Διαχείριση Edge και Cloud δικτύων βασισμένων στο λογισμικό (CSIS109)

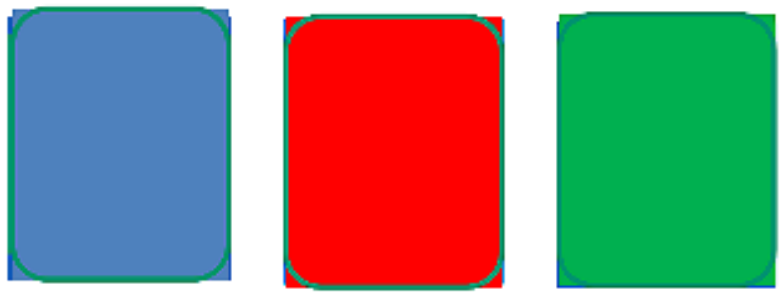
Δρ. Ειρήνη Λιώτου

eliotou@hua.gr

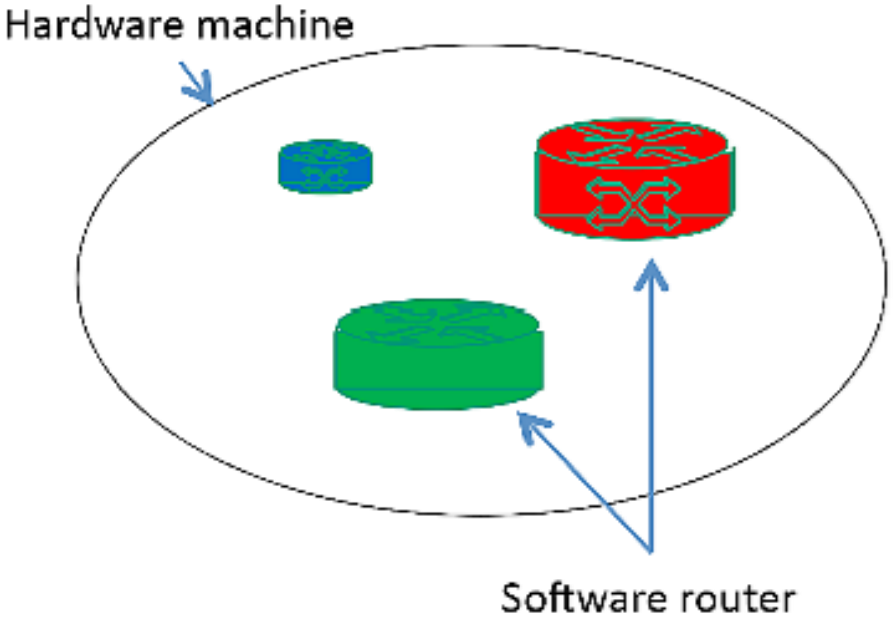
4/3/2025

Virtualization: Concept

- Motivation: Constructing software networks to replace hardware networks
- Transform HW machines to SW machines (with the exception of those which handle the reception of terrestrial and wireless signals)



Physical router



Virtualization: Benefits

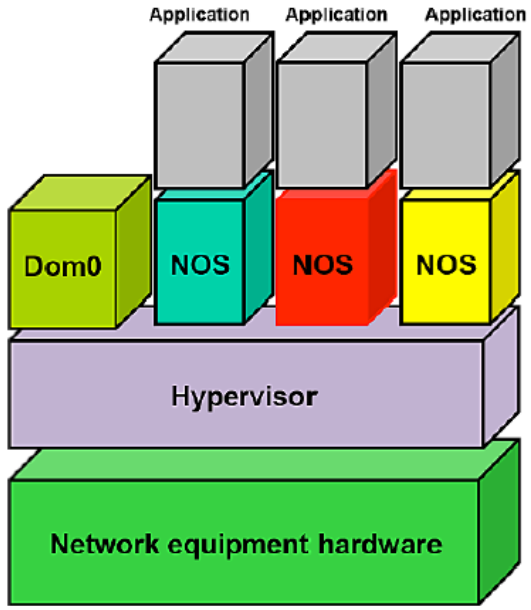
- Flexibility: size of the virtual devices (e.g. routers can change depending on their workload)
  - Little resources at night-time vs. very large at peak times
- Energy efficiency:
  - Share the resources more effectively
  - Move those resources
  - Group them together on physical machines
  - Put other idle machines on stand-by

Hypervisor

- This is a software program that enables multiple virtual machines (VMs) to run simultaneously
  - Macroscopic scale → simultaneously (in parallel)
  - Microscopic scale → sequentially
  - Each VM's processing time must be sufficiently short to give the impression of parallelism

Hypervisor

- The hypervisor is a Virtual Machine Monitor (VMM), operating on standard hardware platforms
- VMs are “domains” running on top of the hypervisor
- Each VM may have its own operating system and applications
- The VMM isolates the different VMs and manages the sharing of resources, so that the execution of one VM does not affect the performance of the others
- Multiplexing – De-multiplexing



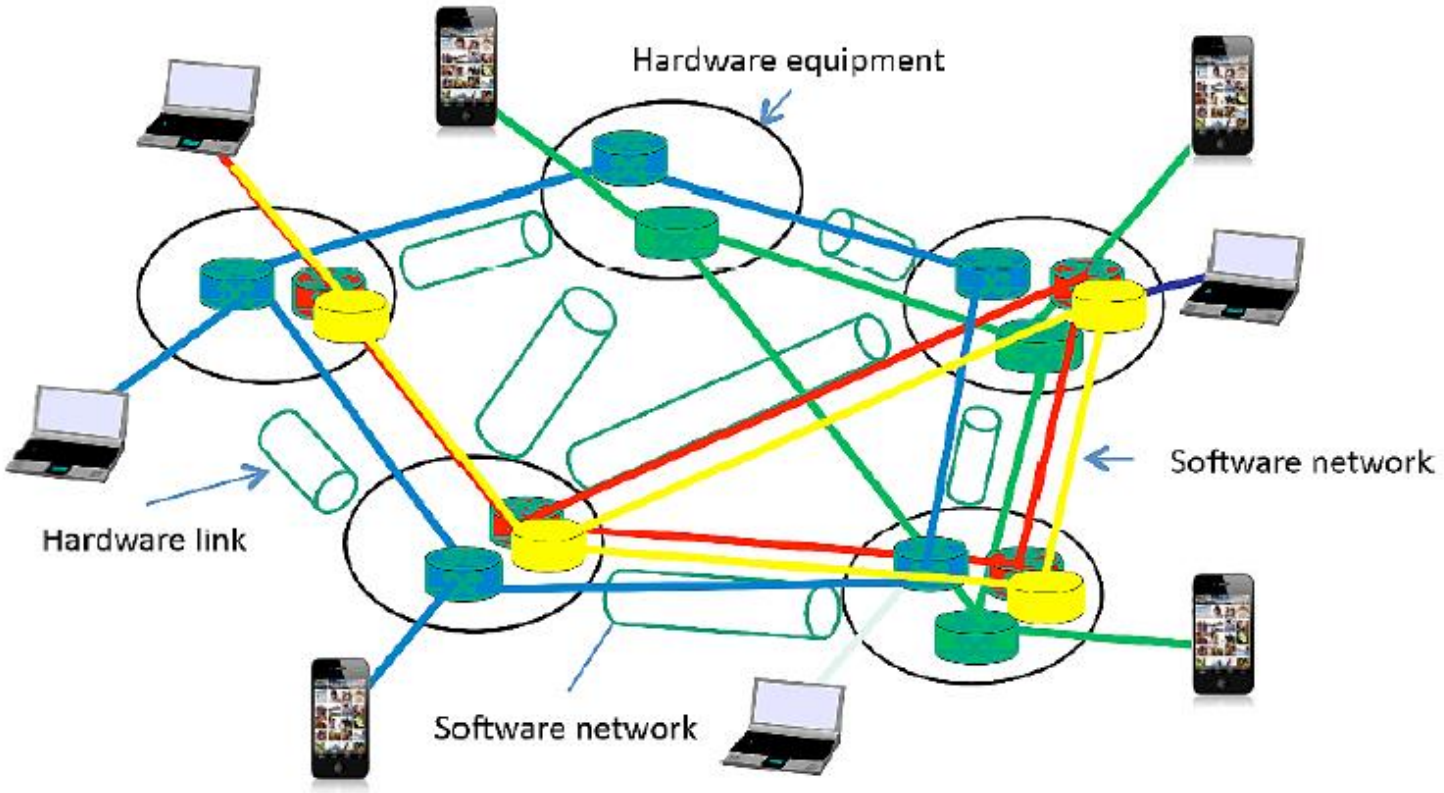
NOS: Network Operating System

User domain interfaces

- The user domains employ virtual input/output peripherals, controlled by virtual drivers, to ask the “dom0” for access to the peripheral.
- Each user domain has its own virtual network interfaces, known as **foreground** interfaces, required for network communication
- The **background** interfaces are created in the dom0, corresponding to each foreground interface in a user domain, and act as proxy
- The foreground interfaces are perceived by the operating systems, working on the user domains, as real interfaces
- “Bridge mode”

Software networks

- VMs can be used to create virtual networks, a.k.a. software networks
- Need to link VMs together as if they were different physical machines



