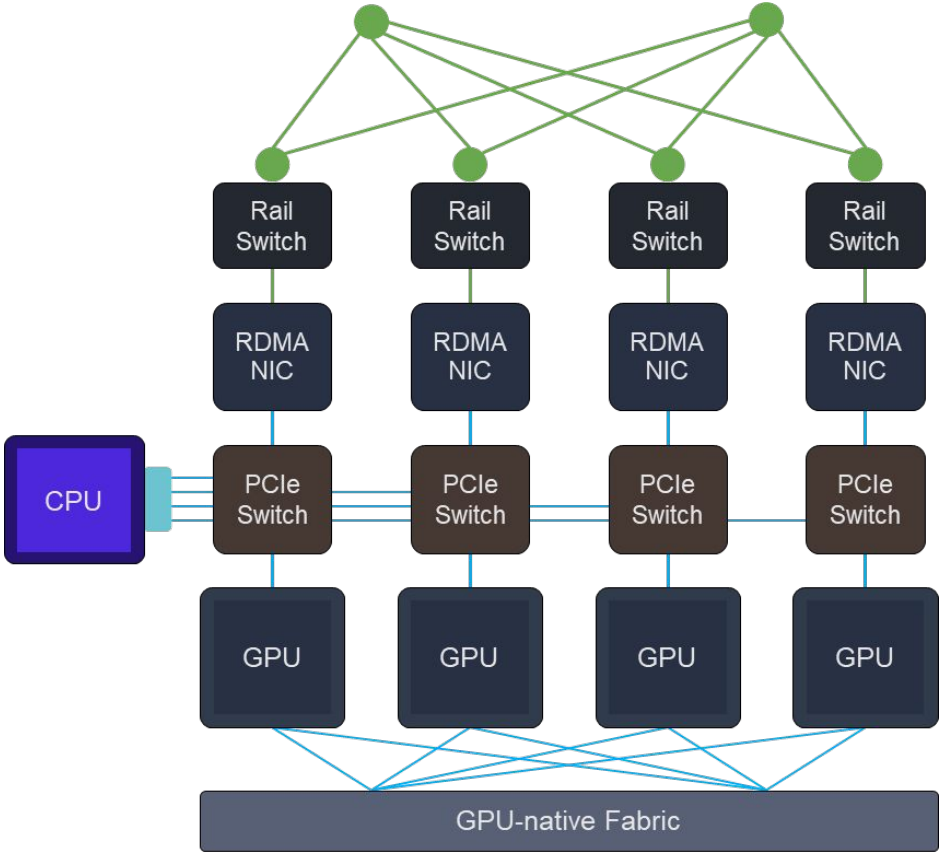
AI Networking workshop

Shrijeet Mukherjee
Nikolay Aleksandrov

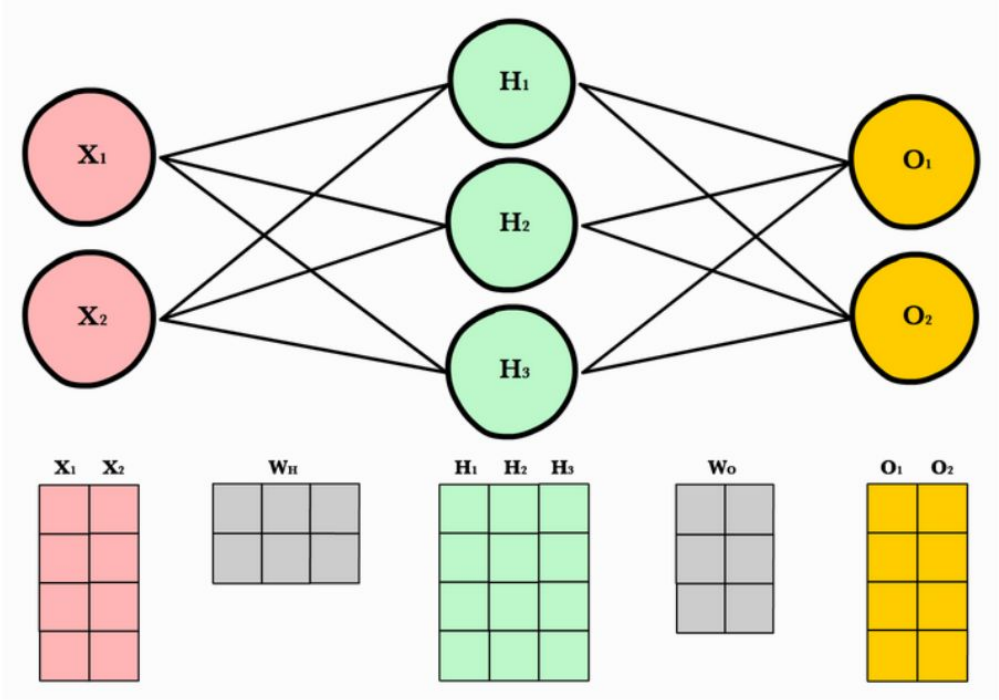
Costin Raiciu

Why the hullabaloo over ML and networking



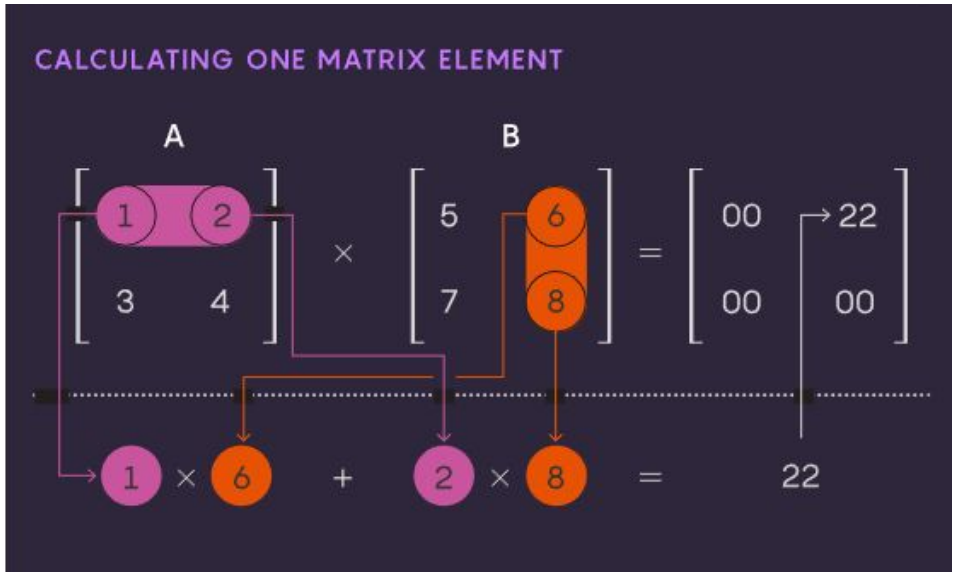
- HPC networking has existed for decades
  - Relatively a small targeted market with strict latency and overhead requirements
- Data Center (and Hyperscalar) networking had a different focus
  - Resiliency
  - Aggregate bandwidth over wide compute area
  - Each flow/compute task was small and fit in a node
- ML changed that to
  - Single fault domain compute with scale-out and scale-up needs

