Software Defined Networking (2/2)

Xuetao Wei
weixt@sustech.edu.cn

Outline

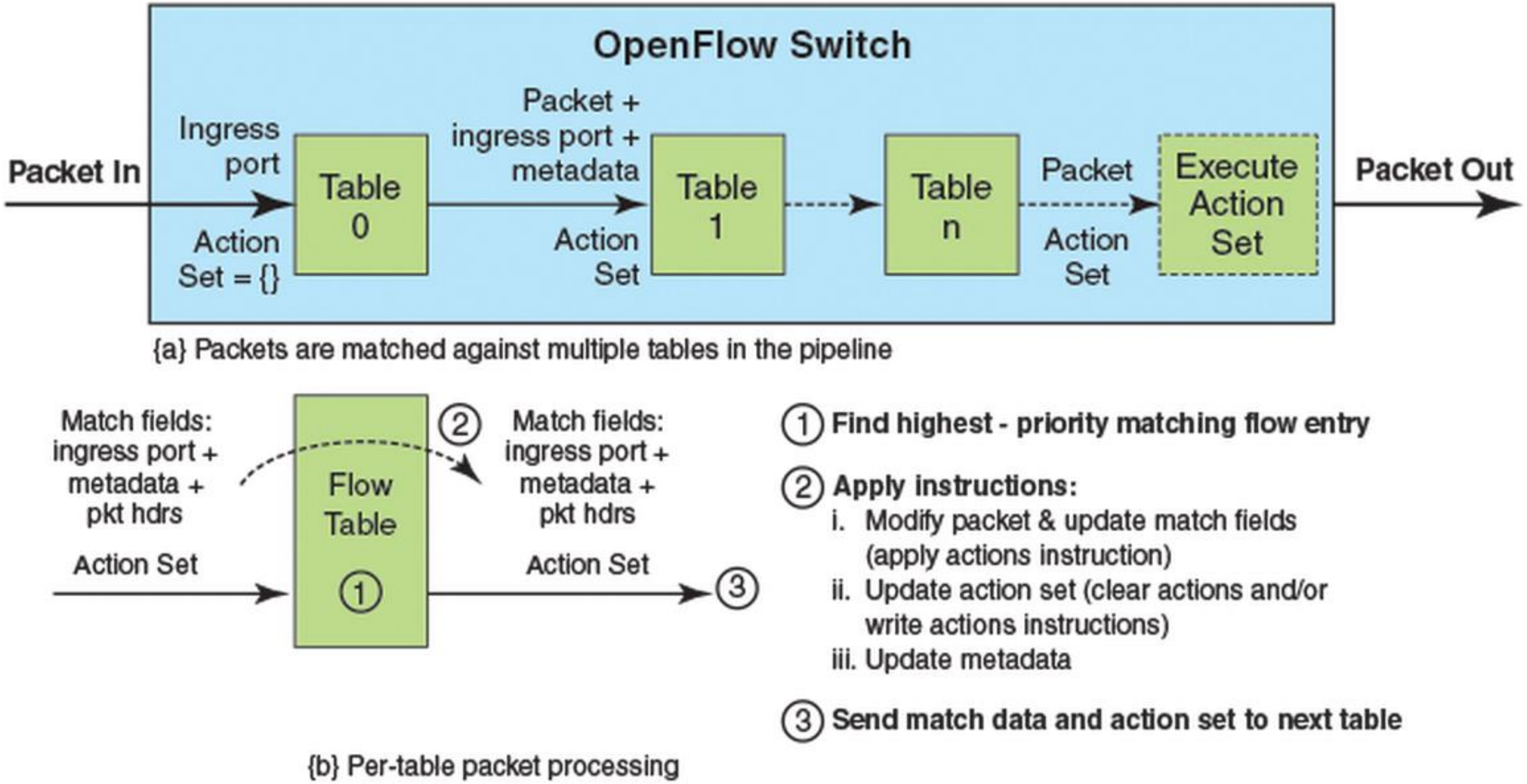
- **Open vSwitch (OVS)**
- Towards OpenFlow 2.0
- New Forwarding Plane Architectures
- Programming the Forwarding Plane

Where It's Going

- **OF v1.x**
  - multiple tables: leverage additional tables
  - tags and tunnels
  - multipath forwarding
  - per flow meters

- **OF v2+** (yet to come)
  - generalized matching and actions: protocol independent forwarding

Multiple Tables (OF v1.x)

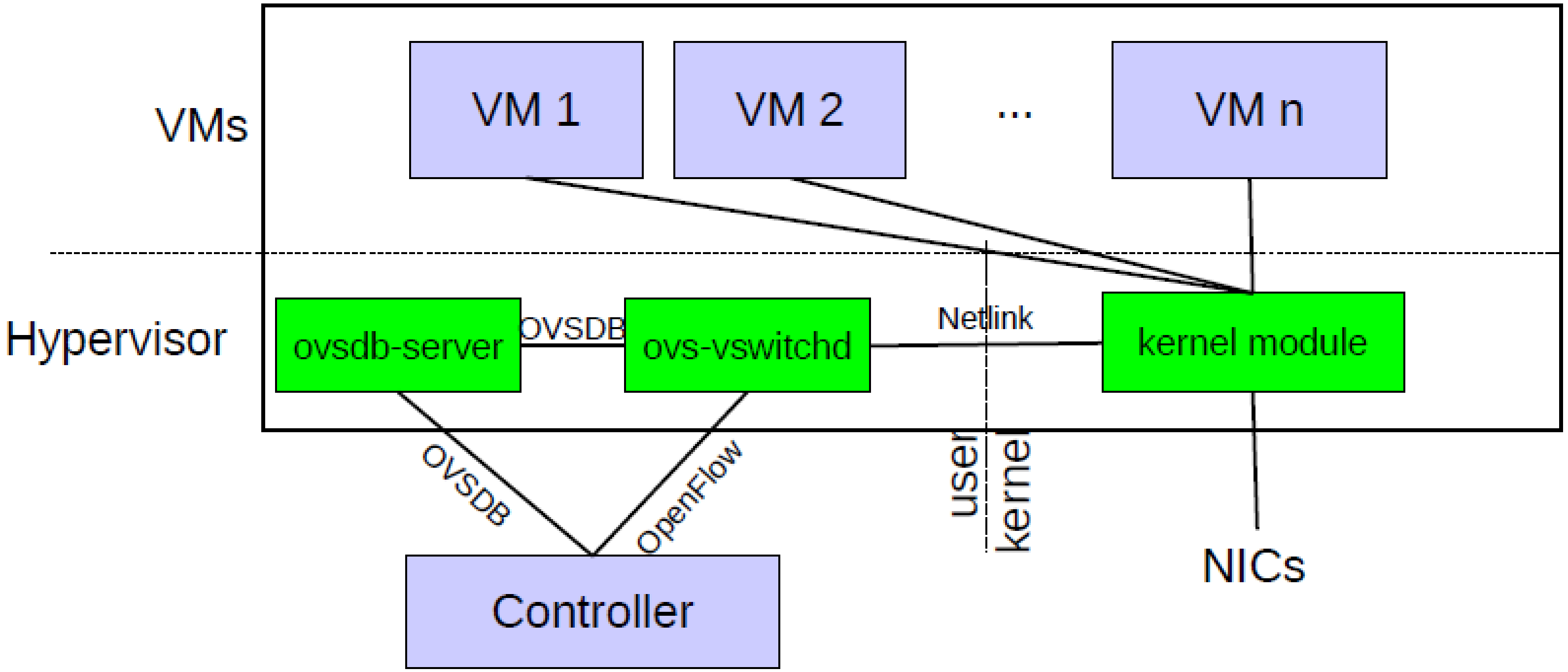


Open vSwitch (OVS)

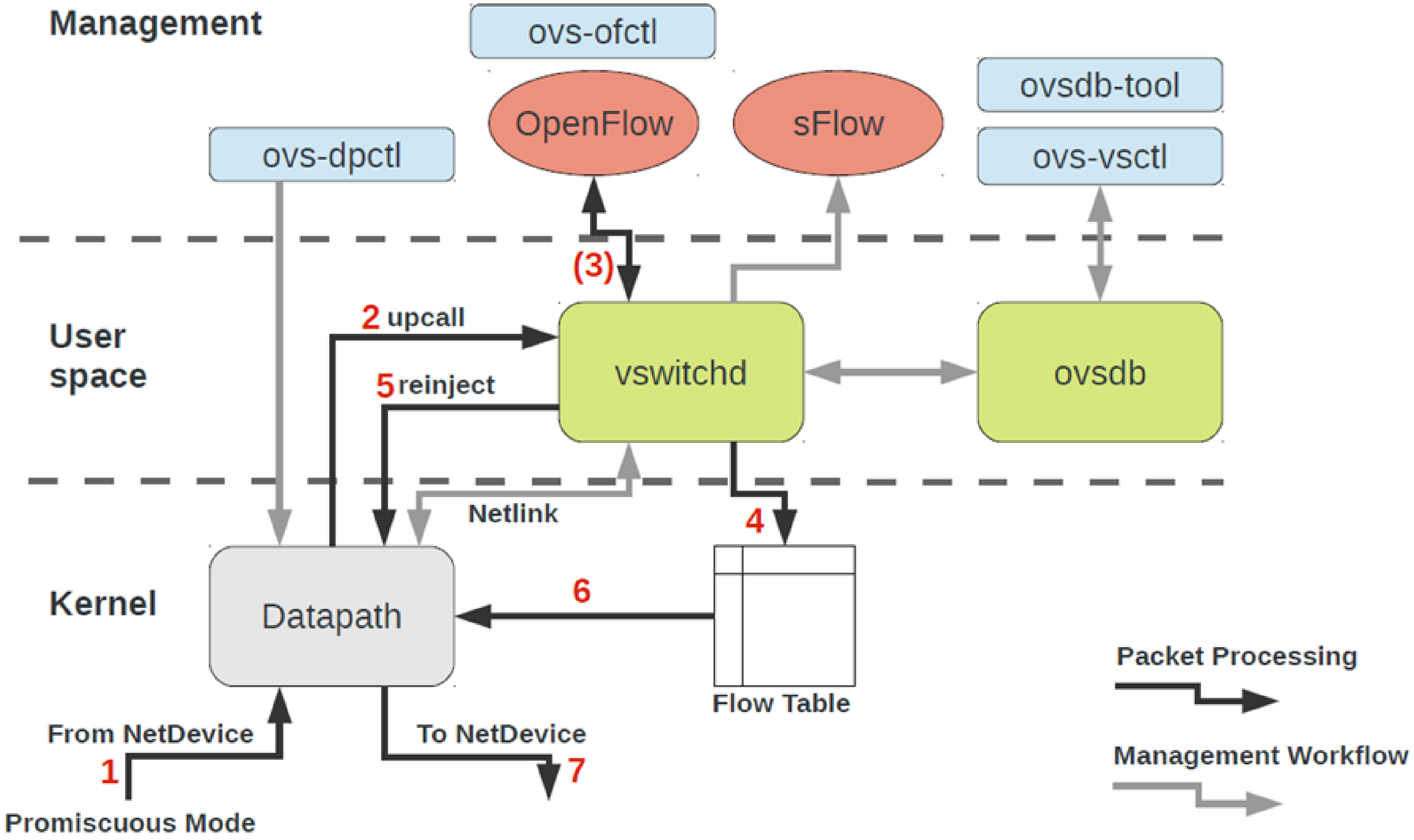
- A virtual switch conforming to OpenFlow standard that is implemented in software
- Available from *openvswitch.org*
- Development code is available in git
- User-space (controller and tools) is under the Apache license
- Kernel (datapath) is under the GPLv2

OVS Architecture

- **ovs-vswitchd**: core implementation of the switch
- **ovsdb-server**: manipulates the database of the vswitch configuration & flows



OVS Workflow



Usage (1/4)

- Check OpenFlow version supported by OVS

```
$ ovs-ofctl --version
ovs-ofctl (Open vSwitch) 1.11.
Compiled Oct 28 2013 14:17:17
OpenFlow versions 0x1:0x4
```

