SESSION 3

Using ONOS as the control plane

What is ONOS?

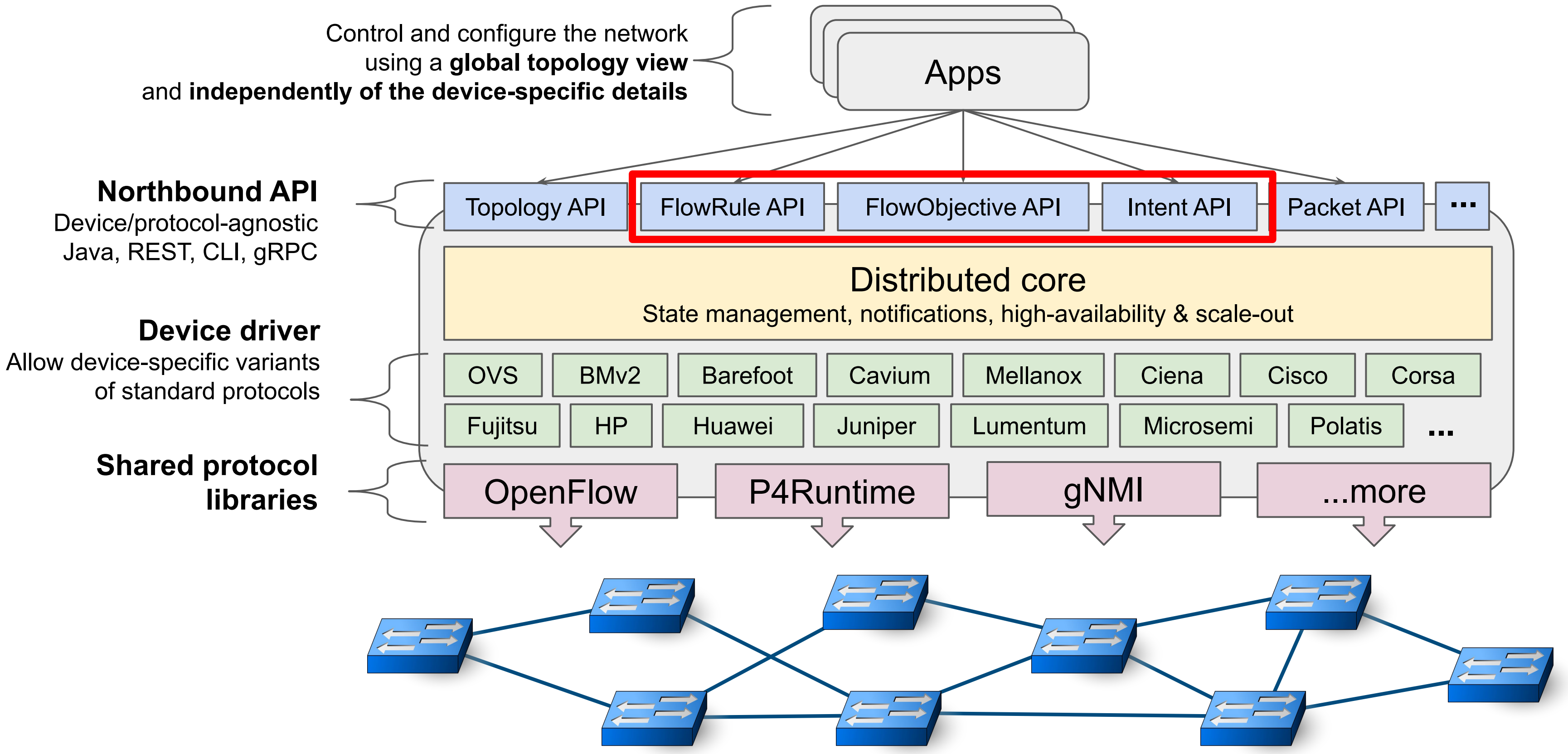
- **Open Network Operating System (ONOS)**
- **Provides the control plane for a software-defined network**
  - Logically centralized remote controller
  - Provides APIs to make it easy to create apps to control a network
- **Runs as a distributed system across many servers**
  - For scalability, high-availability, and performance
- **Focus on service provider for access/edge applications**
  - In production at scale with a major US telecom provider controlling OpenFlow devices

ONOS releases

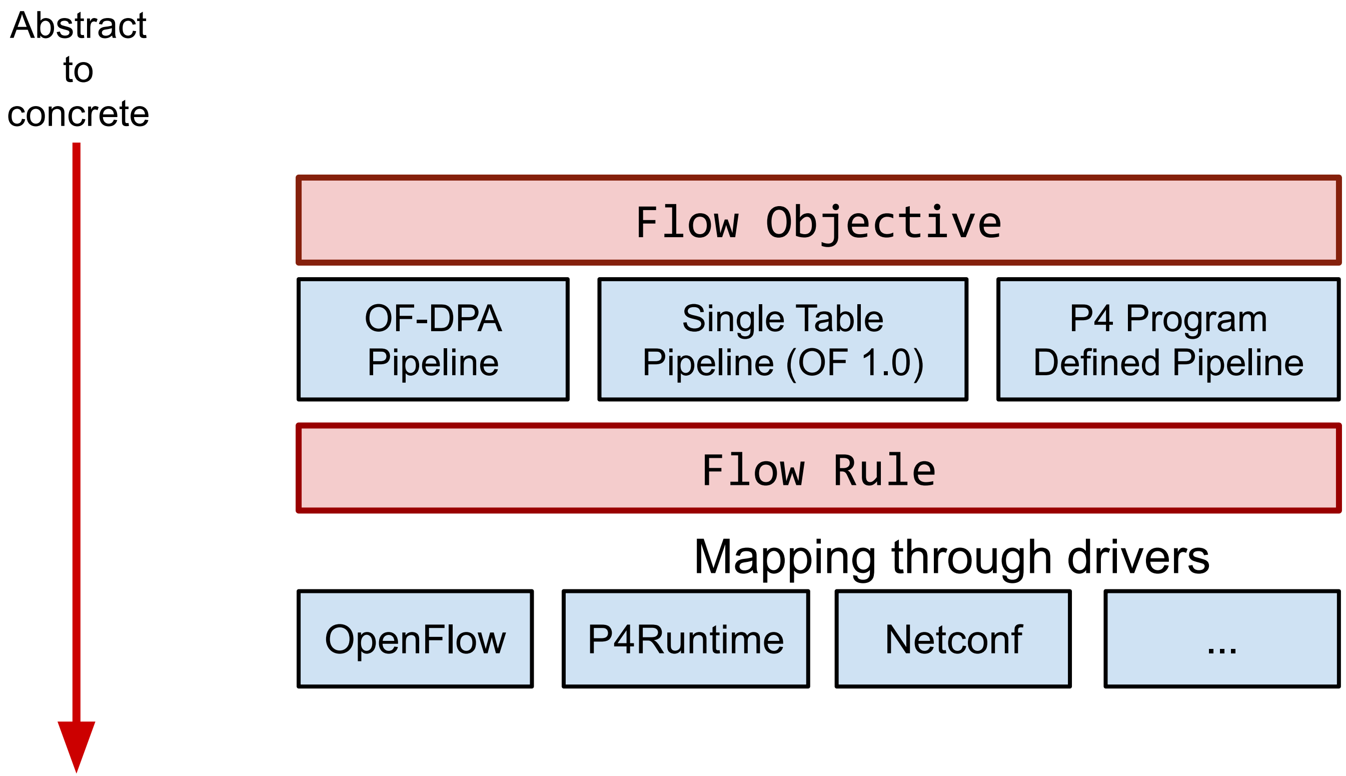
4-month release cycles

Avocet (1.0.0)	**2014-12**
…	
Loon (1.11.0)	**2017-08** *(Initial P4Runtime support)*
…	
Raven (2.2.0)	***2019-08*** *(latest - with P4Runtime, gNMI, gNOI)*

