P4: Protocol-Independent Packet Processors

Lecturer: Shan-Hsiang Shen

National Taiwan University of Science and Technology

Agenda

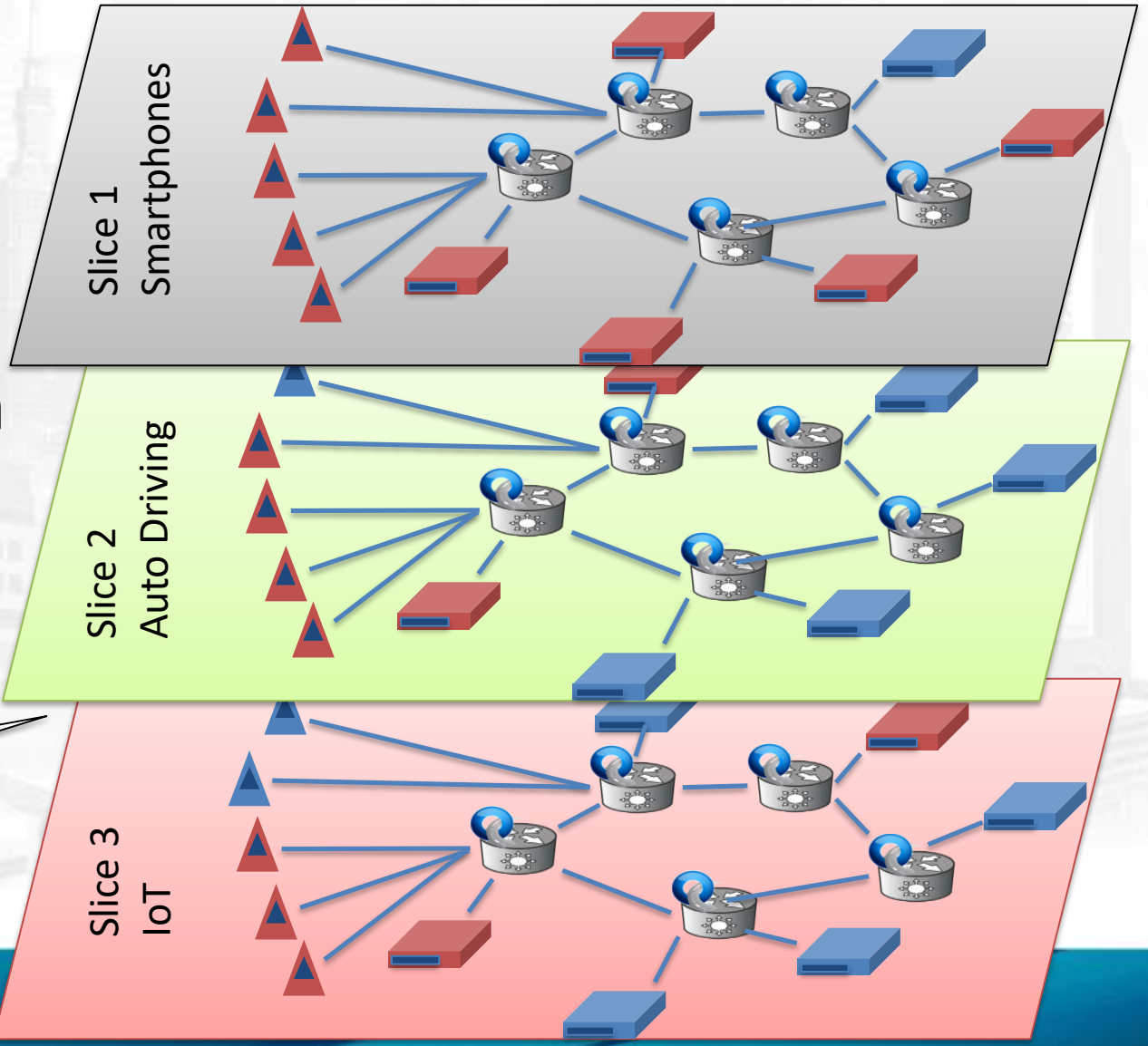
- The limitation of SDN
- Why P4?
- Packet processing in P4 architecture
- Basic P4 programming
- Table matching and action in P4
- P4 + SDN: a more complete future network

Agenda

- The limitation of SDN
- Why P4?
- Packet processing in P4 architecture
- Basic P4 programming
- Table matching and action in P4
- P4 + SDN: a more complete future network

Network Slicing

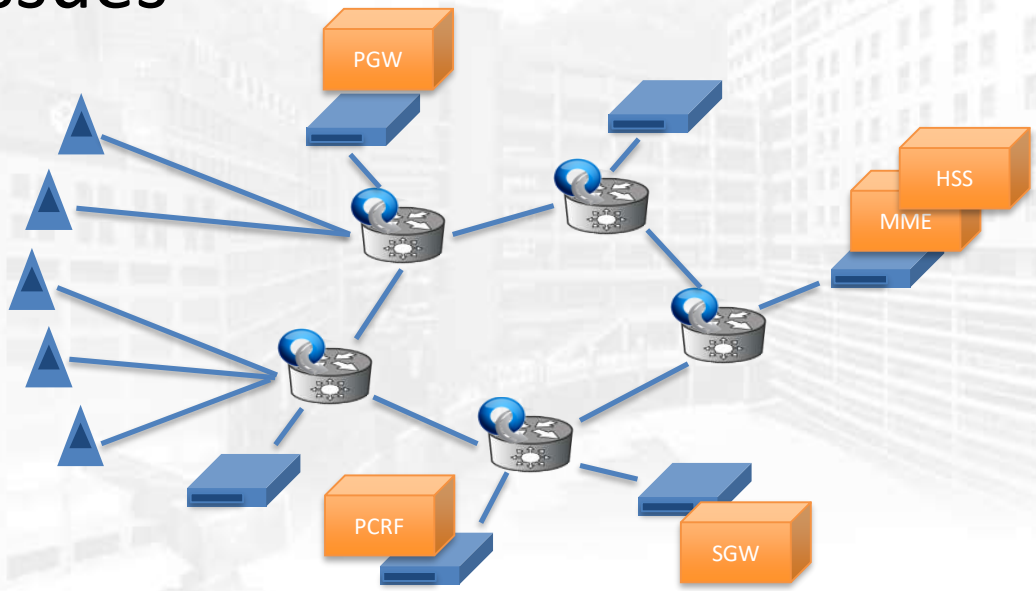
- To support multiple applications
- Virtualization layer to support the slicing



The virtualization layer may be attacked

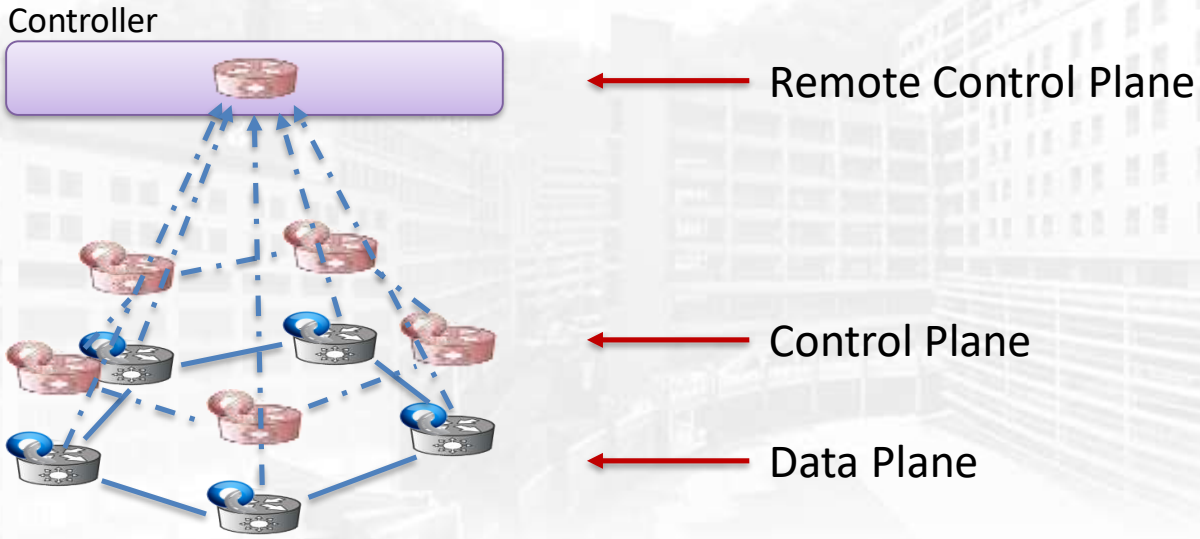
5G Mobile Networks

- SDN/NFV based networks
- New network architecture encounters new security issues



