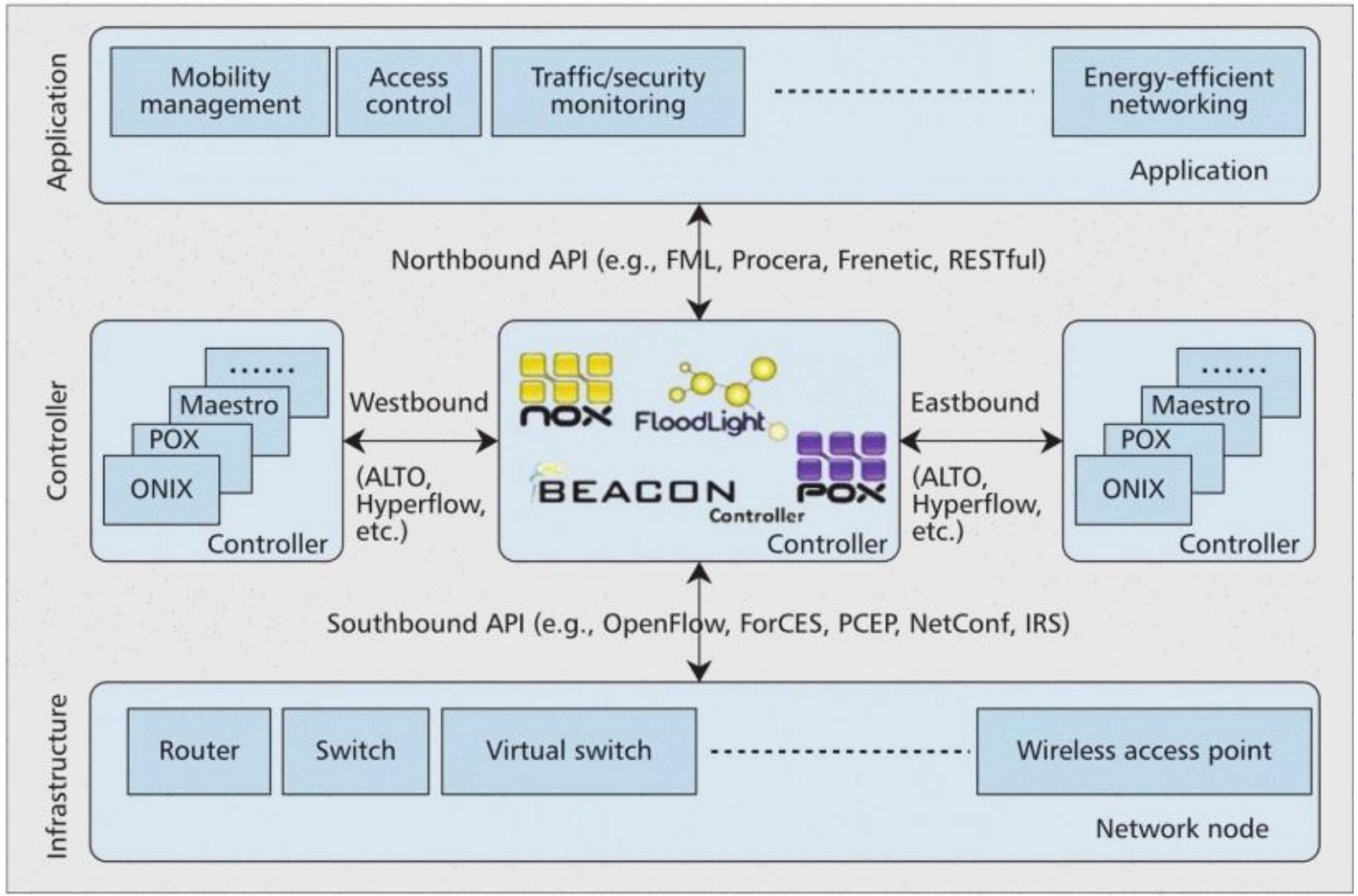
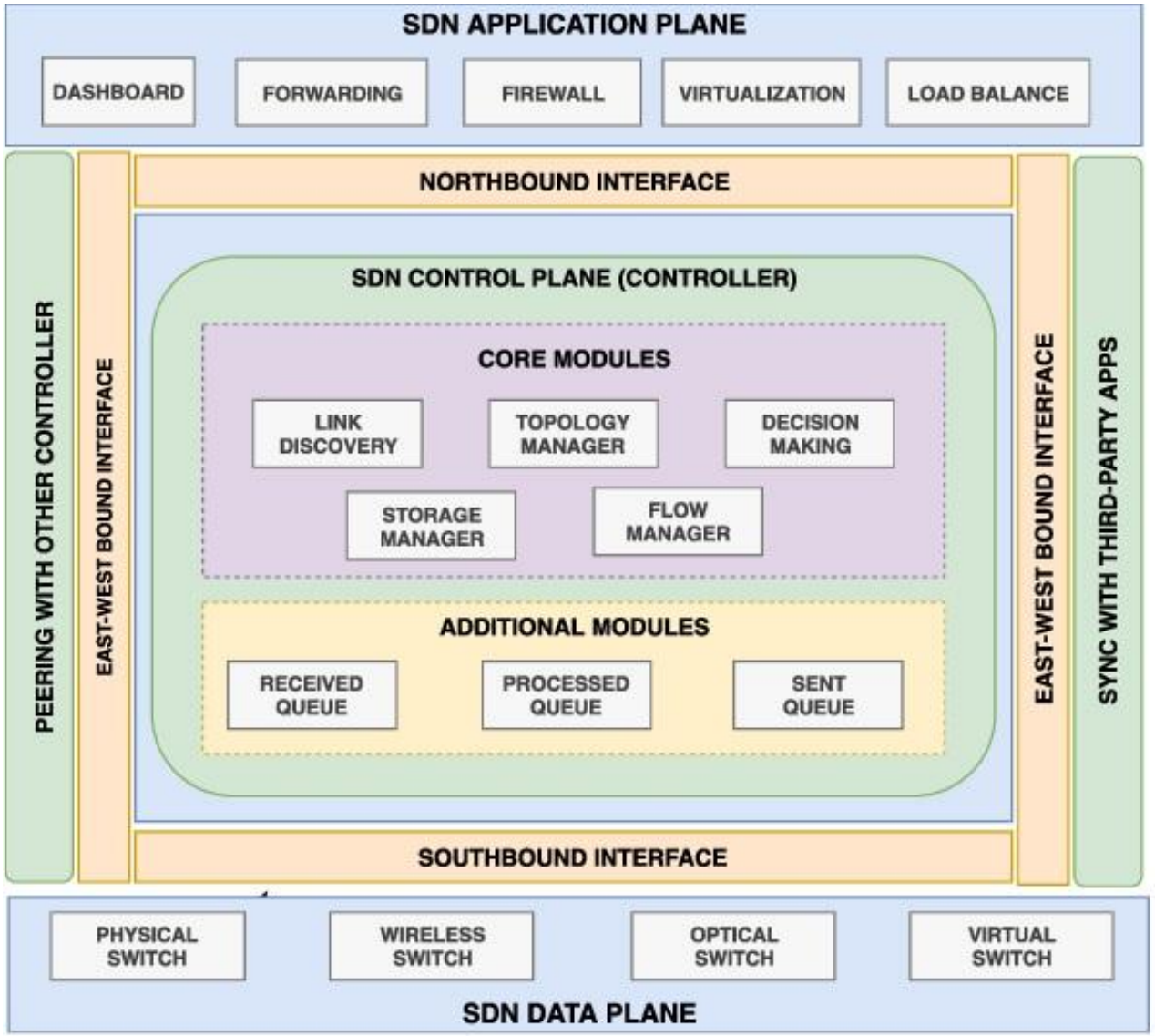
SDN CONTROLLER

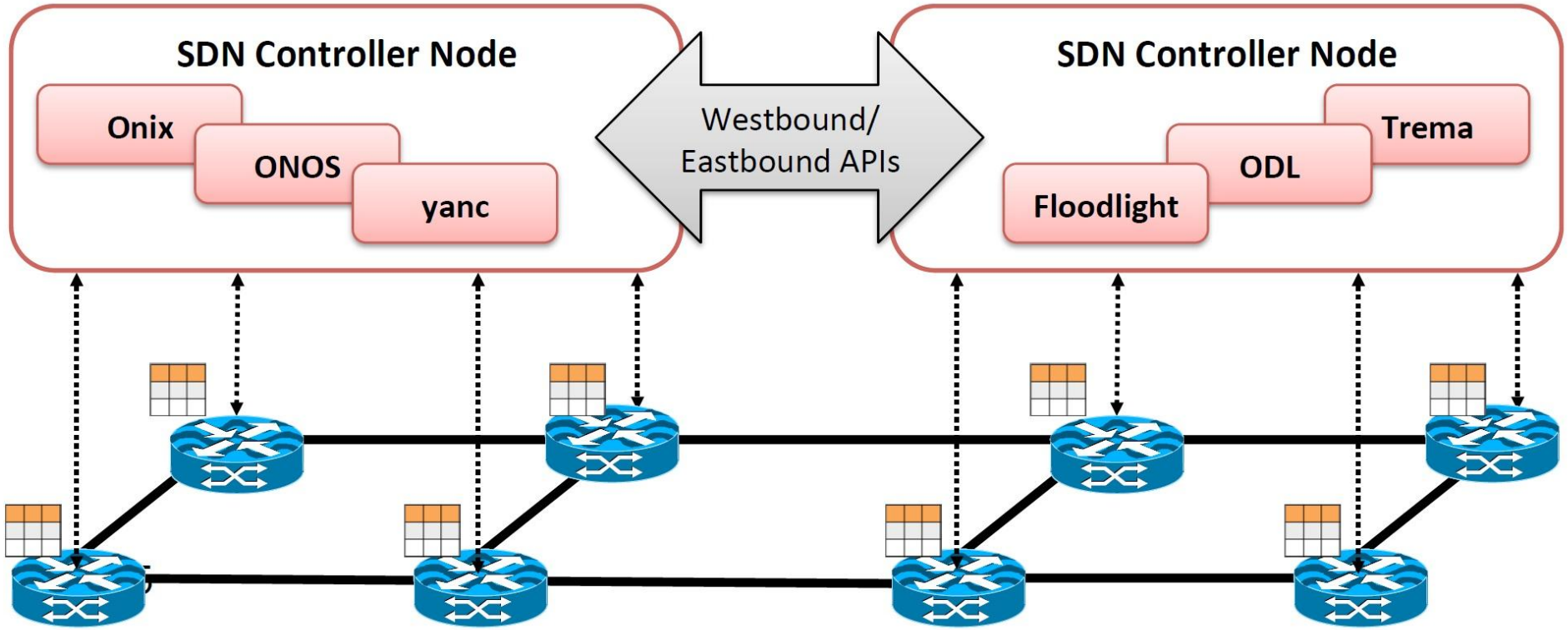
https://aptira.com/sdn-controller-comparison/


Application
Mobility management
Access control
Traffic/security monitoring
Energy-efficient networking
Application
Northbound API (e.g., FML, Procera, Frenetic, RESTful)
Controller
Maestro
POX
ONIX
Controller
Westbound
(ALTO, Hyperflow, etc.)
nox
FloodLight
BEACON
POX
Controller
Controller
Eastbound
(ALTO, Hyperflow, etc.)
Maestro
POX
ONIX
Controller
Southbound API (e.g., OpenFlow, ForCES, PCEP, NetConf, IRS)
Infrastructure
Router
Switch
Virtual switch
Wireless access point
Network node

