



Programmable Data Plane at Terabit Speeds

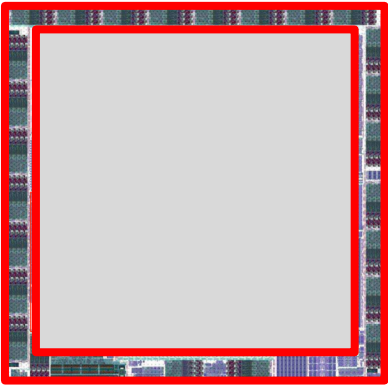
Vladimir Gurevich
May 16, 2017

“Programmable switches are 10-100x slower than fixed-function switches. They cost more and consume more power.”

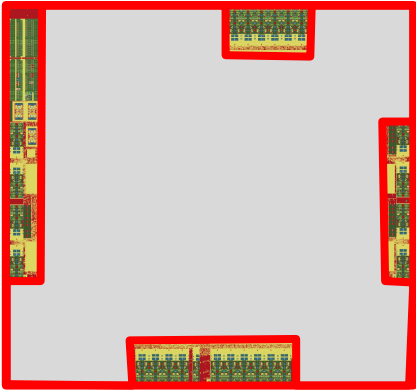
Conventional wisdom in networking

Moore's Law and Simple Math

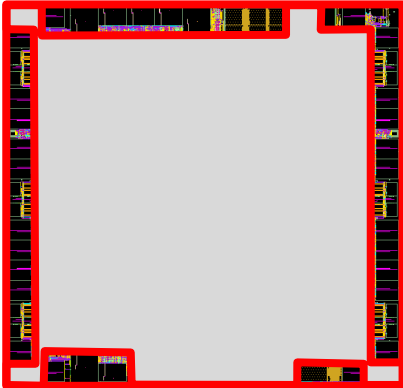
Serial I/O: About 30% of switch chip area



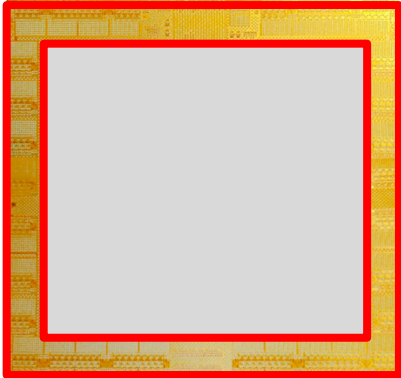
Intel Alta (2011)



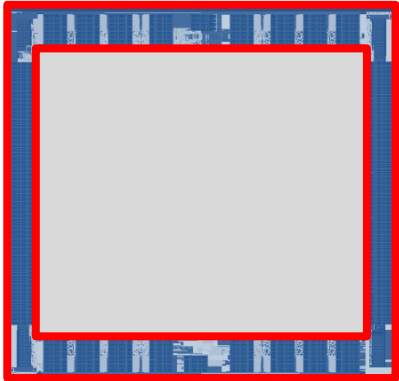
Cisco (2011)



Ericsson Spider (2011)



Broadcom Tomahawk (2014)



Barefoot Tofino (2016)

30% Serial I/O

50% Memory
Lookup Tables
Packet Buffer

20% Logic
Packet
Processing

30% Serial I/O

50% Memory
Lookup Tables
Packet Buffer

20% Logic
Packet
Processing

30% Serial I/O

50% Memory
Lookup Tables
Packet Buffer

20% Logic
Packet
Processing

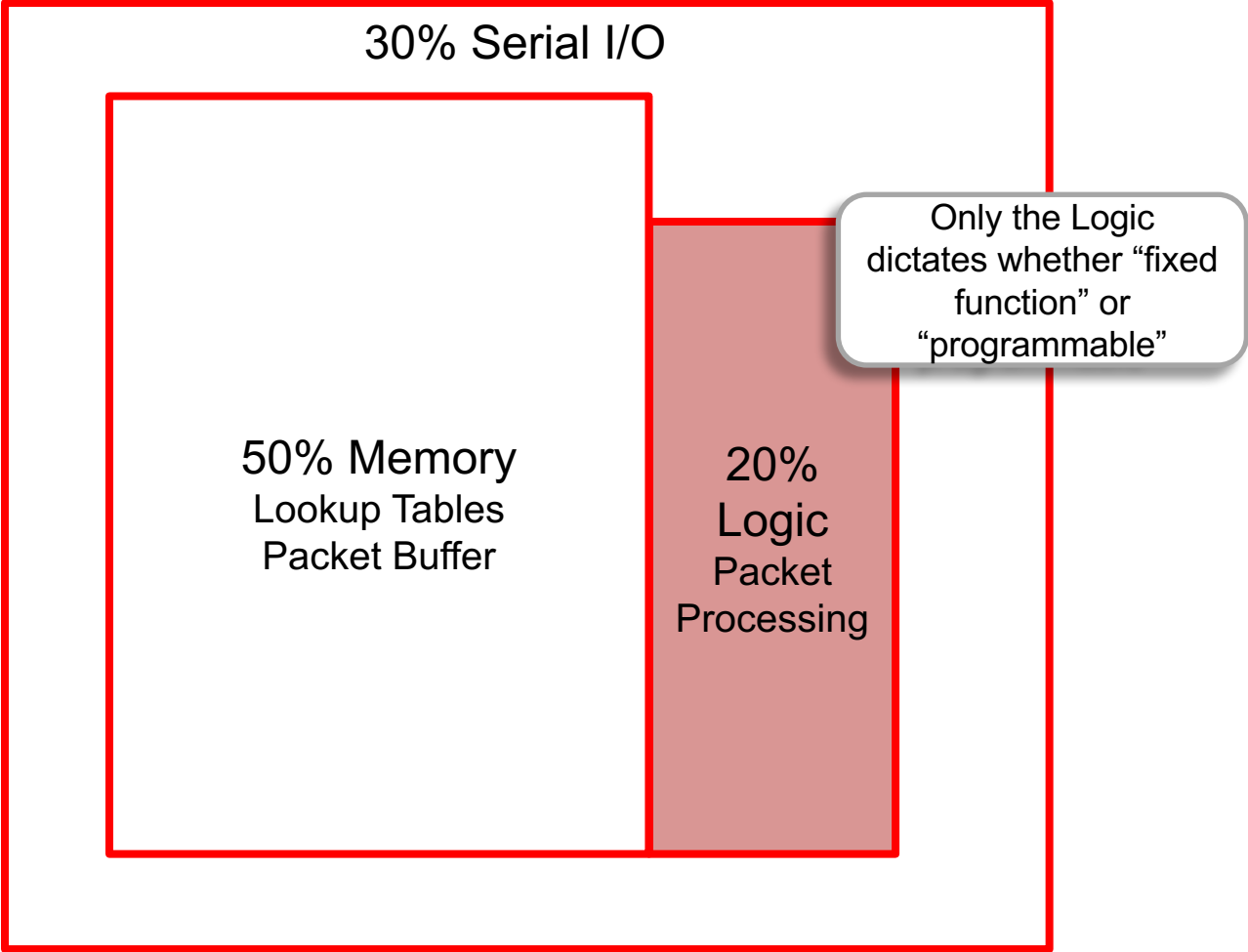
Only the Logic
dictates whether “fixed
function” or
“programmable”

