THE EVOLUTION OF PROGRAMMABLE NETWORKS: FROM ACTIVE NETWORKS TO SOFTWARE DEFINED NETWORKS (SDN)

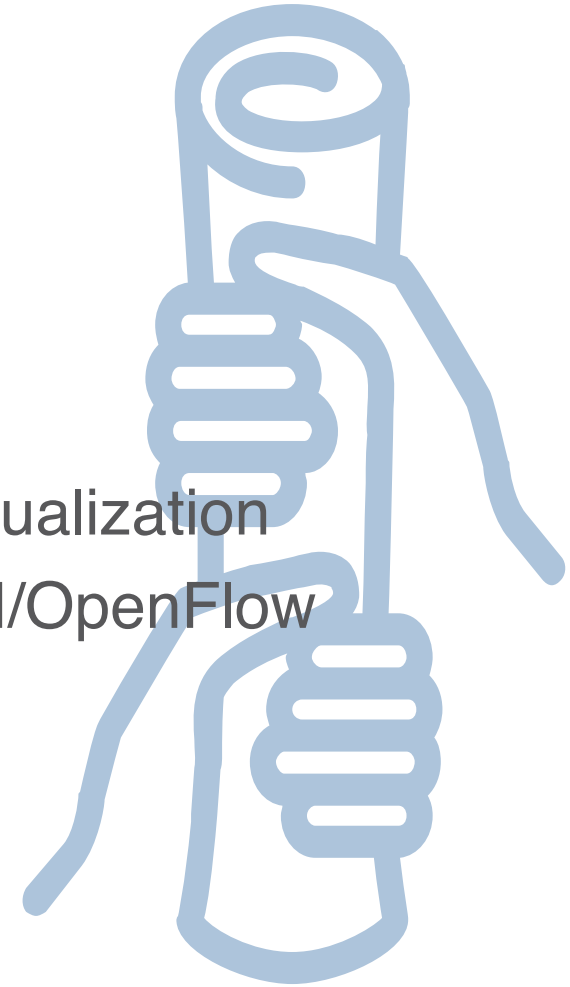
Ahmad Rostami, PhD
Ericsson Research, Stockholm
ahmad.rostami@ericsson.com

OUTLINE OF TUTORIAL – I

- Programmability in Telephony Networks
- Introduction to Programmable Networks
  - Definition (***What***) & ***Why*** we need that
  - How can we evaluate different approaches
  - Approaches to PN (***How***): A Quick Retrospective Review
    - Active Networks
    - (G)MPLS
    - ForCES

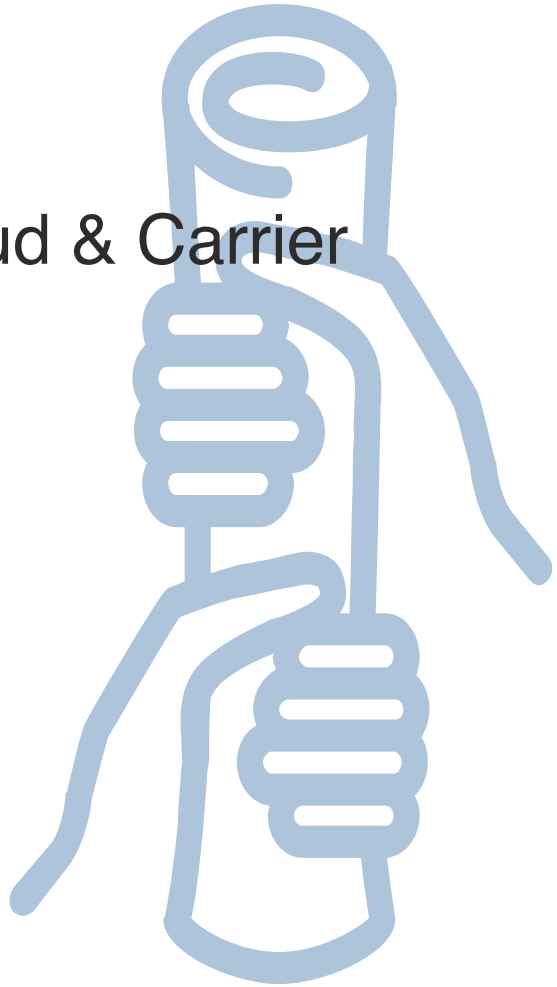
OUTLINE OF TUTORIAL – II

- Software Defined Networking (SDN)
  - Introduction & Basic Concept
  - SDN Realization Example: OpenFlow
  - SDN/OpenFlow Benefits and Issues
  - SDN Controllers
  - SDN Use-case Example: Network Virtualization
  - Programmability Beyond Current SDN/OpenFlow



OUTLINE OF TUTORIAL – III

- Network Function Virtualization (NFV)
  - Relation to SDN
- SDN and NFV in Action: Unifying Cloud & Carrier Networks
- Pointers to Relevant Bodies

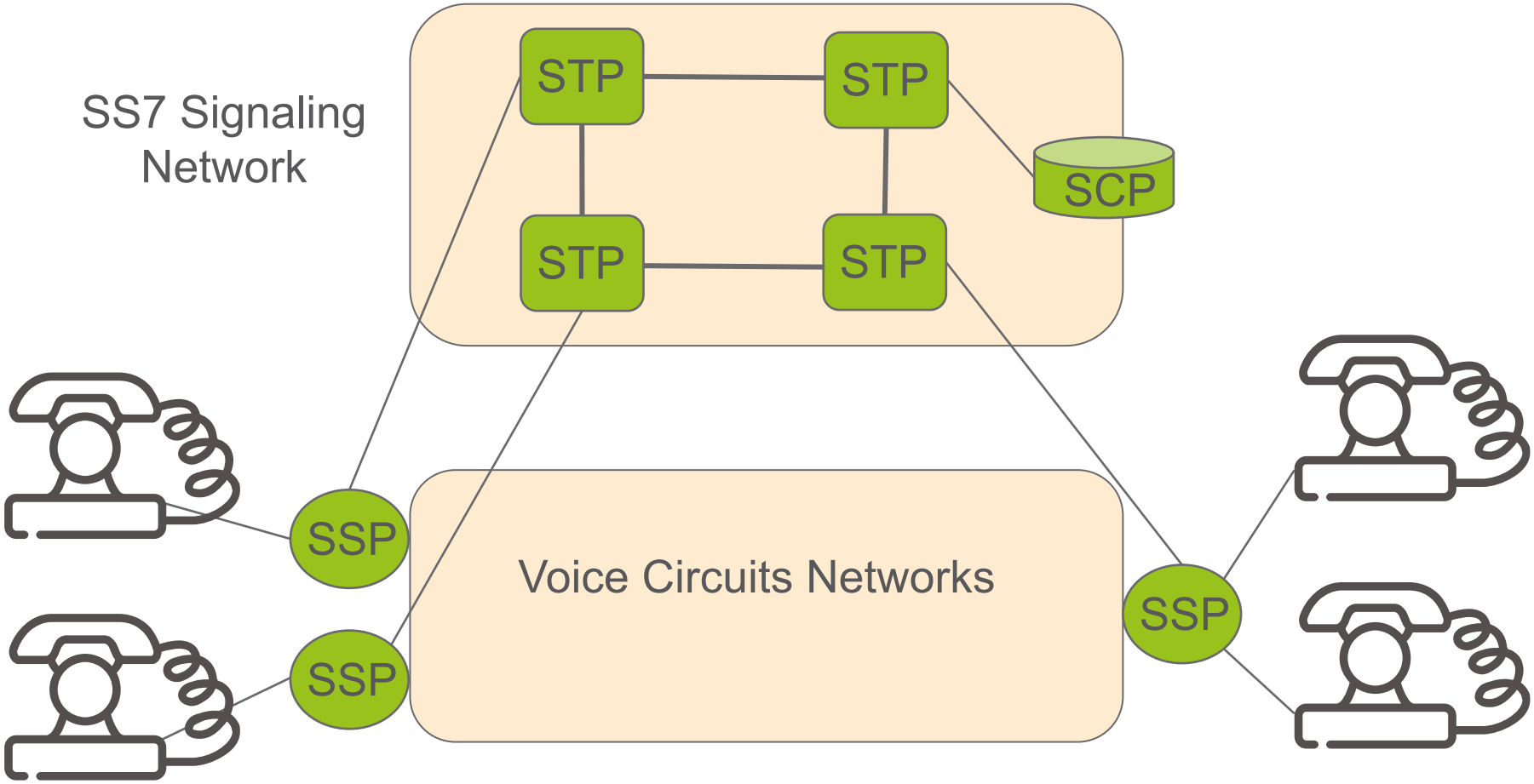


A QUICK LOOK AT THE CONTROL IN TELEPHONY NETWORKS

—OR—

HOW WAS PROGRAMMABILITY INTRODUCED INTO TELEPHONY NETWORKS MORE THAN 20 YEARS AGO?

SIGNALING SYSTEM NO. 7 (SS7)



SSP: Service Switching Point
STP: Signal Transfer Point
SCP: Service Control Point

INTELLIGENT NETWORKS (IN) – I

Value-Added Telephony Services

Introduction of digital switches in 80´s opened opportunities for offering new value-added services: call forwarding, call waiting, …

Approach & Shortcomings

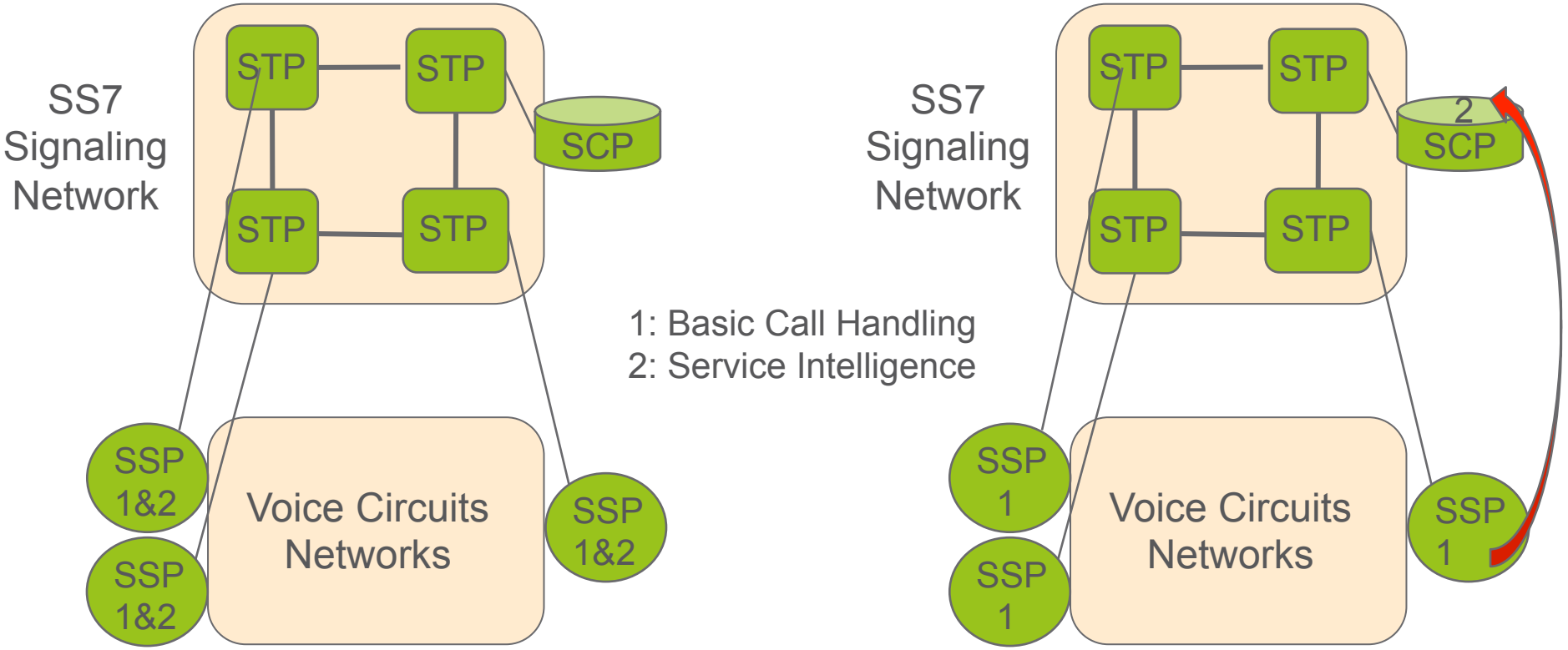
The main approach to new services was based on the pre-programmed switches (SSPs). Lack of Programmability of Switches led to lengthy process of creating new Services.

Introduction of IN

Decouple the service intelligence from the basic call control in a switch.

Push the service intelligence to (a few) centralized location inside the network.

INTELLIGENT NETWORKS (IN) – II

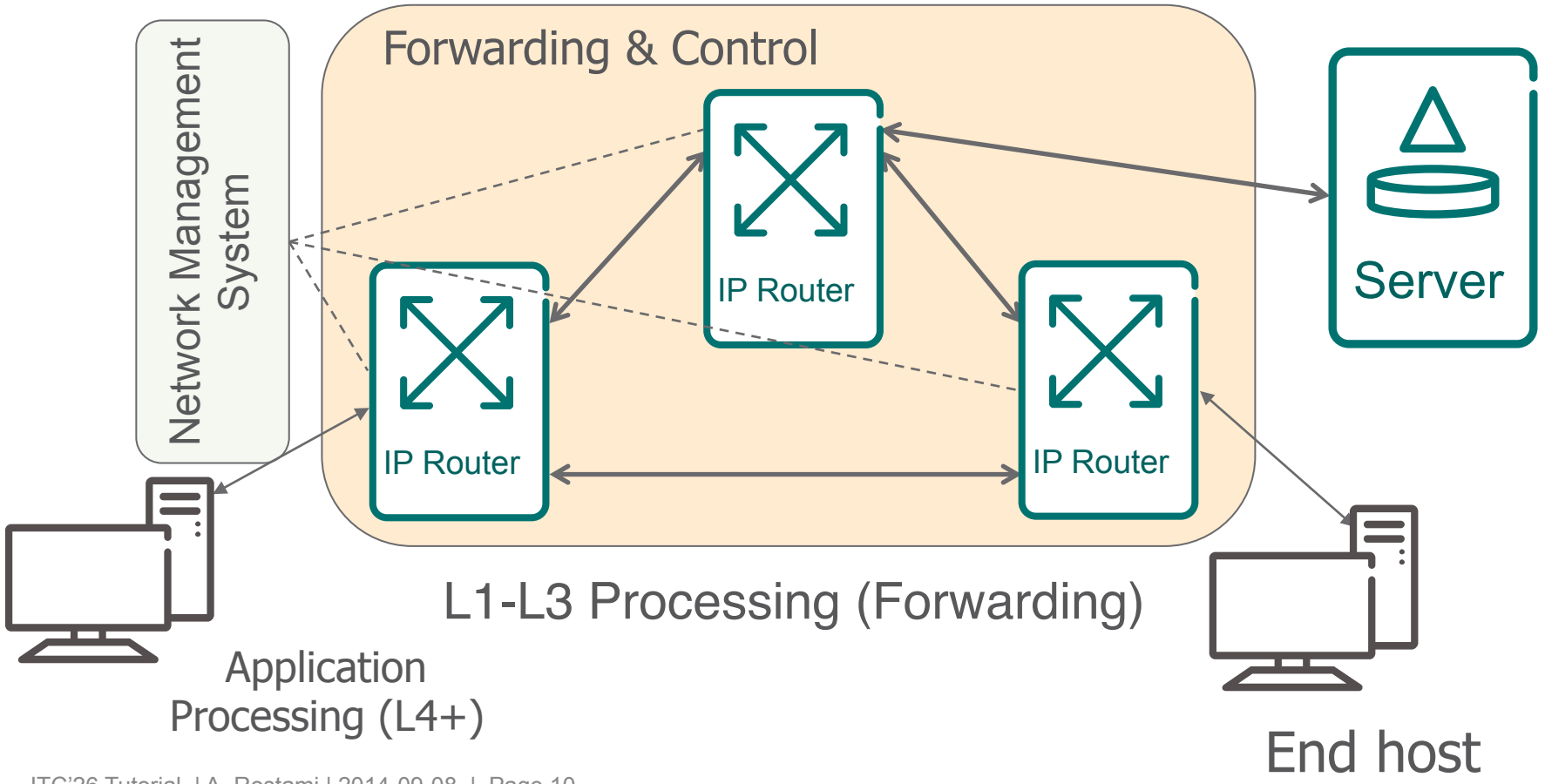


Making the call control programmable enabled (facilitated) many interesting value-added services, e.g., 0800-xxx numbers.

