Low Design-Risk Checkpointing Storage Solution for Exascale Supercomputers

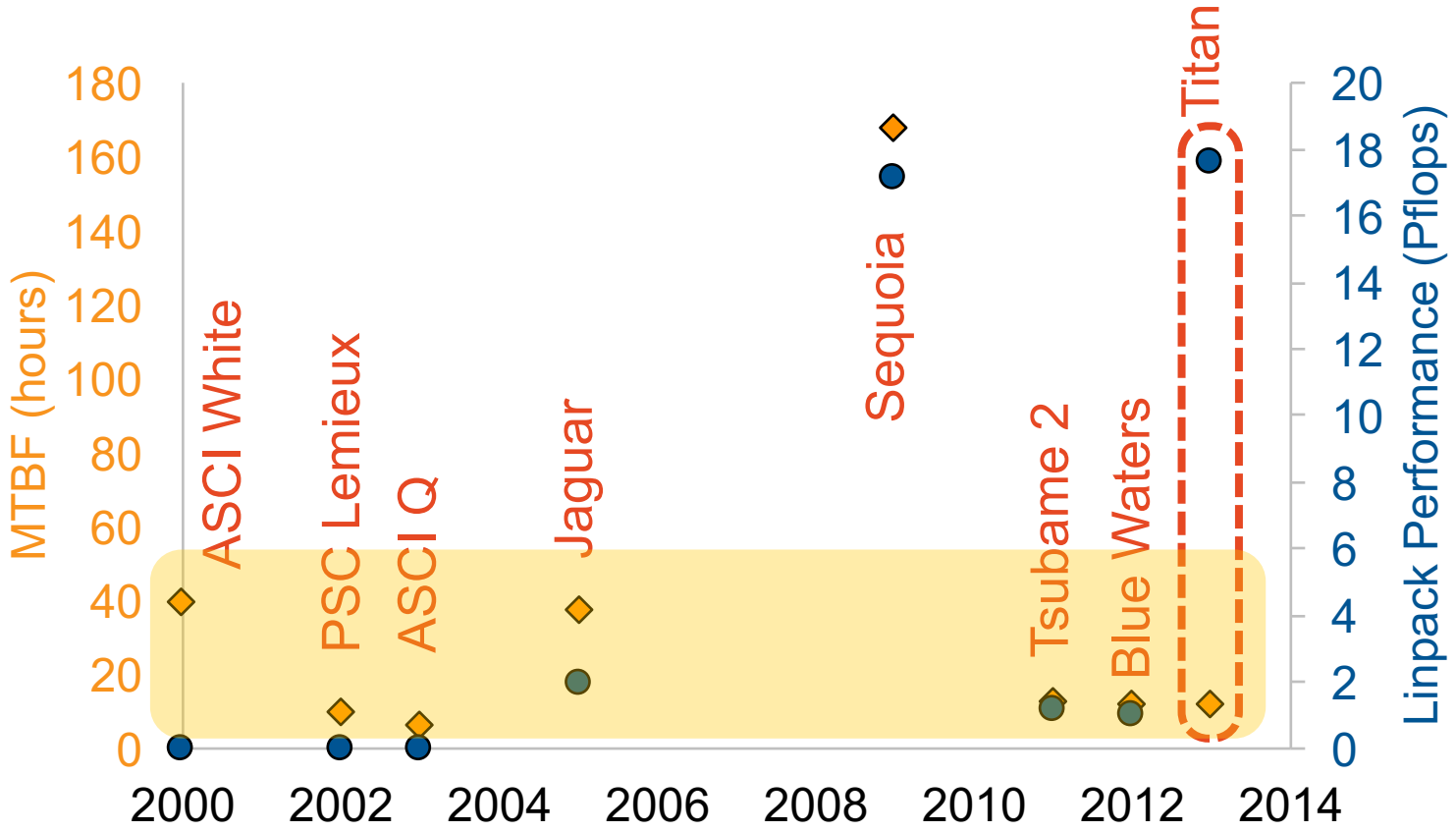
Nilmini Abeyratne and Trevor Mudge

SC16 Doctoral Showcase, Salt Lake City UT USA
November 15, 2016

UNIVERSITY OF MICHIGAN

In collaboration with
ASU ARIZONA STATE UNIVERSITY

Supercomputers fail frequently



Cause	Occurrence	Percentage
Hardware	14341	60.4%
Software	5361	22.6%
Network	421	1.8%
Human	149	0.6%
Facilities	362	1.5%
Undetermined	3105	13.1%
Total	23739	100%

