
intel
connectivity
academy

intel connectivity academy

Introduction to P4 and Data Plane Programmability

Vladimir Gurevich, Intel
Principal Engineer, Switch & Fabric Group (XFG)
Director, Intel Connectivity Academy

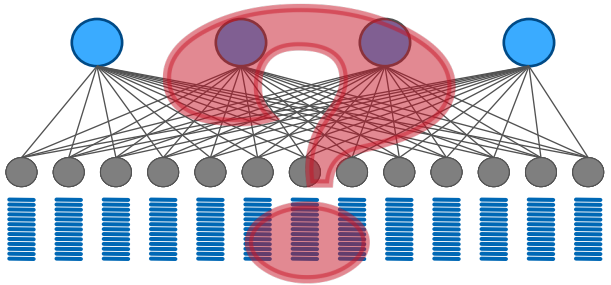
February 16, 2022

Agenda

- Why do we need programmable switches?
- Is it possible to build a programmable switch?
- How can you describe packet processing algorithms?
- Next steps
- Q & A

Innovation in Networking?

- **Having established standards didn't kill innovation (yet)**
- **Main Drivers (the power of scale and centralized control)**
  - Big Scale Data Centers
  - Large ISPs
  - Governments
- **Main Goals**
  - Scale
  - Flexibility
  - Efficiency
  - New use cases of the old Ethernet (reliable communication)
- **Network infrastructure continues to evolve**
  - But very slowly…



How to create a new killer product?

The World of Computing

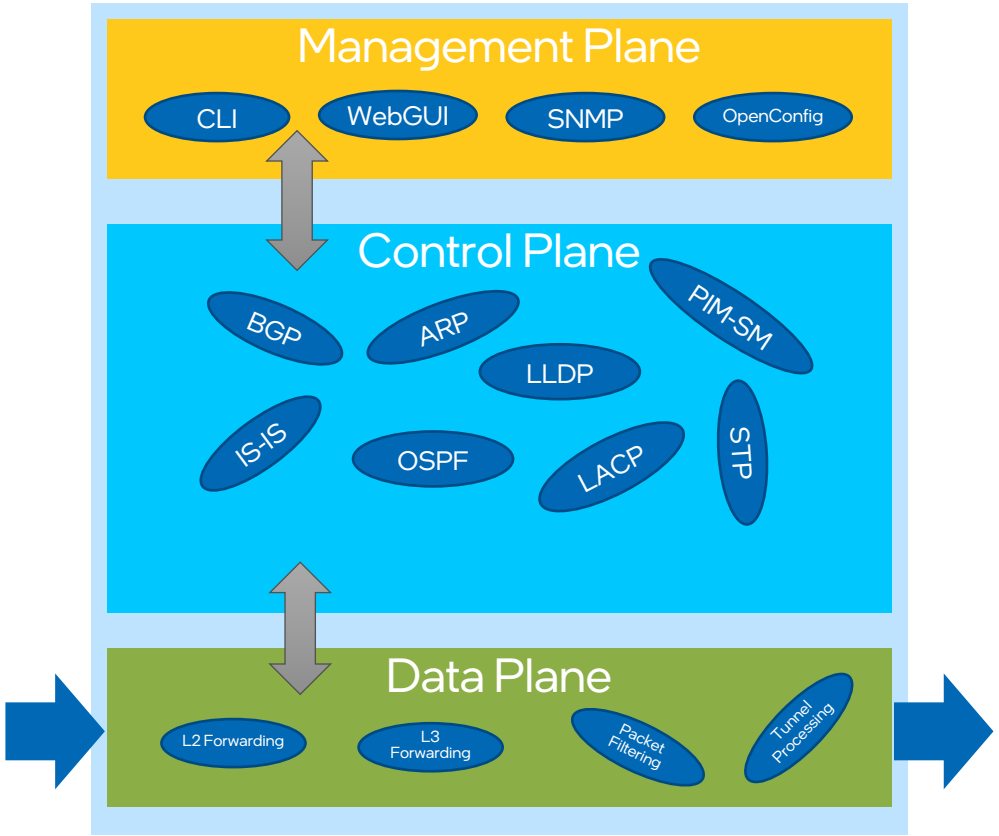
- **Buy a computing device**
  - An embedded board
  - A computer
  - A VM in the cloud
- **Write a program in a high-level language**
- **Profit!**

The World of Networking

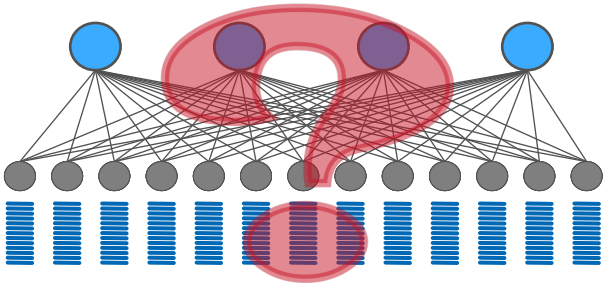
- **Beg the OEM**
- **Beg the ASIC Vendor**
- **Wait 2 years**
- **Forget it!**

Standard Telecommunications System Architecture

- **Three Classes of Algorithms**
  - Data (Forwarding) Plane
  - Control Plane
  - Management (Configuration) Plane
- **Data Plane ultimately determines the system performance and functionality**

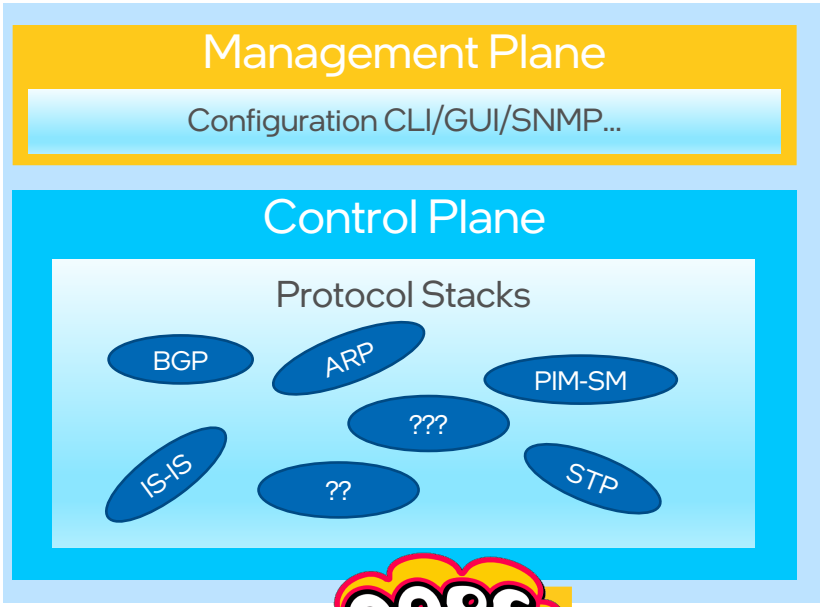


Network Equipment Designer Challenge



That's the system I want to build!

Requirements



- **Data Plane Algorithms are <u>not</u> an integral part of a Network OS ☹**
  - They are built into the switch ASIC
  - Not very well documented
  - Emphasize standard-compliant processing
  - Require years to develop and even modify

And here is mine!

Here is my Packet Processing Algorithm!

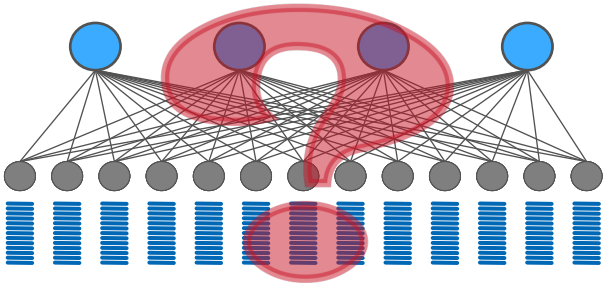
Hardware for Data Plane (before Intel/Barefoot)

	Extra Hardware	Practical Throughput	Development System	Programmability	Power per port	Price
Switch ASIC		12.8 Tbit/s		Low	1	1
General Purpose CPU	Ethernet NIC	~200 Gbit/s	DPDK + C	Very High	10-100x	10-100x
NPU		~400 Gbit/s	C-Like	Very High	10x	10x
FPGA		~100 Gbit/s	Verilog, etc.	Very High	20x	20x

"Programmable switches are 10-100 times slower than non-programmable ones. They are more expensive and consume more power"

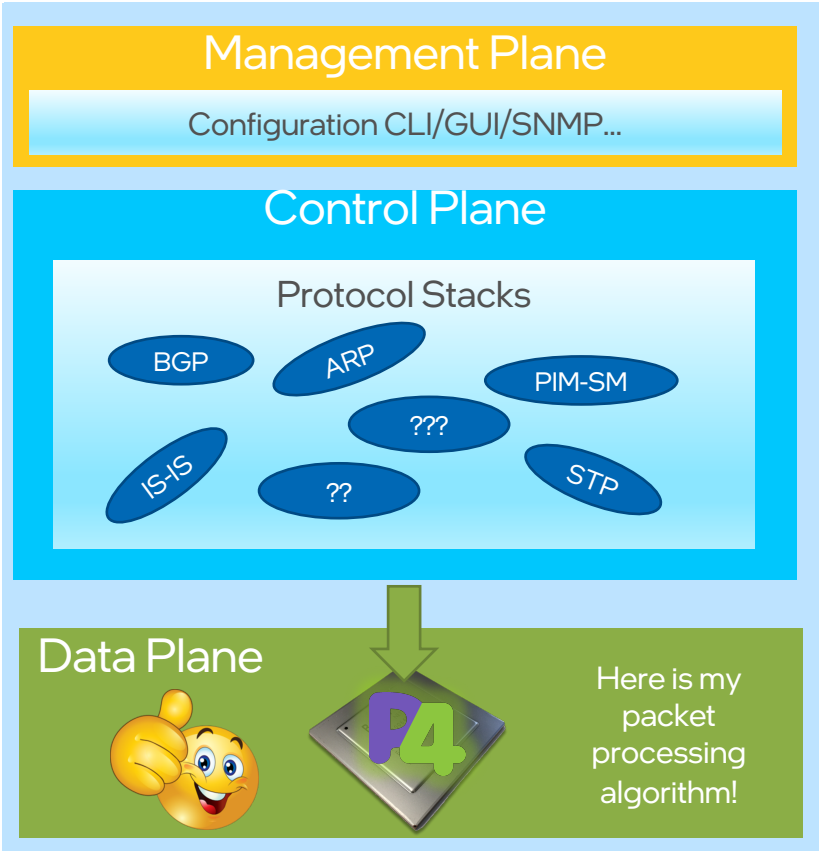
Common "wisdom"

