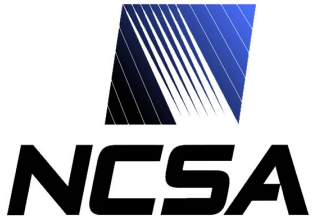
Joint Laboratory for Petascale Computation

INRIA NCSA

Rollback-Recovery Protocols for Send-Deterministic Applications

Amina Guermouche, Thomas Ropars, Elisabeth Brunet, Marc Snir and Franck Cappello



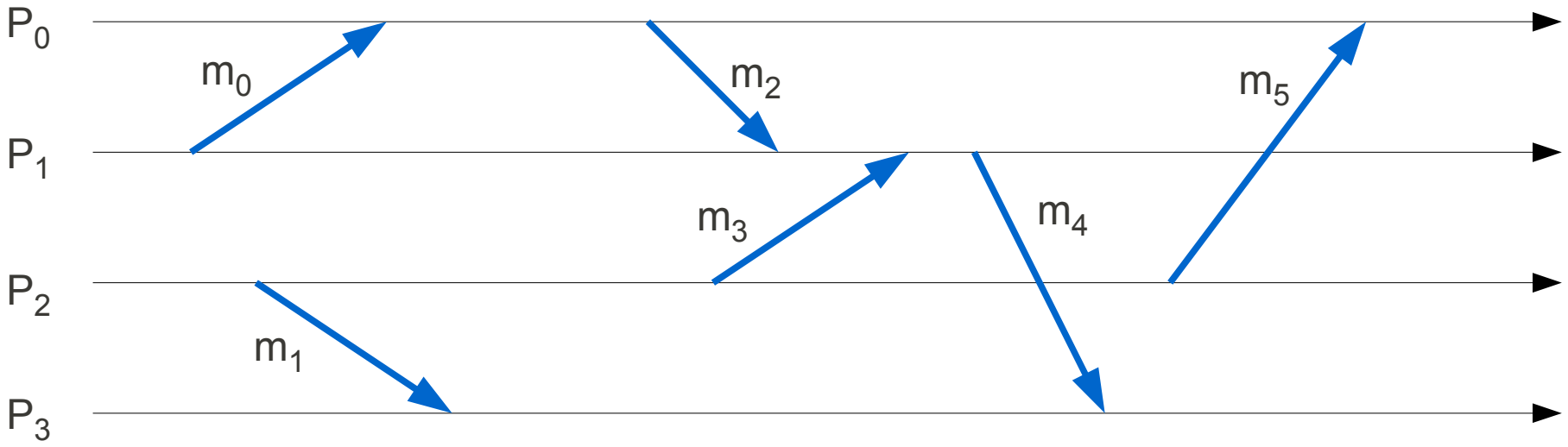
Fault Tolerance in HPC Systems is Mandatory

- Resiliency is a key problem at very large scale
  - MTBF of a few hours at Exascale
  - Rollback-recovery is needed to allow applications to terminate
    - Saving information on a reliable storage
    - Based on checkpoints
- Power consumption is another major issue
  - Limit the amount of rolled back computation in the event of a failure

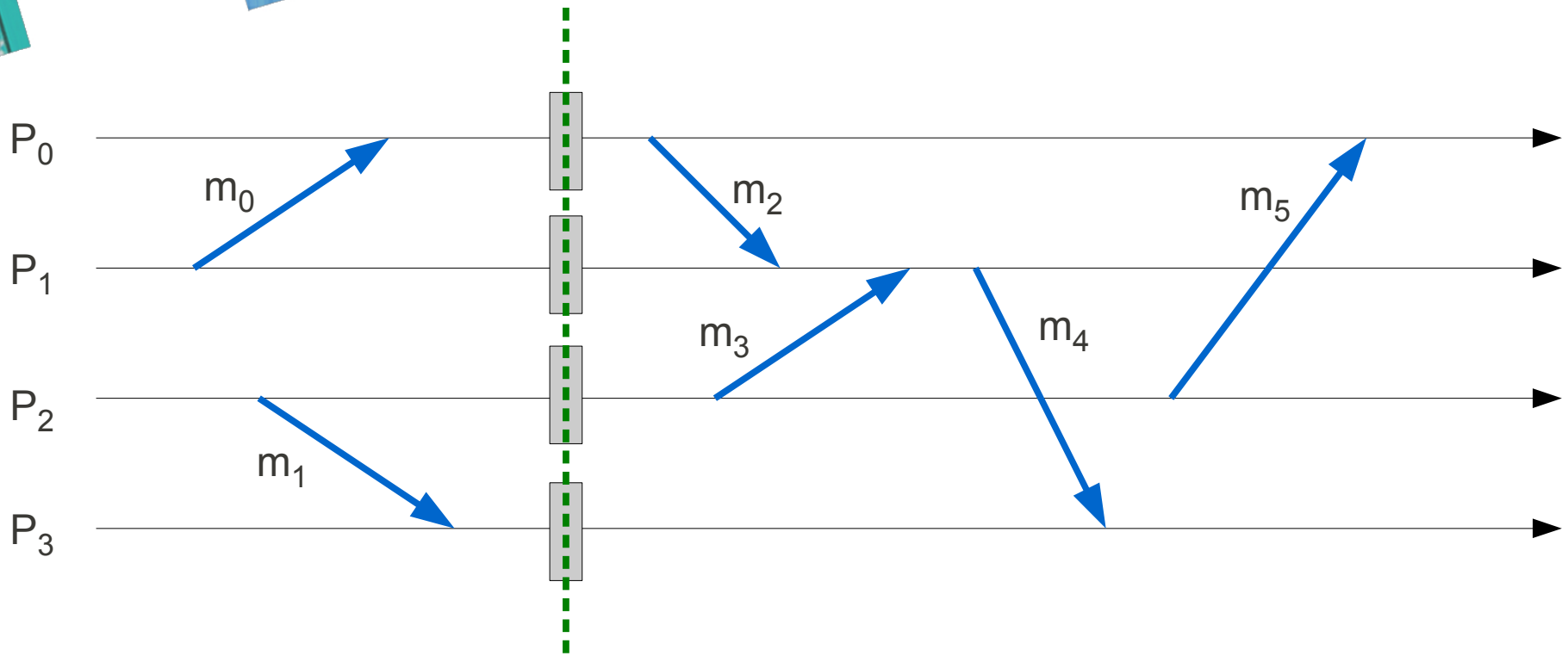
System Model

- MPI applications
  - Finite set of processes
- Asynchronous distributed system
  - Processes communicate by exchanging messages
    - → Causal dependencies between processes states (Lamport's happened-before relation)
  - FIFO reliable channels
- Fail-stop failure model
  - Multiple concurrent failures