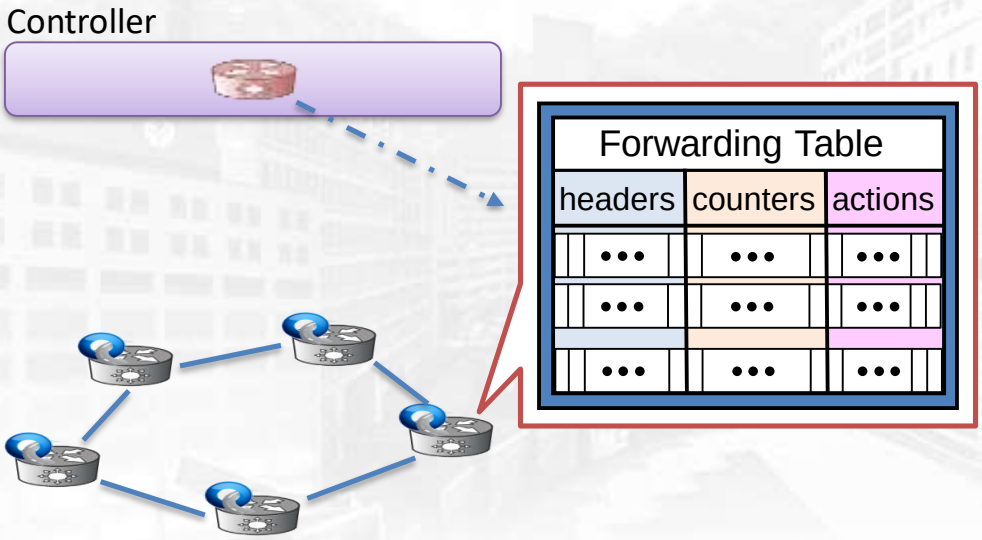
Software-Defined Networking

- Network devices are divided into control and data planes



Software-Defined Networking

- The controller install rules into forwarding table in switches to control traffic routing



The Limitation of SDN

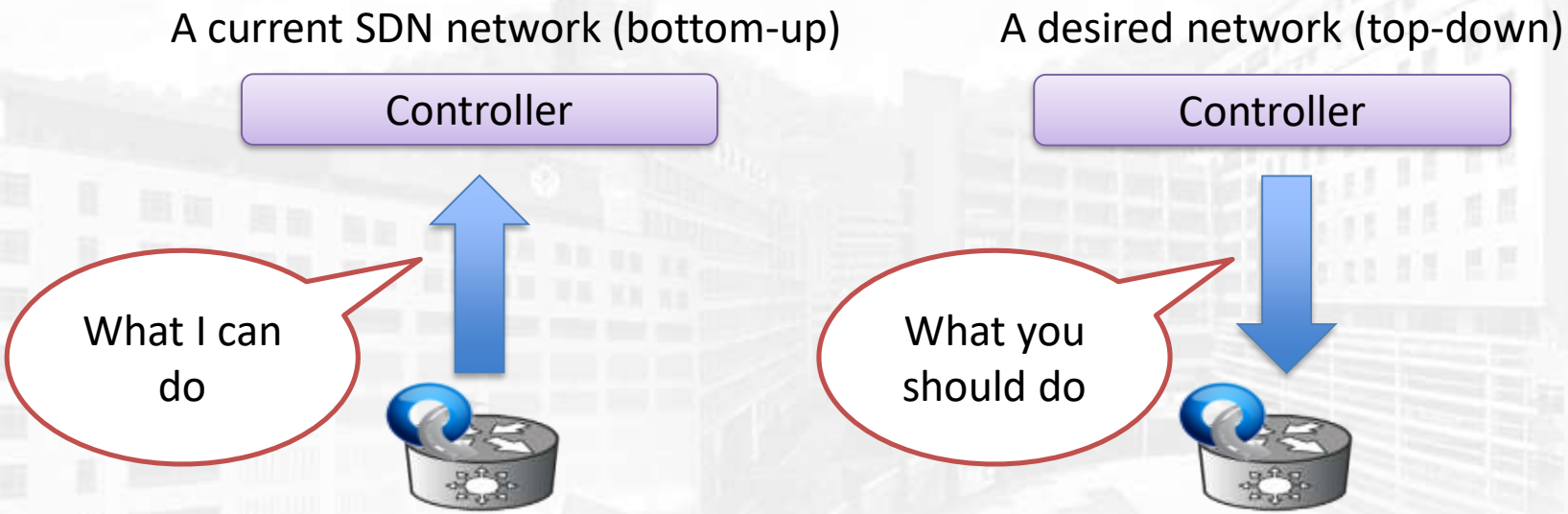
- The packet fields that can be matched are limited by OpenFlow protocol
- The actions we can take on packets are also limited
- OpenFlow switches reserve TCAM space for all possible fields → waste TCAM space

```
enum oxm_ofb_match_fields {
  OFPXMT_OFB_IN_PORT,
  OFPXMT_OFB_IN_PHY_PORT,
  OFPXMT_OFB_METADATA,
  OFPXMT_OFB_ETH_DST ,
  OFPXMT_OFB_ETH_SRC ,
  OFPXMT_OFB_ETH_TYPE ,
  OFPXMT_OFB_VLAN_VID ,
  OFPXMT_OFB_VLAN_PCP ,
  OFPXMT_OFB_IP_DSCP ,
  OFPXMT_OFB_IP_ECN ,
  OFPXMT_OFB_IP_PROTO ,
  OFPXMT_OFB_IPV4_SRC ,
  OFPXMT_OFB_IPV4_DST ,
  OFPXMT_OFB_TCP_SRC ,
  OFPXMT_OFB_TCP_DST ,
  OFPXMT_OFB_UDP_SRC ,
  OFPXMT_OFB_UDP_DST ,
  OFPXMT_OFB_SCTP_SRC ,
  OFPXMT_OFB_SCTP_DST ,
  OFPXMT_OFB_ICMPV4_TYPE ,
  OFPXMT_OFB_ICMPV4_CODE ,
  OFPXMT_OFB_ARP_OP ,
  OFPXMT_OFB_ARP_SPA ,
  OFPXMT_OFB_ARP_TPA ,
  OFPXMT_OFB_ARP_SHA ,
  OFPXMT_OFB_ARP_THA ,
  OFPXMT_OFB_IPV6_SRC ,
  OFPXMT_OFB_IPV6_DST ,
  OFPXMT_OFB_IPV6_FLABEL ,
  OFPXMT_OFB_ICMPV6_TYPE ,
  OFPXMT_OFB_ICMPV6_CODE ,
  OFPXMT_OFB_IPV6_ND_TARGET,
  OFPXMT_OFB_IPV6_ND_SLL,
  OFPXMT_OFB_IPV6_ND_TLL,
  OFPXMT_OFB_MPLS_LABEL,
  OFPXMT_OFB_MPLS_TC,
  OFPXMT_OFP_MPLS_BOS,
  OFPXMT_OFB_PBB_ISID,
  OFPXMT_OFB_TUNNEL_ID,
  OFPXMT_OFB_IPV6_EXTHDR,
  OFPXMT_OFB_PBB_UCA
};
```

```
enum ofp_action_type {
  OFPAT_OUTPUT,
  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_QUEUE,
  OFPAT_GROUP,
  OFPAT_SET_NW_TTL,
  OFPAT_DEC_NW_TTL,
  OFPAT_SET_FIELD,
  OFPAT_PUSH_PBB,
  OFPAT_POP_PBB,
  OFPAT_EXPERIMENTER
};
```

