Systems for Machine Learning (CMSC828G)

Introduction to Systems / HPC

Abhinav Bhatele, Daniel Nichols

UNIVERSITY OF MARYLAND

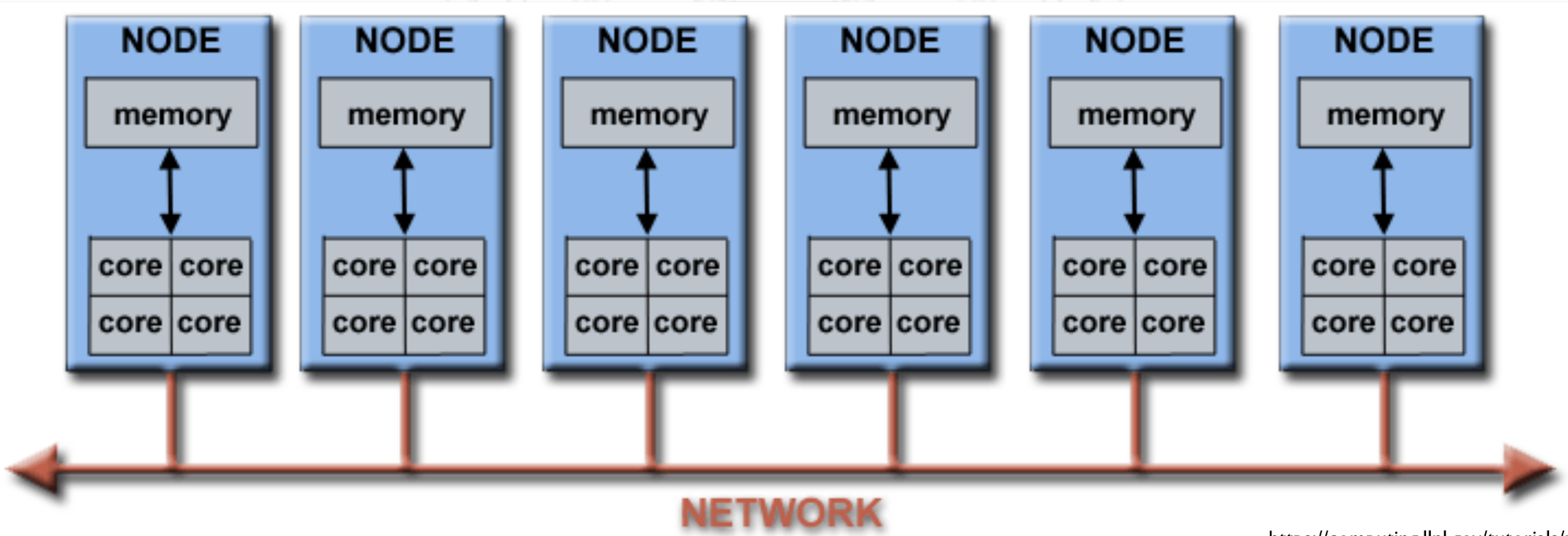
Getting started with zaratan

- Over 360 nodes with AMD Milan processors (128 cores/node, 512 GB memory/node)
- 20 nodes with four NVIDIA A100 GPUs (40 GB per GPU)
- 8 nodes with four NVIDIA H100 GPUs (80 GB per GPU)

```
ssh username@login.zaratan.umd.edu
```

Data center / HPC cluster

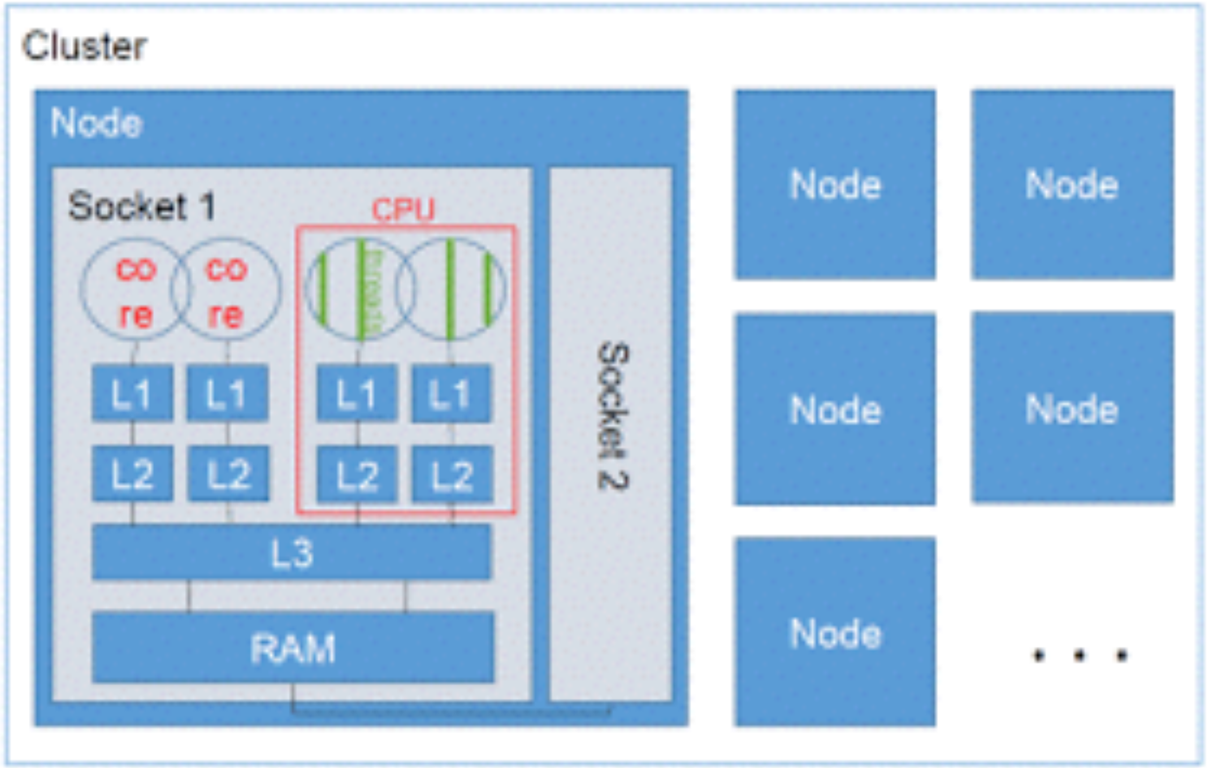
- A set of nodes or processing elements connected by a network.
- Compute node: A shared-memory unit (optionally has GPUs)



https://computing.llnl.gov/tutorials/parallel_comp

Cores, sockets, nodes

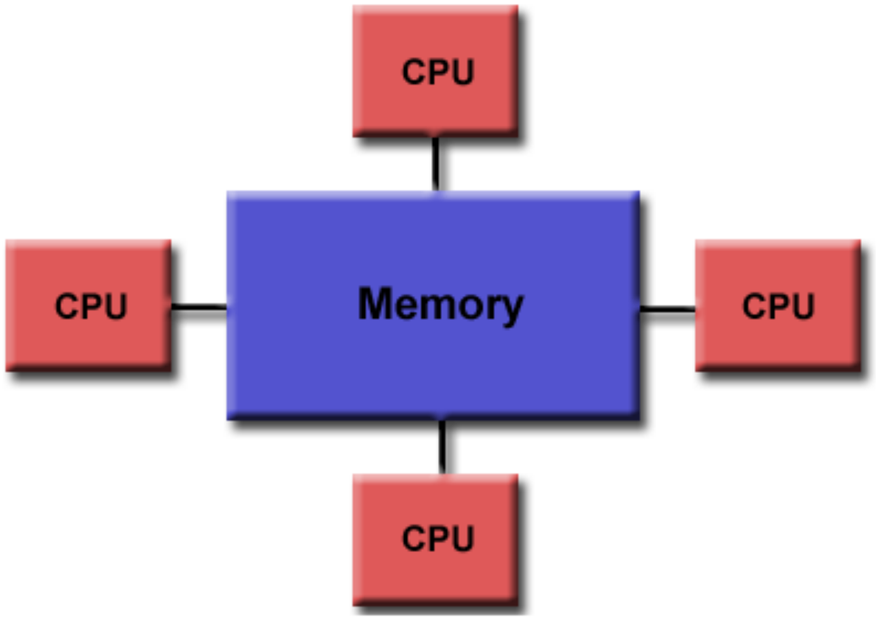
- Core: a single execution unit that has a private L1 cache and can execute instructions independently
- Processor: several cores on a single Integrated Circuit (IC) or chip are called a multi-core processor
- Socket: physical connector into which an IC/chip or processor is inserted.
- Node: a packaging of sockets — motherboard or printed circuit board (PCB) that has multiple sockets