Coordinated Checkpointing is not the Solution

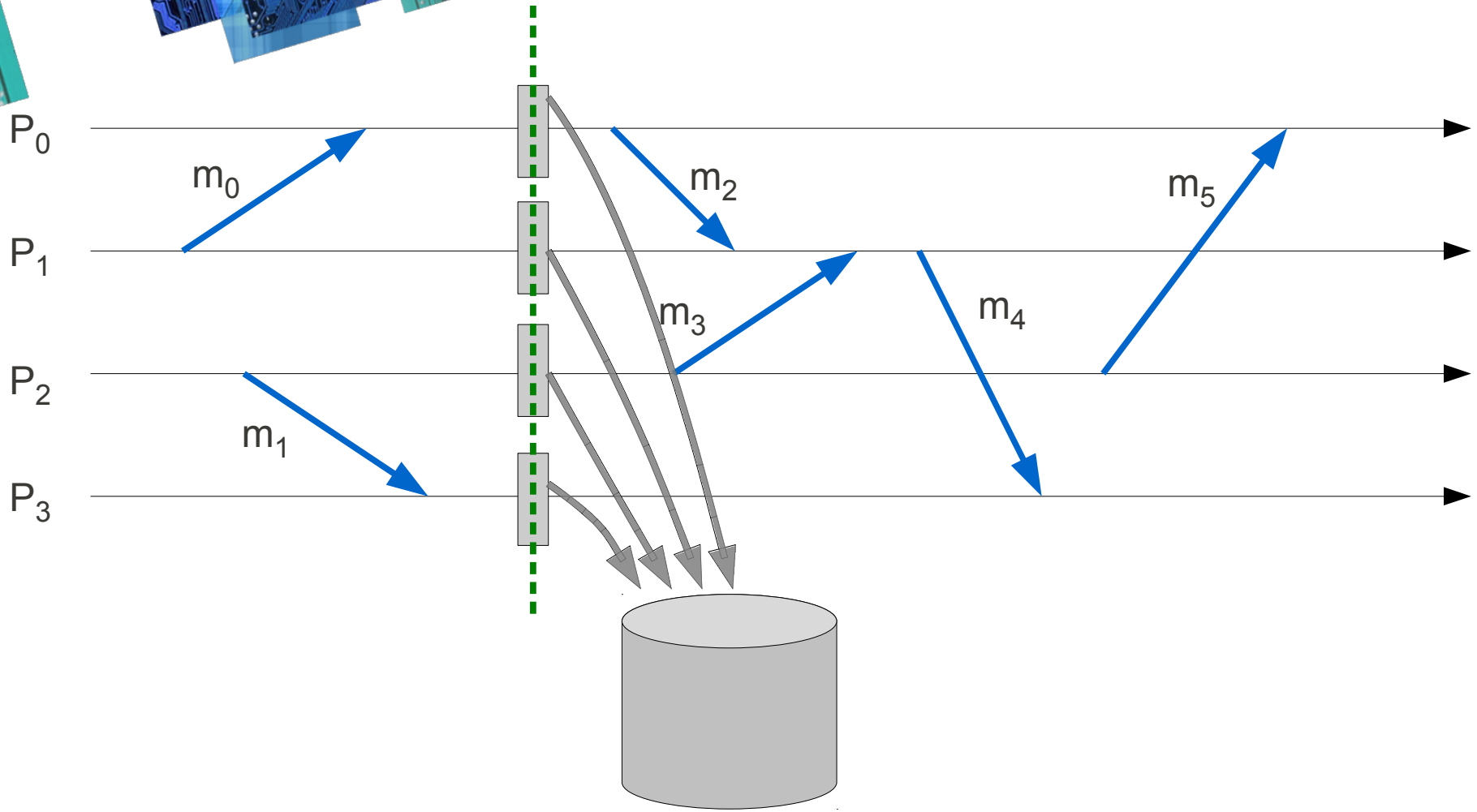


Coordinated Checkpointing is not the Solution



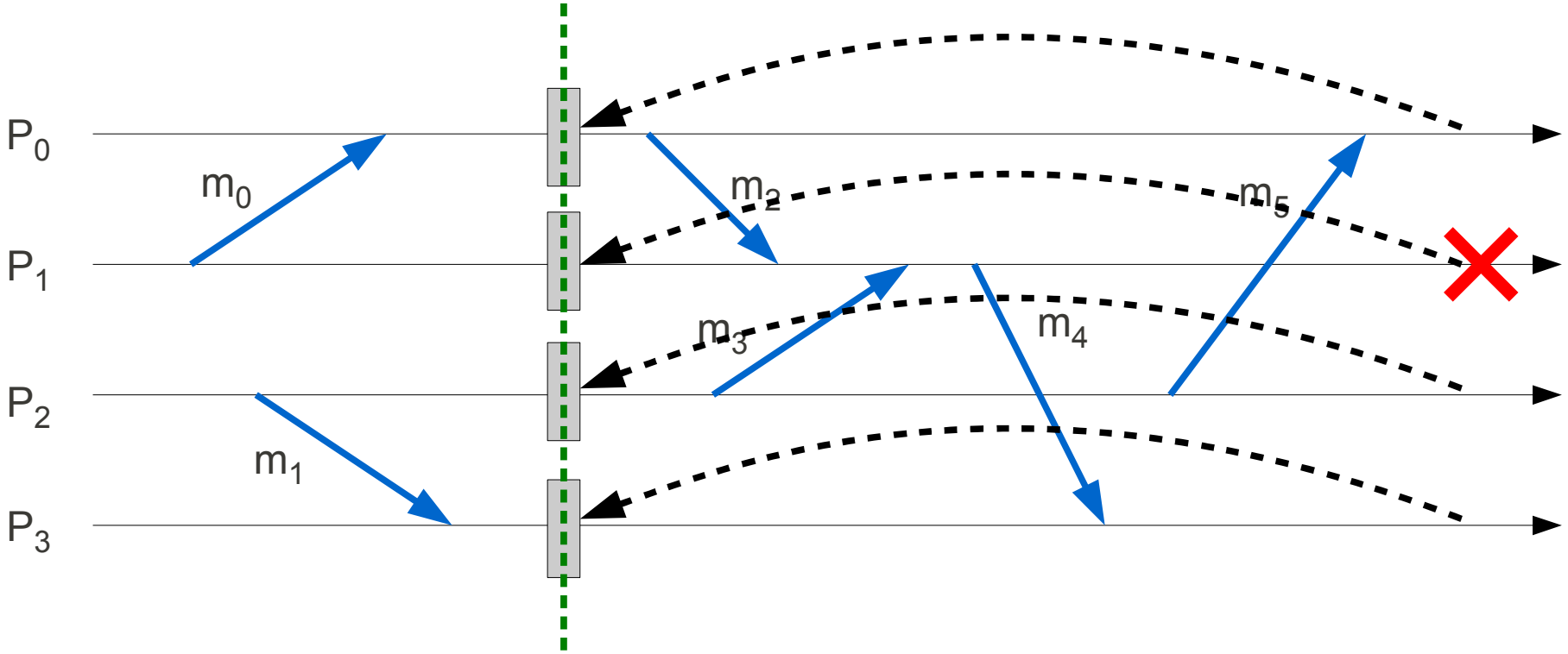
- Synchronization at checkpoint time to ensure a consistent global state
  - Easy to implement
  - Efficient garbage collection
  - Works for non-deterministic applications

Coordinated Checkpointing is not the Solution



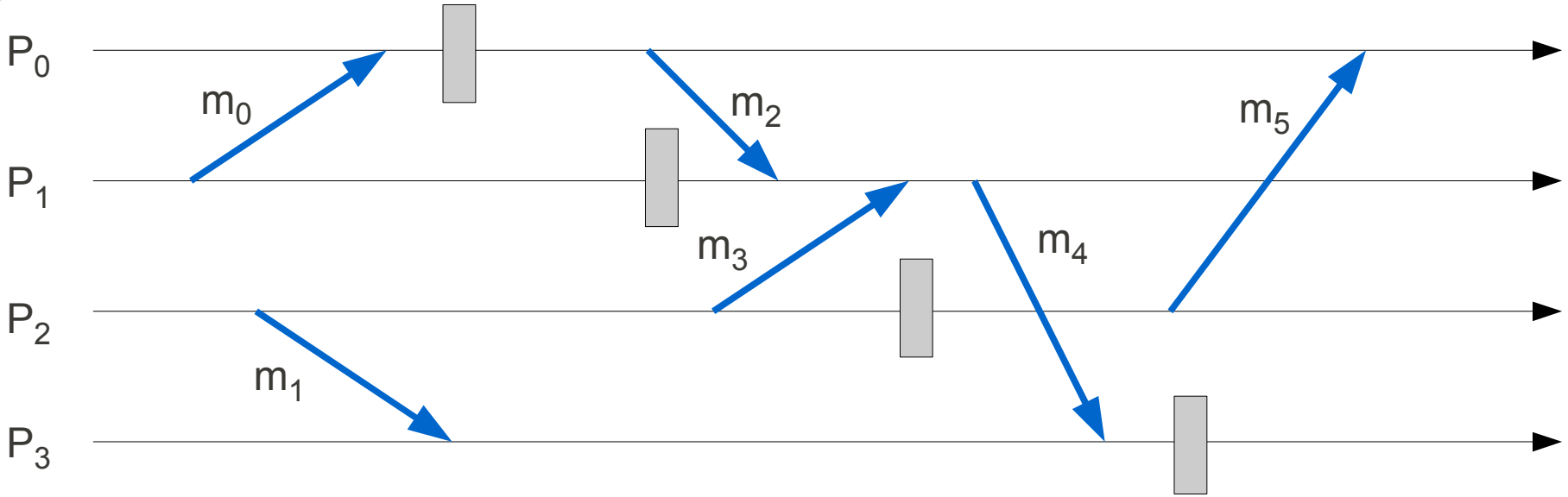
- All checkpoints are written *at the same time* on reliable storage
  - High stress of the file system

Coordinated Checkpointing is not the Solution



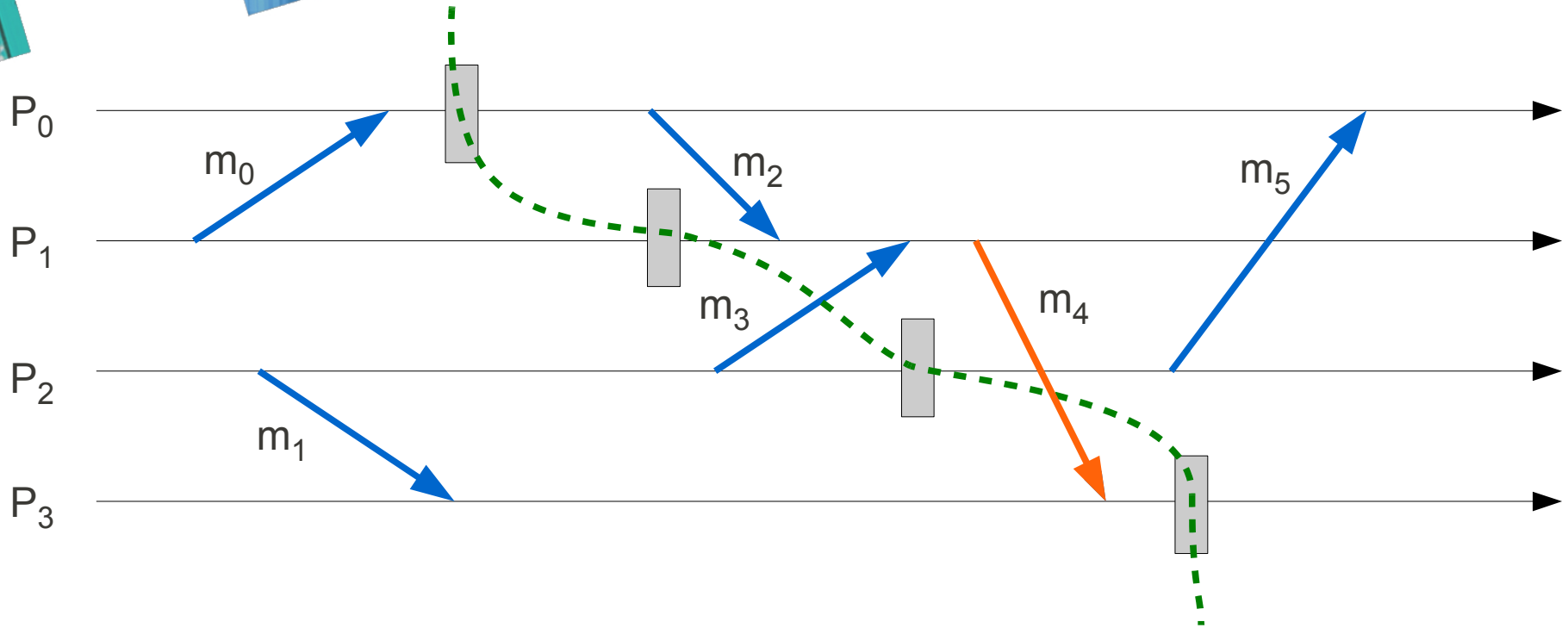
- One failure makes all processes rollback
  - Recovery is costly regarding power consumption

The Existing Alternatives also have Drawbacks



- Uncoordinated checkpointing
  - Checkpoints can be scheduled

The Existing Alternatives also have Drawbacks



- Uncoordinated checkpointing
  - Checkpoints can be scheduled
  - Suffers from the **domino effect**
    - Orphan messages have to be rolled back
    - Recovery and garbage collection is complex

The Existing Alternatives also have Drawbacks

- Message logging protocols
  - Can be combined with uncoordinated checkpointing without the domino effect
  - Only a subset of the processes have to rollback after a failure
  - Messages content and delivery event have to be saved

- Assumption
  - **piecewise deterministic** applications
    - → The only non-deterministic event are the messages reception event.

Many HPC Applications are Send-Deterministic

- Definition of Send-Determinism
  - Given a set of input parameters, sequences of message sendings are always the same in any correct execution
  - Messages reception order doesn't change processes behavior

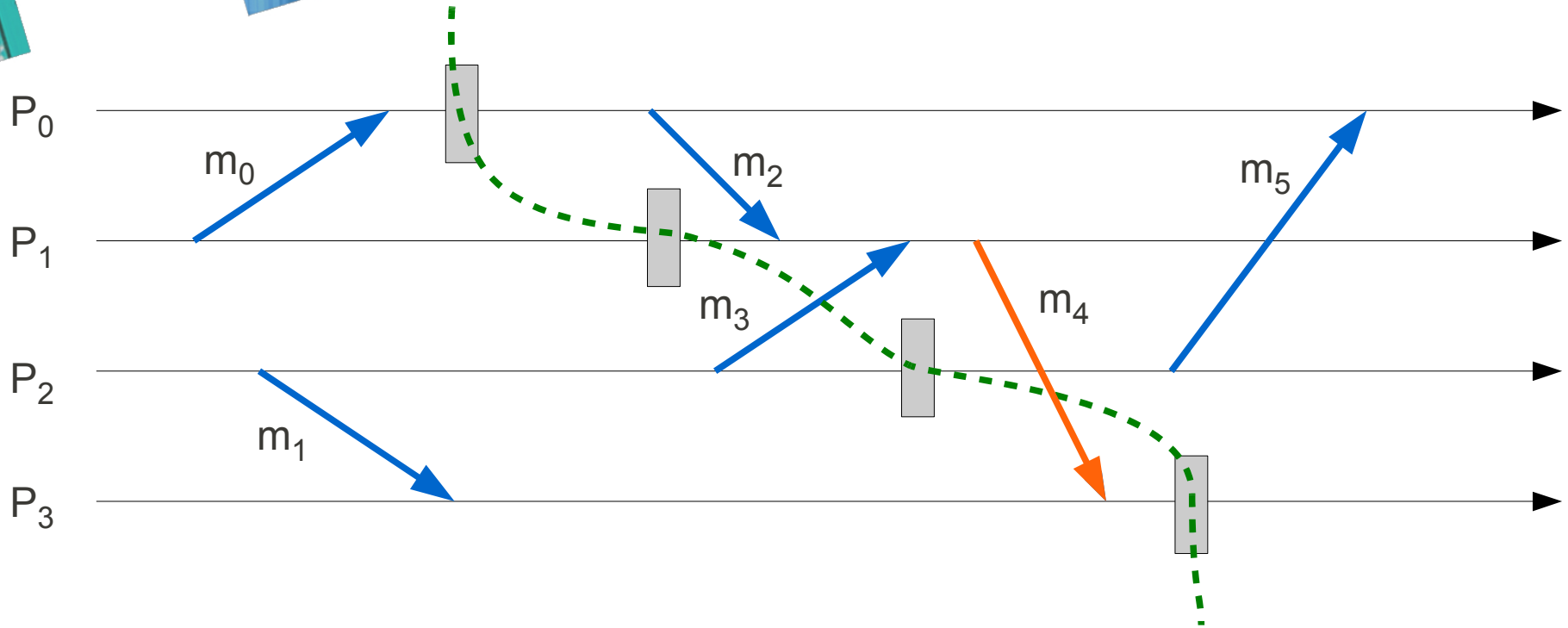
- Static analysis of 27 HPC applications (*Cappello 2010*)
  - NAS Benchmarks
  - 6 NERSC Benchmarks
  - 2 USQCD Benchmarks
  - 6 Sequoia Benchmarks
  - SpecFEM3D, Nbody, Ray2mesh
  - ScaLAPACK SUMMA