Network Designs

- A Bottom-up design V.S. a top-down design

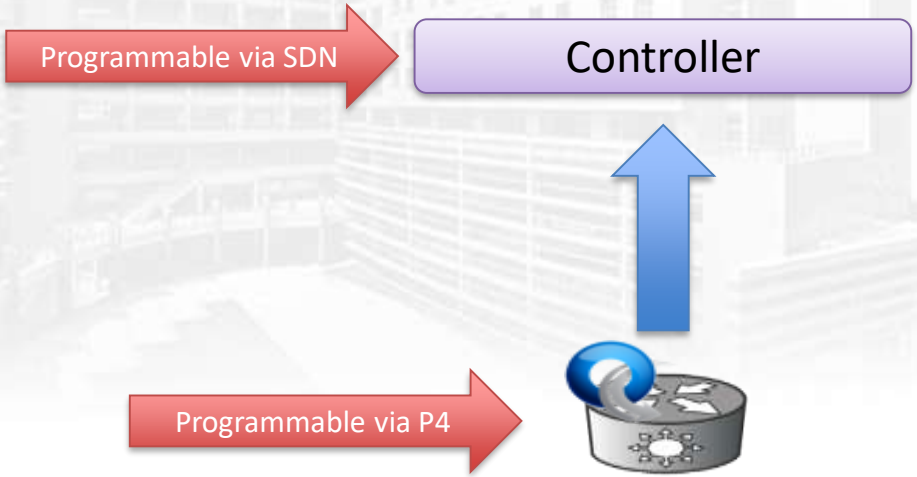


Agenda

- The limitation of SDN
- Why P4?
- Packet processing in P4 architecture
- Basic P4 programming
- Table matching and action in P4
- P4 + SDN: a more complete future network

What is P4

- P4 is a language to describe what a switch should do
- SDN → programmable control plane
- P4 → programmable data plane
- The goals:
  - Reconfigurability
  - Protocol-independence
  - Target Independence



Benefits of Data Plane Programmability

- **New Features** – Add new protocols
- **Reduce complexity** – Remove unused protocols
- **Efficient use of resources** – flexible use of tables
- **Greater visibility** – New diagnostic techniques, telemetry, etc.
- **SW style development** – rapid design cycle, fast innovation, fix data plane bugs in the field
- **You keep your own ideas**

Think programming rather than protocols…

Hardware for P4

- new custom ASICs can achieve such flexibility at terabit speeds [Kangaroo INFOCOM ’10, SDN Chip SIGCOMM ’13, Intel FM6000 switch silicon]
- some switches are more programmable than others:
  - FPGA (Xilinx, Altera, Corsa)
  - NPU (Ezchip, Netronome)
  - CPU (OVS, …)

Brief History

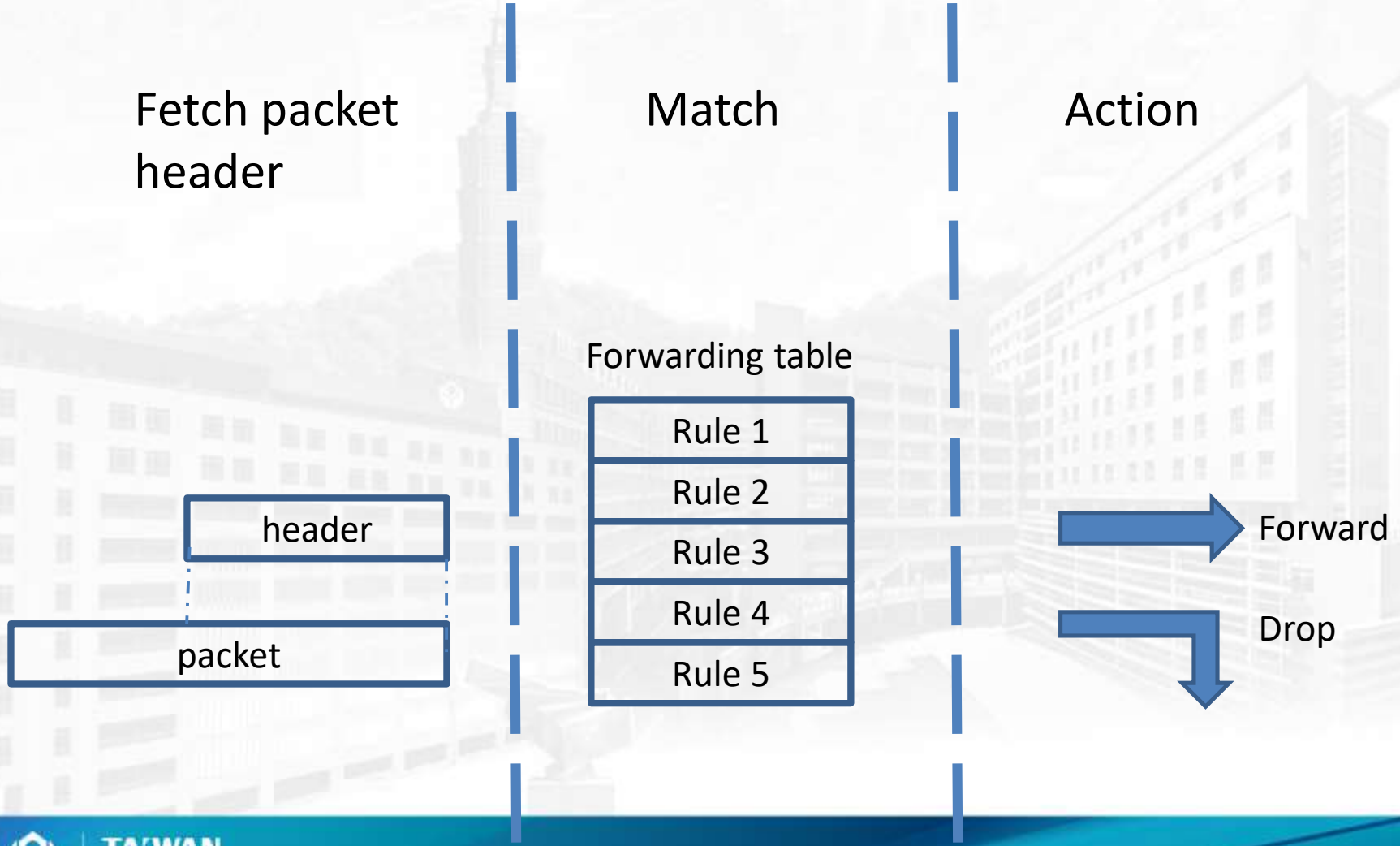
- **May 2013: Initial idea and the name "P4"**
- **July 2014: First paper (SIGCOMM ACR)**
- **Aug 2014: First P414 Draft Specification (v0.9.8)**
- **Sep 2014: P414 Specification released (v1.0.0)**
- **Jan 2015: P414 v1.0.1**
- **Mar 2015: P414 v1.0.2**
- **Nov 2016: P414 v1.0.3**
- **May 2017: P414 v1.0.4**

