

# SDN, NFV, and MEC

# Why SDN and NFV for 5G

SDN and NFV are widely considered to be essential technologies for 5G

We have already mentioned the use of slicing and MEC in xHaul  
slicing is usually considered to require SDN  
and MEC can be considered to be a kind of NFV

**Slicing** needs to be end-to-end and include air interface, xhaul and core

In order to *dynamically* set up slices  
we need a new kind of network control (SDN!)

**Mobile Edge Computing** involves placing compute power in new locations

This computation may involve

- network functionality (NFV!)
- end-user applications

# Why SDN and NFV ?

Before explaining **what** SDN and NFV are  
we need to explain **why** SDN and NFV are

Its all started with two related trends ...

1. The blurring of the distinction  
between **computation** and **communications**  
revealing a fundamental disconnect  
between **software** and **networking**
2. The decrease in profitability  
of **traditional communications service providers**  
along with the increase in profitability  
of **Cloud and Over The Top service providers**

The 1<sup>st</sup> led directly to SDN  
and the 2<sup>nd</sup> to NFV  
but today both are intertwined

# 1. *Computation and communications*

Once there was little overlap  
between *communications* (telephone, radio, TV)  
and *computation* (computers)

Actually communications devices always ran complex algorithms  
but these are hidden from the user

But this dichotomy has become blurred

Most home computers are not used for *computation* at all  
rather for entertainment and communications (email, chat, VoIP)

Cellular telephones have become computers

The differentiation can still be seen in the terms *algorithm* and *protocol*  
Protocol design is fundamentally harder  
since there are two interacting entities (the *interoperability* problem)

SDN academics claim that packet forwarding is a computation problem  
and protocols as we know them should be avoided

# 1. Rich communications services

Traditional communications services are pure *connectivity* services  
transport data from A to B

with constraints (e.g., minimum bandwidth, maximal delay)

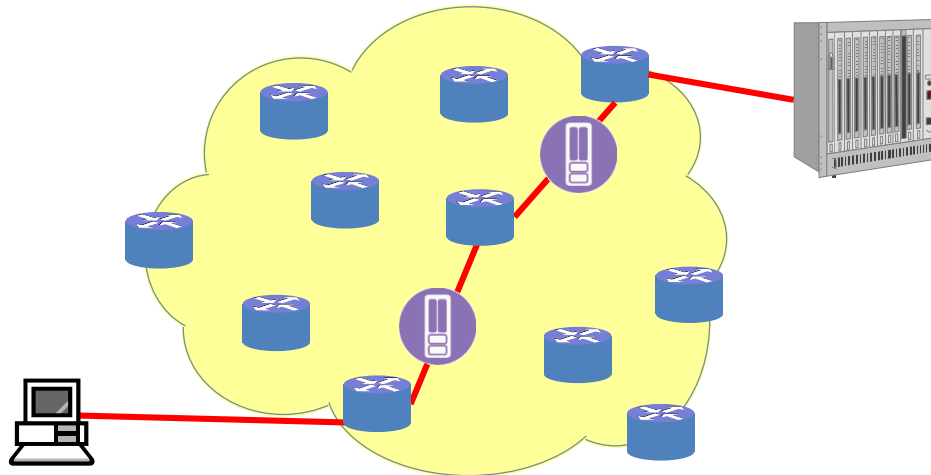
with maximal efficiency (minimum cost, maximized revenue)

Modern communications services are richer

combining connectivity and network functionalities

e.g., firewall, NAT, load balancing, CDN, parental control, ...

Such services further blur the computation/communications distinction  
and make service deployment optimization more challenging



# 1. *Software and networking speed*

Today, developing a new *iOS/Android* app takes hours to days  
but developing a new communications service takes months to years

Even adding new instances of well-known services  
is a time consuming process for conventional networks

When a new service types requires new protocols, the timeline is

- protocol standardization (often in more than one SDO)
- hardware development
- interop testing
- vendor marketing campaigns and operator acquisition cycles
- staff training
- deployment

how long has it been since the first IPv6 RFC ?

**This leads to a *fundamental disconnect*  
between software and networking development timescales**

An important goal of SDN and NFV is  
to create new network functionalities at the *speed of software*

## 2. Today's communications world

Today's infrastructures are composed of many different Network Elements (NEs)

