POLITECNICO DI MILANO

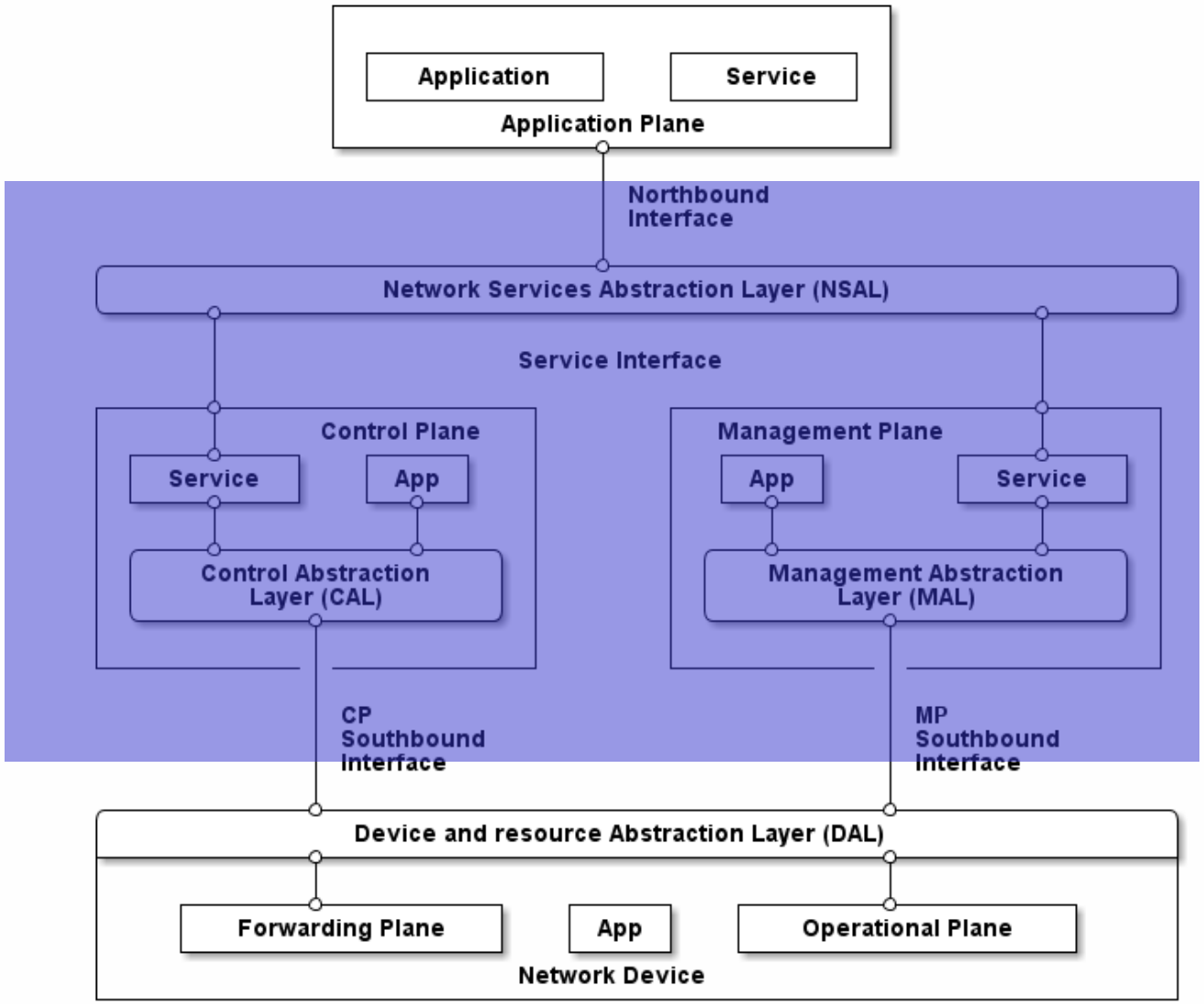




ONOS

The Open Network Operating System

ONOS in the SDN landscape



ONOS

Network Abstractions

Data Plane

- packet forwarding (plus buffeering, scheduling, …)
- local information and local decisions
- abstracted with tables (routing tables, packet classification, …)

