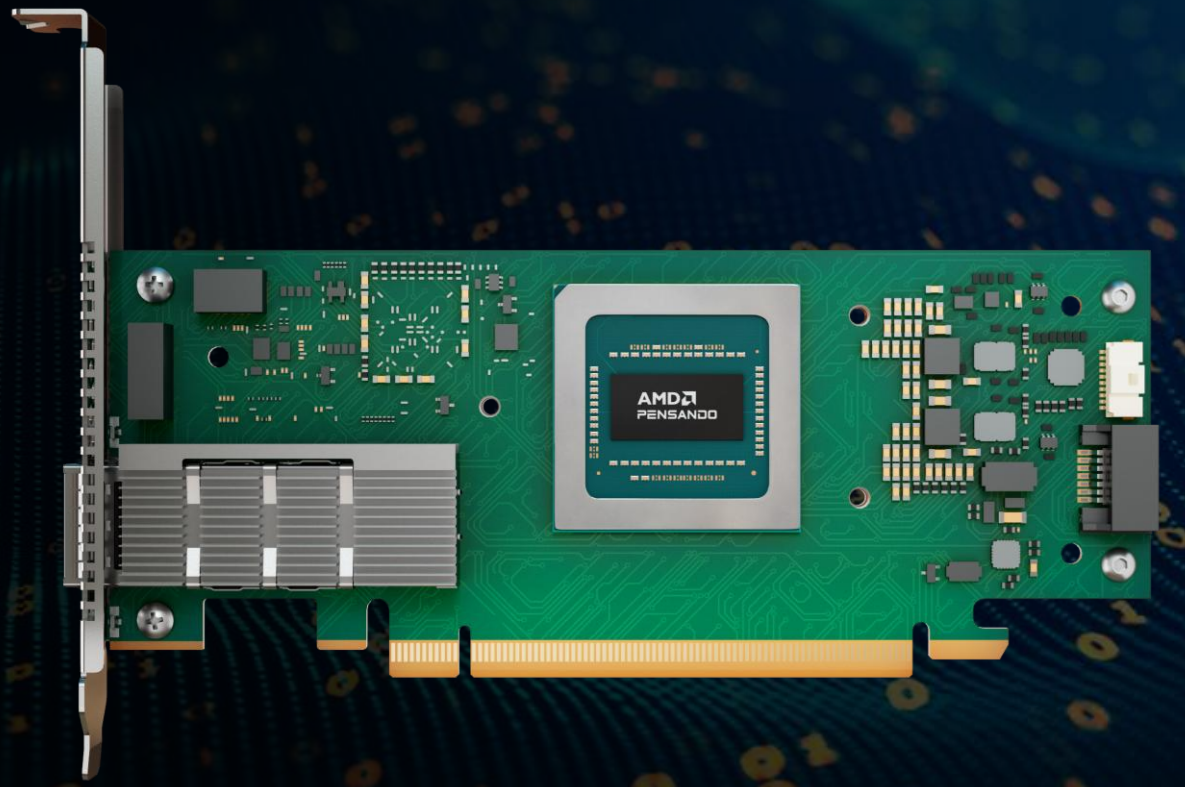
# AMD Pensando™ Pollara 400 AI NIC Architecture and Application




**Kevin Chu, Sr. Fellow**
**NTSG SoC Architect**

Hot Chips 2025

# Agenda

## AMD Pensando™ Pollara 400 AI NIC Overview

## AMD Pensando P4 Architecture

## Enhancements

- Address translation
- Atomic operations
- Pipeline cache coherency

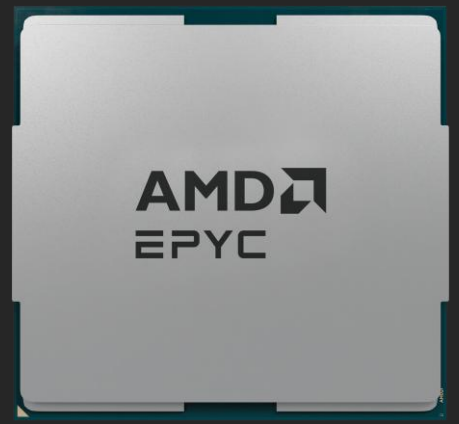
## AI Scale-Out network challenges and solutions

- Adaptive packet spray
- Path aware congestion avoidance
- Explicit loss notification and selective retransmission

## AMD Pensando Pollara 400 AI NIC is UEC Ready

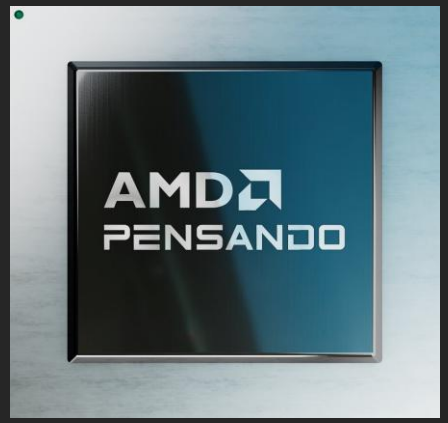
# Advancing Data Center Solutions

## Data Center CPUs




## Data Center GPUs

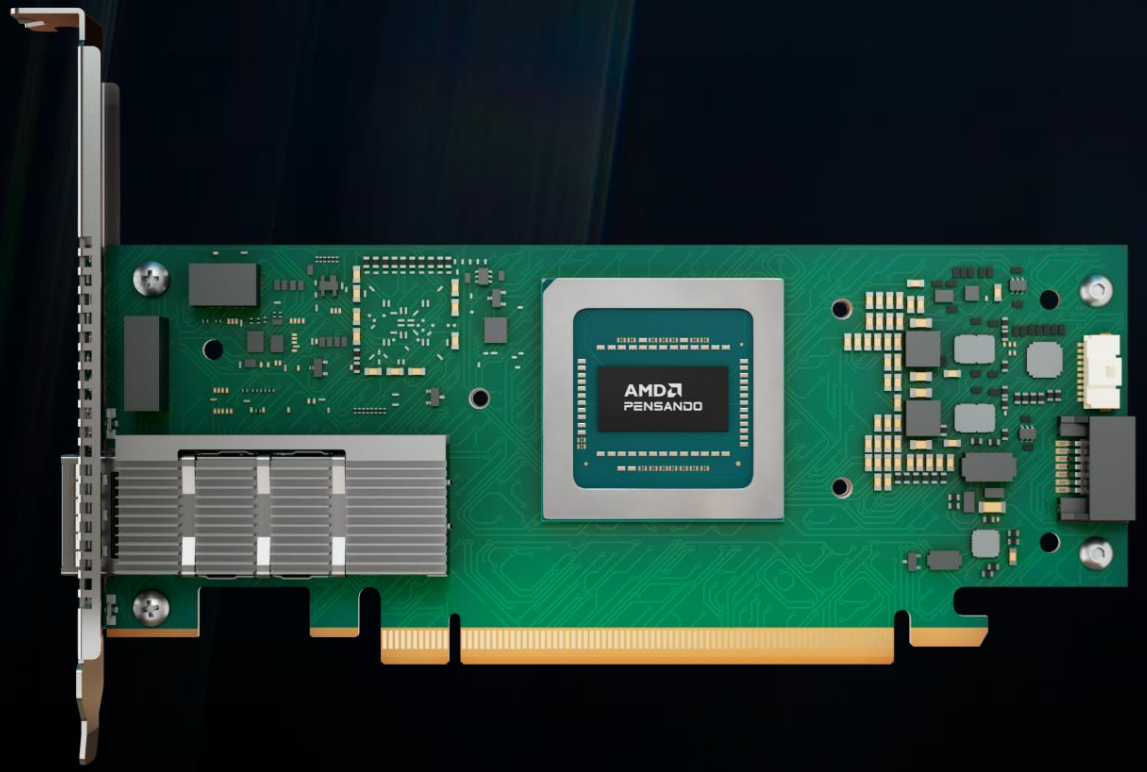

## Networking



# AMD Pensando™ Pollara 400 AI NIC

## Industry's first Ultra Ethernet Consortium ready AI NIC

- Programmable Hardware Pipeline
- Up to 1.25x Performance Boost*
- 400 Gbps
- Open Ecosystem
- UEC Ready RDMA
- Reduction in Job Completion Times
- High Availability



