Software and Virtualization Technologies in Mobile Communication Networks

Comunicações Móveis

DETI – UA

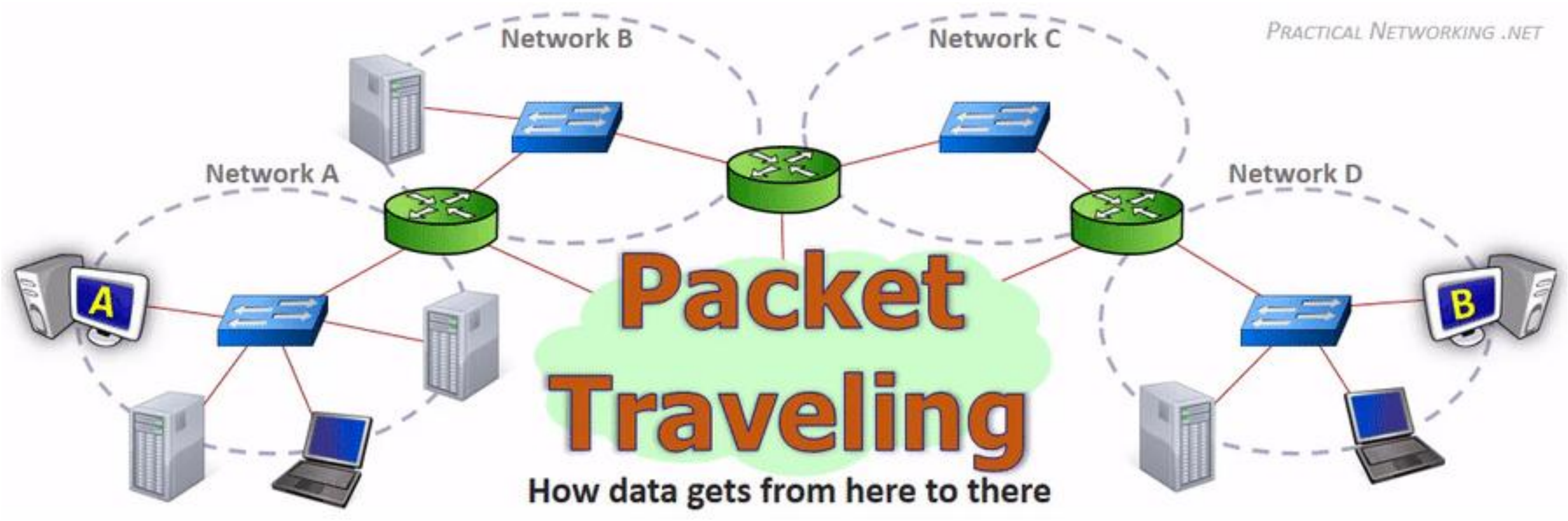
2025/2026

Outline

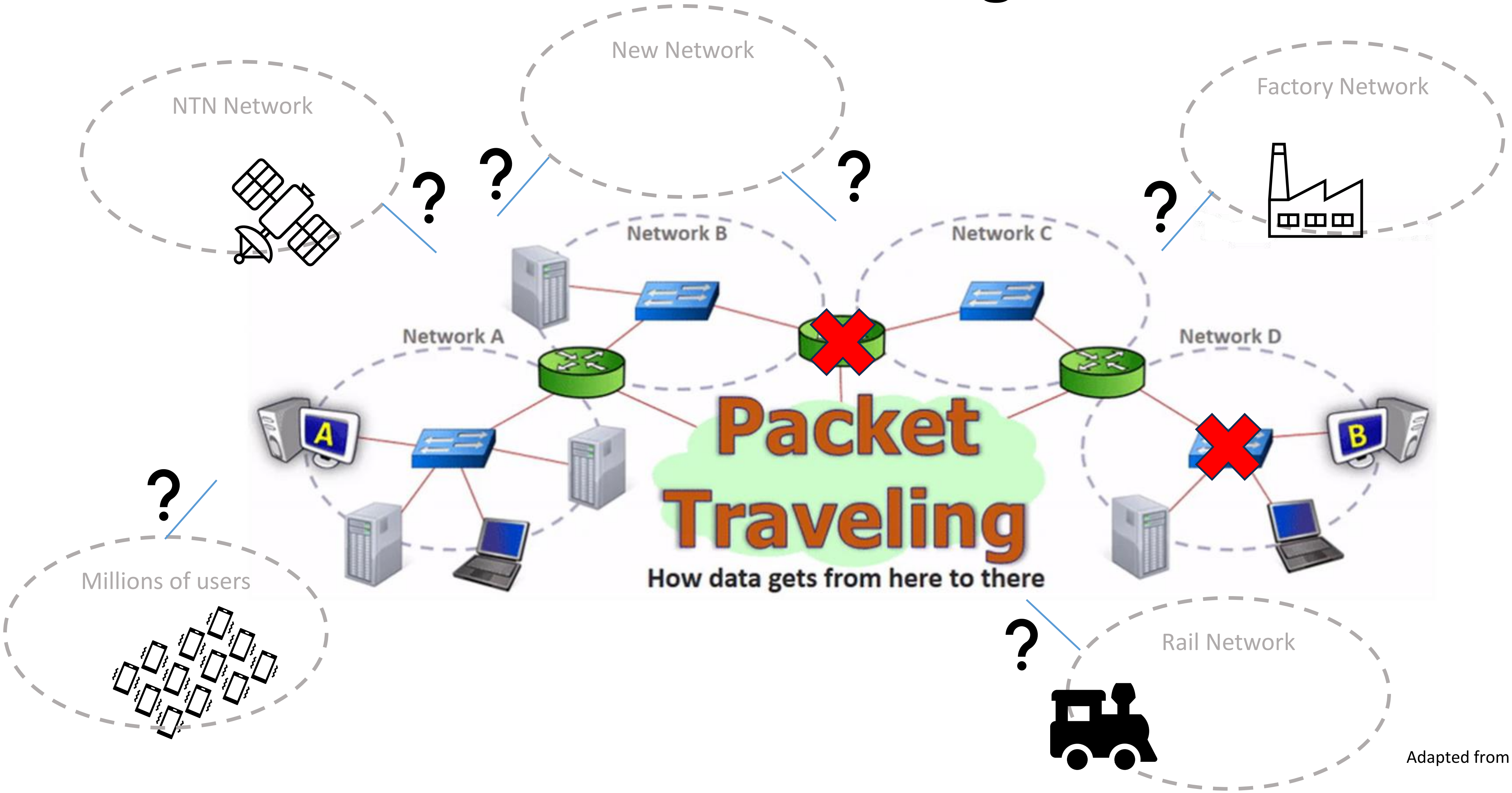
- Dynamic Networks
- Network Function Virtualization
- Management and Orchestration
- Software Defined Networking

The evolution of Networking

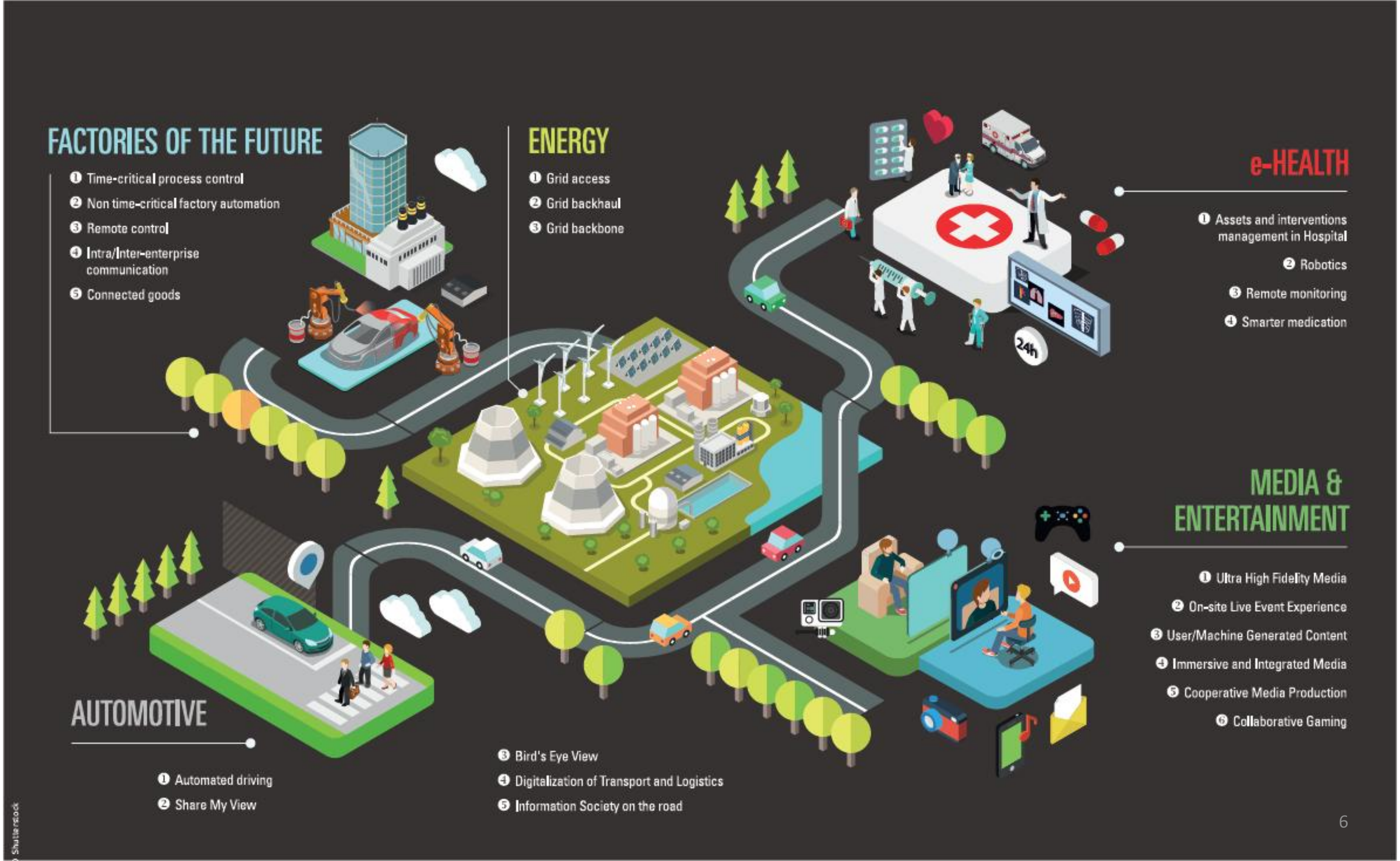
The evolution of Networking



The evolution of Networking



Adapted from practicalnetworking.net


FACTORIES OF THE FUTURE
❶ Time-critical process control
❷ Non time-critical factory automation
❸ Remote control
❹ Intra/Inter-enterprise communication
❺ Connected goods
ENERGY
❶ Grid access
❷ Grid backhaul
❸ Grid backbone
e-HEALTH
❶ Assets and interventions management in Hospital
❷ Robotics
❸ Remote monitoring
❹ Smarter medication
24h
MEDIA & ENTERTAINMENT
❶ Ultra High Fidelity Media
❷ On-site Live Event Experience
❸ User/Machine Generated Content
❹ Immersive and Integrated Media
❺ Cooperative Media Production
❻ Collaborative Gaming
AUTOMOTIVE
❶ Automated driving
❷ Share My View
❸ Bird's Eye View
❹ Digitalization of Transport and Logistics
❺ Information Society on the road
© Shutterstock

Network Function Virtualization

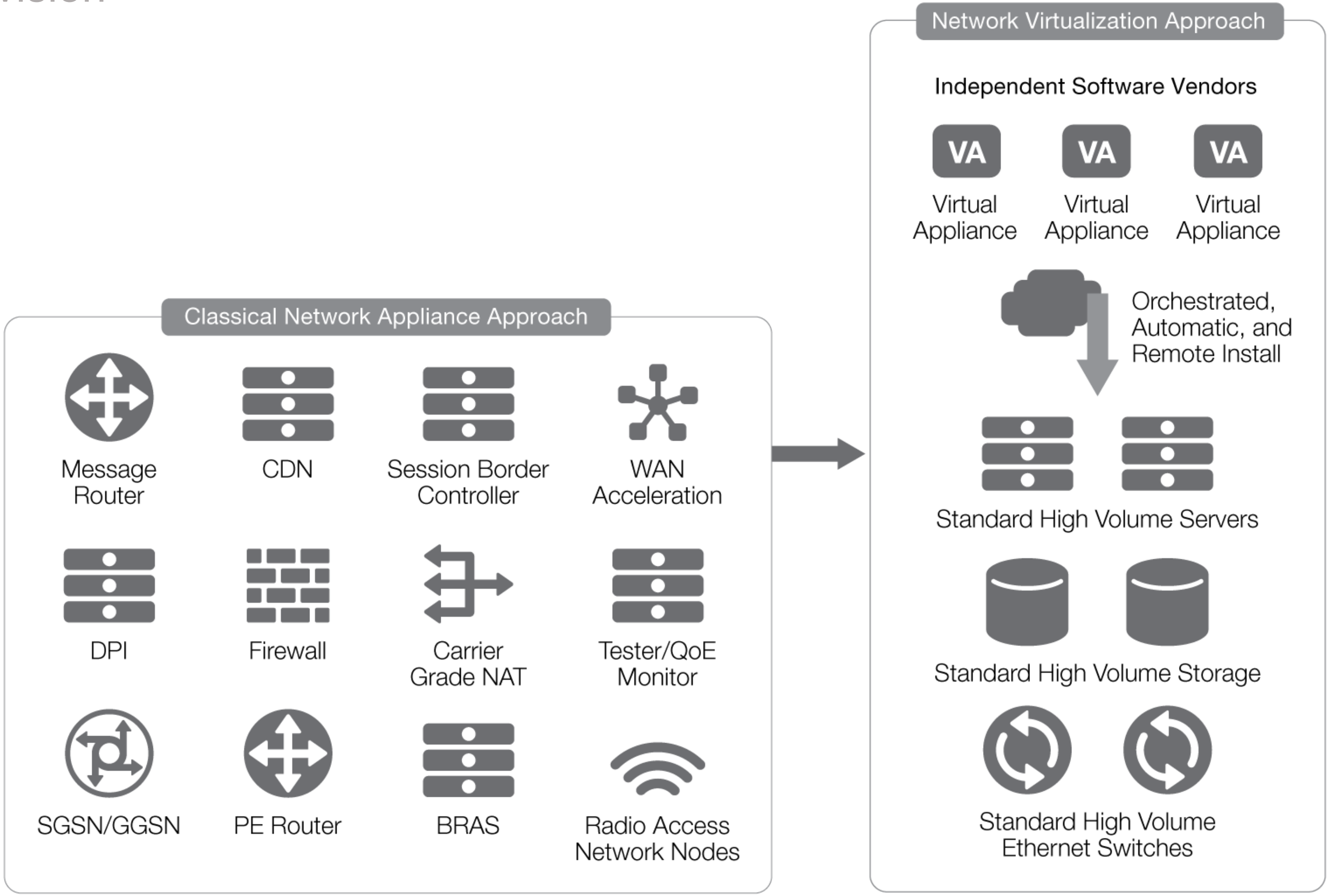
NFV

Virtualization

- 5G brought new trends
- Virtualization
  - Simulate a hardware platform, in software: VM’s, containers
    - Higher portability
    - Higher scalability
    - More cost-effective
- Virtualized networks
  - Logical software-based routers, switches, etc.
  - Network services are easier to deploy and manage
  - The physical part only needs to handle packet forwarding

Network Function Virtualization

vision



Source: https://www.f5.com/es_es/services/resources/white-papers/network-functions-virtualization-everything-old-is-new-again

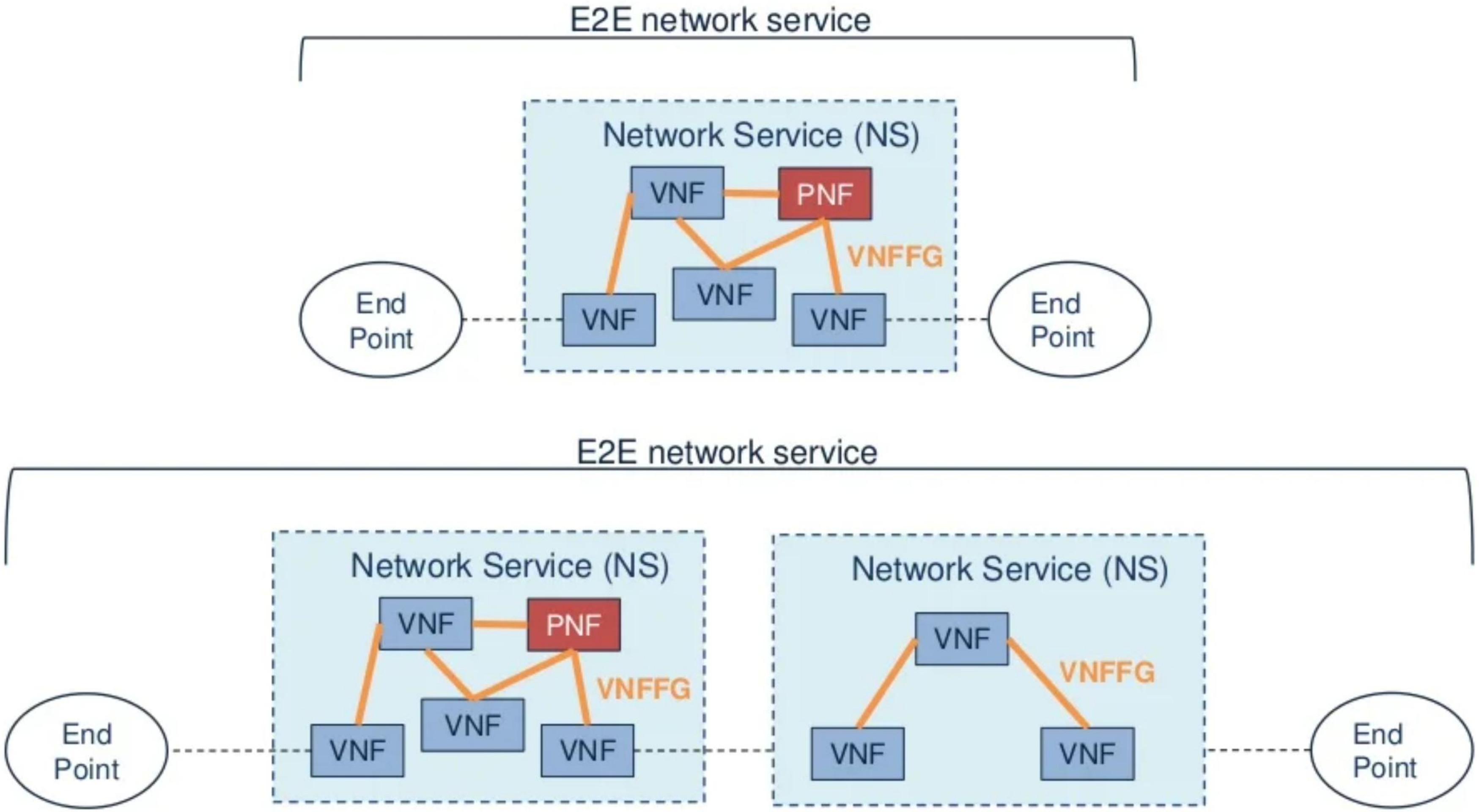
Economy of Scale and Flexibility

- Economy of scale
  - CAPEX – Capital Expenditure
    - <CAPEX → Less investment amounts in infrastructure
      - Less dedicated hardware
  - OPEX – Operational Expenditure
    - <OPEX → Lower costs in operating the infrastructure
      - Less upgrades
      - Less licenses
      - Less air conditioned
      - Less technicians
      - …

