What Does Scalable Resilience Look Like

Laxmikant (Sanjay) Kale
http://charm.cs.illinois.edu
Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana Champaign

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

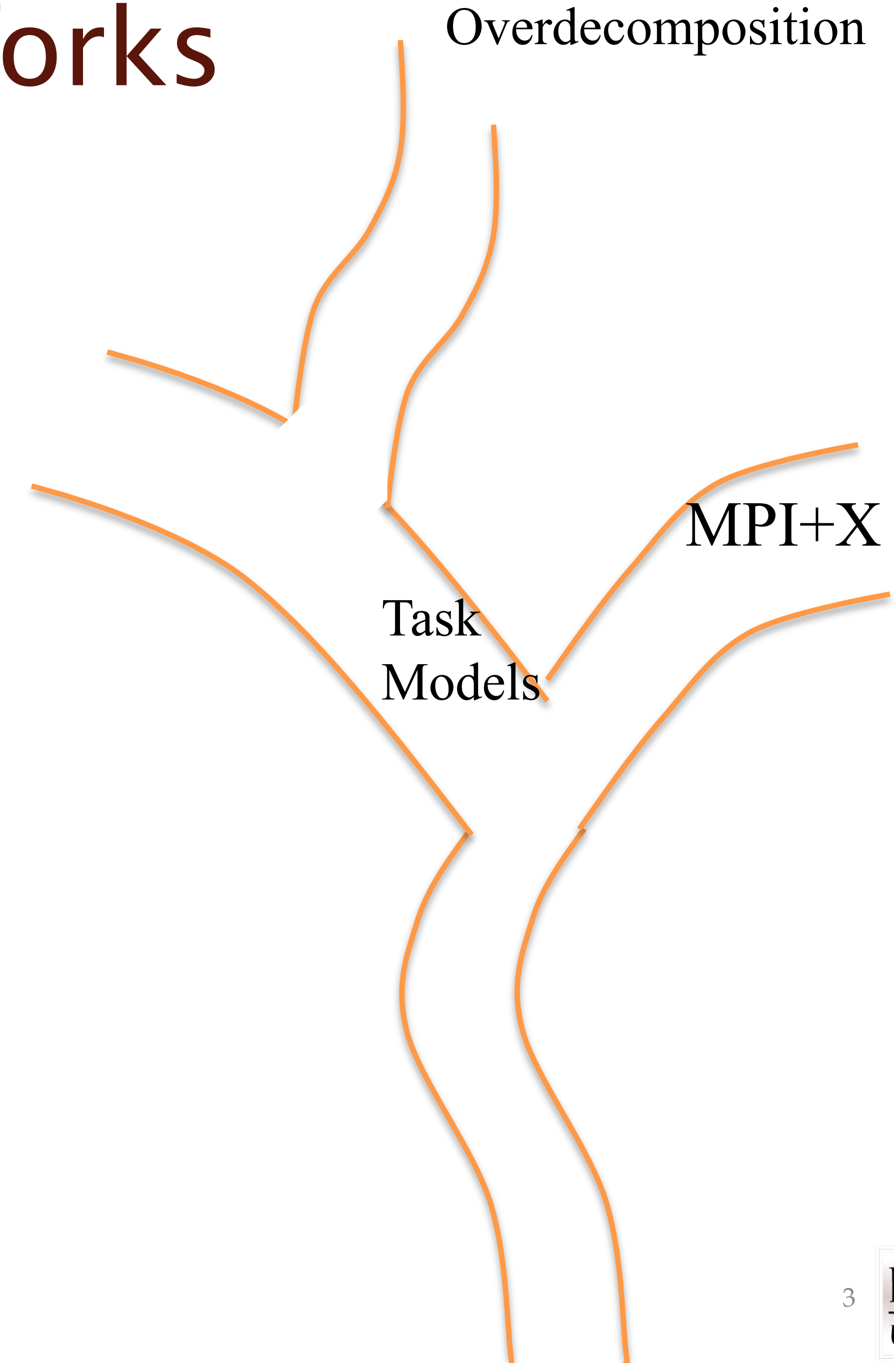
PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

Outline

- SDCs are all the rage, but:
- Failstop failures are not going away
- We need schemes to handle them both
  - "all bets are off" – nathan (not true: we can combine both)
- RTS based solutions can isolate applications from having to deal with failures
  - Somehow, no talk covered this
- Overdecomposition based solutions contribute some unique solutions/enhancements
- Checkpoint/restart (charm++ has been supporting fault detection and automatic restart for 5+ years now)
  - Optimized by non-blocking protocols
  - Burst buffers?? We can do without for many apps.. Reuse the memory (make it multi-purpose)
  - Many apps have relatively small mem footprint at checkpoint
  - These can be combined with SDC detection schemes
- But the real fun is message-logging schemes

A couple of forks

- MPI + x
- “Task Models”
  - Asynchrony
- Overdecomposition:
  - Most adaptivity



A Runtime System based on
Over-decomposition and
Migratability
can support resilience effetively

Runtime Systems can play a role

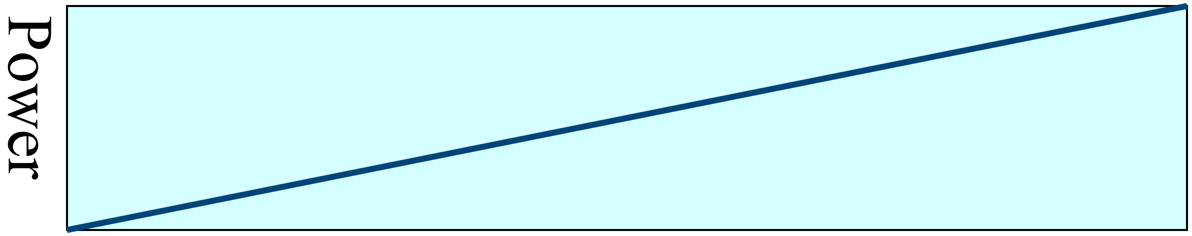
- RTS based solutions to resilience are desirable
  - They insulate the application from failures
  - RTS has information about the the machine status and application status
  - Applications can provide information to the RTS

