ONOS Developer Workshop

for ONOS 1.2.0 (Cardinal)

Open Networking Summit
June 15, 2015

Developer Workshop Sessions

- ONOS Fundamentals
  - architecture, checking out, building & running ONOS
- ONOS Distributed Core
  - OSGi, anatomy of a core subsystem, distributed stores & primitives
- ONOS App Development
  - background, Maven archetypes, app deployment on ONOS cluster
- ONOS Interfaces
  - component config service, flow objective service, intent service
  - CLI, REST and GUI overview
- Wrap-up

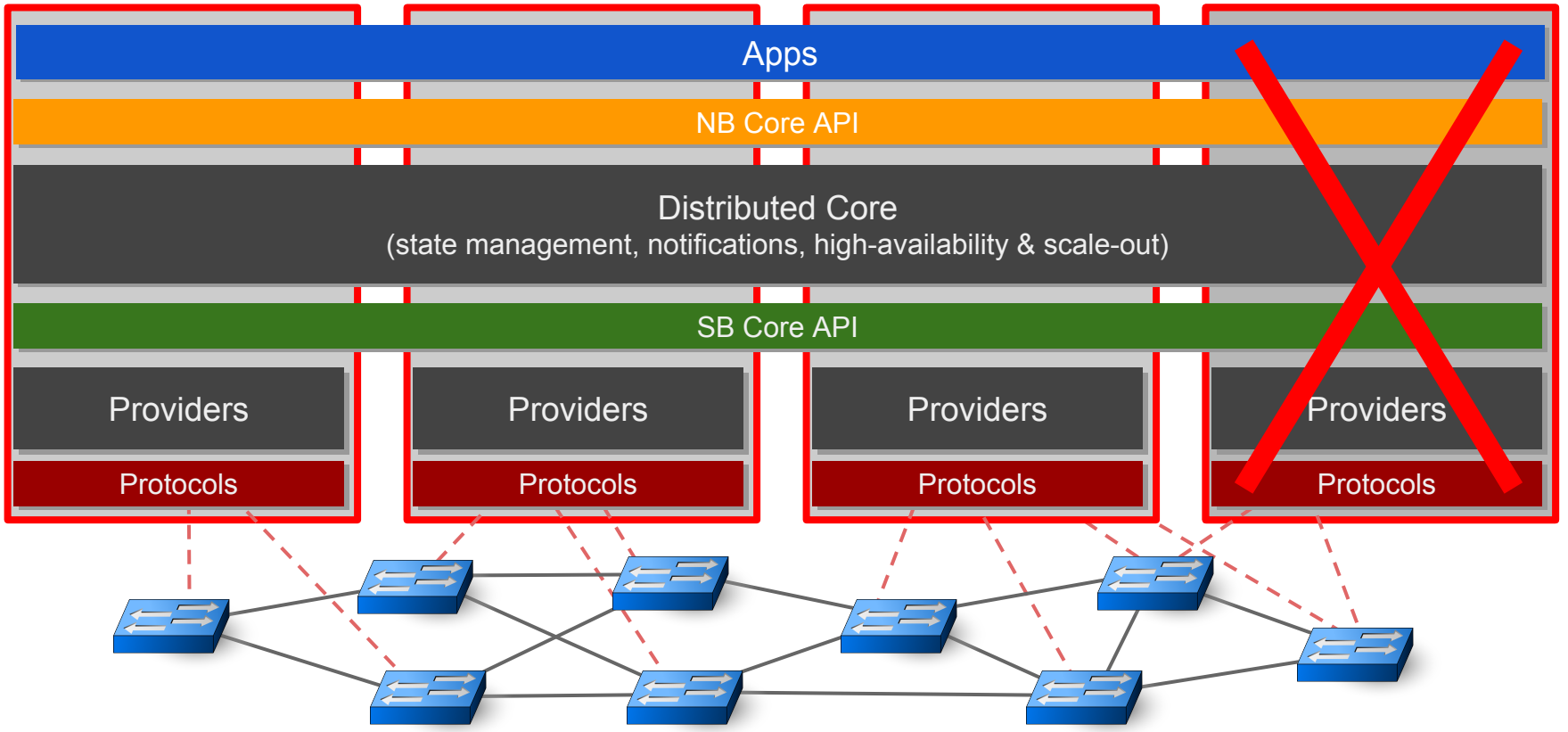
ONOS Fundamentals 8:30-10:00

- ONOS Overview (~30)
  - tenets, architecture, tiers, subsystems
  - general subsystem structure
- Getting started with ONOS (~30)
  - check-out code, environment customization and build
  - describe Maven, hierarchical POM structure
  - run ONOS locally on developer machine
  - CLI & GUI demo with a simple mininet topology (torus,8,8,2)
- Code organization (~30)
  - importing ONOS source into IDE (demo with IntelliJ)
  - core (api, net, stores), web (gui, rest), providers (openflow, netconf)

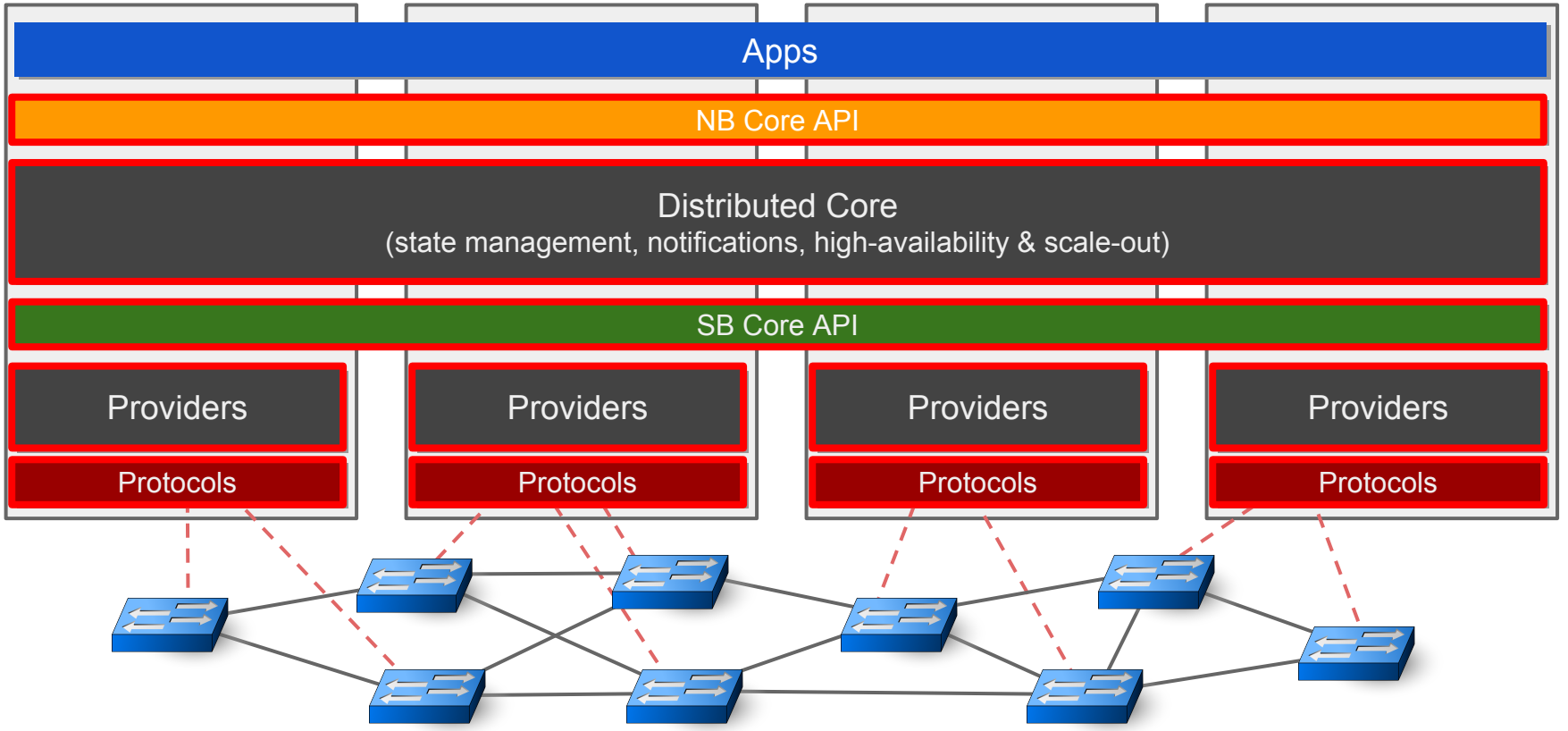
Architectural Tenets

- 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

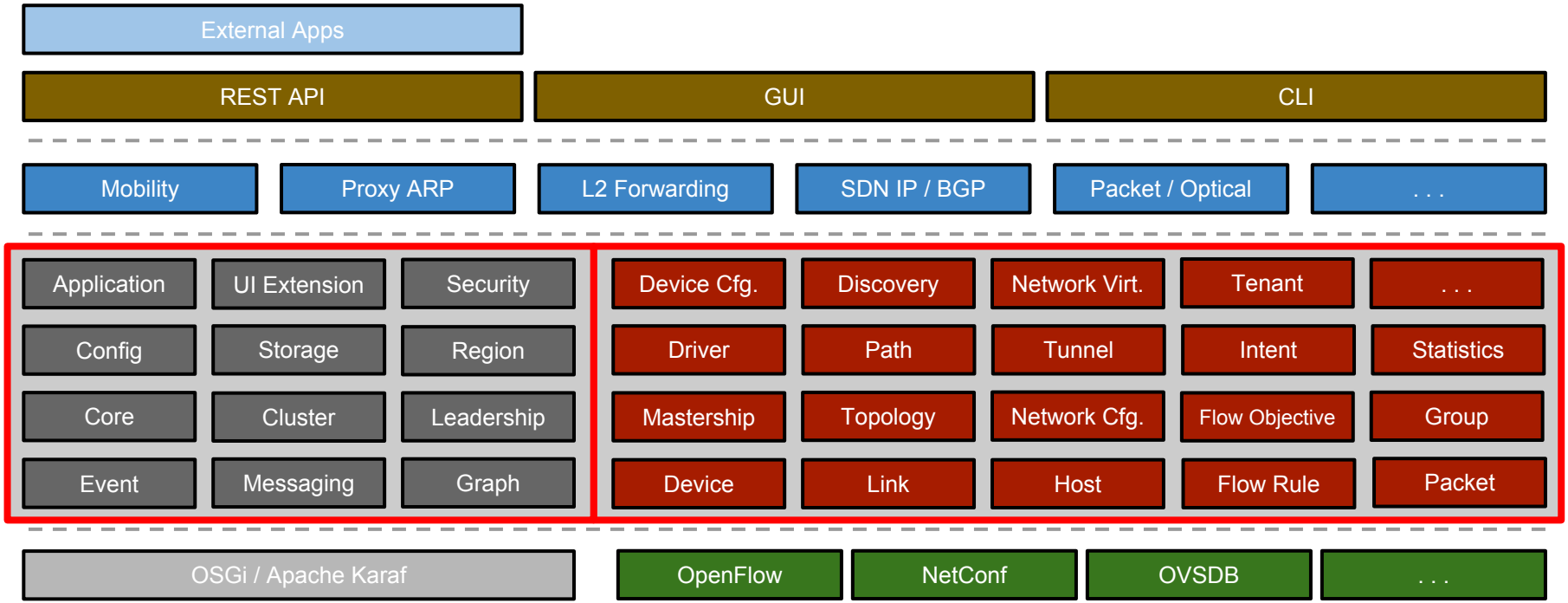
ONOS Distributed Architecture



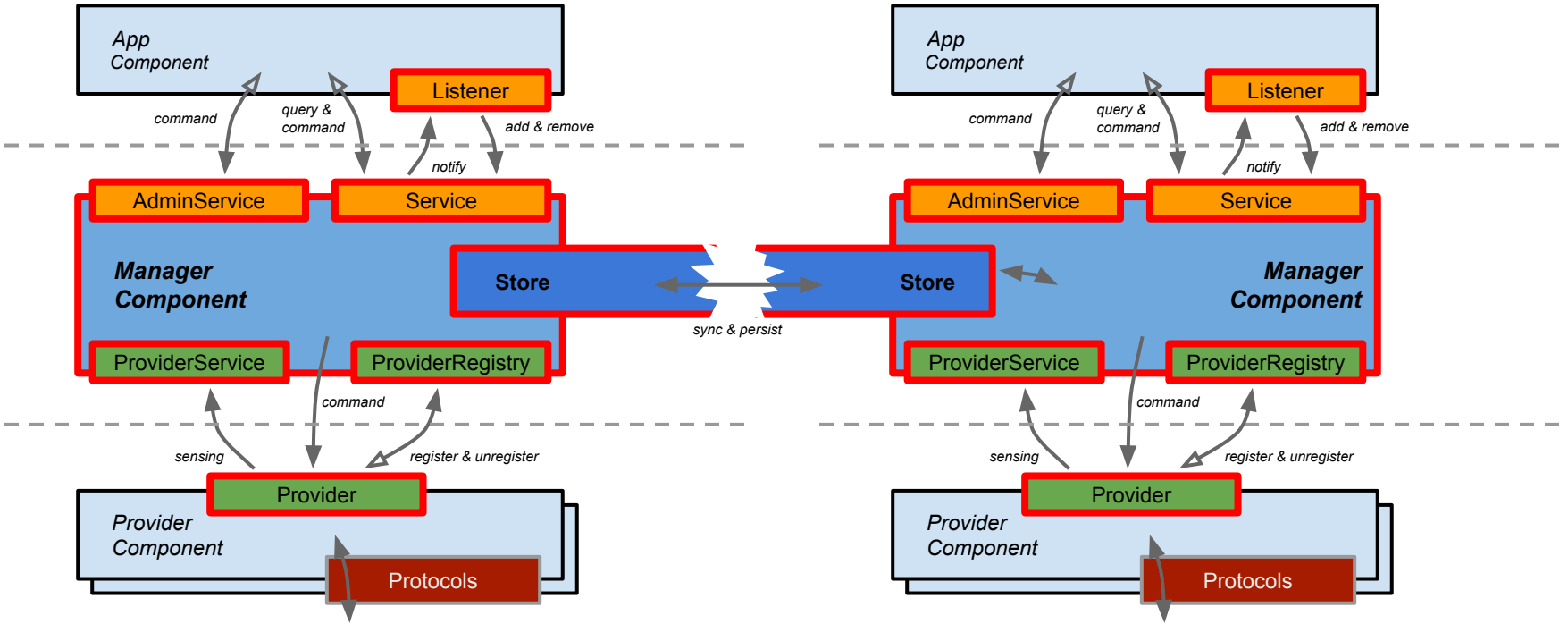
ONOS Distributed Architecture



ONOS Core Subsystems



ONOS Core Subsystem Structure



ONOS Fundamentals 8:30-10:00

- ONOS Overview (~30)
  - tenets, architecture, tiers, subsystems
  - general subsystem structure
- Getting started with ONOS (~30)
  - check-out code, environment customization and build
  - describe Maven, hierarchical POM structure
  - run ONOS locally on developer machine
  - CLI & GUI demo with a simple mininet topology (torus,8,8,2)
- Code organization (~30)
  - importing ONOS source into IDE (demo with IntelliJ)
  - core (api, net, stores), web (gui, rest), providers (openflow, netconf)

Development Pre-Requisites

- Bash (or similar shell)
- Oracle Java Development Kit 8+
  `http://www.oracle.com/technetwork/java/javase/downloads`
- Apache Maven 3.3.x+
  `https://maven.apache.org/download.html`
- Apache Karaf 3.0.3+ (for local development)
  `http://karaf.apache.org/index/community/download.html`
- IDE of your choice (IntelliJ, Eclipse, …)

Getting started with ONOS

- Check-out ONOS code via git
- Tailor bash environment for ONOS development
- Build ONOS via Maven
- Run ONOS locally on developer machine
- Activate pre-installed reactive forwarding application
- Test ONOS via a mininet simulated network
- Explore ONOS GUI & CLI

Check-out ONOS code

- Open a new terminal shell and clone ONOS repo:

```
$ git clone https://gerrit.onosproject.org/onos
```

Tailor shell for ONOS development

- Edit the shell profile:

```
$ vi ~/.bash_profile
```

- Add the following lines at the top:

```
export ONOS_ROOT=~/onos
source $ONOS_ROOT/tools/dev/bash_profile
```

- Source the modified shell profile

```
$ source ~/.bash_profile
```

Build ONOS via Maven

- Change directory to your ONOS root and invoke Maven

```
$ cd ~/onos
$ mvn clean install
```

- Alternatively, use `onos-build` command or `ob` alias from anywhere

```
$ onos-build
```

Run ONOS locally

- Use `onos-karaf` command or `ok` alias to setup and run ONOS via Apache Karaf

```
$ onos-karaf clean
```

- Use ONOS shell commands to explore

```
onos> summary
onos> nodes
onos> apps -s
```

Activate reactive forwarding app

- Use app command to activate a pre-installed app:

```
onos> app activate org.onosproject.fwd
```

Test ONOS via mininet

- Start ONOS gui via `onos-gui` command or `gui` alias:

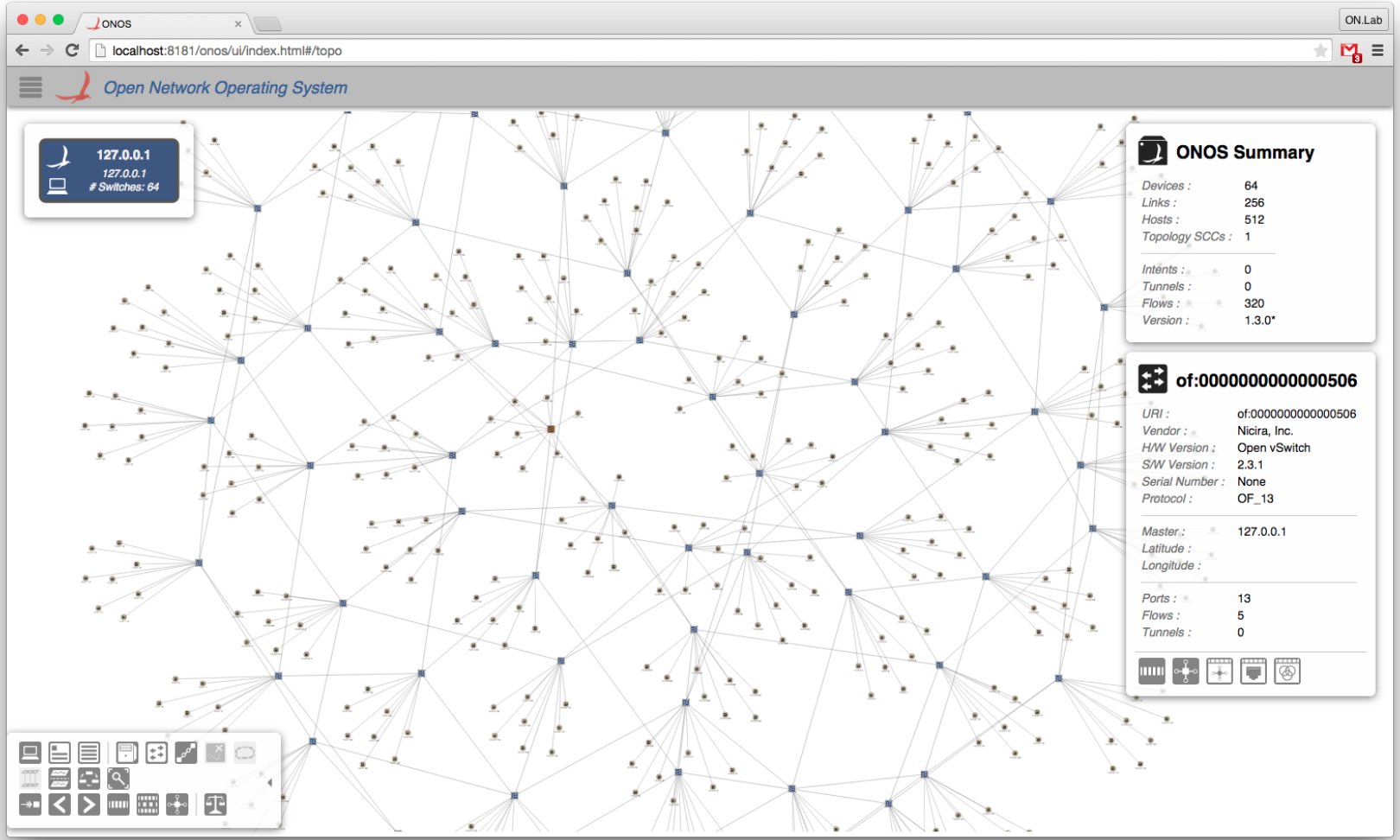
```
$ onos-gui
```

- Start 8x8 torus topology with 2 hosts per switch:

```
$ sudo mn --topo torus,8,8,2 \
    --controller remote,ip=192.168.56.1
```

- Ping all hosts in mininet

```
mininet> pingall
```


ONOS
ON.Lab
localhost:8181/onos/ui/index.html#/topo
Open Network Operating System
127.0.0.1
127.0.0.1
Switches: 64
ONOS Summary
Devices : 64
Links : 256
Hosts : 512
Topology SCCs : 1
Intents : 0
Tunnels : 0
Flows : 320
Version : 1.3.0*
of:0000000000000506
URI : of:0000000000000506
Vendor : Nicira, Inc.
H/W Version : Open vSwitch
S/W Version : 2.3.1
Serial Number : None
Protocol : OF_13
Master : 127.0.0.1
Latitude :
Longitude :
Ports : 13
Flows : 5
Tunnels : 0

Explore ONOS GUI & CLI

- Enable host display via tool icon or `H` key
- Explore Devices, Links and Hosts views, etc.

- Explore ONOS CLI via the following shell commands:

```
onos> summary
onos> devices
onos> links
onos> hosts
onos> flows
onos> onos:*
```

ONOS Fundamentals 8:30-10:00

- ONOS Overview (~30)
  - tenets, architecture, tiers, subsystems
  - general subsystem structure
- Getting started with ONOS (~30)
  - check-out code, environment customization and build
  - describe Maven, hierarchical POM structure
  - run ONOS locally on developer machine
  - CLI & GUI demo with a simple mininet topology (torus,8,8,2)
- Code organization (~30)
  - importing ONOS source into IDE (demo with IntelliJ)
  - core (api, net, stores), web (gui, rest), providers (openflow, netconf)

Tour of ONOS code-base

- Import ONOS into IDE
- Tour of ONOS core (api, net, stores, etc.)
- Tour of other areas (apps, providers, web, etc.)

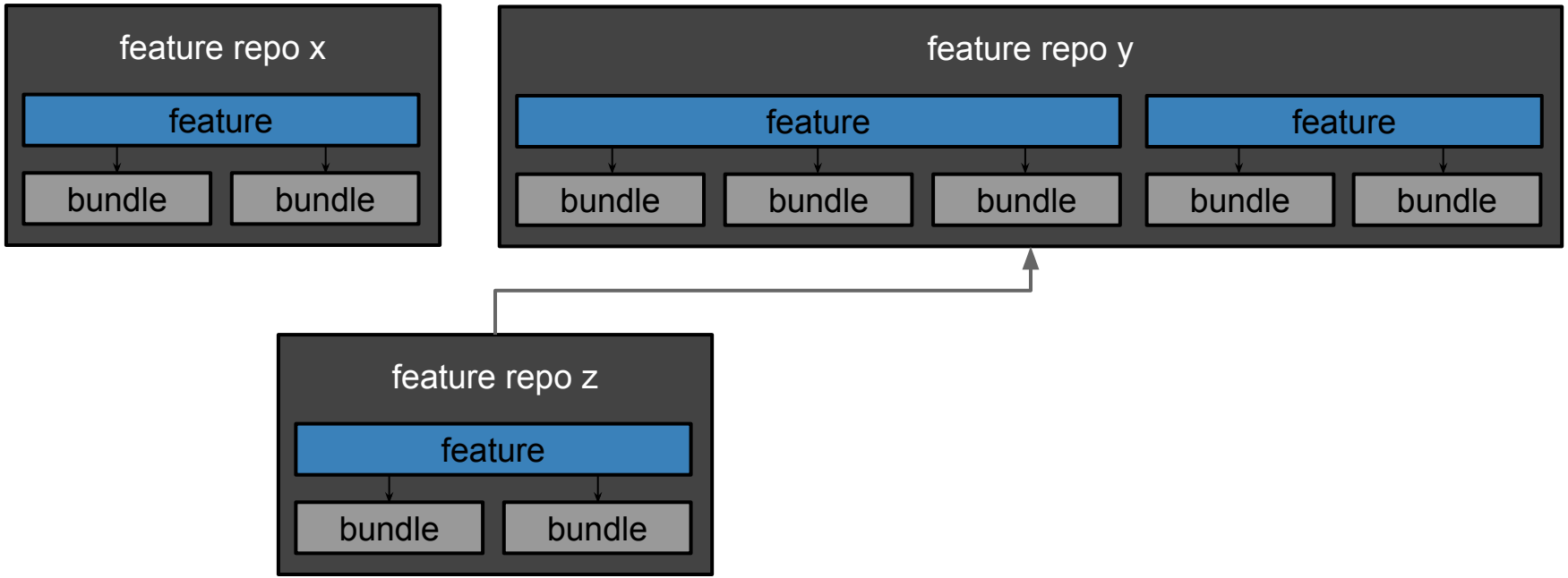
Break

xkcd.com

ONOS Distributed Core 10:30-12:00

- ONOS, OSGi & Apache Karaf (~15)
  - OSGi bundles, Karaf features & SCR components
  - use `bundle:*`, `feature:*`, `scr:*` commands & webconsole
- Anatomy of an ONOS Core Subsystem (~15)
  - use device subsystem as an example
- Distributed stores & primitives (~60)
  - physically distributed & logically centralized
  - deep dive on distributed topology stores (devices, links, hosts)
  - other types of state treatment for other types of state
  - primitives, e.g. eventually consistent/strongly consistent map, leadership, distributed set, distributed queue, atomic counter

OSGi Bundles & Karaf Features



OSGi Bundles & Karaf Features

- OSGi bundles are Java JAR files with an enhanced Manifest
  - bundles have name and version
  - bundles explicitly require/import other Java packages
  - bundles explicit provide/export Java packages for others
- Karaf features are means to install or uninstall a set of bundles as a group
  - features are defined via an XML artifact - a feature repository
  - features reference, but do not deliver the bundle JAR artifacts
- Karaf uses Maven repos as OSGi Bundle Repositories for retrieval of feature and bundle artifacts

Service Component Runtime

- Components are effectively stateful singletons whose life-cycle is controlled by the framework
  - components defined by `OSGI-INF/*.xml` files at run-time
  - ONOS uses `maven-scr-plugin` to convert Java annotations to `OSGI-INF/*.xml` files at compile-time
- Components can provide `@Services` to others
- Components can `@Reference` services from others
- `@Activate`, `@Modified` and `@Deactivate` methods serve as component life-cycle hooks

Bundle & Feature Shell Commands

- Bundle related commands

```
onos> bundle:*
```

- Feature related commands

```
onos> feature:*
```

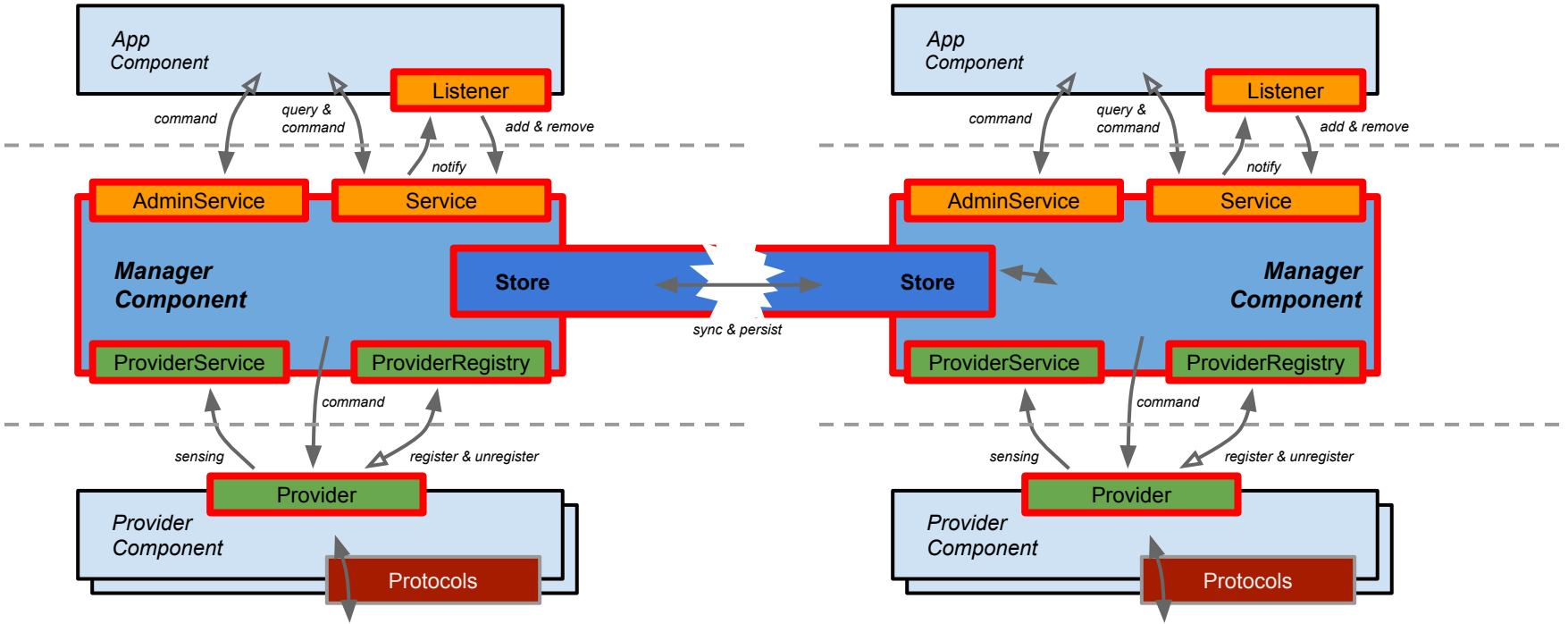
- Service Component Runtime related commands

```
onos> scr:*
```

ONOS Distributed Core 10:30-12:00

- ONOS, OSGi & Apache Karaf (~15)
  - OSGi bundles, Karaf features & SCR components
  - use `bundle:*`, `feature:*`, `scr:*` commands & webconsole
- Anatomy of an ONOS Core Subsystem (~15)
  - use device subsystem as an example
- Distributed stores & primitives (~60)
  - physically distributed & logically centralized
  - deep dive on distributed topology stores (devices, links, hosts)
  - other types of state treatment for other types of state
  - primitives, e.g. eventually consistent/strongly consistent map, leadership, distributed set, distributed queue, atomic counter

Anatomy of an ONOS subsystem



Device Subsystem - Northbound

- `DeviceManager` component exposes `DeviceService` to apps and other services
  - note the `@Component` and `@Service` annotations
- Apps can subscribe to asynchronous `DeviceEvent` notifications via `DeviceListener` mechanism
- `DeviceManager` also exposes `DeviceAdminService` to privileged apps for select administrative actions

Device Subsystem - Southbound

- `DeviceManager` offers `DeviceProviderRegistry` service
- `DeviceProvider` entities can register and acquire a reference to `DeviceProviderService`
- Core uses registered `DeviceProvider` for sending edicts and queries down to the environment in a protocol-agnostic manner
- Provider uses issued `DeviceProviderService` for sending device and port sensory data up to the core in a protocol-agnostic manner

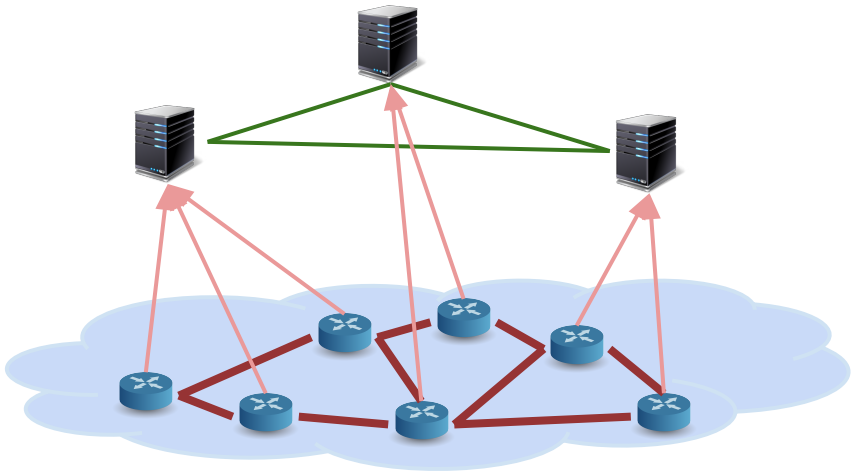
Device Subsystem - East / West

- `DeviceManager` delegates to peer `DeviceStore` all cluster-related tasks
  - manager becomes `DeviceStoreDelegate` to the `DeviceStore`
- `DeviceStore` is responsible for indexing, persisting and synchronizing the global device and port state with all cluster peers
- `DeviceStore` delegates incoming `DeviceEvents` to its `DeviceStoreDelegate` for local actions and for local event dispatching

ONOS Distributed Core 10:30-12:00

- ONOS, OSGi & Apache Karaf (~15)
  - OSGi bundles, Karaf features & SCR components
  - use `bundle:*`, `feature:*`, `scr:*` commands & webconsole
- Anatomy of an ONOS Core Subsystem (~30)
  - use device subsystem as an example
- Distributed stores & primitives (~60)
  - physically distributed & logically centralized
  - deep dive on distributed topology stores (devices, links, hosts)
  - other types of state treatment for other types of state
  - primitives, e.g. eventually consistent/strongly consistent map, leadership, distributed set, distributed queue, atomic counter

Distributed Stores & Primitives



“The unavoidable price of reliability is simplicity.”