Root causes of failures collected by LANL during 1996-2005. [1]

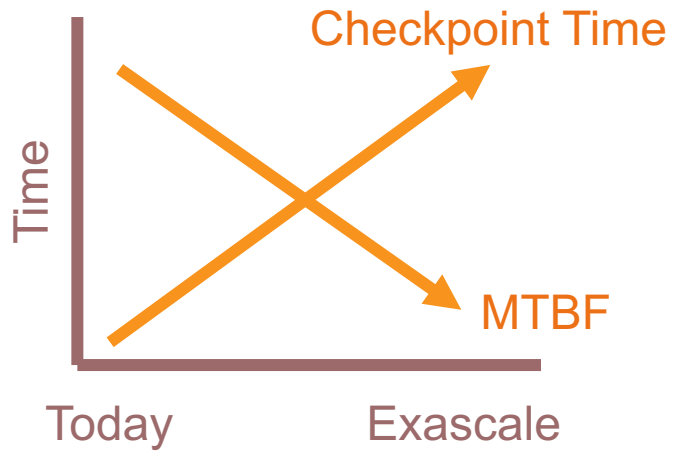
[1] “Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems.” Xiangyu Dong et al. , SC 2009.

Overview of checkpoint/restart

- Simplest, most intuitive fault tolerance technique
- Save the memory state. After a crash, restore saved state

However…

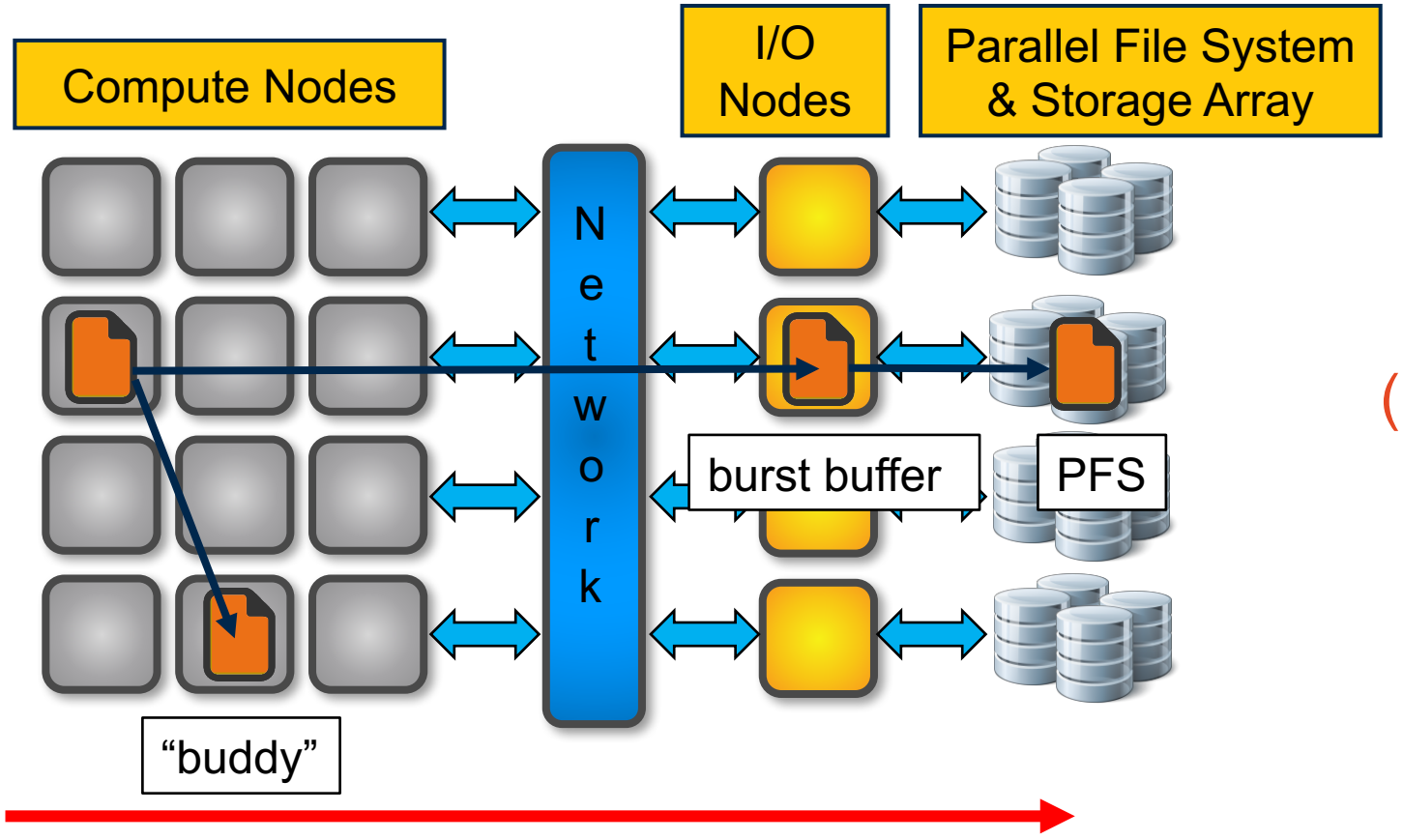
- Checkpoint/restart is getting harder at exascale