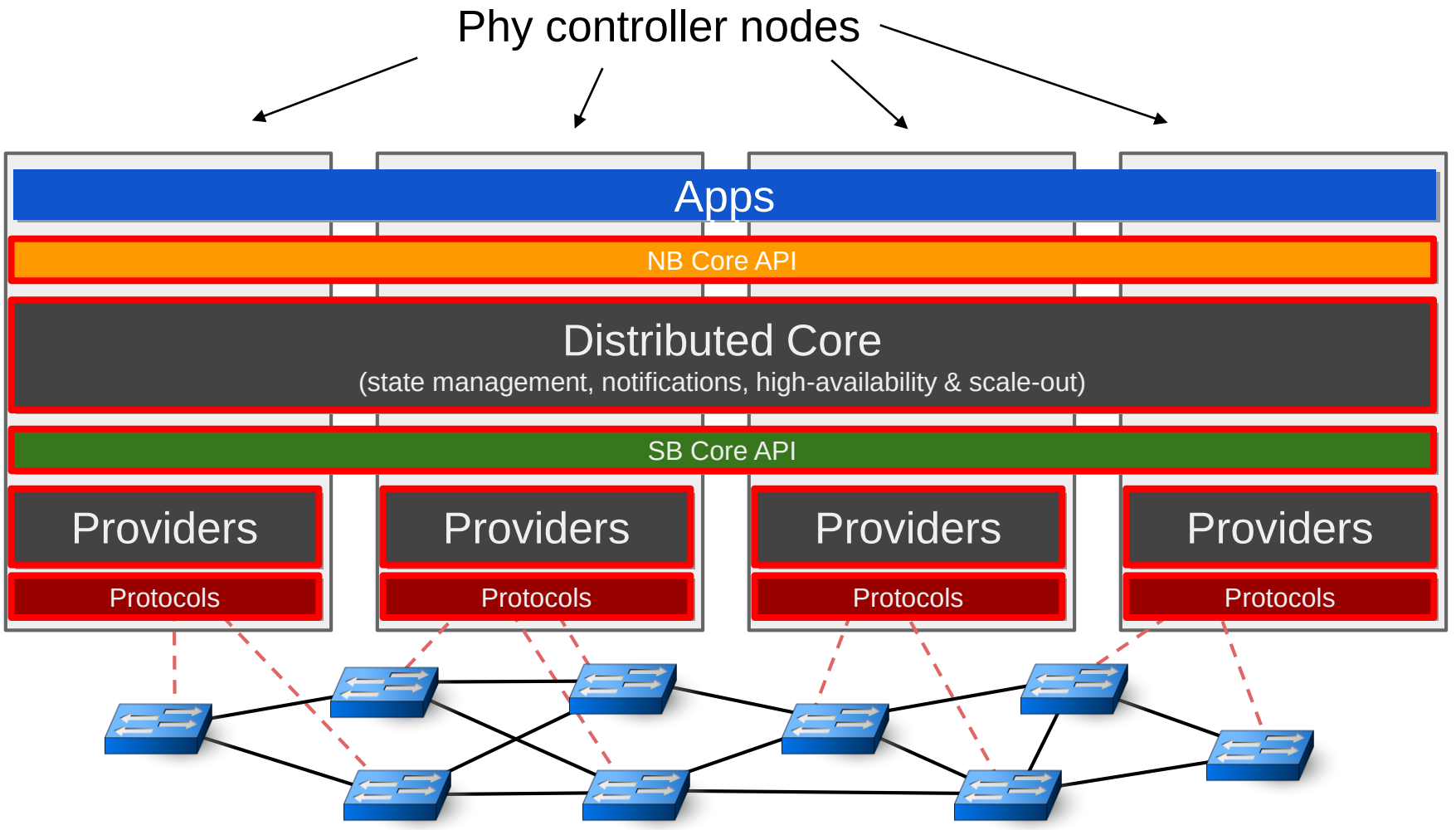
Control Plane & Management Plane

- compute the configuration of devices (routing, traffic engineering)
- global information and global decisions
- abstracted with paths, graphs, …

ONOS design goals

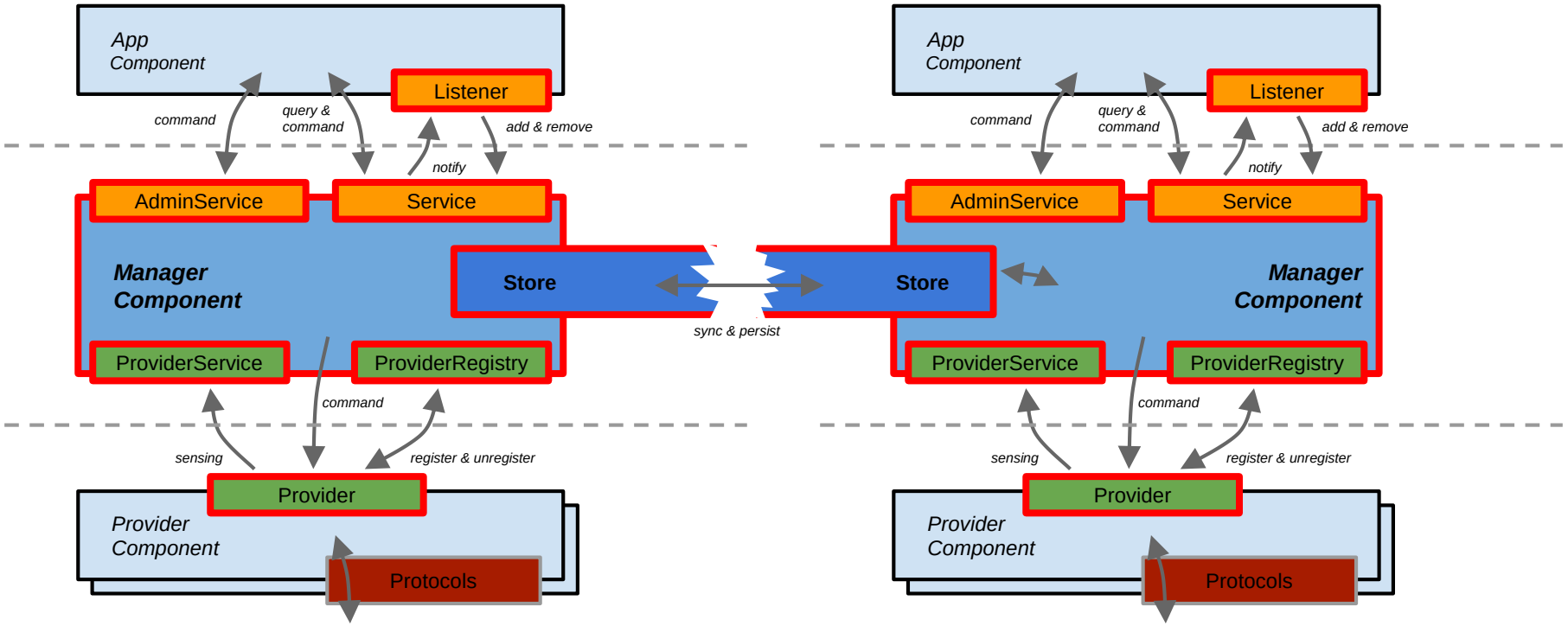
- **Being the SDN network OS**
  - provide strong abstractions and simplicity (to make it easy to create apps and service to control a network)
  - manage multiple protocols and devices
  - separation of concerns for easy customization
- **Scalability, high availability, and performance**
  - Distributed architecture
  - Performance targets:
    - 0.5M - 1M path setup/s
    - 3M – 6M network state ops/s
    - 0.5TB – 1TB network state data
- **Focus on service provider networks, but not limited to it**

