

# Communication Support for the Ultra-Scale

**Richard L. Graham**  
**Computer Science & Mathematic Division**  
**Oak Ridge National Laboratory**

# Contributors to the effort

- **Pavel Shamis**
- **Ishai Rabinovitz**
- **Gil Bloch**
- **Noam Bloch**
- **Hillel Chapman**
- **Michael Kagan**
- **Stephen Poole**
- **Ariel Shahar**
- **Gilad Shainer**

# Acknowledgements

- US Department of Energy FASTOS program
- HPC Advisory Council (computer resources)
  - [www.hpcadvisorycouncil.com](http://www.hpcadvisorycouncil.com)

# Outline

- **Problems being addressed**
- **InfiniBand overview**
- **New InfiniBand capabilities**
- **Software design for collective operations**
- **Results**
- **Next steps**

# Problems Being Addressed – Collective Operations

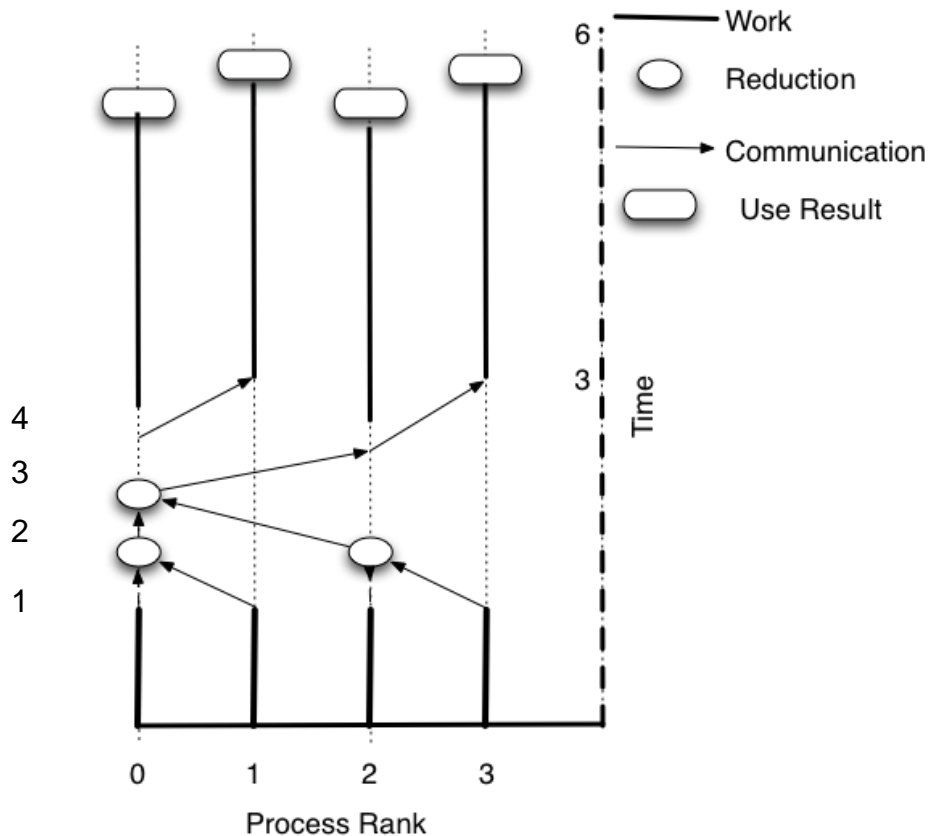
- **Communication characteristics at scale**
- **Overlapping computation with communication – true asynchronous communications**
  - **Goal: Avoid using the CPU for communication processing**
- **System noise**
- **Application skew**
- ➔ **Scalability**
- **Collective communication performance**

# Collective Communications

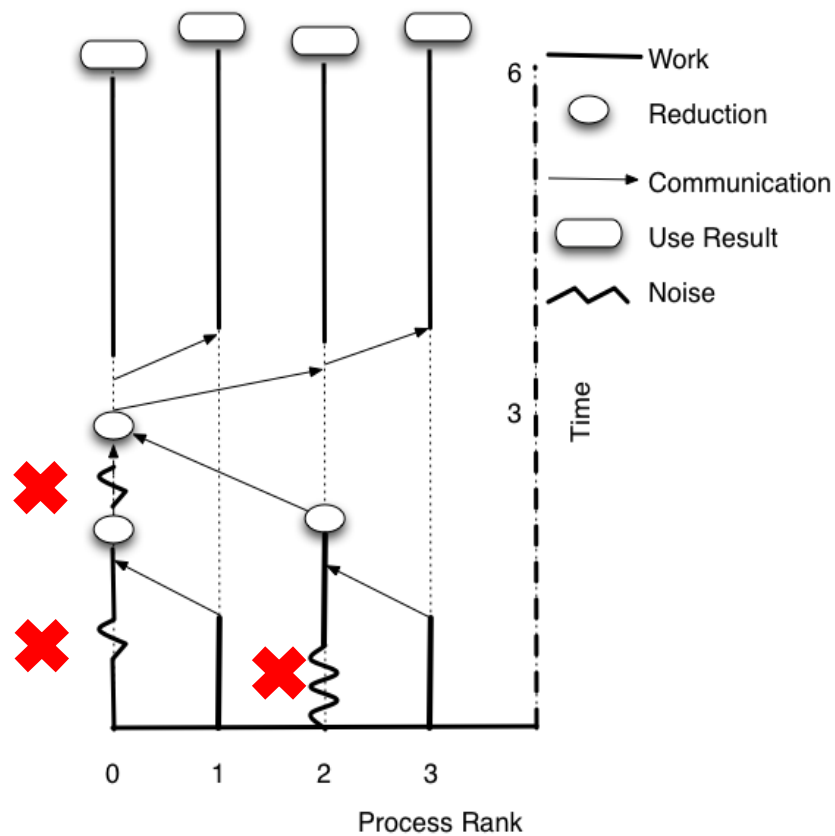
- **Communication pattern involving multiple processes (in MPI, all ranks in the communicator are involved)**
- **Optimized collectives involve a communicator-wide data-dependent communication pattern**
- **Data needs to be manipulated at intermediate stages of a collective operation**
- **Collective operations limit application scalability**
- **Collective operations magnify the effects of system-noise**