So what is being communicated



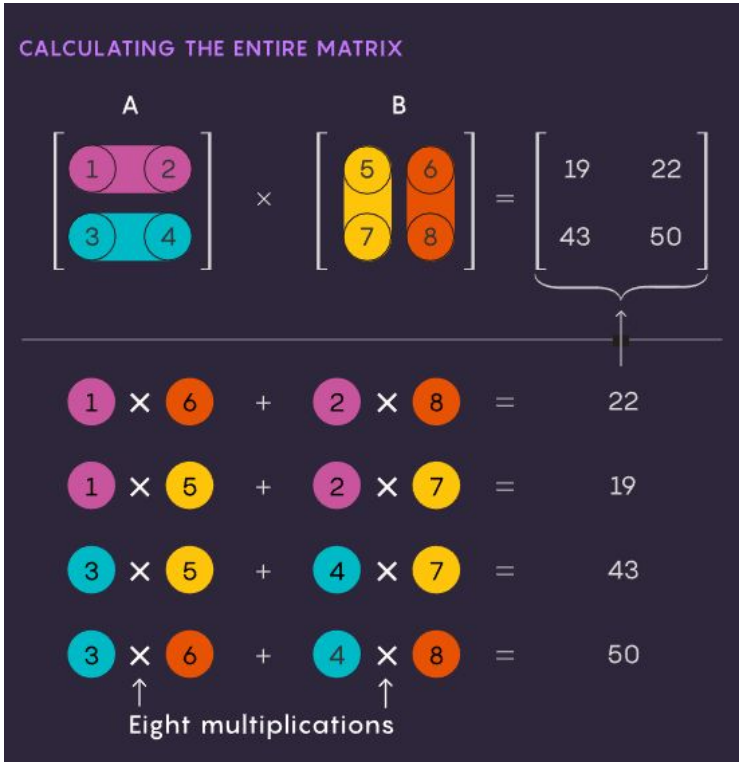
https://www.linkedin.com/pulse/explaining-multilayer-perceptrons-terms-general-matrix-ajit-jaokar-c5aje/

Matrix math : why we need to communicate



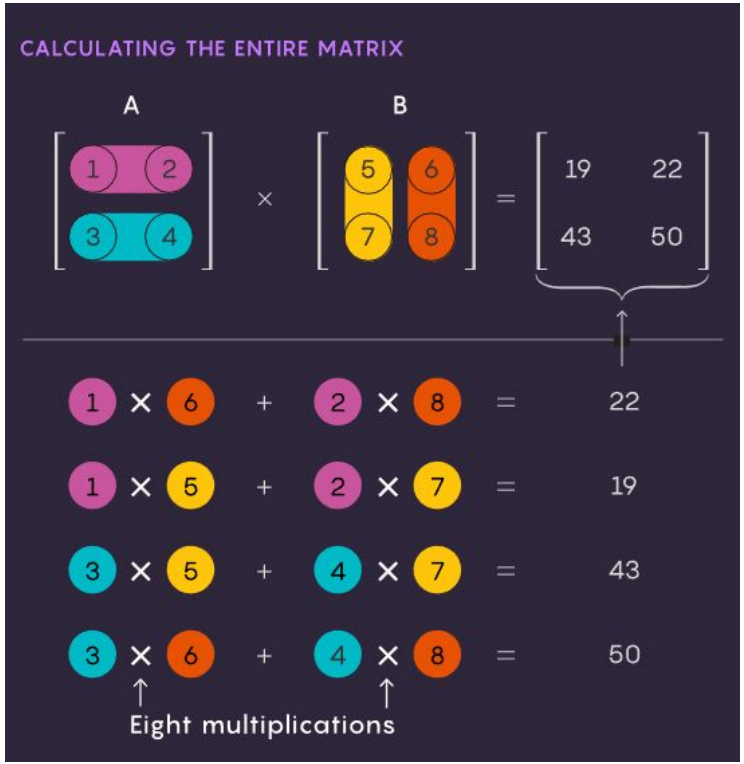
Merrill Sherman/Quanta Magazine

Matrix math with subdivisions : aka how we shard



Merrill Sherman/Quanta Magazine

Matrix math with subdivisions

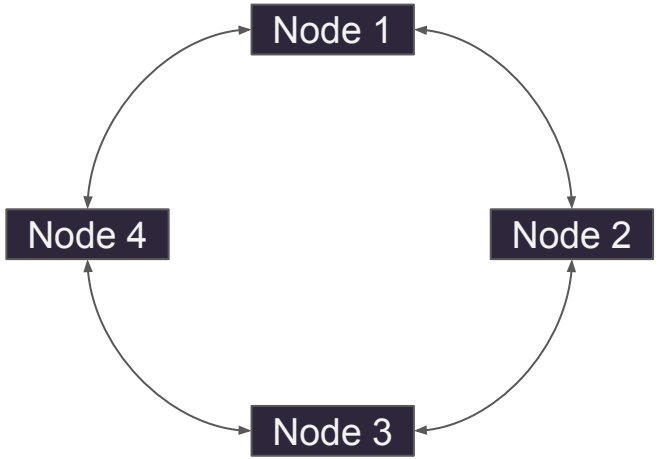


Merrill Sherman/Quanta Magazine

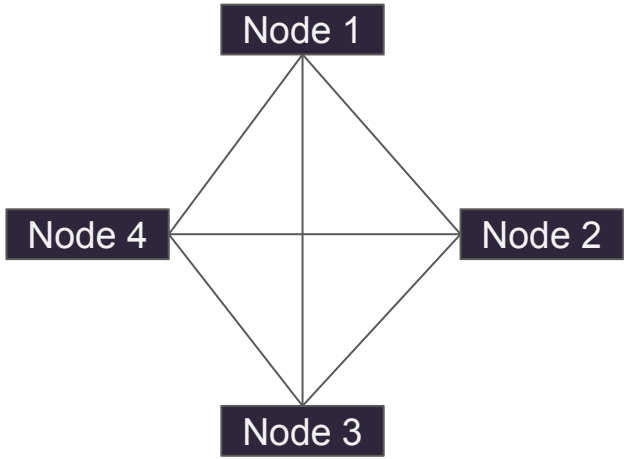
Mat(a,b) X Mat(x,y) = Mat(a,y)

- Parallelize operations by replicating one matrix in all local memories and shard the other matrix across multiple compute elements
- The goal is to have all compute elements have the full matrix multiply result
- Implemented using
  - Scatter-Reduce (above method)
  - And then All-Gather (broadcast)