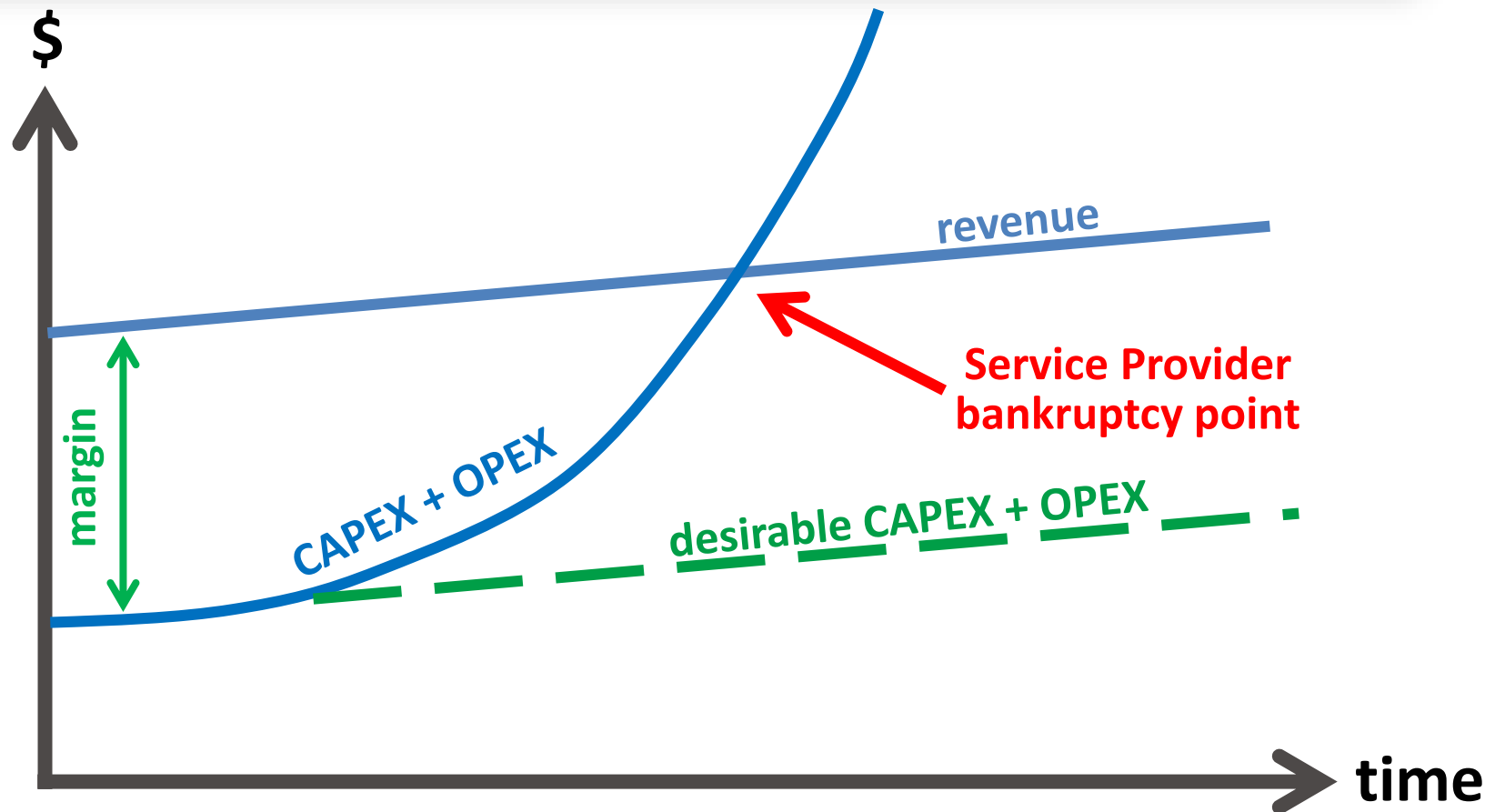
- sensors, smartphones, notebooks, laptops, desk computers, servers,
- DSL modems, Fiber transceivers,
- SONET/SDH ADMs, OTN switches, ROADMs,
- Ethernet switches, IP routers, MPLS LSRs, BRAS, SGSN/GGSN,
- NATs, Firewalls, IDS, CDN, WAN acceleration, DPI,
- VoIP gateways, IP-PBXes, video streamers,
- performance monitoring probes , performance enhancement middleboxes,
- etc., etc., etc.

New and ever more complex NEs are being invented all the time,  
and while equipment vendors like it that way  
Service Providers find it hard to shelve and power them all !

In addition, while service innovation is accelerating  
the increasing sophistication of new services  
the requirement for backward compatibility  
and the increasing number of different SDOs, consortia, and industry groups  
which means that

it has become very hard to experiment with new networking ideas  
NEs are taking longer to standardize, design, acquire, and learn how to operate  
NEs are becoming more complex and expensive to maintain

## 2. The service provider crisis



This is a *qualitative* picture of the service provider's world  
Revenue is at best increasing with number of users  
Expenses are proportional to bandwidth – doubling every 9 months  
This situation obviously can not continue forever !



# Two complementary solutions

## Software Defined Networks (SDN)

*SDN* advocates replacing standardized networking protocols with centralized software applications that configure all the NEs in the network

Advantages:

- easy to experiment with new ideas
- control software development is much faster than protocol standardization
- centralized control enables stronger optimization
- functionality may be speedily deployed, relocated, and upgraded

## Network Functions Virtualization (NFV)

*NFV* advocates replacing hardware network elements with software running on COTS computers that may be housed in POPs and/or datacenters

Advantages:

- COTS server price and availability scales with end-user equipment
- functionality can be located where-ever most effective or inexpensive
- functionalities may be speedily combined, deployed, relocated, and upgraded

SDN

# Abstractions

SDN was triggered by the development of networking technologies not keeping up with the speed of software application development

Computer science theorists theorized that this derived from not having the required **abstractions**

In CS an *abstraction* is a representation that reveals semantics needed *at a given level* while hiding implementation details thus allowing a programmer to focus on necessary concepts without getting bogged down in unnecessary details

Programming is fast because programmers exploit abstractions

Example:

It is very slow to code directly in assembly language (with few abstractions, e.g. opcode mnemonics)  
It is a bit faster to coding in a low-level language like C (additional abstractions : variables, structures)  
It is much faster coding in high-level imperative language like Python  
It is much faster yet coding in a declarative language (coding has been abstracted away)  
It is fastest coding in a domain-specific language (only contains the needed abstractions)

**In contrast, in protocol design we return to *bit level* descriptions every time**

# Packet forwarding abstraction

The first abstraction relates to how network elements forward packets

At a high enough level of abstraction

all network elements perform the same task

## **Abstraction 1** *Packet forwarding as a computational problem*

The function of any network element (NE) is to

- receive a packet
- observe packet fields
- apply algorithms (classification, decision logic)
- optionally edit the packet
- forward or discard the packet

For example

- An Ethernet switch observes MAC DA and VLAN tags, performs exact match, forwards the packet
- A router observes IP DA, performs LPM, updates TTL, forwards packet
- A firewall observes multiple fields, performs regular expression match, optionally discards packet

We can replace all of these NEs with a configurable ***whitebox switch***

# Network state and graph algorithms

How does a whitebox switch learn its required functionality ?