https://hpc-wiki.info/hpc/HPC-Dictionary

Shared memory architecture

- All processors/cores can access all memory as a single address space

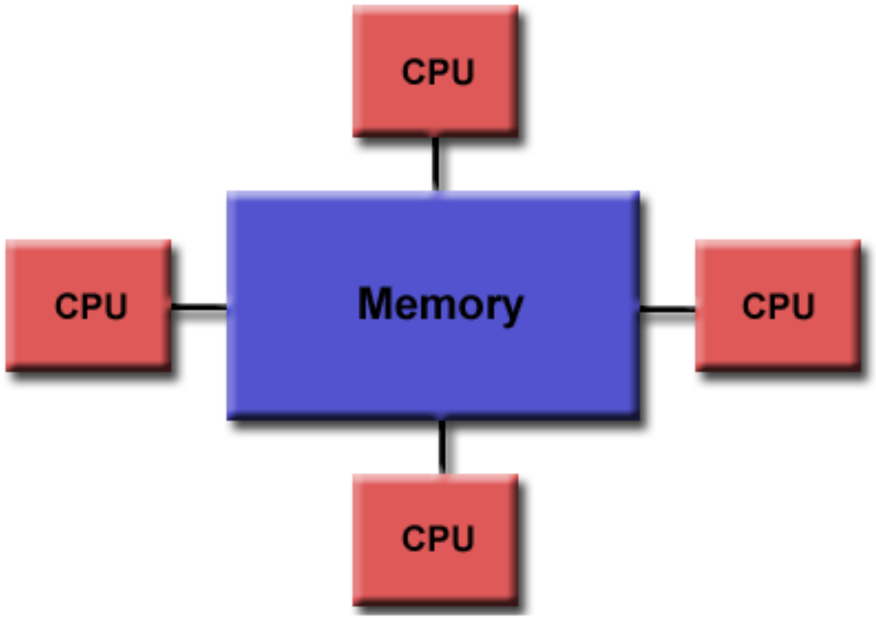


Uniform Memory Access

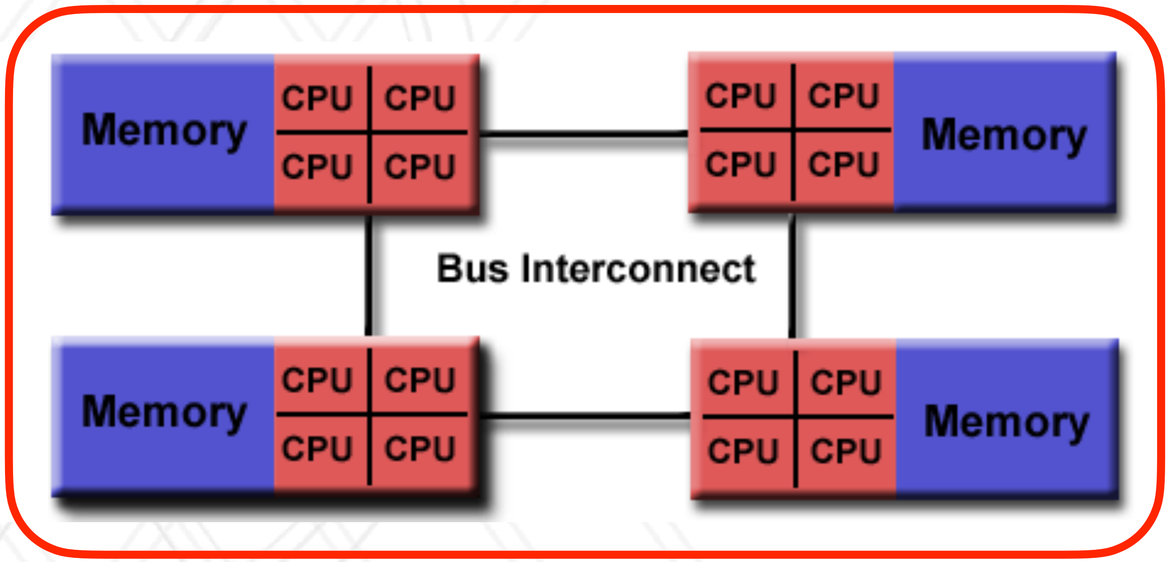
https://computing.llnl.gov/tutorials/parallel_comp/#SharedMemory

Shared memory architecture

- All processors/cores can access all memory as a single address space



Uniform Memory Access



Non-uniform Memory Access (NUMA)

https://computing.llnl.gov/tutorials/parallel_comp/#SharedMemory

Hopper H100 SM

- CUDA Core
  - Single serial execution unit
- Each H100 Streaming Multiprocessor (SM) has:
  - 128 FP32 cores
  - 64 INT32 cores
  - 64 FP64 cores
  - 84 Tensor cores
- CUDA capable device or GPU
  - Collection of SMs

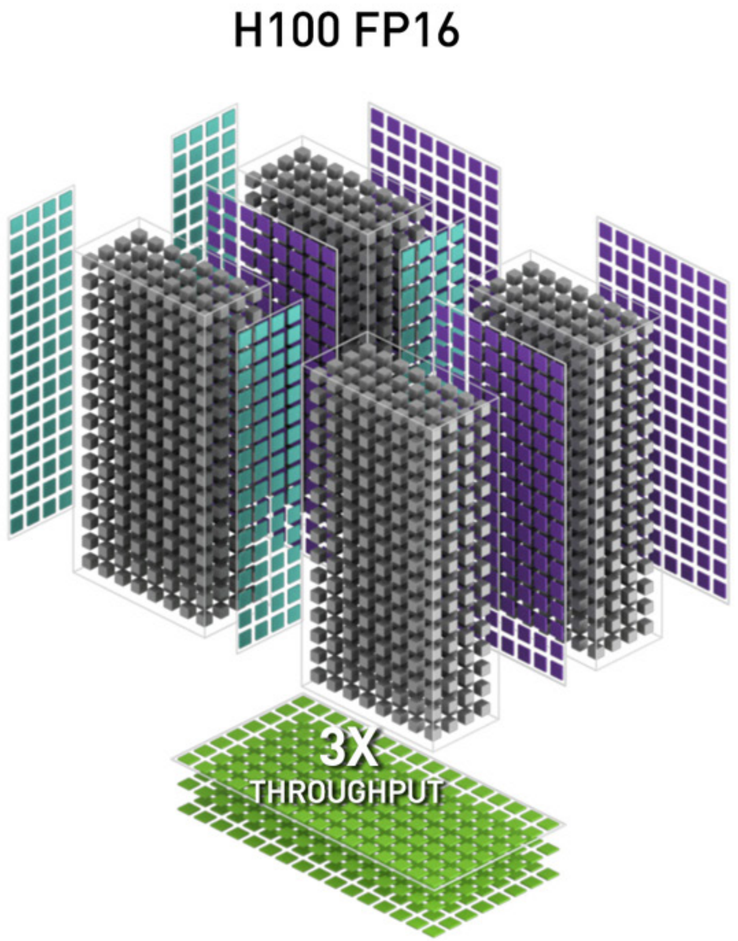
Hopper H100 SM

- CUDA Core
  - Single serial execution unit
- Each H100 Streaming Multiprocessor (SM) has:
  - 128 FP32 cores
  - 64 INT32 cores
  - 64 FP64 cores
  - 84 Tensor cores
- CUDA capable device or GPU
  - Collection of SMs

NVIDIA H100 chip

H100 tensor cores

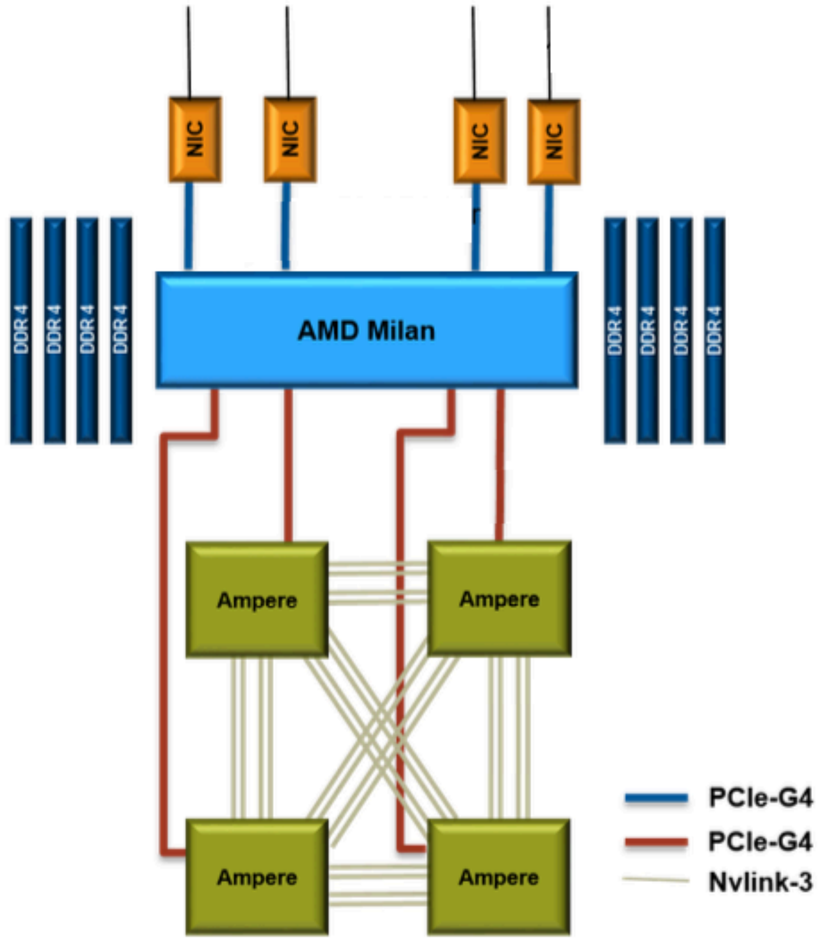
- Tensor cores are specialized cores for matrix multiply accumulate operations
- Operate in parallel across all SMs
- Multiply two 4 x 4 FP16 matrices and add to a 4 x 4 FP16 or FP32 matrix
- Mixed precision