Logical network topologies for ML

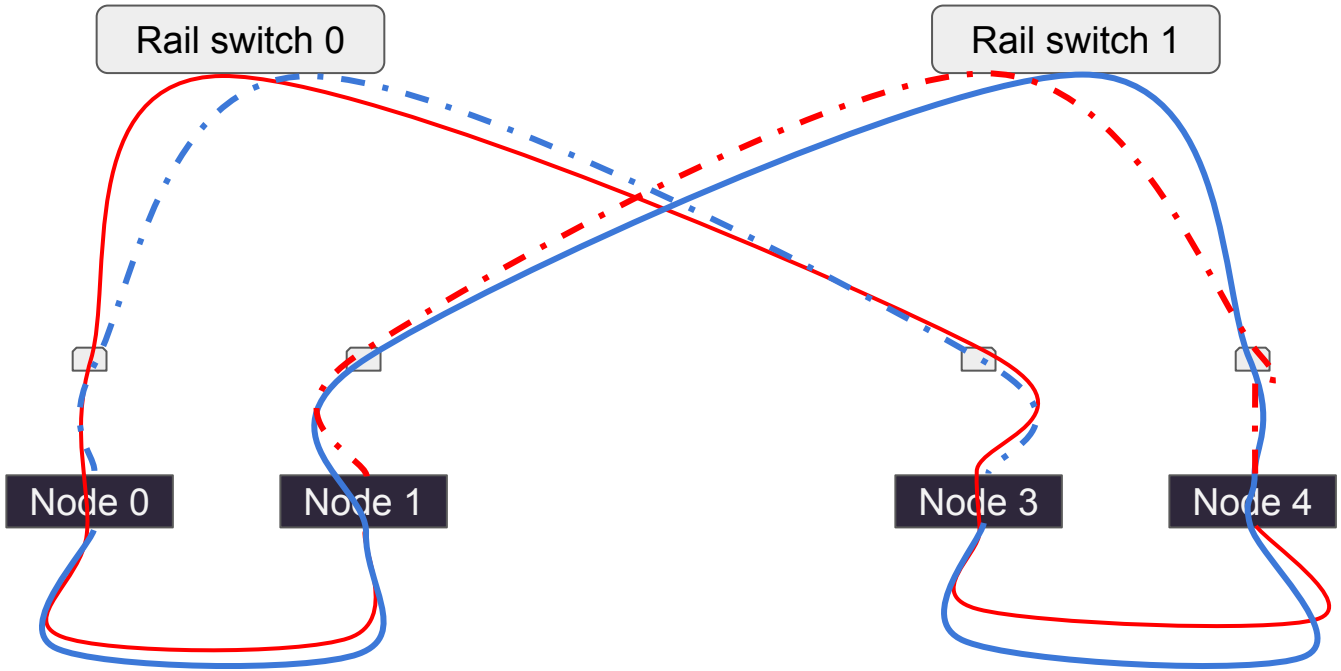


Ring Topology : 4 links
N links for N nodes

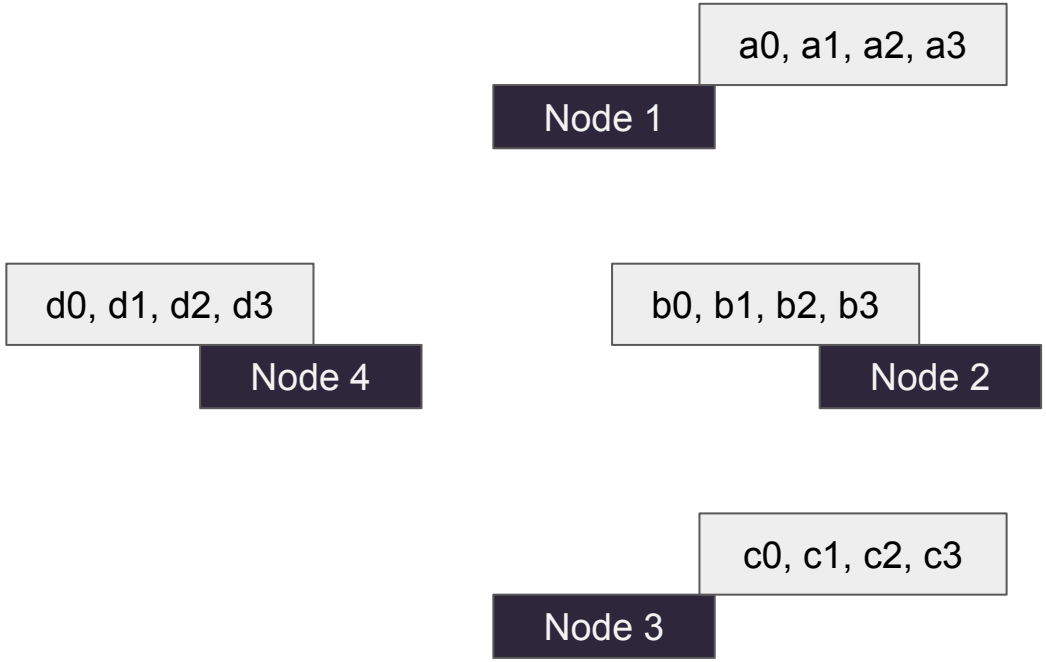


Mesh Topology : 6 links
N*(N-1)/2 for N nodes

Rings in practice



All-mighty reduce operations

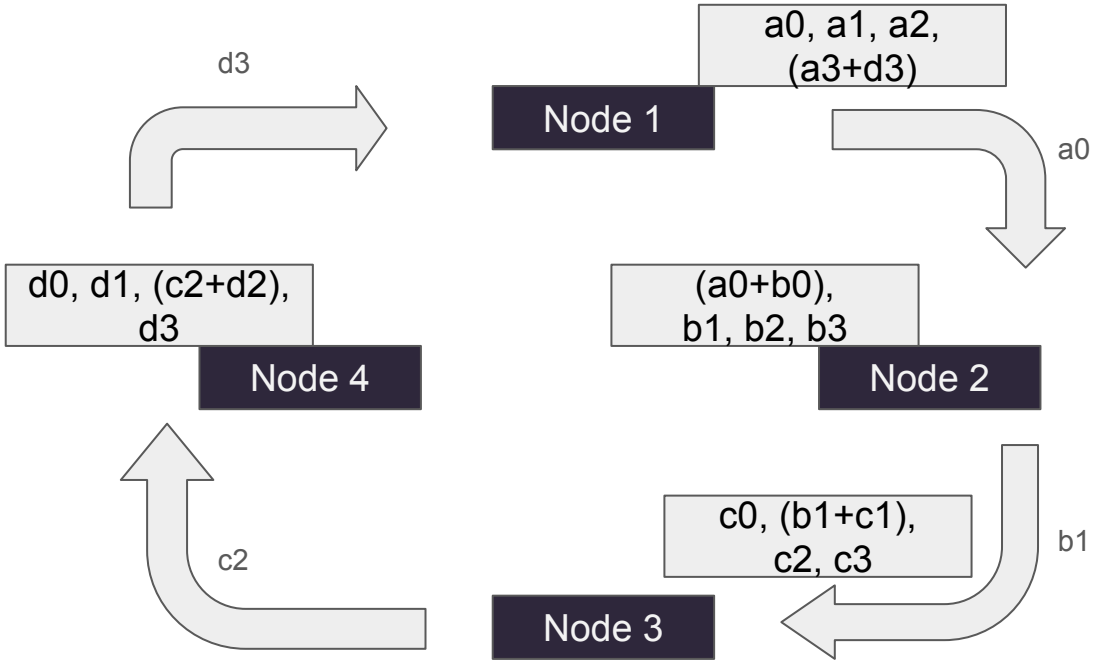


Example has 4 nodes

- Each node has a partial sum
- Goal is to get each node the full set of all sums i.e the actual final matrix

