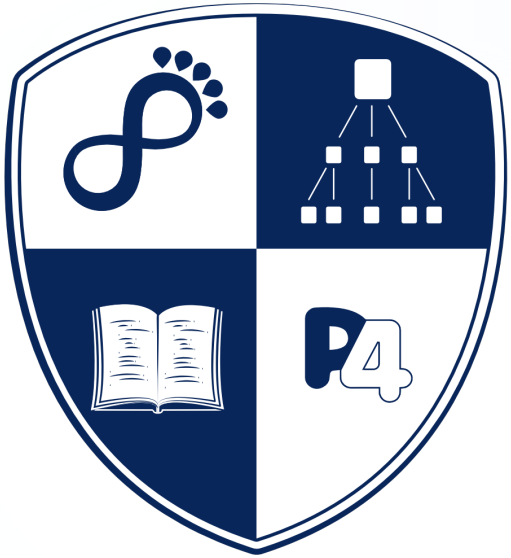
P4$_{16}$ Programming for Intel® Tofino™ using Intel P4 Studio™

Vladimir Gurevich, Principal Engineer, Intel
Andy Fingerhut, Principal Engineer, Intel



intel® Connectivity Academy

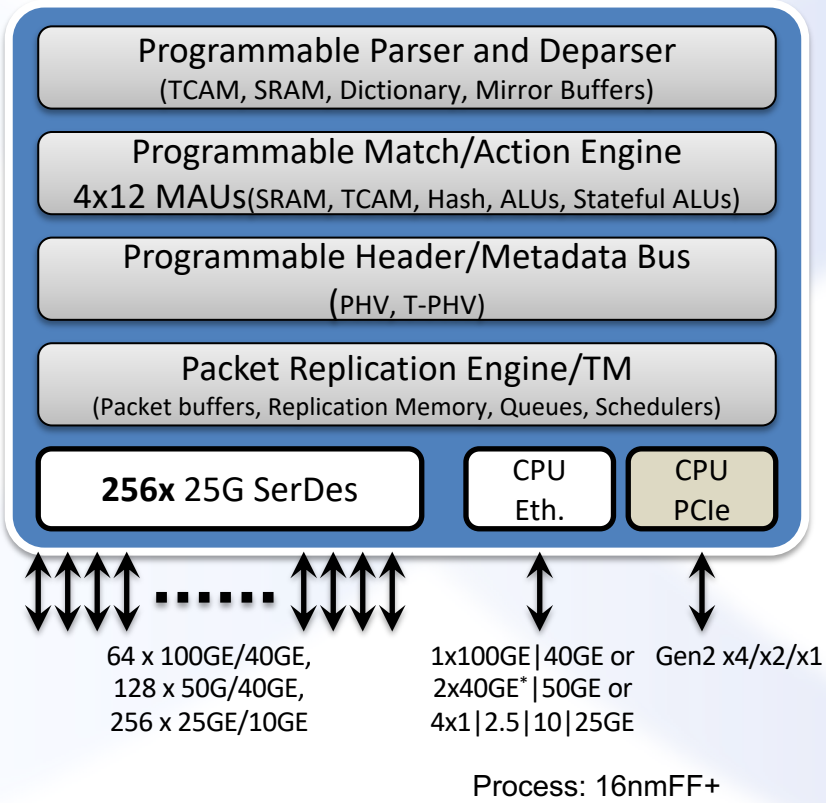
P4$_{16}$ Programming for Intel® Tofino™ using Intel P4 Studio™

Andy Fingerhut, Vladimir Gurevich
Principal Engineers, Intel

May 21, 2021

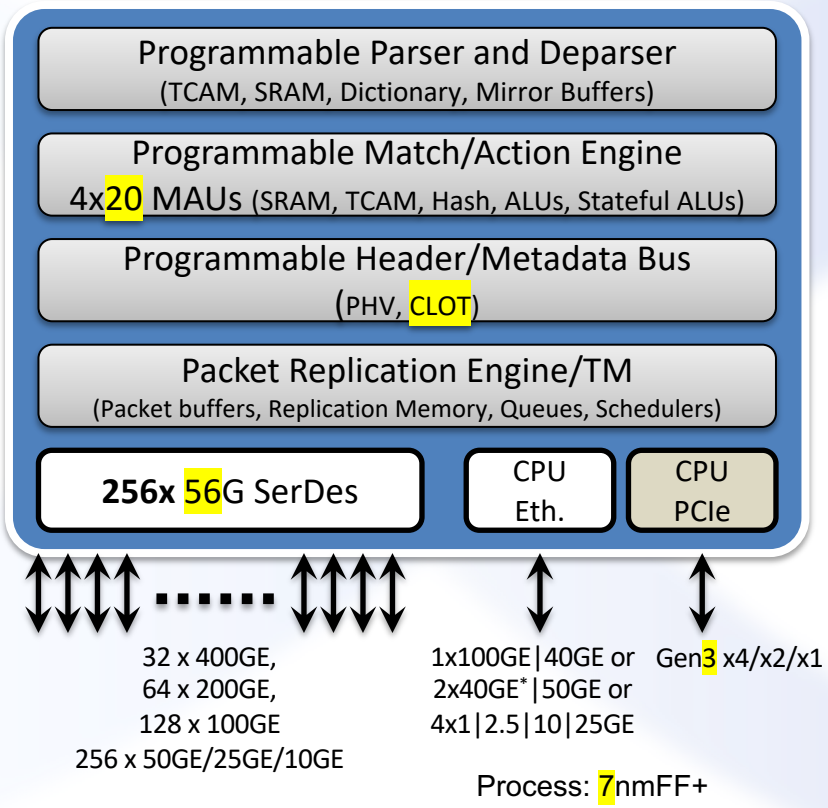
Tofino feature summary

- **6.5Tbps single chip Ethernet switch**
  - Programmable P4 Target
  - All features operate at 6.5Tb/s
- **Port Configurations**
  - 65 x 100GE/40GE
  - 130 x 50GE
  - 260 x 25GE
- **SerDes**
  - 25G with Integrated FEC CL91 & CL74 (10/25/50/100)
  - 25G Ethernet Consortium spec compliant
- **CPU Interfaces**
  - PCIe: Gen2 x4/x2/x1
  - Ethernet: {1x100GE|40GE} | {2x50GE|40GE*} | {4x25GE|10GE|2.5GE|1GE}
- **Single Unified Packet Buffer (22MB)**
- **Customizable Low Latency**
- **1588 & SyncE Support**

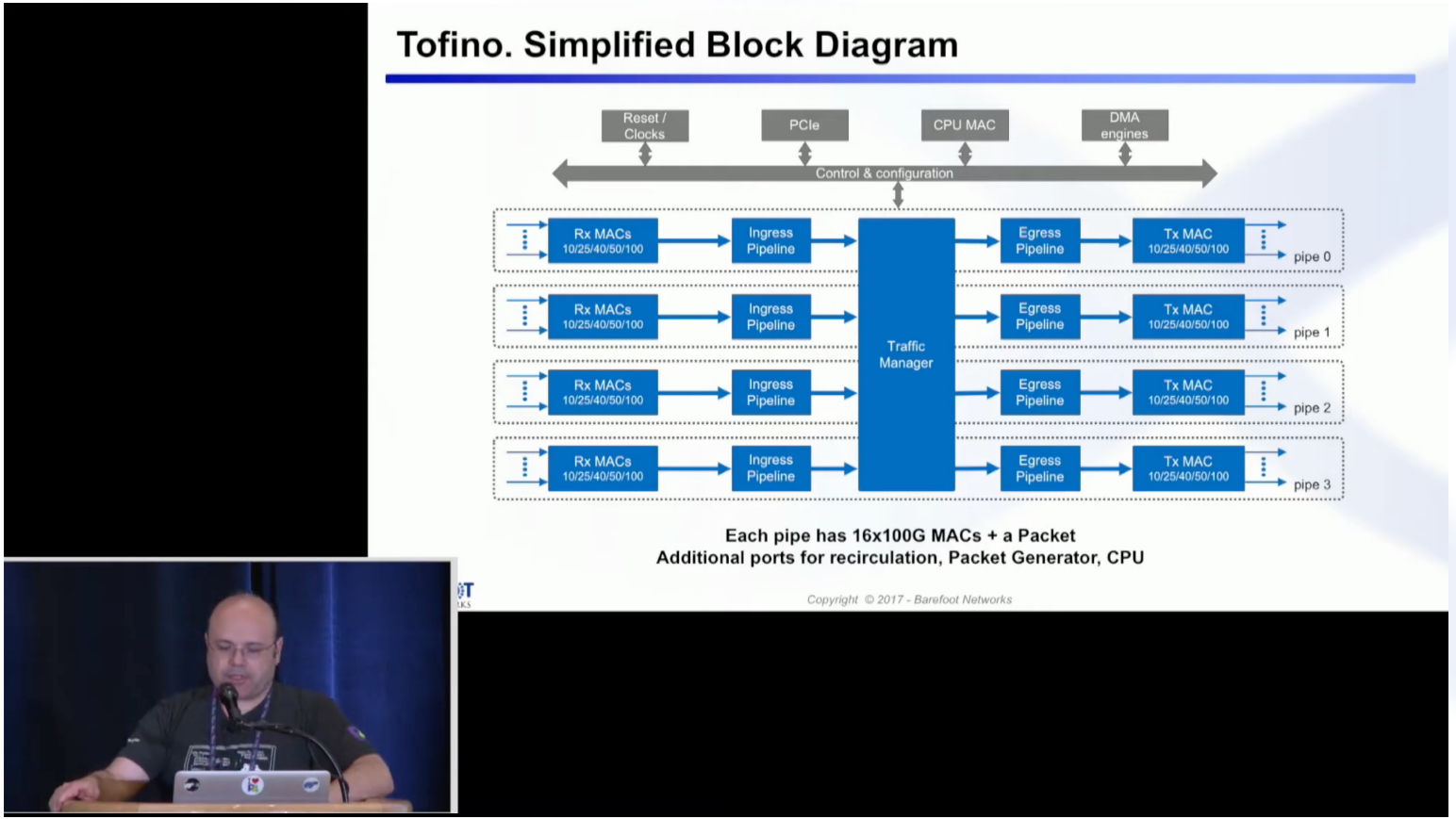


Tofino2 feature summary

- **12.9Tbps single chip Ethernet switch**
  - Programmable P4 Target
  - All features operate at 12.9Tb/s
- **Port Configurations**
  - 32 x 400GE
  - 64 x 200GE
  - 128 x 100GE
  - 256 x 50/25/10GE
- **SerDes**
  - 56G with Integrated FEC CL91 & CL74 (10/25/50/100/200/400)
  - 25G Ethernet Consortium spec compliant
- **CPU Interfaces**
  - PCIe: Gen3 x4/x2/x1
  - Ethernet: {1x100GE|40GE} | {2x50GE|40GE*} | {4x25GE|10GE|2.5GE|1GE}
- **Single Unified Packet Buffer (64MB)**
  - 2-level queuing
- **Customizable Low Latency**
- **1588 & SyncE Support**

