L06: parallel HW (3)

- Stefano Cozzini
- CNR-IOM and eXact lab srl

Recap so far

- Basic MPI
- Point to point communication:
  - MPI send/recv
  - Blocking/Non blocking
- Collective operations
- Exercises

Today

- Part 1: lab
    - Collect doubts/questions on assigment01   (if any)
    - Solve together exercise1
    - Play together with blocking/non blocking operation
- Part 2;
    - parallel HW

Overview of MPI send modes

MPI has a number of different "send modes." These represent different choices of buffering (where is the data kept until it is received) and synchronization (when does a send complete).

- MPI_Send
  - MPI_Send will not return until you can use the send buffer. It may or may not block (it is allowed to buffer, either on the sender or receiver side, or to wait for the matching receive).
- MPI_Bsend
  - May buffer; returns immediately and you can use the send buffer. A late add-on to the MPI specification. Should be used only when absolutely necessary.
- MPI_Ssend
  - will not return until matching receive posted
- MPI_Rsend
  - May be used ONLY if matching receive already posted. User responsible for writing a correct program.

Overview of MPI send modes

- Recommendation:
  - The best performance is likely if you can write your program so that you could use just MPI_Ssend; in that case, an MPI implementation can completely avoid buffering data.
  - Use MPI_Send instead; this allows the MPI implementation the maximum flexibility in choosing how to deliver your data
  - (Unfortunately, one vendor has chosen to have MPI_Send emphasize buffering over performance; on that system, MPI_Ssend may perform better.)
  - If nonblocking routines are necessary, then try to use MPI_Isend or MPI_Irecv. Use MPI_Bsend only when it is too inconvienent to use MPI_Isend
  - The remaining routines, MPI_Rsend, MPI_Issend, etc., are rarely used but may be of value in writing system-dependent message-passing code entirely within MPI.

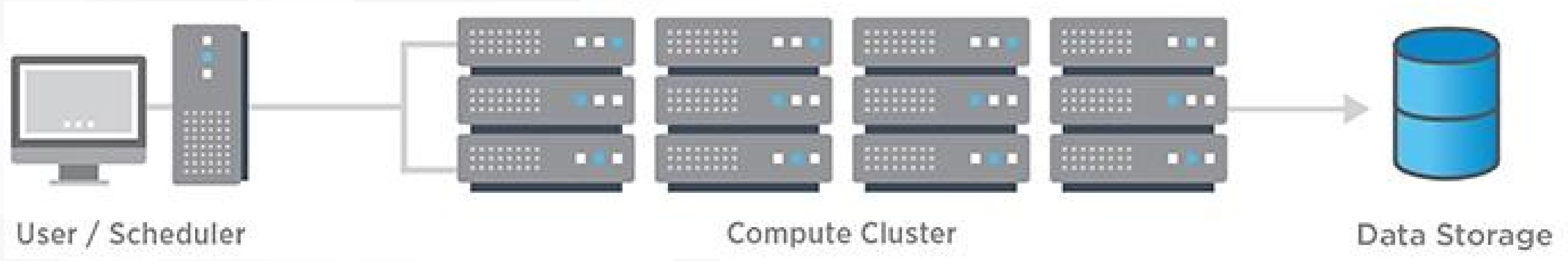
Exercise:

- Modify pi_mpi.c to use collective operation instead of naive communication algorithm

Agenda

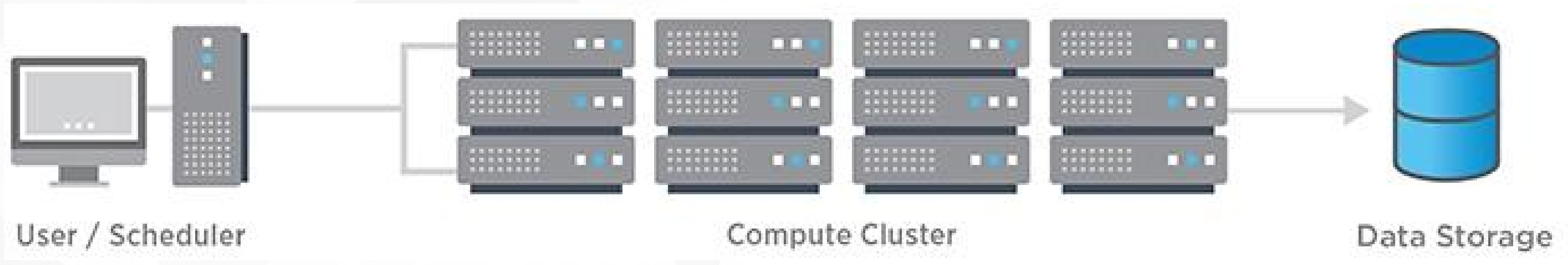
- HPC cluster components: HW
  - Buses on nodes
  - Networks
  - Storage
- Software tools and how to use them at best
  - MPI libraries
- EXERCISE:
  - Start evaluating the overall performance of more than one node

The basic distributed memory machine: clusters



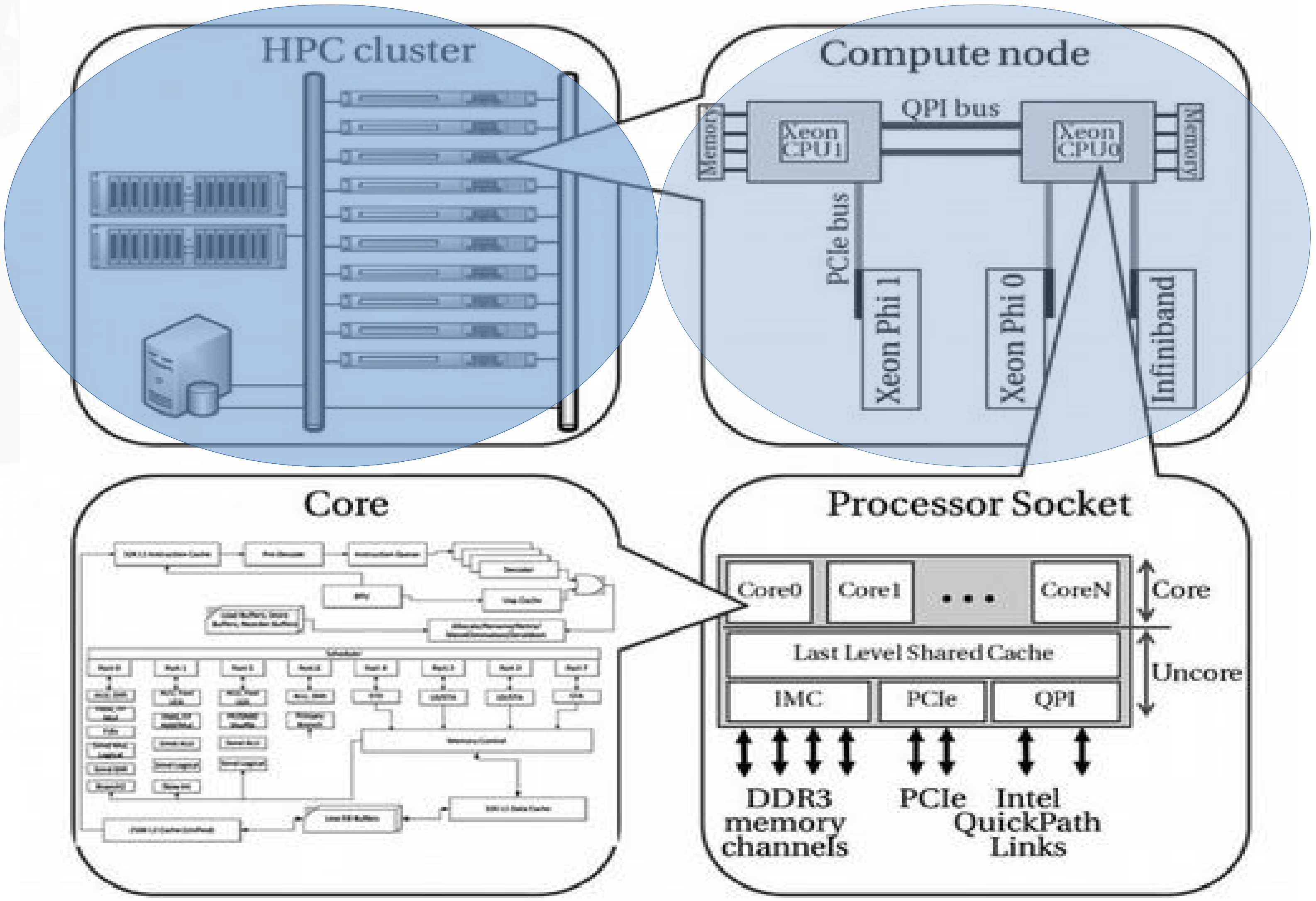
- Several computers (nodes) often in special cases for easy mounting in a rack
- One or more networks (interconnects) to hook the nodes together
- Storage facilities.