25 over 27 are send-deterministic

Contributions

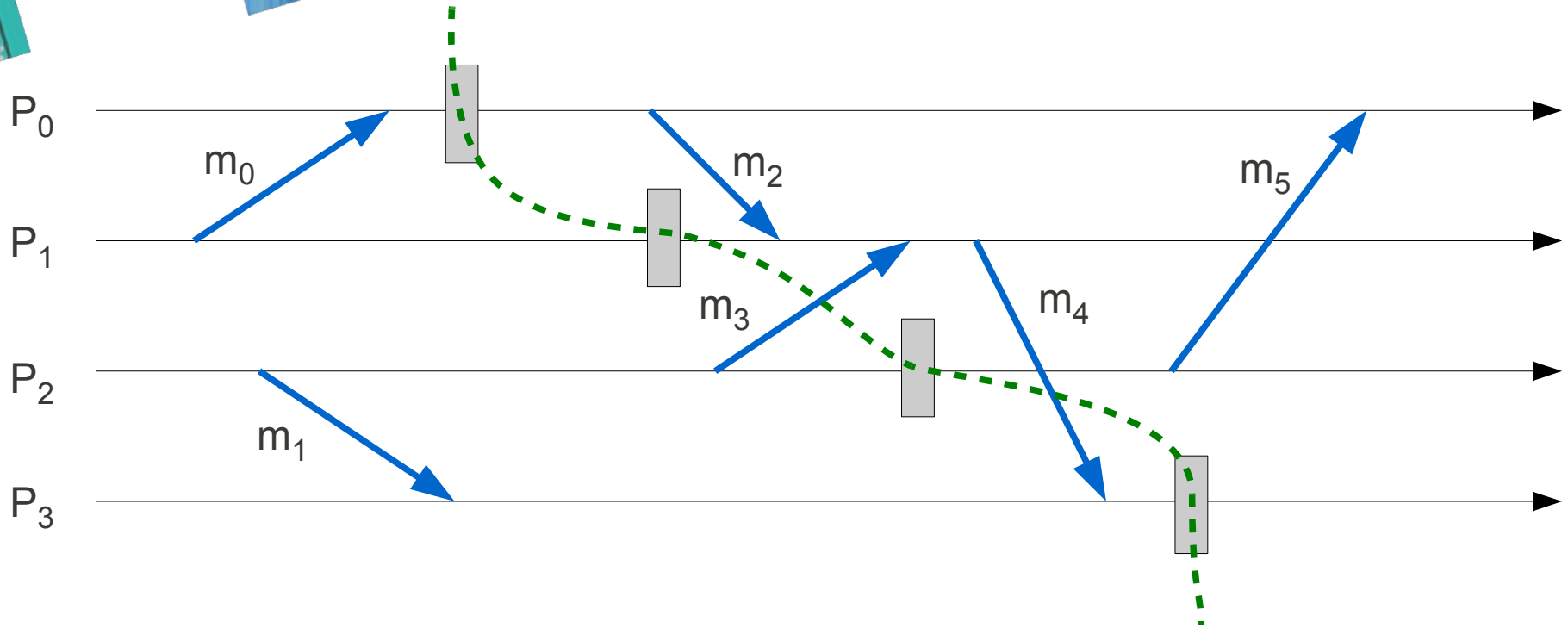
- An uncoordinated checkpointing protocol without domino effect
  - Small subset of logged messages
  - Allow partial restart
  - High performance on failure free execution (MX)
- Taking into account applications communications patterns
  - Improving the protocol using clustering techniques
    - 50% of rolled back processes on average
    - At most 50% messages logged
  - A new hierarchical cluster-based protocol
    - Message logging between clusters
    - Limit the number of rolled back processes to one cluster

Uncoordinated Checkpointing without Domino Effect



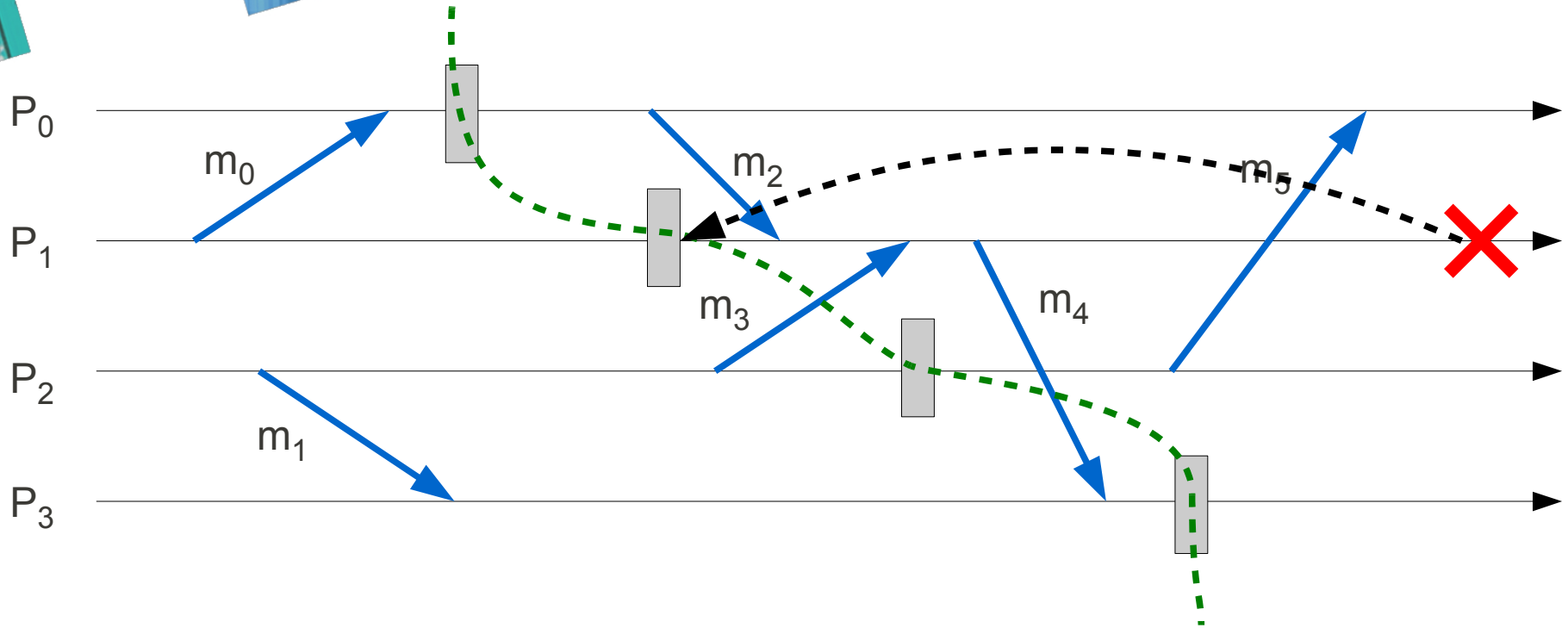
- Consequence of send-determinism:
  - Orphan messages don't need to be rolled back
  - The domino effect is avoided

Uncoordinated Checkpointing without Domino Effect



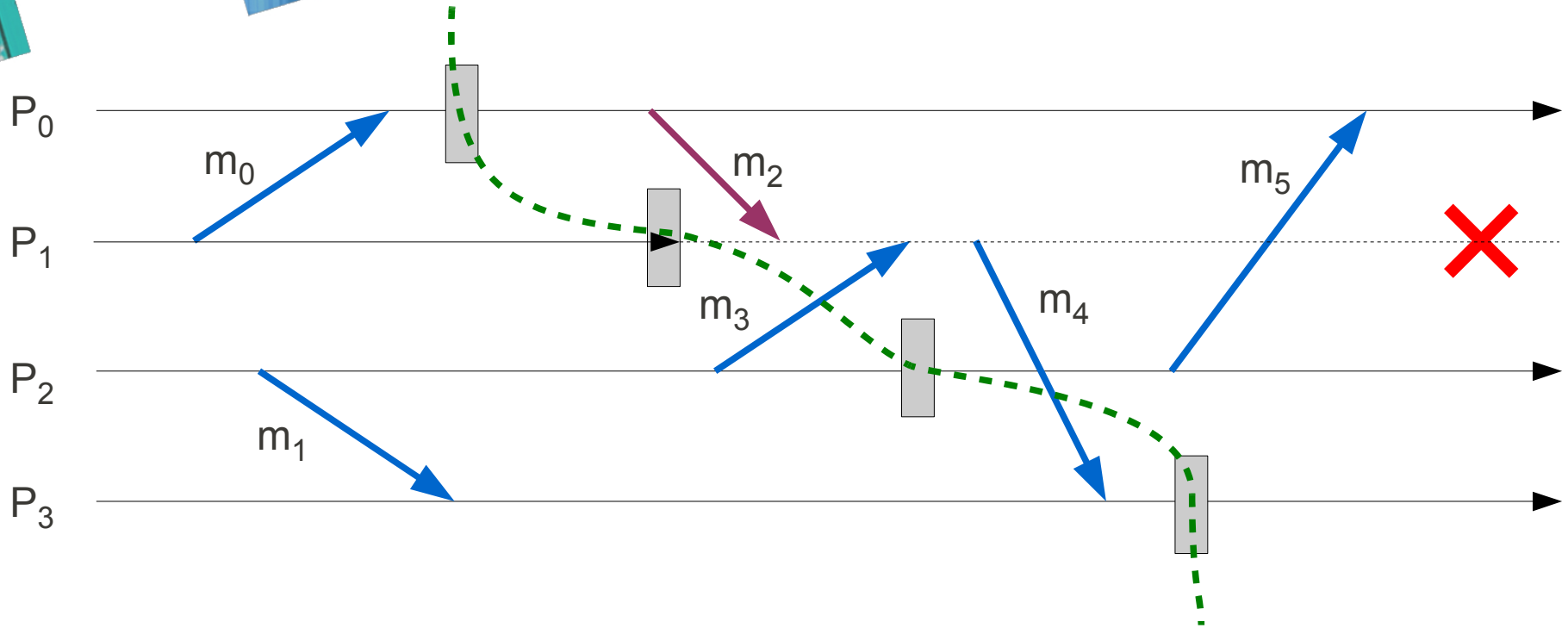
- Avoiding logging all messages
  - Processes roll back to send again the missing messages

