



Parallel Networks and File Systems

Abhinav Bhatele, Department of Computer Science



UNIVERSITY OF
MARYLAND

Announcements

- Extra credit assignments do not have an automatic due date extension policy

High-speed interconnection networks

- Typically supercomputers and HPC clusters are connected by low latency and high bandwidth networks
- The connections between nodes form different topologies
- Popular topologies:
 - Fat-tree: Charles Leiserson in 1985
 - Mesh and torus networks
 - Dragonfly networks

Network components

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

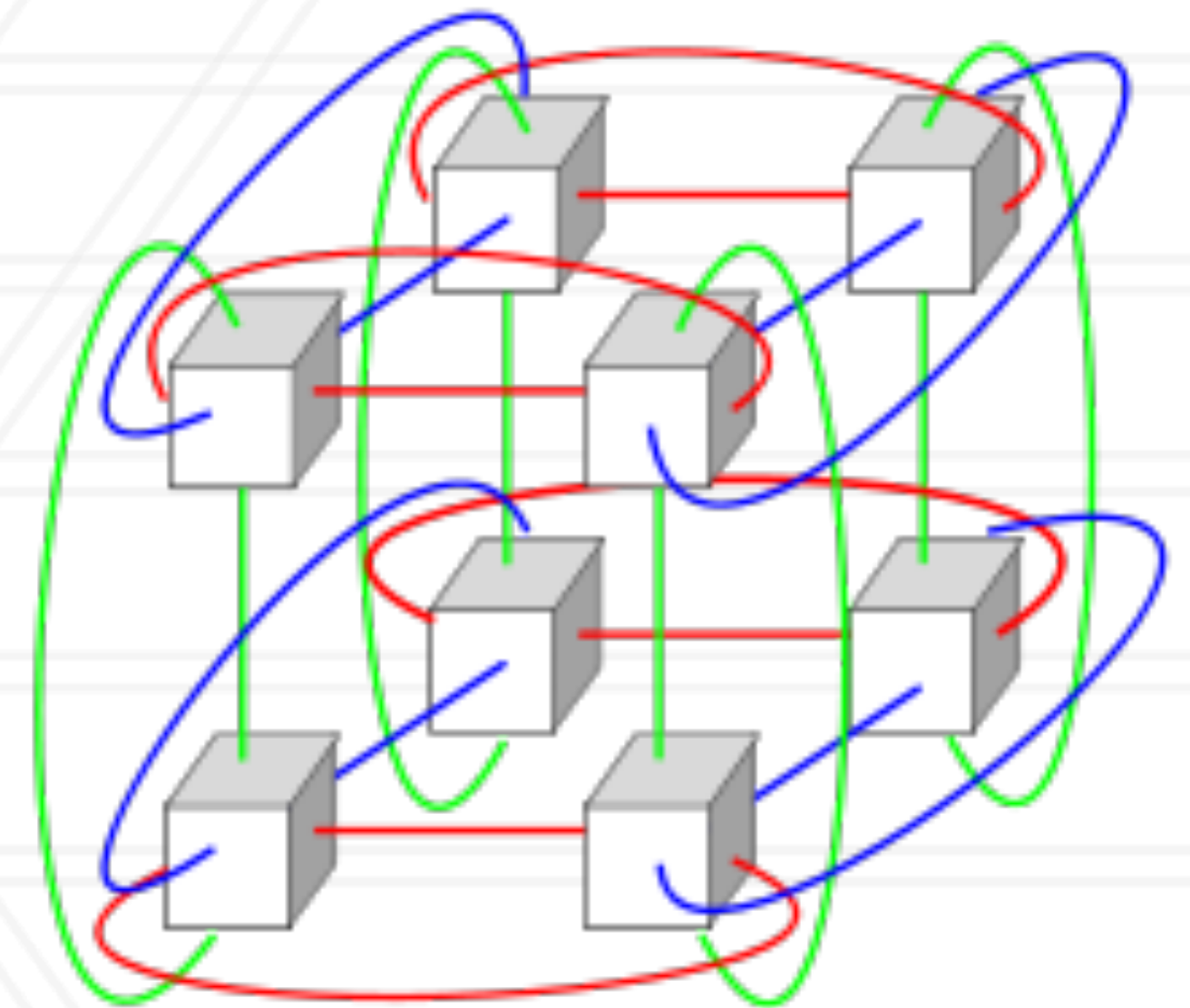


Definitions

- Network hops/Distance: Number of links a message must travel between the source and destination switch
- Network diameter: length of the shortest path between the most distant switches on the network.
 - Longest shortest path between any switch-pair
 - Gives an idea of the worst case latency on a 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 one or two)
- Each switch has direct links to $2n$ switches where n is the number of dimensions
- Torus = mesh + wraparound links
- Examples: IBM Blue Gene, Cray X* machines



Network properties of mesh/torus

- Let's say the number of switches is s , and number of nodes per switch is a small constant, c
- Diameter of 1-D mesh: $s - 1$
- Diameter of 1-D torus: $\lfloor \frac{s}{2} \rfloor$
- Diameter of d -dimensional mesh: $(\sqrt[d]{s} - 1) \times d$
- Diameter of d -dimensional torus: $\lfloor \frac{\sqrt[d]{s}}{2} \rfloor \times d$
- Maximum number of nodes: $c \times s$

Fat-tree network

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

Fat-tree network

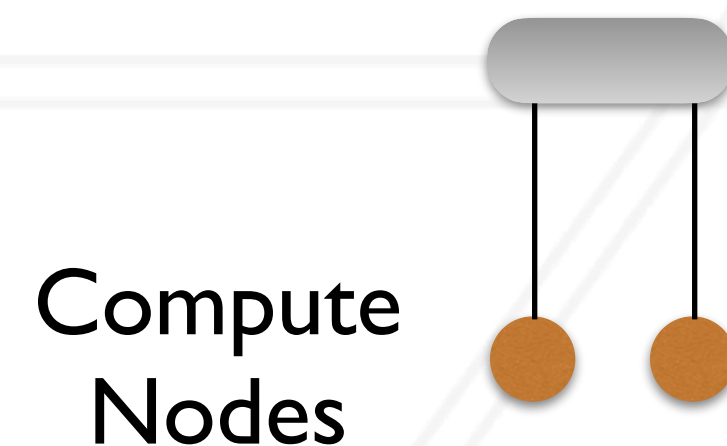
- Router radix = k , Number of nodes on each router = $k/2$
- A pod is a group of $k/2$ switches (at each level), Max. number of pods = k

