

Programmable Switch Hardware

ECE/CS598HPN

Radhika Mittal

Conventional SDN

- Programmable *control plane*.
- Data plane can support high bandwidth.
 - But has limited flexibility.
- Restricted to conventional packet protocols.

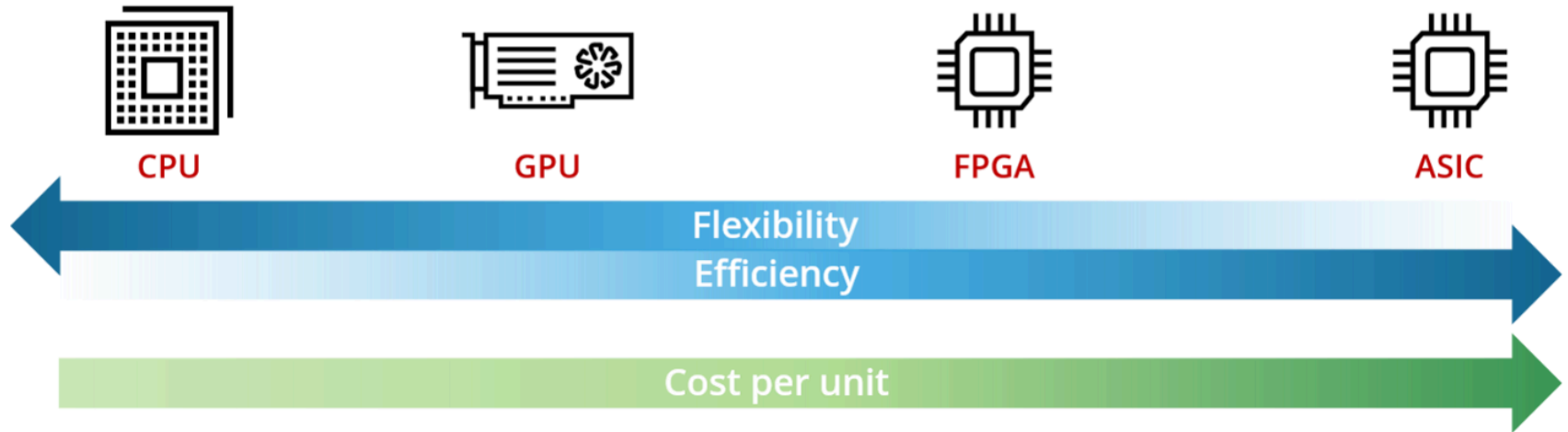
Software Dataplane

- Very extensible and flexible.
- Extensive parallelization to meet performance requirements.
 - Might still be difficult to achieve 100's of Gbps.
- Significant cost and power overhead.

Programmable Hardware

- More flexible than conventional switch hardware.
- Less flexible than software switches.
- Slightly higher power and cost requirements than conventional switch hardware.
- Significantly lower than software switches.

Other alternatives?



Forwarding Metamorphosis: Fast Programmable Match- Action Processing in Hardware for SDN

Pat Bosshart, Glen Gibb, Hun-Seok Kim,
George Varghese, Nick McKeown, Martin Izzard,
Fernando Mujica, Mark Horowitz

Acknowledgements: Slides from Pat Bosshart's SIGCOMM'13 talk

What are the limitations of a fixed function switch?

Need for flexibility....

- Flexibility to:
 - Trade one memory size for another
 - Add a new table
 - Add a new header field
 - Add a different action
- SDN accentuates the need for flexibility
 - Gives programmatic control to control plane, expects to be able to use flexibility
 - OpenFlow designed to exploit flexibility.

What the Authors Set Out To Learn

- How to design a flexible switch chip?
- What does the flexibility cost?

RMT Switch Model

Enables flexibility through....

- Programmable parsing: support arbitrary header fields
- Ability to configure number, topology, width, and depths of match-tables.
- Programmable actions: allow a flexible set of actions (including arbitrary packet modifications).