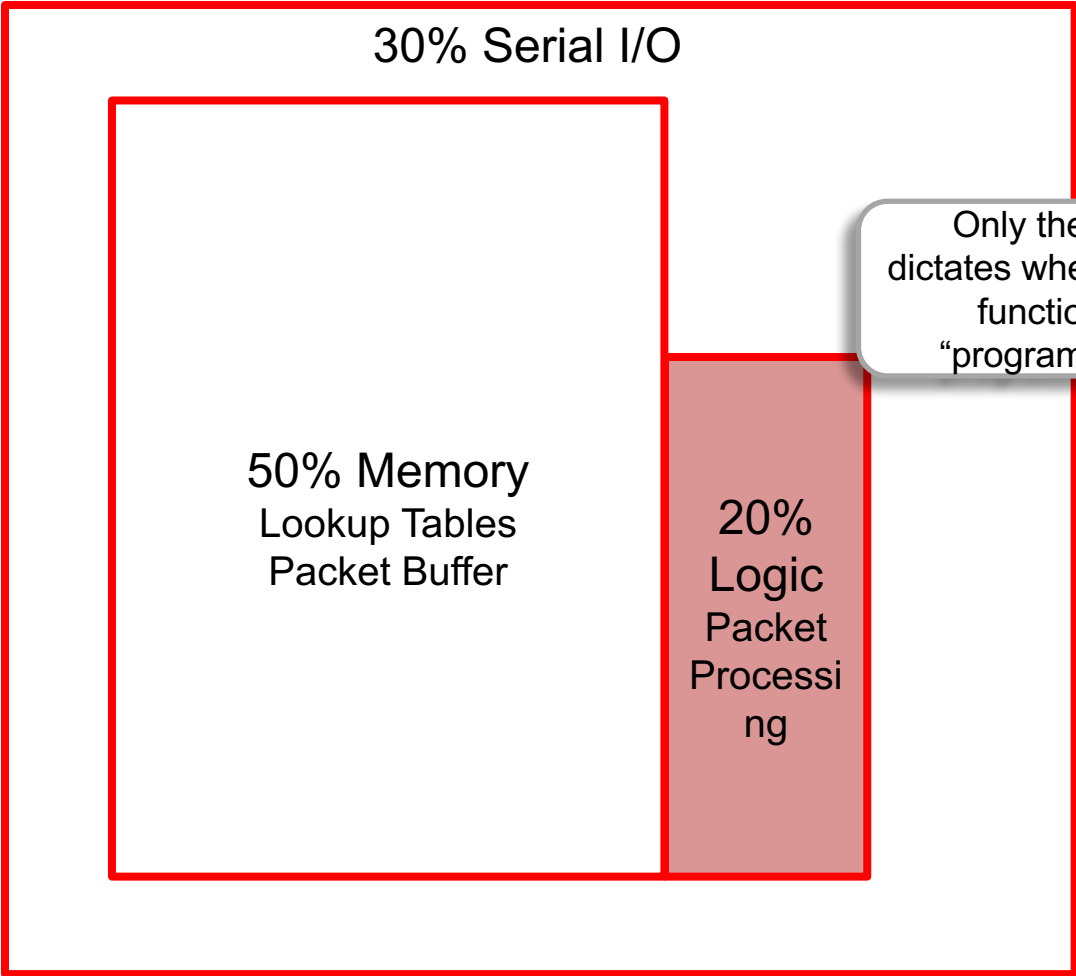
30% Serial I/O

50% Memory
Lookup Tables
Packet Buffer

20% Logic
Packet
Processing

Only the Logic
dictates whether “fixed
function” or
“programmable”

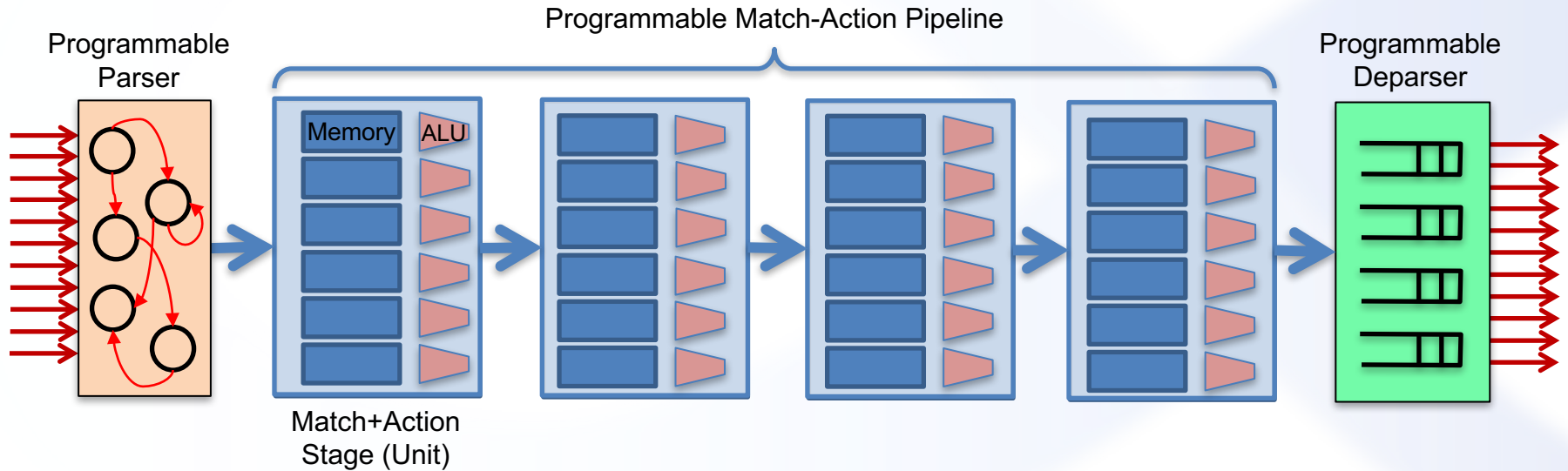




Parallelism and alternatives

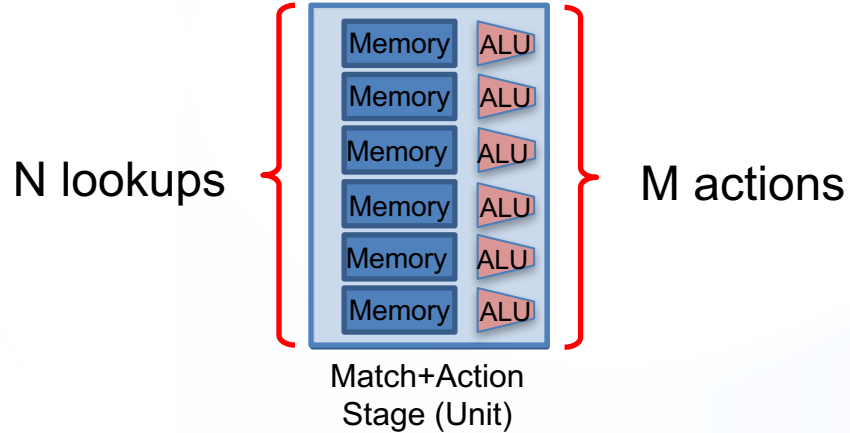
- **Sequential semantics does not prohibit parallelism**
- **Doing everything does not mean doing everything all the time**

PISA: Important Details



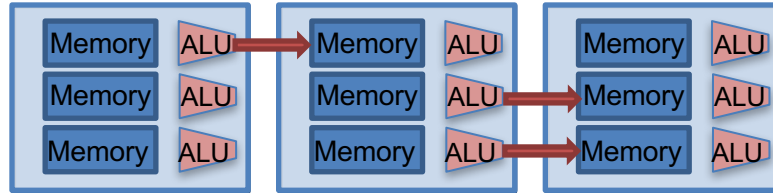
PISA: Important Details

- Multiple simultaneous lookups and actions can be supported



PISA: Match and Action are Separate Phases

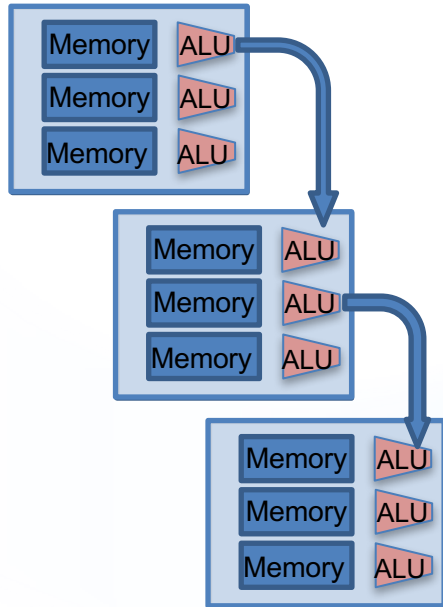
- Sequential Execution (Match dependency)



Total Latency = 3

PISA: Match and Action are Separate Phases

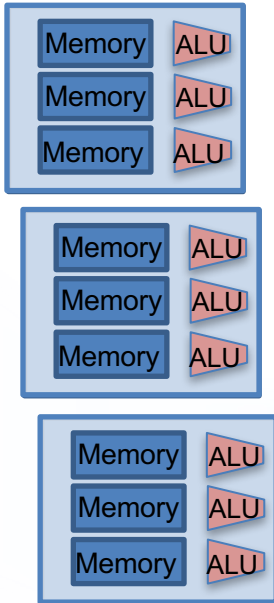
- Sequential Execution (Match Dependency)
- Staggered Execution (Action Dependency)