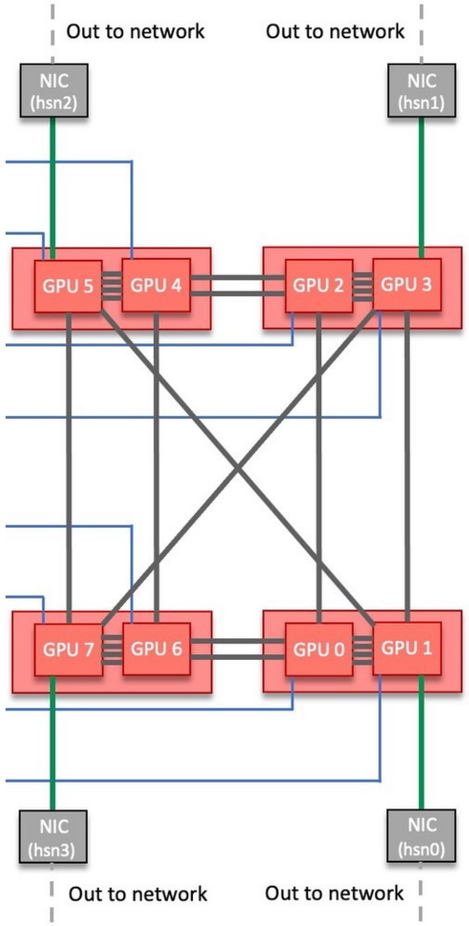
https://resources.nvidia.com/en-us-tensor-core

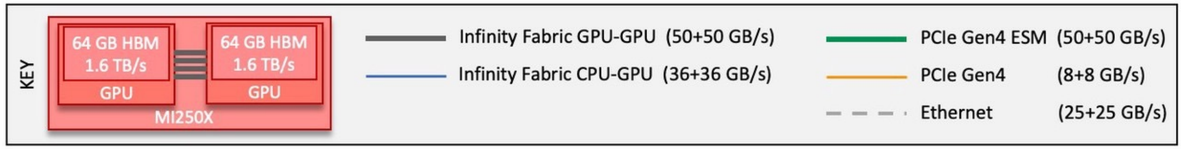
Nodes with GPUs

- NIC: Network interface card that connects the node to the network
- PCIe: high-speed interface often used to connect CPUs and GPUs
- NVLink: NVIDIA's high-speed interface often used between GPUs

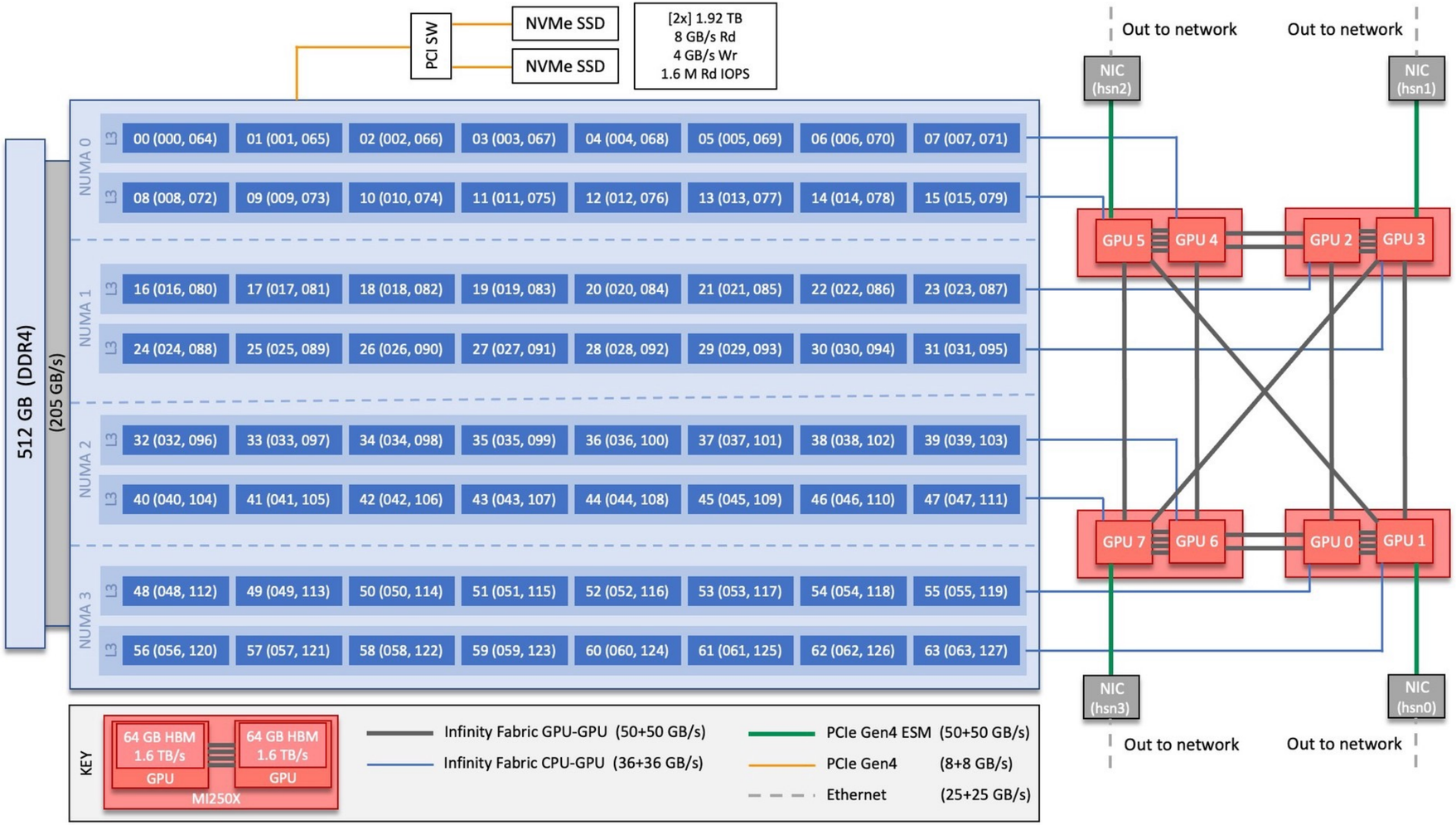


Alternative node diagram



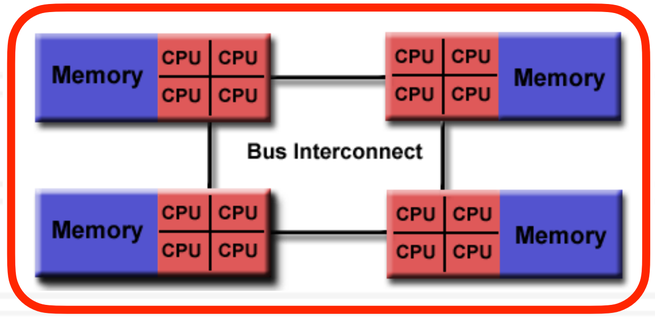


Alternative node diagram

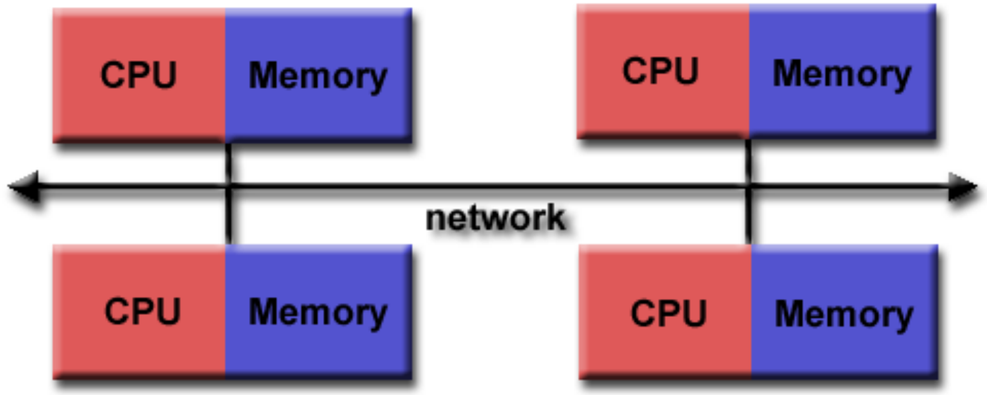


Distributed memory architecture

- Groups of processors/cores have access to their local memory
- Writes in one group's memory have no effect on another group's memory