— C.A.R. Hoare

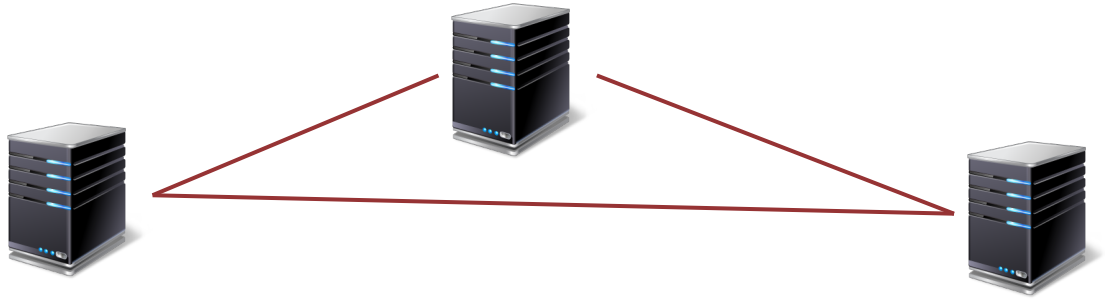
“Simplicity is prerequisite for reliability.”

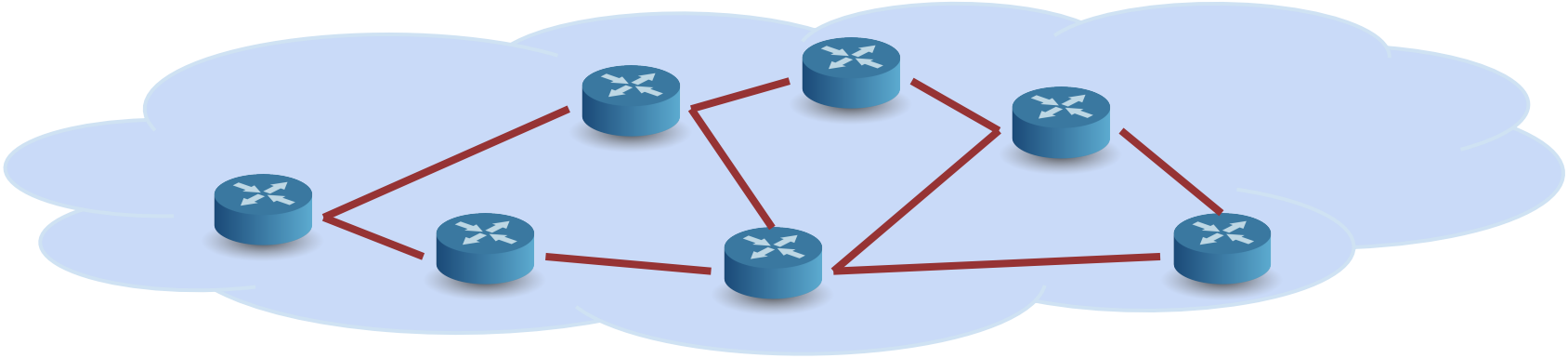
— E. W. Dijkstra

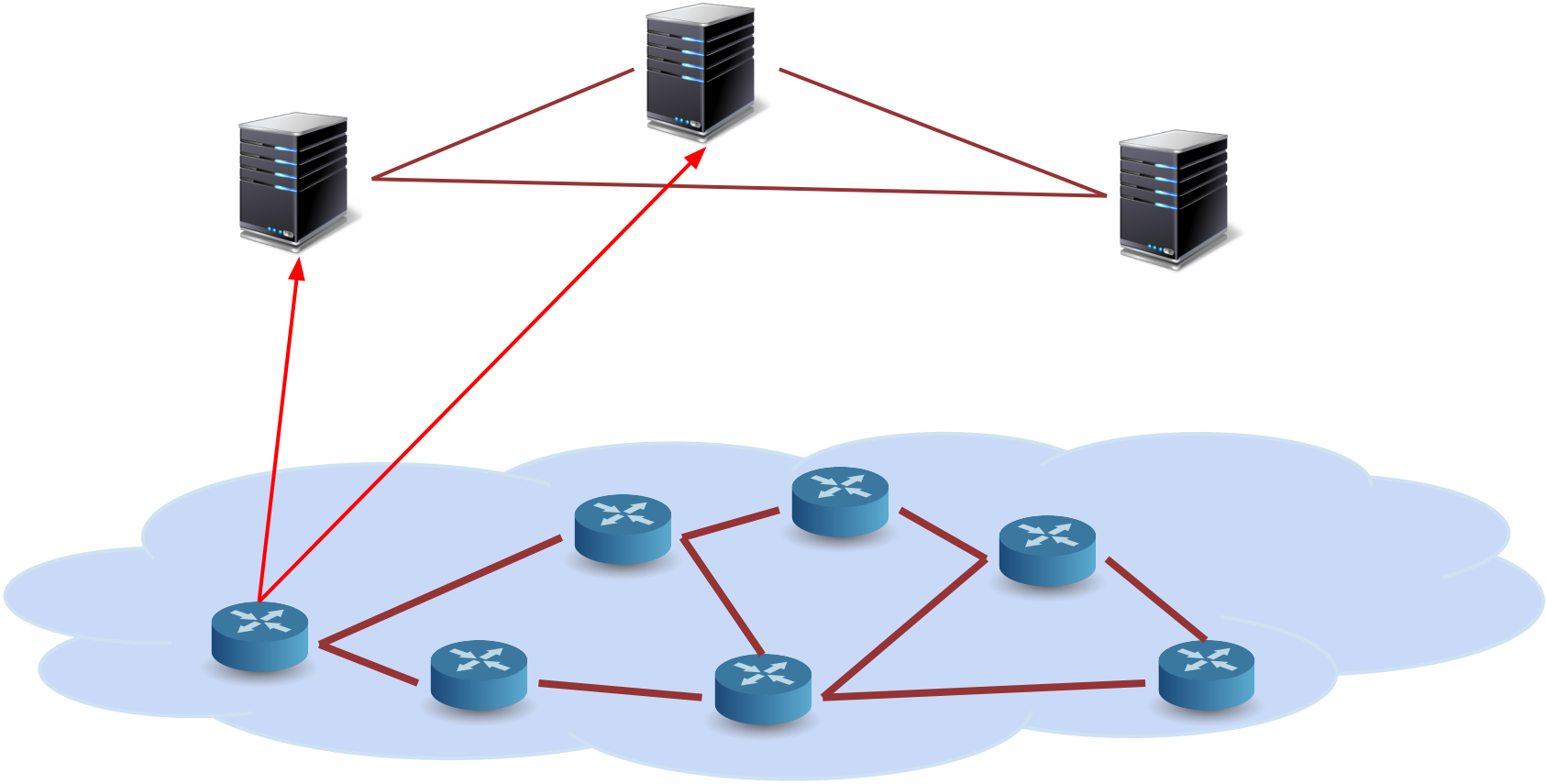
Distributed State Management

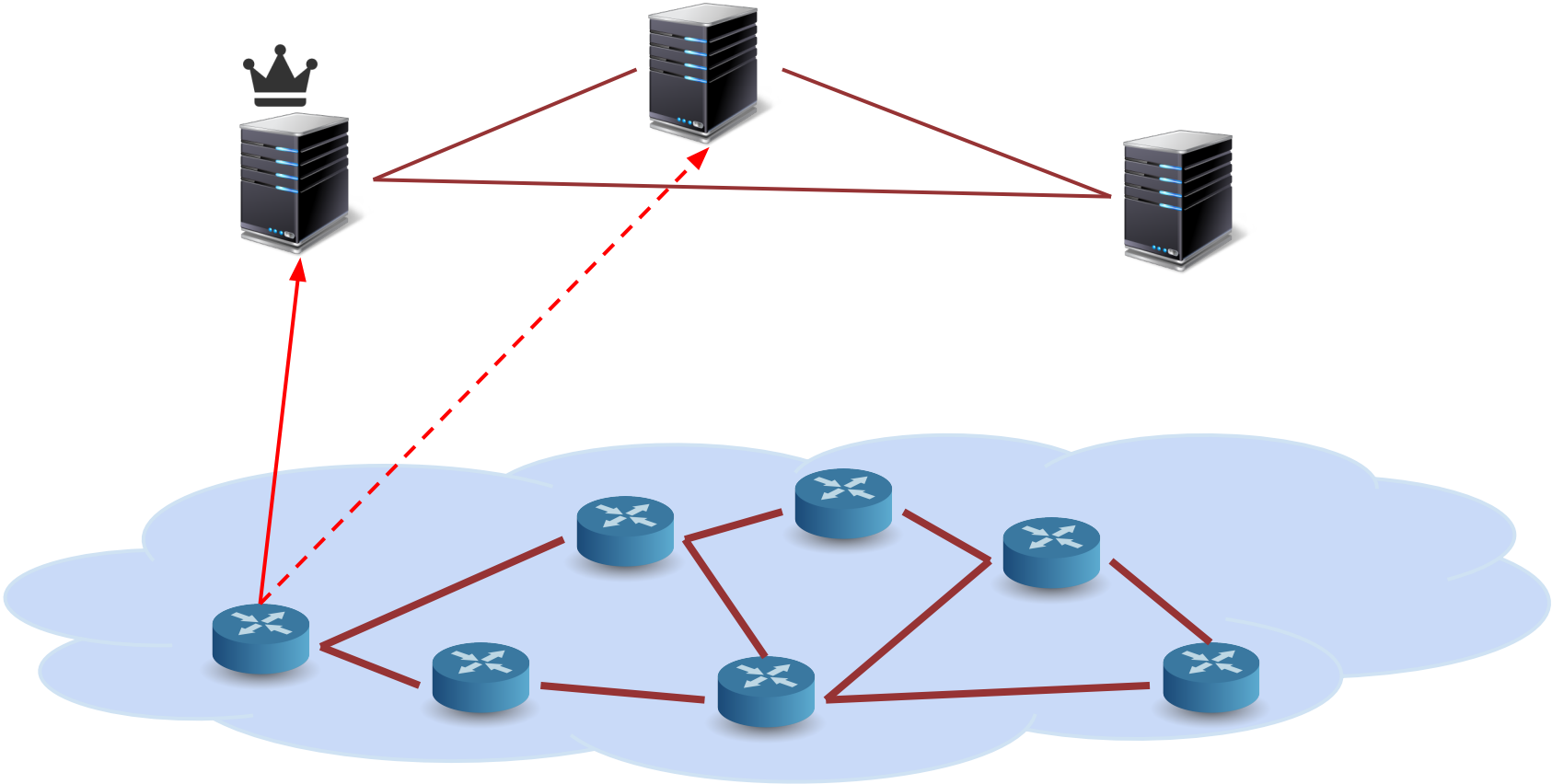
- Using specific examples - introduce various distributed state management primitives ONOS provides

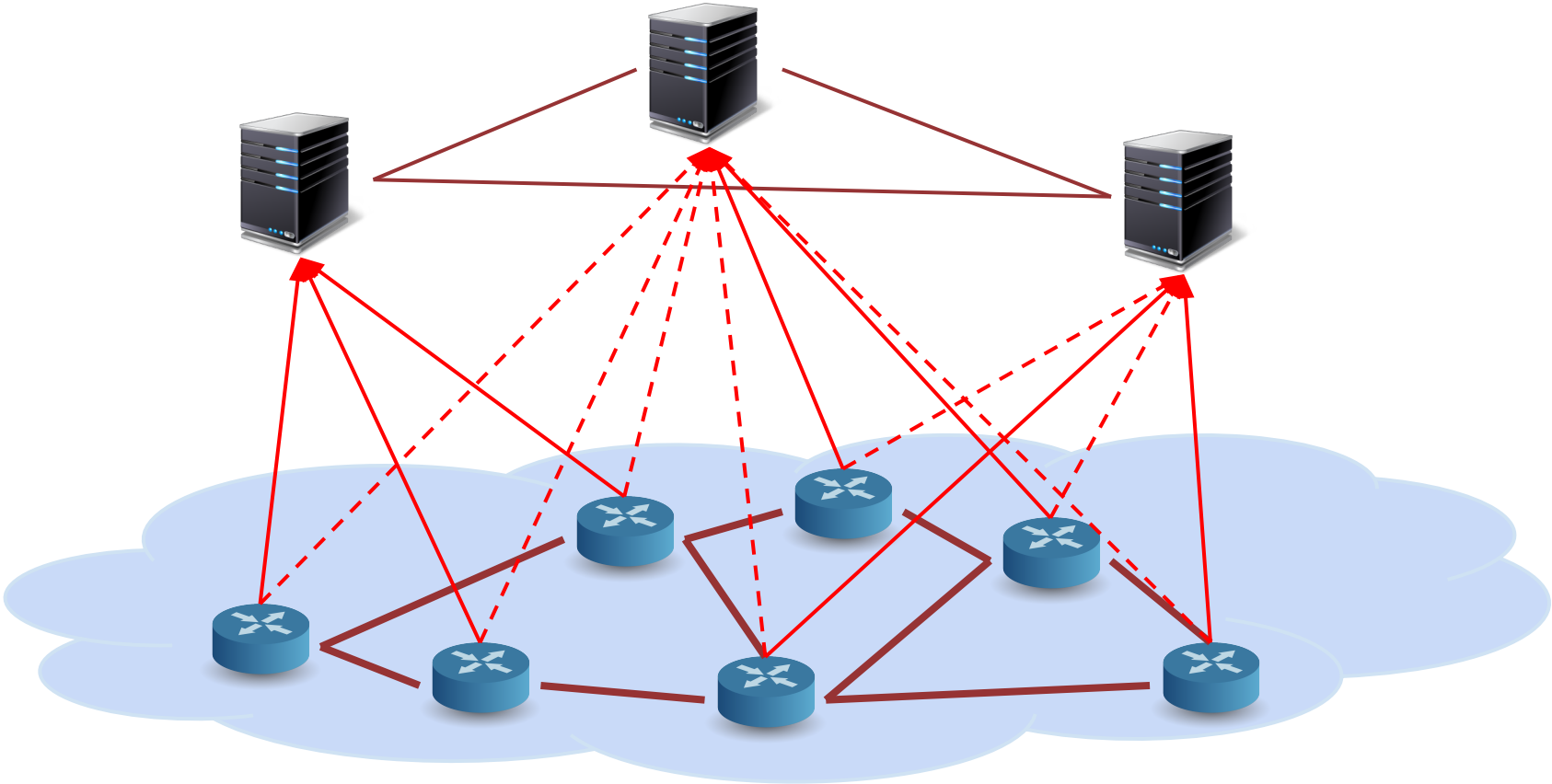
Distributed Control Plane

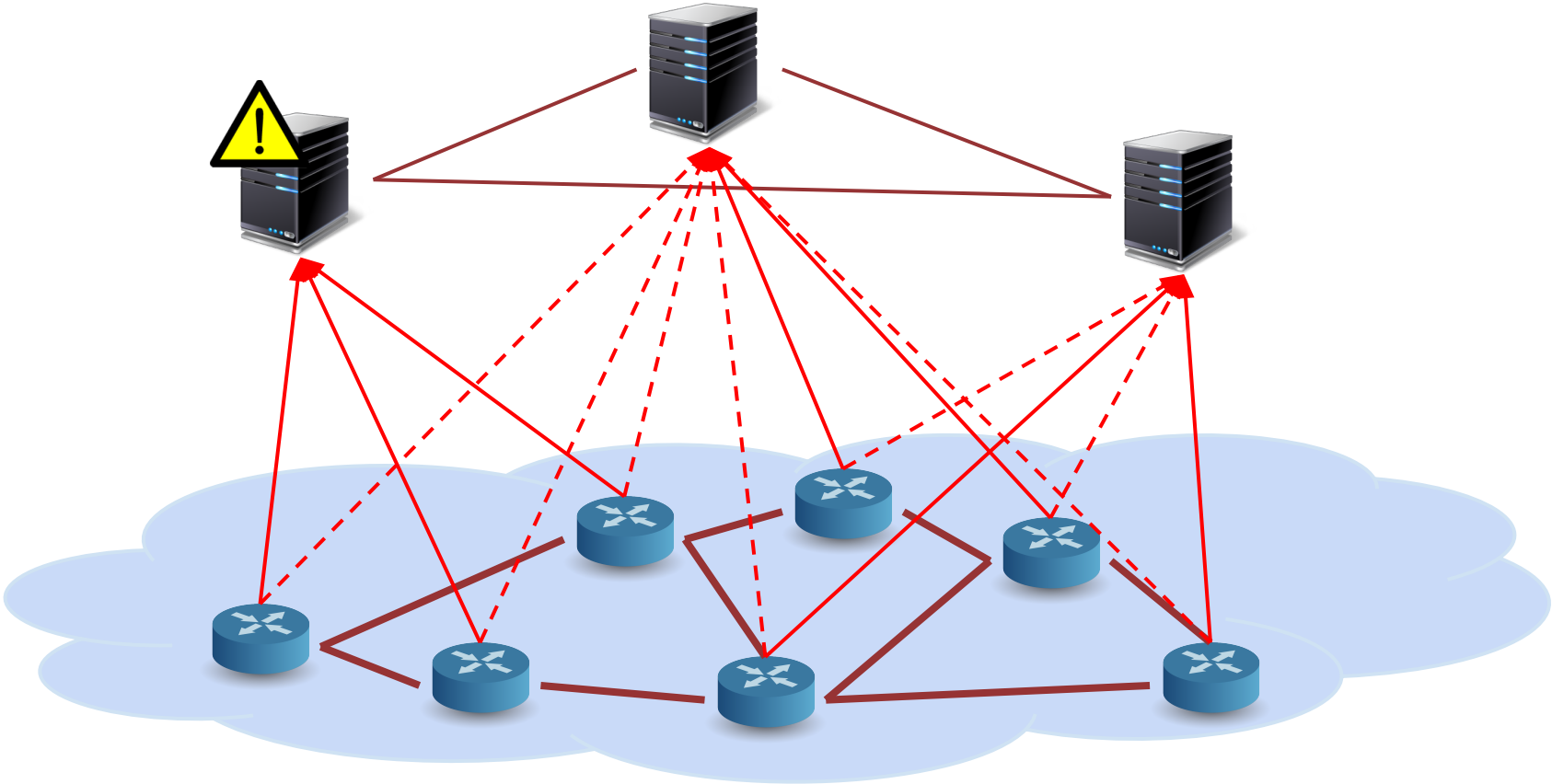


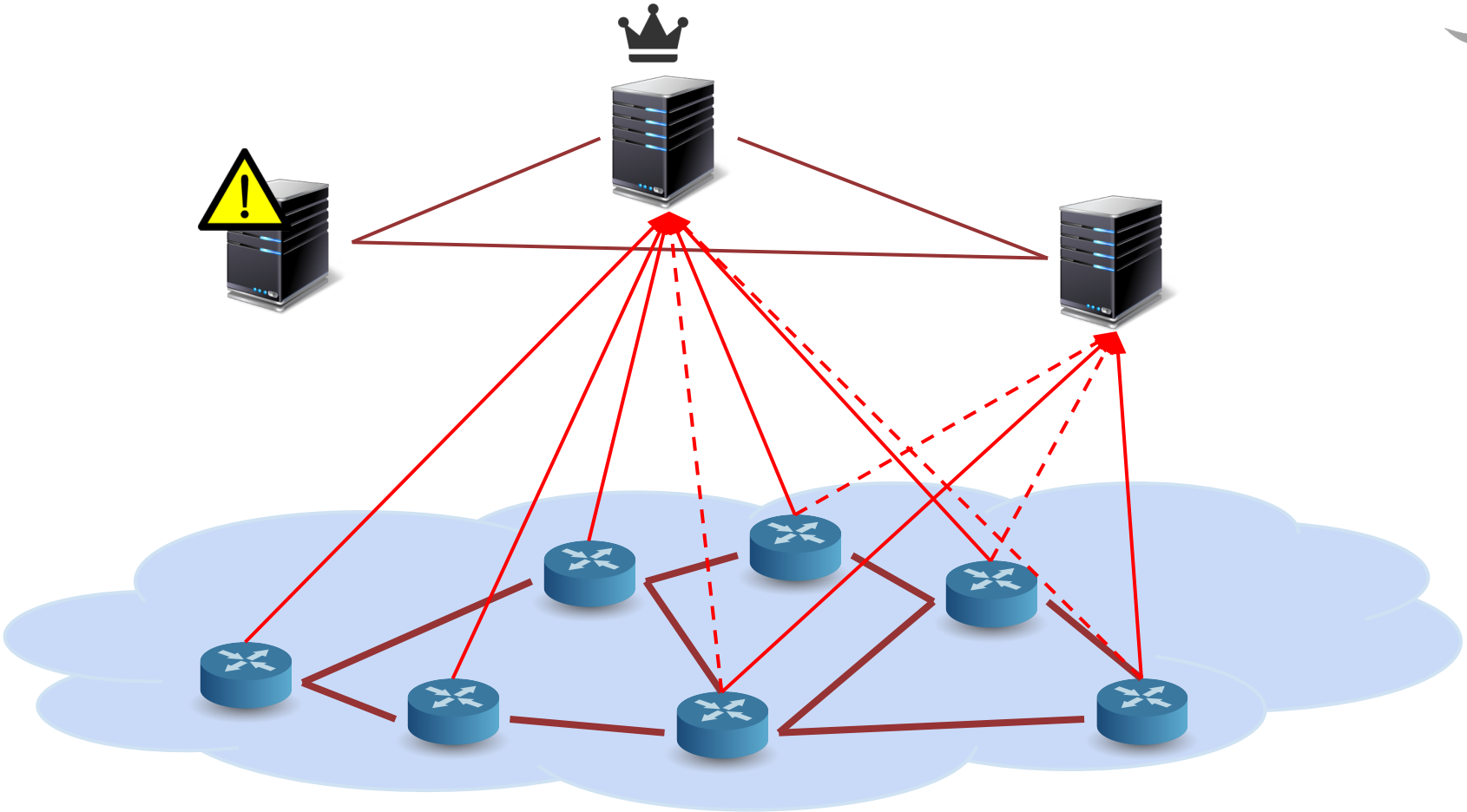










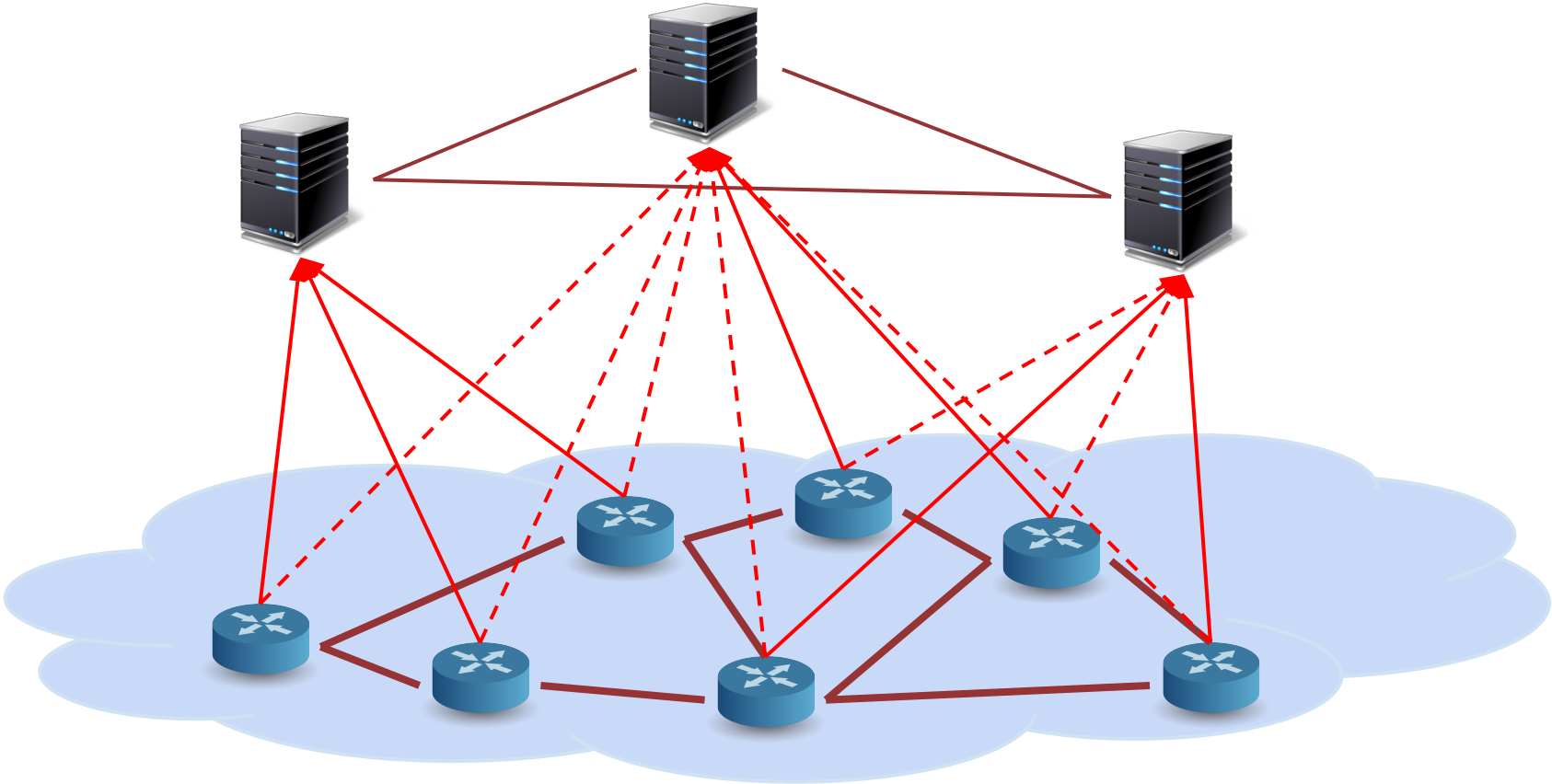


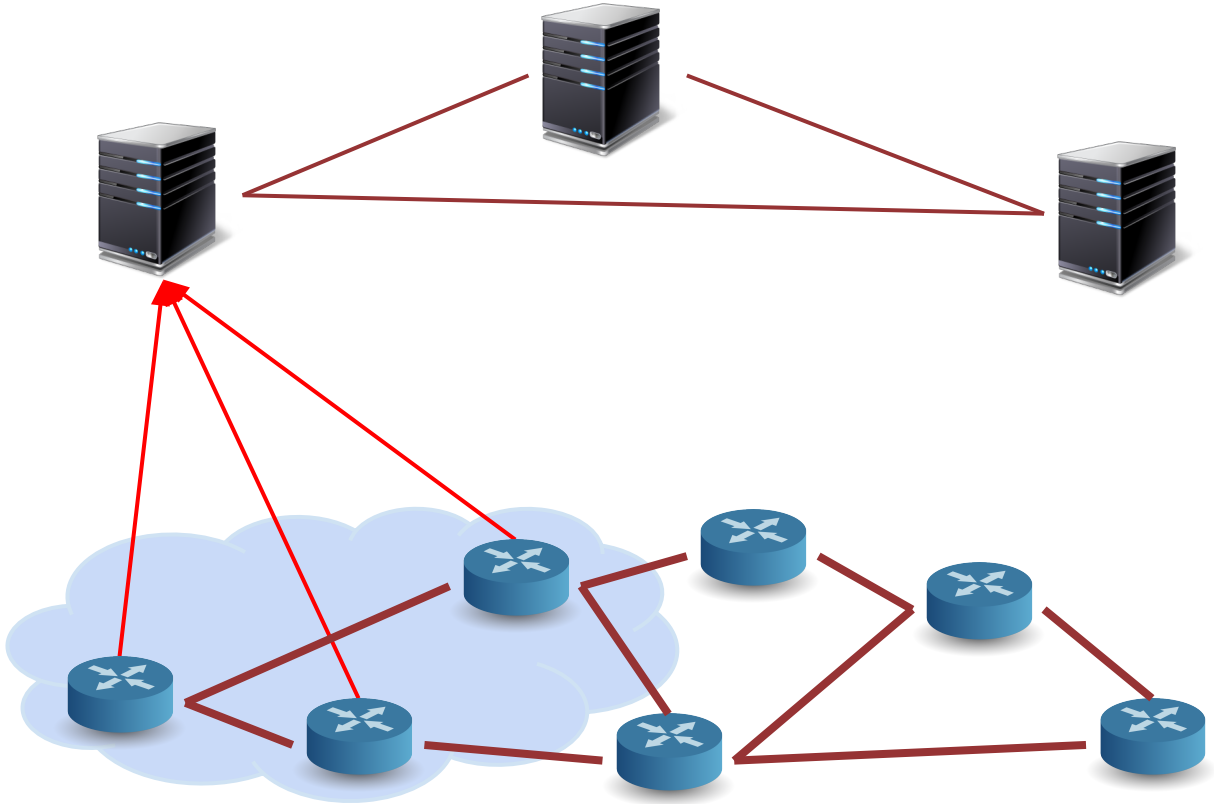
Distributed Primitive: Leadership

- Support for **mutual exclusion** in a distributed setting
- Available as a primitive for applications to consume
- Implemented via the **Raft consensus** protocol
- Cluster **majority** is needed to reach consensus

