ZenosIC Lanting - A New P4-Programmable ASIC: Essentials and Programming Tools

Asterfusion

Turnkey Solutions of Open Networking in Cloud, Enterprise and AI

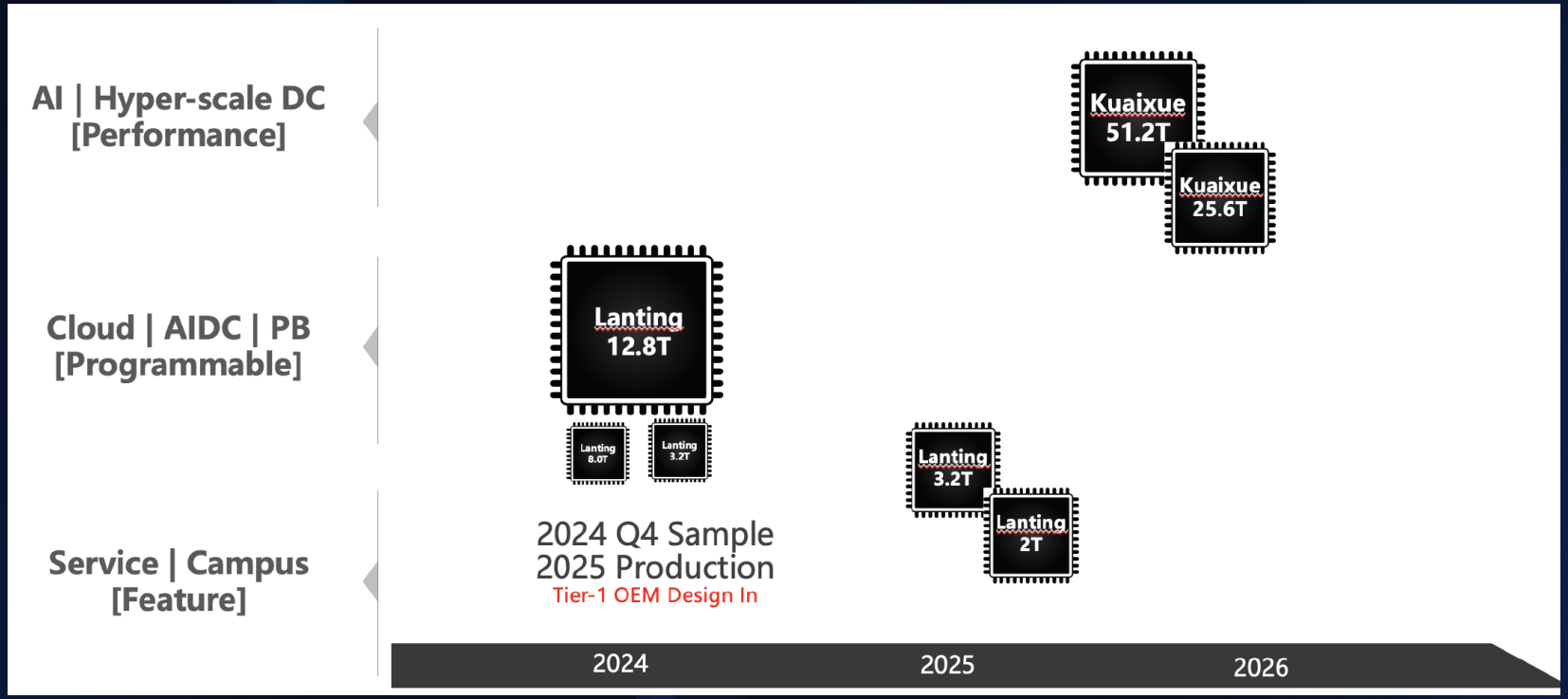
Background: The Challenge of Intel Tofino EOL

- Intel Tofino, the industry's first P4-programmable Ethernet switch chip, was widely adopted in data centers, network telemetry, and load balancing.
- In 2024, Intel announced the End of Life (EOL) for the Tofino series, ceasing support and production.
- Challenges:
  1. Growing demand for programmable networking (data centers, security, 5G, AI workloads).
  2. Tofino users face migration issues: no direct replacement; fixed-function ASICs lack flexibility.
  3. Need for higher performance and lower power solutions.

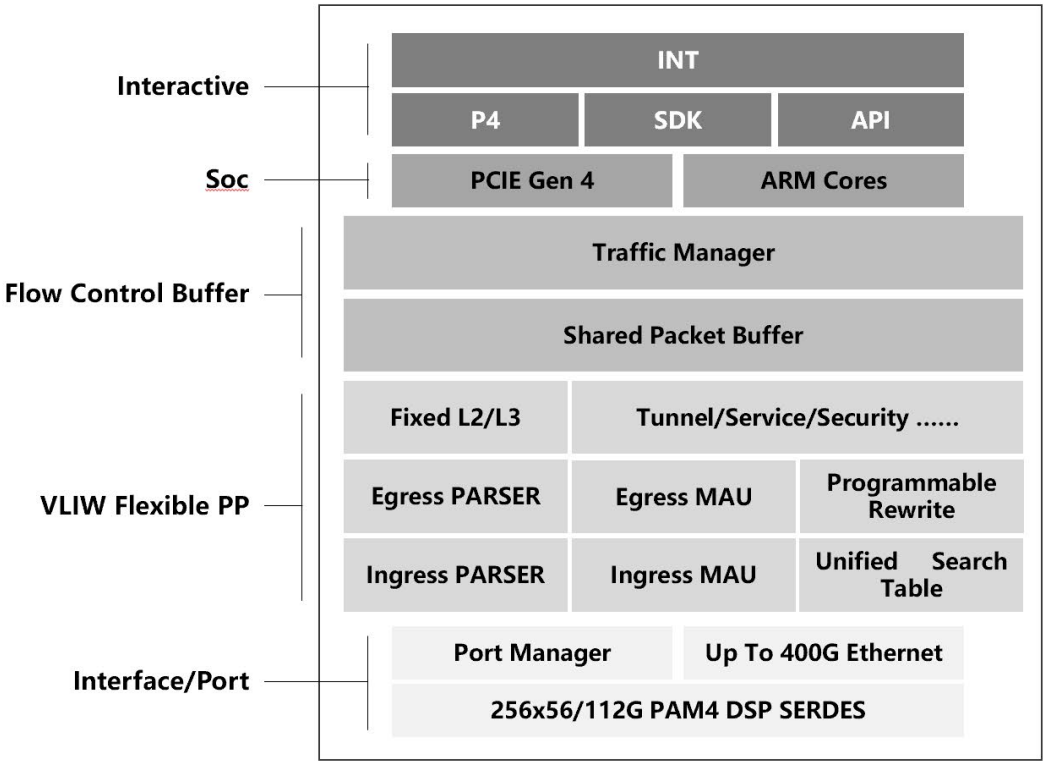
Solution – A new generation chip for future's requirement