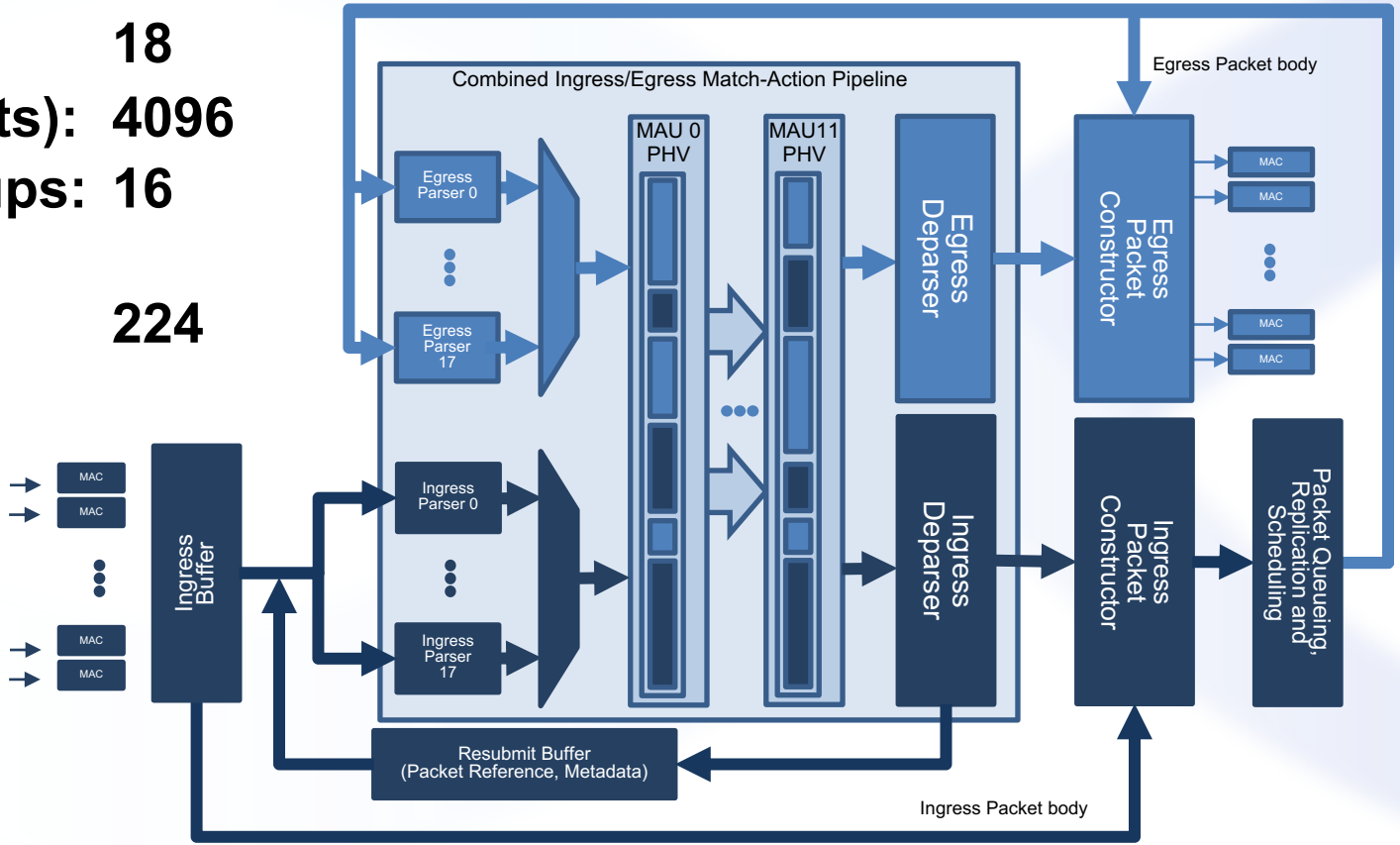


P4 Mapping to Barefoot Tofino

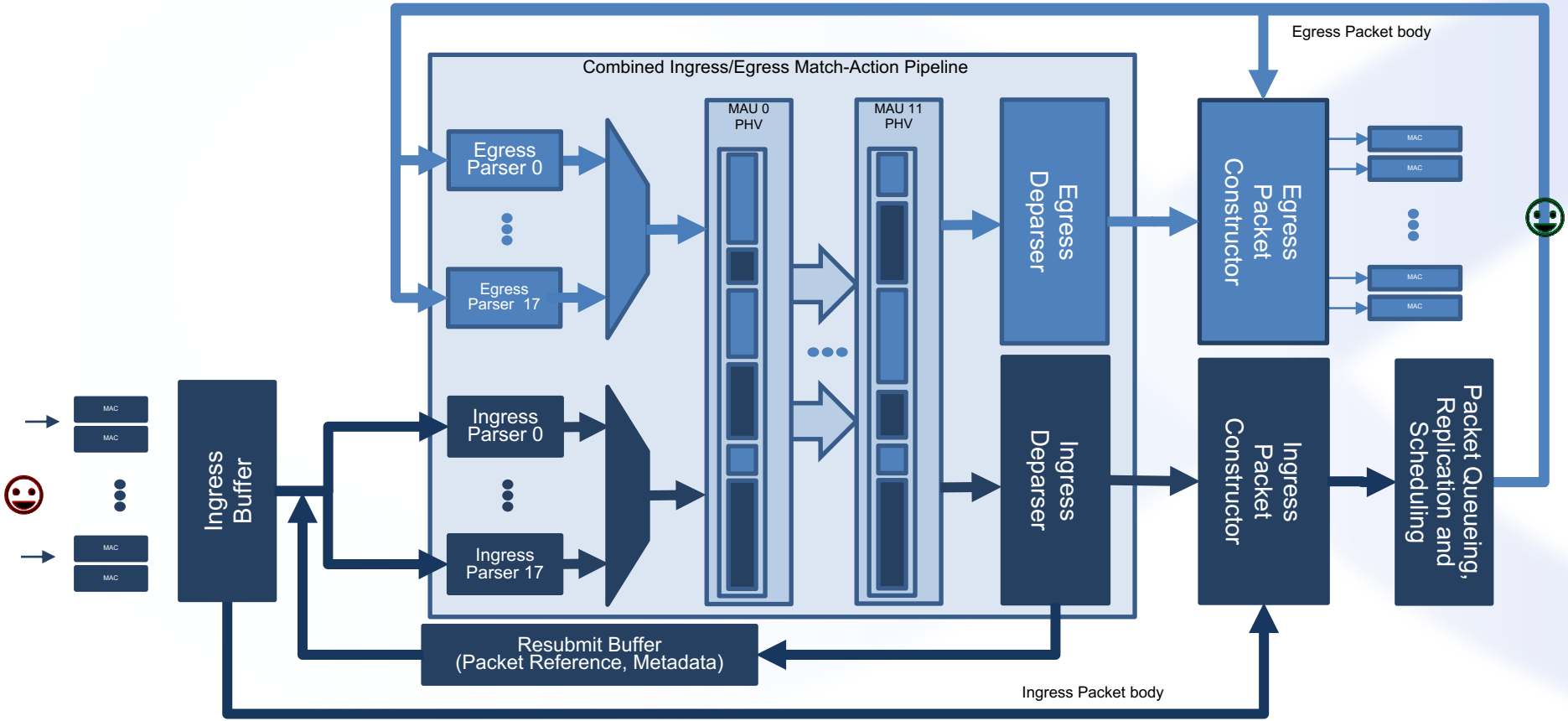


Tofino Processing Pipe Organization

- **Parsers:** 18
- **Bus width (bits):** 4096
- **Parallel lookups:** 16
  - per MAU
- **Parallel ops:** 224
  - per MAU



Pipe Organization: Pipelined packet processing



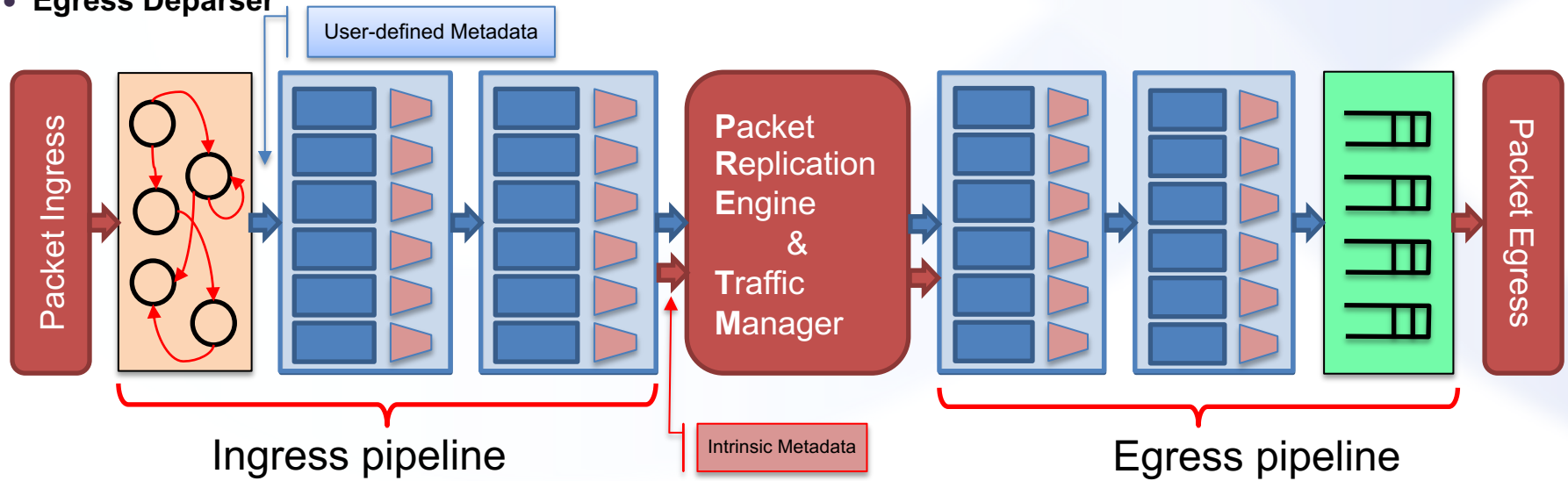
v1model Architecture

Programmable components

- Ingress Parser
- Ingress Checksum Unit
- Ingress Match-Action Pipeline
- Egress Match-Action Pipeline
- Egress Checksum Unit
- Egress Deparser

Configurable (Fixed-Function) Components

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



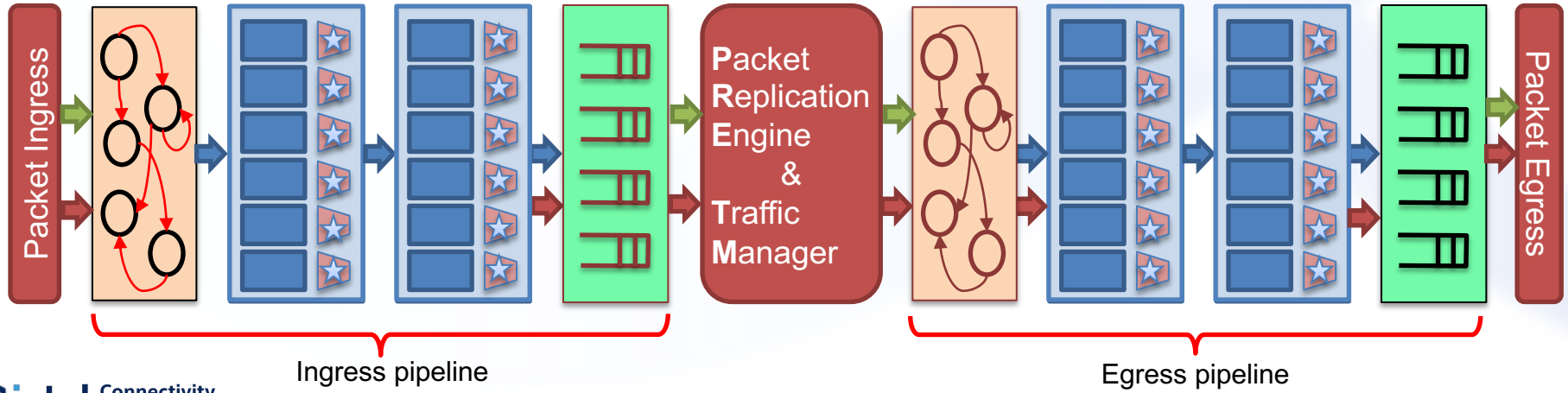
Portable Switch Architecture (PSA)