- Create a new OVS switch

```
$ ovs-vsctl add-br ovs-switch
```

- Create a new port p0 with port number 100

```
$ ovs-vsctl add-port ovs-switch p0 -- set Interface p0 ofport_request=100
```

Usage (2/4)

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Check the switch created

```
$ ovs-ofctl show ovs-switch
OFPT_FEATURES_REPLY (xid=0x2): dpid:00001232a
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STA
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STR
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC S
 100(p0): addr:54:01:00:00:00:00
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 101(p1): addr:54:01:00:00:00:00
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 102(p2): addr:54:01:00:00:00:00
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(ovs-switch): addr:12:32:a2:37:ea:45
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal
```

Usage (3/4)

- Deny all packets

```
$ ovs-ofctl add-flow ovs-switch "table=0, dl_src=01:00:00:00:00:00/01:00:00:00:00:00, actions=drop"
```

- Dump all entries in flow tables

```
ovs-ofctl dump-flows ovs-switch
```

- Change source address to 9.181.137.1 for all packets received on port p0

```
$ ovs-ofctl add-flow ovs-switch "priority=1 idle_timeout=0,\
    in_port=100,actions=mod_nw_src:9.181.137.1,normal"
```

Usage (4/4)

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Sending testing packet from port p0 (192.168.1.100) to port p1 (192.168.1.101)

```
$ ip netns exec ns0 ping 192.168.1.101
```

- Monitoring received packets at port p1, found out that its source address has been changed to 9.181.137.1

```
$ ip netns exec ns3 tcpdump -i p2 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on p2, link-type EN10MB (Ethernet), capture size 65535 bytes
16:07:35.677770 IP 192.168.1.100 > 192.168.1.101: ICMP echo request, id 23147, seq 25, length 64
16:07:36.685824 IP 192.168.1.100 > 192.168.1.101: ICMP echo request, id 23147, seq 26, length 64
```

Using Controller for OVS

- **Install floodlight**
  - http://www.projectfloodlight.org/getting-started/
  - git clone git://github.com/floodlight/floodlight.git
  - cd floodlight/
  - ant
  - nohup java -jar target/floodlight.jar > floodlight.log 2>&1 &

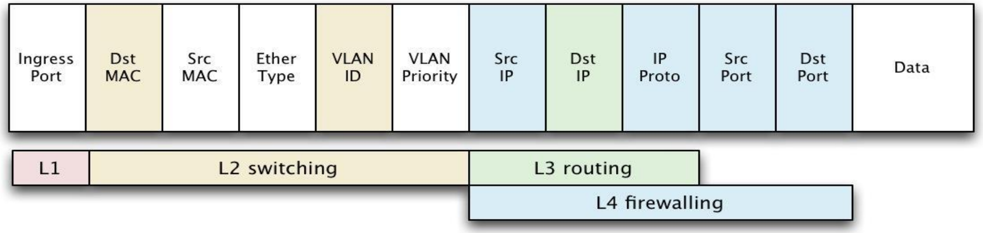
- **Set Controller**
  - ovs-vsctl set-controller ubuntu_br tcp:192.168.100.1:6633

Outline

- Open vSwitch (OVS)
- **Towards OpenFlow 2.0**
- New Forwarding Plane Architectures
- Programming the Forwarding Plane

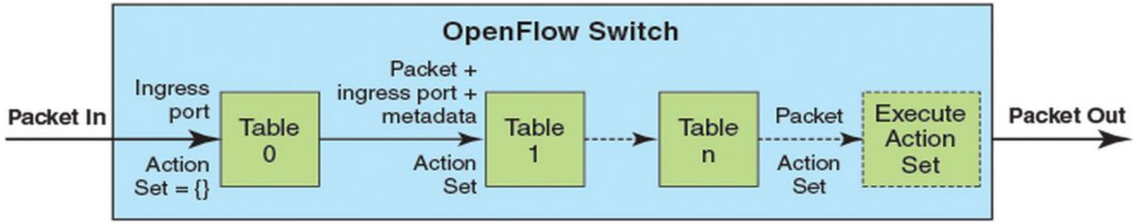
Evolving of OpenFlow

- OpenFlow 1.0 (2008)
  - Single table with 12 fields



Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

- OpenFlow 1.1 ~ 1.4 (2011 ~ 2013)
  - Multiple tables in pipelined processing
  - The number of fields increases from 12 to 40+



Evolving of OpenFlow

- **OpenFlow 2.0** (2013~)
  - Match & Action on any #fields, any combination of fields, and even any sequences of header bits
- **PIF**: Protocol Independent Forwarding (2013)
  - Stanford, Princeton
  - Coupled integrated with switch architecture and programming language
  - More towards ASIC implementations
- **POF**: Protocol Oblivious Forwarding (2013)
  - Huawei
  - Without specific target switch architecture in mind
  - More towards CPU/NPU implementations

POF: Protocol Oblivious Forwarding

- Current OpenFlow-enabled Device
  - Decoupled control/data planes, “dumb” packet forwarding devices
  - Centralized control of network, intelligent controllers
  - Multiple flow tables

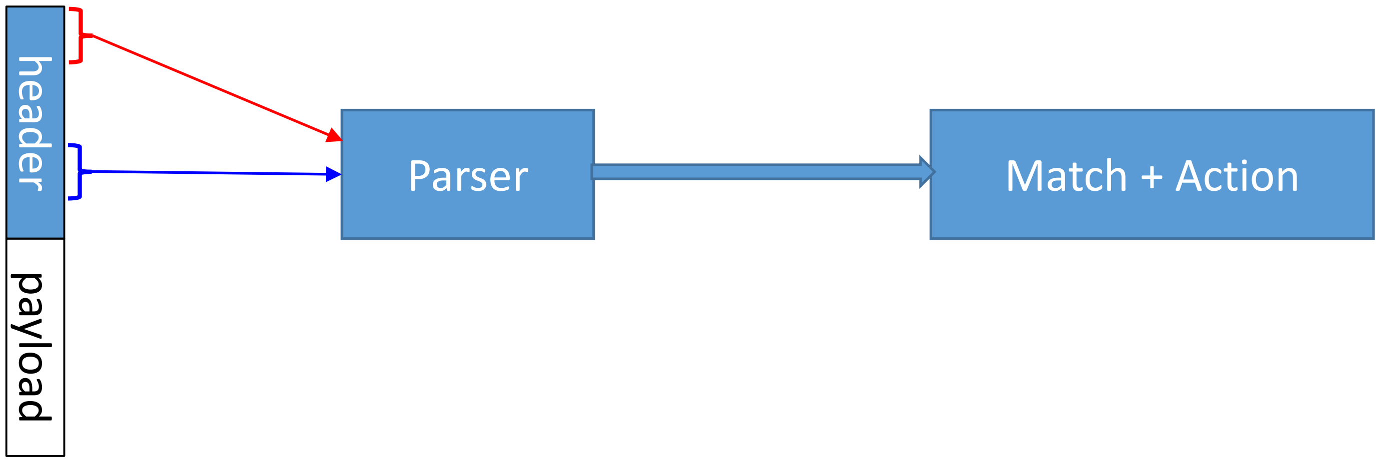
  But...

- Still protocol conscious packet forwarding

Read:
Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane ***[HotSDN'13]***

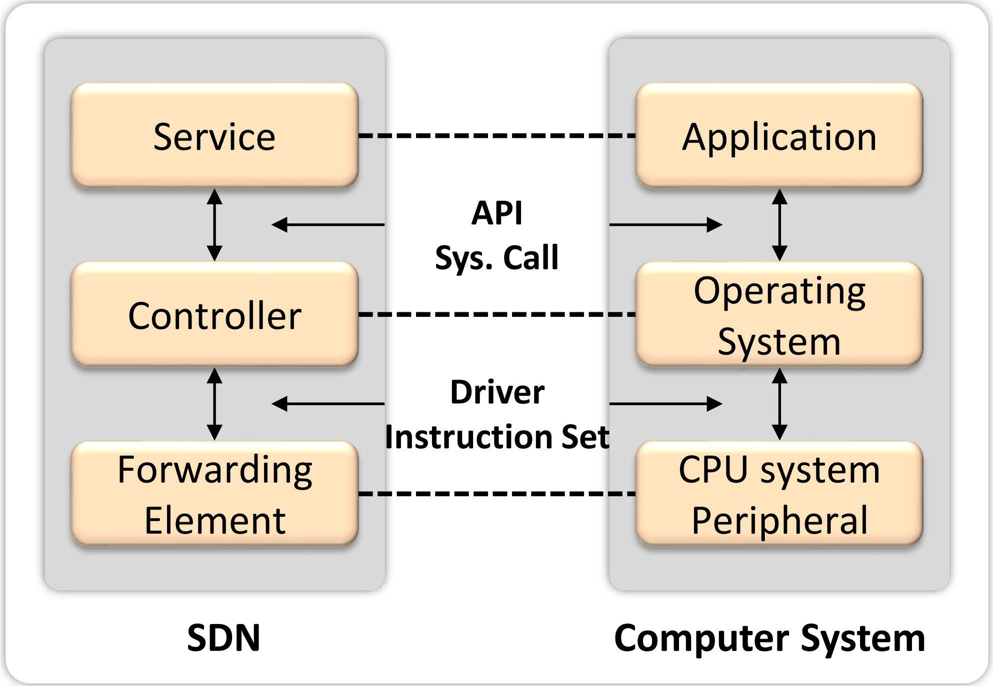
Protocol Conscious Packet Forwarding

- Fixed parser extracts fixed set of header fields
- Pre-defined tables implement corresponding protocols or services
- Once deployed, switch functionalities cannot be programmed to support other protocols or services (only table content can be programmed)



Operate your Network Device like a PC

- Simple & generic instruction set
- Ultimate flexibility & extensibility
- Upgrade only on performance



Computer system components have been decoupled from the vertical integration model. SDN is on the track to mimic this transition. But, current OpenFlow still doesn’t embrace this model to the full extent.

Core Concept of POF

- Table search keys are defined as {offset, length} tuples
- Instructions/Actions access packet data or metadata using {offset, length} tuples
- Include other math, logic, move, branching, and jump instructions

	Current OpenFlow		POF
Match	~40 matching header fields defined yet still many uncovered protocols/headers	→	{offset, length} covers any frame based formats
Action	OFPAT_COPY_TTL_OUT OFPAT_COPY_TTL_IN OFPAT_SET_MPLS_TTL OFPAT_DEC_MPLS_TTL OFPAT_PUSH_VLAN OFPAT_POP_VLAN OFPAT_PUSH_MPLS OFPAT_POP_MPLS OFPAT_SET_NW_TTL OFPAT_DEC_NW_TTL and on and on and on …	→	POFAT_SET_FIELD POFAT_ADD_FIELD POFAT_DELETE_FIELD POFAT_MOD_FIELD Period.

Packet field parsing and handling are abstracted as generic instructions to enable flexible and future proof forwarding elements. This is simple yet has profound implications to SDN.

