Software Defined Network: Control Plane

Peng Zhang
School of Computer Science and Technology
Xi'an Jiaotong University

Spring 2024

Programing Levels

- Level 1: South-Bound Interface
  - Program switches directly through OpenFlow
- Level 2: SDN Controller
  - Program with general-purpose language like C, Java, Python
- Level 3: Network Programming Languages
  - Program with domain-specific languages for networks

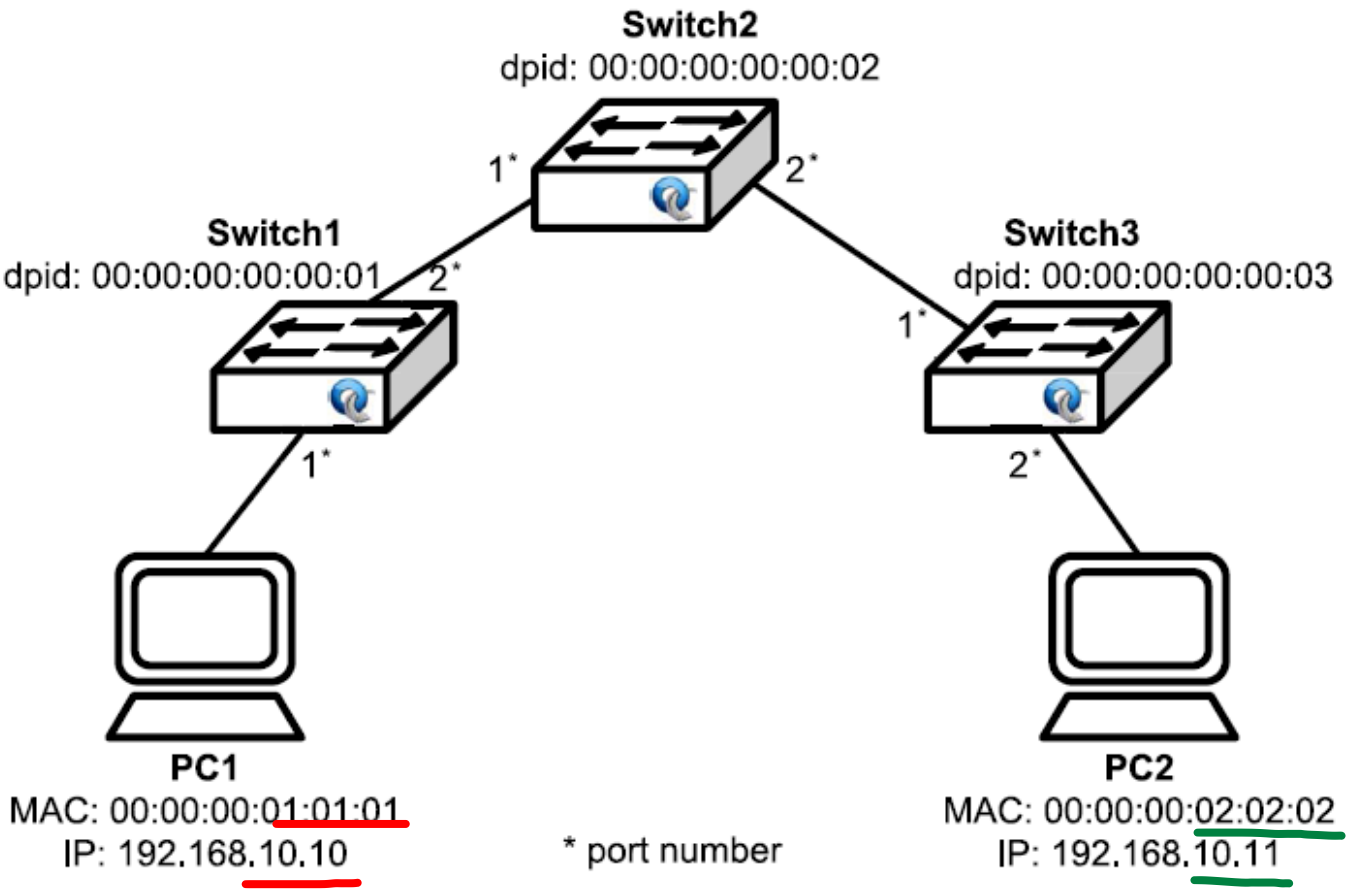
A Domain-Specific Language (DSL) is a programming language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain

Outline

- South-Bound Interface
- SDN Controllers
- Network Programming Languages

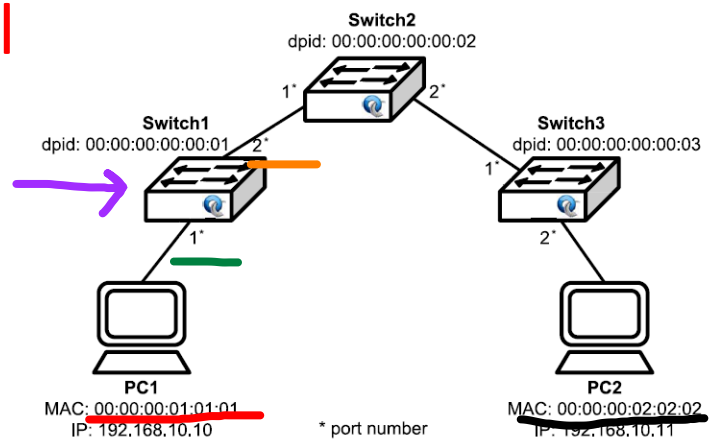
Level 1: South-Bound Interface

- Example: make PC1 and PC2 reachable on layer 2



Level 1: South-Bound Interface

- Use ovs-ofctl



注意这里Sw Port 和 节点 MAC地址的对应关系

Switch1

```
ovs-ofctl add-flow Switch1 priority=100,dl_dst=00:00:00:01:01:01,actions=output:1
ovs-ofctl add-flow Switch1 priority=100,dl_dst=00:00:00:02:02:02,actions=output:2
```

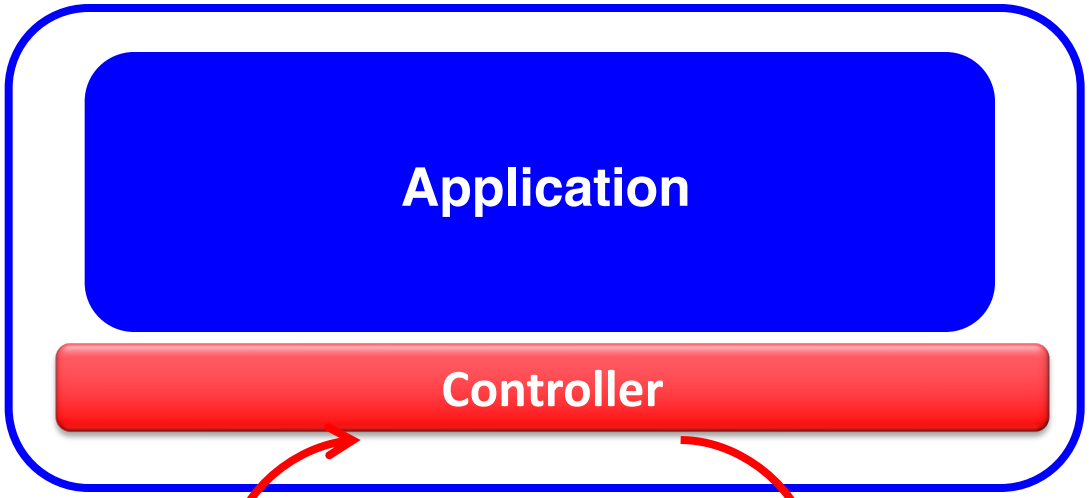
Switch2

```
ovs-ofctl add-flow Switch2 priority=100,dl_dst=00:00:00:01:01:01,actions=output:1
ovs-ofctl add-flow Switch2 priority=100,dl_dst=00:00:00:02:02:02,actions=output:2
```

Switch3

```
ovs-ofctl add-flow Switch3 priority=100,dl_dst=00:00:00:01:01:01,actions=output:1
ovs-ofctl add-flow Switch3 priority=100,dl_dst=00:00:00:02:02:02,actions=output:2
```