Programmable components

- **Ingress and Egress Pipeline**
  - Parser
  - Match-Action Pipeline
  - Deparser
- **PSA Intrinsic Metadata**
- **PSA Externs**

Fixed-Function Components

- **Packet I/O (Unspecified)**
- **Packet Replication Engine**
  - PSA-specified features
- **Traffic Manager**



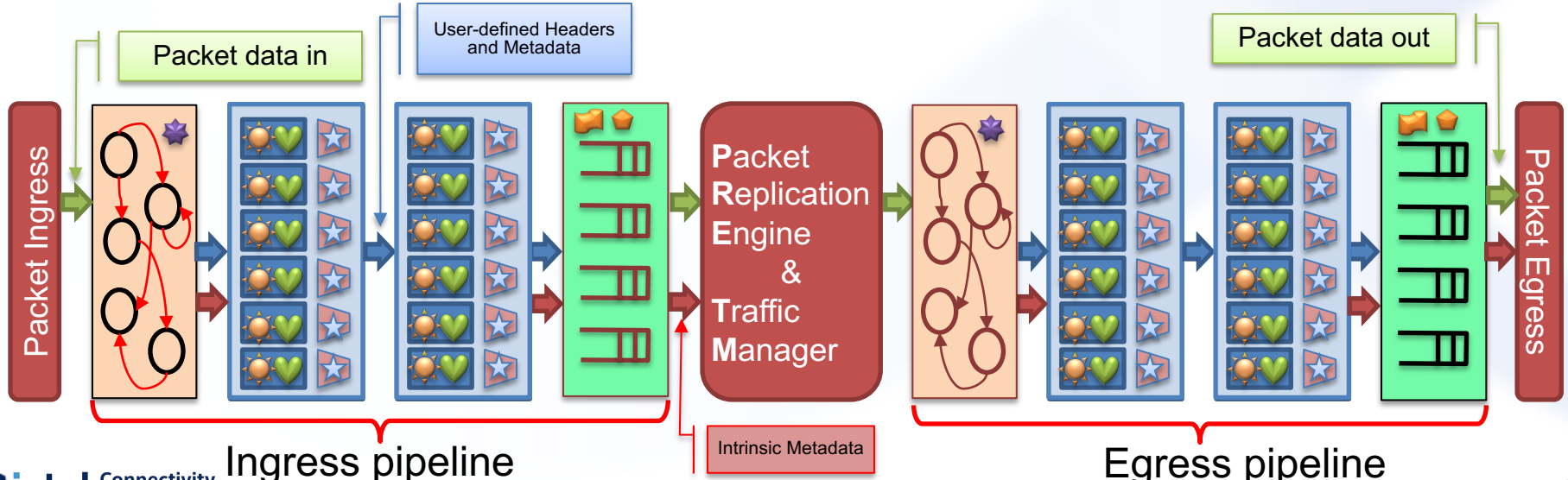
Tofino Native Architecture (TNA)

Programmable components

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

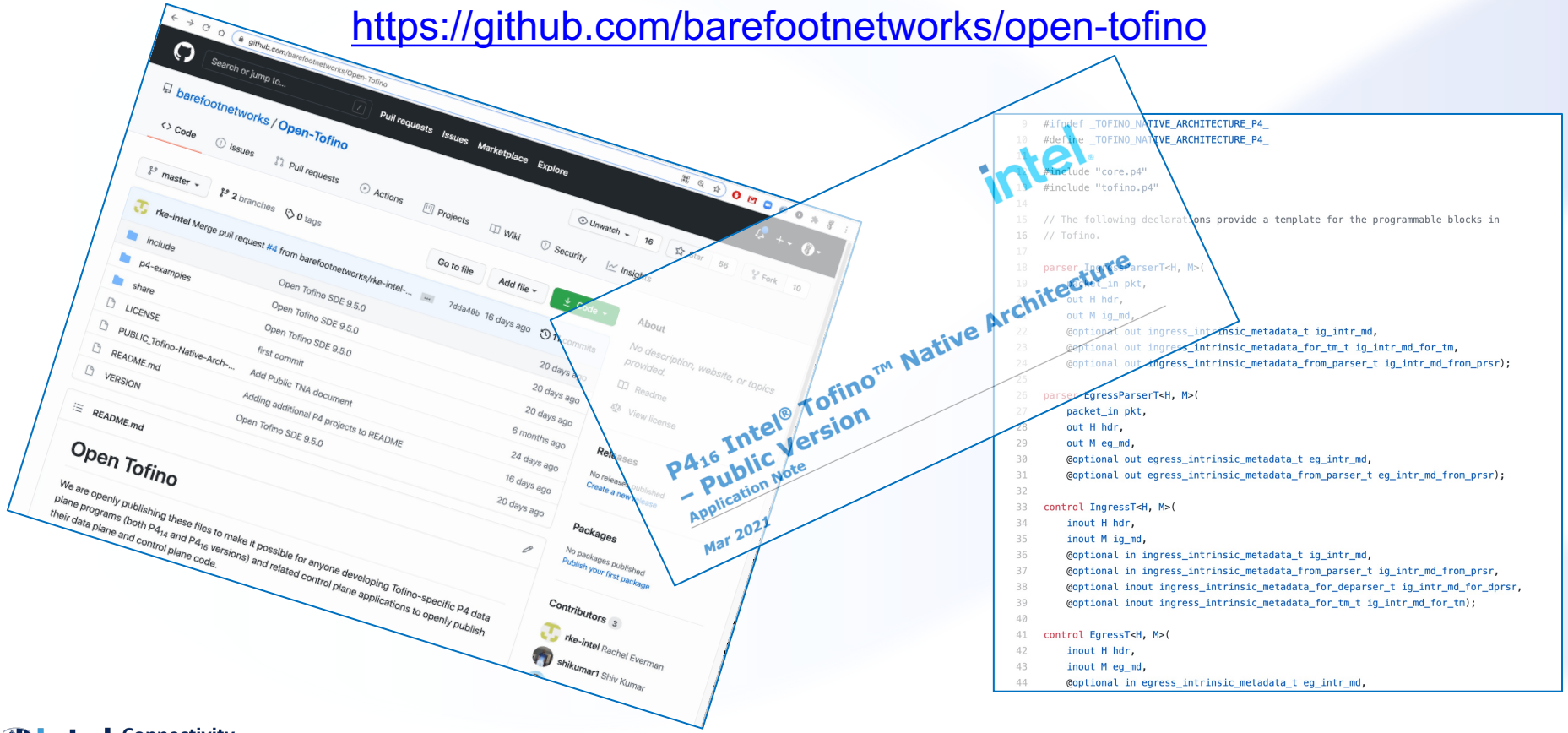
Fixed-Function Components

- **Packet I/O (MAC/SerDes/DMA)**
- **Packet Replication Engine**
  - Extended Tofino features
- **Traffic Manager**



Open-Tofino Repository

https://github.com/barefootnetworks/open-tofino



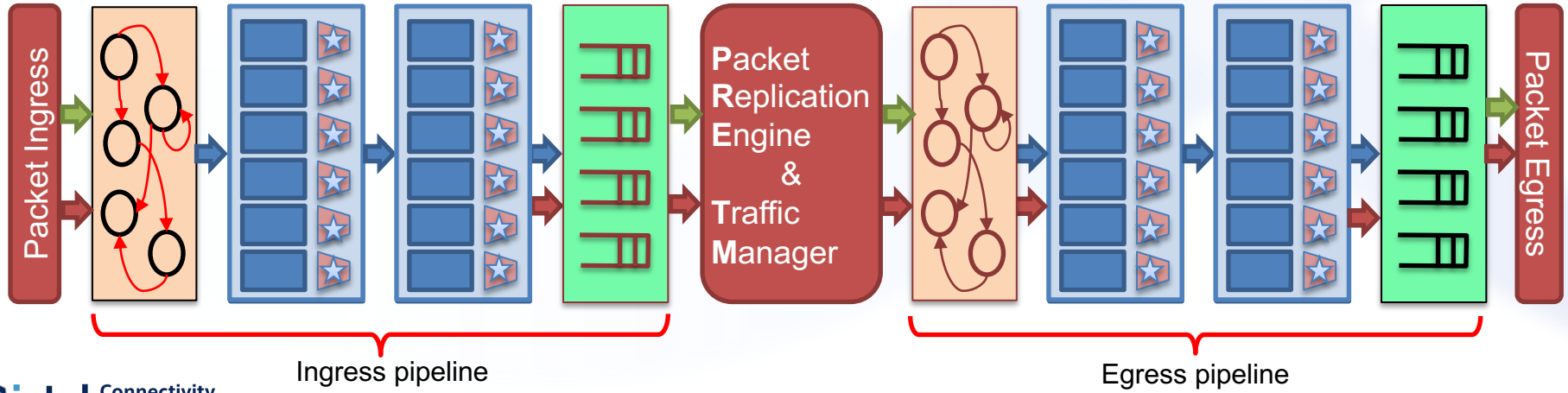
Portable Switch Architecture (PSA)

Programmable components

- **Ingress and Egress Pipeline**
  - Parser
  - Match-Action Pipeline
  - Deparser
- **PSA Intrinsic Metadata**
- **PSA Externs**

Fixed-Function Components

- **Packet I/O (Unspecified)**
- **Packet Replication Engine**
  - PSA-specified features
- **Traffic Manager**



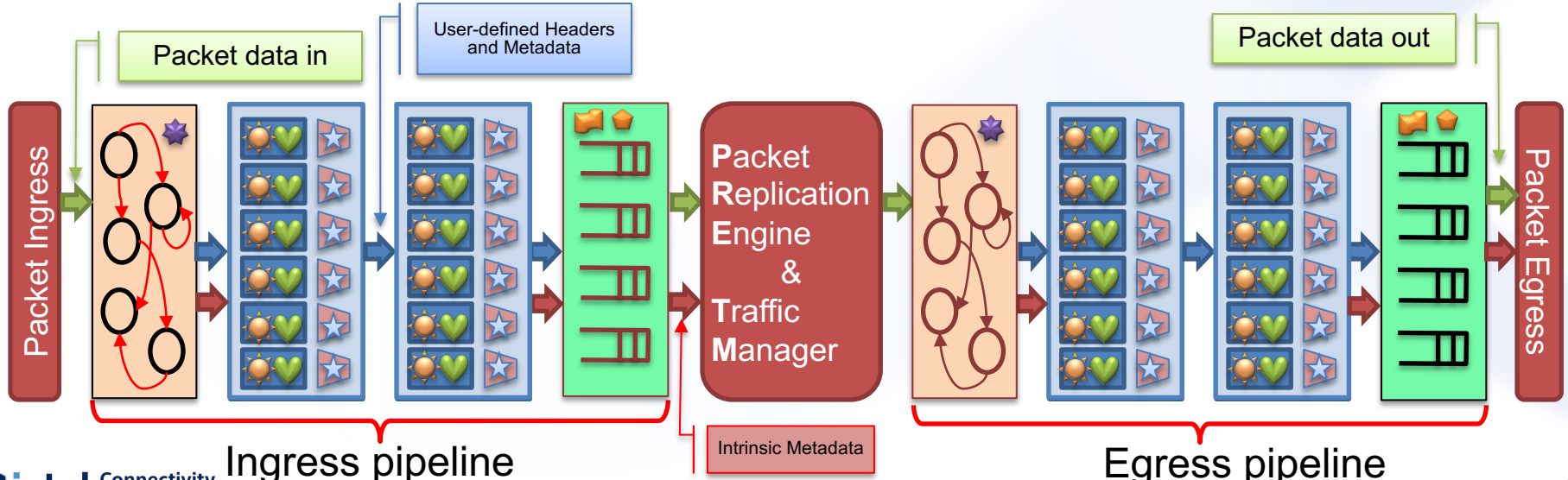
Tofino Native Architecture (TNA)

