Grand Large
Université Paris Sud / INRIA

INRIA

MPICH-V

A Multi-Protocols Fault Tolerant MPI

Thomas Herault

INRIA Grand-Large
Université Paris Sud-XI
University of Tennessee
herault@lri.fr,
http://mpich-v.lri.fr/

UNIVERSITÉ PARIS-SUD 11

THE UNIVERSITY of TENNESSEE

Outline

- **Introduction**
- MPICH-V fault tolerance generic Framework
- Comparison of protocols
- MPICH-PCL, and Open MPI-V: optimized protocols
- FT-MPI / MPI-3-FT

A long history of research

Rollback Recovery in Message Passing: a long history of research!

2 main parameters distinguish the proposed FT techniques:

Transparency: application checkpointing, prog. controlled ckpt, **automatic**.

prog. cont. ckpt: an API returns errors handled by the programmer (FT-MPI)

application ckpt: applications store intermediate results and can restart form them

automatic: runtime detects faults and handles recovery

Checkpoint coordination: coordinated, uncoordinated.

coordinated: all processes are synchronized and compute a snapshot
all processes rollback from the same snapshot

uncoordinated: each process checkpoint independently of the others
each process is restarted independently of the others

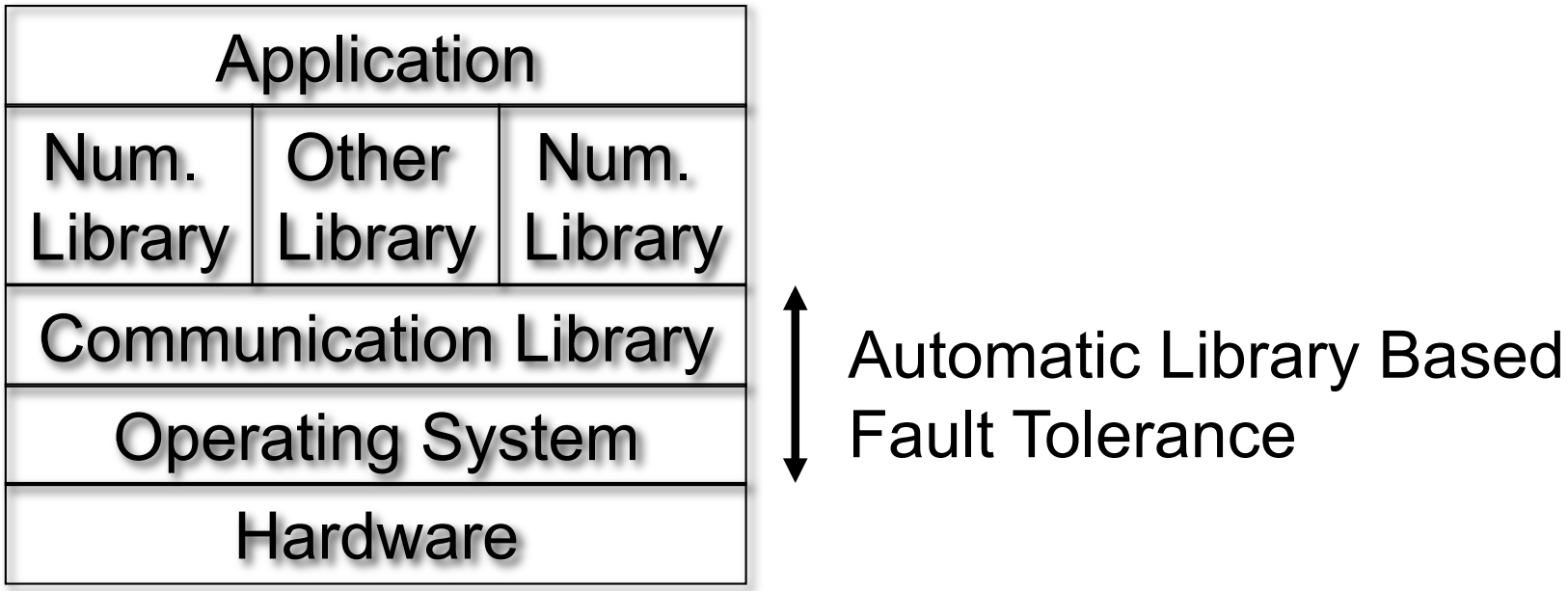
***Message logging*:** no, pessimistic, optimistic, causal.

pessimistic: all messages are logged on reliable media and used for replay

optimistic: all messages are logged on non reliable media. If 1 node fails, replay is done according to other nodes logs. If >1 node fail, rollback to last coherent checkpoint → may lead to domino effect!!!

causal: optimistic+Antecedence Graph, reduces the recovery time

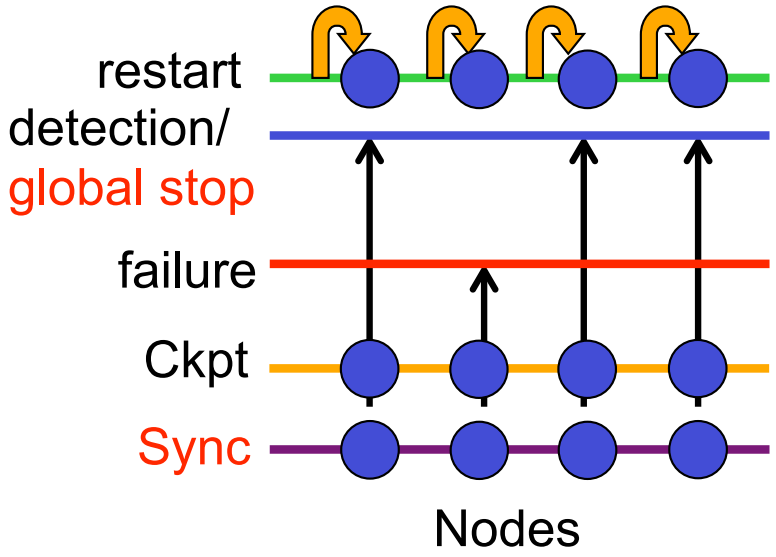
Software Stack



Coordinated & uncoodinated ckpt.

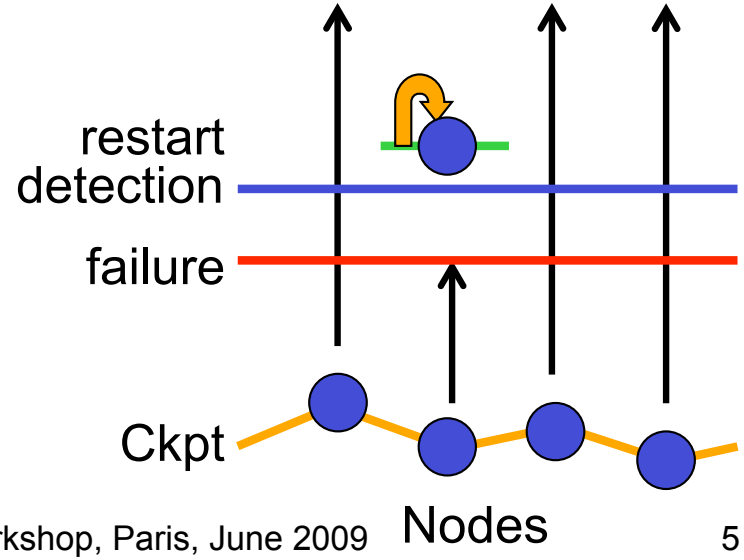
Coordinated Checkpoint (Chandy/Lamport)

The objective is to checkpoint the application when there is no in transit messages between any two nodes
→ global synchronization
→ network flush
→ supposed to be not scalable



Uncoordinated Checkpoint

No global synchronization (scalable)
→Nodes may checkpoint at any time (independently of the others)
→Need to log non deterministic events: In-transit Messages
→ communication perf. may be poor



Outline

- Introduction
- **MPICH-V fault tolerance generic Framework**
- Comparison of protocols
- MPICH-PCL, and Open MPI-V: optimized protocols
- FT-MPI / MPI-3-FT

MPICH-V Objectives

Main Goals:

I) Study, design and implement existing and new F. T. MPI protocols
II) Compare them fairly in the contexts of Cluster and Grid

Fault tolerance context:

a) Fault model: Machine crash (detected by a fault detector)