- **Apr 2016: P416 – first commits**
- **Dec 2016: First P416 Draft Specification**
- **May 2017: P416 Specification released**

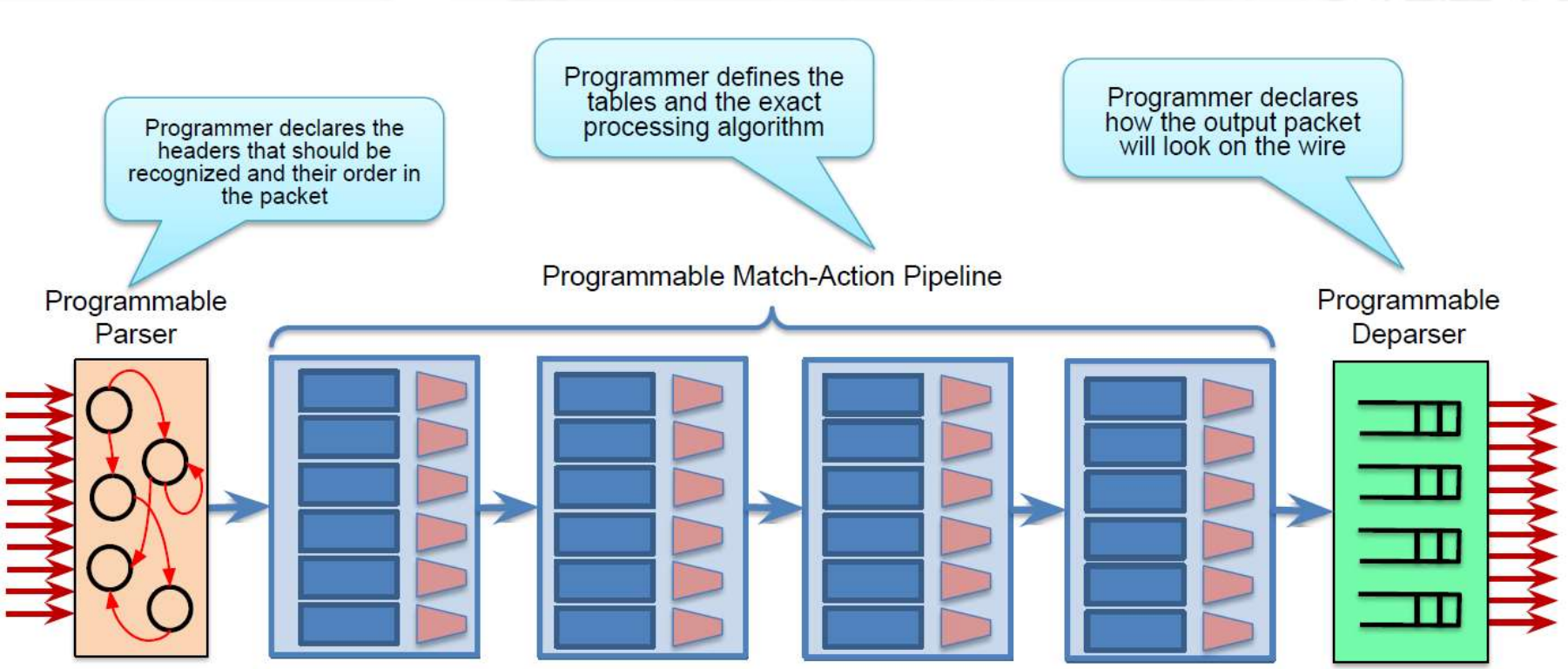
Agenda

- The limitation of SDN
- Why P4?
- Packet processing in P4 architecture
- Basic P4 programming
- Table matching and action in P4
- P4 + SDN: a more complete future network

Packet Processing in Switches



Protocol-Independent Switch Architecture



P4 Language Elements

Expressions

Basic operations and operators

Data Types

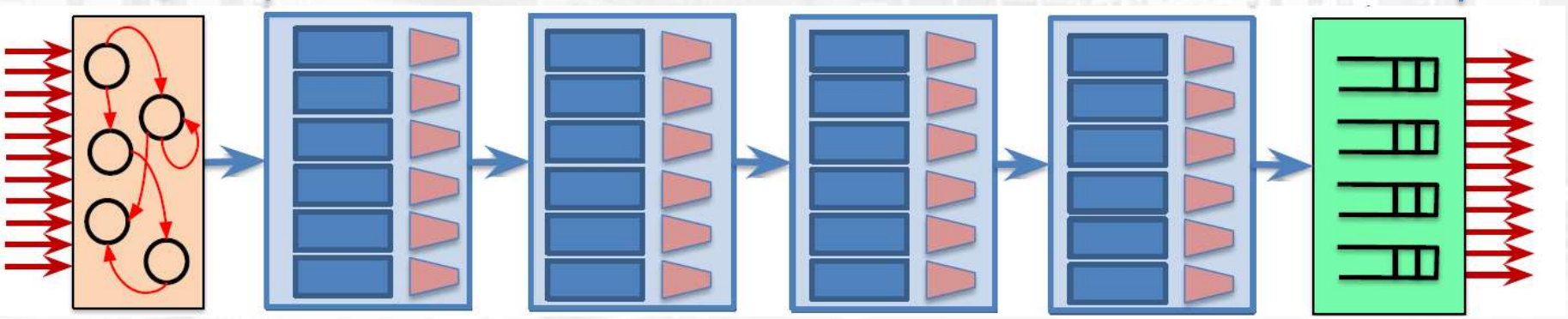
Bit strings, headers, structures, arrays