Uncoordinated Checkpointing without Domino Effect



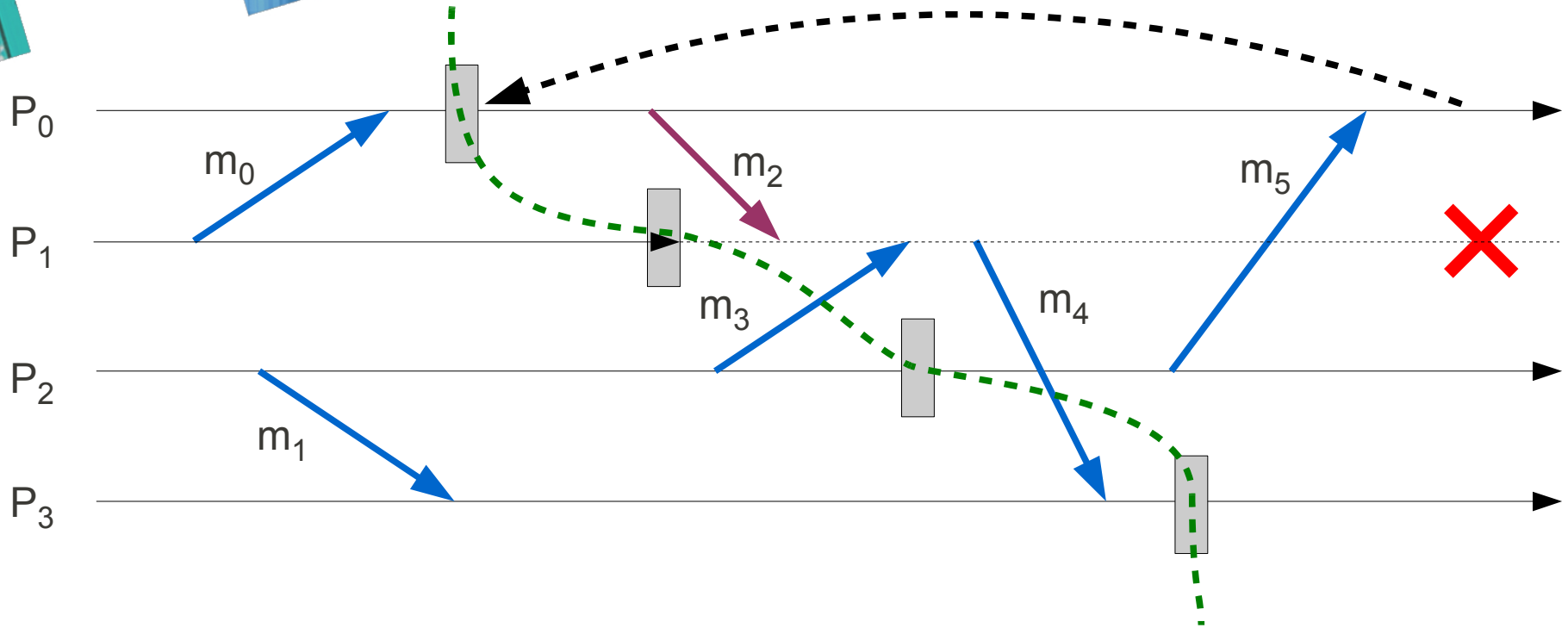
- Avoiding logging all messages
  - Processes roll back to send again the missing messages

Uncoordinated Checkpointing without Domino Effect



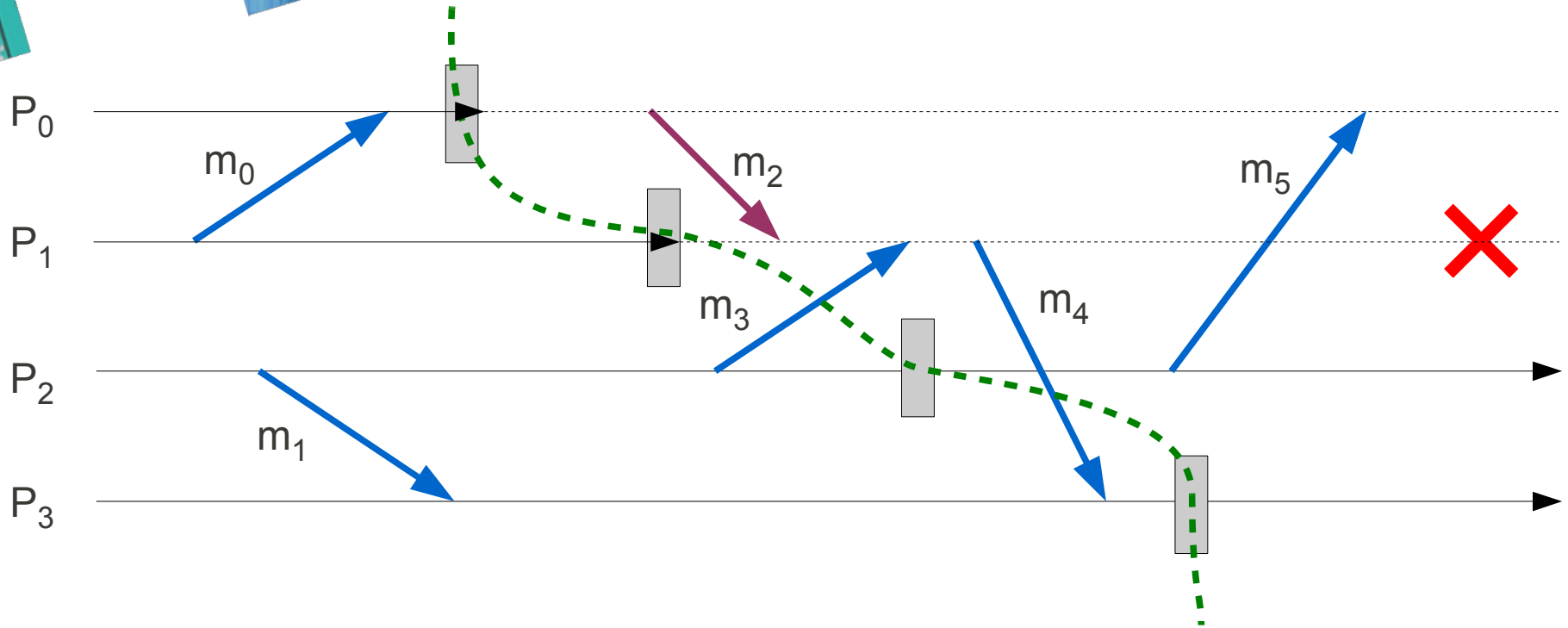
- Avoiding logging all messages
  - Processes roll back to send again the missing messages

Uncoordinated Checkpointing without Domino Effect



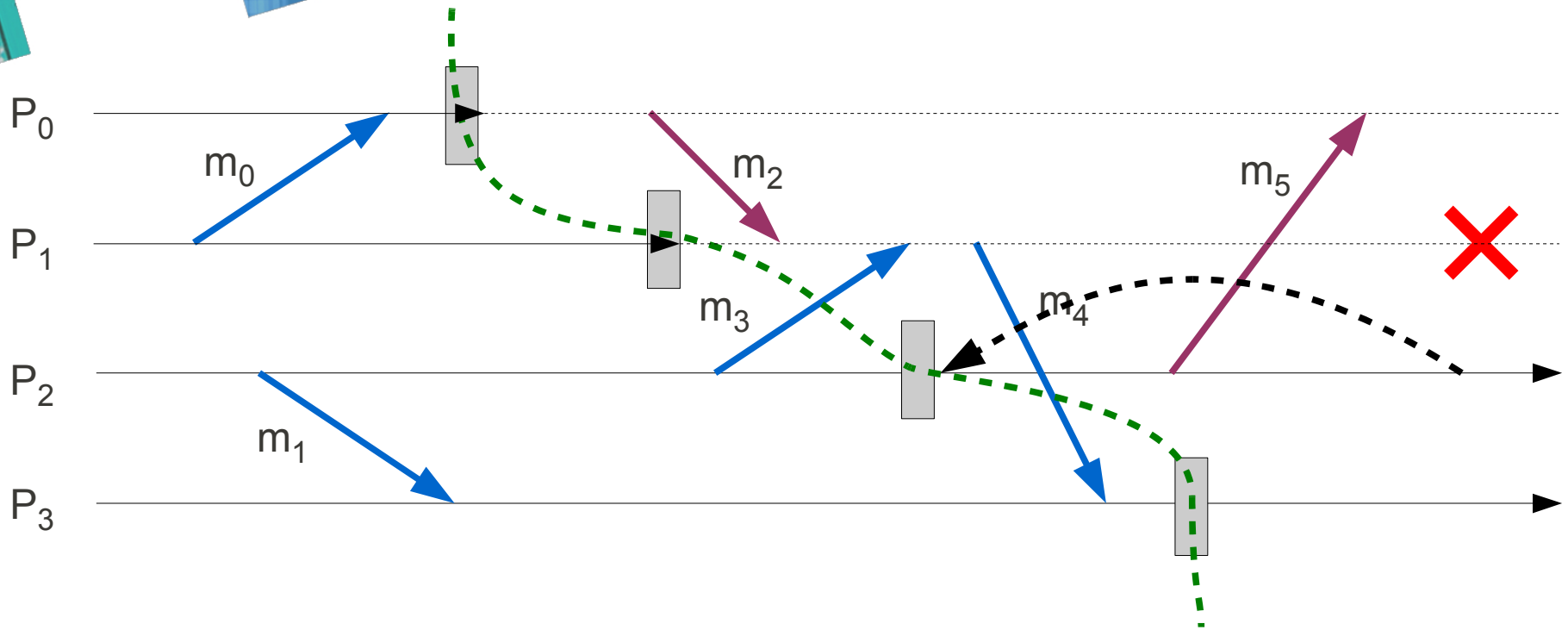
- Avoiding logging all messages
  - Processes roll back to send again the missing messages

Uncoordinated Checkpointing without Domino Effect



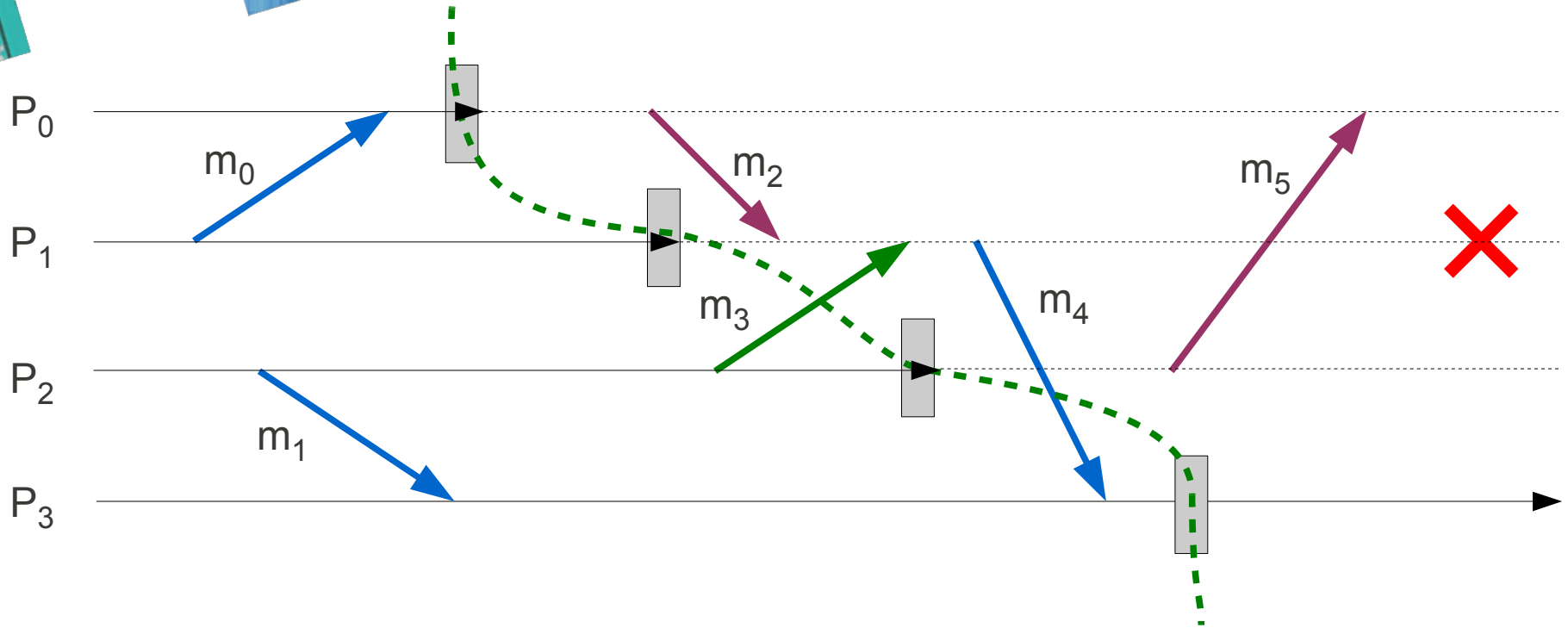
- Avoiding logging all messages
  - Processes roll back to send again the missing messages