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**
  - Extended Tofino features
- **Traffic Manager**



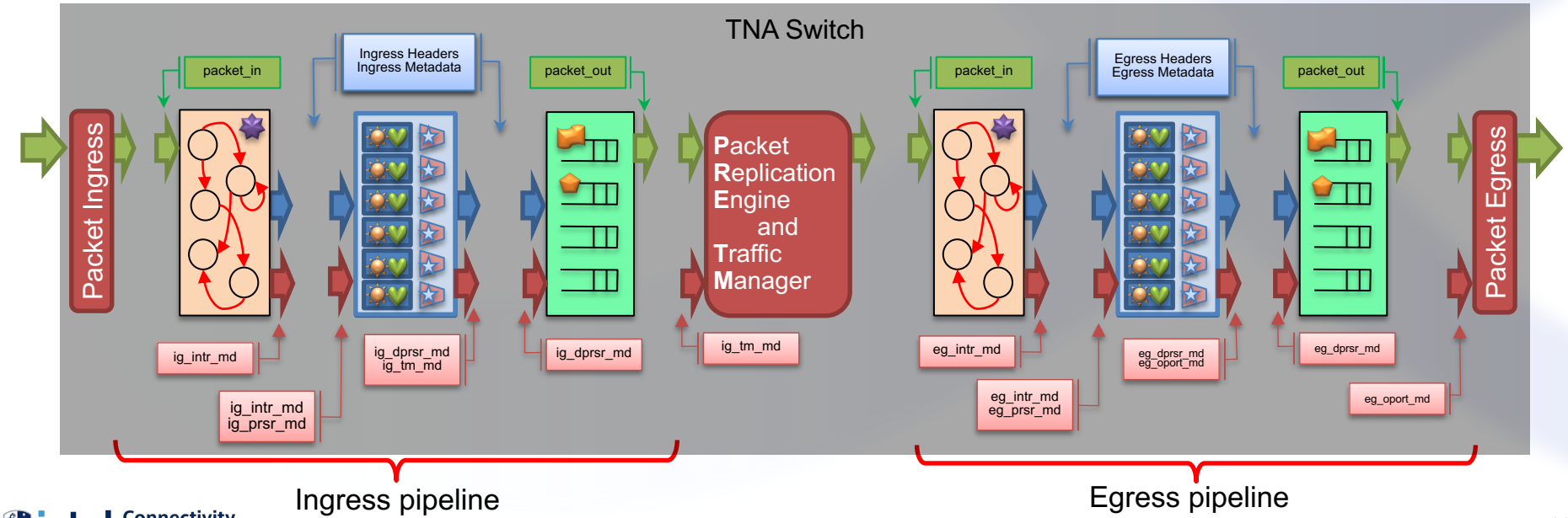
Tofino Native Architecture (TNA). Component View

Programmable components

- **Ingress and Egress Pipeline**
  - Parser
  - Match-Action Pipeline
  - Deparser
  - User-defined Headers and Metadata
- **Tofino-specific Intrinsic Metadata**
- **Tofino-Specific Externs**

Fixed-Function Components

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



P4_16 Program Template for TNA

```
/* -*- P4_16 -*- */

#include <core.p4>
#include <tna.p4>

/*** C O N S T A N T S    A N D    T Y P E S  ****/

/***** H E A D E R   D E F I N I T I O N S  *****/

/******  I N G R E S S   P I P E L I N E ********/
struct my_ingress_headers_t  { }
struct my_ingress_metadata_t { }

parser  MyIngressParser() { }
control MyIngress() { }
control MyIngressDeparser() { }

/********  E G R E S S   P I P E L I N E ********/
struct my_egress_headers_t  { }
struct my_egress_metadata_t { }

parser  MyEgressParser() { }
control MyEgress() { }
control MyEgressDeparser() { }
```

- **Use C Preprocessor**
  - Insert Emacs mode line
  - Include standard headers
  - Split the program into multiple files
- **Write code for all P4-programmable components, allowed by the architecture**
  - *Your* Headers and Metadata
  - *Your* Parsers
  - *Your* (Match-Action) Controls
  - *Your* Deparsers
- **Assemble everything in a package**


```
/******** F I N A L   P A C K A G E ***************/
Pipeline(
    MyIngressParser(), MyIngress(), MyIngressDeparser(),
    MyEgressParser(),  MyEgress(),  MyEgressDeparser()
) pipe;

Switch(pipe) main;
```

Parsers

Parser Prototypes (TNA)

Ingress Parser

```
parser IngressParser(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 {
        /* TNA-specific Code for simple cases */
        pkt.extract(ig_intr_md);
        pkt.advance(PORT_METADATA_SIZE);
        transition parse_ethernet;
    }
}
```

Egress Parser

```
parser EgressParser(packet_in          pkt,
    out my_egress_headers_t            hdr,
    out my_egress_metadata_t           meta,
    out egress_intrinsic_metadata_t    eg_intr_md)
{
    state start {
        pkt.extract(eg_intr_md);
        transition parse_ethernet;
    }
}
```

Important Constructs

packet_in methods

```
pkt.extract(hdr.ipv4)
pkt.extract(hdr.ipv4_options,
            (bit<32>)((hdr.ipv4.ihl - 5) * 32))
pkt.advance(const_number_of_bits)
l4_lookup = pkt.lookahead<l4_lookup_t>()
```

States and statements

```
state parse_ethernet {
    pkt.extract(hdr.ethernet);
    meta.src_mac = hdr.ethernet.src_addr;
    transition select(hdr.ethernet.ether_type) {
        0x0000 &&& 0x0C00 : parse_llc;
        0x0400 &&& 0x0E00 : parse_llc;
        0x8100            : parse_vlan;
        0x0800            : parse_ipv4;
        default           : accept;
     }
}

state parser_drop {
    transition select(<any header field>) {
       /* Empty select() will result in a TCAM miss */
    }
}
```

Controls

Match-Action Control Prototypes (TNA)

Ingress Control

```
control Ingress(
    /* User */
    inout my_ingress_headers_t                      hdr,
    inout my_ingress_metadata_t                     meta,
    /* Intrinsic */
    in    ingress_intrinsic_metadata_t              ig_intr_md,
    in    ingress_intrinsic_metadata_from_parser_t  ig_prsr_md,
    inout ingress_intrinsic_metadata_for_deparser_t ig_dprsr_md,
    inout ingress_intrinsic_metadata_for_tm_t       ig_tm_md)
{
    /* Resources= definitions */
    apply {
       /* The algorithm */
    }
}
```

Egress Control

```
control Egress(
    inout my_egress_headers_t                       hdr,
    inout my_egress_metadata_t                      meta,
    in    egress_intrinsic_metadata_t               eg_intr_md,
    in    egress_intrinsic_metadata_from_parser_t   eg_prsr_md,
    inout egress_intrinsic_metadata_for_deparser_t  eg_dprsr_md,
    inout egress_intrinsic_metadata_for_output_port_t eg_oport_md)
{
    ...
}
```

Deparser Control Prototypes (TNA)

Ingress Deparser

```
control MyIngressDeparser(packet_out pkt,
    /* User */
    inout my_ingress_headers_t                      hdr,
    in    my_ingress_metadata_t                     meta,
    /* Intrinsic */
    in ingress_intrinsic_metadata_for_deparser_t    ig_dprsr_md)
{
    /* Resource Definitions */

    apply {
        /* The algorithm */
        pkt.emit(hdr);
    }
}
```

Egress Deparser

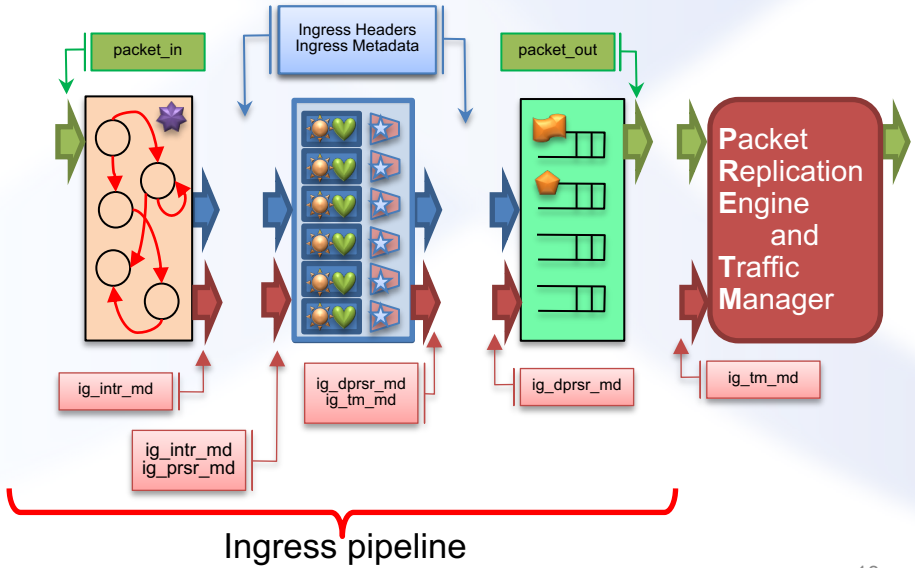
```
control MyEgressDeparser(packet_out pkt,
    inout my_egress_headers_t                       hdr,
    in    my_egress_metadata_t                      meta,
    in    egress_intrinsic_metadata_for_deparser_t  eg_dprsr_md)
{
    /* Resource Definitions */

    apply {
        pkt.emit(hdr); }
    }
}
```

Intrinsic Metadata (Ingress Pipeline Inputs)

```
typedef bit<9>  PortId_t;

header ingress_intrinsic_metadata_t {
    bit<1>     resubmit_flag;       // 0 for normal packets.
    bit<6>     _pad1;
    PortId_t   ingress_port;        // Ingress physical port id
    bit<48>    ingress_mac_tstamp;  // IEEE 1588 timestamp (nsec)
}

struct ingress_intrinsic_metadata_from_parser_t {
    bit<48> global_tstamp;          // Parser entry time (nsec)

    bit<16> parser_err;
}
```

```
/* Tofino Parser Error Codes */
const bit<16> PARSER_ERROR_OK             = 0x0000;
const bit<16> PARSER_ERROR_NO_TCAM        = 0x0001;
const bit<16> PARSER_ERROR_PARTIAL_HDR    = 0x0002;
const bit<16> PARSER_ERROR_CTR_RANGE      = 0x0004;
const bit<16> PARSER_ERROR_TIMEOUT_USER   = 0x0008;
const bit<16> PARSER_ERROR_TIMEOUT_HW     = 0x0010;
const bit<16> PARSER_ERROR_SRC_EXT        = 0x0020;
const bit<16> PARSER_ERROR_DST_CONT       = 0x0040;
const bit<16> PARSER_ERROR_PHV_OWNER      = 0x0080;
const bit<16> PARSER_ERROR_MULTIWRITE     = 0x0100;
const bit<16> PARSER_ERROR_ARAM_SBE       = 0x0200;
const bit<16> PARSER_ERROR_ARAM_MBE       = 0x0400;
const bit<16> PARSER_ERROR_FCS            = 0x0800;
```