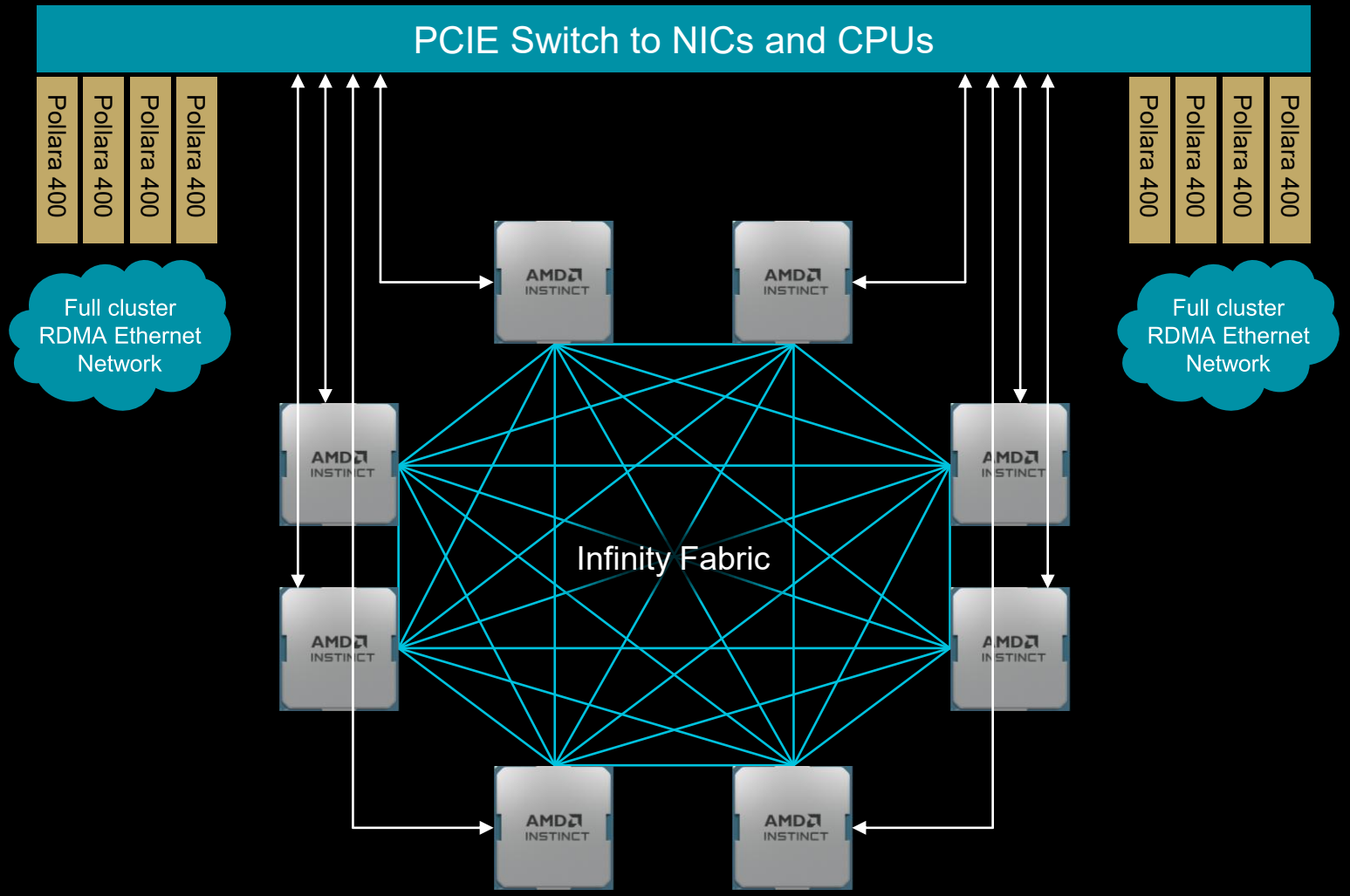

Ultra Ethernet Consortium



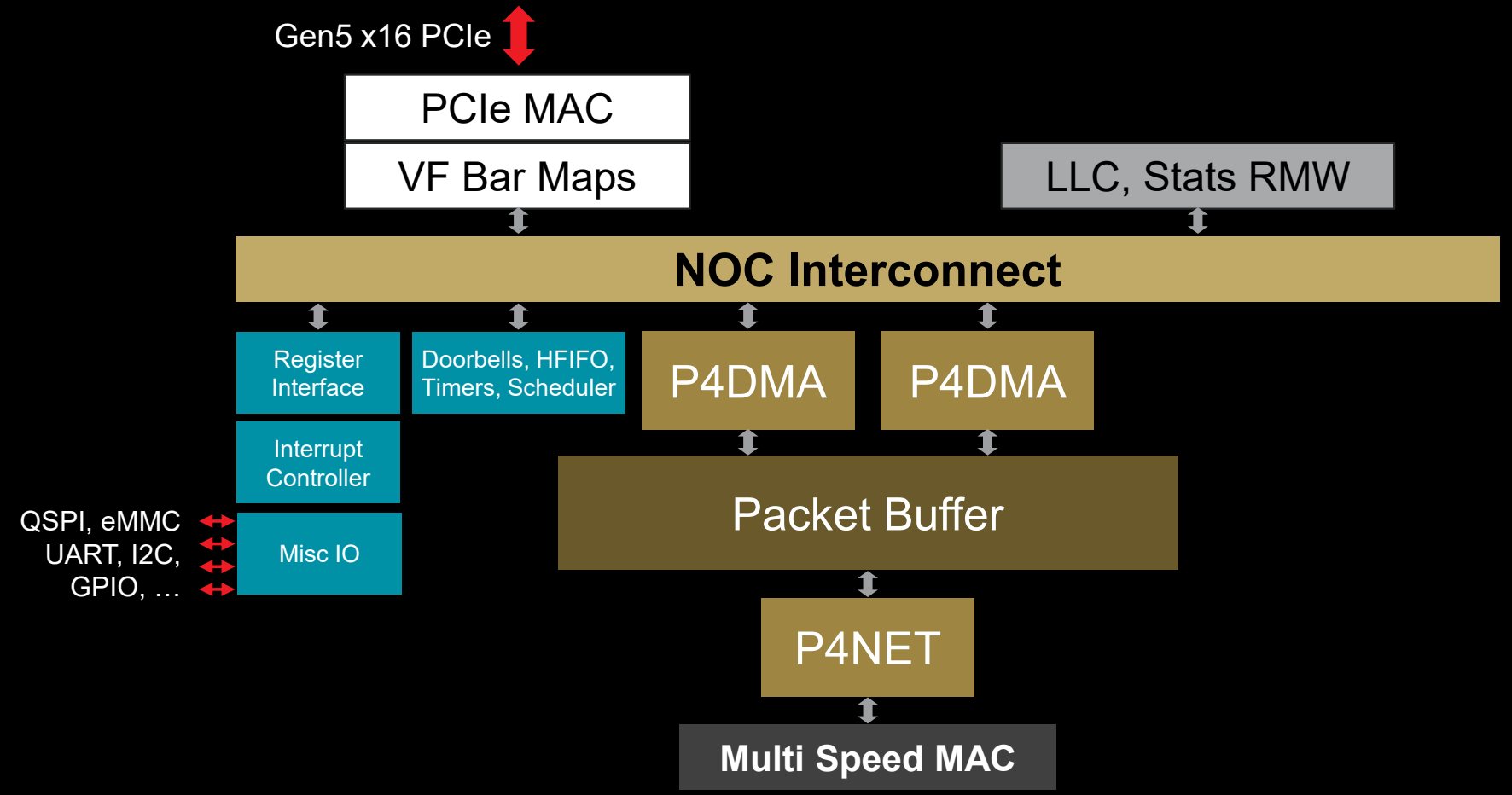
* See Endnotes: PEN-016

AMD

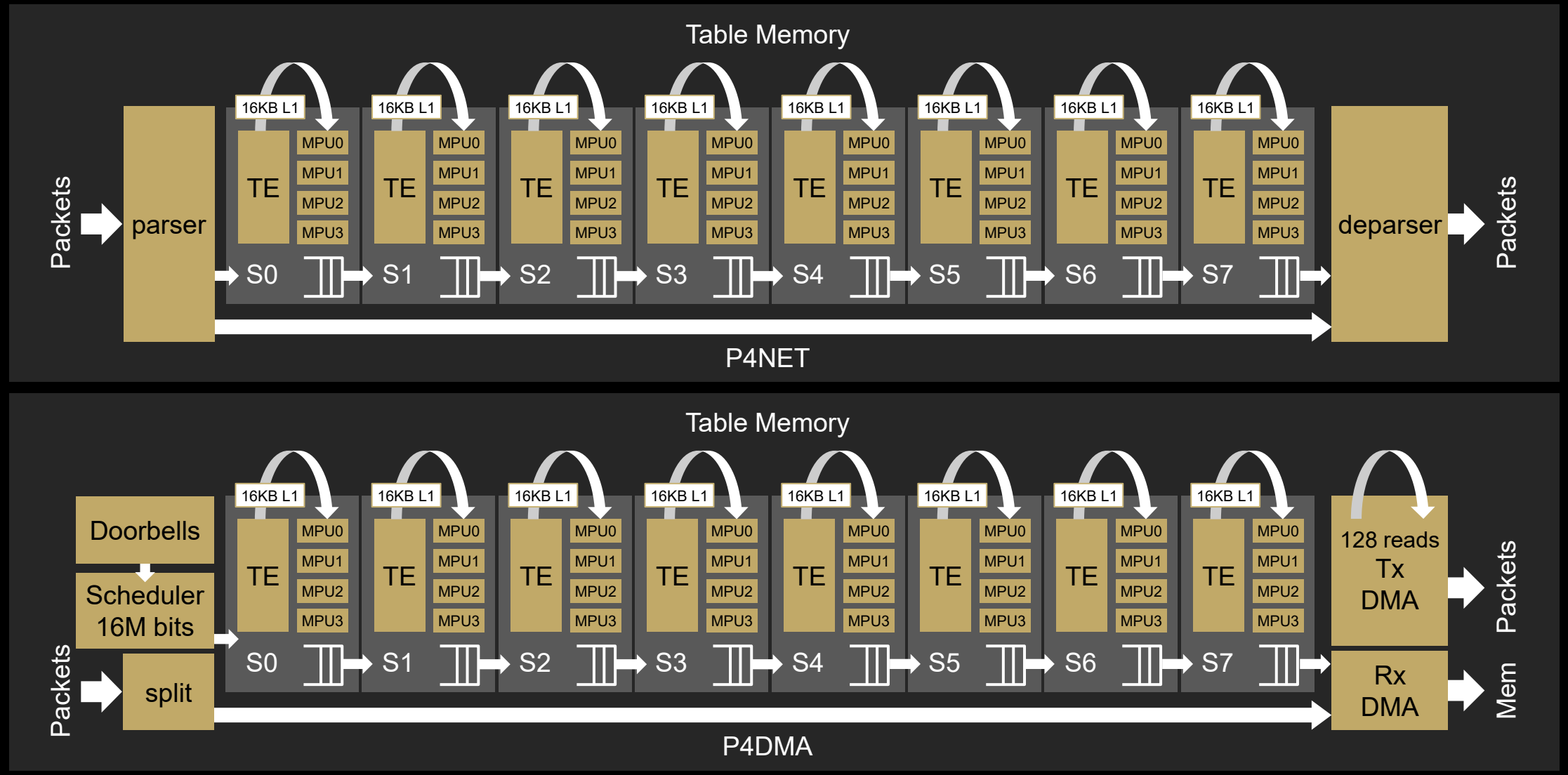
# System Architecture



# Block Diagram



# AMD Pensando™ P4 Architecture






AMD

# P4 Pipeline Components

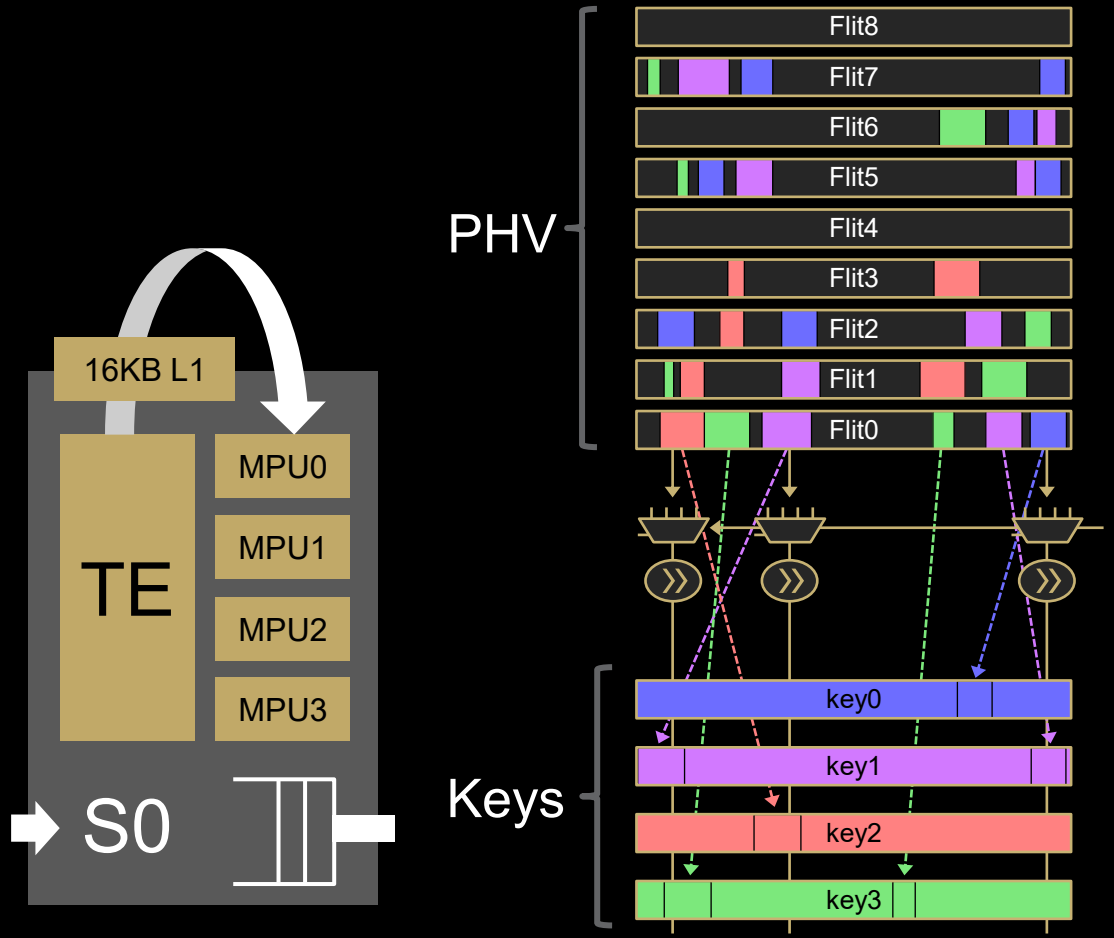
## Table Engine (TE)

### Table Engine

Generates table keys from the packet header vector (PHV); hash or direct

Issues memory reads based on table type

- SRAM – high bandwidth tables of limited scale
- TCAM – data pattern match tables
- Host/GPU memory – direct user space addressing