# Scalability of Collective Operations

## Ideal Algorithm



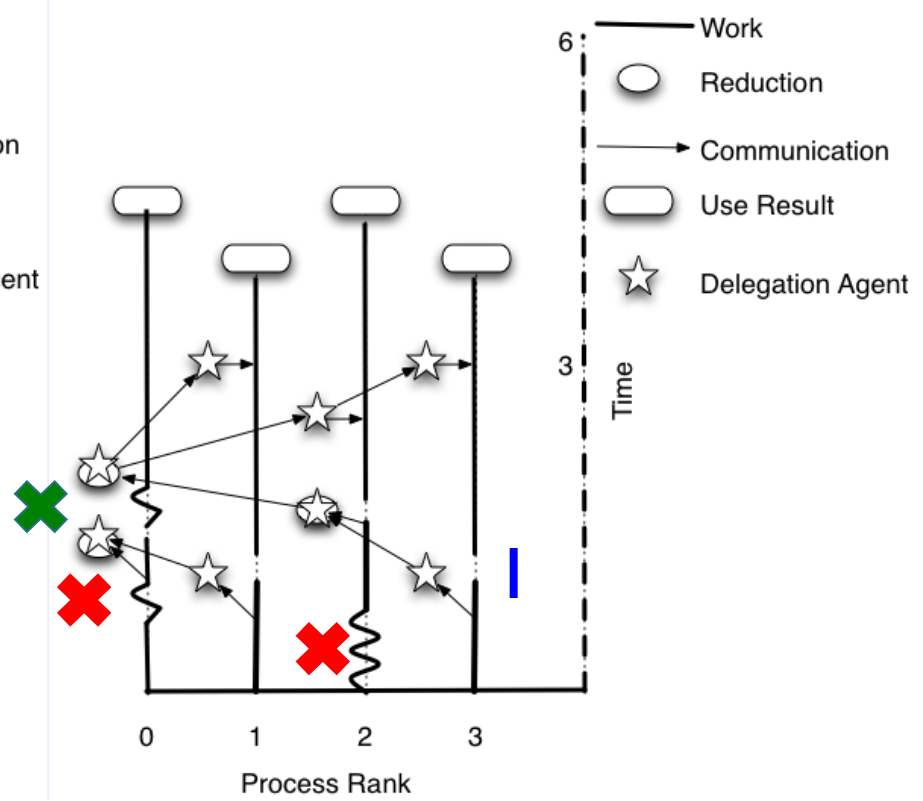
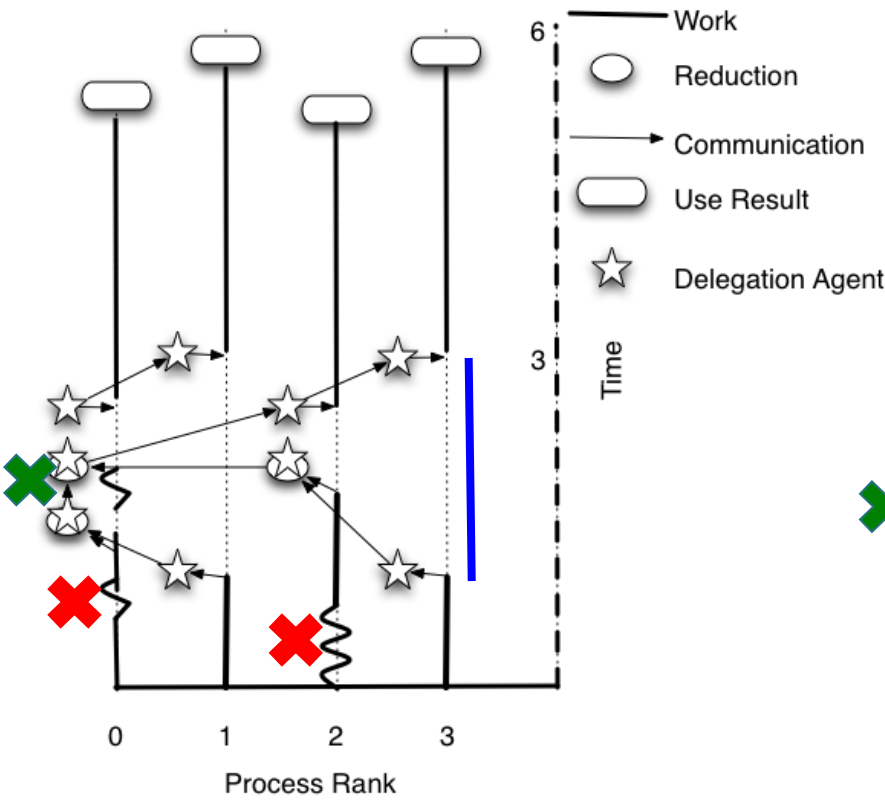
## Impact of System Noise



# Scalability of Collective Operations - II

Offloaded Algorithm

Nonblocking Algorithm



- Communication processing

# Approach to solving the problem

- **Co-design**

- **Network stack design (Mellanox)**
- **Hardware development (Mellanox)**
- **Application level requirement (ORNL)**
- **MPI/Shmem level implementation (Joint)**

# InfiniBand Collective Offload – Key idea

- **Create local description of the communication patterns**
- **Hand the description to the HCA**
- **Manage collective communications at the network level**
- **Poll for collective completion**
- **Add new support for**
  - **Synchronization primitives (hardware)**
    - **Send Enable task**
    - **Receive Enable task**
    - **Wait task**
  - **Multiple Work Request**
    - **A sequence of network tasks**
  - **Management Queue**

# InfiniBand Hardware Changes

- **Tasks defined in the current standard**
  - **Send**
  - **Receive**
  - **Read**
  - **Write**
  - **Atomic**
- **New support**
  - **Synchronization primitives (hardware)**
    - **Send Enable task**
    - **Receive Enable task**
    - **Wait task**
  - **Multiple Work Request**
    - **A sequence of network tasks**
  - **Management Queue**