Deep dive into Topology state

- Formulate the problem
- Evaluate solution space
- Description of ONOS' approach
- Performance evaluation

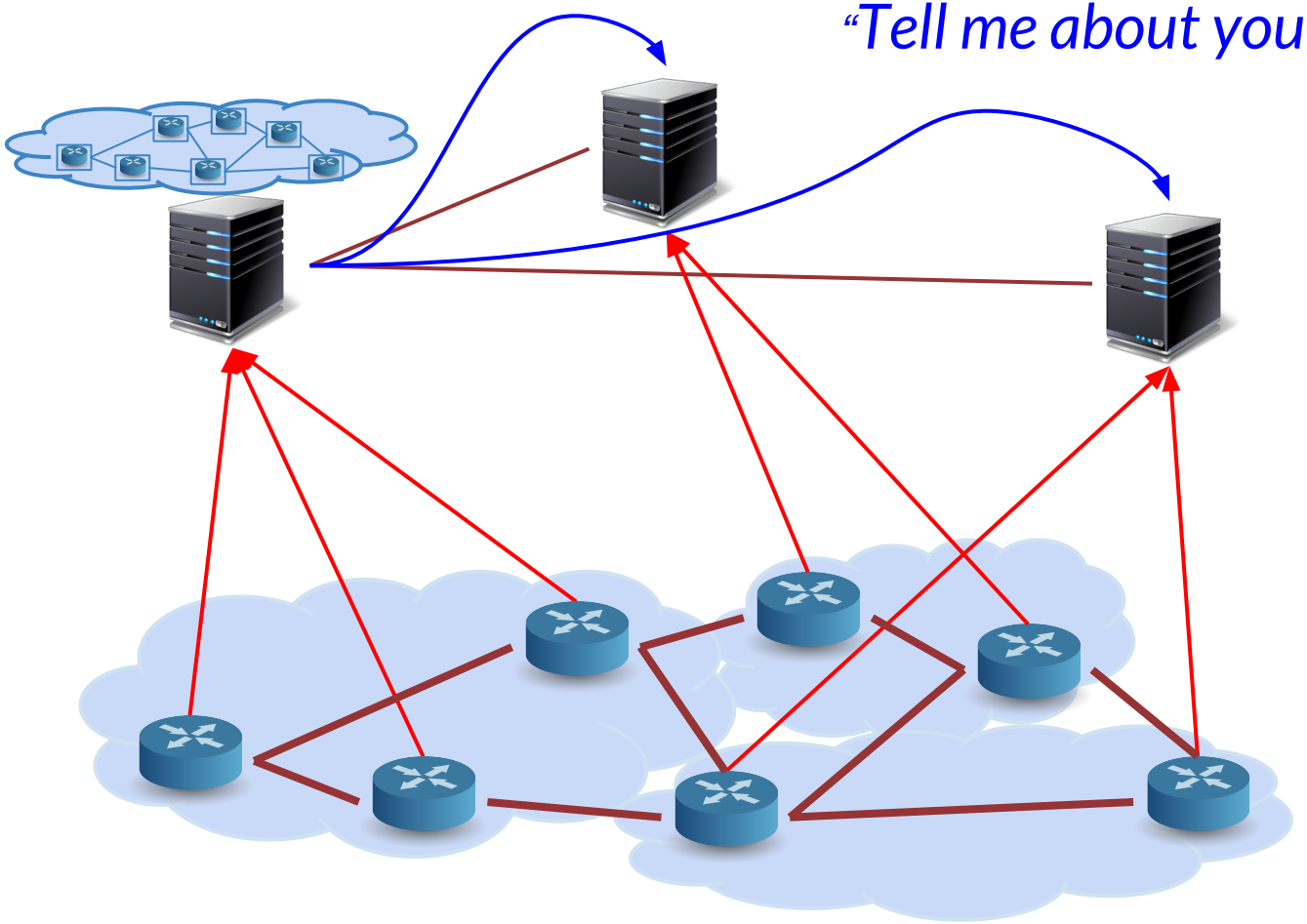




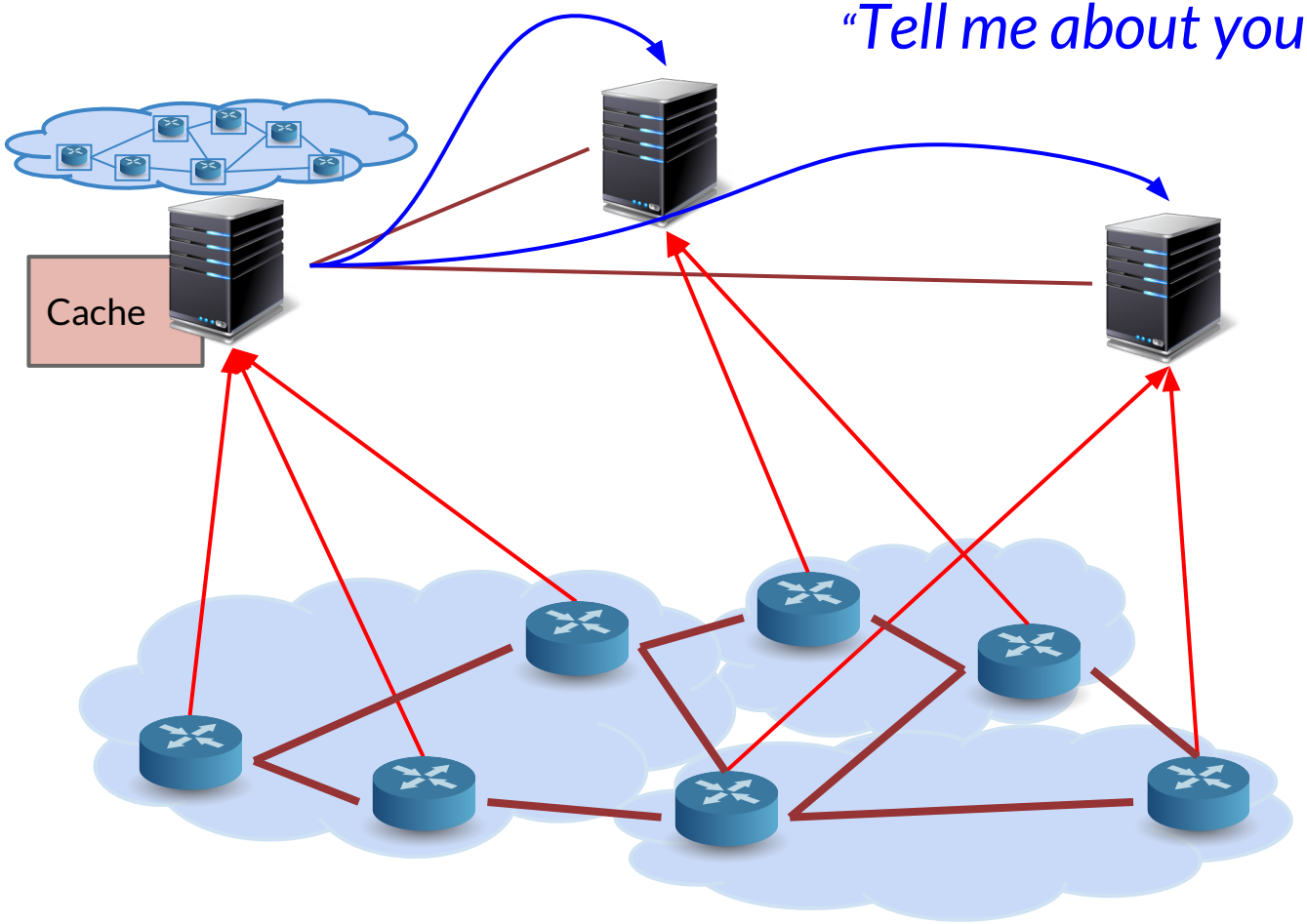
Solution space

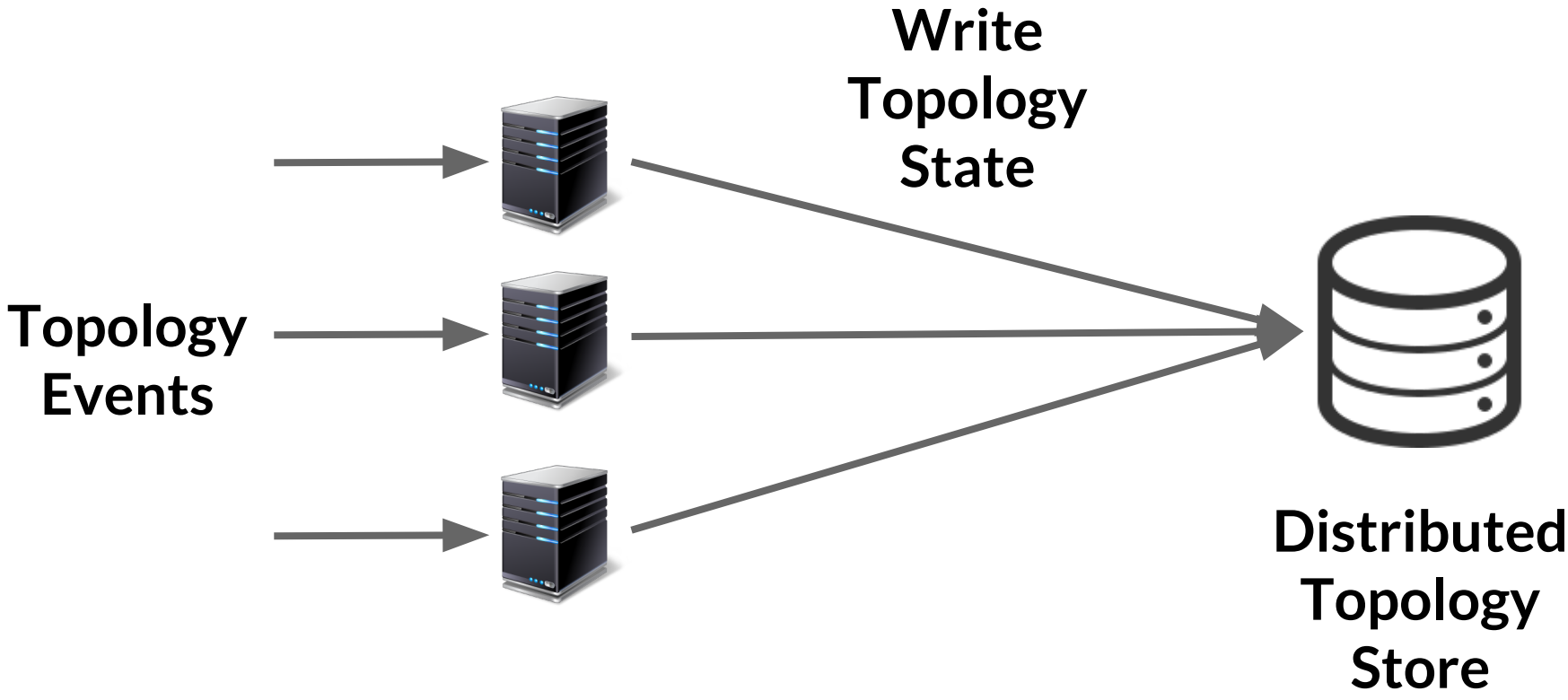
- Partitioned by mastership
- Logically centralized data store
- ONOS approach

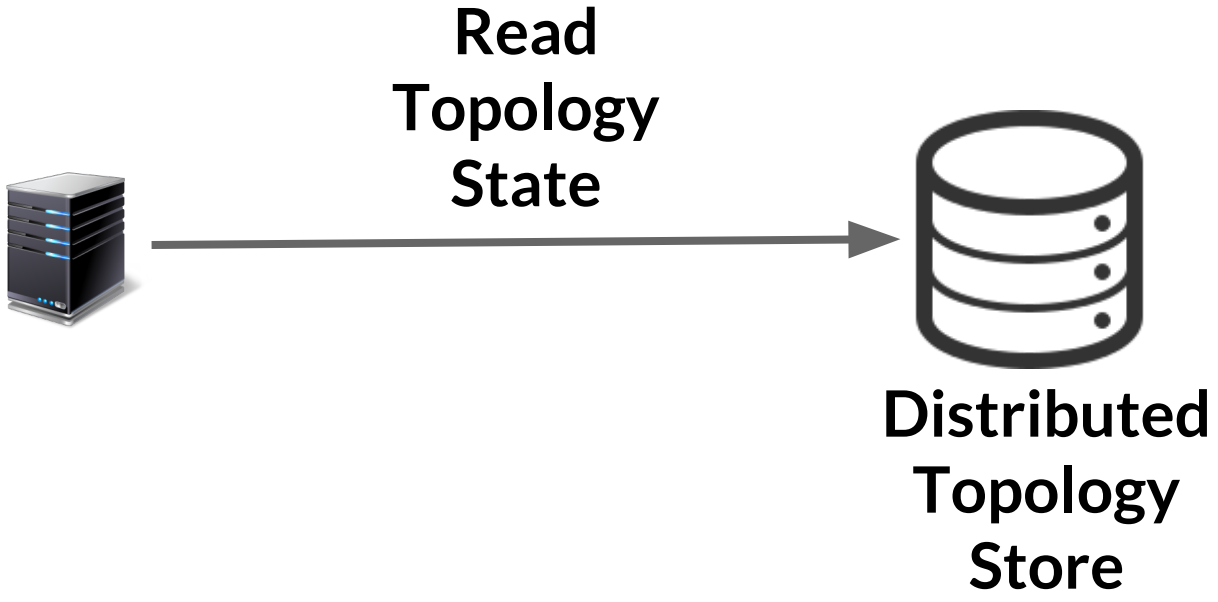
“Tell me about your slice?”

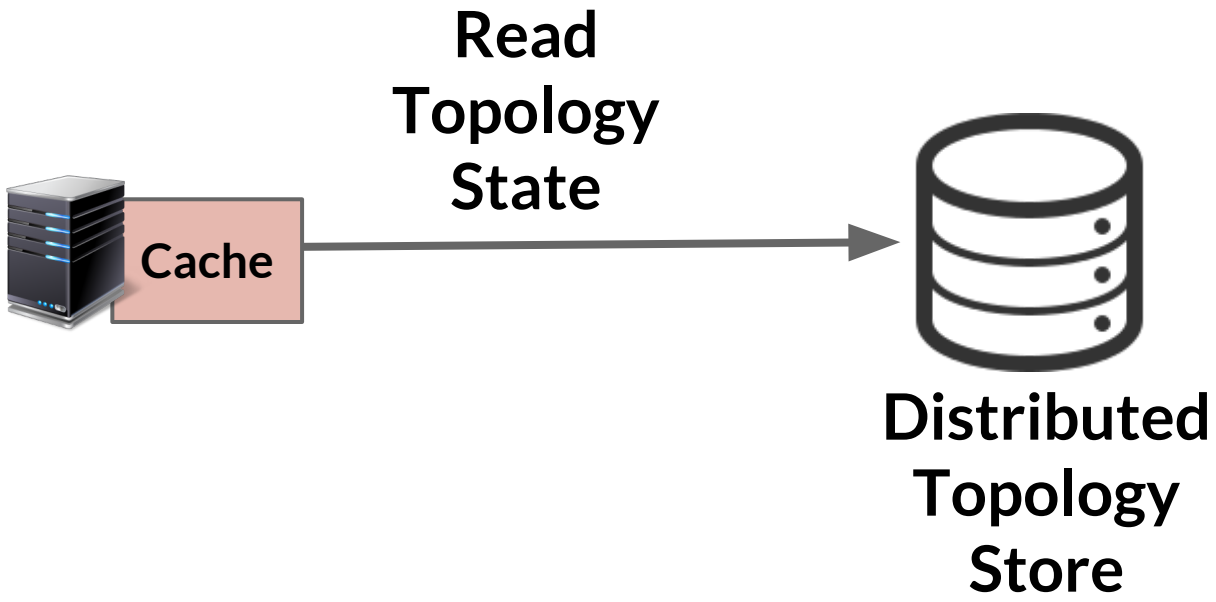


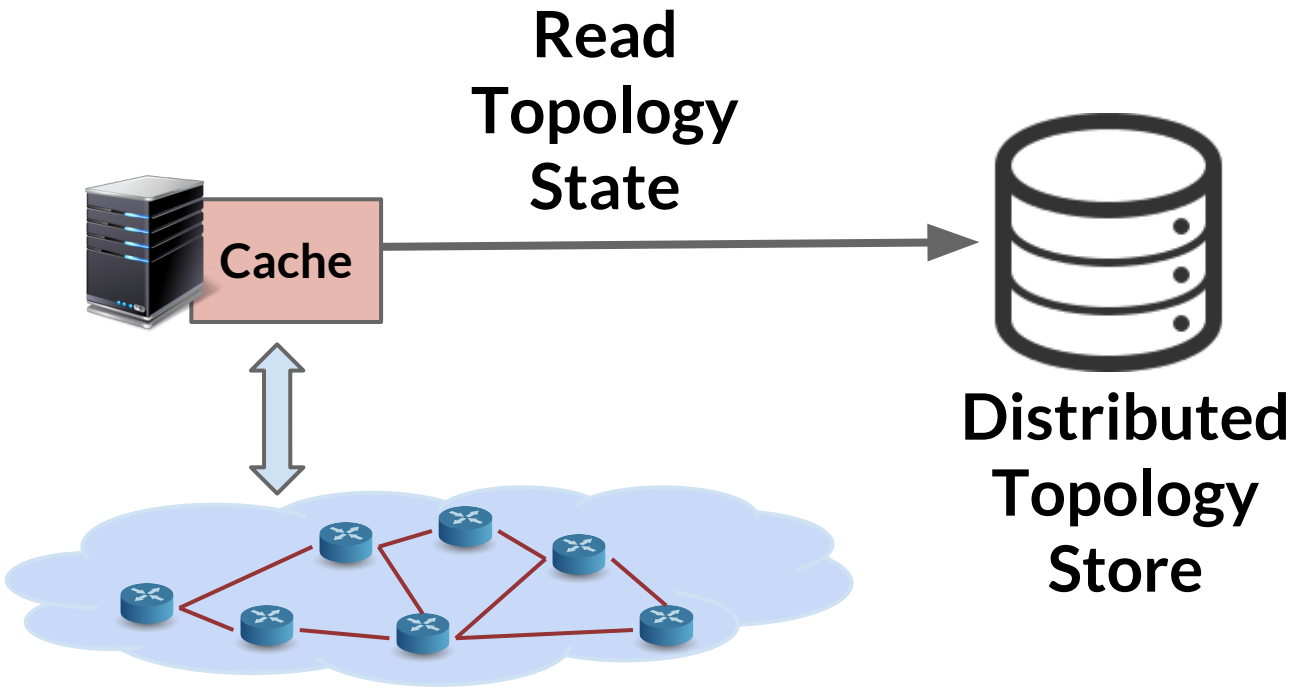
“Tell me about your slice?”