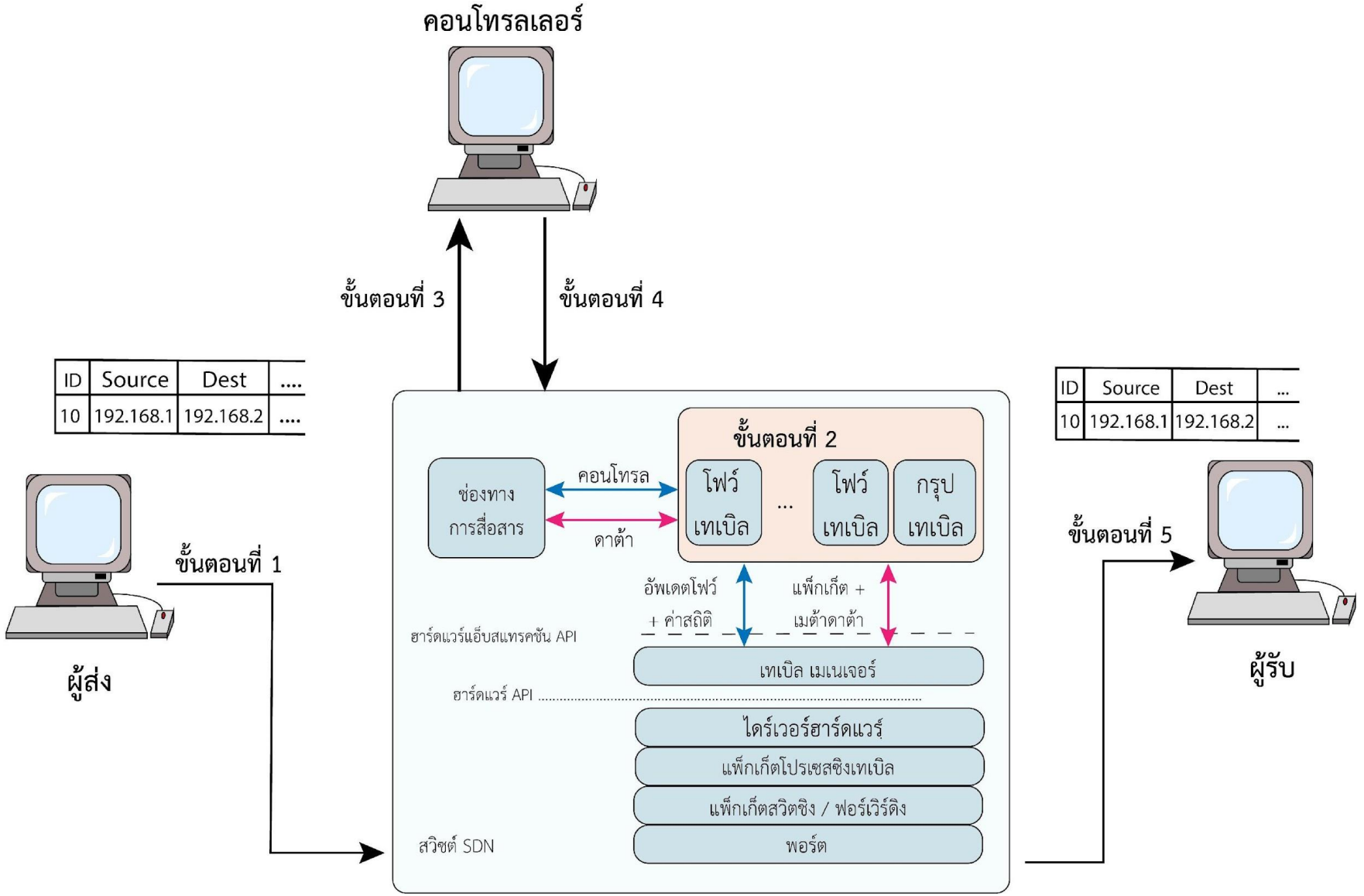
SDN Controller Architecture



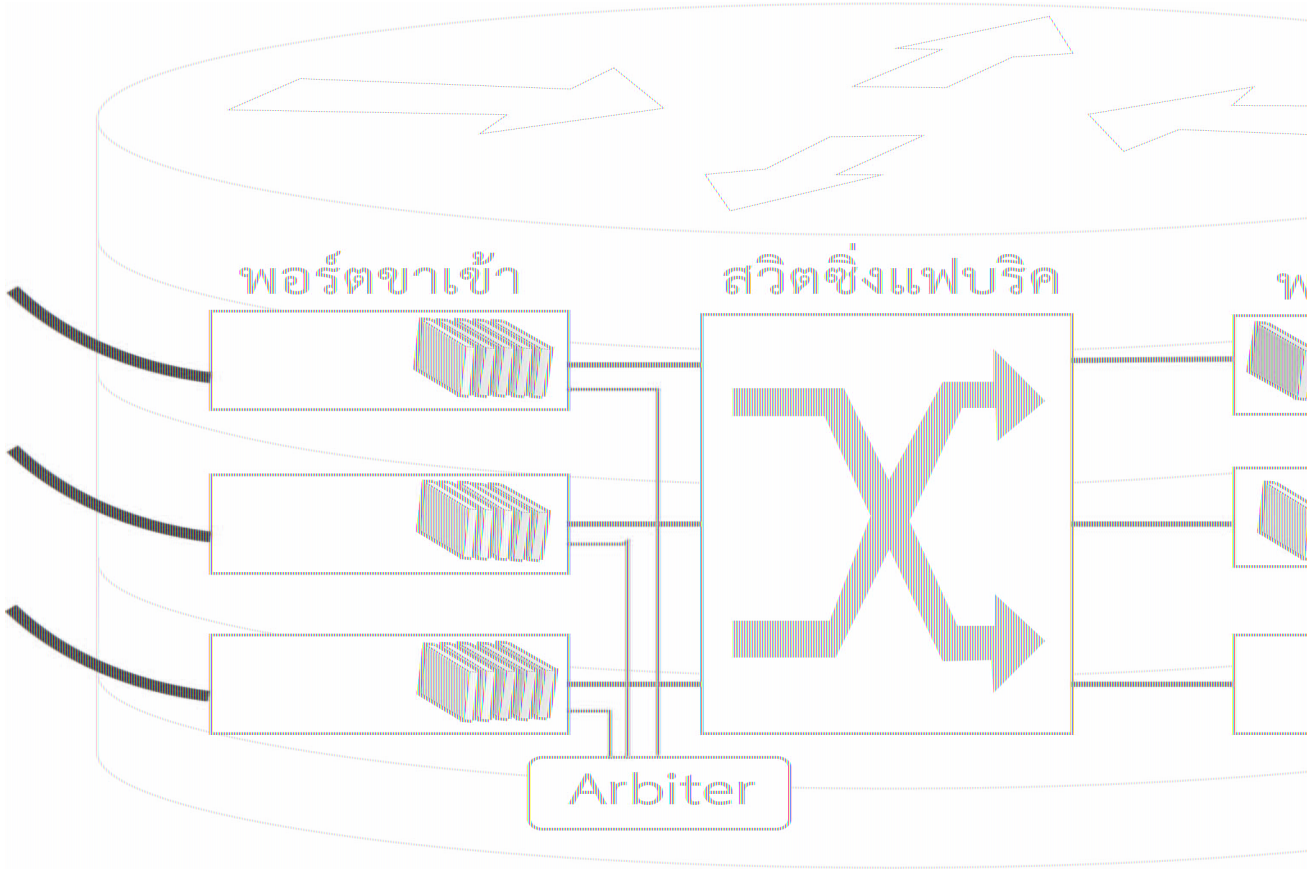
Eastbound /Westbound



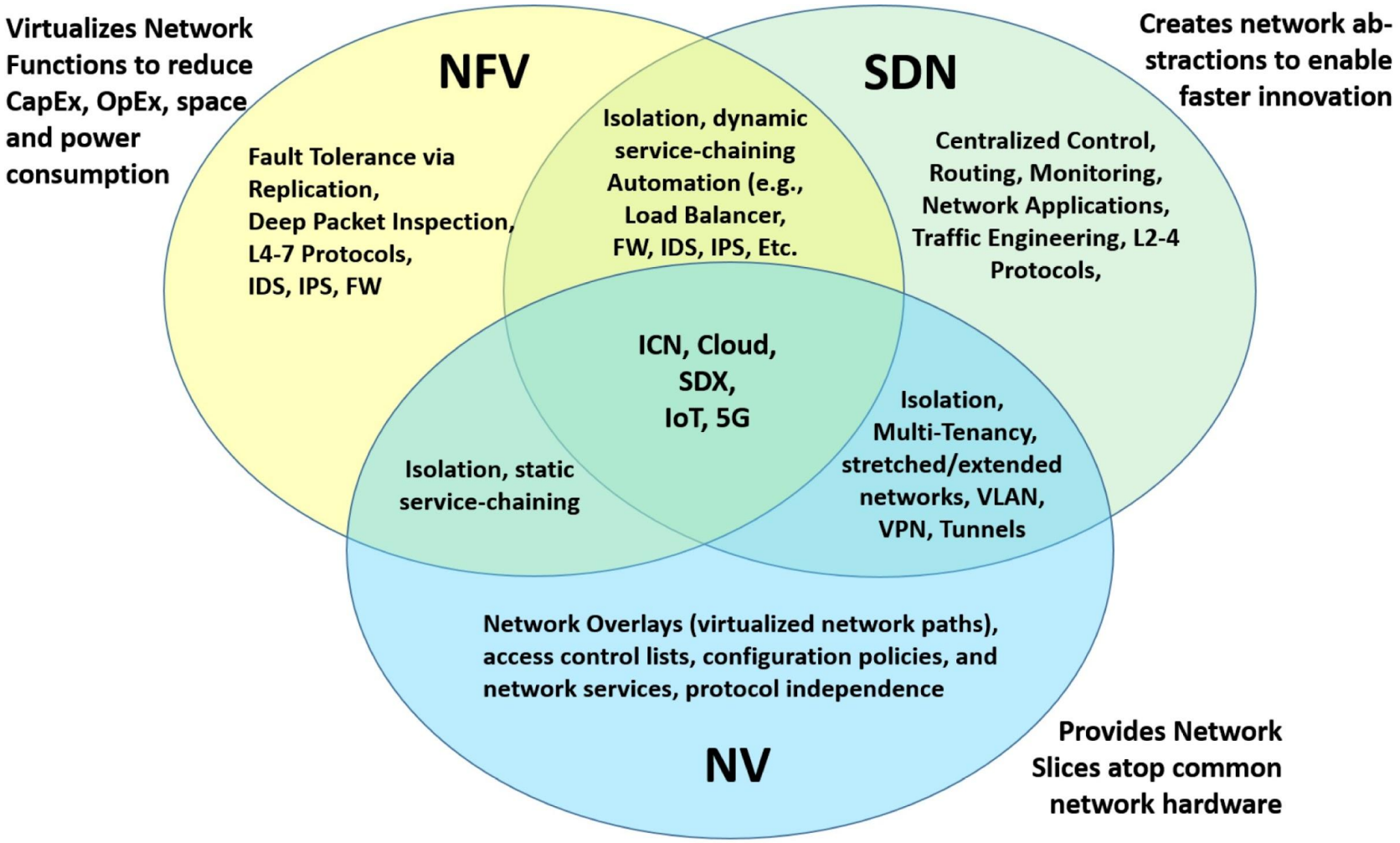
SDN Operations



Fabric Switch



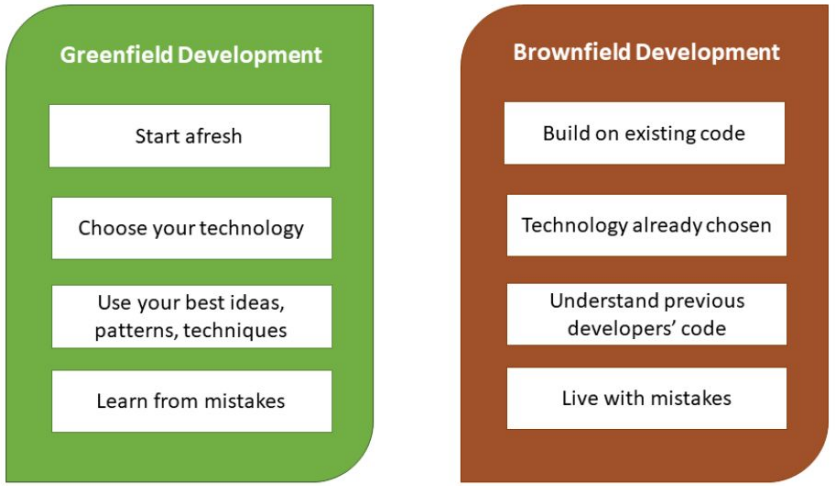
SDN/NFV Relationship



Example of SDN Controller

Open Network Operating System (ONOS)

- ONOS is designed to be distributed, stable and scalable with a focus on Service Provider networks. The Open Network Operating System is the only SDN controller platform that supports the transition from legacy “brown field” networks to SDN “green field” networks. This enables exciting new capabilities, and disruptive deployment and operational cost points for network operators.
- **Score:** 92.0%
- **Best For:** Communication Service Providers, SD-WAN deployments
- **Architecture:** Distributed, three-tier
- **Key Strength:** Extensive API support and production-ready scalability
- **Programming Language:** Java
- **Community:** Linux Foundation Networking



OpenDayLight (ODL)

- OpenDaylight is a modular open platform for customising and automating networks of any size and scale. The OpenDaylight Project arose out of the SDN movement, with a clear focus on network programmability. It was designed from the outset as a foundation for commercial solutions that address a variety of use cases in existing network environments.
- **Score:** 84.5%
- **Best For:** Cloud integration, SD-LAN, telco environments with OpenStack/ONAP
- **Architecture:** Modular, three-layer with OSGi
- **Key Strength:** Largest community support, extensive integration options
- **Programming Language:** Java
- **Community:** Linux Foundation Networking (largest community)

OpenKilda

- OpenKilda is a Telstra developed OpenFlow based SDN controller currently being used in production to control the large Pacnet infrastructure. It has been shown to be successful in a distributed production environment. Designed to solve the problem of implementing a distributed SDN control plane with a network that spans the Globe, OpenKilda solves the problem of latency while providing a scalable SDN control & data-plane and end-to-end flow telemetry.
- **Score:** 71.5%
- **Best For:** Global distributed networks, production telemetry at scale
- **Architecture:** Distributed, decentralized with Apache Storm
- **Key Strength:** End-to-end flow telemetry, proven at global scale
- **Programming Language:** Java
- **Community:** Small but active (Telstra-developed)

Ryu

- Ryu is a very different proposition to the other options being put forward. Although boasting a core set of programs that are run as a ‘platform’, Ryu is better thought of as a toolbox, with which SDN controller functionality can be built. Ryu is a component-based software defined networking framework that provides software components with well defined API that make it easy for developers to create new network management and control applications.
- **Score:** 73.2%
- **Best For:** Academic research, custom SDN development, OpenStack integration
- **Architecture:** Component-based framework (toolbox approach)
- **Key Strength:** Complete flexibility for custom solutions
- **Programming Language:** Python
- **Community:** Active development community

Faucet

- Built on top of Ryu, Faucet is a lightweight SDN Controller adding a critical northbound function for operations teams. Faucet is a compact open source OpenFlow controller, which enables network operators to run their networks the same way they do server clusters. Faucet moves network control functions (like routing protocols, neighbor discovery, and switching algorithms) to vendor independent server-based software, where those functions are easy to manage, test, and extend with modern systems managem
- **Score:** 69.3%
- **Best For:** CI/CD pipelines, lightweight deployments, container environments
- **Architecture:** Lightweight, distributed with YAML configuration
- **Key Strength:** Configuration-as-code approach, simple deployment
- **Programming Language:** Python
- **Community:** Active and well-supported
- ent best practices and tools.

Cisco ACI (Application Centric Infrastructure)

- A commercial SDN solution developed by Cisco
  - Policy-based networking with a centralized controller.
  - Robust security features.
  - Tight integration with Cisco hardware.
- Suitable for enterprise networks with a strong Cisco hardware presence and a need for policy-based networking

AI: Claude 4.5

Protocol Support

Controller	OpenFlow	NETCONF	BGP	PCEP	P4
ONOS	1.0-1.5	Yes	Yes	Yes	Yes
ODL	1.0-1.3	Yes	Yes	Yes	Limited
Ryu	1.0-1.5	No	Limited	No	No
Floodlight	1.0-1.3	No	No	No	No
APIC	Proprietary	Yes	Yes	No	No

AI: Claude 4.5

Architecture

Controller	Clustering	HA	Scalability	REST API
ONOS	Native	Active-Active	Excellent	Yes
ODL	Yes (Akka)	Active-Standby	Good	Yes
Ryu	No	No	Limited	Yes
Floodlight	Limited	Master-Slave	Moderate	Yes
APIC	Yes	Active-Active	Excellent	Yes

AI: Claude 4.5

Performance Metrics

Controller	Flow Setup (flows/sec)	Topology Discovery	Latency
ONOS	~100K	Fast (distributed)	~10ms
ODL	~50K	Moderate	~15ms
Ryu	~10K	Basic	~20ms
Floodlight	~30K	Good	~15ms
APIC	Proprietary	Very Fast	<10ms

AI: Claude 4.5

Application Ecosystem

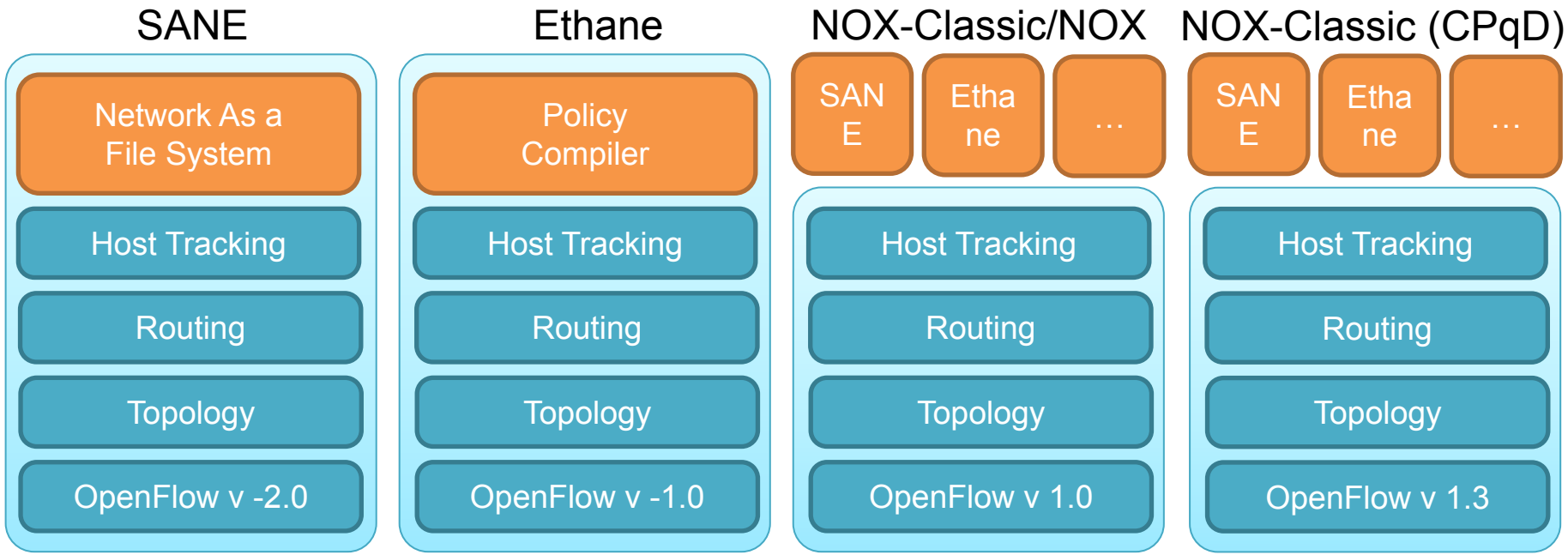
Controller	Built-in Apps	Intent-based	GUI	Programming Interface
ONOS	30+	Yes (strong)	Web UI	Java, REST
ODL	50+ modules	Limited	DLUX	Java, REST, Python
Ryu	10+	No	Basic	Python
Floodlight	15+	No	Web UI	Java, REST
APIC	Cisco apps	Yes (policy)	Advanced GUI	Python, REST

NOX & POX

History of NOX

History

- Started from SANE project and applied to Ethane in 2006
- Ethane project was announced via SIGCOMM in 2007
- Announced the OpenFlow spec 0.1.0 in Nov. 2007, spec 0.8.0 in May, 2008
- NOX was initially developed by Nicira side-by-side with OpenFlow, and open sourced as the first OpenFlow controller in 2008
- NOX was further developed by CPqD to support OpenFlow 1.3 in Nov. 2012



NOX Classic vs NOX (1/2)

NOX Versions

Version Name	Branch Name	Version No.	Release Date	OF Spec	GUI
NOX Classic	Zaku	0.9.0	2010-09-15	v1.0	Support
	Destiny	0.9.1	2012-07-20	v1.0	Support
	CPqD ver.	N/A	2012-11-10	v1.3	Not support
(new) NOX	Verity	0.9.2	2012-05-11	v1.0	Not support

NOX-Classic

- Original NOX (now, officially deprecated)
- C++ based SDN controller, applications can be developed using Python
  - Use SWIG to integrate Python with C++

```
guni@ottawa:~/itce710e/nox-classic/src/nox/coreapps$ ls
coretests   __init__.py  messenger     simple_c_py_app  switch
examples    Makefile.am  pyrt          simple_py_app    testharness
hub         Makefile.in  simple_c_app   snmp
```

Sample apps written in C++

Sample apps written in Python

Sample apps written in Python and C++ using SWIG lib

NOX Classic vs NOX (2/2)

NOX

- New NOX, separated from NOX-classic branch
- Only support C++ for application development
- Fewer default applications than NOX-classic
- Enhanced performance and better source readability
  - Written on top of Boost library
- No graphic user interface


```
    void configure(const Configuration*);

    void install();

    Disposition handle(const Event&);

private:
    typedef hash_set<Mac_source, Hash_mac_s
    Source_table sources;
```

NOX-Classic Code Snippet

```
struct datapath_hasher {
    static size_t hash(const datapathid& o) {
        return boost::hash_value(o.as_host());
    }
    static bool equal(const datapathid& o1, con
    {
        return o1 == o2;
    }
};
typedef boost::unordered_map<ethernetaddr, int>
typedef tbb::concurrent_hash_map<datapathid, ma
```

NOX Code Snippet

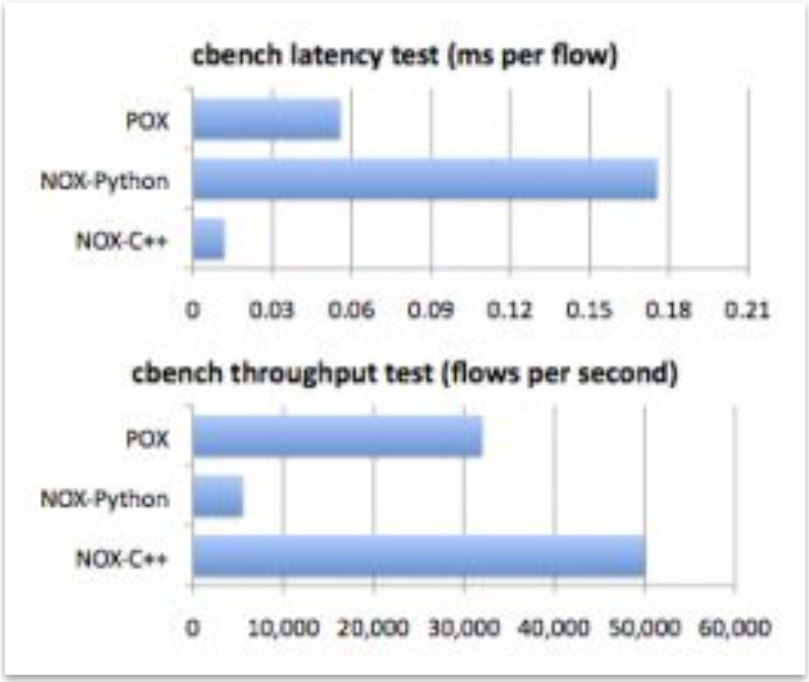
POX Overview

POX Versions

Controller Name	Branch Name	Version No.	Release Date	OF Spec	GUI
POX	Angler	0.0.0	2012 Fall	v1.0	Support
	Betta	0.1.0	2013 Spring	v1.0	Support
	Carp	0.2.0	2013 Fall	v1.0	Support
	Dart	0.3.0	2014 Summer	v1.0	Support

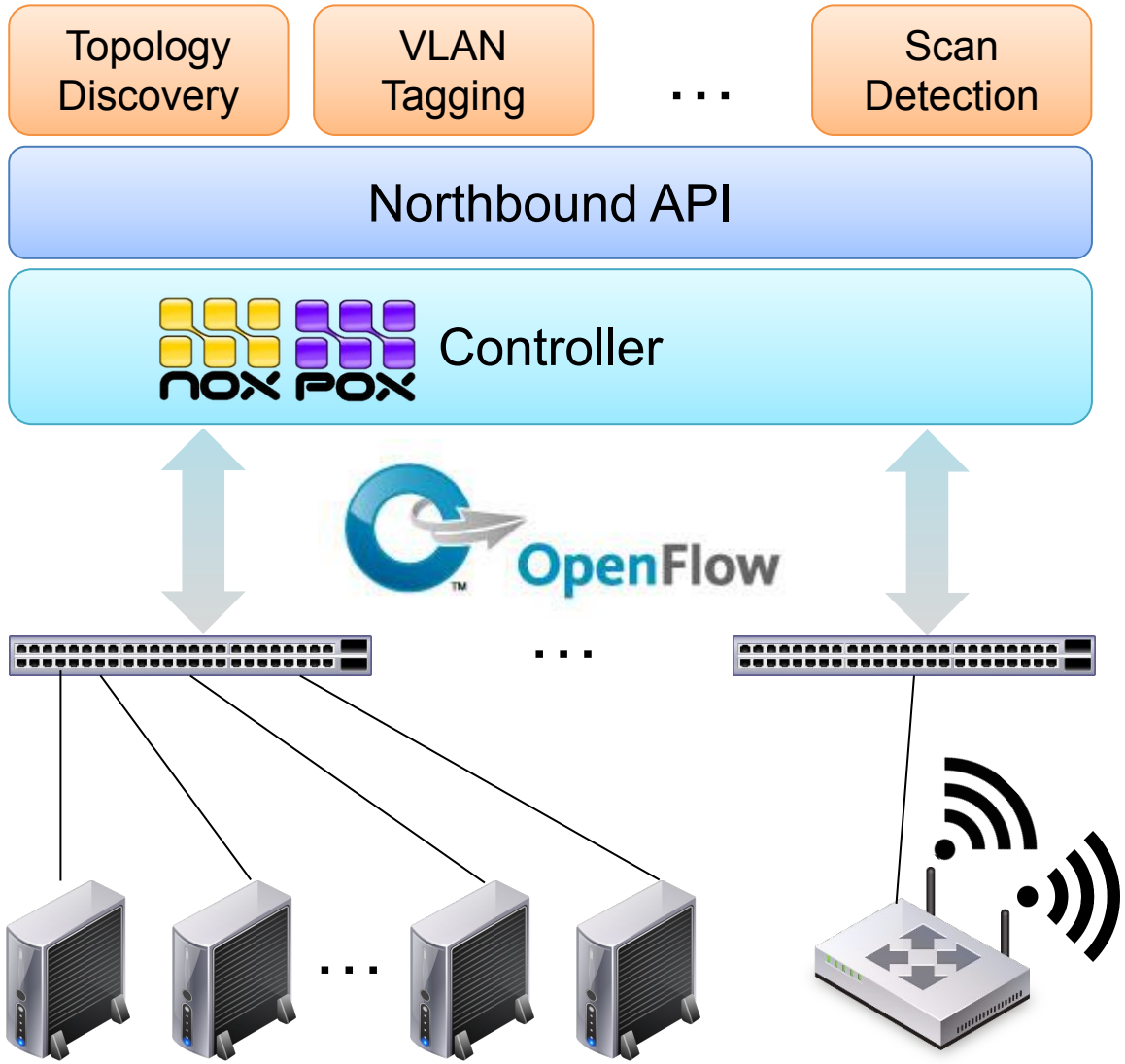
POX

- NOX's younger sibling implemented using Python
- For the rapid development and prototyping of network control software
- Support all OS (e.g., Linux, Mac, Windows)
- Can bundle with install-free PyPy runtime
- Performs well compared to NOX-classic Python
  - Note that NOX-classic does not support PyPy runtime
- Used for research and education purpose