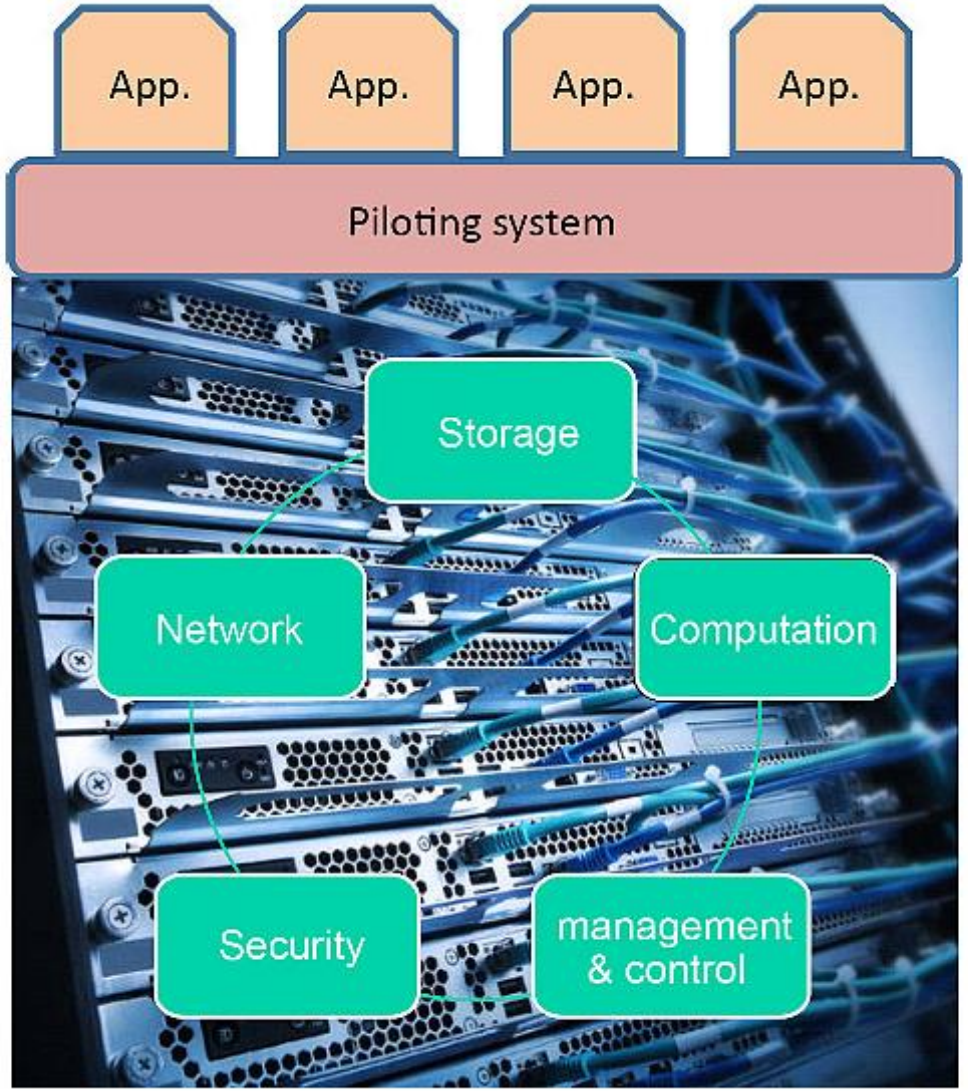
"Per-service" or "personalized" software network

Software networks' properties

- Migration of e.g. a router from one physical node to another
  - When a node begins to fail or is overloaded
- Migration of a node does not involve transporting the whole code
  - The program needed is already present in the remote node, but is idle
  - Begin running the program and send the node configuration info
- Isolation is important, so that an attack on one network does not affect the other networks
  - A token-based algorithm is used (The networks spend their tokens on the basis of certain tasks performed, such as the transmission of *n* bytes)

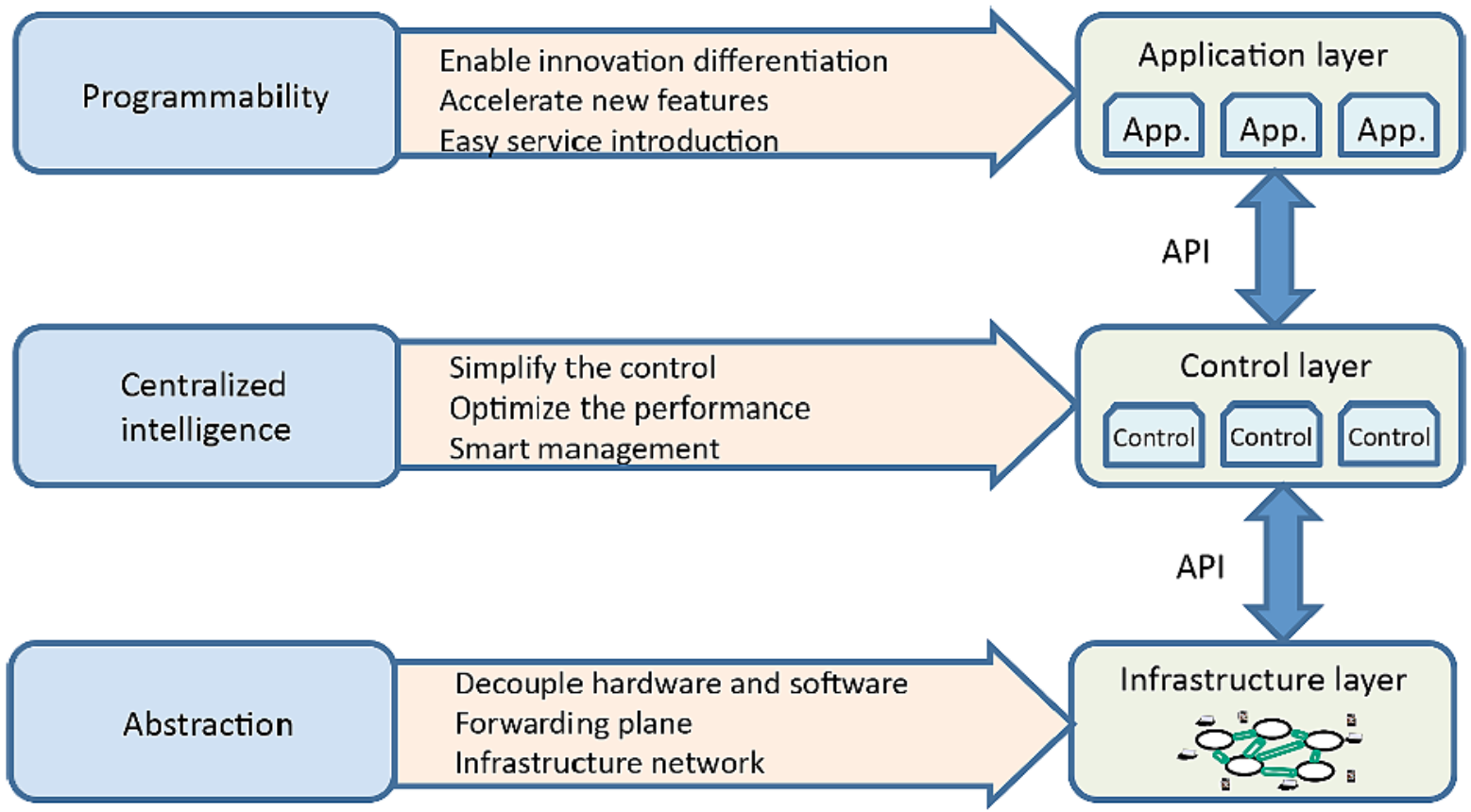
The 5 domains necessary

- Used to characterize an information- and operation system for a company
- Business applications
- Applications to control or orchestrate the environment
- “Autopilot systems” (orchestrators)