Failstop Faults

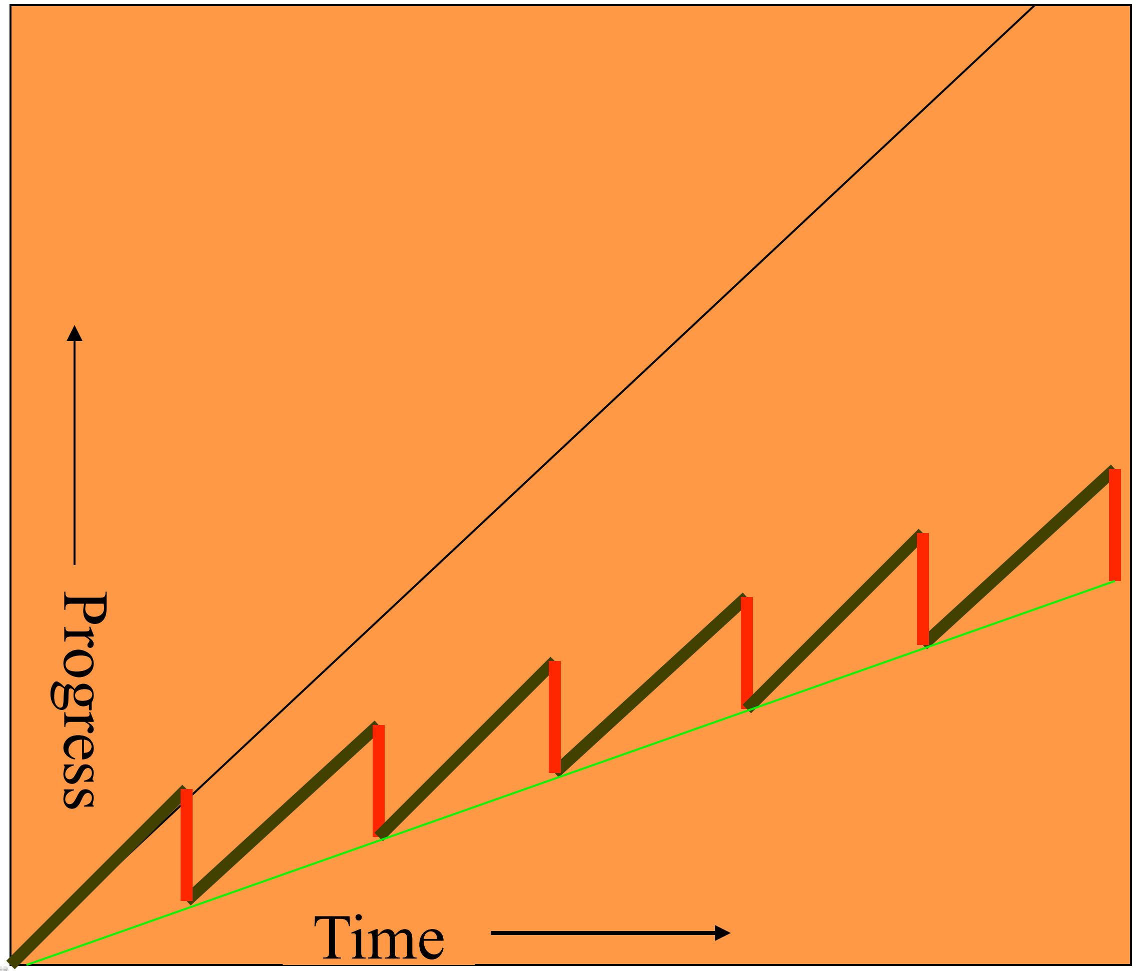
- Silent Data Corruption is what everyone is talking about
  - It is important
  - But failstop faults are not going away
  - We need to handle them both

Progress Rate is the right metric

- Compared with
  - distributed systems theory, or
  - a mission to mars, or
  - real–time systems
- HPC needs are different
  - We will accept a small probability of failure
    - In that case, we will redo the simulation
  - But we care about application making progressin presence of faults