Network Services

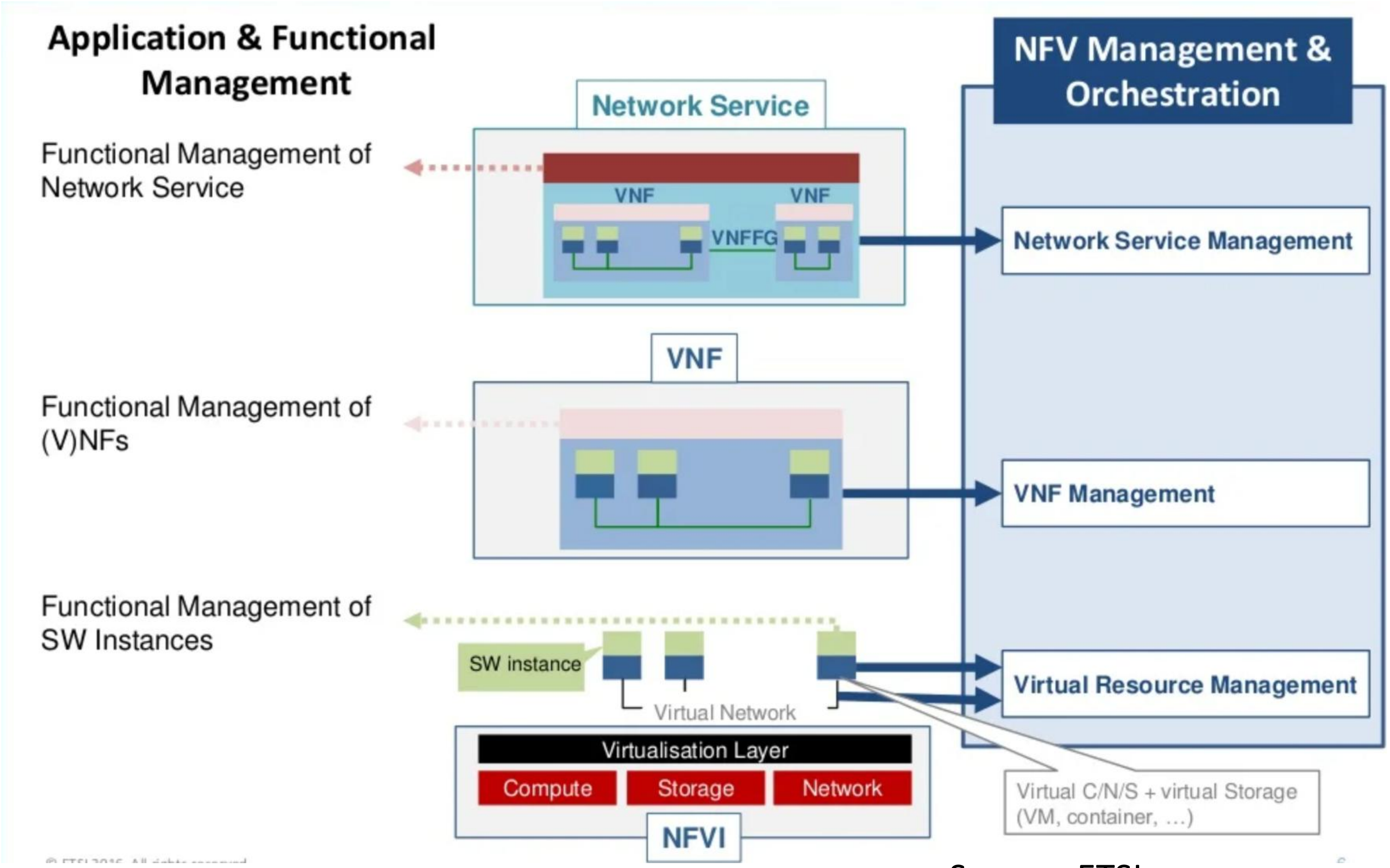
- A set of chained VNFs
- VNFs are interconnected and need to communicate with each other
  - Connectivity Forwarding Graph (FG) indicates how they are connected
    - VNFFG – VNF Forwarding Graph
- Can connect to physical network functions
  - Can also be part of the forwarding graph
- VNFs+VNFFG = Network Service
  - Is managed and orchestrated together
  - Lifecycle mgmt for VNFs
  - Management of the VNFFG and NS lifecycle mgmt is going to orchestrate across the whole lifecycle mgmt of those VNFs that form the NS
  - The NS defines some external connection points that can be connected to the end-dpoints, defining na end-to-end service.
  - We can concatenate differente NS to for an end-2-end network service

Network Services



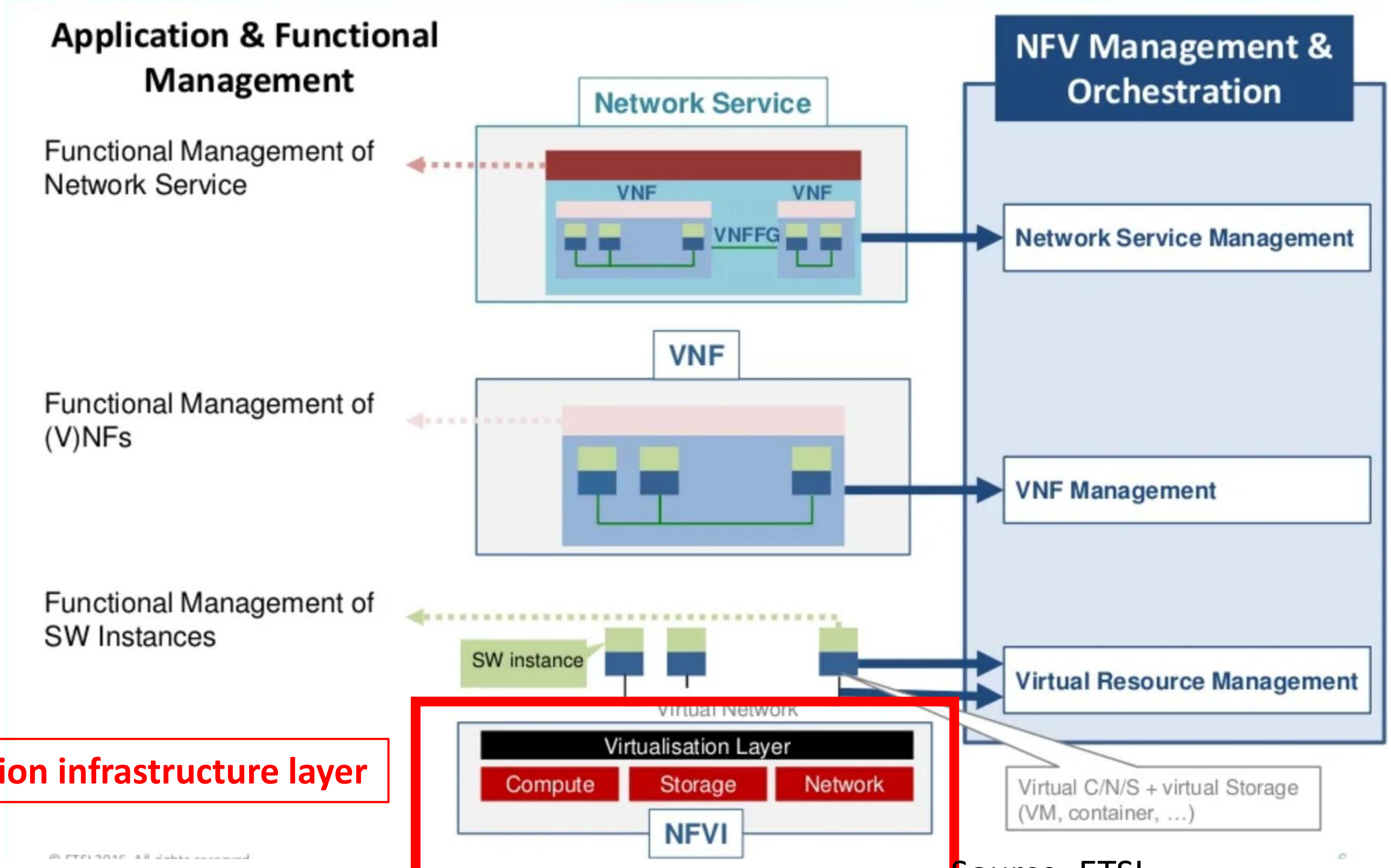
Source: ETSI

Overall Concepts and how they interconnect



Source: ETSI

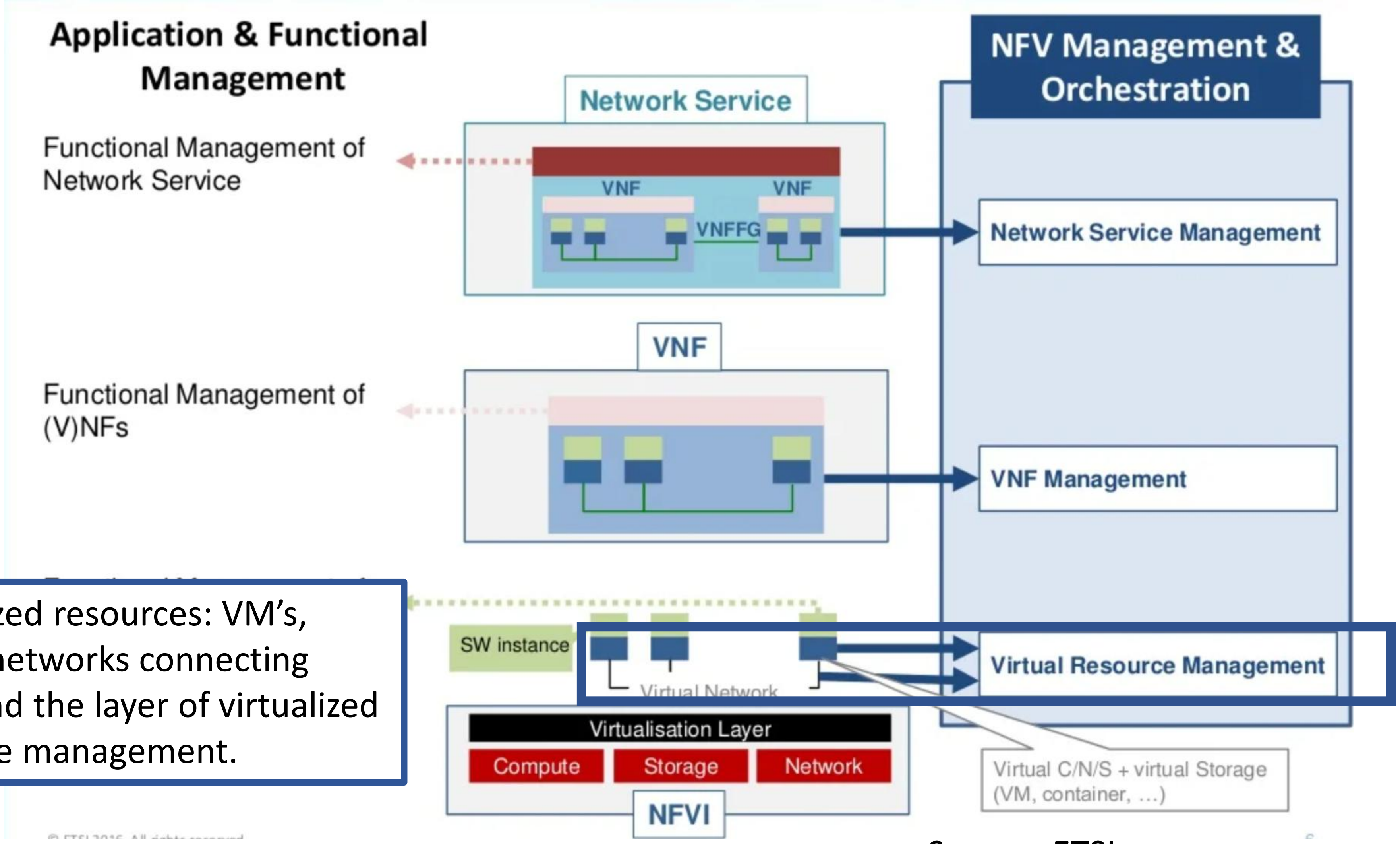
Overall Concepts and how they interconnect



Virtualization infrastructure layer

Source: ETSI

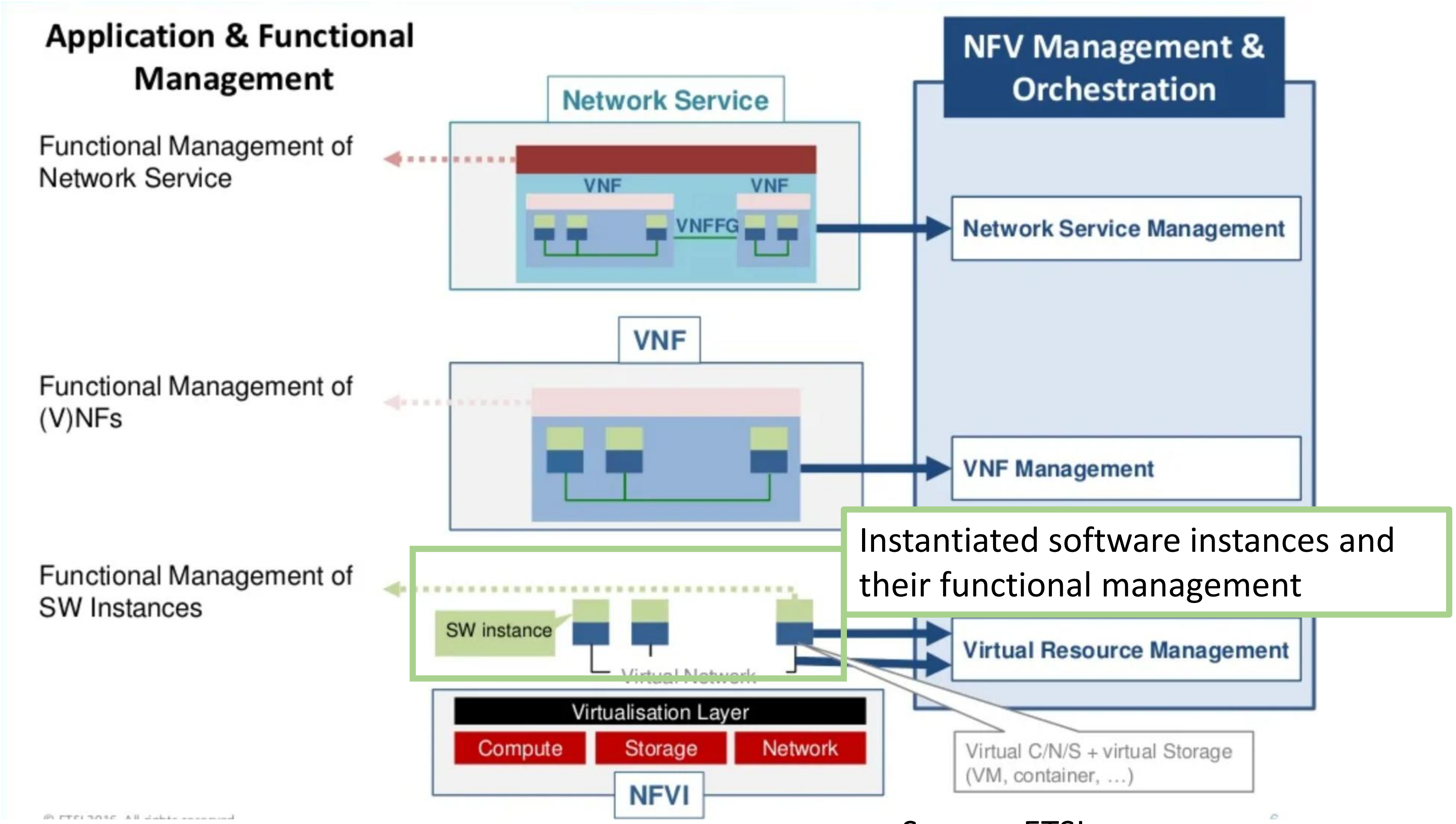
Overall Concepts and how they interconnect



Virtualized resources: VM's, virtual networks connecting VMs, and the layer of virtualized resource management.