Parsers

State machine, bitfield extraction

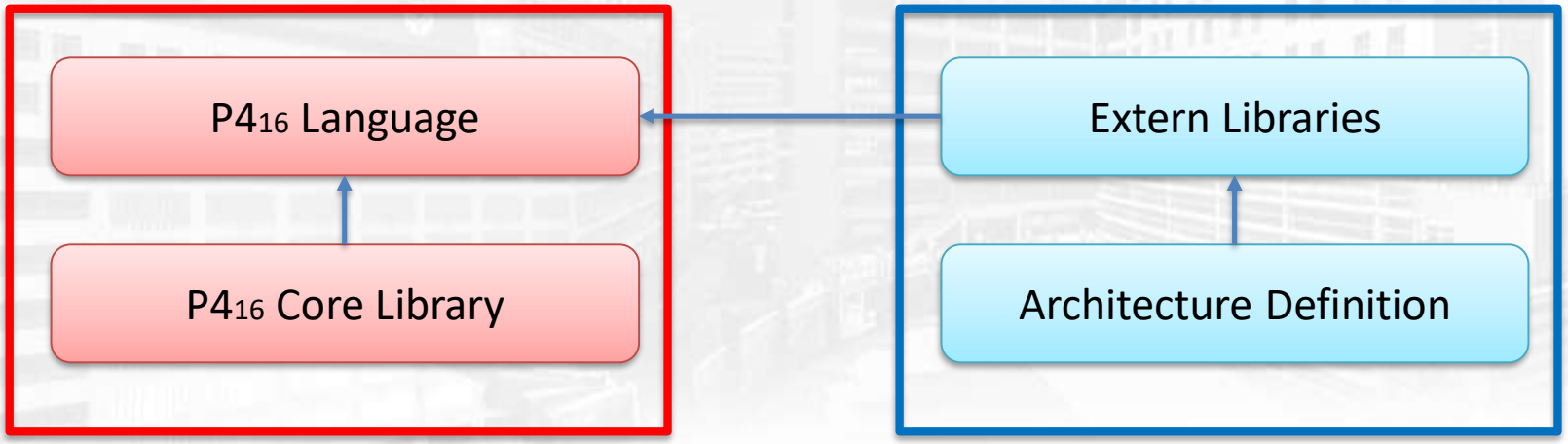
Controls

Tables, Actions, control flow statements

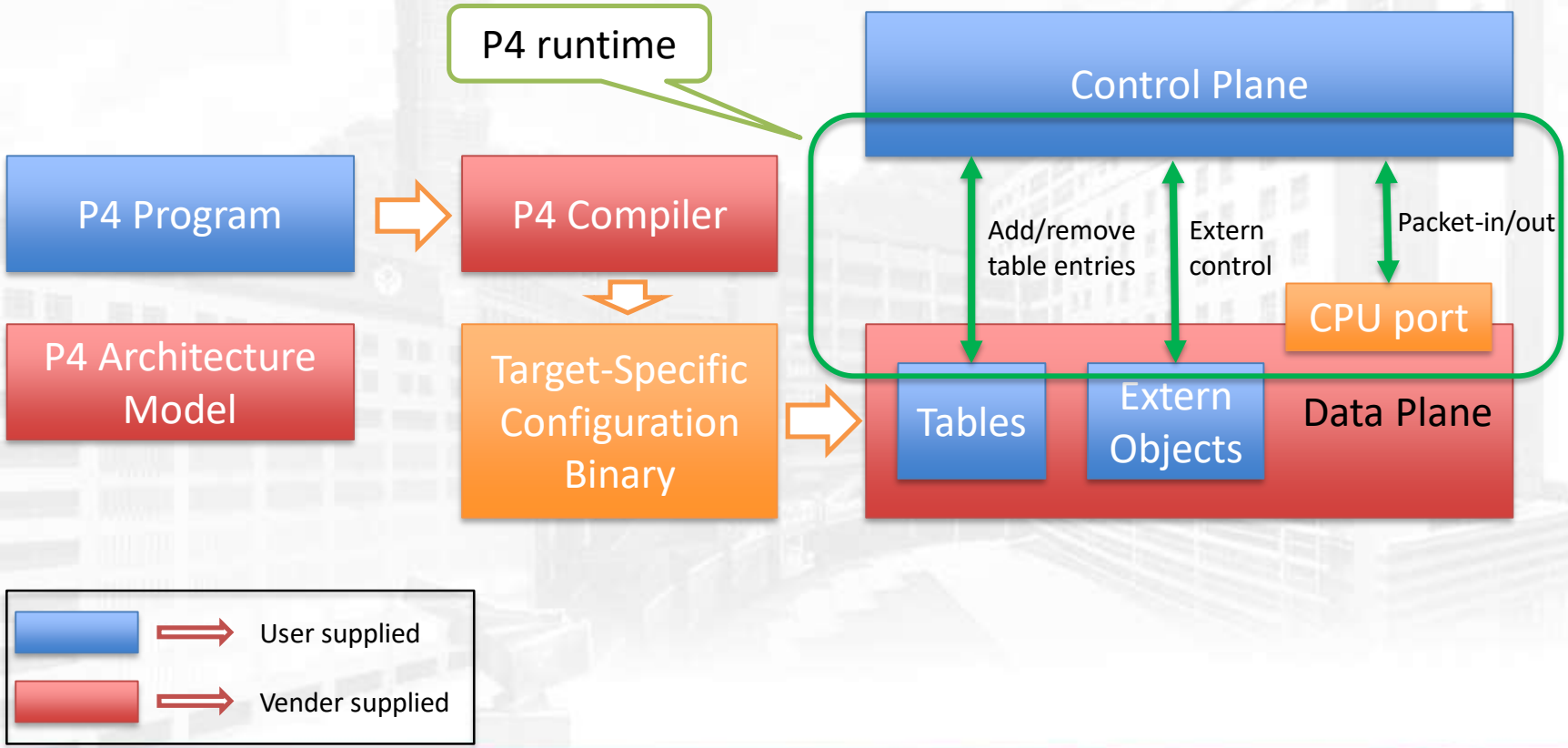


P4$_{16}$ Building Blocks

- P4 Target: An embodiment of a specific hardware implementation
- P4 Architecture: Provides an interface to program a target via some set of P4-programmable components, externs, fixed components

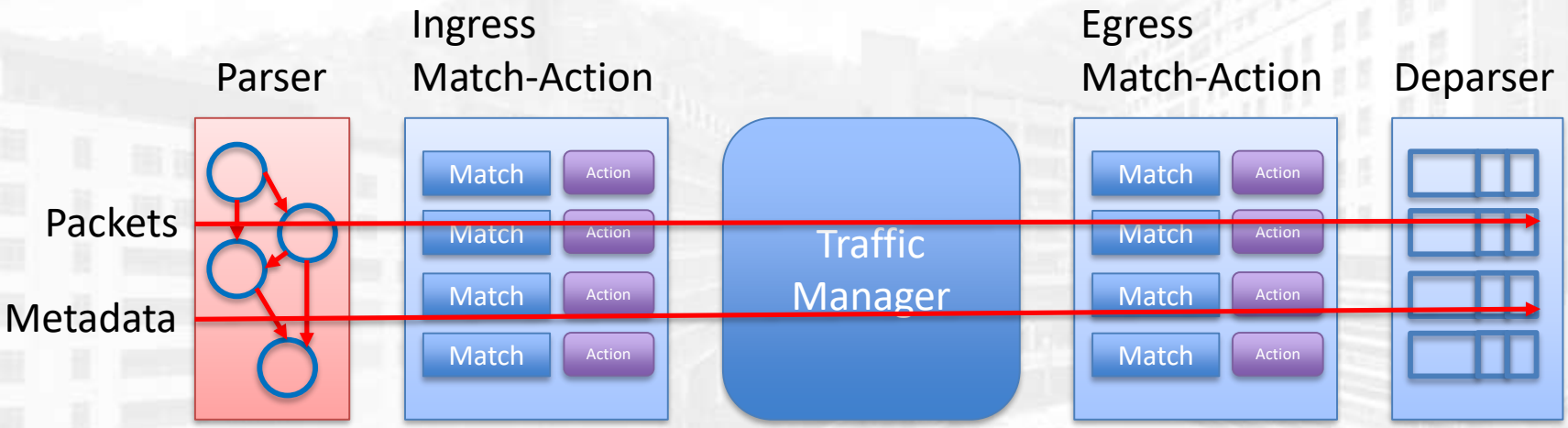


P4 Target Programming



V1Model Architecture

- Implemented on top of Bmv2's simple_switch target



Agenda

- The limitation of SDN
- Why P4?
- Packet processing in P4 architecture
- Basic P4 programming
- Table matching and action in P4
- P4 + SDN: a more complete future network

P4$_{16}$ Program Template (V1Model)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
ethernet_t ethernet;
ipv4_t ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
out headers hdr,
inout metadata meta,
inout standard_metadata_t smeta) {
...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(
inout metadata meta) {
...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
inout metadata meta,
inout standard_metadata_t std_meta) {
...
}
```

Egress match-action

Parser

Ingress match-action

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
inout metadata meta,
inout standard_metadata_t std_meta) {
...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
inout metadata meta) {
...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
inout metadata meta) {
...
}
/* SWITCH */
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

Deparser

Architecture description

V1Model Standard Metadata

