Technical University of Sofia

Computer Systems Department

BlueGene Supercomputer Architecture

Assist. Prof. Desislava Ivanova

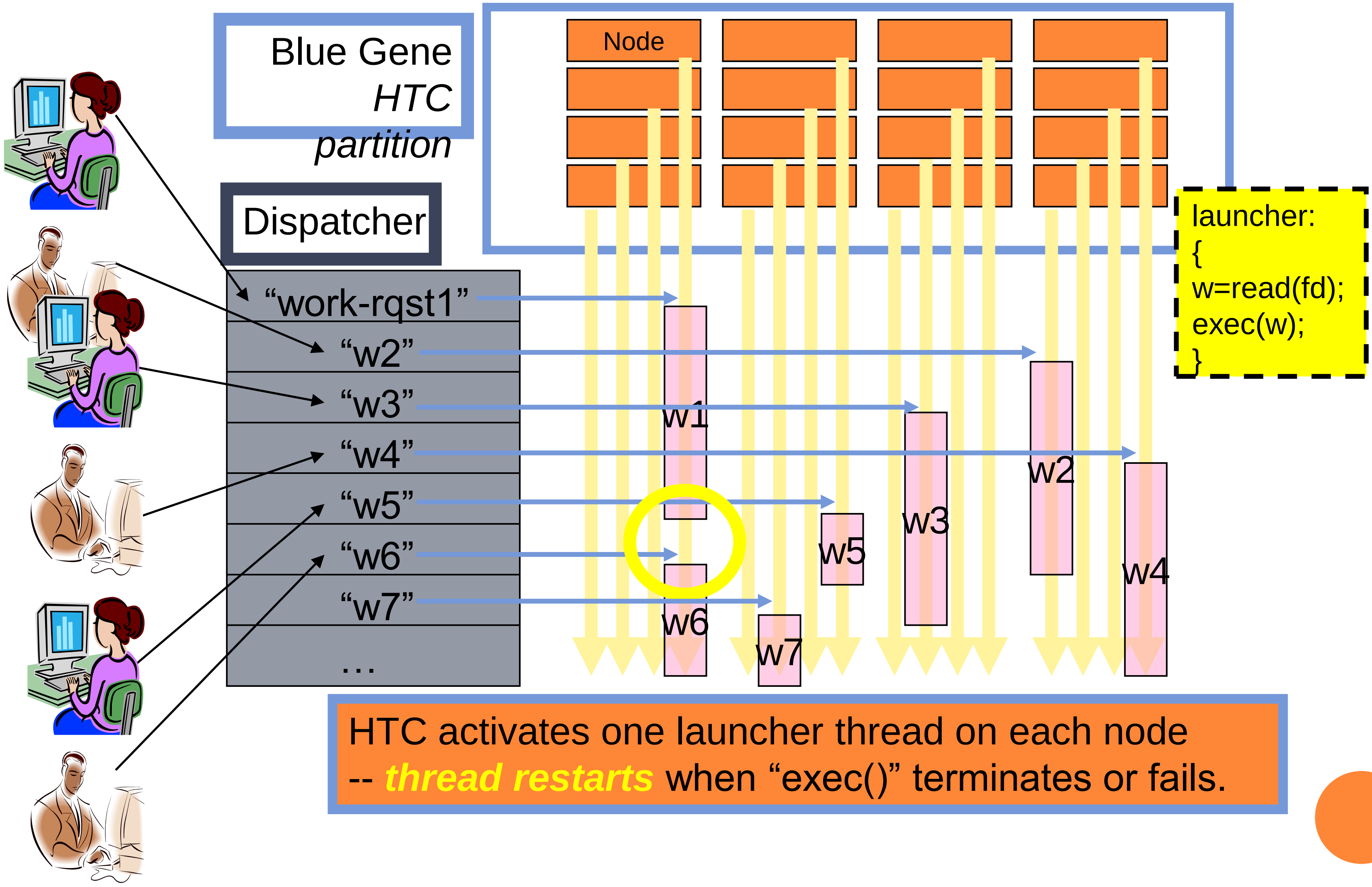
HPC vs. HTC Comparison

- **High Performance Computing (HPC) Model**
  - Parallel, tightly coupled applications
    - Single Instruction, Multiple Data (SIMD) architecture
  - Programming model: typically MPI
  - Apps need tremendous amount of computational power over short time period
- **High Throughput Computing (HTC) Model**
  - Large number of independent tasks
    - Multiple Instruction, Multiple Data (MIMD) architecture
  - Programming model: non-MPI
  - Apps need large amount of computational power over long time period
  - Traditionally run on large clusters
- **HTC and HPC modes co-exist on Blue Gene**
  - Determined when resource pool (partition) is allocated

Blue Gene for HTC Motivation

- **High processing capacity with minimal floor space**
  - High compute node density – 2,048 processors in one Blue Gene rack
  - Scalability from 1 to 64 racks (2,048 to 131,072 processors)
- **Resource consolidation**
  - Multiple HTC and HPC workloads on a single system
  - Optimal use of compute resources
- **Low power consumption**
  - **#1 on Green500 list @ 112 MFlops/Watt (www.green500.org/CurrentLists.html)**
  - Twice the performance per watt of a high frequency microprocessor
- **Low cooling requirements enable extreme scale-up**
- **Centralized system management**
  - Blue Gene Navigator

Generic HTC Flow on BlueGene



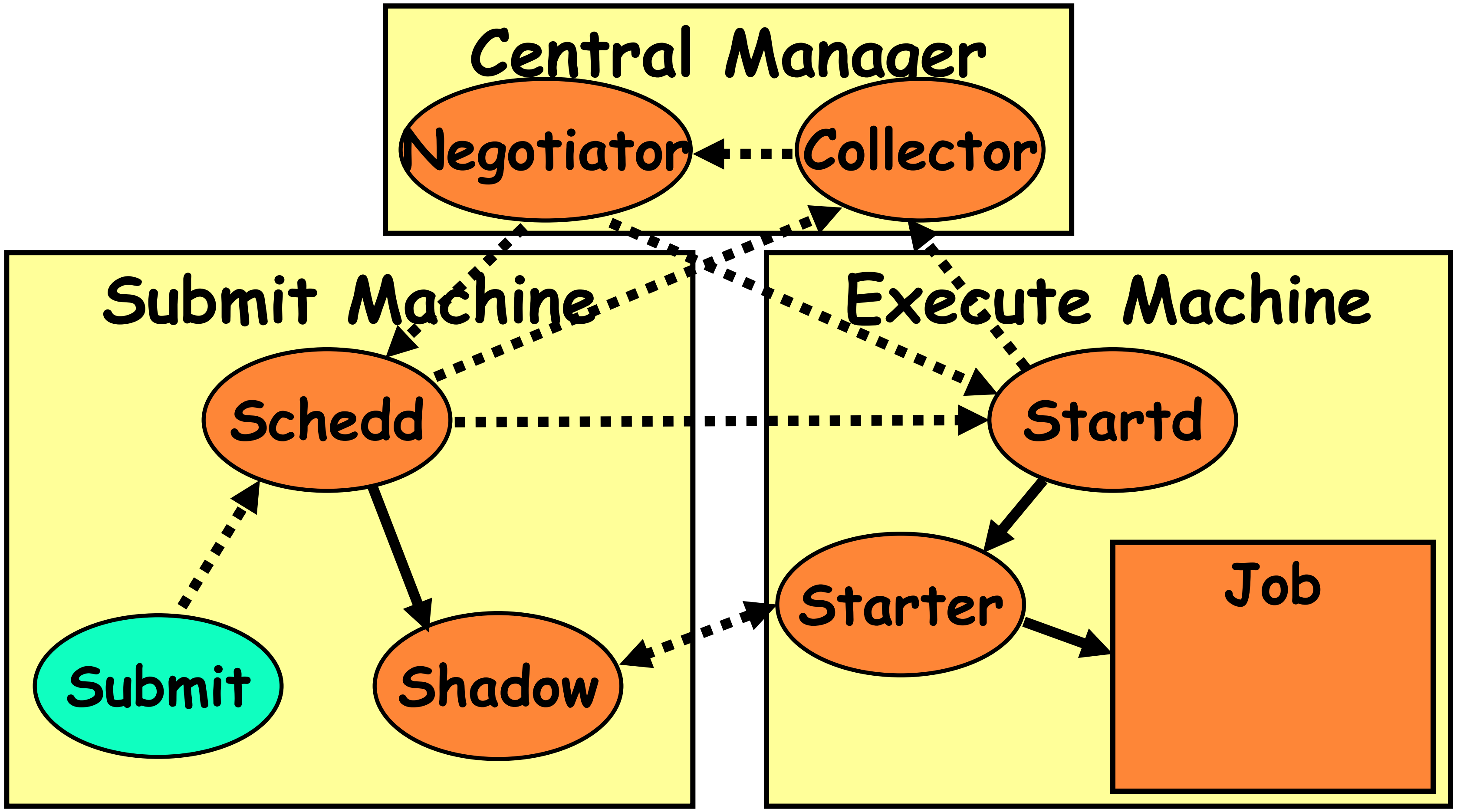
```
launcher:
{
w=read(fd);
exec(w);
}
```

HTC activates one launcher thread on each node
-- ***thread restarts*** when "exec()" terminates or fails.

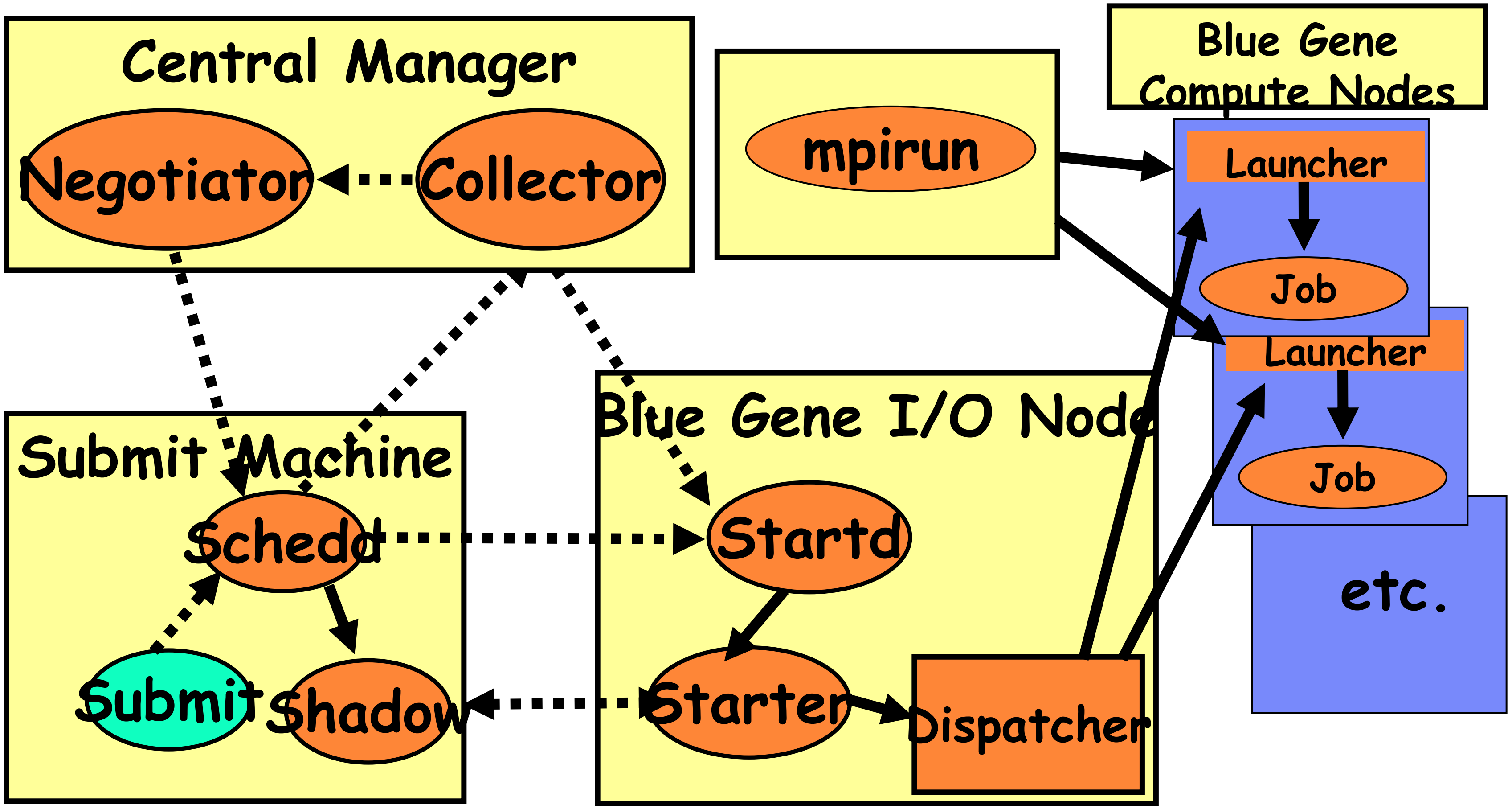
Condor and IBM Blue Gene Collaboration

- **Both IBM and Condor teams engaged in adapting code to bring Condor and Blue Gene technologies together;**
- **Initial Collaboration (Blue Gene/L)**
  - Prototype/research Condor running HTC workloads on Blue Gene/L
    - Condor developed dispatcher/launcher running HTC jobs
    - Prototype work for Condor being performed on Rochester On-Demand Center Blue Gene system
- **Mid-term Collaboration (Blue Gene/L)**
  - Condor supports HPC workloads along with HTC workloads on Blue Gene/L
- **Long-term Collaboration (Next Generation Blue Gene)**
  - I/O Node exploitation with Condor
  - Partner in design of HTC services for Next Generation Blue Gene
    - Standardized launcher, boot/allocation services, job submission/tracking via database, etc.
  - Study ways to automatically switch between HTC/HPC workloads on a partition
  - Data persistence (persisting data in memory across executables)
    - Data affinity scheduling
  - Petascale environment issues