Uncoordinated Checkpointing without Domino Effect



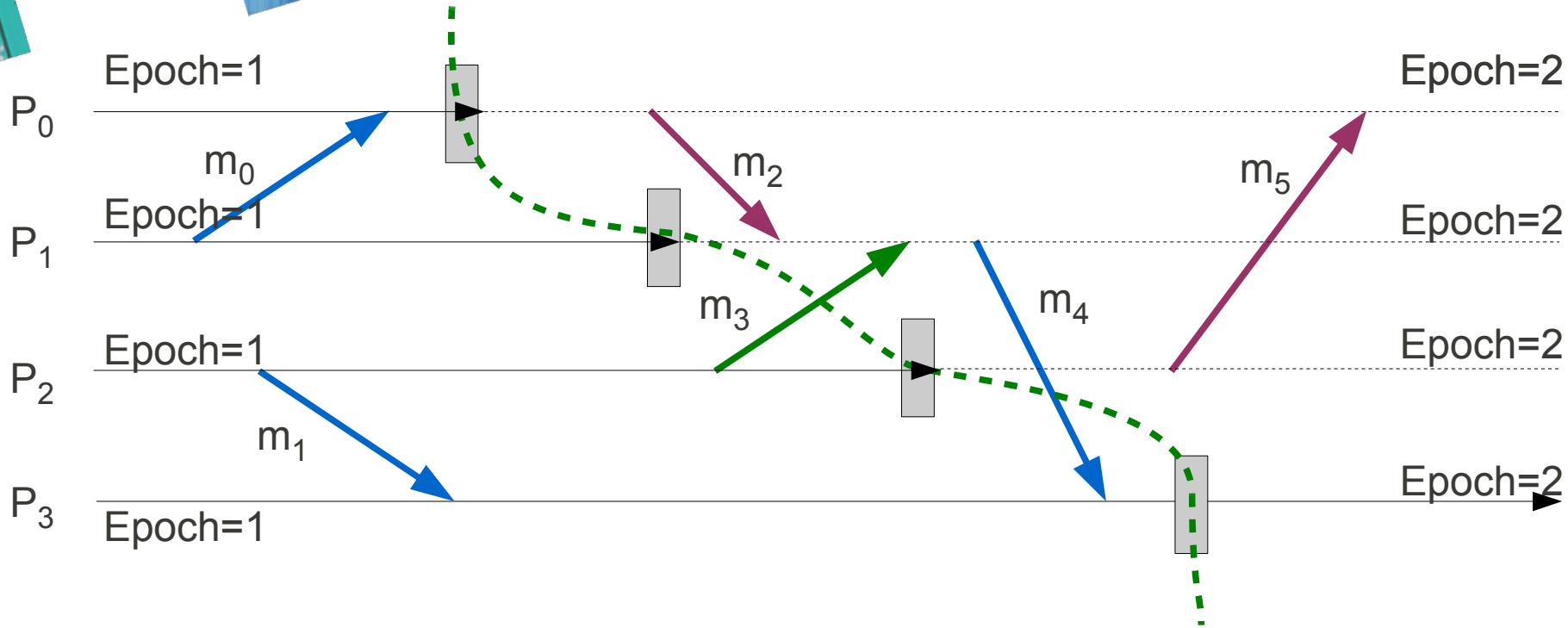
- Avoiding logging all messages
  - Processes roll back to send again the missing messages

Uncoordinated Checkpointing without Domino Effect



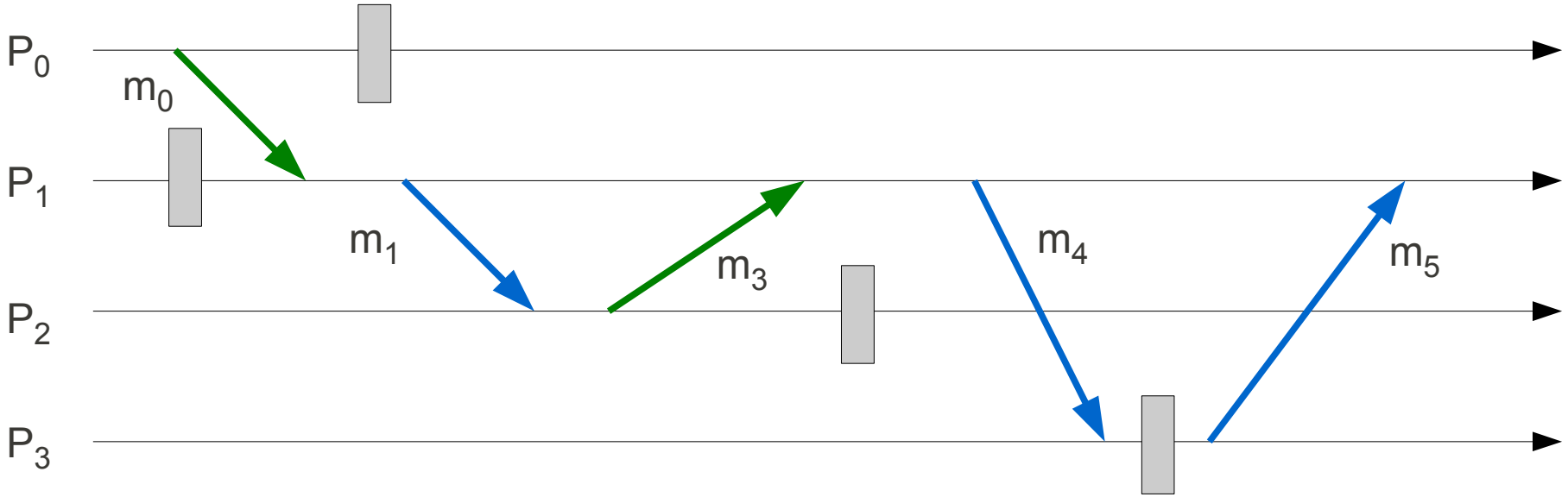
- Logging messages that could lead to a domino effect
  - Sender-based message logging

Uncoordinated Checkpointing without Domino Effect

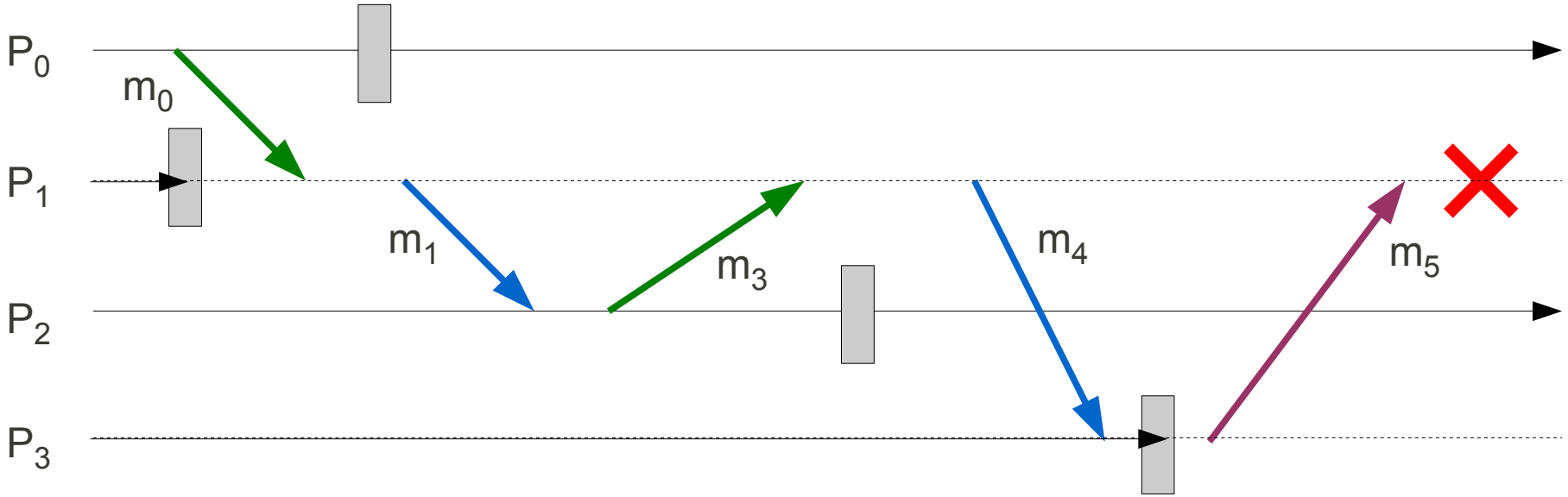


- Logging messages that could lead to a domino effect
  - Sender-based message logging
  - Using *Epoch* numbers
    - ➔ Sender Epoch < Receiver Epoch

Managing Causal Dependencies During Recovery

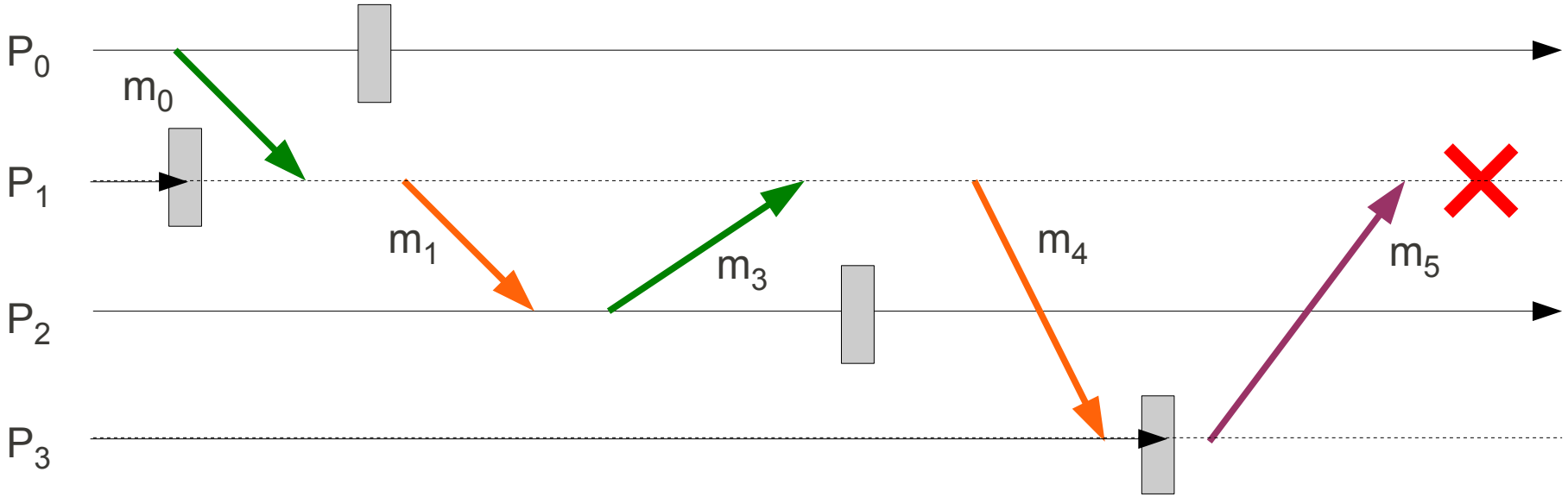


Managing Causal Dependencies During Recovery



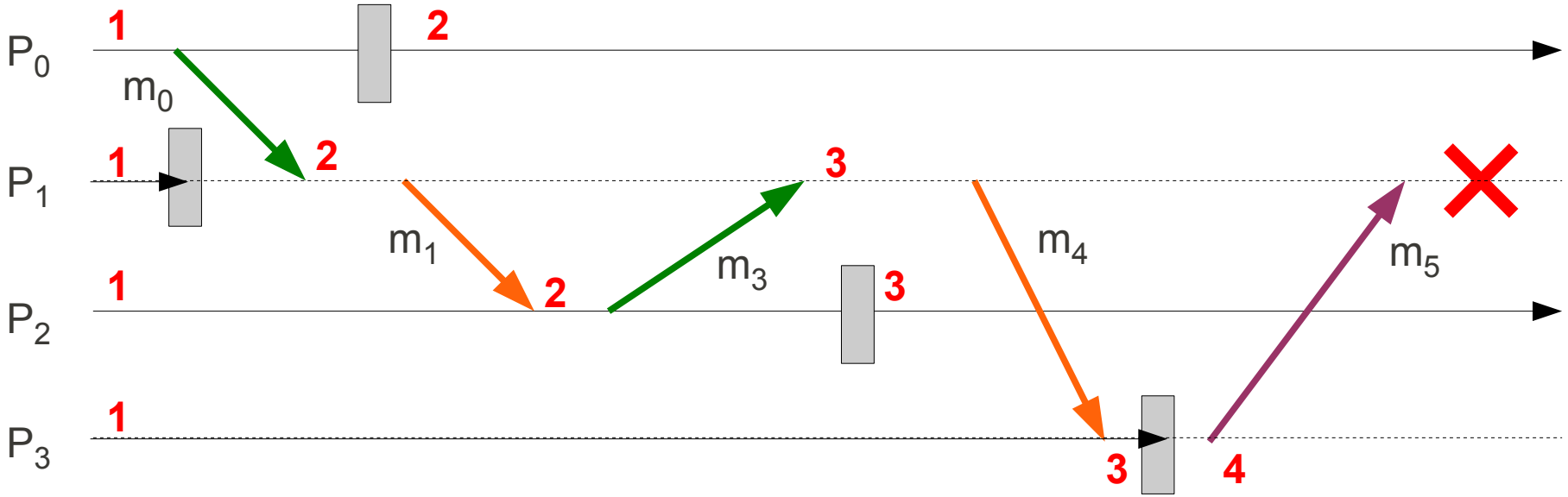
- Restarting from an inconsistent state
  - Messages that are causally dependent could be sent at the same time

