Match-action Tables

Jorge Crichigno

College of Engineering and Computing, University of South Carolina

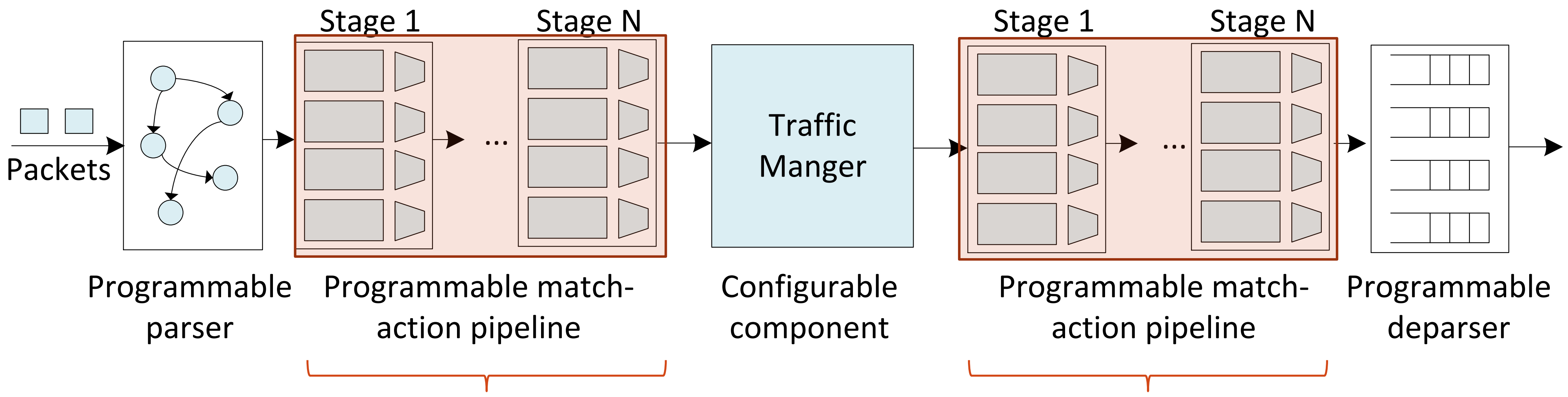
A Hands-on Tutorial on P4 Programmable Data Planes

Tuesday March 7, 2023

Match-action Tables

Match-action Pipeline

- Tables are the fundamental unit of a Match-Action Pipeline; they define the processing logic inside the match-action pipeline
- They can be used to implement traditional switch tables (e.g., routing, flow lookup, access-control lists)
- They can implement custom user-defined complex logic



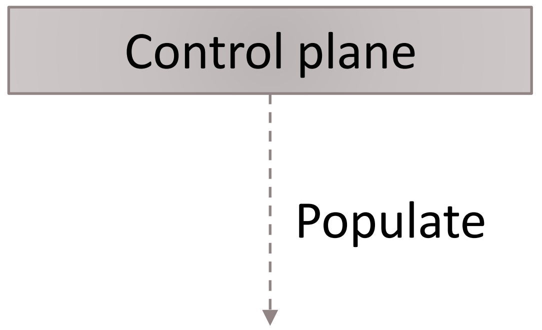
Tables are implemented here

Tables are implemented here

Match-action Table

- Specifies what data to match on
- Specifies a list of possible actions
- Optionally specifies a number of table properties; e.g.,
  - Size
  - Default action
  - Static entries
- An entry contains
  - A specific key to match on
  - An action that is executed when a packet matches the entry
  - Action data (possibly empty)