Power consumption is continuous



Normal Checkpoint-Resart method

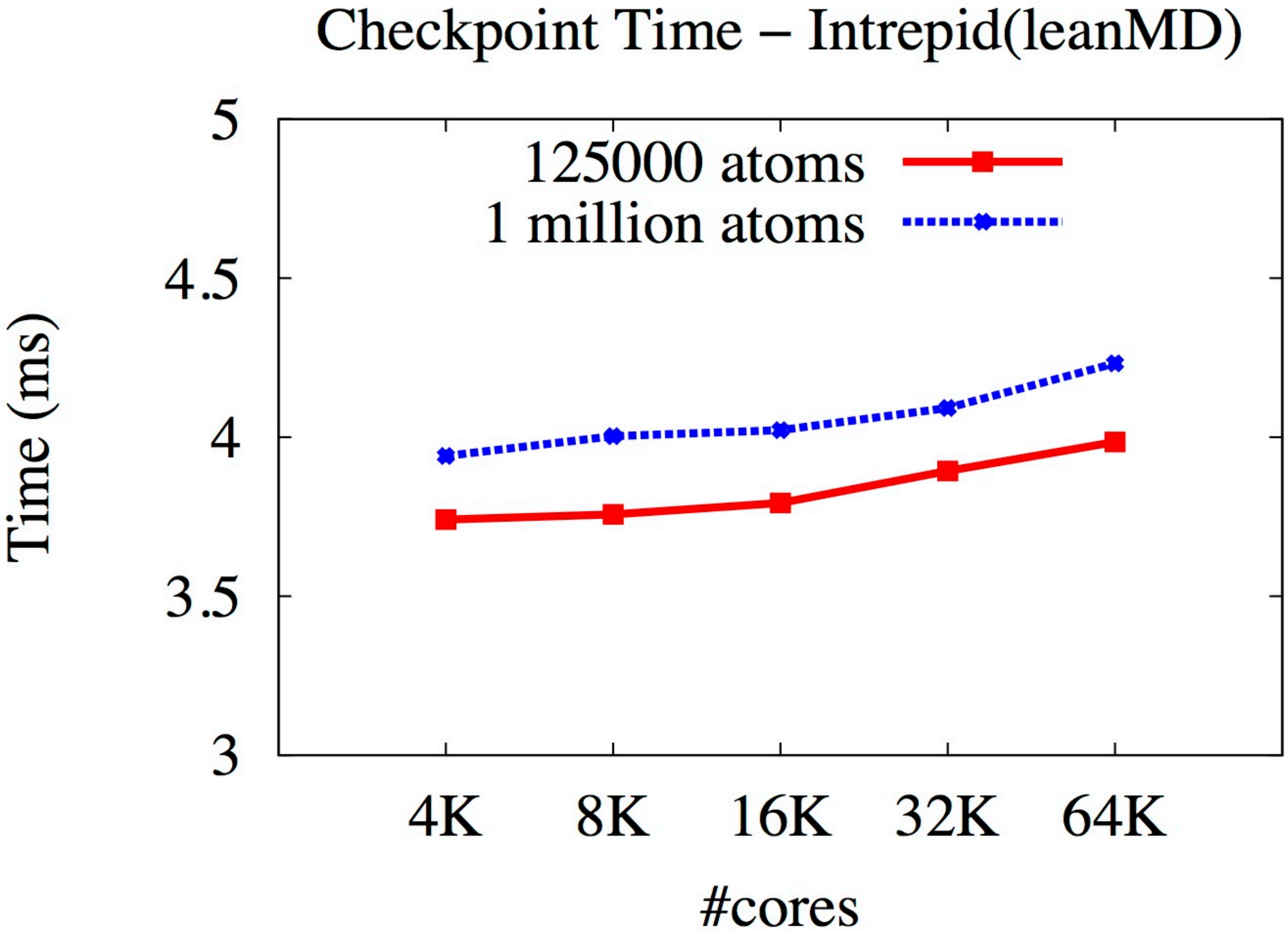
Progress is slowed down with failures

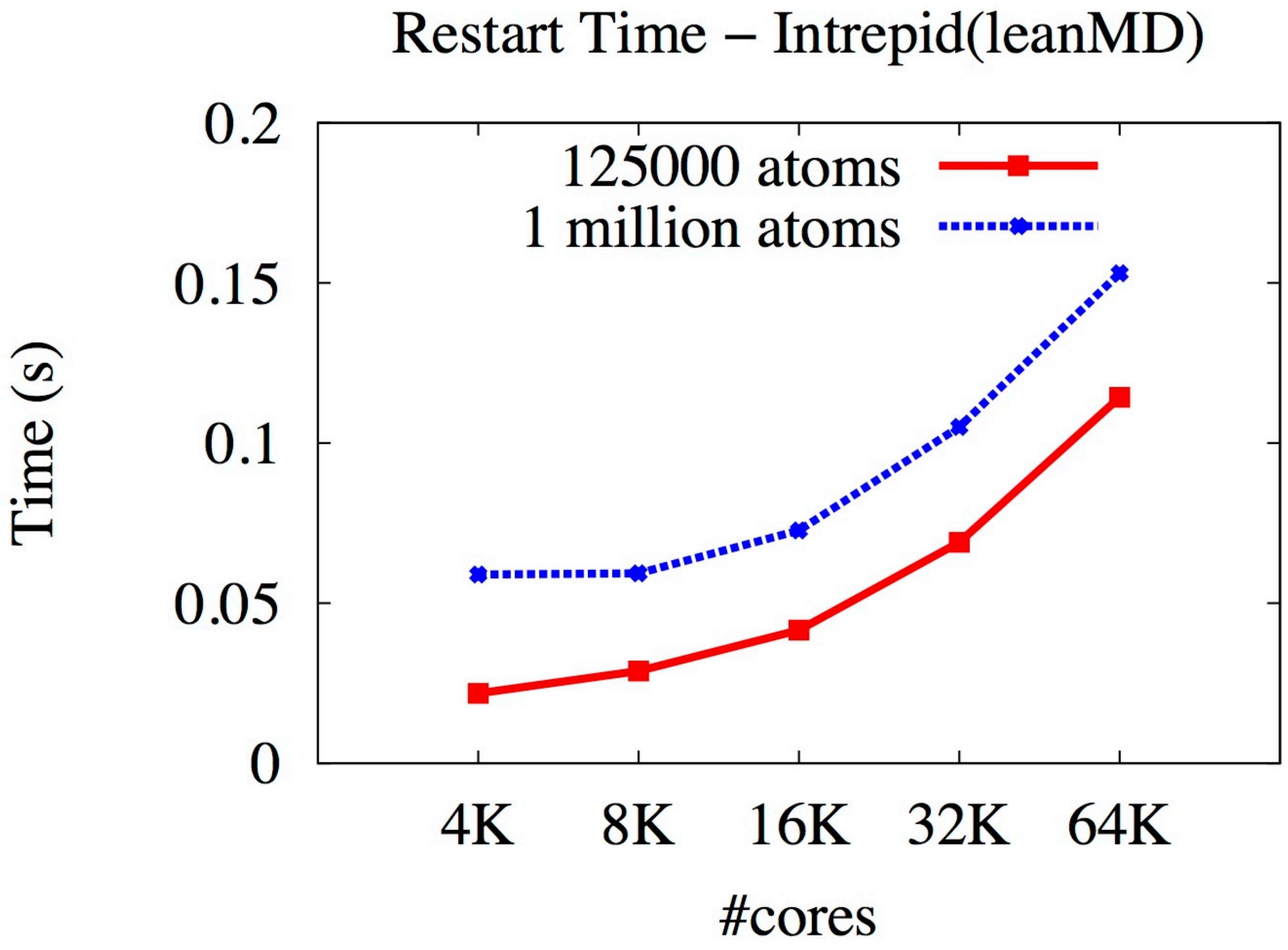
Fault Tolerance in Charm++/AMPI

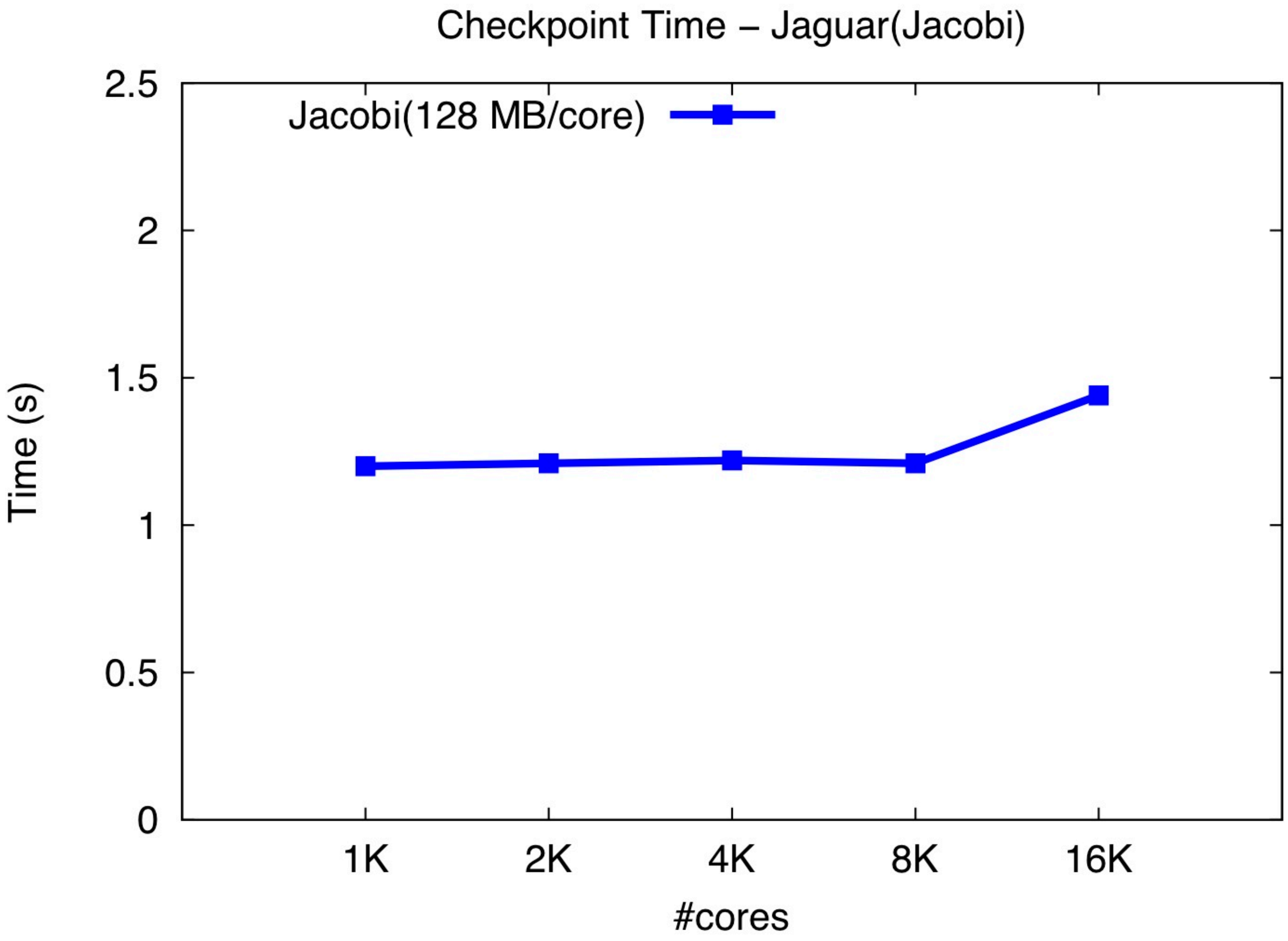
- Four approaches available:
  - Disk-based checkpoint/restart
  - In-local-storage double checkpoint w auto restart
    - Demonstrated on 64k cores
  - Proactive object migration
  - Message-logging: scalable fault tolerance
    - Can tolerate frequent faults
    - Parallel restart and potential for handling faults during recovery