# MPI Queue Design

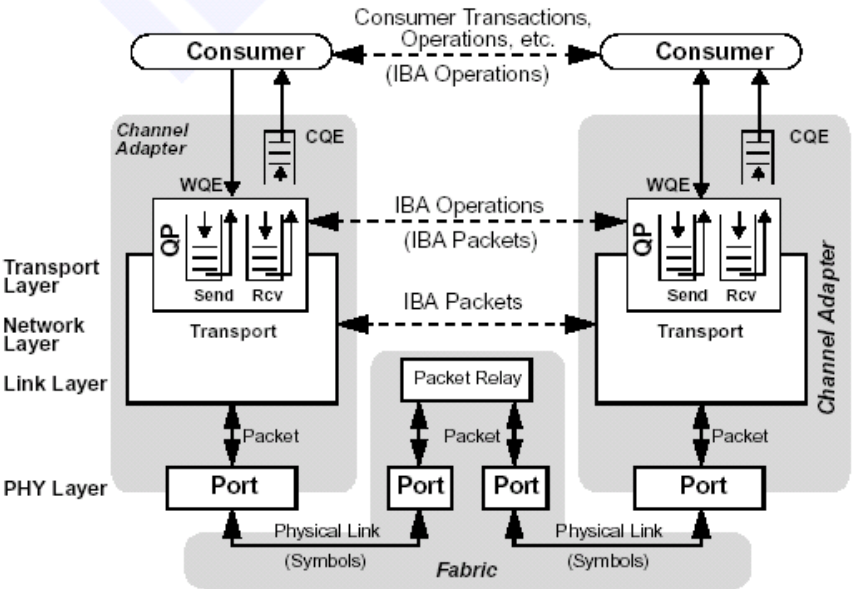
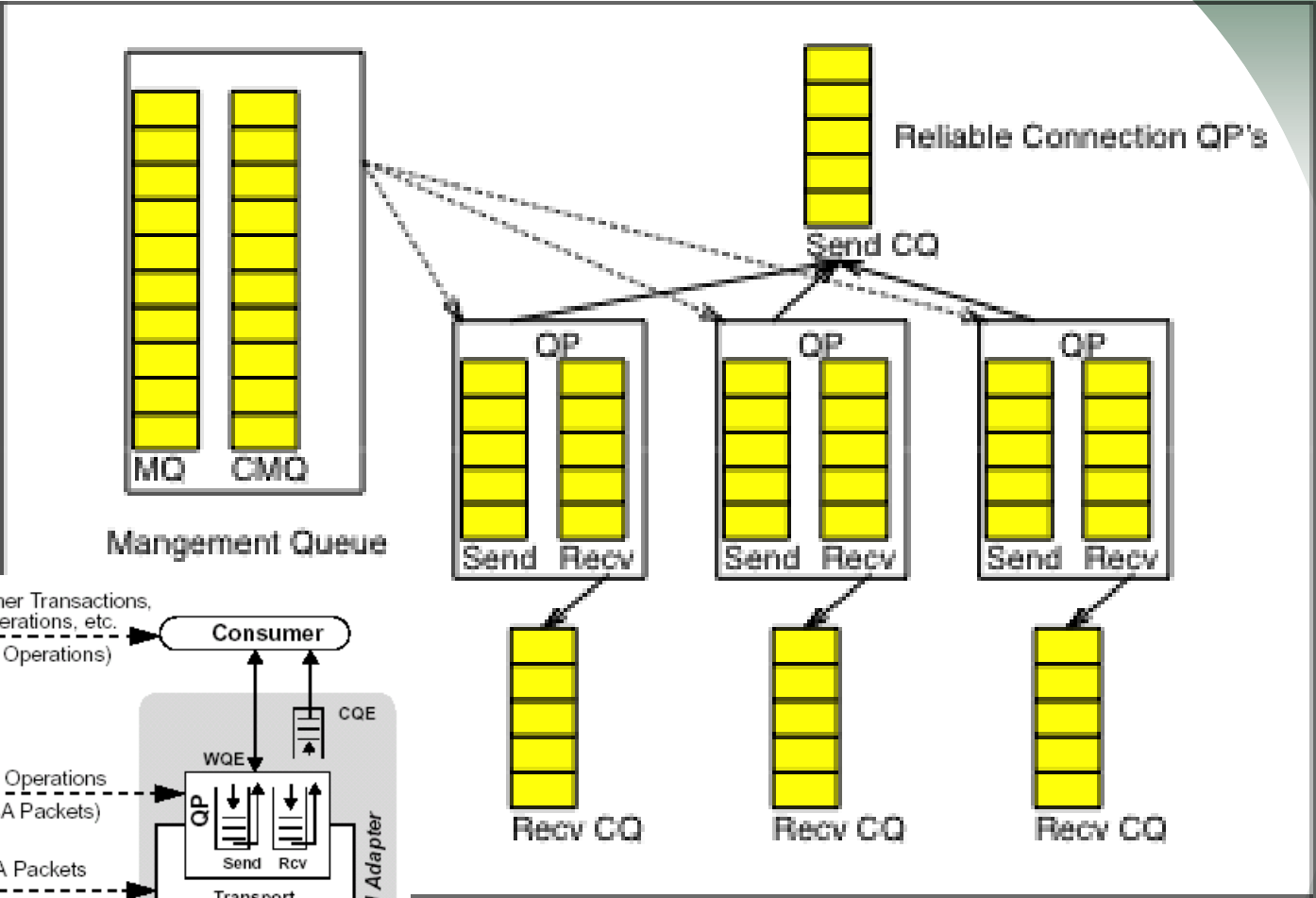
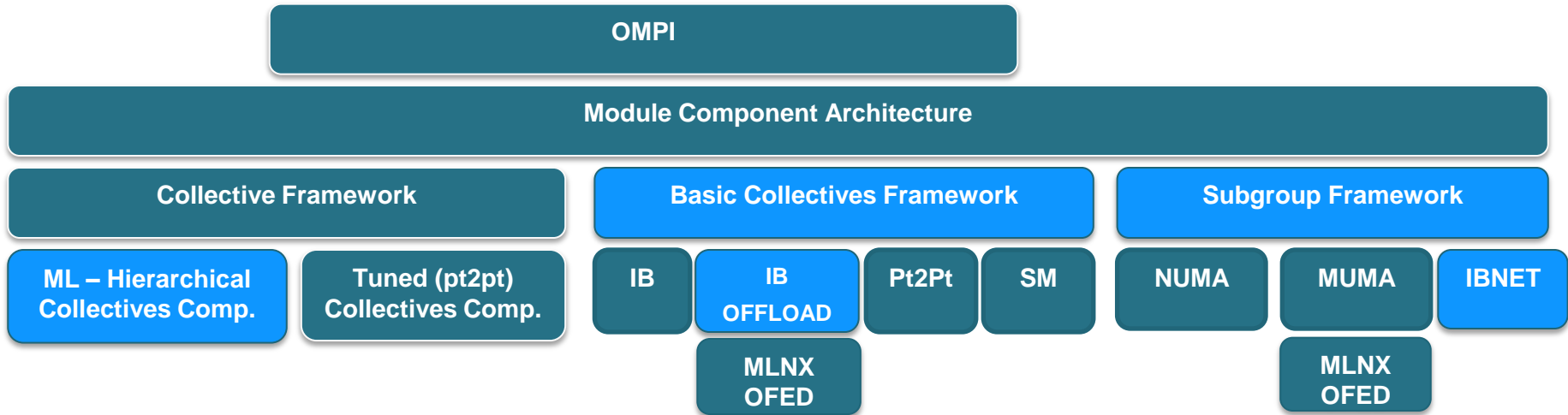
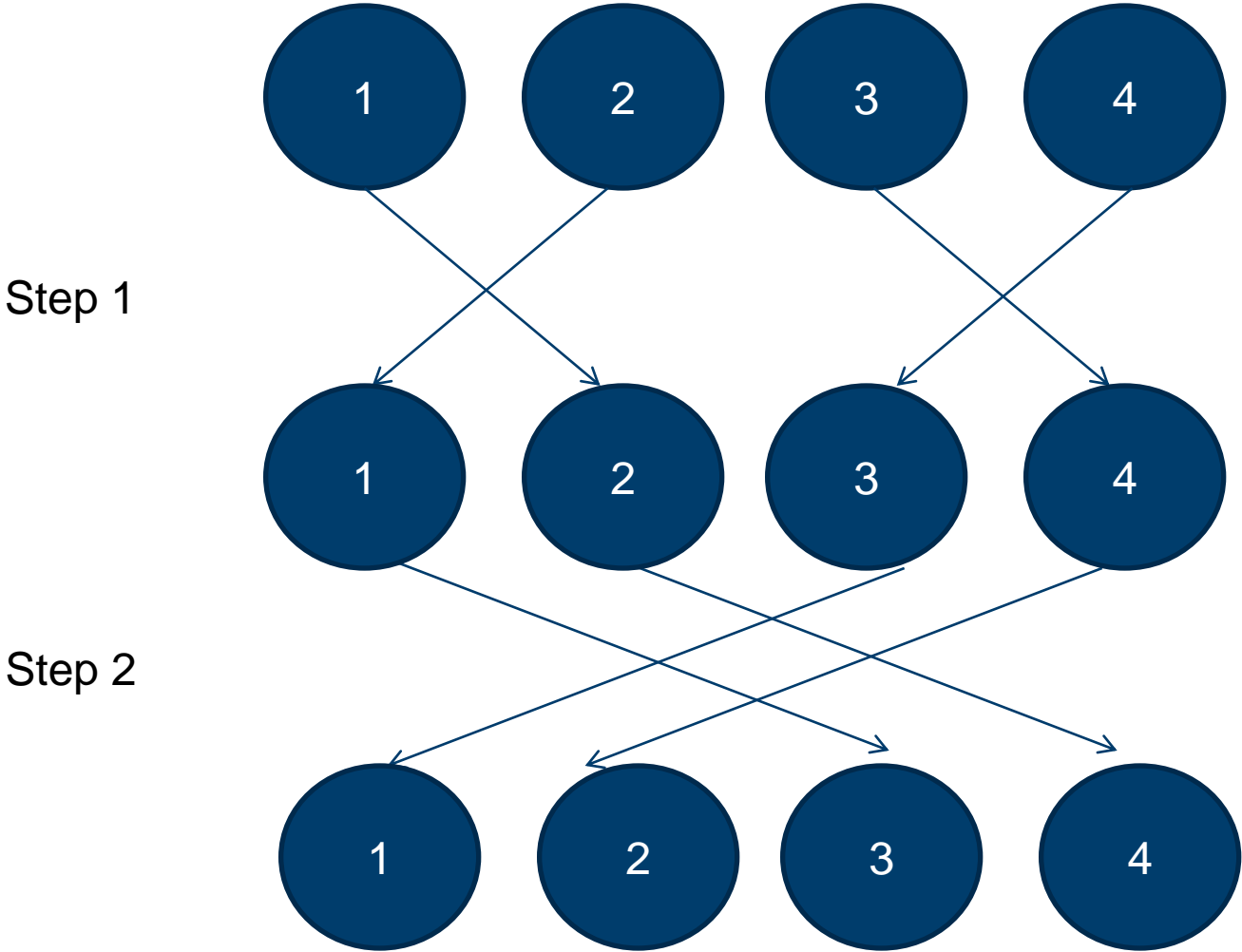


Figure 13 IBA Communication Stack

# Collectives – Software Layers



# Example – 4 Process Recursive Doubling



# 4 Process Barrier Example

Algorithm

Proc 0	Proc 1	Proc 2	Proc 3
Exchange With proc 1	Exchange With proc 0	Exchange With proc 3	Exchange With proc 2
Exchange With proc 2	Exchange With proc 3	Exchange With proc 0	Exchange With proc 1

MWR

Proc 0	Proc 1	Proc 2	Proc 3
Send to proc 1	Send to proc 0	Send to proc 3	Send to proc 2
Wait on recv from 1	Wait on recv From 0	Wait on recv From 3	Wait on recv From 2
Send to proc 2	Send to proc 3	Send to proc 0	Send to proc 1
Wait on recv from 2	Wait on recv From 3	Wait on recv From 0	Wait on recv From 1

# 4 Process Barrier Example – Queue view

Send QP

Proc 0	Proc 1	Proc 2	Proc 3
Send to proc 1 - enabled	Send to proc 0 – enabled	Send to proc 3 - enabled	Send to proc 2 - enabled
Send to 2 – not enabled	Send to 3 – not enabled	Send to 0 – not enabled	Send to 1 – not enabled

MQ

Proc 0	Proc 1	Proc 2	Proc 3
Recv wait from 1	Recv wait from 0	Recv wait from 3	Recv wait from 2
Send enable 1	Send enable 0	Send enable 3	Send enable 2
Recv wait from 2	Recv wait from 3	Recv wait from 0	Recv wait from 1

# 8 Process Barrier Example – Queue view – no MQ, View at rank 0

QP 1	QP 2	QP 4
Send QP 1	Wait QP 1	Wait QP 1
	Send QP 2	Wait QP 2
		Send QP 4
		Wait QP 4

