ONOS

Building a Distributed, Network Operating System

Madan Jampani & Brian O'Connor

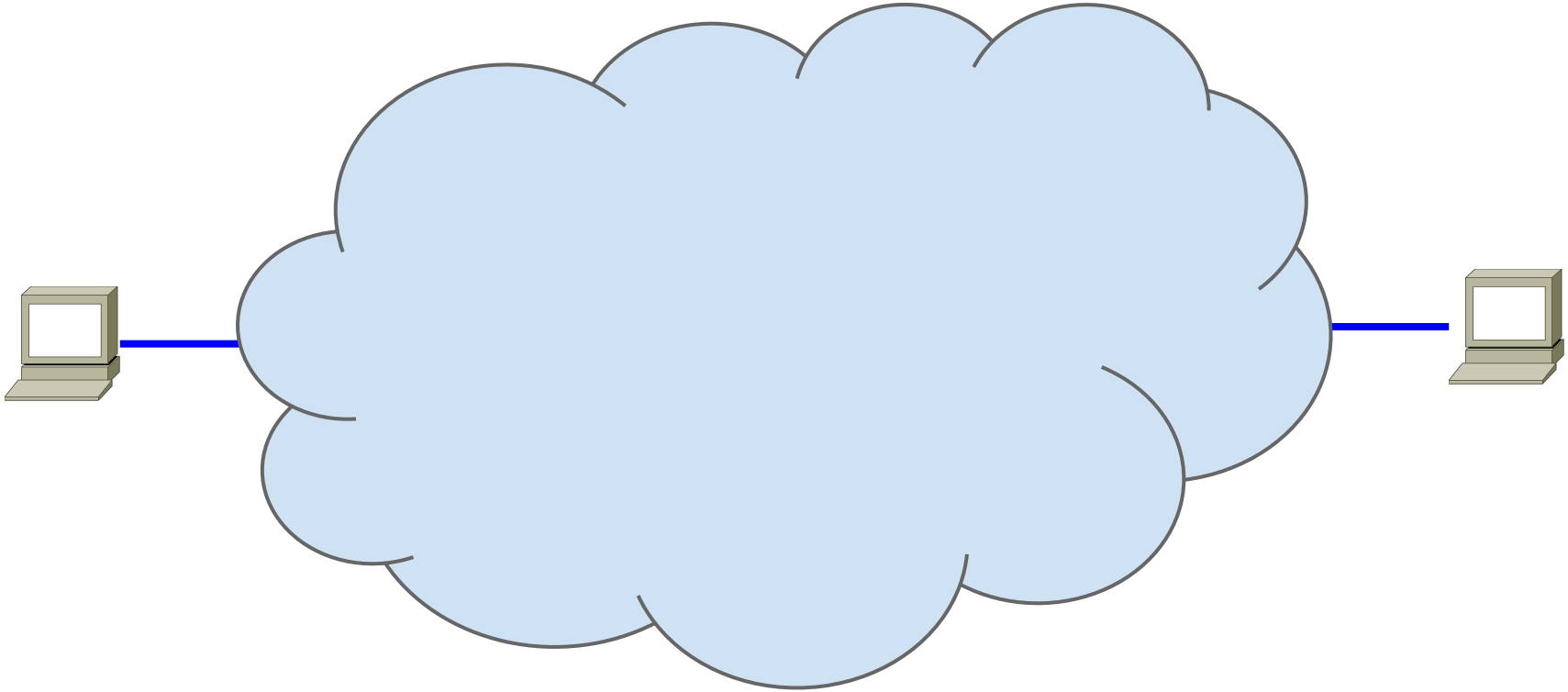
BANV Meetup

3/16/2015

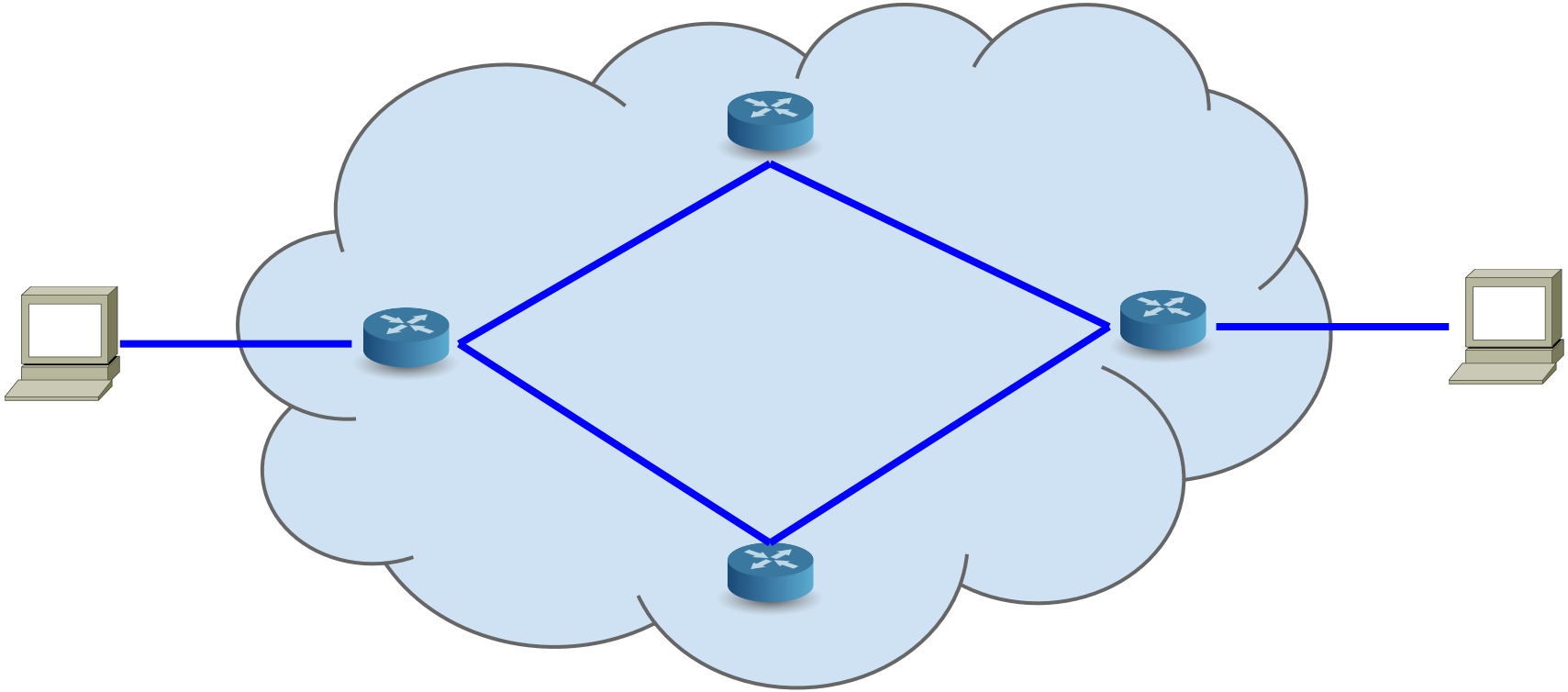
Outline

- A simple example as motivation
- ONOS Architecture
- Distributed primitives for managing state
- APIs and a template for implementation
- Intent Framework
- Live Demo
- Q/A

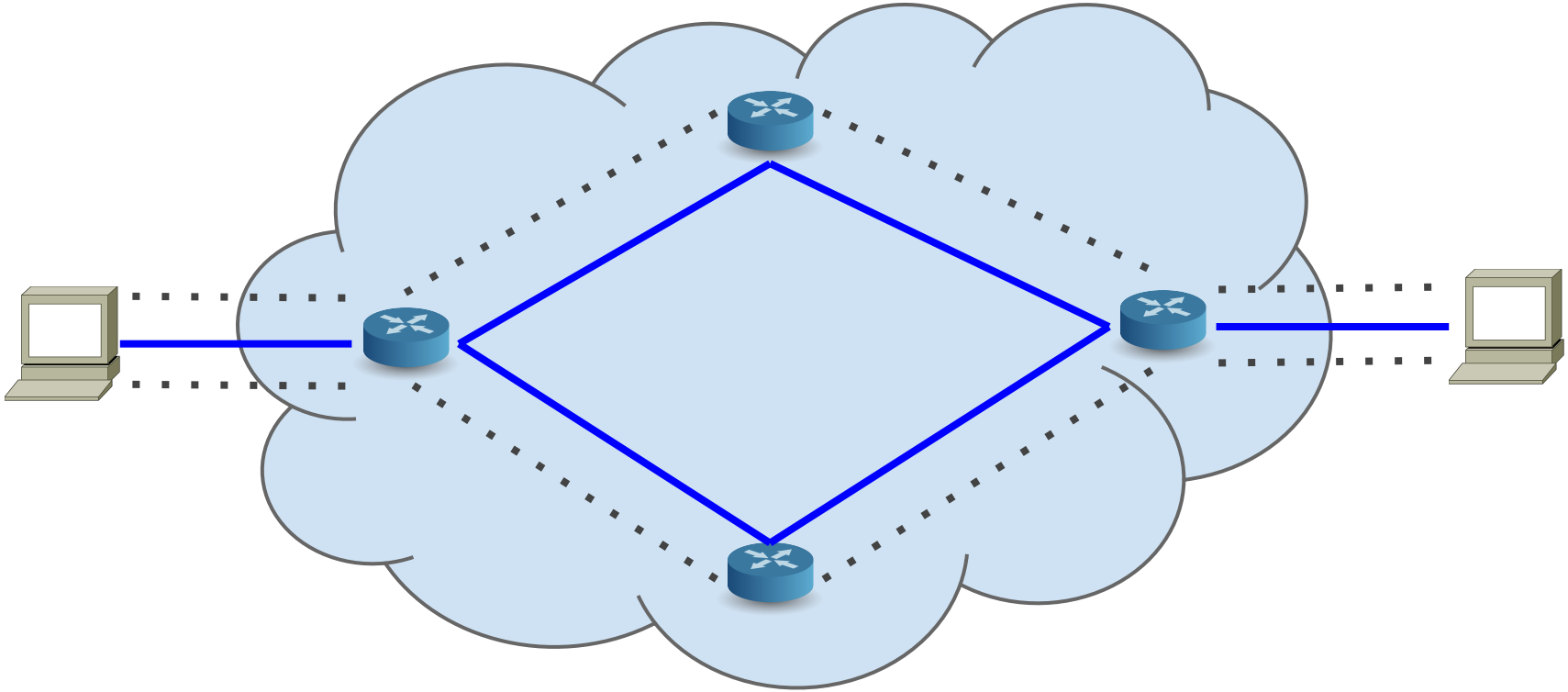
Problem: Set up a Flow



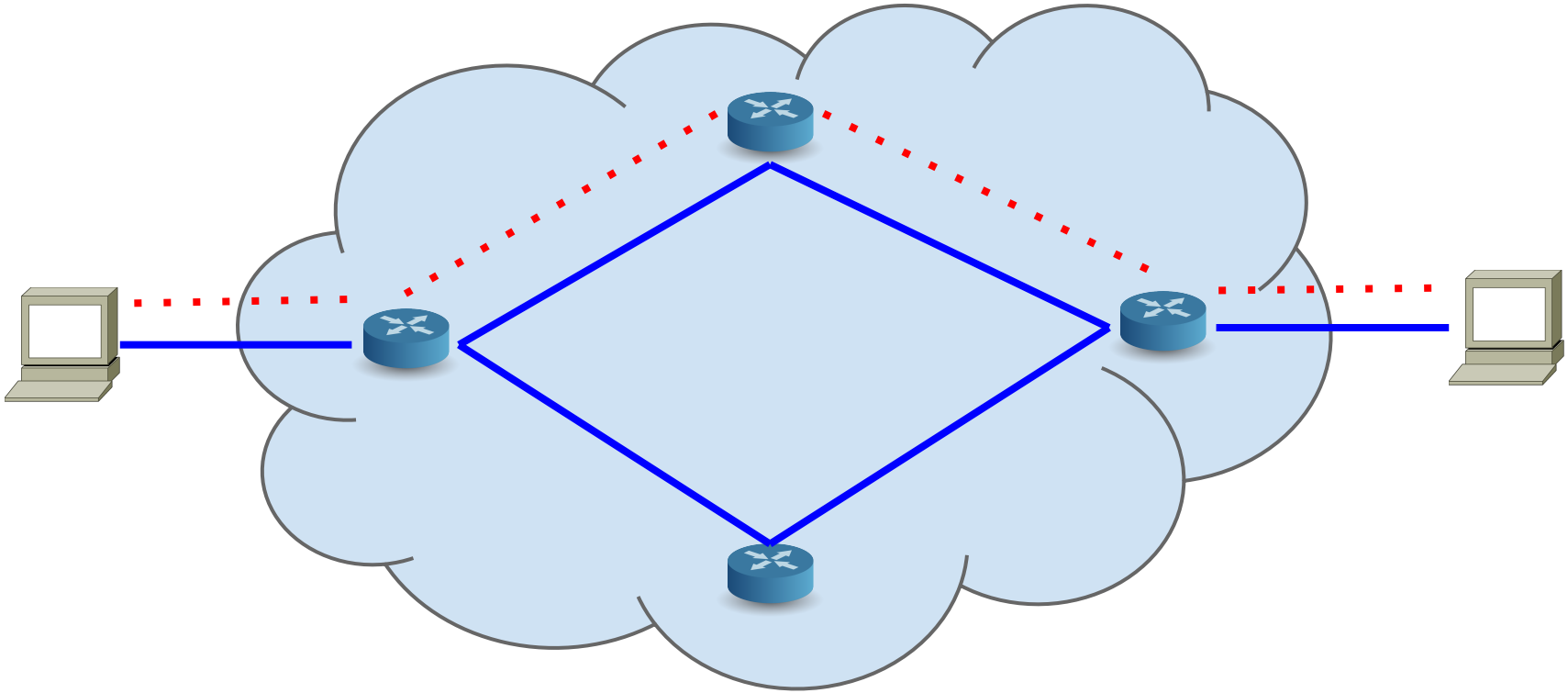
Topology Discovery



Path Computation

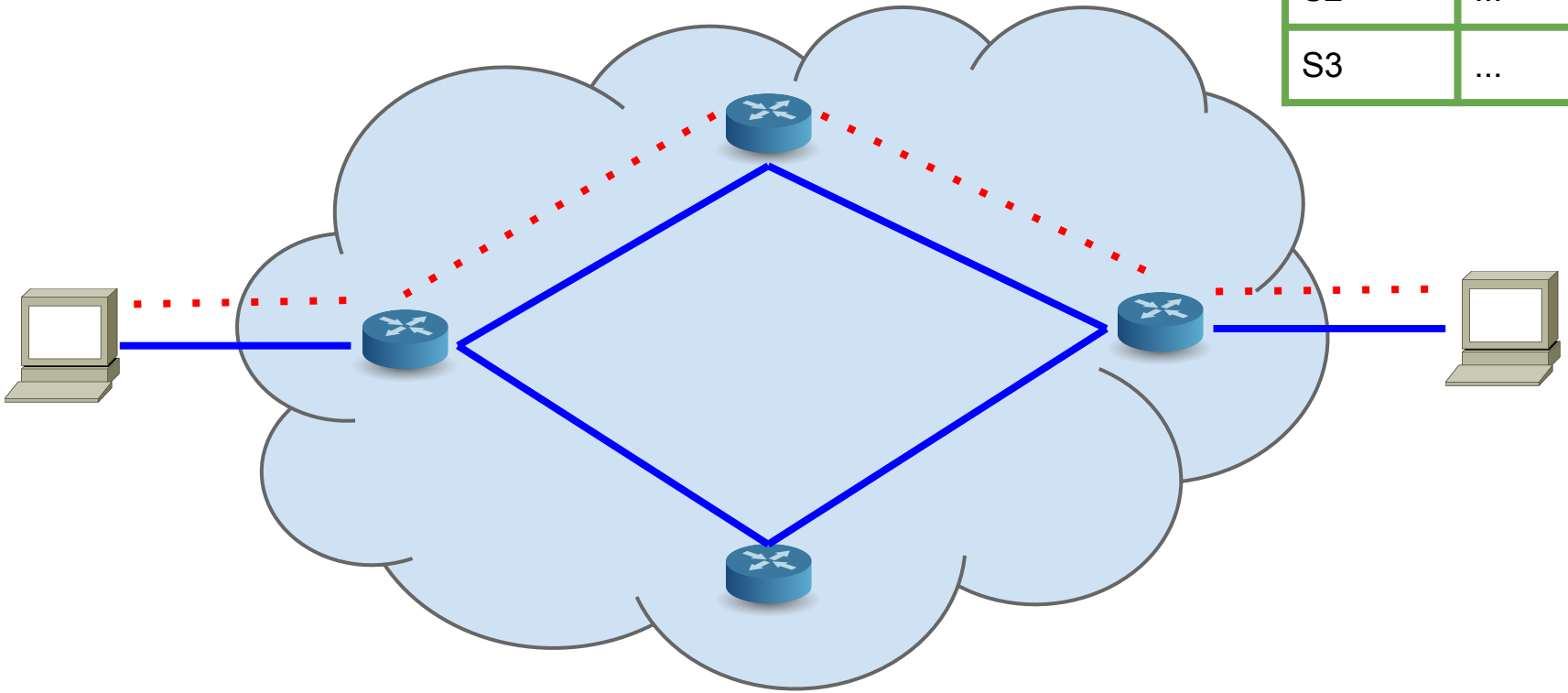


Path Selection



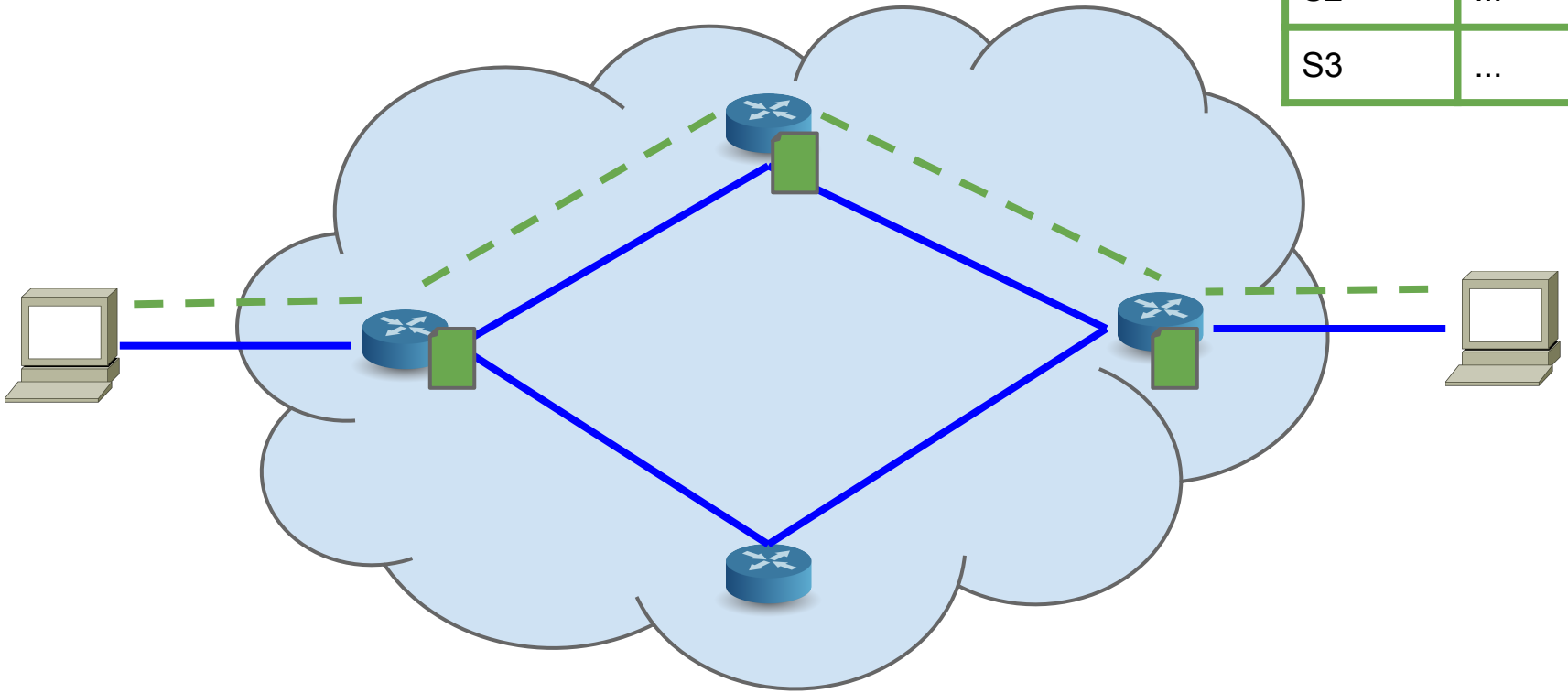
Compute Flow Mods

Switch	FlowEntry
S1	...
S2	...
S3	...



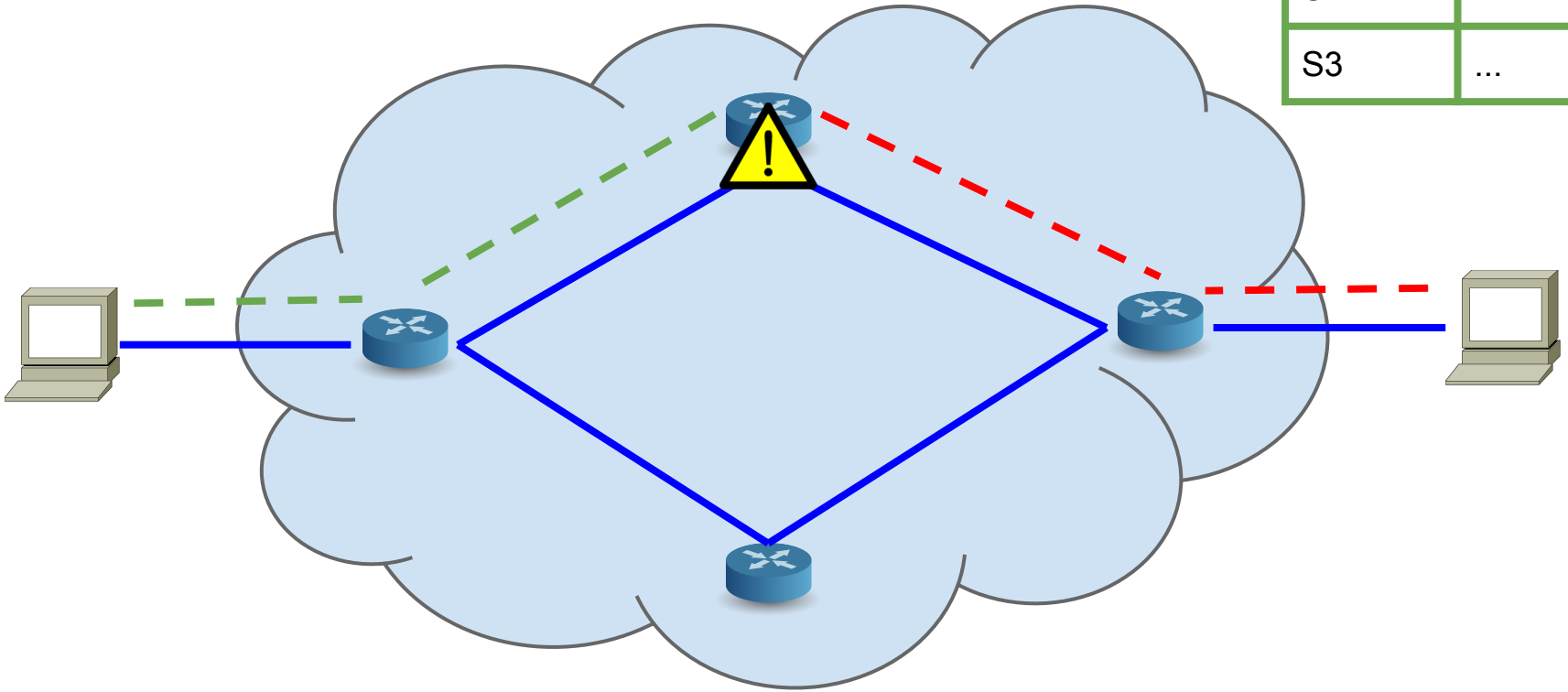
Install Flow Mods

Switch	FlowEntry
S1	...
S2	...
S3	...