This example shows the usage of a ring topology for optimal BW utilization

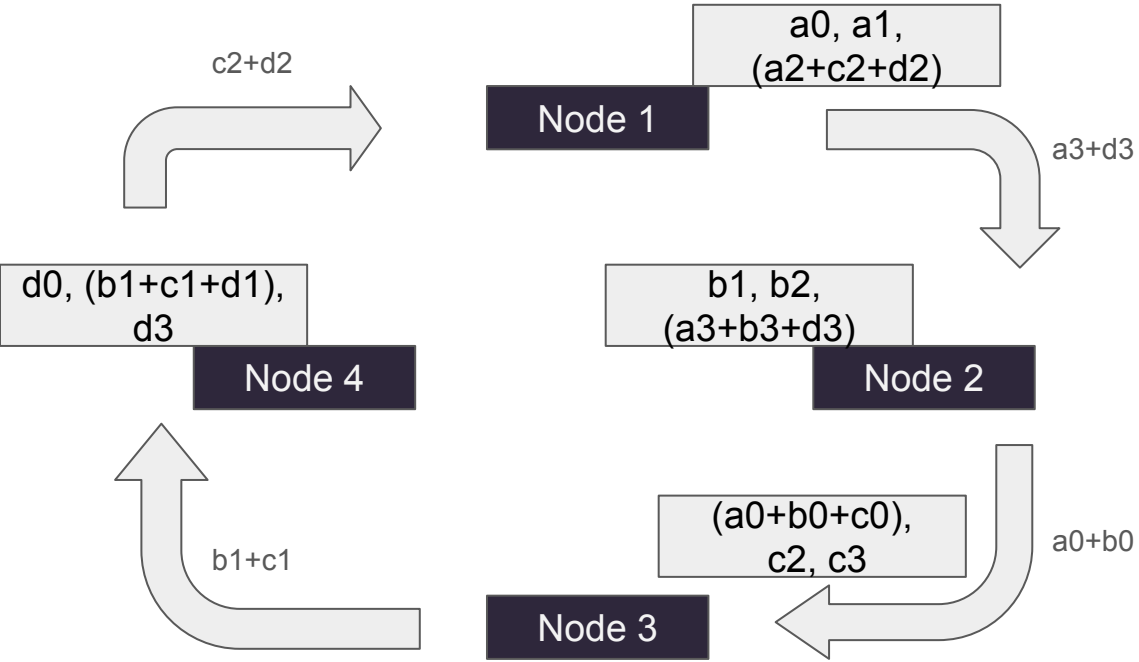
All-mighty reduce operations



Phase 1

- Every node sends one of it's local copies to one neighbor
- The index of the array sent is based on node location

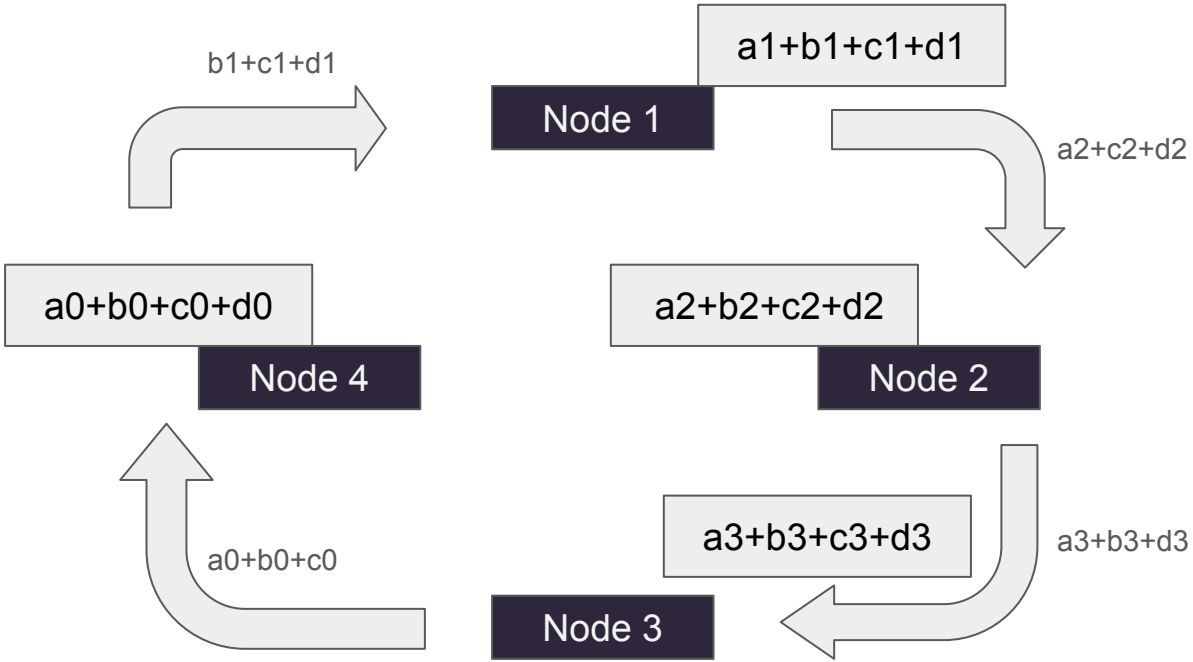
All-mighty reduce operations



Phase 2

- Each node sends the partial sums computed in the last phase to it's next neighbor
- Each node add's the partial sum to it's local value at the appropriate index