Network Equipment Design with a Programmable ASIC



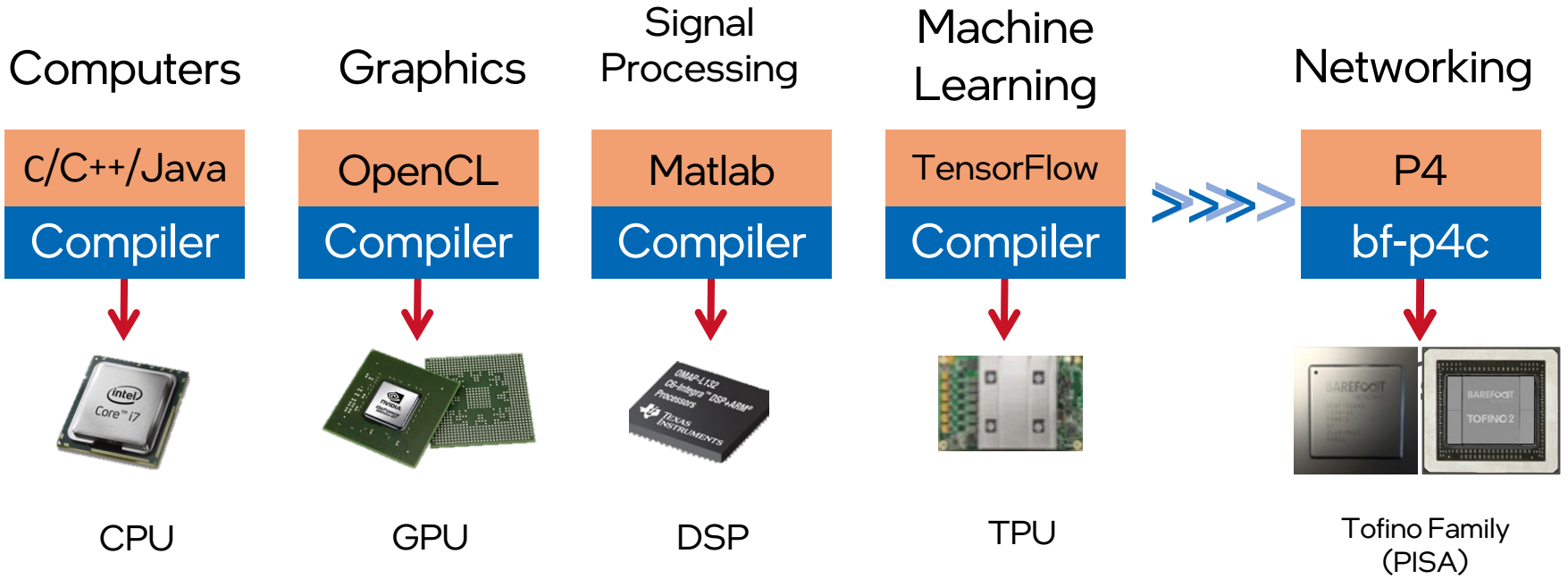
This is how my network equipment should work!

Requirements



- **Developers should have full control of their systems**
  - Packet processors must be programmable
  - We need a high-level language to express data plane algorithms
    - Without compromising performance

Standard Evolution of Computing



- **From a fixed algorithm to a programmable solution!**
  - Application-specific Hardware – Domain-Specific Programming Architecture
  - Fixed Algorithm – Domain-Specific Programming Language

Advantages of the programmable solutions

- **Standard Use Cases**
  - Full control over the features and table scale
  - Formal algorithm description with the possibility of formal verification
  - Better energy efficiency
  - One SKU – multiple roles
  - Tracing and Telemetry
- **Non-standard Use Cases and Protocols**
  - They can be implemented!
    - Load Balancers
    - Network Cache
    - In-network computing
    - Non-standard, experimental protocols
  - Data plane algorithm is “just a program”.
  - No need to disclose IP to anyone

Programmable Switches are a Reality Now!

64 x 100GE
Legacy,
Fixed Function ASIC

Parameter	Measurement Unit	Comparison
Throughput	Packets/s	21% higher
Power Consumption	Switching Troughput/W (pps/W)	53% lower
Table Scale	ACL, NAT, tunnels	20x
	Routes (IPv4/IPv6)	10x
	ECMP	2x
Non-standard Application Support	Smart Load balancing	∞
	Segment routing	∞
	In-band Telemetry	1000x

64x100GE
Barefoot Tofino

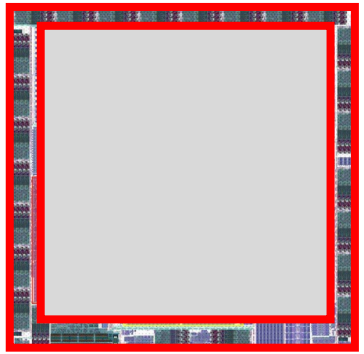
Programmable Switches beat non-programmable ones at the same power/price point!

intel connectivity academy

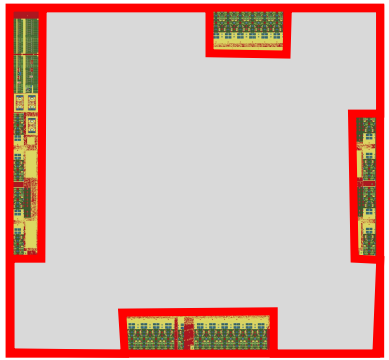
How is this possible?

- The area, dedicated to packet processing logic gets progressively smaller

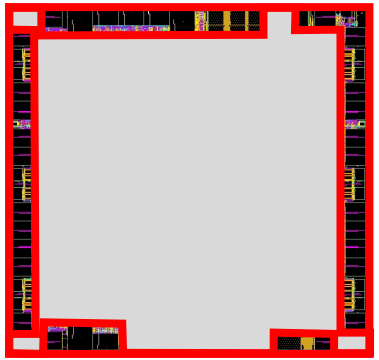
SerDes: 30% area



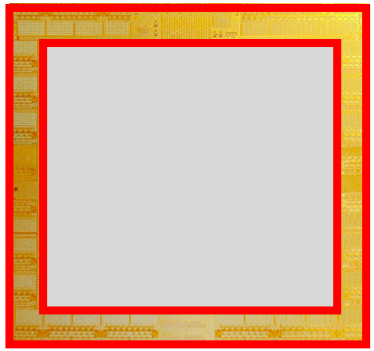
Intel Alta (2011)



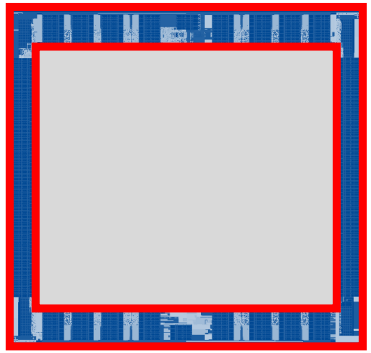
Cisco (2011)



Ericsson Spider (2011)