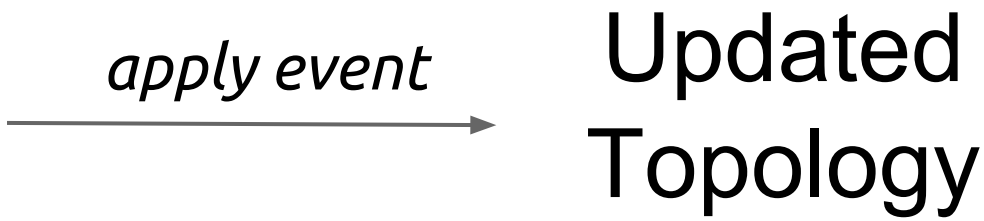

Write
Topology
State
Topology
Events
Distributed
Topology
Store


Read
Topology
State
Distributed
Topology
Store

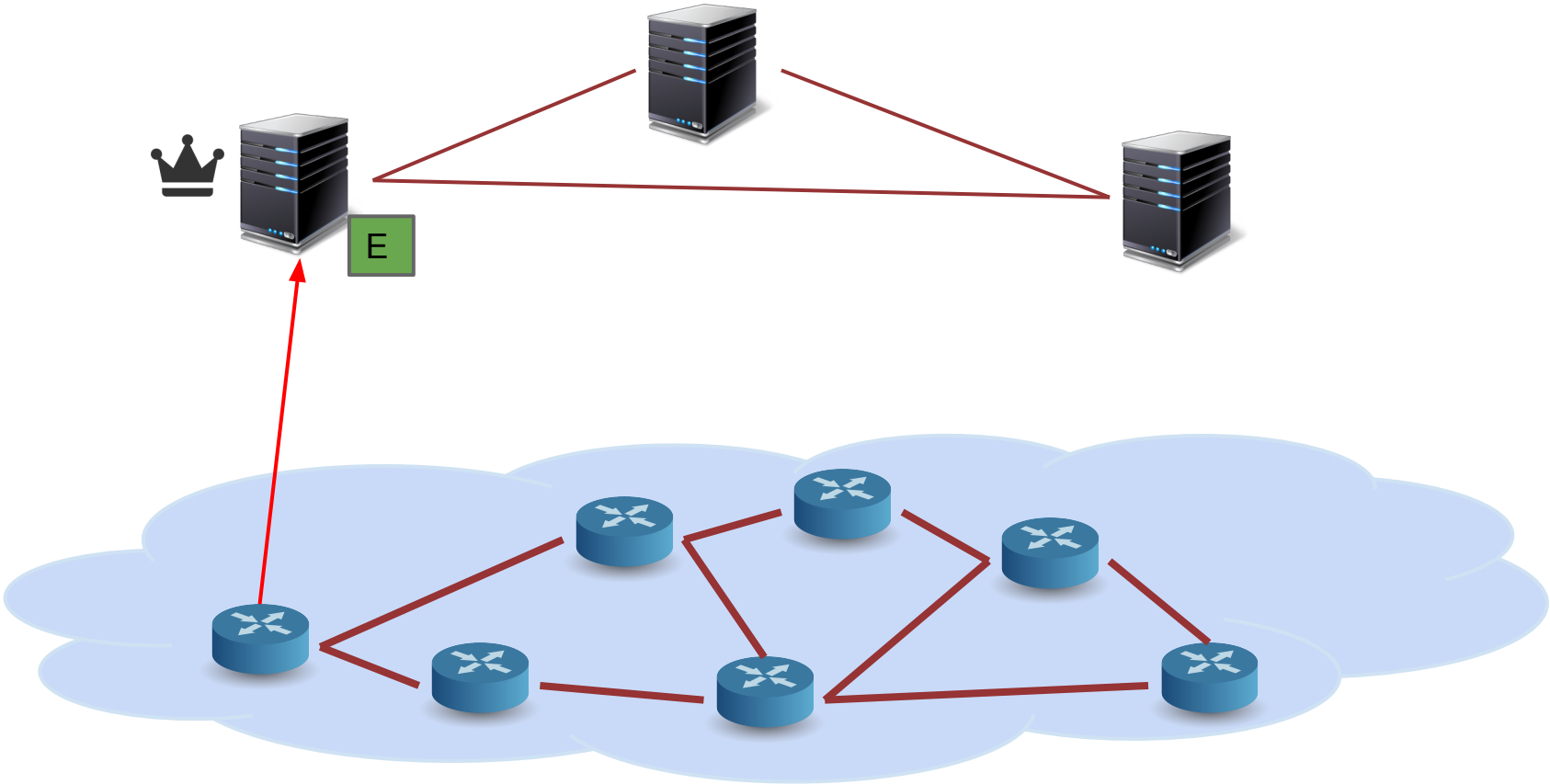

Read
Topology
State
Cache
Distributed
Topology
Store

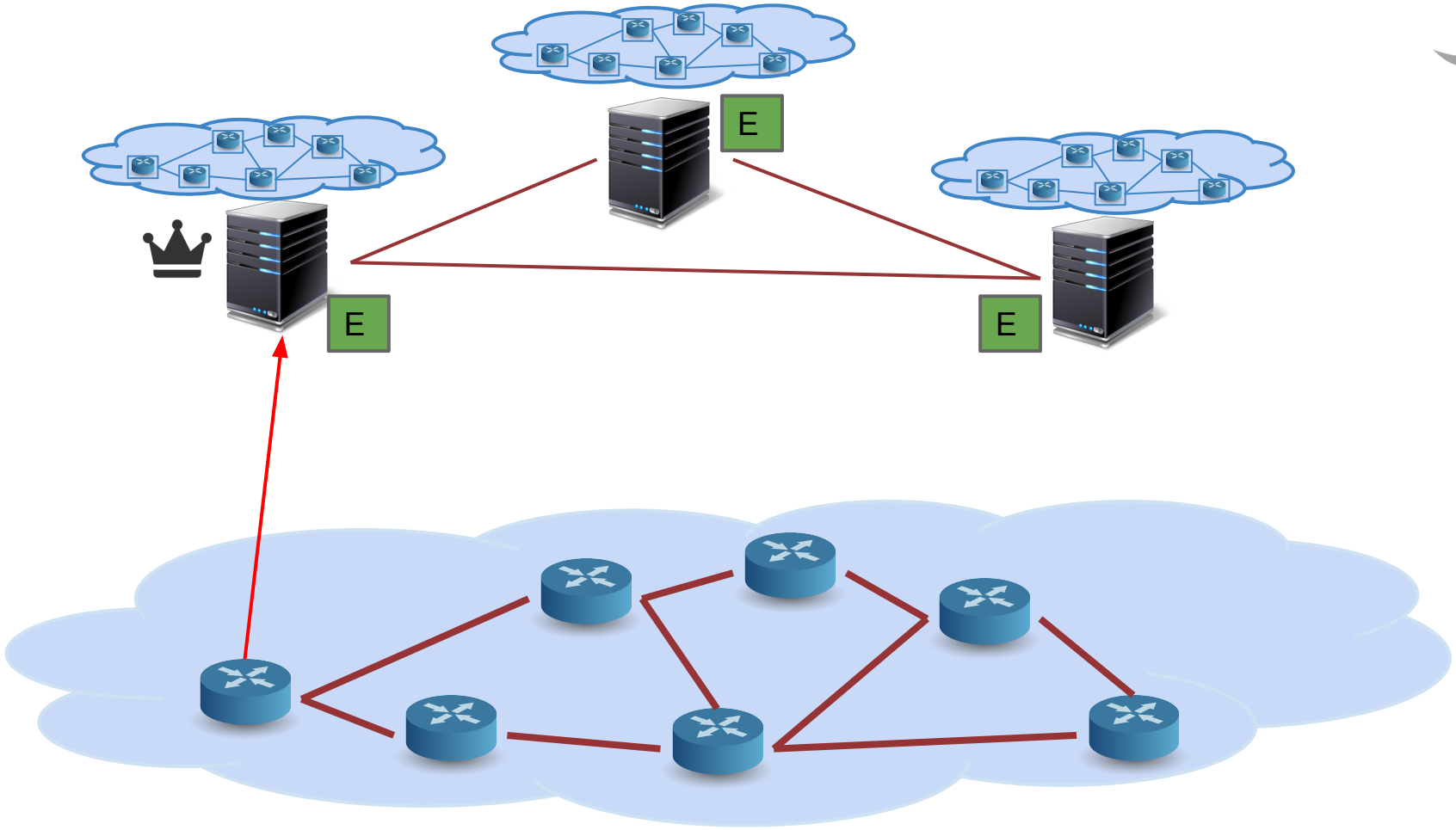

Read
Topology
State
Cache
Distributed
Topology
Store

Topology as a State Machine

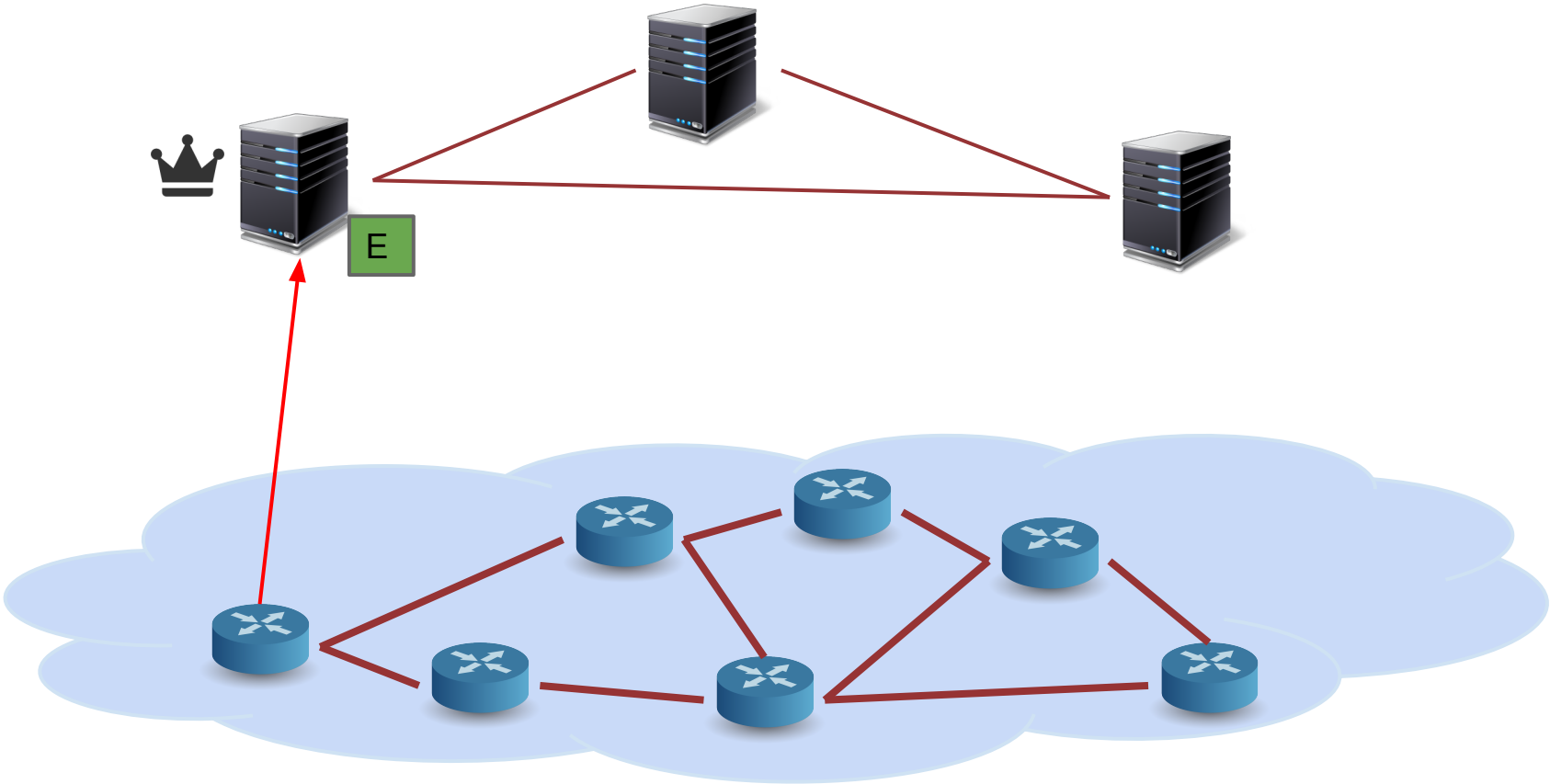


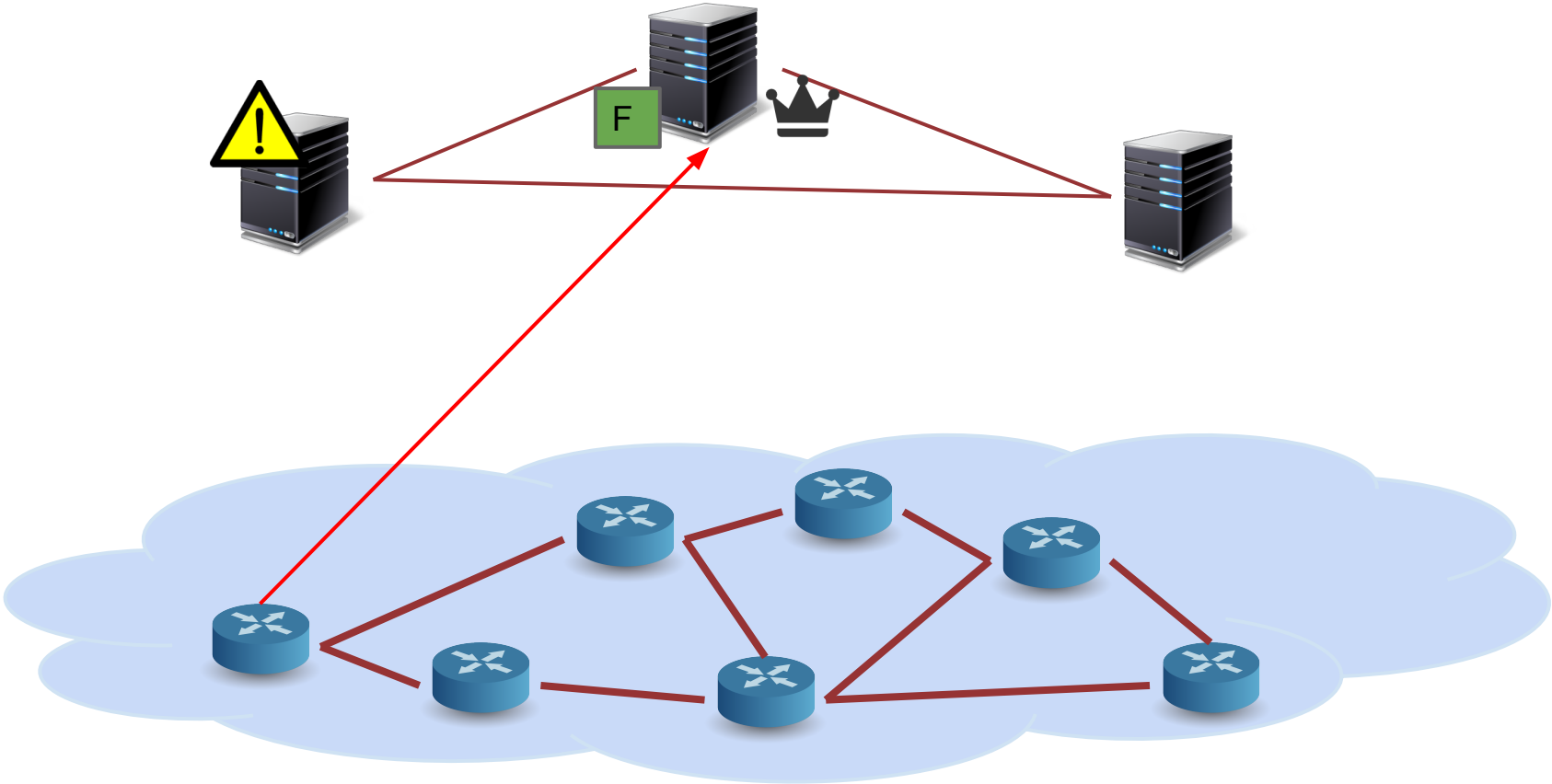
Events are Switch/Port/Link up/down

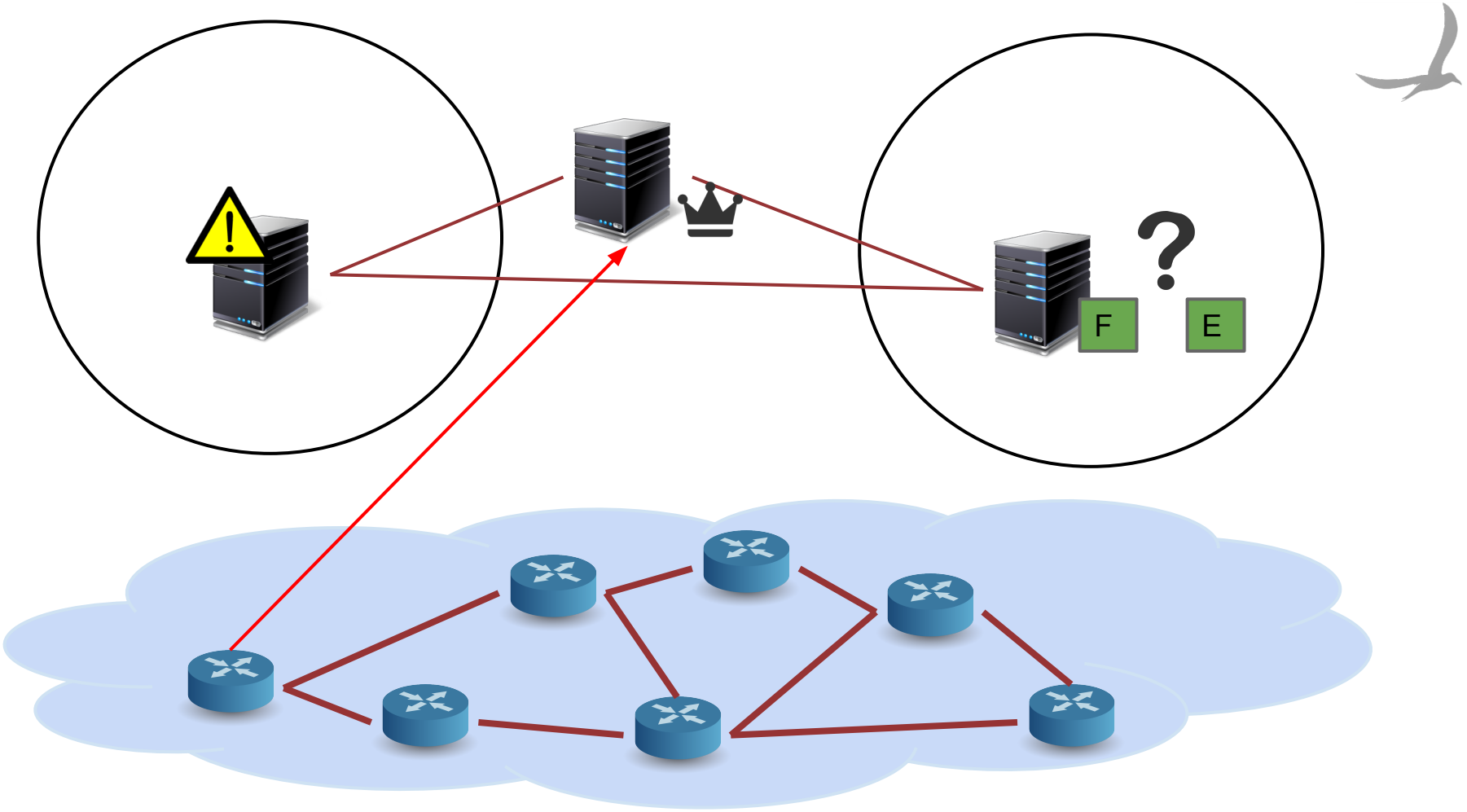

E


E
E
E

Pros
● Simple ● Fast

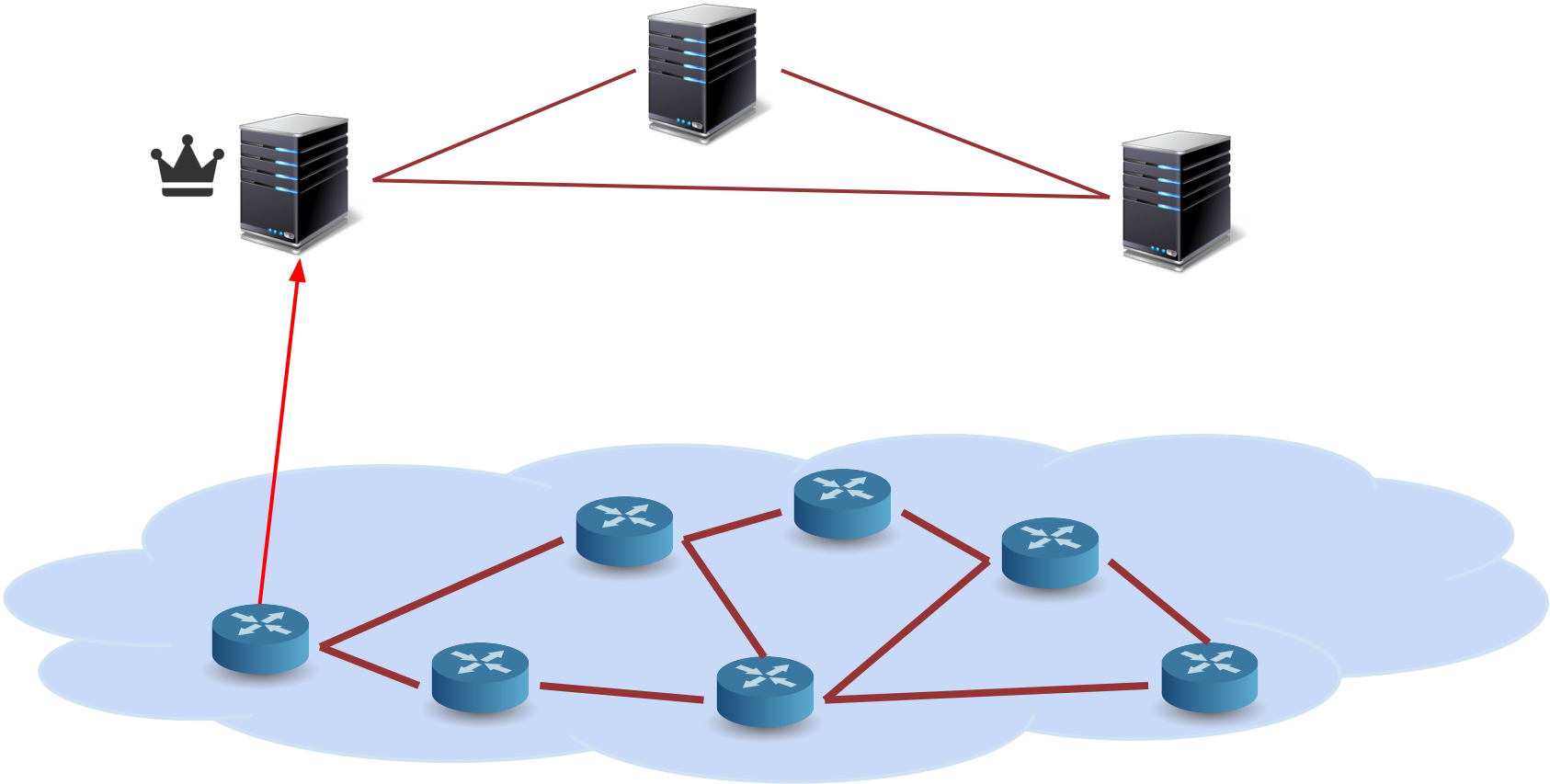

E

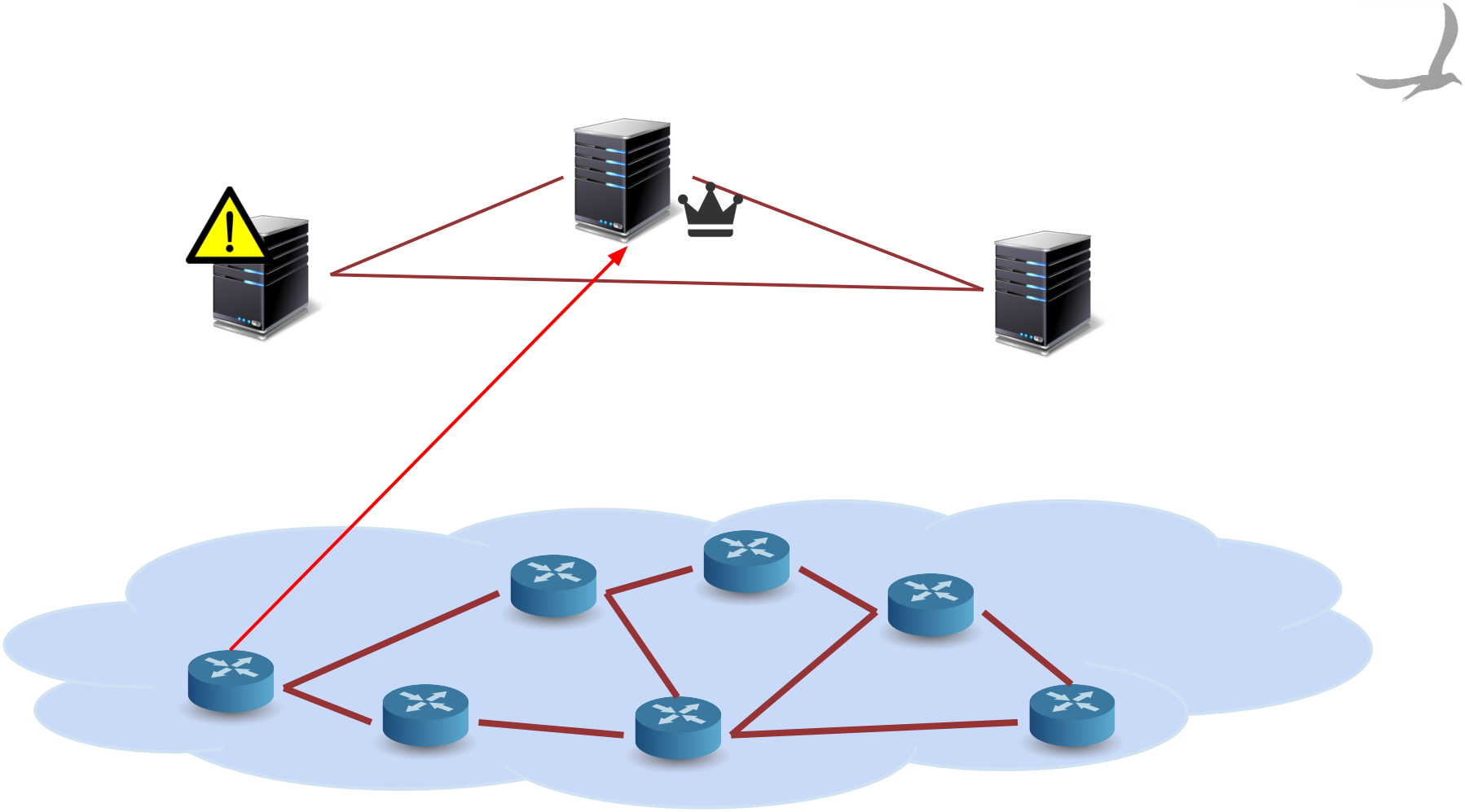

F


F
E

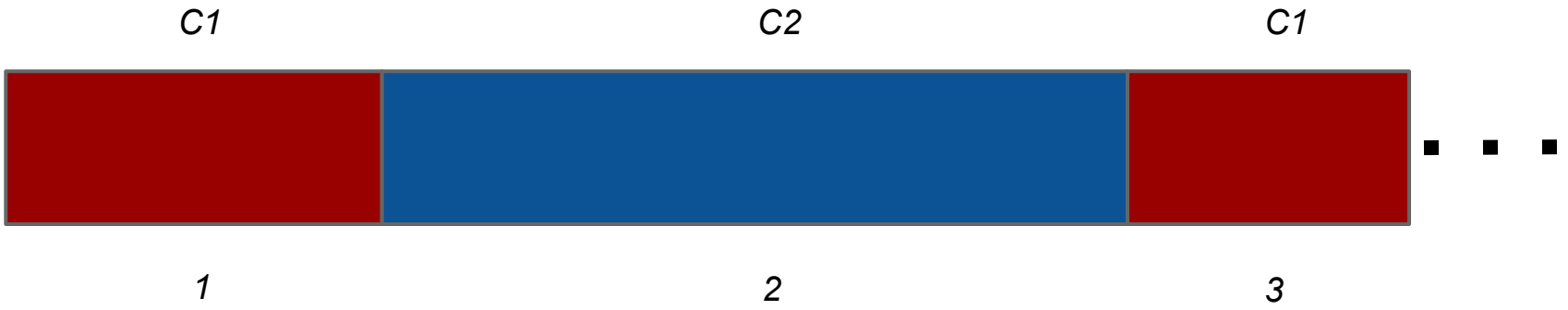
Pros	Cons
● Simple ● Fast	● Dropped messages ● Reordered messages

How to address the cons without sacrificing the pros?