Compute
Nodes



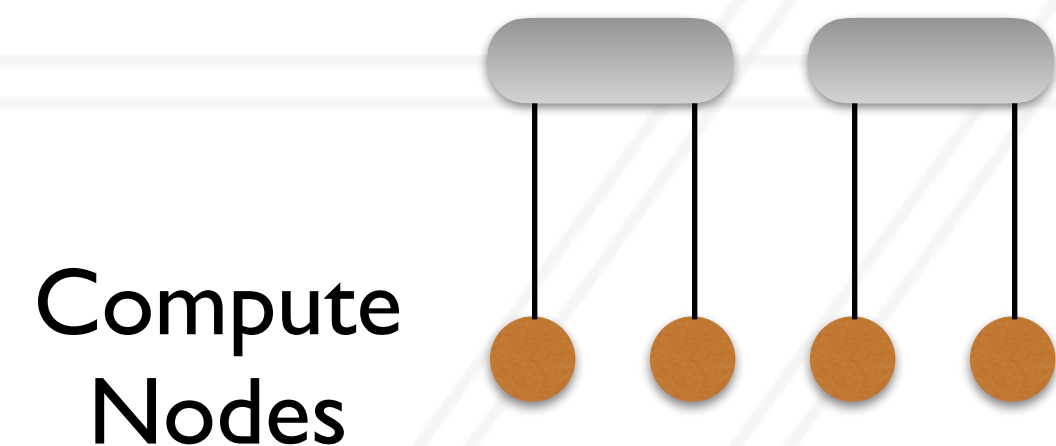
Fat-tree network

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



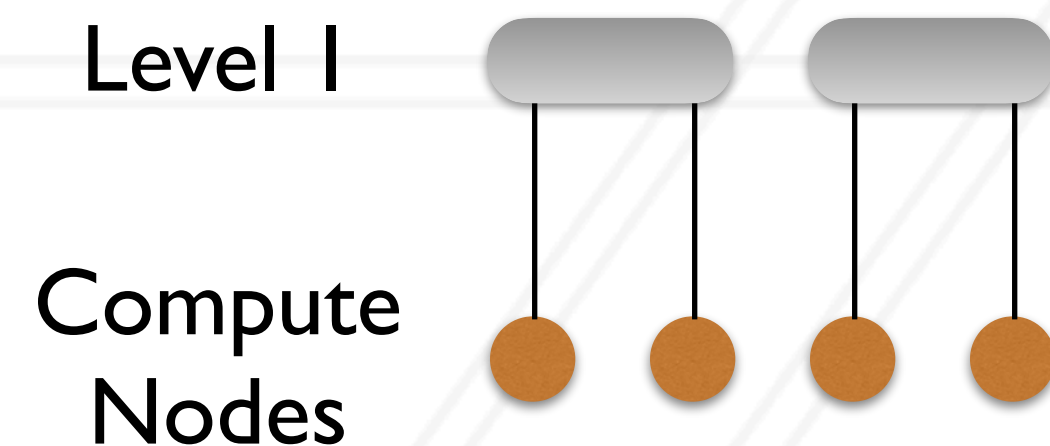
Fat-tree network

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



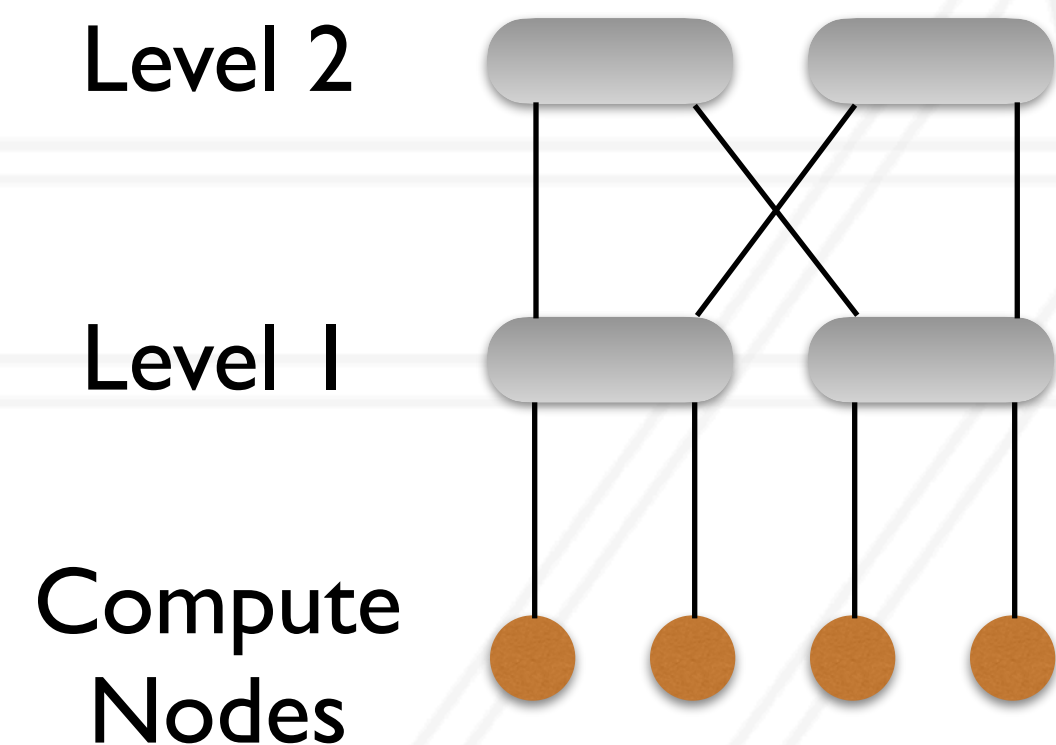
Fat-tree network

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



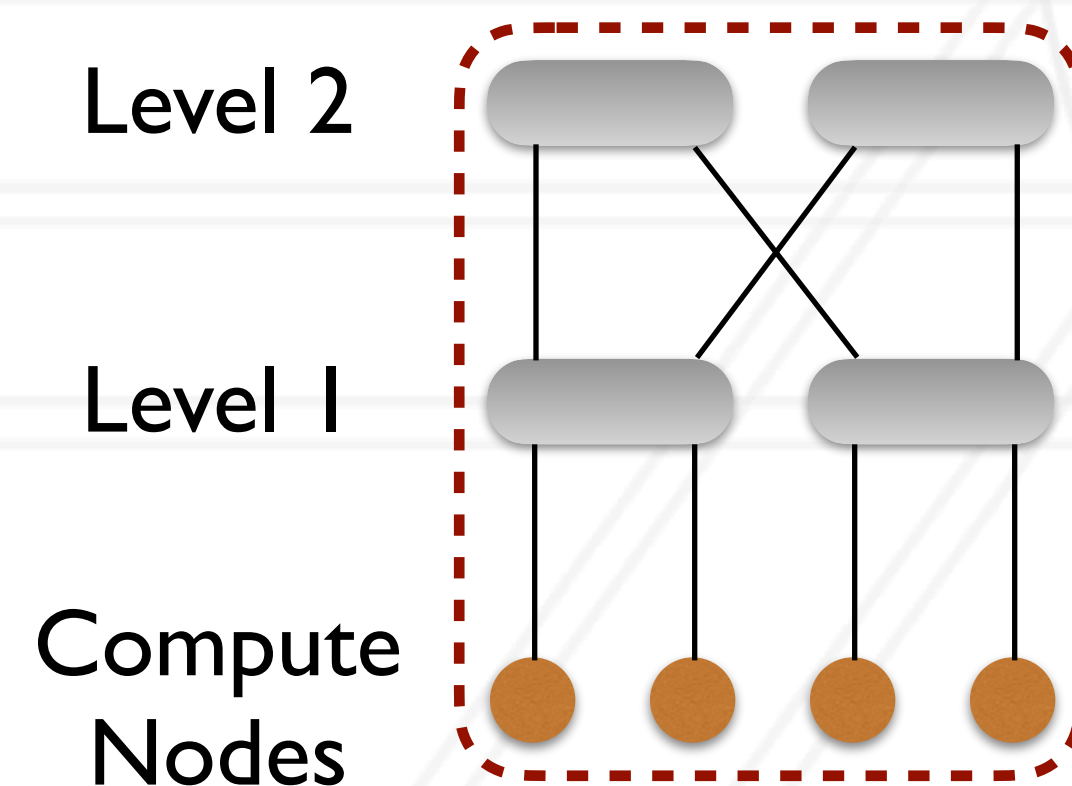
Fat-tree network

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



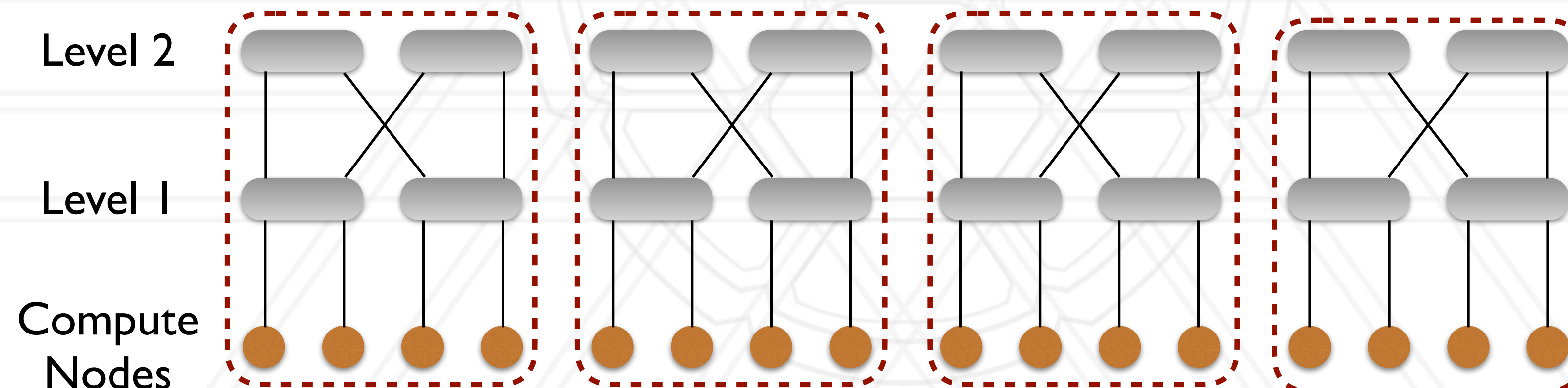
Fat-tree network

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



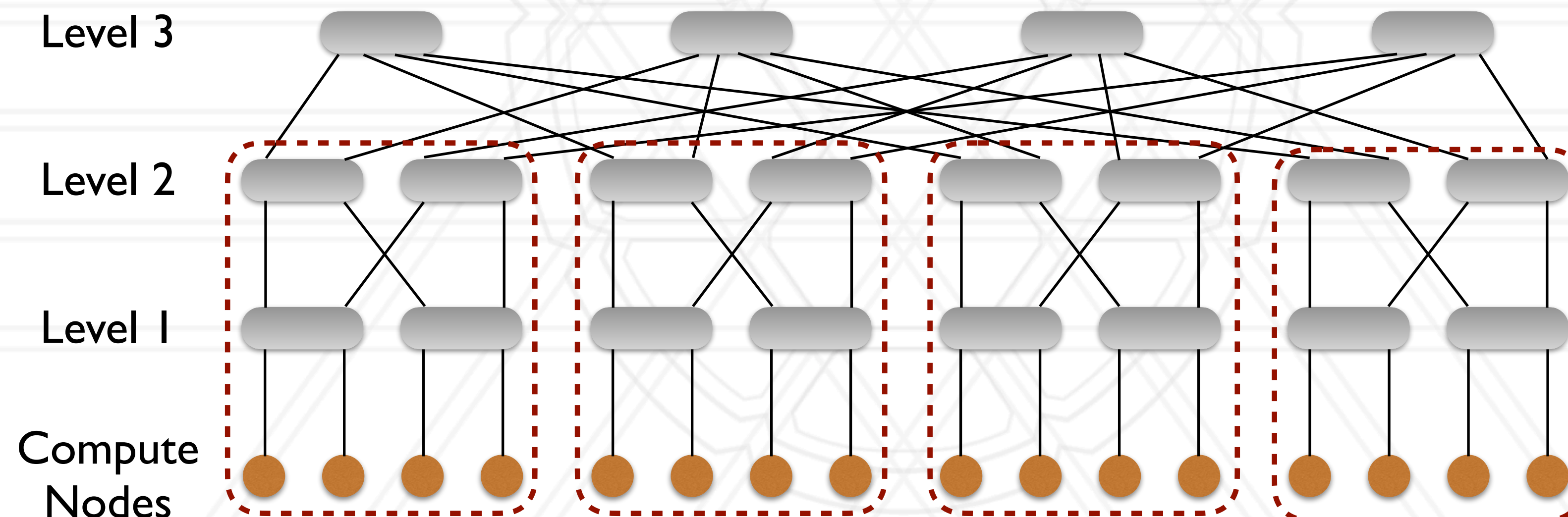
Fat-tree network

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



Fat-tree network

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

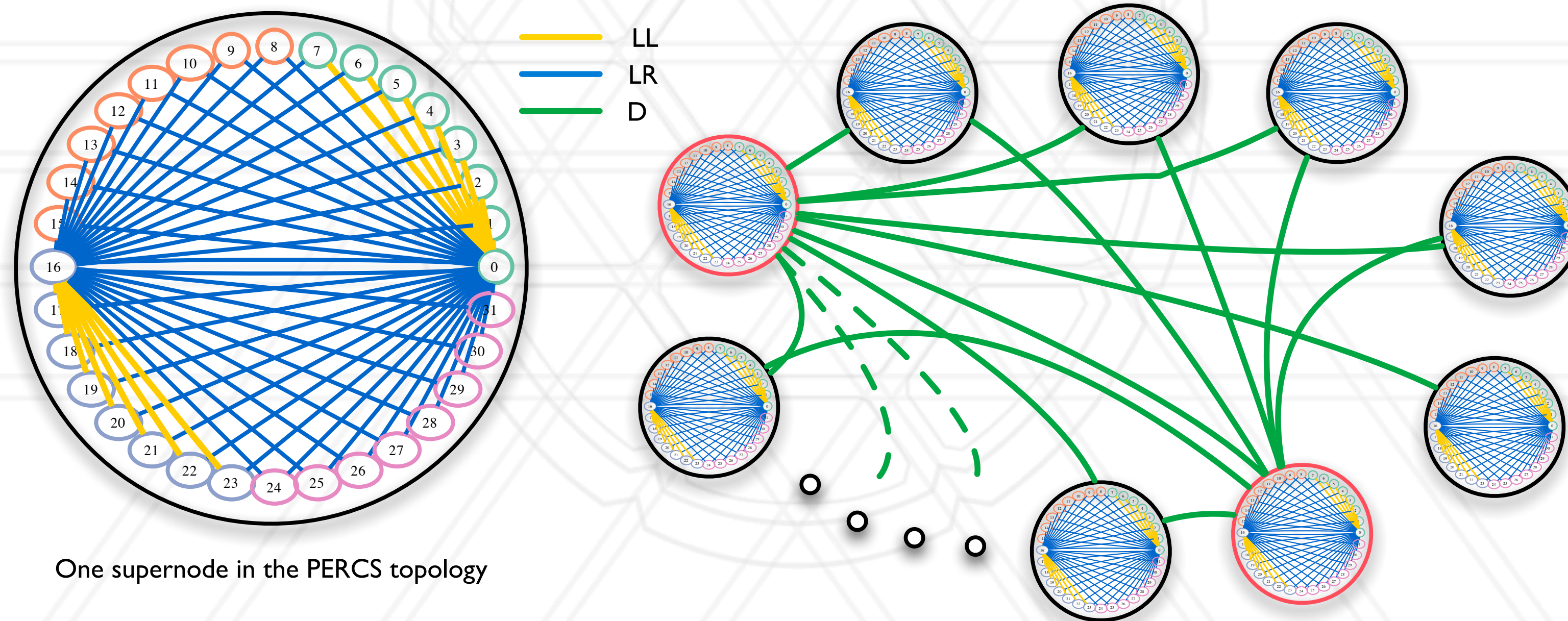


Network properties of fat-tree

- Let's say the number of switches is s , and router radix is k
- Diameter of 2-level fat-tree: 2
- Diameter of 3-level fat-tree: 4
- Diameter of a l -level fat-tree: $(l - 1) \times 2$
- Maximum number of nodes: $k \times \frac{k}{2} \times \frac{k}{2}$

Dragonfly network

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



Network properties of dragonfly

- Diameter of dragonfly with all-to-all connections within supernode: 3
- Diameter of dragonfly with row-column all-to-all connections within supernode: 5

Life-cycle of a message

Source

Source

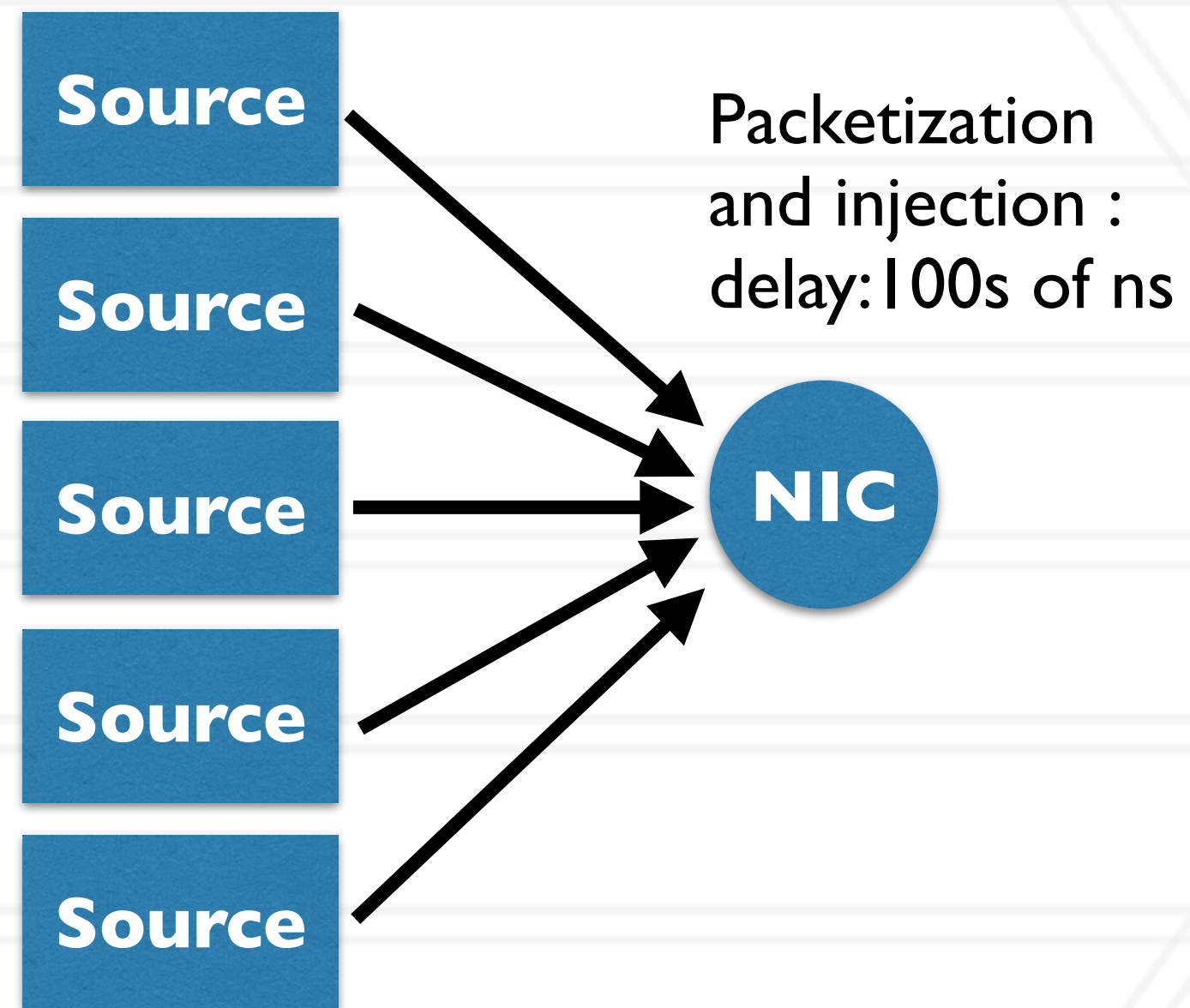
Source

Source

Source

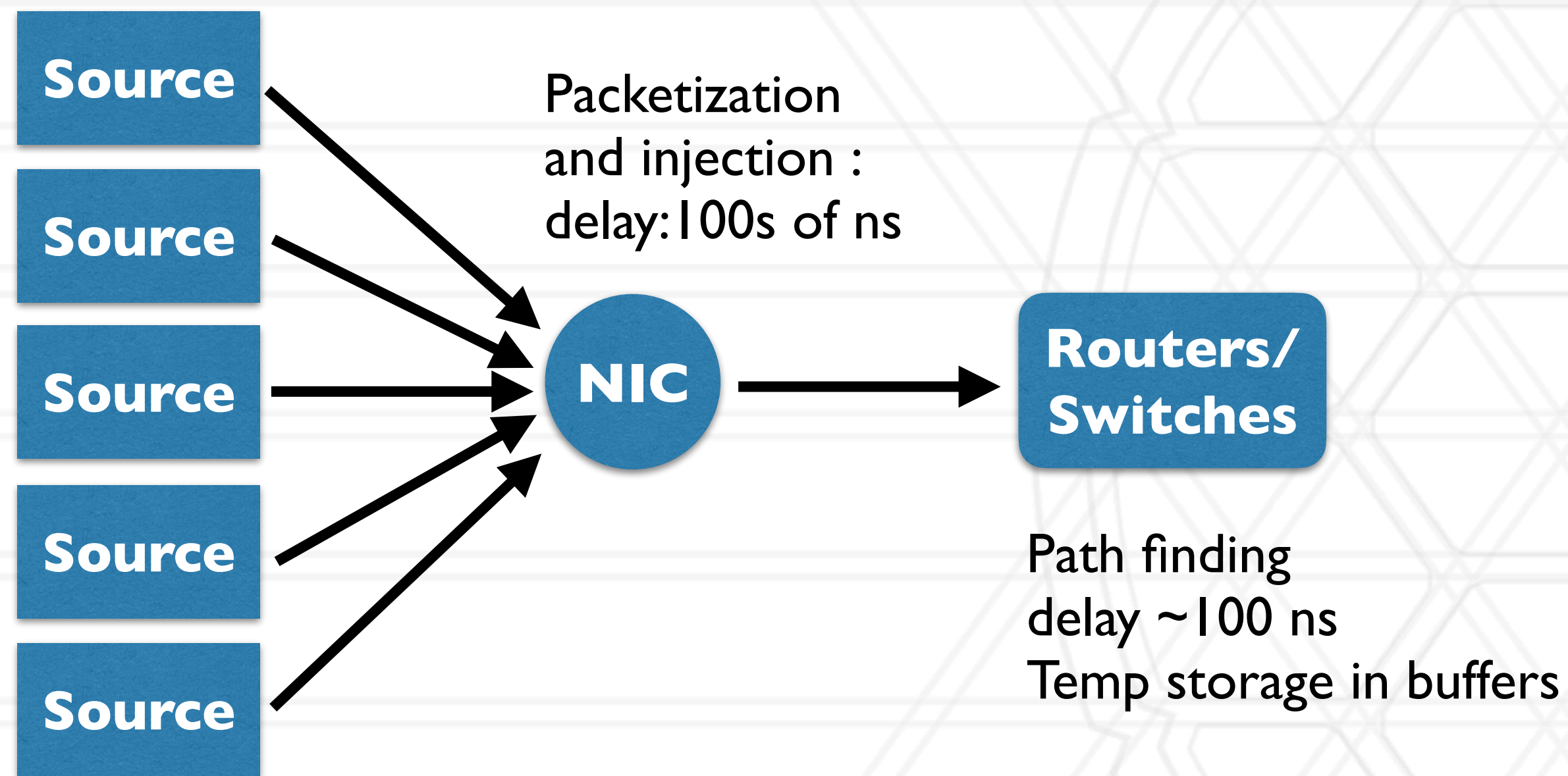
Message origin points :
destination, frequency,
size, etc. determined
by application
1 micro sec - 10s of sec

Life-cycle of a message



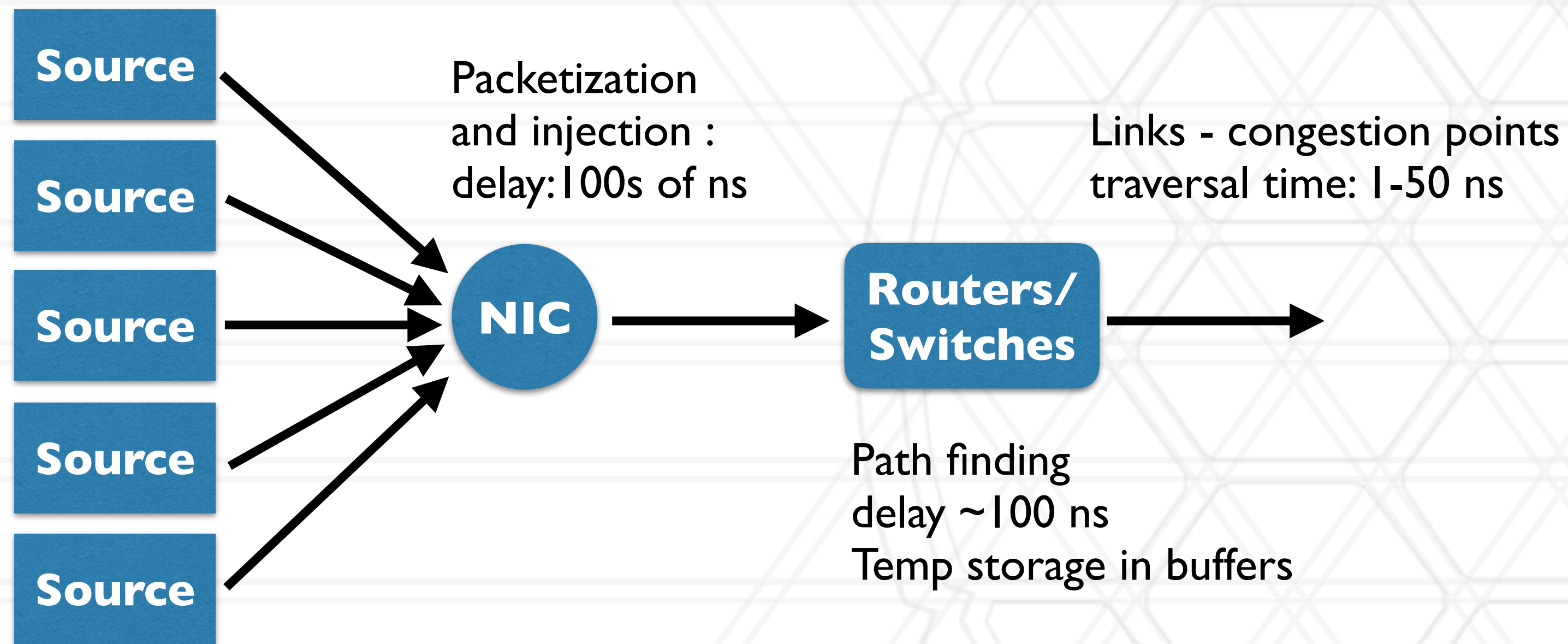
Message origin points :
destination, frequency,
size, etc. determined
by application
1 micro sec - 10s of sec

Life-cycle of a message



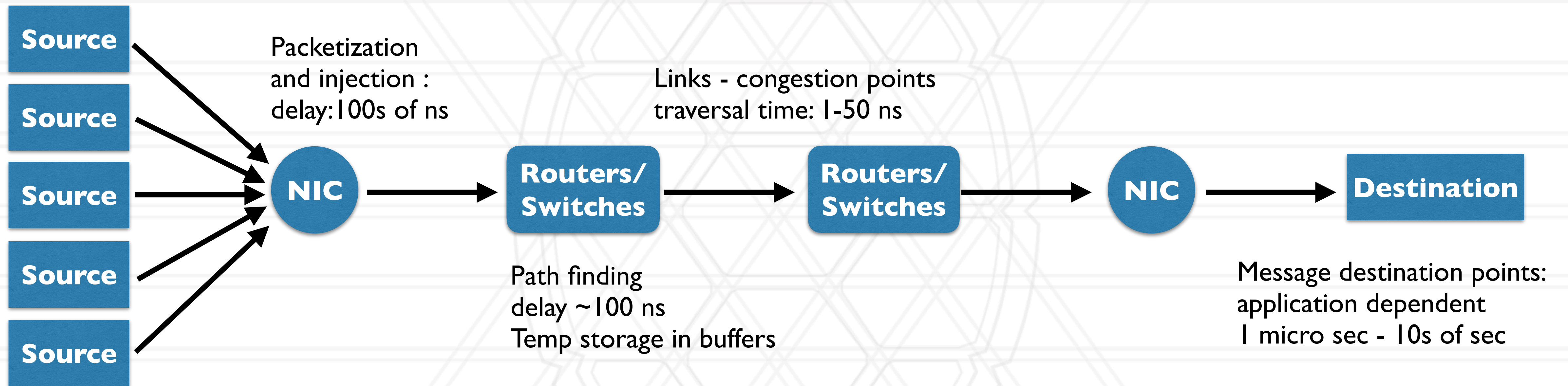
Message origin points :
destination, frequency,
size, etc. determined
by application
1 micro sec - 10s of sec

