# BG/L Compute Node Kernel

Pat McCarthy Tom Gooding
IBM Rochester

# BG/L Compute Node Kernel Agenda

- CNK quick overview
- Additional function in Release 2 and Release 3
  - Release 3 is not available or announced – items listed are not guaranteed
- Future ideas

# CNK Quick Overview

- For more details – Tom Gooding presentation at Edinburgh – 2005
- Simple Linux-like kernel
  - Runs one process at a time
  - Uses small amount of memory – rest for the application
  - Supports attaching debuggers
- CNK provides a subset of the Linux system calls
  - File I/O
  - Directory operations
  - Signals (ANSI C only)
  - Process information
  - Time
  - Sockets
- Goal is to stay out of the way and let the application run

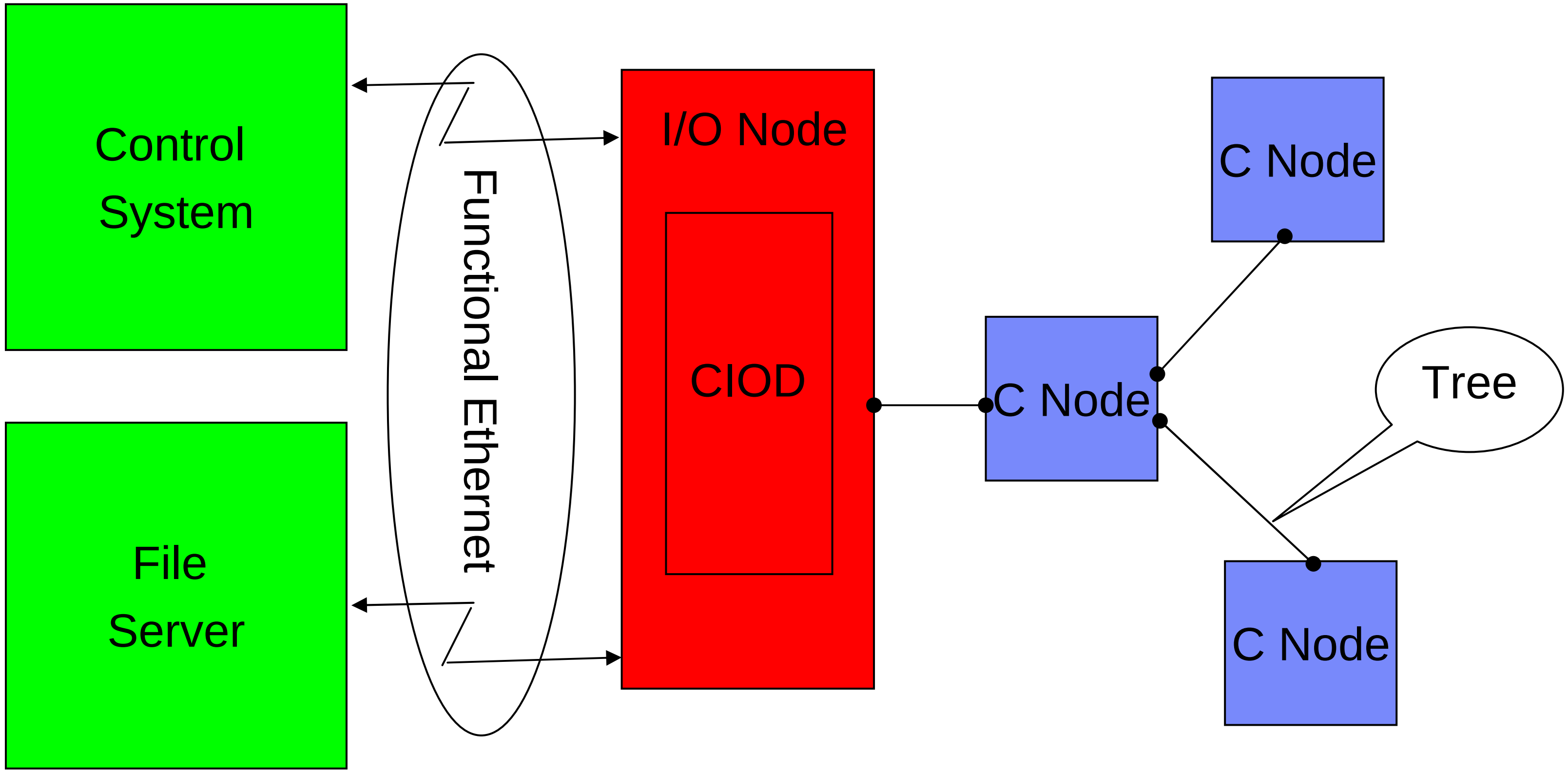
# CNK Function Shipping

- All I/O must be processed on the I/O node
- CIOD is a user mode application running on the I/O node that:
  - Manages the compute nodes for the control system
  - Manages descriptors, working directory, umask for compute nodes
  - Performs all I/O for all compute nodes
  - Manages the debugger connections to the compute nodes
- Ratio of compute nodes to I/O nodes differs between machines
- All communication between CIOD and compute nodes is over virtual channel 0 of the tree network