# P4 Pipeline Components

## Match Processing Unit (MPU)

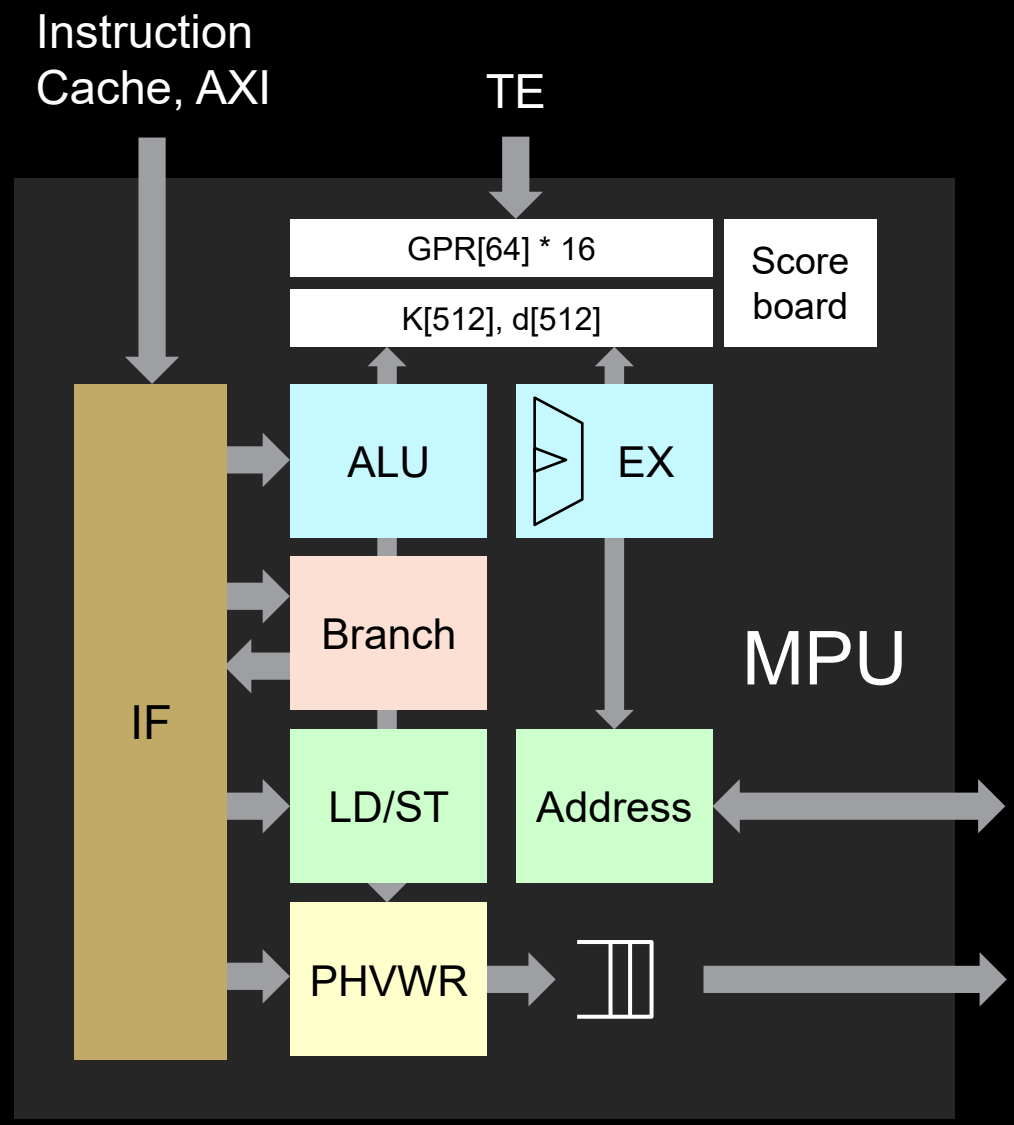
### Domain Specific Processor

Efficient opcodes for field manipulation by compiler

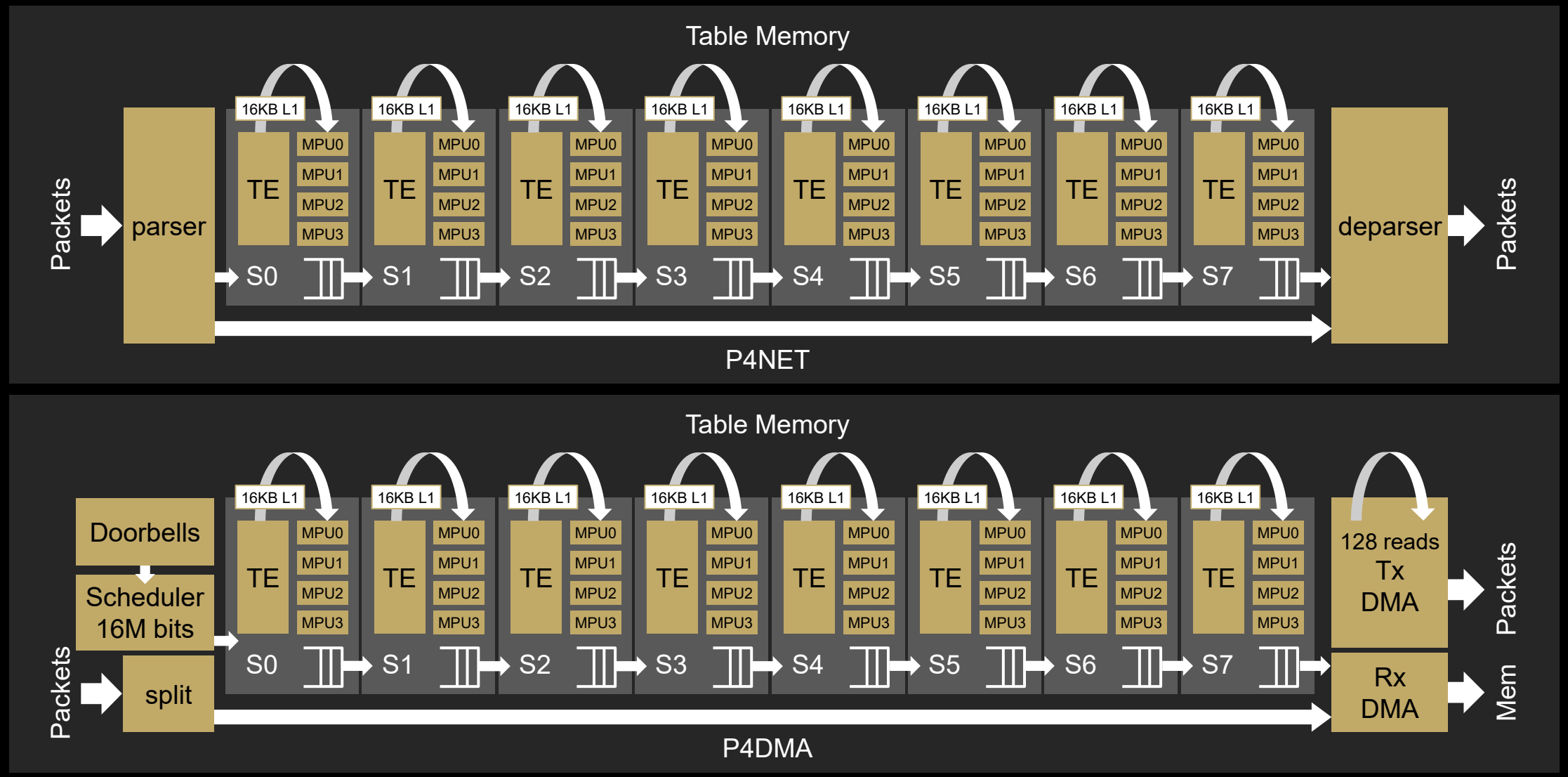
- ALU:  and / or / xor / add / mul / clog2 / rotr/...
- Branch:  beq / ble / blt/br / bcf / cswitch /...
- LD/ST:  ld / st / cpref / tblwr / tbladd /...
- PHV: phvwr / pvhwrp / phvwrm / phvfence /...

### Separate memory, table, and PHV interfaces

- Independent 512-bit accesses
- No collisions between packet, control, and table data



# AMD Pensando™ P4 Architecture





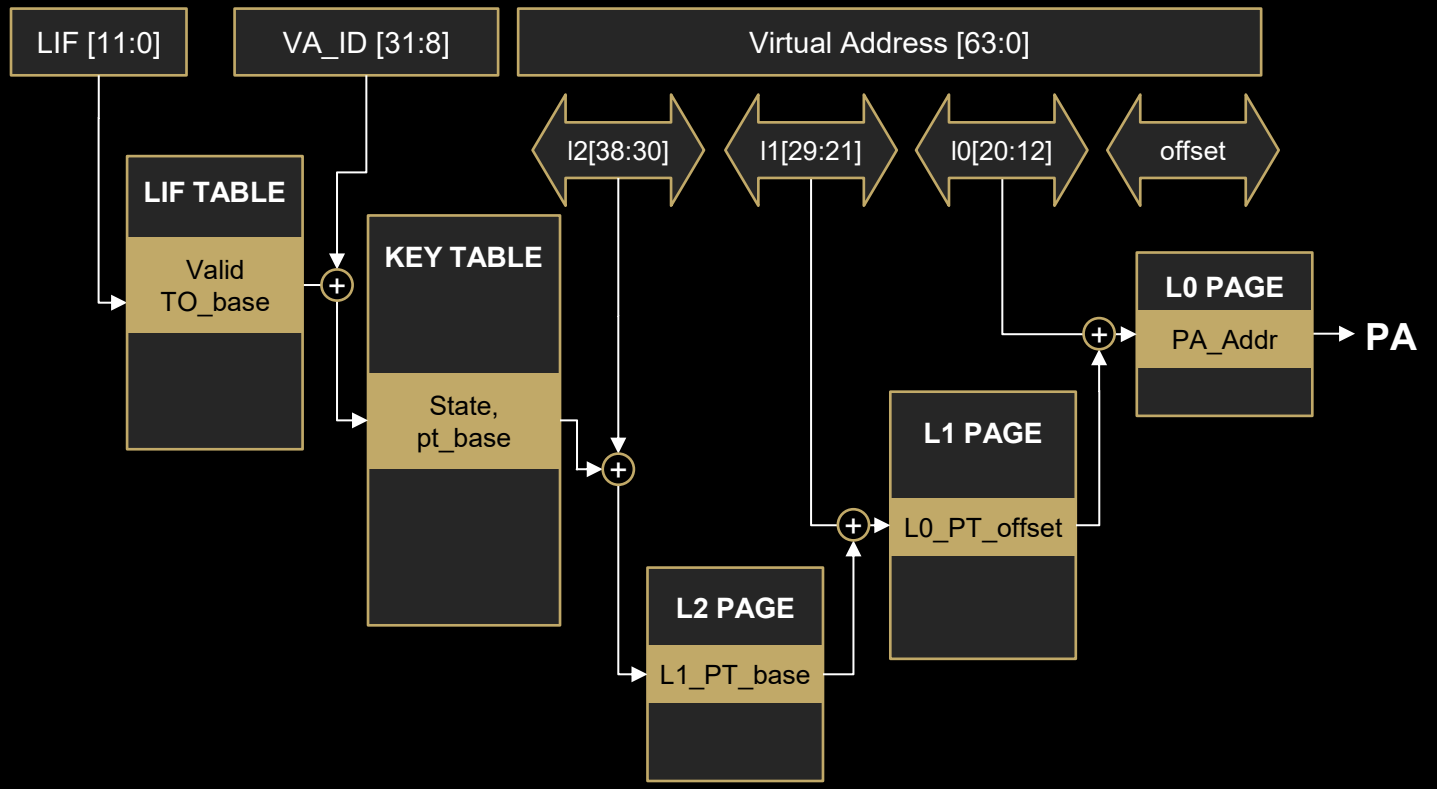

AMD

# Enhancements

## Virtual Address to Physical Address (va2pa)

### Address Translation (va2pa)

- Per logical interface (LIF), per Memory Region (MR) key page tables
- 4K, 2M, 1G pages, 0-5 levels per key
- Software controlled state, registration, mapping

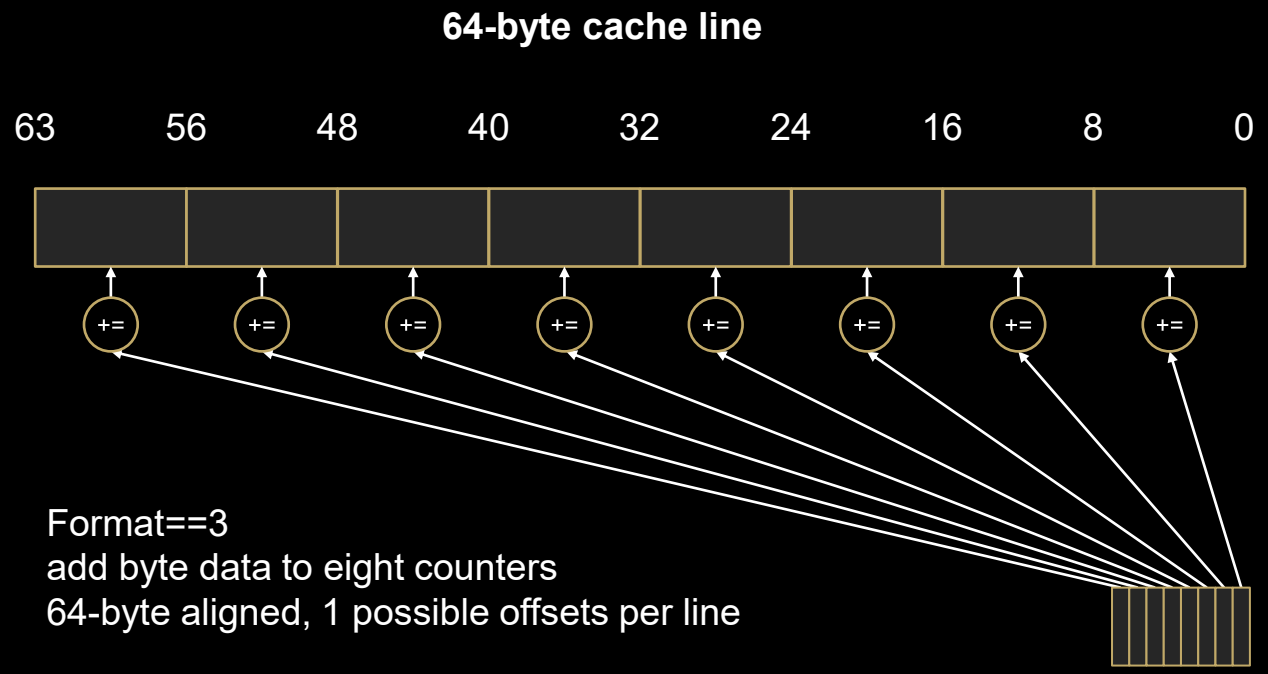


# Enhancements

## Atomic Operations

### Atomic memory operations implemented adjacent to SRAM

- Atomic add, 1 to 8 counters per write using 8/16/32/64-bit operand
- Atomic 64-bit vector set, clear, or read and clear
- Atomic read and increment, read and decrement
- Meter update/token bucket