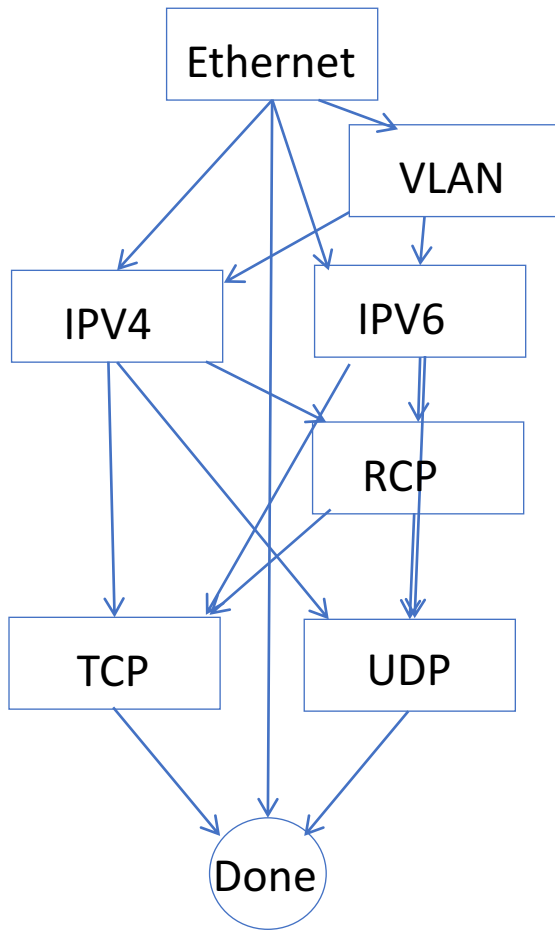
Design Considerations

- Chip size
- High frequency
- Wiring and crossbars
- Amount of memory

The RMT Abstract Model

- Parse graph
- Table graph

Parse Graph and Table Graph



Parse Graph

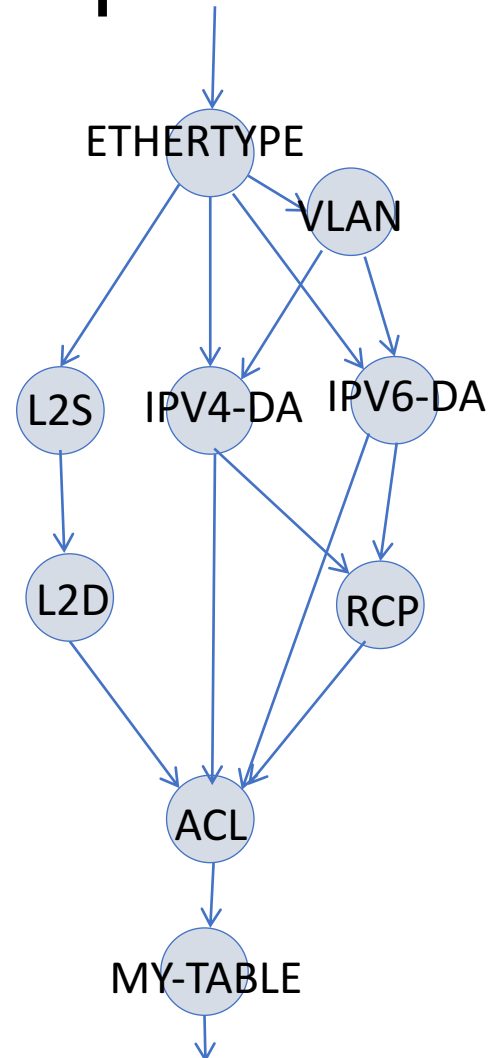


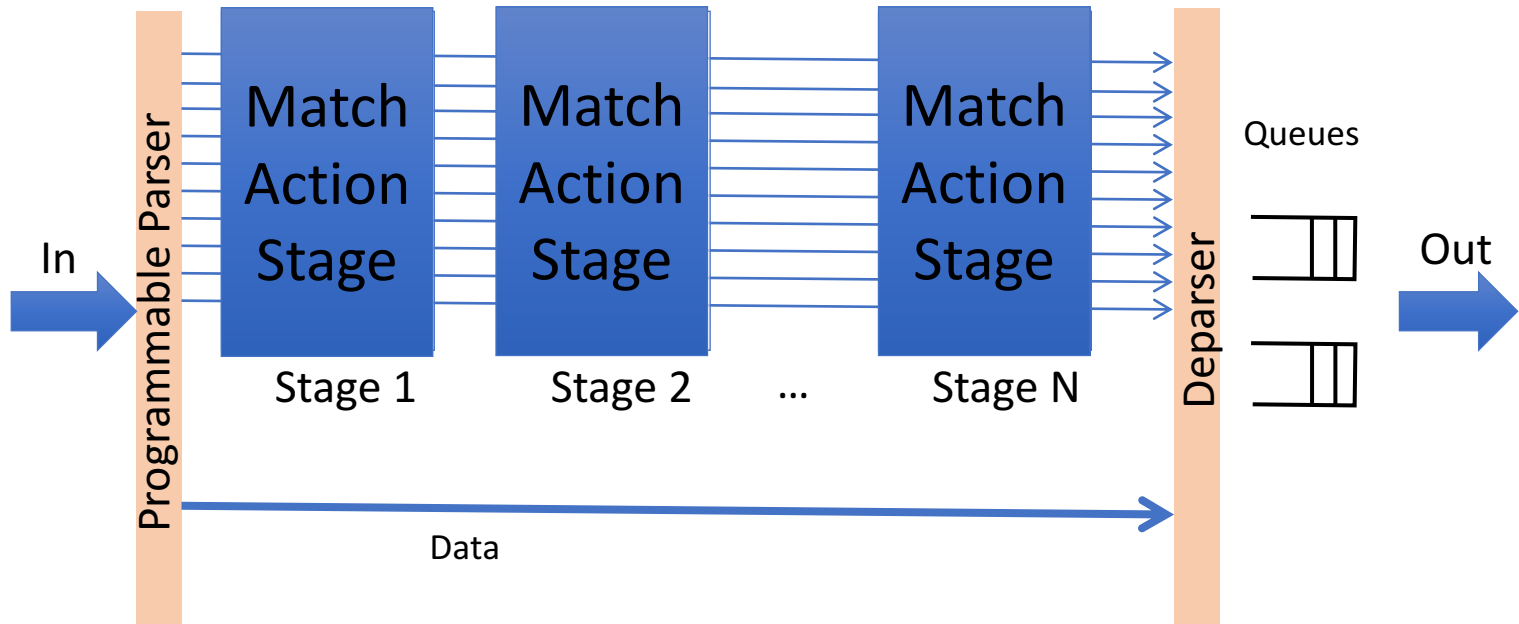
Table Graph

Other alternatives to RMT

- SMT (Single Match-Action)
- MMT (Multiple Match-Action)

What are the limitations?

Match/Action Forwarding Model



Logical to physical mapping

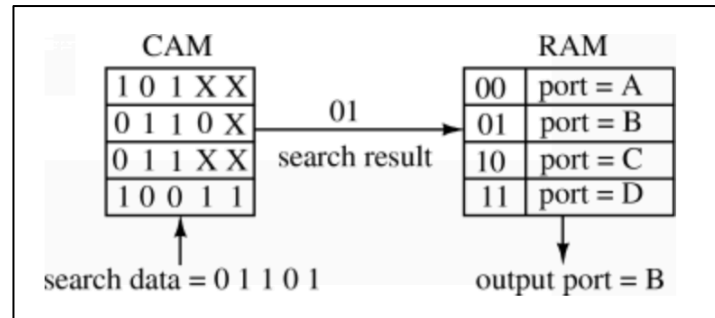
Tiny Detour: CAMs and RAMs

- RAM:
 - Looks up the value associated with a memory address.
- CAM
 - Looks up memory address of a given value.
 - Two types:
 - Binary CAM: Exact match (matches on 0 or 1)
 - Can be implemented using SRAM.
 - Ternary CAM (TCAM): Allows wildcard (matches on 0, 1, or X).