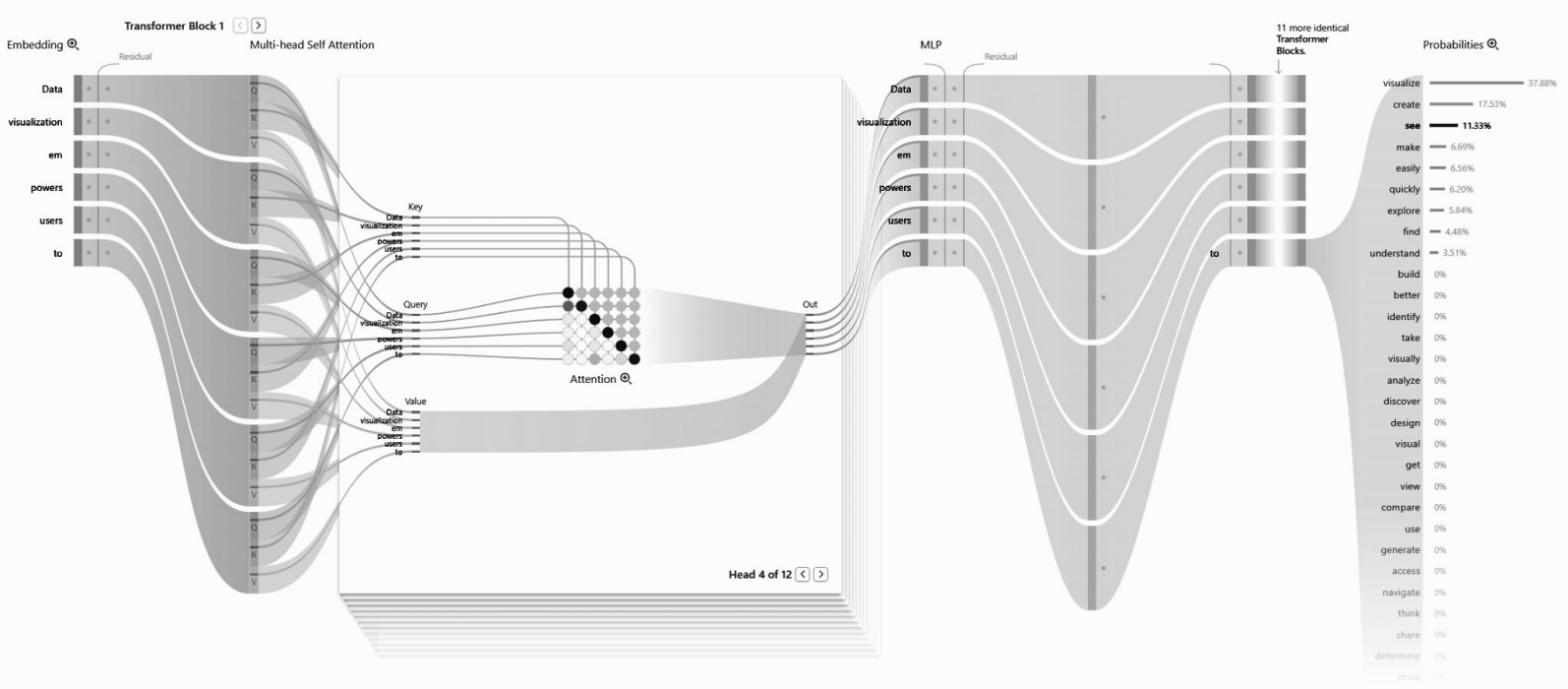
All-mighty reduce operations



Phase 3

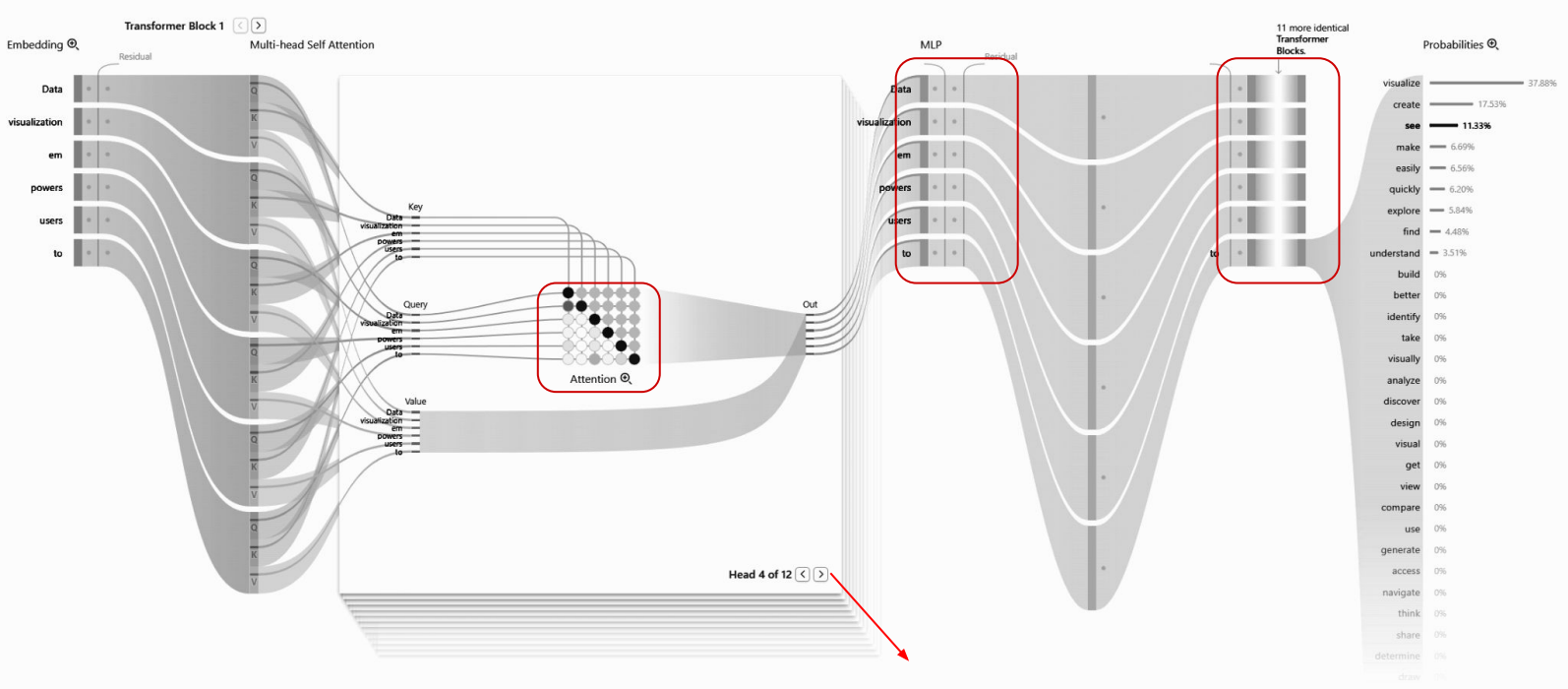
- Each node has the full sum for one of the 4 indices
- All that is missing is a broadcast/multicast to exchange all values with everyone else

The Basic’s of LLM’s



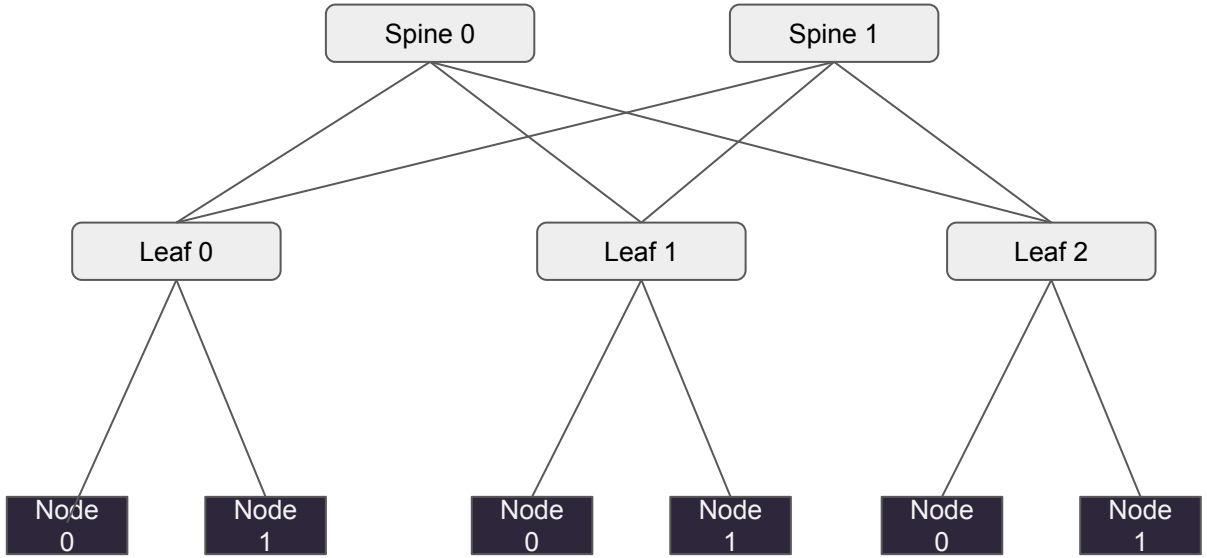
https://poloclub.github.io/transformer-explainer/

