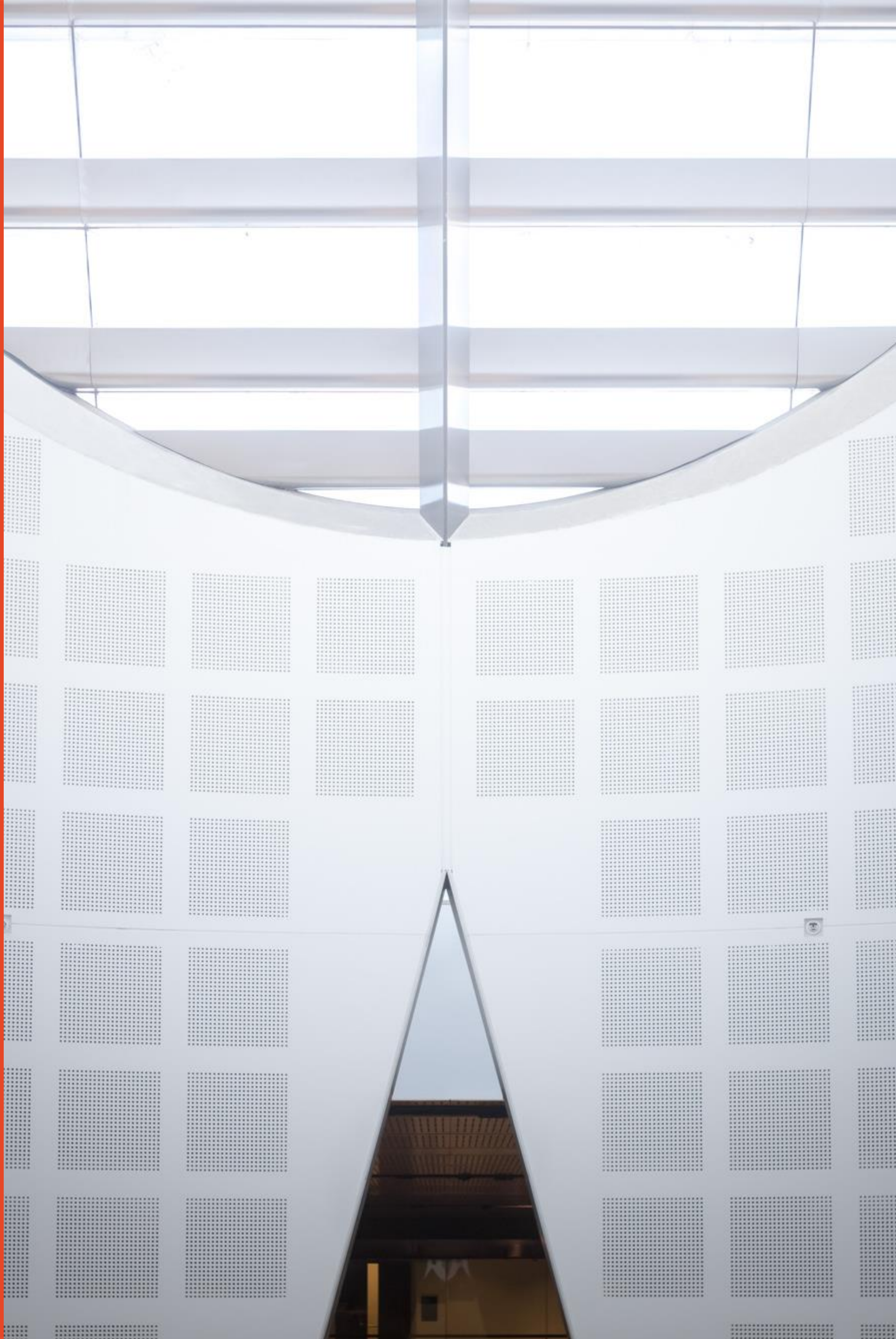
Topic 6:
ONOS Controller

Presented by
Dong YUAN

School of Electrical and Computer Engineering

dong.yuan@sydney.edu.au

THE UNIVERSITY OF SYDNEY



Today’s 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 and K8s
- 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

Network operating system

- It manages the resources on behalf of users
- It isolates and protects users from each other
- It provides useful abstractions
  - Easy access resource
  - Shield difference of devices
- It provides security from the external world to users
- It supplies useful services

- What information should a Network OS store?
  - Topology? status of network device? route calculated? network traffic?

Introduction to ONOS

- ONOS: Open Network Operating System
  - SDN OS for service provider networks
  - Key features
    - Scalability, high availability & performance
    - Northbound & southbound abstraction
    - Modularity
      - Various usage purposes, customization and development
  - History

ON.LAB

Founded – 2012

ONOS Prototype 1 – 2013

(scalability, high availability)

ONOS Prototype 2 – 2013

(performance)

ONOS VERSION 1 –
Open sourced on Dec 5th, 2014

ONOS Ecosystem

ON.LAB

- Non-profit, Carrier and vendor neutral
- Build core platform
- Provide technical shepherding, core team
- Build community

Service Providers

- Provide funding
- Provide requirements
- Develop use cases
- Drive POCs, deployments
- Bring vendors along

Vendors

- Provide funding
- Provide engineering resources
- Build products and solutions
- Provide integration, test and support services

Community

- Drive every aspect- technical, process, roadmap, deployments
- Bring in diversity
- Help ONOS evolve and thrive

Prior Work

Single Instance

NOX, POX, Beacon, Floodlight, Trema controllers

Helios, Midonet, Hyperflow, Maestro, Kandoo, …

Distributed: ONIX

Distributed control platform for large-scale networks

ONIX: closed source; datacenter + virtualization focus

ONOS design influenced by ONIX

Community needs an open source distributed network OS

Design Goals

ONOS is a multi-module project whose modules are managed as OSGi bundles, leverages Apache Karaf as application container, and enables YANG schema language to become a programming language.

- Same as the architecture of OpenDayLight

- Code Modularity : It should be possible to introduce new functionalities as self-contained units.

- Configurability : It should be possible to load and unload various features, whether it be at startup or at runtime.

- Separation of Concern : There should be clear boundaries between subsystems to facilitate modularity.

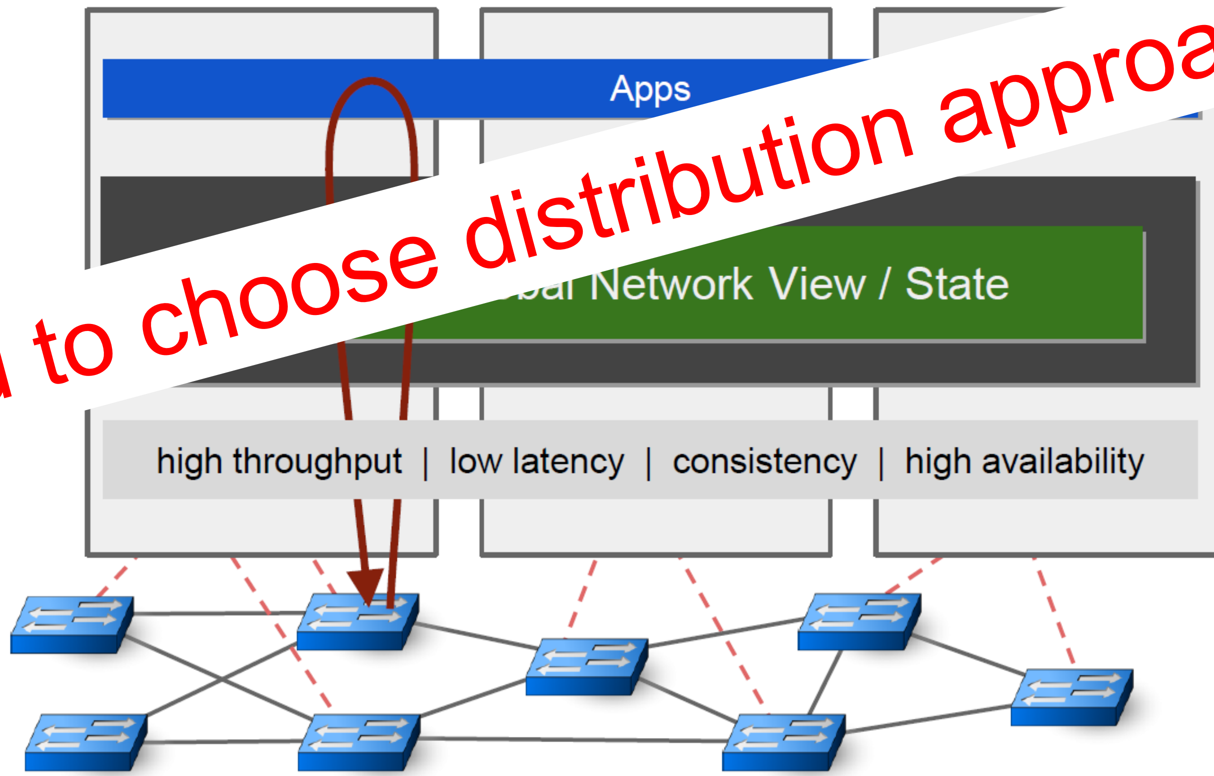
- Protocol agnosticism : It, and its applications, should not be bound to specific protocol libraries or implementations.

ONOS functions

- Distributed Network OS
  - Network Graph Northbound Abstraction
  - Horizontally Scalable
  - Highly Available
  - Built using open source components
- Exploring performance & reactive computation frameworks
- Expand graph abstraction for more types of network state
- Control functions: intra-domain & inter-domain routing

Key Performance Requirements

- Requirements for Supporting Service Provider Networks
  - High throughput
    - 500K ~ 1M paths setups / second, 3-6M network state operations / second
  - High volume
    - 500GB ~ 1TB of network state data



Need to choose distribution approach!

