Software Defined Networking

What is SND?

Software-Defined Networking (SDN) is a new approach to network programmability, i.e. the ability to initialize, control, modify and dynamically manage network behavior through open interfaces. [RFC7426]

In short, the key elements of SDN are:

- a level of abstraction between the user plane and the control plane
- open interfaces
  - between user plan and control plan
  - between control plan and applications
- possibility for applications to program network behavior

Virtualization

Virtualization is a very common abstraction.

Virtual = not existing as a physical object, but made to appear as such thanks to the software

Something that can be used as if it were real, but actually shared with others.

- Ease of use
- Sharing and more efficient use of resources

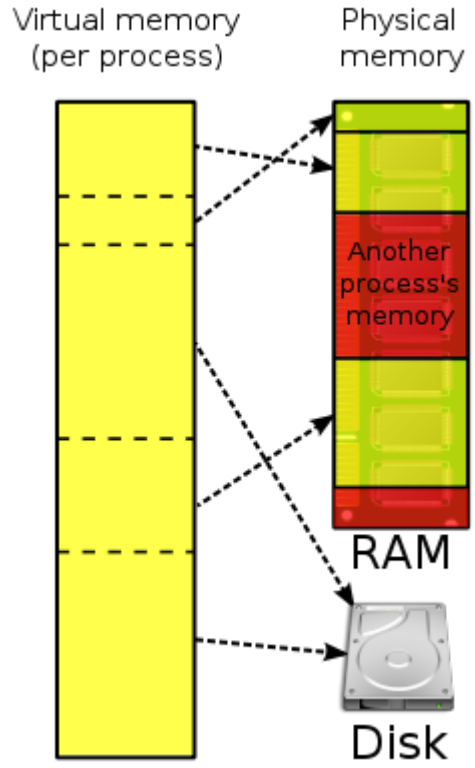
Virtual Memory

Abstraction of physical memory

The programmer sees a single memory dedicated to his process

The software efficiently manages the shared use of physical memory

It hides complexity and simplifies development

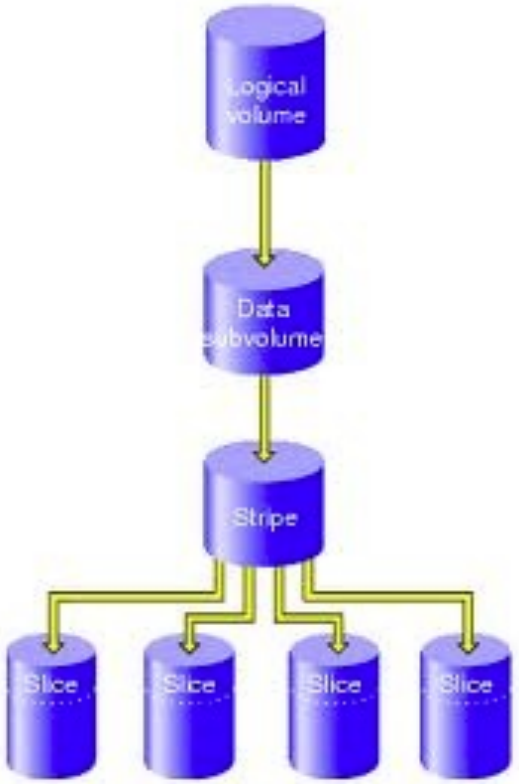


Virtual Disks

Abstraction of physical disks

They permit to dynamically change the size of each partition

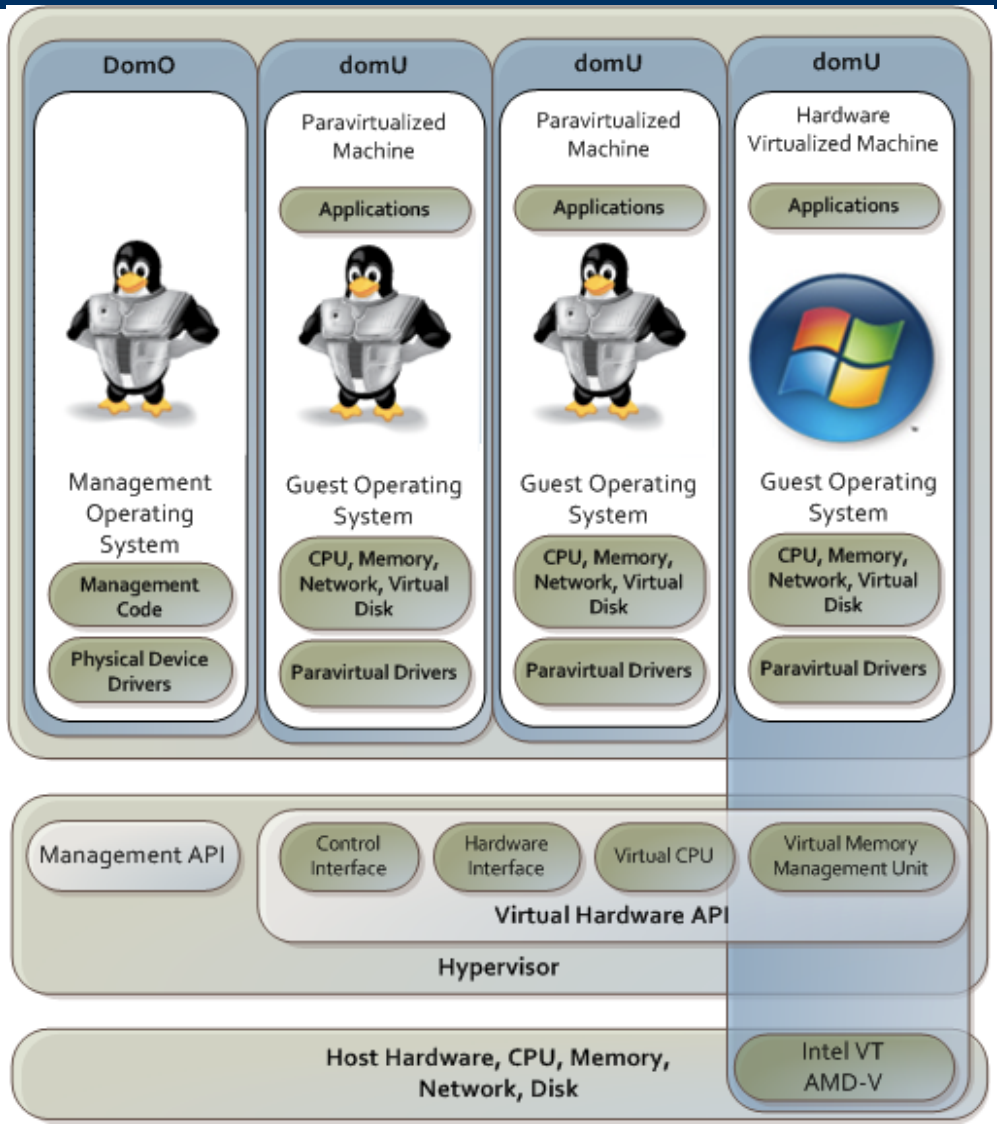
They provide additional redundancy services



