Parallel Networks and File Systems

Alan Sussman, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Midterm – grades posted
  - Regrade requests due by end of this week
  - Grades – Median: 79 Average: 75 Std. dev.: 18
- Assignment 3 due tomorrow, Apr. 12
  - Questions?
- Quiz 2: next week
- Assignment 4 on CUDA out next week

High-speed interconnection networks

- Typically supercomputers and HPC clusters are connected by low latency, high bandwidth networks
  - High bandwidth easier to obtain – parallelism
  - Low latency via fast hardware, RDMA, short software paths (minimize copies)
- The connections between nodes form different topologies
- Popular topologies:
  - Fat-tree: Leiserson in 1985, variant of CLOS network – provably efficient communication
  - Mesh and torus (2D, 3D)
  - Dragonfly

Network components

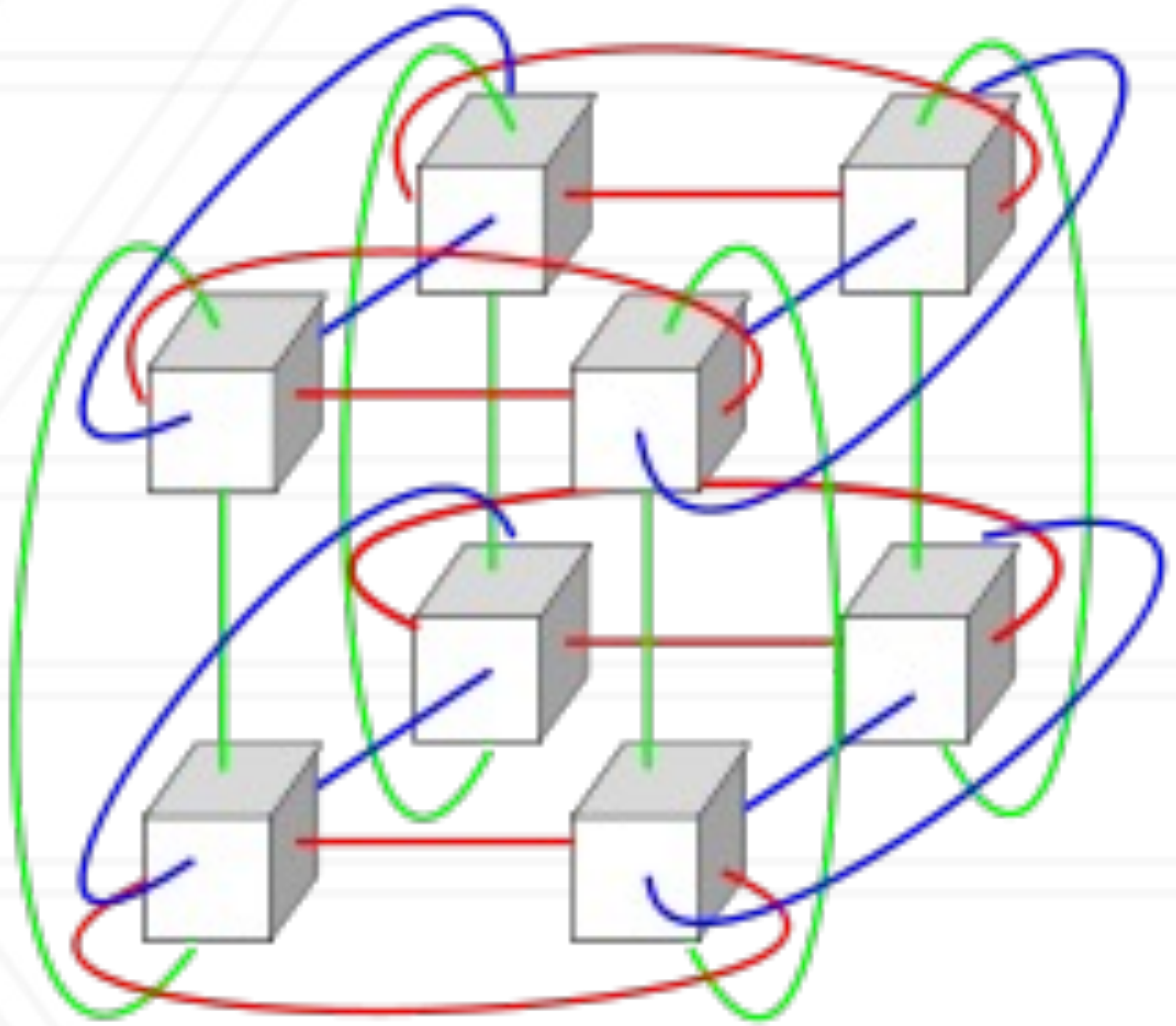
- Network interface controller or card
- Router or switch
- Network cables: copper or optical

Definitions

- Network diameter: length of the shortest path between the most distant nodes on the network.
- Radix: number of ports on a router

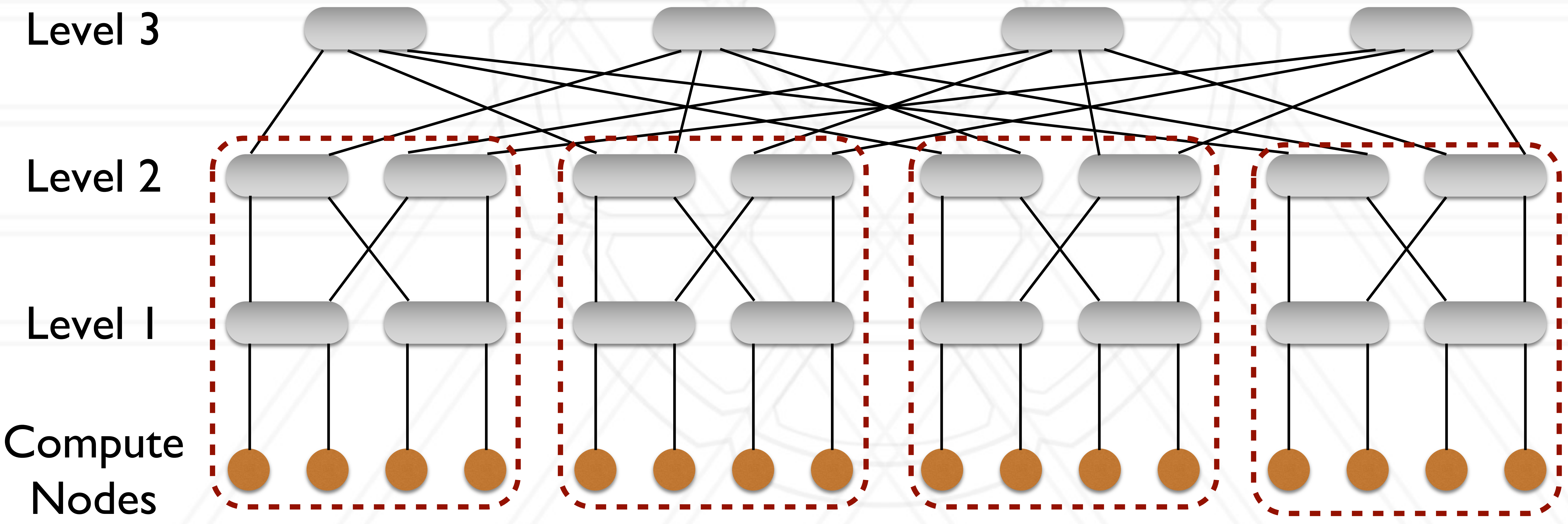
N-dimensional mesh / torus networks

- Each switch has a small number of nodes connected to it (often 1)
- Each switch has direct links to 2N switches where N is the number of dimensions
- Torus = wraparound links (no edges as in a mesh)
- Examples: IBM Blue Gene, Cray X* machines