```
struct standard_metadata_t {
bit<9> ingress_port;
bit<9> egress_spec;
bit<9> egress_port;
bit<32> clone_spec;
bit<32> instance_type;
bit<1> drop;
bit<16> recirculate_port;
bit<32> packet_length;
bit<32> enq_timestamp;
bit<19> enq_qdepth;
bit<32> deq_timedelta;
bit<19> deq_qdepth;
bit<48> ingress_global_timestamp;
bit<32> lf_field_list;
bit<16> mcast_grp;
bit<1> resubmit_flag;
bit<16> egress_rid;
bit<1> checksum_error;
}
```

- **ingress_port** - the port on which the packet arrived
- **egress_spec** - the port to which the packet should be sent to
- **egress_port** - the port on which the packet is departing from (read only in egress pipeline)

P4$_{16}$ Types (Basic and Header Types)

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
macAddr_t dstAddr;
macAddr_t srcAddr;
bit<16> etherType;
}
header ipv4_t {
bit<4> version;
bit<4> ihl;
bit<8> diffserv;
bit<16> totalLen;
bit<16> identification;
bit<3> flags;
bit<13> fragOffset;
bit<8> ttl;
bit<8> protocol;
bit<16> hdrChecksum;
ip4Addr_t srcAddr;
ip4Addr_t dstAddr;
}
```

Basic Types

- **bit<n>**: Unsigned integer (bitstring) of size n
- **bit** is the same as **bit<1>**
- **int<n>**: Signed integer of size n (>=2)
- **varbit<n>**: Variable-length bitstring

Header Types: Ordered collection of members

- Can contain **bit<n>**, **int<n>**, and **varbit<n>**
- Byte-aligned
- Can be valid or invalid
- Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**

Typedef: Alternative name for a type

P4$_{16}$ Types (Other Types)

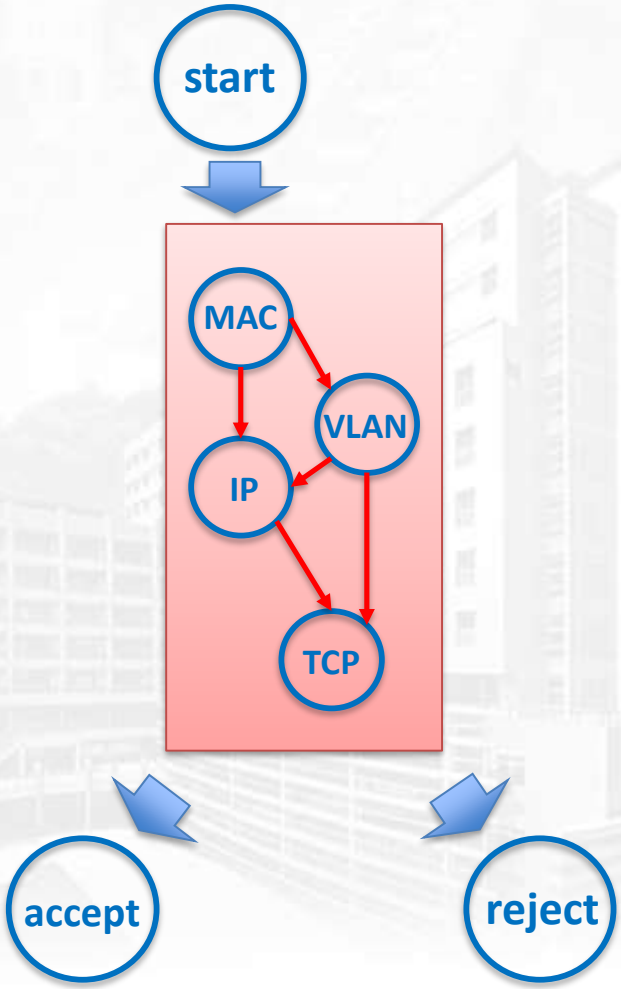
```
/* Architecture */
struct standard_metadata_t {
bit<9> ingress_port;
bit<9> egress_spec;
bit<9> egress_port;
bit<32> clone_spec;
bit<32> instance_type;
bit<1> drop;
bit<16> recirculate_port;
bit<32> packet_length;
...
}
/* User program */
struct metadata {
...
}
struct headers {
ethernet_t ethernet;
ipv4_t ipv4;
}
```

Other useful types

- **Struct**: Unordered collection of members (with no alignment restrictions)
- **Header Stack:** array of headers
- **Header Union:** one of several headers

P4$_{16}$ Parsers

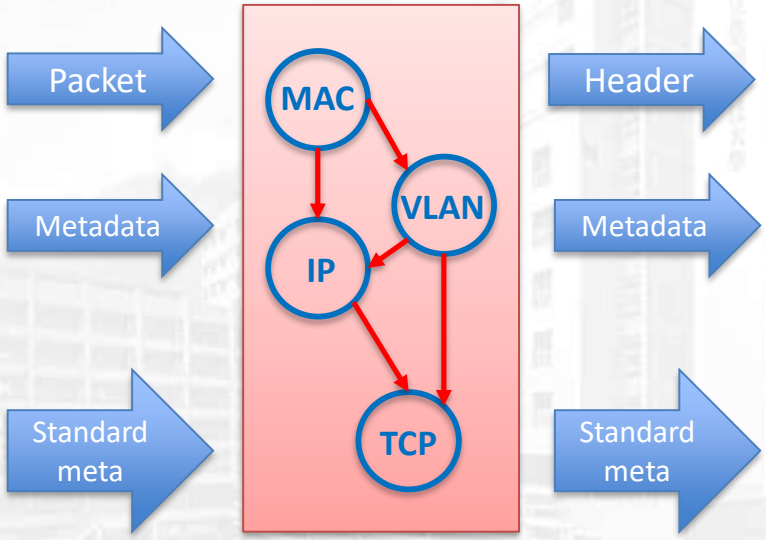
- Parsers are functions that map packets into headers and metadata, written in a state machine style
- Every parser has three predefined states
  - start
  - accept
  - reject
- Other states may be defined by the programmer
- In each state, execute zero or more statements, and then transition to another state (loops are OK)



Parser Codes

```
/* From core.p4 */
extern packet_in {
void extract<T>(out T hdr);
void extract<T>(out T variableSizeHeader,
in bit<32> variableFieldSizeInBits);
T lookahead<T>();
void advance(in bit<32> sizeInBits);
bit<32> length();
}
/* User Program */
parser MyParser(packet_in packet,
out headers hdr,
inout metadata meta,
inout standard_metad
state start {
packet.extract(hdr.ethernet);
transition accept;
}
}
```

Extract packet header



State Transition

```
state start {
transition parse_ethernet;
}
state parse_ethernet {
packet.extract(hdr.ethernet);
transition
select(hdr.ethernet.etherType) {
0x800: parse_ipv4;
default: accept;
}
}
```

- P416 has a select statement that can be used to branch in a parser
- Similar to case statements in C or Java, but without “fall-through behavior”—i.e., break statements are not needed
- In parsers it is often necessary to branch based on some of the bits just parsed
- For example, etherType determines the format of the rest of the packet
- Match patterns can either be literals or simple computations such as masks

P4$_{16}$ Controls

- Similar to C functions (without loops)
- Can declare variables, create tables, instantiate externs, etc.
- Functionality specified by code in apply statement
- Represent all kinds of processing that are expressible as DAG:
  - Match-Action Pipelines
  - Deparsers
  - Additional forms of packet processing (updating checksums)
- Interfaces with other blocks are governed by user- and architecture-specified types (typically headers and metadata)

P4$_{16}$ Controls: an Example (Reflector)