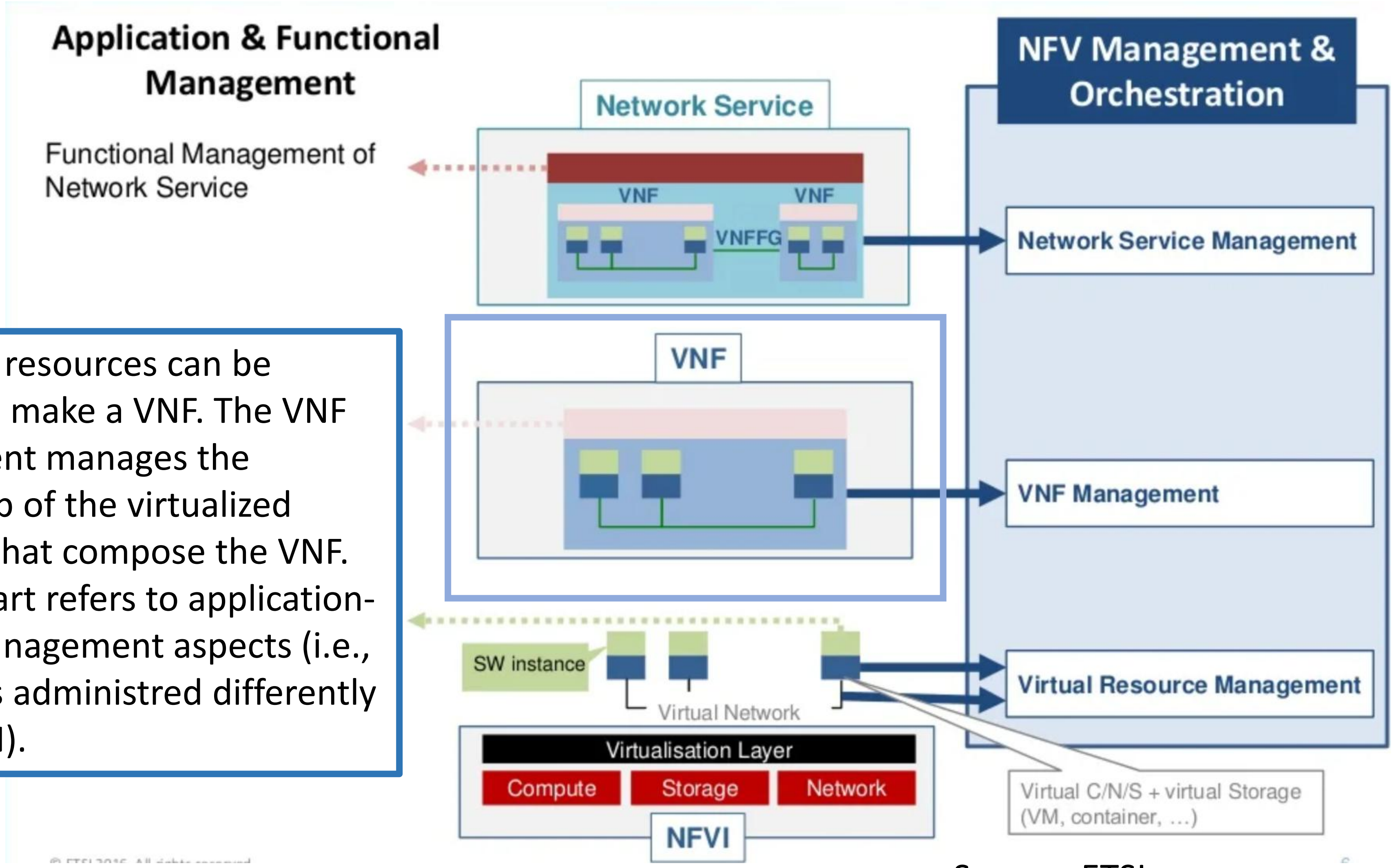
Source: ETSI

Overall Concepts and how they interconnect



Source: ETSI

Overall Concepts and how they interconnect

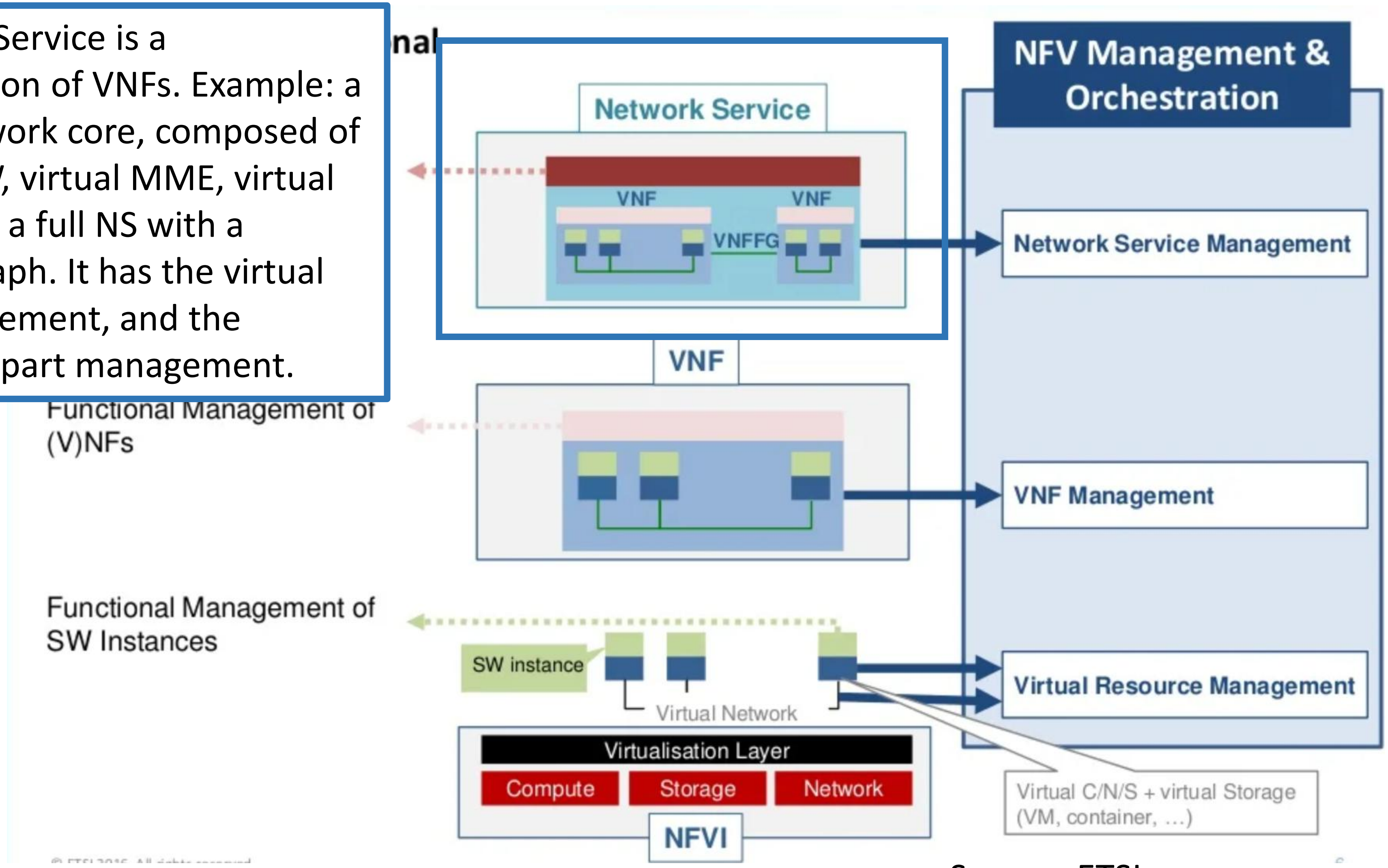


Virtualized resources can be grouped to make a VNF. The VNF management manages the relationship of the virtualized resources that compose the VNF. The pink part refers to application-specific management aspects (i.e., a firewall is administred differently than a CDN).

Source: ETSI

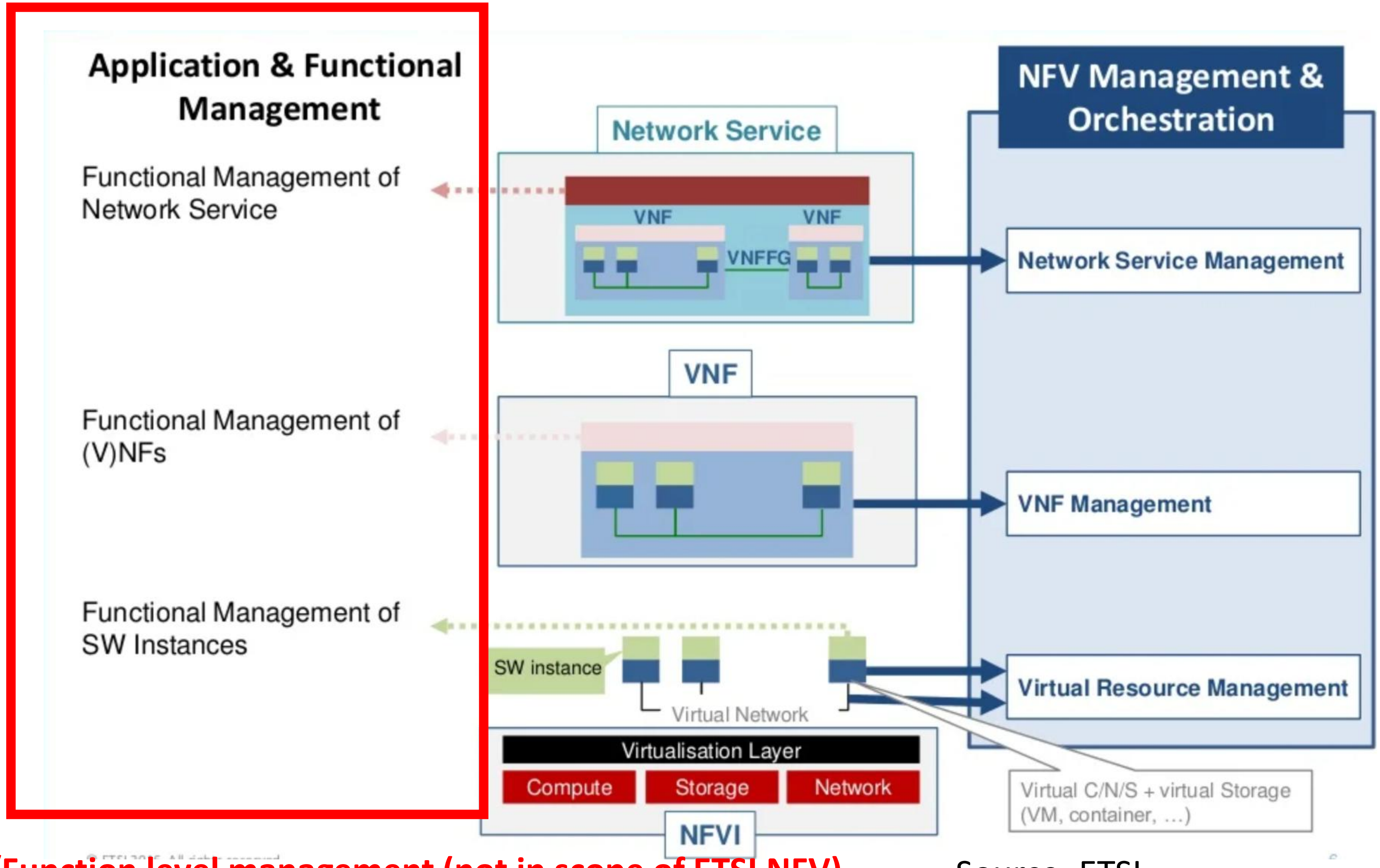
Overall Concepts and how they interconnect

A Network Service is a concatenation of VNFs. Example: a virtual network core, composed of virtual PGW, virtual MME, virtual SGW, forms a full NS with a network graph. It has the virtual part management, and the application part management.



Source: ETSI

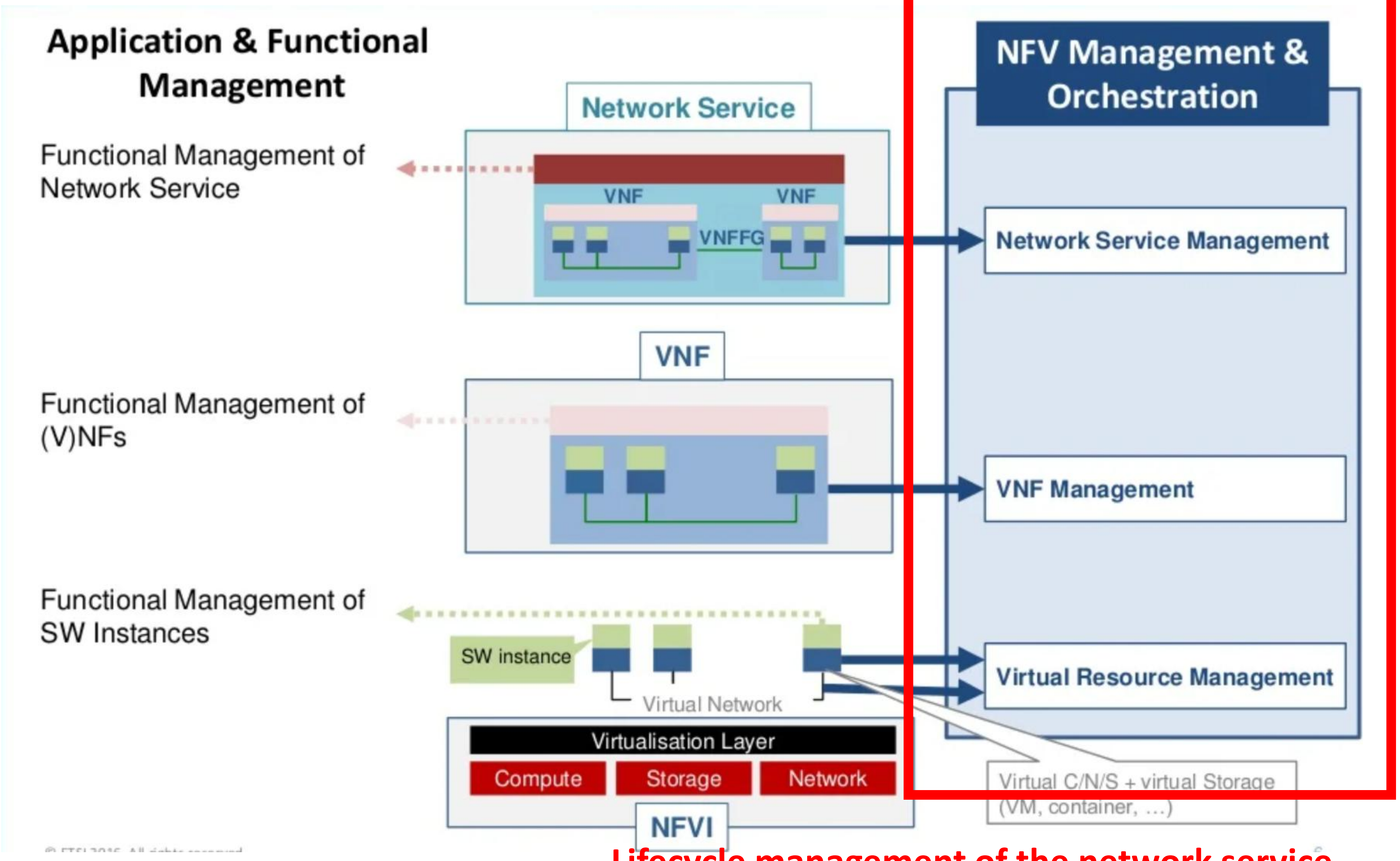
Overall Concepts and how they interconnect



Application/Function level management (not in scope of ETSI NFV)

Source: ETSI

Overall Concepts and how they interconnect

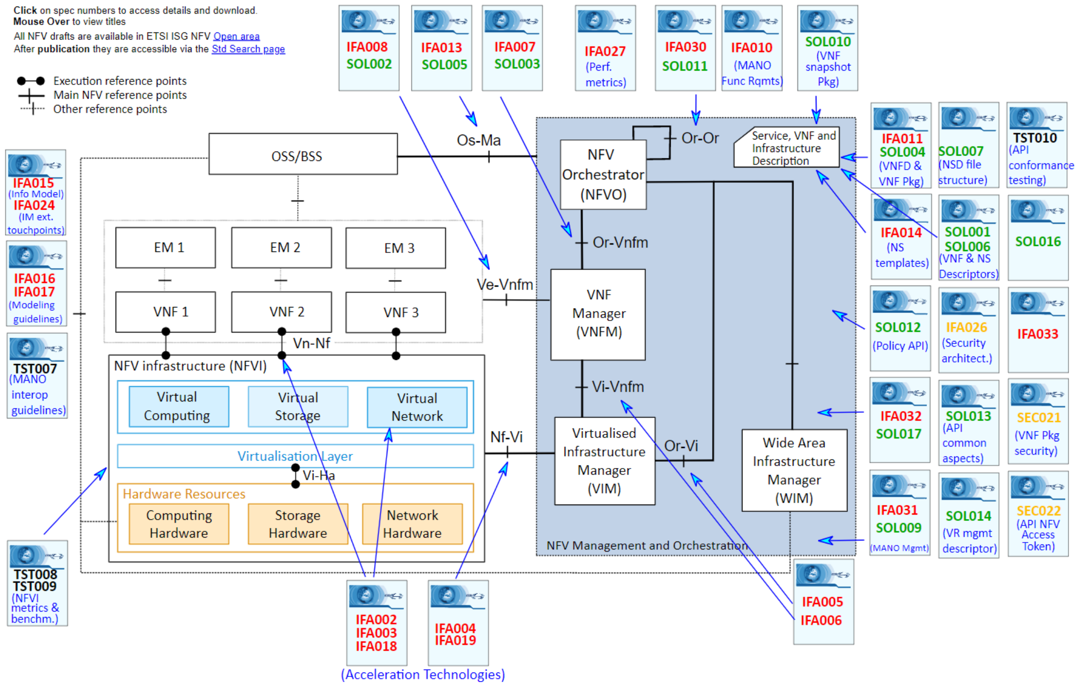


Source: ETSI

Lifecycle management of the network service

NFV Standardisation

- ETSI NFV research groups
  - TST – Testing
  - SOL – Protocols and Data models
  - REL – Reliability
  - IFA – Interfaces and Architecture
  - EVE – Evolution and Ecossystem
  - SEC – Security



- Clickable architecture:

https://www.etsi.org/images/articles/NFV_Architecture.svg

VIM – Virtual Infrastructure Managers

- OpenStack
  - https://www.openstack.org/
- Vmware vCloud Director
  - https://www.vmware.com/products/cloud-director.html
- Amazon Web Services
- Microsoft Azure
- Google Cloud Platform
- Sandboxes
  - DevStack
    - https://docs.openstack.org/devstack/latest/
  - MicroStack
    - https://opendev.org/x/microstack

Management and Orchestration

MANO

Virtualized Network Services

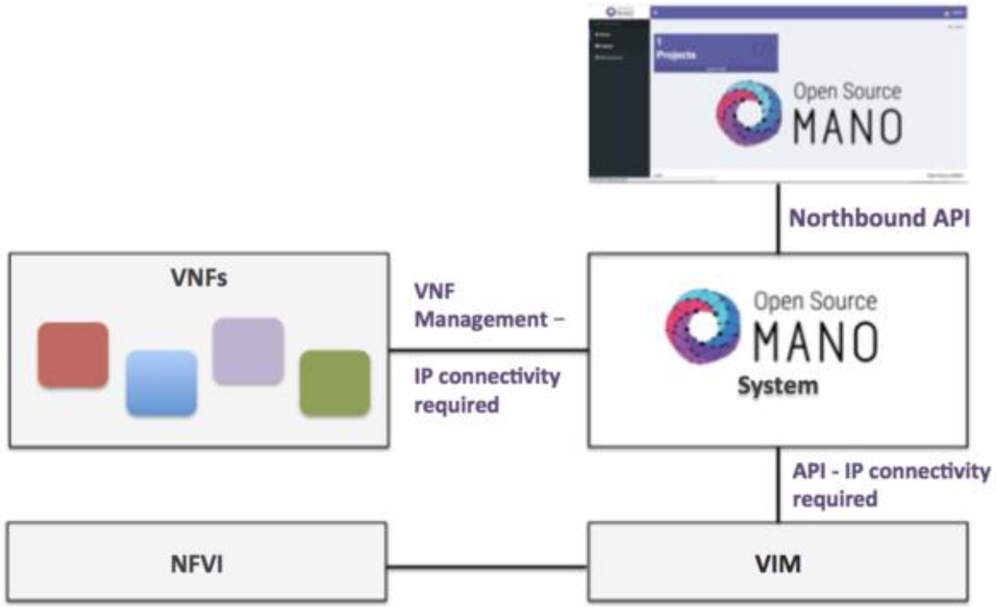
- Managing and coordinating resources and networks needs a special entity
  - NFV Orchestrator
- A powerful tool
  - Spans large numbers of networks
  - … software elements
  - … hardware platforms
- Needs to be able to work with many different standards

ETSI NFV MANO

- Framework for managing and orchestrating all resources in the cloud datacenter
  - Includes computing, networking, storage and VM resources
- Objective
  - Allow flexible on-boarding of network services
  - Handle network componentes spin-up
- The standard composes 3 elements
  - NFVO – NFV Orchestrator
  - VNF Manager – VNFM
  - Virtualized Infrastructure Manager – VIM

ETSI OSM – Open Source MANO

- Open source NFV Management and Orchestration stack
- Uses well established open source tools and working procedures
- Complements the standardisation work



https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html

ONAP – Open Network Automation

- https://www.onap.org/
- Also Open Source
- More than just MANO
- Common platform for end-to-end servisse and infrastructure management and provisioning
- Support from major vendors
- Supports TOSCA and YANG unified design specifications

TOSCA - Topology and Orchestration Specification for Cloud Applications

- facilitate the creation cloud applications and services
- provides mechanisms to control workflow, describe relationships and reflect dependencies between resources
- in the cloud, TOSCA can describe the creation of a web hosting service where a webserver on multiple hosts is connected to a database and includes networking functions such as DNS, load balancer and firewall, which all need to be configured.

