Software and Virtualization Technologies in Mobile Communication Networks

Comunicações Móveis

DETI – UA

2022/2023

Outline

- Network Function Virtualization
- Management and Orchestration
- Software Defined Networking
- Multi-access Edge Computing
- OpenRAN
- Network automation

Network Function Virtualization

NFV

Virtualization

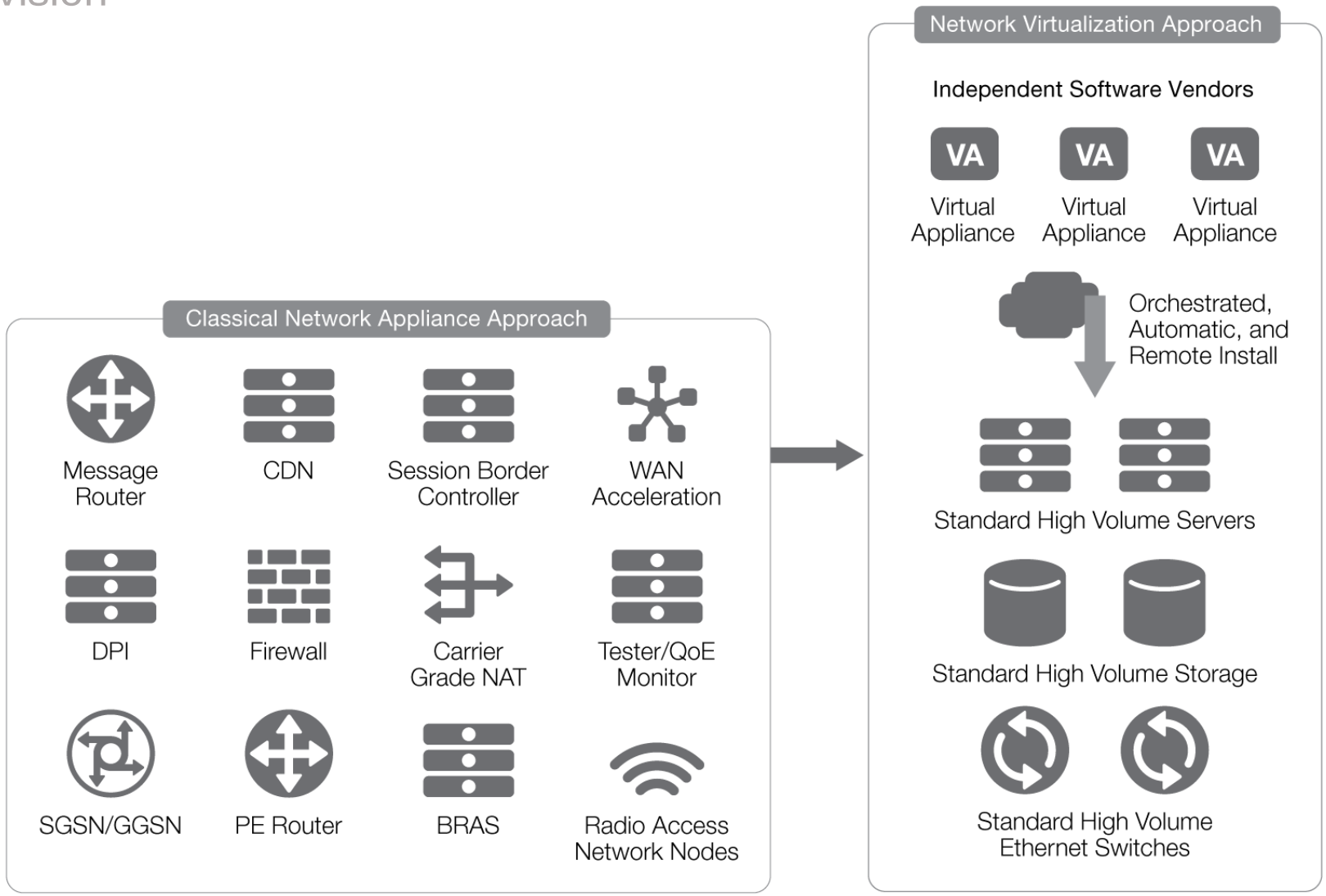
- 5G brought new trends
- Virtualization
  - Simulate a hardware platform, in software: VM’s, containers
    - Higher portability
    - Higher scalability
    - More cost-effective
- Virtualized networks
  - Logical software-based routers, switches, etc.
  - Network services are easier to deploy and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

Virtualized Network Functions

- VNFs
- Network functions running in a virtual way
- … over virtualization infrastructure
- As they are
  - Lift and shift
- Or employing cloud-native principles

Network Function Virtualization

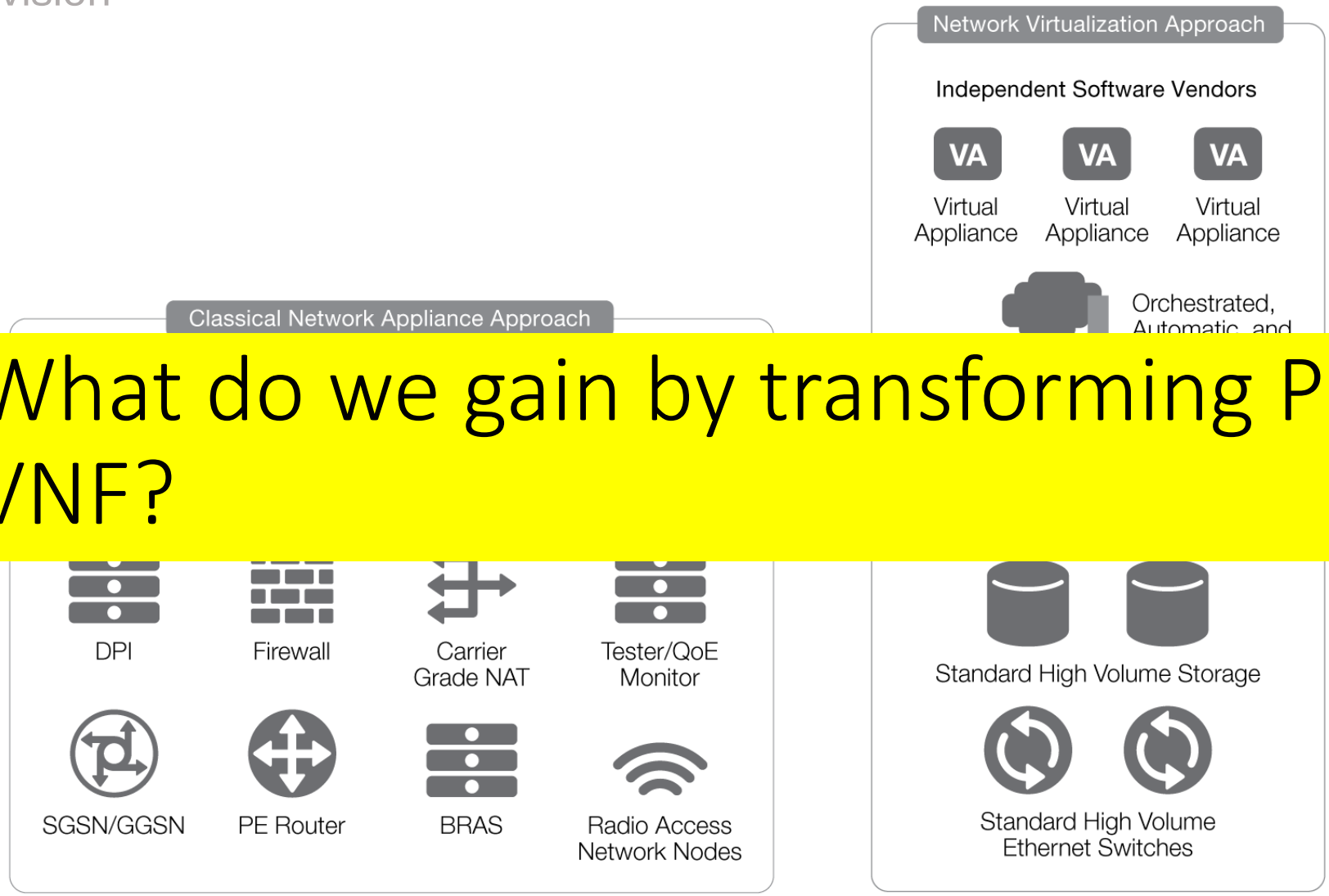
vision



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

Network Function Virtualization

vision



What do we gain by transforming PNF into VNF?

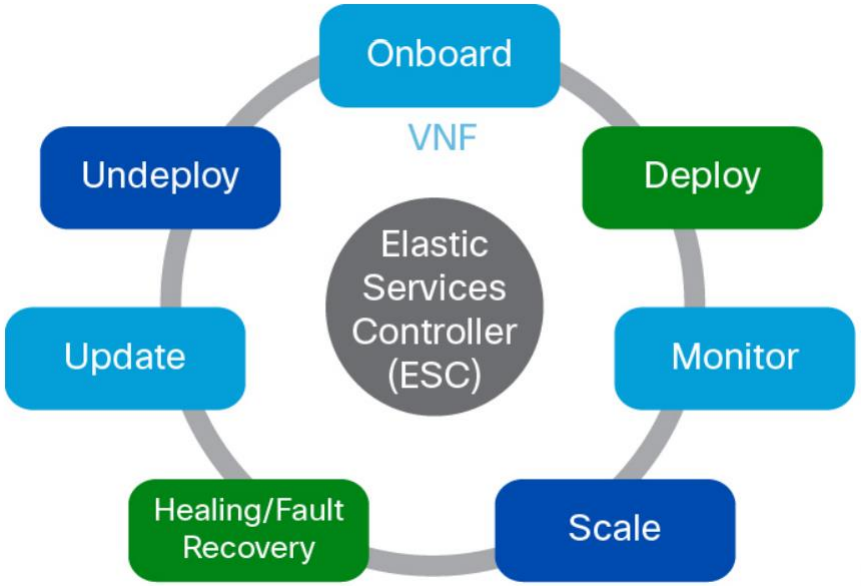
Source: https://www.f5.com/es_es/services/resources/white-papers/network-functions-virtualization-everything-old-is-new-again

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
      - …

Flexibility: VNF Lifecycle Management

- VNFs are deployed after a service request is done, via northbound interface
- That service request is composed of templates that consist of XML payloads and configuration parameters



Source: Cisco

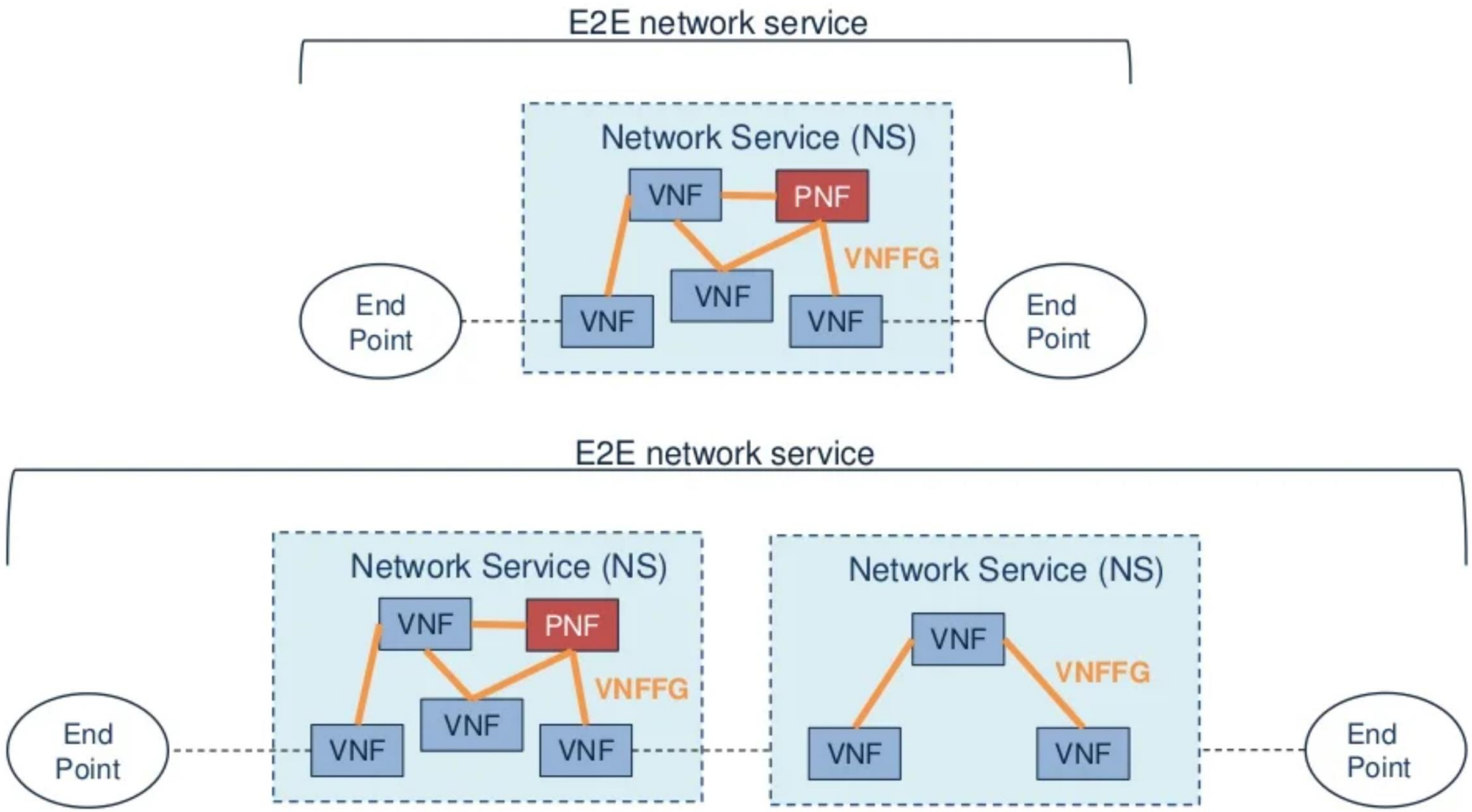
VNF Lifecycle Management

- Onboarding – placing the VNF image/template and configuration, in the system's catalogue;
  - Openstack qcow2 and cmdk disk formats and config drive
- Deploying – placing the resources in the virtualization infrastructure, along with zero-day configuration (credentials, licenses, network information, etc.)
- Monitoring – check the health of virtual machines regarding network status, CPU, memory consumption;
- Healing – after monitoring of failure conditions exposed as Key Performance Indicators (KPIs), the system can heal the VM (restart, expand, etc.)
- Updating – change the details of a virtual machine, set of virtual machines or networks that interconnect them
- Undeploy – remove a VM from the system.

Network Services

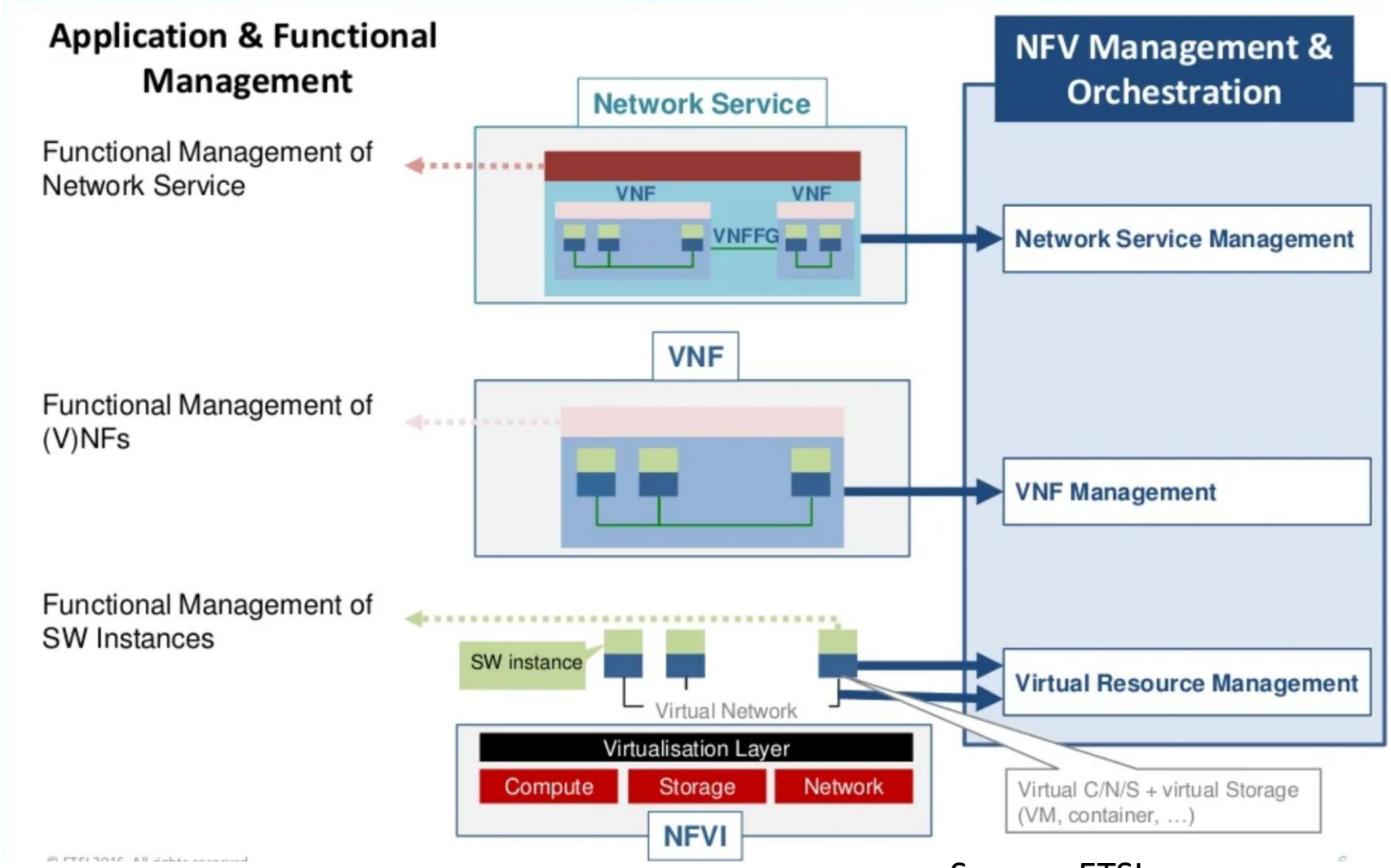
- A set of chained VNFs
- VNFs are interconnected and need to communicate with each other
  - Connectivity Forwarding Graph (FG) indicates how they are connected
    - VNFFG – VNF Forwarding Graph
- Can connect to physical network functions
  - Can also be part of the forwarding graph
- VNFs+VNFFG = 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



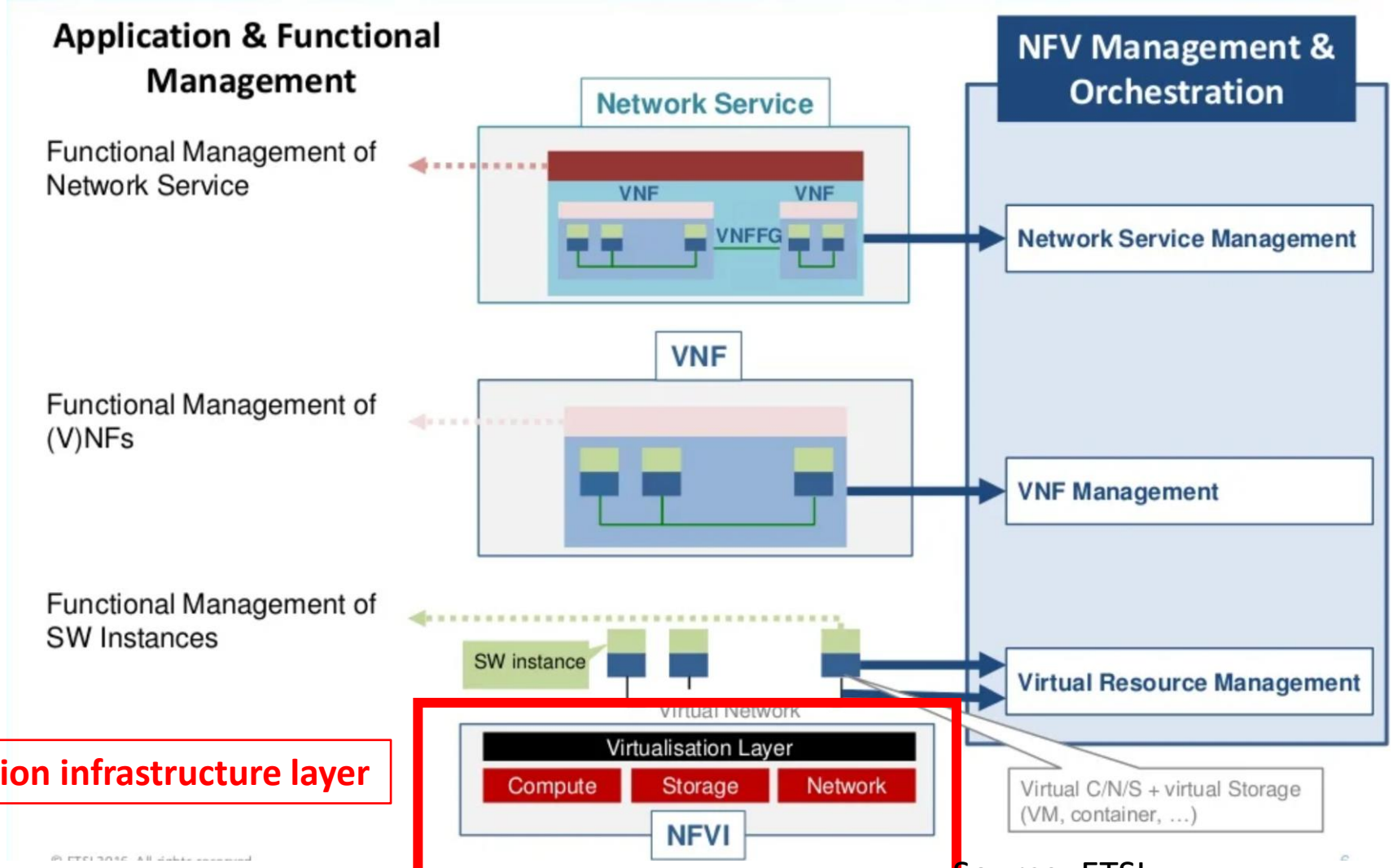
Source: ETSI

Overall Concepts and how they interconnect



Source: ETSI

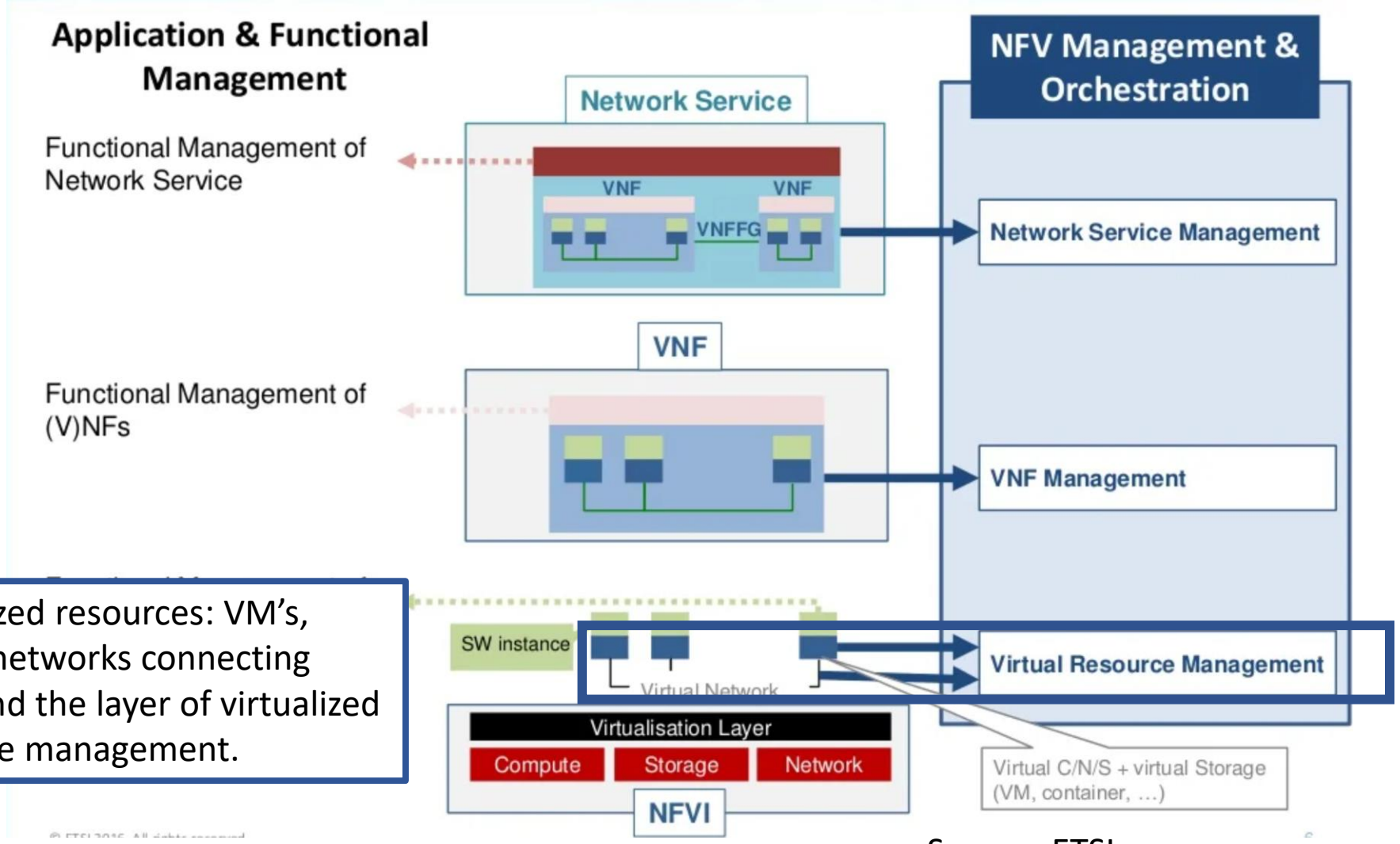
Overall Concepts and how they interconnect