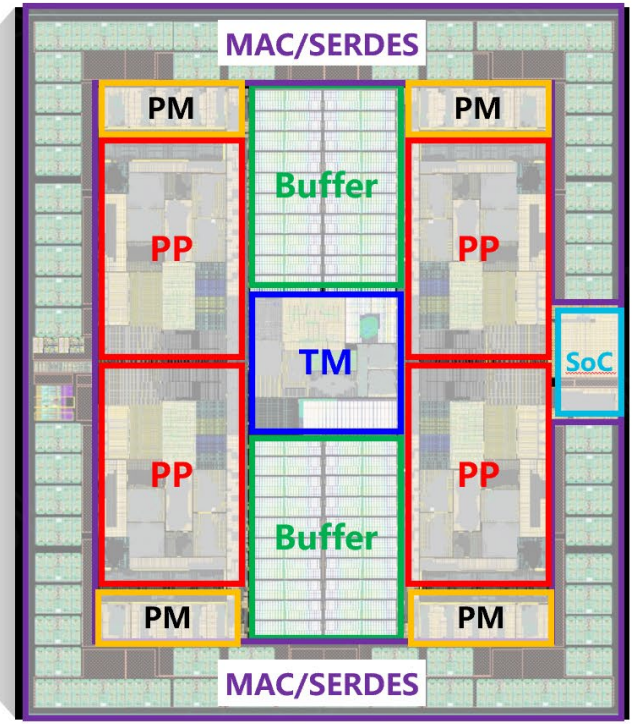
- Goal: Seamlessly replace Tofino while enhancing performance and flexibility.
- Core Principles:
  1. Retain P4 programmability for compatibility with existing ecosystems.
  2. Boost throughput and efficiency for hyperscale data centers and edge computing.
  3. Fix some shortcomings in the Tofino/Tofino2 specification and implementation.
  4. Offer innovative toolchain to accelerate development and deployment.
- Value:
  1. Fills the gap left by Tofino EOL.
  2. Paves the way for next-gen networks (e.g., 400~800GbE, AI-driven infrastructure).