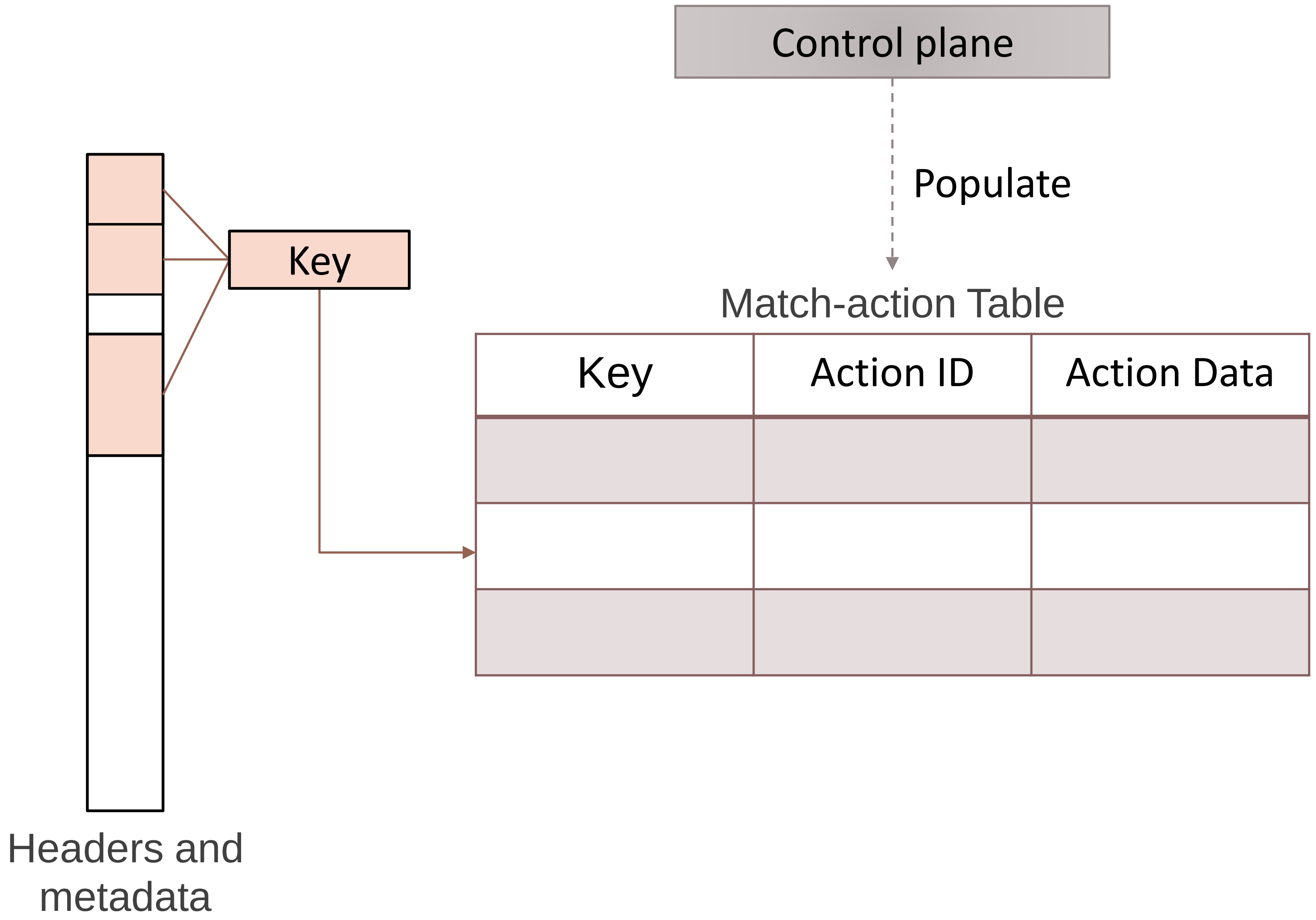
Match-action Table



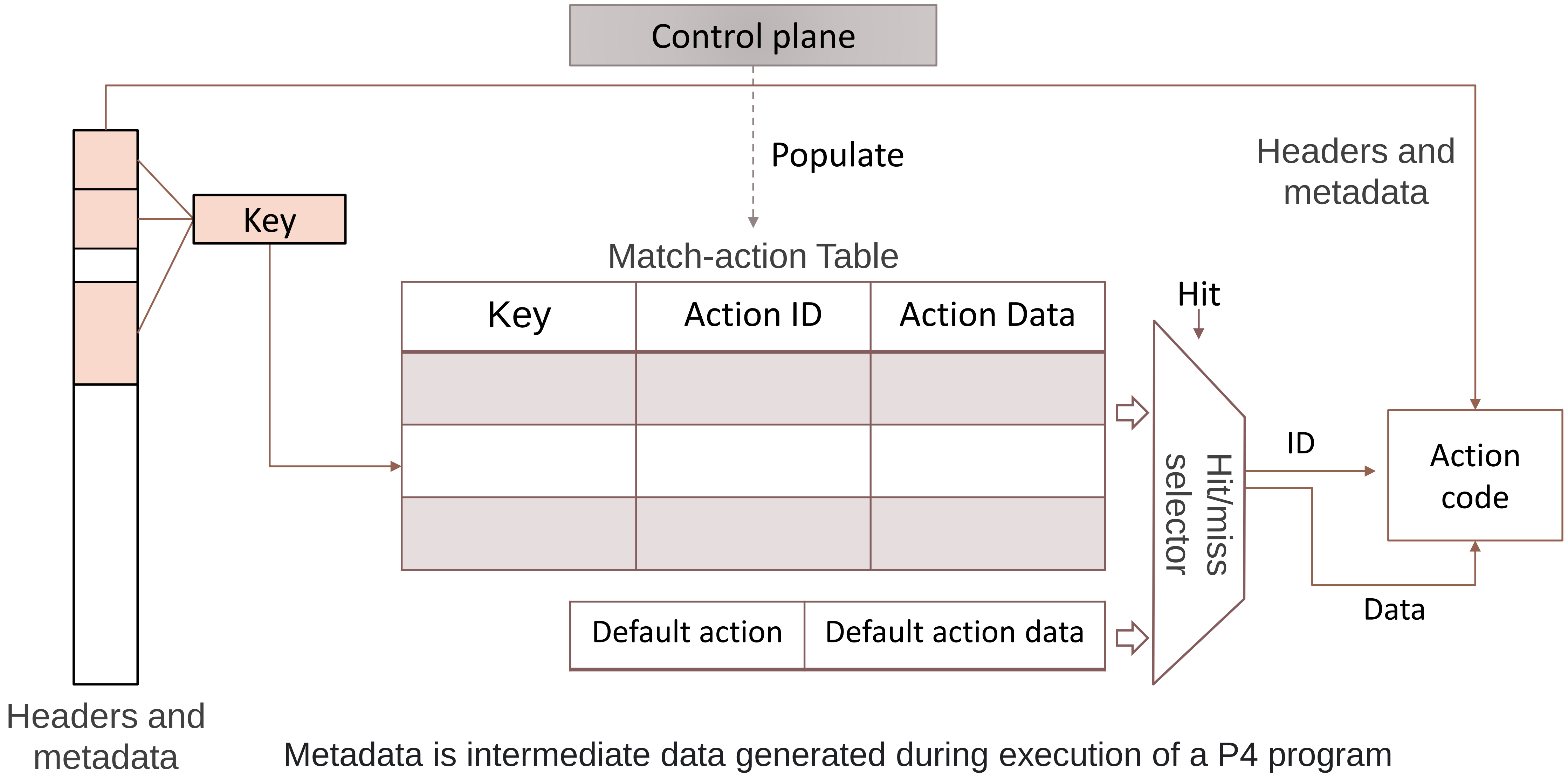
Match-action Table

Key	Action ID	Action Data

Match-action Table



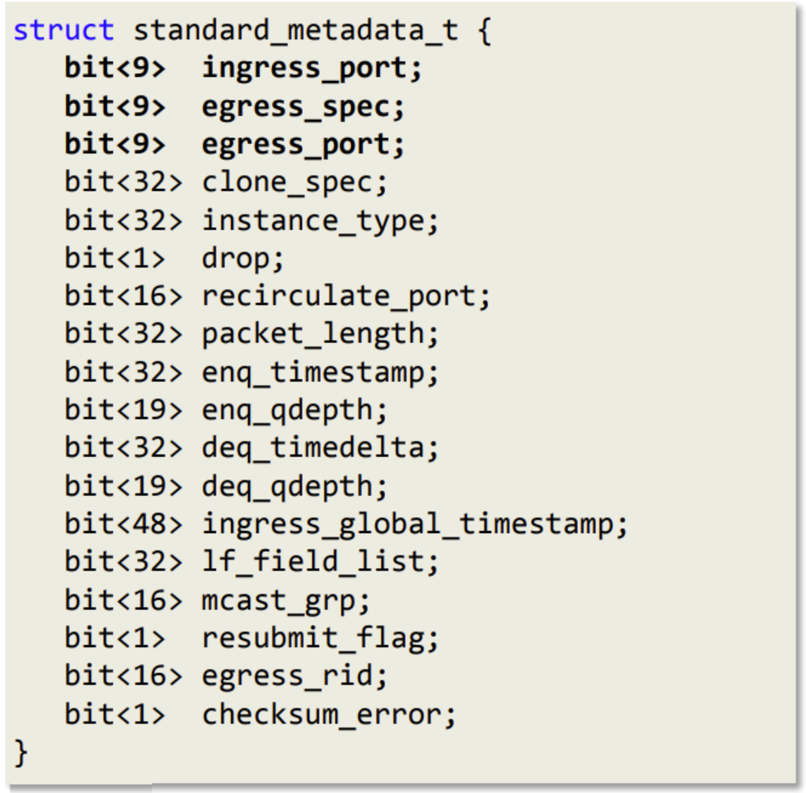
Match-action Table



Metadata is intermediate data generated during execution of a P4 program

Match-action Table

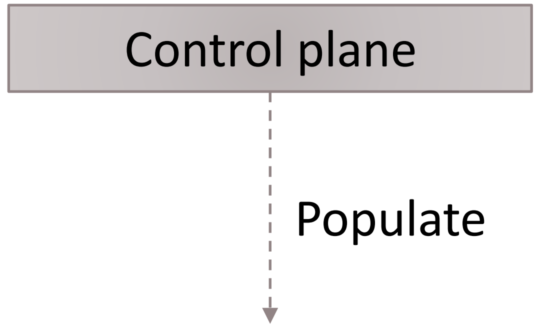
- Metadata is intermediate data generated during execution of a P4 program
- Standard metadata - data that must be provided by targets
  - `ingress_port`: port on which the packet arrived
  - `egress_spec`: port to which the packet should be sent to
  - `egress_port`: port on which the packet is departing from (read only in egress pipeline; useful value on ingress pipeline only)



```
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;
}
```