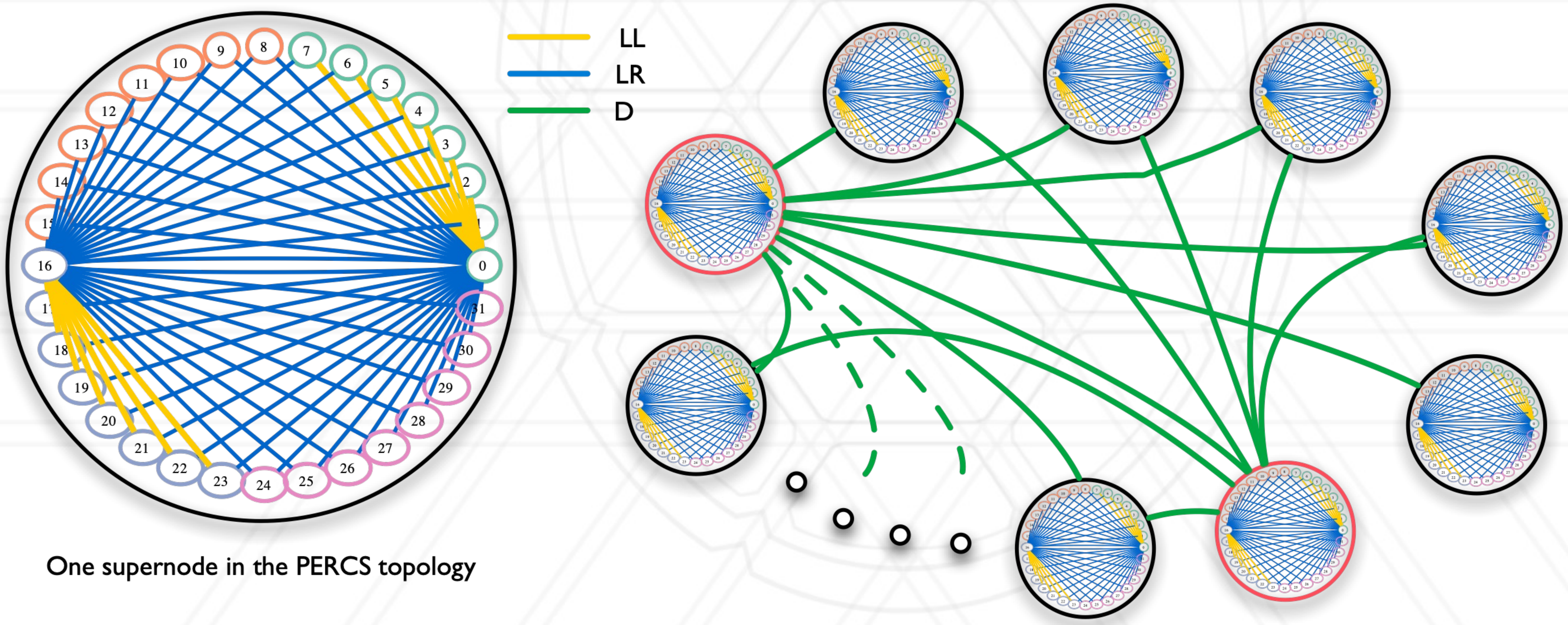
Fat-tree network

- Router radix = k, Number of nodes on each router = k/2
- A pod is a group of k/2 switches, Max. number of pods = k