Level 2: Controller



Events from switches

Topology changes,
Traffic statistics,
Arriving packets

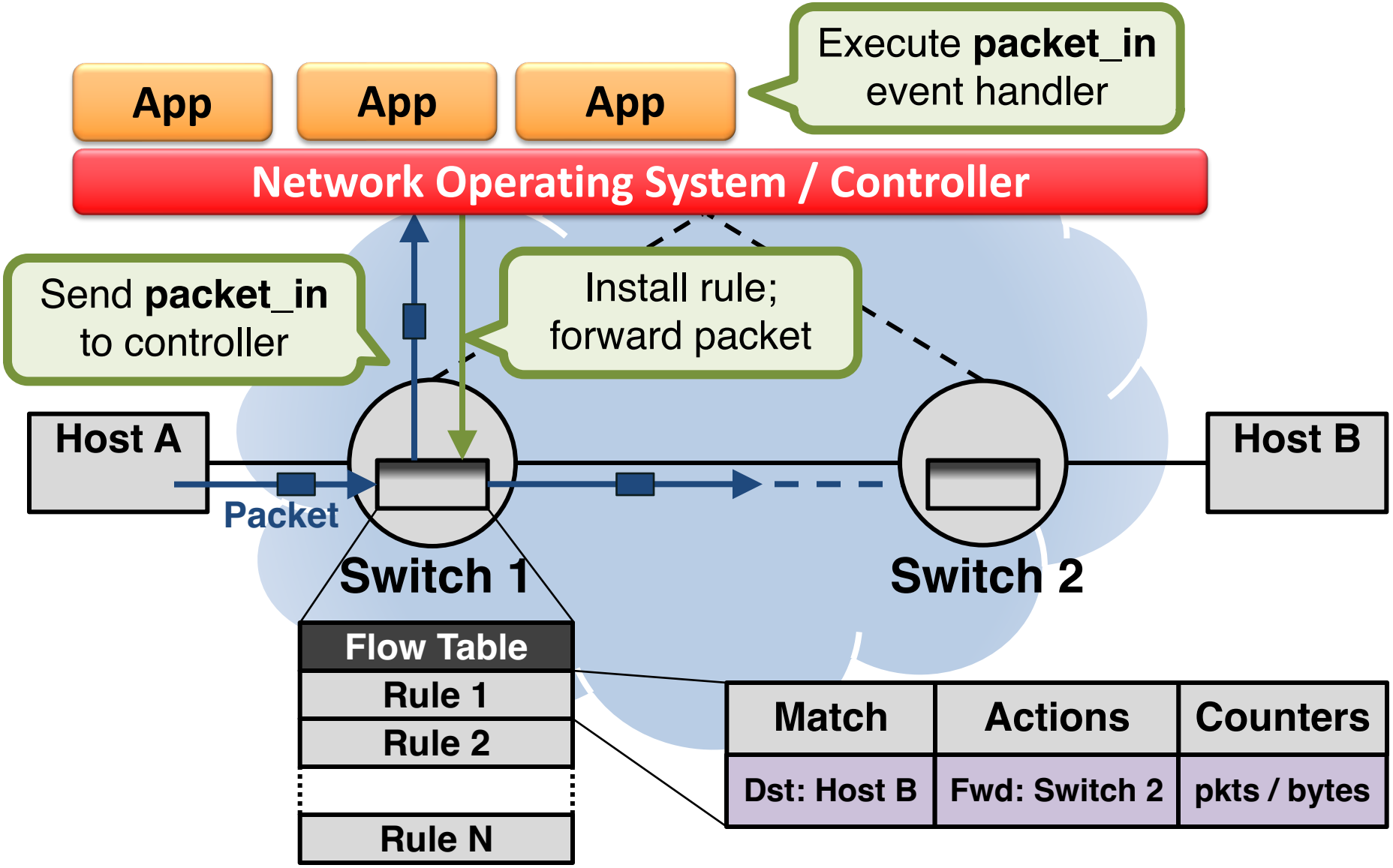
Commands to switches

(Un)install rules,
Query statistics,
Send packets

Modes

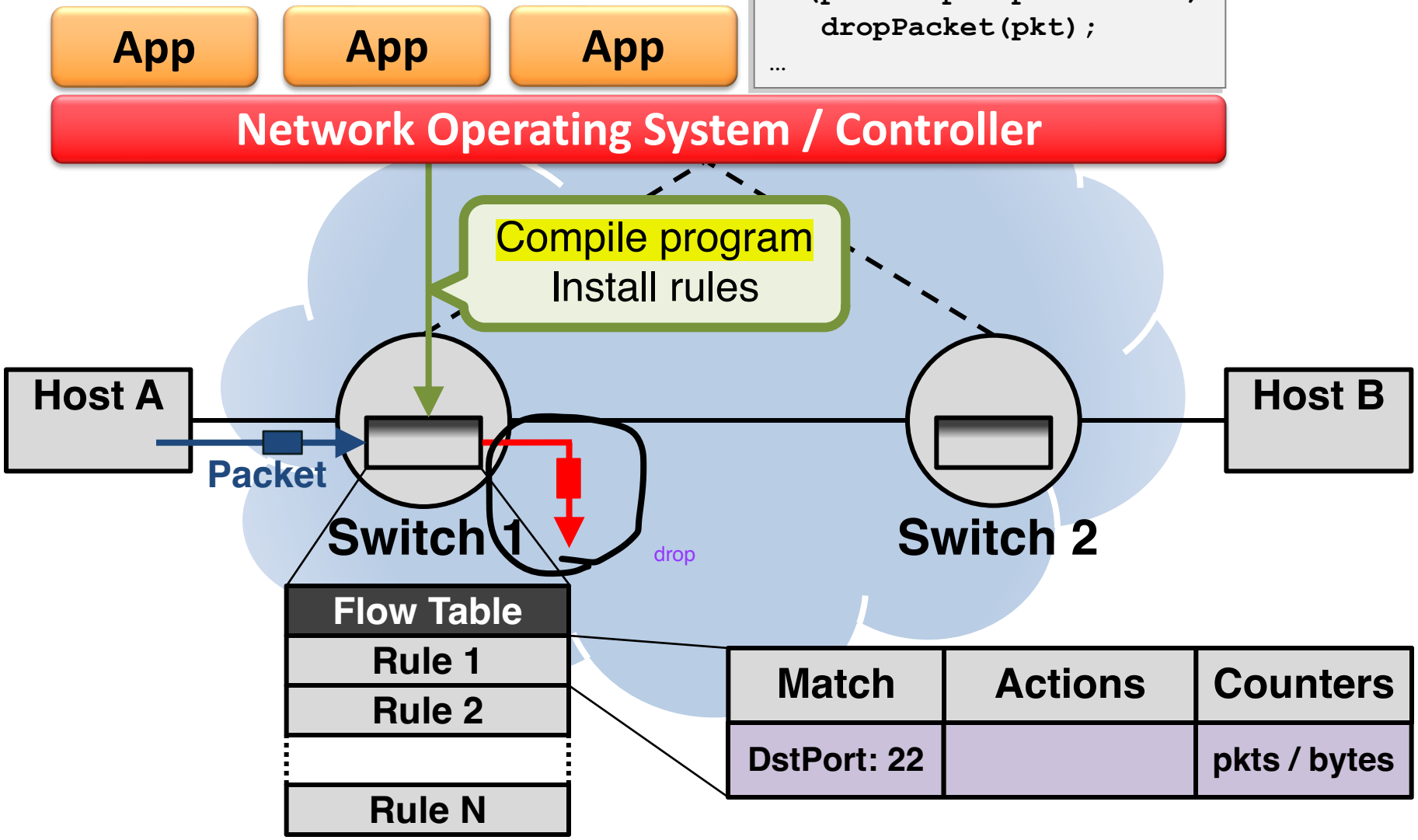
- Reactive vs. Proactive
- Centralized vs. Distributed

Reactive Control



Proactive Control

```
firewall.c
if(pkt->tcp->dport == 22)
    dropPacket(pkt);
…
```



Reactive vs. Proactive

Reactive

First packet of flow triggers controller to insert flow entries

Efficient use of flow table

Every flow incurs an additional flow setup time

If control connection lost, switch has limited utility

Proactive

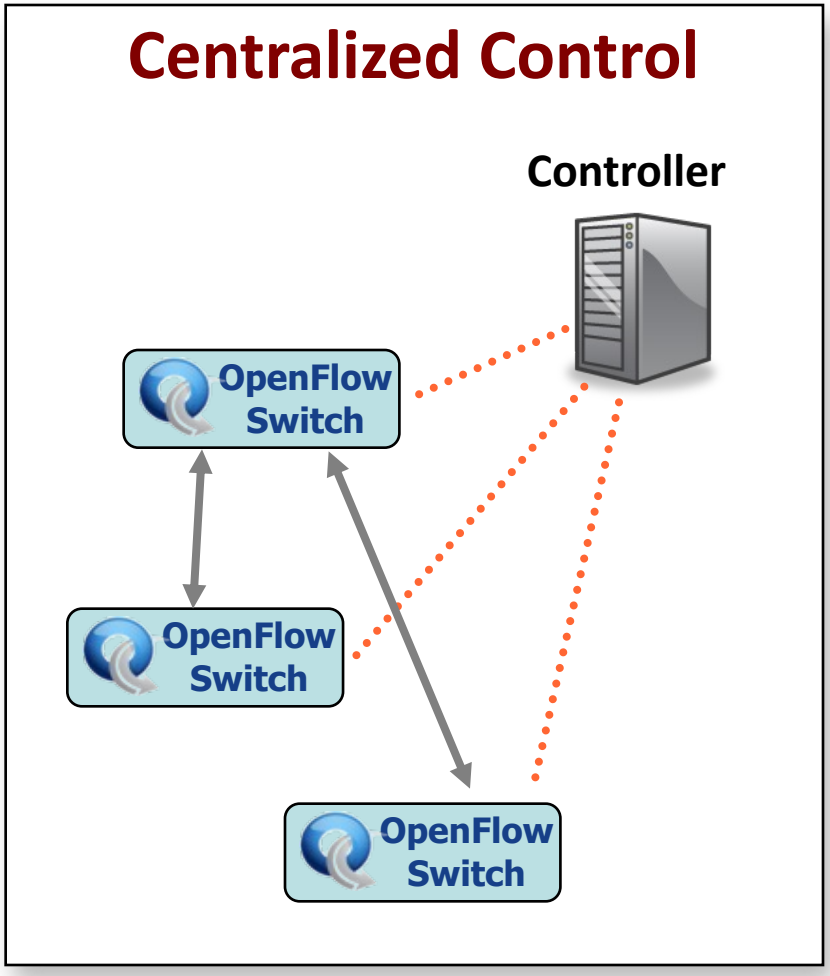
Controller pre-populates flow table in switch