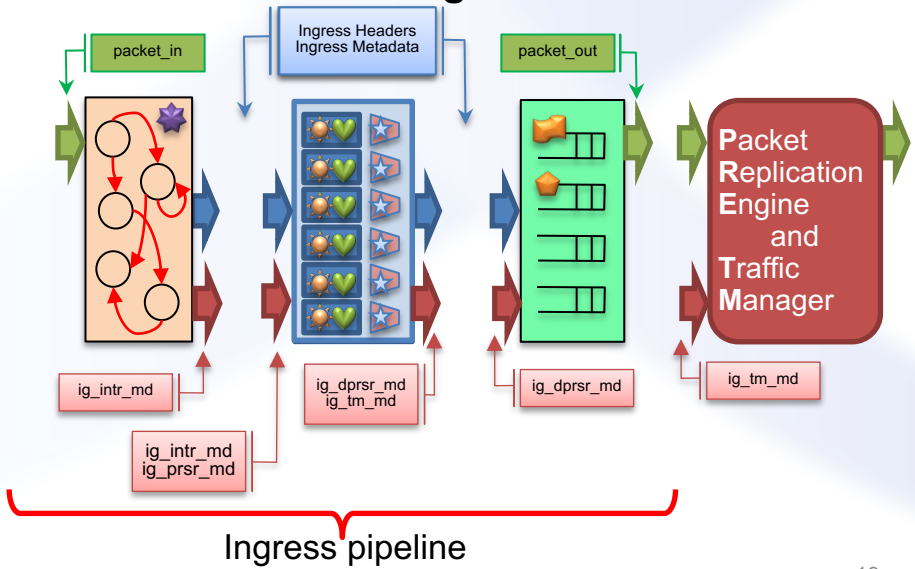
- **Nanosecond-precision timestamps**
- **Detailed parser error information**
  - Allows safe parsing of short packets



Intrinsic Metadata (Ingress Pipeline Outputs)

```
typedef bit<16> MulticastGroupId_t;
typedef bit<5>  QueueId_t;
typedef bit<16> ReplicationId_t;

struct ingress_intrinsic_metadata_for_deparser_t {
    bit<3> drop_ctl;
    bit<3> digest_type;
    bit<3> resubmit_type;
    bit<3> mirror_type;
}

struct ingress_intrinsic_metadata_for_tm_t {
    PortId_t            ucast_egress_port;
    bit<1>              bypass_egress;
    bit<1>              deflect_on_drop;
    bit<3>              ingress_cos;
    QueueId_t           qid;
    bit<3>              icos_for_copy_to_cpu;
    bit<1>              copy_to_cpu;
    bit<2>              packet_color;
    bit<1>              disable_ucast_cutthru;
    bit<1>              enable_mcast_cutthru;
    MulticastGroupId_t  mcast_grp_a;
    MulticastGroupId_t  mcast_grp_b;
    bit<13>             level1_mcast_hash;
    bit<13>             level2_mcast_hash;
    bit<16>             level1_exclusion_id;
    bit<9>              level2_exclusion_id
    ReplicationId_t     rid;
}
```

- **Up to 4 independent destination types**
  - Including 2 independent multicast groups
- **Low-latency support**
  - Egress bypass
  - Cut-Through
- **Queueing control**
- **Advanced Multicasting**



Intrinsic Metadata (Ingress Pipeline Outputs)

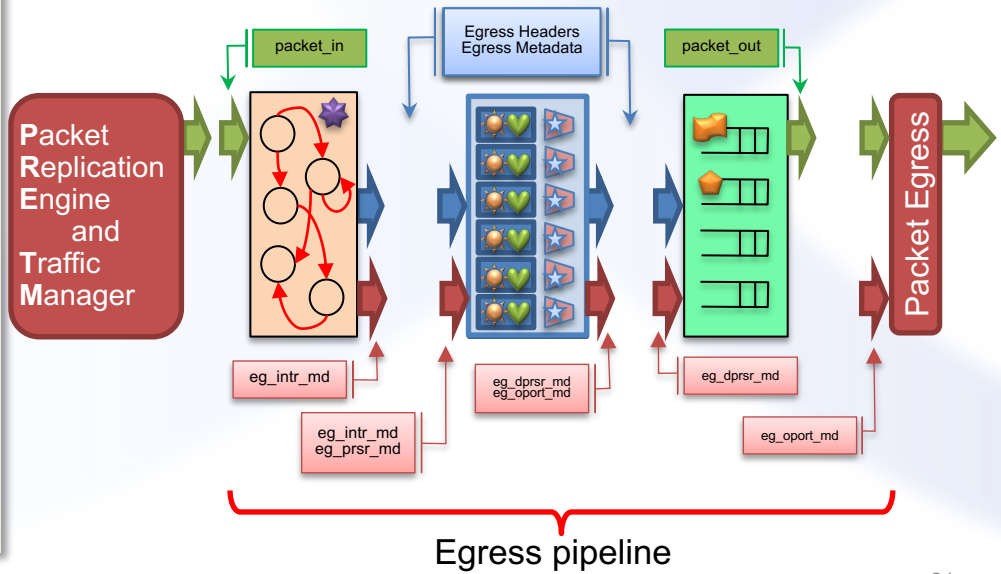
```
typedef bit<16> MulticastGroupId_t;
typedef bit<5>  QueueId_t;
typedef bit<16> ReplicationId_t;

struct ingress_intrinsic_metadata_for_deparser_t {
    bit<3> drop_ctl;
    bit<3> digest_type;
    bit<3> resubmit_type;
    bit<3> mirror_type;
}

struct ingress_intrinsic_metadata_for_tm_t {
    PortId_t            ucast_egress_port;
    bit<1>              bypass_egress;
    bit<1>              deflect_on_drop;
    bit<3>              ingress_cos;
    QueueId_t           qid;
    bit<3>              icos_for_copy_to_cpu;
    bit<1>              copy_to_cpu;
    bit<2>              packet_color;
    bit<1>              disable_ucast_cutthru;
    bit<1>              enable_mcast_cutthru;
    MulticastGroupId_t  mcast_grp_a;
    MulticastGroupId_t  mcast_grp_b;
    bit<13>             level1_mcast_hash;
    bit<13>             level2_mcast_hash;
    bit<16>             level1_exclusion_id;
    bit<9>              level2_exclusion_id
    ReplicationId_t     rid;
}
```

```
action quick_send(PortId_t port, bit<1> bypass_egress) {
    ig_tm_md.ucast_egress_port = port;
    ig_tm_md.qid               = (QueueId_t)meta.pcp;
    ig_tm_md.bypass_egress     = bypass_egress;
}

action igmp_snoop(MulticastGroupId_t mcg) {
    ig_tm_md.mcast_grp_a          = mcg;
    ig_tm_md.copy_to_cpu          = 1;
    ig_tm_md.enable_mcast_cutthru = 1;
}

action fast_drop() {
    ig_dprsr_md.drop_ctl = 3; /* Normal + Copy-to-CPU */
    exit;
}

action resurrect() {
    ig_dprsr_md.drop_ctl = 0;
}
```

Intrinsic Metadata (Egress Pipeline Inputs)

```
header egress_intrinsic_metadata_t {
    bit<7>  _pad0;  PortId_t         egress_port;
    bit<5>  _pad1   bit<19>          enq_qdepth;  /* in cells */
    bit<6>  _pad2;  bit<2>           enq_congest_stat;
    bit<14> _pad3;  bit<18>          enq_tstamp;
    bit<5>  _pad4;  bit<19>          deq_qdepth;
    bit<6>  _pad5;  bit<2>           deq_congest_stat;
                    bit<8>           app_pool_congest_stat;
                    ReplicationId_t  egress_rid;
    bit<7>  _pad7;  bit<1>           egress_rid_first;
    bit<3>  _pad8;  QueueId_t        egress_qid;
    bit<5>  _pad9;  bit<3>           egress_cos;
    bit<7>  _pad10; bit<1>           deflection_flag;
                    bit<16>          pkt_length;
}

struct egress_intrinsic_metadata_from_parser_t {
    bit<48> global_tstamp;
    bit<32> global_ver;
    bit<16> parser_err;
}
```

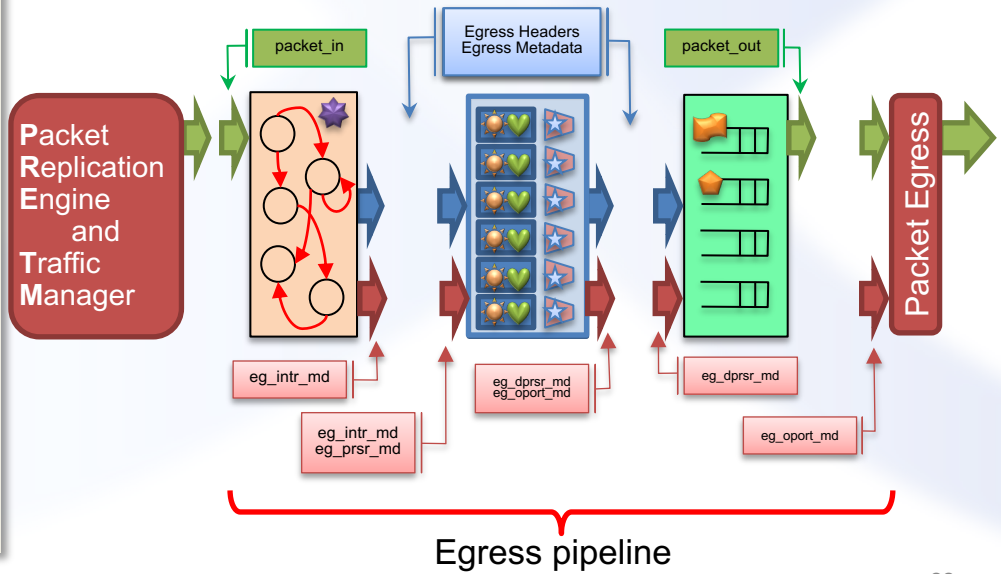
- **Extensive INT support**



Intrinsic Metadata (Egress Pipeline Outputs)

```
struct egress_intrinsic_metadata_for_deparser_t {
    bit<3>   drop_ctl;
    bit<3>   mirror_type;
}

struct egress_intrinsic_metadata_for_output_port_t {
    bit<1>   capture_tstamp_on_tx;
    bit<1>   update_delay_on_tx;
    bit<1>   force_tx_error;
}
```

- **1588 Support**
  - 1-Step
  - 2-Step



Cool Tofino Capabilities

- Custom Polynomials
- Approximate Algebra and Calculus on Tofino
- Packet Generators

Custom Hashing

P4$_{16}$/TNA Definitions

Standard Hash Algorithms

```
enum HashAlgorithm_t {
    IDENTITY,
    RANDOM,  // Hash Matrix will be randomly populated
    CRC8,
    CRC16,
    CRC32,
    CRC64,
    CUSTOM  // Algorithm is specified via CRCPolynomial extern
}
```

CRCPolynomial Extern for Custom Hashes

```
extern CRCPolynomial<T> {
    CRCPolynomial(T     coeff,
                  bool  reversed,
                  bool  msb,
                  bool  extended,
                  T     init,
                  T     xor);
}
```

Hash Extern

```
extern Hash<W> {
    Hash(HashAlgorithm_t algo);
    Hash(HashAlgorithm_t algo, CRCPolynomial<_> poly);

    W get<D>(in D data); // Compute W bits of has from data
}
```

Usage

Standard Hash Calculation

```
Hash<bit<32>>(HashAlgorithm_t.CRC32) crc32;
bit<32> result;
. . .
result = crc32.get({field_1, field_2, … , field_N});
```