INTRODUCTION TO NETWORK PROGRAMMABILITY

CLASSICAL INTERNET ARCHITECTURE

Internet: an interconnection of comm. nodes each *monolithically* realizing *forwarding & control* based on a *layered architecture*



INTERNET ARCHITECTURE SIMPLICITY

- Simplicity as a key driving force of the Internet success
  - Network provides communication between nodes as the best effort service
  - Keep the network as simple as possible, push the (service) intelligence to the edge

- BUT, advanced services need more support from the network → Several patches and fixes (look at IETF)

- Is the Internet architecture still simple?

ISSUES WITH THE INTERNET ARCHITECTURE — I

- *Complexity of Network Control and Management*
  - *Layered architecture: blessing and cursing*
  - *Redundant & contradicting functions at different layers*

- *Complexity of Changing Network Control*
  - *Introducing new protocols/protocol stacks is hard*
  - *Consider how an operator can apply a customized routing mechanism*

ISSUES WITH THE INTERNET ARCHITECTURE — II

- *Complexity of Creating New Services*
  - *Mapping service requirements to vertically integrated devices is difficult*
  - *Applications may need cross-layer interactions*
  - *Applications may need L4+ in-network processing*
  - *Workarounds using inefficient overlays or middle-boxes*

Need for more flexibility/adaptation in the network
→ Programmable Networks

WHAT IS NETWORK PROGRAMMABILITY?

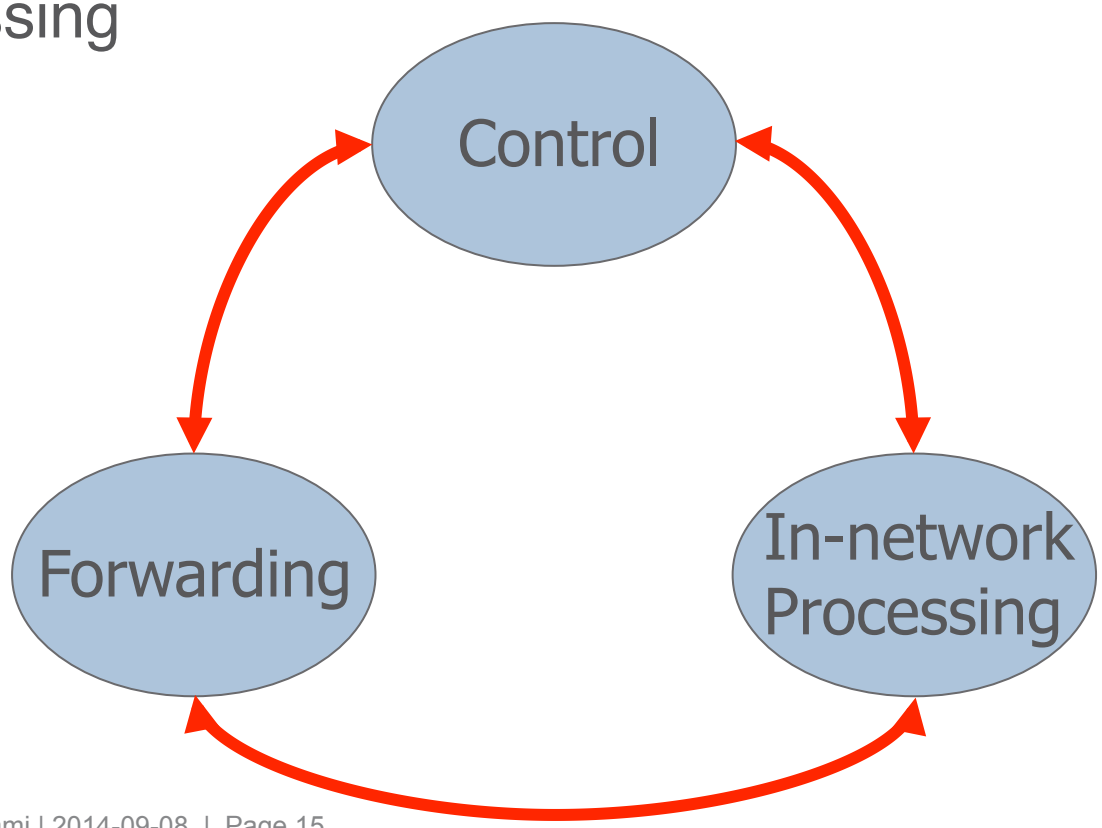
- The ability to *dynamically* introduce a new architecture, protocols and services on top of a networking infrastructure (without requiring manual configurations)

- Examples of programmability:
  - Customize treatments for a group of packets
  - Assign new semantics to pieces of packets' headers

- Network Programmability has been a hot topic for research for the last 20+ years, influenced a lot by computing technology

NETWORK PROGRAMMABILITY: HOW? (I)

- Restructure the main blocks/functions
  - Where to place the control intelligence?
  - Balance between basic forwarding and In-network processing



NETWORK PROGRAMMABILITY: HOW? (II)

- Design (standard) interfaces between the architectural components
- Select the components that should be open to programming
  - Programming Interfaces
  - Programming levels
  - Who should take control?
- Important factors: Flexibility, Scalability, Security, Openness, Cost

APPROACHES TO NETWORK PROGRAMMABILITY

Active Networks

- Initiated by academia in mid 90s
- Not deployed in production networks

(G)MPLS*

- Initiated mainly By vendors in late 90s
- Standardization going on in IETF since early 2000s
- Deployed in production networks

ForCES

- Initiated mainly by vendors in early 2000s
- Standardization going on in IETF since mid 2000s
- Not deployed in production networks

SDN

- Initiated mainly by academia in late 2000s
- Standardization going on in ONF since early 2010s
- Partially Deployed in production networks

*(G)MPLS is not designed explicitly as an approach to programmable networks. But, it applies principles which are fundamental to network programmability.

EVALUATION FRAMEWORK

- What part is programmed?
  - Forwarding plane, Control plane, …
- What is the level of programmability?
  - Packet, Flow, Class
- Who takes the control over programmability?
  - Network Operator, Service Provider, End User
- Methodology: How is the network programmed?
  - Separation of control and forwarding plane
  - Logical vs. physical separation & centralized vs. distributed control
- Availability and extent of In-network Processing

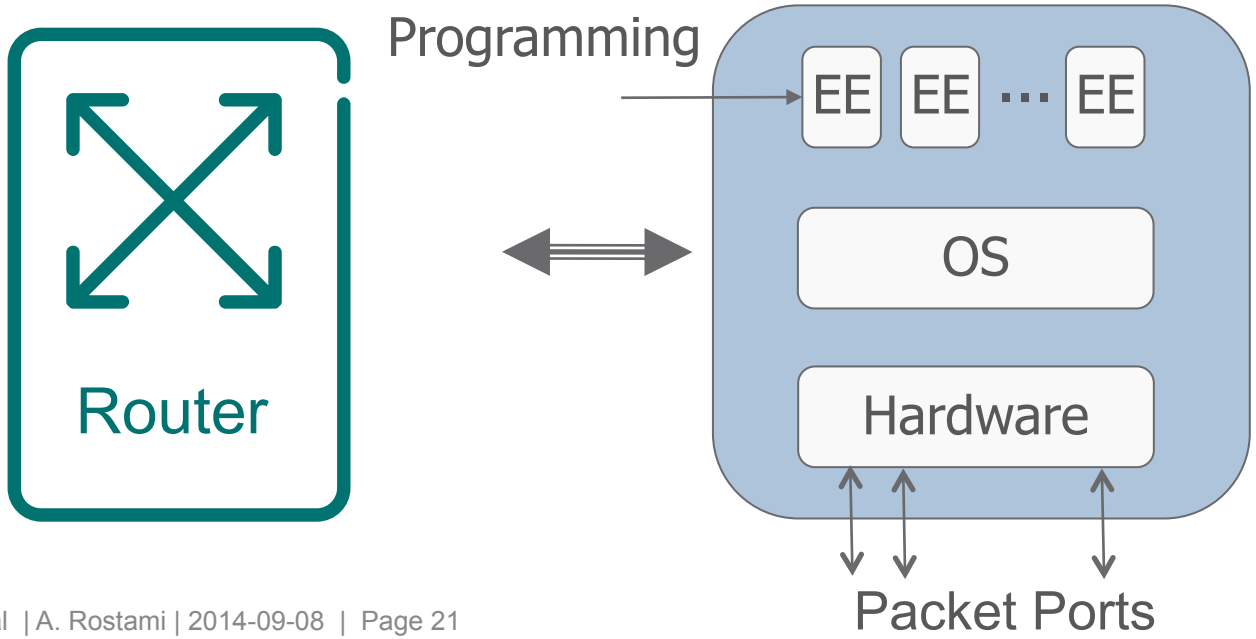
ACTIVE NETWORKS

ACTIVE NETWORKS – BACKGROUND

- Research on Active Networks began in early 90's and supported by several projects
- Research on Active Networks was motivated by the widespread use of the Internet (applications) & complexity in testing new ideas in real networks
- To a high extent influenced by the approach in the computing (PC) industry
- Active Networks requires drastic changes to the existing networking nodes & architecture

ACTIVE NETWORKS PRINCIPLES

- Replace ad-hoc intelligence in network devices with *generic execution environments (EE)*
- Expose an interface (e.g. instruction set) to the EE: used to realize a variety of service-specific packet processing



PROGRAMMING THE PROCESSING

- Service providers &/Or end users (applications) directly *program* network nodes via access to EE to realize desired functionality
- Customized processing on packets: traffic *actively* select how it is treated/processed

- **Programming Approaches:**
  - ***Integrated:*** codes are carried together with data packets

  - ***Discrete:*** Code is first loaded into nodes through out-of-band (management) interface and then data packets are injected.

ACTIVE NETWORKS – SUMMARY I

- Aggressive approach with the highest level of flexibility
- Programmability directly at the data plane
- Challenging resource management
  - Balance between processing and forwarding in nodes
- Complexity
- Network operator concerns (e.g., inconsistent applications/codes, security)
- Lack of a killer application at the time

ACTIVE NETWORKS – SUMMARY II

- Who Programs the network?
  - *End user, service provider*
- What is programmed?
  - *Network elements, both control and forwarding*
- Level of programmability
  - *Packets*
- Methodology
  - *No clear separation of control & forwarding*
  - *Low-level interface to network elements*
- In-network Processing
  - *In every node*

Active Networks contributed to the evolution of programmable networks, but as an approach it was not successful.

(GENERALIZED) MULTI-PROTOCOL LABEL SWITCHING (GMPLS)

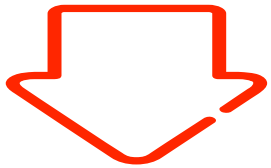
(G)MPLS – BACKGROUND

- Motivation for MPLS introduced in early 2000
  - Concerns about scalability of IP packet processing in the backbone
  - Need to attach some notion of connection to traffic
  - Need for more control over the traffic routing & for traffic engineering
- Started for packet networks and extended to other switching technologies like TDM and optical circuits → GMPLS

- A different approach than active networks
  - Limited flexibility in packet processing

(G)MPLS – PRINCIPLES

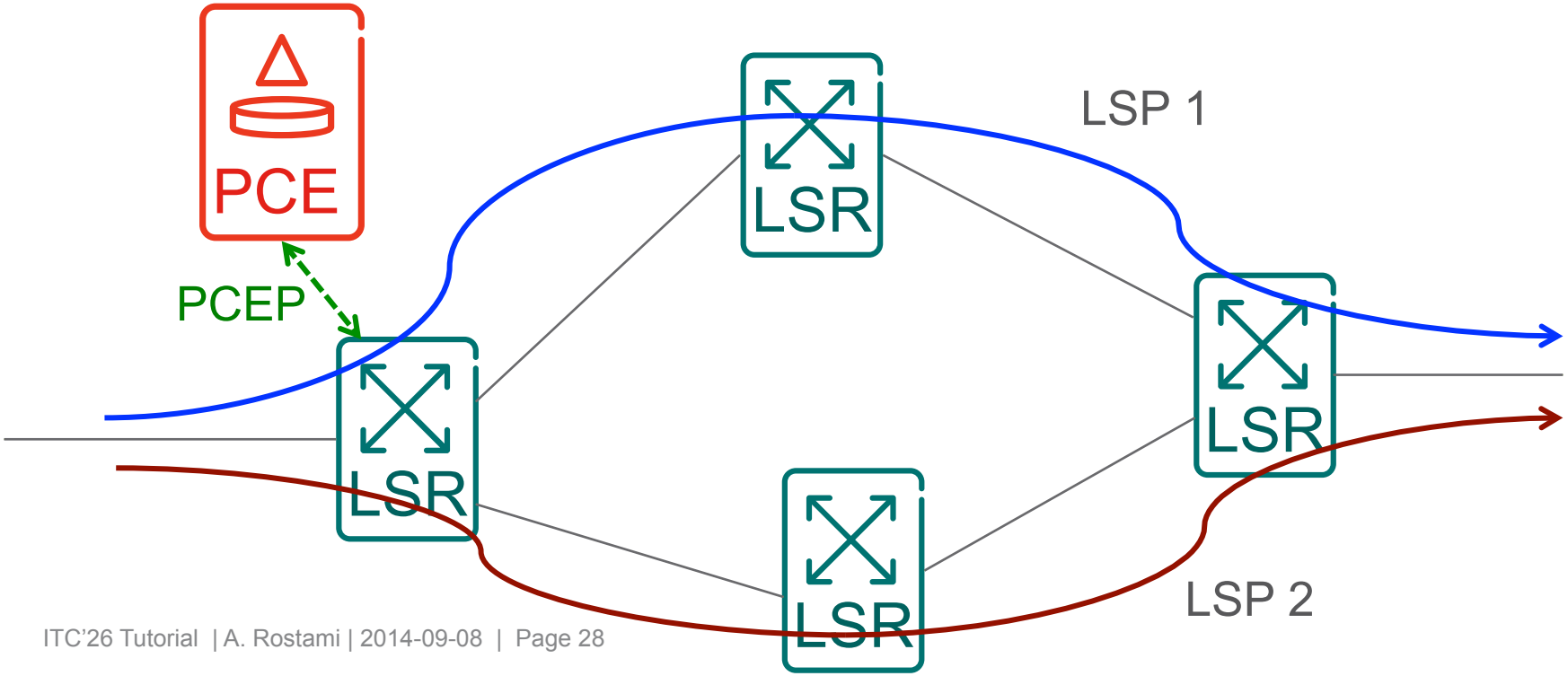
- (Logical) separation of control & forwarding planes
- Simplify the forwarding plane (in comparison to IP)
- Push the intelligence/processing from core to edge of networks



- Partition the entire set of possible packets into a set of Forwarding Equivalence Classes (FECs)
- Select a path for each FEC→ Labeled Switched Path (LSP)
- At edge of network: process packets and assign them to FECs and label them
- In core: forward packets only based on their FEC & associated LSPs

TRAFFIC ENGINEERING IN (G)MPLS NETWORKS

- Partitioning traffic space into FECs (and LSPs) facilitate more flexible routing and resource allocation in the network
  - → constraint-based path selection
- (Logically) Centralized Path Computation Element (PCE)



(G)MPLS – SUMMARY I

- Separation of control and forwarding
  - A step forward towards flexible network control
- Centralized PCE
- Programmability limited to constraint-based path computation and
  - No customized control beyond routing
  - No in-network processing

(G)MPLS – SUMMARY II

- Who Programs the Network?
  - *Network Operator*
- What is programmed?
  - *Forwarding treatment, path selection*
- Level of programmability
  - *Traffic classes*
- Methodology
  - *Logical separation of control & forwarding*
- In-network Processing
  - *Not available*

FORWARDING & CONTROL ELEMENT SEPARATION (FORCES)