Ask a Dumb FE to Do Smart Things

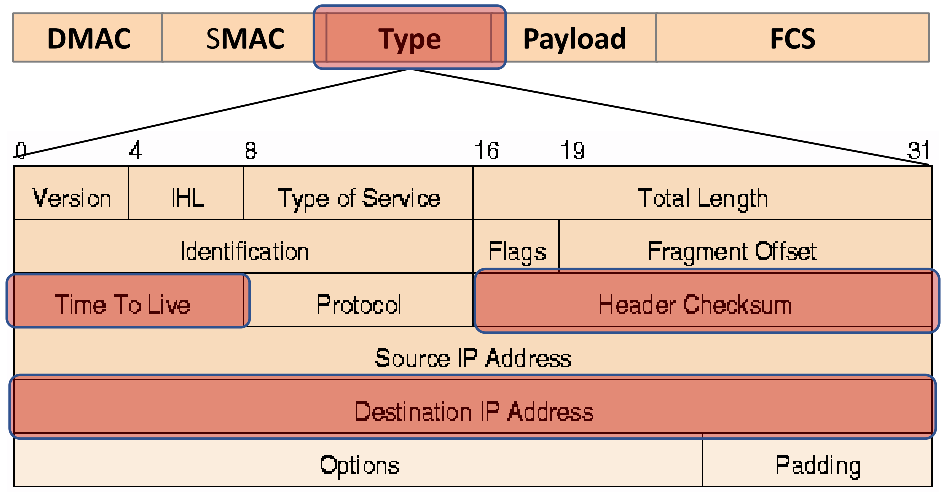
- The fine-grained bit-level manipulations used to be hardcoded or micro-coded in the FE are now explicitly described by controller

Is this an IP packet?
↓
Use destination IP to search FIB
↓
Process TTL
↓
Update Checksum

Conventional FE Forwarding Flow

Data @ {12B, 2B} = 0x0800?
↓
Extract Data @ {30B, 4B};
Use it as key to search LPM table *x*
↓
Decrement Data @ {22B, 1B};
If result is 0, drop the packet
↓
Clear Data @ {24B, 2B};
Use algorithm *y* to calculate checksum over Data @ {14B, 20B}; Write result @ {24B, 2B}

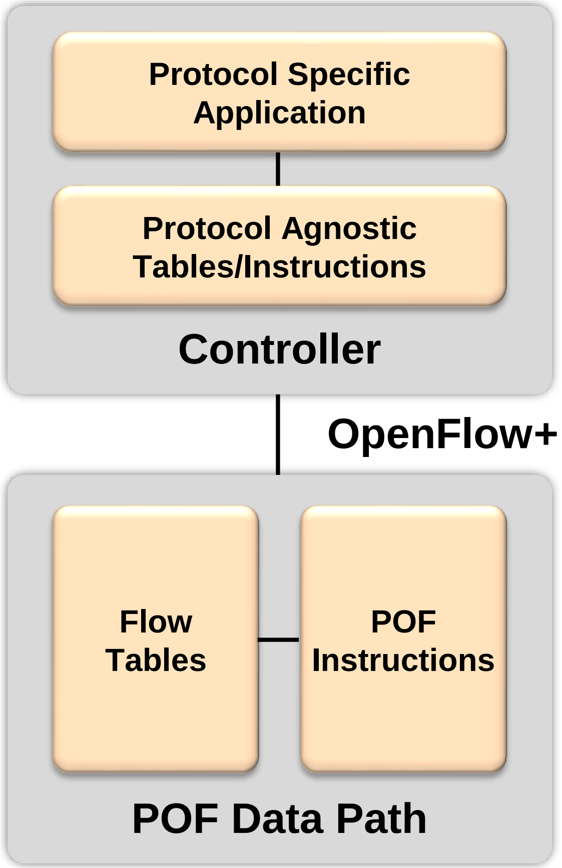
POF Forwarding Flow

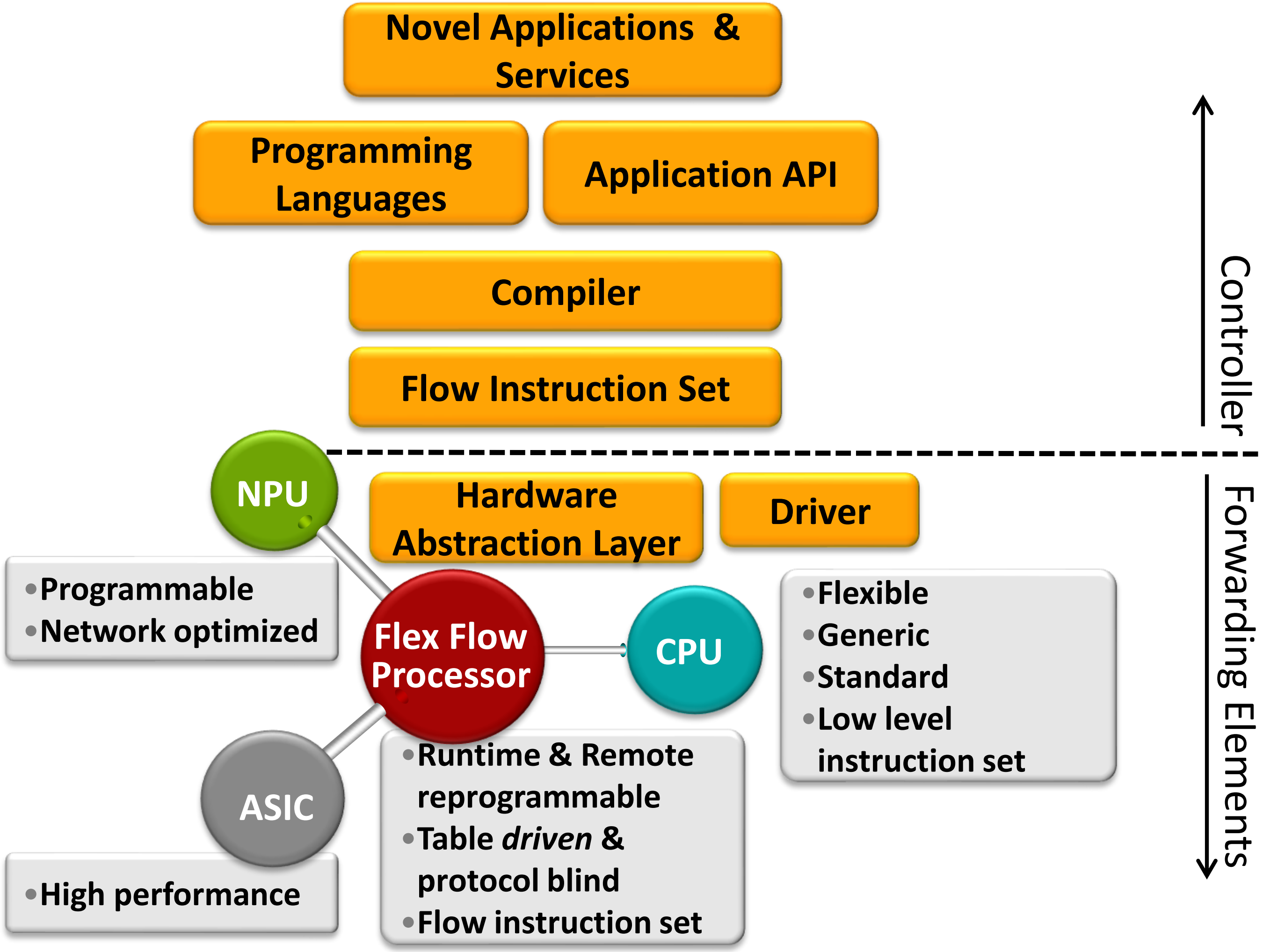


Ethernet/IPv4 Packet Format

OpenFlow's high level semantics ("what") is simple in communication but demands forwarding plane intelligence; POF's low level semantics ("how") moves all the intelligence up to the controller

POF-based SDN Architecture





PIF: Protocol Independent Forwarding

- **Phase 0**: Initially, the switch does not know what a protocol is, or how to process packets (Protocol Independence)

- **Phase 1**: We tell the switch how we want it to process packets (Configuration)

- **Phase 2**: The switch runs (Run-time)

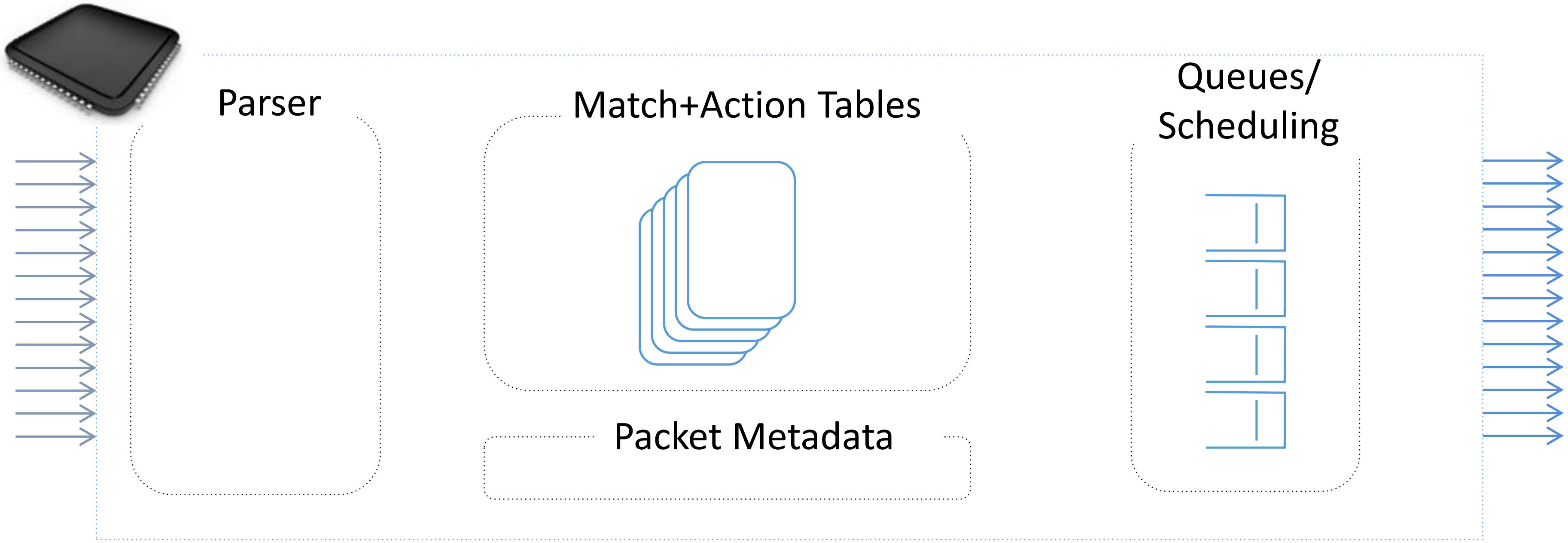
Three Goals

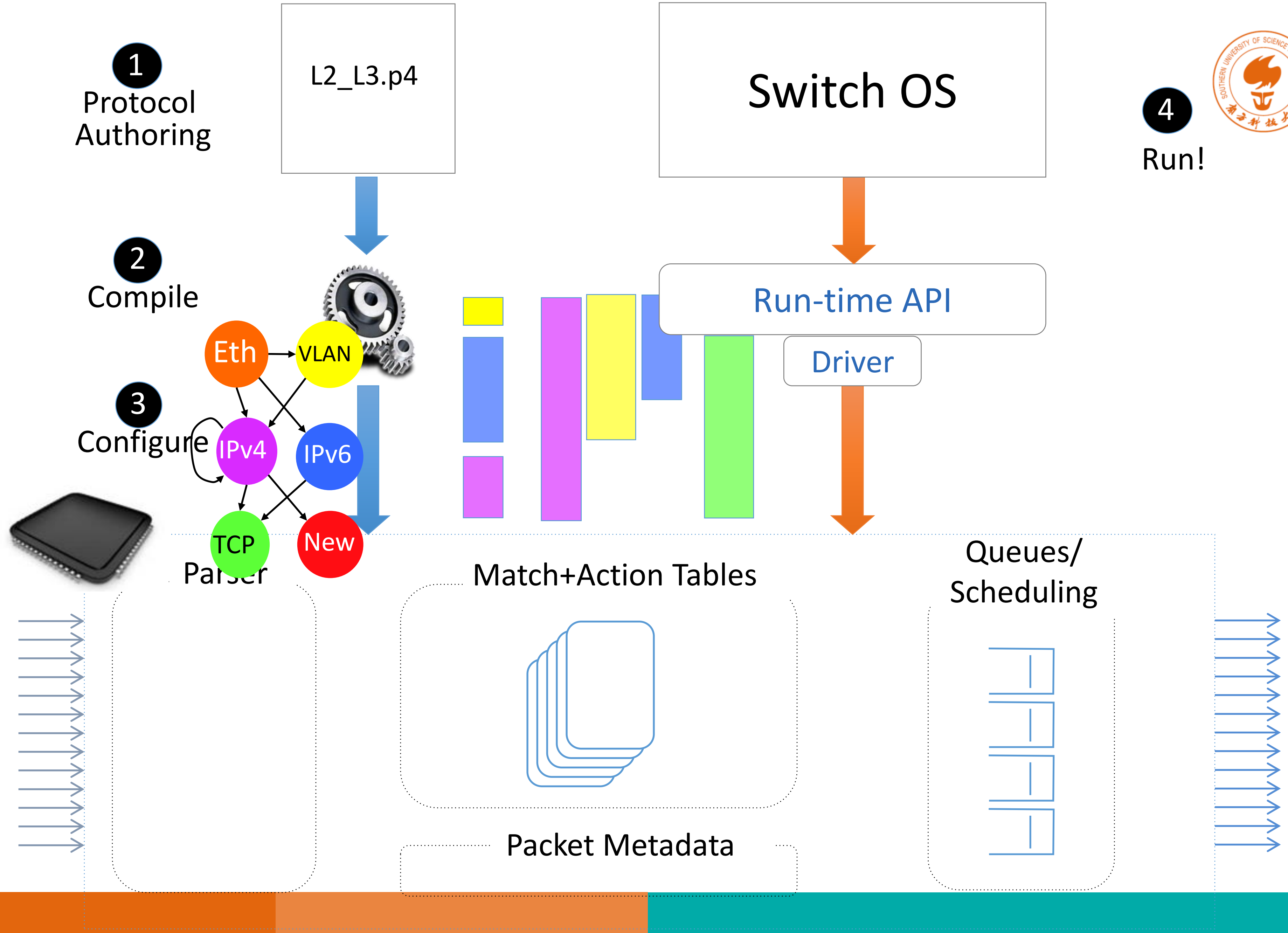
- **Protocol independence**
  - Configure a packet parser to extract relevant header fields
  - Define a set of typed <match, action> tables
- **Target independence**
  - Program without knowledge of switch details
  - Rely on compiler to configure the target switch
- **Reconfigurability**
  - Change parsing and processing in the field