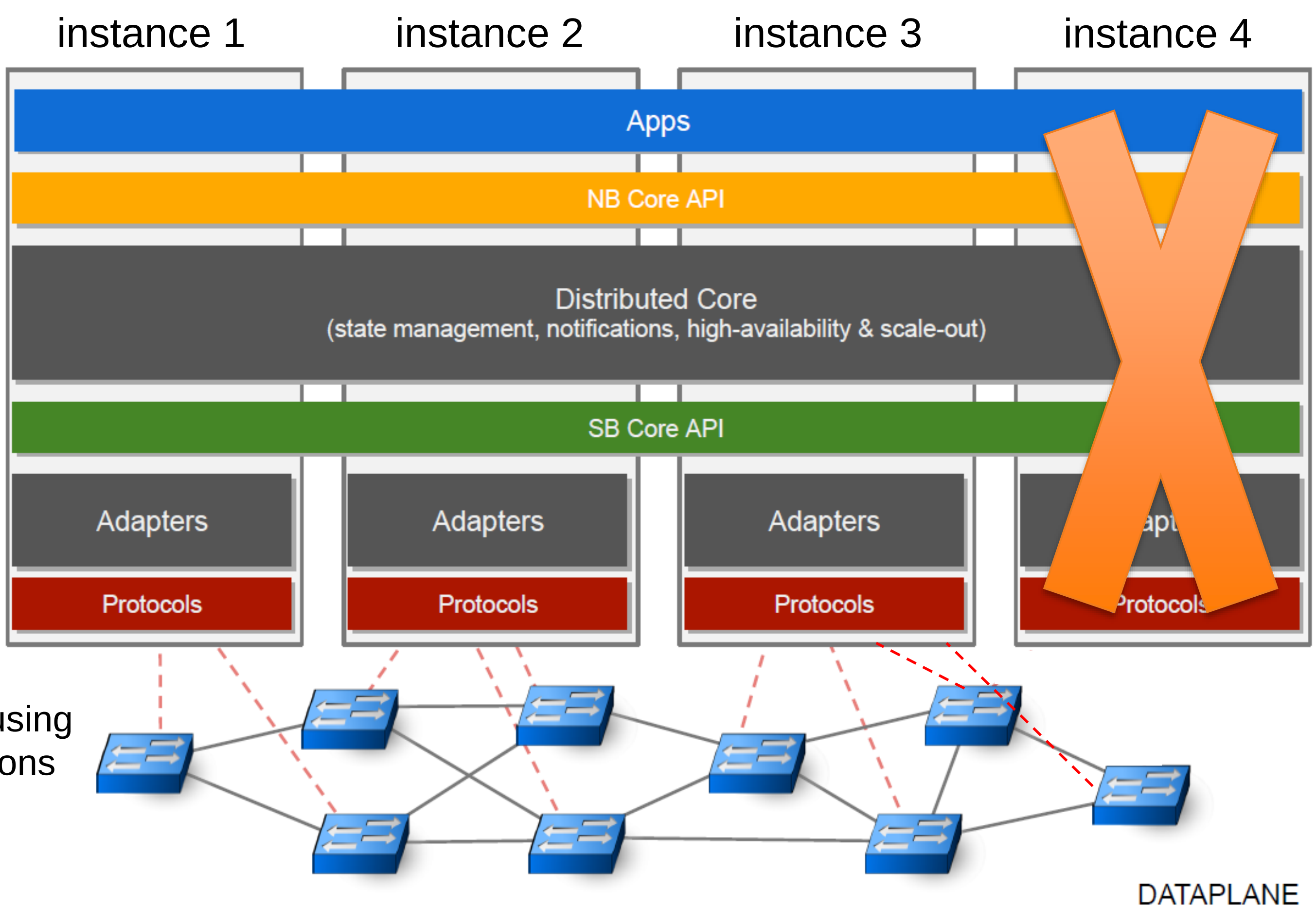
Key features of ONOS

- Distributed Core
  - Scalability, availability and performance
- Northbound Abstraction/APIs
  - Ease the development of control, management and configuration services.
- Southbound abstraction/APIs
  - Enable pluggable southbound protocols fro controlling both openflow and legacy devices.
- Software Modularity
  - Easy for community developers to develop, maintain, debug and upgrade ONOS

ONOS Tiers and Distributed Architecture

– Distributed Architecture
  – Six tiered architecture
  – Each ONOS instance is equipped with the same software stack

- Northbound Abstraction
  - Network graph
  - Application intents
- Core
  - Distributed
  - Protocol independent
- Southbound Abstraction
  - Generalized OpenFlow
  - Pluggable & extensive
- Adapters
  - Multiple southbound protocol enabling layer
- Protocols
  - Self-defined protocols using generalized SDN functions



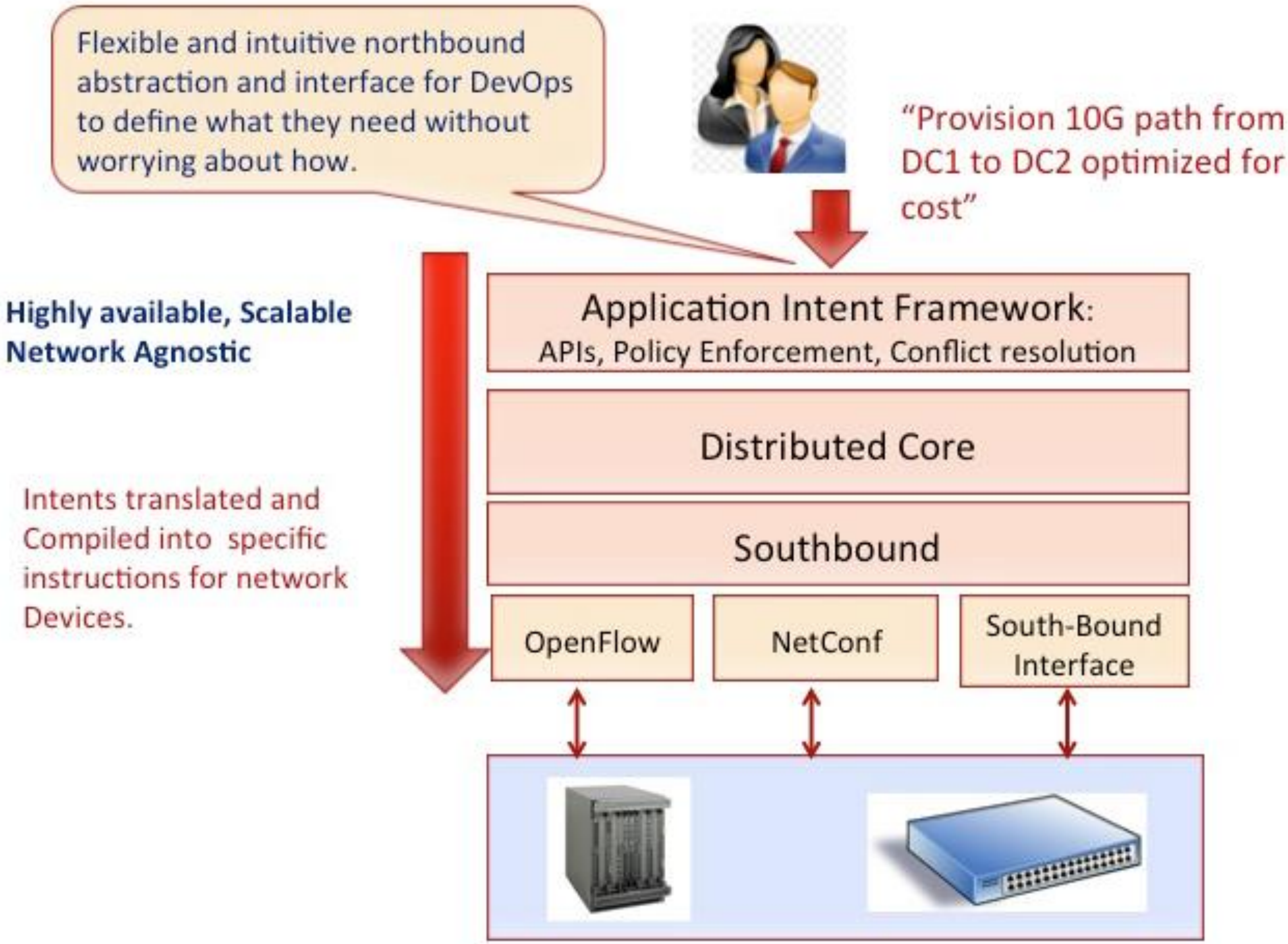
Distributed Cores

- ONOS is deployed as a service on a cluster of servers, same software runs on each server.
- The operator can add servers incrementally, as needed for control plane capacity.
- The ONOS instances work together to create what appears as a single platform.
  - Applications and devices do not have to know if they are working with a single instance or multiple instances
  - This feature makes ONOS scalable
  - The distributed Core does heavy lifting to realise this

Northbound abstraction

- Two powerful northbound abstractions
  - Intent Framework and Global Network View
- The Intent Framework allows an application to request a service from the network without having to know its details.
  - Allow operators and application developers to program the network at a high level
  - They just simply specify their intent, i.e. a script.
- The Global Network View provides a view of the network – the hosts, switches, links, etc.
  - Applications can program this network view through APIs