Forwarding decisions are optimal

when they are based on full global knowledge of the network

With full knowledge of topology and constraints

the path computation problem can be solved by a graph algorithm

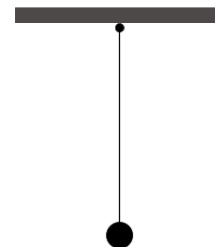
While it may sometimes be possible to perform path computation (e.g., Dijkstra) in a distributed manner

It makes more sense to perform them centrally

## **Abstraction 2** *Routing as a computational problem*

Replace distributed routing protocols with graph algorithms performed at a central location

Note with SDN, the pendulum that swung from the completely centralized PSTN to the completely distributed Internet swings back to completely centralized control



# Configuring the whitebox switch

How does a whitebox switch acquire the information needed to forward that has been computed by an omniscient entity at a central location ?

## Abstraction 3 *Configuration*

Whitebox switches are directly *configured* by an *SDN controller*

Conventional network elements have two parts:

1. smart but slow CPUs that create a Forwarding Information Base
2. fast but dumb switch fabrics that use the FIB

Whitebox switches only need the dumb part, thus

- eliminating distributed protocols
- not requiring intelligence

The API from the SDN controller down to the whitebox switches is conventionally called the *southbound API* (e.g., OpenFlow, ForCES)

Note that this SB API is in fact a *protocol* but is a simple configuration protocol not a distributed routing protocol

# Separation of data and control

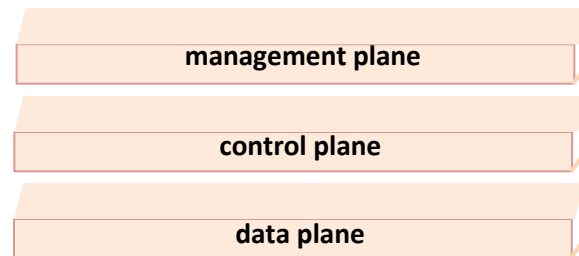
You will often hear stated that the *defining attribute* of SDN is the separation of the *data* and *control* planes

This separation was not invented recently by SDN academics  
Since the 1980s all well-designed communications systems have enforced logical separation of 3 planes :

- data plane (forwarding)
- control plane (e.g., routing )
- management plane (e.g., policy, commissioning, billing)

What SDN really does is to

- 1) insist on ***physical*** separation of data and control
- 2) erase the difference between control and management planes



# Flows

It would be too slow for a whitebox switch to query the centralized SDN controller for every packet received

So we identify packets as belonging to **flows**

## **Abstraction 4** *Flows* (as in OpenFlow)

Packets are handled solely based on the flow to which they belong

Flows are thus just like **Forwarding Equivalence Classes**

Thus a flow may be determined by

- an IP prefix in an IP network
- a label in an MPLS network
- VLANs in VLAN cross-connect networks

The granularity of a flow depends on the application



# Control plane abstraction

In the standard SDN architecture, the SDN controller is omniscient but does not itself *program* the network since that would limit development of new network functionalities

With software we create building blocks with defined APIs which are then used, and perhaps inherited and extended, by programmers

With networking, each network *application* has a tailored-made control plane with its own element discovery, state distribution, failure recovery, etc.