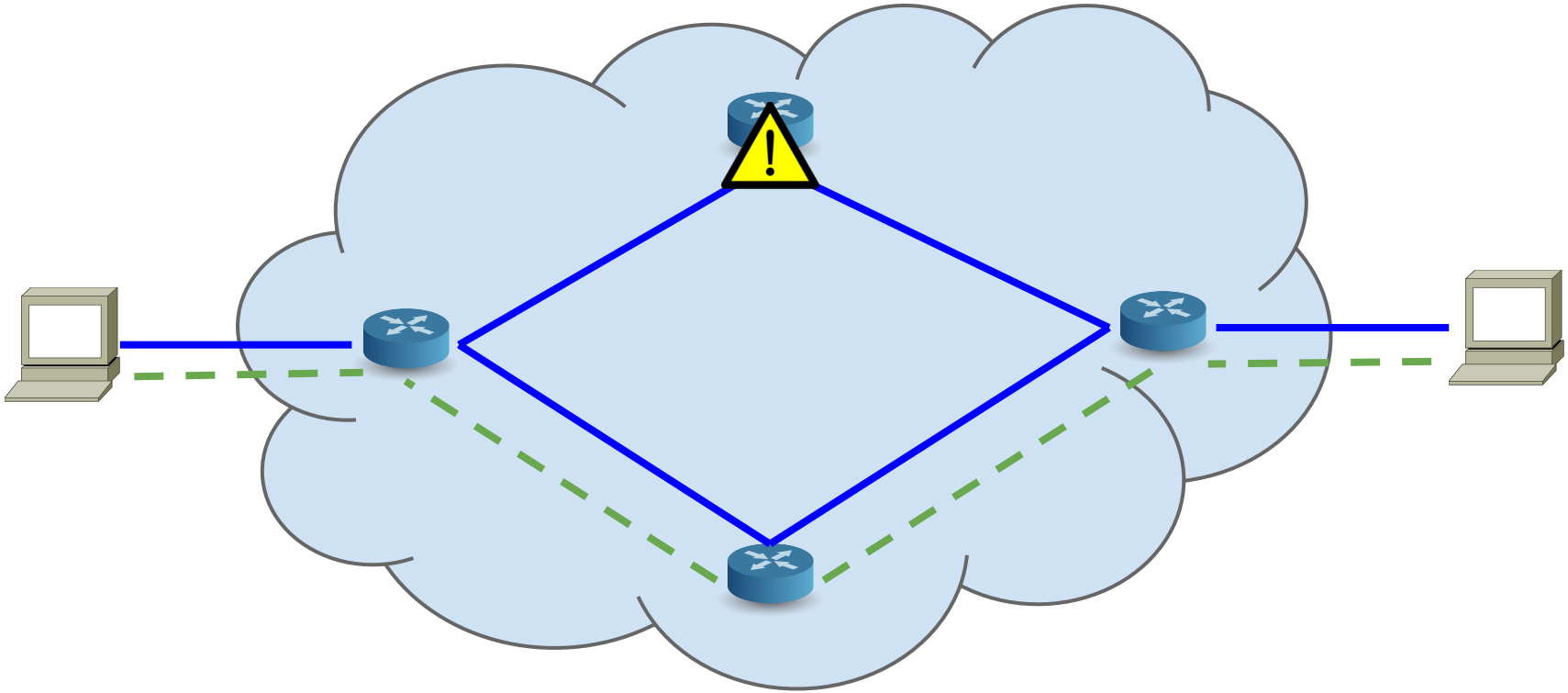


Topology Monitoring

Switch	FlowEntry
S1	...
S2	...
S3	...



Flow Repair



Wishlist

- Global Network View
- Path Computation
- Flow Mod Computation / Installation
- Monitor and React to network events

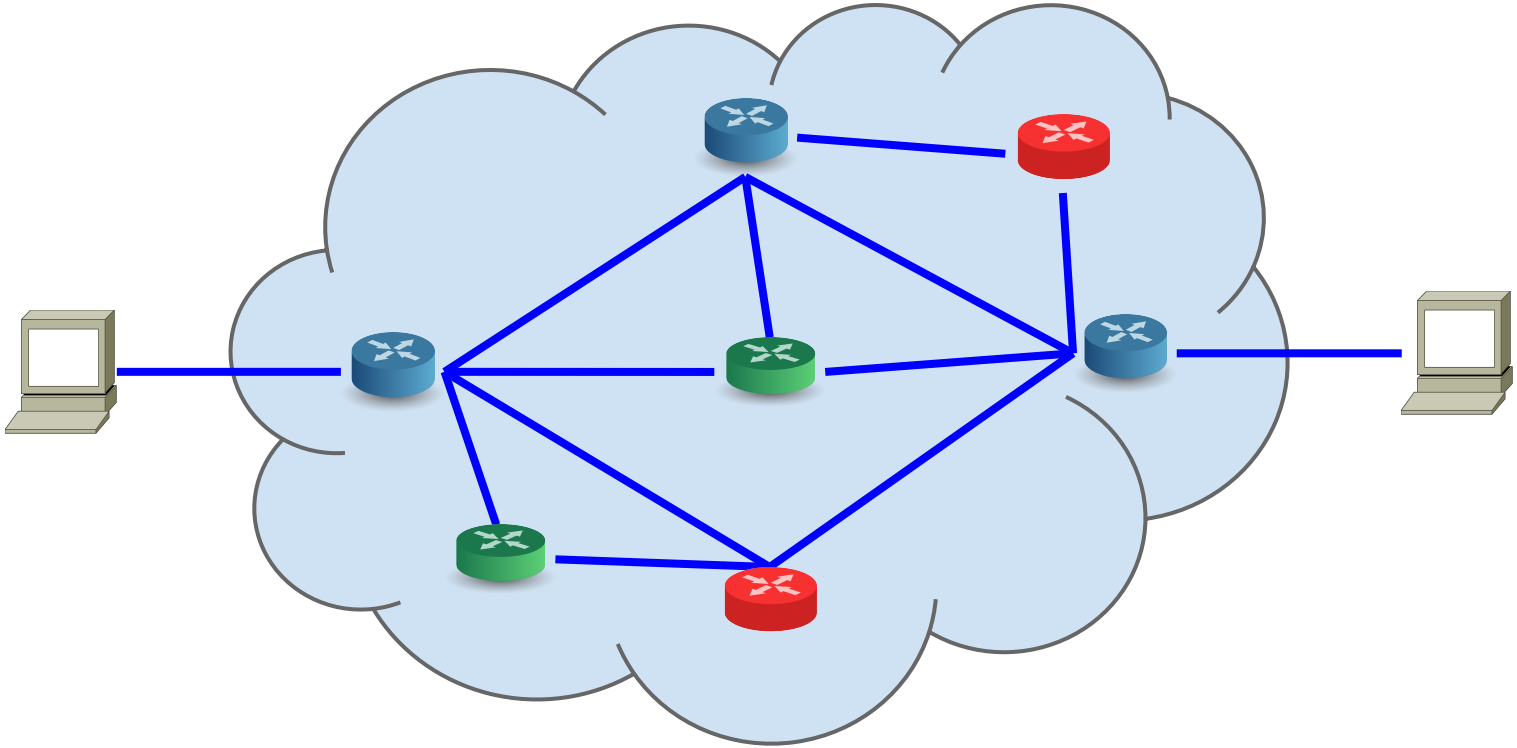
Do all this with...

- High Availability
- At Scale

Wishlist

- Global Network View
- Path Computation
- Flow Mod Computation / Installation
- Monitor and React to network events
- High Availability and Scale

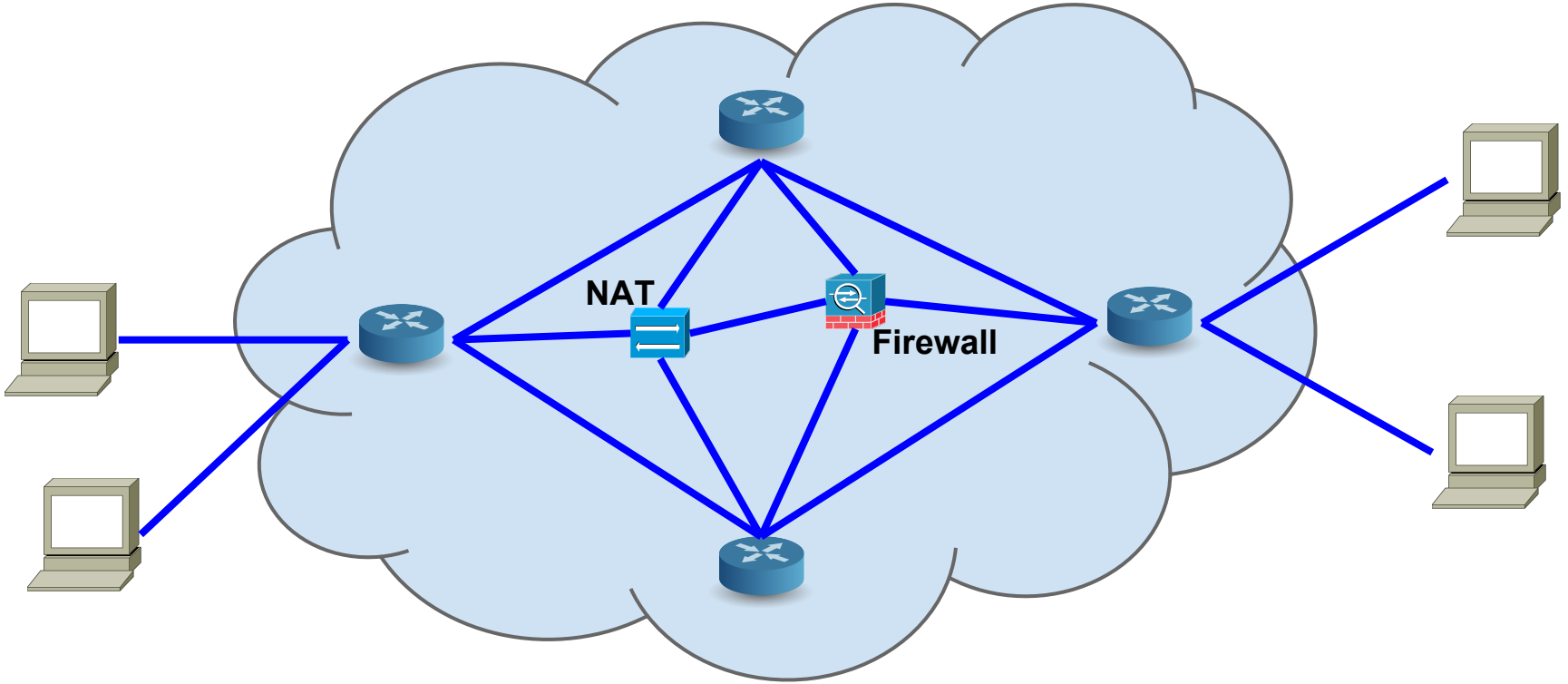
Support variety of devices and protocols



Wishlist

- Global Network View
- Path Computation
- Flow Mod Computation/Installation
- Monitor and React to events
- High Availability and Scale
- Extensible and Abstracted Southbound

Managing Complexity

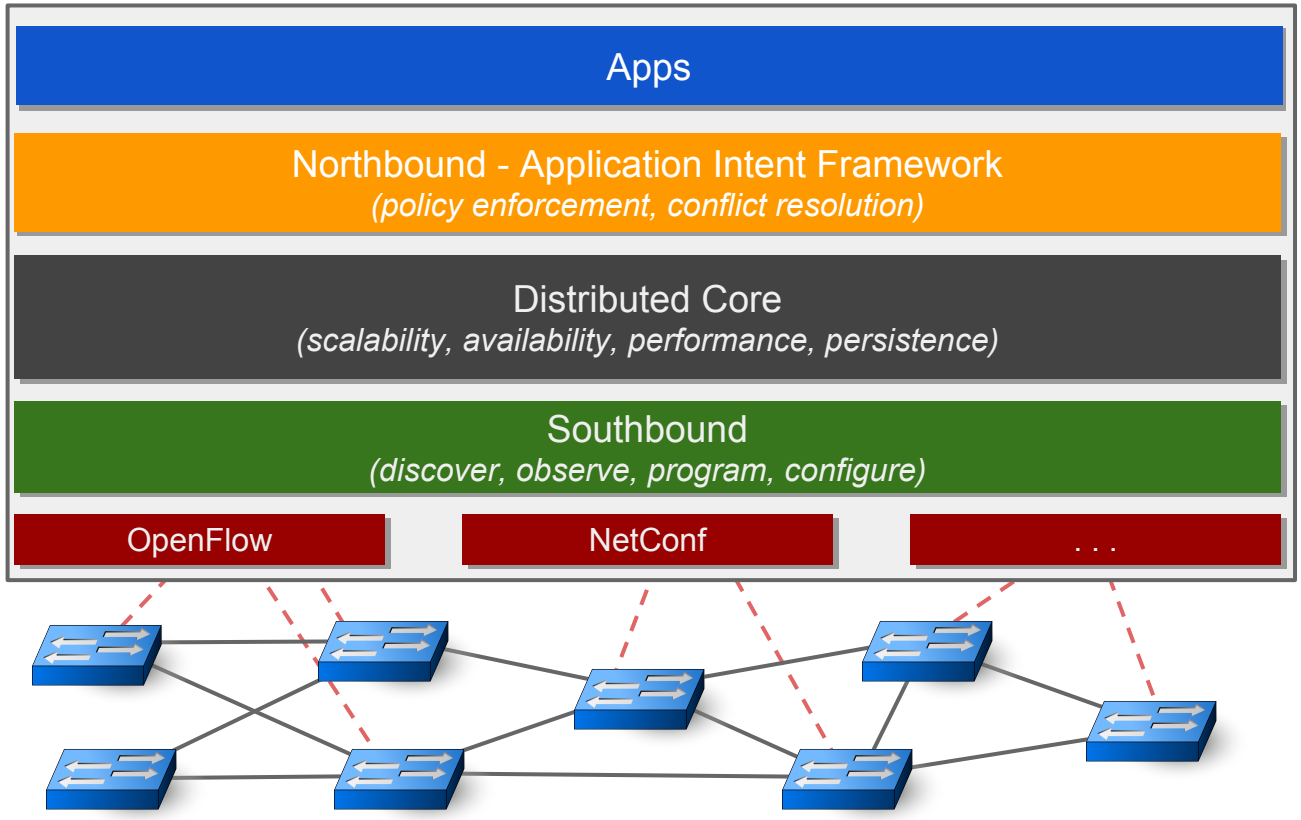


Wishlist

- Global Network View
- Path Computation
- Flow Mod Computation/Installation
- Monitor and React to events
- High Availability and Scale
- Pluggable Southbound
- Modularity and Customizability

Note: We've also done this exercise on more complex, real-world use cases.
You can read more about them in the wiki: https://wiki.onosproject.org/display/ONOS/ONOS+Use+Cases

ONOS Architecture Tiers



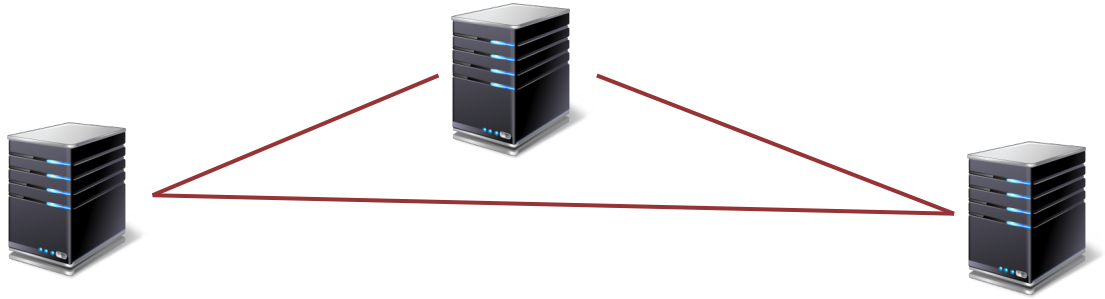
Distributed Core

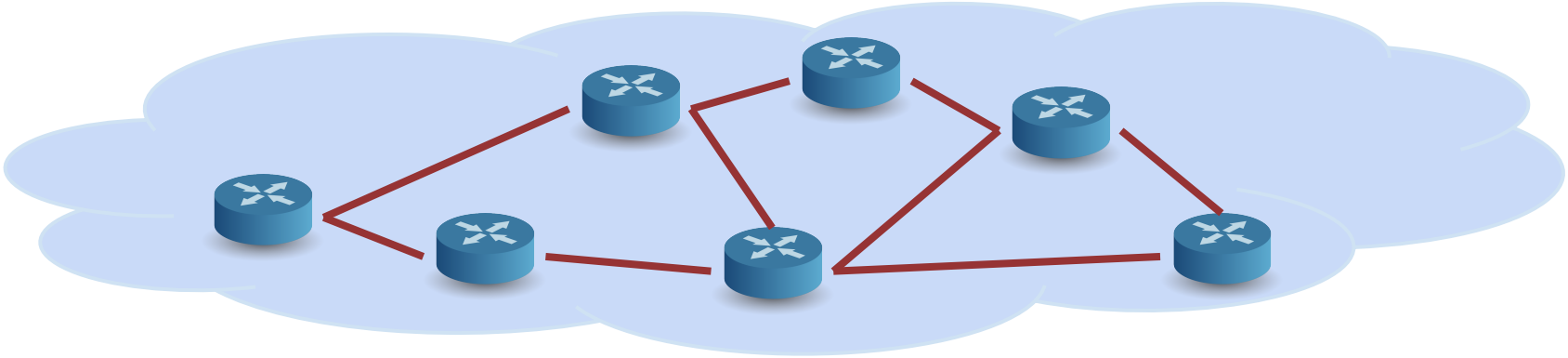
- Distributed state management framework
  - built for high availability and scale-out

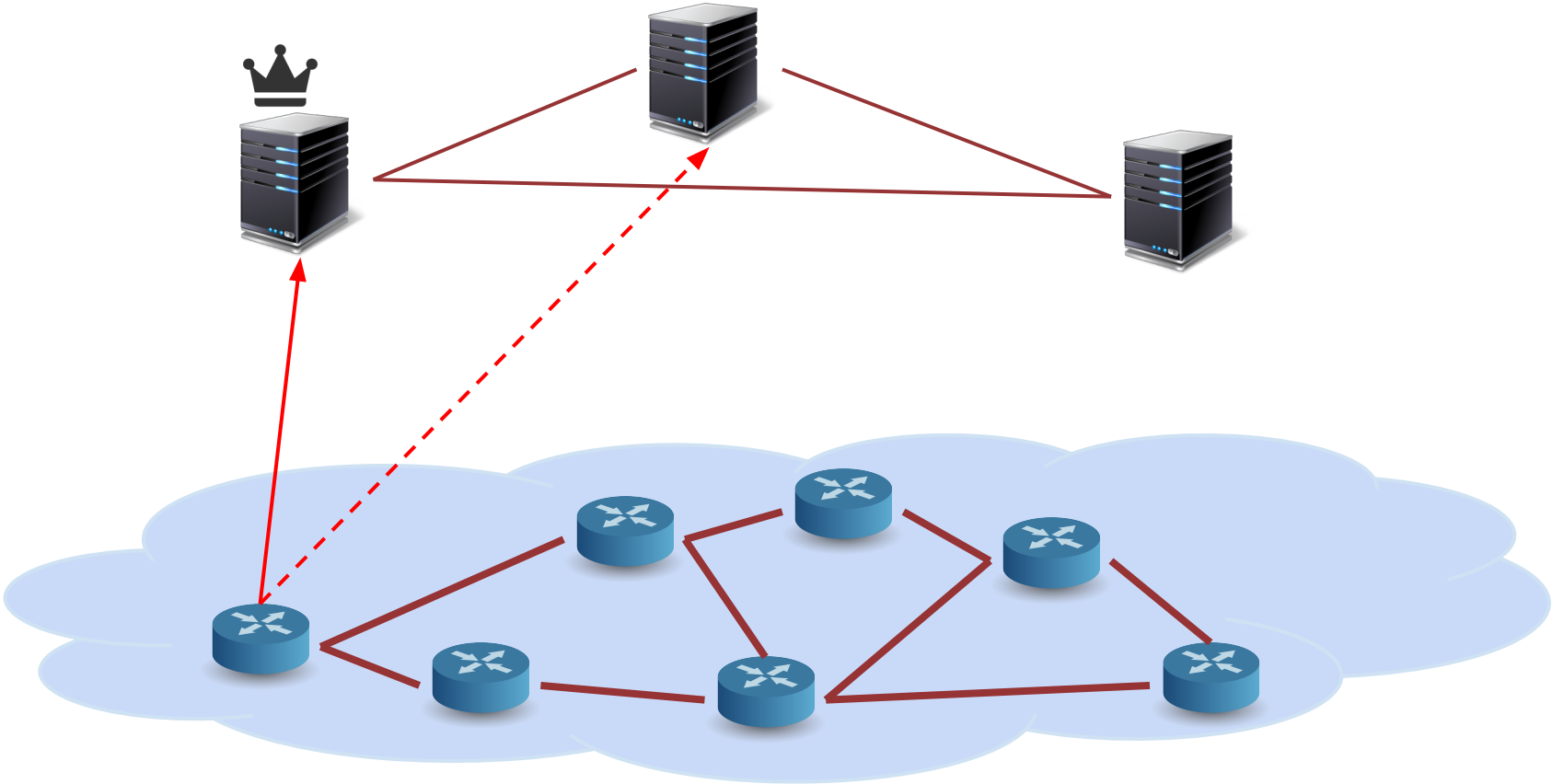
- Different types of state require different handling
  - fully replicated and eventually consistent
  - partitioned and strongly consistent