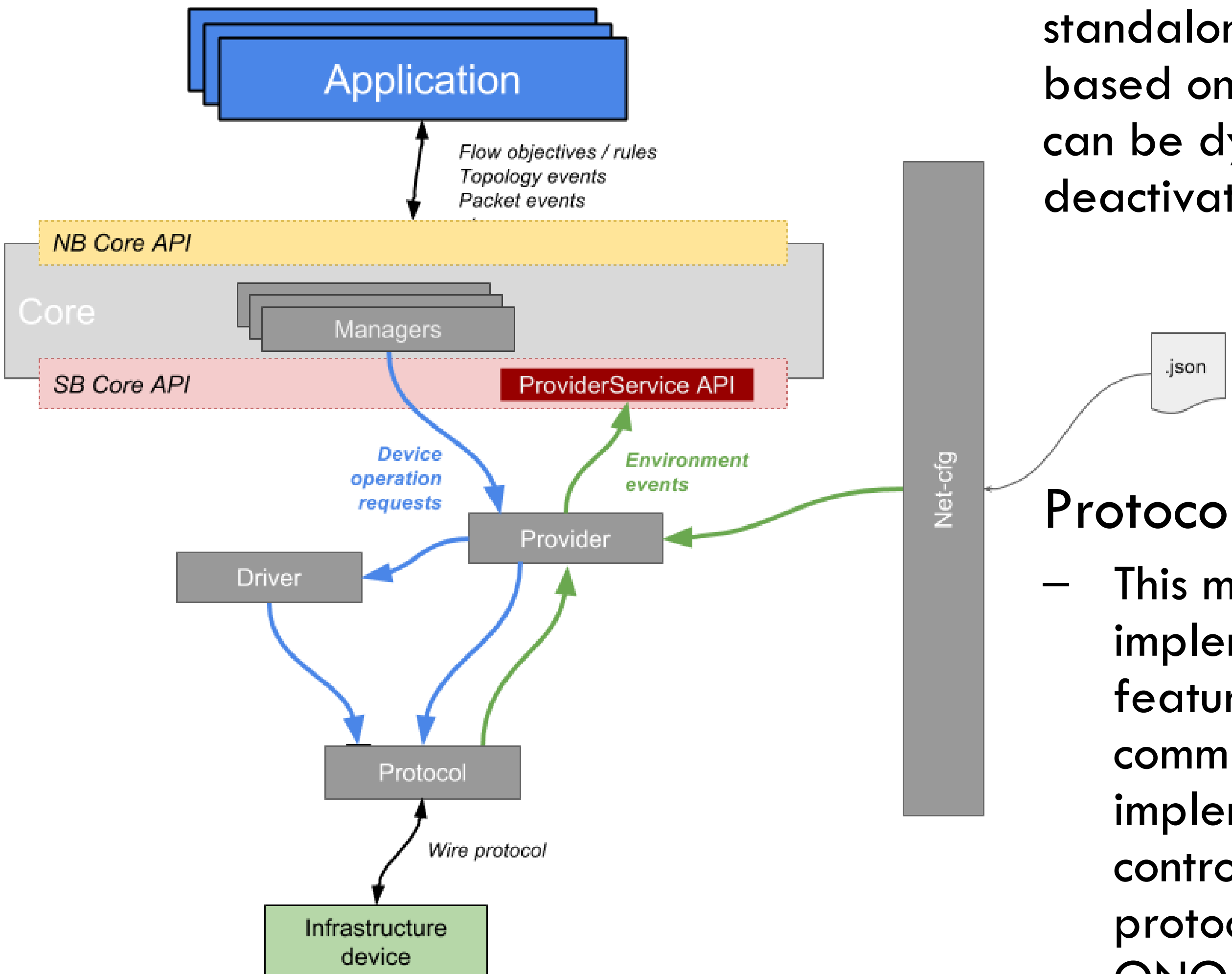
Northbound abstraction



Southbound Abstraction

- The southbound abstraction of ONOS represents each network element as an object in a generic form.
- Allow plug-ins for various southbound protocols and devices.
- Ability to manage different devices using different protocols.
- Ability to add new devices and protocols to the system.
- Ease for migration from legacy devices and protocols

Southbound Design



Provider

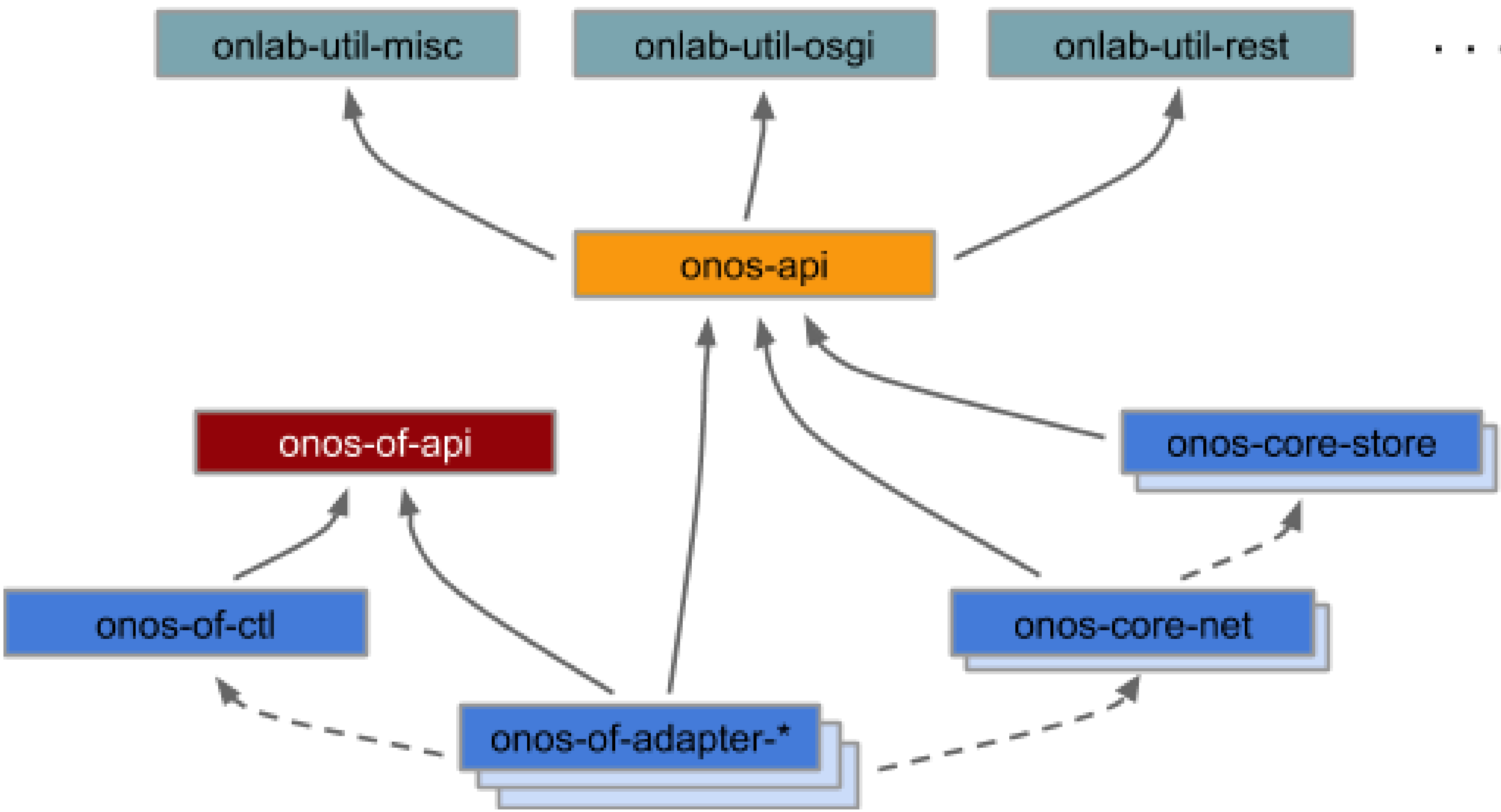
– Providers in ONOS are standalone ONOS applications based on OSGi components that can be dynamically activated and deactivated at runtime.

Protocol

– This module contains the implementation of all those features needed by ONOS to communicate with devices implementing a specific runtime control and/or management protocol. Examples of what ONOS considers protocols are OpenFlow, NETCONF, SNMP, OVSDB, etc.

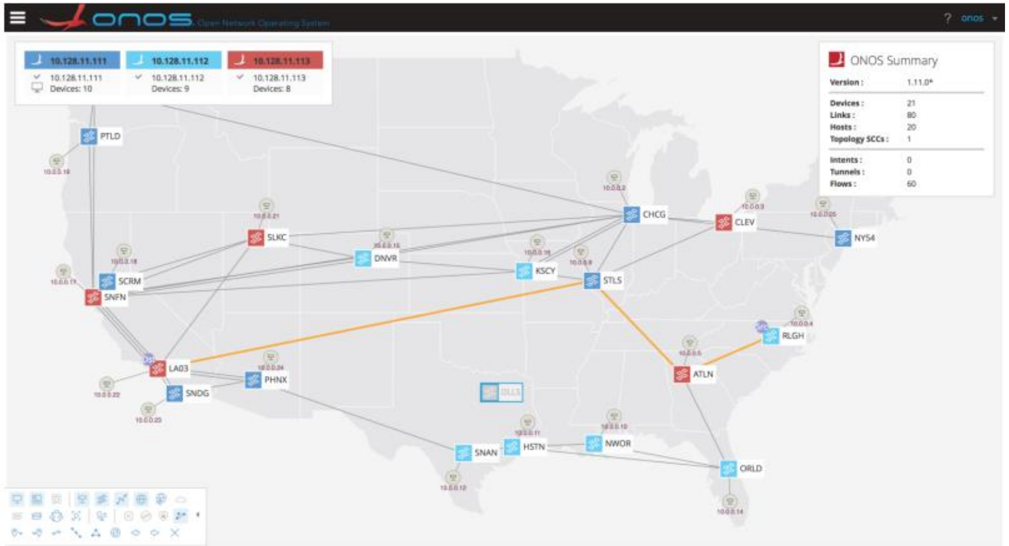
Software Modularity

– It is how the software is structured into components and how those components relate to one another.

