ONOS

Architecture, Abstractions & Performance

What is ONOS?

Open Network Operating System (ONOS) is an open source SDN network operating system. Our mission is to enable Service Providers to build real SDN/NFV Solutions.

ONOS Community

Brief Retrospective

- Started with a minimal platform with only a few apps
  - built with sound structure and solid code & minimalistic REST API
  - 4 apps and 1 SB plugin
- Added new core functionality and apps with each release
  - deliberately balancing investments in platform vs. use-cases and apps
  - show innovation, but also take pragmatic steps to be deployment-ready
  - maintain coherence of architecture and quality of code
- Now a platform with many features and apps
  - new capabilities, distributed primitives and even greater extensibility
  - now 70+ apps, including SB plugins, drivers, and samples

Quarterly Releases

- **Avocet** (1.0.0) released 2014-12
  - Initial release of clean and modular code-base, protocol independence
- **Blackbird** (1.1.0) released 2015-03
  - Improved performance, scale-out, increased robustness
- **Cardinal** (1.2.0) released 2015-06
  - New use-cases, additional core features, additional SB protocols
- **Drake** (1.3.0) released 2015-09
  - Platform enhancements, security, UI enhancements
- **Emu** (1.4.0) - released 2015-12
  - CORD features, prototype of dynamic cluster scaling
- **Falcon** (1.5.0) - released 2016-03
  - Dynamic cluster scaling, model extensibility, intents on flow objectives

Quarterly Releases

- **Falcon** (1.5.0) - released 2016-03
  - dynamic cluster scaling, model extensibility, intents on flow objectives
- **Goldeneye** (1.6.0) - planned for 2016-06
  - spring cleaning, intent framework, YANG tools, GUI scaling, P4 PoC
- **H...** (1.7.0) - planned for 2016-09
  - separate platform & core, network hypervisor, YANG at NB, P4 support
- . . .

Platform Hardening

- Significantly improved performance
  - published white-paper and established relevant performance metrics
- Further increased quality and fault-tolerance
  - fixed defects and added a repertoire of robust distributed structures
  - fixed defects in 3rd party code and contributed changes upstream
- Improved security
  - northbound (REST, CLI & GUI), southbound and east-west secured
- Improved usability and supportability
  - deployment, component configurability, centralized app management
  - network configuration, GUI enhancements & extensibility
  - dynamic cluster scaling and model extensibility

Process Enhancements

- Established deprecation policy for API compatibility
  - give fair warning to app developers before APIs change or vanish
  - balances stability vs. ability to innovate or respond to feedback
- Incubating functionality over multiple releases
  - development of some features takes more time than a single release
  - introduce preliminary functionality in one release
  - harden & refine in the next release
- Broadening the set of code submitters
  - granting ability to +2/submit to developers based on code/review merit
  - serves both to empower the community and to off-load the core team

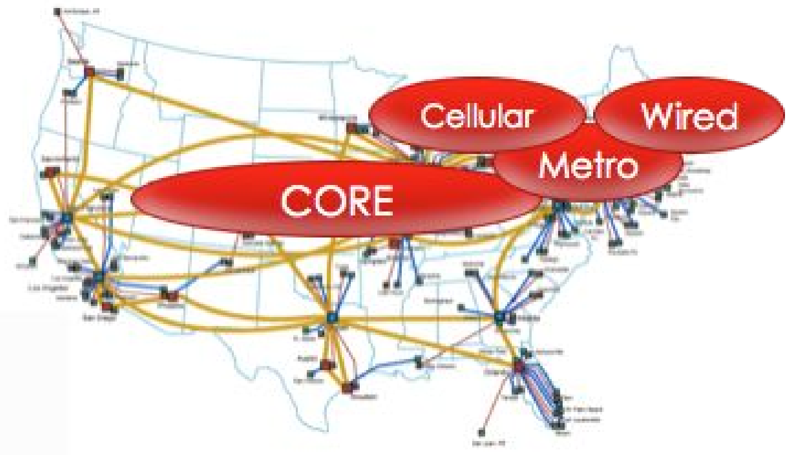
ONOS & Approach to SDN

- Move with urgency, but deliberately
- Mind the fundamentals & beware of yak-shaving
- Keep balance between innovation, utility and stability
- Allow *legacy* devices to participate in SDN, but not to deform or diminish the SDN vision

Why ONOS?

Service Provider Networks

- WAN core backbone
  - Multi-Protocol Label Switching (MPLS) with Traffic Engineering (TE)
  - *200-500 routers, 5-10K ports*
- Metro Networks
  - Metro cores for access networks
  - *10-50K routers, 2-3M ports*
- Cellular Access Networks
  - LTE for a metro area
  - *20-100K devices, 100K-100M ports*
- Wired access / aggregation
  - Access network for homes; DSL/Cable
  - *10-50K devices, 100K-1M ports*

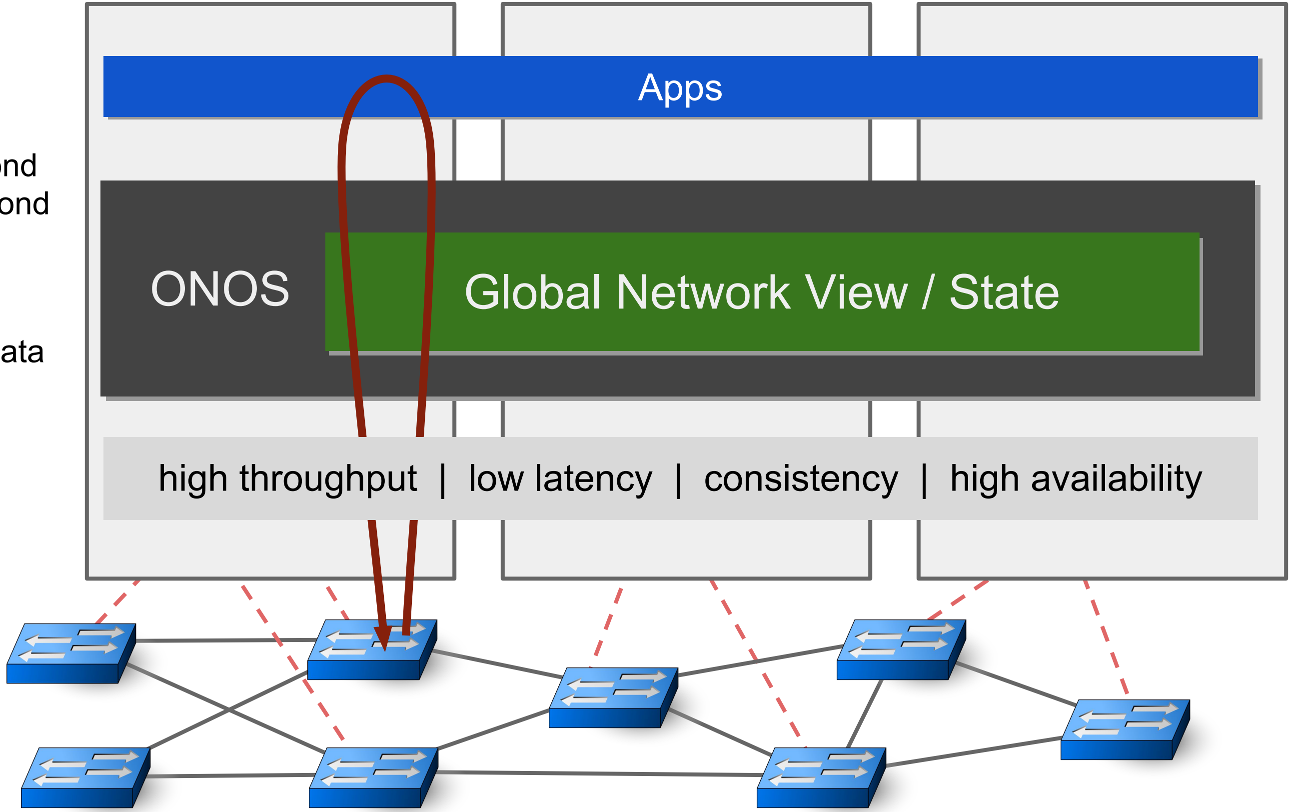


Key Performance Requirements

High Throughput:
~500K-1M paths setups / second
~3-6M network state ops / second

High Volume:
~500GB-1TB of network state data

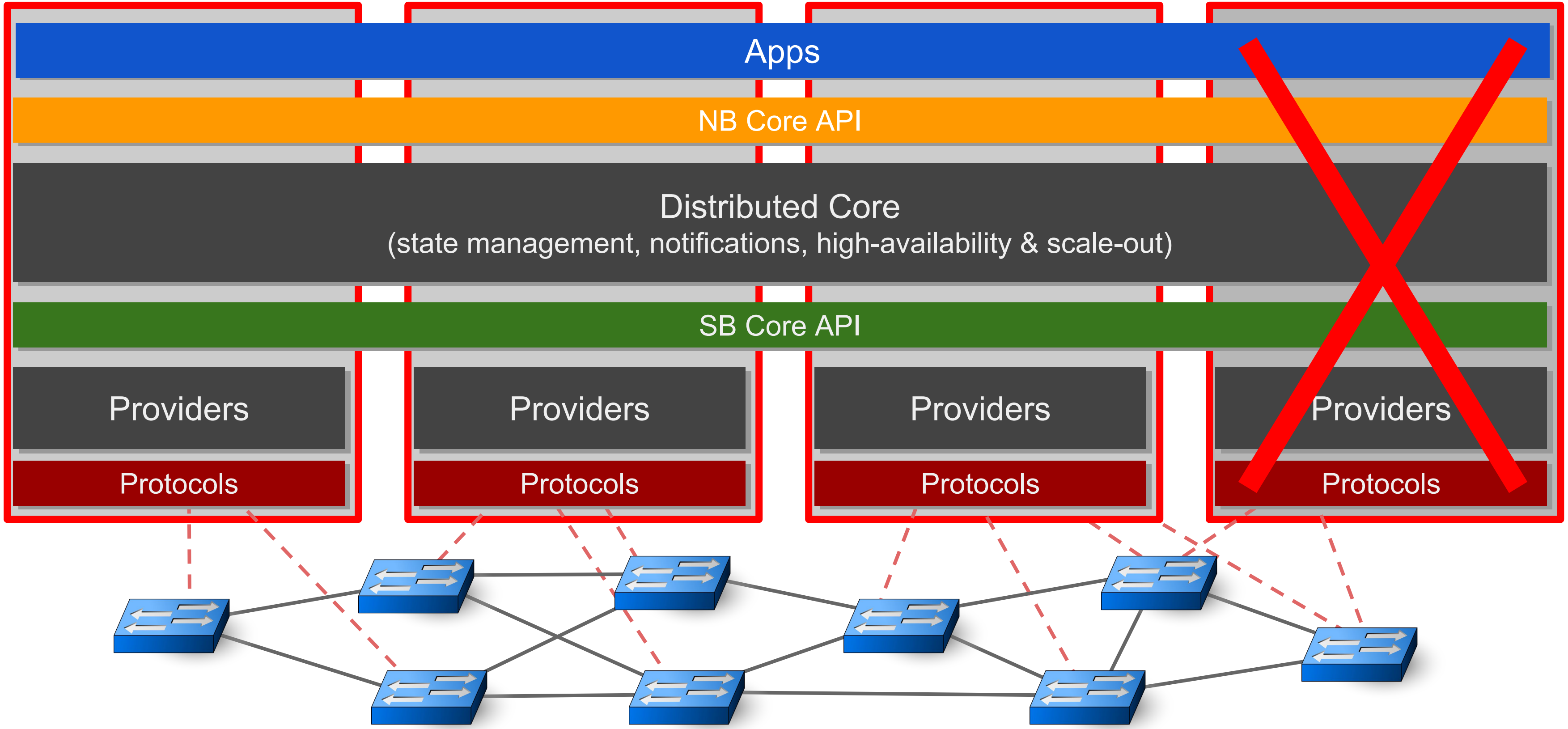
Difficult challenge!