Managing Causal Dependencies During Recovery



- Restarting from an inconsistent state
  - Messages that are causally dependent could be sent at the same time
- Causal dependency paths are *broken* by:
  - Checkpoints
  - Logged messages

Managing Causal Dependencies During Recovery



- Using **phase** numbers to show when a causality path is broken
  - Similar to *Lamport clocks*
    - ➔ Incremented when a causality path is broken
  - Allows to order causally dependent messages

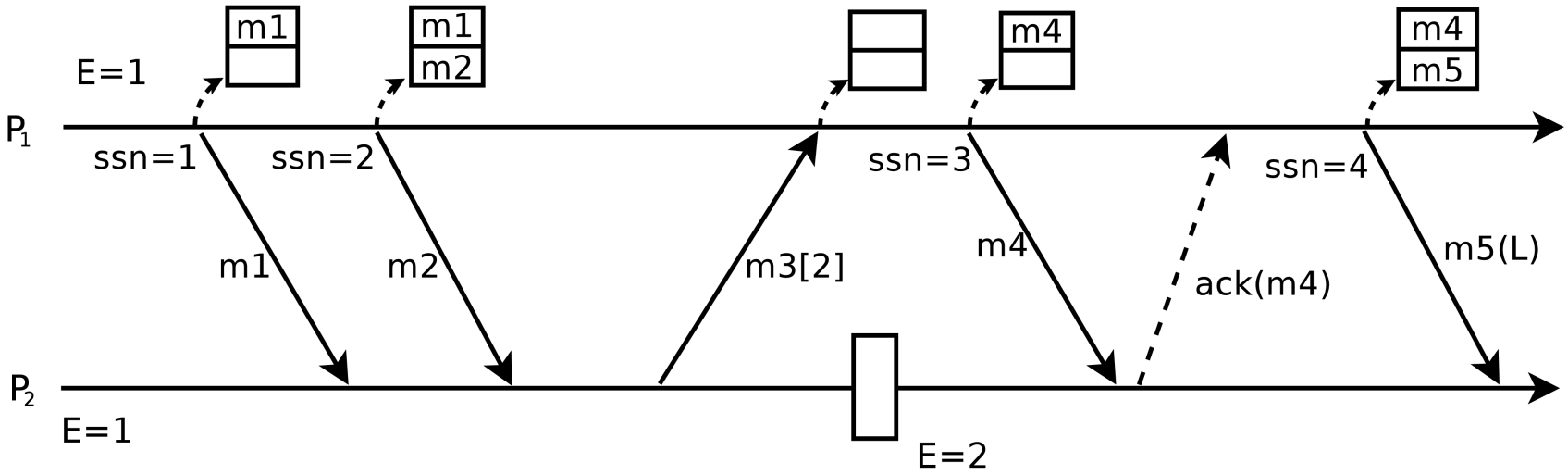
Our Prototype in MPICH2

- Communication management in CH3/Nemesis
  - Implementation over TCP and Myri-10G (MX)
  - Message logging

- Rollback-recovery management in HYDRA (process manager)
  - Uncoordinated process checkpointing (BLCR)
  - Computation of the set of processes to rollback
    - Using a centralized process
  - Process restart (ongoing work with MPICH2 team)
    - Restarting one process without restarting the application

Message Logging

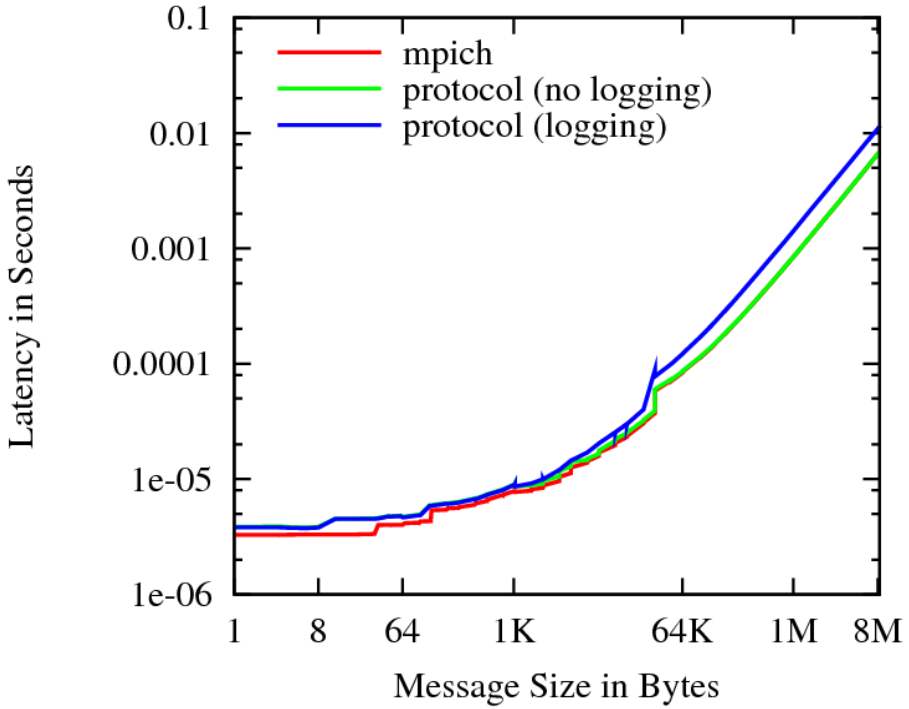
- Using acknowledgements to detect messages to log
  - Compare the epoch of the sender and the epoch of the receiver
  - Sending an ack for every message is too costly for latency
- Optimized implementation
  - Small messages (< 1 KB)
    - Messages content are copied without waiting for the ack
    - Acks are piggybacked on messages
    - Only logged messages generate an explicit ack

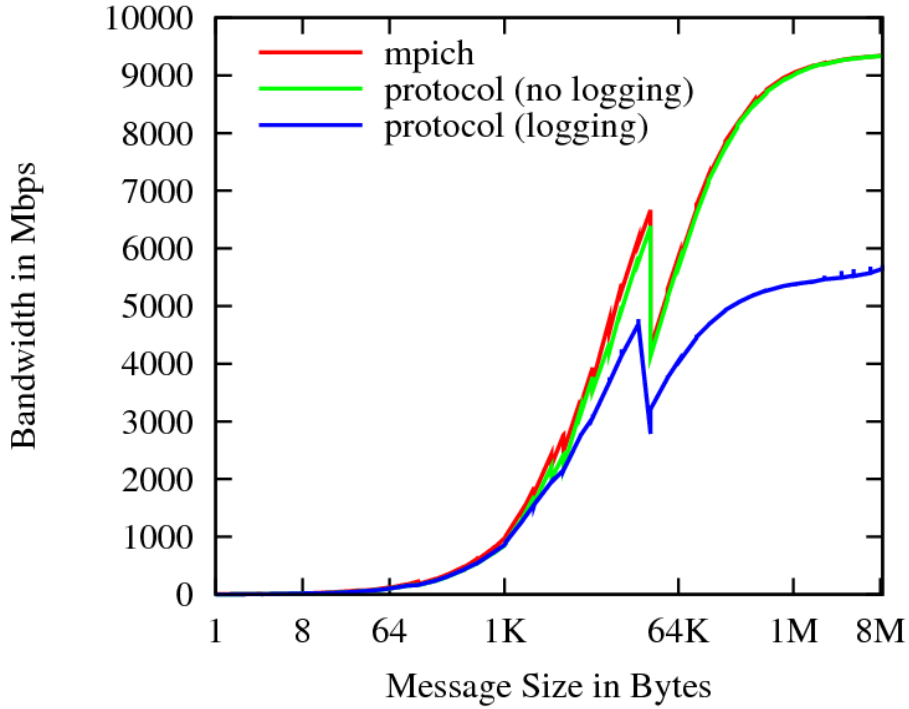


Experimental Setup

- Lille cluster of Grid5000
  - 45 nodes
  - 2 Intel Xeon E5440 QC processors
  - 8 GB of memory
  - 10G-PCIE-8A-C Myri-10G NIC
  - Linux kernel 2.6.26
- NetPipe Ping-Pong test over MX
  - Latency
  - Bandwidth
- Performance evaluation for 3 NAS benckmarcks

Performance Evaluation

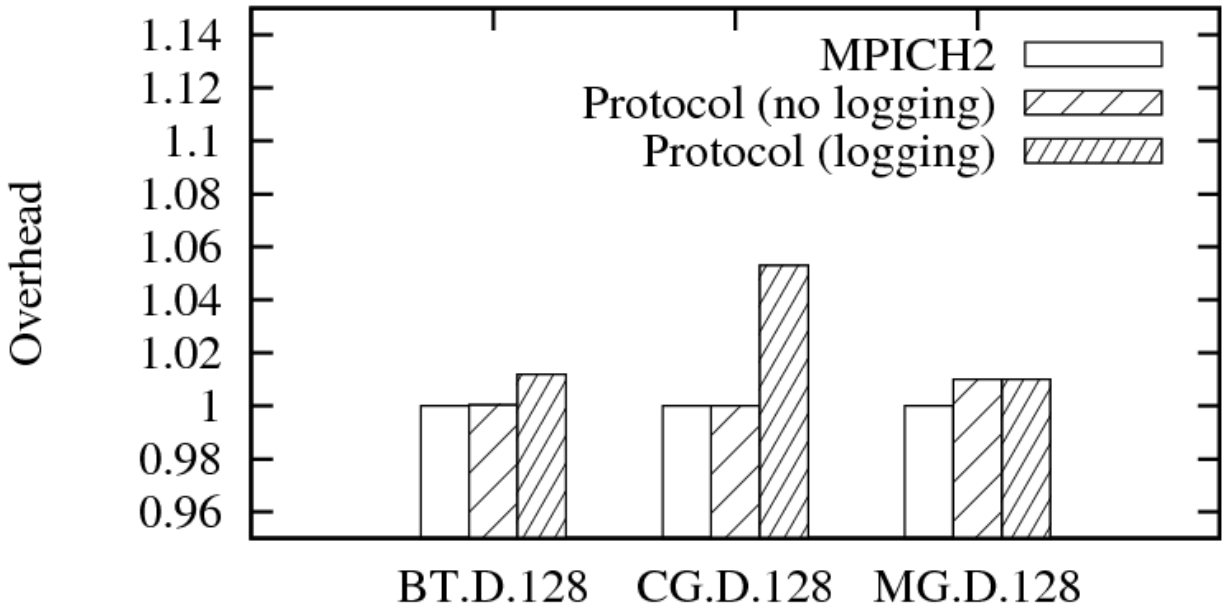




- At most 0.5 µs (15%) overhead on latency for small messages
- 40% bandwidth reduction for large messages (logging)