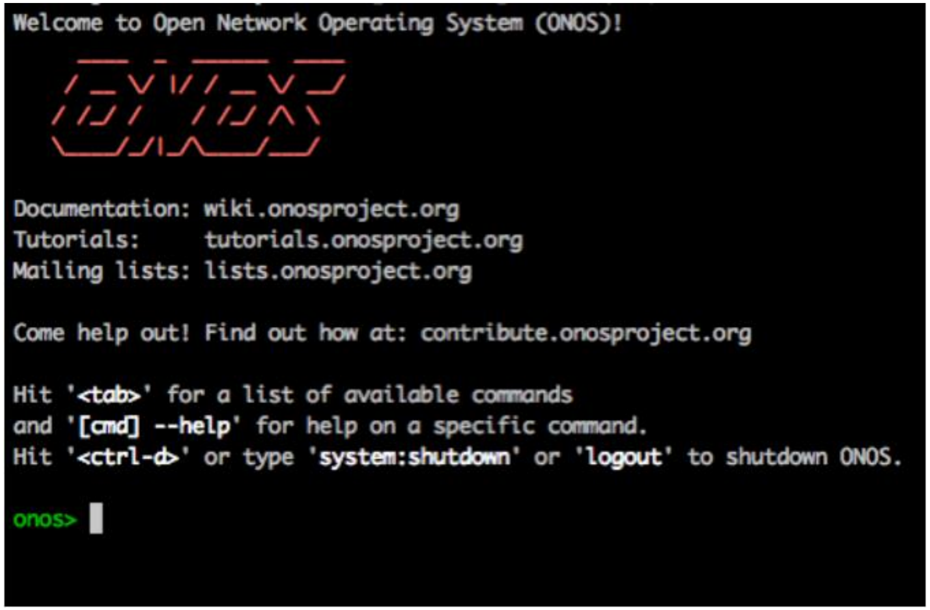


Interacting with ONOS

- ONOS Web GUI
- (http://...:8181/onos/ui)

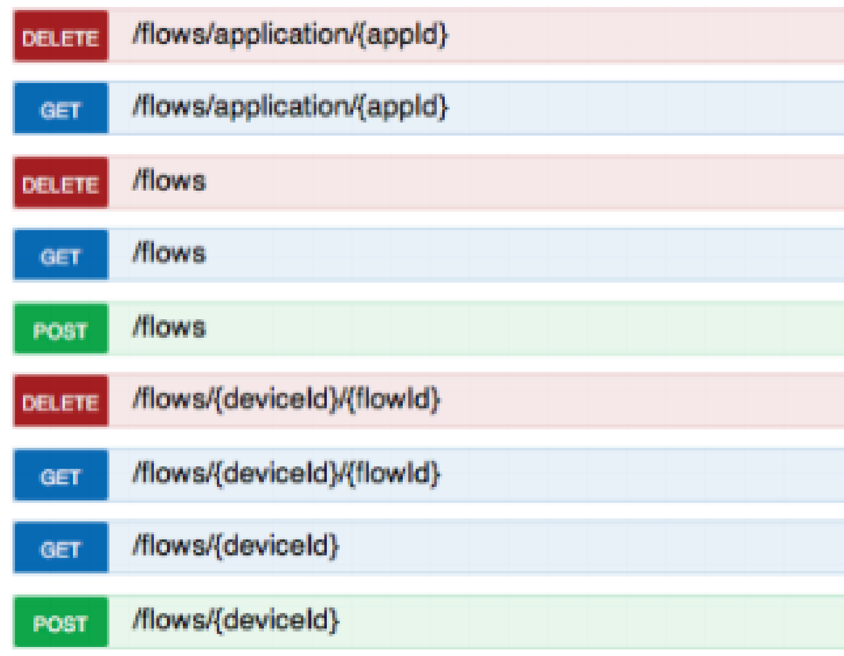


- ONOS CLI
  - An extension of Karaf's CLI and Leverage features such programmatic extensibility



Interacting with ONOS

- REST API
  - Provides a way to interact with off-platform applications
  - JSON, HTTP/1.1 based communication
  - Swagger based REST documents



- gRPC
  - Faster access than REST calls by using HTTP/2 connection multiplexing and bidirectional streaming

ONOS High Level Architecture

Network Graph
Eventually consistent

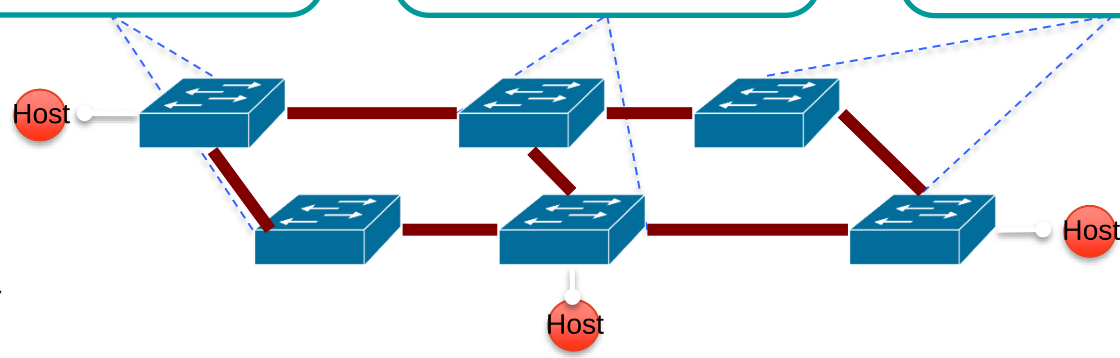
Titan Graph DB

Cassandra In-Memory DHT

Distributed Registry
Strongly Consistent

Zookeeper

Instance 1	Instance 2	Instance 3
OpenFlow Controller+	OpenFlow Controller+	OpenFlow Controller+