Condor Architecture



CONDOR WITH BLUE GENE/L



IBM's Blue Gene/P Supercomputer

- **Applying Innovation that Matters to High Performance Computing**
- – Ultrascale performance
- – High reliability
- – Easy manageability
- – Ease of programming, familiar open/standard operating Environment
- – Simple porting of parallel codes
- – Space saving design
- – Low power requirements
- – Limited performance and memory per core

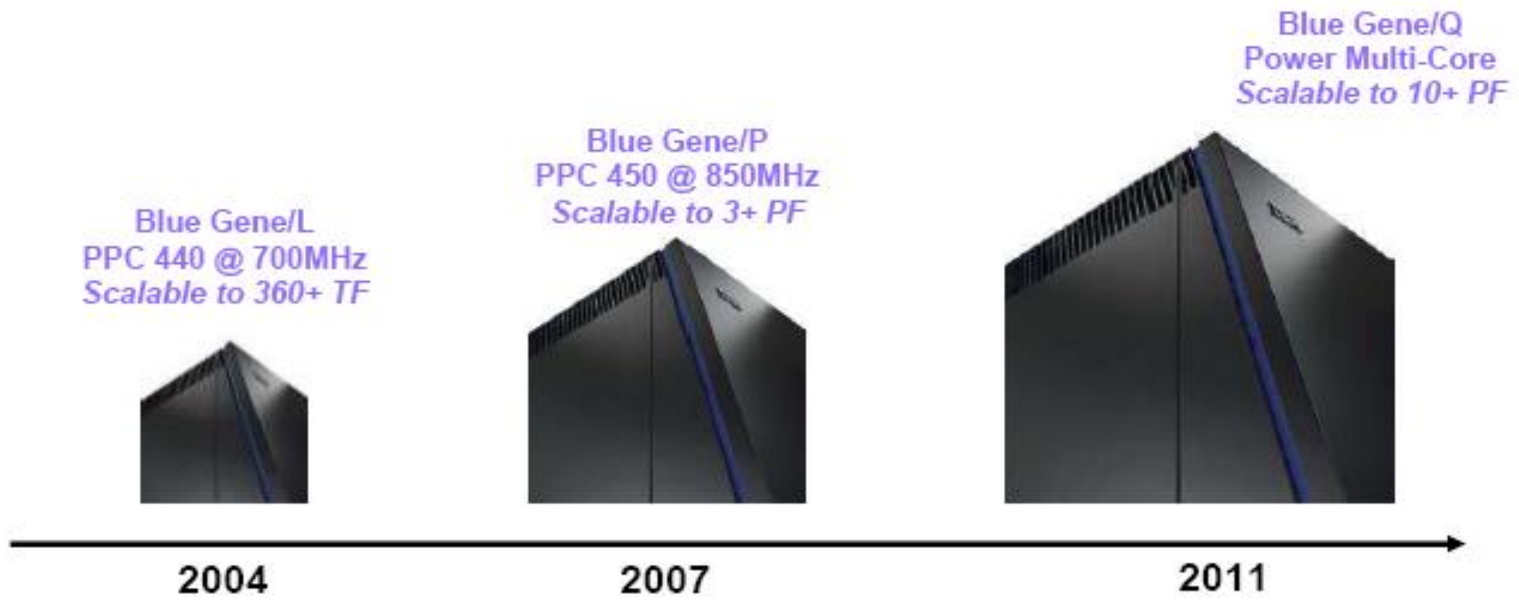
What is Blue Gene?

- **Massively Parallel Architecture**

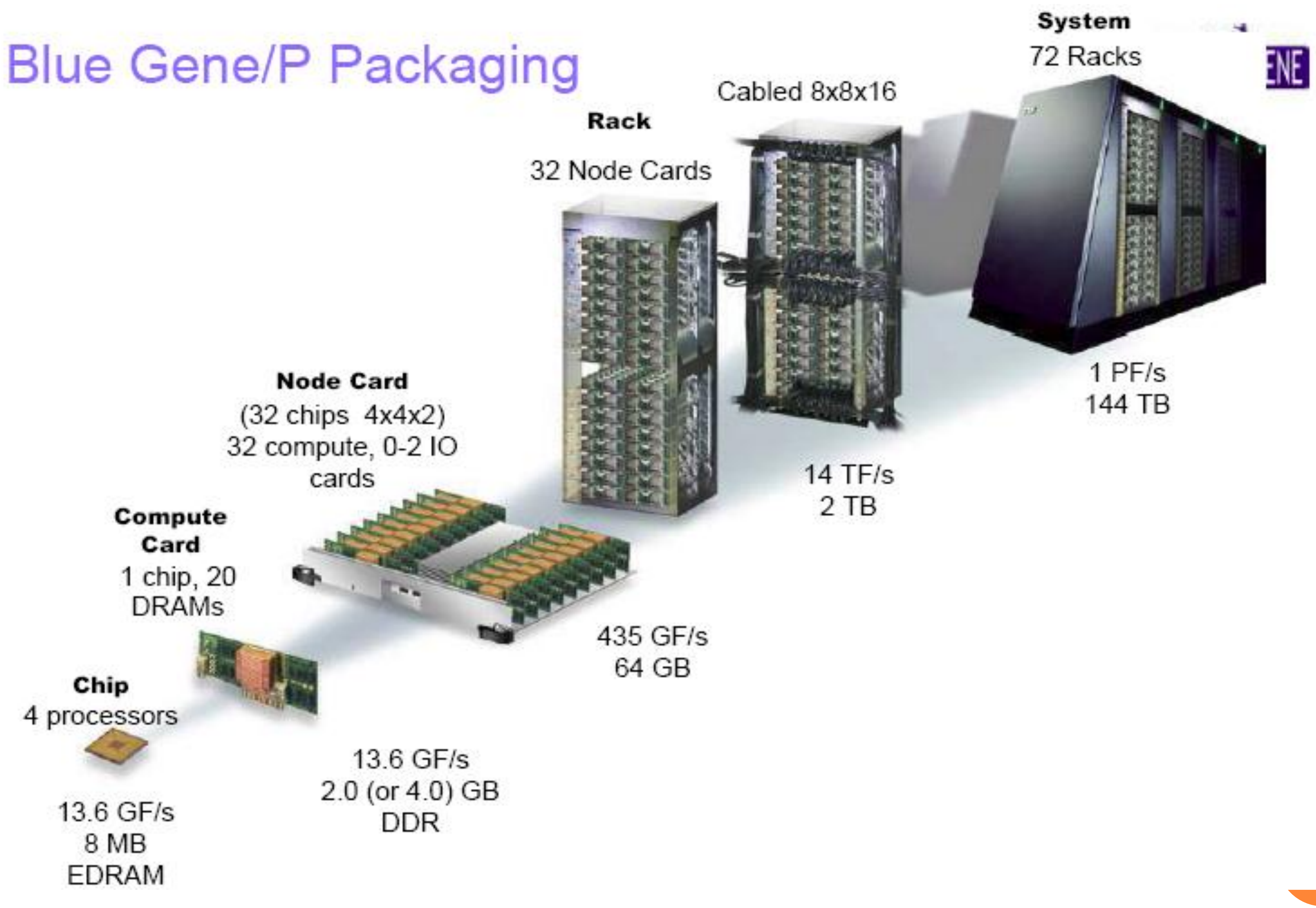
– A computer system with many independent arithmetic units or entire microprocessors, that are connected together to be one very large computer