Performance Evaluation

- NAS Performance over MX
  - Almost no impact without logging
  - At most 5% overhead when all messages are logged



Computation of the recovery line

- Off-line computation
  - Processes flush data about messages they send every 30s
  - Off-line computation considering all failure scenario
    - → For 1 failure

- 6 NAS Benchmarks, class D, 128 processes
  - All processes roll back in almost every case

Conclusion (First Part)

- An uncoordinated checkpointing protocol for send-deterministic applications
  - No domino effect (easy garbage collection)
  - Few messages logged
  - Allow partial restart
- Prototype Evaluation
  - Good performance on failure free execution (MX)
  - All processes roll back in case of one failure
- Technical report
  - Full description of the protocol
  - Proof
- Summary
  - Avoid checkpoint coordination
  - **Costly on recovery**

Taking Into Account Communications Patterns

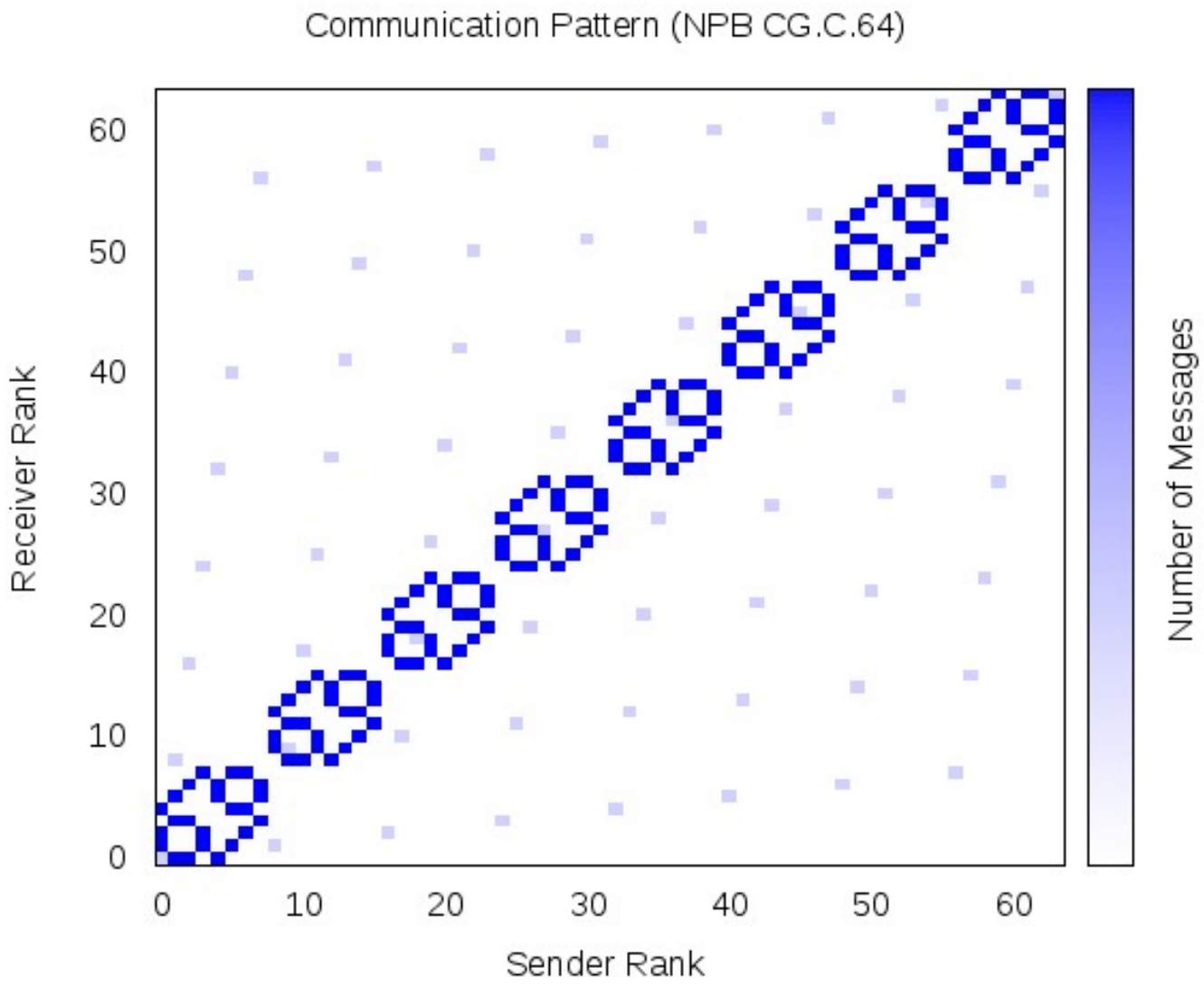
- Improving our uncoordinated protocol using process clustering

- Ongoing work
  - A new hierarchical checkpointing protocol based on process clustering for send-deterministic applications

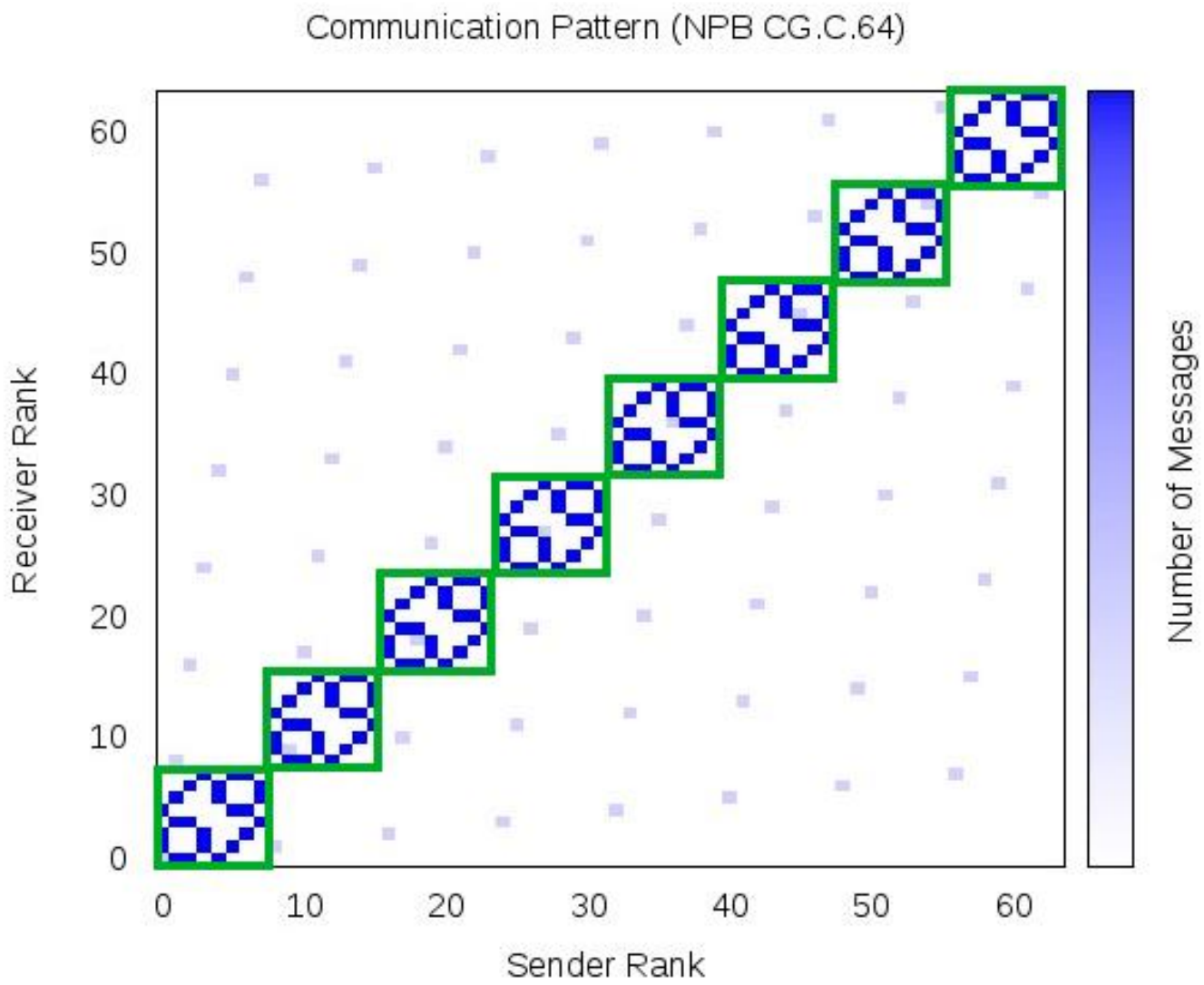
Improving our Protocol

- Our protocol: logging based on epochs
  - Log messages going from epoch E1 to epoch E2 if E1 < E2
- Basic idea
  - Create clusters of processes
  - Force message logging between clusters using different epoch numbers
  - Take into account communication patterns to minimize the number of logged messages