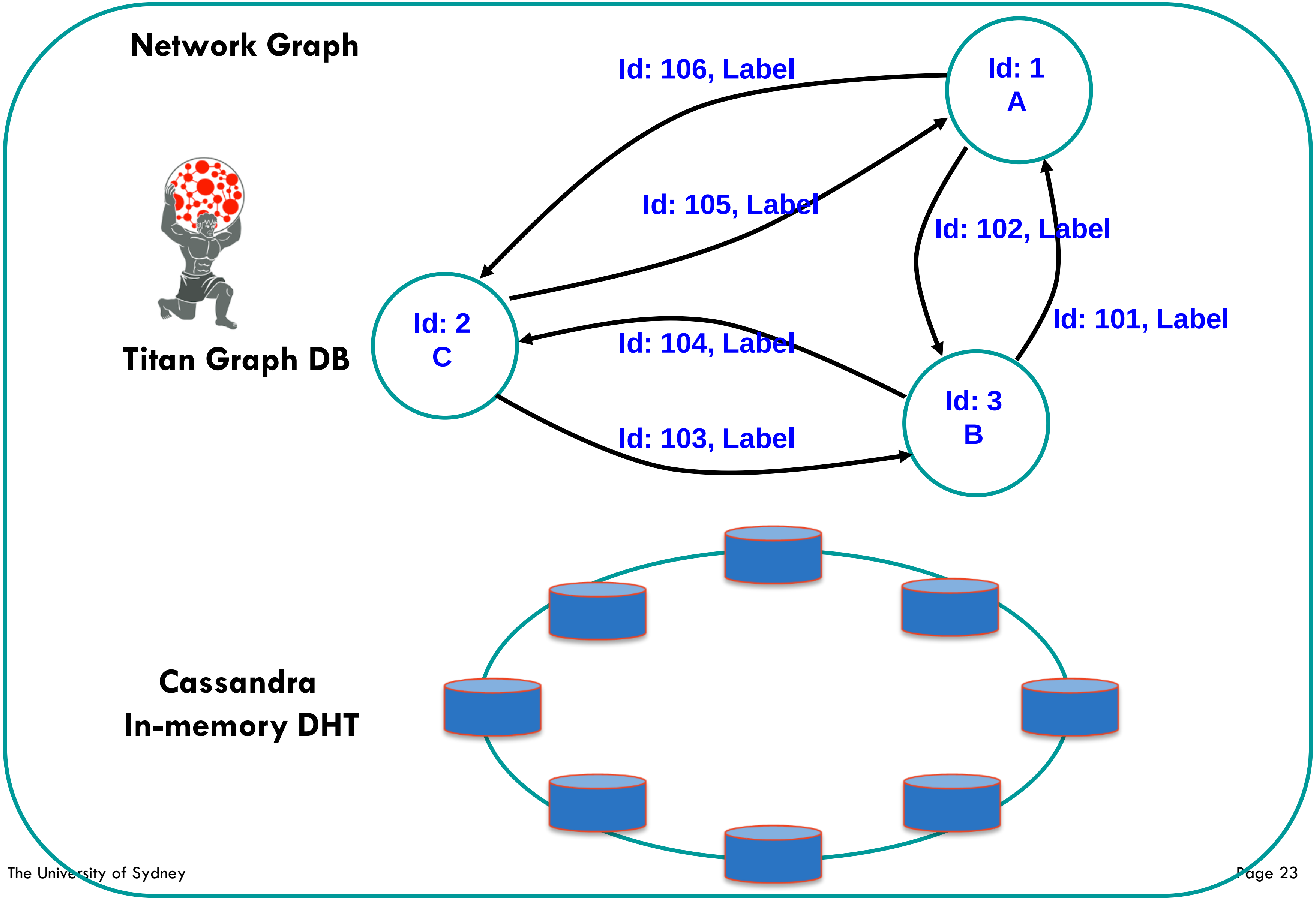


The software

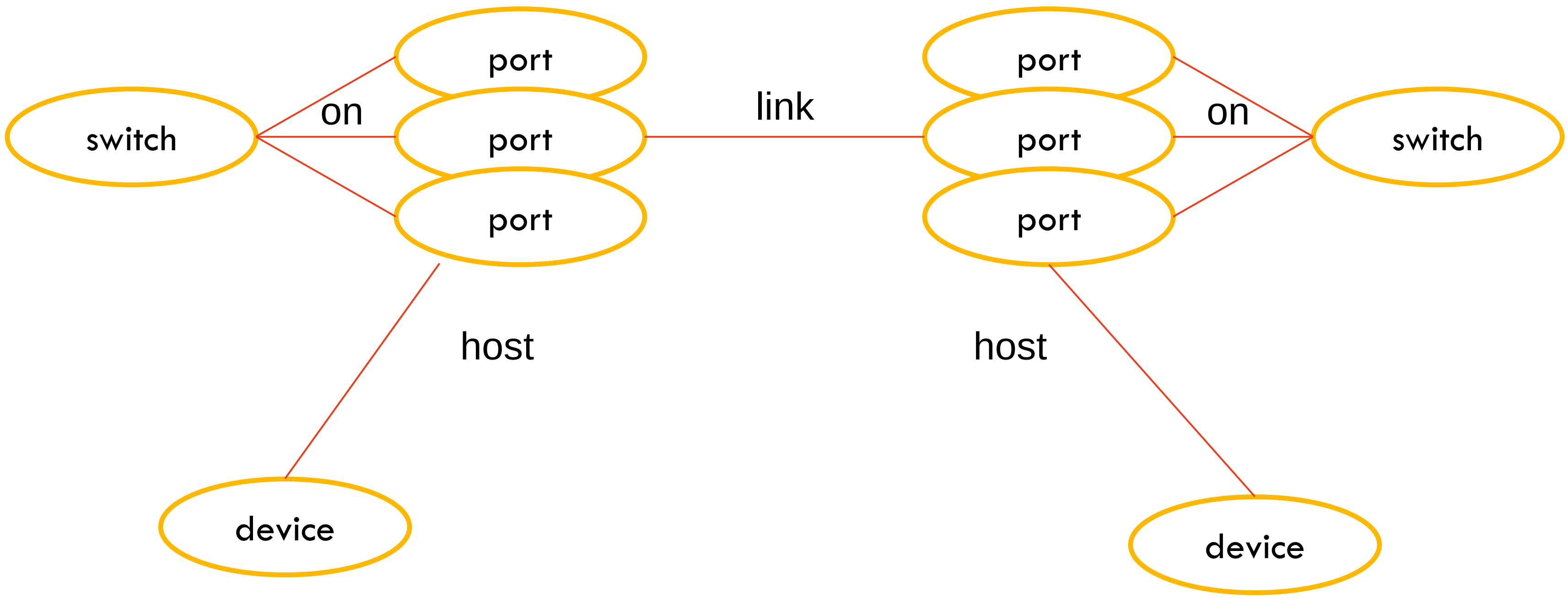
- **Titan Graph Database**
  - Titan is a scalable graph database optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster.
  - Titan is a transactional database that can support thousands of concurrent users executing complex graph traversals in real time.

- **Apache Cassandra**
- A free and open-source distributed wide column store NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.
- Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients.

ONOS Network Graph Abstraction

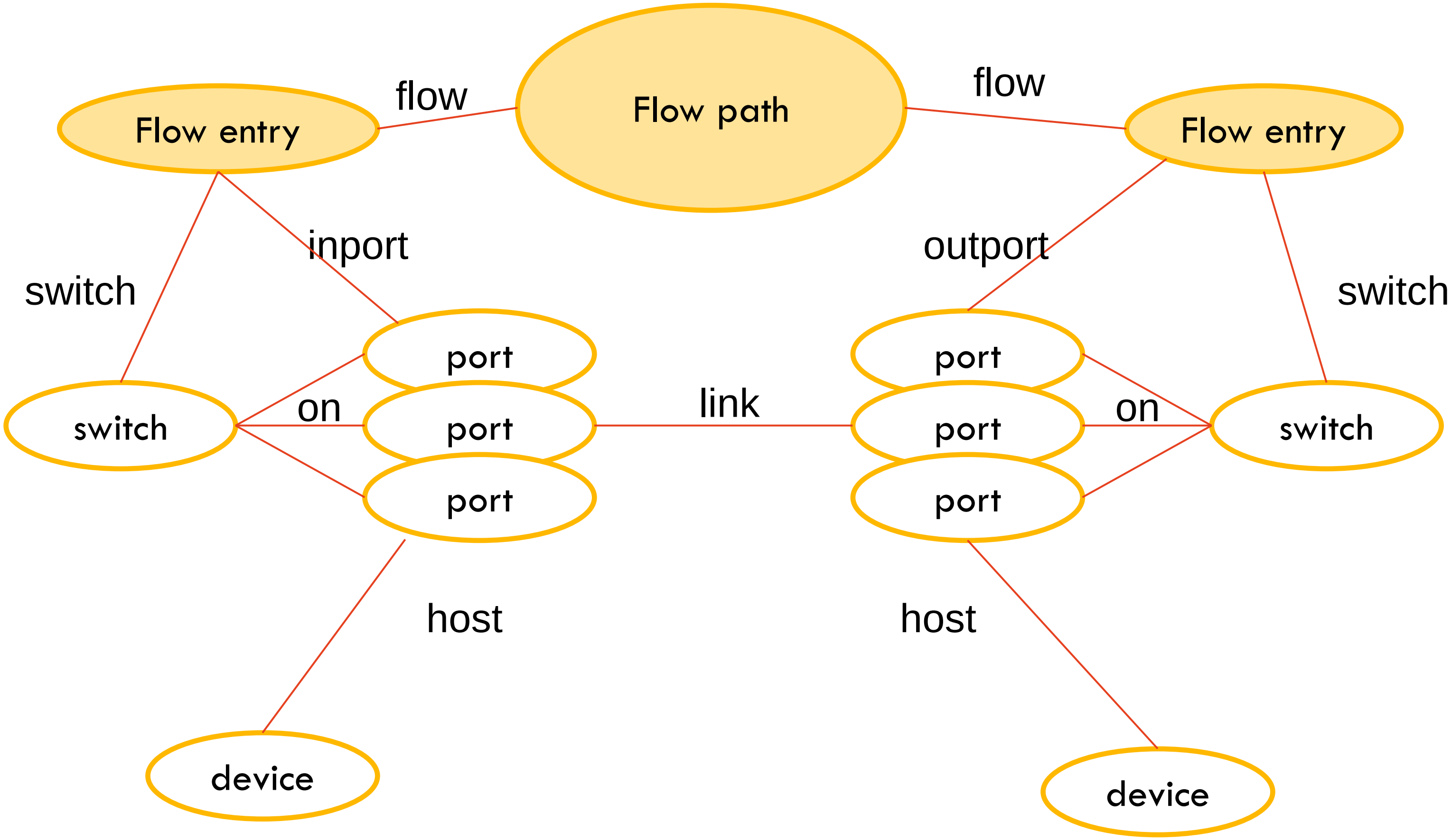


Network Graph



- Network state is naturally represented as a graph
- Graph has basic network objects like switch, port, device and links
- Application writes to this graph & programs the data plane

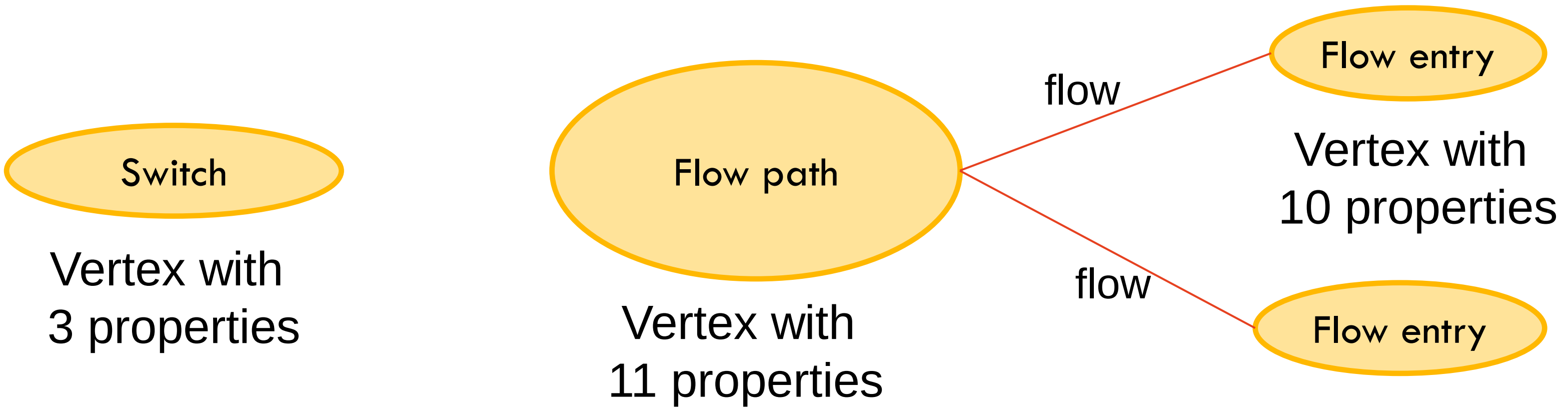
Example: Path Computation App on Network Graph