Total Latency = 2

PISA: Match and Action are Separate Phases

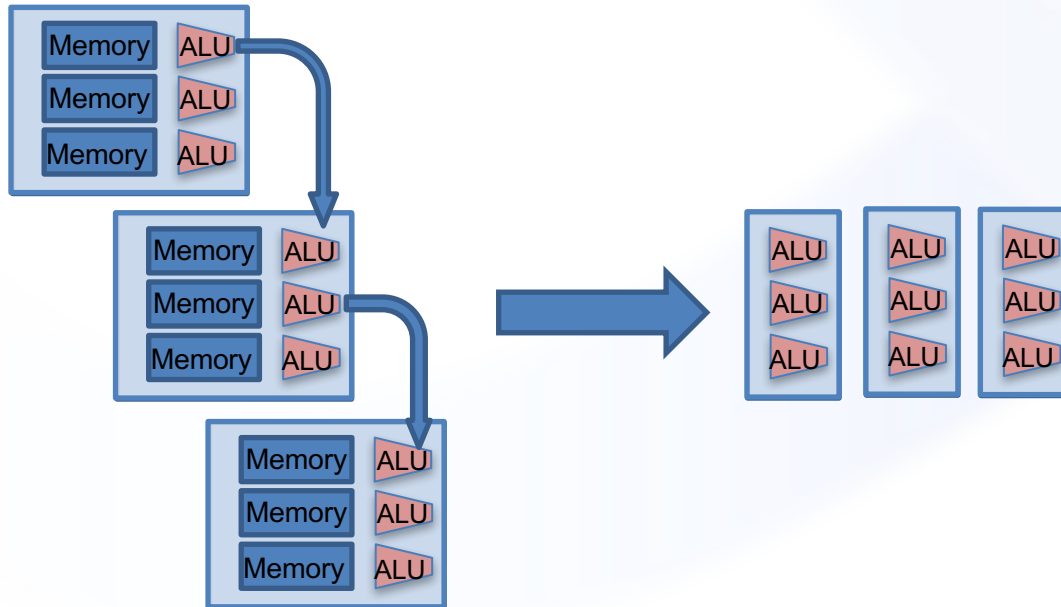
- Sequential Execution (Match Dependency)
- Staggered Execution (Action Dependency)
- Parallel Execution (No Dependencies)



Total Latency = 1.1

PISA: Match is not required

- A sequence of actions is an imperative program



Symmetric Switch Model

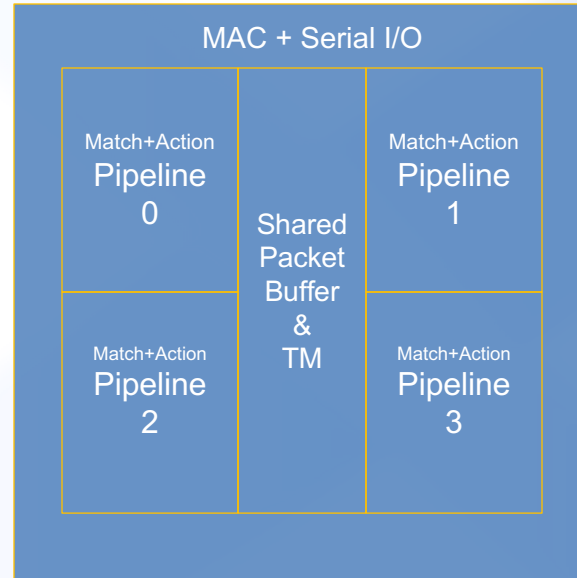
- Why not share the hardware between ingress and egress?



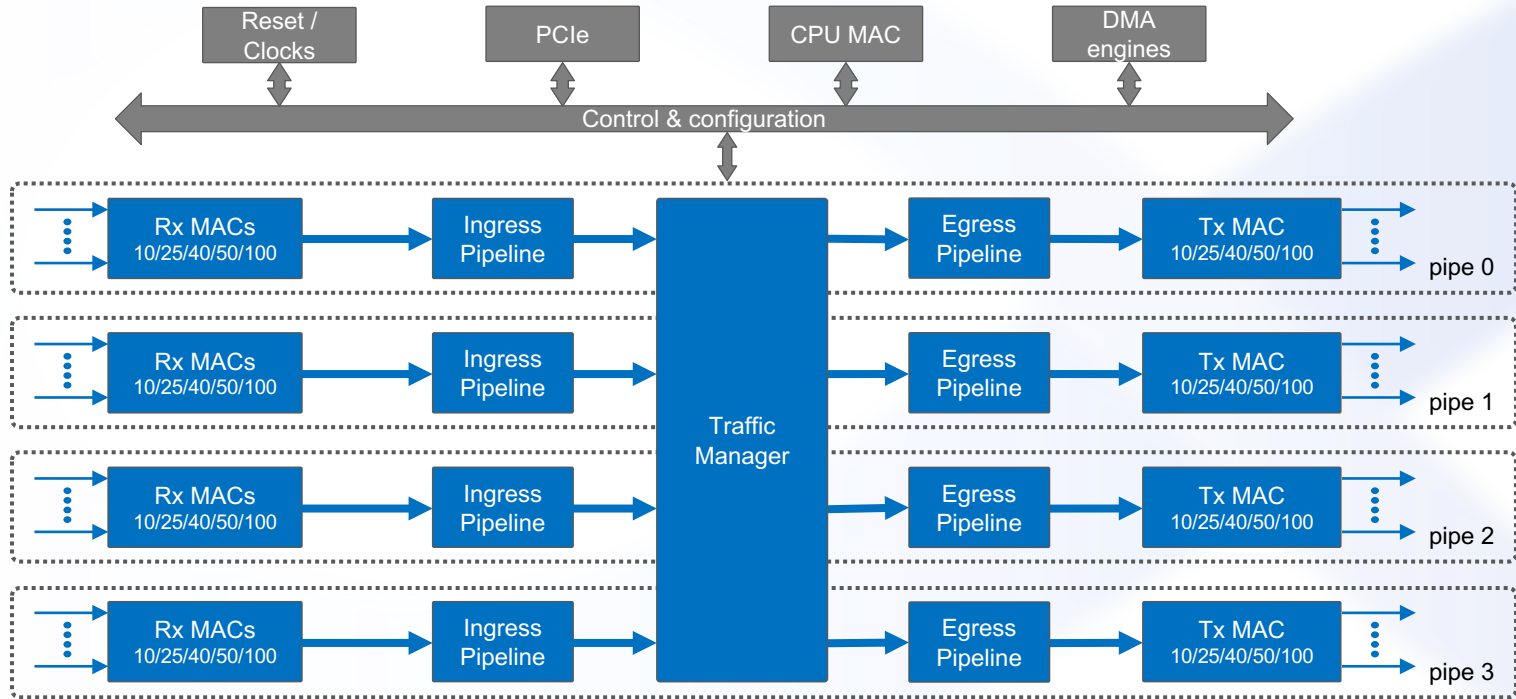
Introducing Tofino

6.5Tb/s Tofino™ Summary

- **State of the art design**
 - Single Shared Packet Buffer
 - TSMC 16nm FinFET+
- **Four Match+Action Pipelines**
 - Fully programmable PISA Embodiment
 - All compiled programs run at line-rate.
 - Up to 1.3 million IPv4 routes
- **Port Configurations**
 - 65 x 100GE/40GE
 - 130 x 50GE
 - 260 x 25GE/10GE
- **CPU Interfaces**
 - PCIe: Gen3 x4/x2/x1
 - Dedicated 100GE port



Tofino. Simplified Block Diagram



Each pipe has 16x100G MACs + a Packet
Additional ports for recirculation, Packet Generator, CPU