FORCES – BACKGROUND

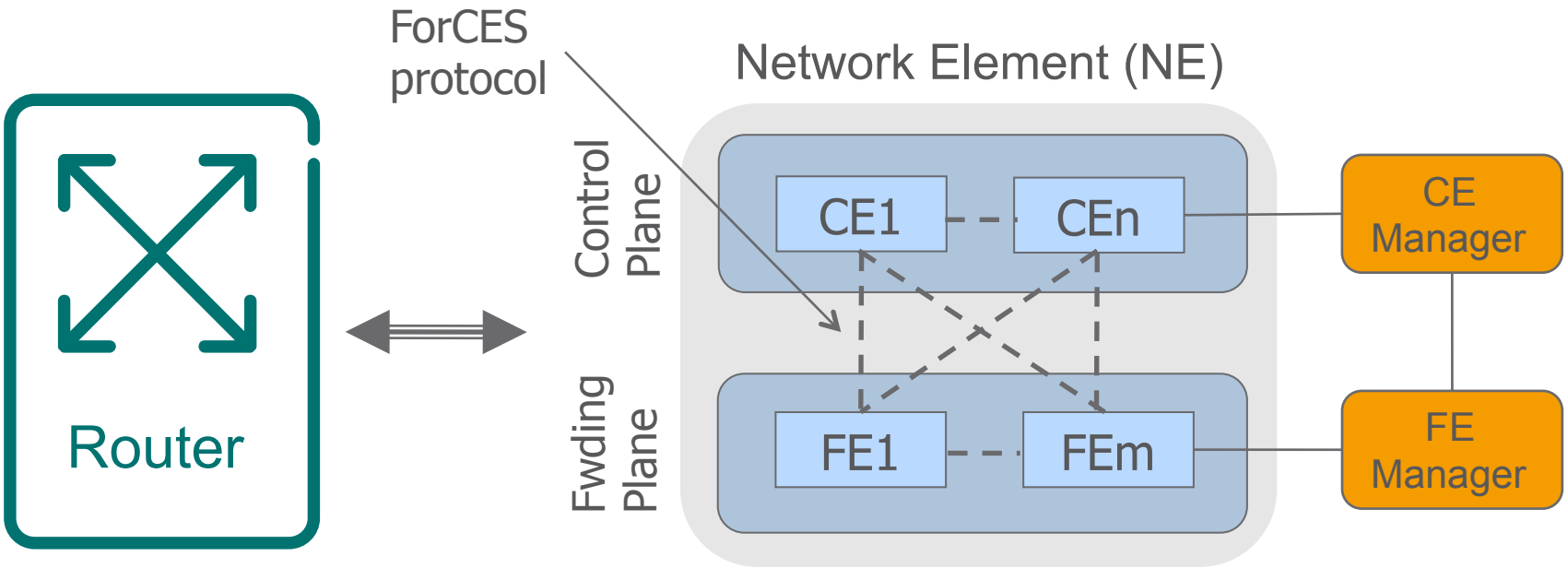
- Motivations for ForCES introduced in early 2000
  - Separating Control & Forwarding in network elements
  - Powerful Network Processing Units (NPUs) allowing realization of customized functions in data plane
  - Need for a modular functional architecture describing components in data and control planes
  - Need to specify (& standardize) the interface between components of the functional architecture

- A Modular architecture supporting network programmability
  - A great support for in-network processing
  - Object-oriented programming concept applied to the nodes' components management

FORCES – PRINCIPLES

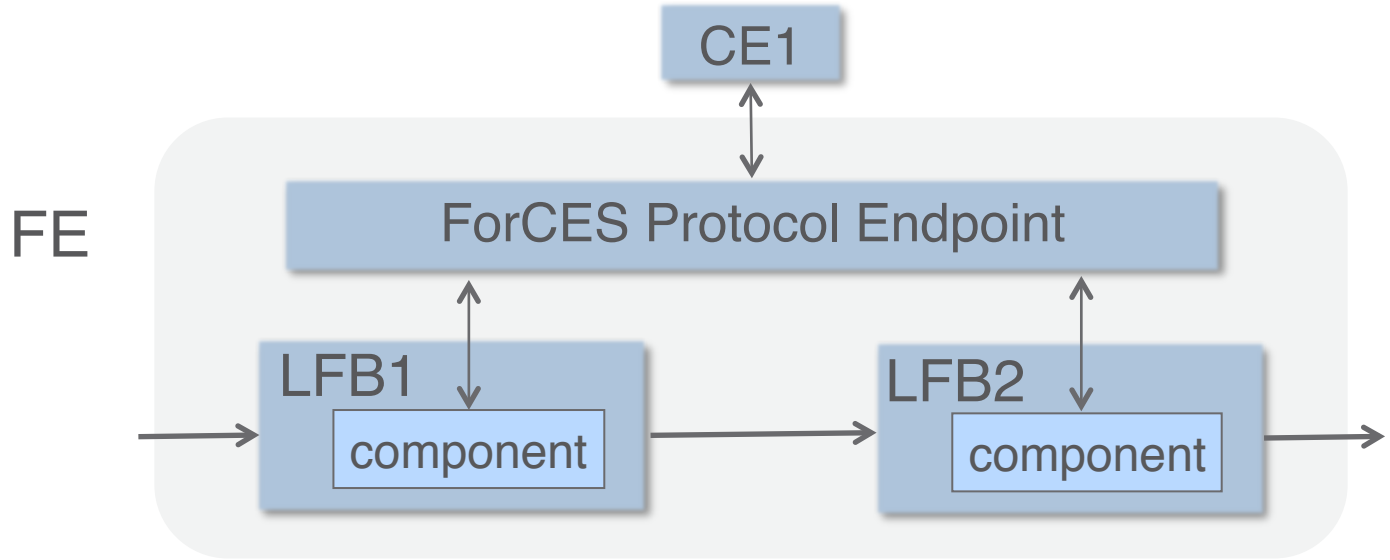
› (Logical) separation of control & forwarding planes

› Modular (& dynamic) interconnection of architectural elements



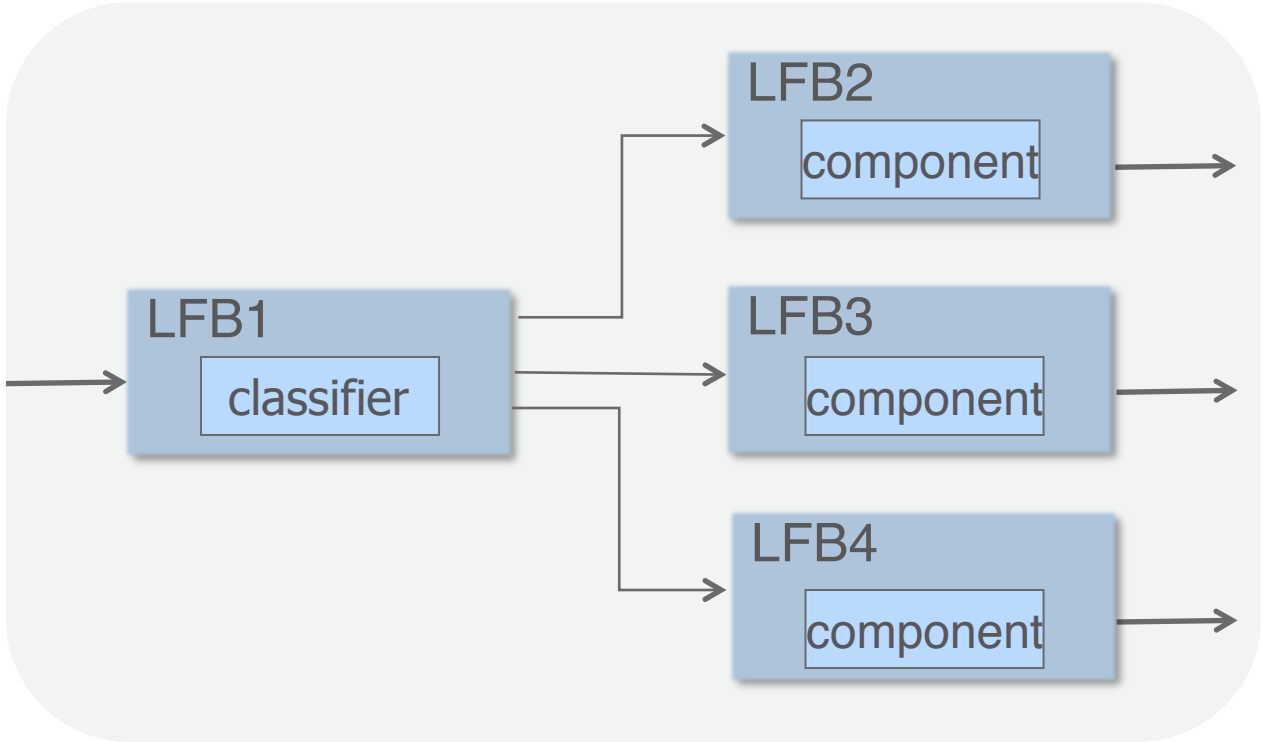
› Multiple instances of control elements (CE) & forwarding elements (FE) in an NE → interconnected over variety of technologies

PROGRAMMABILITY WITH FORCES



- Forwarding model is based on an abstraction of logical functional blocks (LFB)
- Each LFB defines a single action on passing packets
- FE is composed of multiple LFBs interconnected in a directed graph
- A CE can *dynamically* create, instantiate, update or delete LFBs within a FE from LFB library

SERVICE COMPOSITION WITH FORCES



› The outcome of a LFB processing determines how a packet will be processed in downstream LFBs

→ Flow/Class-based Programming

FORCES – SUMMARY I

- Standard interface between control and forwarding
  - Independent evolution of the two components
- More flexibility & generality than MPLS
  - MPLS functions can be implemented in ForCES
- Supports a gradual deployment of ForCES nodes
  - Transparent interconnection to conventional nodes
- Support for in-network processing
- No programming interface to control plane

FORCES – SUMMARY II

- Who Programs the Network?
  - *Network operator*
- What is programmed?
  - ***Forwarding** & control*
- Level of programmability
  - *Class/Flow*
- Methodology
  - *Logical separation of control & forwarding*
  - *Modular functional architecture*
  - *Standard interface between modules*
- In-network processing
  - *Possible at every node*

SOFTWARE DEFINED NETWORKING (SDN)

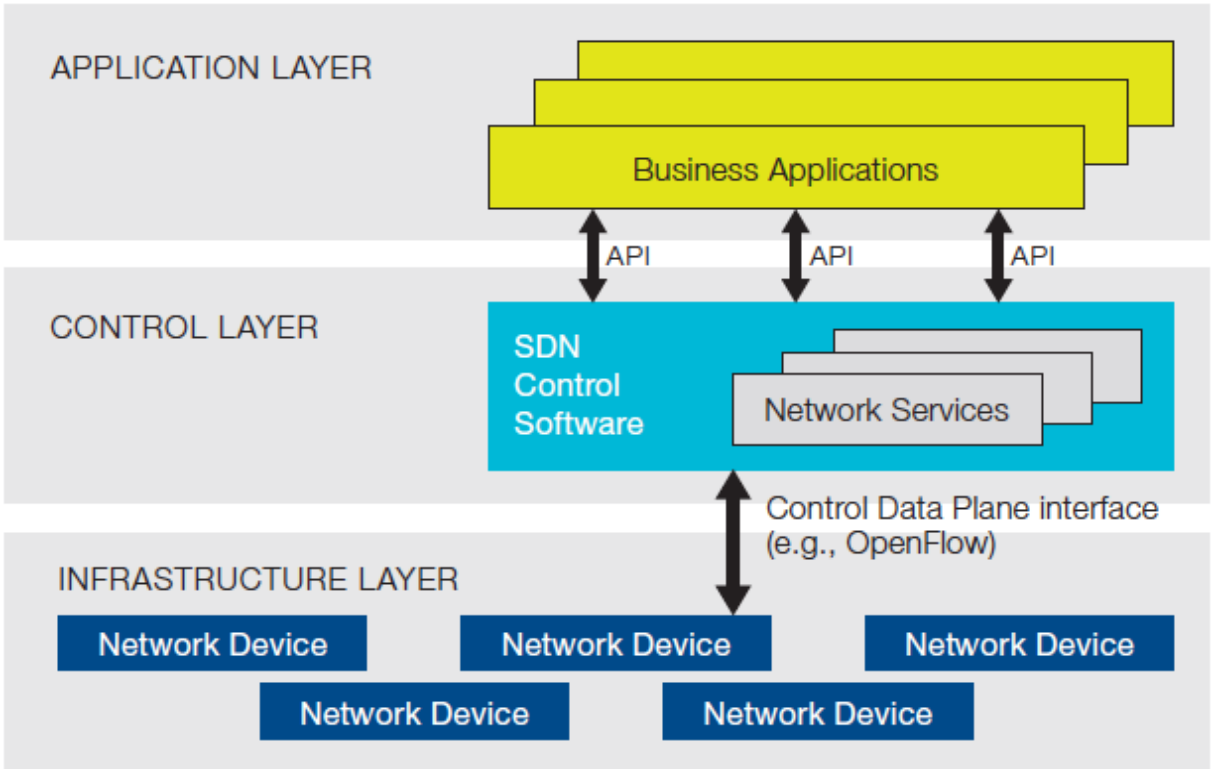
SDN – BACKGROUND

- Motivations for SDN introduced in late 2000
  - Separating control and forwarding (!)
  - No success with previous attempts to programmable networks
  - The need for (large scale) experimental networking platforms

- SDN borrows several concepts from previous approaches to programmable networks & computing, e.g.,
  - Separation of control from forwarding
  - High level programming interfaces

SDN – PRINCIPLES

- Physical separation of control & forwarding
  - Specify an (open standard) interface between control and forwarding
- Delegate control to a logically centralized controller
- Abstract the network resources from higher layer applications

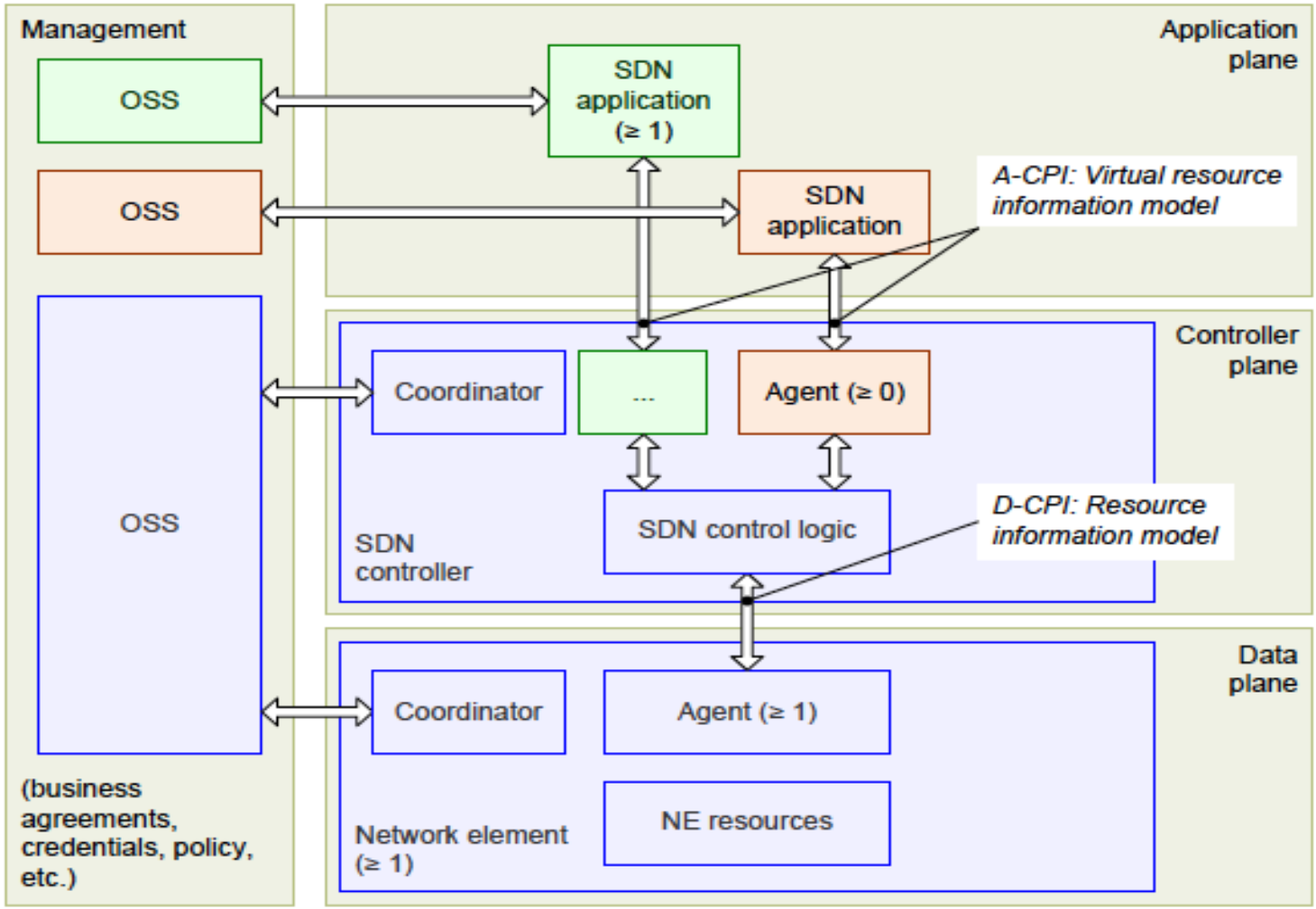


Source of Fig.: ONF white paper on SDN

SDN ABSTRACTION LAYERS

- SDN makes use of abstraction layers to simplify the programmability
- Forwarding abstraction
  - Design a Control-Forwarding interface & centralize the control logic
- Control-plane abstractions
  - Hide details of network resources & state distribution from control applications
- There are more than one way to do the abstractions
- Abstractions can be designed to allow a recursive control architecture

SDN ARCHITECTURE – ONF VIEW



Source: ONF SDN Architecture, June 2014

BENEFITS OF SDN – I

- Control plane can be directly programmed
  - Customized control logics
  - Rapid creation of new services
- Simplified programming of new applications
  - Programming on an abstract view of network
  - State distribution carried by the controller (Network OS)
- Enabling more efficient control applications
  - Controller provides a global view of resources