ONOS Distributed Architecture



Resistant to failures of physical nodes

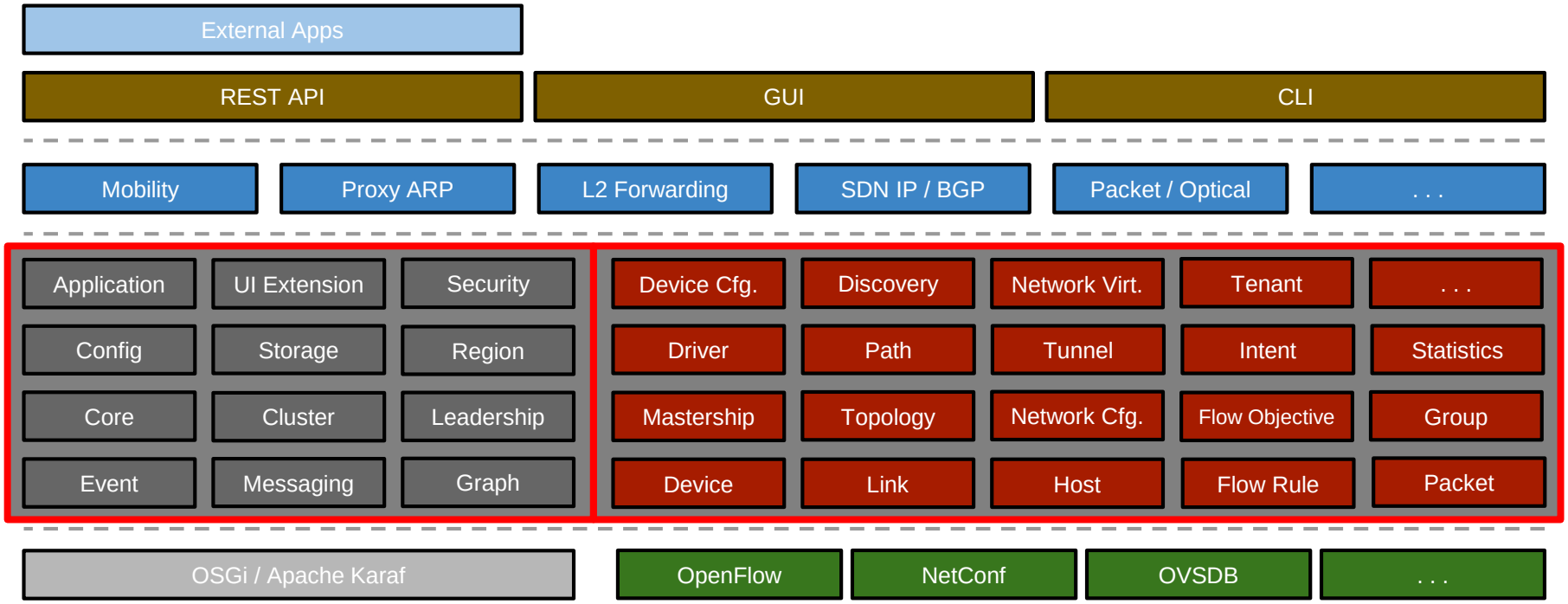
ONOS Core Subsystem Structure



ONOS Applications

- **Application as a Component**
  - no API, self-contained
  - interacts only with the network
  - e.g. reactive forwarding (similar to learning switch)
- **Application with Service Interface**
  - has API (command line, REST, GUI)
  - interacts with external entities (e.g. users)
- **External Applications**
  - using REST API
- Applications may have their own state (may use ONOS store)

ONOS Core Subsystems



Distributed Architecture

Distributed → Set up as a cluster of instances

Symmetric → Each instance runs identical software and configuration

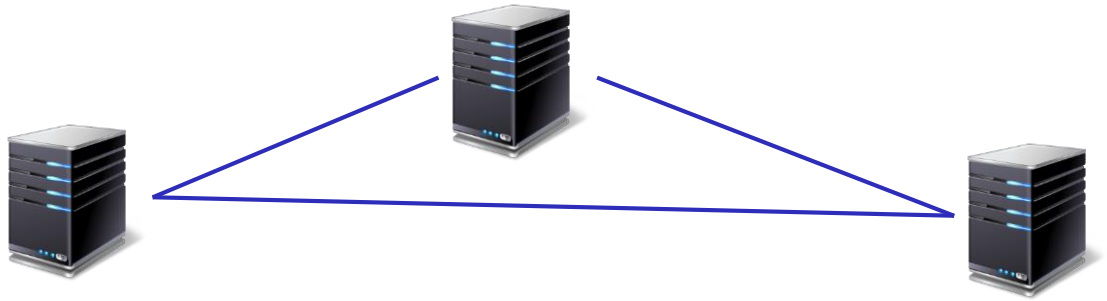
Fault-tolerant → Cluster remains operational in the face of node failures

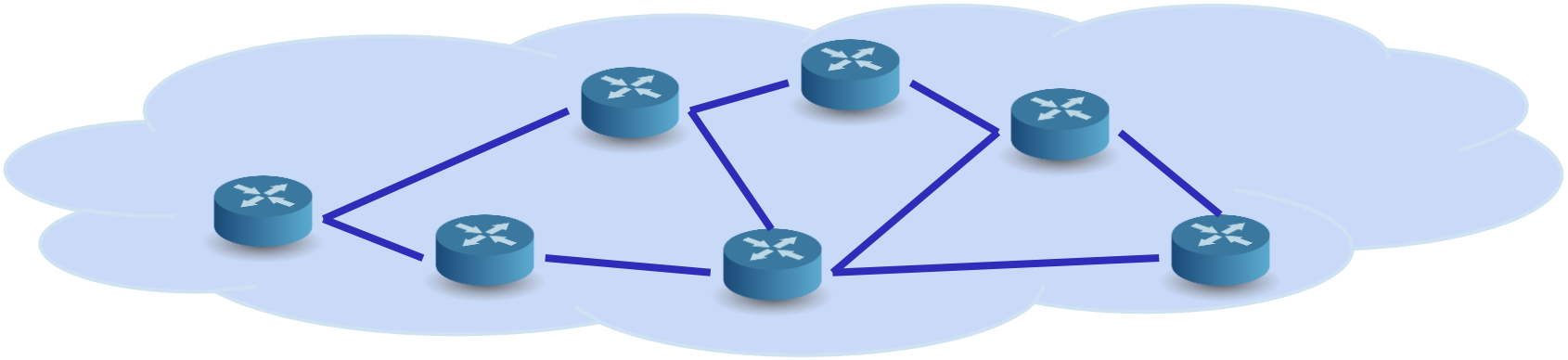
Location Transparent → A client can interact with any instance. The cluster presents the abstraction of a single logical instance