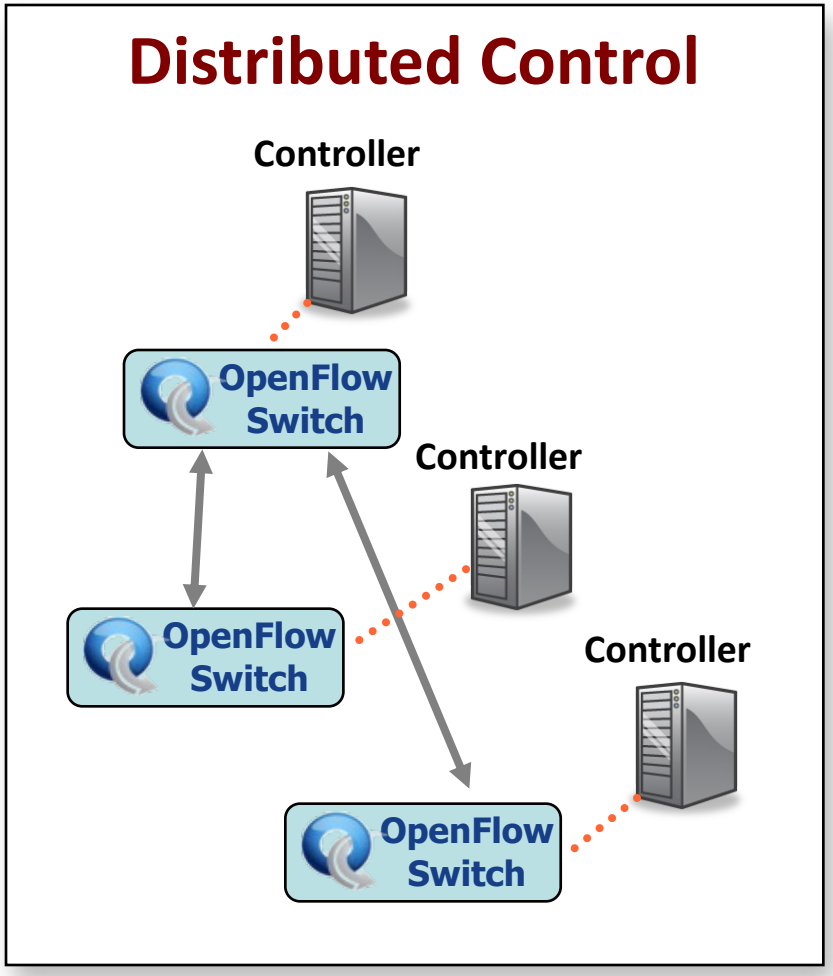
Requires aggregated rules

Zero additional flow setup time

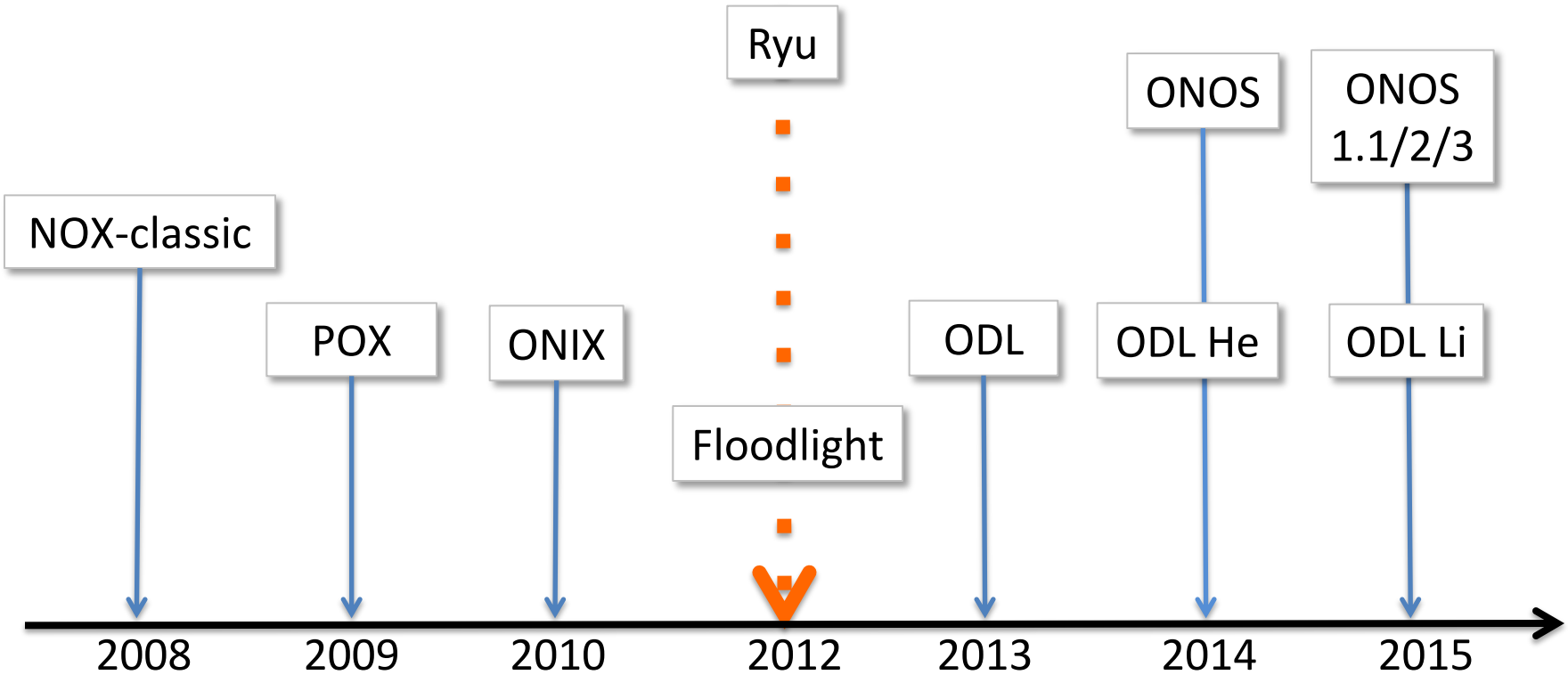
Loss of control connection does not disrupt traffic

Centralized vs Distributed Control





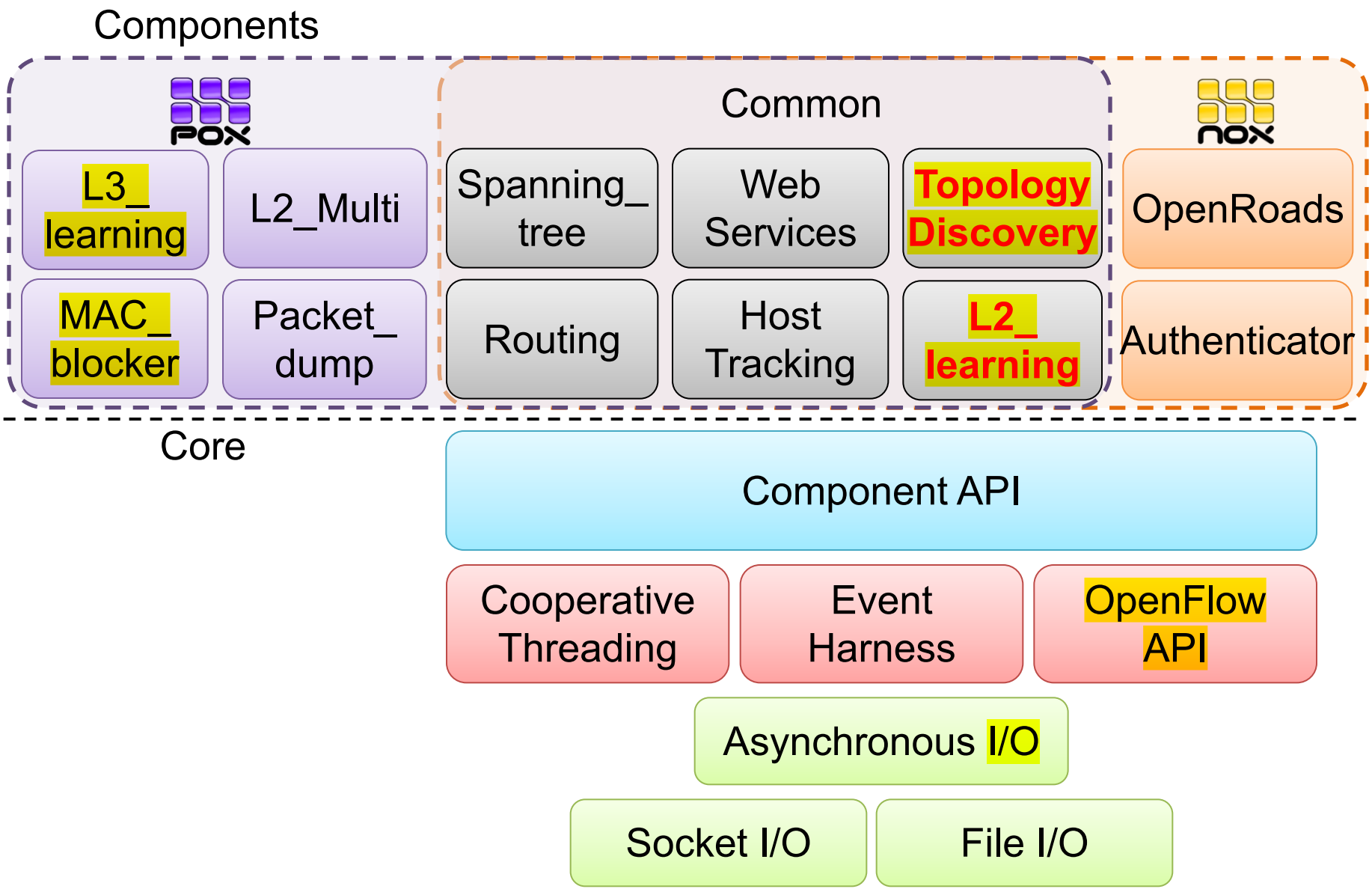
History of SDN Controllers



NOX and POX

- NOX
  - The first OpenFlow controller in 2008
  - Further developed by CPqD to support OpenFlow 1.3 in Nov. 2012
- POX
  - NOX's younger sibling implemented using Python
  - Performs well compared to NOX-classic

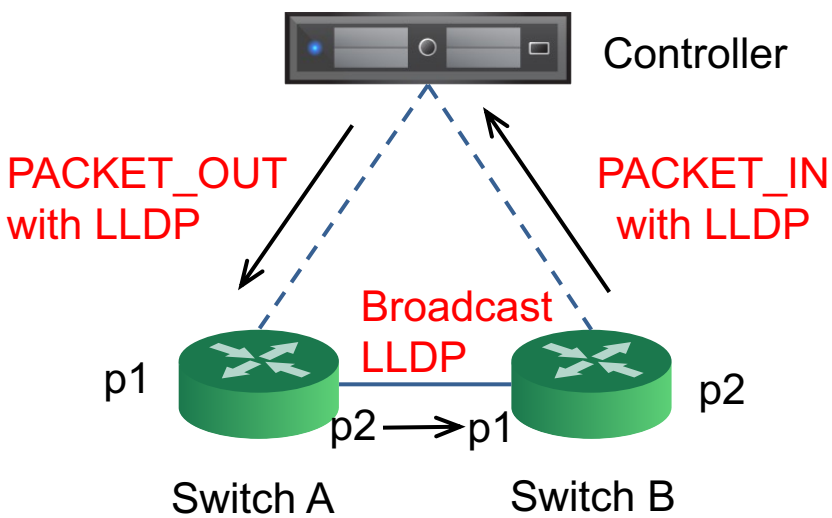
NOX/POX Architecture



Example 1: Topology Discovery

- Link Layer Discovery Protocol (LLDP)
  - A vendor neutral link layer protocol in the Internet Protocol Suite used by network devices for advertising their identity, capabilities and neighbors on an IEEE 802 LAN.

SRC Switch	Switch Port	Dest Switch	Switch Port
A	p2	B	p1
B	p1	A	p2



Example 2: Learning Switch

```
def _handle_PacketIn (self, event):
  """
  Handle packet in messages from the switch to implement above algorithm.
  """
  self.macToPort[packet.src] = event.port # 1

  if not self.transparent: # 2
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
      drop() # 2a
      return

  if packet.dst.is_multicast:
    flood() # 3a
  else:
    if packet.dst not in self.macToPort: # 4
      flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
    else:
      port = self.macToPort[packet.dst]
```

Example 2: Learning Switch

```
if port == event.port: # 5
  # 5a
  log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
      % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
  drop(10)
  return
# 6
log.debug("installing flow for %s.%i -> %s.%i" %
          (packet.src, event.port, packet.dst, port))
msg = of.ofp_flow_mod()
msg.match = of.ofp_match.from_packet(packet, event.port)
msg.idle_timeout = 10
msg.hard_timeout = 30
msg.actions.append(of.ofp_action_output(port = port))
msg.data = event.ofp # 6a
self.connection.send(msg)
```