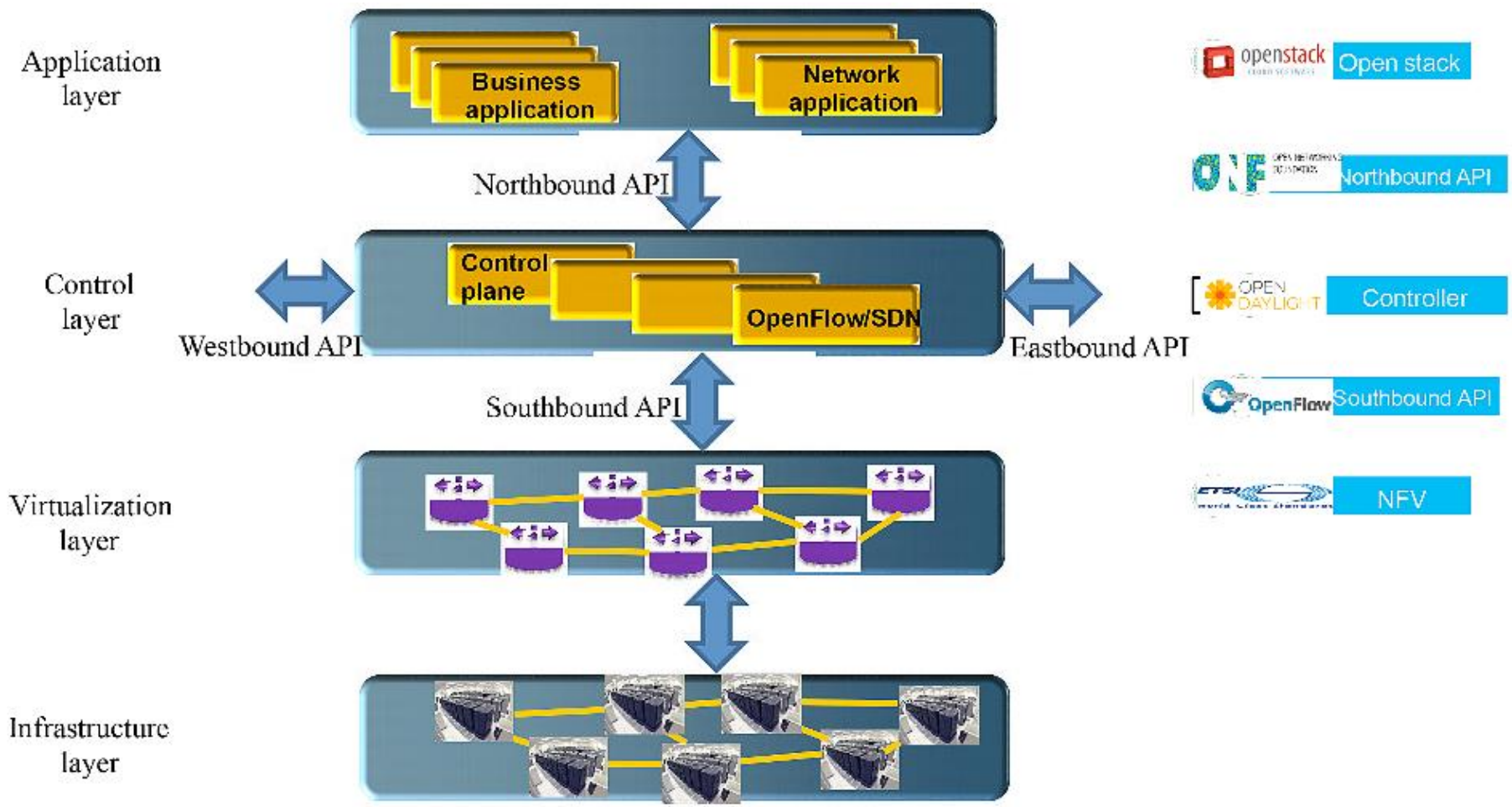


ONF architecture

- Standardization of SDN by Open Network Foundation (ONF)



Detailed ONF architecture

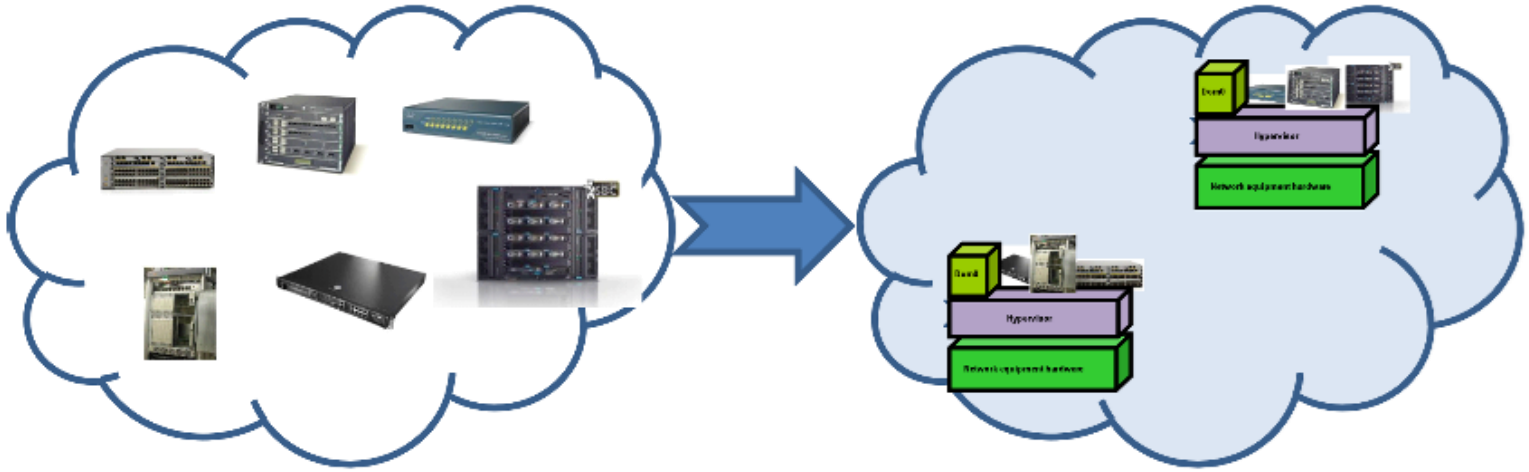


Interfaces

- The northbound interface facilitates communication between the application level and the controller. Its purpose is to describe the needs of the application and to pass along the commands to orchestrate the network.
- The southbound interface describes the signaling necessary between the control plane and the virtualization layer. The controller must be able to determine the elements that will make up the software network for which it is responsible.
- In the other direction, the current network resource consumption must be fed back so that the controller has as full a view as possible of the usage of the resources.
- The eastbound interface enables two controllers of the same type to communicate with one another and make decisions together.
- The westbound interface must also facilitate communication between two controllers, but ones which belong to different sub-networks.

NFV (Network Functions Virtualization)

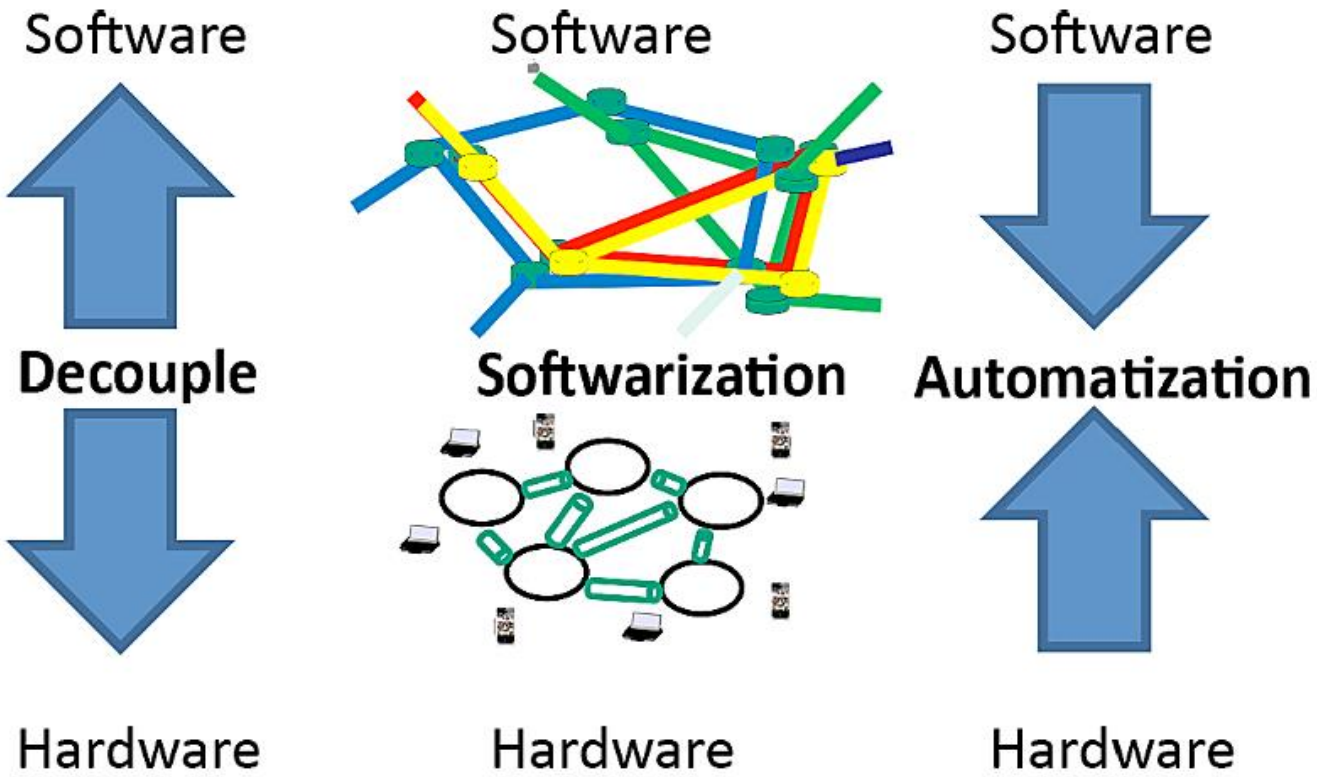
- Goal: decouple the network functions from the network equipment
- Standardize network functions, virtualizing them and facilitating their execution in different places from the original physical machine.
- Enable to position the software performing the functions of a device on a different machine than the device itself (e.g. in the Cloud)