Slower checkpointing due to
i. Bigger applications, more data
ii. Faster cores, slow I/O

Frequent failures due to
i. Increasing component count
ii. Decreasing feature sizes

Effective methods to reduce checkpointing time

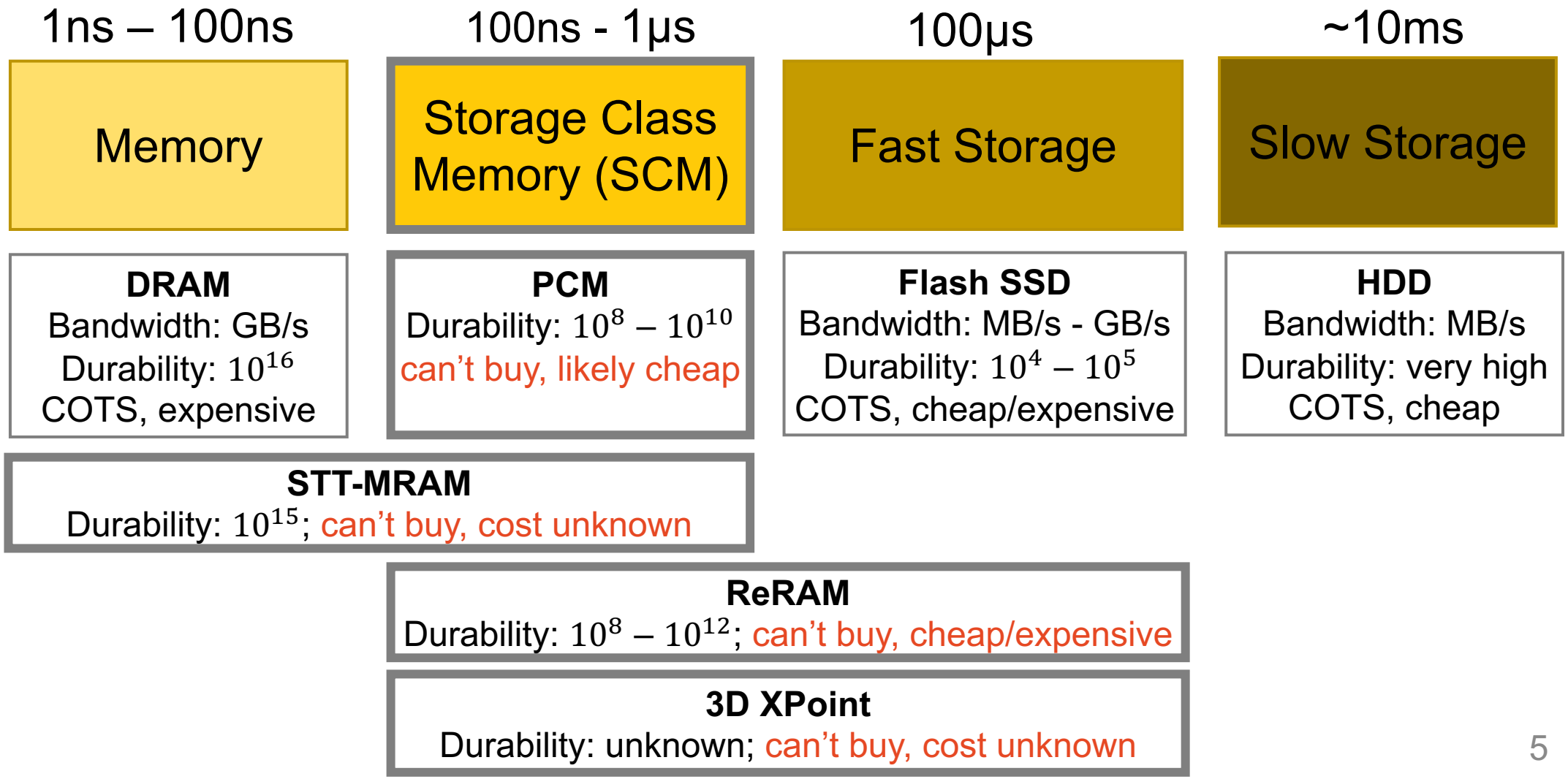


Checkpoint to PFS takes 15-30 minutes due to limited bandwidth

Multi-level checkpointing
(in-memory, buddy, burst-buffers)

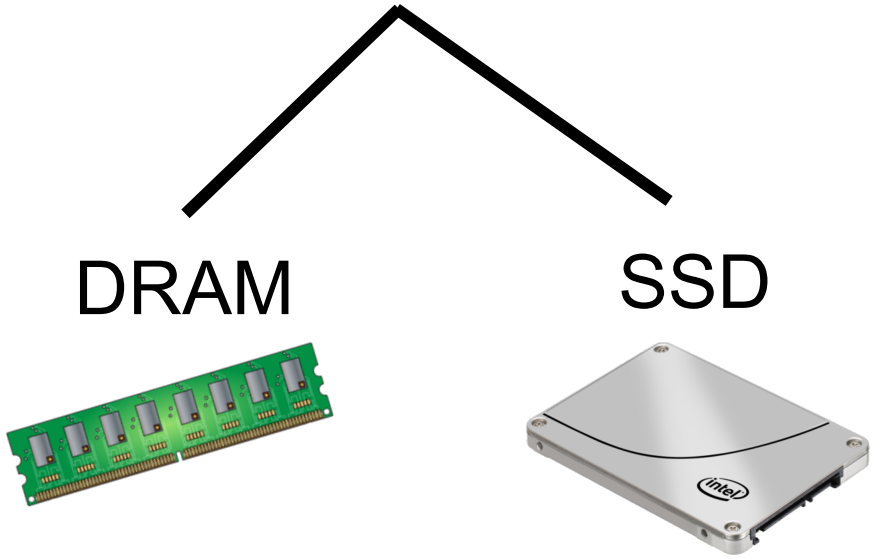
Use non-volatile memory
(flash-SSD, PCM, STT-MRAM)

Non-volatile memories are speculative

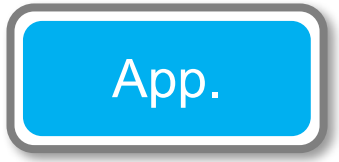