- **Application computes path by traversing the links from source to destination**
- **Application writes each flow entry for the path**

Thus path computation app does not need to worry about topology maintenance

Network Graph Representation



Vertex represented as Cassandra row

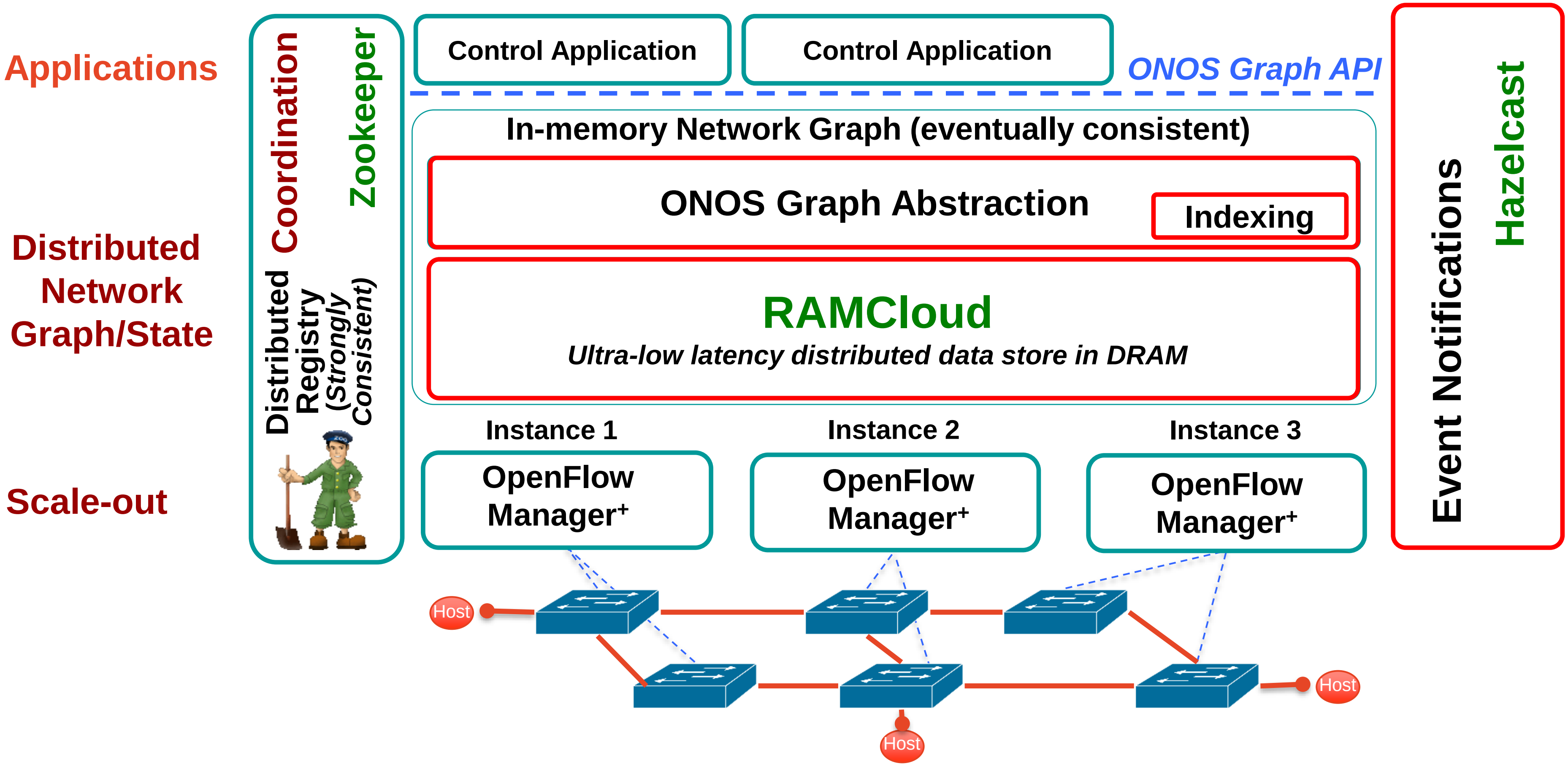
Property (e.g. dpid)	Property (e.g. state)	Property	…	Edge	Edge

Row indices for fast vertex centric queries

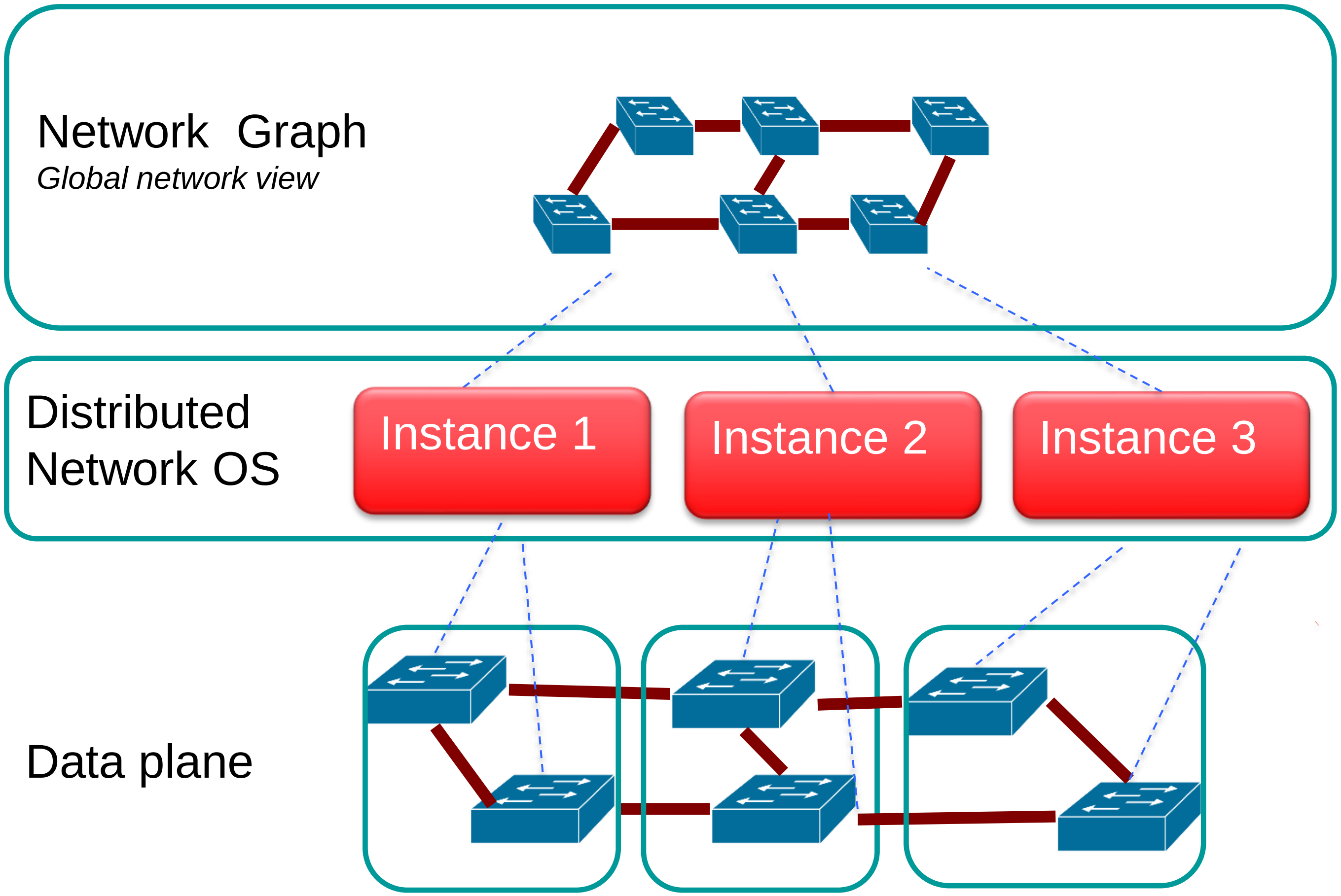
Edge represented as Cassandra column

Column				Value	
Label id + direction	Primary key	Edge id	Vertex id	Signature properties	Other properties

ONOS Architecture (Prototype 2)