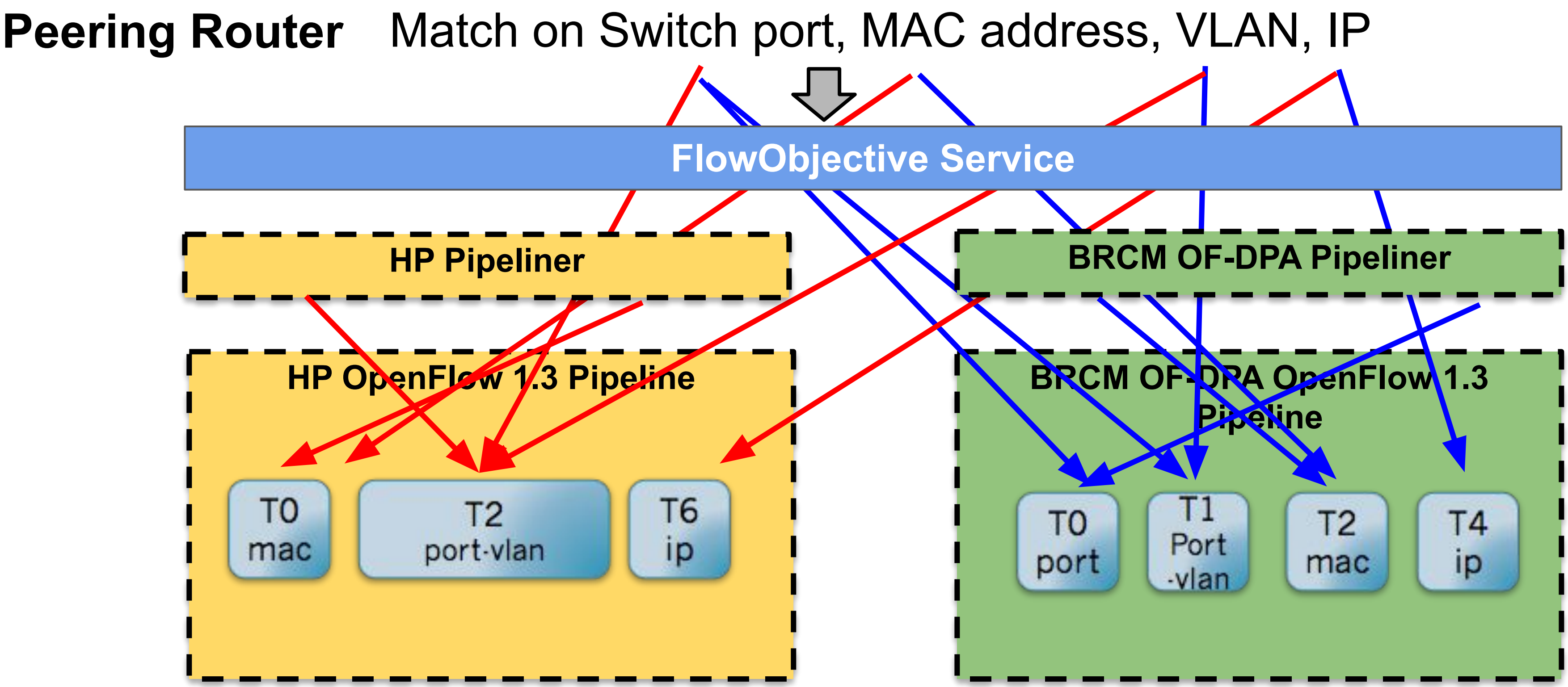
ONOS architecture



Network programming API

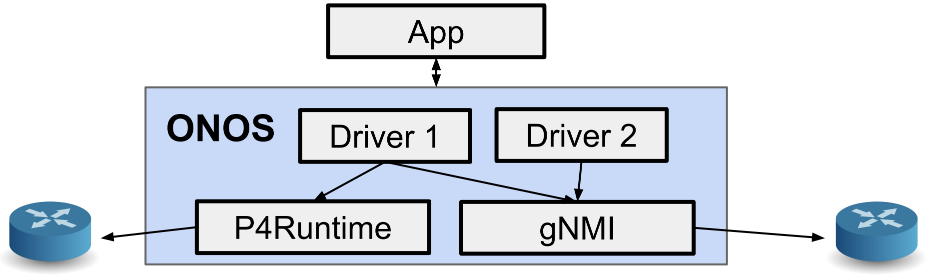


Flow objective example



Driver behaviors in ONOS

- **ONOS defines APIs to interact with device called “behaviors”**
  - `DeviceDescriptionDiscovery` → Read device information and ports
  - `FlowRuleProgrammable` → Write/read flow rules
  - `PortStatisticsDiscovery` → Statistics of device ports (e.g. packet/byte counters)
  - `Pipeliner` → FlowObjective-to-FlowRules mapping logic
  - Etc.
- **Behavior = Java interface**
- **Driver = collection of one or more behavior implementations**
  - Implementations use ONOS protocol libraries to interact with device



ONOS key takeways

- **Apps are independent from switch control protocols**
  - High level network programming APIs
  - Same app can work with OpenFlow and P4Runtime devices
- **Different network programming APIs**
  - FlowRule API – pipeline-dependent
  - FlowObjective API – pipeline-independent
    - Drivers translate 1 FlowObjective to many FlowRule
- **FlowObjective API enables application portability**
  - App using FlowObjectives can work with switches with different pipelines
  - For example, switches with different P4 programs

P4 and P4Runtime support in ONOS

P4 and P4Runtime support in ONOS

ONOS originally designed to work with OpenFlow and fixed-function switches.

Extended it to:

1. **Allow ONOS users to bring their own P4 program**
   - For example, today's tutorial
2. **Allow built-in apps to control *any* P4 pipeline without changing the app**
   - Today: topology and host discovery via packet-in / packet-out
3. **Allow new apps to control custom/new protocols as defined in the P4 program**

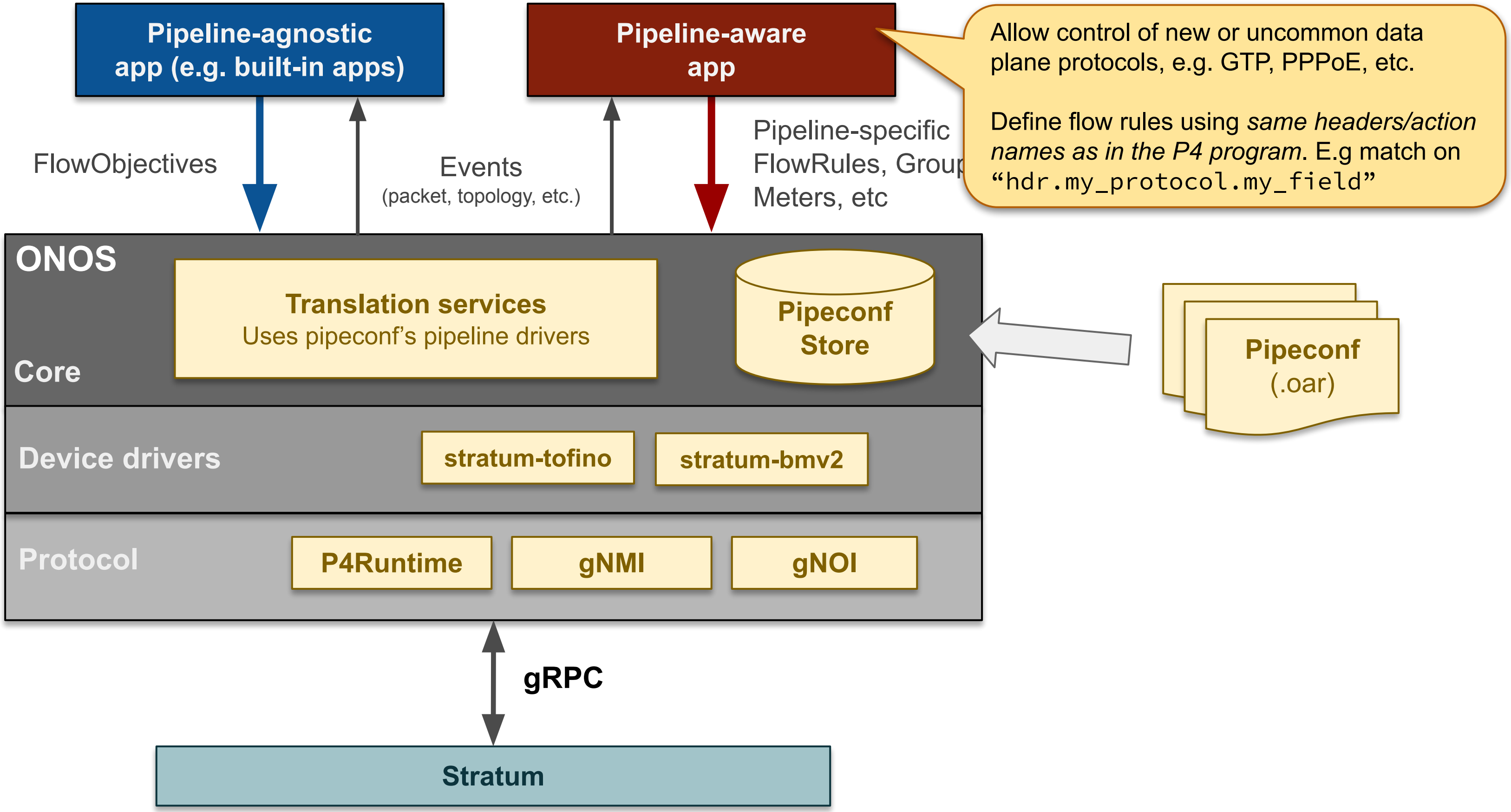
Pipeconf - Bring your own pipeline!