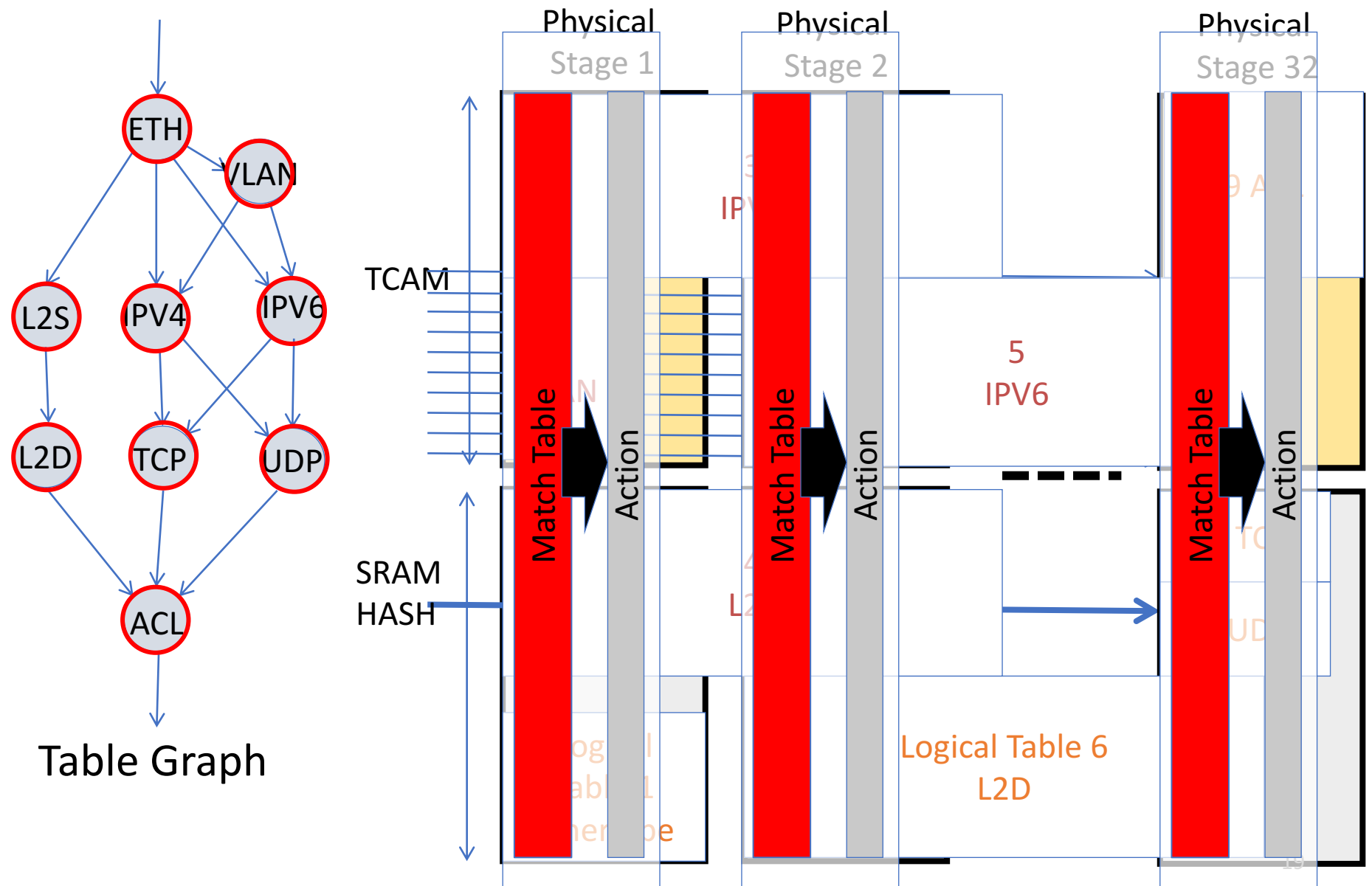
Tiny Detour: CAMs

Line No.	Address (Binary)	Output Port
----------	------------------	-------------