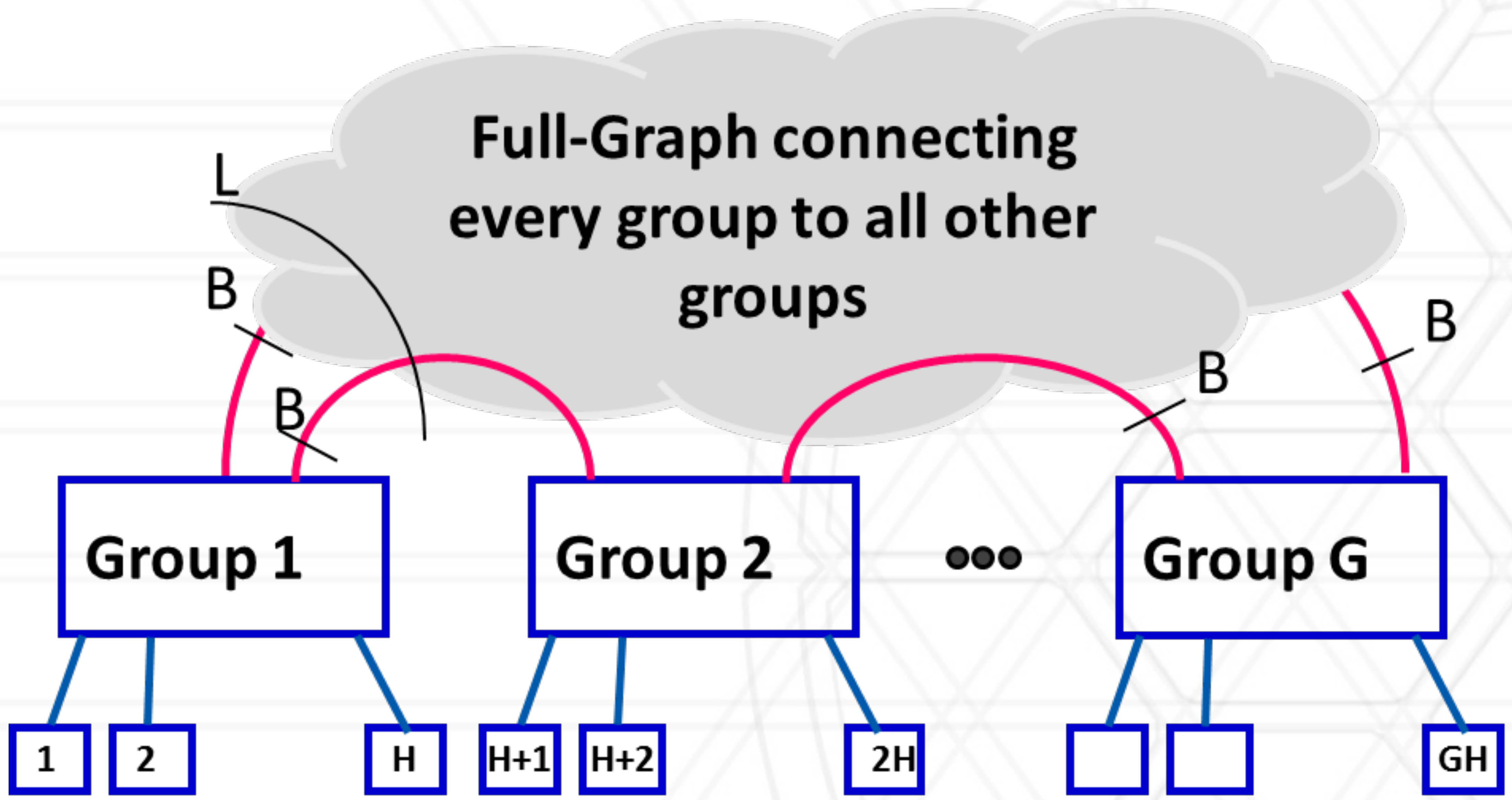
Dragonfly network

- Two-level hierarchical network using high-radix routers
- Low network diameter



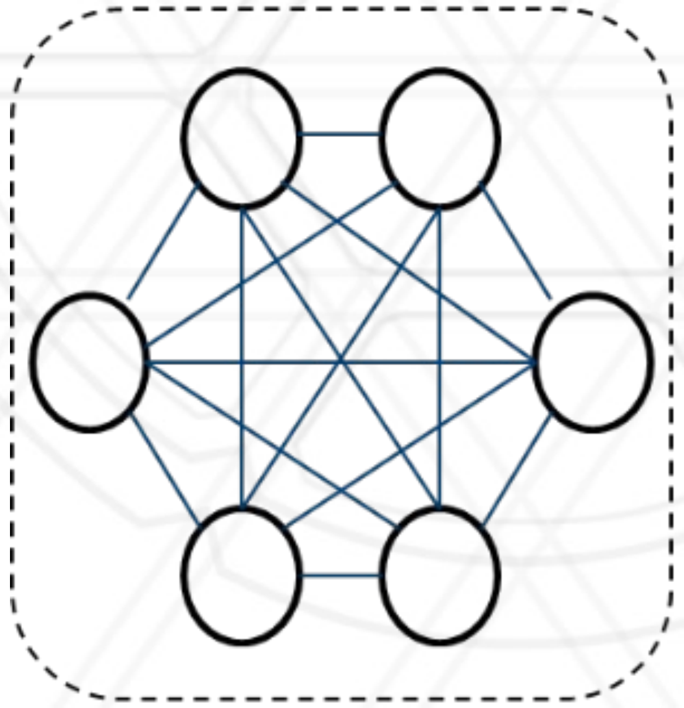
One supernode in the PERCS topology

Dragonfly network

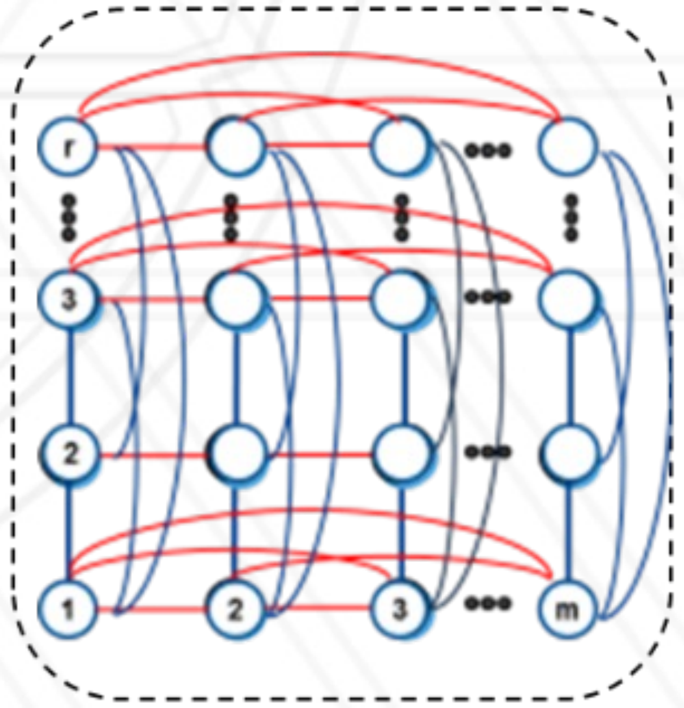


Full graph connecting groups/supernodes

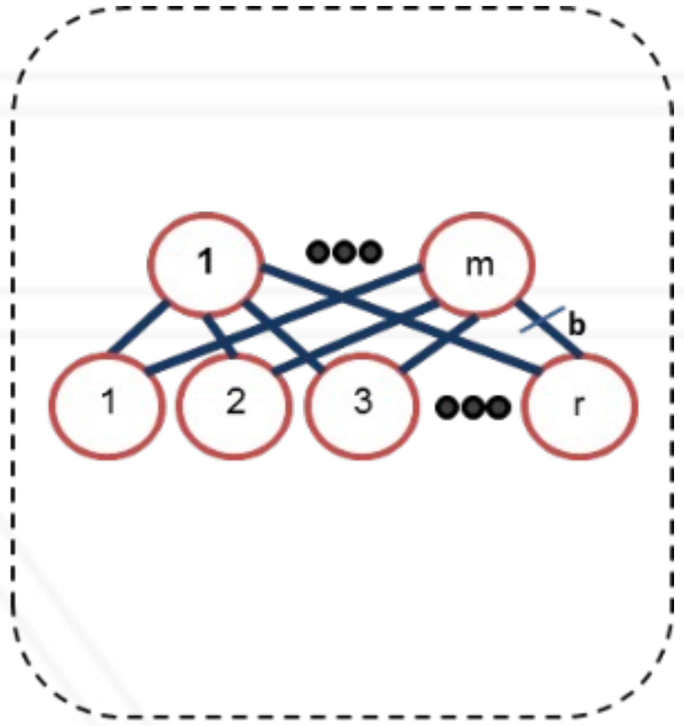
Various options for connecting nodes within a group



1D = Full Graph

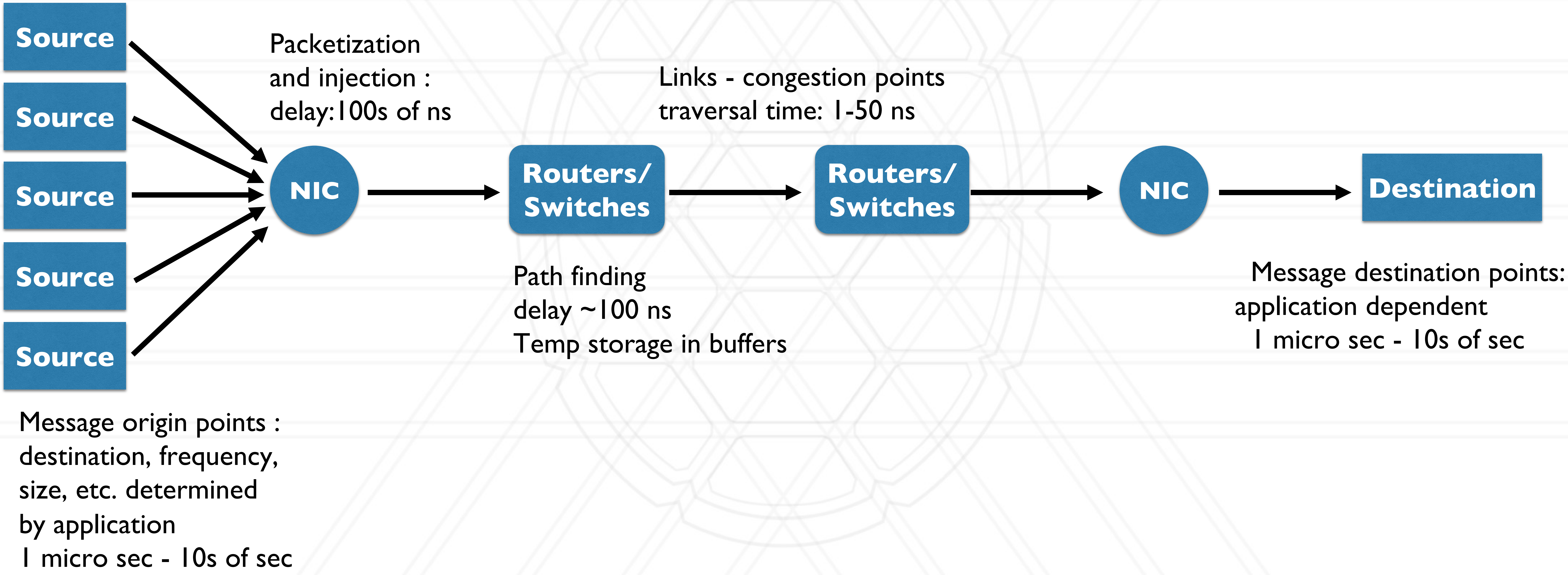


2D = GHC(2,m,r)



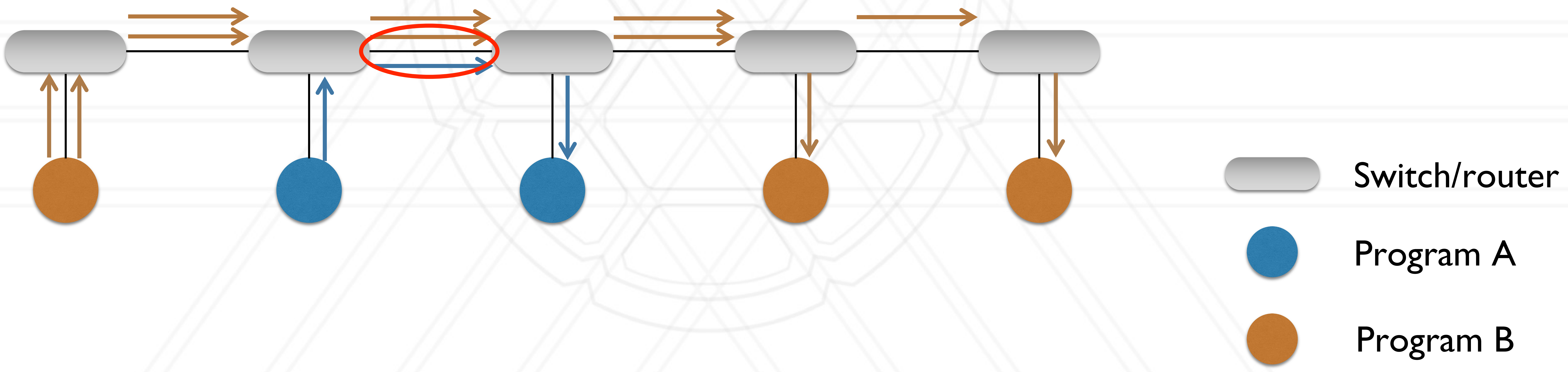
DF+ = Full Bipartite

Life-cycle of a message



Congestion due to network sharing

- Sharing refers to network flows of different programs using the same hardware resources: links, switches
- When multiple programs communicate on the network, they all suffer from congestion on shared links

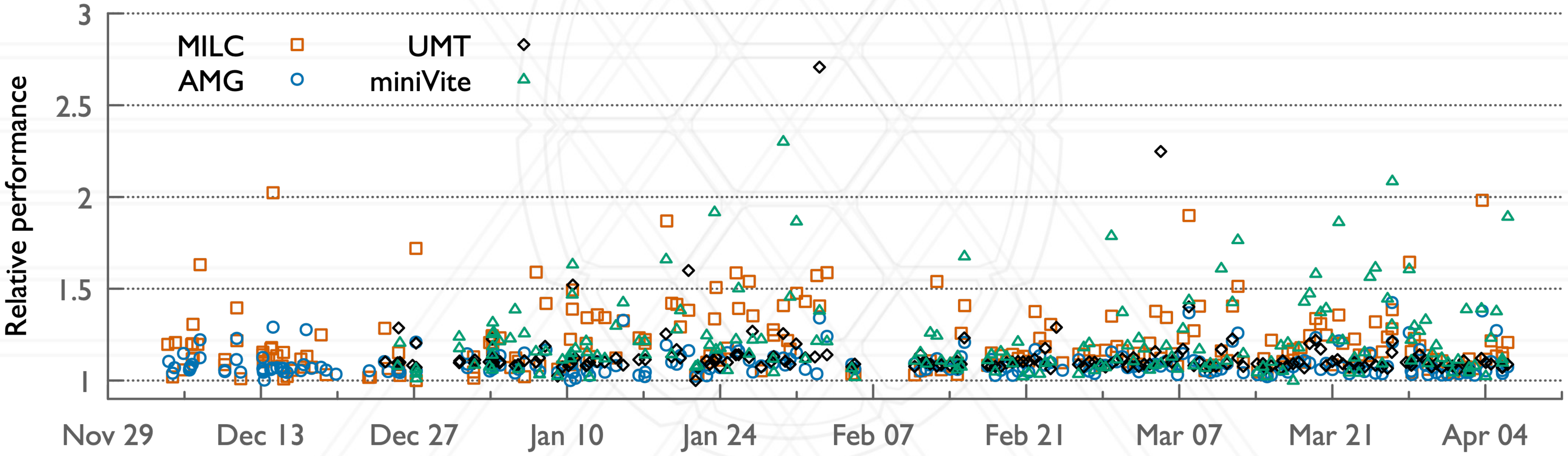


Routing algorithm

- Decides how a packet is routed between a source and destination switch
- Static routing: each router is pre-programmed with a routing table
  - Can change it at boot time
- Dynamic routing: routing can change at runtime
- Adaptive routing: adapts to network congestion