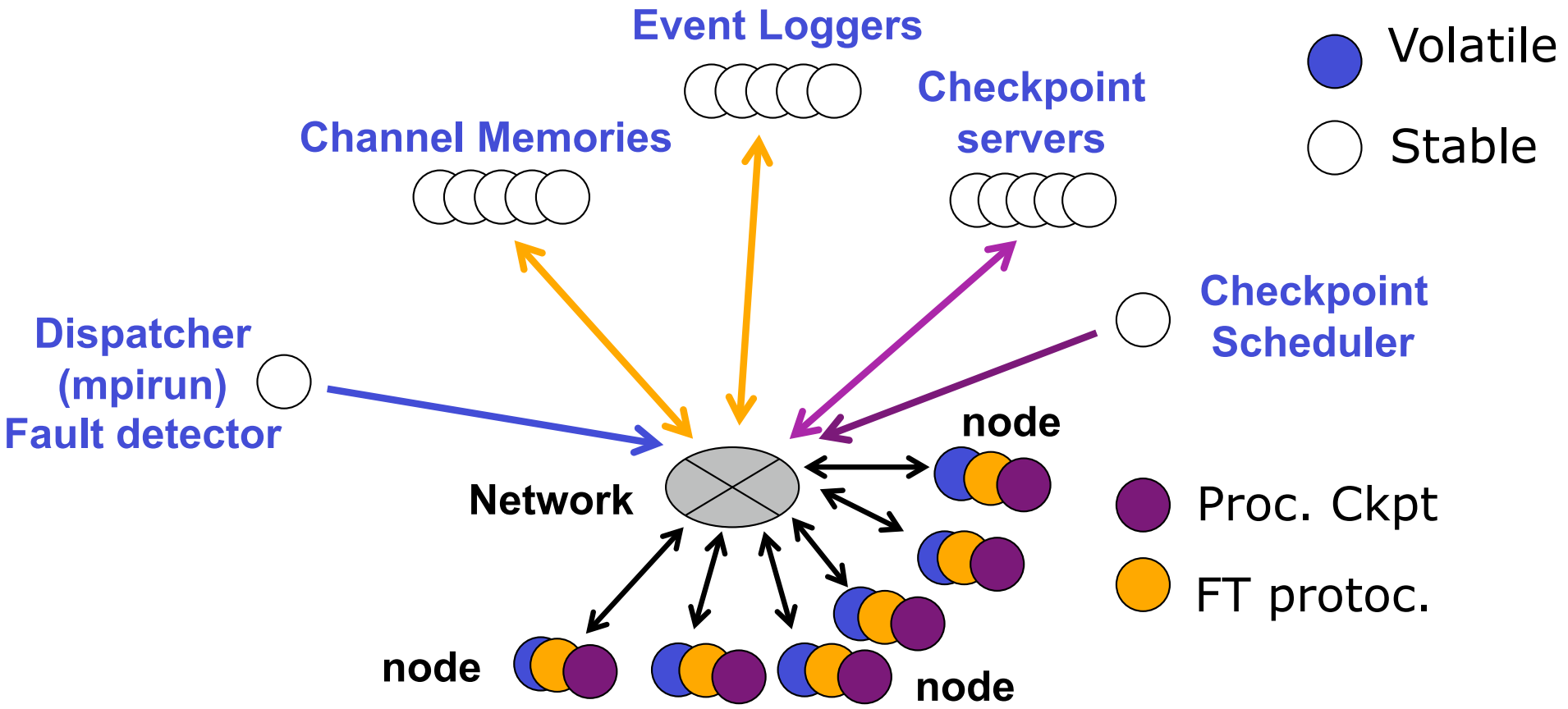
b) Distributed execution model: PieceWise Deterministic (each process execution is modeled as consisting of a number of state intervals bounded by message receiving events)

MPICH-V Objectives:

1) Automatic fault tolerance
2) Transparency for the programmer & user
3) Tolerate n faults (n being the #MPI processes)
4) Scalable Infrastructure/protocols
5) Theoretical verification of protocols

MPICH-V components

- Several stable components
- 2 components on every node (ckpt lib + deamon).
- A MPICH-V protocol uses a subset of these components

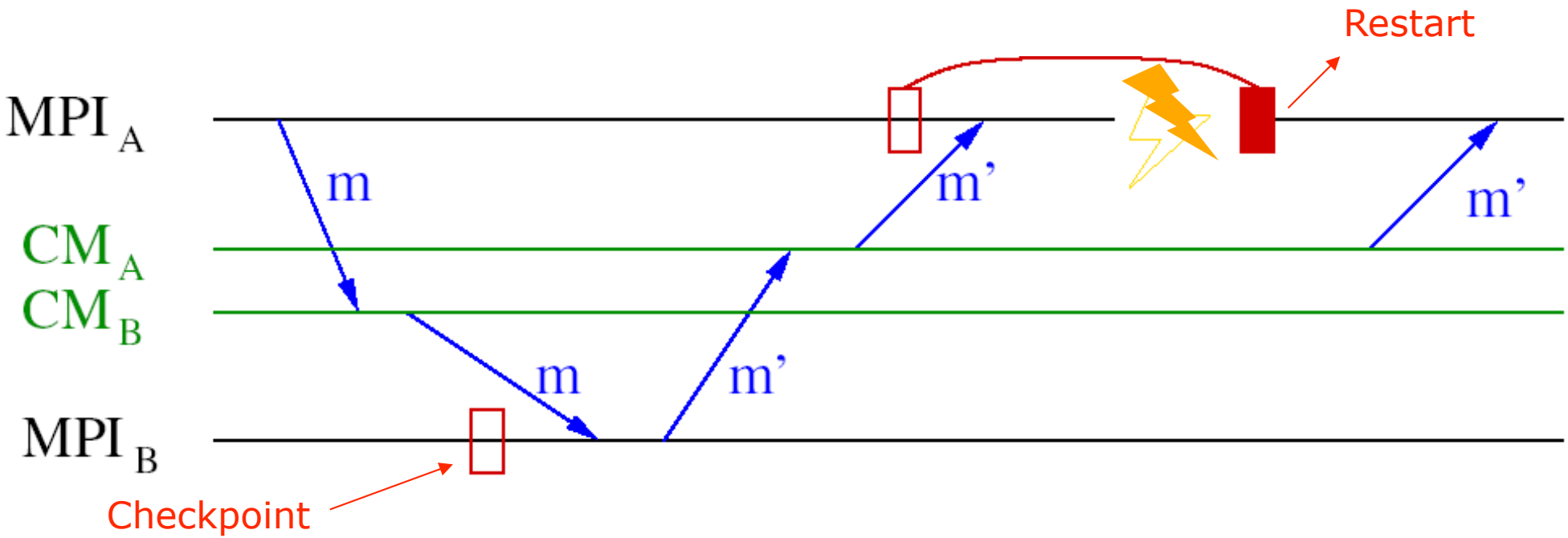


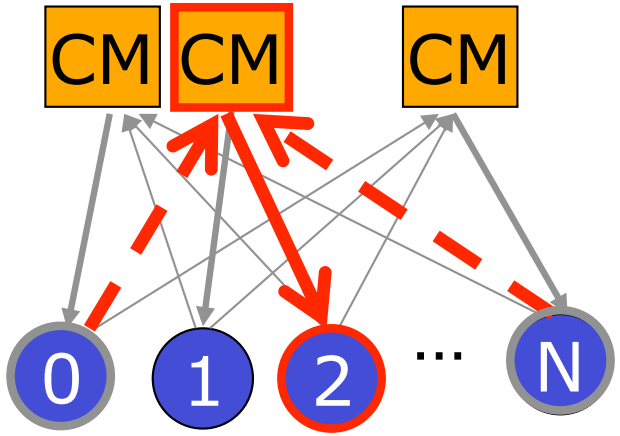
Fault Tolerant Protocols

- V1: Uncoordinated checkpointing + remote pessimistic message logging
- V2: Uncoordinated checkpointing + sender based pessimistic message logging
- Vcausal: Uncoordinated checkpointing + causal message logging
- V/CL: Coordinated Non Blocking implementation (Chandy/ Lamport)

MPICH-V1 protocol [SC2002]