– Opposed to Clusters / Shared Memory Processing

- **Two solutions available by now, one in the roadmap**

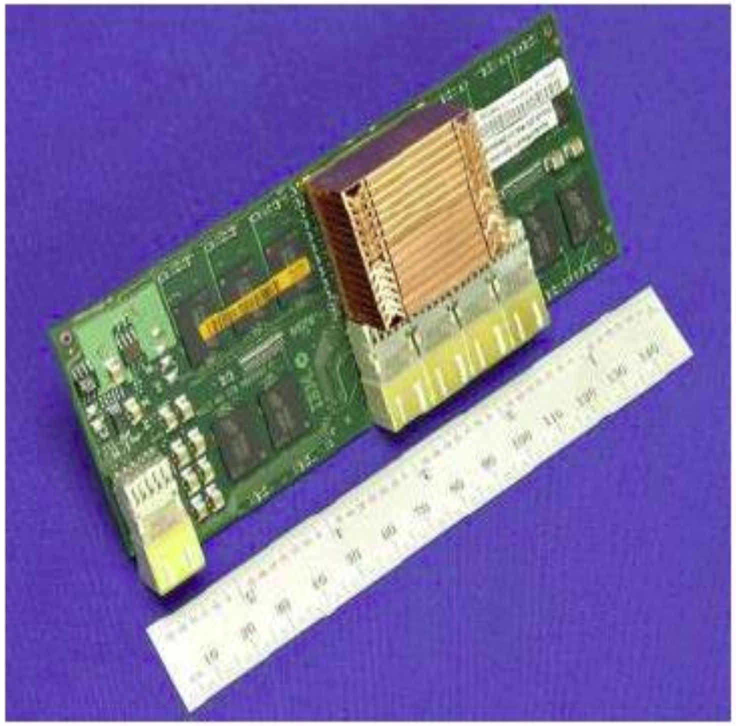


Blue Gene/P Packaging



Blue Gene/P Compute Card

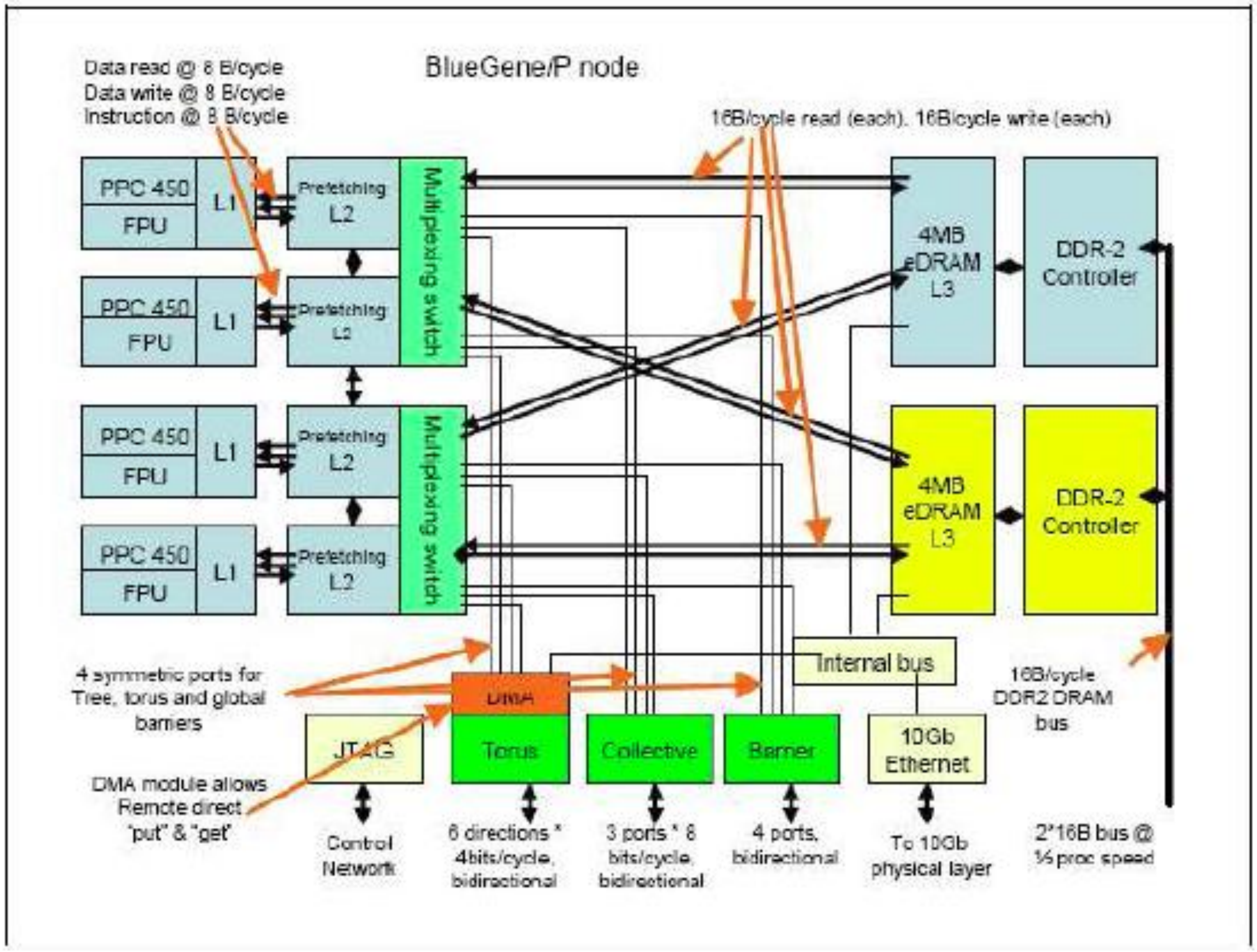
Blue Gene/P Compute Card



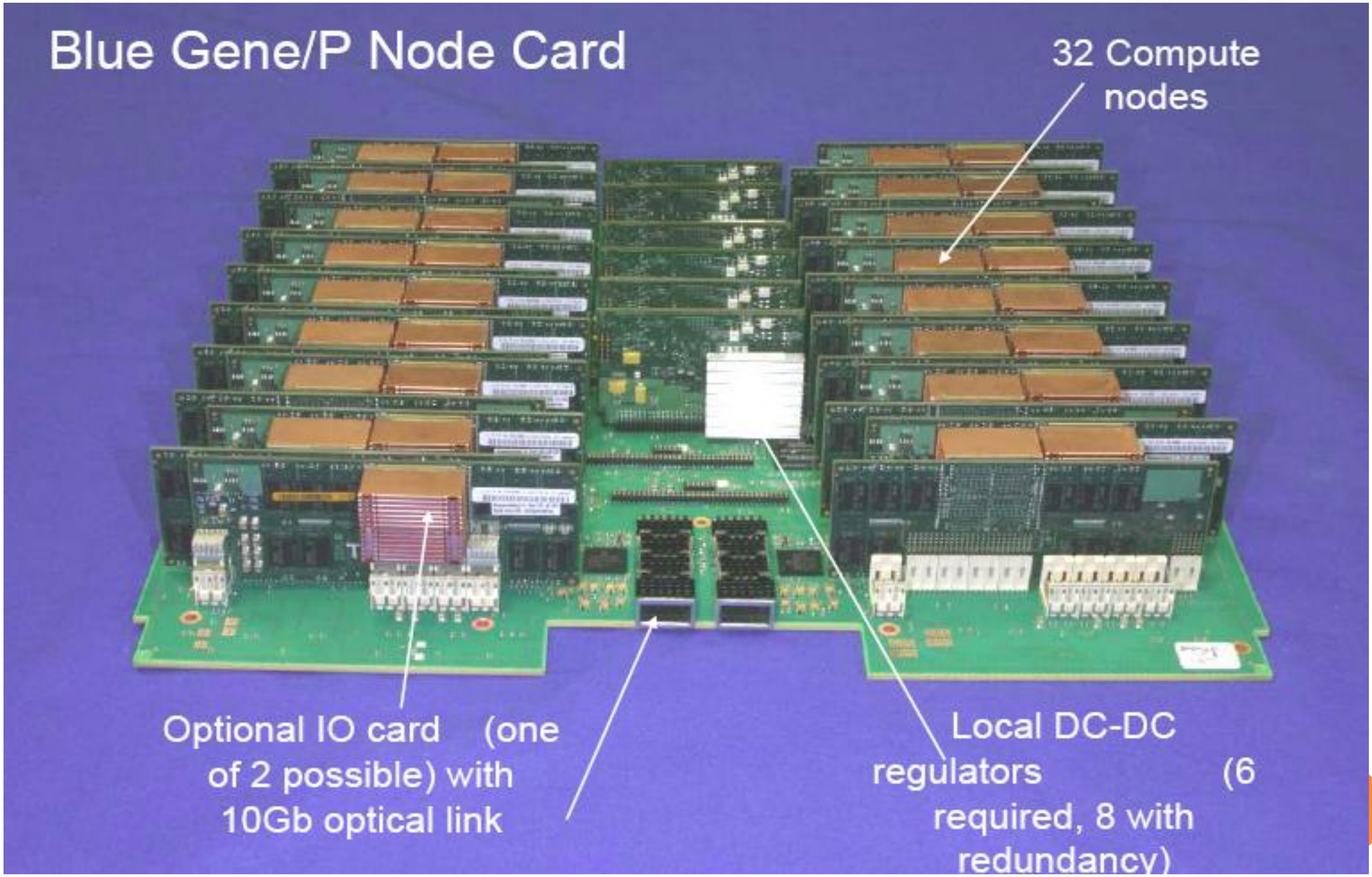
- System-on-a-Chip (SoC)
- PowerPC 450 CPU
  - 850 MHz Frequency
  - Quad Core
- 2 GB RAM
- Network Connections

Blue Gene/P ASIC

Blue Gene/P ASIC



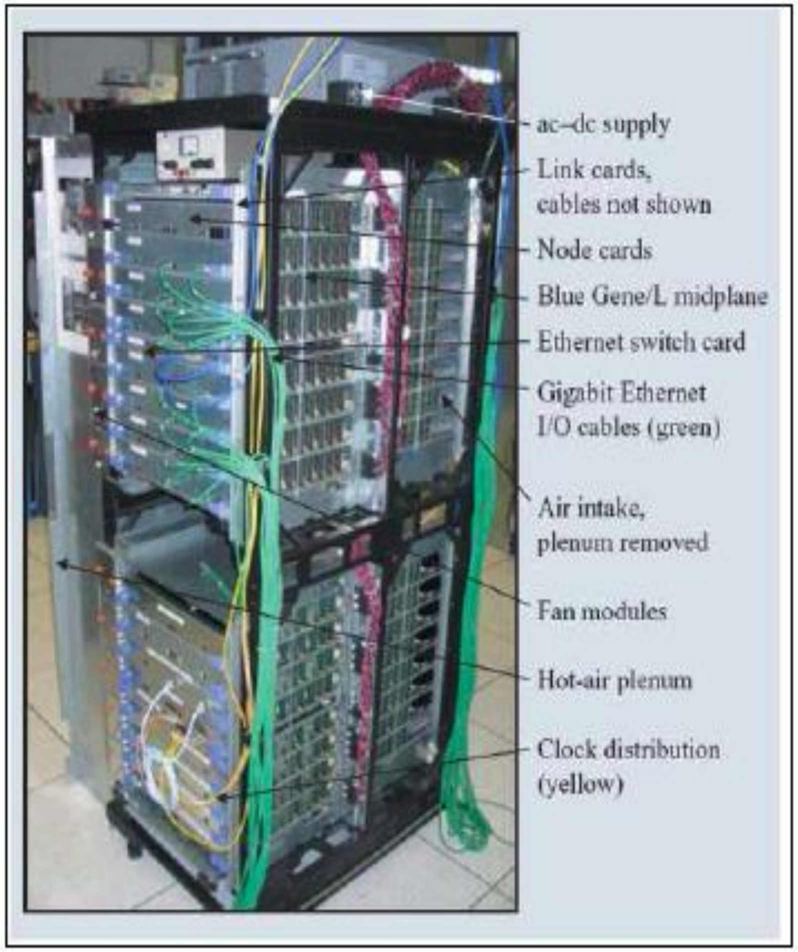
Blue Gene/P Node Card


Blue Gene/P Node Card
32 Compute nodes
Optional IO card (one of 2 possible) with 10Gb optical link
Local DC-DC regulators (6 required, 8 with redundancy)

Blue Gene/P Rack Content

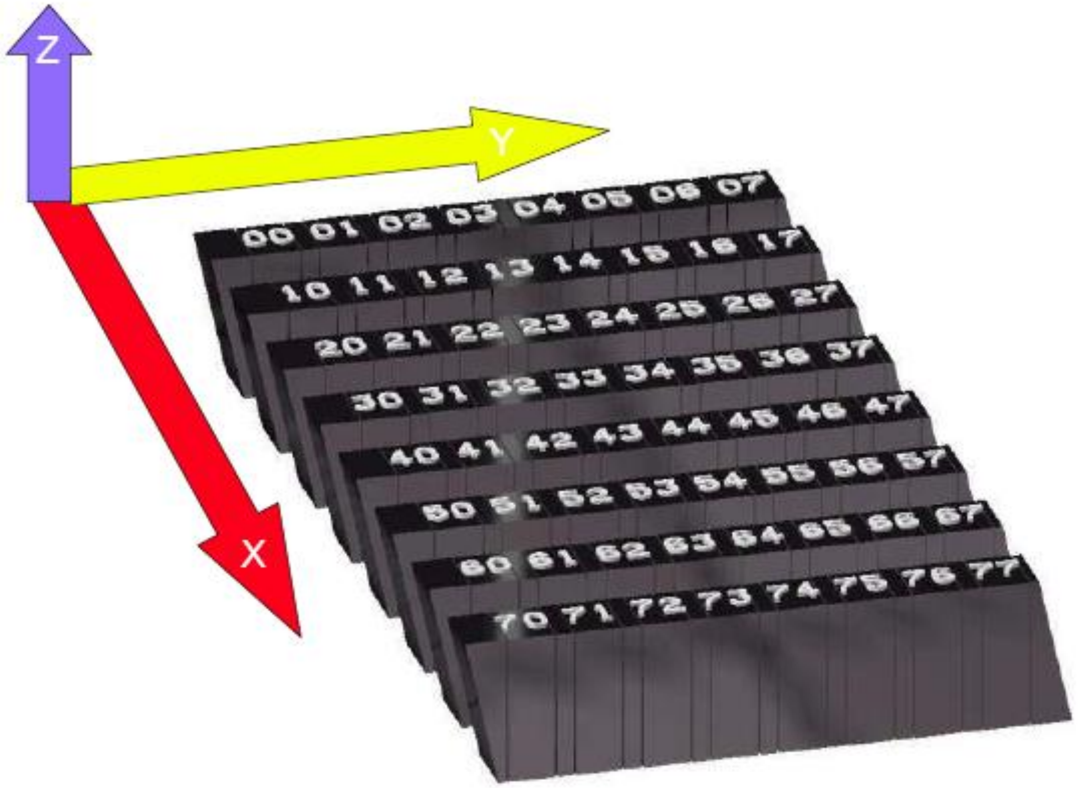
Blue Gene/P Rack Content

- **32 Compute Nodes populate a Node Card**
  - Node cards may be hot plugged for service
- **0-2 I/O Nodes populated on a Node Card**
  - Flexible ratio of compute to I/O nodes
  - I/O Nodes are identical to Compute Nodes other than placement in the Node Card which defines network connections
- **16 Node Cards form a Midplane**
- **2 Midplanes form a rack**
  - 1024 Compute Nodes per rack
  - 8 to 64 I/O nodes per rack: 80Gb to 640Gb Ethernet bw/rack

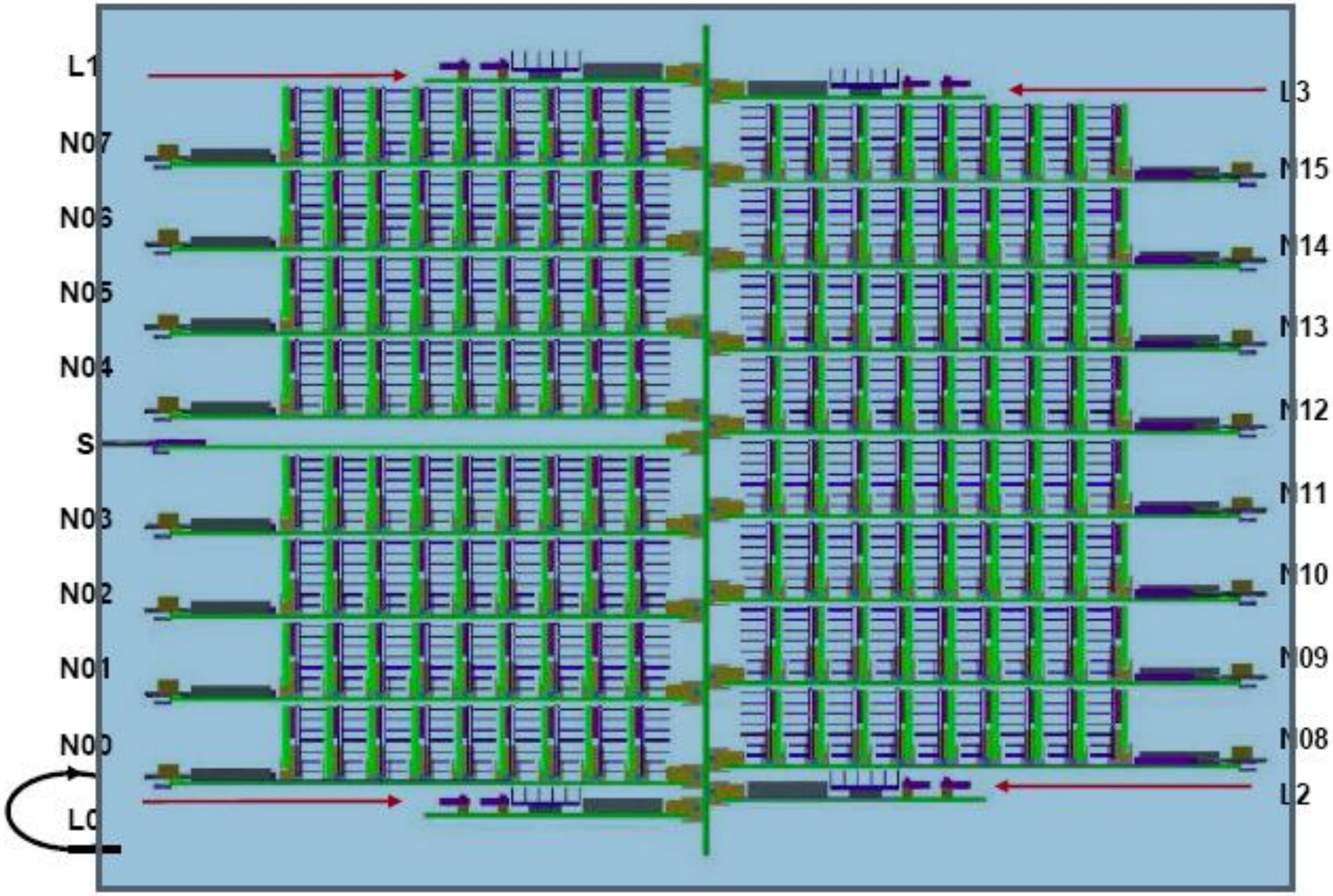

ac–dc supply
Link cards, cables not shown
Node cards
Blue Gene/L midplane
Ethernet switch card
Gigabit Ethernet I/O cables (green)
Air intake, plenum removed
Fan modules
Hot-air plenum
Clock distribution (yellow)