The Abstract Forwarding Model

The Abstract Forwarding Model

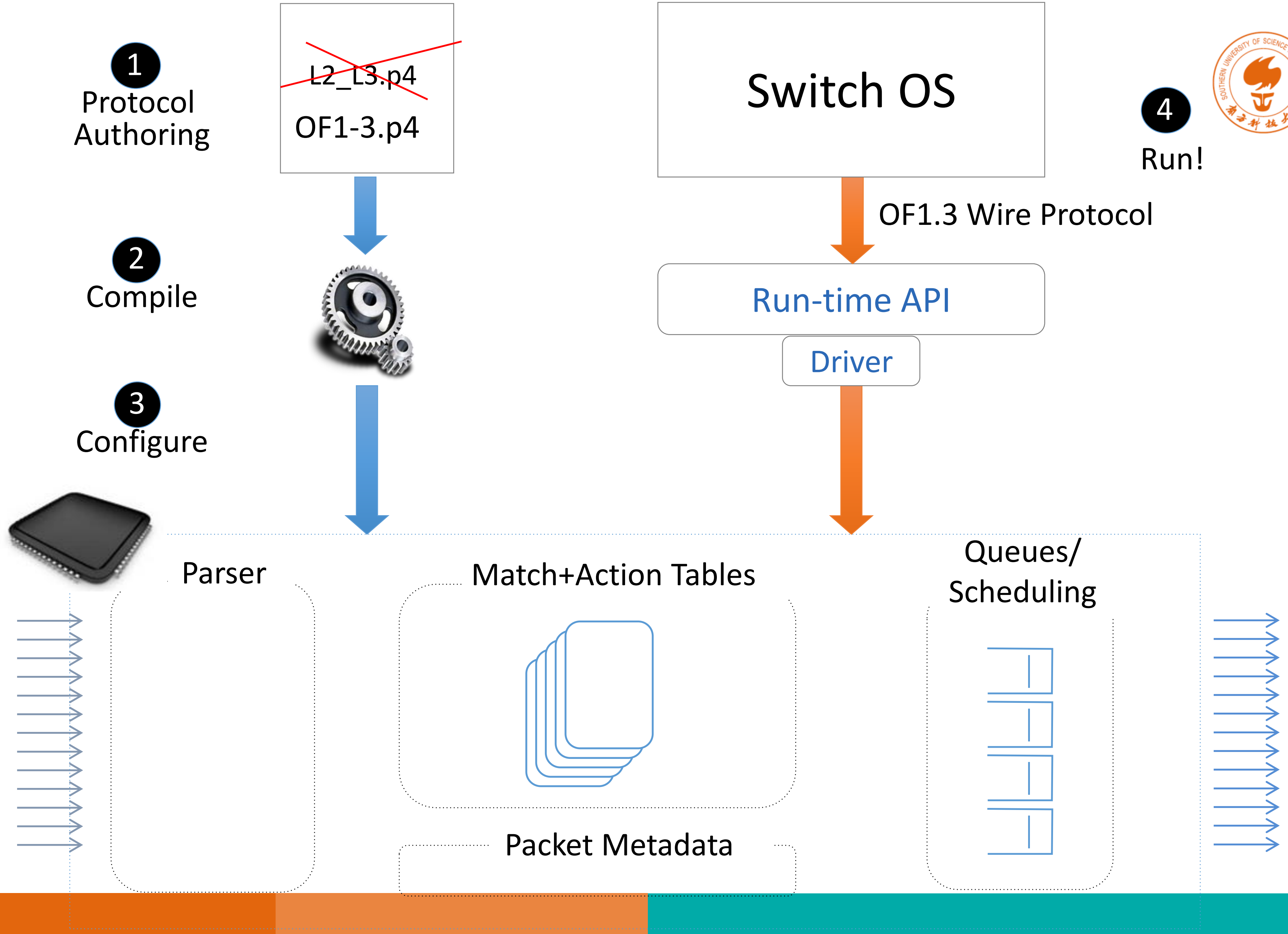
Initially, a switch is unprogrammed
and does not know any protocols.





1
Protocol Authoring
L2_L3.p4
Switch OS
4
Run!
2
Compile
Run-time API
Driver
Eth
VLAN
3
Configure
IPv4
IPv6
TCP
New
Parser
Match+Action Tables
Queues/ Scheduling
Packet Metadata

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



Outline

- Open vSwitch (OVS)
- Towards OpenFlow 2.0
- **New Forwarding Plane Architectures**
- Programming the Forwarding Plane

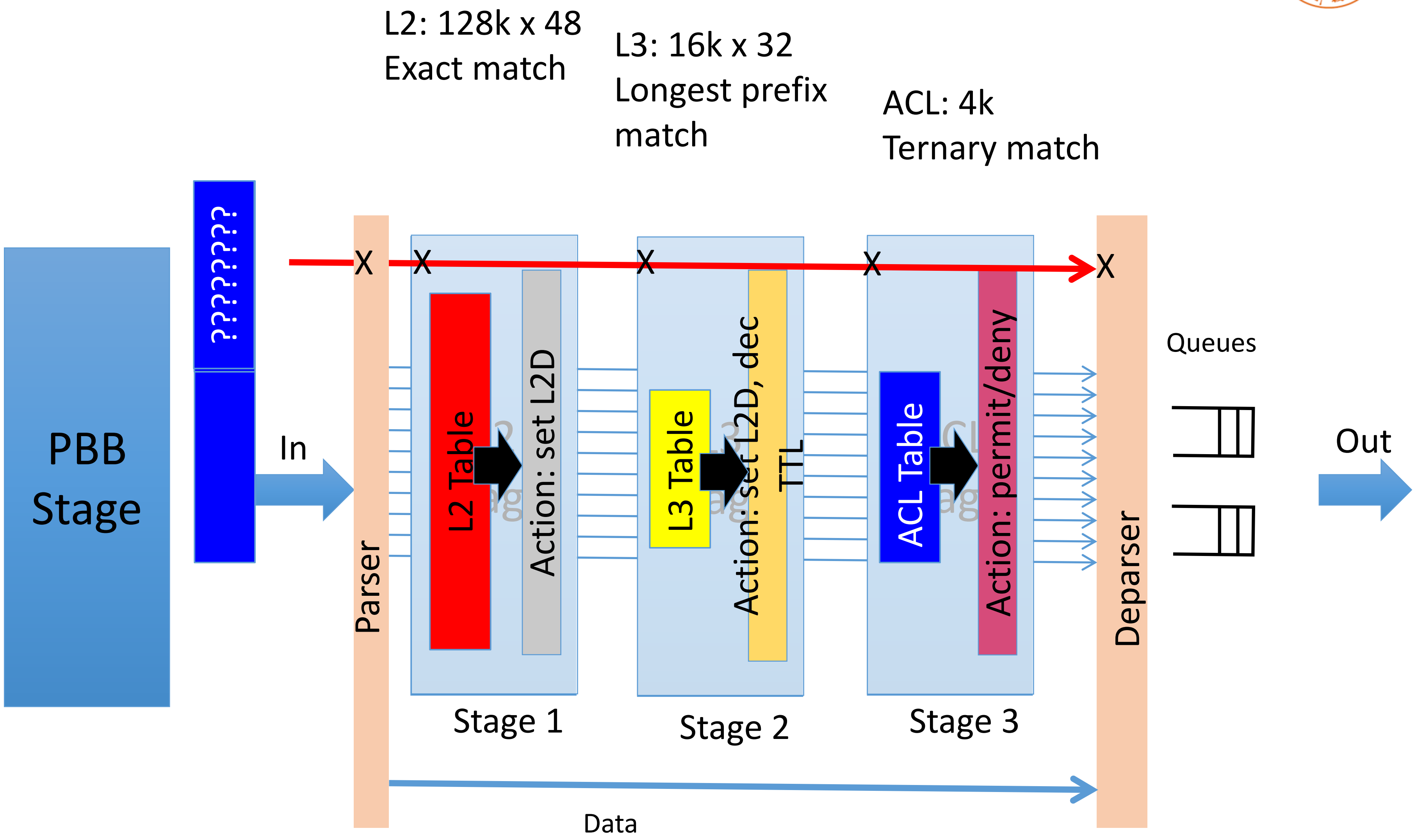
Match-Action Models

- **SMT (Single Match Table)**
  - The controller tells the switch to match any set of packet header fields against entries in a single match table
  - It assumes that a parser extracts the correct header fields to match against the table
  - SMT can be implemented using a wide Ternary Content Addressable Memory (TCAM)

- **MMT (Multiple Match Tables)**
  - It allows multiple smaller match tables to be matched by a subset of packet fields
  - The match tables are arranged into a pipeline of stages
  - Existing switch chips implement a small (4-8) number of tables whose widths, depths, and execution order are set when the chip is fabricated, severely limiting their fexibility