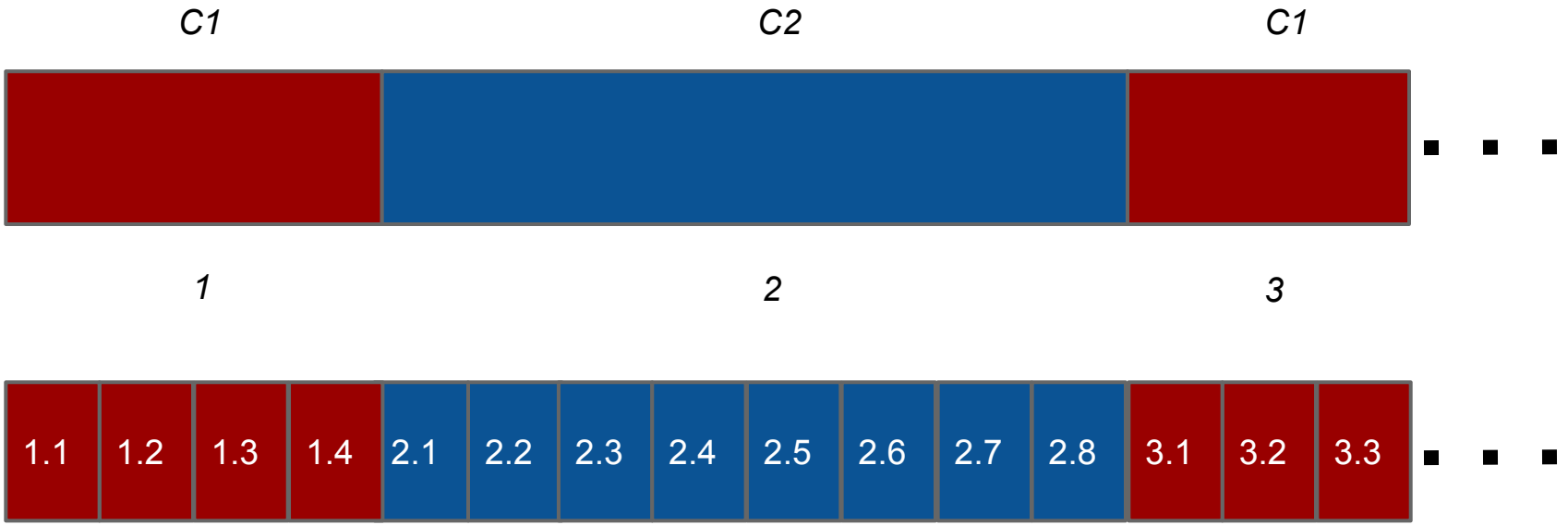


Switch Mastership Terms



We track this in a strongly consistent store

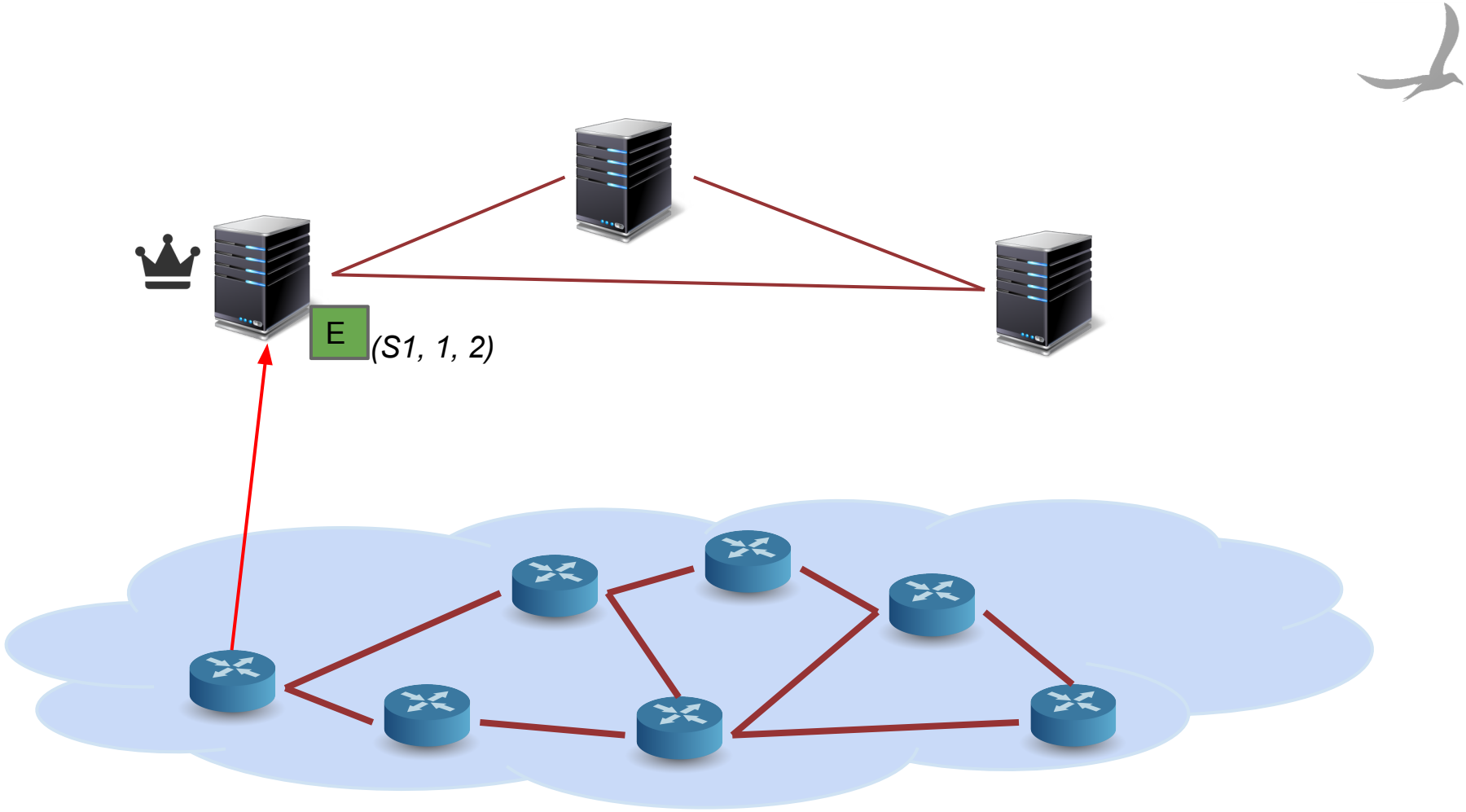
Switch Event Numbers

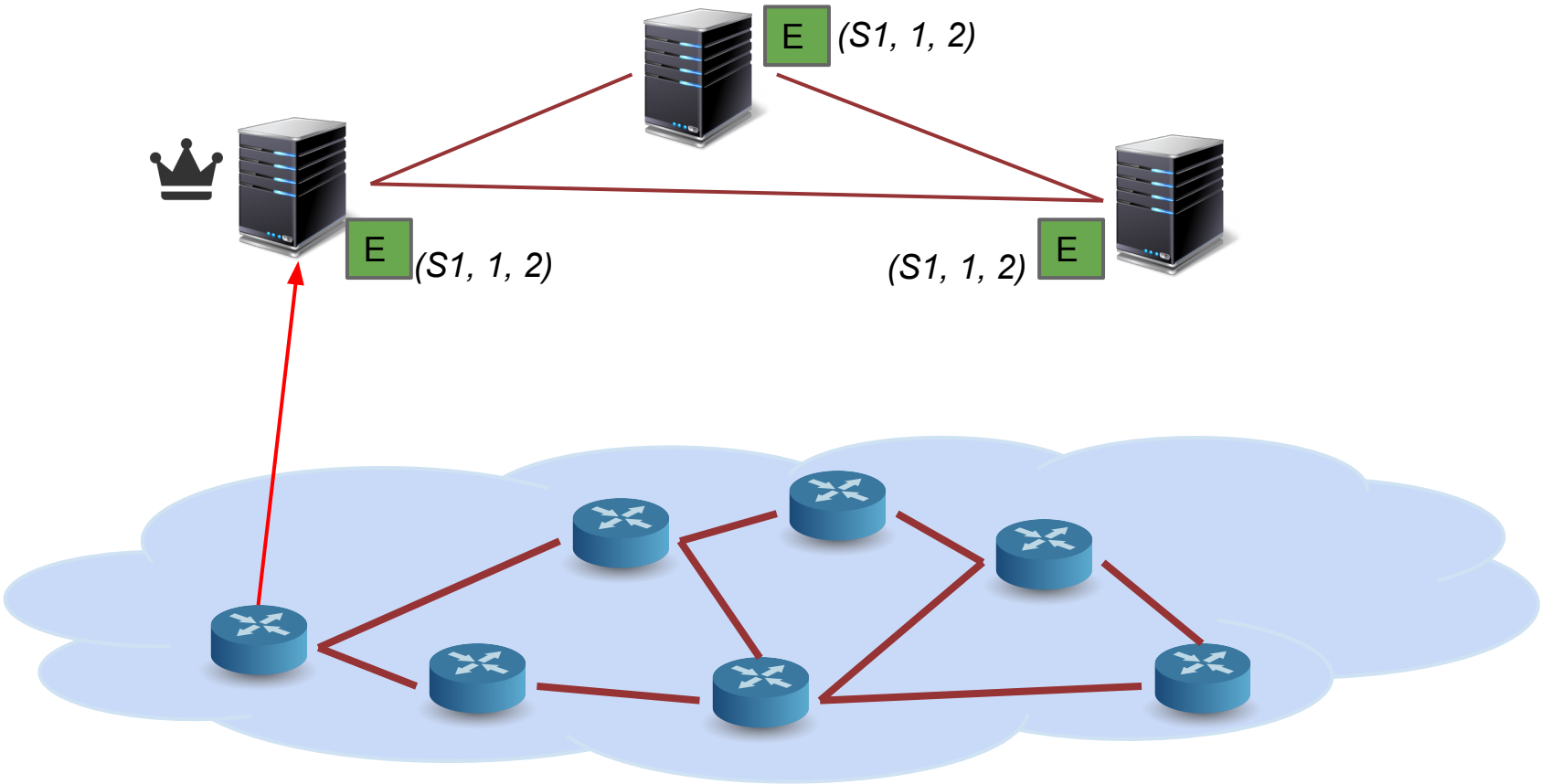


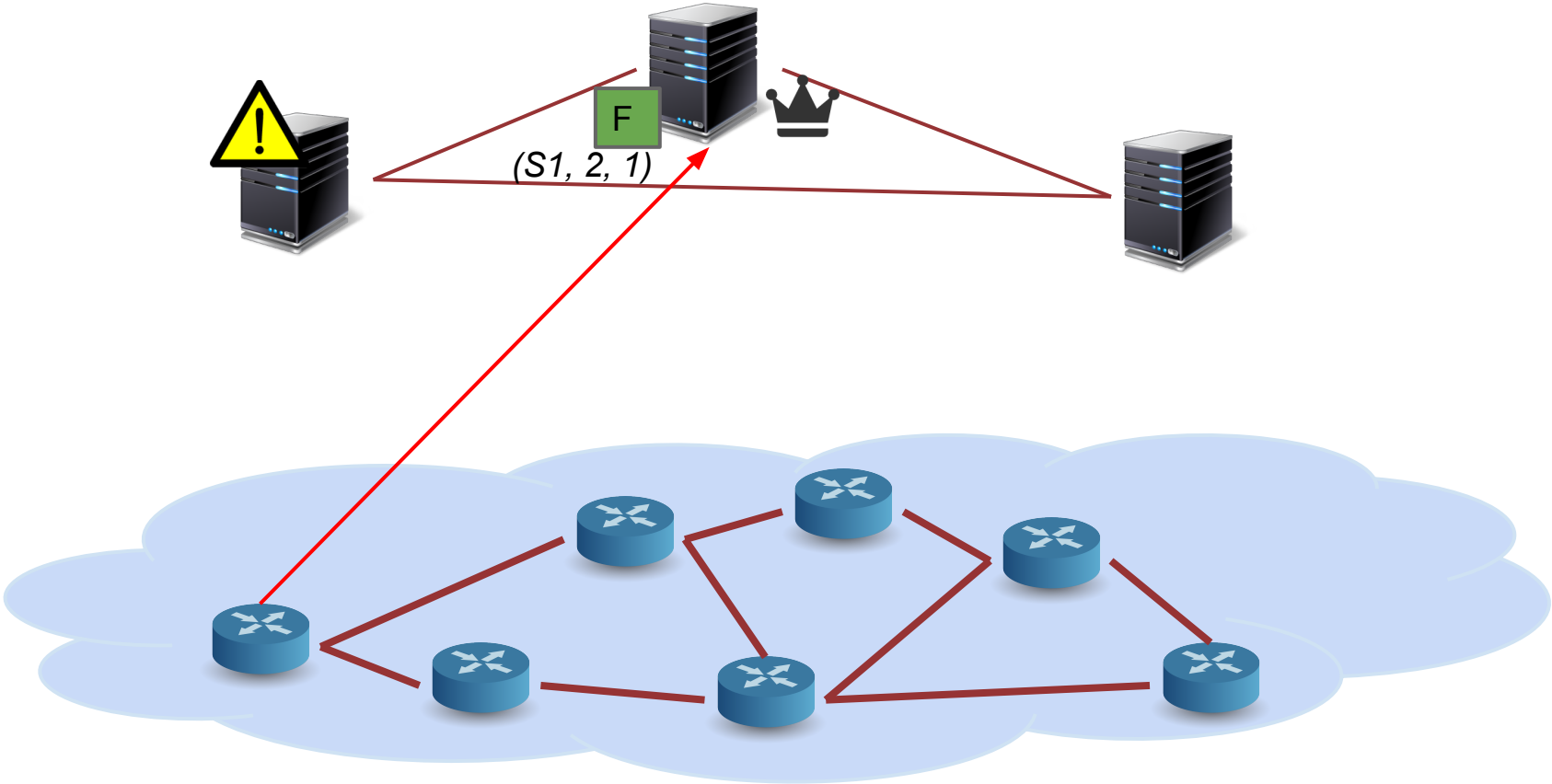
Partial Ordering of Events

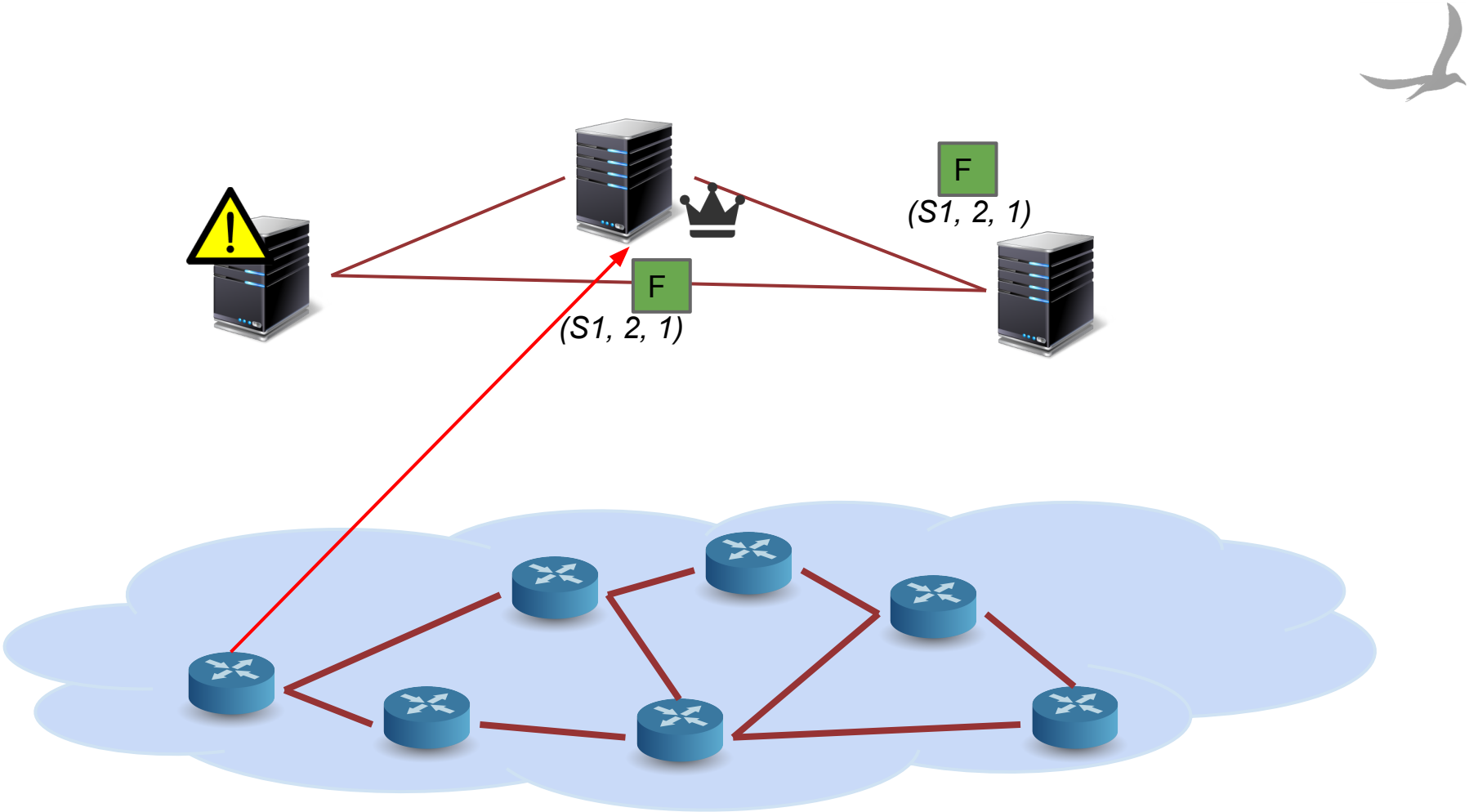
Each event has a unique logical timestamp

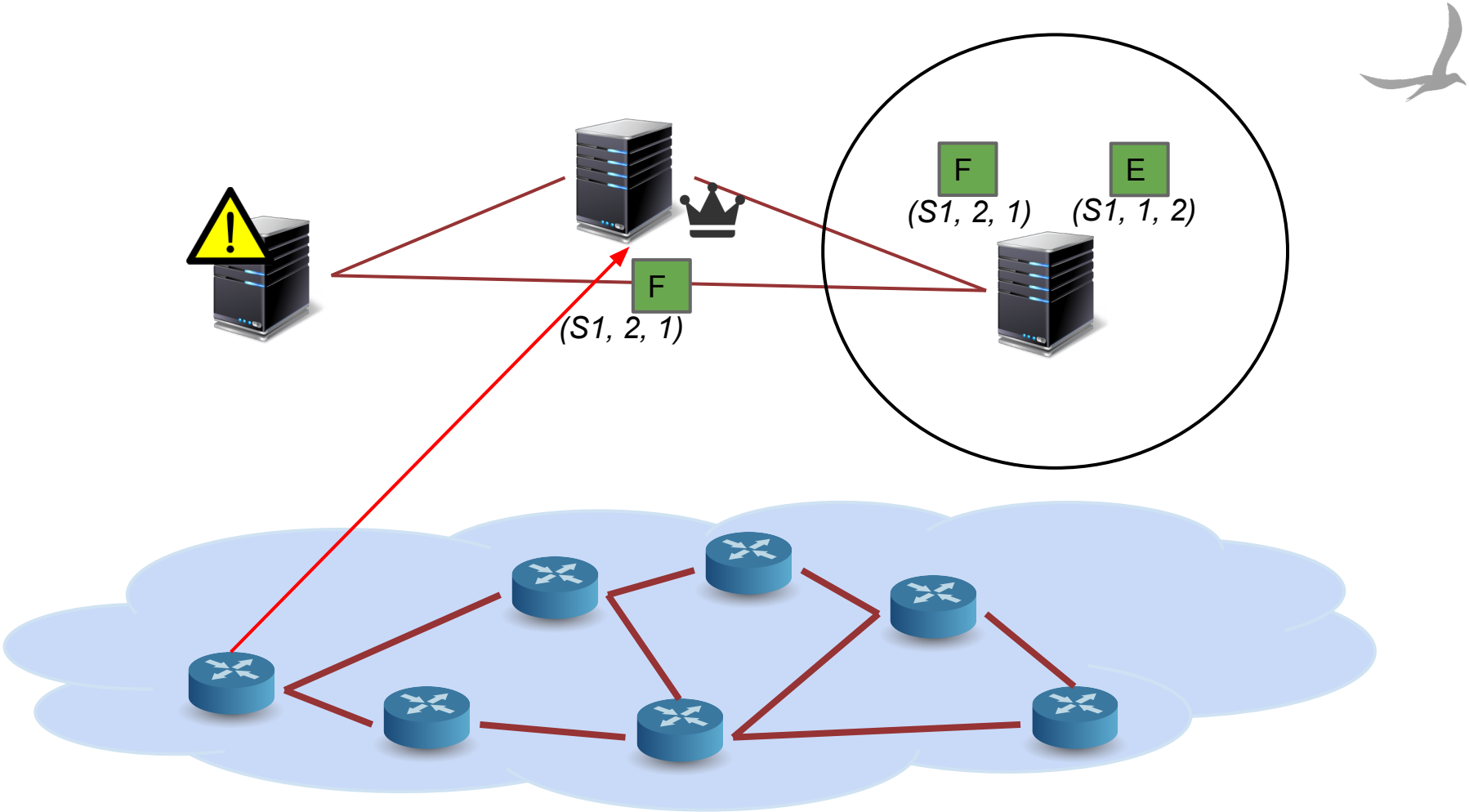
(Switch ID, Term Number, Event Number)

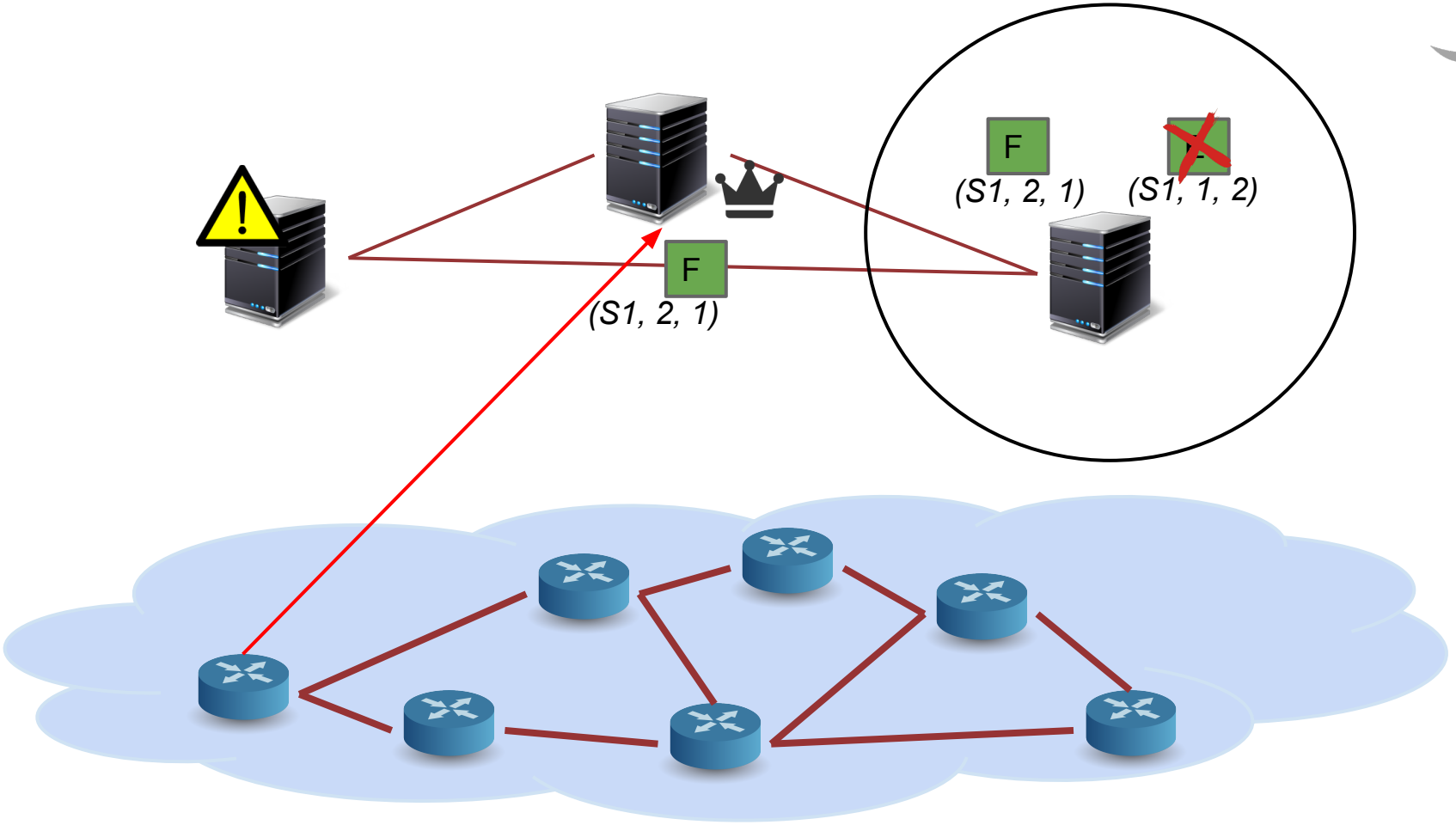

E
(S1, 1, 2)


E
(S1, 1, 2)
E
(S1, 1, 2)
(S1, 1, 2)
E


F
(S1, 2, 1)


F
(S1, 2, 1)
F
(S1, 2, 1)


F
(S1, 2, 1)
E
(S1, 1, 2)
F
(S1, 2, 1)


F
(S1, 2, 1)
F
(S1, 1, 2)
F
(S1, 2, 1)

To Summarize

- Each instance has a full copy of network topology
- Events are timestamped on arrival and broadcasted
- Stale events are dropped on receipt

To Summarize

- Each instance has a full copy of network topology
- Events are timestamped on arrival and broadcasted
- Stale events are dropped on receipt

What about dropped messages?


Did you hear about the port that went offline?
© Angie Brown

Anti-Entropy

Gossip-style

Lightweight

Decentralized

A model for state tracking

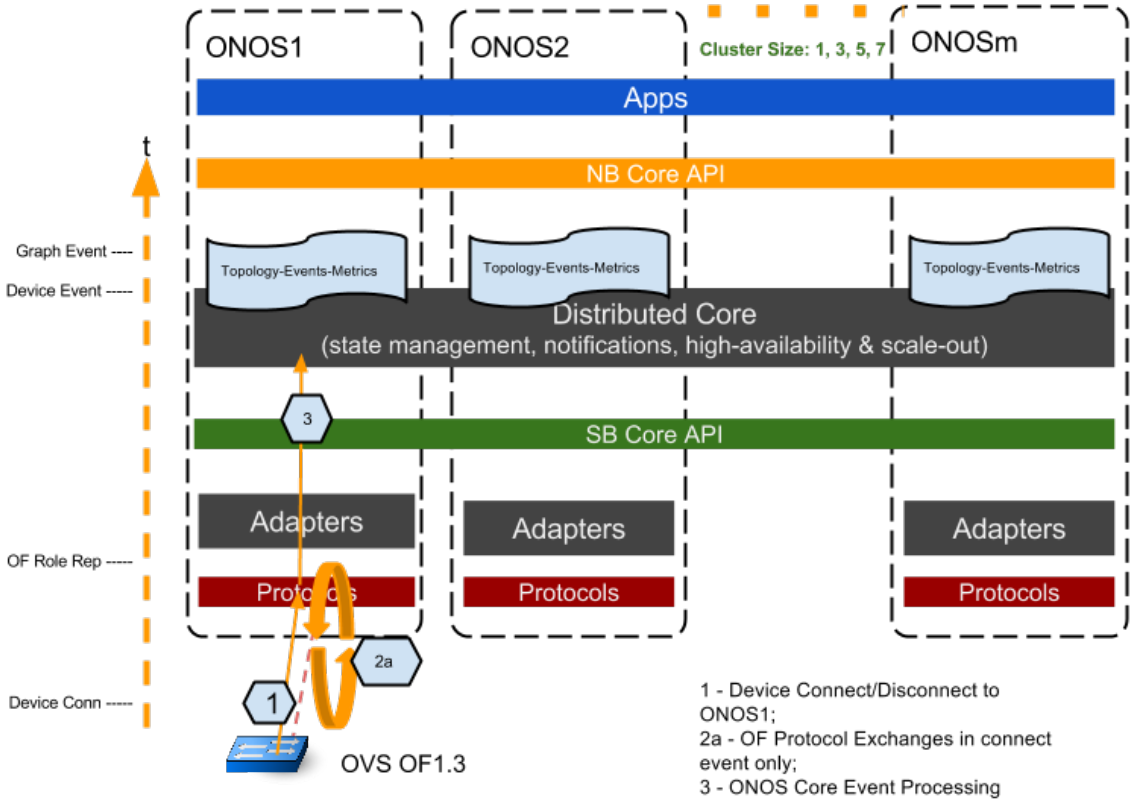
- If you are the observer, **Eventual Consistency** is the best option
- For topology state, ONOS provides **monotonic read consistency**
- View should be consistent with the network, not with other views

Distributed Primitive

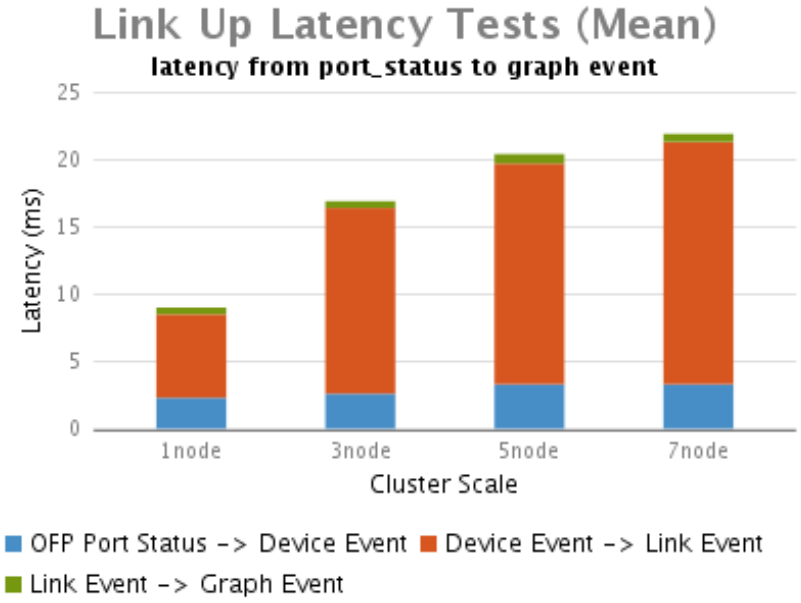
EventuallyConsistentMap<K, V>

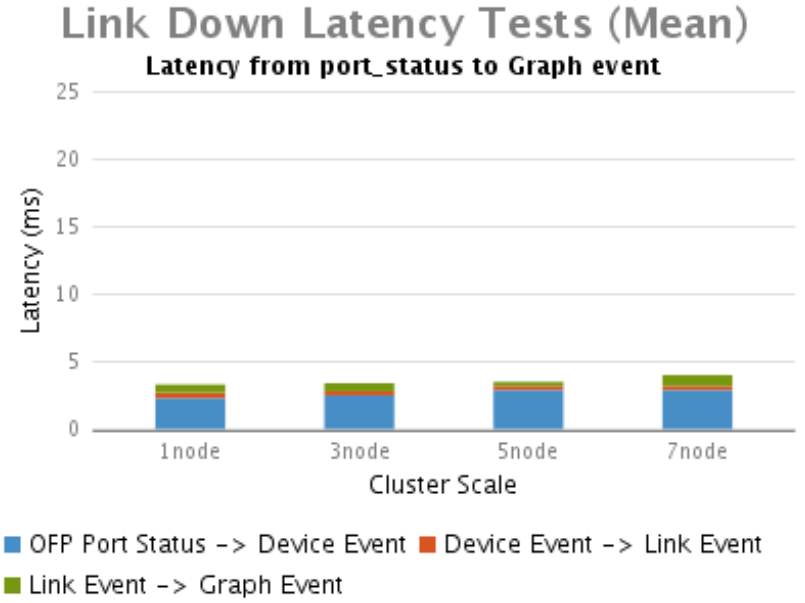
- Allows plugging in custom timestamp mechanism

Device & Link Sensing Latency



Link Up/Down Latency





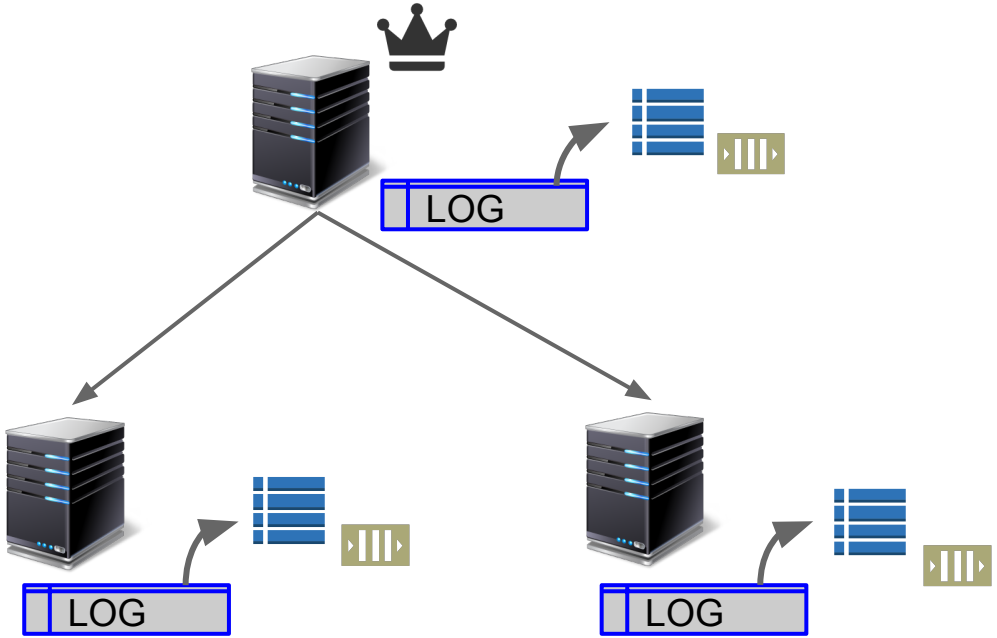
- Since we use LLDP & BDDP to discover links, it takes longer to discover a link coming up than going down

- Port down event trigger immediate teardown of the link.