P4₁₆ Language Elements

Parsers

State machine,
bitfield extraction

Controls

Tables, Actions,
control flow
statements

Expressions

Basic operations
and operators

Data Types

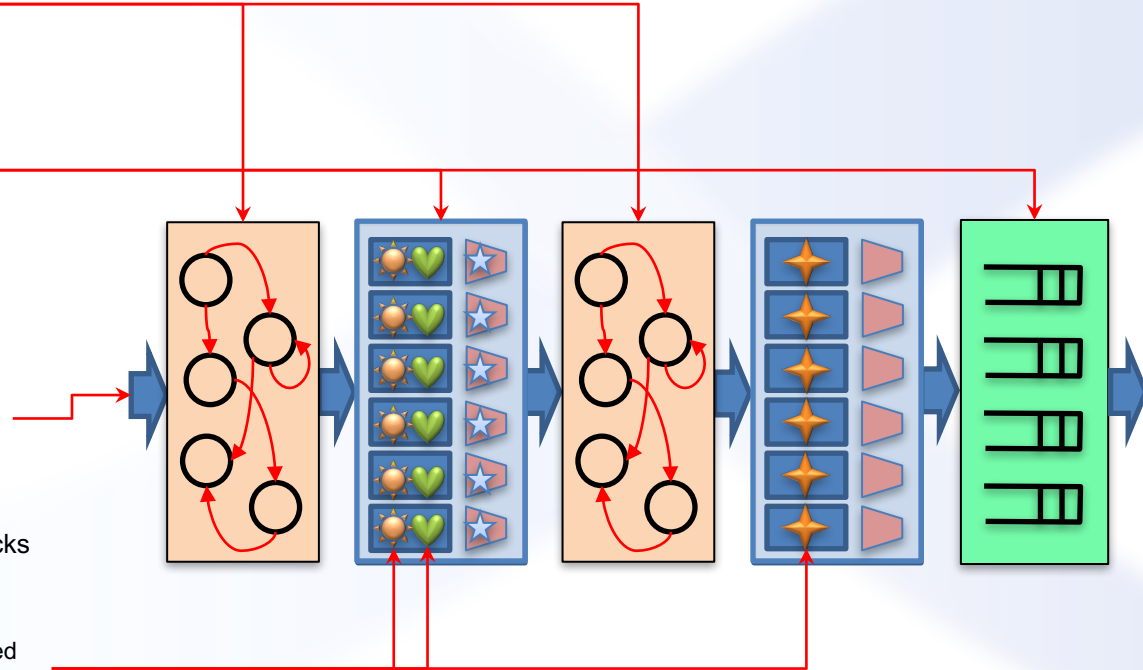
Bistrings, headers,
structures, arrays

Architecture
Description

Programmable blocks
and their interfaces

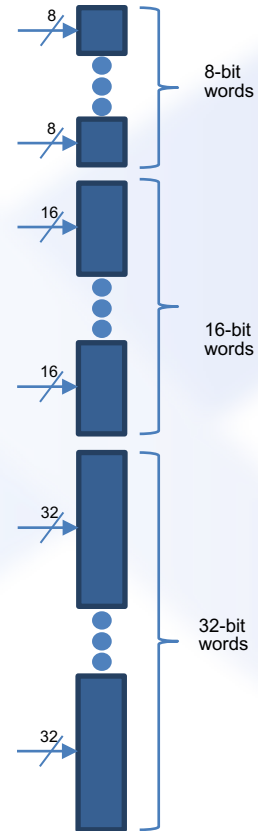
Extern Libraries

Support for specialized
components



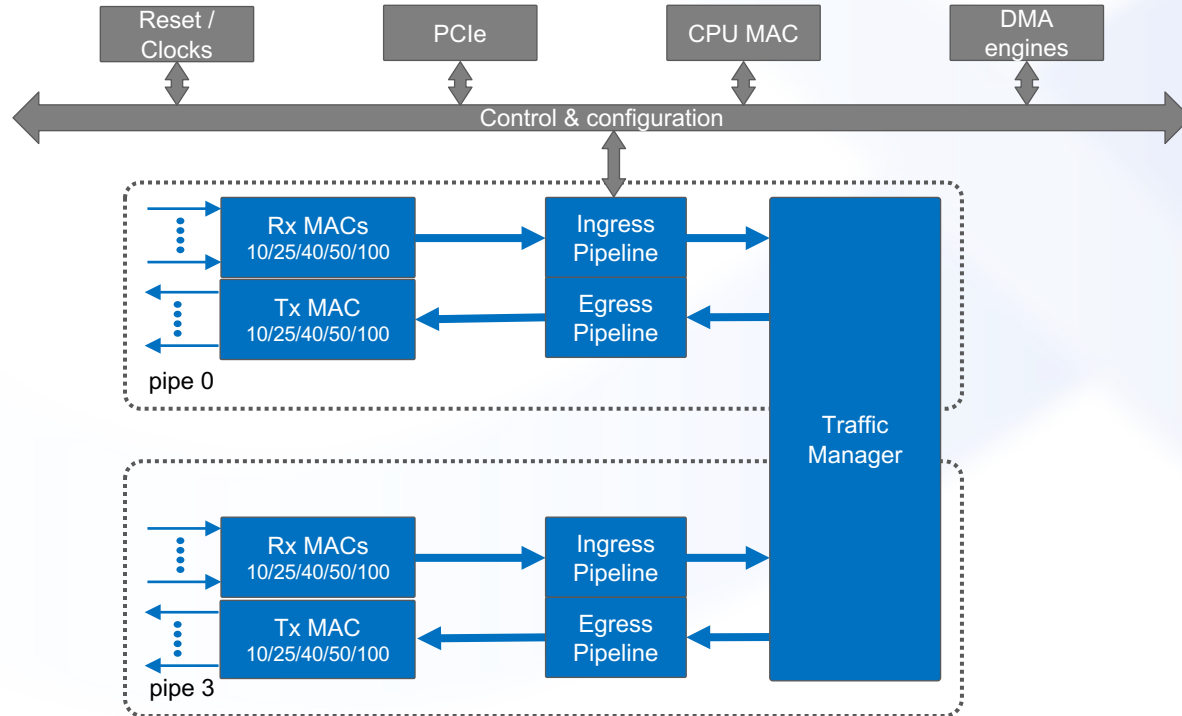
Packet Header Vector (PHV)

- A set of uniform containers that carry the headers and metadata along the pipeline
- Fields can be packed into any container or their combination
- PHV Allocation step in the compiler decides the actual packing

