Registers, Packet Digests

Jorge Crichigno[1], Mariam Kiran[2]
[1]University of South Carolina, [2]ESnet

Lab Assistants: Elie Kfoury, Ali AlSabeh, Jose Gomez
University of South Carolina

WASTC 2022 virtual Faculty Development Weeks (vFDW)
June 17, 2022

Registers

Registers

- Registers are stateful memories whose values can be read and written in actions in the data plane
- They can also be read and written by the control plane
- They are more general than counters; arbitrary data can be stored in registers
- Registers are global memory resources; any match-action tables can reference to them

Registers in V1 Model

- The definition of the V1 Model register includes
  - `register` instantiation that receives an input parameter –number of elements of the register

```
/* Definition in v1model.p4 */

extern register<T> {
    register(bit<32> instance_count);
    void read(out T result, in bit<32> index);
    void write(in bit<32> index, in T value);
}
```

1. V. Gurevich, Introduction to P416. Online: https://tinyurl.com/2h93pnyd

Registers in V1 Model

- The definition of the V1 Model register includes
  - `register` instantiation that receives an input parameter –number of elements of the register
  - `read` method that receives an output parameter –where to store the register value– and an input parameter –index

```
/* Definition in v1model.p4 */

extern register<T> {
    register(bit<32> instance_count);
    void read(out T result, in bit<32> index);
    void write(in bit<32> index, in T value);
}
```

1. V. Gurevich, Introduction to P416. Online: https://tinyurl.com/2h93pnyd

Registers in V1 Model

- The definition of the V1 Model register includes
  - `register` instantiation that receives an input parameter –number of elements of the register
  - `read` method that receives an output parameter –where to store the register value– and an input parameter –index
  - `write` method that receives two input parameters, index and value to store in the register

```
/* Definition in v1model.p4 */

extern register<T> {
    register(bit<32> instance_count);
    void read(out T result, in bit<32> index);
    void write(in bit<32> index, in T value);
}
```

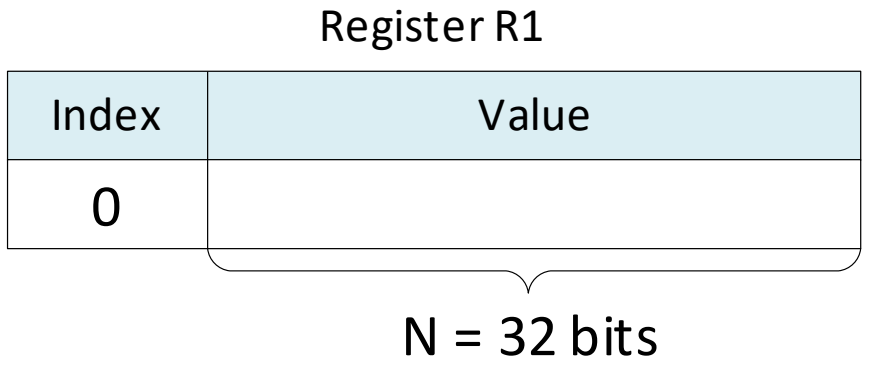
1. V. Gurevich, Introduction to P416. Online: https://tinyurl.com/2h93pnyd

Instantiating a Single Element Register

- The syntax below shows how to instantiate a single element register in P4

```
register<bit<N>>(1) R1;
```

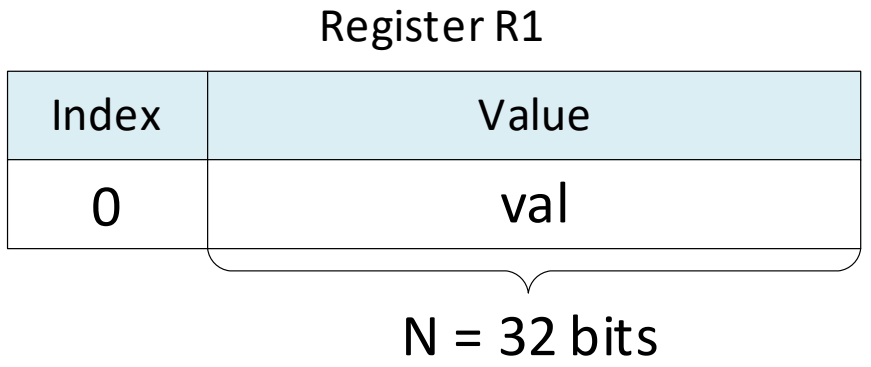
- Register R1 contains a single N-bit element