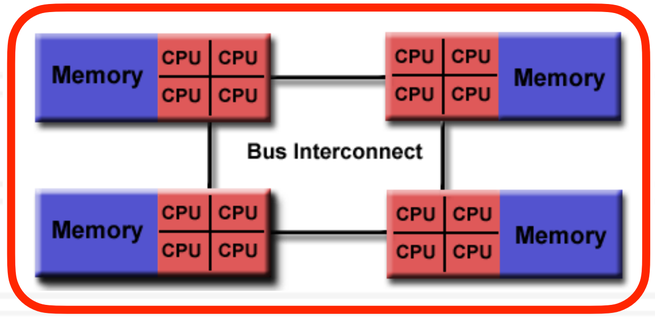
Shared memory (NUMA)



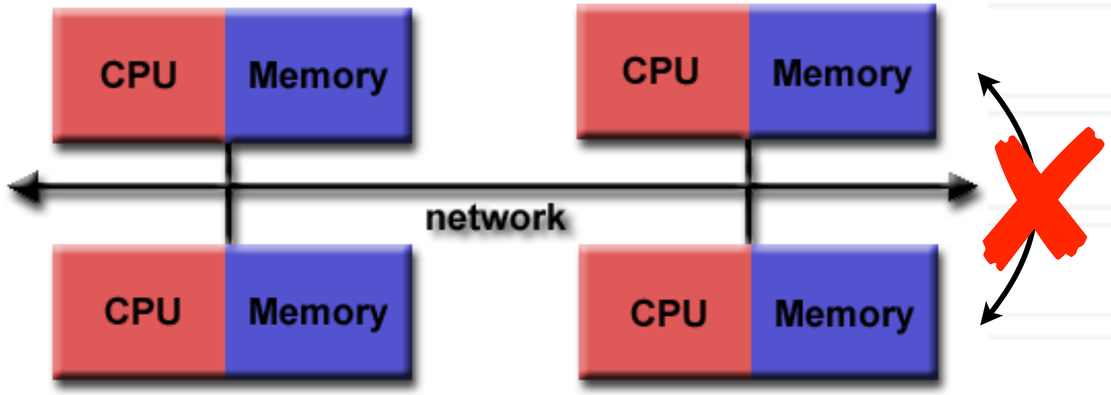
Distributed memory

Distributed memory architecture

- Groups of processors/cores have access to their local memory
- Writes in one group's memory have no effect on another group's memory



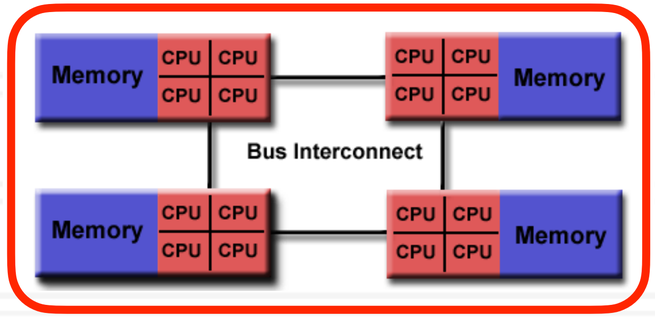
Shared memory (NUMA)



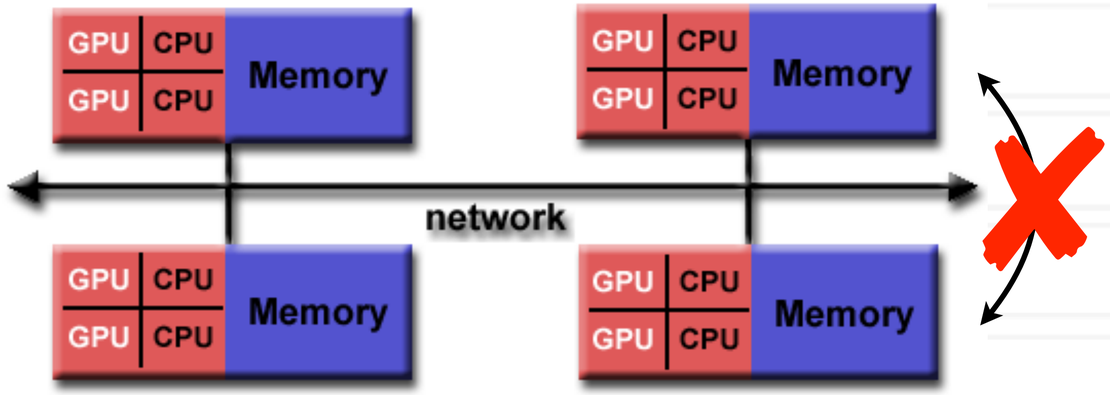
Distributed memory

Distributed memory architecture

- Groups of processors/cores have access to their local memory
- Writes in one group's memory have no effect on another group's memory



Shared memory (NUMA)



Distributed memory

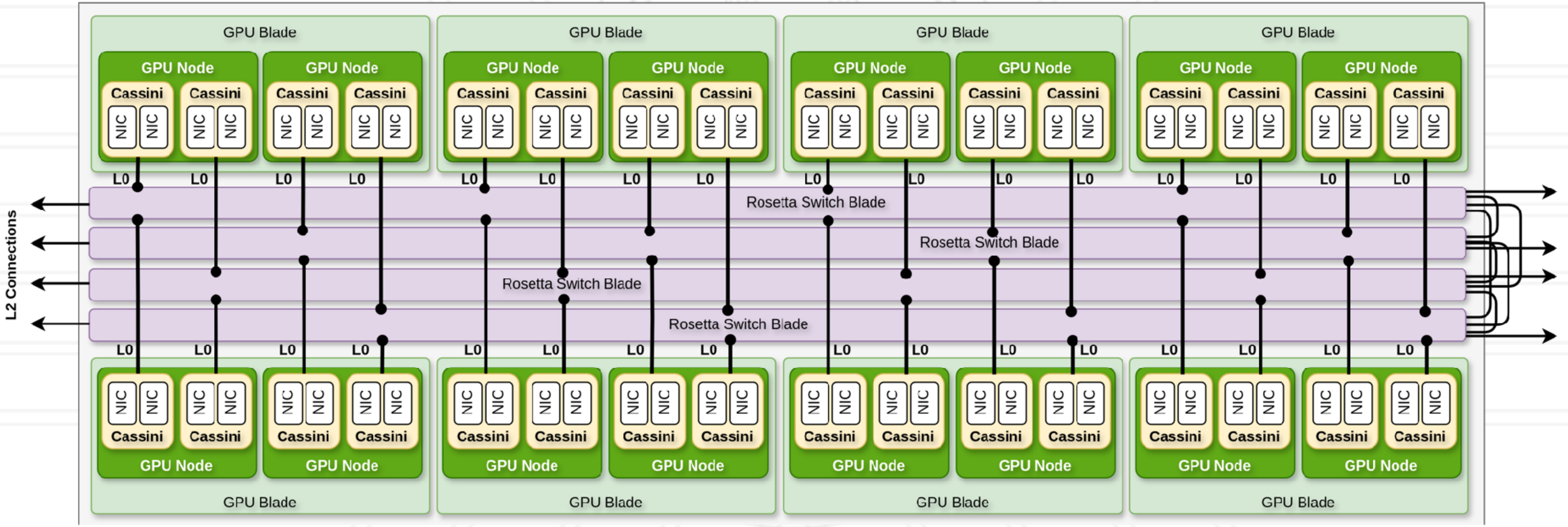
Multithreading vs. SMT

- Multithreading: multiple threads can share the same CPU or core via time-sharing
  - If one thread stalls, another can start running
- SMT: Simultaneous multithreading or hyperthreading
  - Implemented in hardware
  - Two or more threads can run simultaneously on a single core
  - Requires the core to be a superscalar

