Toward Exascale Resilience

Part 6: System protection with checkpoint/restart

Mattan Erez
The University of Texas at Austin

July 2015

- Hardware reliability never perfect
  - Not worth the cost
    - Though can get close with high cost
- Software also has errors
  - Can never fully debug

Application and system must go on

Detect→contain→repair→recover

- Detection is most critical piece
  - We’re in luck – silent-data-corruption very rare
  - Strong ECC
  - Not very vulnerable logic
- The paranoid among us aren’t convinced though
  - More when we talk about cross-layer approaches

What to do on detected errors?

– Simplest idea: failstop!
  - Contain the error and don’t let it propagate

If system stopped, can the application continue?

- Yes, if prepared
- Checkpoint/restart

- Periodically, take checkpoint
  - Stop the system
  - Copy *all* state somewhere sage
  - Keep going
- Rollback
  - Recover saved state
- Restart
  - Recompute and keep going

Things to think about (outline)

- How frequently to checkpoint?
- How important is checkpoint time?
  - And what can we do to improve it
- Who decides to take a checkpoint?
  - System or user
- Do we really have to stop everyone to take a checkpoint?
  - Coordinated vs. uncoordinated checkpointing
  - Always (for now) coordinated restart

Great survey paper:

ElNozahy et al., “A Survey of Rollback-Recovery Protocols in Message-Passing Systems” (http://www.cs.utexas.edu/~lorenzo/papers/SurveyFinal.pdf)

How often should we take a checkpoint?

- Want to maximize efficiency
- Too frequent – time wasted on checkpointing
- Too infrequent – time wasted on re-execution
- Can we optimally balance the two?
  - Sure, let’s see how

What determines CP/RS effectiveness?

- How long between failures (failstops)
- How long to repair and recover
- How long to take checkpoint

Time between failures (AMTTI)

- Not up to CP/RS scheme – system parameter
- Currently ~5 hours and trending down

Time to repair

- If spare nodes: repair takes seconds (at most)
- If no spare nodes: repair hidden by other jobs
  - Deallocate and reallocate failed job
- From system perspective, repair time very short

Time to recover

- Read state back in
  - Usually similar to taking checkpoint
- Re-execute all lost work

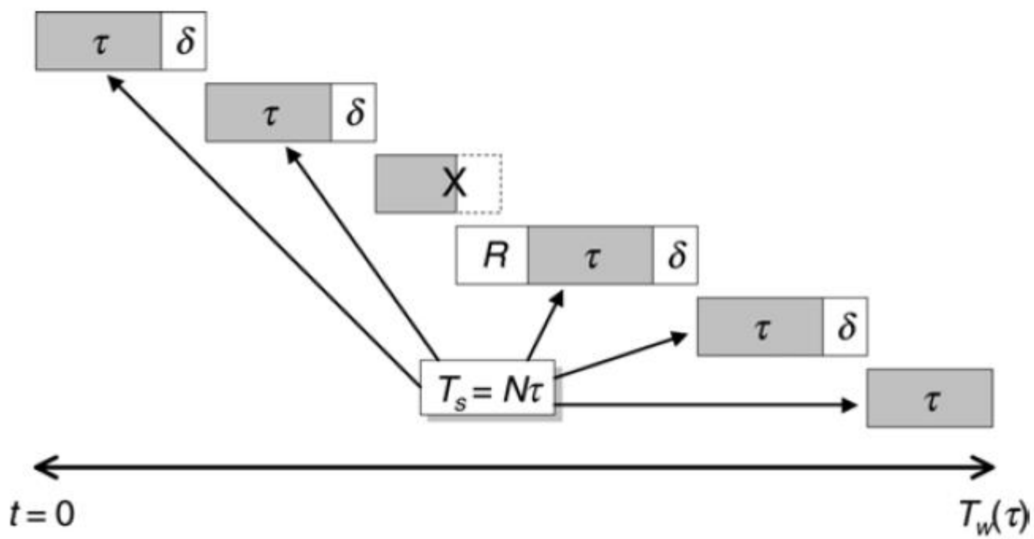
Time to take the checkpoint

- Stop the system
  - Depends on technique used, but can be fast
- Copy state somewhere safe
  - ~25**M**B/s per node typical global file system BW
    - Worse when many nodes write together
  - ~100GB per node
- Continue execution

- Kind of long, let's see what that means

Estimating execution time with CP/RS

- Excellent paper by J. Daly, Future Generation Computer Systems 22 (2006)
  - Took figures and equations from that paper



Total = solve + checkpoint + reexec + repair

First order model

- No interrupts during checkpoint or recovery
- Interrupts occur in the middle of an interval (expected)
- Poisson process for interrupts

Total, Ideal, # checkpoints

$$T_w(\tau) = T_s + \left(\frac{T_s}{\tau} - 1\right)\delta$$

$$+ [\tau + \delta]\phi(\tau + \delta)n(\tau) + Rn(\tau)$$

recover time, repair, # interrupts

When Poisson and when checkpoint interval << MTTI

then:

$$n(\tau) = \frac{T_s}{\tau}(e^{(\tau+\delta)/M} - 1)$$

$$\cong \frac{T_s}{\tau}\left(\frac{\tau+\delta}{M}\right) \quad \text{for } \frac{\tau+\delta}{M} \ll 1$$

First-order equation:

$$T_w(\tau) = T_s + \left(\frac{T_s}{\tau} - 1\right)\delta + \left[\frac{1}{2}(\tau+\delta) + R\right]\frac{T_s}{\tau}\left(\frac{\tau+\delta}{M}\right)$$

Minimize time (control interval)

$$-\frac{1}{\tau^2}(2\delta M + 2\delta R + \delta^2) + 1 = 0$$

$$\tau_{\text{opt}} = \sqrt{2\delta(M+R)} \quad \text{for } \tau + \delta \ll M$$

How good is the first order model?

– Great yesterday, w/ 5min checkpoints and 25h MTTI

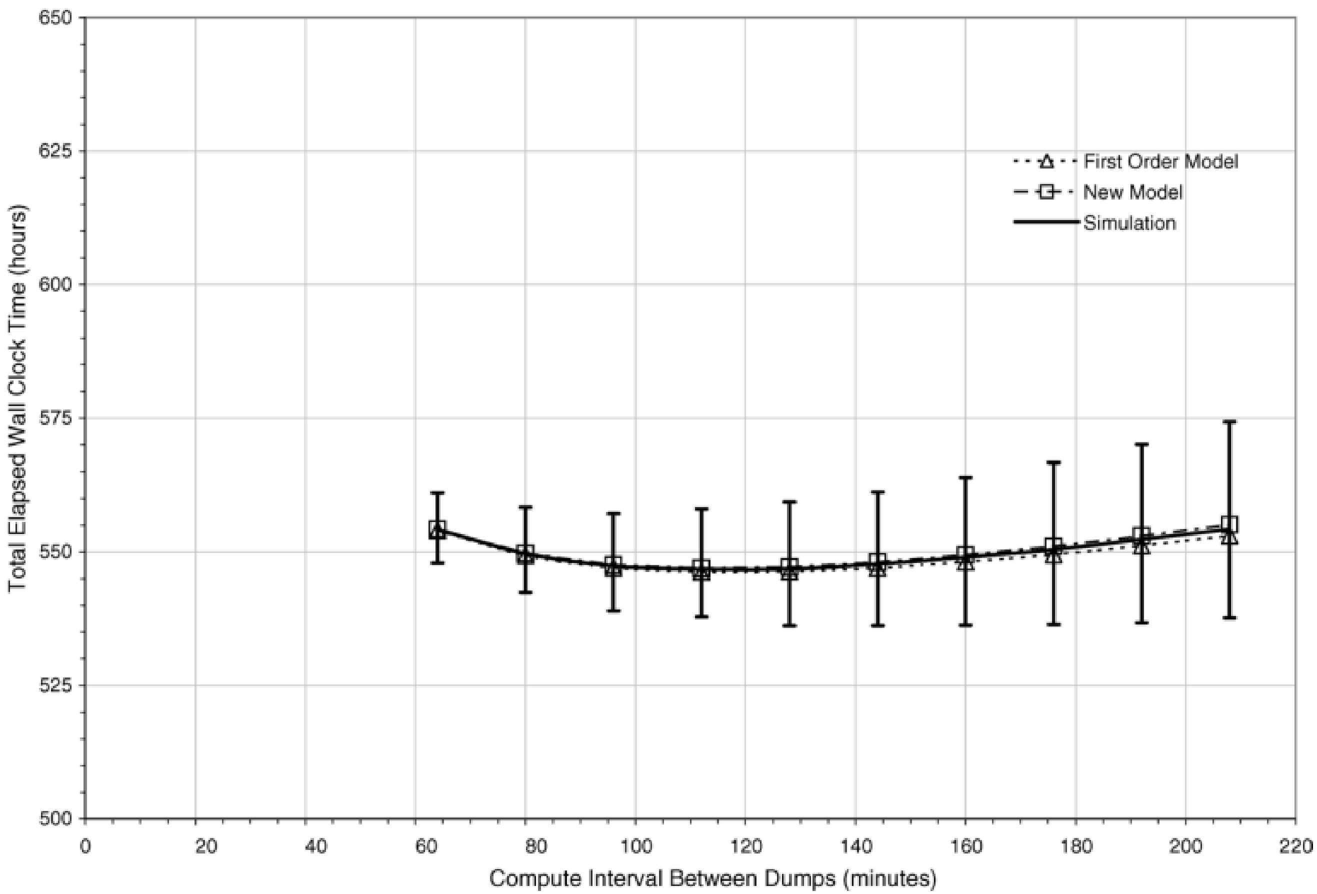


Fig. 3. Comparison of model and simulation results for $M = 24\,\mathrm{h}$, $T_s = 500\,\mathrm{h}$, $R = 10\,\mathrm{min}$, and $\delta = 5\,\mathrm{min}$. The new model predicts $\tau_{opt} = 117\,\mathrm{min}$.