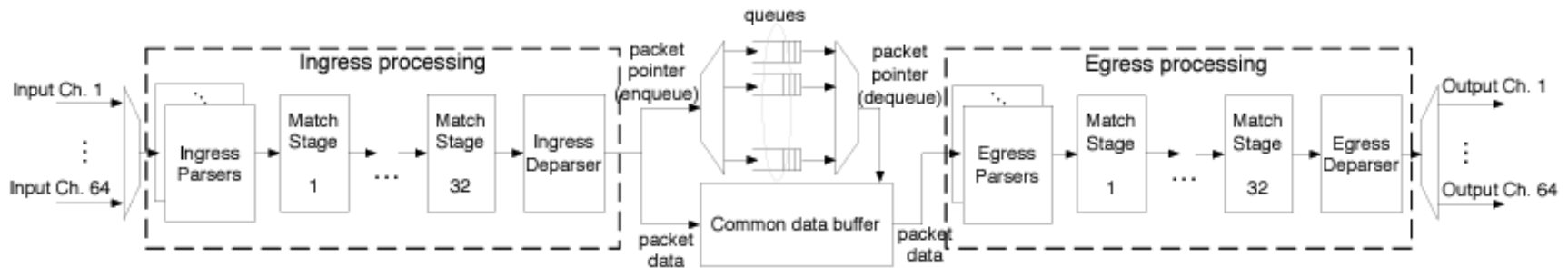
1	101XX	A
2	0110X	B
3	011XX	C
4	10011	D



Logical to physical mapping

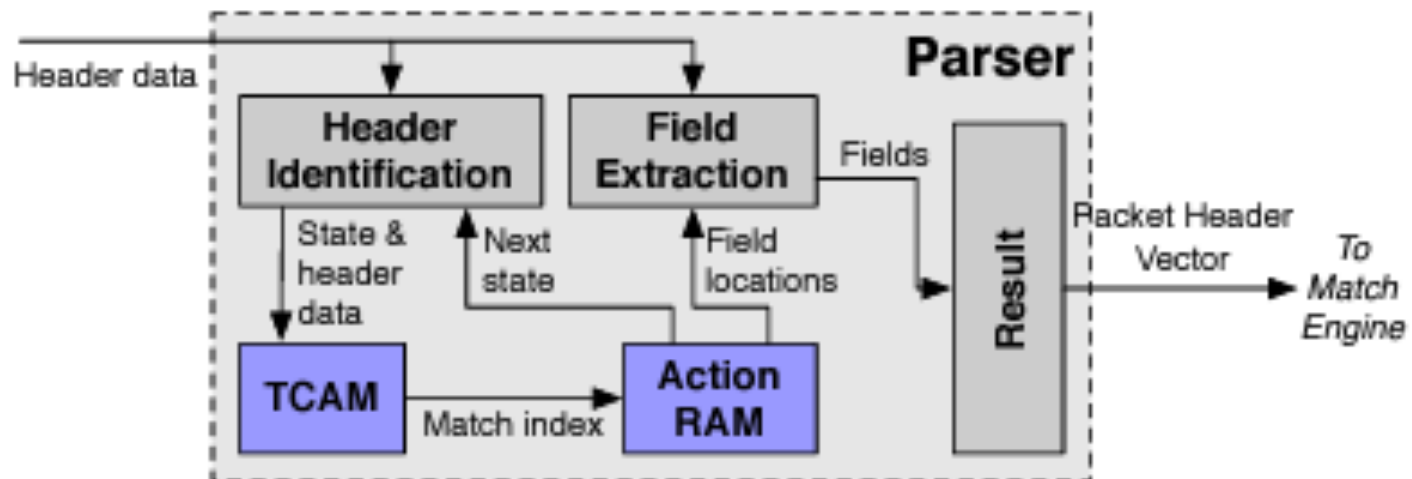


Overall architecture

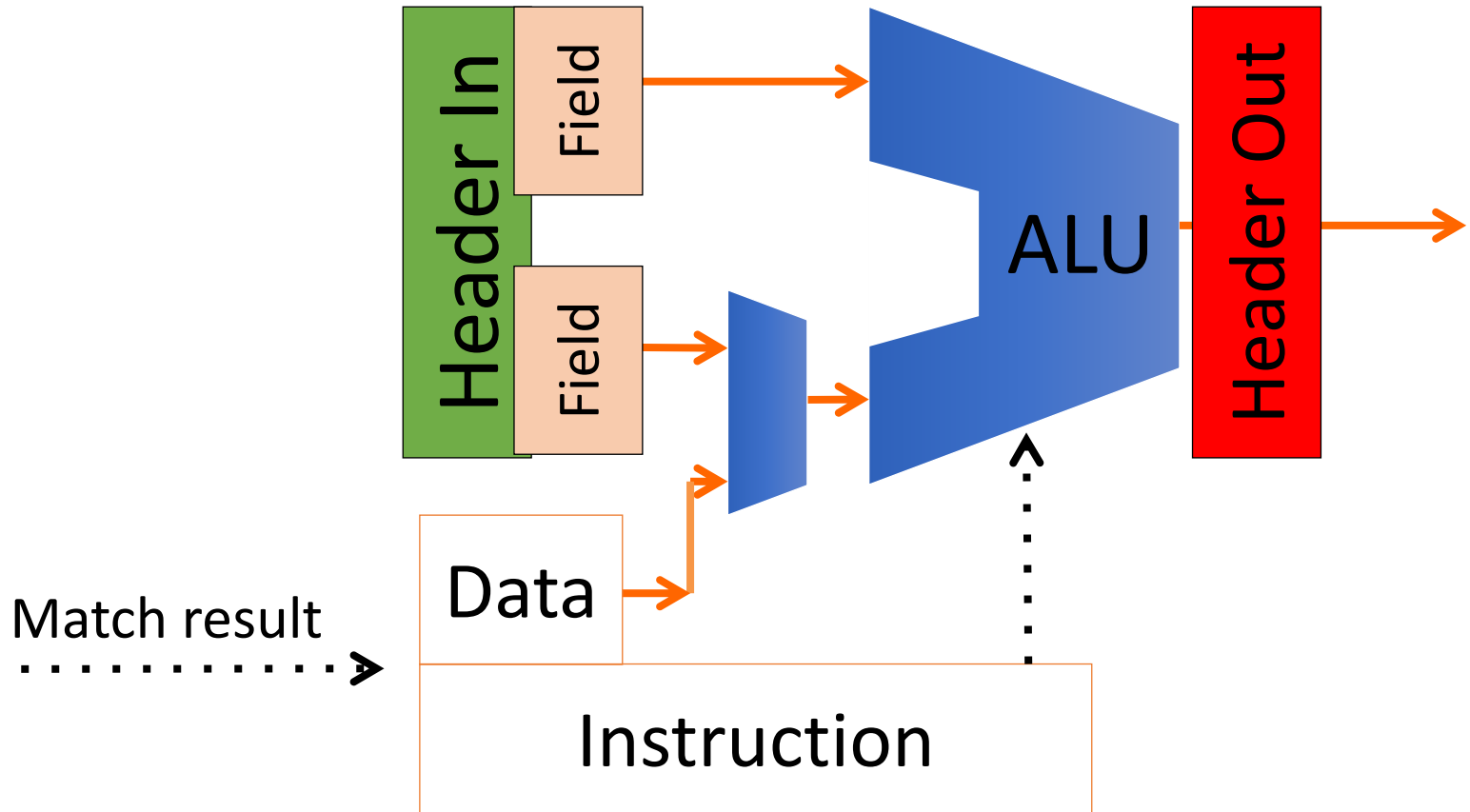


Referred as PISA: Protocol Independent Switch Architecture

Configurable Parser



Action Processing Model



RMT Switch Design

- 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