Group Projects

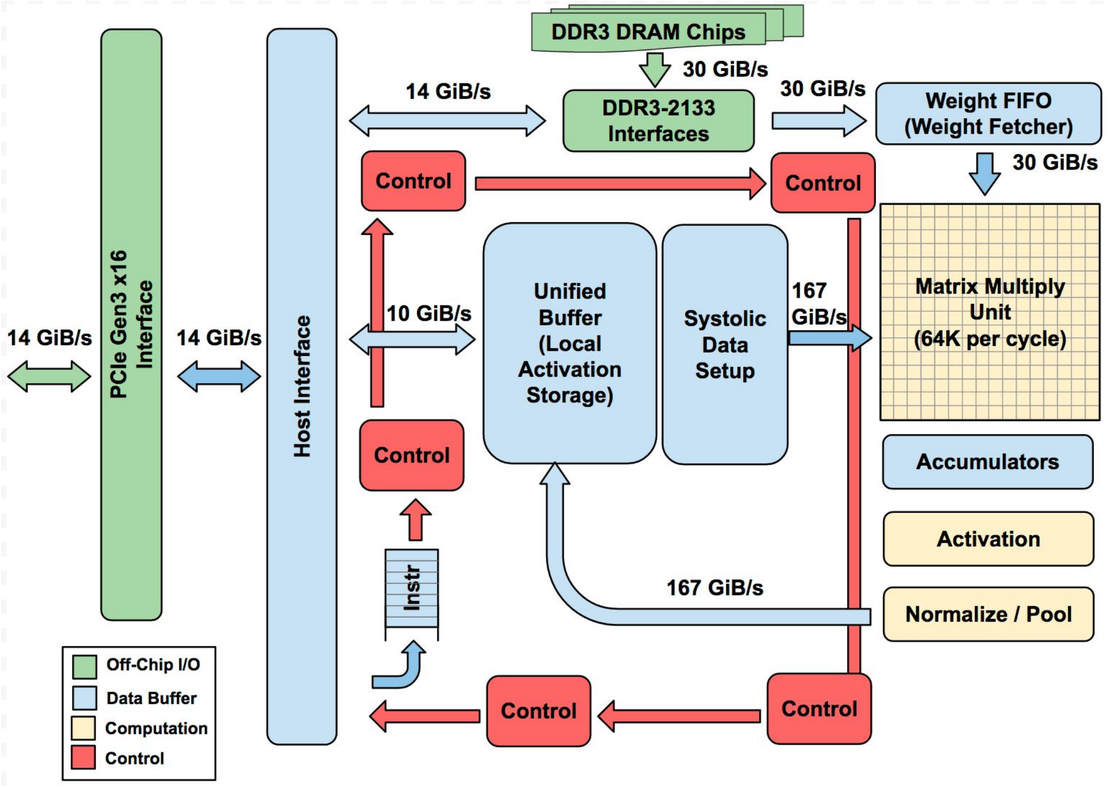
- Self form into groups of 2-3
- Project will be ideally at the intersection of systems + ML
  - Using parallel systems to optimize an ML workload
- Timeline (all deadlines are midnight):
  - Group formation and project proposal: March 4
  - Interim report: April 17
  - Final presentation: May 6-13
  - Final report and code: May 15

A realistic cluster



Google's Tensor Processing Unit

- TPU is an ASIC (Application-specific Integrated Circuit)
- Co-processor just like GPUs
- Each TPU can have one or multiple MMUs
- TPU Pod is a collection of TPUs



https://arxiv.org/pdf/1704.04760

http://web.cecs.pdx.edu/~mperkows/temp/May22/0020.Matrix-multiplication-systolic.pdf

Network components

- Network interface controller or card
- Router or switch
- Network cables: copper or optical

Life-cycle of a message

Source

Source

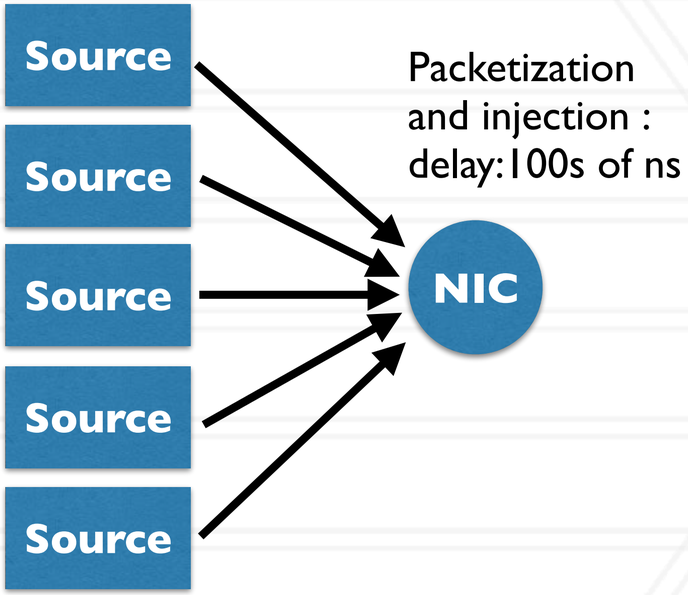
Source

Source

Source

Message origin points : destination, frequency, size, etc. determined by application
1 micro sec - 10s of sec

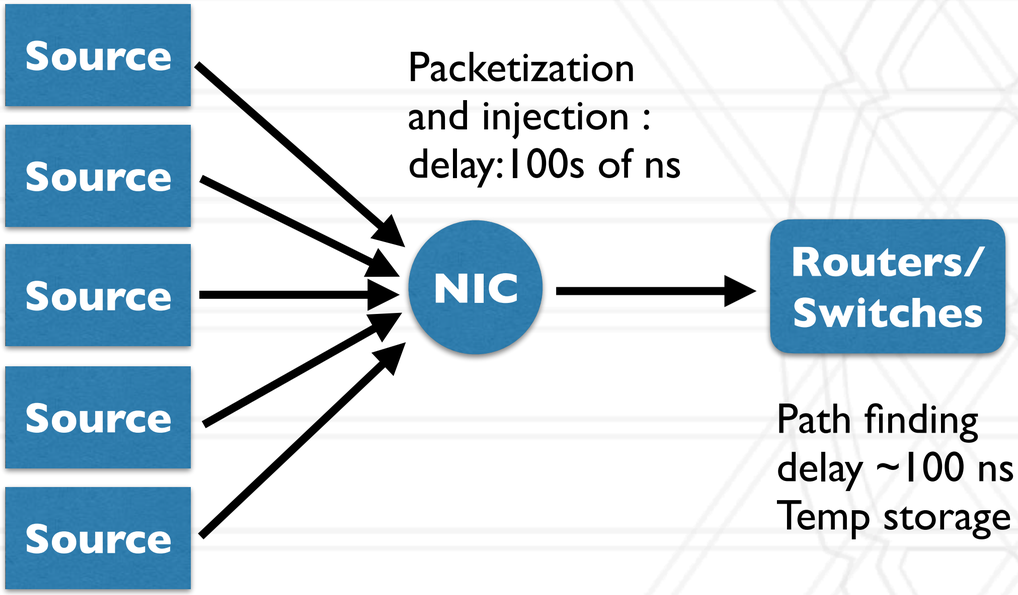
Life-cycle of a message



Packetization and injection : delay: 100s of ns

Message origin points : destination, frequency, size, etc. determined by application
1 micro sec - 10s of sec

Life-cycle of a message

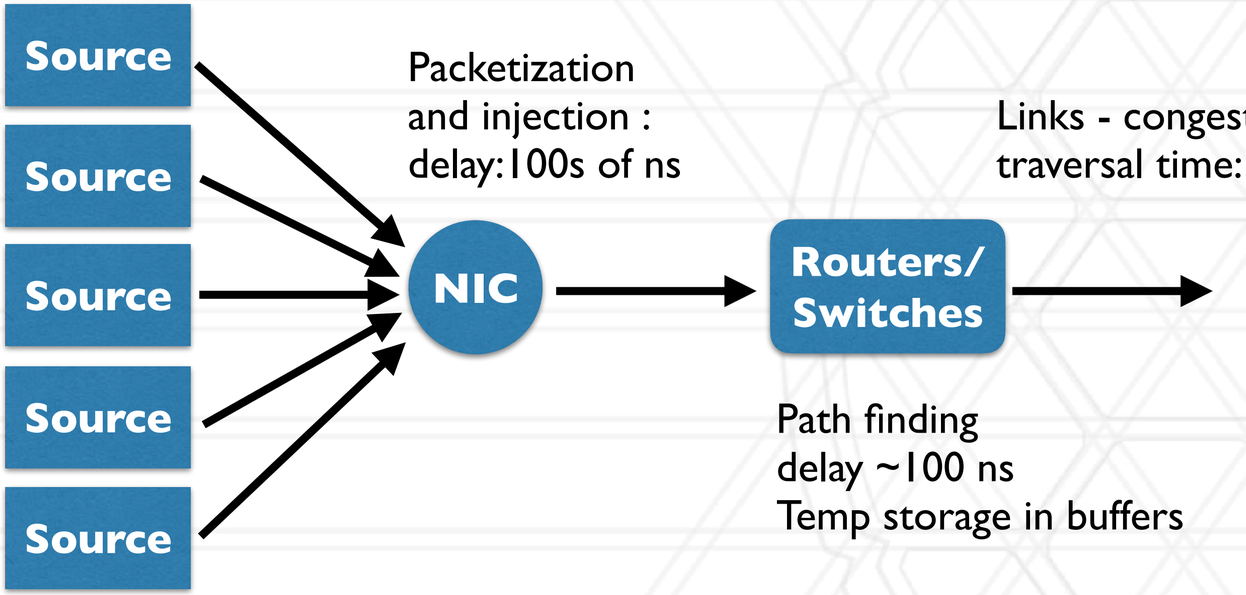


Packetization
and injection :
delay:100s of ns

Path finding
delay ~100 ns
Temp storage in buffers

Message origin points :
destination, frequency,
size, etc. determined
by application
1 micro sec - 10s of sec