Matmult where art thou



https://poloclub.github.io/transformer-explainer/

Mapping to “standard” DC scale-out networks



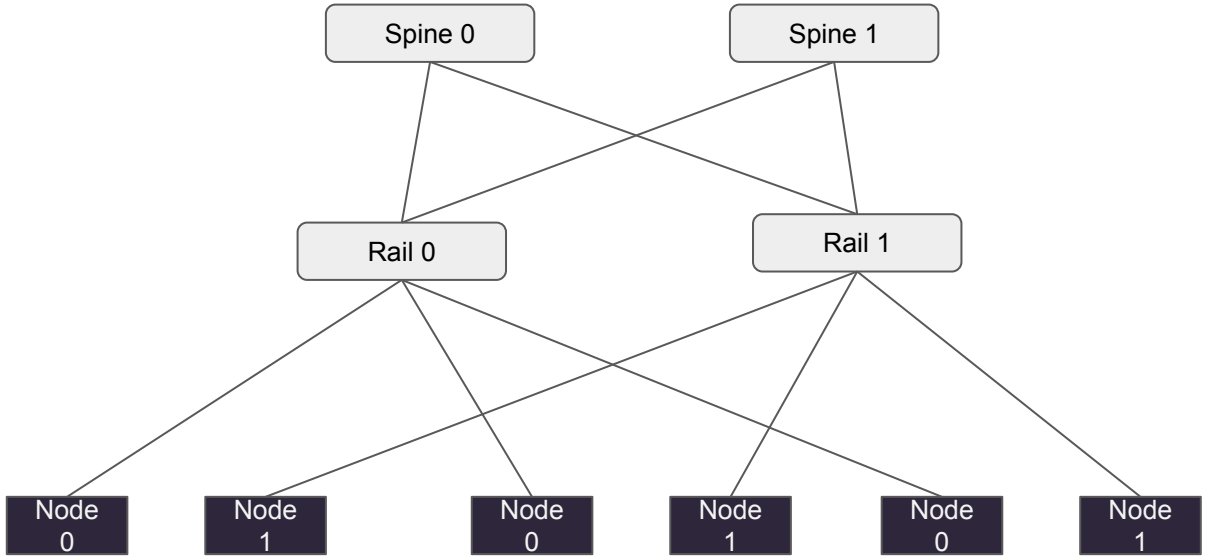
Classic Scale out networks

- Build for client server model
- Resiliency achieved by Software redundancy
- High Aggregate BW but not always per node pair

What is needed

- Very high inter node bandwidth 800g and above
- Very high link utilization using packet spray
- An application interface that allows direct HW access for the fastpath

Now “Rail optimized”



Packet and Message level spraying is needed to avoid building rail optimized designs

Introducing Ultra Ethernet Consortium (UEC)

Introducing Ultra Ethernet Consortium (UEC)

Physical Layer

In the world of Ethernet, the Physical Layer is the foundation upon which everything else rests. Our Physical Layer Working Group is dedicated to enhancing the performance, reducing latency, and improving the management of Ethernet's physical infrastructure. This includes the development of specifications for the Ethernet physical layer, electrical and optical signaling characteristics, application program interfaces, and data structures. Our goal is to make the foundation stronger, ensuring that Ethernet can meet the rigorous demands of AI, and HPC.

Link Layer

The Link Layer is where data packets are organized for efficient communication. Our Link Layer Working Group is all about enhancing the performance, reducing latency, and improving the management of Ethernet's link layer. We develop specifications that optimize Ethernet's efficiency, security, and scalability. We're extending and replacing existing link and transport protocols to ensure data flows smoothly and securely in AI, and HPC environments.

Transport Layer

The Transport Layer is crucial for end-to-end data delivery. Our Transport Layer Working Group focuses on developing specifications for an AI/HPC transport that delivers enhanced throughput, reduced latency, greater scalability, and improved management in Ethernet networks. We're ensuring that Ethernet can handle the high-performance demands of AI, and HPC applications without missing a beat.

Software Layer

In the age of software-defined everything, our Software Layer Working Group takes center stage. We're developing specifications, software APIs, and open-source code to support a wide range of AI/HPC use cases and applications. This includes optimizing remote memory access, enabling In Network Computations (INC), and addressing security, management, and storage concerns. We're making Ethernet more flexible, programmable, and adaptable for AI, and HPC environments.

Storage Working Group

The Storage Working Group adds storage services to Ultra Ethernet (UEC) based AI and HPC workloads, collaborating with other UEC workgroups for seamless integration. Our focus spans compatibility with industry-standard physical connections, link-level capabilities negotiation, and packet encoding mechanisms as well as addresses emerging storage services. We integrate transport services, enhance security, and ensure RDMA API compatibility through collaboration with the software workgroup. Finally, we address the challenging task of integrating industry-established practices of storage management with the UEC, bolstering the foundation for high-performance storage traffic.

Compliance Working Group

The Compliance Working Group is focused on UEC specifications for AI and HPC workloads and is all about making sure services and devices meet the UEC defined technology. We define specs and create tests to assess UEC implementations, ensuring rigorous compliance with ratified UEC standards. We also define interoperability goals among UEC compliant network devices (e.g., integrated NICs, PCIe NICs and switches) according to UEC defined AI and HPC profiles.