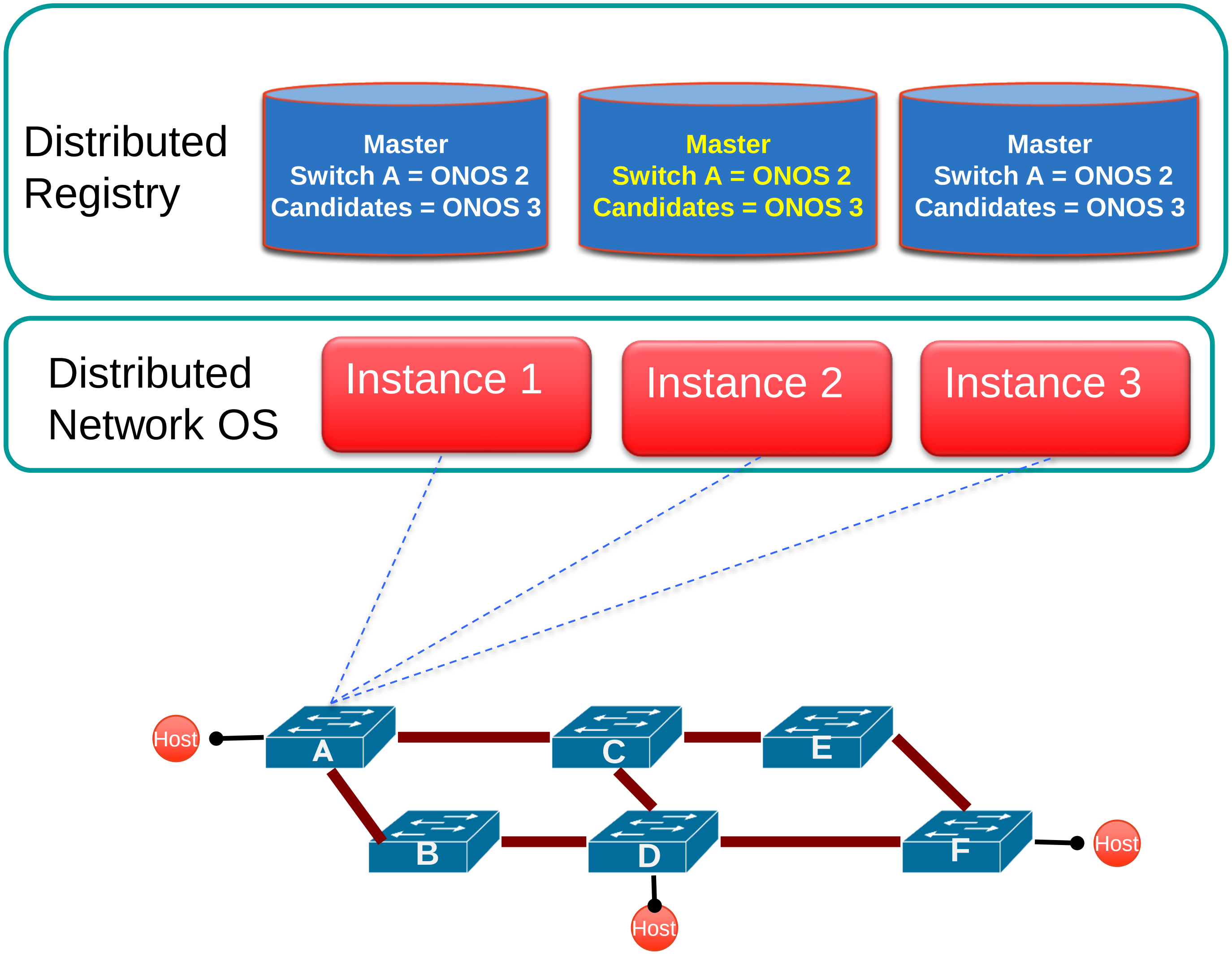
ONOS Scale-Out



An instance is responsible for maintaining a part of network graph

Control capacity can grow with network size or application need

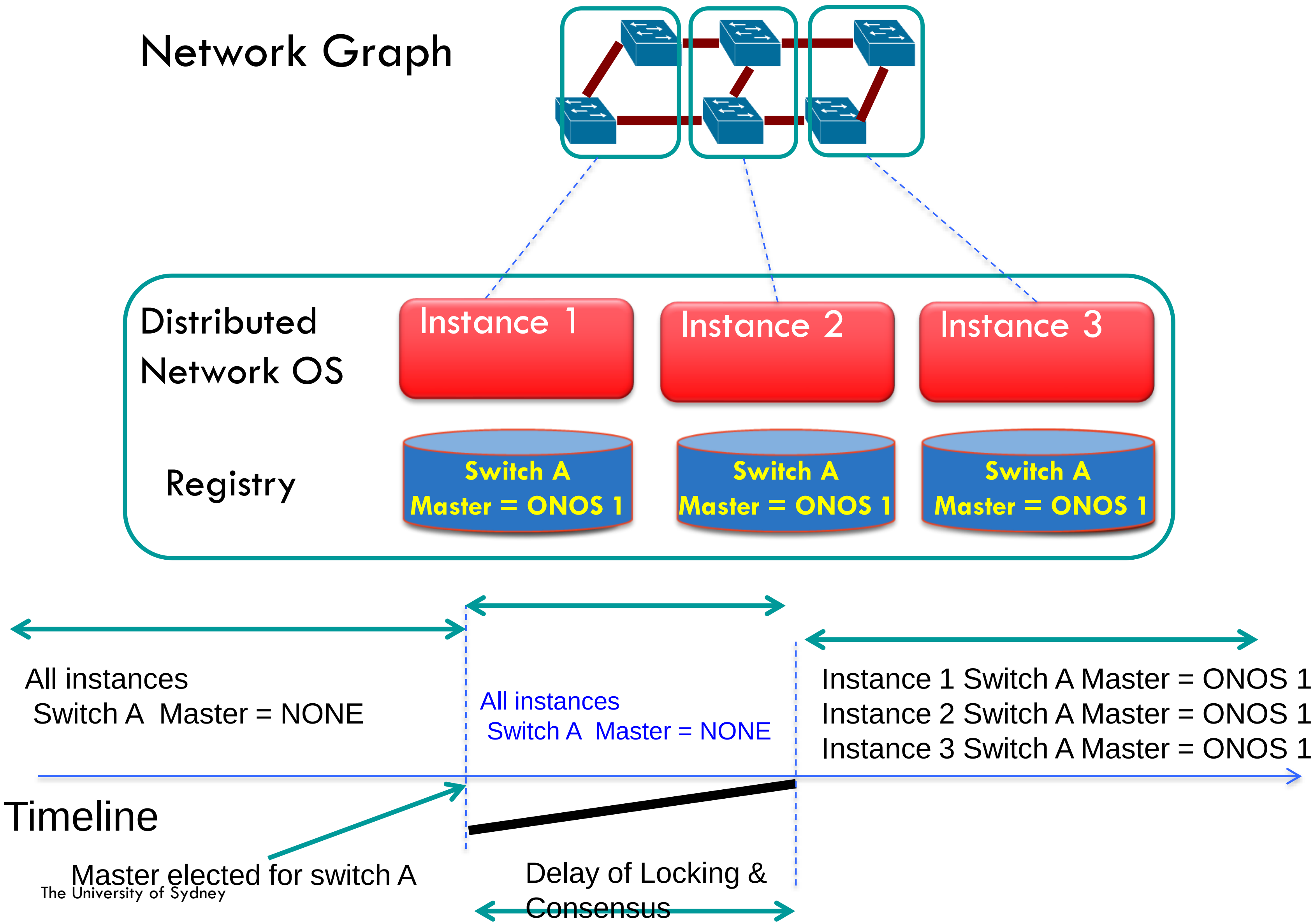
ONOS Control Plane Failover



Consistency Definition

- Strong Consistency: Upon an update to the network state by an instance, all subsequent reads by any instance returns the last updated value.
- Strong consistency adds complexity and latency to distributed data management.
- Eventual consistency is slight relaxation – allowing readers to be behind for a short period of time.

Strong Consistency using Registry



Why Strong Consistency is needed for Master Election

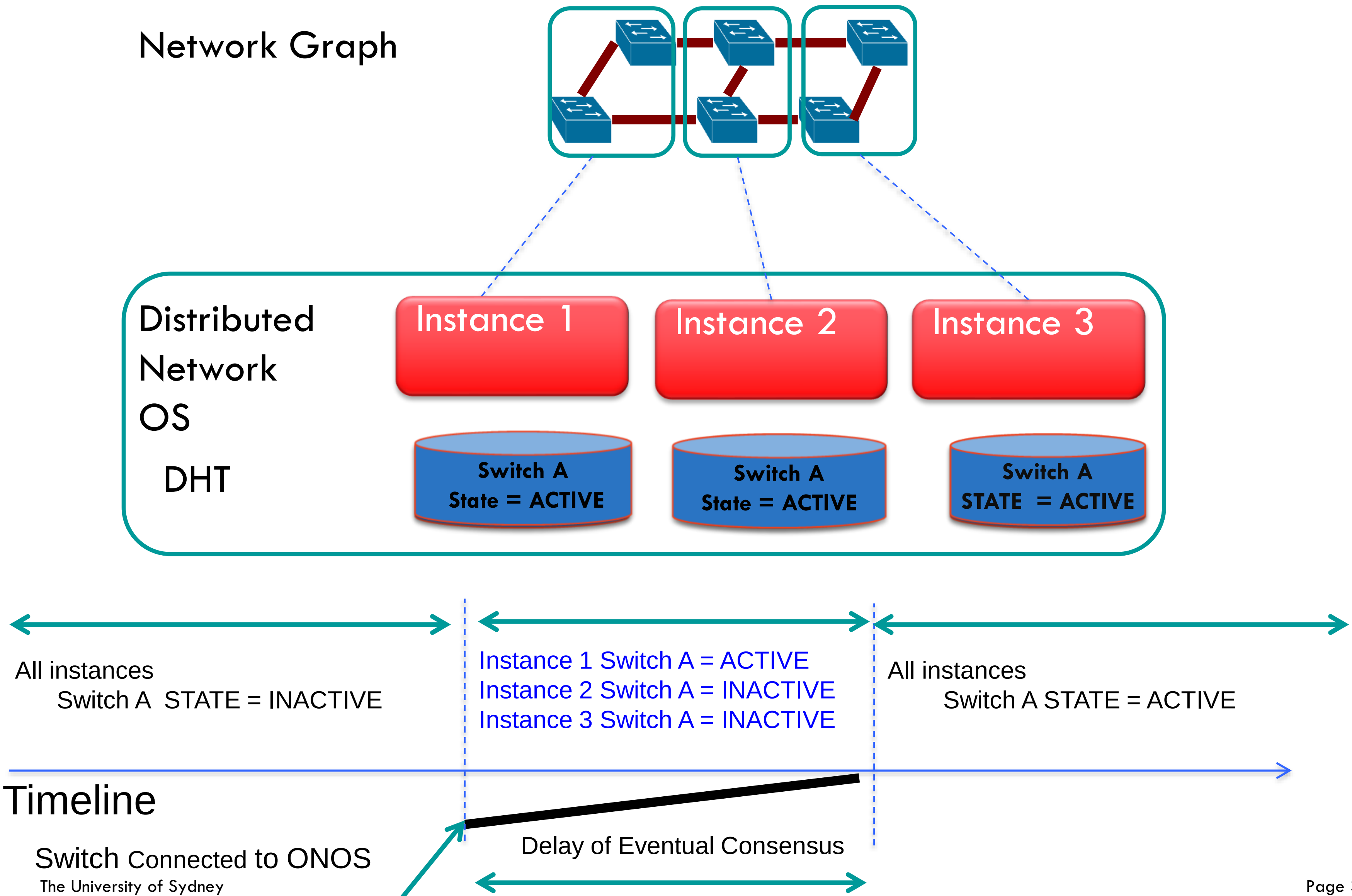
- Weaker consistency might mean Master election on instance 1 will not be available on other instances.