- **RMT (Reconfigurable Match Tables)**

A Fixed Function Switch



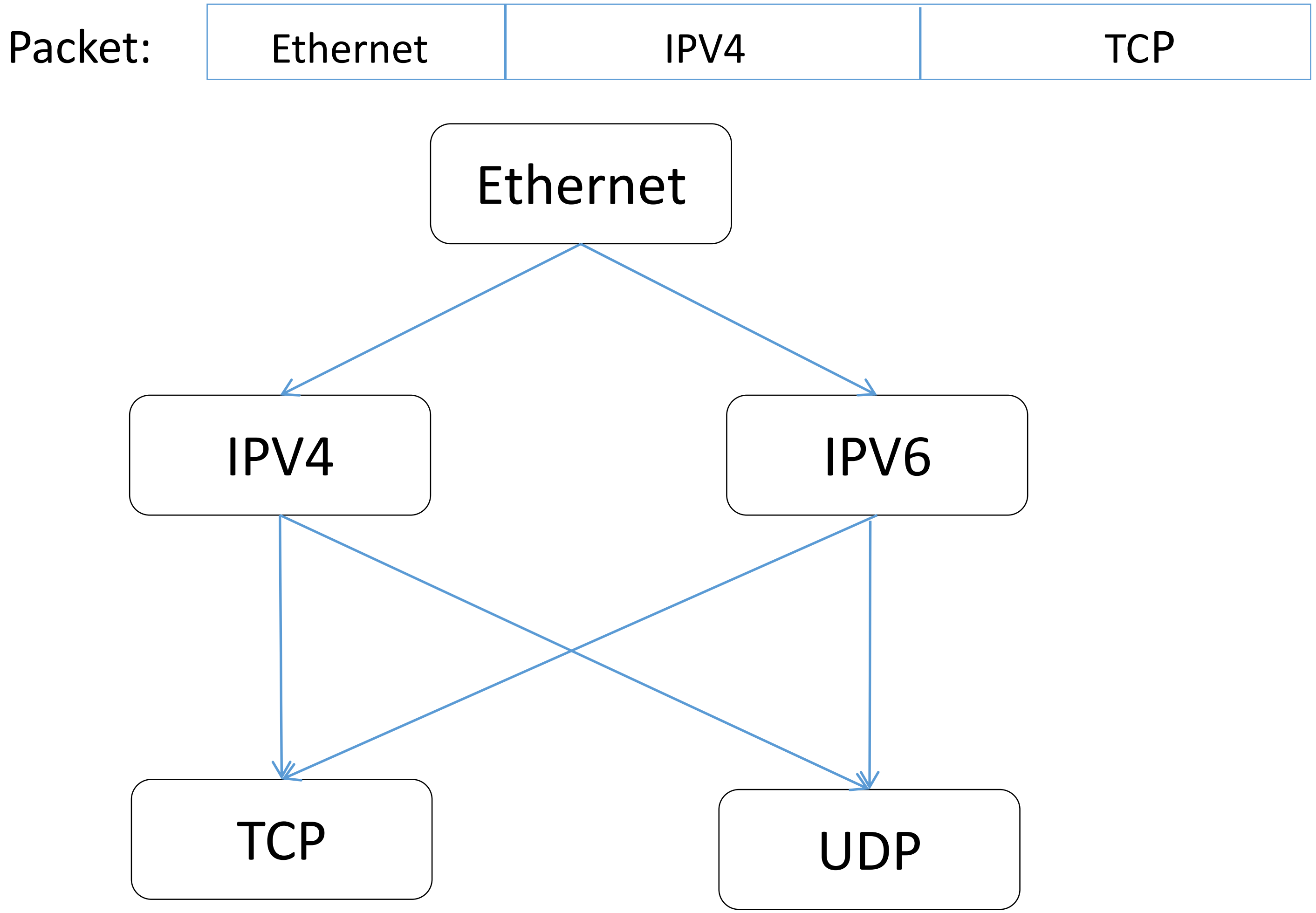
The RMT Abstract Model

- RMT: Reconfigurable Match Tables
- **Parse graph**
- **Table graph**

Read:
Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN ***[SIGCOMM'13]***

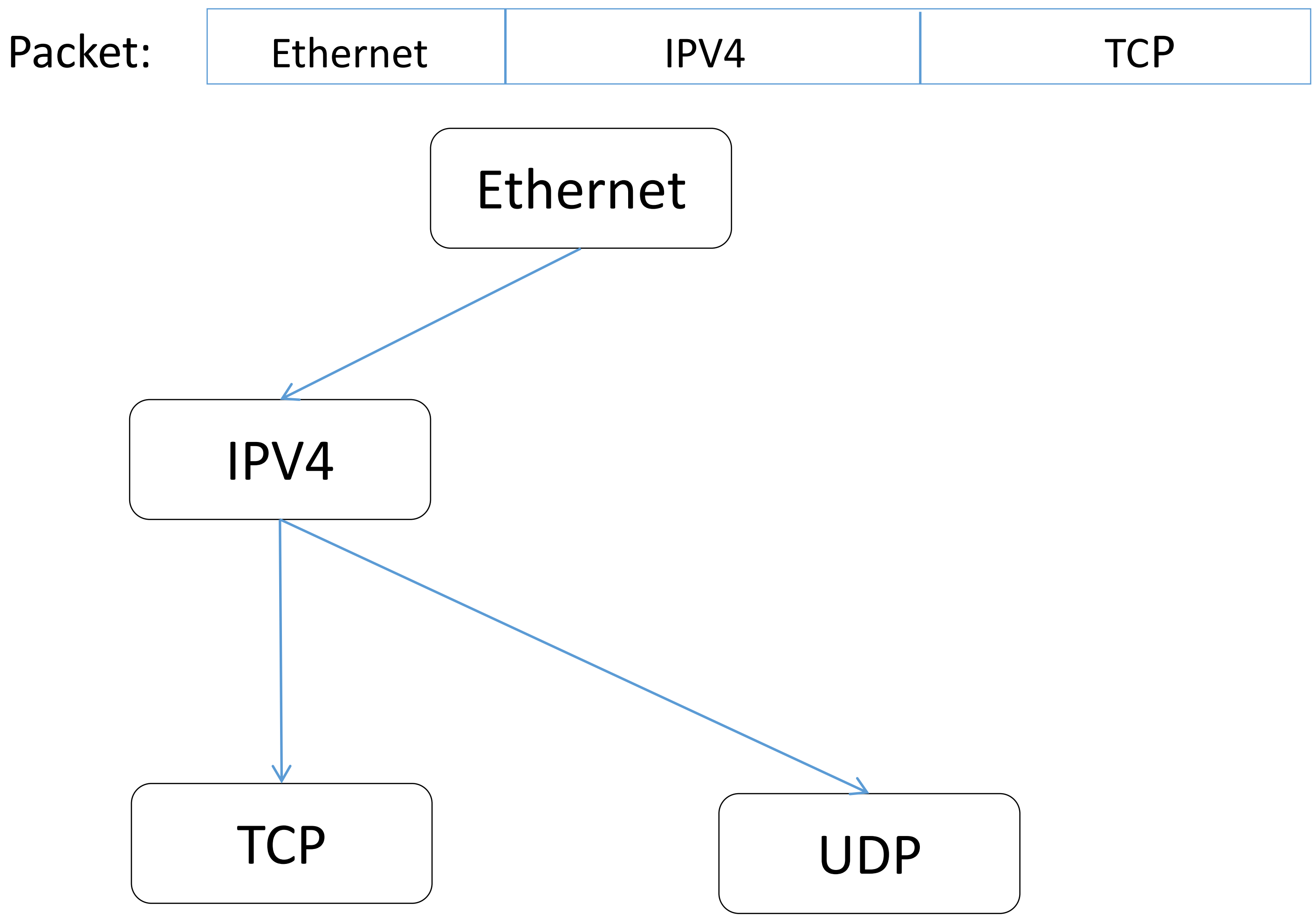
Parse Graph

- Describing arbitrary header fields to be extracted



Parse Graph

- Describing arbitrary header fields to be extracted



Parse Graph

Packet:

Ethernet	RCP	IPV4	TCP

- Describing arbitrary header fields to be extracted

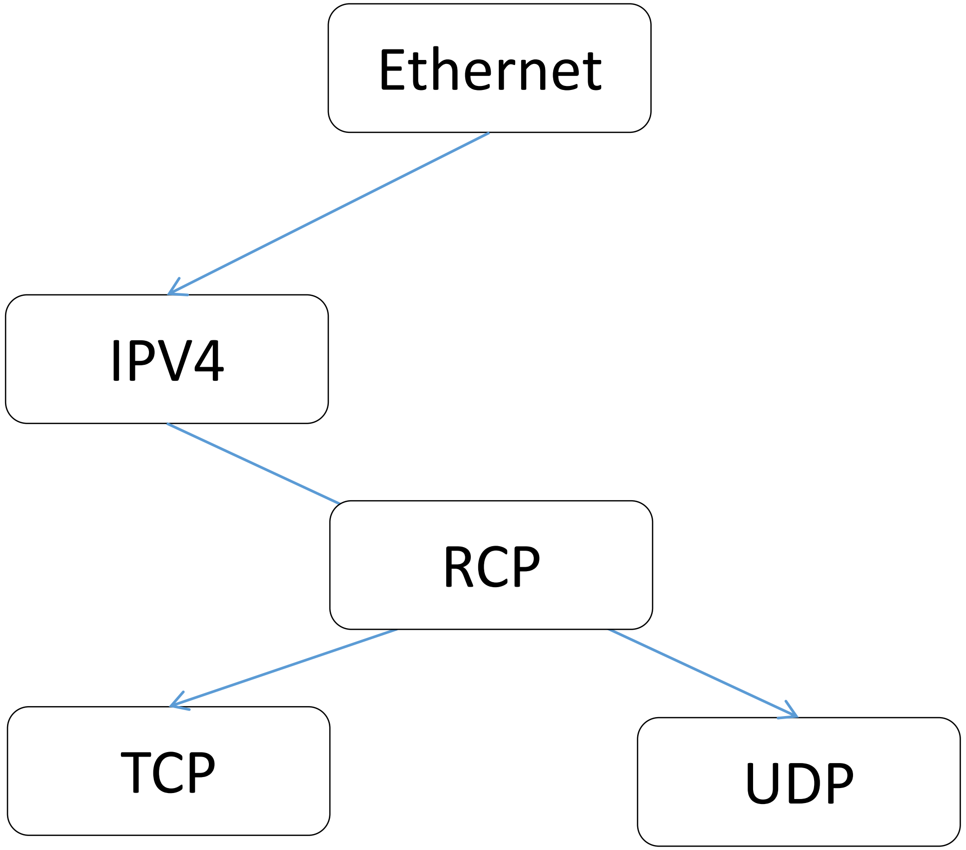
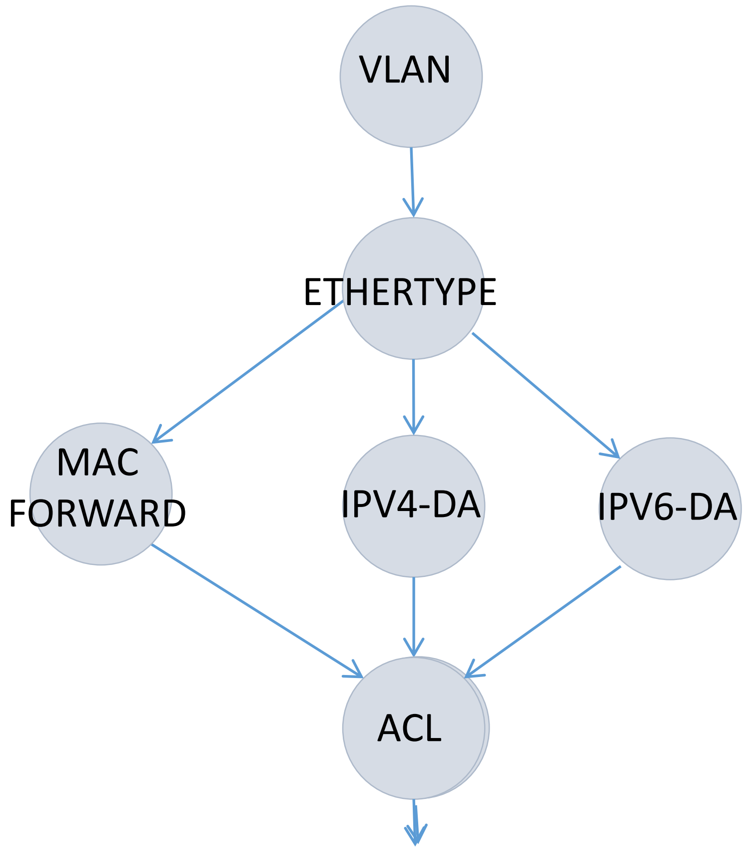
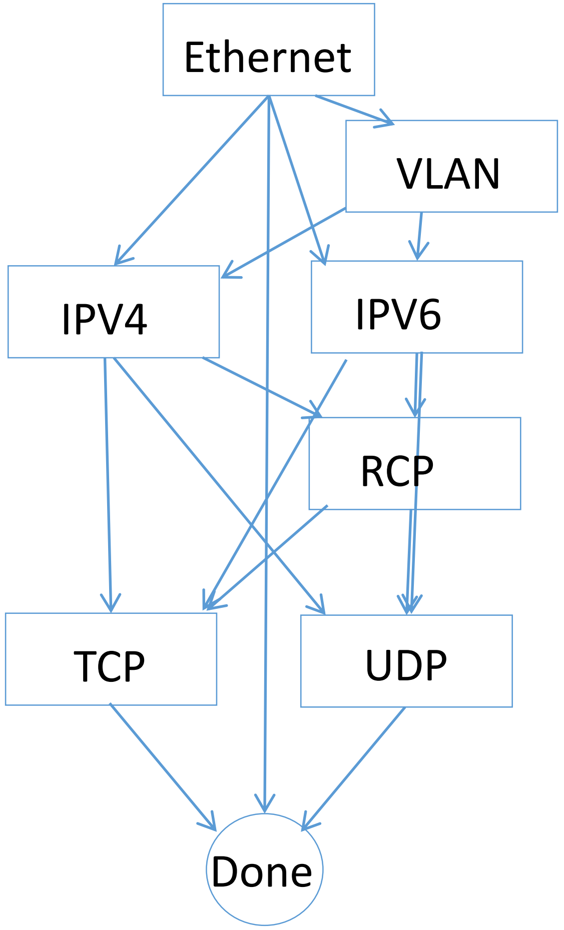
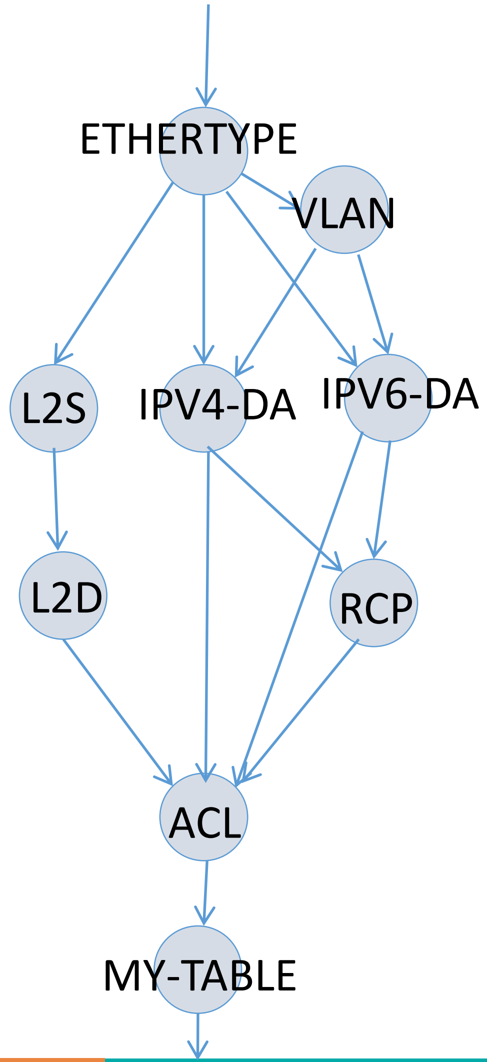