The basic distributed memory machine: clusters

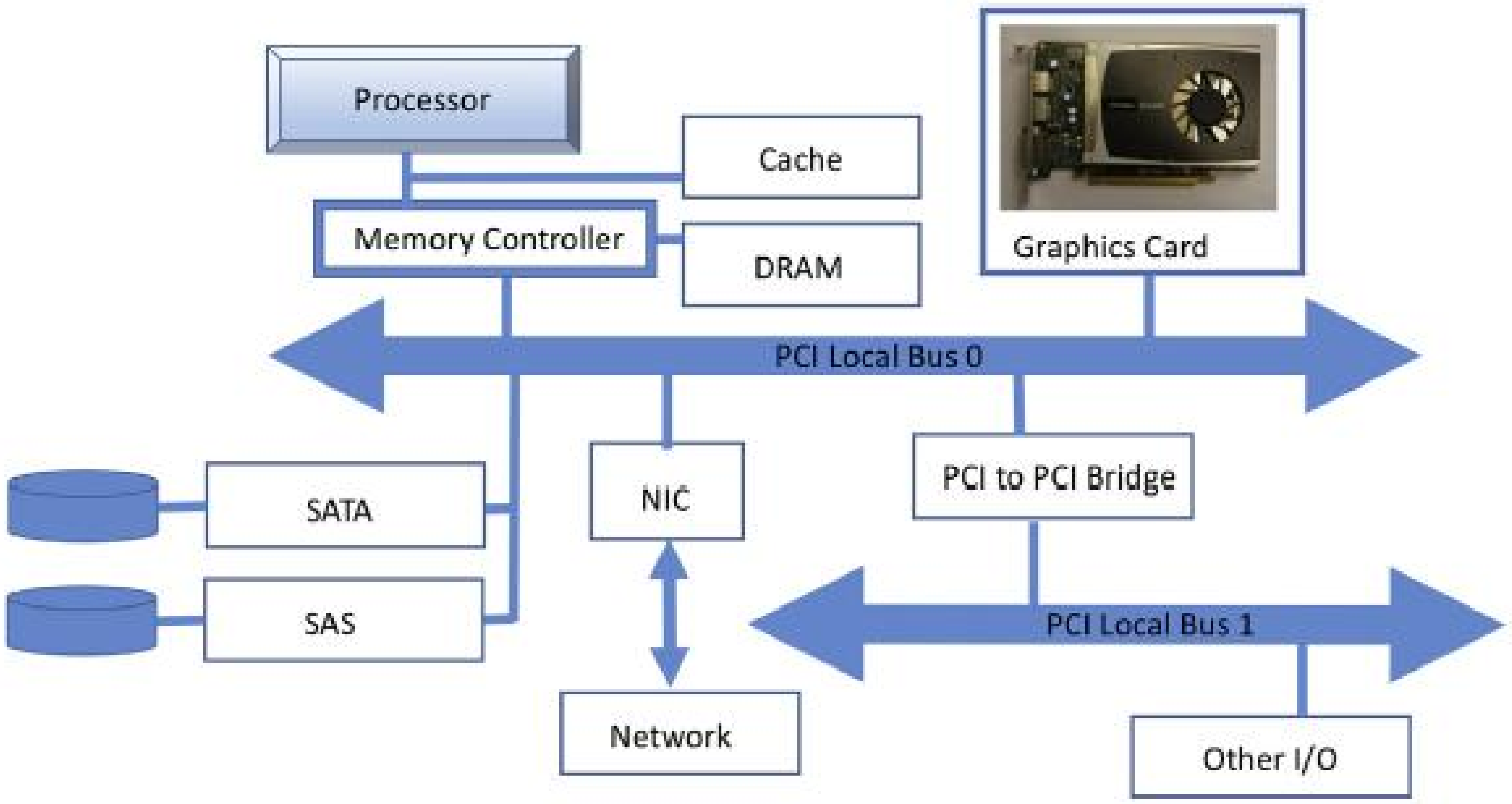


- The performance of the system are influenced by:
  - Features of the node (RAM/cores/CPU frequency/ acceler ator)
  - Features (Topology and other) of the interconnection network

The building blocks of the HPC cluster



Buses within a computer

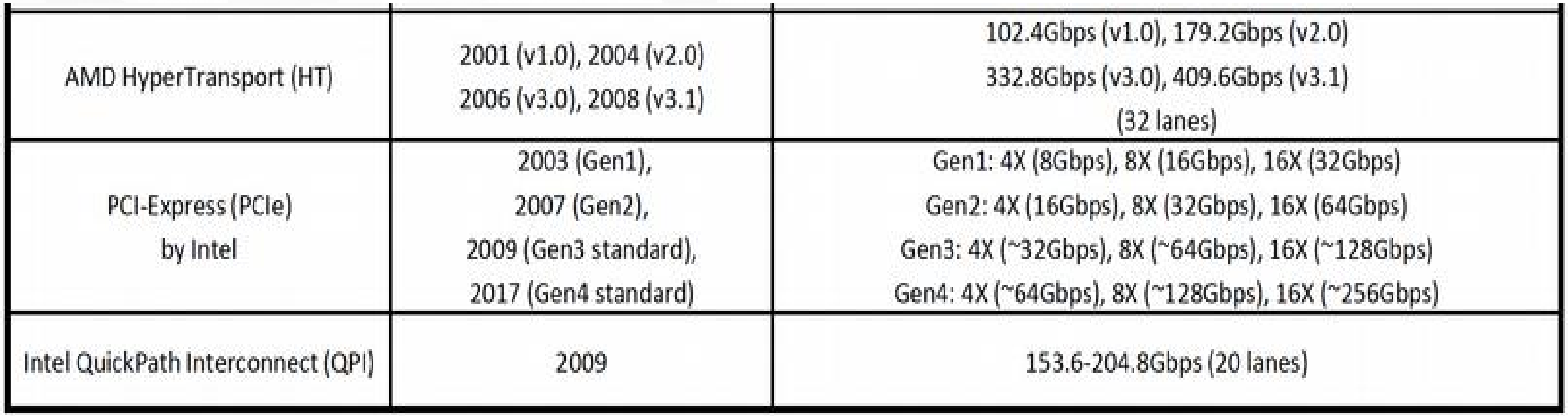


Buses on modern HPC nodes

- Peripheral Component Interconnect (PCI) buses:
  - PCI: Developed by Intel in 1992
    - several version : v3.0 last one in 2004
  - PCI-X: designed in 1999
    - 66 MHz (can be found on older servers)
    - 133 MHz (most common on modern servers)
  - PCIe: designed adopted in 2004
    - version v4.0 recently released
    - Version 2.0/version 3.0 adopted on modern HPC nodes
- Several of them on one node with different characteristics

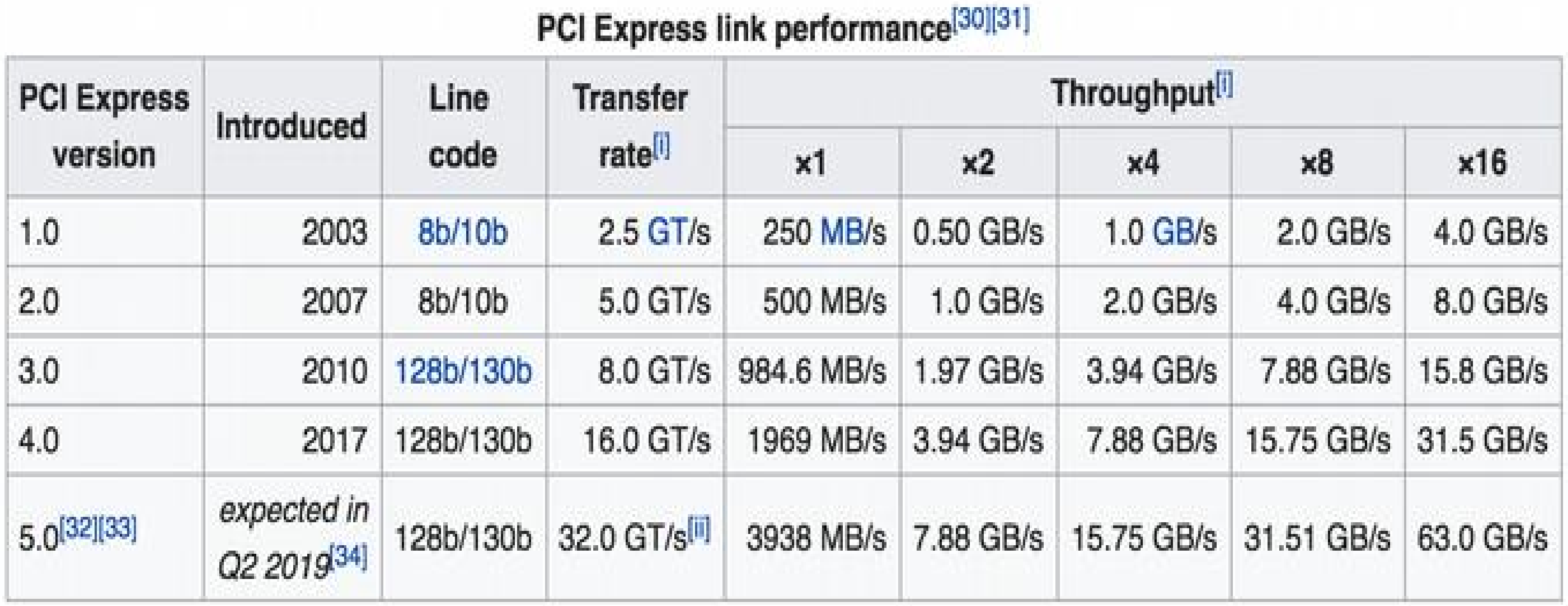
Communication interfaces within server

- Recent trends in I/O interfaces show that they are nearly matching state of the art network speeds



AMD HyperTransport (HT)	2001 (v1.0), 2004 (v2.0) 2006 (v3.0), 2008 (v3.1)	102.4Gbps (v1.0), 179.2Gbps (v2.0) 332.8Gbps (v3.0), 409.6Gbps (v3.1) (32 lanes)
PCI-Express (PCIe) by Intel	2003 (Gen1), 2007 (Gen2), 2009 (Gen3 standard), 2017 (Gen4 standard)	Gen1: 4X (8Gbps), 8X (16Gbps), 16X (32Gbps) Gen2: 4X (16Gbps), 8X (32Gbps), 16X (64Gbps) Gen3: 4X (~32Gbps), 8X (~64Gbps), 16X (~128Gbps) Gen4: 4X (~64Gbps), 8X (~128Gbps), 16X (~256Gbps)
Intel QuickPath Interconnect (QPI)	2009	153.6-204.8Gbps (20 lanes)

PCI-express speed (from wikipedia)

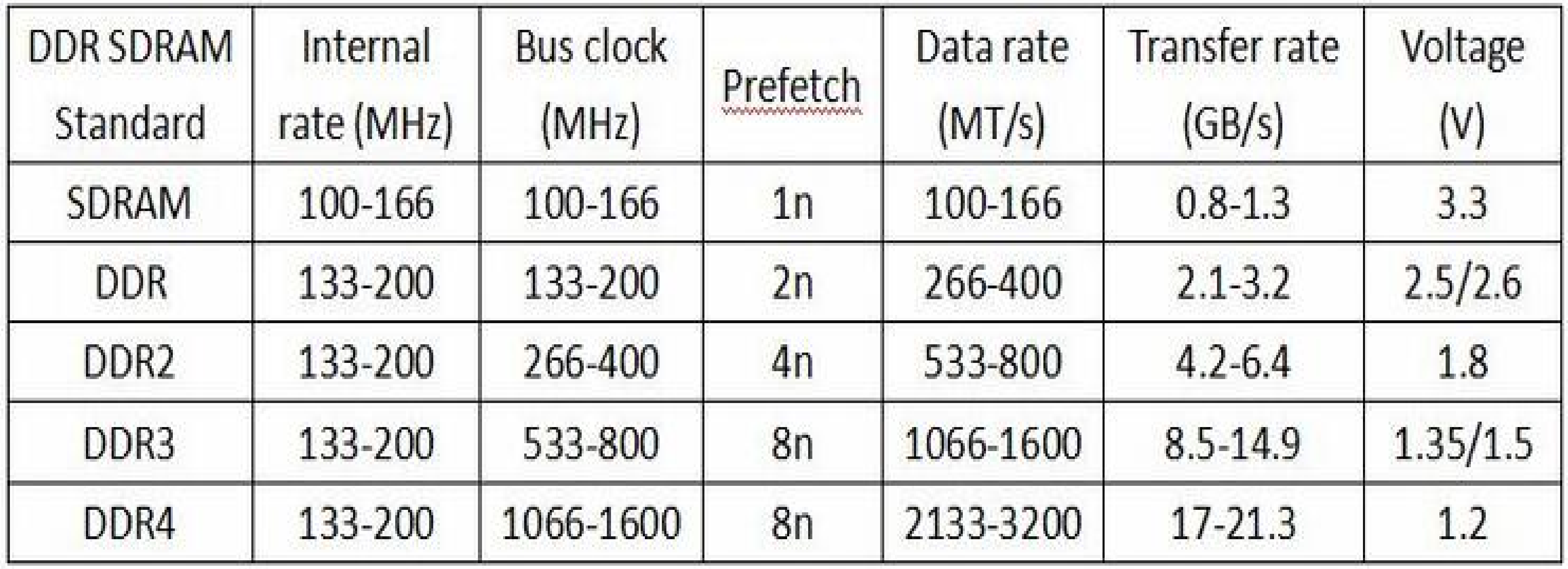


PCI Express link performance[30][31]

PCI Express version	Introduced	Line code	Transfer rate[i]	Throughput[i]				
				×1	×2	×4	×8	×16
1.0	2003	8b/10b	2.5 GT/s	250 MB/s	0.50 GB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s
2.0	2007	8b/10b	5.0 GT/s	500 MB/s	1.0 GB/s	2.0 GB/s	4.0 GB/s	8.0 GB/s
3.0	2010	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.88 GB/s	15.8 GB/s
4.0	2017	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.88 GB/s	15.75 GB/s	31.5 GB/s
5.0[32][33]	*expected in Q2 2019*[34]	128b/130b	32.0 GT/s[ii]	3938 MB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s	63.0 GB/s

How fast are memories ?