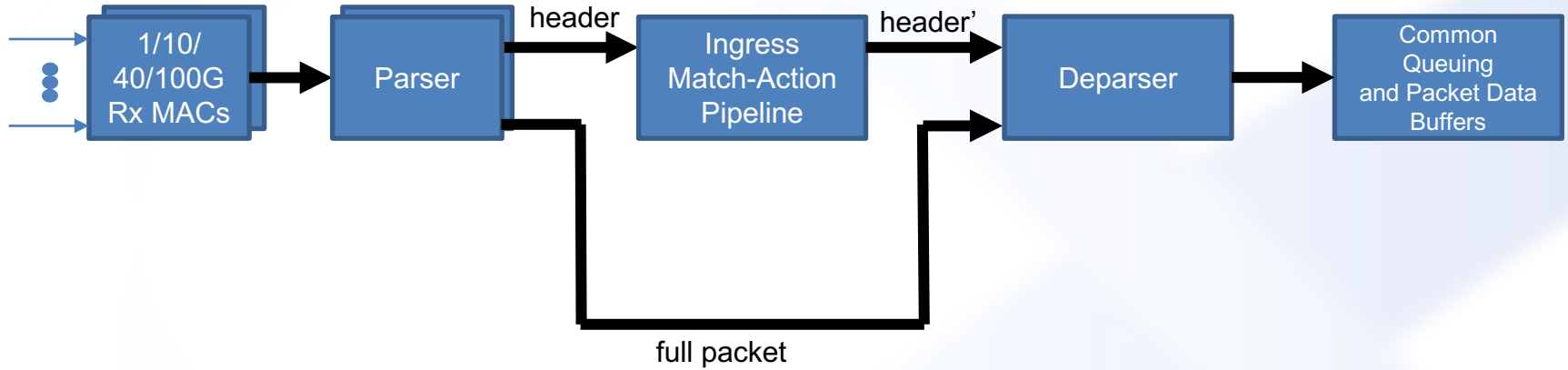


Unified Pipeline

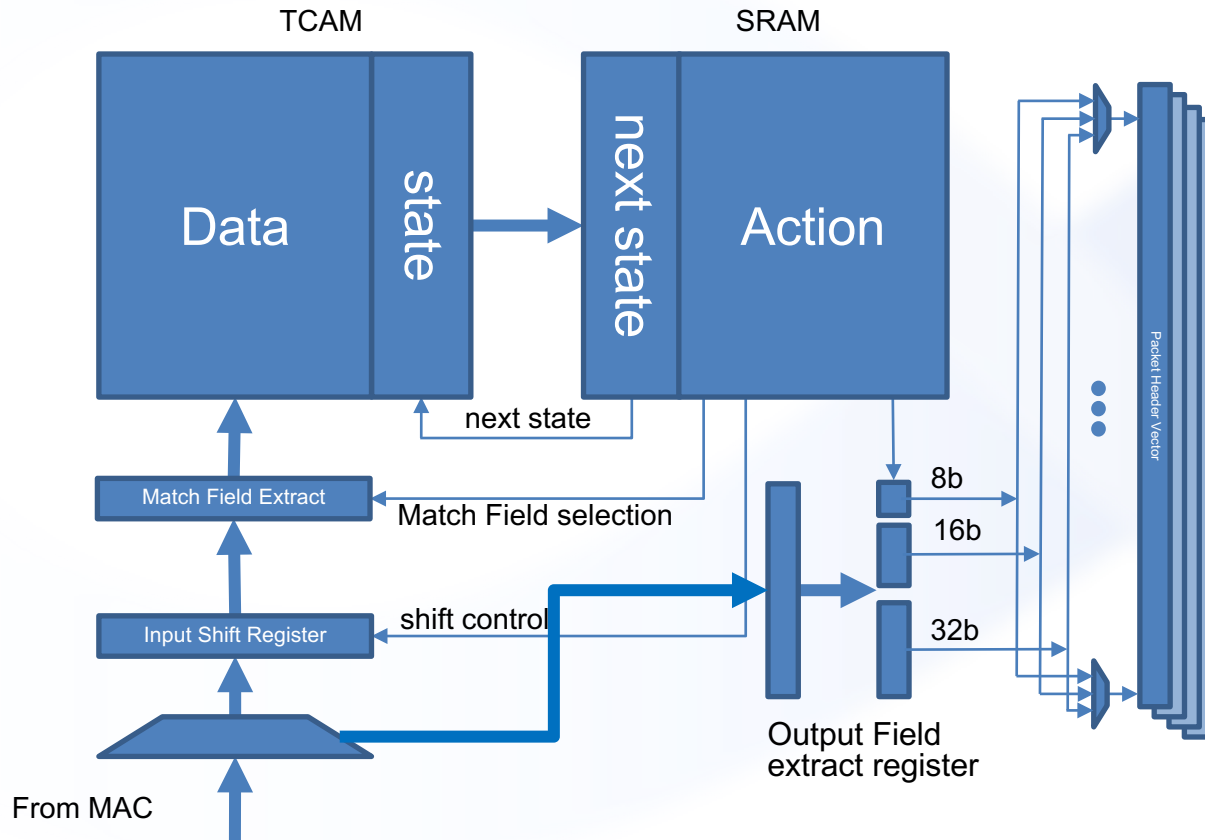
- There is no difference between ingress and egress processing
 - The same blocks can be efficiently shared



The Basic Structure



Parser



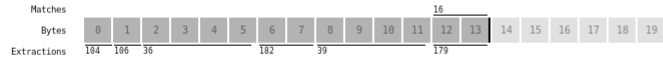
Parser Visualization

Row 147

State start_parse_ethernet (from state <POV initialization>_<Ingress intrinsic metadata>_<Phase 0>)

[.] Raw register data

[.] Input buffer



PHV 58 |= 0x400000

PHV 58 |= 0x1

[.] Transitions

```
16b
0000 && fe00 -> Row 146 (state parse_llc_header)
0000 && fa00 -> Row 145 (state parse_llc_header)
9000 -> Row 144 (state parse_fabric_header)
8100 -> Row 143 (state parse_vlan)
9100 -> Row 142 (state parse_qinq)
8847 -> Row 141 (state parse_mpls_it0)
0800 -> Row 140 (state parse_ipv4)
86dd -> Row 139 (state parse_ipv6)
0806 -> Row 138 (state parse_arp_rarp)
88cc -> Row 137 (state parse_set_prio_high)
8809 -> Row 136 (state parse_set_prio_high)
Default -> Row 135 (state <leaf>)
```

Previous states: [Row 255](#)

Row 140

State parse_ipv4 (from state start_parse_ethernet)