TOSCA EXAMPLE

- TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0

YANG

- data modeling language used to describe configuration and state information
- used to model networking devices and services – i.e., an object and its attributes
- defines the data models that are then manipulated through the NETCONF protocol.
  - a device such as a router can be modeled in YANG
  - The router can then be configured via NETCONF

YANG

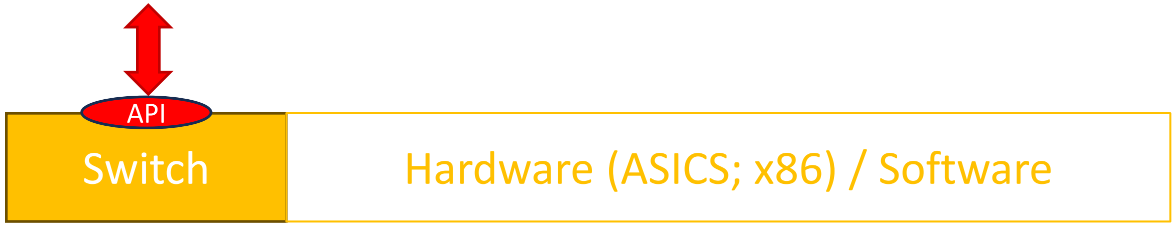
```
module: ietf-interfaces
   +--rw interfaces
   |  +--rw interface* [name]
   |     +--rw name                        string
   |     +--rw description?                string
   |     +--rw type                        identityref
   |     +--rw enabled?                    boolean
   |     +--rw link-up-down-trap-enable?   enumeration
   +--ro interfaces-state
      +--ro interface* [name]
         +--ro name               string
         +--ro type               identityref
         +--ro admin-status       enumeration
         +--ro oper-status        enumeration
         +--ro last-change?       yang:date-and-time
         +--ro if-index           int32
         +--ro phys-address?      yang:phys-address
         +--ro higher-layer-if*   interface-state-ref
         +--ro lower-layer-if*    interface-state-ref
         +--ro speed?             yang:gauge64
         +--ro statistics
            +--ro discontinuity-time    yang:date-and-time
            +--ro in-octets?            yang:counter64
            +--ro in-unicast-pkts?      yang:counter64
            +--ro in-broadcast-pkts?    yang:counter64
            +--ro in-multicast-pkts?    yang:counter64
            +--ro in-discards?          yang:counter32
            +--ro in-errors?            yang:counter32
            +--ro in-unknown-protos?    yang:counter32
            +--ro out-octets?           yang:counter64
            +--ro out-unicast-pkts?     yang:counter64
            +--ro out-broadcast-pkts?   yang:counter64
            +--ro out-multicast-pkts?   yang:counter64
            +--ro out-discards?         yang:counter32
            +--ro out-errors?           yang:counter32
```

Software Defined Networking

SDN

Software Defined Networking

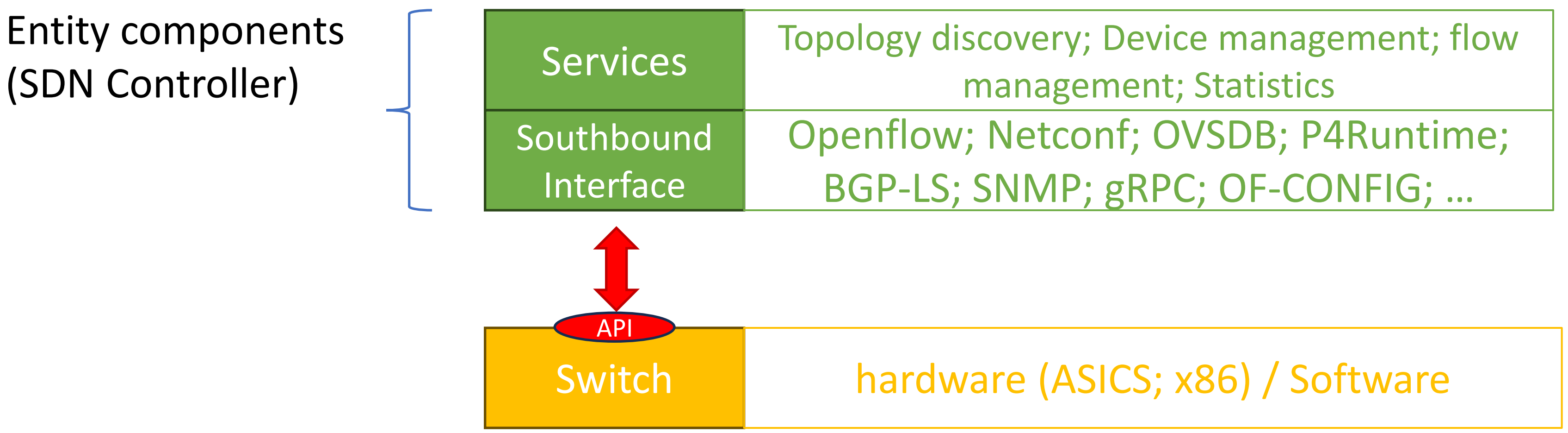
- Network switches have an API



ASICS: Application-Specific Integrated Circuits

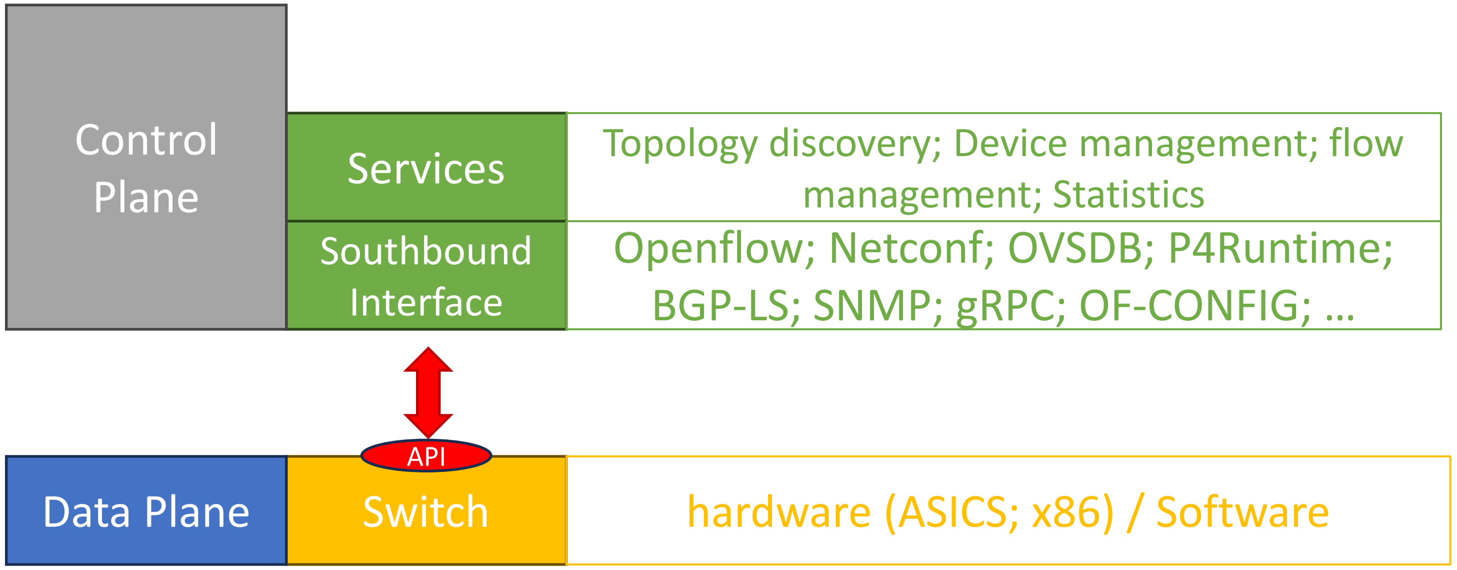
Software Defined Networking

- Allows other entities to use that API to interact with the networking infrastructure and direct traffic



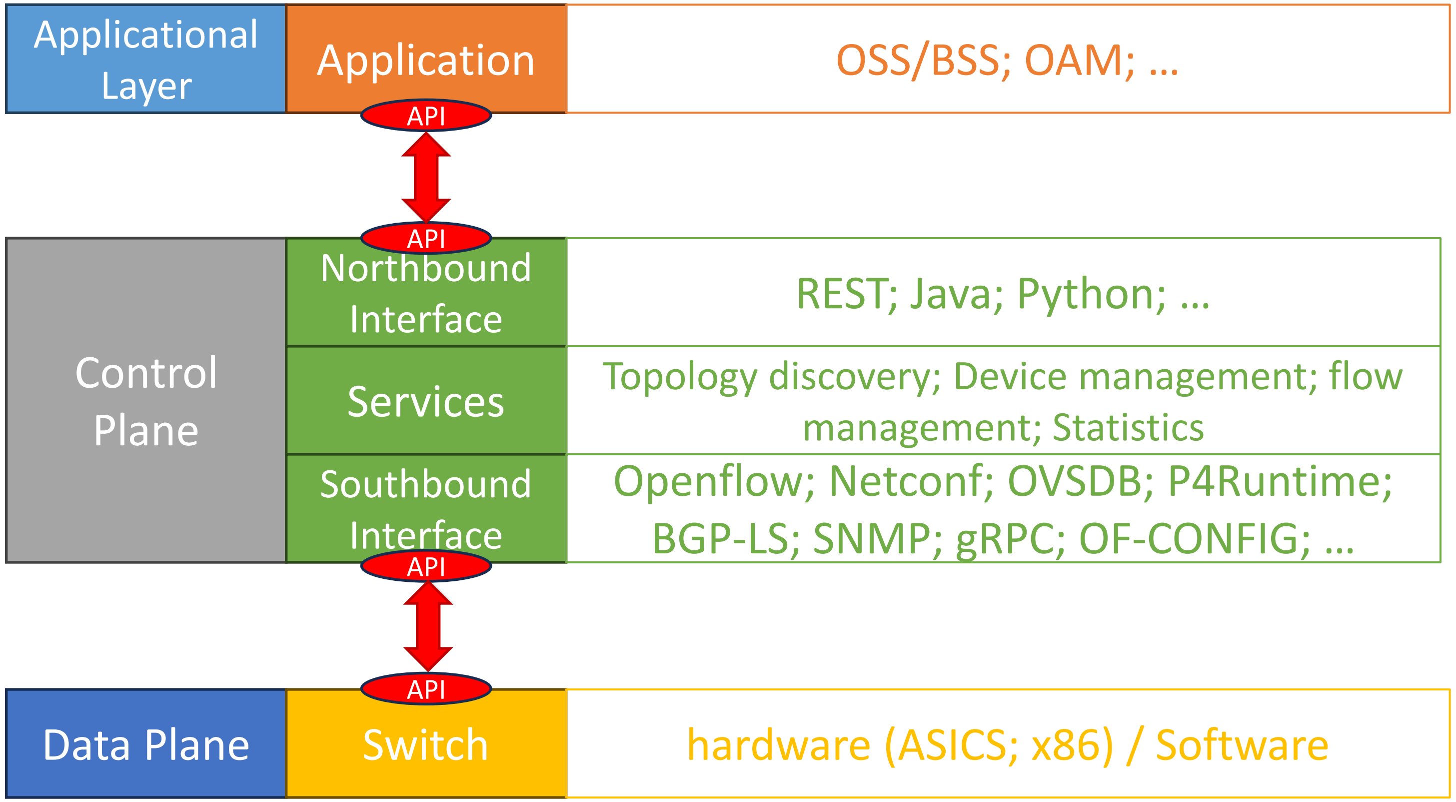
Software Defined Networking

- Separates the control plane
  - Network part which decides where the traffic needs to be sent
- … from the dataplane
  - Network part which forwards traffic towards its destiny



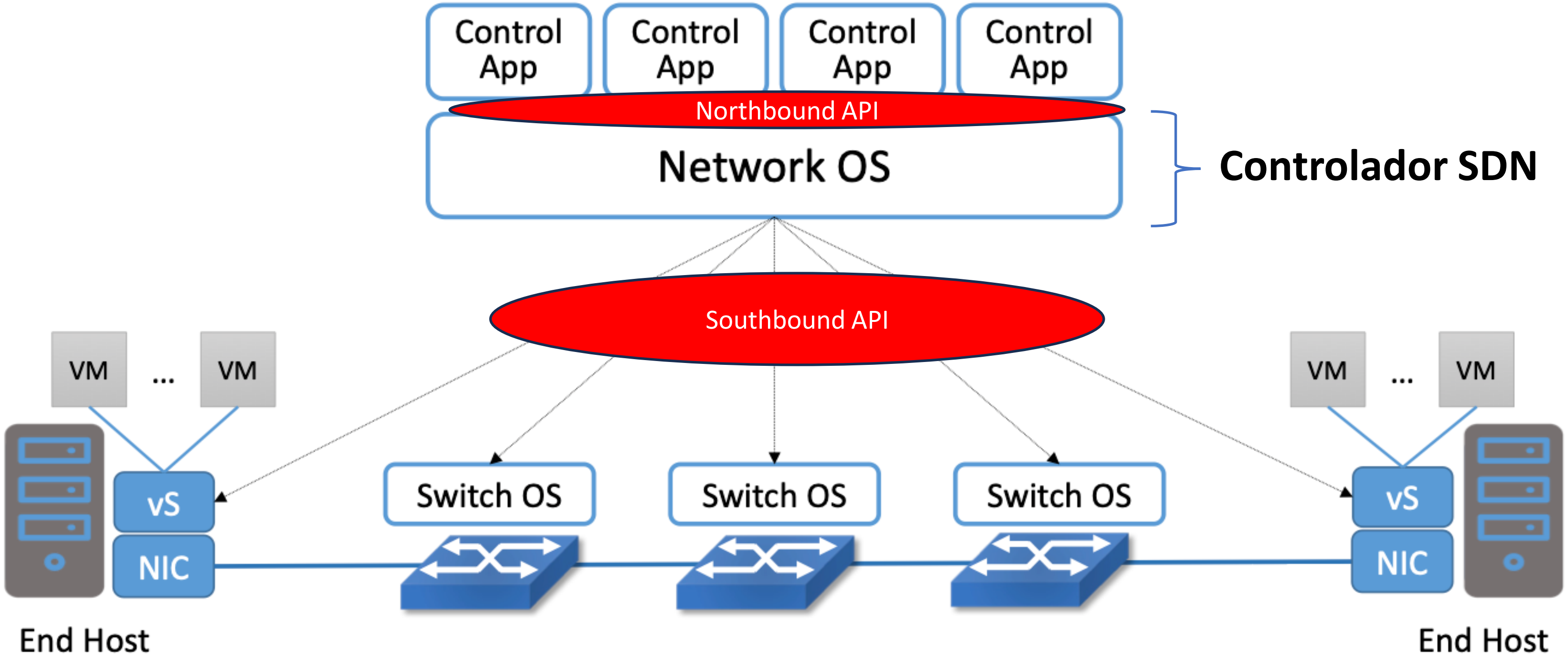
Software Defined Networking

- Applications (Business Domain) use the APIs to control the network



Software Defined Networking

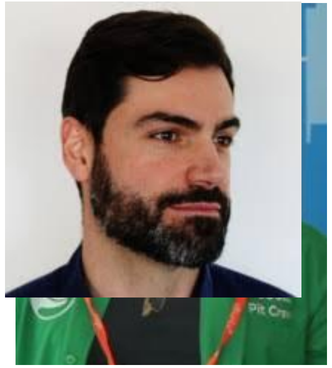
- “Centralized management of policy creation and their deployment into multiple types of network devices via programmable mechanisms” – Mark Darnell (SUSE)

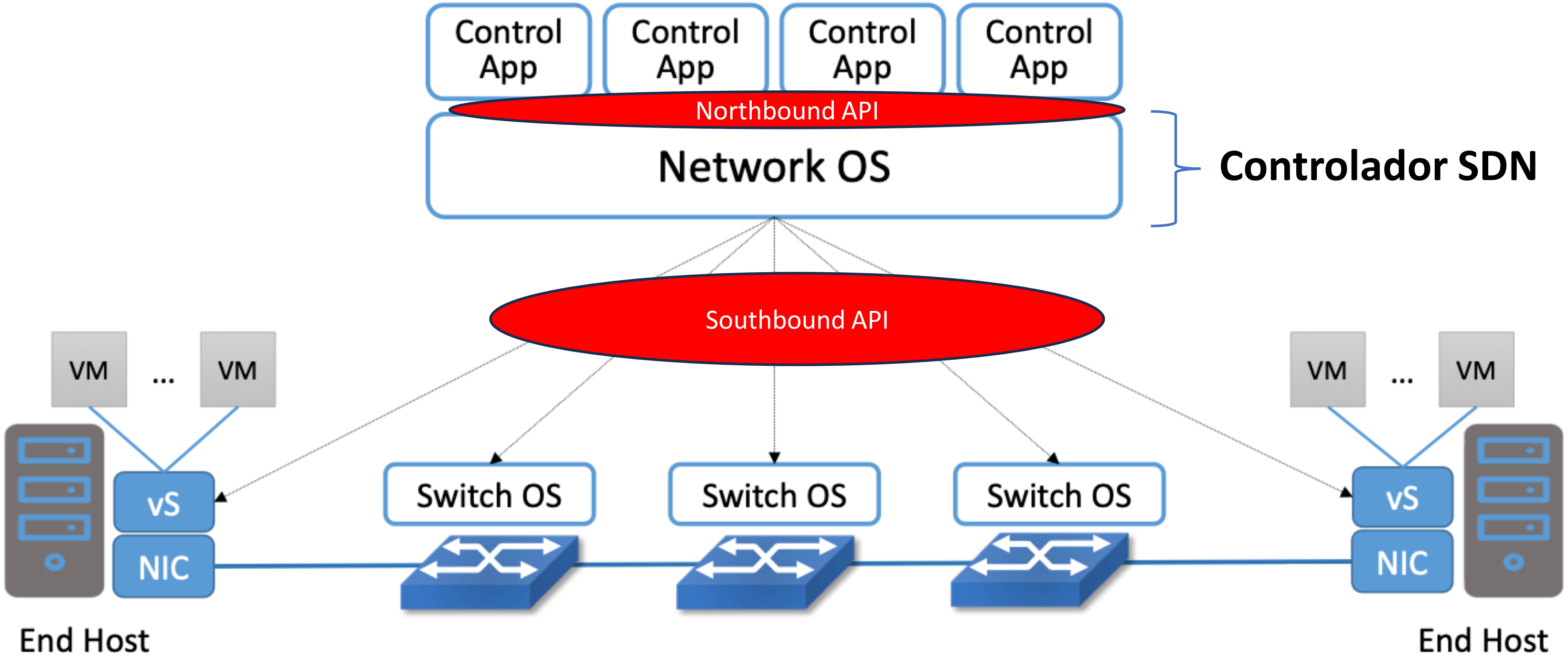


Source: Software-Defined Networks: A systems Approach

Redes Definidas por Software

- “Centralized ***<<and dynamic>>*** management of policy creation and their deployment into multiple types of network devices via programmable mechanisms” – Mark Darnell (SUSE)