Floodlight

- An open, free, OpenFlow controller in Java
- Slowly supporting OpenFlow v1.3

Apache

Research and commercial friendly

Java

Easy to build, run, and develop

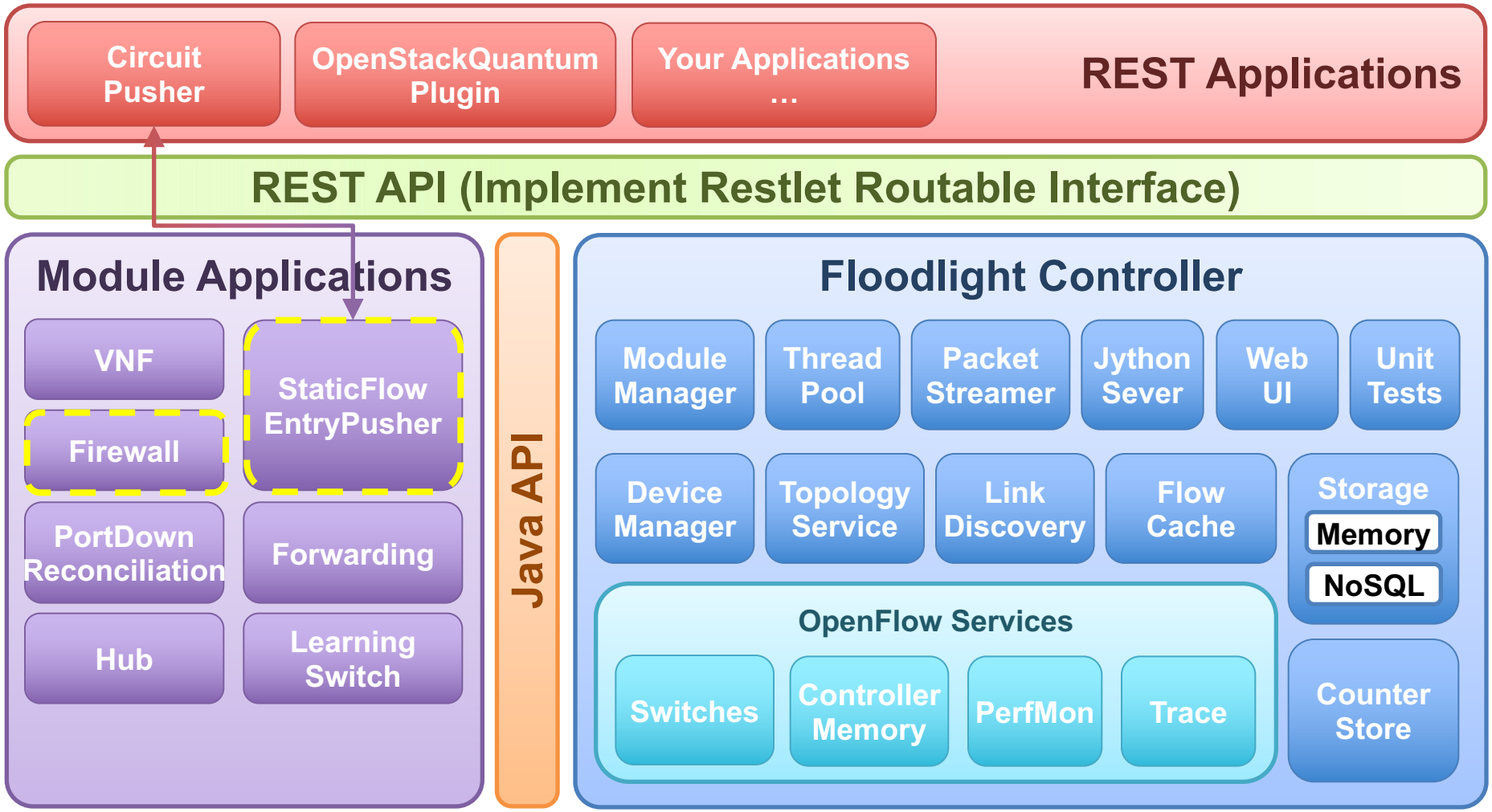
Toolchain

Rich set of build and debugging tools

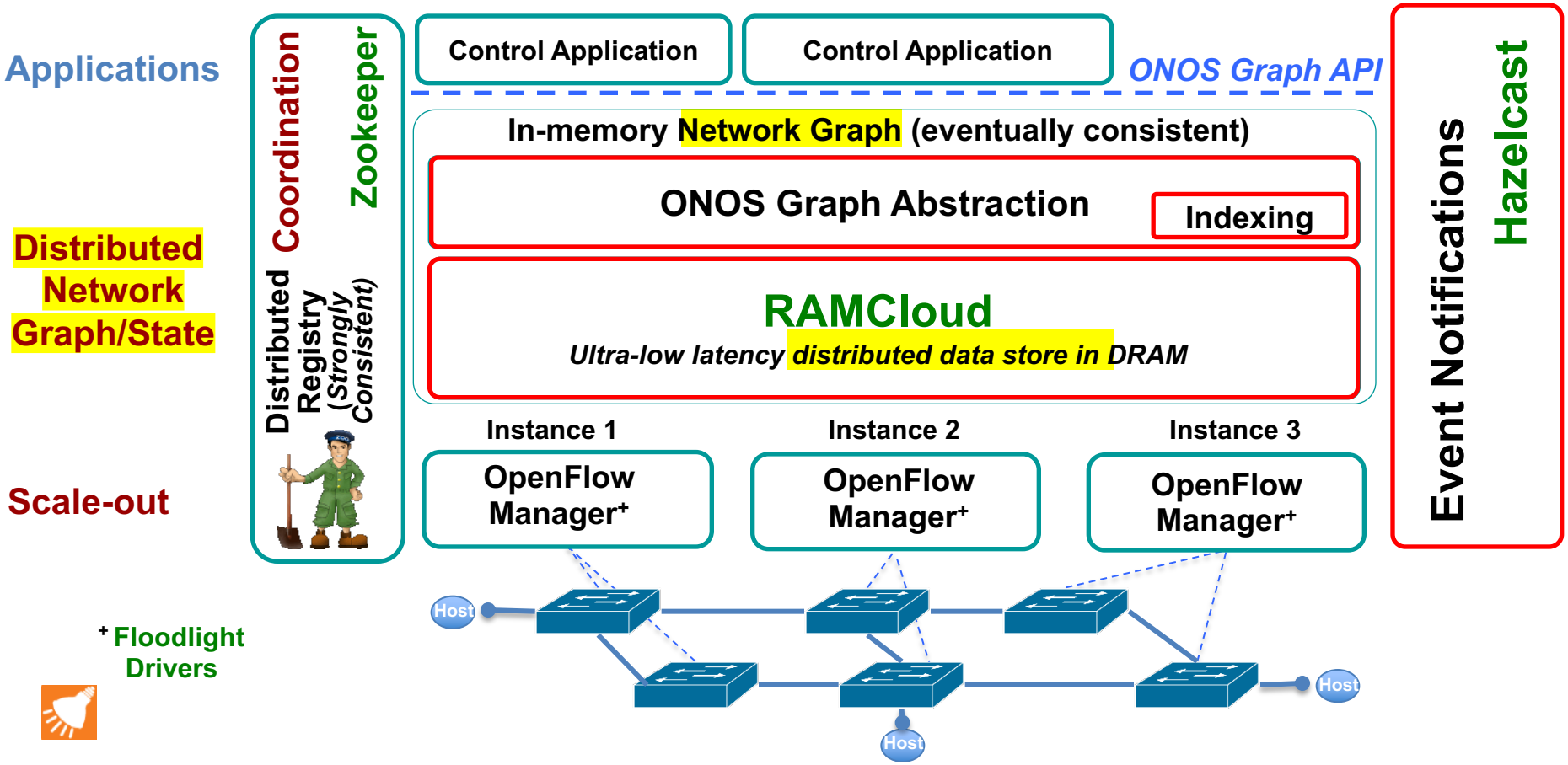
big switch networks

Community of OpenFlow experts, access to commercial upgrades, and frequent testing