- **Package together everything necessary to let ONOS understand, control, and deploy an arbitrary pipeline**
- **Provided to ONOS as an app**
  - Can use `.oar` binary format for distribution

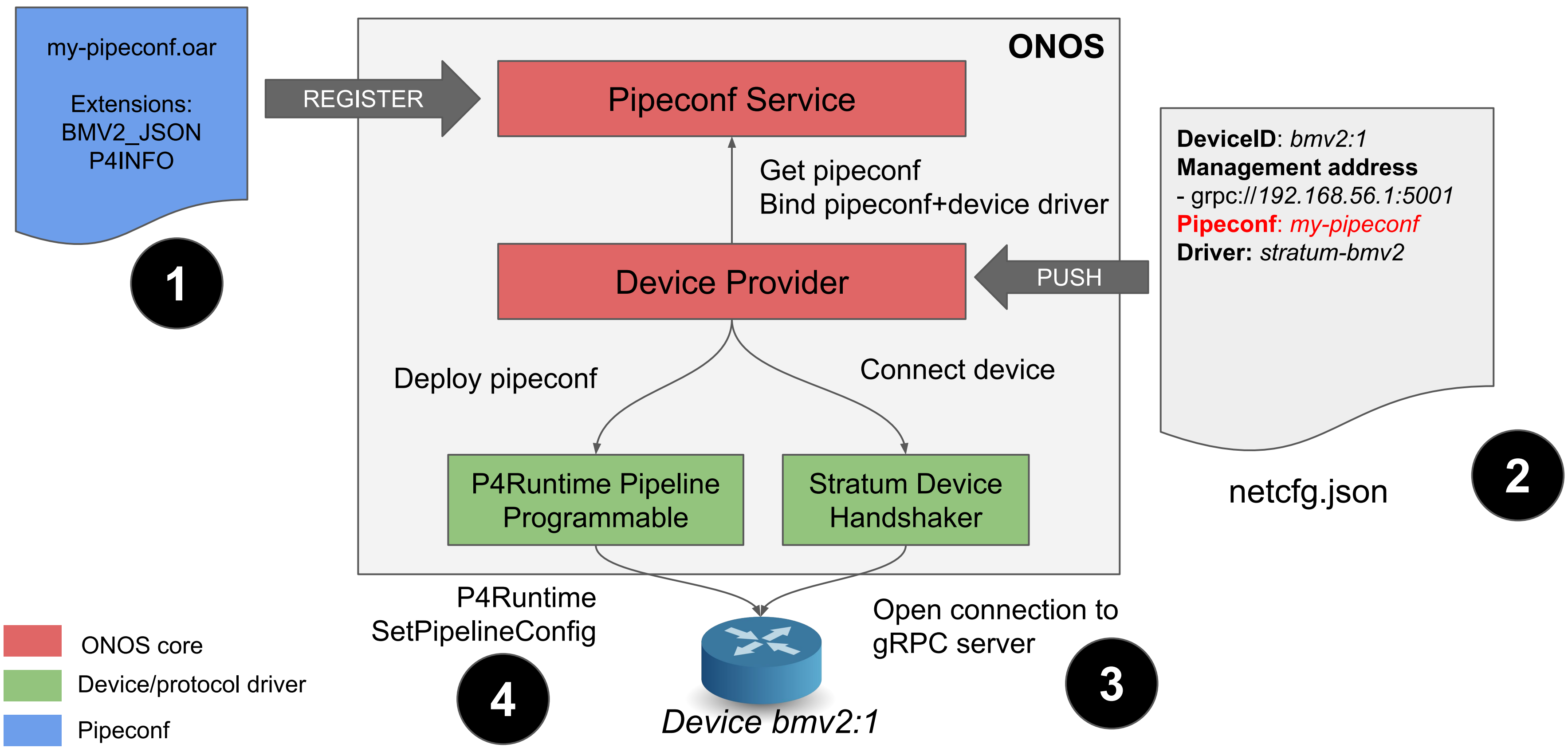
pipeconf.oar

1. **Pipeline model**
   - Description of the pipeline understood by ONOS
   - Automatically derived from P4Info
2. **Target-specific extensions to deploy pipeline to device**
   - E.g. BMv2 JSON, Tofino binary, etc.
3. **Pipeline-specific driver behaviors**
   - E.g. “Pipeliner” implementation: logic to map FlowObjectives to P4 pipeline

Pipeconf support in ONOS



Device discovery and pipeconf deploy



Flow operations

Pipeconf-based 3 phase translation:

1. Flow Objective → Flow Rule

- Maps 1 flow objective to many flow rules

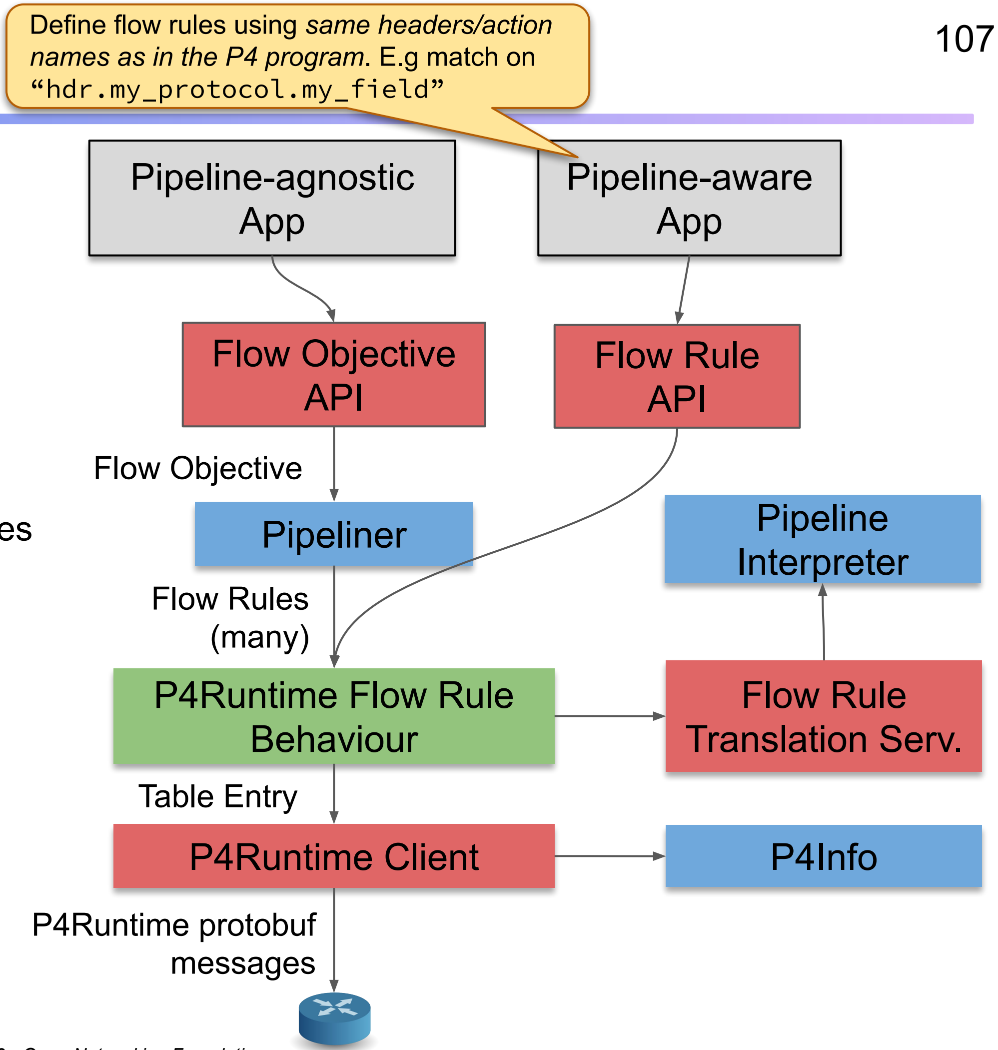
2. Flow Rule → Table entry

- Maps standard headers/actions to P4-defined ones
  E.g. ETH_DST→“`hdr.ethernet.dst_addr`”

3. Table Entry → P4Runtime message

- Maps P4 names to P4Info numeric IDs

ONOS Core
Device/protocol driver
Pipeconf driver



Pipeline Interpreter

- **Driver behavior**
- **Provides mapping between ONOS well-known types and P4 program-specific ones**

Mapping	ONOS (Java)	P4 (P4Info)
Match field	`ETH_DST` (enum)	**`“hdr.ethernet.dst_addr”`** Match field name in P4Info table definition
Packet-in	`InboundPacket.java` `.receivedFrom().port()`	**`“ingress_port”`** *Name of metadata field in P4Runtime `PacketIn` message. Defined in P4Info as `controller_packet_metadata` with name “packet_in”*
...	`...`	`...`

P4Runtime support in ONOS 2.2 (Sparrow)

P4Runtime control entity	ONOS northbound API
Table entry	Flow Rule Service, Flow Objective Service Intent Service
Packet-in/out	Packet Service
Action profile group/members, PRE multicast groups, clone sessions	Group Service
Meter	Meter Service (indirect meters only)
Counters	Flow Rule Service (direct counters)
Pipeline Config	Pipeconf

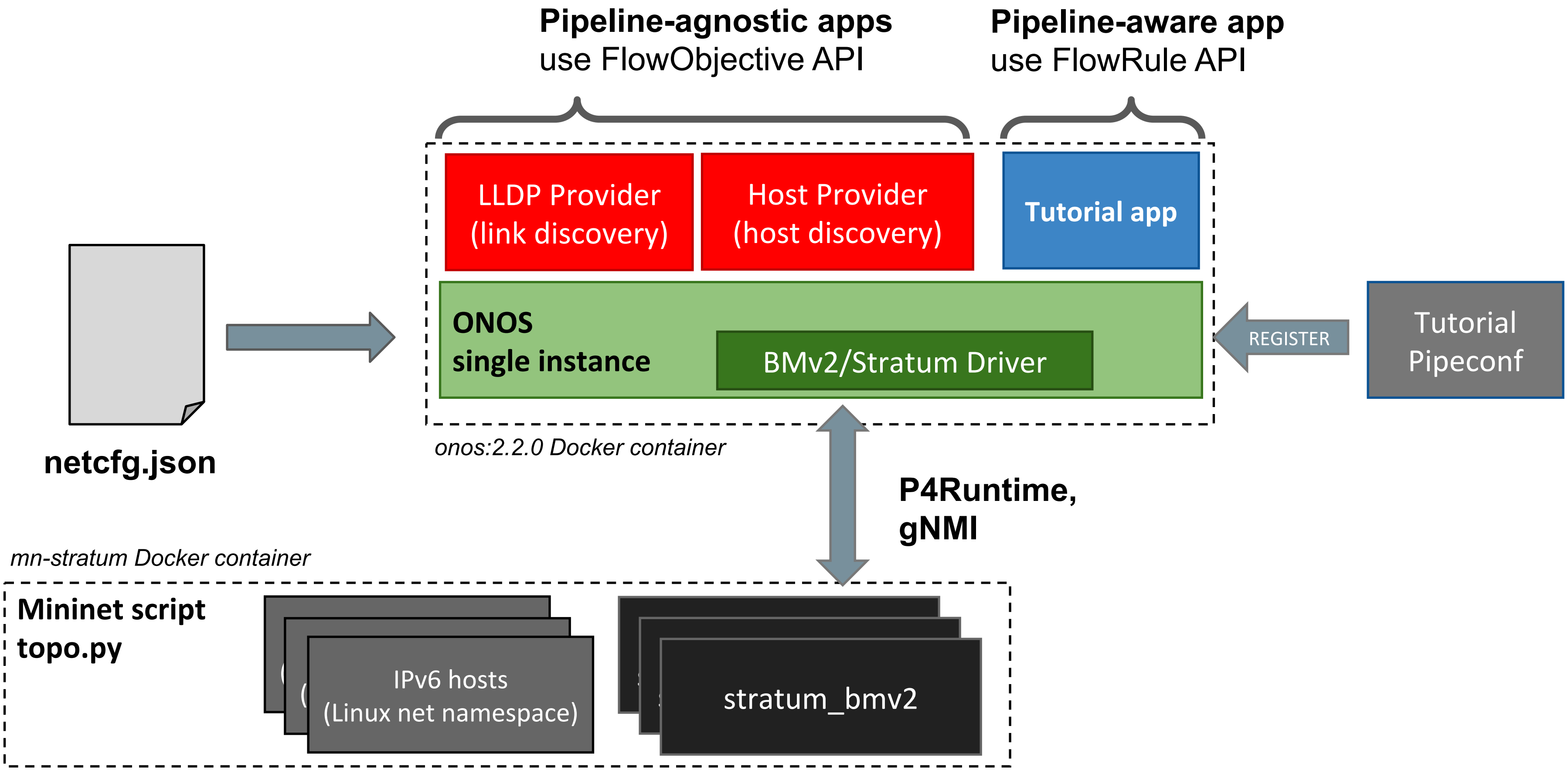
Unsupported features - community help needed!
Parser value sets, registers, digests

ONOS+P4 workflow recap

- **Write P4 program and compile it**
  - Obtain P4Info and target-specific binaries to deploy on device
- **Create pipeconf**
  - Implement pipeline-specific driver behaviours (Java):
    - Pipeliner (optional - if you need FlowObjective mapping)
    - Pipeline Interpreter (to map ONOS known headers/actions to P4 program ones)
    - Other driver behaviors that depend on pipeline
- **Use existing pipeline-agnostic apps**
  - Apps that program the network using FlowObjectives
- **Write new pipeline-aware apps**
  - Apps can use same string names of tables, headers, and actions as in the P4 program