How good is the first order model?

– OK today, w/ 5min checkpoints and 5h MTTI

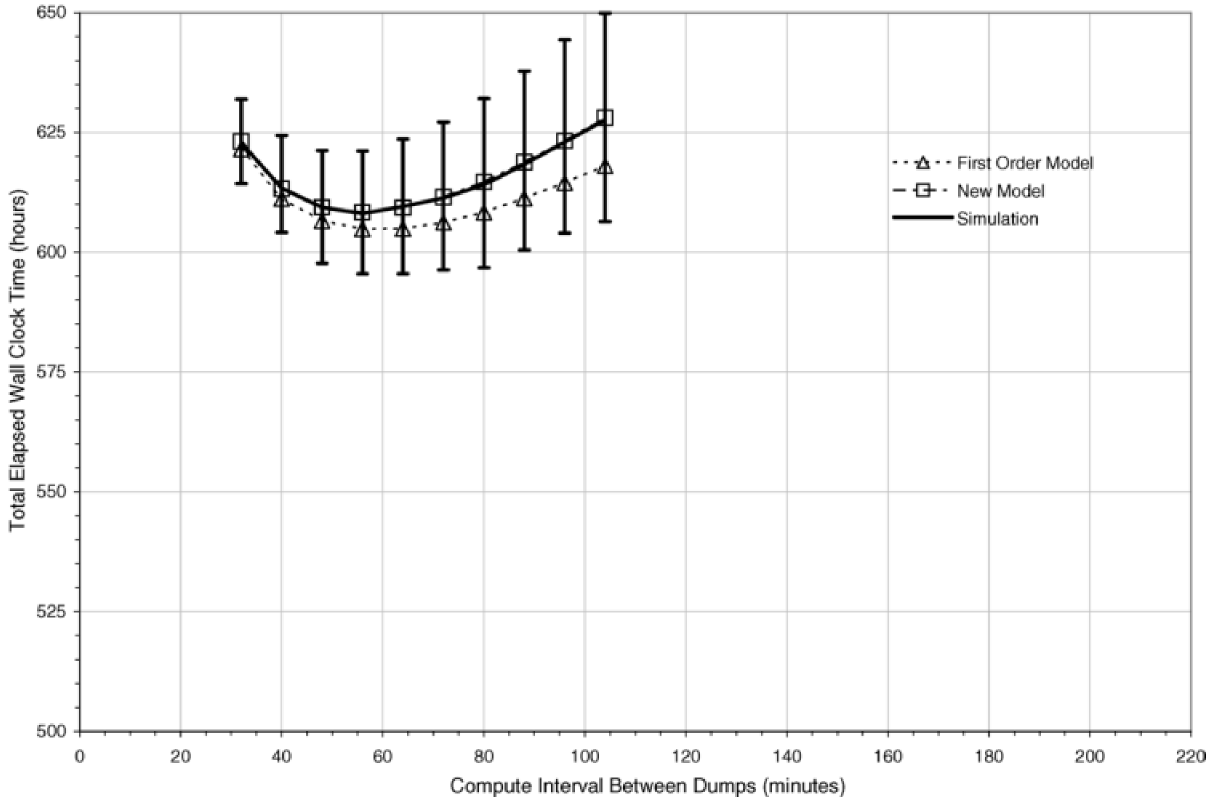
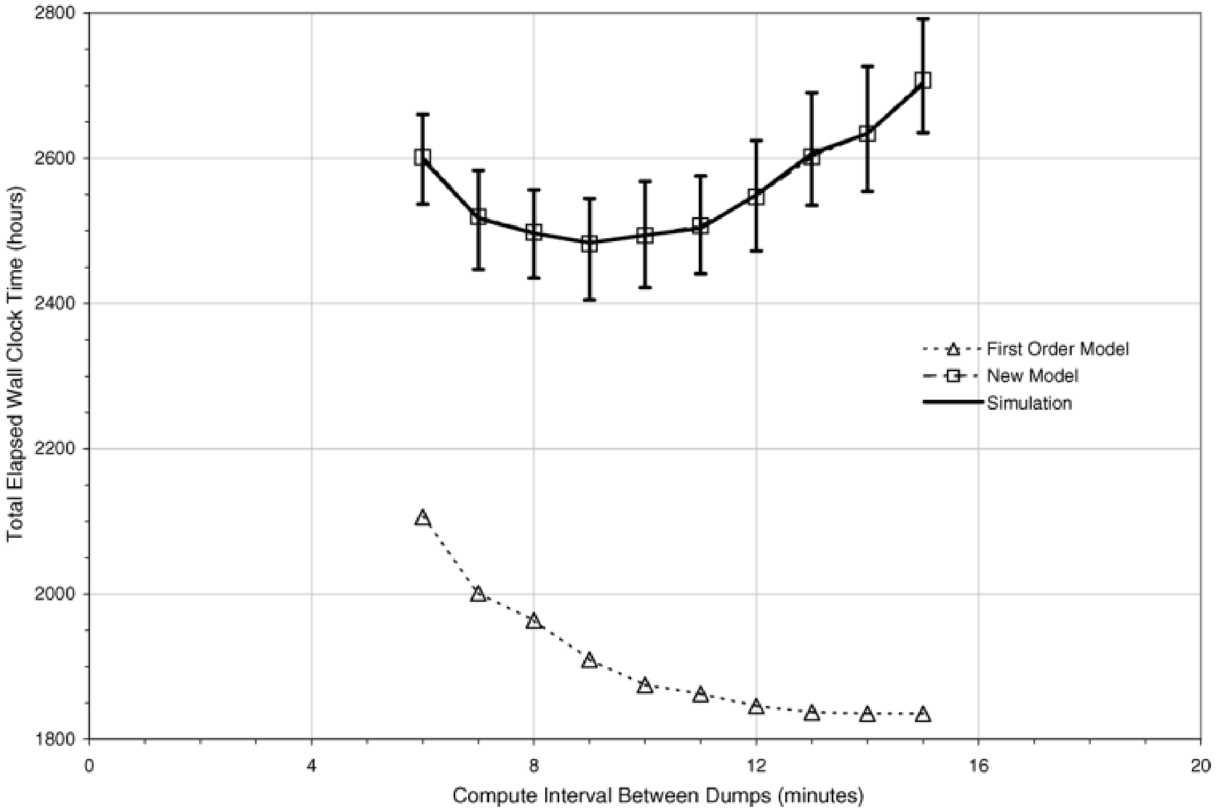


Fig. 4. Comparison of model and simulation results for $M = 6$ h, $T_s = 500$ h, $R = 10$ min, and $\delta = 5$ min. The new model predicts $\tau_{opt} = 57$ min.

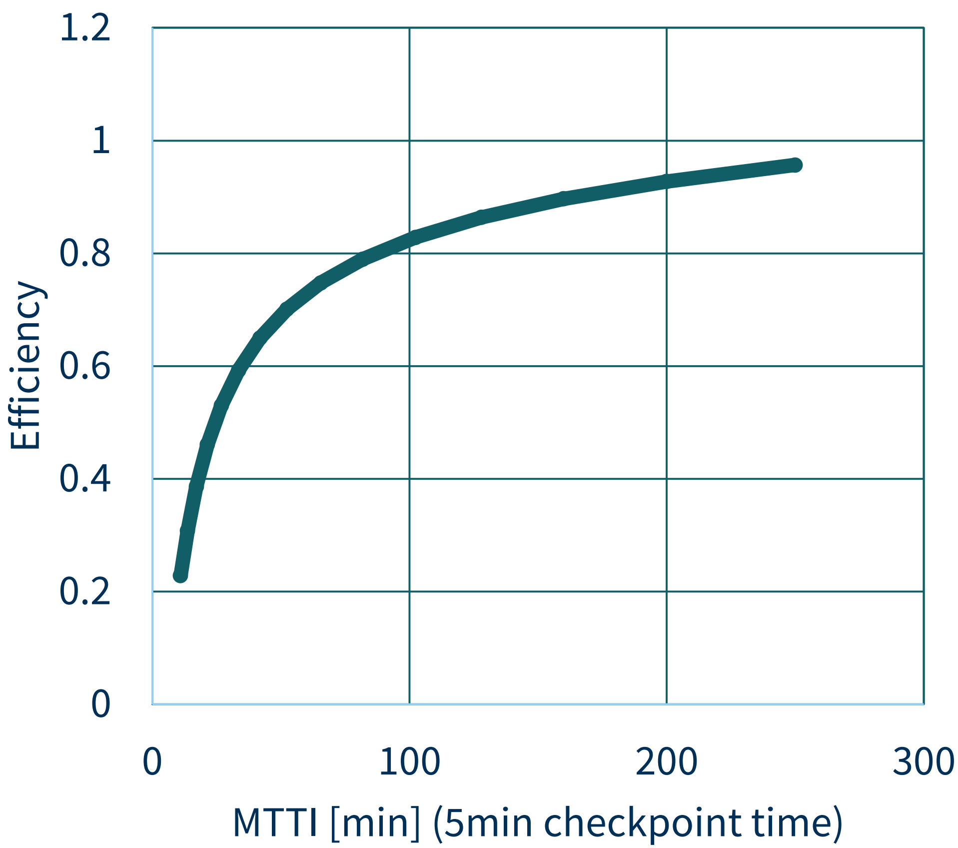
How good is the first order model?