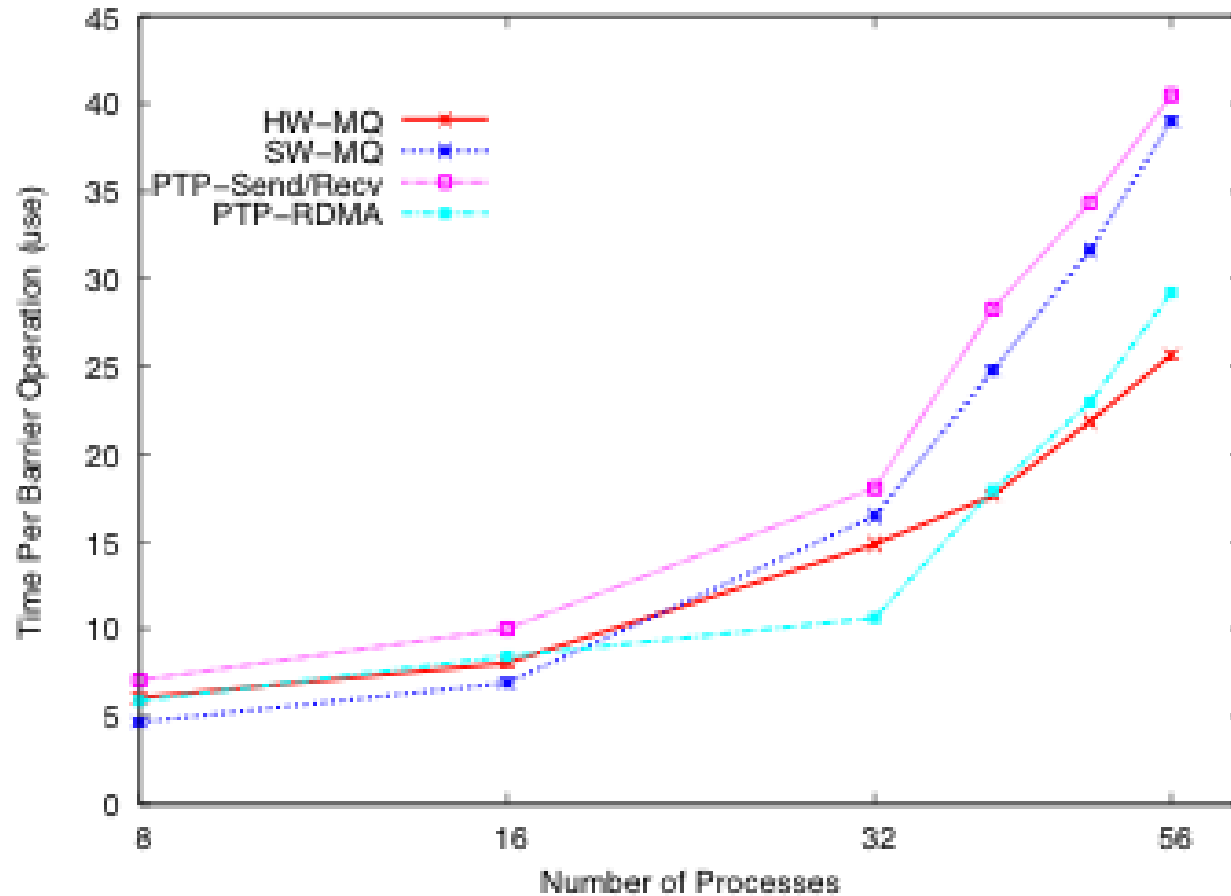
# Benchmarks

# System setup

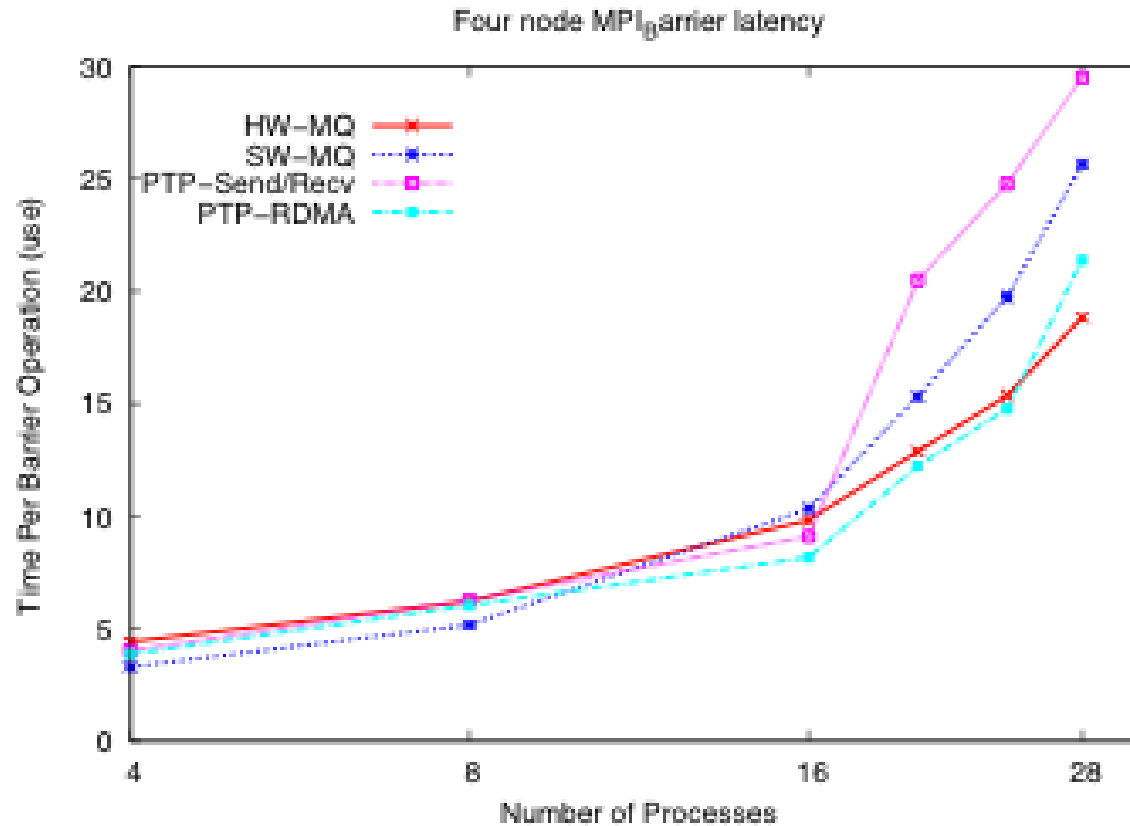
- **8 node cluster**
- **Node Architecture**
  - 3 GHz Intel Xeon
  - Dual socket
  - Quad core
- **Network**
  - ConnexX-2 HCA
  - 36 port QDR switch running pre-release firmware