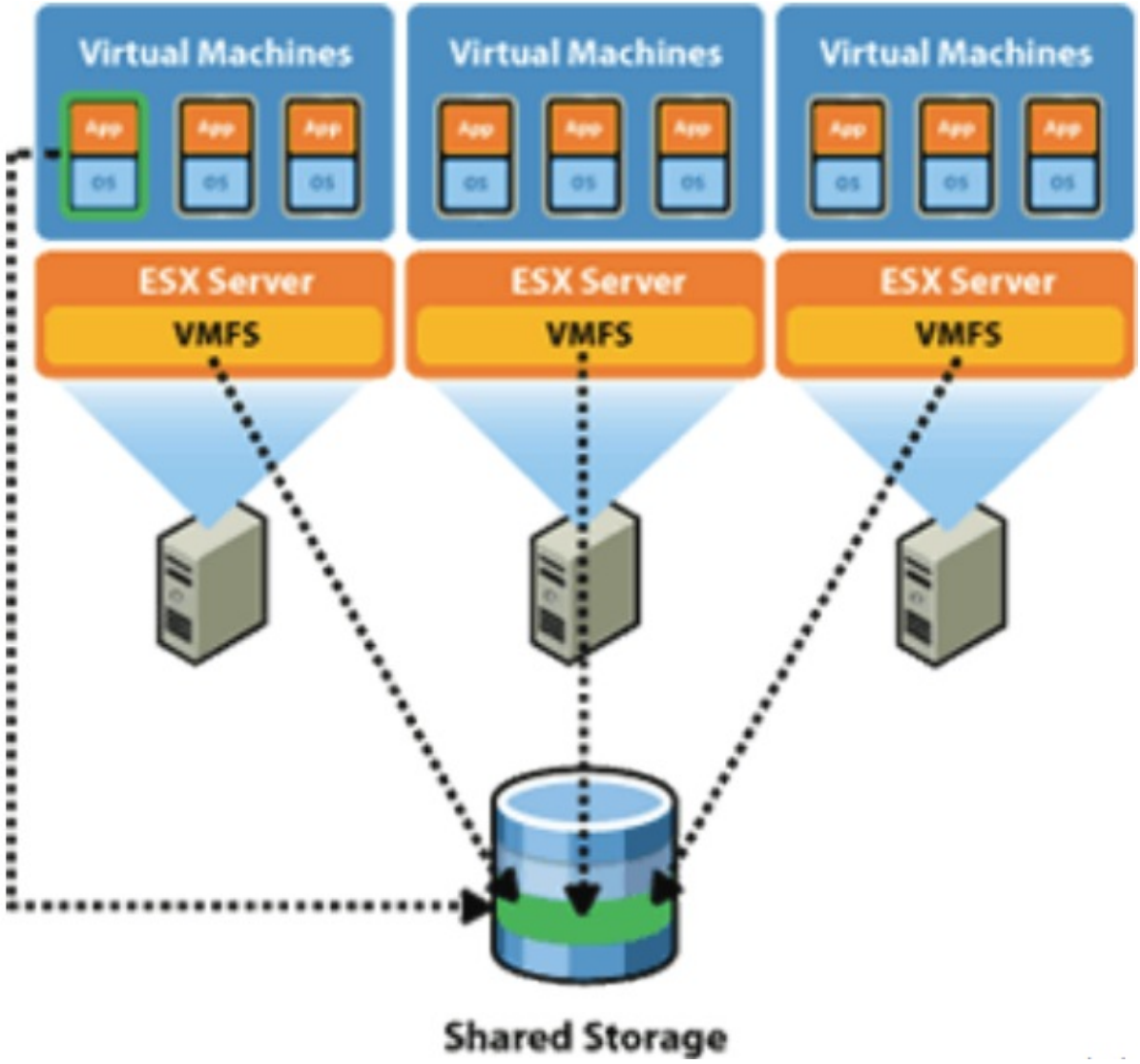
Virtual CPUs

Abstraction of physical machines

They permit to easily instantiate new machines by simplifying administration



Combined Virtualization



Network Virtualization?

Virtual networks are commonly used:

- VLAN (L2)
- Tunnel (L2 / L3)
- VPN (L2 / L3)

Can you have level 2 - 4 abstractions that simplify network management?