- Bad in the future (MTTI = 15min, checkpoint = 5min)
  - Ignored interrupts during checkpoint and restart
  - Ignored greater likelihood of failing early in a period



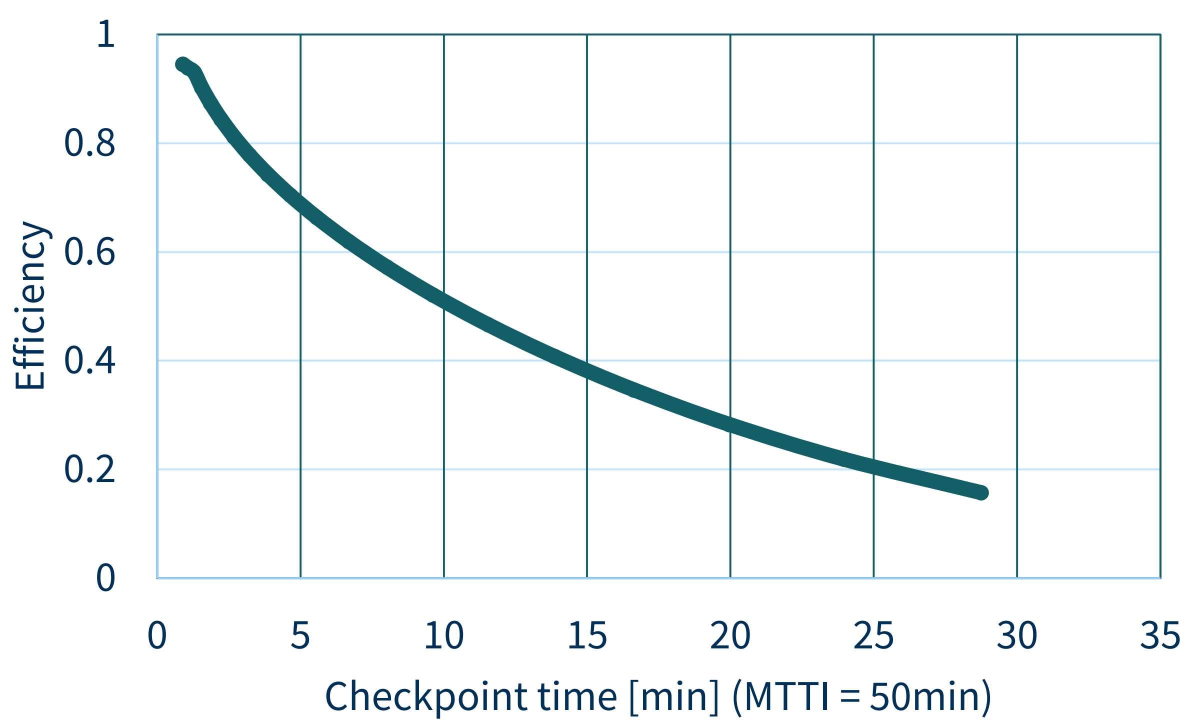
So what is the rough efficiency?

– Good today, but tomorrow



What can we do?

- Improve reliability
  - Expensive
- Reduce checkpoint overhead?



How can we reduce the checkpoint overhead?

- Copy less state
- Overlap copy and compute
- Copy faster

Copy less state?

- Only preserve critical live state
  - Ask programmer
  - Analyze
- Incremental checkpointing
  - Only save the delta from prior checkpoint
    - Need to track what changed (can use OS protection mechanisms)
    - Garbage collection a big issue – when do we no longer need old state?
    - Recovery much more complicated

Overlap copy and compute?

- Take quick checkpoint locally (e.g., to SSD)
  - BW can be 2GB/s per node – 100x improvement
- Slowly copy out to global file system during computation
  - What happens on interrupt?
    - May have to rollback to previous full checkpoint
- "Burst buffers"
- Can also use memprotect to copy the checkpoint in the background
  - Similar to VM live migration

Copy faster

- Build more BW to global file system
- Partition and replicate file system
- Use hierarchy?

"Scalable checkpoint restart"

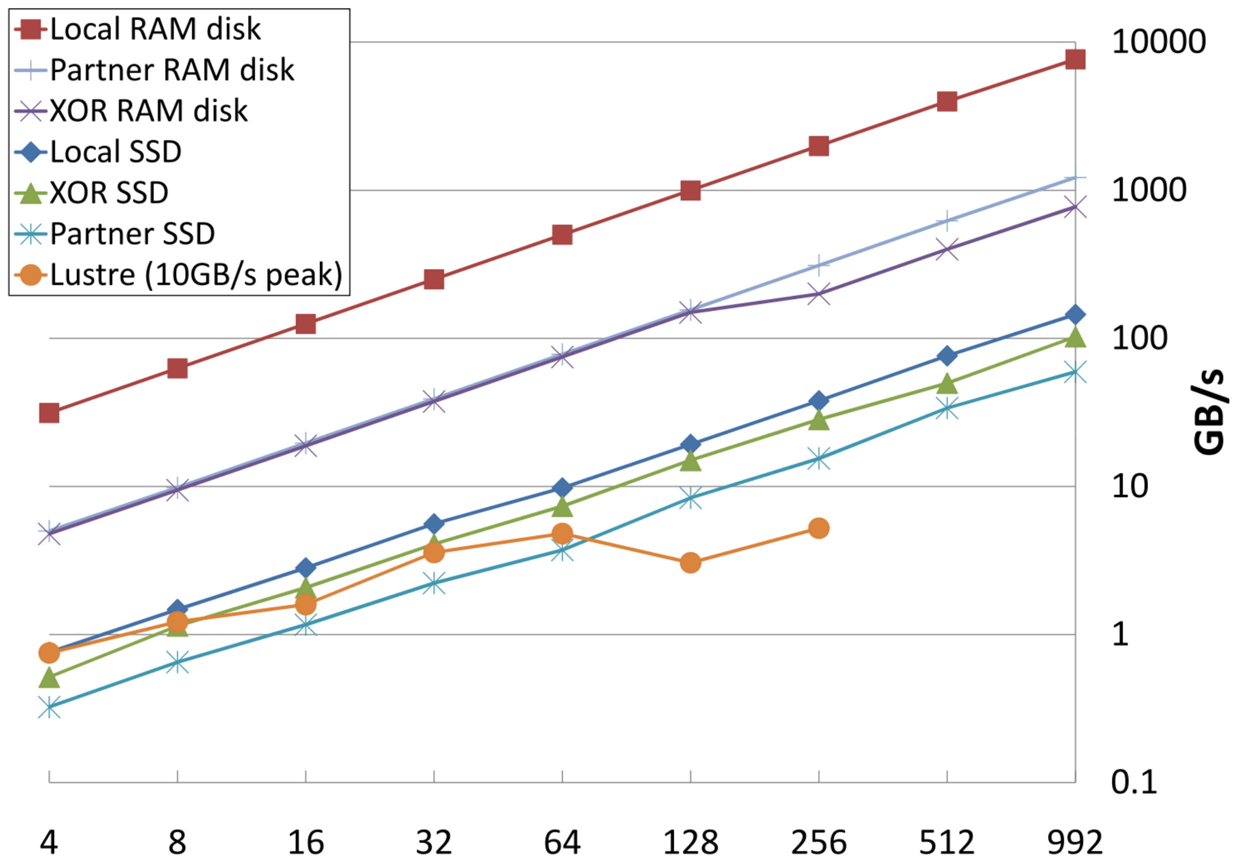
- Take frequent checkpoints to memory
- Less frequent to SSD
- Less frequent to global file system
- Can adjust number of levels