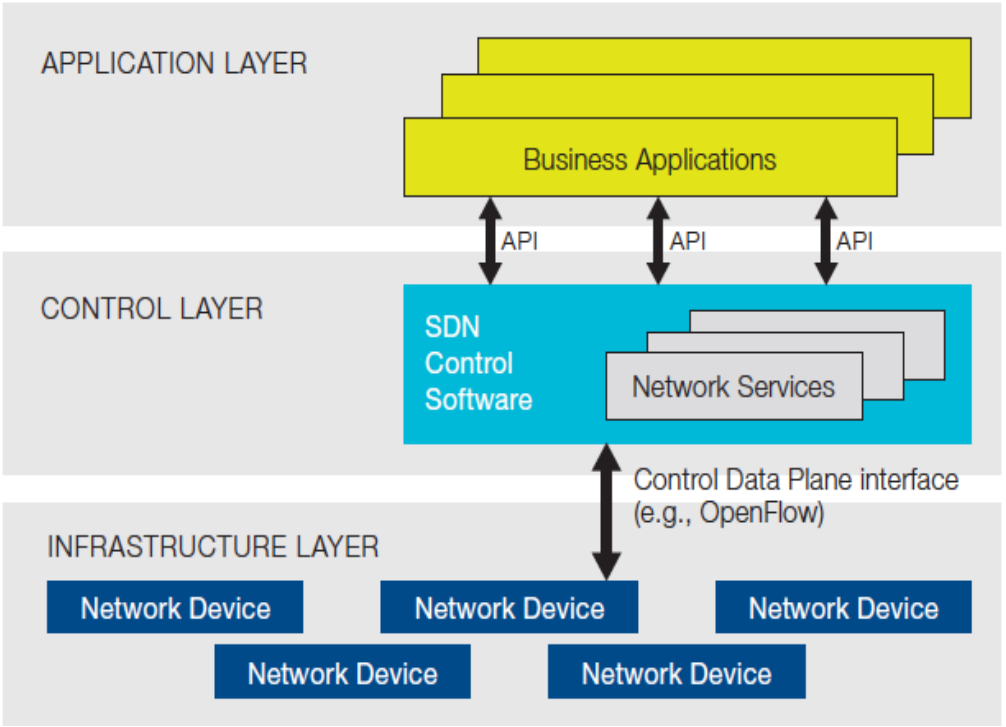
BENEFITS OF SDN – II

- Vendor independent network control
  - Using a standard API between forwarding & control
- Simplified forwarding elements
  - Support only a generic API instead of many protocols
- Data & control planes can evolve independently
  - Higher rates of innovations
  - More competitive market

SDN REALIZATION EXAMPLE: OPENFLOW

SDN REALIZATION

- SDN is an architectural framework for programmability
- A realization of SDN requires concrete:
  - Data-plane abstraction & API
  - SDN Controller (Network OS)
  - Network abstraction and Control API



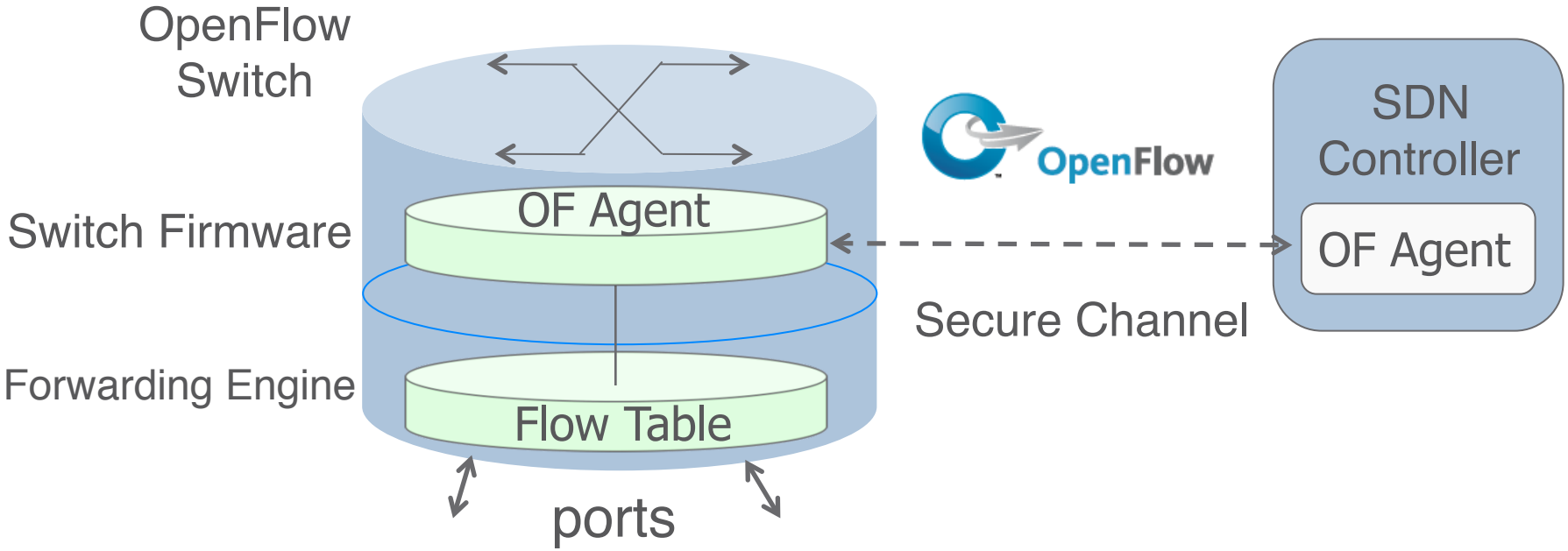
OPENFLOW — A CONCRETE REALIZATION OF SDN

- OpenFlow is one of the initial works leading to SDN
  - Started in academia in late 2000
  - Initially focused on utilizing Ethernet-based campus networks as an experimental platform
- OpenFlow specifies a forwarding plane abstraction together with an API to forwarding devices
- OpenFlow does not specify the network controller, nor does it specify a controller-application API

FORWARDING ABSTRACTION

› OpenFlow started with abstracting data plane of the Ethernet switch

› In OpenFlow a forwarding element is abstracted as a flow table applying the **Match+Action** principle



› OpenFlow specifies an instruction set to manage the flow table from a remote controller

FLOW TABLE

Match	Priority	Counters	Instructions	Timeouts	Cookie

› Match: A flow is flexibly defined as a group of packets sharing certain header values. Match + Priority → unique flow identifier

Ingress Port	Ether Src	Ether Dst	Ether Type	VLAN	IP Src	IP Dst	IP Proto	L4 Src	L4 Dst

› Instructions: modify actions set associated with a flow: sent to a port, modify a field, etc.

› Counters: collect statistics

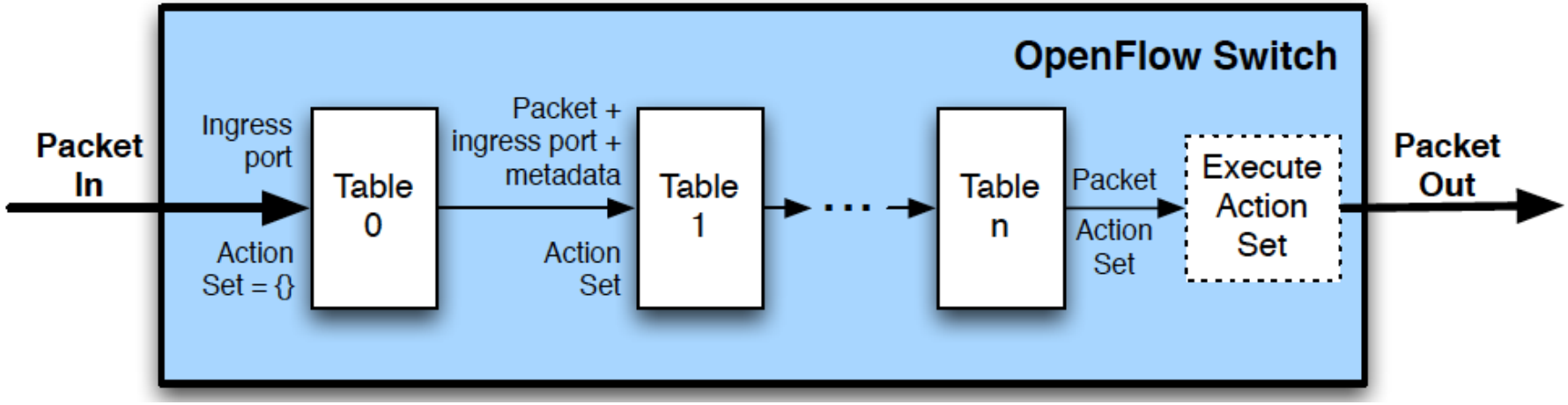
› Timeout: specify validity of a flow entry in time

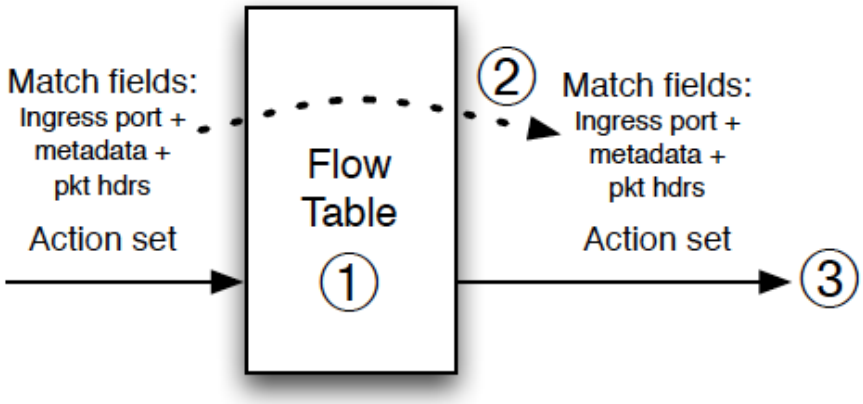
› Cookie: used by controller for filtering groups of flows

FLOW TABLE: AN EXAMPLE

Ingress Port	MAC Src	MAC Dst	Ether Type	VLAN	IP Src	IP Dst	IP Proto	L4 Src	L4 Dst	Actions
*	*	00…	*	*	*	*	*	*	*	Port 2
3	*	*	*	*	*	*	*	*	22	Drop
1	*	*	*	*	*	192.168.1.0/24	*	*	*	Port 5
*	*	*	*	*	*	*	*	*	*	Send to Controller

DATA-PATH PROCESSING PIPELINE

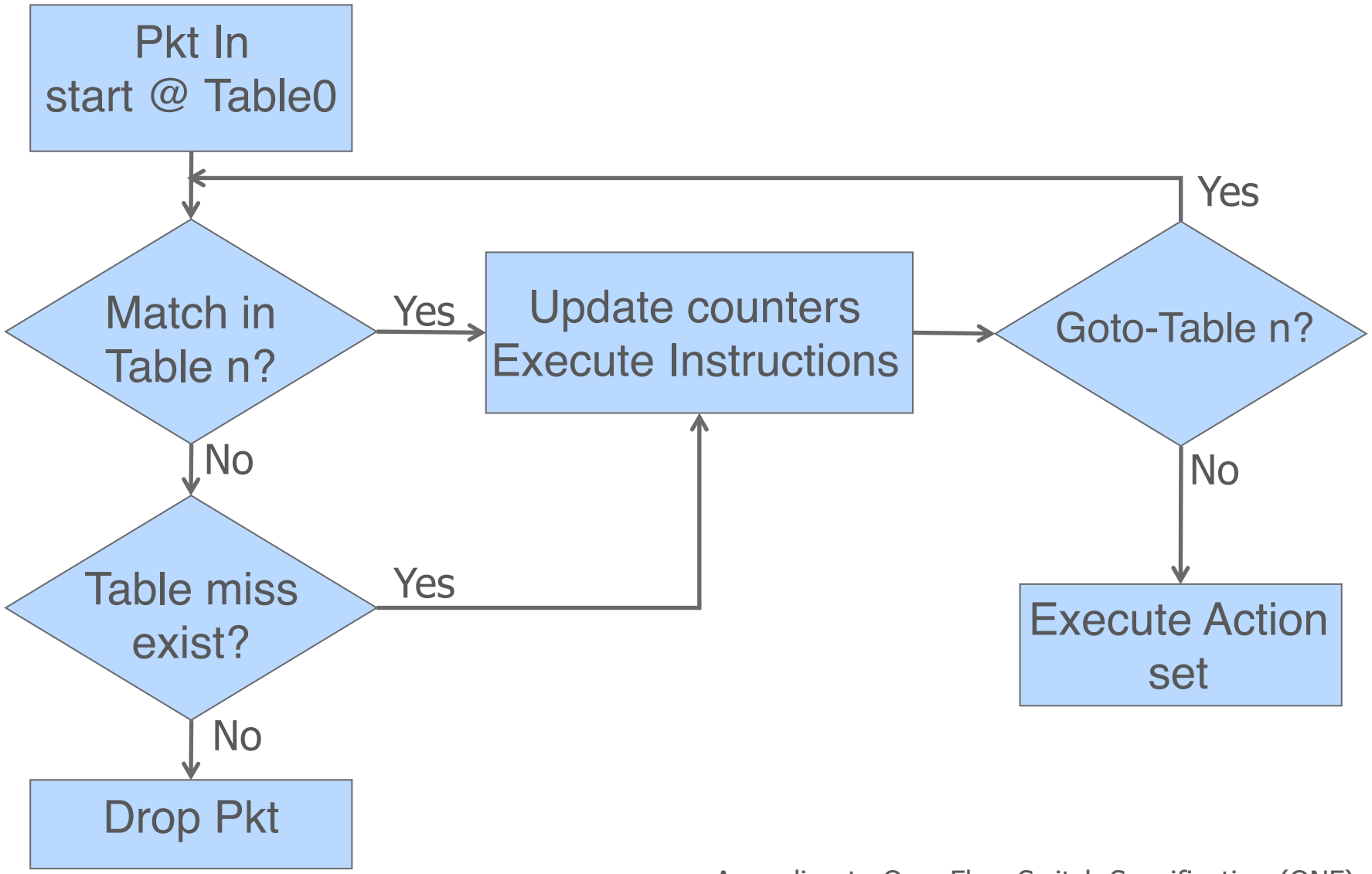




① Find highest-priority matching flow entry
② Apply instructions:
  i. Modify packet & update match fields
  ii. Update action set
  iii. Update metadata
③ Send meta data and action set to next table

Source: OpenFlow Switch Specification (ONF)

PACKET FLOW IN SWITCH



According to OpenFlow Switch Specification (ONF)

SDN/OPENFLOW BENEFITS

- Flow-level programming: OpenFlow enables flexible definition of flows & assignment of individual forwarding treatment to them
- Flat Control: OpenFlow collapses L1-L4 control into a single layer → multi-layer optimization
- Traffic Statistics per flow, table, port, …

- OpenFlow provides a powerful tool for efficient control of enterprise networks
- SDN/OpenFlow already applied to WANs & large datacenters

SDN/OPENFLOW ISSUES/OPEN PROBLEMS – I

- Programmability of data-path packet processing limited by OpenFlow API
  - No direct support for stateful processing in data-path
- Protocol dependent nature of OF API
  - Matching fields extended from 12 in OF 1.0 to 41 in OF 1.4
- No network abstraction atop OF API
  - Developing control modules still require dealing with low-level flow API

SDN/OPENFLOW ISSUES/OPEN PROBLEMS – II

- Scalability
  - Controller processing, flow-table size, OF channel, …
- Interface among Controllers
  - SDN/OpenFlow targets single domain
  - Multi-domain networks, distributed control
- Security concerns

SDN/OPENFLOW SUMMARY

- Who Programs the Network:
  - Service provider
- What is programmed?
  - (mainly) control plane
- Level of programmability:
  - Traffic flow
- Methodology:
  - decoupling control intelligence from datapath
  - Abstract forwarding based on “match+action” flow tables
  - An open interface between controller & forwarding
- In-network processing:
  - Not supported directly


SDN CONTROLLER

SDN CONTROLLER

- SDN Controller acts as a network operating system
  - Abstracting the network resources
  - Providing a programmatic interface for developing network applications in SW
  - Configuring networking resources as a service to network applications (state distribution)
  - Ensuring consistent usage of resources among control Apps
  - Enabling virtualization of networking resources