Pessimistic remote message logging, Uncoordinated checkpointing





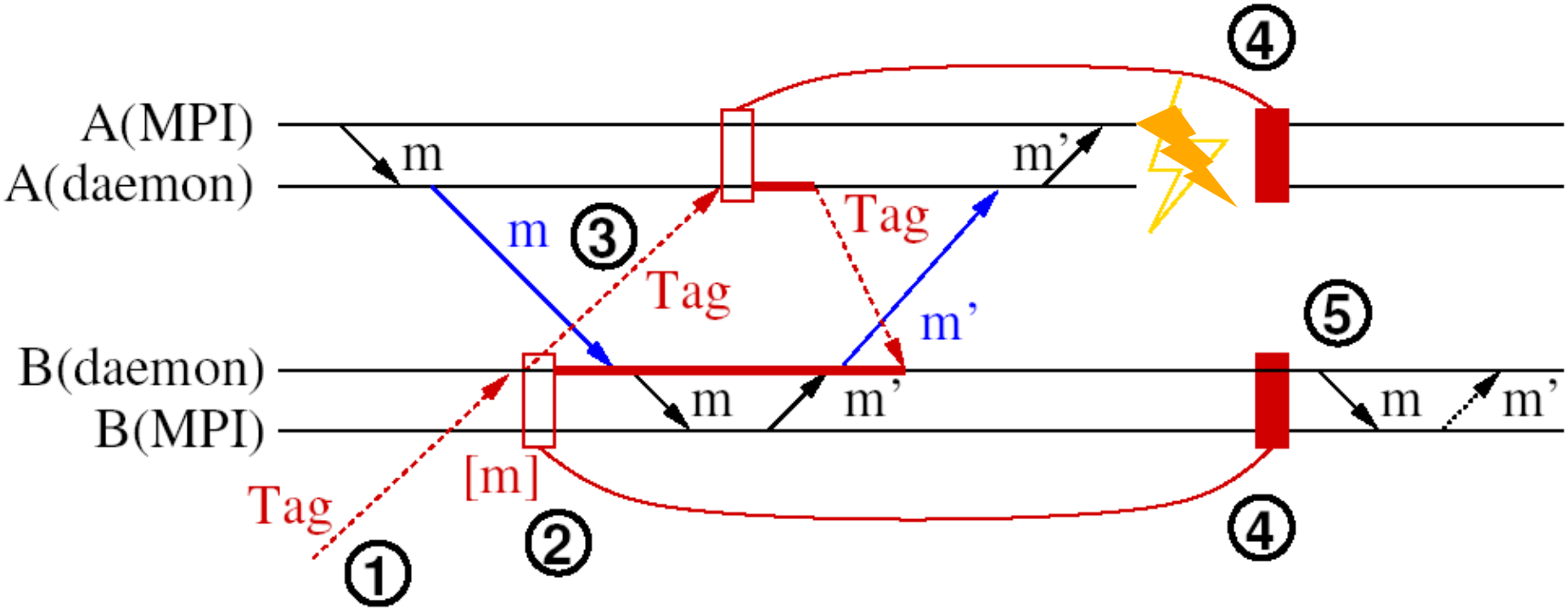
Channel Memories (stable)

Nodes (volatiles)

MPICH-V/CL protocol [Cluster2003]

Coordinated checkpointing, (Chandy-Lamport)

Reference protocol for coordinated checkpointing

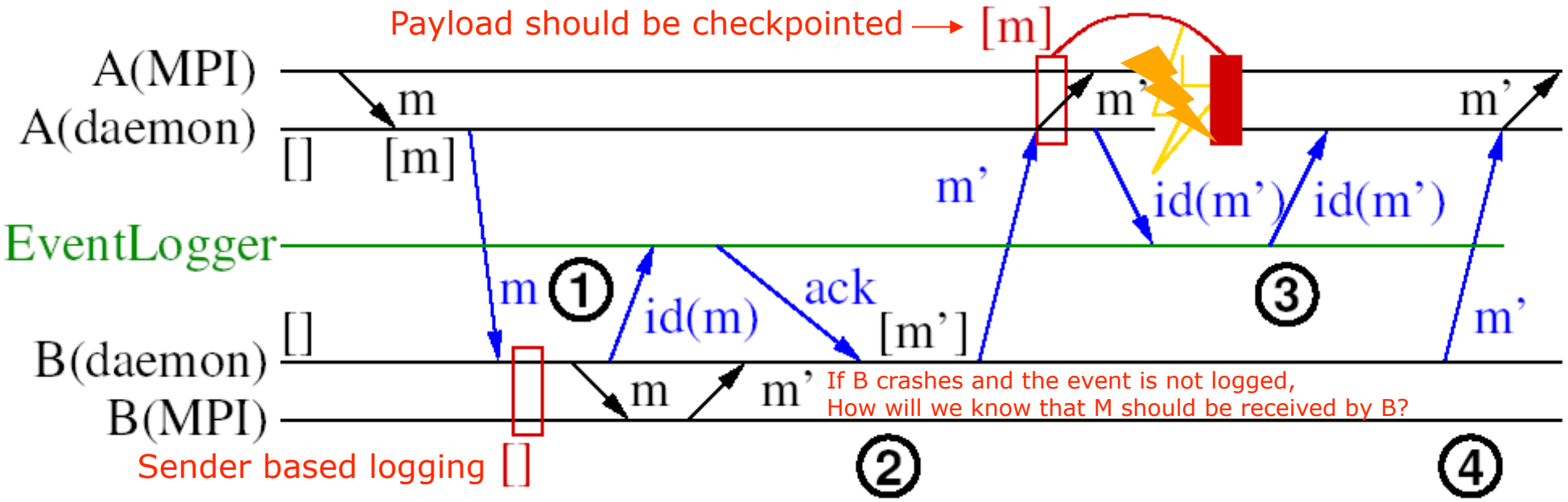


1) When receiving a Checkpoint tag, start checkpoint + store any incoming mess.
2) Store all incoming messages in checkpoint image
3) Send checkpoint tag to all neighbors in the topology.
   Checkpoint is finished when a Tag has been received from all neighbors
4) After a crash, all nodes retrieve checkpoint images from the CS
5) Deliver stored in-transit messages to restarted processes

MPICH-V2 protocol [SC2003]

Pessimistic sender based message logging, Uncoordinated ckpt.

Improve bandwidth (direct communications) →remove Channel Memories
Pessimistic message logging: only reception events should be saved

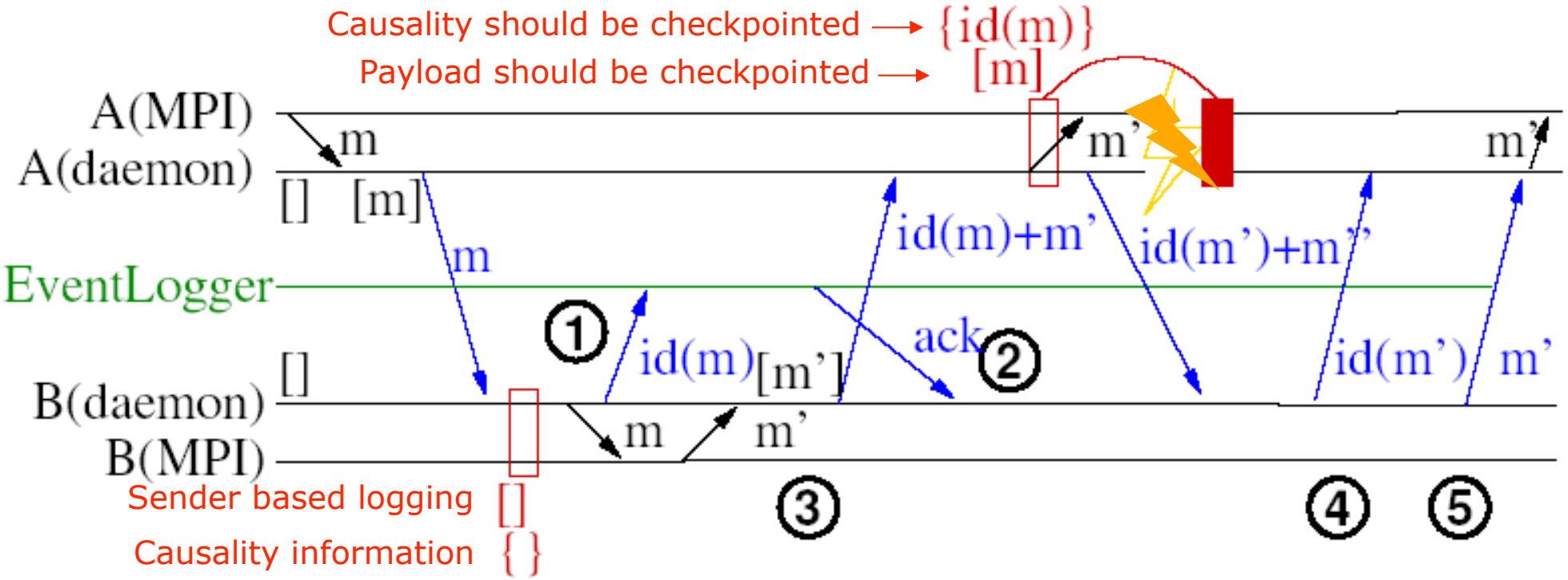


1) Send information about reception to Event Logger
2) When sending, first wait for EL acknowledge of prev. mess. And store mess. payload in mem (memory mapped file).
3) After a crash, retrieve ordered list of reception from EL
4) Contact initial senders for replay

MPICH-Vcausal protocol [Cluster 2004]

Causal message logging, Uncoordinated checkpointing

Improve latency → asynchronous event logging
All message event should be saved (non logged events) on the on nodes
→ piggyback message event to MPI messages



1) Send information about reception to Event Logger, asynchronously (total order)
2) EL acknowledge of prev. mess. asynchronously
3) If no ack from the EL, piggyback causality info to messages
4) After a crash, retrieve ordered list of reception from EL **and other nodes**
5) Contact initial senders for replay

Outline

- Introduction
- MPICH-V fault tolerance generic Framework
- **Comparison of protocols**
- MPICH-PCL, and Open MPI-V: optimized protocols
- FT-MPI / MPI-3-FT