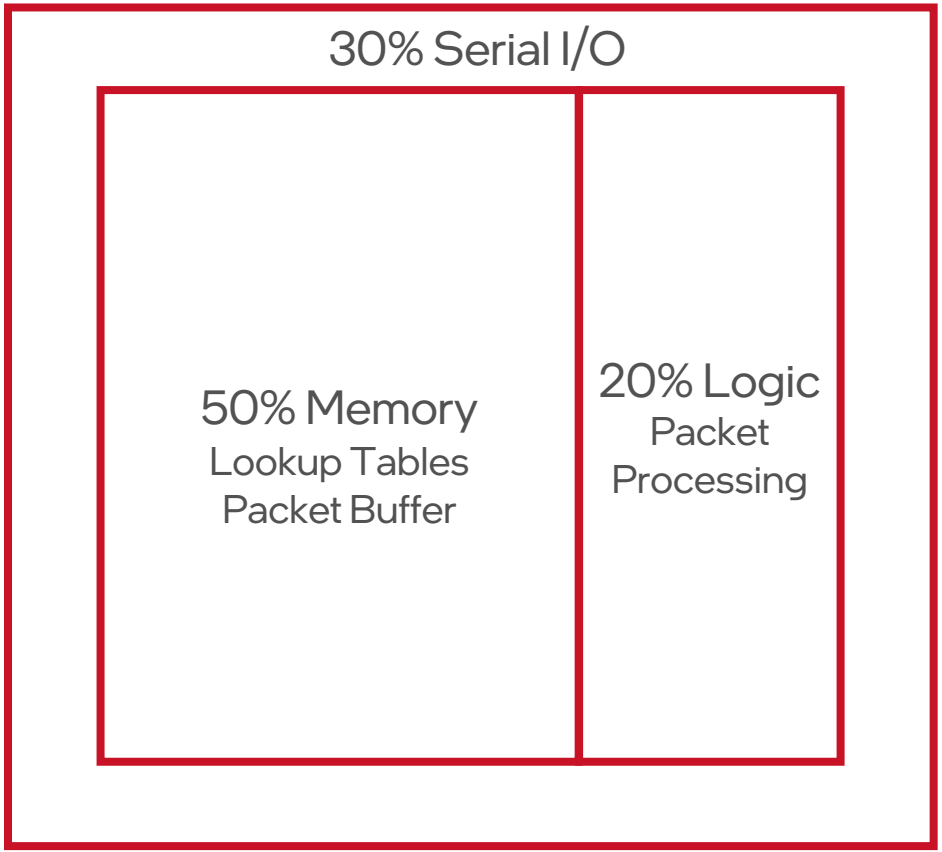


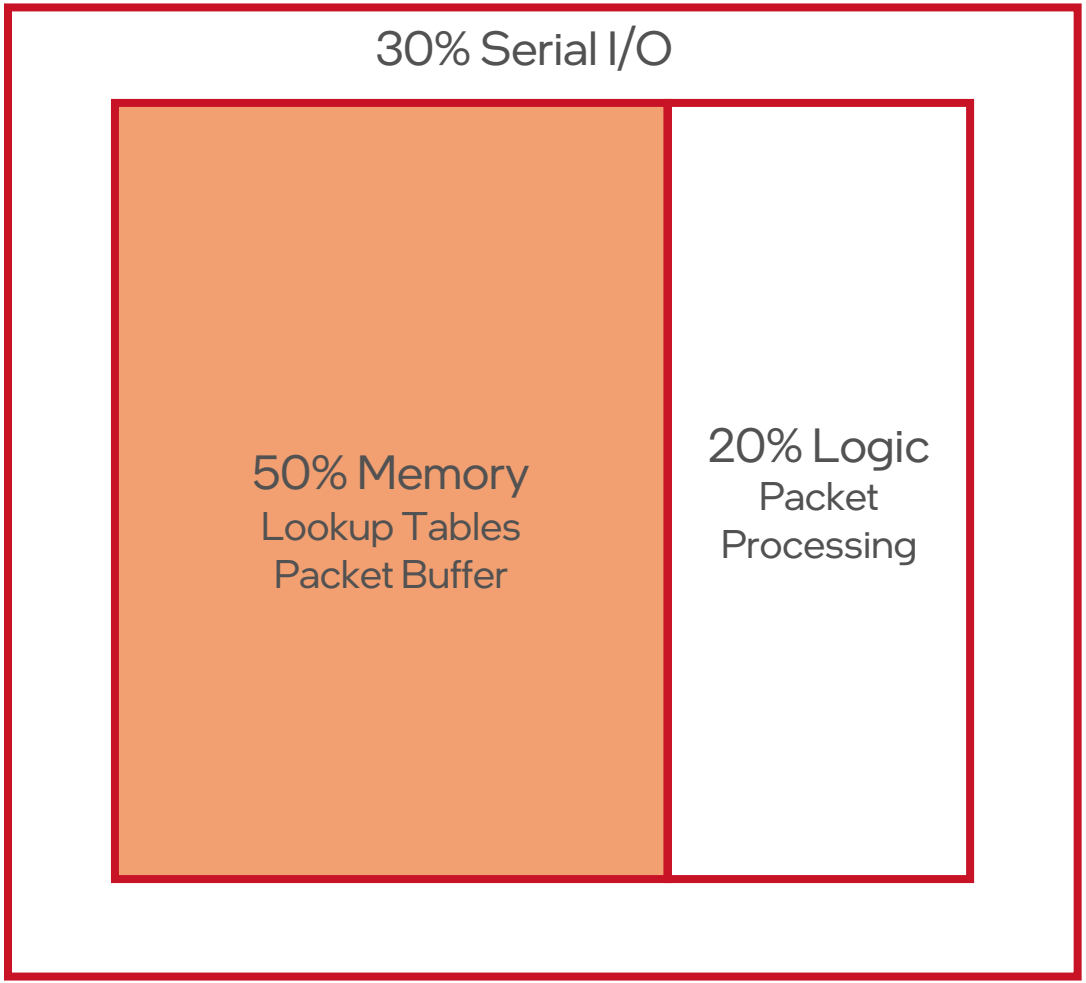
Broadcom Tomahawk (2014)

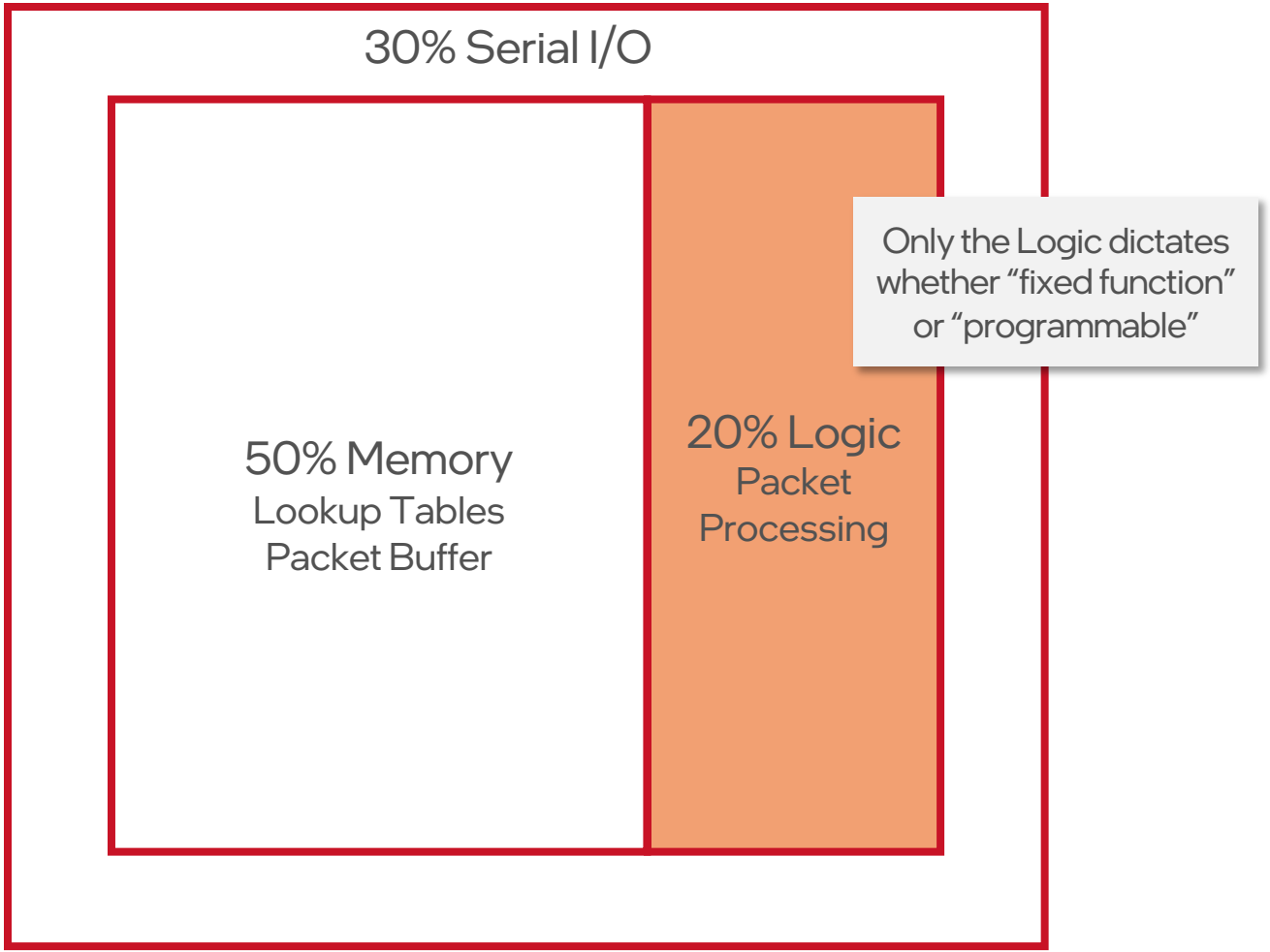


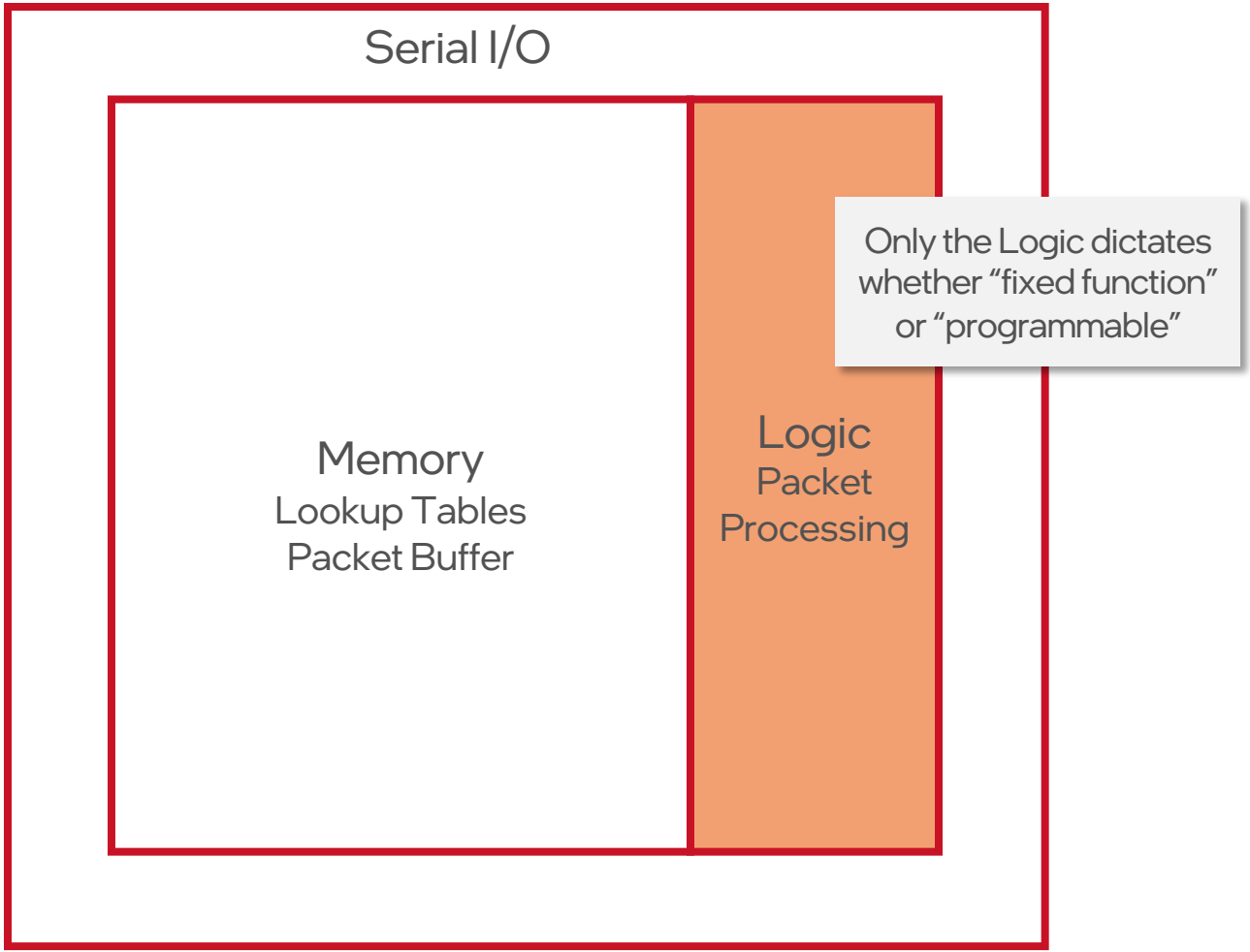
Barefoot Tofino (2016)