Let’s build a checkpoint framework

on COTS devices



DRAM vs. SSD

	Always Checkpointing to Ramdisk	Always Checkpointing to SSD
Node's Local DRAM		
Node's Local SSD		

✅ Fast
✅ High endurance (10^{16} writes)

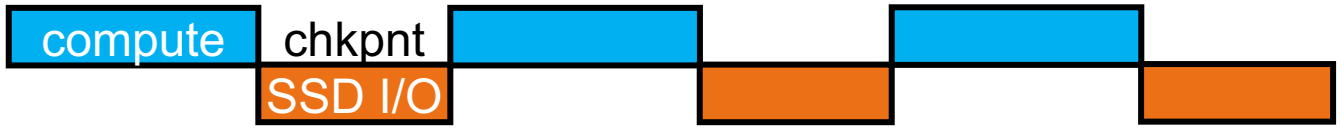
❌ Small capacity
❌ Volatile
❌ More susceptible to soft errors
❌ Less memory for apps
❌ High memory pressure

✅ Non-volatile
✅ Capacious
✅ Cheap (71¢ per GB)
✅ Resilient to soft errors
✅ More memory for apps

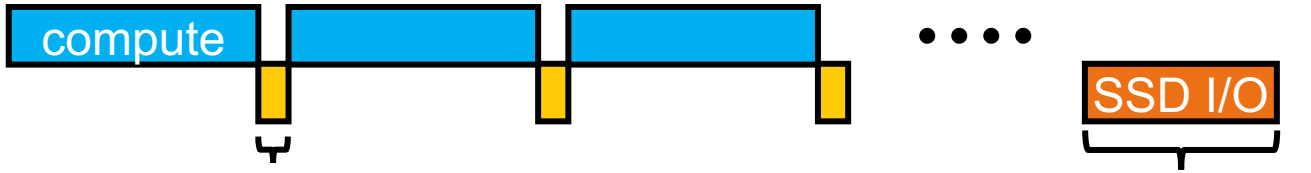
❌ Slow
❌ Low endurance (10^5 write cycles)