A first comparison

	Main Advantage	Main Drawback
V1 (2002)	Uncoordinated Checkpoiting	Every com. crosses the Channel Memory
V2 (2003)	Direct communications	Synchronous Event Logging
Vcausal (2004)	Asynchronous Event Logging	Piggyback information (events) in every message
V/CL (2003)	Direct, CP piggyback free communications	Coordinated Checkpointing

Experimental platform
(early experiments)

32 node Cluster with Ethernet 100 network
(other experiments on Myrinet and SCI in papers)

Athlon XP 2800+ (2Ghz) CPU
1 GB memory (DDR SDRAM)
70 GB IDE ATA 100 hard drive
100 Mbits/s Ethernet NIC

All nodes connected by a 48 ports Ethernet switch

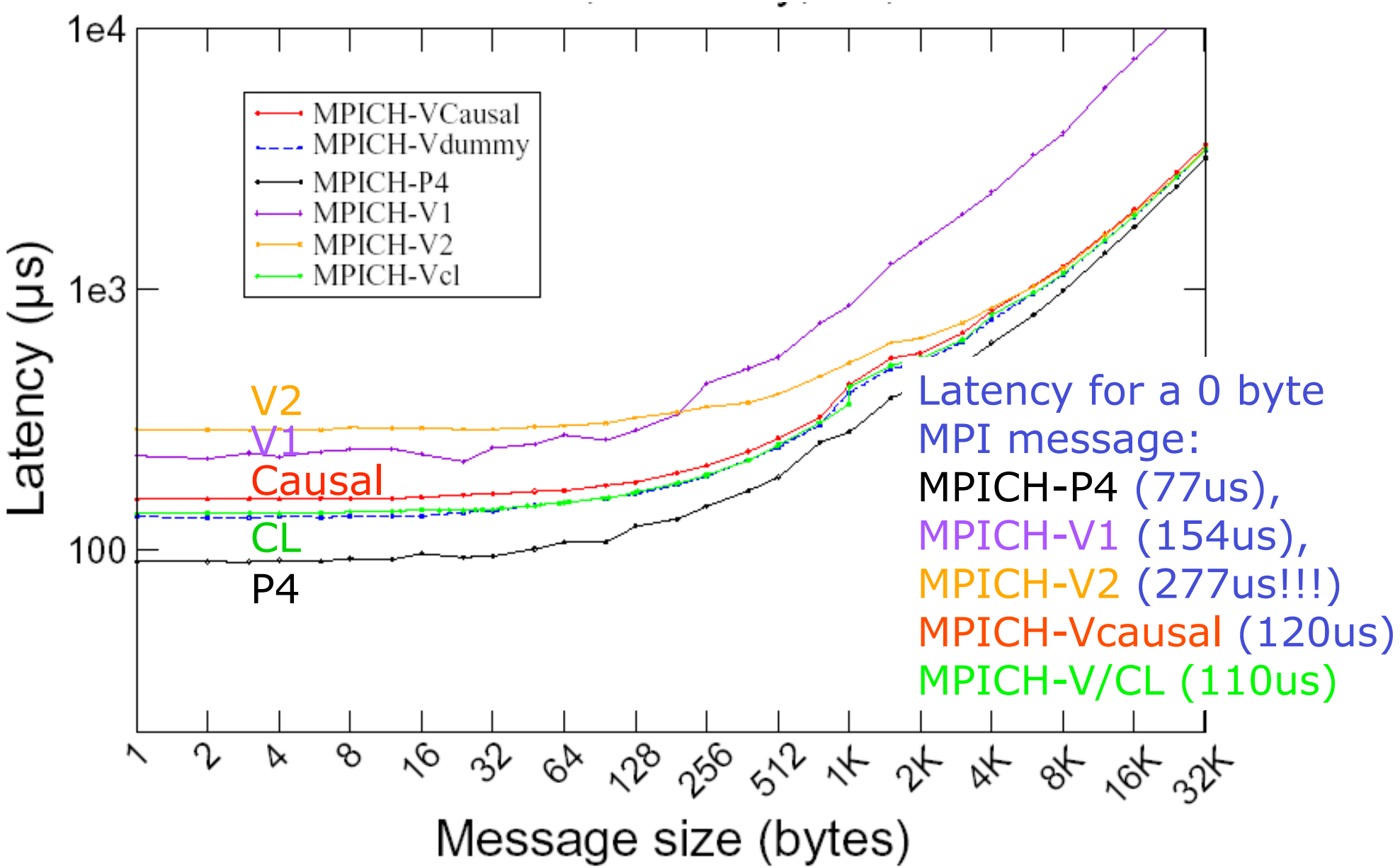
Linux 2.4.20
MPICH 1.2.5
Benchmark compiled with GCC –O3 and PGF77

Tests in dedicated mode
Measurement repeated 5 times (only mean is presented)

Microbenchmark with NetPIPE utility

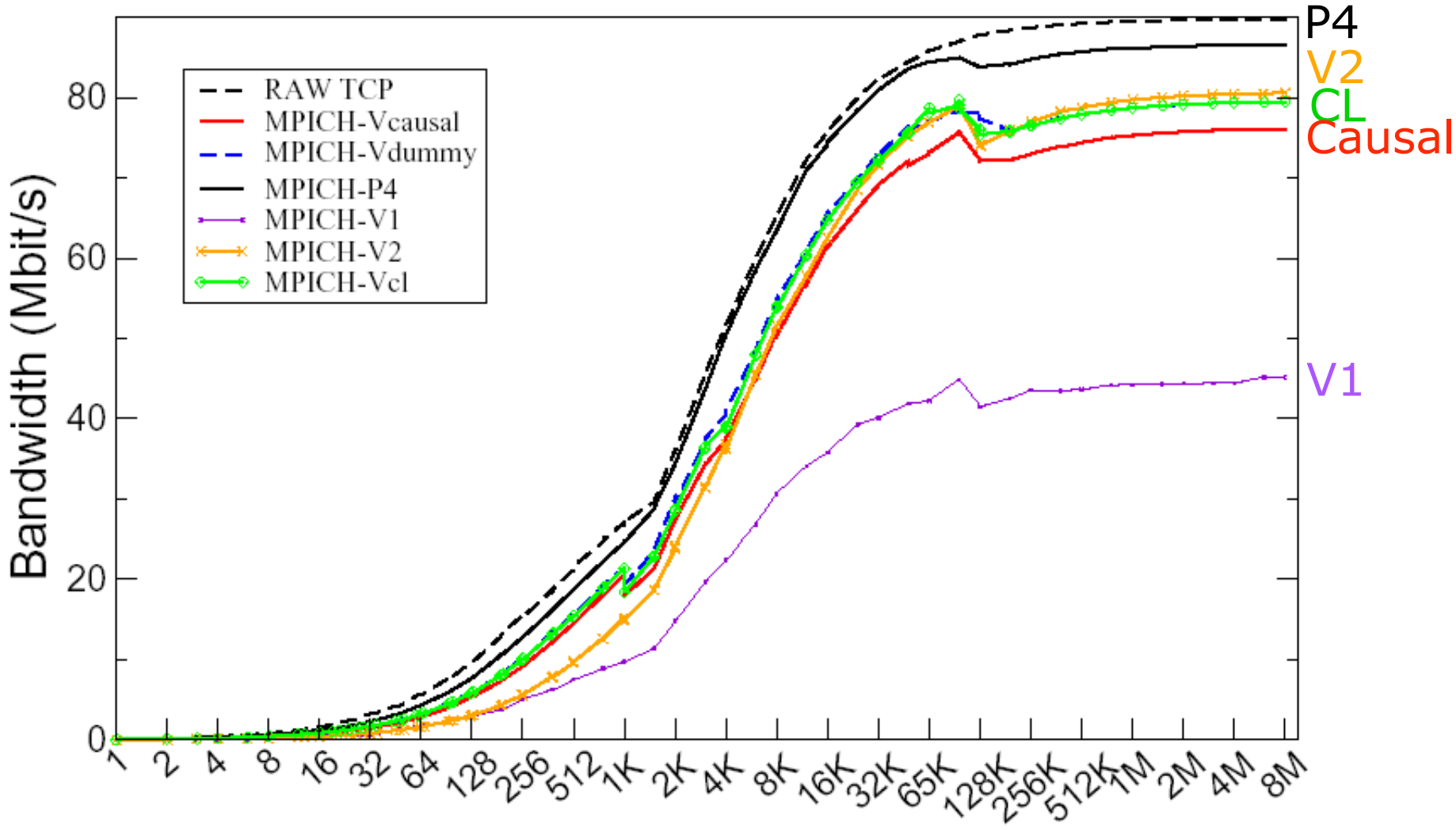
A single Checkpoint server for all experiments