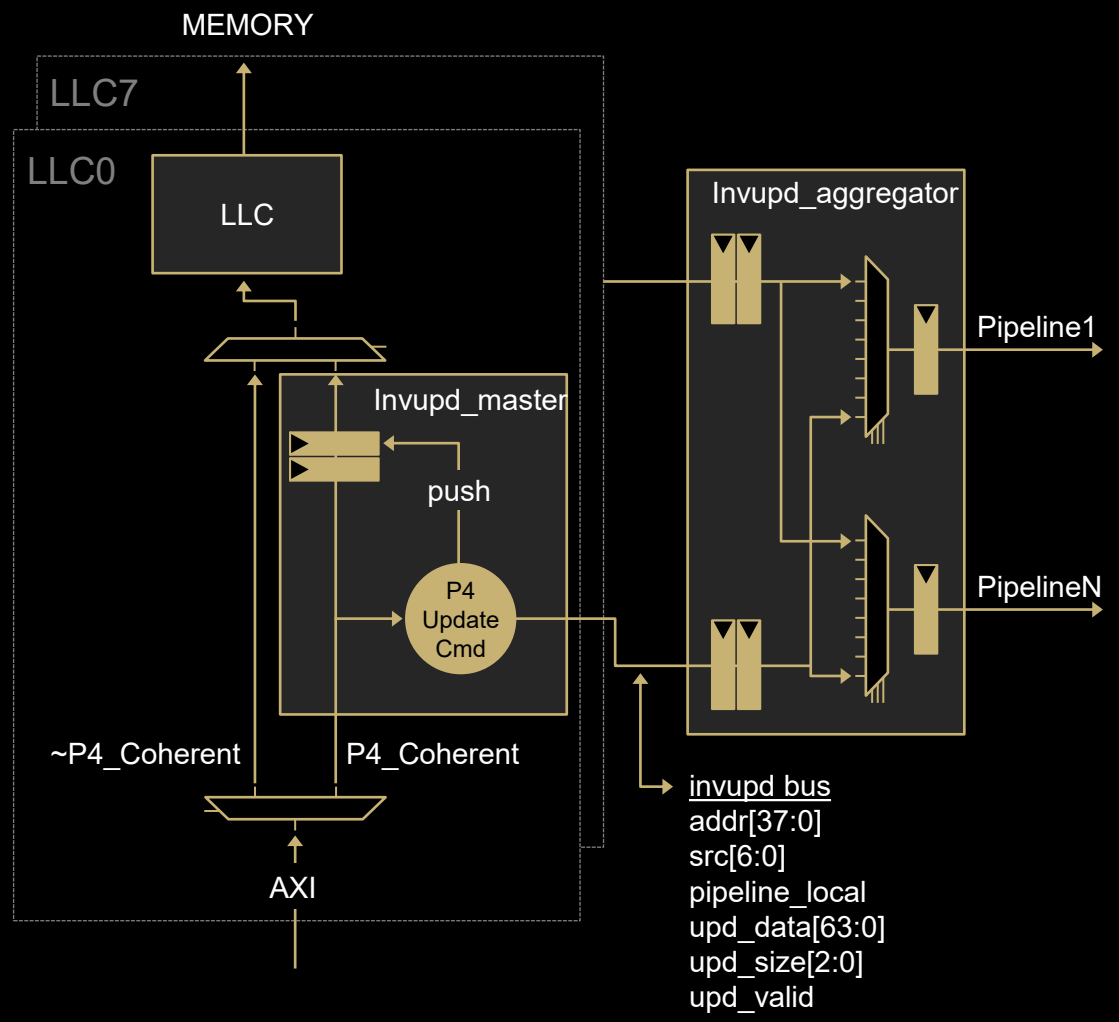


# Enhancements

## Pipeline Cache Coherency

### Pipeline cache coherency handled by invalidate/update logic

- Coherency is maintained by a broadcast invalidate/update network which originates at the central memory interfaces and terminates at each dcache in the system
- Enables P4 coherency on an address range basis
- Direct data update can be supported for common usage cases



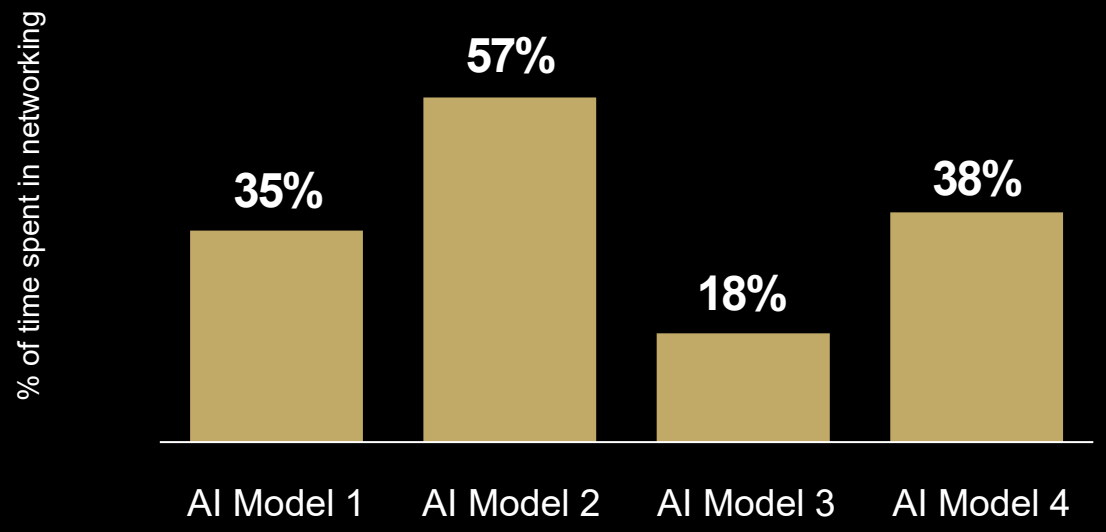
# AI Scale-Out Network

## Scale-out networks drive AI system performance

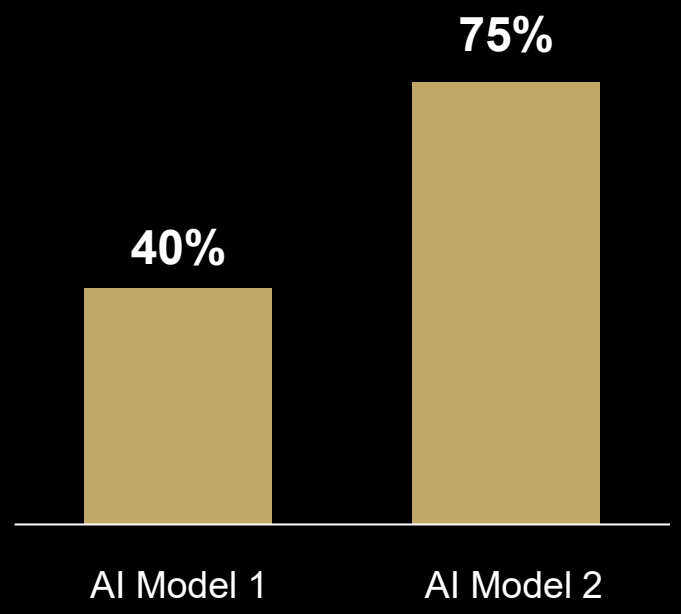
## Challenges

- Poor link utilization from ECMP load balancing
- Network and node congestion
- Network packet loss
- …

# Back-end Networks Drive AI System Performance




At an average 30% of training cycle time is elapsed in waiting for networking[1]




Communication accounts for 40%-75% of time with Training and Distributed Inference Models[2]

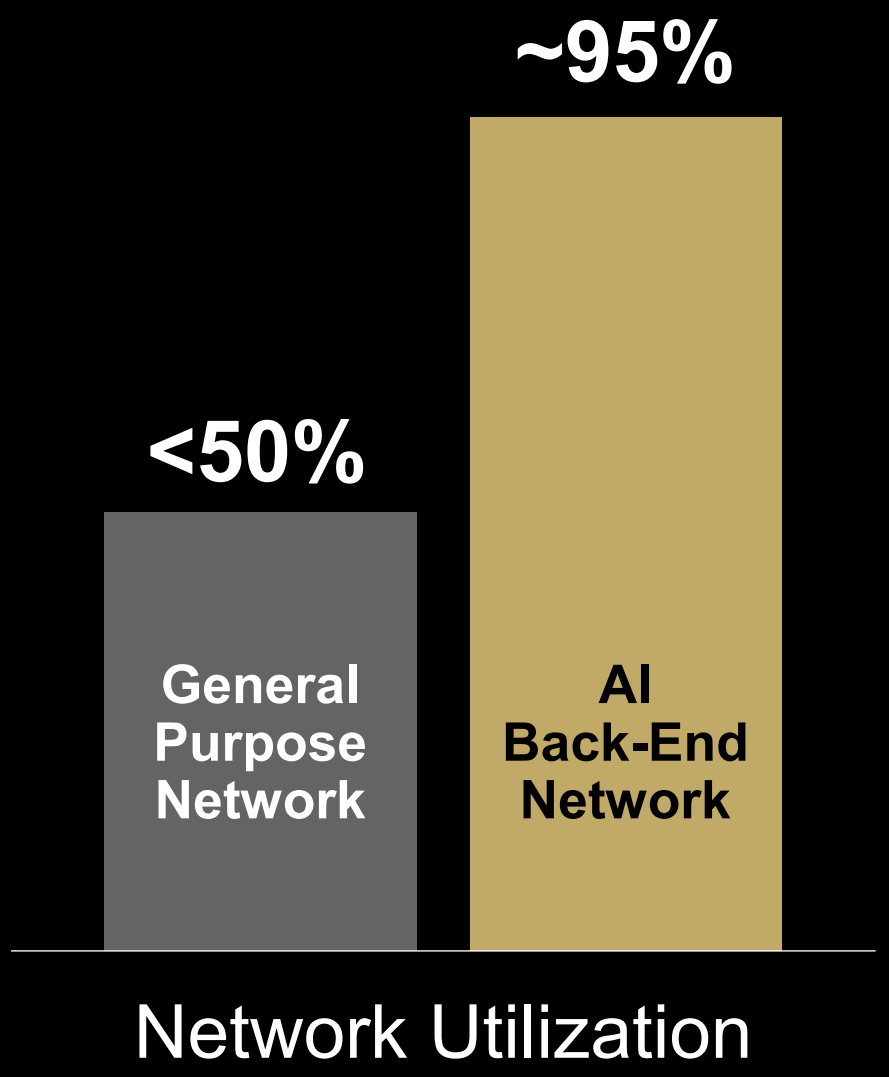


Source: 1) 2022 OCP Keynote by Alexis Bjorlin, VP at Meta,
2) Computation vs. Communication Scaling for Future Transformers on Future Hardware https://arxiv.org/pdf/2302.02825

# The Challenge of High Network Utilization

## AI Backend Networks Drive Sustained Data Transfers

- Intelligent Load Balancing
- Congestion Management
- Fast Failover and Loss Recovery