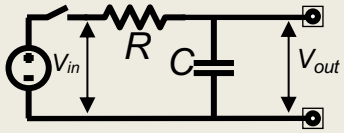
Hash with Custom Polynomial

```
CRCPolynomial<bit<32>>(
    coeff    = 0x0EDC6F41,
    reversed = true,
    msb      = false,
    extended = false,
    init     = 0xFFFFFFFF,
    xor      = 0xFFFFFFFF) poly;
Hash<bit<32>>(HashAlgorithm_t.CUSTOM, poly) crc32c;

result = crc32c.get({
            hdr.ipv4.src_addr, hdr.ipv4.dst_addr,
            hdr.ipv4.protocol,
            meta.l4_sport, meta.l4_dport });
```

Universal Smoother (simple_lpf) program

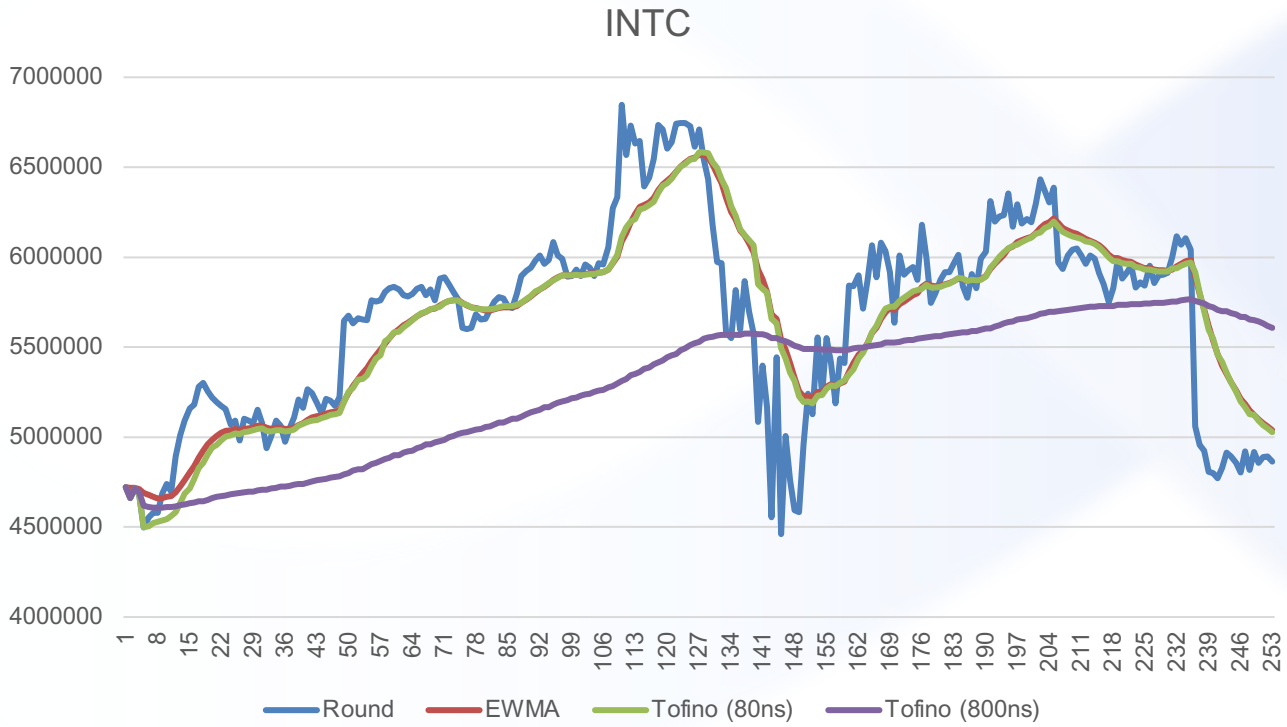
```
enum bit<16> ether_type_t {
    LPF = 0xD011
}

typedef bit<32> value_t;

header lpf_h {
    value_t  input;     /* LPF Input */
    value_t  output_1;  /* Output from the first  filter */
    value_t  output_2;  /* Output from the second filter */
    value_t  output_3;  /* Output from the third  filter */
}

parser IngressParser(. . .) {
   . . .
   state parse_ethernet {
        pkt.extract(hdr.ethernet);
        transition select(hdr.ethernet.ether_type) {
            ether_type_t.LPF : parse_lpf;
        }
    }

    state parse_lpf {
        pkt.extract(hdr.lpf);
        transition accept;
    }
}
```

```
control Ingress(...)
{
    action send_back() {
        ig_tm_md.ucast_egress_port = ig_intr_md.ingress_port;
        ig_tm_md.bypass_egress     = 1;
    }

    Lpf<value_t, index_t>(1) lpf_1;
    Lpf<value_t, index_t>(1) lpf_2;
    Lpf<value_t, index_t>(1) lpf_3;

    apply {
        hdr.lpf.output_1 = lpf_1.execute(hdr.lpf.input, 0);
        hdr.lpf.output_2 = lpf_2.execute(hdr.lpf.output_1, 0);
        hdr.lpf.output_3 = lpf_3.execute(hdr.lpf.output_2, 0);
        send_back();
    }
}
```



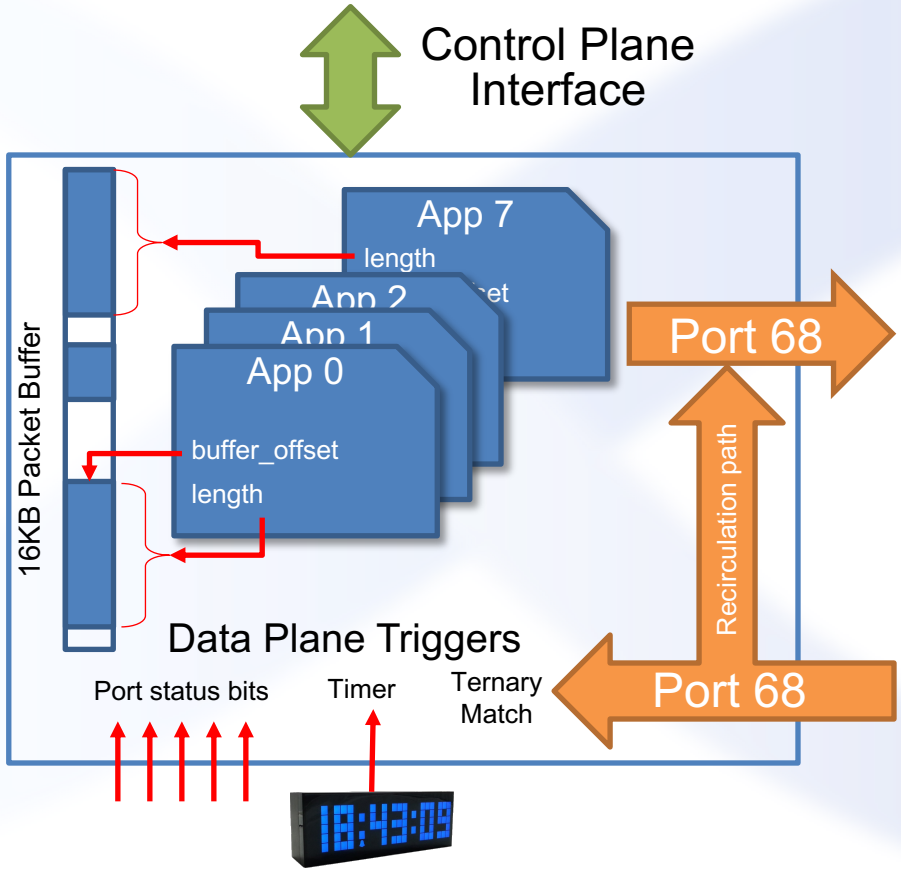
LPF_INDEX	LPF_Type	$\tau_{gain}(ns)$	$\tau_{decay}(ns)$	$Scale_{down}$
0	SAMPLE	≈160	≈160	0

Smoothing Stock Prices



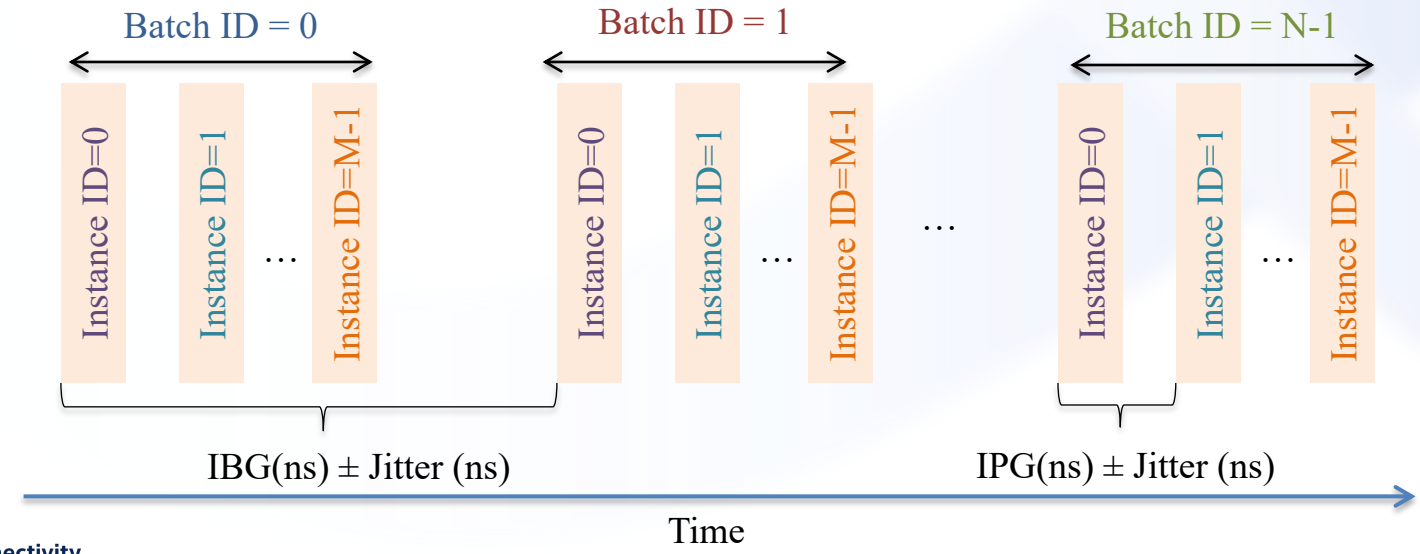
Packet Generators

- **Pipeline Interface: Pipe Port 68**
- **8 Independent Generators (Apps)**
- **16KB Packet Data Buffer**
  - Each App can have its own packet space (allocated by Control Plane)
- **Data Plane Triggers**
  - Each App has its own trigger
    - Timer
      - Periodic
      - One-shot timer
    - Port Down
    - First 4 bytes of a recirculated packet