Exercise 3 overview

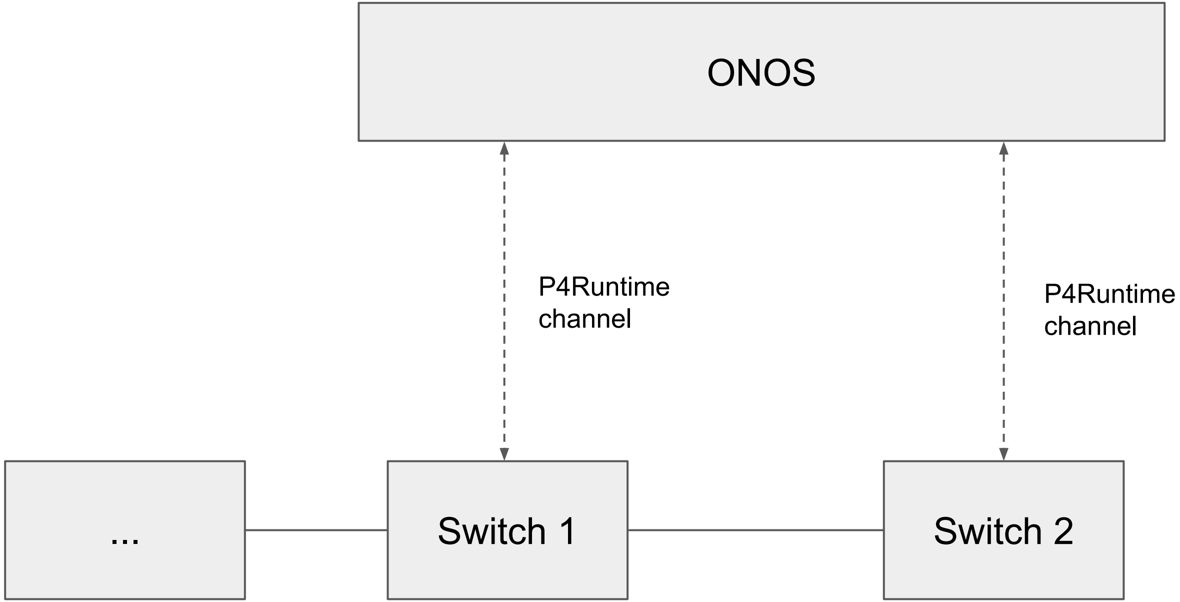
Environment



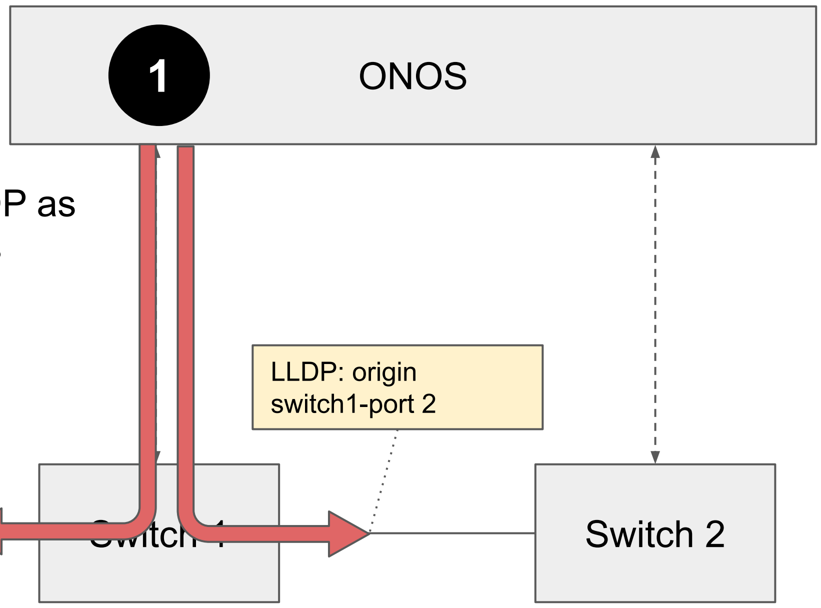
Exercise 3 steps

- **Modify pipeconf Java implementation**
  - Map P4Runtime packet-in/out to ONOS-specific representation
- **Start ONOS and Mininet**
- **Load app with pipeconf and netcfg.json**
- **Verify that link discovery works**
  - Requires both packet-in and packet-out support
- **Verify ping for hosts in the same subnet (via bridging)**
  - Requires packet-in support for host discovery

Topology discovery via packet-in

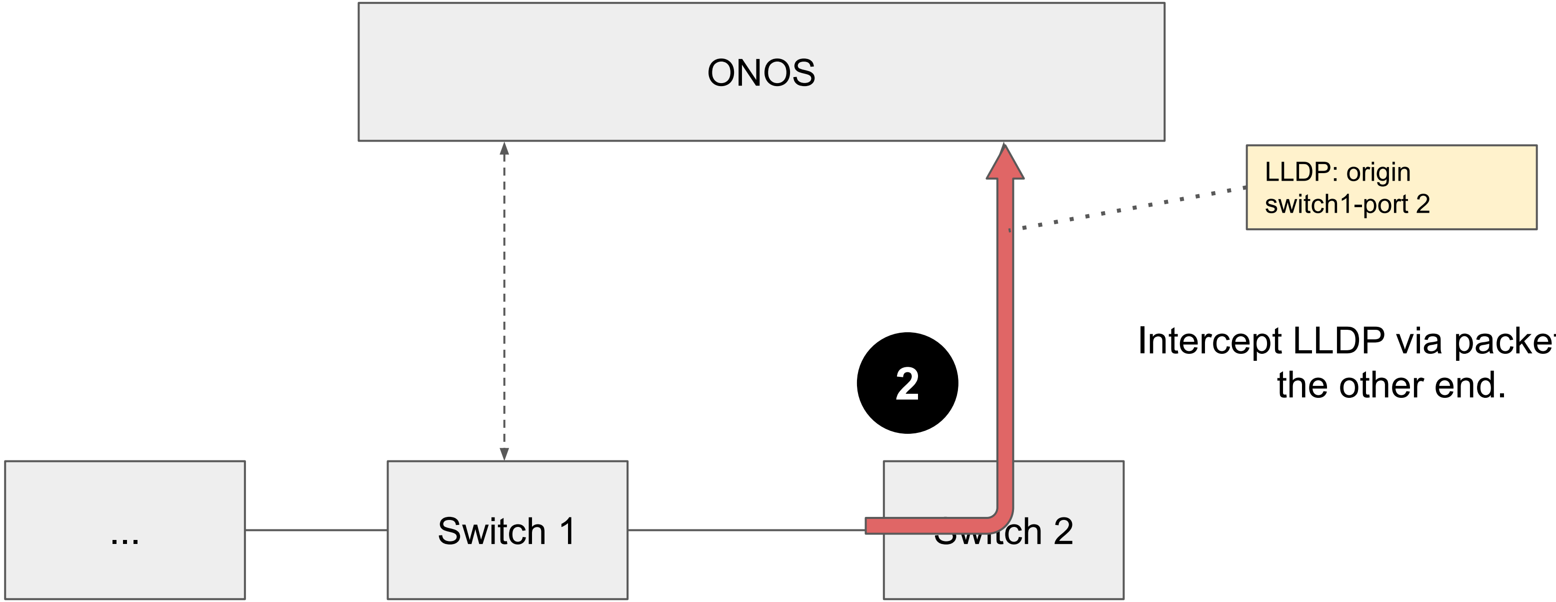


Topology discovery via packet-out/in



For each switch, send LLDP as packet-out on all ports (discovered via gNMI)