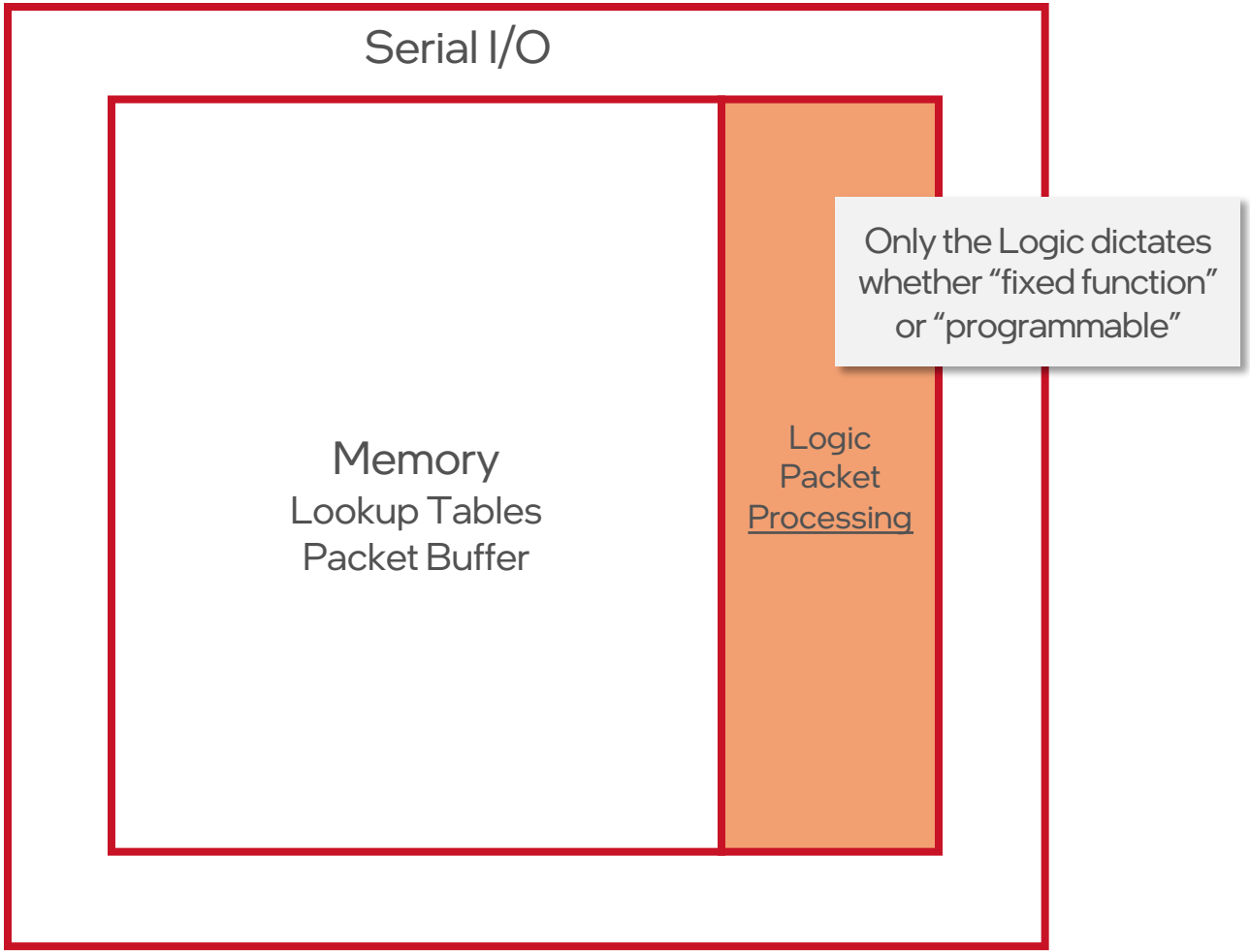
Tables and Packet Buffers: 50% area

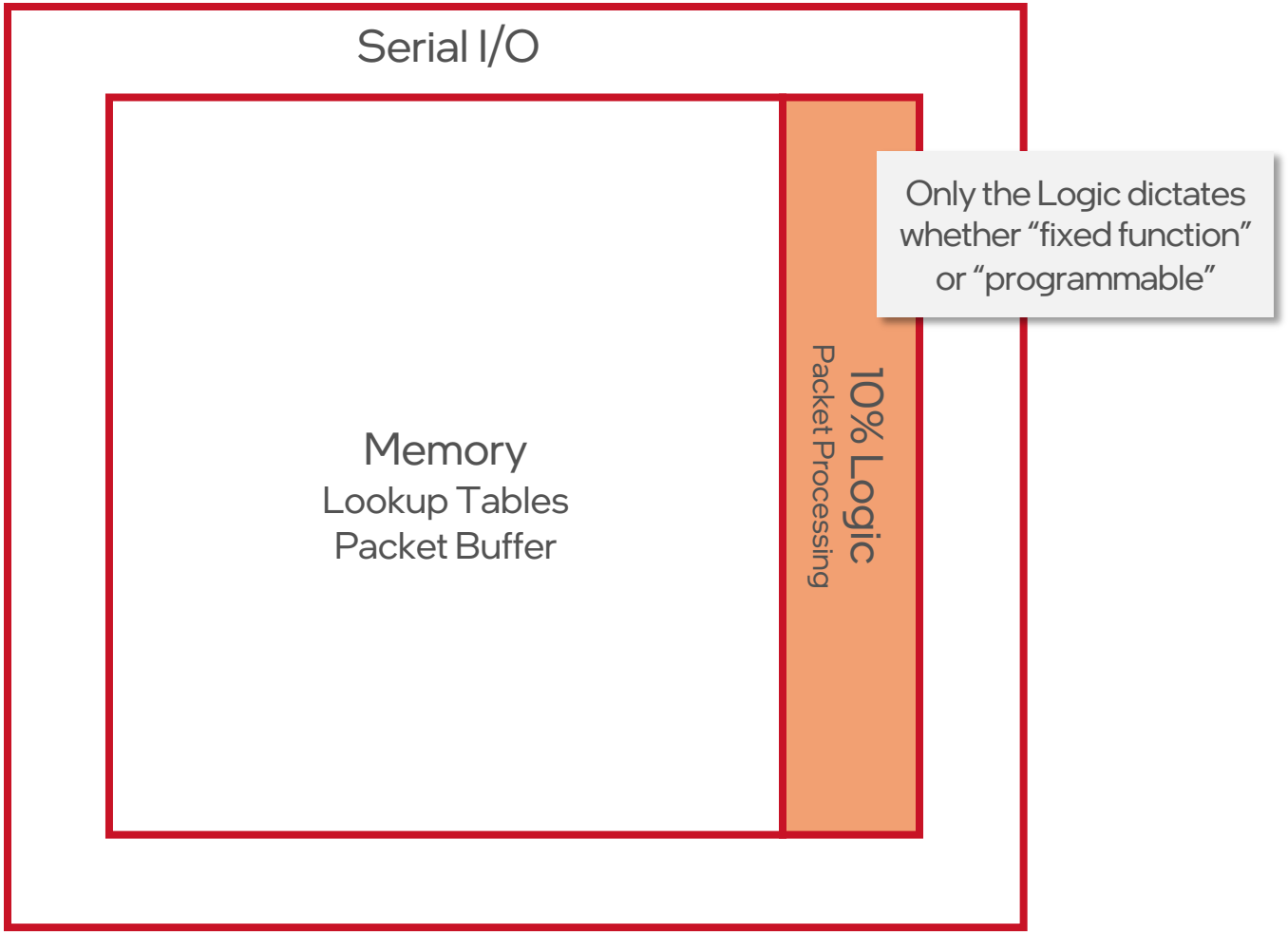






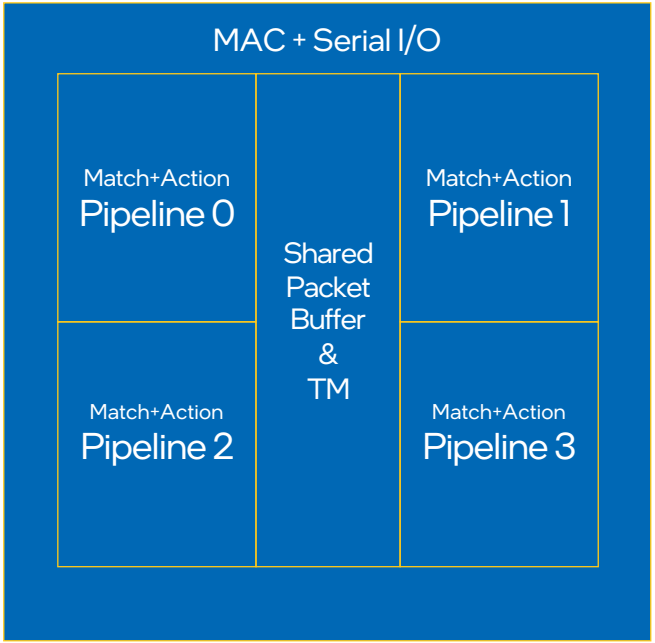






Barefoot Tofino™ (2016)