Life-cycle of a message



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

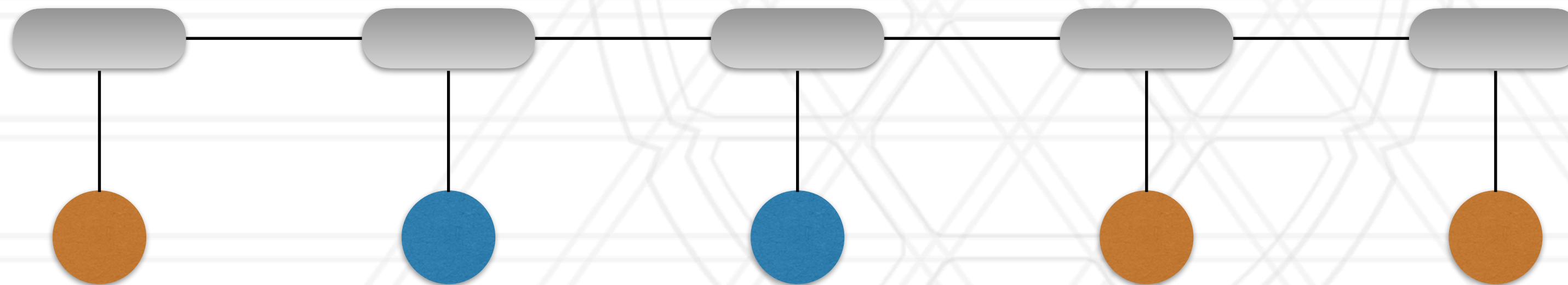
Life-cycle of a message



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

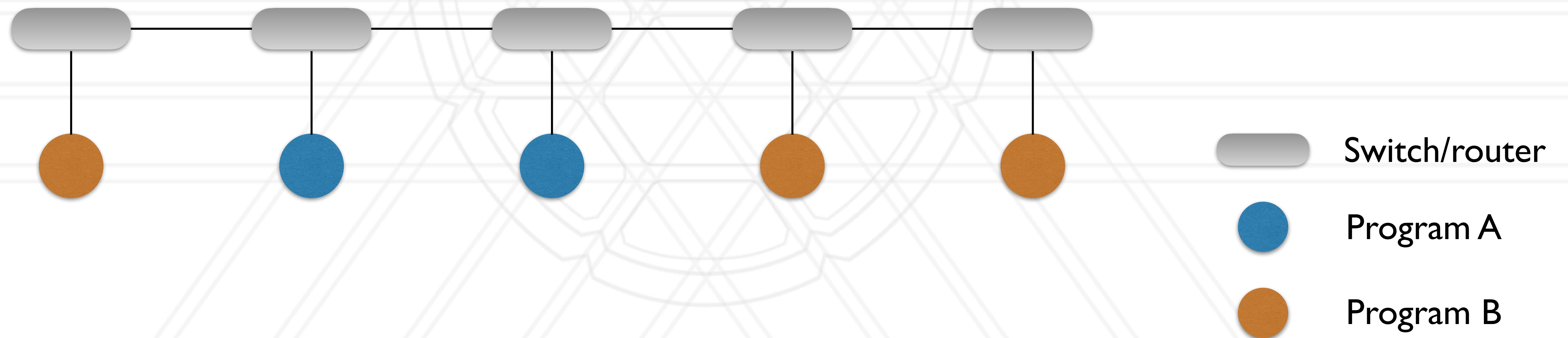
Congestion due to network sharing

- 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



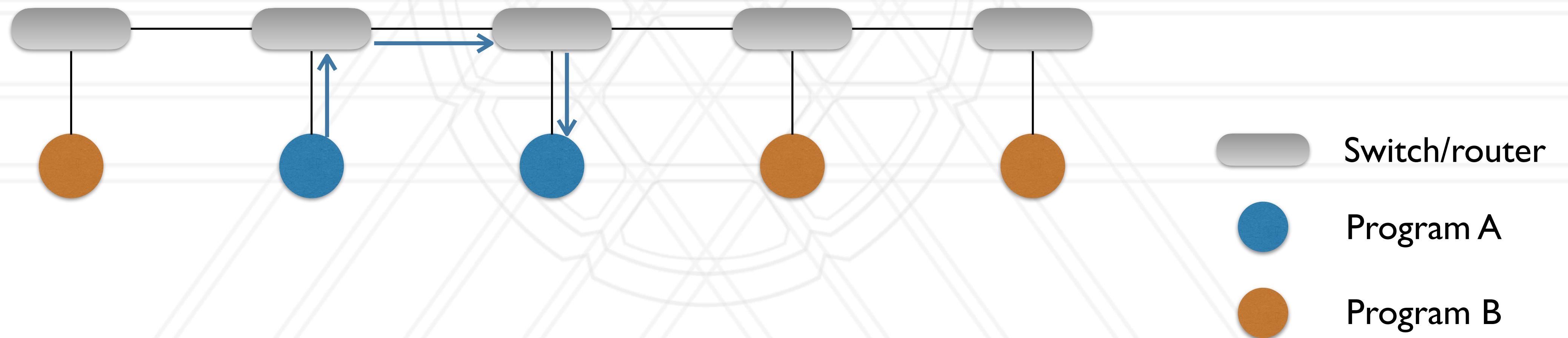
Congestion due to network sharing

- 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



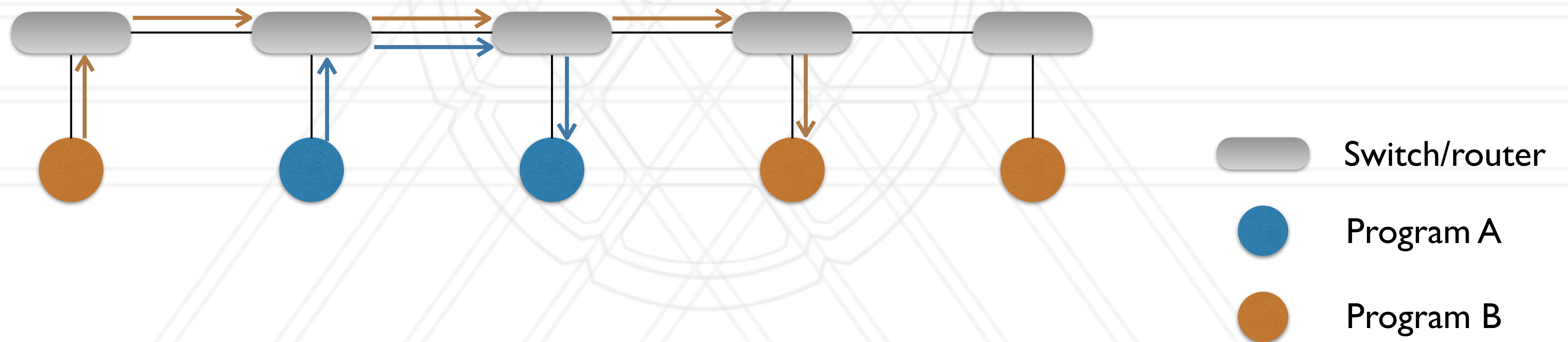
Congestion due to network sharing

- 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



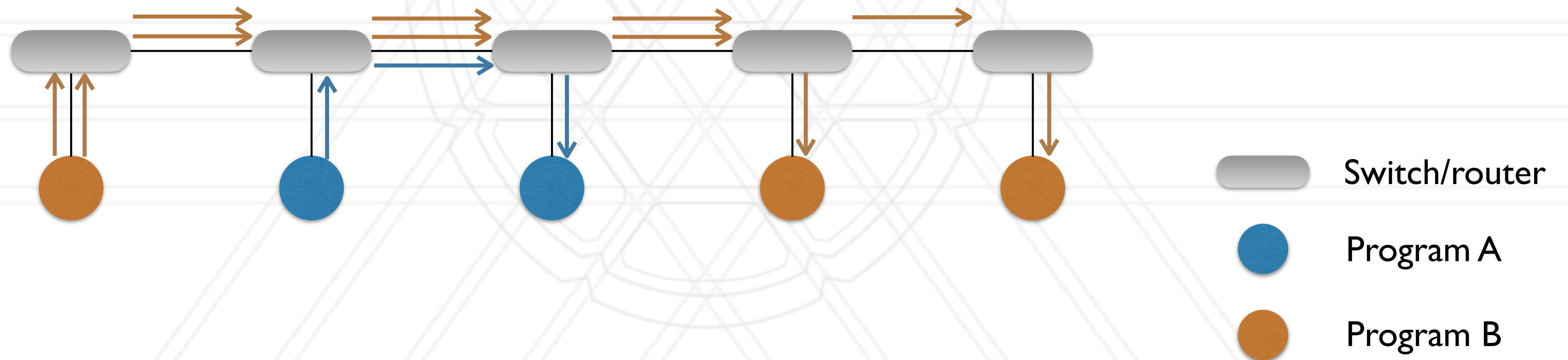
Congestion due to network sharing

- 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



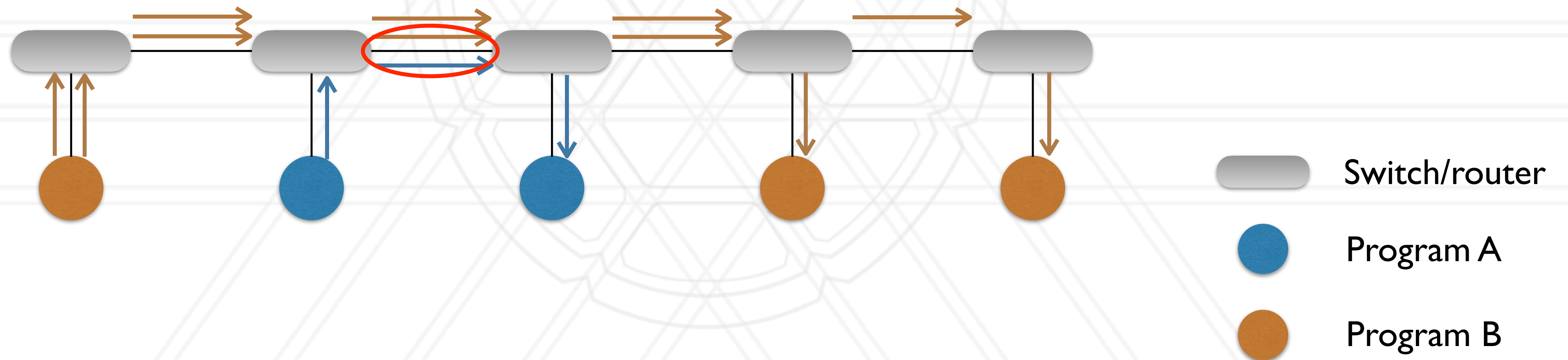
Congestion due to network sharing

- 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



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

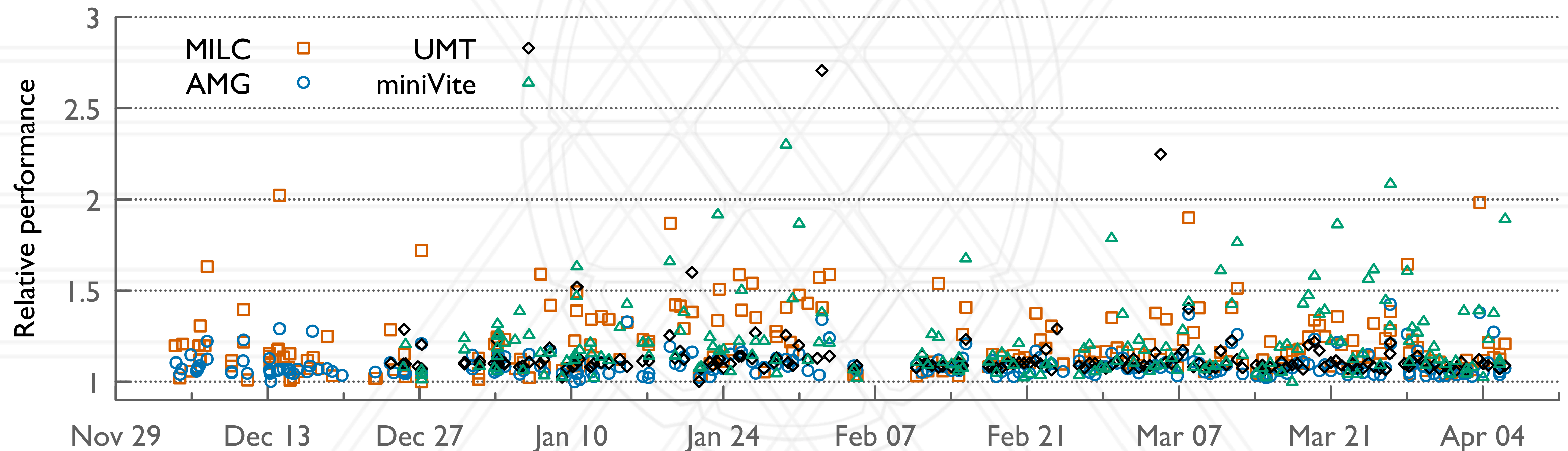


Routing algorithm

- 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

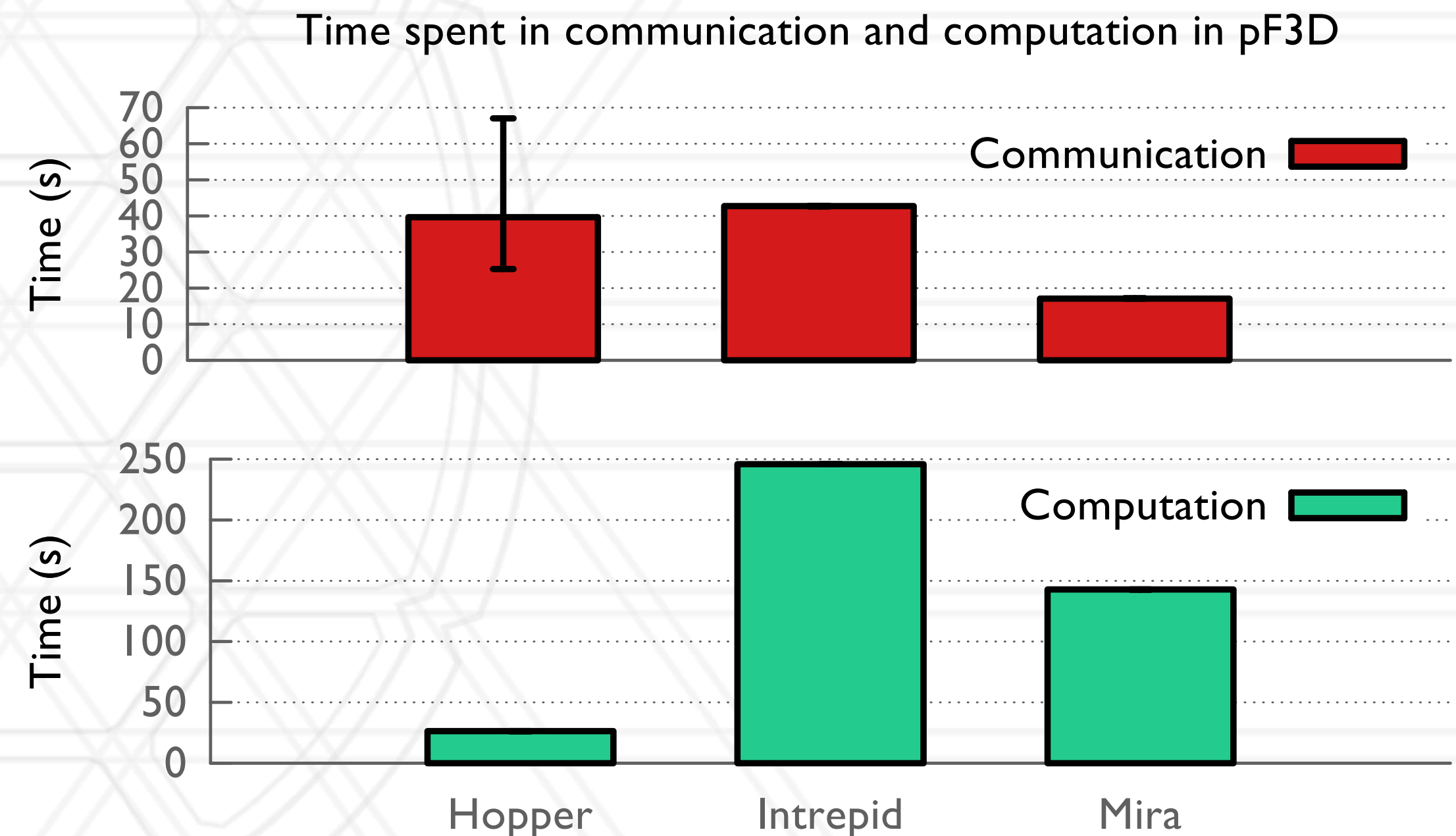
Performance of control jobs running the same executable and input varies as they are run from day-to-day on 128 nodes of Cori in 2018-2019



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

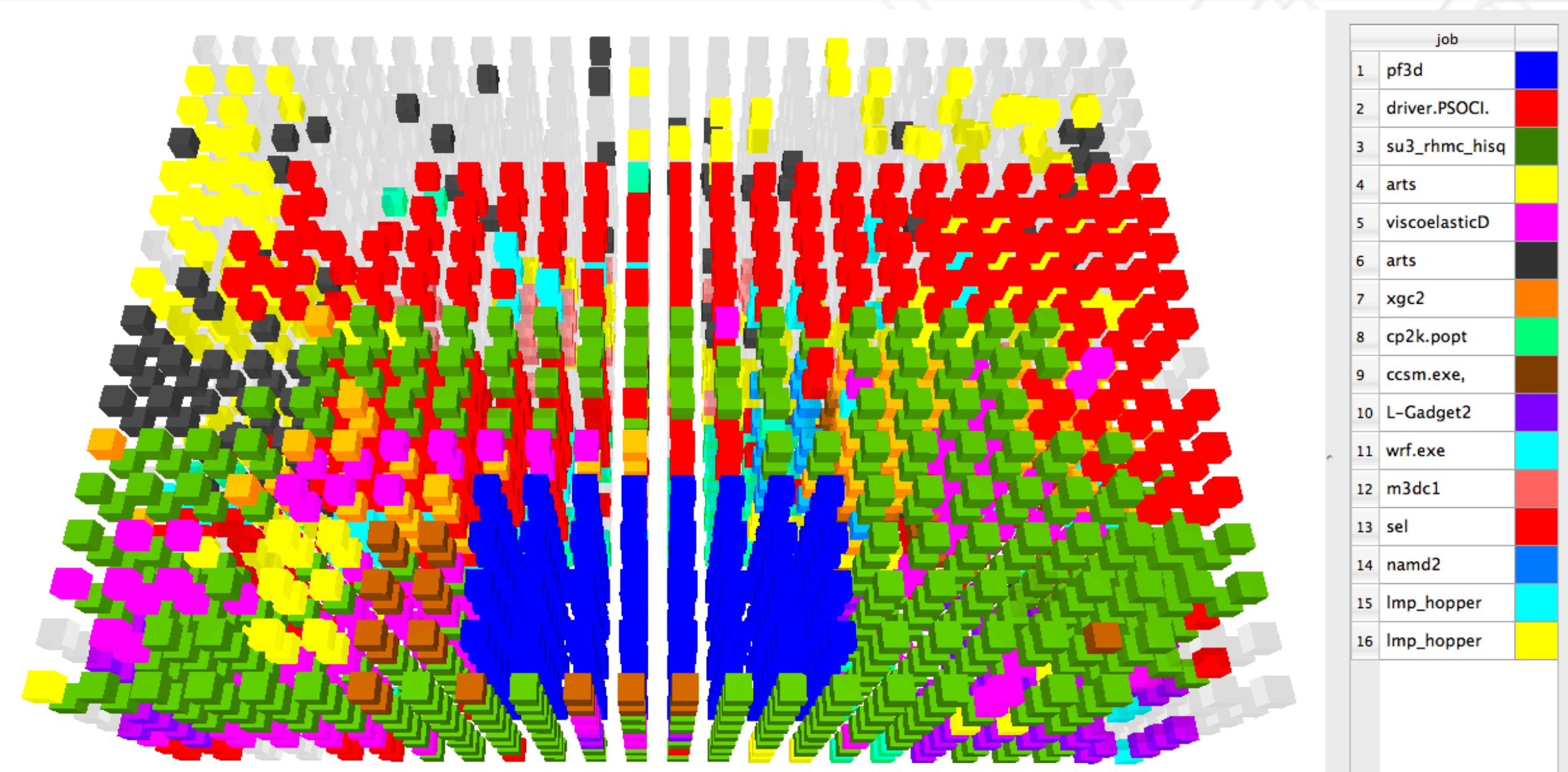
Performance variability due to congestion