The Status and Roadmap of New Generation Programmable Chips

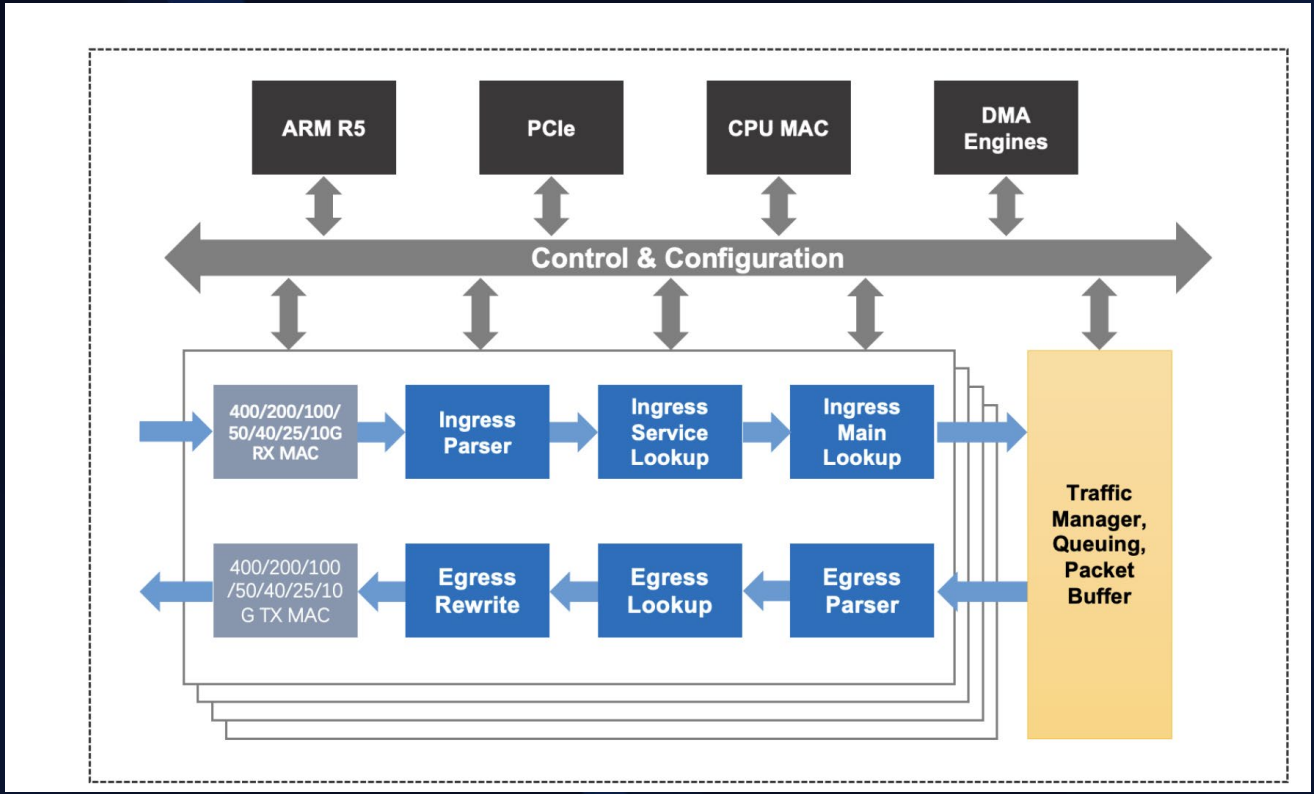


LanTing 12.8T Architecture

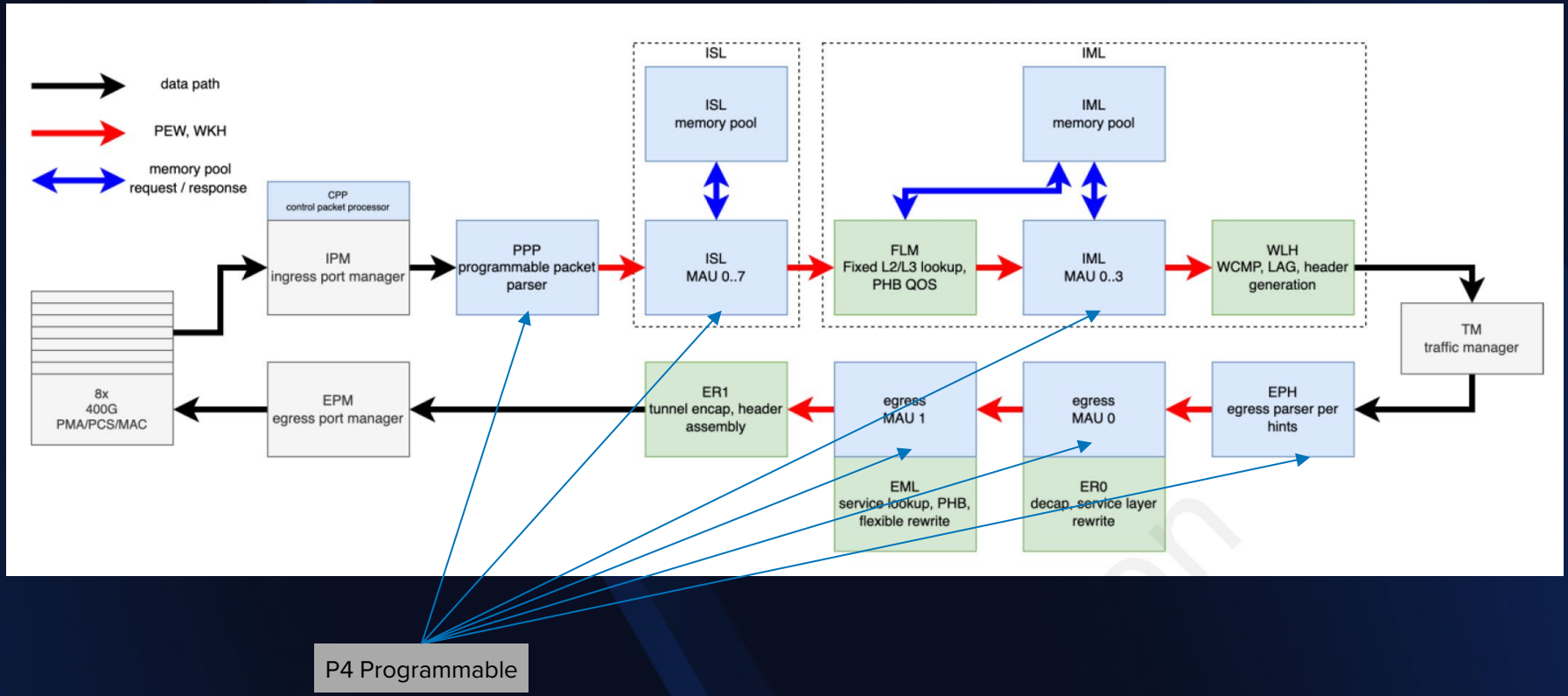




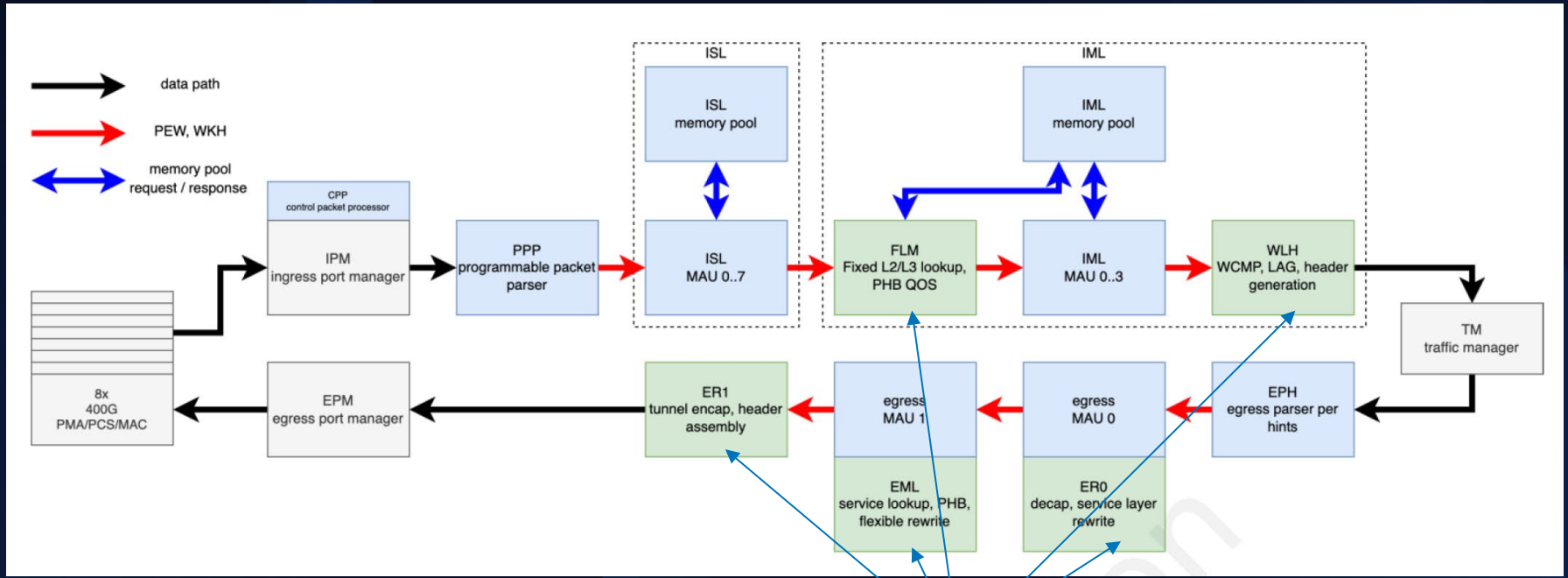
4 x 3.2G Pipelines That Built with Native Programmable Capabilities