V-protocols Latency



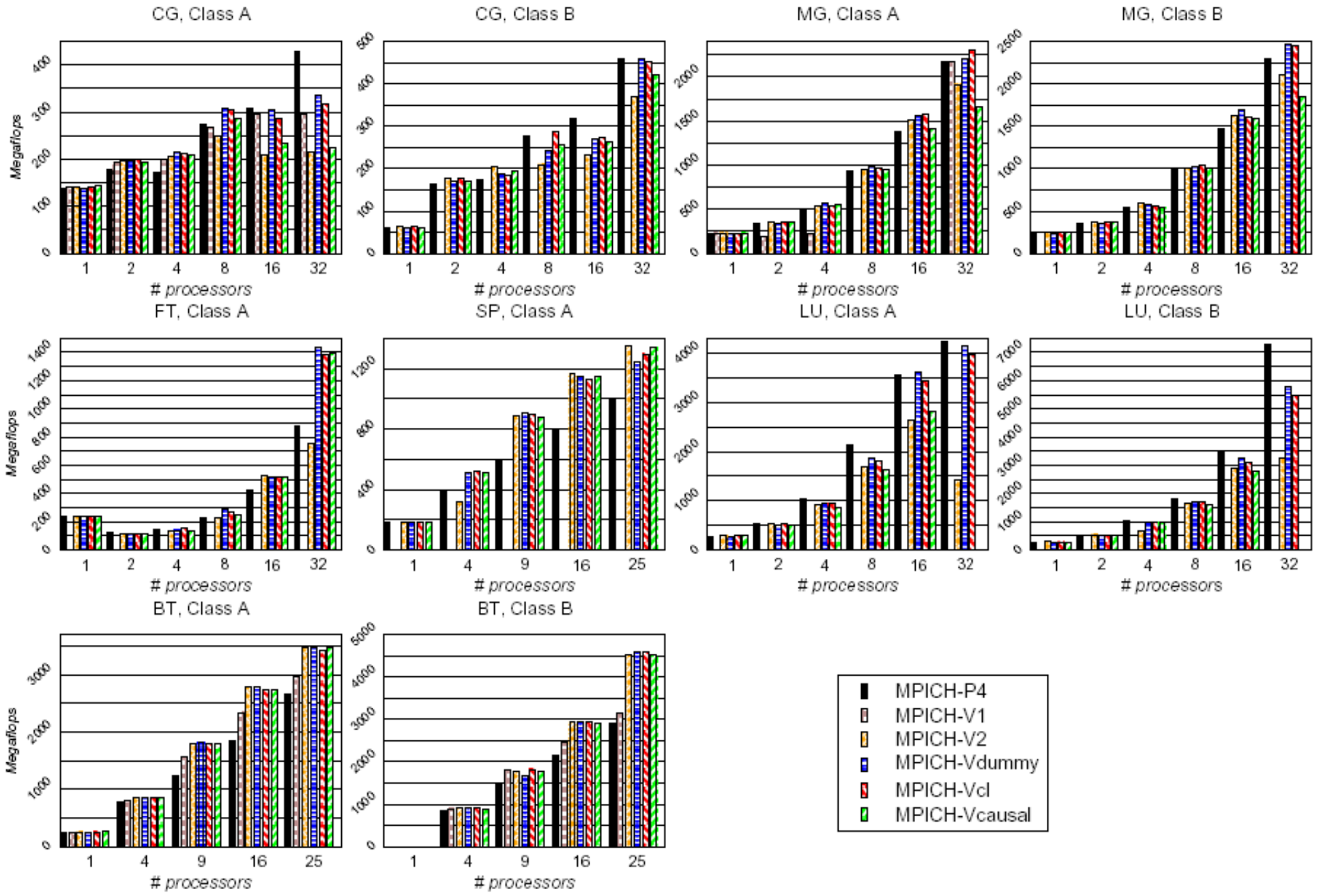
Latency for a 0 byte MPI message:
MPICH-P4 (77us),
MPICH-V1 (154us),
MPICH-V2 (277us!!!)
MPICH-Vcausal (120us)
MPICH-V/CL (110us)