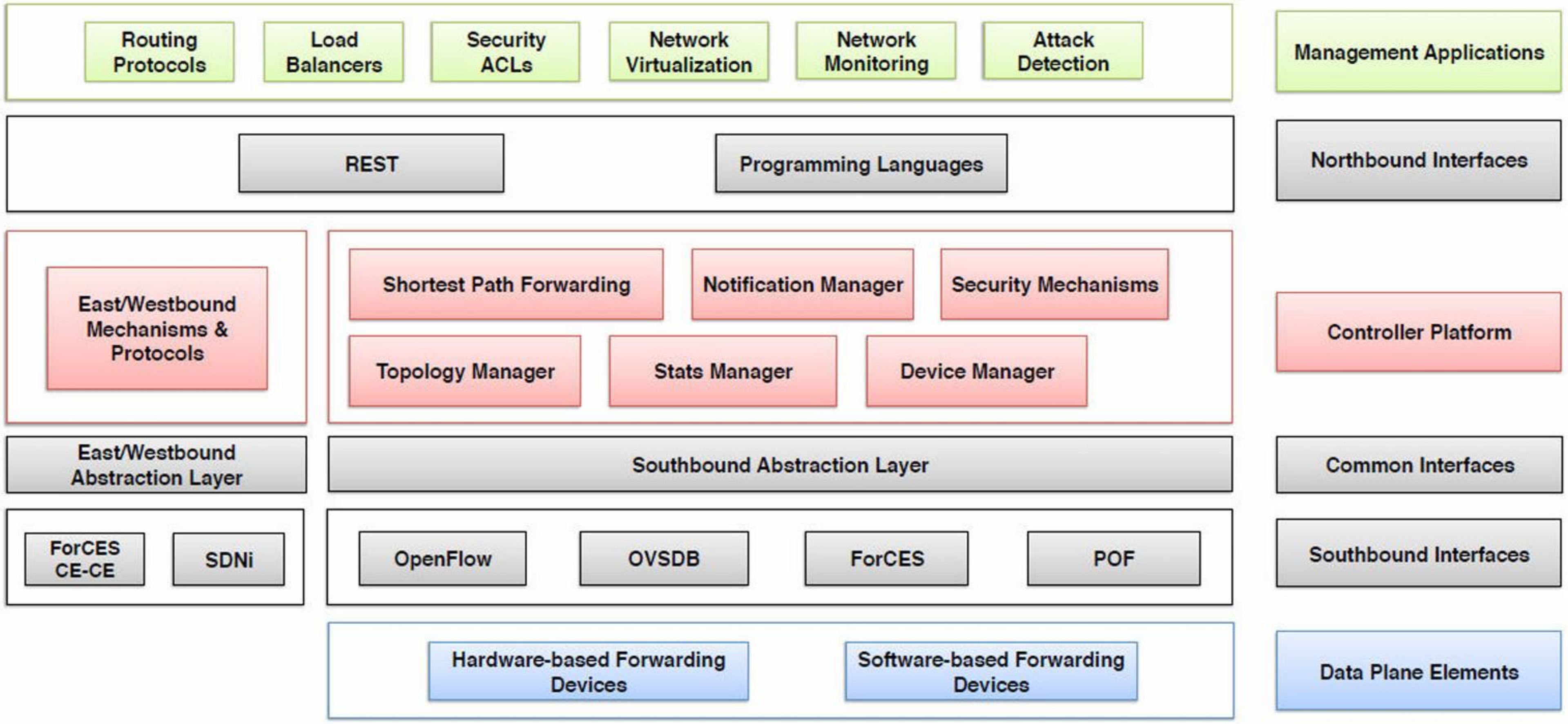


Source: Software-Defined Networks: A systems Approach

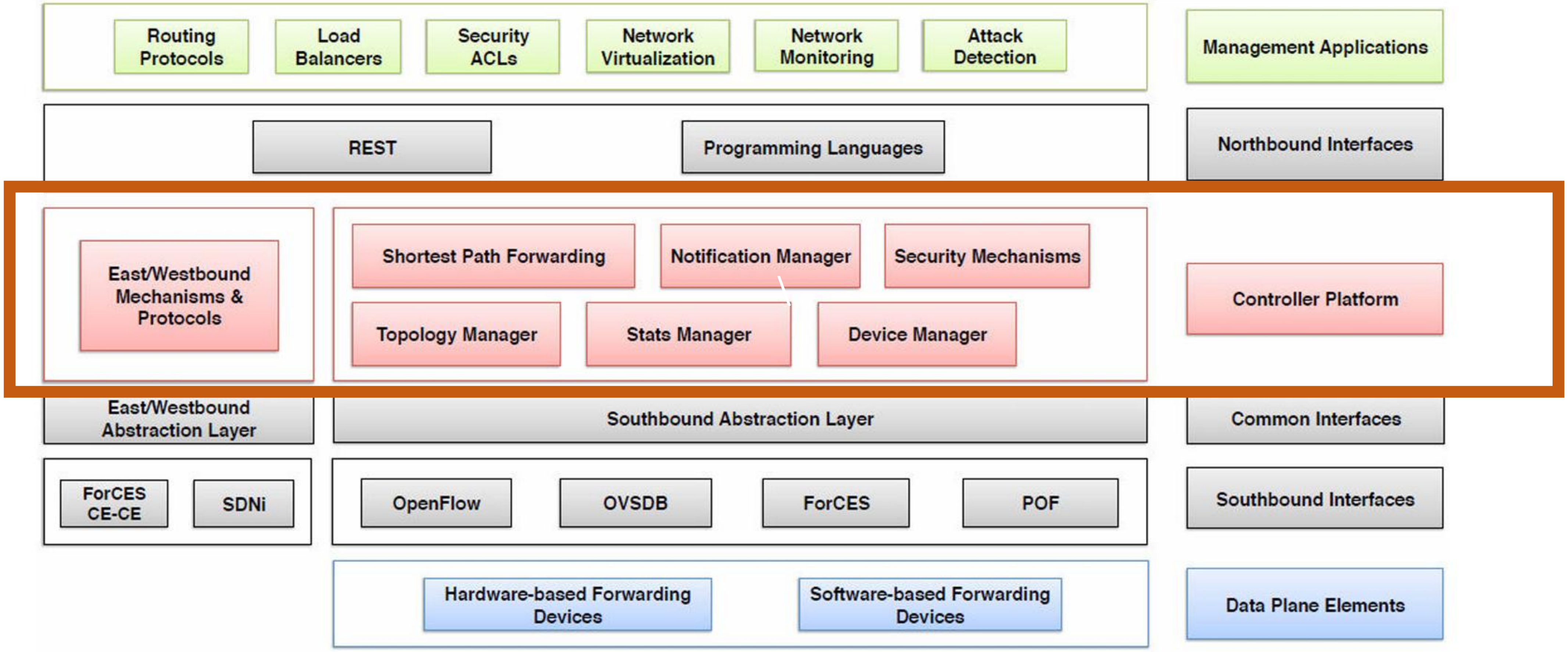
Software Defined Networking

- Objective
  - A tool to enable a higher degree of control over network devices and traffic flow
- Main aspects
  - Control plane is separated from the device implementing the data plane
  - A single control plane is bale to manage multiple network devices
- Initial deployments
  - Universities: to try out radical new protocols in parallel with existing traffic
  - Busy data centres: overcome the VLAN ID tag limit (4095)
- Protocols:
  - OpenFlow (first)
  - P4 (now)

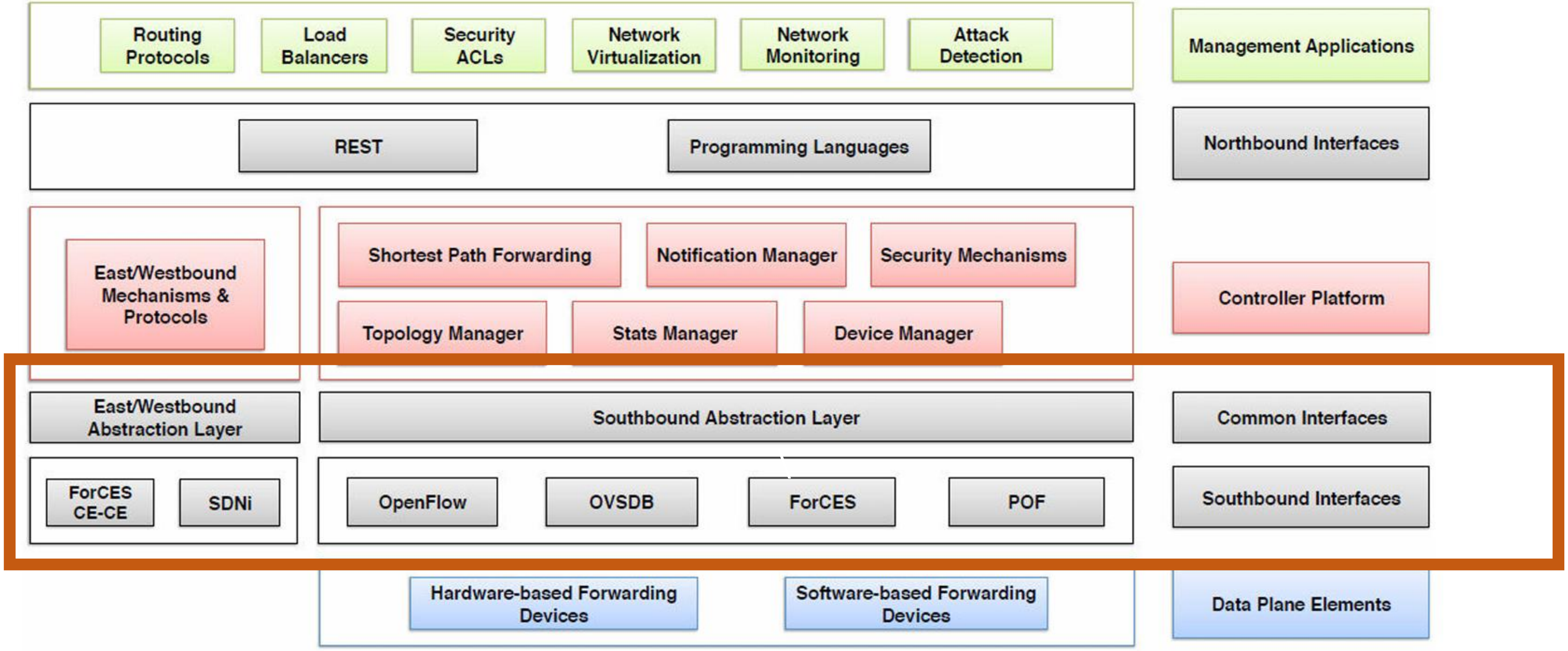
SDN Architecture



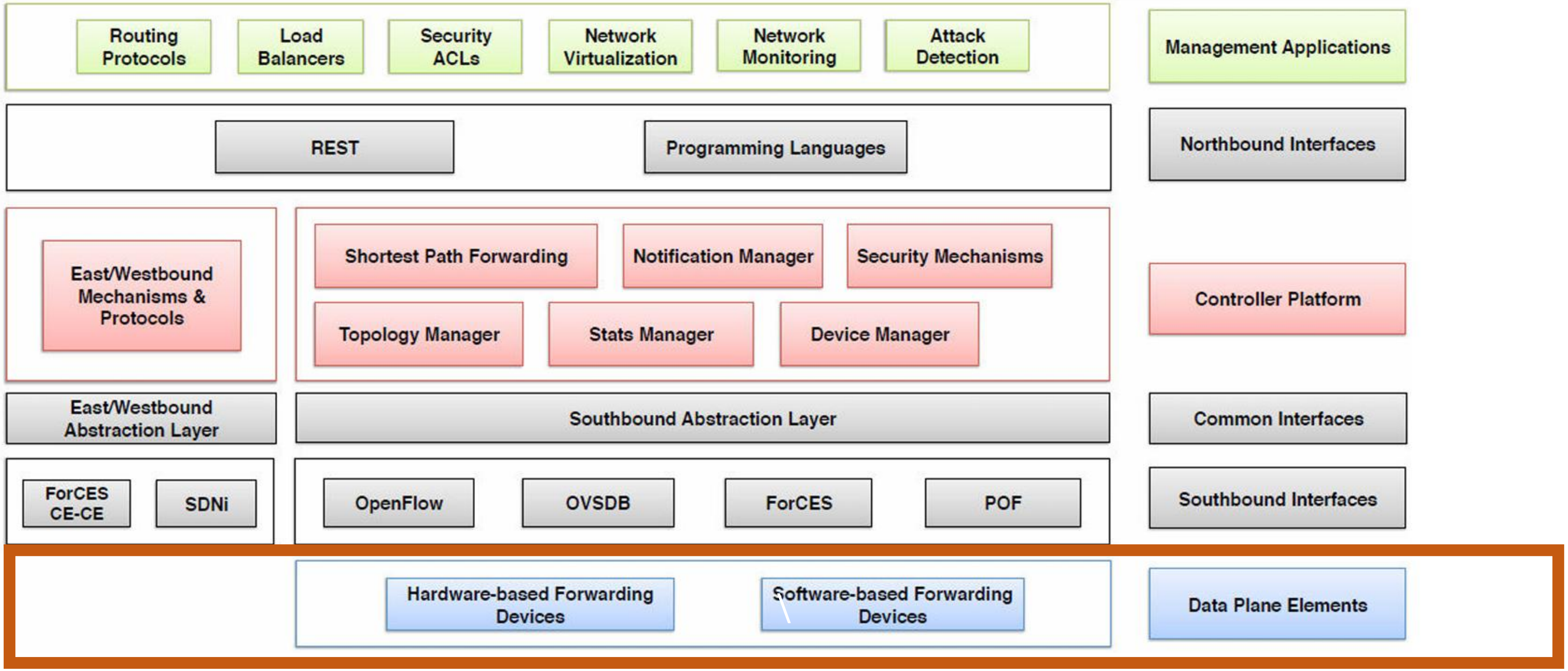
SDN Architecture - Controller



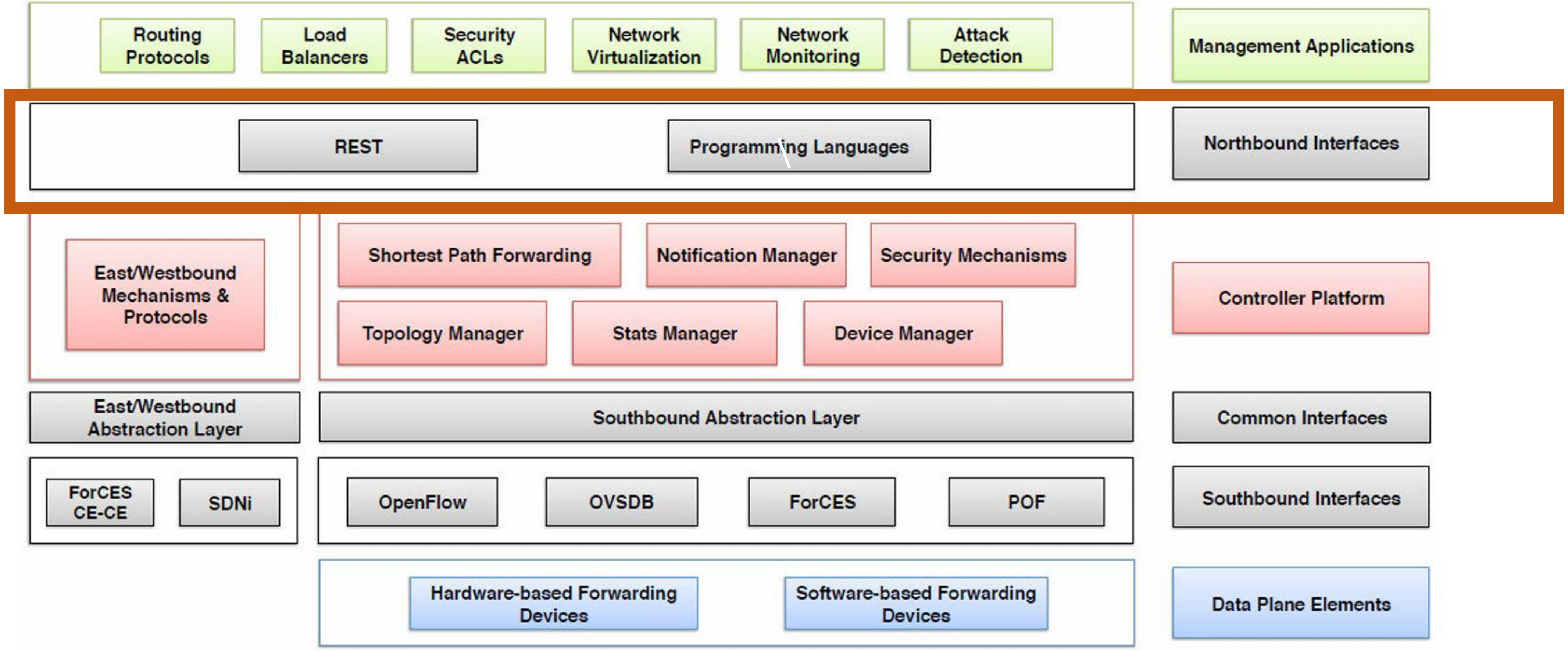
SDN Architecture – Southbound Interface



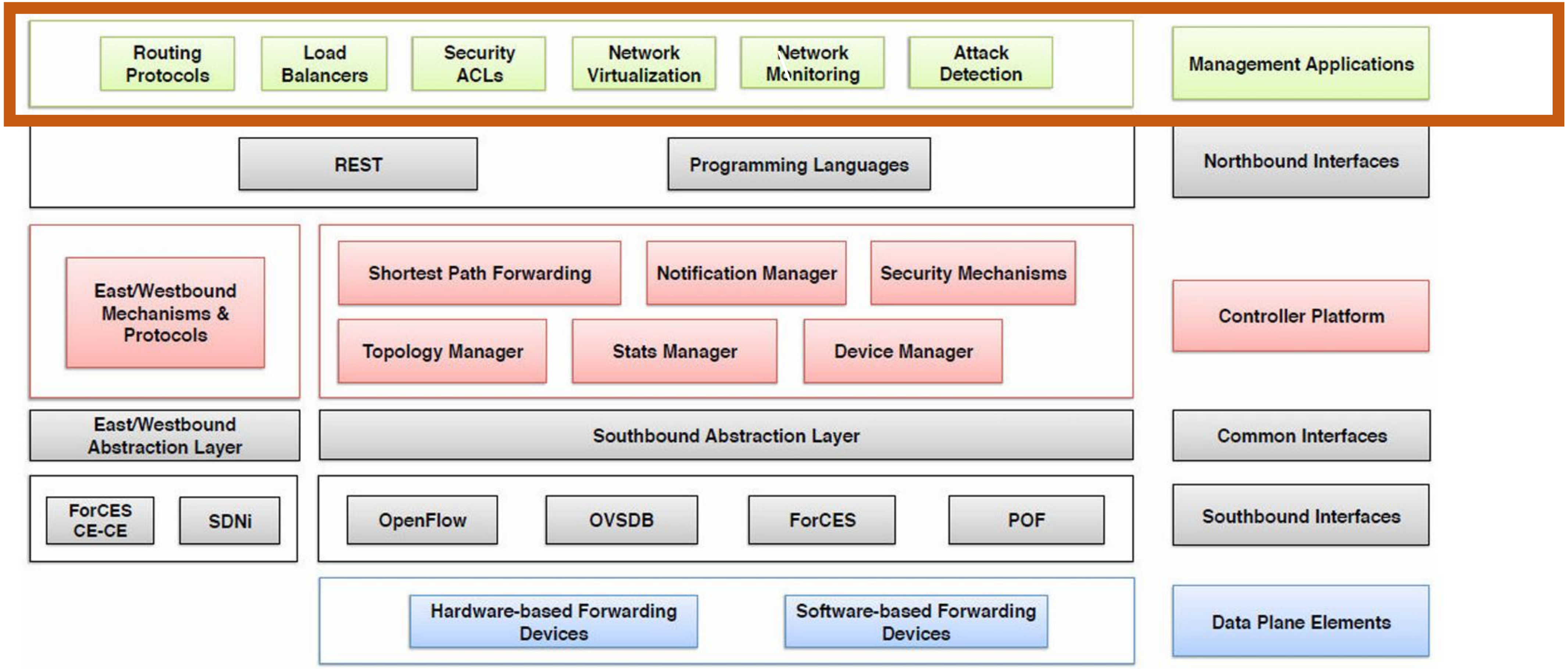
SDN Architecture – Dataplane Devices



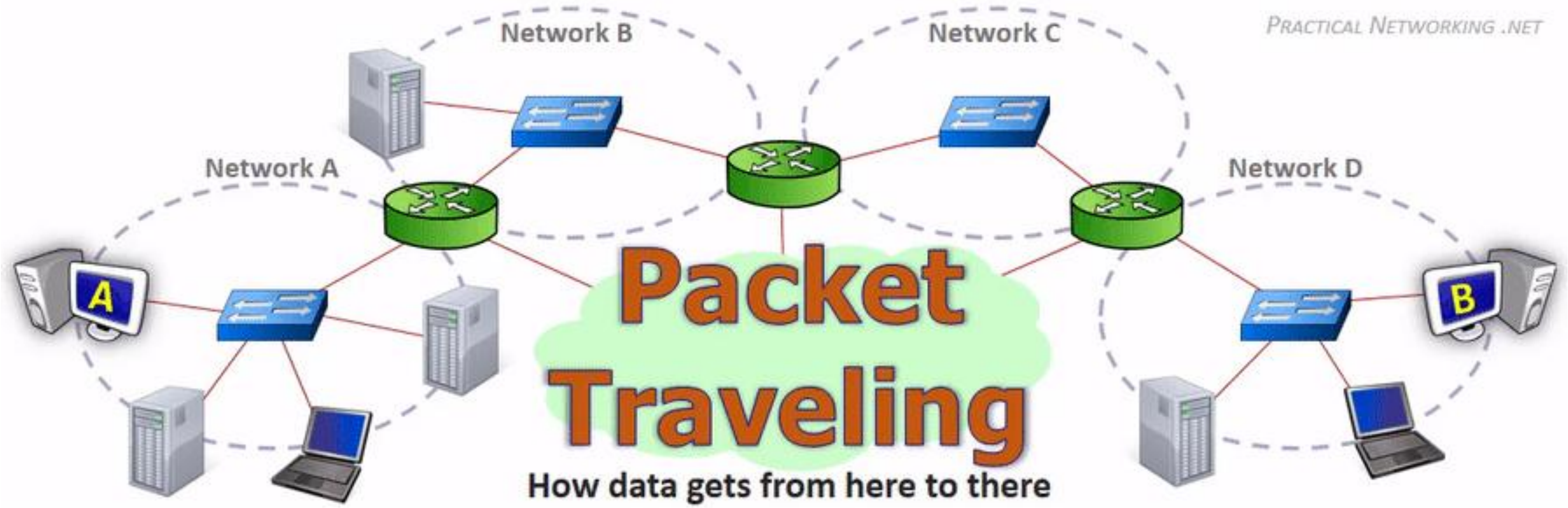
SDN Architecture – Northbound Devices



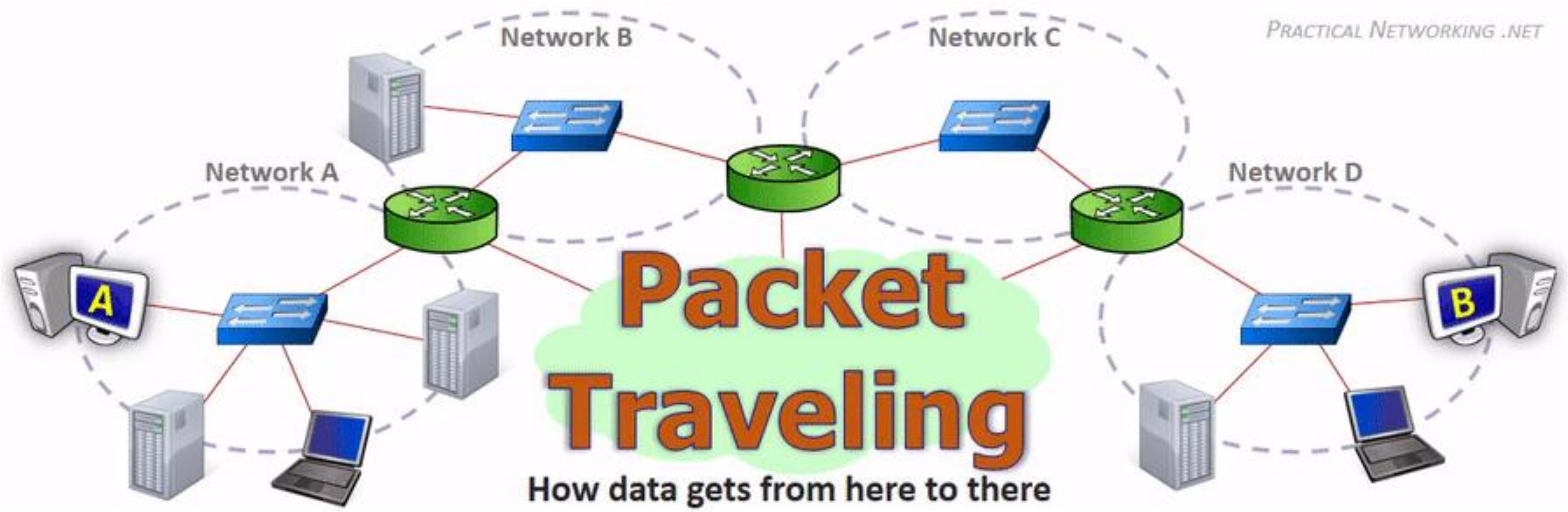
SDN Architecture – (Controller) Applications