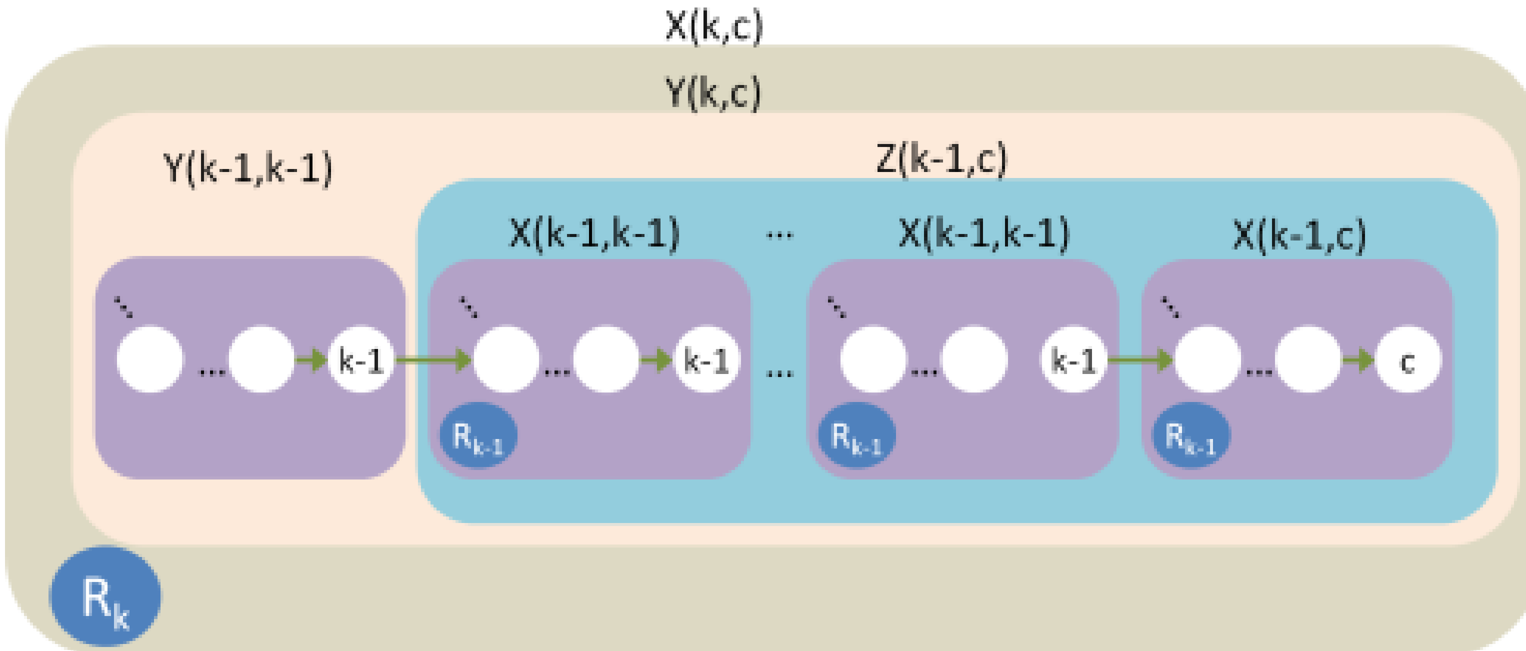
Moody et al., SC10

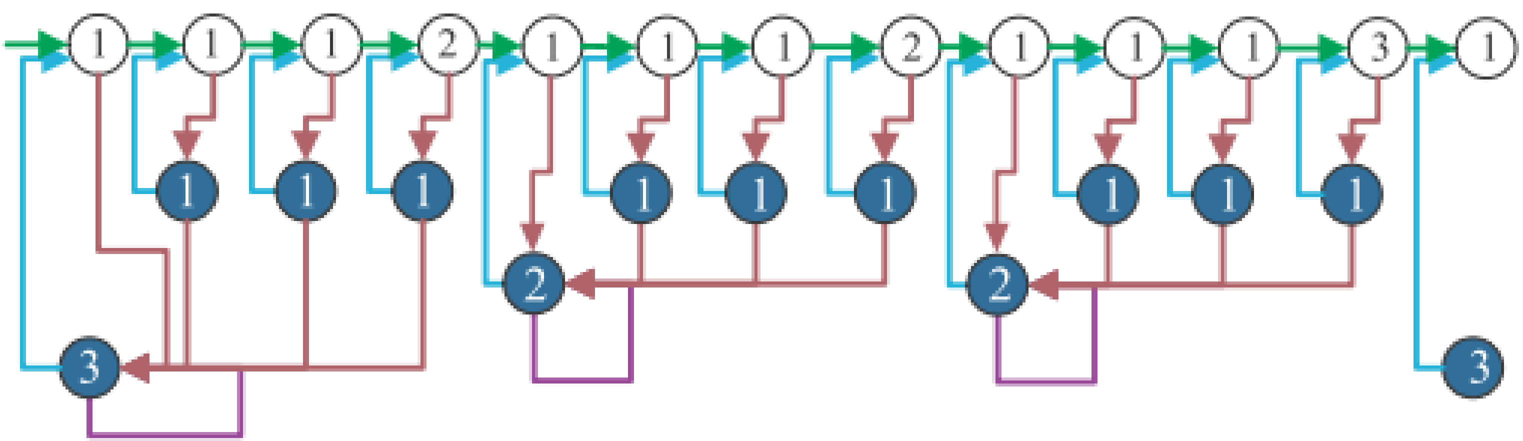
Saving locally not very effective for node failure

– Copy to a "buddy" instead

– Coding can reduce memory requirements



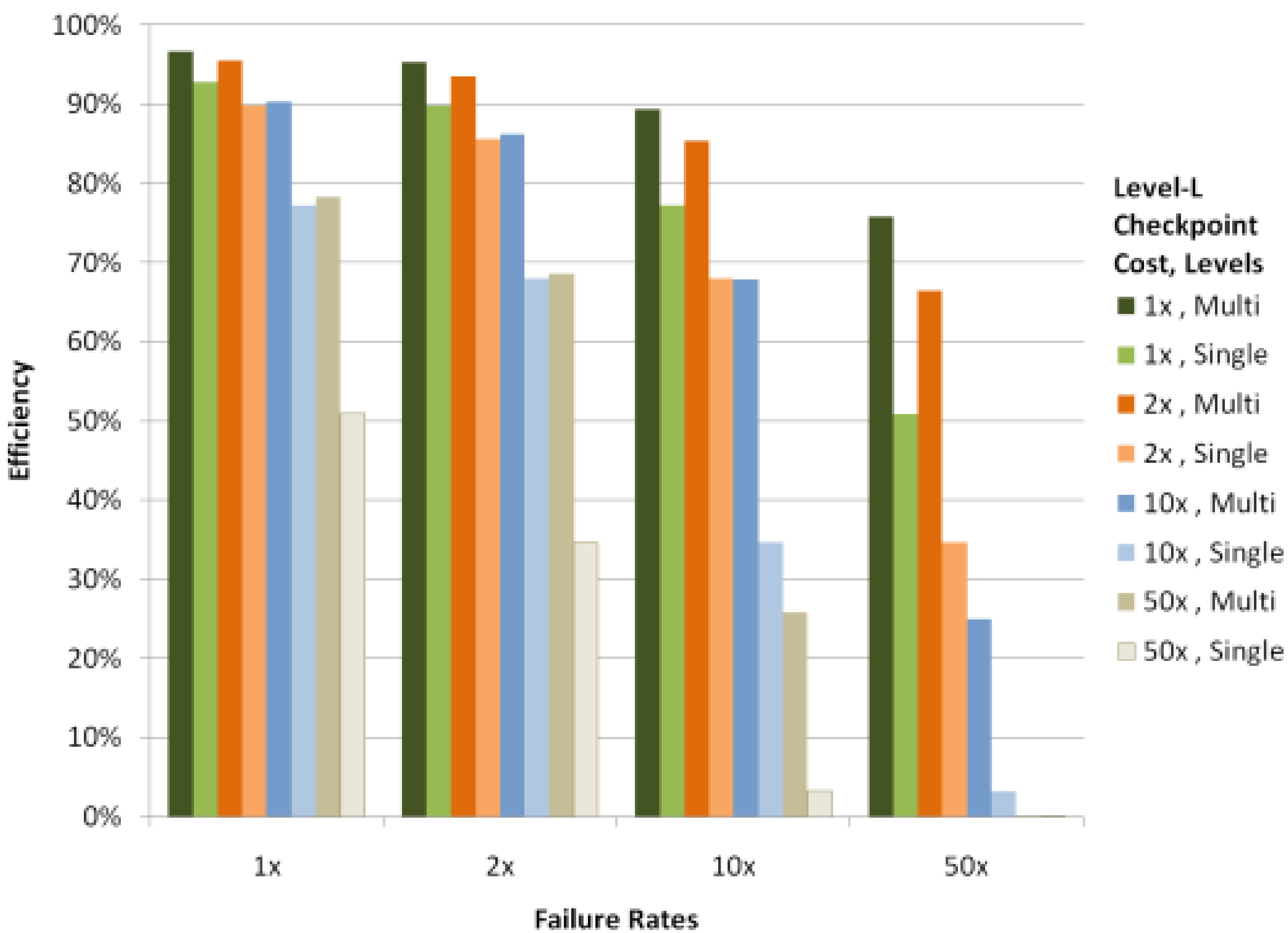

X(k,c)
Y(k,c)
Y(k-1,k-1)
Z(k-1,c)
X(k-1,k-1)
...
X(k-1,k-1)
X(k-1,c)
k-1
k-1
k-1
c
R_{k-1}
R_{k-1}
R_{k-1}
R_k