Virtualization infrastructure layer

Source: ETSI

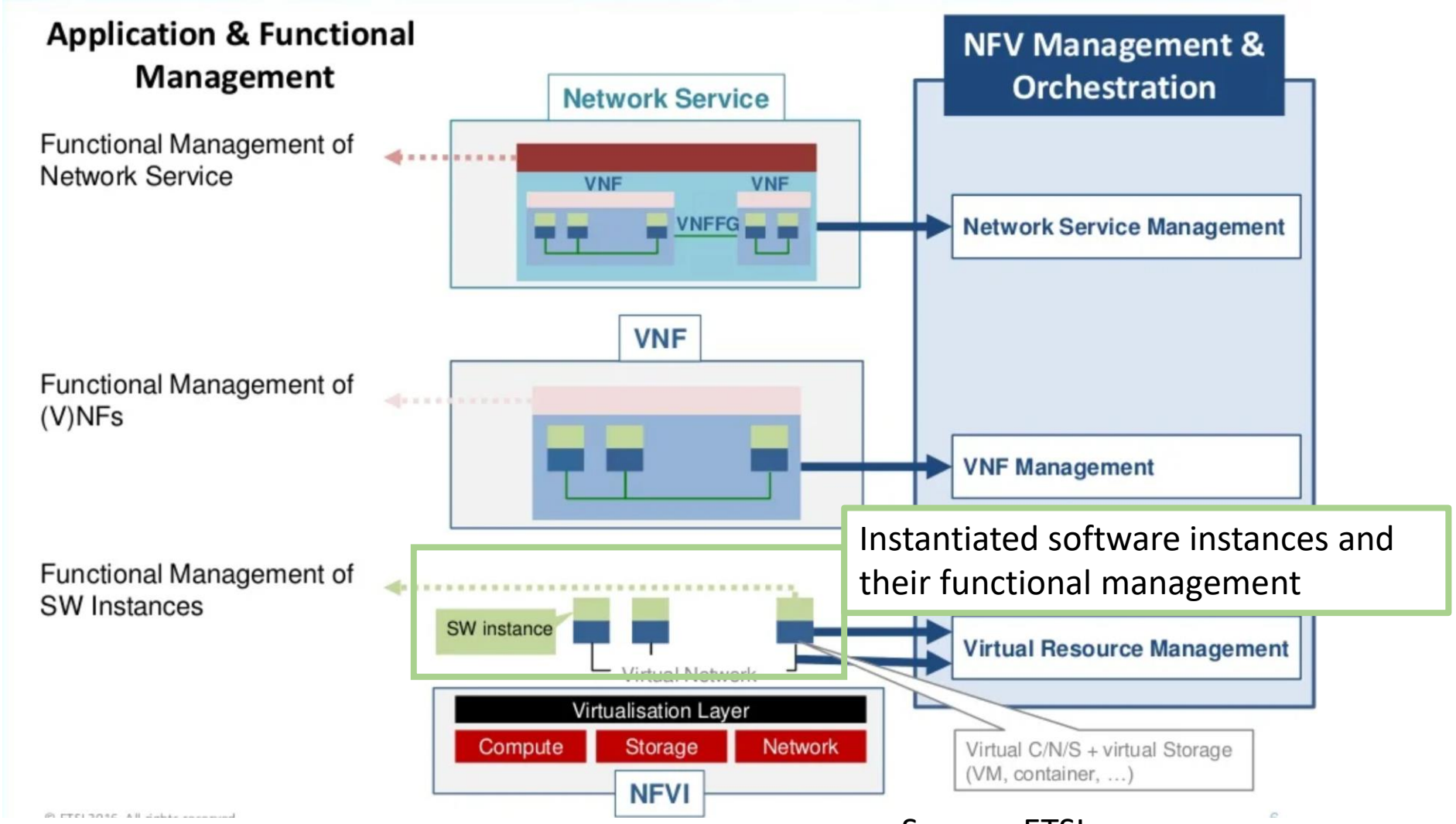
Overall Concepts and how they interconnect



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

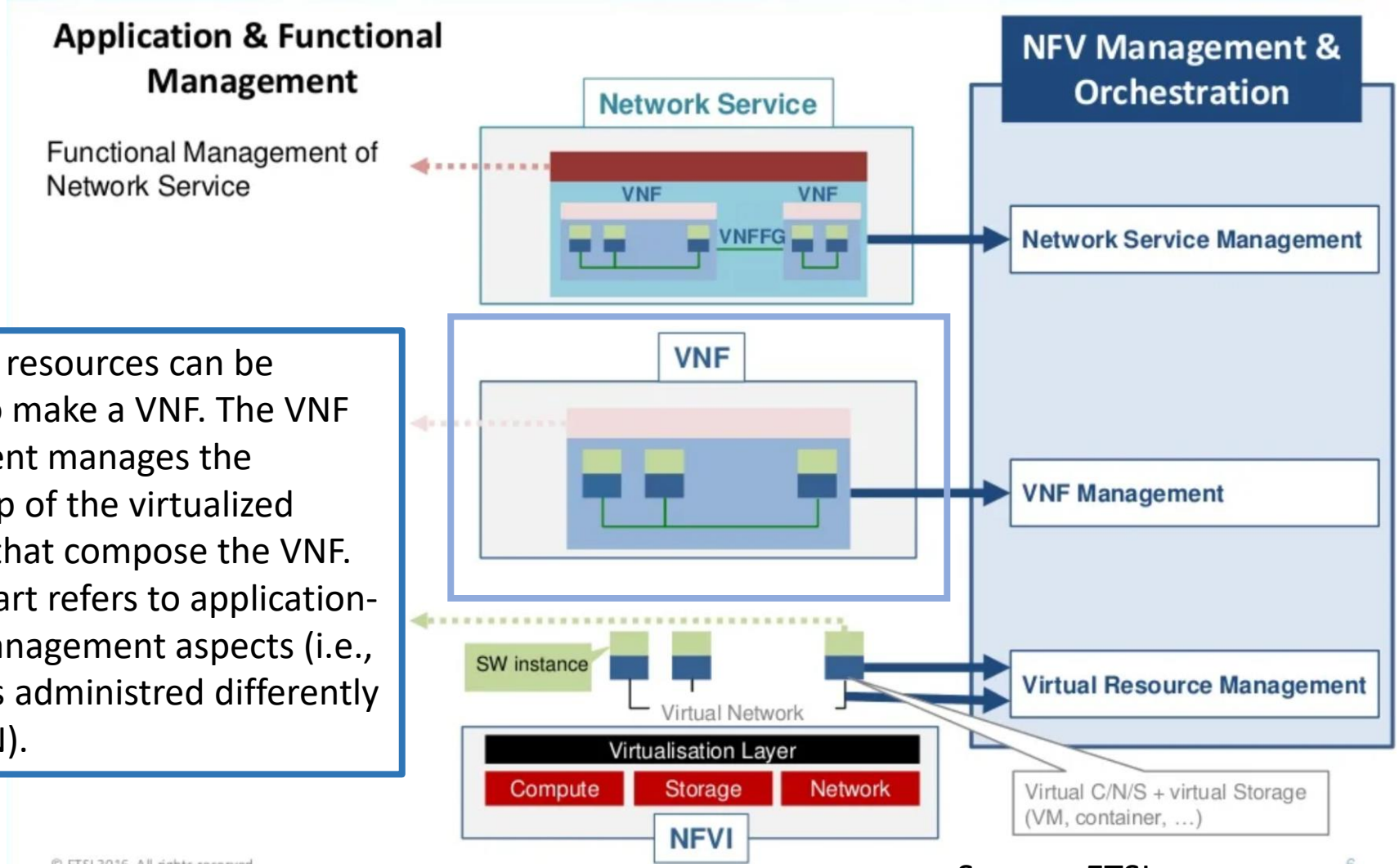
Source: ETSI

Overall Concepts and how they interconnect



Source: ETSI

Overall Concepts and how they interconnect

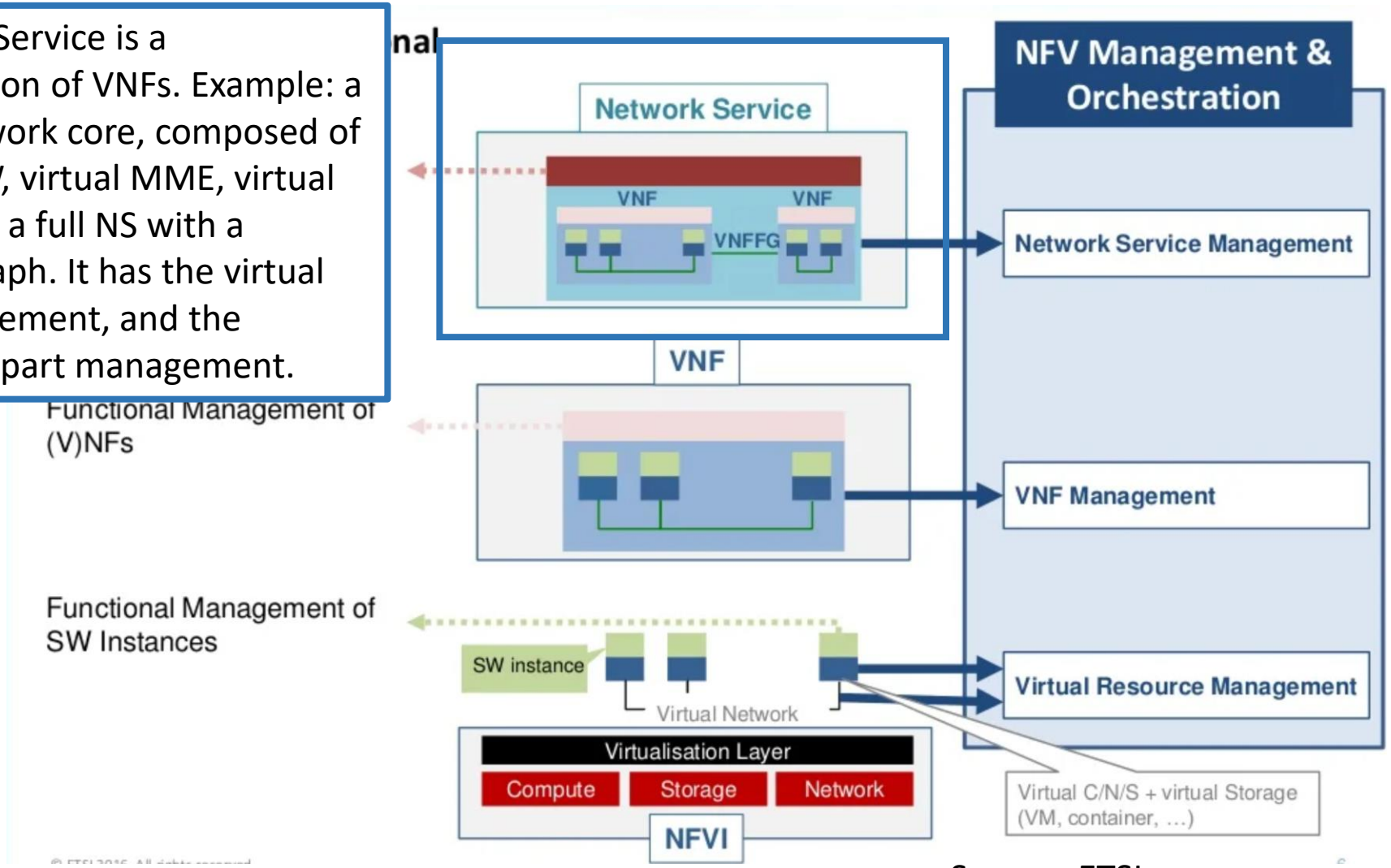


Virtualized resources can be grouped to make a VNF. The VNF management manages the relationship of the virtualized resources that compose the VNF. The pink part refers to application-specific management aspects (i.e., a firewall is administred differently than a CDN).

Source: ETSI

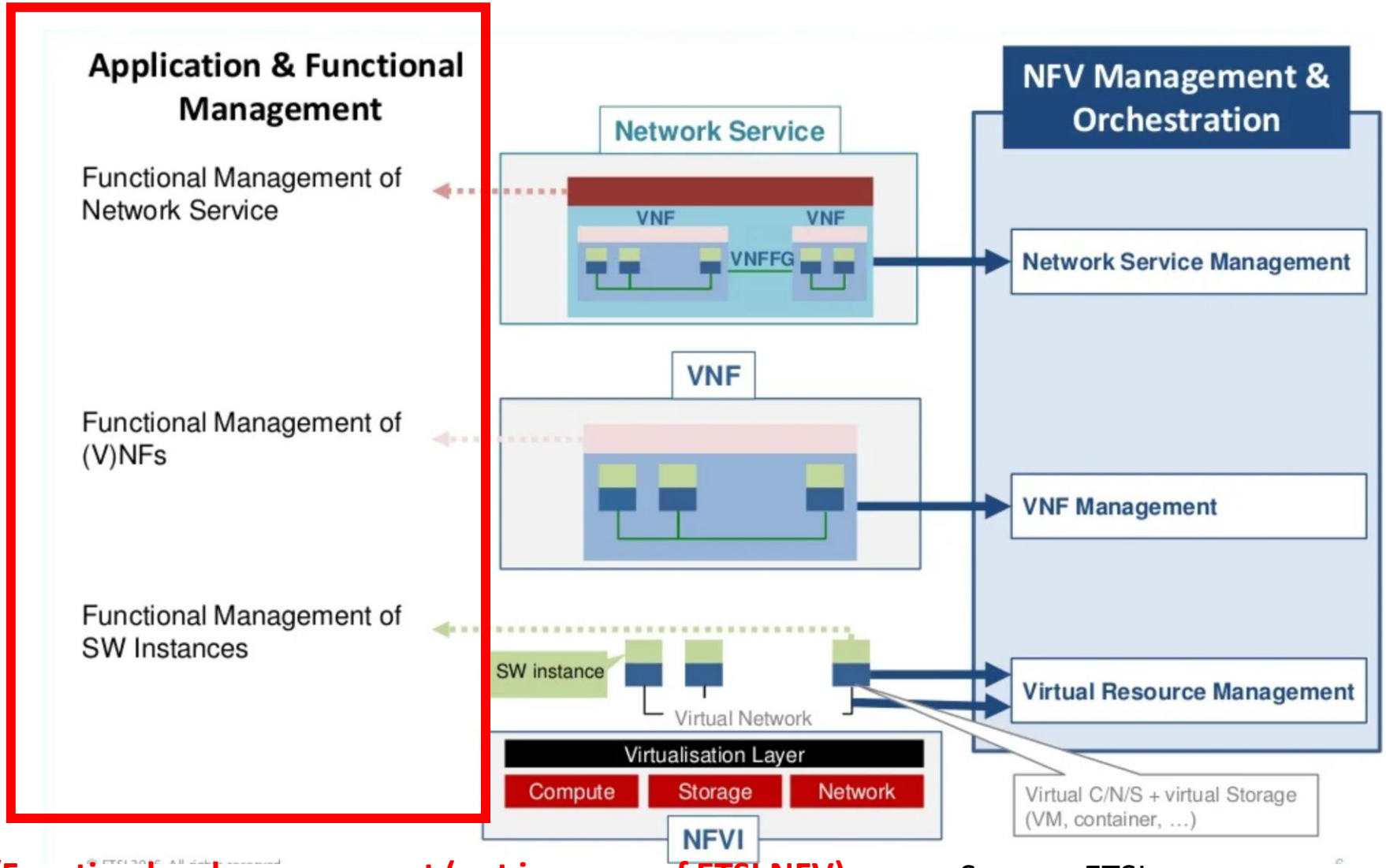
Overall Concepts and how they interconnect

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

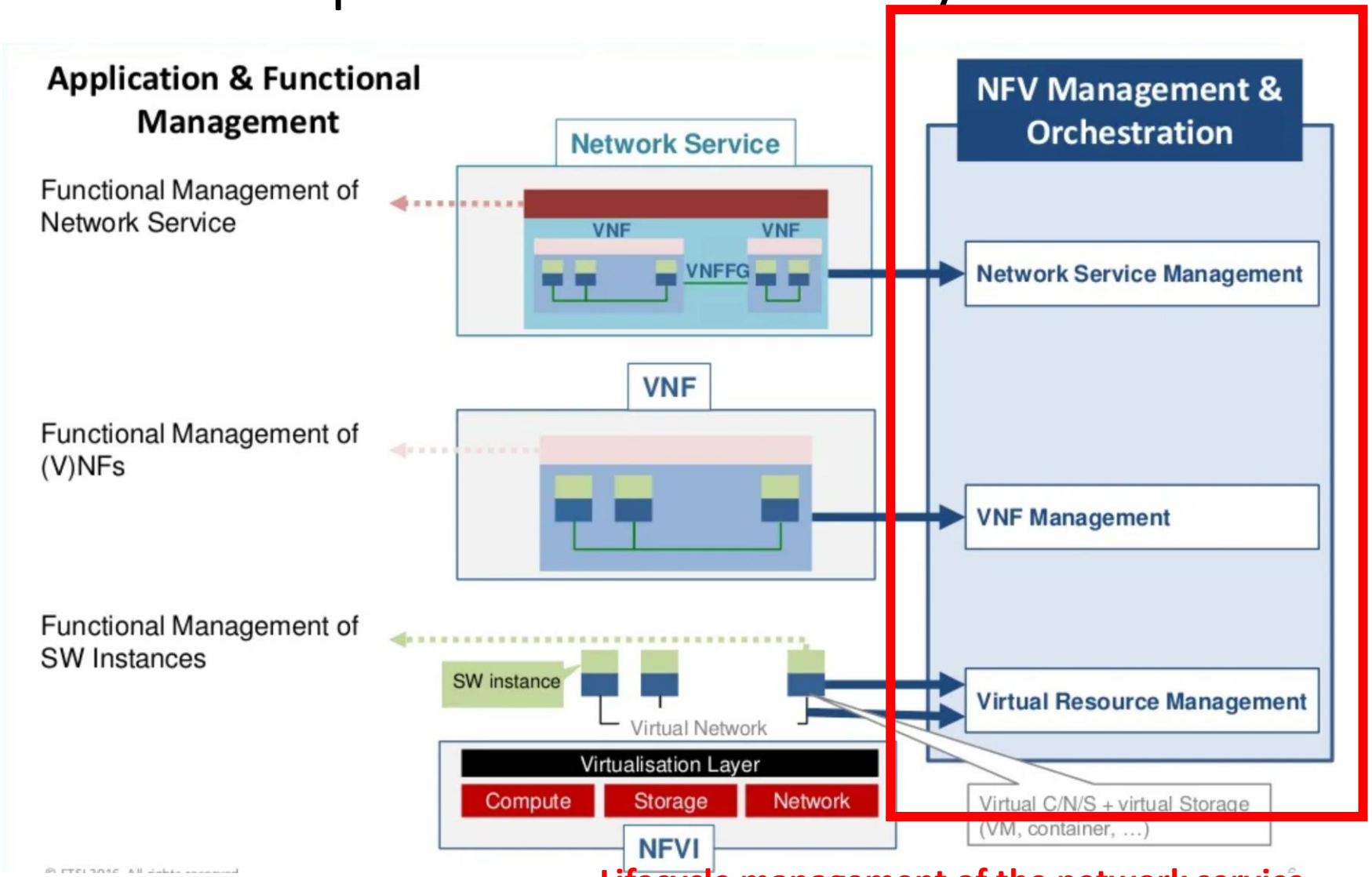
Overall Concepts and how they interconnect



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

Source: ETSI

Overall Concepts and how they interconnect



Source: ETSI

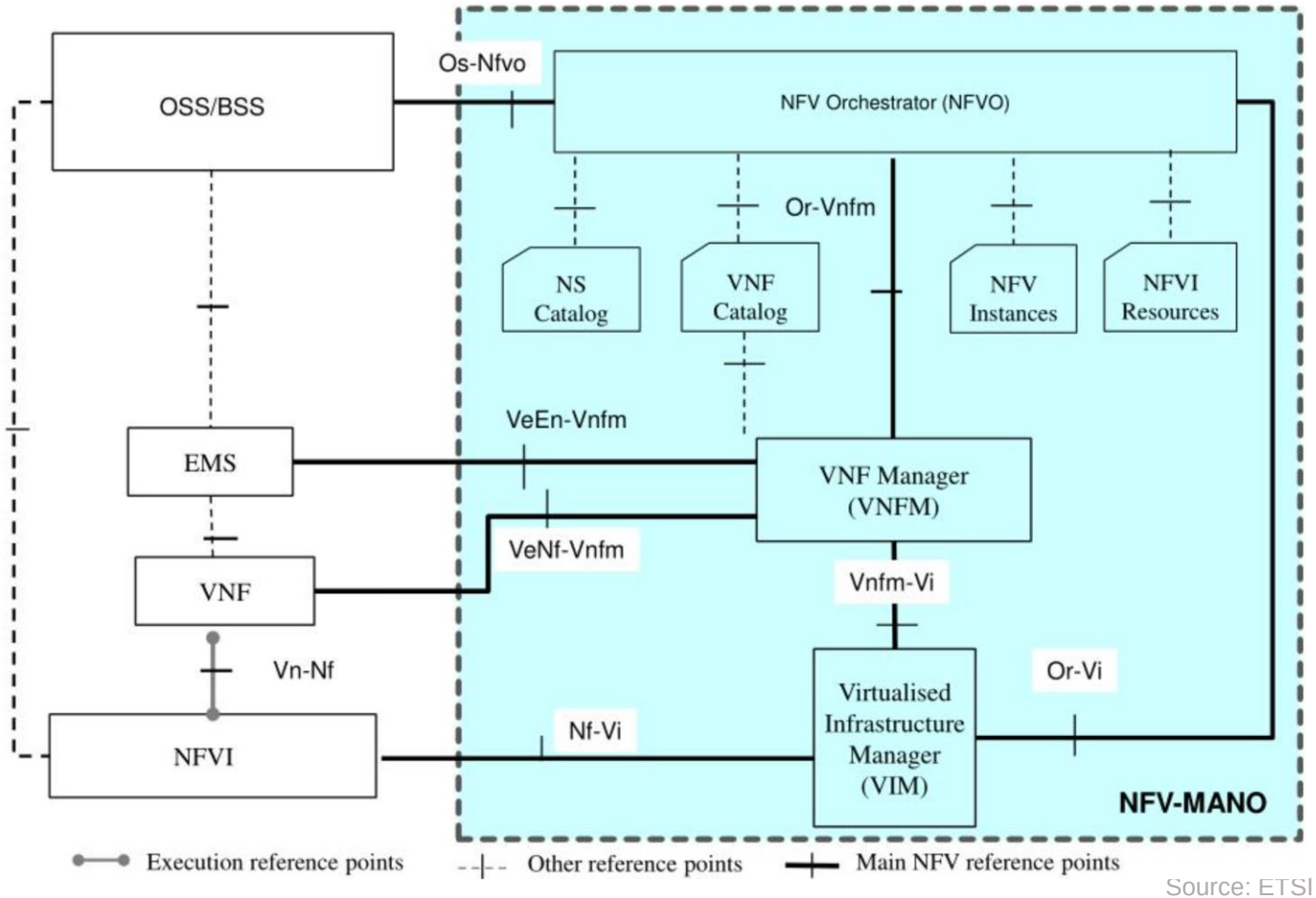
Lifecycle management of the network service

NFV Concept summary

- No longer isolated single purspose physical network services and functions
- A standards-based approach to manage network services, using virtualization infrastructure
- Network services are built from VNF aggregation
- VNFs have their own lifecycle
- But this is managed then at the NS level
  - And at the inter-NS level too (both require orchestration)
- There is the need for management (and orchestration) of the network services
  - Separate from the management of the applications/services themselves

Network Function Virtualization

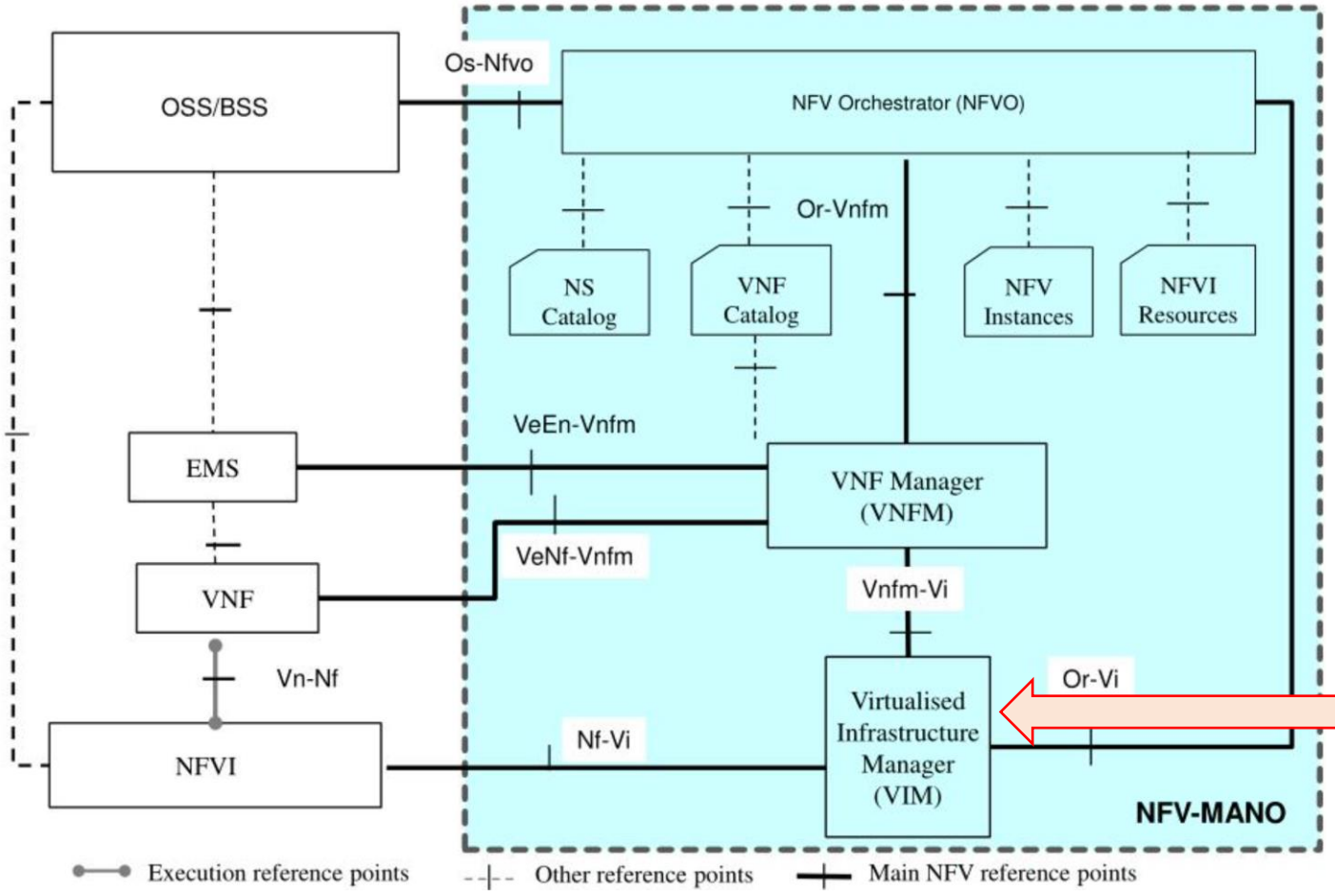
reference architecture



Source: ETSI

Network Function Virtualization

reference architecture

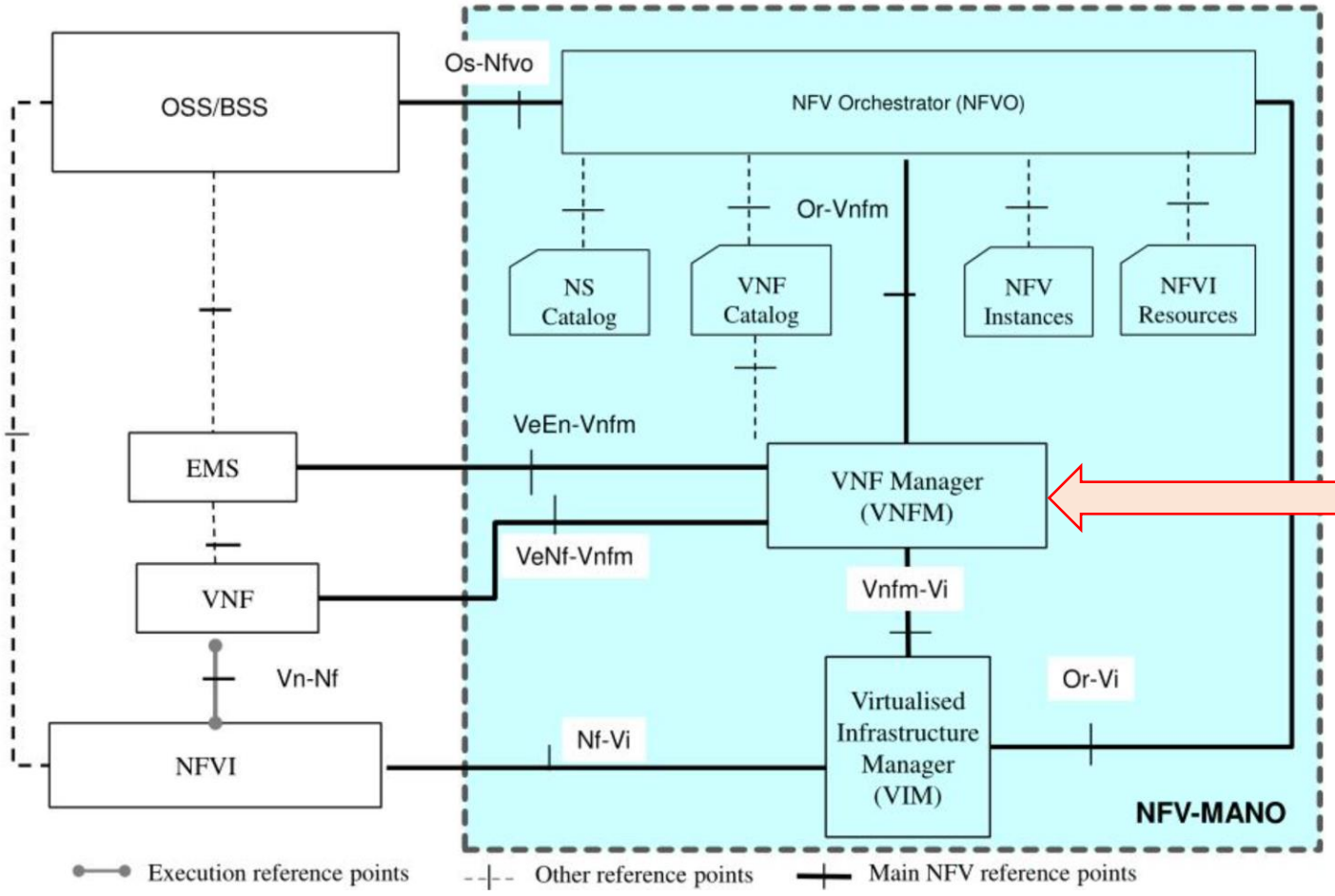


- Manages the NFV infrastructure resources (compute, network and storage) in one or more NFVI-PoPs
  - NFV Infrastructure Point of Presence
- Exposes virtualized resource management interfaces/APIs to the VNFM and NFVO
  - Management and Orchestration
- Sends virtualized resource management notifications to the VNFM and the NFVO