In-memory checkpointing

- Actually: In local-storage double checkpoint, with automatic failure detection and restart
- Is practical for many apps
  - Relatively small footprint at checkpoint time
- Very fast times…
- Demonstration challenge:
  - Works fine for clusters
  - For MPI-based implementations running at centers:
    - Scheduler does not allow job to continue on failure
    - Communication layers not fault tolerant
  - Fault injection: dieNow(),
  - Spare processors


Checkpoint Time – Intrepid(leanMD)
125000 atoms
1 million atoms
Time (ms)
5
4.5
4
3.5
3
4K
8K
16K
32K
64K
#cores


Restart Time – Intrepid(leanMD)
125000 atoms
1 million atoms
Time (s)
0.2
0.15
0.1
0.05
0
4K
8K
16K
32K
64K
#cores


Checkpoint Time – Jaguar(Jacobi)
Jacobi(128 MB/core)
Time (s)
2.5
2
1.5
1
0.5
0
1K
2K
4K
8K
16K
#cores

Extensions to fault recovery

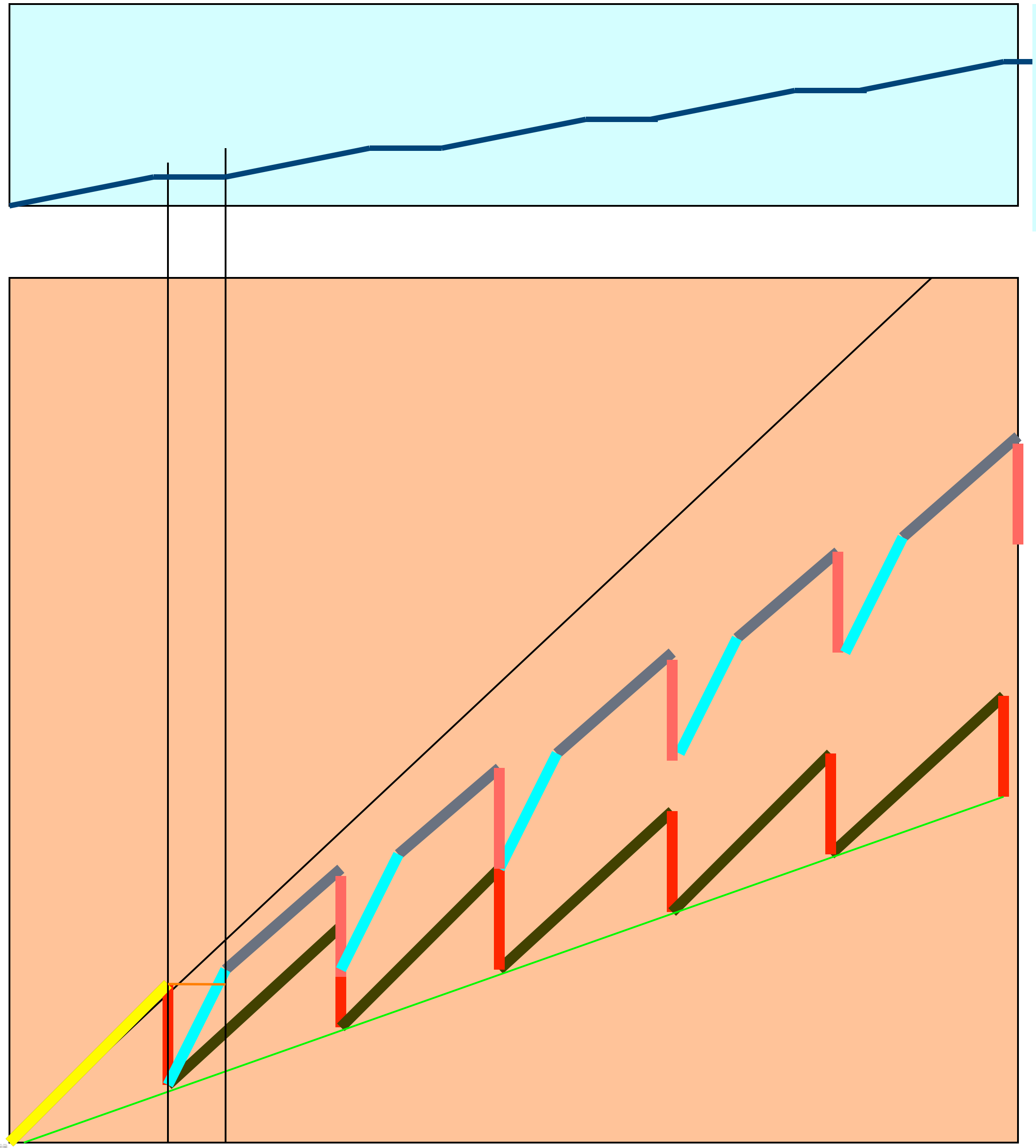
- Based on the same over-decomposition ideas
  - A surprisingly large number of applications have low memory footprint at checkpoint
  - But, if not:
  - Use NVRAM instead of DRAM for checkpoints
    - Non-blocking variants
    - [Cluster 2012] Xiang Ni et al.
  - Replica-based soft-and-hard-error handling
    - As a “gold-standard” to optimize against
    - [SC 13] Xiang Ni, E. Meneses, N. Jain, et al.