V1 model standard metadata

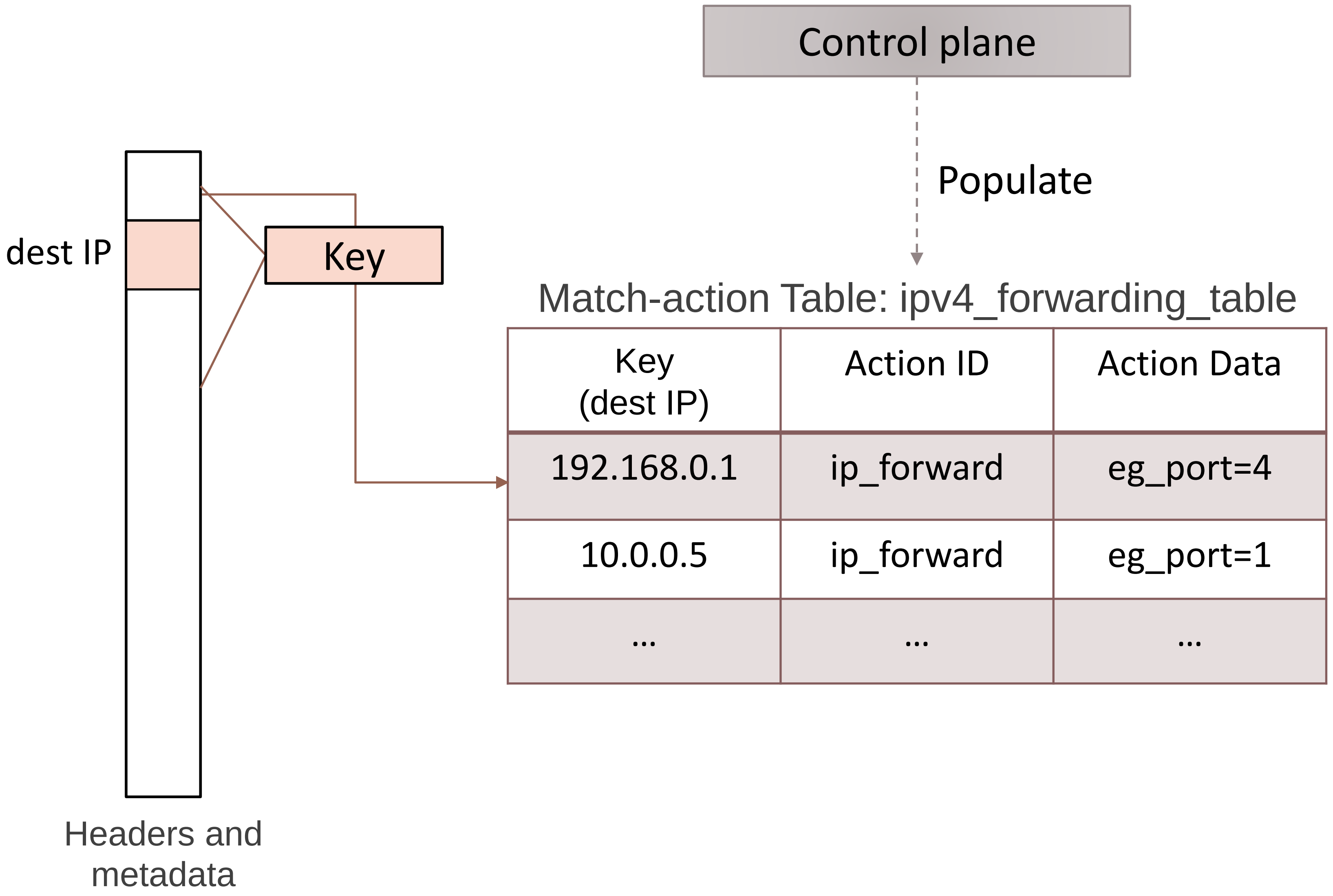
Example: IPv4 Forwarding



Match-action Table: ipv4_forwarding_table

Key (dest IP)	Action ID	Action Data
192.168.0.1	ip_forward	eg_port=4
10.0.0.5	ip_forward	eg_port=1
…	…	…