Source: ETSI

Network Function Virtualization

reference architecture



- Manages the lifecyle of VNFs
- Manages VNF initial configuration via the interfaces exposed by the VNF
- Exposes VNF lifecycle management interfaces/APIs to the VNF; EM and NFVO
- Sends VNF lifecycle management notifications to the VNF, EM and NFVO
- Manages virtualized resources associated to the VNF it manages via the interfaces exposed by the VIM or NFVO

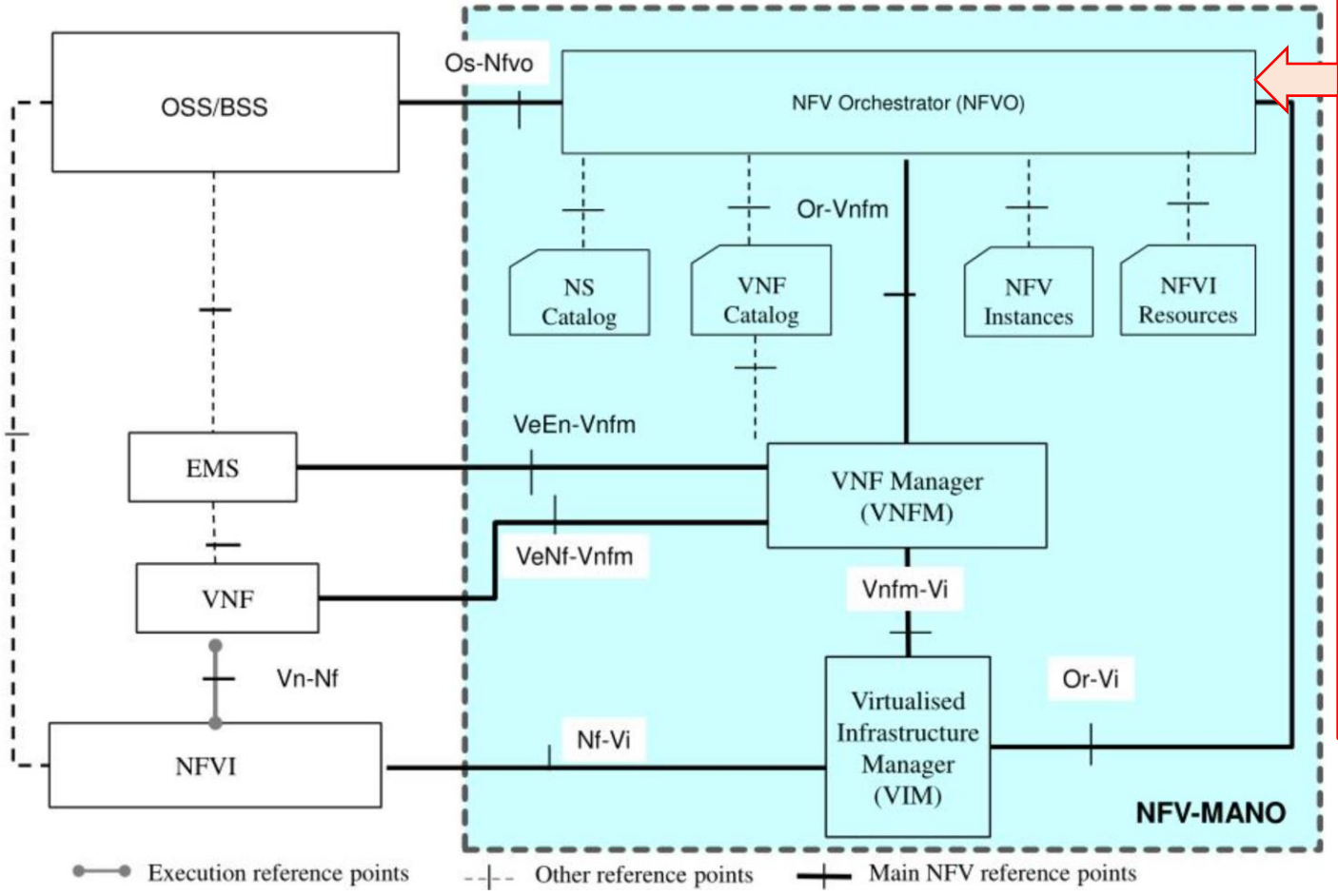
Source: ETSI

VNF Descriptor

- One very important tool used by the VNF Manager (VNFM)
- VNF Deployment template that specifies
  - How the VNF needs to be instantiated
  - How many VM's the VNF has
  - How those VM's are connected
  - What are the external connection points
  - What are the requirements of the VNF, the virtualized resources, etc.
  - Contains scripts that define for VNF scaling, healing, …
- Provided by the VNF vendor
- When the VNF is onboarded, the VNFD is onboarded in the VNFM

Network Function Virtualization

reference architecture



- Manages the lifecyle of NS
- Exposes NS lifecycle management interfaces to the OSS/BSS
- Sends NS lifecycle management notifications to the OSS/BSS
- Manages virtualized resources via the interfaces exposed by the VIM
- Exposes virtualized resource management interfaces to the VNFM
- Sends virtualized resource management notifications to the VNFM
- Manages the VNF lifecycle via the interfaces exposed by the VNFM

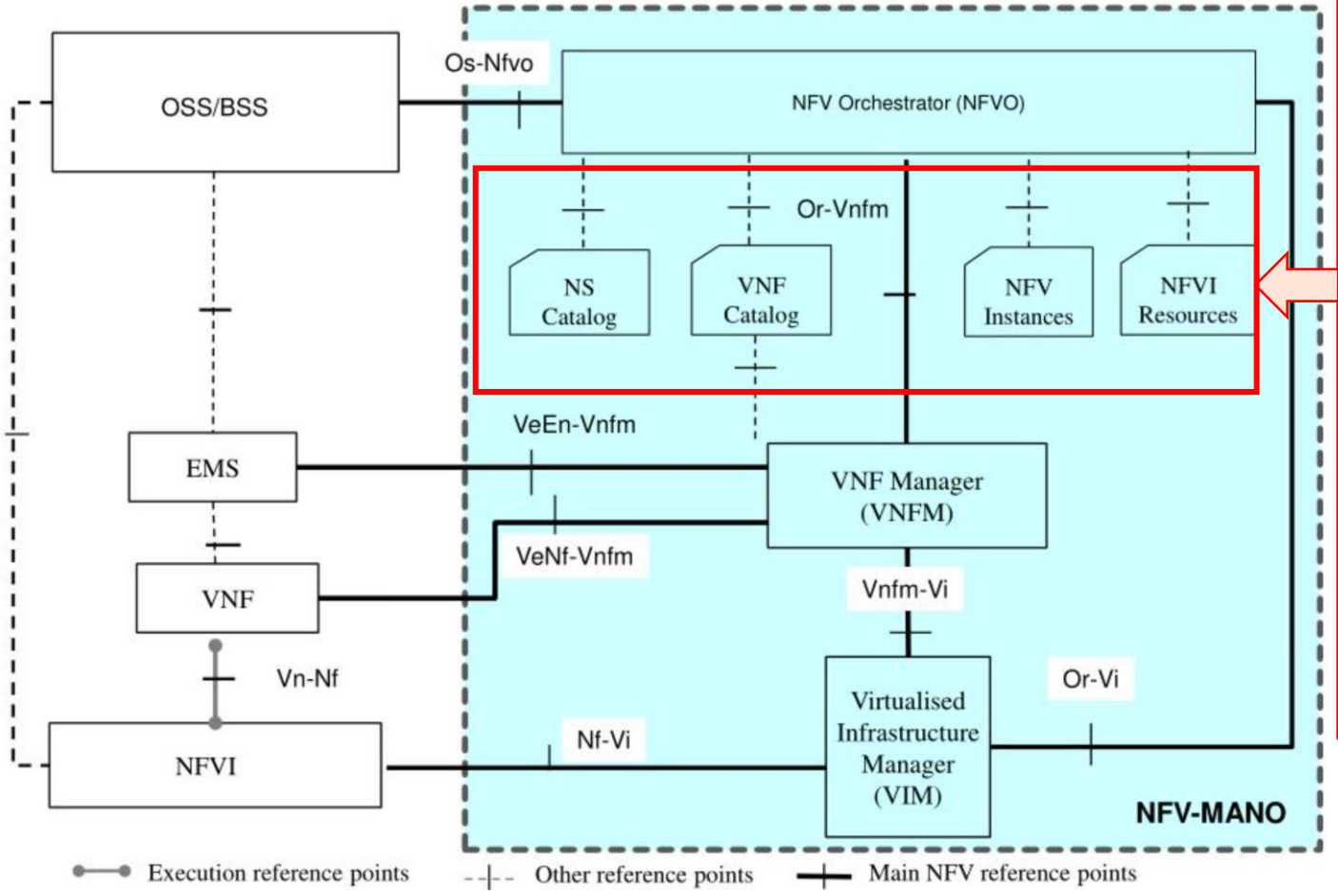
Source: ETSI

NS Descriptor

- Template that describes how the NS is composed
  - In terms of VNF
  - The VNFFG
  - Scripts and indicators that defines what the orchestrator needs to do when certain events or indicators are received
    - These appear during the lifecycle management of the NS

Network Function Virtualization

reference architecture

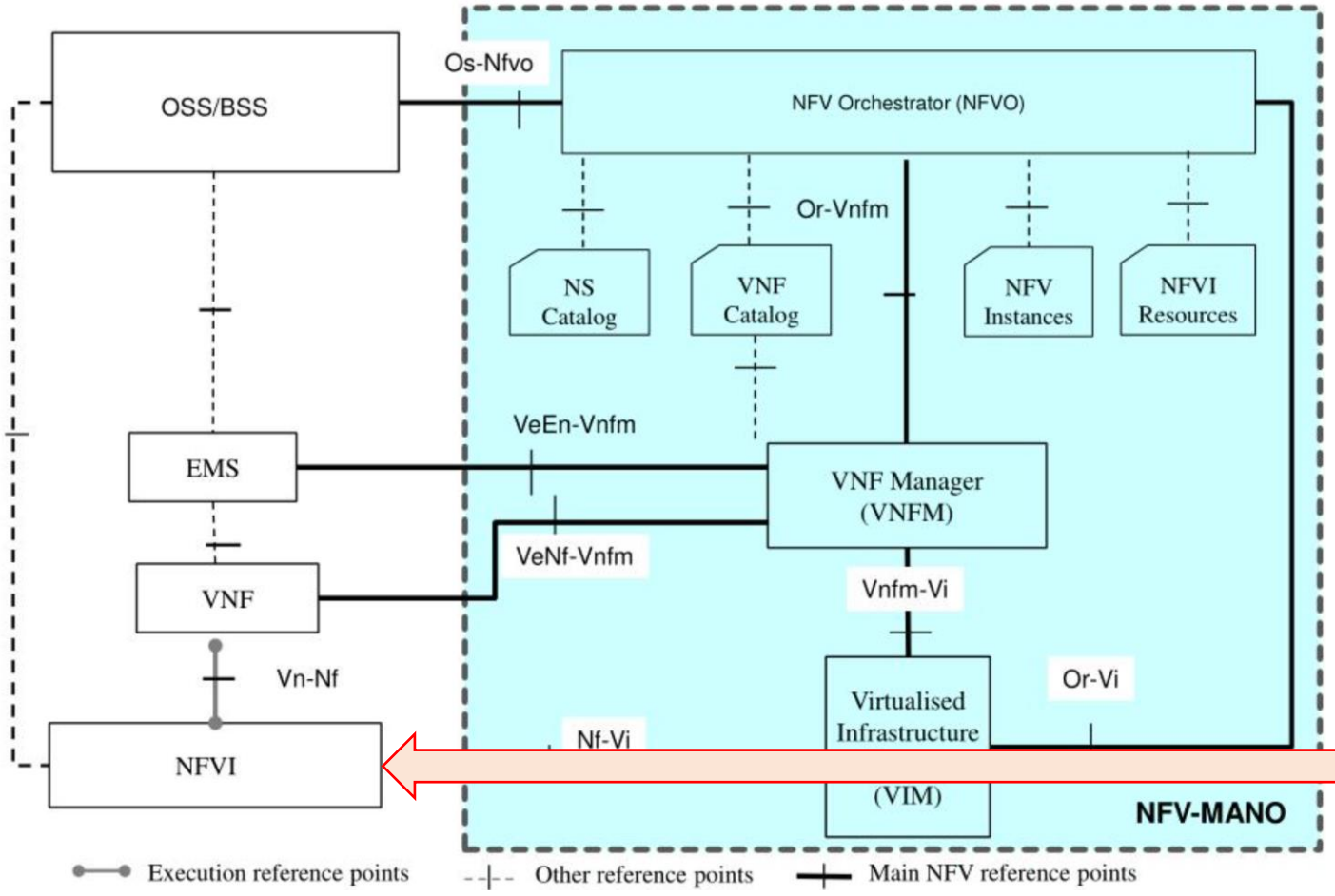


- Catalogues
  - NS Catalog – Network Services that can be instantiated in this NFVI, along with all the information for its instantiation (i.e., VNFFG, lists of VNFs, configurations, etc.)
  - VNF Catalog – List of VNFs available at the NFVI and their lifecycle characteristics
- Repositories
  - NFV Instances – Indication of which NFV instances are operating
  - NFVI Resources – provides information to the orchestrator about the NFVI resources

Source: ETSI

Network Function Virtualization

reference architecture

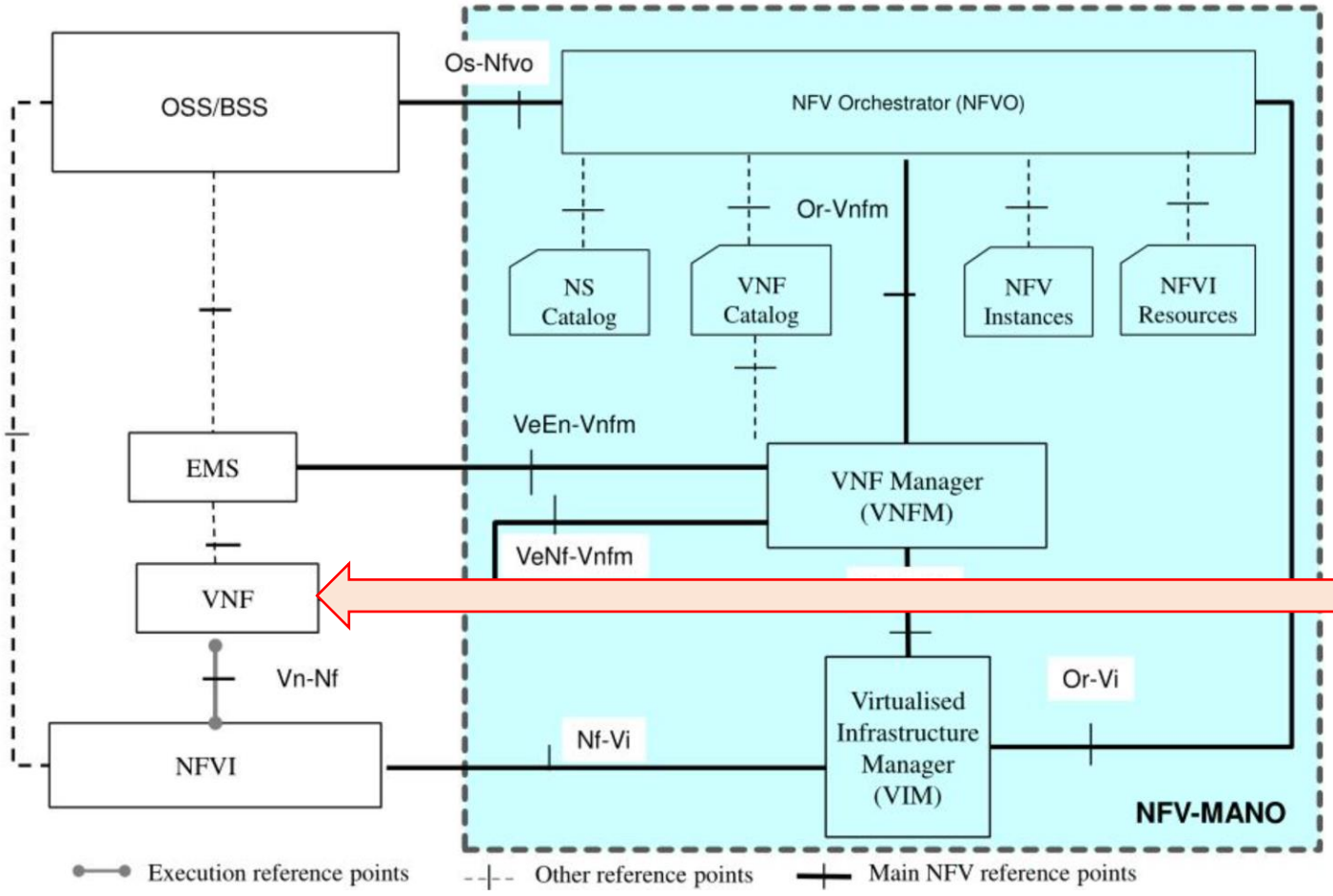


Source: ETSI

- Network Function Virtualization Infrastructure
- The actual infrastructure
- Servers
- Switches
- Routers
- Racks

Network Function Virtualization

reference architecture

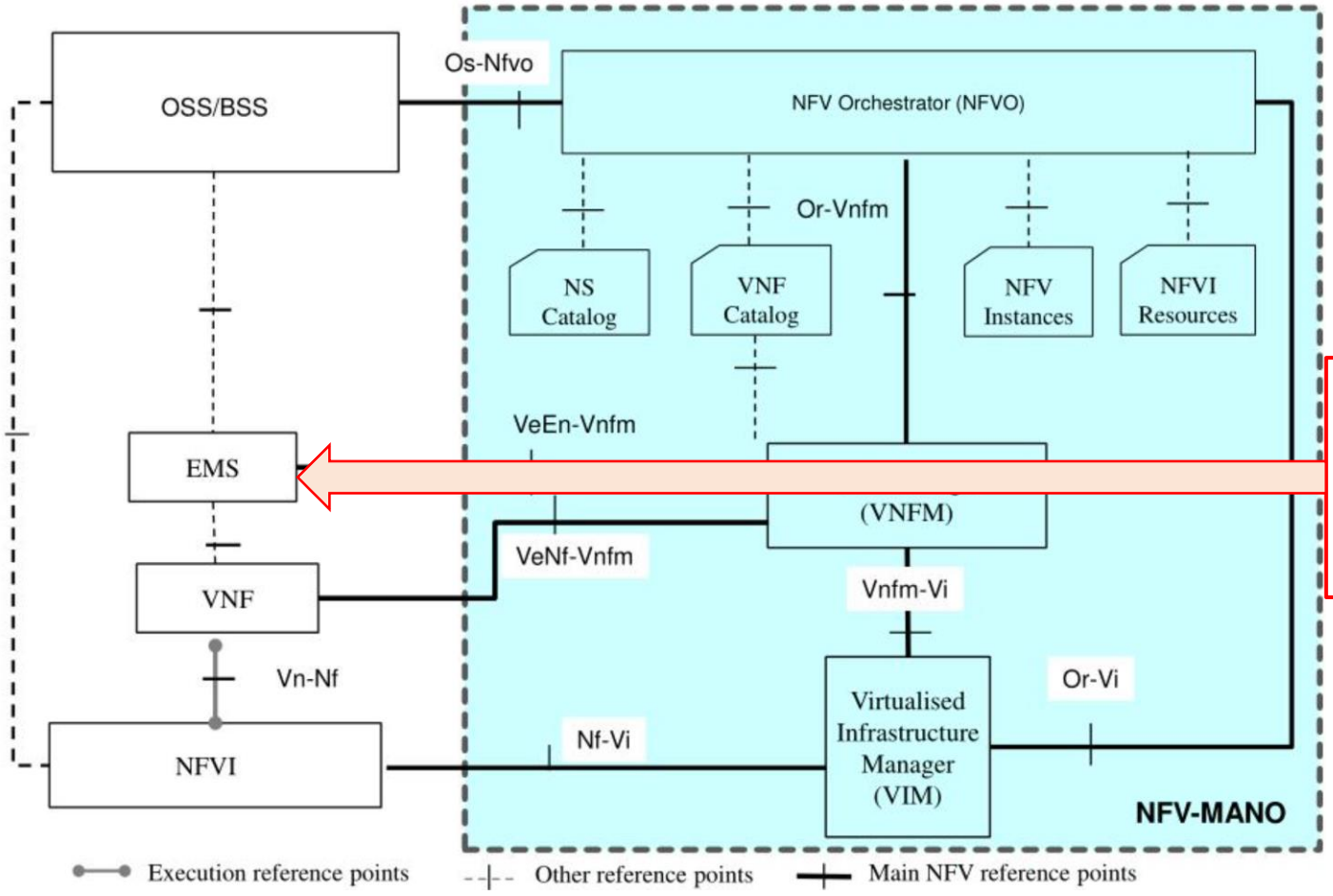


- Actual VNF application
- Router, PGW, Firewall, etc.

Source: ETSI

Network Function Virtualization

reference architecture

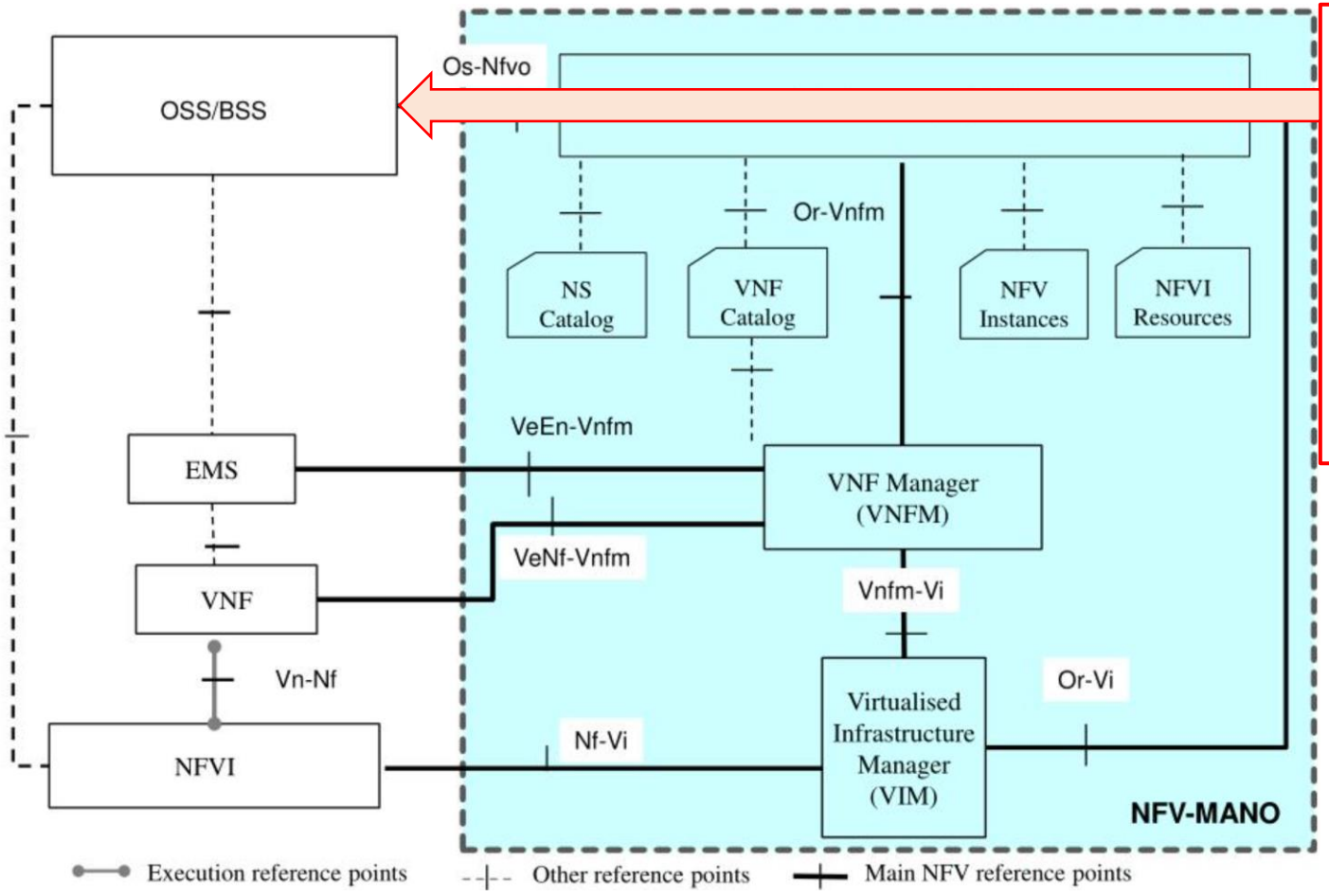


- Element Management Service/Function
- Management to the fault, alarm and configuration management associated to the application-level of a VNF