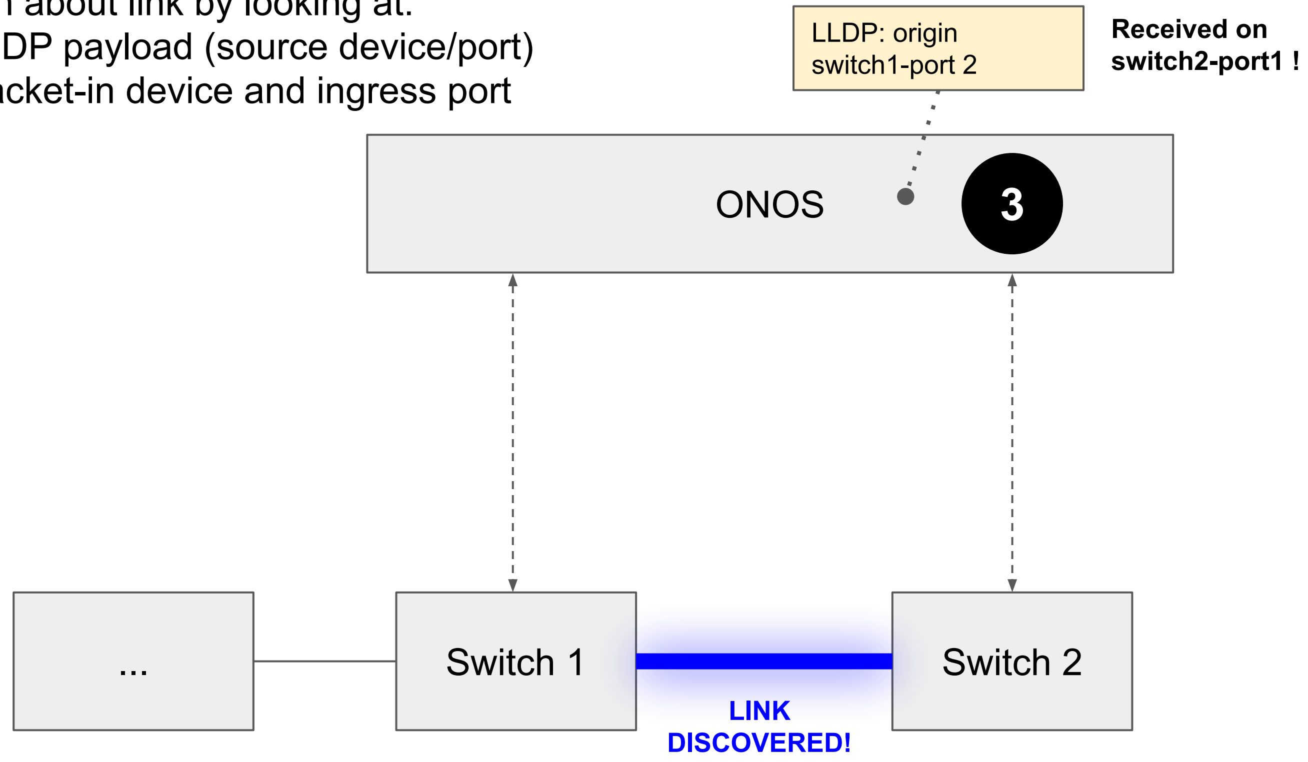
Topology discovery via packet-in



Intercept LLDP via packet-in at the other end.

Topology discovery via packet-in

Learn about link by looking at:
1. LLDP payload (source device/port)
2. Packet-in device and ingress port



LLDP Provider App

- **Automatically discover network links by injecting LLDP packets in the network**
- **Reacts to device events (e.g., new switch connection)**
  - Periodically sends LLDP packets via packet-out for each switch port
- **Install packet-in requests (flow objective) on each device**
    - Match: ETH_TYPE = LLDP, BDDP
    - Instructions: OUTPUT(CONTROLLER)

Host Provider App

- **Learns location of hosts and IP-to-MAC mapping by intercepting ARP, NDP and DHCP packets**

- **Reacts to device events (e.g., new switch connection)**
- **Install packet-in requests (flow objective) on each device**
  - Match: ARP, NDP, etc
  - Instructions: OUTPUT(CONTROLLER)

- **Parses packet-in to discover hosts**

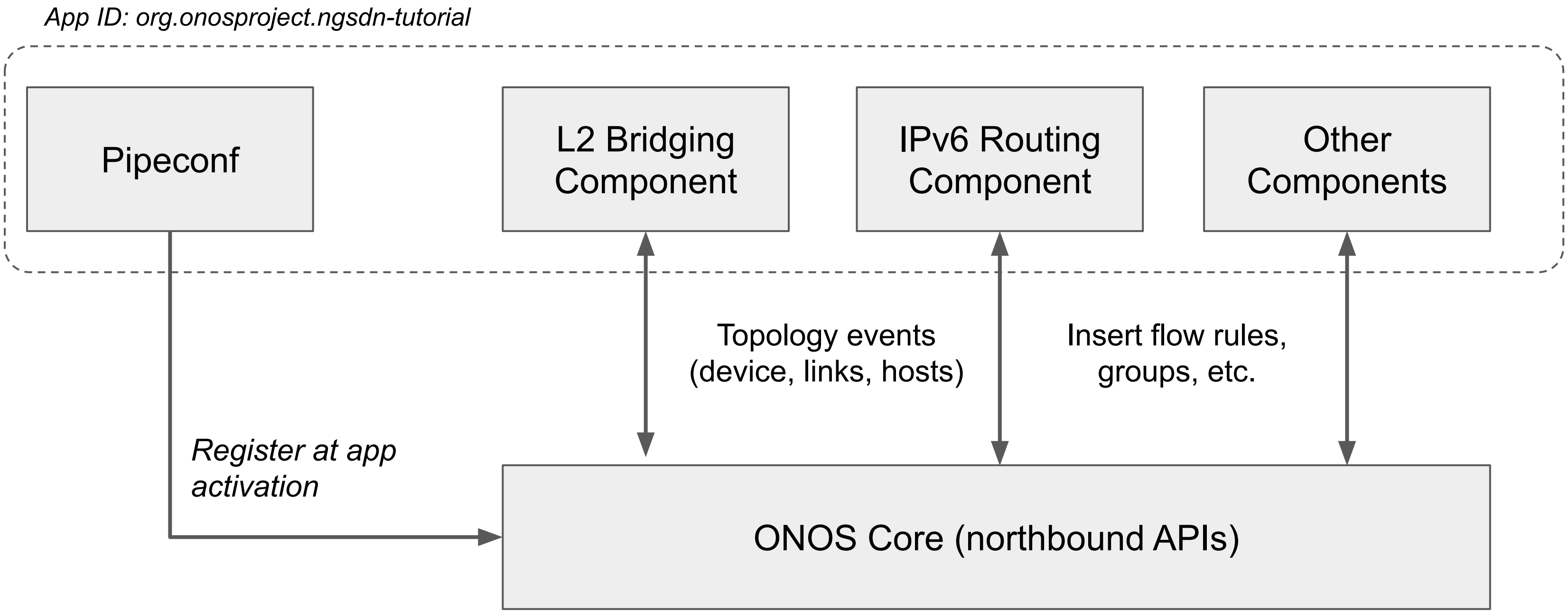
Pipeconf implementation

- **ID:** `org.onosproject.ngsdn-tutorial`
- **Driver behaviors:**
  - **Pipeliner**
    - Maps FlowObjective from LLDP and HostProvider apps
    - Use P4Runtime/v1model clone sessions to send packets to the CPU (packet-in)
  - **Interpreter**
    - Maps packet-in/out to/from ONOS internal representation
    - Maps ONOS known headers to P4Info-specific ones:
      - e.g. ETH_TYPE → “`hdr.ethernet.type`”
- **Target-specific extensions**
  - `bmv2.json, p4info.txt`

netcfg.json (devices)

```
{
  "devices": {
    "device:leaf1": {
      "basic": {
        "managementAddress": "grpc://mininet:50001?device_id=1",
        "driver": "stratum-bmv2",
        "pipeconf": "org.onosproject.ngsdn-tutorial"
      },
      "fabricDeviceConfig": {
        "myStationMac": "00:aa:00:00:00:01",
        "isSpine": false
      }
    },
    ...
```