Performance variability

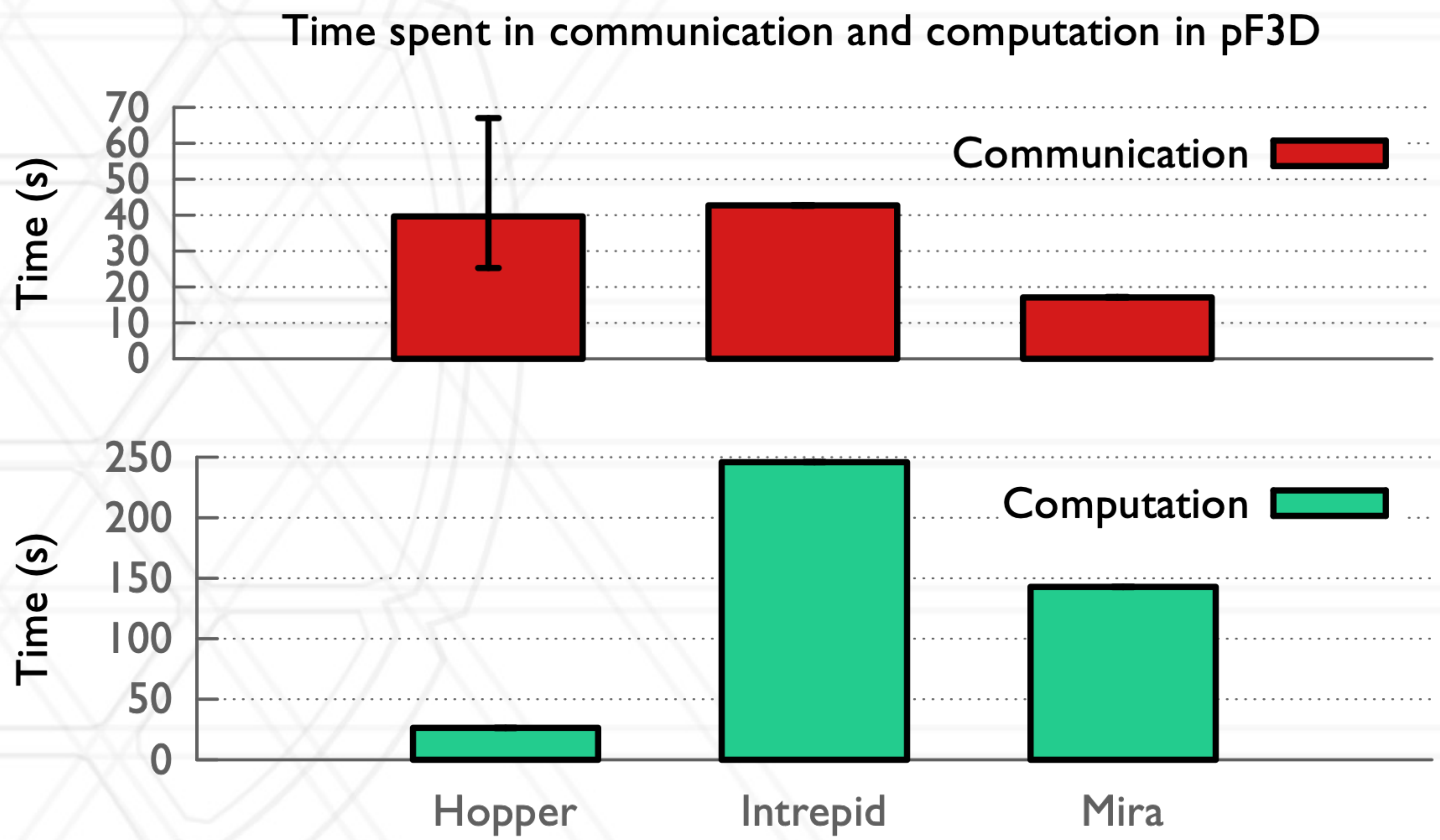
Performance of control jobs running the same executable and input varies as they are run from day-to-day on 128 nodes of Cori in 2018-2019



Bhatele et al. The case of performance variability on dragonfly-based systems, IPDPS 2020

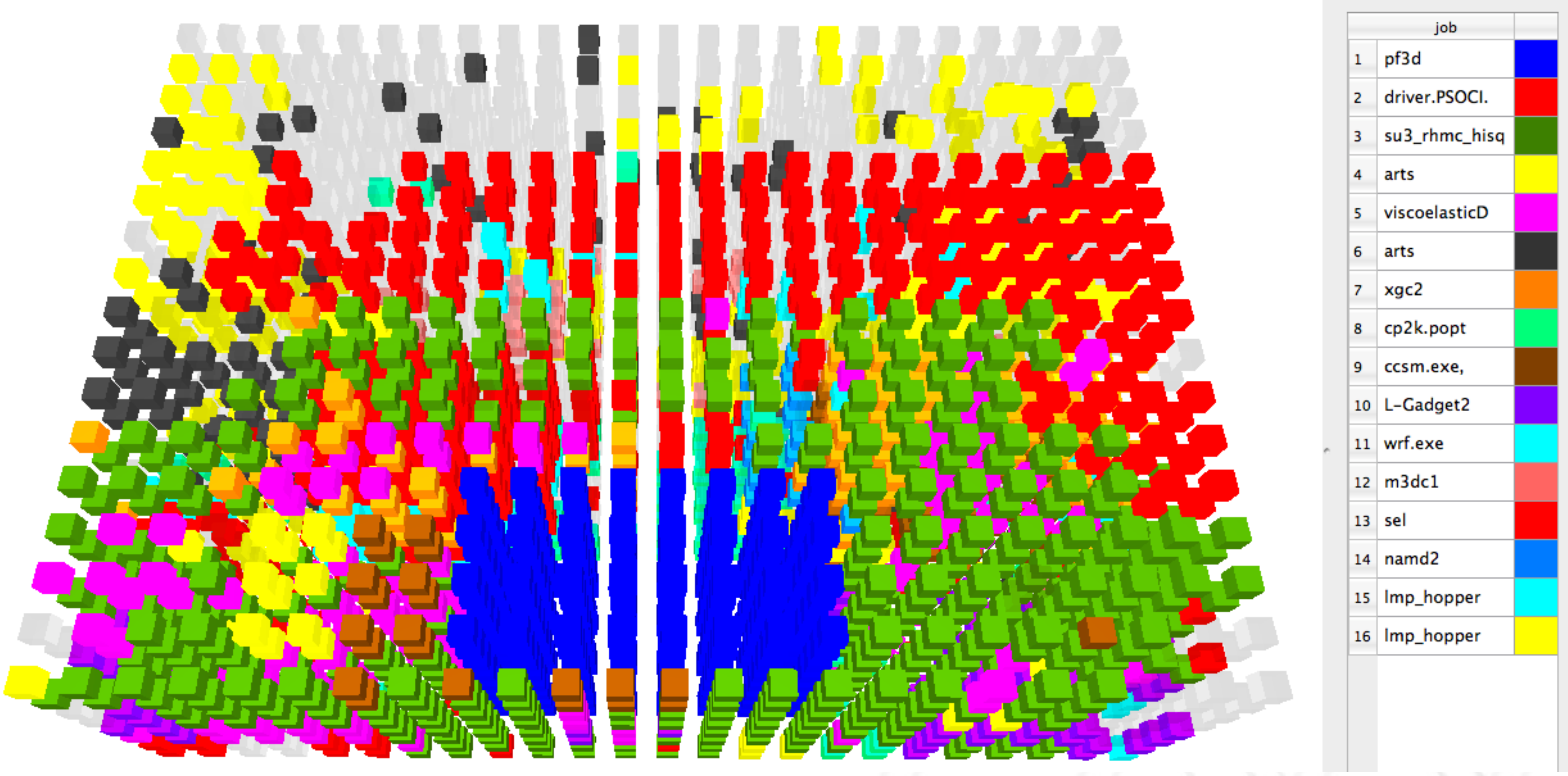
Performance variability due to congestion