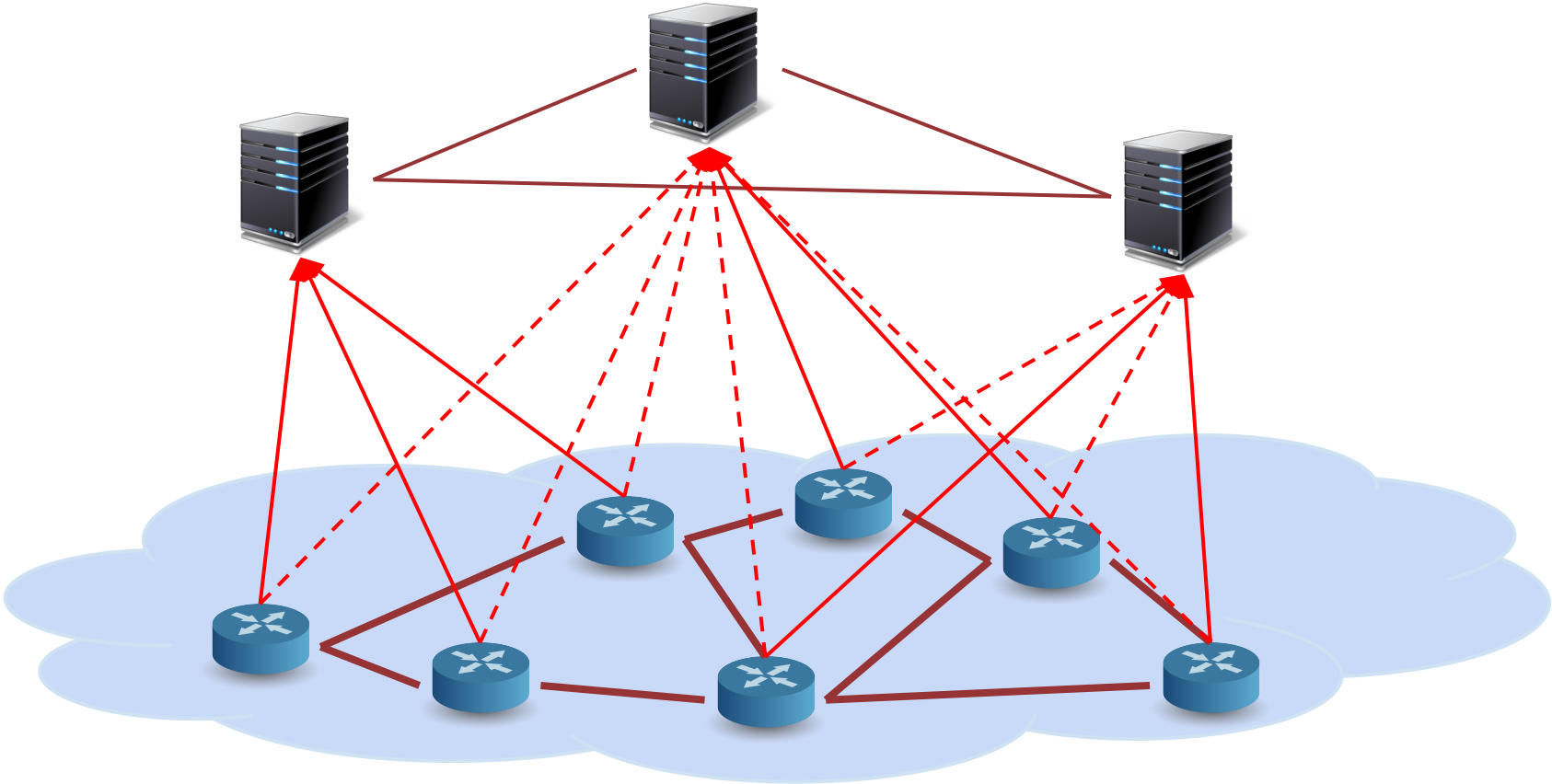
Distributed Core

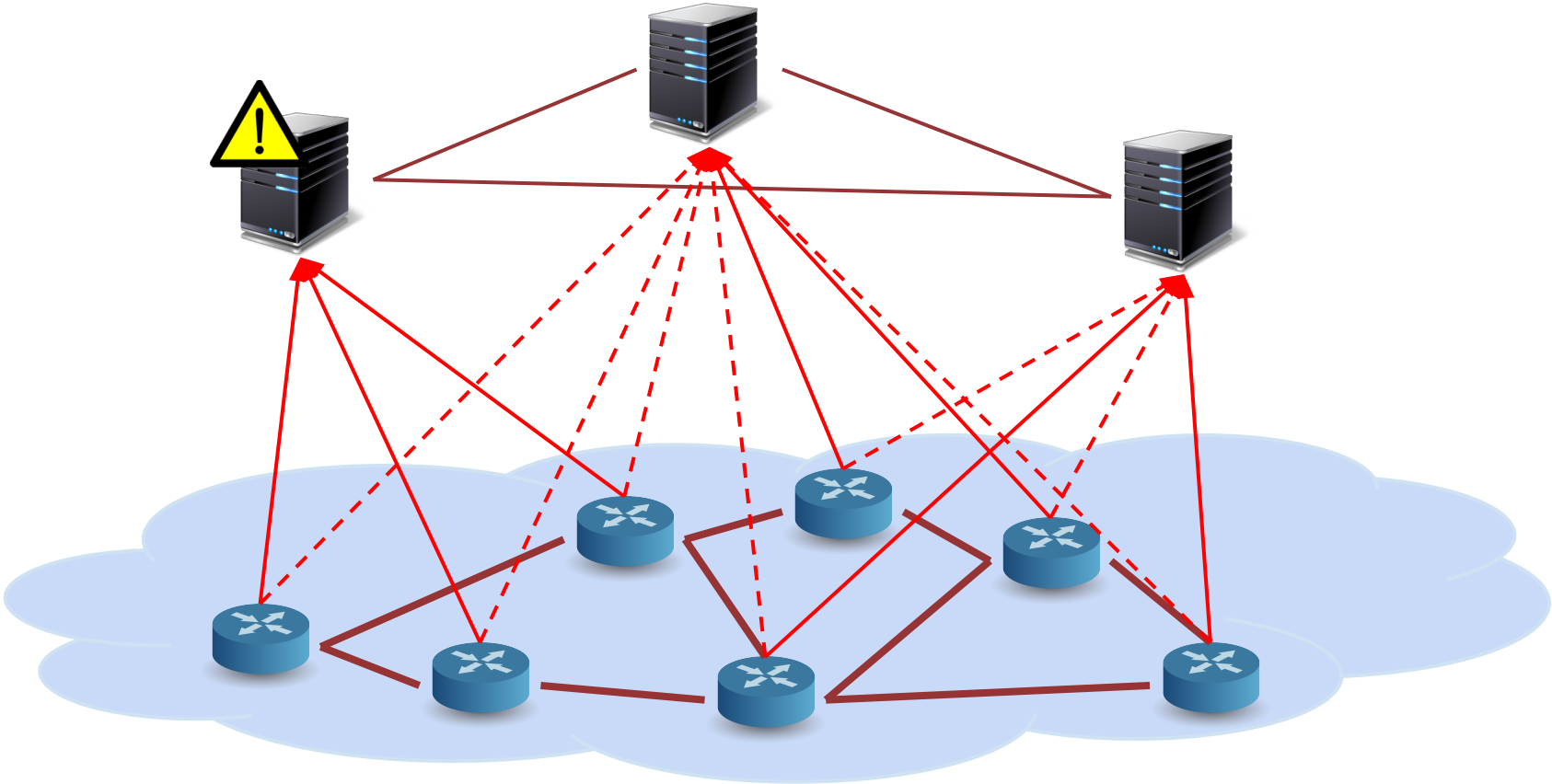
- Global Network View
- Scaling strong consistency