Packet Processing Pipeline Inside



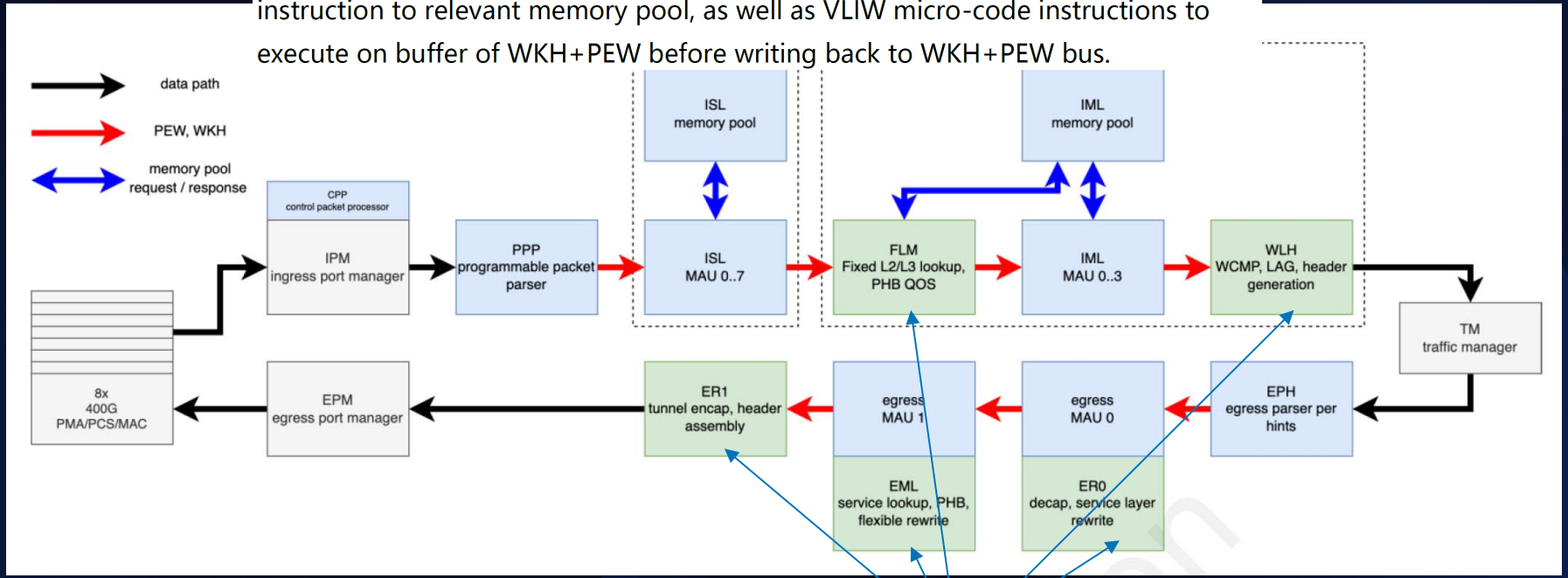
Packet Processing Pipeline Inside



P4 Programmable + Classic L2/L3, Qos and WCMP

Packet Processing Pipeline Inside

There are total 12 MAU (match action unit) instances at ingress pipeline, and 2 instances at egress. Each MAU classifies PEW flags to decide lookup and read instruction to relevant memory pool, as well as VLIW micro-code instructions to execute on buffer of WKH+PEW before writing back to WKH+PEW bus.

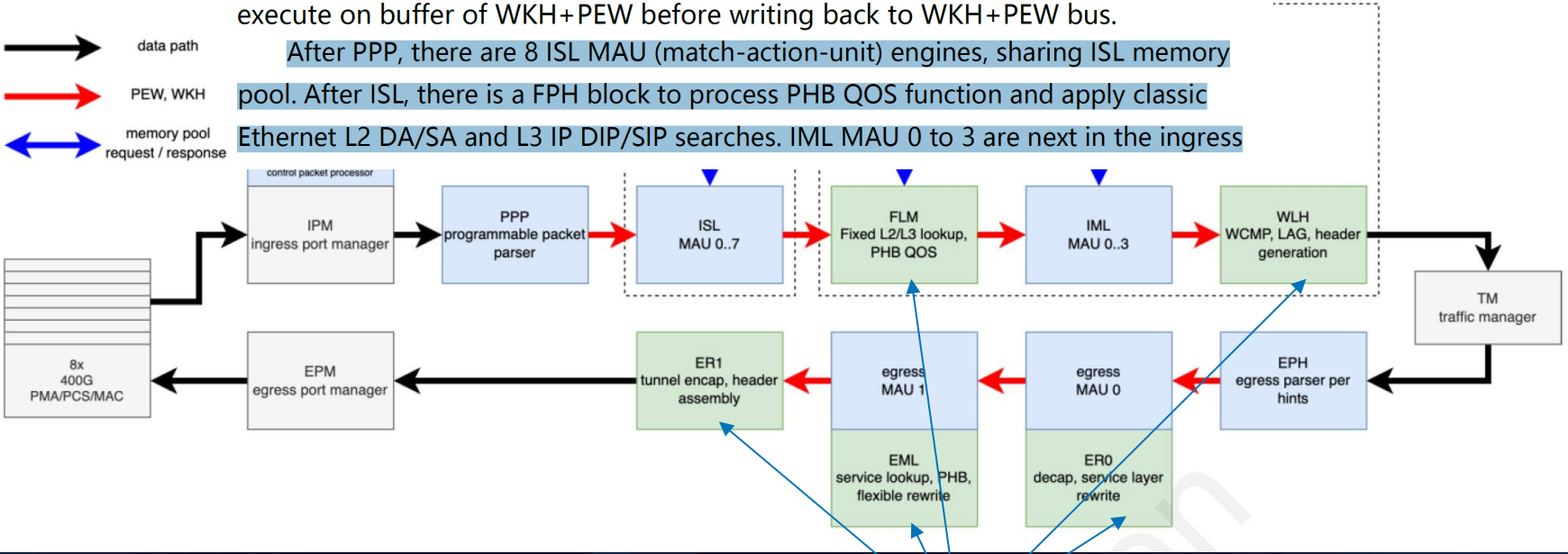


P4 Programmable + Classic L2/L3, Qos and WCMP

Packet Processing Pipeline Inside

There are total 12 MAU (match action unit) instances at ingress pipeline, and 2 instances at egress. Each MAU classifies PEW flags to decide lookup and read instruction to relevant memory pool, as well as VLIW micro-code instructions to execute on buffer of WKH+PEW before writing back to WKH+PEW bus.

After PPP, there are 8 ISL MAU (match-action-unit) engines, sharing ISL memory pool. After ISL, there is a FPH block to process PHB QOS function and apply classic Ethernet L2 DA/SA and L3 IP DIP/SIP searches. IML MAU 0 to 3 are next in the ingress