- Synchronous dynamic random-access memory (SDRAM)
- Double Data Rate (DDR) with ECC
- DDR ->DDR2->DDR3-->DDR4

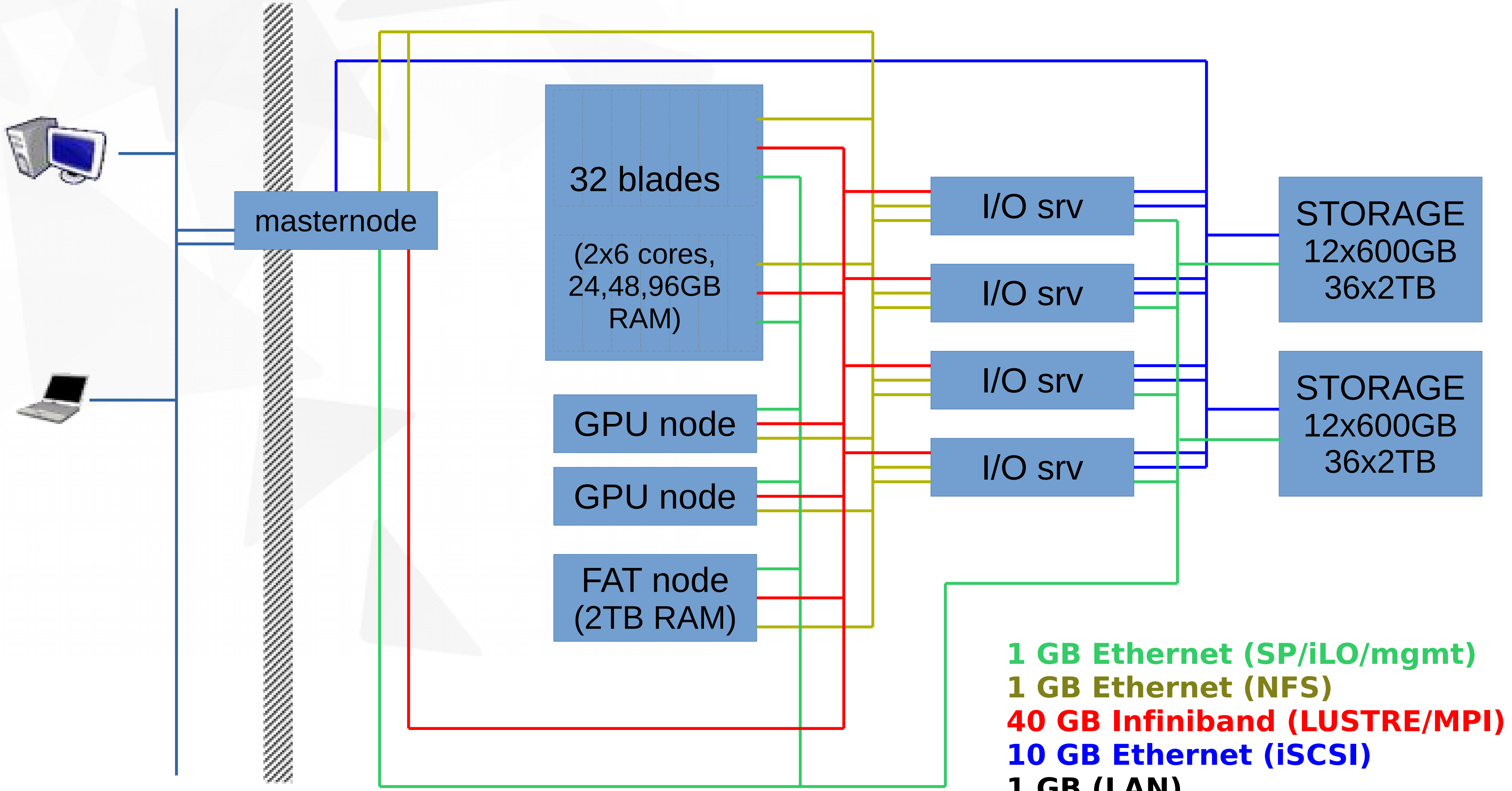


DDR SDRAM Standard	Internal rate (MHz)	Bus clock (MHz)	Prefetch	Data rate (MT/s)	Transfer rate (GB/s)	Voltage (V)
SDRAM	100-166	100-166	1n	100-166	0.8-1.3	3.3
DDR	133-200	133-200	2n	266-400	2.1-3.2	2.5/2.6
DDR2	133-200	266-400	4n	533-800	4.2-6.4	1.8
DDR3	133-200	533-800	8n	1066-1600	8.5-14.9	1.35/1.5
DDR4	133-200	1066-1600	8n	2133-3200	17-21.3	1.2

Network Clusters classification

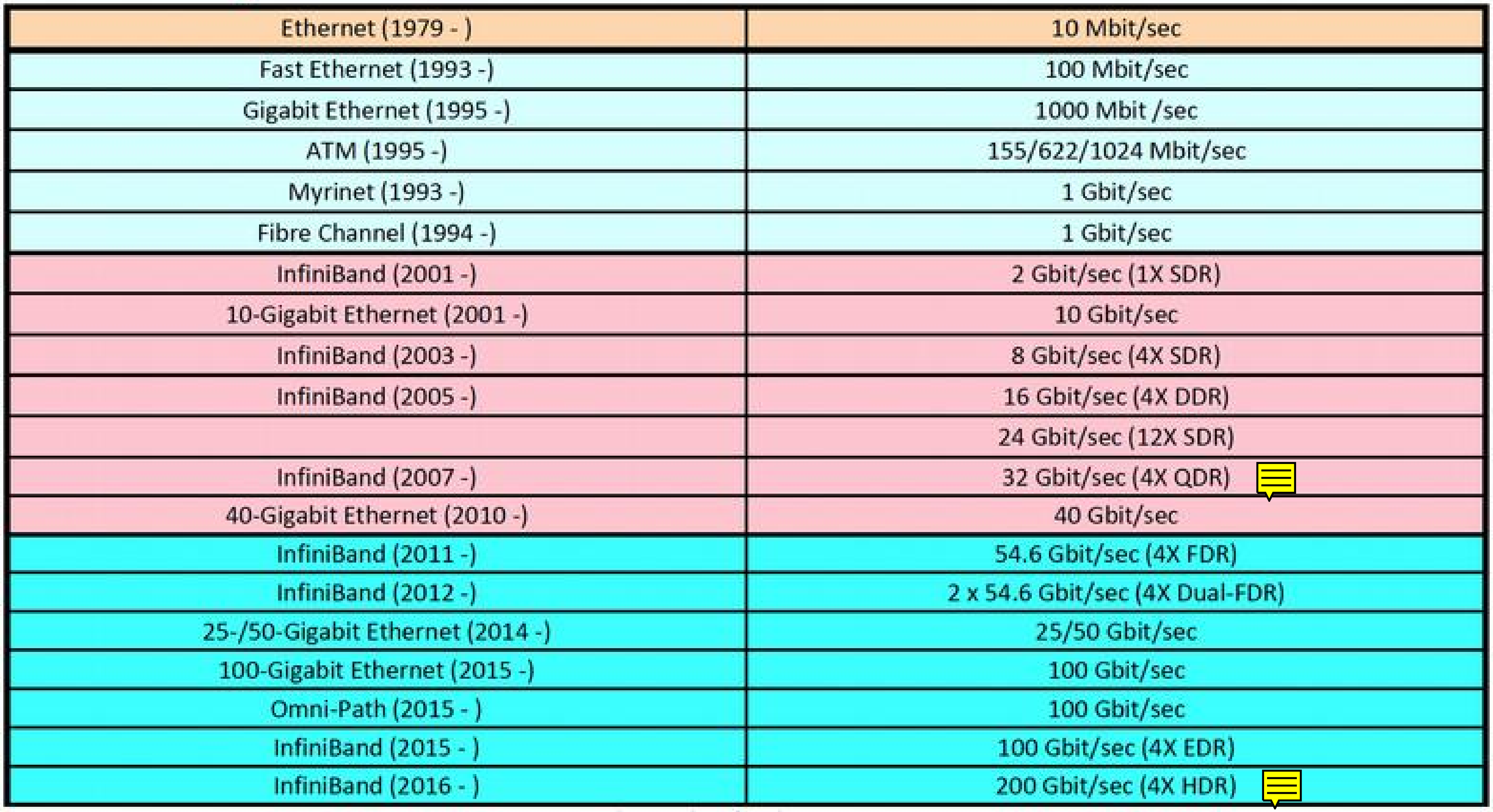
- HIGH SPEED NETWORK
  - parallel computation
    - low latency /high bandwidth
    - Usual choices: Infiniband…
- I/O NETWORK
  - I/O requests (NFS and/or parallel FS)
    - latency not fundamental/ good bandwidth
    - GIGABIT could be ok /10Gb and/or Infiniband better
- Management network
  - management traffic
    - any standard network

Cluster example (internal network)



1 GB Ethernet (SP/iLO/mgmt)
1 GB Ethernet (NFS)
40 GB Infiniband (LUSTRE/MPI)
10 GB Ethernet (iSCSI)
1 GB (LAN)

Network speed acceleration in the last 15 years



Ethernet (1979 -)	10 Mbit/sec
Fast Ethernet (1993 -)	100 Mbit/sec
Gigabit Ethernet (1995 -)	1000 Mbit /sec
ATM (1995 -)	155/622/1024 Mbit/sec
Myrinet (1993 -)	1 Gbit/sec
Fibre Channel (1994 -)	1 Gbit/sec
InfiniBand (2001 -)	2 Gbit/sec (1X SDR)
10-Gigabit Ethernet (2001 -)	10 Gbit/sec
InfiniBand (2003 -)	8 Gbit/sec (4X SDR)
InfiniBand (2005 -)	16 Gbit/sec (4X DDR)
	24 Gbit/sec (12X SDR)
InfiniBand (2007 -)	32 Gbit/sec (4X QDR)
40-Gigabit Ethernet (2010 -)	40 Gbit/sec
InfiniBand (2011 -)	54.6 Gbit/sec (4X FDR)
InfiniBand (2012 -)	2 x 54.6 Gbit/sec (4X Dual-FDR)
25-/50-Gigabit Ethernet (2014 -)	25/50 Gbit/sec
100-Gigabit Ethernet (2015 -)	100 Gbit/sec
Omni-Path (2015 -)	100 Gbit/sec
InfiniBand (2015 -)	100 Gbit/sec (4X EDR)
InfiniBand (2016 -)	200 Gbit/sec (4X HDR)

Latency&bandwidth

NETWORK	Latency	Bandwidth (GB/sec)
Gigabit	70-40	~ 0.125
10G	<5	~1.250
Infiniband 4DDR	~1.5/1.9	~ 3.2
Infiniband FDR	<1.0	~ 5

What is the UNIT OF MEASURE OF LATENCY ?