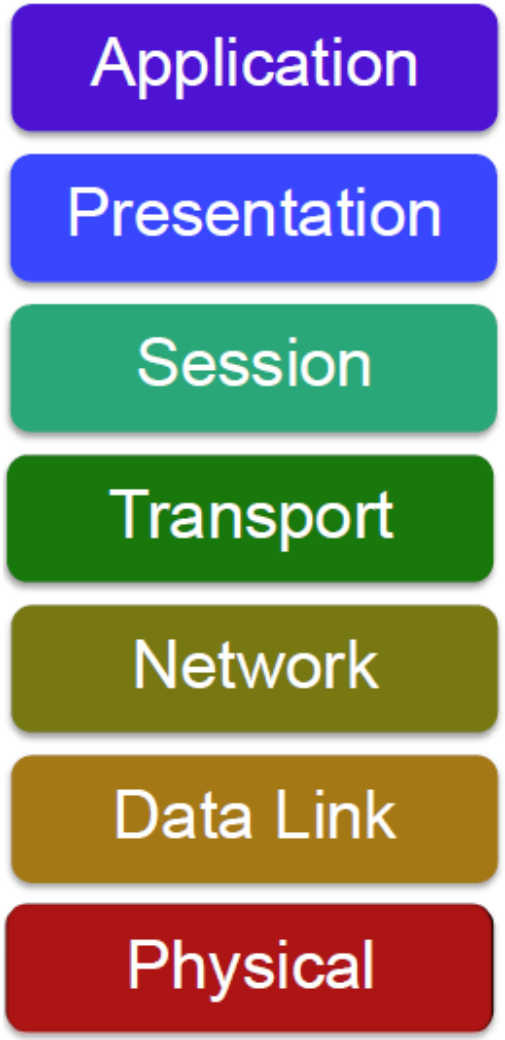
Abstraction in Networks

The OSI model is an abstraction of *the data plane*:

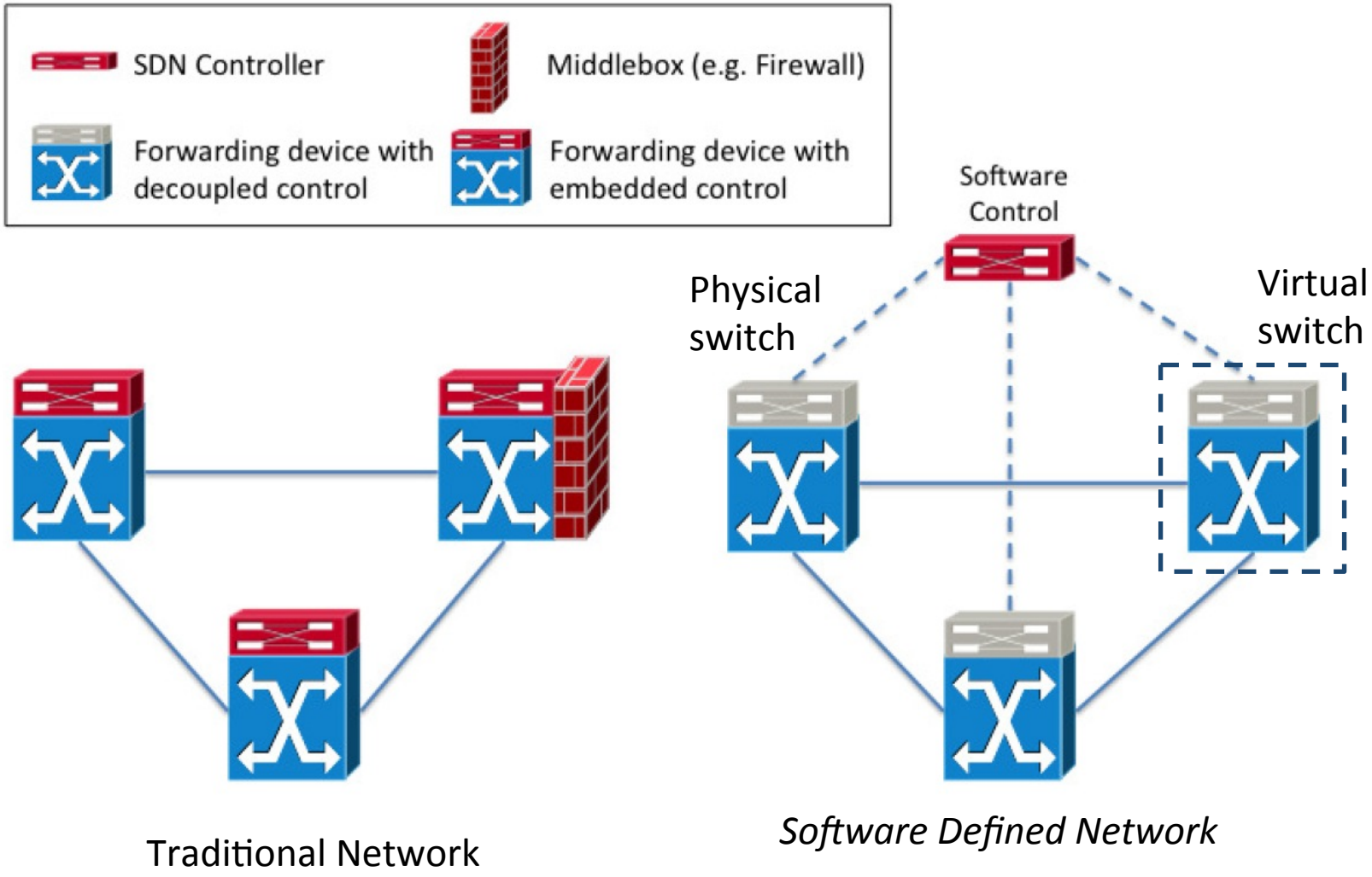
- it simplifies the project and implementation
- it introduces inefficiencies.