Next topic: Strong consistency

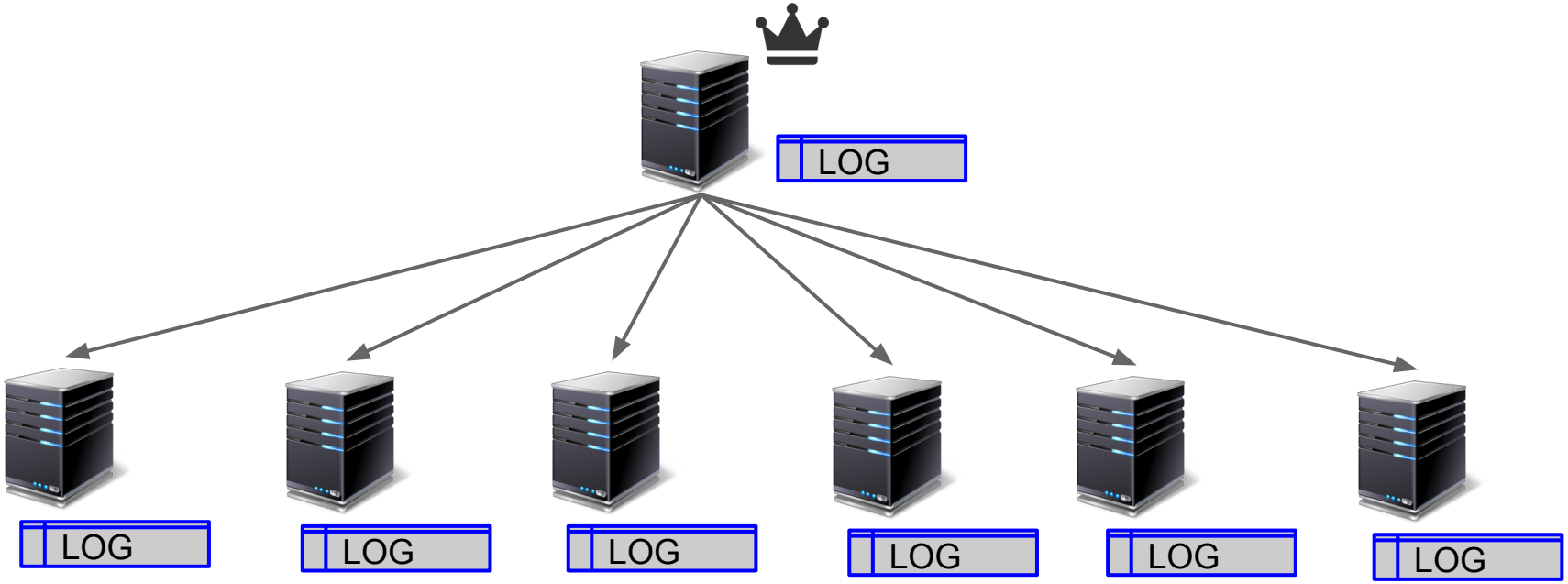
- **EventualConsistency** is necessary but not sufficient
- Some state requires stronger guarantees
  - Reservations is a prime example
- Hard to layer abstractions when you a have a store that only provides weak guarantees

Strong Consistency via Consensus

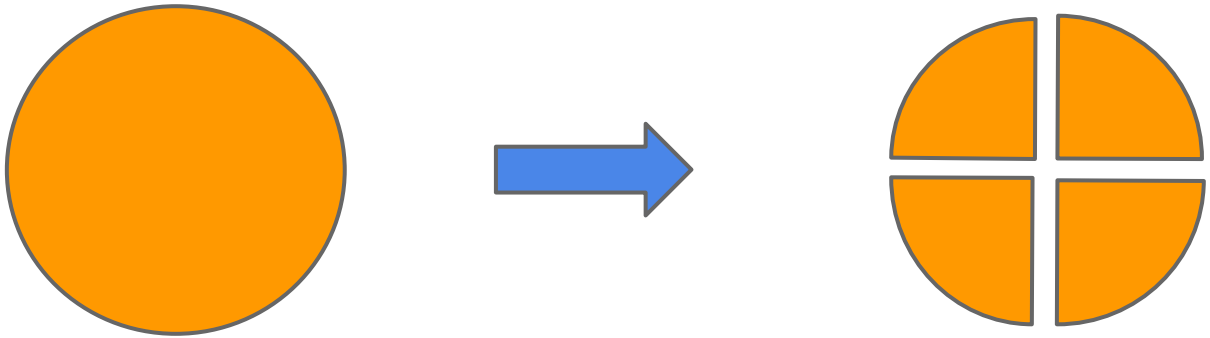


Replicated State Machine

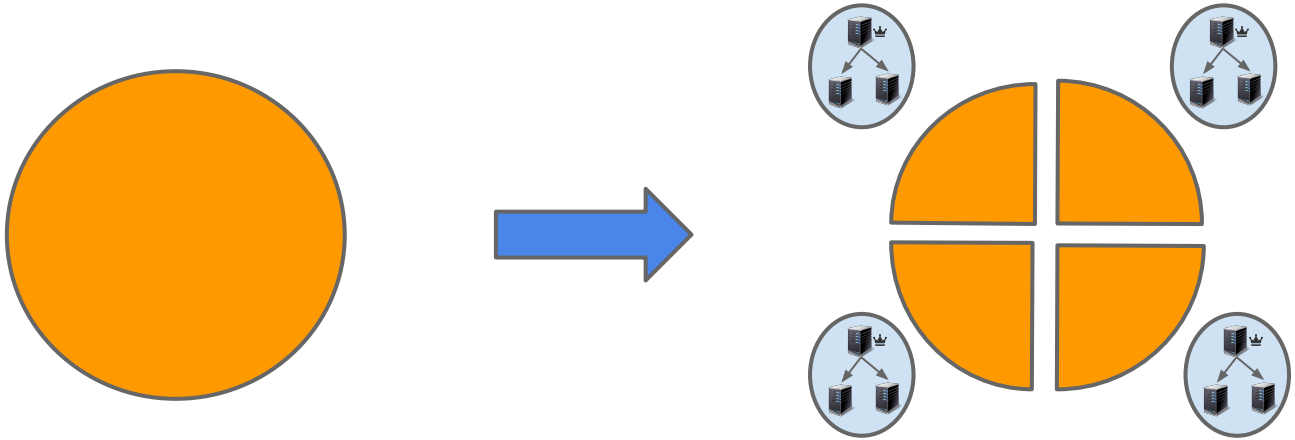
Strong Consistency via Consensus



Scaling Strong Consistency via Partitioning



Scaling Strong Consistency via Partitioning



Distributed Primitives

- ConsistentMap<K, V>
- TransactionalMap<K, V>

Provides transactional semantics for single and group updates

Other Distributed Primitives

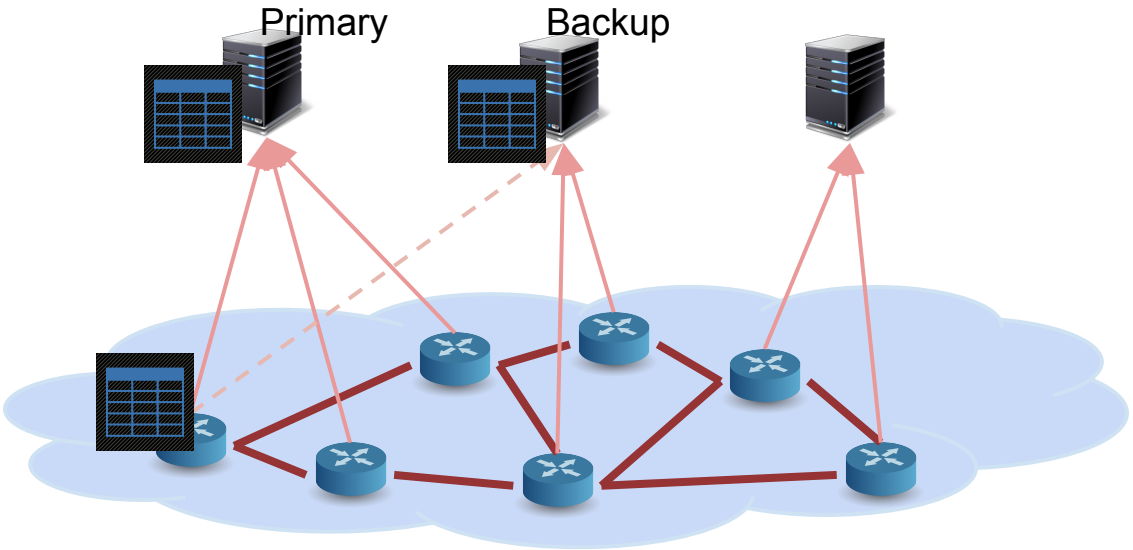
- DistributedSet<E>
- DistributedQueue<E>
- AtomicCounter

Final example: Flows

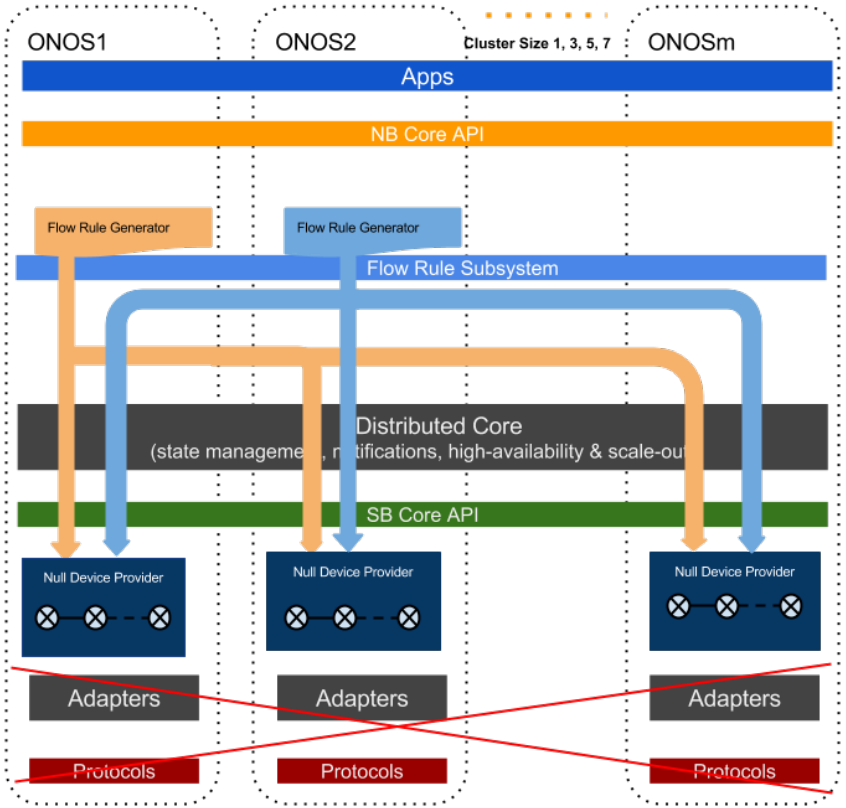
- Forwarding plane rules
- Programmed onto individual switches
- Control plane is the arbiter of truth

Flows

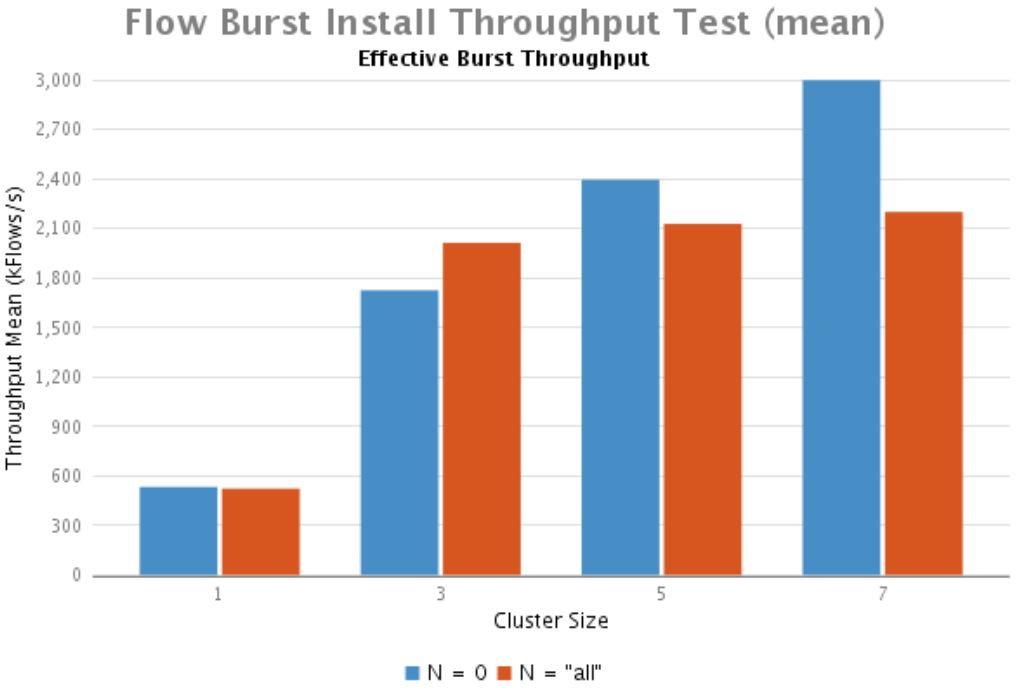
- Primary-backup replication
- Switch master is *primary* and **next** master is *backup*
- Fast read/write access
- Minimal overhead on failover



Flow Rule Operations Throughput

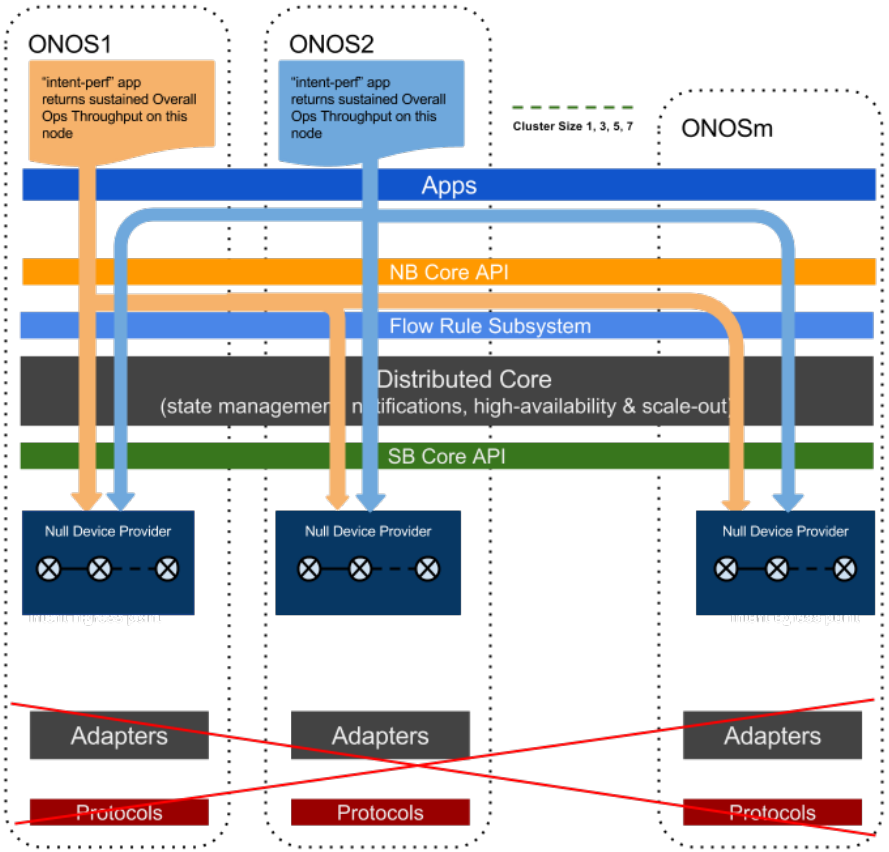


Flow Throughput results

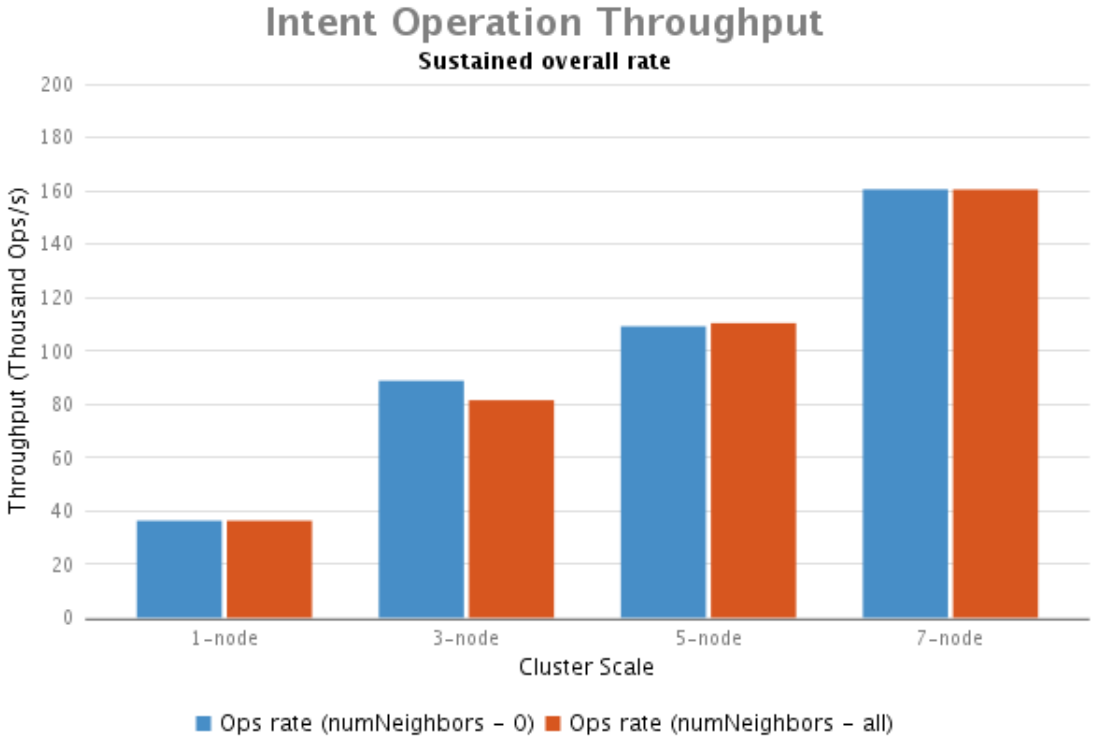


- Single instance can install over 500K flows per second
- ONOS can handle 3M local and 2M non local flow installations
- With 1-3 ONOS instances, the flow setup rate remains constant no matter how many neighbours are involved
- With more than 3 instances injecting load the flow performance drops off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

Intent Throughput Experiment

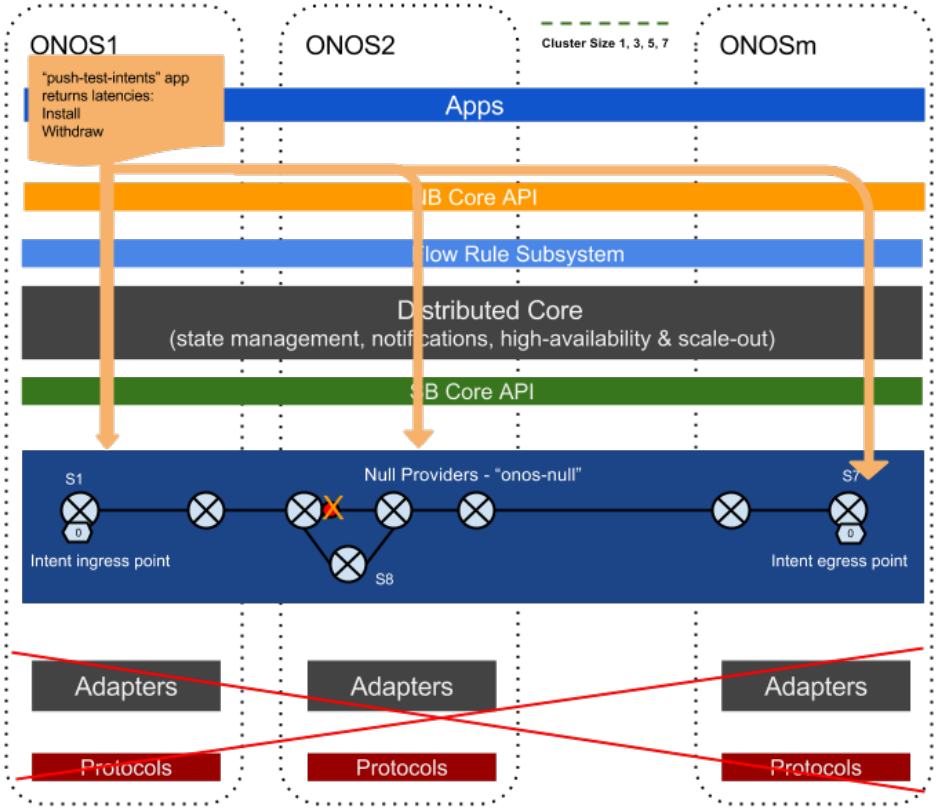


Intent Throughput Results

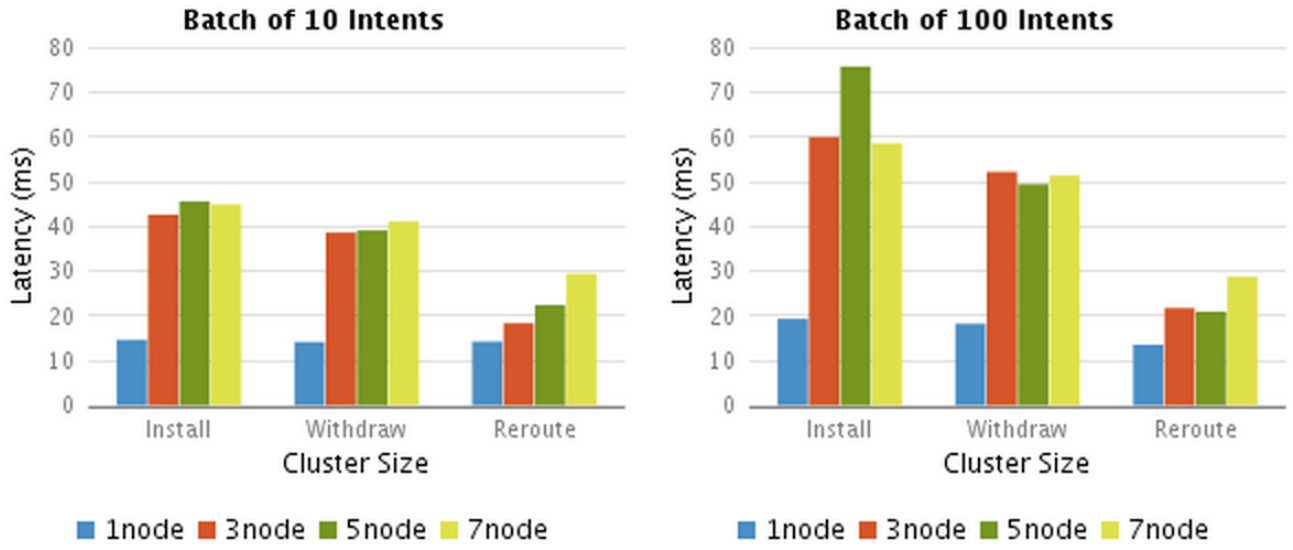


- Processing clearly scales as cluster size increases

Intent Latency Experiment

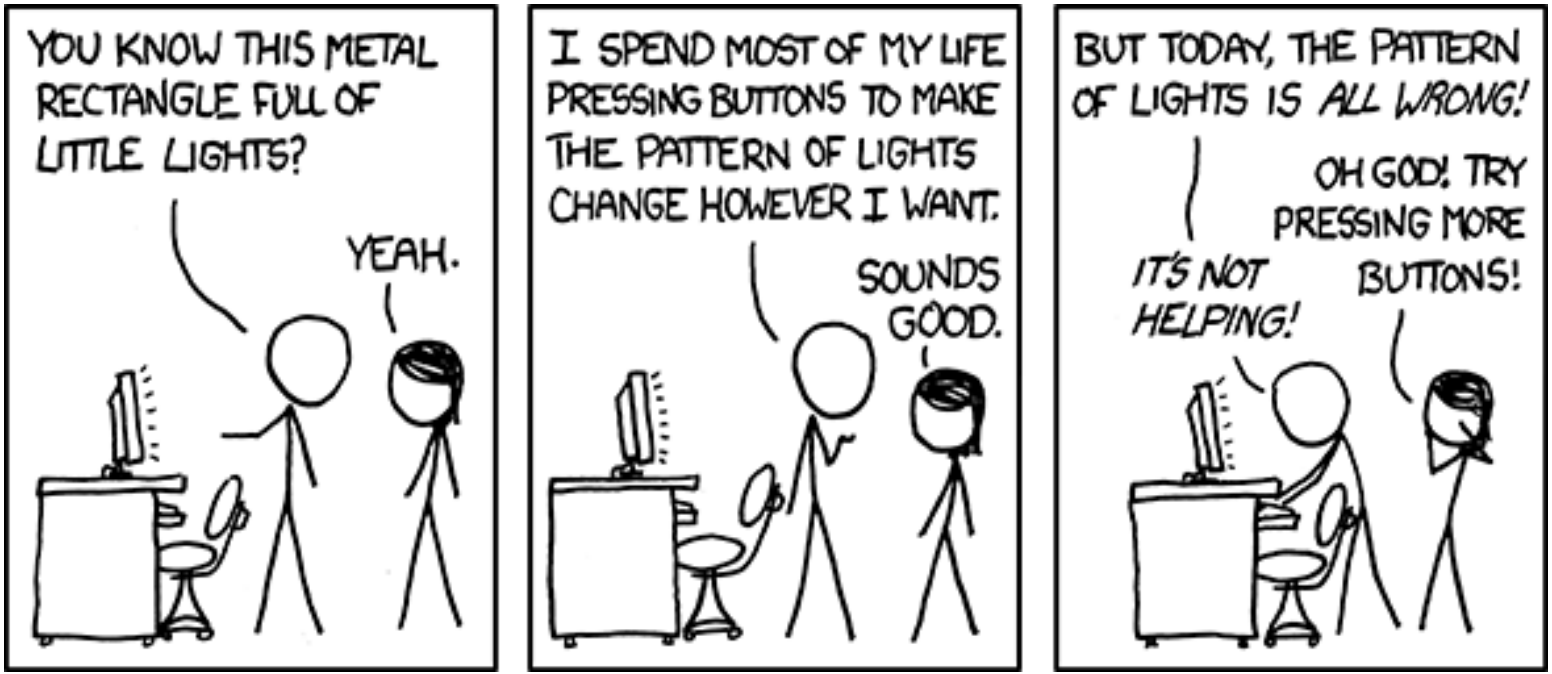


Intent Latency Results



- 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

Lunch Break



xkcd.com

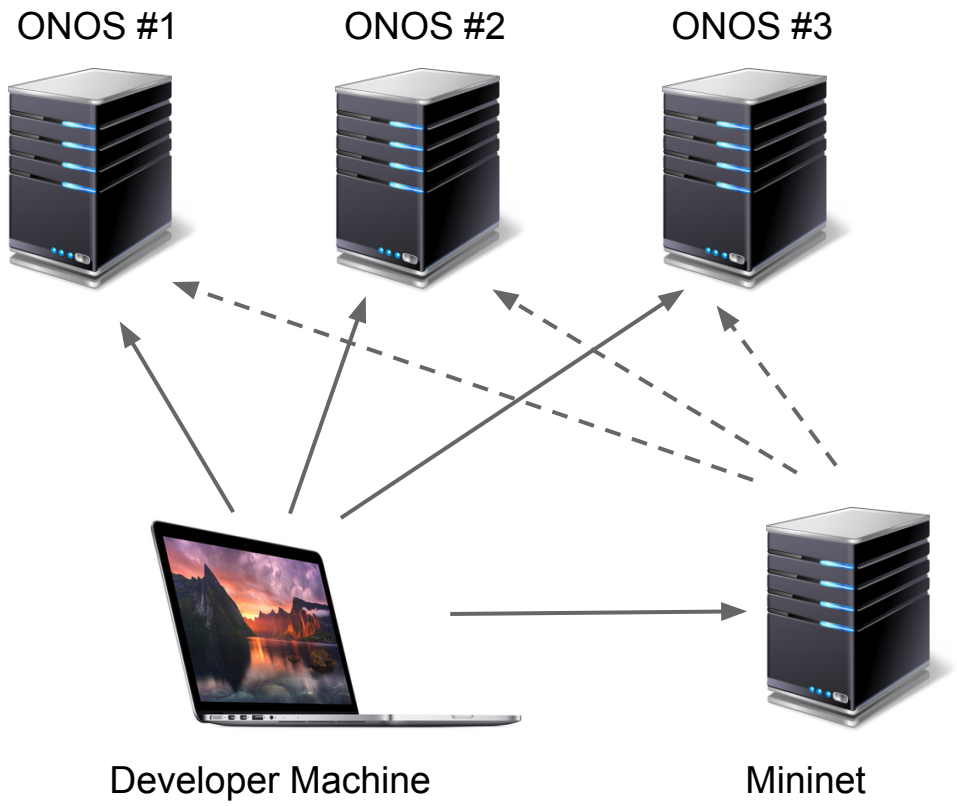
ONOS Dev Workshop Outline

- *ONOS Fundamentals*
  - *architecture, checking out, building & running ONOS*
- *ONOS Distributed Core*
  - *OSGi, anatomy of a core subsystem, distributed stores & primitives*
- ONOS App Development
  - background, Maven archetypes, app deployment on ONOS cluster
- ONOS Interfaces
  - component config service, flow objective service, intent service
  - CLI, REST and GUI overview
- Wrap-up

ONOS App Development 13:15-14:45

- Installing ONOS cluster from onos.tar.gz (~10)
  - untar, `onos-service`, `onos-form-cluster`
- ONOS Applications (~10)
  - OSGi bundles, Karaf features, ONOS apps & OAR files
  - application lifecycle - install, activate, deactivate, uninstall
  - ONOS CLI & GUI
- Developing ONOS apps (~50)
  - use `mvn archetype:generate` and `onos-create-app`
  - archetype overlays for CLI and UI - generated & pre-canned
  - iterative and demonstrating use of `onos-app` tool or GUI

Working with ONOS cluster



Install ONOS cluster from tar.gz

- Install ONOS on all three instances
  - we will use iTerm to do so simultaneously

```
$ onos-cell demo && onos-iterm-cli
```

- Fetch ONOS bits via wget

```
$ wget http://downloads.onosproject.org/onos-1.3.0.dev.tar.gz
```

- Untar bits

```
$ tar zxf onos-1.3.0.dev.tar.gz
```

Start ONOS instances

- Start ONOS

```
$ cd onos-1.3.0.dev
$ bin/onos-service server &
```

- Verify install as individual instances

```
$ bin/onos
onos> nodes
```

- Check logs

```
$ tail -f log/karaf.log
```

Form ONOS cluster

- Form ONOS cluster from individual instances

```
$ bin/onos-form-cluster 10.128.11.1 10.128.11.2 10.128.11.3
```

- Verify install as a cluster

```
$ bin/onos
onos> nodes
```

- Check logs

```
$ tail -f log/karaf.log
```

Setup test network

- Connect mininet to cluster machines

```
$ sudo mn --topo tree,3,4 --controller remote,...
```

- Activate proxy ARP and reactive forwarding apps

```
onos> app activate org.onosproject.proxyarp
onos> app activate org.onosproject.fwd
```