[.] Raw register data

[.] Input buffer



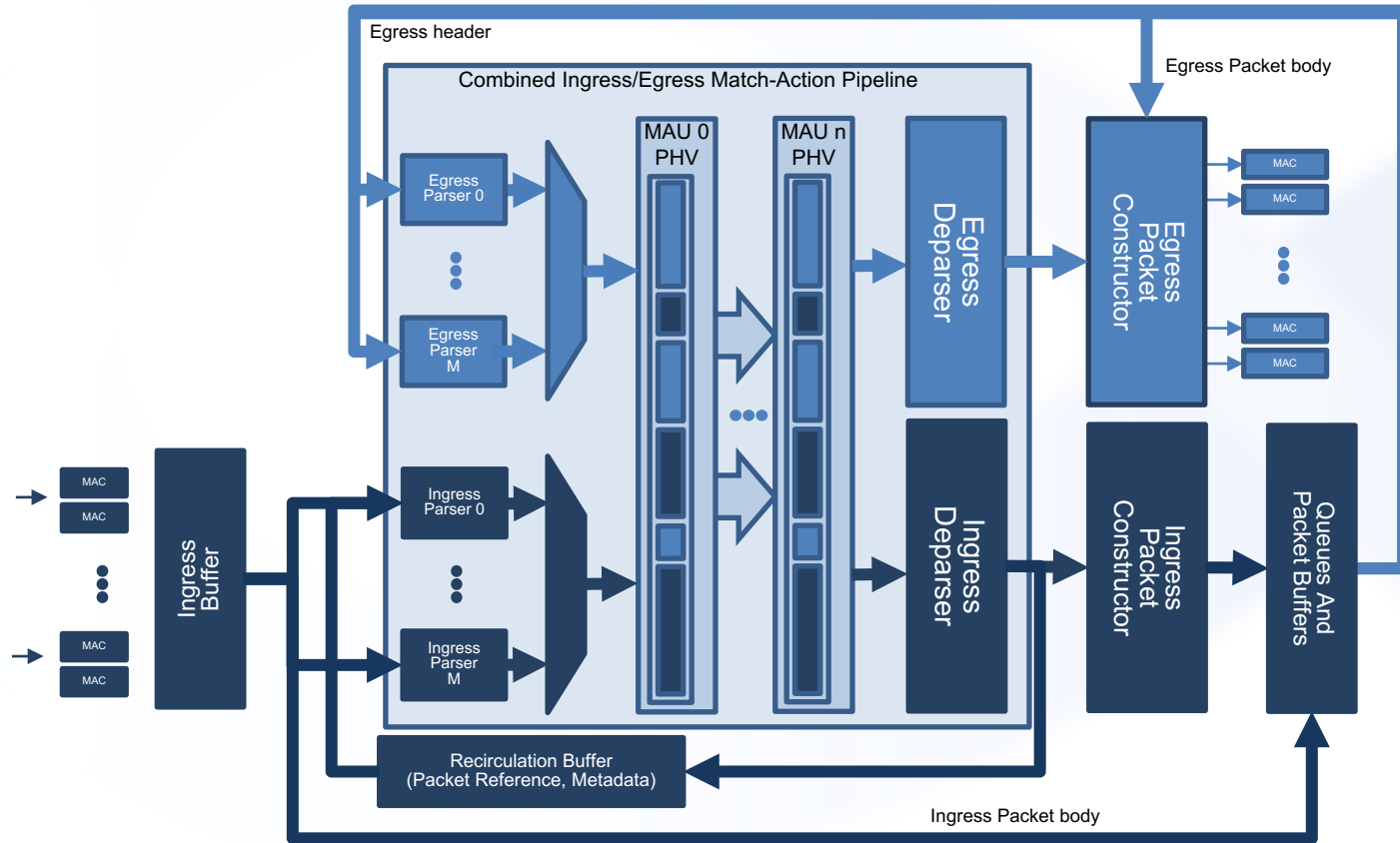
PHV 58 |= 0x8

[.] Transitions

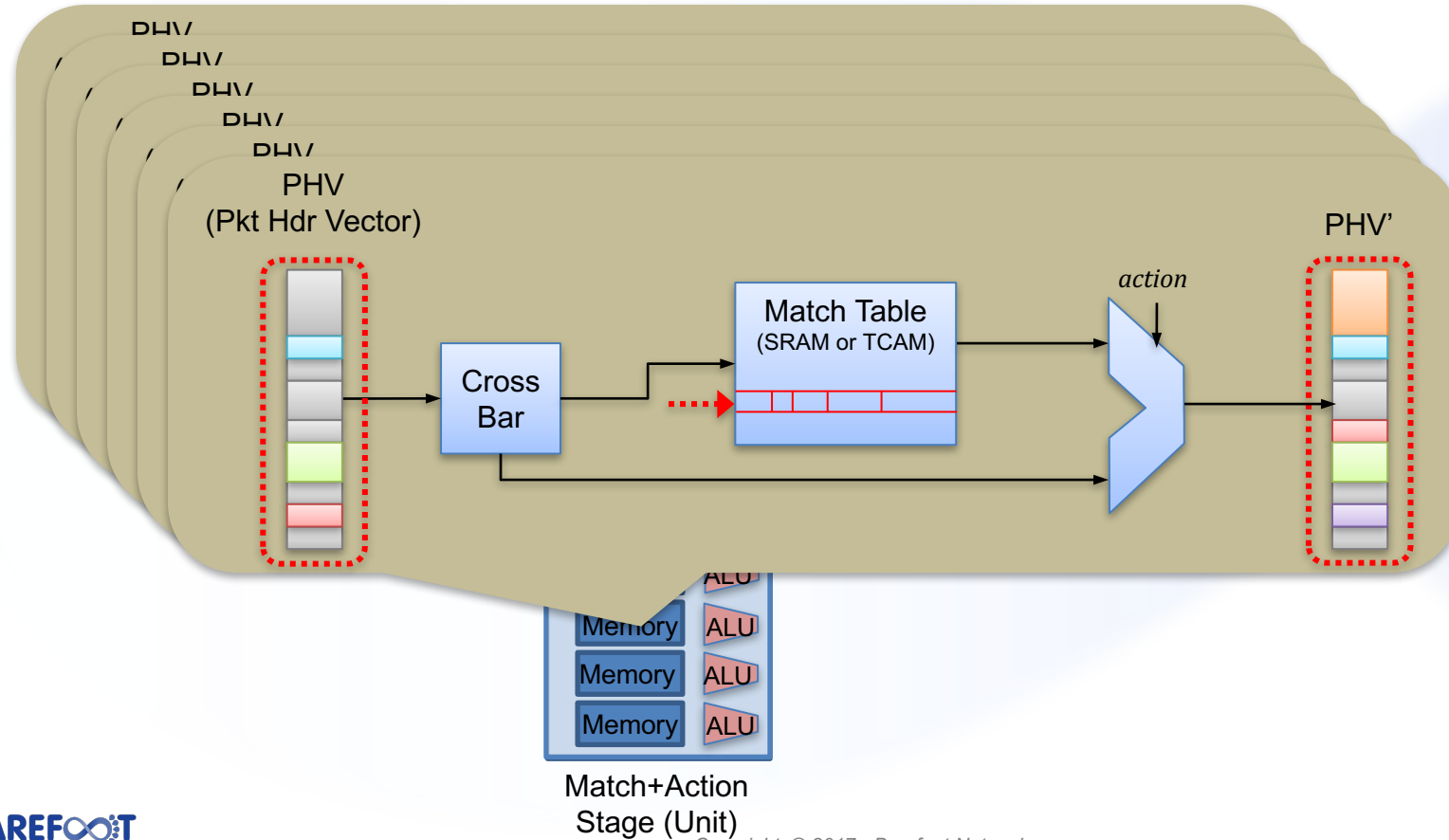
```
16b 0b(0) 0b(1)
0000 && 1fff 05 && 0f 01 -> Row 217 (state parse_icmp)
0000 && 1fff 05 && 0f 06 -> Row 216 (state parse_tcp)
0000 && 1fff 05 && 0f 11 -> Row 215 (state parse_udp)
0000 && 1fff 05 && 0f 2f -> Row 214 (state parse_gre)
0000 && 1fff 05 && 0f 04 -> Row 213 (state parse_ipv4_in_ip)
0000 && 1fff 05 && 0f 29 -> Row 212 (state parse_ipv6_in_ip)
0000 && 1fff 00 && 0f 02 -> Row 211 (state parse_set_prio_med)
0000 && 1fff 00 && 0f 58 -> Row 210 (state parse_set_prio_med)
0000 && 1fff 00 && 0f 59 -> Row 209 (state parse_set_prio_med)
0000 && 1fff 00 && 0f 67 -> Row 208 (state parse_set_prio_med)
0000 && 1fff 00 && 0f 70 -> Row 207 (state parse_set_prio_med)
Default -> Row 206 (state <leaf>)
```

Previous states: [Row 147](#)

Pipeline Organization



What Happens Inside?



Parallelism in P4

```
apply {
  /* Parallel lookups possible */
  subnet_vlan.apply();
  mac_vlan.apply();
  protocol_vlan.apply();
  port_vlan.apply();

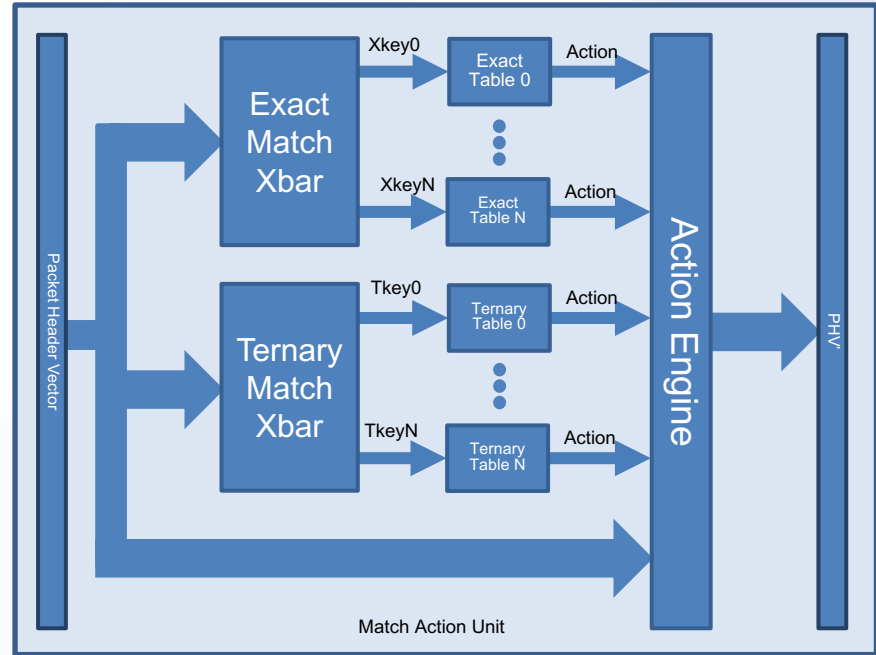
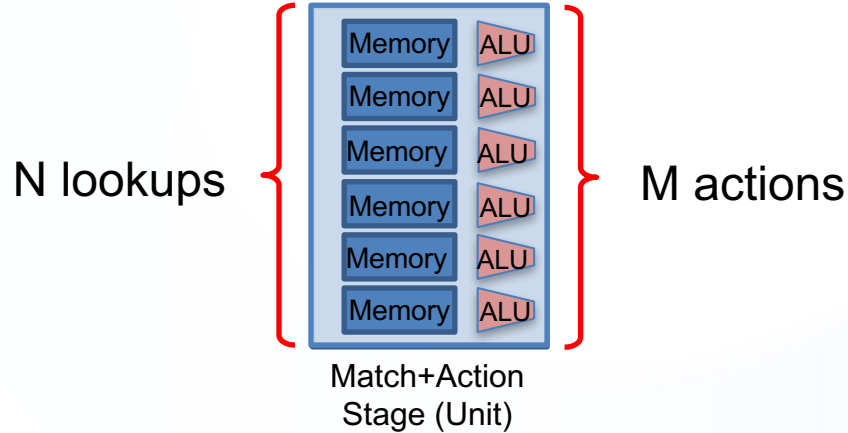
  /* Resolution in next stage */
  resolve_vlan.apply();
}

apply {
  if (!subnet_vlan.apply().hit) {
    if (!mac_vlan.apply().hit) {
      if (!protocol_vlan.apply().hit) {
        port_vlan.apply();
      }
    }
  }
}
```

- **Most P4 programs have inherent parallelism**
- **Others can be executed speculatively**
- **Switch.p4**
 - ~100 tables and if() statements
 - ~22 stages divided between ingress and egress
 - Degree of parallelism ~4.5

How Tofino Supports Parallel Processing

- Multiple tables mean multiple parallel lookups
- All actions from all active tables are combined