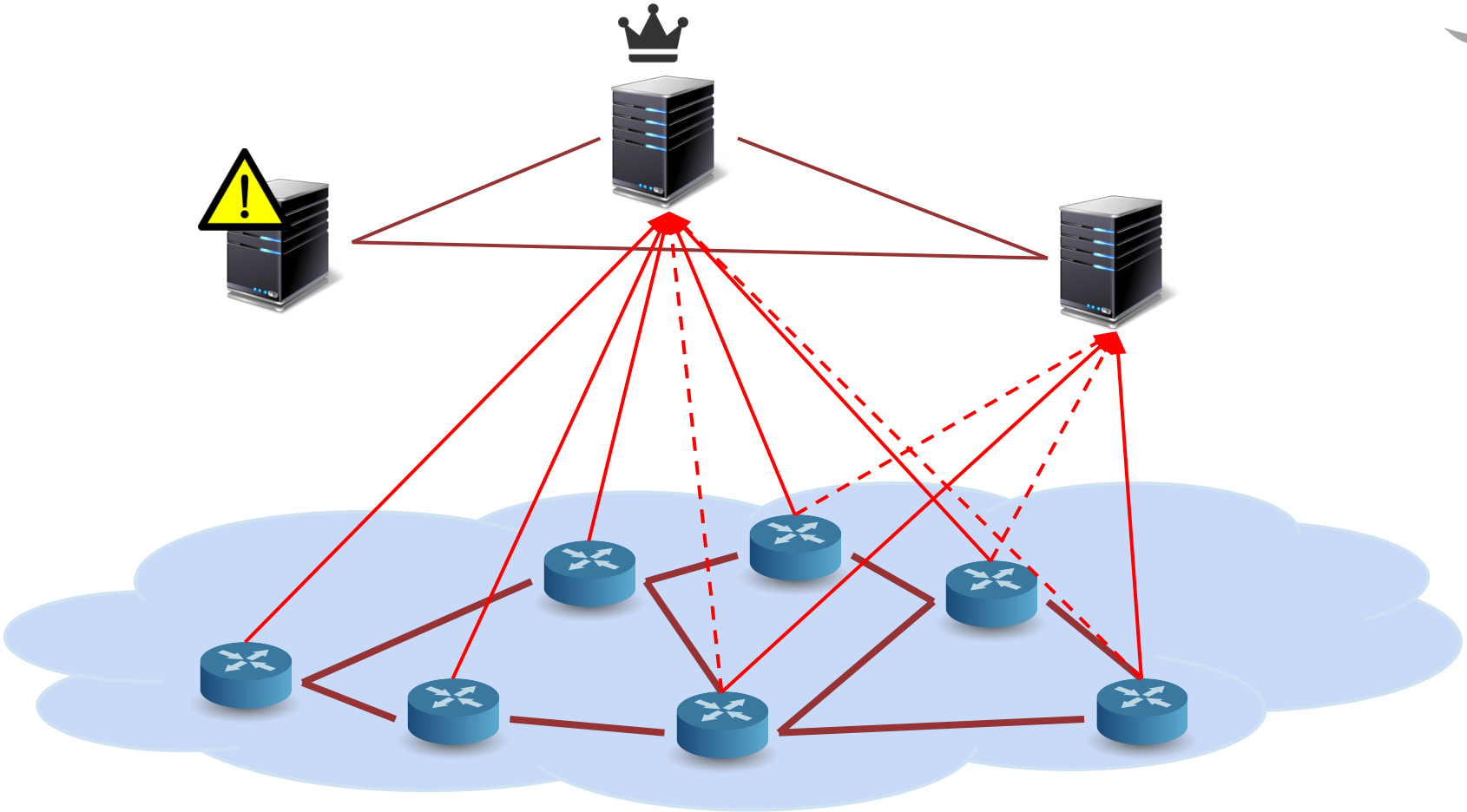


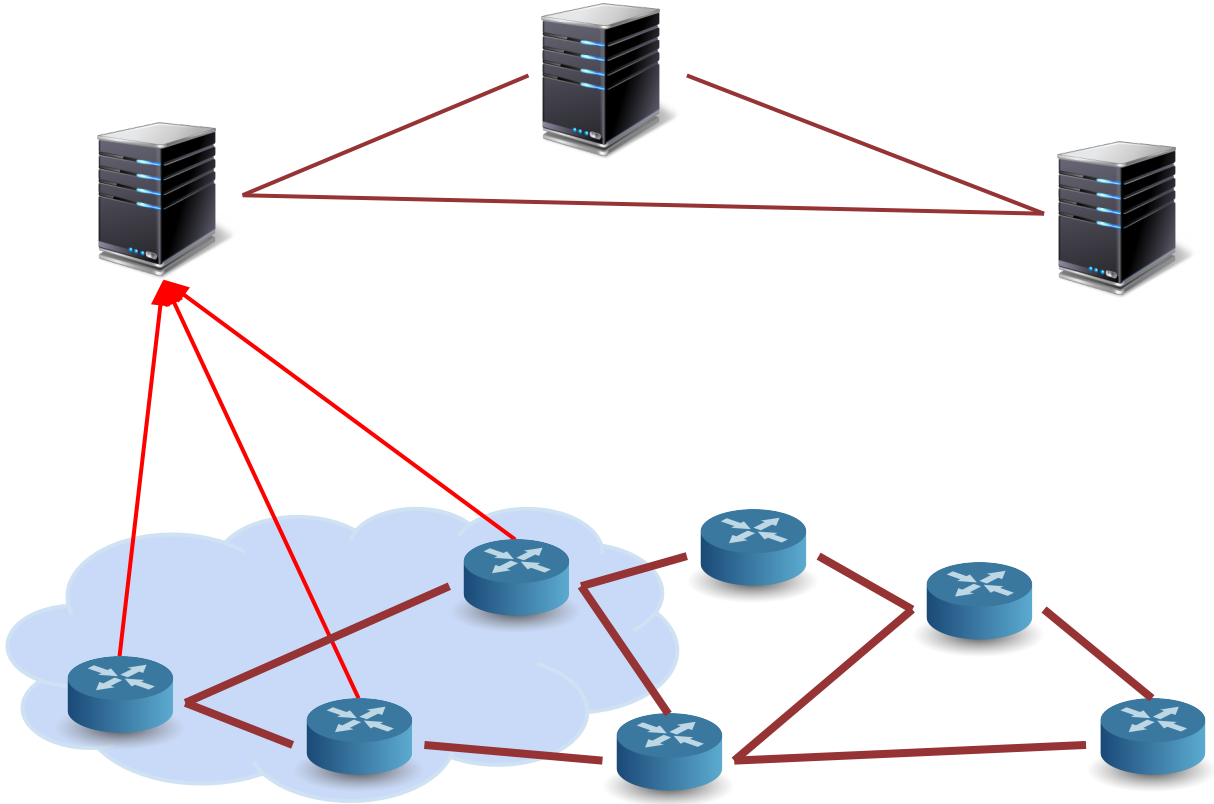


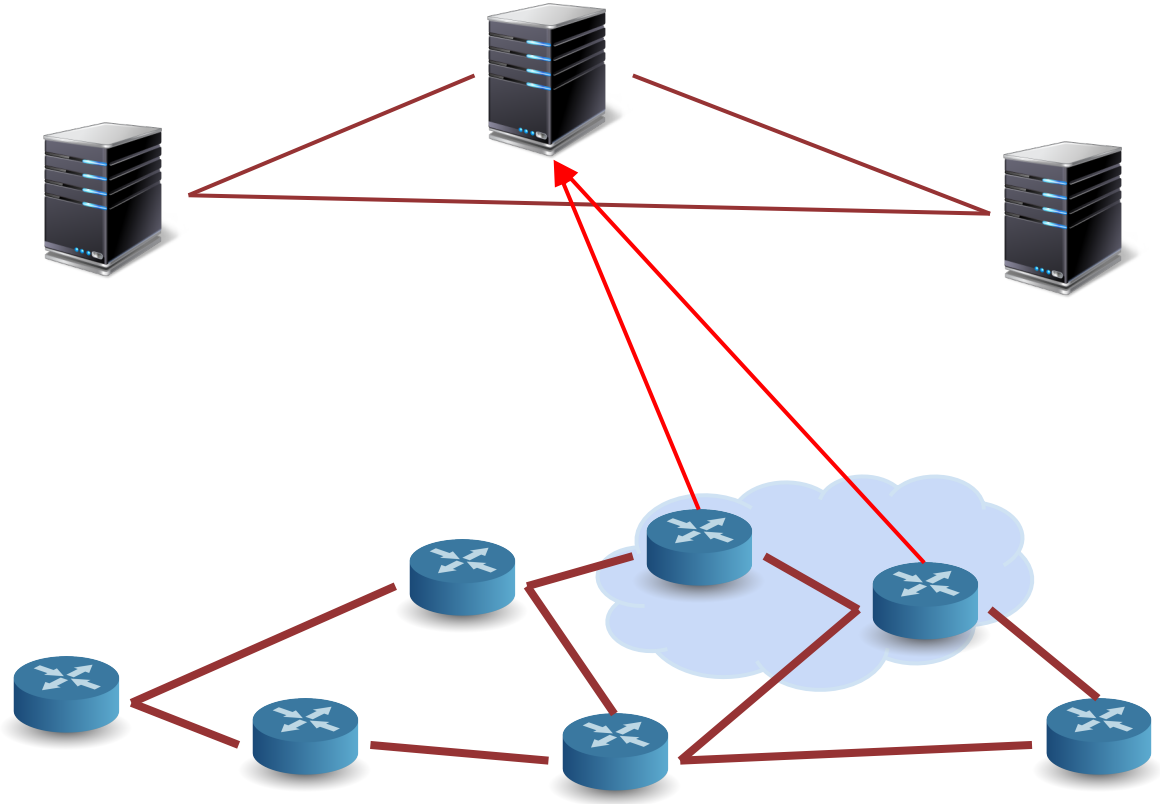


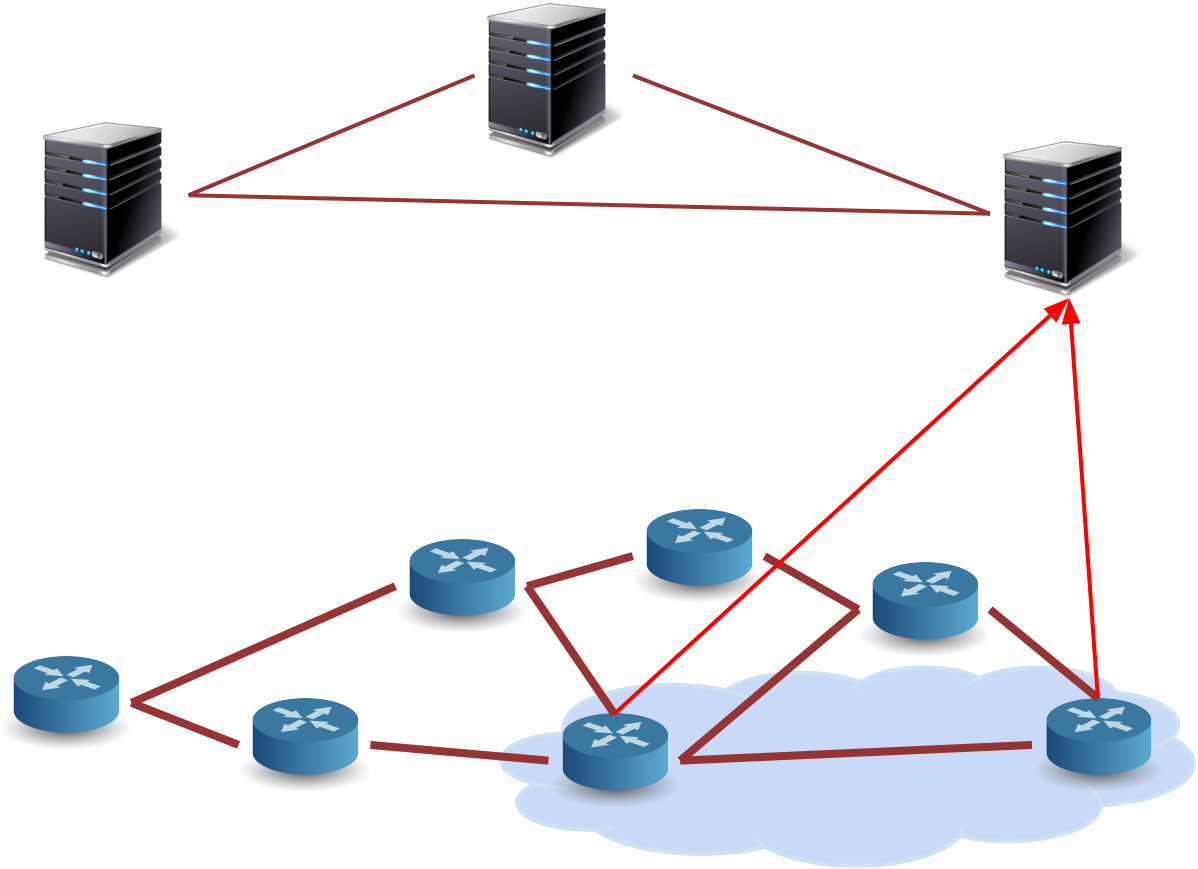


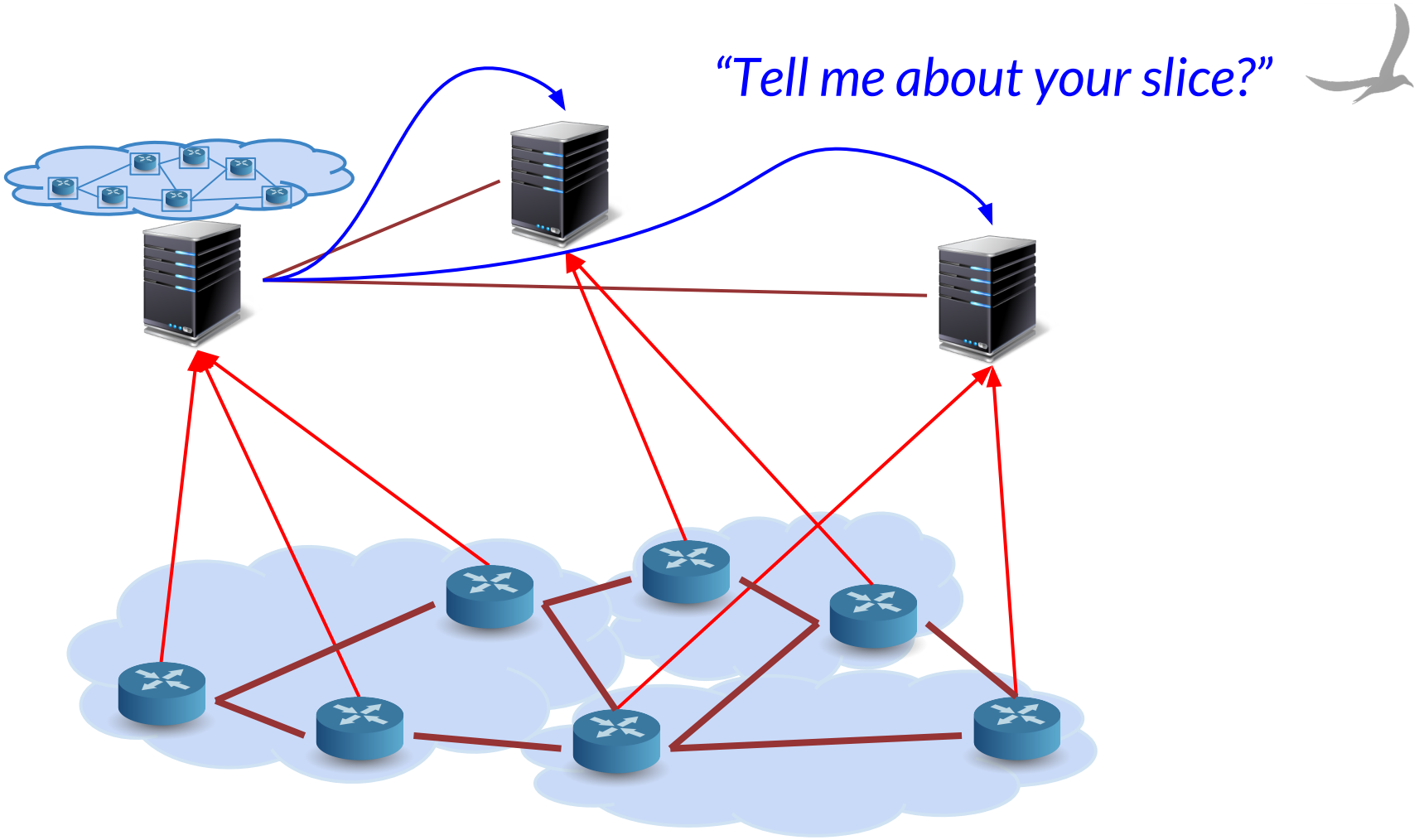


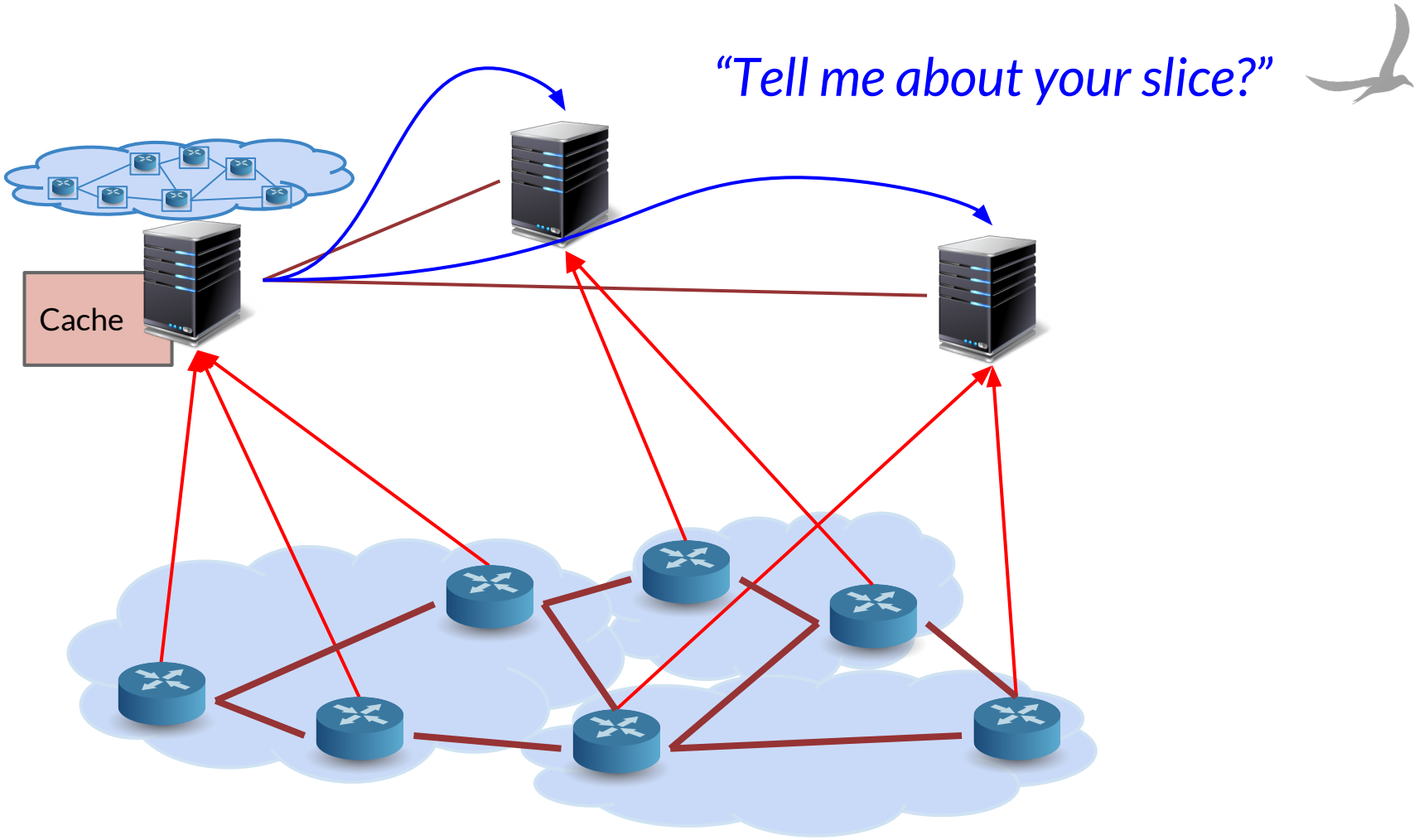


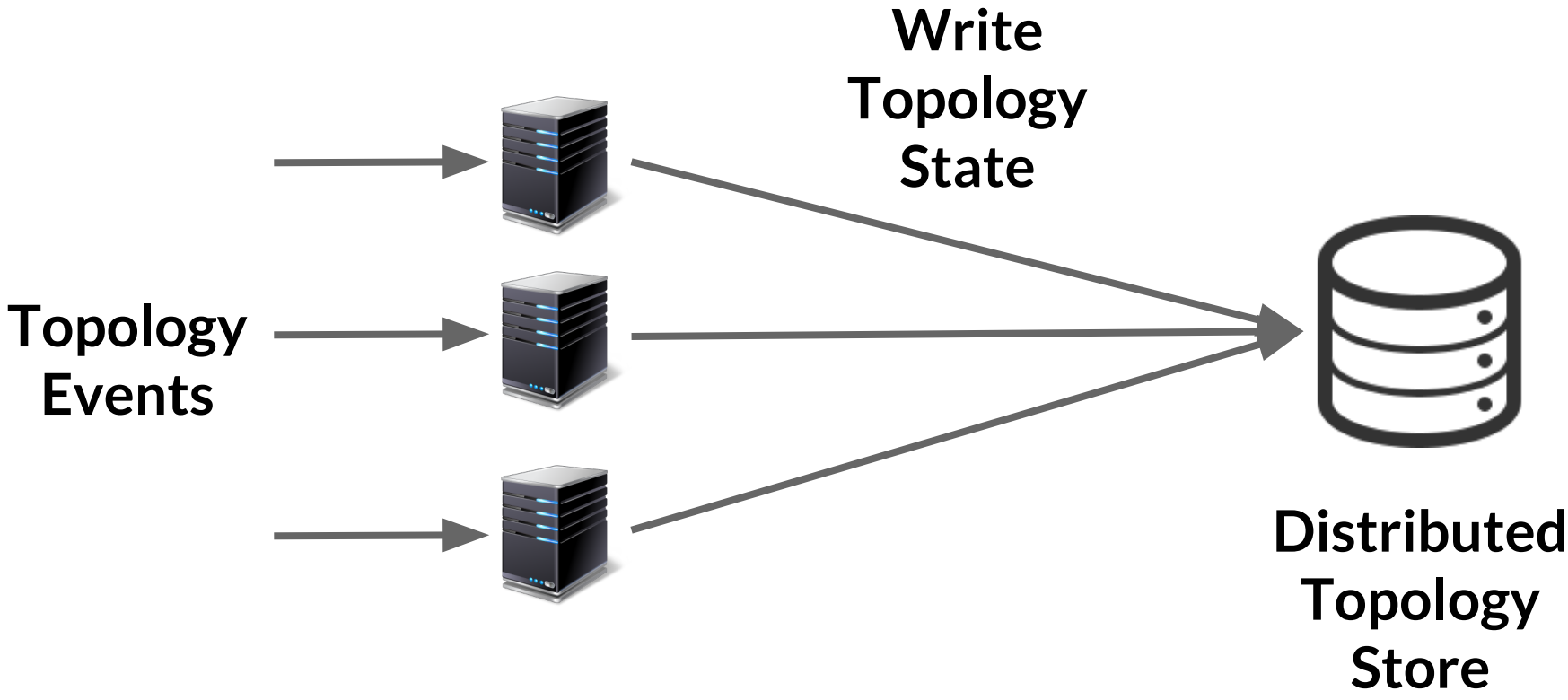


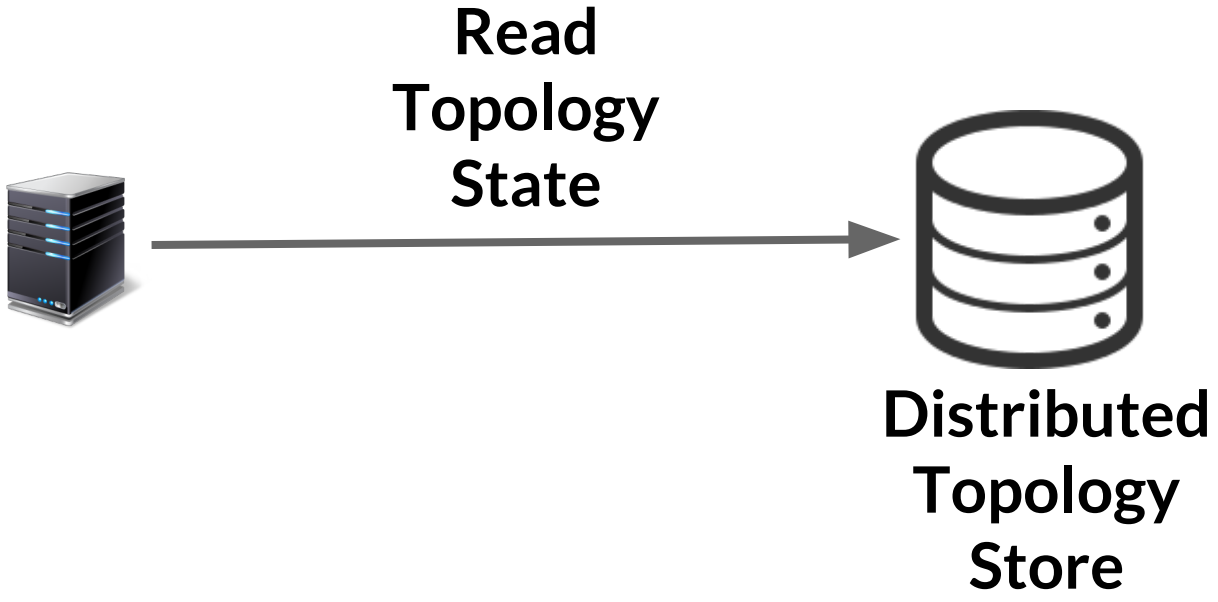


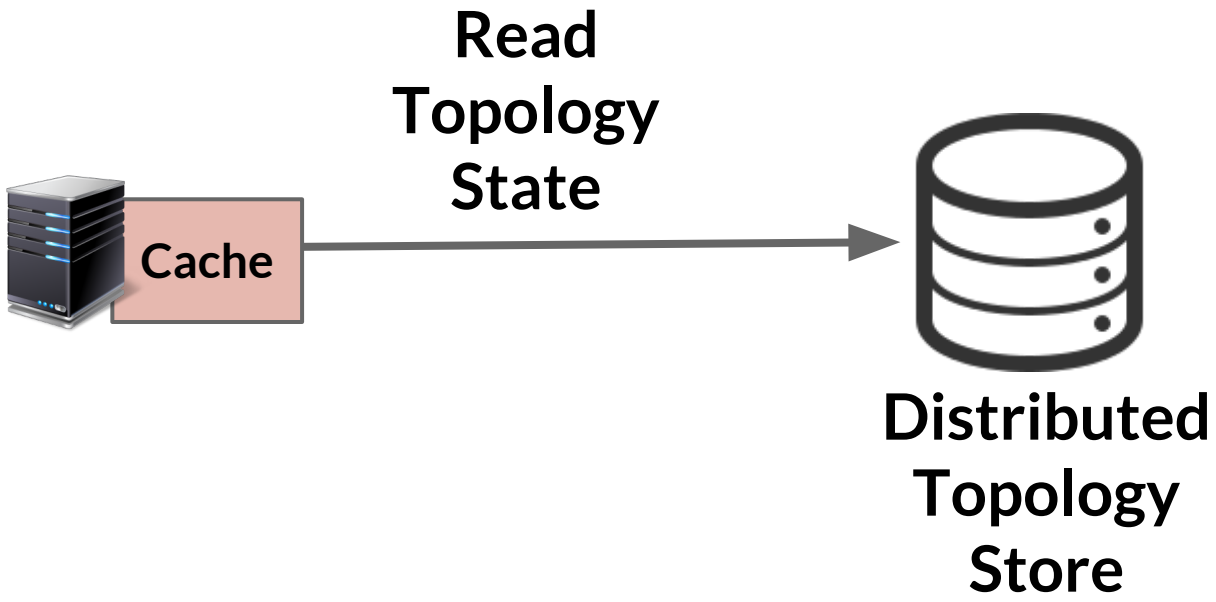


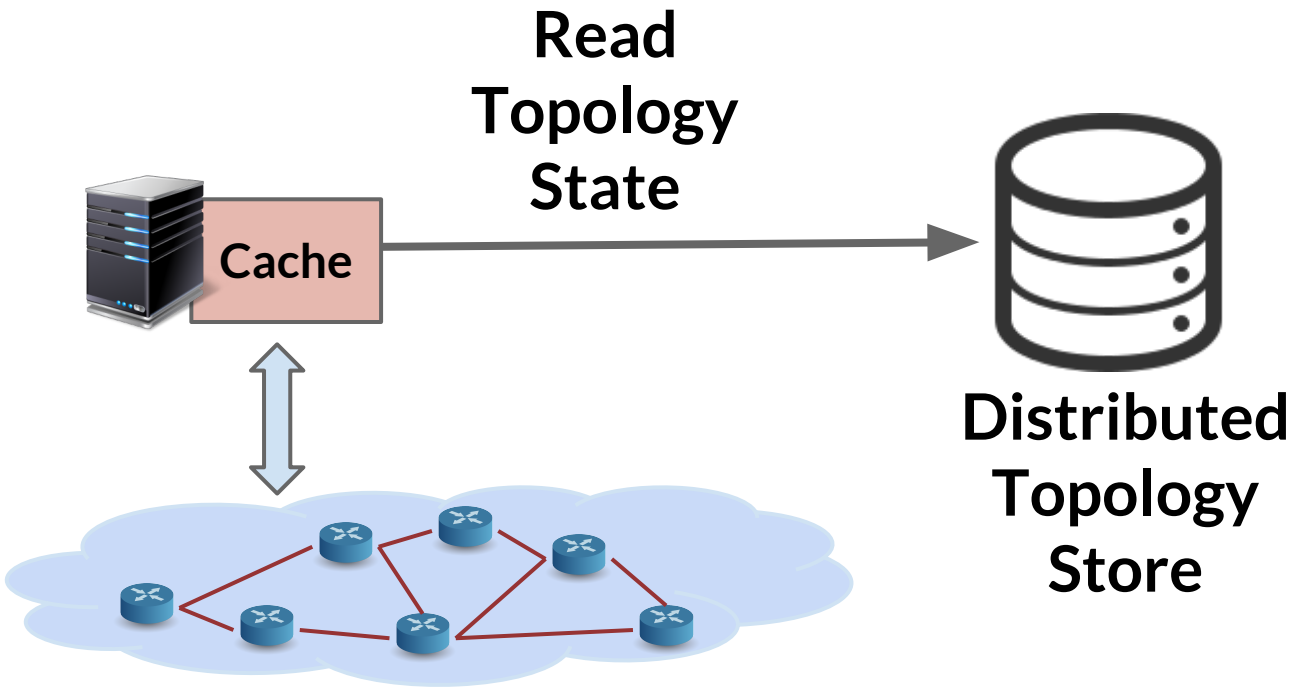

“Tell me about your slice?”