Scalable Fault tolerance

- Faults will be frequent at exascale (true??)
  - Failstop, and soft failures are both important
- Checkpoint-restart *may* not scale
  - Or will it?
  - Requires all nodes to roll back even when just one fails
    - Inefficient: computation and power
  - As MTBF goes lower, it becomes infeasible

Message-Logging

- Basic Idea:
  - Only the processes/objects on the failed node go back to the checkpoint!
  - Messages are stored by senders during execution
  - Periodic checkpoints still maintained
  - After a crash, reprocess “resent” messages to regain state
- Does it help at exascale?
  - Not really, or only a bit: Same time for recovery!
- But with over-decomposition,
  - work in one processor is divided across multiple virtual processors; thus, *restart can be parallelized*
  - Virtualization helps fault-free case as well

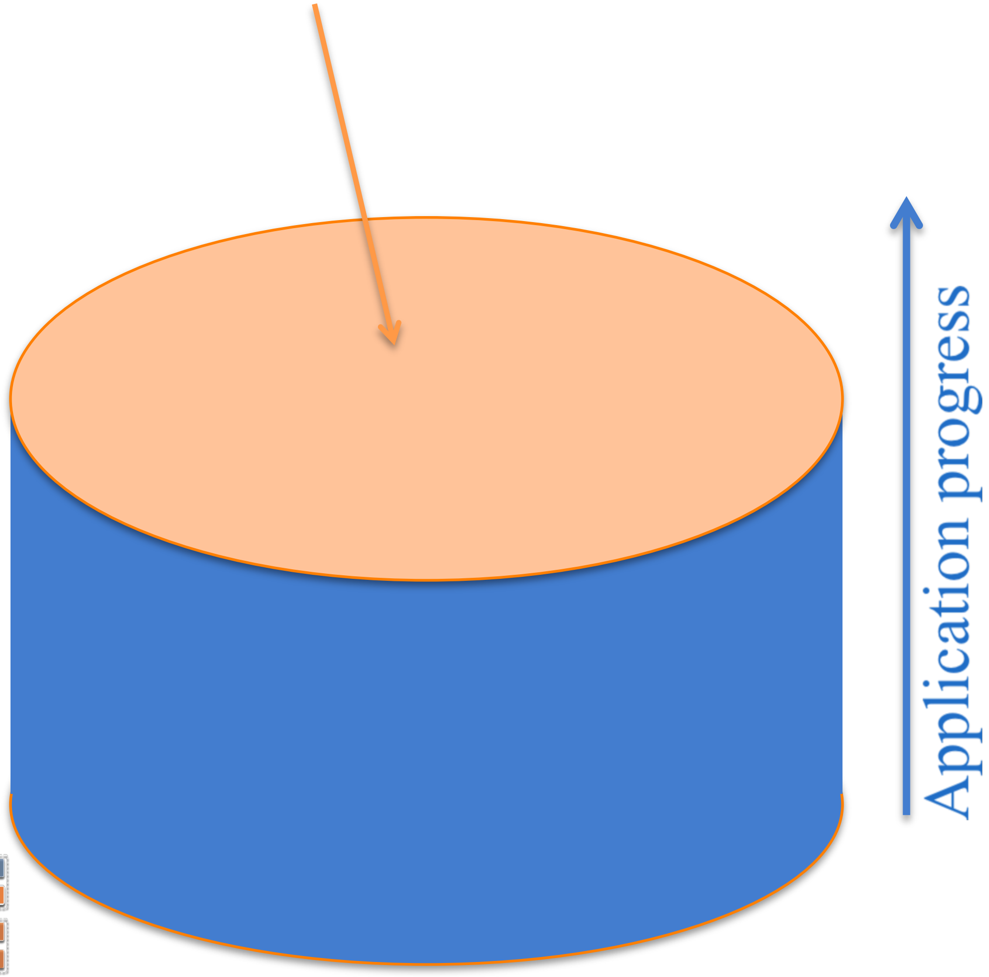


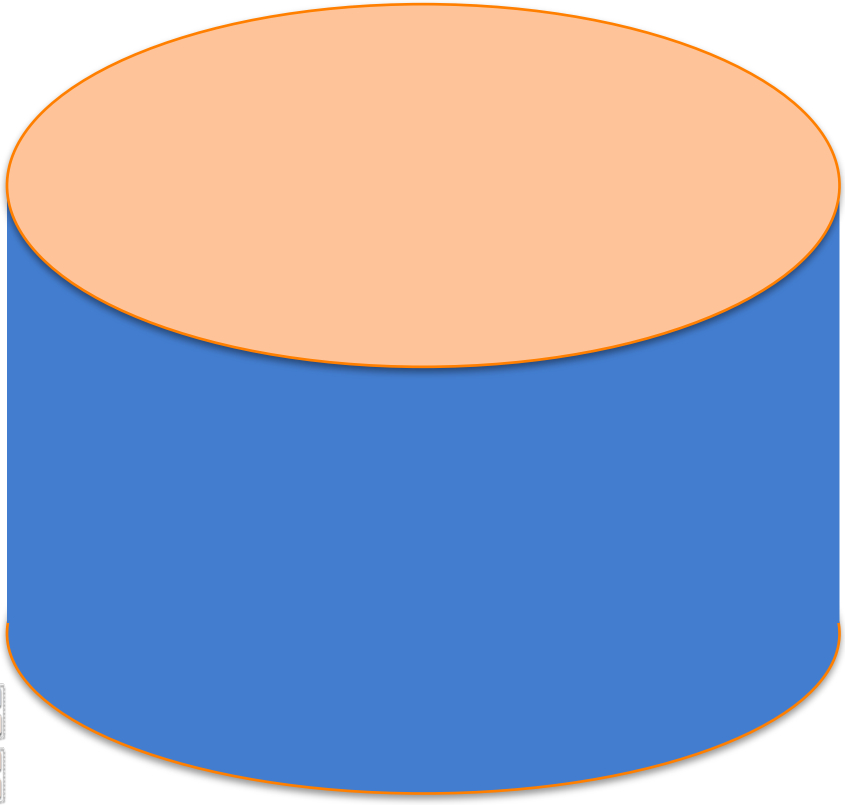
Power consumption is lower during recovery

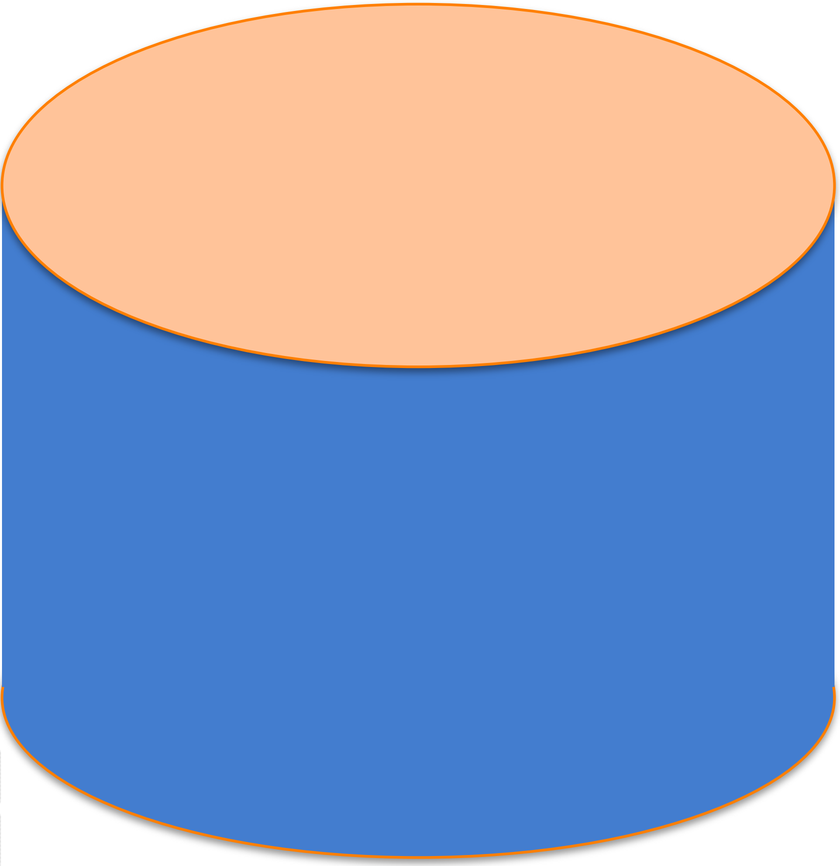
Message logging + Object-based virtualization