P4 Programmable + Classic L2/L3, Qos and WCMP

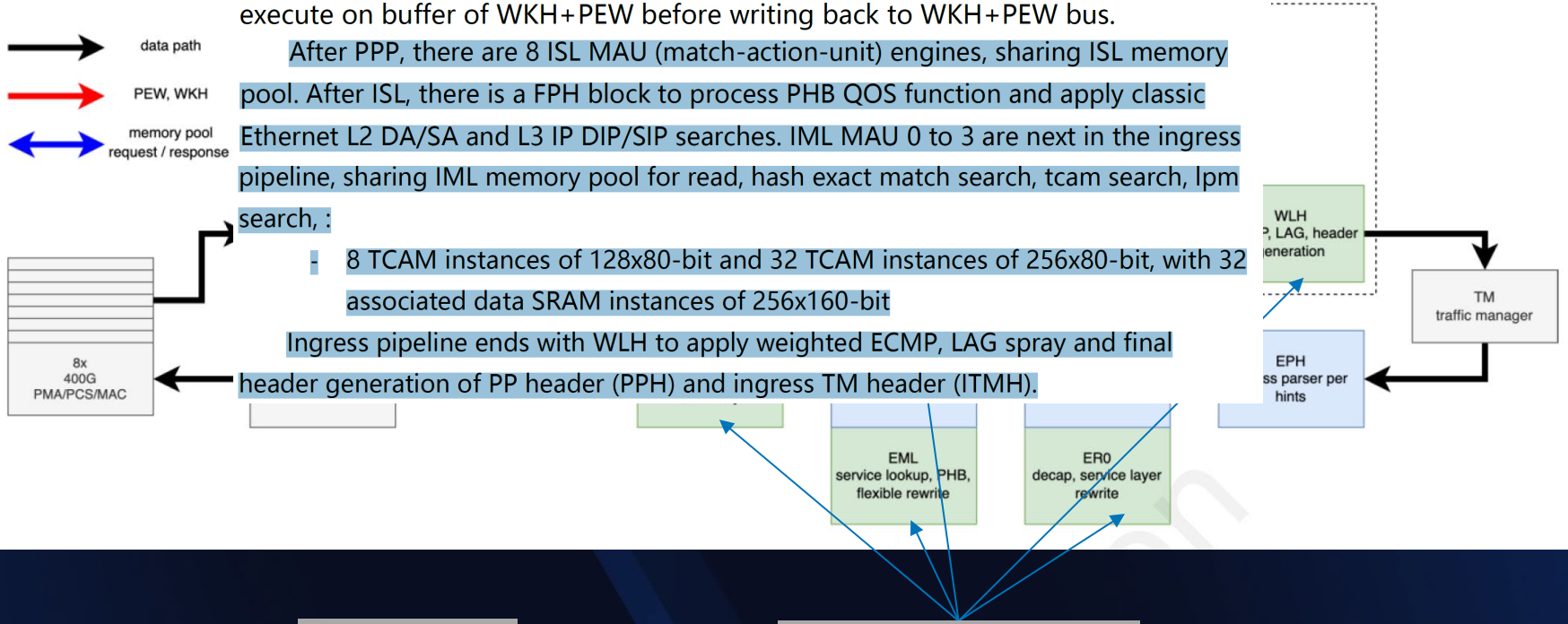
Packet Processing Pipeline Inside

There are total 12 MAU (match action unit) instances at ingress pipeline, and 2 instances at egress. Each MAU classifies PEW flags to decide lookup and read instruction to relevant memory pool, as well as VLIW micro-code instructions to execute on buffer of WKH+PEW before writing back to WKH+PEW bus.

After PPP, there are 8 ISL MAU (match-action-unit) engines, sharing ISL memory pool. After ISL, there is a FPH block to process PHB QOS function and apply classic Ethernet L2 DA/SA and L3 IP DIP/SIP searches. IML MAU 0 to 3 are next in the ingress pipeline, sharing IML memory pool for read, hash exact match search, tcam search, lpm search, :

- 8 TCAM instances of 128x80-bit and 32 TCAM instances of 256x80-bit, with 32 associated data SRAM instances of 256x160-bit

Ingress pipeline ends with WLH to apply weighted ECMP, LAG spray and final header generation of PP header (PPH) and ingress TM header (ITMH).



P4 Programmable + Classic L2/L3, Qos and WCMP

Packet Processing Pipeline Inside

There are total 12 MAU (match action unit) instances at ingress pipeline, and 2 instances at egress. Each MAU classifies PEW flags to decide lookup and read instruction to relevant memory pool, as well as VLIW micro-code instructions to execute on buffer of WKH+PEW before writing back to WKH+PEW bus.

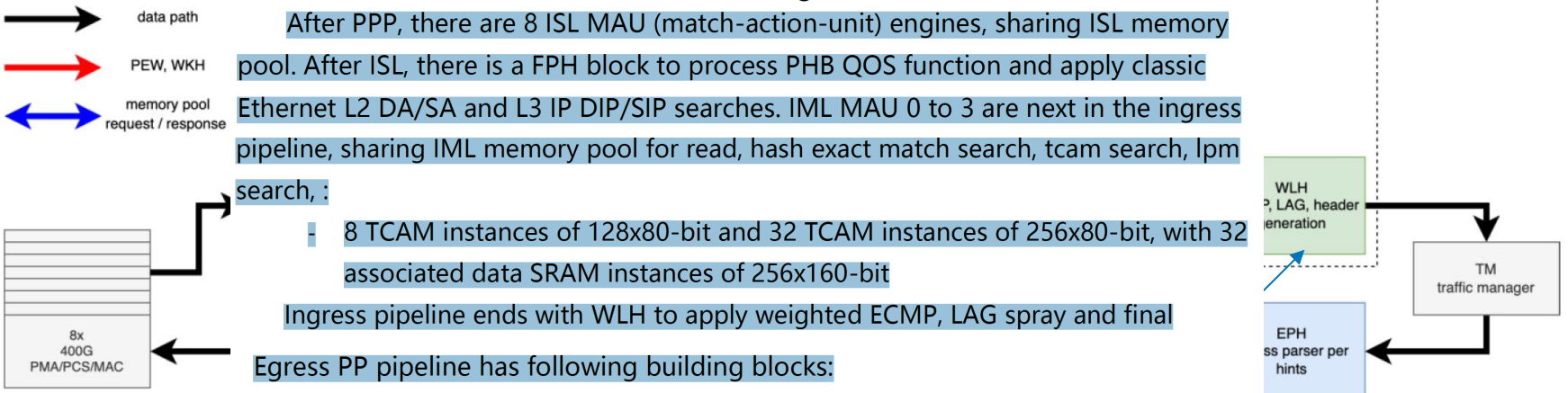
After PPP, there are 8 ISL MAU (match-action-unit) engines, sharing ISL memory pool. After ISL, there is a FPH block to process PHB QOS function and apply classic Ethernet L2 DA/SA and L3 IP DIP/SIP searches. IML MAU 0 to 3 are next in the ingress pipeline, sharing IML memory pool for read, hash exact match search, tcam search, lpm search, :

- 8 TCAM instances of 128x80-bit and 32 TCAM instances of 256x80-bit, with 32 associated data SRAM instances of 256x160-bit

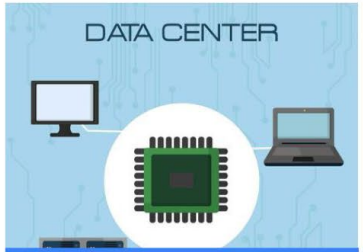
Ingress pipeline ends with WLH to apply weighted ECMP, LAG spray and final

Egress PP pipeline has following building blocks:

- EPH (Egress Parser per Hints) to parse header per hints conveyed through PP header (PPH)
- ER0 to apply decapsulation and service layer rewrite; egress MAU 0 is part of ER0
- EML to apply service lookup, PHB QOS and flexible rewrite per PPH instructions; egress MAU 1 is part of EML
- ER1 to apply tunnel encapsulation, and final header assembly