PFSD FAILURES ON THREE DIFFERENT CLUSTERS

Clusters	Coastal	Hera	Atlas	Total
Time span	Oct 09 - Mar 10	Nov 08 - Nov 09	May 08 - Oct 09	
Number of jobs	135	455	281	871
Node hours	2,830,803	1,428,547	1,370,583	5,629,933
Total failures	24	87	80	191
LOCAL required	2 (08%)	36 (41%)	21 (26%)	59 (31%)
PARTNER/XOR required	18 (75%)	32 (37%)	54 (68%)	104 (54%)
Lustre required	4 (17%)	19 (22%)	5 (06%)	28 (15%)

System	Expected Efficiency	Observed Efficiency	Duration of Observation
Coastal	95.2%	94.68%	716,613 node-hours
Atlas	96.7%	92.39%	553,829 node-hours


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Efficiency
1x
2x
10x
50x
Failure Rates
Level-L Checkpoint Cost, Levels
1x , Multi
1x , Single
2x , Multi
2x , Single
10x , Multi
10x , Single
50x , Multi
50x , Single

Things to think about (outline)

- How frequently to checkpoint?
- How important is checkpoint time?
  - And what can we do to improve it
- **Who decides to take a checkpoint?**
  - **System or user**
- Do we really have to stop everyone to take a checkpoint?
  - Coordinated vs. uncoordinated checkpointing
  - Always (for now) coordinated restart

Great survey paper:

ElNozahy et al., “A Survey of Rollback-Recovery Protocols in Message-Passing Systems” (http://www.cs.utexas.edu/~lorenzo/papers/SurveyFinal.pdf)

Who decides when to checkpoint

User or system?

- Requirement: consistent checkpoint
- Implication: no inflight messages
  - (for coordinated)
- What gets checkpointed?

Identifying consistent points

- User knows
  - Annotate
- Runtime may know
  - Certain runtime calls imply consistency
  - MPI collectives and barriers
- System doesn’t
  - Must quiesce network to take checkpoint
  - Often impractical, but SDN might help?

What data should be checkpointed?

- Programmer can identify minimal set
  - But what if programmer missed something?
- Compiler analysis may help
- System can use OS page table and protection mechanisms to checkpoint

Both are in use today

- Application-level libraries (more common)
  - User can ask if checkpointed needed and then decide to call checkpoint routine
  - User can ask for checkpoint periodically and library can skip
- System-level
  - Integrated with kernel
  - Dumps entire application state (or incremental one)

Things to think about (outline)

- How frequently to checkpoint?
- How important is checkpoint time?
  - And what can we do to improve it
- Who decides to take a checkpoint?
  - System or user
- **Do we really have to stop everyone to take a checkpoint?**
  - **Coordinated vs. uncoordinated checkpointing**
  - Always (for now) coordinated restart

Great survey paper:

ElNozahy et al., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems" (http://www.cs.utexas.edu/~lorenzo/papers/SurveyFinal.pdf)

Coordinated checkpointing is straightforward

- Find consistent time
- Checkpoint

Uncoordinated is not

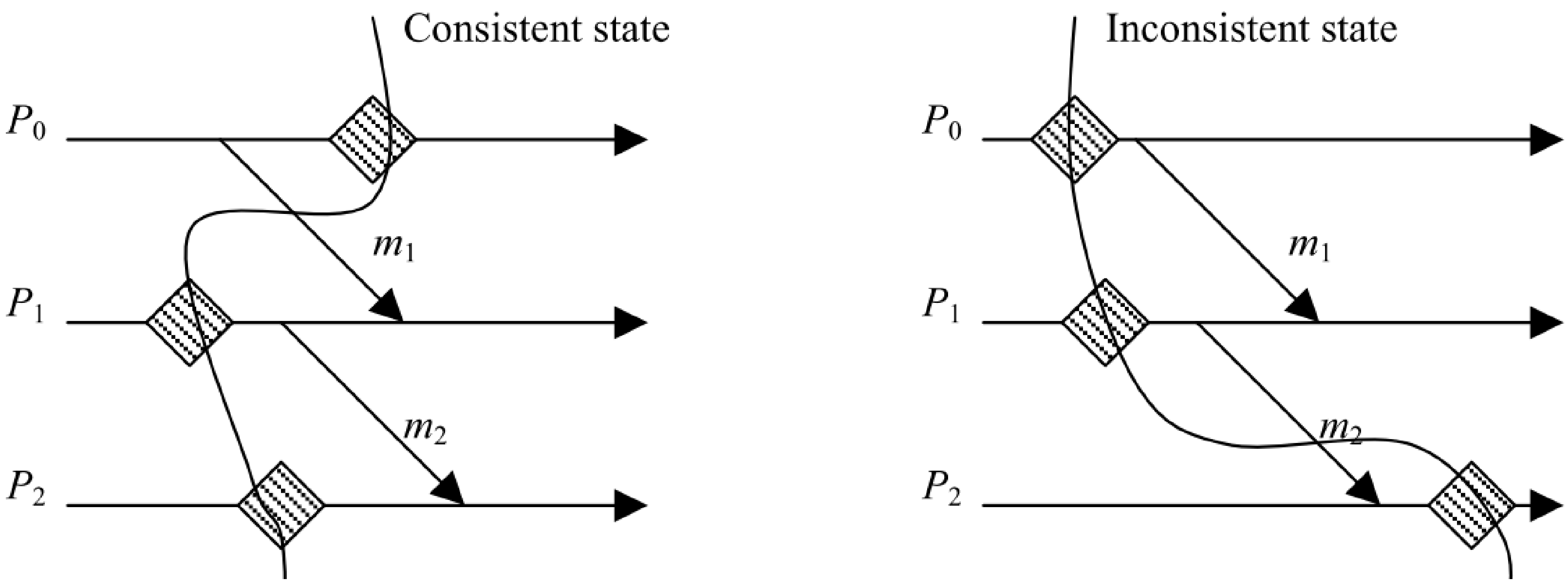
- How to deal with in-flight messages?
- How to deal with in-flight remote loads/stores?

Side note: message passing vs. global address space

- Two sided or one sided communication
- Send/recv or put/get

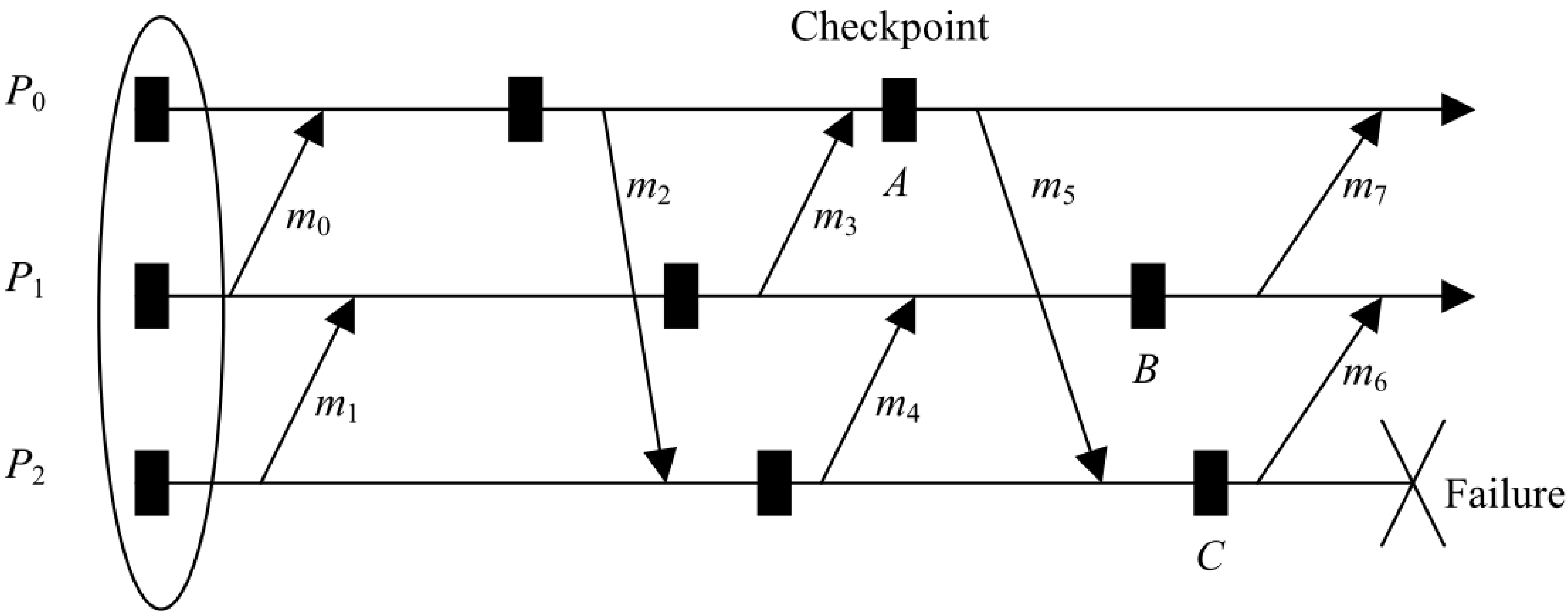
Uncoordianted checkpointing w/ messages

- Distributed systems theory (and practice)
- Checkpoint each process mostly independently
- Goal is to enable rollback to a consistent state
  - Everyone re-executes and agrees on all results