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

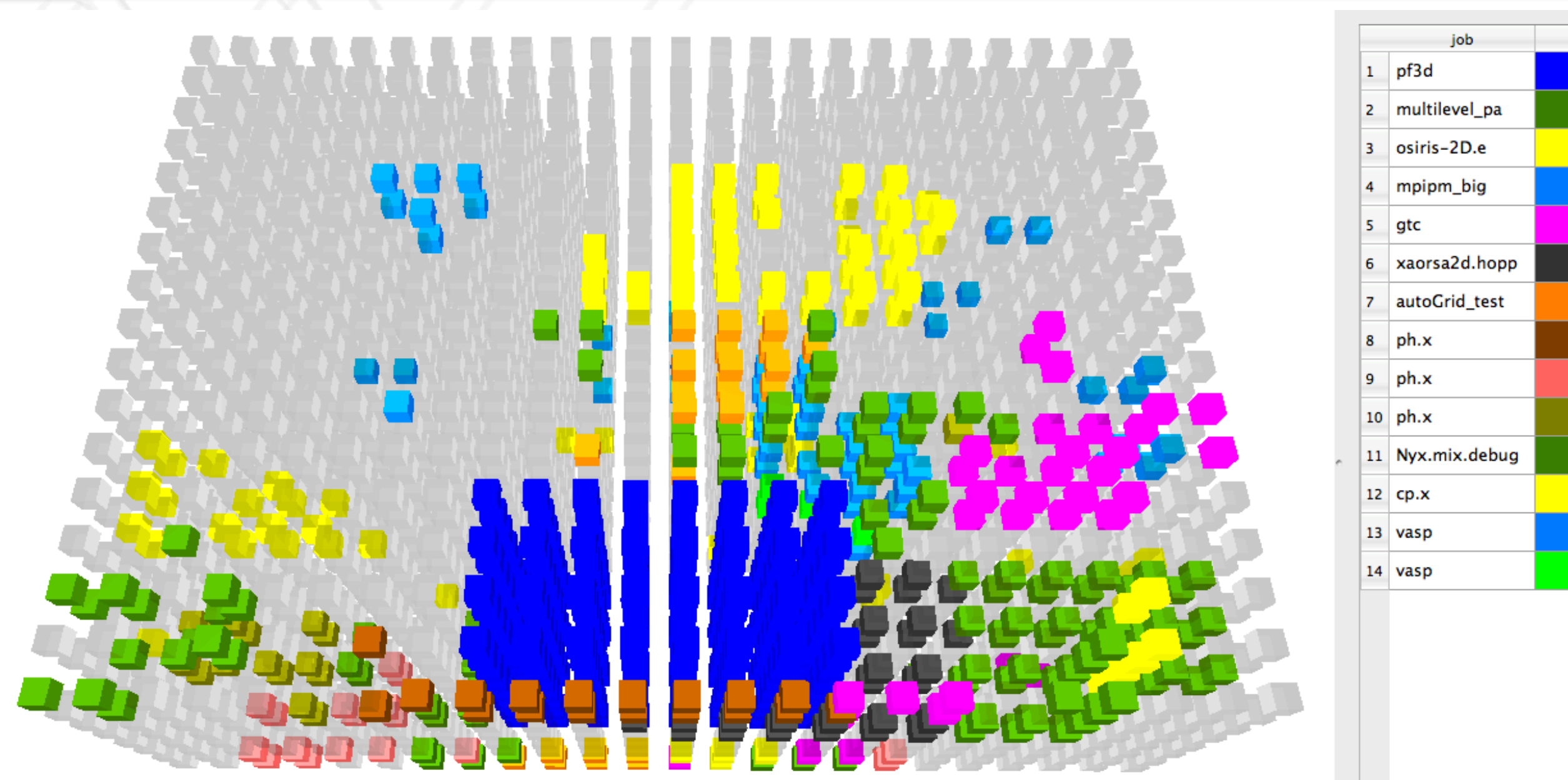


Bhatele et al. <http://www.cs.umd.edu/~bhatele/pubs/pdf/2013/sc2013a.pdf>

Impact of other jobs

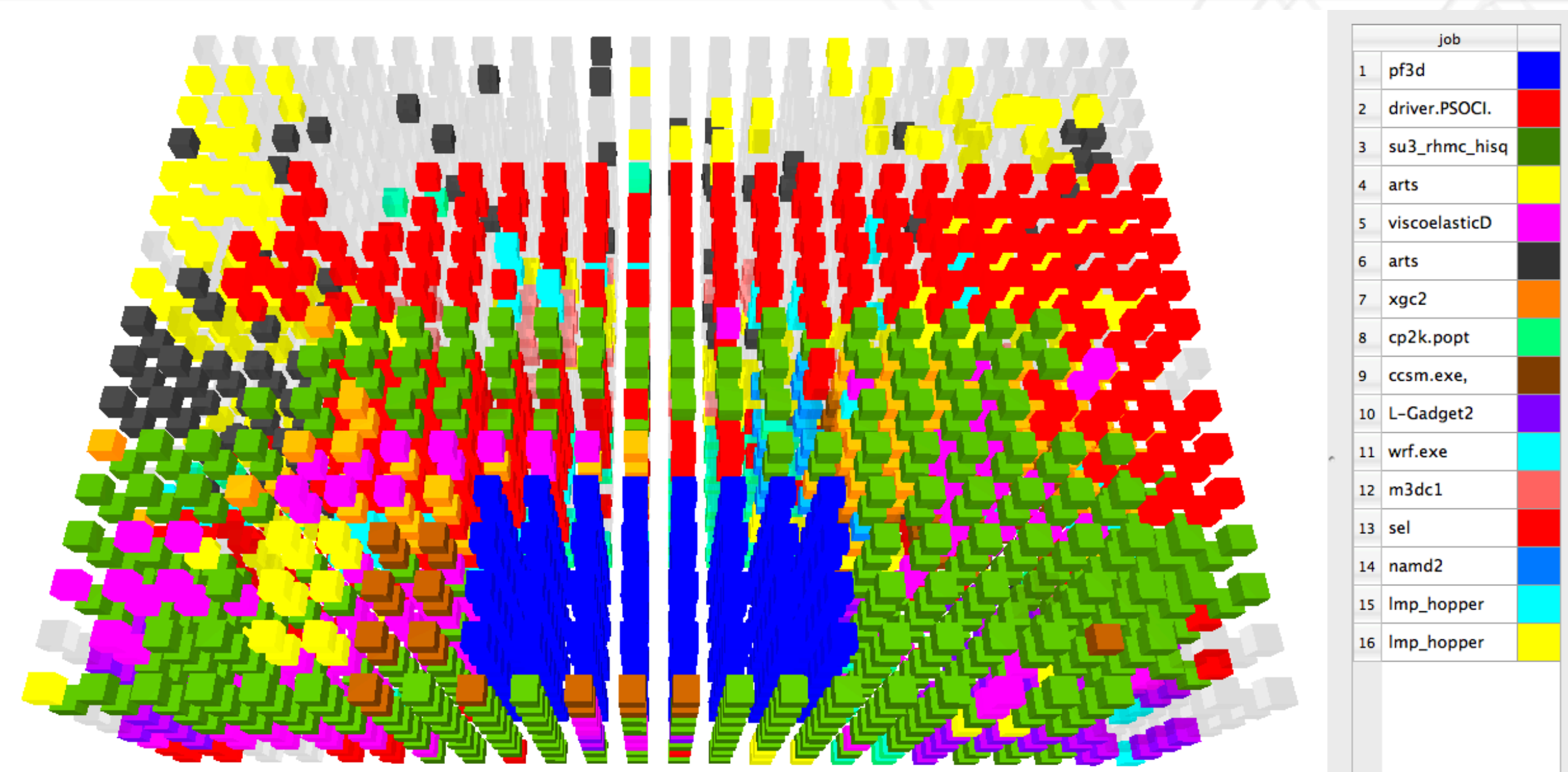


April 11

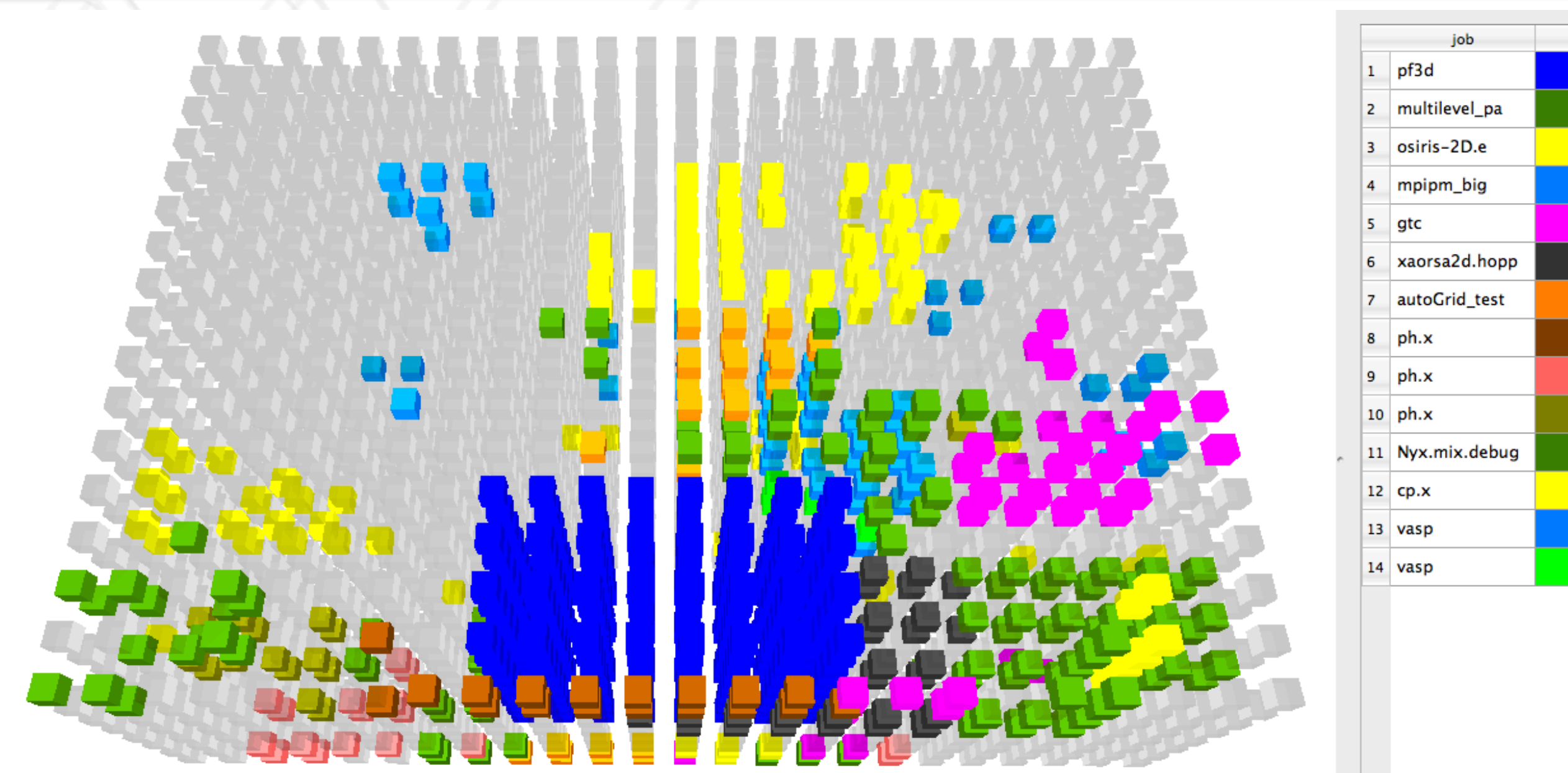


April 16

Impact of other jobs



April 11
MILC job in green



April 16
25% higher messaging rate

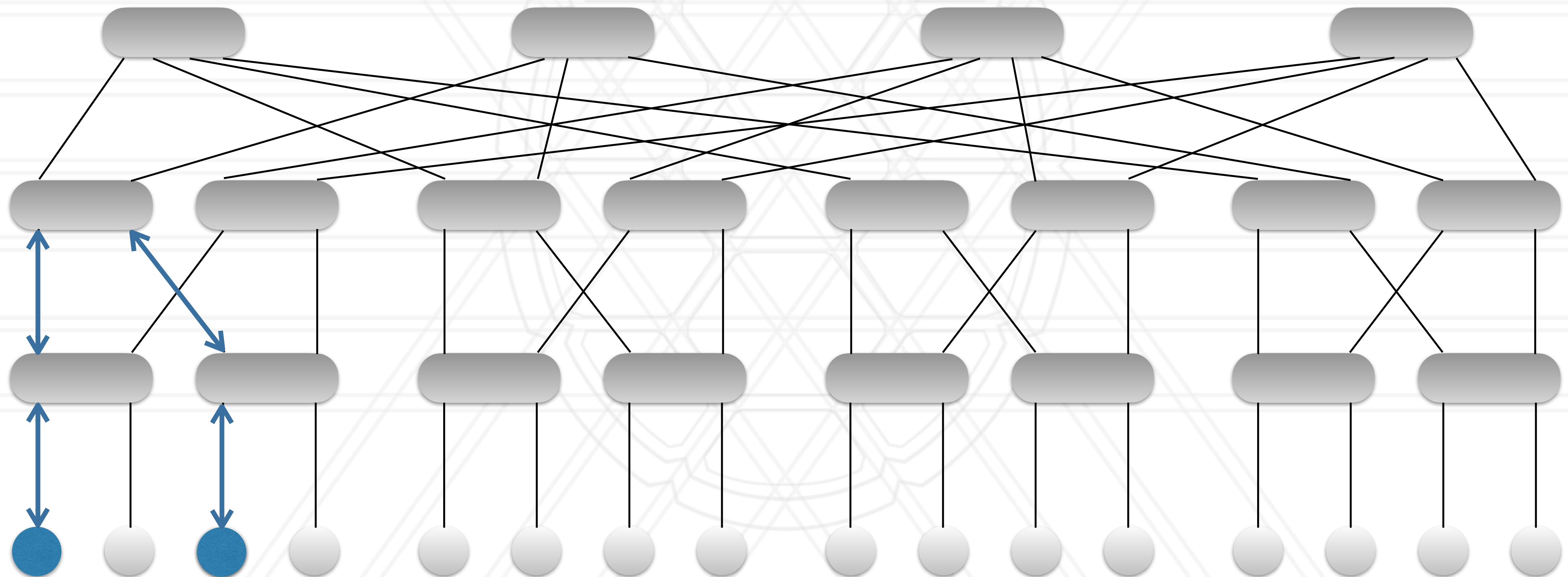
Announcements

- Quiz 3 will be posted this week

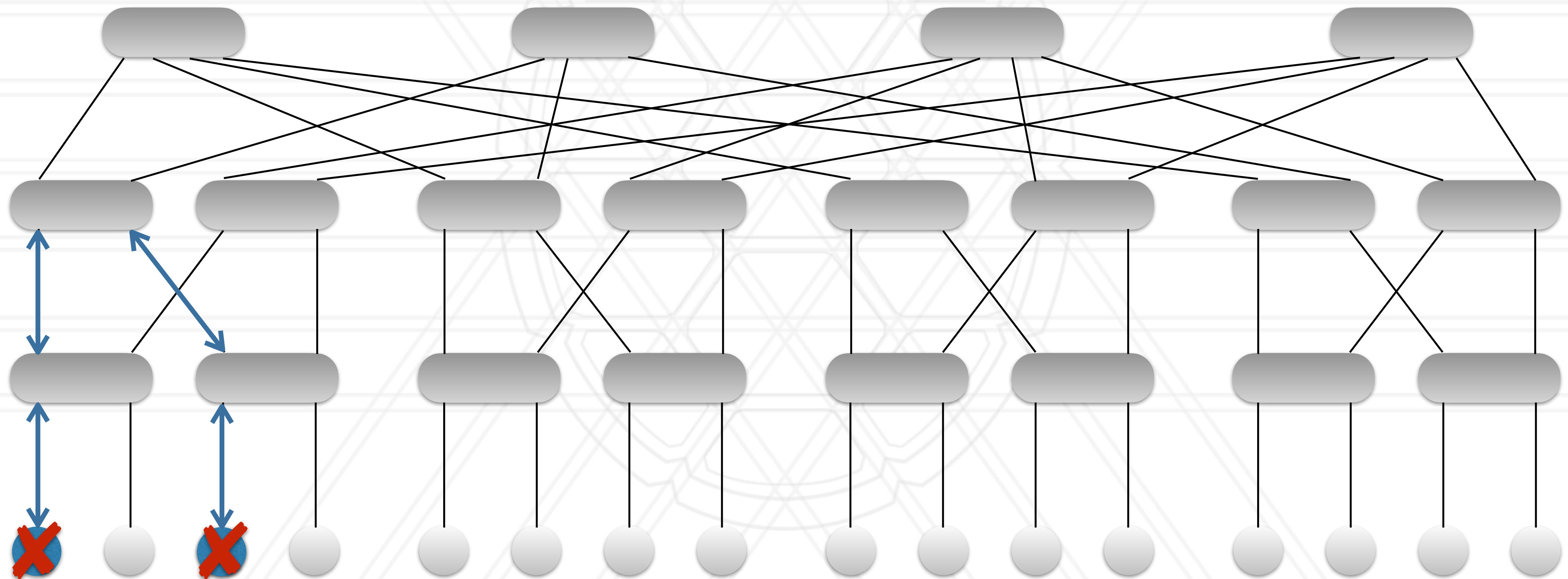
Different approaches to mitigating congestion