Flexibility costs less than 15%

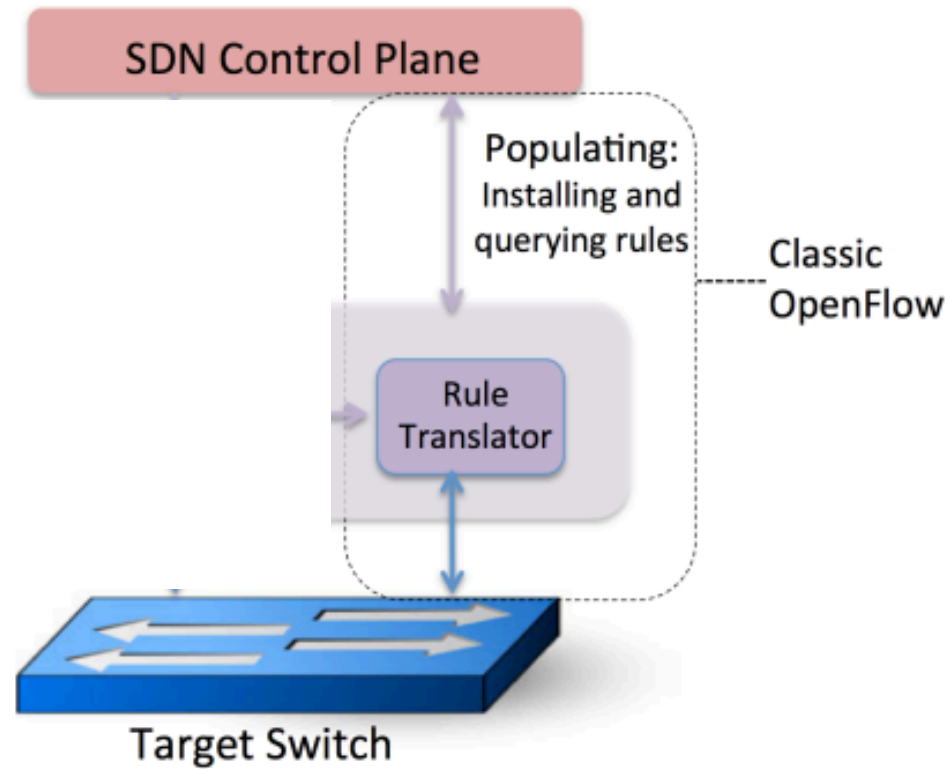
An interface to program such switches



P4 Goals

- Protocol independence
 - Switches are not tied to specific packet formats.
- Reconfigurability
 - Controller can redefine packet parsing and processing in the field.
- Target Independence
 - User program need not be tied to a specific hardware.
 - Compiler's job to do the mapping.