Application Cases and Performance Data Summary



Switching

Performance First

- 4*3.2T pipeline
- <600ns with IP route and ECMP
- Fixed MAU for L2-3 forwarding

Gateway

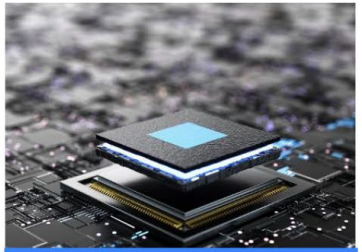
Programmability / Scalability

- 5.4BPPS
- Up to 20Kx160-bit ingress ACL rules per pipeline

Routing

Programmability / Scalability

- Special mode to use multiple pipelines to double/quadruple MAUs & Tables



NPB

Flexibility First

- 130MB fully shared packet buffer
- Routing capabilities with Switching performance

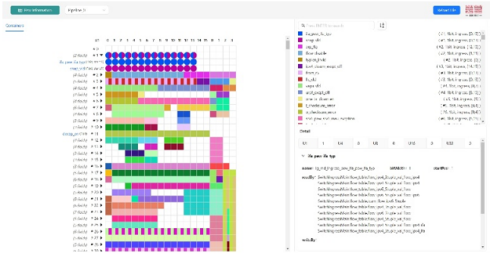
Developer Info: SDK and Tools

- Support P4 and Zenosic extensions
- 100% TCAM/SRAM utilization
- Support auto, user-defined and hybrid table configuration
- Support both SDK control API (Done) & P4 run-time
- Visualizing pipeline programming and table configuration
- Display mem blocks and all the table information of MAUs by XML

SDK at Glance

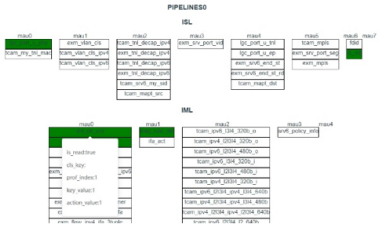
Detailed P4 Insight

- Visualizing pipeline programming and table configuration



Configuration & Tuning Toolbox

- Display mem blocks and all the table information of MAUs by XML



Side by Side P4 Example (Parser)

```
state parse_ipv6 {
    pkt.extract(hdr.ipv6);
    ig_md.cal_data.key_meta.val1 = key_meta_alu((bit<16>)hdr.ipv6.next_hdr, 0x00ff, 0x0);
    hdr.internal_hdr.hic.outer_l3_hint = SWITCH_L3_HINT_V6;
    STORE_OUTER_IPV6(ig_md, hdr.ipv6);
    STORE_OUTER_IPV6_L4_PROTO(ig_md, hdr.ipv6);
    transition select(ig_md.cal_data.key_meta.val1) {
        (bit<16>)IP_PROTOCOLS_TCP: parse_tcp;
        (bit<16>)IP_PROTOCOLS_UDP: parse_udp;
        (bit<16>)IP_PROTOCOLS_UDP_LITE: parse_udp_lte;
        (bit<16>)IP_PROTOCOLS_ICMPV6: parse_icmp;
        (bit<16>)IP_PROTOCOLS_SCTP: parse_sctp;
        (bit<16>)IP_PROTOCOLS_GRE: parse_gre;
        (bit<16>)IP_PROTOCOLS_SRV6: parse_srv6;
        (bit<16>)IP_PROTOCOLS_FRAGV6: parse_fragv6;
        (bit<16>)IP_PROTOCOLS_AH: parse_ah;
        (bit<16>)IP_PROTOCOLS_IPV6_HOP_BY_HOP: parse_ipv6_hop_by_hop;
        (bit<16>)IP_PROTOCOLS_ESP: parse_esp_over_ipv6;
        (bit<16>)IP_PROTOCOLS_IFA: parse_ifa_header;
        (bit<16>)IP_PROTOCOLS_ETHERNET: parse_inner_ethernet_over_ip;
        (bit<16>)IP_PROTOCOLS_IPV4: parse_inner_ipv4_over_ip;
        (bit<16>)IP_PROTOCOLS_IPV6: parse_inner_ipv6_over_ip;
        default: accept;
    }
}
```

```
    state parse_udp {
        pkt.extract(hdr.udp);
        ig_md.cal_data.key_meta.val1 = key_meta_alu(hdr.udp.l4_dport, 0xffff, 0x0);
        hdr.internal_hdr.hic.outer_l4_hint = SWITCH_OUTER_L4_HINT_UDP;
        ig_md.ingress_pew.ip_tnl_present = 1;
        STORE_OUTER_UDP(ig_md, hdr.udp);
        transition select(ig_md.cal_data.key_meta.val1) {
            UDP_PORT_VXLAN: parse_vxlan;
            UDP_PORT_VXLAN_GPE: parse_vxlan_gpe;
#ifndef  SUPPORT_MPLS_OFF
            UDP_PORT_MPLS: parse_udp_mpls;
#endif
            UDP_PORT_ROCEV2: parse_rocev2;
            UDP_PORT_GENEVE: parse_geneve;
            UDP_PORT_BFD_SINGLE_HOP: parse_bfd_single_hop;
            UDP_PORT_BFD_ECHO: parse_bfd_echo;
            UDP_PORT_BFD_MULTI_HOP: parse_bfd_multi_hop;
            UDP_PORT_BFD_OVER_LAG: parse_bfd_over_lag;
            default: accept;
        }
    }
```

```
    state parse_ipv6 {
#ifdef IPV6_ENABLE
        pkt.extract(hdr.ipv6);
#ifdef INNER_HASH_ENABLE
        local_md.hash_fields.ip_type = SWITCH_IP_TYPE_IPV6;
        local_md.hash_fields.ip_src_addr = hdr.ipv6.src_addr;
        local_md.hash_fields.ip_dst_addr = hdr.ipv6.dst_addr;
        local_md.hash_fields.ip_proto = hdr.ipv6.next_hdr;
        local_md.hash_fields.ipv6_flow_label = hdr.ipv6.flow_label;
#endif /* INNER_HASH_ENABLE */
#ifdef NAT_ENABLE
        tcp_checksum.subtract({hdr.ipv6.src_addr,hdr.ipv6.dst_addr});
        udp_checksum.subtract({hdr.ipv6.src_addr,hdr.ipv6.dst_addr});
#endif
        transition select(hdr.ipv6.next_hdr) {
            IP_PROTOCOLS_ICMPV6 : parse_icmp;
            IP_PROTOCOLS_TCP : parse_tcp;
            IP_PROTOCOLS_UDP : parse_udp;
#if defined(GRE_ENABLE) || defined(NVGRE_ENABLE)
            IP_PROTOCOLS_GRE : parse_ip_gre;
#endif /* GRE_ENABLE || NVGRE_ENABLE */
#ifdef IPINIP_ENABLE
            IP_PROTOCOLS_IPV4 : parse_ipinip;
            IP_PROTOCOLS_IPV6 : parse_ipv6inip;
#endif
#ifdef SRV6_ENABLE
            IP_PROTOCOLS_ROUTING : parse_srh_base;
#endif /* SRV6_ENABLE */
            default : accept;
        }
#else
        transition accept;
#endif
    }

    state parse_udp {
        pkt.extract(hdr.udp);
#ifdef INNER_HASH_ENABLE
        local_md.hash_fields.l4_src_port = hdr.udp.src_port;
        local_md.hash_fields.l4_dst_port = hdr.udp.dst_port;
#endif /* INNER_HASH_ENABLE */
#ifdef NAT_ENABLE
        udp_checksum.subtract_all_and_deposit(local_md.tcp_udp_checksum);
        udp_checksum.subtract({hdr.udp.checksum});
        udp_checksum.subtract({hdr.udp.src_port, hdr.udp.dst_port});
#endif
        transition select(hdr.udp.dst_port) {
            UDP_PORT_GTP_U : parse_gtp_u;
#ifdef VXLAN_ENABLE
            udp_port_vxlan : parse_vxlan;
#endif
#ifdef BFD_OFFLOAD_ENABLE
            UDP_PORT_BFD_1HOP : parse_bfd;
            UDP_PORT_BFD_MHOP : parse_bfd;
            UDP_PORT_BFD_ECHO : parse_bfd;
#endif /* BFD_OFFLOAD_ENABLE */
            UDP_PORT_ROCEV2 : parse_rocev2;
                default : accept;
            }
    }
```