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

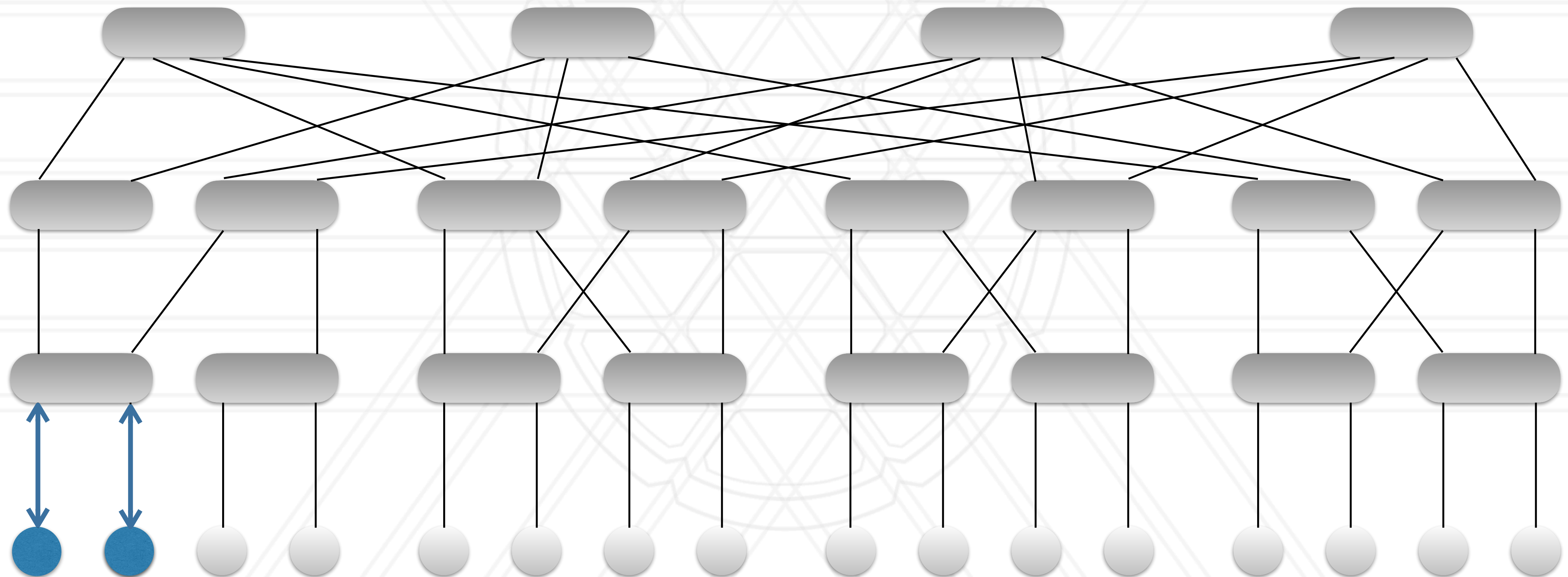
Topology-aware node allocation



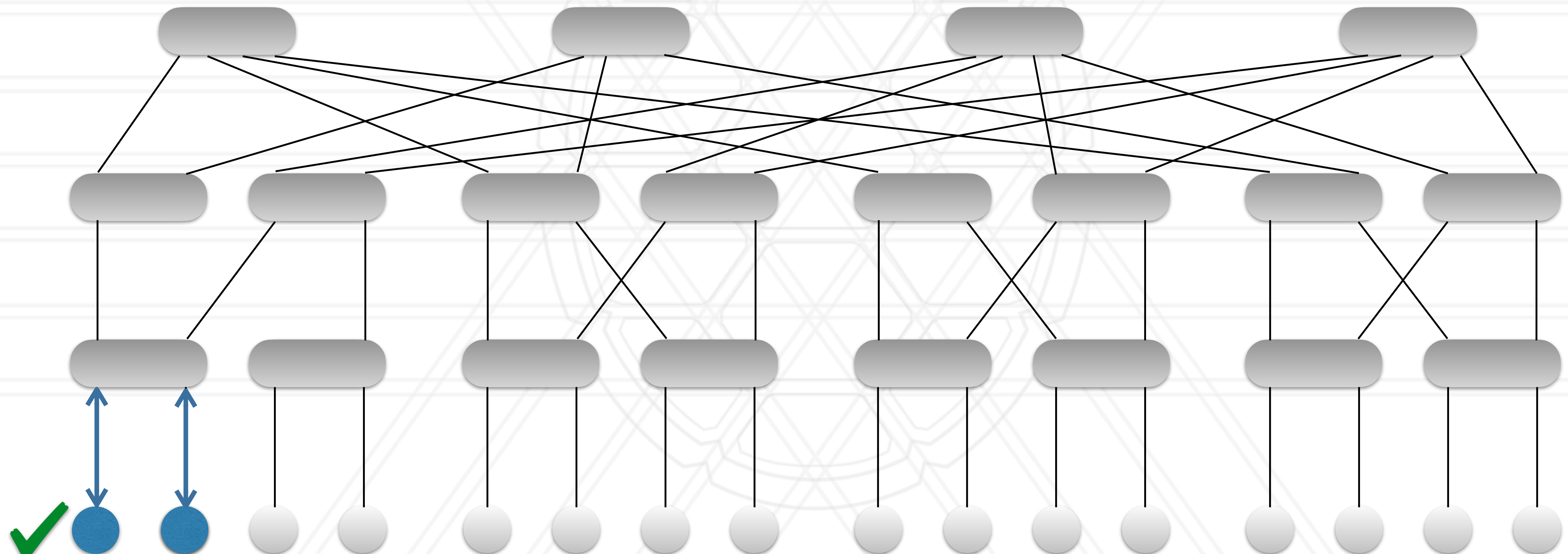
Topology-aware node allocation



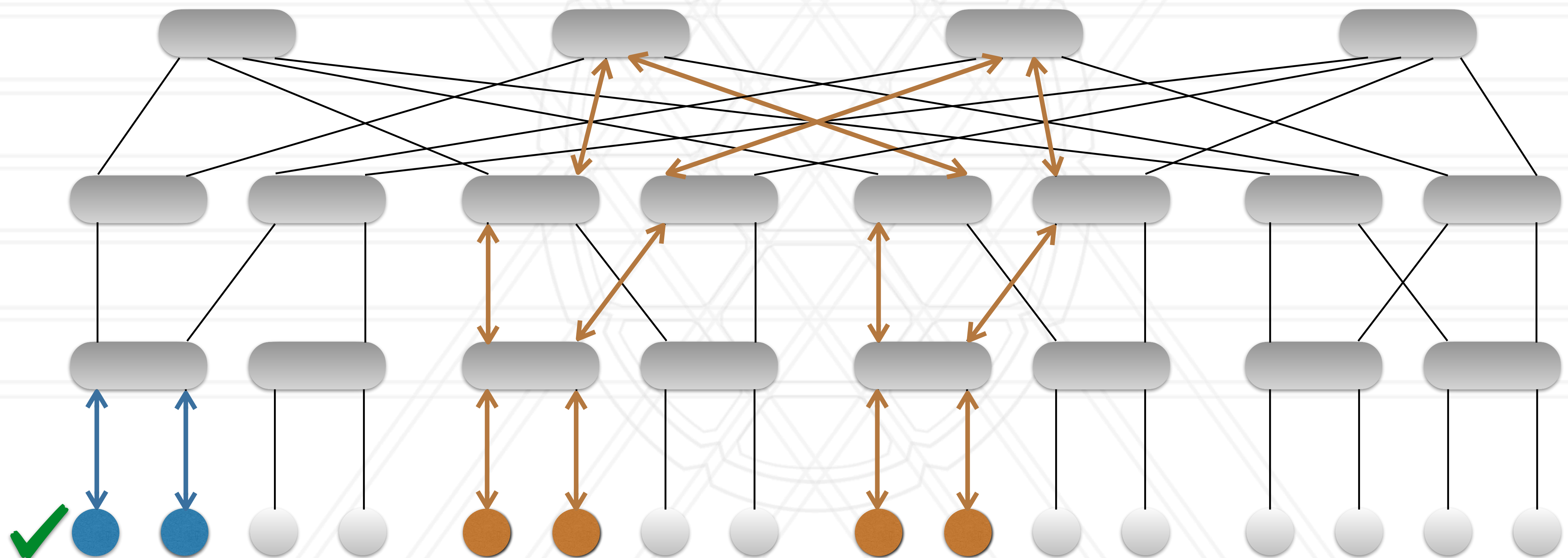
Topology-aware node allocation



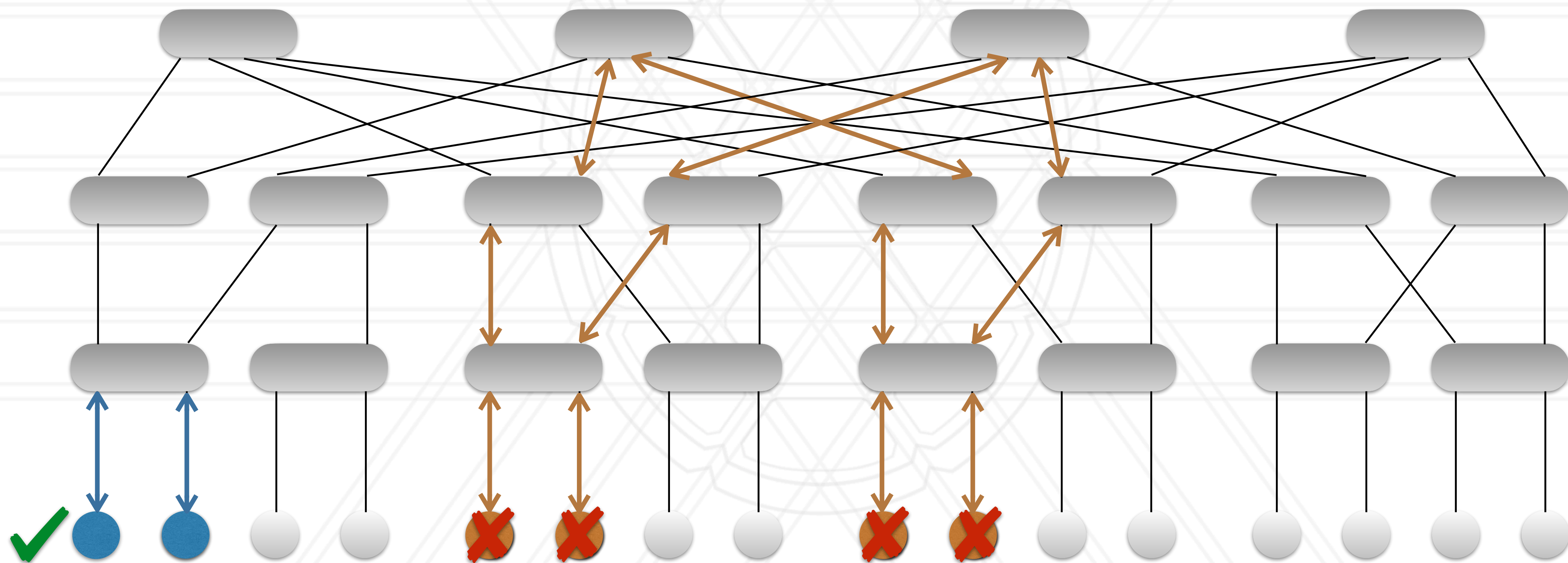
Topology-aware node allocation



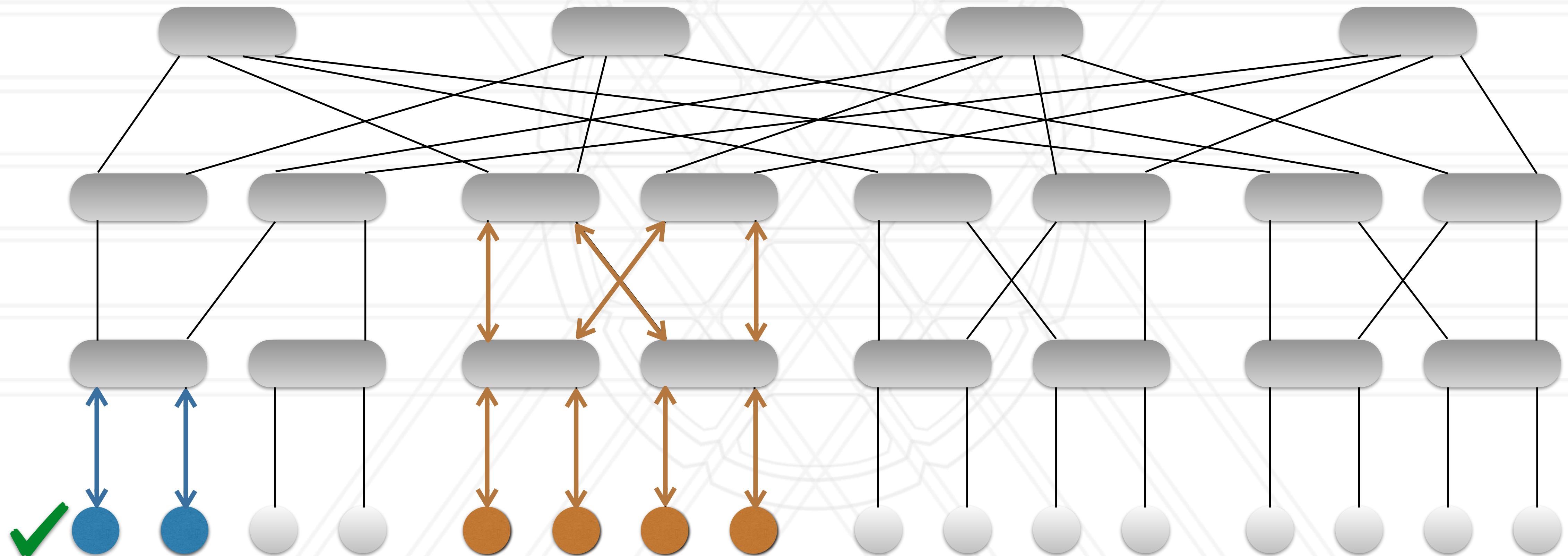
Topology-aware node allocation



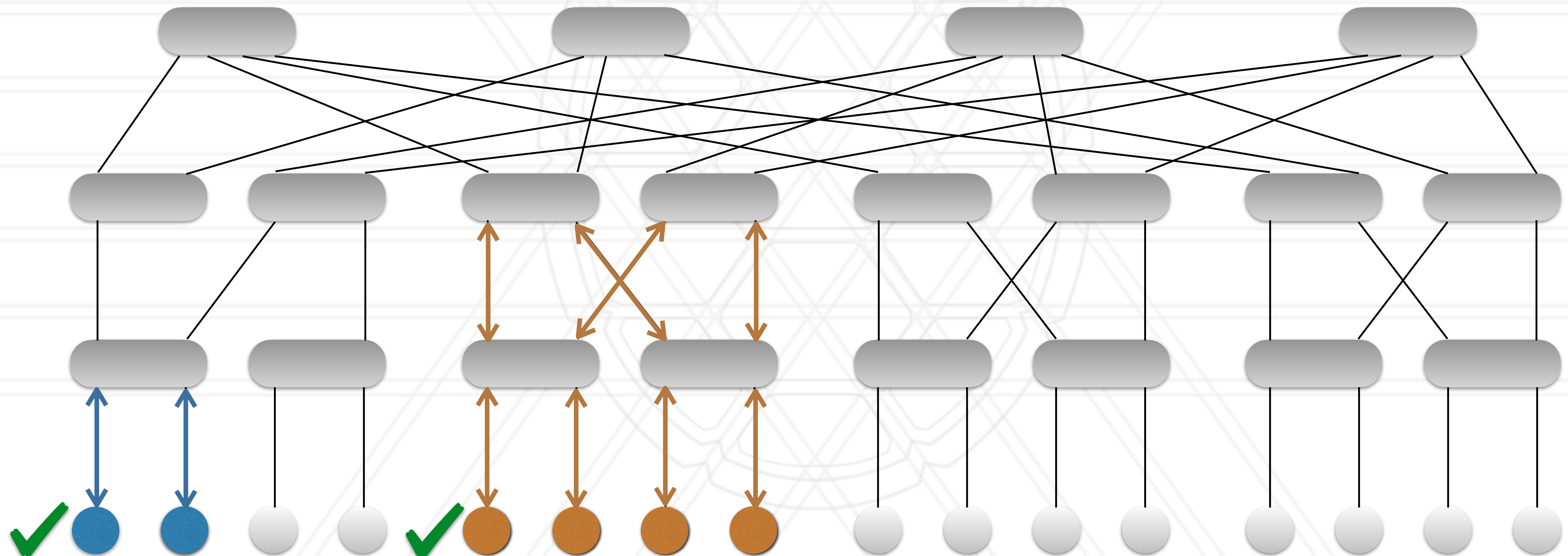
Topology-aware node allocation



Topology-aware node allocation

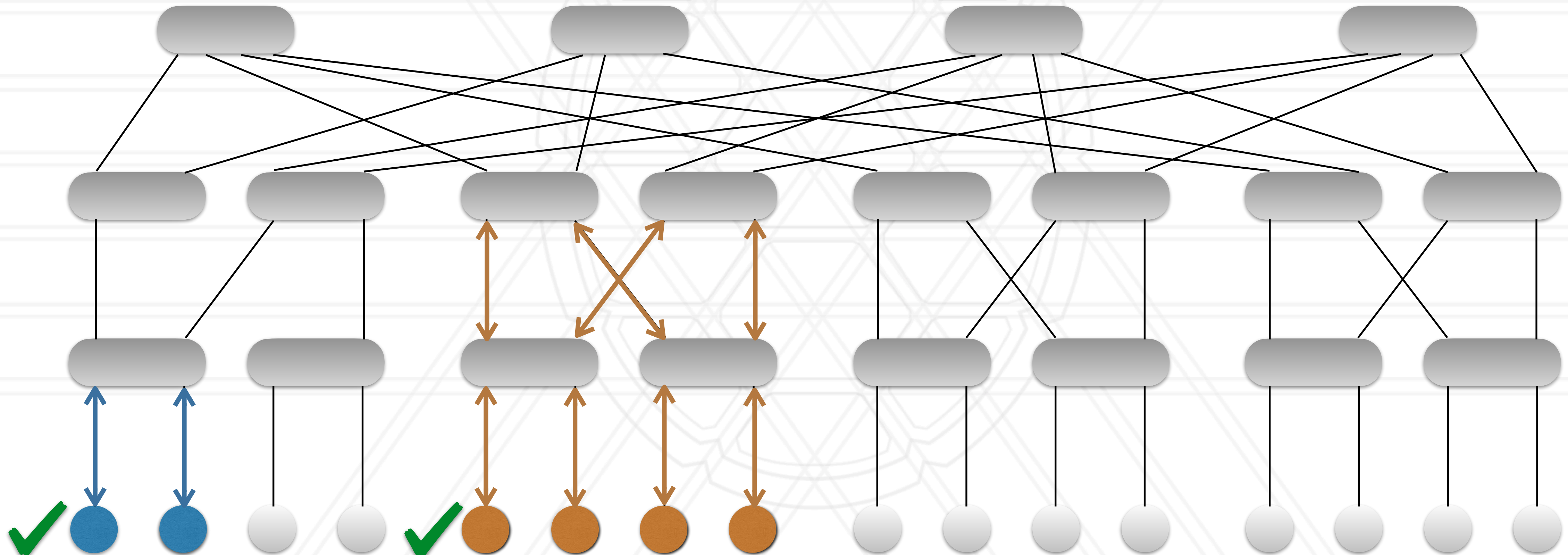


Topology-aware node allocation