# Barrier Data

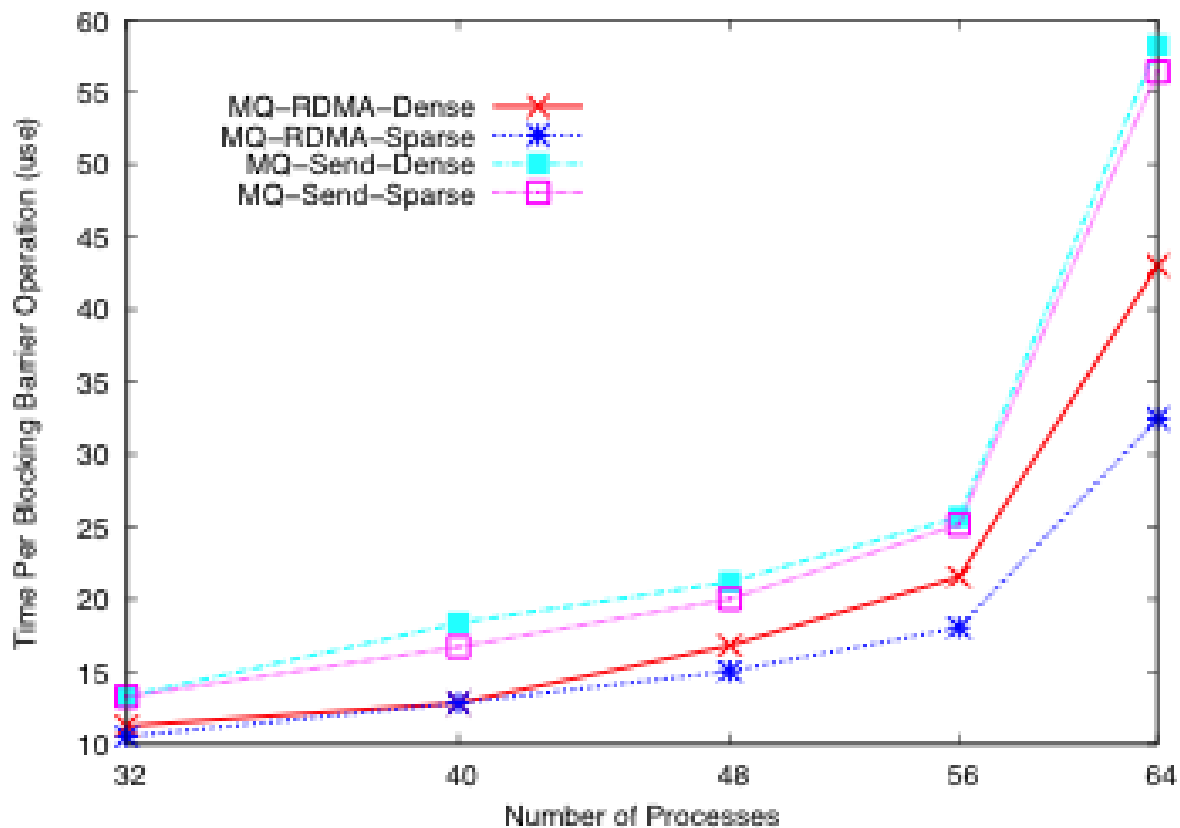
# 8 Node Blocking MPI Barrier



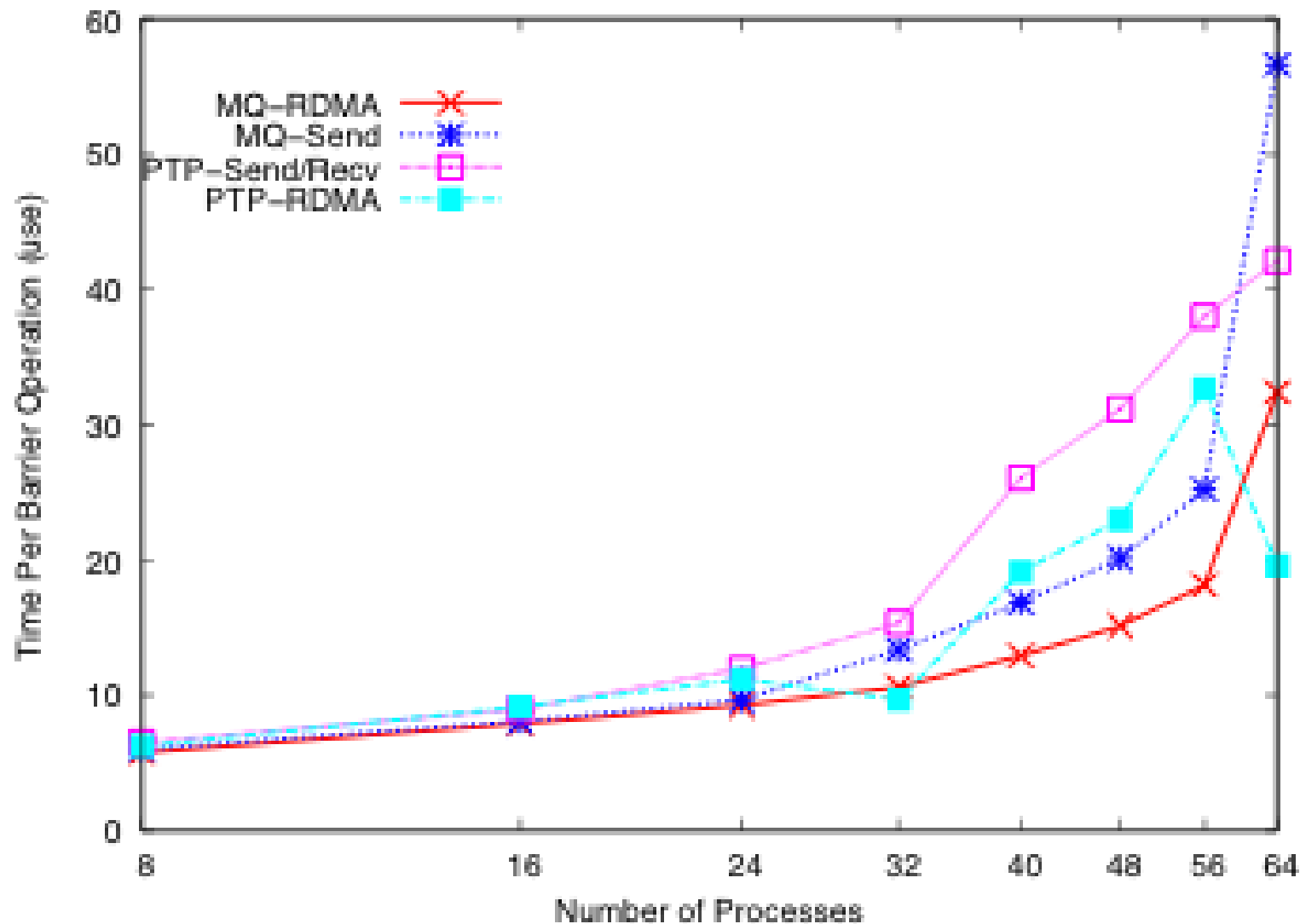
# 4 Node Blocking MPI Barrier



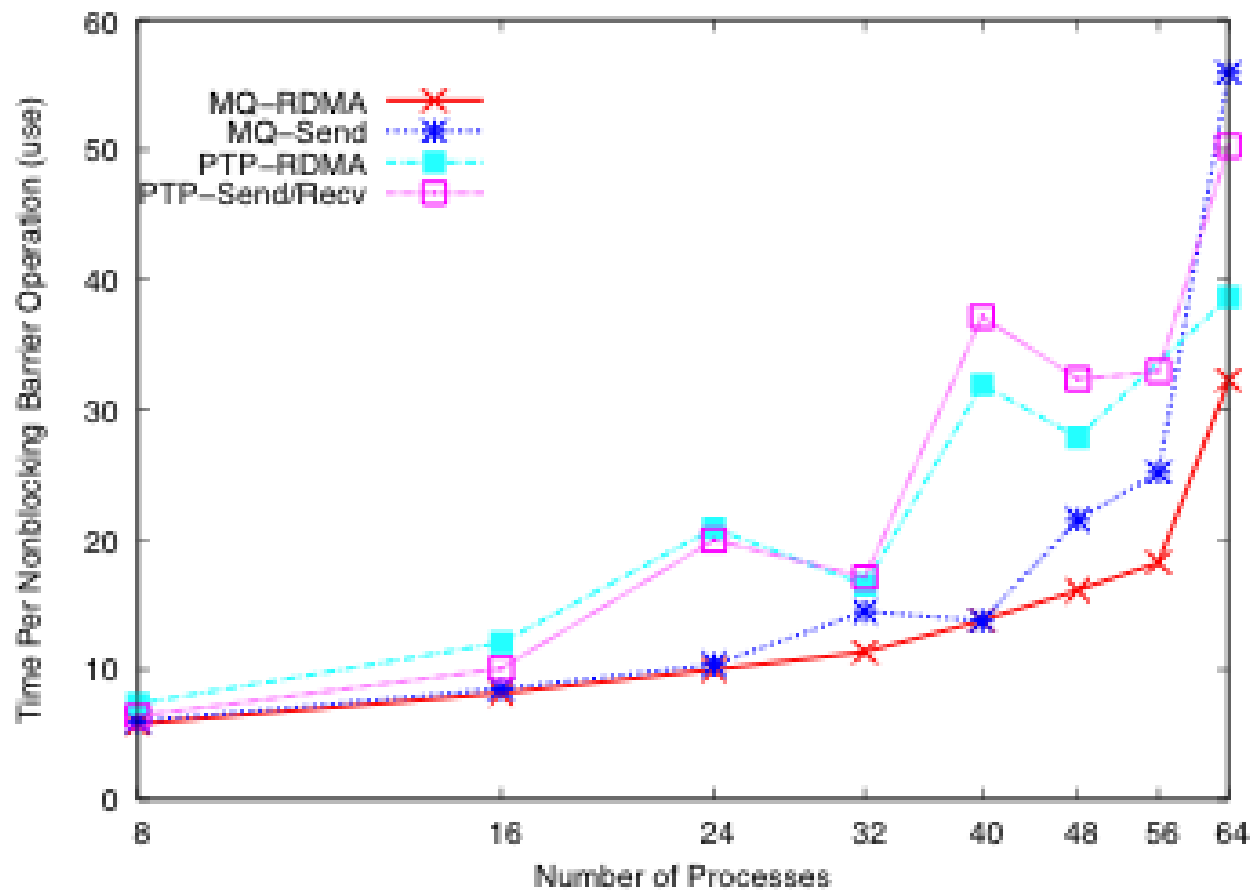
# MPI Barrier - Offloaded



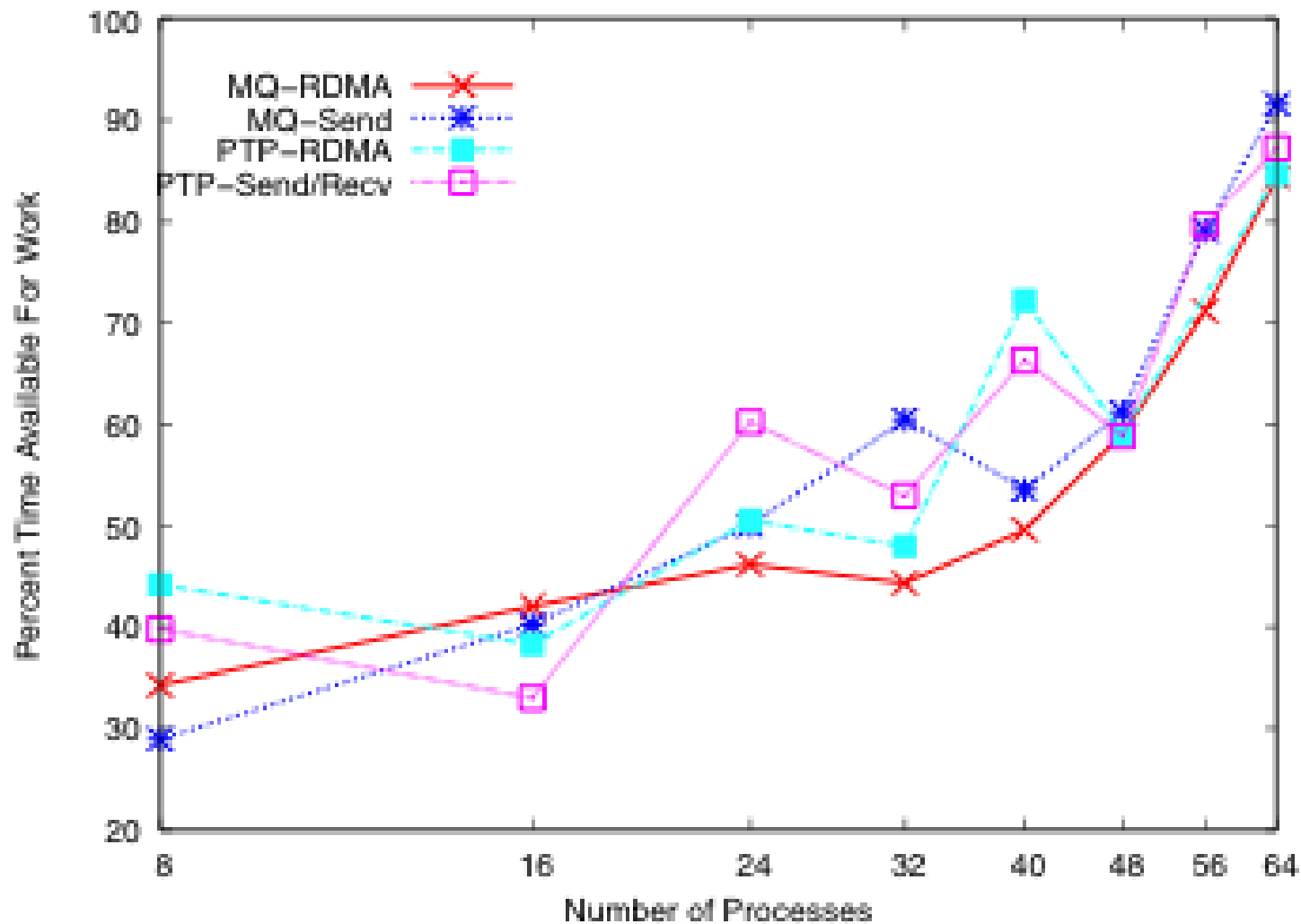