- 6.5Tb/s switch
- 260 lanes of 25G SerDes
- 260 x 25G Ethernet ports, 130 x 50G, 65 x 100G, or combinations
- 11B transistors
- 16nm technology
- 1220 MHz
- Equivalent in area and power to fixed function chips



Barefoot Tofino 2 (2019)

- 12.8Tb/s Switch
- 256 lanes of 50G SerDes + 4 lanes of 25G SerDes
- 256x10/25/50G ports, 128x100G, 32x400G + 100Gbps CPU
- 7nm process
- 1500 MHz

12.8 Tbps

8.0 Tb/s

6.4 Tb/s

intel connectivity academy

Inside a Programmable Switch

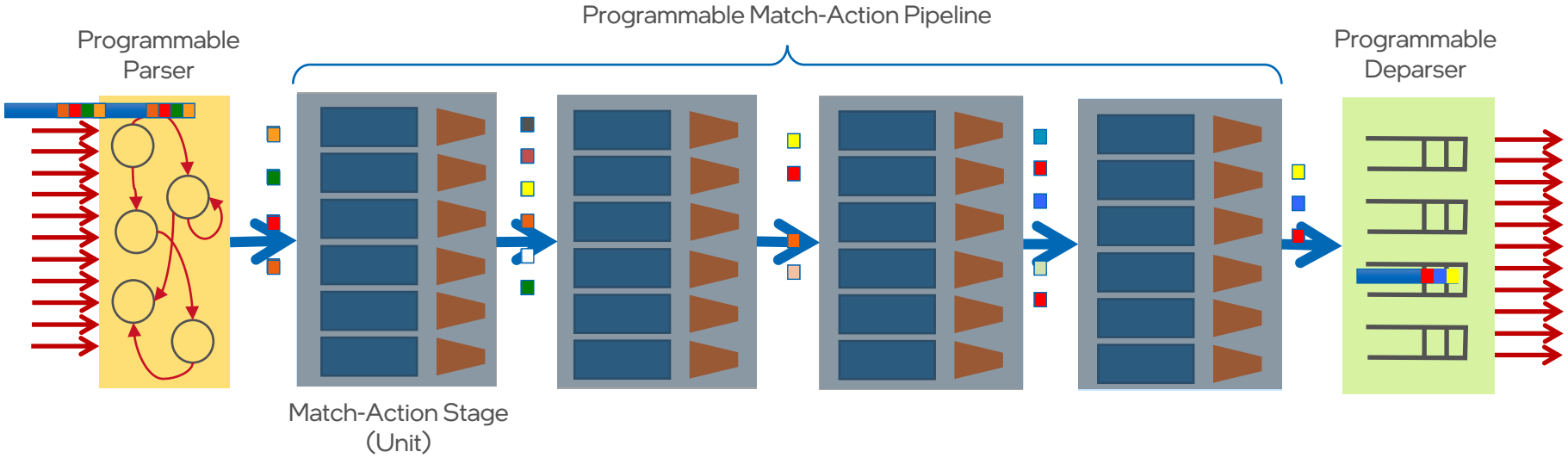
Key Insights

- **Data Plane Algorithms are simple**
  - Simple packet header operations
  - Simple switch state manipulation
  - Simple controls for external components
- **Most Operations are easily parallelizable**
  - Packet header fields are processed independently
- **Pipelining is the key**

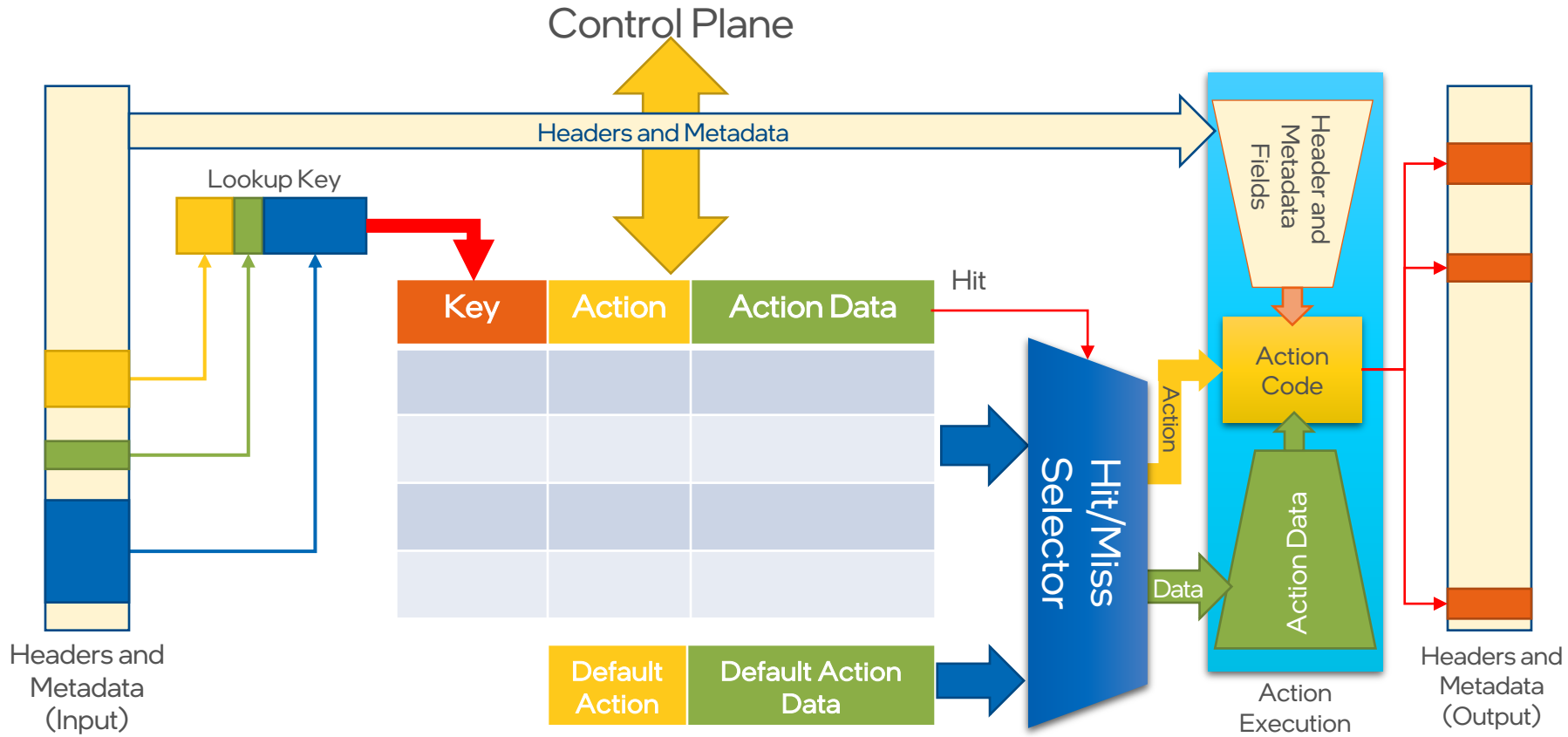
Protocol Independent Switching Architecture (PISA) in Action

- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is deparsed (serialized)

- Feed-forward architecture
  - Constant processing latency
- Stage-local resources
  - Multiple simultaneous lookups are possible
- One packet per clock
  - Many packets are processed in parallel



Match-Action Table Operation



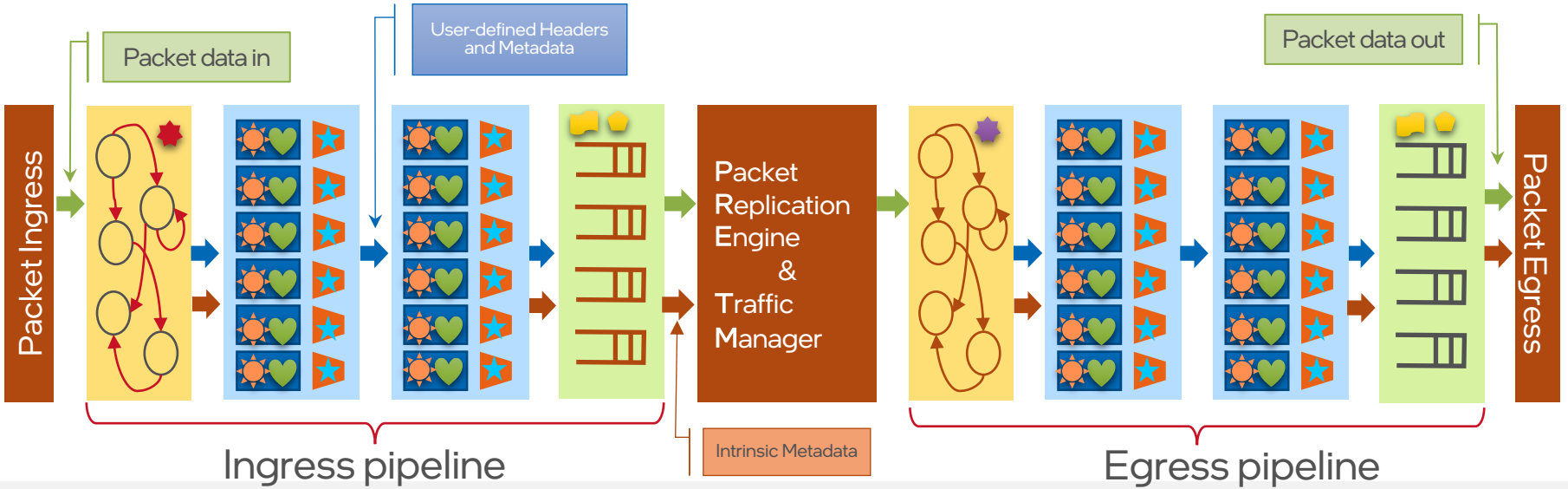
Tofino Native Architecture (TNA)