Table Graph



Changes to Parse Graph and Table Graph



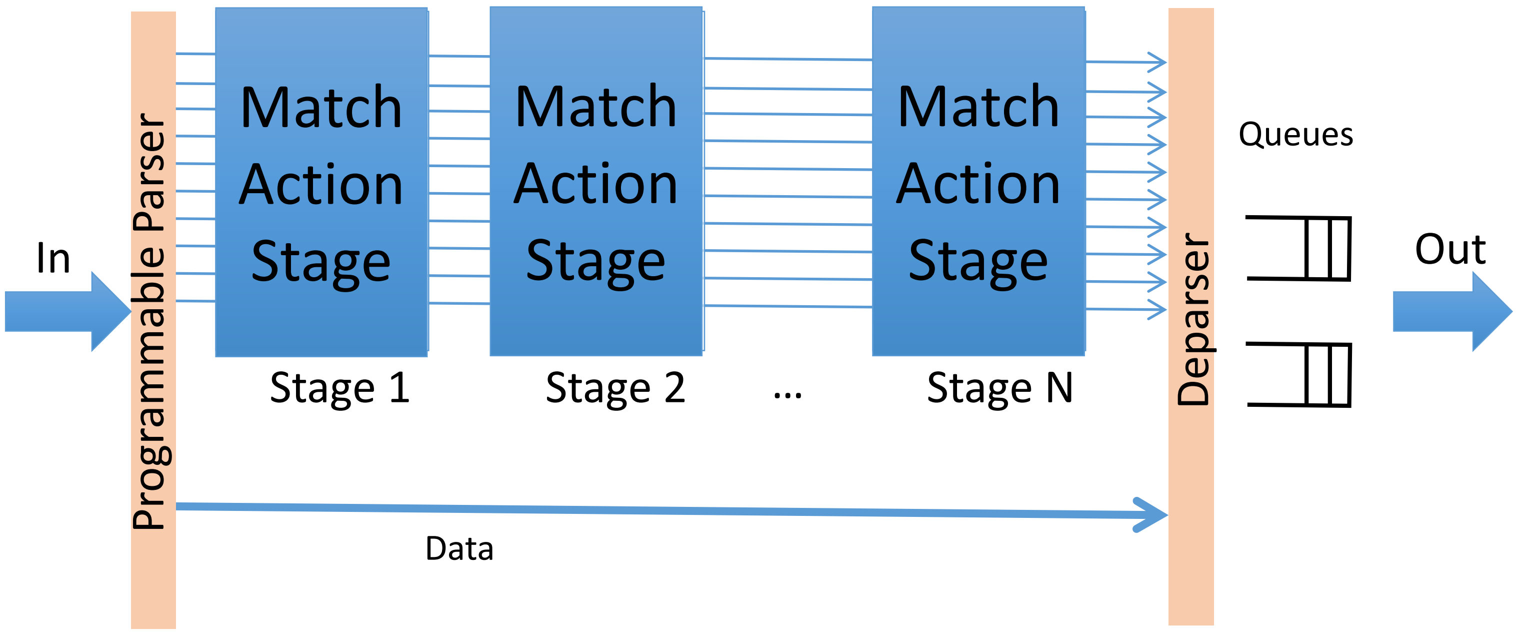


Parse Graph

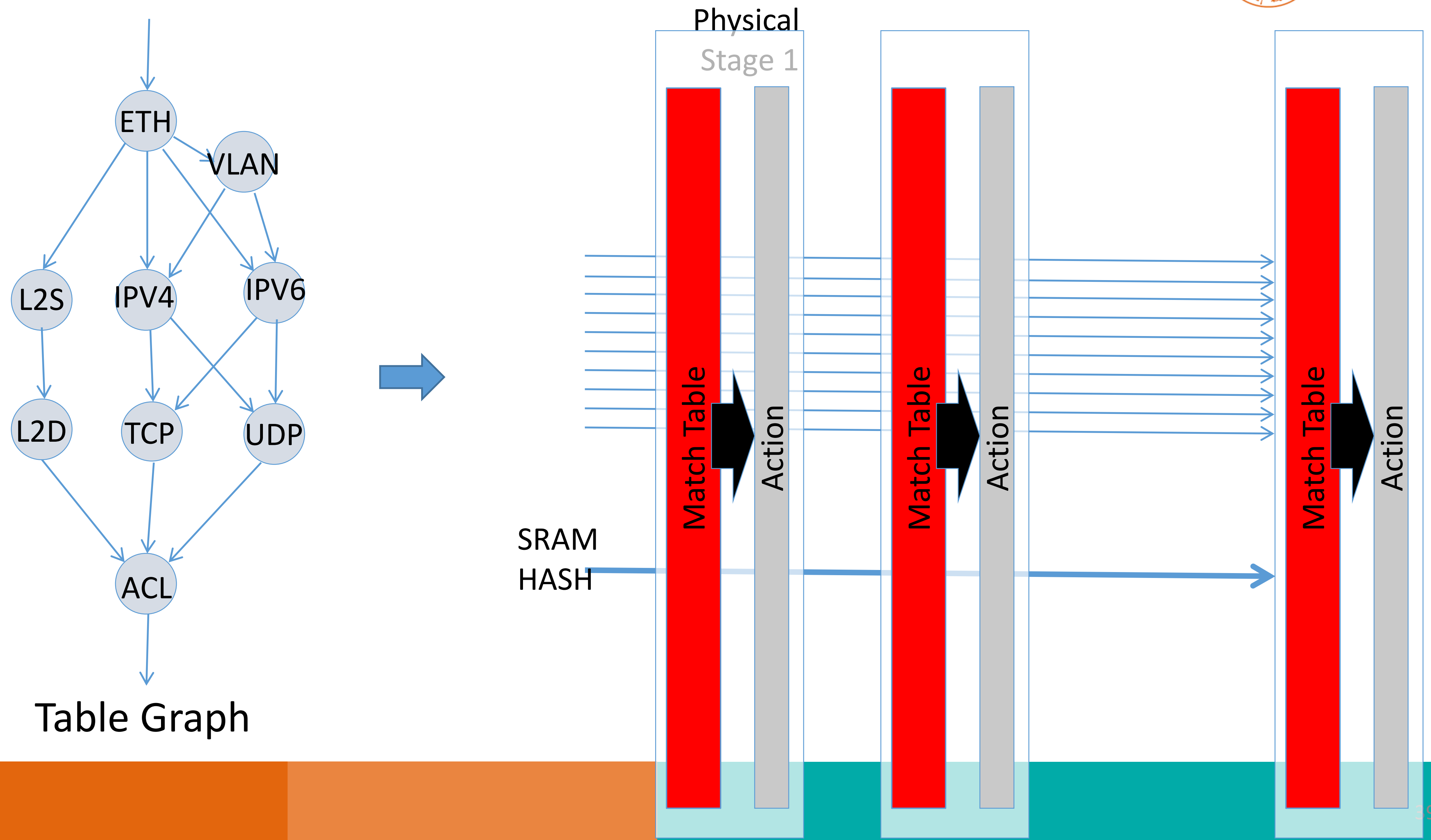
Table Graph

But the **Parse Graph** and **Table Graph** don’t show you how to build a switch

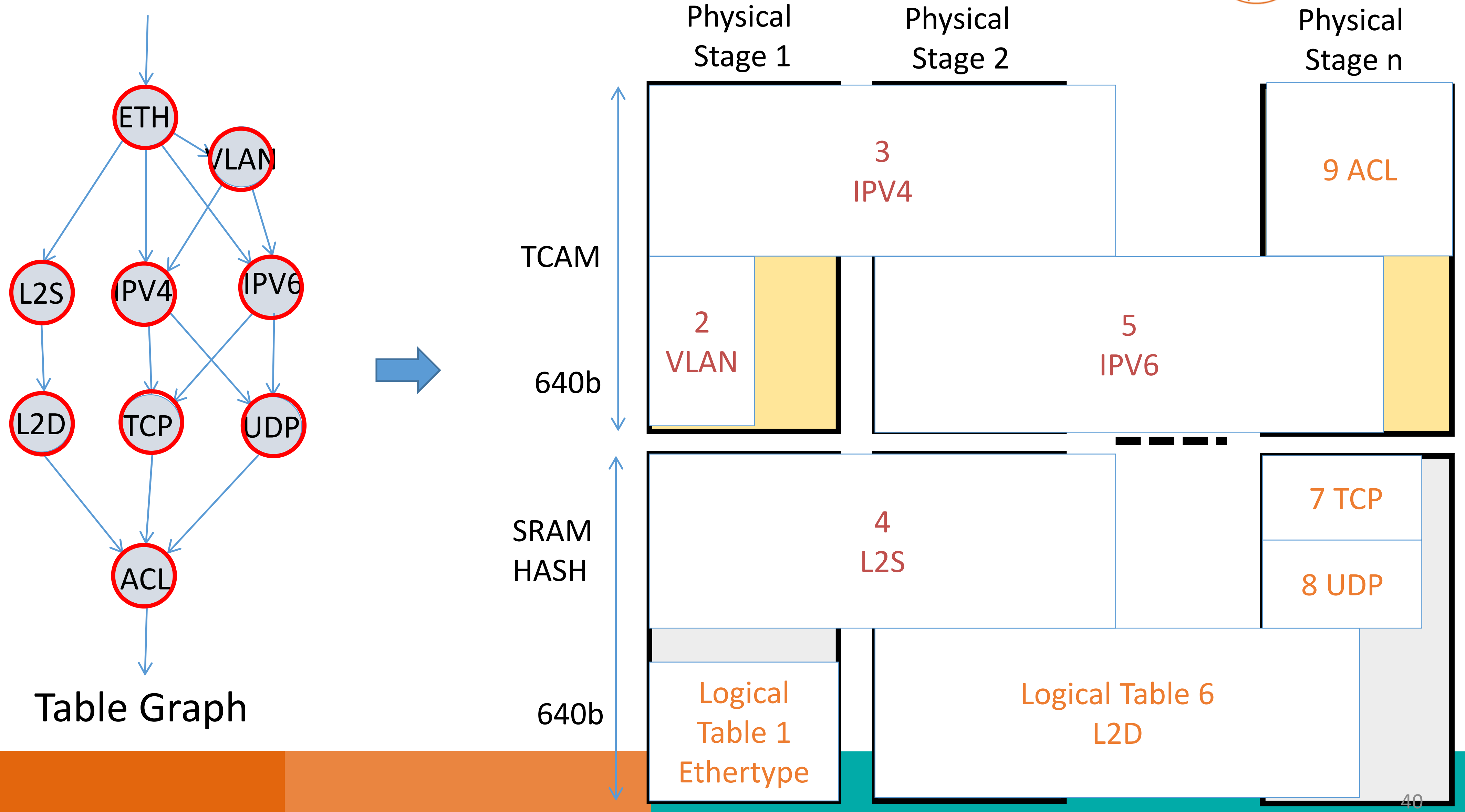
Match/Action Forwarding Model



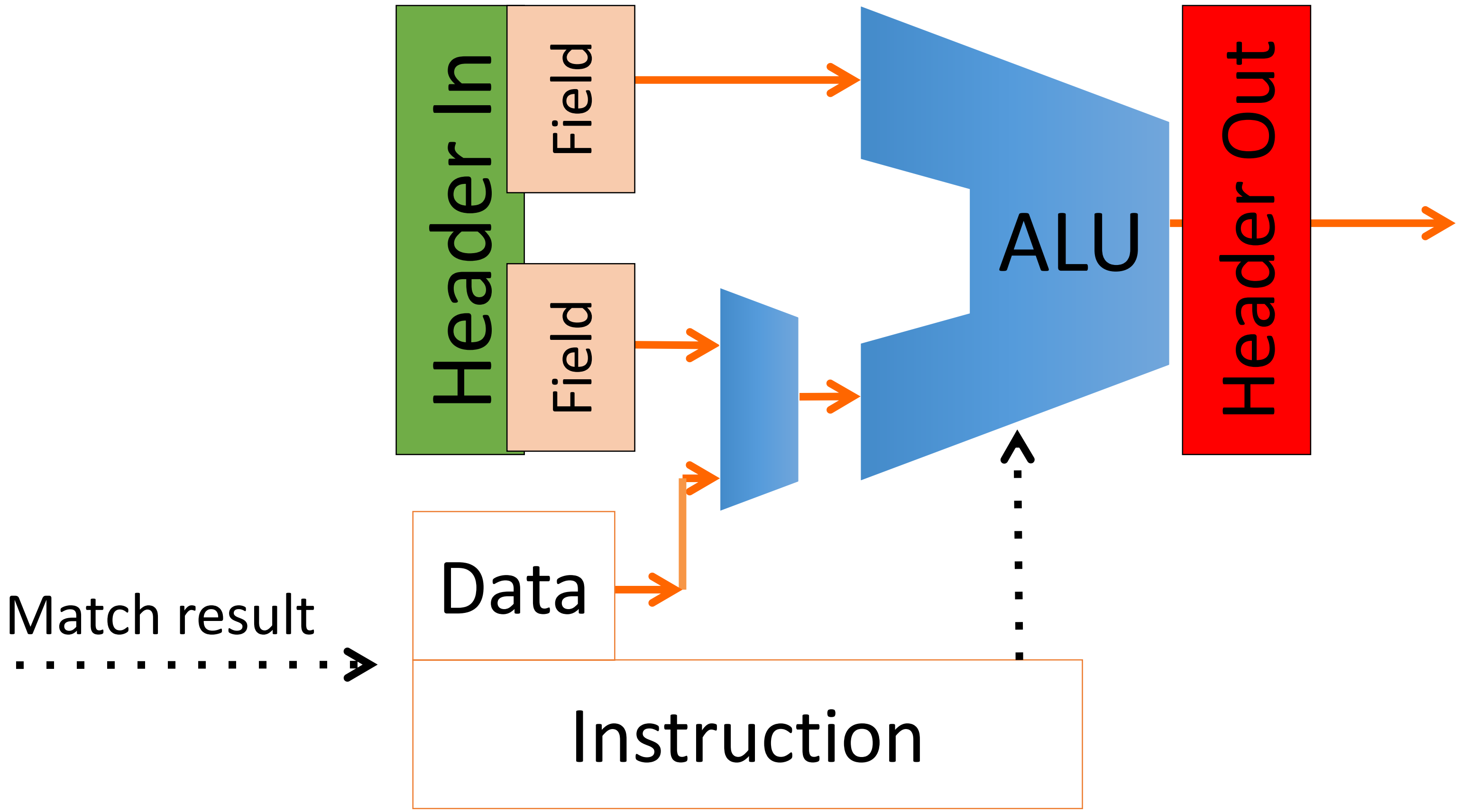
RMT Logical to Physical Table Mapping



RMT Logical to Physical Table Mapping

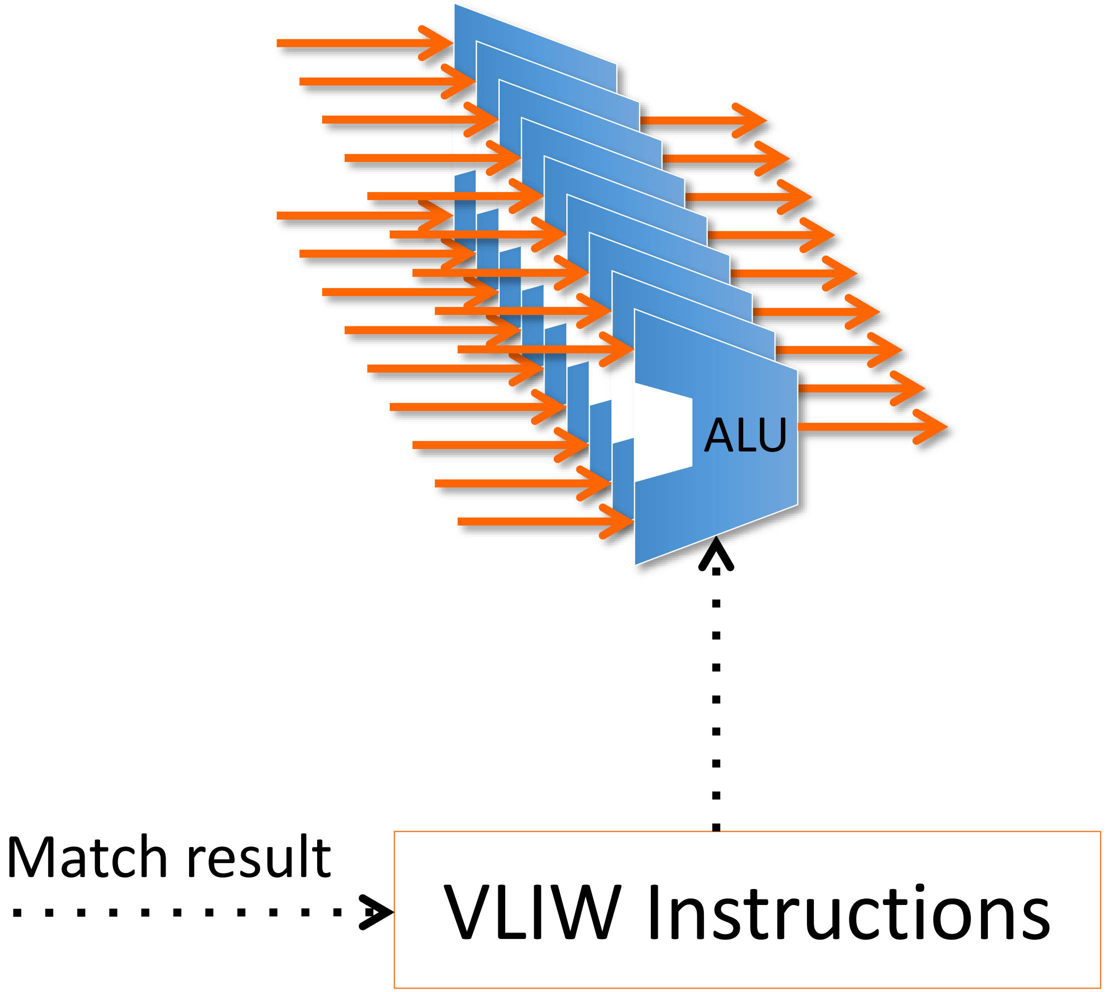


Action Processing Unit



Modeled as Multiple VLIW CPUs per Stage

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



RMT Chip Configuration

- 64 x 10Gb ports
  - 960M packets/second
  - 1GHz pipeline
- Programmable parser
- 32 Match/action stages