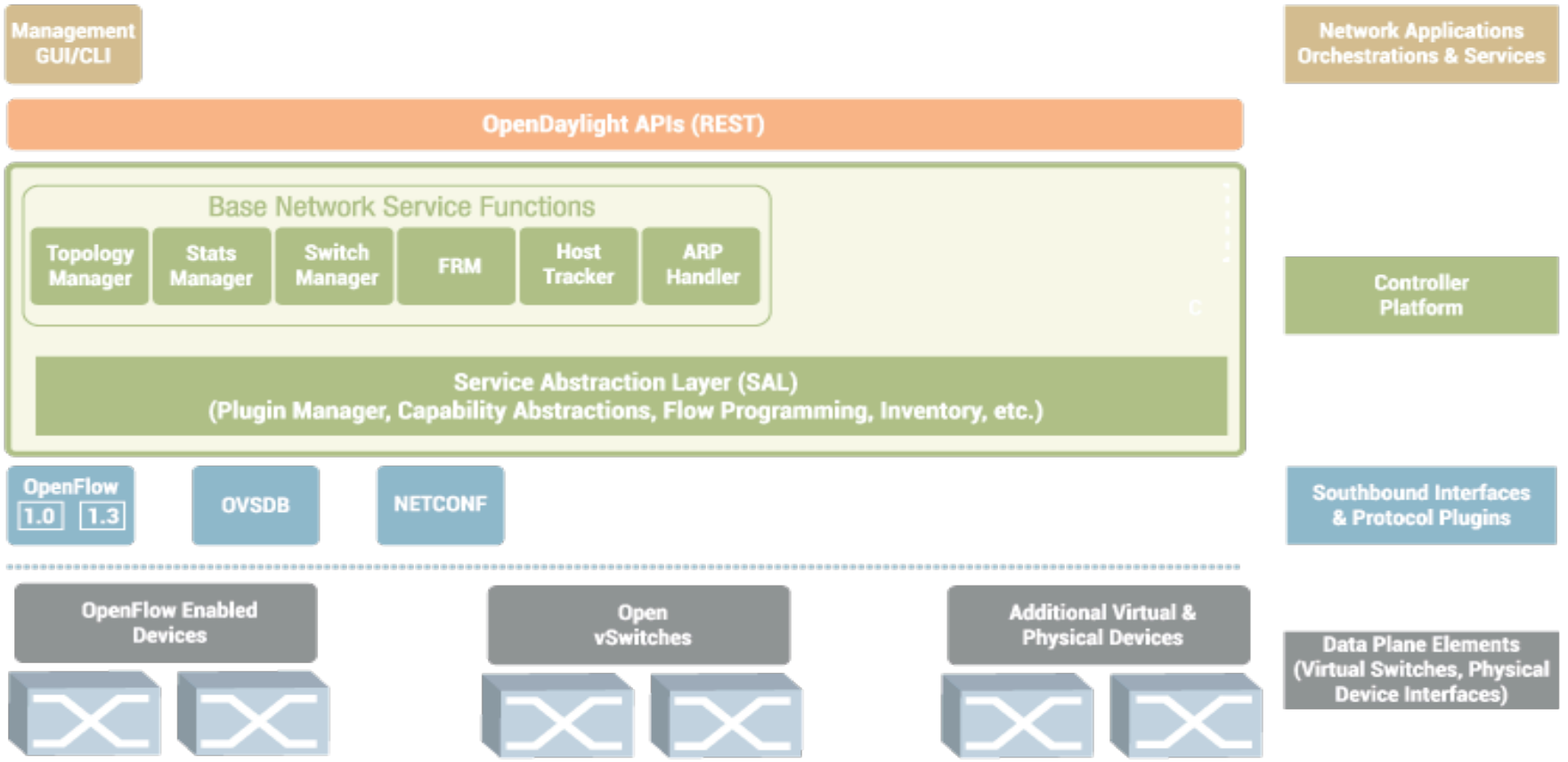
SDN CONTROLLER PLATFORMS

- Several commercial and educational SDN controllers developed in recent years, primarily focusing on OpenFlow
- **OpenDayLight (ODL):** An open source collaborative project started in 2013 to develop a common platform for SDN controller
- ODL supports several control-dataplane interfaces
- ODL project runs under the umbrella of Linux Foundation, with contribution from many companies:
  - Ericsson, IBM, Cisco, HP, Microsoft, Juniper, …
- First ODL implementation released Feb. 2014 with code name Hydrogen

OPENDAYLIGHT ARCHITECTURE

OPEN DAYLIGHT
"HYDROGEN"
BASE EDITION

VTN: Virtual Tenant Network
oDMC: Open Dove Management Console
D4A: Defense4All Protection
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase Protocol
BGP: Border Gateway Protocol
PCEP: Path Computation Element Communication Protocol
SNMP: Simple Network Management Protocol
FRM: Forwarding Rules Manager
ARP: Address Resolution Protocol



Source: OpenDayLight Project (opendaylight.org)

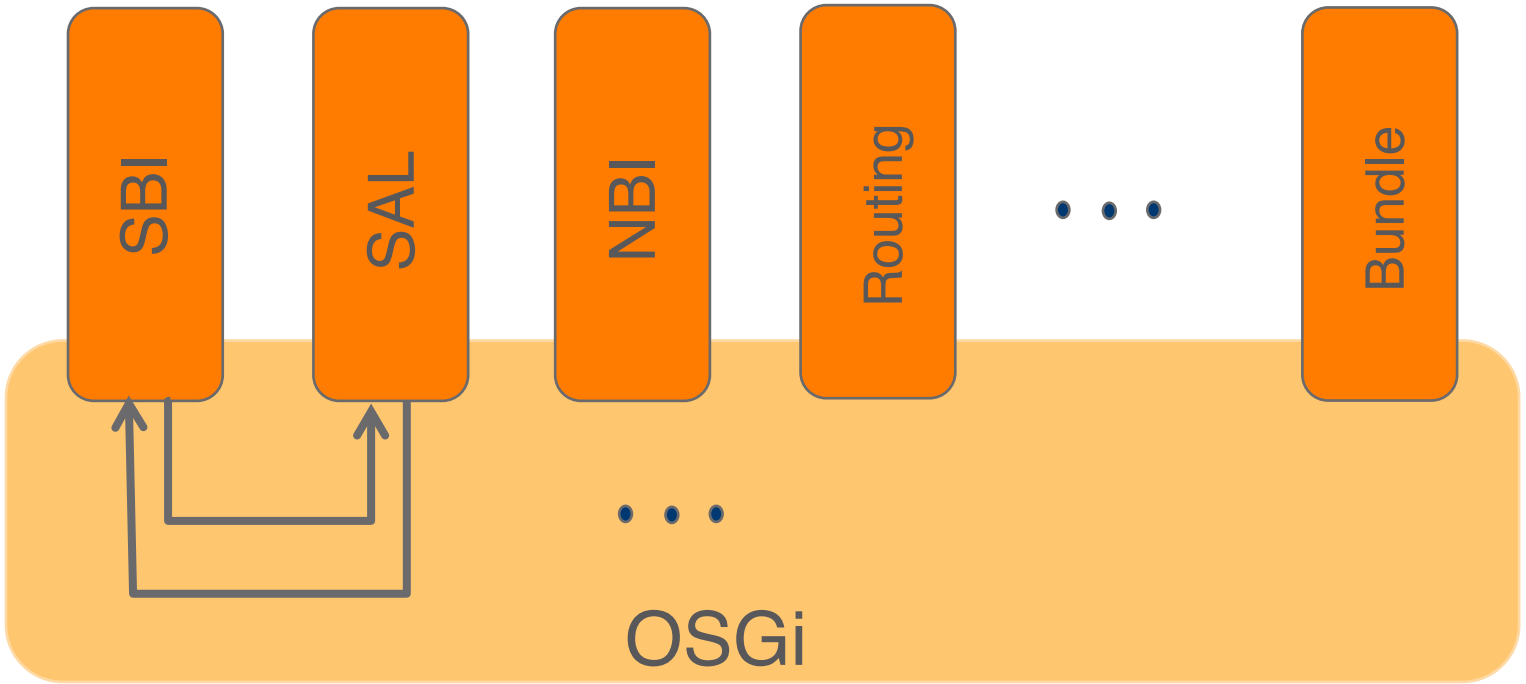
OPENDAYLIGHT SAL

- SAL is the a major architectural component of ODL, differentiating it from conventional SDN controllers

- SAL abstracts services and capabilities of various data-plane elements towards higher functional layers of controller
  Major SAL Services:
  - ***DataPacket:*** dispatching data-packets towards higher layers for processing
  - ***Flowprogrammer:*** allows higher-layer modules to request data-plane programming at the flow level
  - ***Topology:*** allows to collect topology from different SBI plugins and to stich them in higher functional modules
  - ***Inventory:*** provides inventory data to applications, etc.

OPENDAYLIGHT SW ARCHITECTURE

- ODL is mainly programmed in JAVA
- The Open Service Gateway initiative (OSGi) framework is used to support the modularity



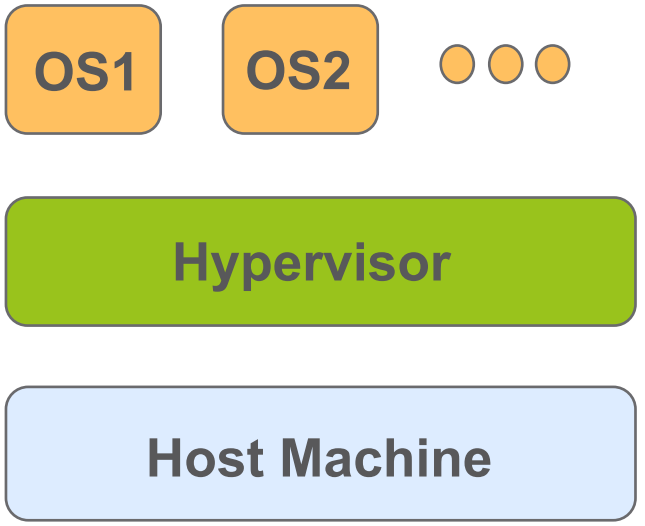
SDN USE-CASE EXAMPLE: NETWORK VIRTUALIZATION

RESOURCE VIRTUALIZATION

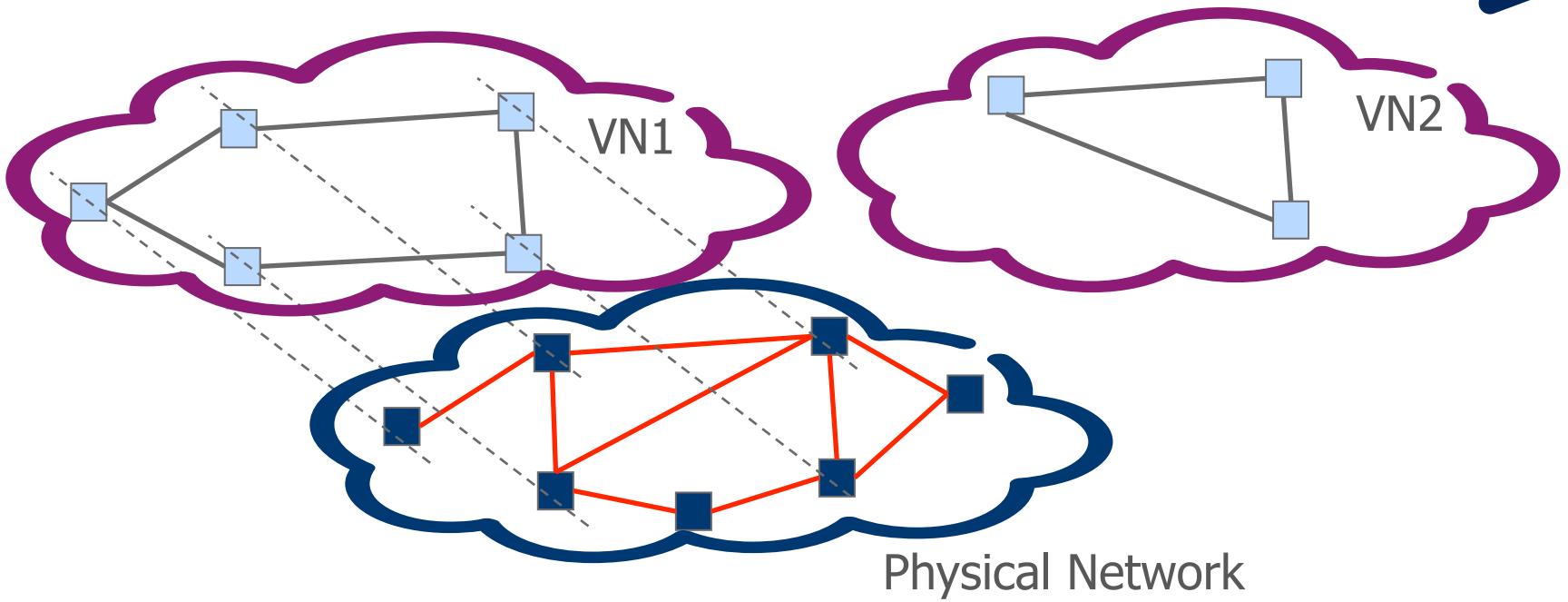
- Virtualization: creation of (multiple) logical instances of a physical entity (element/resource/functionality)
  - logical entities share the resources available in physical one
  - each logical entity can be controlled/managed independently

- Server Virtualization is a common practice in IT industry:

→ A hypervisor allows several OSes to share a single host machine.



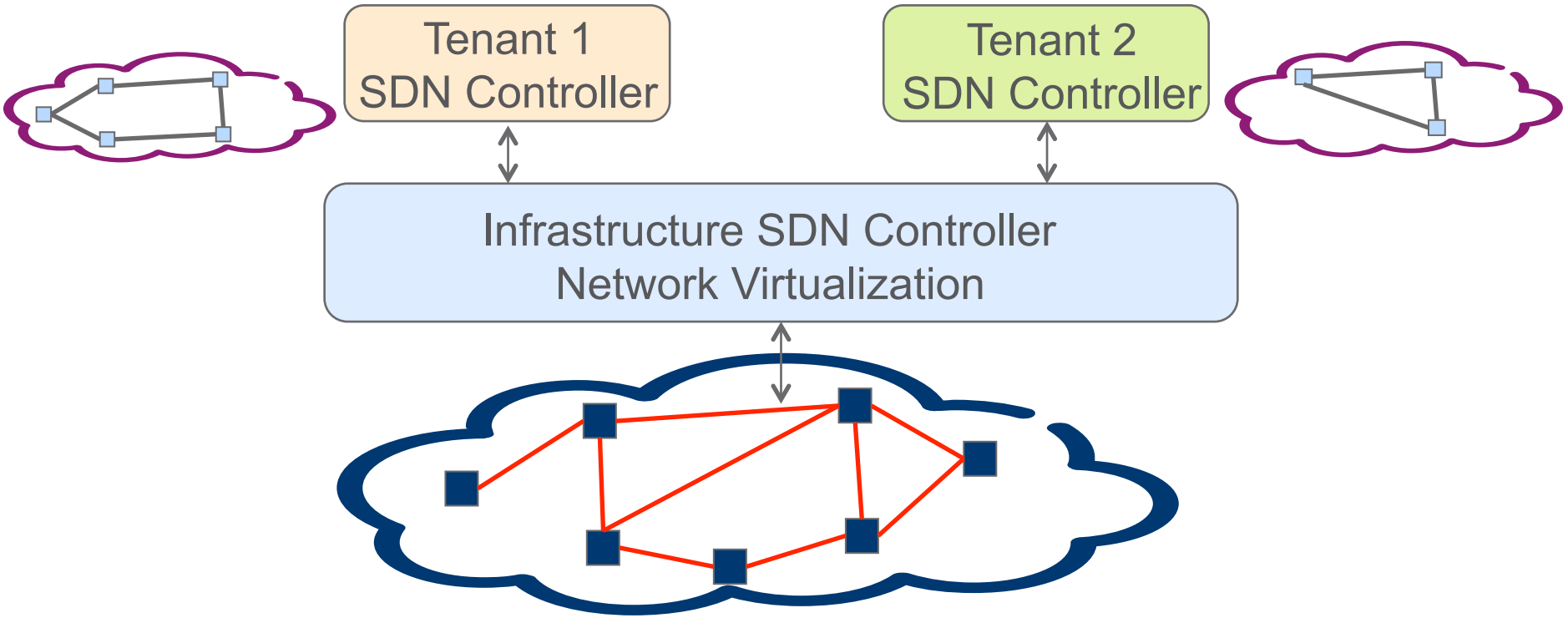
NETWORK VIRTUALIZATION



Resource Isolation between virtual networks & Independent Control

- Network Virtualization
  - Link Virtualization: slice/aggregate link capacity
  - Node Virtualization: slice/aggregate packet forwarding and processing resources within switches/routers
- Several approaches to NV: e.g., network overlay, SDN

NETWORK VIRTUALIZATION WITH SDN



- Infrastructure controller manages the slicing & ensures consistent resource usage between the tenants
- Each tenant network has full control over its (virtual) network, e.g., for routing
- Helpful in datacenter environment together with server virtualization