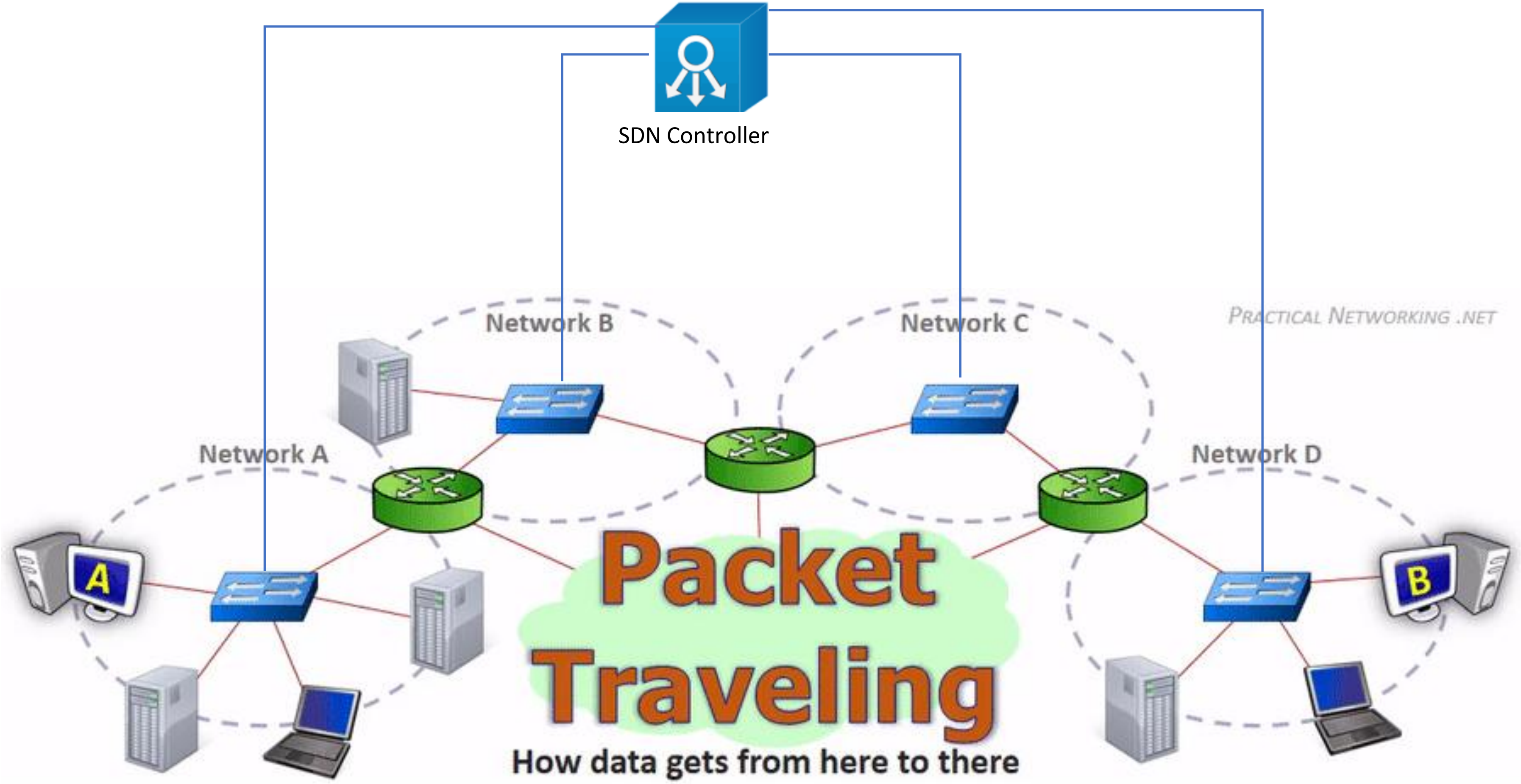
Dataplane / Controlplane separation



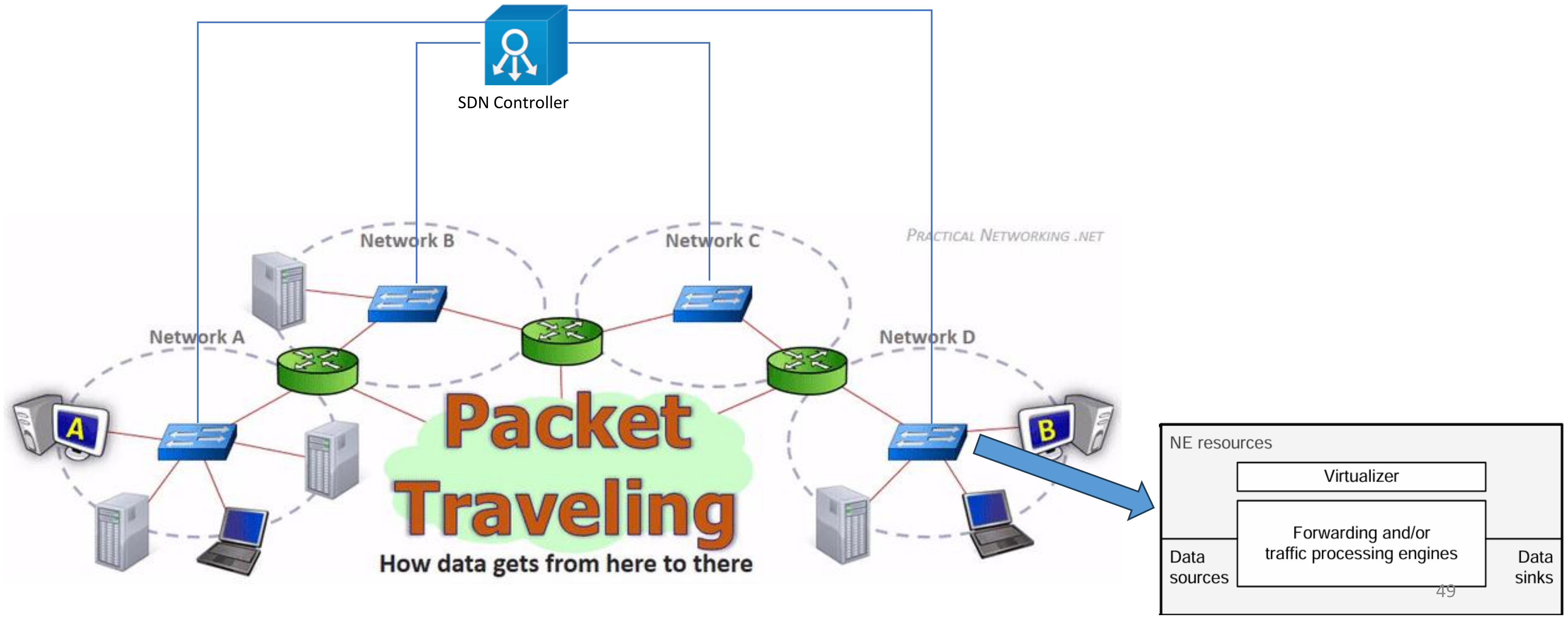
Dataplane / Controlplane separation



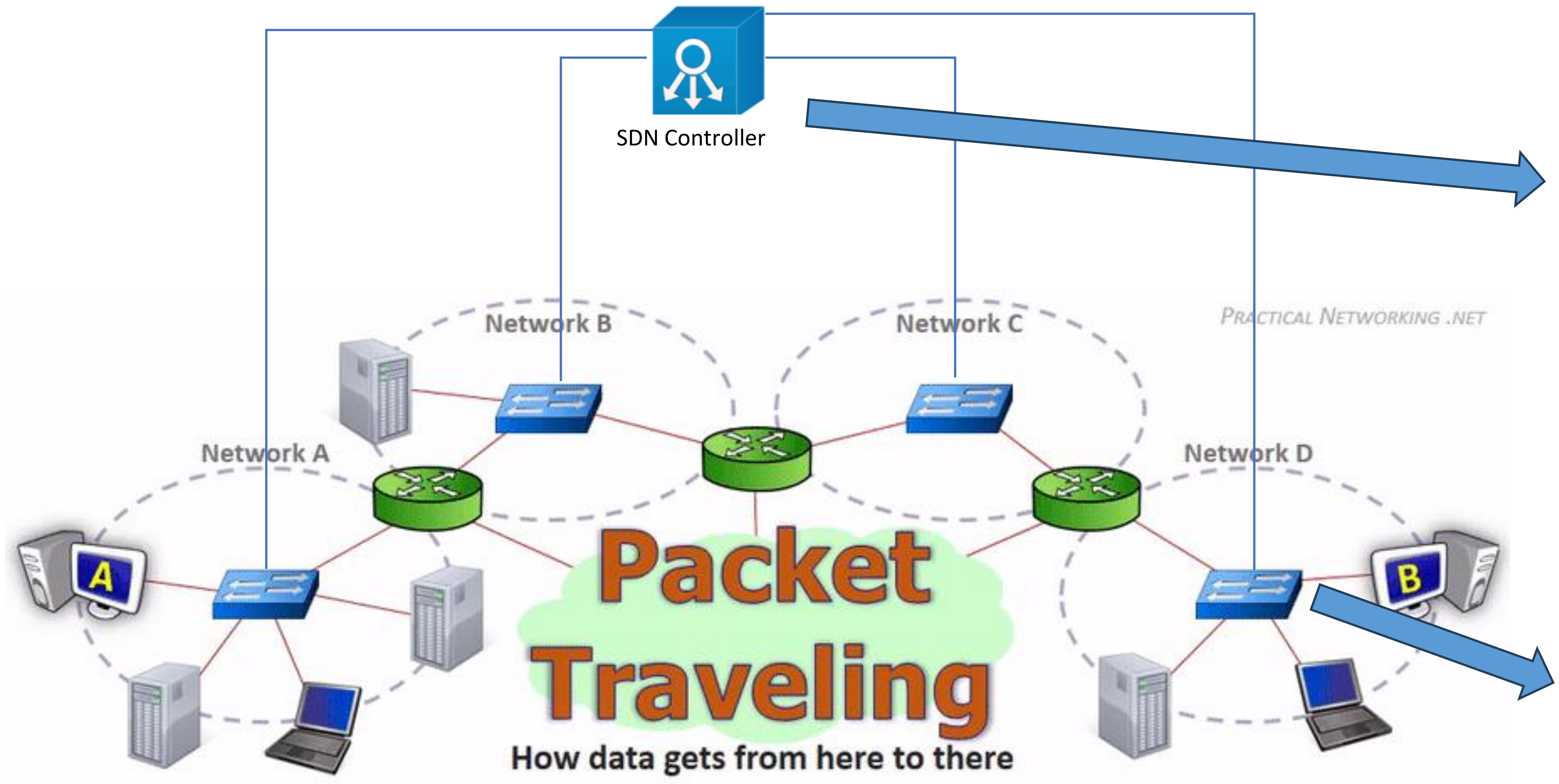
Dataplane / Controlplane separation

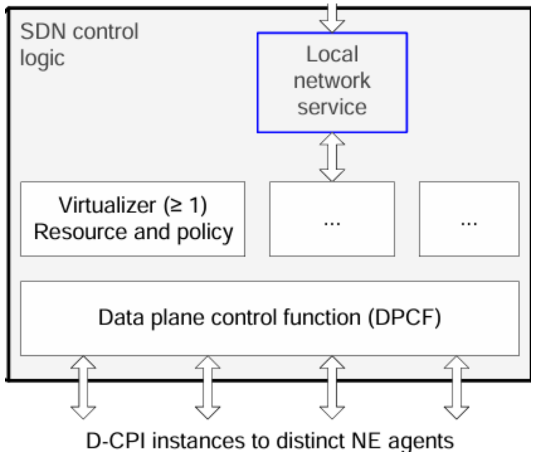


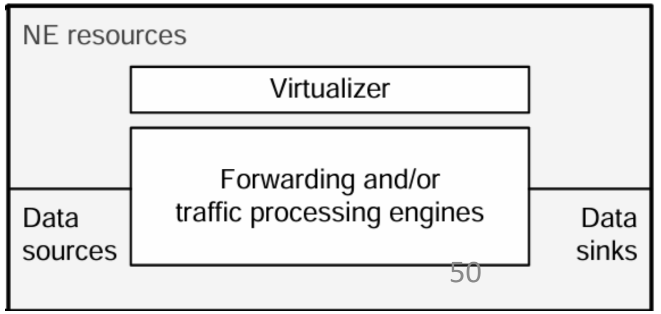
Dataplane / Controlplane separation



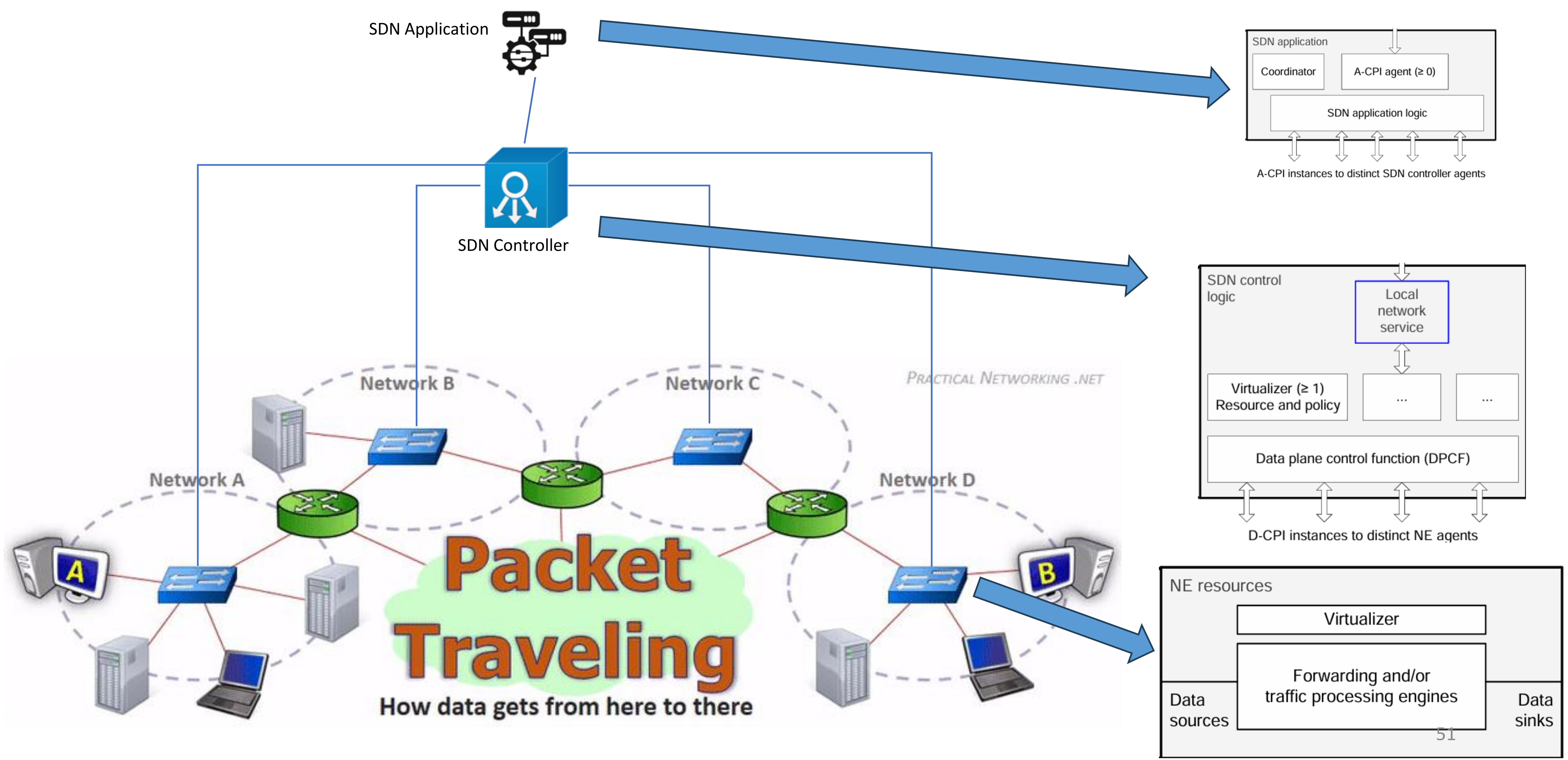
Dataplane / Controlplane separation





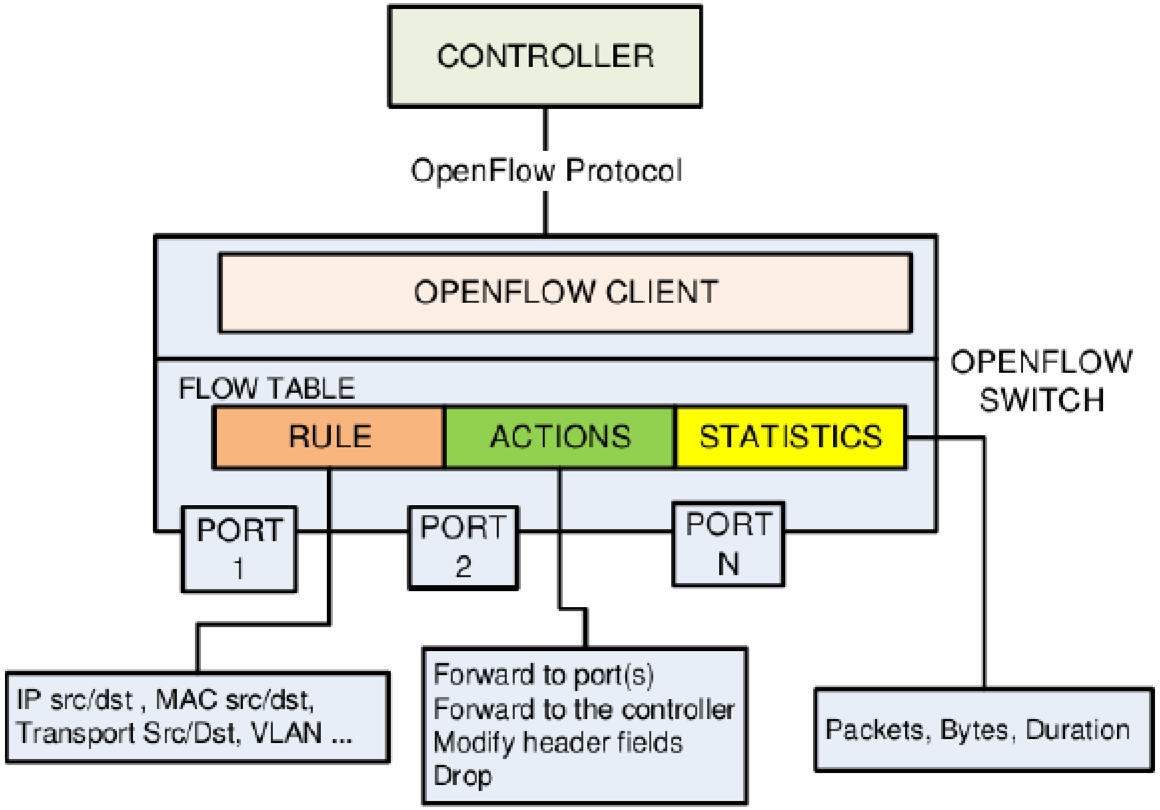


Dataplane / Controlplane separation

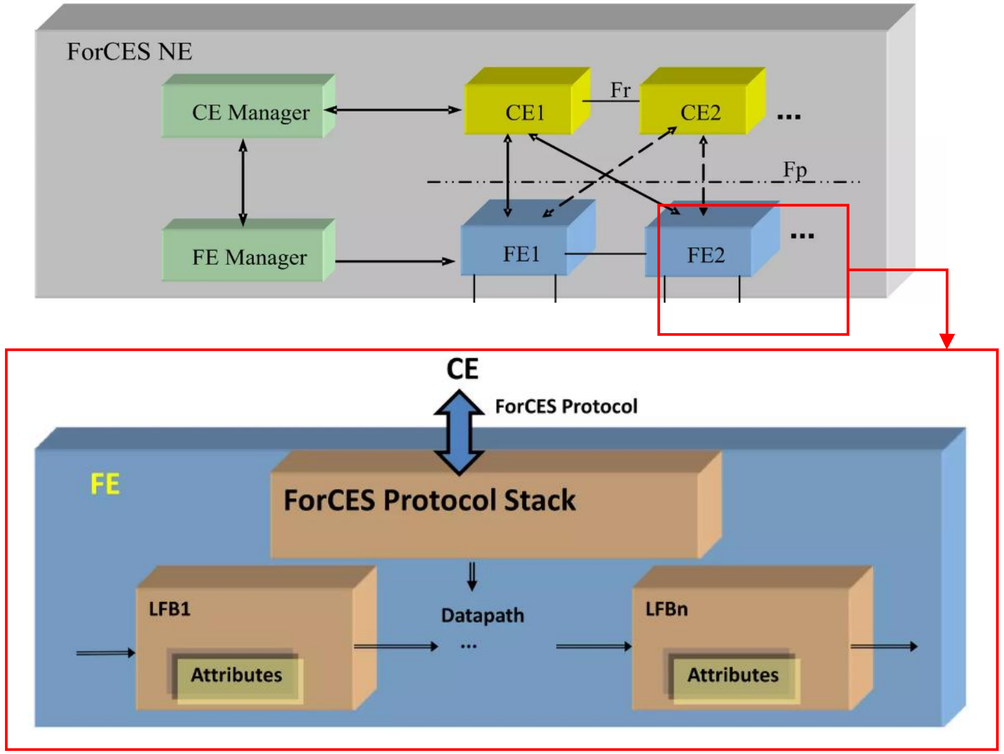


SDN Architecture Instantiations

OpenFlow



ForCES – Forwarding and Control Element Separation

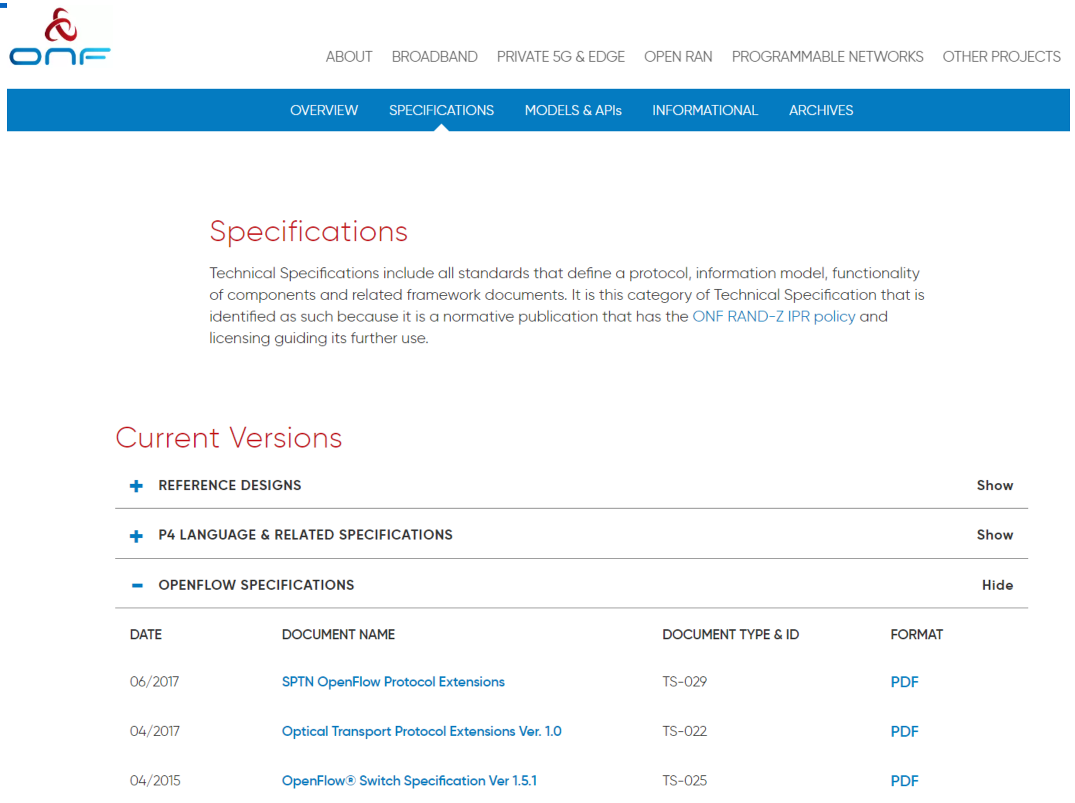


Source: Bruno Nunes Astuto, Marc Mendonça, Xuan Nam Nguyen, Katia Obraczka, Thierry Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. Communications Surveys and Tutorials, IEEE Communications Society, 2014, 16 (3), pp.1617- 1634. 10.1109/SURV.2014.012214.00180.

Antonio Capone, Carmelo Cascone, "From dumb to smarter switches in software defined networks: an overview of data plane evolution", IEEE NETSOFT 2015

OpenFlow

- Standardised by the ONF – Open Networking Foundation
- https://opennetworking.org/software-defined-standards/specifications/
- Currently on version 1.5



ABOUT BROADBAND PRIVATE 5G & EDGE OPEN RAN PROGRAMMABLE NETWORKS OTHER PROJECTS

OVERVIEW SPECIFICATIONS MODELS & APIs INFORMATIONAL ARCHIVES

Specifications

Technical Specifications include all standards that define a protocol, information model, functionality of components and related framework documents. It is this category of Technical Specification that is identified as such because it is a normative publication that has the ONF RAND-Z IPR policy and licensing guiding its further use.

Current Versions

+ REFERENCE DESIGNS Show

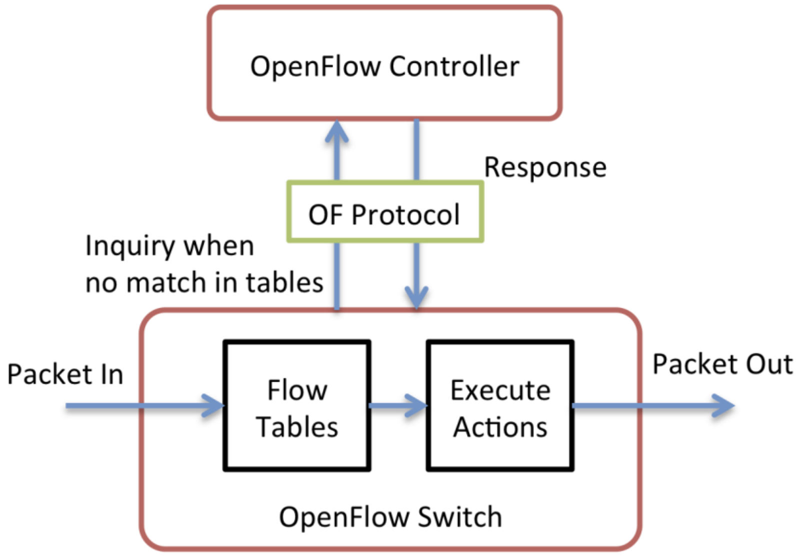
+ P4 LANGUAGE & RELATED SPECIFICATIONS Show

– OPENFLOW SPECIFICATIONS Hide

DATE	DOCUMENT NAME	DOCUMENT TYPE & ID	FORMAT
06/2017	SPTN OpenFlow Protocol Extensions	TS-029	PDF
04/2017	Optical Transport Protocol Extensions Ver. 1.0	TS-022	PDF
04/2015	OpenFlow® Switch Specification Ver 1.5.1	TS-025	PDF

OpenFlow

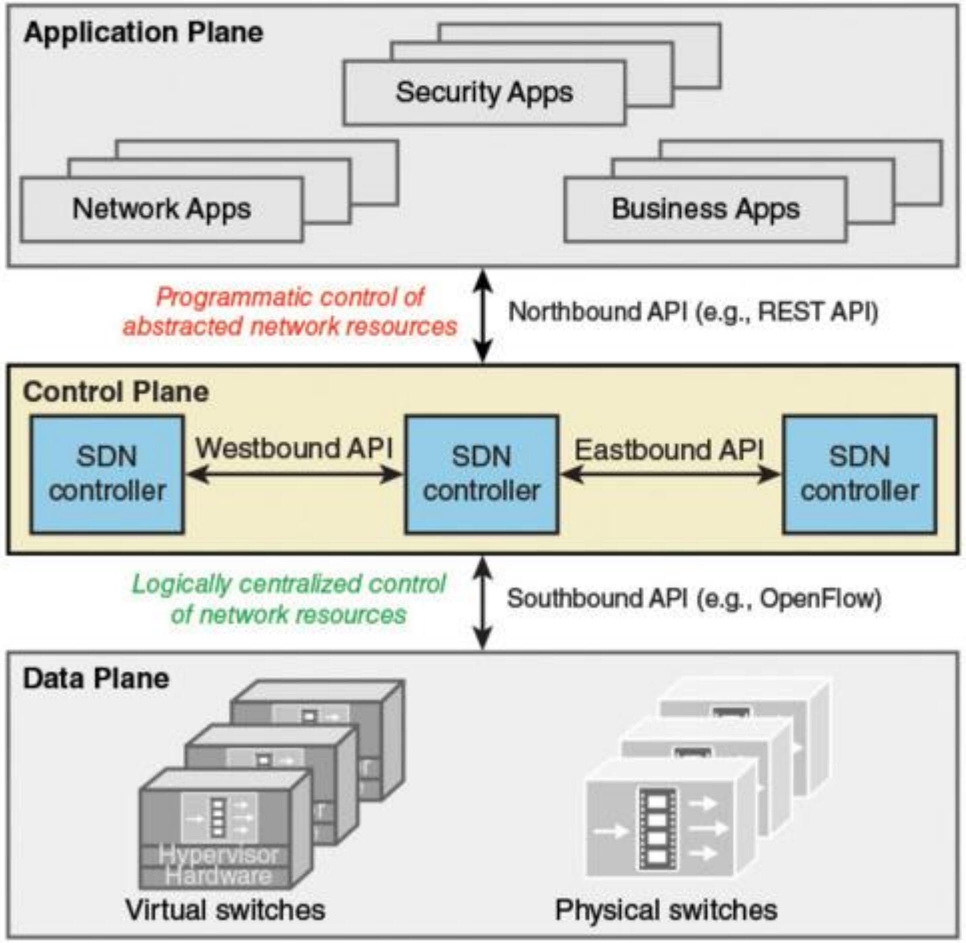
- A protocol existing between SDN switches and a new entity
  - SDN controller
- Allows the SDN controller to manage flow tables in SDN switches
- SDN switch contains
  - Openflow agent
  - Flow tables
  - Performs packet lookup and forwarding
  - Is able to communicate (securely) with the controller
- A flow table is composed of
  - Flow entries (matches properties in packets' headers)
  - Counters (for activity)
  - A set of actions to be applied (to matching packets)
  - When no actions are presente, the switch can
    - Drop the packet
    - Ask the controller what to do



Source: fibre-optic-transceiver-module.com

OpenFlow Controller

- Controller
  - Service existing in a server, which uses the OpenFlow protocol to interact with SDN switches
  - Formulates flows and programs switches
  - Is able to receive directives from external applications via northbound REST API



OpenFlow Controller

- Many existing flavours
  - ONOS – Open Network Operating System
    - https://opennetworking.org/onos/
  - TeraFlow SDN
    - https://tfs.etsi.org
  - OpenDaylight
    - https://opendaylight.org
  - Floodlight (last version from 2016)
    - https://floodlight.atlassian.net