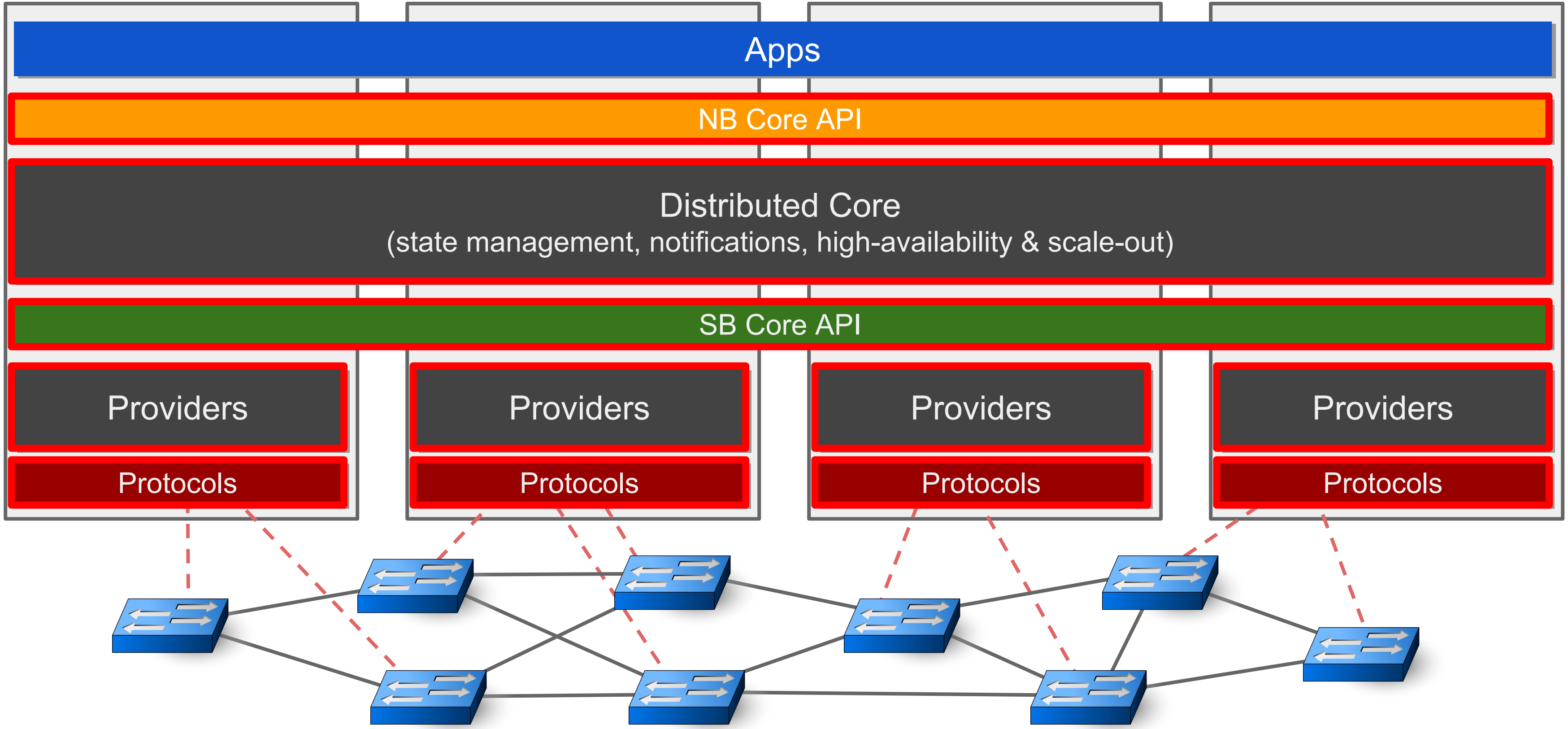
Architectural Tenets

- High-availability, scalability and performance
  - required to sustain demands of service provider & enterprise networks
- Strong abstractions and simplicity
  - required for development of apps and solutions
- Protocol and device behaviour independence
  - avoid contouring and deformation due to protocol specifics
- Separation of concerns and modularity
  - allow tailoring and customization without speciating the code-base

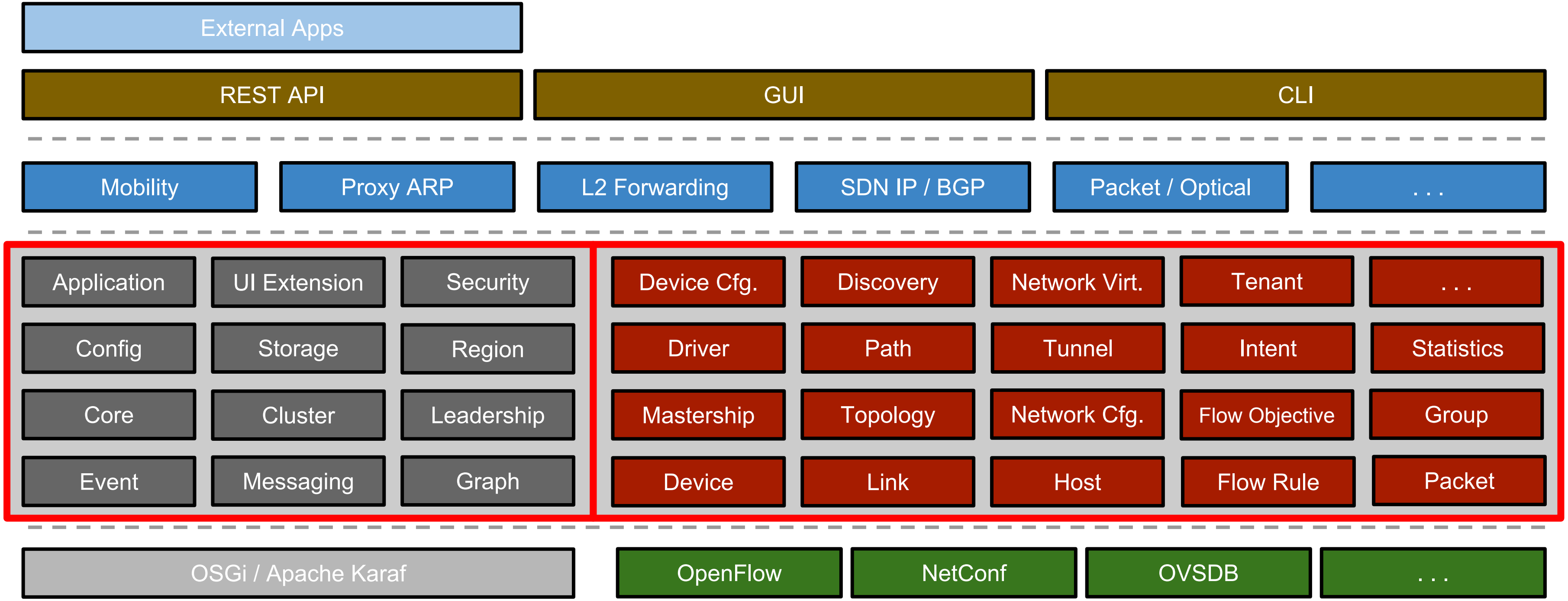
ONOS Distributed Architecture



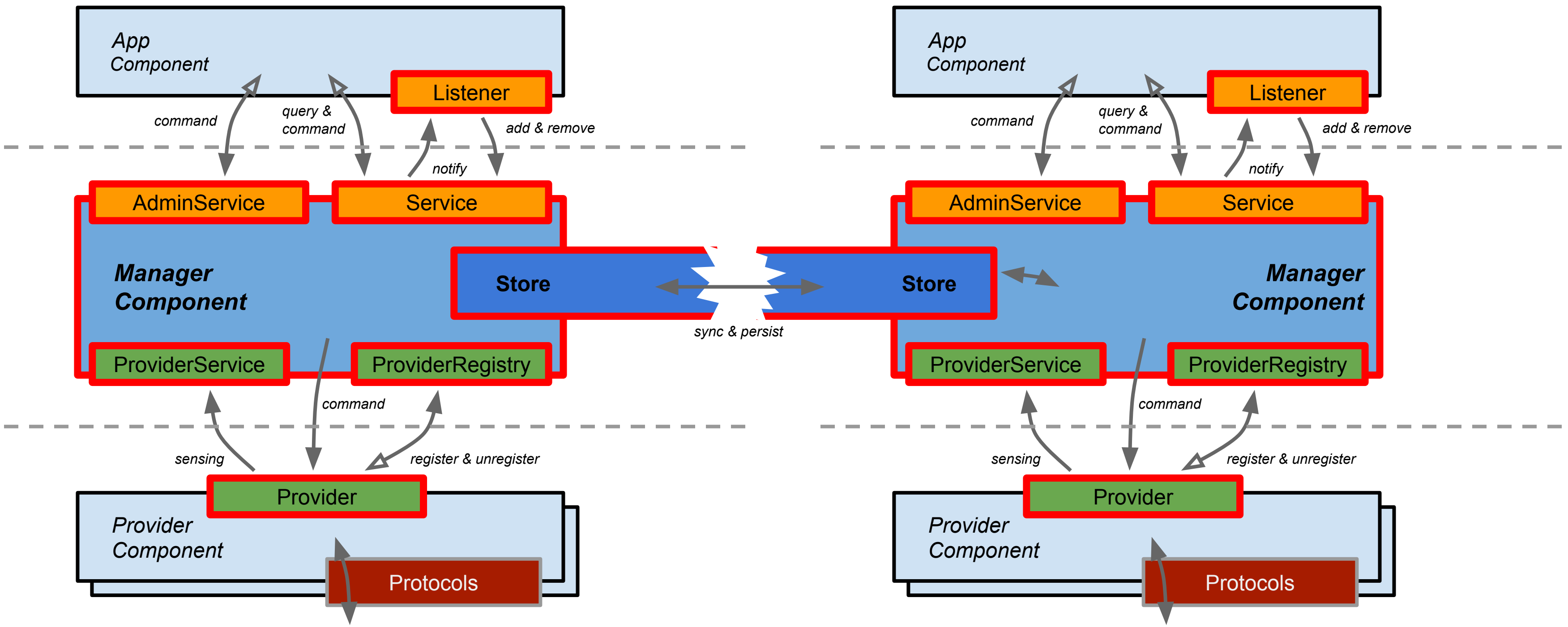
ONOS Distributed Architecture



ONOS Core Subsystems



ONOS Core Subsystem Structure

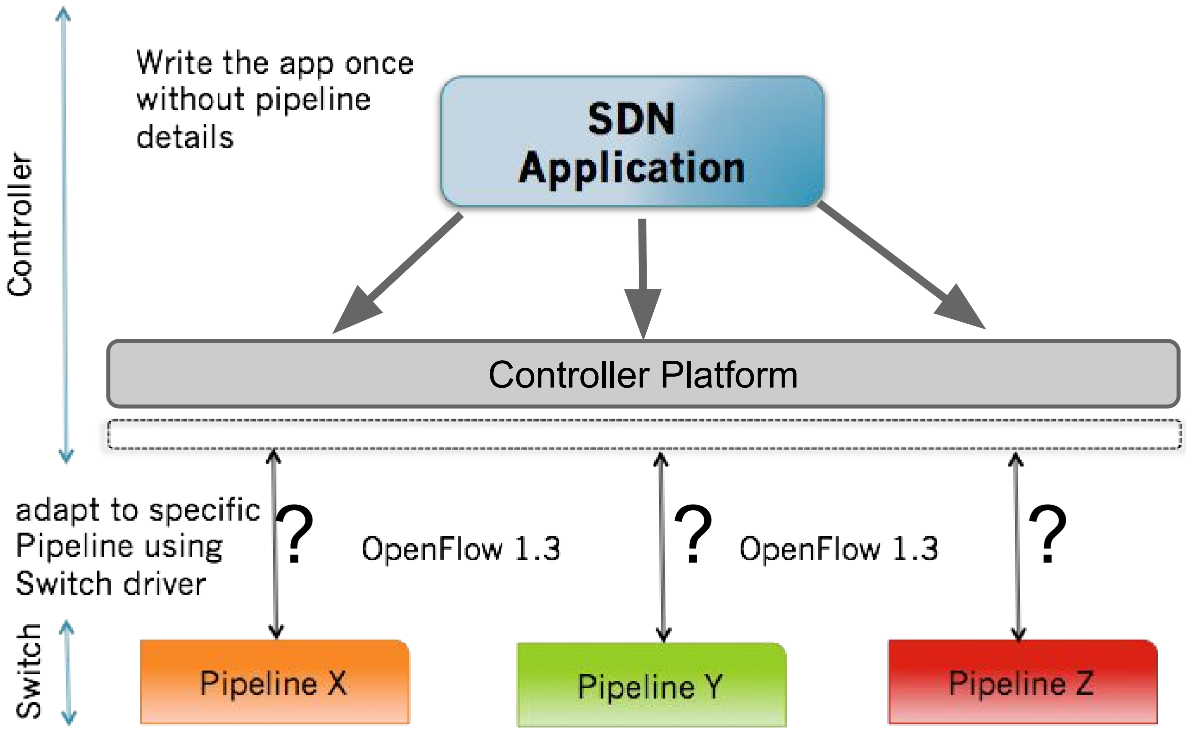


Key Northbound Abstractions

- Network Graph
  - Directed, cyclic graph comprising of infrastructure devices, infrastructure links and end-station hosts
- Flow Objective
  - Device-centric abstraction for programming data-plane flows in table pipeline-independent manner
- Intent
  - Network-centric abstraction for programming data-plane in topology-independent manner