PROGRAMMABILITY BEYOND CURRENT SDN/OPENFLOW

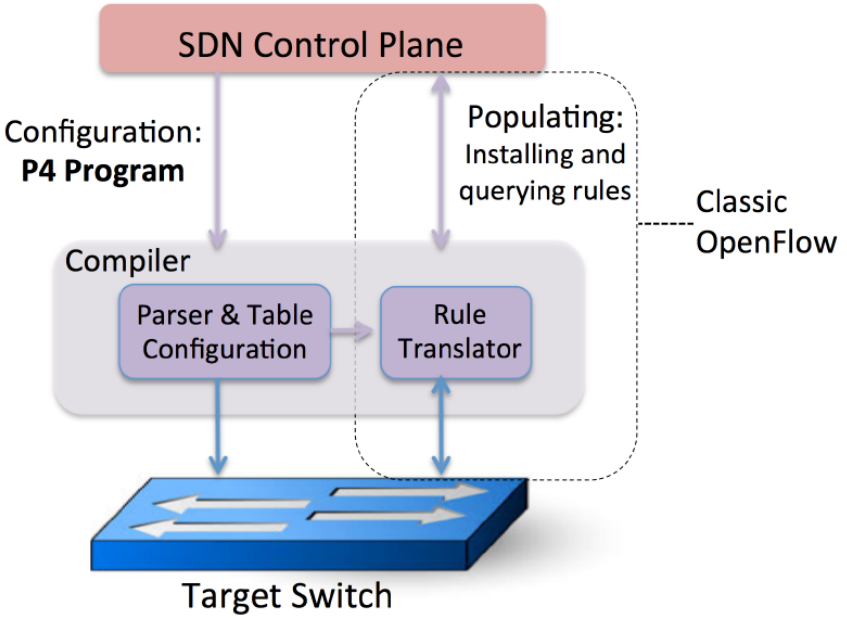
PROGRAMMABILITY BEYOND CURRENT SDN/OPENFLOW

- SDN approaches investigated so far (e.g. OF) merely expose more control knobs for programming the network
- The knobs' functions are dictated by the fixed functionality of forwarding devices
- What if we need to (dynamically) reprogram the functionality of forwarding devices?
  - E.g., to define new protocol stacks and associated packet parsing and processing

PROTOCOL INDEPENDENT PACKET FORWARDING

- Protocol Independent Forwarding adds another dimension to the programmable networks.
  - (Re)configure the forwarding element → specify packet parsing & processing model
  - Control the forwarding operation (traffic dependent)

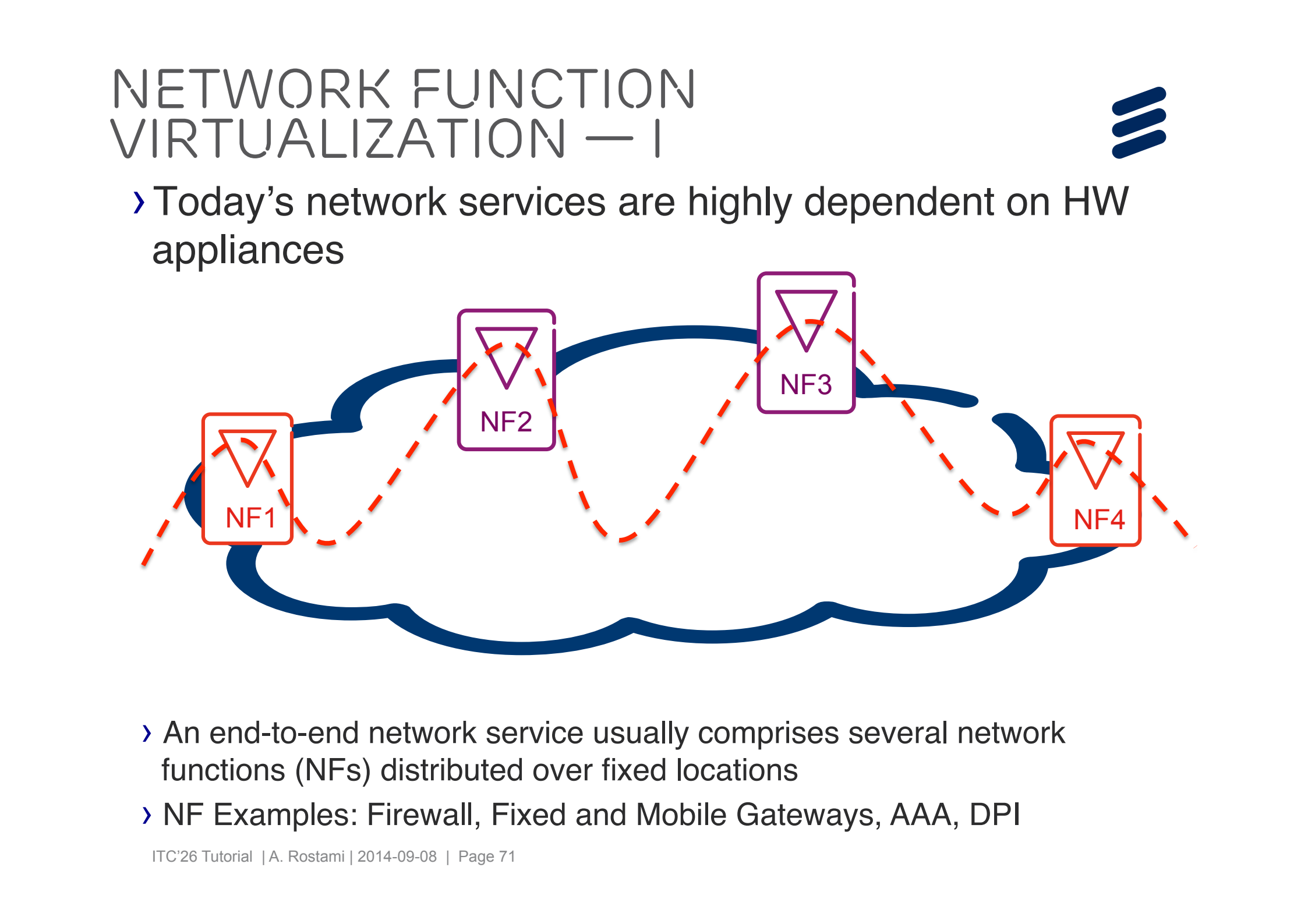
- Application to SDN/OpenFlow
  - Configure the parsing & forwarding models in “match +action” tables
  - Populate the tables and control the traffic forwarding



Source of Fig.: P4 – Programming Protocol-Independent Packet Processors, ACM CCR, July 2014

NETWORK FUNCTION VIRTUALIZATION (NFV)

NETWORK FUNCTION VIRTUALIZATION — I

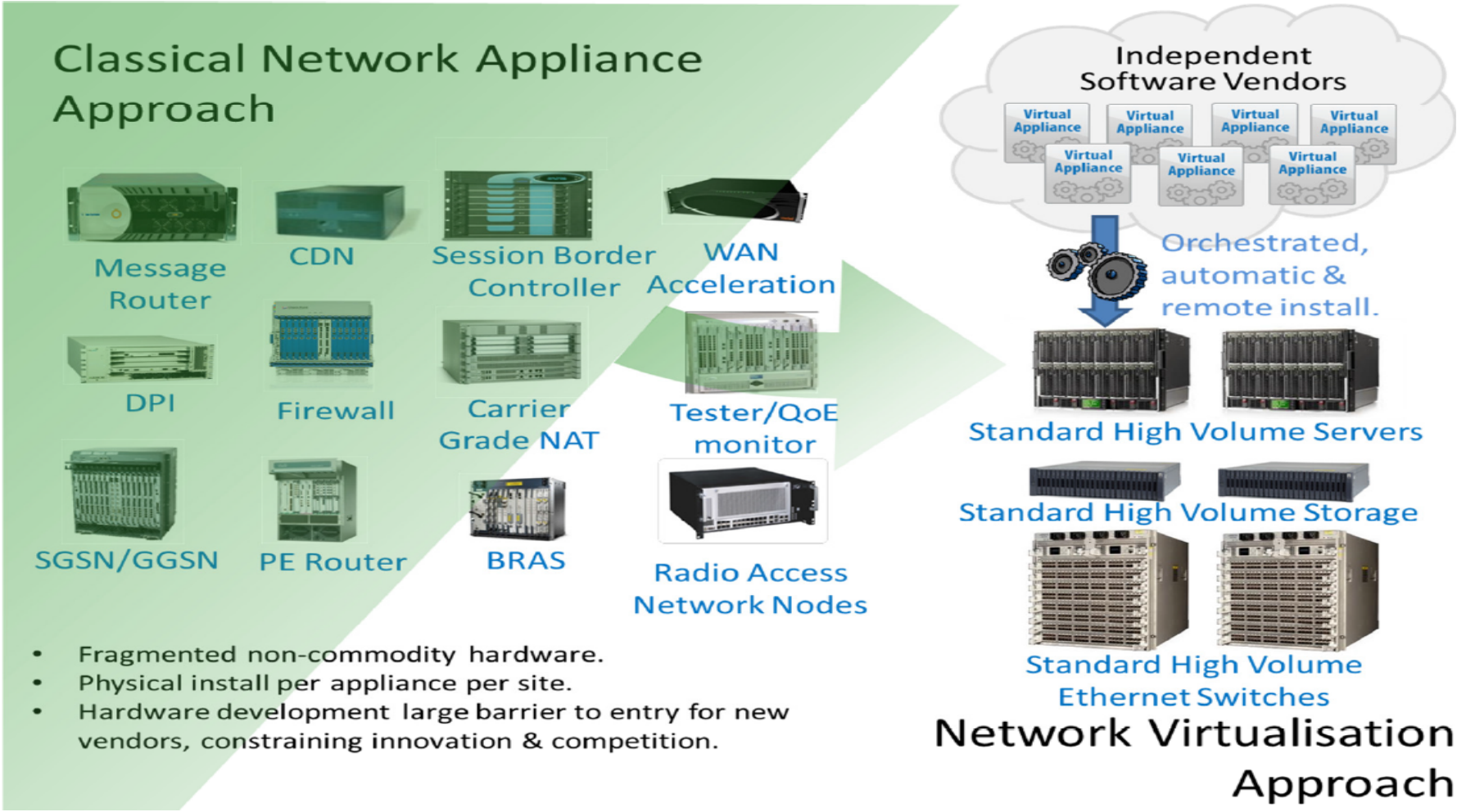


- Today's network services are highly dependent on HW appliances

- An end-to-end network service usually comprises several network functions (NFs) distributed over fixed locations
- NF Examples: Firewall, Fixed and Mobile Gateways, AAA, DPI

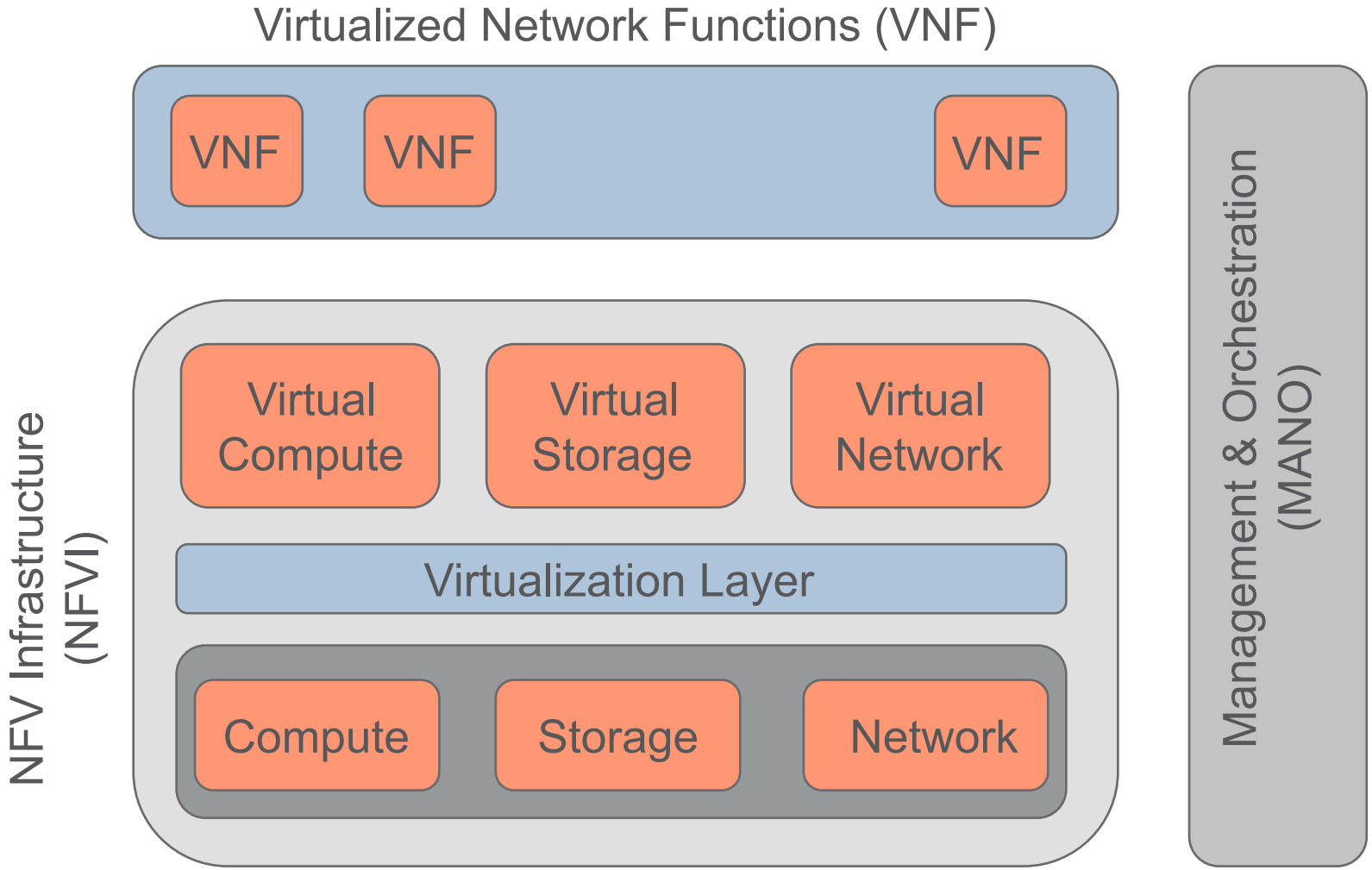
NETWORK FUNCTION VIRTUALIZATION — II

› Virtualizing functions in SW & running them on commodity servers (e.g., in a cloud)



Source: ETSI NFV Architectural Framework, Oct. 2013

NFV ARCHITECTURE (ETSI VIEW)