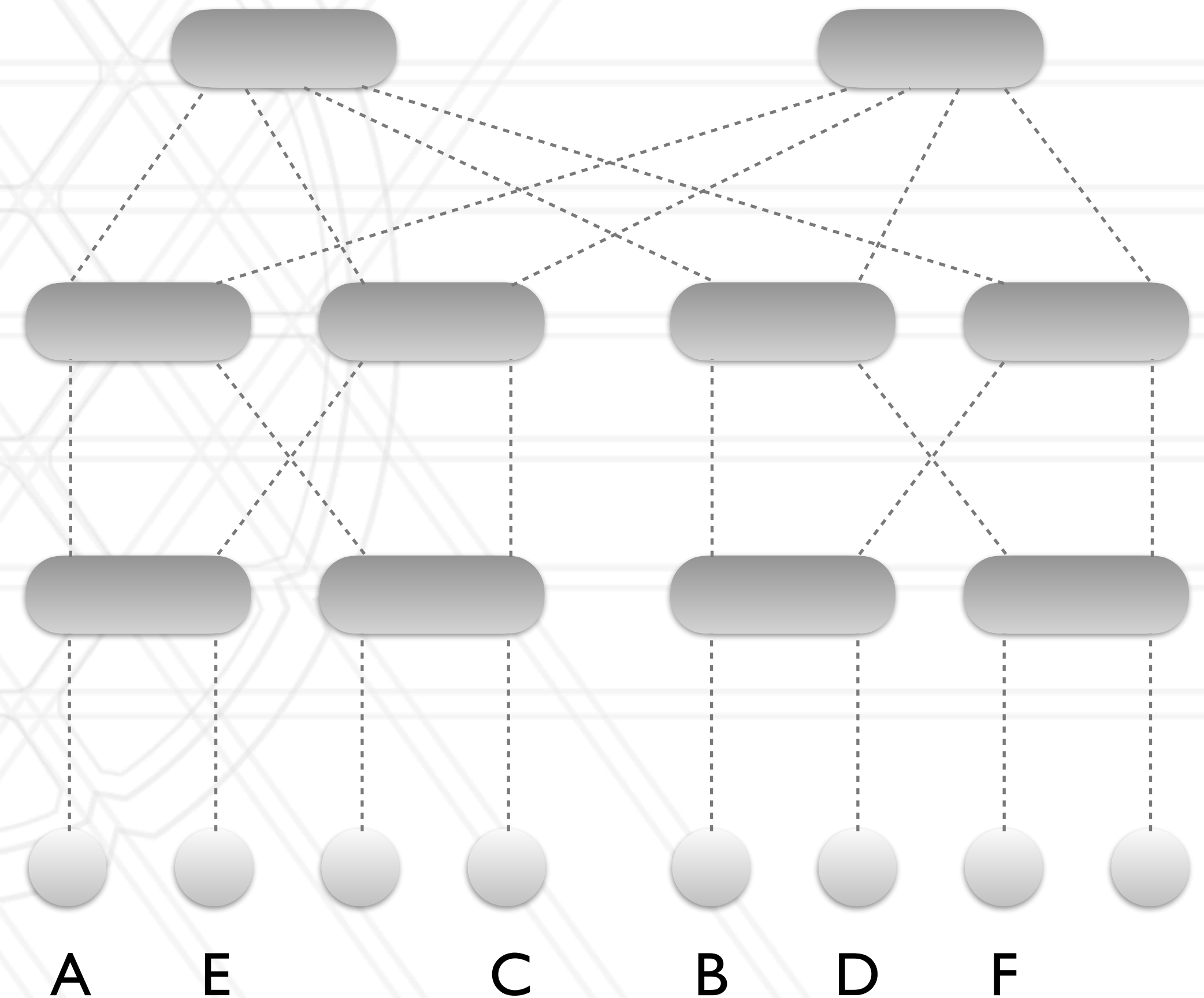


Topology-aware node allocation

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



AFAR: adaptive flow aware routing

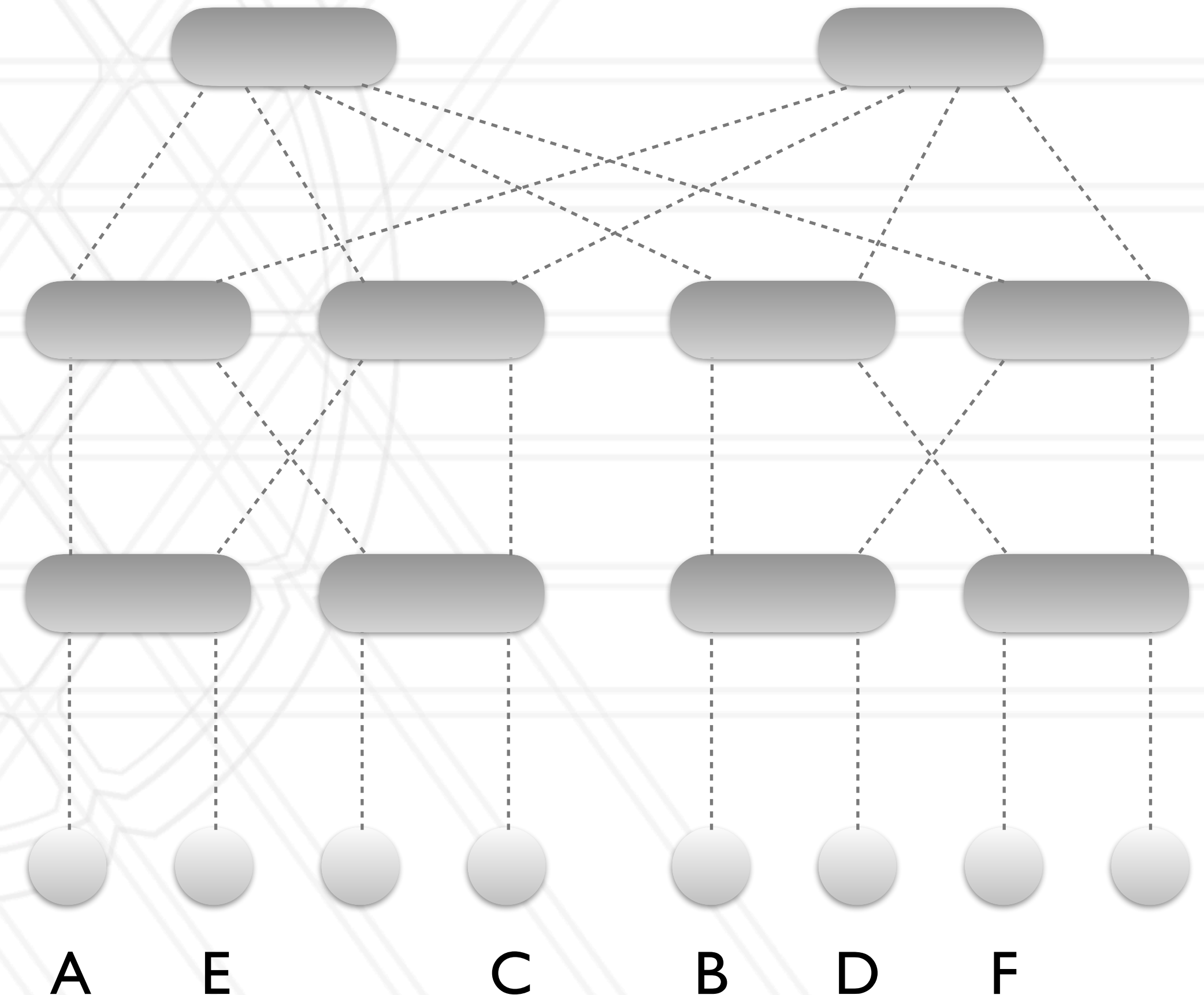


AFAR: adaptive flow aware routing

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

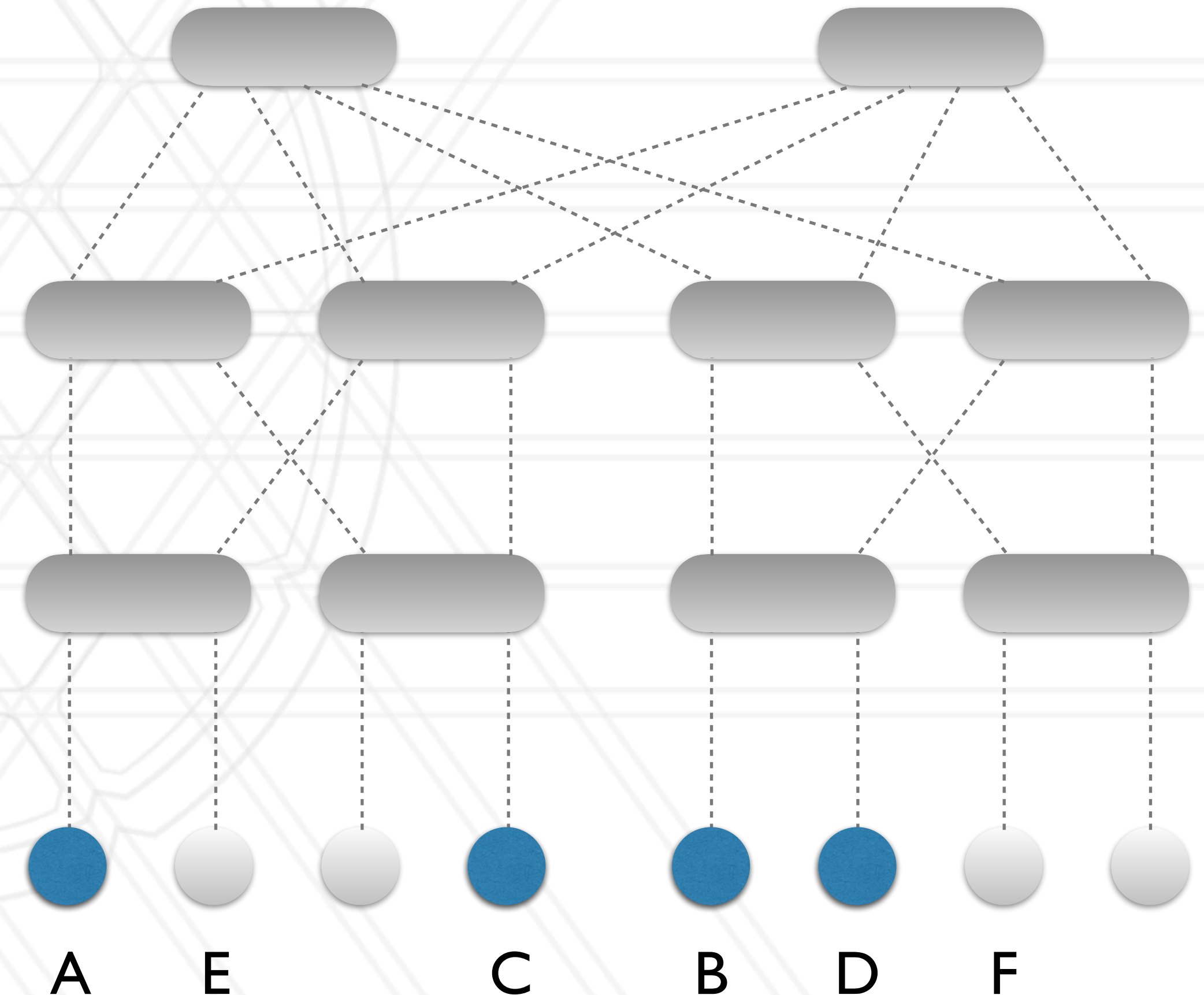


AFAR: adaptive flow aware routing

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

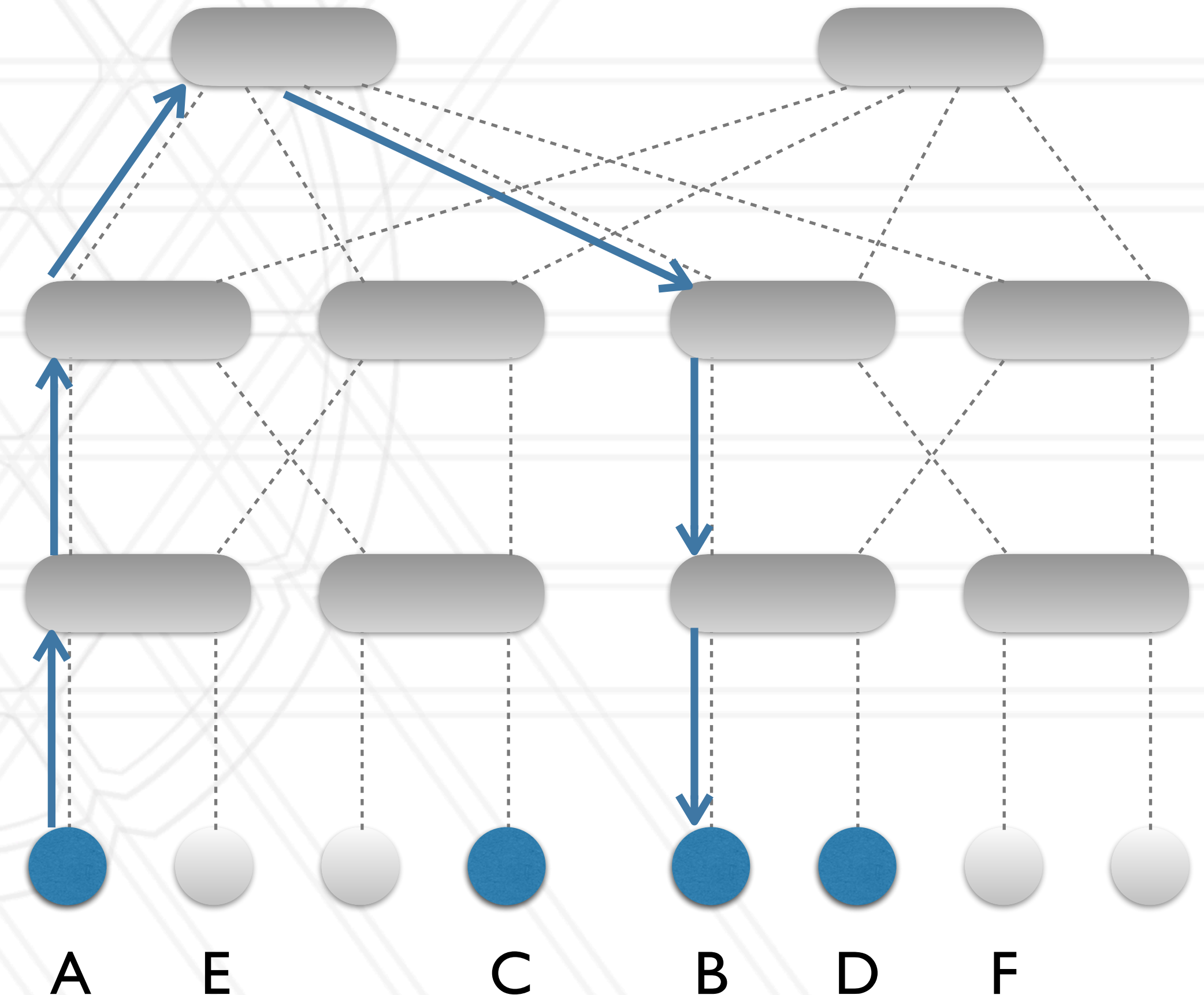


AFAR: adaptive flow aware routing

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

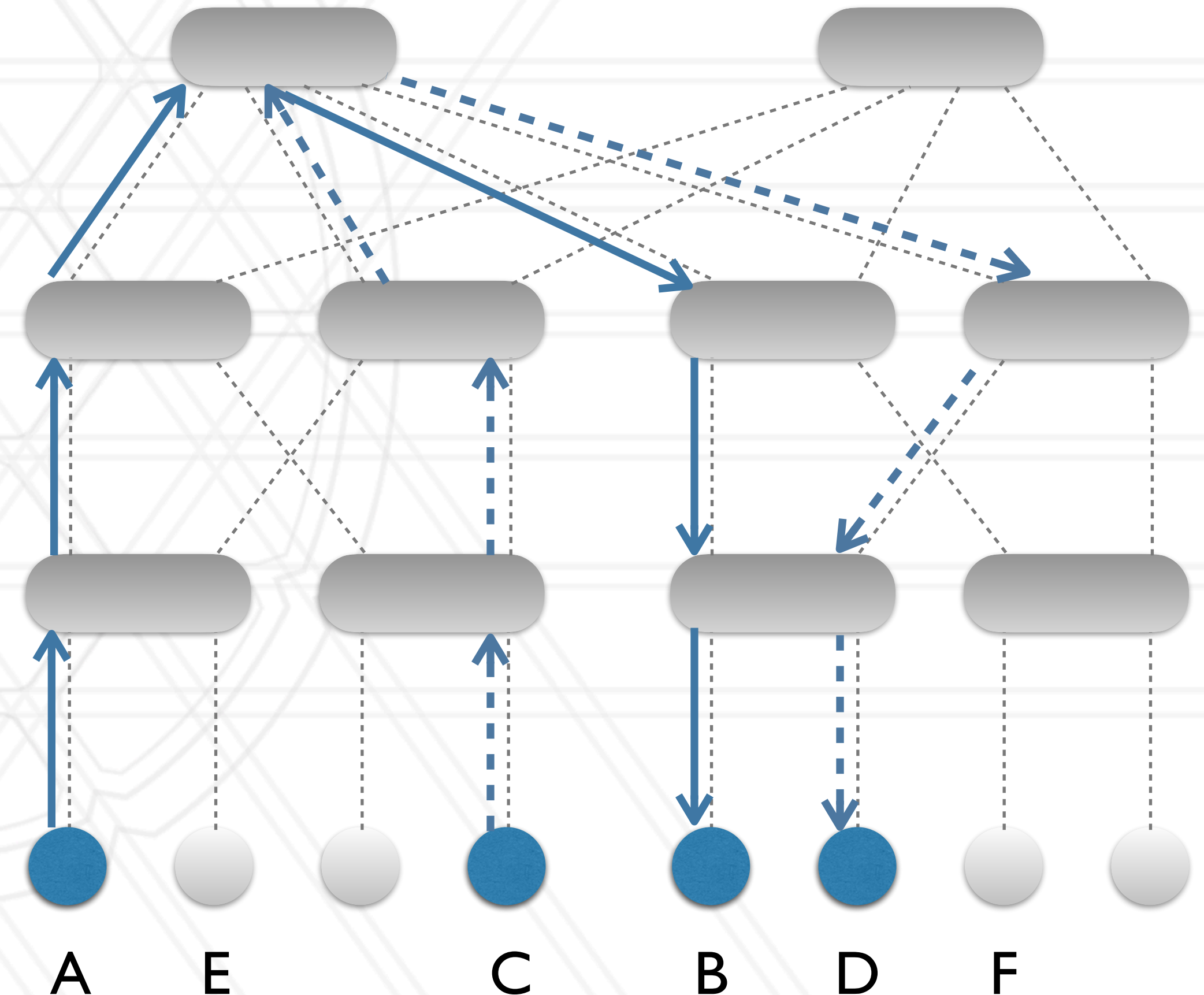


AFAR: adaptive flow aware routing