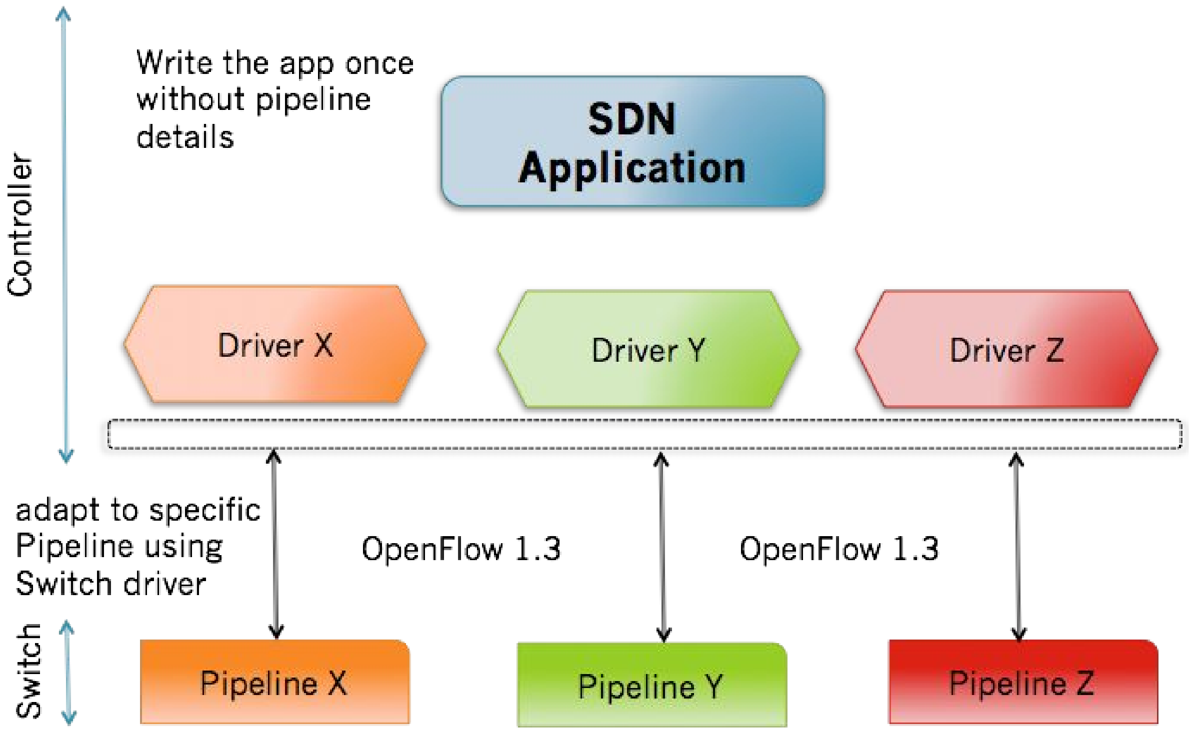
Flow Objective Subsystem

- **Problem:** Applications today must be pipeline aware, effectively making them applicable to specific HW.



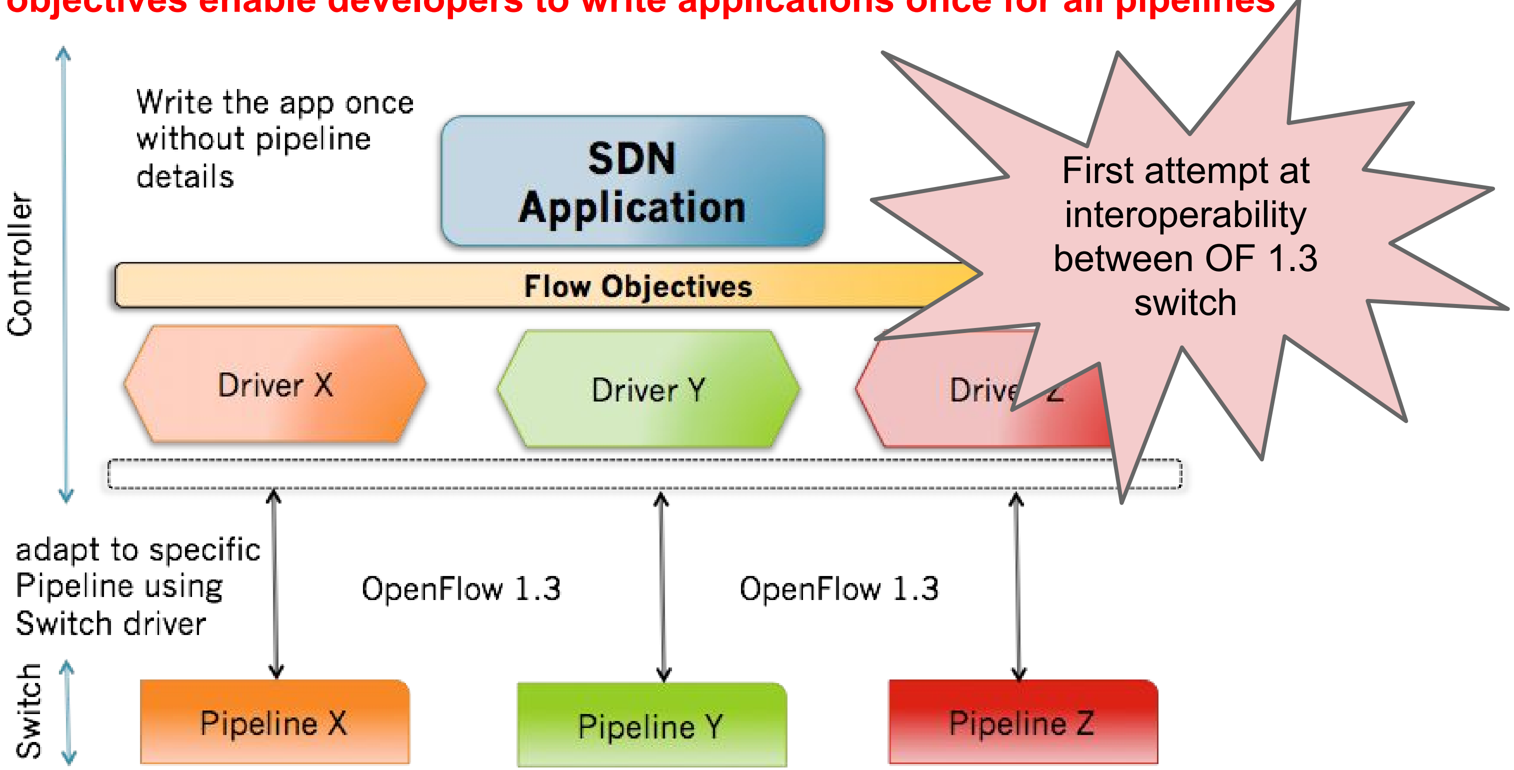
Flow Objective Subsystem

- **Problem:** Applications today must be pipeline aware, effectively making them applicable to specific HW.

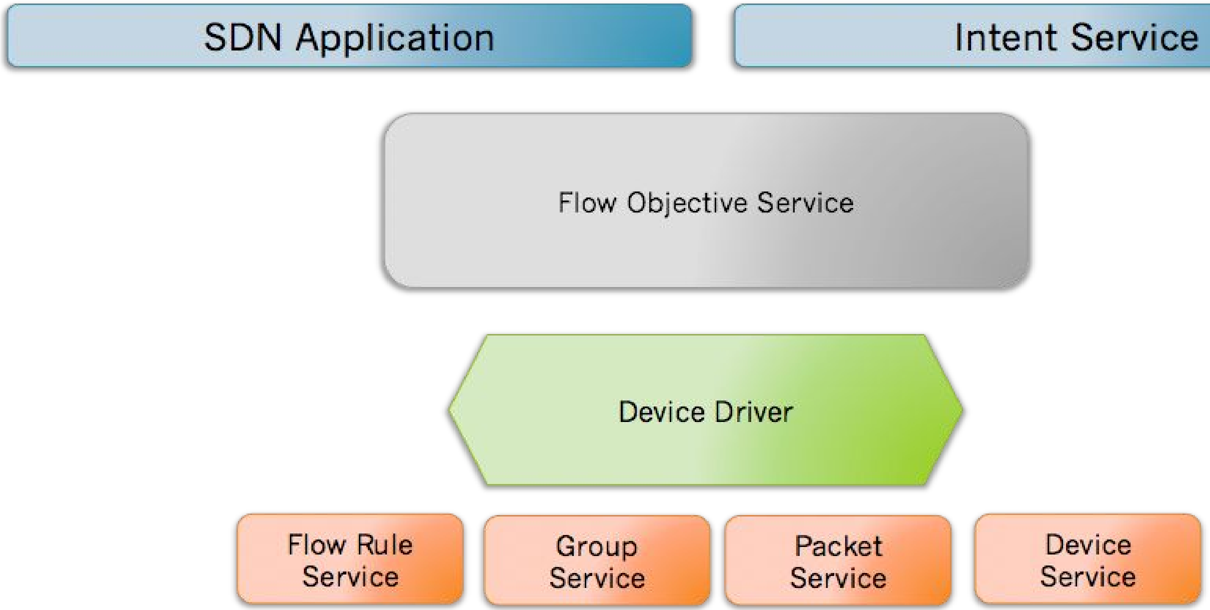


Flow Objective Abstraction

- **Problem:** Applications currently must be pipeline aware, effectively making applicable on specific HW.
  Flow objectives enable developers to write applications once for all pipelines



Flow Objective Service



- Applications use Objective to take advantage multi-table architectures
- Other services also make use of the Objective service (eg. Intent Service)
- Device driver translates objectives to the specific flow rules for a given device

Flow Objectives

- Flow Objectives describe a SDN application's objective behind a **flow** it is sending to a **device**

- We currently only have three types of objectives:
    1. Filtering Objective
    2. Forwarding Objective
    3. Next Objective

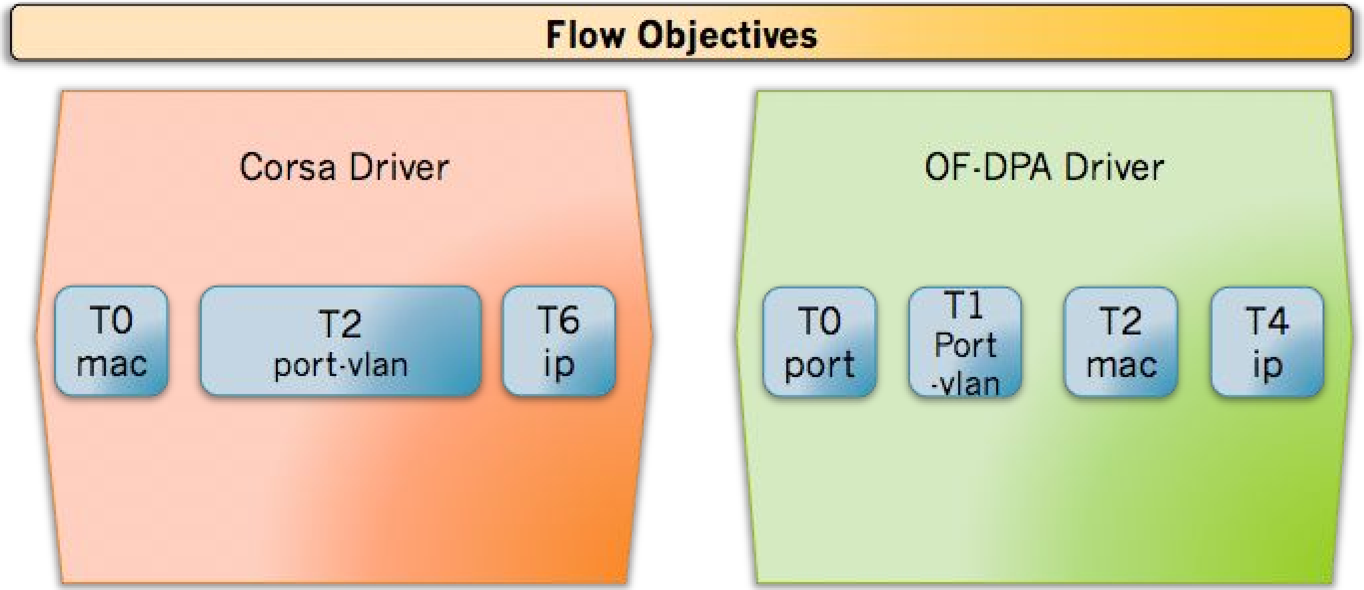
Filtering Objective

- Filter -> only Permit or Deny options
- On criteria (match fields)