Writing a Single Element Register

- The syntax below shows how to write (store) a value *val* in register R1, element 0

```
R1.write(0,val)
```

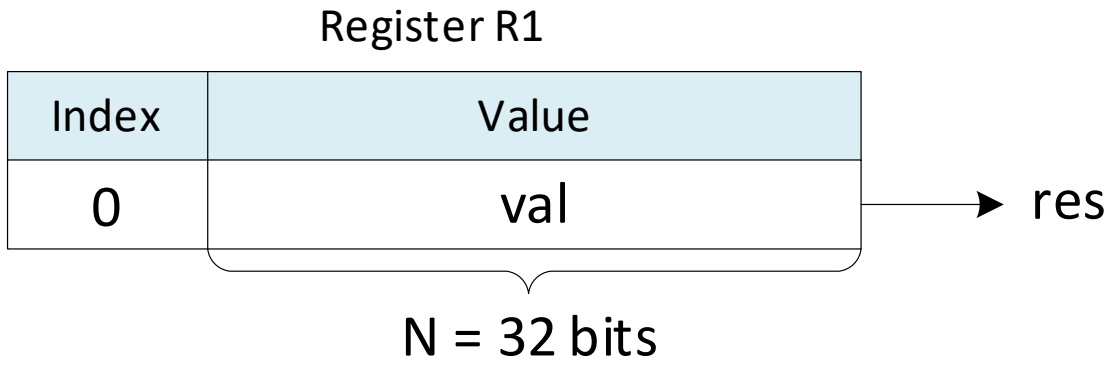


Reading a Single Element Register

- The syntax below shows how to read the value stored in element 0 of the register, and store it into the variable `res`

```
R1.read(res,0)
```

- Note that the value `val` is stored in the variable `res`



Registers in V1 Model

- Example: computing the time between two consecutive packets of a flow (inter-packet gap)

Code

```
register<bit<48>>(16384) last_seen;

action get_inter_packet_gap(out bit<48> interval, bit<32> flow_id)
{
 bit<48> last_pkt_ts;

 /* Get the time the previous packet was seen */
 last_seen.read(last_pkt_ts, flow_id);

 /* Calculate the time interval */
 interval = standard_metadata.ingress_global_timestamp - last_pkt_ts;

 /* Update the register with the new timestamp */
 last_seen.write(flow_id, standard_metadata.ingress_global_timestamp);

 ...
}
```

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;
}
```

1. L. Vanbever, Programming Network Data Planes. Online: https://tinyurl.com/ywr3c6rb

Atomicity

- Hardware and software targets use atomic operations on P4 stateful objects
- For Intel Tofino switch (Vladimir Gurevich)[1]:

> In case of stateful objects, a complex read-modify-write operation counts as one access and is performed by a special ALU (counter ALU, meter ALU, stateful ALU, etc.)
>
> Since this counts as one operation, it is atomic for all practical intents and purposes. For example, it is impossible to see a stateful object in some "intermediate" state. Similarly, when the same object (instance) is accessed by the next packet, it does see it fully modified.

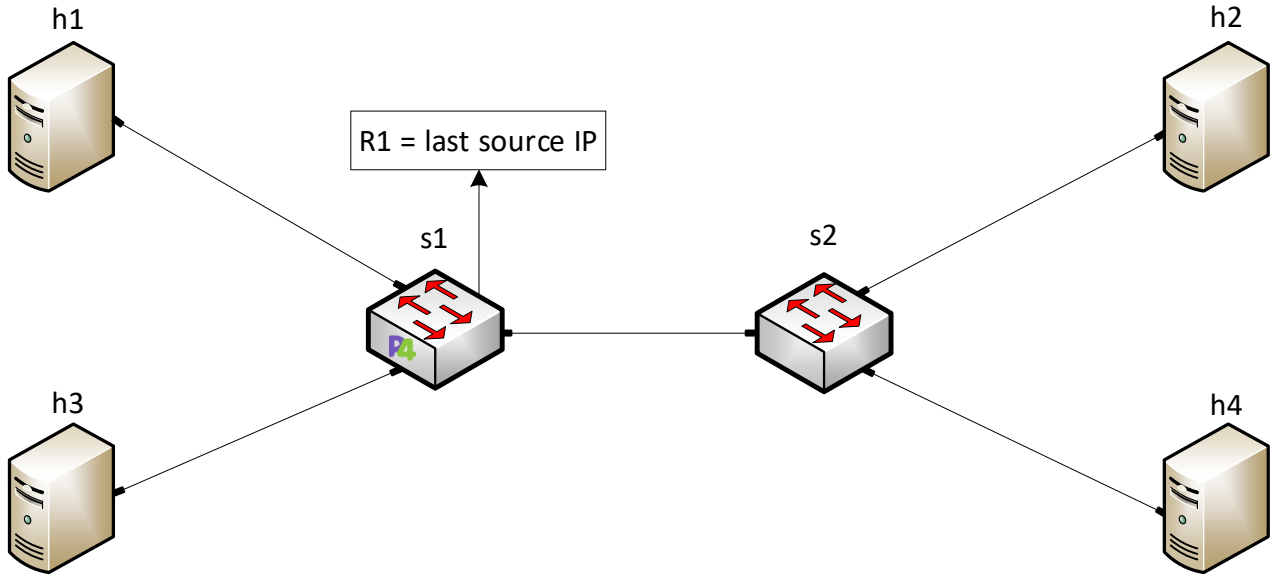
- For BMv2 v1model implementation[2]:

> The BMv2 v1model implementation supports parallel execution. It uses locking of all register objects accessed within an action to guarantee that the execution of all steps within an action are atomic, relative to other packets executing the same action, or any action that accesses some of the same register objects.

1. Intel® Connectivity Research Program (Private). Memory semantics of Tofino architecture. Online: https://tinyurl.com/yz7hzydr
2. P4Lang Consortium, The BMv2 Simple Switch target. Online: https://tinyurl.com/26b762m3

Lab 9 Topology and Objectives

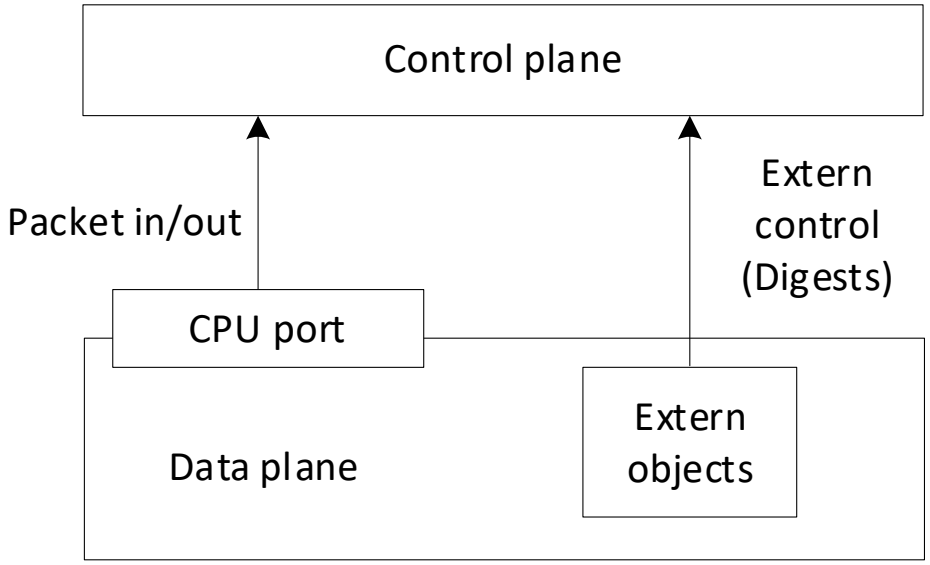
- This lab requires the learner to write a P4 program that stores the last observed source IP address into a register
- The topology consists of four hosts, one P4 switch, and one legacy switch
- The objectives are
  - Be able to write P4 programs using registers
  - Read, write, and reset registers from the control plane



Packet Digests

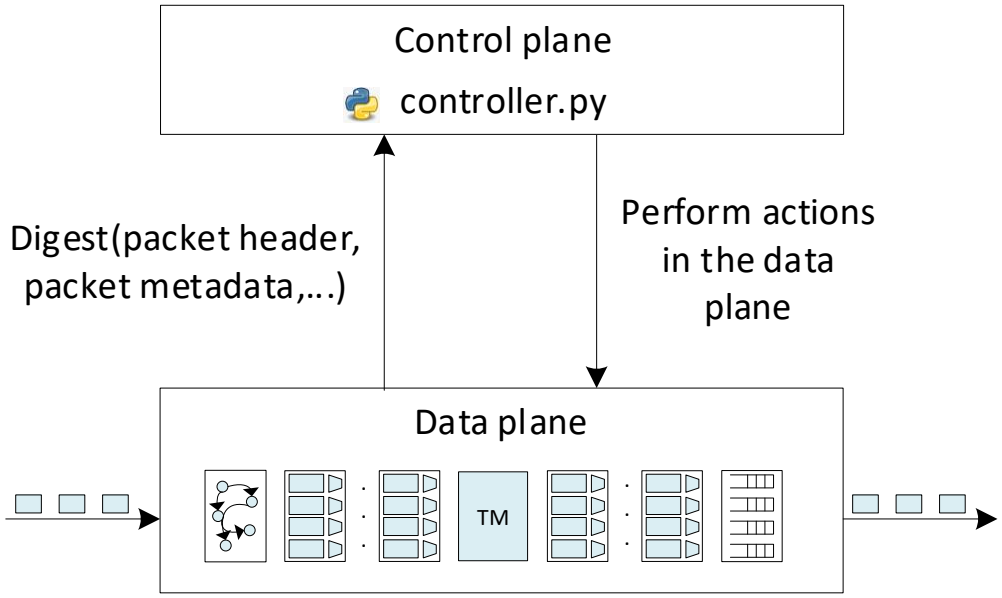