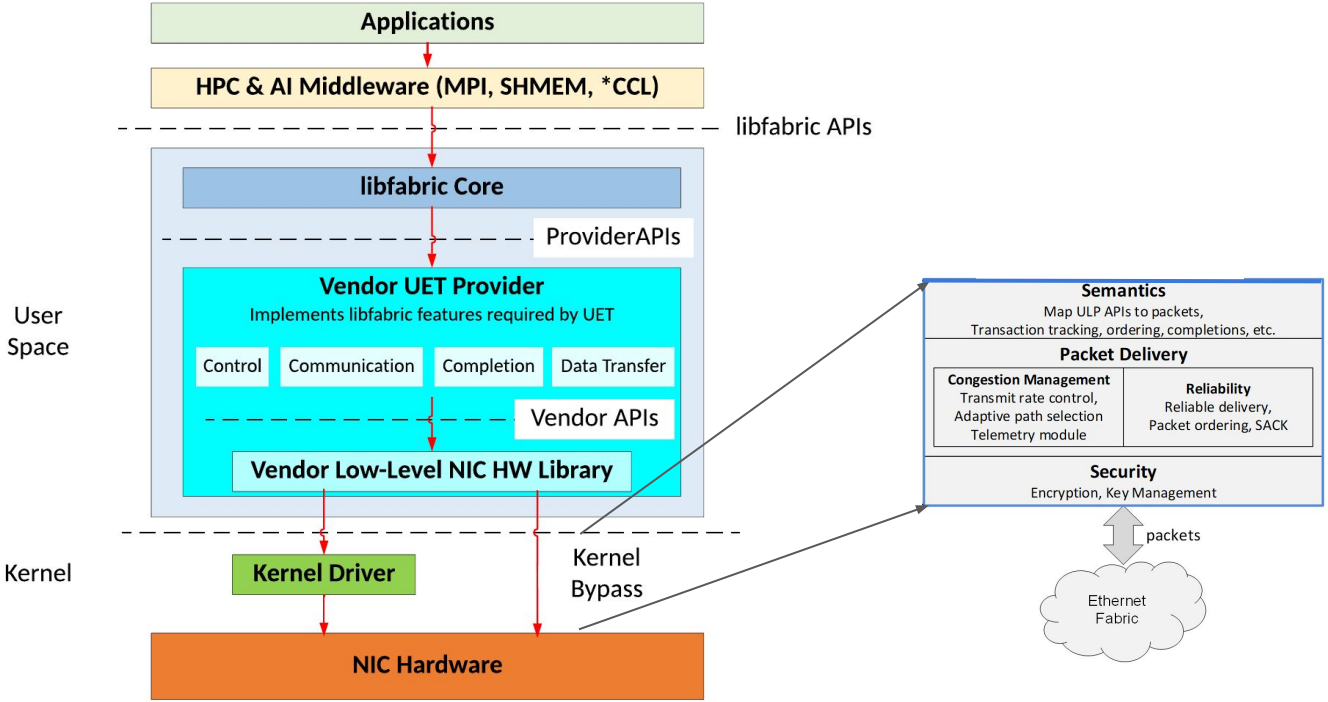
Management Working Group

The Management Working Group enhances UEC fabric manageability by collaborating with other workgroups to integrate evolving management specifications. We define management elements, RPCs, and models for switch and UEC compliant Transport Fabric End Point (FEP) management. Our efforts include topology discovery, capability discovery, monitoring, and interoperability queries to ensure seamless interaction among UEC fabric components.

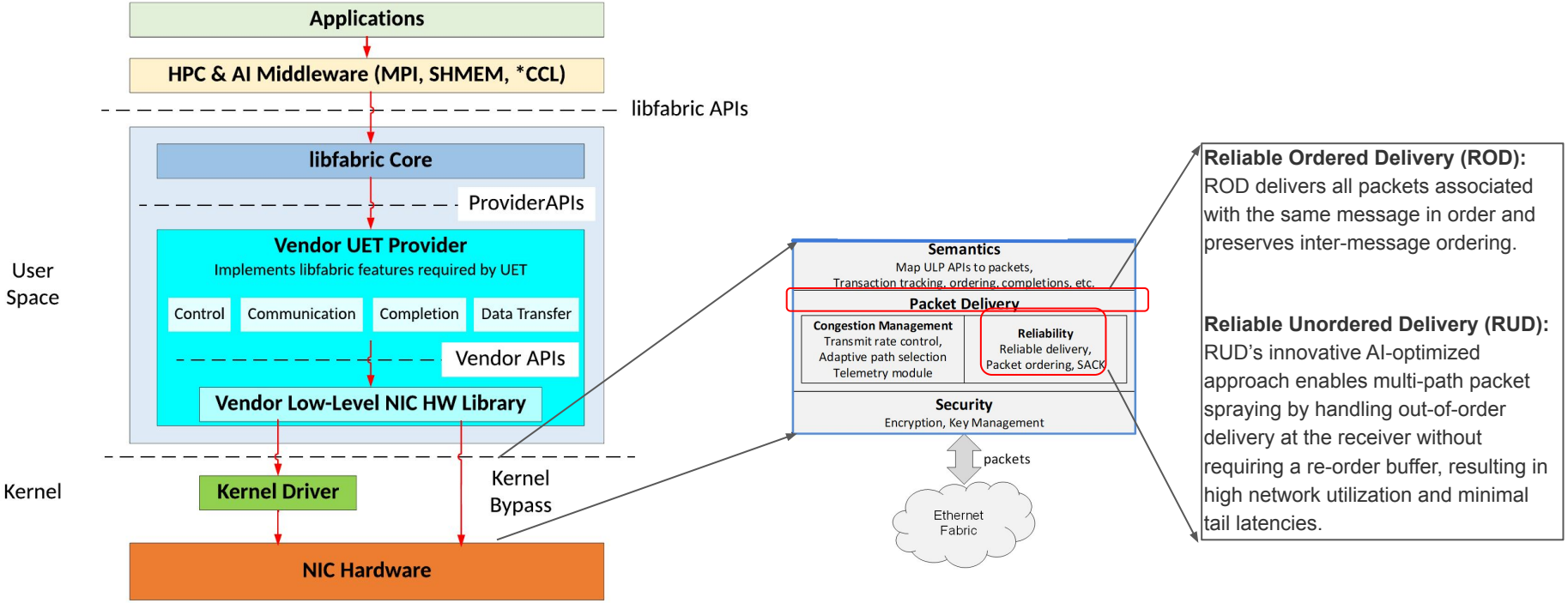
Performance and Debug Working Group

The Performance and Debug Working Group defines performance benchmarks and debug capabilities and tools for AI and HPC workloads. We align with evolving UEC specifications and define performance metrics, benchmarks, and debug features. Our aim is to create performance metrics that communicate to customers UEC's capabilities, performance and unique merits for their application. It will also assist developers, DevOps and network operation teams by enhancing visibility and debuggability in UEC-compliant implementations. We provide test cases and software tools for validation, creating a robust environment for AI and HPC applications

UE Transport : basic building blocks



UE Transport : basic building blocks



Ultra Ethernet Linux kernel modules

- Ultra Ethernet core (ultraeth.ko)
  - UET context management
  - Job management
  - UET genetlink interface
  - Generic UET device management
  - IB verbs character device

- Ultra Ethernet software device model (uecon.ko)
  - Dependent on UE core (obviously)
  - Implements UET sublayers in software
  - Implements UET congestion management
  - UDP tunnel network device