Torus X-Y-Z

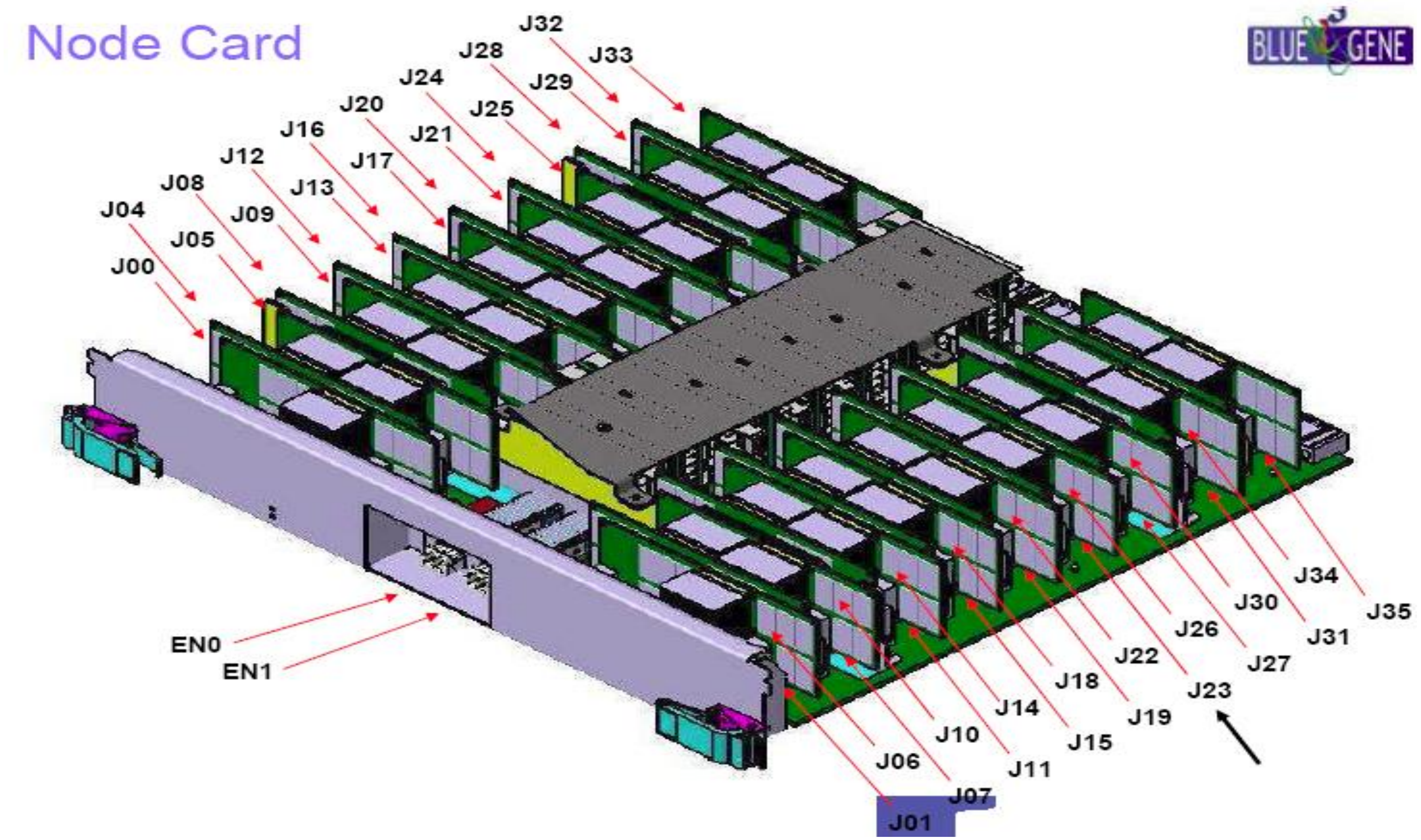
Torus X-Y-Z



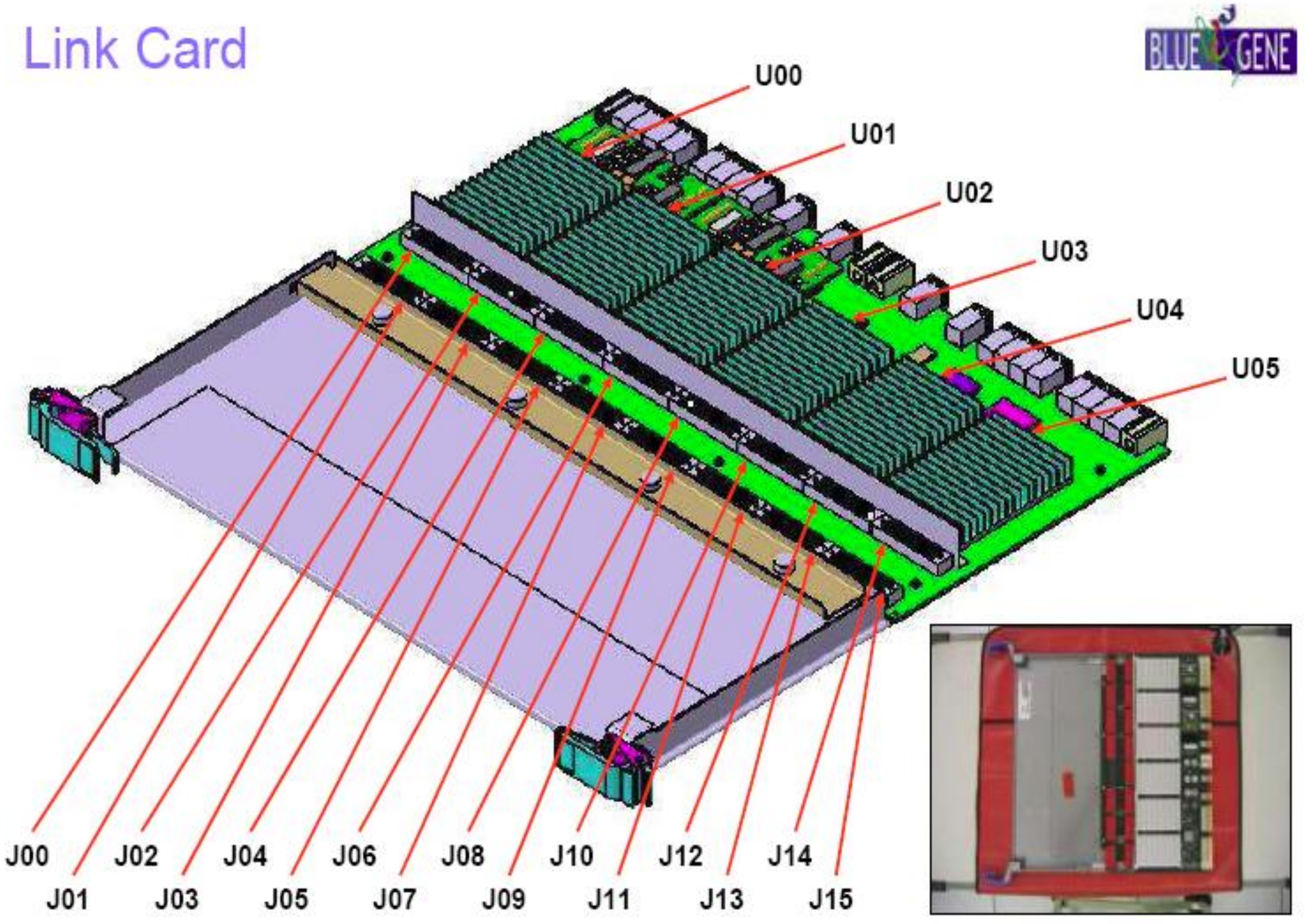
Node, Link, Service Card Names



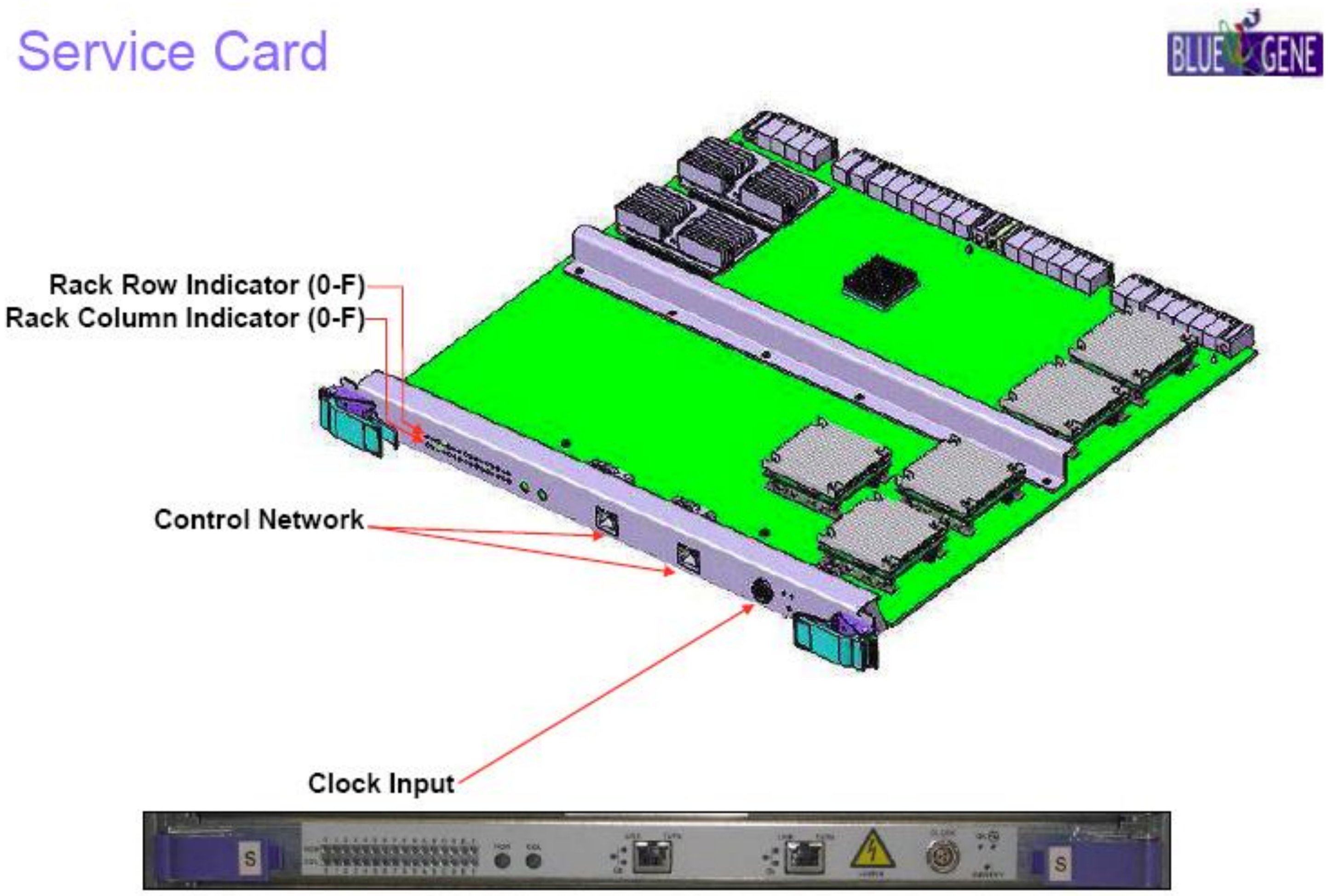
Node Card


Node Card
BLUE GENE
J32
J28
J33
J24
J29
J20
J25
J16
J21
J12
J17
J08
J13
J04
J09
J05
J00
EN0
EN1
J34
J30
J35
J26
J31
J22
J27
J18
J23
J14
J19
J10
J15
J06
J11
J07
J01

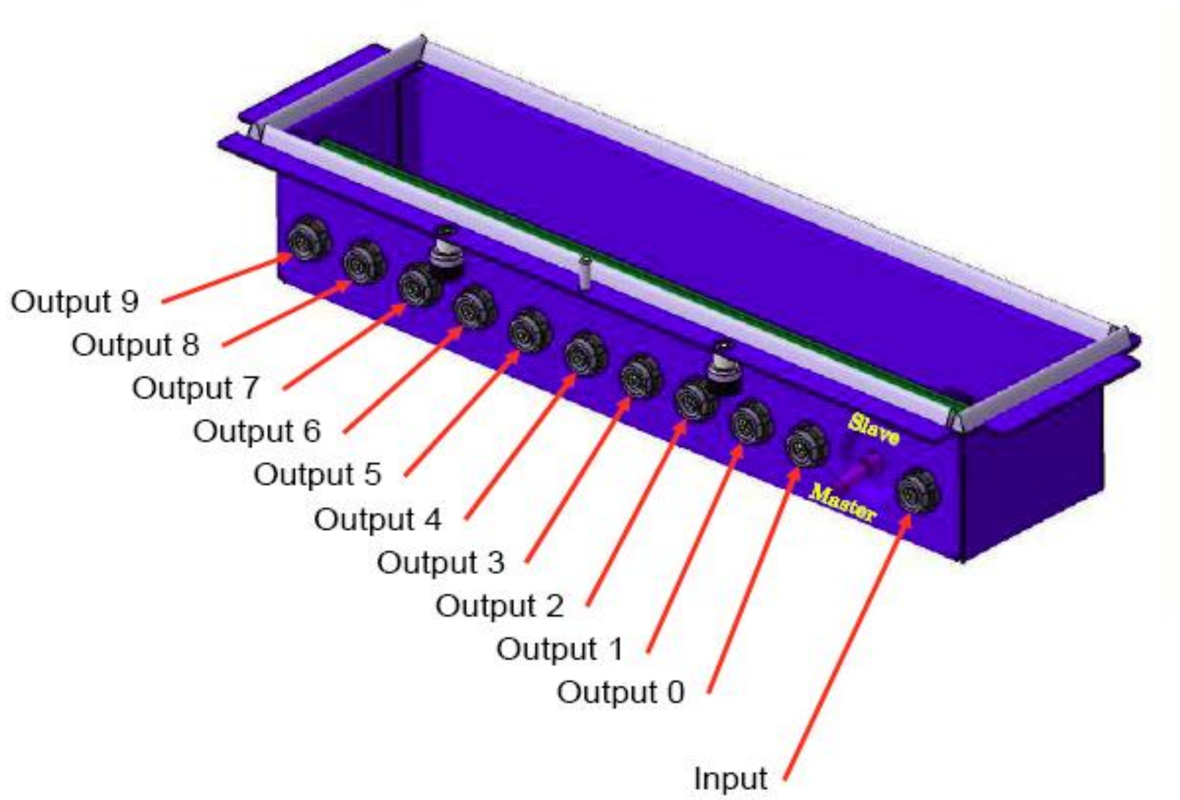
Link Card


Link Card
BLUE GENE
U00
U01
U02
U03
U04
U05
J00
J01
J02
J03
J04
J05
J06
J07
J08
J09
J10
J11
J12
J13
J14
J15

Service Card



Clock Card


Output 9
Output 8
Output 7
Output 6
Output 5
Output 4
Output 3
Output 2
Output 1
Output 0
Input
Slave
Master

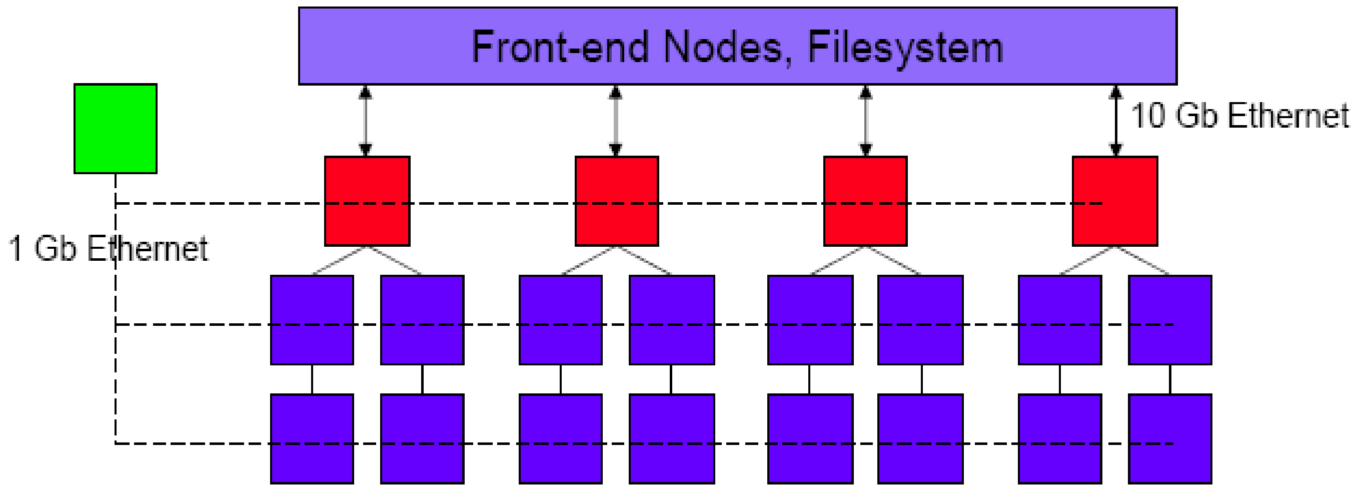
Blue Gene/P System Components

Blue Gene/P System Components

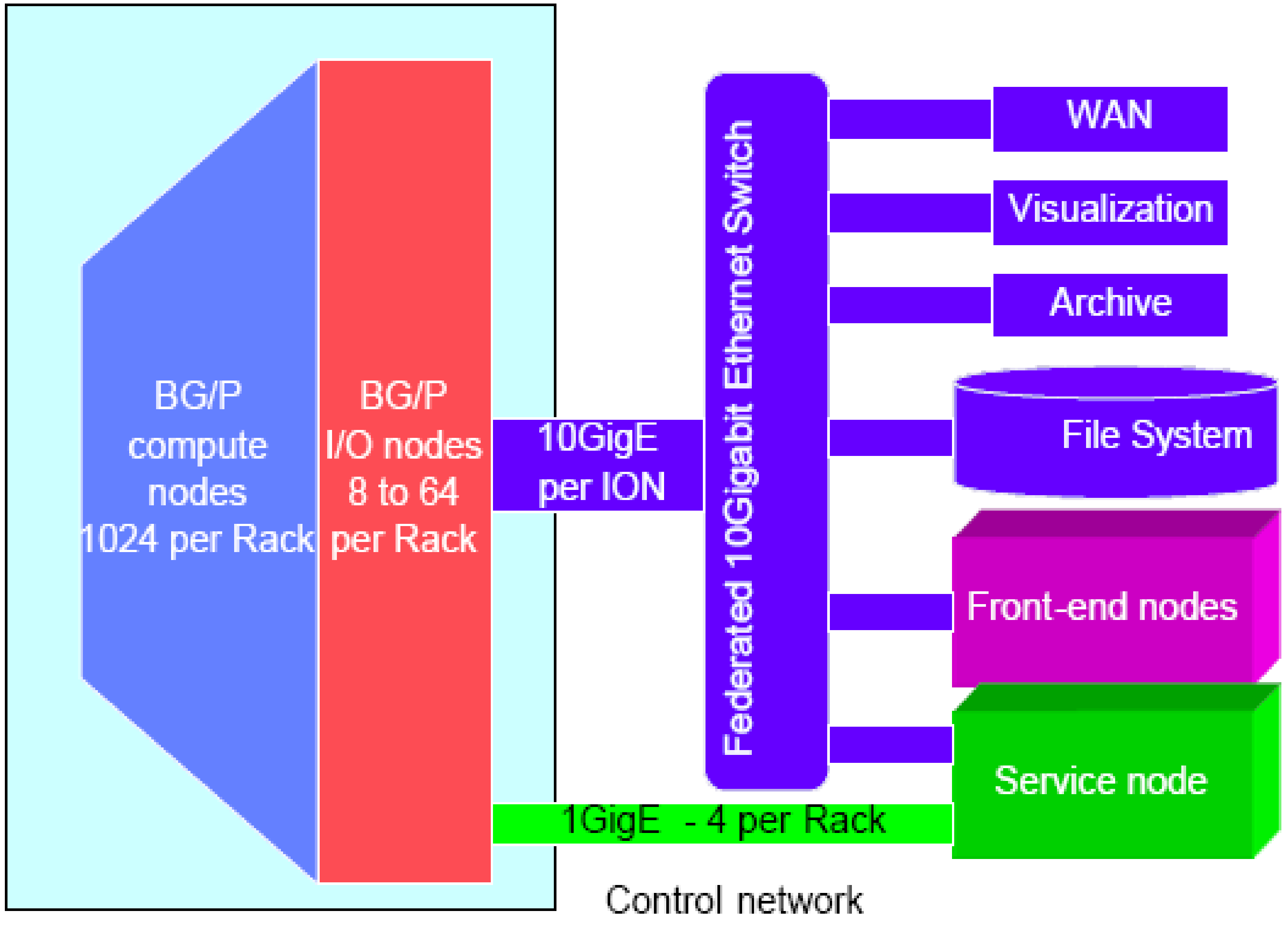
- **Blue Gene/P Rack(s)**
  - 1024 Quad-Core Compute Nodes per Rack > 13.7 TF/sec peak
  - 2 GB main memory per Node > 2 TB main memory per Rack
  - Up to 64 I/O Nodes per Rack > 80 GB/sec peak
  - Scalable to 256 racks > 3.5 PF/sec
- **Host System**
  - Service and Front End Nodes: System p
  - Software Stack: SLES10, DB2, XLF/C Compilers, HPC Cluster SW
  - Ethernet Switches: Force10, Myrinet
  - Storage System: IBM, DDN

Blue Gene Blocks Hierarchical Organization

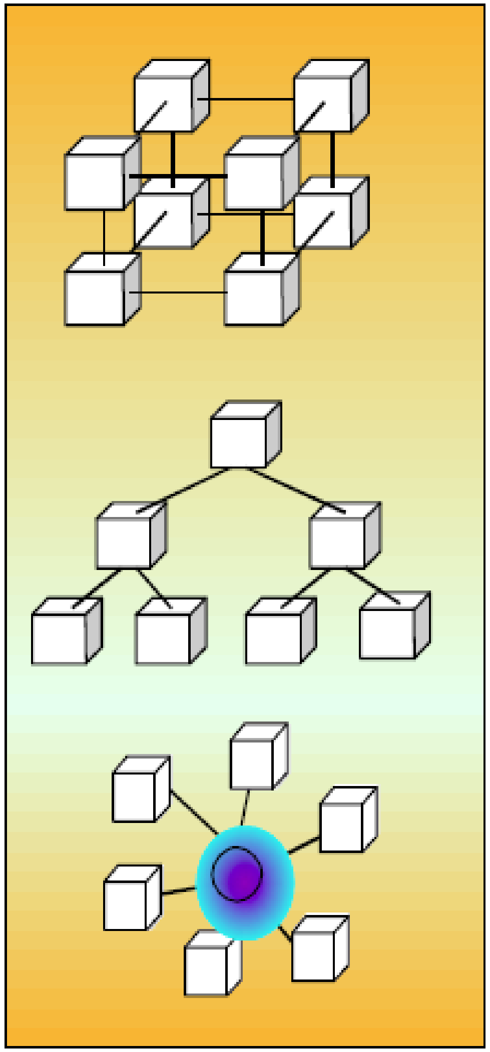
- Compute Nodes dedicated to running user application, and almost nothing else - simple compute node kernel (CNK)
- I/O Nodes run Linux and provide a more complete range of OS services – files, sockets, process launch, signaling, debugging, and termination
- Service Node performs system management services (e.g., partitioning, heart beating, monitoring errors) - transparent to application software



Blue Gene/P Full Configuration



Blue Gene/P Interconnection Networks



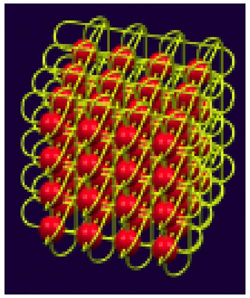
- **3-Dimensional Torus**
  - Interconnects all compute nodes
  - Virtual cut-through hardware routing
  - 3.4 Gb/s on all 12 node links (5.1 GB/s per node)
  - 0.5 µs latency between nearest neighbors, 5 µs to the farthest
  - MPI: 3 µs latency for one hop, 10 µs to the farthest
  - Communications backbone for computations
  - 1.7/3.9 TB/s bisection bandwidth, 188TB/s total bandwidth
- **Collective Network**
  - One-to-all broadcast functionality
  - Reduction operations functionality
  - 6.8 Gb/s of bandwidth per link
  - Latency of one way tree traversal 1.3 µs, MPI 5 µs
  - ~62TB/s total binary tree bandwidth (72k machine)
  - Interconnects all compute and I/O nodes (1152)
- **Low Latency Global Barrier and Interrupt**
  - Latency of one way to reach all 72K nodes 0.65 µs (MPI 1.6 µs)
- **Other networks**
  - 10Gb Functional Ethernet
  - I/O nodes only
  - 1Gb Private Control Ethernet
  - Provides JTAG access to hardware. Accessible only from Service Node system

Blue Gene Networks

- Torus
  - Compute nodes only
  - Direct access by app
  - **DMA**
- Collective
  - Compute and I/O node attached
  - 16 routes allow multiple network configurations to be formed
  - Contains an ALU for collective operation offload
  - Direct access by app
- Barrier
  - Compute and I/O nodes
  - Low latency barrier across system (< 1usec for 72 rack)
  - Used to synchronize timebases
  - Direct access by app
- 10Gb Functional Ethernet
  - I/O nodes only
- 1Gb Private Control Ethernet
  - Provides JTAG, i2c, etc, access to hardware. Accessible only from Service Node system
- Clock network
  - Single clock source for all racks

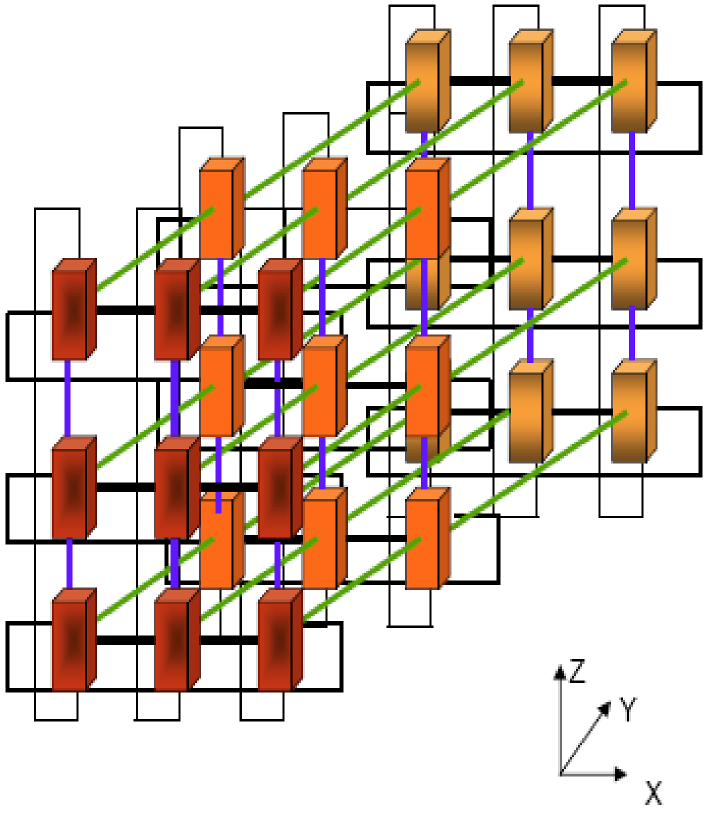
Three-Dimensional Torus Network: Point-to-Point