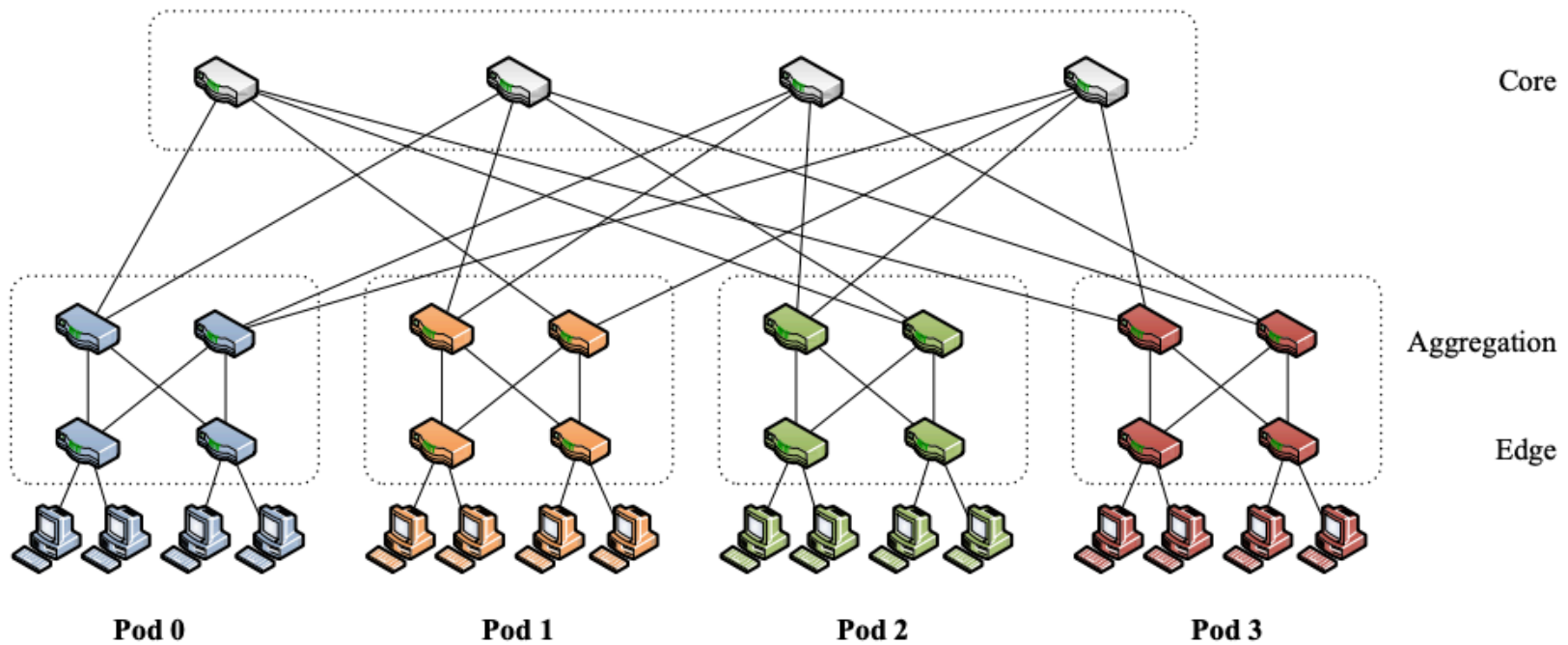
P4 vs OpenFlow



Components of a P4 program

- Header definitions
- Parser definition
- Tables: what fields to match on, and which action to execute
- Action definition.

Example



Example

```
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;  
    }  
}
```

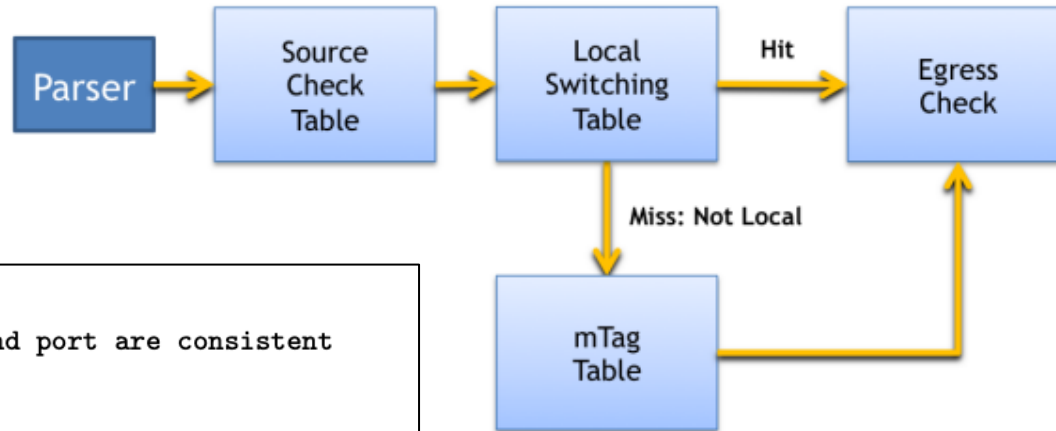
```
header mTag {  
    fields {  
        up1 : 8;  
        up2 : 8;  
        down1 : 8;  
        down2 : 8;  
        ethertype : 16;  
    }  
}
```

Example

```
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  
    }  
}
```