Side by Side P4 Example (Table)

```
table srv_port_vid {
    key = {
        ig_md.ingress_pew.vid.cvid : exact;
        ig_md.ingress_pew.vid.svid : exact;
        ig_md.ingress_pew.port_cfg_pew.lgc_port : exact;
        typ : exact;
    }

    actions = {
        srv_port_vid_hit;
        NoAction;
    }

    const default_action = NoAction;
    size = exm_srv_table_size;
    mem_type = SWITCH_MEM_TYPE_FLEX;
    counters = vid_cnt;
}
```

```
@name(".port_double_tag_to_bd_mapping")
table port_double_tag_to_bd_mapping {
    key = {
        local_md.ingress_port_lag_index : exact;
        hdr.vlan_tag[0].isValid() : exact;
        hdr.vlan_tag[0].vid : exact;
        hdr.vlan_tag[1].isValid() : exact;
        hdr.vlan_tag[1].vid : exact;
    }

    actions = {
        NoAction;
        port_vlan_miss;
        set_bd_properties;
    }

    const default_action = NoAction;
    implementation = bd_action_profile;
    size = double_tag_table_size;
}
```

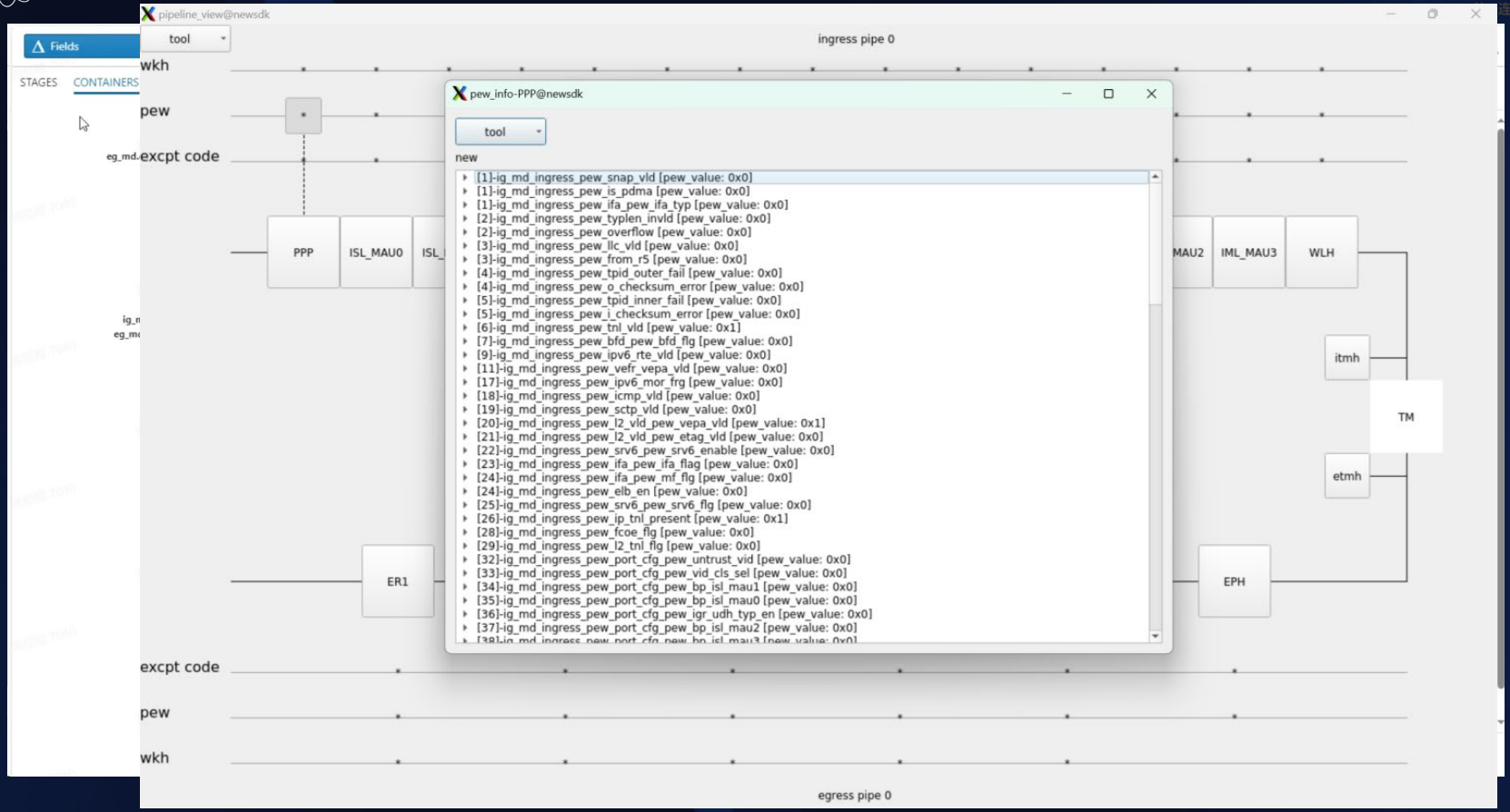
Side by Side P4 Example (Action)

```
action fdid_hit(
    bit<16> l2_dmn_mtr,
    bit<16> l2_intf_mtr,
    bit<14> map_fdid,
    bit<6>  iacl_prof_idx,
    bit<15> l2_bum_idx,
    bit<3>  udh_type,
    bit<1>  l2_lrn_excpt_move,
    bit<1>  l2_bum_is_eid,
    bit<2>  l2_ipv6_mc_en,
    bit<2>  l2_ipv4_mc_en,
    bit<1>  l2_lrn_excpt_miss,
    bit<1>  l2_lrn_dis,
    bit<8>  l2_topo_idx,
    bit<8>  pif_bdid_flt_idx,
    bit<14> l3_intf) {
    ig_md.ingress_pew.l2l3_off_bdid_pew.bdid = map_fdid;
    ig_md.ingress_pew.iacl_prof_idx = iacl_prof_idx | ig_md.ingress_pew.iacl_prof_idx;
    ig_md.ingress_pew.l2_topo_idx = l2_topo_idx;
    ig_md.ingress_pew.l2l3_mtr.l2_dmn_mtr = l2_dmn_mtr;
    ig_md.ingress_pew.l2l3_mtr.l2_intf_mtr = l2_intf_mtr;
    ig_md.ingress_pew.l2l3_flg_pew.l2_lrn_excpt_move = l2_lrn_excpt_move;
    ig_md.ingress_pew.l2l3_flg_pew.l2_bum_is_eid = l2_bum_is_eid;
    ig_md.ingress_pew.l2_bum_idx_pew.l2_bum_idx = l2_bum_idx;
    ig_md.ingress_pew.pif_bdid_flt_idx = pif_bdid_flt_idx;
    ig_md.ingress_pew.udh_type = udh_type;
    ig_md.ingress_pew.ctl_flag.l2_lrn_dis = l2_lrn_dis | ig_md.ingress_pew.ctl_flag.l2_lrn_dis;

    l2_bd_l2_lrn_dis_lur            = l2_lrn_dis;
    l2_bd_l2_ipv6_mc_en_lur         = l2_ipv6_mc_en;
    l2_bd_l2_ipv4_mc_en_lur         = l2_ipv4_mc_en;
    l2_lrn_excpt_miss_lur           = l2_lrn_excpt_miss;
    l2_bd_l3_intf_id_lur            = l3_intf;
}
```

```
    @name(".set_bd_properties")
    action set_bd_properties(switch_bd_t bd,
                             switch_vrf_t vrf,
                             bool vlan_arp_suppress,
                             switch_packet_action_t vrf_ttl_violation,
                             bool vrf_ttl_violation_valid,
                             switch_packet_action_t vrf_ip_options_violation,
                             bool vrf_unknown_l3_multicast_trap,
                             switch_bd_label_t bd_label,
                             switch_stp_group_t stp_group,
                             switch_learning_mode_t learning_mode,
                             bool ipv4_unicast_enable,
                             bool ipv4_multicast_enable,
                             bool igmp_snooping_enable,