# CIOD Overview

- CIOD processes requests from:
  - Control system using a socket to the service node
  - Debug server using a pipe to a local process
  - Compute nodes using the tree network
- CIOD has one main thread which does all of the work
  - Only one processor is used on the I/O node
  - It must never block when processing a request
  - One secondary thread handles the service connection

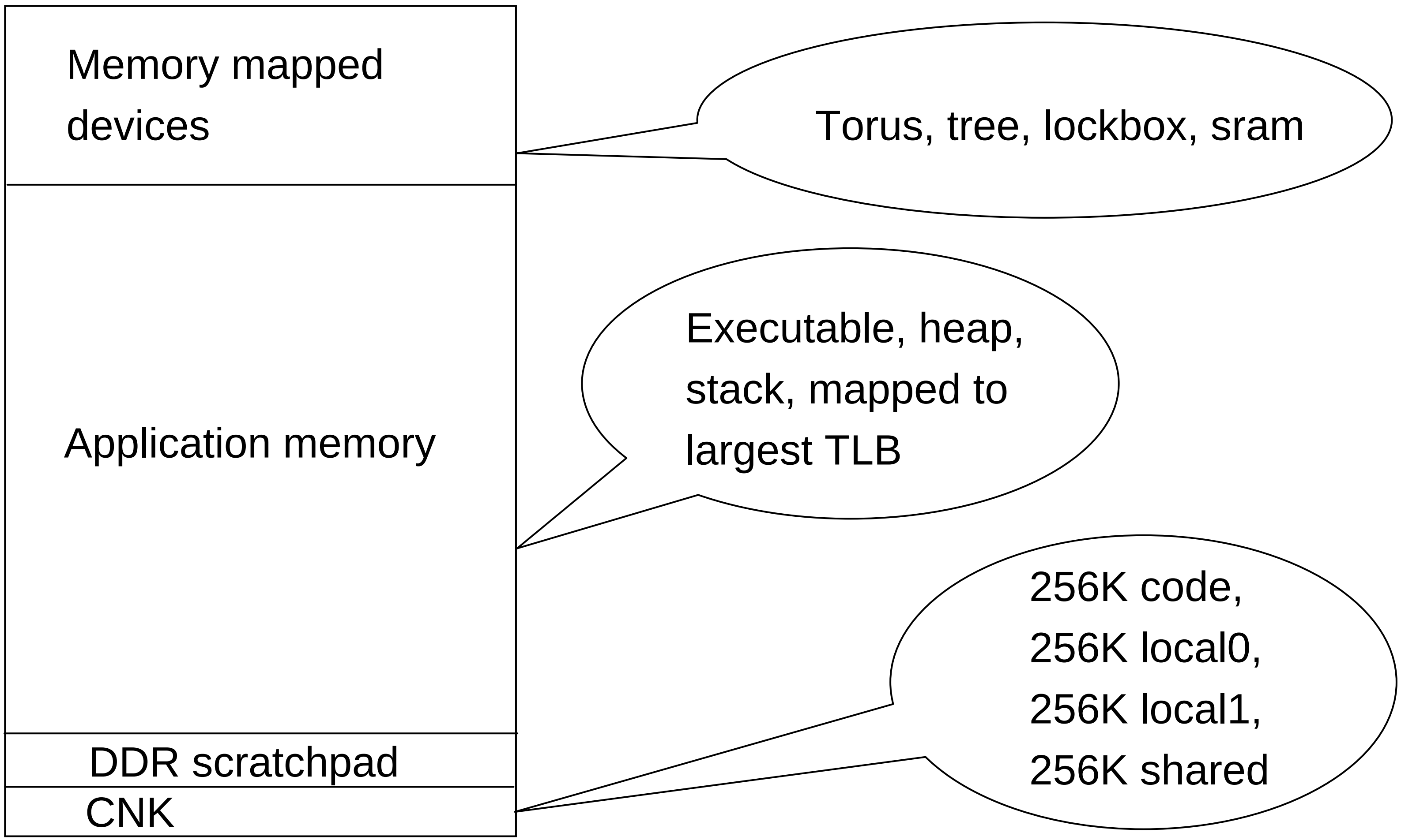
# CNK Function Shipping



# CNK Modes

- Coprocessor mode
  - Application runs on processor 0
  - Very limited environment for running code on processor 1
  - MPI uses coprocessor for offloading communications
- Virtual node mode
  - Application is loaded and runs on both processors
  - Memory is divided in half
  - Application is responsible for sharing resources
- Mode is selected at job start time