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);  
    }  
}
```

Example

```
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;
}
```

Example

```
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);
}
```

P4 Compiler

- If the target is a fixed-function switch?
 - Check if specified parser and match-action tables are supported.
 - If not, return error.
- If target is a software switch?
 - Full flexibility to execute specified program.
 - May use specific software data structures for optimizations.
- If target is an RMT switch?
 - Figure out table layout
 - mapping logical stages to physical ones.
 - When to use RAM vs TCAM
 - If tables don't fit, an action not supported, etc: return an error.

P4 Tutorial

<https://github.com/p4lang/tutorials>

[Additional instructions](#) for those using Apple Silicon M1.

How is this paradigm different from active networking?

What are some new opportunities enabled by P4/RMT?

What are the limitations of P4/RMT?

Since 2013....

- 2013: RMT switch was commercialized (2013)
 - **Barefoot Tofino:** 6.5Tb/s
 - Other switches supporting PISA architecture: Cavium Xpliant, Intel FlexPipe.
- 2017: Adoption of Tofino switches at Alibaba, Baidu, Tencent, AT&T
- 2019: Barefoot acquired by Intel
- *Jan 2023: Intel stopped future investment in it;*
 - *(will continue to support existing products and customers).*

Lots of follow-on work!

Event-driven packet processing

- Restricted to packet ingress and egress events.
- May want to react to other types of events:
 - Periodic (generate probes, reset counters), link failures.
- Updated switch architecture design that allows reacting on such events.
- Ibanez et. al., HotNets'19.

External memory for switches

- Switches require high memory bandwidth.
 - Use fast, but expensive on-chip SRAM and TCAM.
 - Limited in size.
- Memory size could be a limiting factor for many applications.
- Let's access endhost memory remotely....
- Kim et. al., HotNets'18, SIGCOMM'20

On P4 compilers and programs

- Some “target-awareness” is still needed when writing P4 program (e.g. array sizes etc). How to enable true target-independence?
 - Use elastic data structures
 - P4All, Hogan et. al., HotNets’20, NSDI’22
- Compiler must somehow map a given P4 program to target hardware. How to optimize this?
 - Profile-guided optimizations, Wintermeyer et. al., HotNets’20
- Mapping a P4 program to multiple subprograms on different switches and running them as a distributed system.
 - FlightPlan, Sultana et. al., NSDI’21

Enabling runtime programmability

- In-situ Programmable Switching
 - Feng et. al, HotNets'21, NSDI'22
- Runtime Programmable Switches
 - Xing et. al., HotNets'21, NSDI'22
- ActiveRMT
 - Das et. al., HotNets'20, SIGCOMM'23

Extending/testing capabilities

- Statistics in P4
 - Gao et. al., HotNets'21
- Floating point operations in P4
 - Yuan et. al., NSDI'22

On state management

- Vision for enabling more stateful packet processing
 - Gebara et. al., HotNets'20
- Distributed state management for a program running on multiple switches
 - Zeno et. al., HotNets'20, NSDI'22
- Fault-tolerance for switch state
 - RedPlane, Kim et. al., SIGCOMM'22

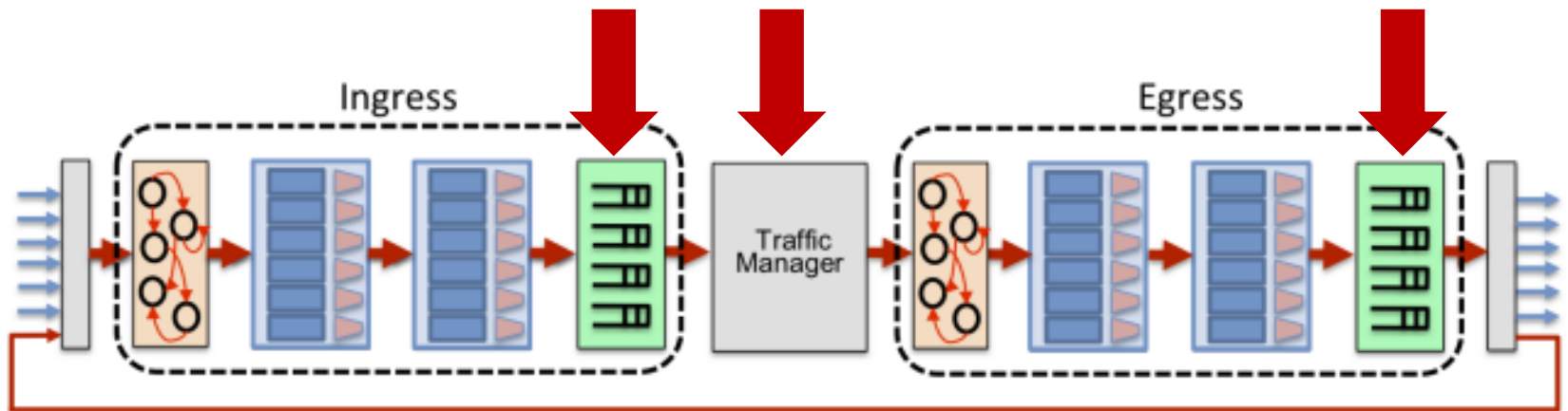
Alternate architectures

- dRMT: Disaggregated memory and compute resources.
 - Chole et al, SIGCOMM'17
- Trio: Juniper's programmable chipset
 - Yang et al, SIGCOMM'22
 - Deviates from pipeline architectures, uses customized processor cores with massive parallelism, and a run-to-completion model.

Usecases of P4/RMT

- Lots and lots of work on this over subsequent years.
- Next class!

Flexible queuing and scheduling



- Universal packet scheduling (Mittal et al, NSDI'16)
- Programmable packet scheduling (Sivaraman et al, SIGCOMM'16)
- Lots of follow-on work to emulate generic scheduling behavior using discrete number of priority queues.