- **The torus network is used for general-purpose, point-to-point message passing and multicast operations to a selected "class" of nodes**
- **The topology is a three-dimensional torus constructed with point-to-point, serial links between routers that are embedded within the Blue Gene/P ASICs.**
- **Torus direct memory access (DMA), which provides memory access for reading, writing,or doing both independently of the processing unit**
- **Each CNodes (ASIC) has six nearest-neighbor connections, some of which can traverse relatively long cables.**
  - All the partitions do not have a Torus shape (only multiple midplans partition (x 512 nodes); others have a mesh shape, i.e. the opposite sides of the 3D cube are not linked)
- **The target hardware bandwidth for each torus link is 425 MBps in each direction of the link for a total of 5.1 GBps bidirectional bandwidth per node.**
  - Interconnection of all Compute Nodes (73,728 for a 72 rack system)
  - Virtual cut-through hardware routing
  - 3.4 Gbps on all 12 node links (5.1 GBps per node)
  - 1.7/3.8 TBps bisection bandwidth, 67 TBps total bandwidth



Torus VS Mesh

- The basic block is the midplane, shape 8x8x8= 512 Computes Nodes (2048 cores)
- Only multiple midplanes partition is a Torus; i.e. each CNode has 6 nearest-neighbours
- All the other partition is a mesh
- Capability from LoadLeveler to request a Torus or a Mesh with the field:
  - # @ bg_connection= torus/mesh
- The default is a mesh



Collective Network: Global Operations

- The global collective network is a high-bandwidth, one-to-all network that is used for collective communication operations, such as broadcast and reductions, and to move process and application data from the I/O Nodes to the Compute Nodes
- Each Compute and I/O Node has three links to the global collective network at 850 MBps per direction for a total of 5.1 GBps bidirectional bandwidth per node
- Latency on the global collective network is less than 2 µs from the bottom to top of the Collective, with an additional 2 µs latency to broadcast to all
- The global collective network supports the following features:
  - One-to-all broadcast functionality
  - Reduction operations functionality
  - 6.8 Gbps of bandwidth per link; latency of network traversal 2 µs
  - 62 TBps total binary network bandwidth
  - Interconnects all compute and I/O Nodes (1088)

Other Networks

- **Global Interrupt: Low latency barriers and interrupts**
  - The global interrupt network is a separate set of wires based on asynchronous logic, which forms another network that enables fast signaling of global interrupts and barriers (global AND or OR).
  - Round-trip latency to perform a global barrier over this network for a 72 K node partition is approximately 1.3 microseconds.
  - From MPI:

1 rack (Argonne)	8 racks (Argonne)	40 racks	72 racks
2.3 usec	2.5 usec	2.7 usec	3.0 usec

- **10 Gigabit Ethernet: File I/O and host interface**
  - This network consists of all I/O Nodes and discrete nodes that are connected to the external10 Gigabit Ethernet switch.
  - The Compute Nodes are not directly connected to this network. All traffic is passed from the Compute Node over the global collective network to the I/O Node and then onto the 10 Gigabit Ethernet network.

- **Control Network: Boot, monitoring, and diagnostics**
  - This network consists of a JTAG interface to a 1 GEthernet interface with direct access to shared SRAM in every Compute and I/O Node.
  - It is used for system boot, debug, and monitoring.
  - It allows the Service Node to provide runtime non-invasive RAS support as well as non-invasive access to performance counters.

Partitioning

- Partition = Subdivision of a single Blue Gene system
- Partitions are software defined
- Torus, Collective and Barrier networks are completely isolated from traffic from other partitions
- A single job runs on a partition
  - Jobs never share resources or interfere with each other
- Custom kernels may be booted in a partition

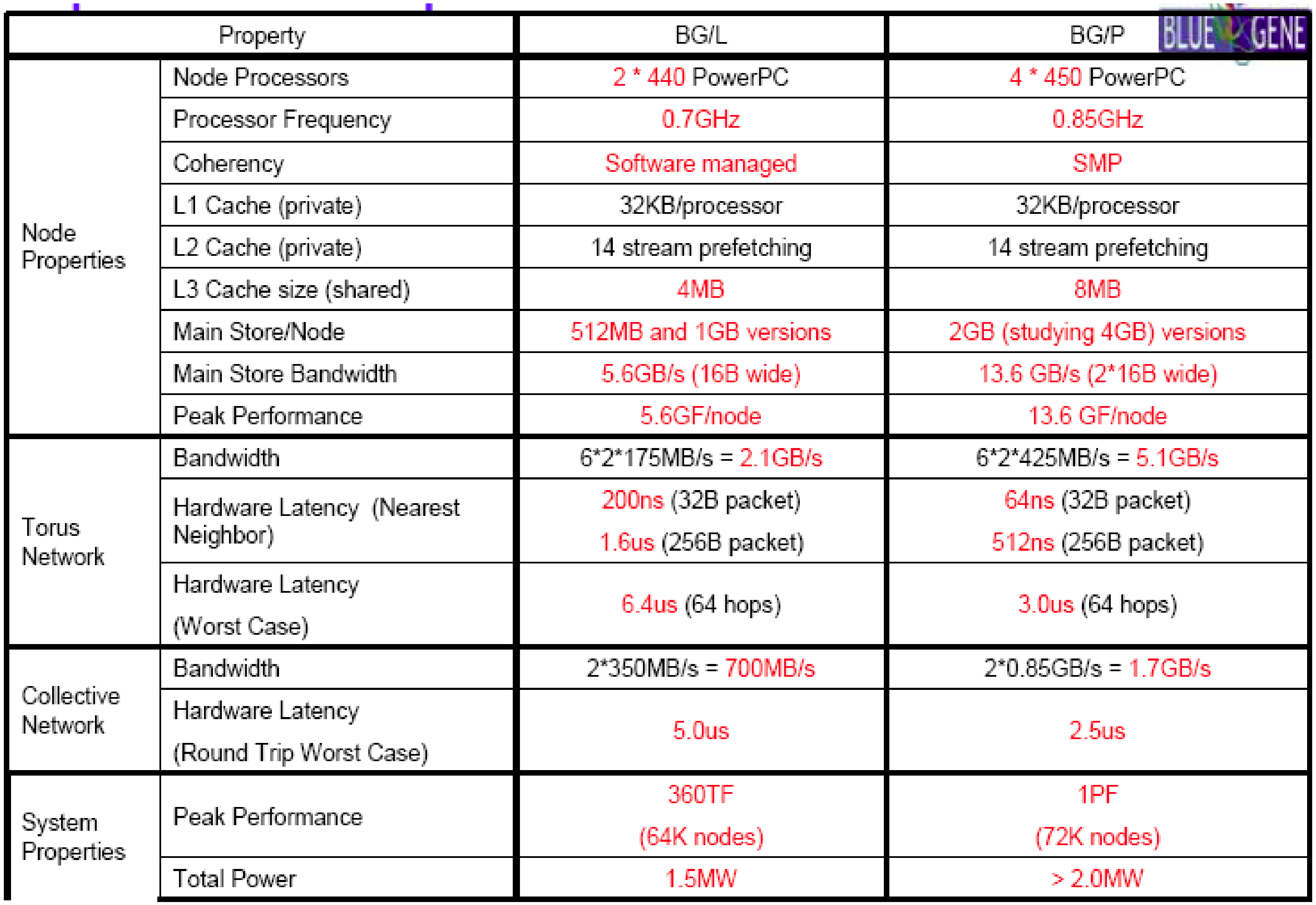
Sub-Midplane Partitions

- **Sub-Midplane Partition**
  - Partition that is smaller than a midplane
    - < 512 Nodes = 2048 Cores
  - Smallest partition is ½ node card (16 nodes)
  - Multiple node cards may form a partition
- **Limitations**
  - Mesh (not a true Torus)
  - Each partition must have at least one I/O node
  - Node cards combined into a partition must be adjacent
- **Sub-midplane partition sizes are powers of 2**
  - 16 node, 32 node, 64 node, 128 node and 256 node

Multi-Midplane Partitions

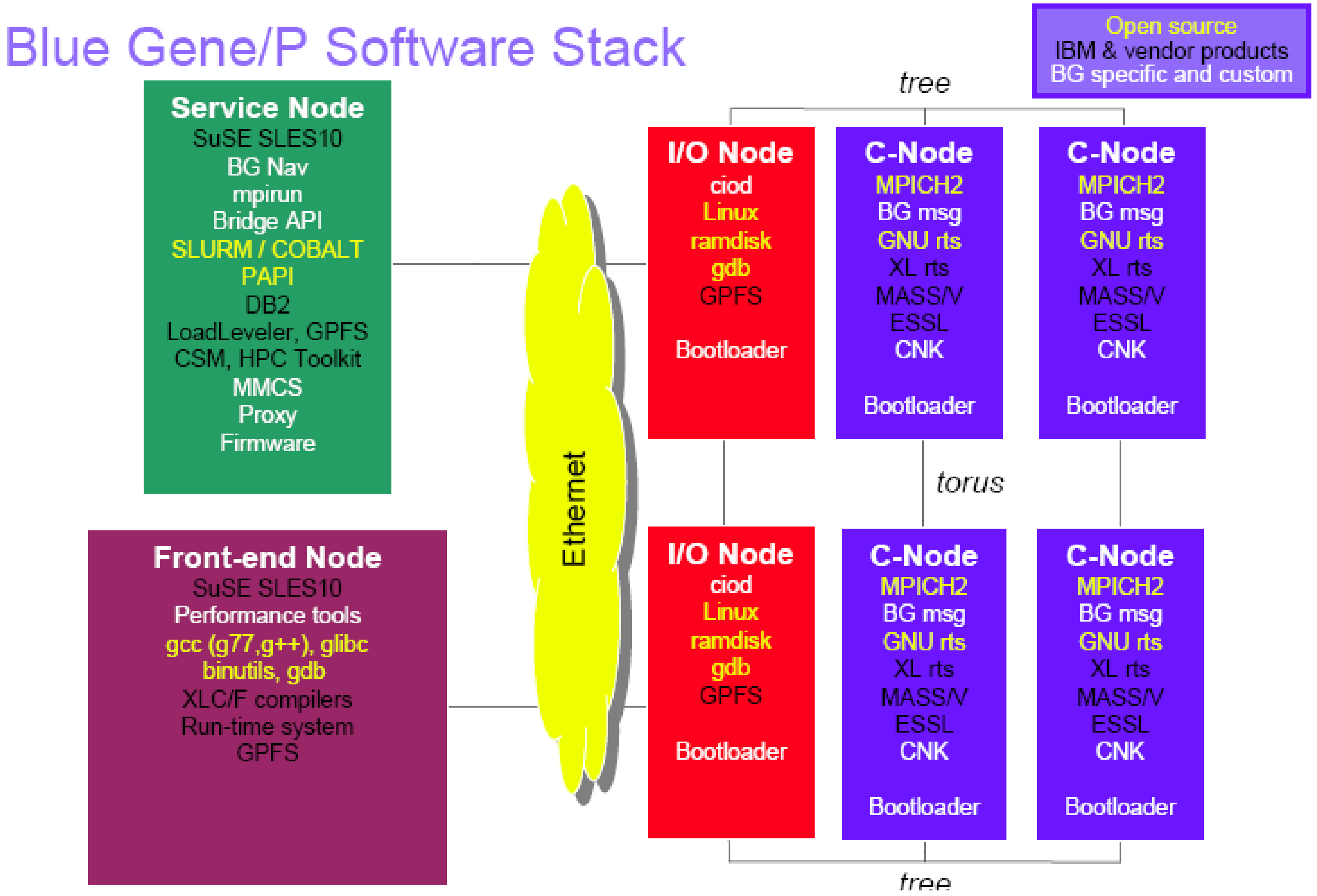
- **One or more midplanes can be grouped into a large partition**
- **Limitations**
  - Midplanes must be organized to form a torus
    - Constant number of midplanes in X, Y and Z
  - Torus connections may pass through unused midplanes
    - Traffic switched around those midplanes – no “noise” introduced
    - Unused midplanes can often be combined into partitions
  - Inter-rack cables are a limited resource
  - Inter-rack cables carry full network bandwidth
- **A single midplane may always be used as a torus**
  - No cables required

Spec/feature comparison



Property		BG/L	BG/P
Node Properties	Node Processors	2 * 440 PowerPC	4 * 450 PowerPC
	Processor Frequency	0.7GHz	0.85GHz
	Coherency	Software managed	SMP
	L1 Cache (private)	32KB/processor	32KB/processor
	L2 Cache (private)	14 stream prefetching	14 stream prefetching
	L3 Cache size (shared)	4MB	8MB
	Main Store/Node	512MB and 1GB versions	2GB (studying 4GB) versions
	Main Store Bandwidth	5.6GB/s (16B wide)	13.6 GB/s (2*16B wide)
	Peak Performance	5.6GF/node	13.6 GF/node
Torus Network	Bandwidth	6*2*175MB/s = 2.1GB/s	6*2*425MB/s = 5.1GB/s
	Hardware Latency (Nearest Neighbor)	200ns (32B packet) 1.6us (256B packet)	64ns (32B packet) 512ns (256B packet)
	Hardware Latency (Worst Case)	6.4us (64 hops)	3.0us (64 hops)
Collective Network	Bandwidth	2*350MB/s = 700MB/s	2*0.85GB/s = 1.7GB/s
	Hardware Latency (Round Trip Worst Case)	5.0us	2.5us
System Properties	Peak Performance	360TF (64K nodes)	1PF (72K nodes)
	Total Power	1.5MW	> 2.0MW