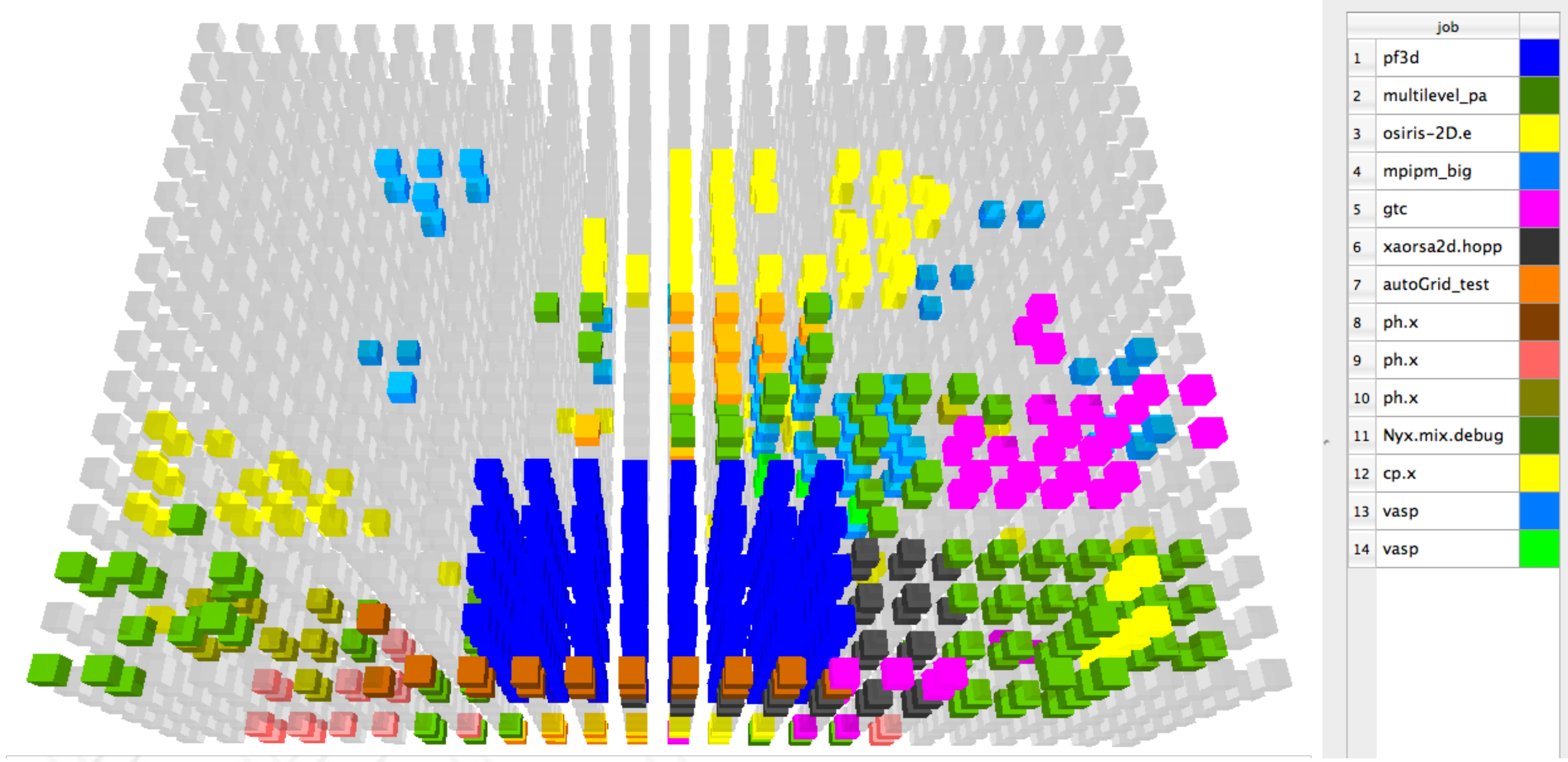
- No variability in computation time
- All of the variability can be attributed to communication performance
- Factors:
  - Placement of jobs
  - Contention for network resources



Bhatele et al. http://www.cs.umd.edu/~bhatele/pubs/pdf/2013/sc2013a.pdf

Impact of other jobs





April 11
MILC job in green

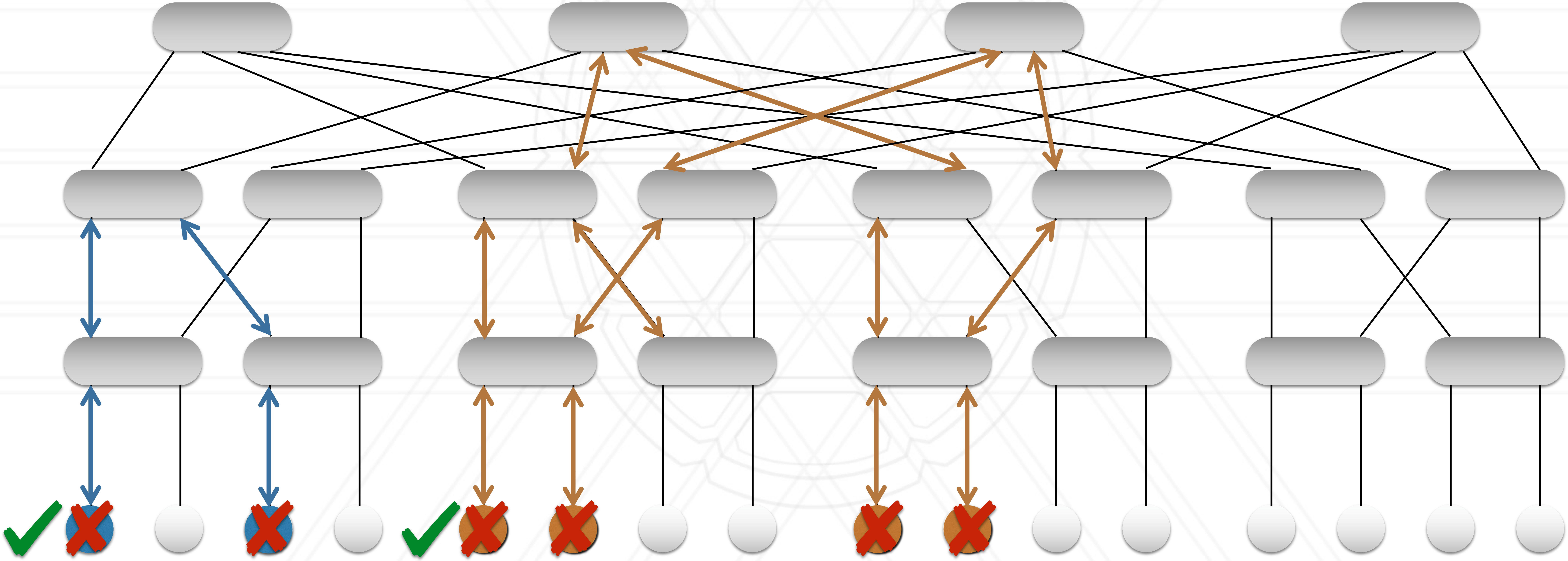
April 16
25% higher messaging rate

Different approaches to mitigating congestion

- Network topology aware node allocation
- Congestion or network flow aware adaptive routing
- Within a job: network topology aware mapping of processes to allocated nodes

topology-aware node allocation

Solution: allocate nodes in a manner that prevents sharing of links by multiple jobs while maintaining high utilization