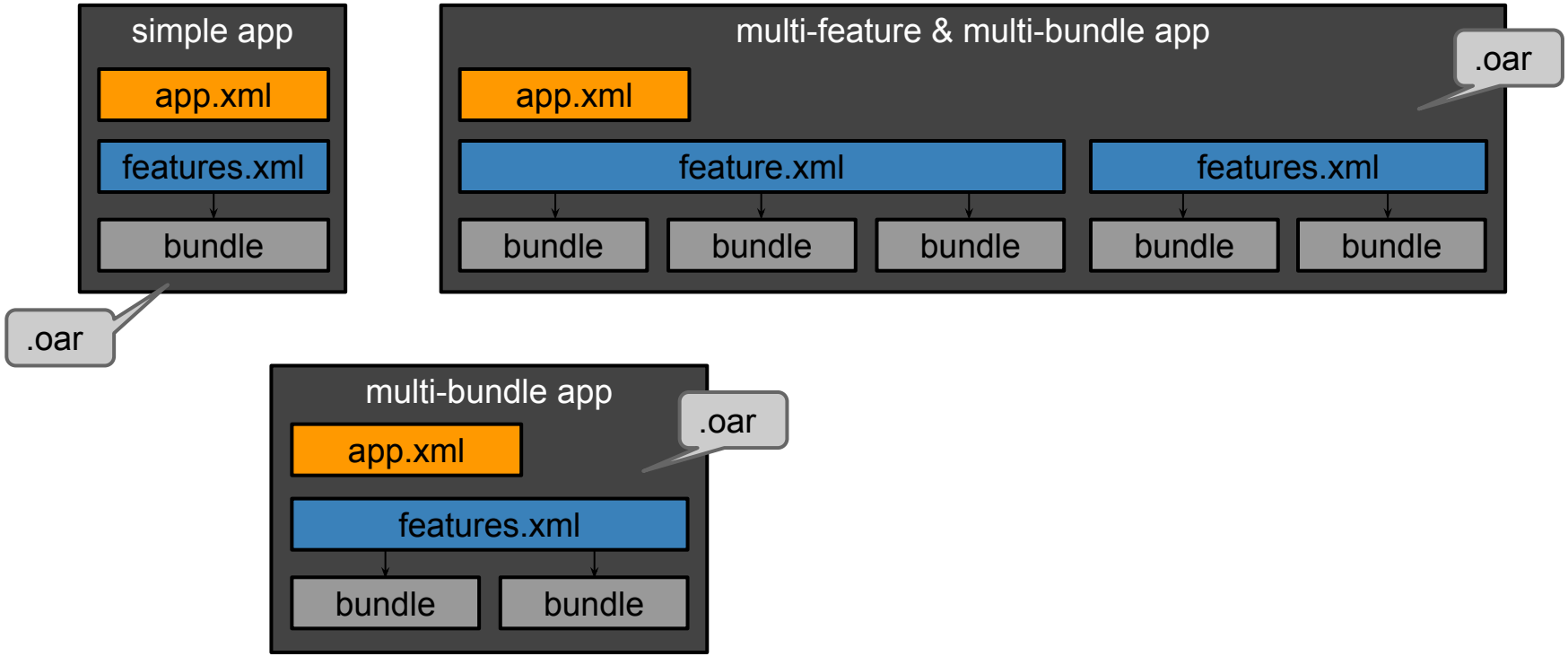
- Issue pingall a few times to show unencumbered flow

```
mininet> pingall
mininet> pingall
```

ONOS App Development 13:15-14:45

- Installing ONOS cluster from onos.tar.gz (~10)
  - untar, `onos-service`, `onos-form-cluster`
- ONOS Applications (~10)
  - OSGi bundles, Karaf features, ONOS apps & OAR files
  - application lifecycle - install, activate, deactivate, uninstall
  - ONOS CLI & GUI
- Developing ONOS apps (~50)
  - use `mvn archetype:generate` and `onos-create-app`
  - archetype overlays for CLI and UI - generated & pre-canned
  - iterative and demonstrating use of `onos-app` tool or GUI

Bundles, Features & ONOS Apps



Bundles, Features & ONOS Apps

- Apps are delivered via ONOS App aRchive (`.oar`) files
  - OAR is a JAR with `app.xml`, `features.xml` and bundle artifacts
  - `onos-maven-plugin` generates an `*.oar` file as part of Maven build
- Apps are managed on the entire ONOS cluster
  - via REST API: `GET|POST|DELETE /onos/v1/applications`
  - via shell tool: `onos-app {install|activate|deactivate|uninstall}`
  - via CLI: `onos:app {install|activate|deactivate|uninstall}`
  - via GUI Applications view
- Back-end installation and activation is done via normal feature & bundle services

Developing ONOS apps

- Maven archetypes
  - `onos-bundle-archetype` - basis for an ONOS bundle or an app
  - `onos-cli-archetype` - overlay for apps with CLI extensions
  - `onos-ui-archetype` - overlay for apps with GUI extensions
  - `onos-api-archetype` - basis for an app Java API bundle
- Run `mvn archetype:generate` to create a working minimal project module
- For simpler usage run `onos-create-app` shell tool
  - simply a convenience wrapper for `mvn`

ONOS App Development 13:15-14:45

- Installing ONOS cluster from onos.tar.gz (~10)
  - untar, `onos-service`, `onos-form-cluster`
- ONOS Applications (~10)
  - OSGi bundles, Karaf features, ONOS apps & OAR files
  - application lifecycle - install, activate, deactivate, uninstall
  - ONOS CLI & GUI
- Developing ONOS apps (~50)
  - use `mvn archetype:generate` and `onos-create-app`
  - archetype overlays for CLI and UI - generated & pre-canned
  - iterative and demonstrating use of `onos-app` tool or GUI

One Ping Only ONOS app

One Ping Only ONOS app

Minimal ONOS app project

- Create minimal app project via `onos-create-app` tool

```
$ onos-create-app app org.oneping oneping 1.0-SNAPSHOT
```

- Build the app via Maven

```
$ cd oneping
$ mvn clean install
```

- Deploy the app to ONOS cluster via `onos-app` tool

```
$ onos-app 10.128.11.1 install! target/oneping*.oar
```

Code & deploy OnePingOnly app

- *“Code”* our app as `OnePing.java`

- Rebuild the app via Maven and redeploy it

```
$ mvn clean install
$ onos-app 10.128.11.1 reinstall! target/oneping*.oar
```

Show one-ping-only enforcement

- Issue one ping between two hosts

```
mininet> h1 ping -c1 h2
mininet> h1 ping -c1 h3
```

- Issue another ping between the same two hosts

```
mininet> h1 ping -c1 h2
```

- Show failed second ping and messages in the logs

ONOS app overlay archetypes

- Some ONOS archetypes can be used to enhance an existing project with additional capabilities
  - add CLI extensions
  - add GUI extensions
  - others in future

- Simply run `onos-create-app` tool in the same project

- First show that sample command does not exist

```
onos> sample
```

Overlay CLI archetype

- Overlay CLI additions via `onos-create-app`

```
$ onos-create-app cli org.oneping oneping 1.0-SNAPSHOT
```

- Rebuild and redeploy the enhanced app

```
$ mvn clean install
$ onos-app 10.128.11.1 reinstall! target/oneping*.oar
```

- Show that sample command now exists

```
onos> sample
```

Overlay GUI archetype

- Show existing GUI navigation menu
- Overlay GUI additions via `onos-create-app` and rebuild

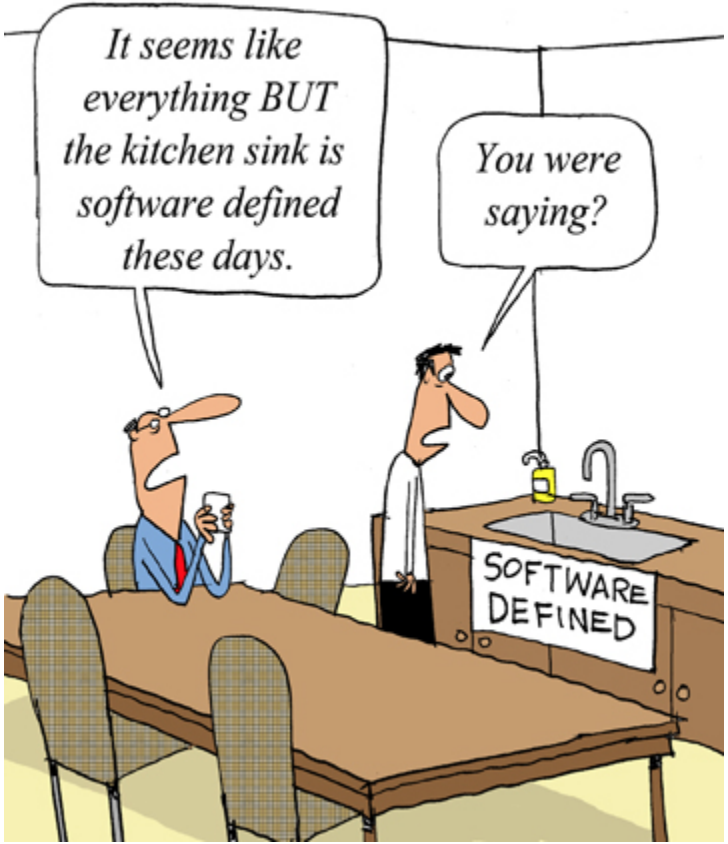
```
$ onos-create-app ui org.oneping oneping 1.0-SNAPSHOT
```

- Rebuild and redeploy the enhanced app

```
$ mvn clean install
$ onos-app 10.128.11.1 reinstall! target/oneping*.oar
```

- Refresh GUI and show enhanced GUI navigation menu

Break

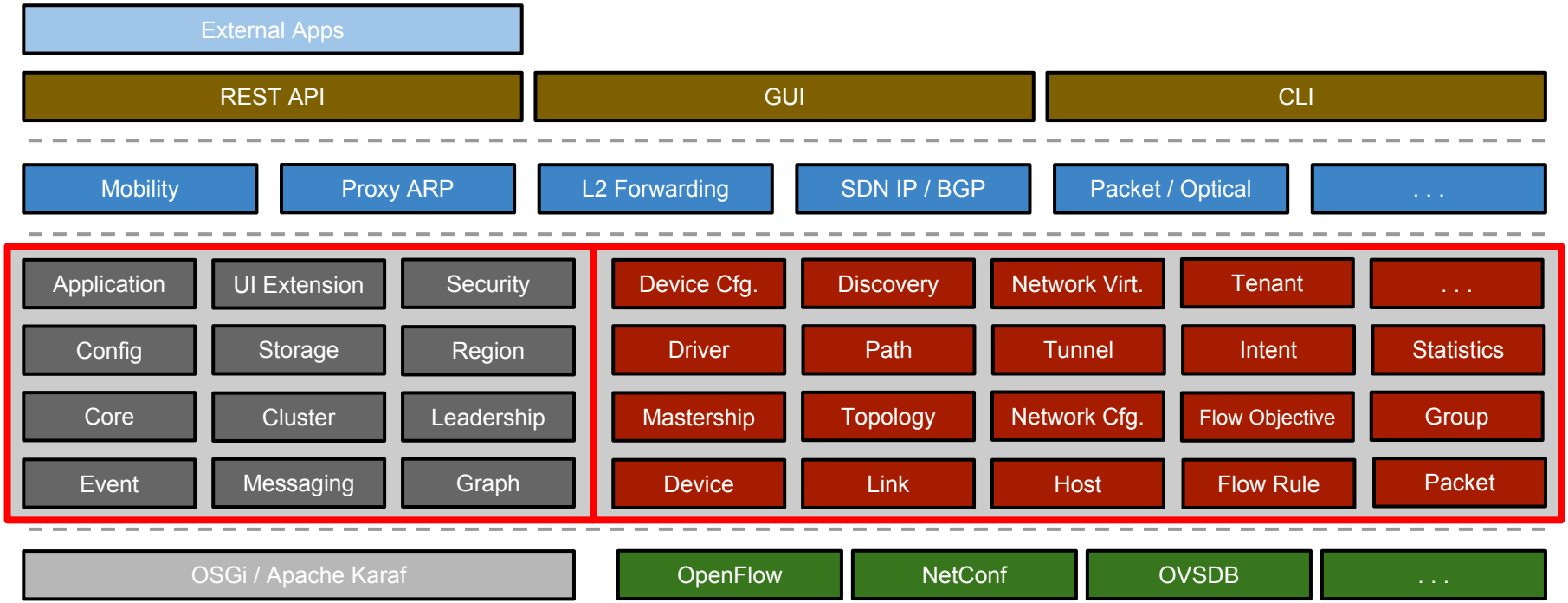

It seems like everything BUT the kitchen sink is software defined these days.
You were saying?
SOFTWARE DEFINED

ONOS Interfaces

15:15-17:00

- Component Config Service (~10)
- Device Drivers Service (~10)
- Flow Objective Service (~15)
- Intent Service (~15)
- GUI overview (~15)
- CLI overview (~5)
- REST API overview (~5)
- Wrap-up & Final QA (~15)

ONOS Core Subsystems



ONOS Interfaces

15:15-17:00

- Component Config Service (~10)
- Device Drivers Service (~10)
- Flow Objective Service (~15)
- Intent Service (~15)
- GUI overview (~15)
- CLI overview (~5)
- REST API overview (~5)
- Wrap-up & Final QA (~15)

Component Config Subsystem

- Provide centralized and cluster-wide component configuration
- Avoid drift between configurations on cluster instances
- Built atop the OSGi `ConfigurationAdmin` service
- Components accept configuration via standard OSGi `@Modified` hook
- Catalogue of config properties for each component generated at build-time via `onos-maven-plugin`

Component Config Subsystem

- Component registers its properties on activation

```
void registerProperties(getClass());
```

- Component unregisters its properties on deactivation

```
void unregisterProperties(getClass(), false);
```

- Component provides configuration method hook

```
@Modified
public void modified(ComponentContext context) { … }
```

ONOS Interfaces

15:15-17:00

- Component Config Service (~10)
- Device Drivers Service (~10)
- Flow Objective Service (~15)
- Intent Service (~15)
- GUI overview (~15)
- CLI overview (~5)
- REST API overview (~5)
- Wrap-up & Final QA (~15)

Device Driver Subsystem

- Isolate device-specific code
- Allow asynchronous delivery & dynamic loading
  - by ON.Lab, vendors or others
- Allow abstracting of how to interact with devices
  - via Java API (`Behaviour` interfaces)
- Avoid a monolithic driver approach
  - different facets of behaviour can come from different sources
- Facilitate specialization via inheritance
  - families of devices share many similar characteristics

Driver

- Representation of a specific family of devices or a specific device
  - has a unique name, e.g. `ciena-6500`
  - supports set of `Behaviour` classes
  - may *inherit* behaviours from another `Driver`

- `DefaultDriver` is a default implementation class

DriverAdminService

- Service for tracking driver providers

  ```
  Set<DriverProvider> getProviders()
  registerProvider(DriverProvider)
  unregisterProvider(DriverProvider)
  ```

- `DriverProvider` provides a set of driver definitions

  ```
  Set<Driver> getDrivers()
  ```

DriverService

Service for locating appropriate drivers for the device

- by driver name
- by manufacturer, H/W version & S/W version patterns
- by supported behaviour
- by device ID

Behaviour & HandlerBehaviour

- `Behaviour` is an interface for talking about a device

- `HandlerBehaviour` is an interface for interacting with a device
  - `OpenFlowController`
  - `OpenFlowPipeliner`
  - `...`

DriverData & DriverHandler

- `DriverData` is a container for data about a device
  - provides behaviours for talking about a device
  - has parent driver

- `DriverHandler` is a context for interacting with a device
  - provides behaviours for interacting with a device
  - has `DriverData`
  - has parent driver

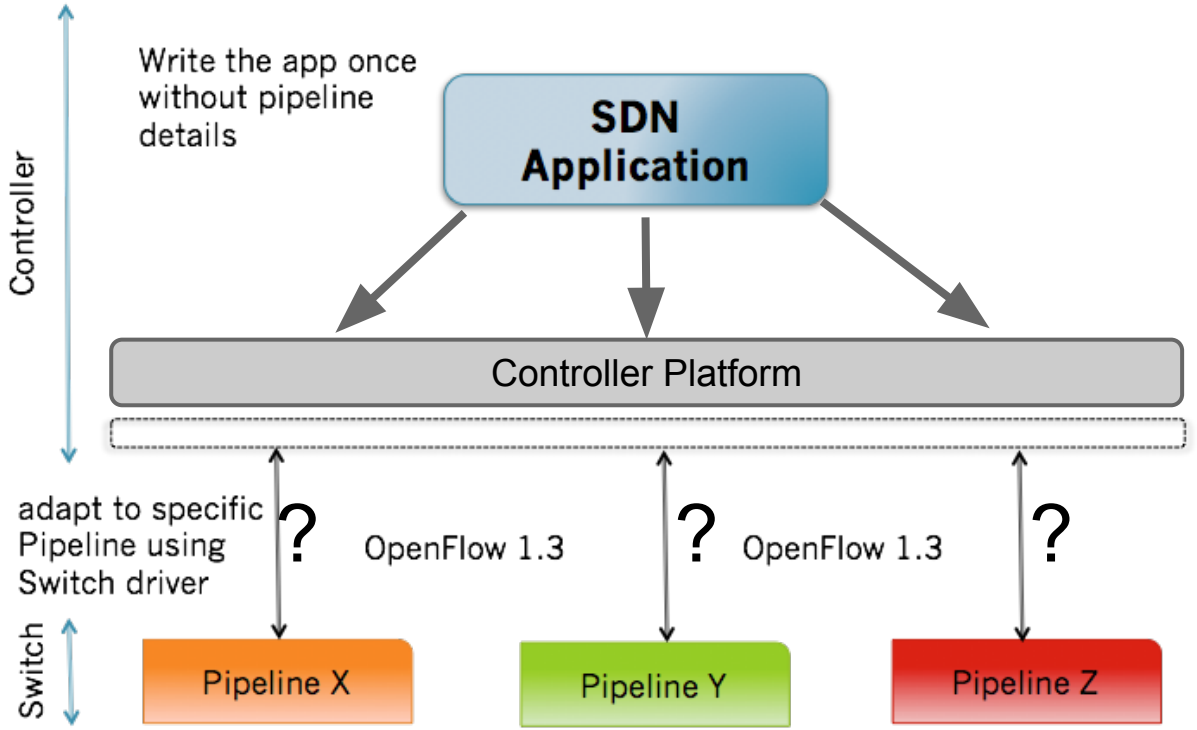
ONOS Interfaces

15:15-17:00

- Component Config Service (~10)
- Device Drivers Service (~10)
- Flow Objective Service (~15)
- Intent Service (~15)
- GUI overview (~15)
- CLI overview (~5)
- REST API overview (~5)
- Wrap-up & Final QA (~15)

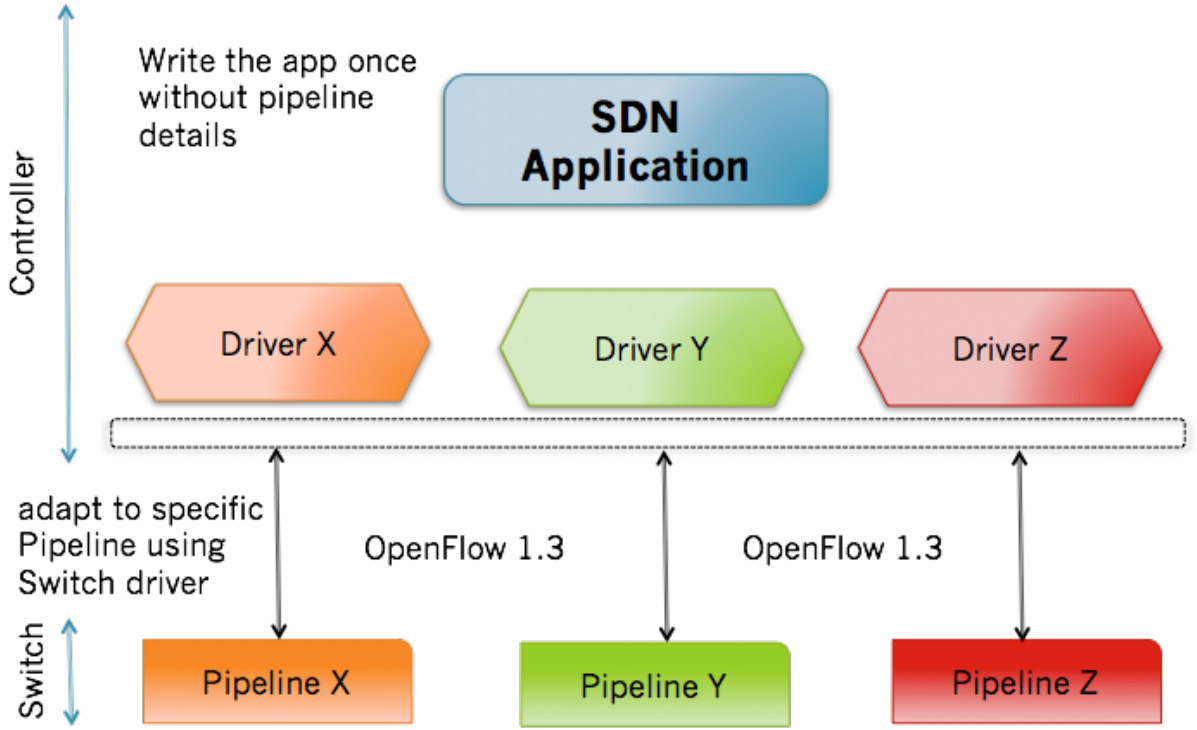
Flow Objective Subsystem

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



Flow Objective Subsystem

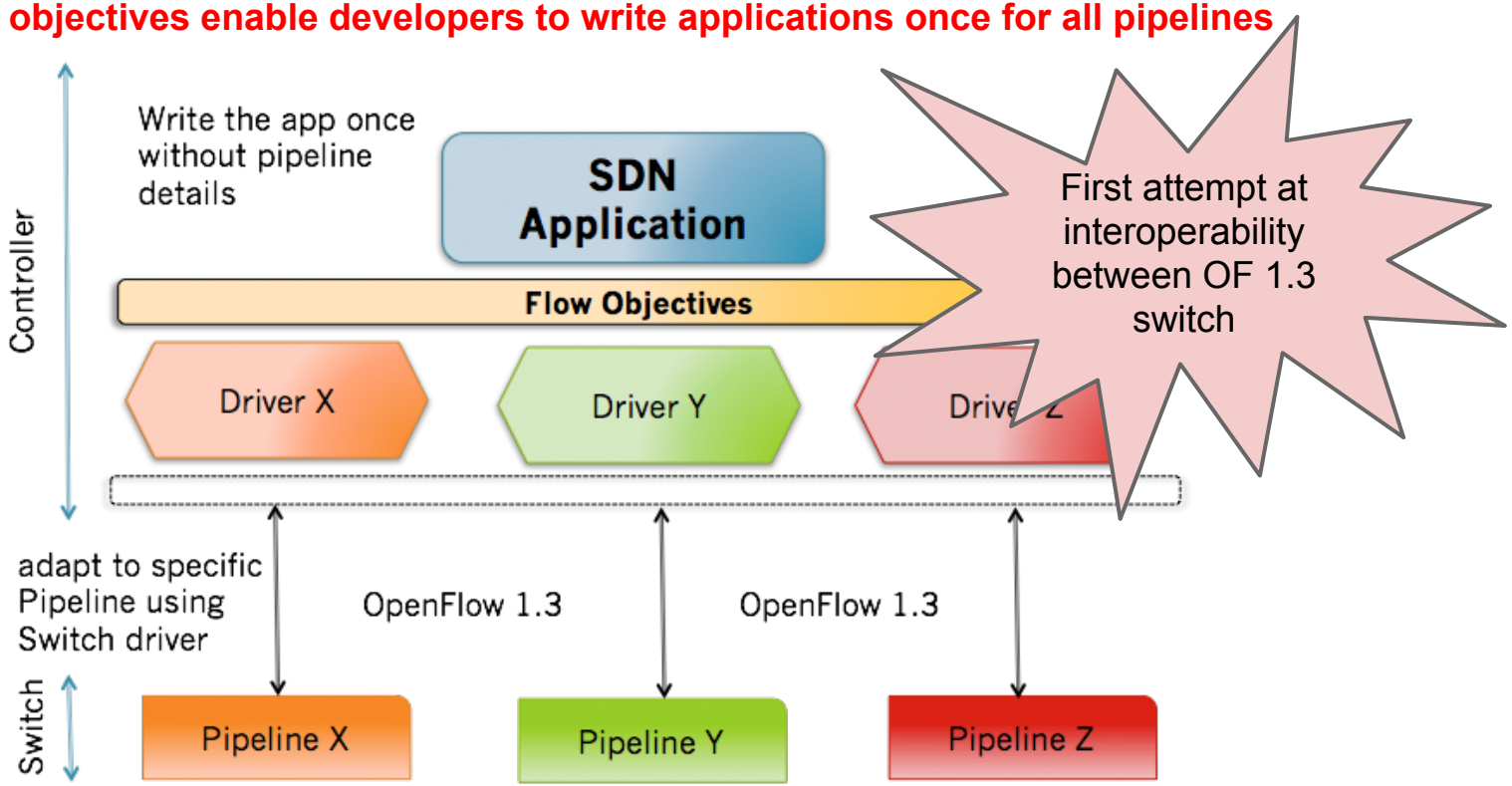
- **Problem:** Applications today must be pipeline aware, effectively making them applicable to specific HW.