Programmable components

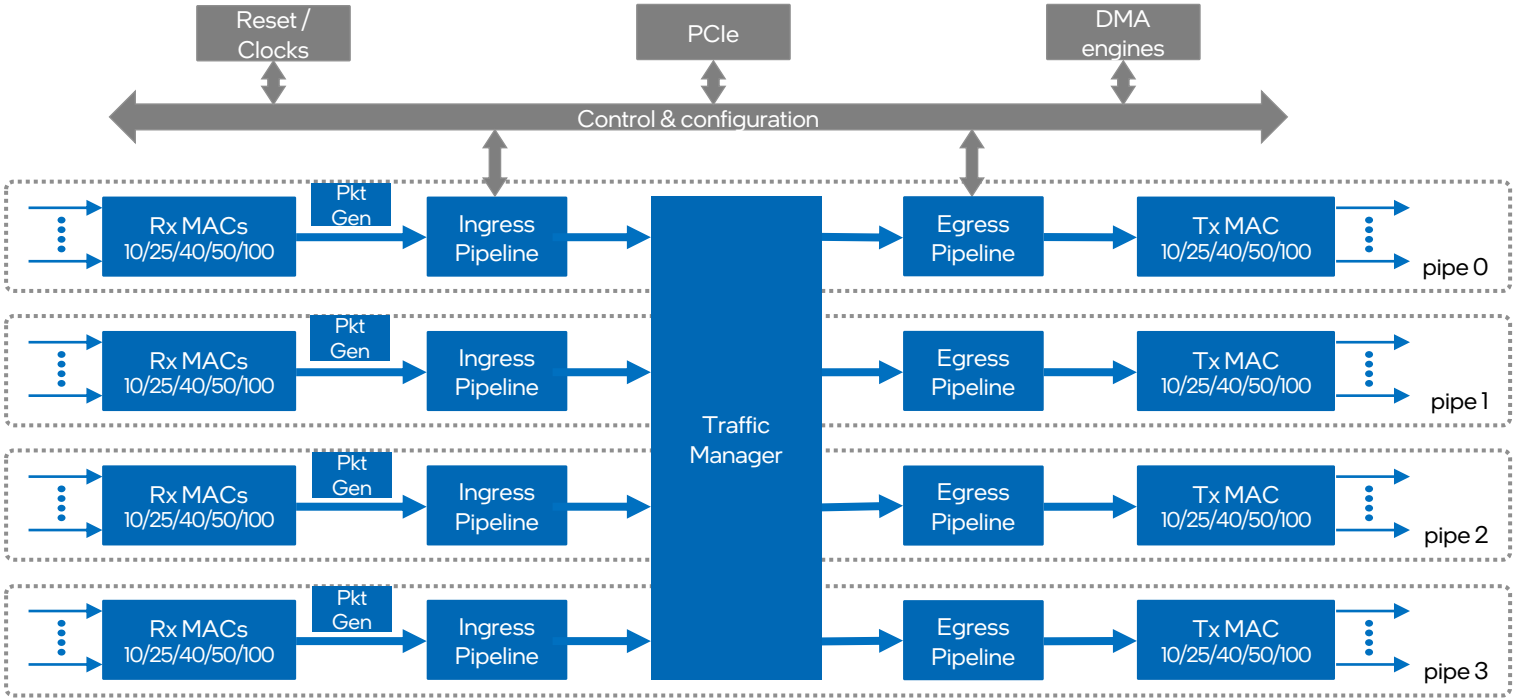
- Ingress and Egress Pipeline
  - Parser
  - Match-Action Pipeline
  - Deparser
- Tofino-specific Intrinsic Metadata
- Tofino-Specific Externs

Fixed-Function Components

- Packet I/O (MAC/SerDes/DMA)
- Packet Replication Engine
- Traffic Manager

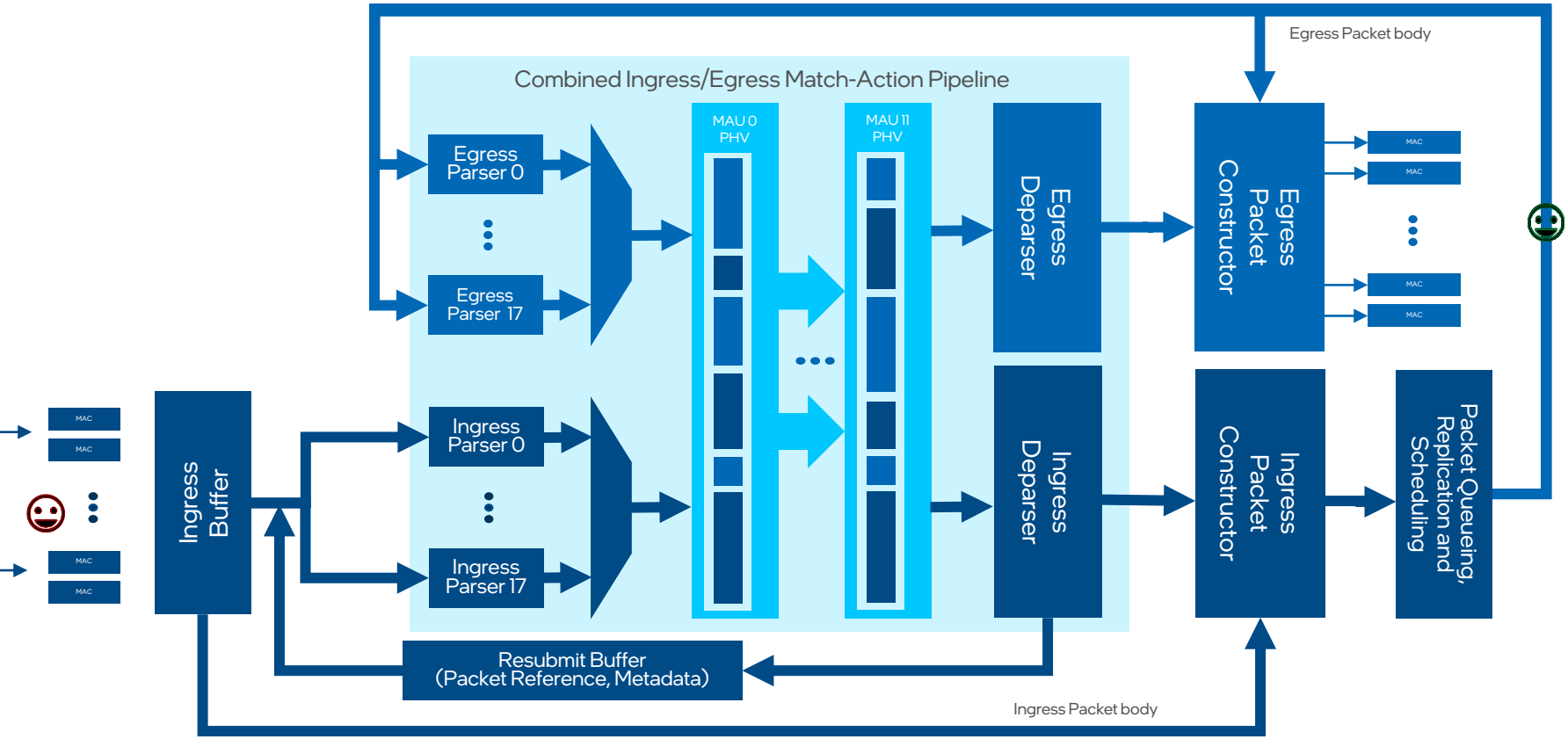


Simplified Tofino Block Diagram (Theme: Parallelization)



Each pipe has 16x100G MACs
Additional ports for recirculation, Packet Generator, CPU (18 Quads total)

Pipe Organization: Pipelined packet processing



intel connectivity academy

Describing Data Plane Algorithms

- $P4_{16}$ Programming Language

P4 Design Goals

- **Data Plane Programming Language**
  - Simple packet header transformations and external component interfaces
- **Protocol independence**
  - Programmer defines packet header formats and processing algorithms
- **Target independence**
  - No knowledge of low-level hardware organization required
  - Compiler compiles the program for the target device
- **Explicit Language-Architecture separation**
  - Supports any combination of programmable and non-programmable components
  - Supports custom components and extensions
- **Consistent Control Plane Interface**
  - Control Plane APIs are automatically generated by the compiler
- **Reconfigurability**
  - Data Plane program can be changed in the field
- **Community-driven Design**
  - http://p4.org

P4.org Membership

- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Member of Linux Foundation. An ONF Project

Brief History and Trivia

- May 2013: Initial idea and the name "P4"
  - Programming Protocol-Independent Packet Processors
- July 2014: First paper (SIGCOMM CCR)
- Aug 2014: First $P4_{14}$ Draft Specification (v0.9.8)
- Sep 2014: $P4_{14}$ Specification released (v1.0.0)