Example:
Peering Router

Switch Port : **X**

Permit: MAC 1, VLAN 1, IP 1, 2, 3
Permit: MAC 1, VLAN 2, IP 4, 5



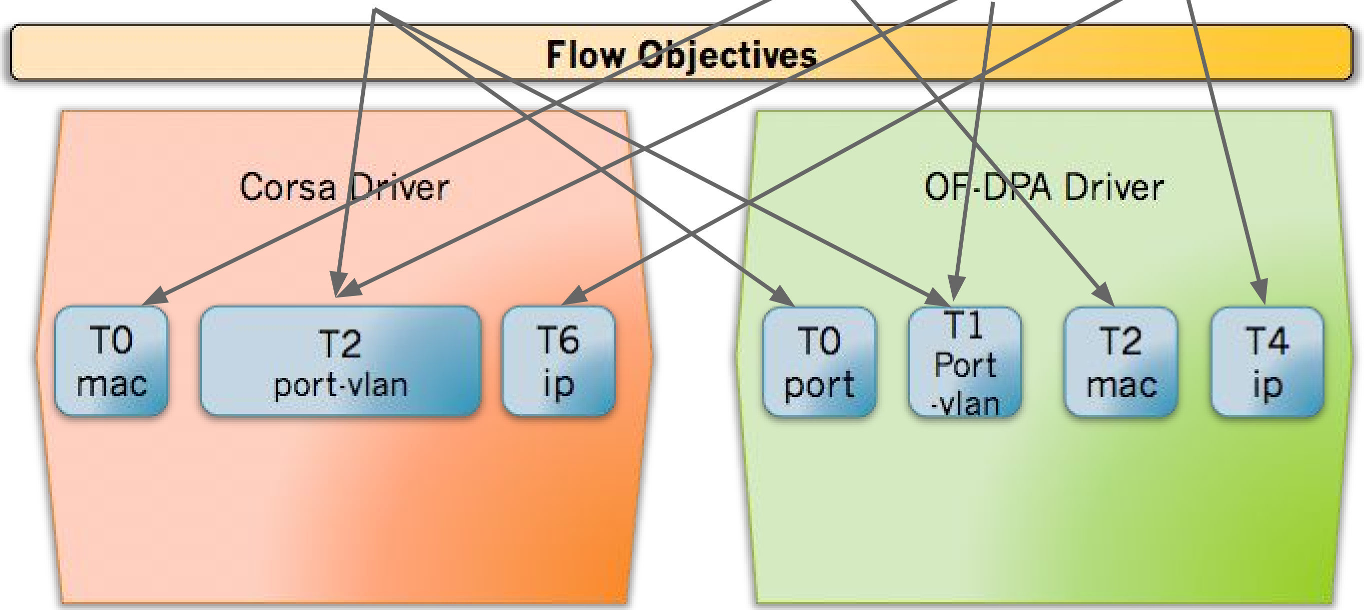
Filtering Objective

- Filter -> only Permit or Deny options
- On criteria (match fields)

Example:
Peering Router

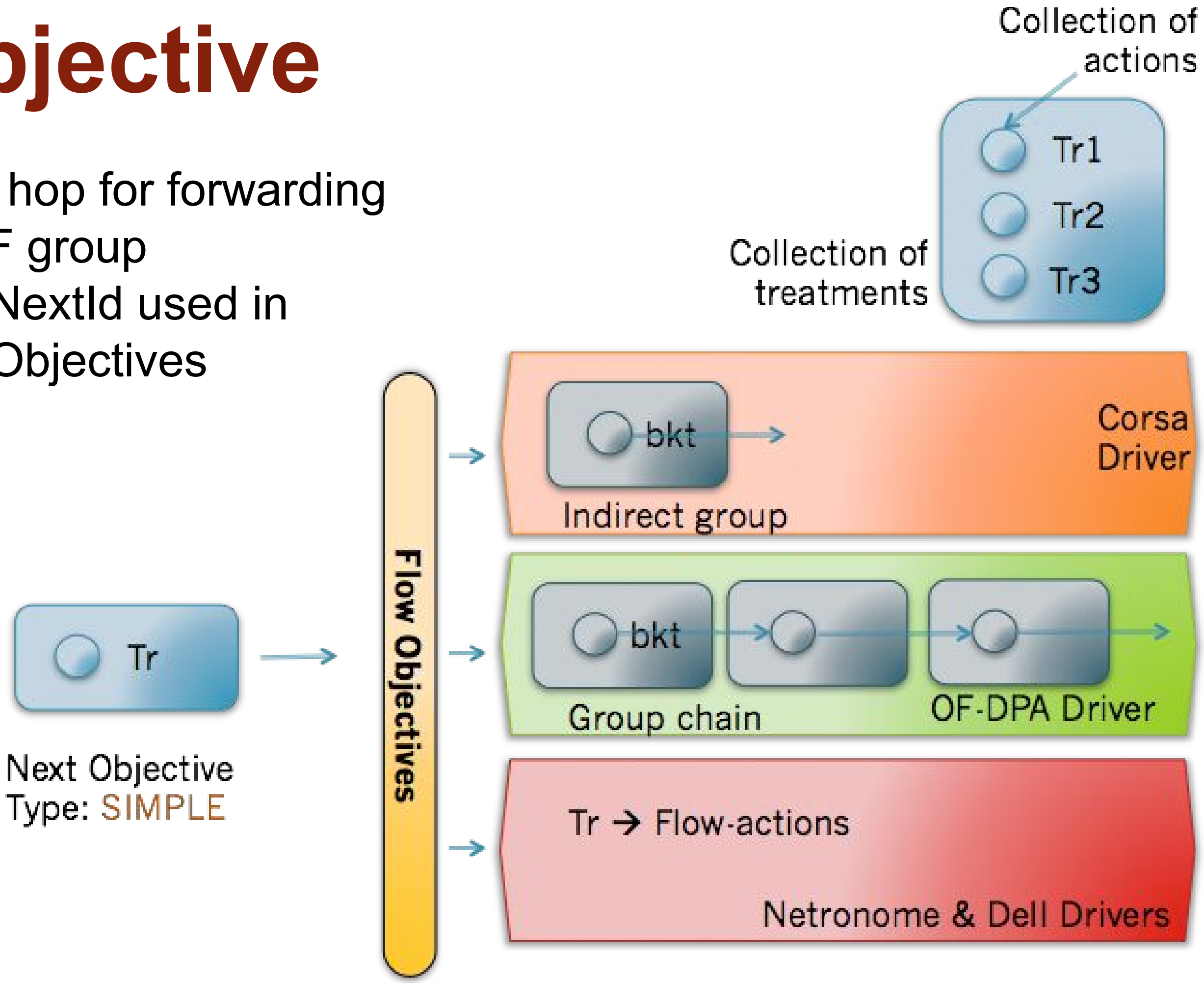
Switch Port : **X**

Permit: MAC 1, VLAN 1, IP 1, 2, 3
Permit: MAC 1, VLAN 2, IP 4, 5



Next Objective

- Next -> next hop for forwarding
- Similar to OF group
- Keyed by a NextId used in Forwarding Objectives



Forwarding Objective

- Forwarding: { Selector -> Next Id }
- Forwarding Types: Specific or Versatile
  - Specific -> MAC, IP, MPLS forwarding tables
  - Versatile -> ACL table
- NextId is resolved to whatever the driver previously built for the corresponding Next Objective

Objectives - Simpler applications

```java
for (InterfaceIpAddress ipAddr : intfIps) {
    log.debug("adding rule for IPs: {}", ipAddr.ipAddress());
    selector = DefaultTrafficSelector.builder();
    treatment = DefaultTrafficTreatment.builder();

    selector.matchEthType(Ethernet.TYPE_IPV4);
    selector.matchIPDst(IpPrefix.valueOf(ipAddr.ipAddress(), 32));
    treatment.transition(Type.ACL);

    rule = new DefaultFlowRule(deviceId, selector.build(),
                               treatment.build(), HIGHEST_PRIORITY, appId,
                               0, true, FlowRule.Type.IP);

    ops = install ? ops.add(rule) : ops.remove(rule);
}
```

```java
for (MacAddress mac : intfMacs) {
    log.debug("adding rule for MAC: {}", mac);
    selector = DefaultTrafficSelector.builder();
    treatment = DefaultTrafficTreatment.builder();

    selector.matchEthDst(mac);
    treatment.transition(FlowRule.Type.VLAN_MPLS);

    rule = new DefaultFlowRule(deviceId, selector.build(),
                               treatment.build(),
                               CONTROLLER_PRIORITY, appId, 0,
                               true, FlowRule.Type.FIRST);

    ops = install ? ops.add(rule) : ops.remove(rule);
}
```

```java
for (Map.Entry<PortNumber, VlanId> portVlan : portVlanPair.entrySet()) {
    log.debug("adding rule for VLAN: {}", portVlan);
    selector = DefaultTrafficSelector.builder();
    treatment = DefaultTrafficTreatment.builder();

    selector.matchVlanId(portVlan.getValue());
    selector.matchInPort(portVlan.getKey());
    treatment.transition(Type.ETHER);
    treatment.deferred().popVlan();

    rule = new DefaultFlowRule(deviceId, selector.build(),
                               treatment.build(), CONTROLLER_PRIORITY, appId,
                               0, true, FlowRule.Type.VLAN);

    ops = install ? ops.add(rule) : ops.remove(rule);
}
```

```java
private void processIntfFilters(boolean install, Set<Interface> intfs) {
    log.info("Processing {} router interfaces", intfs.size());
    for (Interface intf : intfs) {
        FilteringObjective.Builder fob = DefaultFilteringObjective.builder();
        fob.withKey(Criteria.matchInPort(intf.connectPoint().port()))
            .addCondition(Criteria.matchEthDst(intf.mac()))
            .addCondition(Criteria.matchVlanId(intf.vlan()));
        intf.ipAddresses().stream()
            .forEach(ipaddr -> fob.addCondition(
                                Criteria.matchIPDst(ipaddr.subnetAddress())));
        fob.permit().fromApp(appId);
        flowObjectiveService.filter(deviceId,
                                    Collections.singletonList(fob.add()));

    }
}
```

Flow Objective Summary

- *Flow Objective Service:* **Abstraction** for applications to be **pipeline unaware** while **benefiting** from scalable, multi-table architectures
- Aims to make it **simple** to write apps
- First attempt at achieving **interoperability** between OF 1.3 implementations

Building Network Applications

- Each application requires complex path computation and rule installation engines and state machines
- Inconsistent behavior in the face of failures
    - Failures may be handled in different ways (or not at all)
- Bugs need to fixed in multiple places (applications)
- Expensive to upgrade/refactor behavior across all applications; e.g.
    - Improve performance
    - Support new types of devices
    - Implement better algorithms
- Difficult or impossible to resolve conflicts with other applications

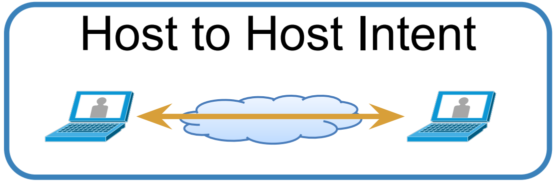
Intent Framework

- Provides **high-level**, **network-centric** interface that focuses on ***what*** should be done rather than ***how*** it is specifically programmed
- Abstracts unnecessary network complexity from applications
- Maintains requested semantics as network changes
- High availability, scalability and high performance