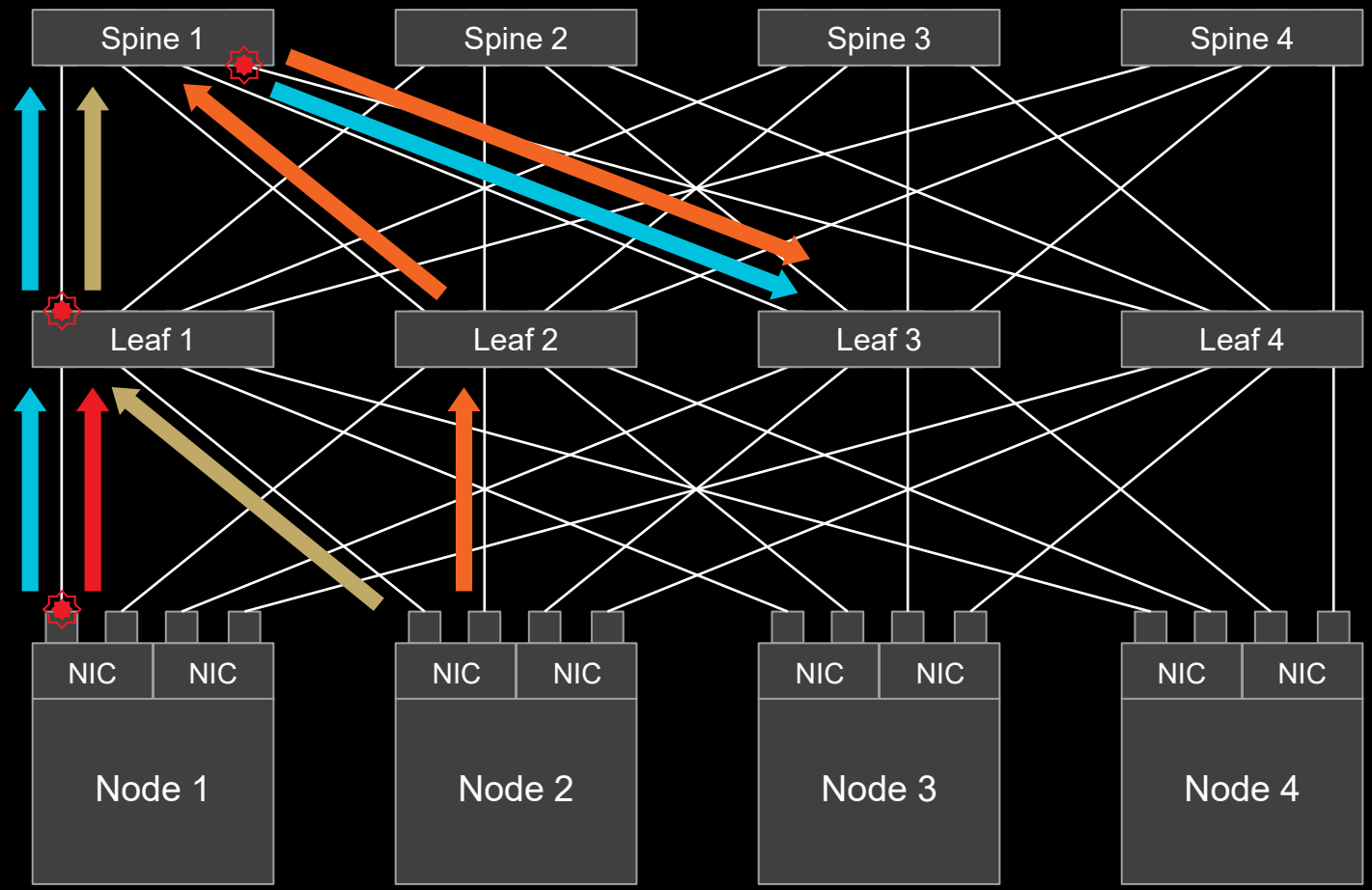



Source: Alibaba Cloud Predictable network: Built for AI Infrastructure

# Challenge: Poor Link Utilization from ECMP Load Balancing

**Challenge:** static ECMP assigns flows to the same network paths, even when other paths have underutilized capacity

**Solution:** implement packet spraying with reordering at the destination



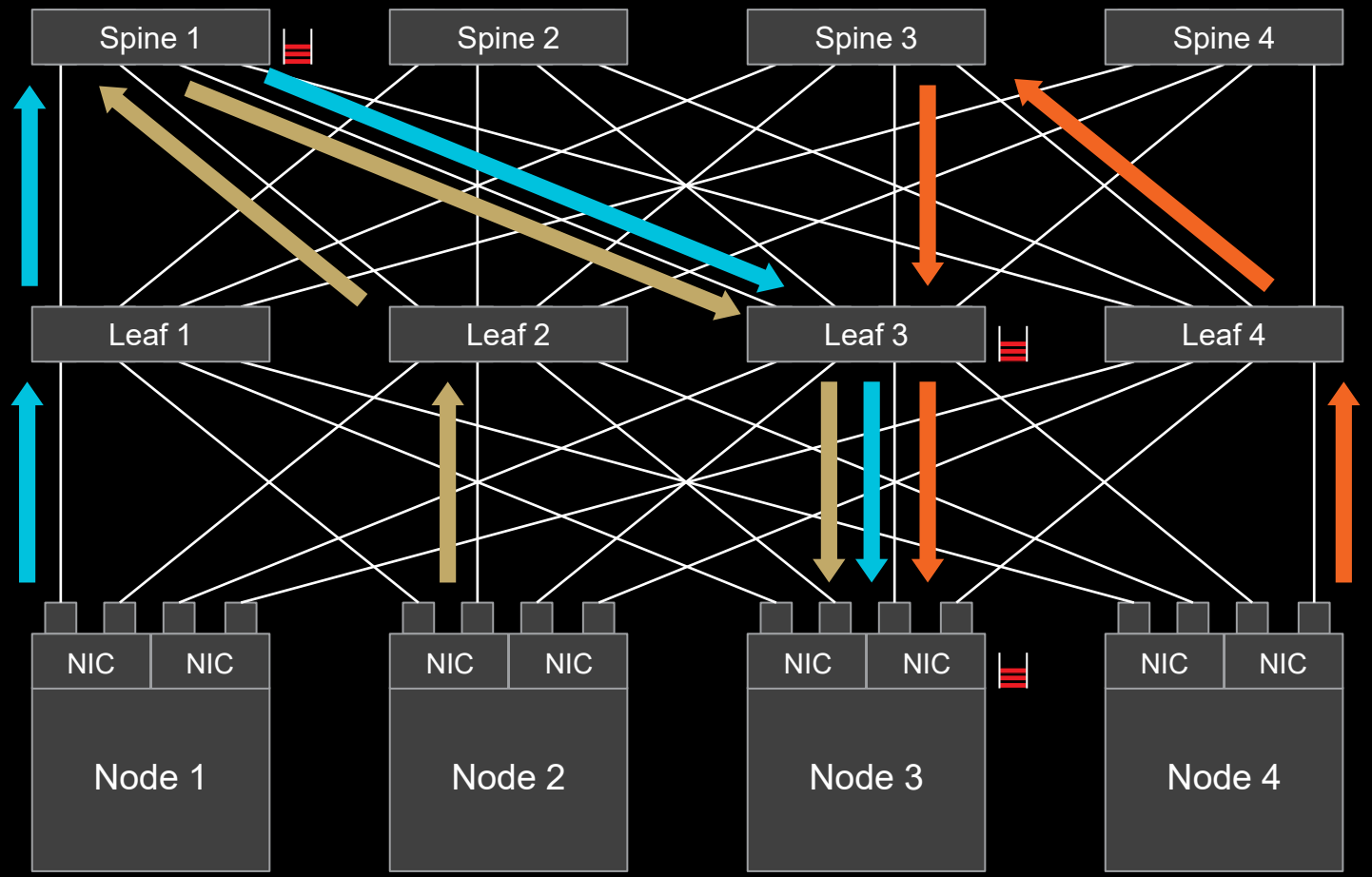



AMD

# Challenge: Network and Node Congestion

**Challenge:** traffic bursts and many-to-one traffic patterns can cause congestion at switches and NICs

**Solution:** apply congestion control using RTT-based feedback to dynamically adjust sender transmission rates



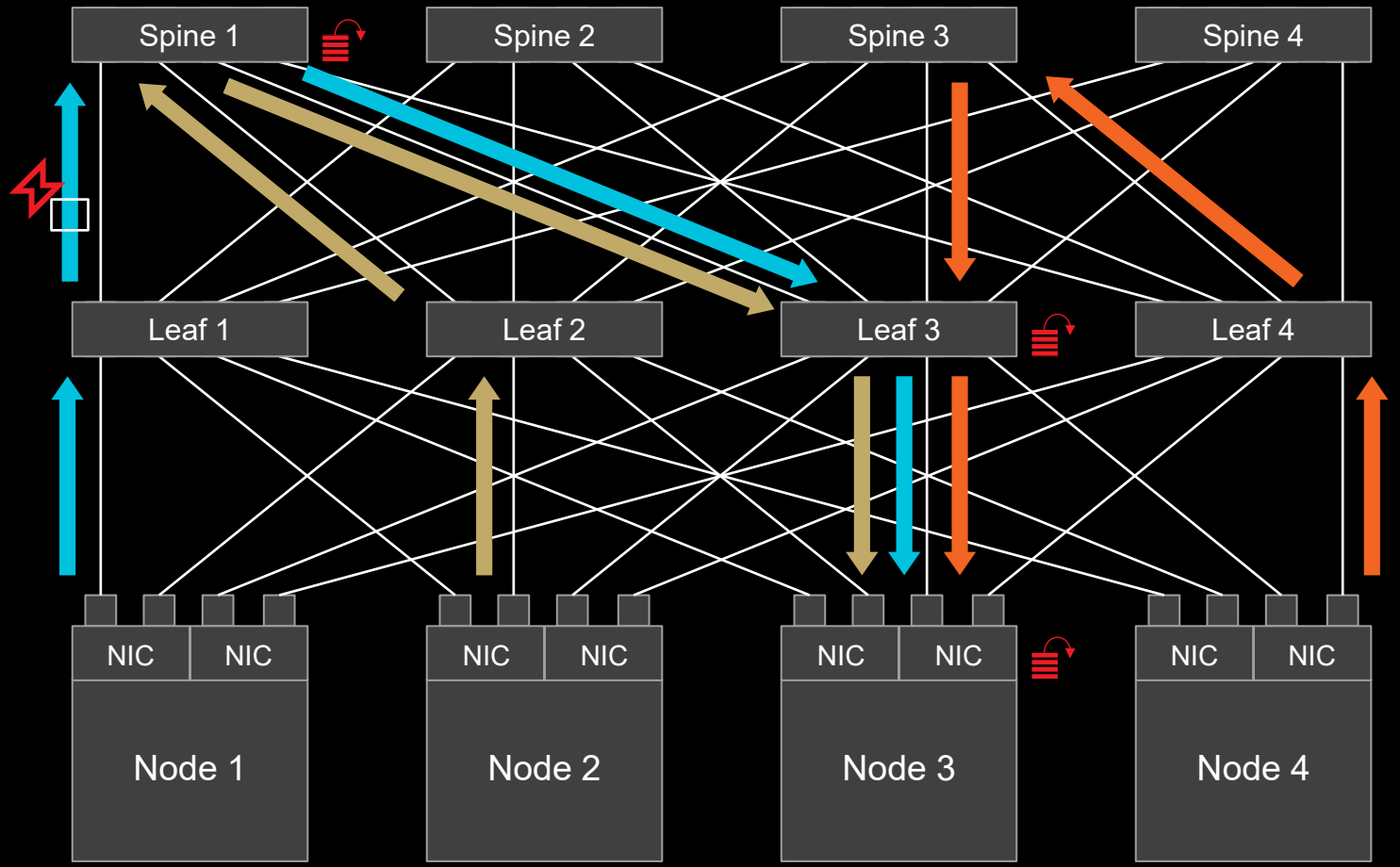



AMD

# Challenge: Network Packet Loss

**Challenge:** packet loss due to buffer overflow or link bit errors can cause a significant amount of retransmissions

**Solution:** use transport protocol with packet retransmission based on selective ACK






AMD


Evolve ethernet as an open, interoperable, high performance, full-communications stack architecture to meet the growing network demands of AI and HPC at scale

UEC 1.0 Specification

Performant | Scalable | Cost Effective



AMD

# AMD Pensando™ Pollara 400 AI NIC is UEC Ready

**Based on RDMA, and UEC AI transport**

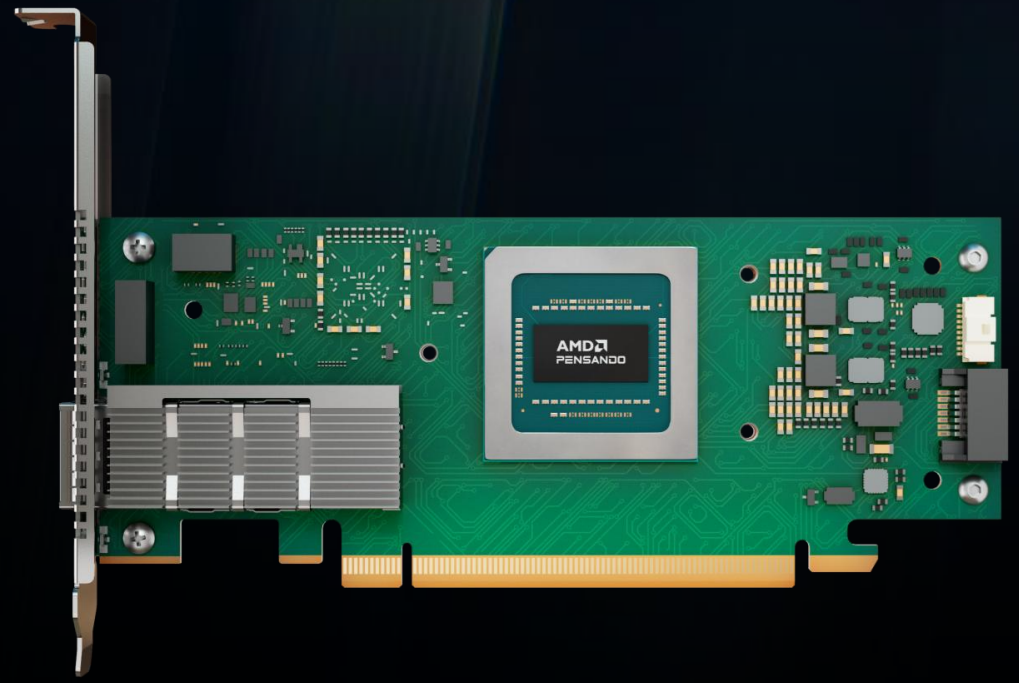
**Multipath Packet-Spray and Out-of-Order Data Reception**

**Programmable Congestion Control**

- Handles various network congestion: round trip delay, congestion marking, trimming
- Works for lossy networks, no PFC needed in the switch

**Fast data-loss recovery via explicit Ack and Selective Ack**

**Extensible for future enhancements**



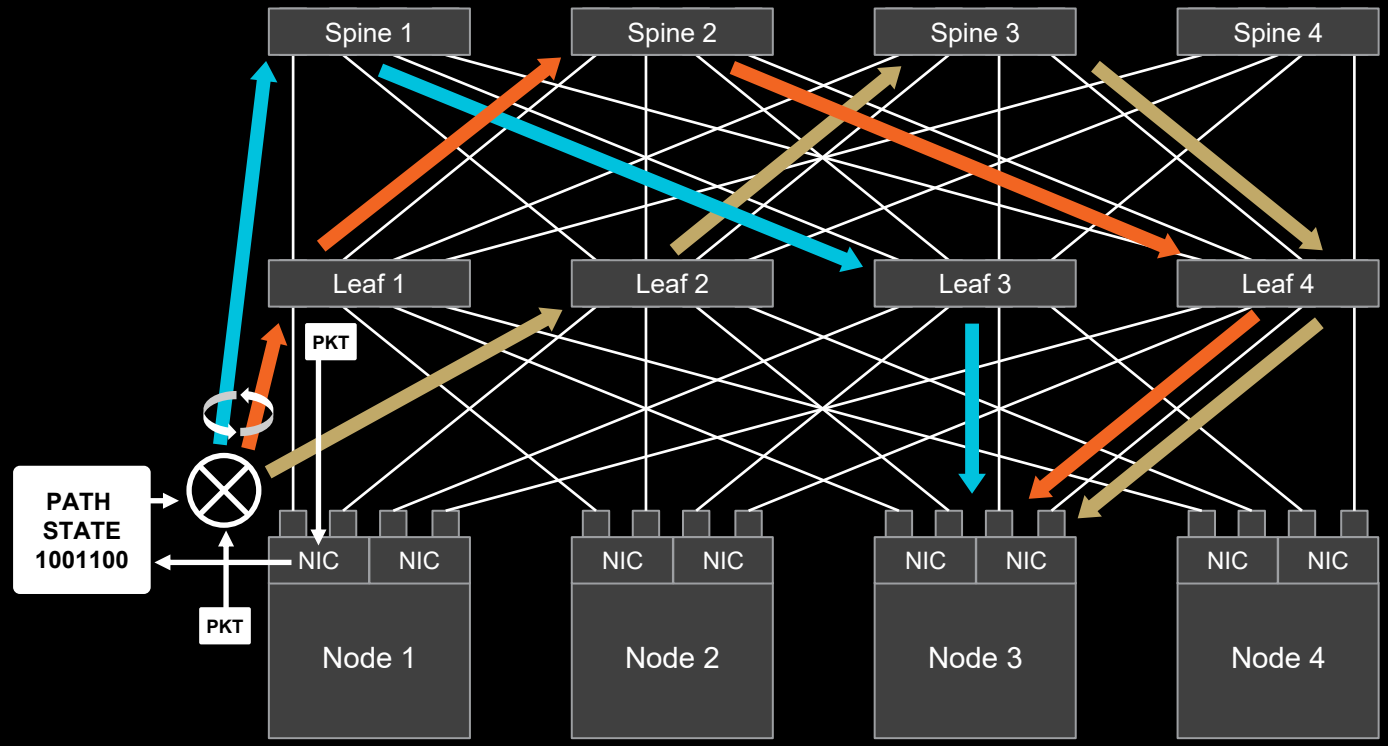

Ultra Ethernet Consortium



AMD

# UEC Multipathing: Use of entropy values for paths

- Packet spray into different paths, reorder completion posting at receiver
- Mark UDP port number/UEC entropy value to control path selection
- Keep track of path state based on ECN, trimmed-packet feedback
- Adjust path usage based on path state