                             bool ipv6_unicast_enable,
                             bool ipv6_multicast_enable,
                             bool mld_snooping_enable,
                             bool mpls_enable,
                             switch_multicast_rpf_group_t mrpf_group,
                             switch_nat_zone_t zone) {
        local_md.bd = bd;
        local_md.flags.vlan_arp_suppress = vlan_arp_suppress;
        local_md.ingress_outer_bd = bd;
#ifdef INGRESS_ACL_BD_LABEL_ENABLE
        local_md.bd_label = bd_label;
#endif
        local_md.vrf = vrf;
        local_md.flags.vrf_ttl_violation = vrf_ttl_violation;
        local_md.flags.vrf_ttl_violation_valid = vrf_ttl_violation_valid;
        local_md.flags.vrf_ip_options_violation = vrf_ip_options_violation;
        local_md.flags.vrf_unknown_l3_multicast_trap = vrf_unknown_l3_multicast_trap;
        local_md.stp.group = stp_group;
        local_md.multicast.rpf_group = mrpf_group;
        local_md.learning.bd_mode = learning_mode;
        local_md.ipv4.unicast_enable = ipv4_unicast_enable;
        local_md.ipv4.multicast_enable = ipv4_multicast_enable;
        local_md.ipv4.multicast_snooping = igmp_snooping_enable;
        local_md.ipv6.unicast_enable = ipv6_unicast_enable;
        local_md.ipv6.multicast_enable = ipv6_multicast_enable;
        local_md.ipv6.multicast_snooping = mld_snooping_enable;
#ifdef MPLS_ENABLE
        local_md.mpls_enable = mpls_enable;
#endif
#ifdef NAT_ENABLE
```

Side by Side P4 Example (P4i)



More About Lanting's Platform

- Hardware: (Q3/25 GA)

----32x400G (Test in Lab)

----32x100G (w/t 2x 100G DPDK/VPP DPU extension slots) (in development)

----128x100G (in planning)

- Software:

----P4 compiler and simulation model as daily development tools in OEM partner

----SONiC 202411 community version and commercial version AsterNOS ready to run

----DC Border Gateway and VTEP Gateway in Tier 1 CSP finished testing as alternative to Tofino

----Packet Broker in SONiC finished several testing for tech standard compliance as alternative to Tofino

A Few Words of ZenosIC

- Founded in 2021
- HQ in Shanghai, China
- Stealth mode , 3 series funding

- Engineers: 100+ IC design and verification
- Execs. from H3C, Cisco, MTK and Barefoot

- Design-in at one top NEP in China as alternative to Tofino
- SONiC/SAI native SDK support from partner

- 3.2T to 12.8T Programmable switch ASIC
- Classic L3 and P4 mixed pipeline design

A Few Words of Asterfusion

- Established in 2017
- HQ in Suzhou, China, R&D Centers in Xi'An and Wuhan ， 120+ software developers working on SONiC, 20+ H/W hardware engineers; Factory: Suzhou
- Working with STORDIS GmbH as strategy partner in Europe to deliver Open Networking Solutions including latest P4 switches

- Turnkey full stack SONiC eco solution in DC, enterprise and AI
- Single repository, commercial ready SONiC for Marvell Teralynx (AI), Marvell Prestera(Enterprise) , Broadcom StrataXGS (Cloud) , and P4
- Best low latency 51.2 T RoCEv2 switch with 6K 800G ports installation
- Widely deployed large scale OpenWiFi APs and Core/Access switches with SONiC running on A52
- 2K+ plus successful installation of P4 Data Center Border Leafs and Gateways


Asterfusion
Asteria Dashboard
FusionNOS
AsterNOS
Thank You!