Advanced Computer Networks

CS740

Software-Defined Network

https://pages.cs.wisc.edu/~mgliu/CS740/F25/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
  - Network virtualization in data center networks (II)

- Today
  - SDN and programmable networks (I)

- Announcements
  - Lab2 due 11/05/2025 11:59 PM
  - Midterm report due 11/04/2025 11:59 PM

Where we are?

Data Center Network

Multiple communication paths exist when accessing and traversing data center networks!

Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

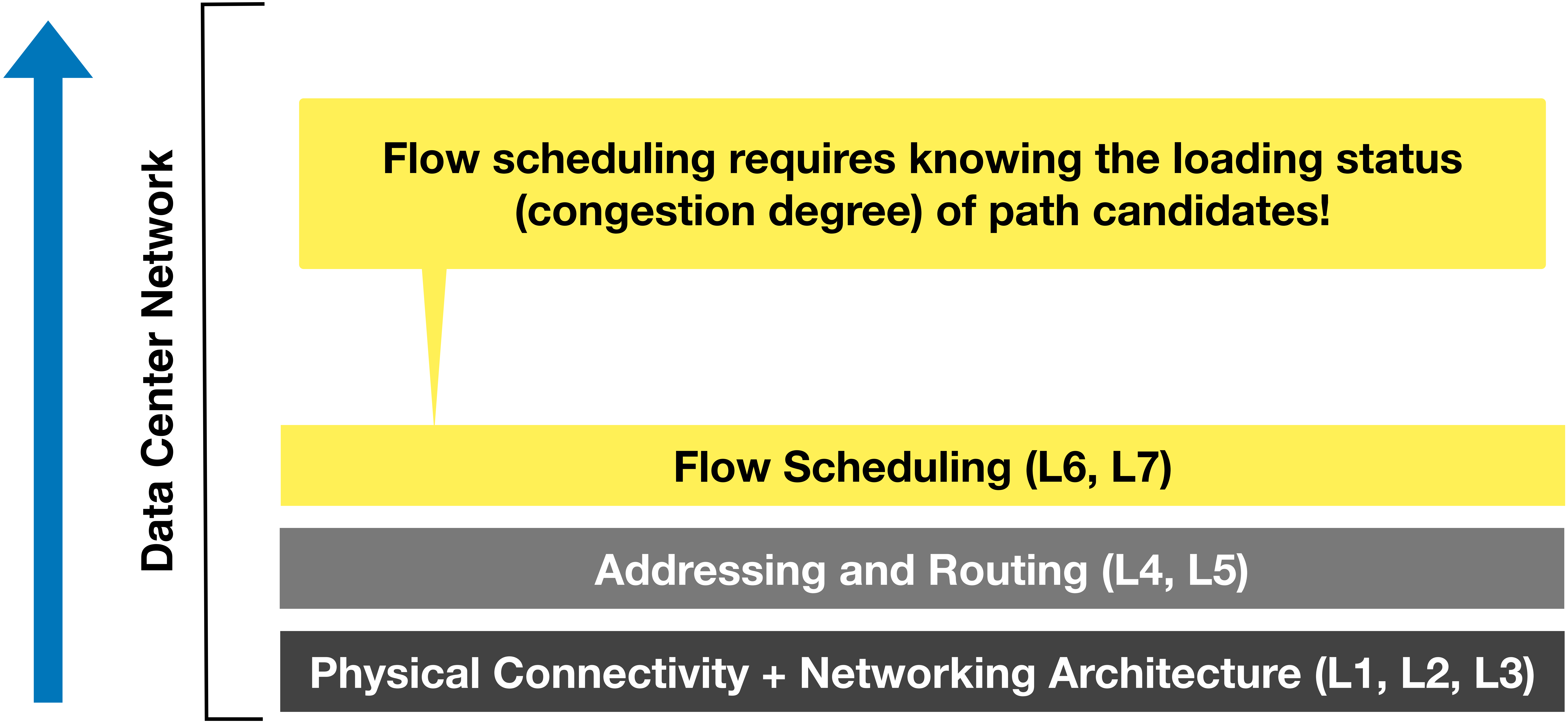
Data Center Network

The forwarding (destination) address and routing table determine how packets are forwarded!

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?


Data Center Network
Flow scheduling requires knowing the loading status (congestion degree) of path candidates!
Flow Scheduling (L6, L7)
Addressing and Routing (L4, L5)
Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

Data Center Network

A performant load-balancer design requires per-packet and per-flow processing at line rate with traffic monitoring.

Load balancing (L8, L9)

Flow Scheduling (L6, L7)

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

Data Center Network

A privileged networking layer stack ensure security isolation and performance isolation.

Network Virtualization (L10, L11)

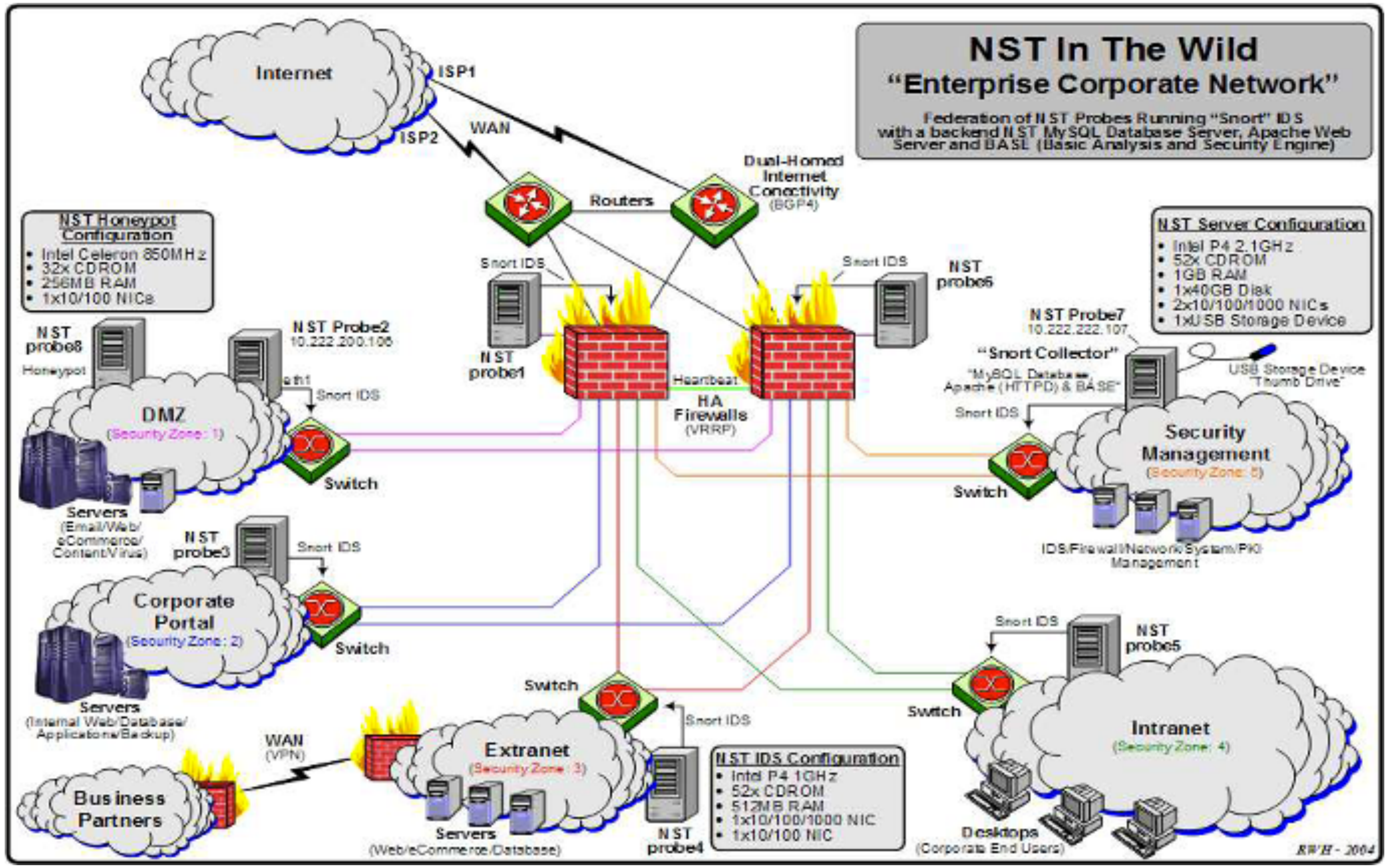
Load balancing (L8, L9)

Flow Scheduling (L6, L7)

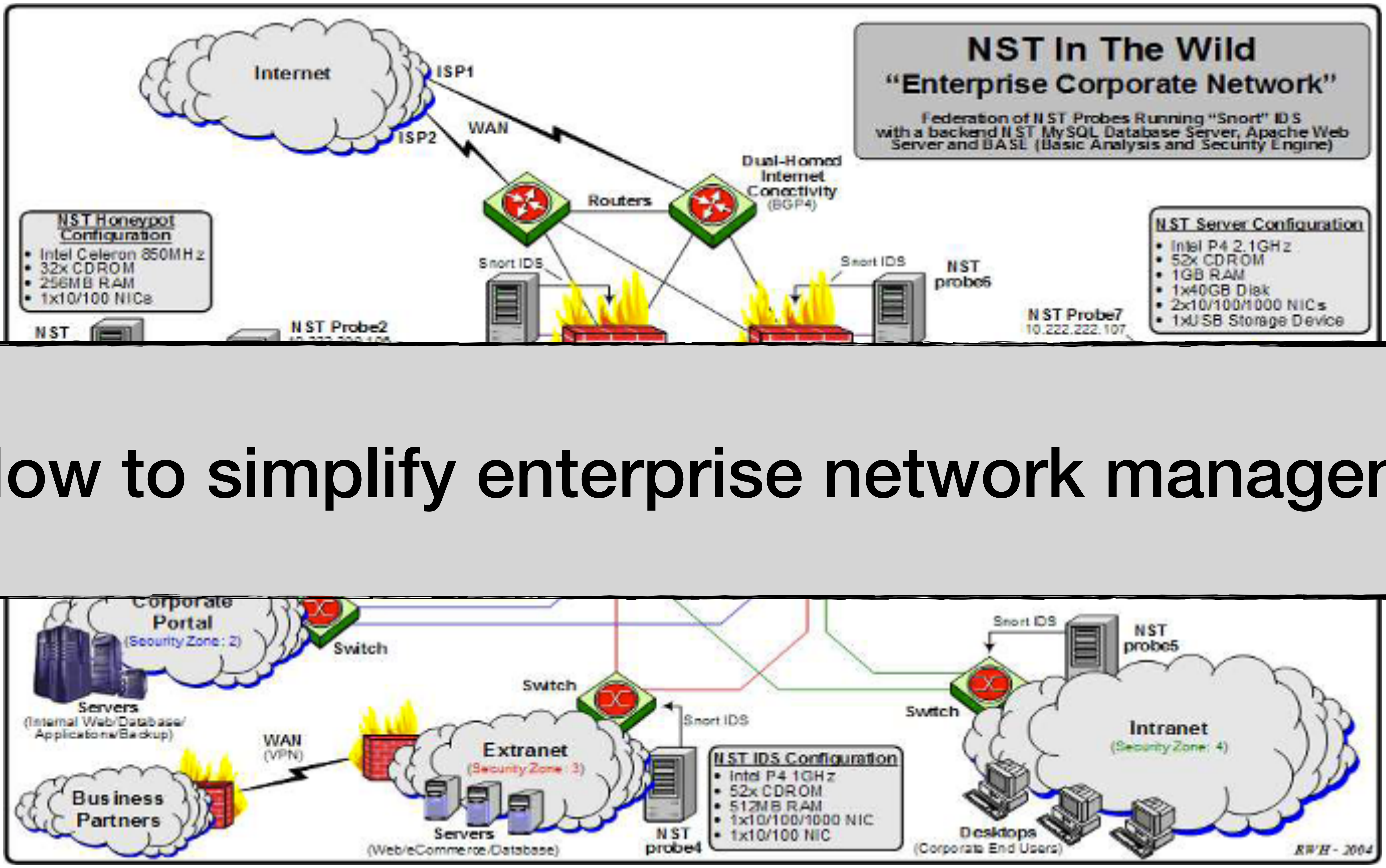
Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

Messy Enterprise Networks



Messy Enterprise Networks



Problem: How to simplify enterprise network management?

Key idea: a centralized controller

Start From the Scratch

A Central Controller

Controller Functionality:

-

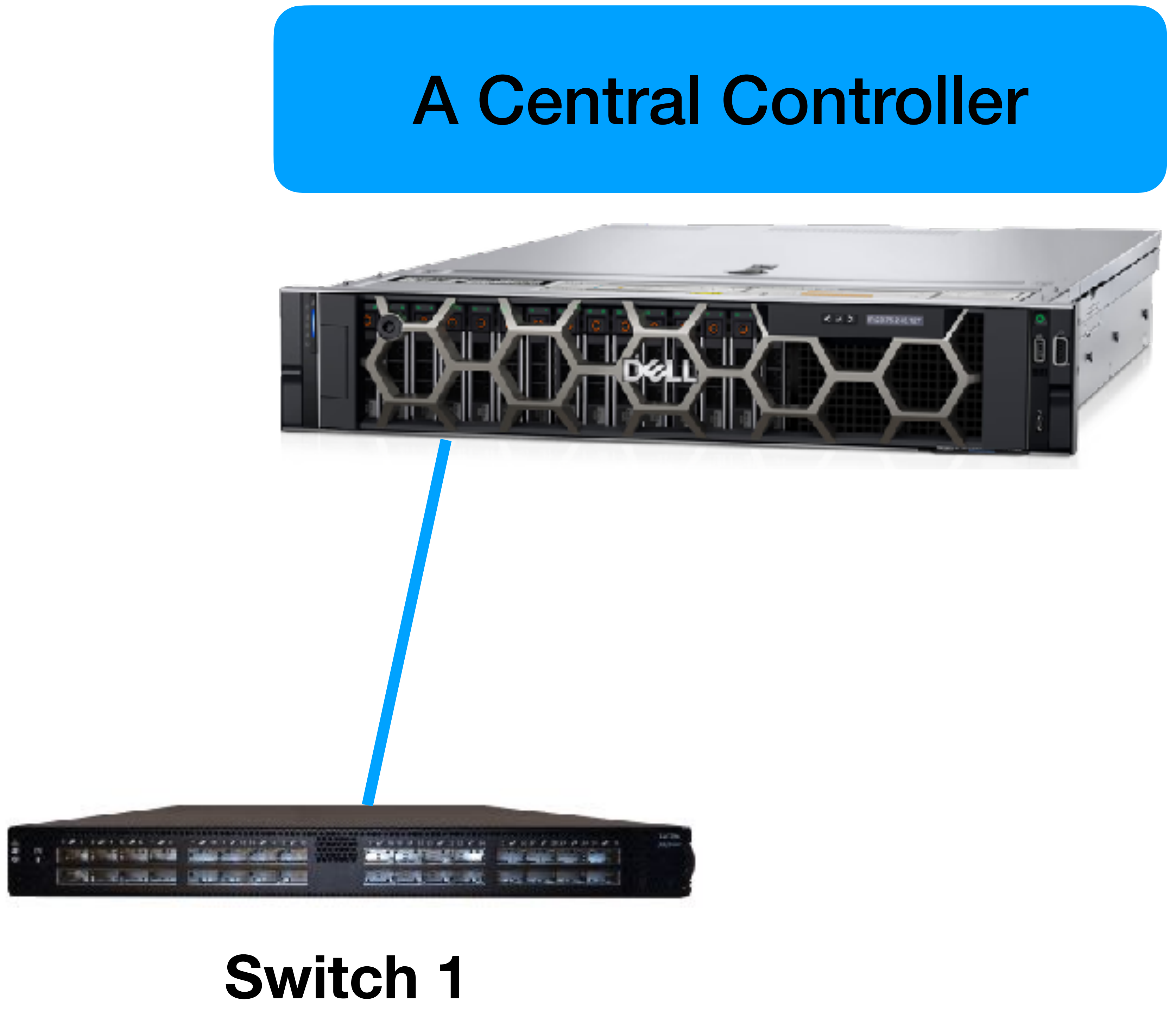
#1: Add a Switch

A Central Controller

Switch 1

Controller Functionality:

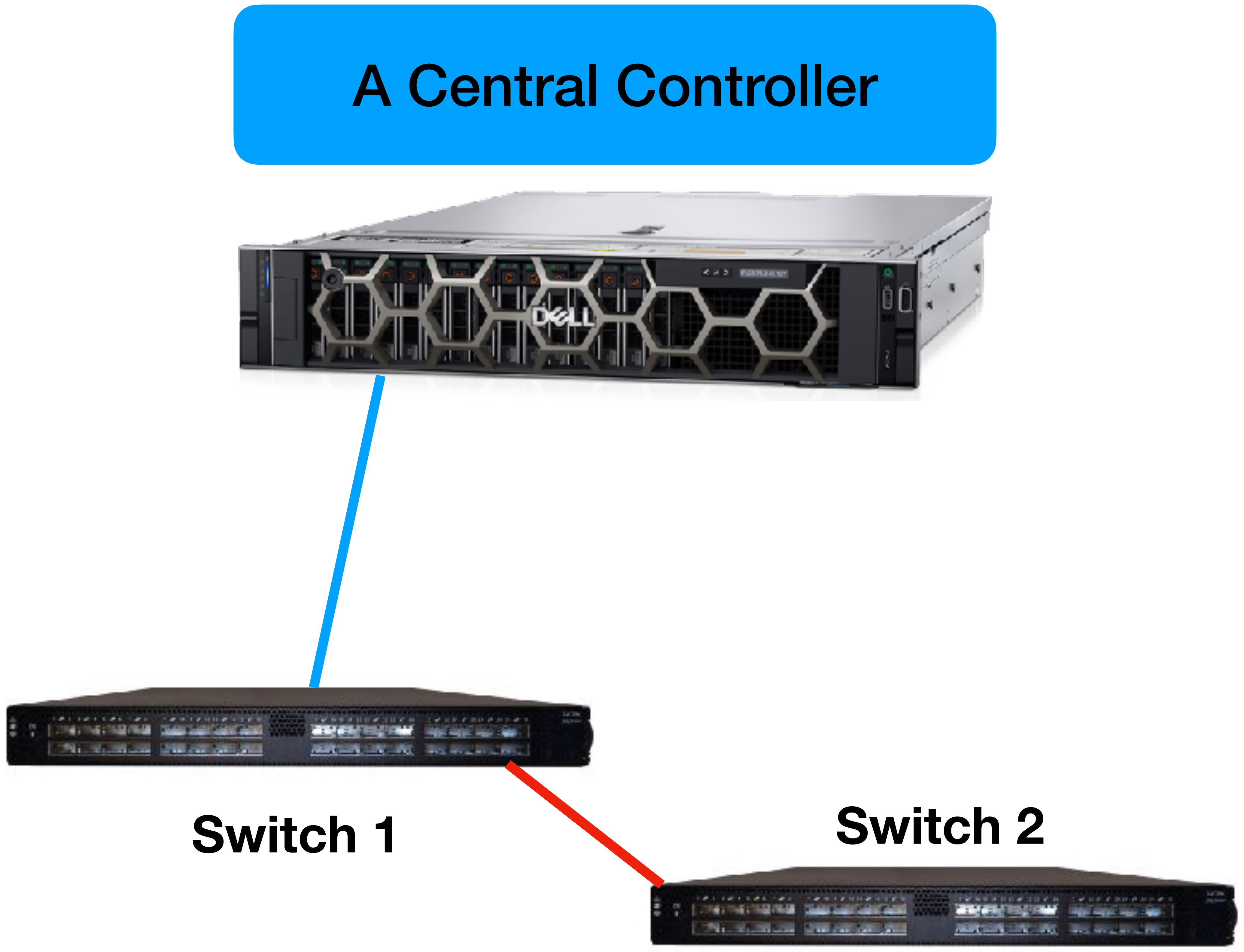
#1: Add a Switch



Controller Functionality:

- Register devices and employ a side network

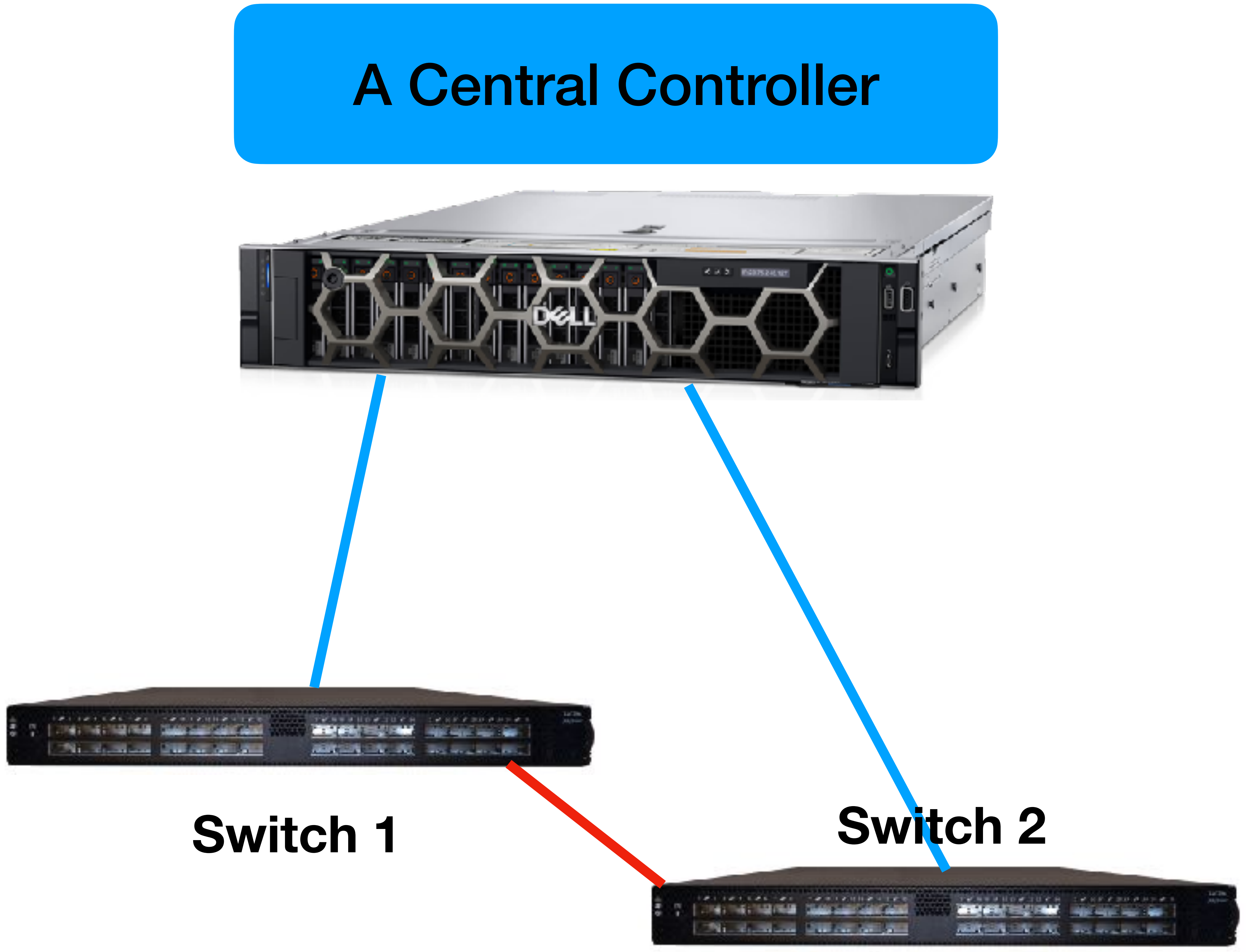
#1: Add a Switch



Controller Functionality:

- Register devices and employ a side network

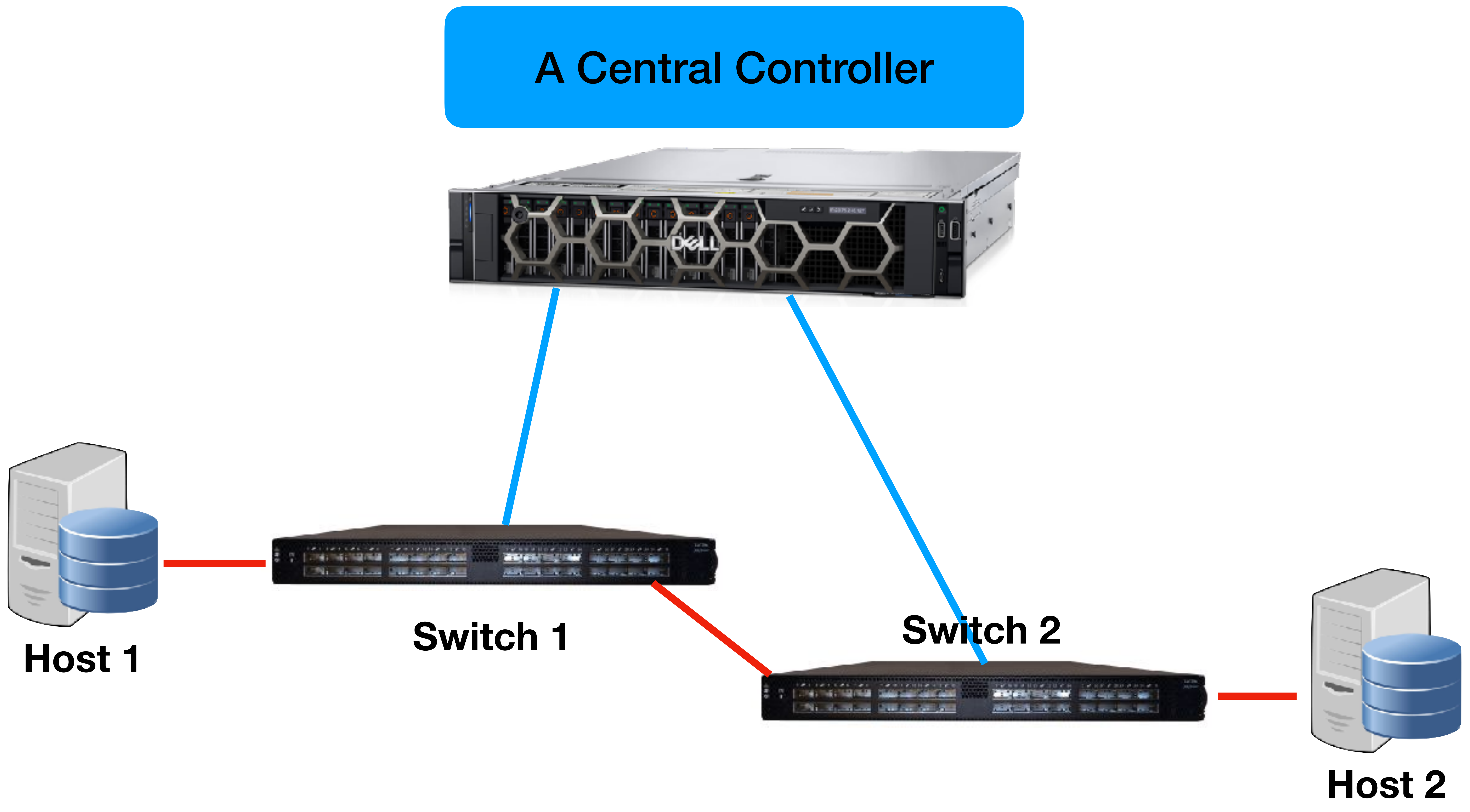
#1: Add a Switch



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology

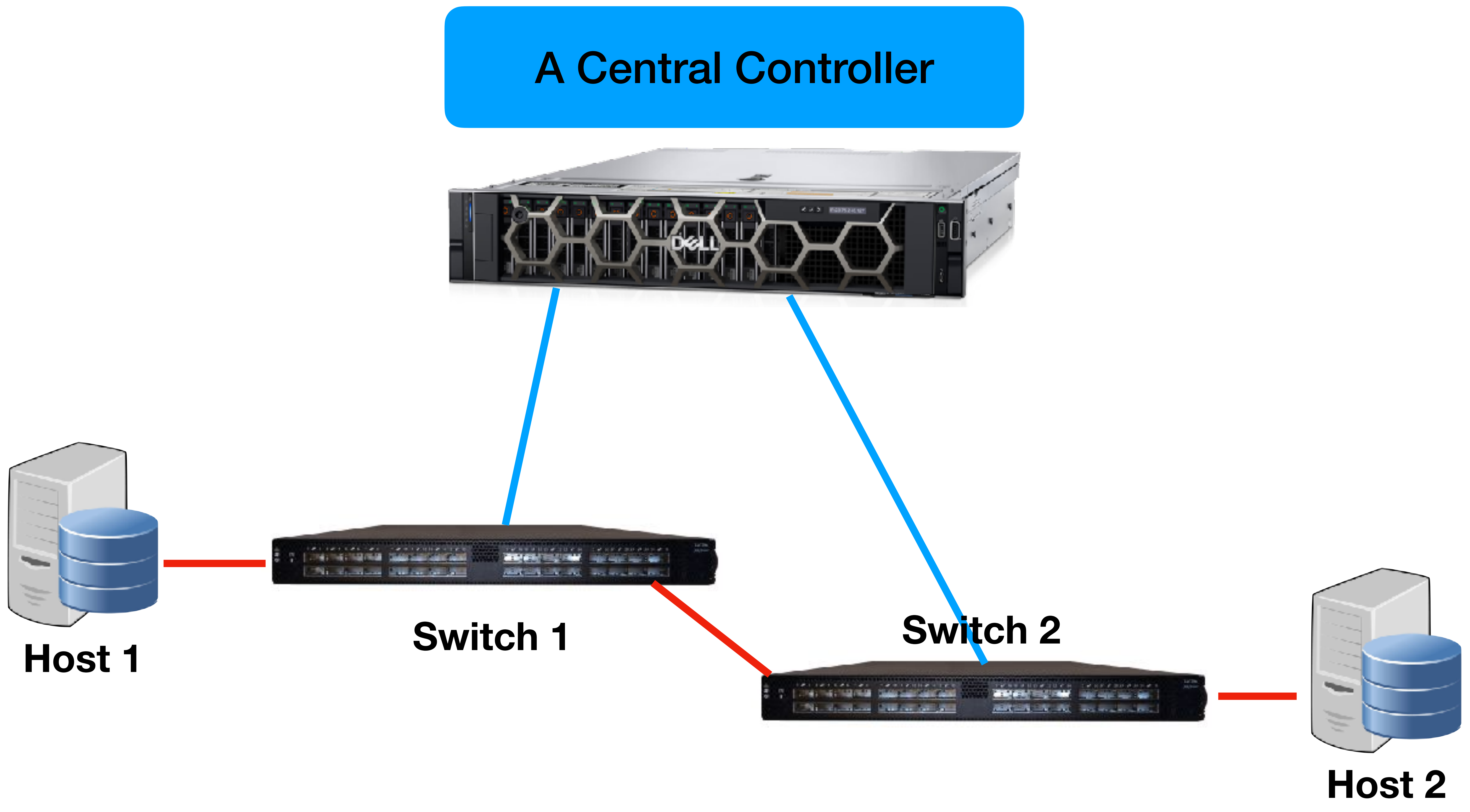
#2: Add a host machine



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology

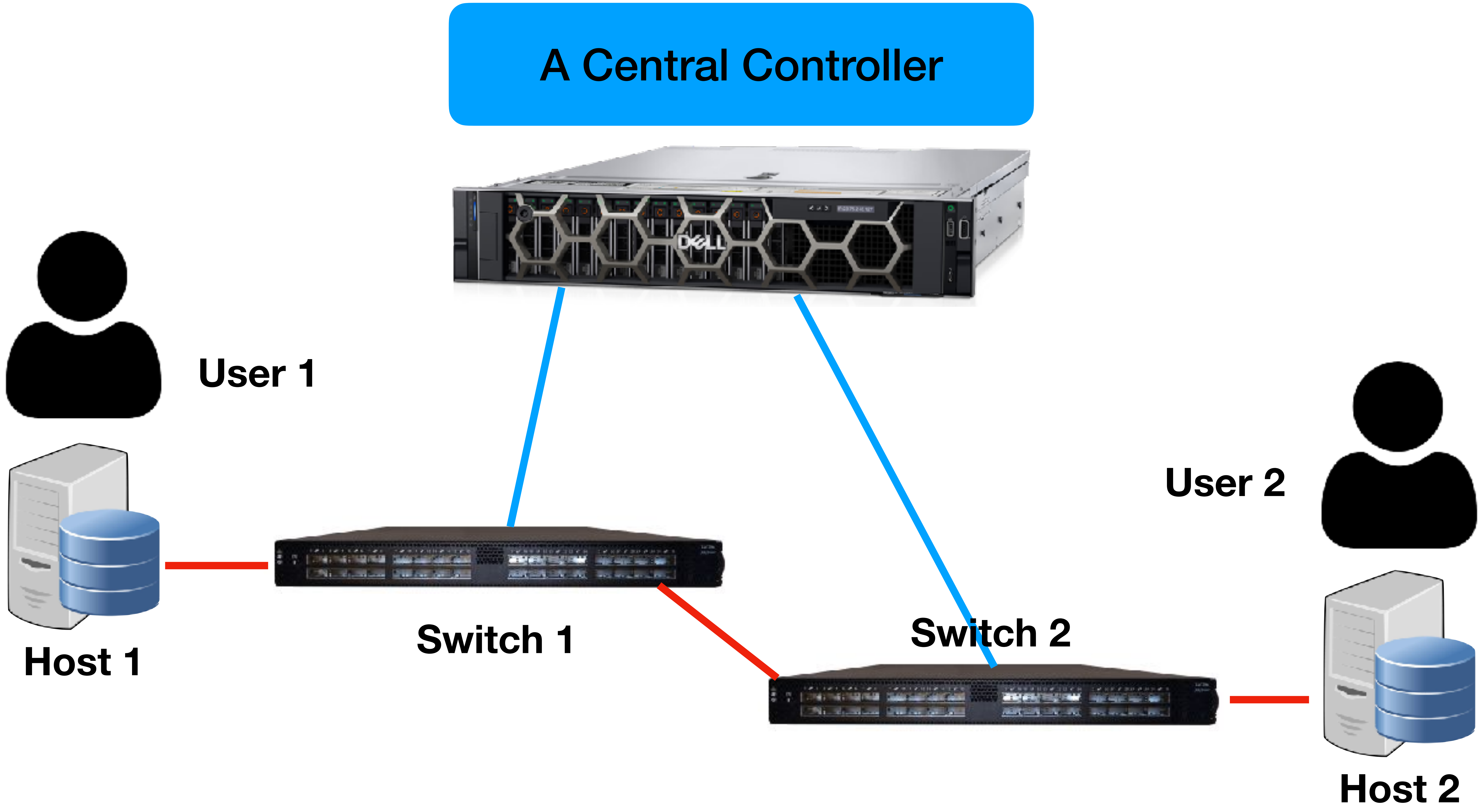
#2: Add a host machine



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines

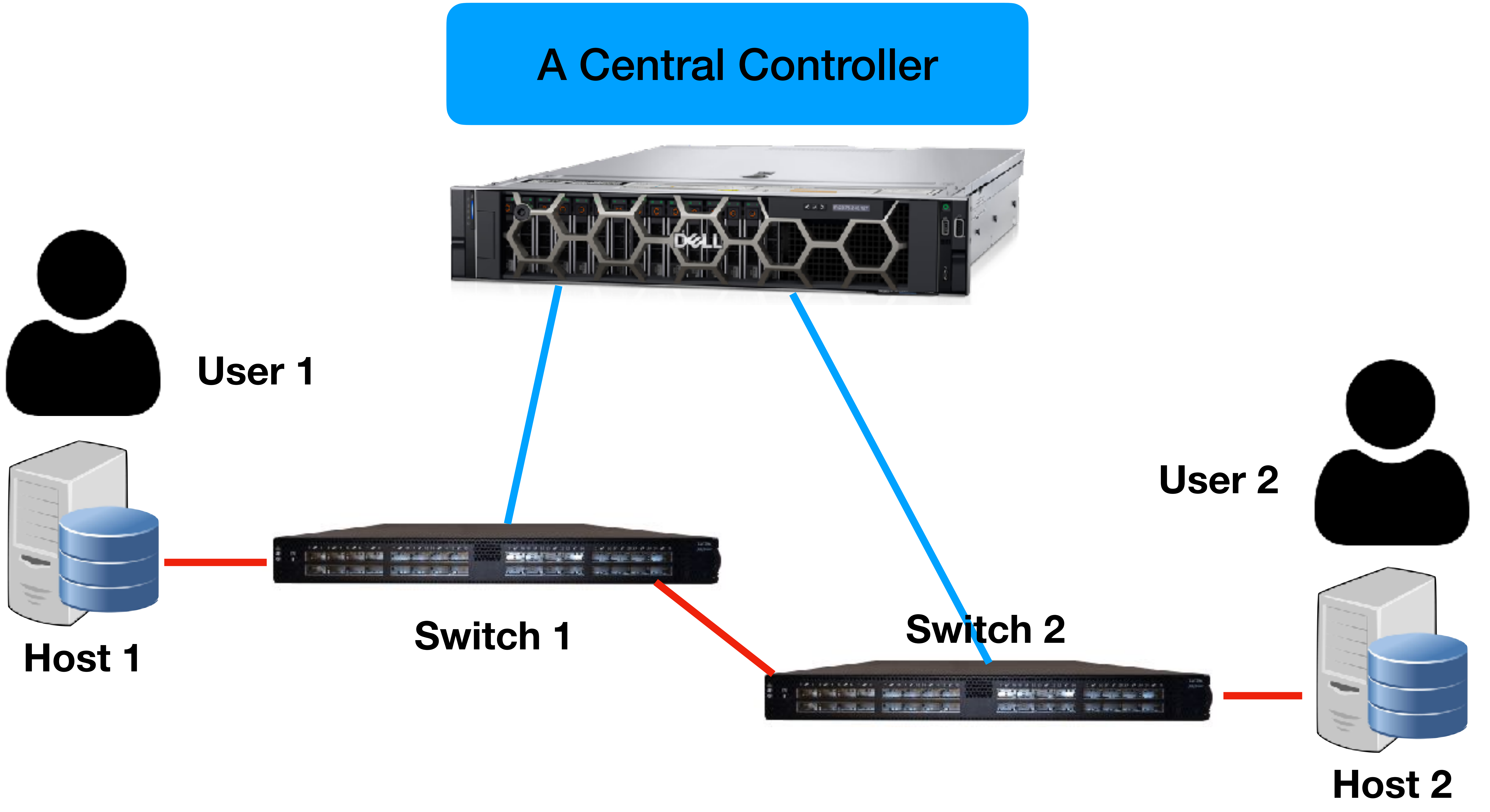