  - NOX (10 w/o maintenance)
  - POX (Python version of NOX w/ some maintenance)
    https://noxrepo.github.io/pox-doc/html/
  - Ryu (w/o maintenance since 2017)
    - https://ryu-sdn.org/
  - Trema (5y since last update)
    - https://github.com/trema/trema
  - Frenetic (last release: 2019)
    - https://github.com/frenetic-lang/frenetic

Differences between SDN Controllers

- Research vs production
- Programming language
- Performance
- Learning Curve
- User base and Support
- Focus
  - Southbound API Support
  - Northbound API
  - OpenFlow version

A Qualitative and Quantitative assessment of SDN Controllers

Pedro Bispo, Daniel Corujo, Rui L. Aguiar
Instituto de Telecomunicações e Universidade de Aveiro, Portugal
Email:{pedrobispo, ruilaa}@ua.pt; dcorujo@av.it.pt

Abstract—**With the increasing number of connected devices, new challenges are being raised in the networking field. Software Defined Networking (SDN) enables a greater degree of dynamism and simplification for the deployment of future 5G networks. In such networks, the controller plays a major role by being able to manage forwarding entities, such as switches, through the application of flow-based rules via a southbound (SB) interface. In turn, the controller itself can be managed by means of actions and policies provided by high-level network functions, via a northbound (NB) interface.**

The growth of SDN integration in new mechanisms and network architectures led to the development of different controller solutions, with a wide variety of characteristics. Despite existing studies, the most recent evaluations of SDN controllers are focused only on performance and are not up to date, since new versions of the most popular controllers are constantly being released. As such, this work provides a wider study of several open-source controllers, (namely, OpenDaylight (ODL), Open Network Operative System (ONOS), Ryu and POX), by evaluating not only their performance, but also their characteristics in a qualitative way. Taking performance as a critical issue among SDN controllers, we quantitatively evaluated several criteria by benchmarking the controllers under different operational conditions, using the Cbench tool.

Keywords—Software-Defined Networking, OpenFlow, SDN controller.

reachability optimization towards the users, and the users themselves want overall better service. Simultaneously satisfying involved actors is a highly complex task, whose harmonization is only achieved through careful planning and overprovisioning of networking resources. Nonetheless, the increase in generated data [1] and the need to dynamically adapt to changing situations in a cost-effective way, are demanding for more flexible and adaptive network control mechanisms. As a result, SDN has emerged.

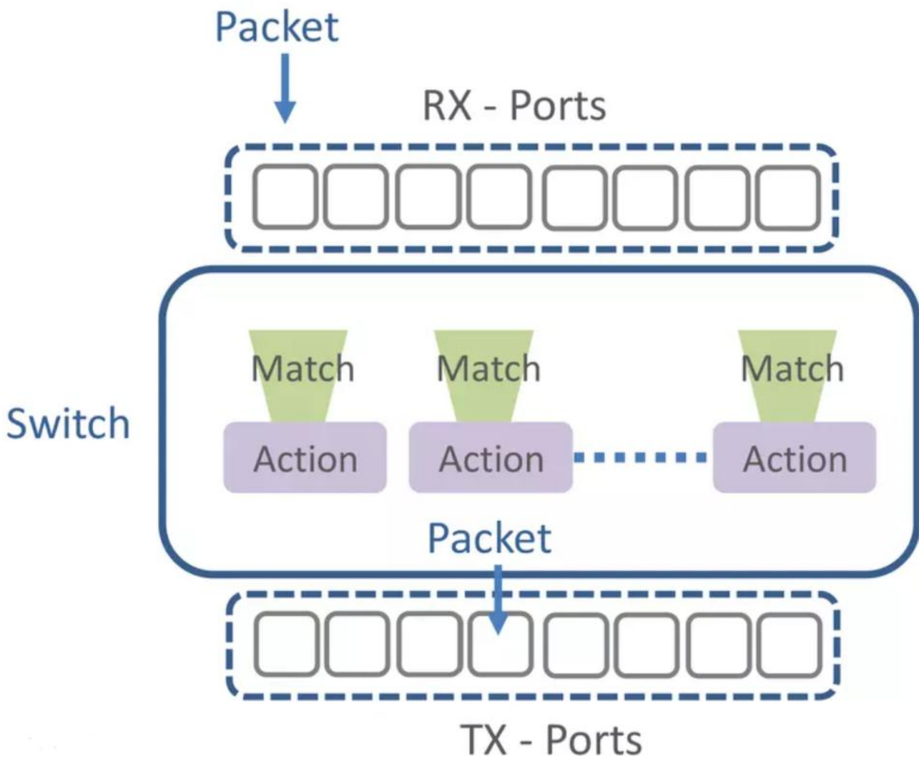
SDN provides the separation of the network control plane from the forwarding plane, allowing more control, adaptability, agility and overall cost reduction. By having a complete view of the network, an SDN controller plays an extremely important role in such networks as it can manage the network structure and services dynamically.

Several SDN controllers exist, with most of them under continuous development. This diversity, which originated from the different needs of operators and research teams that resulted in the development of their own controller versions, made comparison efforts more difficult.