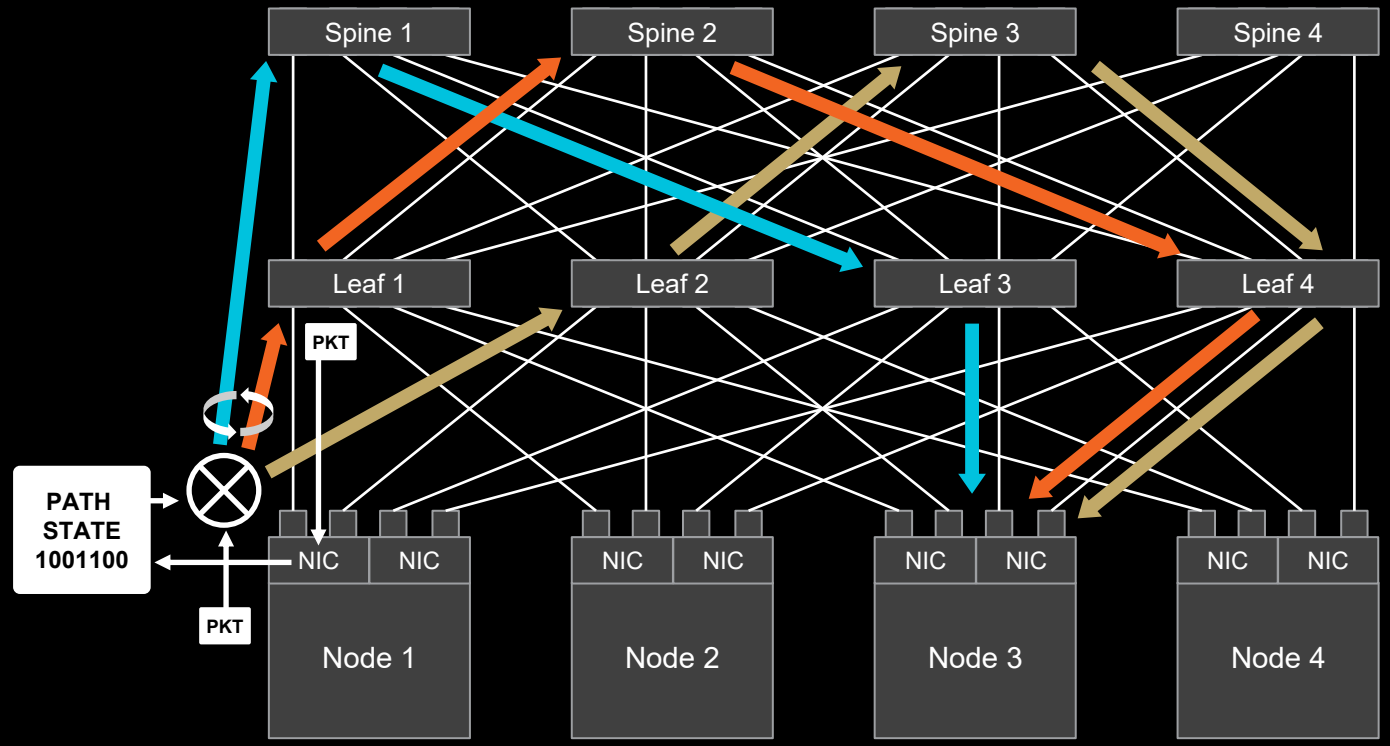


# UEC Multipathing: Use of entropy values for paths

## NIC Transmit Side

```
table rdma_req_tx_puec_cc_tbl {
    key = {
        p_req.p4_txdma_intr.qstate_addr: table_index;
    }
    actions = {
        rdma_req_tx_puec_cc;
    }
}

action rdma_req_tx_puec_cc(
        rdma_puec_path_t sqcb6_path) {
    ...
    bit<64> ev_to_use =
            rdma_puec_ev_search(sqcb6_path,
                                sqcb6_path.ev_next);
    p_req.cc.puec.udp_sport = sqcb6_path.ev_base +
                              (bit<16>) ev_to_use;
    sqcb6_path.ev_next = ev_to_use + 1;
    ...
}
```





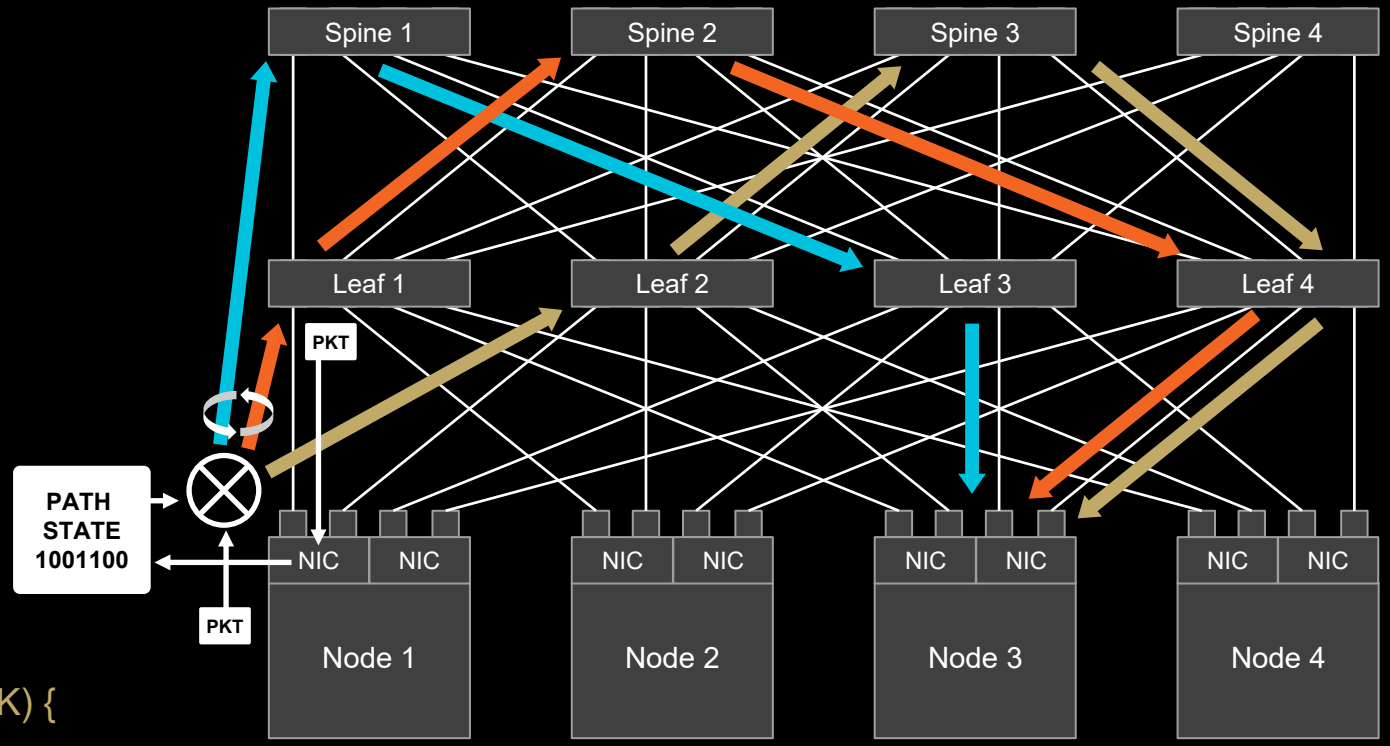

AMD

# UEC Multipathing: Use of entropy values for paths

## NIC Received Side

```
table rdma_req_rx_puec_path_update_tbl {
    key = {
        p.p4_rxdma_intr.qstate_addr: table_index;
    }
    actions = {
        rdma_req_rx_puec_path_update;
    }
}

action rdma_req_rx_puec_path_update(
        rdma_puec_path_t sqcb6_path) {
    ...
    rdma_puec_path_index_t ev =
        (rdma_puec_path_index_t) (p.seth_entropy -
                                  sqcb6_path.ev_base);
    if (p.bth_opcode == RDMA_PKT_OPC_PUEC_NACK) {
        rdma_puec_ev_skip(sqcb6_path, ev);
    }
    ...
}
```






AMD

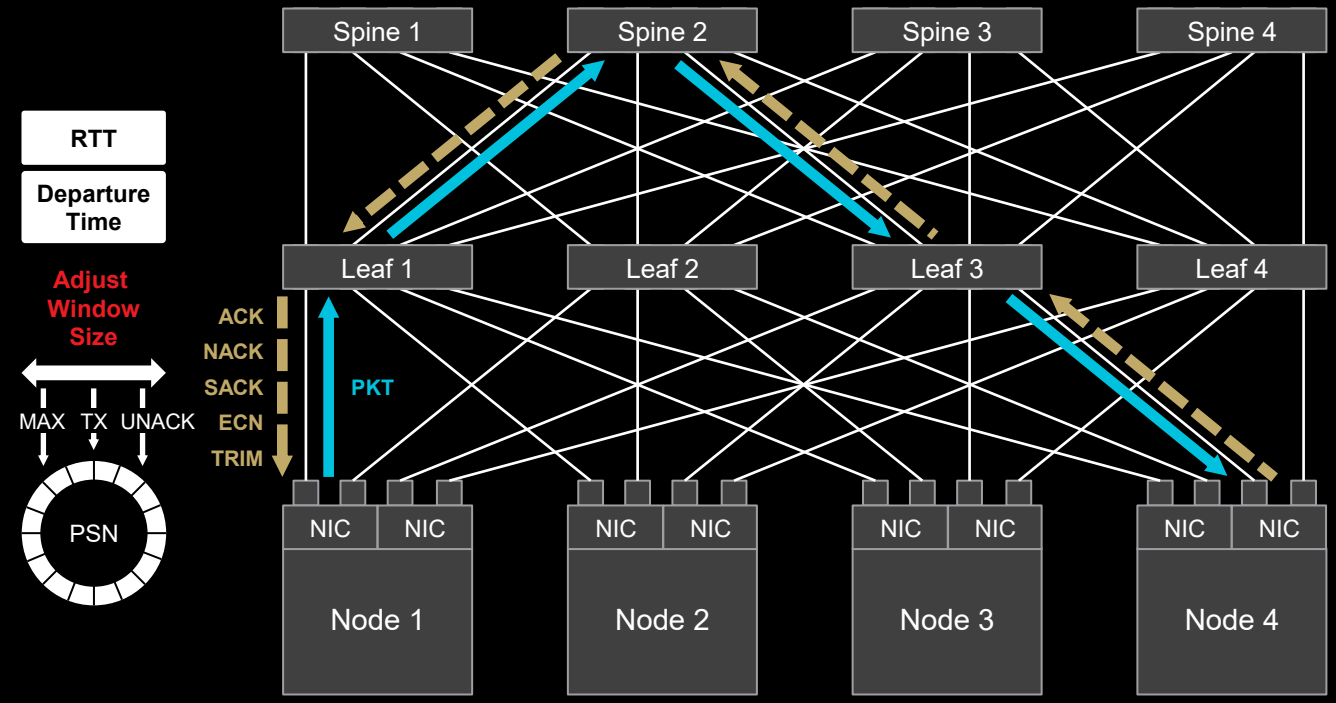
# UEC Congestion Control: Overview based on NSCC