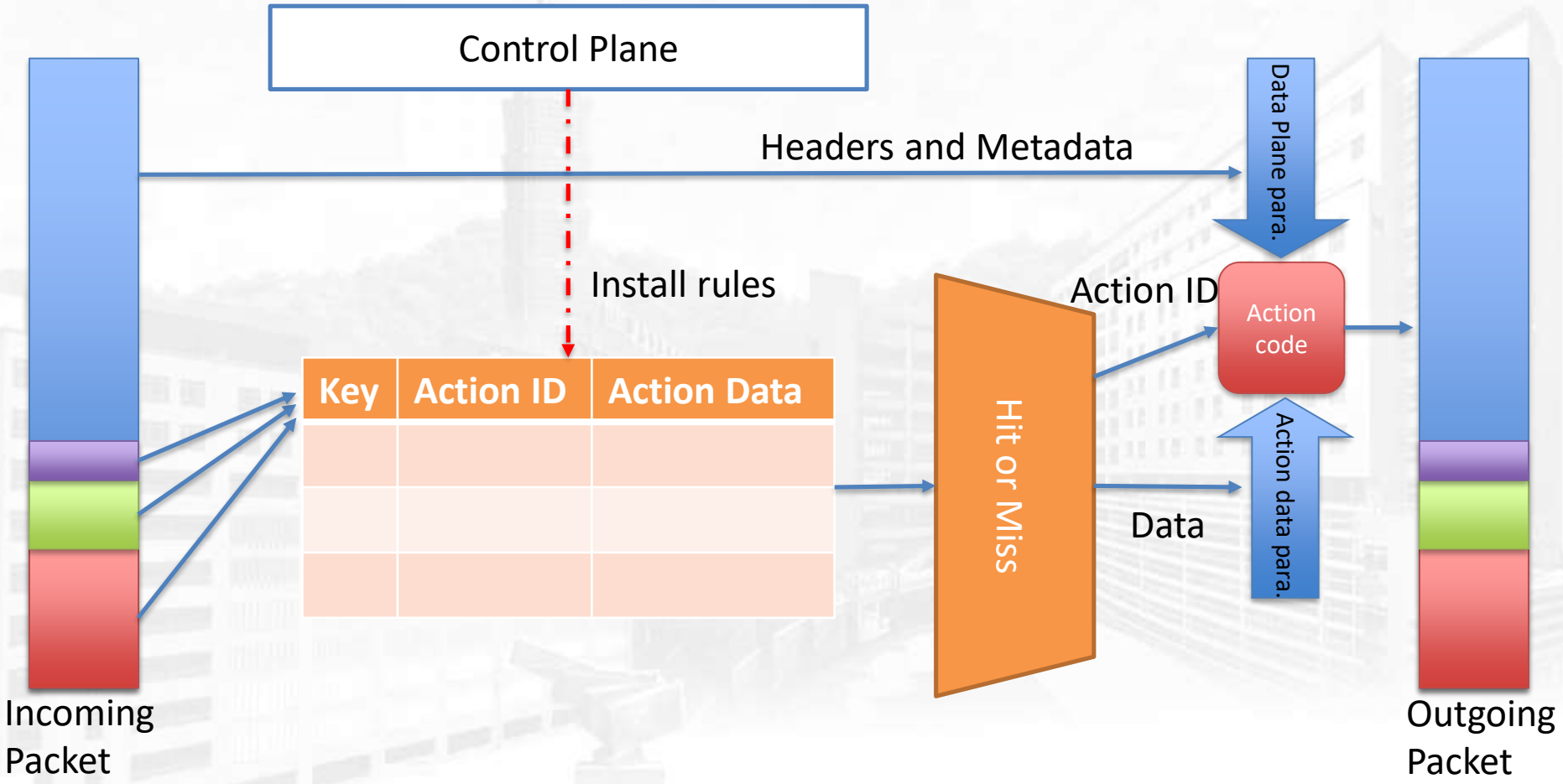
```
control MyIngress(inout headers hdr,
inout metadata meta,
inout standard_metadata_t std_meta) {
action swap_mac(inout bit<48> src,
inout bit<48> dst) {
bit<48> tmp = src;
src = dst;
dst = tmp;
}
apply {
swap_mac(hdr.ethernet.srcAddr,
hdr.ethernet.dstAddr);
std_meta.egress_spec = std_meta.ingress_port;
}
}
```

- Very similar to C functions
- Can be declared inside a control or globally
- Parameters have type and direction
- Variables can be instantiated inside
- Many standard arithmetic and logical operations are supported
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- Non-standard operations:
  - Bit-slicing: [m:l] (works as l-value too)
  - Bit Concatenation: ++

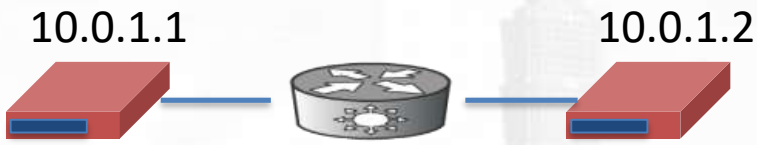
Agenda

- The limitation of SDN
- Why P4?
- Packet processing in P4 architecture
- Basic P4 programming
- Table matching and action in P4
- P4 + SDN: a more complete future network

Tables: Match-Action Processing



Example: IPv4_LPM Table



Key	Action	Action Data
10.0.1.1/32	Ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	Drop	
*	NoAction	

- Data Plane (P4) Program
  - Defines the format of the table
    - Key Fields
    - Actions
    - Action Data
  - Performs the lookup
  - Executes the chosen action
- Control Plane (IP stack, Routing protocols)
  - Populates table entries with specific information
    - Based on the configuration
    - Based on automatic discovery
    - Based on protocol calculations

IPv4_LPM Table in P4 Code

```
table ipv4_lpm {
key = {
hdr.ipv4.dstAddr: lpm;
}
actions = {
ipv4_forward;
drop;
NoAction;
}
size = 1024;
default_action = NoAction();
}
```

```
/* core.p4 */
match_kind {
exact,
ternary,
lpm
}
```

Defining Actions for L3 forwarding

```
/* core.p4 */
action NoAction() {
}
/* basic.p4 */
action drop() {
mark_to_drop();
}
/* basic.p4 */
action ipv4_forward(macAddr_t
dstAddr,
bit<9> port) {
...
}
```

- Actions can have two different types of parameters
  - Directional (from the Data Plane)
  - Directionless (from the Control Plane)
- Actions that are called directly:
  - Only use directional parameters
- Actions used in tables:
  - Typically use directionless parameters
  - May sometimes use directional parameters too

Applying Tables in Controls

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
        table ipv4_lpm {
                ...
        }
        apply {
                ...
                ipv4_lpm.apply();
                ...
        }
}
```

P4$_{16}$ Deparsing