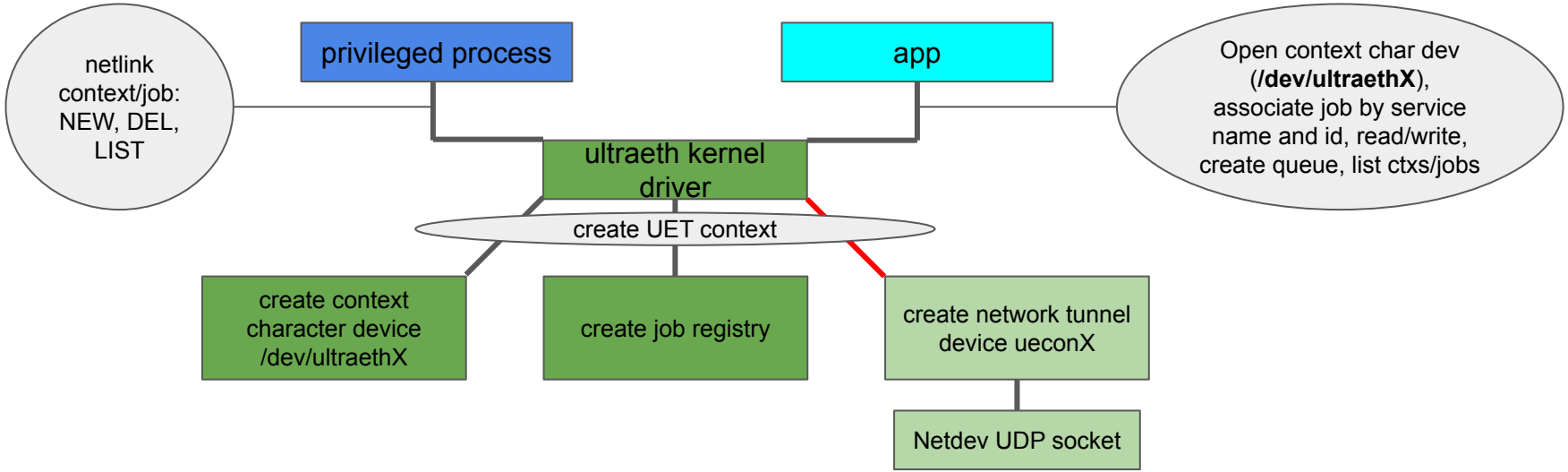
Uecon : software Ultra Ethernet model

- uecon is a **single** UET driver implementing the UET specifications for communication, it will be separate from UET core (context mgmt, jobs mgmt, generic resource mgmt) which will end up in drivers/ultraeth/core/
  - think of it as a single software UET device driver that can be loaded and created on-demand

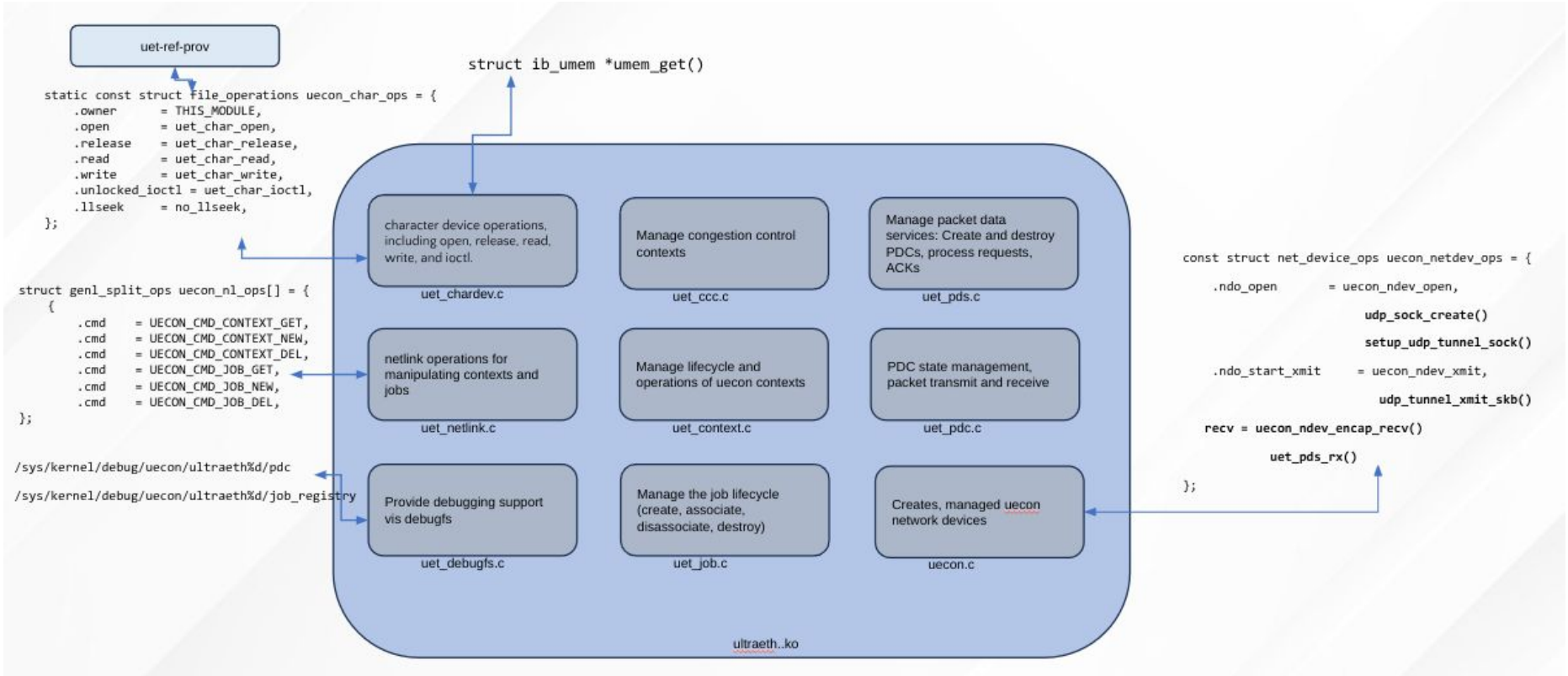
- Implement Packet Delivery Sublayer (PDS) specification
  - Responsible for dynamically creating Packet Delivery Contexts
  - Keeps track of PDC ids which are unique per-PDC
  - Responsible for packet delivery over IP/Ethernet network
  - Tx/Rx NACK packets for various events
  - Finds (or creates dynamically) PDCs on Rx/Tx based on endpoint addresses and unique PDC ids
  - Packet types: RUD/ROD/RUDI/UUD Request, RUDI Response, ACK, NACK, Control message

- PDCs are dynamic connections between two Fabric Endpoints (FEPs)
  - Responsible for packet reliability, ordering, duplicate elimination and congestion management
  - Track Tx/Rx Packet Sequence Number (PSN) spaces
  - Support coalescing ACK (CACK) and selective ACK (SACK)
  - Can establish multiple PDCs between the same two FEPs
  - Have a specific mode:
    - Reliable, Ordered Delivery (ROD)
    - Reliable, Unordered Delivery (RUD)
    - Reliable, Unordered Delivery for Idempotent Operations (RUDI)
    - Unreliable, Unordered Delivery (UUD)

Ultra Ethernet (ultraeth) driver flow



Code blocks and organization



UET code organization

- Current (RFC set) state: all code together in linux/drivers/ultraeth/
  - PDS is part of UET context (will move to uecon)
  - uecon created dynamically with contexts (will be created only on demand)
  - custom character device created with contexts (use special UET IB verbs device)
  - New kconfig option: CONFIG_ULTRAETH
  - Netlink API for UET core resource management (e.g. context, job)

- Future:
  - UET core (contexts, jobs, generic UET resources): linux/drivers/ultraeth/core/
  - UET device drivers: linux/drivers/ultraeth/devices (TBD)
  - uecon moves to its own directory with PDS and all sublayers that are expected to be executed in hw, separate kconfig option (CONFIG_ULTRAETH_UECON)
  - uecon congestion management and additional UET sublayers
  - UET core <-> UET device in-kernel API
  - UET core <-> user-space API

Outstanding actions

- Get involved
- Need to leverage (and not rewrite) the RDMA layers
  - IB device, Fixed queue maps etc need slight tweaking
  - Memory management is entirely reusable
  - RFC patches being prepped that
- User and HW interfaces are mapped as libfabric
  - Kernel subsystem testing without libfabric will require work
  - In kernel implementation (e.g storage) is being examined