**Transmitter Based**

**Measures RTT**

**Adjusts Transmission Window based on**

- Round Trip Time
- ACK/NACK/SACK
- ECN
- Trimmed Packets

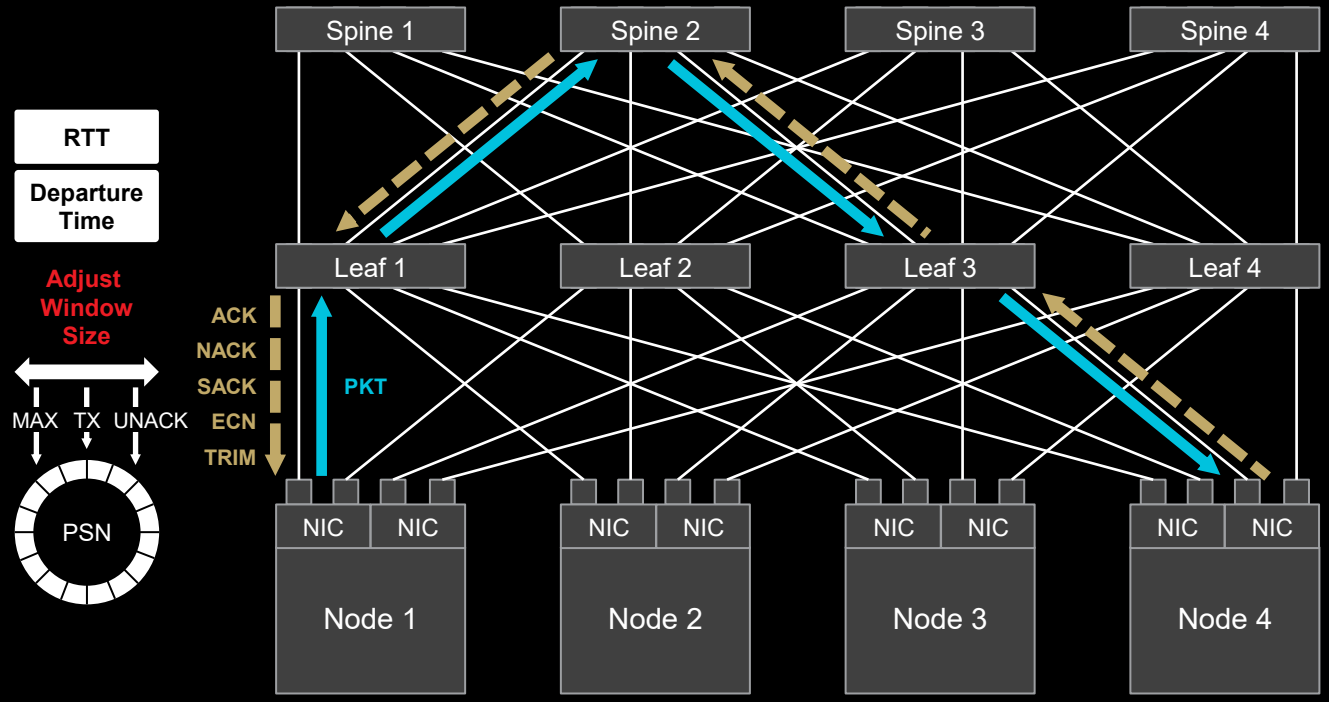


# UEC Congestion Control: Overview based on NSCC

## NIC Transmit Side

```
table rdma_req_tx_tbl {
    ref_common_name = "rdma_req_tx_tbl";
    actions = {
        rdma_req_tx;
    }
}

action rdma_req_tx(rdma_sqcb_t sqcb) {
    ...
    cc_stop = __circular_lt24(sqcb.lsn, sqcb.ssn);
    if (__unlikely(cc_stop)) {
        _sq_sched_disable();
        _sq_ack_timer_start(sqcb);
    }
    ...
}
```

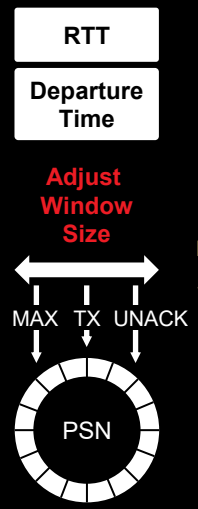


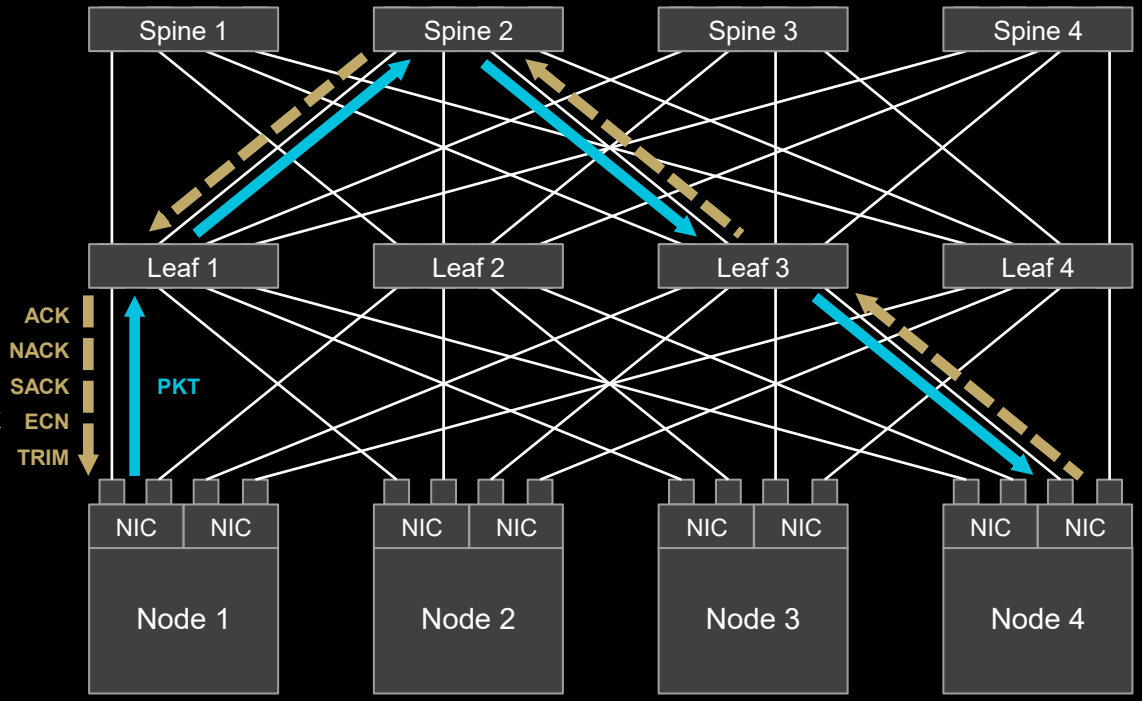
# UEC Congestion Control: Overview based on NSCC

## NIC Received Side

```
table rdma_req_rx_puec_congestion_tbl {
    key = {
        p.p4_rxdma_intr.qstate_addr: table_index;
    }
    actions = {
        rdma_req_rx_puec_congestion;
    }
}

action rdma_req_rx_puec_congestion(rdma_sqcb_t sqcb_cc) {
    ...
if (!(bool) p.ecn_marked) {
        if (over_delay_target) {
            _fair_increase(sqcb_cc, p.newly_acked_bytes);
        } else {
            _proportional_increase(sqcb_cc,
                                   p.newly_acked_bytes,
                                   delay, max_cwnd);
        }
    } else {
        if (over_delay_target) {
            _multiplicative_decrease(sqcb_cc);
        }
    }
}
```