Microseconds: 3 order of magnitude larger than unit of measure of FP operations

Network topology

- How the components are connected.
- Important properties
  - Diameter: maximum distance between any two nodes in the network (hop count, or # of links).
  - Nodal degree: how many links connect to each node.
  - Bisection bandwidth: The smallest bandwidth between half of the nodes to another half of the nodes.
- A good topology: small diameter, small nodal degree, large bisection bandwidth

Bisection bandwidth

- Split N nodes into two groups of N/2 nodes such that the bandwidth between these two groups is minimum: that is the bisection bandwidth

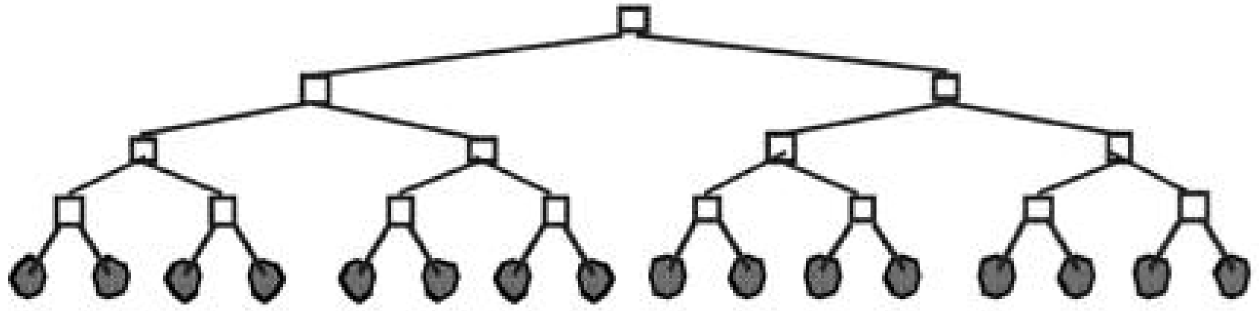
Why is Bisection Bandwidth relevant ?

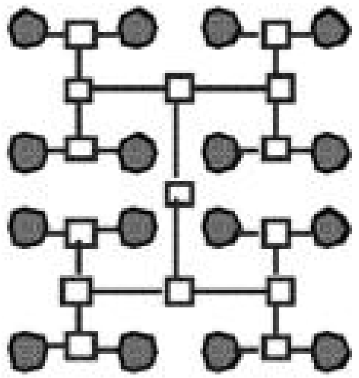
- if traffic is completely random, the probability of a message going across the two halves is ½
- if all nodes send a message, the bisection bandwidth will have to be N/2
- The concept of bisection bandwidth confirms that some network topology network is not suited for random traffic patterns
- your worst case scenario of HPC workload is to have random traffic patterns..

Topologies

- Common network topologies
  - Fat tree
  - Mesh
  - 3D torus
  - CBB (Constant Bi-sectional Bandwidth)
    - type of Fat tree can be oversubscribed 2:1 to 8:1
    - oversubscription can reduce bandwidth but most applications do not fully utilize it anyway

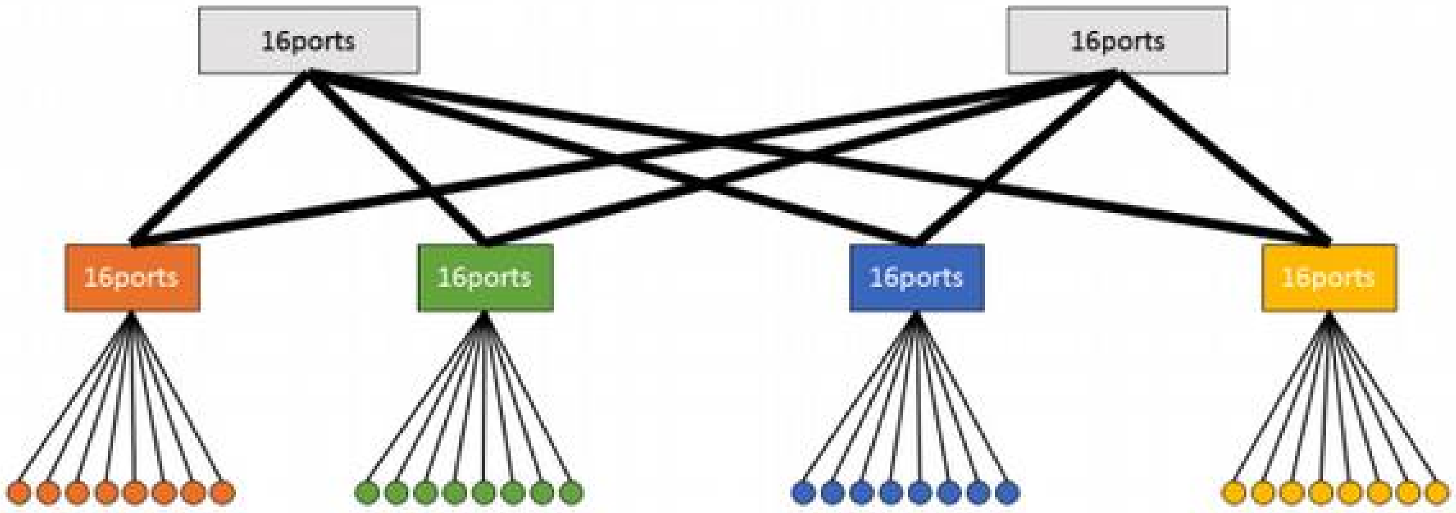
Tree Topology





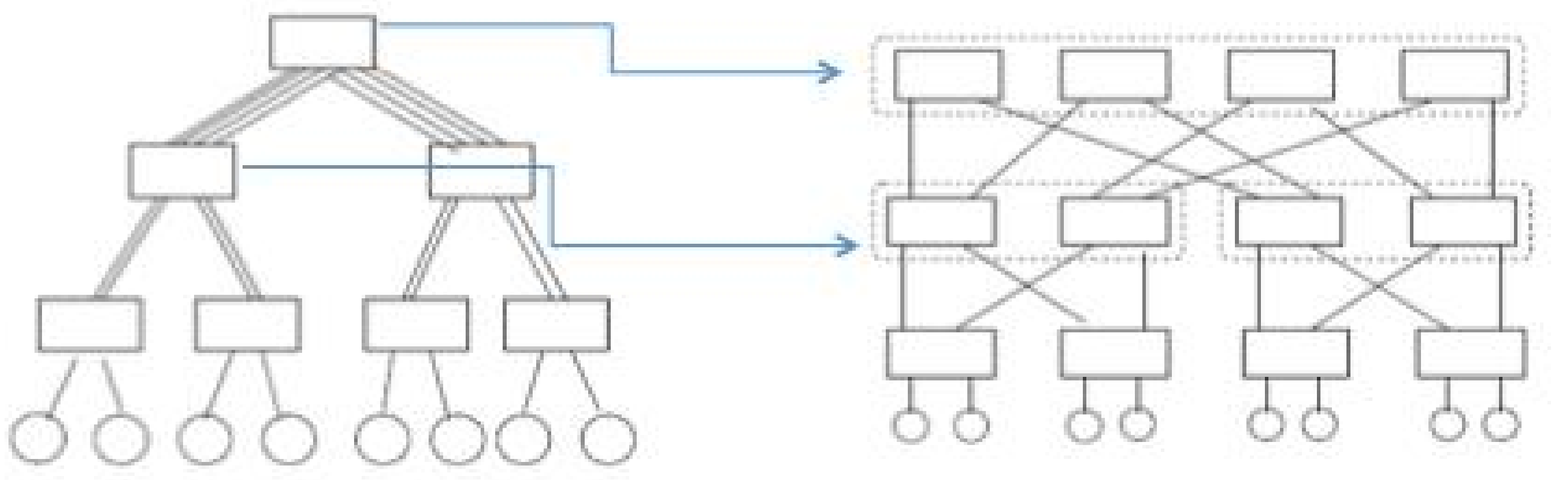
- Fixed degree, log(N) diameter, O(1) bisection bandwidth.
- Routing: up to the common ancestor than go down.

Fat tree topology



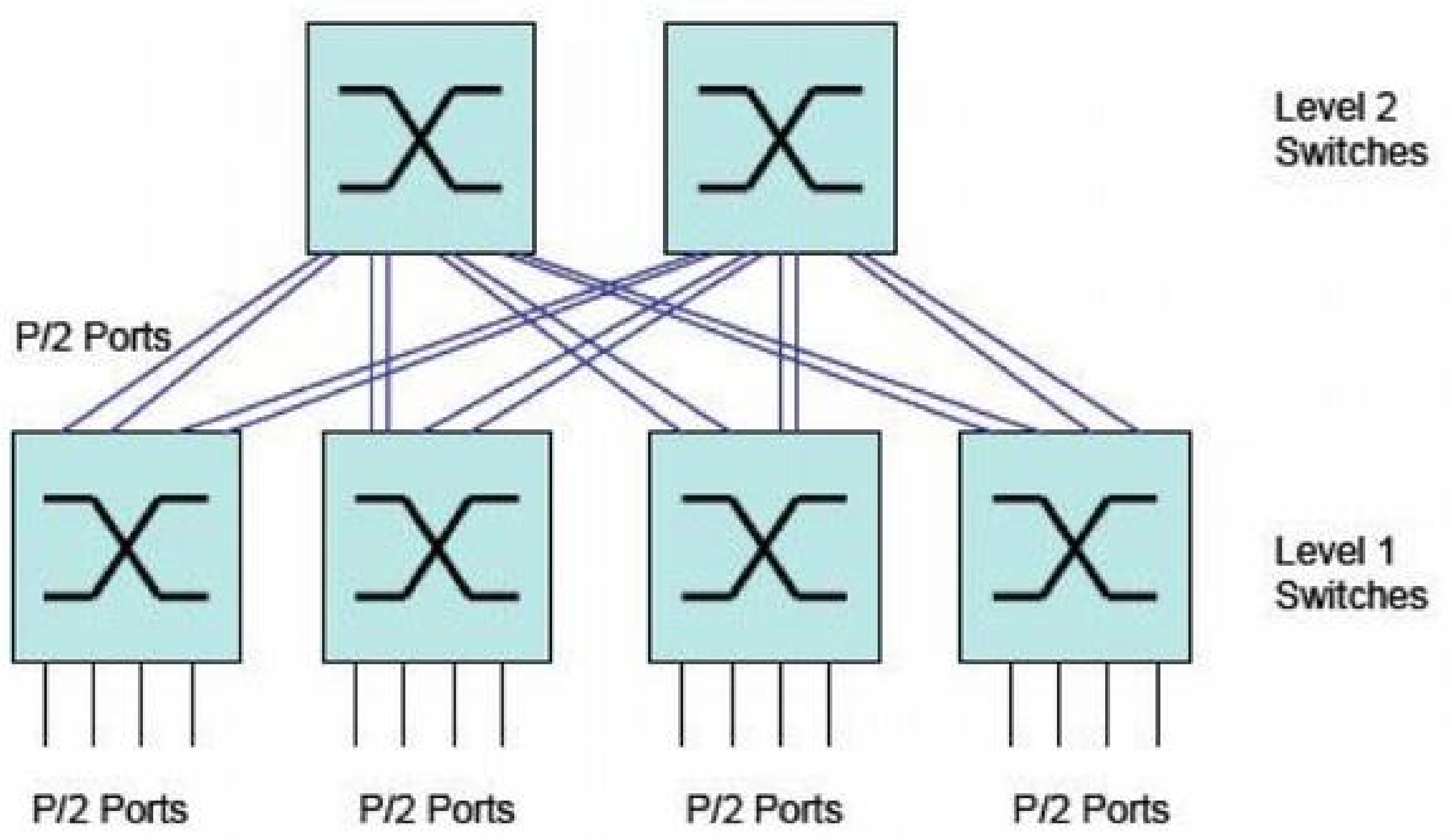
- Fatter links (really more of them) as you go up so bisection BW scales with N
- Not practical. Root is NXN switch