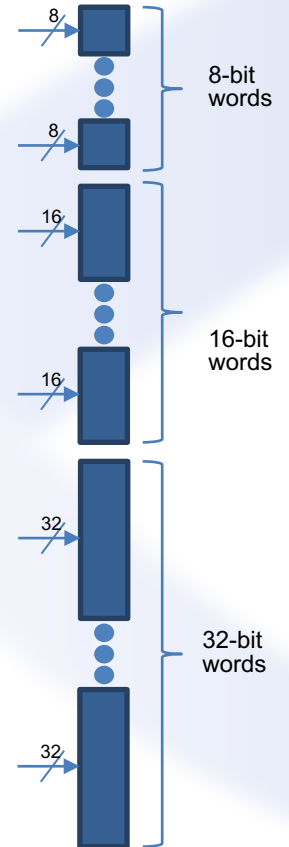
Parallelism in P4

```
action ipv4_in_mpls(in bit<20> label1, in bit<20> label2) {
  hdr.mpls[0].setValid();
  hdr.mpls[0].label = label1;
  hdr.mpls[0].exp = 0;
  hdr.mpls[0].bos = 0;
  hdr_mpls[0].ttl = 64;

  hdr.mpls[1].setValid();
  hdr.mpls[1] = { label2, 0, 1, 128 };

  if (hdr.vlan_tag.isValid()) {
    hdr.vlan_tag.etherType = 0x8847;
  } else {
    hdr.ethernet.etherType = 0x8847;
  }
}
```

- **Most actions can be easily parallelized**
 - This action can be executed in 1 cycle
 - Number of parallel operations: 12
- **Keep fields in separate containers**



P4 Visualizations (PHV Allocation)



switch PHV Allocation

file:///Users/vgurevich/trees/bfr/p4examples/switch/switch.tofino/visualization/phv_allocation.html

Stage 1

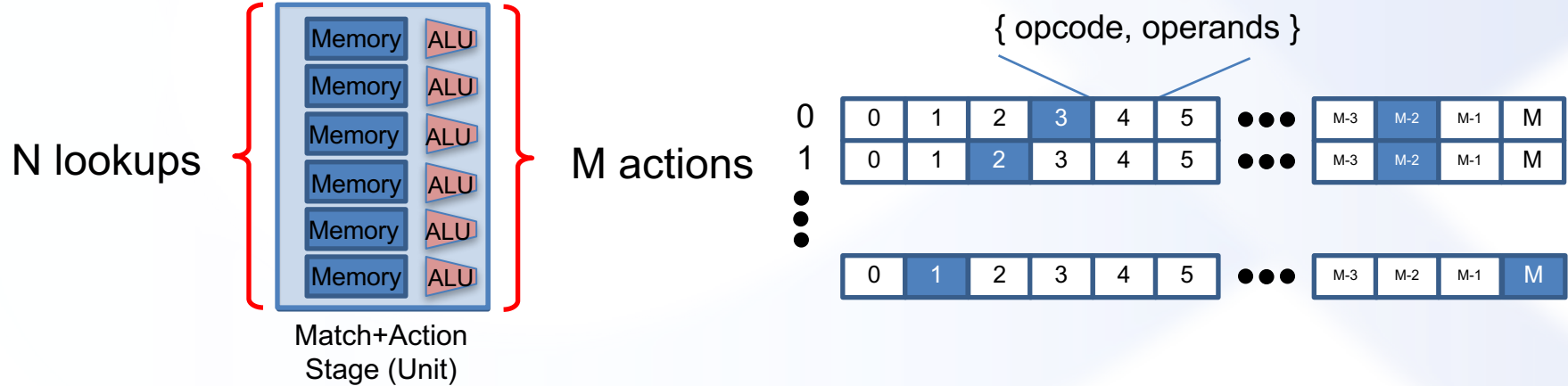
PHV Group: 1
Assigned to Ingress
Container Bit Width: 32
Container Address: 26

inner_ethernet.dstAddr[31:0] in container bits [31:0]
arp_rarp_ipv4.dstProtoAddr[31:0] in container bits [31:0]

Field inner_ethernet.dstAddr read by table mpls's action terminate_eompls
Field inner_ethernet.dstAddr read by table mpls's action terminate_vpls

How Tofino Supports Parallel Processing

- Multiple tables mean multiple parallel lookups
- All actions from all active tables are combined
- Each PHV container has its own, independent processor



Dependency Analysis

- **Independent tables**
- **Match Dependency**
- **Action Dependency**

Independent Tables

```
action ing_drop() {
    modify_field(ing_metadata.drop, 1);
}

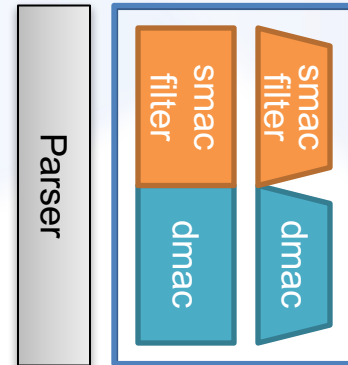
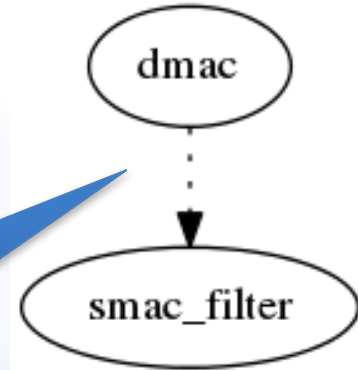
action set_egress_port(egress_port) {
    modify_field(ing_metadata.egress_spec, egress_port);
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
    }
    actions {
        nop;
        set_egress_port;
    }
}

table smac_filter {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        ing_drop;
    }
}

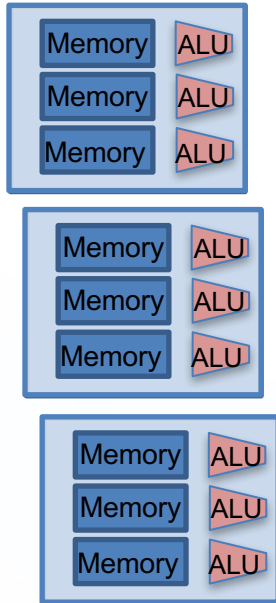
control ingress {
    apply(dmac);
    apply(smac_filter);
}
```

Tables are independent: both matching and action execution can be done in parallel



Match and Action are Separate Phases

- Parallel Execution (No Dependencies)



Total Latency = 1.1

Action Dependency

```
action ing_drop() {
    modify_field(ing_metadata.drop, 1);
}

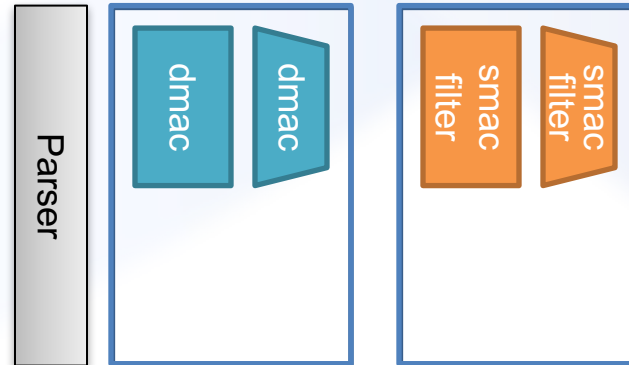
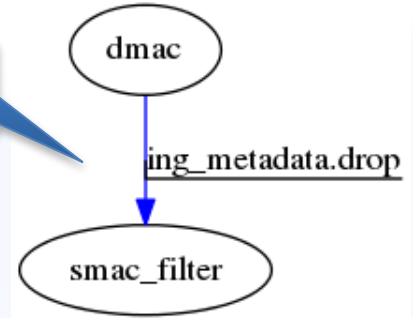
action set_egress_port(egress_port) {
    modify_field(ing_metadata.egress_spec, egress_port);
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
    }
    actions {
        nop;
        ing_drop;
        set_egress_port;
    }
}

table smac_filter {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        ing_drop;
    }
}

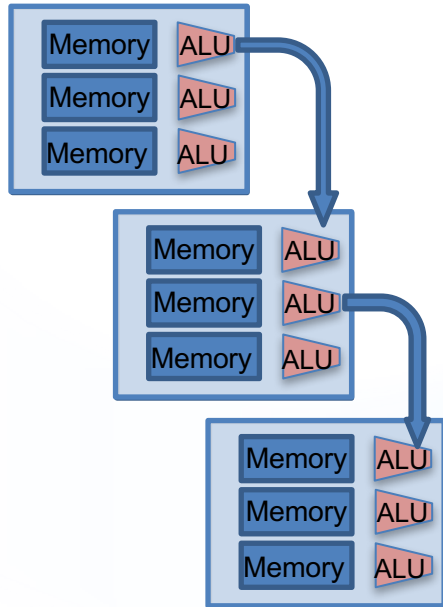
control ingress {
    apply(dmac);
    apply(smac_filter);
}
```

Tables act on the same field and therefore must be placed in separate stages



Match and Action are Separate Phases

- Staggered Execution (Action Dependency)