Data Plane to Control Plane Communication

- The data plane can send a packet to the control plane via a particular port reserve for this purpose
- Another mechanism for the data plane to communication with the control plane is packet digest



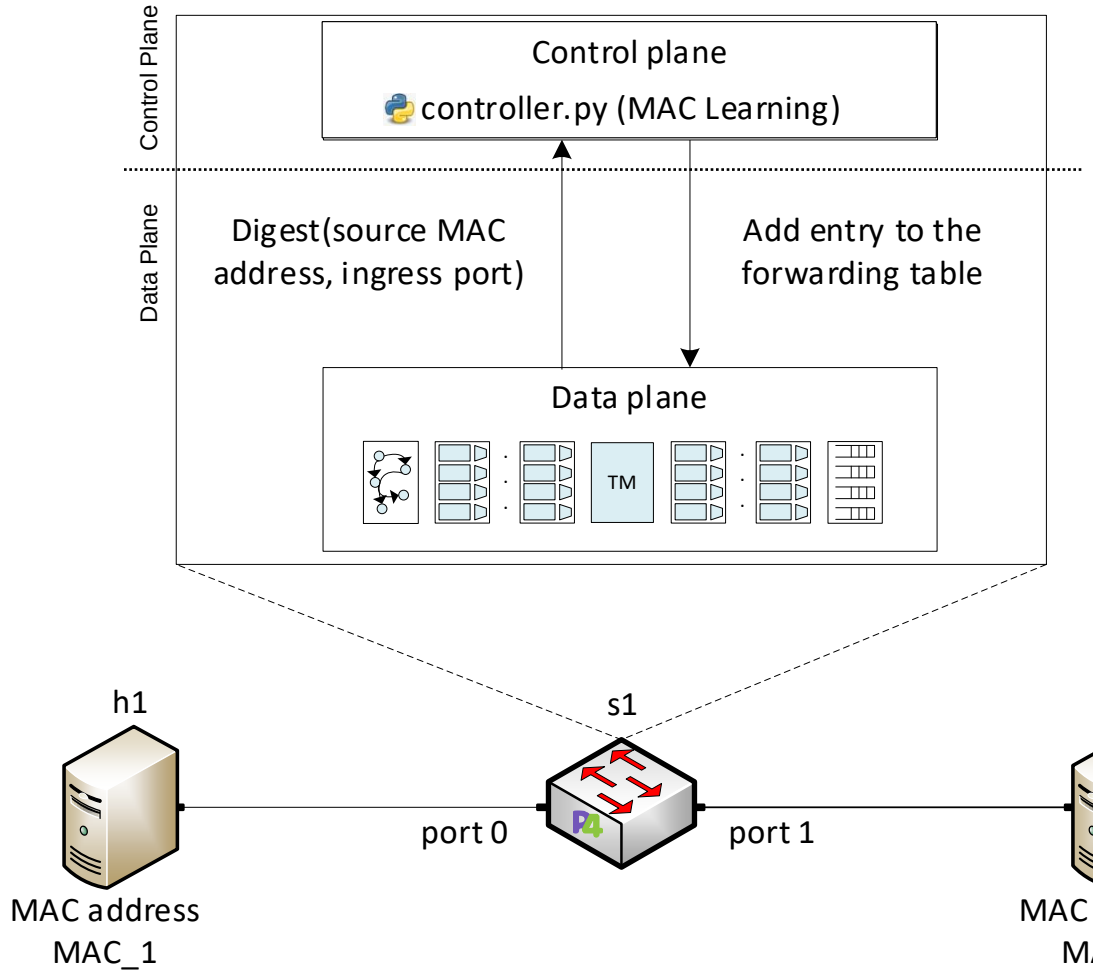
Packet Digests

- The contents of a digest for one packet are typically much smaller than the packet
  - E.g., packet header/s and/or metadata to be processed by a program in the control plane
- The controller computes digests and communicates with the data plane using runtime APIs



Lab Scenario: MAC Learning

- Initially the forwarding table is empty

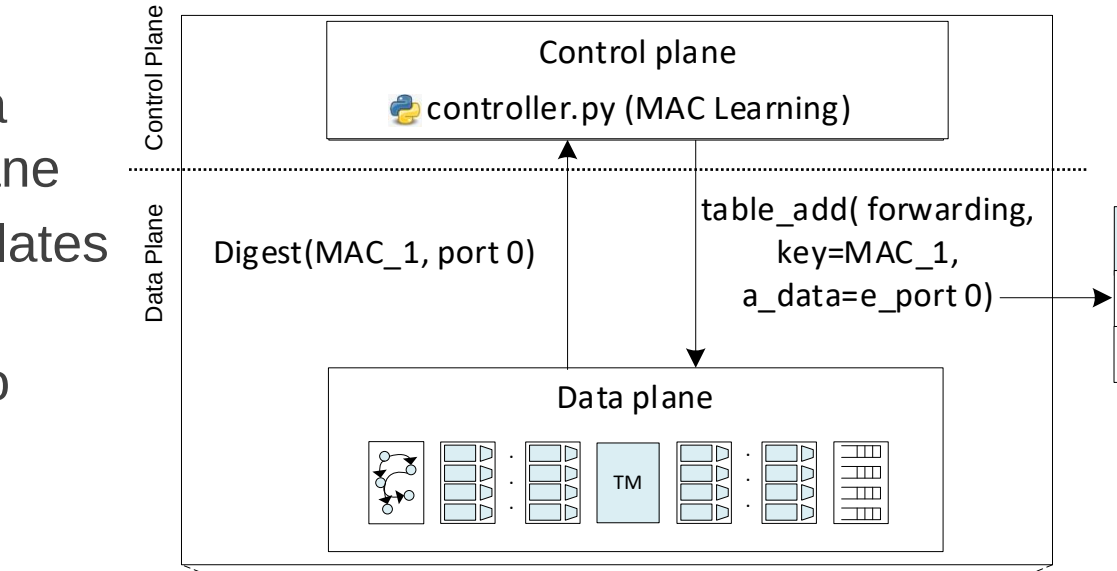


Forwarding table

Key	Action	Action Data

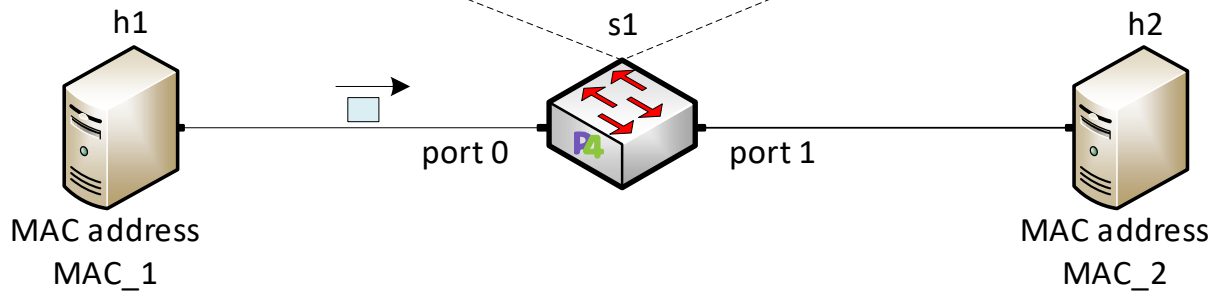
Lab Scenario: MAC Learning

- Switch s1 receives a packet from host h1
- The data plane sends a digest to the control plane
- The control plane populates the forwarding table