Practical fat tree topology



- Use smaller switches to approximate large switches.
- Most commodity large clusters use this topology.
- Also call constant bisection bandwidth network (CBB)

Two level CBB



Capabilities of high speed networks

- Intelligent Network Interface Cards
  - Support entire protocol processing completely in hardware (hardware protocol offload engines)
- Provide a rich communication interface to applications
  - User-level communication capability
- No software signaling between communication layer
  - All layers are implemented on a dedicated hardware unit, and not on a shared host CPU

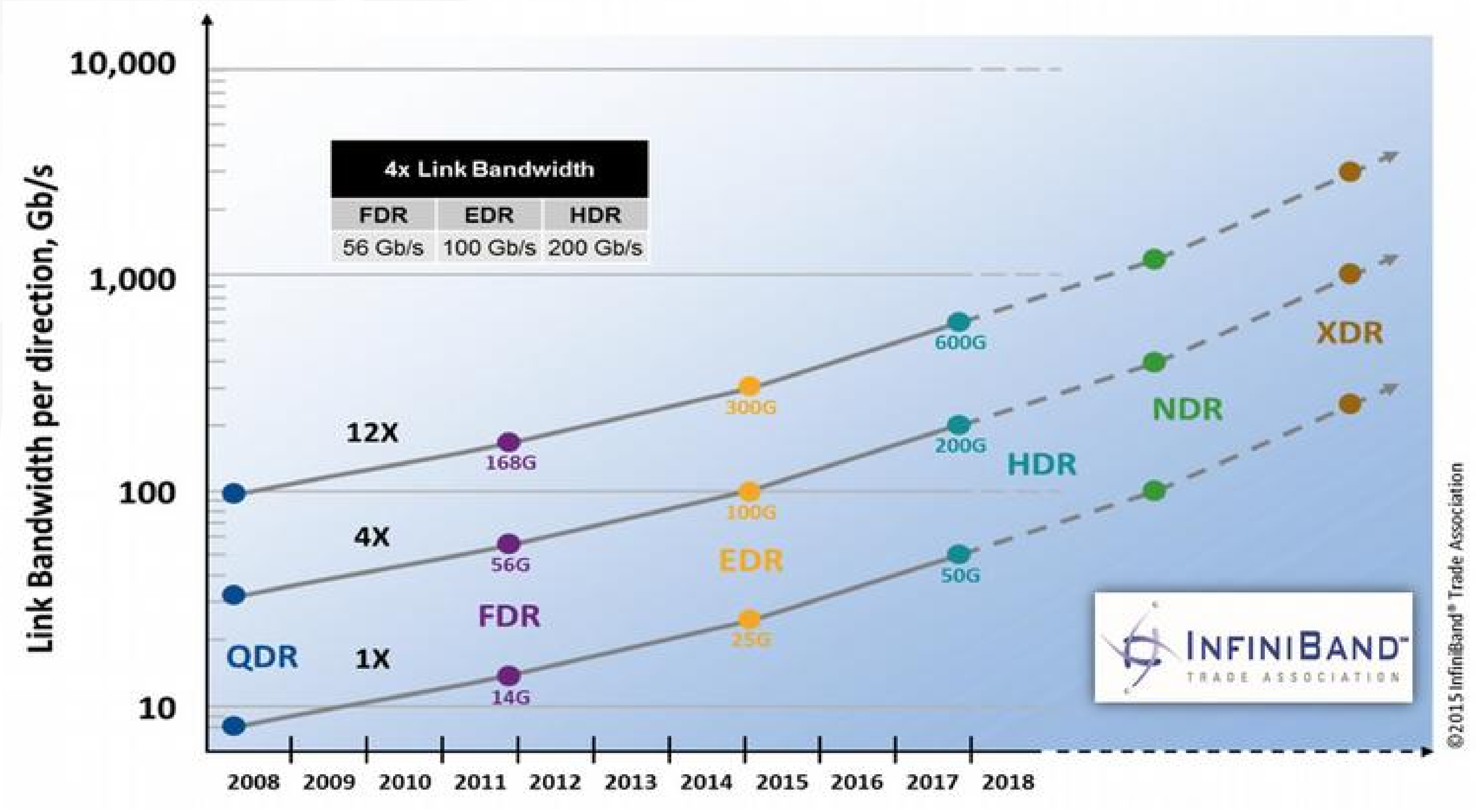
Which high speed network ?

- Infiniband
  - The defacto standard
  - 27% of ToP500 are based on infiniband
- Omni Path
  - started by Intel in 2015
  - one of the youngest HPC interconnects
  - 8.6% of Top4500 are Omni-Path systems
- Both are used behind a MPI implementation..

What is InfiniBand?

- **Industry standard** defined by the InfiniBand Trade Association – Originated in 1999
- **InfiniBand specification** defines an input/output **architecture** used to interconnect servers, communications infrastructure equipment, storage and embedded systems
- InfiniBand is a pervasive, **low-latency, high-bandwidth interconnect** which requires low processing overhead and is ideal to carry multiple traffic types (clustering, communications, storage, management) over a single connection.
- InfiniBand is now used in thousands of high-performance compute clusters and beyond that scale from small scale to large scale: **de-facto standard**

Infiniband roadmap



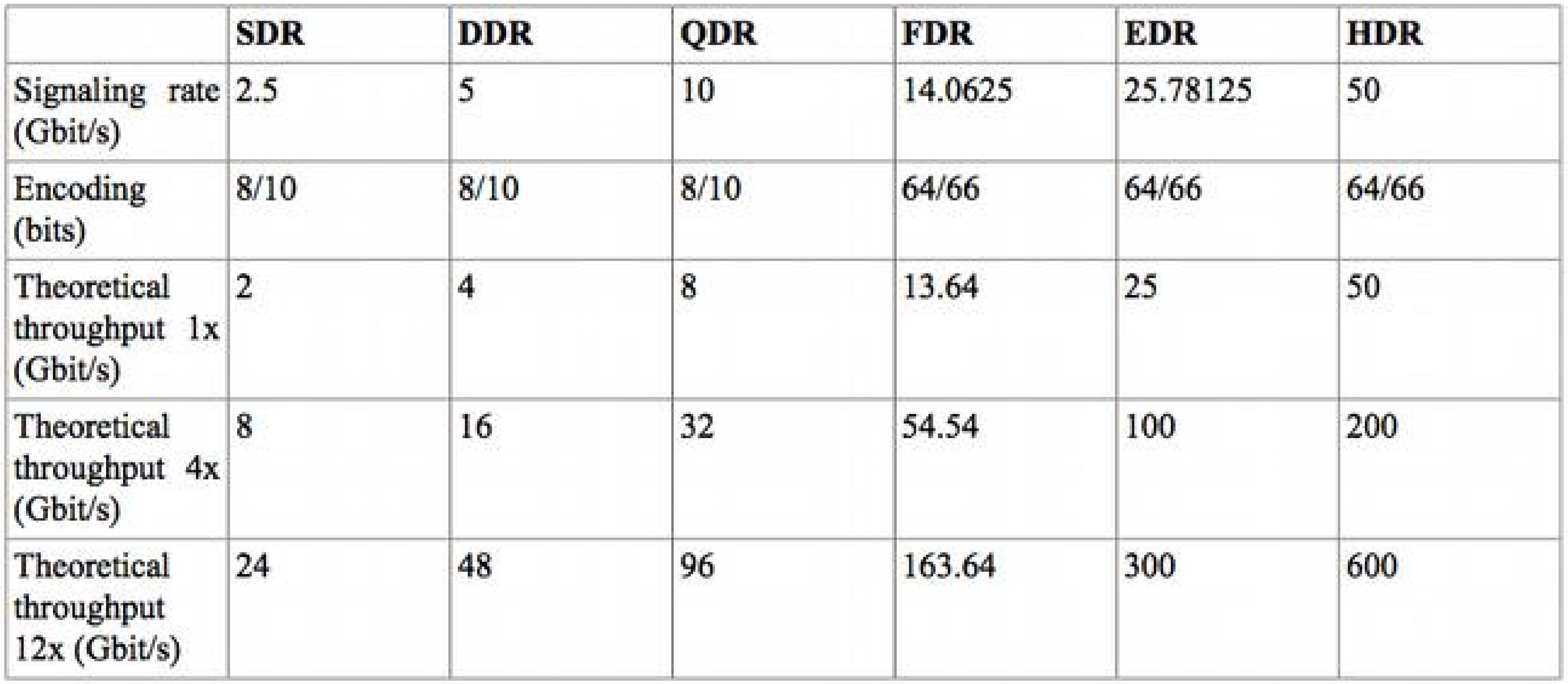
InfiniBand speed (physical layer)

- InfiniBand uses serial stream of bits for data transfer
- Linkwidth
  - 1x – One differential pair per Tx/Rx
  - 4x – Four differential pairs per Tx/Rx
  - 12x - Twelve differential pairs per Tx and per Rx
- LinkSpeed
  - Single Data Rate (SDR) - 2.5Gb/s per lane (10Gb/s for 4x)
  - Double Data Rate (DDR) - 5Gb/s per lane (20Gb/s for 4x)
  - Quad Data Rate (QDR) - 10Gb/s per lane (40Gb/s for 4x)
  - Fourteen Data Rate (FDR) - 14Gb/s per lane (56Gb/s for 4x)
  - Enhanced Data rate (EDR) - 25Gb/s per lane (100Gb/s for 4x)
- • Linkrate
  - Multiplication of the link width and link speed
  - Most common shipping today is 4x ports QDR

Infiniband speed for data transfer..

- For SDR, DDR and QDR, links use 8b/10b encoding:
  - every 10 bits sent carry 8bits of data
- Thus single, double, and quad data rates carry 2, 4, or 8 Gbit/s useful data, respectively.
- For FDR and EDR, links use 64b/66b encoding
  - every 66 bits sent carry 64 bits of data.