Source: ETSI

Network Function Virtualization

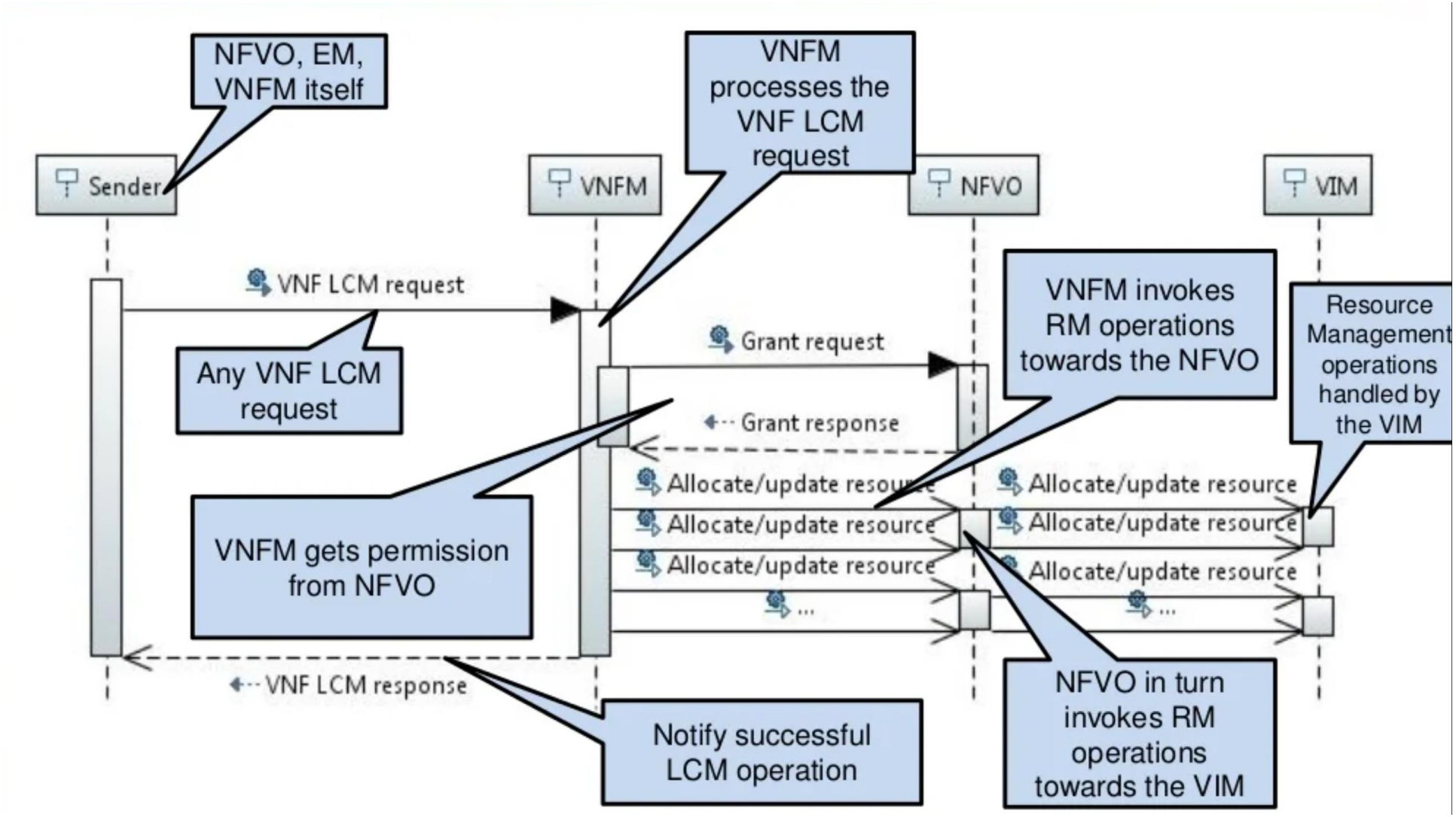
reference architecture



- Operating Support System
- Business Support System
- Full set of tools and systems needed for the end-to-end operation of the network
  - Not in scope of ETSI NFV
- Inventory
- Provisioning
- …

Source: ETSI

Resource Management (Simplified)



Network Services

- Modeled through Information elements
  - VNF Forwarding Graph
  - Physical Network Function
  - Virtual Link
  - Virtual Network Function
  - Network Forwarding Path

NFV Standardisation

- Started in 2012
  - Release 1 (2013-2014, “NFV” series of documents)
    - Use cases (NFV 001)
    - Requirements (NFV 004)
    - Architectural framework (NFV 002)
    - Terminology (NFV 003)
    - Proof of Concepts (NFV-PER 002)
  - Release 2 (2015-2016, “NFV-IFA” series of documents – Interfaces & Arch.)
    - Management aspects
    - Lifecycle management
    - Performance metrics
    - VNF package and software management
    - Information modelling

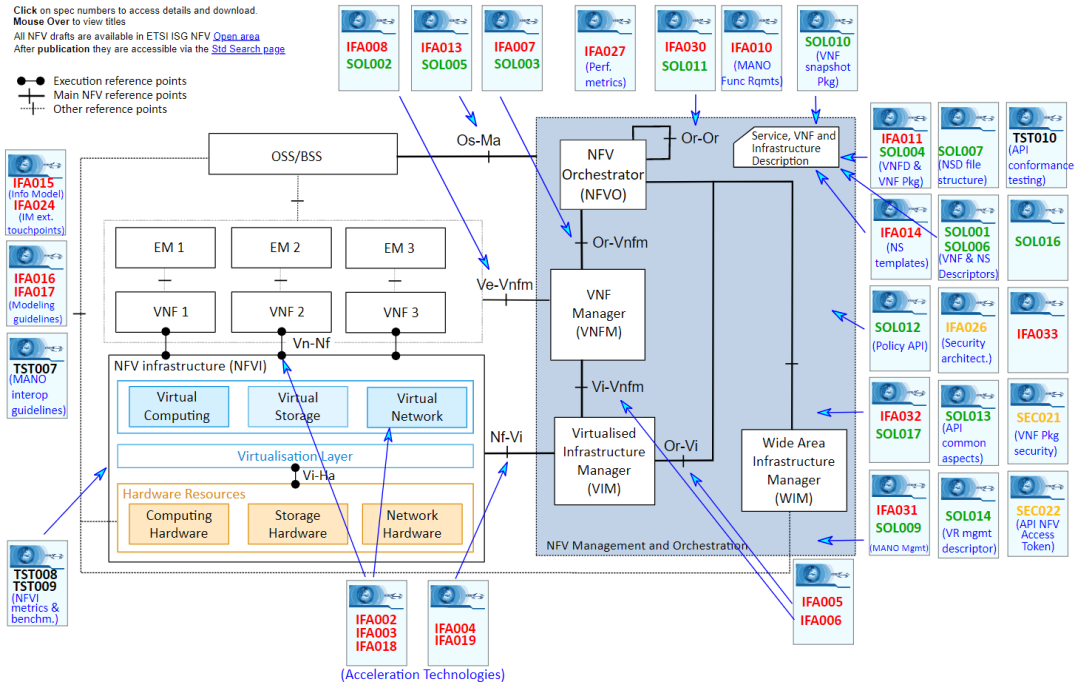
NFV Standardisation

- Started in 2012
  - Release 3 (2017-2018)
    - Interfaces and architecture refinement
    - Support for Edge computing and slicing
    - Virtualization advances (i.e., cloud-native) Support
  - Release 4 (2019-2020)
    - NFVI evolution
    - Automation capabilities
    - MANO evolution and exposure
    - Reliability
    - Policy models

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

World Class Standards



- 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

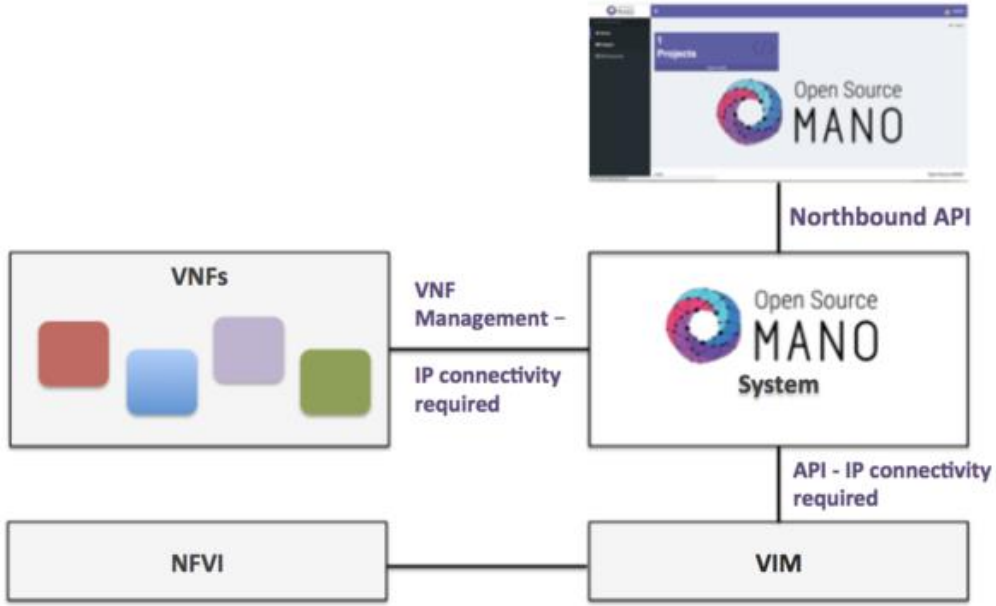
- Managing and coordinating resources and networks needs a special entity
  - NFV Orchestrator
- A powerful tool
  - Spans large numbers of networks
  - … software elements
  - … hardware platforms
- Needs to be able to work with many different standards

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

Software Defined Networking

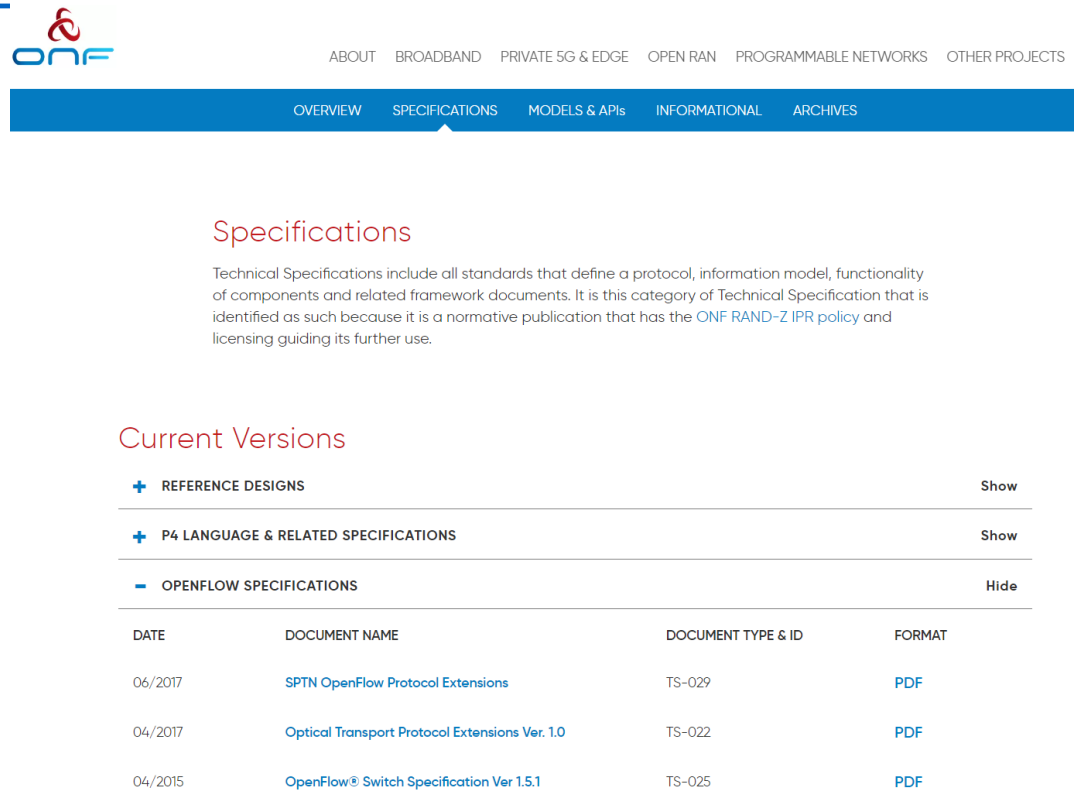
SDN

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)

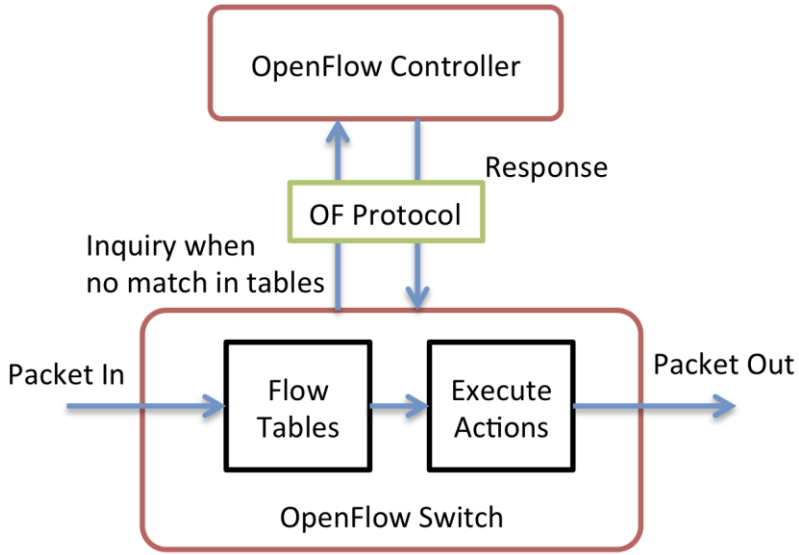
OpenFlow

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



OpenFlow

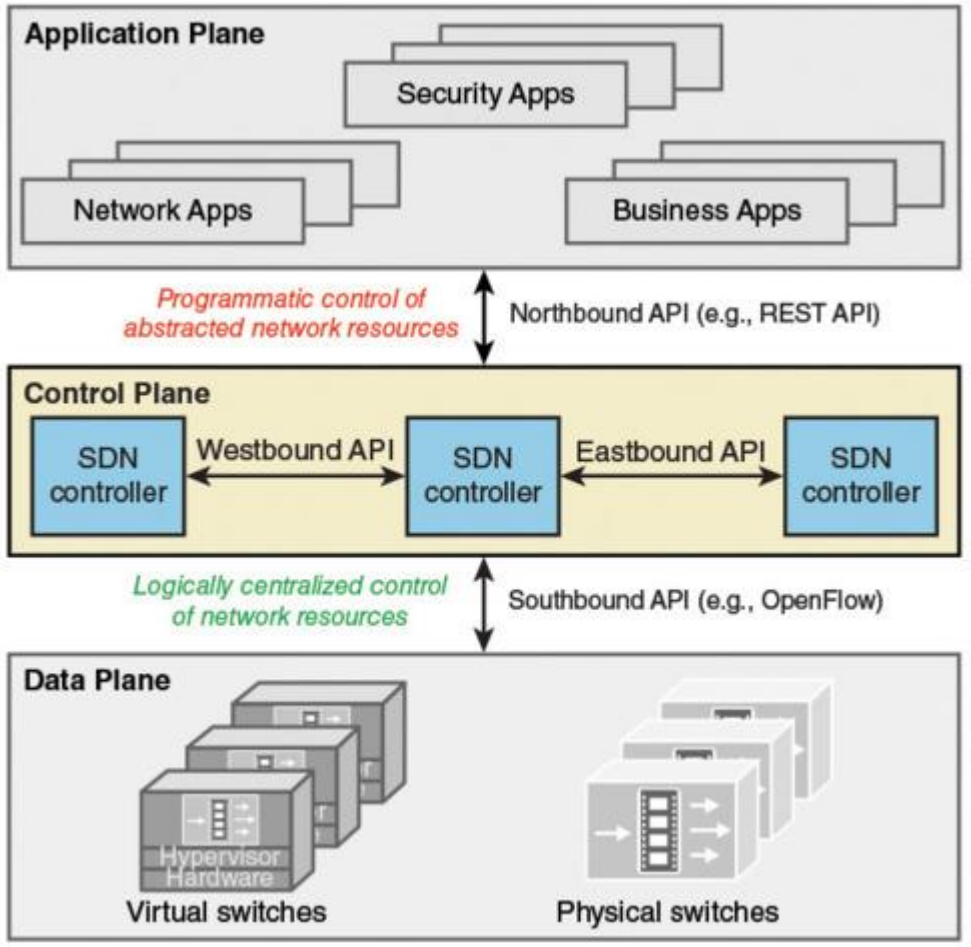
- A protocol existing between SDN switches and a new entity
  - SDN controller
- Allows the SDN controller to manage flow tables in SDN switches
- SDN switch contains
  - Openflow agent
  - Flow tables
  - Performs packet lookup and forwarding
  - Is able to communicate (securely) with the controller
- 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



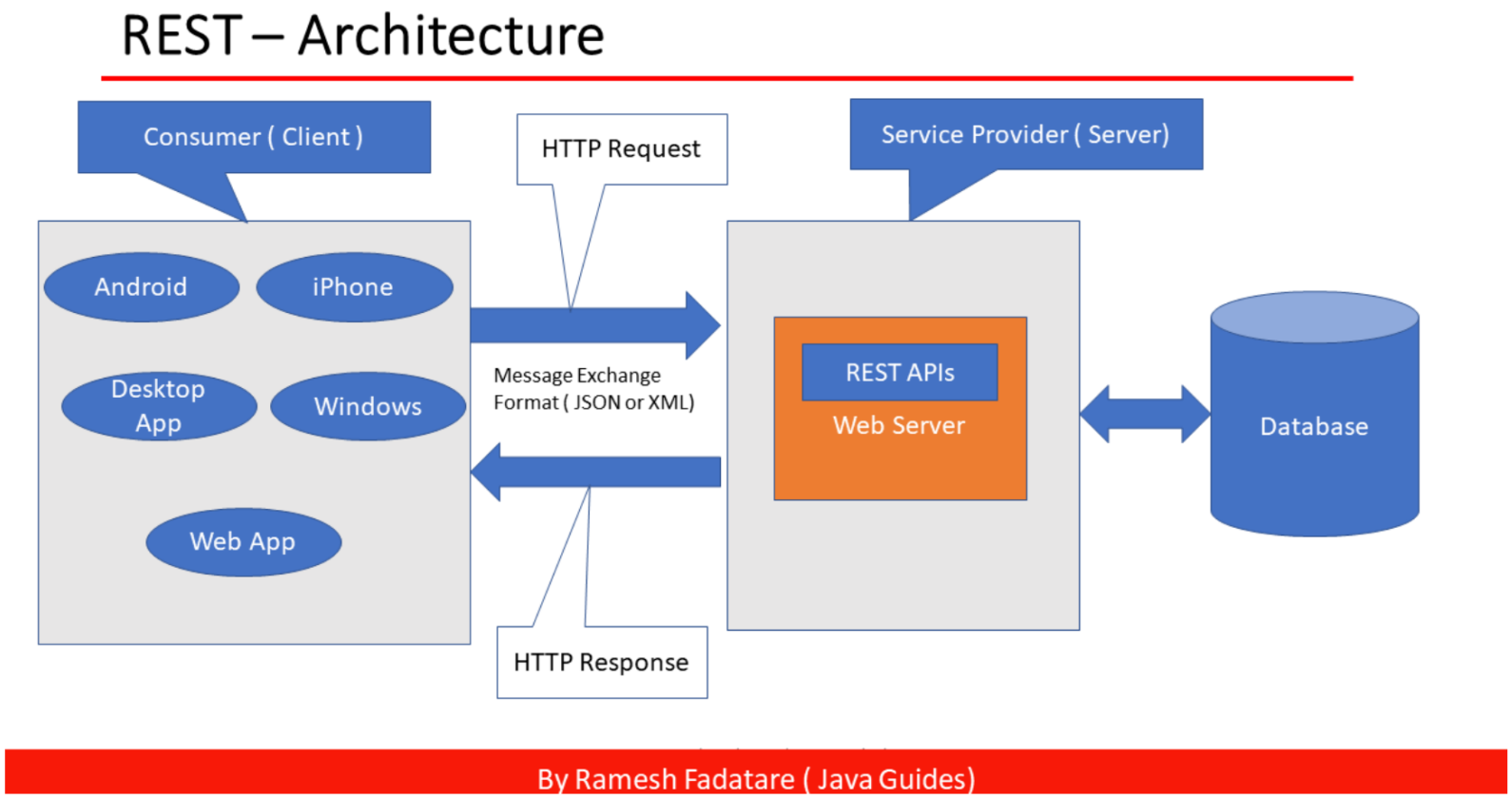
Source: fibre-optic-transceiver-module.com

OpenFlow Controller

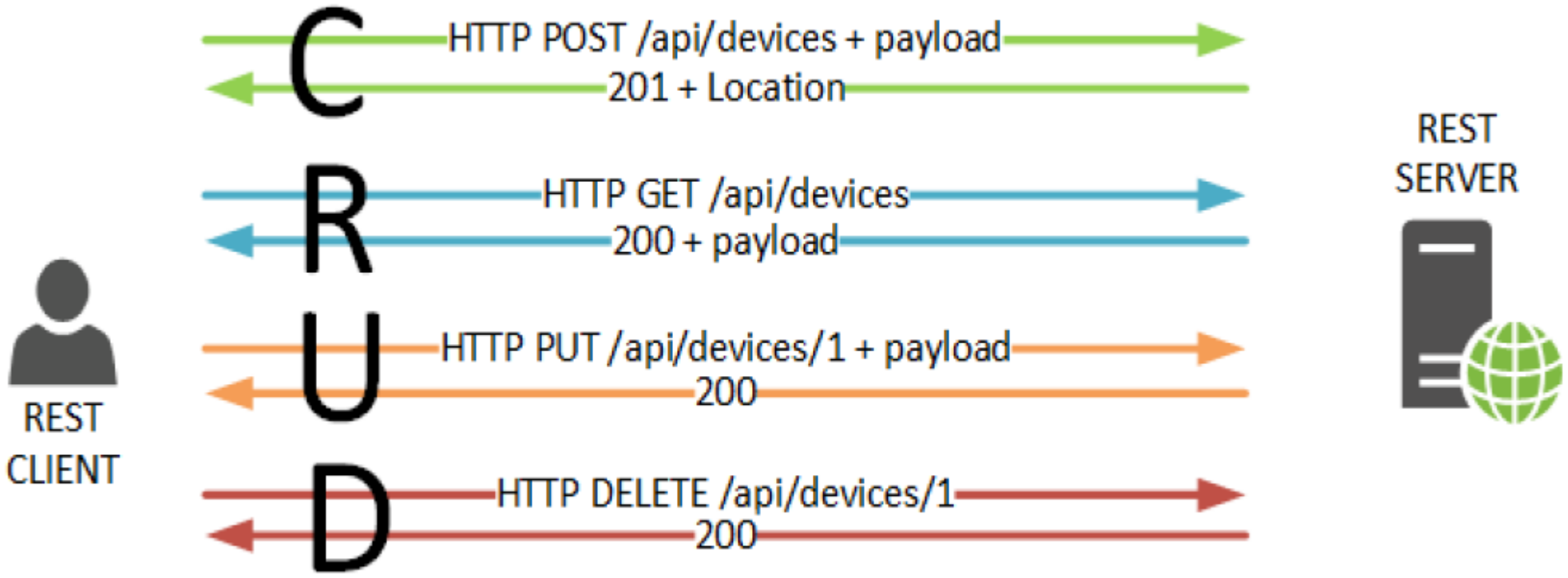
- Controller
  - Service existing in a server, which uses the OpenFlow protocol to interact with SDN switches
  - Formulates flows and programs switches
  - Is able to receive directives from external applications via northbound REST API



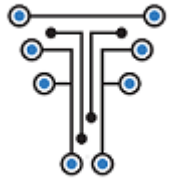
(REST API)


REST – Architecture
Consumer (Client)
HTTP Request
Service Provider (Server)
Android
iPhone
Desktop App
Windows
Web App
Message Exchange Format (JSON or XML)
REST APIs
Web Server
Database
HTTP Response
By Ramesh Fadatare (Java Guides)

(REST API???)


C
HTTP POST /api/devices + payload
201 + Location
R
HTTP GET /api/devices
200 + payload
U
HTTP PUT /api/devices/1 + payload
200
D
HTTP DELETE /api/devices/1
200
REST CLIENT
REST SERVER

(Back to) 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.