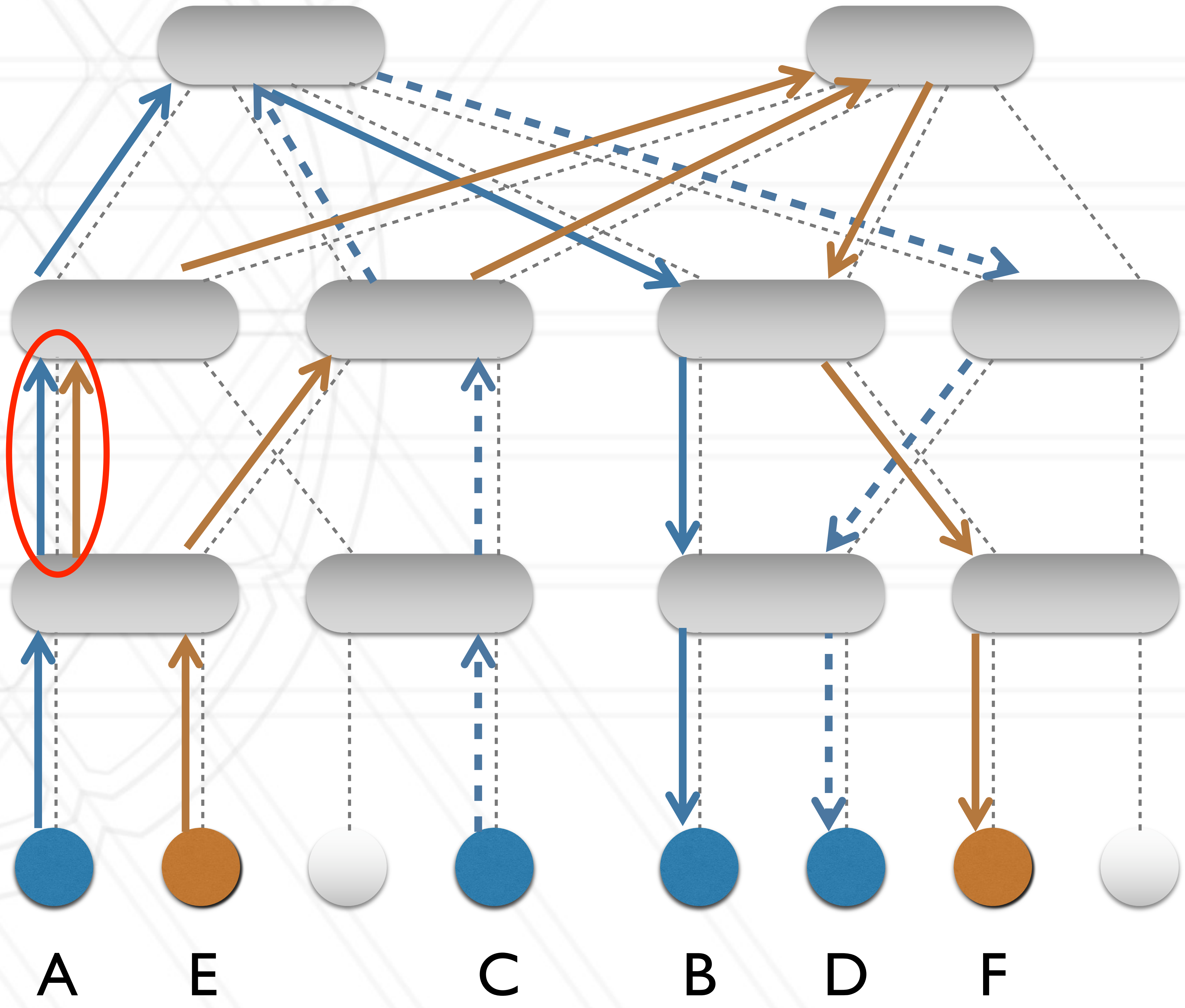


AFAR: adaptive flow aware routing

Solution: dynamically re-route traffic to alleviate hot-spots

Given: traffic for each pair of nodes in the system and the current routing

1. Calculate current load (network traffic) on all links in system
2. Find link with maximum load
3. If maximum > threshold, re-route one flow crossing that link to an under-utilized link
4. Repeat from 1. using new routing



Topology-aware mapping

- Within a job allocation, map processes to nodes intelligently
- Inputs: application communication graph, machine topology
- Graph embedding problem (NP-hard)
- Many heuristics to come up with a solution
- Can be done within a load balancing strategy

Introduction to Parallel Computing (CMSC416)

Parallel Networks and File Systems

Alan Sussman, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Assignment 3 late deadline tonight
  - Questions?
- Quiz 2: next week
- Assignment 4 on CUDA out on Tuesday
  - Due in 2 weeks, May 5

When do parallel programs perform I/O?

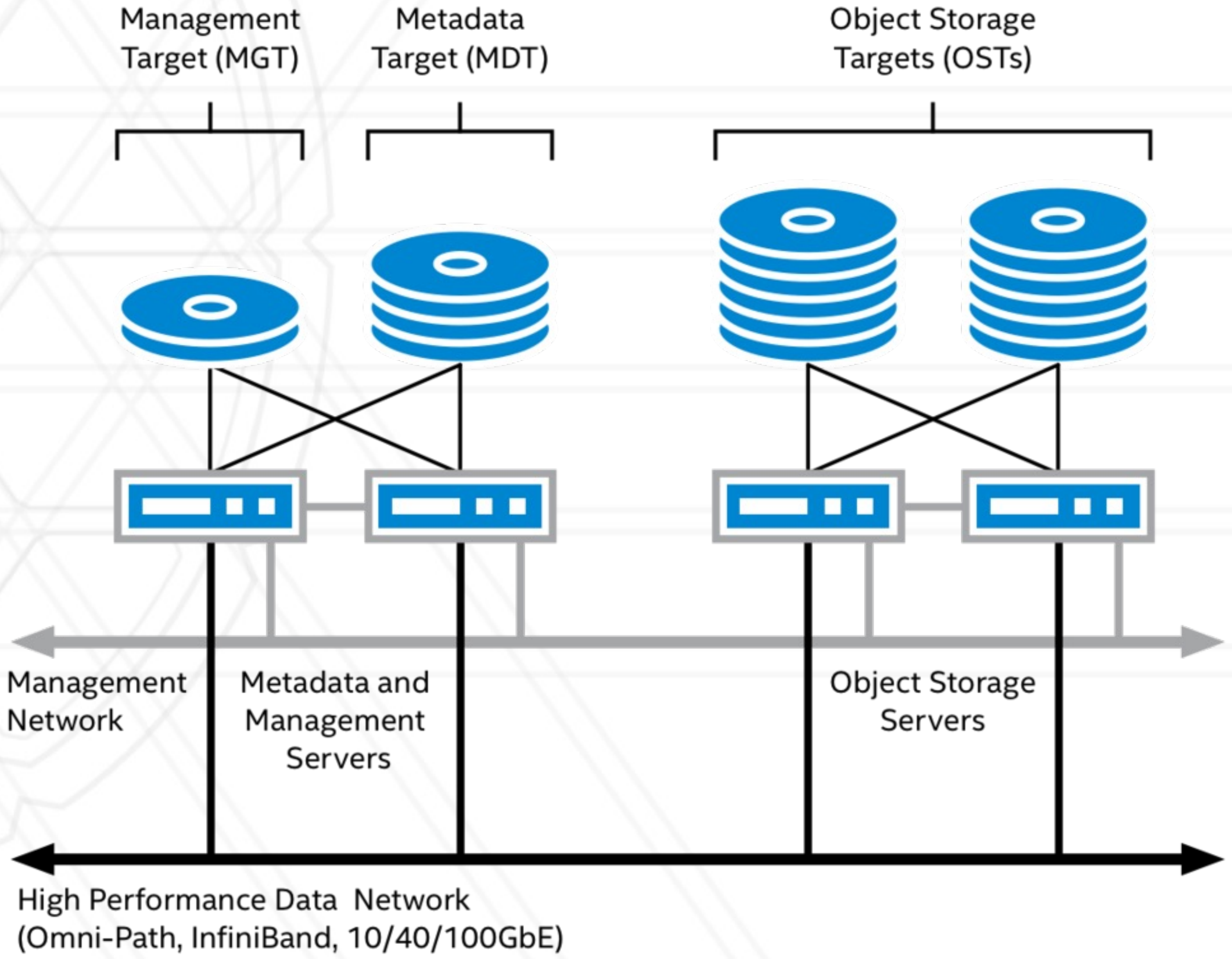
- Reading input datasets
- Writing numerical output
- Writing checkpoints

Non-parallel I/O

- Designated process does I/O
- All processes send data to/receive data from that one process
- Not scalable

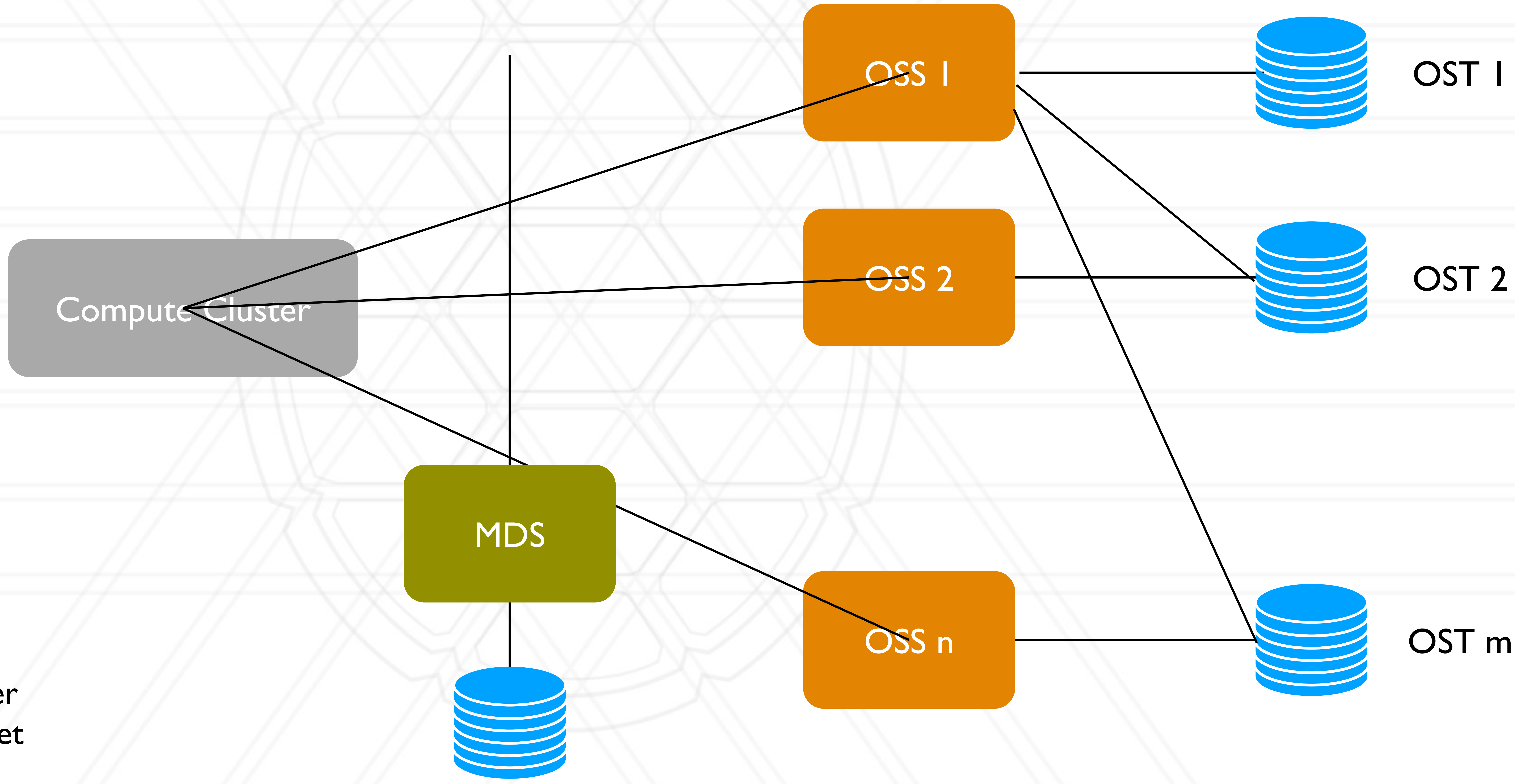
Parallel filesystem

- Home directories and scratch space are typically on a parallel file system
- Mounted on all login and compute nodes
- Also referred to as I/O sub-system



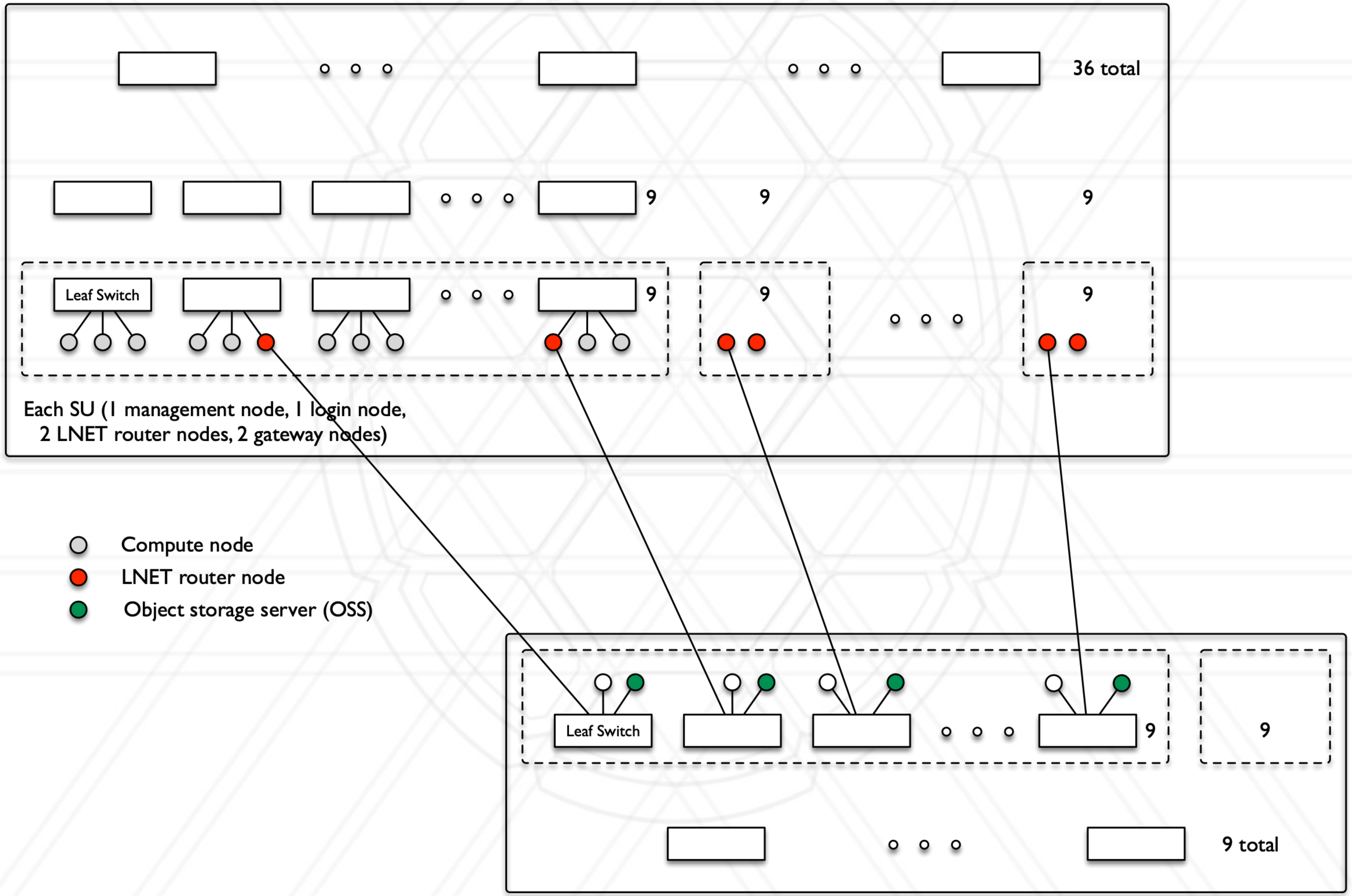
https://wiki.lustre.org/Introduction_to_Lustre

Parallel filesystem



MDS = Metadata Server
OSS = Object Storage Server
OST = Object Storage Target

Links between cluster and filesystem



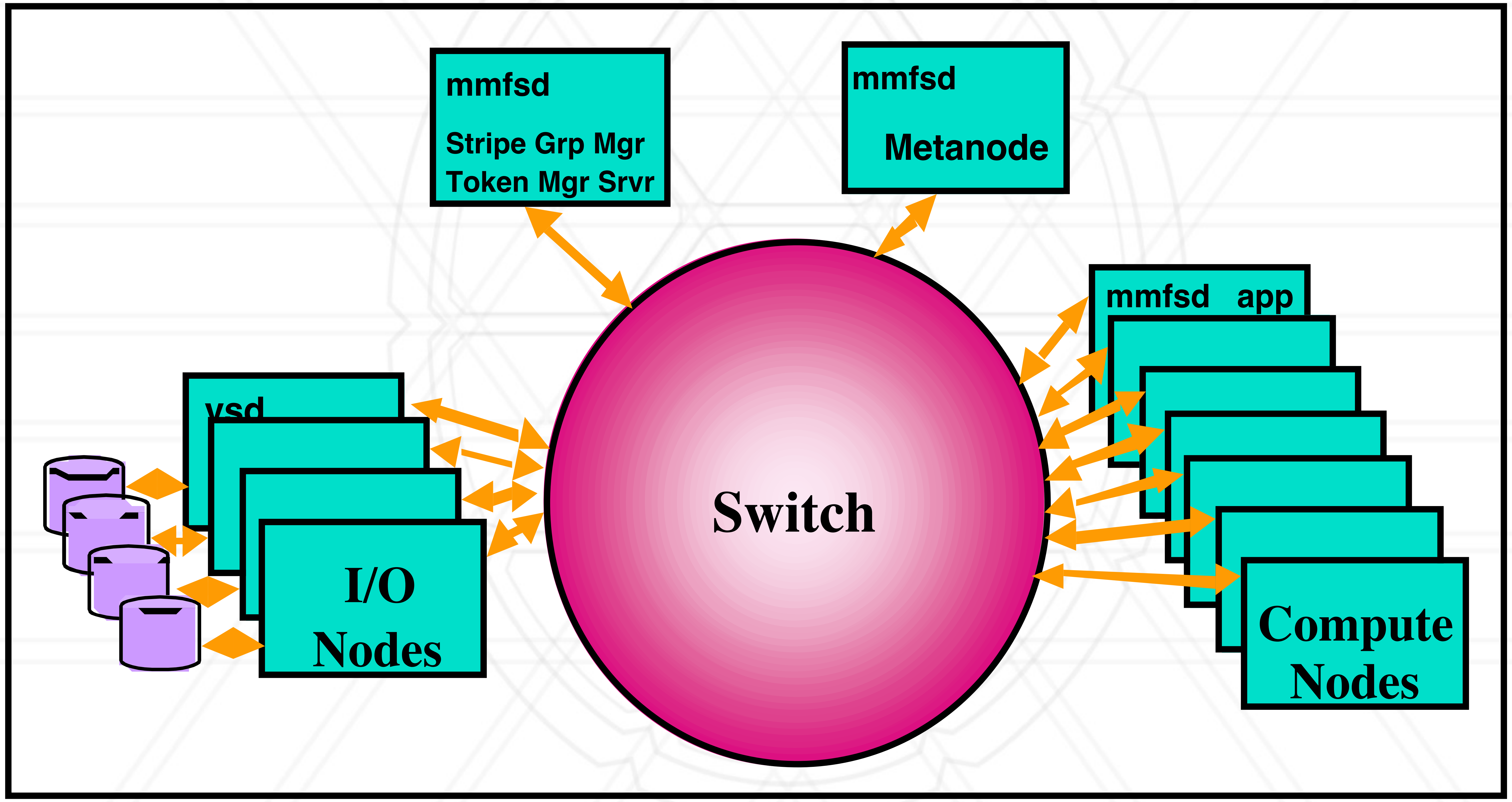
Different parallel filesystems

- Lustre: open-source (lustre.org)
- BeeGFS: community supported (beegfs.io)
  - Commercial support too
- GPFS: General Parallel File System from IBM, now called Spectrum Scale
- PVFS: Parallel Virtual File System

Example: GPFS

- Designed to support high throughput parallel applications, including multimedia
  - well suited for scientific computations
  - still used in some of Top 500 supercomputers
- Main idea is to use parallel I/O to increase performance and scale to large configurations
  - increase bandwidth by spreading reads and writes (even to a single file) across multiple disks, especially for sequential access
  - avoid the "one file per parallel process" model, or sending all I/O through one node
  - use internal high performance switch, plus separate I/O nodes, for I/O from parallel processes running on compute nodes
  - files can be both striped across multiple I/O nodes, and across multiple disks in each I/O node

GPFS architecture



GPFS details

- Each node runs a demon (mmfsd) to provide I/O services
  - one demon runs a metanode service, to serve file metadata (ownership, permissions), and inode/directory updates
  - one demon runs a stripe group manager, to keep track of available disks
  - a token manager to synchronize concurrent access to files, maintain consistency across caches
  - each application node demon mounts a file system and performs file accesses (through switch, to I/O nodes that have the disks with the data)
- Client-side caching
  - inside Virtual Shared Disk (VSD) layer in kernel (server is on I/O nodes)
  - pagepool in each application node's memory
  - read-ahead discovers sequential and constant stride access patterns
  - write behind allows application to continue after data copied into pagepool – cost is extra copy to pagepool

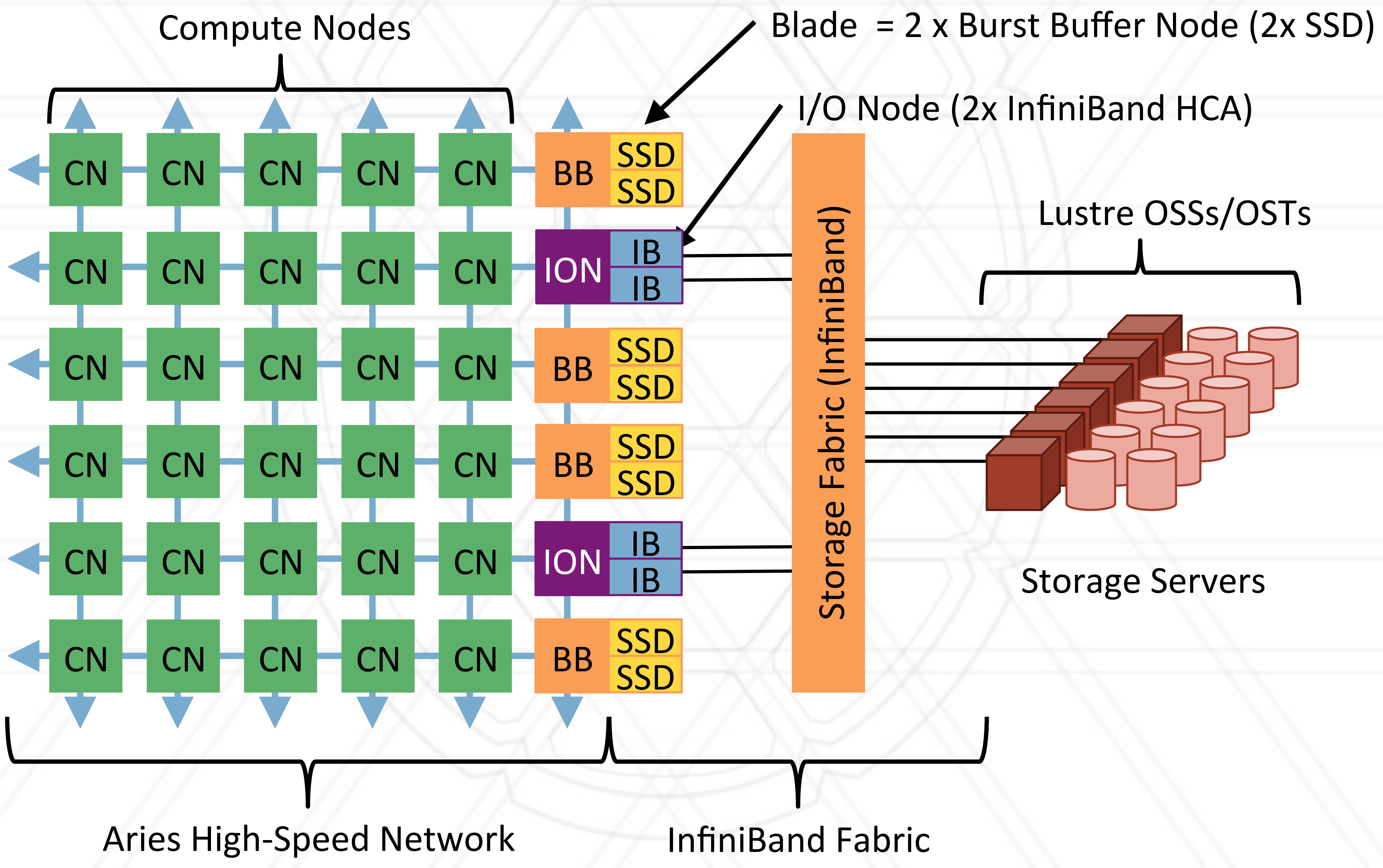
Tape drive

- Store data on magnetic tapes
- Used for archiving data
- Use robotic arms to access the right tape: https://www.youtube.com/watch?v=d-eWDuEo-3Q

Burst buffer

- Fast, intermediate storage between compute nodes and the parallel filesystem
  - Typically some form of non-volatile (NVM) memory, for persistence, high capacity, and speed (reads and writes)
  - Slower, but higher capacity, than on-node memory (DRAM)
  - Faster, but lower capacity, than disk storage on parallel file system
- Two designs:
  - Node-local burst buffer
  - Remote (shared) burst buffer
  - Either way, looks like a separate filesystem to the compute nodes

Burst buffer in DOE NERSC Cori



Burst buffer use cases

- Main target is high bandwidth checkpoint-restart
  - Long-running applications periodically save their state, in case of a failure
- But several other scenarios at NERSC
  - Complex I/O patterns with high IOPs – e.g., non-sequential table lookups
  - Out-of-core applications
  - Workflows – to couple multiple applications – e.g., store data between simulation and analysis components, or for analysis/visualization (in-situ, in-transit, or interactive)

I/O libraries

- High-level libraries: HDF5, NetCDF
  - Self-describing data formats w/associated libraries
  - Metadata stored in same file with the data
  - Data is usually multi-dimensional arrays
  - Also interoperate with parallel filesystems
- Middleware: MPI-IO
  - MPI-like I/O interface for collective I/O
- Low-level: POSIX IO
  - Standard Unix/Linux I/O interface

Different I/O patterns

- One process reading/writing all the data
  - Not scalable, but simple
- Multiple processes reading/writing data from/to shared file
  - What parallel filesystems target for high I/O bandwidth
- Multiple processes reading/writing data from/to different files
- Performance depends upon number of readers/writers (how many processes/threads), file sizes, filesystem organization, etc.

I/O profiling tools

- Darshan (https://www.mcs.anl.gov/research/projects/darshan/)
  - Lightweight profiling tool from Argonne National Lab, still under active development (as of Dec. 2022)
  - Captures an accurate picture of application I/O behavior, including properties such as patterns of access within files, with minimum overhead
- Recorder (https://github.com/uiuc-hpc/Recorder)
  - Library for understanding I/O activity in HPC applications
    - tracing framework that can capture I/O function calls at multiple levels of the I/O stack, including HDF5, MPI-IO, and POSIX I/O
  - Research prototype from UIUC

UNIVERSITY OF
MARYLAND