NOX/POX Overview (1/4)

Components



Network Application Services
- New functions as software services

Northbound API
- Provide interface to network applications
- Not yet standardized

NOX Controller – Network OS
- Provide system-wide abstractions
- Turn networking into a software problem

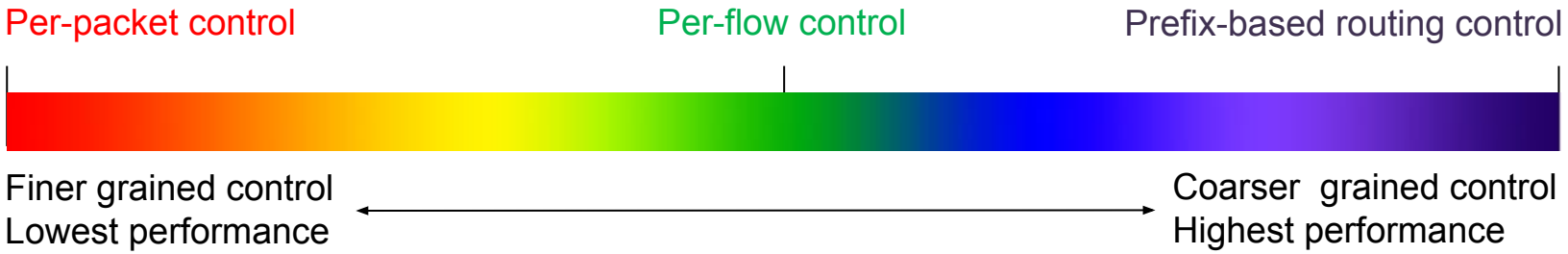
Southbound API
- Standardized OpenFlow protocol
- Controller, switch interoperability

OpenFlow Enabled Switches
- New functions as software services

NOX/POX Overview (2/4)

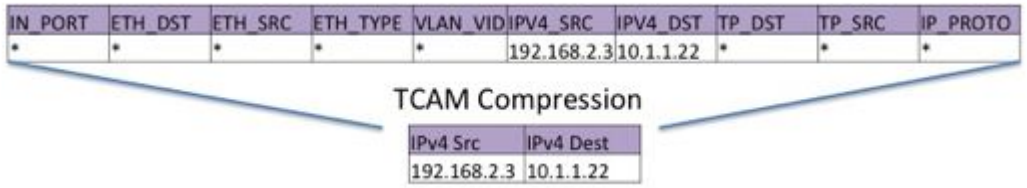
Granularity

- Choosing granularity involves trading off scalability against flexibility
- Observation: provides adequate information and changes slowly
  - Includes switch-level topology, does not include the current state of network traffic
- Control: scale to large networks, while providing flexible control



Switch Abstraction

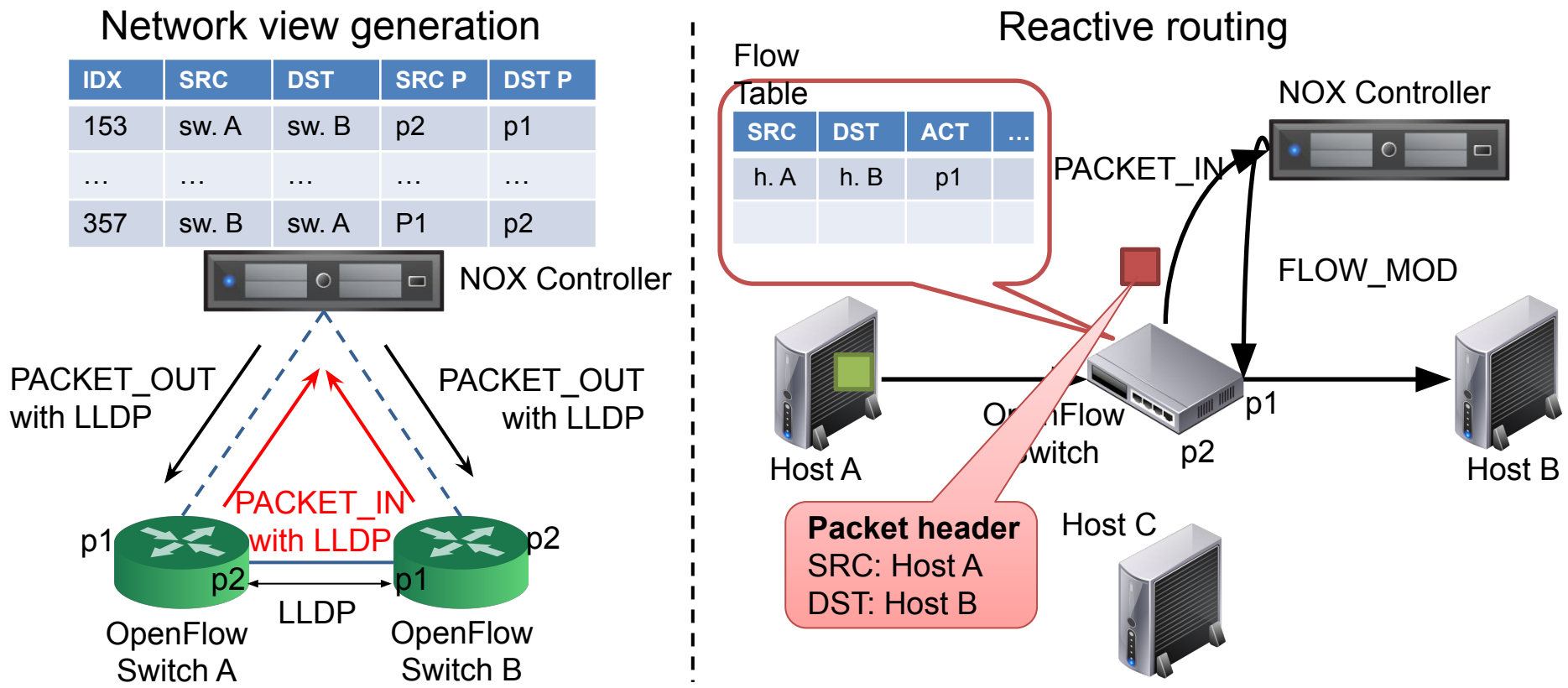
- Switch instructions should be independent of the particular HW and support the flow-level control granularity
- Solution: adopt OpenFlow switch abstraction



NOX/POX Overview (3/4)

Operation

- Observation: construct the network view
  - Use DNS, DHCP, LLDP and flow-initiation
- Control: determine whether to forward traffic, and to which route
  - Access control, routing and failover applications



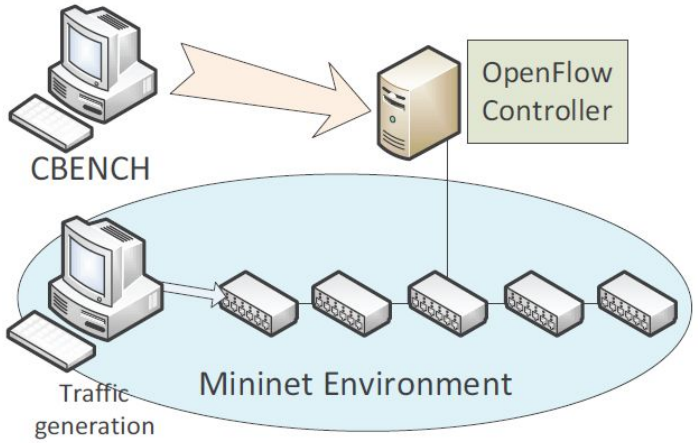
NOX/POX Overview (4/4)

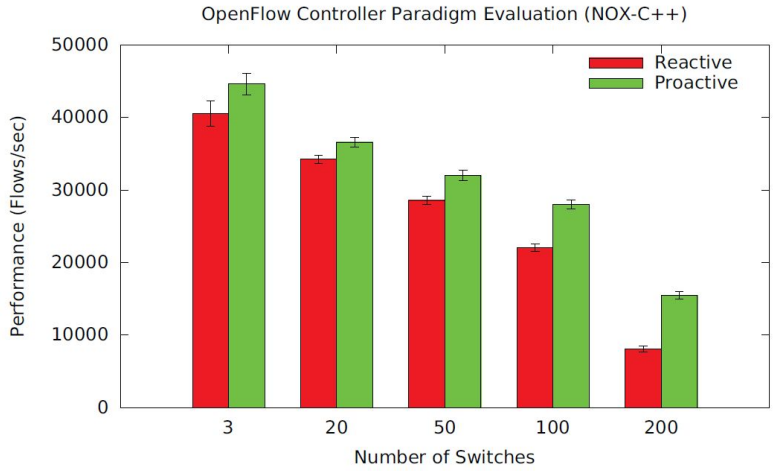
Scaling

- Packet level parallelization
  - Millions of arrival packets per second
  - Packets are handled by individual switches
- Flow level parallelization
  - Flow initialization rate is high but lower than packet arrival rate
  - Flow initialization is handled by individual controller
- Periodic synchronization
  - Network view changes slowly enough that can be maintained centrally

Public Release

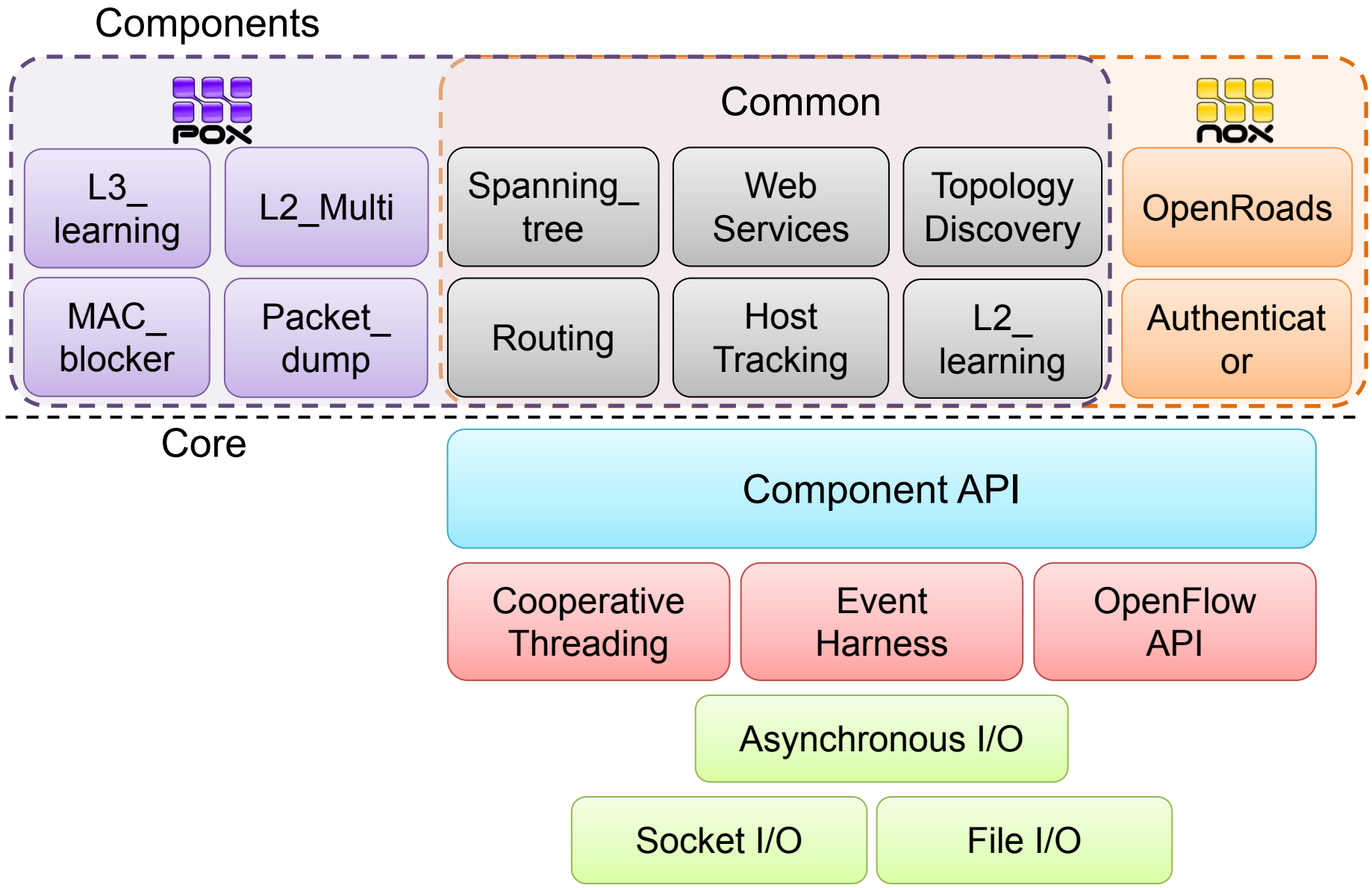
- A PoC of OpenFlow controller
- Follow GPL license
- Many network applications





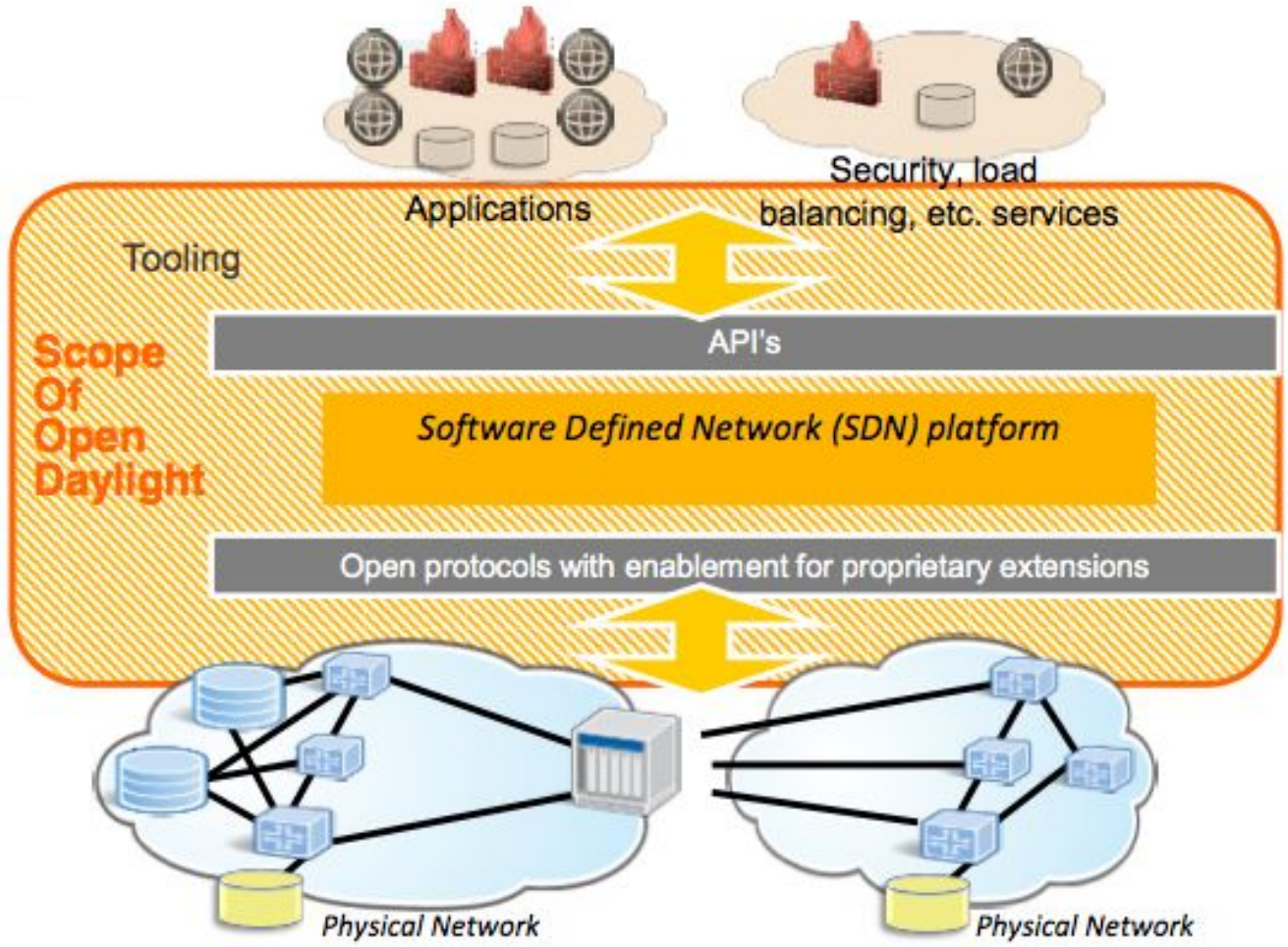
Source: Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive. International Conference on Advanced Information Networking and Applications

NOX/POX Architecture



ODL: Open Daylight

OpenDaylight Scope



OpenDaylight Scope and Projects

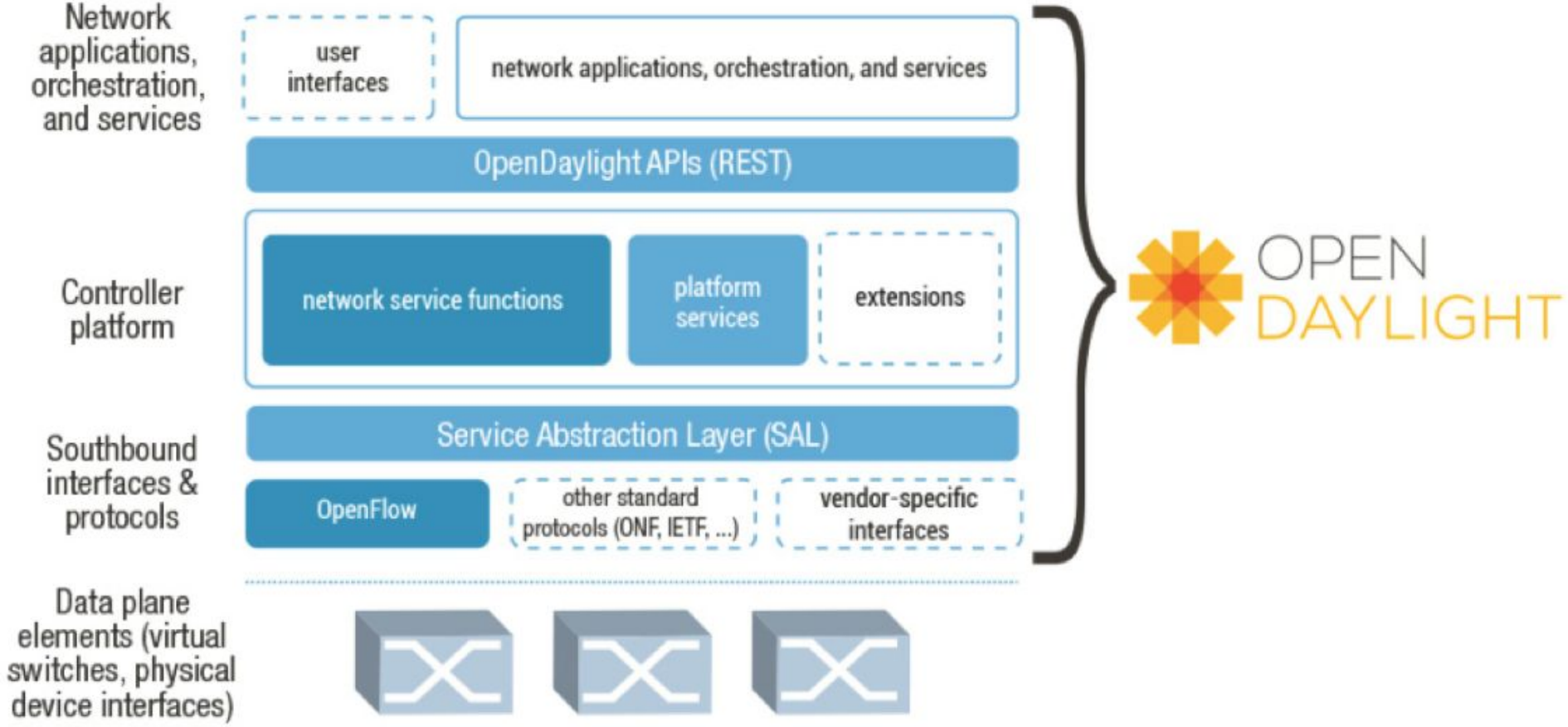
OpenDaylight Projects

- 15 projects currently in bootstrap or incubation

Project	Description	Originator
Controller	Modular, extensible, scalable, and multi-protocol SDN controller based on OSGi	Cisco
YANG Tools	Java-based NETCONF and YANG tooling for OpenDaylight projects	Cisco
OpenFlow Protocol Library	OF 1.3 protocol library implementation	Pantheon
OpenFlow Plugin	Integration of OpenFlow protocol library in controller SAL	Ericsson, IBM, Cisco
Defense4All	DDoS detection and mitigation framework	Radware
OVSDB	OVSDB configuration and management protocol support	Univ. of Kentucky
LISP Flow Mapping	LISP plugin, LISP mapping service	ConteXtream