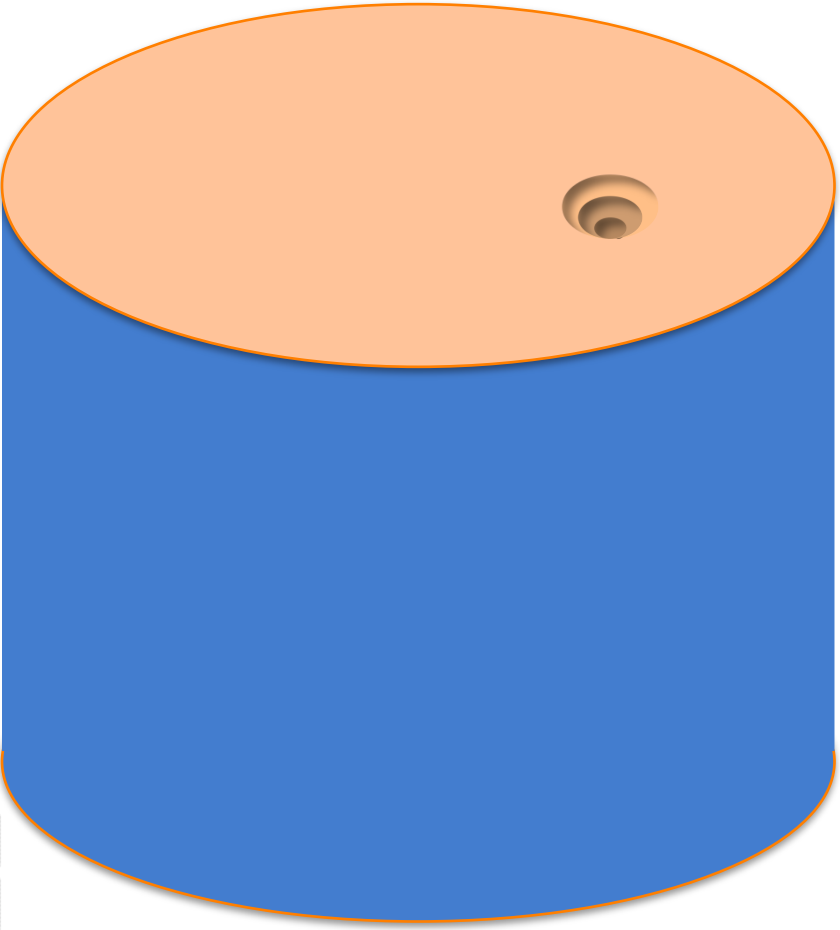
Progress is faster with failures

Fail-stop recovery with message logging: A research vision

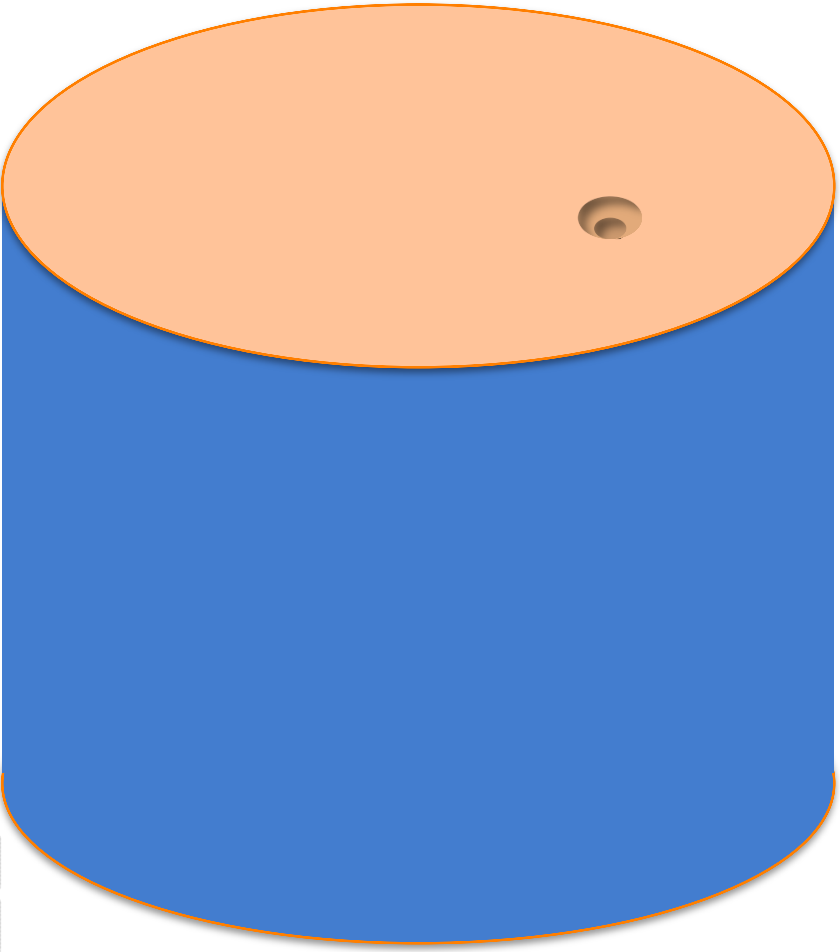






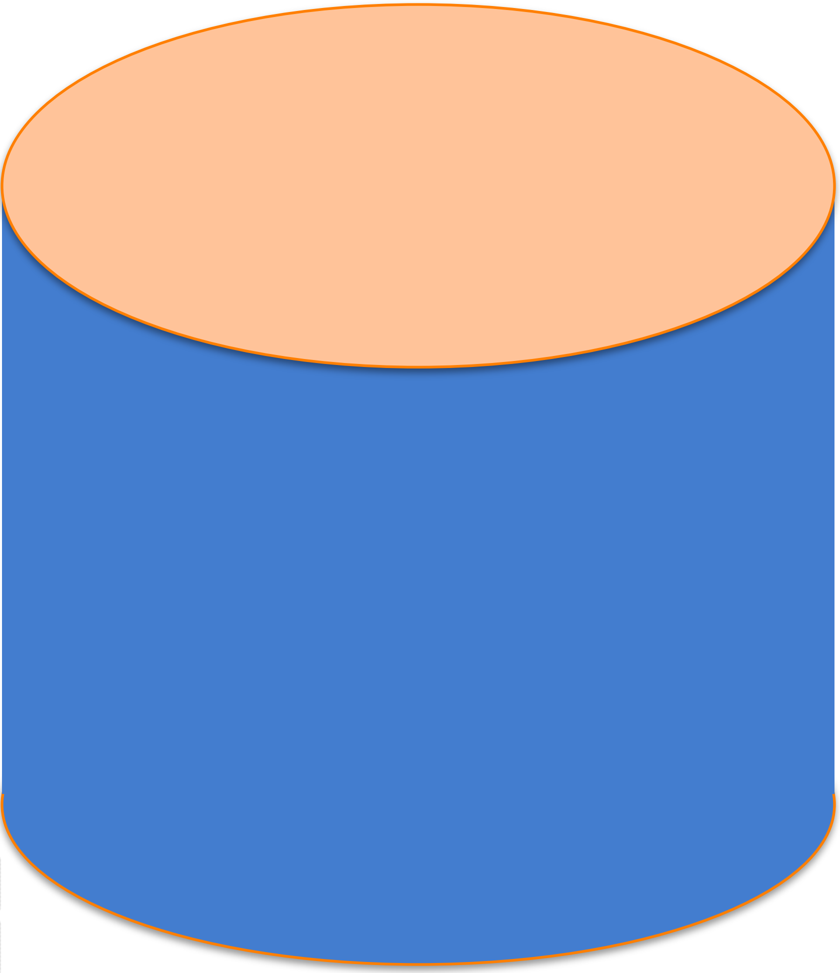


- A fault hits a node
- It regresses..