Blue Gene/P Software Stack

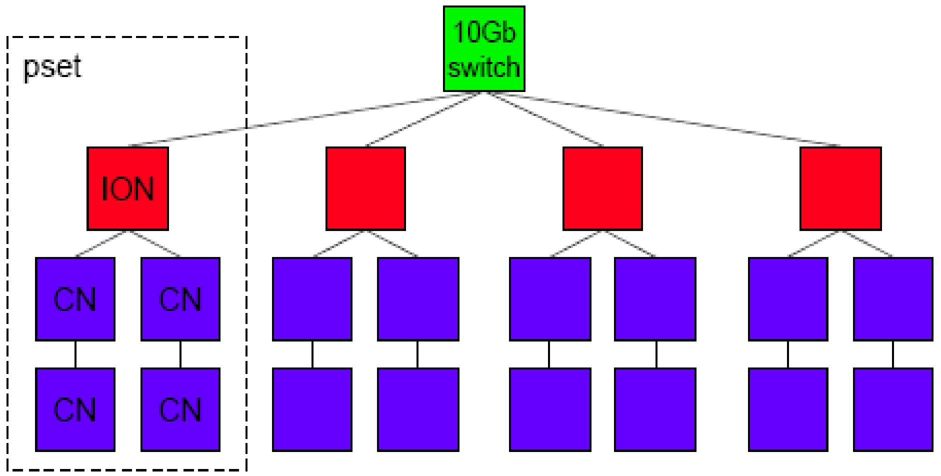


I/O Node Kernel

- **SMP Linux**
- **No persistent store**
  - Network filesystems only
  - No swap
- **10Gb Ethernet interface**
- **Several CNK System calls are function shipped to here**
  - Linux compatibility by executing these syscalls on Linux
  - Function shipping occurs over Collective Network
  - The ciod daemon manages a fixed set of compute nodes in a processing set (pset)
  - Linux provides the portable filesystem and network layer interfaces

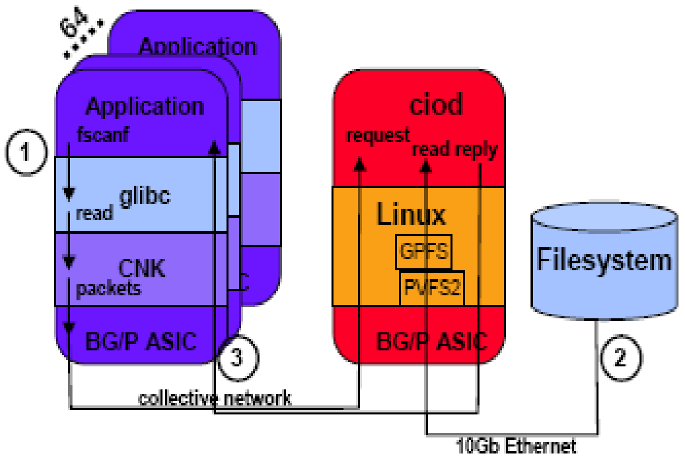
PROCESSING SETS (PSETS)

- I/O node dedicated to a fixed group of compute nodes
- Compute to I/O ratio is fixed within a partition
  - 128:1, 64:1, 32:1, 16:1



Syscall Function Shipping

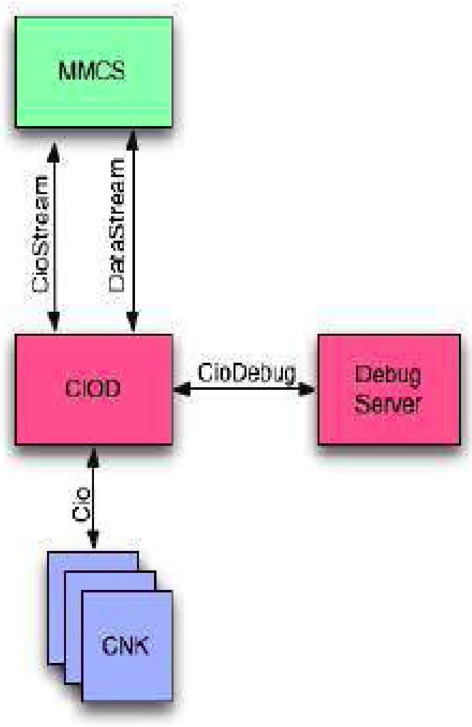
- I/O System Calls are forwarded by the Compute Node to the I/O Node
- Removes need for filesystem in CNK
- Manageable 1152 I/O nodes as filesystem clients in a 72 rack system



- Application performs an fscanf() library call
  - library uses read() syscall
  - CNK forwards read() as message over collective network
  - ciod receives read request
- Ciod performs read syscall under Linux
  - syscall performs Linux read via filesystem over 10Gb Ethernet
- Ciod reply to CNK contains read results
  - results returned to library, then app

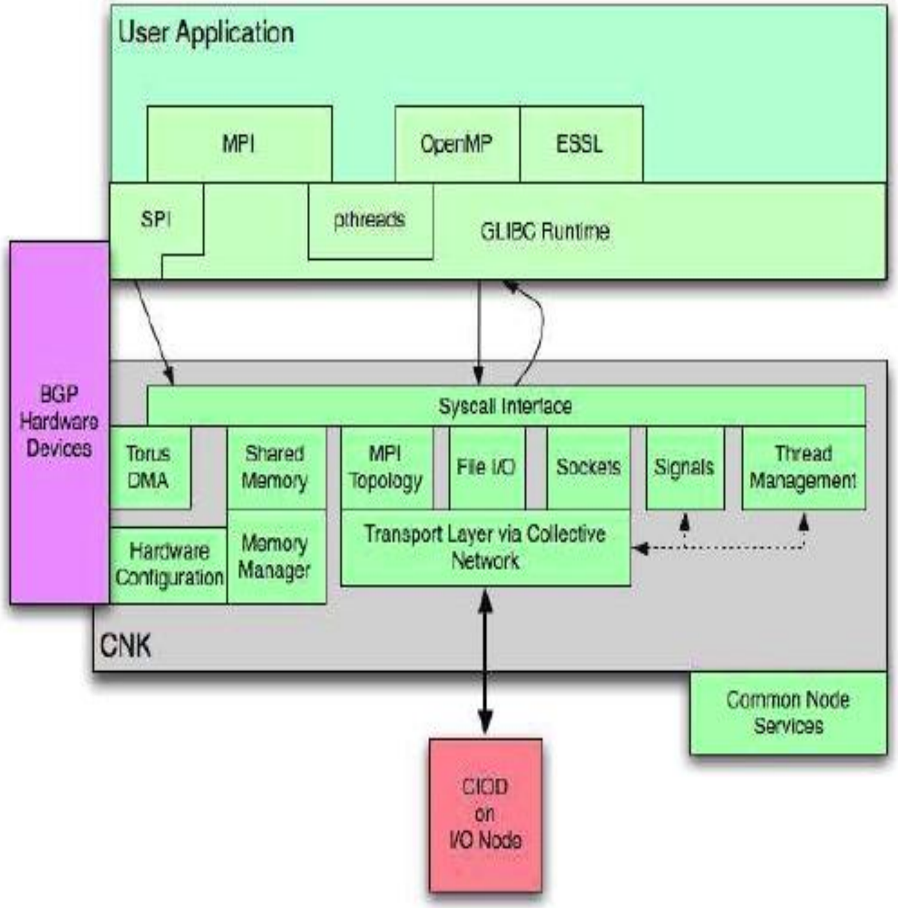
Compute I/O Daemon (ciod)

- **ciod on the I/O Node serves these roles**
  - Interface for job launch and kill
  - I/O syscall requests from CNs within the pset
  - Debug requests from a parallel debugger
- **Syscall compatibility**
  - ciod forks an "ioproxy" process for each CNK in the pset (actually each virtual node)
  - This process' context represents that compute node
    - open files, locks, working dir, I/O blocking state, etc
- **Service connection**
  - may be admin enabled
  - telnet to port 7201 and type "help"



Compute Node Software Stack

- Compute Node Kernel (CNK) controls all access to hardware, and enables bypass for application use
- User-space libraries and applications can directly access torus and tree
- Application code can use both processors in a compute node
- Application code can directly touch hardware only through SPI



Compute Node Kernel (CNK)

- **CNK is a lightweight kernel and is NOT Linux**
- **Goals**
  - Be as Linux compatible as possible
  - Provide entire node's resources to the application…and get out of the way!
- **OS noise is minimal by design**
  - TLBs are by default statically mapped – no page faults
  - Only the user application is running – no system daemons
  - No source of *normal* interrupts except for:
    - Timer interrupts as requested by the application
    - Torus DMA completion interrupts

CNK Modes of Execution

- **CNK provides a fixed process model**
  - SMP: single process with four threads and all memory
  - DUAL: two processes with two threads each and half memory
  - VN: “virtual node mode” with four processes and quarter memory each
  - e.g. mpirun –mode SMP to select the execution mode
- **Shared memory can be used between these processes**
- **Threading interfaces are fixed**
  - Each thread is fixed to a cpu
  - More threads than cpus are not allowed
  - Special I/O threads do exist (more later)
- **Memory usage is fairly rigid due to focus on pinned memory**

CNK System Calls

- **Direct Implementation**
  - exit, time, getpid, getuid, alarm, kill, times, brk, getgid, geteuid, getegid, getppid, sigaction, setrlimit, getrlimit, getrusage, gettimeofday, setitimer, getitimer, sigreturn, uname, sigprocmask, sched_yield, nanosleep, set_tid_address, exit_group
- **Implementation through forward to I/O Node**
  - open, close, read, write, link, unlink, chdir, chmod, lchown, lseek, utime, access, rename, mkdir, rmdir, dup, fcntl, umask, dup2, symlink, readlink, truncate, ftruncate, fchmod, fchown, statfs, fstatfs, socketcall, stat, lstat, fstat, fsync, llseek, getdents, readv, writev, sysctl, chown, getcwd, truncate64, ftruncate64, stat64, lstat64, fstat64, getdents64, fcntl64
- **Restricted Implementation**
  - mmap, munmap, clone, futex

CNK mmap and clone restrictions

- **These syscalls are partially implemented to enable the GLIBC pthread library and dynamic linker to function**
- **mmap is limited**
  - anonymous memory maps (malloc uses this)
  - maps of /dev/shm for shm_open()
  - file maps using MAP_COPY
  - mmap may choose different placement under CNK than Linux
  - mmap will not protect mapped files as readonly
  - mmap does NOT provide demand paging – files are completely read during the map
- **clone() may only create a total of 4 threads**
  - In VNM no additional clones allowed per process
  - In DUAL one clone allowed per process
  - In SMP three additional clones allowed
  - Fork is not supported

GLIBC pthread Support

- **New Posix Thread Library (NPTL) unchanged from Linux**
- **Attempt to create more threads than allowed results in EAGAIN error**
  - Linux rarely fails with this error (but can)
- **Stack overflow is caught by debug address exception**
  - Unlike Linux which uses guard pages
  - May be unavailable when under debugger
  - XLSMPOPTS runtime variable controls the stack size

CNK Dynamic Linking & Python (2.5)

- Dynamic linking provided by GLIBC ld.so
- Uses limited mmap implemented by CNK
- Placement of objects less efficient than static linking
- As in Linux, static linked apps can still use dlopen() to dynamically load additional code at runtime
- Goal is to enable use of Python to drive apps
  - pynamic has been demonstrated

CNK Full socket support

- CNK provides socket support via the standard Linux socketcall() system call. The socketcall() is a kernel entry point for the socket system calls. It determines which socket function to call and points to a block that contains the actual parameters, which are passed through to the appropriate call.
- The CNK function-ships the socketcall() parameters to the CIOD, which then performs the requested operation. The CIOD is a user-level process that controls and services applications in the compute node and interacts with the Midplane Management Control System (MMCS)
- This socket support allows the creation of both outbound and inbound socket connections using standard Linux APIs. For example, an outbound socket can be created by calling socket(), followed by connect(). An inbound socket can be created by calling socket()followed by bind(), listen(), and accept().

Blue Gene/P Software Tools

IBM Software Stack

- XL (FORTRAN, C, and C++) Compilers
  - Externals preserved
  - Optimized for specific BG functions
  - OpenMP support
- LoadLeveler Job Scheduler
  - Same externals for job submission and system query functions
  - Backfill scheduling to achieve maximum system utilization
- GPFS Parallel Filesystem
  - Provides high performance file access, as in current pSeries and xSeries clusters
  - Runs on I/O nodes and disk servers
- MASS/MASSV/ESSL Scientific Libraries
  - Optimization library and intrinsics for better application performance
  - Serial Static Library supporting 32-bit applications
  - Callable from FORTRAN, C, and C++
- MPI Library
  - Message passing interface library, based on MPICH2, tuned for the Blue Gene architecture

Other Software Support

- **Parallel File Systems**
  - Lustre at LLNL, PVFS2 at ANL
- **Job Schedulers**
  - SLURM at LLNL, Cobalt at ANL
  - Altair PBS Pro, Platform LSF (for BG/L only)
  - Condor HTC (porting for BG/P)
- **Parallel Debugger**
  - Etnus TotalView Debugger
  - Allinea DDT and OPT (porting for BG/P)
- **Libraries**
  - FFT Library - Tuned functions by TU-Vienna
  - VNI (porting for BG/P)
- **Performance Tools**
  - HPC Toolkit: MP_Profiler, Xprofiler, HPM, PeekPerf, PAPI
  - Tau, Paraver, Kojak, SCALASCA

Blue Gene/P Software Ecosystem 1

- **IBM Software Stack**
  - XL Compilers
    - Externals preserved
    - New options to optimize for specific Blue Gene functions
  - TWS LoadLeveler
    - Same externals for job submission and system query functions
    - Backfill scheduling to achieve maximum system utilization
  - Engineering Scientific Subroutine Library (ESSL)/MASSV
    - Optimization library and intrinsic for better application performance
    - Serial static library supporting 32 bit applications
    - Callable from FORTRAN, C, and C++
  - GPFS (General Parallel File System)
    - Provides high performance file access as in current System p and System x clusters
    - Runs on I/O Nodes and disk servers

Blue Gene/P Software Ecosystem 2

- **IBM Software Stack …**
  - MPI
    - Message passage library tuned for Blue Gene architecture
- **Performance Tools**
  - HPC Toolkit
- **Hardware**
  - DataDirect Networks (DDN), IBM System DCS9550; storage
  - Myricom, Force10; switches
- **3rd party ISV suppliers**
  - Visual Numerics (VNI) developers of IMSL
  - TotalView Technologies Inc. developers of TotalView Debugger
  - Allinea developers of allinea ddt and allinea opt
  - Tsunami Development LLC developers of the Tsunami Imaging Suite
  - Condor developers of software tools
  - Gene Network Sciences (GNS) developers of biosimulation models