Life-cycle of a message



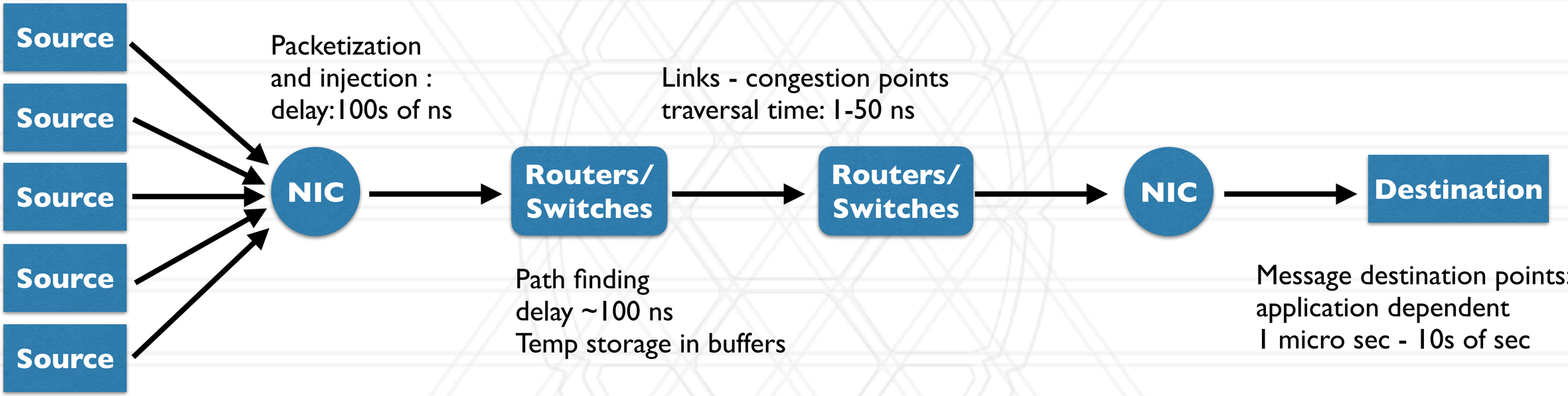
Packetization
and injection :
delay:100s of ns

Links - congestion points
traversal time: 1-50 ns

Path finding
delay ~100 ns
Temp storage in buffers

Message origin points :
destination, frequency,
size, etc. determined
by application
1 micro sec - 10s of sec

Life-cycle of a message



Packetization and injection : delay:100s of ns

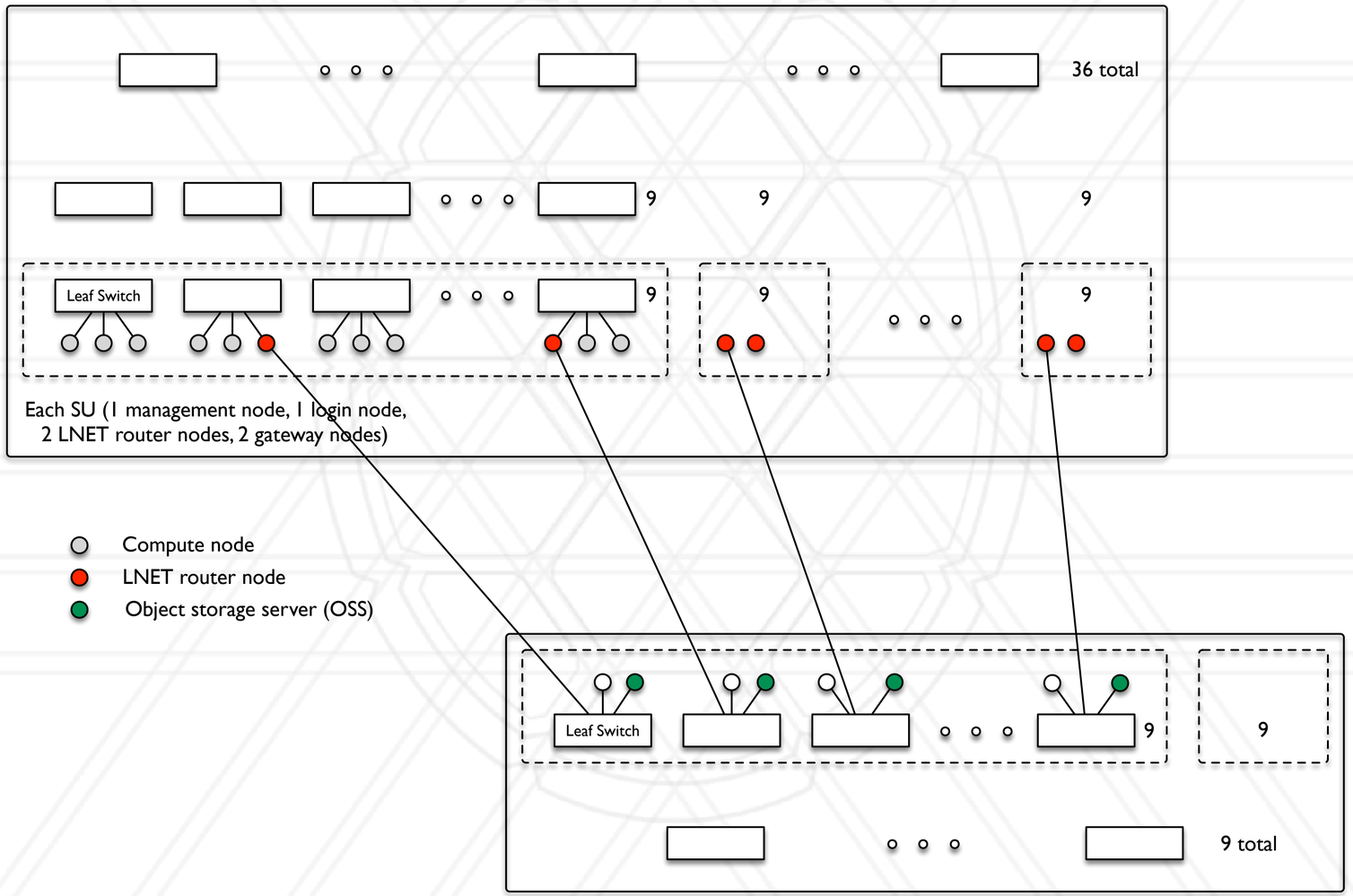
Links - congestion points traversal time: 1-50 ns

Path finding delay ~100 ns Temp storage in buffers

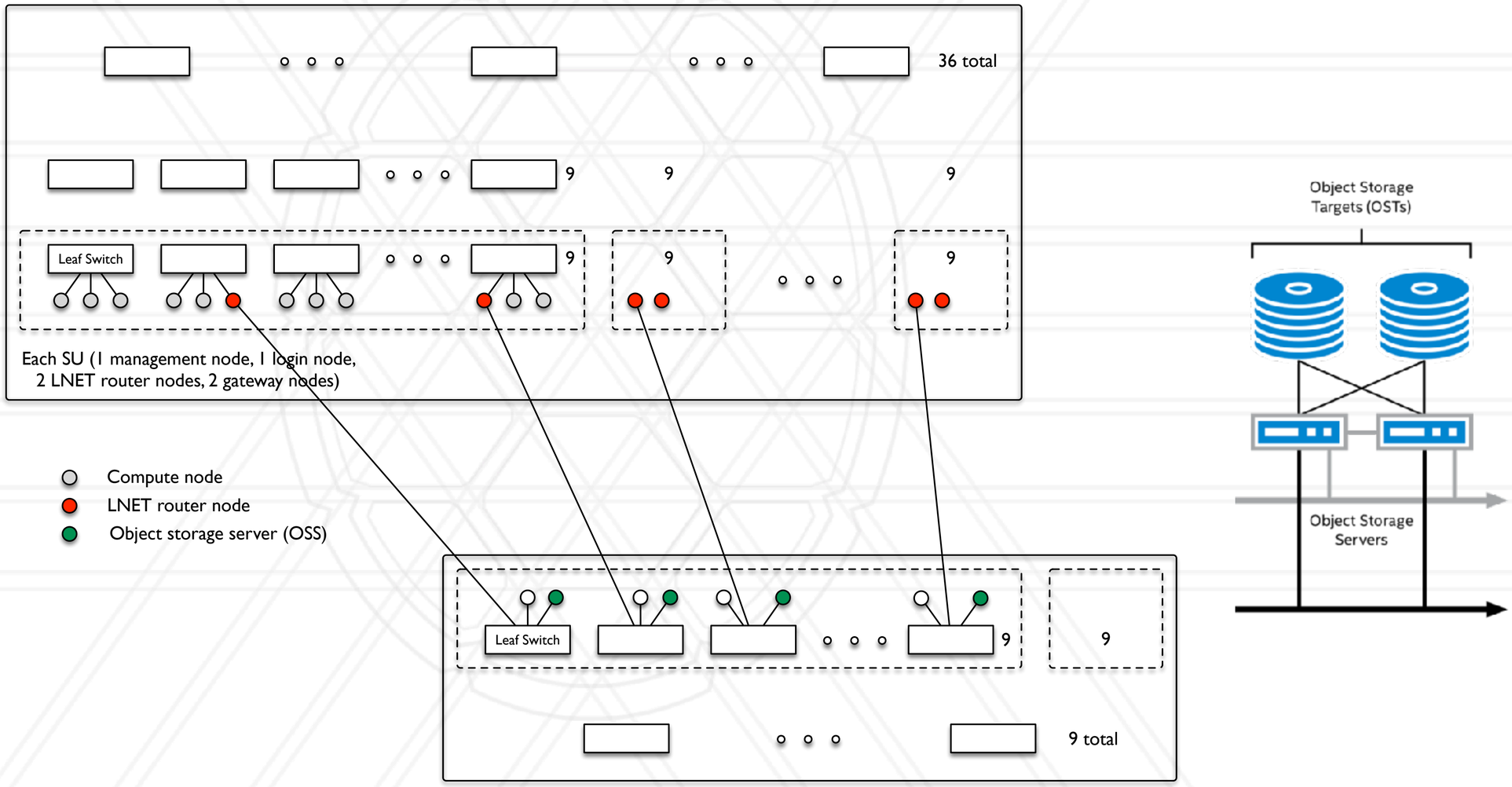
Message destination points: application dependent 1 micro sec - 10s of sec