Events, Batches and Packets

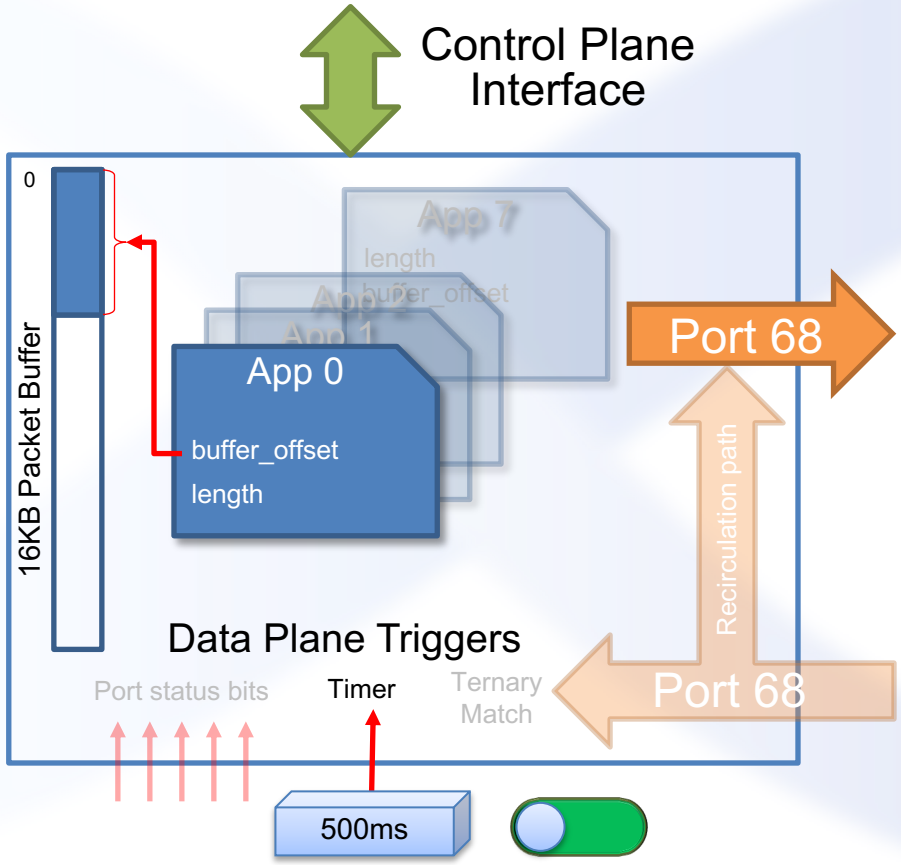
- **At each triggered event an app produces N batches of M Packets**
  - $1 \leq N \leq 65536, 1 \leq M \leq 65536$
- **Inter-Batch and Inter-Packet Gap are programmable**
  - With optional random jitter
- **Batch and Packet Instance IDs are passed within the packet**



Continuous Packet Stream with Periodic Timer

- **App 0**
  - Packet
    - Simple IPv4/UDP packet
    - buffer_offset = 0, length = …
  - Trigger
    - Periodic Timer
    - Period: 0.5s

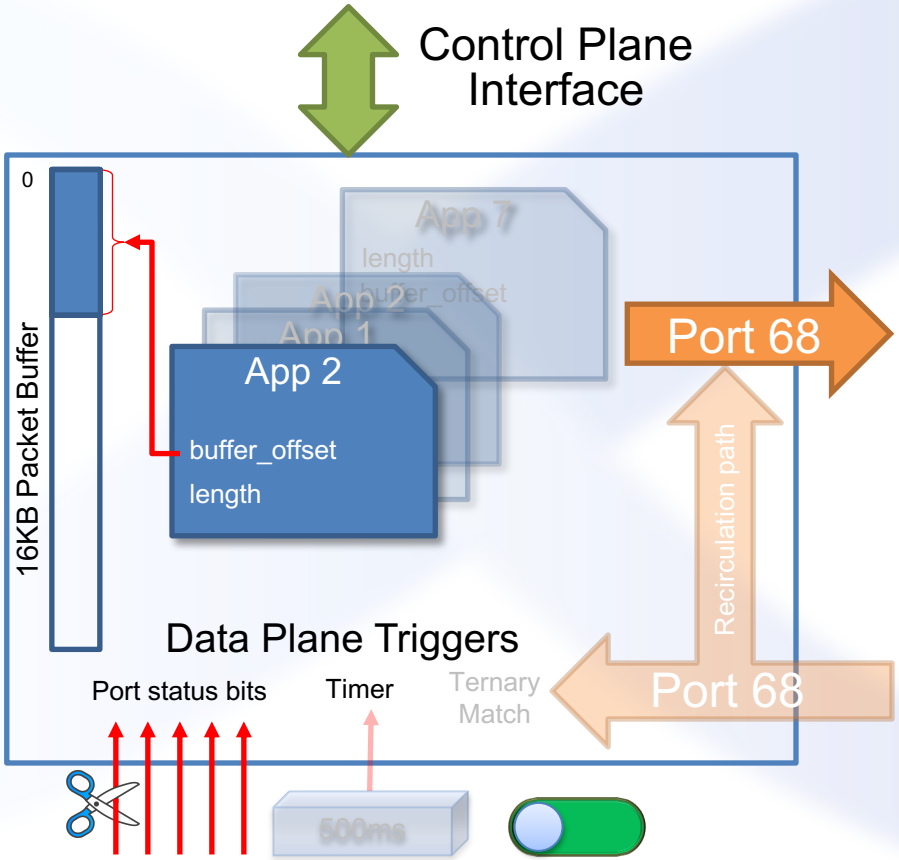
pktgen_timer_header_t



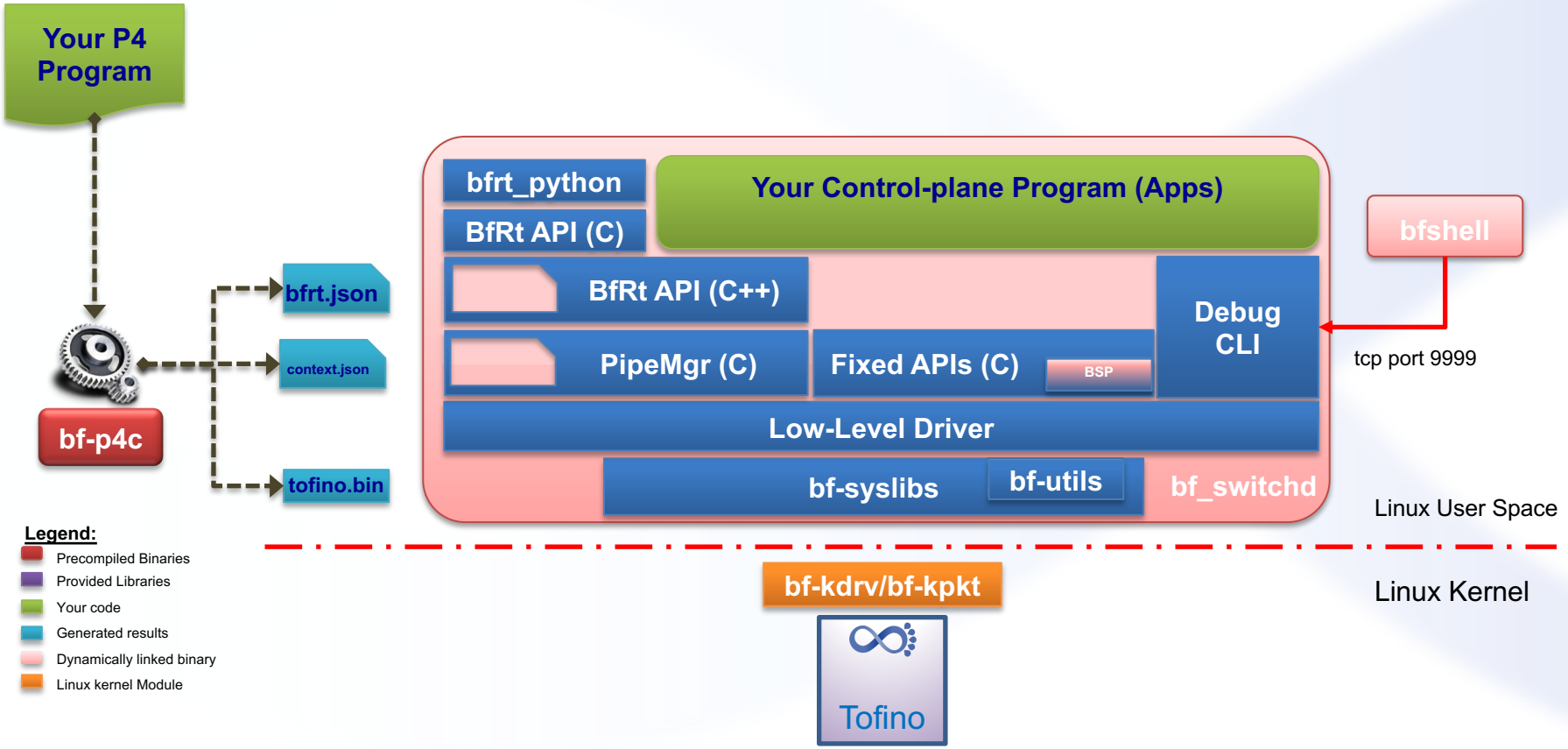
Fast Failover with Port Down Trigger

- **App 2**
  - Packet
    - Simple IPv4/UDP packet
    - buffer_offset = 0, length = …
  - Trigger
    - Port Link Down
  - 1 Batch, 4 Packets