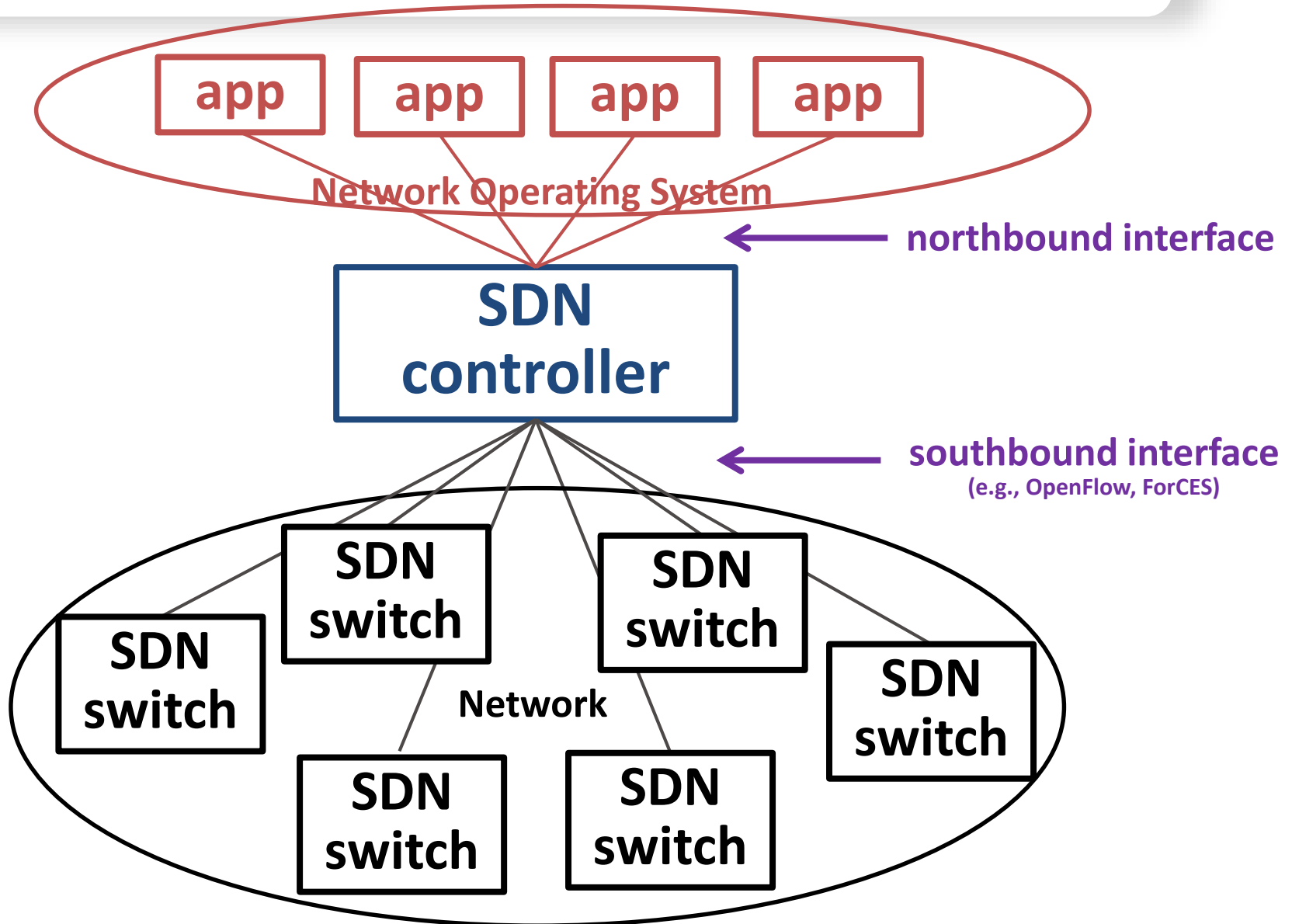
Note the subtle change of terminology we have just introduced instead of calling switching, routing, load balancing, etc. network *functions* we call them network *applications* (similar to software *apps*)

## **Abstraction 5** Northbound APIs instead of protocols

Replace control plane protocols with well-defined APIs to network applications

This abstraction hide details of the network from the network application revealing high-level concepts, such as requesting connectivity between A and B but hiding details unimportant to the application such as details of switches through which the path  $A \rightarrow B$  passes

# SDN overall architecture



NFV

# Virtualization of computation

In the field of computation, there has been a major trend towards **virtualization**

*Virtualization* here means the creation of a **virtual machine** (VM) that acts like an independent physical computer

A **VM** is software that emulates hardware (e.g., an x86 CPU) over which one can run software as if it is running on a physical computer

The VM runs on a *host* machine and creates a *guest* machine (e.g., an x86 environment)

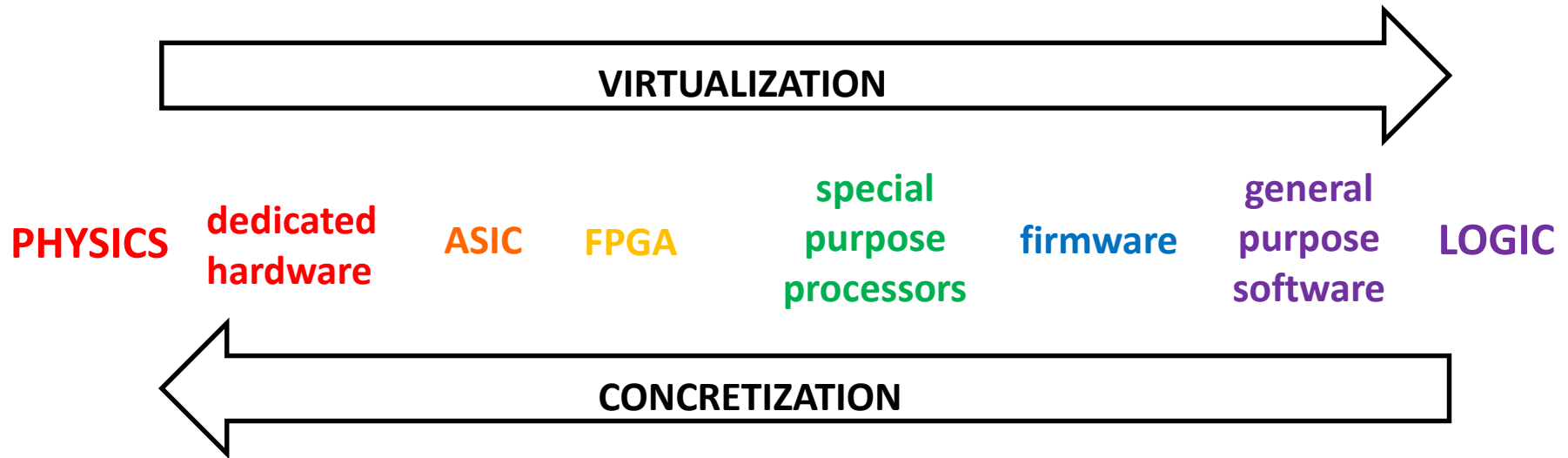
A single host computer may host many fully independent guest VMs and each VM may run different Operating Systems and/or applications