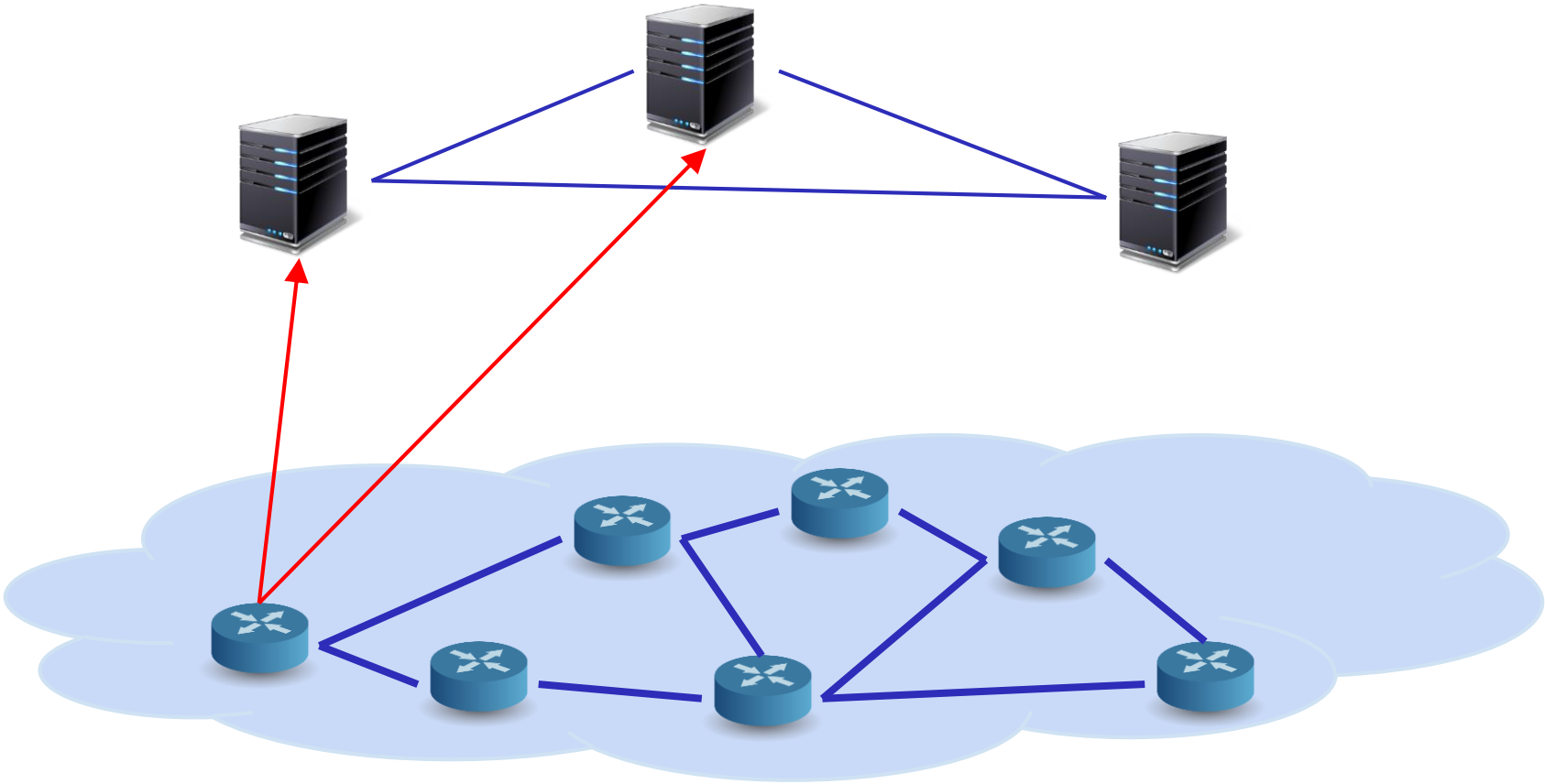
Dynamic → The cluster can be scaled up/down to meet usage demands

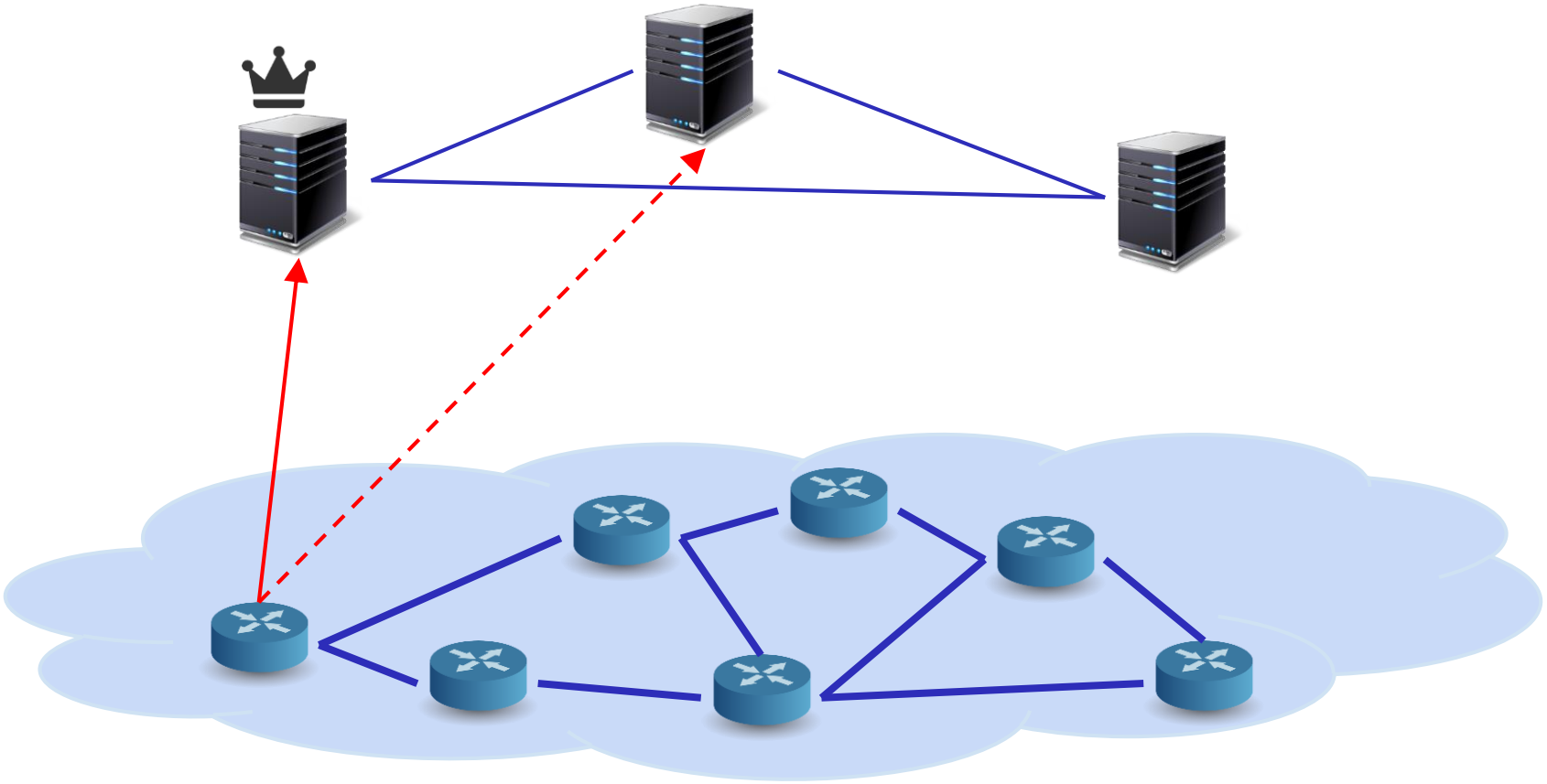
Raft consensus → Replicated State Machine

