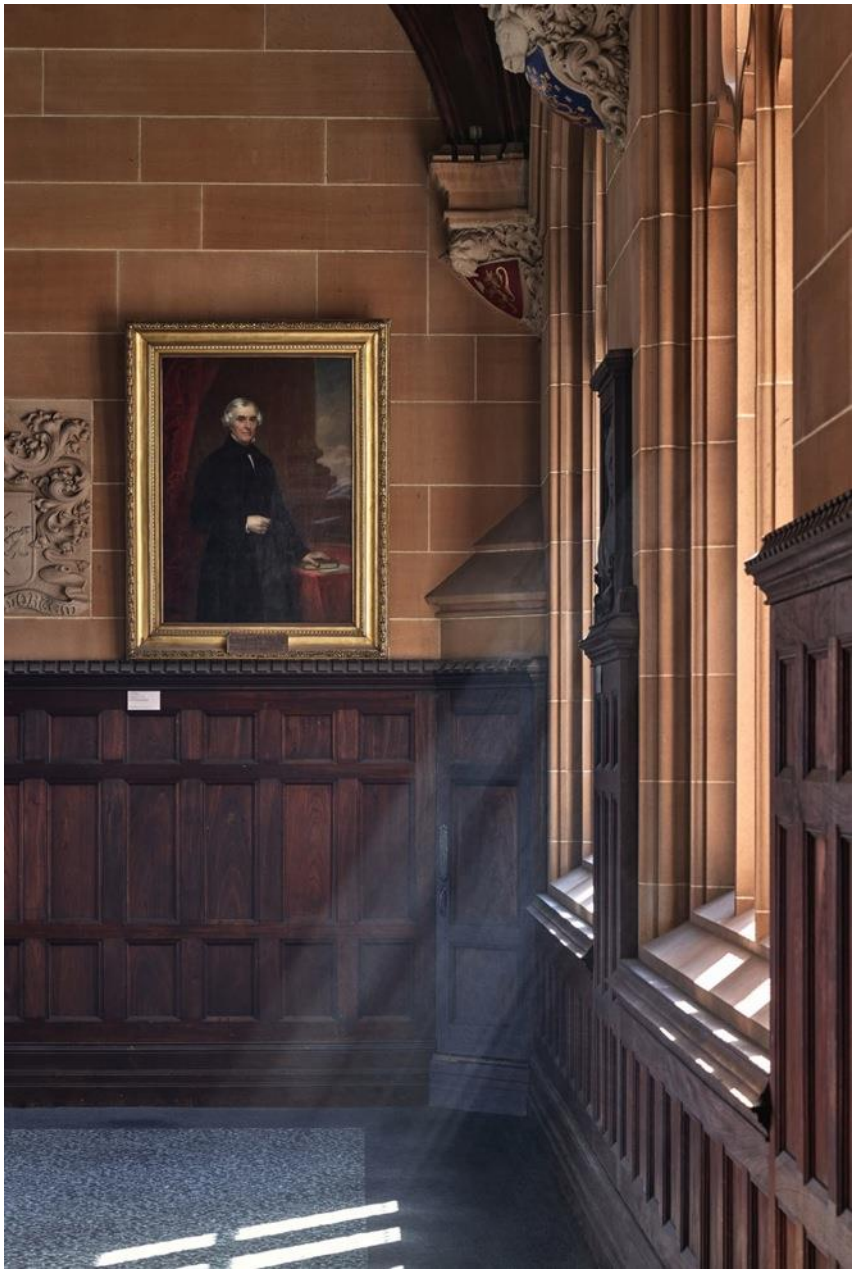
Thank you!

References:

https://www.scs.gatech.edu/news/195201/free-online-sdn-course

https://www.sdxcentral.com/sdn/?c_action=num_ball

https://www.opennetworking.org/



THE UNIVERSITY OF SYDNEY