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

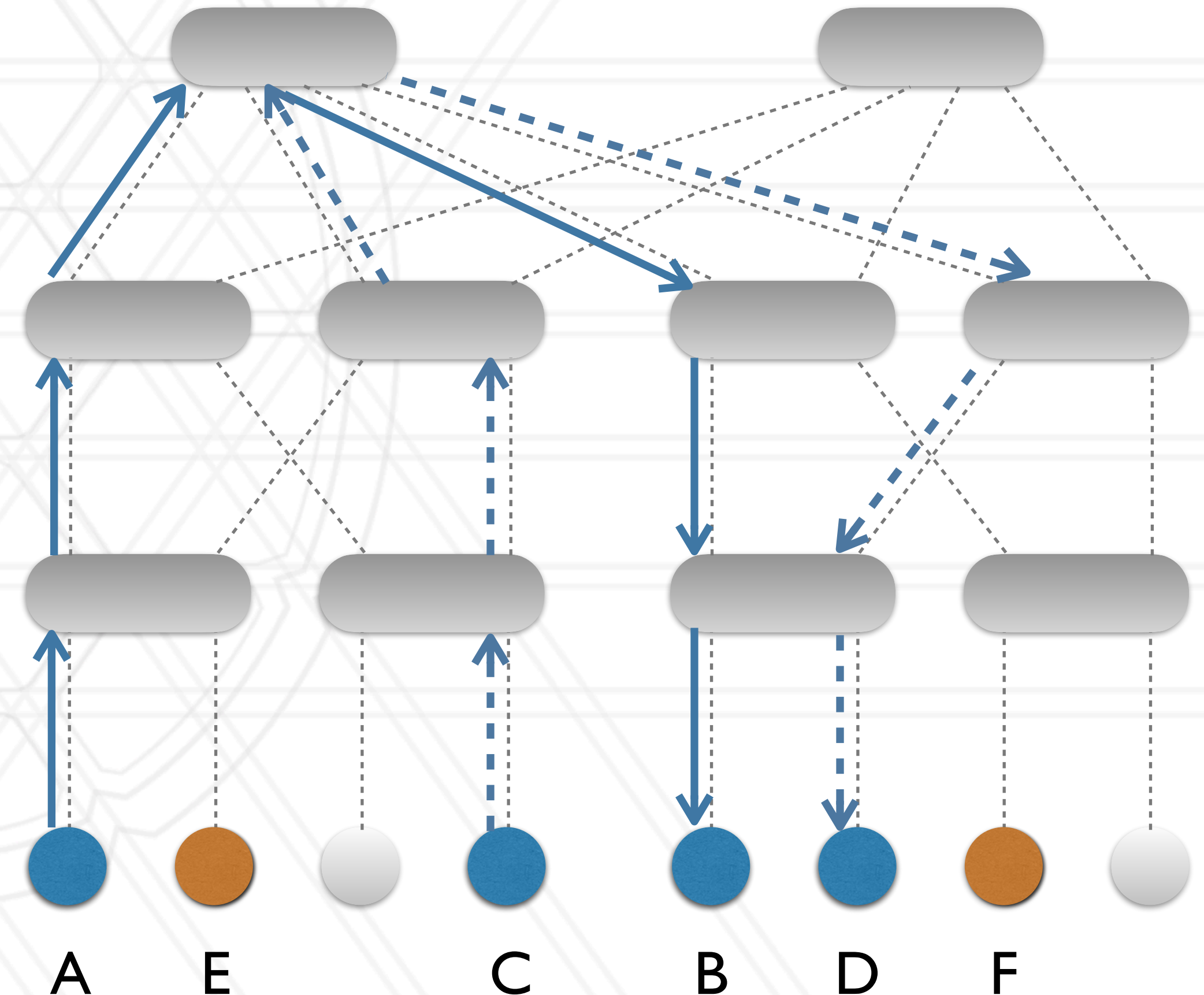


AFAR: adaptive flow aware routing

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

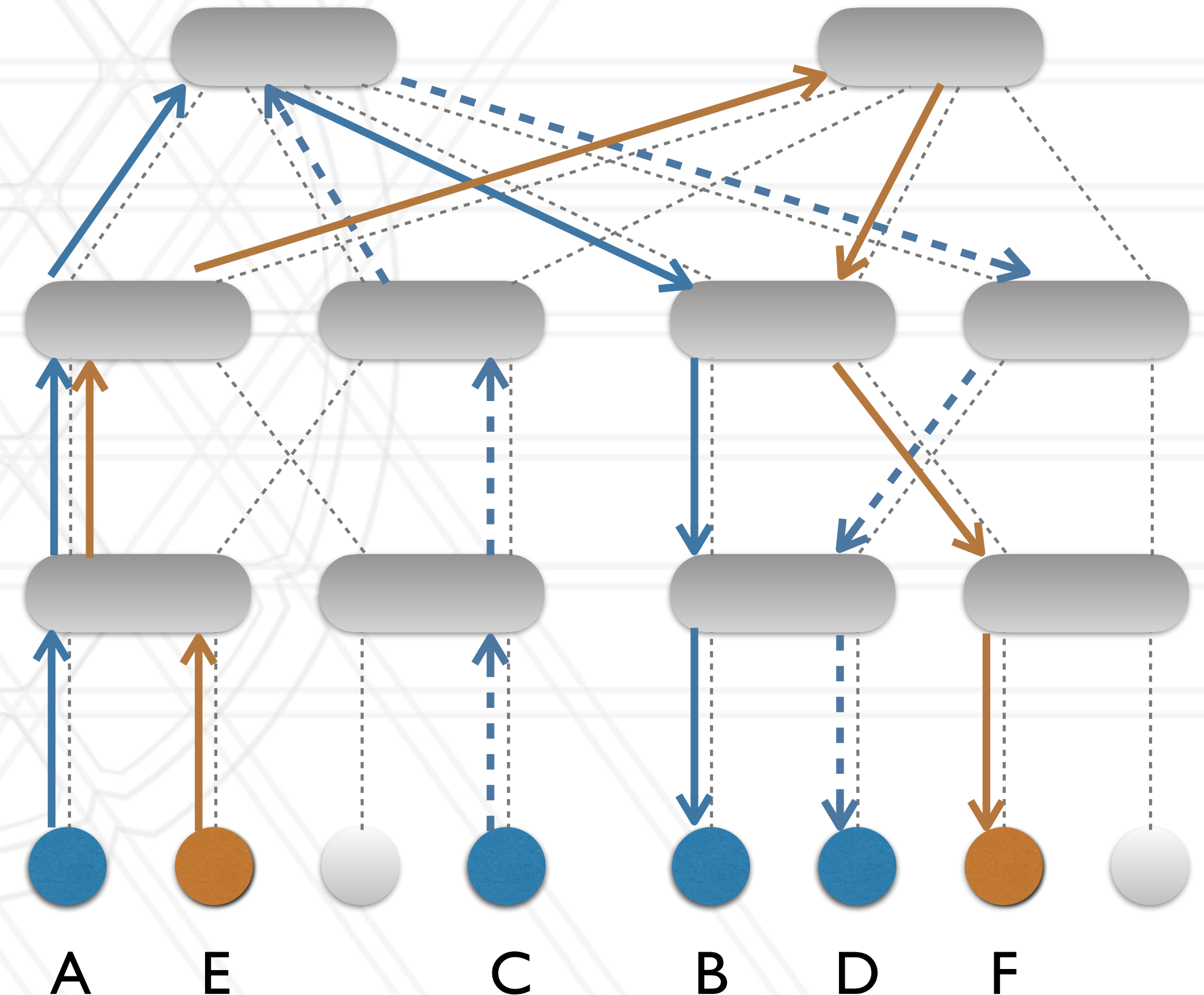


AFAR: adaptive flow aware routing

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

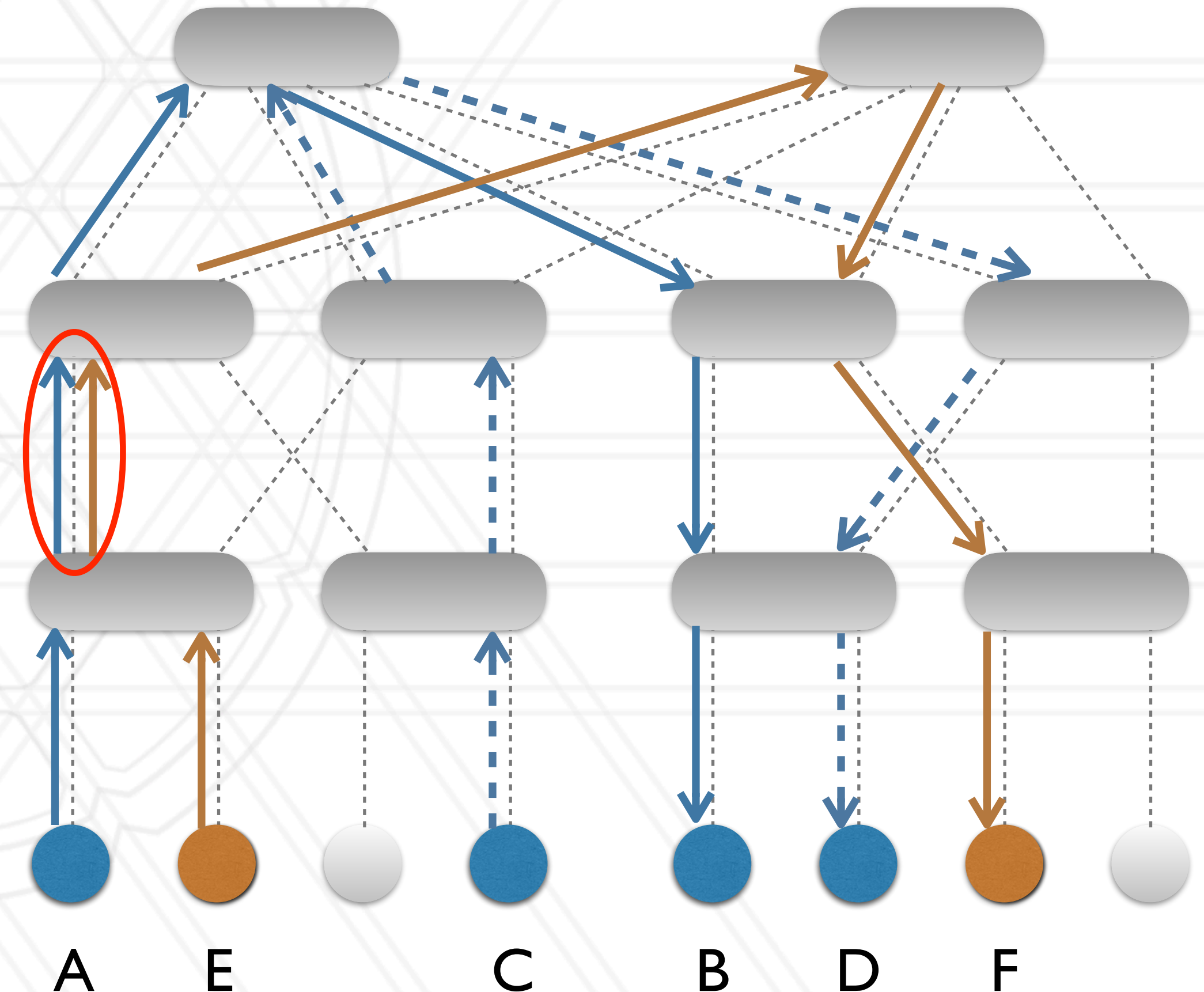


AFAR: adaptive flow aware routing

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

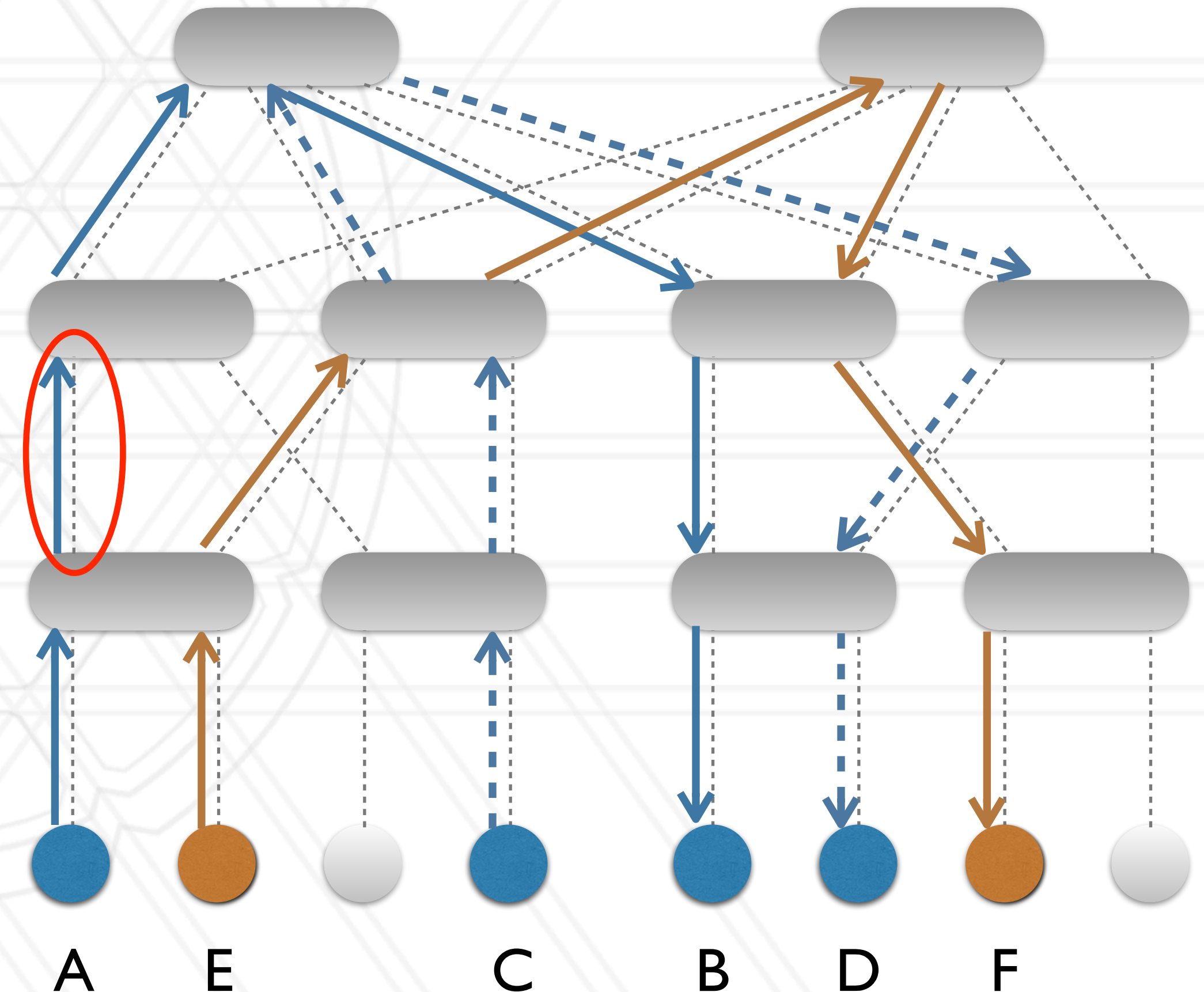


AFAR: adaptive flow aware routing

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

When do parallel programs perform I/O?