App architecture

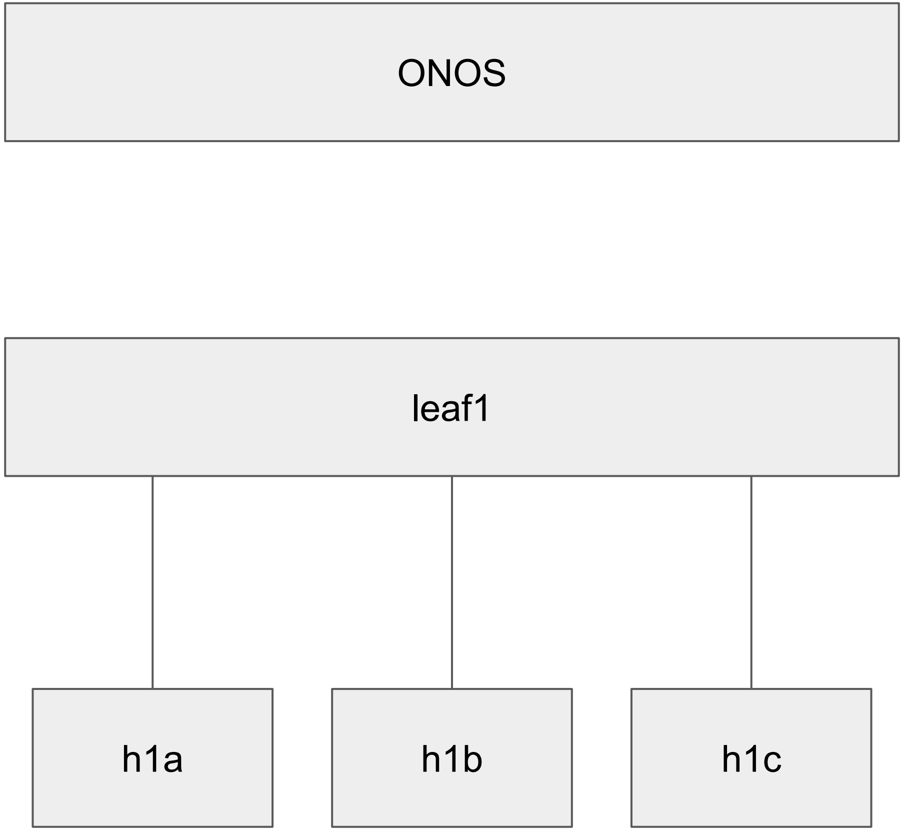


L2 bridging

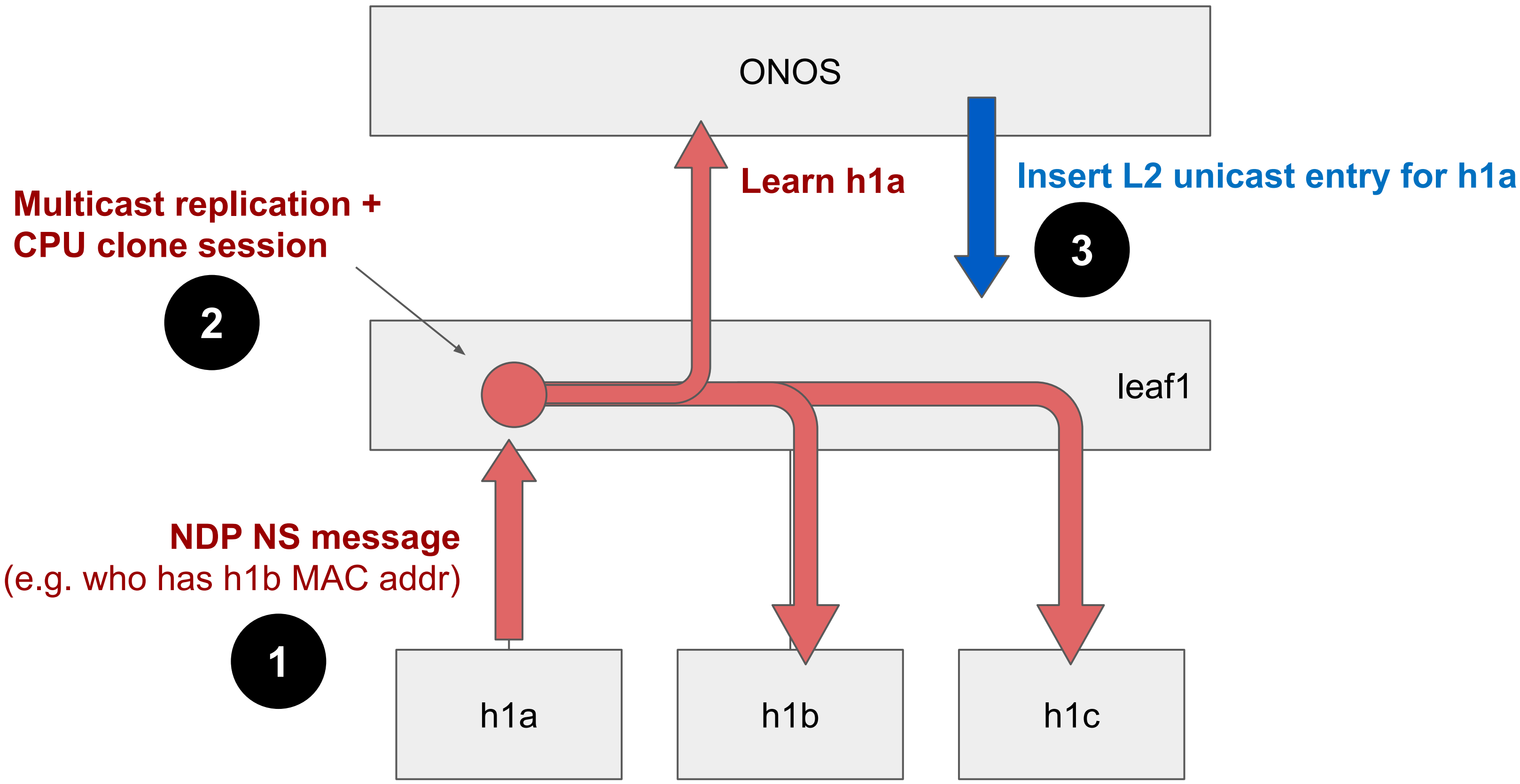
Leaf switches should provide bridging for hosts in the same subnet:

- **Hosts send NDP Neighbor Solicitation (NS) requests to resolve the MAC address of other hosts**
- **NDP NS packets are replicated (multicast) to all host-facing ports**
- **Other host replies with NDP Neighbor Advertisement (NA)**
- **ONOS learns about hosts by requesting a clone of all NDP NA/NS packets as packet-ins (hostprovider built-in app)**
- **For each learned host, app installs a flow rule to forward packets for the host MAC destination (bridging table)**

