

# Software Defined Networking

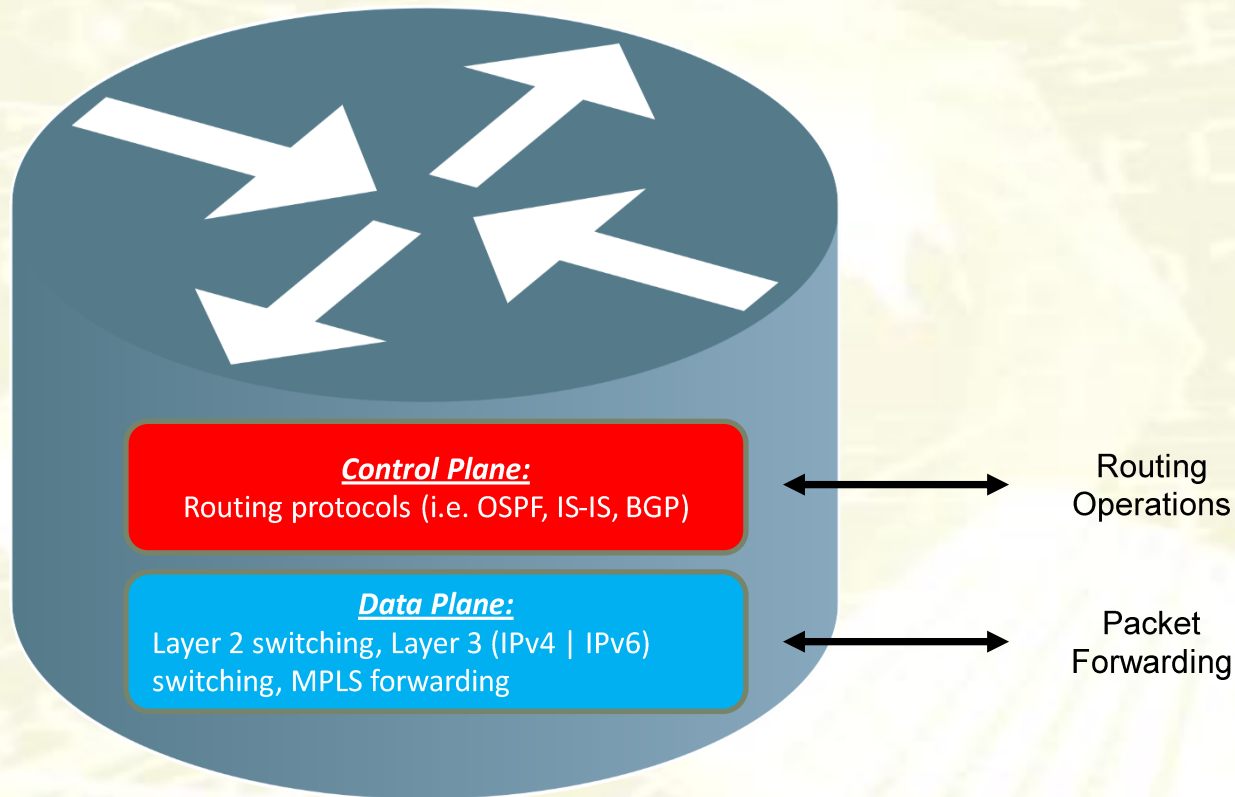
---

Carlo Vallati

Assistant Professor@ University of Pisa

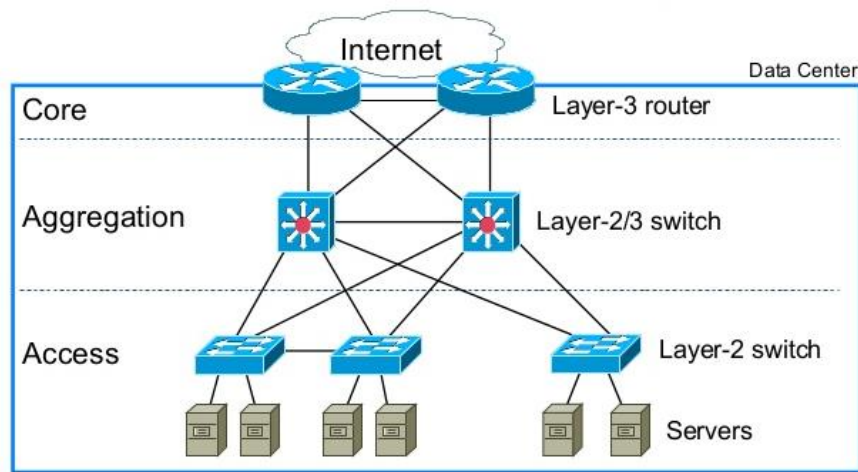
[c.vallati@iet.unipi.it](mailto:c.vallati@iet.unipi.it)

# Traditional Networks



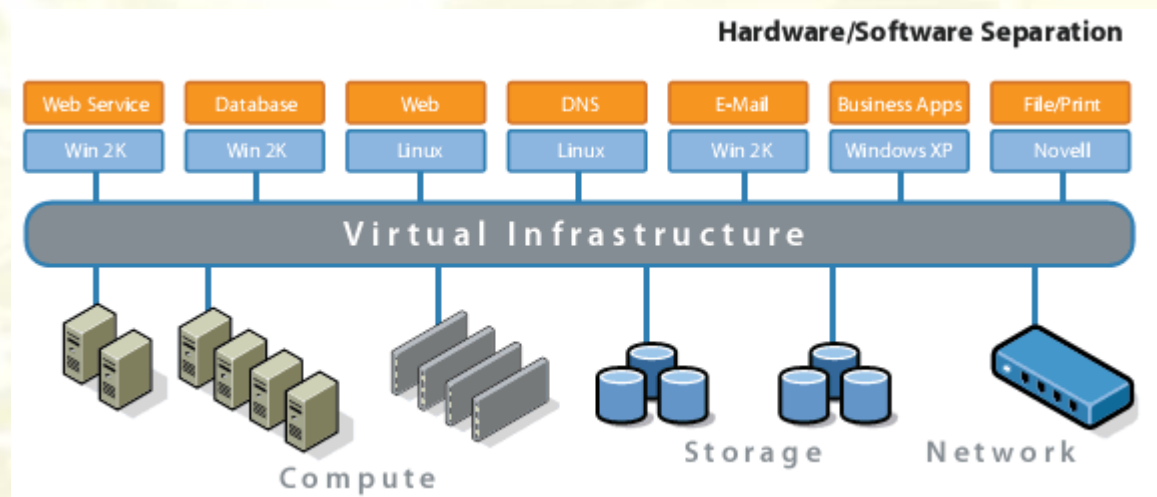
All Operations are implemented on the same device

# Use Case: Datacenters

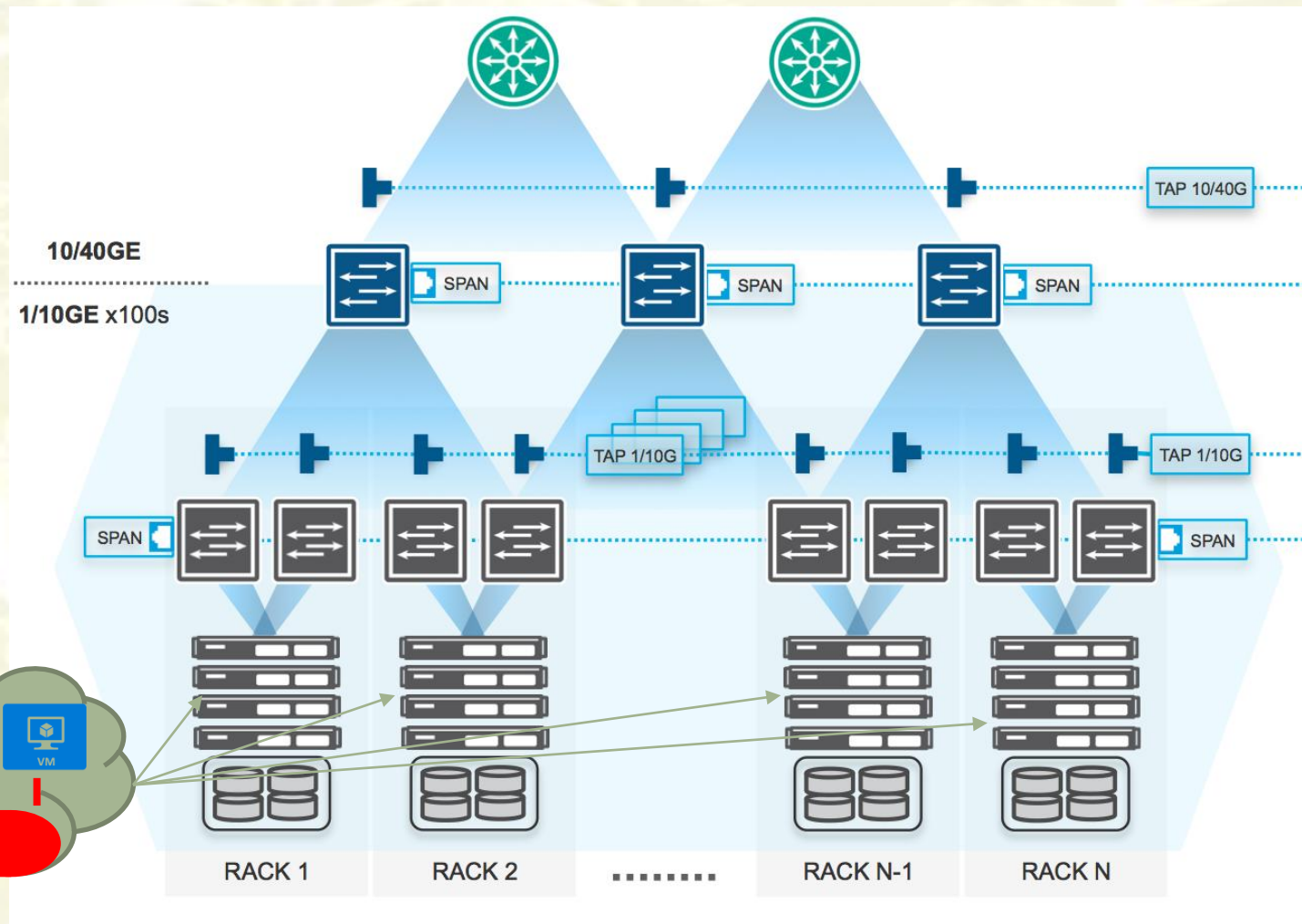


Traditional Datacenter architecture

Datacenter using Virtualization

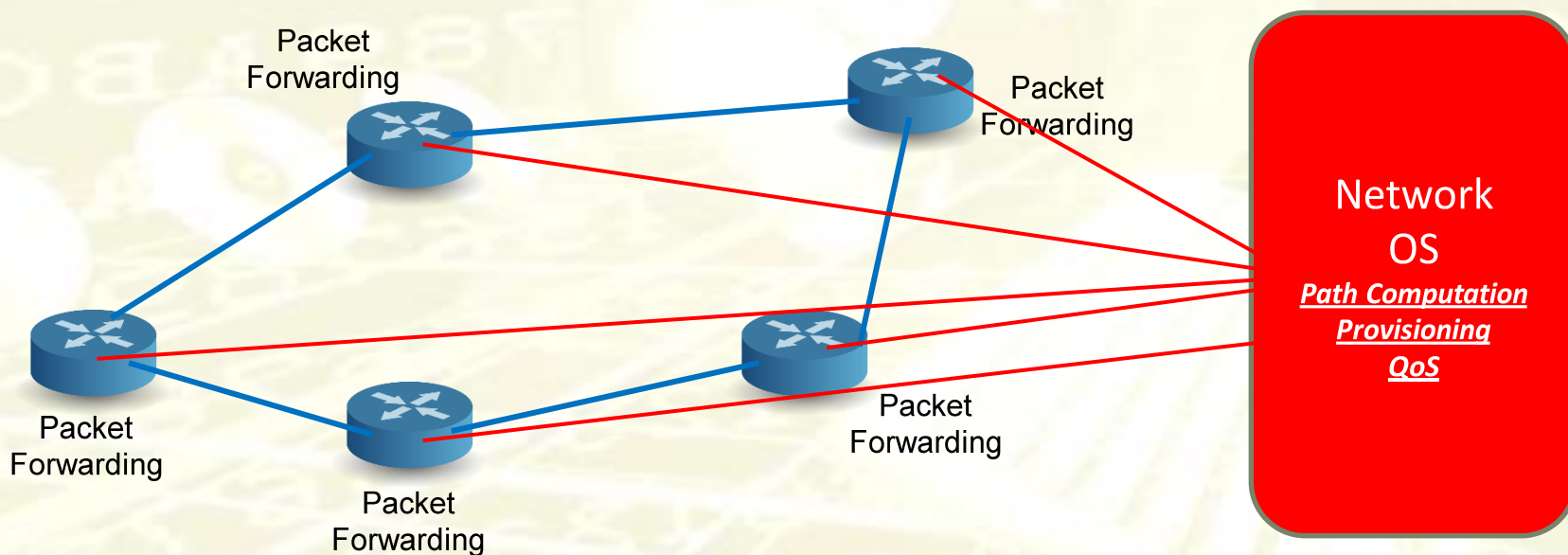


# Datacenter Networking



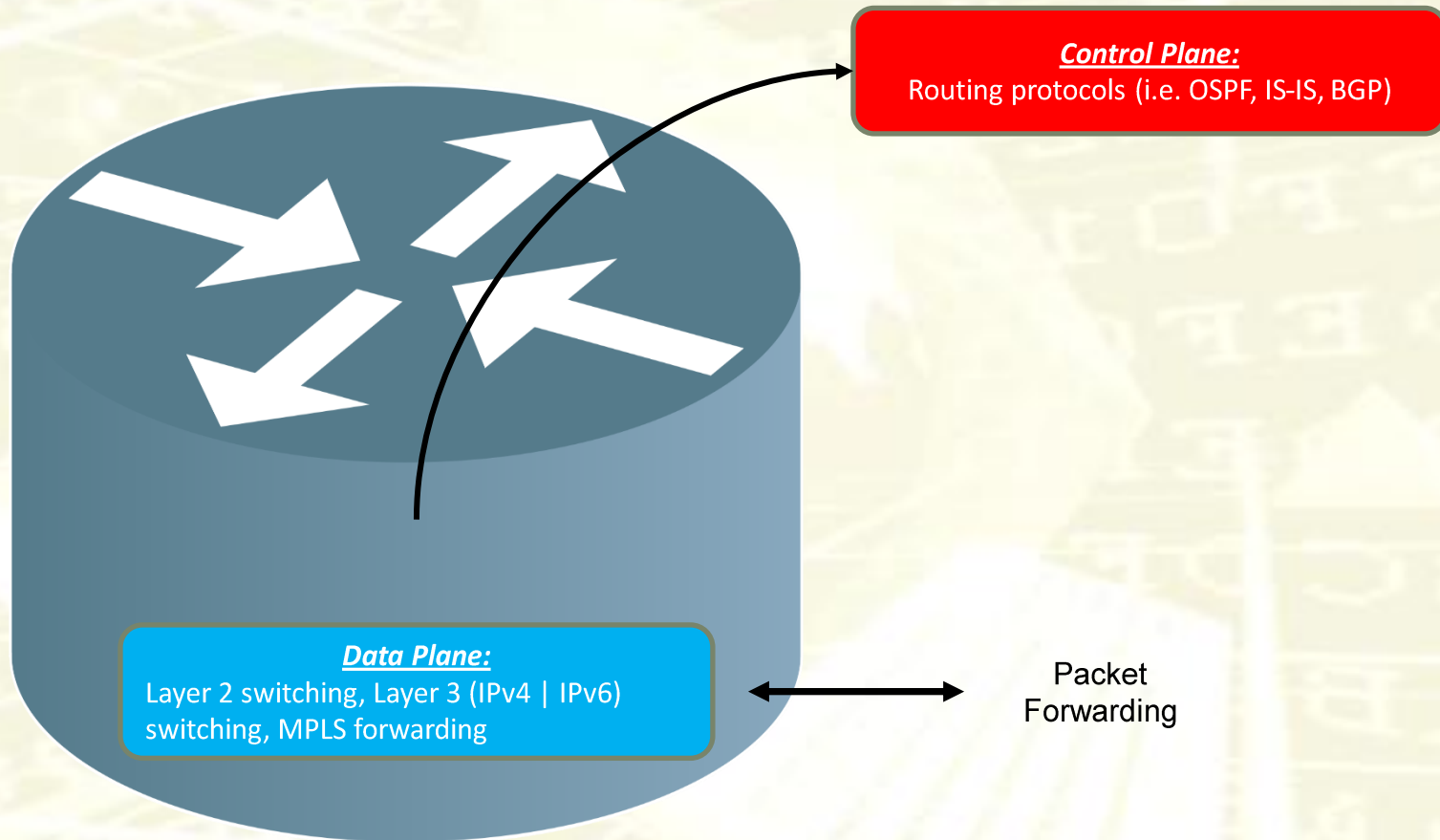
# Concept

- **Software defined networking**: Physical **separation** of the network **control** plane from the **forwarding** plane, where control plane controls several devices
- Centralization of control





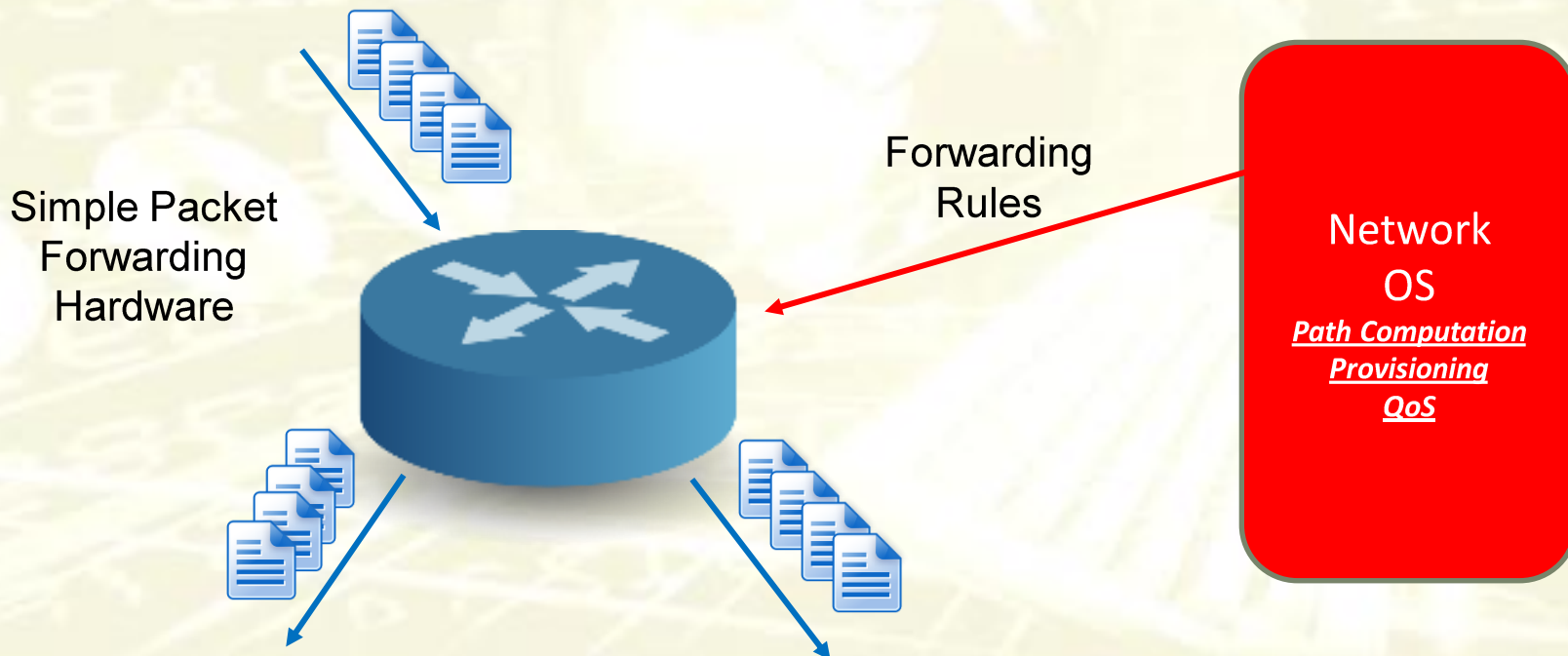
# SDN Networks



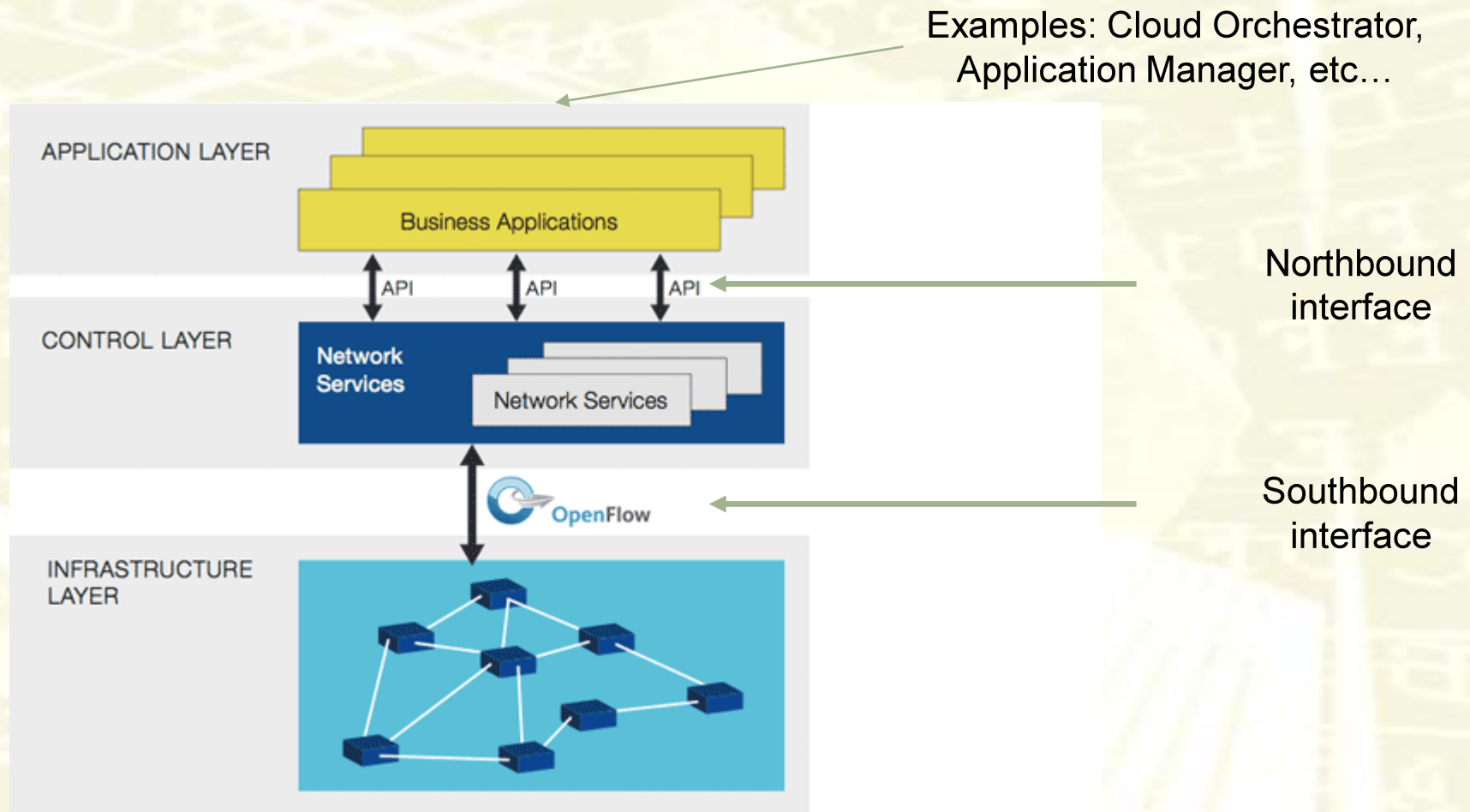
Not all the operations are implemented on the same device

# SDN Hardware

- Routers become *simple hardware* for packet forwarding (switch)
- A *centralized controller* is responsible for defining forwarding rules (controller)



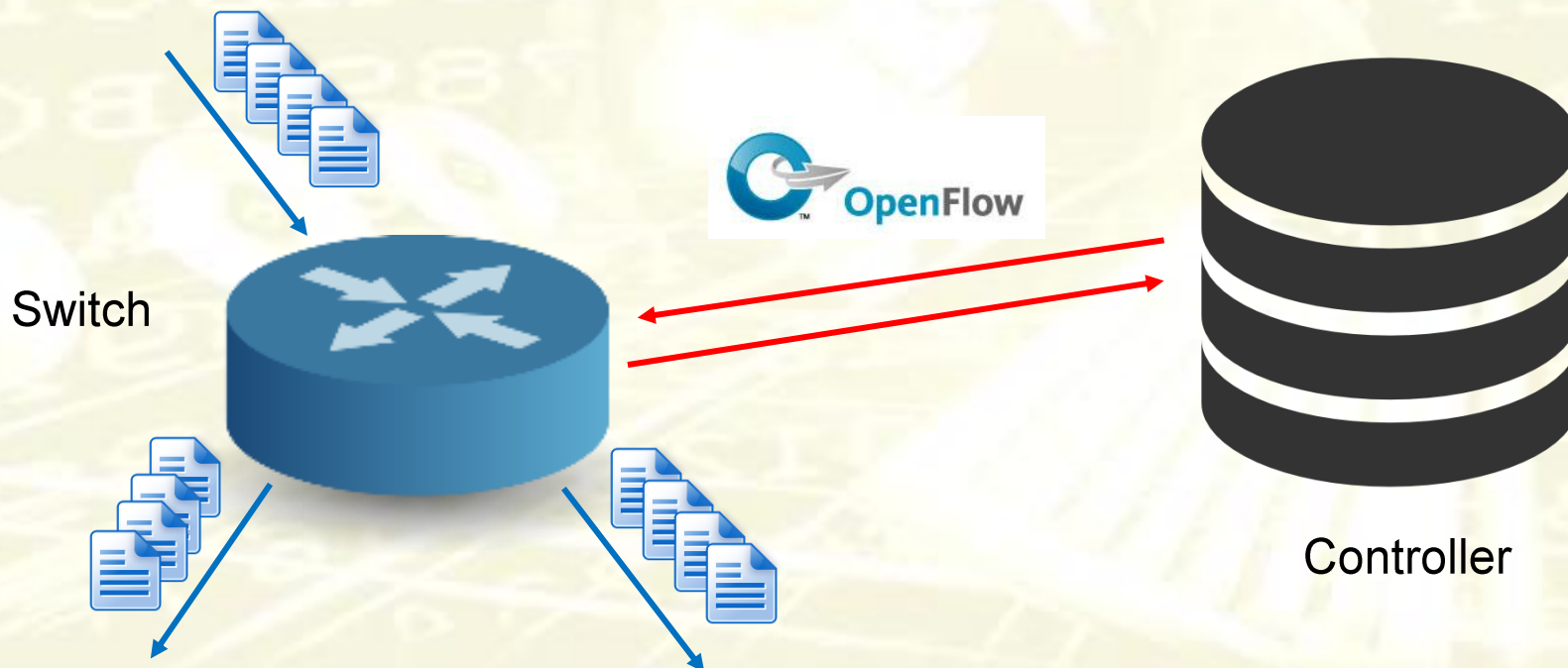
# SDN Architecture





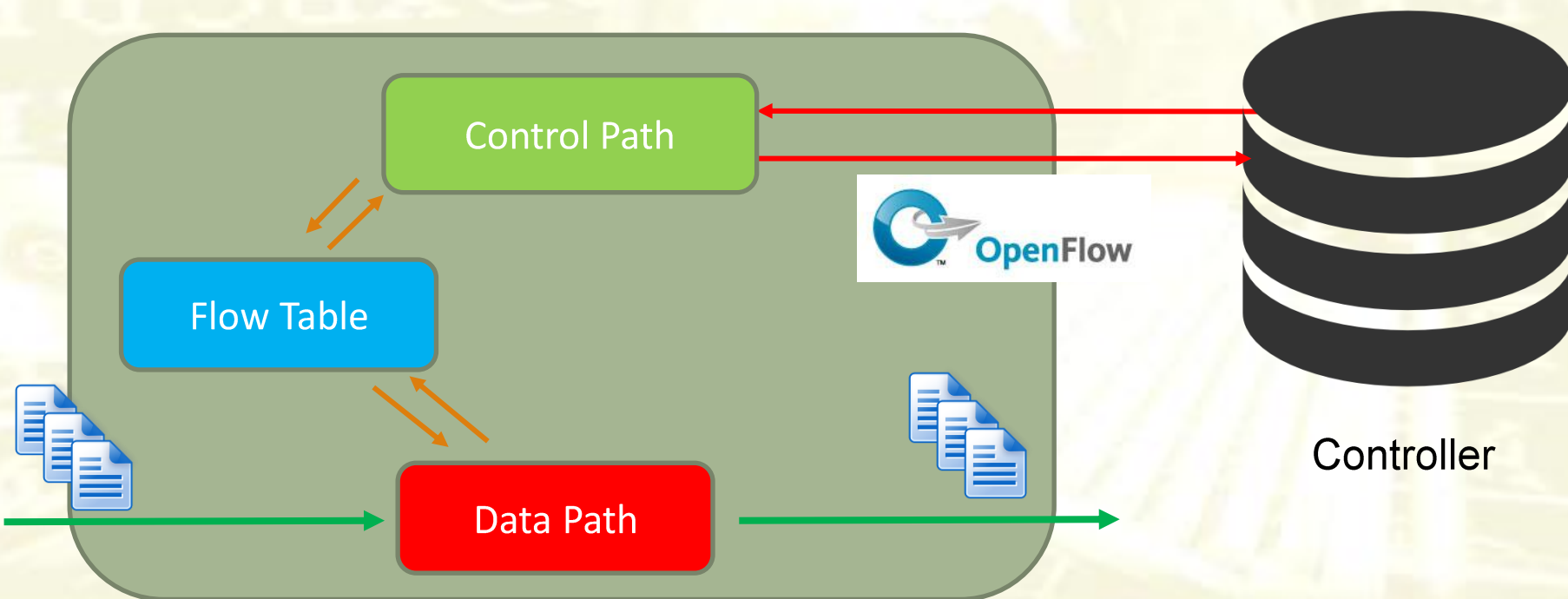
# Open Flow

- Standard communication protocol that defines the interaction between switches and controllers
- It allows remote administration of packet forwarding table



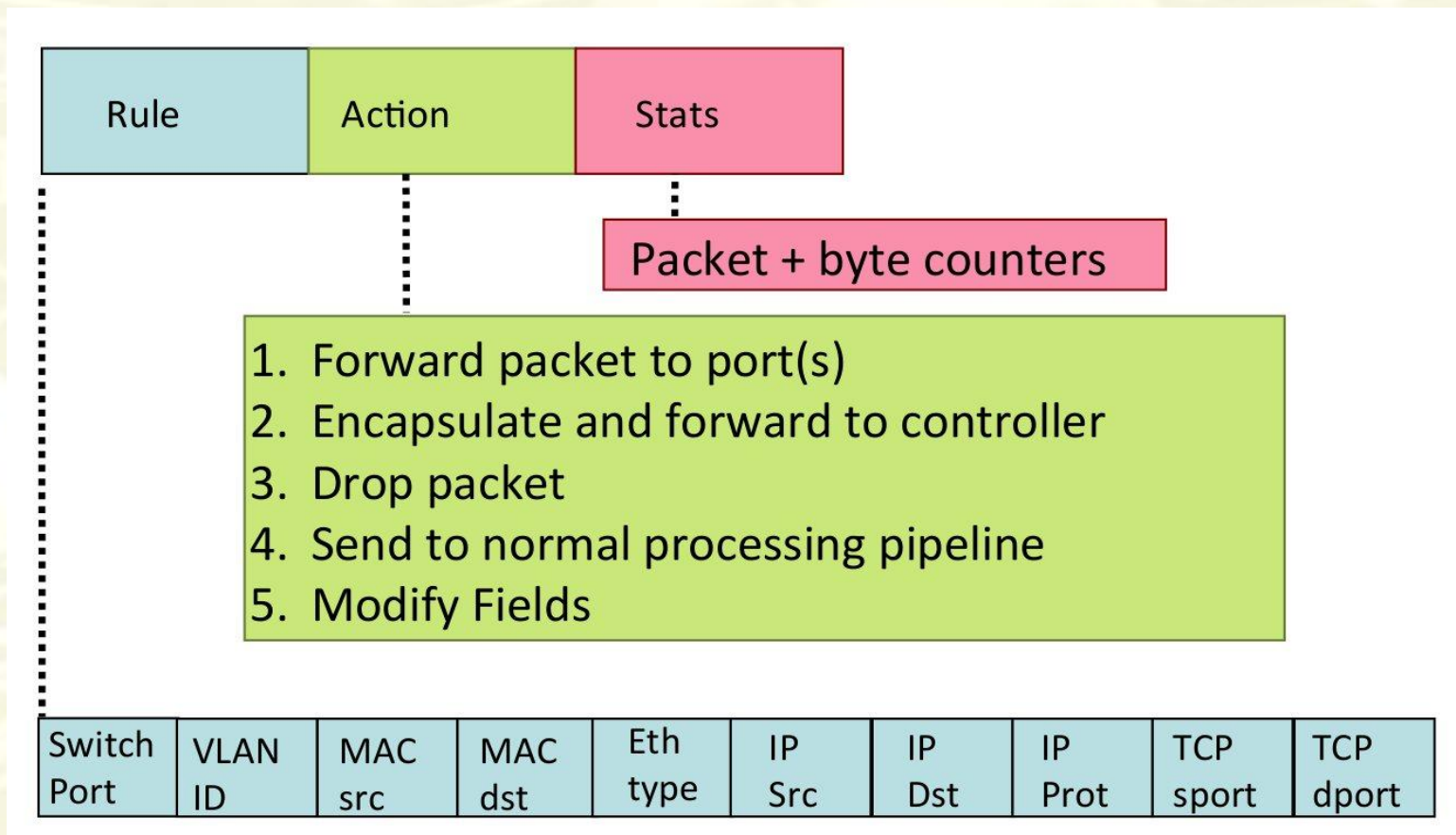
# Open Flow Switch

- Packets are forwarded according to a simple **flow table**
- Controller uses the Open Flow protocol to populate the Flow Table



# Flow Table

- Set of rules (similar to cisco ACL) that determines the action to be performed for each packet



# Rules Examples

- Cross-layer rules for packet classification
- Different functionalities can be implemented:
  - Switching
  - Routing
  - Firewall

## Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

## Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20:..	00:1f:..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

## Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

## Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6



# OF Messages - Startup

- At startup a set of startup messages is sent to allow the controller to discover the capabilities of the switch

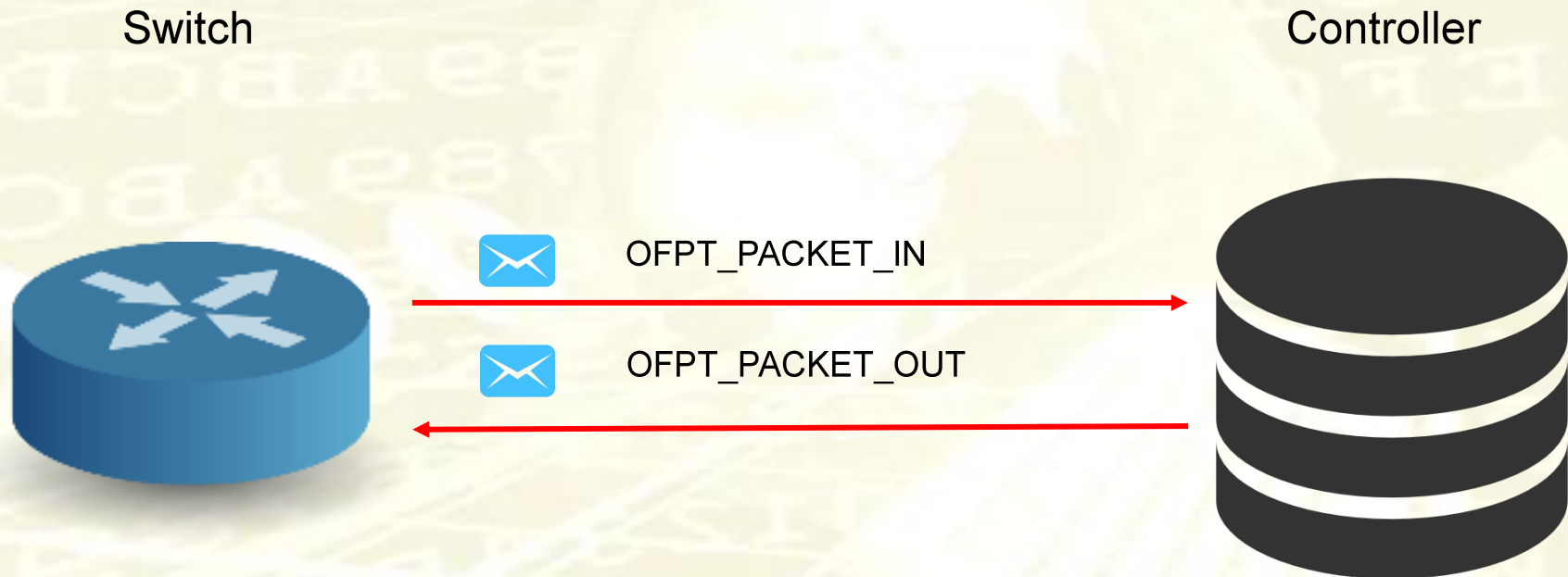




# OF Messages – Normal Operations

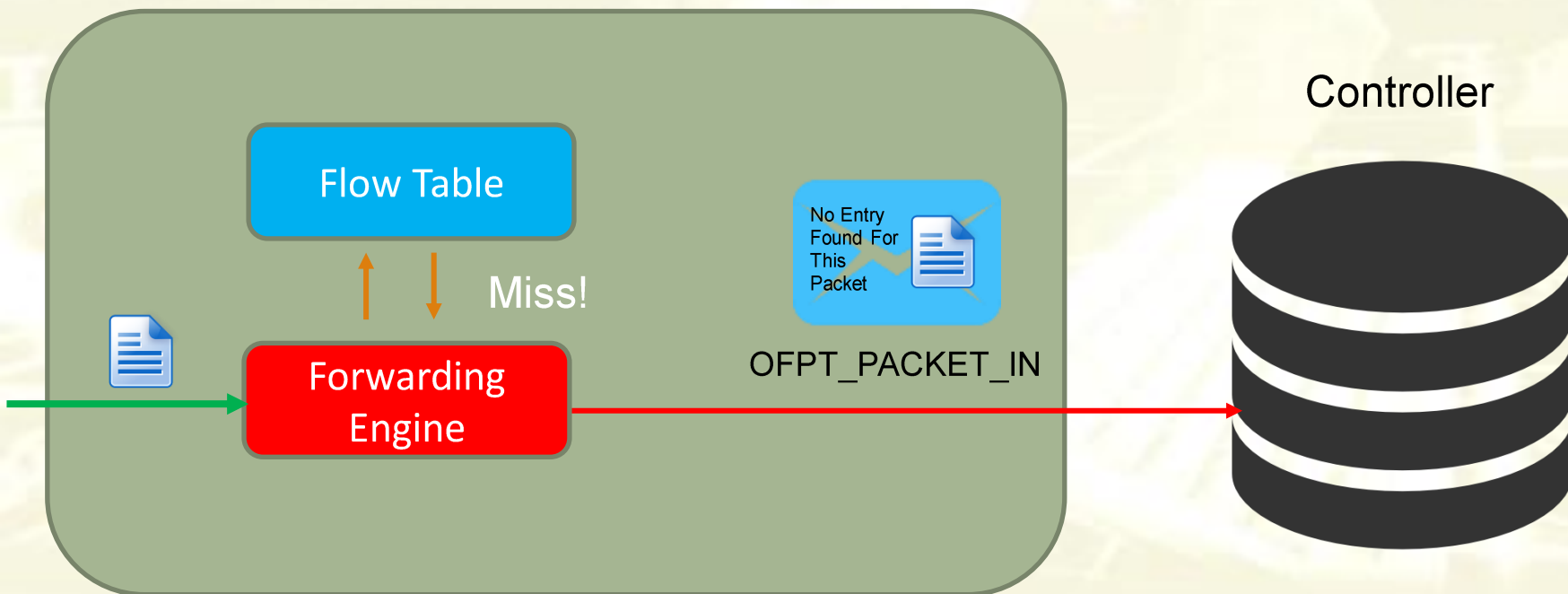


- During normal operations switch and controller interacts with IN and OUT packets



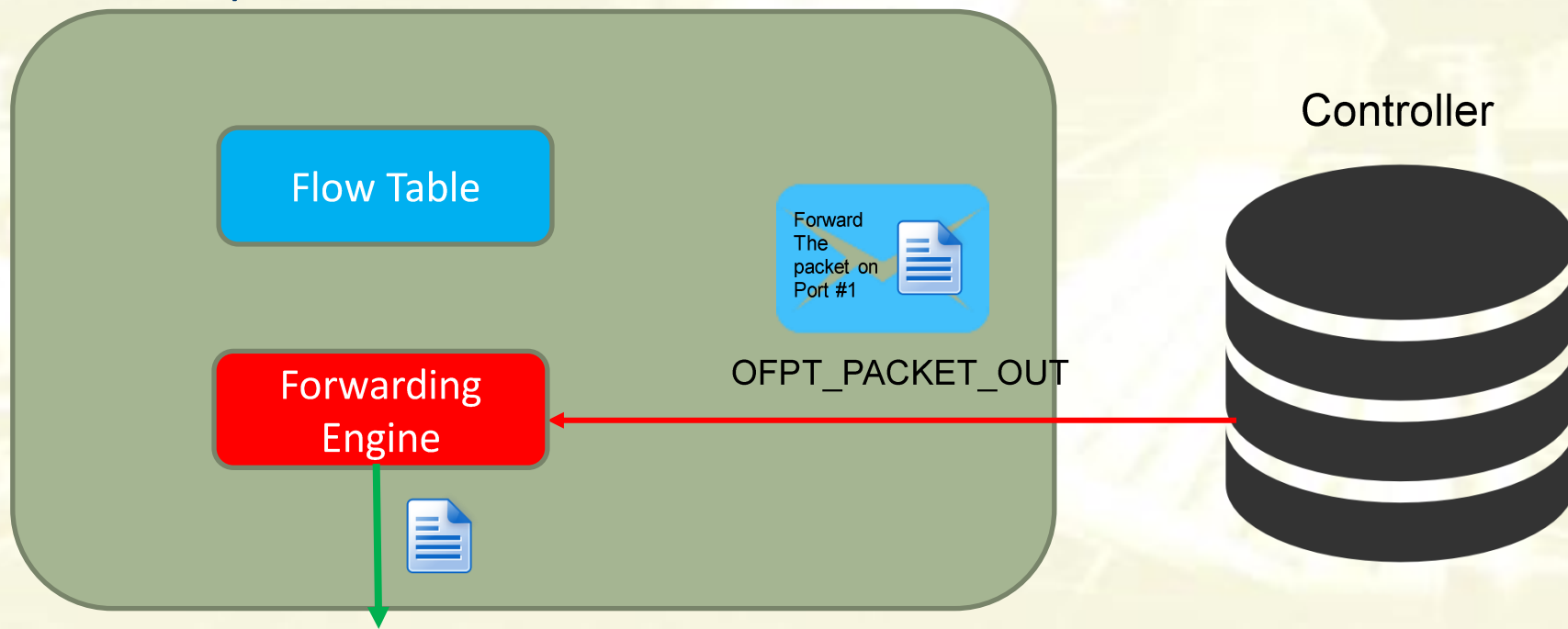
# Switch Operations

- For each received packet the Flow Table is looked up
- If a match is **found** the action is executed, otherwise the **packet (or a reference) is forwarded to the controller encapsulated into a Packet-In**



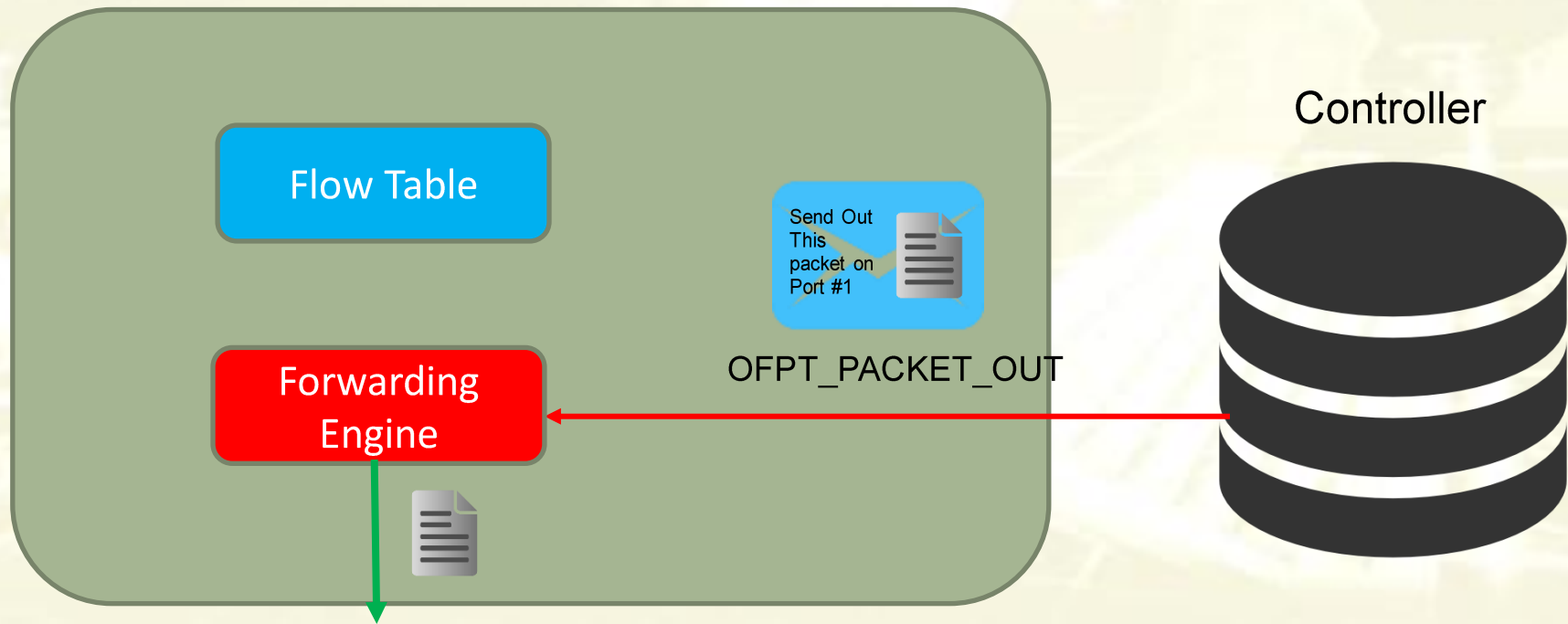
# Switch Operations

- The controller can reply with a Packet-Out specifying the action to be performed (e.g. forward the packet on port #1) and the packet or the reference
- It will be executed only once (no modifications to the Flow Table)



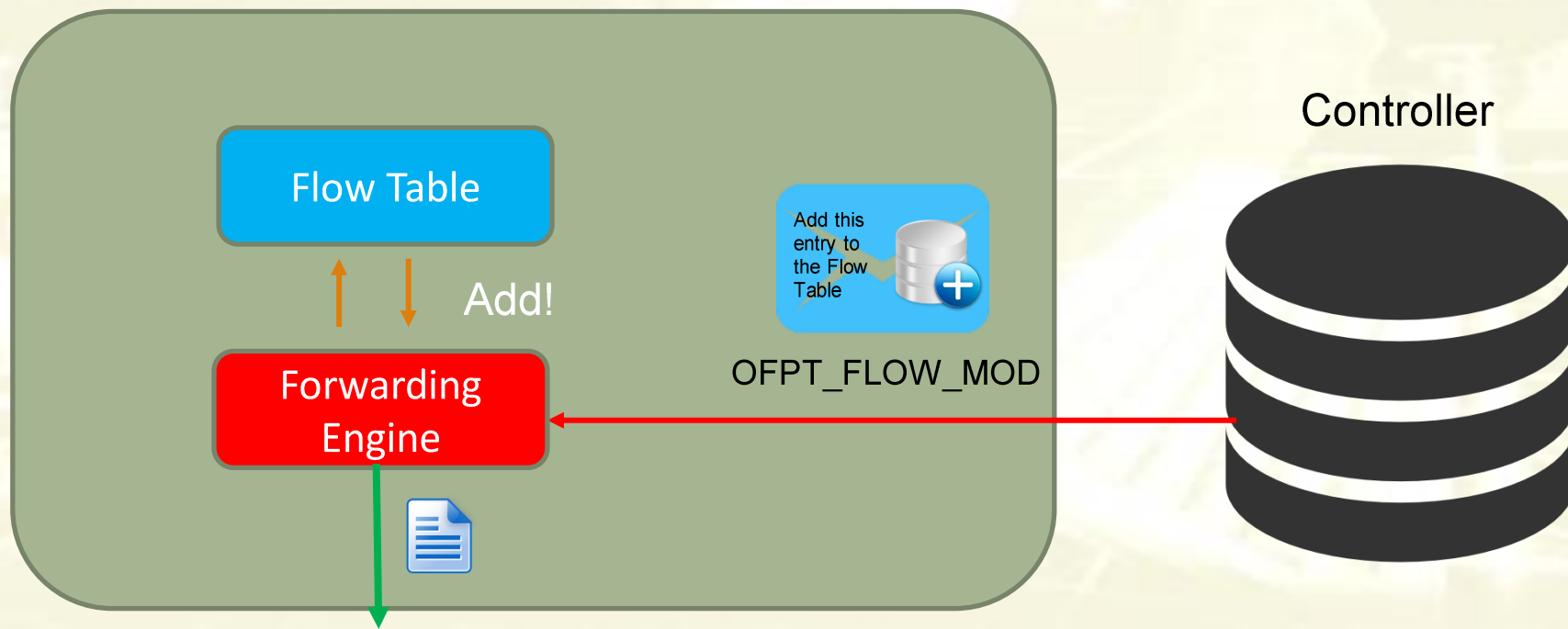
# Switch Operations

- The controller can reply with a Packet-Out specifying a new packet to be sent out
- It will be executed only once (no modifications to the Flow Table)



# Switch Operations

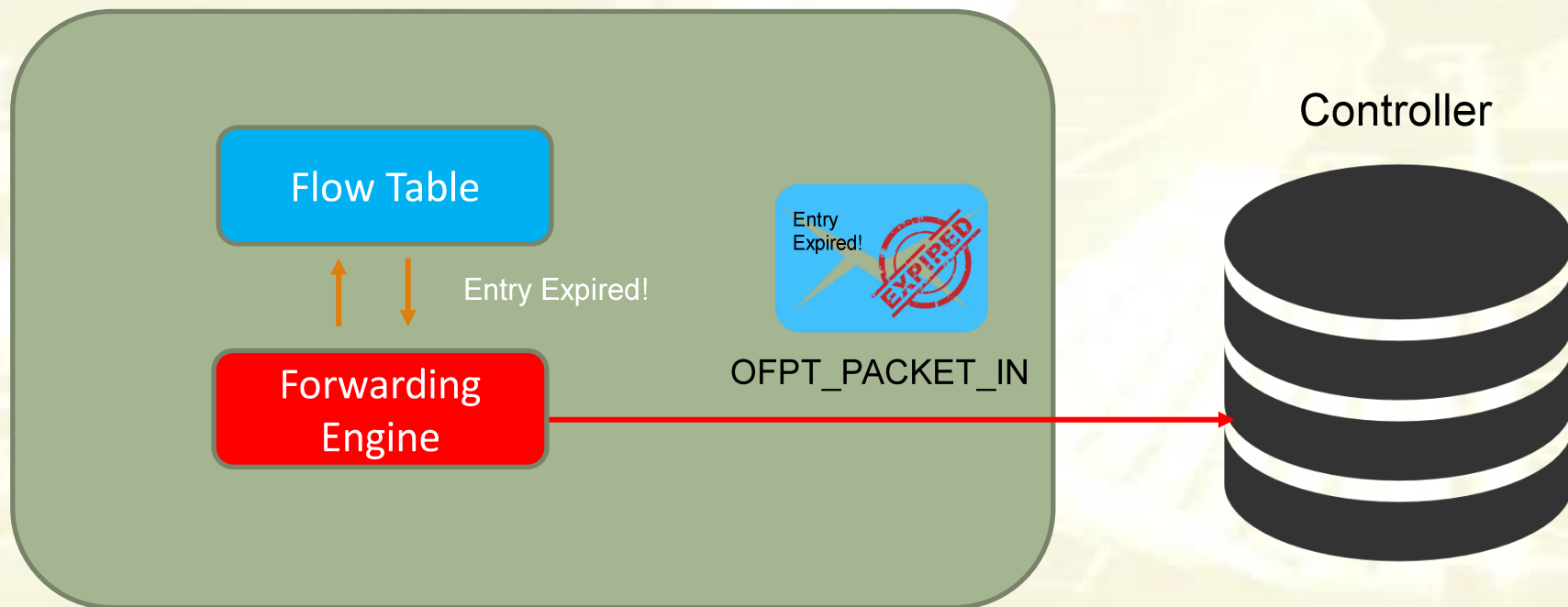
- The controller can reply with a **Flow-Mod** message that instructs the switch to add a new entry to its table
- The new entry will instruct the switch to perform a certain operation without contacting the controller
- The operation associated with the new entry is then executed





# Entry Management

- Entries in the flow table expire
- As the entry is expired a Packet-In is sent to the Controller containing a **Flow-Expired** message
- Entries expire after a hard timeout (always) or after an idle timeout (if packets matching with the entry are not received)



# Mininet

---

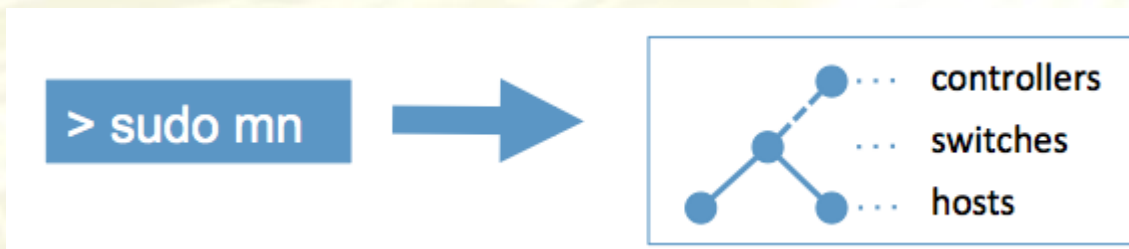
Carlo Vallati

Assistant Professor@ University of Pisa

[c.vallati@iet.unipi.it](mailto:c.vallati@iet.unipi.it)

# Mininet

- Mininet is a virtual network emulator for testing of SDN deployments
- It allows in one program to emulate a network composed of OpenFlow switches and hosts which can generate traffic
- The network of OpenFlow switches can be connected to a real controller



# Mininet



- Launch the simulator:

```
sudo mn --topo single,3  
--ipbase=10.0.0.0
```

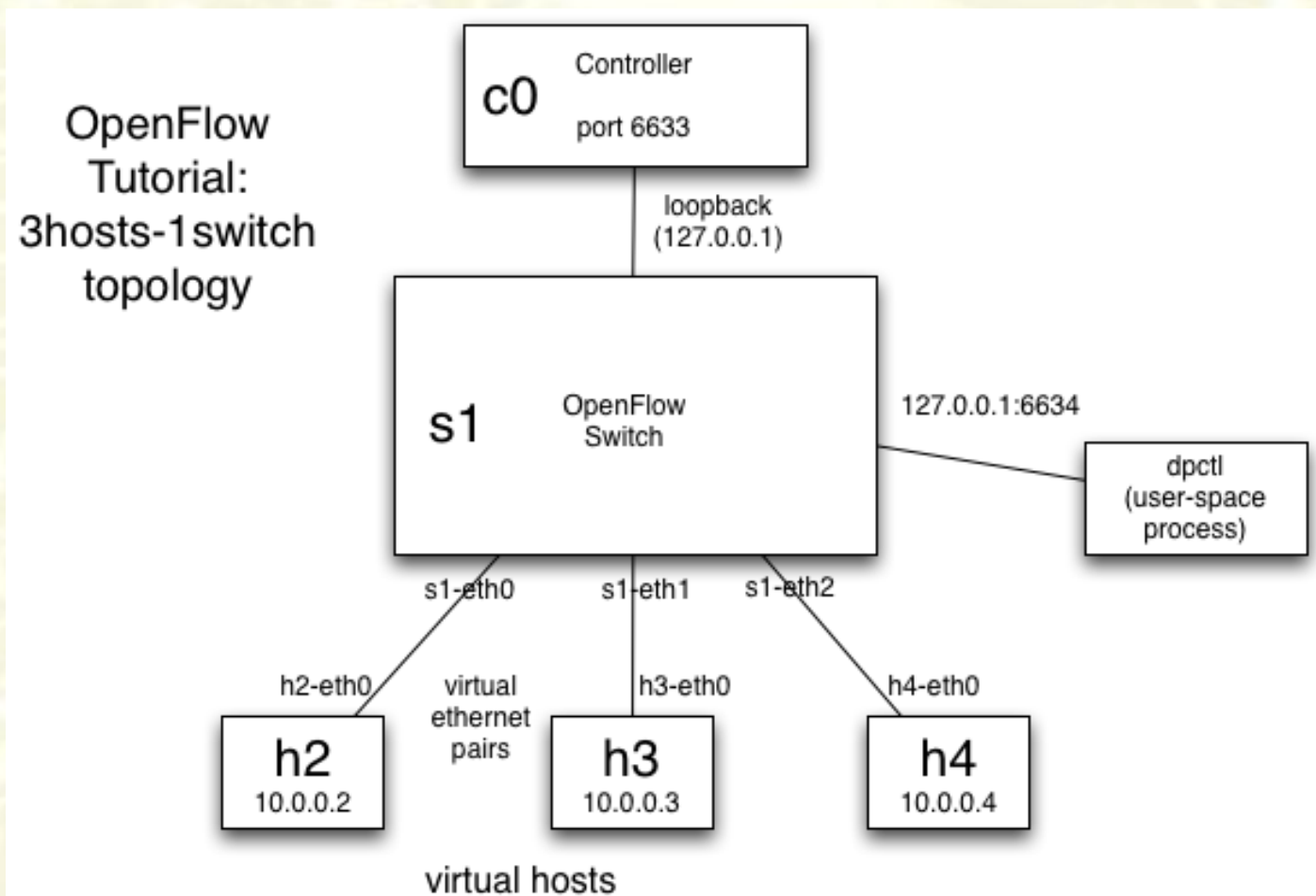
Type of the topology,  
topology with one  
single switch and  
three hosts

IP network subnet for  
simulated hosts

Without a controller the simulated  
switches behave as normal  
switches bridging the different  
networks

# Simulated Architecture

- The basic topology has the following architecture





# Mininet Basics

- Run a program on a host
  - `host_name command`
    - `h1 ping 10.0.0.2`
    - `h1 ifconfig -a`
    - `h1 ifconfig h1-eth0 10.0.0.1`
- Open a separate terminal on a host
  - `xterm host_name`
    - `xterm h1`
    - From the terminal for example you can run `wireshark!`

# Floodlight

---

Carlo Vallati  
Assistant Professor@ University of Pisa  
[c.vallati@iet.unipi.it](mailto:c.vallati@iet.unipi.it)

# Floodlight



- Floodlight is a java framework that allows the implementation of OpenFlow controllers
- It not only provides an implementation of the OpenFlow protocol but also an implementation of some basic operations implemented by controllers



# External Controller



## How to connect mininet to an external controller

- Launch the simulator:

```
sudo mn --topo single,3  
--mac --switch ovsk  
--controller remote,  
ip=127.0.0.1,port=6653,protocols=OpenFlow13  
--ipbase=10.0.0.0
```

The version of the OpenFlow protocol the switch will use to communicate with the controller, 1.3 in this case

IP and port of the real OpenFlow controller (in this case it runs on localhost listening on port 6653)

# Controller



- If you launch the controller now you obtain the following message from the emulator:
  - Unable to contact the remote controller at 127.0.0.1:6633
- Compile and Execute the controller using the “*ant run*” command
- Or open eclipse to run floodlight directly into eclipse
- Wireshark can be executed on the loopback interface to capture OpenFlow messages between controller and switch

```
Main [Java Application] /usr/lib/jvm/java-7-openjdk-i386/bin/java (Nov 19, 2014, 1:20:59 PM)
13:21:05.093 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] New switch connection from /127.0.0.1:43122
13:21:05.099 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] Disconnected switch [/127.0.0.1:43122 DPID[?]]
13:21:05.227 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] New switch connection from /127.0.0.1:43123
13:21:05.259 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Switch OFSwitchBase [/127.0.0.1:43123 DPID[00:00:00:00:00:00:00:01]] bound to class class net.floodlight
13:21:05.261 INFO [n.f.c.OFSwitchBase:New I/O server worker #2-2] Clearing all flows on switch OFSwitchBase [/127.0.0.1:43123 DPID[00:00:00:00:00:00:00:01]]
13:21:05.263 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 connected.
```



# Floodlight

- Floodlight has a modular structure, each module implements one functionality
- Inbound packets are processed in cascade by each module, each module can interrupt the pipeline
- The modules included in the pipeline and their order are specified inside the file
  - `src/main/resources/floodlightdefault.properties`

```
1 floodlight.modules=\
2 net.floodlightcontroller.jython.JythonDebugInterface,\
3 net.floodlightcontroller.counter.CounterStore,\
4 net.floodlightcontroller.storage.memory.MemoryStorageSource,\
5 net.floodlightcontroller.core.internal.FloodlightProvider,\
6 net.floodlightcontroller.threadpool.ThreadPool,\
7 net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl,\
8 net.floodlightcontroller.devicemanager.internal.DefaultEntityClassifier,\
9 net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher,\
0 net.floodlightcontroller.firewall.Firewall,\
1 net.floodlightcontroller.hub.Hub,\
2 net.floodlightcontroller.forwarding.Forwarding,\
3 net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager,\
4 net.floodlightcontroller.topology.TopologyManager,\
5 net.floodlightcontroller.flowcache.FlowReconcileManager,\
6 net.floodlightcontroller.debugcounter.DebugCounter,\
7 net.floodlightcontroller...
```

# New Module

- To create a new module and add it to the pipeline you need to create a new Java class implementing the *IOFMessageListener* and *IFloodlightModule* interfaces
- Eclipse tools can be used to generate a skeleton:

## Add Class In Eclipse

1. Expand the "floodlight" item in the Package Explorer and find the "src/main/java" folder.
2. Right-click on the "src/main/java" folder and choose "New/Class".
3. Enter "net.floodlightcontroller.mactracker" in the "Package" box.
4. Enter "MACTracker" in the "Name" box.
5. Next to the "Interfaces" box, choose "Add...".
6. Add the "IOFMessageListener" and the "IFloodlightModule", click "OK".
7. Click "Finish" in the dialog.

# Initialization and dependences

- Each module that wants to process OF packets need to connect with the **FloodlightProvider** which dispatches the messages
- Explicit dependency on its creation needs to be declared
- At initialization a reference to it needs to be gathered

```
protected IFloodlightProviderService floodlightProvider; // Reference to the provider

// Called at initialization time. Retrieve reference to the provider
@Override
public void init(FloodlightModuleContext context) throws FloodlightModuleException {
    floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);
}

// Called to specify the dependences. Add dependency on the provider
@Override
public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
    Collection<Class<? extends IFloodlightService>> l =
        new ArrayList<Class<? extends IFloodlightService>>();
    l.add(IFloodlightProviderService.class);
    return l;
}
```

# Handle Packet-In Messages

- Each module that wants to process Packet-In messages needs to register and define a **receive** function

```
// Set module name
@Override
public String getName() {
    return ModuleExample.class.getSimpleName();
}

// Called at startup time (after all the modules have been initialized)
@Override
public void startUp(FloodlightModuleContext context) {
    floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
}

// Called every time a Packet-In is received
@Override
public net.floodlightcontroller.core.IListener.Command receive(IOFSwitch sw,
    OFMessage msg, FloodlightContext cntx) {

    Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
        IFloodlightProviderService.CONTEXT_PI_PAYLOAD);

    // Print the source MAC address
    Long sourceMACHash = Ethernet.toLong(eth.getSourceMACAddress().getBytes());
    System.out.printf("MAC Address: {%s} seen on switch: {%s}\n",
        HexString.toHexString(sourceMACHash),
        sw.getId());

    // Let other modules process the packet
    return Command.CONTINUE;
}
```



# Register the new module

- Each needs to be registered in the pipeline

**Append the name of the class in the file**

```
src/main/resources/META-INF/services/net.floodlight.core.module.IFloodlightModule
net.floodlightcontroller.unipi.ModuleExample
```

**Add the module into the pipeline**

```
src/main/resources/floodlightdefault.properties
floodlight.modules = <leave the default list of modules in place>,
net.floodlightcontroller.unipi.ModuleExample
```

- Test it!



# Intercept OpenFlow packets

- You can use Wireshark to intercept OpenFlow packets
- Open Wireshark and capture the packets on the Loopback interface
- Set the following filter to get only OpenFlow packets:  
*tcp.port == 6653*

\*Loopback: lo

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 6653

No.	Time	Source	Destination	Protocol	Length	Info
6553...	220.076678434	127.0.0.1	127.0.0.1	TCP	74	34632 → 6653 [PSH, ACK] Seq=1178229 Ack=7135 Win=49152 Len=8 TSval=99434543 TSecr=99434543 [TCP segment of a reassembled
6553...	220.076702592	127.0.0.1	127.0.0.1	TCP	66	6653 → 34632 [ACK] Seq=7135 Ack=1178237 Win=465920 Len=0 TSval=99434543 TSecr=99434543
6617...	222.077712482	127.0.0.1	127.0.0.1	OpenFl...	74	Type: OFPT_ECHO_REQUEST
6617...	222.077876276	127.0.0.1	127.0.0.1	TCP	74	34632 → 6653 [PSH, ACK] Seq=1178237 Ack=7143 Win=49152 Len=8 TSval=99436543 TSecr=99436543 [TCP segment of a reassembled
6617...	222.077888926	127.0.0.1	127.0.0.1	TCP	66	6653 → 34632 [ACK] Seq=7143 Ack=1178245 Win=465920 Len=0 TSval=99436543 TSecr=99436543
6682...	224.078583364	127.0.0.1	127.0.0.1	OpenFl...	74	Type: OFPT_ECHO_REQUEST
6682...	224.078764704	127.0.0.1	127.0.0.1	TCP	74	34632 → 6653 [PSH, ACK] Seq=1178245 Ack=7151 Win=49152 Len=8 TSval=99438543 TSecr=99438543 [TCP segment of a reassembled
6682...	224.078784412	127.0.0.1	127.0.0.1	TCP	66	6653 → 34632 [ACK] Seq=7151 Ack=1178253 Win=465920 Len=0 TSval=99438543 TSecr=99438543
6705...	225.011470505	127.0.0.1	127.0.0.1	OpenFl...	181	Type: OFPT_PACKET_OUT
6705...	225.014881531	127.0.0.1	127.0.0.1	OpenFl...	181	Type: OFPT_PACKET_OUT
6705...	225.015084923	127.0.0.1	127.0.0.1	TCP	66	34632 → 6653 [ACK] Seq=1178253 Ack=7381 Win=49152 Len=0 TSval=99439479 TSecr=99439475
6705...	225.015957046	127.0.0.1	127.0.0.1	OpenFl...	181	Type: OFPT_PACKET_OUT
6705...	225.067461244	127.0.0.1	127.0.0.1	TCP	66	34632 → 6653 [ACK] Seq=1178253 Ack=7496 Win=49152 Len=0 TSval=99439531 TSecr=99439480
6706...	225.109278081	127.0.0.1	127.0.0.1	OpenFl...	189	Type: OFPT_PACKET_OUT
6706...	225.109333166	127.0.0.1	127.0.0.1	TCP	66	34632 → 6653 [ACK] Seq=1178253 Ack=7619 Win=49152 Len=0 TSval=99439573 TSecr=99439573
6706...	225.109860986	127.0.0.1	127.0.0.1	OpenFl...	189	Type: OFPT_PACKET_OUT
6706...	225.109867756	127.0.0.1	127.0.0.1	TCP	66	34632 → 6653 [ACK] Seq=1178253 Ack=7742 Win=49152 Len=0 TSval=99439574 TSecr=99439574
6706...	225.110326361	127.0.0.1	127.0.0.1	OpenFl...	189	Type: OFPT_PACKET_OUT
6706...	225.110334281	127.0.0.1	127.0.0.1	TCP	66	34632 → 6653 [ACK] Seq=1178253 Ack=7865 Win=49152 Len=0 TSval=99439574 TSecr=99439574
6760...	227.113101620	127.0.0.1	127.0.0.1	OpenFl...	74	Type: OFPT_ECHO_REQUEST
6760...	227.113184933	127.0.0.1	127.0.0.1	TCP	66	34632 → 6653 [ACK] Seq=1178253 Ack=7873 Win=49152 Len=0 TSval=99441576 TSecr=99441576
6760...	227.113418099	127.0.0.1	127.0.0.1	TCP	74	34632 → 6653 [PSH, ACK] Seq=1178253 Ack=7873 Win=49152 Len=8 TSval=99441576 TSecr=99441576 [TCP segment of a reassembled