Total Latency = 2

Match Dependency

```
action set_bd(bd) {
    modify_field(ing_metadata.bd, bd);
}

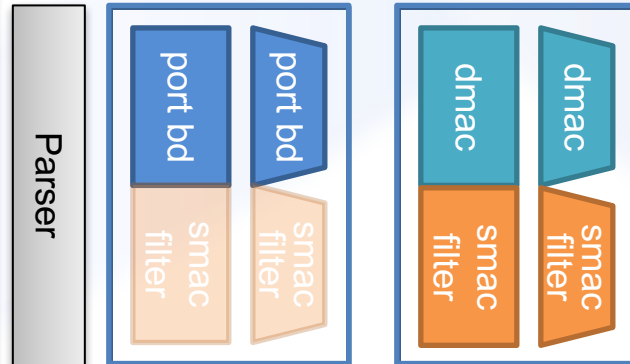
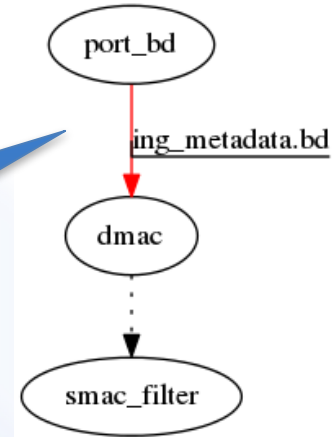
table port_bd {
    reads {
        ing_metadata.ingress_port : exact;
    }
    actions {
        set_bd;
    }
}

table dmac {
    reads {
        ethernet.dstAddr : exact;
        ing_metadata.bd : exact;
    }
    actions {
        nop;
        set_egress_port;
    }
}

table smac_filter {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        nop;
        ing_drop;
    }
}

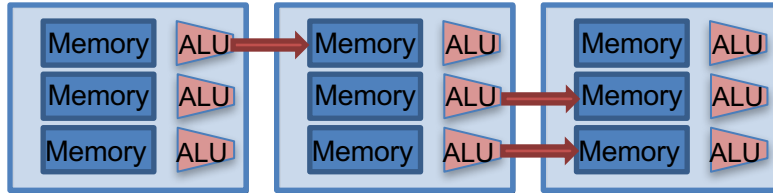
control ingress {
    apply(port_bd);
    apply(dmac);
    apply(smac_filter);
}
```

The second table matches on the field, modified by the first.



Reducing Latency: Match and Action are Separate Phases

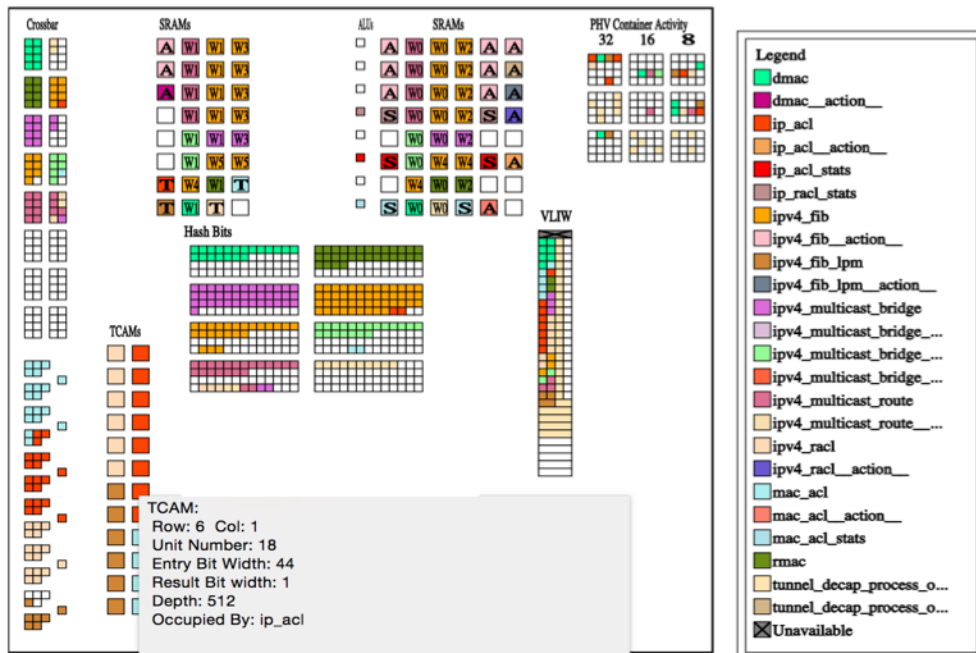
- Sequential Execution (Match dependency)



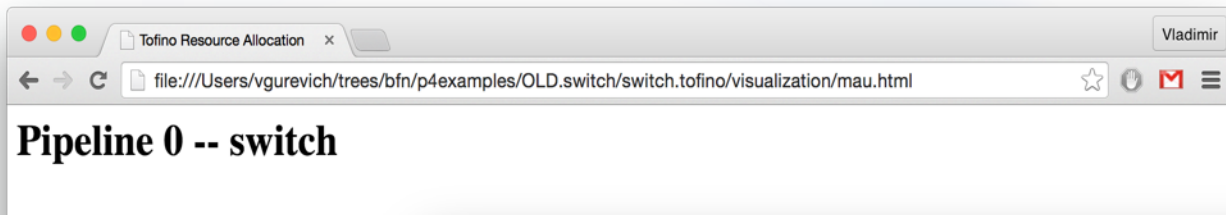
Total Latency = 3

P4 Visualizations (Resource Allocation)

MAU Stage 4



P4 Visualizations (Resource Usage Summary)



Pipeline 0 -- switch

Resource Usage Summary

Stage Number	Exact Match Input Crossbar	Ternary Match Input Crossbar	Hash Bit
0	18	29	100
1	38	20	144
2	16	5	65
3	25	11	147
4	66	62	224
5	15	7	133
6	6	7	93
7	11	10	77
8	20	0	103
9	78	52	140

A screenshot of a web browser window displaying the 'Resource Percentage Summary' table. The browser title is 'Pipeline 0 -- switch' and the address bar shows the same file path as the previous window. The table contains 15 columns representing different resource types and 10 rows representing stages 0 through 9. Each cell in the table contains a percentage value, with some cells highlighted in green, yellow, or red to indicate usage levels.

Resource Percentage Summary

Stage Number	Exact Match Input Crossbar	Ternary Match Input Crossbar	Hash Bit	Hash Distribution Unit	Gateway	SRAM	Map RAM	TCAM	VLIW Instructions	Meter ALU	Statistics ALU	Stash	Action Data Bus Bytes	Logical Table ID
0	14.06%	43.94%	24.04%	0.00%	43.75%	98.75%	0.00%	33.33%	38.71%	0.00%	0.00%	0.00%	25.62%	50.00%
1	29.69%	30.30%	34.62%	0.00%	31.25%	36.25%	0.00%	50.00%	32.26%	0.00%	0.00%	0.00%	15.62%	56.25%
2	12.50%	7.58%	15.62%	0.00%	18.75%	33.75%	0.00%	8.33%	19.35%	0.00%	0.00%	0.00%	14.38%	37.50%
3	19.53%	16.67%	35.34%	0.00%	6.25%	98.75%	62.50%	8.33%	29.03%	0.00%	0.00%	0.00%	4.38%	18.75%
4	51.56%	93.94%	53.85%	0.00%	37.50%	86.25%	16.67%	100.00%	83.87%	0.00%	75.00%	0.00%	30.00%	68.75%
5	11.72%	10.61%	31.97%	0.00%	0.00%	98.75%	0.00%	100.00%	29.03%	0.00%	0.00%	0.00%	21.88%	31.25%
6	4.69%	10.61%	22.36%	16.67%	6.25%	48.75%	18.75%	8.33%	25.81%	0.00%	25.00%	0.00%	14.38%	18.75%
7	8.59%	15.15%	18.51%	0.00%	12.50%	13.75%	0.00%	8.33%	16.13%	0.00%	0.00%	0.00%	9.38%	18.75%
8	15.62%	0.00%	24.76%	0.00%	12.50%	46.25%	4.17%	0.00%	25.81%	25.00%	0.00%	0.00%	13.75%	18.75%
9	60.94%	78.79%	33.65%	50.00%	25.00%	40.00%	12.50%	41.67%	61.29%	25.00%	50.00%	0.00%	23.75%	43.75%

Conclusions

- **Programmable Data Plane at the highest performance point is a reality now**
- **P4 and real-world programs provide a plenty of optimization opportunities**

Thank you