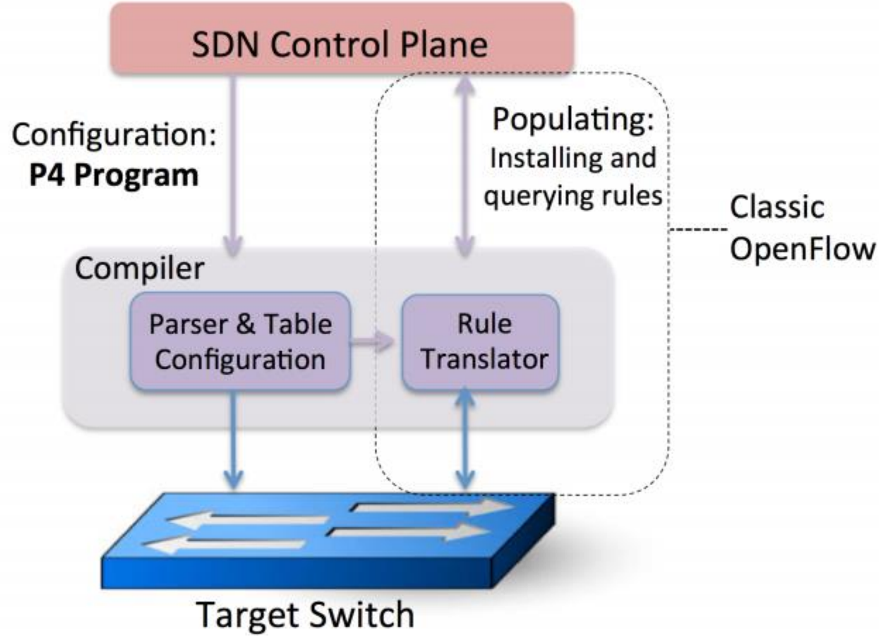
- Huge TCAM: 10x current chips
  - 64K TCAM words x 640b
- SRAM hash tables for exact matches
  - 128K words x 640b
- 224 action processors per stage
- All OpenFlow statistics counters

Outline

- Open vSwitch (OVS)
- Towards OpenFlow 2.0
- New Forwarding Plane Architectures
- **Programming the Forwarding Plane**

P4 Language

- P4: **P**rogramming **P**rotocol-Independent **P**acket **P**rocessors
  - An open source language allowing the specification of packet processing logic
  - Based on a Match+Action forwarding model



- Participated by
  - Stanford, Princeton
  - Google, Intel, Microsoft, Barefoot

Read:
P4: Programming Protocol-independent Packet Processors ***[SIGCOMM Review'14]***