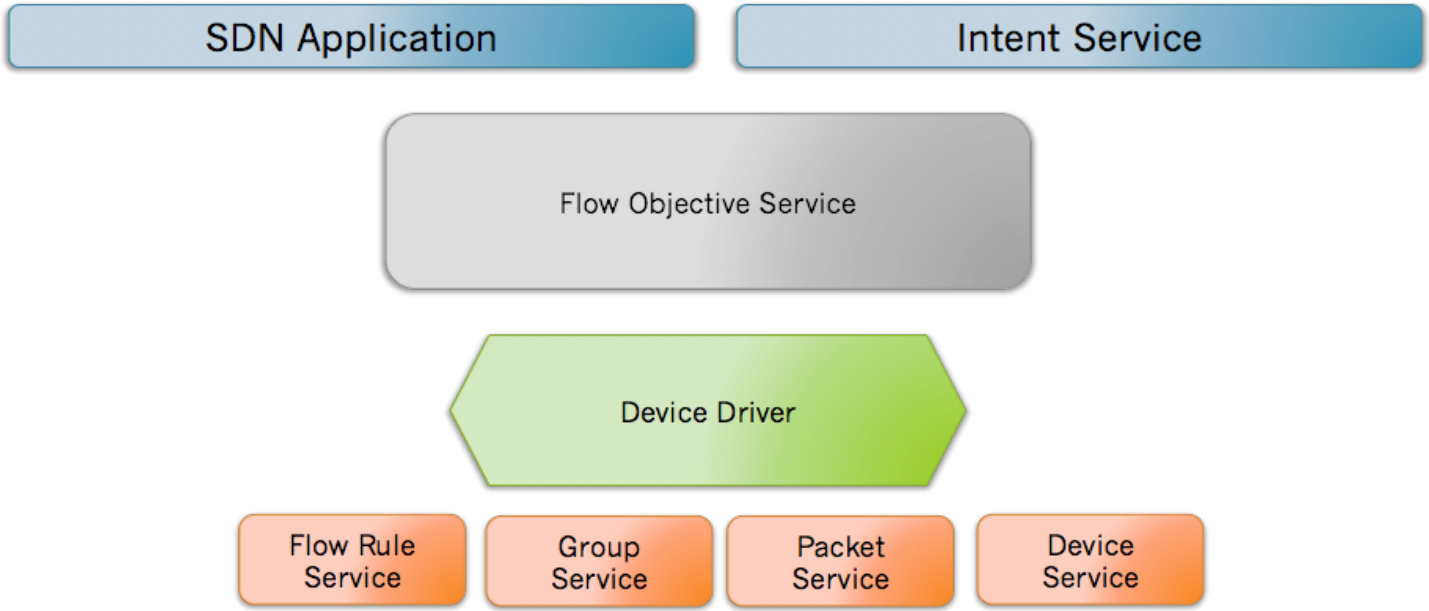
Flow Objective Abstraction

- **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

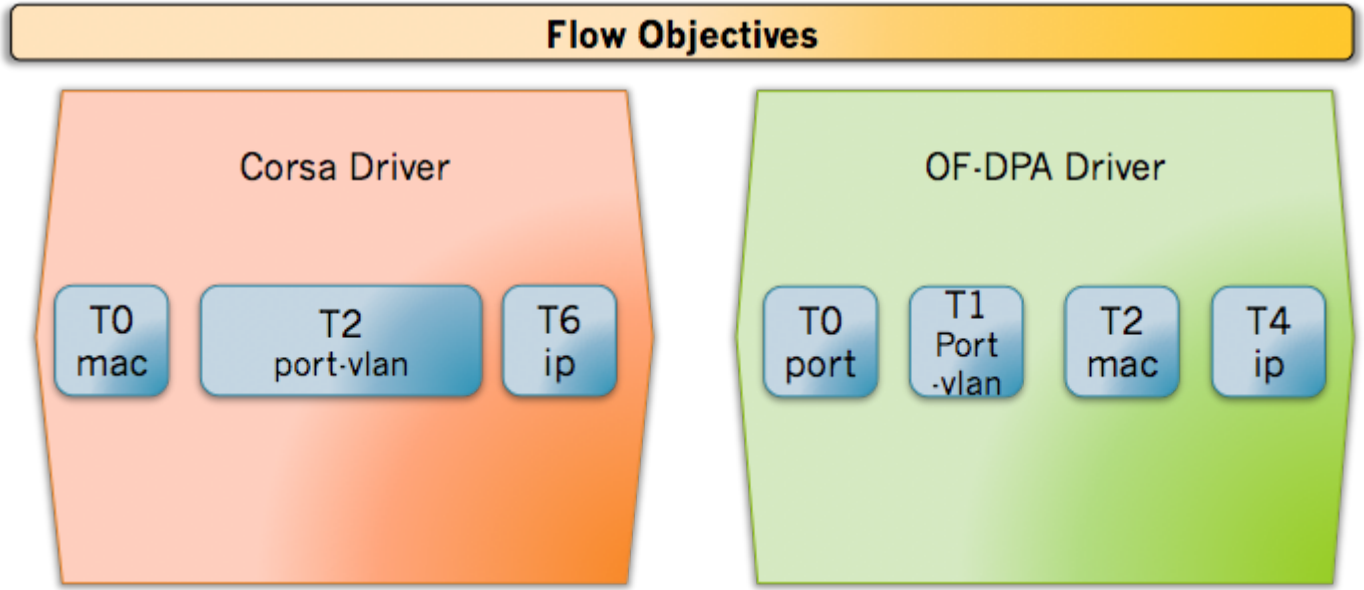
Filtering Objective

- 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



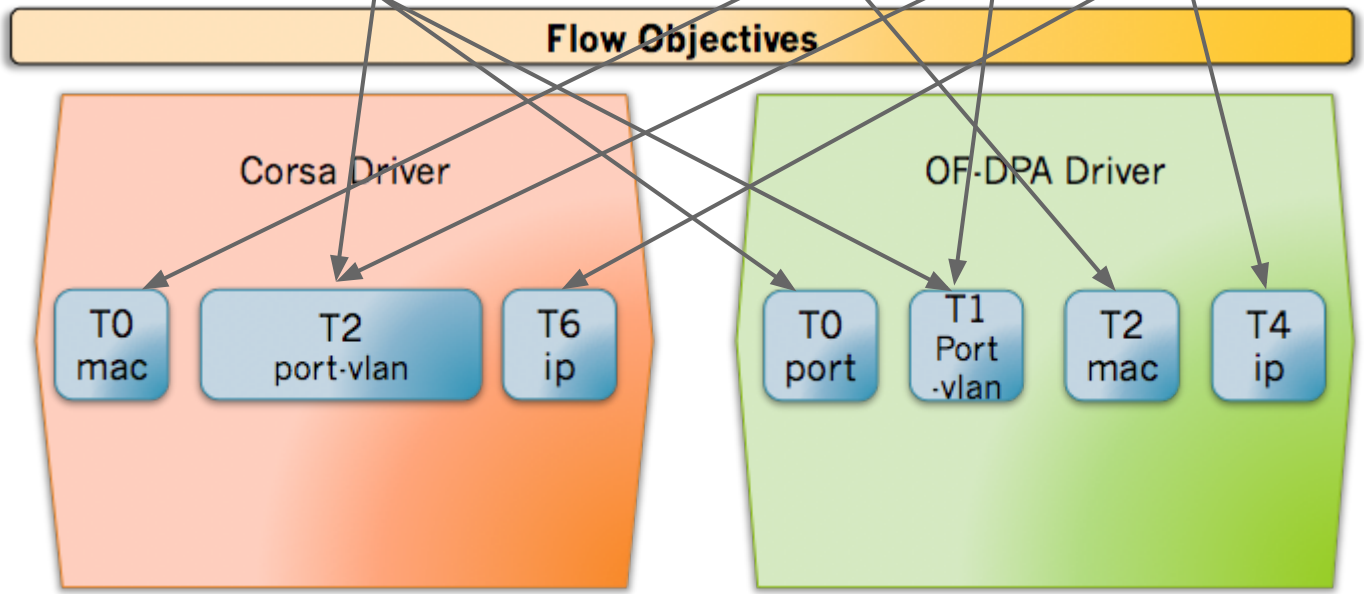
Filtering Objective

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

Example:
Peering Router

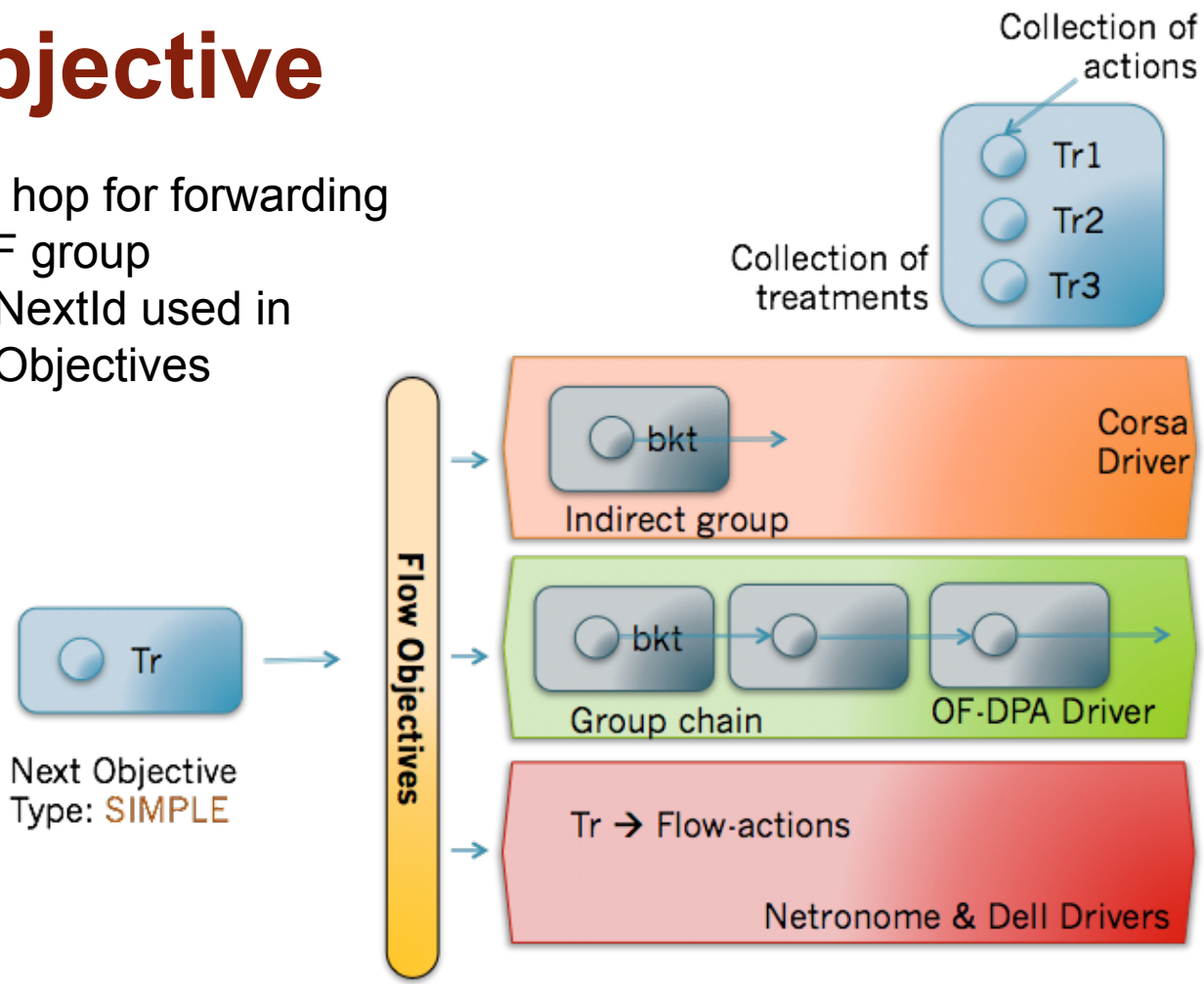
Switch Port : **X**

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

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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

ONOS Interfaces

15:15-17:00

- Component Config Service (~10)
- Device Drivers Service (~10)
- Flow Objective Service (~15)
- Intent Service (~15)
- GUI overview (~15)
- CLI overview (~5)
- REST API overview (~5)
- Wrap-up & Final QA (~15)

Building Network Applications

Objective: Connect Host 1 and Host 2

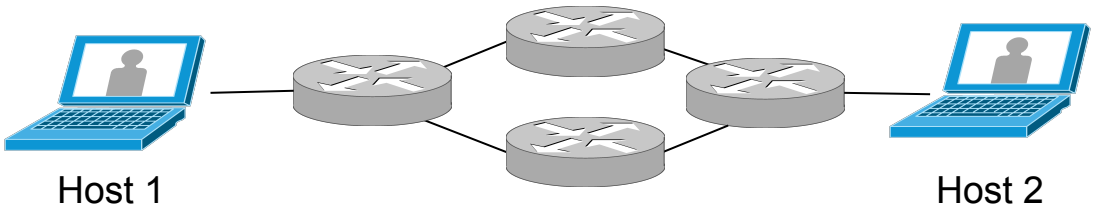
Host 1

Host 2

Building Network Applications

Objective: Connect Host 1 and Host 2

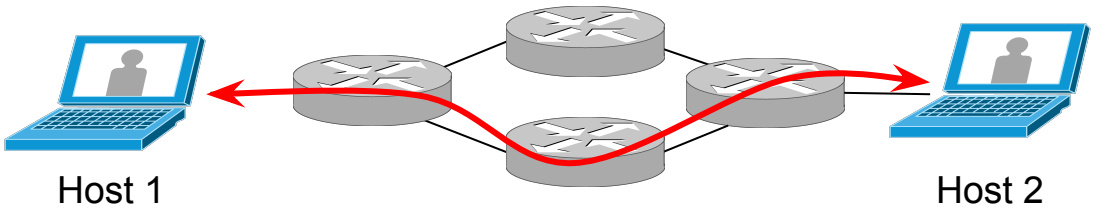
1. Read/discover the topology



Building Network Applications

Objective: Connect Host 1 and Host 2

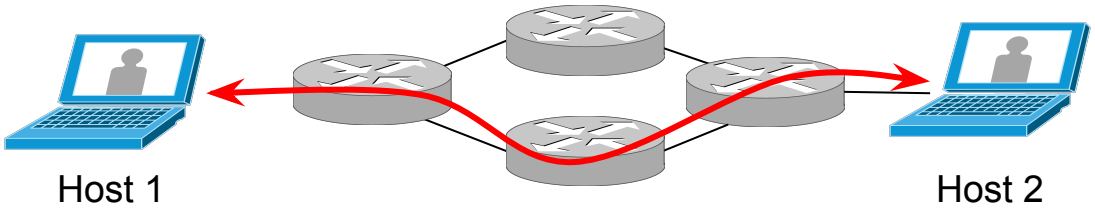
1. Read/discover the topology
2. Compute a path



Building Network Applications

Objective: Connect Host 1 and Host 2

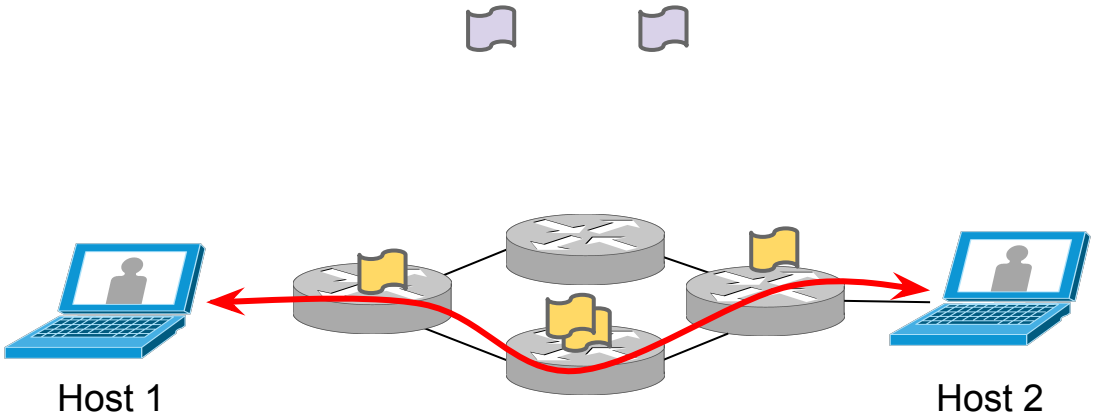
1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device



Building Network Applications

Objective: Connect Host 1 and Host 2

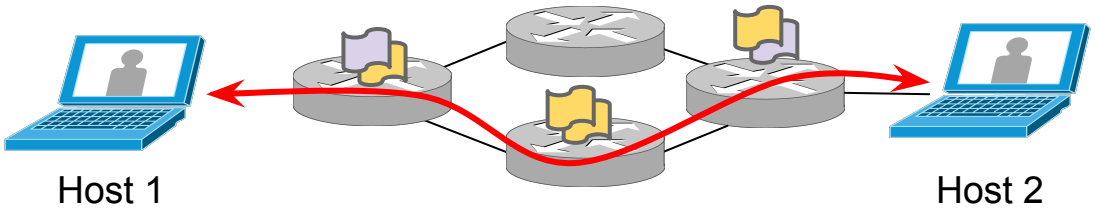
1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device
4. Install rules in consistent way



Building Network Applications

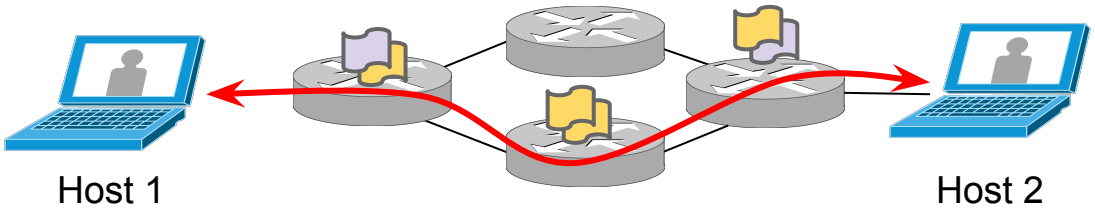
Objective: Connect Host 1 and Host 2

1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device
4. Install rules in consistent way



Building Network Applications

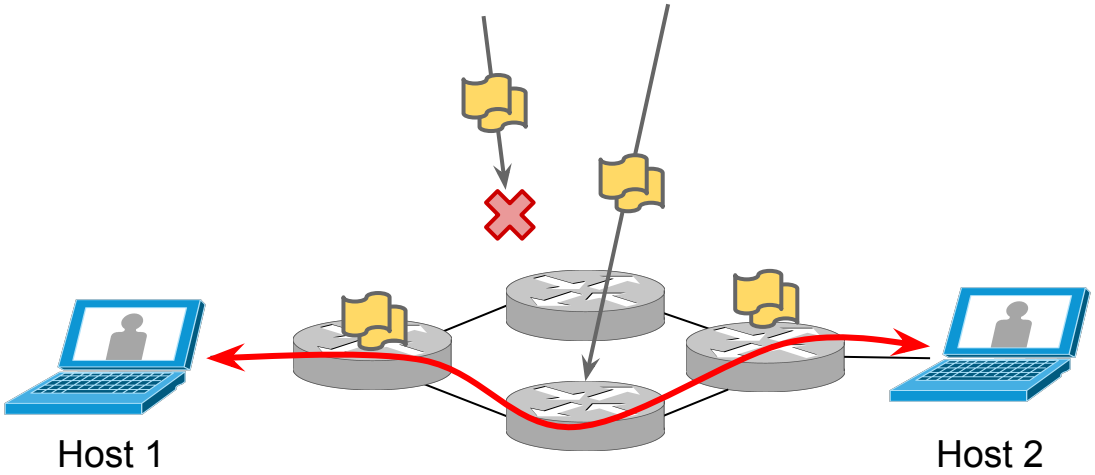
What can go wrong?



Building Network Applications

What can go wrong?

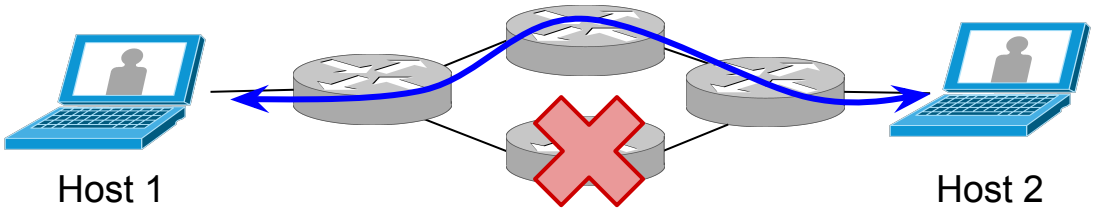
- Missing / rejected / dropped rules
  - Monitor devices connections
  - Send barriers between rule updates
  - Poll flow state



Building Network Applications

What can go wrong?

- Missing / rejected / dropped rules
  - Monitor devices connections
  - Send barriers between rule updates
  - Poll flow state
- Topology changes
  - Listen to switch, port, link and host events
  - Compute new path that leverage or remove old flows



Building Network Applications

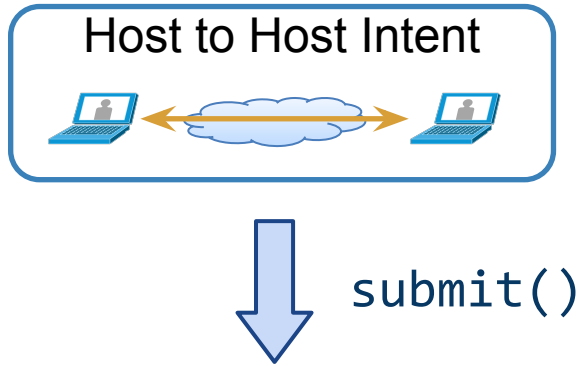
- Each application requires complex path computation and rule installation engines and state machines
- Inconsistent behavior in the face of failures
  - Failures may be handled in different ways (or not at all)
- Bugs need to fixed in multiple places (applications)
- Expensive to upgrade/refactor behavior 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 Subsystem

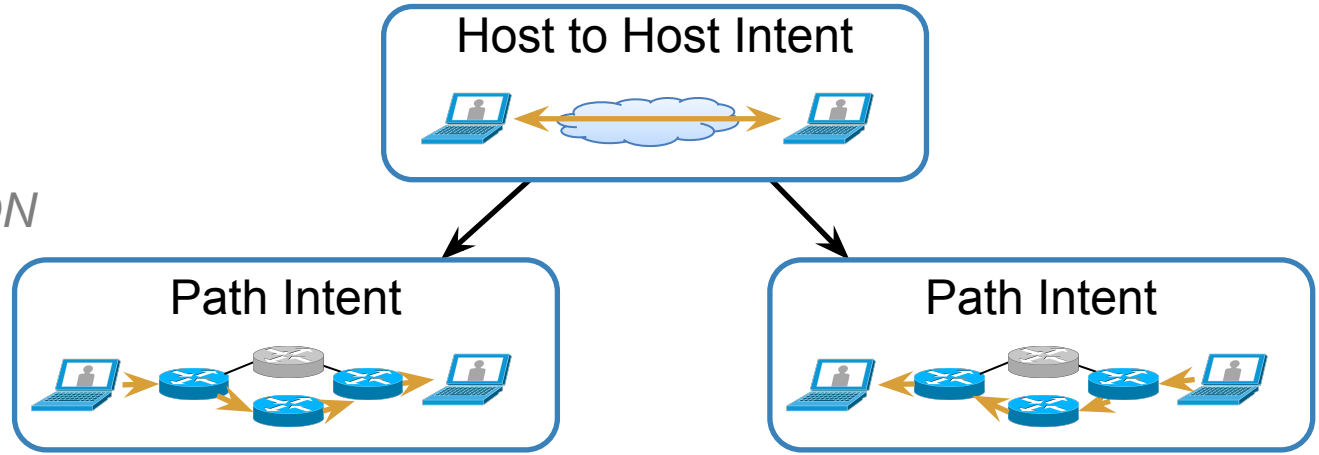
- Provides high-level, network-centric interface that focuses on what should be done rather than how it is specifically programmed
- Abstracts unnecessary network complexity from applications
- Maintains requested semantics as network changes
- High availability, scalability and high performance

Intent Example

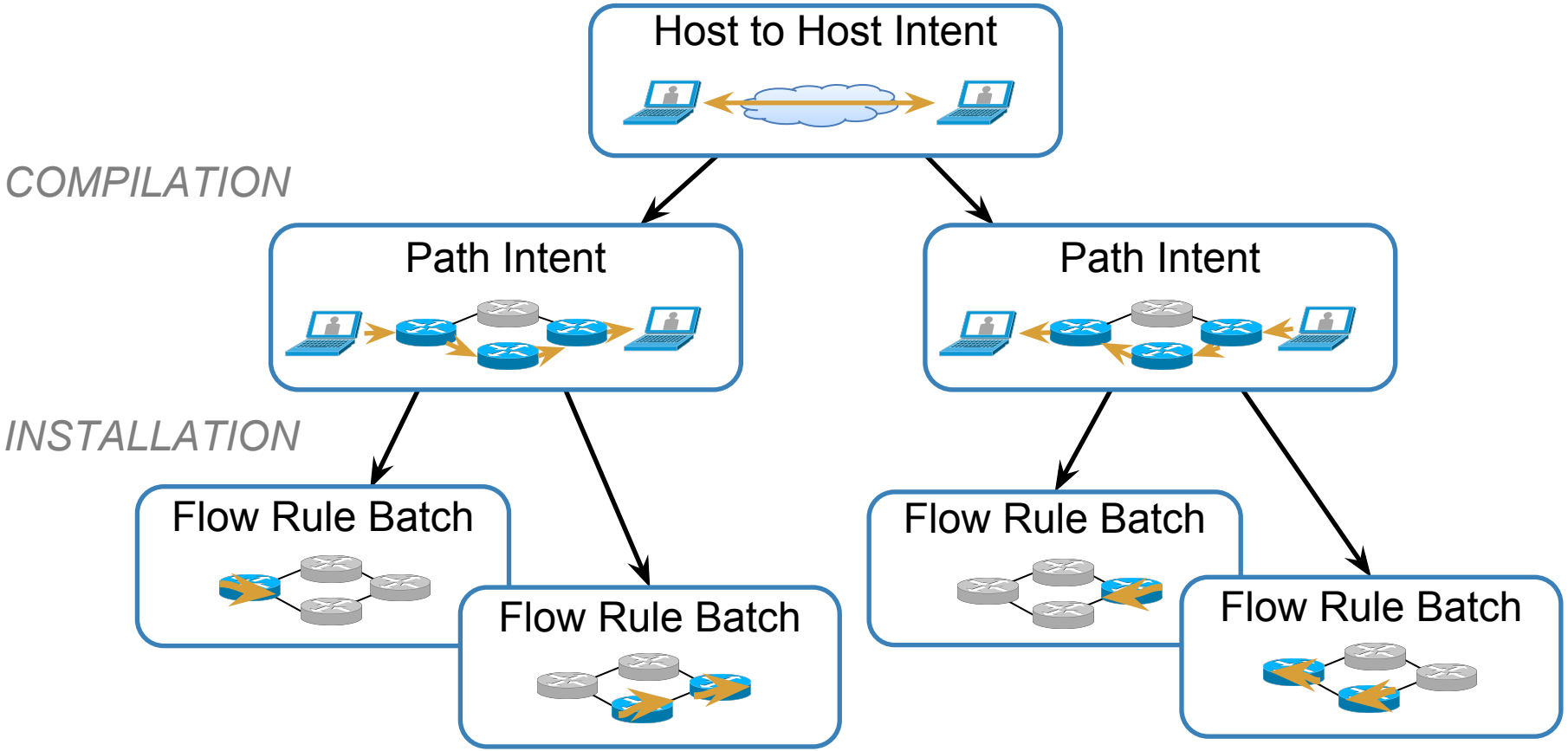
Intent Example



Intent Example



Intent Example



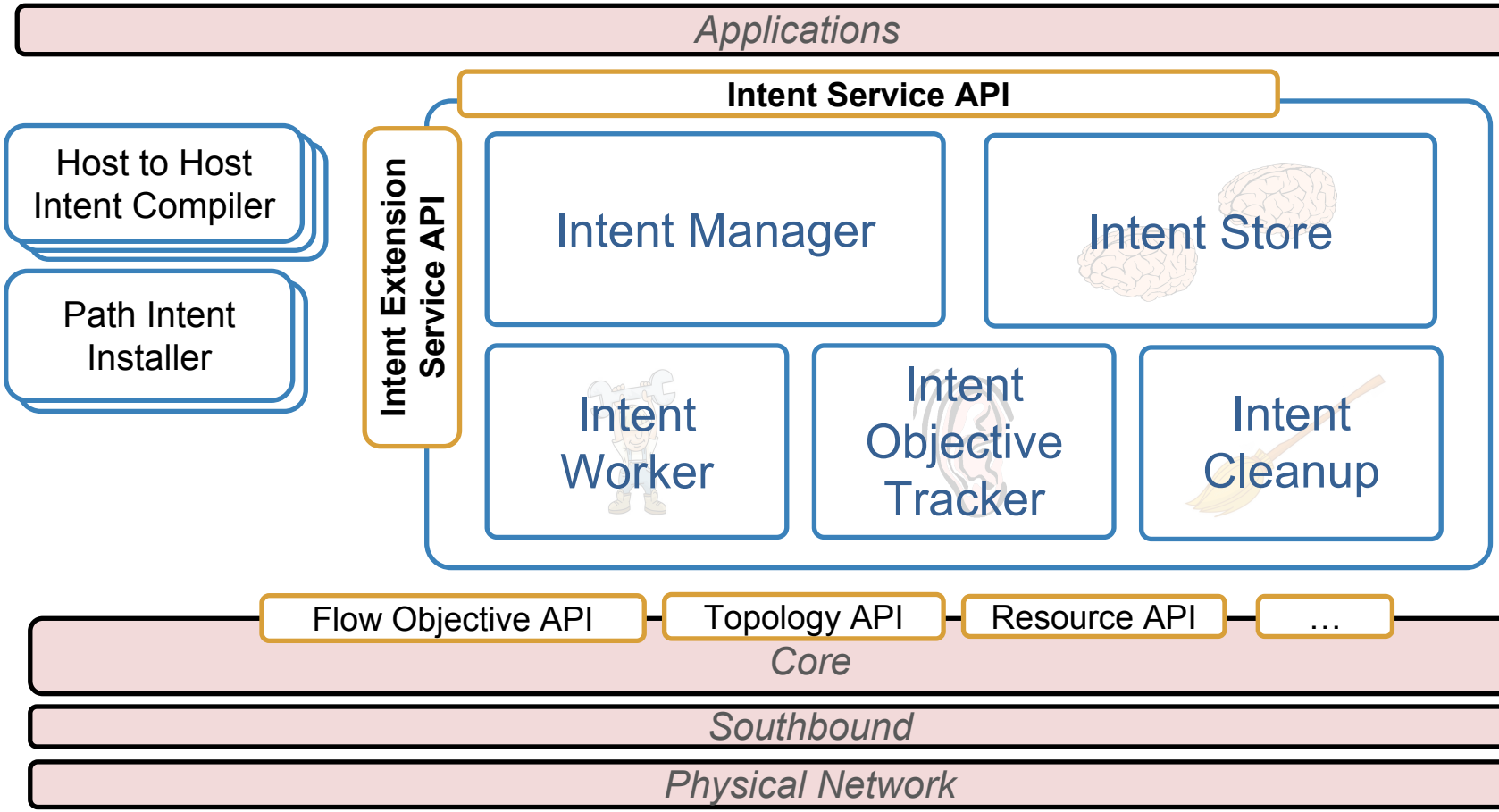
Intent Compilers

- Produce more specific Intents given the environment
- Works on high-level network objects, rather than device specifics
- “Stateless” – free from HA / distribution concerns

Intent Installers

- Transform Intents into device commands
- Allocate / deallocate network resources
- Define scheduling dependencies for workers
- “Stateless”

Intent Framework Design



ONOS Interfaces

15:15-17:00

- Component Config Service (~10)
- Device Drivers Service (~10)
- Flow Objective Service (~15)
- Intent Service (~15)
- GUI overview (~15)
- CLI overview (~5)
- REST API overview (~5)
- Wrap-up & Final QA (~15)

GUI Overview

- Single page application at `/onos/ui`
  - AngularJS
  - d3.js (Data-Driven Documents)
  - SVG
  - Web-Socket
- Shared web-socket for asynchronous & bidirectional exchange
- REST also used for select few interactions
- Dynamically extensible navigation & views
  - apps can contribute new views at run-time

ONOS Interfaces

15:15-17:00

- Component Config Service (~10)
- Device Drivers Service (~10)
- Flow Objective Service (~15)
- Intent Service (~15)
- GUI overview (~15)
- CLI overview (~5)
- REST API overview (~5)
- Wrap-up & Final QA (~15)

CLI Overview

- Built using Apache Karaf shell mechanism
  - `ssh -p8101` …
  - `onos` shell tool provided for convenience
- All commands in `onos:` namespace
- Most arguments support TAB-completion
- Plain-text output format for human readability
- JSON output format for easy scripting via `--json` option
- Noun oriented
  - a few verb-oriented exceptions will be addressed

REST API Overview

- Core REST APIs available under path `/onos/v1`
  - all resources support read (get)
  - not all resources support write (post/put/delete) yet
- Resources available today:

`/topology`
`/devices`
`/links`
`/hosts`
`/paths`
`/flows`
`/intents`
`/cluster`
`/applications`
`/configuration`
`/config`

ONOS Dev Workshop Outline

- *ONOS Fundamentals*
  - *architecture, checking out, building & running ONOS*
- *ONOS Distributed Core*
  - *OSGi, anatomy of a core subsystem, distributed stores & primitives*
- *ONOS App Development*
  - *background, Maven archetypes, app deployment on ONOS cluster*
- *ONOS Interfaces*
  - *component config service, flow objective service, intent service*
  - *CLI, REST and GUI overview*
- Wrap-up

Wrap-Up

- Browse ONOS Wiki
  `https://wiki.onosproject.org/`
- Watch ONOS how-to screencasts on YouTube
  `https://goo.gl/8Druv0`
- Browse ONOS Java API
  `http://api.onosproject.org/`

- Join ONOS `onos-dev@onosproject.org` mailing list
  `https://wiki.onosproject.org/display/ONOS/ONOS+Mailing+Lists`

- Engage with ONOS developers & community