- Its objects start re-execution,
  - IN PARALLEL on neighboring nodes!

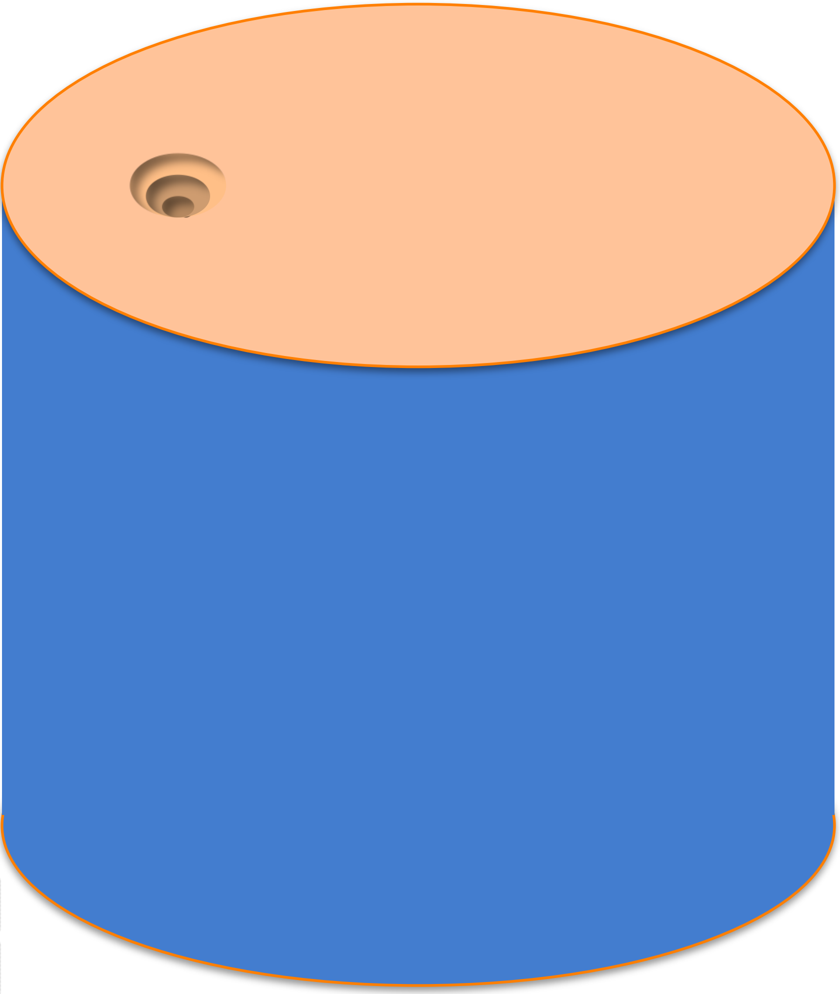


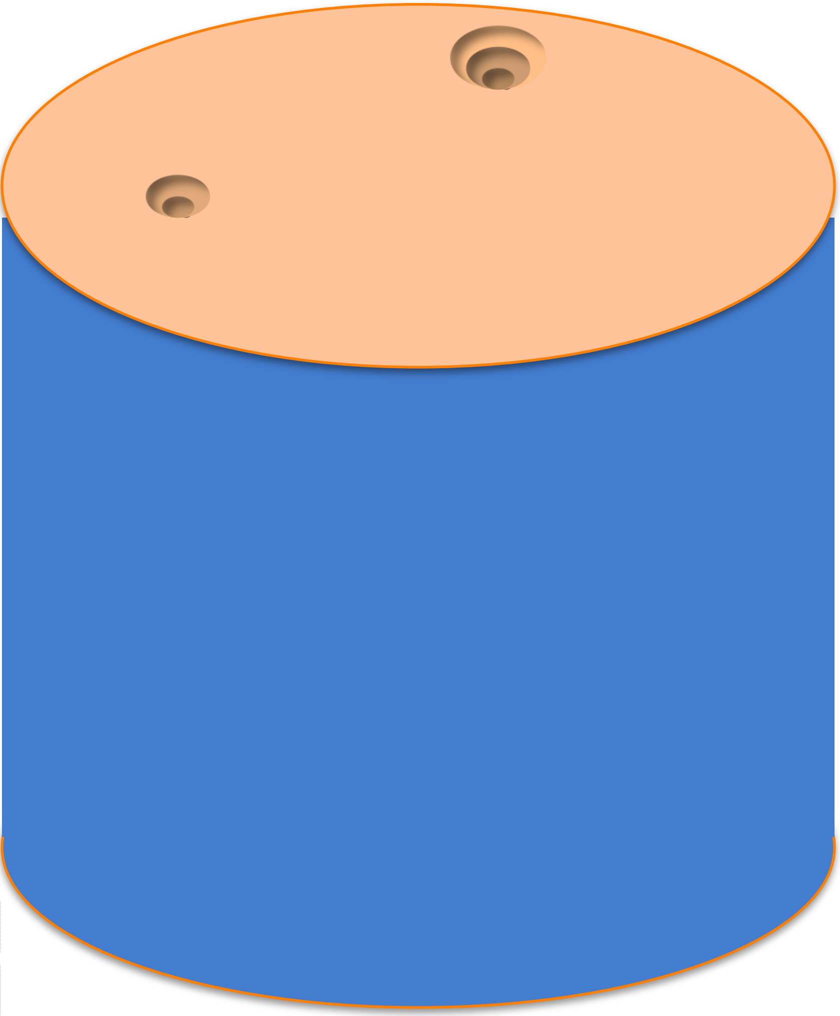
- Re-execution continues even as other nodes continue forward
- Due to “parallel re-execution” the neighborhood catches up