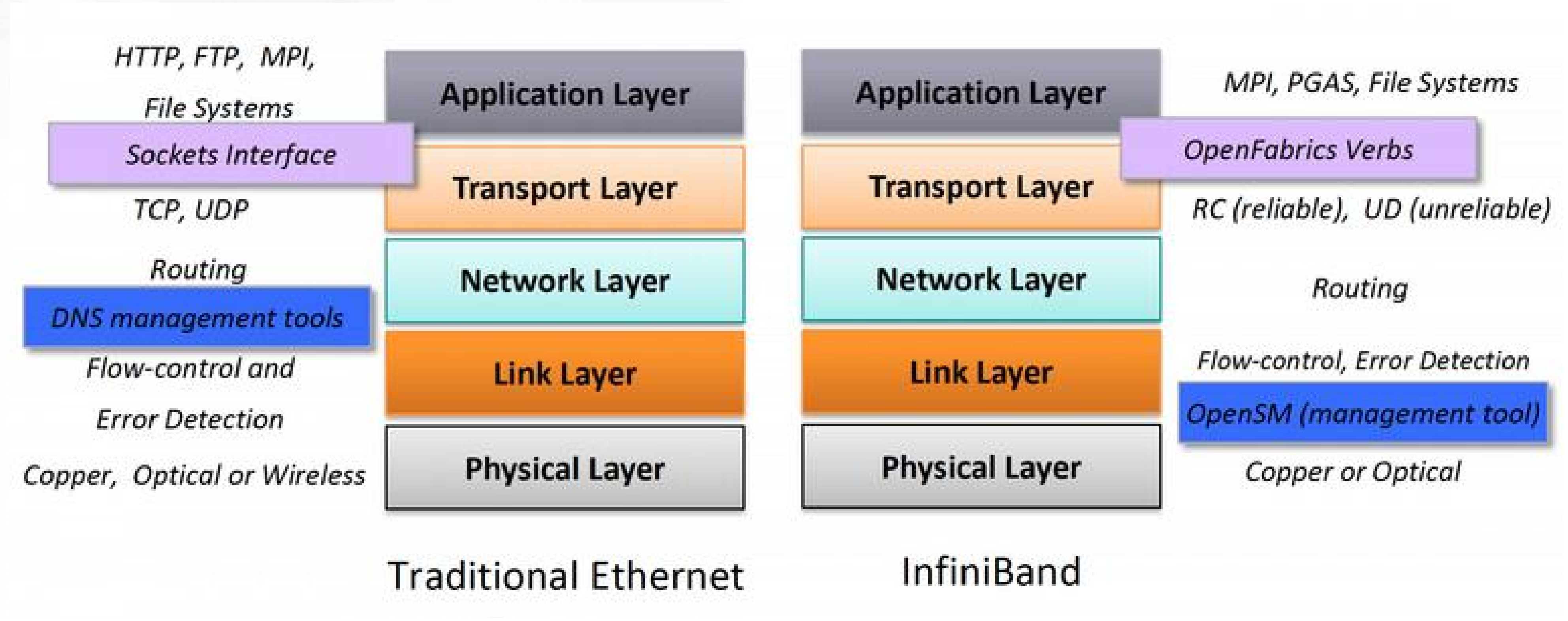
Infiniband performance



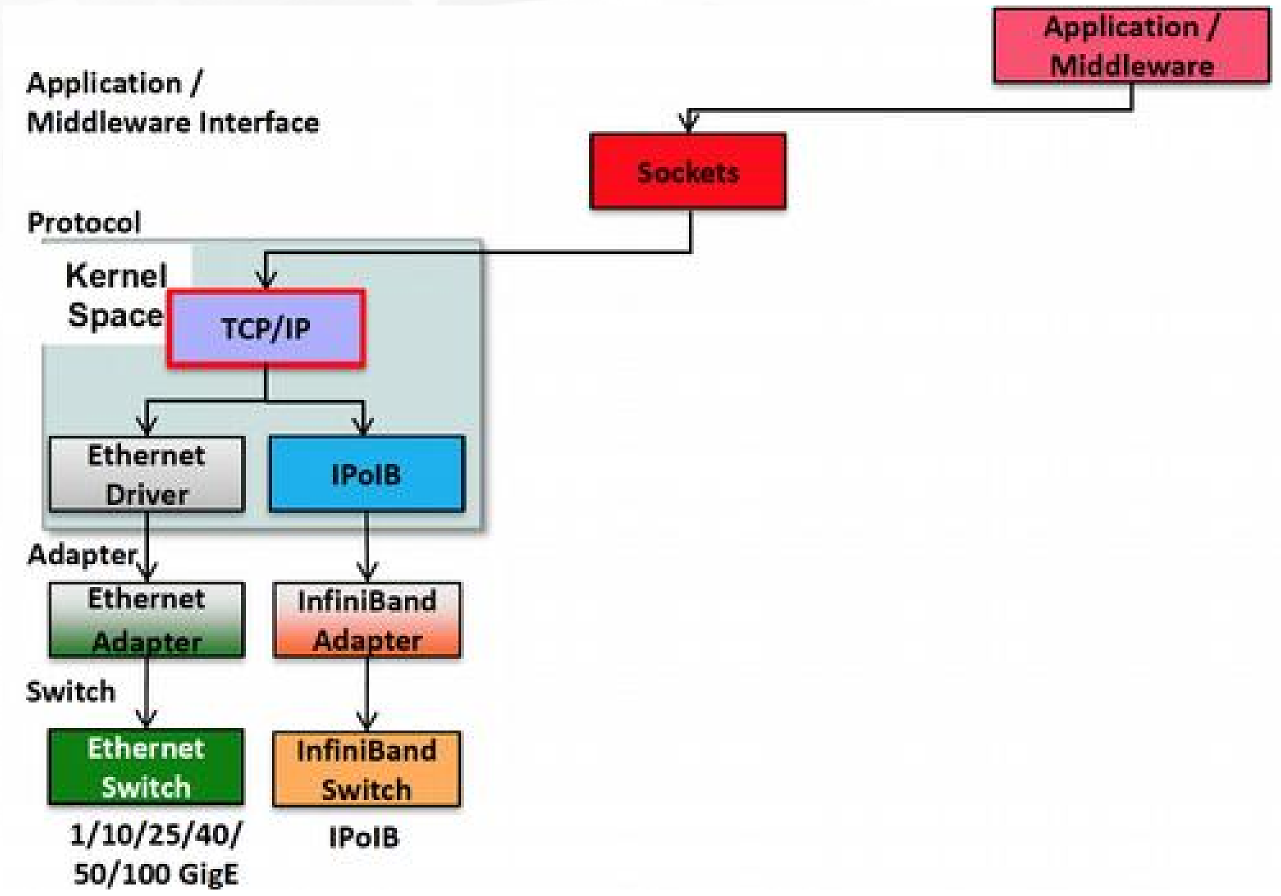
	SDR	DDR	QDR	FDR	EDR	HDR
Signaling rate (Gbit/s)	2.5	5	10	14.0625	25.78125	50
Encoding (bits)	8/10	8/10	8/10	64/66	64/66	64/66
Theoretical throughput 1x (Gbit/s)	2	4	8	13.64	25	50
Theoretical throughput 4x (Gbit/s)	8	16	32	54.54	100	200
Theoretical throughput 12x (Gbit/s)	24	48	96	163.64	300	600

We do not take into account the additional physical layer overhead requirements for common characters or software protocol requirements..

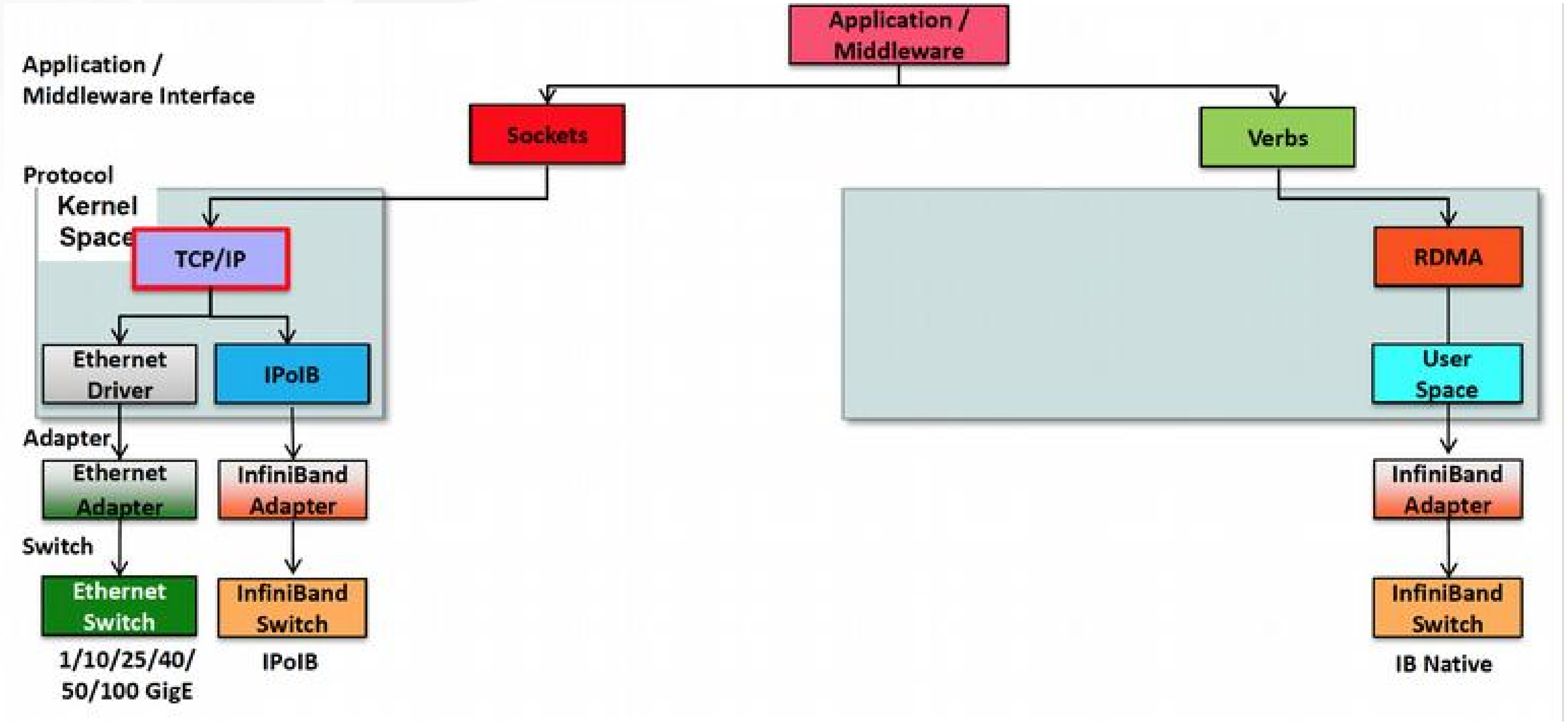
Infiniband vs Ethernet..



TCP/IP and IPoIB protocol



TCP/IP and IPoIB protocol vs native infiniband ones



IB software

- Provided by OpenFabric (www.openfabrics.org)
- Open source organization (formerly OpenIB)
- Support for Linux and Windows Design of complete stack with `best of breed' components
- Linux Distribution is now including it (check out carefully which version)
- Users can download the entire stack and run
  - Latest release is OFED 5.3

OFED...