#3: Add a user



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users

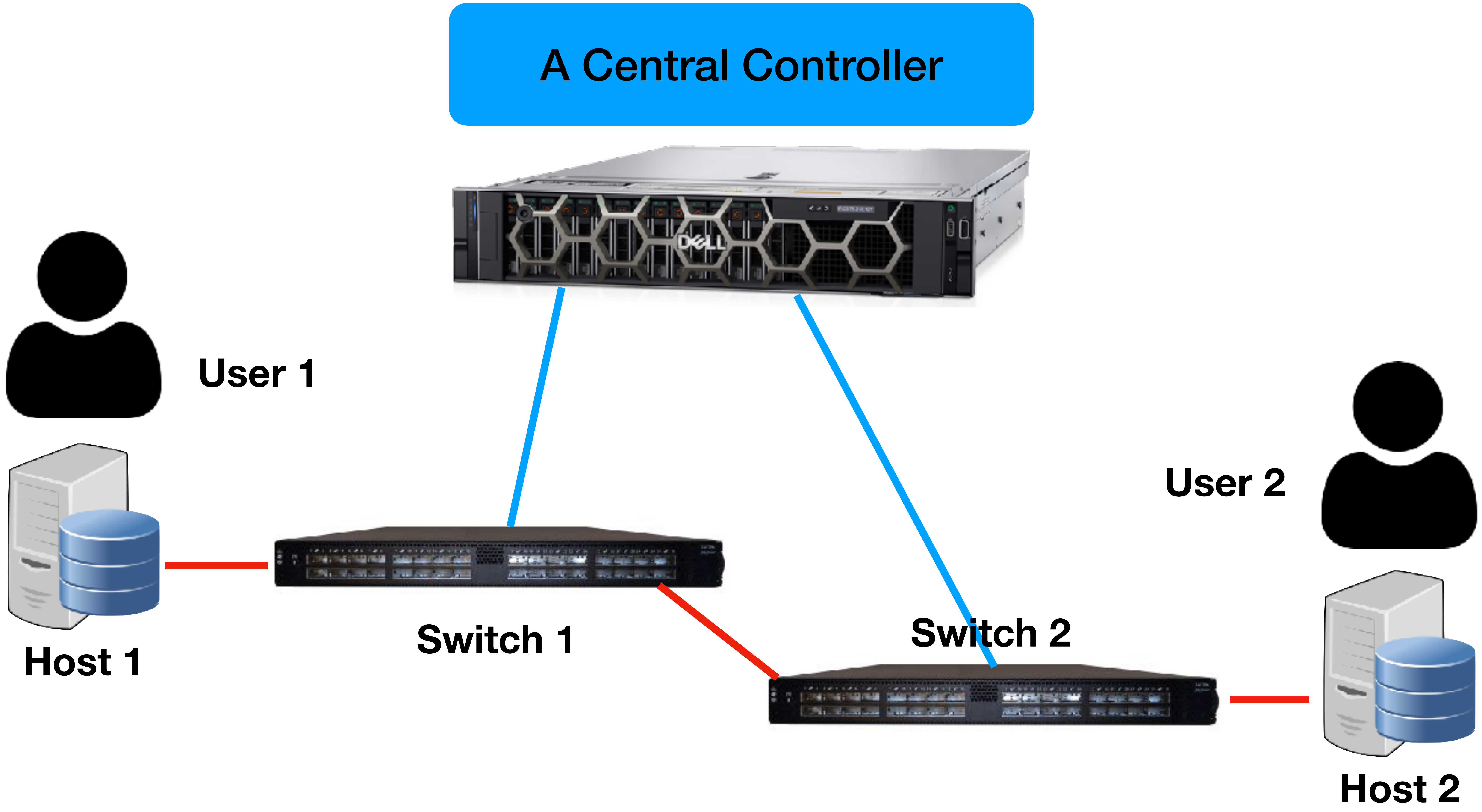
#4: Prevent malicious hosts/users



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication

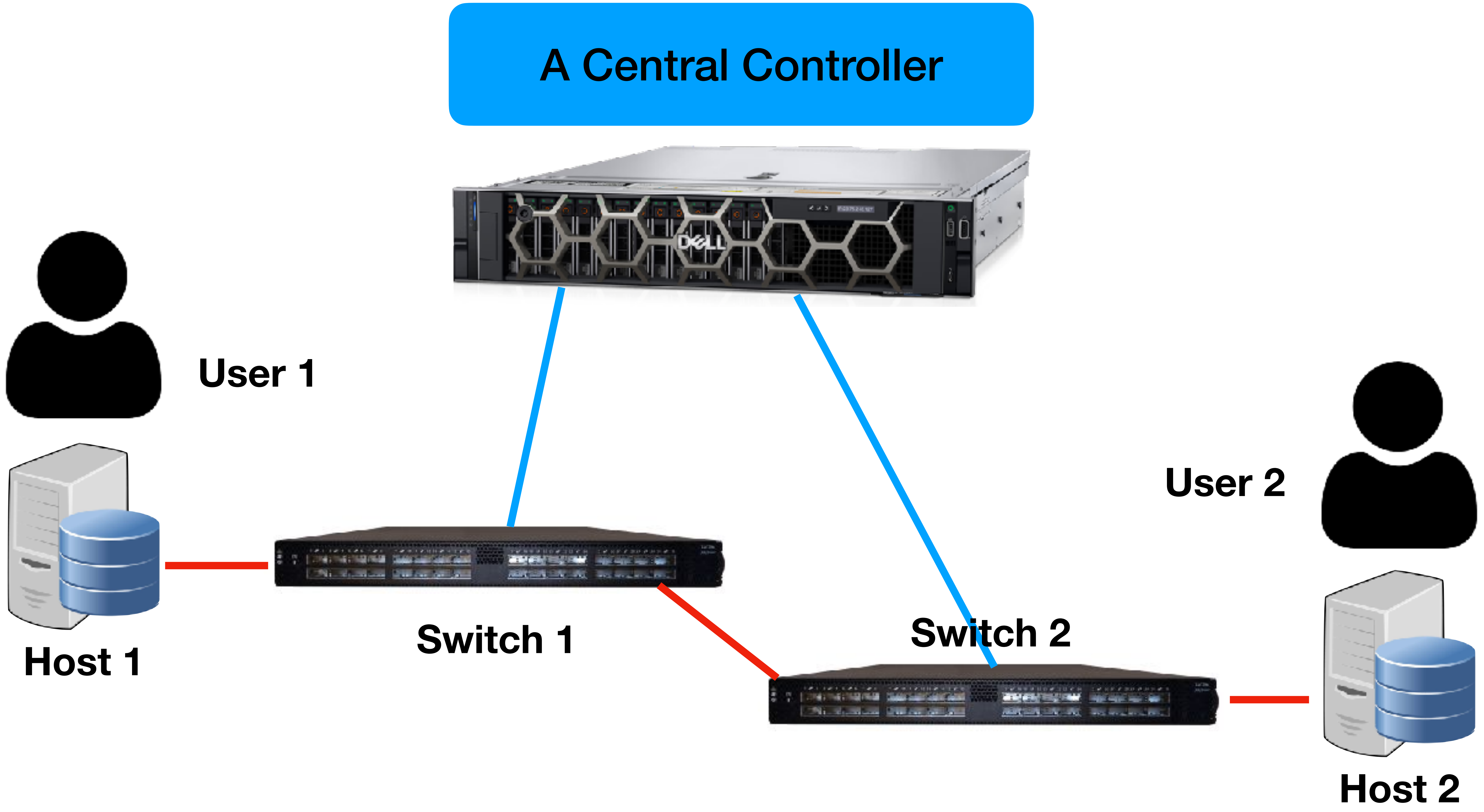
#5: Request an IP Address



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication

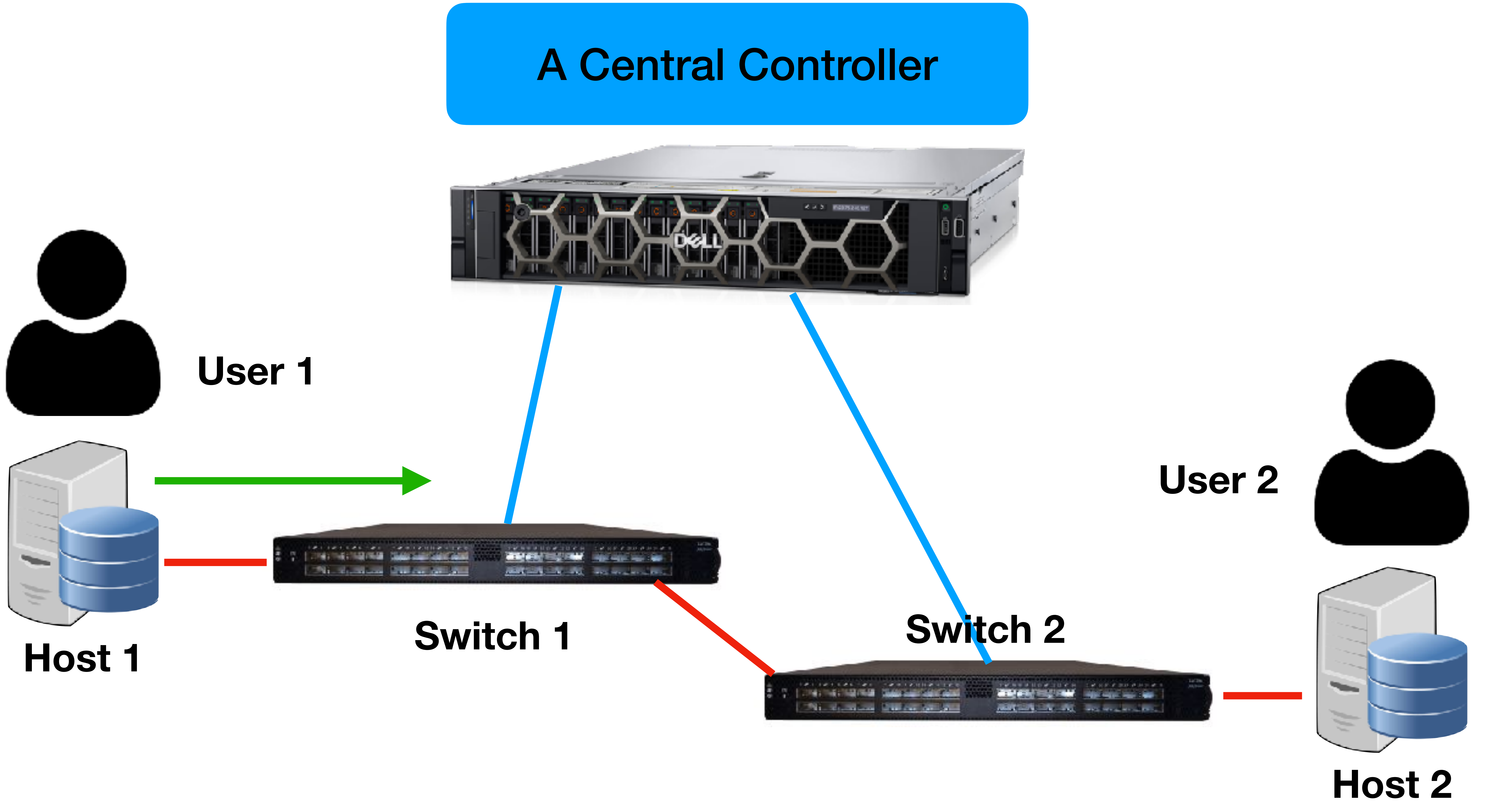
#5: Request an IP Address



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings

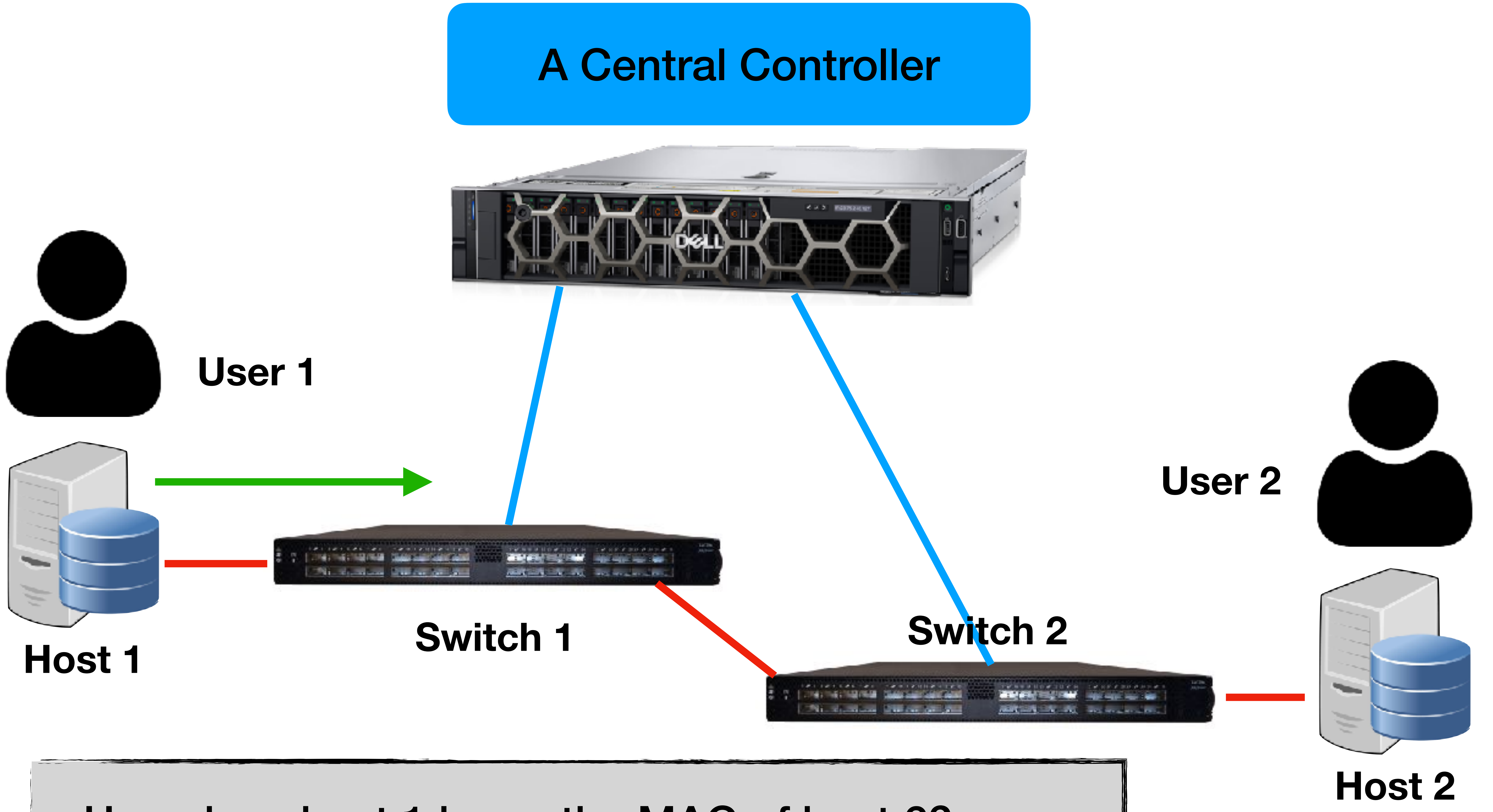
#6: User 1 —> User 2



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings

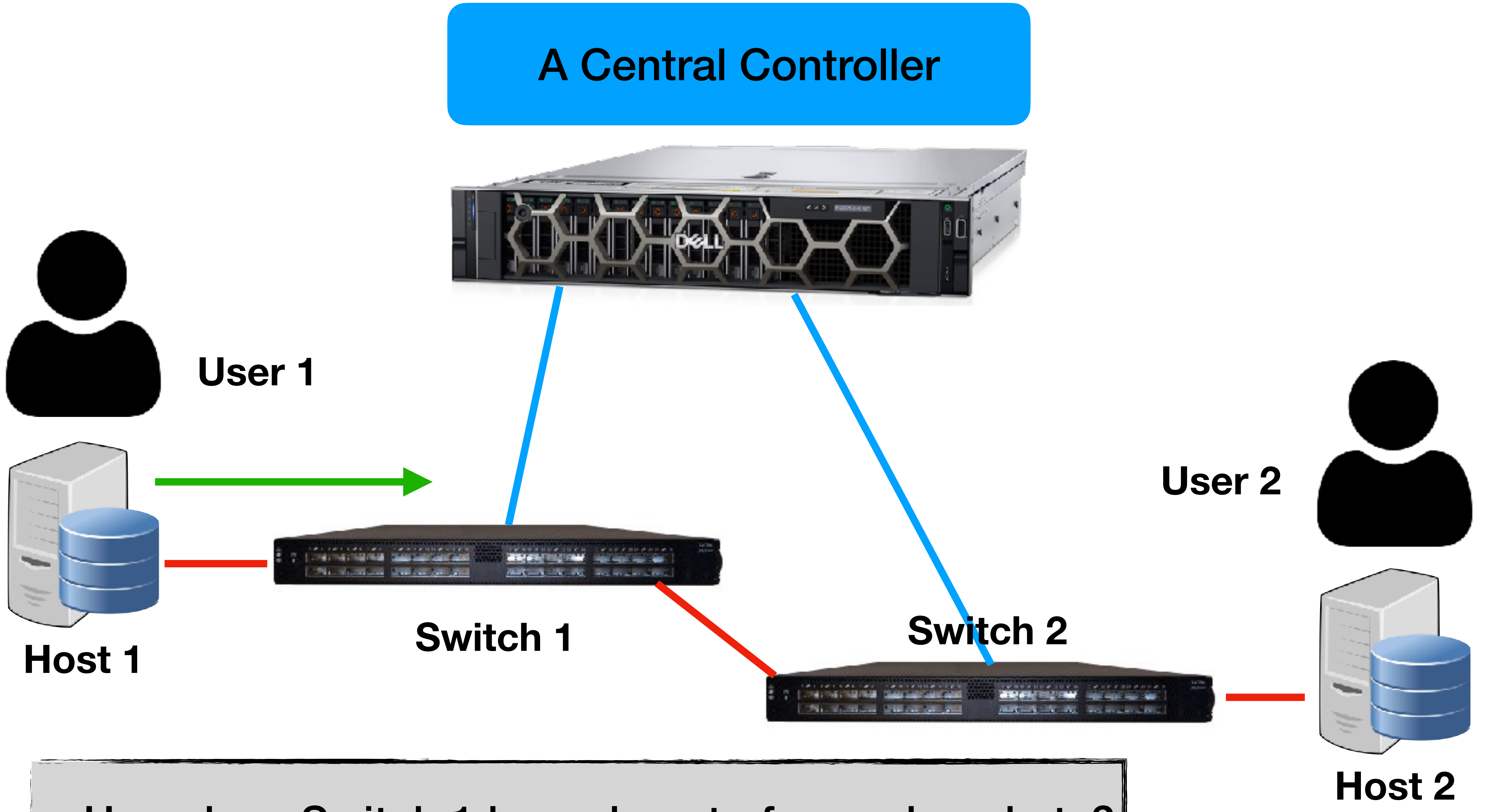
#6: User 1 —> User 2



- How does host 1 know the MAC of host 2?

Controller Functionality:
- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings

#6: User 1 —> User 2

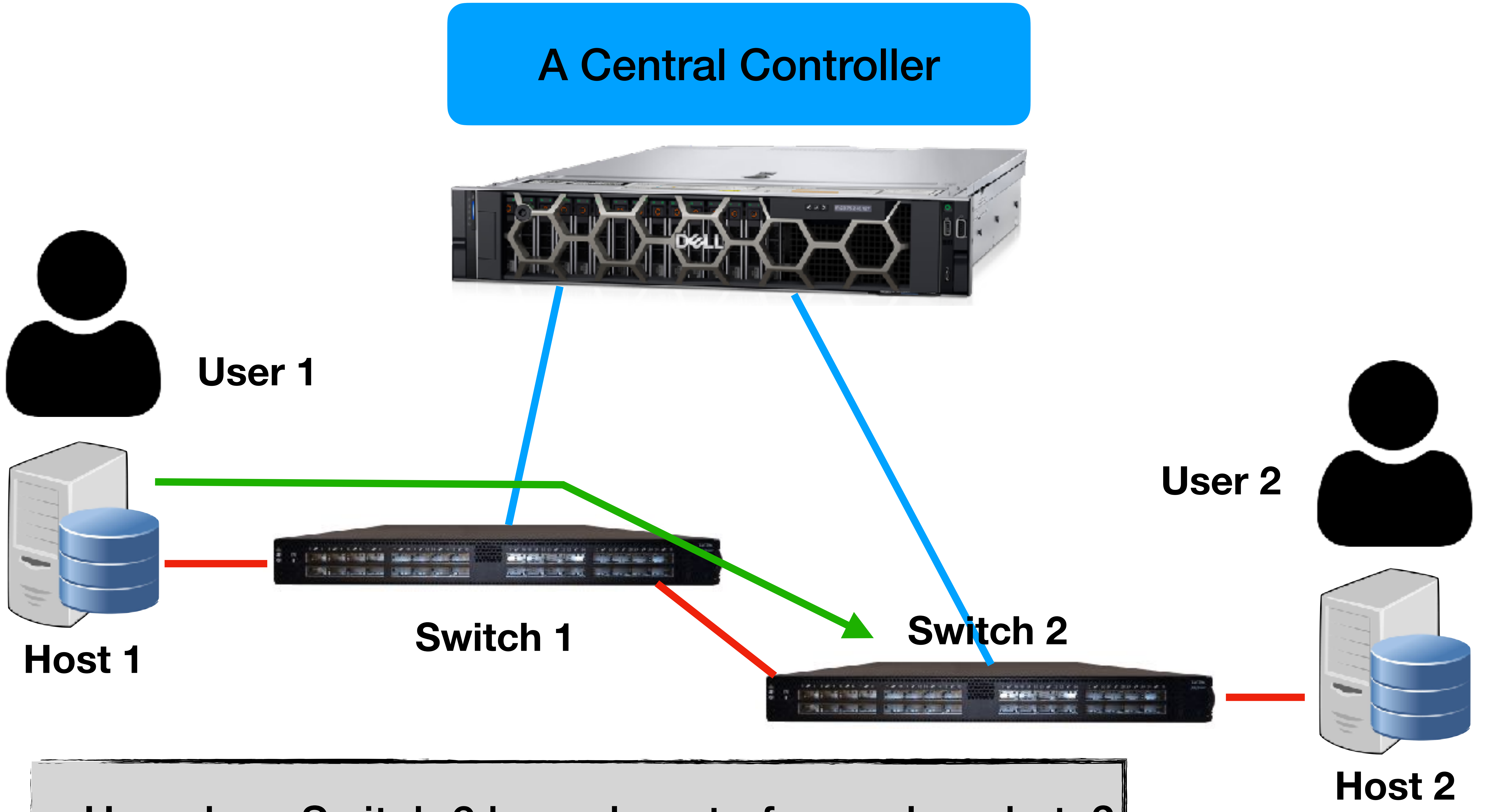


- How does Switch 1 know how to forward packets?

Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings

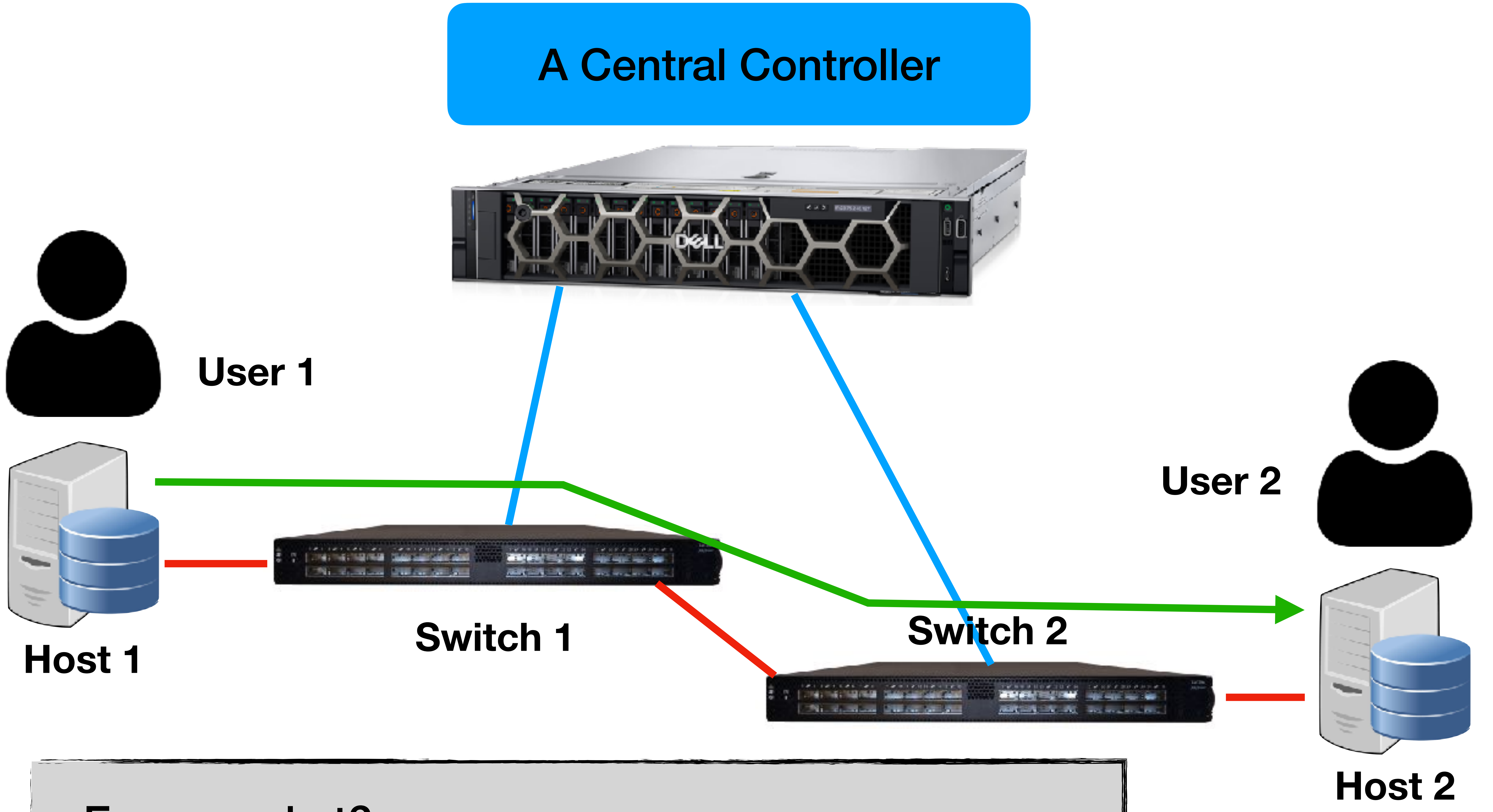
#6: User 1 —> User 2



- How does Switch 2 know how to forward packets?

Controller Functionality:
- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings

#6: User 1 —> User 2

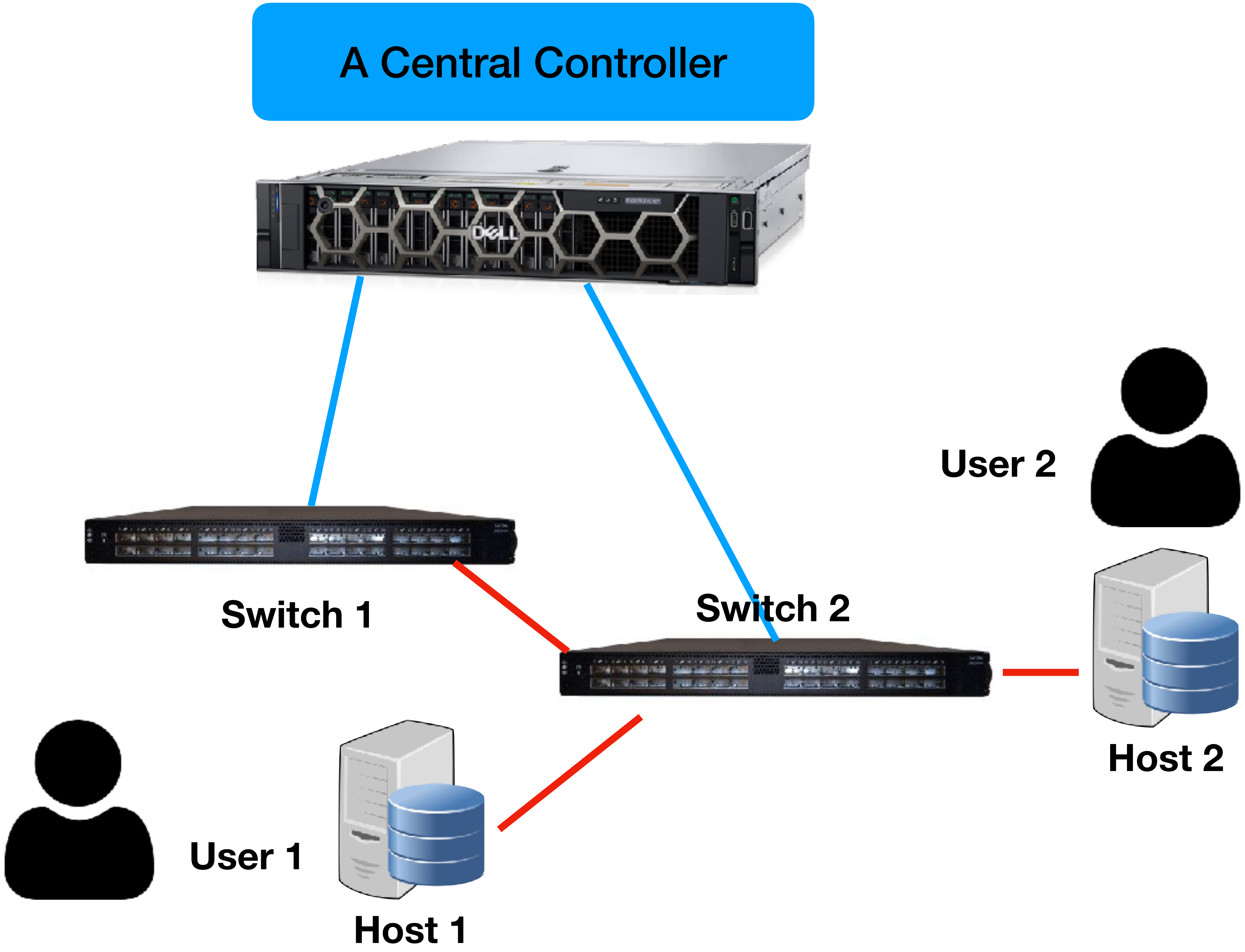


- Every packet?

Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings
- Route calculation
- Forwarding entry installation

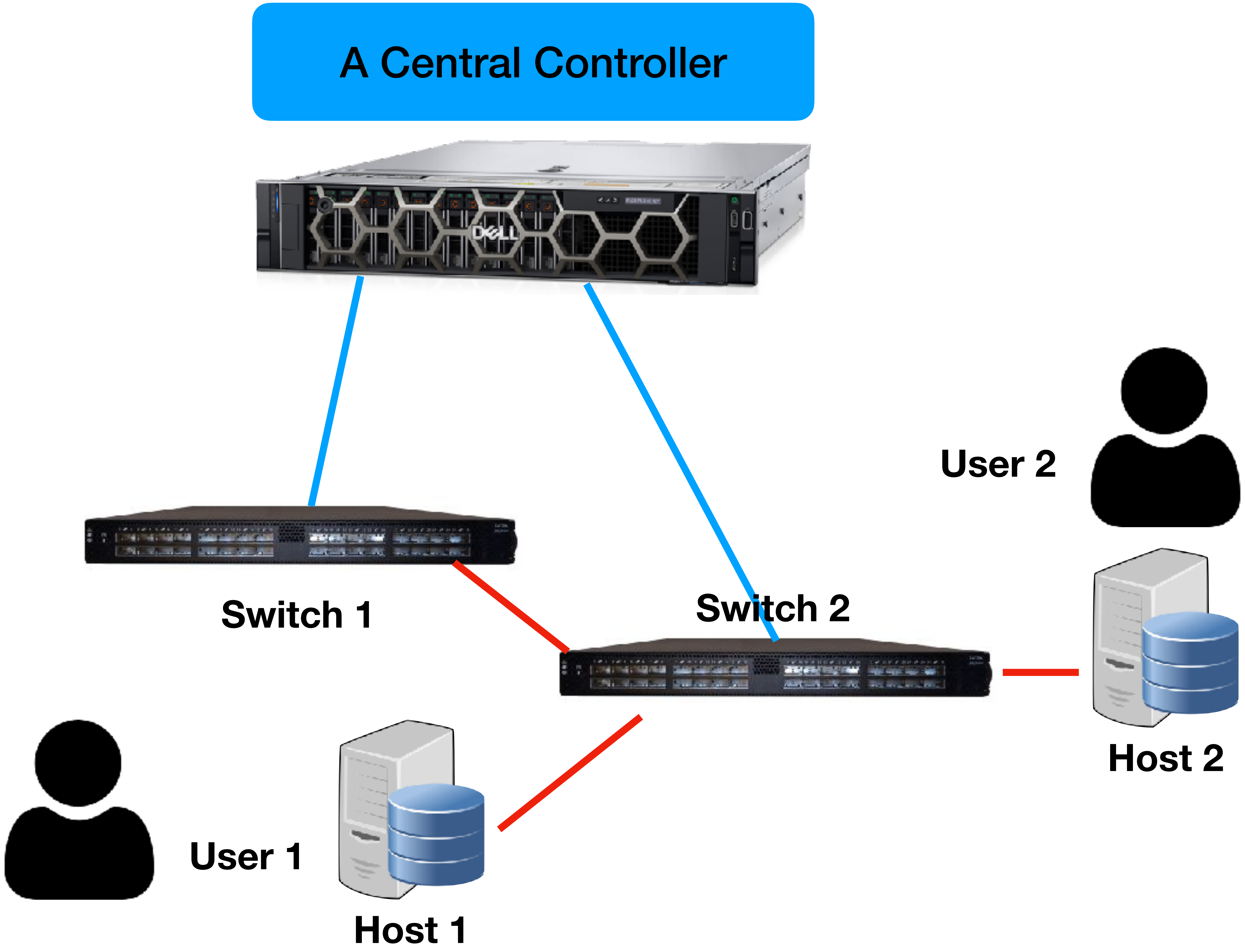
#7: What happens if Host 1 moves?