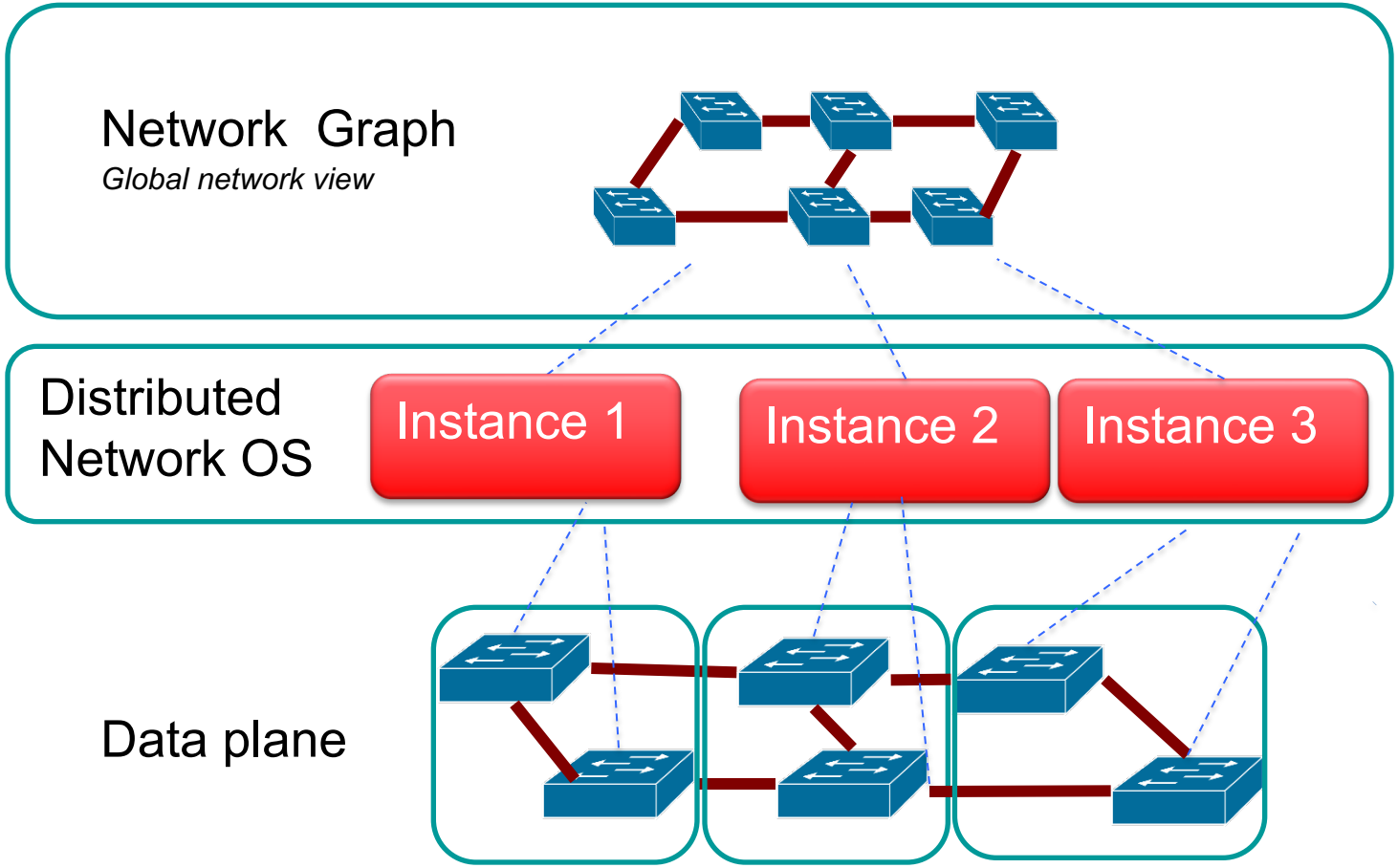
Floodlight Architecture



ONOS: Open Network Operating System



ONOS Scale-Out

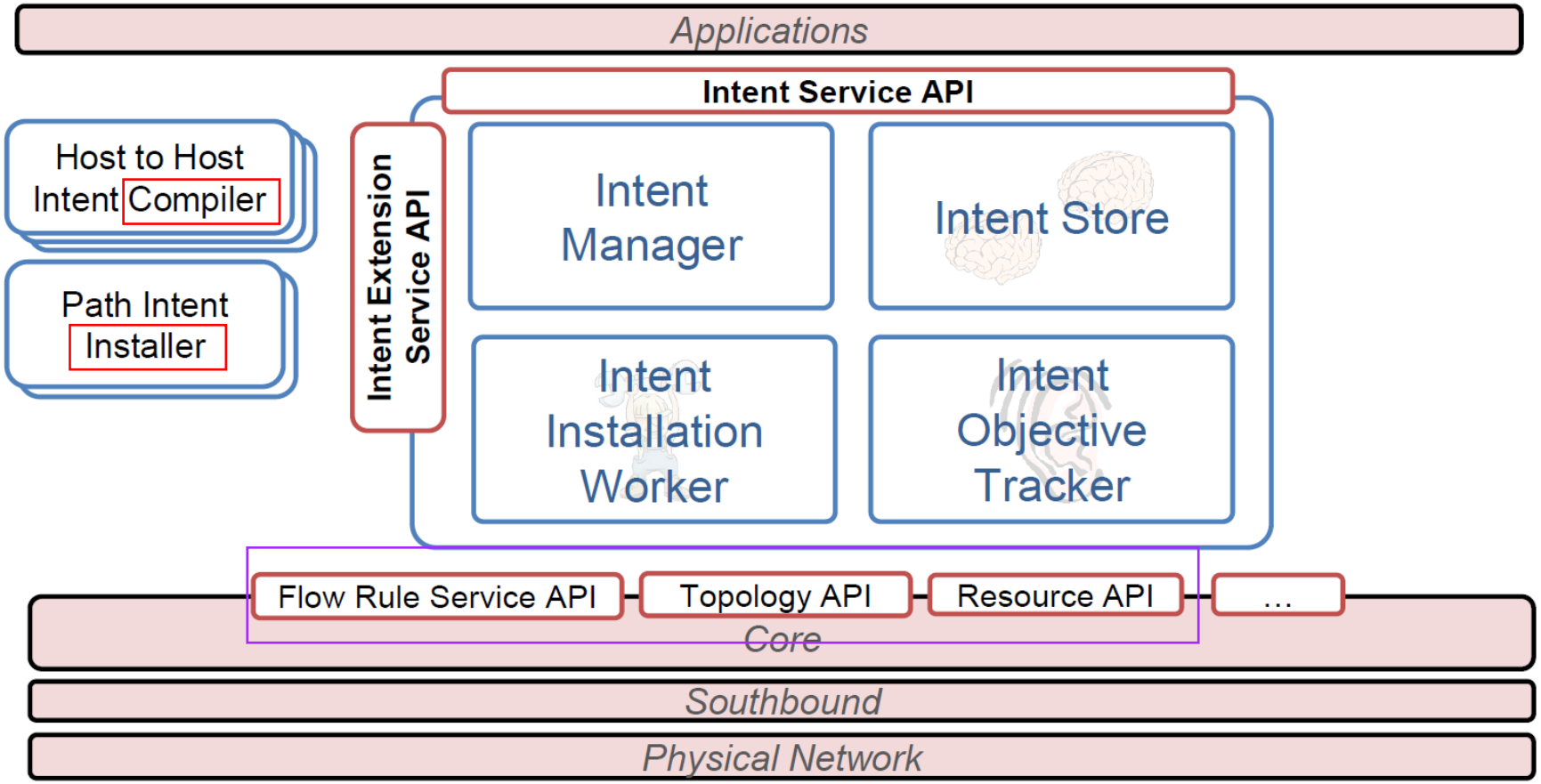


An instance is responsible for maintaining a part of network graph

Control capacity can grow with network size or application need

ONOS Intent Framework

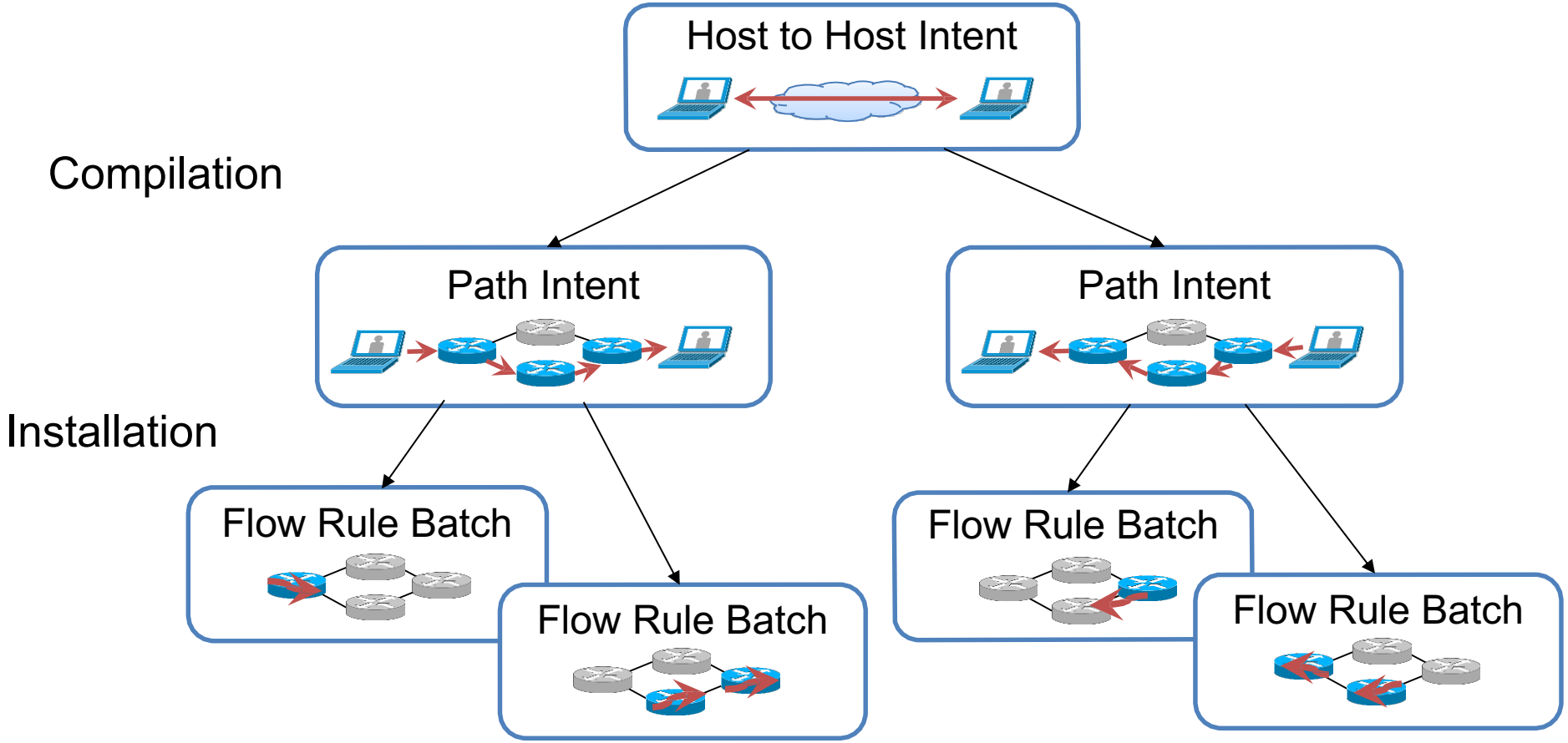
- Translates intents into device instructions



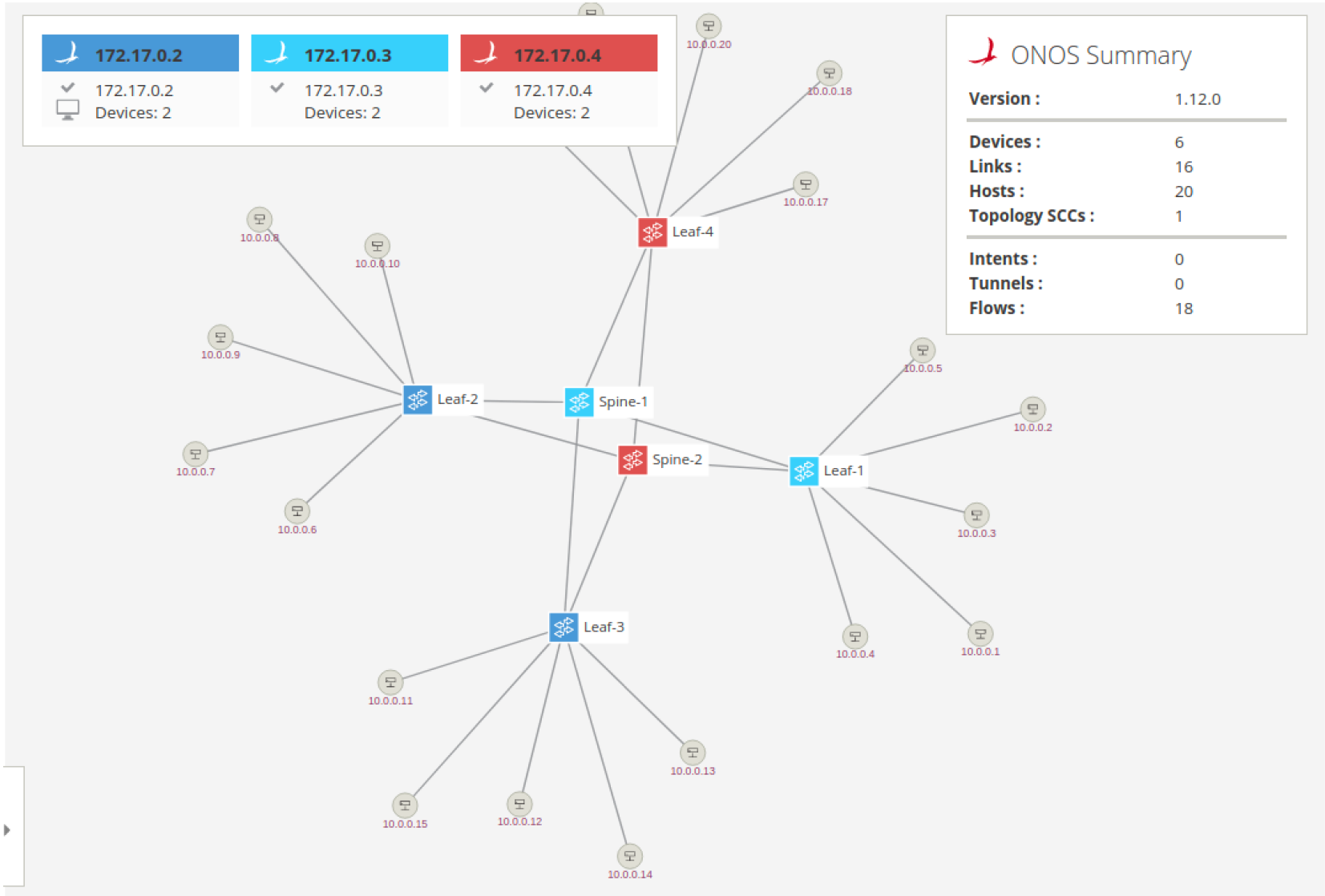
ONOS Intent Example

意图

- Compiler: produce more specific Intents given the environment
- Installer: transform Intents into device commands

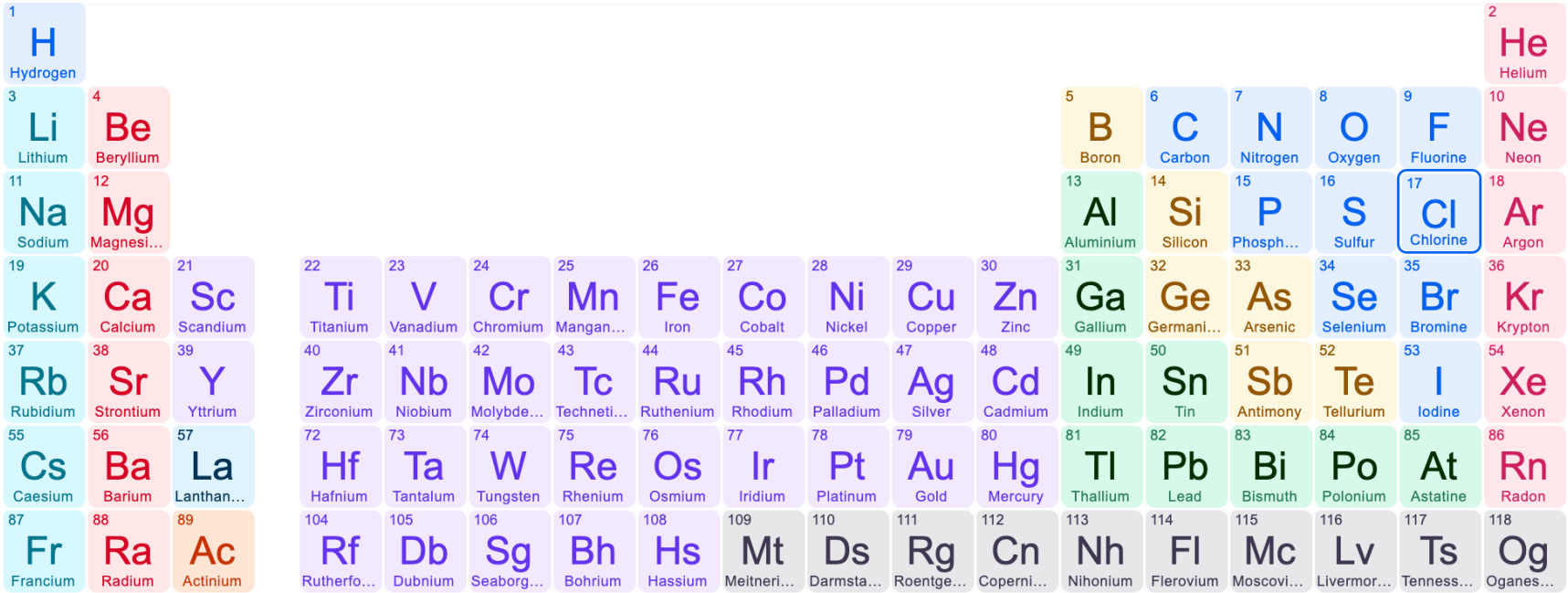


ONOS GUI


172.17.0.2
172.17.0.2
Devices: 2
172.17.0.3
172.17.0.3
Devices: 2
172.17.0.4
172.17.0.4
Devices: 2
ONOS Summary
Version : 1.12.0
Devices : 6
Links : 16
Hosts : 20
Topology SCCs : 1
Intents : 0
Tunnels : 0
Flows : 18
Leaf-4
Leaf-2
Spine-1
Spine-2
Leaf-1
Leaf-3
10.0.0.20
10.0.0.18
10.0.0.17
10.0.0.8
10.0.0.10
10.0.0.9
10.0.0.7
10.0.0.6
10.0.0.5
10.0.0.2
10.0.0.3
10.0.0.4
10.0.0.1
10.0.0.11
10.0.0.13
10.0.0.15
10.0.0.12
10.0.0.14

OpenDaylight (ODL)

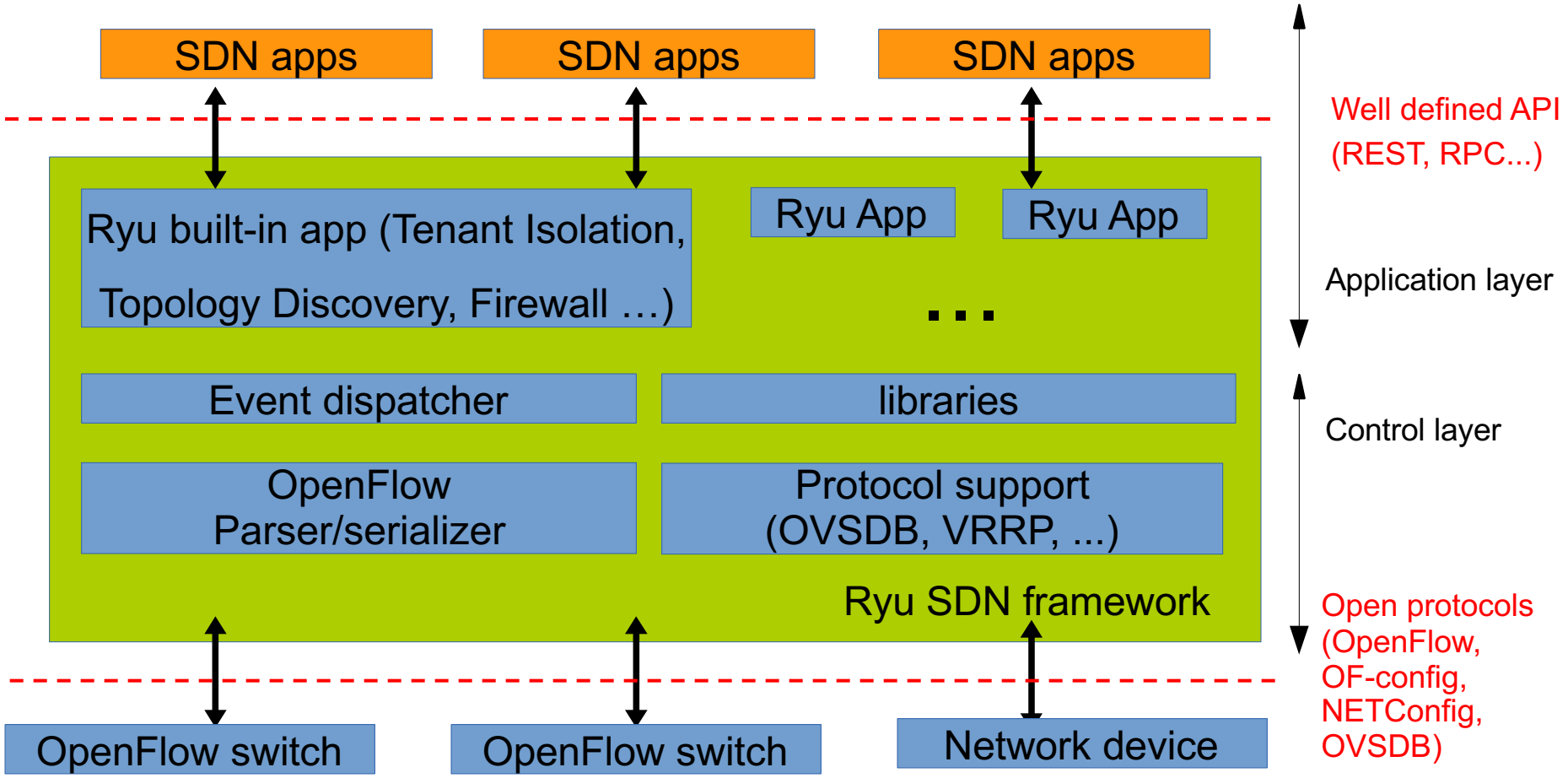
- Open-source project hosted by the Linux Foundation
  - Language: Java
  - License: Eclipse Public License 1.0
- Releases are named with chemical elements
  - Stable release: Chlorine (17) October 2022



Ryu

- Implemented with Python
- Open source, Apache 2.0 license
- Support various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc.
  - For OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions

Ryu Architecture



Comparison of SDN Controllers

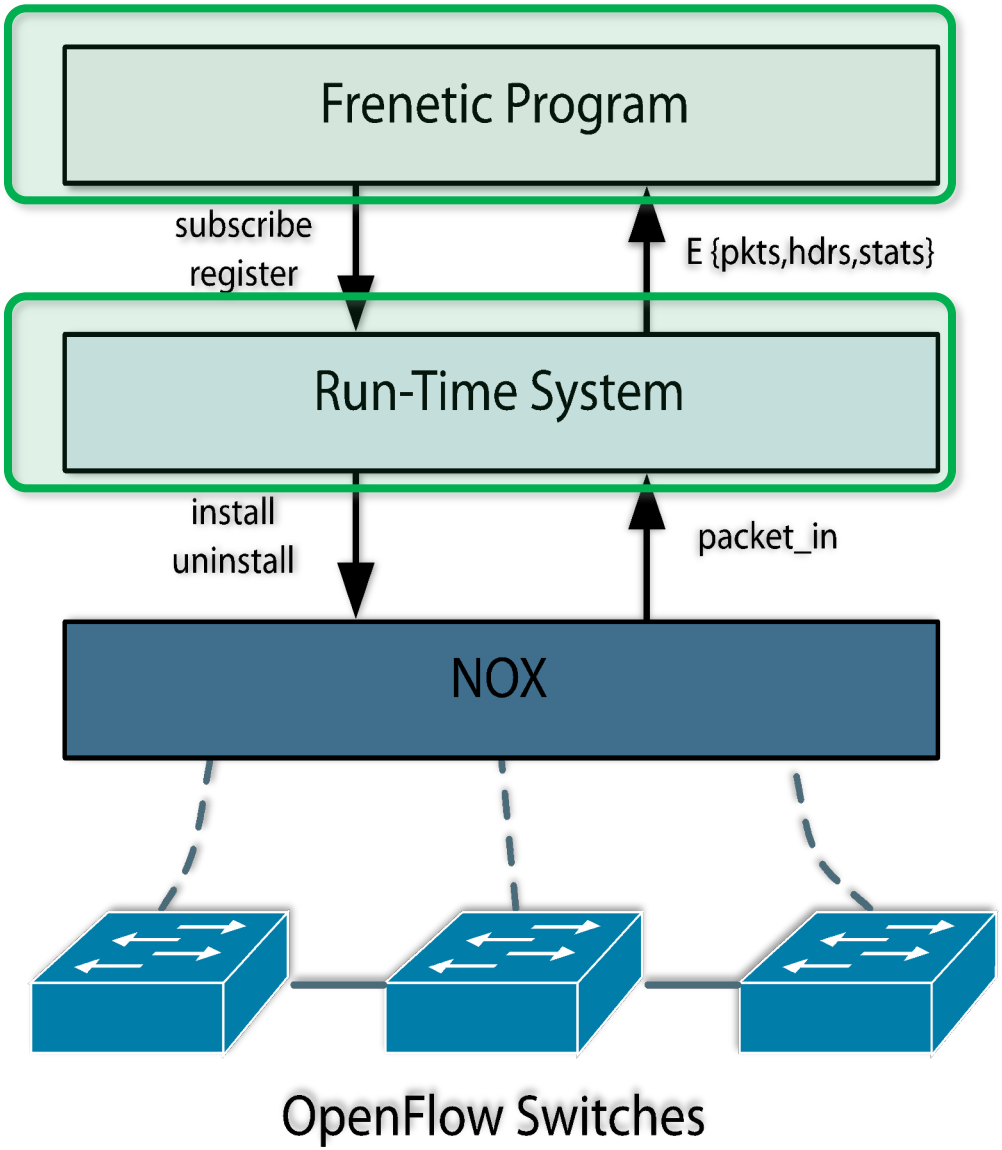
	NOX (2008)	Ryu (2012)	Floodlight (2012.1)	ONOS (2014.12)	ODL (2013.12)
Architecture	Centralized	Centralized	Centralized	Distributed	Distributed
Language	C++/Python	Python	Java	Java	Java
South Bound Interface	OpenFlow	Full	OpenFlow	Full	Full
OpenFlow	v1.0	v1. {0,2,3,4,5 } OF++	V1. {0,3 }	v1.0	v1. {0,3 }
Learning Curve	Medium	Easy	Medium	Medium	Hard
License	GPLv3	Apache 2.0	Apache	Apache 2.0	EPL v1.0
Active	28 / 6	2460 / 57	2511 / 53	4453 / 55	6384 / 60

Outline

- Control Plane Overview
- SDN Controllers
- Network Programming Languages

Frenetic

- High-level language
  - On top of NOX
  - Query language
  - Composition of forwarding policies

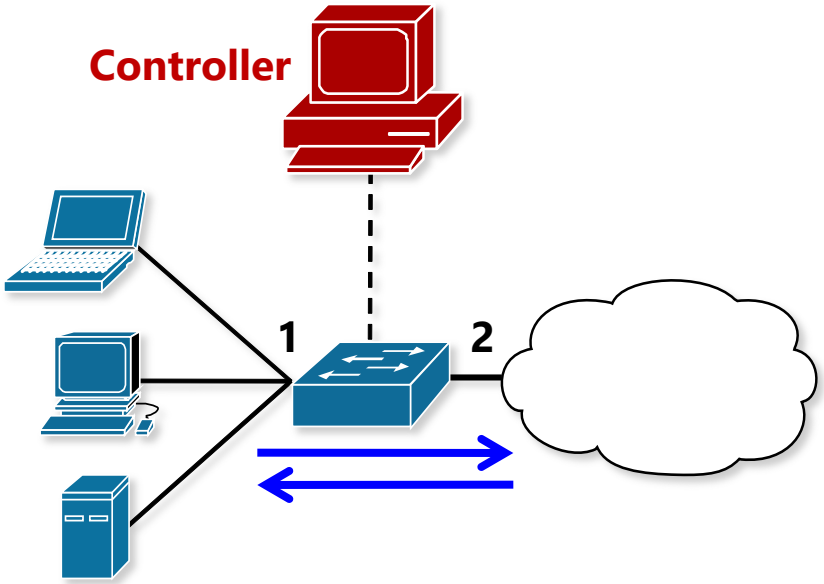


Frenetic

- Program snippet: simple repeater
  - When a switch joins the network, install two forwarding rules.

```
def switch_join(switch):
  # Repeat Port 1 to Port 2
  p1 = {in_port:1}
  a1 = [forward(2)]
  install(switch, p1, DEFAULT, a1)

  # Repeat Port 2 to Port 1
  p2 = {in_port:2}
  a2 = [forward(1)]
  install(switch, p2, DEFAULT, a2)
```



Frenetic

- Query language for traffic monitoring
  - Provide a declarative SQL-like query language for classifying and aggregating network traffic
- Program snippet: summarize the total volume of traffic arriving on physical port 2, grouped by destination host, every 60 seconds.

```
def host_query():
  return (Select(sizes) *
          Where(inport_fp(2)) *
          GroupBy([dstmac]) *
          Every(60))
```

Frenetic

- Modular Composition

```
# Static repeating between ports 1 and 2
def repeater():
 rules=[Rule(inport:1, [forward(2)]),
        Rule(inport:2, [forward(1)])]
 register(rules)
```

Repeater

```
# Monitoring Web traffic
def web_monitor():
 q = (Select(bytes) *
      Where(inport:2 & tp_src:80) *
      Every(30))
 q >> Print()
```

Monitor

Repeater + Monitor

```
# Composition of two separate modules
def main():
 repeater()
 web_monitor()
```

Pyretic

- Program snippet: a static switch

```
from pyretic.lib.corelib import *

def static_switch():
  return ((match(dstmac=EthAddr('00:00:00:01:01:01')) >> fwd(1)) +
          (match(dstmac=EthAddr('00:00:00:02:02:02')) >> fwd(2)))

def main():
  return static_switch()
```

- The Pyretic language is embedded in Python, and every program must have a *main* method and import at minimum the Pyretic core library

Merlin

- Program snippet: limit PC1 and PC2 to exchange up to 5GB of HTTP data per minute.

```
(bytes
   { location { PC1, PC2 } and tcp.src ~ 80 ,
       sliding(60) }
 ) < 5GB
```

Network Programming Languages

Language	License	Implemented in	Repository
FML [59]	—	C++, Python	—
Nettle [60]	SD-3-Clause	Haskell	https://github.com/AndreasVoellmy/nettle-openflow
Frenetic [61]	—	Python	—
Procera [62]	—	Haskell	—
Flog [63]	—	—	—
NetCore [69]	BSD-3-Clause	Haskell	https://github.com/frenetic-lang/netcore-1.0
Frenetic-Ocaml [70]	LGPL-3.0	OCaml	https://github.com/frenetic-lang/frenetic
Pyretic [53]	BSD-3-Clause	Python	https://github.com/frenetic-lang/pyretic
FlowLog v2 [85]	LGPL-3.0	OCaml	https://github.com/tnelson/FlowLog
FatTire [73]	LGPL-3.0	OCaml	https://github.com/frenetic-lang/fattire
NetKAT [71]	LGPL-3.0	OCaml	https://github.com/frenetic-lang/netkat
Merlin [57]	LGPL-3.0	OCaml	https://github.com/merlin-lang/merlin
PonderFlow [65]	—	Java	—
NoF [66]	—	Python	—
Kinetic [74]	BSD-3-Clause	Python	https://github.com/frenetic-lang/pyretic/tree/kinetic

Credits

[1] Jennifer Rexford, Course of Computer Networks, Princeton University

[2] McKeown et al., How SDN will shape networking, in Open Network Summit 2011

[3] Ben Pfaff et al., The Design and Implementation of Open vSwitch, In NSDI'15

[4] Suresh et al., Towards Programmable Enterprise WLANs With Odin, in HotSDN'12