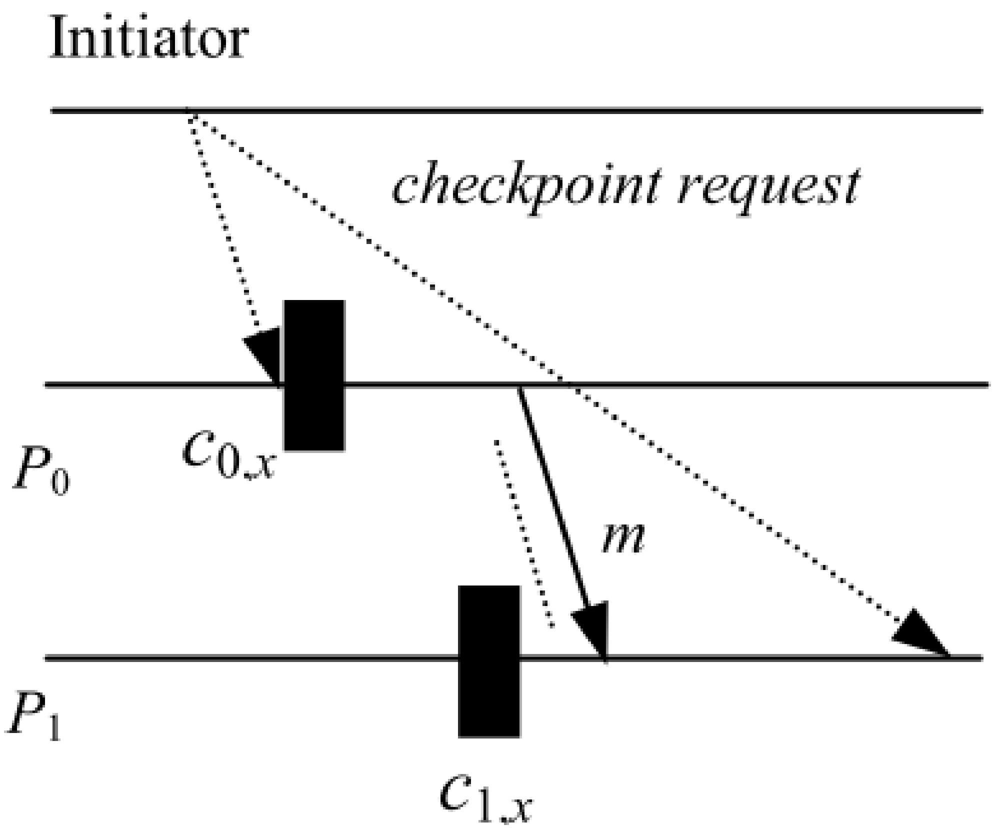
From ElNozahy et al., ACM Comp Surveys, 06/2002

The Domino Effect (Randell, 1975)



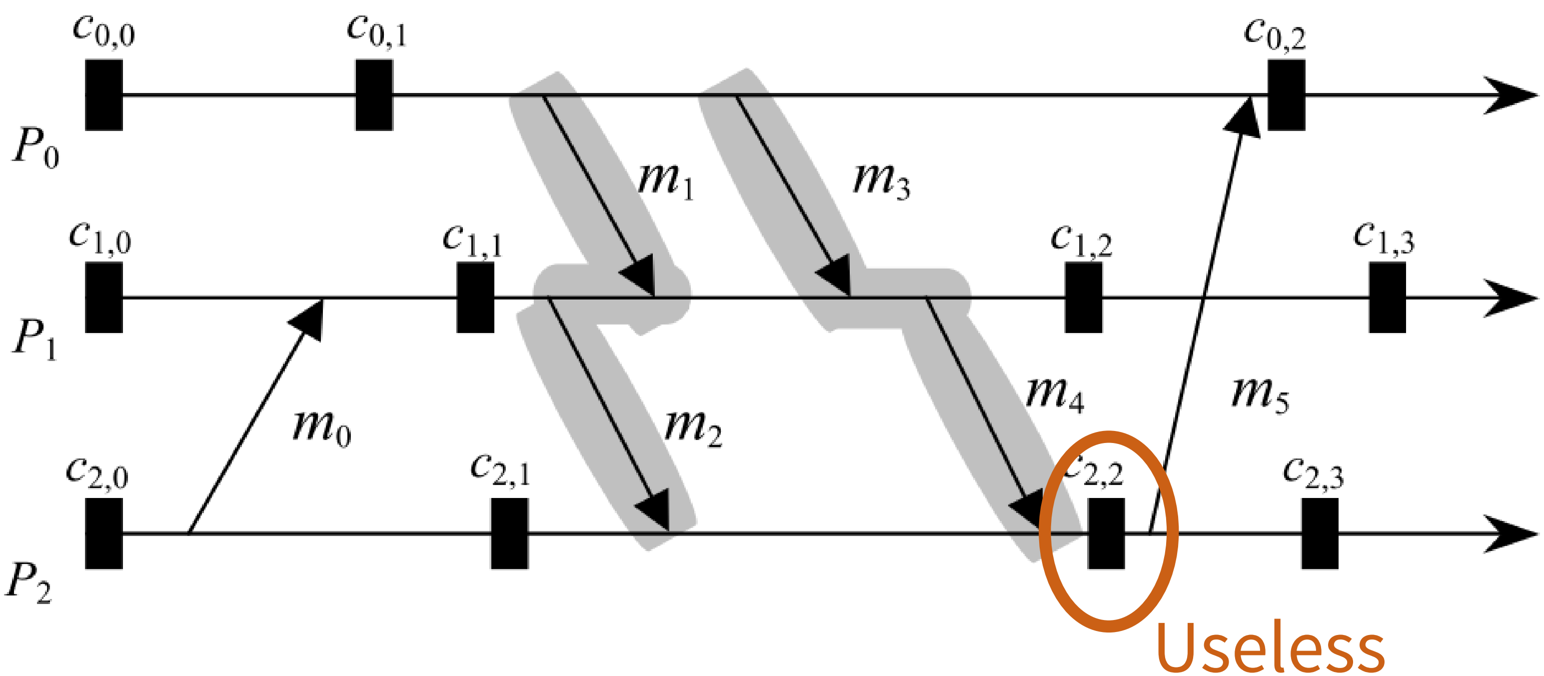
From ElNozahy et al., ACM Comp Surveys, 06/2002