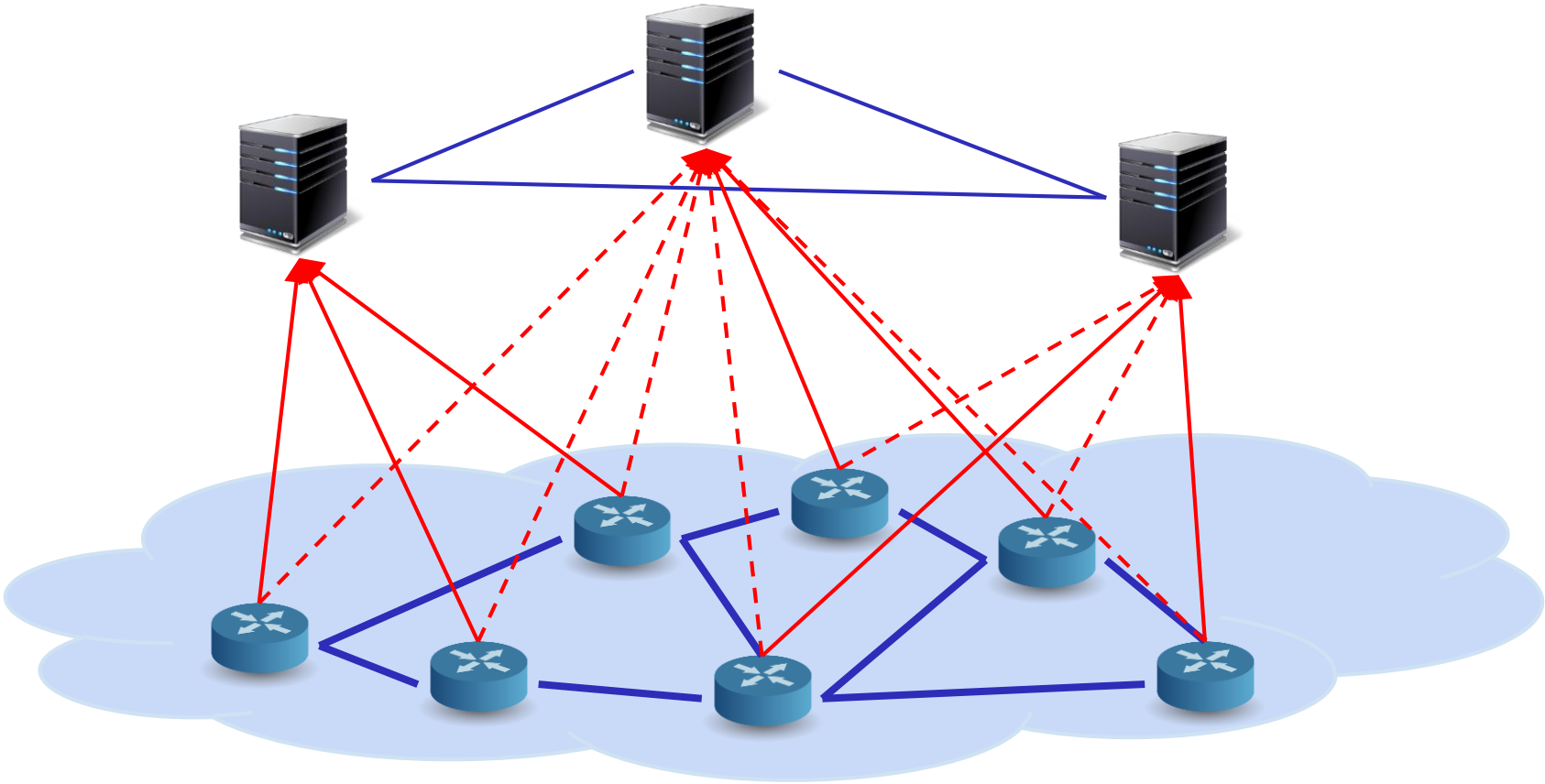
Distributed Control Plane

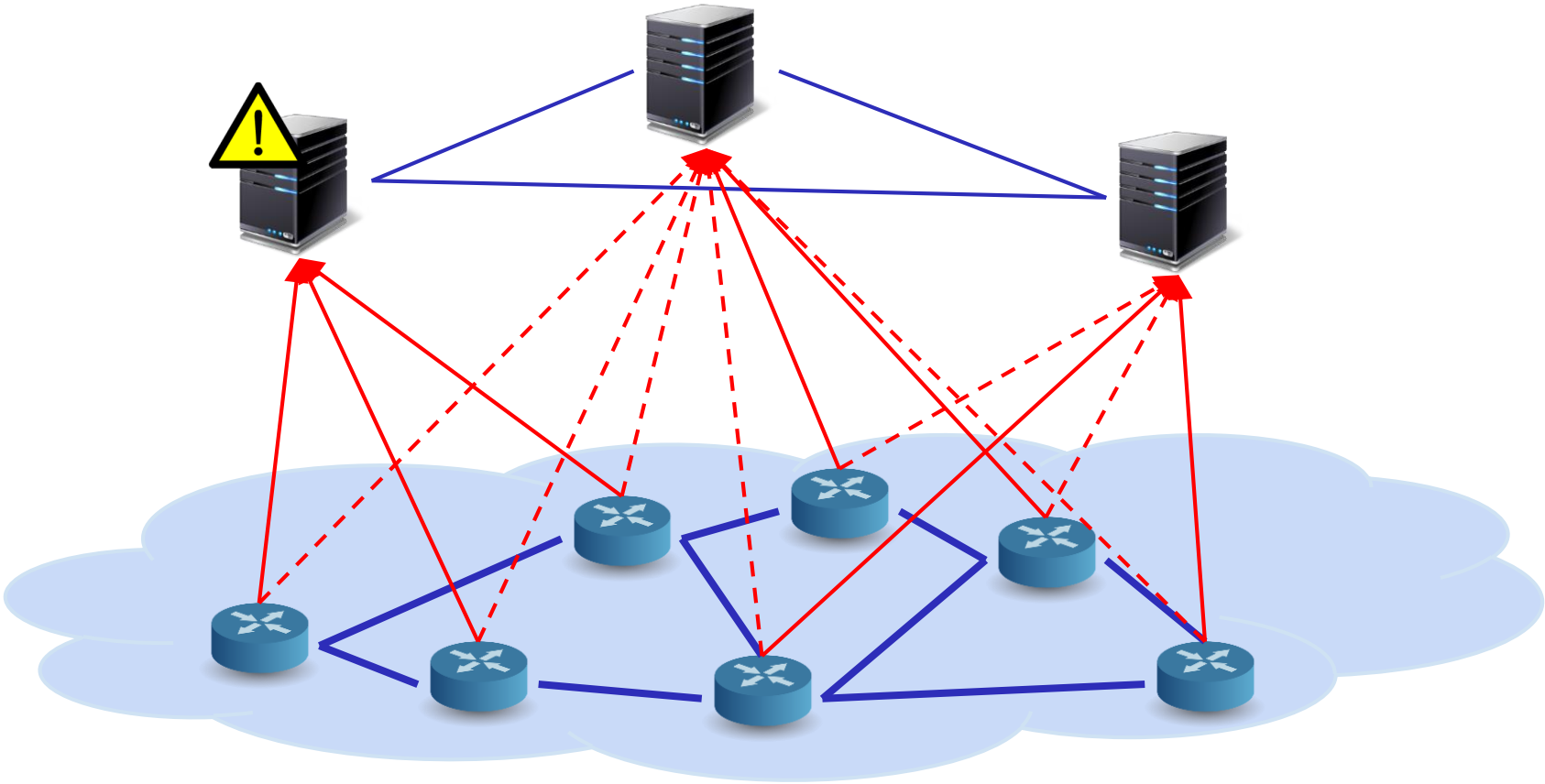


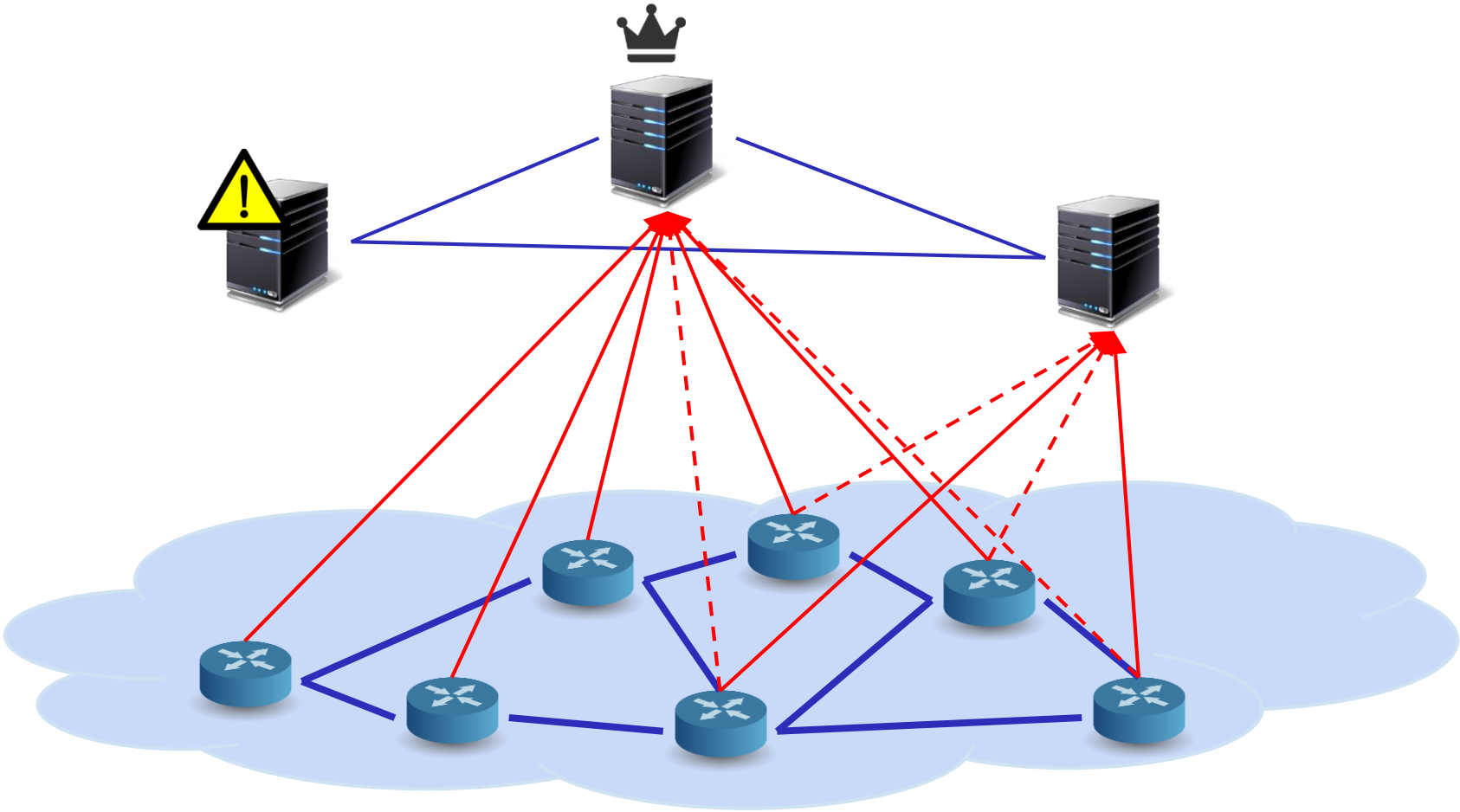












Northbound Abstractions

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 version and vendor-independent manner

Intent

- Network-centric abstraction for programming data-plane in topology-independent manner

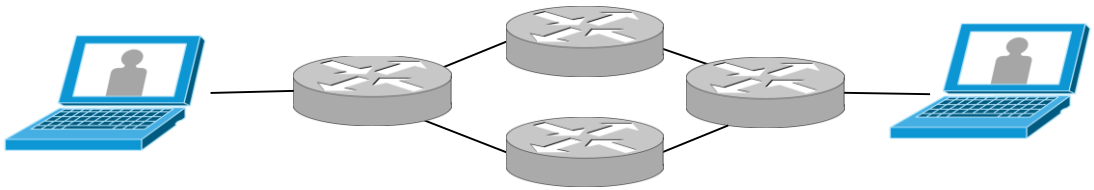
Building Network Applications

Objective: Connect Host 1 and Host 2

Building Network Applications

Objective: Connect Host 1 and Host 2

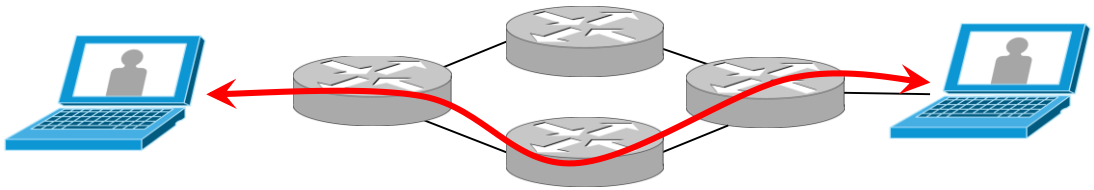
1. Read/discover the topology



Building Network Applications

Objective: Connect Host 1 and Host 2

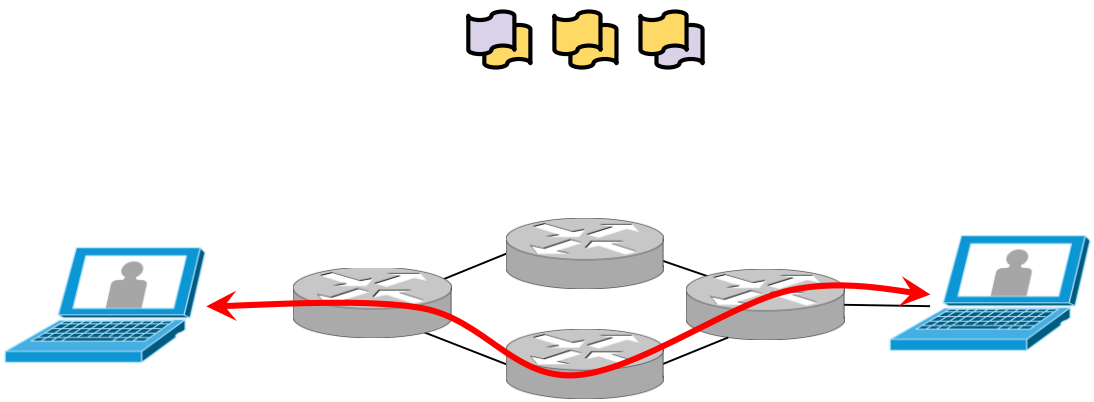
1. Read/discover the topology
2. Compute a path



Building Network Applications

Objective: Connect Host 1 and Host 2

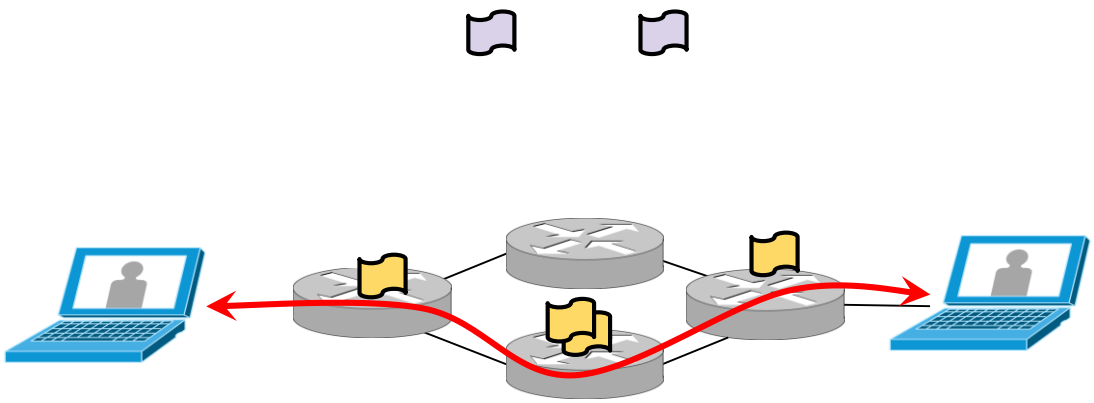
1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device



Building Network Applications

Objective: Connect Host 1 and Host 2

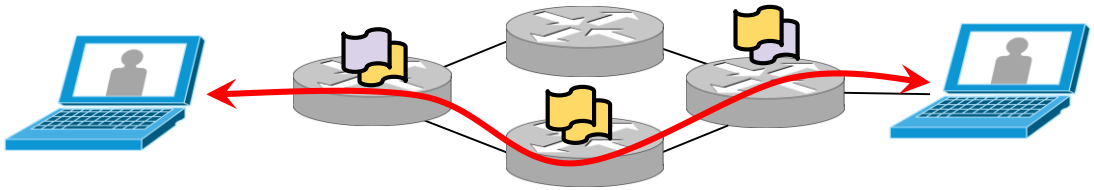
1. Read/discover the topology
2. Compute a path
3. Build flow objectives for each device
4. Install rules in consistent way



Building Network Applications

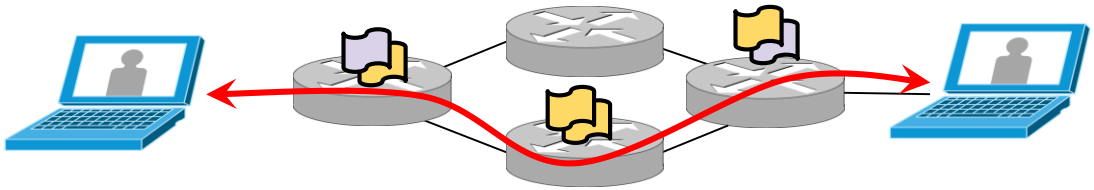
Objective: Connect Host 1 and Host 2

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

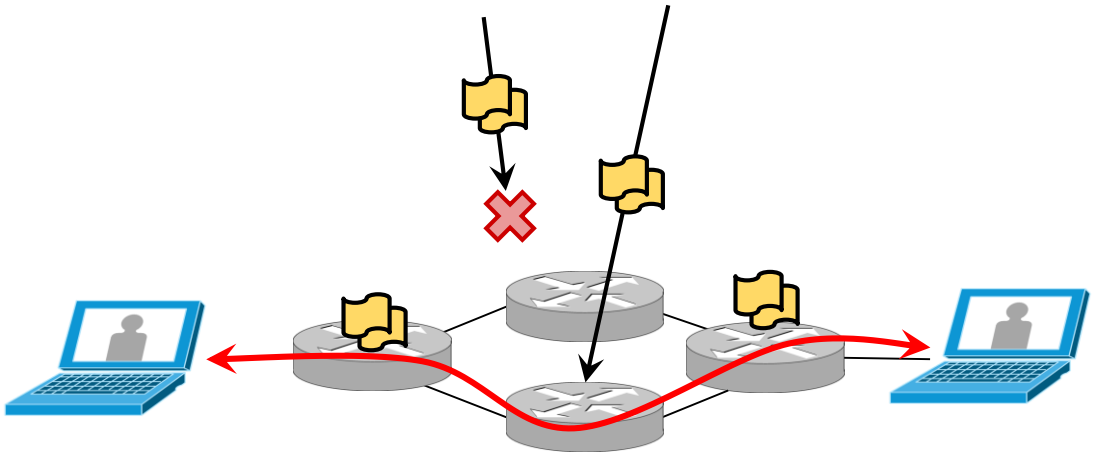
What can go wrong?



Building Network Applications

What can go wrong?

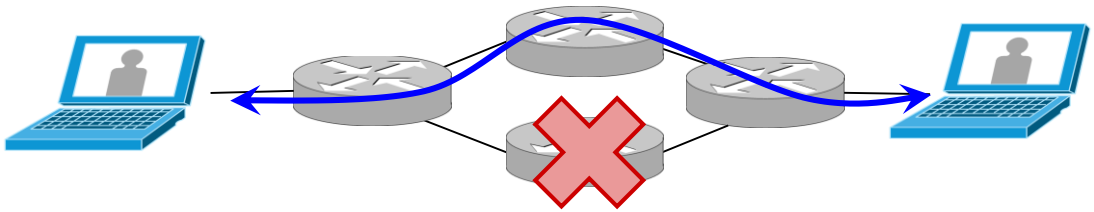
- Missing / rejected / dropped rules
  - Monitor devices connections
  - Send barriers between rule updates
    - Poll flow state



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

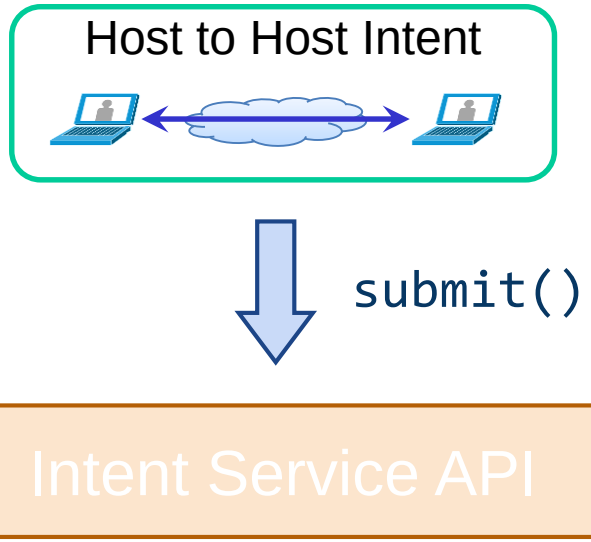
Abstracts unnecessary network complexity from applications

Maintains requested semantics as network changes

High availability, scalability and high performance

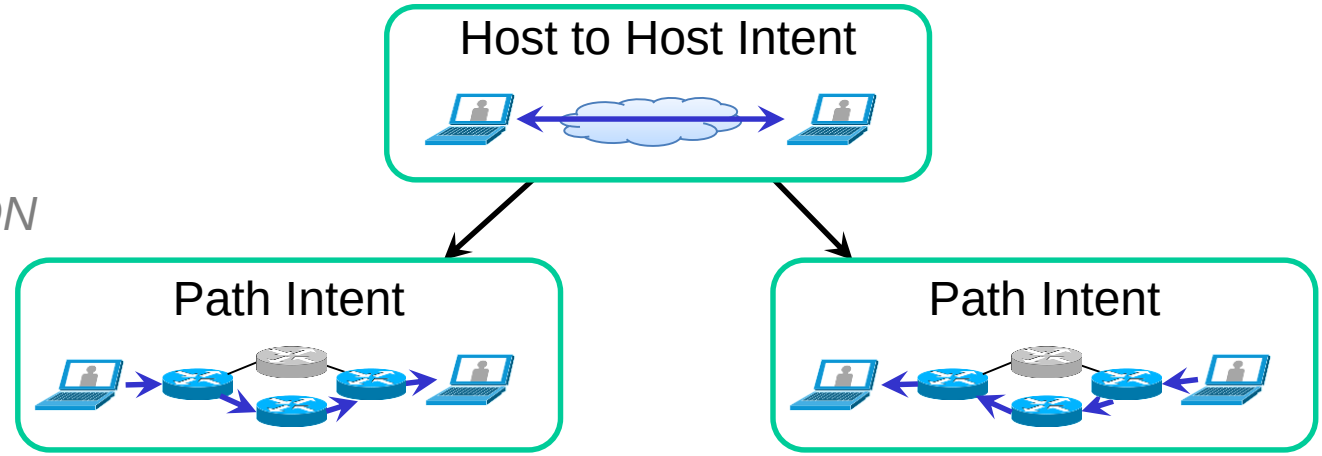
Intent Example

Intent Example

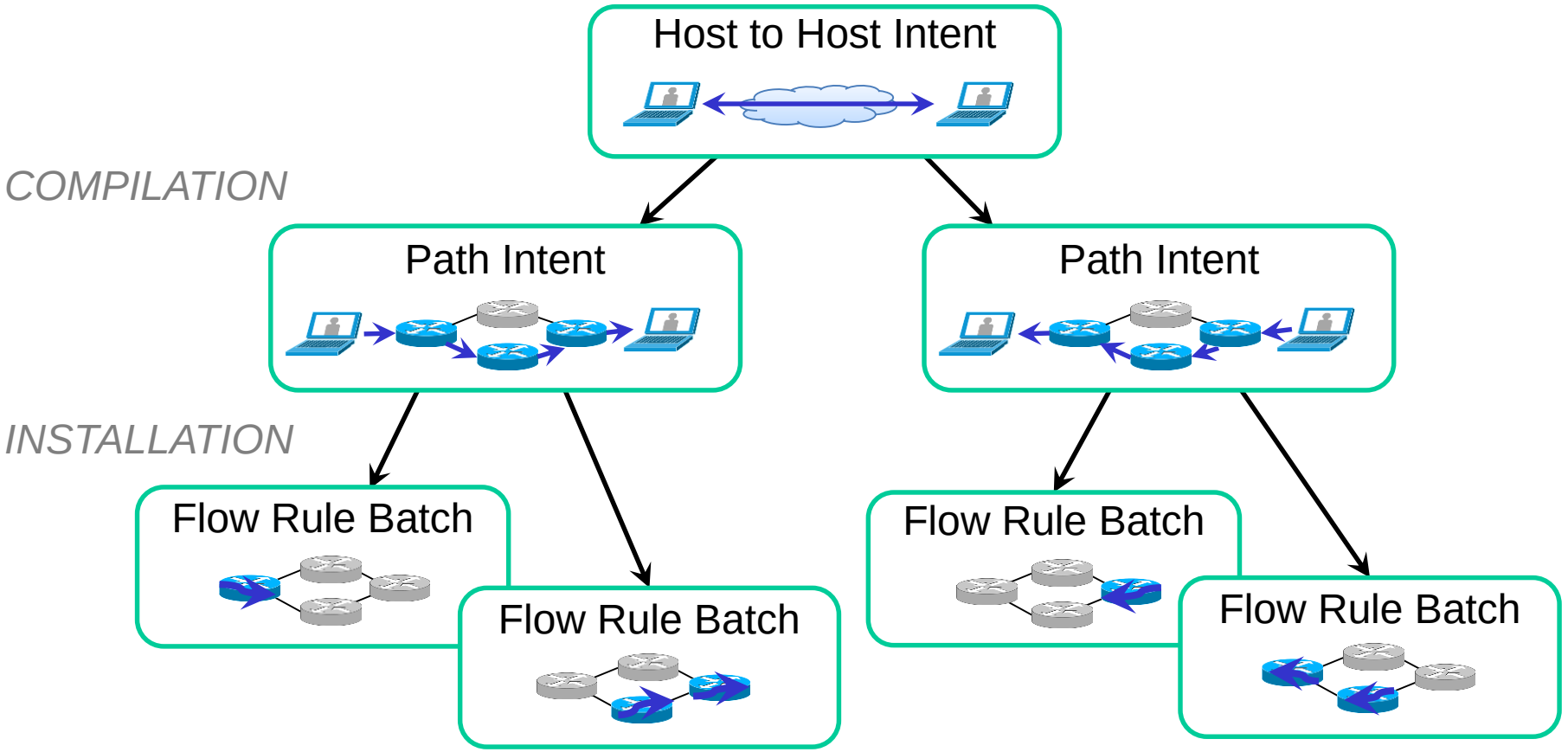


Intent Example

COMPILATION



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”