- That can lead to having multiple masters for a switch.

- Multiple Masters will break the semantic of control isolation.

- Strong locking semantic is needed for Master Election

Eventual Consistency in Network Graph



Cost of Eventual Consistency

- Short delay will mean the switch A state is not ACTIVE on some ONOS instances in previous example.
- Applications on one instance will compute flow through the switch A while other instances will not use the switch A for path computation.
- Eventual consistency becomes more visible during control plane network congestion.

Why is Eventual Consistency good enough for Network State?

- Physical network state changes asynchronously
  - Strong consistency across data and control plane is too hard
  - Control apps know how to deal with eventual consistency

- In the current distributed control plane, each router makes its own decision based on old info from other parts of the network and it works fine

- Strong Consistency is more likely to lead to inaccuracy of network state as network congestions are real.

Consistency learning

- One Consistency does not fit all
- Consequences of delays need to be well understood
- More research needs to be done on various states

  using different consistency models

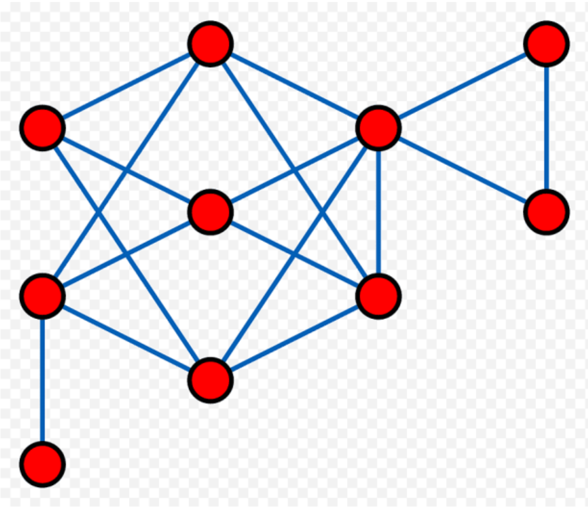
ONOS Distributed Core

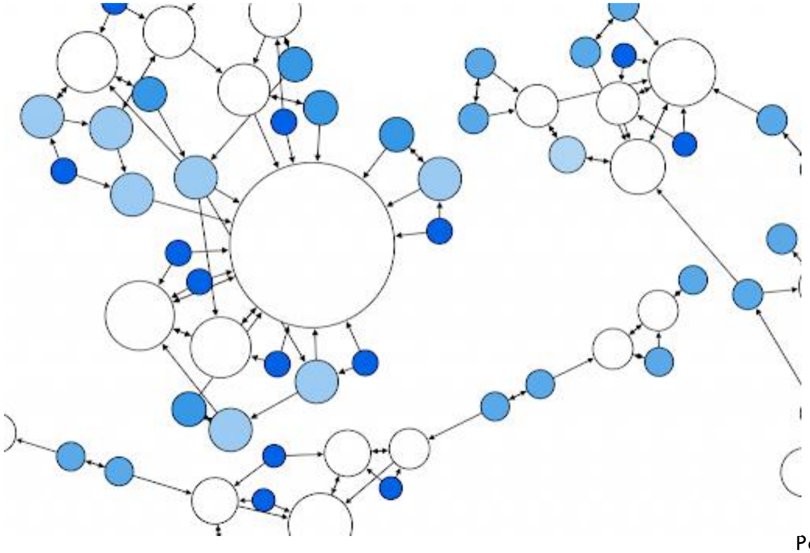
- Distributed Core
  - Responsible for all state management concerns
  - Organized as a collection of "STORES"
    - E.g., topology, links, link resources and etc.
  - State management choices (ACID vs. BASE)
    - ACID (Atomicity, Consistency, Isolation, Durability)
    - BASE (Basically Available, Soft state, Eventually consistency)
- State and Properties

State	Properties
Network Topology	Eventually consistent, low latency access
Flow Rules, Flow Stats	Eventually consistent, shardable, soft state
Switch – Controller Mapping Distributed Locks	Strongly consistent, slow changing
Application Intents Resource Allocations	Strongly consistent, durable

Controller Placement in SDN

– How many controller should we use and where to place them?
  – Optimisation objectives: Average latency, resilience, QoS, throughput, etc.
  – Constrains: Controller Locations, Processing Latency, Number of Controllers, Switch load, Traffic Profile, etc.
  – Graph Model: G (V, E, C), . Let $\phi : V \rightarrow C$ return the controller assigned to a switch, i.e., switch v is assigned to controller $\phi(v)$.





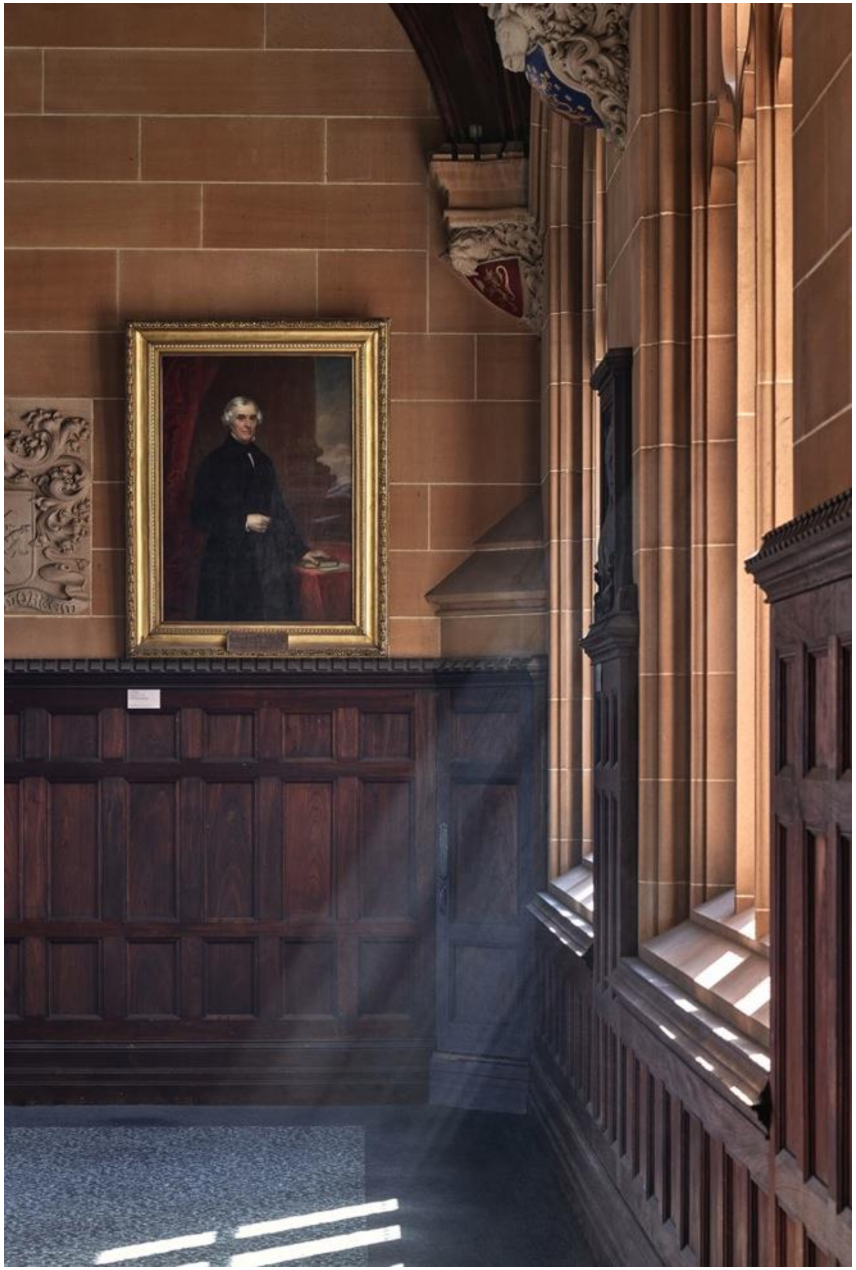
Thank you!

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