For example

- a datacenter may have many racks of server cards
- each server card may have many (host) CPUs
- each CPU may run many (guest) VMs

A **hypervisor** is software that enables *creation* and *monitoring* of VMs

# Concretization and Virtualization



*Concretization* means moving a task *to the left*

Justifications for concretization include :

- cost savings for mass produced products
- miniaturization/packaging constraints
- need for high processing rates
- energy savings / power limitation / low heat dissipation

*Virtualization* is the opposite - moving a task *to the right*

(although frequently reserved for the extreme case of HW → SW)

# Network Functions Virtualization

CPUs are not the only hardware device that can be virtualized

Many (but not all) NEs can be replaced by software running on a CPU or VM

This would enable

- using standard COTS hardware (whitebox servers)
  - reducing CAPEX and OPEX
- fully implementing functionality in software
  - reducing development and deployment cycle times, opening up the R&D market
- consolidating equipment types
  - reducing power consumption
- optionally concentrating network functions in datacenters or POPs
  - obtaining further economies of scale. Enabling rapid scale-up and scale-down

For example, switches, routers, NATs, firewalls, IDS, etc.

are all good candidates for virtualization

as long as the data rates are not too high

Physical layer functions (e.g., Software Defined Radio) are not ideal candidates

High data-rate (core) NEs will probably remain in dedicated hardware

# Function relocation

Once a network functionality has been virtualized  
it is relatively easy to relocate it

By relocation we mean

placing a function somewhere other than its conventional location  
e.g., at **P**oints of **P**resence and **D**ata **C**enters

Many (mistakenly) believe that the main reason for NFV  
is to move networking functions to data centers  
where one can benefit from economies of scale

Some telecomm functionalities need to reside at their conventional location

- Loopback testing
- E2E performance monitoring

but many don't

- routing and path computation
- billing/charging
- traffic management
- DoS attack blocking

Note: even nonvirtualized functions *can* be relocated

# Example of relocation with SDN

SDN is, in fact, a specific example of function relocation

In conventional IP networks routers perform 2 functions

- forwarding
  - observing the packet header
  - consulting the **F**orwarding **I**nformation **B**ase
  - forwarding the packet
- routing
  - communicating with neighboring routers to discover topology (routing protocols)
  - runs routing algorithms (e.g., Dijkstra)
  - populating the FIB used in packet forwarding

SDN enables moving the routing algorithms to a centralized location

- replace the router with a simpler but configurable whitebox switch
- install a centralized SDN controller
  - runs the routing algorithms (internally – w/o on-the-wire protocols)
  - configures the NEs by populating the FIB



# Micro-services

When building physical networks elements

there is pressure to put all functionality into a single box

Modern software systems are designed to be flexible

by using *micro-services* and *function chaining*

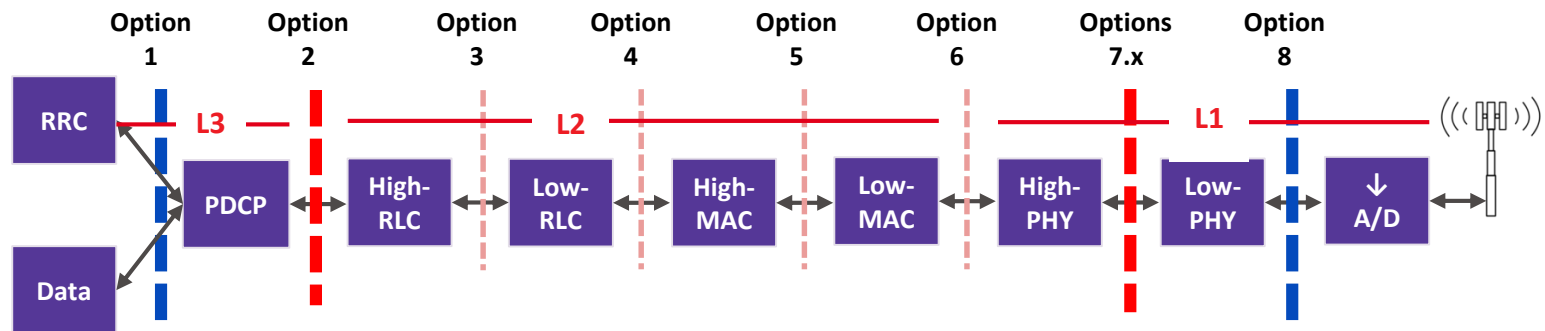
For example, many network functions utilize (deep) packet inspection