"Tell me about your slice?"
Cache

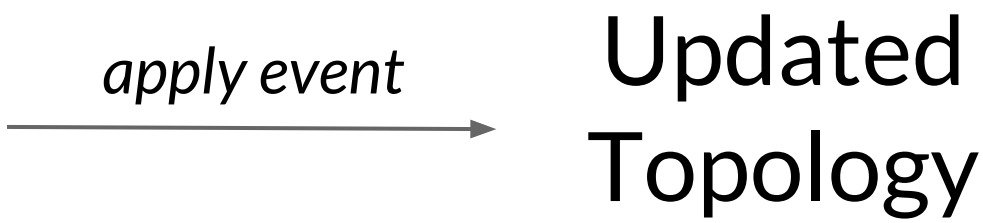

Write
Topology
State
Topology
Events
Distributed
Topology
Store


Read
Topology
State
Distributed
Topology
Store

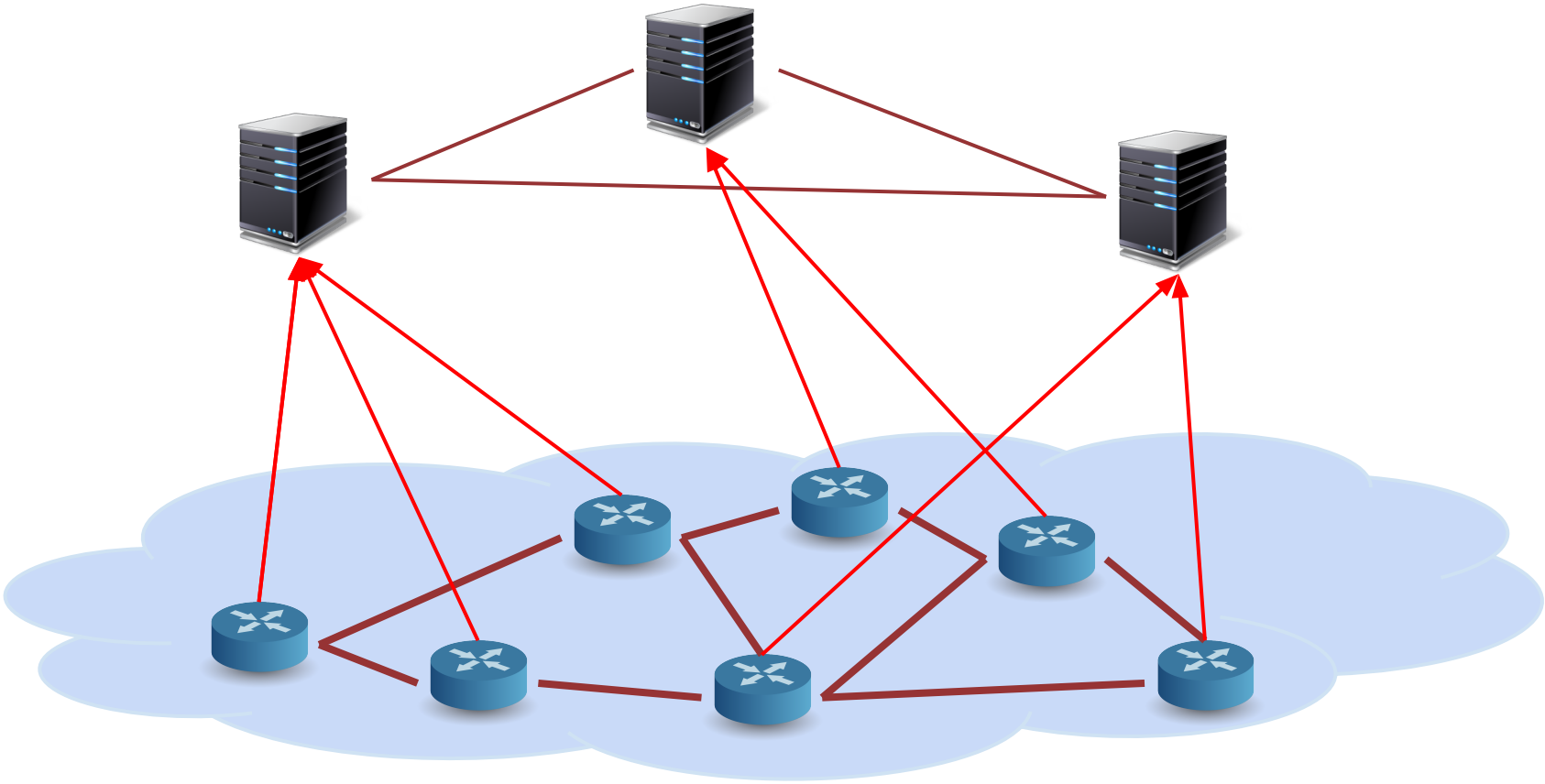

Read
Topology
State
Cache
Distributed
Topology
Store

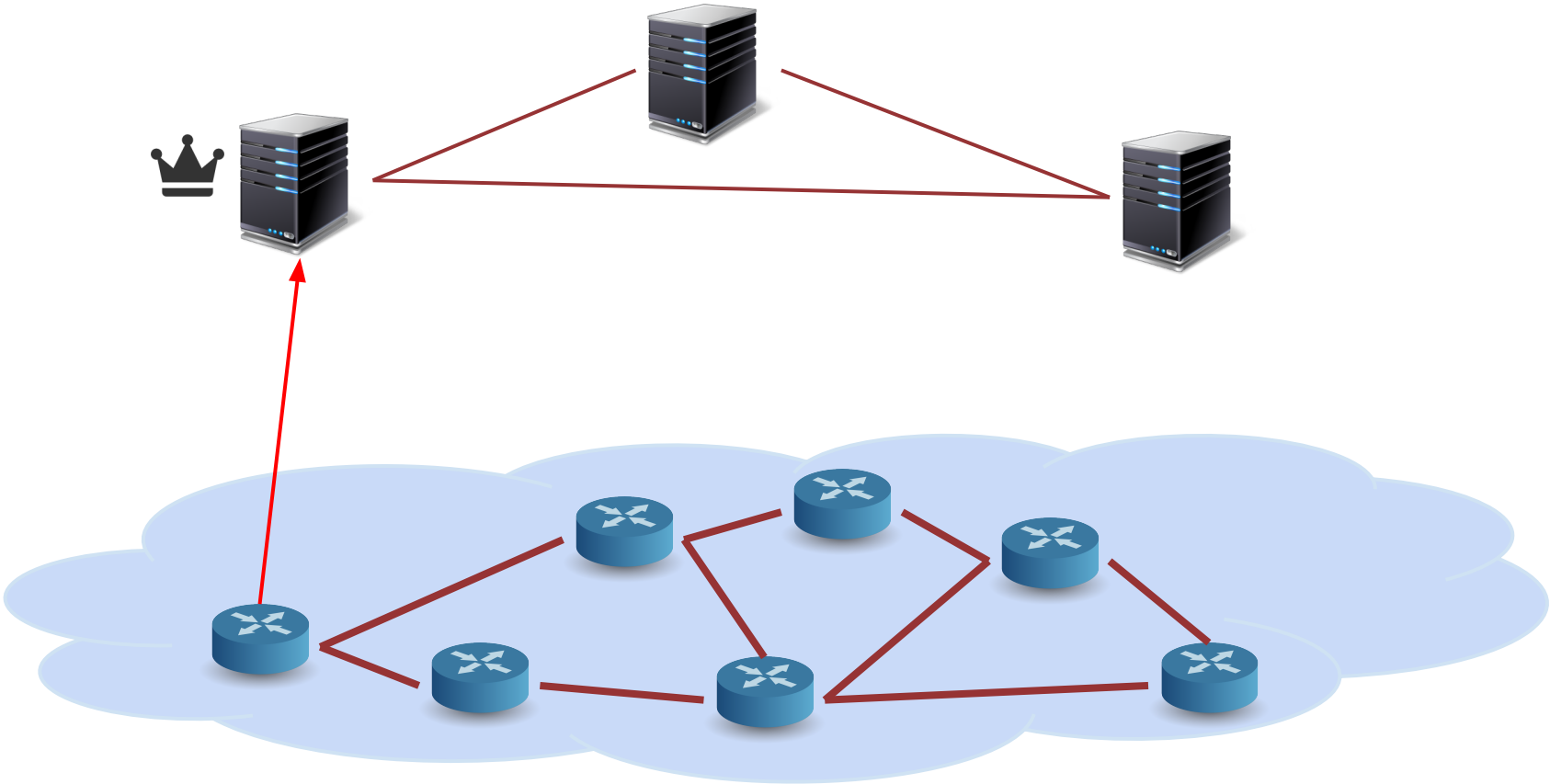

Read
Topology
State
Cache
Distributed
Topology
Store

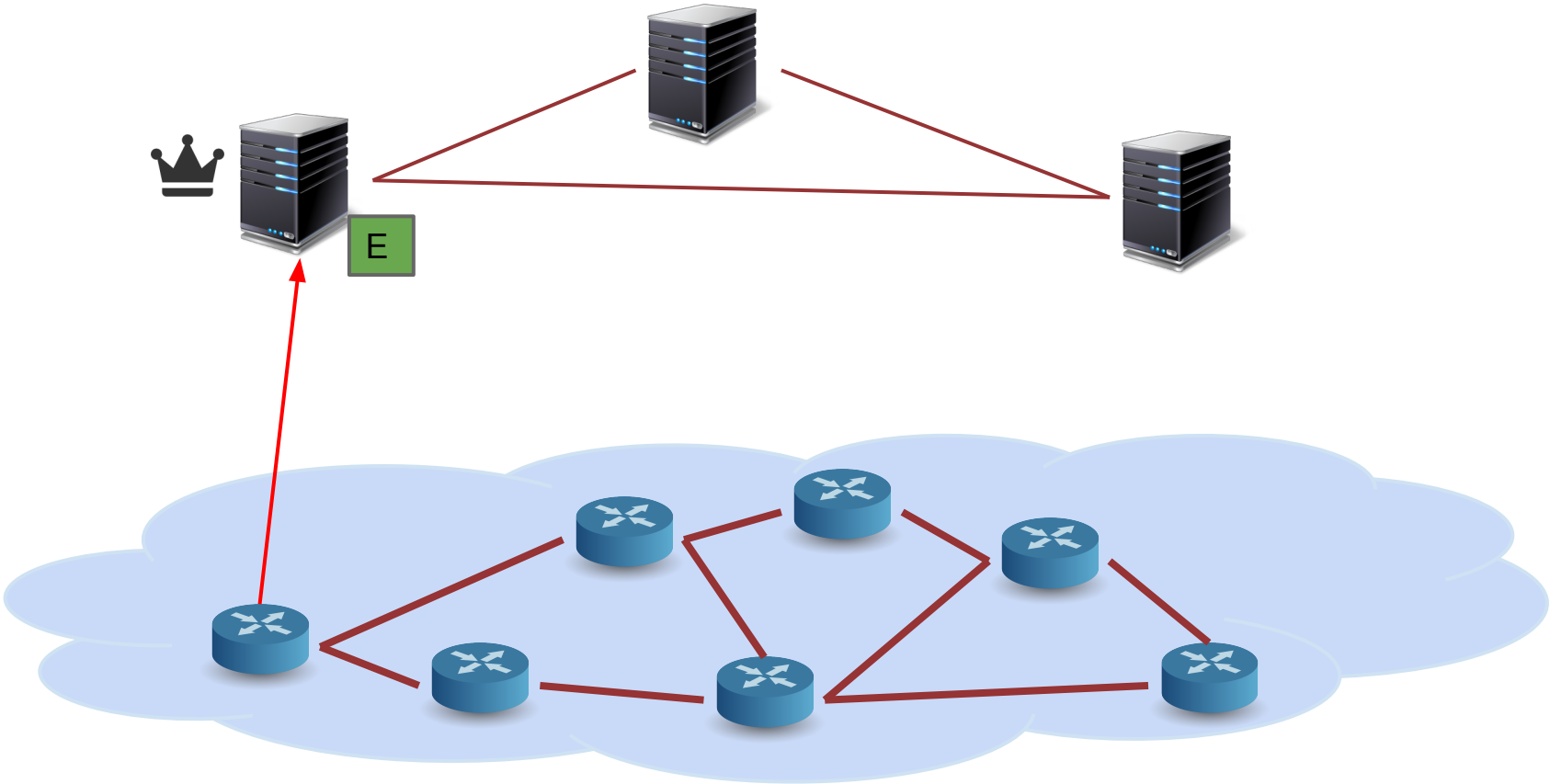
Topology as a State Machine

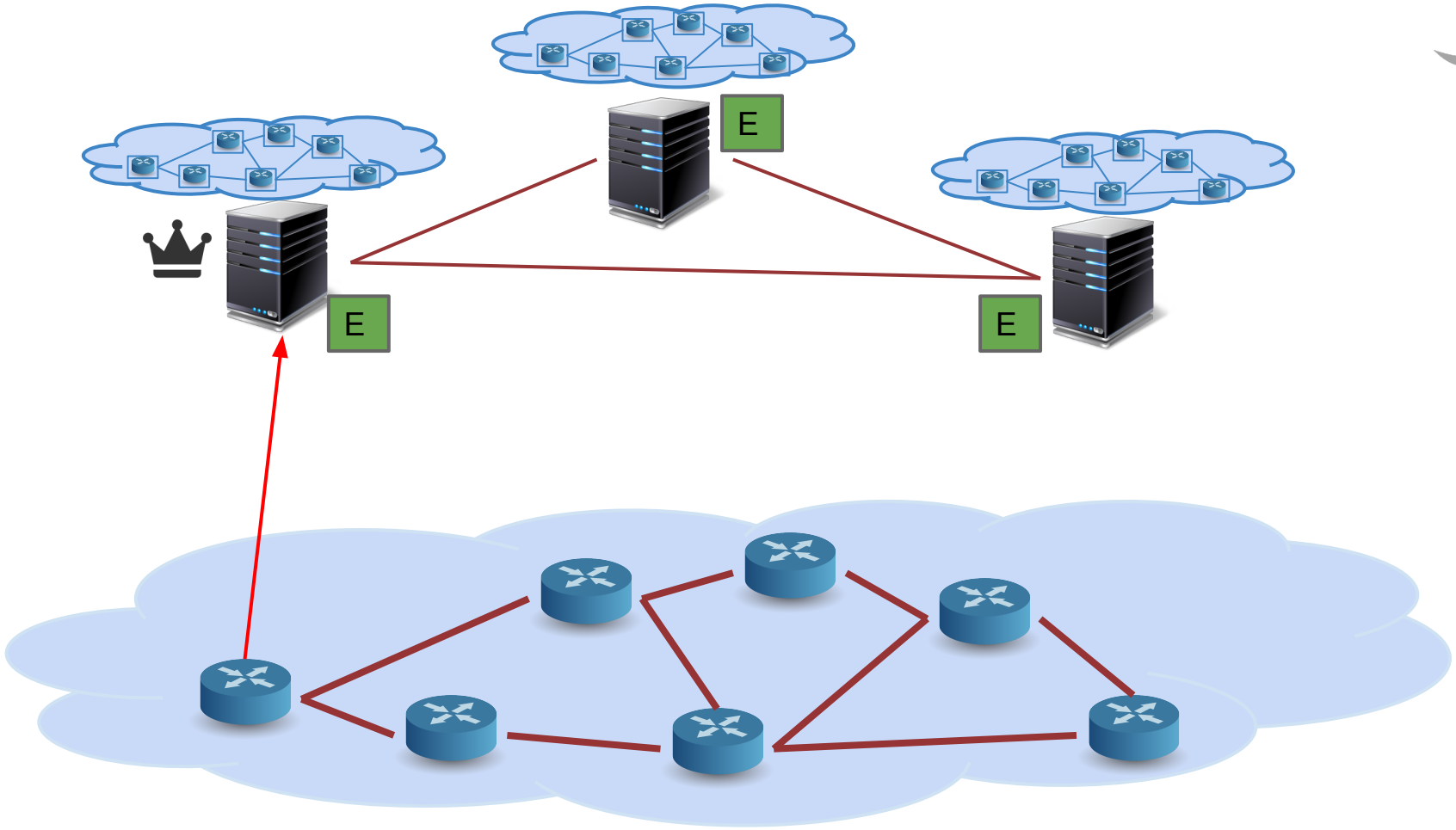


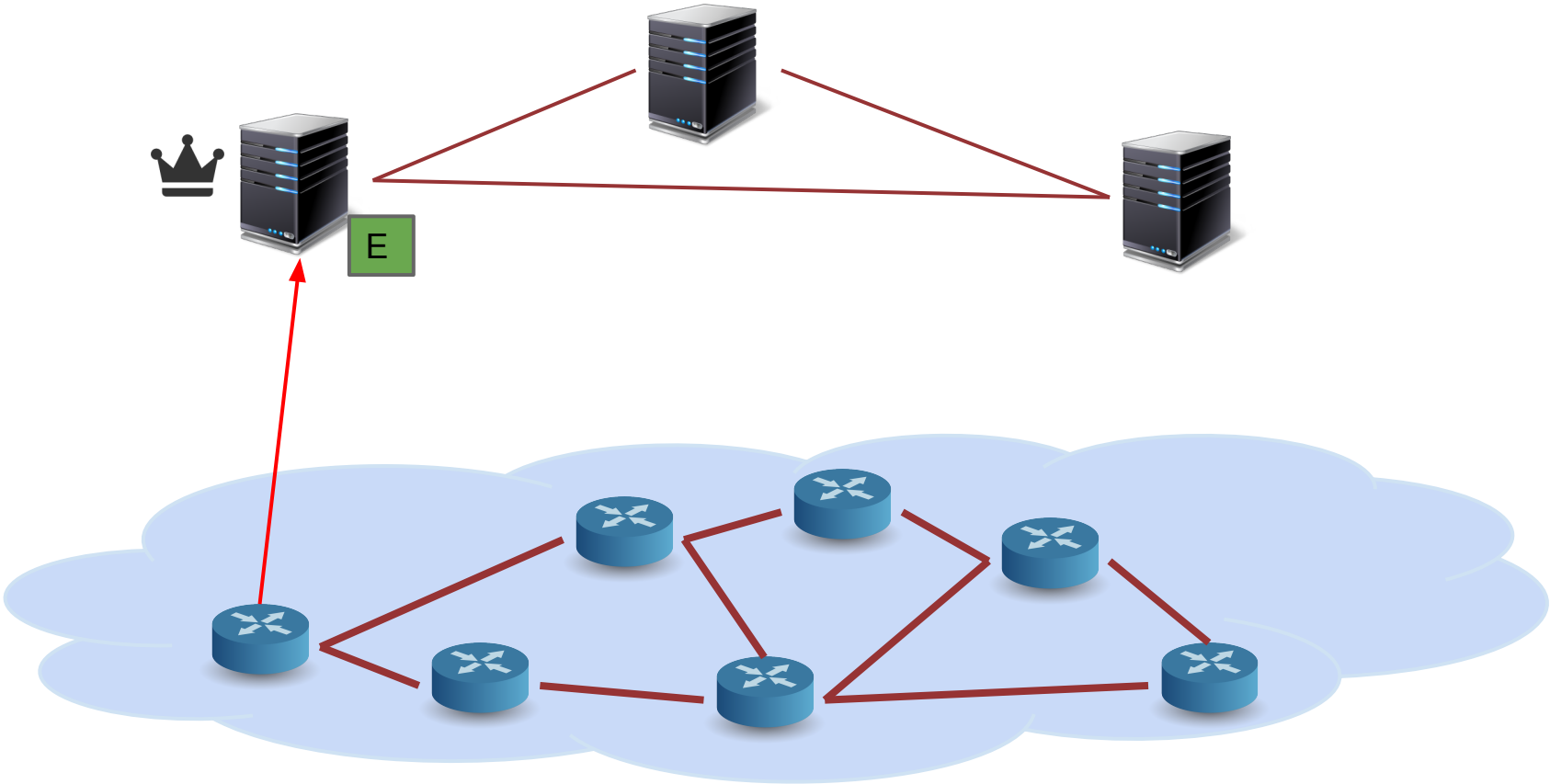
Events are Switch/Port/Link up/down

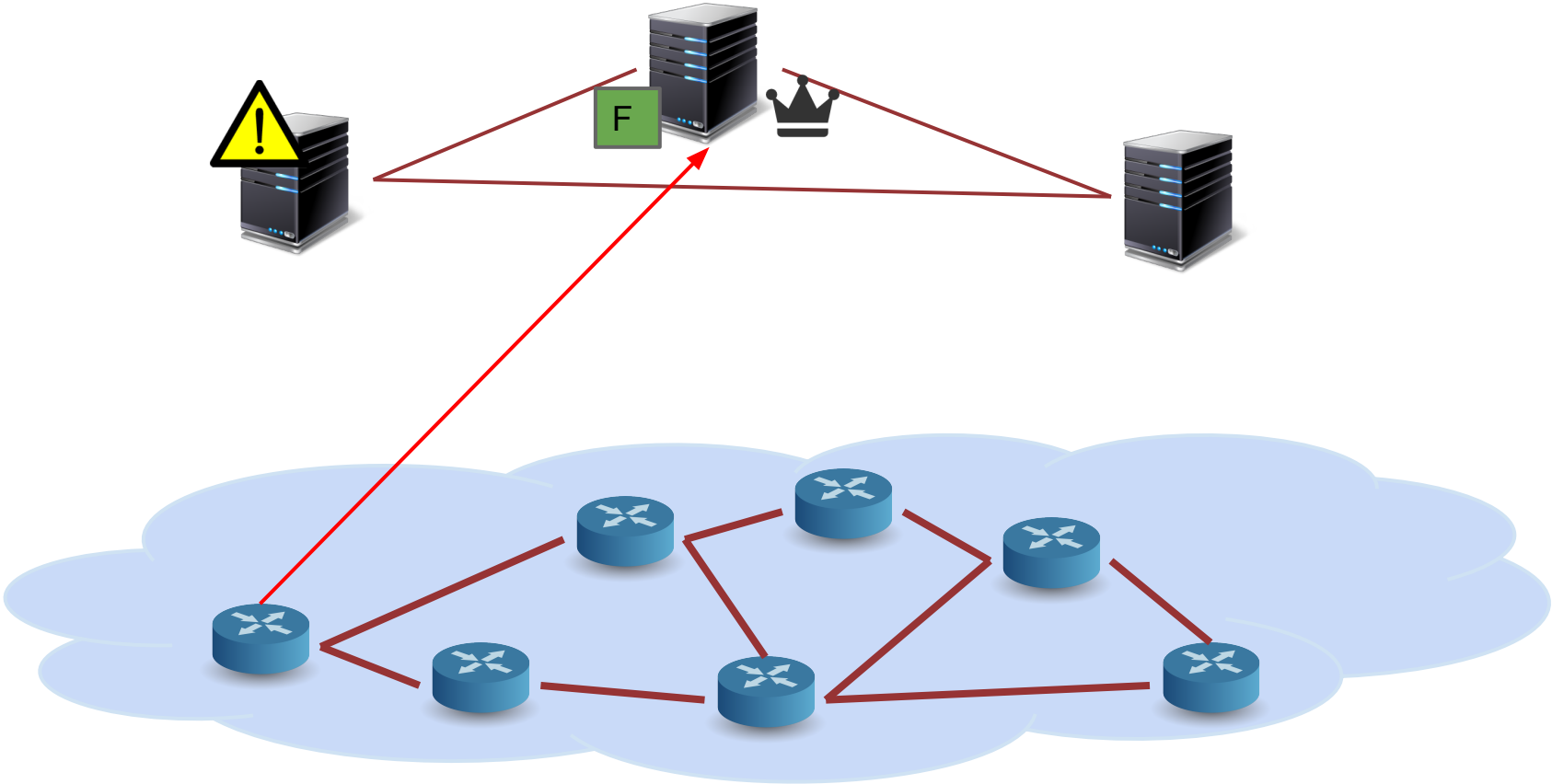