NFV working groups

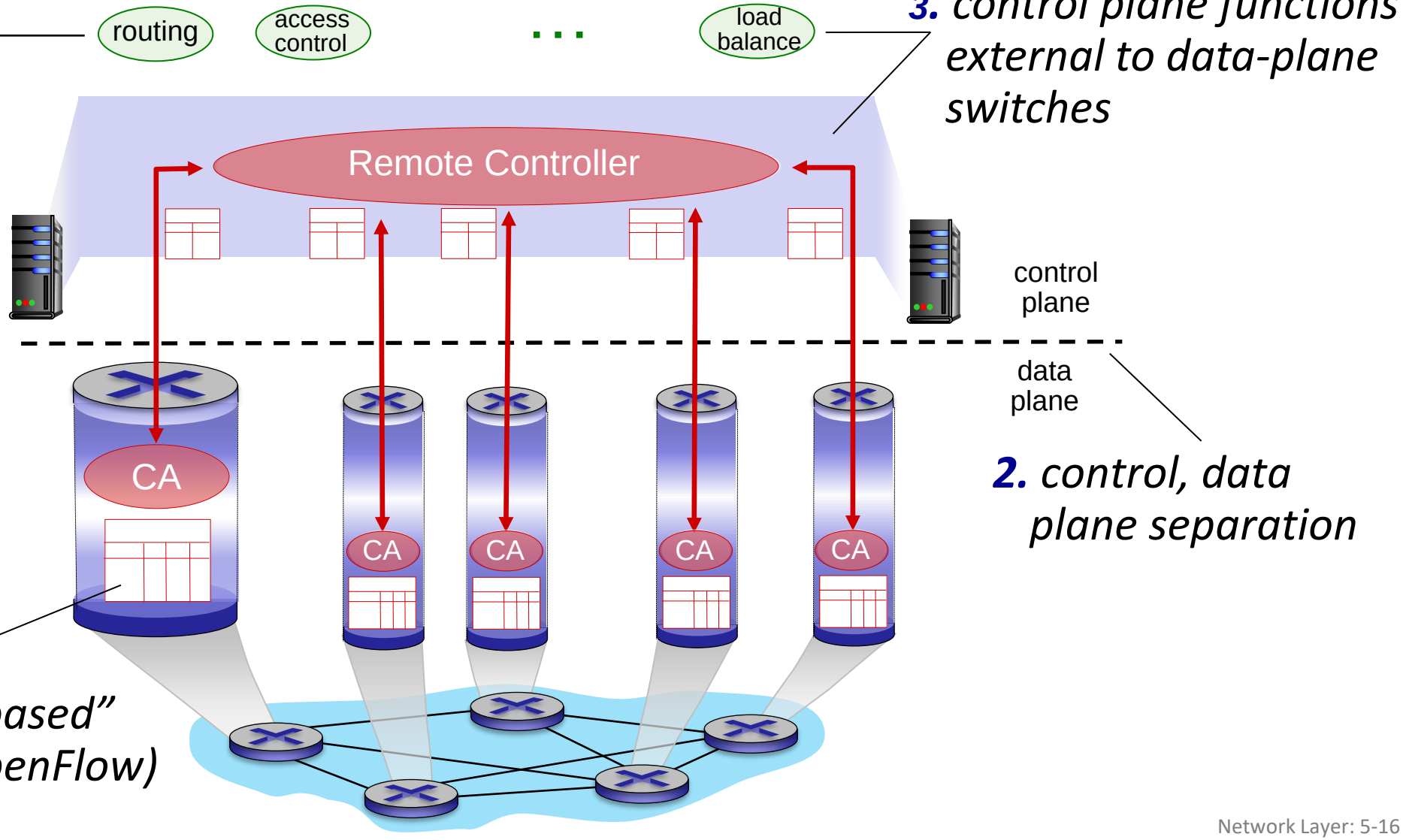
1. “Architecture of the Virtualization, produces a reference architecture for a virtualized infrastructure and points of reference to interconnect the different components of that architecture.
2. “Management and Orchestration”, defines the rollout, instantiation, configuration and management of network services.
3. “Software Architecture”, defines the reference software architecture for the execution of virtualized functions.
4. “Security Expert Group”, works on the security of the software architecture.
5. “Performance and Portability Expert Group”, provides solutions to optimize performances and manage the portability of the VMs.

SDN principles



Software defined networking (SDN)

4. *programmable control applications*



3. *control plane functions external to data-plane switches*

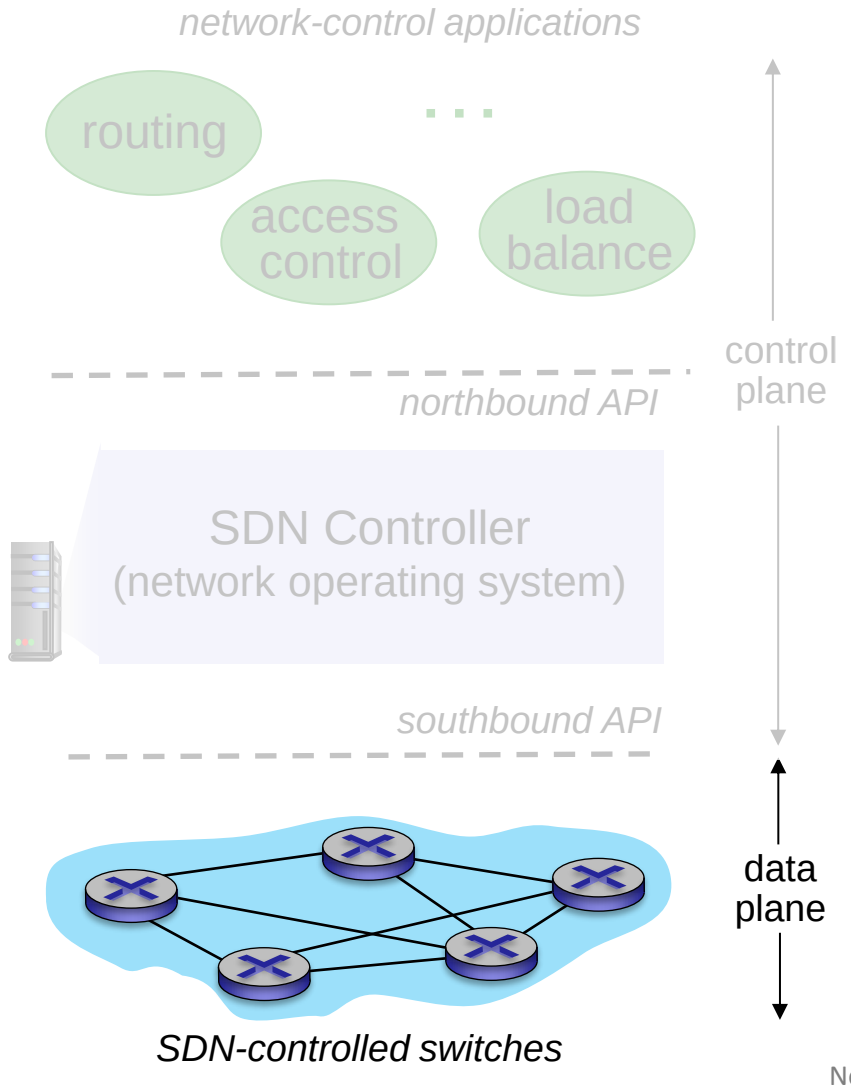
2. *control, data plane separation*

1: *generalized "flow-based" forwarding (e.g., OpenFlow)*

Software defined networking (SDN)

Data-plane switches:

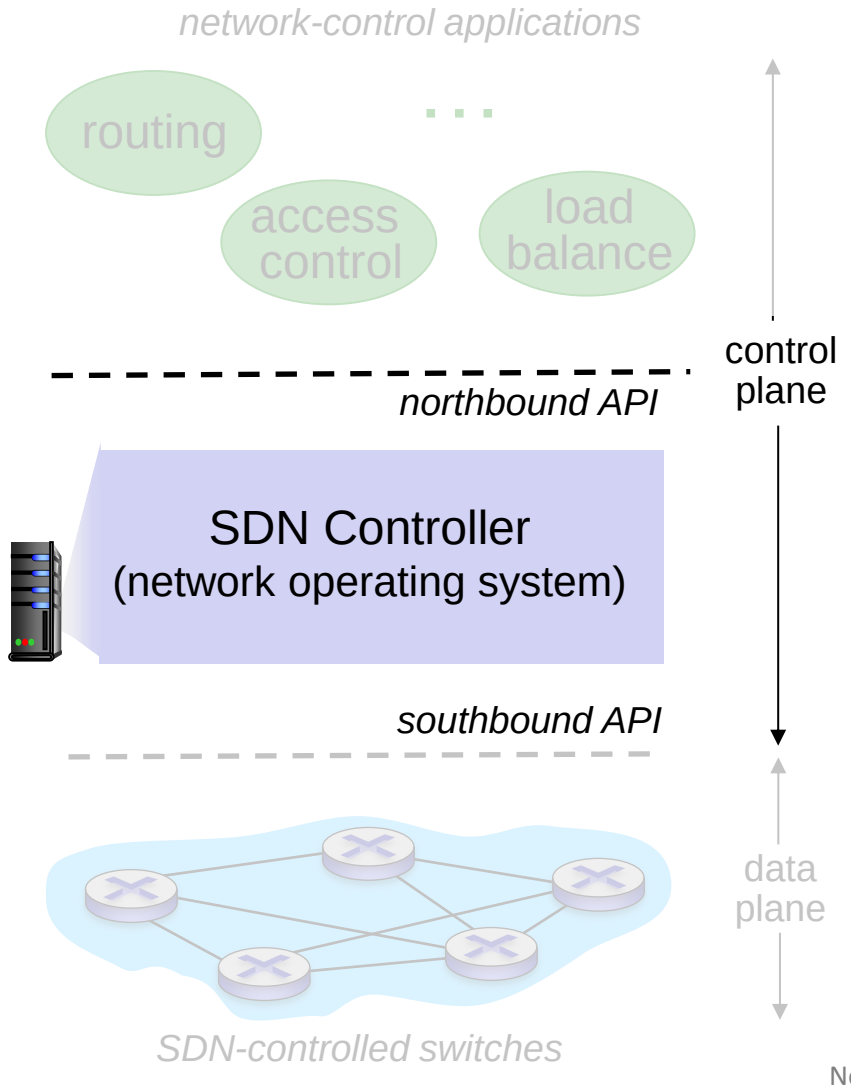
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)