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings
- Route calculation
- Forwarding entry installation

#8: What happens if the controller is down?



Controller Functionality:

- Register devices and employ a side network
- Maintain the network topology
- Register machines/users
- Authentication
- Maintain the name-address bindings
- Route calculation
- Forwarding entry installation

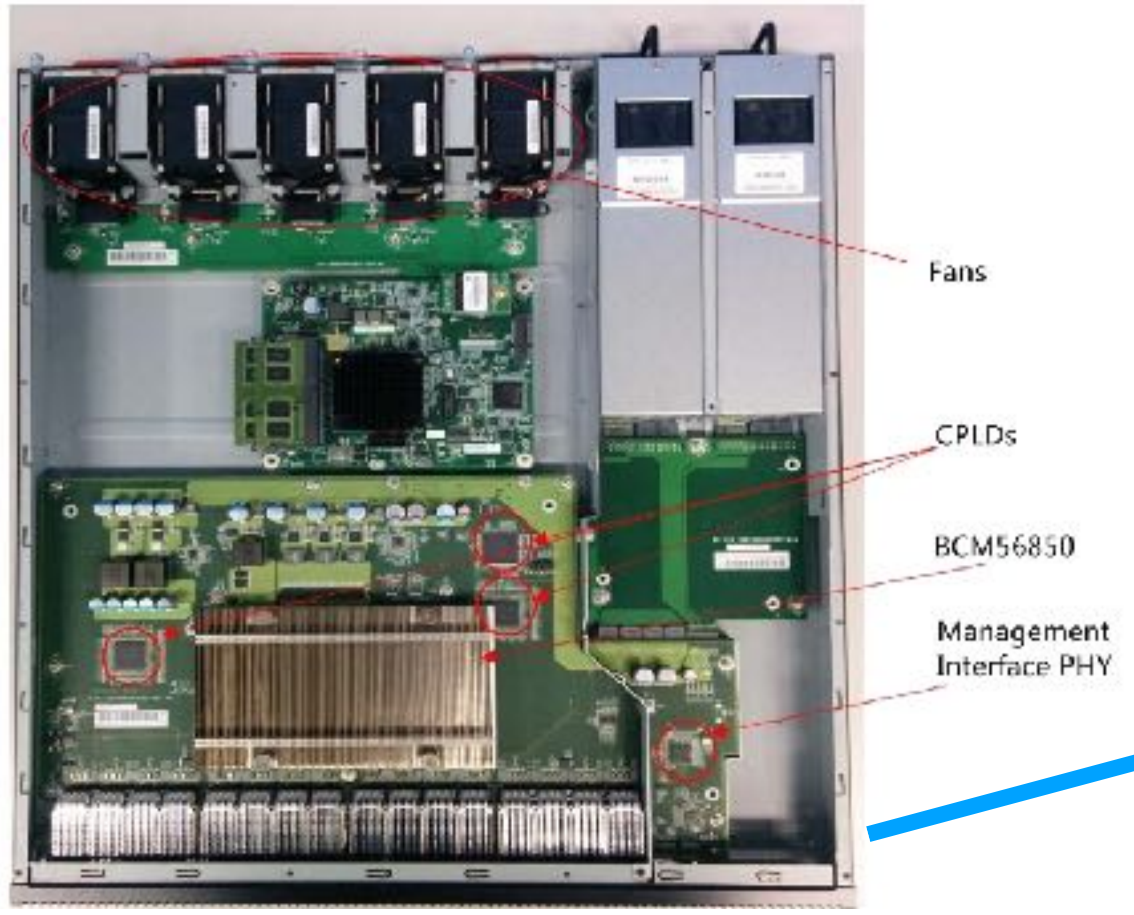
Key functionalities behind Ethane:

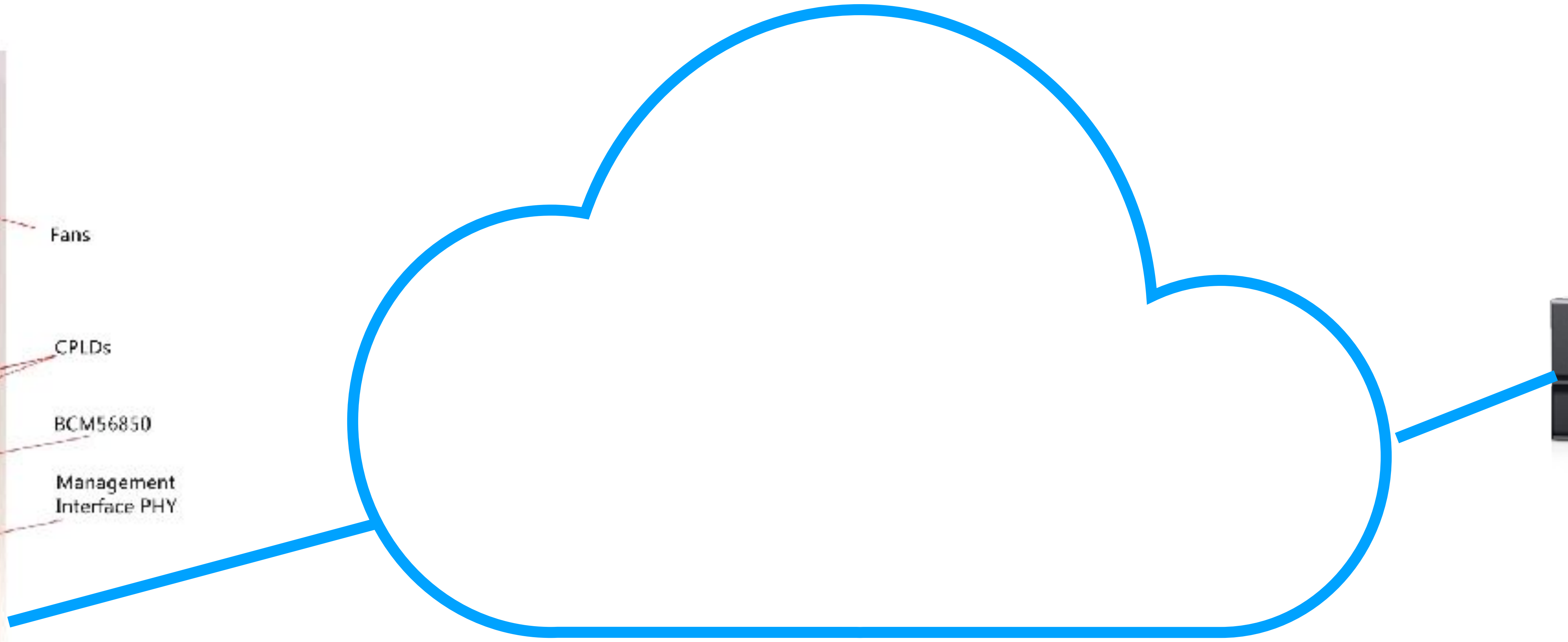
- Registration
- Bootstrapping
- Authentication
- Flow setup
- Forwarding

Enabling Technique #1: Match-action Table

Simple switch logic:

- Small line cards or SoCs
- Forward/Drop/Statistics bookkeeping
- Establish a secure channel with the controller



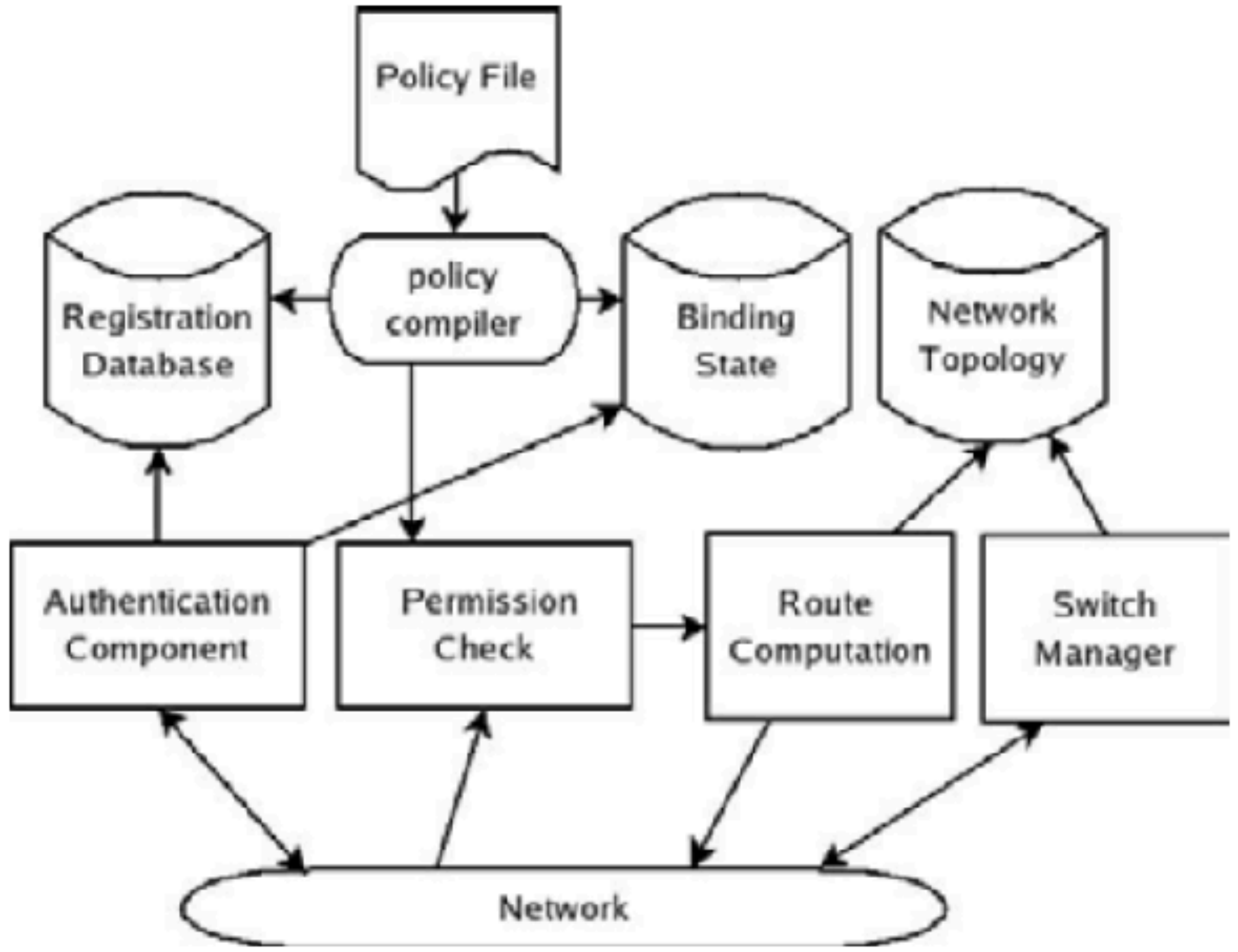


A Central Controller

Enabling Technique #2: Controller

The OS for the Enterprise Network

- Registration
- Authentication
- Tracking Bindings
- Namespace Interface
- Permission Check and Access Granting
- Enforcing Resource Limits



Enabling Technique #3: Fault Tolerance

Handle the controller failure

- cold-standby –> simple
- warm-standby –> weak-consistency
- full-replicated –> Minimal downtime

Enabling Technique #3: Fault Tolerance

Handle the controller failure

- cold-standby -> simple
- warm-standby -> weak-consistency
- full-replicated -> Minimal downtime