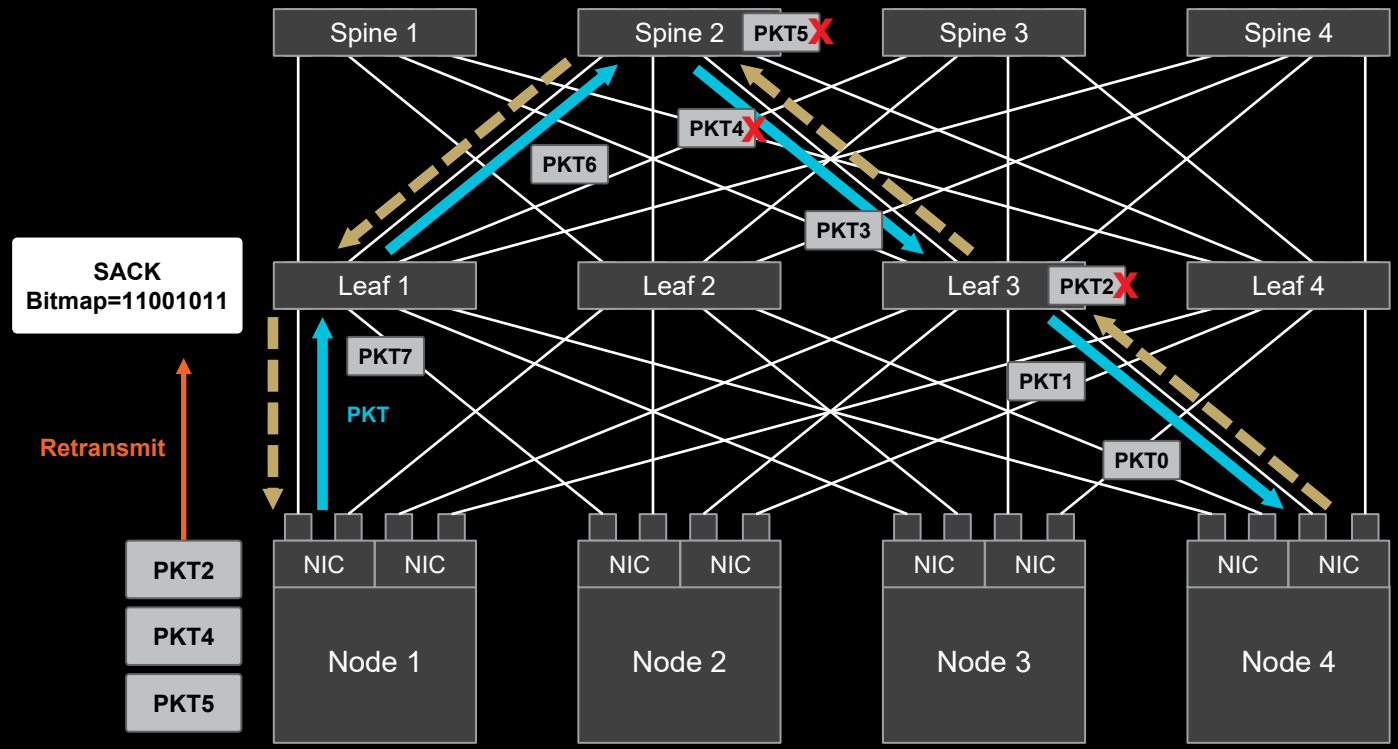



# UEC Efficient Loss Retransmission: Selective Ack

## SACK triggered when

- Total bytes rec > threshold
- ECN/trimmed packets

## Sender re-transmit missing packets based on SACK bitmap



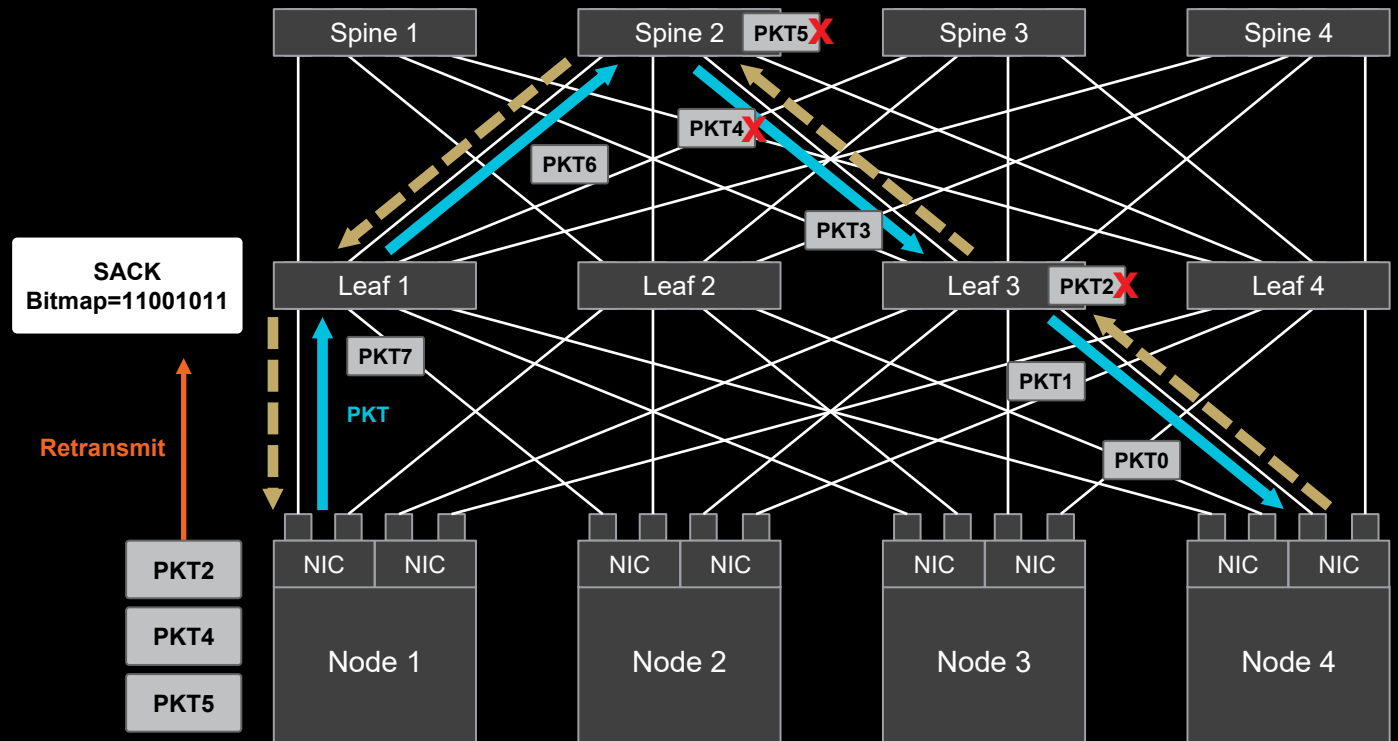



AMD

# UEC Efficient Loss Retransmission: Selective Ack

## NIC Responder Receive Side

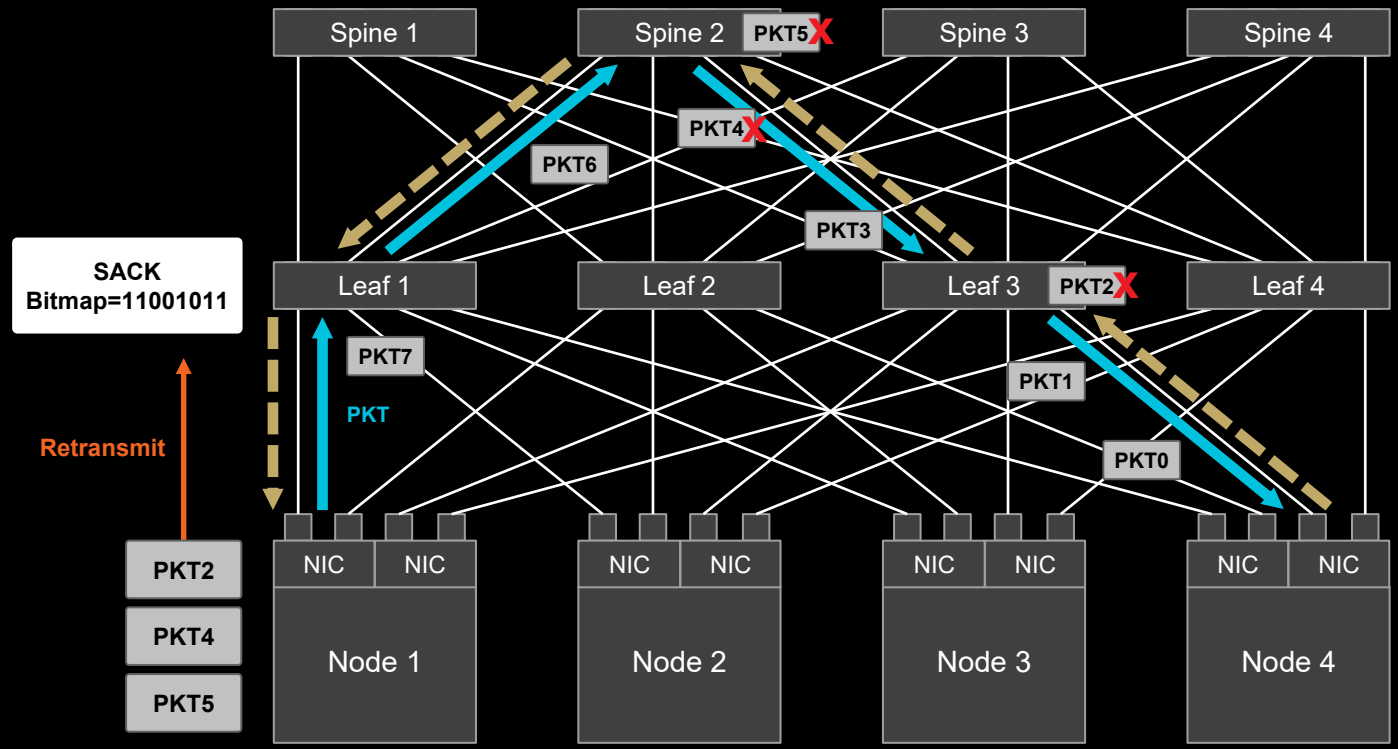
```
table rdma_resp_rx_puec_ack_tbl {
    key = {
        p.p4_rxdma_intr.qstate_addr : table_index;
    }
    actions = {
        rdma_resp_rx_puec_ack;
    }
}
action rdma_resp_rx_puec_ack(rdma_rqcb2_t rqcb2) {
    ...
    rqcb2.sack_mark = p_puec.ecn == 1 ?
                      RDMA_PUEC_SETH_M_SKIP :
                      RDMA_PUEC_SETH_M_NONE;
    rqcb2.sack_psn_offset =
            (bit<16>)(p_puec.lowest_unsacked_psn -
                      p.ack_info.psn + 1);
    rqcb2.sack_bmp =
            __table_read<bit<64>>((reg_t)p.sack_rel_psn_bit)
    rqcb2.ack_nak_psn = p.ack_info.psn - 1;
    rqcb2.rcv_bytes = p_puec.puec_tot_rcv_bytes;
    // Ring the doorbell via DMA
    p.flags.dbell = 1;
}
```



# UEC Efficient Loss Retransmission: Selective Ack

## NIC Requester Receive Side

```
table rdma_req_rx_puec_sack_bitmap_tbl {
    key = {
        p.p4_rxdma_intr.qstate_addr: table_index;
    }
    actions = {
        rdma_req_rx_puec_sack_bitmap;
    }
}
action rdma_req_rx_puec_sack_bitmap(
          rdma_puec_req_sack_t sqcb7_sack) {
    ...
    bool holes;
    _check_for_holes(sqcb7_sack,
                     sqcb7_sack.retx_upper_bound,
                     retx_upper_bound, holes);
    if (holes) {
        bit<24> too_recent = 2 * p.cwnd >>
                             sqcb7_sack.log_pmtu;
        bit<24> retx_upper_bound = p.tx_psn - too_recent;
        sqcb7_sack.retx_upper_bound = retx_upper_bound;
        p.ring_bt_doorbell = 1;
    }
}
```






AMD

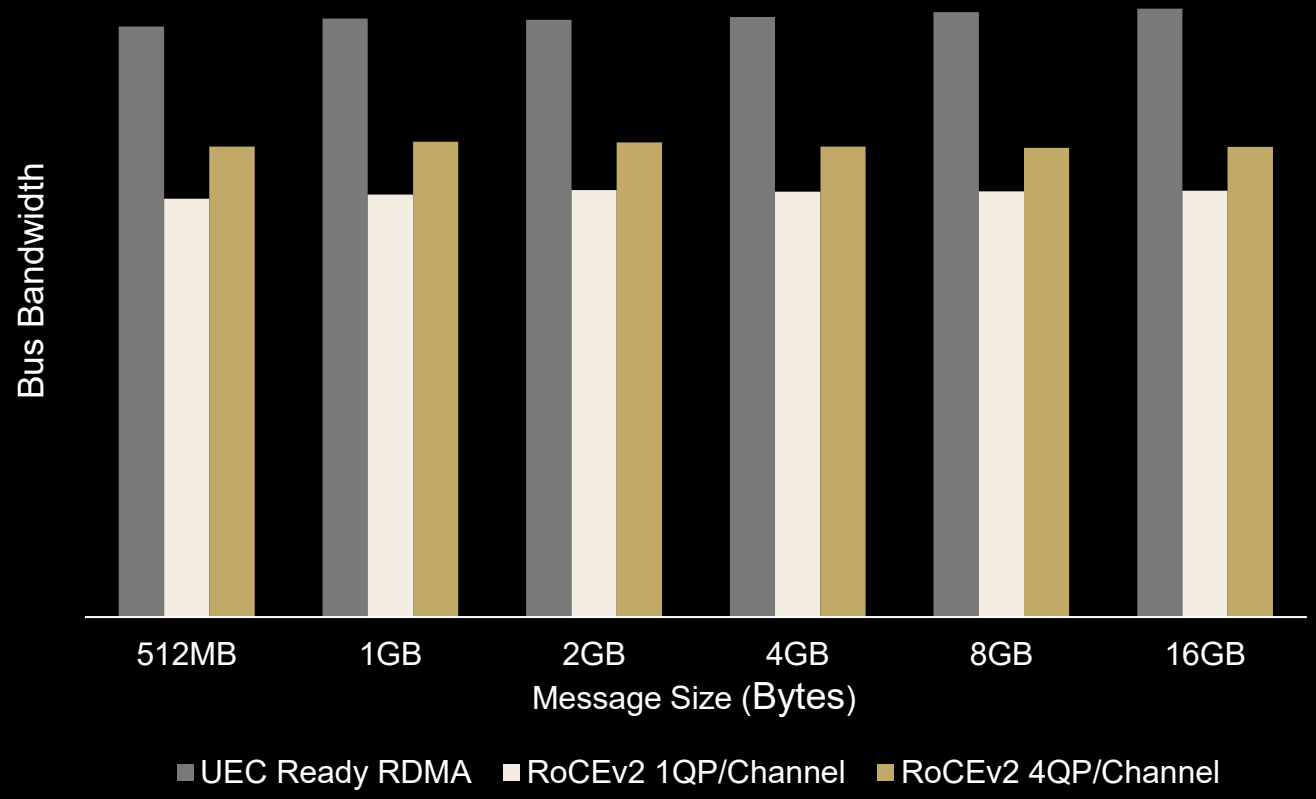
# RCCL All Reduce Performance: Pollara RoCEv2 vs. Pollara UEC-ready RDMA

## Up to 40% Gain in Performance with UEC Ready RDMA

**Higher network utilization with load balancing**

**1.25x** Performance Gain for UEC Ready RDMA vs RoCEv2 4 Qpairs*

**1.4x** Performance Gain for UEC Ready RDMA vs RoCEv2 1 Qpair*






* See Endnotes: PEN-016

AMD

# Summary

## Pollara 400 AI NIC and P4 Architecture

## Enhancements from Previous P4 Generations

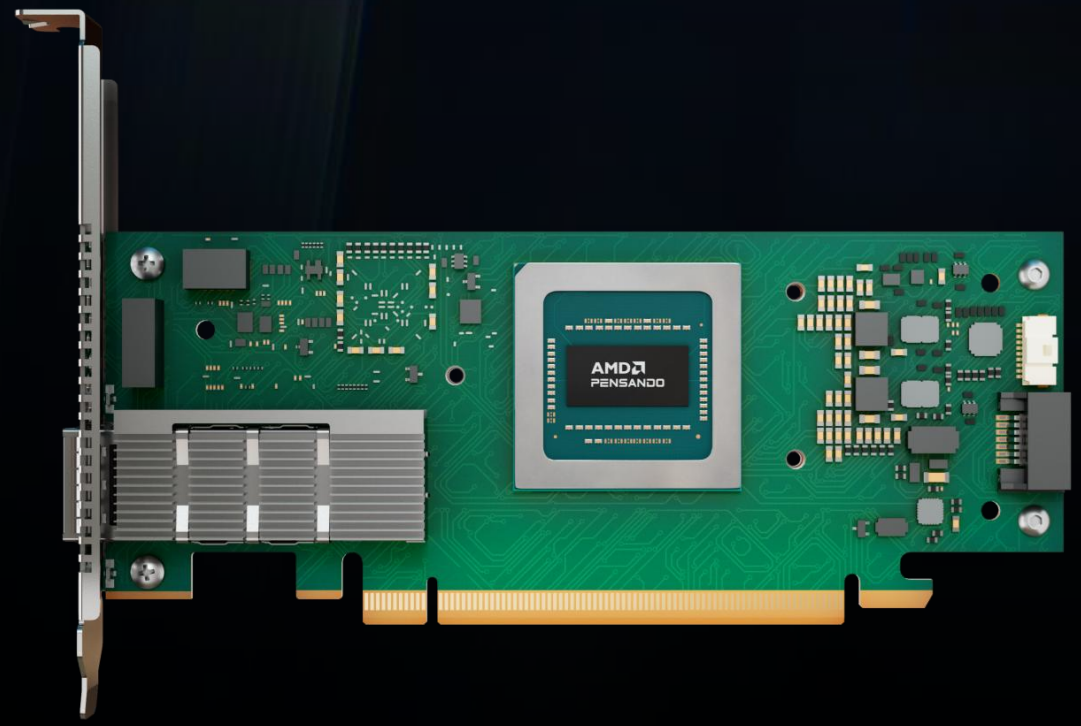
- Address translation
- Atomic operations
- Pipeline cache coherency

## AI Scale-Out network challenges and solutions

- Adaptive packet spray
- Path aware congestion avoidance
- Explicit loss notification and selective retransmission

## AMD Pensando™ Pollara is the First Ultra Ethernet Consortium ready AI NIC

- Resolves AI network challenges by leveraging UEC with programmability



# Endnotes

PEN-016 - Testing conducted by AMD Performance Labs as of [28th April 2025] on the [AMD Pensando™ Pollara 400 AI NIC ], on a production system comprising of: 2 Nodes of 8xMI300X AMD GPUs (16 GPUs): Broadcom Tomahawk-4 based leaf switch (64x400G) from MICAS network; CLOS Topology; AMD Pensando Pollara AI NIC – 16 NICs; CPU Model in each of the 2 nodes - Dual socket 5th gen Intel® Xeon® 8568 - 48 core CPU with PCIe® Gen-5 BIOS version 1.3.6 ; Mitigation - Off (default); System profile setting - Performance (default) SMT- enabled (default); Operating System Ubuntu 22.04.5 LTS, Kernel 5.15.0-139-generic. Following operation were measured: Allreduce; Average 25% for All-Reduce operations with 4QP and using UEC ready RDMA vs the RoCEv2 for multiple different message size samples (512MB, 1GB, 2GB, 4GB, 8GB, 16GB). The results are based on the average at least 8 test runs.

# Disclaimer and Attribution

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18u.


AMD