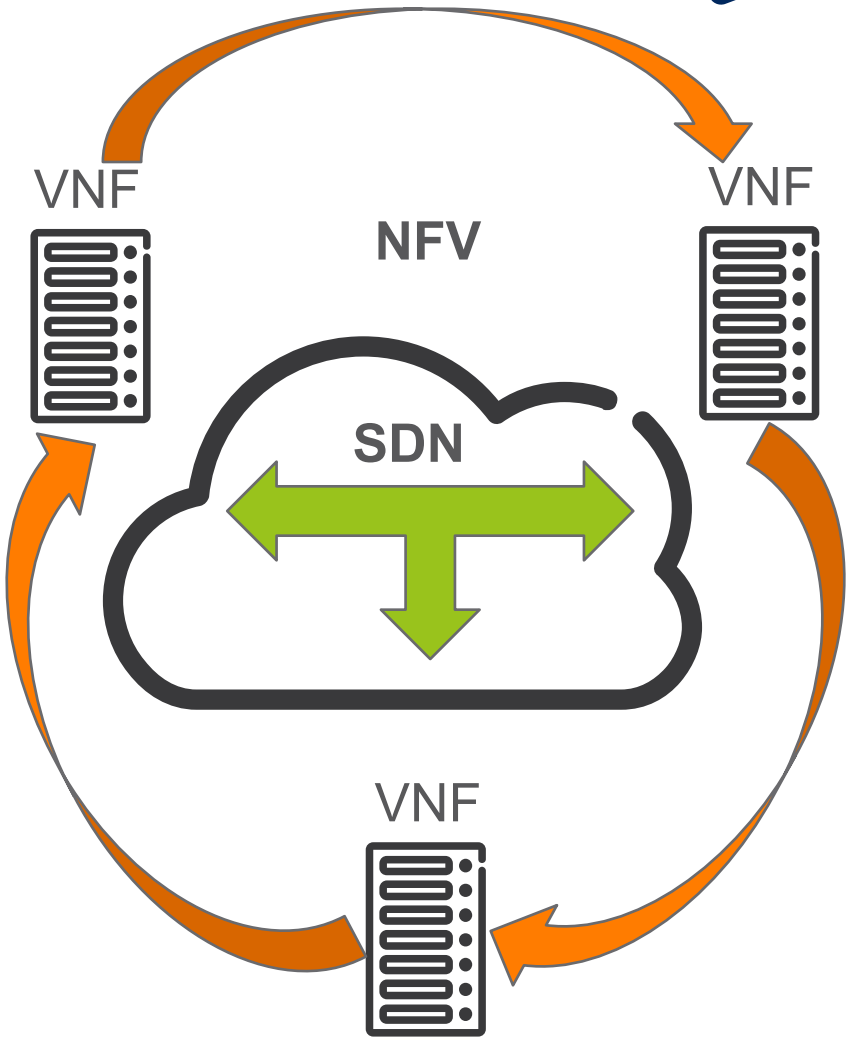


NFV BENEFITS

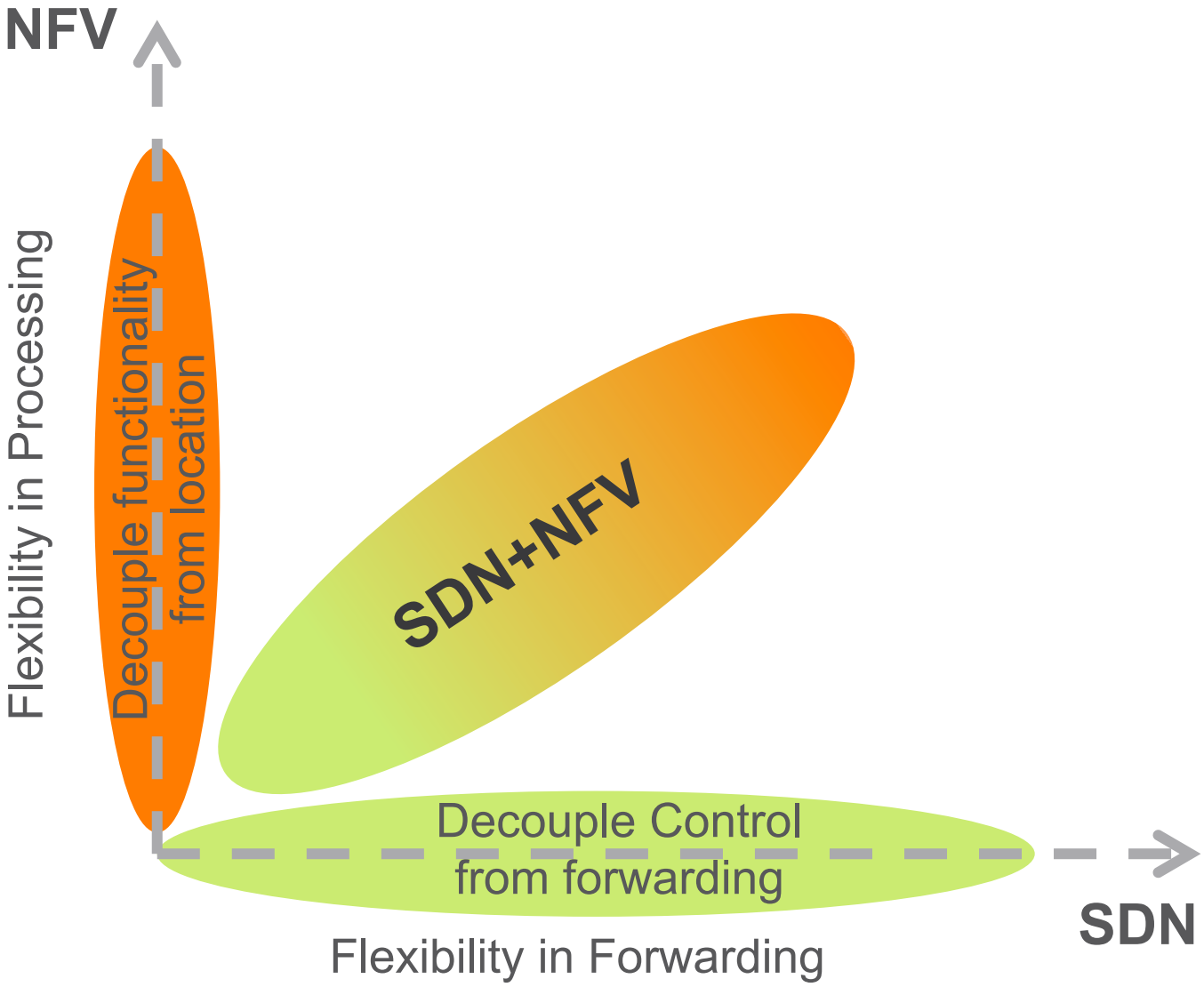
- Efficient resource utilization
- Agile creation of new networking services
- Decoupling the evolution of network functions (services) from underlying platforms
- Reduced Cost (CapEX & OpEX)

SDN VS. NFV

- SDN → flexible forwarding & steering of traffic in a physical or virtual network environment
- NFV → flexible placement of virtualized network functions across the network & cloud
- SDN & NFV are complementary tools for achieving full network programmability



FLEXIBILITY WITH SDN & NFV



SDN & NFV IN ACTION:

UNIFYING THE PROGRAMMABILITY OF CLOUD AND CARRIER INFRASTRUCTURE

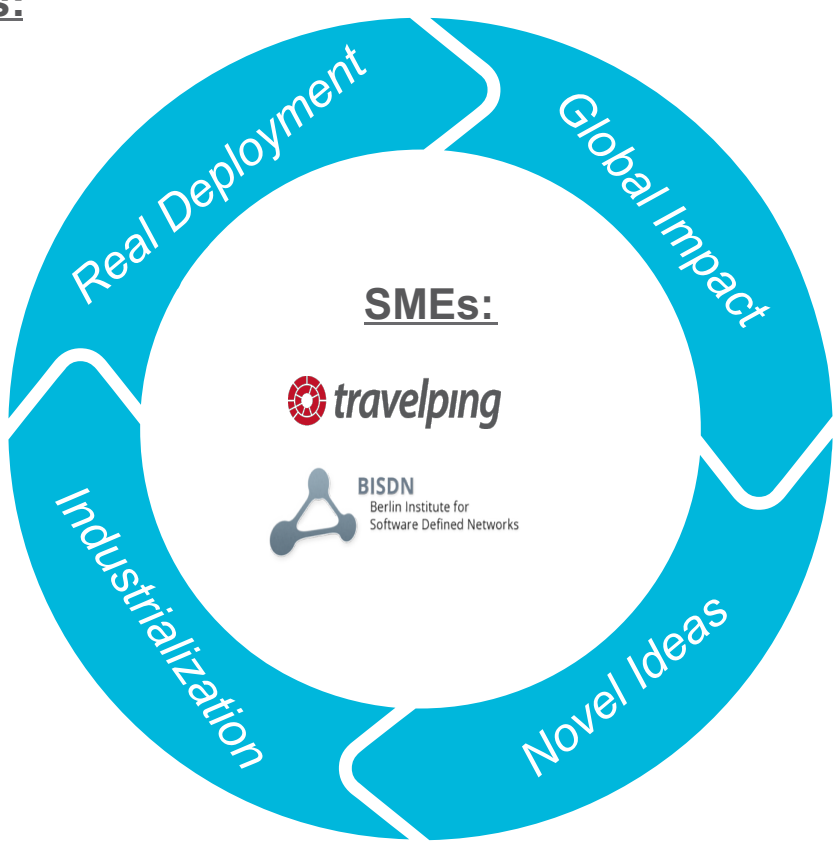
UNIFY CONSORTIUM
2013 NOVEMBER - 2016 APRIL

Major Service Providers:

Deutsche Telekom

TELECOM ITALIA

OTE



Major Vendors:

ERICSSON

intel

Research Institutes:

iMinds
CONNECT.INNOVATE.CREATE

SWEDISH ICT SICS

acreo

Universities:

MŰEGYETEM 1782

Universidad del País Vasco

Euskal Herriko Unibertsitatea

POLITECNICO DI TORINO

berlin

Project Management: eict

Source: http://fp7-unify.eu/

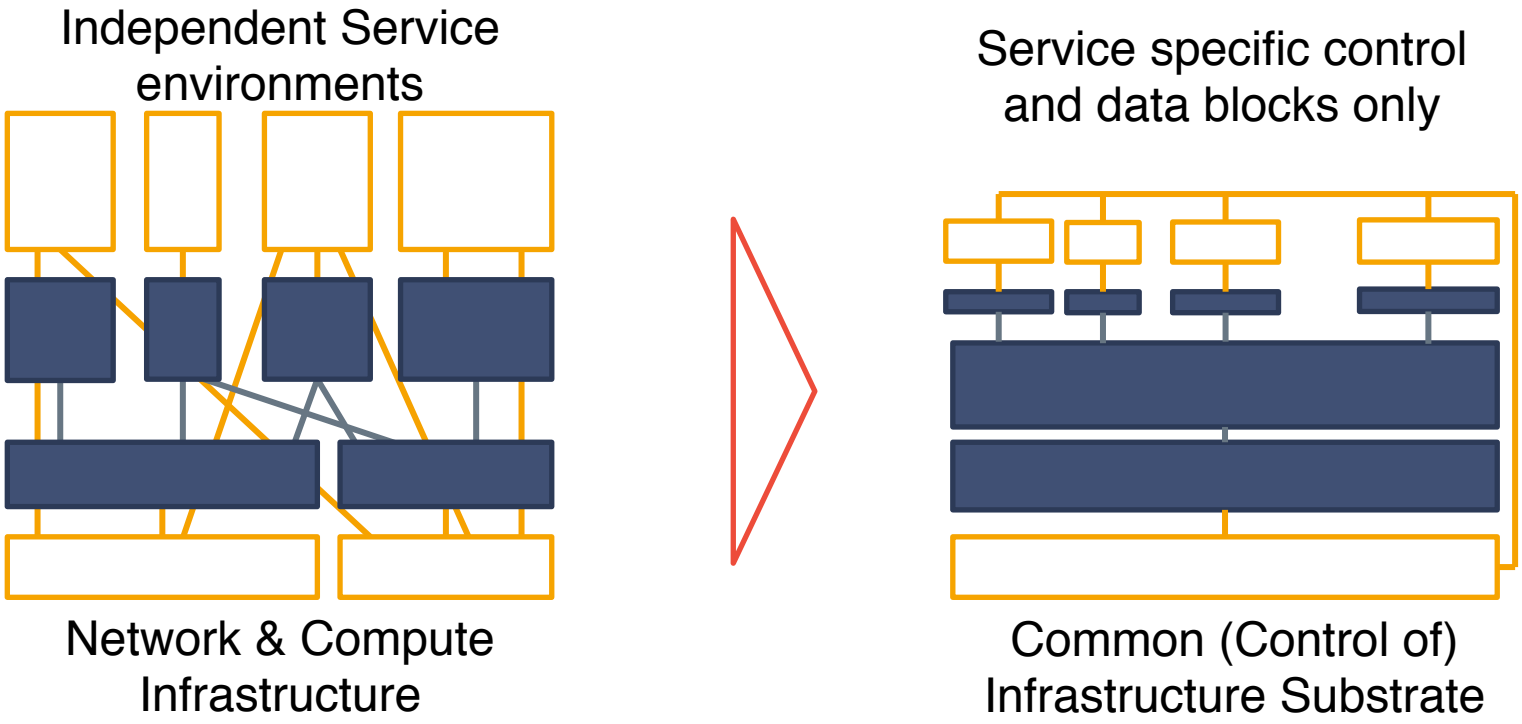
UNIFY REQUIREMENTS

- *Today, rigid network control limits the flexibility of service creation*

- Operators 
  - Agility/Velocity
  - Flexibility/Granularity
  - Simplicity/Automation
  - Scalability
  - Lower OPEX
  - Integration/Shareconomy

- Users 
  - Rich services,
  - Quality of experience,
  - Rapid provisioning of elastic services,
  - On-demand SLA configuration & monitoring,
  - “Follow-me” services

UNIFIED PRODUCTION ENVIRONMENT



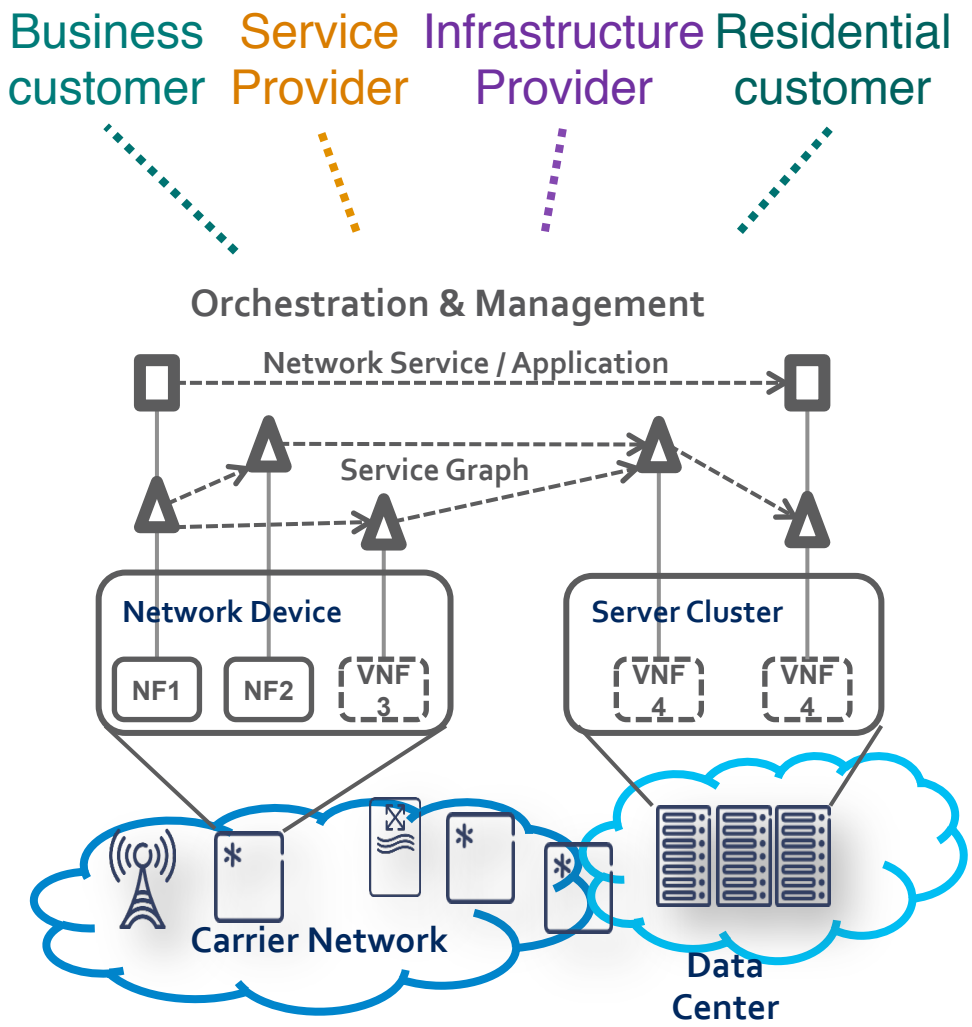
Unified Production Environment with dynamic service creation platform, leveraging a fine-granular service chaining architecture.