# MPI Barrier – Comparison with PtP



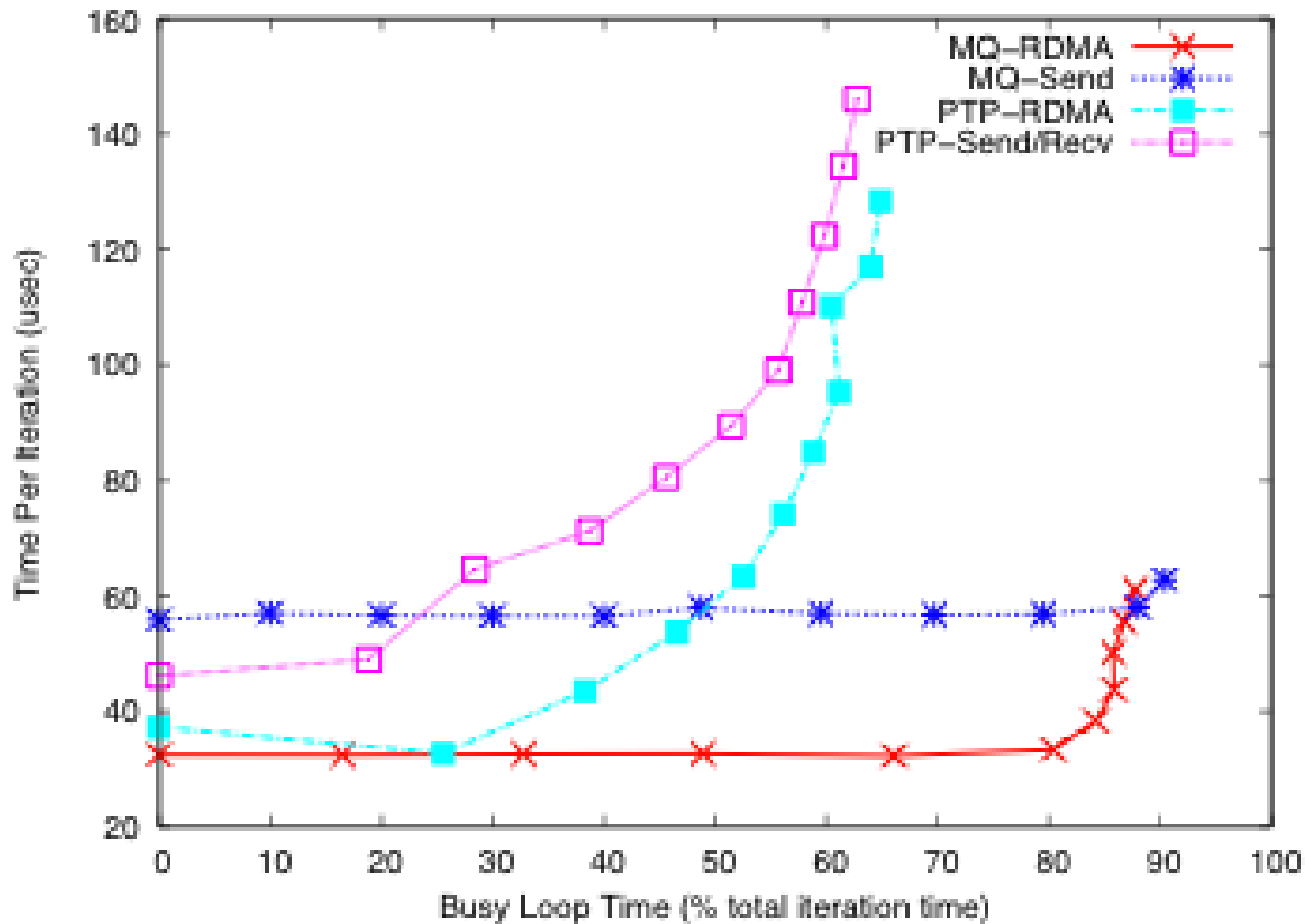
# MPIX\_Ibarrier Performance



# Nonblocking Barrier – Overlap – Multiple Work Quanta

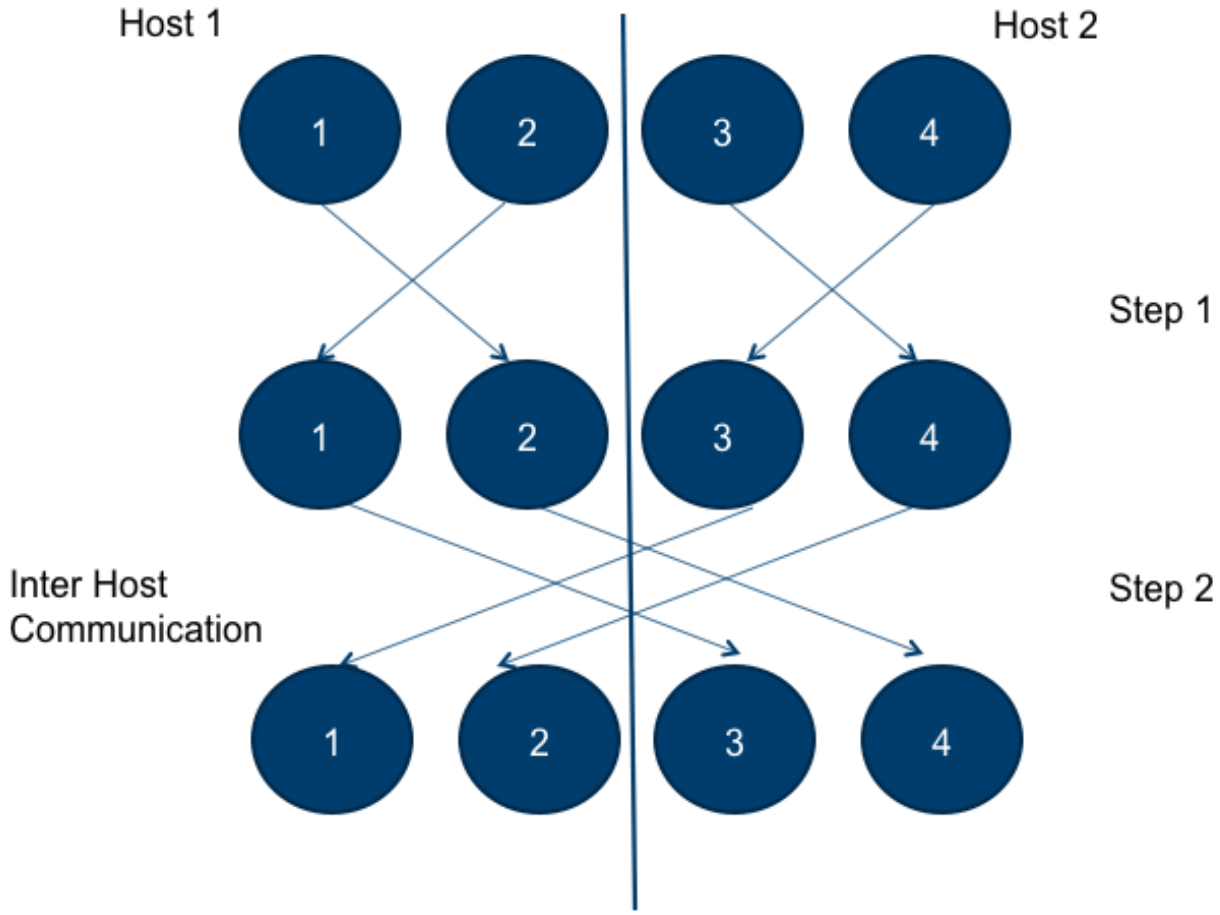


# Nonblocking Barrier – Overlap – 1 Work Quanta

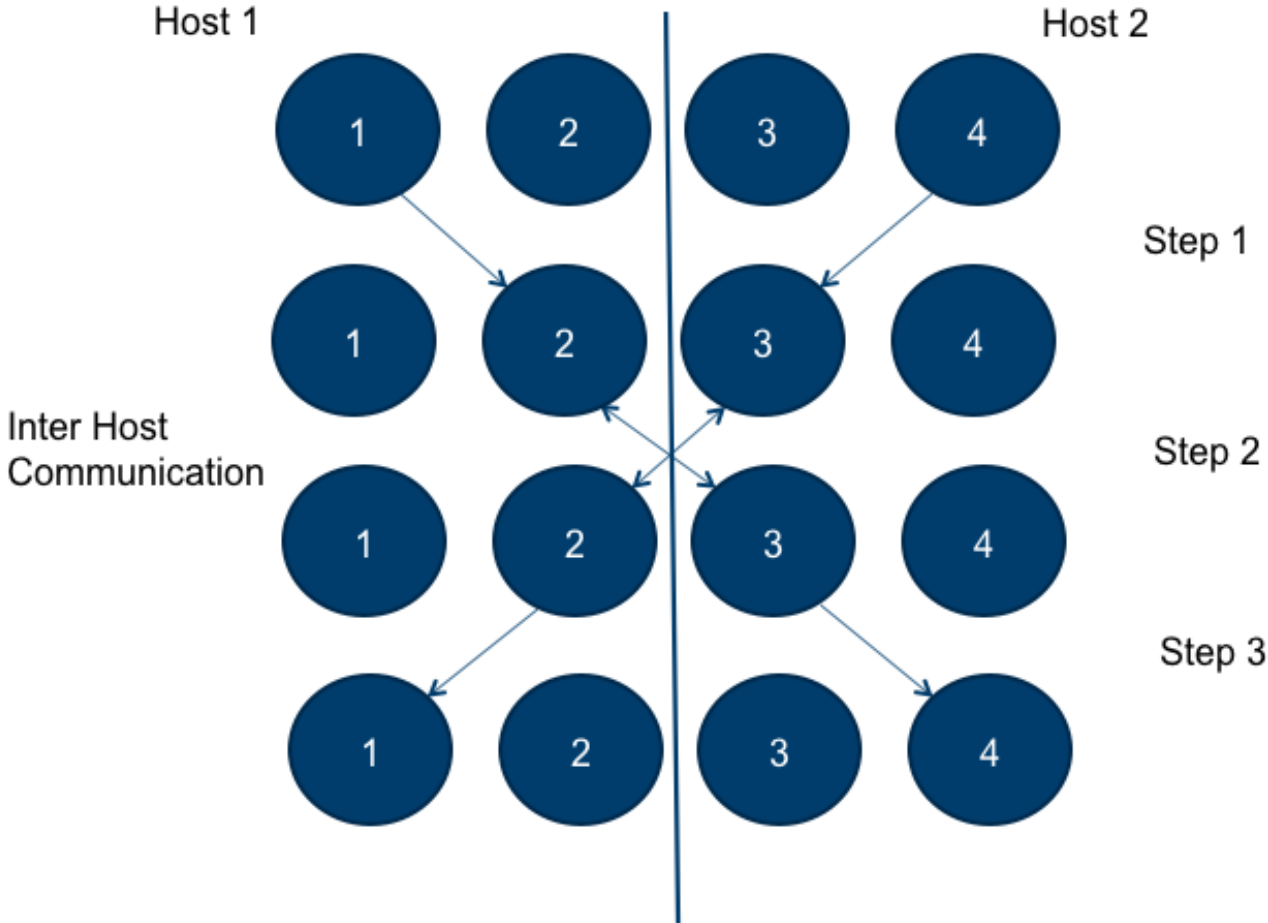