This diversity is evidenced as each controller presents different Northbound (NB) and Southbound (SB) interfaces (which allow it to be interfaced by high-level entities and to control forwarding entities, respectively), development

SDN switch

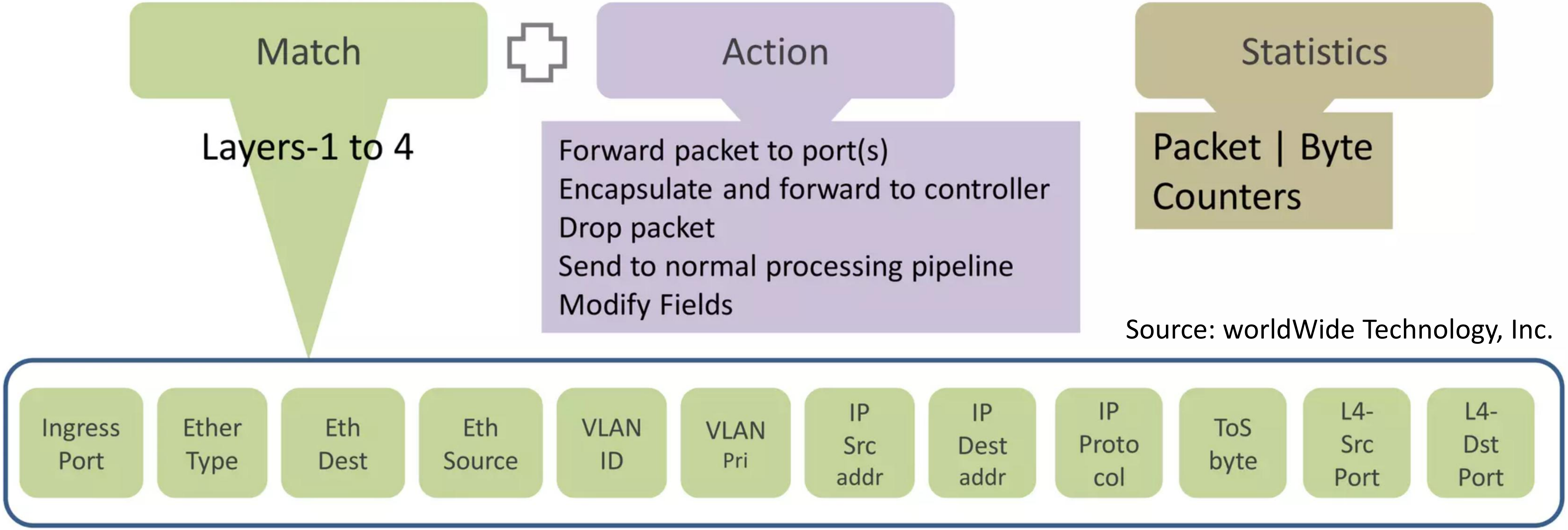
- Has an OpenFlow agent that is able to communicate with the controller
- Processes commands received by the controller
- Dataplane of a switch
  - Ports
  - Flow tables
  - Flows
  - Classifiers (Match)
  - Modifiers and actions
- Packets are matched to flows in flow tables using the match/classifiers
- Flows contain sets of modifiers and actions which are applied to each packet that it matches



Source: worldWide Technology, Inc.

SDN Switch – Flow table

- Each flow table entry contains: Match, Action and counters



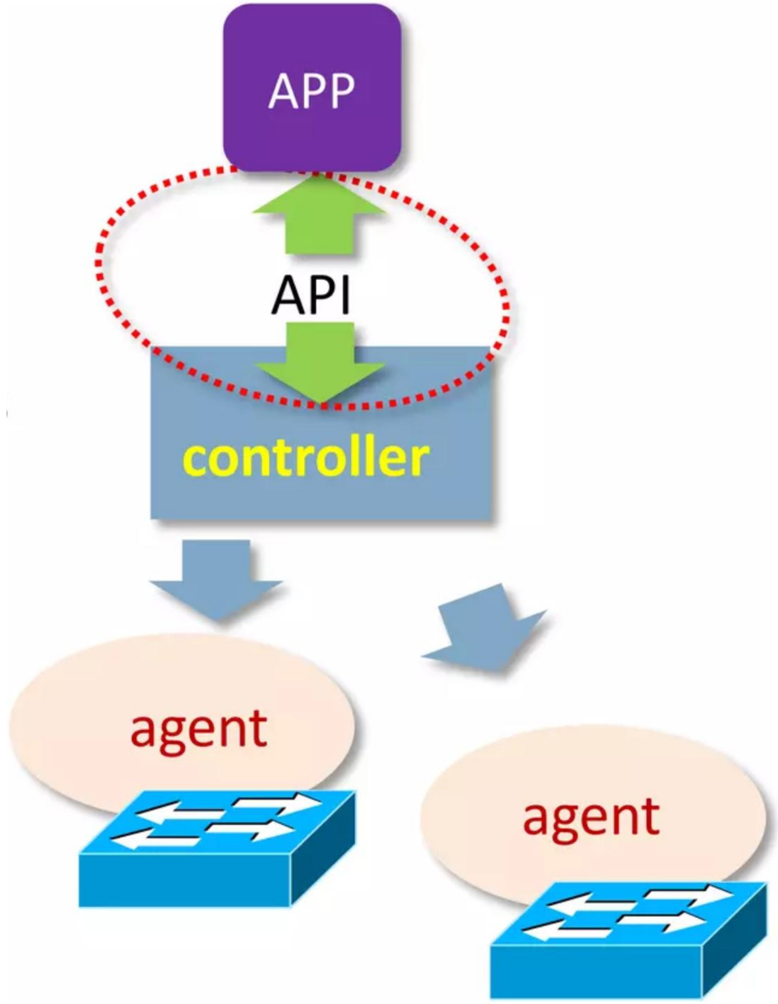
Source: worldWide Technology, Inc.

SDN Switch - Actions

- When a switch first connects to a controller, it specifies which actions are supported
  - Not all switches need to implemente all OpenFlow actions
- Examples of actions
  - Forward a packet to a set of ports
  - Drop a packet
  - Add, modify or remove VLAN ID or priority on a per-destination-port basis
  - Modify the IP DSCP (i.e., QoS)
  - Modify the destination MAC address
  - Send the packet to the OpenFlow controller (Packet In)
  - Receive the packet from the OpenFlow controller and send it to ports (Packet out)

How to "direct" the controller?

- Northbound interface allows for Northbound protocols
- Allows applications and orchestration systems to program the network and request services
- Provides a network abstraction interface to applications
- The abstraction is important when managing dissimilar network elements



Northbound Protocol Examples

- There is nothing standardized, but there are “types” of protocols used
- REST (web based) API – applications which run on different machine or address space on the controller
- Web Browser
  - http://<SDN Controller IP>:8080/ ….
- WebSockets
- OSGi framework is used for applications that will run in the same address space as the controller.
  - Open Service Gateway Initiative

Other SDN Protocols – NETCONF e YANG

- A network's efficient control requires well defined abstractions to manipulate network elements
- Isn't the objective to manage a network after all?
  - SNMP, CMIP, Corba, CIM, SOAP, REST, …
- SDN is not just OpenFlow
- You need to ming aspects such as security, reliability, extensibility,…
  - YANG
    - Defines datamodels that represent and work as network element abstractions, facilitating their manipulation
  - NETCONF
    - API and transactions that allow to manipulate the abstractions defined by a datamodel

Other SDN Protocols – NETCONF

- Network management protocol designed to support the activation and service provisioning
- Uses XML over SSH+TCP
- XML namespaces are extensible
- Supports transactions, in a “all or nothing” and “all together” way, to simplify network management applications
- Addresses multiple network elements in parallel

Other SDN Protocols – YANG

- Datamodel designed to be used with NETCONF
- TO be used in a similar way to existing network operator procedures, namely CLI and NetUI
- Allows to define new concepts using its syntax, to be validated by compilers
- Non-programmers can read YANG models

Other SDN Protocols – NETCONF+YANG

- How does NETCONF+YANG relate with SDN?
  - Complements it!
- Device Configuration Management
  - Beyond OpenFlow
  - And similar protocols
    - OF-CONFIG
      - OpenFlow Management and Configuration Protocol (ONF TS-016 standard)
      - https://opennetworking.org/wp-content/uploads/2017/07/of-config-1.2.pdf
- Allowed to be used via Northbound Interface

Towards the next generation 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.

Categories and Subject Descriptors

C.2 [**Internetworking**]: Routers

General Terms

Experimentation, Design

Keywords

Ethernet switch, virtualization, flow-based

1. THE NEED FOR PROGRAMMABLE NETWORKS

Networks have become part of the critical infrastructure of our businesses, homes and schools. This success has been both a blessing and a curse for networking researchers; their work is more relevant, but their chance of making an impact is more remote. The reduction in real-world impact of any given network innovation is because the enormous installed base of equipment and protocols, and the reluctance 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 they wish. A slice could extend across the backbone, into access networks, into college campuses, industrial research labs, and include wiring closets, wireless networks, and sensor networks.

Virtualized programmable networks could lower the barrier to entry for new ideas, increasing the rate of innovation in the network infrastructure. But the plans for nationwide facilities are ambitious (and costly), and it will take years for them to be deployed.

This whitepaper focuses on a shorter-term question closer to home: *As researchers, how can we run experiments in our campus networks?* If we can figure out how, we can start soon and extend the technique to other campuses to benefit the whole community.

To meet this challenge, several questions need answering, including: In the early days, how will college network administrators get comfortable putting experimental equipment (switches, routers, access points, etc.) into their network? How will researchers control a portion of their local network in a way that does not disrupt others who depend on it? And exactly what functionality is needed in network

2008

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennifer Rexford[**], Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David Walker[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label.

1. INTRODUCTION

Software-Defined Networking (SDN) gives operators programmatic control over their networks. In SDN, the control plane is physically separate from the forwarding plane, and one control plane controls multiple forwarding devices. While forwarding devices could be programmed in many ways, having a common, open, vendor-agnostic interface (like OpenFlow) enables a control plane to control forwarding devices from different hardware and software vendors.

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

Table 1: Fields recognized by the OpenFlow standard

The OpenFlow interface started simple, with the abstraction of a single table of rules that could match packets on a dozen header fields (e.g., MAC addresses, IP addresses, protocol, TCP/UDP port numbers, etc.). Over the past five years, the specification has grown increasingly more complicated (see Table 1), with many more header fields and multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new "OpenFlow 2.0" API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

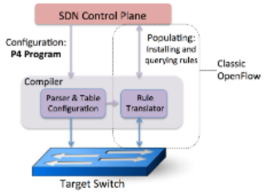


Figure 1: P4 is a language to configure switches.

Recent chip designs demonstrate that such flexibility can be achieved in custom ASICs at terabit speeds [1, 2, 3]. Programming this new generation of switch chips is far from easy. Each chip has its own low-level interface, akin to microcode programming. In this paper, we sketch the design of a higher-level language for Programming Protocol-independent Packet Processors (P4). Figure 1 shows the relationship between P4—used to configure a switch, telling it how packets are to be processed—and existing APIs (such as OpenFlow) that are designed to populate the forwarding tables in fixed function switches. P4 raises the level of abstraction for programming the network, and can serve as a

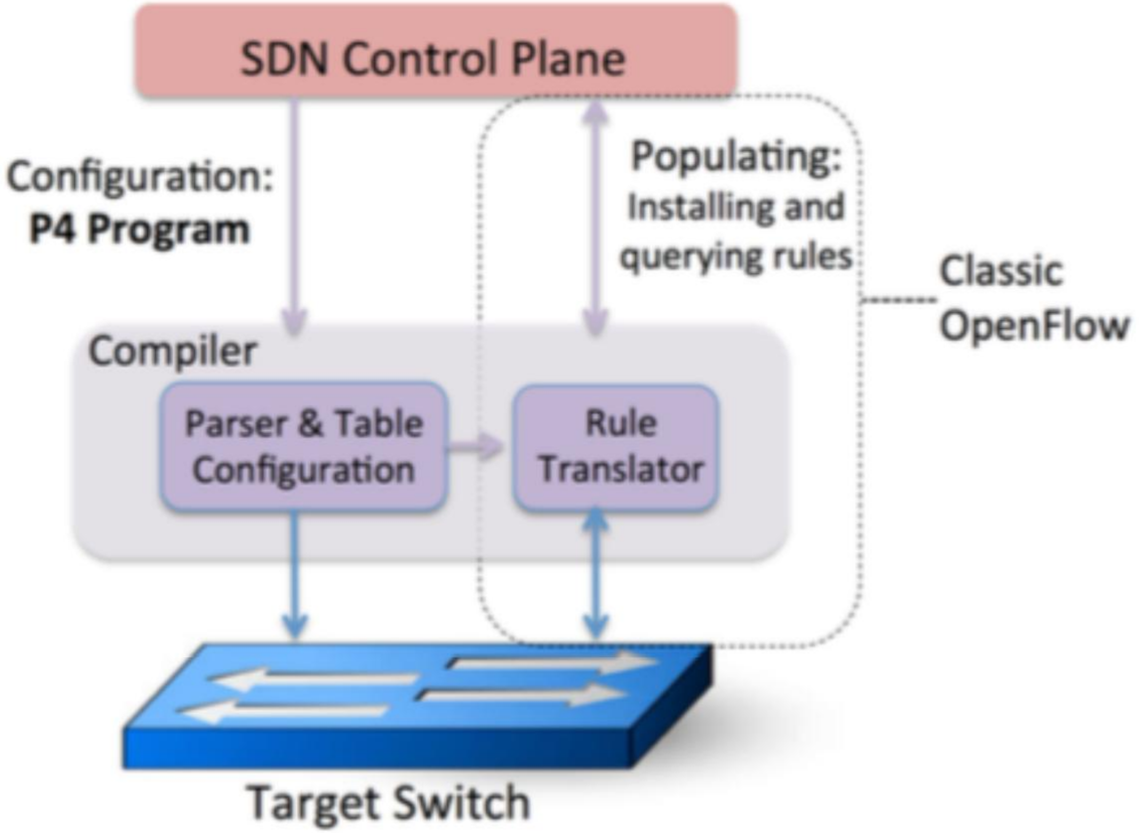
2014

Programming Protocol-Independent Packet Processors (P4)

- Objectives
  - Reconfigurable
    - Data Plane program can be changed in the field
  - Protocol-Independence
    - No knowledge of low-level hardware organization is required
    - Compiler compiles the program for the target device
    - Architecture dependente – e.g. v1model.p4, p4c-xdp.p4, psa.p4
  - Switch/vendor Independence
  - Consistent Control Plane Interface
    - Control plane APIs are automatically generated by the compiler
  - Community-driven design
    - https://p4.org

Programming Protocol-Independent Packet Processors (P4)

- P4 program is a high-level programm that configures forwarding behavior (abstract forwarding model)
- P4 compiler generates the low-level code to be executed by the desired target
- OpenFlow can still be used to install and query rules once forwarding model is defined
- Allows the definition of arbitrary headers and fields



P4 Header and Fields

- Fields have a bit width and other attributes
- Headers are collections of fields
  - Like an instantiated class in Java

```
header_type ethernet_t {
  fields {
    dstAddr     : 48;
    srcAddr     : 48;
    etherType   : 16;
  }
}

/* Instance of eth header */
header ethernet_t inner_ethernet;
```

```
header_type egress_metadata_t {
  fields {
    nhop_type    : 8;  /* 0: L2, 1: L3, 2: tunnel */
    encap_type   : 8;  /* L2 Untagged; L2ST; L2DT */
    vnid         : 24; /* gnve/vxlan vnid/gre key */
    tun_type     : 8;  /* vxlan; gre; nvgre; gnve*/
    tun_idx      : 8;  /* tunnel index */
  }
}

metadata egress_metadata_t egress_metadata;
```

Parser

- Extracts header instances
- Selects a next “state” by returning another parser function

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_CPU  : parse_cpu_header;
    ETHERTYPE_VLAN : parse_vlan;
    ETHERTYPE_MPLS : parse_mpls;
    ETHERTYPE_IPV4 : parse_ipv4;
    ETHERTYPE_IPV6 : parse_ipv6;
    ETHERTYPE_ARP  : parse_arp_rarp;
    ETHERTYPE_RARP : parse_arp_rarp;
    ETHERTYPE_NSH  : parse_nsh;
  }
}
```

Match + Action table

- Parsed representation of headers gives contexto for processing of the packets
- An action function consists of several primitive actions

```
table acl {
    reads {
        ipv4.dstAddr : ternary;
        ipv4.srcAddr : ternary;
        ipv4.protocol : ternary;
        udp.srcPort : ternary;
        udp.dstPort : ternary;
        ethernet.dstAddr : exact;
        ethernet.srcAddr : exact;
        ethernet.etherType : ternary;
    }
    actions {
        no_op;    /* permit */
        acl_drop; /* reject */
        nhop_set; /* policy-based routing */
    }
}
```

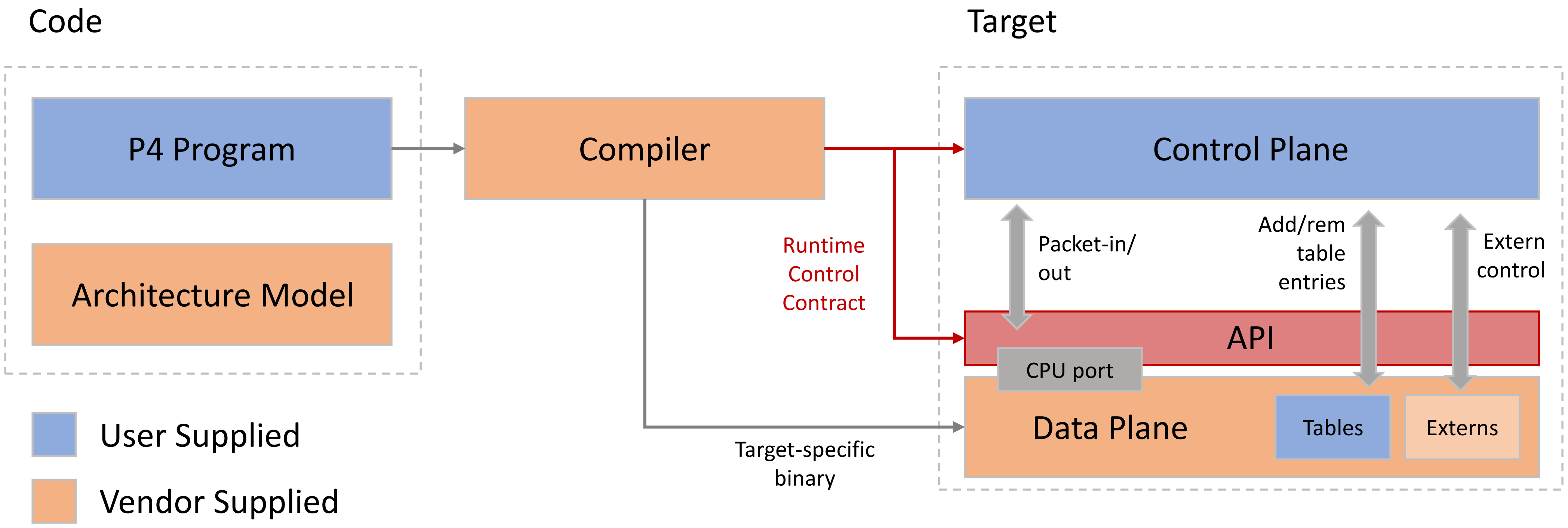
Match semantics

- **exact**
  - port_index : exact
- **ternary**
  - ethernet.srcAddr : ternary
- **valid**
  - vlan_tag[0] : valid
- **lpm**
  - ipv4.dstAddr : lpm
- **range**
  - udp.dstPort : range

Primitive actions

- `modify_field, add_to_field, add, set_field_to_hash_index`
- `add_header, remove_header, copy_header`
- `push, pop`
- `count, meter`
- `generate_digest, truncate`
- `resubmit, recirculate`
- `clone_*`
- `no_op, drop`

Programming a target



P4Runtime

- Framework for runtime control of P4-defined data planes
  - Open-source API and a server implementation
- P4 program-independent
  - API doesn't change with the P4 program
- Enables field-reconfigurability
  - Ability to push new P4 program without recompiling the software stack of target switches
- API based on protobuf (serialization) and gRPC (cliente/server transport)
  - Makes it easy to implement a P4Runtime cliente/server by auto-generating code for different languages
- P4Info as a contract between control and data plane
  - Generated by P4 compiler
  - Needed by the control plane to format the body of P4Runtime messages

P4 and P4Runtime are two different things

- P4
  - Programming language used to define how a switch processes packets
  - Specifies the switch pipeline
    - Which fields does it match upon?
    - What actions does it perform on the packets?
    - In which order does it perform the matches and actions
  - Specify the behavior of an existing device
  - Specify a logical abstraction for the device
- P4Runtime
  - An API used to control switches whose behavior has already been specified in the P4 language
  - Works for different types of switches
    - Fixed
    - Semi-programmable
    - Fully programmable