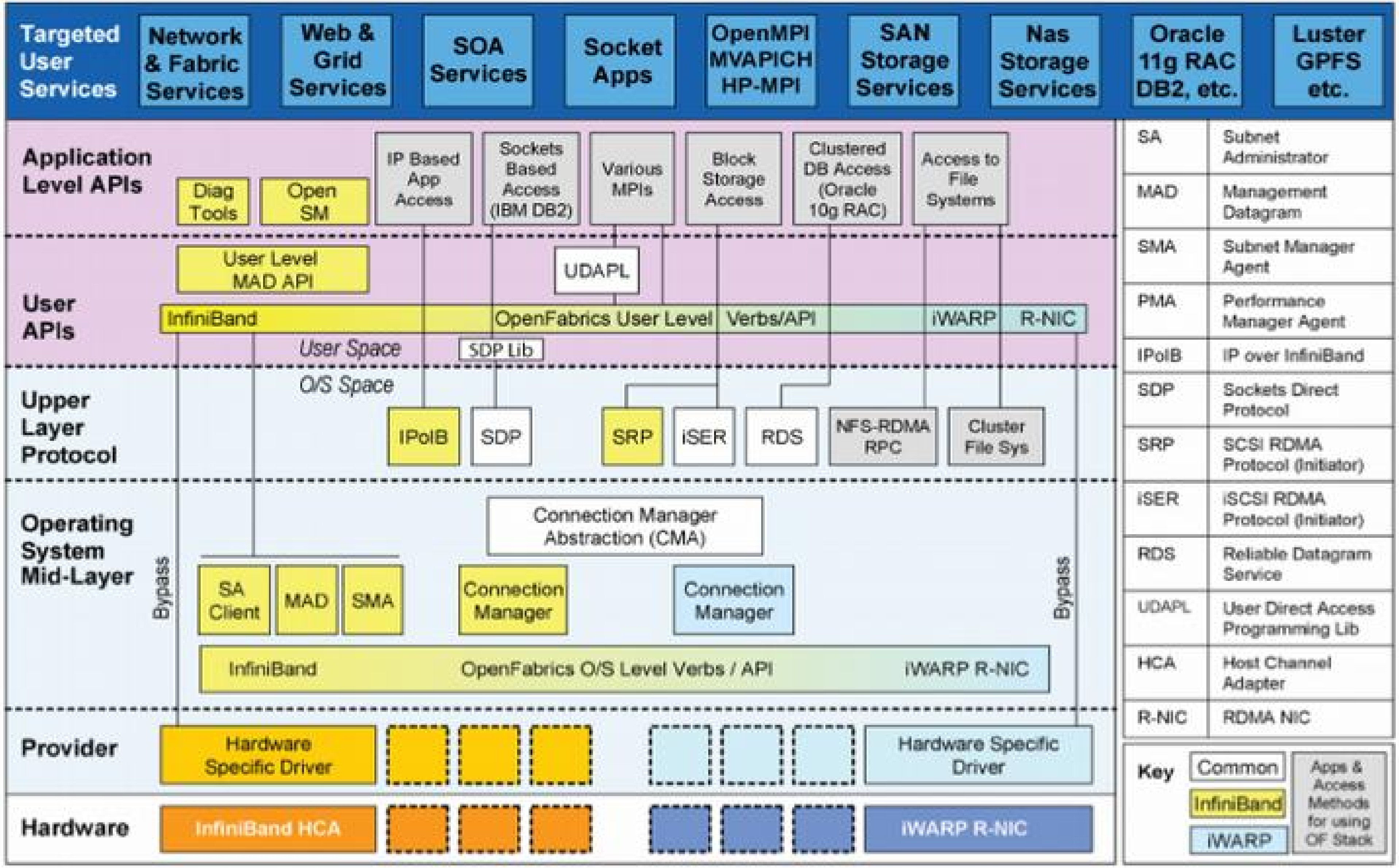
OFED stack includes:

- device drivers
- performance utilities
- diagnostic utilities
- protocols (IPoIB, SDP, SRP,...)
- MPI implementations (OpenMPI, MVAPICH)
- libraries
- subnet manager

Subnet Manager

- The Subnet Manager (SM) is mandatory for setting up port ID, links and routes
- OpenSM is an Infiniband compliant subnet manger included with OFED
- Ability to run several instance of osm on the cluster in a Master/Slave(s)configuration for redundancy.
- Routing is typically static:The subnet manager tries to balance the routes on the switches.
  - A sweep is done every 10 seconds to look for new ports or ports that are no longer present.
  - Established routes will typically remain in effect if possible.
  - Enhanced routing algorithms:
    - Min-hop, up-down, fat-tree, LASH, DOR, Torus2QOS

IB software stack..



SA	Subnet Administrator
MAD	Management Datagram
SMA	Subnet Manager Agent
PMA	Performance Manager Agent
IPoIB	IP over InfiniBand
SDP	Sockets Direct Protocol
SRP	SCSI RDMA Protocol (Initiator)
iSER	iSCSI RDMA Protocol (Initiator)
RDS	Reliable Datagram Service
UDAPL	User Direct Access Programming Lib
HCA	Host Channel Adapter
R-NIC	RDMA NIC

Why is (low) latency so important?

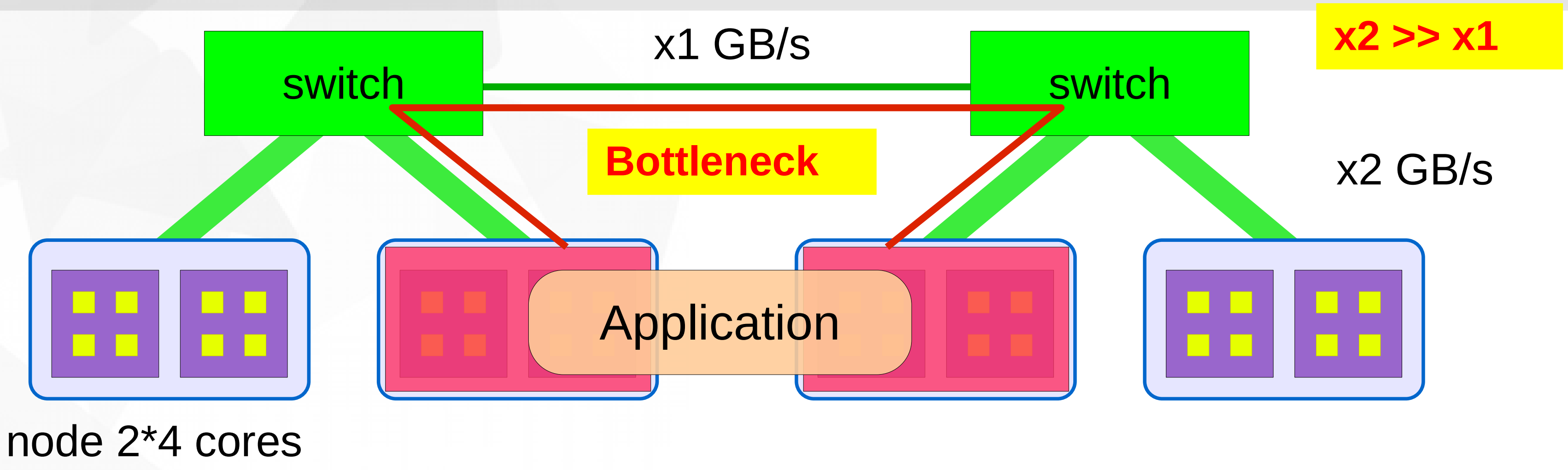
According to **Amdahl's law:**

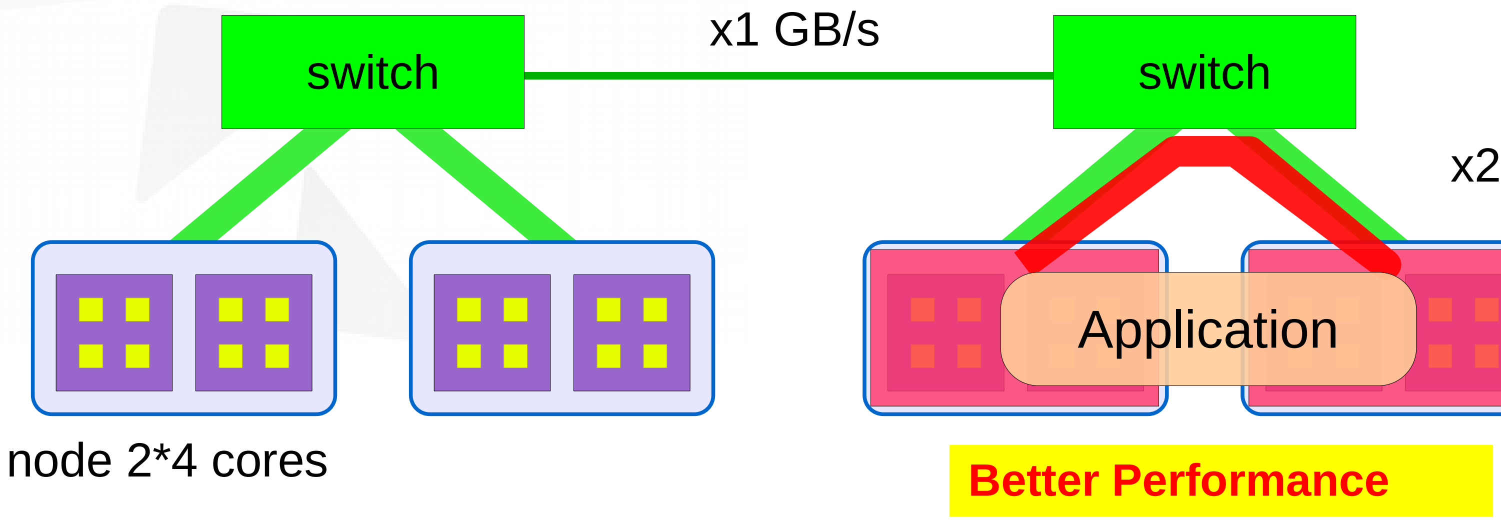
- **a high-performance parallel system tends to be bottlenecked by its slowest sequential process**
- in all but the most embarrassingly parallel supercomputer workloads, **the slowest sequential process is often the latency of message transmission across the network**

A few more considerations

- In general the compute/communication ratio in a parallel program remains fairly constant.
- So as the computational power increases the network speed must also be increased.
- As multi-core processors proliferate, it is increasingly common to have 8, 10 or even 16 MPI processes sharing the same network device.
- Contention for the interconnect device can have a significant impact on performance.