Software Defined Networking is an abstraction of *the control plane*

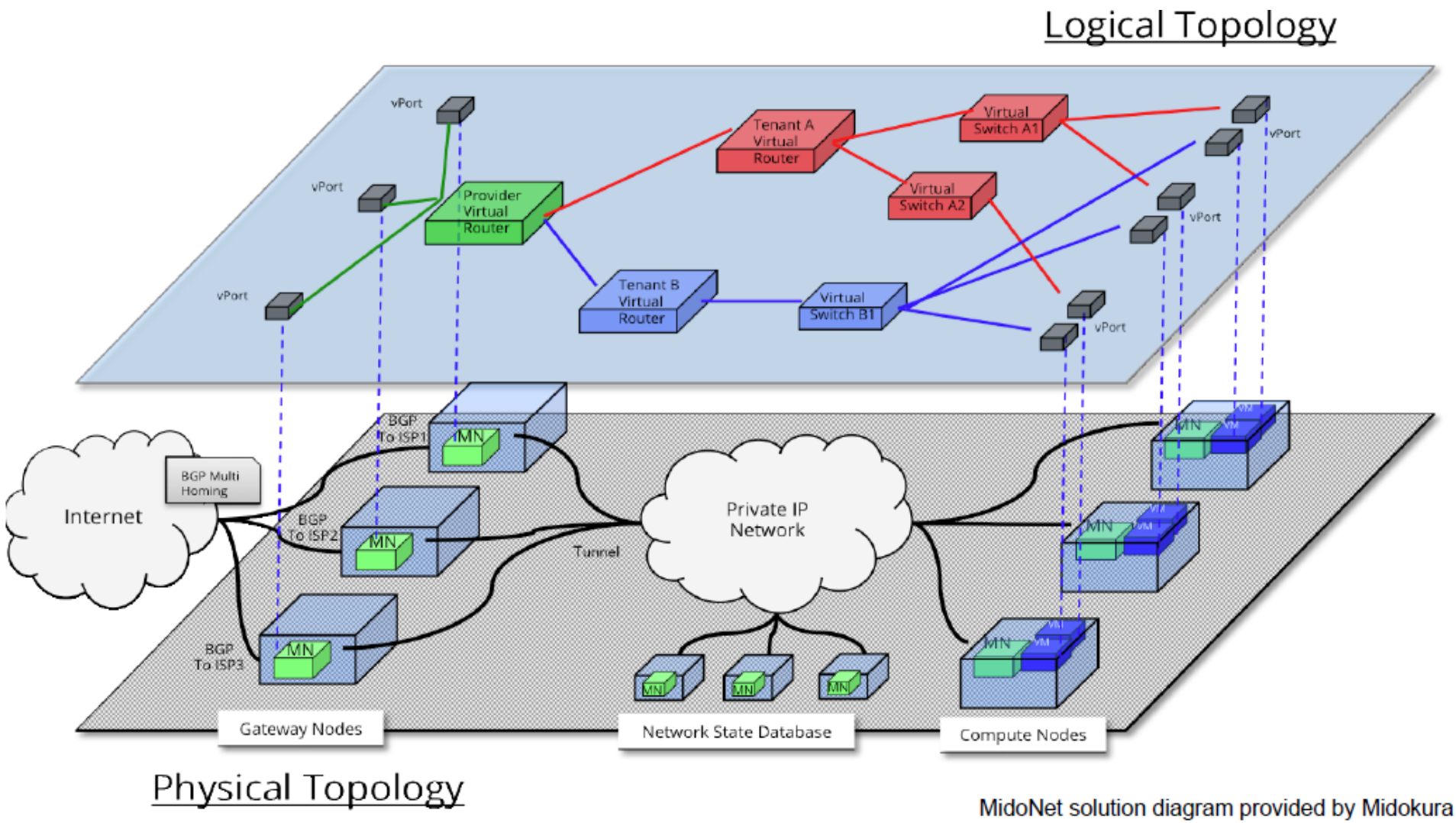
- It permits to instantiate virtual networks to manage specific services
- It integrates with CPU virtualization