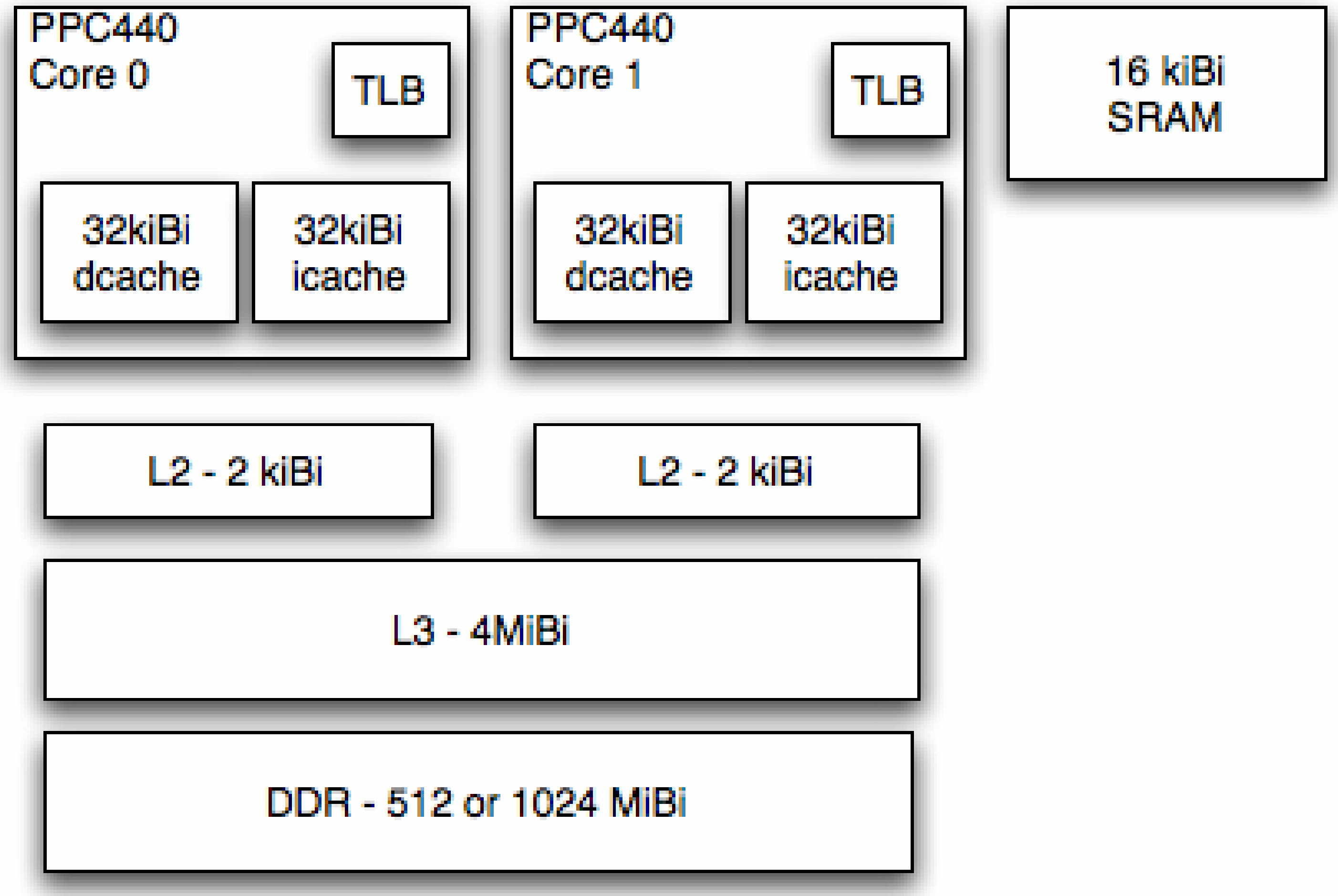
# Compute Node Memory Map



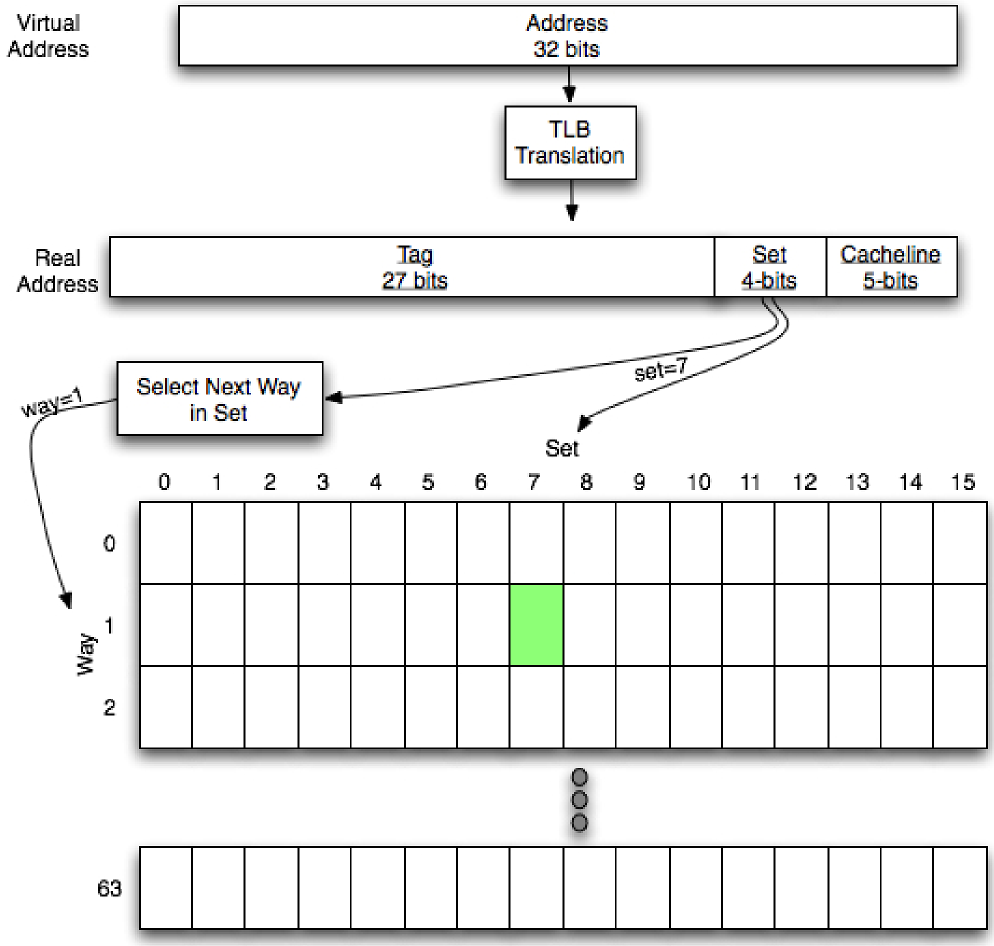
# Release 2 and Release 3 enhancements

- BG/L Memory subsystem is very flexible and powerful
  - Allow adjustments to memory subsystem on an application basis
  - Job level L1 cache modes for improved performance
- Improvements for QCD applications
  - SRAM allocate
  - SWOA region
- Open source tree device driver
- Interrupt driven communication support
  - One sided ops
- File and socket IO performance enhancements
- CIOD callouts during job startup
- MPMD (Multi-program / Multi-data) support
- Continued code improvements

# Memory Subsystem Overview



# L1 Cache Overview



# L1 cache modes for improved performance

- The L1 cache characteristics for a job can be specified using environment variables
- SWOA – store without allocate mode (BGL_APP_L1_SWOA)
  - TLB attribute and/or MMUCR flag
    - Per address range or system wide
  - Store misses do not allocate a line in the data cache
    - If line already in the cache – store still occurs to the cache
  - Performance of certain applications can be affected by the allocation of cache lines on store misses
    - If the store accesses for a particular application are distributed sparsely in memory and if the data is not reused after the store
    - Avoids the latency and bus bandwidth associated with filling the entire cache line
- Many applications have been helped by this option
  - Avoids loading the line into L1 for a read-modify-write when the line is not already in cache
  - Evicting cache lines can be more efficient because cache is cleaner
  - Greatest benefit when in write through mode
- Could slow down some applications – try both ways

# Improvements for QCD applications

- In order to get improved performance, the QCD community needed access to low latency memory and L1 cache settings
- SRAM allocate
  - SRAM is low latency 16 KB of memory accessible by both 440 cores
  - No caching of SRAM – coherent between cores
  - Allows fast communication between the cores
  - APIs provided to gain access to SRAM storage
- SWOA region
  - Small amount of storage to have the SWOA attribute, but not all memory as provided by the SWOA environment variables
  - Implementation similar to DDR_scratchpad (used for MPI)
    - Linker script to move application to open up storage for SWOA region
    - System call to get the address and size of the SWOA region

# Open source tree device driver

- Request from ANL
  - Enables ZeptoOS research
  - 2.6 linux version
- Interfaces provided
  - Open
    - /dev/tree – virtual channel 0
    - /dev/tree0 – virtual channel 0
    - /dev/tree1 – virtual channel 1
  - Provides a memory mapping to the tree hardware
  - Does not provide interface to actually read/write the device
    - Enables best performance because a system call is not required to read or write the device - device mapped into user storage

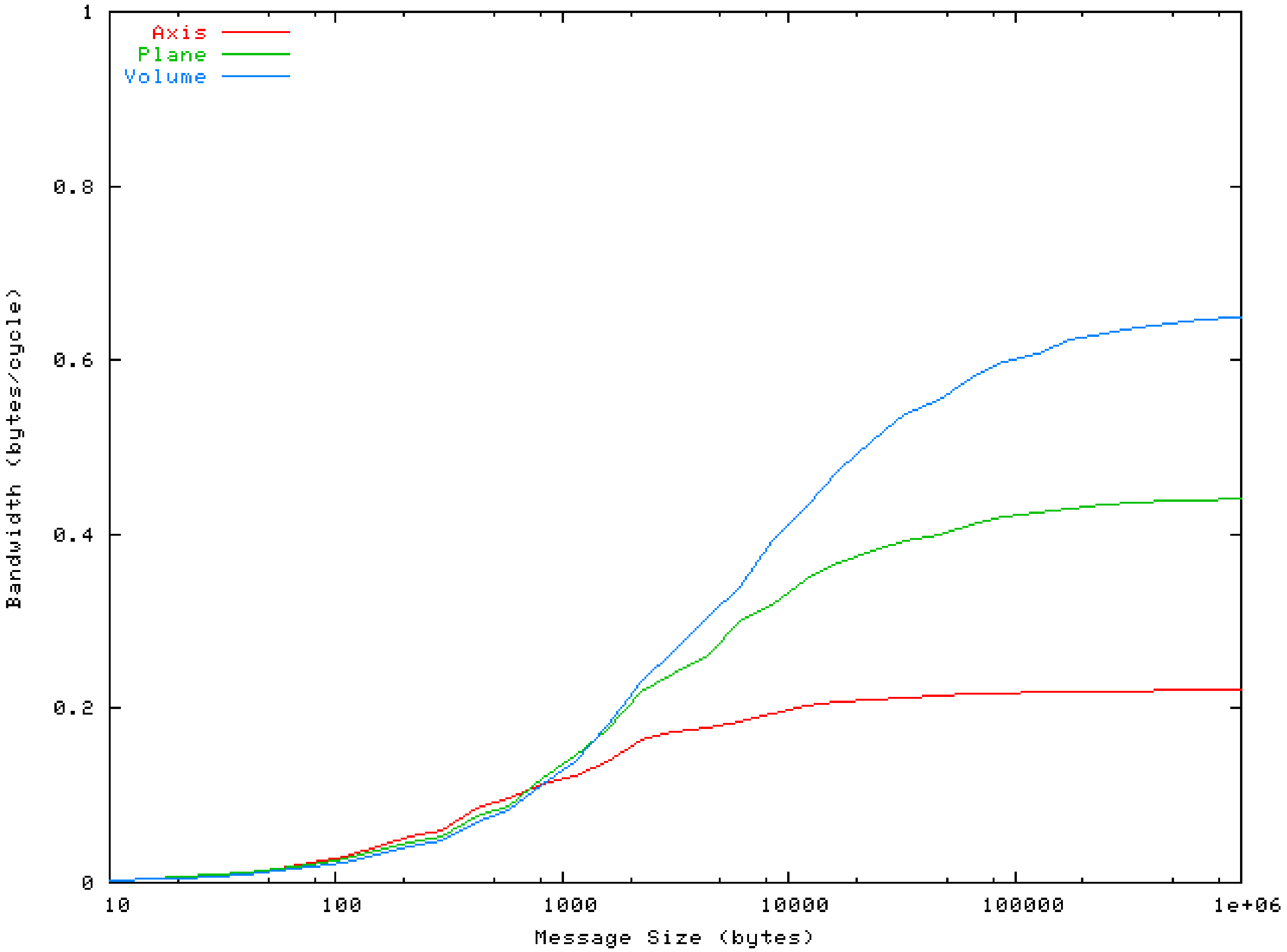
# Interrupt driven communication support

- Kernel support for one sided operations
- Uses watermark hardware for the torus to interrupt the kernel on the arrival of a torus packet and invoke a message layer handler
- Current enabling  ARMCI (Aggregate Remote Memory Copy Interface)
- Global Arrays uses both MPI and ARMCI
- ARMCI and GA are PNNL (Pacific Northwest National Labs) libraries
- Significant applications
  - NWChem
  - GAMESS-UK
  - GPSHMEM

# Message Layer improvements

- Better queue management
- Early results look promising (charts)
  - Axis – communicator in one dimension/axis
    - Theoretical peak .22 bytes/cycle
  - Plane – communicator in two dimensions/axis
    - Theoretical peak .44 bytes/cycle
  - Volume – communicator in three dimensions/axis
    - Theoretical peak .66 bytes/cycle
  - These are measurement of  one direction data transfer ½ the maximum distance of the communicator in a non-congested network

# Early results from one way communications



# File and socket IO performance enhancements

- In the previous implementation, during a read operation data was copied from the tree hardware to a kernel buffer and then copied into the user buffer
- Data is now copied directly from the tree hardware to the user buffer
- Implemented to solve socket performance issue but also will increase performance for file reads when small number of nodes in a pset are reading

# File and Socket IO Environment Variables

- These variables allow adjustment which may be used to improve performance in specific situations
  - For most applications, the default setting are the best performance
- CIOD_RDWR_BUFFER_SIZE
  - Buffer size for read/write system calls
  - CIOD allocates one buffer for each compute node
  - Default - 87600 in V1R1 and V1R2 – TBD in V1R3 (doing performance test)
- CIOD_SOCKET_BUFFER_SIZE
  - Size of send/receive buffers for sockets
- CIOD_TREE_MULTIPLIER
  - Controls how often CIOD checks for traffic form the compute nodes before checking for control or waiting IO
- CIOD_TREE_RECV_RETRY
  - The number of time CIOD reads the tree device before deciding a packet is not available and yielding the processor
- CIOD_TREE_SEND_RETRY
  - The number of times CIOD tries to send a packet to the tree device before yielding the processor
  - A larger value causes CIOD to pay more attention to the tree

# CIOD callouts during job startup

- Request  to allow file system code to register a script to be called by CIOD when
  - Job startup and termination
- Parameter passed to script include
  - Job mode – virtual node mode vs coprocessor mode
  - Partition size
  - Job ID

# MPMD (Multi-program / Multi-data) support

- Request to support CCSM application
- Based on a design originally prototyped by Watson Research and will now be integrated into the product
  - Kernel will provide an rts_exec interface to allow a compute node to switch the application running on the node
  - Will only be allowed once per node / per job (no multiple execs allowed)
  - The database entry for the job will contain the original program started on all the nodes at job startup (probably a simple launch program which would rts_exec the program on each node)
- Integration of external tools (debuggers) is an issue to be resolved
  - Can specify executable when starting the debug client (gdb)
- Support for checkpoint / restart
  - Checkpoint restart uses MPI barriers
- Environment variables design still evolving
- All applications in the partition must be compiled with the same tool chain and same level of MPI libraries

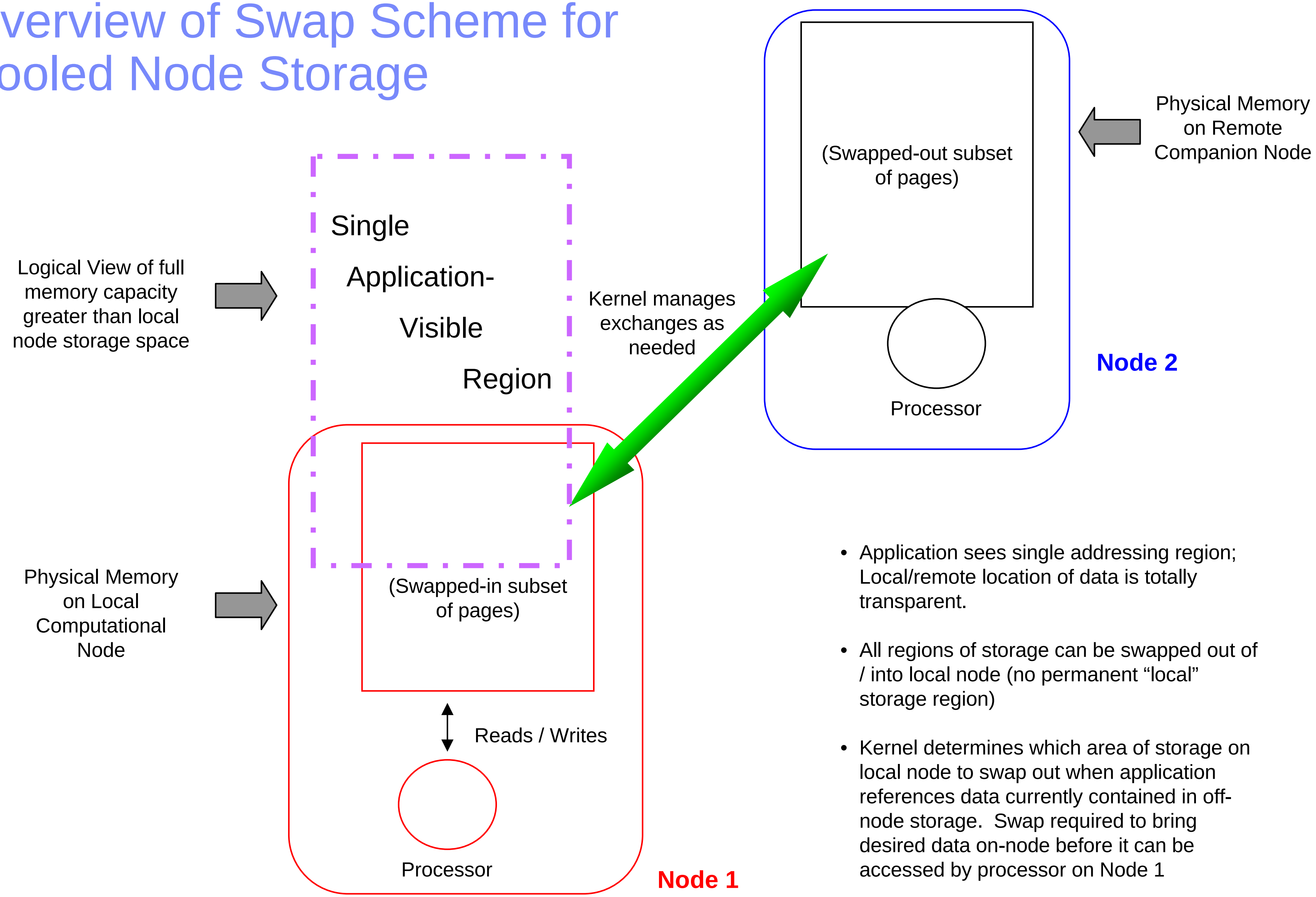
# Continued code improvements

- IBM continues to make enhancements to improve the general quality of the product
  - CN -> CIOD message format enhancements
  - IO kernel upgrade
    - SLES 9 SP1 -> SLES 9 SP2
  - RAS improvements
    - Link chip training failures isolation improvements
  - Interrupt handling improvements for multiple signals

# Future Ideas

- These are items we are considering – but funding is a consideration
- Pooled node storage
  - Adds ability for nodes to share memory between nodes
  - Memory swapping between the nodes
- Improvements for QCD applications
  - L1 transient storage