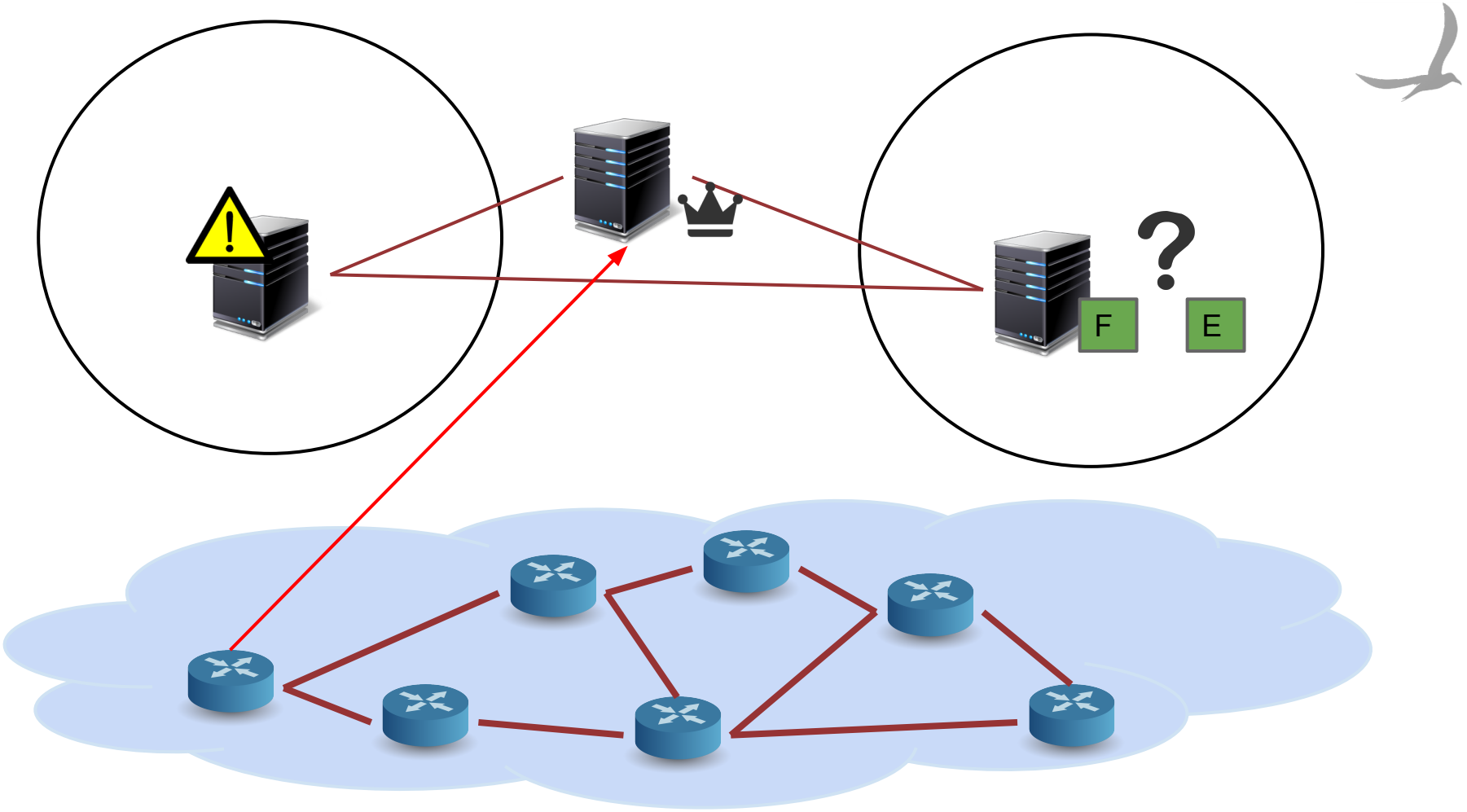


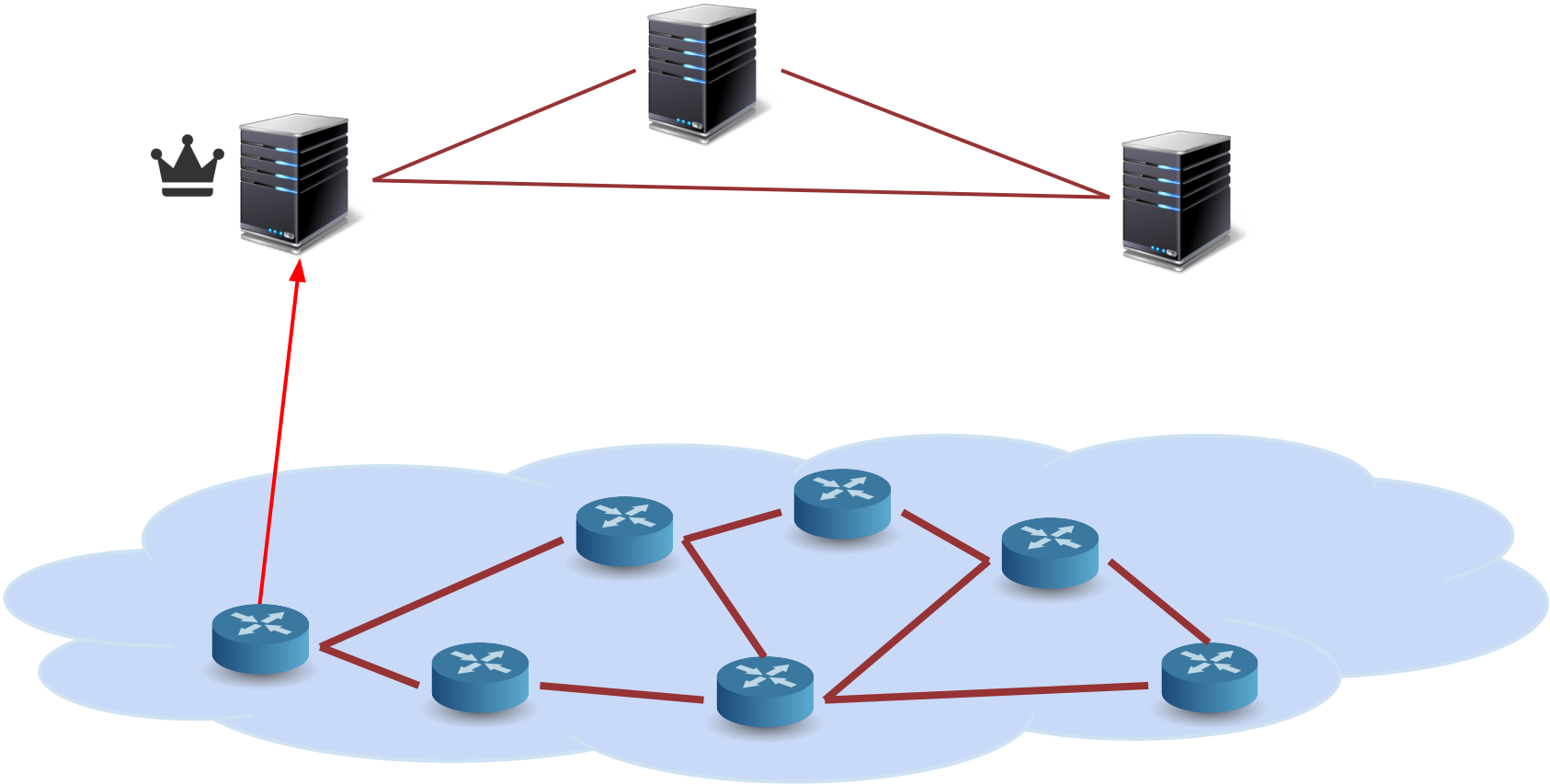

E

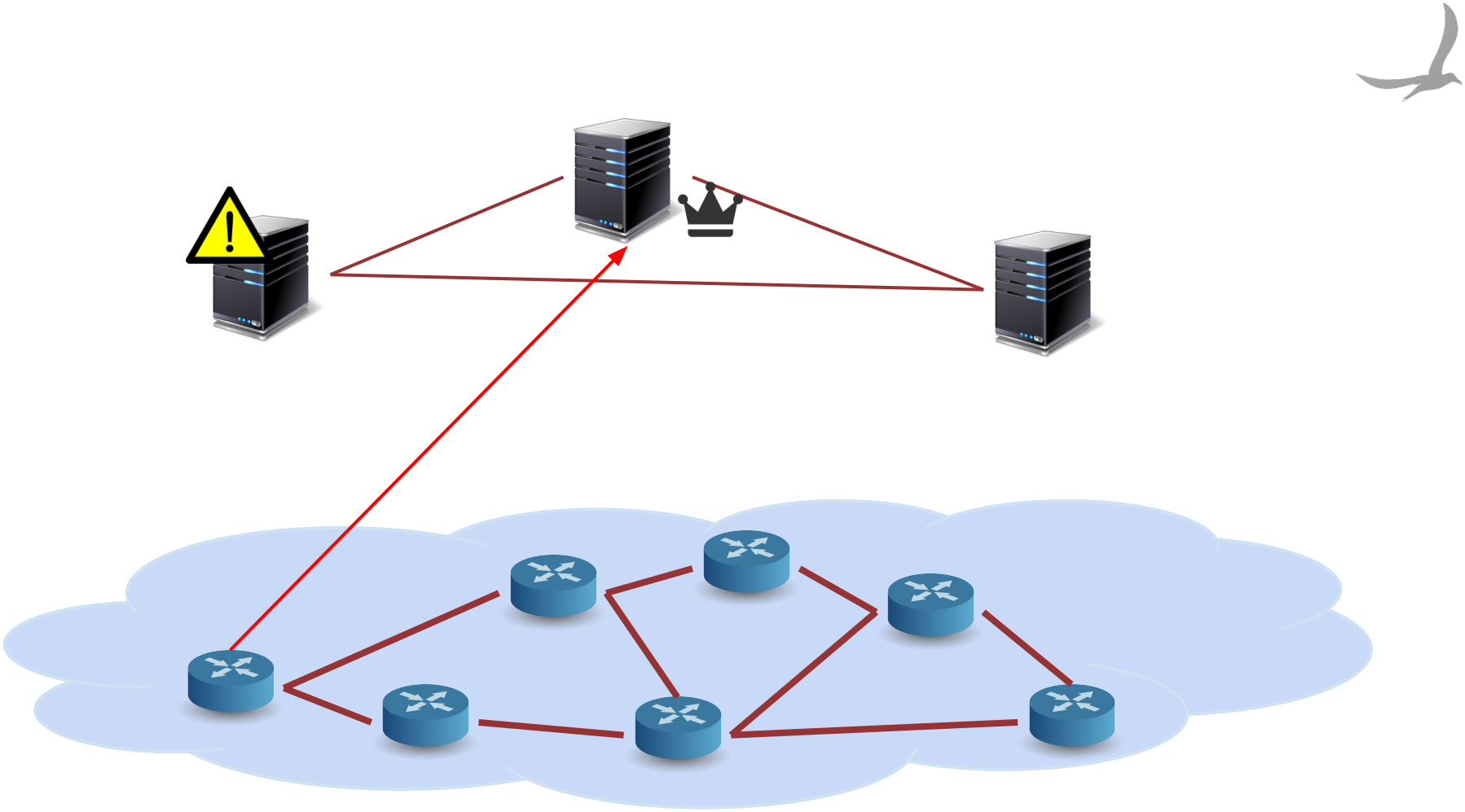

E
E
E


E

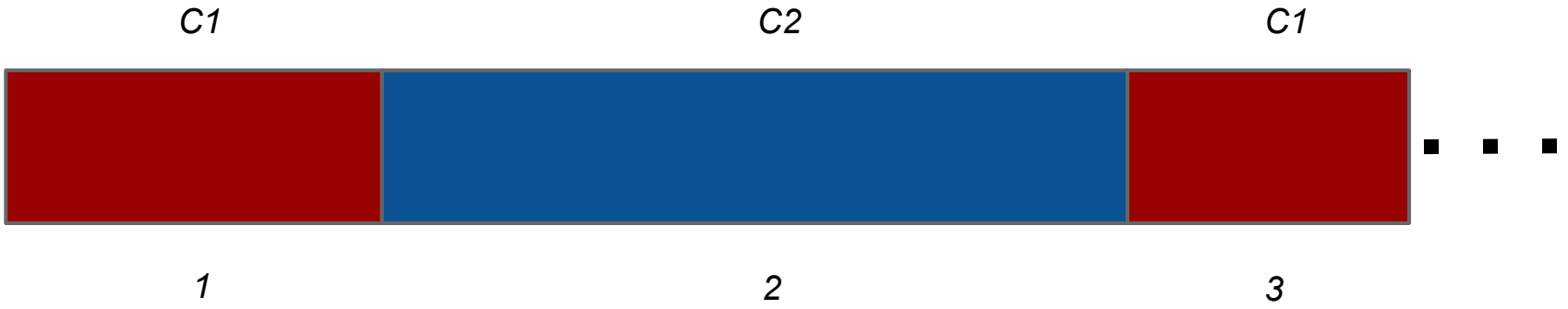

F


F
E



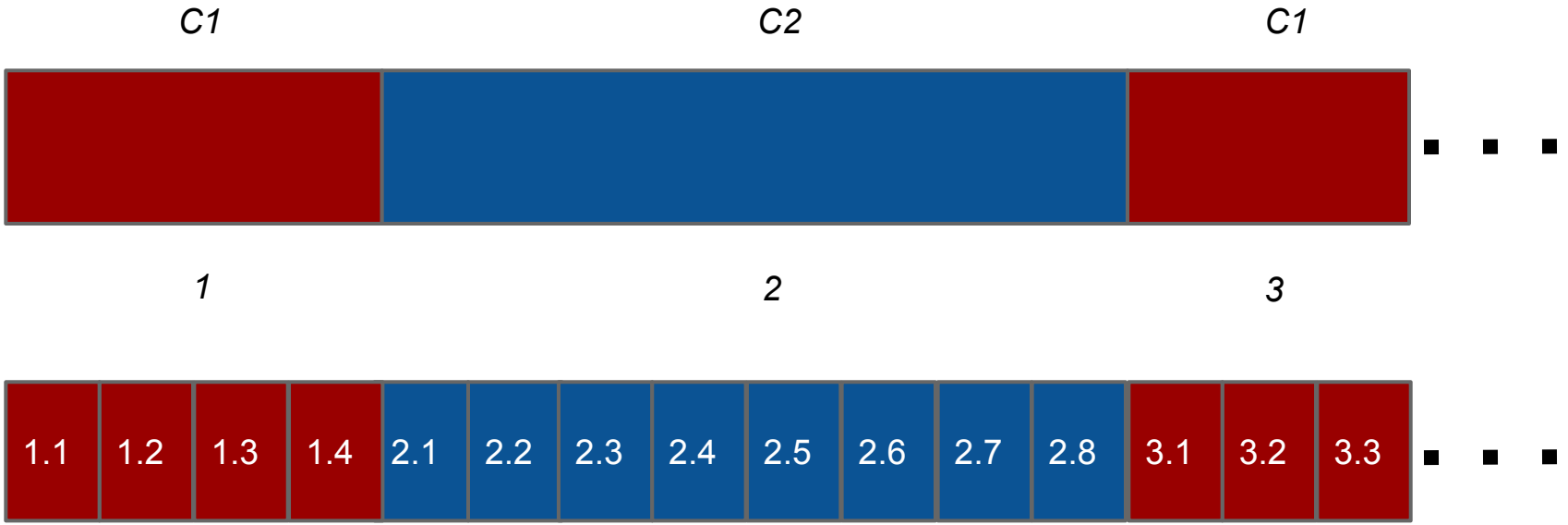


Switch Mastership Terms



We track this in a strongly consistent store

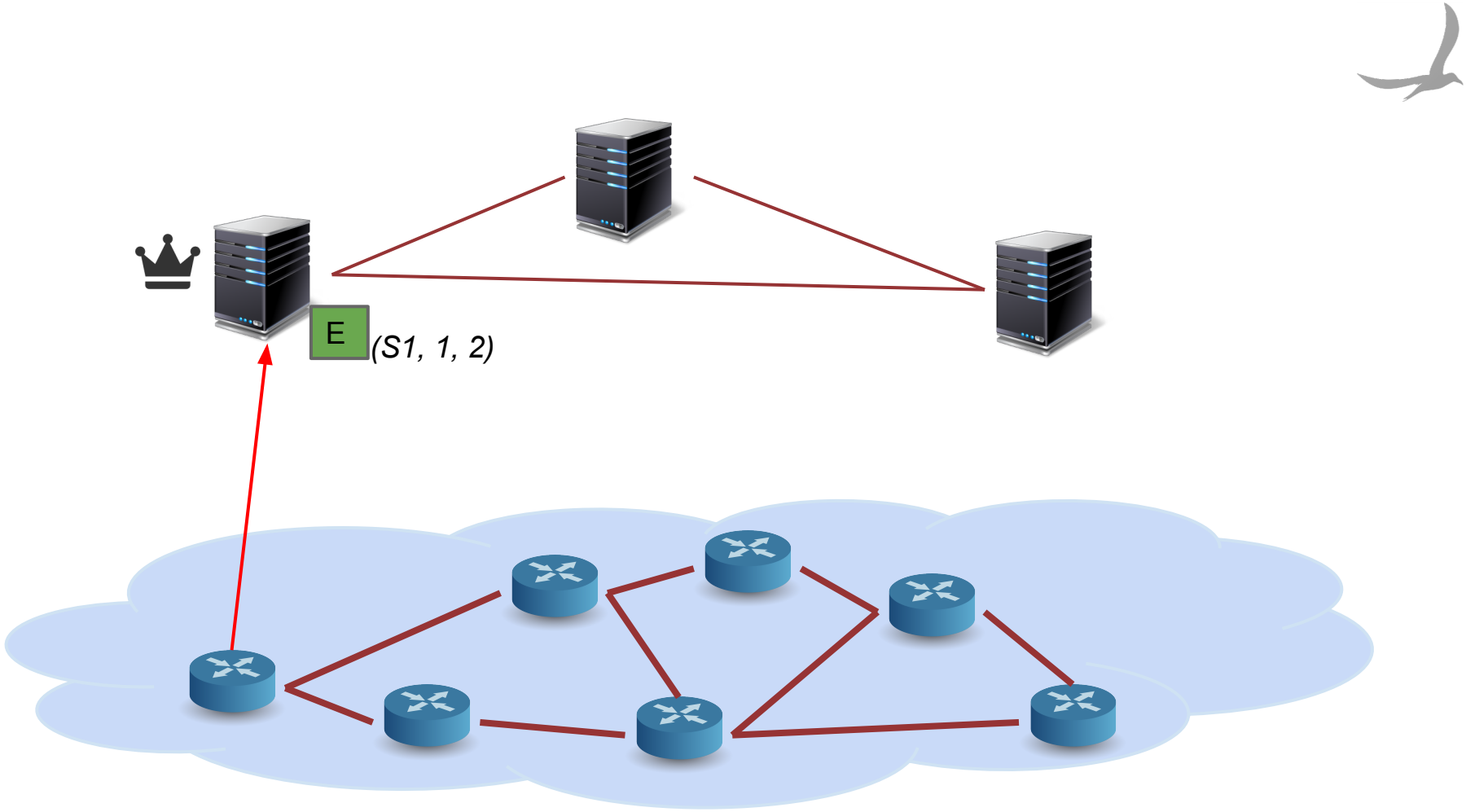
Switch Event Numbers

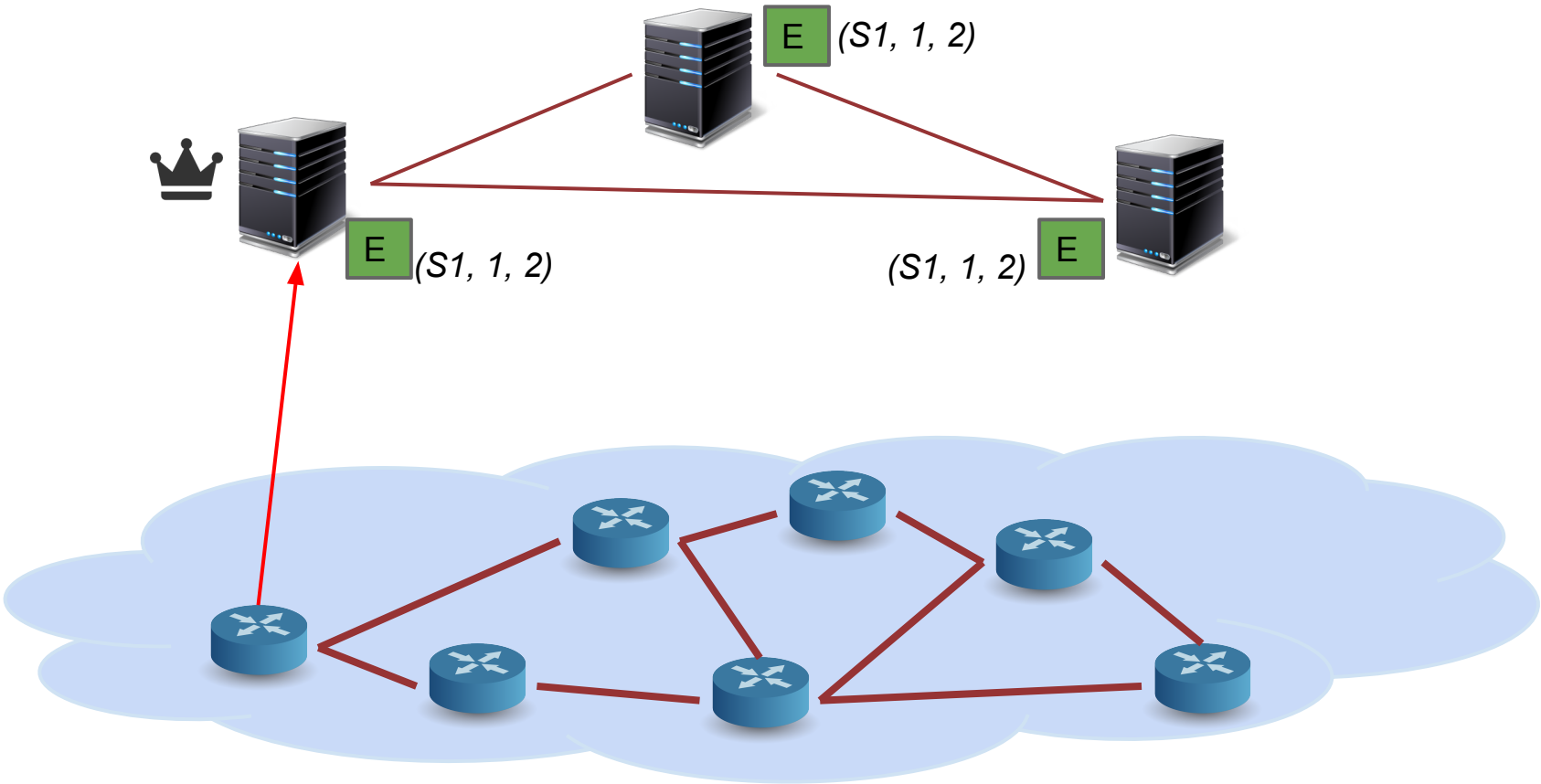


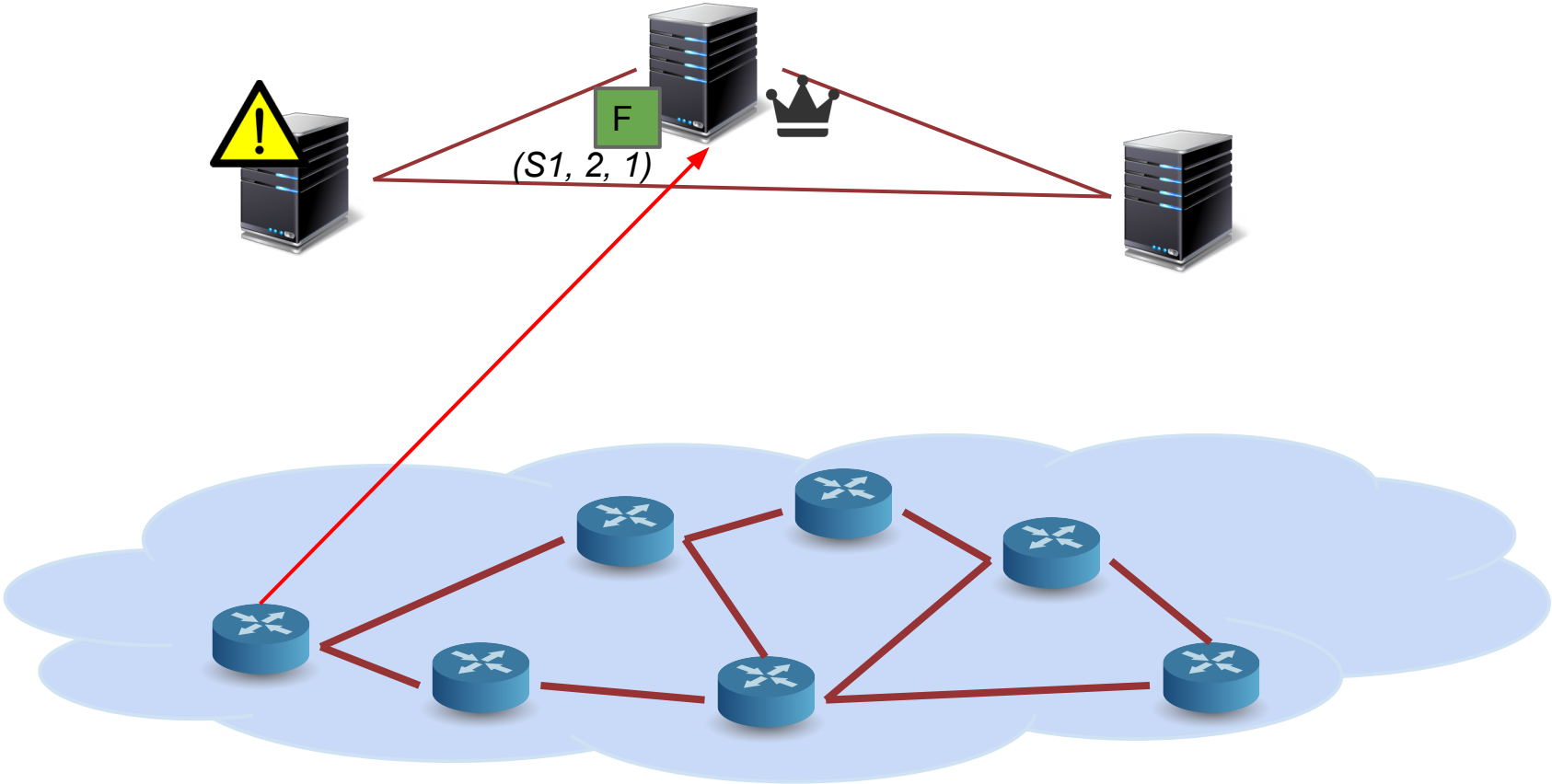
Partial Ordering of Topology Events

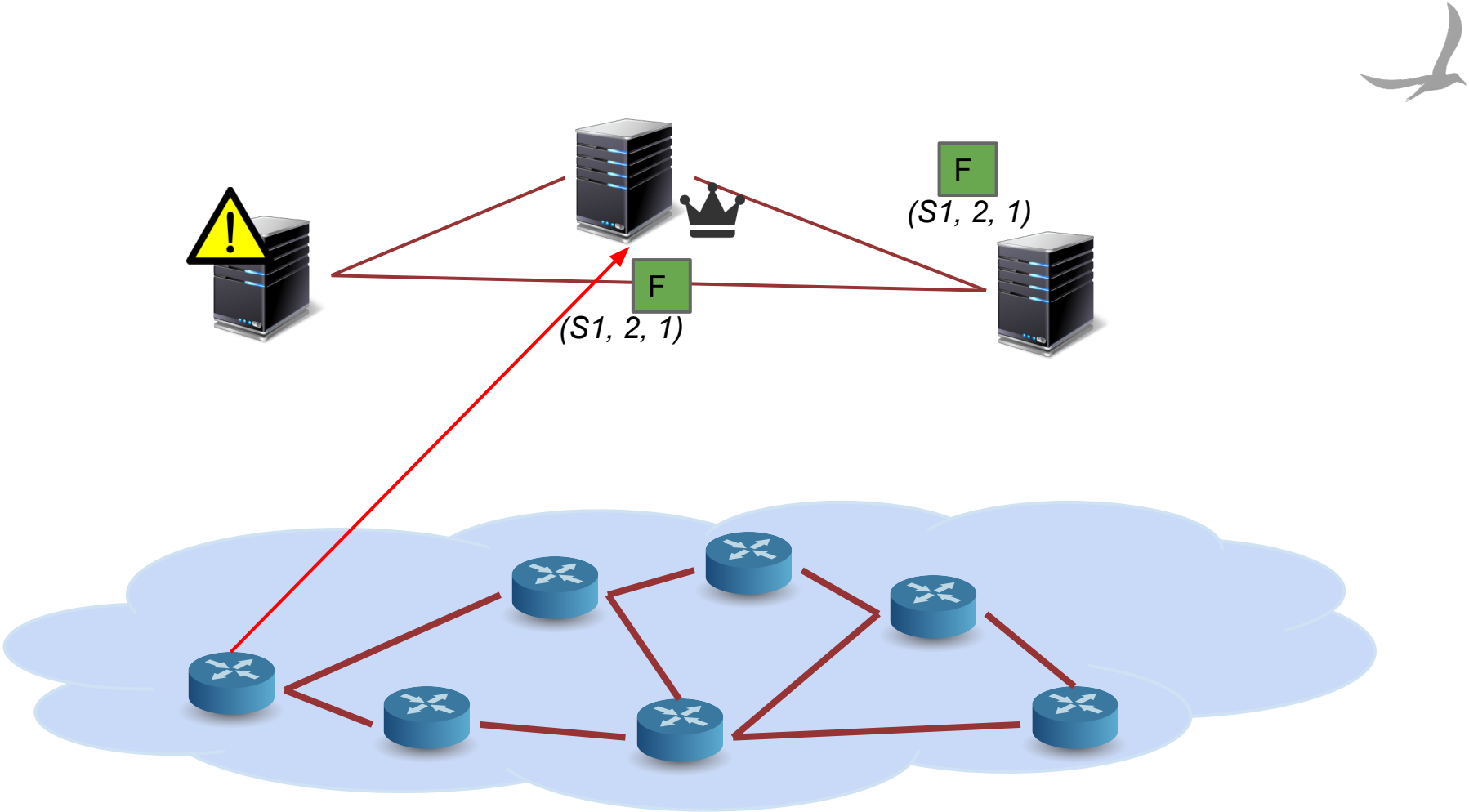
Each event has a unique logical timestamp

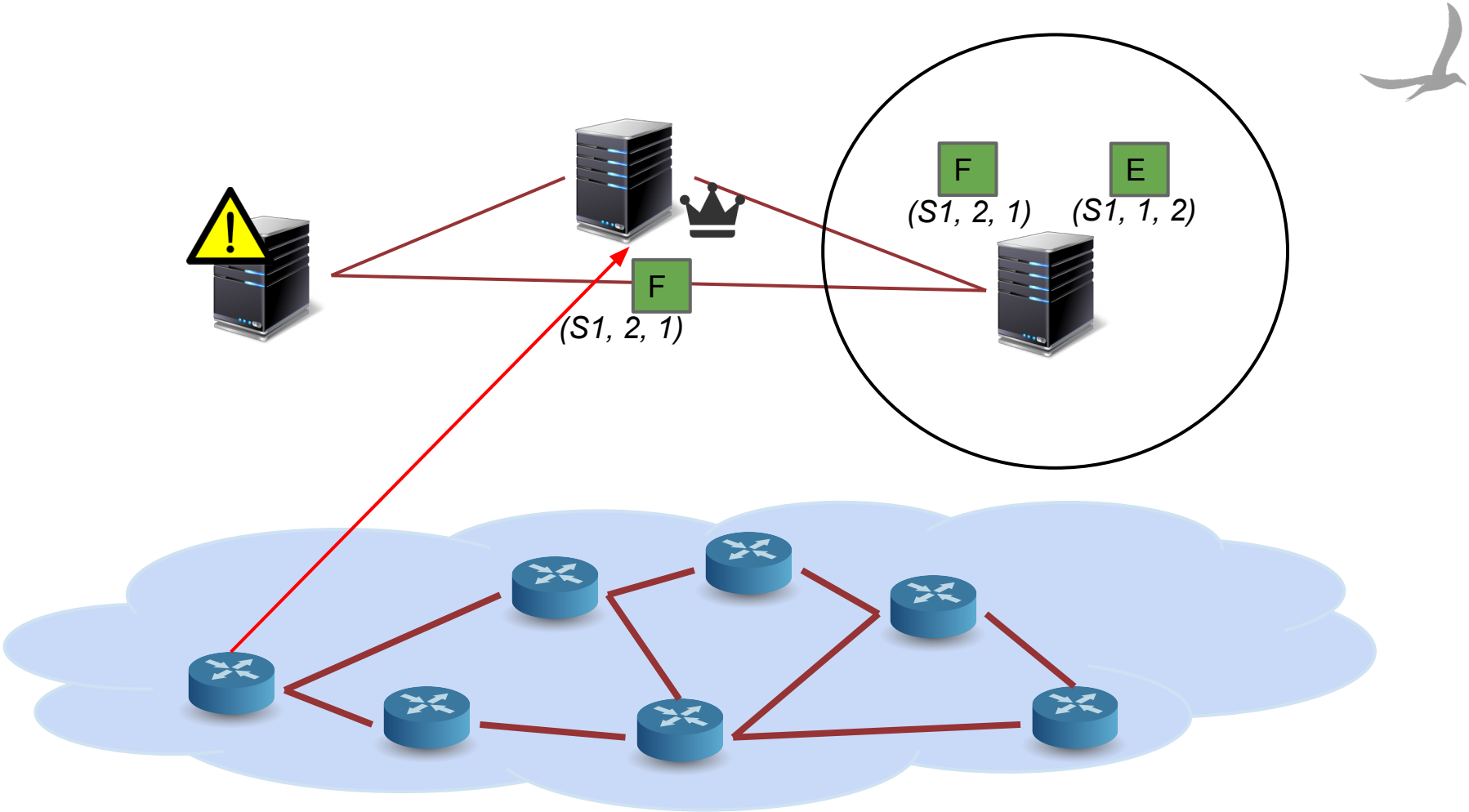
(Switch ID, Term Number, Event Number)

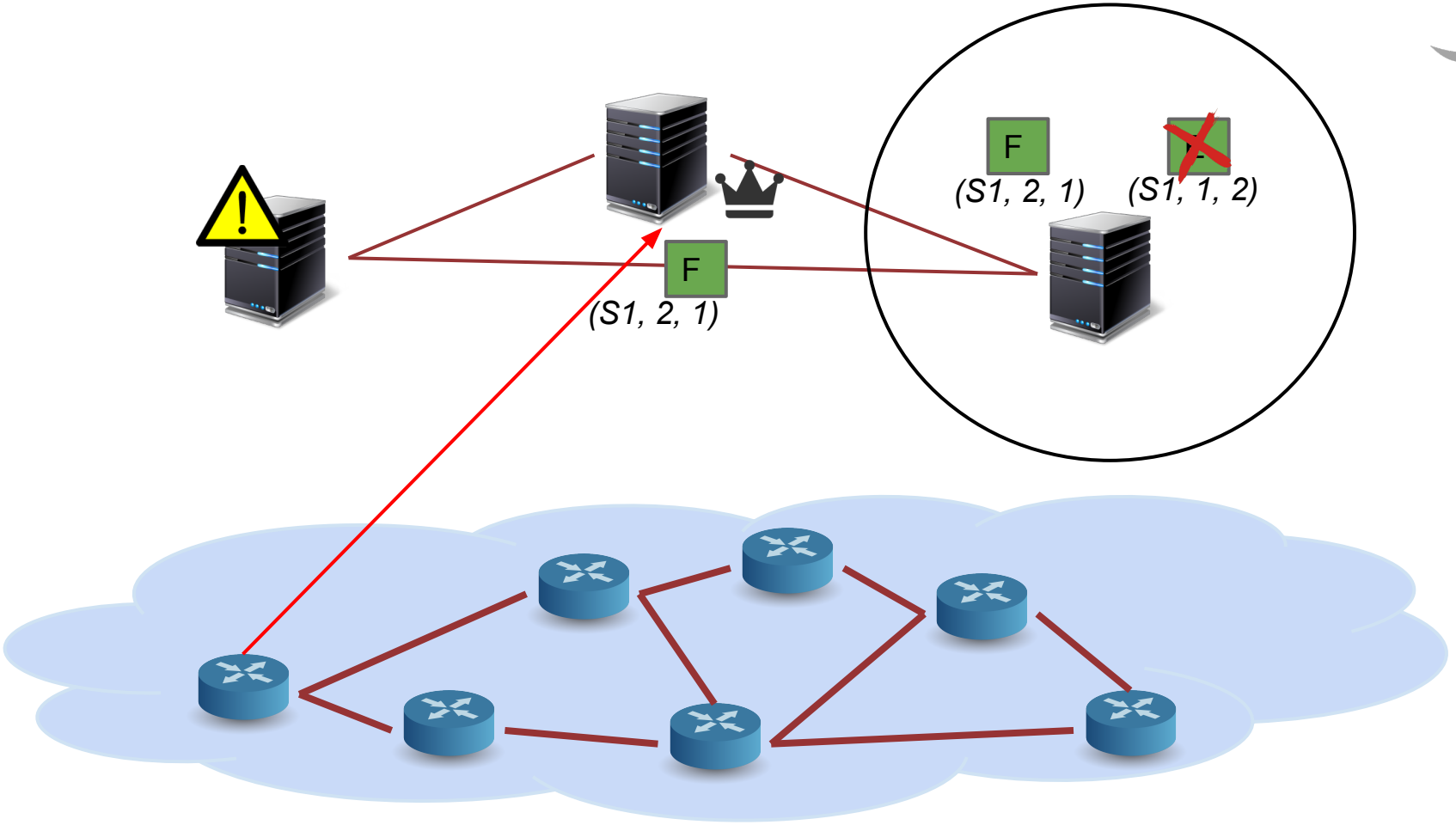

E
(S1, 1, 2)


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

There is one additional step...

What about dropped messages?

Anti-Entropy

Lightweight, Gossip style, peer-to-peer

Quickly bootstraps newly joined nodes

A model for state tracking

If you are the observer, Eventual Consistency is the best option

View should be consistent with the network, not with other views

Hiding the complexity

```
EventuallyConsistentMap<K, V, T>
```

Plugin your own timestamp generation

Other Control Plane state...

- Switch to Controller mapping
- Resource Allocations
- Flows
- Various application generated state

In all case strong consistency is either required or highly desirable

Consistency => Coordination

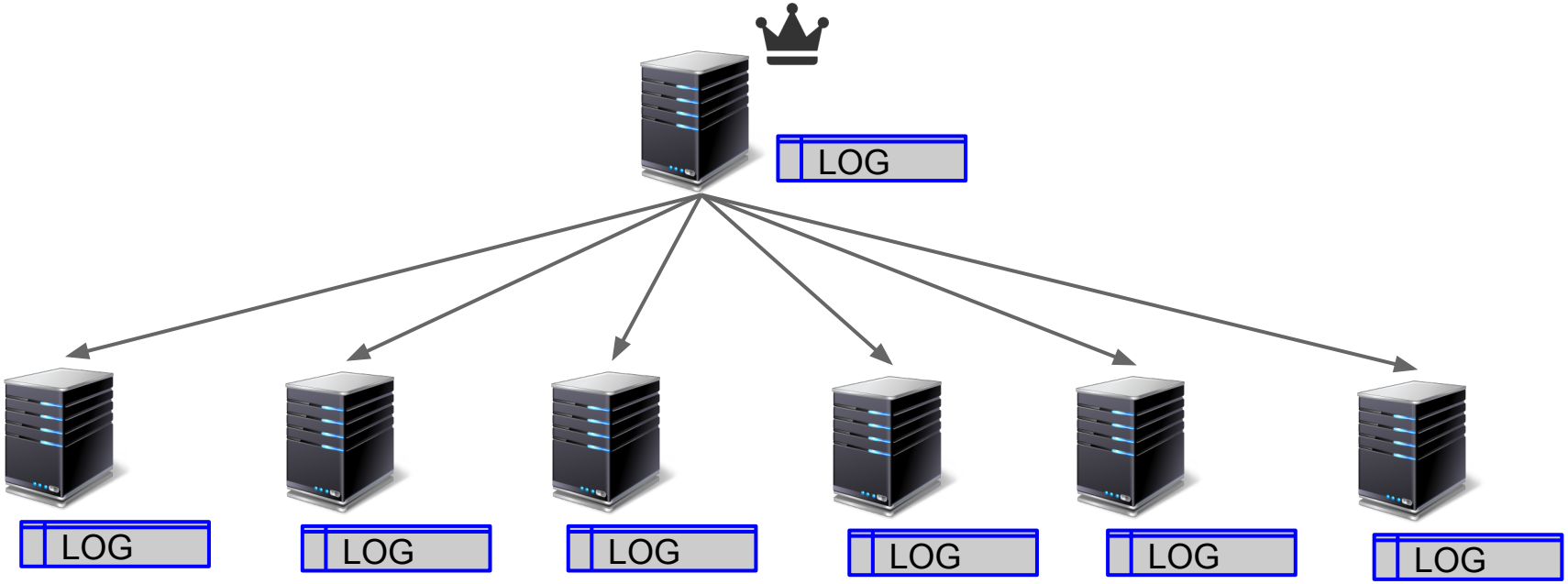
2PC

- All participants need to be available

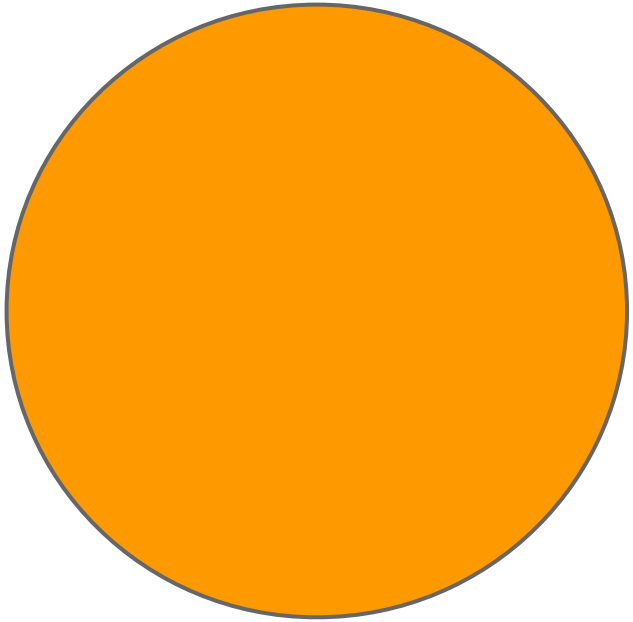
Consensus

- Only a majority need to participate

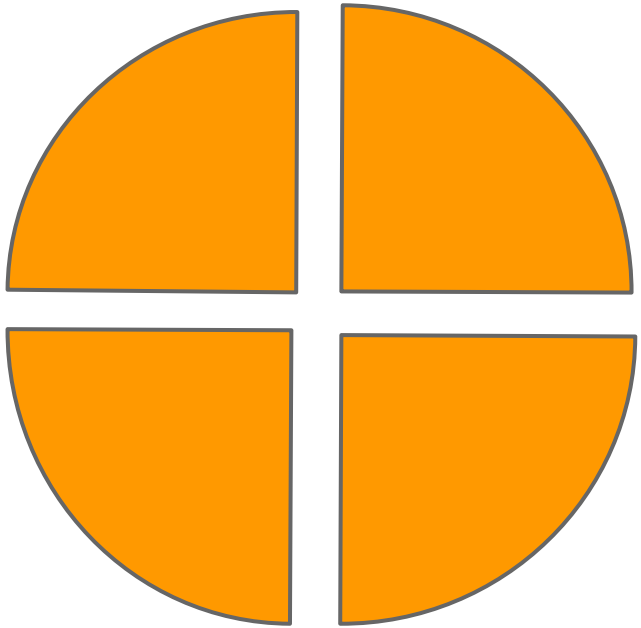
State Consistency through Consensus



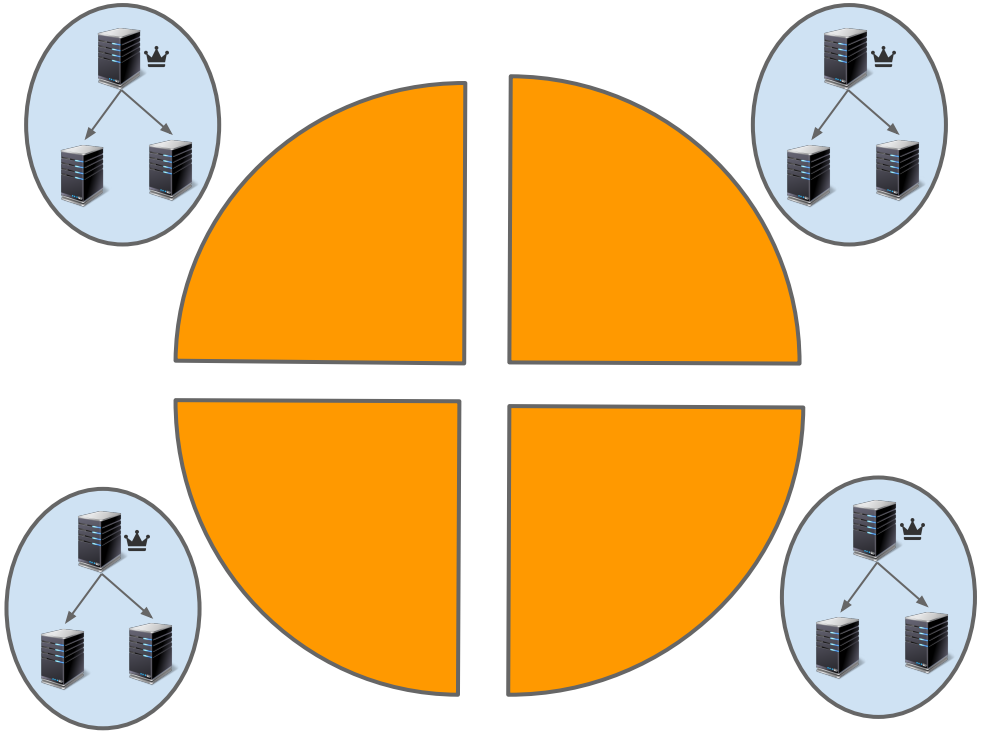
Scaling Consistency



Scaling Consistency



Scaling Consistency



Consistency Cost

- Atomic updates within a shard are “cheap”
- Atomic updates spanning shards use 2PC

Hiding Complexity

```
ConsistentMap<K, V>
```

Outline

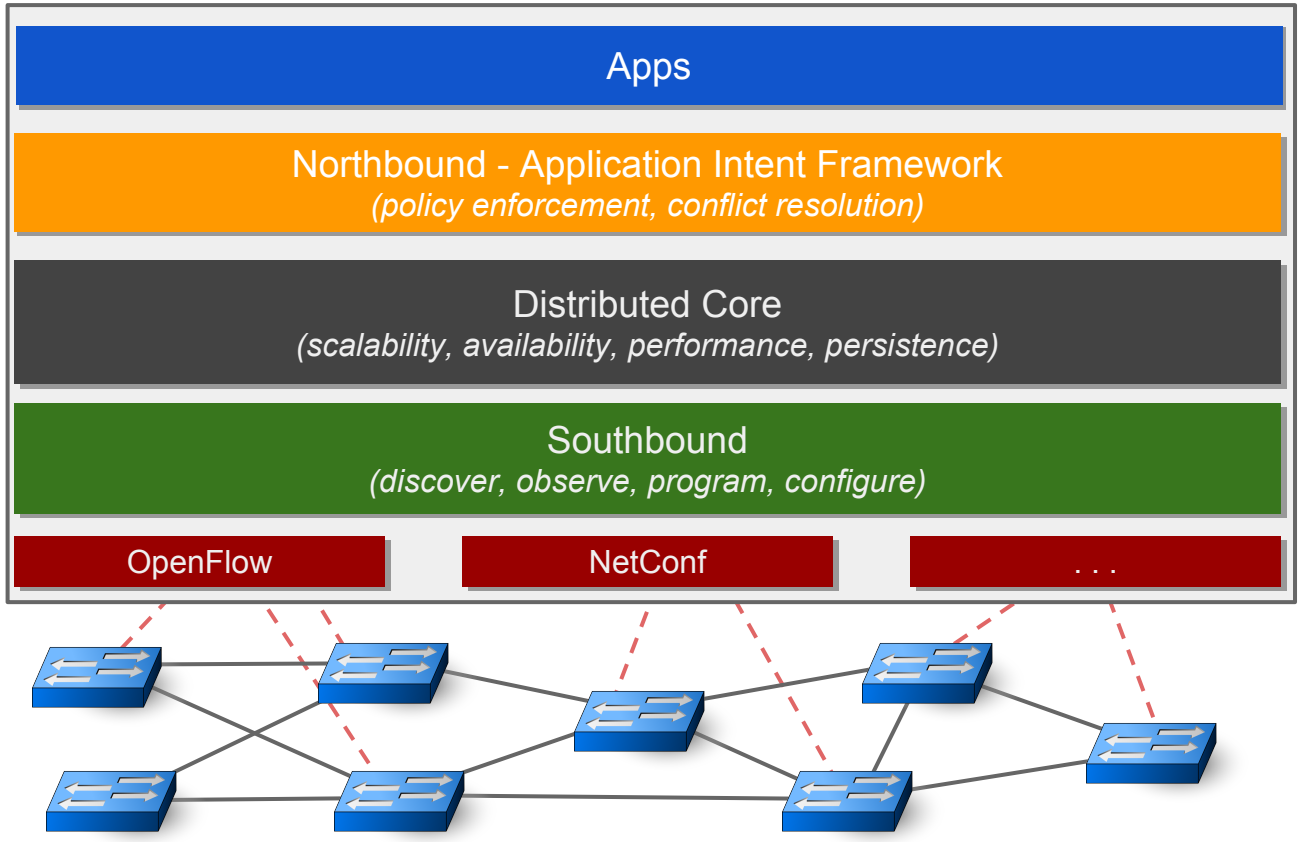
- *A simple example as motivation*
- *ONOS Architecture*
- *Distributed primitives for managing state*
- **APIs and a template for implementation**
- Intent Framework
- Live Demo
- Q/A

ONOS Architecture Tiers

Northbound Abstraction:
- network graph
- application intents

Core:
- distributed
- protocol independent

Southbound Abstraction:
- generalized OpenFlow
- pluggable & extensible



ONOS Service APIs

"Southbound"

- Device
- Link
- Host
- Statistics
- Flow Rule
- Packet
- Resource
- Providers

"Core"

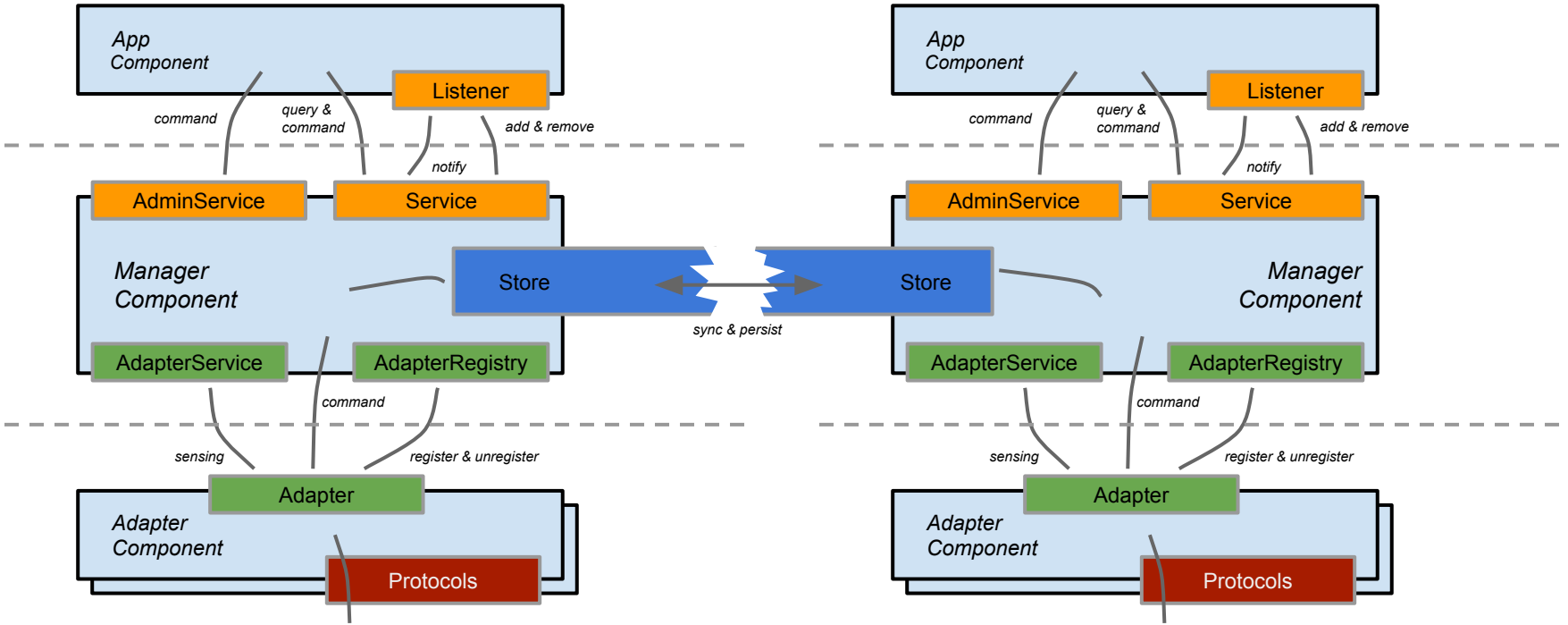
- Core
- Application
- Cluster
- Communication
- Event Delivery
- Storage
- Leadership
- Mastership

...

"Northbound"

- Topology
- Path
- Intent

ONOS Core Subsystem Structure



Intent Framework

- Programming Abstraction
  - Intents
  - Compilers
  - Installers
- Execution Framework
  - Intent Service
  - Intent Store

Intents

- Provide 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

Intent Example

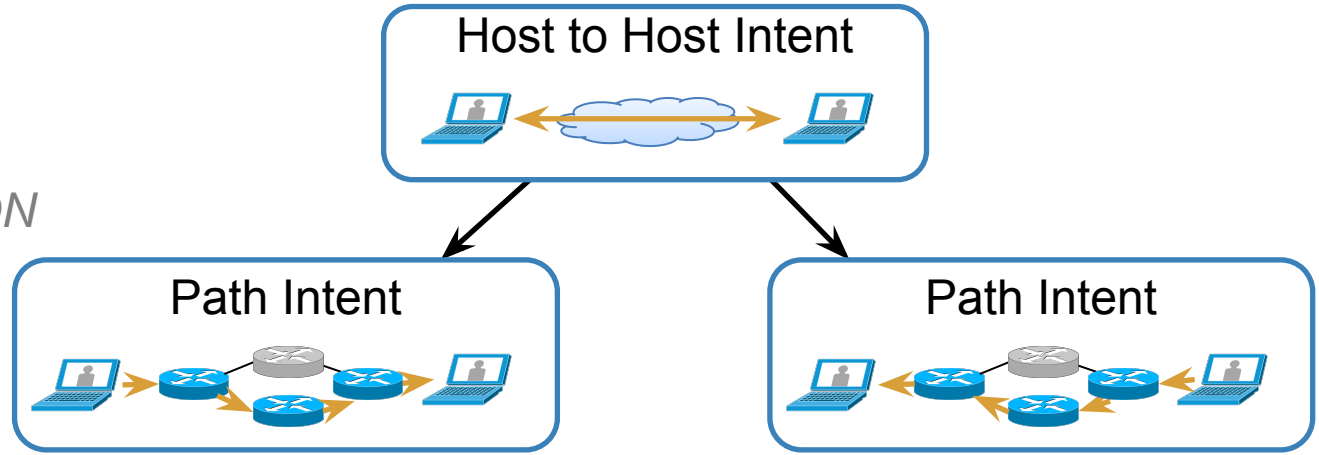

```
public final class HostToHostIntent extends ConnectivityIntent {

   private final HostId one;
   private final HostId two;

   public HostToHostIntent(ApplicationId appId,
                           HostId one, HostId two);

   public HostToHostIntent(ApplicationId appId, Key key,
                           HostId one, HostId two,
                           TrafficSelector selector,
                           TrafficTreatment treatment,
                           List<Constraint> constraints);
}
```

Intent Example



Intent Compilers

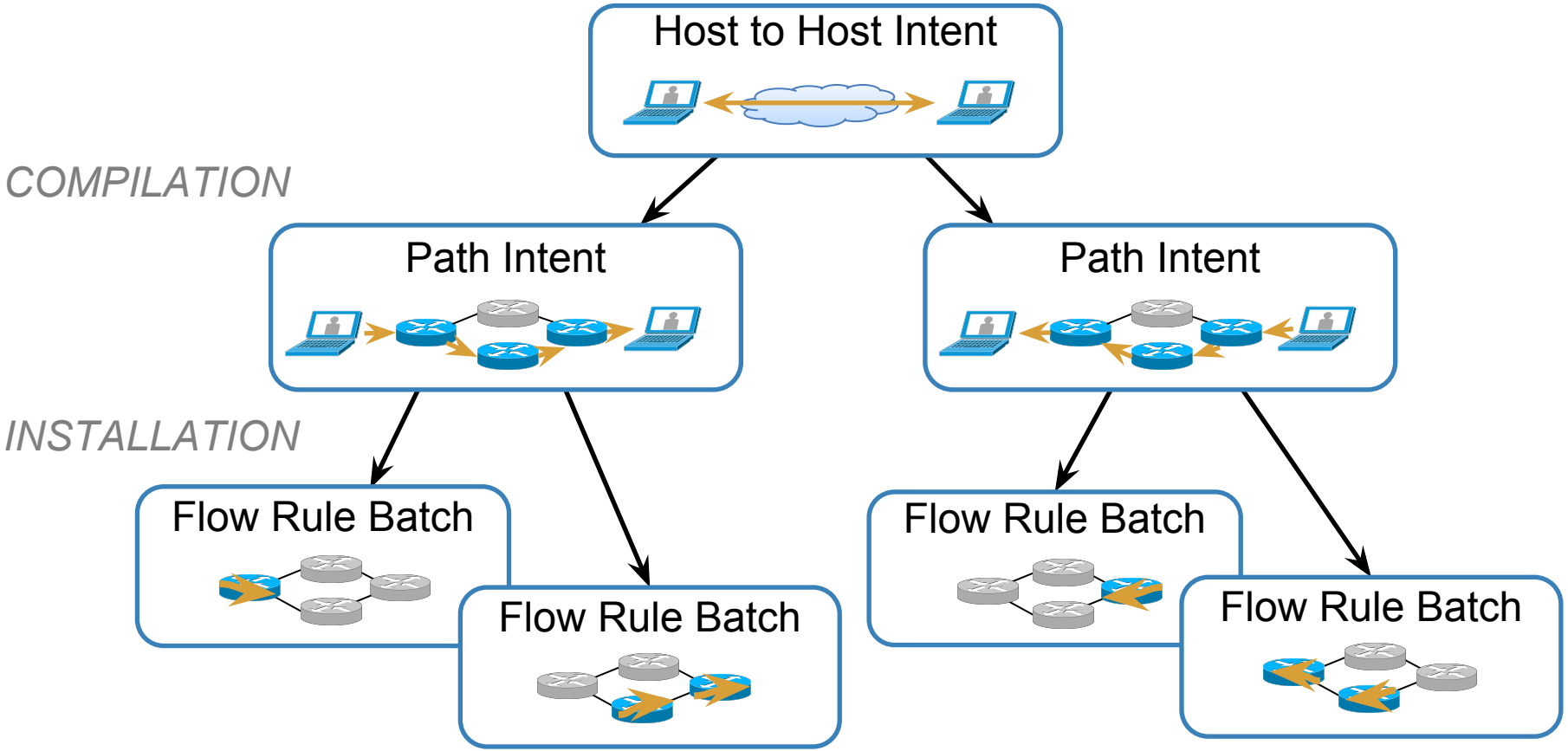
- Produce more specific Intents given the environment
- Works on high-level network objects, rather than device specifics
- “Stateless” – free from HA / distribution concerns

```
interface IntentCompiler<T extends Intent> {
    List<Intent> compile(T intent);
}
```

Intent Example


```
public class HostToHostIntentCompiler extends ConnectivityIntentCompiler<HostToHostIntent> {
   @Override
   public List<Intent> compile(HostToHostIntent intent, ...) {
       Host one = hostService.getHost(intent.one());
       Host two = hostService.getHost(intent.two());
       // compute paths
       Set<Path> paths = pathService.getPaths(intent.one(), intent.two(), intent.constraints());
       // select paths
       Path pathOne = bestPath(intent, paths);
       Path pathTwo = invertPath(pathOne); // reverse path using same links
       // create intents
       PathIntent fwdPath = createPathIntent(pathOne, one, two, intent);
       PathIntent revPath = createPathIntent(pathTwo, two, one, intent);
       return Arrays.asList(fwdPath, revPath);
   }
}
```

Intent Example

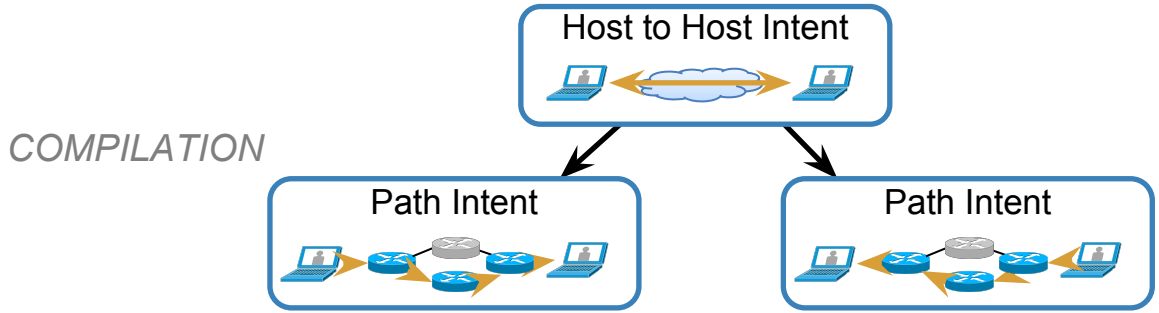


Intent Installers

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

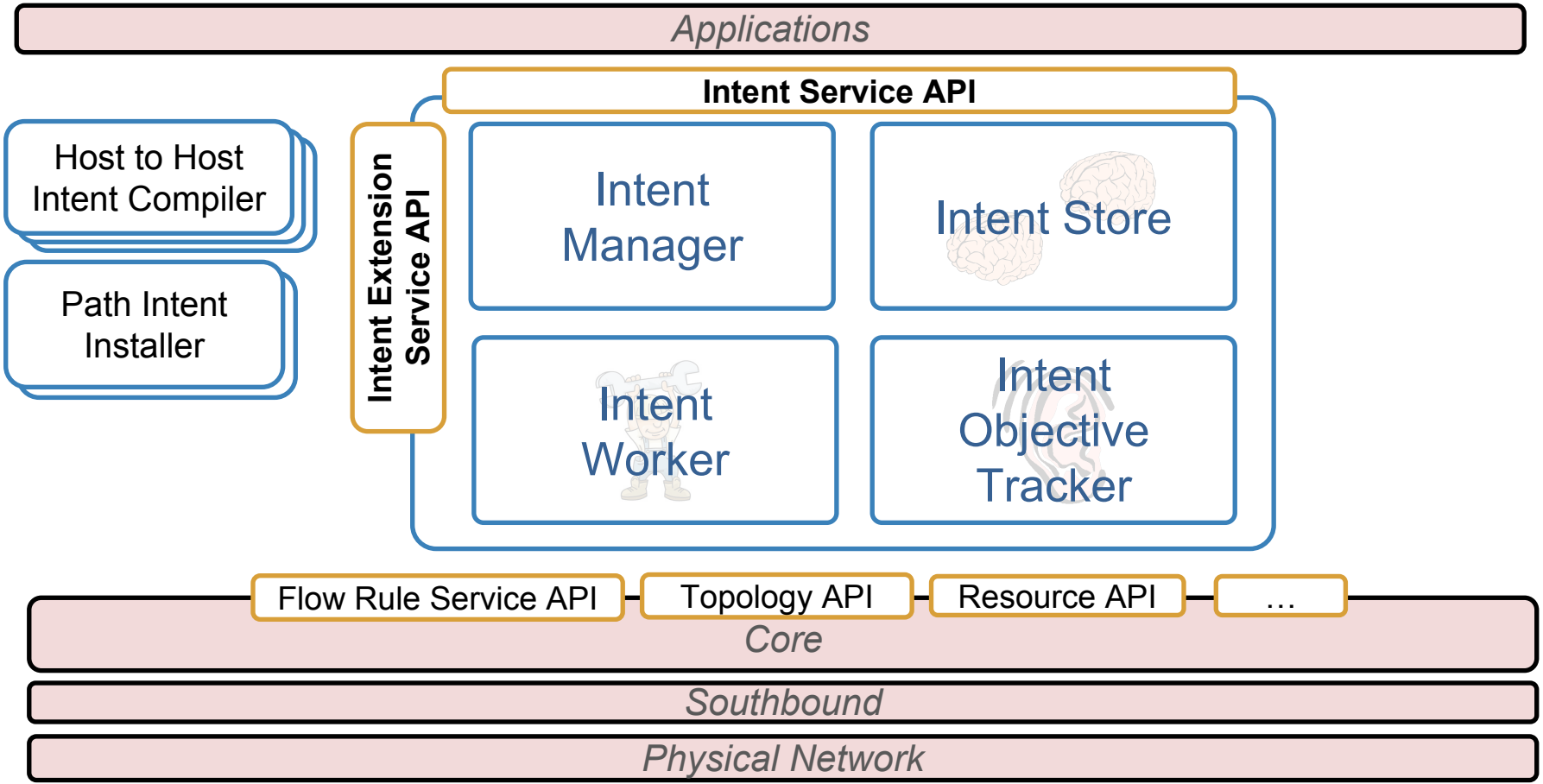
```
interface IntentInstaller<T extends Intent> {
   List<Collection<FlowRuleOperation>> install(T intent);
   List<Collection<FlowRuleOperation>> uninstall(T intent);
   List<Collection<FlowRuleOperation>> replace(T oldIntent, T newIntent);
}
```

Intent Example




```
public class PathIntentInstaller implements IntentInstaller<PathIntent> {
   @Override
   public List<Collection<FlowRuleOperation>> install(PathIntent intent) {
      LinkResourceAllocations allocations = allocateResources(intent);
      Set<FlowRuleOperation> rules = Sets.newHashSet();
      for (Link link : intent.path().links()) {
         FlowRule rule = createFlowRule(link, intent);
         rules.add(new FlowRuleOperation(rule, FlowRuleOperation.Type.ADD));
      }
      return Lists.newArrayList(rules);
   @Override
   public List<Collection<FlowRuleOperation>> uninstall(PathIntent intent) {...}
   @Override
   public List<Collection<FlowRuleOperation>> replace(PathIntent oldIntent, PathIntent newIntent)  {...}
}
```

Intent Framework Design



Intent Framework API

```
package org.onosproject.net.intent; // filepath: onos/core/api/net/intent/

public abstract class Intent {
   private final IntentId id;           // internal, unique ID
   private final ApplicationId appId;   // originating application
   private final Key key;               // user define key for this "policy"
   private final Collection<NetworkResource> resources;
   // constructors and getters
}

public interface IntentService {
   void submit(Intent intent);
   void withdraw(Intent intent);
   Intent getIntent(Key key);
   // other methods and listener registration
}
```

Intent Manager

```
package org.onosproject.net.intent.impl; // filepath: onos/core/net/intent/impl
@Component
@Service
public class IntentManager implements IntentService, IntentExtensionService {
   // Dependencies
   protected CoreService coreService; // used to generate blocks of unique, internal intent IDs
   protected IntentStore store; // stores all intents in the system;
                                // delegates work to be be done to master
   protected ObjectiveTrackerService trackerService; // listens to network events and
                                                     //   triggers recomputation when required
   protected EventDeliveryService eventDispatcher; // used for managing listeners and
                                                   //   dispatching events
   protected FlowRuleService flowRuleService; // responsible for installing flow rules and
                                              //   reporting success or failure
   // ... implementation of service methods
}
```

Intent Store

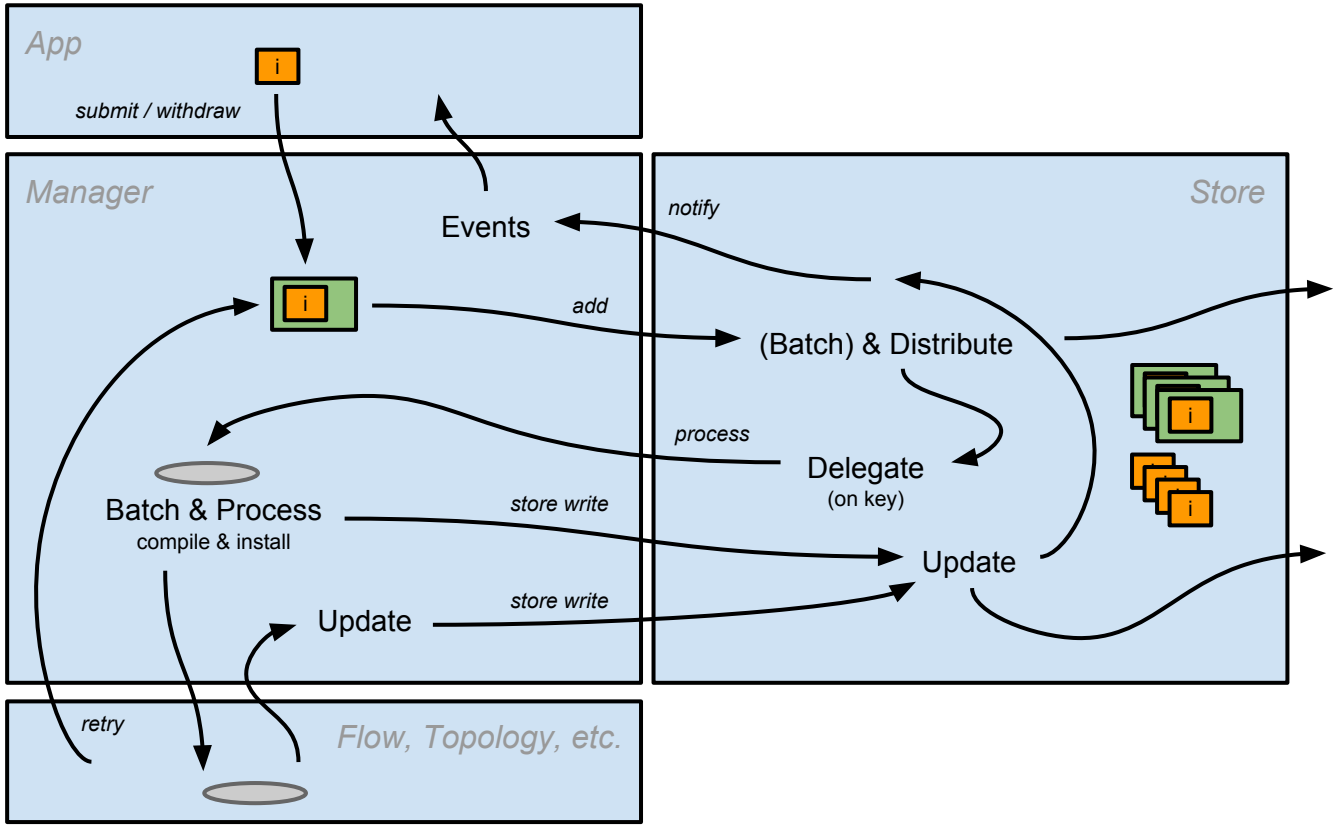
- Key to HA and Scale-Out
- Work is delegated to specific cluster nodes through logical partitions based on Intent key
- Execution steps designed to be idempotent
  - However, when resources are required, duplicate request will be detected and handled
- Failures can be detected by store, and work can be reassigned by electing new leader for partition

Intent Store

```
package org.onosproject.store.intent.impl; // filepath: onos/core/store/dist/intent/impl
@Component
@Service
public class GossipIntentStore extends AbstractStore<IntentEvent, IntentStoreDelegate>
        implements IntentStore {
   private EventuallyConsistentMap<Key, IntentData> currentMap;
   private EventuallyConsistentMap<Key, IntentData> pendingMap;

   // Dependencies
   protected ClusterService clusterService; // provides information about nodes in cluster
   protected ClusterCommunicationService clusterCommunicator; // provides communication
                                                              //   mechanism for map updates
   protected PartitionService partitionService; // gets partition for Intent by key;
                                                // elects and maintains leader for each partition
   // ... implementation of IntentStore methods
}
```

Intent Subsystem Design



Design Modularity

- Intents, Compilers, and Installers can be defined and loaded dynamically
- Framework implements service API, so the entire framework can be swapped out
- Each component also implements an internal API, so they can be replaced as desired

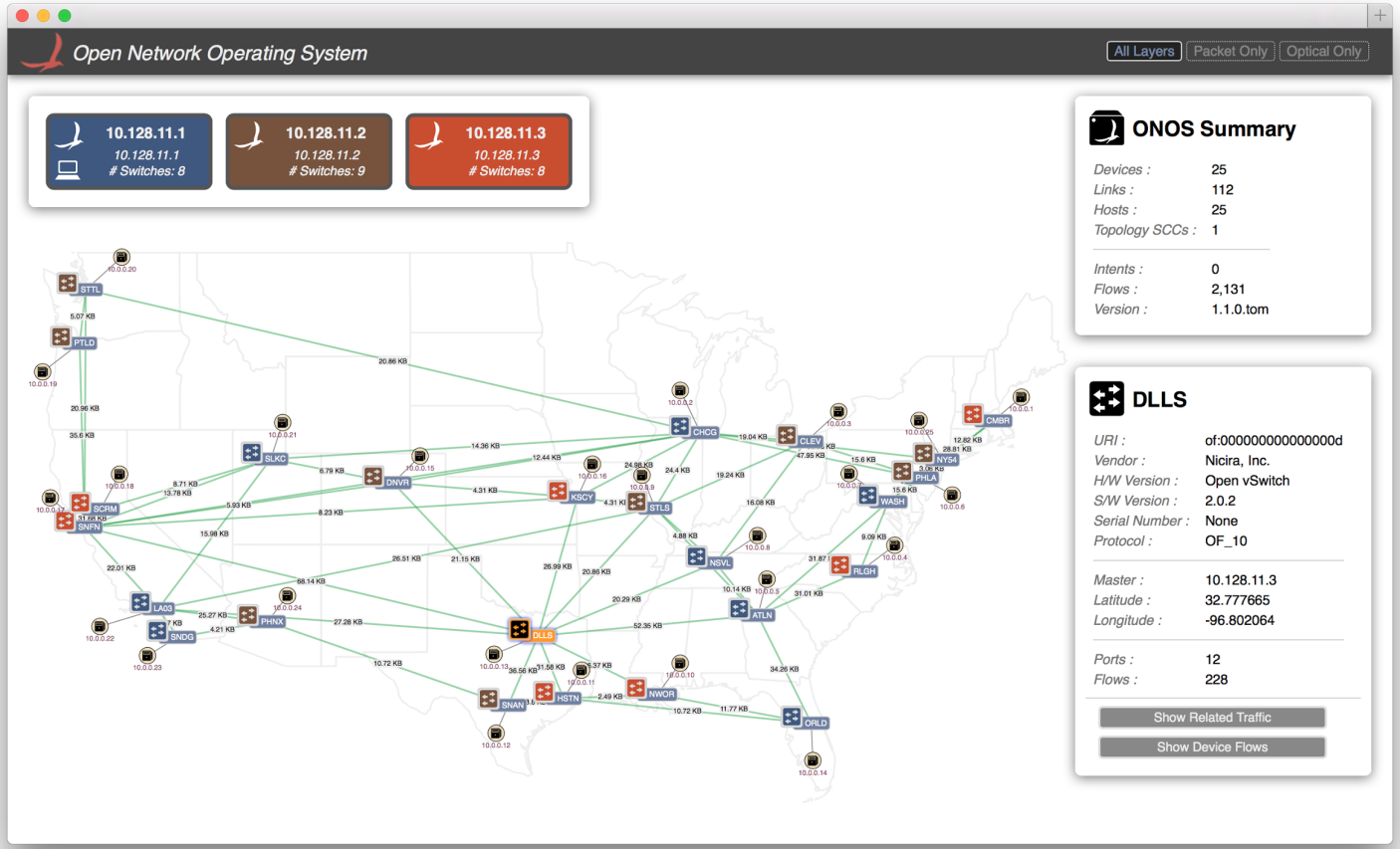
What's Next?

- Join ONOS mailing lists
  - https://wiki.onosproject.org/display/ONOS/ONOS+Mailing+Lists
  - Don’t hesitate to engage with ONOS developers & community
- Try some of the tutorials
  - https://wiki.onosproject.org/display/ONOS/Tutorials+and+Walkthroughs
- Download the source code:
  - https://gerrit.onosproject.org/ (main developer repository)
  - https://github.com/opennetworkinglab/onos (read-only mirror)
- Check out the ONOS API Javadoc
  - http://api.onosproject.org/
- Fix (or file) a bug
  - https://jira.onosproject.org/
- Visit the main project page and wiki
  - http://onosproject.org/ and https://wiki.onosproject.org/

Outline

- *A simple example as motivation*
- *ONOS Architecture*
- *Distributed primitives for managing state*
- *APIs and a template for implementation*
- *Intent Framework*
- **Live Demo**
- Q/A

Live Demo


Open Network Operating System
All Layers
Packet Only
Optical Only
10.128.11.1
10.128.11.1
Switches: 8
10.128.11.2
10.128.11.2
Switches: 9
10.128.11.3
10.128.11.3
Switches: 8
ONOS Summary
Devices : 25
Links : 112
Hosts : 25
Topology SCCs : 1
Intents : 0
Flows : 2,131
Version : 1.1.0.tom
DLLS
URI : of:000000000000000d
Vendor : Nicira, Inc.
H/W Version : Open vSwitch
S/W Version : 2.0.2
Serial Number : None
Protocol : OF_10
Master : 10.128.11.3
Latitude : 32.777665
Longitude : -96.802064
Ports : 12
Flows : 228
Show Related Traffic
Show Device Flows
STTL
PTLD
SCRM
SNPN
SLKC
DNVR
KSCY
CHCG
STLS
NSVL
DLLS
ATLN
CLEV
WASH
NY54
PHLA
CMBR
RLGH
LA03
PHNX
SNDG
SNAN
HSTN
NWOR
ORLD

Outline

- *A simple example as motivation*
- *ONOS Architecture*
- *Distributed primitives for managing state*
- *APIs and a template for implementation*
- *Intent Framework*
- *Live Demo*
- **Q/A**