Coordinated non-blocking



From ElNozahy et al., ACM Comp Surveys, 06/2002

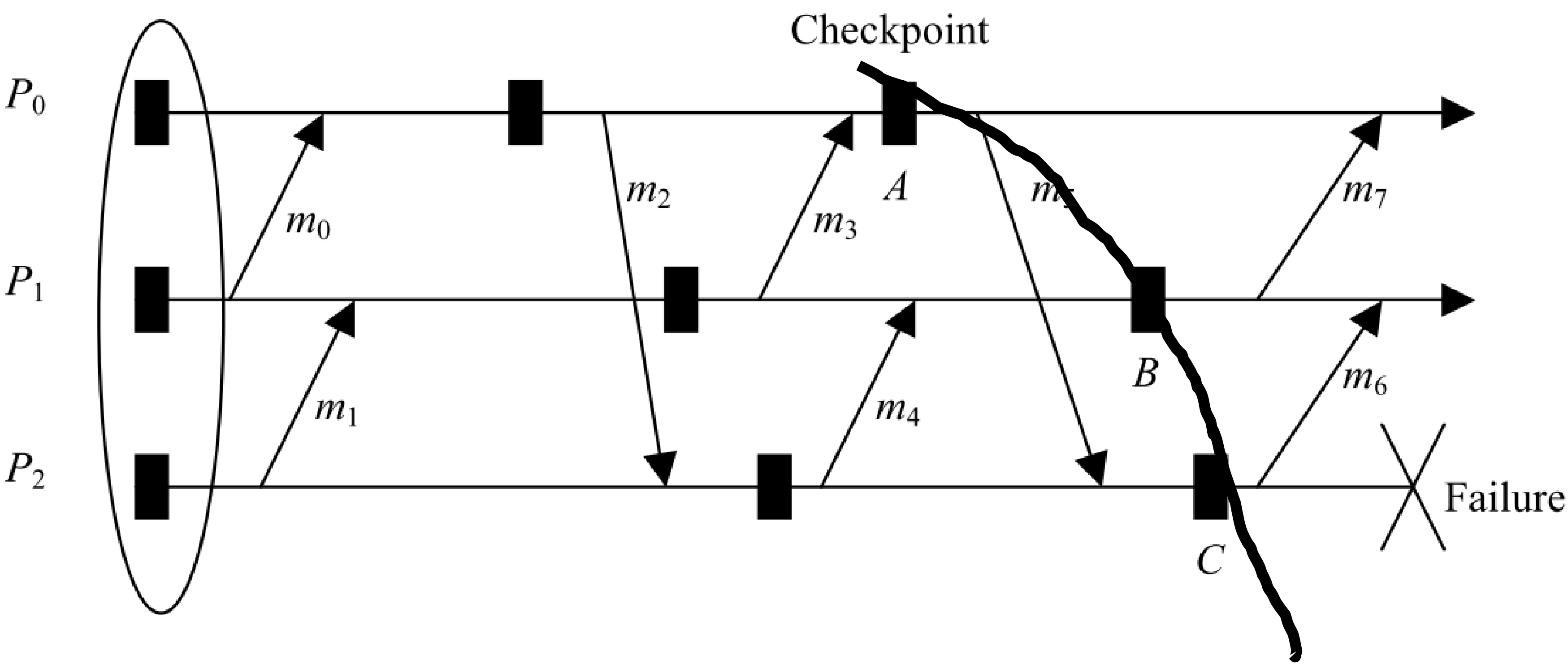
Communication-induced



From ElNozahy et al., ACM Comp Surveys, 06/2002

Uncoordinated

- Message logging
- Pessimistic vs. optimistic

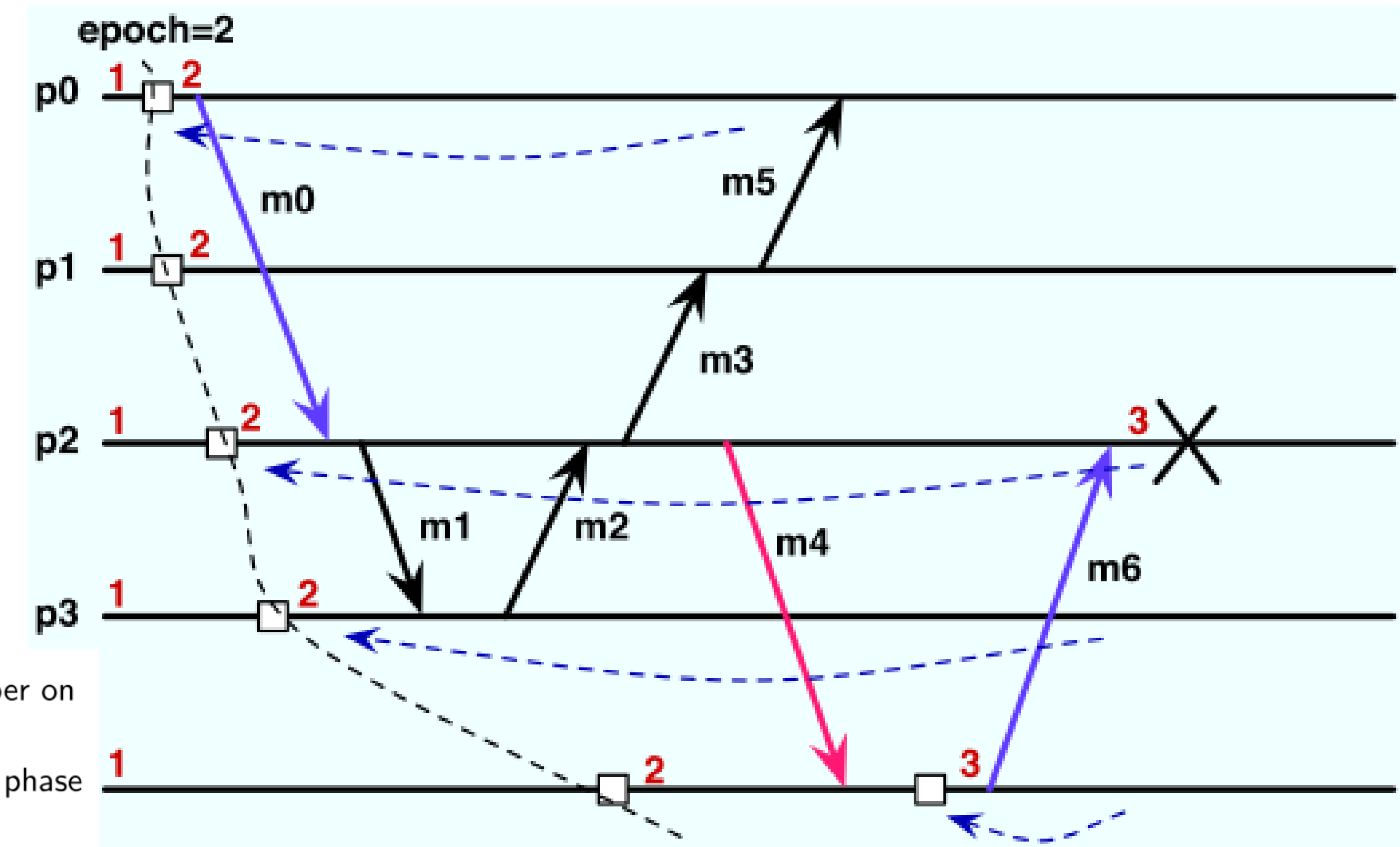


From ElNozahy et al., ACM Comp Surveys, 06/2002

Example protocol

An Uncoordinated Checkpointing Protocol for Send-deterministic HPC Application

Amina Guermouche[1,2], Thomas Ropars[1], Elisabeth Brunet[1], Marc Snir[3], Franck Cappello[1,3]



- Piggyback phase number on each message
- checkpoint: increment phase number
- logged messages: update and increment receiver's phase number if smaller
- non-logged message: update receiver's phase number

Upon Recovery: Messages sent according to phase numbers

From A. Guermouche et al., IPDPS 2012

Sender or receiver logs?

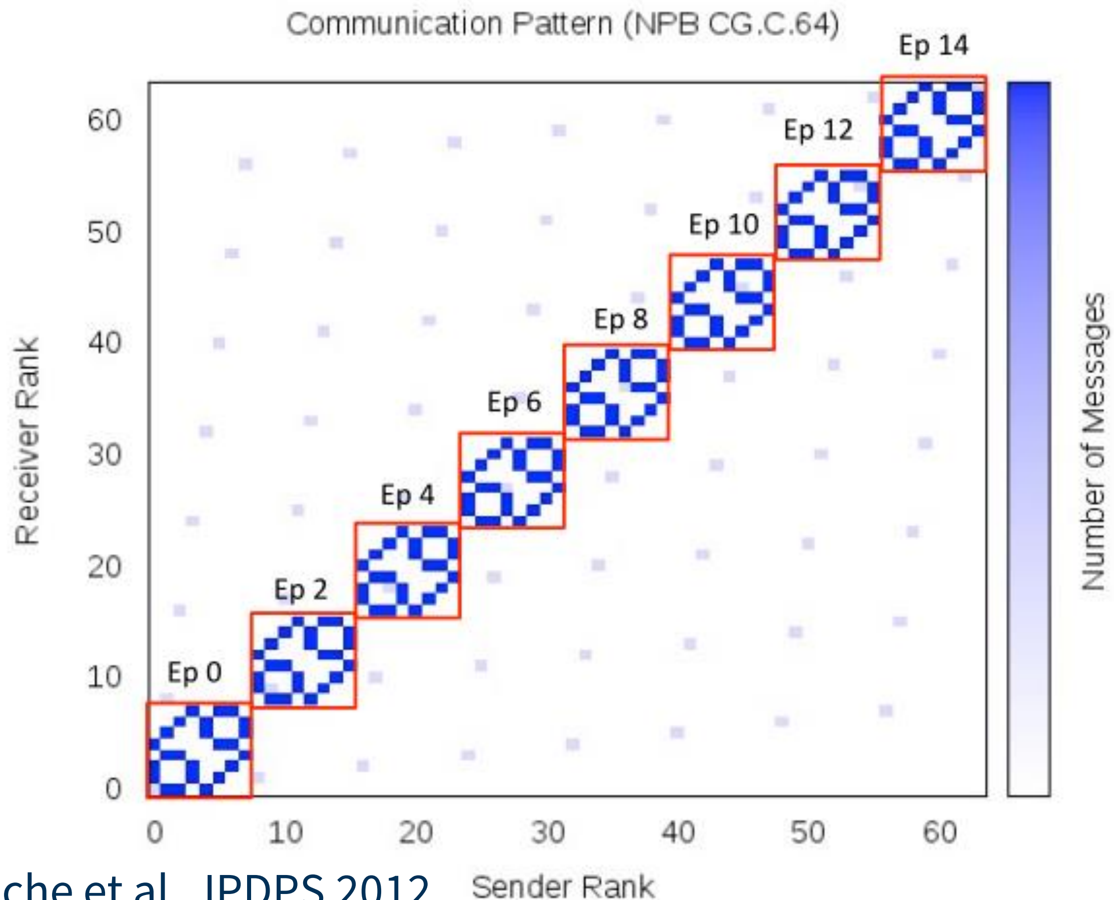
- Receiver is easier, but complicated if optimistic and interrupted
- Sender is easier when logging, but recovery can be complicated
  - As is possibly garbage collection

Non-blocking with message logging

- Coordinate a consistent point to checkpoint, but don’t block
- Start logging messages to eliminate domino effect
- Stop logging when checkpoint consistent and done