What's new with BG/P

- **Torus DMA and numerous communication library optimizations**
- **pthreads and OpenMP support**
- **CNK application compatibility with Linux**
  - Dynamic linking
  - Use of mmap for shared memory
  - Protected readonly data and application code
  - Protection for stack overflow
  - Full socket support (client and server)
  - Better Linux compatibility in ciod on the I/O node
- **MPMD**
  - mpiexec supports multiple executables
  - Some restrictions: executable specified per pset; no DMA (next driver)
- **Numerous control system enhancements**

Programming Models & Development Environment

- **Familiar Aspects**
  - SPMD model - Fortran, C, C++ with MPI (MPI1 + subset of MPI2)
    - Full language support
    - Automatic SIMD FPU exploitation
  - Linux development environment
    - User interacts with system through front-end nodes running Linux – compilation, job submission, debugging
    - Compute Node Kernel provides look and feel of a Linux environment
      - POSIX system calls (with some restrictions)
      - BG/P adds pthread support, additional socket support,
    - Tools – support for debuggers, MPI tracer, profiler, hardware performance monitors, visualizer (HPC Toolkit), PAPI
    - Dynamic libraries
    - Python 2.5
- **Aggregate Remote Memory Copy (ARMCI), Global Arrays (GA), UPC, …**
- **Restrictions (lead to significant scalability benefits)**
  - Space sharing - one parallel job (user) per partition of machine, one process per processor of compute node
  - Virtual memory constrained to physical memory size
    - Implies no demand paging, but on-demand linking
  - MPMD model limitations

Execution Modes

- **Possibilities**
  - Single Node / Multi Node
  - 1, 2 or 4 Processes per Node
  - 1, 2 or 4 Threads per Process
- **Notation**

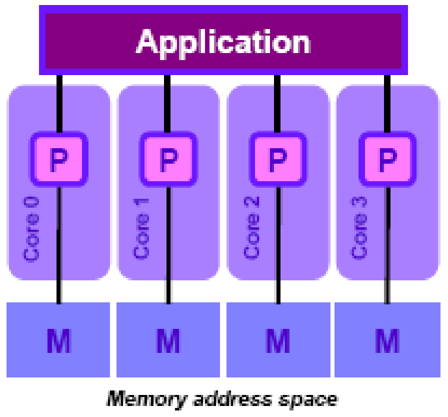
– Virtual Mode	VN	4 MPI Processes Per Node
– Dual Mode	DUAL	2 MPI Processes + 2 Threads Per Process
– Shared Memory	SMP	1 MPI Process + 4 Threads Per Process

- **Limitation**
  - One user process or thread per core

Blue Gene/P Execution Modes

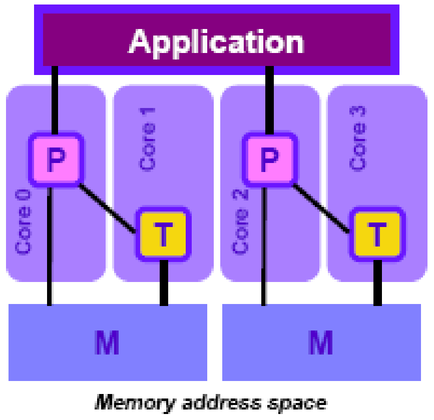
Quad Mode

- Previously called Virtual Node Mode
- All four cores run one MPI process each
- No threading
- Memory / MPI process = ¼ node memory
- MPI programming model



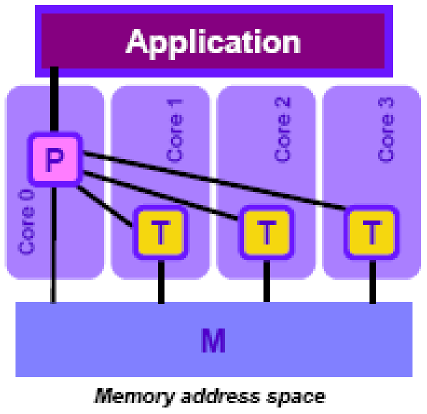
Dual Mode

- Two cores run one MPI process each
- Each process may spawn one thread on core not used by other process
- Memory / MPI process = ½ node memory
- Hybrid MPI/OpenMP programming model



SMP Mode

- One core runs one MPI process
- Process may spawn threads on each of the other cores
- Memory / MPI process = full node memory
- Hybrid MPI/OpenMP programming model

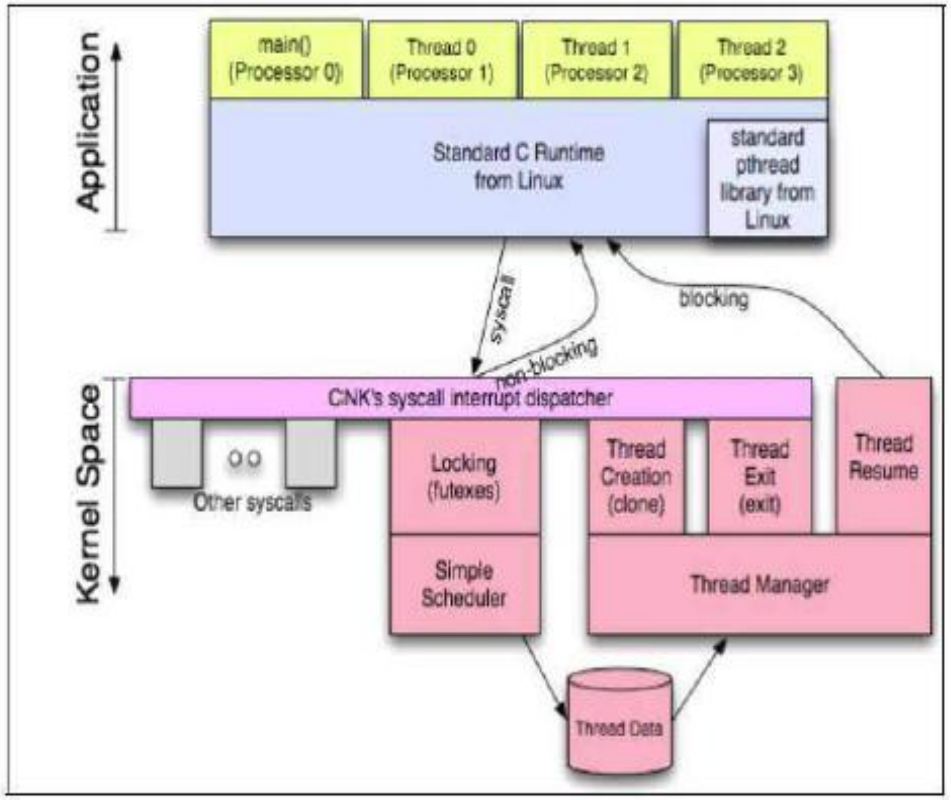


Programming Models

- **MPI Only – "Virtual Node Mode" with enhancements**
  - Separate MPI process for each processor in the compute node
  - DMA support for each MPI process
    - Ensure network does not block when processor is computing
    - Drive network harder
  - Sharing of read-only or write-once data on each node
    - Need programming language extension to identify read-only data
    - Allow applications to overcome memory limits of virtual node mode
- **MPI + OpenMP (or pthread)**
  - OpenMP within each node – relies on cache coherence support
  - Only master thread on each node initiates communication
    - Get benefits of message aggregation
    - Exploit multiple processors to service MPI call

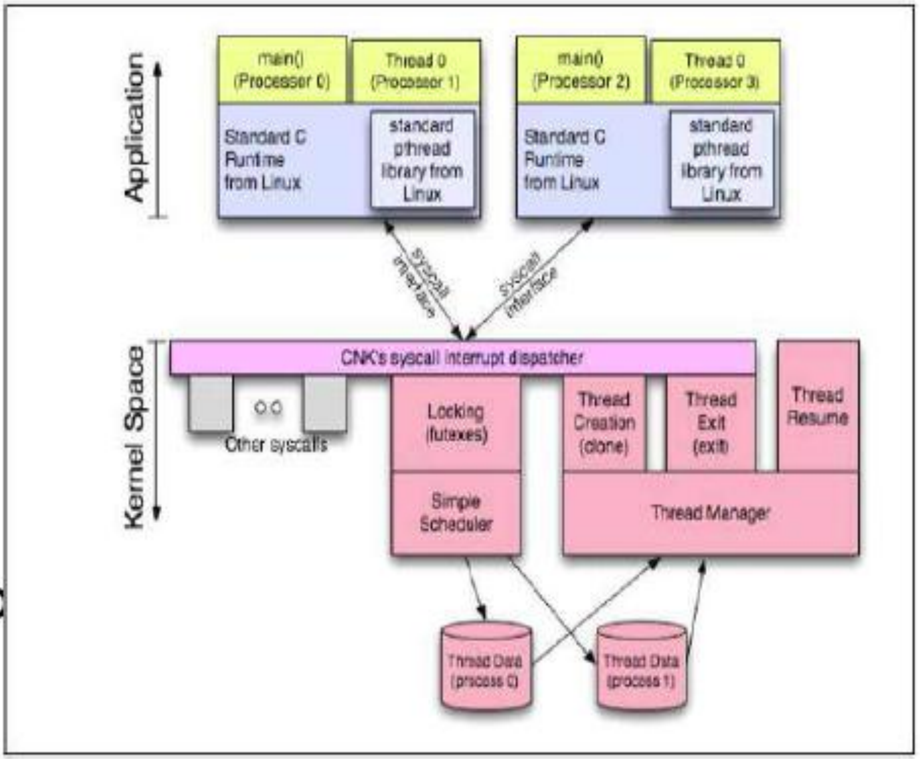
SYMETRICAL MULTI-PROCESSING MODE (SMP)

- SMP
- 1 Process/Node
- 4 Threads/Process
- 2 GB/Process
- pthreads and OpenMP are supported
- 4MB default stack for all new threads



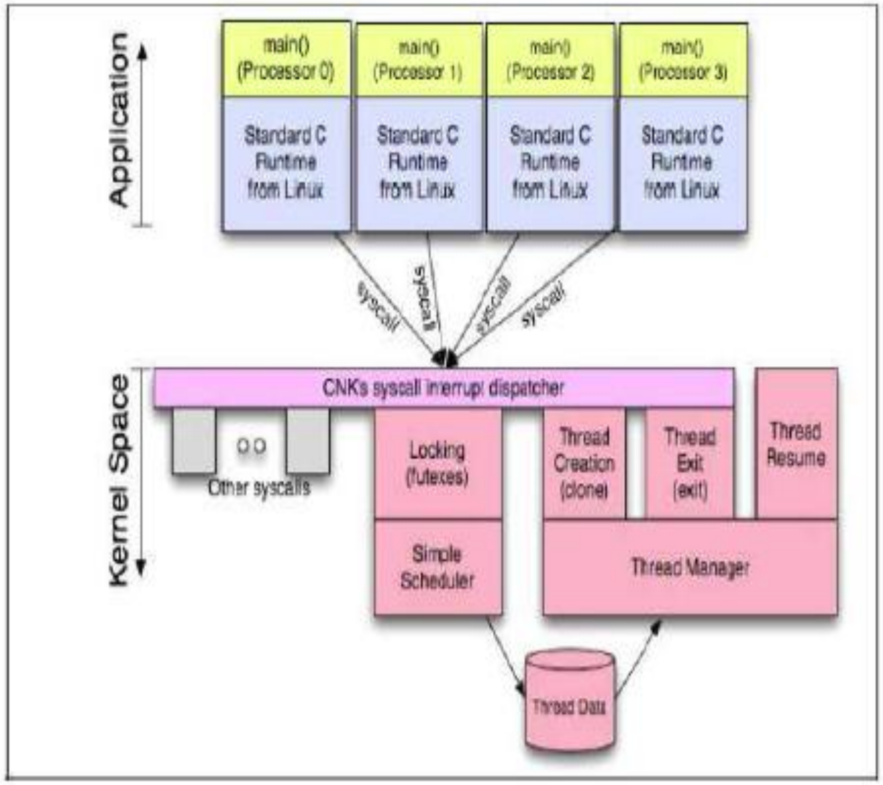
Dual Mode

- Dual
- 2 Processes/Node
- 2 Threads/Process
- 1 GB/Process
- pthreads and OpenMP are supported
- 4MB default stack fo all new threads



Virtual Node Mode

- VN
- 4 Processes/Node
- 1 Thread/Process
- 512 MB/process
- 512 MB/process versus Shared Memory support



High Throughput Computing (HTC) Mode

- **Many applications that run on Blue Gene today are "embarrassingly (pleasantly) parallel"**
  - They do not fully exploit the torus for MPI communication, since that is not needed for their problem
  - They just want a very large number of small tasks, with a coordinator of results
- **High Throughput Computing Mode on Blue Gene**
  - Enables a new class of workloads that use many single-node jobs
  - Leverages the low-cost, low-energy, small footprint of a rack of 1,024 compute nodes
    - Capacity machine ("cluster buster"): run 4,096 jobs on a single rack in virtual node mode (VN)

High Performance VS High Throughput Modes

- **High Performance Computing (HPC) Mode – Best for Capability Computing**
  - Parallel, tightly coupled applications
    - Single Instruction, Multiple Data (SIMD) architecture
  - Programming model: typically MPI
  - Apps need tremendous amount of computational power over short time period
- **High Throughput Computing (HTC) Mode – Best for Capacity Computing**
  - Large number of independent tasks
    - Multiple Instruction, Multiple Data (MIMD) architecture
  - Programming model: non-MPI
  - Applications need large amount of computational power over long time period
  - Traditionally run on large clusters
- **HTC and HPC modes co-exist on Blue Gene**
  - Determined when resource pool (partition) is allocated

- THANK YOU!!!