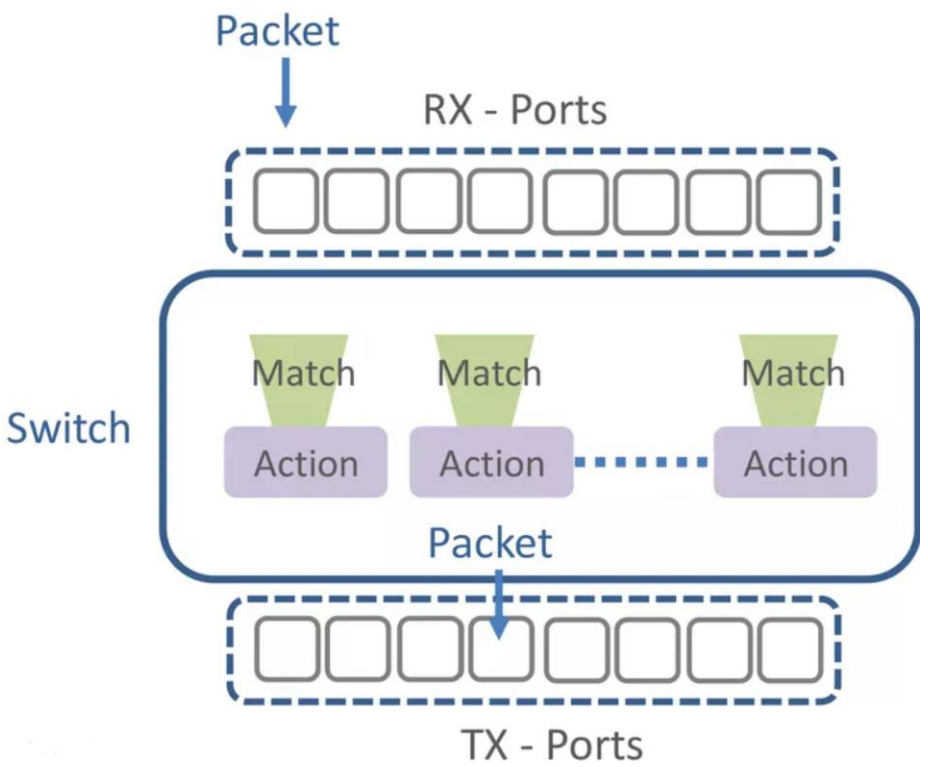
SDN provides the separation of the network control plane from the forwarding plane, allowing more control, adaptability, agility and overall cost reduction. By having a complete view of the network, an SDN controller plays an extremely important role in such networks as it can manage the network structure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

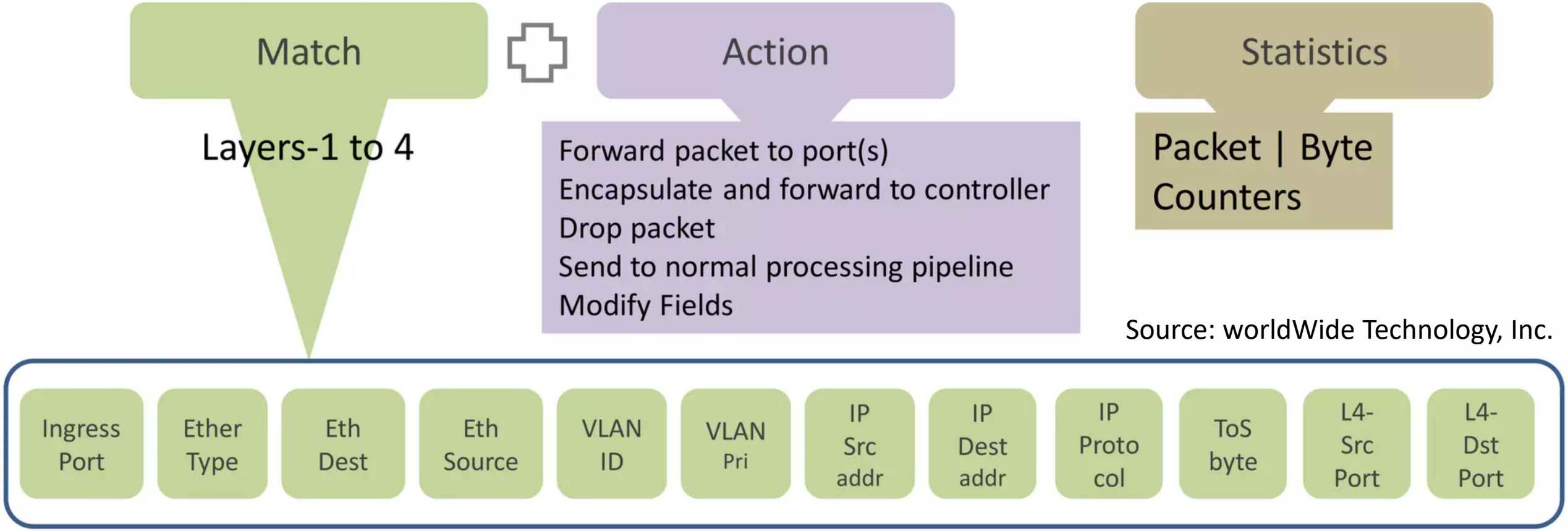
- Has an OpenFlow agent that is able to communicate with the controller
- Processes commands received by the controller
- Dataplane of a switch
  - Ports
  - Flow tables
  - Flows
  - Classifiers (Match)
  - Modifiers and actions
- Packets are matched to flows in flow tables using the match/classifiers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

- Examples of matching
  - TCP port 80 for a /32 IP address (fine matching)
  - All in ingress port 4 (course matching)
- Matching can be done from layer 1 to layer 4 fields
  - OSI Network Stack model
  - **Layer1:** Input port of the switch
  - **Layer2:** Ethernet (L2 Data)
  - **Layer3:** IP (L3 Network, can be subnet masked)
  - **Layer4:** (L4 Transport, TCP/UDP ports)

SDN Switch - Flows

- Example
  - TCP p
  - All in
- Matchin
  - OSI N
  - **Layer**
  - **Layer**
  - **Layer**
  - **Layer**

This allows the SDN Switch to behave like a variety of network devices, such as switch, flow switch, router, firewall…

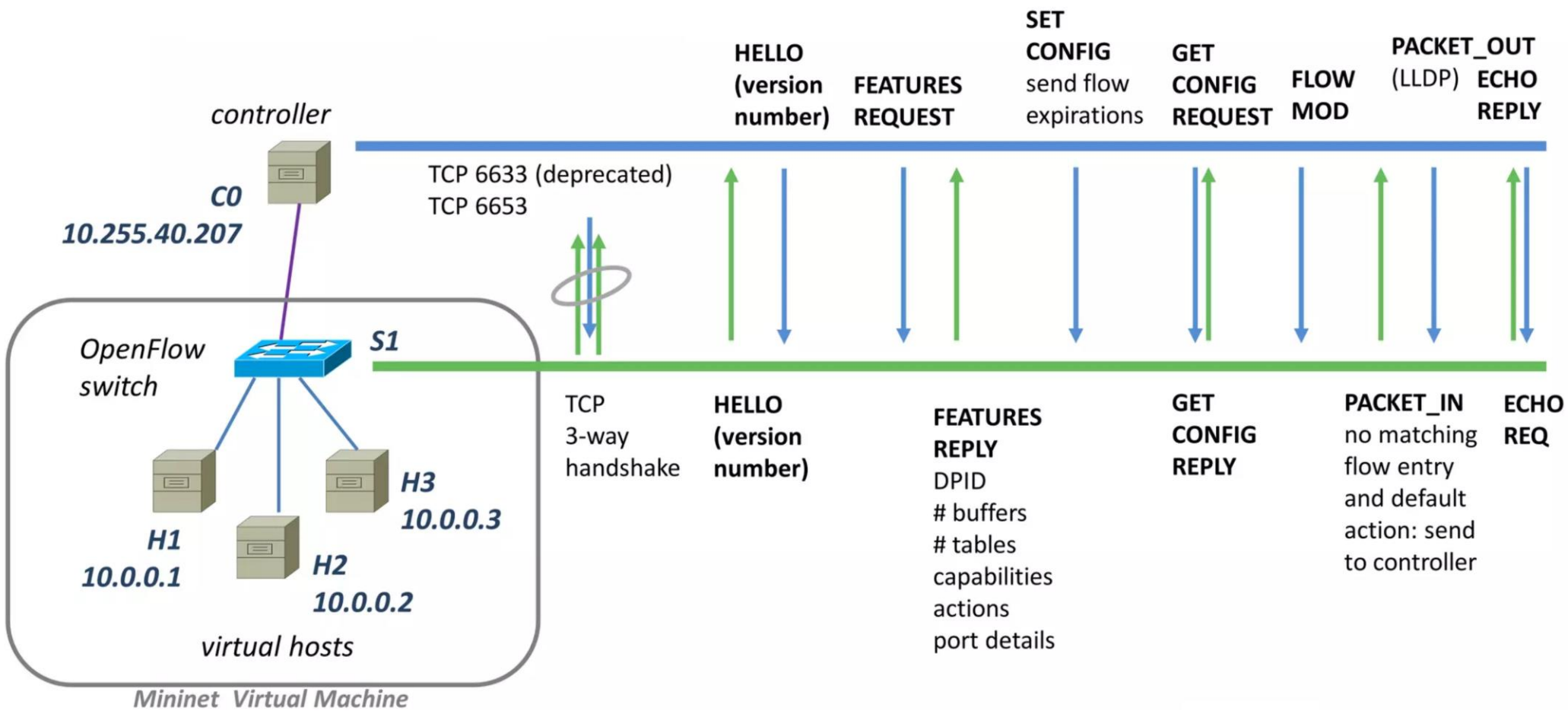
SDN Switch - Actions

- 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)

OpenFlow Protocol

- Supports three types of messages
  - Controller→switch: may not require a response
    - Feature request
    - Configuration
    - Modify state – used to add / delete / modify flows on the switch
  - Asynchronous: switch sends unsolicited message to controller
    - Packet-in
    - Flow removed
  - Symmetric: unsolicited sent by either switch or controller
    - Hello
    - Echo

OpenFlow Switch Start-up example



Source: wirld wide technology Inc.

Message Details

- Controller <-> Switch Connectivity keep alive
  - ECHO_REQUEST / ECHO_REPLY
- Implement flows into the switch’s flow table
  - FLOW_MOD
- Switch acknowledges to the controller that actions were executed
  - BARRIER_REQUEST

Flow insertion into the switch

- Flows can be pre-loaded
  - But this is just like any ordinary non-SDN switch
- SDN supports two other ways, involving interaction between the controller and the switch
  - Reactive
  - Proactive

Reactive flow insertion

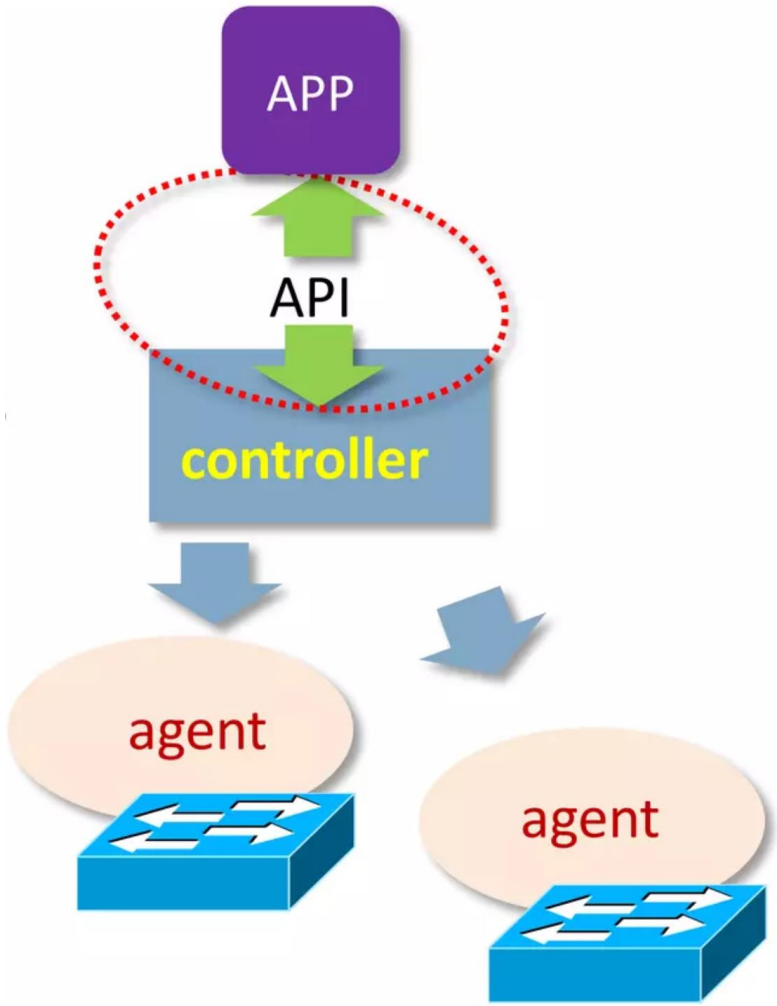
- The flow’s first packet is sent to the controller (PACKET_IN)
- The controller analyses the packet. Based on the packet’s information and directives present in the controller, the controller sends a FLOD_MOD message to the switch, telling it to insert a flow table entry for the packet
- Subsequent packets (in flow) match the entry until idle or hard timeout
- Loss of connectivity between switch and controller limit utility of switch

Proactive flow insertion

- The controller pre-populates flow table in the switch
- There is zero setup time for new flows
- Loss of connection between switches and the controller does not disrupt network traffic
- Switch in proactive forwarding mode only, would default to action = drop
- Northbound REST API used to populate proactive flows.

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

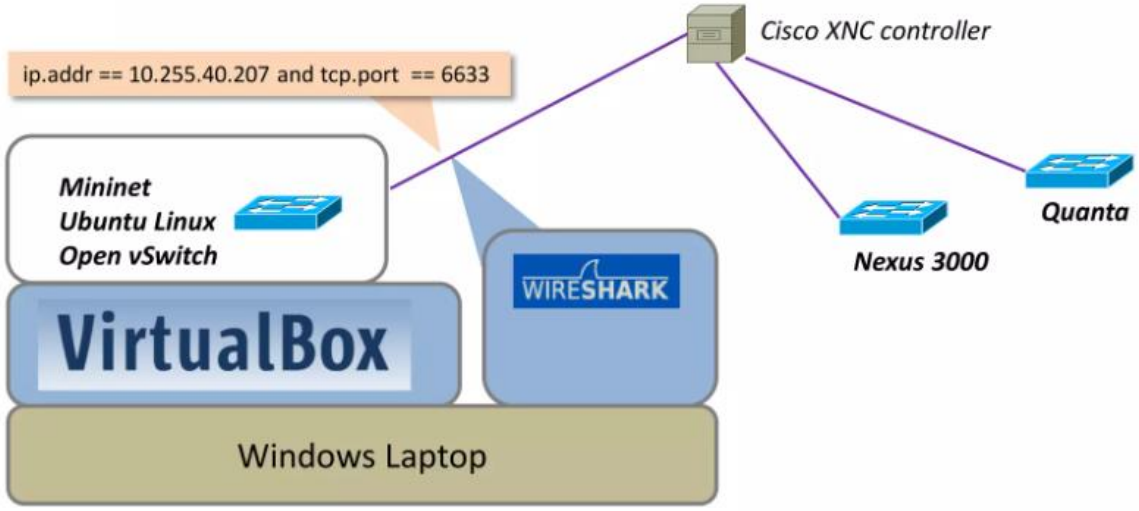


Northbound Protocol Examples

- 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

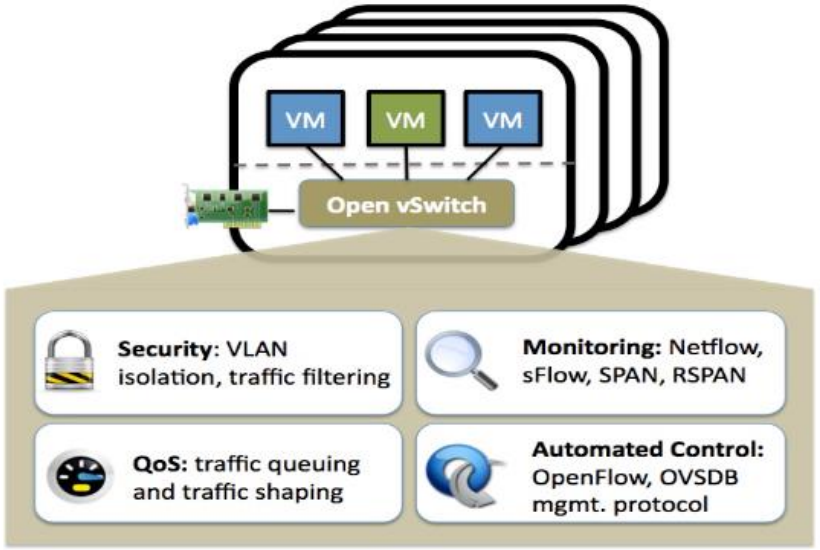
Emulation

- How to “emulate” many switches easily?
  - Mininet
    - An instant Virtual Network on your Laptop
    - http://mininet.org/



Emulation

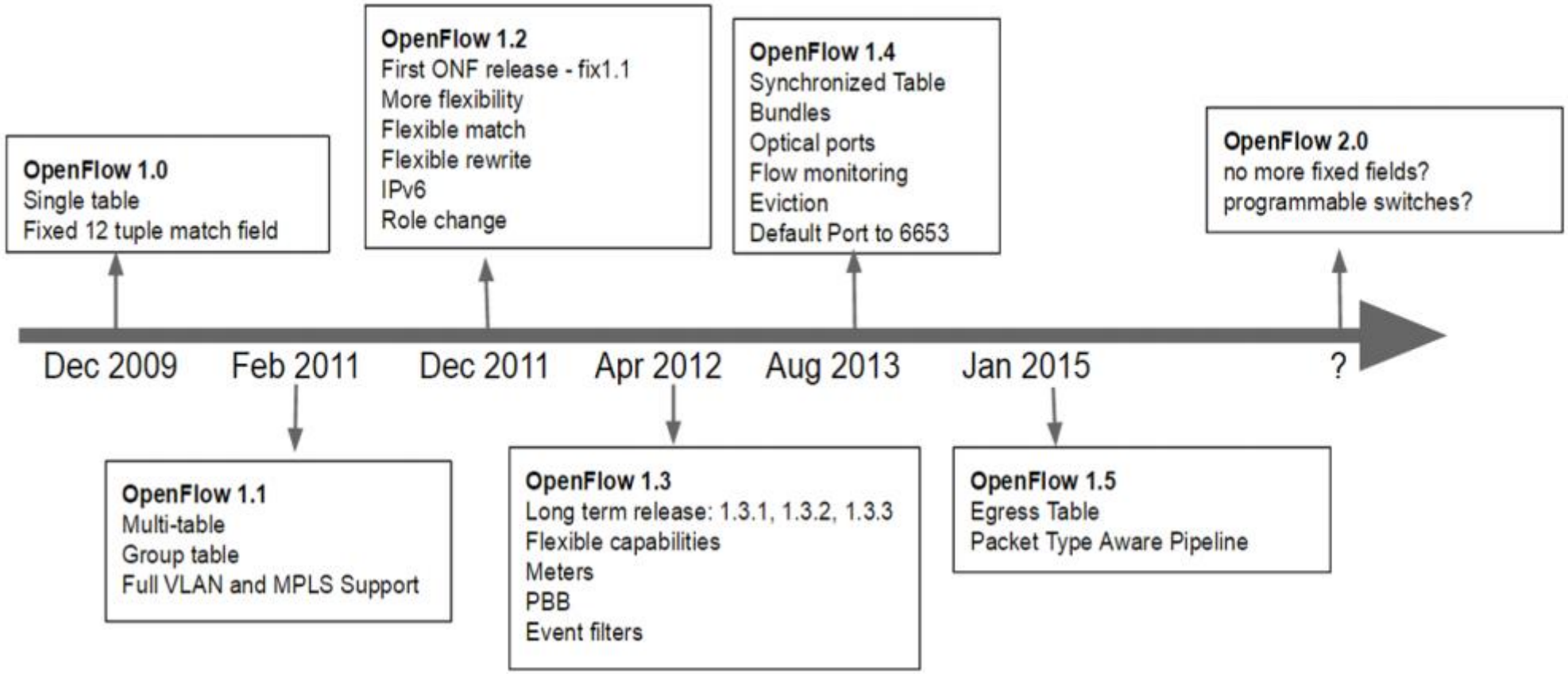
- Open vSwitch
  - https://www.openvswitch.org/
- Virtualized switch
- It is used in production environments (i.e., datacenters)
- You can use it in your projects
- Even with Mininet!



OpenFlow evolution

Version	Major Feature	Reason	Use Cases
1.0 - 1.1	Multiple table	Avoid flow entry explosion	
	Group Table	Enable Applying action sets to group of flows	Load balancing, Failover, Link Aggregation
	Full VLAN and MPLS Support		
1.1 - 1.2	OXM Match	Extend matching flexibility	
	Multiple Controller	HA/Load balancing/Scalability	Controller Failover, Controller Load Balancing
1.2 - 1.3	Meter table	Add QoS and DiffServ capability	
	Table miss entry	Provide flexibility	
1.3 - 1.4	Synchronized Table	Enhance table scalability	Mac Learning/Forwarding
	Bundle	Enhance switch synchronization	Multiple switch configuration
1.4 - 1.5	Egress Table	Enabling processing to be done in output port	
	Scheduled bundle	Further enhance switch synchronization	

OpenFlow Evolution



OpenFlow Problems

- It only controlled flow tables, not the switch pipeline
- Fragmentation adoption
  - Different pace in adopting versions
  - The protocol evolves slowly (i.e., slow adoption of new match fields)
  - Optional fields are not mandatory
- Limited tunnel support
- High codebase footprint
- Limited telemetry
- **It wasn’t designed to be updated**
- **Lack of intelligence at the dataplane**

Also… new switch hardware appeared

- Terabit per second speed
- Application Specific Integrated Cirtuit (ASIC)
- But are very hard to program
- Have a custom, vendor-specific interface
- PISA: Flexible Match+Action ASICs

- A new higher-level interface is needed

Idea to solve issues

- Implement flexible mechanisms for
  - parsing packets
  - Matching header fields
  - … through a common interface
- No longer need to keep extending OpenFlow

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.

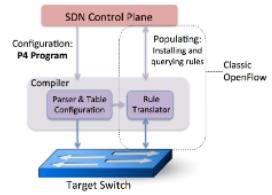


Figure 1: P4 is a language to configure switches.

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)

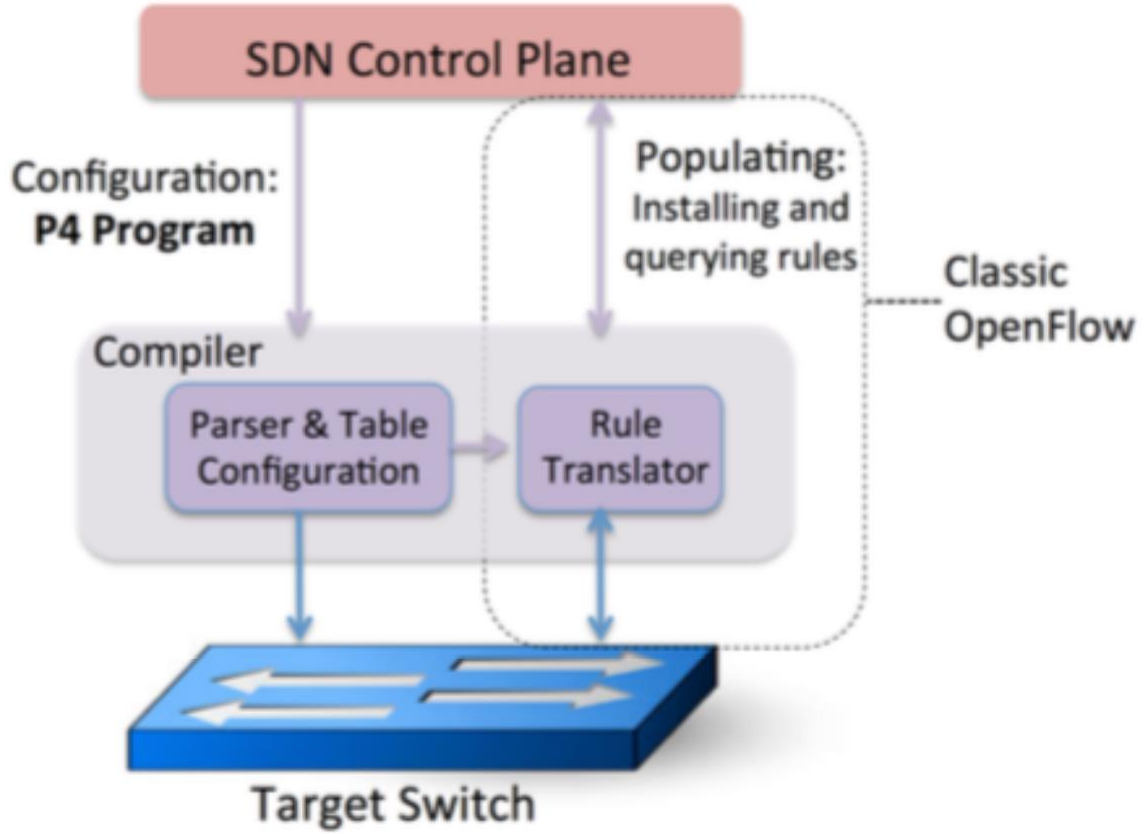
- Benefits
  - New features: add new protocols
  - Reduce complexity: remove unused protocols
  - Efficient use of resources – flexible use of tables
  - Greater visibility – new diagnostic techniques, telemetry, etc.
  - Software style development – rapid design cycle, fast Innovation, fix data plane bugs in the field
  - You keep your own ideas!

Programming Protocol-Independent Packet Processors (P4)

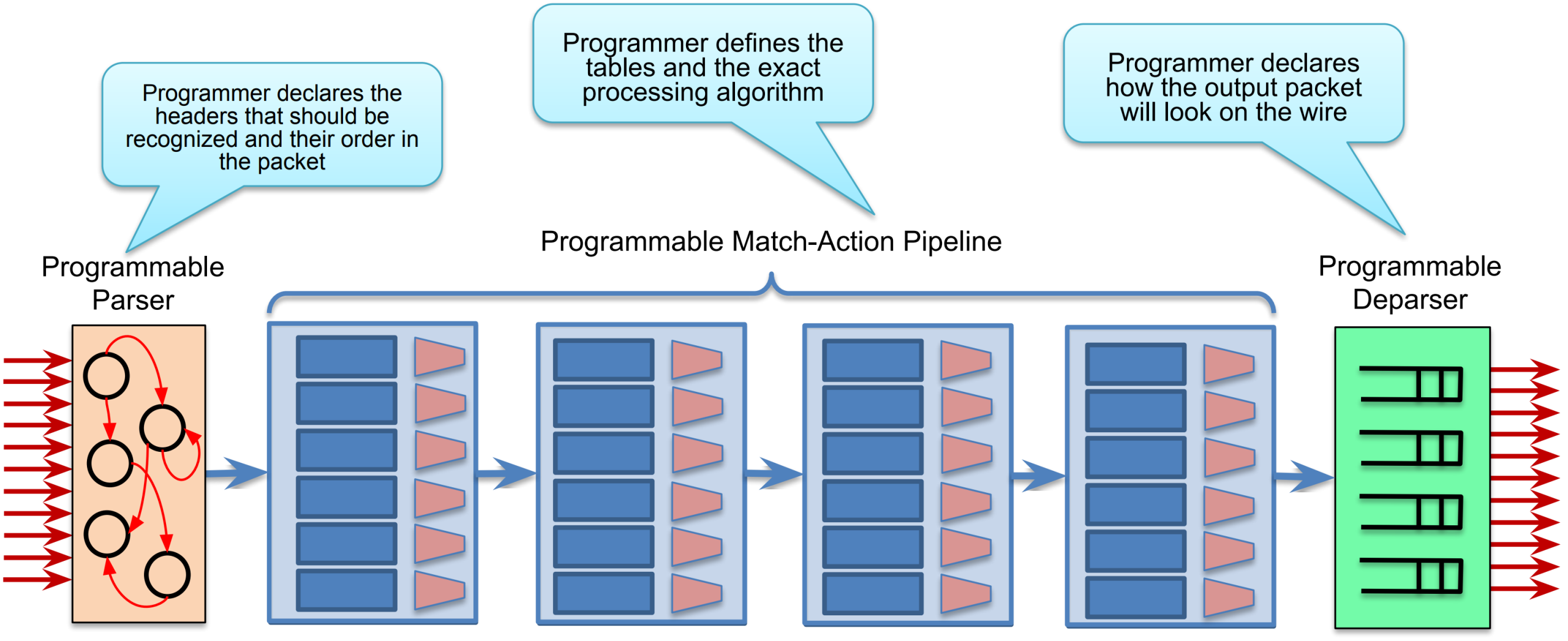
- Evolution
  - P4_14
    - Original version of the language
    - Assumed specific device capabilities
    - Good only for a subset of programmable switch/targets
  - P4_16
    - More mature and stable language definition
    - Does not assume device capabilities, which instead are defined by target manufacturer via external libraries/architecture definition
    - Good for many targets, e.g., switches and NICS, programmable of fixed-function