but this function is not packaged separately as a micro-service

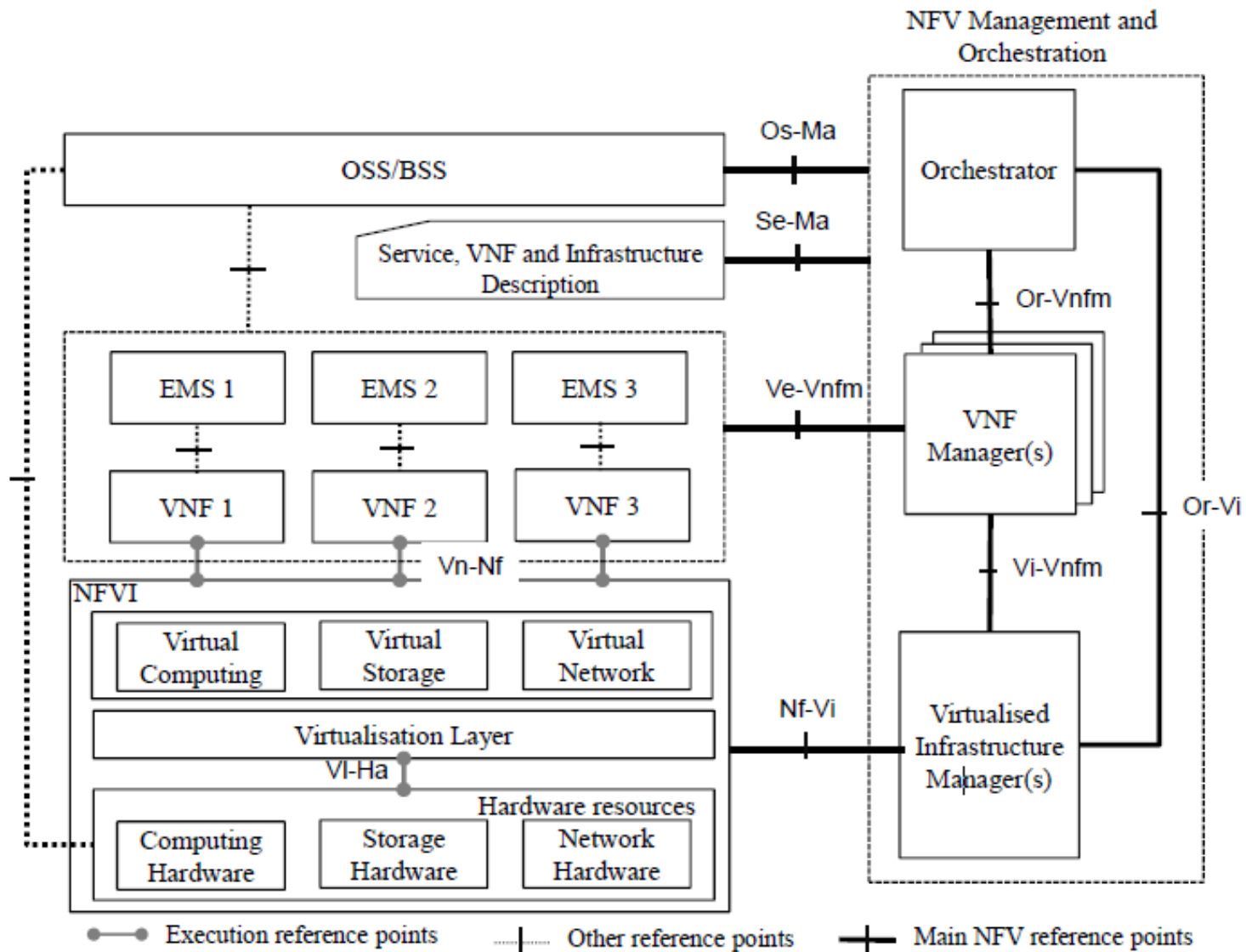
The functional decomposition of a gNB that we have seen before

can be seen to be a chain of micro-services

many of which can be virtualized



# ETSI NFV-ISG architecture



# MANO ? VIM ? VNFM? NFVO?

Traditional NEs have NMS (EMS) and perhaps are supported by an OSS

NFV has *in addition* the MANO (Management and Orchestration) containing :

- an orchestrator
- VNFM(s) (VNF Manager)
- VIM(s) (Virtual Infrastructure Manager)
- lots of reference points (*interfaces*) !

The VIM (usually OpenStack) manages NFVI resources in one NFVI domain

- life-cycle of virtual resources (e.g., set-up, maintenance, tear-down of VMs)
- inventory of VMs
- FM and PM of hardware and software resources
- exposes APIs to other managers

The VNFM manages VNFs in one VNF domain

- life-cycle of VNFs (e.g., set-up, maintenance, tear-down of VNF instances)
- inventory of VNFs
- FM and PM of VNFs

The NFVO is responsible for resource and service orchestration

- controls NFVI resources everywhere via VIMs
- creates end-to-end services via VNFMs

MEC

# Origin of MEC

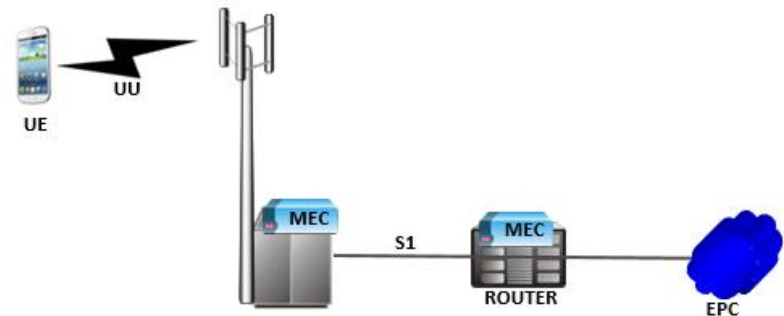
**2012** a group of service providers created ETSI NFV ISG to promote virtualization of network functions (mostly relocating them to data centers)

**2013** RAD proposed Distributed NFV (DNVF) – hosting VNFs in a CPE

**2013** NSN introduced Liquid apps – a Radio Applications Cloud Server (RACS) capable of running VNFs in base stations

**2014** a group of companies created ETSI MEC (Mobile Edge Computing) ISG

During its work, MEC was generalized the concept of edge from the base station to include a PoP in the RAN



**2016** MEC was renamed Multi-access Edge Computing to include edge computing in the wireline case

# Why do we need local processing?

Both uCPE and MEC hinge on processing that needs to be performed locally rather than relocated to a remote data center

What leads to the need for local processing ?