# Barrier Data Hierarchy

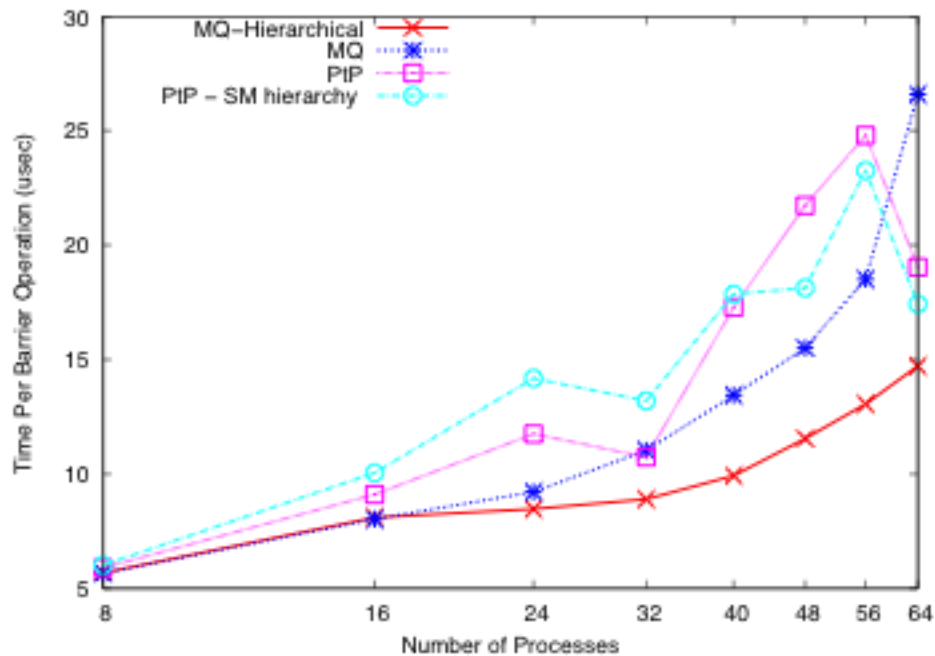
# Flat Barrier Algorithm



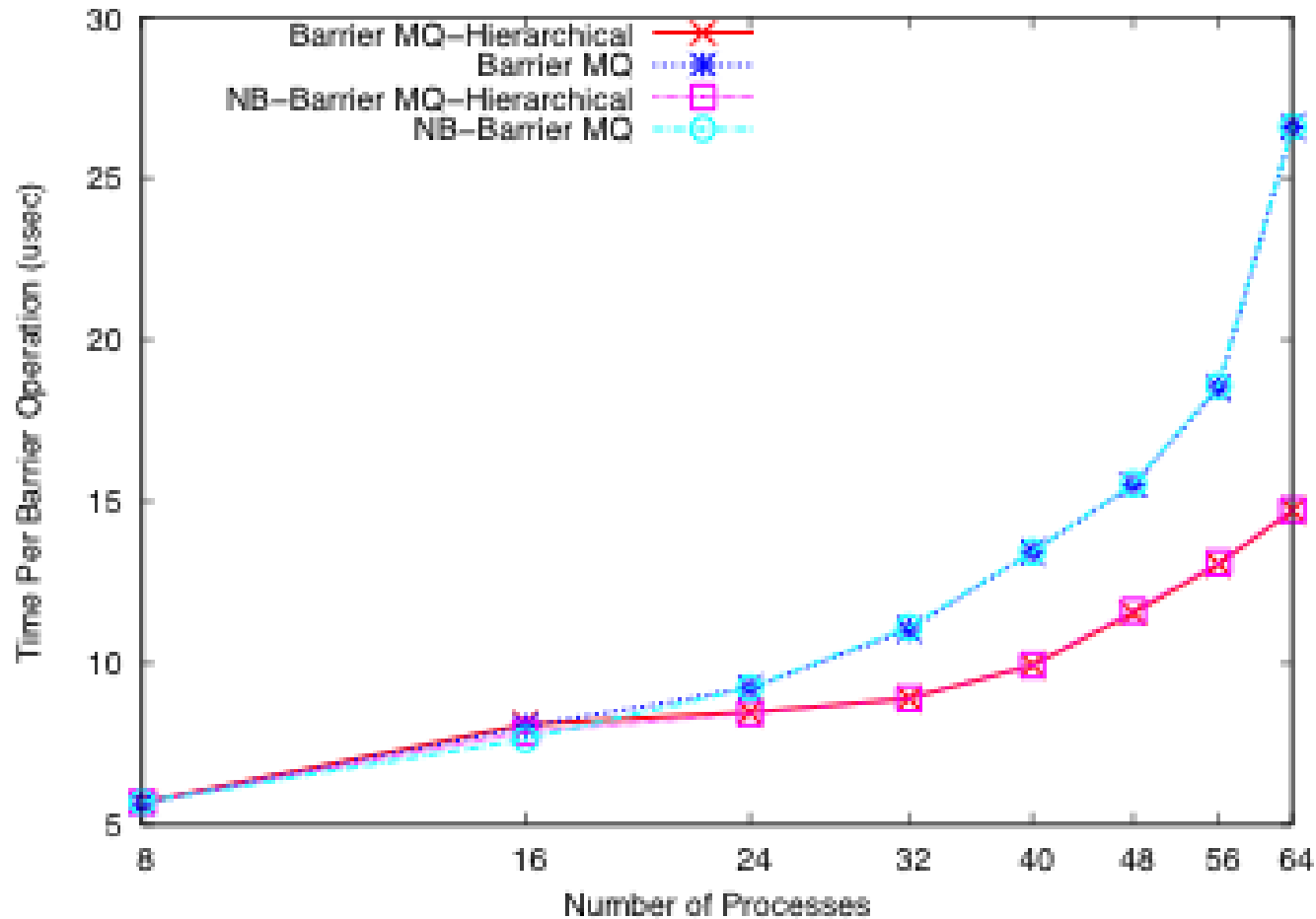
# Hierarchical Barrier Algorithm



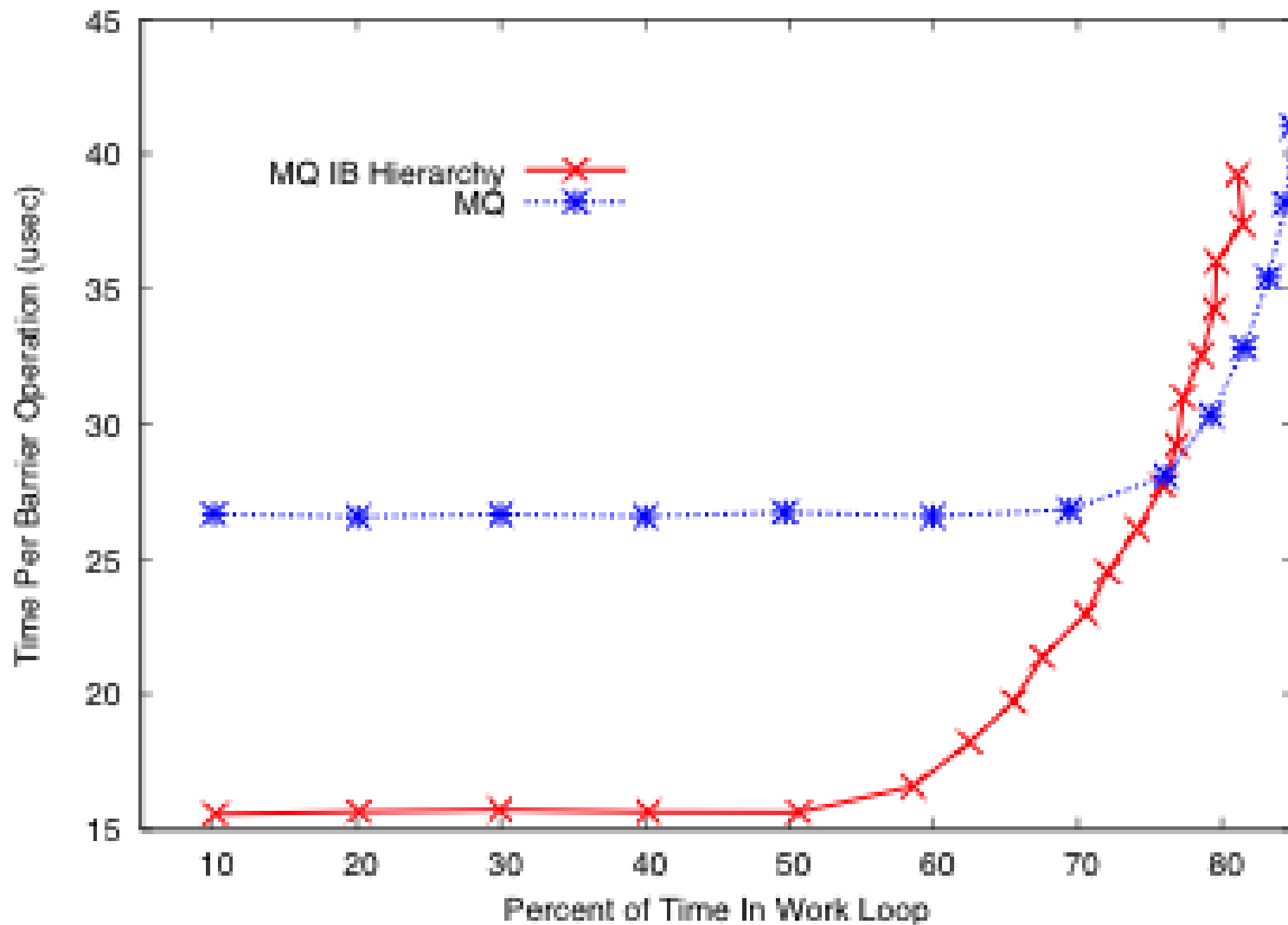
# MPI Barrier timings



# Barrier timings – blocking vs. nonblocking



# Nonblocking Barrier Overlap



# Summary

- **Added hardware support for offloading collective operations**
- **Developed MPI-level support for asynchronous collectives**
- **Good barrier performance**
- **Good overlap capabilities**
- **First MPI software release in the Open MPI code base in the June, 2010 time frame**