V-protocols Bandwidth



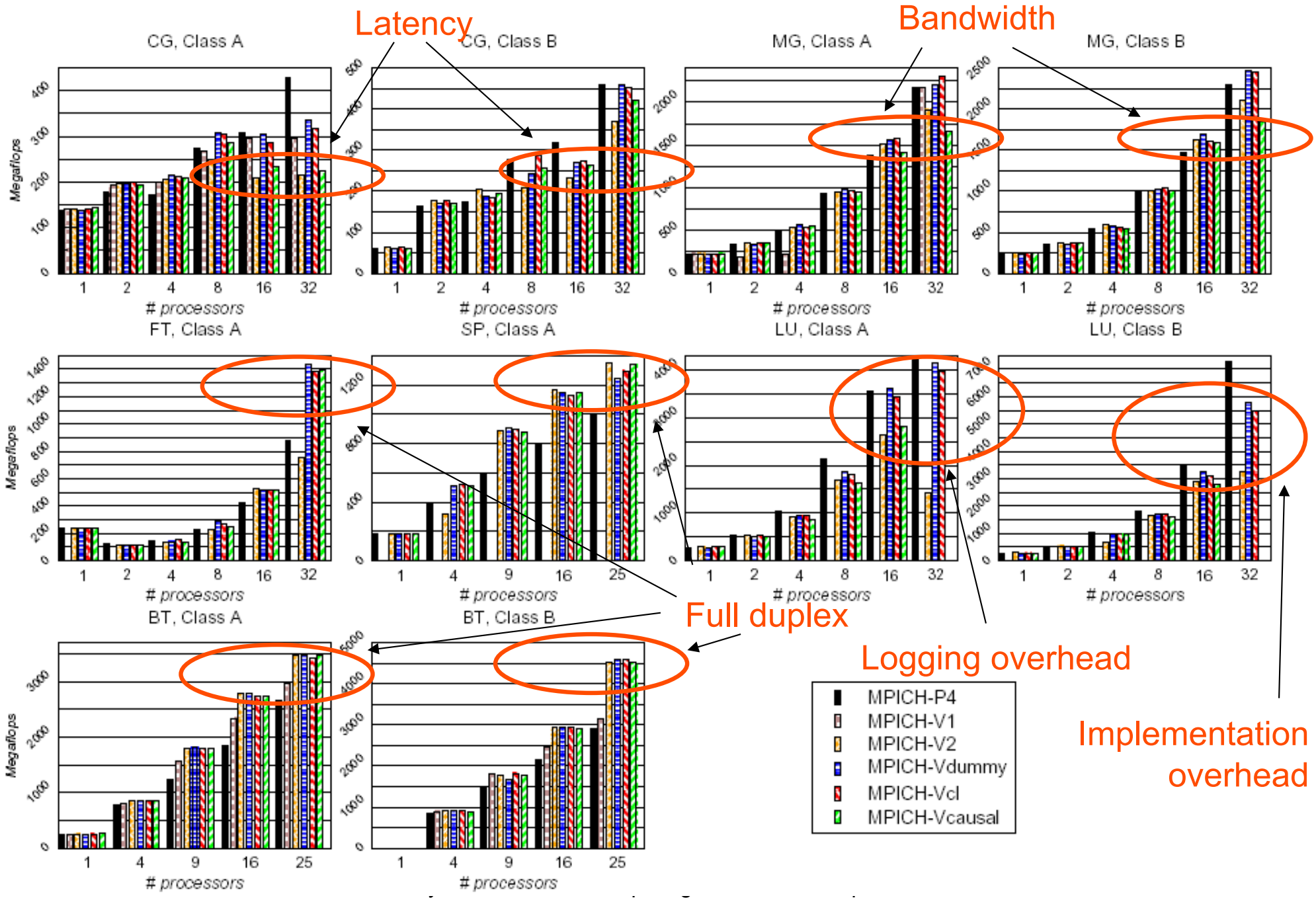
Grand Large

NAS-Benchmark Class A and B

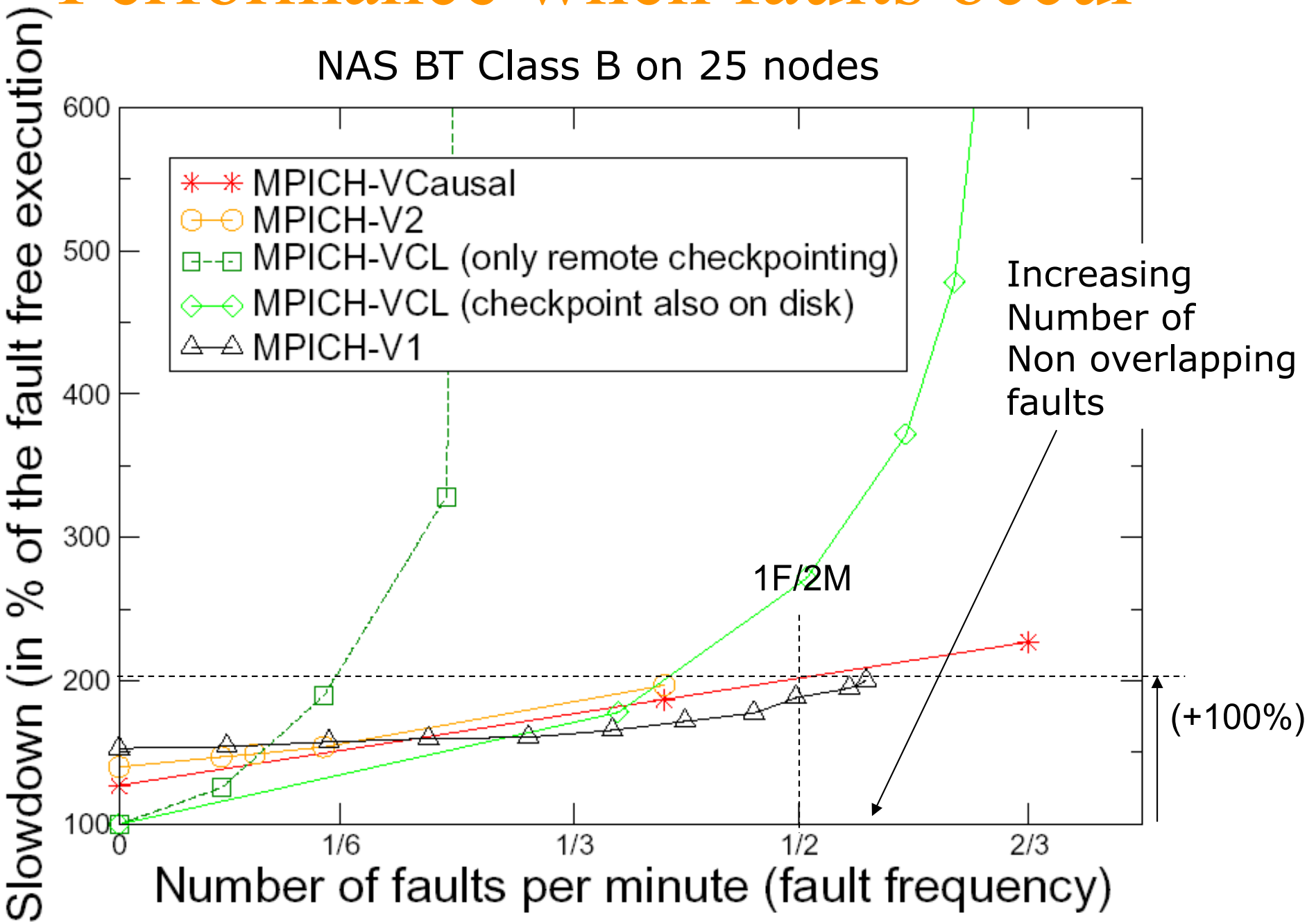


Grand Large

NAS Benchmark Class A and B



Performance when faults occur

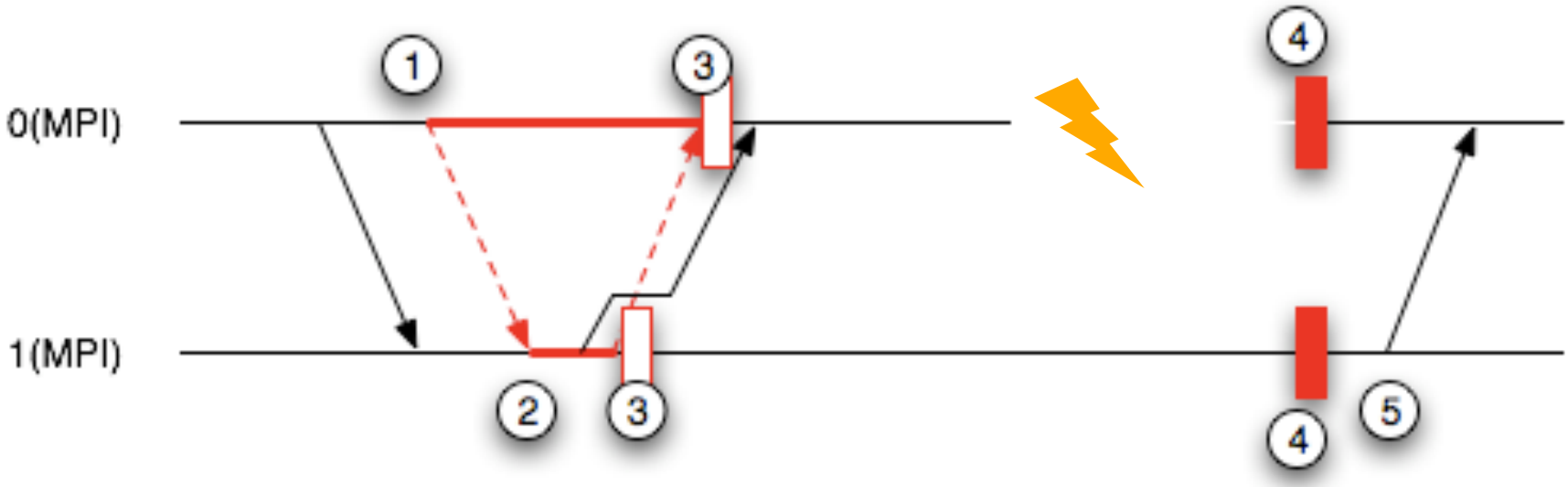


Outline

- Introduction
- MPICH-V fault tolerance generic Framework
- Comparison of protocols
- **MPICH-PCL, and Open MPI-V: optimized protocols**
- FT-MPI / MPI-3-FT

MPICH-P/CL protocol [FGCS'07]

Coordinated Blocking checkpointing

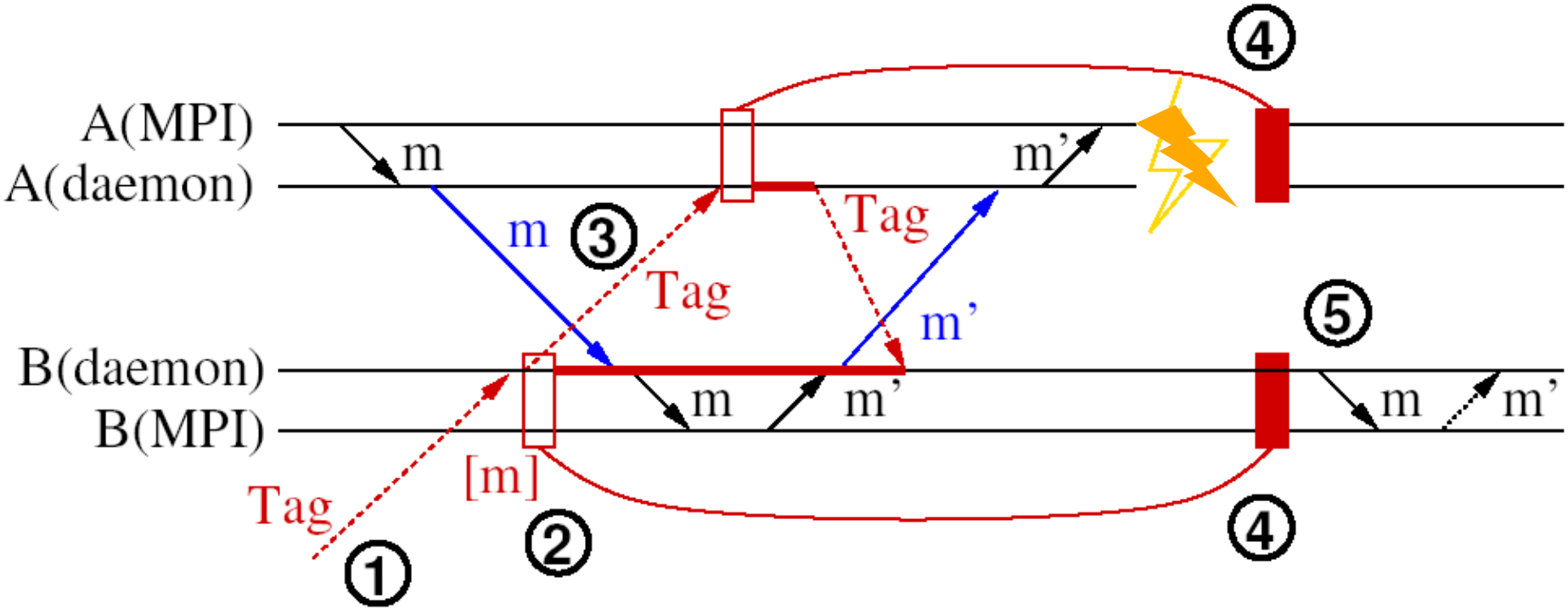


1) Send a checkpoint tag to all others, and block communications (as soon as tag is gone)
2) When receiving a checkpoint tag from any, do 1)
3) When received a checkpoint tag from all, take checkpoint, and unfreeze communications
4) After a crash, all nodes retrieve checkpoint images from the CS
5) In-transit messages are stored in message queues

MPICH-V/CL protocol [Cluster2003]