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



Architecture of a programmable switch



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 */
    }
}
```

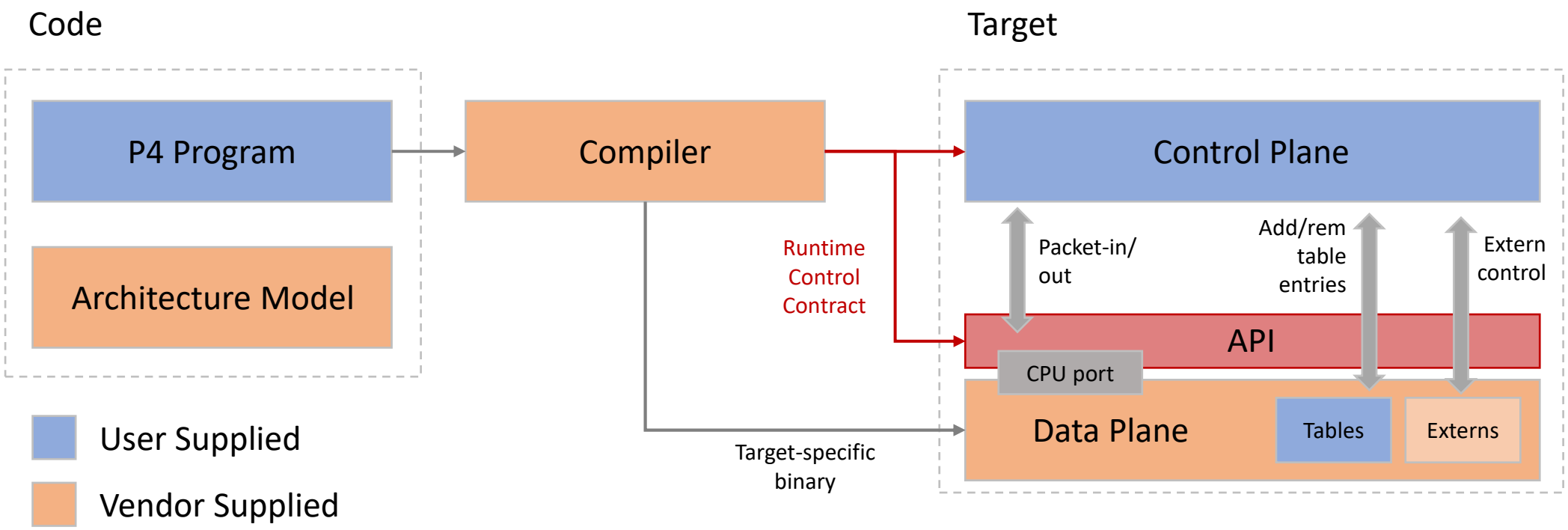
Match semantics

- **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

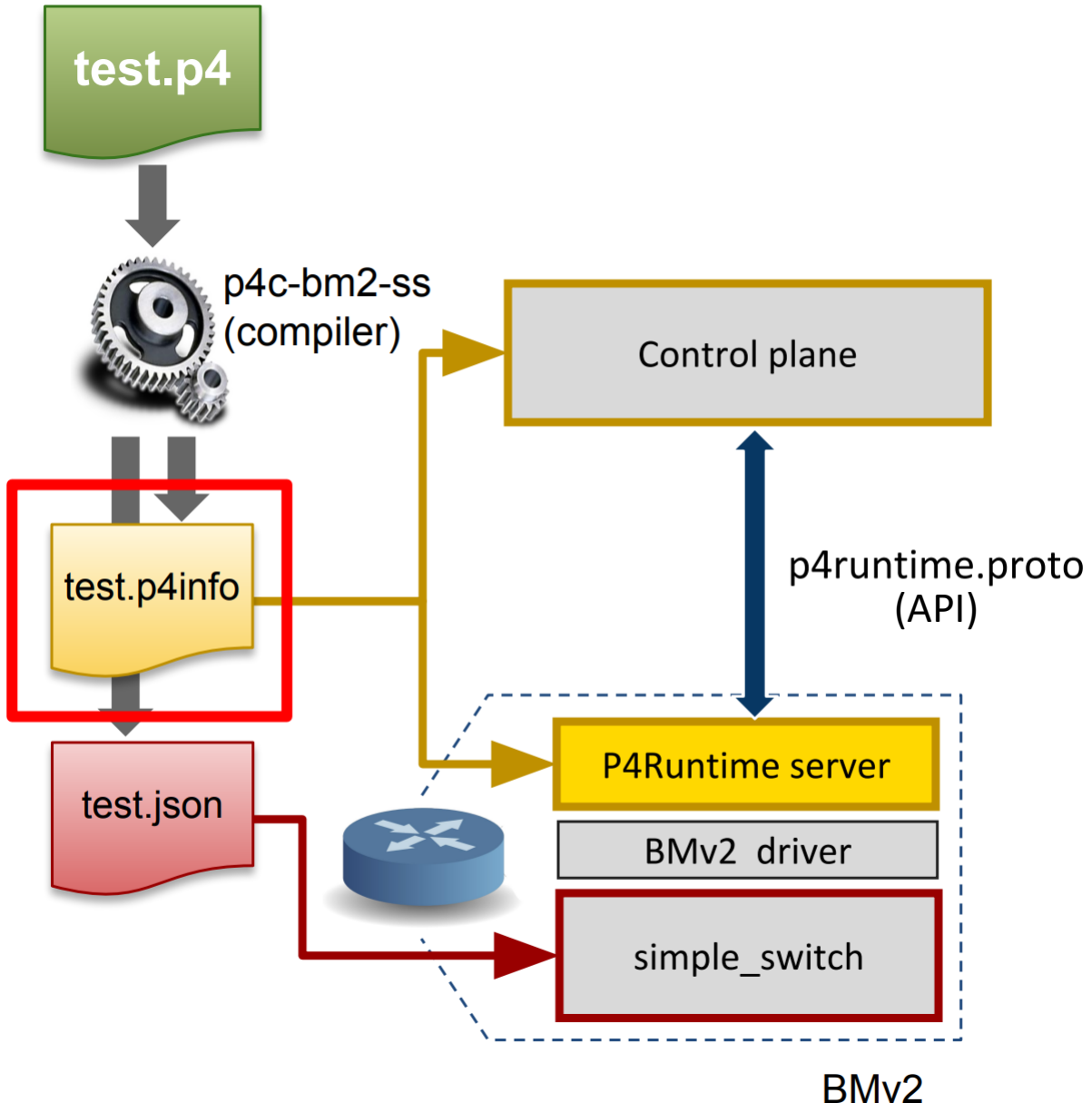
- Framework for runtime control of P4-defined data planes
  - Open-source API and a server implementation
- P4 program-independent
  - API doesn’t change with the P4 program
- Enables field-reconfigurability
  - Ability to push new P4 program without recompiling the software stack of target switches
- 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

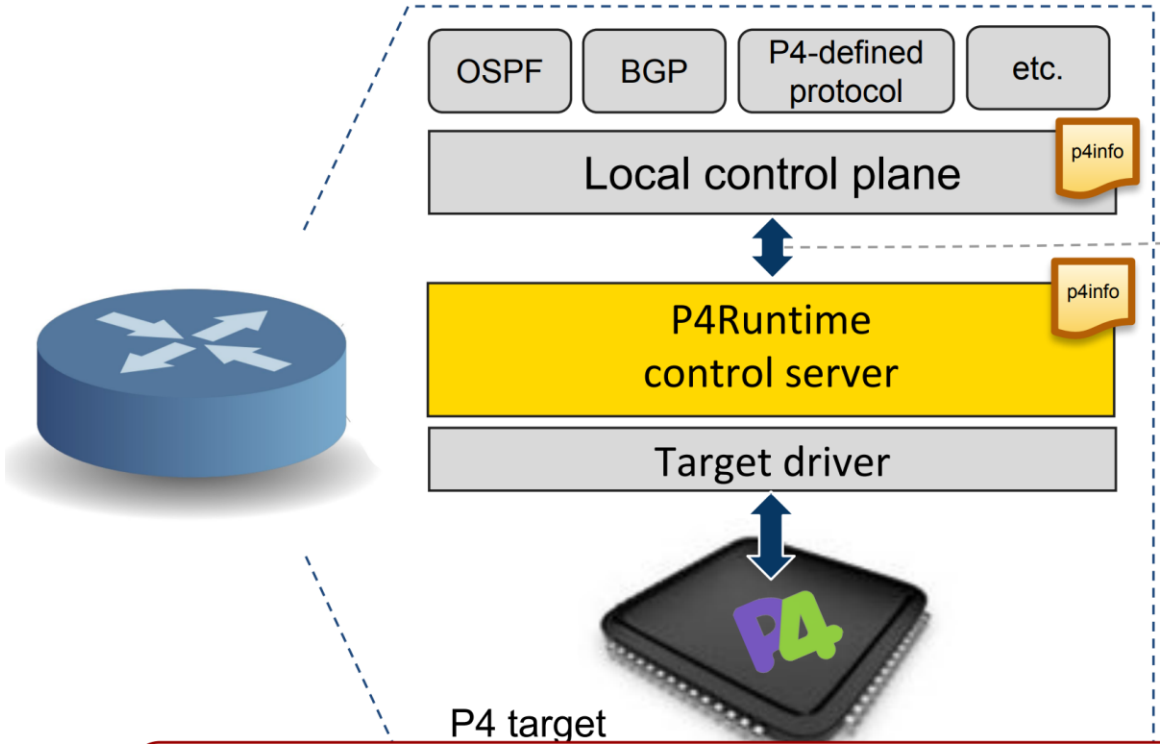
P4Runtime

- P4 compiler generates 2 outputs:
  - Target specific binary
    - Used to realize switch pipeline (i.e., binary config for specific target/switch)
  - P4Info file
    - “Schema” of pipeline for tuntime control
      - Captures P4 program attributes such as tables, actions, parameters, etc.
    - Protobuf-based format
    - Target-independente compiler output
      - Same P4Info for different targets

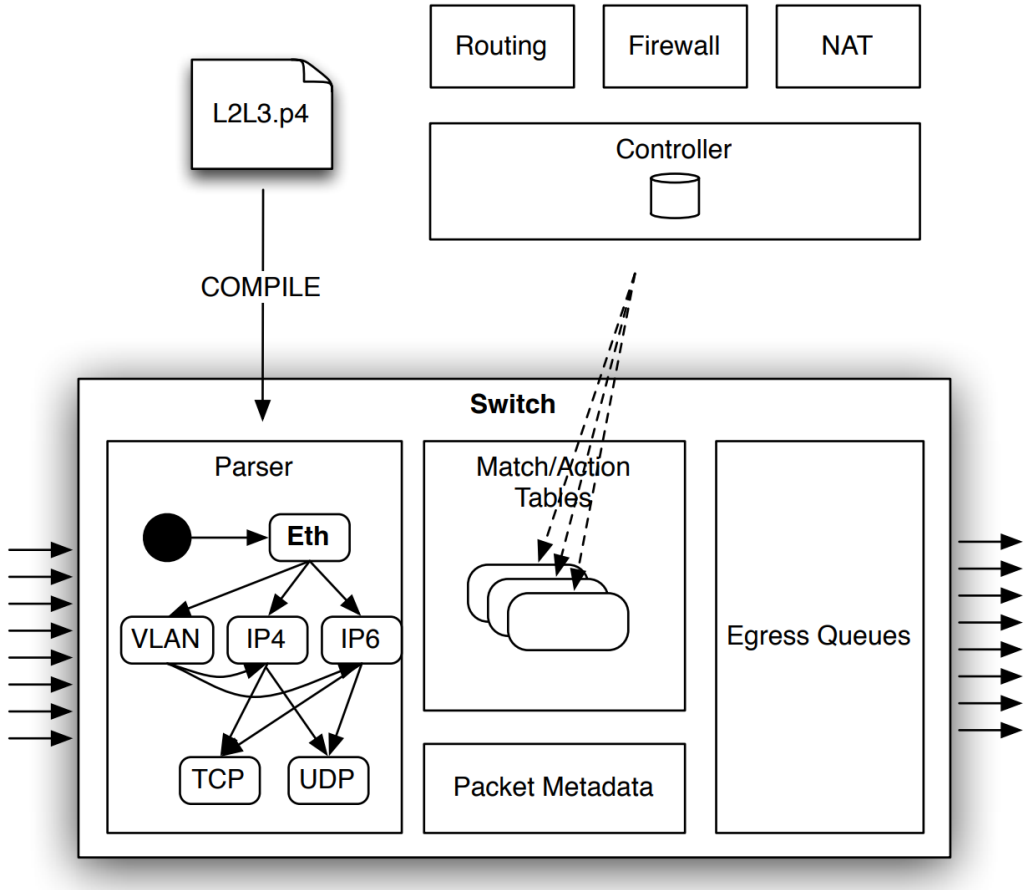


P4Runtime

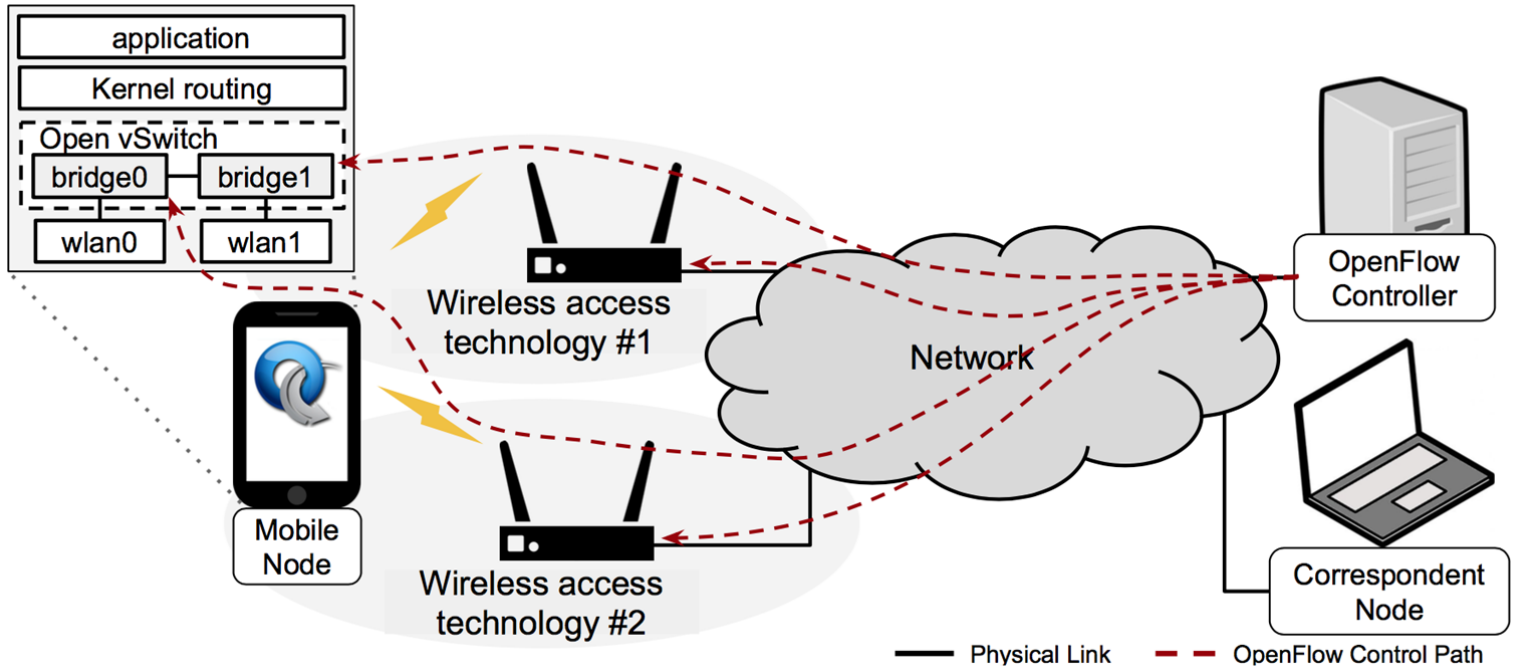
- The p4info file is used by a remote or local control plane
- This capability takes advantage of the fact that the p4info generates the same target-independent protobuf format
  - API between cliente and server



P4 Forwarding Model / Runtime

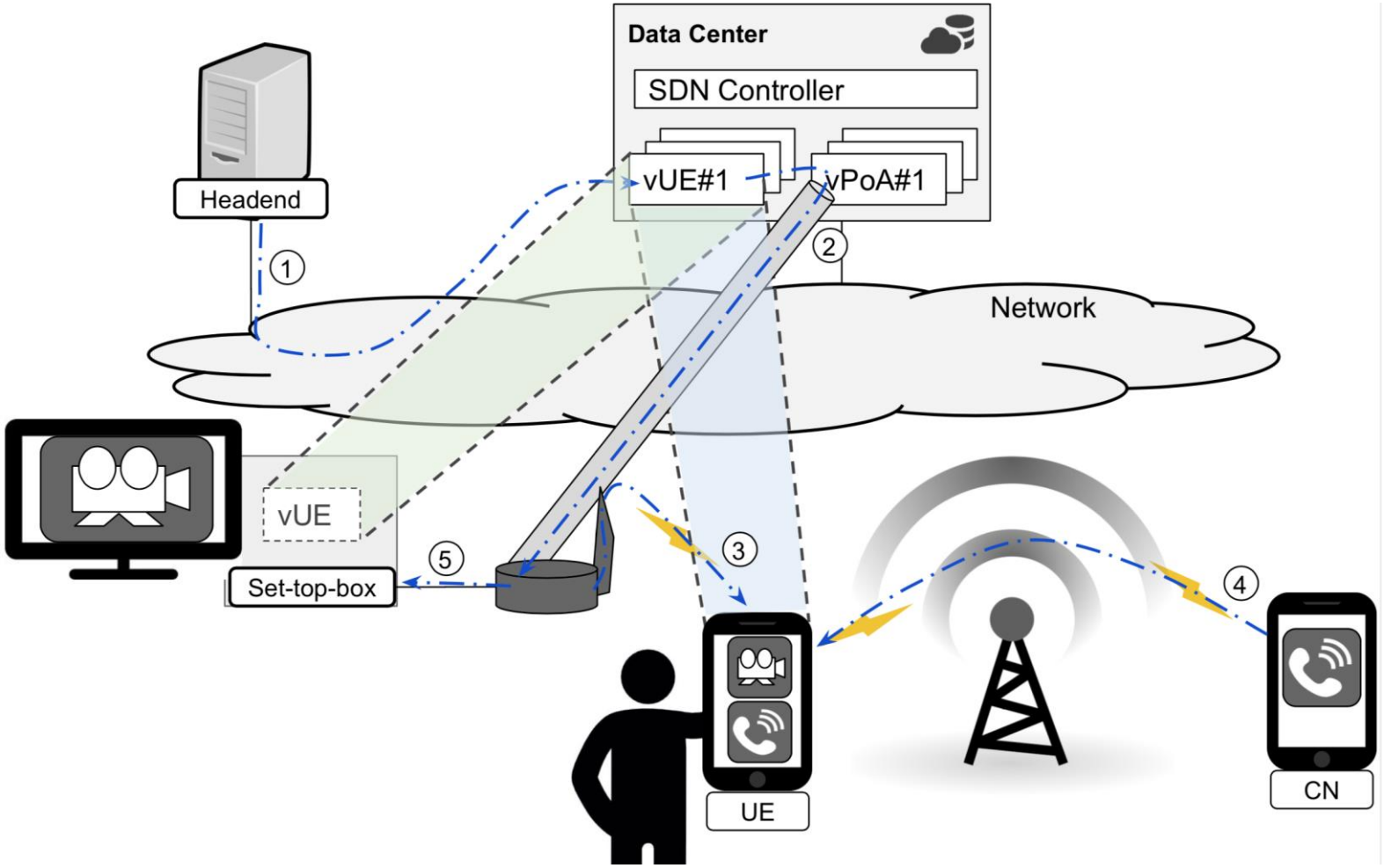


SDN+NFV - SDN in Wireless Environments (all the way to the terminal



Source: F. Meneses, C. Guimarães, D. Corujo, R. Aguiar “SDN-based Mobility Management: Handover Performance Impact in Constrained Devices”, 9th IFIP NTMS, Paris, February 2018

Connectionless environments

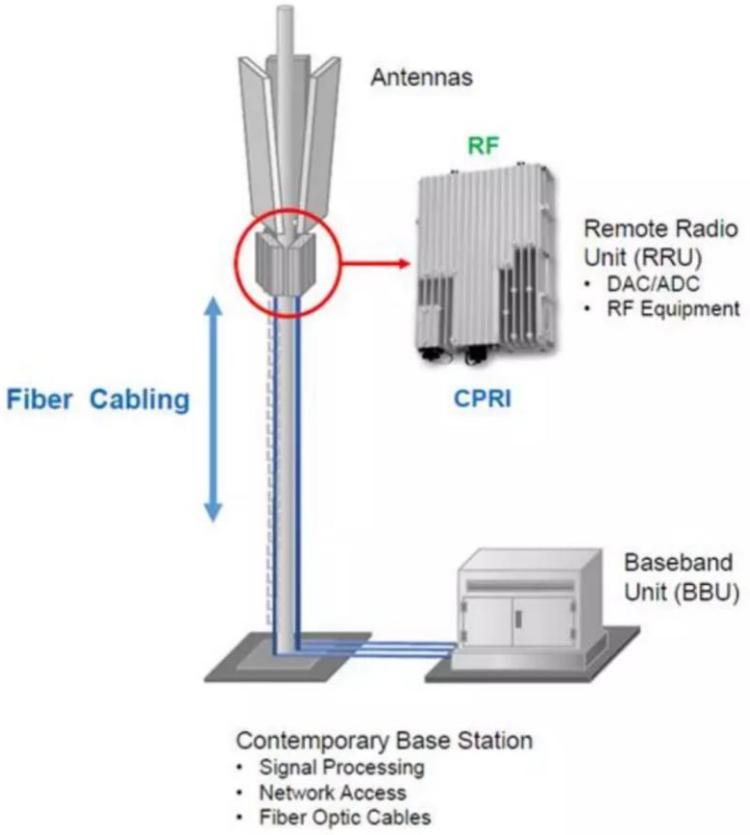


OpenRAN

Open RAN vs OpenRAN

- Open RAN (or O-RAN)
  - From the O-RAN Alliance
    - Aims to define open interfaces of split RAN architecture
    - Open and Virtualized RAN (vRAN)
- OpenRAN
  - From Telecom Infra Project
    - Aims to disintegrate mobile network RAN architecture
    - By having multi-vendor interoperable solutions
    - Built on general purpose COTS
  - Aims to speed up adoption and deployments
  - Uses O-RAN’s interfaces
  - Works with 3GPP, ONF, …

Contemporary RAN



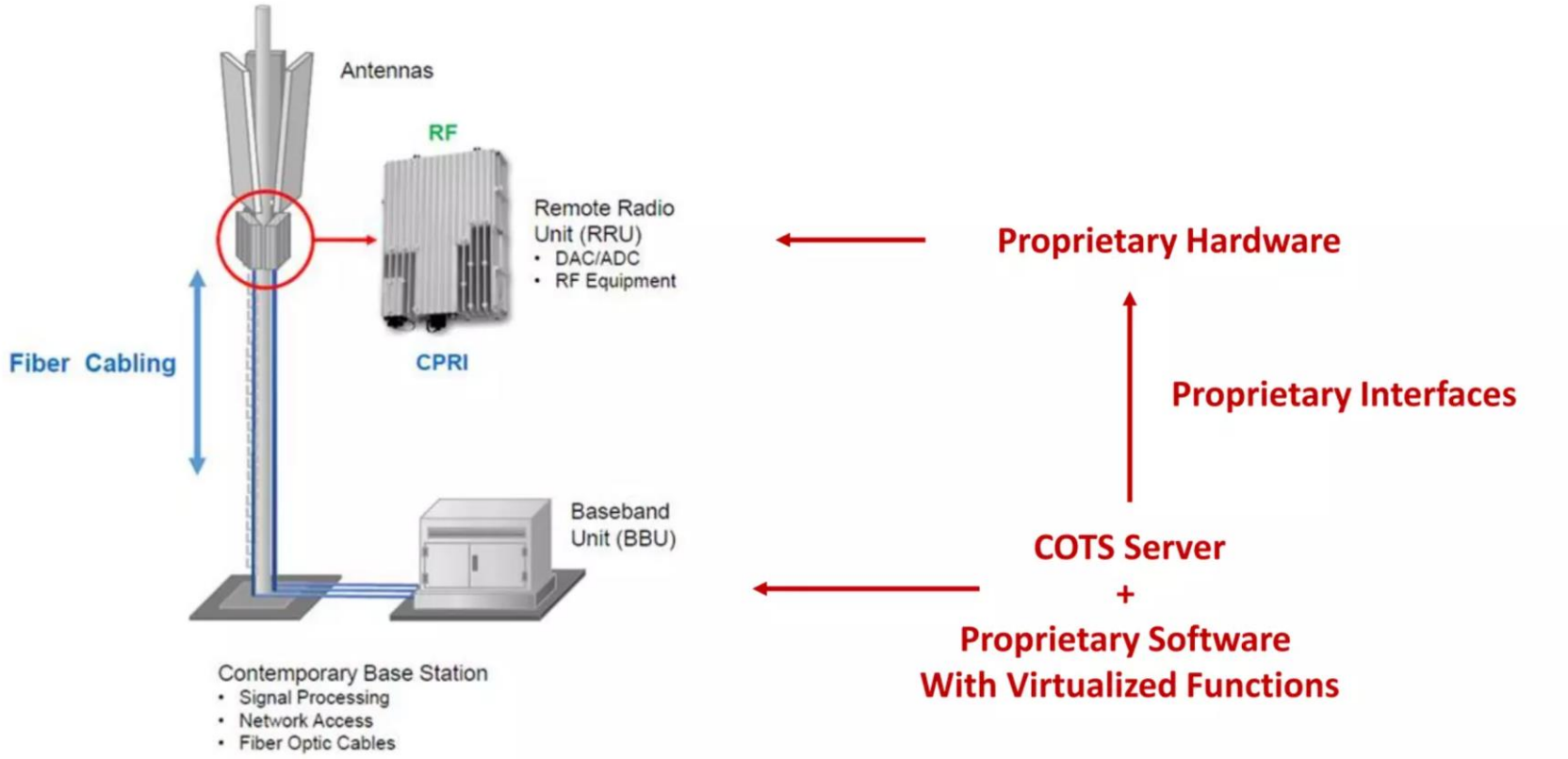
Proprietary Hardware

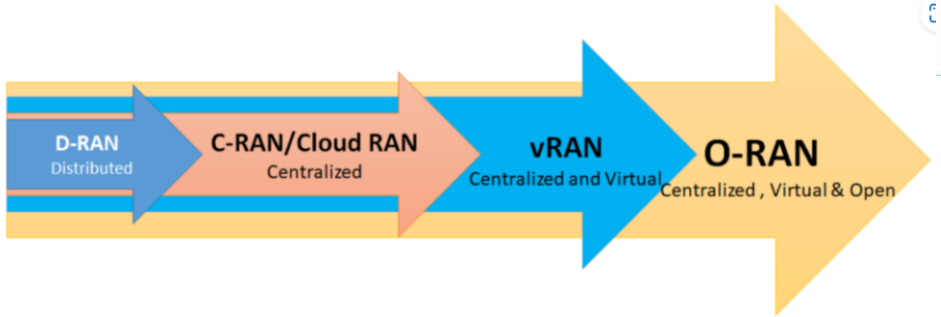
Proprietary Interfaces (not defined by 3GPP)

Proprietary Hardware + Proprietary Software

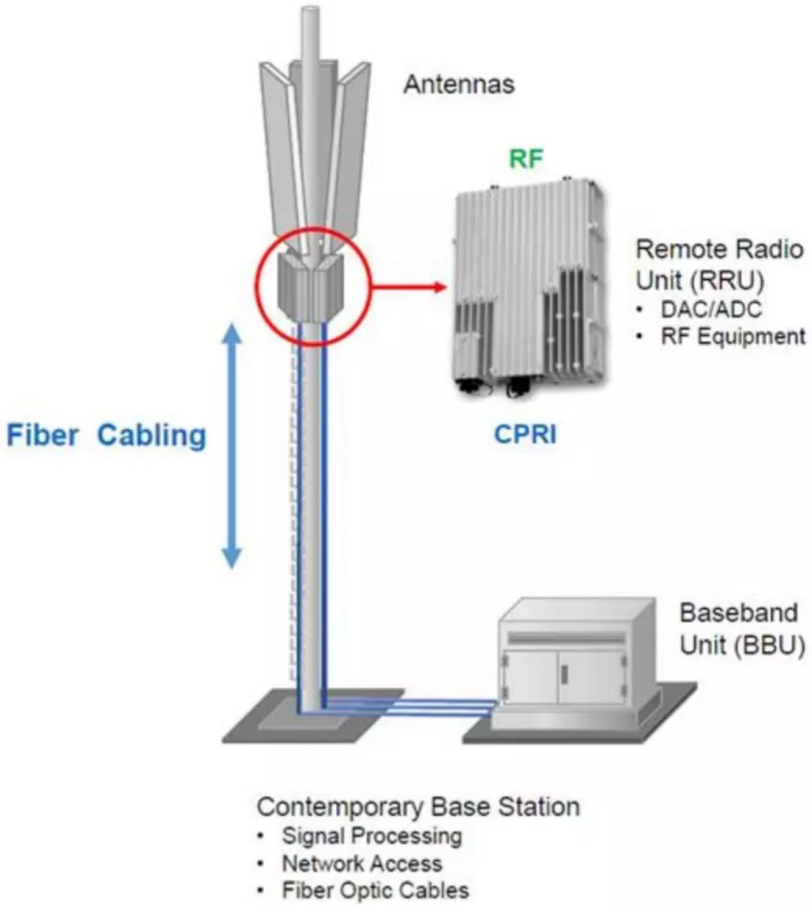
Operators need to buy both RRU and BBU from the same vendor!
5G NSA X2 interface (BBU<->BBU) w/ 4G Core: 4G and 5G same vendor!

Virtualized RAN (vRAN) approach





OpenRAN vision



GPP based COTS Hardware (SDR)
Can be purchased from any ODM / OEM / RAN Hardware Vendor

Open Interface
Any vendor software can work on this hardware

COTS Server
+
Proprietary Software With Virtualized Functions

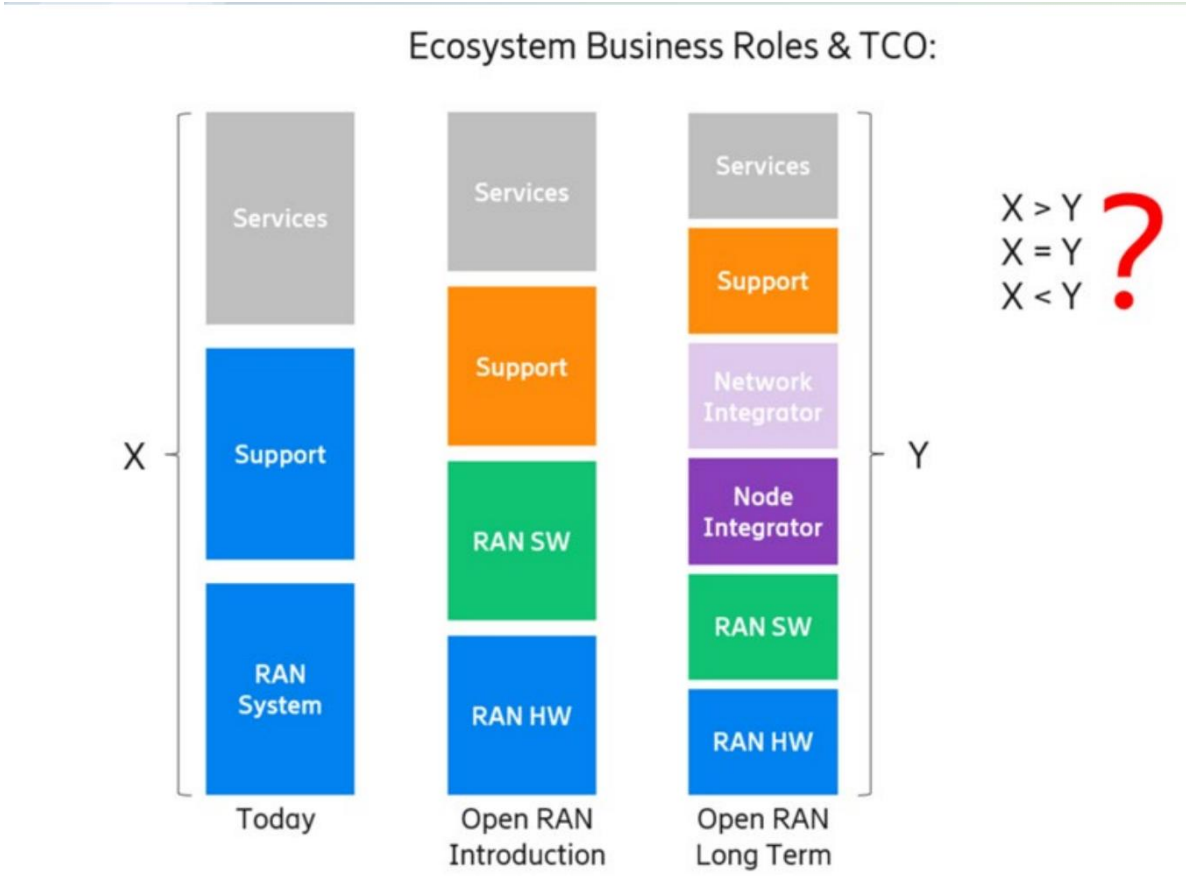
OpenRAN benefits

- Avoid Vendor lock-in
- Reduce cost
- Quick time to market
- Best of breed
- Spur Innovation
- Participate in HW/SW development

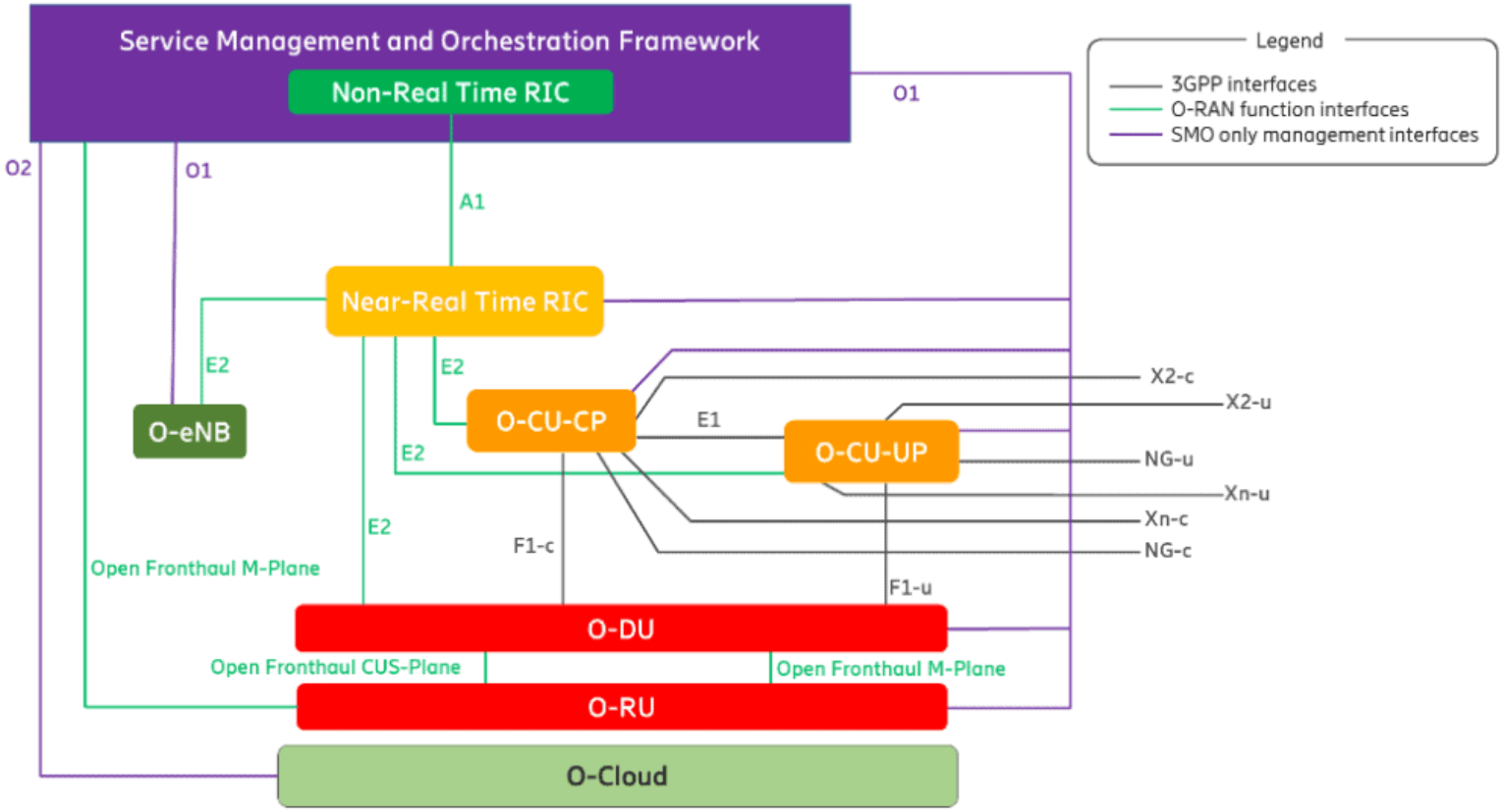