OS (unhelpfully) optimizes SSD I/O

What the application think is happening



What is really happening

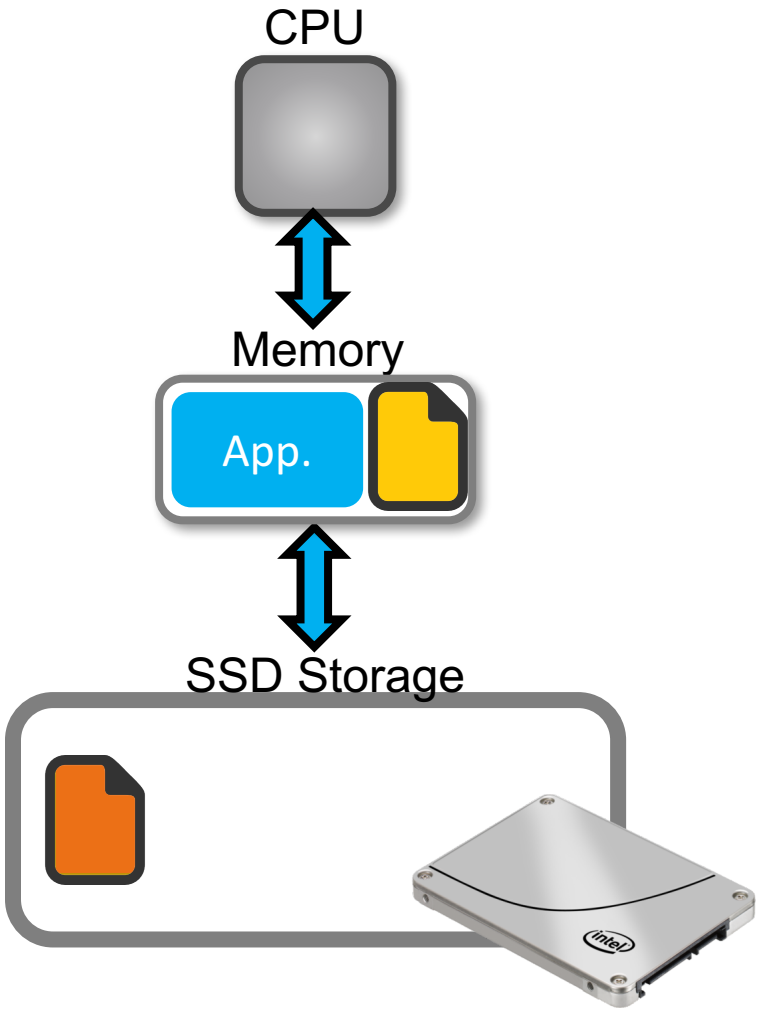


OS hides I/O latency by writing to DRAM's page cache

Delayed I/O flushing

Application thinks checkpoint is in SSD!
Checkpoint is volatile!

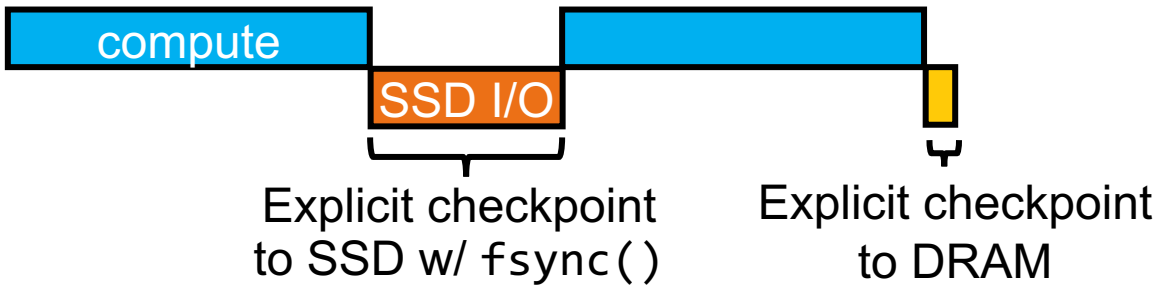
Hybrid DRAM-SSD checkpointing



Best of both worlds

Benefits from DRAM	Benefits from SSD
✅ Speed	✅ Non-volatility / reliability
✅ Endurance	✅ Resilience for soft errors
	✅ Capacity for big checkpoints

Taking explicit control!