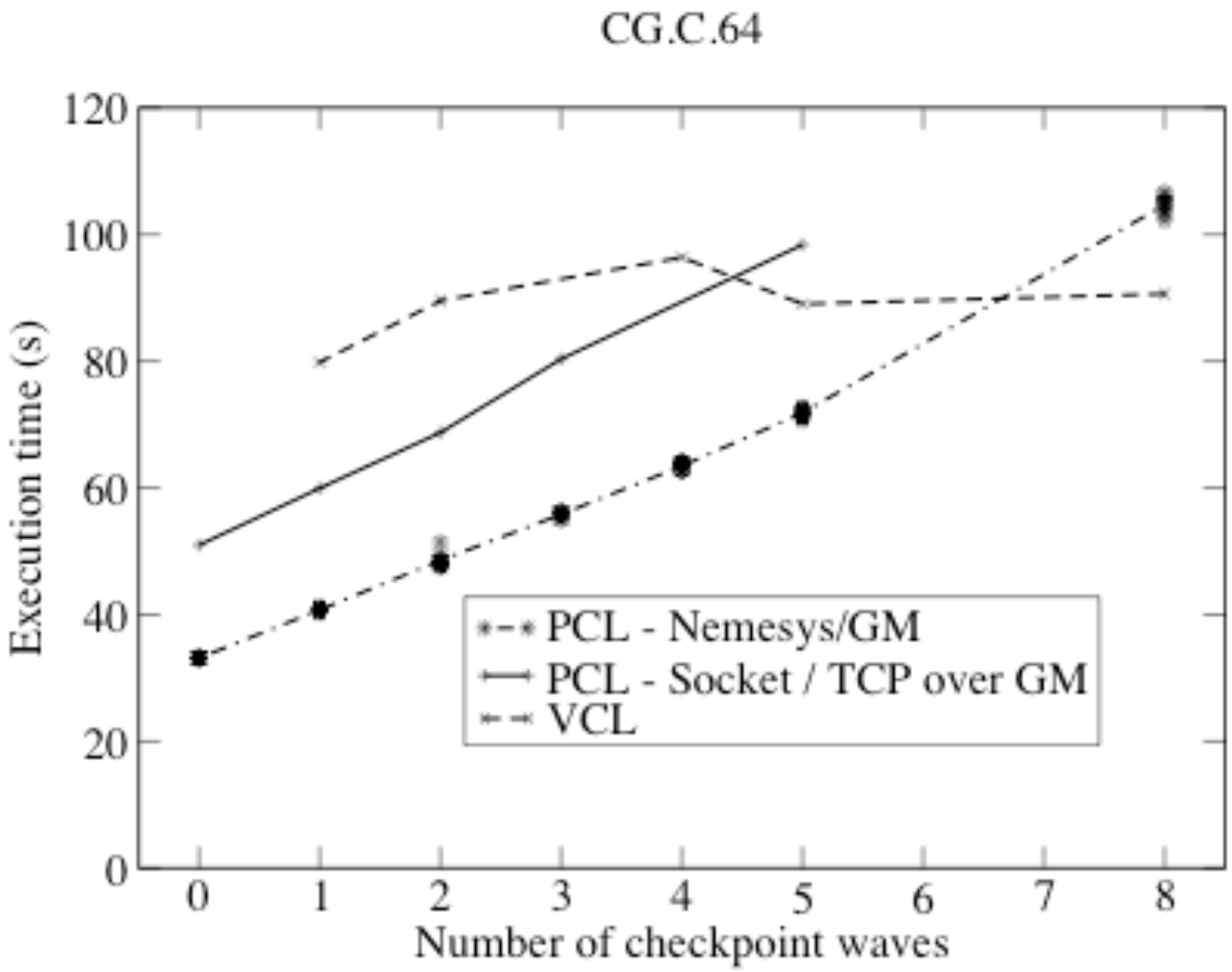
Coordinated checkpointing, (Chandy-Lamport)

Reference protocol for coordinated checkpointing



1) When receiving a Checkpoint tag, start checkpoint + store any incoming mess.
2) Store all incoming messages in checkpoint image
3) Send checkpoint tag to all neighbors in the topology.
   Checkpoint is finished when a Tag has been received from all neighbors
4) After a crash, all nodes retrieve checkpoint images from the CS
5) Deliver stored in-transit messages to restarted processes

MPICH-PCL on Low Latency Network (myri2k - GM driver)



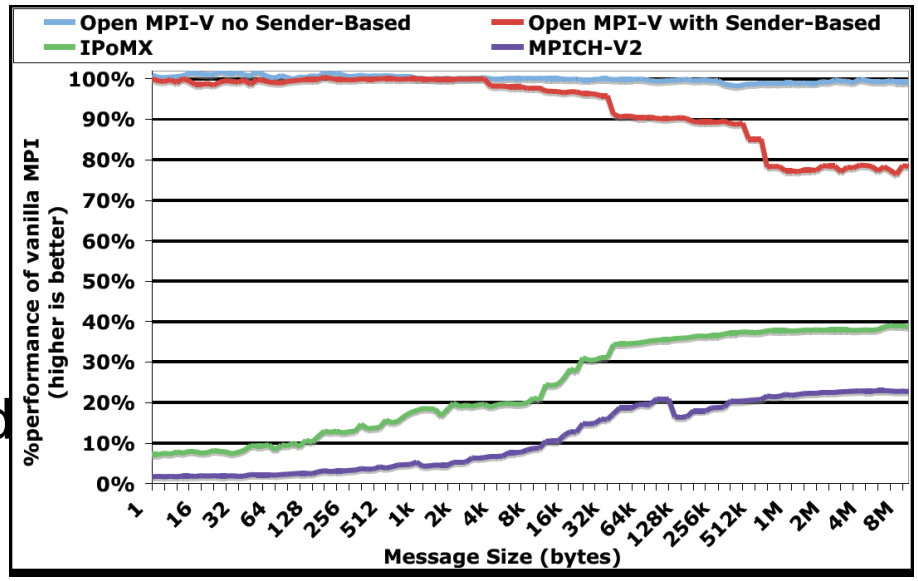
Improved Message Logging

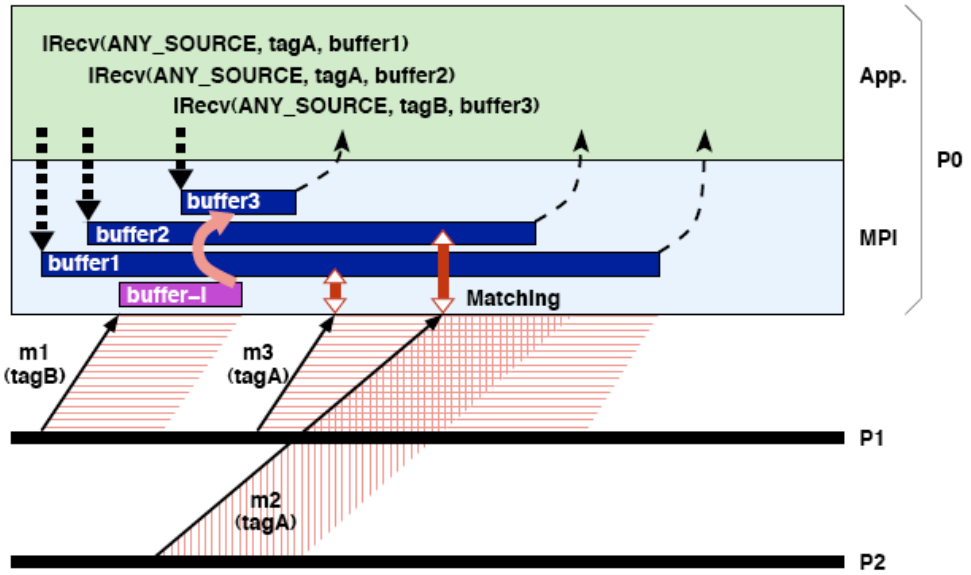
Fig. from Bouteiller

•High speed MPI implementations use Zero Copy and decompose Recv in:
a) Matching, b) Delivery

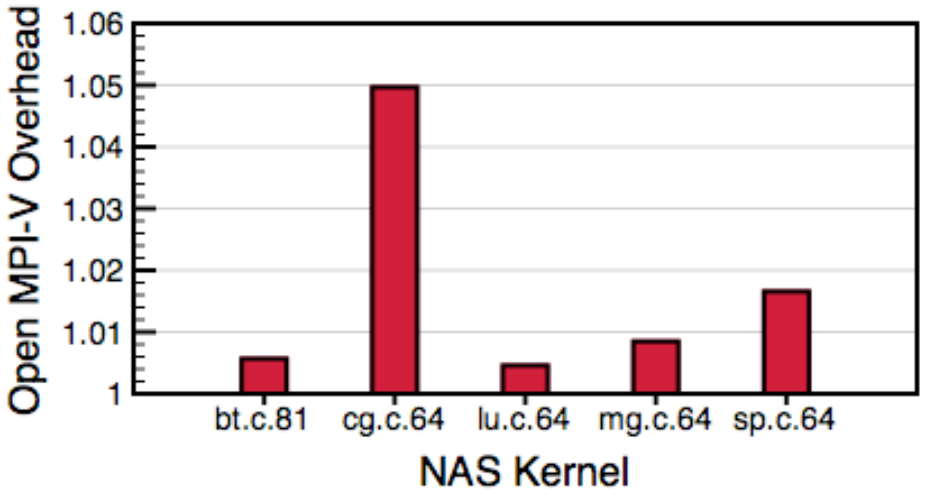
OpenMPI-V implements Mes. Log. within MPI: different event types are managed differently, distinction between determ. and non determ. events, optimized mem. copy