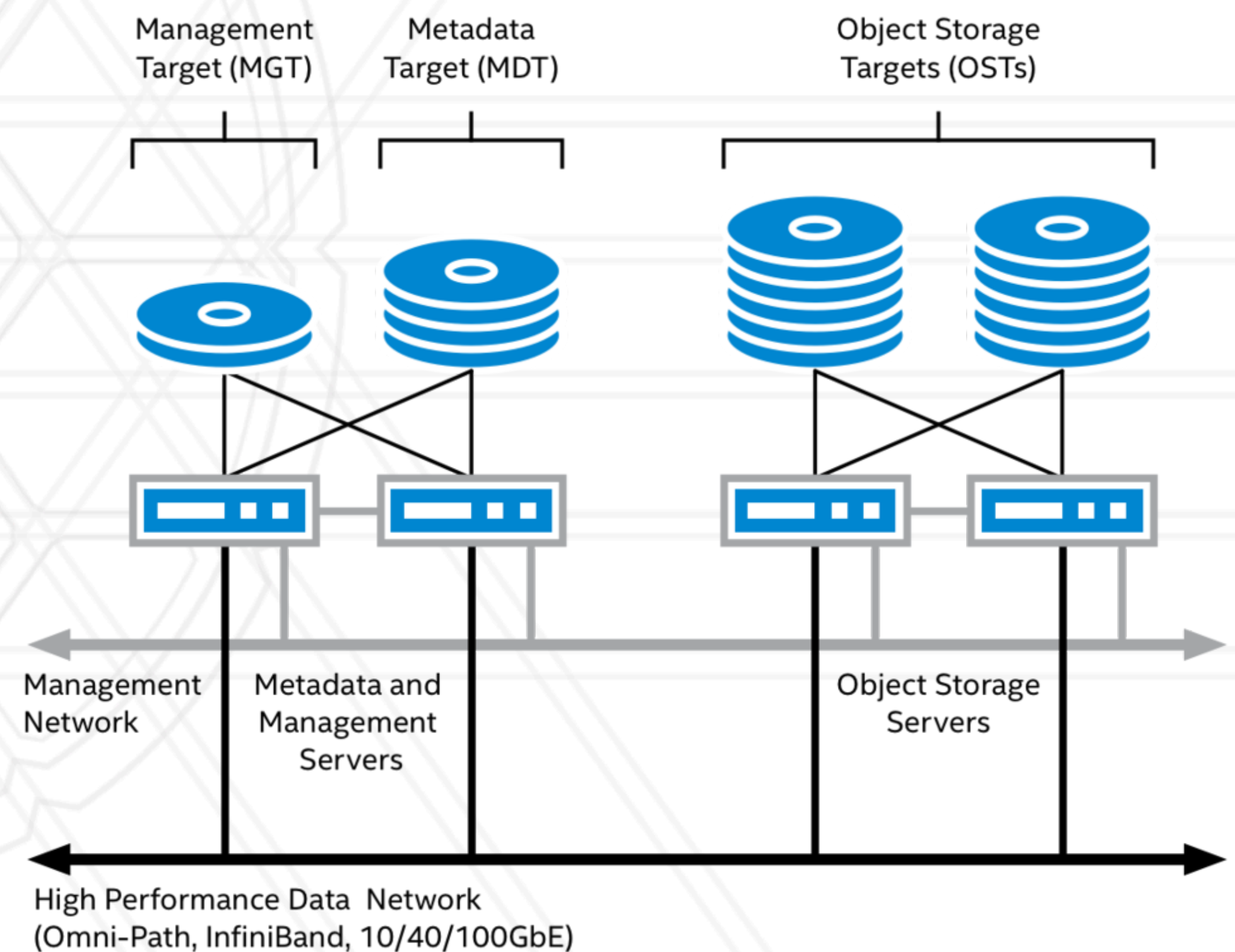
- Reading input datasets
- Writing numerical output
- Writing checkpoints

Non-parallel I/O

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

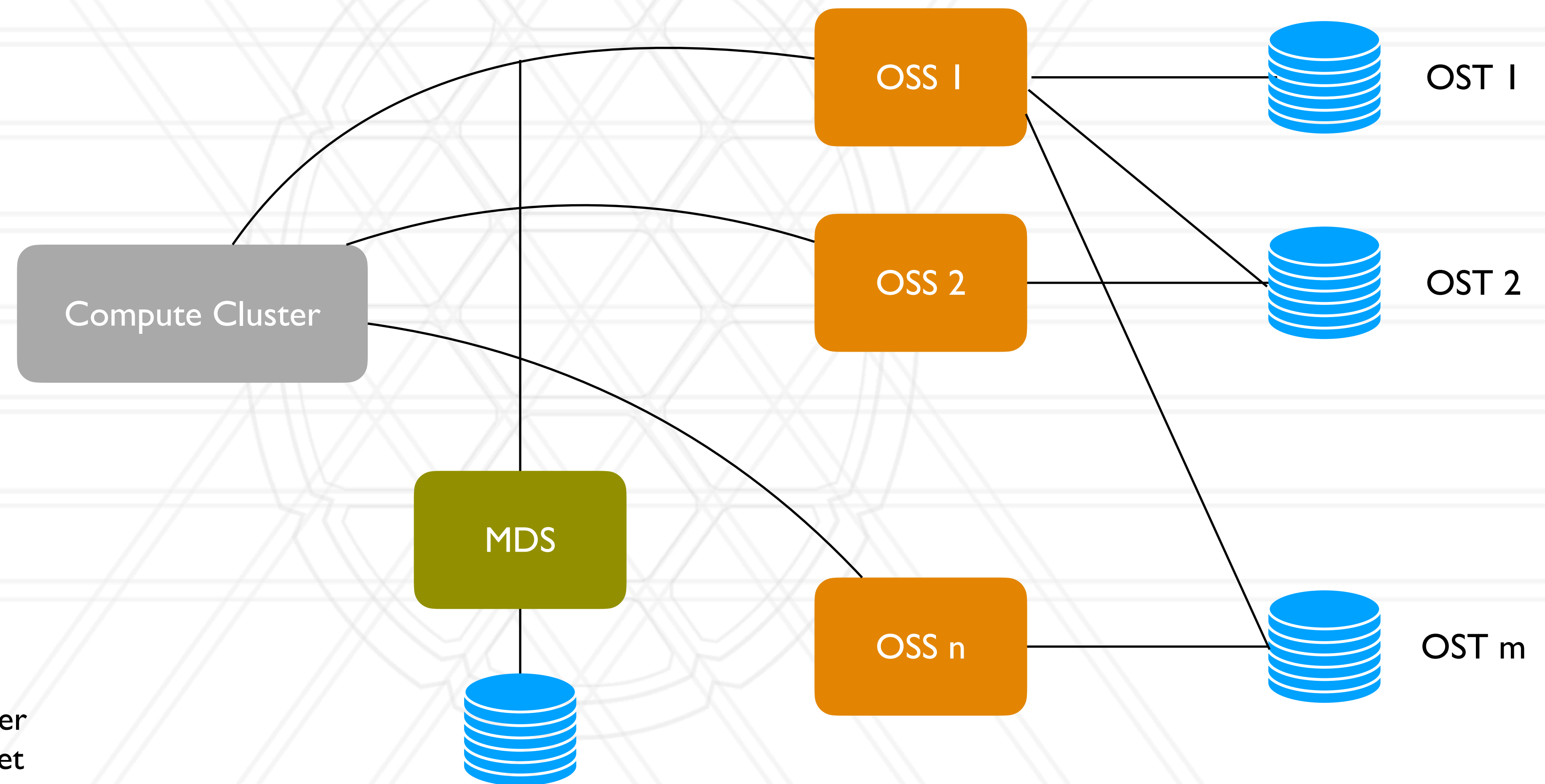
Parallel filesystem

- 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



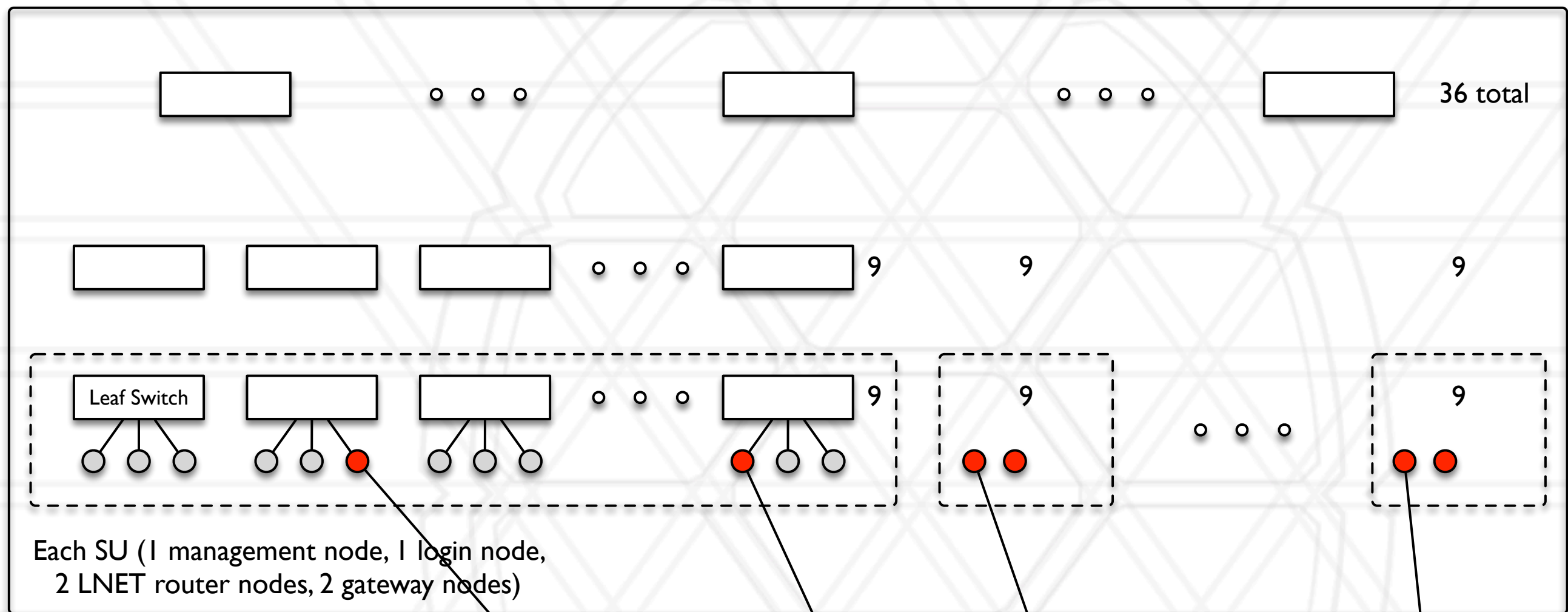
http://wiki.lustre.org/Introduction_to_Lustre

Parallel filesystem

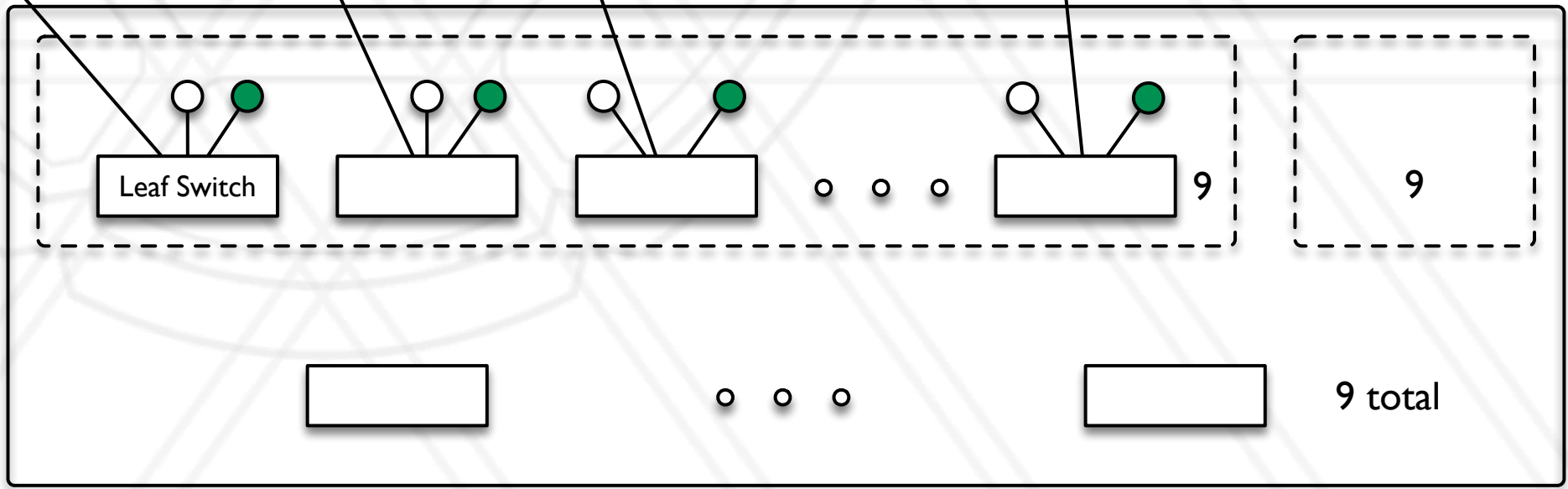


MDS = Metadata Server
MDT = Metadata Target
OSS = Object Storage Server
OST = Object Storage Target

Links between cluster and filesystem



- Compute node
- LNET router node
- Object storage server (OSS)



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

How do parallel filesystems help?

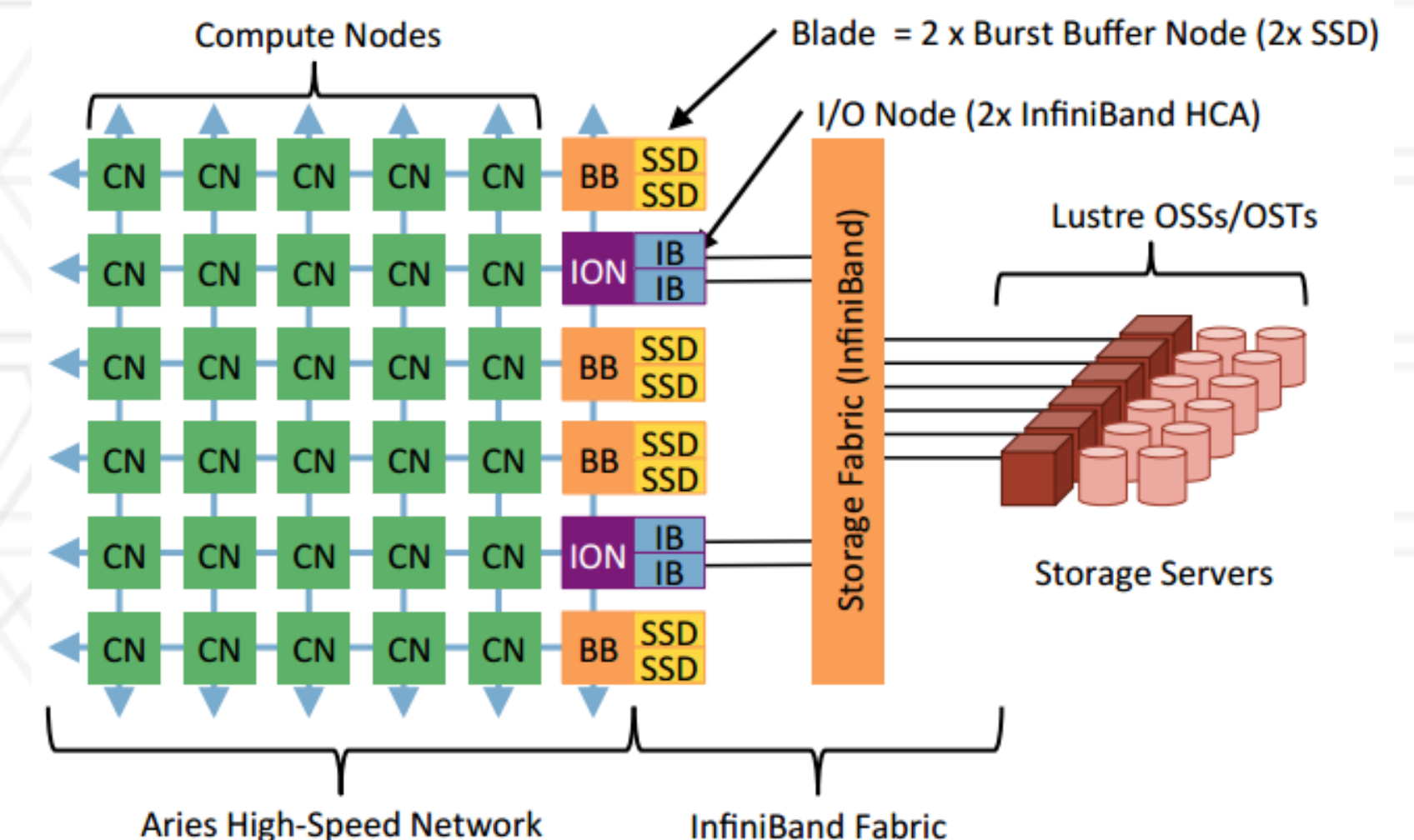
- Improve I/O bandwidth by spreading reads/writes across multiple OSTs (disks), even for single files
- Files can be striped within and across multiple I/O servers (OSSs)
- Each client (compute node) runs an I/O daemon to interact with the parallel filesystem mounted on it
- MDS serves file metadata (ownership, permissions), and inode/directory updates

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



<https://datainscience.com/to-burst-or-not-to-burst-that-is-the-question/>

Burst buffer use cases

- Storing checkpoint data
- Prefetching input data
- Workflows that couple simulations to analysis/visualization tasks

I/O libraries

- High-level libraries: HDF5, NetCDF
 - Both libraries and file formats for n-dimensional data
- Middleware: MPI-IO
 - Support for POSIX like I/O in MPI for parallel I/O
- Low-level: POSIX IO
 - Standard Unix/Linux I/O interface

Different I/O patterns

- One process reading/writing all the data
- Multiple processes reading/writing data from/to shared file
- Multiple processes reading/writing data from/to different files
- Performance depends upon number of readers/writers (how many processes/threads etc.), file sizes, filesystem etc.

I/O profiling tools

- Darshan
 - Lightweight profiling tool from Argonne National Laboratory
- Recorder
 - Research tool from UIUC
 - Tracing framework for capturing I/O activity
 - Provides support for different I/O libraries: HDF5, MPI-IO, POSIX I/O



UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu