



IBM Research

MPI Internals

George Almási, Charles Archer, Xavier Martorell,
Chris Erway, José Moreira

Contents

- Layers of communication software
- BGL/MPI roadmap & status
- MPI Point-to-point messages
- Implementing collective communication primitives
- Process management
- Preliminary performance results
- Lessons learned so far
- Conclusion

Layers of BlueGene/L Communication Software

■ Packet layer

- ❖ Initialize network HW, (tree & torus), send and receive packets
- ❖ As simple as we can afford to make it

■ Torus message layer

- ❖ Active message layer similar to LAPI and GAMMA
 - on top of packet layer
- ❖ Handles hardware complexity
 - alignment, ordering, transmission protocols
 - Cache coherence, processor use policy

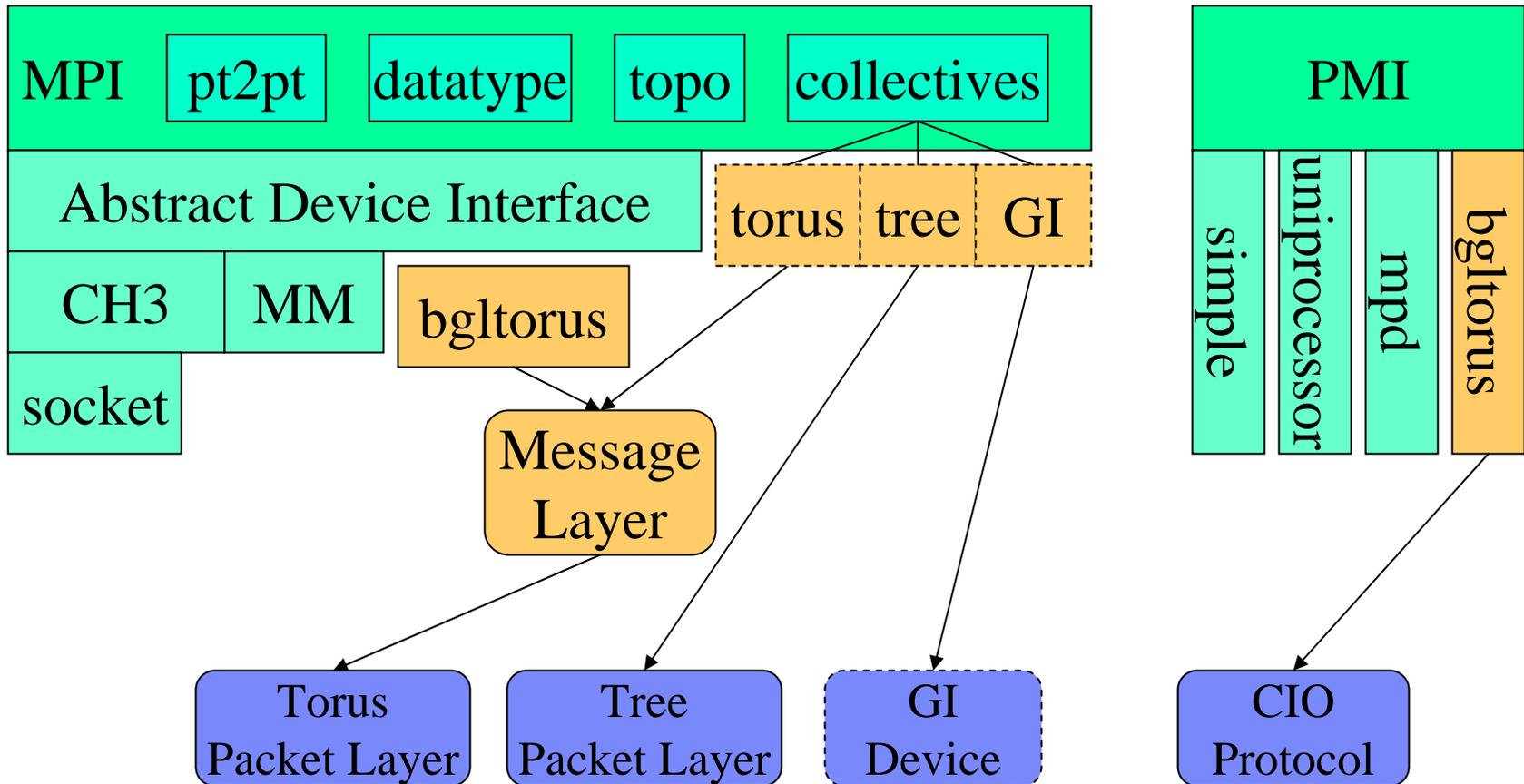
■ MPI

- ❖ BlueGene/L is primarily an MPI machine
- ❖ A port of Argonne National Labs' MPICH2
- ❖ Currently deployed: beta 0.93 – about to upgrade to 0.94

The MPICH2 BG/L Roadmap

Message passing

Process management



MPI Implementation Status Today (10/14/2003)

■ Point-to-point communication

- ❖ MPI-1 compliant, except:
 - No synchronous sends
 - Missing MPI_Cancel
 - Buggy & suboptimal handling of non-contiguous data streams
- ❖ No one-sided comm.
- ❖ Eager protocol only
 - No flow control
- ❖ Heater mode only

■ Process management

- ❖ Two hard-coded processor layouts available (XYZ, ZYX)
- ❖ Underway: user-defined processor layout

■ Optimized collectives:

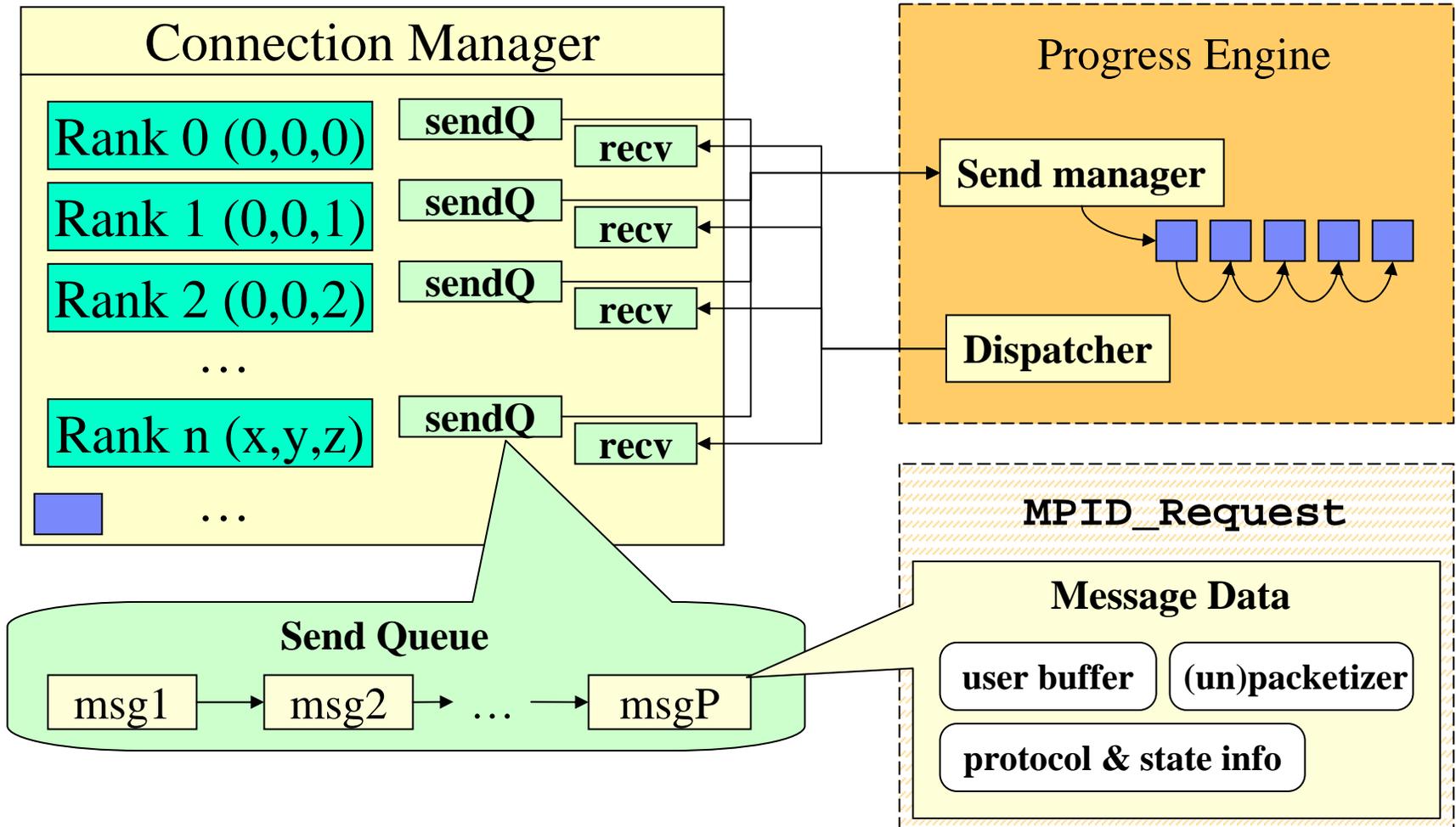
- ❖ First steps towards torus/mesh optimized broadcast

Point-to-point Communication

■ Basic MPI functionality

- ❖ `MPI_Send()`, `MPI_Recv()`
- ❖ Enough to get MPI-1 compliance in MPICH2.
 - MPICH2 provides everything else
- ❖ **Do-or-die**: no high performance MPI without good point-to-point communication performance
- ❖ Implementation:
 - Glue layer (“mpid/bgltorus”): implementation of ADI
 - Torus message layer
 - Torus packet layer

The Torus Message Layer



Message Layer API

■ Initialization & advance

- ❖ `BGLML_Initialize()`
- ❖ `BGLML_RegisterProtocol()`
- ❖ `BGLML_Advance()`
- ❖ ...

■ Message creation

- ❖ `BGLMP_EagerSend_Init()`
- ❖ `BGLMP_RvzSend_Init()`
- ❖ `BGLMP_EagerRecv_Init()`
- ❖ ...

■ Sending:

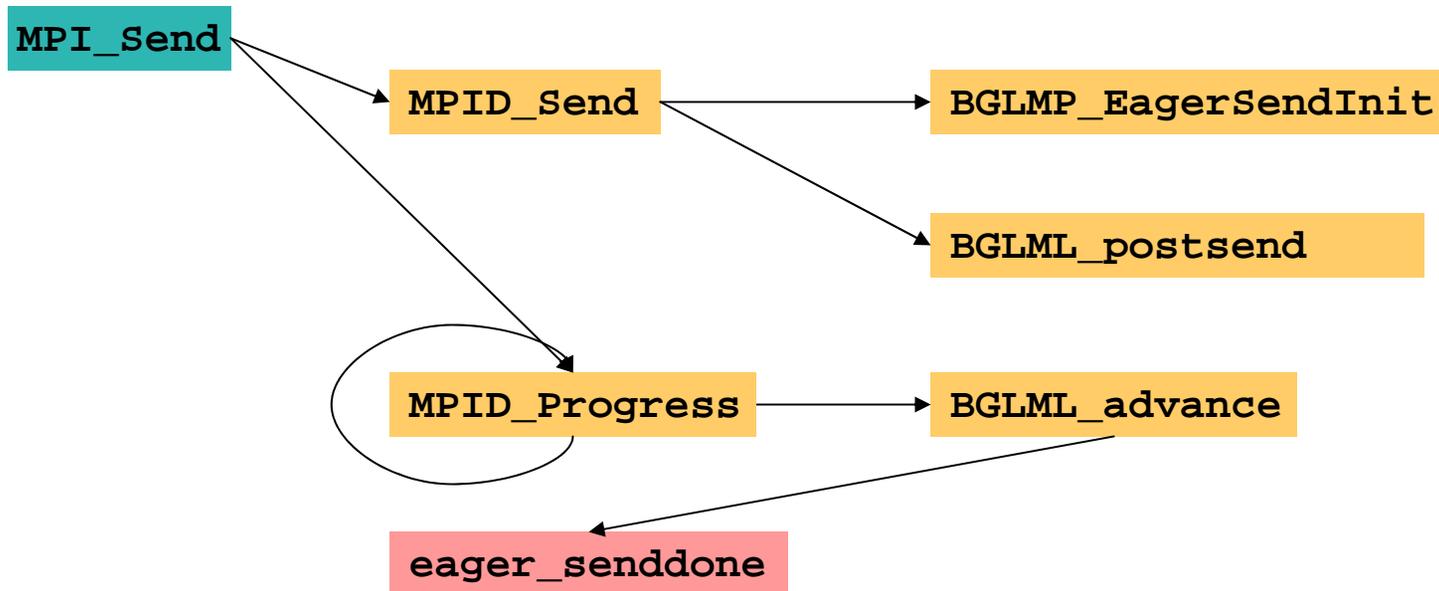
- ❖ `BGLML_postsend()`

■ Upcall prototypes:

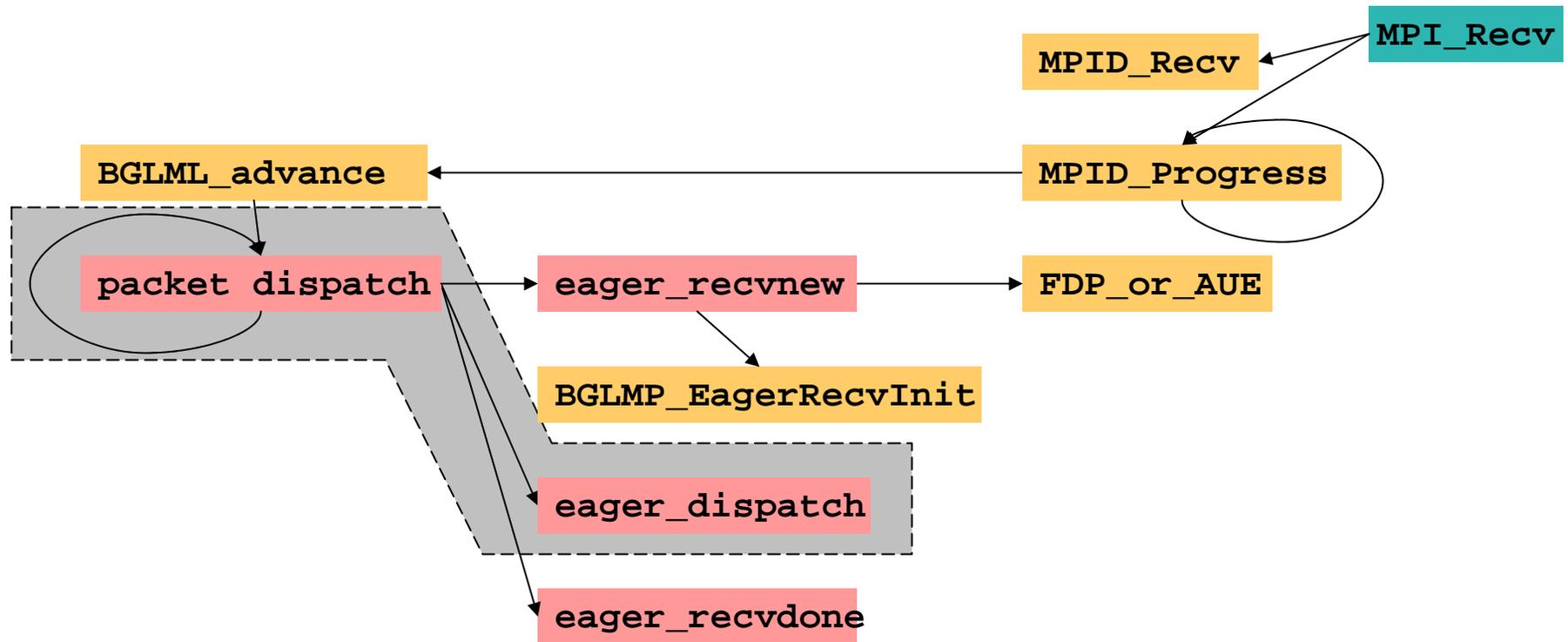
- ❖ `cb_recvnew()`
- ❖ `cb_recvdone()`
- ❖ `cb_senddone()`

- ❖ `cb_dispatch()`

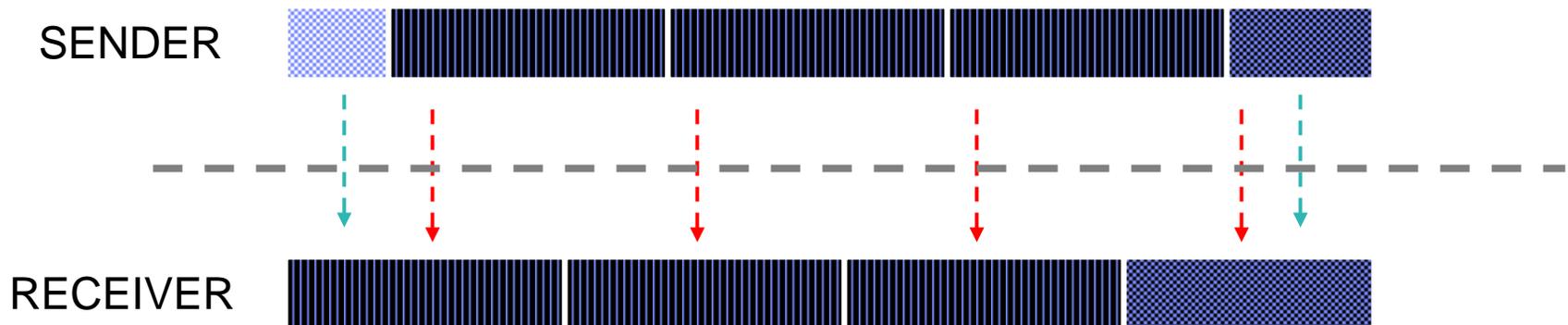
The Eager Message Protocol: send side



The Eager Message Protocol: receive side



Packetization and packet alignment



- **Constraint:** Torus hardware only handles 16 byte aligned data

- When sender/receiver alignments are same:

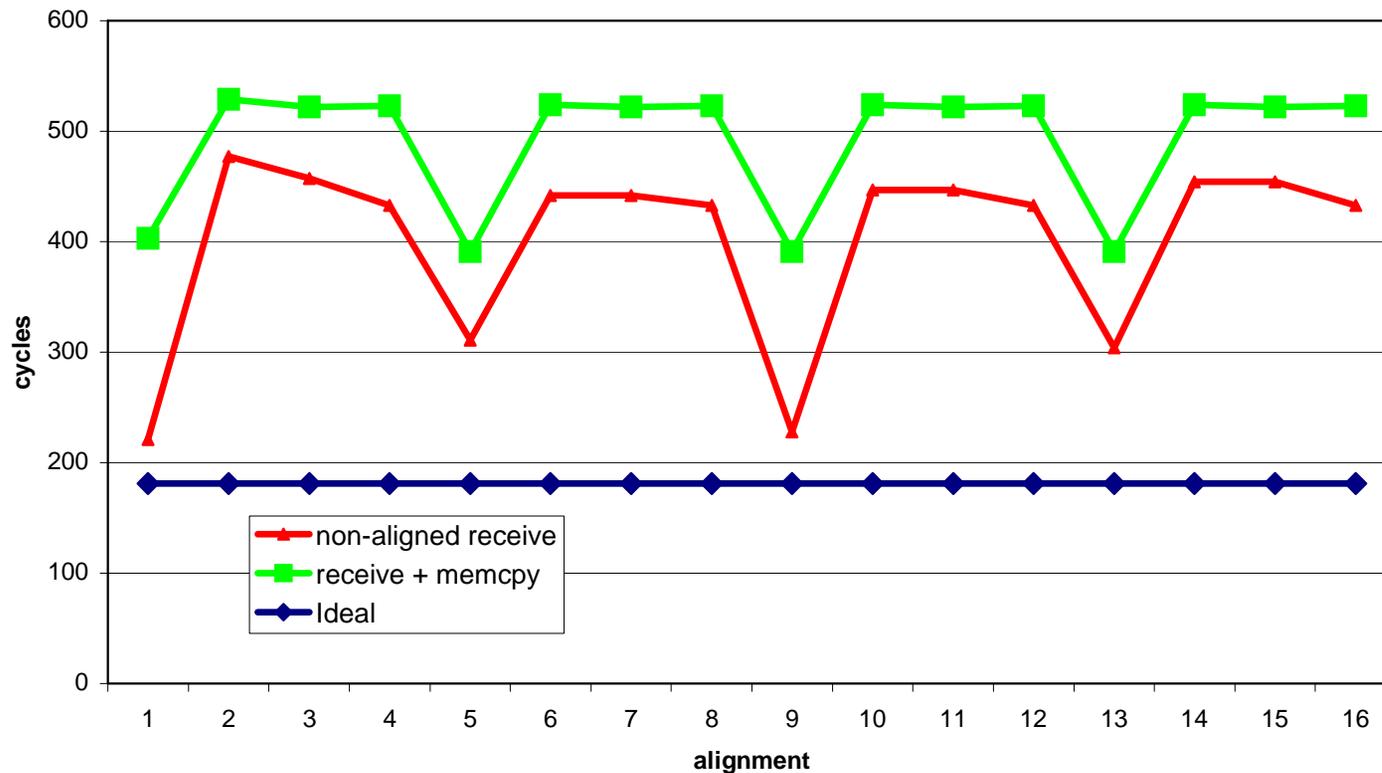
- head and tail transmitted in a single “unaligned” packet
- aligned packets go directly to/from torus FIFOs

- When alignments differ, extra memory copy is needed

- Sometimes torus read op. can be combined with re-alignment op.

The cost of packet re-alignment

- ❖ The cost (cycles) of reading a packet from the torus into un-aligned memory
- ❖ Receiver is responsible for re-alignment (e.g. eager protocol)



Out-of-order packet delivery on torus network

■ Constraint: routing on torus network

- ❖ **Deterministic:** ordered delivery, but prone to network bottlenecks
- ❖ **Adaptive:** good network behavior, but out-of-order packet delivery

■ MPI requires in-order matching of messages received from same host.

- ❖ **Only MPI matching information needs to be delivered in order.**

■ Rendezvous protocol:

- ❖ Packets belonging to message body use adaptive routing, can be unpacked in arbitrary order
- ❖ RTS packets use deterministic routing (so messages are matched in order)

■ Eager protocol, adaptive routing:

- Re-order messages via message numbers
- Temporary storage for packets that arrive early
- Include MPI matching info in every packet belonging to a message
- ❖ Lower bandwidth when traffic is high, because of high per-packet overhead

■ Eager protocol, deterministic routing:

- ❖ Lower per-packet overhead
- ❖ Potential of network bottlenecks



Using the Communication Co-processor

- **Constraint 1:** one CPU cannot keep up with network
- **Constraint 2:** BG/L chip has two non-coherent 440 cores
 - ❖ Original design point: second processor acts as an intelligent DMA engine (“co-processor mode”)
 - ❖ Initial software development done with 2nd processor in an idle loop (“heater mode”)
 - ❖ Considered: “virtual node mode” (2nd processor has its own O/S image and stack, shares all resources equally)
- Simple co-processor solution (1 extra memory copy):
 - ❖ CPU0 and CPU1 interact through common non-cached area (scratchpad)
 - ❖ Simple, but low performance
- Complex 0-copy solution:
 - ❖ Main CPU, coprocessor execute software cache coherency protocol
 - Sequences of cache flush and invalidate instructions
 - Need kernel support
 - Danger of false sharing
 - Complicated, fragile implementation (“heroic programming”)

Co-processor implementation, today (10/14/2003)

- Important because it allows
 - ❖ Overlapping communication and computation
 - ❖ Allows CPUs to keep up with torus network
 - “Simple” solution works, but has low performance
 - Torus read bandwidth: 1.2B/cycle
 - “scratchpad” read bandwidth: 2B/cycle for small (256B) packets
 - We expected 5 B/cycle.
 - Problem exacerbated by out-of-orderness of incoming packets
 - By eager protocol
 - And by careless programming
- We think that the complex solution will not suffer from performance problems.
 - ❖ Rendezvous protocol, combined with co-processor mode and partial packet method

“Partial” packets

- Would like to avoid unnecessary copies
 - ❖ Don't read the packet out of the torus until we know where the data go
- Packet header is necessary to determine data destination
 - ❖ Eager protocol: header contains identity of receiving message
 - ❖ Rendezvous protocol: header contains data buffer address
- Solution: partial packet
 - ❖ Contains the read-out first “chunk” of the packet
 - ❖ A read function can read the rest of the data
 - ❖ Also usable in co-processor mode
 - Read function is memory copy

Scaling problems:

How to crash BGL/MPI in two easy steps

- Torus routing gives pass-through packets preferential treatment
 - ❖ Local packets have a lower chance to get on the network
- In the program to the left, assume that 1 gets preferential treatment: sends much faster than rank 2
- There is no flow control for rank 1. It can send as fast as the network allows.
- Rank 0 is unable to post the receives for rank 1, because it is waiting for rank 2
- All rank 1's messages will be unexpected in rank 0.
- Rank 0 **runs out of memory**.

```
For (I=0;I<1000;I++) {  
  If (rank==0) {  
    MPI_Recv(..., 1, );  
    MPI_Recv(..., 2, );  
  } Else if (rank<=2) {  
    MPI_Send (..., 0, );  
  }  
}
```

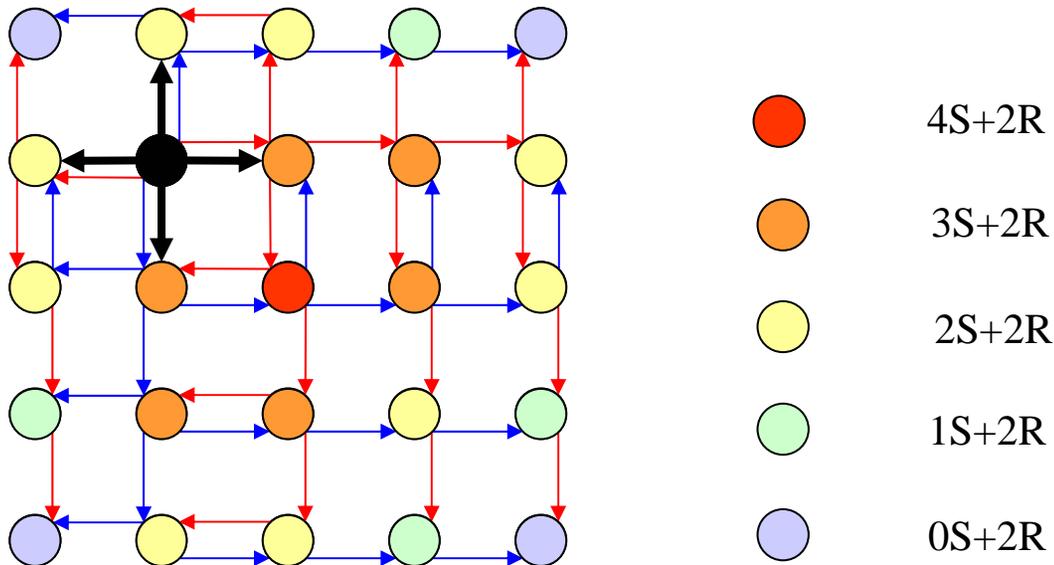
- **Flow control:**
 - Connections own tokens
 - Receiver grants tokens based on traffic
 - Token grants are packets
- Introduces latency, safety

Optimizing Collective Operations

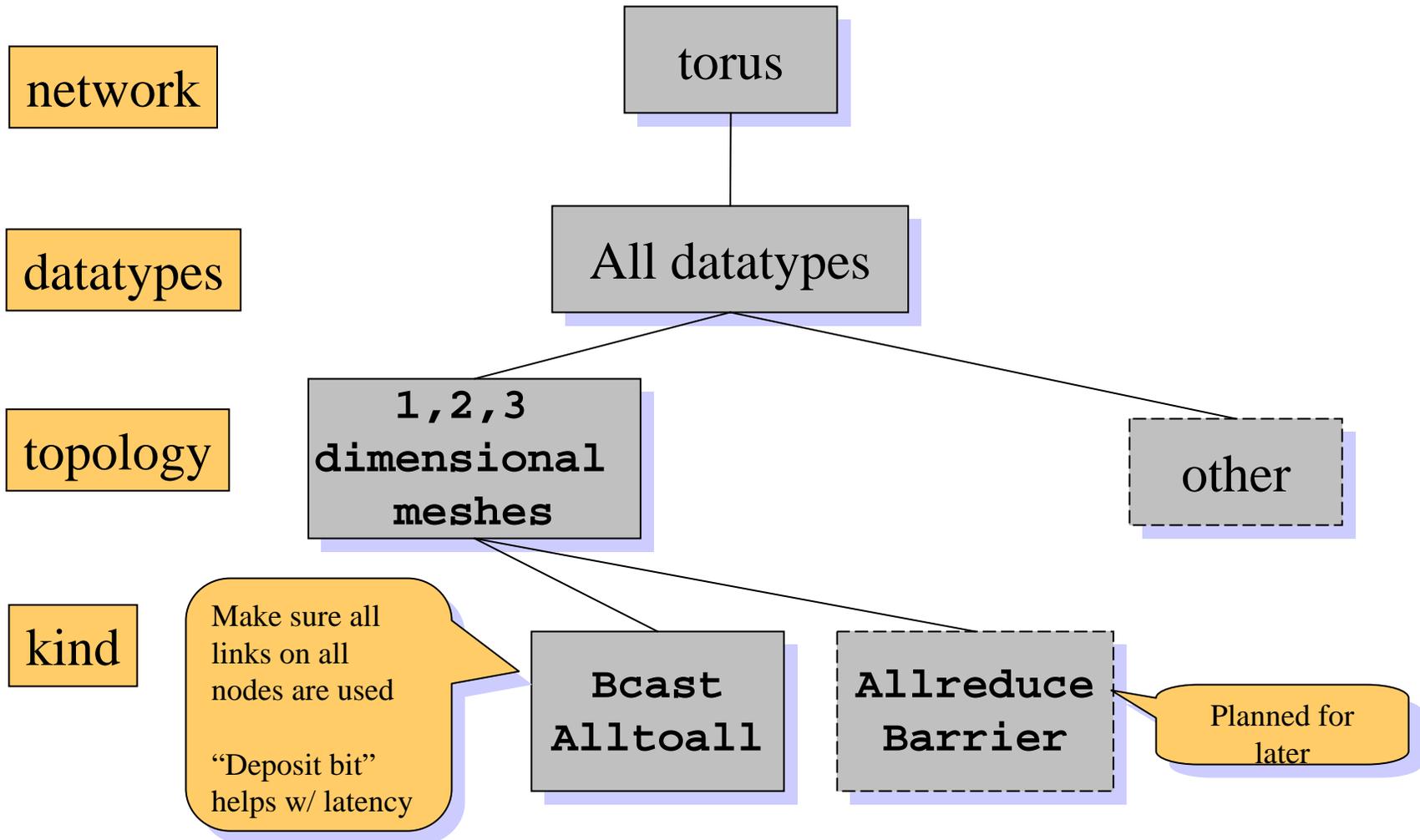
- MPICH2 comes with default collective algorithms
 - Bcast: MST or scatter/allgather
 - Alltoall: recursive dbl., pairwise exchanges
 - Alltoallv: post & waitall
 - Scatter: MST
- Default algorithms not suitable for torus topology
 - Designed for ethernet, switched (crossbar) environments
 - E.g. a good plane broadcast algorithm uses the four available links of a node to the maximum
- Taxonomy of possible optimizations

Red-blue broadcast on a mesh

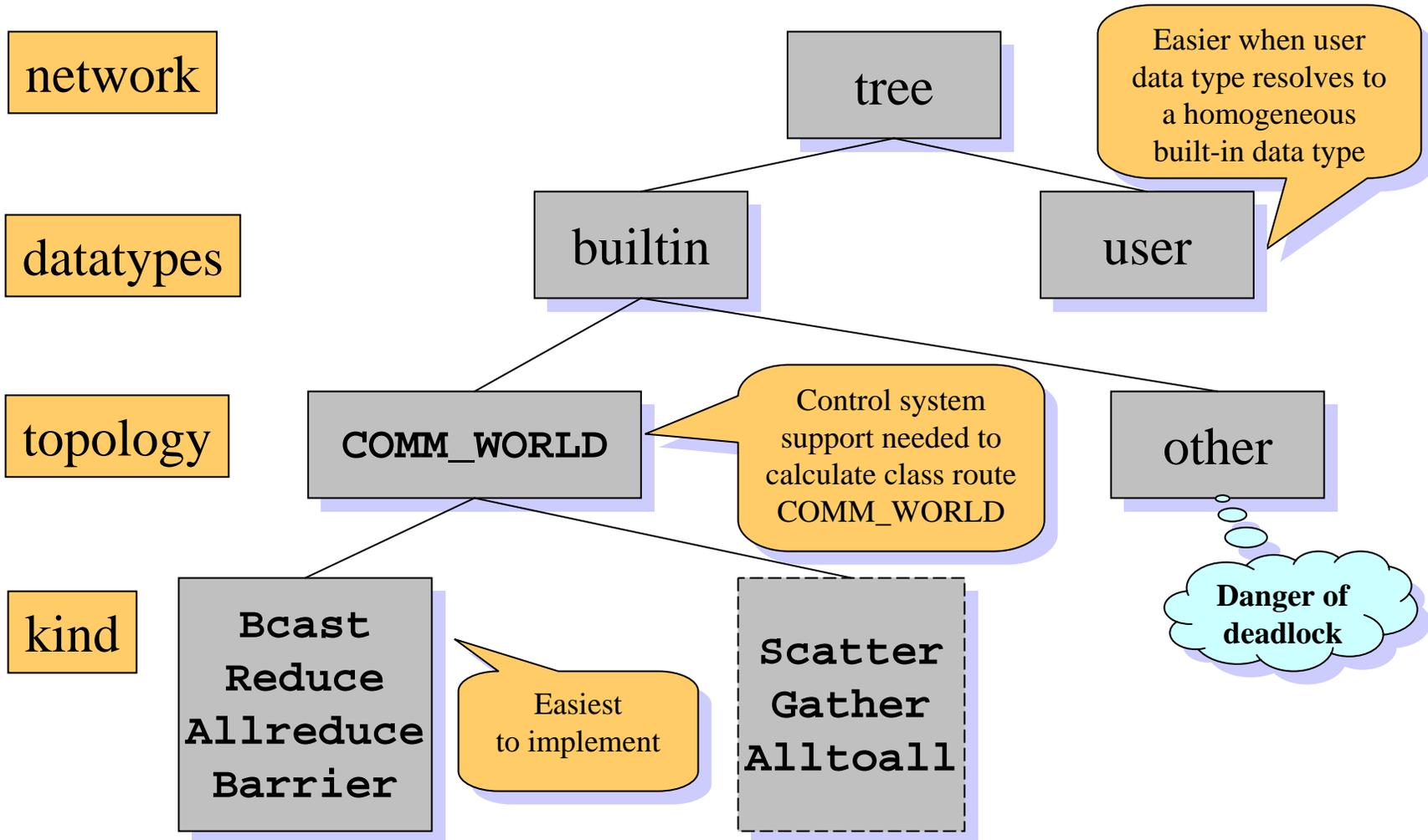
Vernon Austel, John Gunnels, Phil Heidelberger, Nils Smeds



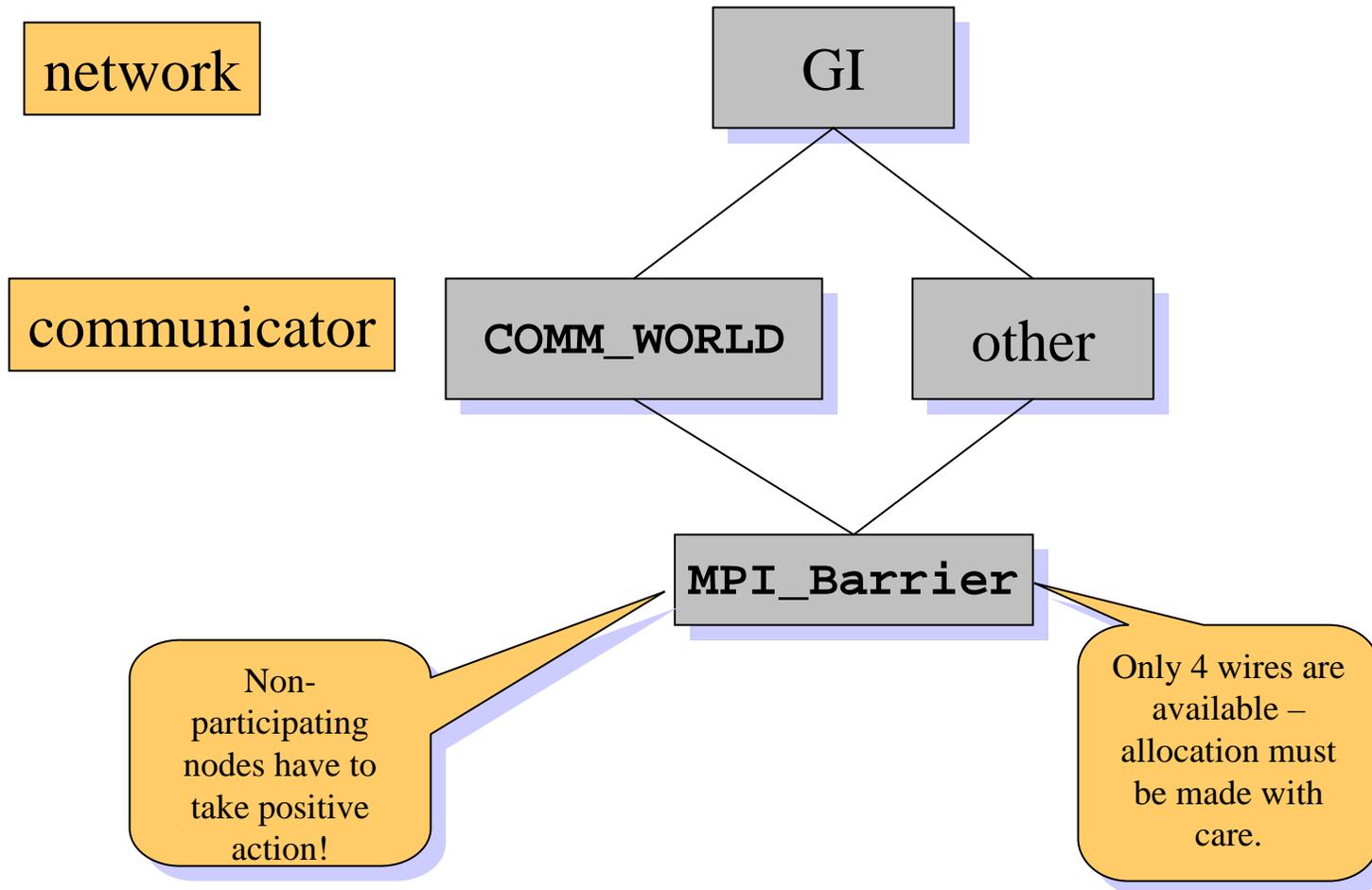
Implementing collectives on the torus network



Implementing collectives on the tree network



Global Interrupts



Process Management in BGL/MPI

■ Process startup and termination

- ❖ Implemented using the BG/L CIO protocol
 - **ciorun** asks control system to start up job
 - Control system contacts CIO daemons residing on 1024 I/O nodes
 - CIO daemons issue commands to 64 compute nodes through tree network
- ❖ Does not (and will not) support dynamic MPI process creation
- ❖ Work in progress: integration with scheduler

■ Mapping of torus coordinates to MPI ranks

- ❖ Today: fixed torus rank mapping can be selected through environment variables at startup
- ❖ Work in progress: arbitrary mapping function provided at job startup time
- ❖ MPI programs are topology portable; MPI performance is not

Performance: Bandwidth and Latency targets

- HW latency: 2.5 μ s (worst case)

- **MPI latency target: 5 μ s**

- HW Bandwidth:

 - ❖ Theoretical peak per link:

 - $B_{TL} = 0.25$ Bytes/cycle

 - ❖ Theoretical peak per node:

 - 12 links (6 snd + 6 rcv)

 - $B_{TP} = 12 B_{TL} = 3$ B/cycle

 - 2100 MB/s @ 700MHz

- **MPI Bandwidth target:**

 - ❖ **240 of 272 bytes payload:**

 - ❖ $B_{MP} = 0.882 B_{TP}$

 - ❖ $B_{MP} = 2.2$ Bytes/cycle

 - ❖ **1850 MB/s @ 700MHz**

- **MPI 6-way send bandwidth:**

 - ❖ $B_{MS} = 0.5 B_{MP}$

 - ❖ $B_{MS} = 1.1$ Bytes/cycle

 - ❖ **926 MB/s @ 700 MHz**

- **MPI 6-way receive bandwidth:**

 - ❖ $B_{MR} = 0.5 B_{MP}$

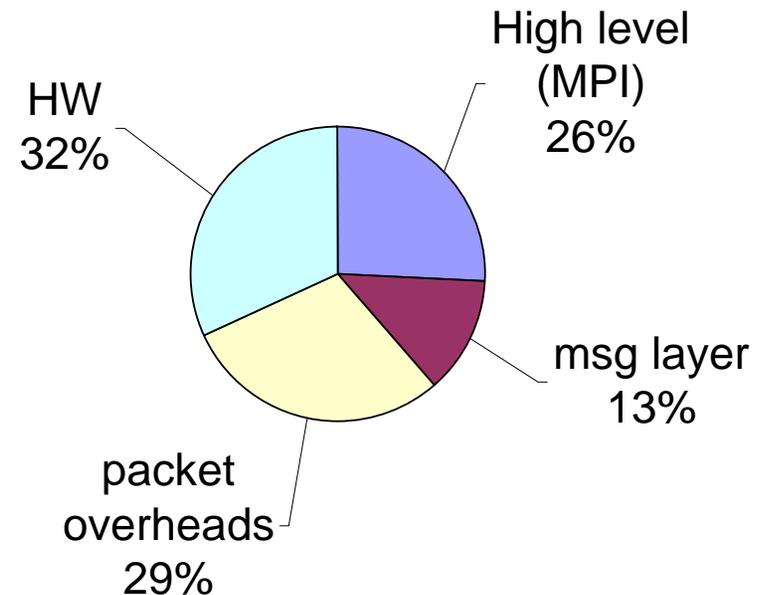
 - ❖ $B_{MS} = 1.1$ Bytes/cycle

 - ❖ **926 MB/s @ 700 MHz**

BGL/MPI Latency (Oct. 2003)

- $\frac{1}{2}$ roundtrip latency: \approx **3000 cycles**
 - ❖ About 6 μ s @ 500MHz
- Measured:
 - ❖ With Dave Turner's mpipong
 - ❖ In heater mode; bound to increase a bit in co-processor mode
 - ❖ Using Nearest neighbors: HW latency is only about 1200 cycles
- Constant up to 192 bytes payload
 - ❖ Single packet

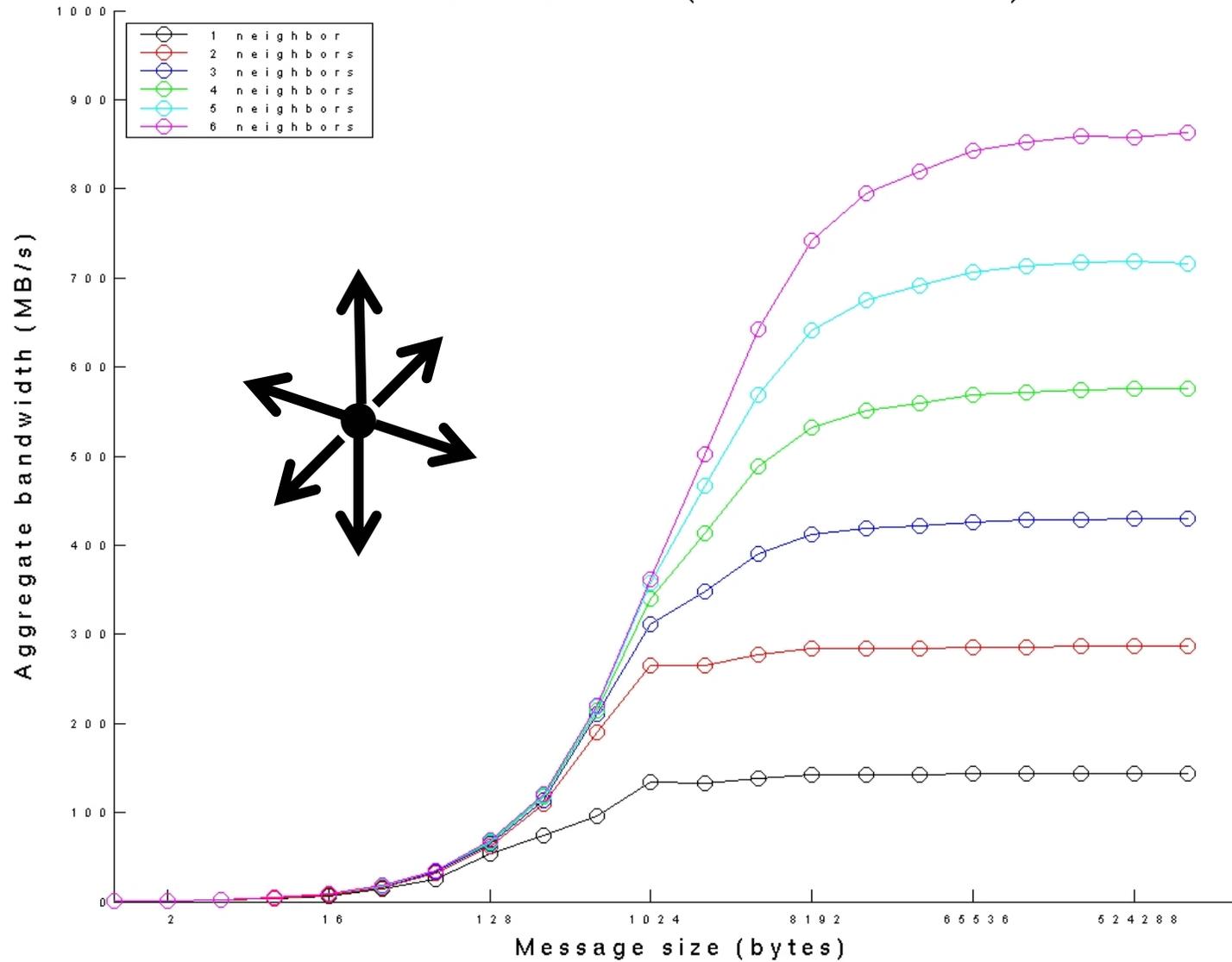
- Composition of roundtrip latency:



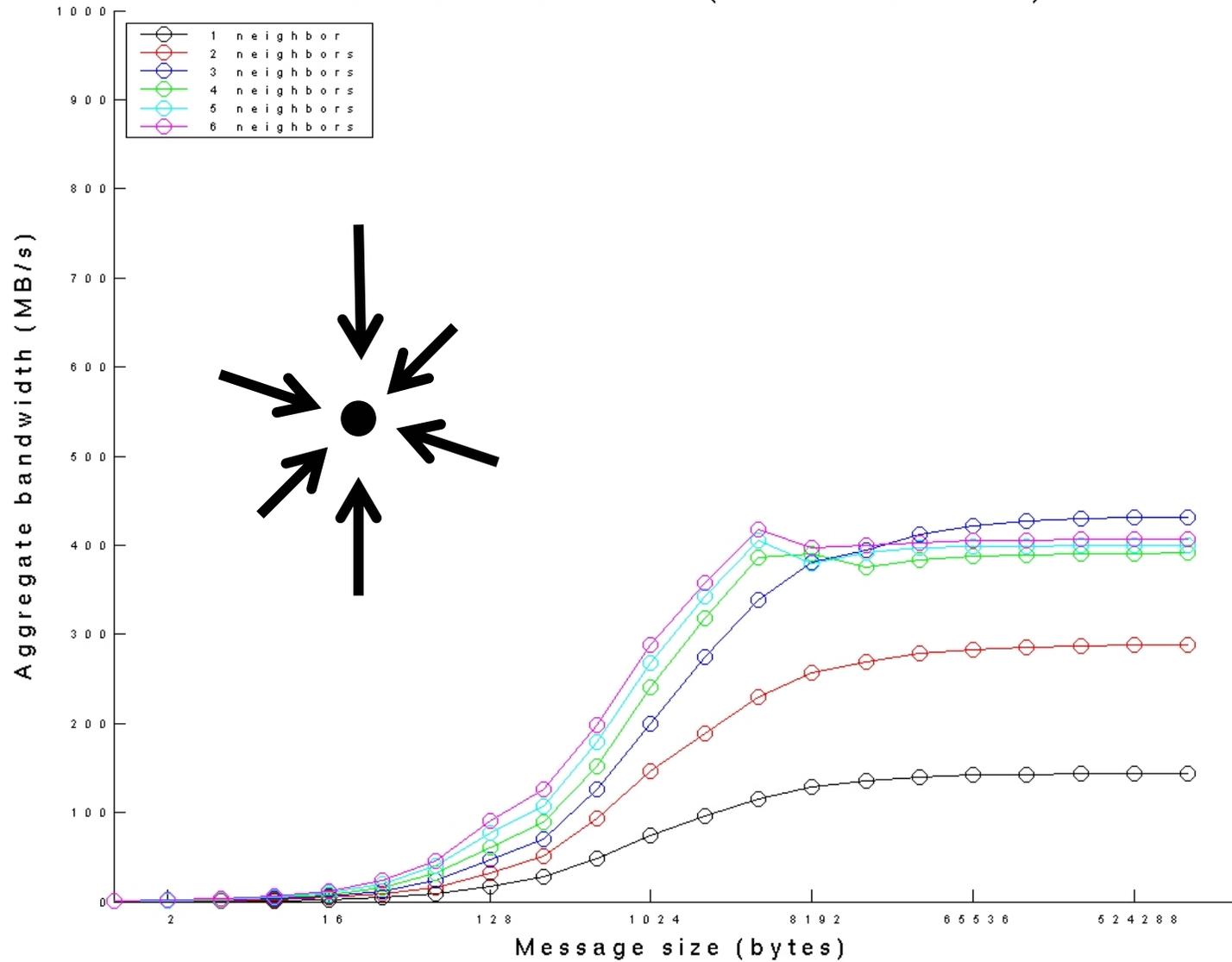
BGL/MPI Bandwidth (Oct. 2003)