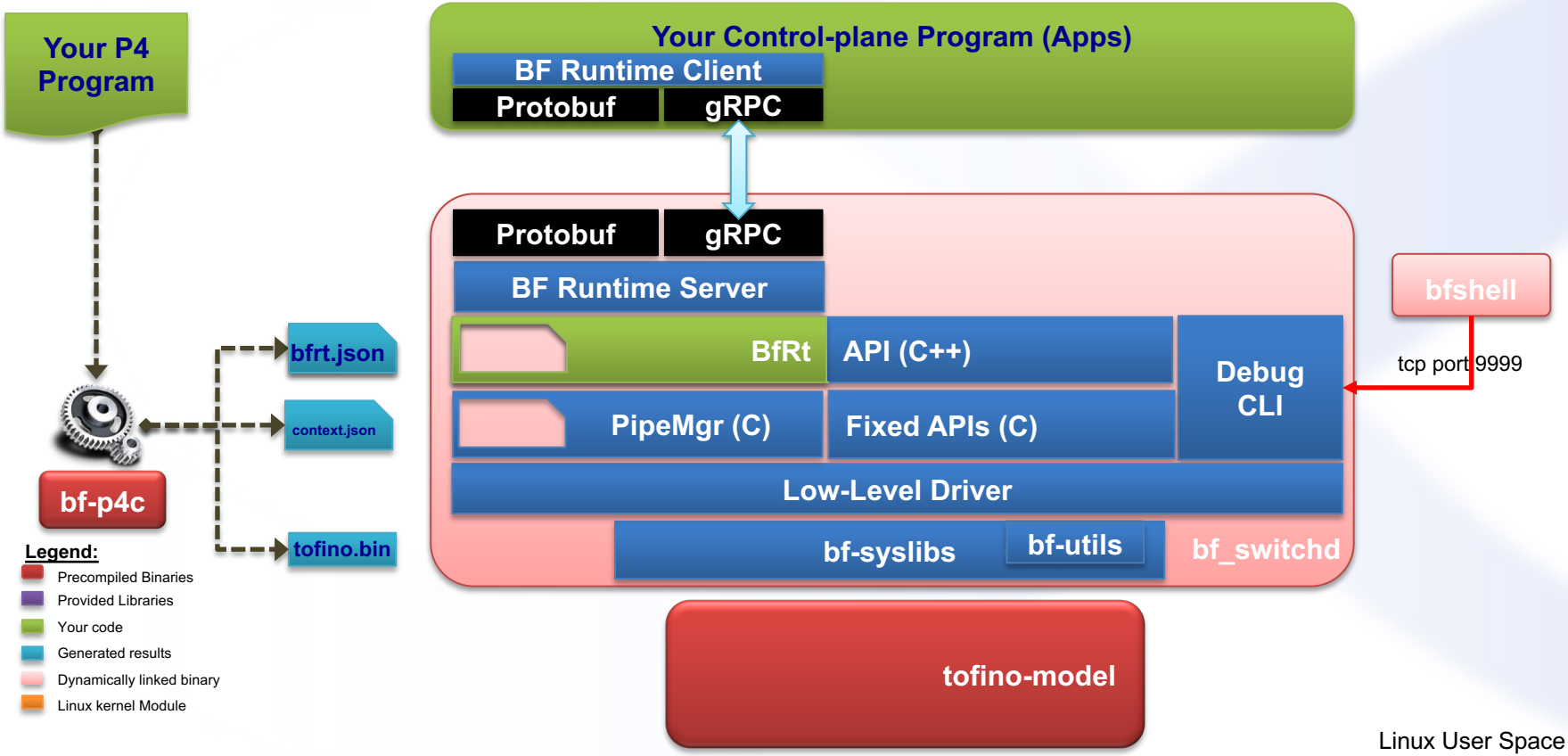
pktgen_port_down_header_t



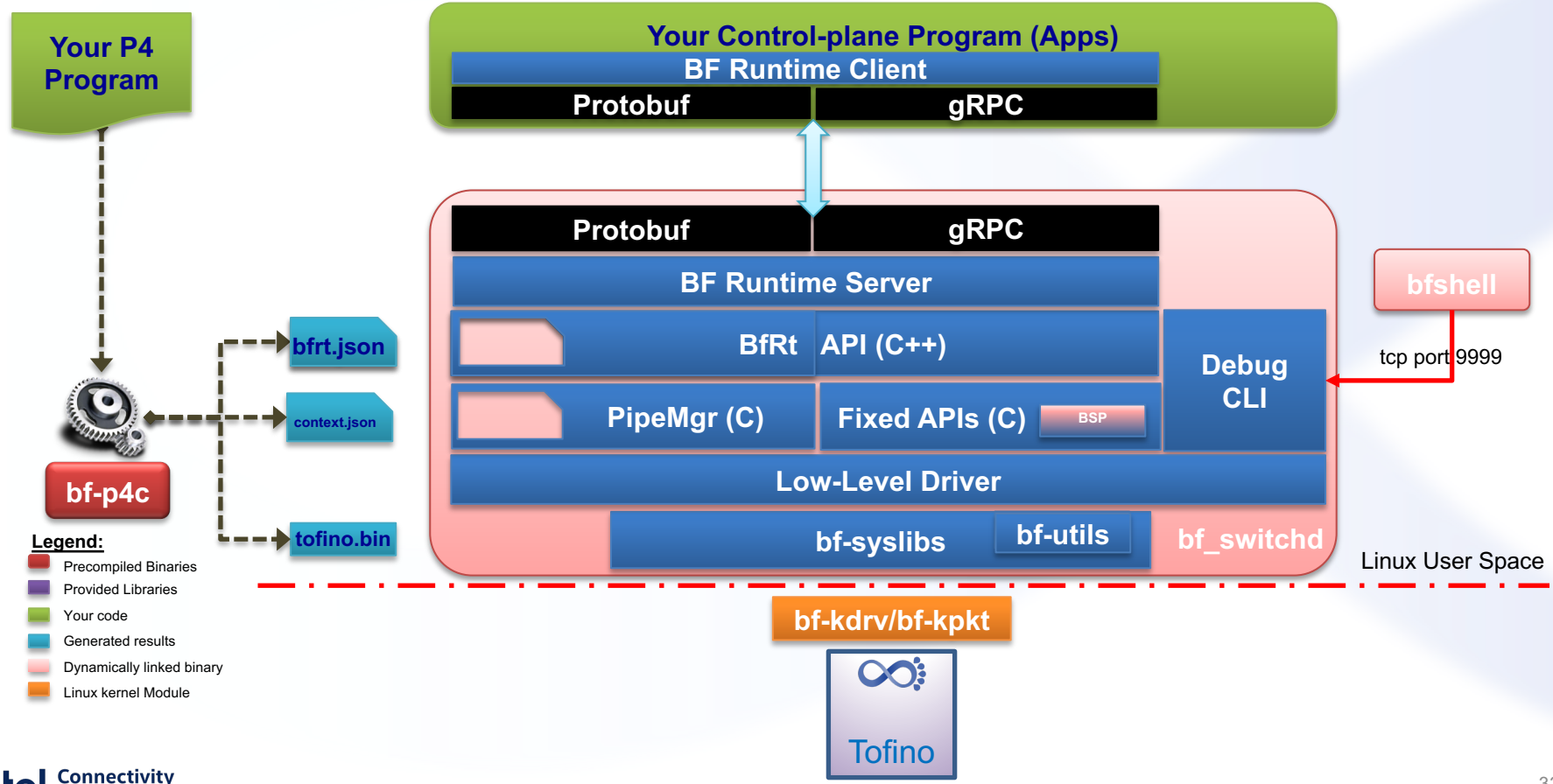
Core Components (Tofino). Single Process



Core Components (Tofino-Model). Remote Control

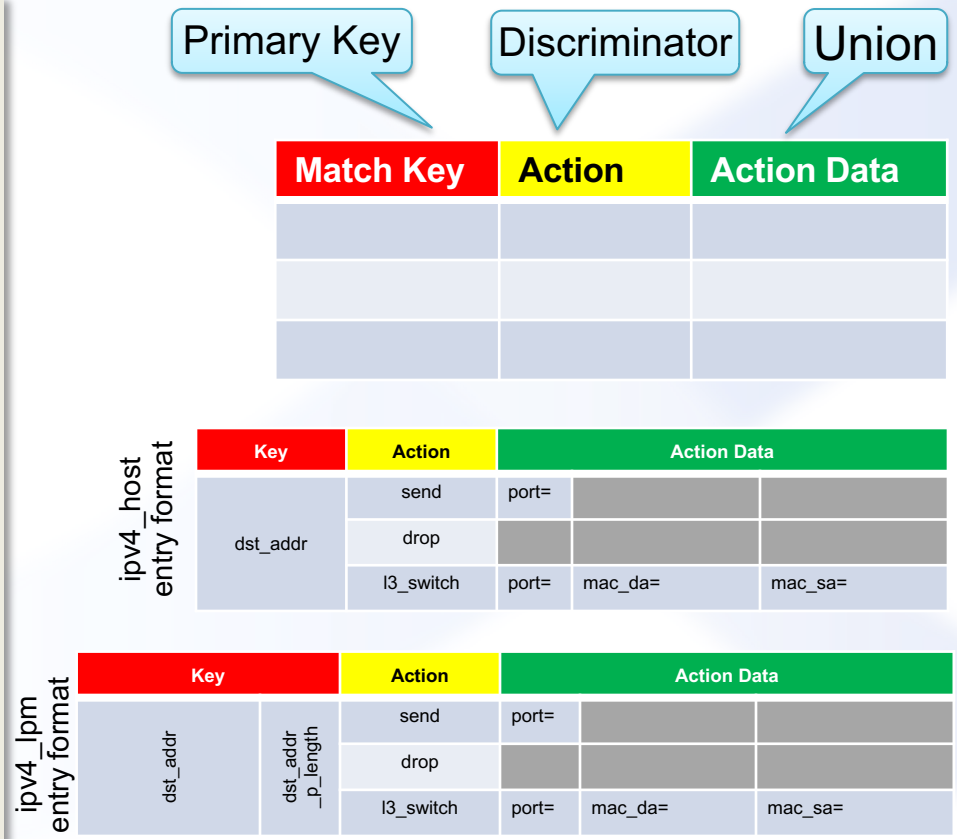


Core Components (Tofino ASIC). Remote Program Load



Representing Match-Action Tables in BRI

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

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

action l3_switch(PortId_t port,
                 bit<48> mac_da, bit<48> mac_sa)
{
    hdr.ethernet.dst_addr = new_mac_da;
    hdr.ethernet.src_addr = new_mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl |-| 1;
    send(port);
}


table ipv4_host {
    key     = { hdr.ipv4.dst_addr : exact; }
    actions = {send; drop; l3_switch; @defaultonly NoAction; }
    size    = 131072;
    const default_action = NoAction();
}

table ipv4_lpm {
    key     = { hdr.ipv4.dst_addr : lpm; }
    actions = {send;  drop; l3_switch; }
    size = 12288;
    default_action = send(64);
}
```



Representing Direct Externs in BRI

```
control Ingress(...)
{
    DirectCounter<bit<64>>
          (CounterType_t.PACKETS_AND_BYTES) ipv4_host_stats;

    action send(PortId_t port) {
        ig_tm_md.ucast_egress_port = port;
    }
    action drop() { ig_dprsr_md.drop_ctl = 1; }

    action ipv4_host_send(PortId_t port) {
        send(port); ipv4_host_stats.count();
    }
    action ipv4_host_drop()    {
        drop();     ipv4_host_stats.count();
    }

    table ipv4_host {
        key      = { hdr.ipv4.dst_addr : exact; }
        actions  = { @defaultonly NoAction;
                     ipv4_host_send; ipv4_host_drop; }
        size     = IPV4_HOST_TABLE_SIZE;
        counters = ipv4_host_stats;
        const default_action = NoAction();
    }
```

- **Each Extern has its own schema**
- **Direct Externs extend the corresponding table entries**
- **Some Externs require explicit synchronization**
  - Counters
  - Registers

Entry Data

Key	Action	Action Data	$COUNTER_SPEC_	
			BYTES	PKTS
dst_addr	ipv4_host_send	port=1		
	ipv4_host_drop	--		

Entry Data

Key	Action	Action Data	$METER_SPEC_			
			CIR_ KBPS	CBS_ KBITS	PIR_ KBPS	PBS_ KBITS
dst_addr	ipv4_host_send	port=				
	ipv4_host_drop					

Representing Indirect Externs in BRI

```
Counter<bit<64>, ipv4_stats_index_t>
(IPV4_STATS_SIZE, CounterType_t.PACKETS_AND_BYTES) ipv4_stats;

Meter<bit<10>>(ACL_METER_SIZE, MeterType_t.BYTES) acl_meter;

Register<data_t, box_num_t>(num_boxes) data_storage;
```

- **The index is the primary key**
  - Like an exact match table
- **No action is needed**
- **All entries are already in the table**
  - No add/delete

Key	(Volatile) Entry Data	
$COUNTER_INDEX	$COUNTER_SPEC_BYTES	$COUNTER_SPEC_PKTS
0		
1		
…		
16383		

Key	(Volatile) Entry Data
$REGISTER_INDEX	data_storage.f1[4]
0	
1	
…	
32767	

Key	Entry Data			
$METER_INDEX	$METER_SPEC_CIR_KBPS	$METER_SPEC_CIR_KBITS	$METER_SPEC_PIR_KBPS	$METER_SPEC_PIR_KBITS
0				
1				
…				
4095				

Representing Fixed-Function Components

- **Fixed-function components are represented as tables too**
  - With or without an action

This field is volatile

port.port

Key	Entry Data								
$DEV_PORT	$SPEED=	$FEC=	$N_LANES=	$ENABLE=	$AUTONEG=	$TX_MTU=	$RX_MTU	$PORT_UP	...
0	100G	None	4	True	False	9216	9216	True	
5	25G	FC	1	True	False	1536	1536	False	
64	100G	RS	4	True	False	9216	9216	True	

pre.node

Key	Entry Data		
$MULTICAST_NODE_ID	$MULTICAST_RID	$MULTICAST_LAG_ID	$DEV_PORT

An arbitrary value, specified by the user

pre.mgid

Key	Entry Data					
$MGID	$MULTICAST_NODE_			$MULTICAST_ECMP_		
	ID	L1_XID_VALID	L1_XID	ID	L1_XID_VALID	L1_XID
7	[10, 11, 15]	[T, F, T]	[20, 0, 20]	[ ]	[ ]	[ ]

Fields can also contain lists of values

Using P4Studio™

- Screencast

Intel Connectivity Academy (http://intel.com/ica)

- Comprehensive training for beginners and experts
- Online Zoom Sessions with live Q&A
- Hands-On Labs

50% Discount for P4 Workshop Attendees!
Coupon Code: P4_ONF_2021

intel® Connectivity Academy