- Back to normal execution

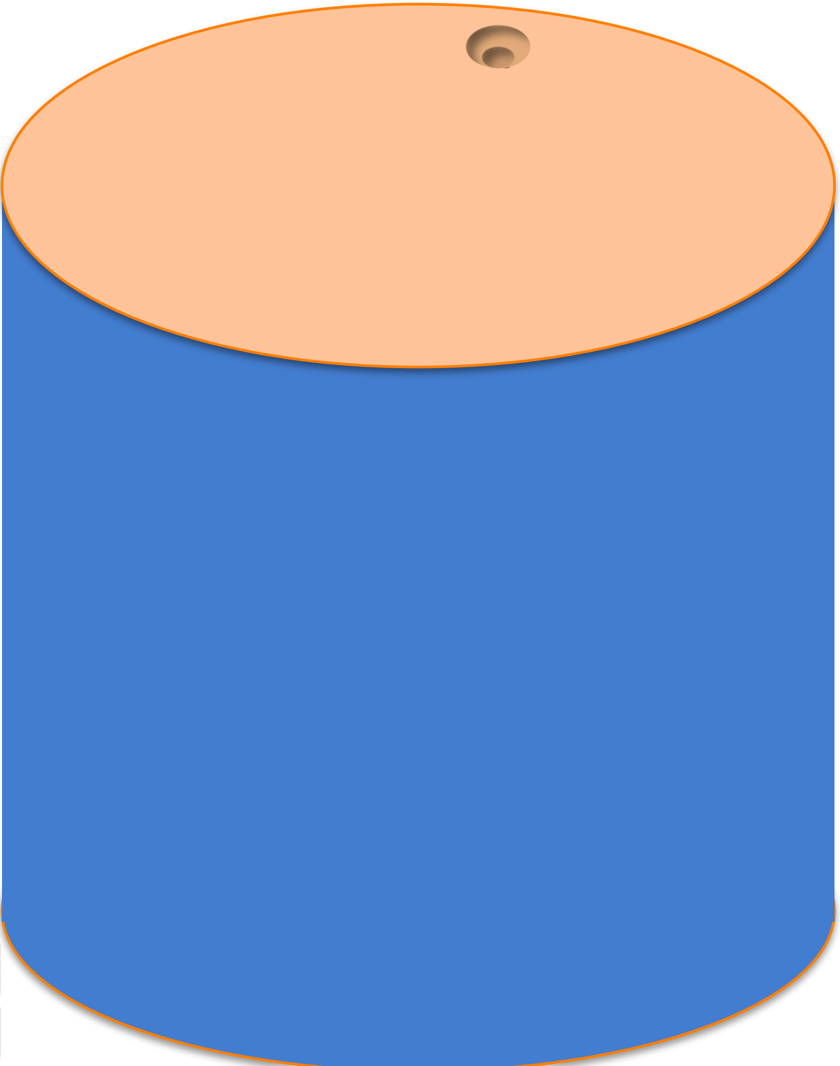


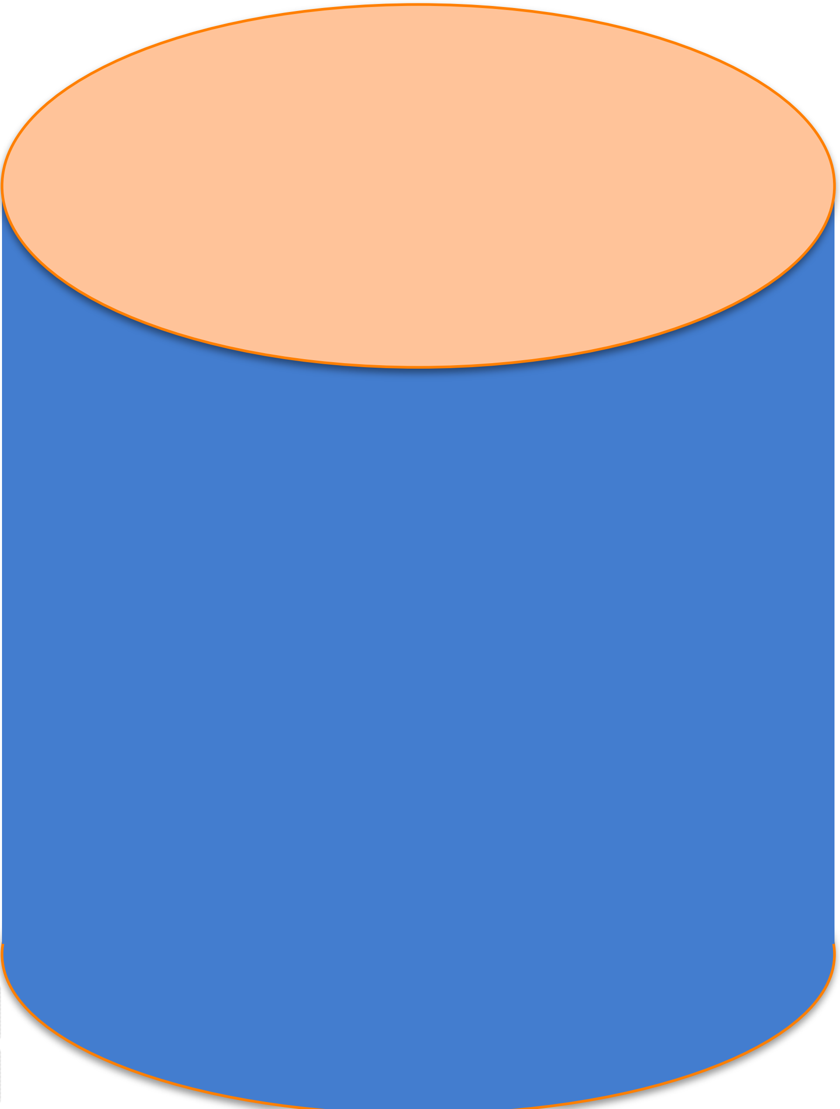
- Another fault





- Even as its neighborhood is helping recover,
- A 3rd fault hits
- Concurrent recovery is possible as long as the two failed nodes are not checkpoint buddies





Takeaway

- Adaptive Runtime System is a good layer to implement resilience strategies
  - Especially with over-decomposition
- In-local-memory double checkpoint with automatic restart works well
- If we need to tolerate more frequent failures
  - Message logging with parallel restart and handling of most concurrent failures will do the job
- Need to combine with SDC handling