```
/* From core.p4 */
extern packet_out {
        void emit<T>(in T hdr);
}
/* User Program */
control DeparserImpl(packet_out packet,
        in headers hdr) {
        apply {
                ...
                packet.emit(hdr.ethernet);
                ...
        }
}
```

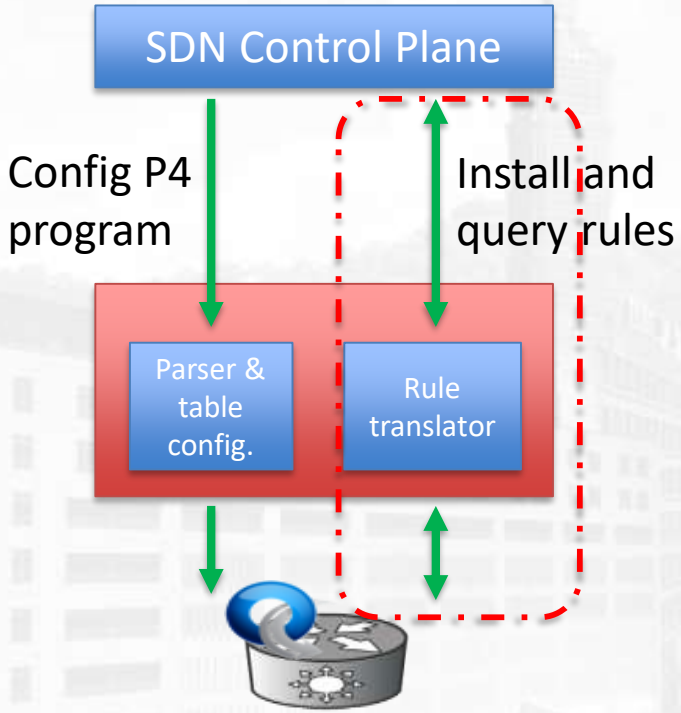
- Assembles the headers back into a well-formed packet
- Expressed as a control function
  - No need for another construct!
- packet_out extern is defined in core.p4: emit(hdr): serializes header if it is valid
- Advantages:
  - Makes deparsing explicit...

    but decouples from parsing

Agenda

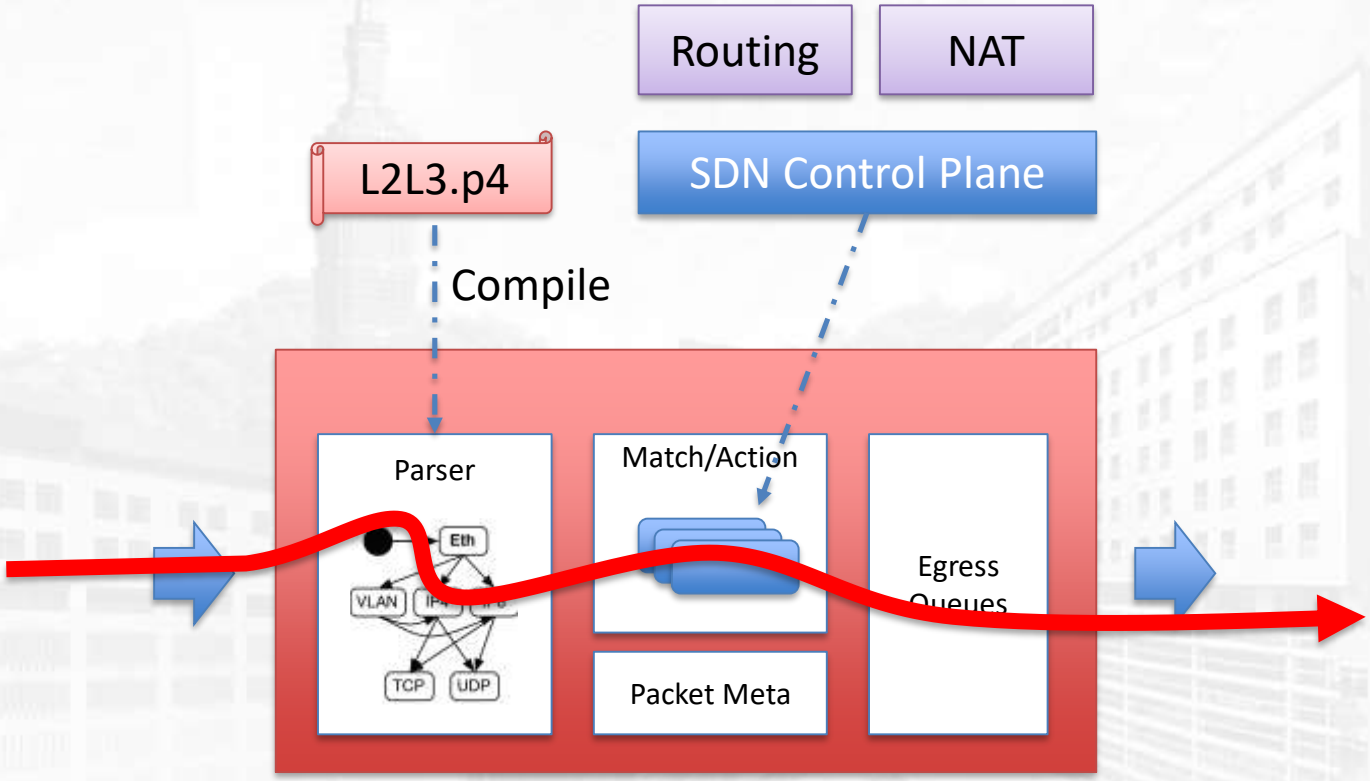
- The limitation of SDN
- Why P4?
- Packet processing in P4 architecture
- Basic P4 programming
- Table matching and action in P4
- P4 + SDN: a more complete future network

SDN + P4 as a Whole

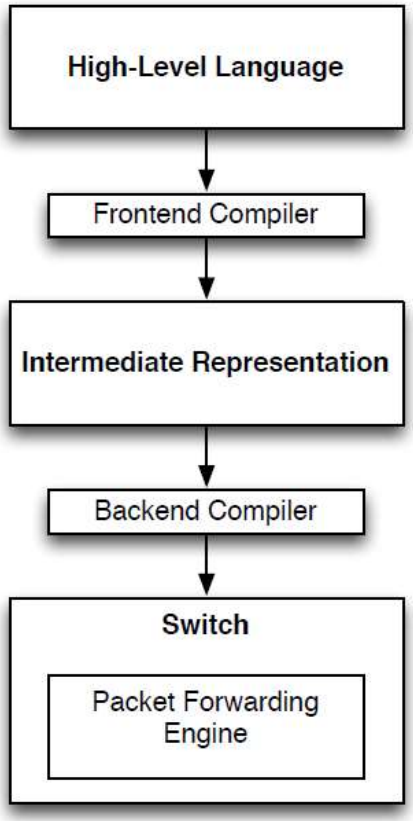


- P4 program configures forwarding behavior (abstract forwarding model)
- express serial dependencies (e.g. ARP/L3 Routing)
- P4 compiler translates into a target-specific representation
- OF can still be used to install and query rules once forwarding model is defined

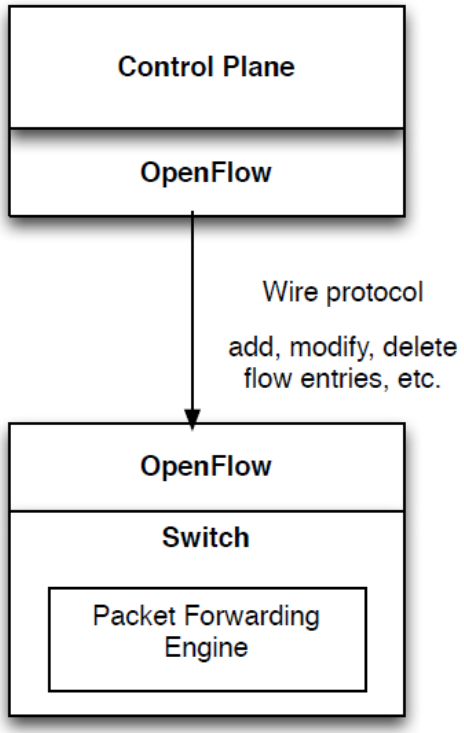
P4 Forwarding Model / Runtime



Conclusion


High-Level Language
Frontend Compiler
Intermediate Representation
Backend Compiler
Switch
Packet Forwarding Engine

Conclusion


Control Plane
OpenFlow
Wire protocol
add, modify, delete
flow entries, etc.
OpenFlow
Switch
Packet Forwarding
Engine