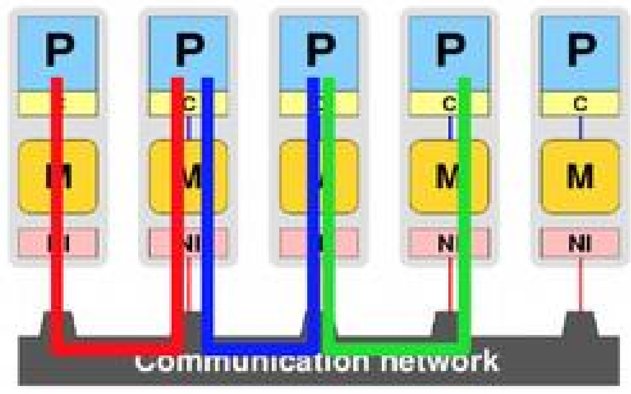
Topology-aware Scheduling

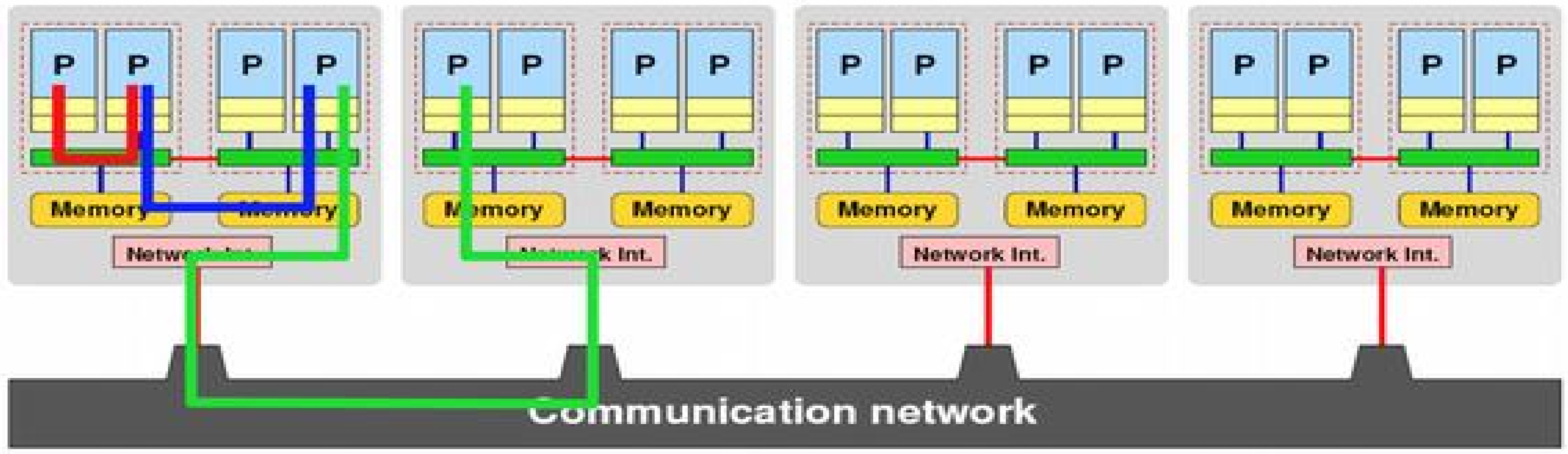




Parallel programming model : MPI

- Machine structure is invisible to user
  - Very simple programming model
  - MPI “knows what to do”!?
- Performance issues
  - Intranode vs. internode MPI
  - Node/system topology



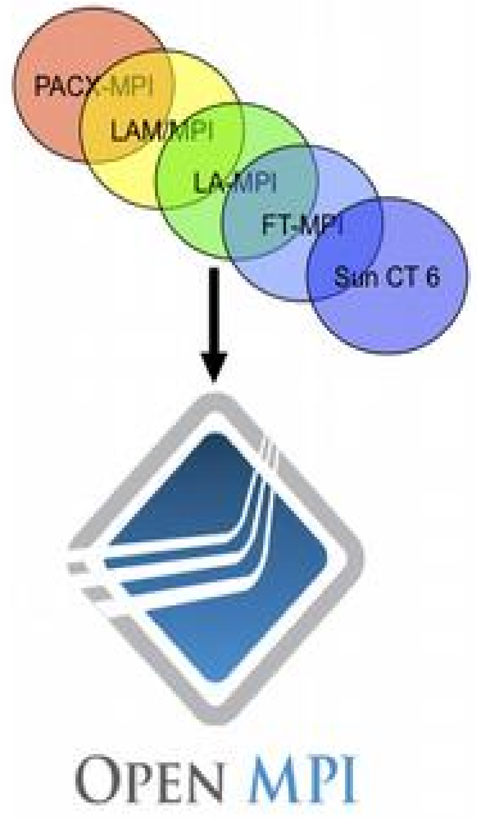


Parallel programming

- MPI is a standard with many implementations
- You need a library to link to your MPI-enable parallel code
- Many implementation available:
    - OpenMPI
    - MVAPICH
    - IntelMPI
    - MPICH
    - Etc..

openMPI

- Evolution of several prior MPI’s
- Open source project and community
- Production quality
- Vendor-friendly
- Research- and academic-friendly
- MPI-3.0 compliant



https://www.open-mpi.org/

OpeMPI and compilers

- OpenMPI works with several compiler suites

```
Module load openmpi/3.1.3..

openmpi/3.1.3/gcc/4.8.5-z2zfbgq
openmpi/3.1.3/gcc/8.2.0-qh4llbm
openmpi/3.1.3/pgi/18.10-ahjhvki
```

Mpirun/mpiexec

- Mpirun and mpiexec
  - Completely identical (in OpenMPI)
- General form:
  - `mpirun -np X your_exe`
  - `mpirun [-np X] --hostfile hostfile your_app`
- If using a scheduler,no need for hostfile or-np

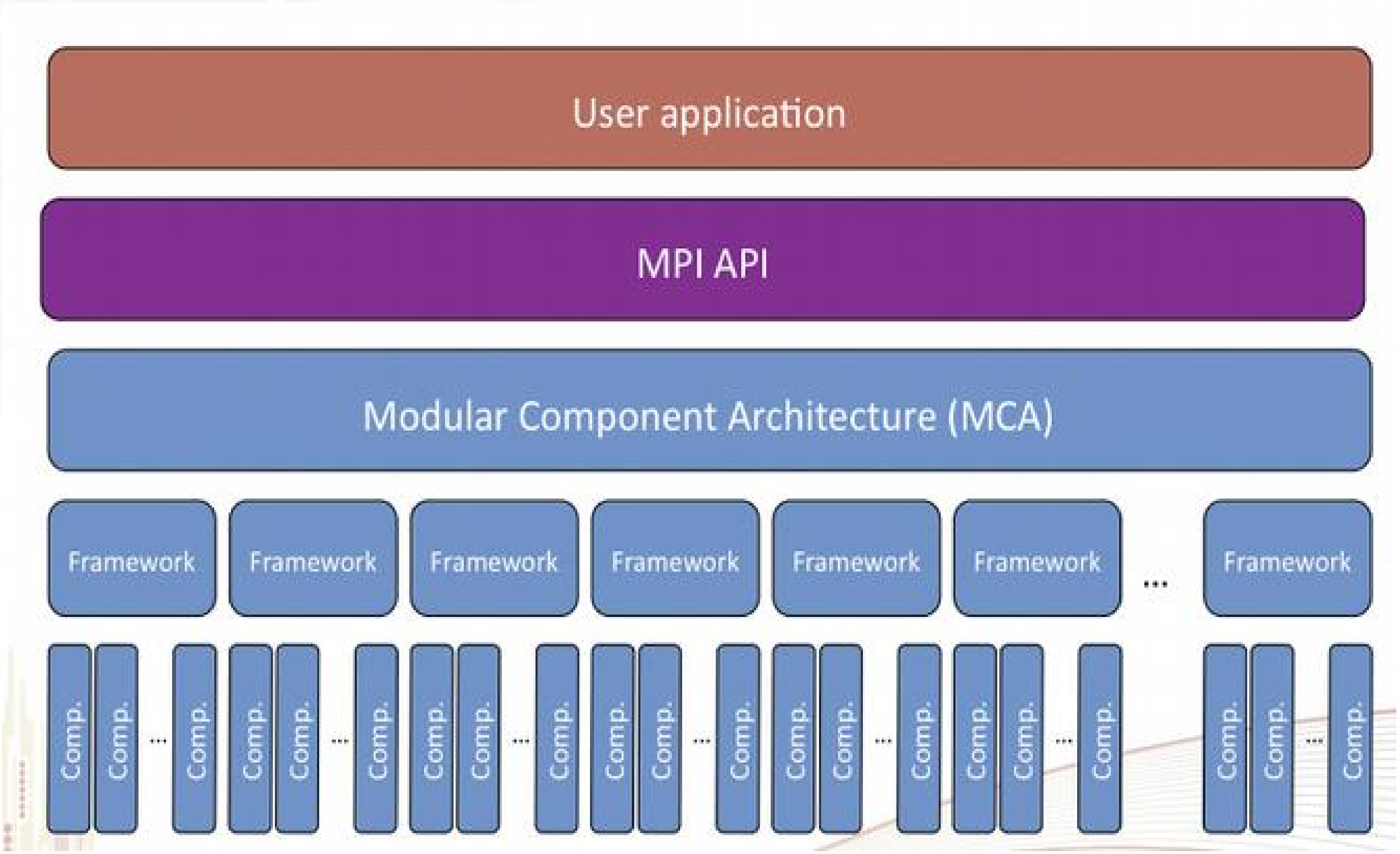
Mpirun useful options

- Assign only a certain number of MPI process on one node
  - `-npernode   X`
- Indicates how many cores to bind per process
  - --cpus-per-proc <#perproc>
- Show how processes are bind to cores/sockets etc..
  - --report-bindings

OpenMPI is Based on Plugins

- Lots and lots of plugin types
  - Back-end network
  - Resource manager support
  - Operating system support
- All can be loaded (or not) at runtime
  - Choice of network is a runtime decision

Plugin high level view



MCA parameters

- Run-time tunable values
  - Per layer
  - Per framework
  - Per component(“plugin”)
- Change behaviors of code at run-time
  - Does not require recompiling/relinking
-

Example: specify BTL

- BTL:Byte Transfer Layer
  - Framework for MPI point-to-pointcommunications
  - Select which network to use for MPI communications

```
mpirun --mca btl tcp,self -np 4 my app
```

- Components
  - tcp:TCP sockets
  - self:Loopback (send–to–self)

Example:specify openIB BTL

```
mpirun --mca btl openib,self -np 4 my
app
```

- Components
  - openib:OpenFabricsverbs(InfiniBand)
  - self:Loopback(send-to-self)

What does this do ?

```
mpirun  -np 4 my app
```

- Use all available components
  - tcp,sm,openib,…
- TCP too?
  - Yes and no..
  - TCP is automatically disable itself in the presence of better network/protocol

What does this do ?

```
   mpirun  -np 4 my app
```

- More specifically:
  - Open each BTL component
  - Query if it wants to be used
  - Keep all that say“yes”

    Rank by bandwidth and latency rank

you can check with `--verbose` option

What does this do ?

```
mpirun  -np 4  --mca btl ^tcp my app
```

- Use all available components except tcp
- More specifically:
  - Open eachBTL component except tcp
  - Query if it wants to be used
  - Keep all that say“yes”

    Rank by bandwidth and latency rank

MPI freely available benchmarks (2)

- IMB-4.0 (now IMB2017) (INTEL MPI benchmark)
  - MPI protocol ()
  - https://software.intel.com/en-us/articles/intel-mpi-benchmarks
- OSU benchmarks: http://mvapich.cse.ohio-state.edu/benchmarks/

Suggested activities

- Play with Intel MPI benchmark
- Compile it using openMPI with different compiler
- Submit your job using two or more nodes
- Play with different BTL
- Report/understand difference