Software defined networking (SDN)

SDN controller (network OS):

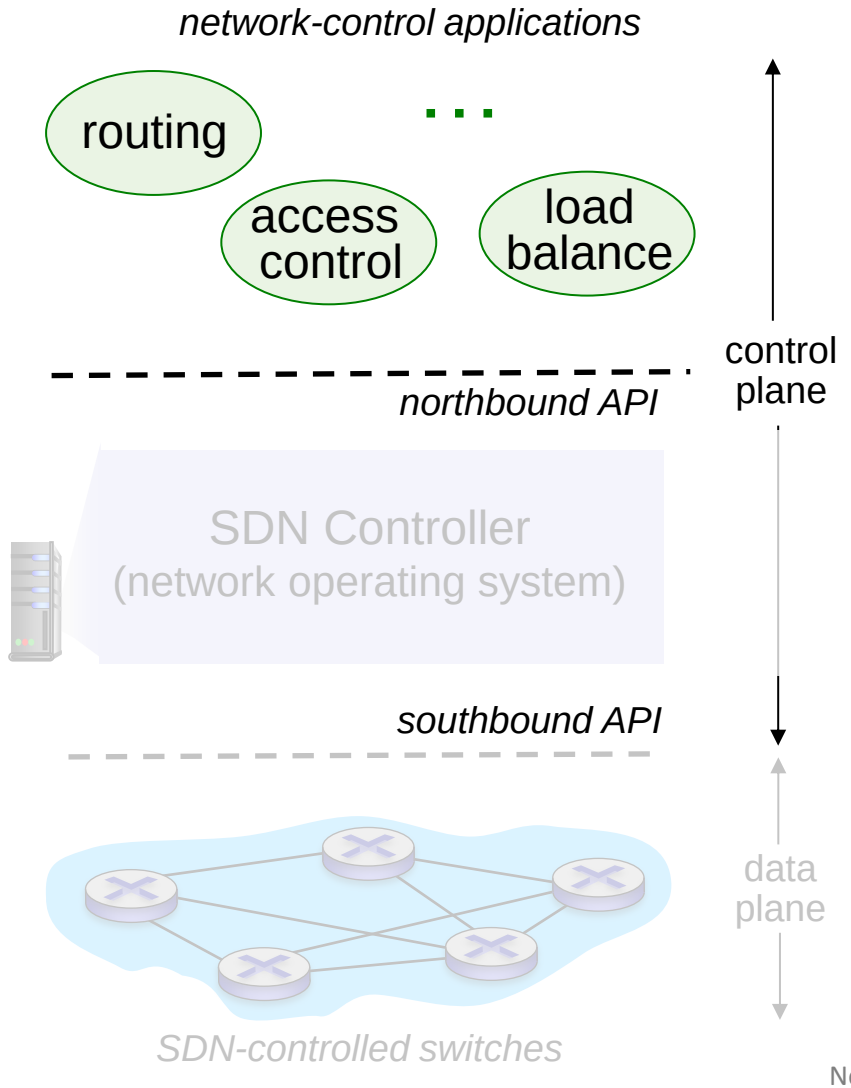
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



Software defined networking (SDN)

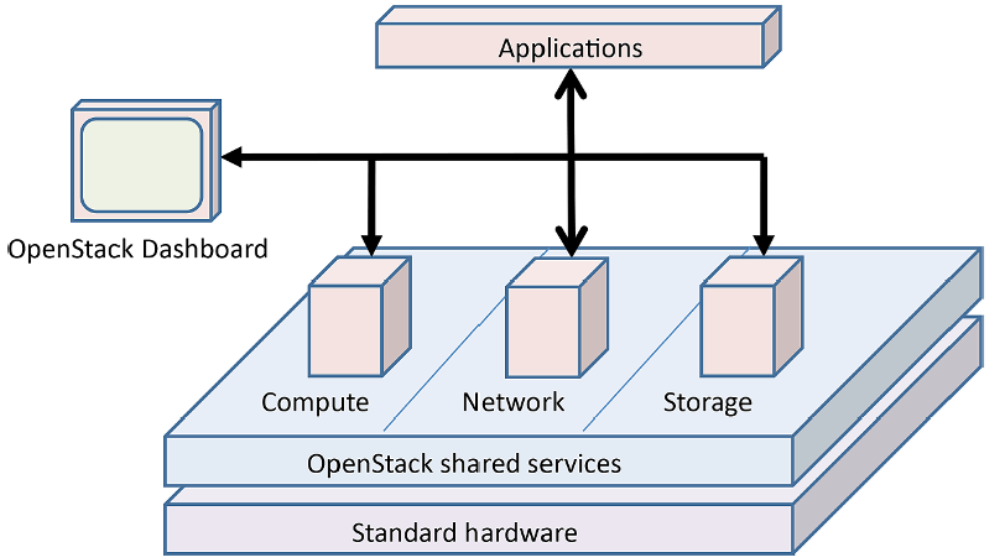
network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller



OpenStack cloud management system

- **Orchestration** is the automated management of compute, networking, and storage resources to deploy and operate applications efficiently.
- **OpenStack** provides orchestration tools that automate the provisioning, scaling, and lifecycle management of workloads, including Virtual Network Functions (VNFs) in Network Function Virtualization (NFV) environments.
- OpenStack is an open-source **cloud computing platform** that allows users to create and manage public and private clouds. It provides Infrastructure-as-a-Service (**IaaS**) by enabling the deployment of VMs, storage, networking, and other cloud resources.

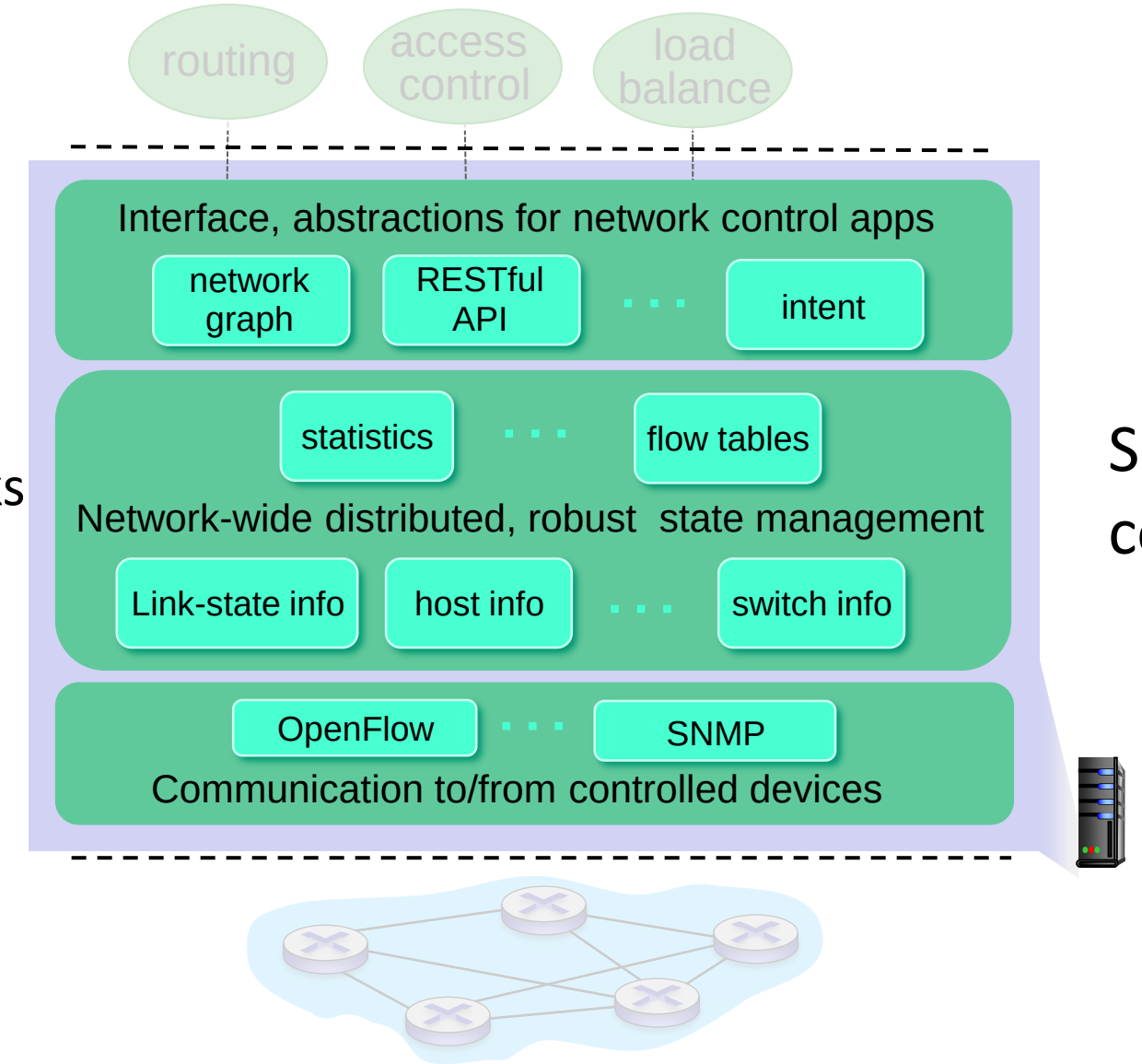


Components of SDN controller

interface layer to network control apps: abstractions API

network-wide state management: state of networks links, switches, services: a *distributed database*

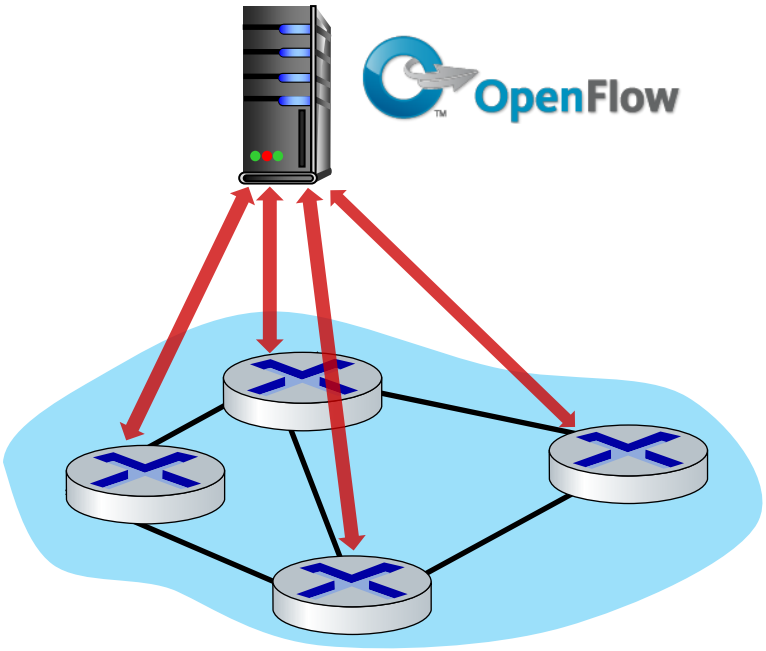
communication: communicate between SDN controller and controlled switches



OpenFlow protocol

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc.)
- distinct from OpenFlow API
  - API used to specify generalized forwarding actions

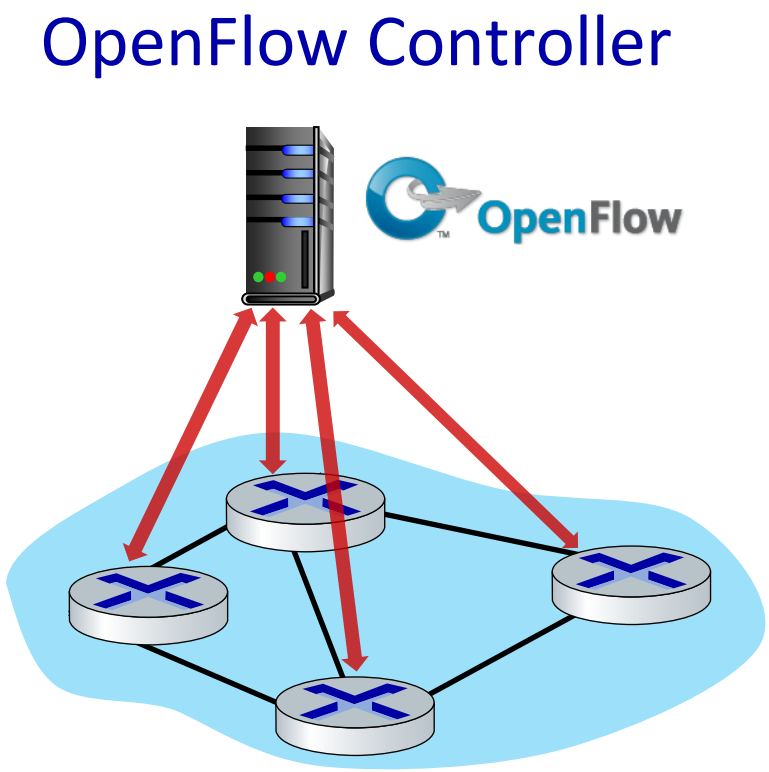
OpenFlow Controller



OpenFlow: controller-to-switch messages

Key controller-to-switch messages

- *features:* controller queries switch features, switch replies
- *configure:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *packet-out:* controller can send this packet out of specific switch port

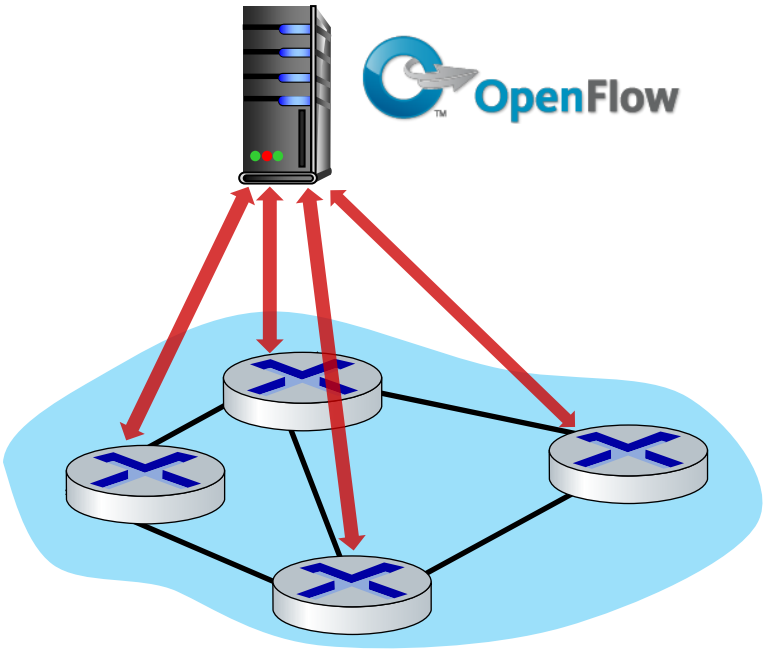


OpenFlow: switch-to-controller messages

Key switch-to-controller messages

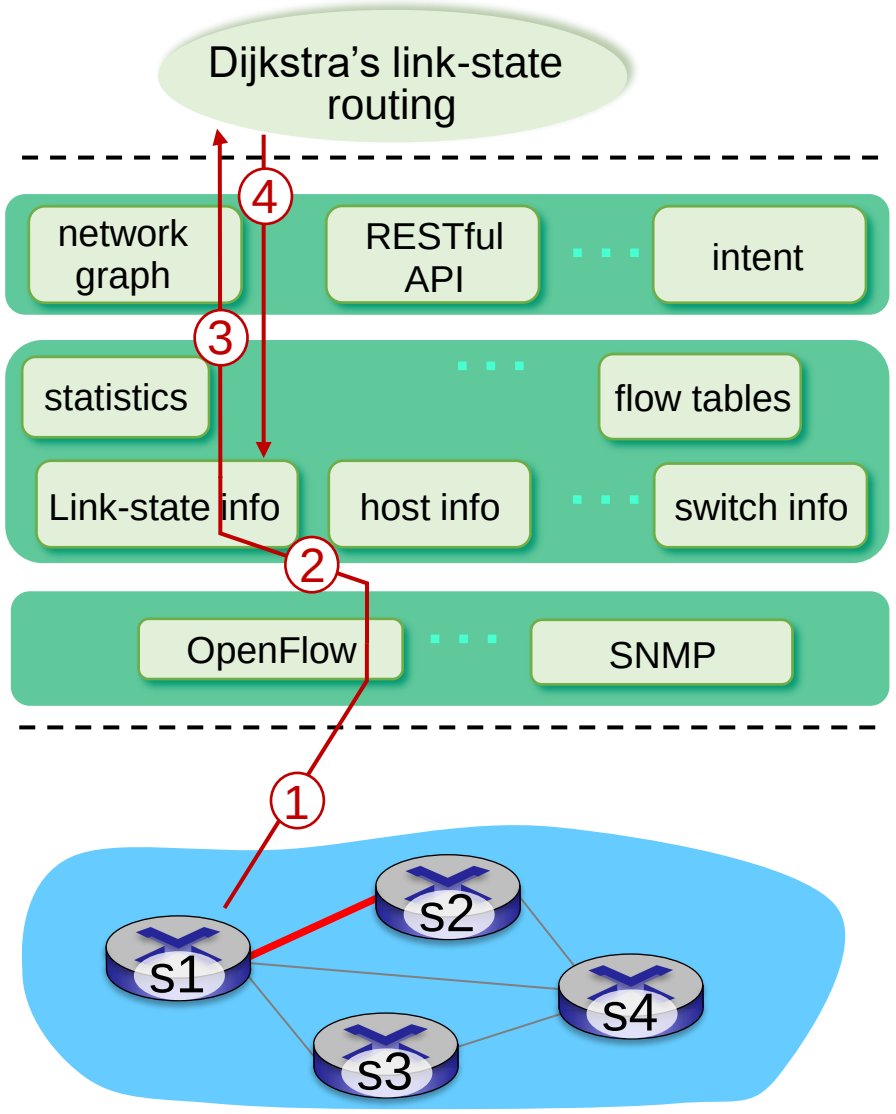
- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.