Programming RMT with P4

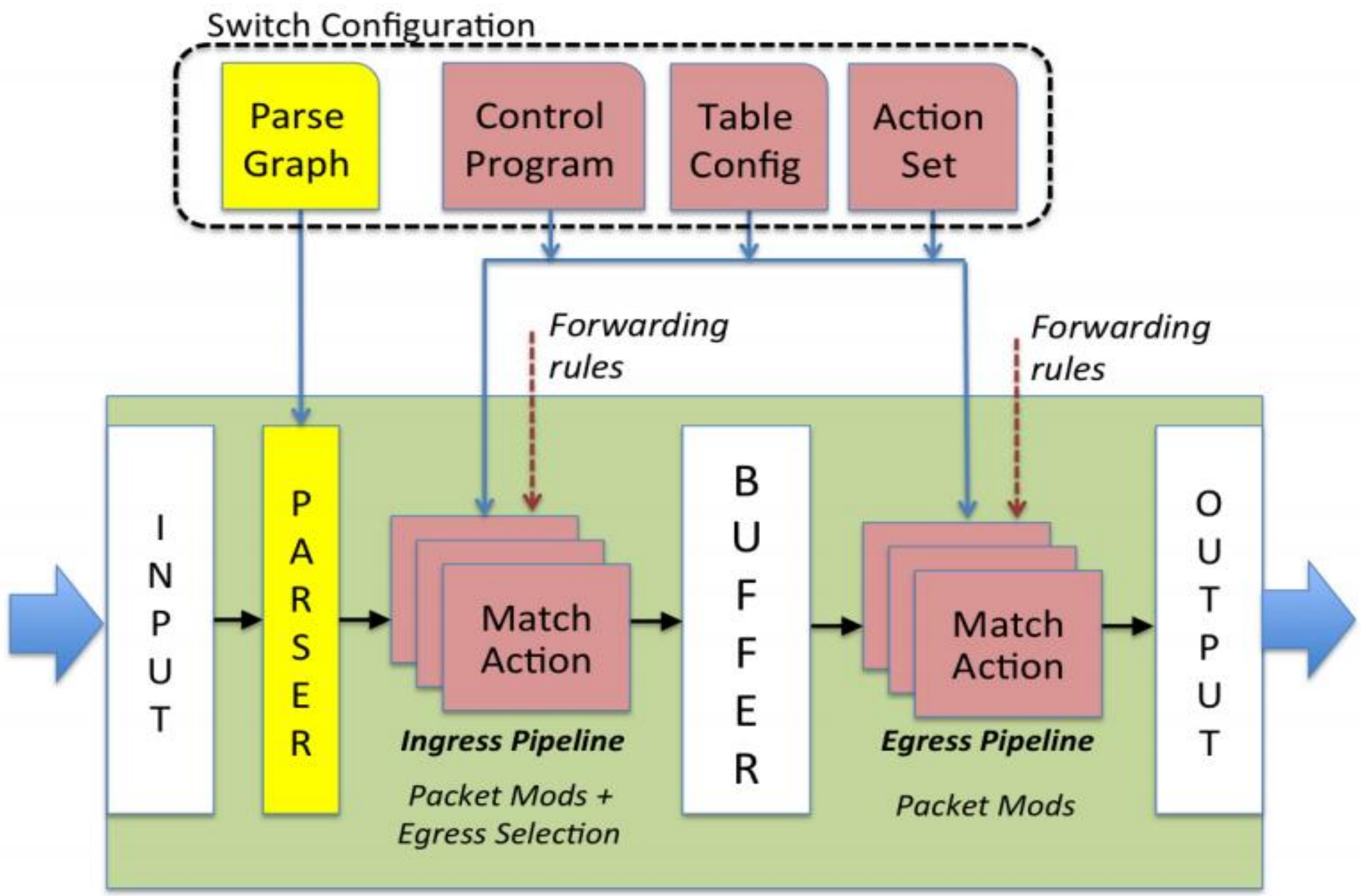
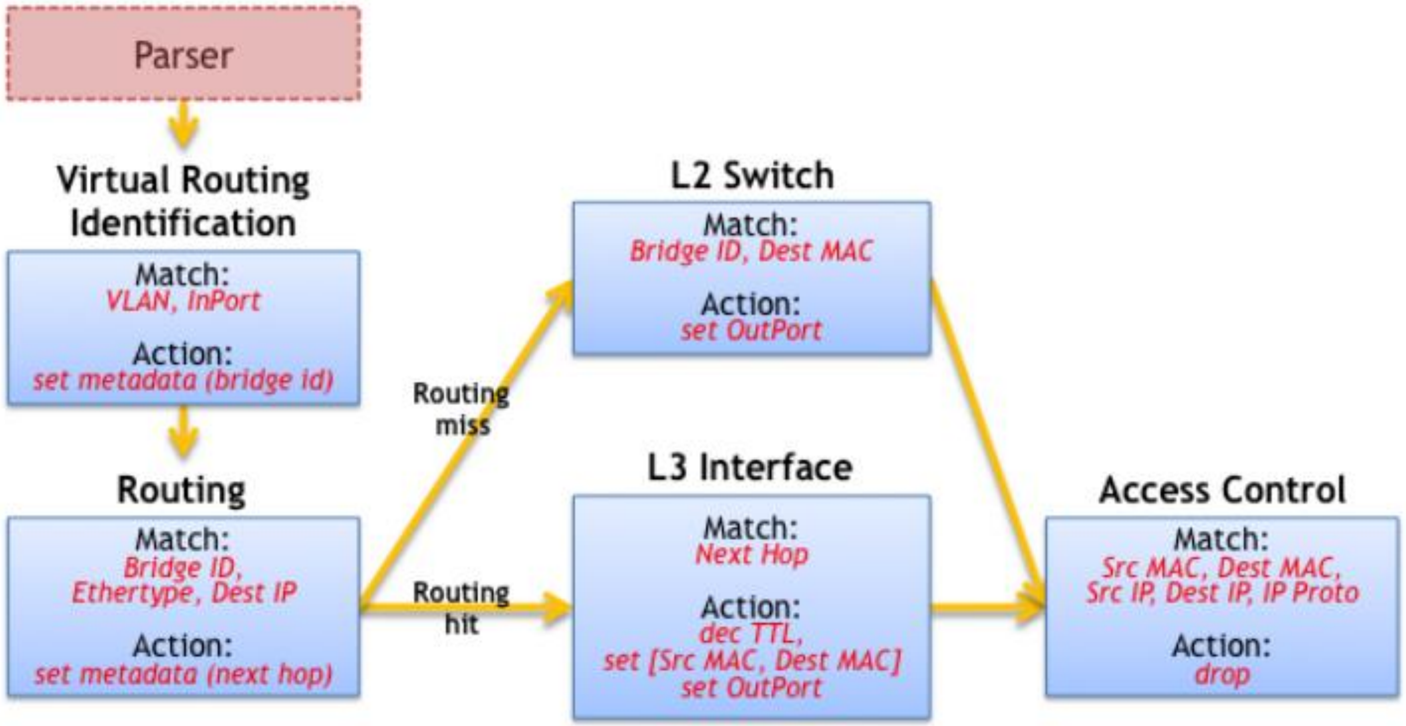


Table dependency graph (TDG) for an L2/L3 switch



P4 Language Elements

- **Headers**
  - A header describes the sequence and structure of a series of fields
- **Parsers**
  - A parser specifies how to identify headers within packets
- **Tables**
  - A table defines the fields on which a packet may match and the actions may take
- **Actions**
  - Complex actions can be constructed from simpler protocol-independent primitives
- **Control Programs**
  - A control program gives the order of match+action tables applied to a packet

Headers and Fields

- **Fields** have width and other attributes
- **Headers** are collections of fields

```
header ethernet {
    fields {
        dst_addr : 48; // width in bits
        src_addr : 48;
        ethertype : 16;
    }
}
```

```
header vlan {
    fields {
        pcp : 3;
        cfi : 1;
        vid : 12;
        ethertype : 16;
    }
}
```

Parser

- Extracts header fields from the packet

```
parser start {
    ethernet;
}

parser ethernet {
    switch(ethertype) {
        case 0x8100: vlan;
        case 0x9100: vlan;
        case  0x800: ipv4;
        // Other cases
    }
}
```

```
parser vlan {
    switch(ethertype) {
        case 0xaaaa: mTag;
        case  0x800: ipv4;
        // Other cases
    }
}

parser mTag {
    switch(ethertype) {
        case 0x800: ipv4;
        // Other cases
    }
}
```

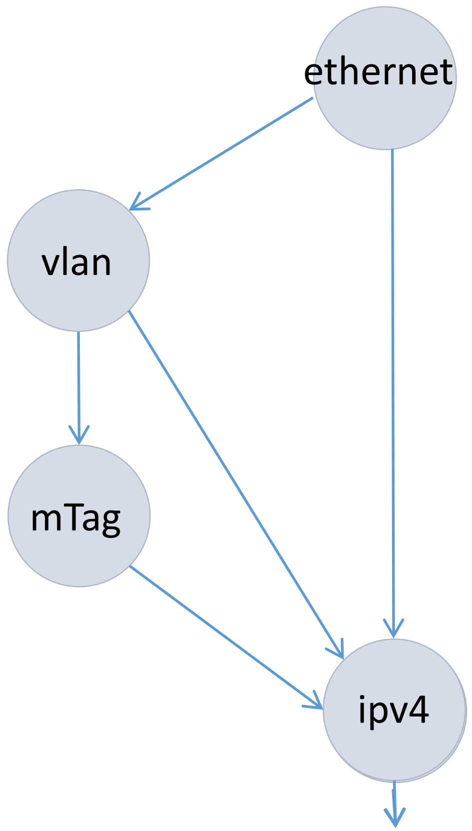


Table Specification

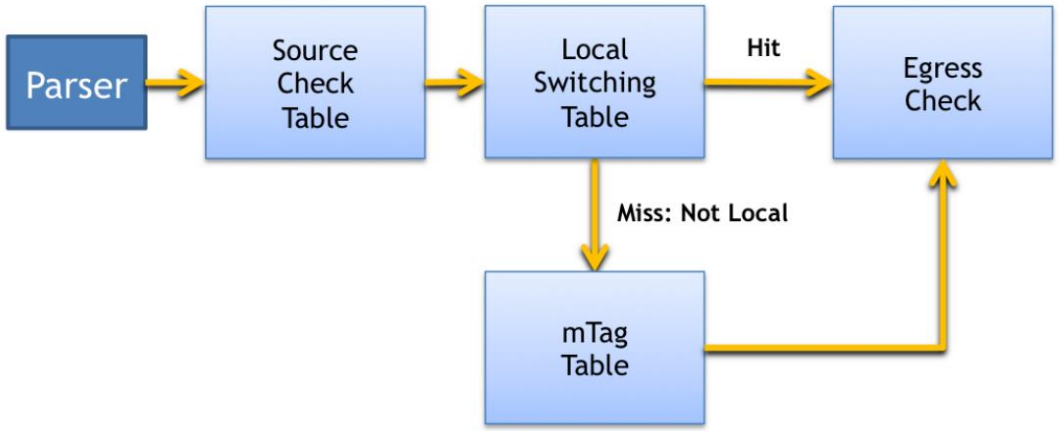
```
table mTag_table {
    reads {
        ethernet.dst_addr : exact;
        vlan.vid : exact;
    }
    actions {
        // At runtime, entries are programmed with params
        // for the mTag action.  See below.
        add_mTag;
    }
    max_size : 20000;
}
```

Action Specification

```
action add_mTag(up1, up2, down1, down2, egr_spec) {
    add_header(mTag);
    // Copy VLAN ethertype to mTag
    copy_field(mTag.ethertype, vlan.ethertype);
    // Set VLAN's ethertype to signal mTag
    set_field(vlan.ethertype, 0xaaaa);
    set_field(mTag.up1, up1);
    set_field(mTag.up2, up2);
    set_field(mTag.down1, down1);
    set_field(mTag.down2, down2);

    // Set the destination egress port as well
    set_field(metadata.egress_spec, egr_spec);
}
```

The Control Program



Flow chart for the mTag example

```
control main() {
    // Verify mTag state and port are consistent
    table(source_check);

    // If no error from source_check, continue
    if (!defined(metadata.ingress_error)) {
        // Attempt to switch to end hosts
        table(local_switching);

        if (!defined(metadata.egress_spec)) {
            // Not a known local host; try mtagging
            table(mTag_table);
        }

        // Check for unknown egress state or
        // bad retagging with mTag.
        table(egress_check);
    }
}
```

Compiling a P4 Program

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- The compiler translates the parser description into a parsing state machine

Current State	Lookup Value	Next State
vlan	0xaaaa	mTag
vlan	0x800	ipv4
vlan	*	stop
mTag	0x800	ipv4
mTag	*	stop

A partial state machine for the vlan and mTag section