# Overview of Swap Scheme for Pooled Node Storage

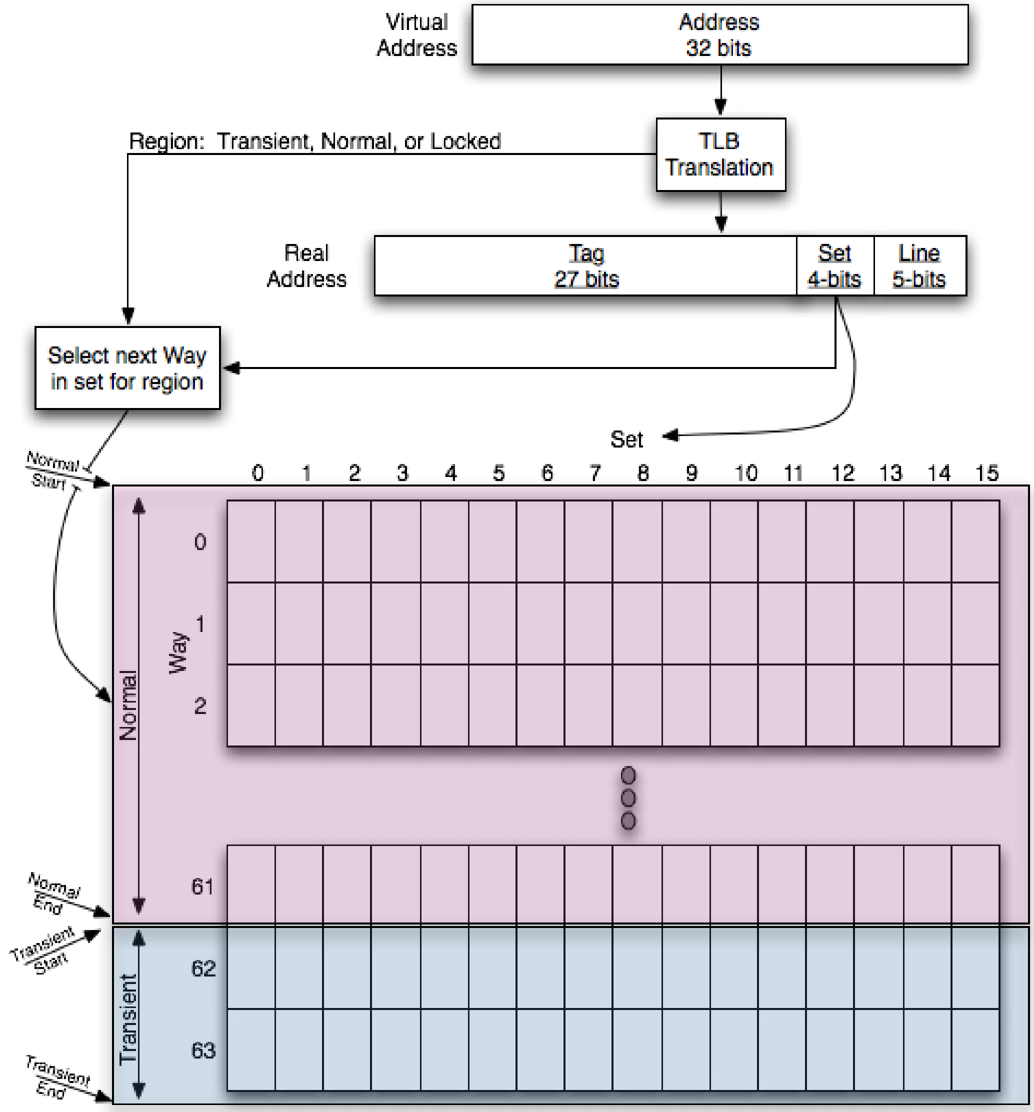



- Application sees single addressing region; Local/remote location of data is totally transparent.
- All regions of storage can be swapped out of / into local node (no permanent “local” storage region)
- Kernel determines which area of storage on local node to swap out when application references data currently contained in off-node storage. Swap required to bring desired data on-node before it can be accessed by processor on Node 1

# Highlights of Pooled Node Storage for Blue Gene/L

- **Expands physical node memory of both 512MB and 1GB machines; (*up to 2X addressable capacity)***
- **Swapping would be an LRU algorithm**
  - 1 meg page size
  - 44 – 48 tlbs for user address range
- **Pools memory from node pairs**
  - Leverages existing connection path between node pairs for new communications facility with minimal contention
- **Flexible trade-off of compute cycles/function vs. memory capacity**
  - One extreme: All app function occurs on “computational node” in pair; Uses all local memory plus all memory on “companion node”
  - Middle ground: App function split across nodes (heterogeneous). E.G. Memory capacity available to Node ‘A’ = 1.5X; to Node ‘B’ = 0.5X
  - Dynamic memory split across node pairs
    - E.G. startup scenario: Actual memory/node = 1X; Node ‘A’ starts with 1.5X memory usage -> Node ‘B’ starts with 1.5X memory -> both run with 1X memory
  - *Changes CNK tlbs from static to dynamic when running in pooled storage mode*
    - *Could be used to implement MPROTECT system calls*

# L1 Cache Regions



# Additional Improvements for QCD applications

- L1 cache can be divided into a normal, transient, and locked regions
  - Cache is divide into sets (rows) and ways (columns) – 32K
  - Effective addresses are hashed into a set
  - Can divide the “ways” of each cache set between the regions
  - Effects cache replacement policy
    - Round robin within the region
- The TLB for an Effective Address will specify the region for the address (normal vs transient)
  - LI cache will be round robin within the region
  - API to allow users to specify the region for a range of EA’s
- Allows an application to be tuned such that one time use data does not pollute the working set of the algorithm
- Can really hurt performance if done incorrectly by actually shrinking the amount of available cache

# Summary

- We continue to enhance the CNK based on
  - Customer needs
  - New opportunities
  - Improved availability
- The simplicity of the original CNK design has allowed for incremental enhancements