OpenFlow Controller



Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

SDN: control/data plane interaction example



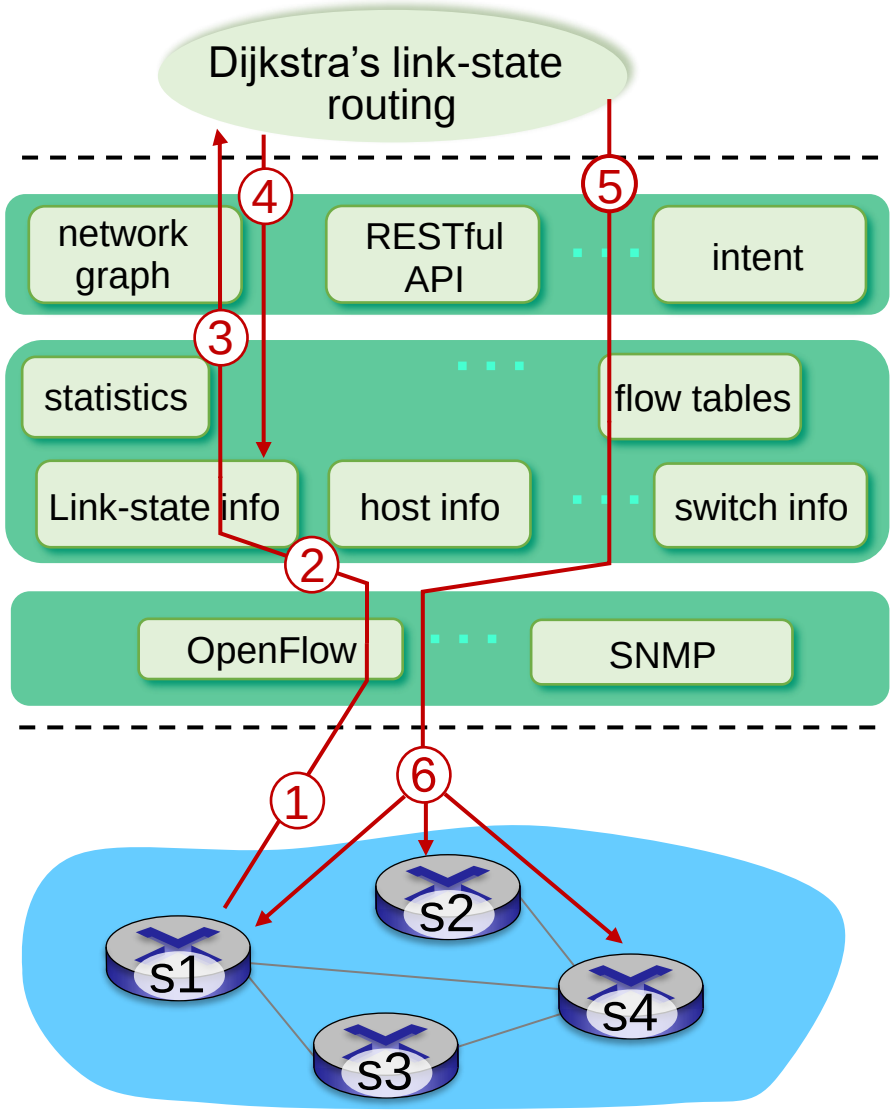
① S1, experiencing link failure uses OpenFlow port status message to notify controller

② SDN controller receives OpenFlow message, updates link status info

③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.

④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

SDN: control/data plane interaction example



⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

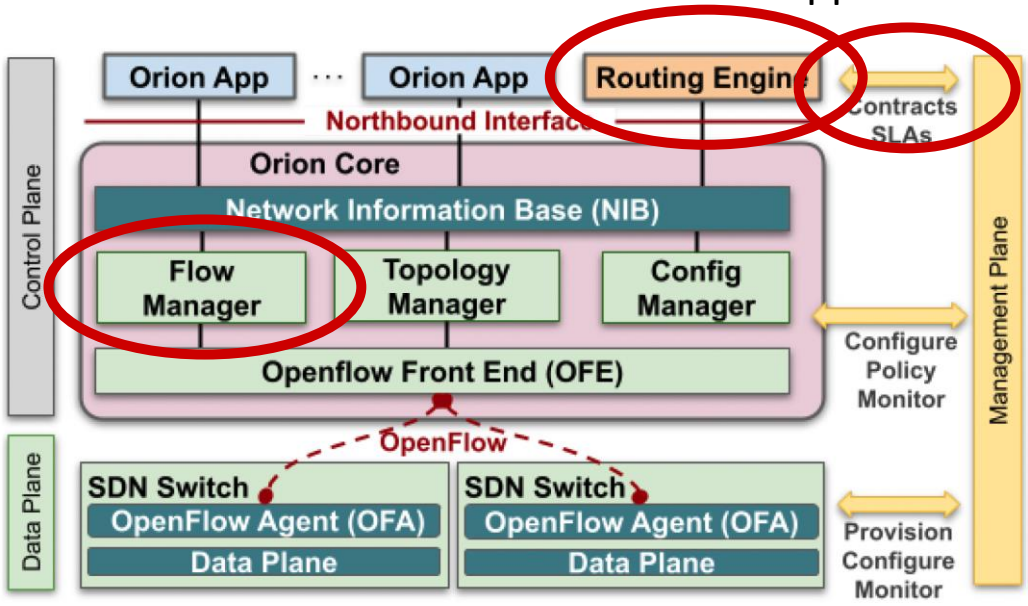
⑥ controller uses OpenFlow to install new tables in switches that need updating

Google ORION SDN control plane

ORION: Google's SDN control plane (*NSDI'21*): control plane for Google's datacenter (Jupiter) and wide area (B4) networks

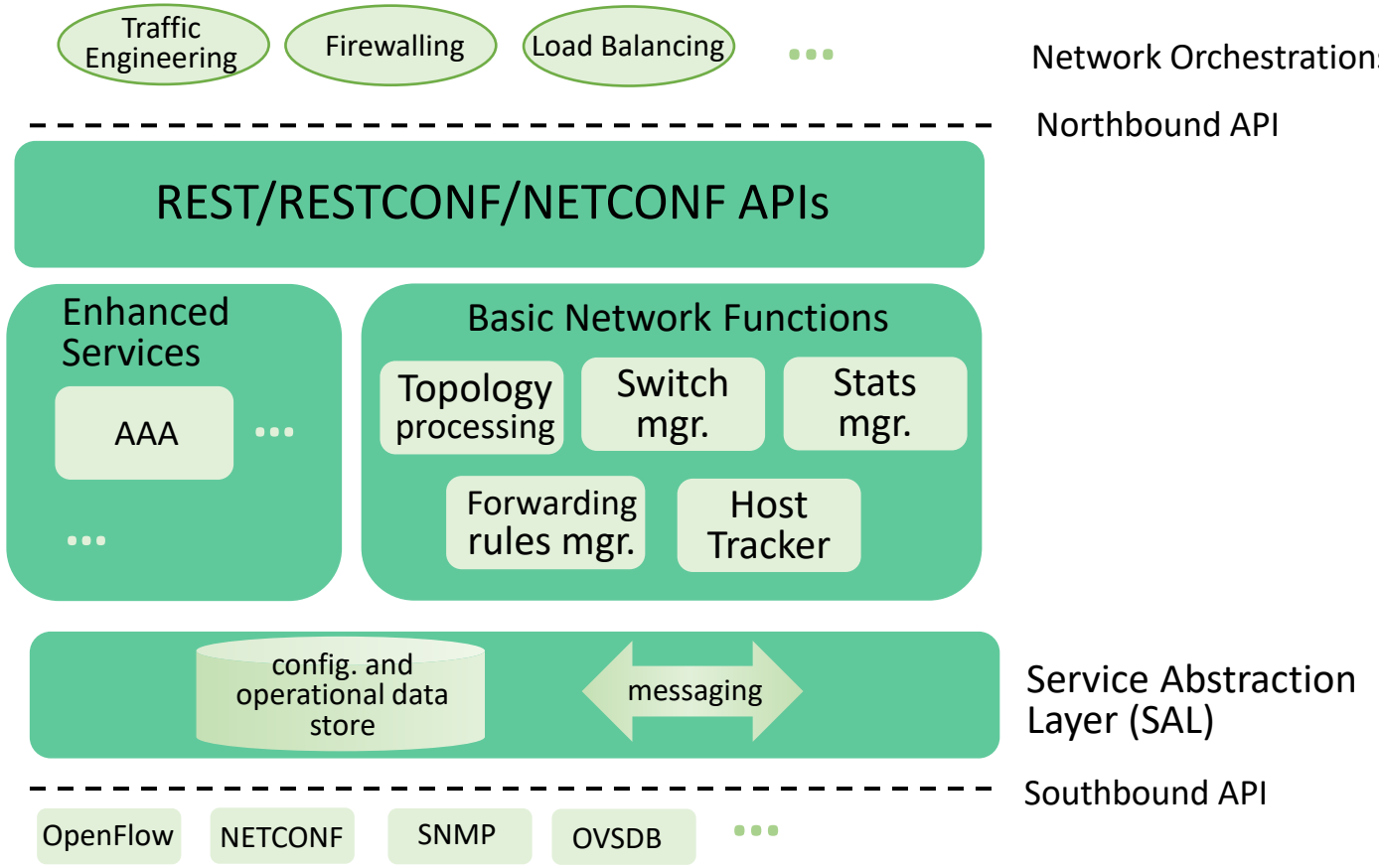
- routing (intradomain, iBGP), traffic engineering: implemented in *applications* on top of ORION core
- edge-edge flow-based controls (e.g., CoFlow scheduling) to meet contract SLAs
- management: pub-sub distributed microservices in Orion core, OpenFlow for switch signaling/monitoring

Orion SDN architecture and core apps



Note: ORION provides *intradomain* services within Google's network

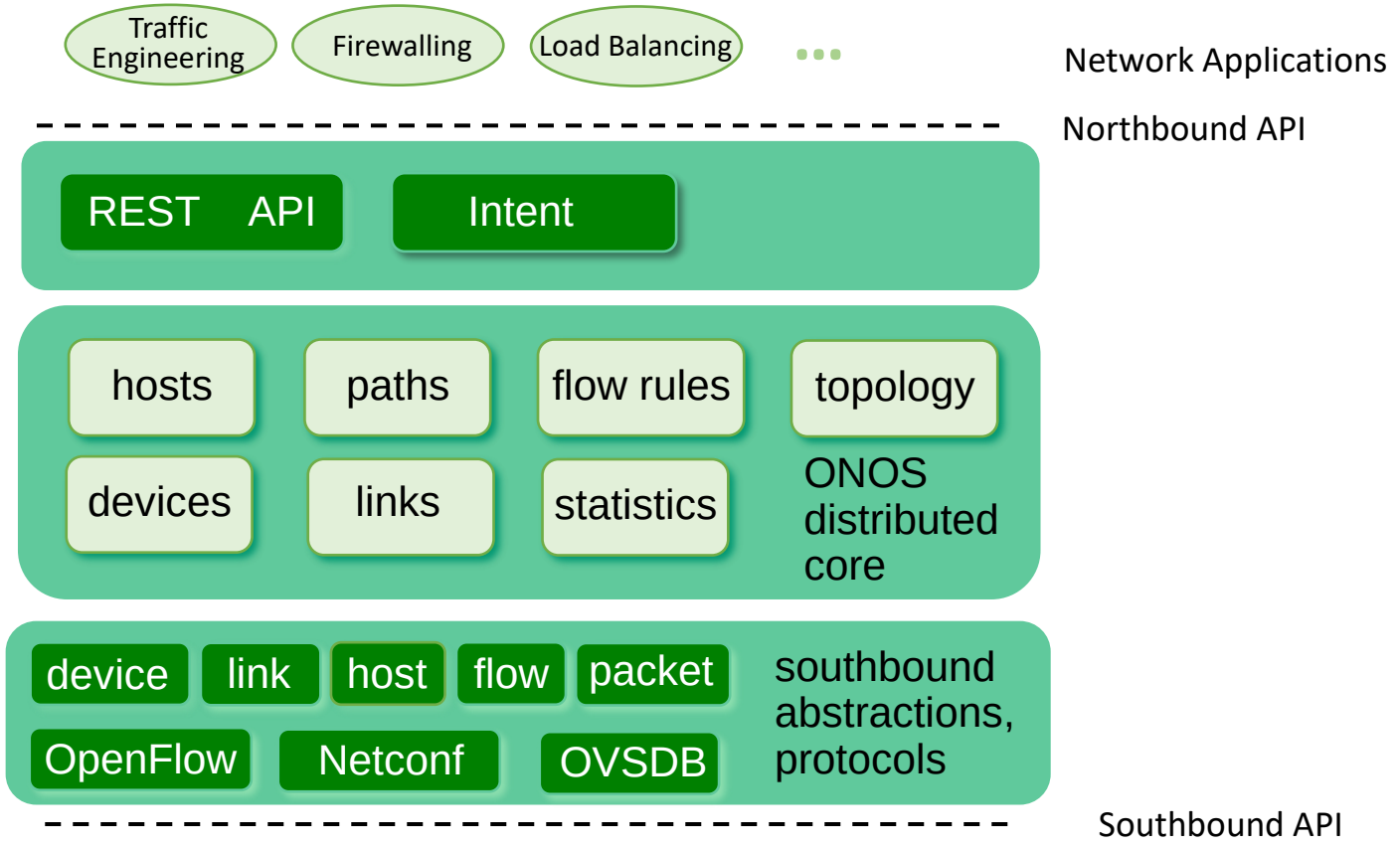
OpenDaylight (ODL) controller



Service Abstraction Layer:

- interconnects internal, external applications and services

ONOS controller

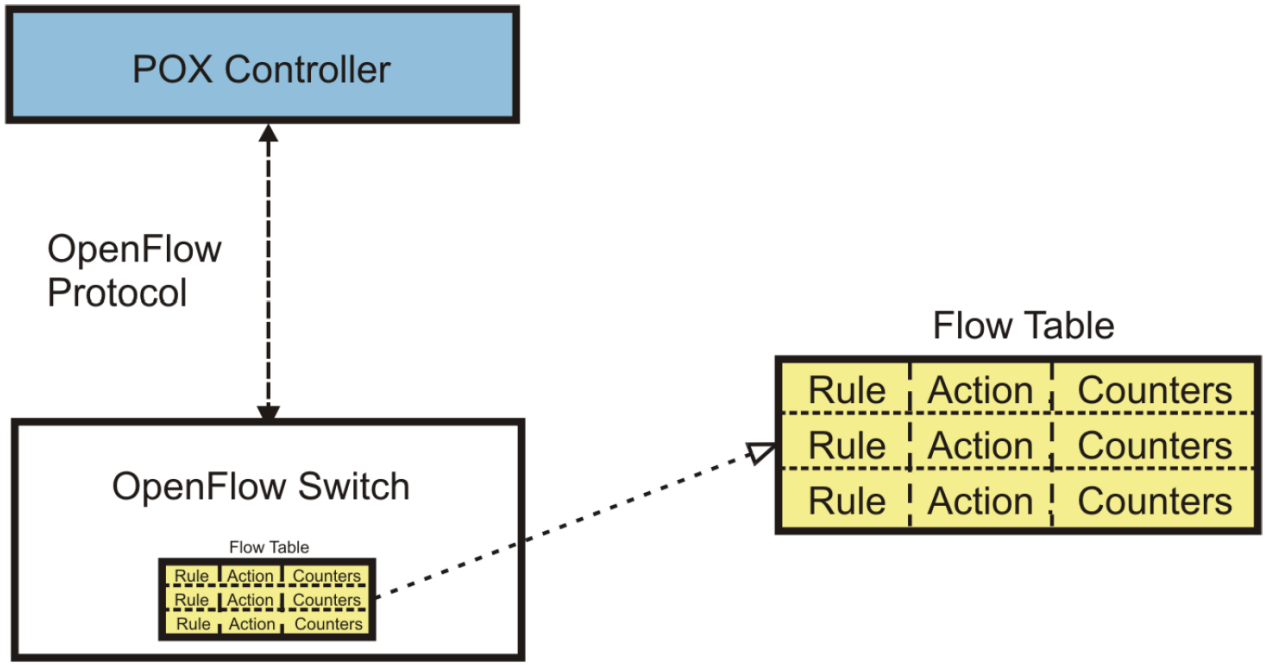


- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

Various Controllers

Name	Language	Original Developers	Description
Ovs	C	Stanford/Nicira	A reference controller. Act as a learning switch
NOX	C++	Nicira	The first OpenFlow controller
POX	Python	Nicira	Open source SDN controller
Beacon	Java	Stanford	A cross platform, modular OpenFlow controller
Maestro	Java	Rice	Network operating system
Trema	Ruby, C	NEC	A framework for developing OpenFlow controller
Floodlight	Java	BigSwitch	OpenFlow controller that work with physical and virtual OpenFlow switches
Flowvisor	C	Stanford/Nicira	Special purpose controller
Ryu	Python	NTT Labs	Ryu is a component based SDN framework
OpenDayLight	Java	ONF	It is open source project

Practical scenario



Thank you!