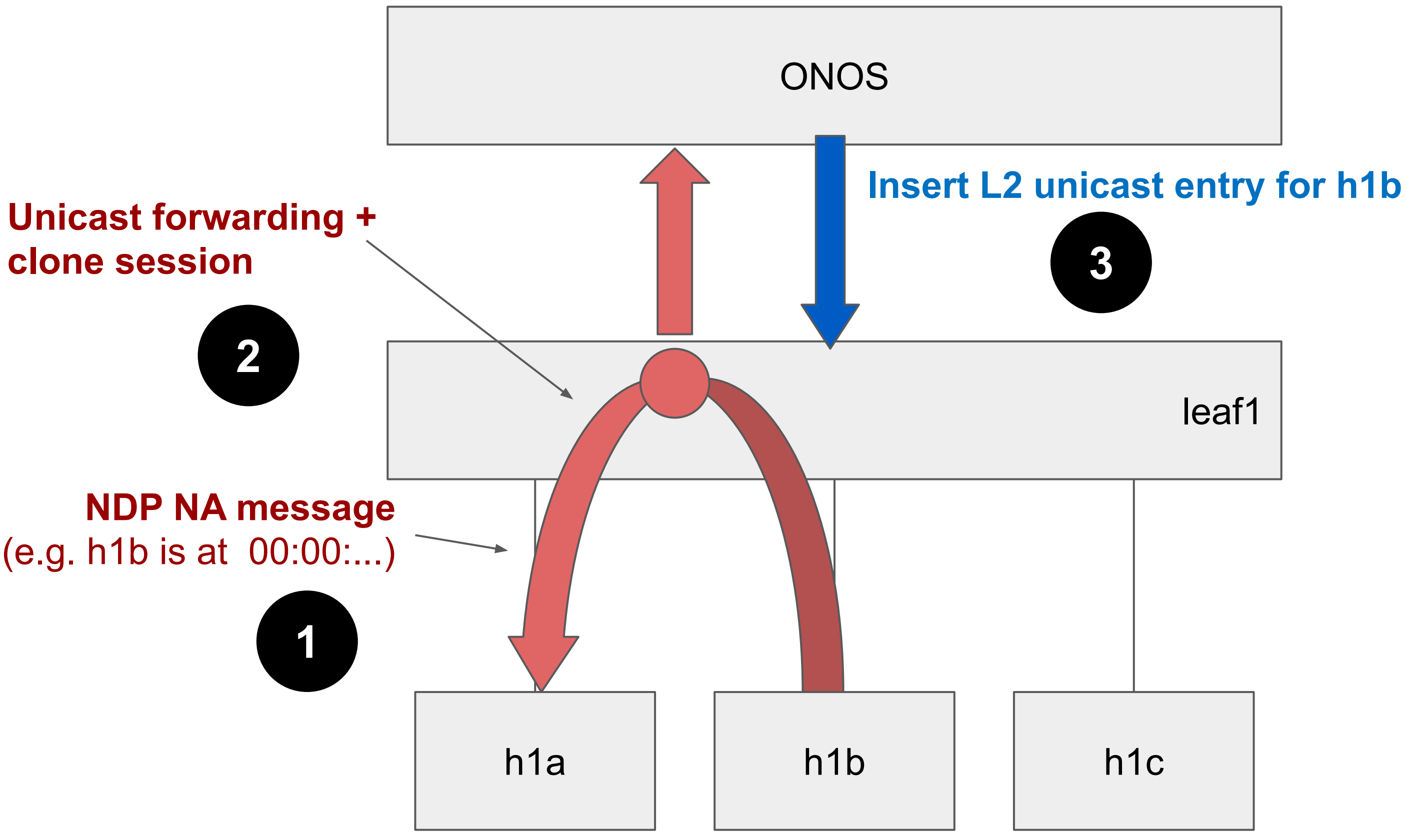
How is bridging implemented?



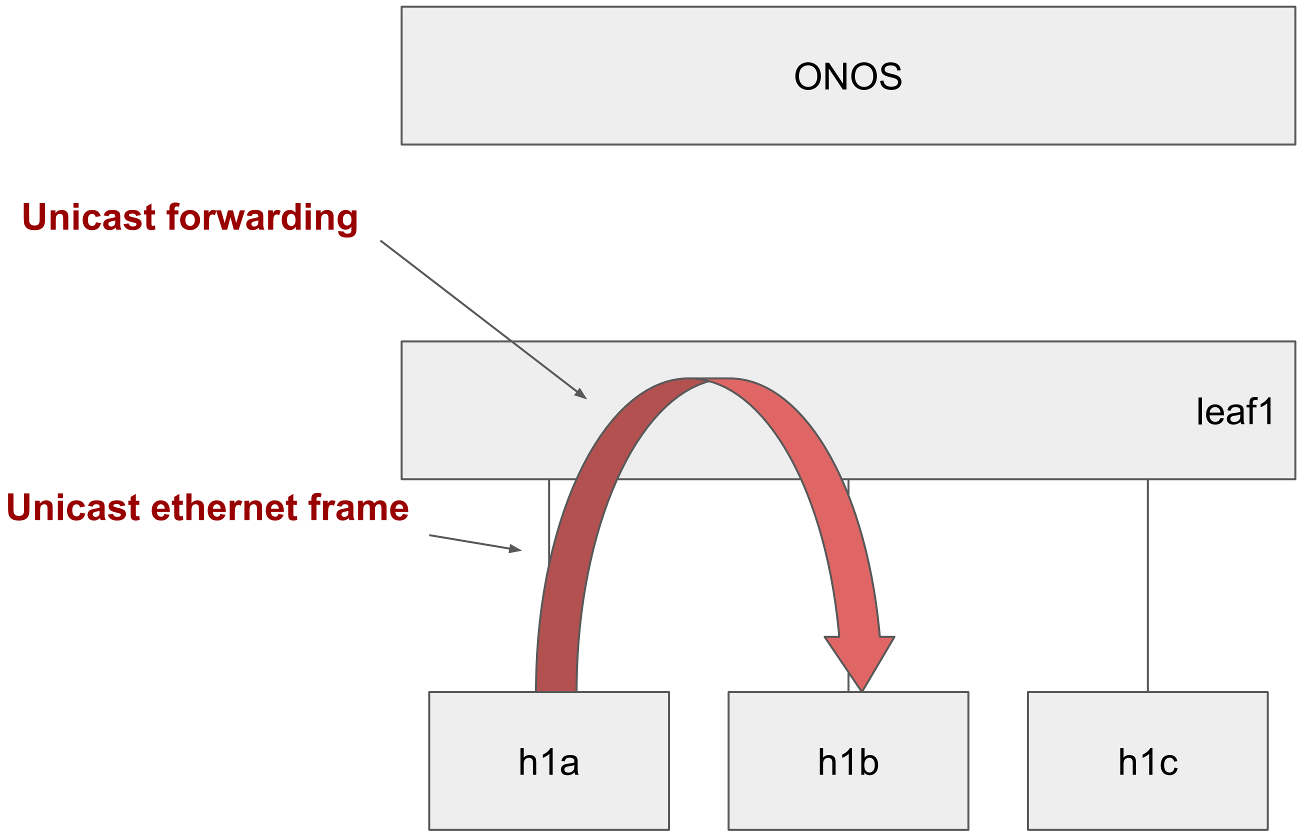
Host discovery (NDP NS)



Host discovery (NDP NA)



Unicast forwarding



L2BridgingComponent.java

- **Listens to device and topology events**
- **For each device, install:**
    - Flow rule and group to replicate NDP NS to all host-facing ports (`l2_ternary_table`)
    - Flow rule to intercept NDP NS/NA (ACL table)
    - Flow rule with L2 forwarding rule for each learned host (`l2_exact_table`)
- **Support bridging only for hosts attached to the same leaf**
- **Looks at topology to derive multicast group with host-facing ports, no need to replicate NDP NS towards spines**

ONOS terminology

- **Criteria**
  - Match fields used in a FlowRule
- **Treatment**
  - Actions/instructions of a FlowRule
- **Pi* classes**
  - Classes used to describe entities similar to P4Runtime ones
  - PI stands for protocol-independent
  - Examples
    - **`PiTableId:`** name of a table as in the P4 program
    - **`PiMatchFieldId:`** name of a match filed in a table
    - **`PiCriterion`**: match fields each one defined by its name and value
    - **`PiAction`**: action defined by its name and list of parameters

Exercise 3: Get Started

These slides: `http://bit.ly/ngsdn-tutorial-slides`

Open:

`~/ngsdn-tutorial/README.md`

`~/ngsdn-tutorial/EXERCISE-3.md`

Or use GitHub markdown preview:

`http://bit.ly/ngsdn-tutorial-lab`

Solution:

`~/ngsdn-tutorial/solution`

Before starting!
Update tutorial repo
(requires Internet access)

```
cd ~/ngsdn-tutorial
git pull origin master
make pull-deps
```

P4 language cheat sheet:
`http://bit.ly/p4-cs`

You can work on your own using the instructions.
Ask for instructors help when needed.

Packet-in/out metadata

```
controller_packet_metadata {
  preamble {
    id: 67135753
    name: "packet_out"
    alias: "packet_out"
    annotations: "@controller_header(\"packet_out\")"
  }
  metadata {
    id: 1
    name: "egress_port"
    bitwidth: 9
  }
  metadata {
    id: 2
    name: "_pad"
    bitwidth: 7
  }
}
```