UNIFY FOCUS
SEAMLESS INTEGRATION OF CLOUD AND NETWORK

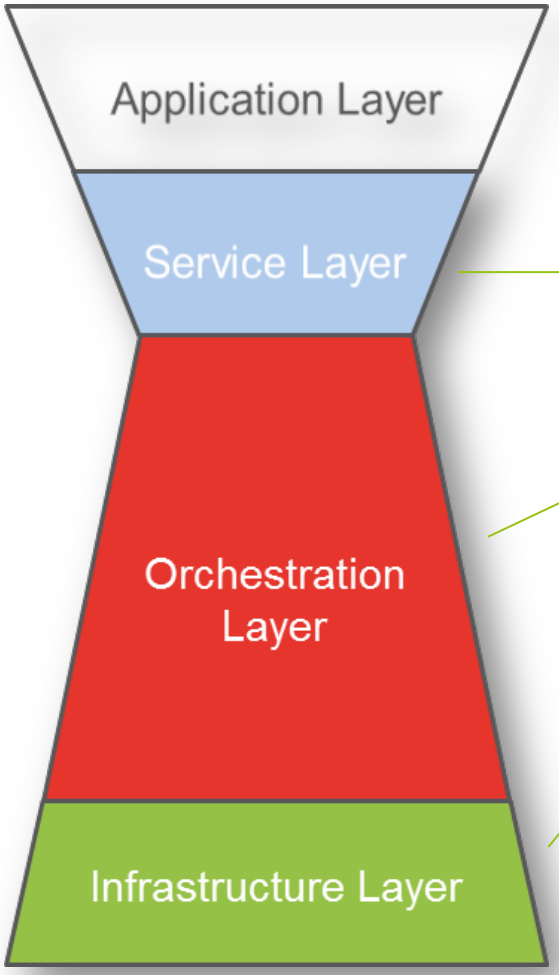
Invocation of
Dynamic Service Chains
(Programmability)
UNIFY Control Plane

Joint orchestration in Network &
Cloud (Abstractions)
UNIFY Control Plane

Data performance
optimized infrastructure
virtualization
(x86 based architecture)
UNIFY Universal Node



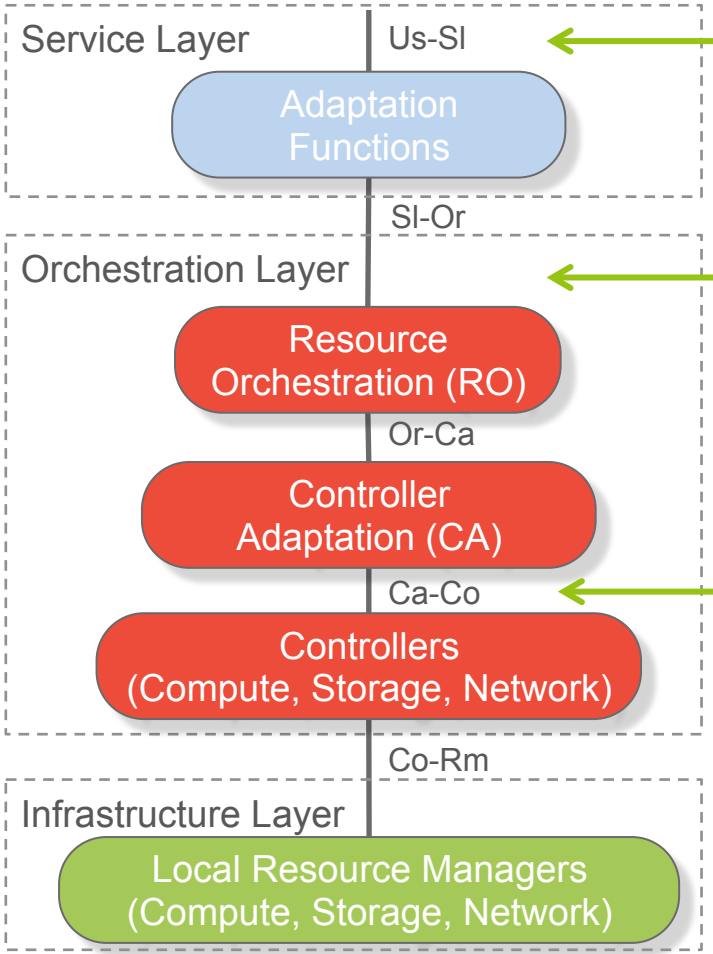
UNIFY LAYERED ARCHITECTURE



UNIFY Control Plane

- Services decomposition and abstraction
- Orchestration for Network Function Forwarding Graphs (NF-FG)
- Combined Compute, Storage & Network abstraction over all resources
  - forwarding elements,
  - compute host capabilities,
  - hardware based network function capabilities,
  - data plane links

UNIFY SLICING THE ELEPHANT – SEPARATION OF PROBLEMS

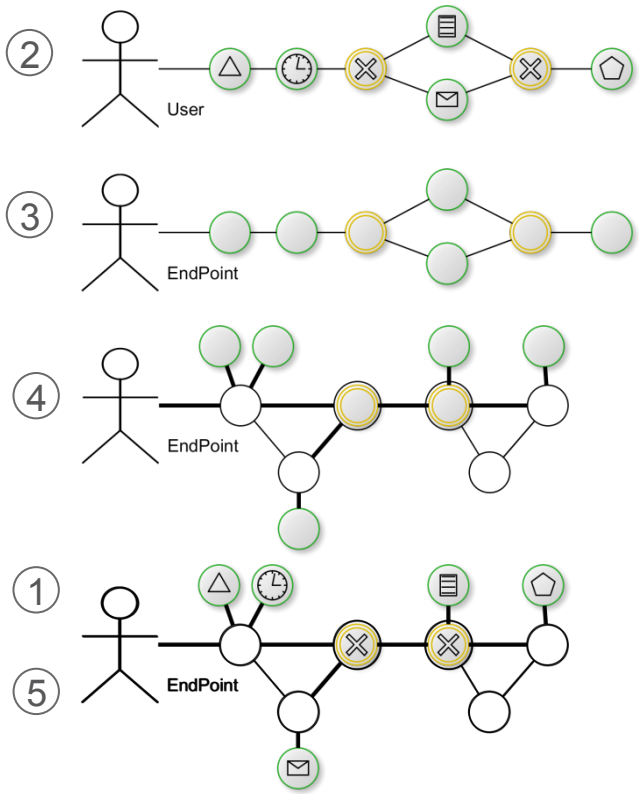


User understandable information
Virtualization (!)
Service Graph

Decomposition into smaller components / network functions
Key Performance Indicators

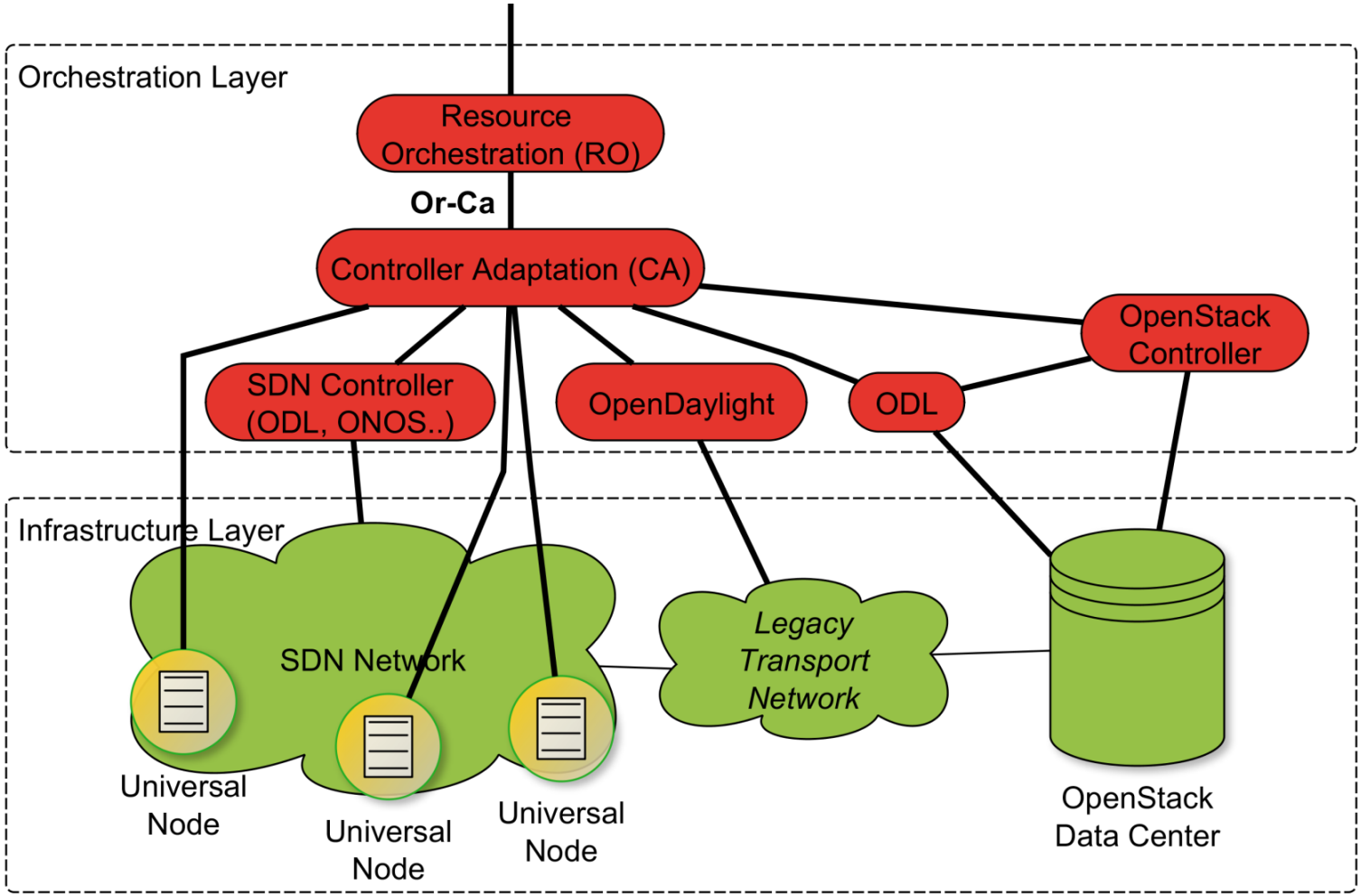
Adaptation to individual controller understandable information for configuration, monitoring, trouble-shooting, etc.

UNIFY ABSTRACTION – AN EXAMPLE

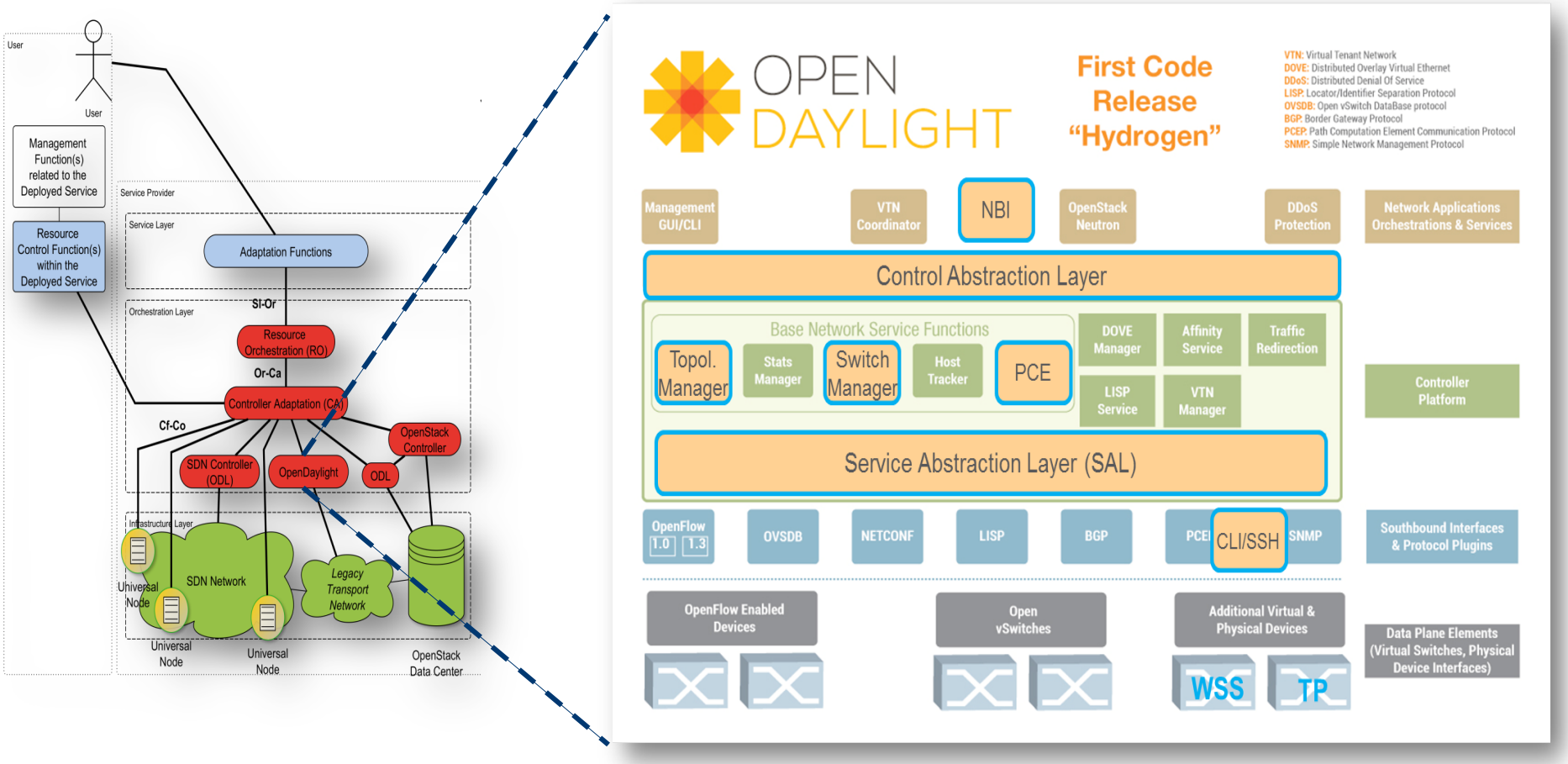


- Service Graph
- Abstract *Network Function Forwarding Graph*
- Abstract Resource Mapping
- Physical Infrastructure
  - Compute, Storage, Network and Topology
  - **Instantiated Service Graph**

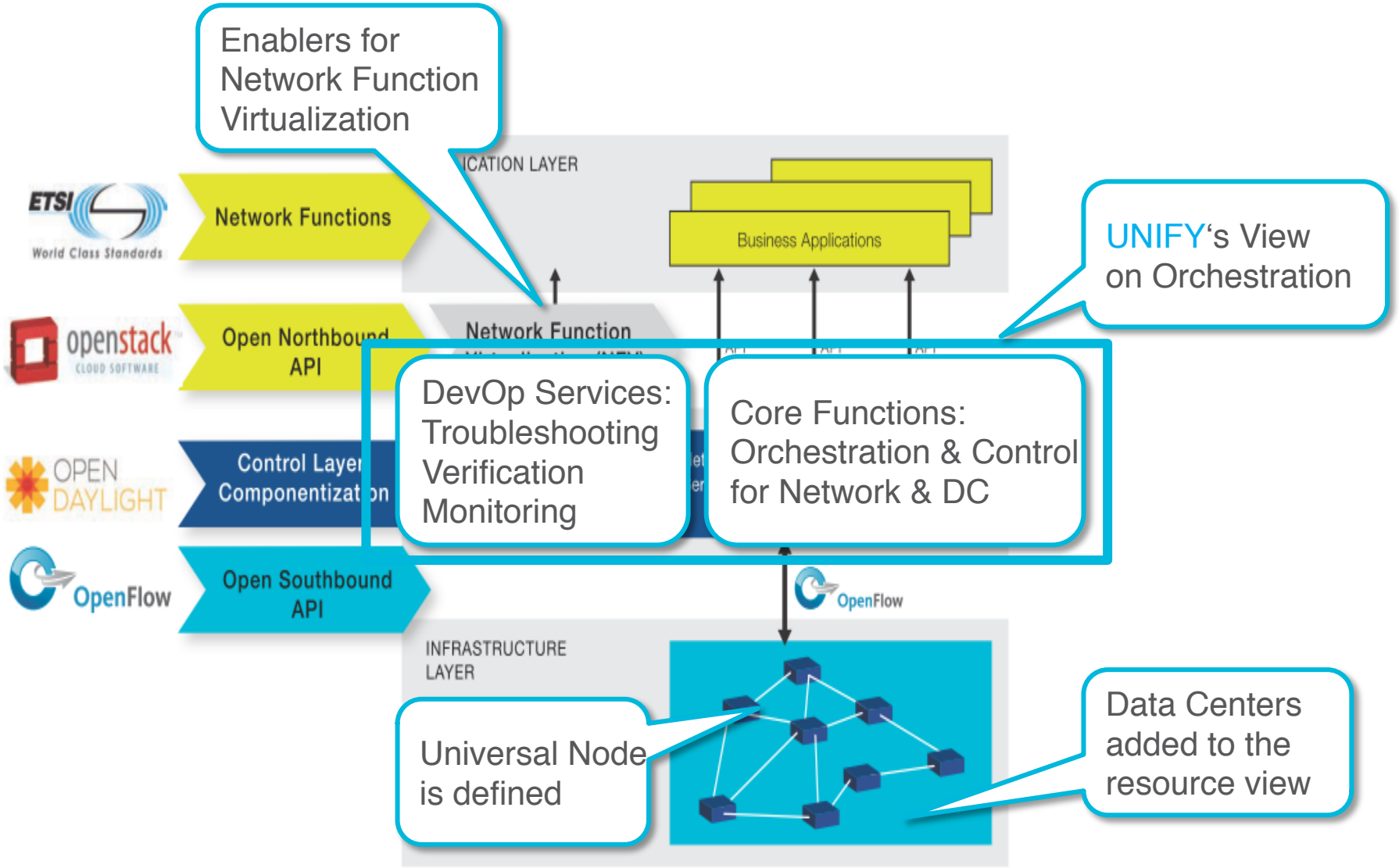
UNIFY EXAMPLE FOR REALIZATION



UNIFY IMPLEMENTATION OPTION – LEGACY OPTICAL TRANSPORT



UNIFY VS. NFV AND SDN INDUSTRY



source: ONF Solution Brief February 17, 2014

LINKS TO RELEVANT BODIES

- Open Networking Foundation (ONF)
  - www.opennetworking.org
- ETSI NFV Initiative
  - http://www.etsi.org/technologies-clusters/technologies/nfv
- OpenDayLight Project
  - http://www.opendaylight.org
- IETF ForCES WG
  - http://tools.ietf.org/wg/forces
- IETF Service Function Chaining WG
  - https://datatracker.ietf.org/wg/sfc/charter
- IRTF SDN Research Group
  - https://irtf.org/sdnrg
- SDN in IEEE
  - http://sdn.ieee.org

ERICSSON