- Functionalities that are *required* to be local (FM, PM, encryption, etc.)
- Applications that require ultra-low delay (e.g., URLLC)
- Functions that perform best when local (e.g., interactive)
- Persistent local storage
- Access to local resources (not available to OTT services)
- Reduce requirements for high bandwidth for long distances thus reducing congestion
- Keep local data local

# MEC Use Cases

MEC ISG identified numerous applications

wherein mobile networks require local processing or storage:

- Enterprise services including VoLTE and breakout to enterprise LAN
- Live video streaming
- Identity based content delivery
- Location based content delivery (retail, consumer, ...)
- Location tracking
- RAN/application aware content optimization
- Distributed content and DNS caching
- AR (location based) / VR including real-time streaming
- Video acceleration and analytics
- IoT detection/processing/aggregation
- V2x (ultra low delay)
- Emergency response / law enforcement

# Required mobile services

MEC facilitates hosting third-party applications in mobile networks

From the use cases one can discern the requirement

for access to certain services from the mobile network, including:

- traffic steering (based on application, user, location, etc.)  
both between MEC applications and to/from network
- local persistent storage
- traffic rule enforcement
- local DNS proxy/server
- UE identification (e.g., the IMSI)
- Radio Network Information Services (cell identifier, handoff occurred, etc.)
- Location (geolocation coordinates)
- Traffic prioritization and bandwidth policy enforcement
- Lawful interception and metadata retention

Much of the MEC ISG's work focused on defining APIs  
for MEC applications to access these services



# MEC platform

As part of the *MEC host* server hosting the *MEC applications*  
MEC defines a *MEC platform* supporting modern cloud methods

The MEC platform enables MEC applications to:

- discover available services
- consume services
- advertise services that the application can provide

It is also responsible for

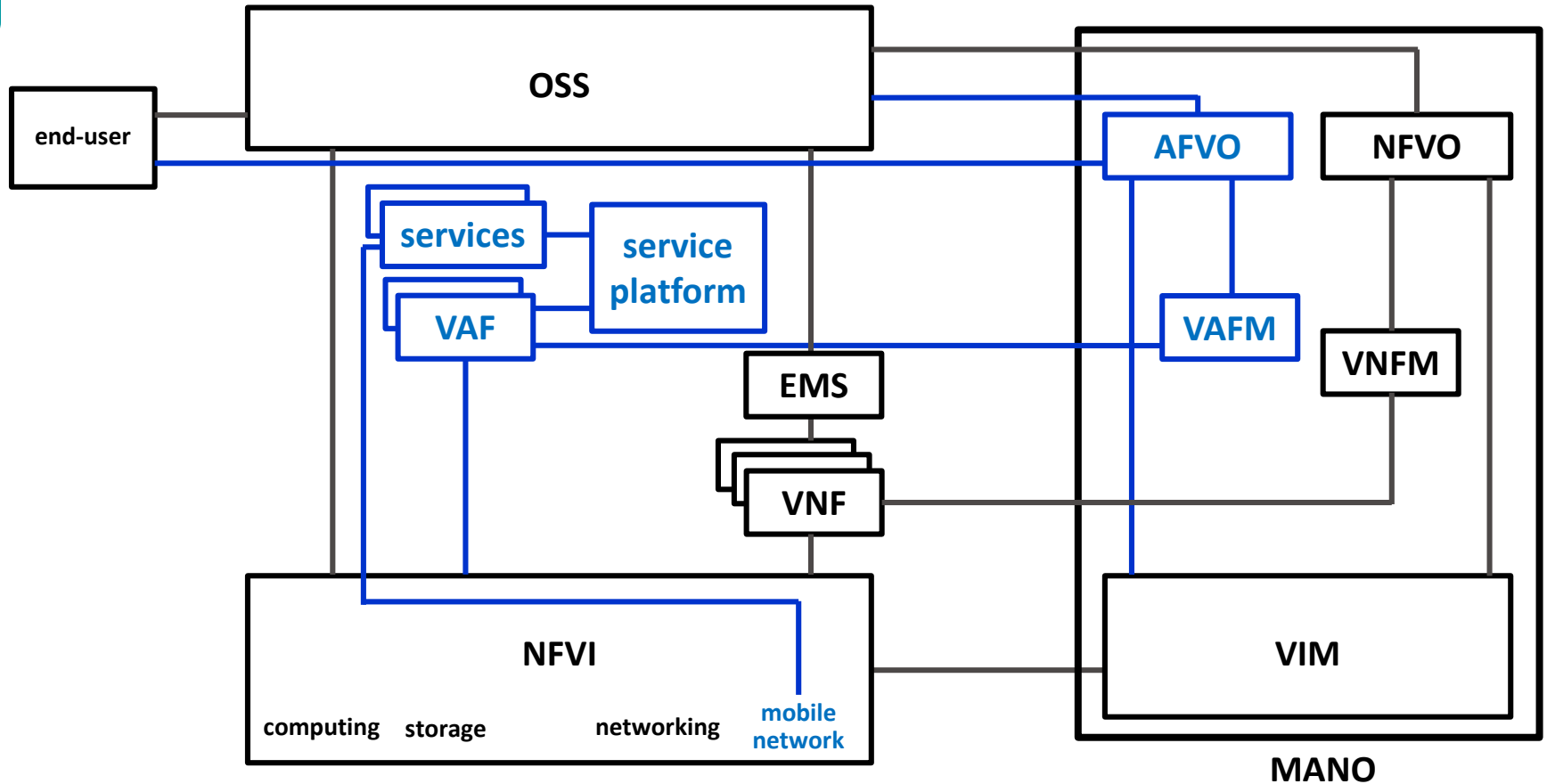
- steering traffic between chained applications
- apply traffic forwarding rules
- configure forwarding plane and DNS based on policies  
this includes using DNS proxy to direct user traffic to MEC application