Hierarchical coordinated/uncoordinated groups

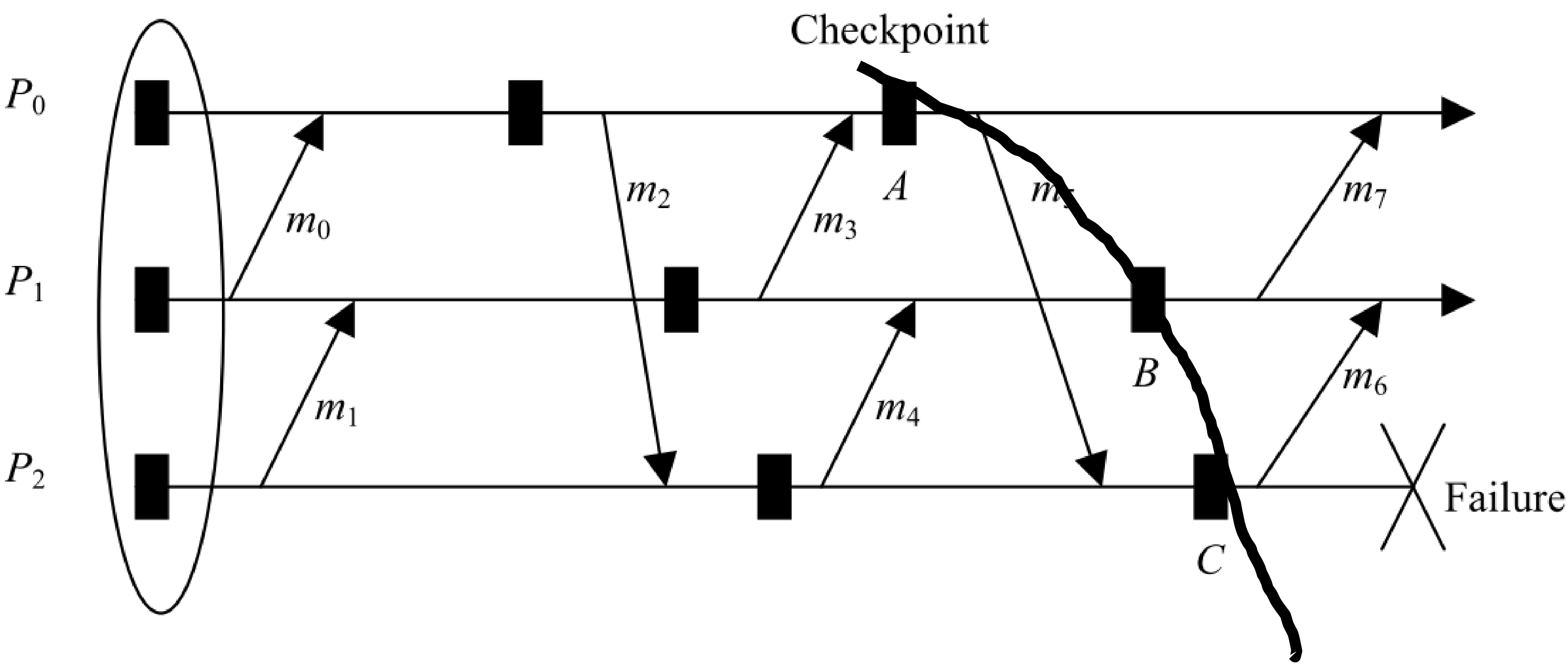
- Log messages hierarchically between groups of grouped processes
- More on this when discussing containment domains



From A. Guermouche et al., IPDPS 2012

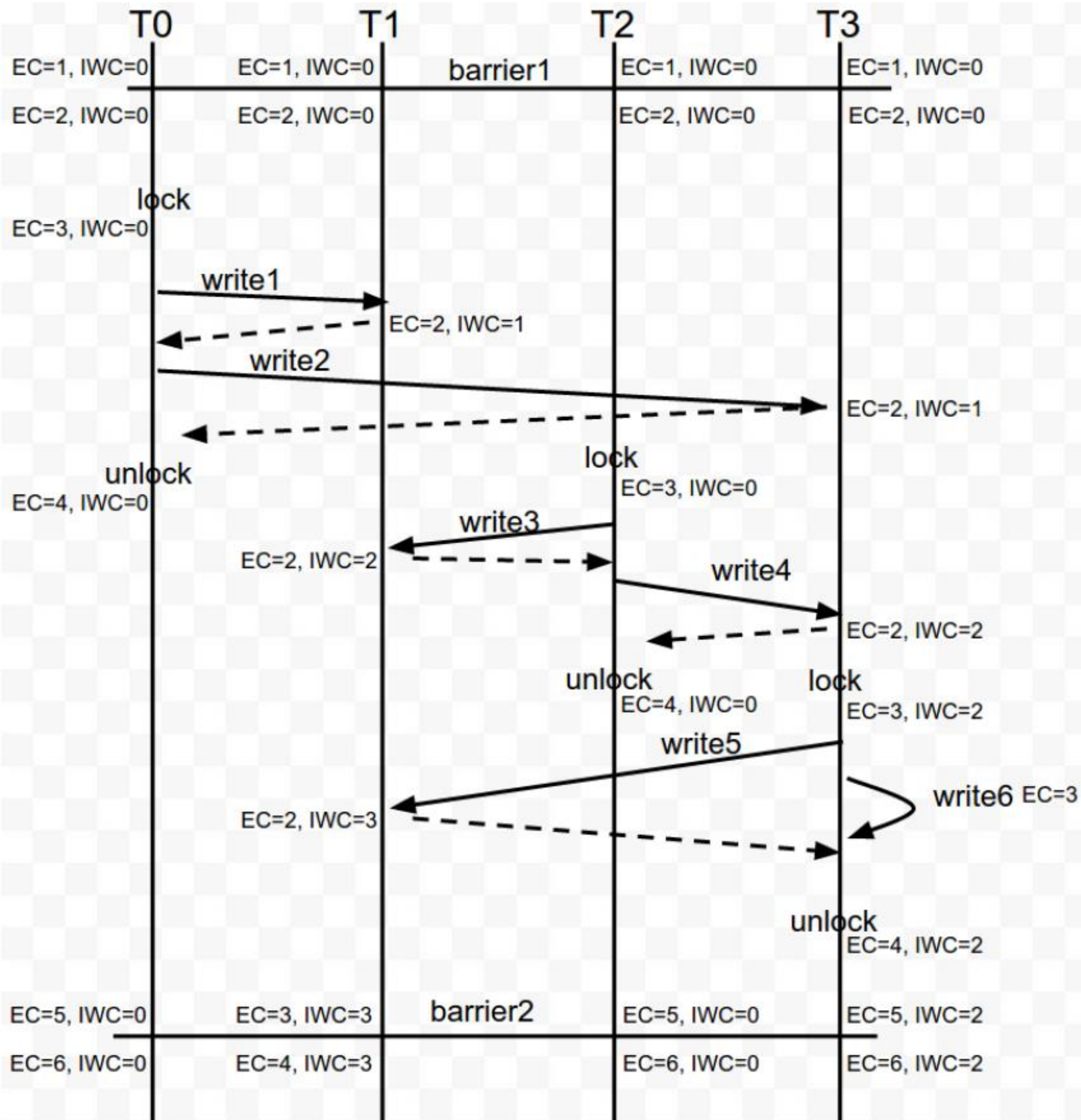
Uncoordinated recovery?

– Possible, but challenging



From ElNozahy et al., ACM Comp Surveys, 06/2002

- Global address space way more complicated
  - One sided communication happens without remote aware of communication
  - Fine-grained sync and comm
  - Very relaxed memory consistency models
  - Consistent points hard to find
  - Quiescing the network too expensive
- Active area of research (with few/no proofs yet)


T0
T1
T2
T3
EC=1, IWC=0
EC=1, IWC=0
barrier1
EC=1, IWC=0
EC=1, IWC=0
EC=2, IWC=0
EC=2, IWC=0
EC=2, IWC=0
EC=2, IWC=0
lock
EC=3, IWC=0
write1
EC=2, IWC=1
write2
EC=2, IWC=1
lock
unlock
EC=3, IWC=0
EC=4, IWC=0
write3
EC=2, IWC=2
write4
EC=2, IWC=2
unlock
lock
EC=4, IWC=0
EC=3, IWC=2
write5
write6 EC=3
EC=2, IWC=3
unlock
EC=4, IWC=2
EC=5, IWC=0
EC=3, IWC=3
barrier2
EC=5, IWC=0
EC=5, IWC=2
EC=6, IWC=0
EC=4, IWC=3
EC=6, IWC=0
EC=6, IWC=2

Detect→contain→repair→recover

- Periodically, take checkpoint
  - Stop the system
  - Copy *all* state somewhere sage
  - Keep going
- Rollback
  - Recover saved state
- Restart
  - Recompute and keep going

Things to think about (outline)

- How frequently to checkpoint?
- How important is checkpoint time?
  - And what can we do to improve it
- Who decides to take a checkpoint?
  - System or user
- Do we really have to stop everyone to take a checkpoint?
  - Coordinated vs. uncoordinated checkpointing
  - Always (for now) coordinated restart

Great survey paper:

ElNozahy et al., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems" (http://www.cs.utexas.edu/~lorenzo/papers/SurveyFinal.pdf)

Redundant MPI

– An alternative to checkpoint/restart