- Switch s1 learns how to reach host h1



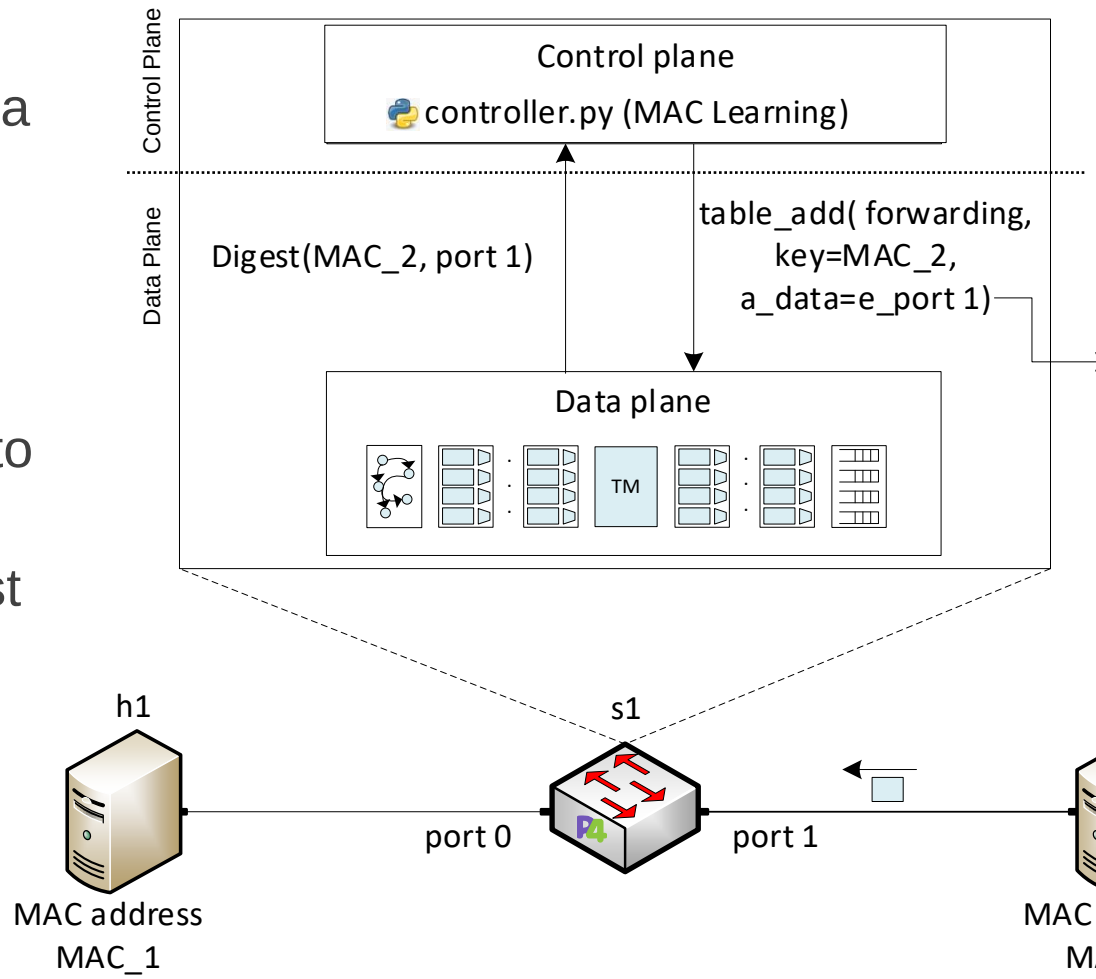
Forwarding table

Key	Action	Action Data
MAC_1	forward	egress port = 0



Lab Scenario: MAC Learning

- Switch s1 receives a packet from host h2
- The data plane sends a digest to the control plane
- The control plane populates the table
- Switch s1 learns how to reach host h2
- Host h1 can reach host h2



Forwarding table

Key	Action	Action Data
MAC_1	forward	egress port = 0
MAC_2	forward	egress port = 1

Lab 11 Topology and Objectives

- The topology consists of two hosts: h1, h2, and one P4 switch: s1
- The objectives are
  - Creating a digest with the source MAC address and ingress port
  - Sending the digest to the control plane
  - Programming a controller with the runtime APIs to create table entries
  - Populating the forwarding table from the control plane
  - Verifying the connectivity between end hosts