OpenRAN Challenges

- New HW and SW requirements
- Interoperability
- Complex automation
- Virtualization security
- Reliability and Availability
- High fronthaul bandwidth low-latency

Ecossystem and Business Roles

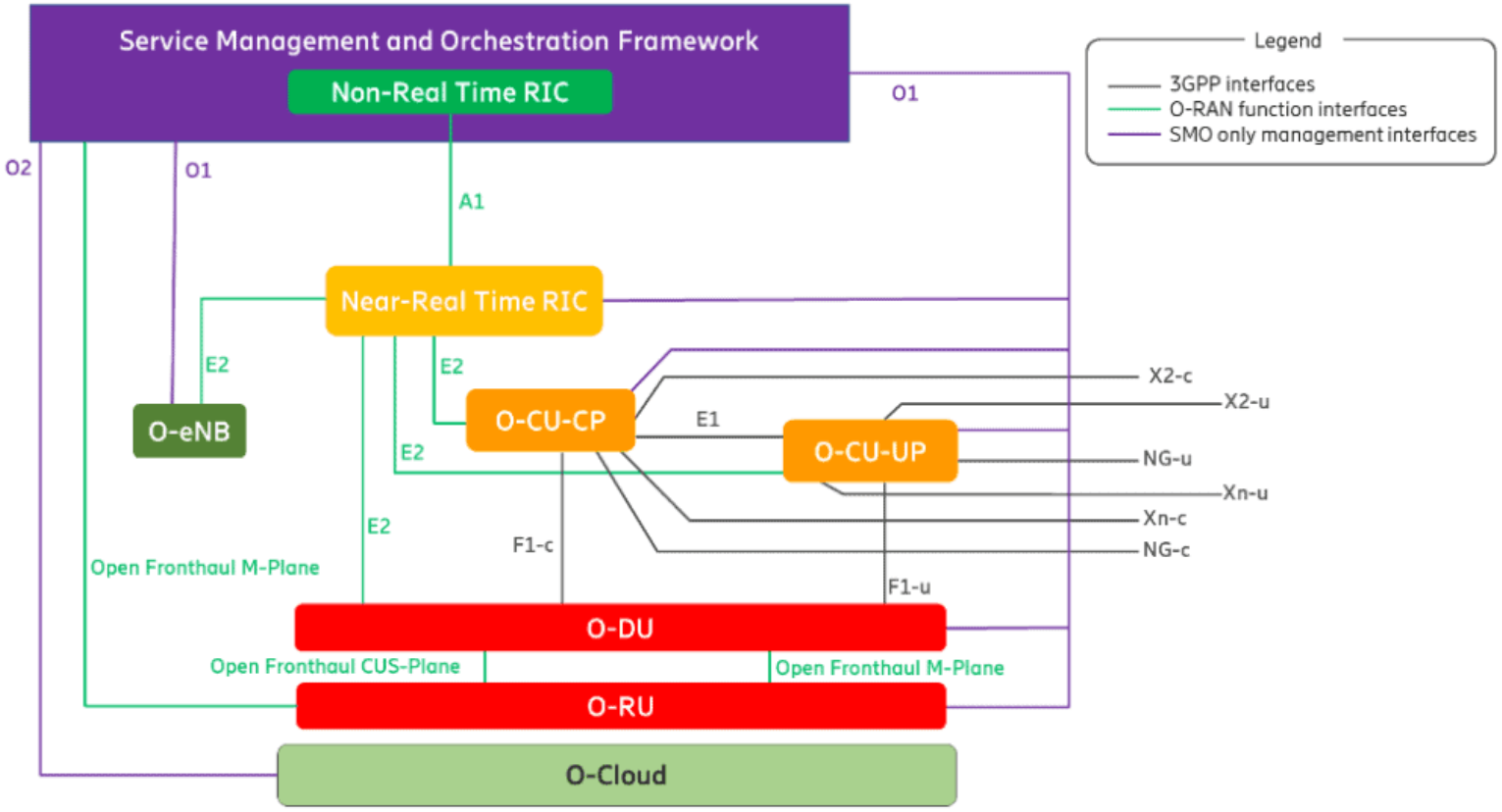


Logical Open RAN Architecture



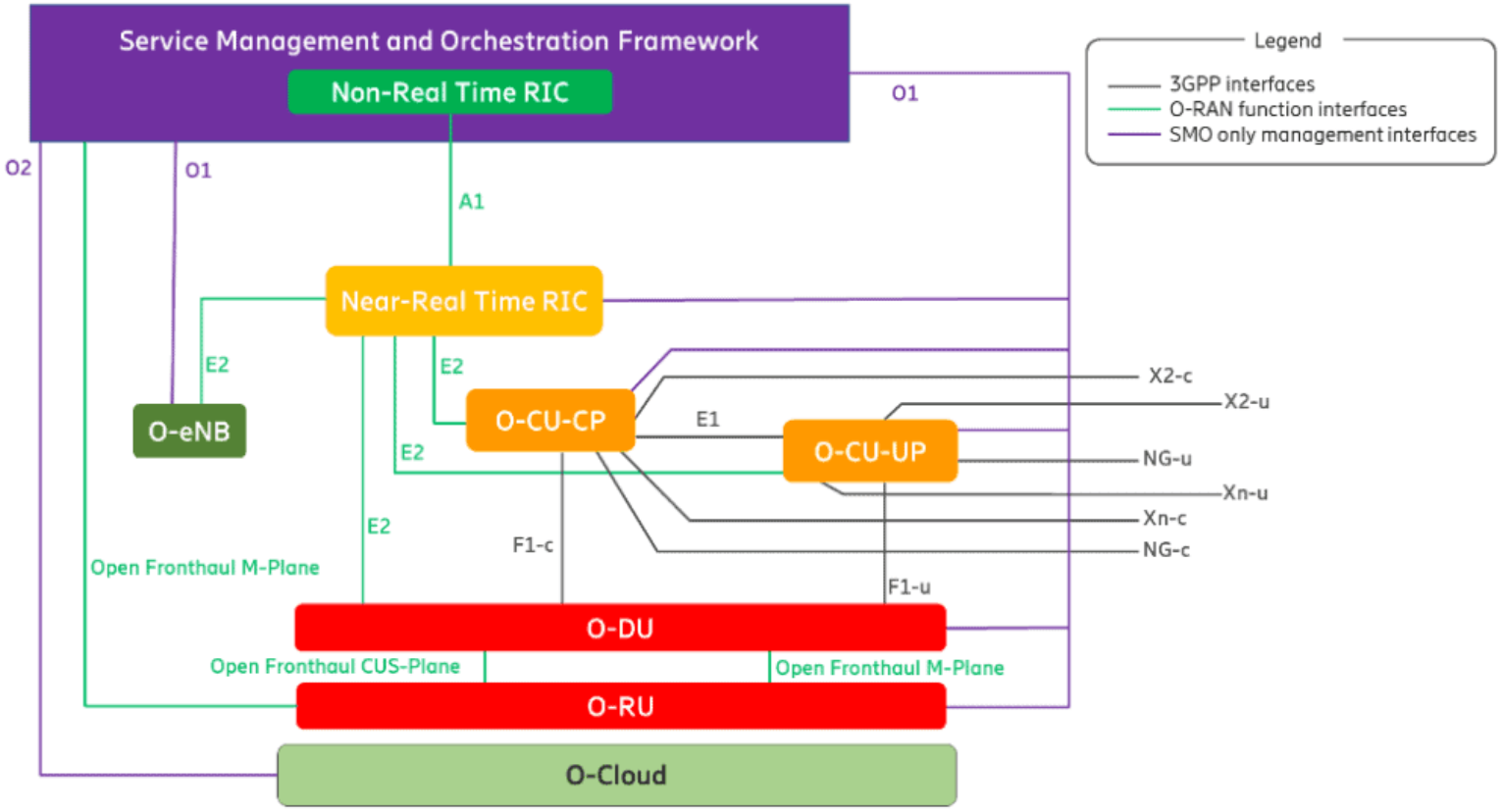
- RIC → RAN Intelligent Controller
- Non-RT RIC
  - FCAPS of O-RAN network functions
    - Fault, configuration, accounting, performance and security
  - RAN optimization in Non-RT (>1 second)

Logical Open RAN Architecture

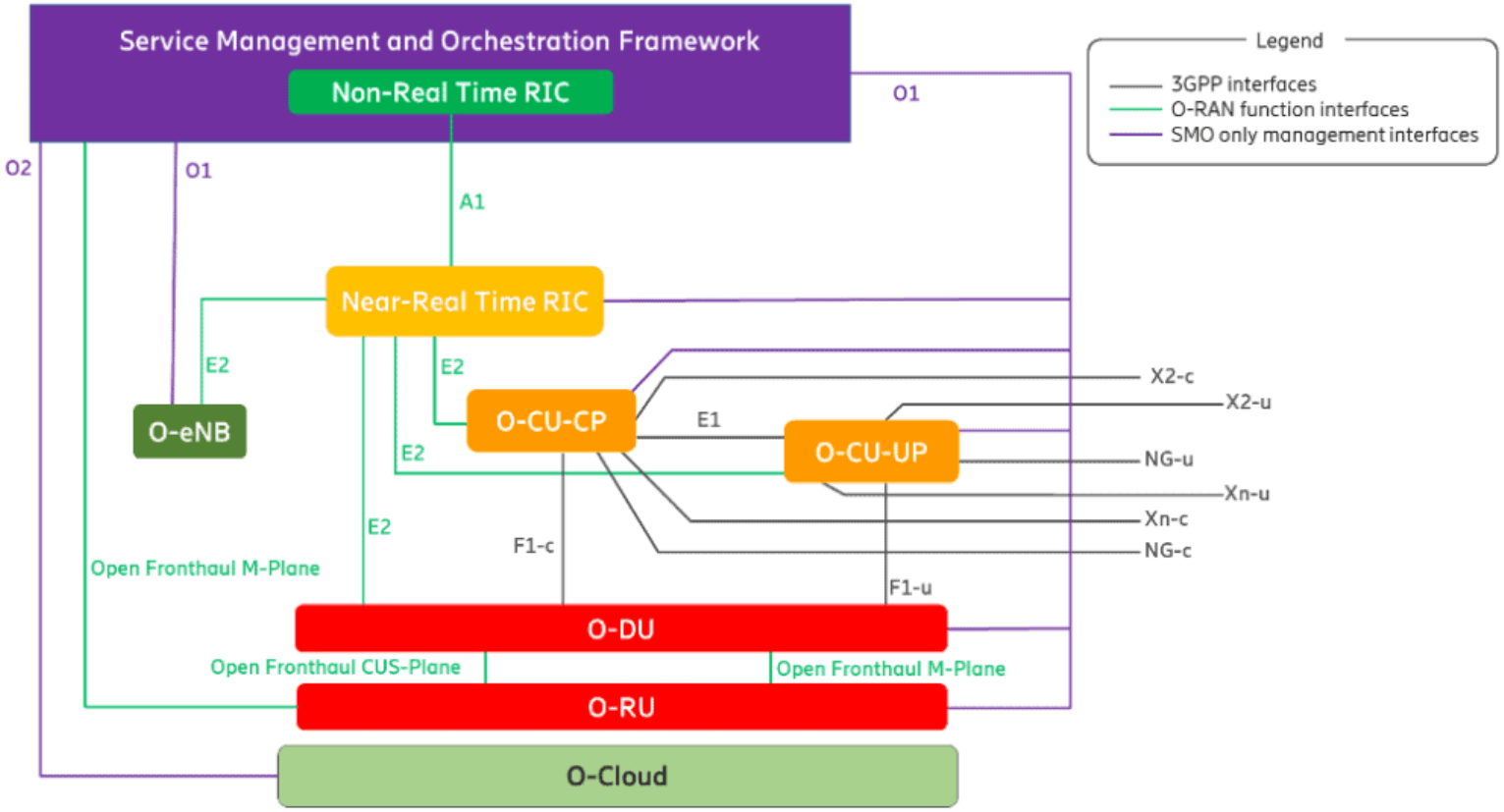


- Near-RT RIC
  - Enables near-time control and optimization of O-RAN nodes (10ms to 1 sec)
    - O-CU
      - Contralized Unit
    - O-DU
      - Distributted Unit
  - Here, decision latency is very importante, thus has to be closer to the ran nodes

Logical Open RAN Architecture

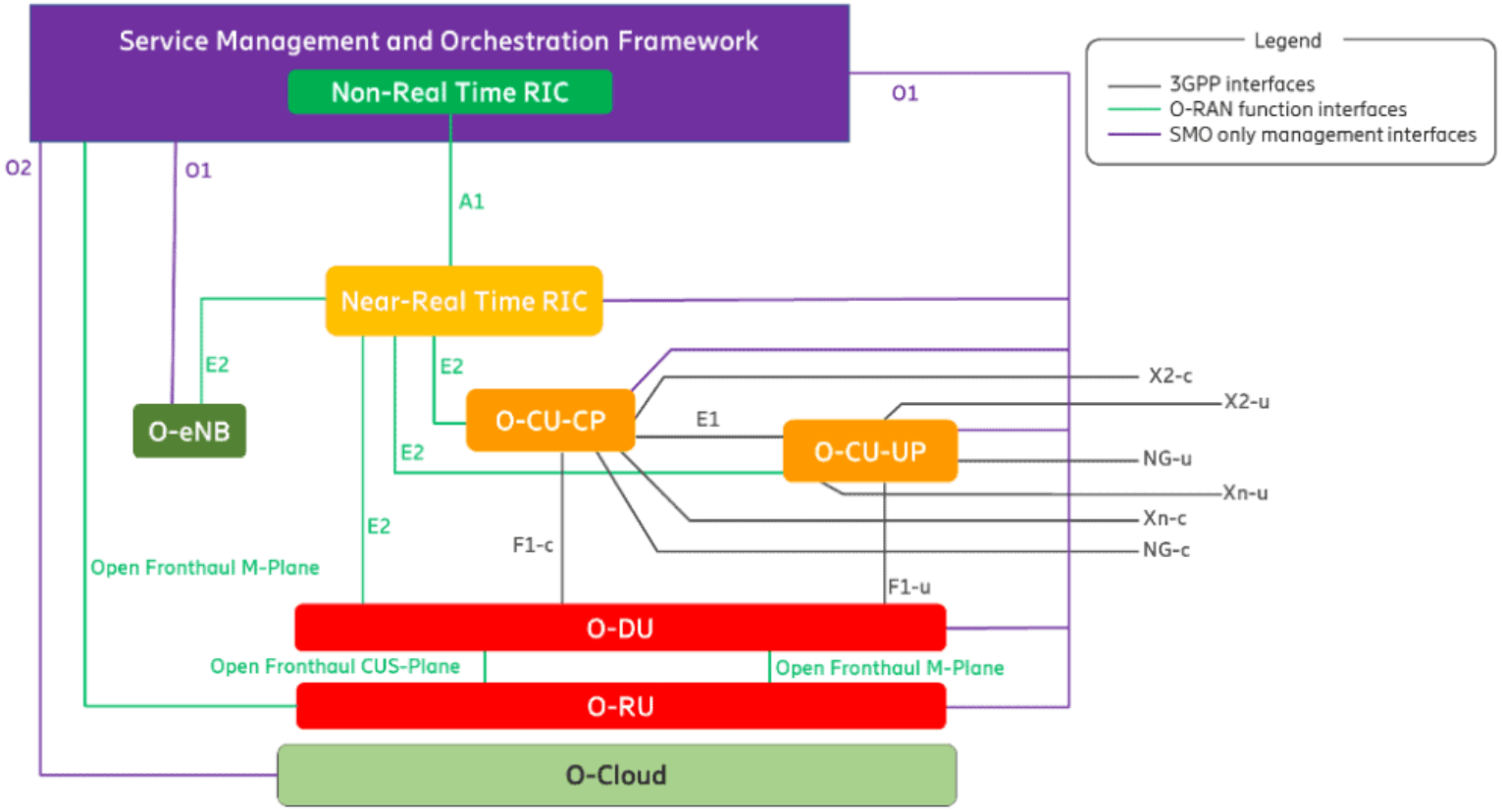


Logical Open RAN Architecture



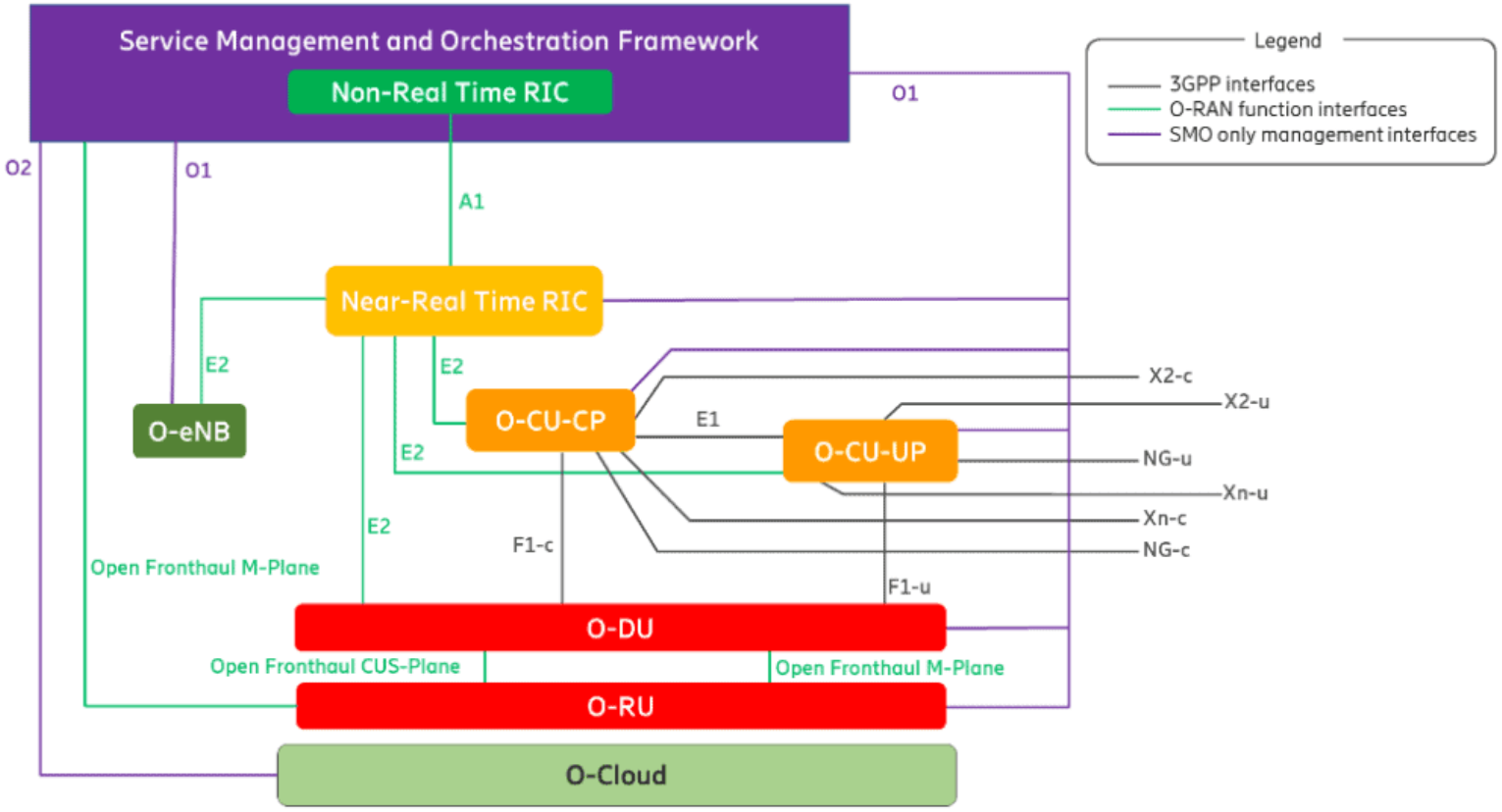
- O-DU
  - Open Distributed Unit
  - Terminates the fronthaul interface
  - Logical node hosting High-PHY layers based on a lower layer functional splut

Logical Open RAN Architecture



- O-CU-CP and O-CU-UP
  - O-RAN Central Unit
  - Control Plane (CP)
  - User Plane (UP)
  - CP
    - Hosts the RRC (Radio Resource Control) and the control plane part of the PDCP (Packet Data Convergence) protocol
  - UP hosts the user plane part of the PDCP protocol and the SDAP (Service Data Adaptation) protocol

Logical Open RAN Architecture



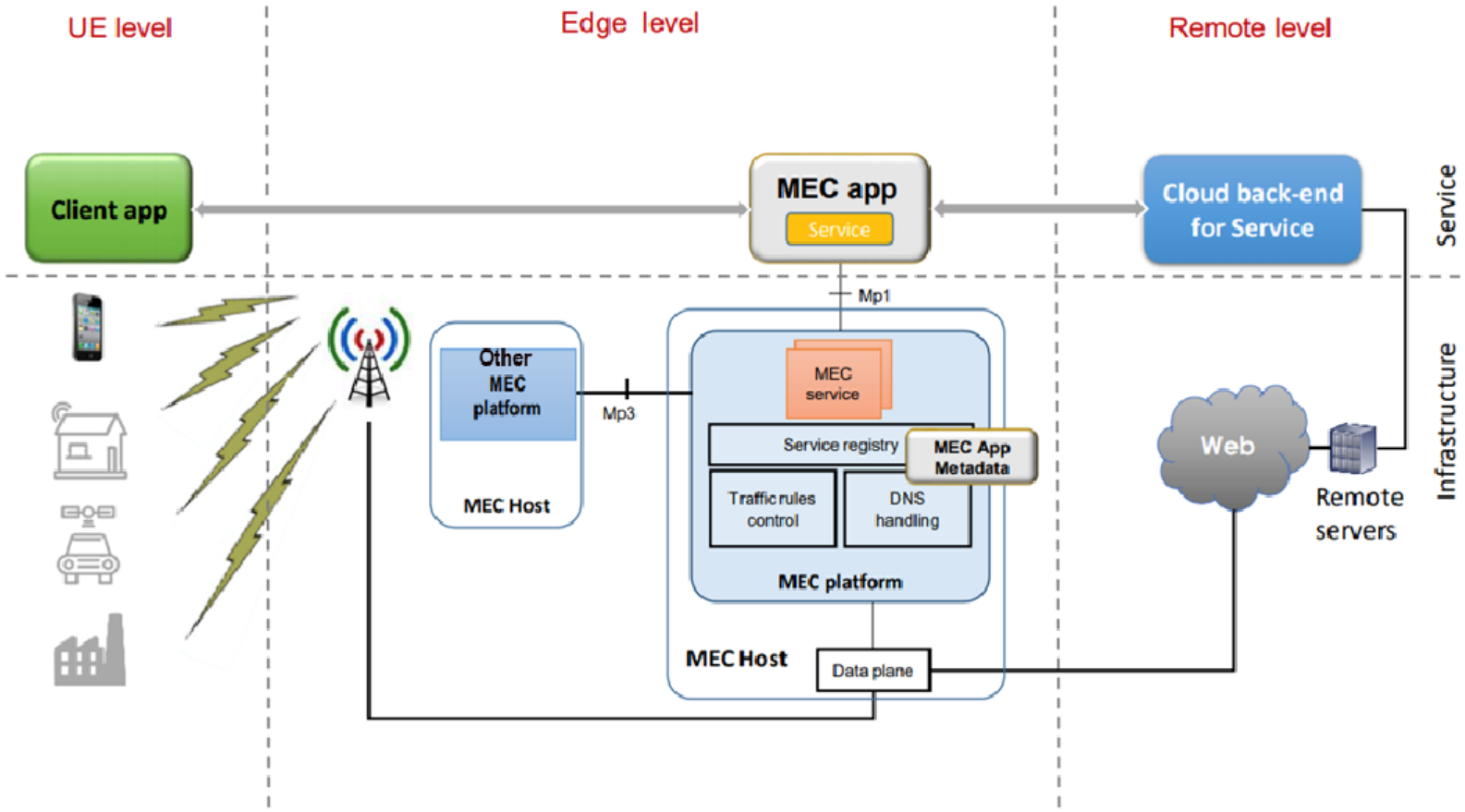
- O-RU
  - O-RAN Radio Unit
  - Hosts Low-PHY layers and RF processing
- O-Cloud
  - The generic cloud infrastructure

Multi-access Edge Computing

MEC

Multi-Access Edge Computing

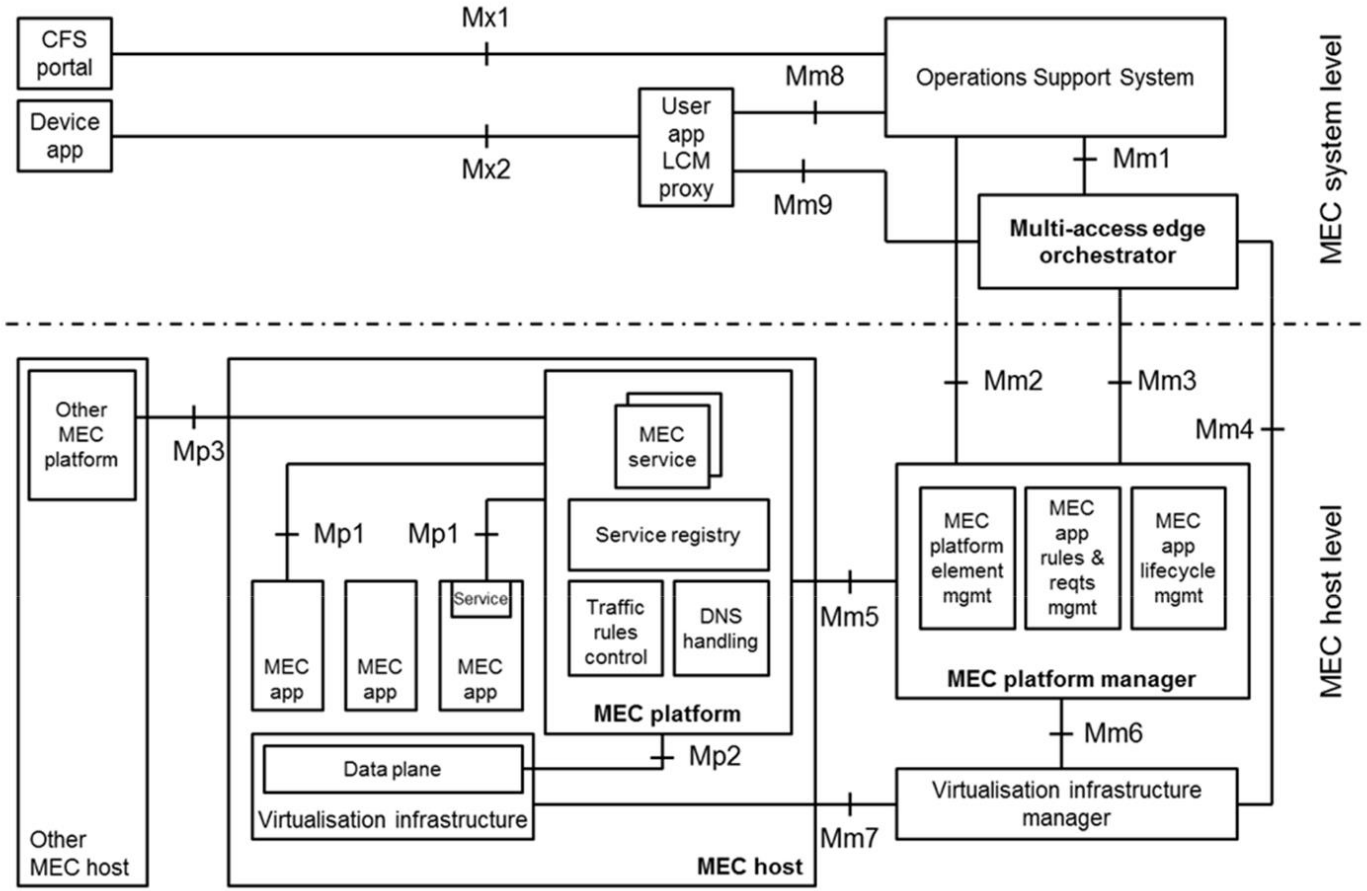
concept



Source: ETSI

Multi-Access Edge Computing

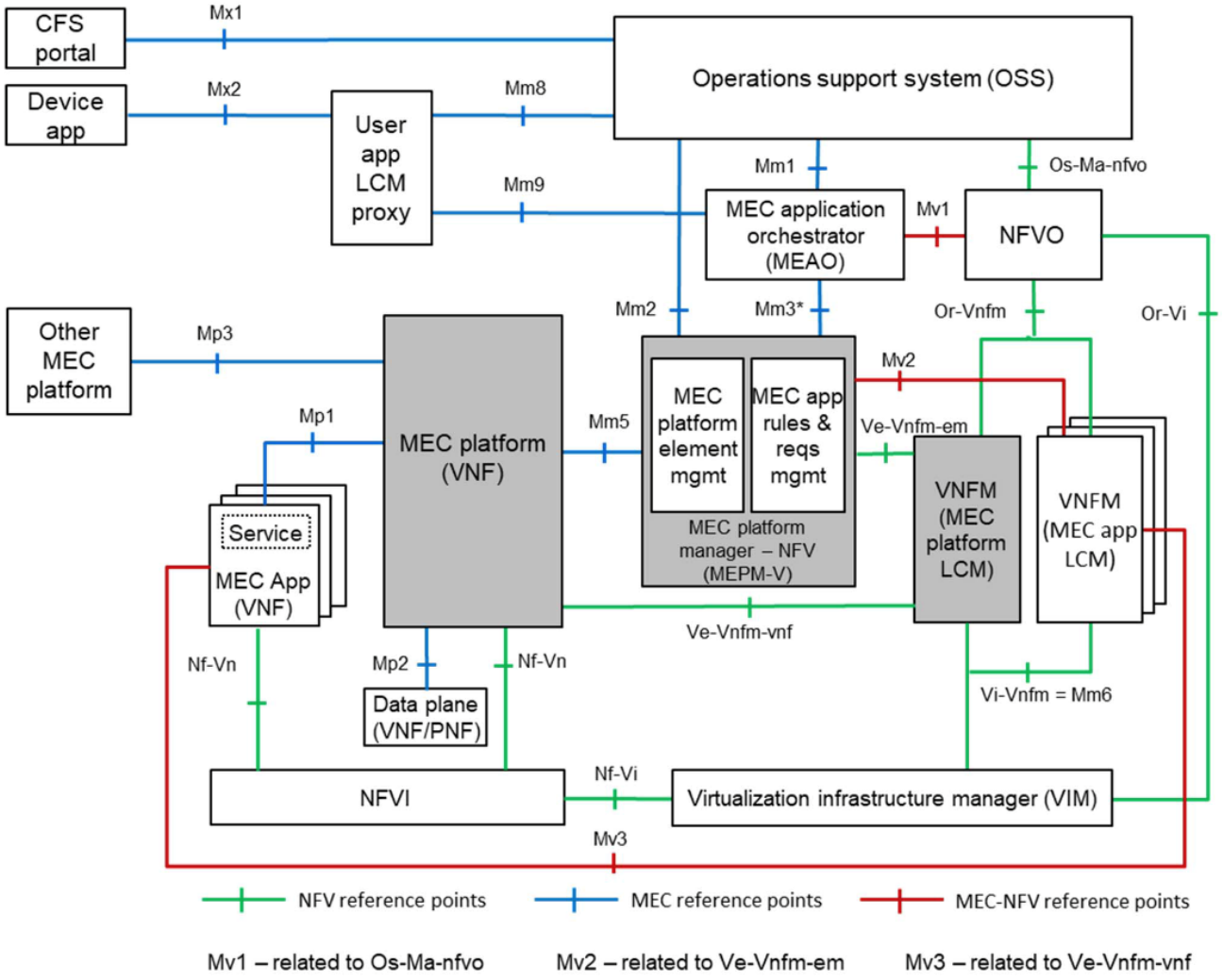
reference architecture



Source: ETSI

Multi-Access Edge Computing

mapping into NFV



Source: ETSI

Network automation

AI application to networks

- Accelerates service provisioning
  - Makes networks understand service intentions
- Reduces the probability of human errors
  - Automates complex networks
- Predicts and quickly rectifies faults
  - Enables networks to optimize themselves
- Improves efficiency of detecting threats
  - Enables networks to predict threats

A self-driving network

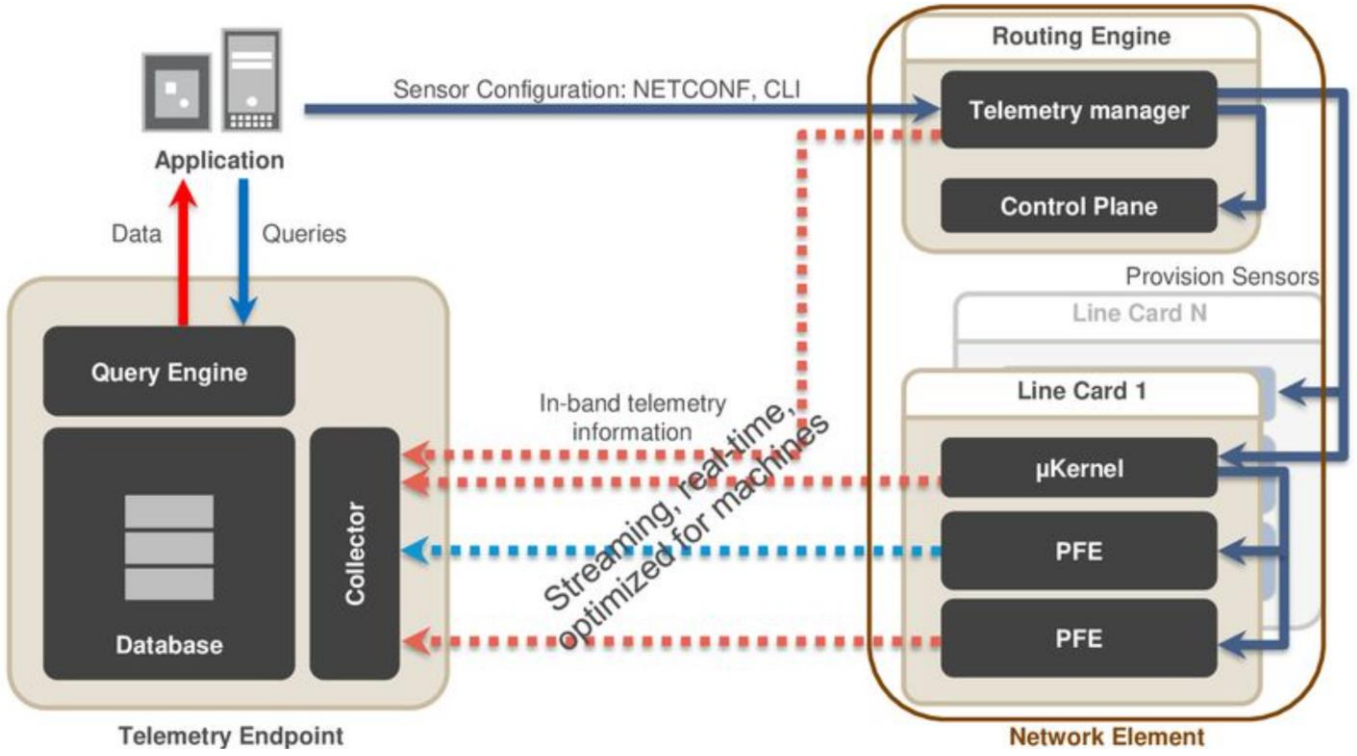
- A network that:
  - Accepts guidelines from the network operator
  - Self-discovers the componentes
  - Organizes and configures itself
  - Monitors itself using probes and other techniques
  - Auto-detects and auto-enables new customers
  - Automatically monitors and updates service delivery
  - Diagnoses itself using machine learning and heals itself
  - Periodically reports by itself

How can this become true?

- Telemetry
- Multidimensional views
- Automation
- Declarative intent
- Decision making

Telemetry

- Obtain information from the network, the data, services, etc.
- Real-time vs. Statistical
- Raw vs. pre-processed
- Key Performance Indicators



Multidimensional views

- What is useful information? To act, to account, to react…