  . . .
- Nov 2018: $P4_{14}$ v1.0.5

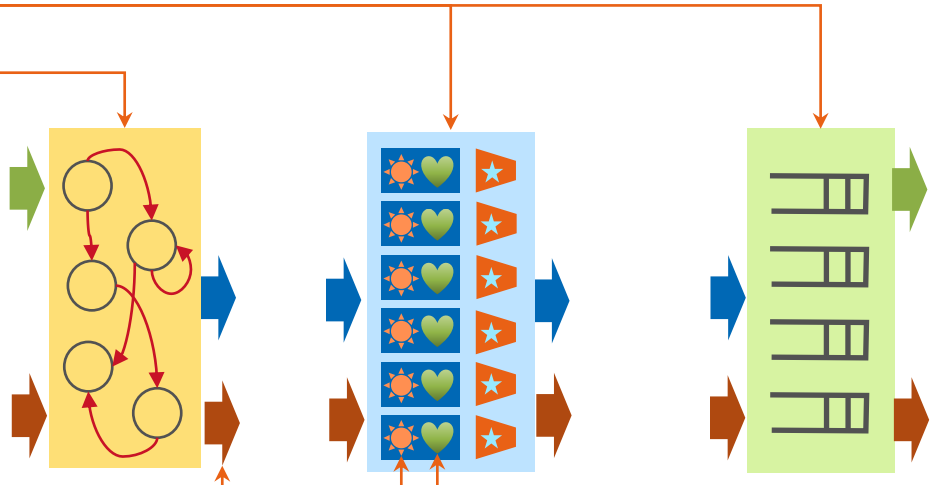
- Apr 2016: $P4_{16}$ – first commits
- Dec 2016: First $P4_{16}$ Draft Specification
- May 2017: $P4_{16}$ Specification released (v1.0.0)
- Nov 2018: $P4_{16}$ v1.1.0
- Nov 2019: $P4_{16}$ v1.2.0
- Jun 2020: $P4_{16}$ v1.2.1
- May 2021: $P4_{16}$ v1.2.2

- Official Spelling `P4_14(P4_16)` on terminals, $P4_{14}$ ($P4_{16}$) in publications

P4$_{16}$ Language Elements

- **Controls** — Tables, Actions, control flow statements
- **Parsers** — State machine, bitfield extraction
- **Expressions** — Basic operations and operators
- **Data Types** — Bitstrings, headers, structures, arrays

- **Architecture Description** — Programmable and non-programmable blocks and their interfaces
- **Externs** — Support for specialized components



Defining Headers and Parsers

```
/***** H E A D E R   D E F I N I T I O N S  *****/
typedef bit<48> mac_addr_t;
typedef bit<32> ipv4_addr_t;

header ethernet_h {
    mac_addr_t   dst_addr;
    mac_addr_t   src_addr;
    ether_type_t ether_type;
}

header vlan_tag_h {
    bit<3>       pcp;
    bit<1>       dei;
    bit<12>      vid;
    ether_type_t ether_type;
}

header ipv4_h {
    bit<4>       version;
    bit<4>       ihl;
    bit<8>       diffserv;
    bit<16>      total_len;
    bit<16>      identification;
    bit<3>       flags;
    bit<13>      frag_offset;
    bit<8>       ttl;
    bit<8>       protocol;
    bit<16>      hdr_checksum;
    ipv4_addr_t  src_addr;
    ipv4_addr_t  dst_addr;
}
```

```
parser MyIngressParser(packet_in      pkt,
    out my_ingress_headers_t          hdr,
    out my_ingress_metadata_t         meta,
    out ingress_intrinsic_metadata_t  ig_intr_md)
{
    state start {
        /* Mandatory code required by Tofino Architecture */
        pkt.extract(ig_intr_md);
        pkt.advance(PORT_METADATA_SIZE);
        transition parse_ethernet;
    }

    state parse_ethernet {
        pkt.extract(hdr.ethernet);
        transition select(hdr.ethernet.ether_type) {
            ether_type_t.TPID:  parse_vlan_tag;
            ether_type_t.IPV4:  parse_ipv4;
            default: accept;
        }
    }

    state parse_vlan_tag {
        pkt.extract(hdr.vlan_tag);
        transition select(hdr.vlan_tag.ether_type) {
            ether_type_t.IPV4 :  parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        pkt.extract(hdr.ipv4);
        transition accept;
    }
}
```

Defining Actions and Tables

```
action send(PortId_t port) {
    ig_tm_md.ucast_egress_port = port;
}

action drop() {
    ig_dprsr_md.drop_ctl = 1;
}

table ipv4_host {
    key = {
        hdr.ipv4.dst_addr : exact;
    }
    actions = {
        send;
        drop;
    }
    size = 131072;
}

table ipv4_lpm {
    key = {
        hdr.ipv4.dst_addr : lpm;
    }
    actions = {
        send;
        drop;
    }
    default_action = send(CPU_PORT);
    size = 12288;
}
```

TNA Intrinsic Metadata

TNA Intrinsic Metadata

- Actions consist of P4 statements
- Actions can have parameters
- Intrinsic metadata controls the device
- Tables specify
  - The key (match fields and match type)
  - List of possible actions
  - Other attributes (default action, size, etc.)

ipv4_host

dst_addr	Action	Action Data
192.168.1.1	send	port=1
192.168.1.2	send	port=2
192.168.1.3	drop	---
Default	NoAction	

To be filled by the control plane!

ipv4_lpm

dst_addr / p_length	Action	Action Data
192.168.1.0 / 24	send	port=1
10.0.16.0 / 22	send	port=5
192.168.0.0 / 16	drop	--
Default	send	port=(CPU_PORT)

Defining the Ingress Control. The Processing Algorithm

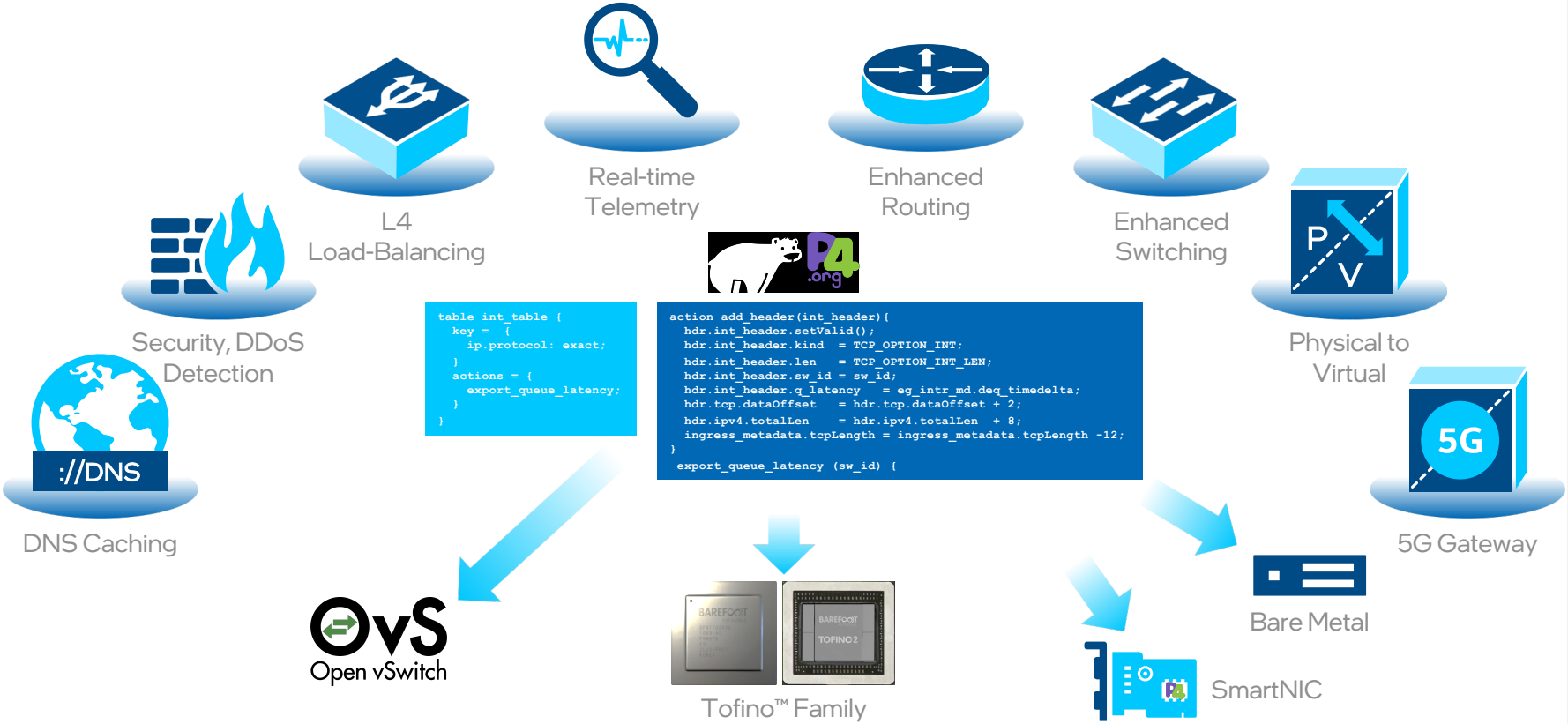
```
control MyIngress (...)
{
    /* Define variables, actions and tables here */
    action send() { ... }
    action drop() { ... }

    table ipv4_host { ... }
    table ipv4_lpm  { ... }

    /* Define the processing algorithm here */
    apply {
        if (hdr.ipv4.isValid()) {
            if (ipv4_host.apply().miss) {
                ipv4_lpm.apply();
            }
        }
    }
}
```

- Direct Acyclic Graph processing (no loops)
- apply {} – the top-level block of statements
  - apply() method of the control you define
- table.apply() – perform match-action processing on a table
  - Perform match
  - Determine the action and action parameters
  - Execute the action
- Optional apply() attributes (returns)
  - table.**apply().hit / miss – Boolean**
  - table.**apply().action_run – special enum**
- Conditional statements
  - Comparison operations (==, !=, >, <, >=, <=)
  - Logical operations (**not**, **and**, **or**)
  - Header validity checks
- Make sure the program operates on valid headers!