OpenDaylight Scope and Projects

Project Framework



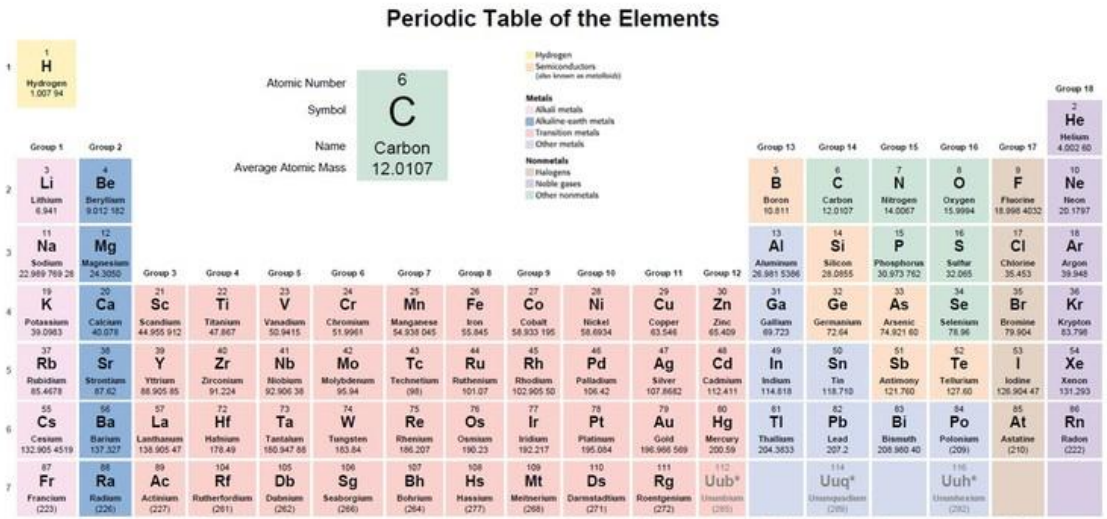
OpenDaylight Controller

Features of OpenDaylight Controller

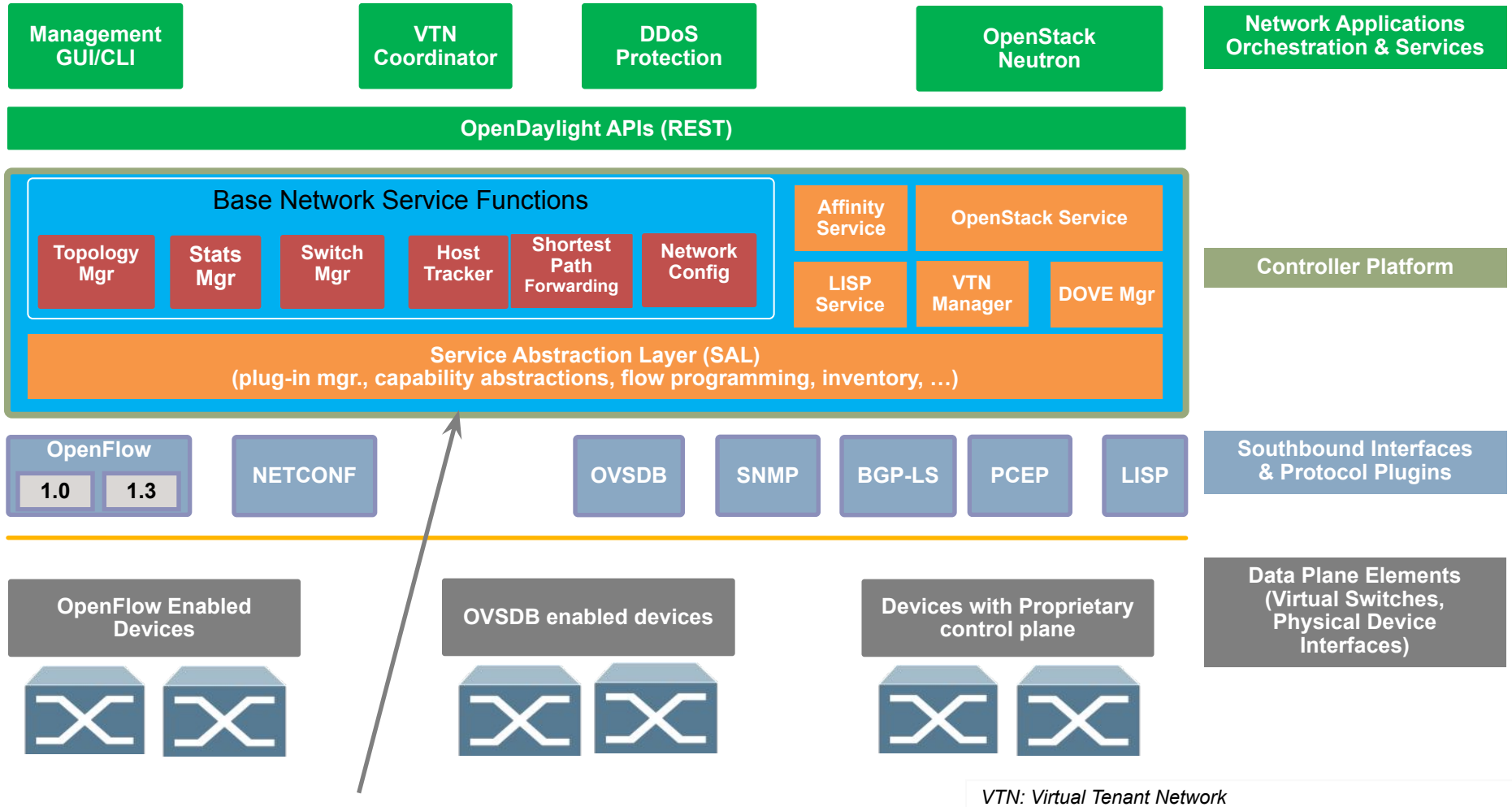
- Built using Java OSGi Framework – Equinox
  - Provides Modularity & Extensibility
  - Bundle Life-cycle management
  - In-Service-Software Upgrade (ISSU) & multi-version support
- Service Abstraction Layer (SAL)
  - Provides Multi-Protocol Southbound support
  - Abstracts/hides southbound protocol specifics from the applications
- High availability & horizontal scaling using clustering

Releases

- Hydrogen
  - Initial release, support OpenFlow 1.0 and 1.3
- Helium
  - Current stable release, plan to support OpenFlow 1.5
- Lithium
  - The next release (6/25/2015 □ SR1 8/13 □ 9/24)
  - New project □ IoT Data Management (IoTDM)



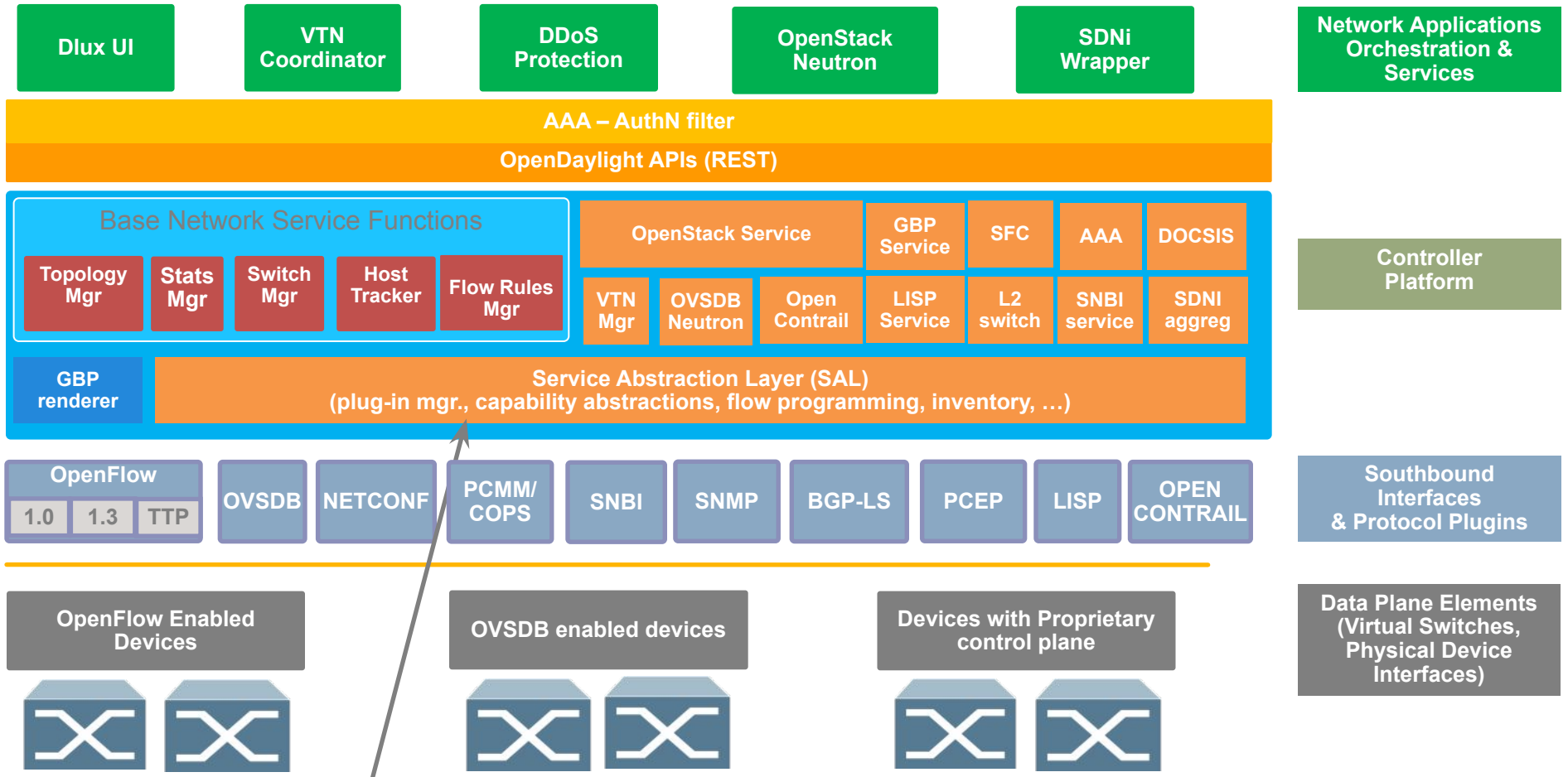
Hydrogen Architecture



Main difference from other OpenFlow-centric controller platforms

VTN: Virtual Tenant Network
DOVE: Distributed Overlay Virtual Ethernet
DDoS: Distributed Denial Of Service
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase Protocol
BGP: Border Gateway Protocol
PCEP: Path Computation Element Communication Protocol
SNMP: Simple Network Management Protocol

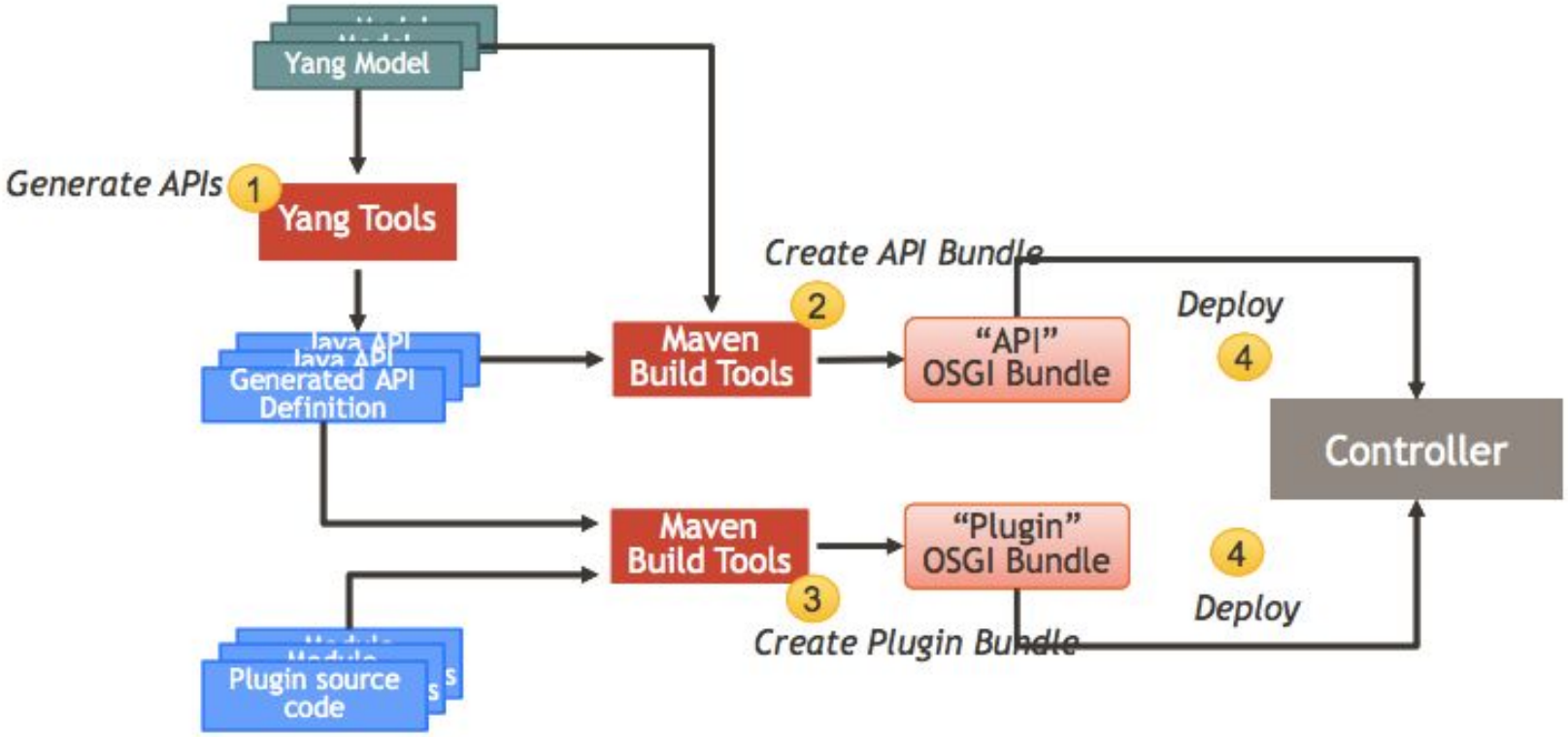
Helium Architecture



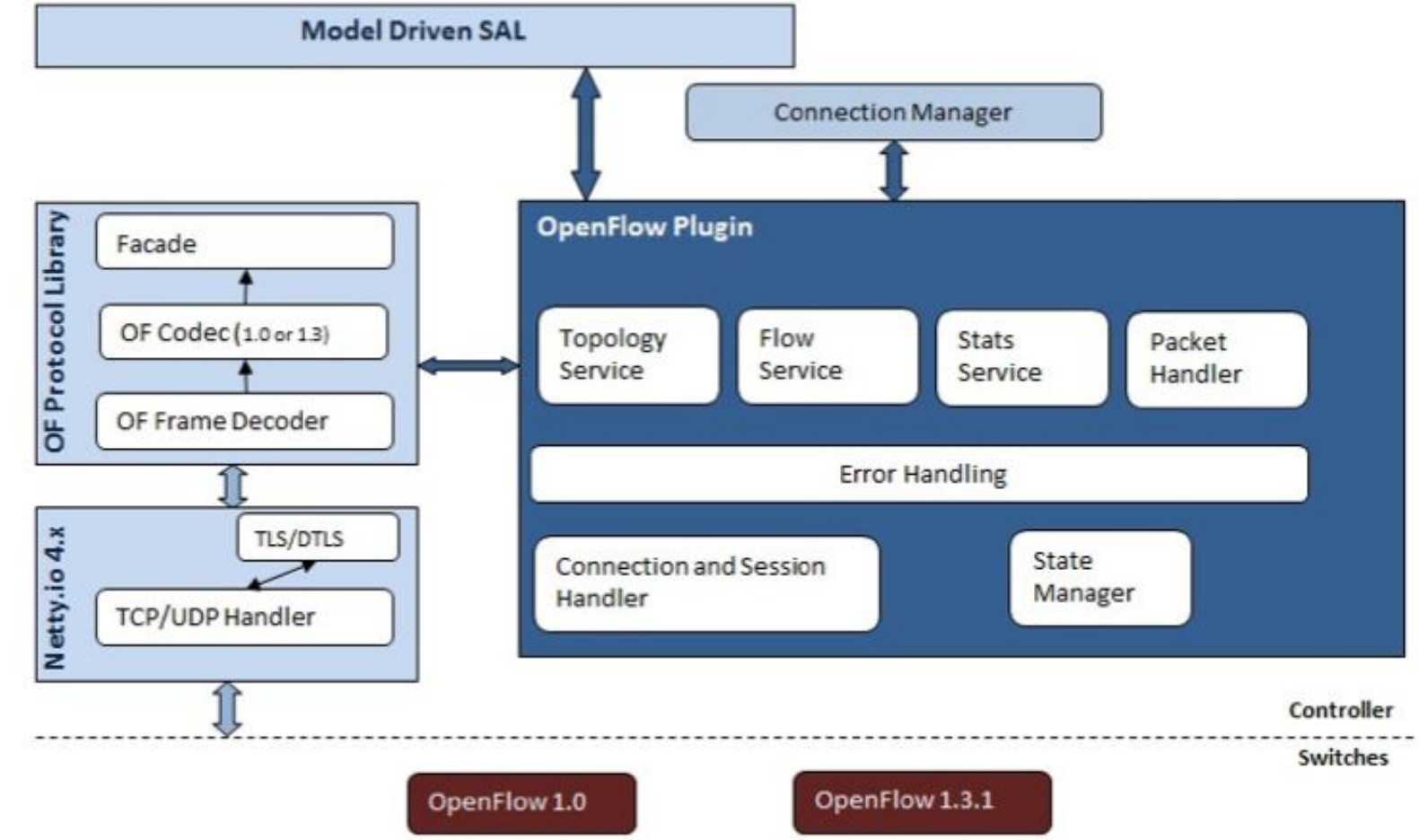
Main difference from other OpenFlow-centric controller platforms

VTN: Virtual Tenant Network
DDoS: Distributed Denial Of Service
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase Protocol
BGP: Border Gateway Protocol
PCEP: Path Computation Element Communication Protocol
PCMM: Packet Cable MultiMedia
SNMP: Simple Network Management Protocol

Plugin Build Process



Hydrogen OpenFlow Plugin Architecture

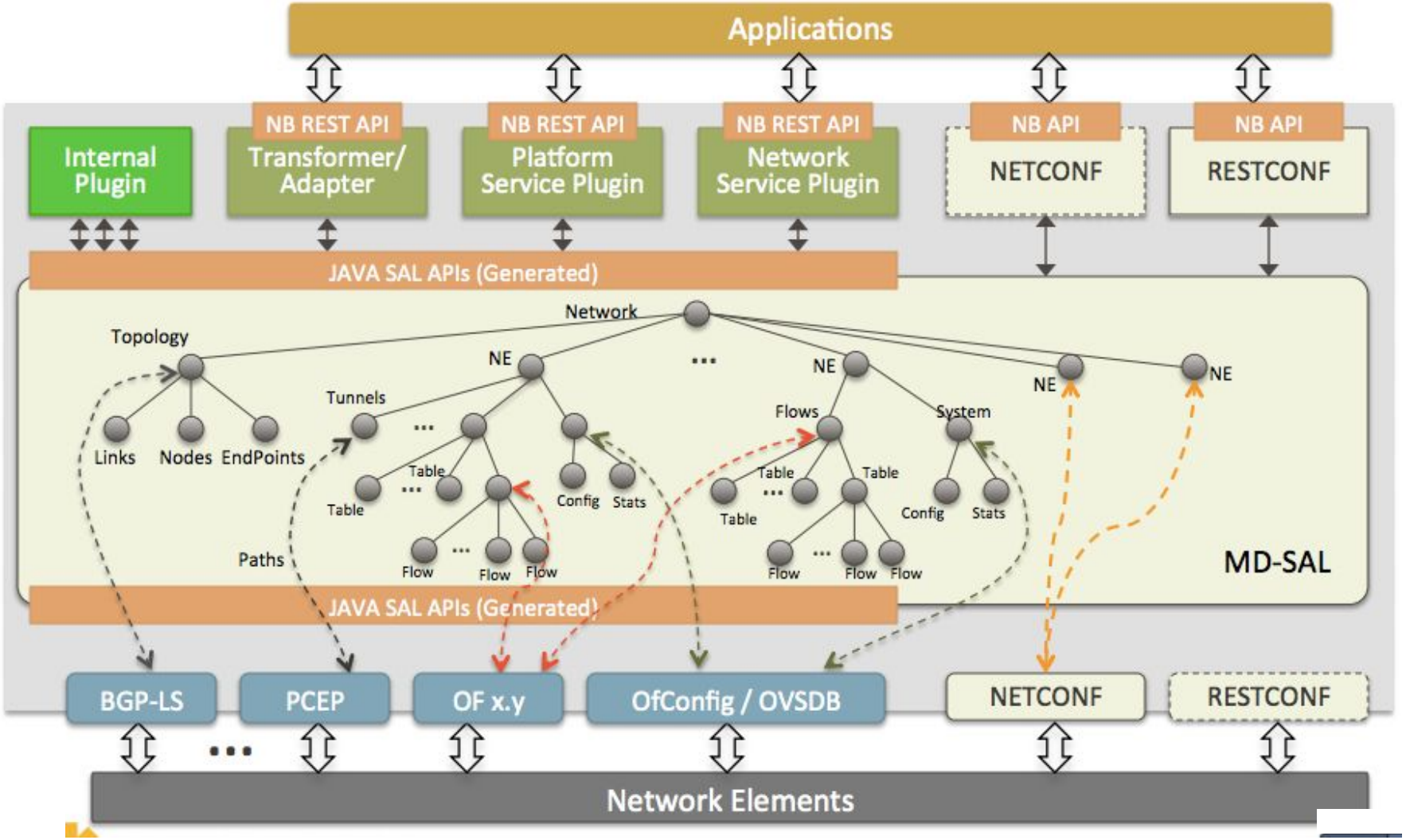


Model-Driven SAL (1/2)

OPEN DAYLIGHT

POSTECH 2020

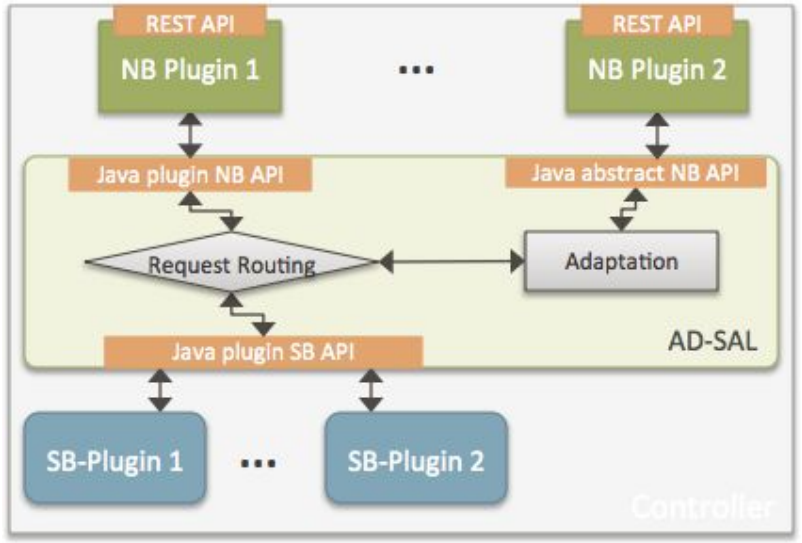
❖ Model-Driven Service Abstraction Layer (SAL)



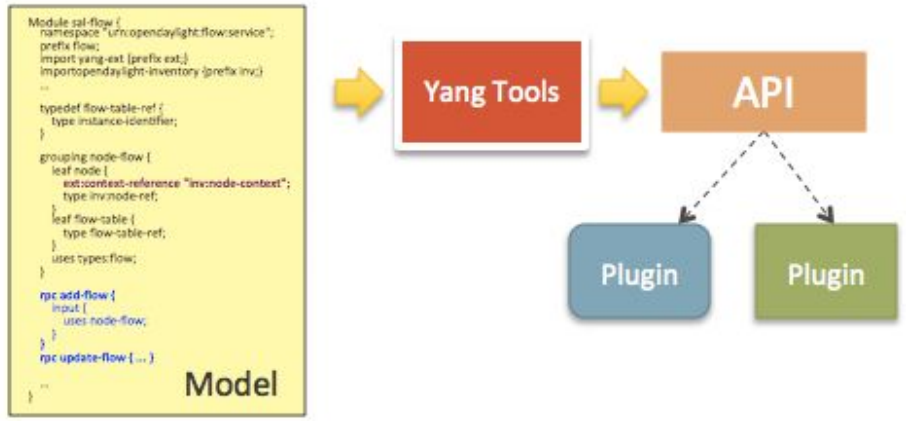
Model-Driven SAL (2/2)

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

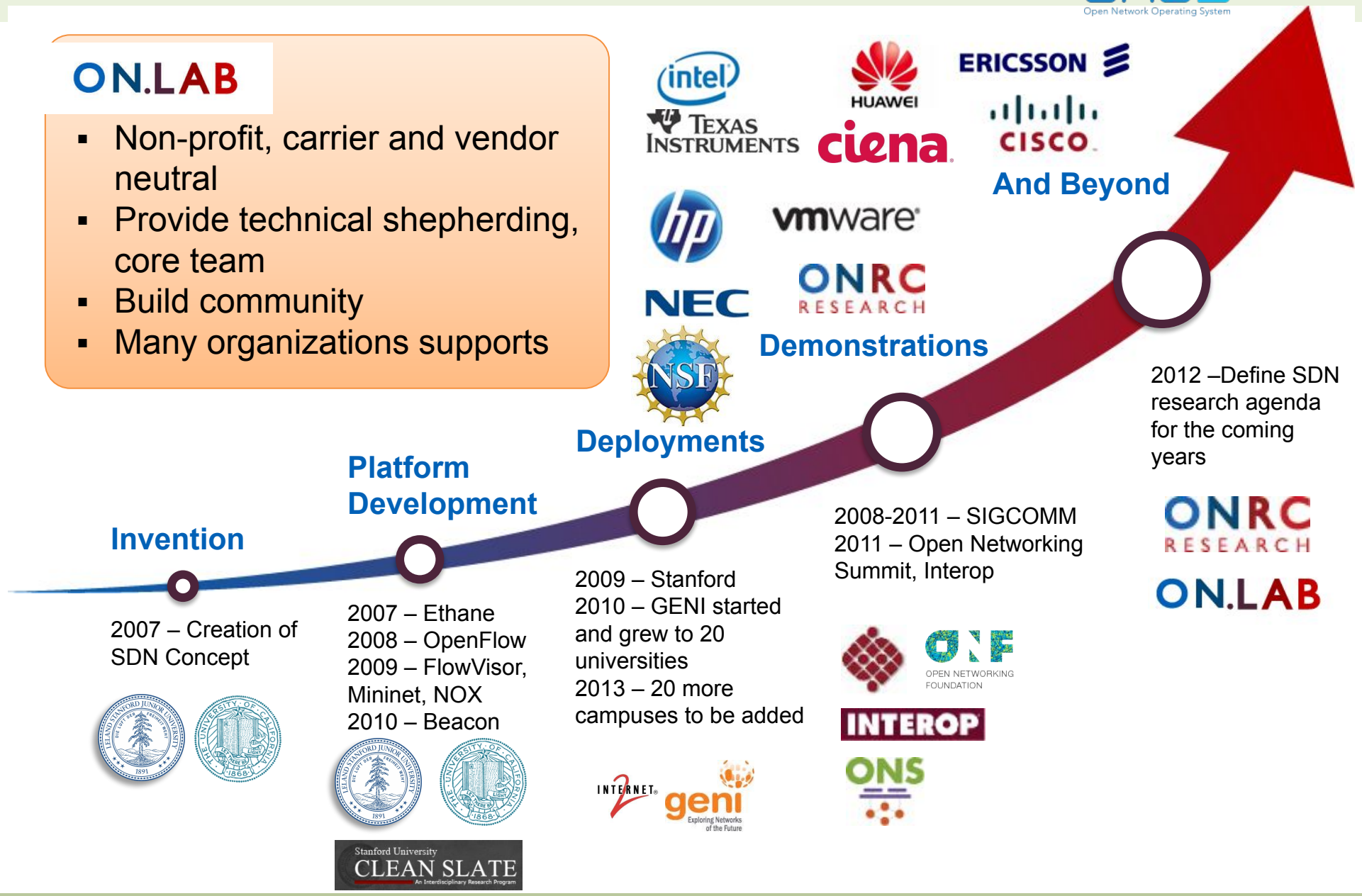
ONOS: Open Network Operating System

SDN Evolution and ON.LAB

ONOS Open Network Operating System

ON.LAB

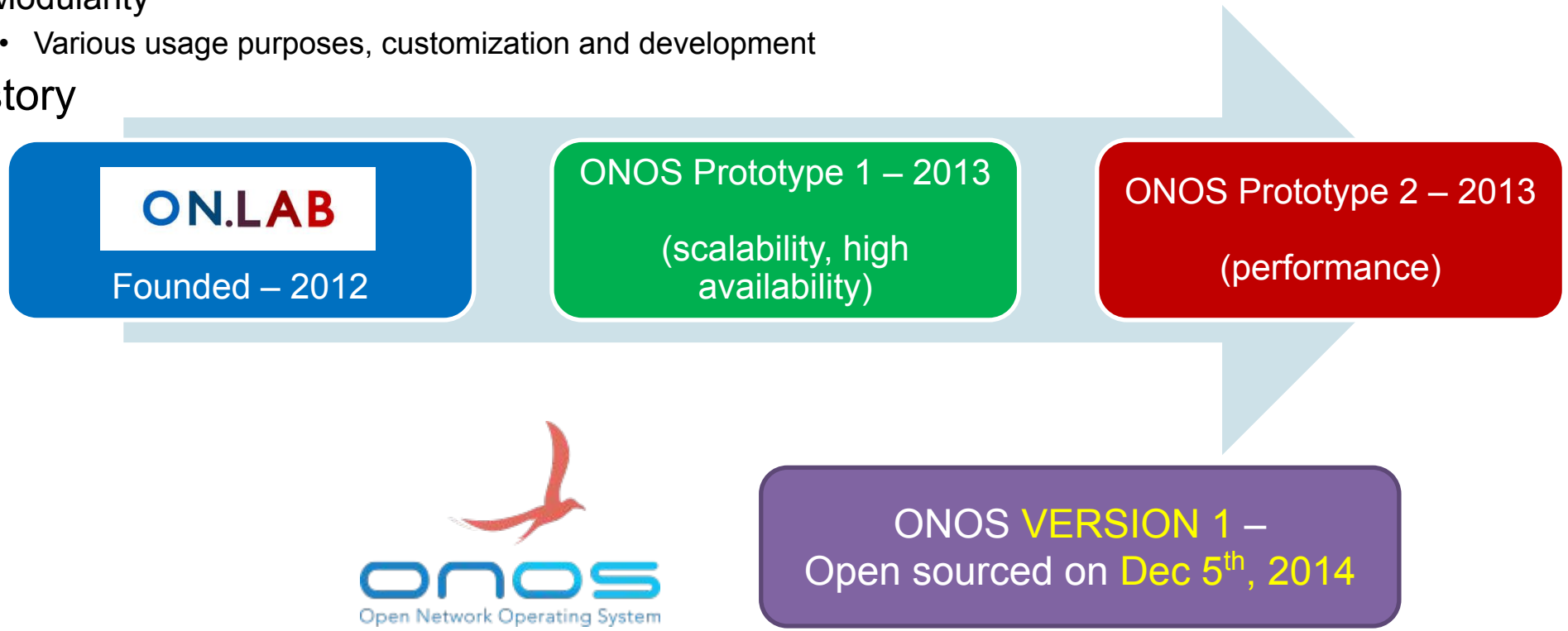
- Non-profit, carrier and vendor neutral
- Provide technical shepherding, core team
- Build community
- Many organizations supports



Introduction

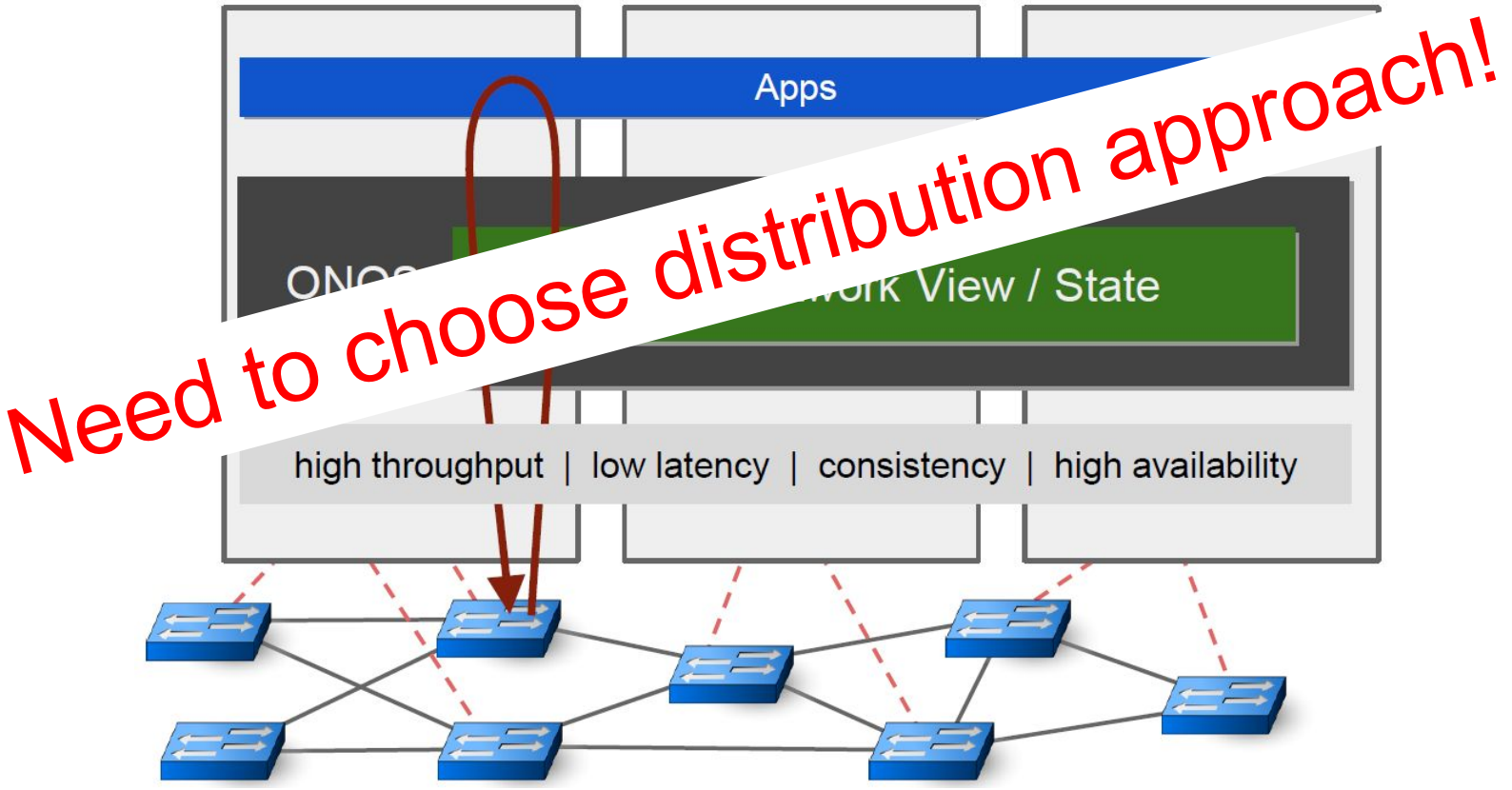
ONOS: Open Network Operating System

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



Key Performance Requirements

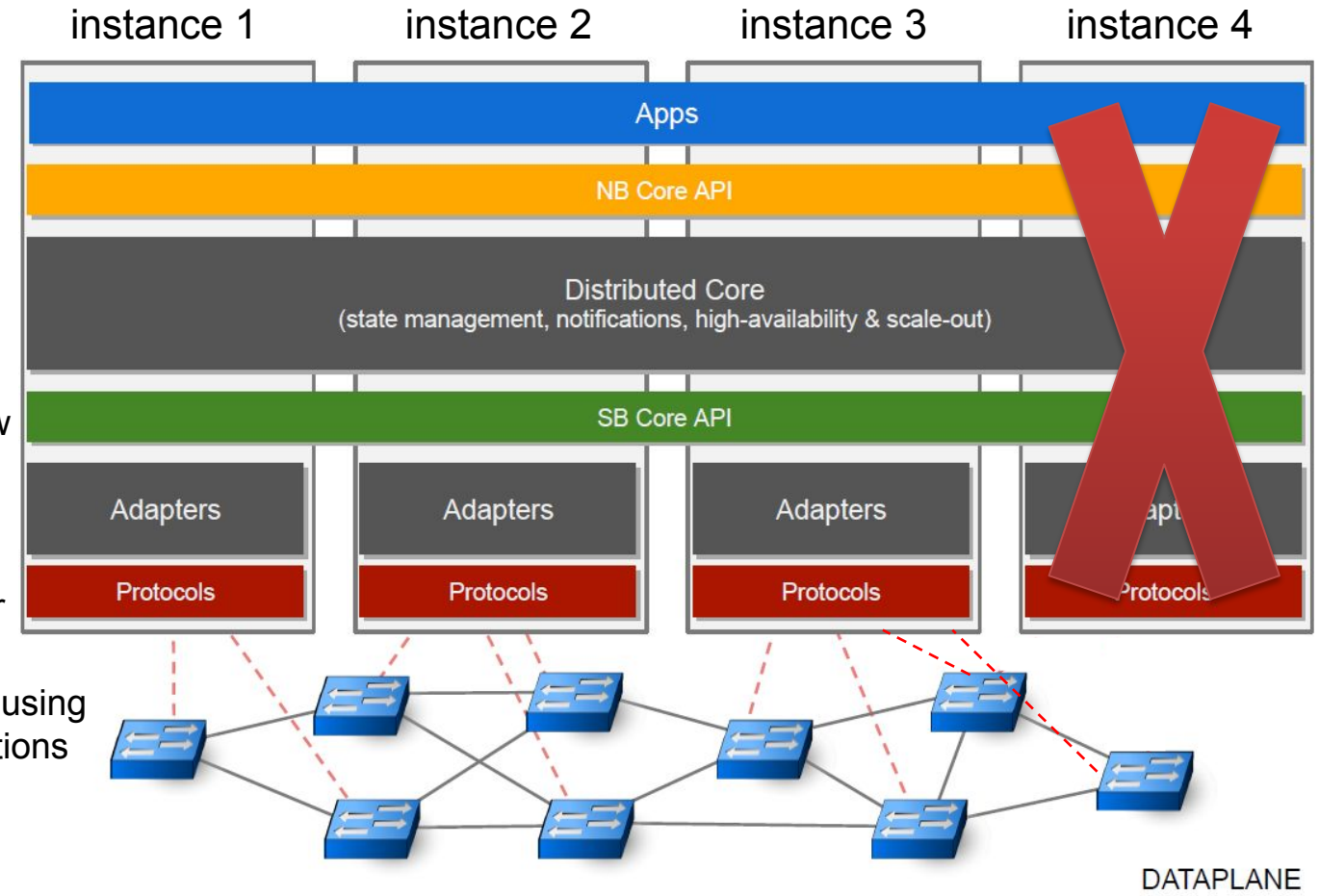
- **Requirements for Supporting Service Provider Networks**
  - High throughput
    - 500K ~ 1M paths setups / second, 3-6M network state operations / second
  - High volume
    - 500GB ~ 1TB of network state data



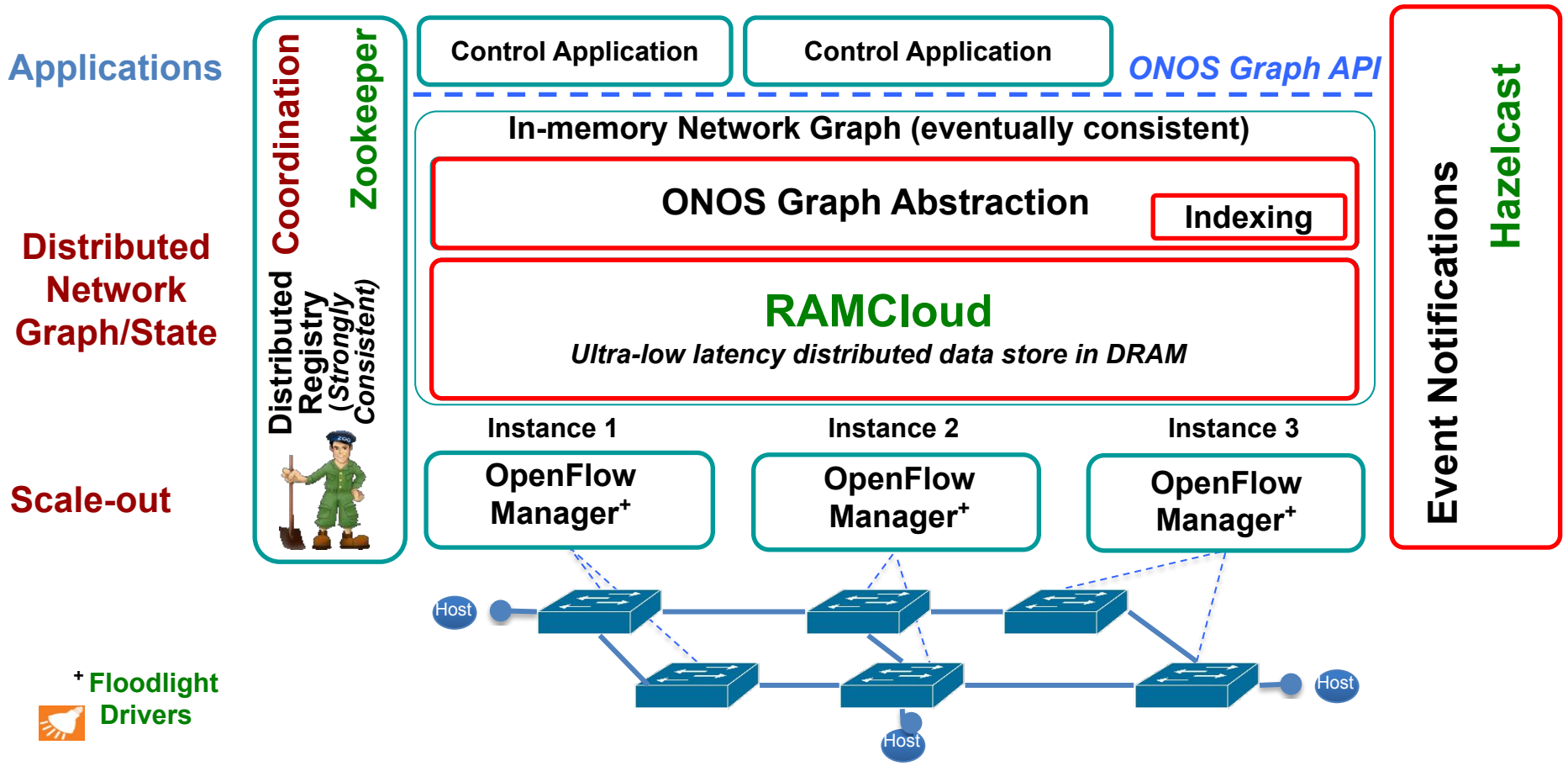
Distributed Architecture

- Six tiered architecture
- Each ONOS instance is equipped with the same software stack

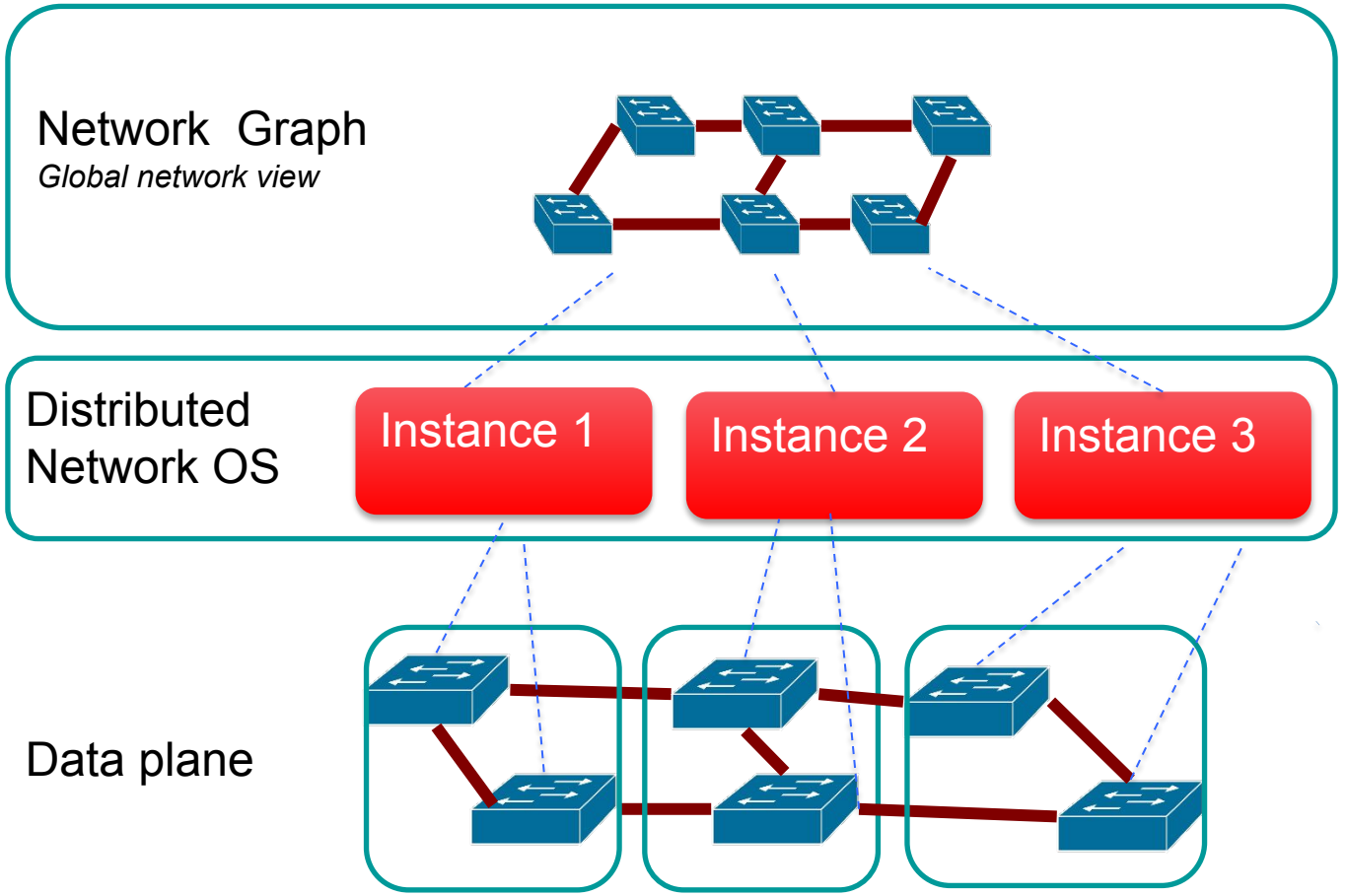
- Northbound Abstraction
  - Network graph
  - Application intents
- Core
  - Distributed
  - Protocol independent
- Southbound Abstraction
  - Generalized OpenFlow
  - Pluggable & extensive
- Adapters
  - Multiple southbound protocol enabling layer
- Protocols
  - Self-defined protocols using generalized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



ONOS Architecture (Prototype 2)



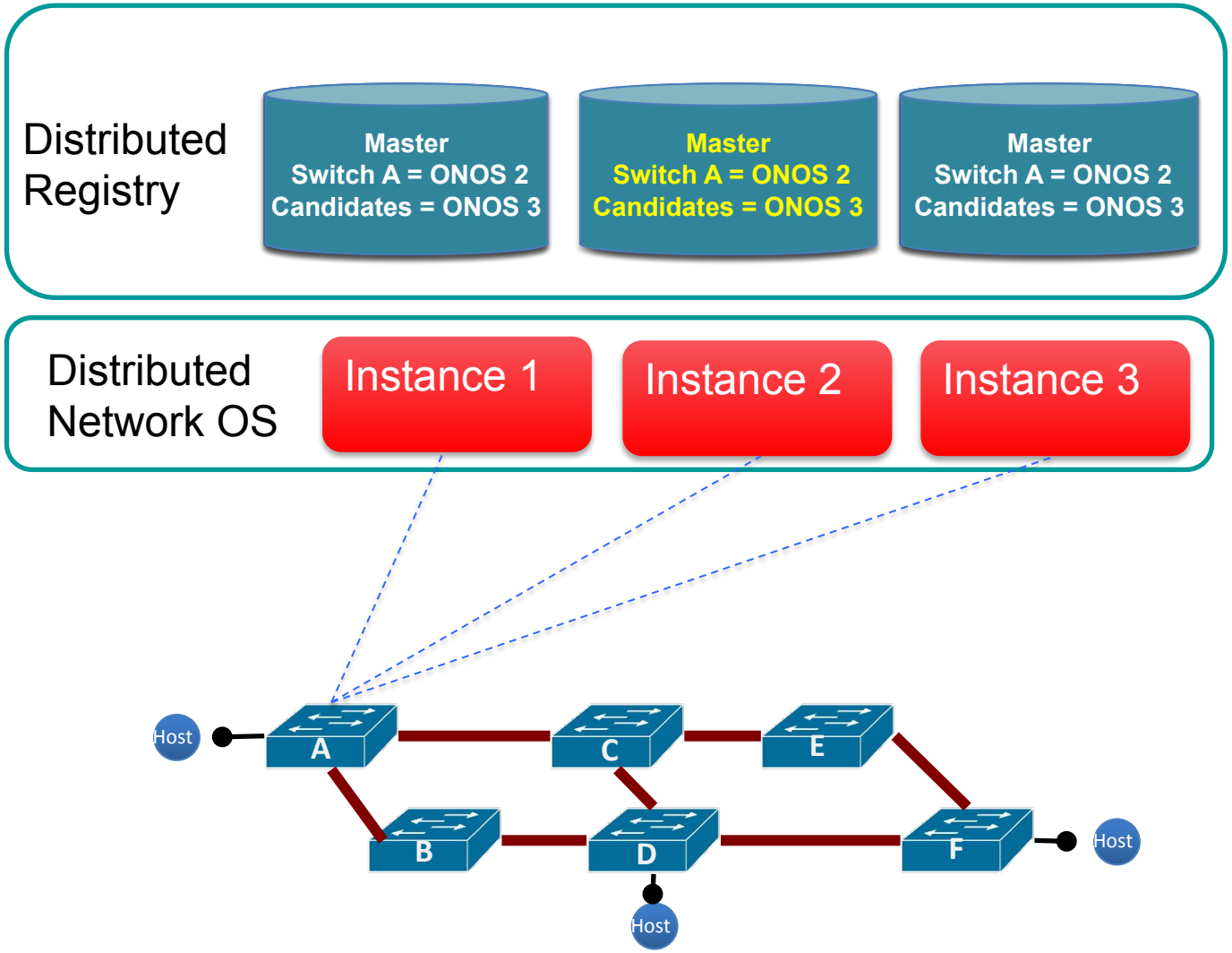
ONOS Scale-Out



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



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)

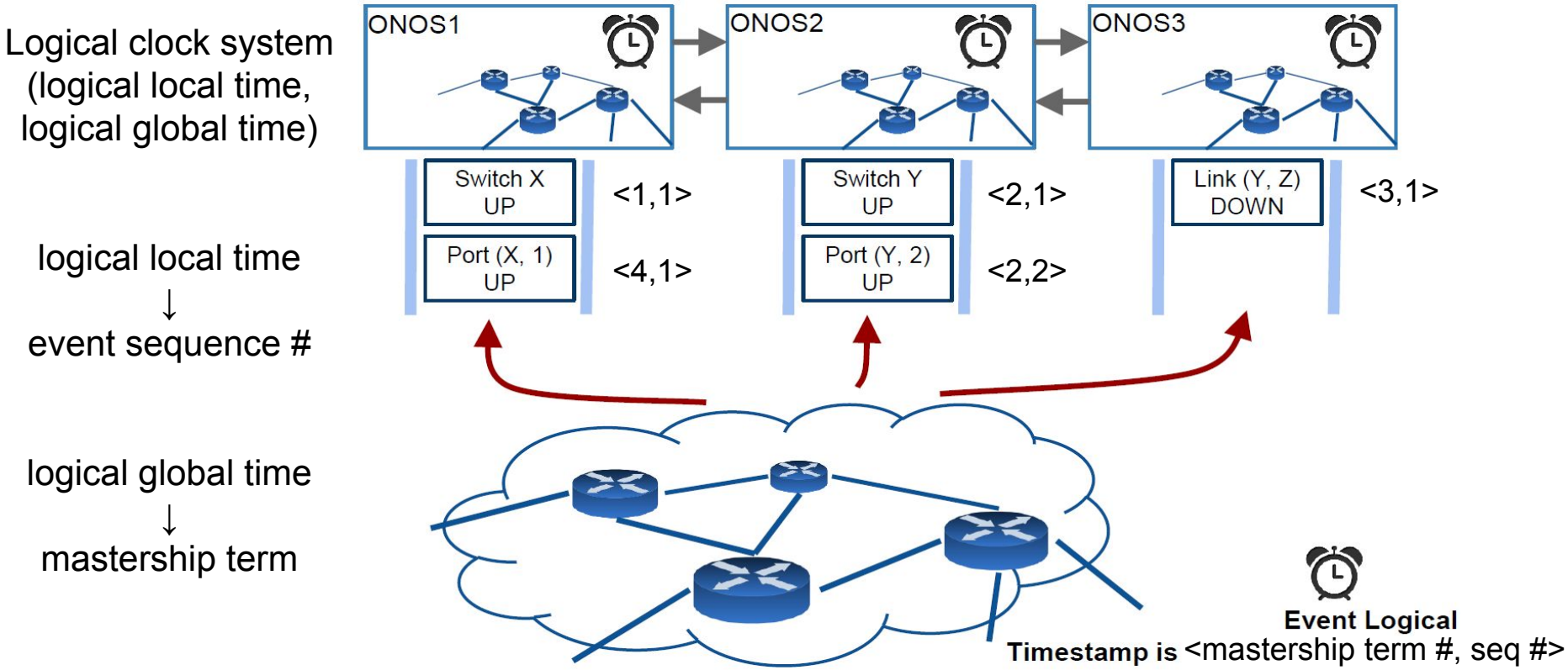
State and Properties

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

Network Topology State

Global Network View for NB Applications

- State □ inventory of devices, hosts and links
- Goal: closely track the state of physical network
- Fully replicated fro scale and low latency access
- Causal consistency using logical timestamps (logical clock systems)



Other States

States of Flow Rules and Flow Stats

- Switch specific state
  - Flow rules □ switch specific match-action pairs
  - Flow stats □ flow specific traffic stats
- Partitioned with current switch master serving as the authoritative copy
  - Can be fully replicated for quick failover
- Soft state
  - Inconsistencies are reconciled by refreshing from source

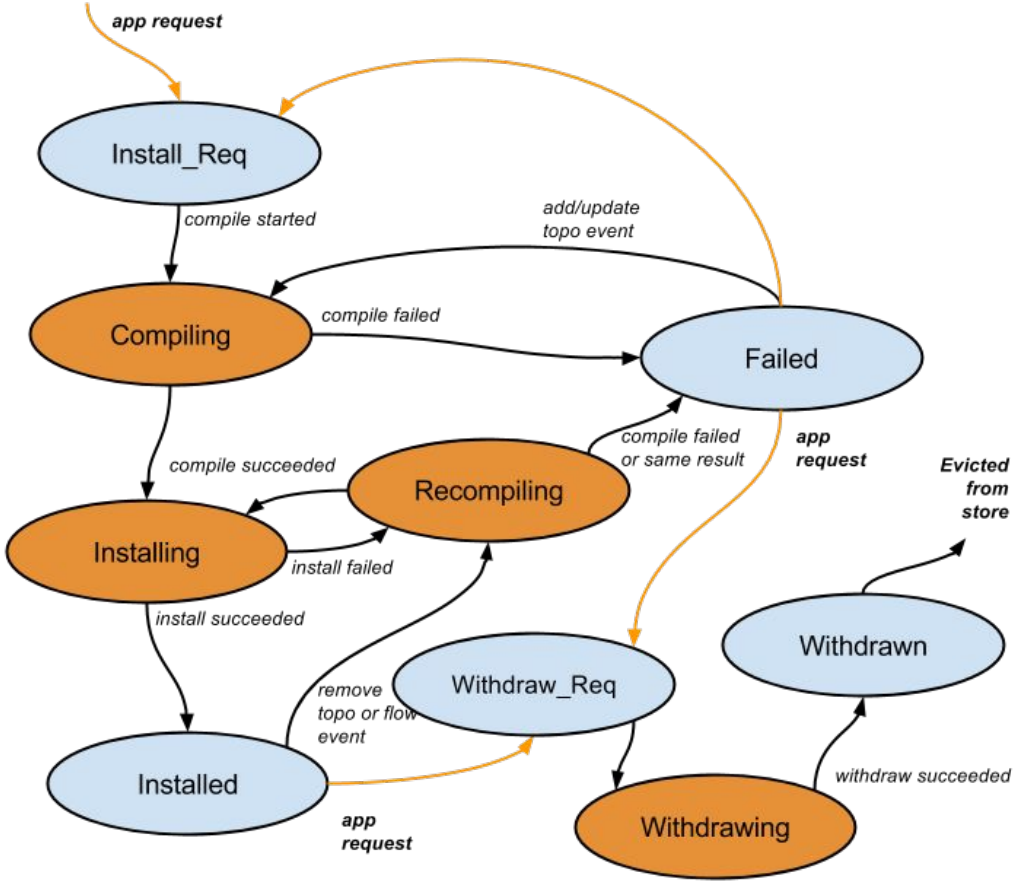
States of Other Elements

- Switch to controller mapping, application intents, resource allocations
- Require strong consistency
- Possible solutions
  - Hazelcast based solution
    - Used in ONOS version 1.0
    - Has some shortcoming on supporting for durability
  - Use of Raft consensus algorithm
    - Provides all the desired consistency and durability properties

Application and Intent Framework

Intent Framework

- Programming abstraction
  - Intents
  - Compilers
  - Installers
- Execution framework
  - Intent service
  - Intent store



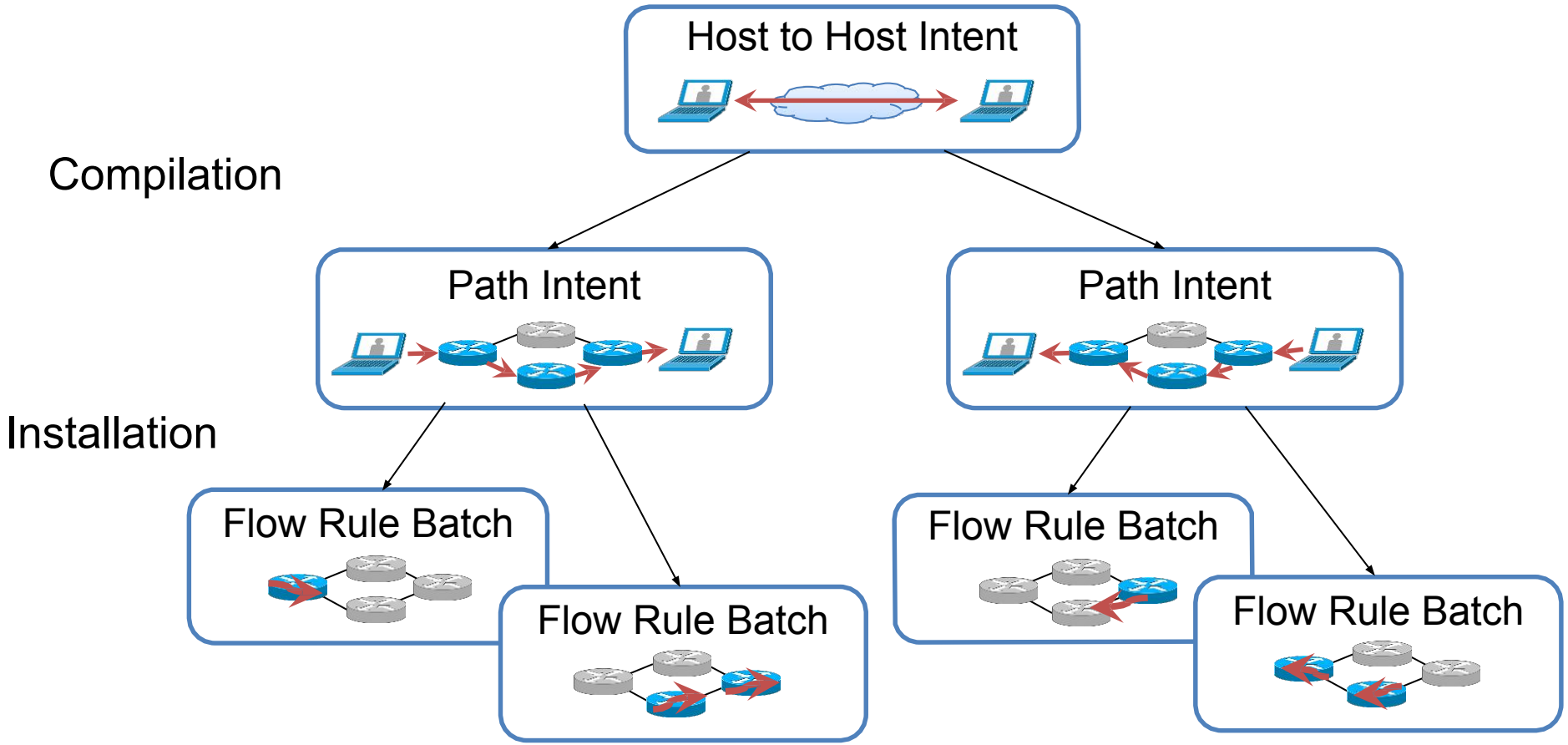
Intents

- Provide a high-level interface that focuses on ***what*** should be done rather than ***how*** it is specifically programmed
- Abstract network complexity from applications
- Extend easily to produce more complex functionality through combinations of other intents

Intent Example

Compiler & Installer

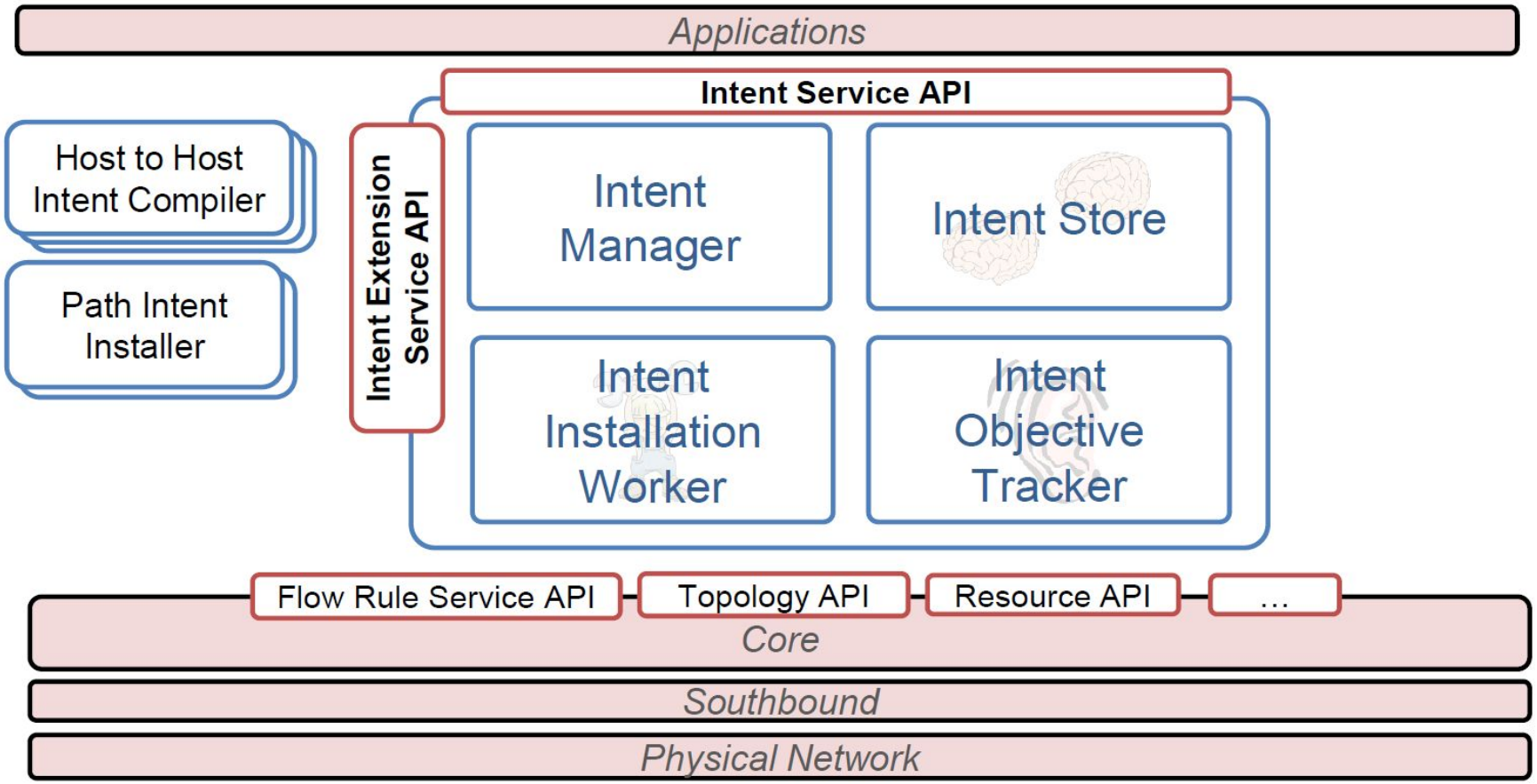
- Compiler: produce more specific Intents given the environment
  - E.g., find the corresponding paths between two hosts
- Installer: transform Intents into device commands



Intent Framework Design

Intent Framework

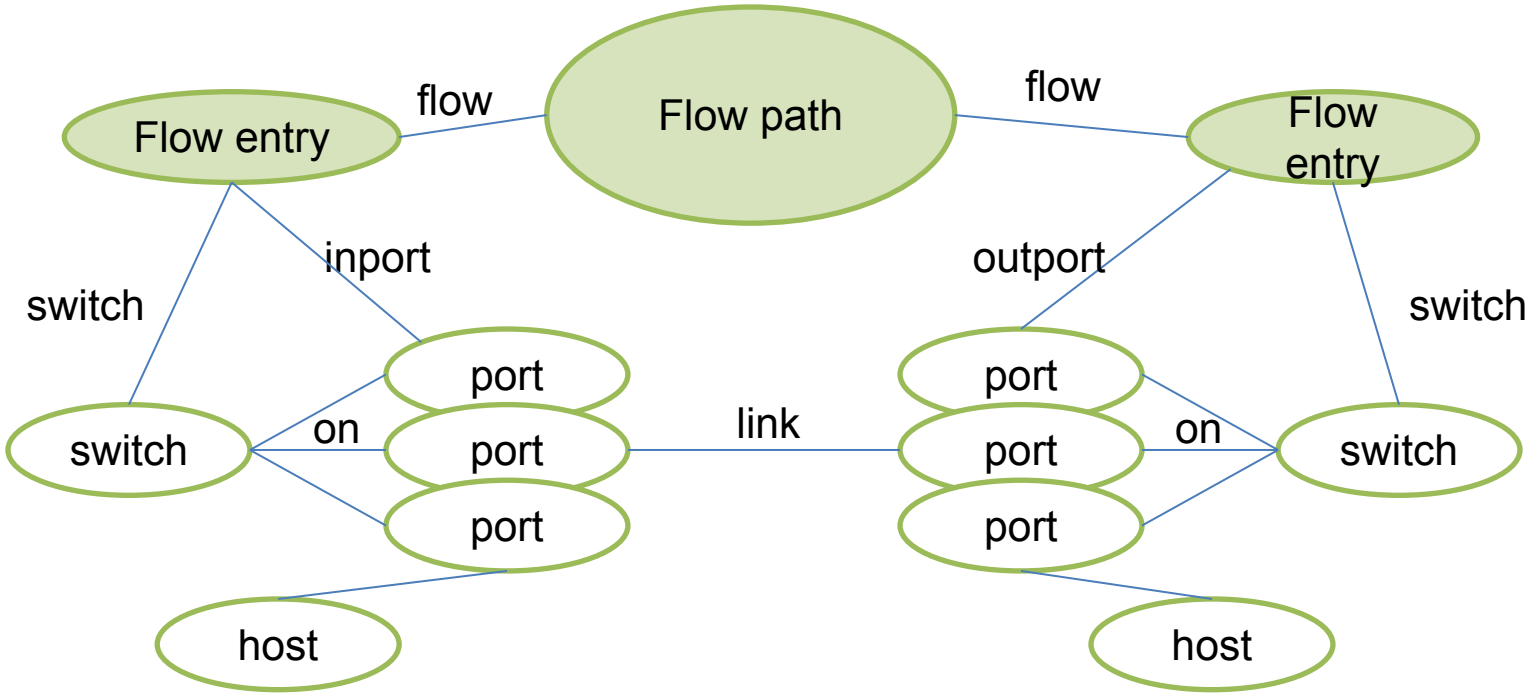
- Translates intents into device instructions (state, policy)
- Reacts to changing network conditions
- Extends dynamically to add, modify functionality (compilers, installers)



Representing Networks

Network Graph

- Graph has basic network objects like switch, port, device and links
- Application computes path by traversing the links from SRC to DST
- Application writes each flow entry for the path



Adapter Layer

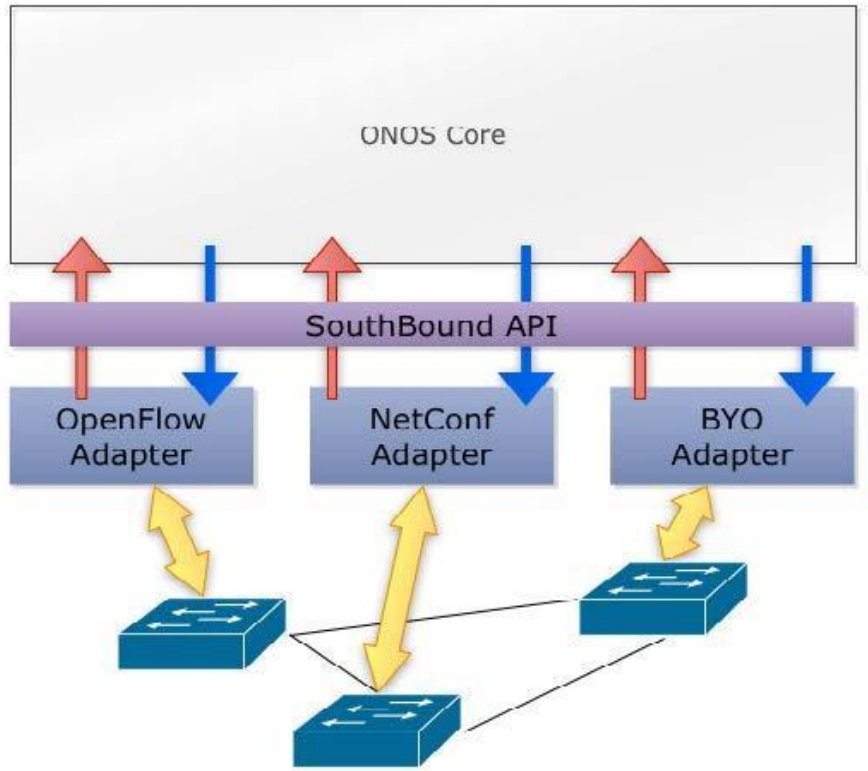
Design Consideration of Adapter

- ONOS supports multiple southbound protocols
- Adapters provide descriptions of data plane elements to the core
- Adapters hide protocol complexity from ONOS

Device Subsystem

- Responsible for discovering and tracking the devices
- Enable operators and apps to control the devices
- Core model relies on the Device and Port model objects

DeviceManager	OFDeviceProvider
Device	OpenFlowSwitch
DeviceId/ElementId	Dpid
Port	OFPortDesc
MastershipRole	RoleState



- POX
- https://brianlinkletter.com/2015/04/using-the-pox-sdn-controller/
- https://opensourcenetworksimulators.com/2015/09/using-pox-components-to-create-a-software-defined-networking-application/

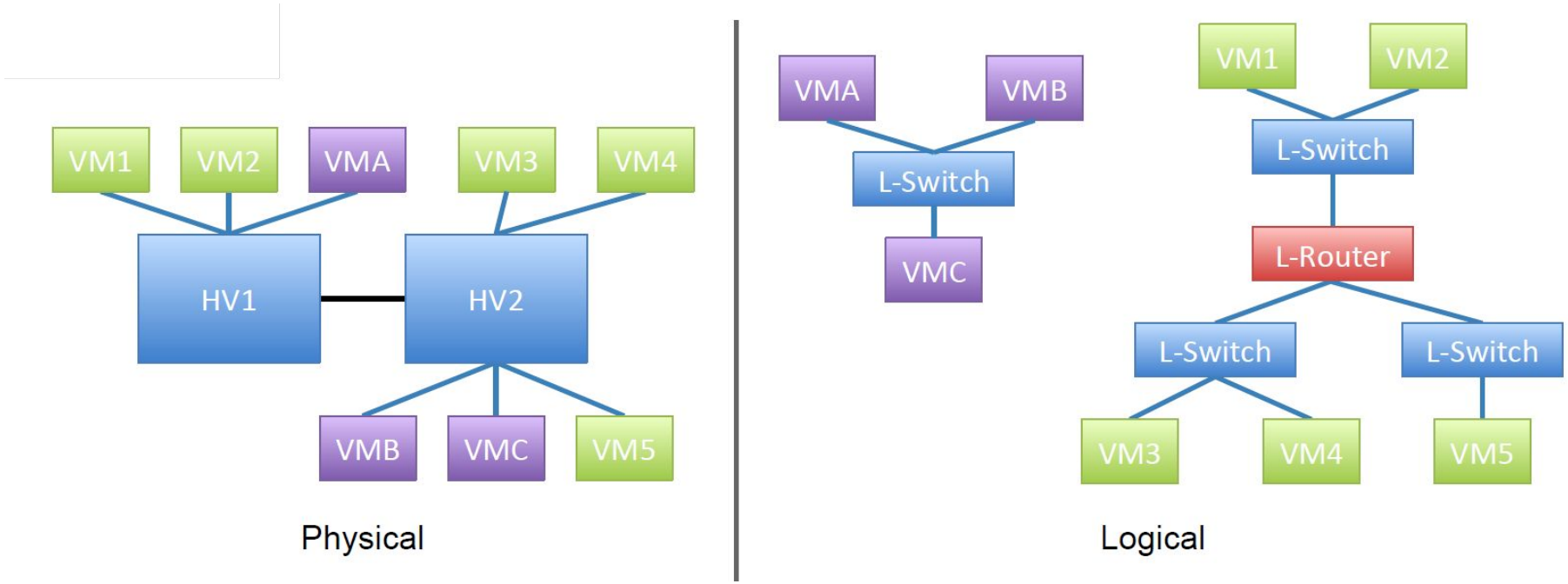
- OpenDaylight
- https://networklessons.com/network-automation/introduction-to-sdn-open daylight

- ONOS
- https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial

OVN: Open Virtual Network for Open vSwitch

Virtual Networking Overview

- Provides a logical network abstraction on top of a physical network



What is OVN?

- Open source virtual networking for Open vSwitch (OVS)
- Provides L2/L3 virtual networking
  - ✓ Logical switches
  - ✓ L2/L3/L4 ACLs (no connection tracking yet)
    - – Logical routers
    - – Security groups
  - ✓ Multiple tunnel overlays (Geneve, STT, and VXLAN)
    - – TOR-based and software-based logical-physical gateways
- Work on same platforms as OVS
  - ✓ Linux (KVM and Xen)
  - ✓ Containers
    - ? DPDK
    - – Hyper-V
- Integration with:
  - ✓ OpenStack
  - – Other CMSes

The Particulars

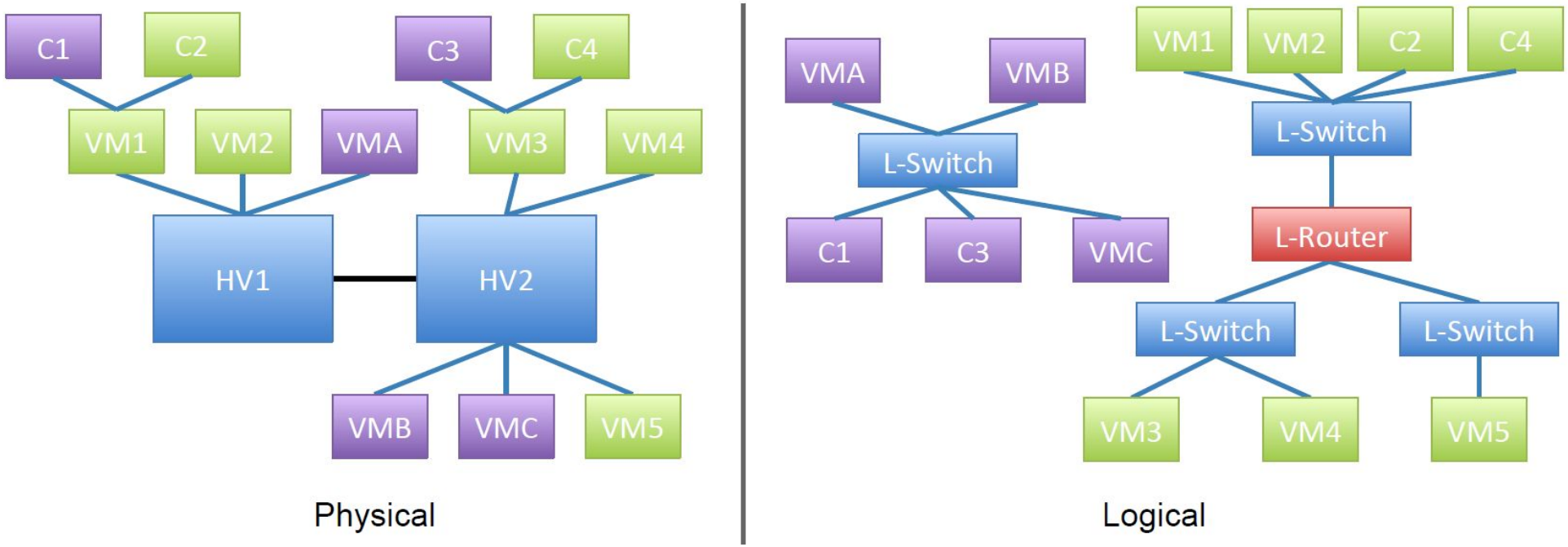
- Developed by the same community as Open vSwitch
- Vendor-neutral
- Architecture and implementation have all occurred on public mailing lists
- Developed under the Apache license

Goals

- Production-quality
- Straight-forward design
- Scale to thousands of hypervisors (each with many VMs and containers)
- Improved performance and stability over existing plugin

Container Integration

- Containers nested inside VMs can be in logical networks too!

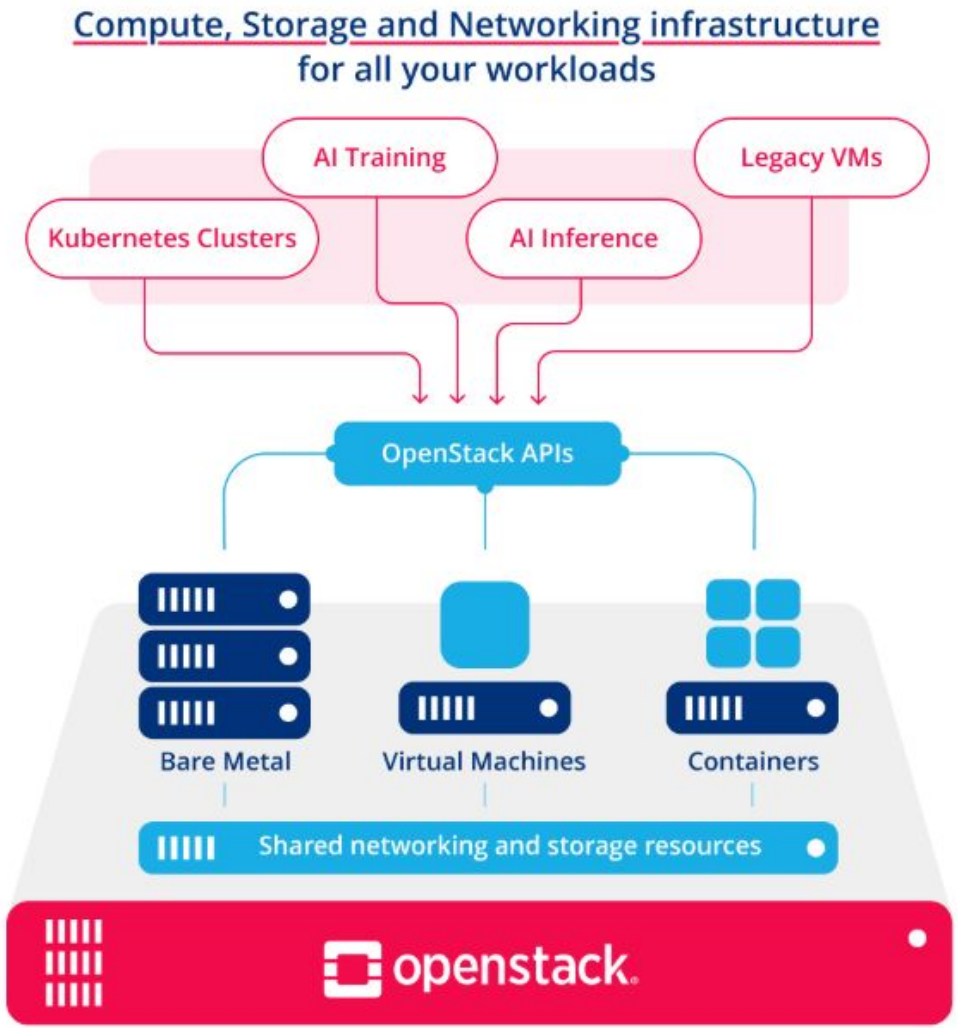


OpenStack

- OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed and provisioned through APIs with common authentication mechanisms.

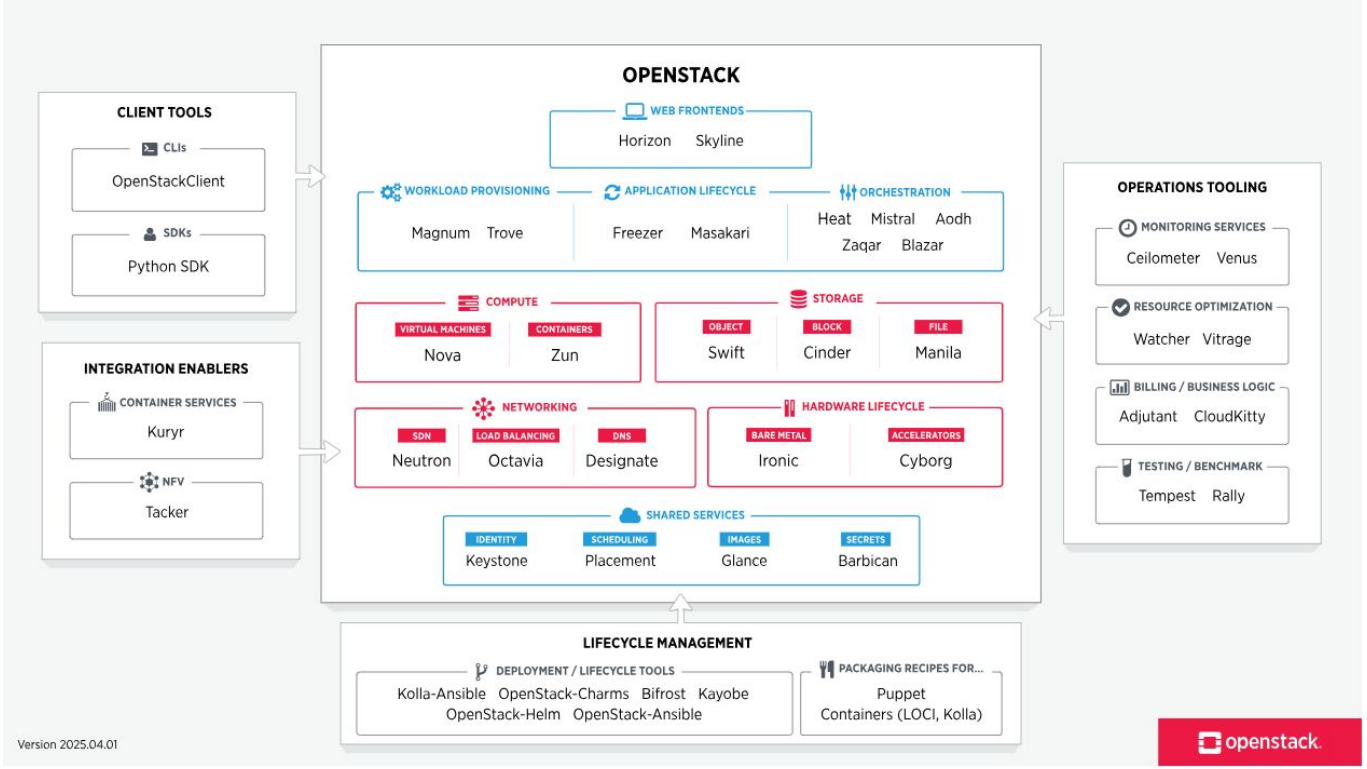
- Dashboards are also available, giving administrators control while empowering their users to provision resources through a web interface. There are OpenStack CLI tools and SDKs giving operators the flexibility to create OpenStack cloud applications in the language of their choice. Supported languages include Go, Python, Ruby, and Java.

- Beyond standard infrastructure-as-a-service functionality, additional components provide orchestration, fault management and service management amongst other services to provide operators flexibility to customize their infrastructure and ensure high availability of user applications.

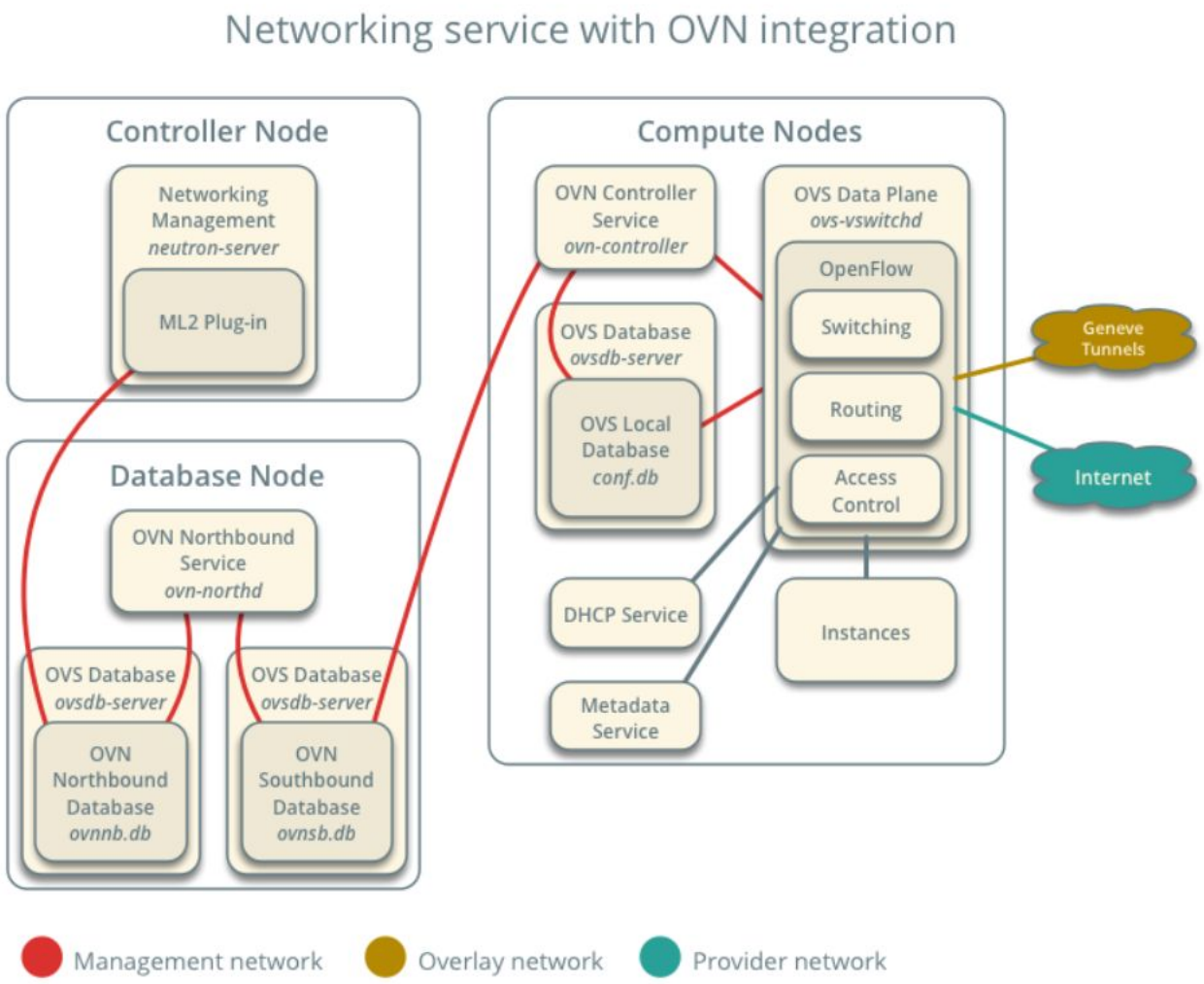


THE OPENSTACK LANDSCAPE

- OpenStack's modular framework allows you to identify and deploy components depending on your needs. The OpenStack map gives you a high level overview of the OpenStack landscape to see where those services fit and how they can work together.



OpenStack Integration with OVN

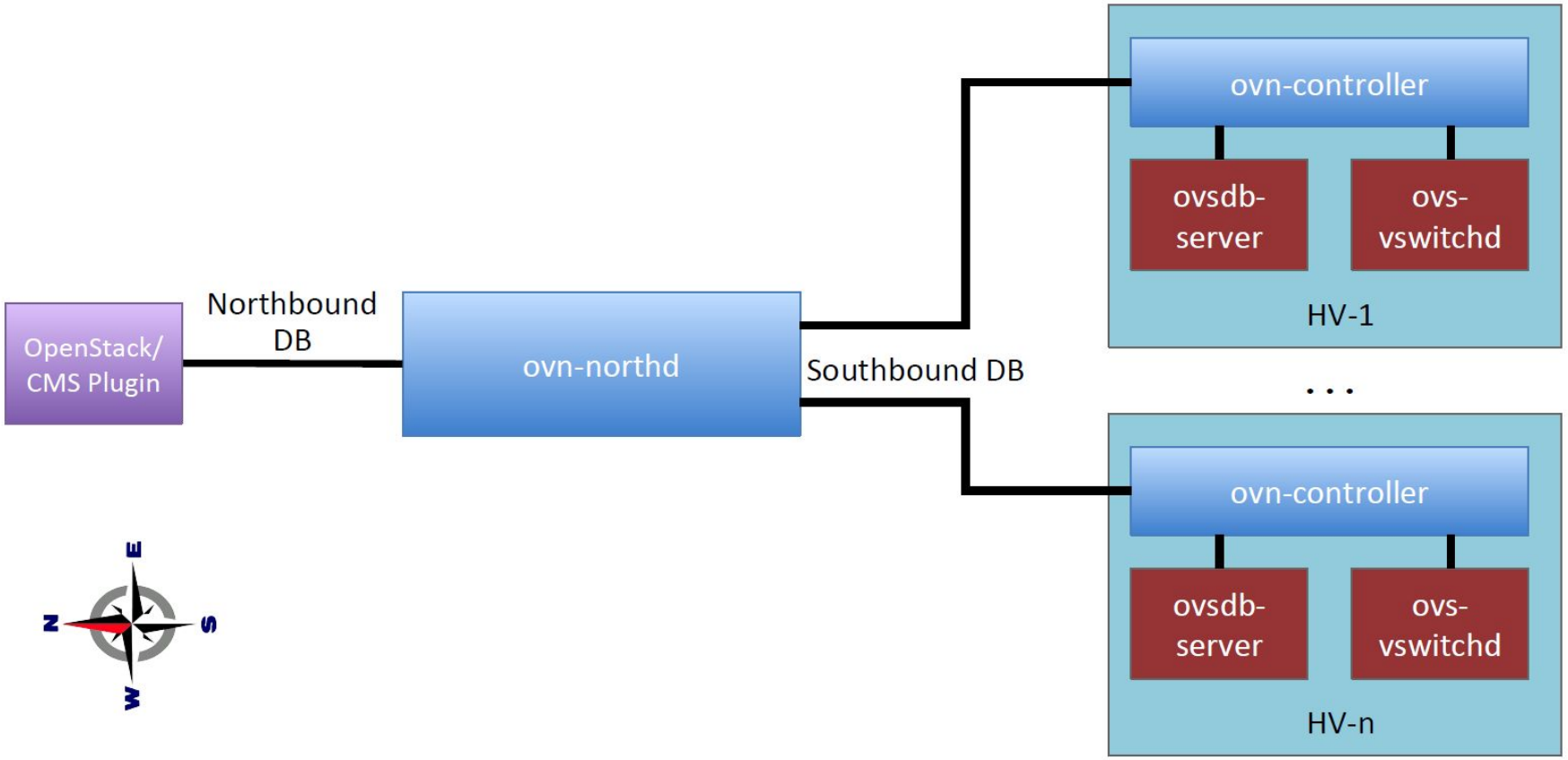


https://docs.openstack.org/networking-ovn/pike/admin/refarch/refarch.html

Designed to Scale

- Configuration coordinated through databases
- Local controller converts logical flow state into physical flow state
- Desired state clearly separated from run-time state

OVN Architecture



The OVN Databases

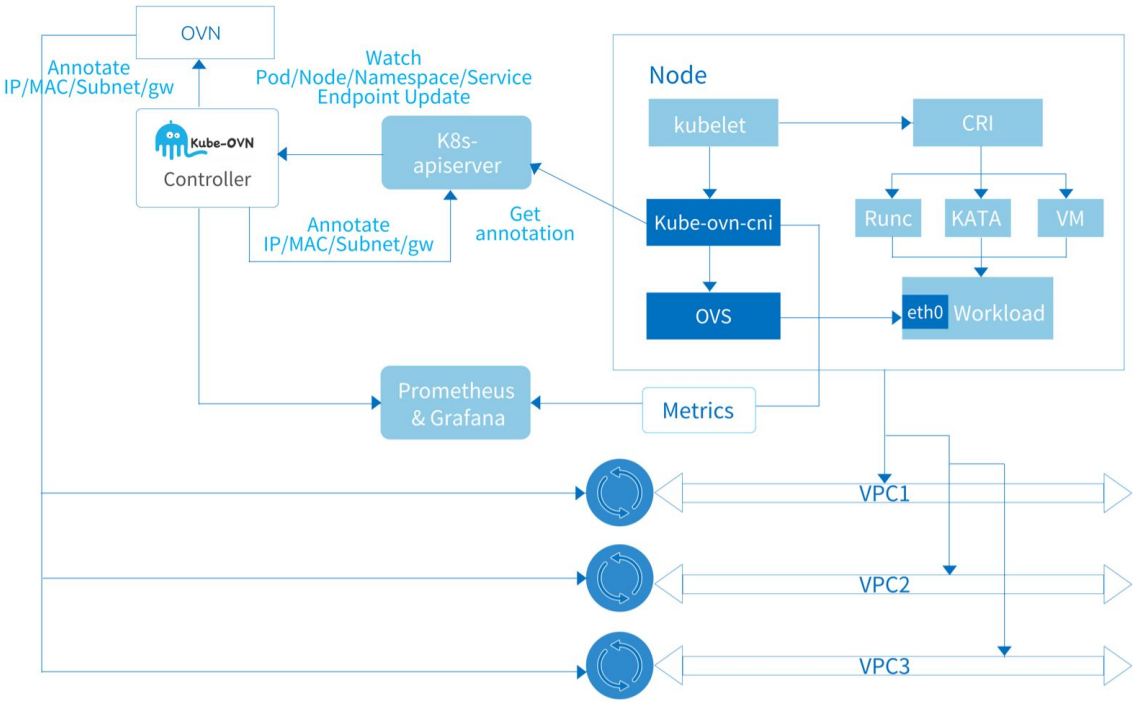
- ovn-northbound
  - OpenStack/CMS integration point
  - High-level, desired state
    - Logical ports -> logical switches -> logical routers
- ovn-southbound
  - Run-time state
    - Location of logical ports
    - Location of physical endpoints
    - Logical pipeline generated based on configured and run-time state

The Daemons

- Central: ovn-northd
  - Converts from the high-level northbound DB to the run-time southbound DB
  - Generates logical flows based on high-level configuration

- Per-hypervisor: ovn-controller
  - Registers chassis and VIFs to southbound DB
  - Converts logical flows into physical flows (ie, VIF UUIDs to OpenFlow ports)
  - Pushes physical configuration to local OVS instance through OVSDB and OpenFlow

Model for Kube-OVN (New in 2025)



- Kube-OVN is an enterprise-level cloud-native network orchestration system under CNCF that combines the capabilities of SDN with cloud-native technologies, providing the most functions, extreme performance and the easiest operation.
- Kube-OVN uses Open Virtual Network (OVN) and Open vSwitch at the underlying layer to implement network orchestration and exposes its rich capabilities to Kubernetes networking.