Uniform view of physical switches and virtual switches

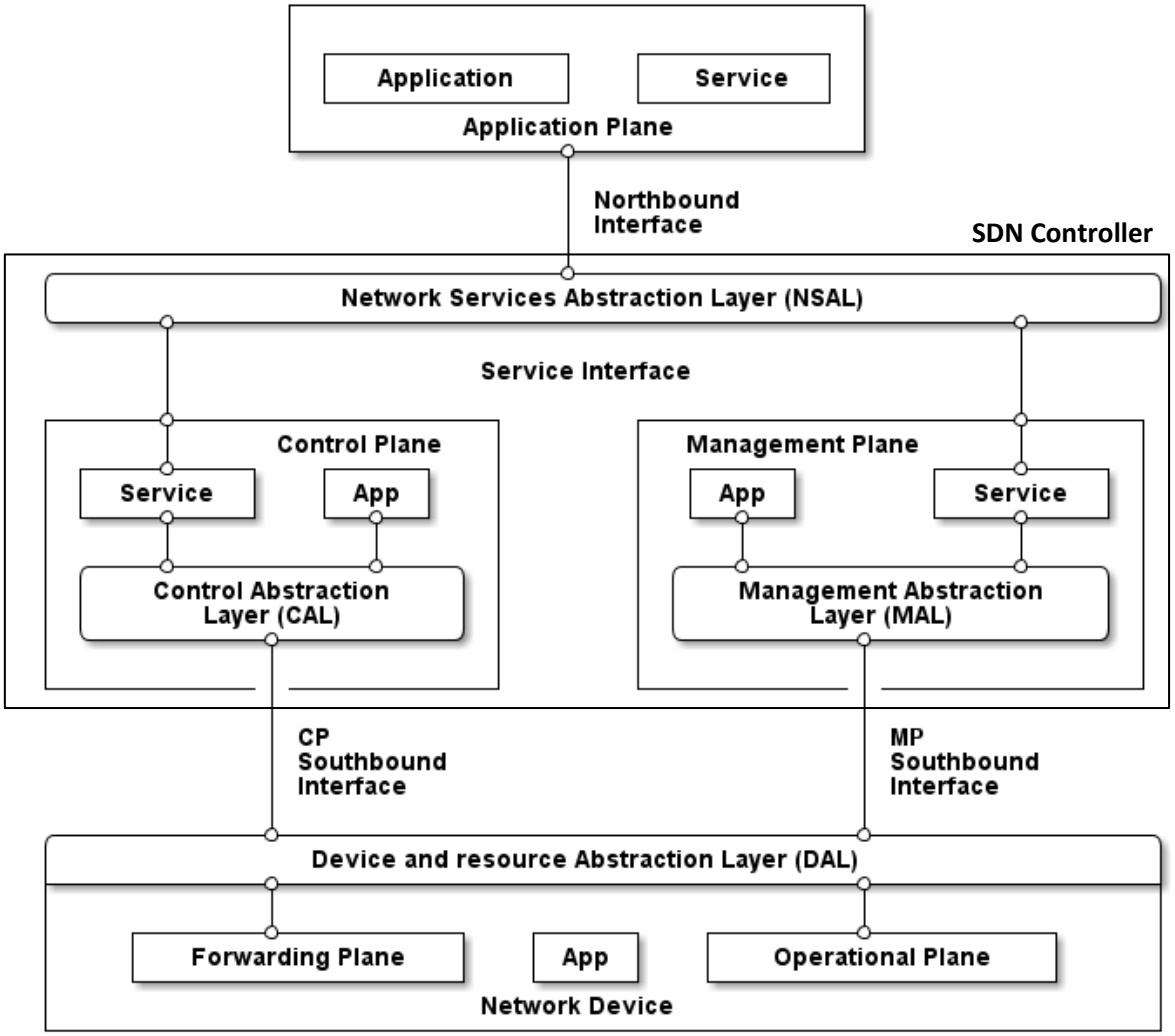


Virtual Network

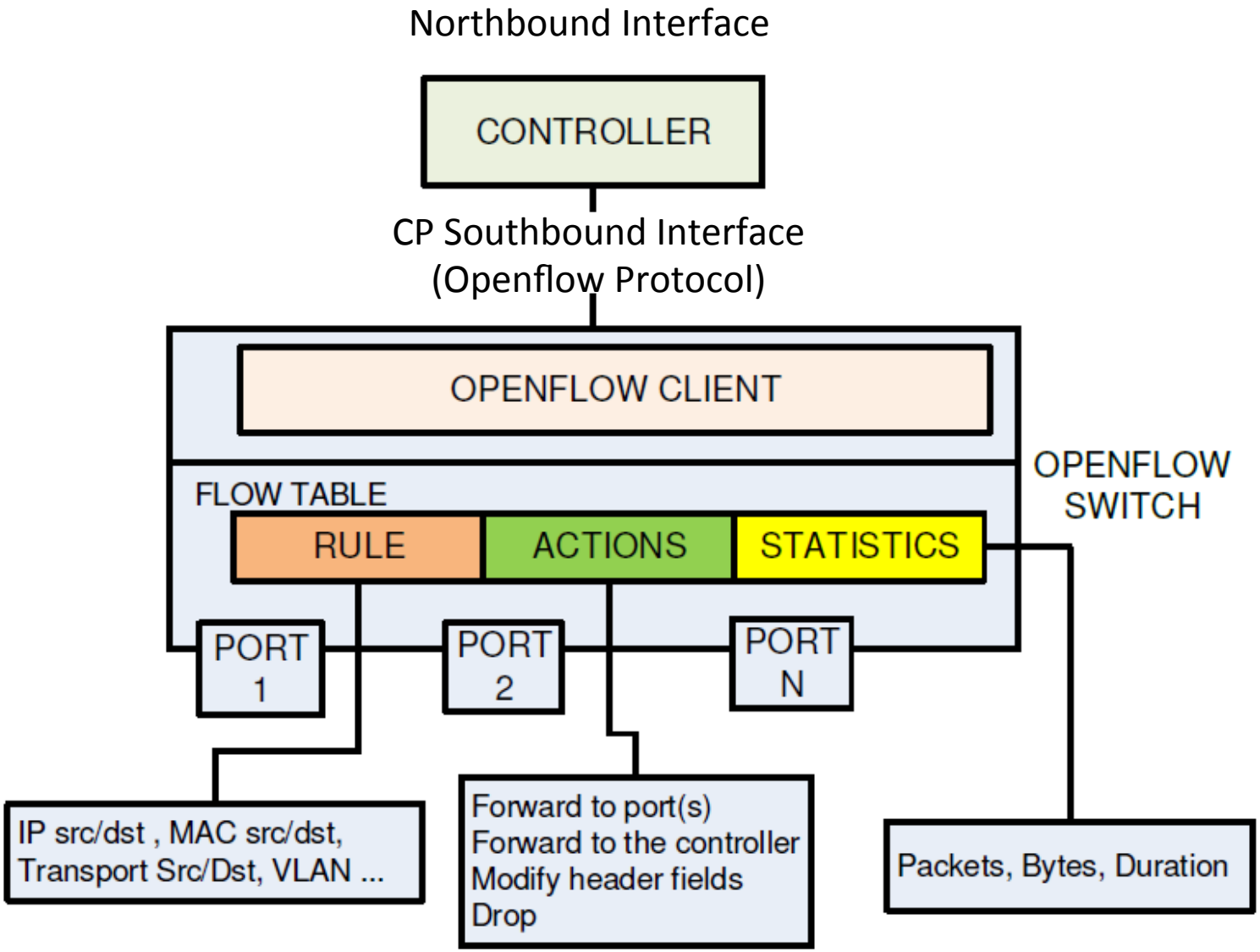


MidoNet solution diagram provided by Midokura

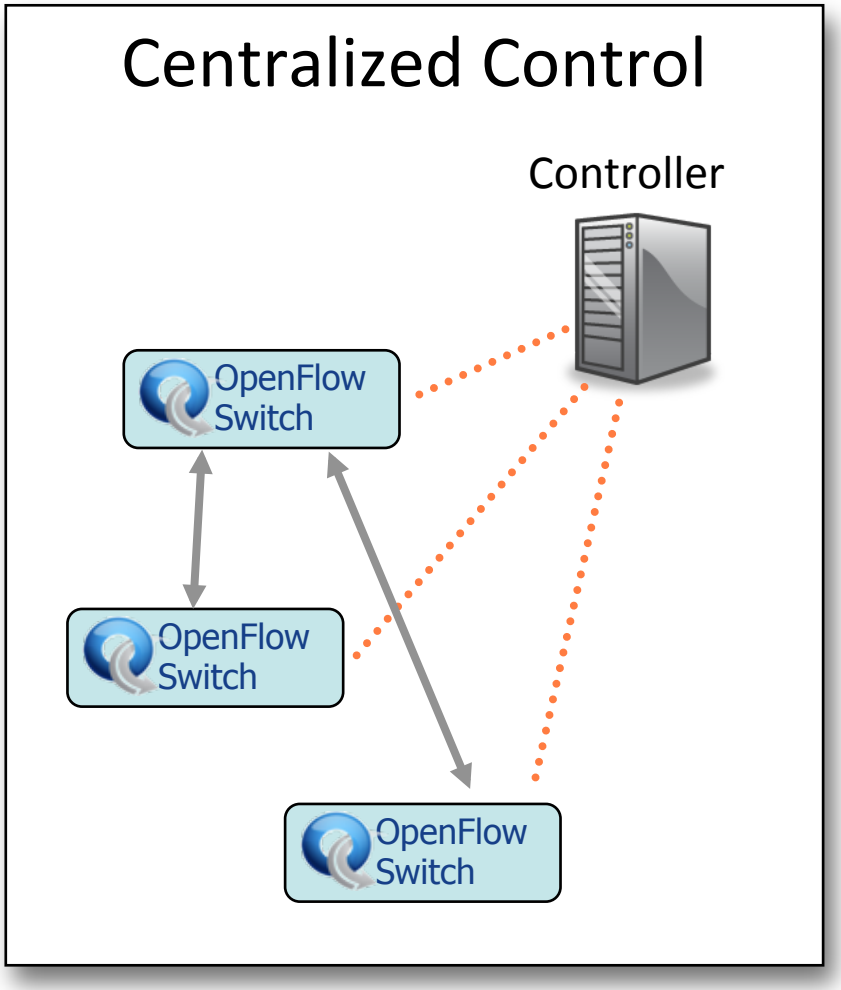
SDN Architecture

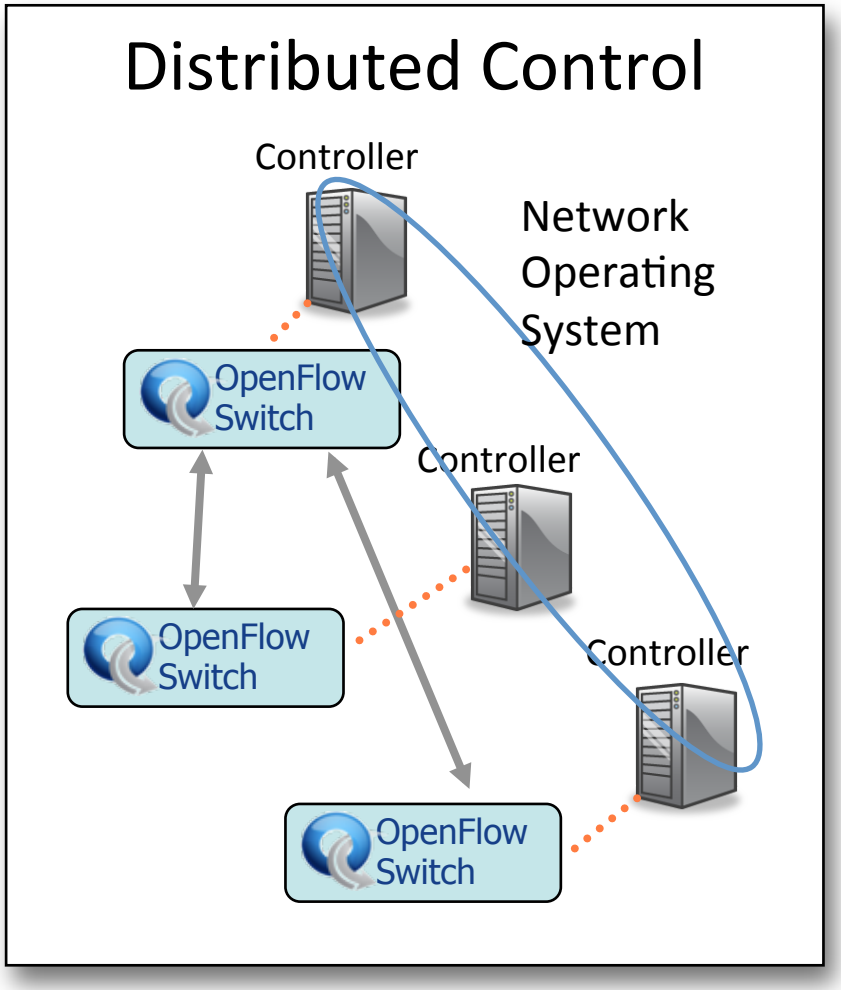


Openflow Architecture



Centralized vs Distributed control models





Control Models
Granularity

Single packet (non-SDN model)

- decisions based on the destination address
- exact match or prefix match

Flow model

- decisions based on multiple fields of the packet
- typically the fields that define a socket pair
- all rules use the same fields
- exact match

Model for aggregation of flows

- decisions based on multiple fields of the packet
- each rule can look at a different set of fields
- multifield packet classification

Control Models
Reactive vs Proactive

Reactive model

- initially empty flow table
- at the first packet of a stream the controller is contacted
- the controller installs a rule for the new flow

Proactive model

- at power on, the switch asks the controller for the rules
- the controller installs all the rules

Openflow-only vs Openflow-hybrid

An Openflow-only switch processes packets exclusively using openflow rules

An Openflow-hybrid switch can process packets according to openflow rules or according to the rules of a normal switch, router or L4 switch

- Packet by packet, based on the physical or logical port, the switch chooses the operating mode
- the openflow part can choose to send the packet to normal processing

Openflow

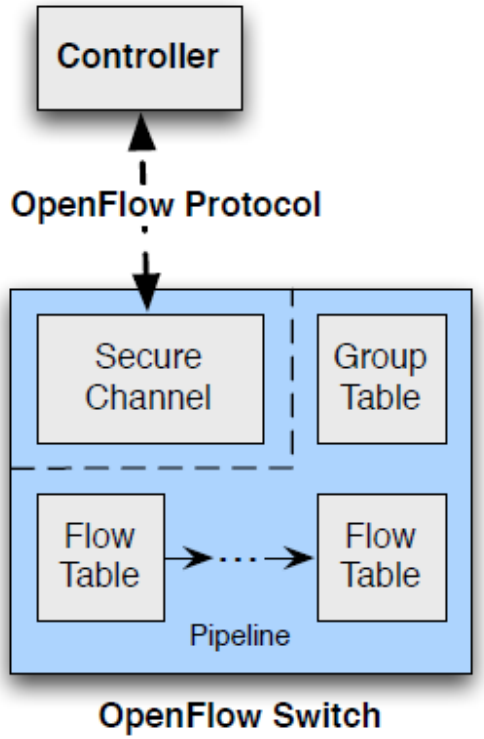
The Openflow standard defines

- an abstract model of switch
- the actions that the controller can perform on the switch
- a southbound protocol

Openflow is **not** an IETF standard

Different versions of the protocol:

- 2008 v. 0.2 (the first)
- April 2012 v. 1.3
- April 2015 v 1.5.1 (most recent)



Openflow Switch
Switch Components

With reference to Openflow 1.3

The switch contains one or more flow tables and a group table

Each flow table contains a set of flow entries

Each flow entry consists of

- match fields
- counters
- set of instructions to be applied to the packets corresponding to the match rule

Using the **openflow protocol** the controller can add, edit or delete flow entries

Matching Verification

Verification of correspondence/matching between packets and rules

- starts with the first table and can continue on the following tables (pipeline processing)
- in each table, verification follows an order of priority
- each table can specify one or more actions and / or send to the next table
- if a subsequent table is not specified, the packet is sent to an interface or to the group table

If no match is found, table-miss entry is executed. If the table-miss entry is not present, the packet discarded.

Openflow Ports

They represent the input and output interfaces from the switch

Types of ports:

physical ports: correspond to the physical interfaces

logical ports: they are abstractions that do not correspond to hardware

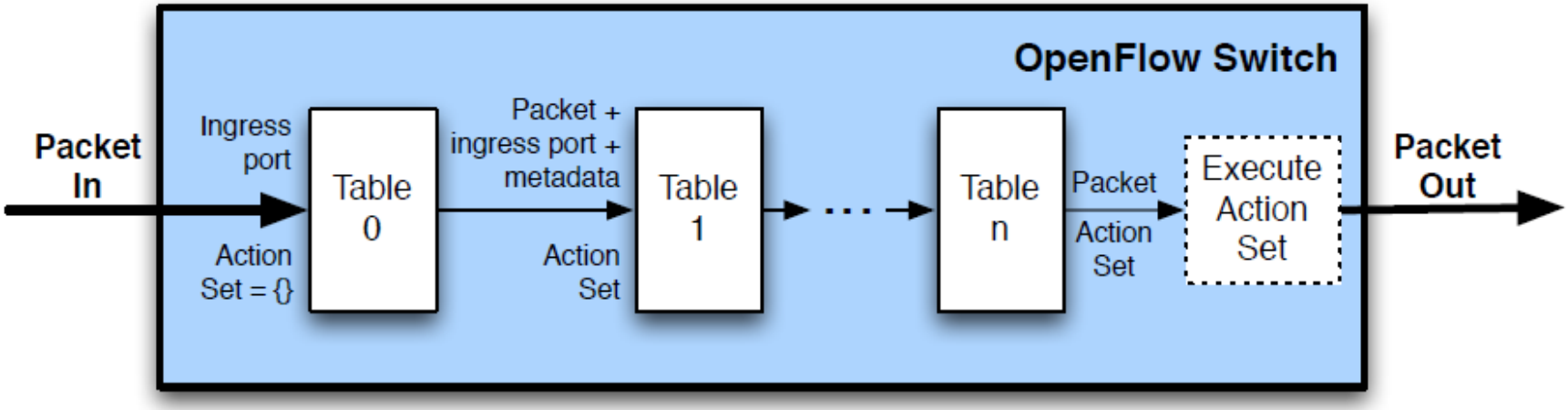
- tunnel
- link aggregations
- loopback

- **reserved ports**: defined by the specification
  - they can be ingress-only or egress-only ports

Reserved Openflow Ports

- ALL
  - only output to all ports except the packet entry port
- CONTROLLER
  - in input, the packet arrived from the controller encapsulated in an openflow control message (packet-out message)
  - as output, the packet is sent to the controller encapsulated in an openflow control message (packet-in message)
- IN_PORT
  - input only, represents the input port of the packet
- other reserved ports: TABLE, ANY, LOCAL, NORMAL, FLOOD

Pipeline openflow



For each table

- finds the corresponding rule with the *highest priority*
- adds the corresponding actions to the action set
- performs any immediate actions
- send to the next table

At the end of the pipeline

- performs the actions in the action set

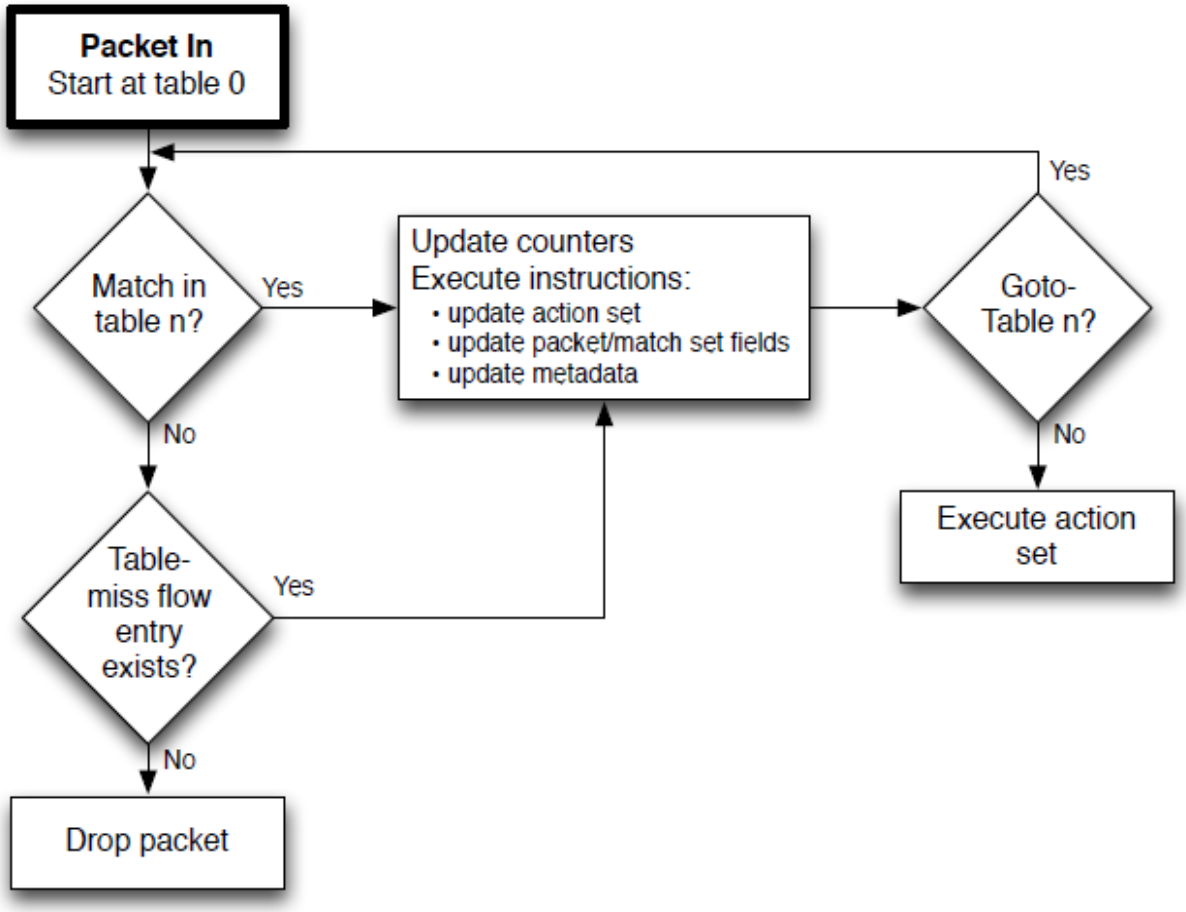
Flow Table

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Each flow entry consists of:

- match fields: ingress port, packet fields, metadata from a previous table
- match priority in the table
- counters updated with each match
- instructions
- timeout: maximum time or maximum idle time before the flow is removed
- cookie: unique identifier chosen by the switch

Matching
General Procedure



Matching

Match search is exact on all fields.

A field can specify a wildcard (ANY). One can also specify a bitmask over some fields

The search returns only the flow entry corresponding to the highest priority

In the case of multiple flow entries with the same priority, the switch returns one of its choice

If there are no matches, the packet is sent to the **table-miss entry**, if any

The table-miss entry has a wildcard on all fields and minimum priority (0)

If the table-miss entry is not present, the packet is *discarded*

Group Table

Group Identifier	Group Type	Counters	Action Buckets

Contains actions that are applied to groups. Allows greater flexibility than flows

- 32 bit identifier
- Group type
  - all: the packet is sent to all action buckets
  - select: the packet is sent to one of the action buckets chosen by the switch
  - indirect: the package is sent to the only action bucket
    - it is used to make many flows point to a single bucket of actions
  - fast failover: the packet is sent to the first bucket associated with a working port

Meter Table

Meter Identifier	Meter Bands	Counters

- 32 bit identifier
- Meter band: list of speeds associated with actions. The packet is sent to the higher value *meter band* lower than the measured flow rate

Band Type	Rate	Counters	Type specific arguments

Band type:

- drop
- dscp remark: decreases the drop probability in the DSCP field of the packet

Counters

Many counters

- per table
- for flow entry
- per door
- ...

Mainly 64 bit

- packet count
- byte count
- duration in seconds (32 bits) and in nanoseconds (32 bits)
- mistakes
- ...

Instructions

Write-Actions <list>: writes the actions in the *action set* to be performed at the end of the pipeline

Meter <id>: send the packet to the specified meter

Apply-Actions <list>: apply actions immediately

Clear-Actions: clear the action set

Write-Metadata <data / mask>: writes the specified bits in the metadata register

Goto-Table <table>: sends the package to the specified table

- new table id must be greater than current table id

Actions

Output <port>: send the package to <port>

Set-Queue <queue-id>: set the queue-id of the package. It is used if the <port> has multiple queues

Drop: the Drop action does not exist. It is the consequence of the absence of instructions

Group <group-id>: send the package to the group <group-id>

Push-Tag / Pop-Tag <tag>: adds the specified <tag>. <tag> can be VLAN, MPLS, PBB

Set-Field <field>, <value>: overwrites a field in the package

Change-TTL: can be set / decrement / copy outwards / copy inwards

OpenFlow Protocol

The OpenFlow protocol is used to exchange messages between switches and controllers. It uses TLS over TCP.

It uses three types of messages:

- controller-to-switch
- asynchronous (switch to controller)
- symmetric (peer-to-peer)

Main *controller-to-switch* messages «Features»

- sent upon activation of the channel between controller and switch
- ask the switch for capabilities
  - datapath ID (~ identifier of the switch)
  - buffer size
  - maximum number of tables
  - …

Main *controller-to-switch* messages «Packet-out»

- used to send individual packets on a specific port
- the payload of the message can be an entire message (layer 2-7) or an identifier (buffer-id) of a packet in the switch
- the message must also contain a list of actions

Main *controller-to-switch* messages «Modify-State»

Used to add, remove, edit flow entries and to set switch properties

Main *asynchronous* messages «Packet-in»

- Used to send packets from the switch to the controller
- the packet (or part of it) is the payload of the message
- sent when the action on the packet is output to the CONTROLLER reserved port
- The switch can bufferize the packet and send in the packet-in only the first bytes of the packet and a buffer-id

Main *asynchronous* messages «Flow-Removed»

Informs the controller of the *removal* of a *flow entry* following an explicit request or due to timeout

Main *asynchronous* messages «Port-status»

Informs about the change of state of a door, for example following a failure