P4 Applications



intel connectivity academy

Conclusion and Next Steps

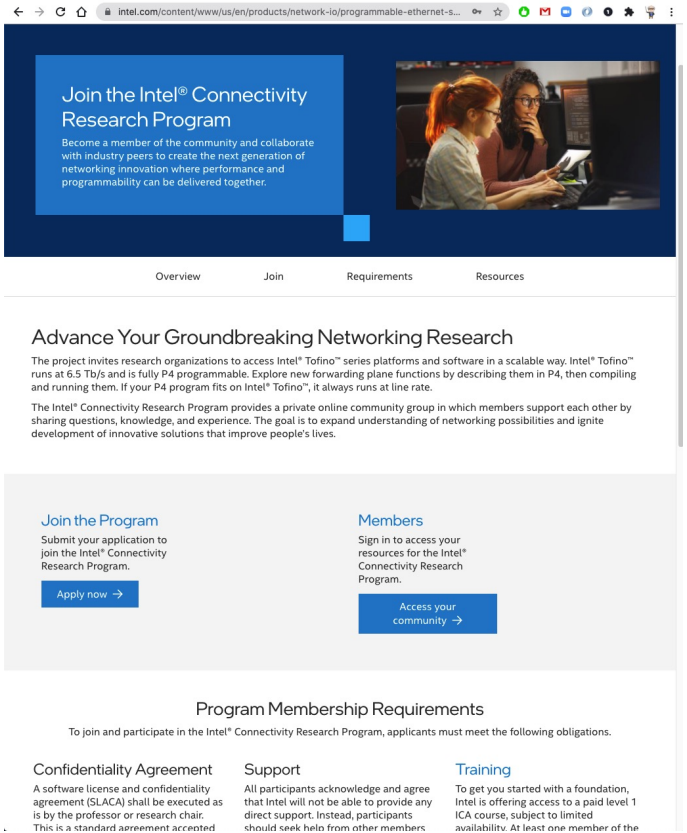
Summary

- **What we discussed**
  - Programmable Packet Processors enable innovation in networking
  - Programmable hardware is a reality today
    - Intel Tofino™ Family
    - $P4_{16}$ language
- **Next Steps**
  - Join Intel Connectivity Research Program (ICRP)
  - Attend Intel Connectivity Academy (ICA)
  - Get the hardware
  - Innovate!

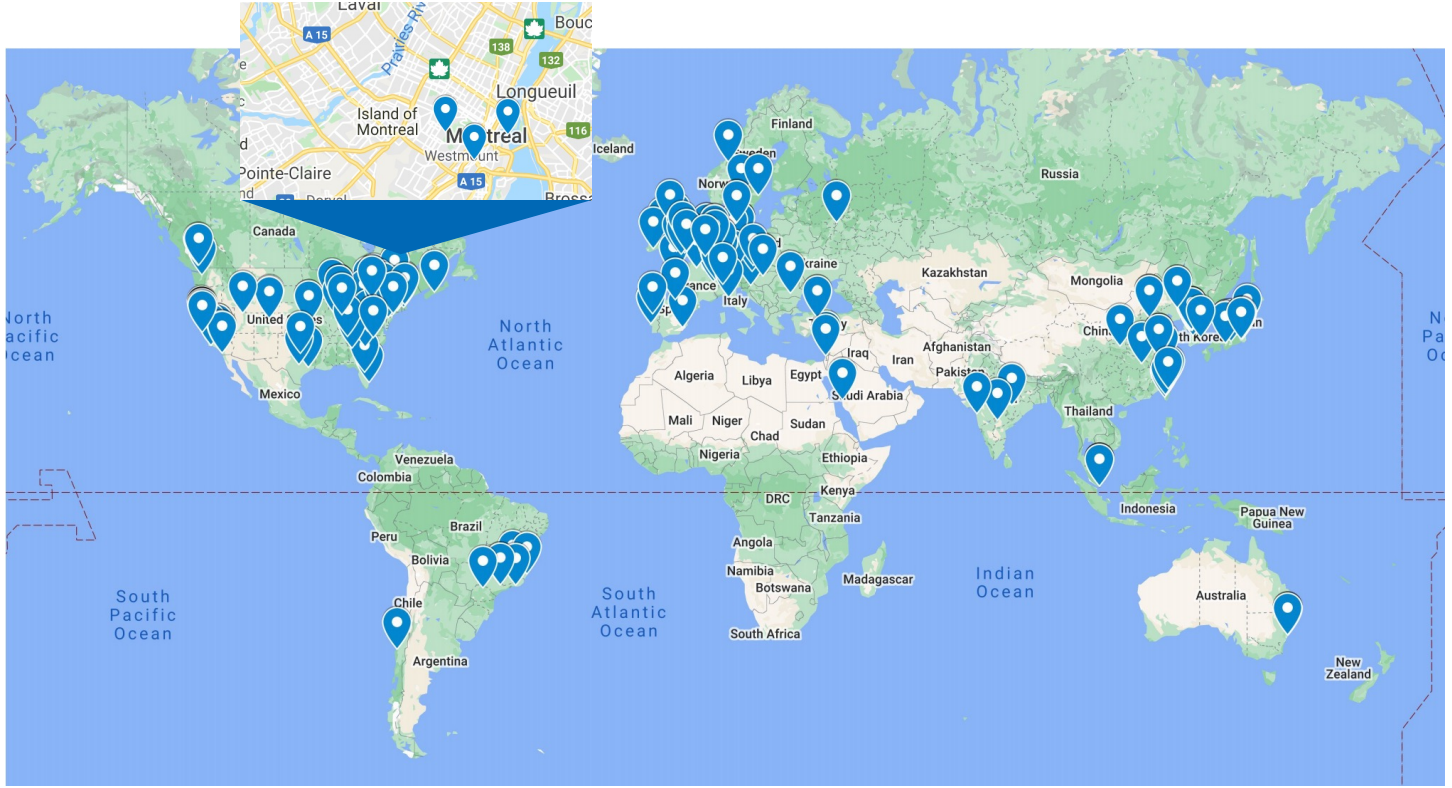
Join Intel Connectivity Research Program (ICRP)

http://intel.com/icrp

- **Free Membership**
- **Simple, click-through application**
  - All academic and research institutions are welcome
  - Established commercial customers are welcome too
    - APS Networks, Edgecore, Senao, Noviflow, ….
- **Rules:**
  - Sign a special NDA/SLA (SLACA) that allows collaboration
  - Buy at least one Tofino-based system
  - Attend Level-1 Intel Connectivity Academy Course
  - Send papers for confidentiality review
- **Benefits:**
  - Online Support Forum
  - Generous discounts for training, equipment and software
  - Free editorial services for the papers



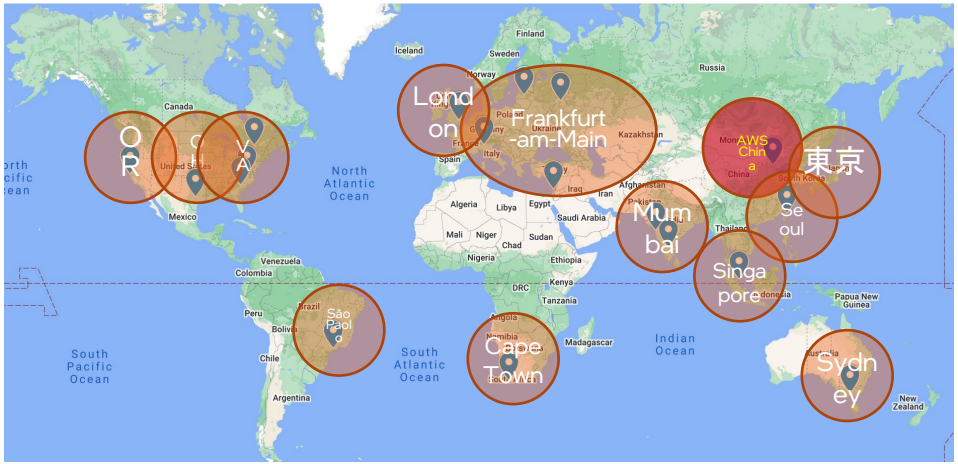
ICRP Community – 250 Organizations Worlwide

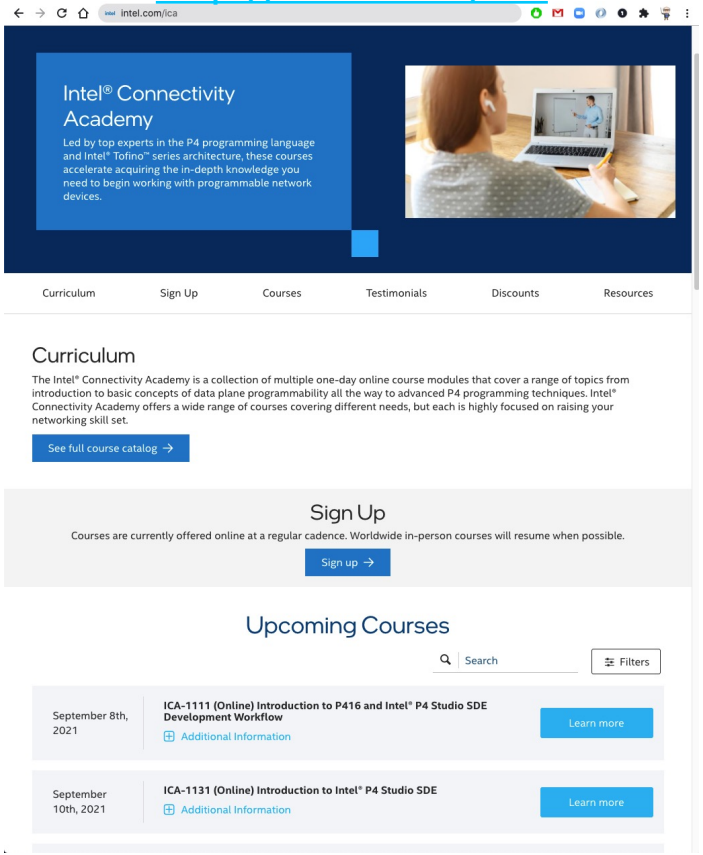


Intel Connectivity Academy (ICA)

http://intel.com/ica

- **A great way to learn about P4 and Tofino**
  - Lectures + Hands-on Labs
    - No Special Setup Required
  - Academic Discount for ICRP Members





intel connectivity academy

Q & A

intel connectivity academy

Thank You!


intel
connectivity
academy