We will see (when studying the 5G core)

that 5G's **S**ervice **B**ased **A**rchitecture learned from MEC principles  
and in particular provides a **N**etwork **E**xposure **F**unction

# An NFV approach

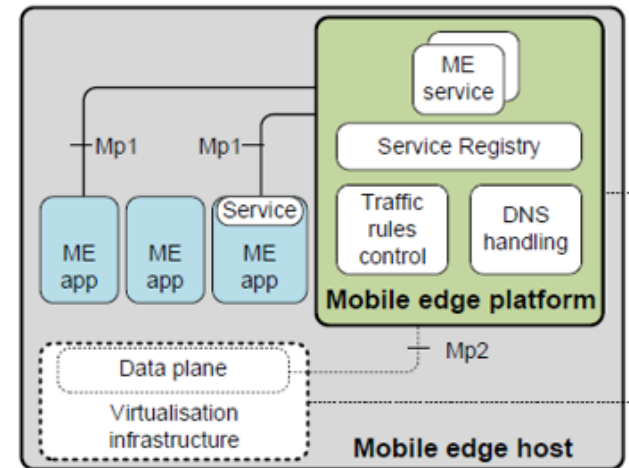
One could envision MEC as an extension to the standard NFV model



# Mobile edge host

The Mobile edge host is composed of:

- the NFVI
  - server
  - virtualization
  - persistent storage
  - networking software and hardware
  - time-of-day clock
- the mobile edge applications and services
- the mobile edge platform (already discussed)

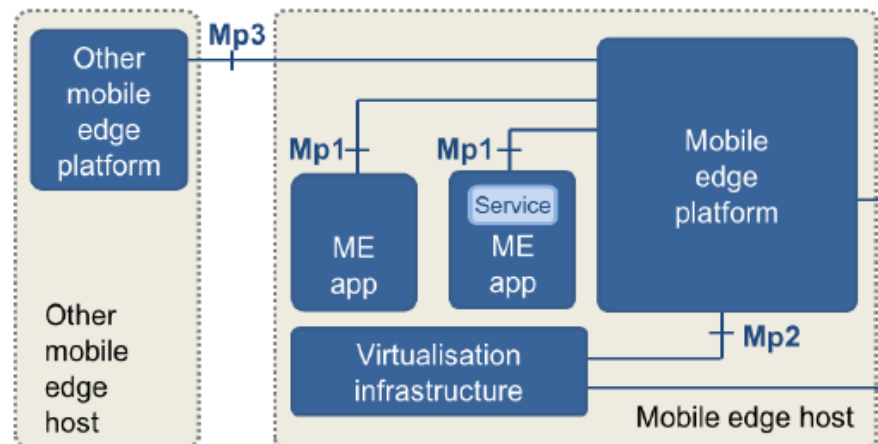


# Mobile Edge Platform (MEP)

The ME platform includes baseline functionalities needed to ME applications

- environment for service discovery, advertisement, consumption
- receiving traffic rules from MEPM/apps/services and instructing forwarding plane
- receiving DNS records from MEPM and configuring DNS proxy/server
- hosting services, e.g., location, RNI, bandwidth management

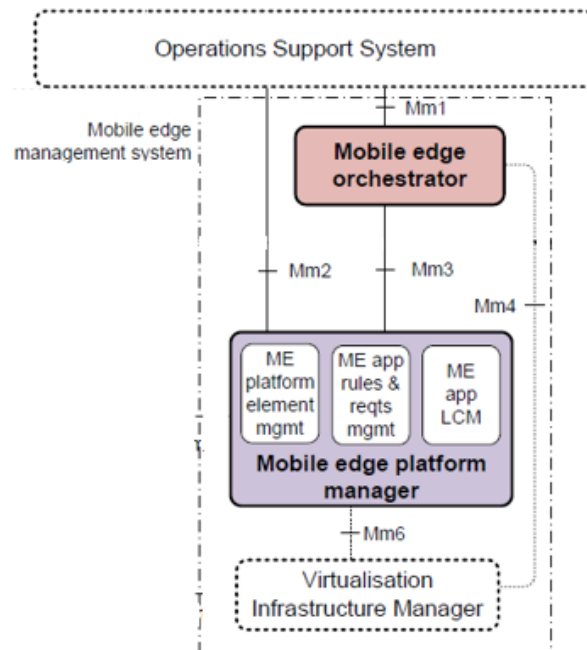
Different ME platforms can communicate via the Mp3 interface



# Mobile Edge Platform Manager (MEPM)

The MEPM is responsible for:

- managing application life-cycle
- informing the MEO of application related events
- managing service authorization, traffic rules, DNS configurations
- receiving and processing FM and PM reports from the VIM



# Mobile Edge Management System

The heart of the MEMS is the Mobile Edge Orchestrator

The MEO is essentially what we called the AFVO, and it is responsible for:

- maintaining database of resources, hosts, available services
- maintaining topology
- on-boarding new applications
  - authenticity / integrity checking
  - comparing application rules with operator policies
  - instructing the VIM on application specific issues
- selecting ME host for instantiation based on availability and latency
- triggering application based on UE application
- terminating application
- relocating application