Today

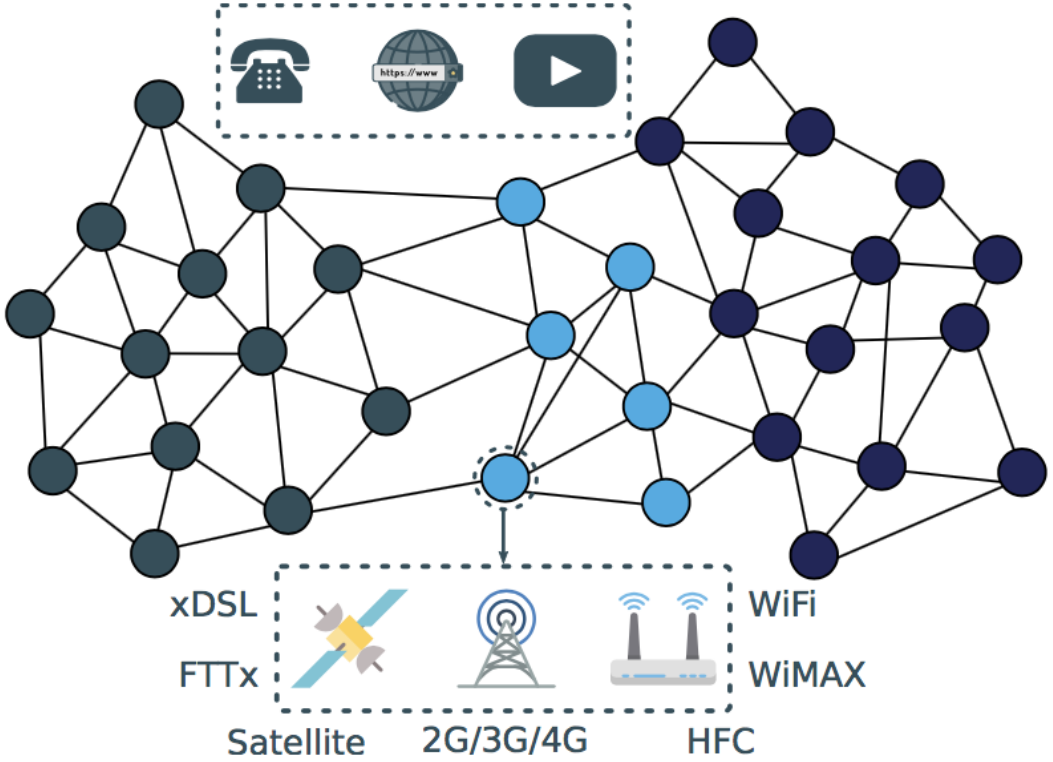
- Neighbors, links
- Exit points, peers
- Devices
- Middle boxes
- Global topology, traffic, flows
- Server and application performance
- Hackers, flash crowds, DDoS

Tomorrow

- Correlated information
  - Different geographies
  - Different layers, peers, clouds
- **Root cause analysis via supervised learning**
- **Time-based trending to establish and adapt baselines**
- **Optimal local decisions based on global state**

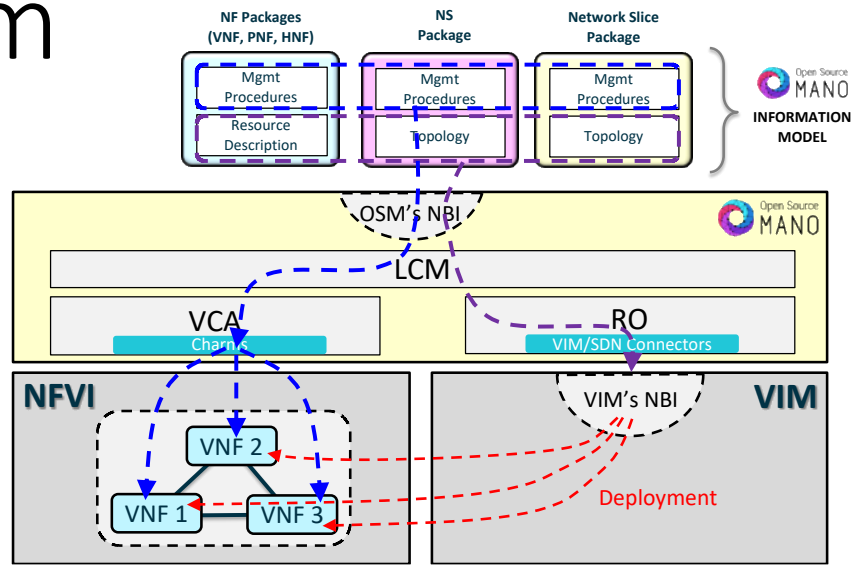
The Aggregation Scenario

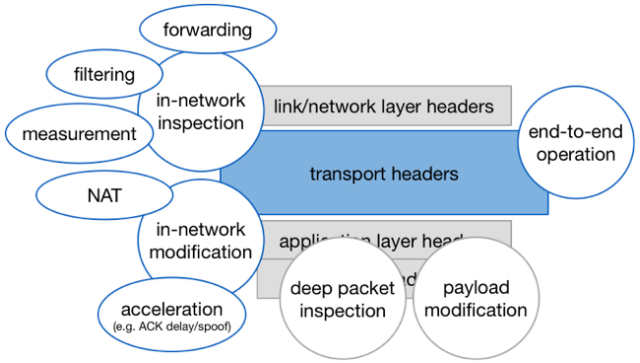
- Support the integration of different data flows
  - Open
  - Automated
  - Secure
  - Scalable
- Deal with heterogeneity at all levels
  - Data sources
  - Data consumers
  - Data models
  - Deployment styles
  - Supporting infrastructures
- Not just data
  - Metadata becomes essential, including semantic mappings
  - What seems to claim for a data stream ontology
  - Not that far away: data modeling is a first step

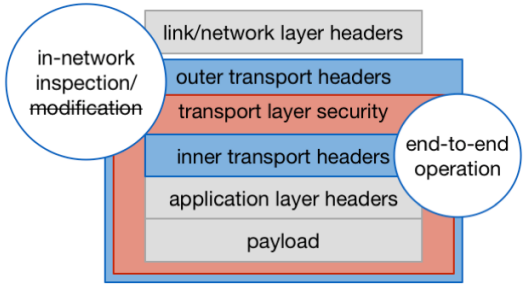


The Actuating (Control) Stream

- OAM actions at a wide variety of different domains
  - Challenging, given the current state-of-the-art
- Initial strategies
  - Domain specific
  - Recommendation systems
  - Autonomic protocols
- SBA approaches and capability models
  - Reusable functionality description
  - Abstractions of network element functionalities usable as building blocks
  - Combined to provide more powerful features
  - Registration mechanisms to support CI/CD
  - Inter-domain collaboration for E2E management

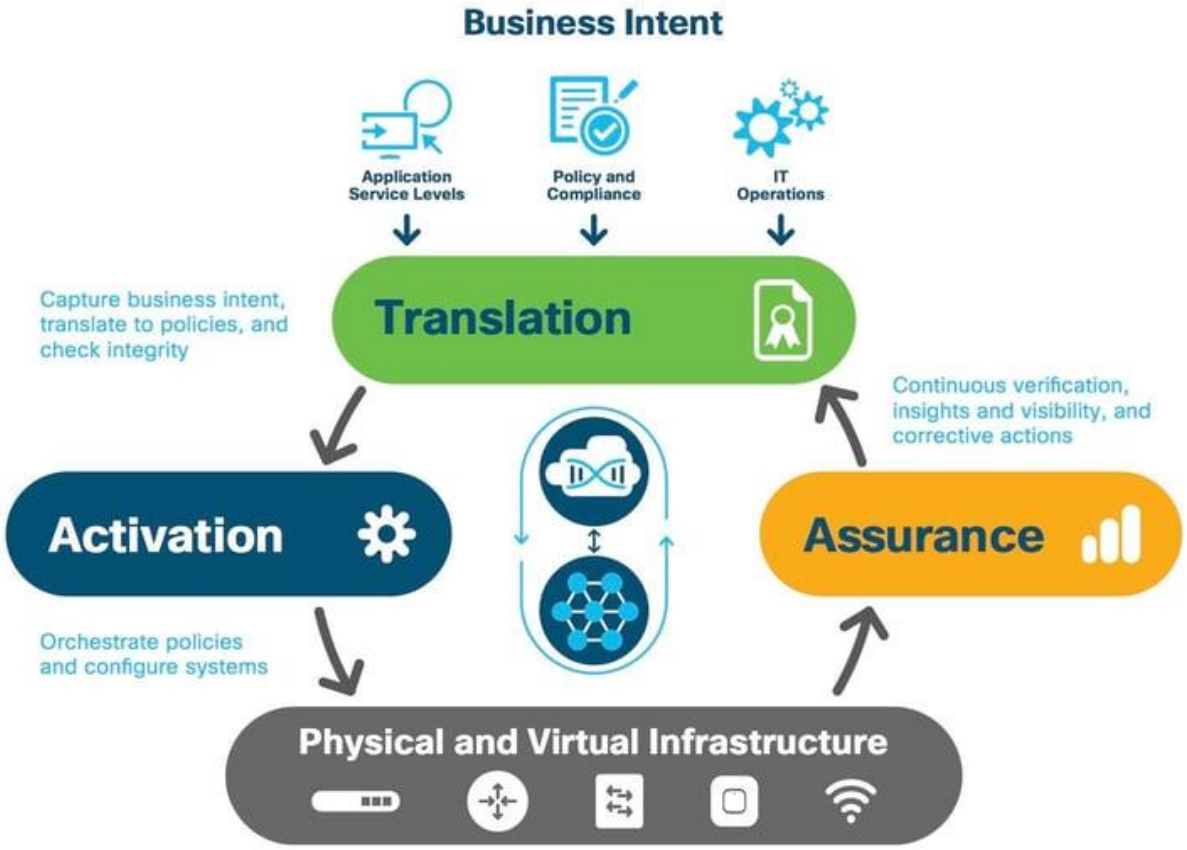






Network Automation

Declarative Intent



Decision Making

- Rule-based learning
  - If ‘X’ happens, do ‘Y’
  - Pros
    - Straightforwarding programming
    - Easy to predict and refine
  - Cons
    - Slow work
    - At scale, hard to manage

- Machine learning
  - Models, learning, percentage
  - Pros
    - Can become creative
    - Fastest way to learn complex behavior
  - Cons
    - Can come to strange conclusions
    - Hard to know what it knows

A transformation of the skillset

- Network engineering → Service design
- Network knowhow → Algorithm development
- The networks get out of the way
  - SLAs are automatically met
- Networks adapt, react, anticipate
- Security becomes Good Guy ‘Bot vs Bad Guy ‘Bot

Network automation

- Remove human from the loop of usual network management tasks
  - Provisioning
  - Detection
  - Diagnosis
  - Remediation
- In reality, is not to fully remove the humans, but remove them from the low-level painful tasks
- Why?
  - Humans are expensive
  - Reliable?
  - How to scale?

Closing the Closed Loops

- The use of closed loops is common everywhere
  - Automatics have been around for a long time
  - An essential aspect of industrial processes
- Not only about offering network data
  - An integral monitoring data substrate
- Well-defined data flow semantics
  - Data models for sources and consumers
  - Registry, discovery and dynamic orchestration
  - Full data sovereignty
- Going beyond
  - Key-Value Indicators distillation
  - Network-hosted AI and learning mechanisms
  - Support for serverless in-network computing