Bandwidth of OpenMPI-V compared to others





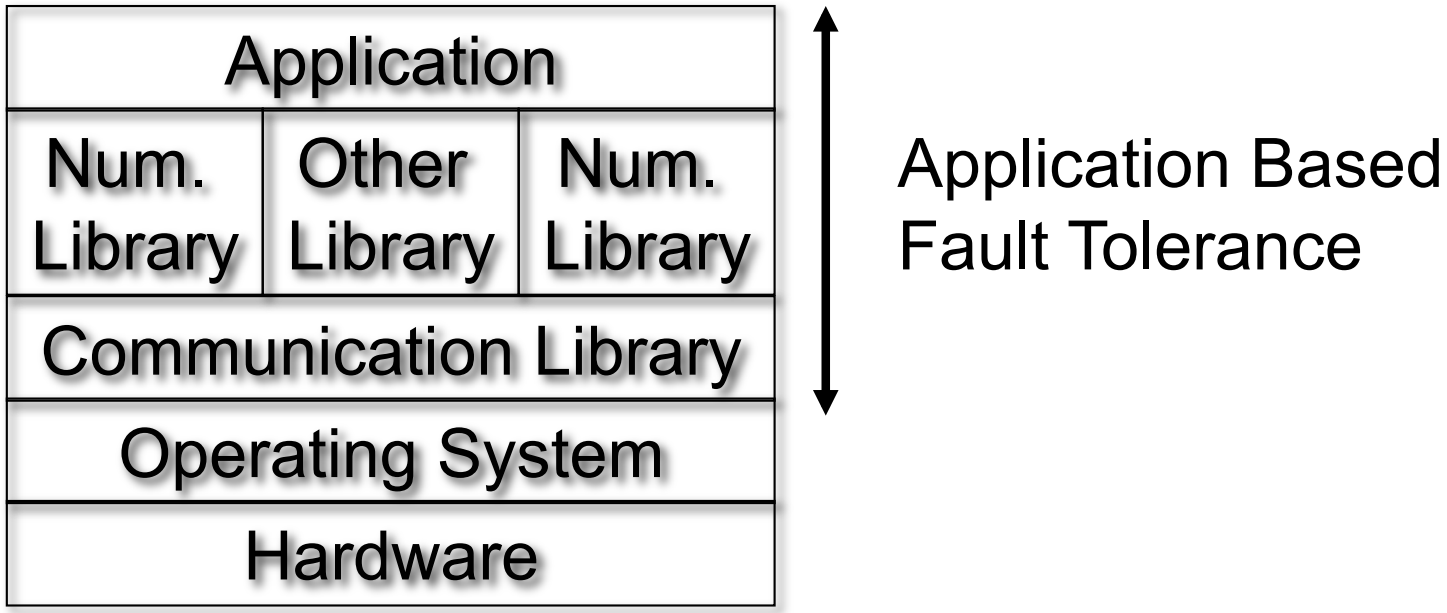
OpenMPI-V Overhead on NAS (Myri10g)



Outline

- Introduction
- MPICH-V fault tolerance generic Framework
- Comparison of protocols
- MPICH-PCL, and Open MPI-V: optimized protocols
- **FT-MPI / MPI-3-FT**

Software Stack



FT-MPI

- Failures are notified to all processes, ASAP
- Repair operation:
  MPI_Comm_dup(MPI_COMM_WORLD)
  - Shrink / Blank / Replace modes
- After repair: state similar as the state after MPI_Init
- Collectives: two-phase commit at the end of each collective

FT-MPI: Diskless Checkpointing

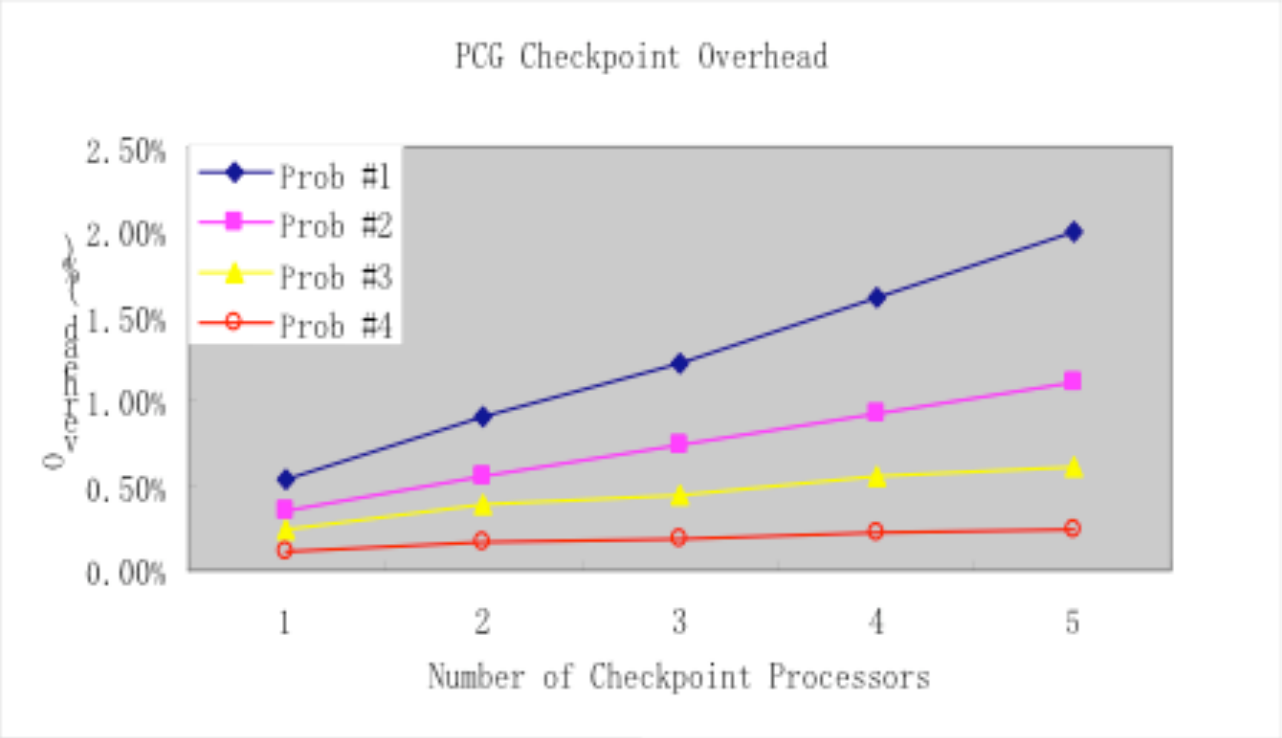


Figure 6: PCG Checkpoint Overhead

Figure from Chen, Fagg, Langou, Angskun, Bosilca, Dongarra

MPI-3 (FT)

- Proposal for MPI-3 (FT WG)
- Enable Distributed Repair
  - Failure notification ALAP
- Repair: either Local or Global (collective)
- Avoid Reseting All
  - Re-Join Communicators

Outline

- Introduction
- MPICH-V fault tolerance generic Framework
- Comparison of protocols
- MPICH-PCL, and Open MPI-V: optimized protocols
- FT-MPI / MPI-3-FT

Conclusion

- Automatic Rollback/Recovery:
  - Which protocol is best fitted
  - Depends on the reliability of the platform, and the communication pattern of the application
  - We suspect: not that much performance difference between protocols
    - Unless under really frequent failures
  - Coordination is not as costly as we first thought
    - Still need evaluation on peta-scale systems

Conclusion

- What prevents Automatic/Transparent Checkpointing in Petascale Computing?
  - Lack of widely available MPI implementation featuring system-level checkpointing
    - Open MPI (still XP)
  - Fault Tolerant Runtime Environment
    - Will come with FT RTE for Application-Level Checkpointing
  - Widely distributed System Level Checkpointer / Standardized System Level Checkpointing
    - BLCR becomes more widely spread
  - Application I/O
    - FileSystem Snapshot ioctl
  - Checkpoint Storage Hardware / Network / Scheduling

Conclusion

- Application-level Fault-Tolerance: The way to go?
  - No known large-scale evaluation of automatic approaches
  - Library Stacking is an issue in MPI
- Costs of Application-level:
  - FT-MPI approach was too synchronous
  - Current proposal puts hidden synchronous operations in the implementation

Grand Large

INRIA

MPICH-V

www.lri.fr/~gk/MPICH-V

A Multi-Protocols Fault Tolerant MPI

Questions ?