Handle the link/switch failure

- Update the topology

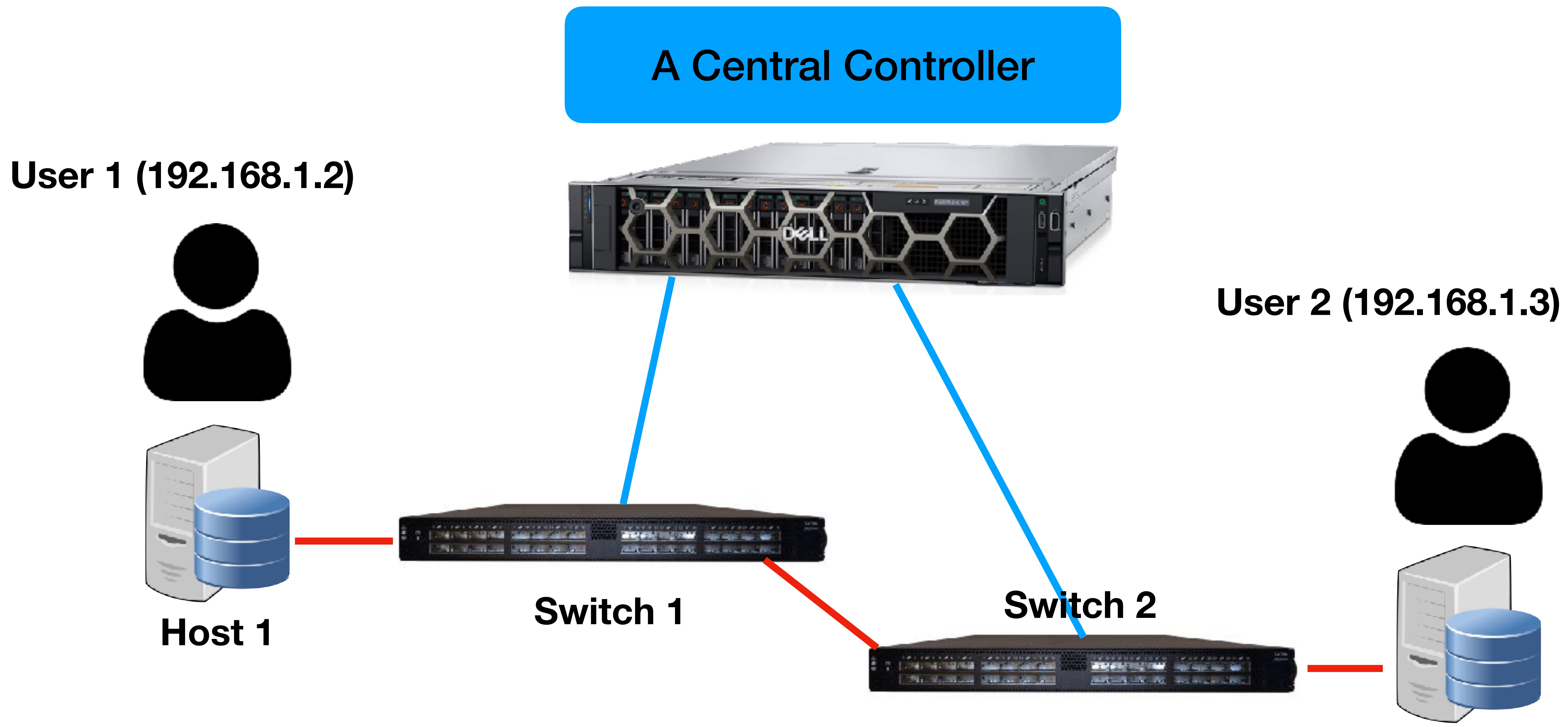
Enabling Technique #4: The Policy Language

A declarative language

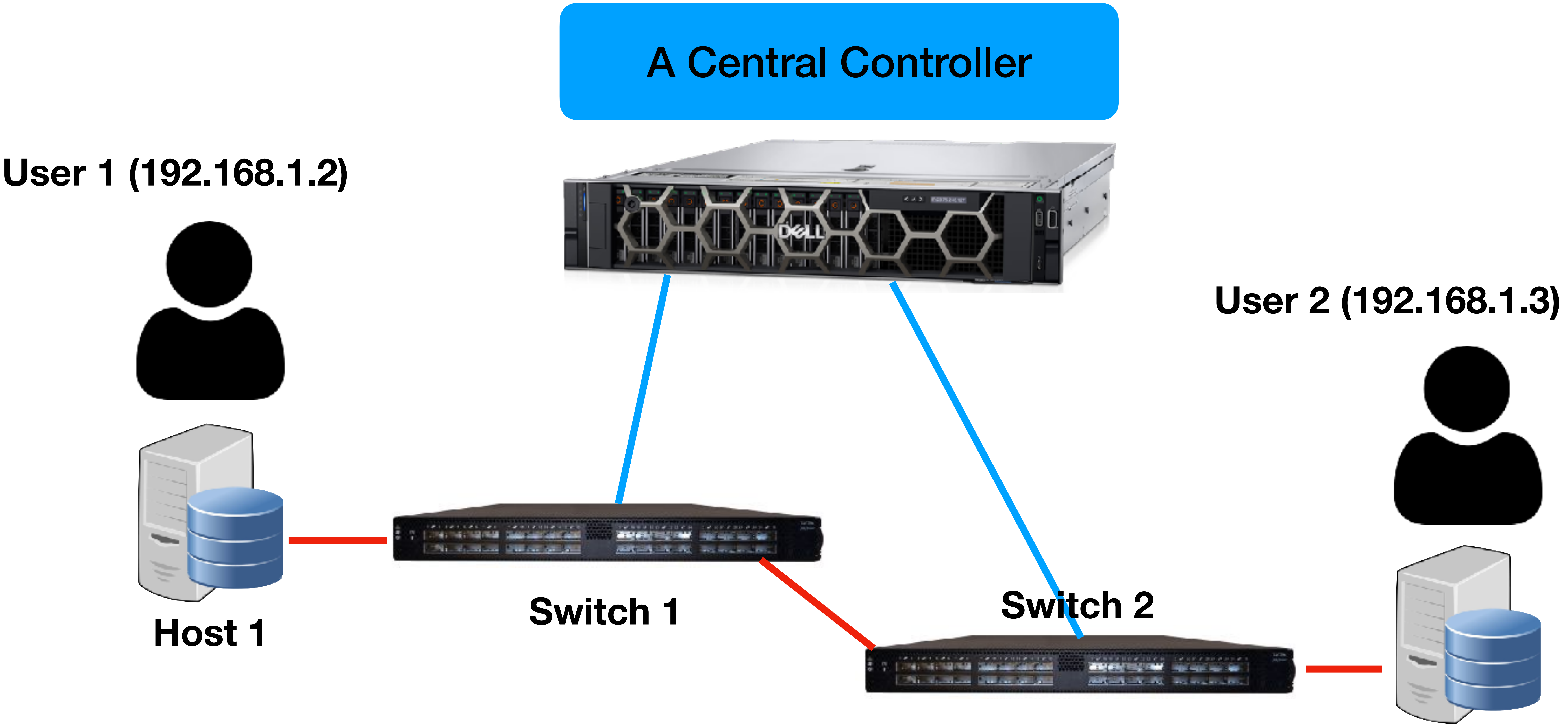
- Condition + Action

[(usrc="bob")∧(protocol="http")∧(hdst="websrv")]:allow;

Seems simple, but…



Case 1: allow flow 192.168.1.2:1234 —> 192.168.1.3:5678



Case 1: allow flow 192.168.1.2:1234 —> 192.168.1.3:5678

Case 2: deny flows 192.168.1.2:* —> 192.168.1.3:* for user 3

Enabling Technique #4: The Policy Language

A declarative language

- Condition + Action

```
[(usrc="bob")∧(protocol="http")∧(hdst="websrv")]:allow;
```

The language system

- Deal with hardware heterogeneity
- Deal with user-defined semantics
- Ensure incrementally correctness
- …

OpenFlow and Software-Defined Network (SDN)

OpenFlow: Enabling Innovation in Campus Networks

Nick McKeown
Stanford University

Tom Anderson
University of Washington

Hari Balakrishnan
MIT

Guru Parulkar
Stanford University

Larry Peterson
Princeton University

Jennifer Rexford
Princeton University

Scott Shenker
University of California, Berkeley

Jonathan Turner
Washington University in St. Louis

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
Authors take full responsibility for this article's technical content.
Comments can be posted through CCR Online.

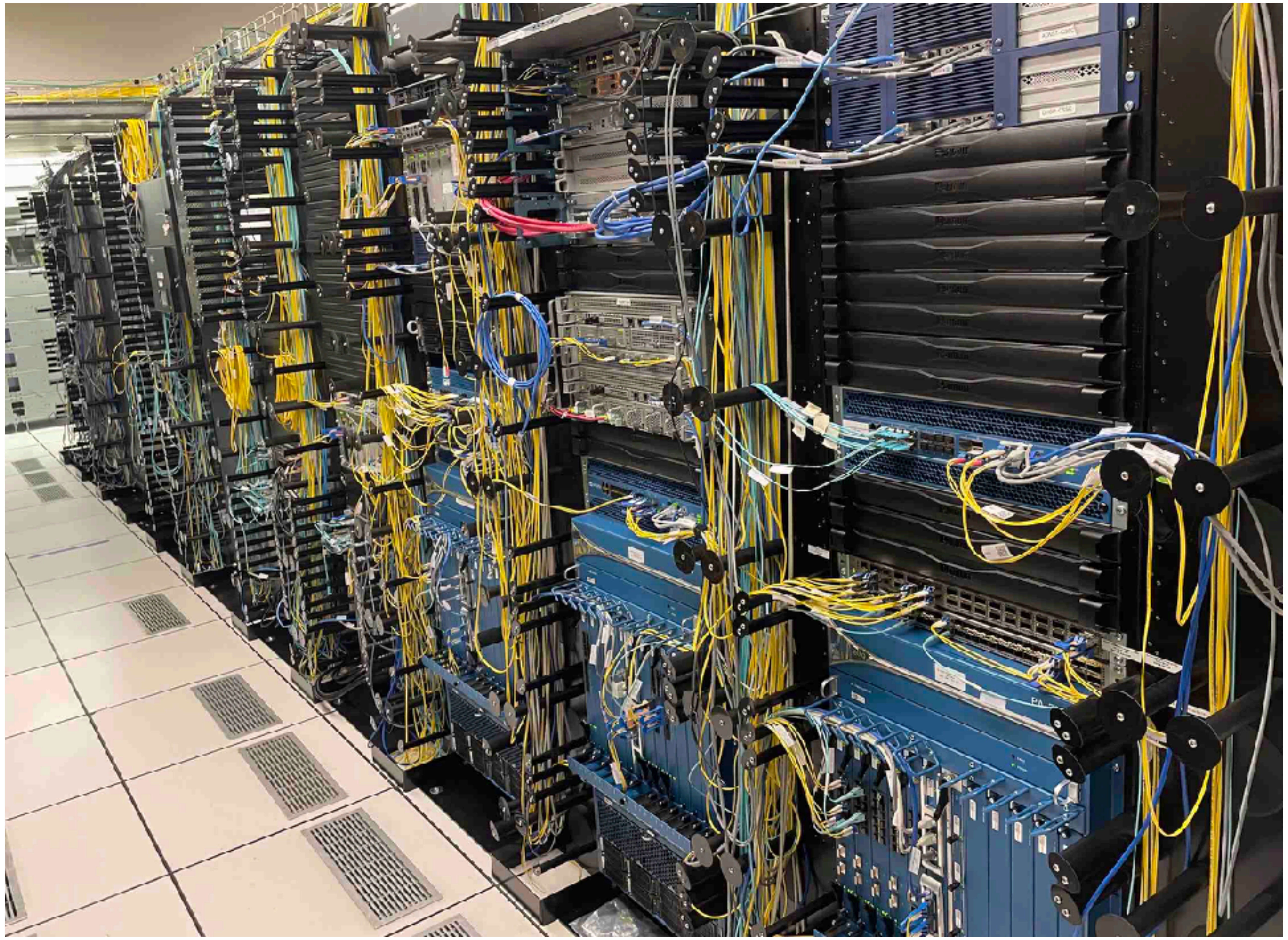
ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

to experiment with production traffic, which have created an exceedingly high barrier to entry for new ideas. Today, there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as

SDN @UW-Madison Campus Backbone



Summary

- Today
  - SDN

- Next
  - RMT (Sigcomm'13) and AFQ (NSDI'18)