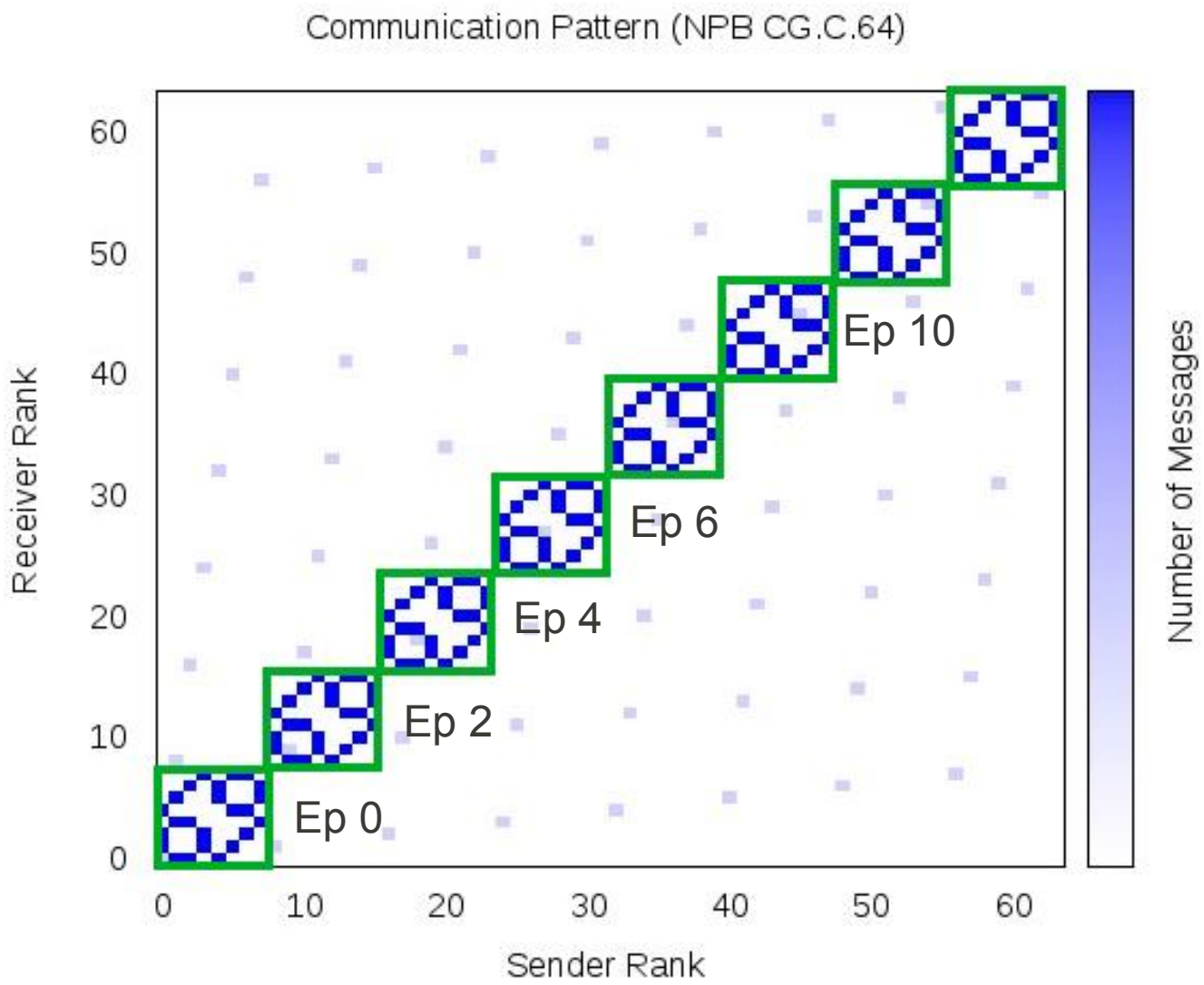
Process Clustering



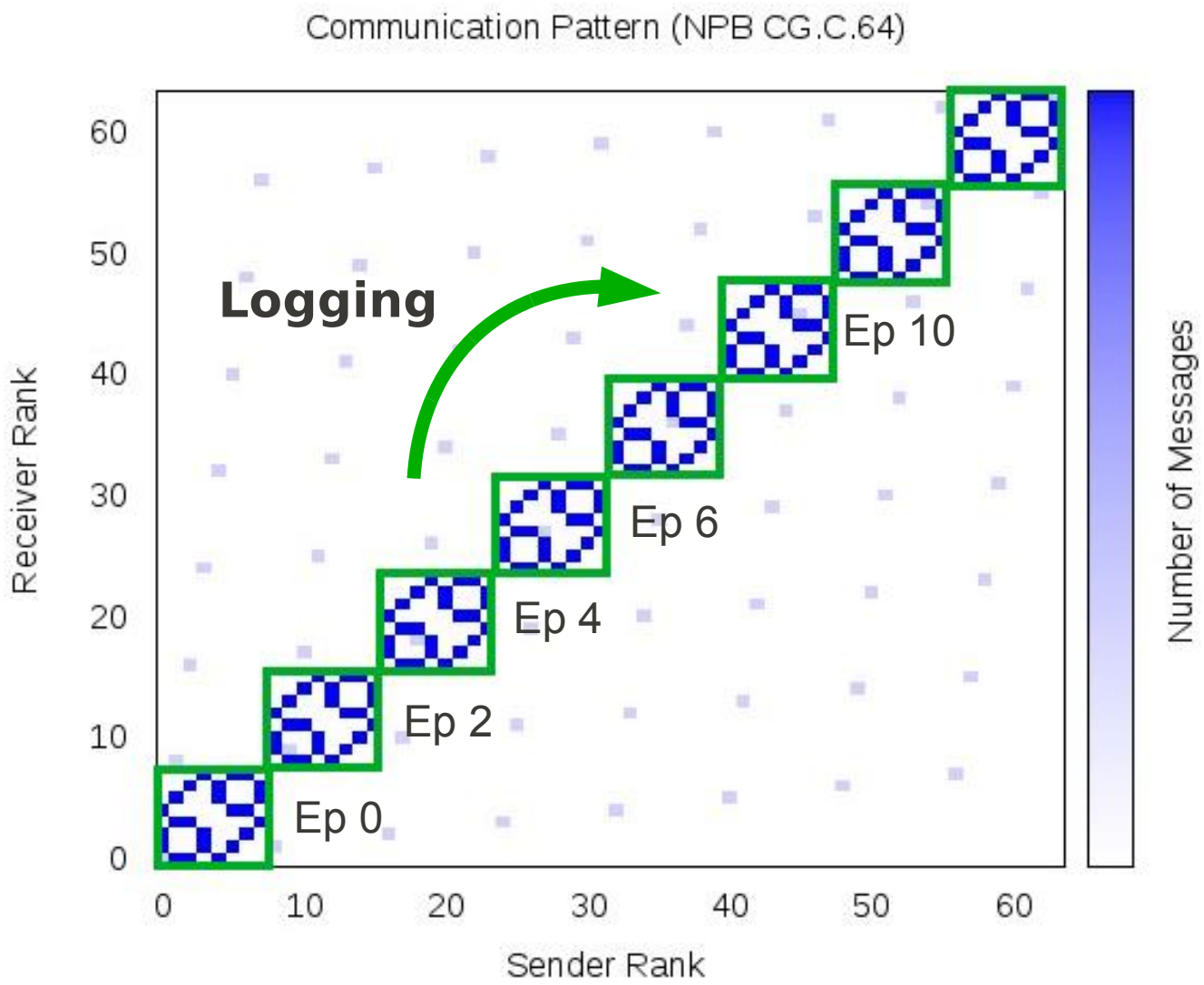
Process Clustering



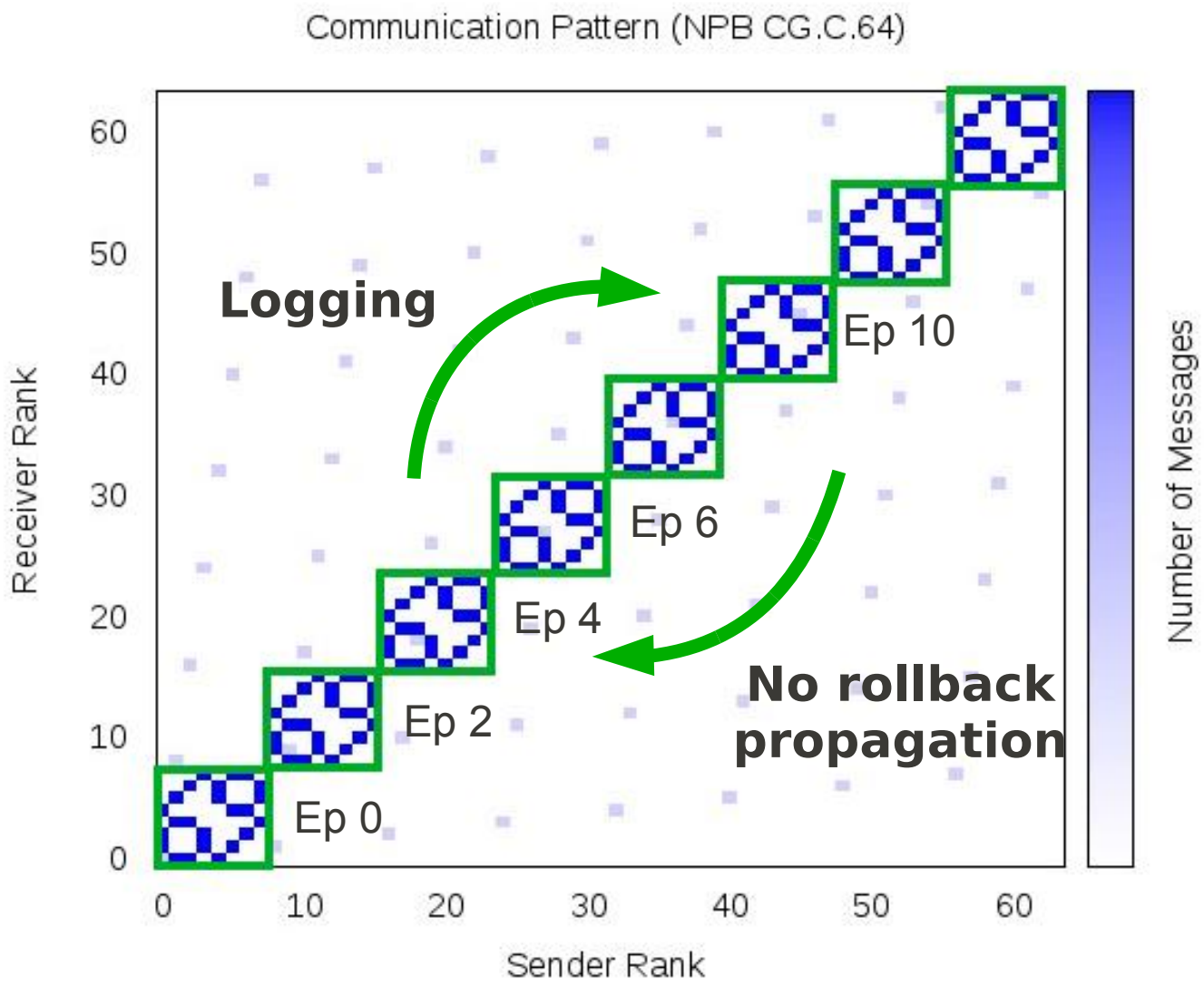
Process Clustering



Process Clustering



Process Clustering



Theoretical Limits

- Average number of cluster to rollback (p clusters)
  - Failure in 1st cluster → p clusters rollback
  - Failure in 2nd cluster → p-1 clusters rollback ...
  - Failure in last cluster → 1 cluster rollback

(P+1)/2 on average

- Maximum number of logged messages
  - A: set of intra-cluster messages
  - B: set of logged inter-cluster messages
  - C: set of non-logged inter-cluster messages
    - → If B > 50% then C < 50%

Less than 50%

Experimental Results

Cluster Size	32		16		8	
	%log	%rl	%log	%rl	%log	%rl
BT	13	62.6	25.2	56.4	36.7	53.3
CG	2.9	62.5	3.4	56.3	15	43.8
FT	37.3	62.5	43.6	56	46.8	53
LU	10.3	62.5	24.1	56.3	25.9	42.1
MG	9.5	62.5	17.1	56.3	25.4	42.1

Class D NAS Benchmarks,128 processes

Experimental Results

Cluster Size	32		16		8	
	%log	%rl	%log	%rl	%log	%rl
BT	13	62.6	25.2	56.4	36.7	53.3
CG	2.9	62.5	3.4	56.3	15	43.8
FT	37.3	62.5	43.6	56	46.8	53
LU	10.3	62.5	24.1	56.3	25.9	42.1
MG	9.5	62.5	17.1	56.3	25.4	42.1

Class D NAS Benchmarks, 128 processes

Experimental Results

Cluster Size	32		16		8	
	%log	%rl	%log	%rl	%log	%rl
BT	13	62.6	25.2	56.4	36.7	53.3
CG	2.9	62.5	3.4	56.3	15	43.8
FT	37.3	62.5	43.6	56	46.8	53
LU	10.3	62.5	24.1	56.3	25.9	42.1
MG	9.5	62.5	17.1	56.3	25.4	42.1

Class D NAS Benchmarks, 128 processes

Conclusion

- An uncoordinated checkpointing protocol for send-deterministic applications
  - No domino effect
  - Avoids process synchronization
  - Allows partial restart
  - Provides good performance of failure free execution (MX)

- Process clustering
  - Limit the number of process to roll back to almost 50% on average
    - → Reducing energy consumption
  - Logging only a small amount of messages (max 50%)

Ongoing Work

- A hierarchical protocol based on process clustering
  - Message logging between clusters
    - Limit the consequences of a failure to one cluster
  - Based on send-determinism
    - The logged messages are still valid after a rollback
    - Phase numbers are needed to deal with causal dependencies
  - Coordinated or uncoordinated checkpointing inside a cluster
- Implementation in MPICH2
- Collaboration with Charm++ team (Esteban Meneses, Zhihui Dai)
  - Considering a single failure
  - First prototype working

Joint Laboratory for Petascale Computation

INRIA NCSA

Rollback-Recovery Protocols for Send-Deterministic Applications

Amina Guermouche, Thomas Ropars, Elisabeth Brunet,
Marc Snir and Franck Cappello