Message origin points : destination, frequency, size, etc. determined by application 1 micro sec - 10s of sec

Parallel file system or I/O sub-system

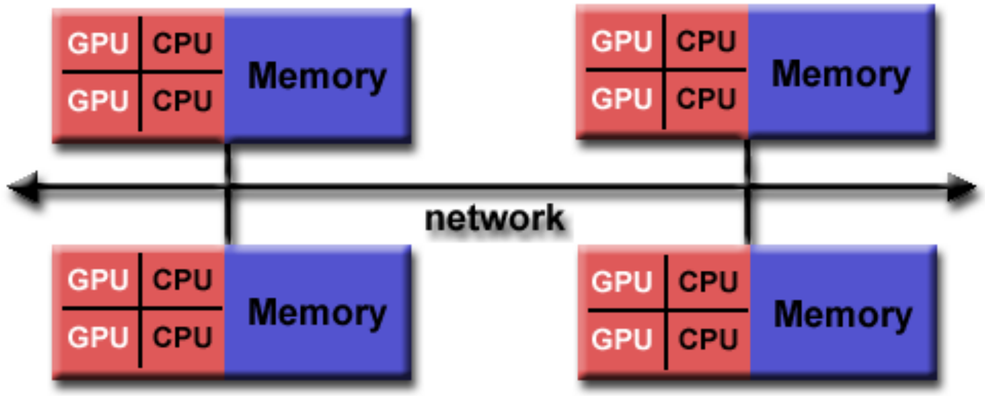


Parallel file system or I/O sub-system



Distributed memory programming models

- Each process only has access to its own local memory / address space
- When it needs data from remote processes, it has to send/receive messages



Message passing

- A distributed message passing program requires the creation of processes on different nodes
- Requires some bookkeeping
  - If GPUs are available: which process manages which GPU
  - Each process has to know the location of other processes

Several libraries provide message passing

- MPI: Message passing interface — an open standard
- NCCL, RCCL — NVIDIA and AMD specific libraries
- PyTorch distributed: higher-level wrapper
  - You can use MPI or NCCL as the backend

Important terms

- Point-to-point messages: between a pair of processes
- Collectives: involve a group of processes
- Blocking vs. non-blocking

send and recv: blocking calls

```
torch.distributed.send(tensor, dst=None, group=None, tag=0,
group_dst=None)

int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int
source, int tag, MPI_Comm comm, MPI_Status *status )

ncclResult_t ncclSend(const void* sendbuff, size_t count,
ncclDataType_t datatype, int peer, ncclComm_t comm, cudaStream_t
stream)



torch.distributed.recv(tensor, src=None, group=None, tag=0,
group_src=None)
```

isend and irecv: non-blocking calls

- Two parts to a non-blocking operation:
  - Posting: post the non-blocking operation
  - Completion: wait for its results at a later point in the program
- Return a distributed request object: opaque
- You can call two methods on this object:
  - `is_completed`: returns `True` if the operation is finished
  - `wait`: will block until the operation is finished

Collective operations

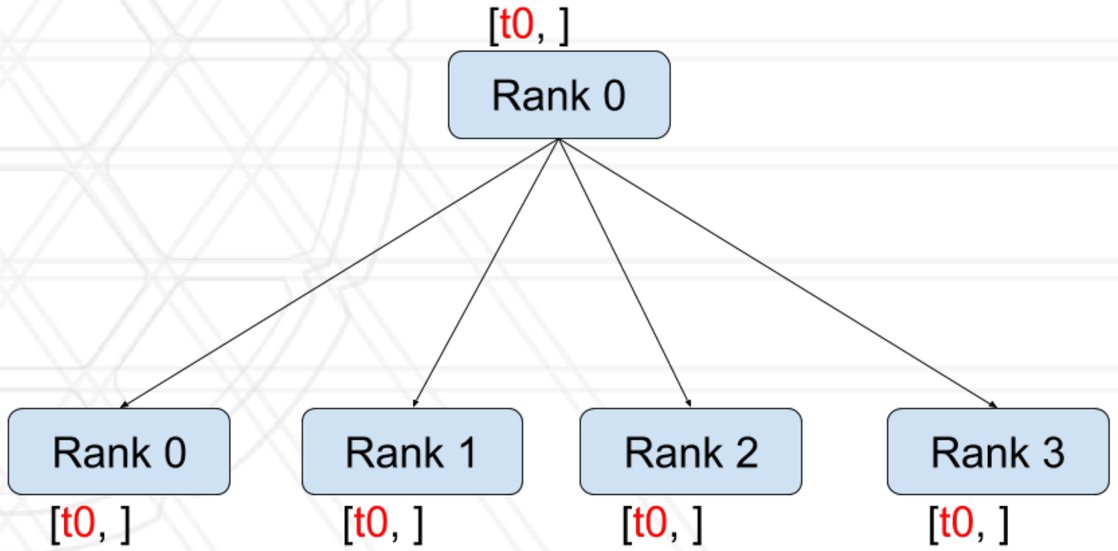
Collective operations

- `torch.distributed.barrier(group=None, async_op=False, device_ids=None)`
  - Blocks until all processes in the group enter this function

Collective operations

- `torch.distributed.barrier(group=None, async_op=False, device_ids=None)`
  - Blocks until all processes in the group enter this function

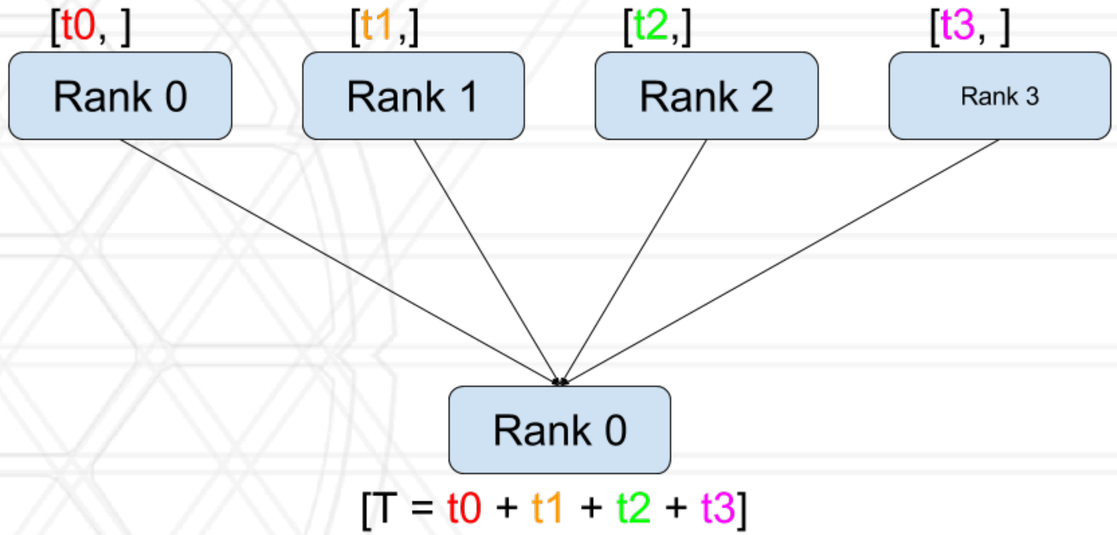
- `torch.distributed.broadcast`
  - Send data from root to all processes in the group



Collective operations

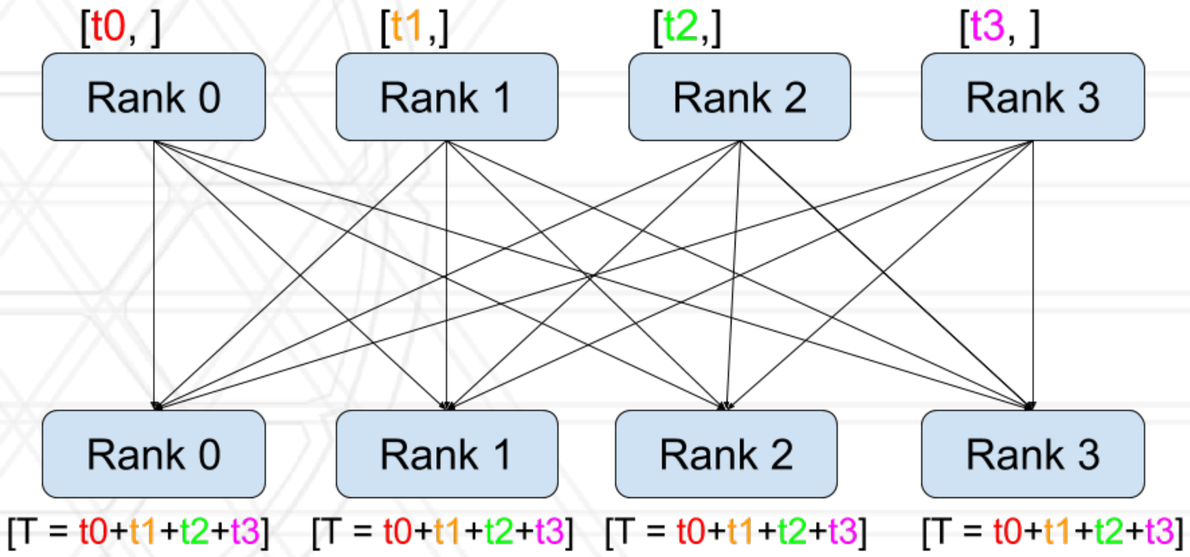
Collective operations

- `torch.distributed.reduce`
  - Reduce data from all processes to the root
  - sendbuf should be valid on all processes
  - Recvbuf only needs to exist on root



Collective operations

- `torch.distributed.reduce`
  - Reduce data from all processes to the root
  - sendbuf should be valid on all processes
  - Recvbuf only needs to exist on root

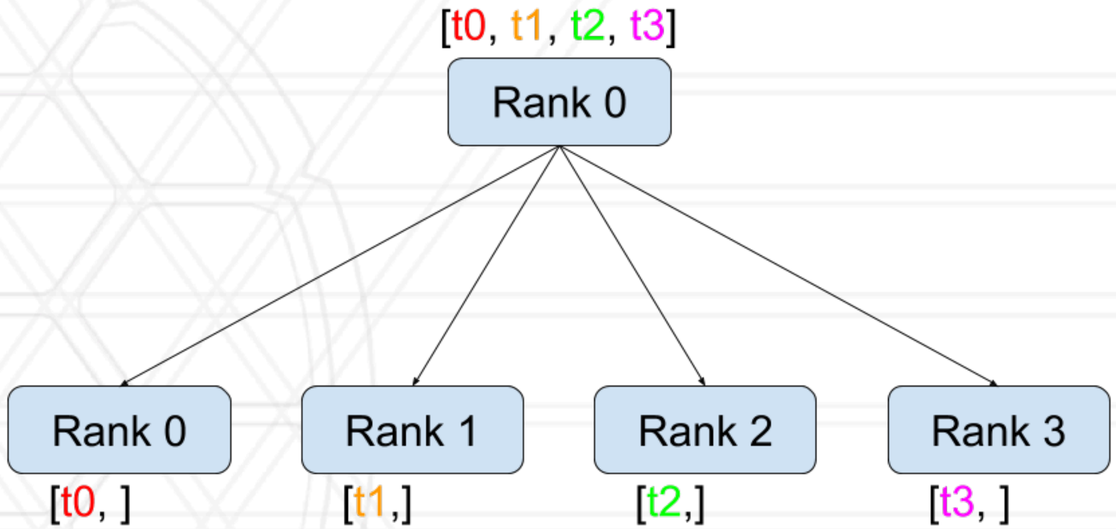


- `torch.distributed.allreduce`
  - Can be used to send the result back to **all** processes

Collective operations

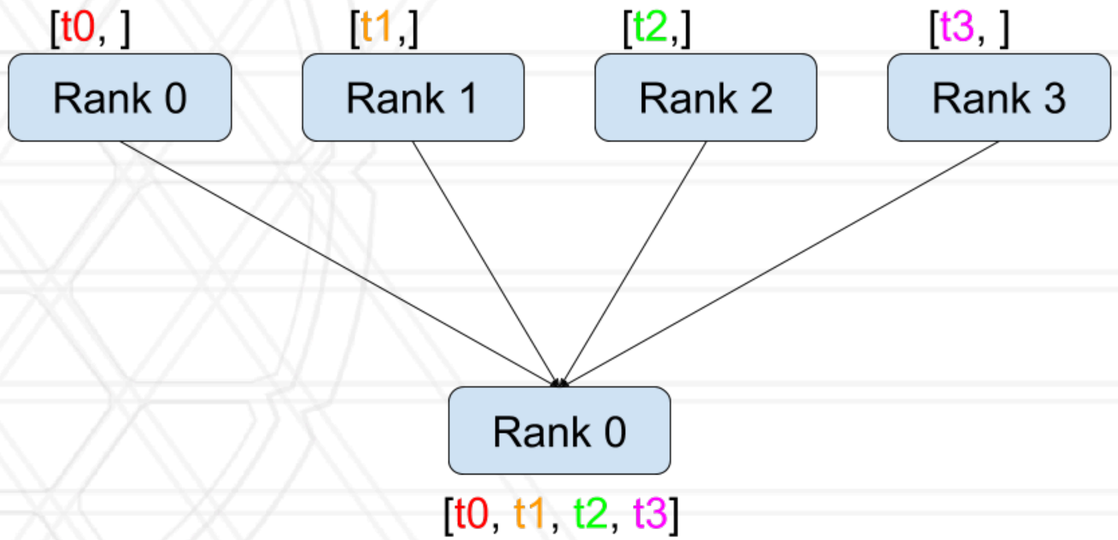
Collective operations

- `torch.distributed.scatter`
  - Send distinct data from root to all processes



Collective operations

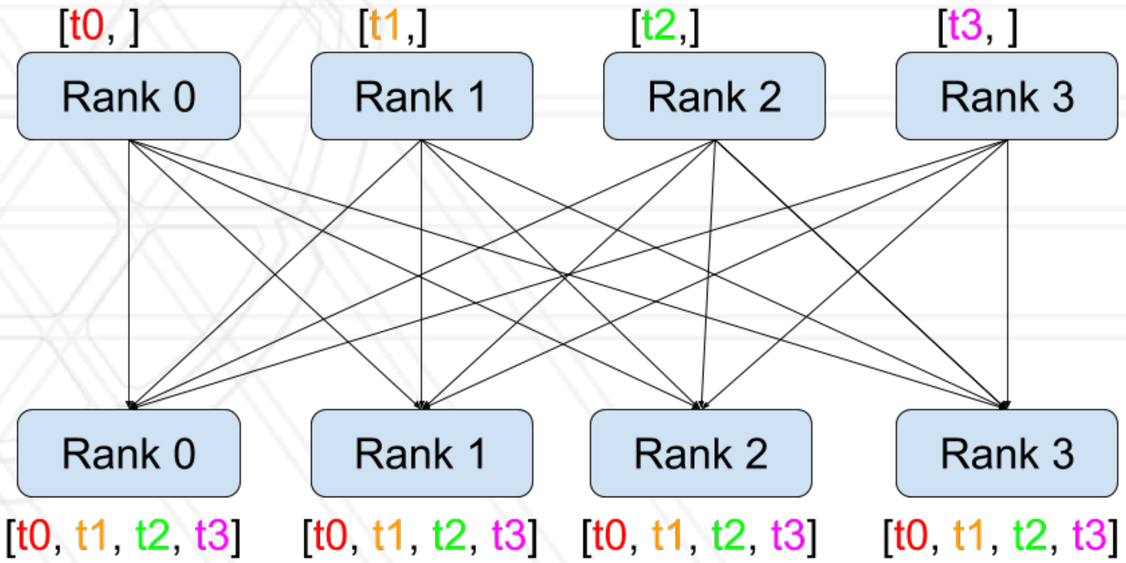
- `torch.distributed.scatter`
  - Send distinct data from root to all processes



- `torch.distributed.gather`
  - Gather distinct data from all processes to the root

Collective operations

- `torch.distributed.scatter`
  - Send distinct data from root to all processes
- `torch.distributed.gather`
  - Gather distinct data from all processes to the root
- `torch.distributed.allgather`



UNIVERSITY OF
MARYLAND