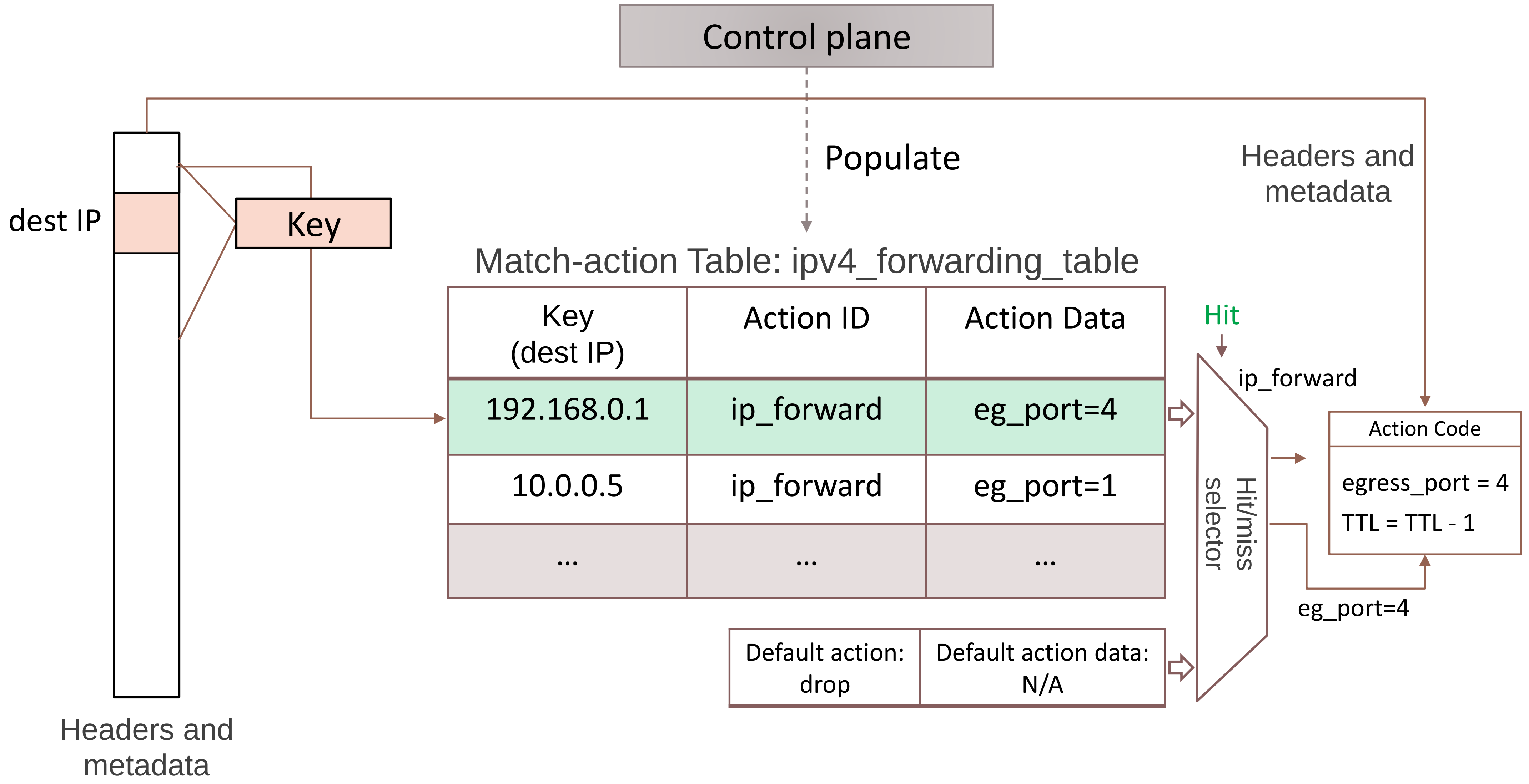
Example: IPv4 Forwarding



Match-action Table: ipv4_forwarding_table

Key (dest IP)	Action ID	Action Data
192.168.0.1	ip_forward	eg_port=4
10.0.0.5	ip_forward	eg_port=1
…	…	…

Example: IPv4 Forwarding



Key (dest IP)	Action ID	Action Data
192.168.0.1	ip_forward	eg_port=4
10.0.0.5	ip_forward	eg_port=1
…	…	…

Default action: drop	Default action data: N/A

Controls

- Similar to C functions (without loops)
- Can declare tables, variables
- Functionality specified by code in `apply` statement

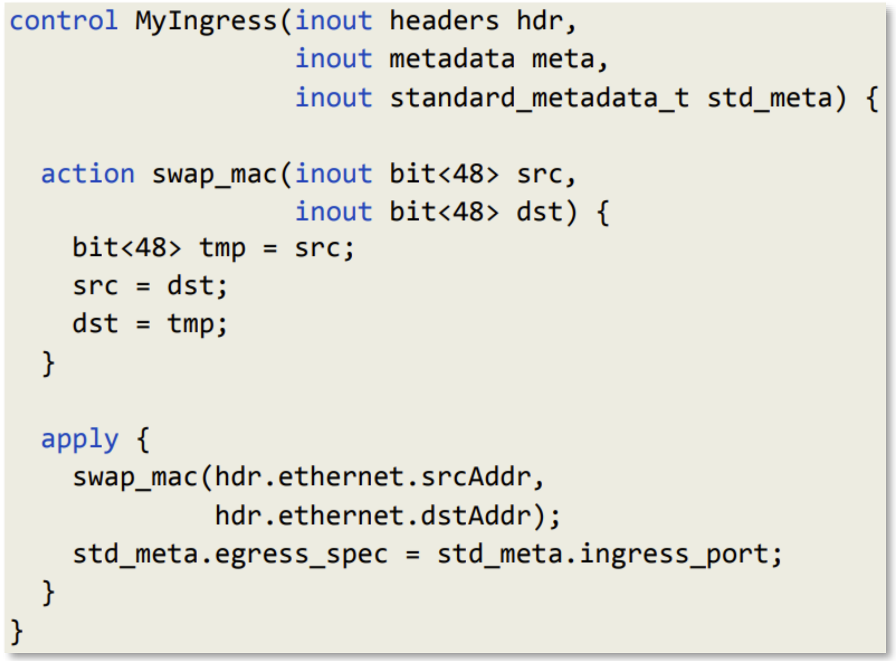
```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  bit<48> tmp;
  apply {
    tmp = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
    hdr.ethernet.srcAddr = tmp;
    std_meta.egress_spec =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

Swap source and destination MAC addresses

Bounce the packet back out on the physical port that it came into the switch on

Actions

- Similar to C functions
- Can be declared inside a control or globally
- Parameters have type and direction



```
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;
  }
}
```

Swap source and destination MAC addresses

Bounce the packet back out on the physical port that it came into the switch on