- On this machine, good bandwidth is harder to achieve than good latency.
 - ❖ Per-packet overhead
- Bandwidth:
 - ❖ Measured with a custom made program that sends nearest neighbor messages
 - ❖ Heater mode
 - ❖ Eager protocol – suboptimally implemented (224 byte packet payload instead of 240)
 - ❖ Max bandwidth = $0.823 * BTP$
 - ❖ (864 MBytes/s send, receive)
- Torus packet writes:
 - ❖ 60 cycles/256 byte packet:
 - ❖ 4.26 Bytes/cycle
 - ❖ Bandwidth limited by torus
 - ❖ (1.5 B/cycle)
- Torus packet reads:
 - ❖ 204 cycles/256 byte packet
 - ❖ 1.2 B/cycle
 - ❖ Bandwidth limited by CPU
- MPI packet reads (eager protocol)
 - ❖ 350 cycles/256 byte packet
 - ❖ Limited to 0.731 B/cycle by CPU
 - ❖ Only about 3 FIFOs worth

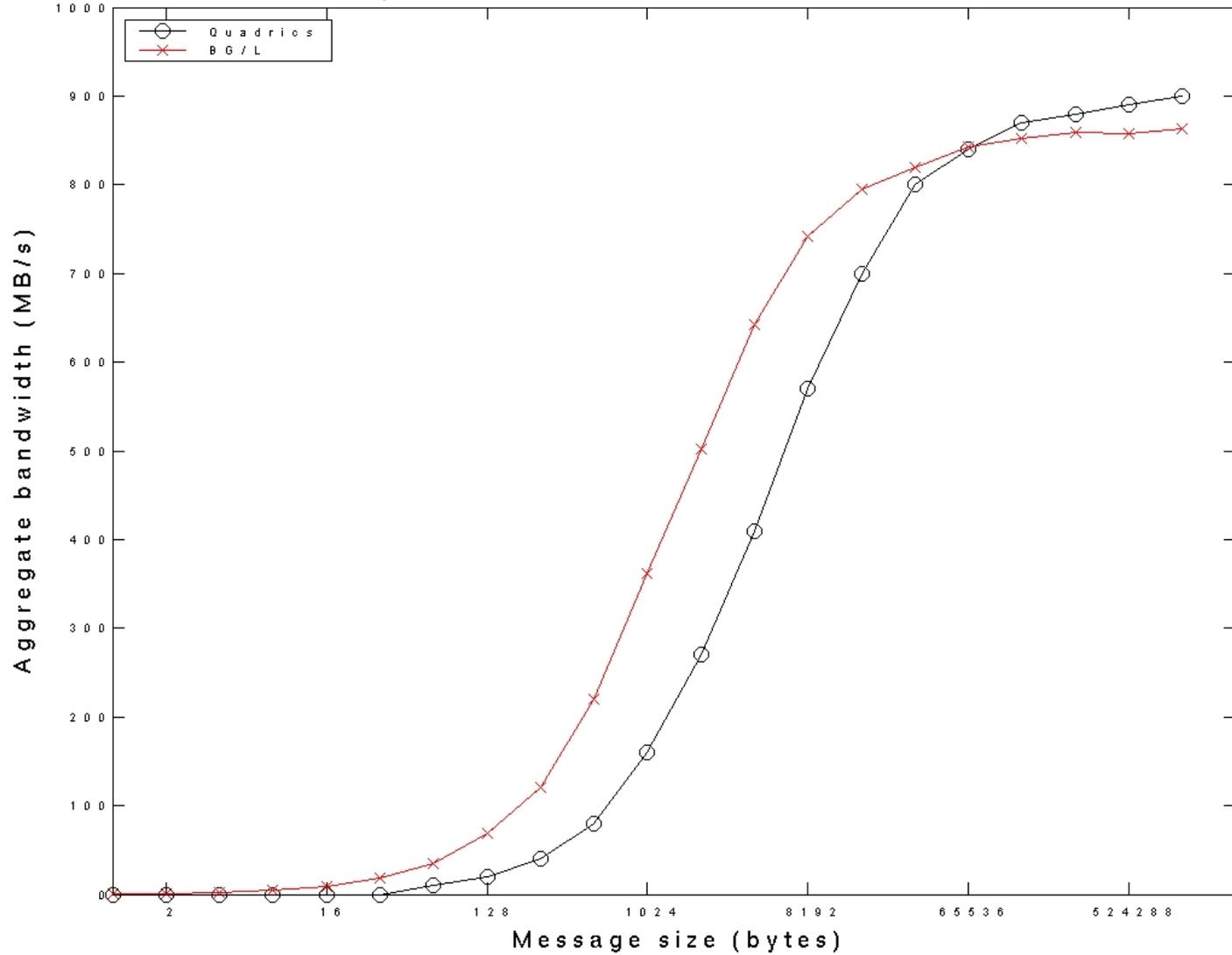
BG/L send bandwidth (700 MHz CPU clock)



BG/L receive bandwidth (700 MHz CPU clock)



Comparison of Quadrics and BG/L bandwidths



Lessons learned during implementation

■ What we thought would happen

- ❖ Packet layer would need no changes
- ❖ Performance will be influenced by message start overhead
- ❖ We will handle out-of-order eager packets
- ❖ Co-processor mode would improve performance quickly
- ❖ Heater mode would have low performance

- ❖ All kinds of low-level optimizations would be needed for collectives

■ What really happened

- ❖ Packet layer had to be re-written almost from scratch
- ❖ Performance was influenced by per-packet overhead
- ❖ Adaptive routing only used for rendezvous protocol
- ❖ Co-processor mode has performance problems
- ❖ Heater mode provides adequate performance, making virtual node mode a viable option

- ❖ Collectives can be implemented using standard pt-2-pt messages, if hardware topology is taken into account



Conclusion

- MPICH2 point-to-point communication is almost MPI-1 compliant
 - ❖ NAS parallel benchmarks ported, run, measured
 - ❖ LLNL, IBM ported and ran several ASCI Purple benchmarks
 - sPPM, sweep3d, UMT2K, SMG2K, DD3D
 - ❖ LANL ported and ran SAGE in a single day
 - ❖ Watson developing high-performance Linpack application
- .Ongoing work in:
 - ❖ Process management primitives
 - ❖ Topology aware collective operations
 - ❖ Functional correctness (sync. send, Cancel, non-contiguous data types)
 - ❖ Improving point-to-point performance:
 - Deploying co-processor mode
 - Deploying rendezvous protocol