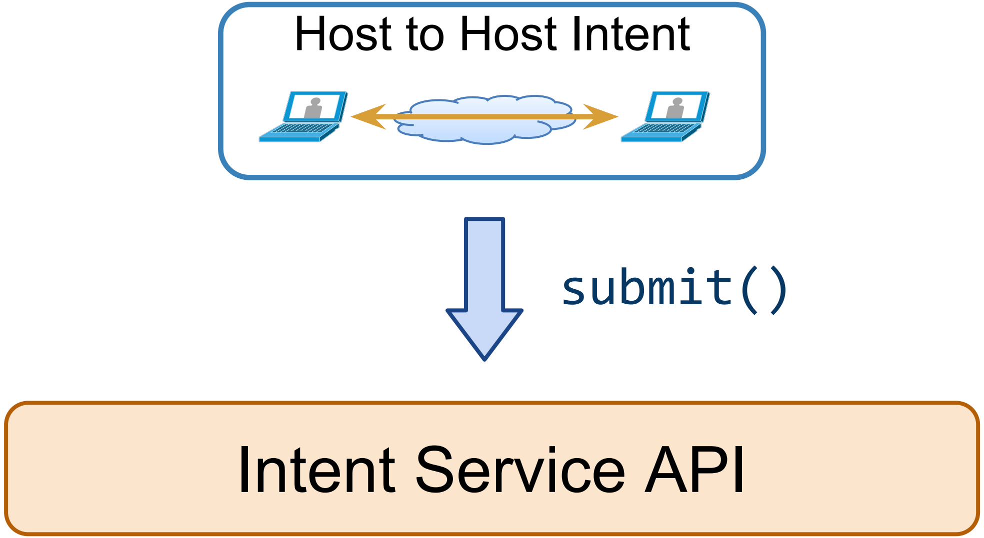
Example Applications

- SDN-IP Peering
  - Connect internal BGP software daemon to external BGP routers
  - Install learned routes to forward IP traffic to appropriate egress point
- Multi-level (IP / Optical) Provisioning
  - Provision optical paths/tunnels with constraints
- Content Acquisition / Video Streaming (DirecTV)
  - Establish multicast forwarding from a sender to set of receivers
- Virtual Network Gateway (vBNG)
  - Provide connectivity between a private host and the Internet
- Bandwidth Calendaring
  - Establish tunnels with bandwidth guarantees between two points at a given time

Intent Example

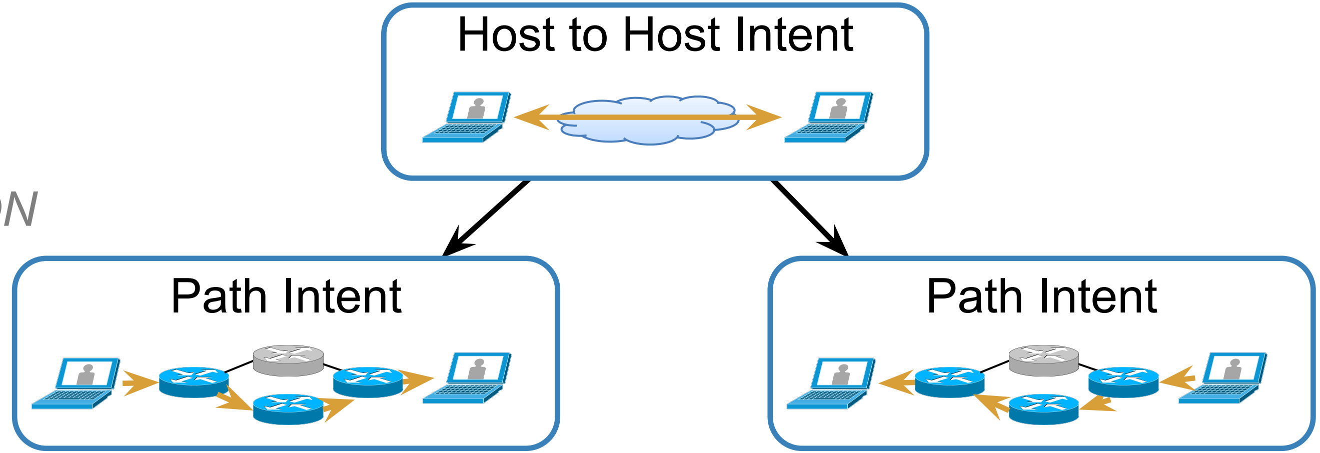


Intent Example

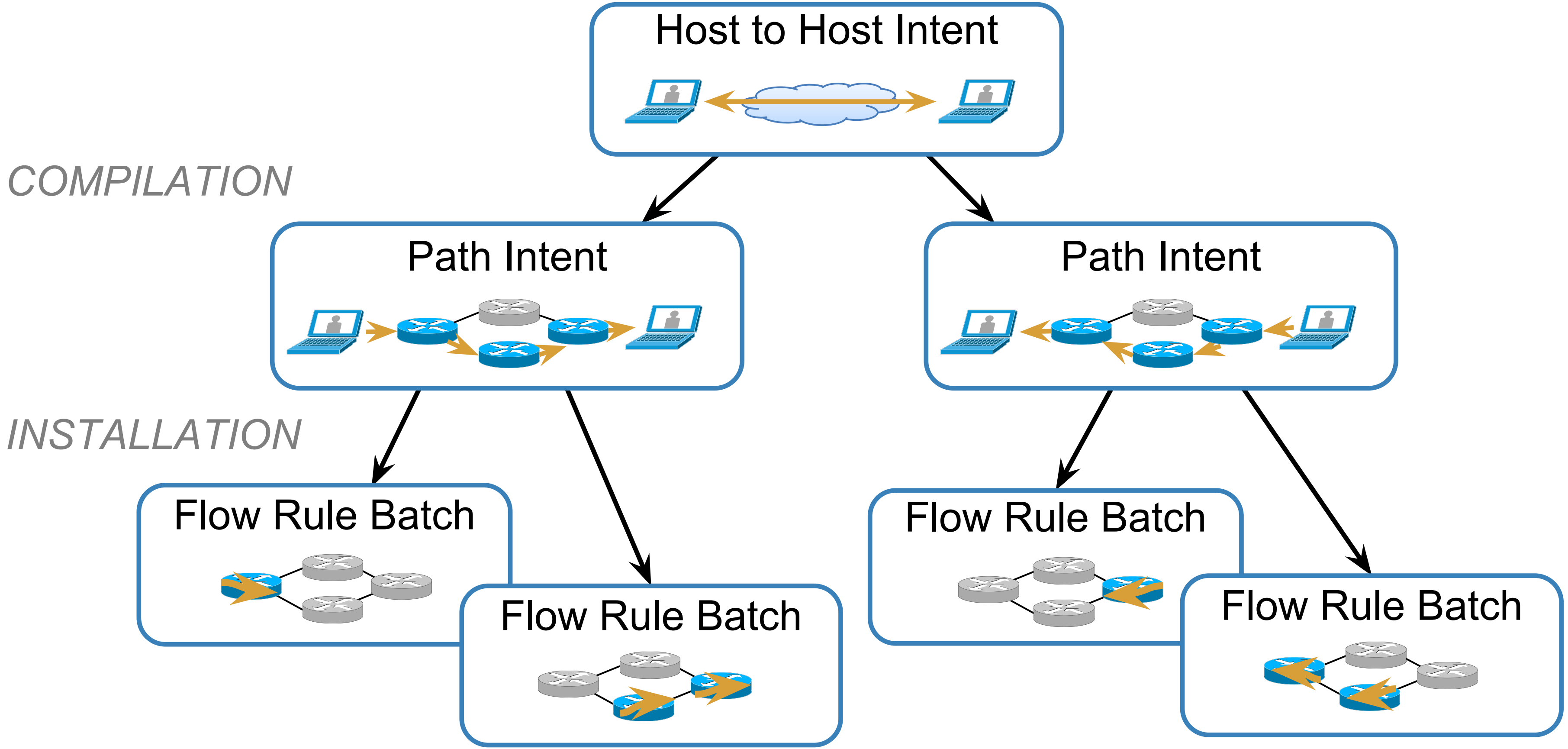


Intent Example

COMPILATION



Intent Example



Intent Framework Summary

- Intents are a ***network-centric programming abstraction*** that **reduce application complexity.**

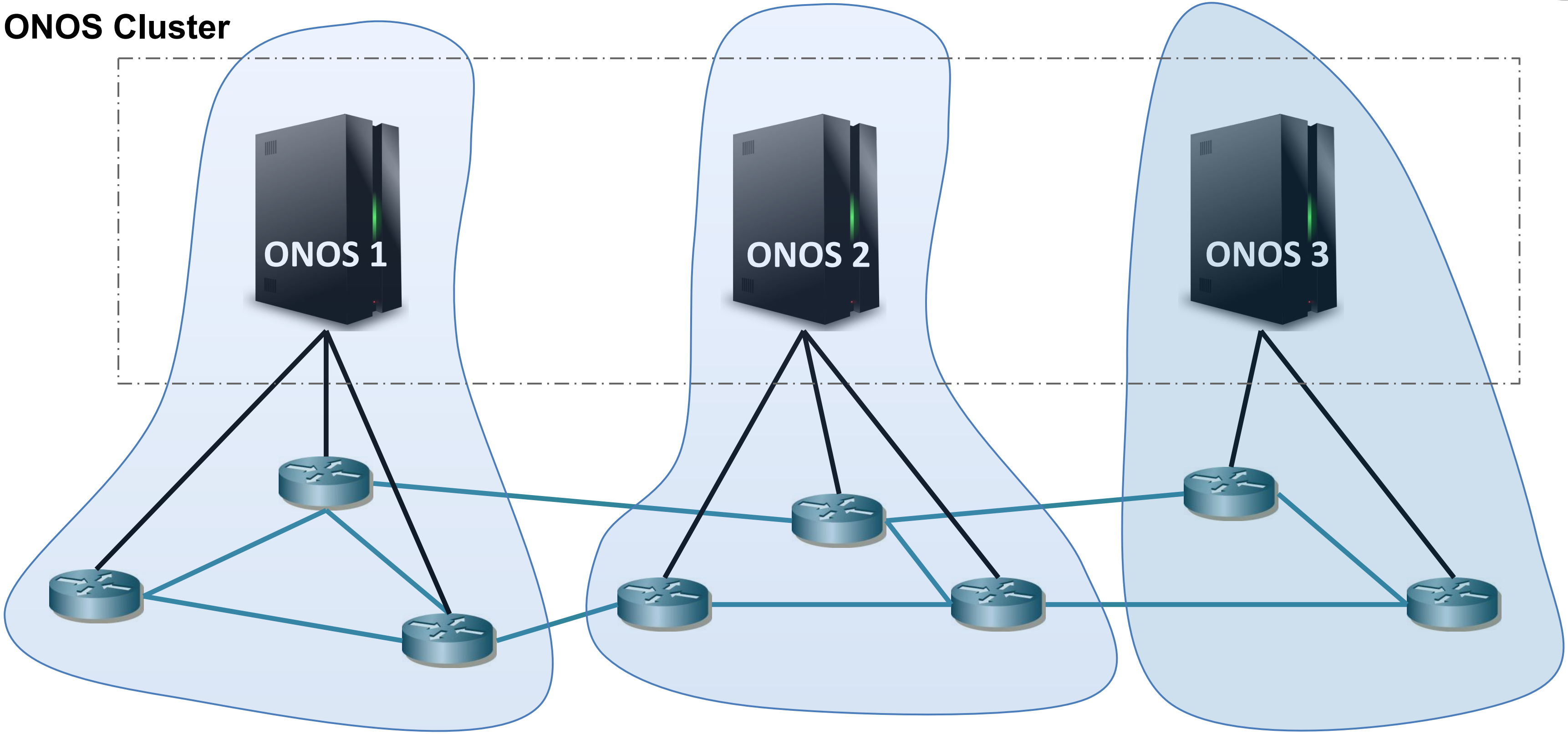
- Intents provide **device-agnostic behavior** with **persistency** and **high performance** across network failures.

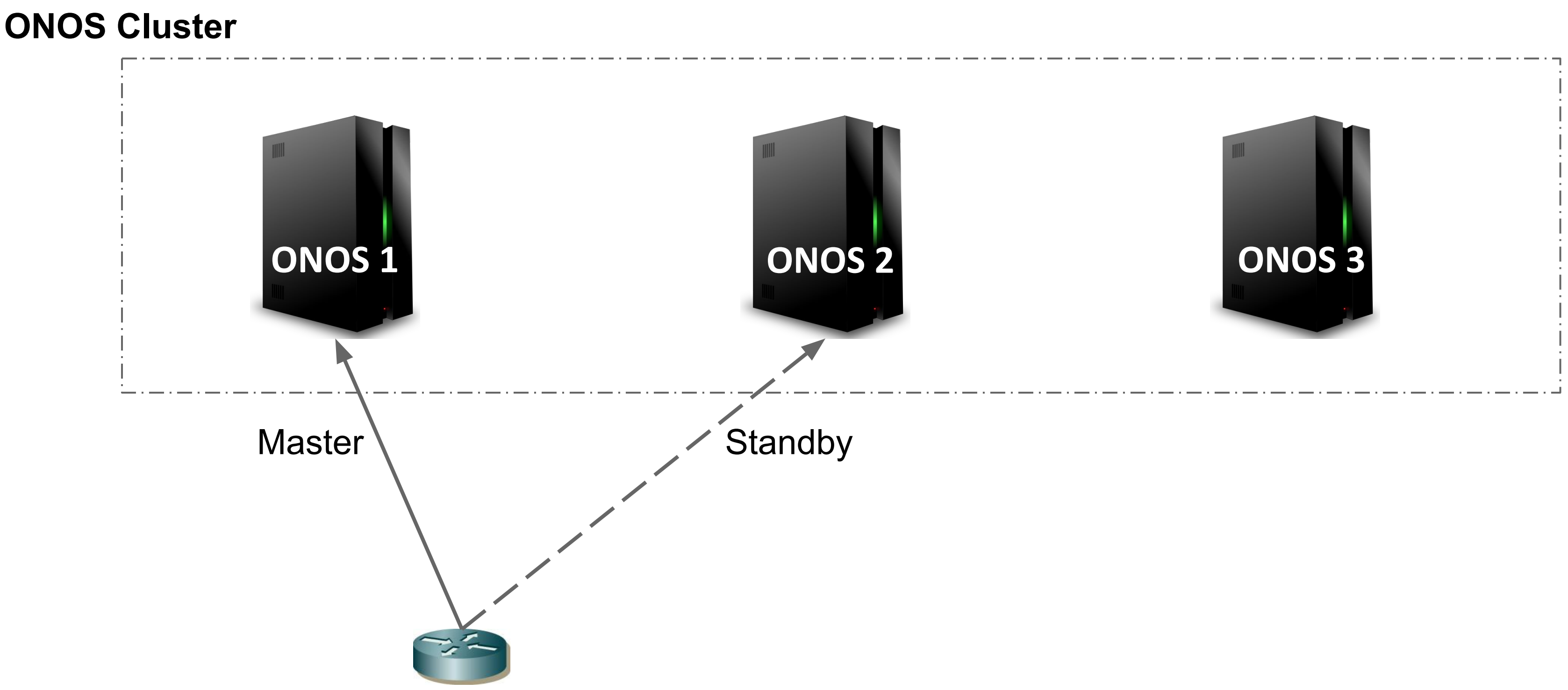
- Intent framework has moved from prototype to **production** deployments.

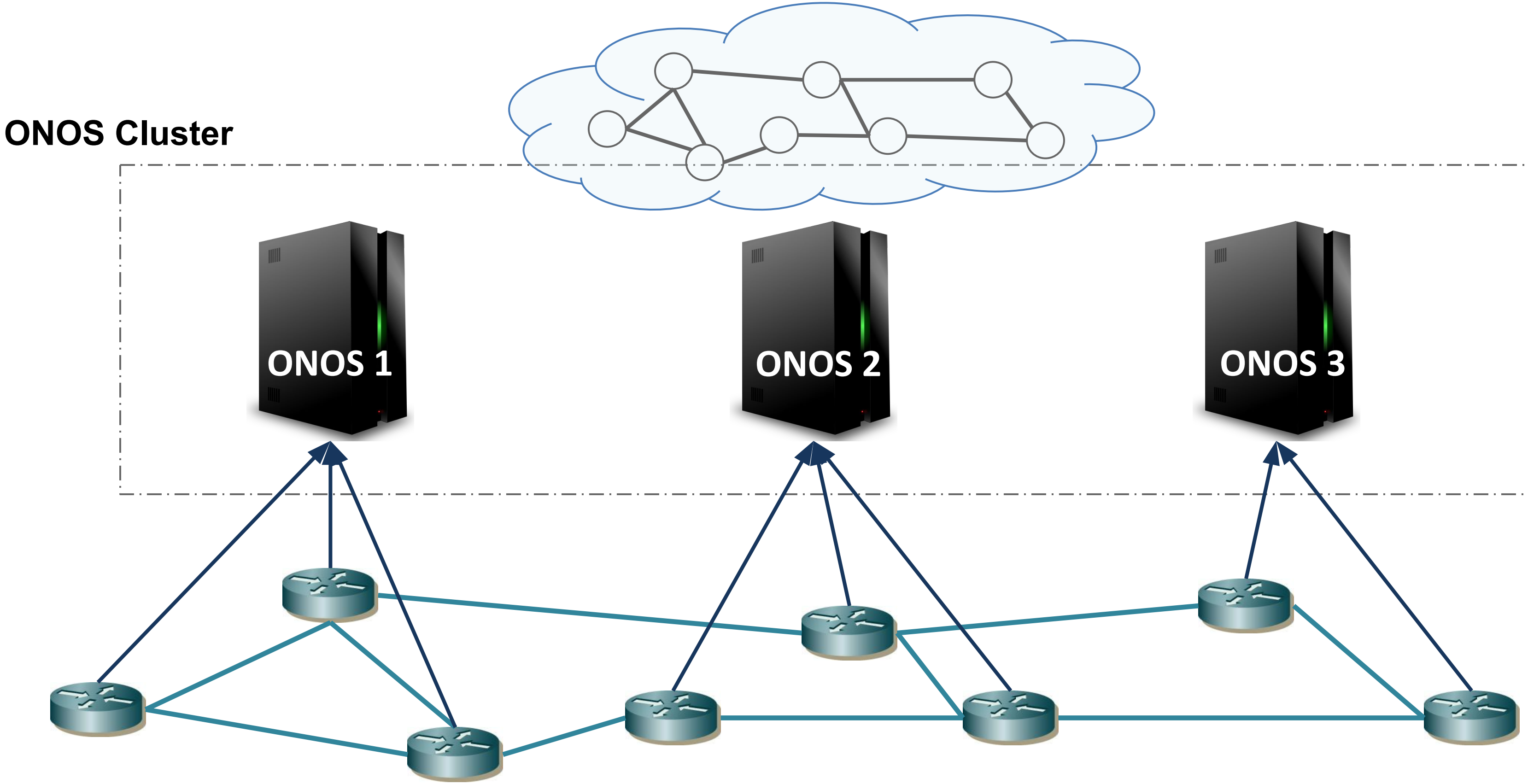
ONOS Distributed Architecture

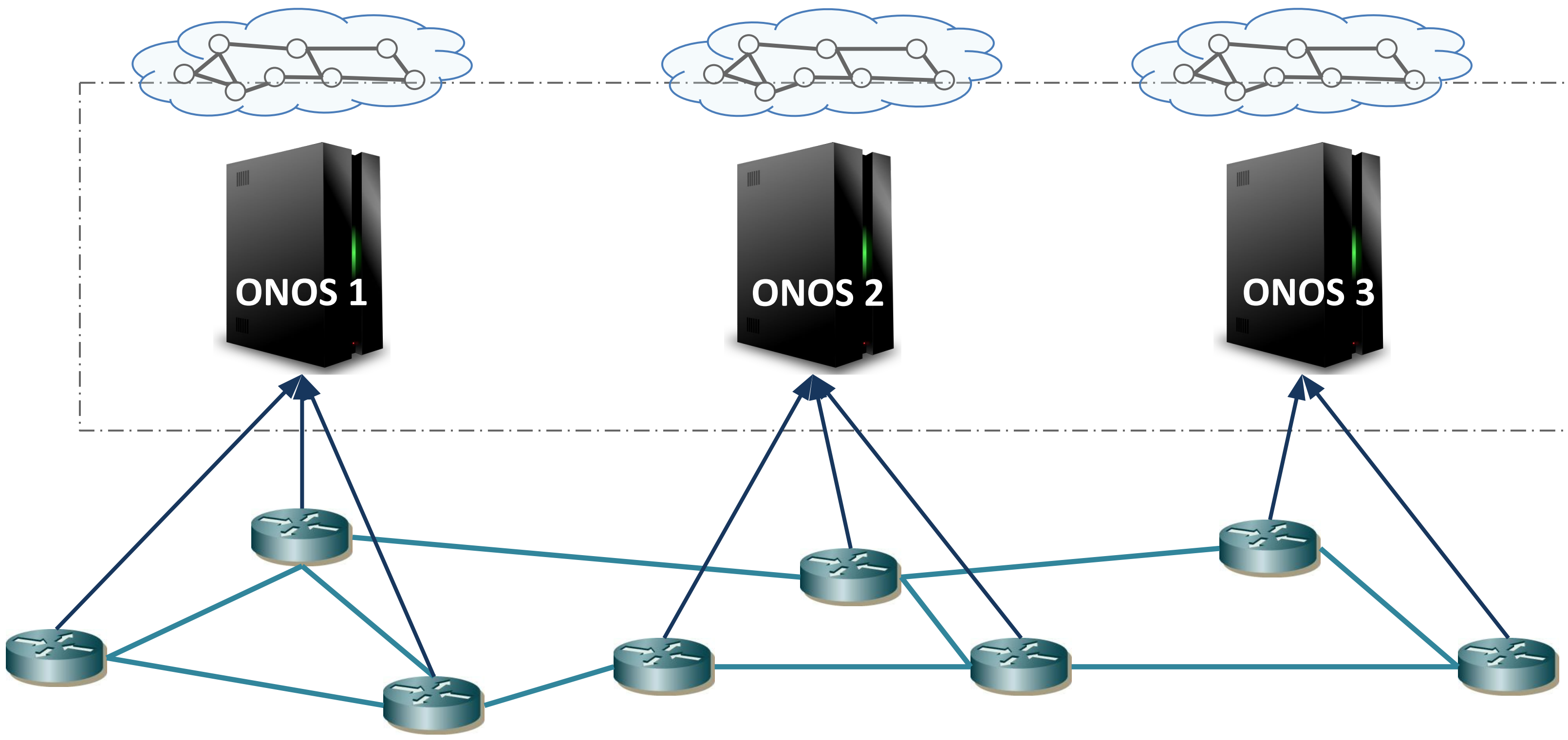
- Distributed
  - Set up as a cluster of instances
- Symmetric
  - Each instance runs identical software and configuration
- Fault-tolerant
  - Cluster remains operational in the face of node failures
- Location Transparent
  - A client can interact with any instance. The cluster presents the abstraction of a single logical instance
- Dynamic *(in progress)*
  - The cluster can be scaled up/down to meet usage demands

ONOS Cluster

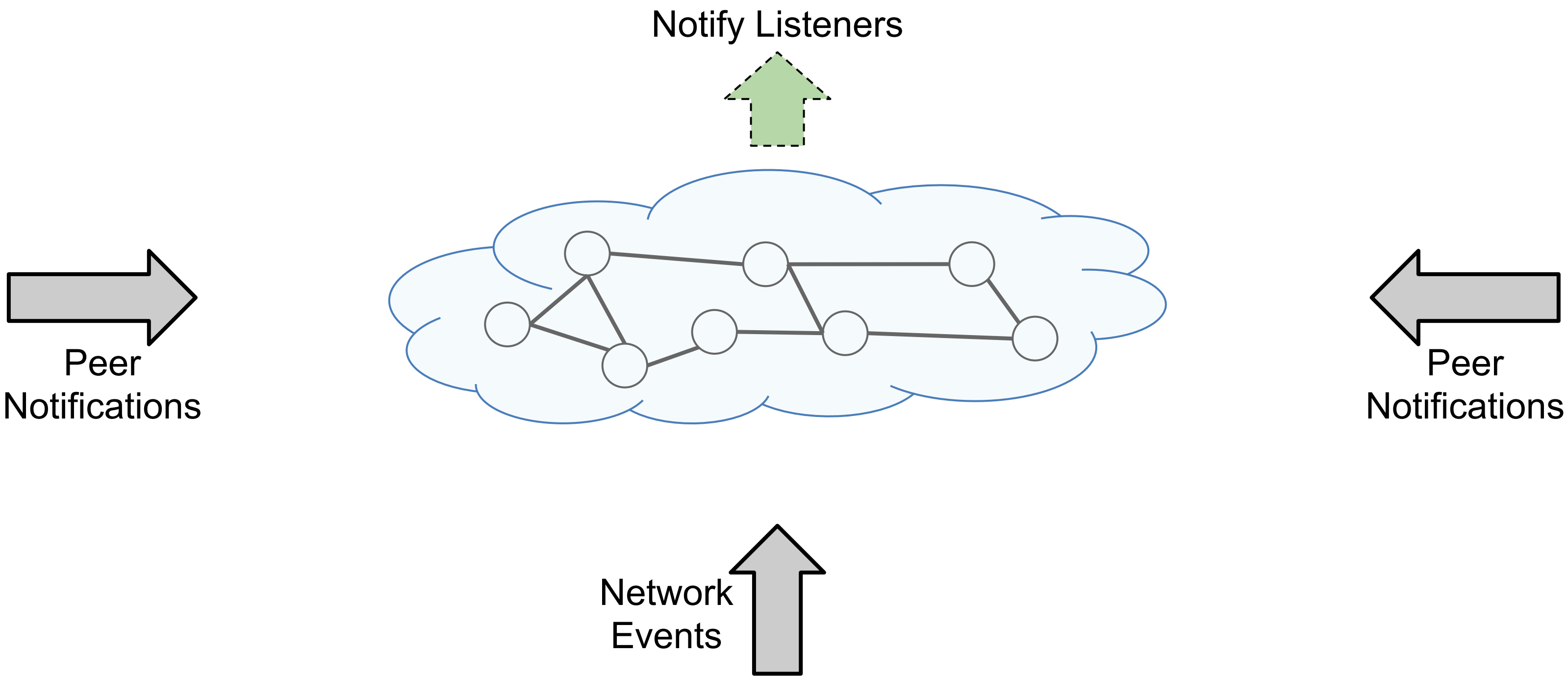



ONOS Cluster
ONOS 1
ONOS 2
ONOS 3
Master
Standby


ONOS Cluster
ONOS 1
ONOS 2
ONOS 3


ONOS 1
ONOS 2
ONOS 3

Topology state inside a Node



Network Events and Ordering

Network Events are timestamped with *(t , s)*

t → mastership term number

s → sequence number in term

Series of timestamps for port X: **… (4, 4) (4, 5) (5, 1) (5, 2) …**

↑

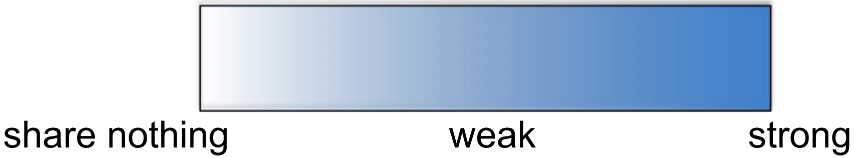
mastership term boundary

Network Topology State

- Eventually Consistent: Reads are **monotonically consistent**
- Low overhead reads and writes
  - 2-3 ms latency for reacting to network events
- Gossip based Anti-Entropy protocol fixes divergent copies
- Generalized as `EventuallyConsistentMap<K, V>`

State Management in ONOS

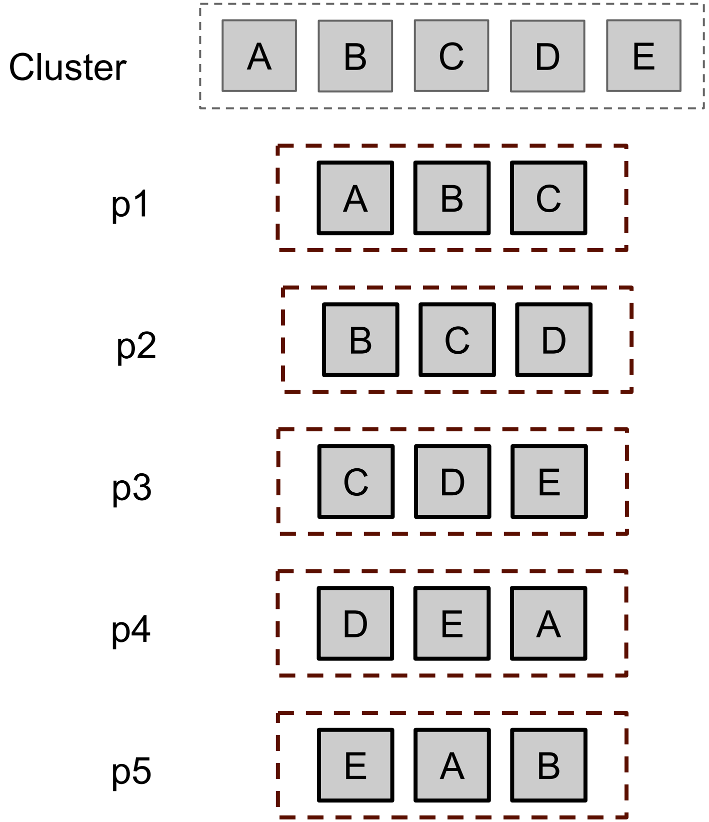
- Core platform feature
- Typically one of hardest pieces to get right and it is better to solve it once
- Better if applications can focus on business logic
- ONOS exposes a set of primitives to cater to different use cases
- Primitives span the consistency continuum



ONOS Distributed Primitives

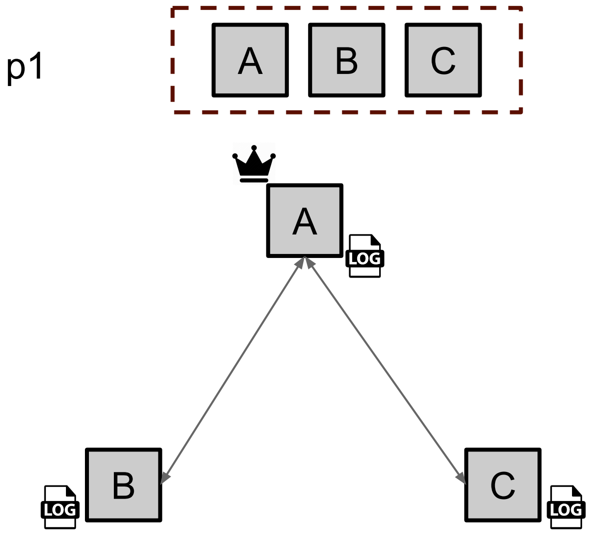
- **EventuallyConsistentMap<K, V>**
  - Map abstraction with eventual consistency guarantee
- **ConsistentMap<K, V>**
  - Map abstraction with strong linearizable consistency
- **LeadershipService**
  - Distributed Locking primitive
- **DistributedQueue<E>**
  - Distributed FIFO queue with long poll support
- **DistributedSet<E>**
  - Distributed collection of unique elements
- **AtomicCounter**
  - Distributed version of Java AtomicLong
- **AtomicValue<V>**
  - Distributed version of Java AtomicReference

Behind the scenes...



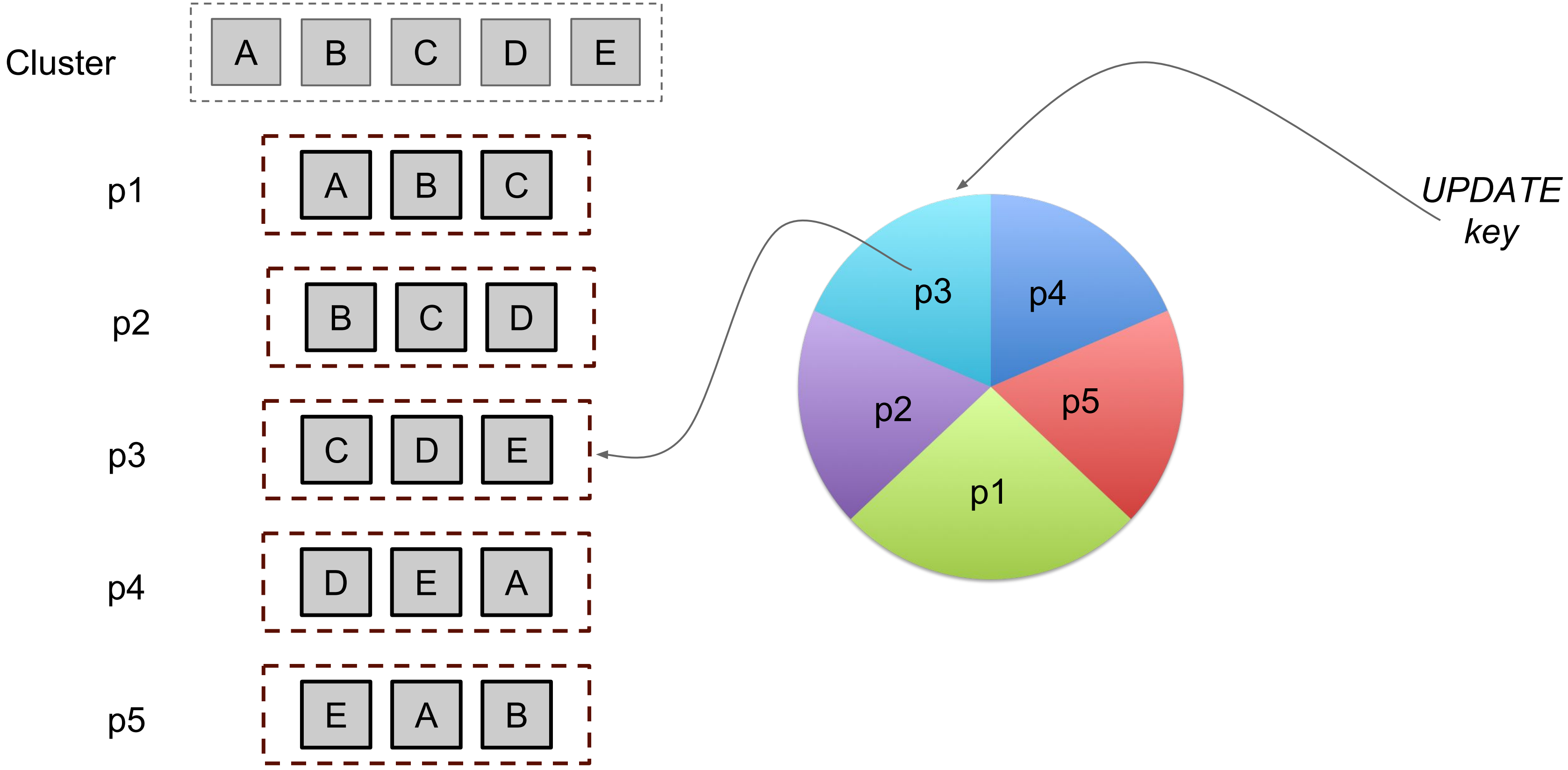
Data is partitioned into Replica Sets

Inside a Replica Set

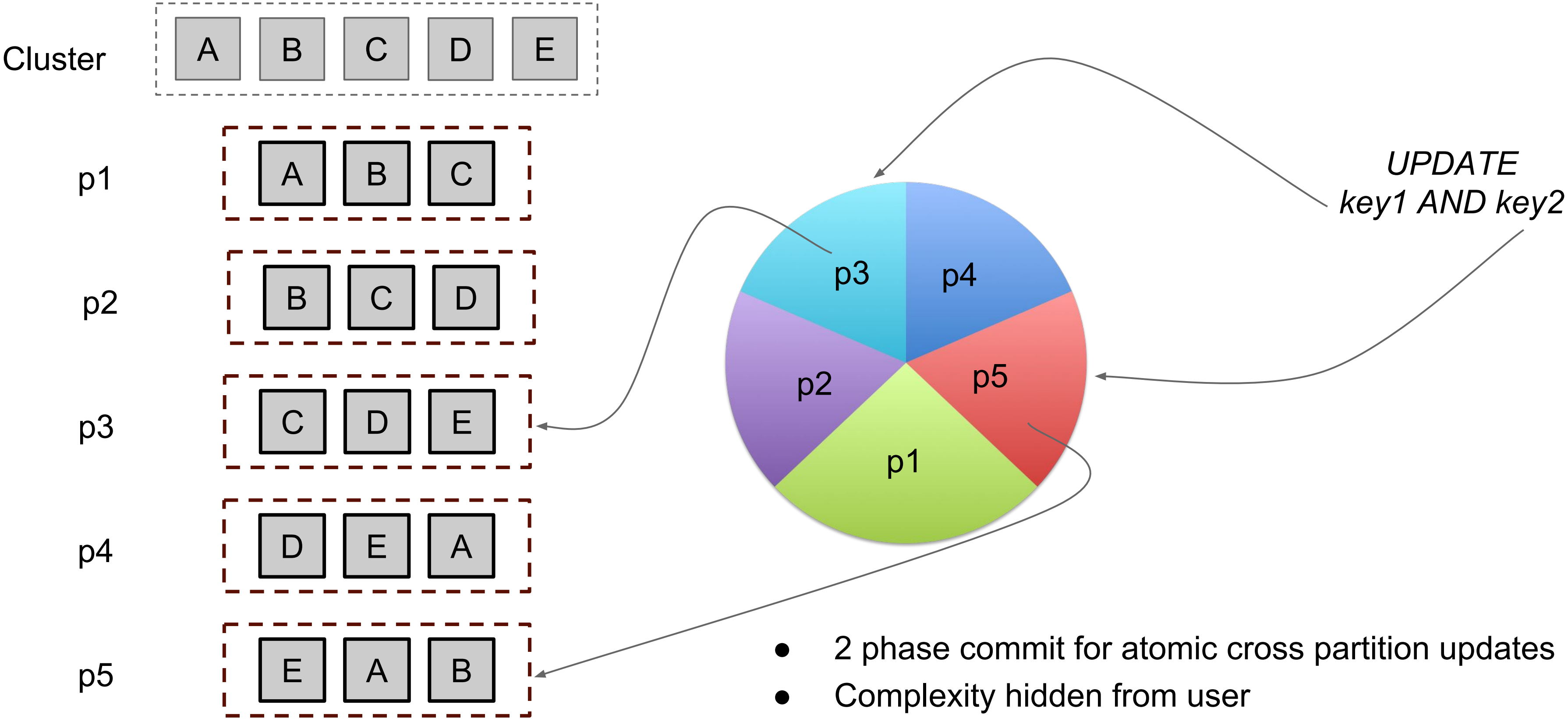


Raft consensus is used to maintain a Replicated State Machine

Data placement



Transactional Updates



- 2 phase commit for atomic cross partition updates
- Complexity hidden from user

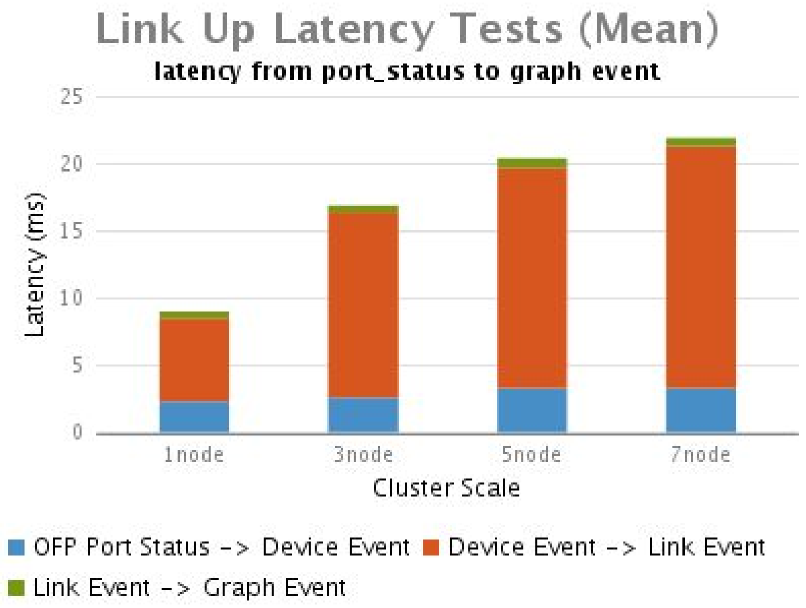
Dynamic Clustering

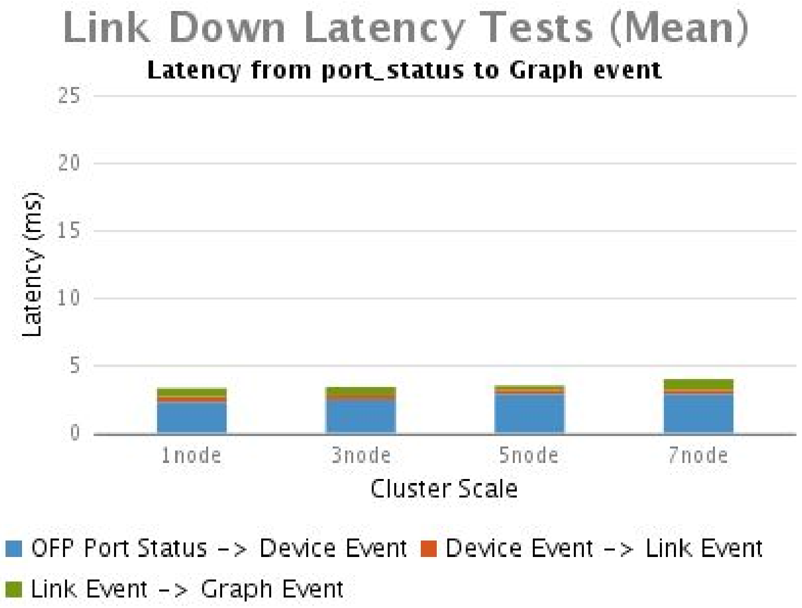
- Ability to grow/shrink a cluster to suit usage demands
- Extract cluster metadata to a separate logical store
- Reshuffle data and control responsibilities to ensure fault-tolerance and load balance

Performance Metrics

- Device & link sensing latency
  - measure how fast can controller react to environment changes, such as switch or port down to rebuild the network graph and notify apps
- Flow rule operations throughput
  - measure how many flow rule operations can be issued against the controller and characterize relationship of throughput with cluster size
- Intent operations throughput
  - measure how many intent operations can be issued against controller cluster and characterize relationship of throughput with cluster size
- Intent operations latency
  - measure how fast can the controller react to environment changes and reprovision intents on the data-plane and characterize scalability

Link Up/Down Latency

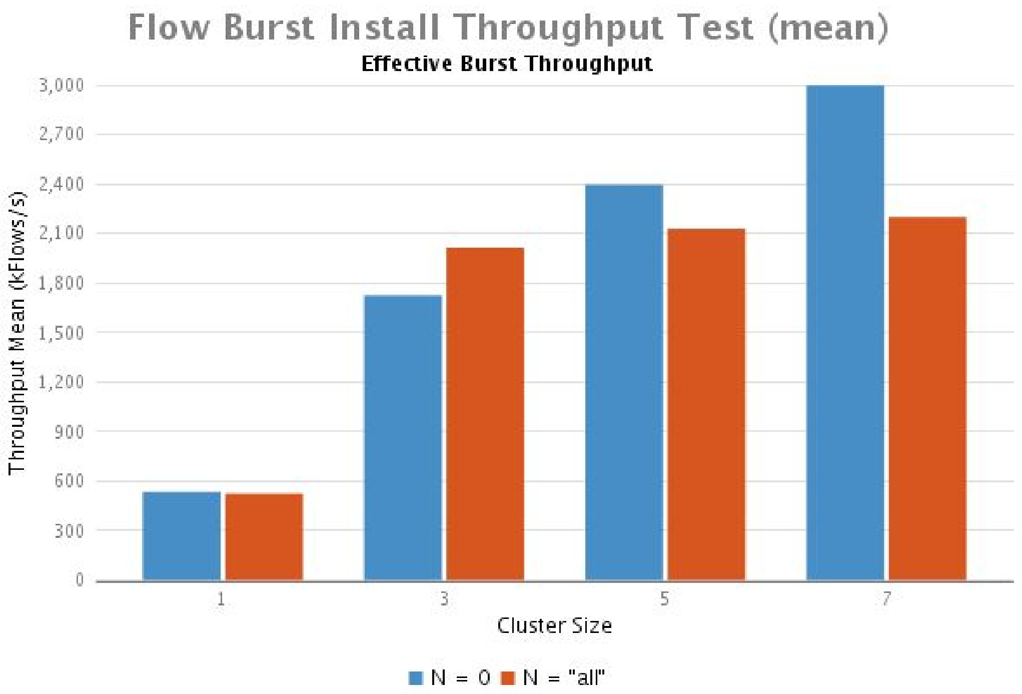




- Since we use LLDP & BDDP to discover links, it takes longer to discover a link coming up than going down

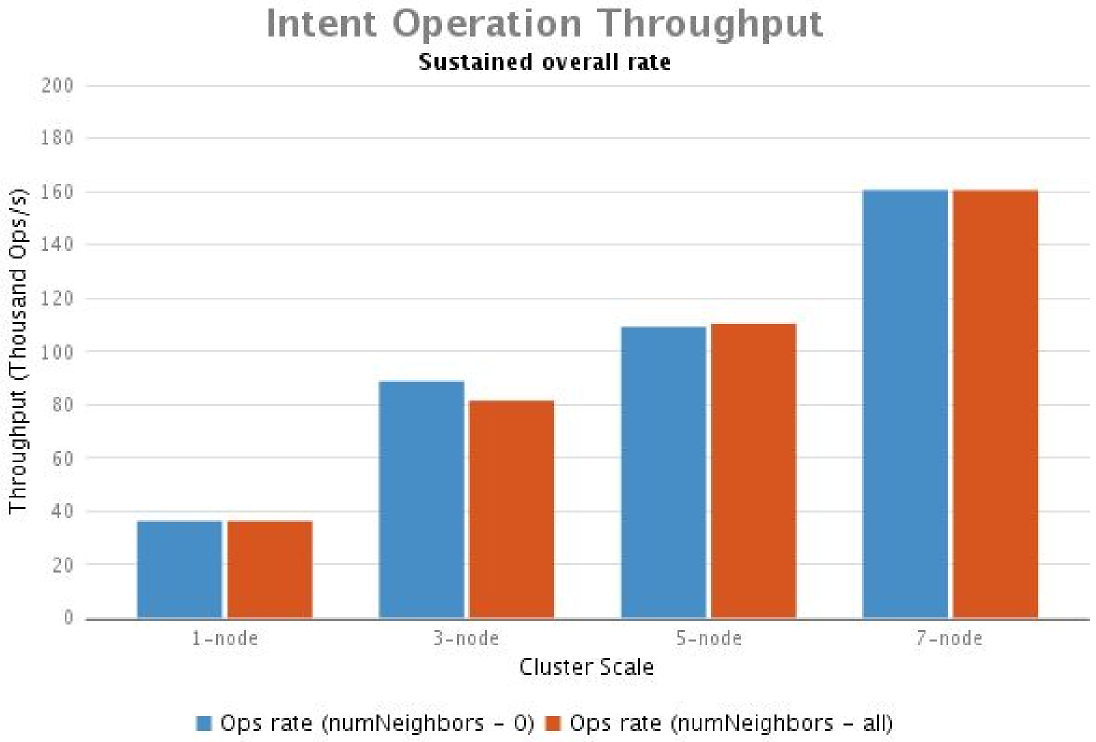
- Port down event trigger immediate teardown of the link.

Flow Throughput results



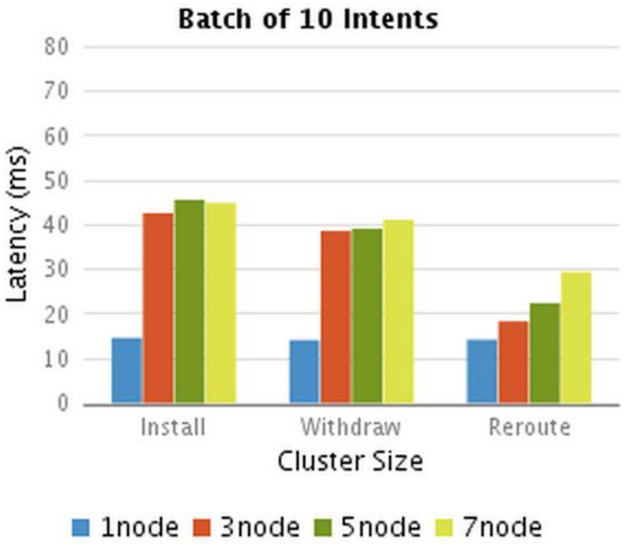
- Single instance can install over 500K flows per second
- ONOS can handle 3M local and 2M non local flow installations
- With 1-3 ONOS instances, the flow setup rate remains constant no matter how many neighbours are involved
- With more than 3 instances injecting load the flow performance drops off due to extra coordination required.

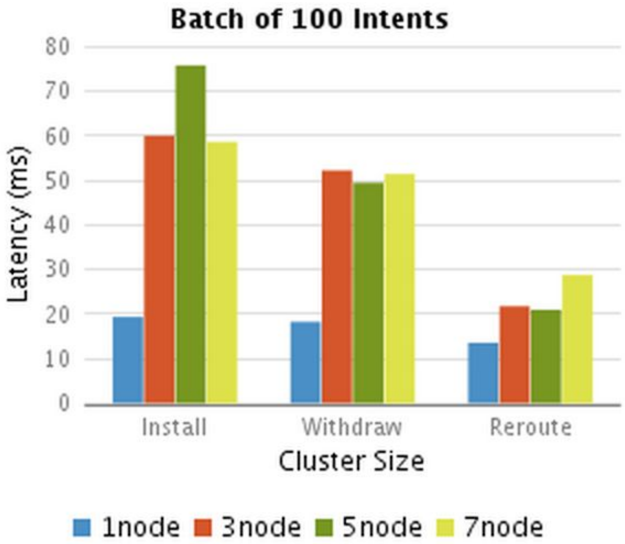
Intent Throughput Results



- Processing clearly scales as cluster size increases

Intent Latency Results





- Less than 100ms to install or withdraw a batch of intents
- Less than 50ms to process and react to network events
  - Slightly faster because intent objects are already replicated

onos

Open Network Operating System

Software Defined Transformation of Service Provider Networks

Join the journey @ onosproject.org