Balance reliability and speed

Outline

1. Balance reliability and speed
   - ❖ Hybrid DRAM-SSD Checkpointing
2. Maintain SSD lifetime
   - ❖ SSD lifetime-aware checkpoint controller
3. Protect DRAM checkpoints against soft errors
   - ❖ Strong ECC for checkpoints in DRAM

Checkpoint Location Controller (CLC)

Three Decision Parameters:

1. SSD Lifetime Estimation

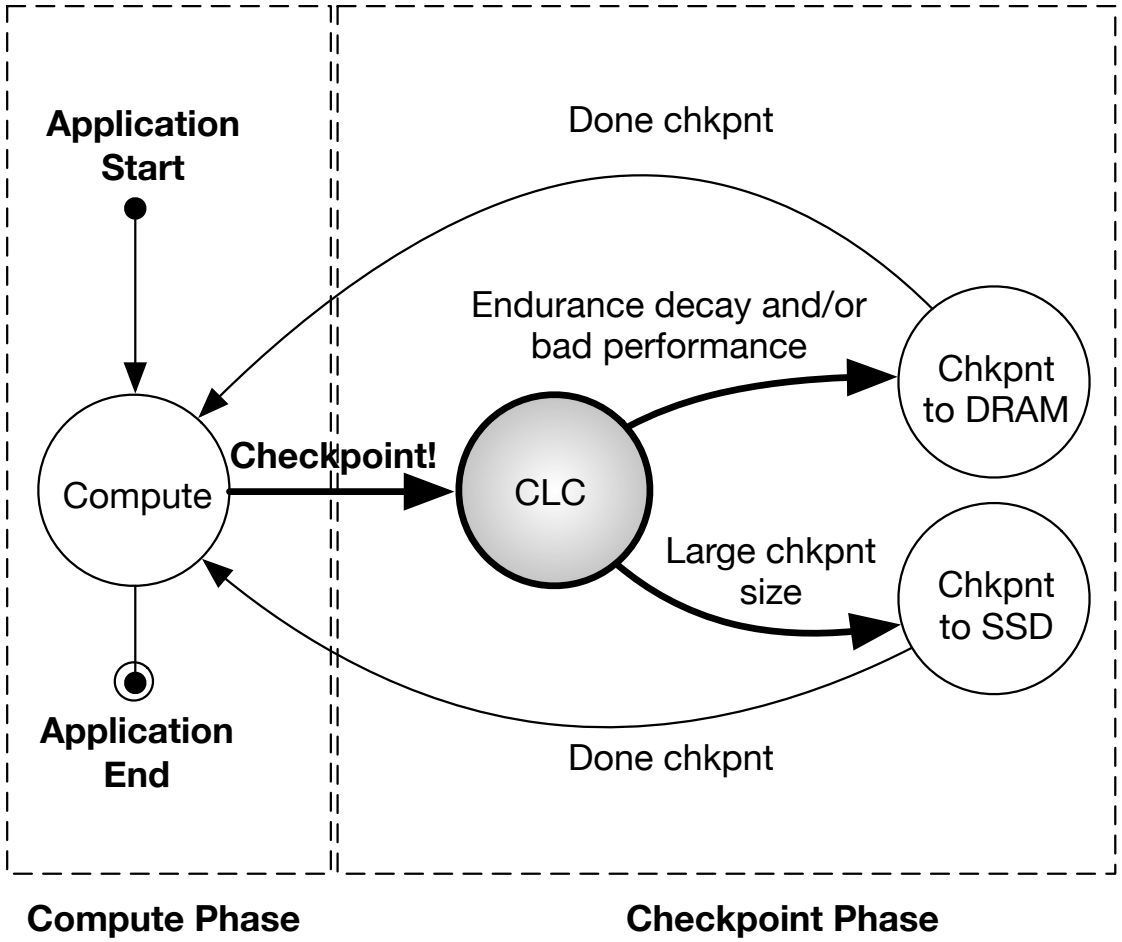
Compare bandwidth to SSD PBW rating

2. Performance Loss Estimation

Compare slowdown to user-set bound

3. Checkpoint Size

Send large checkpoints always to SSD



Checkpoint Location Controller (CLC)

Three Decision Parameters:

1. SSD Lifetime Estimation

Compare bandwidth to SSD PBW rating

e.g. rating = 10 petabytes written over 5 years

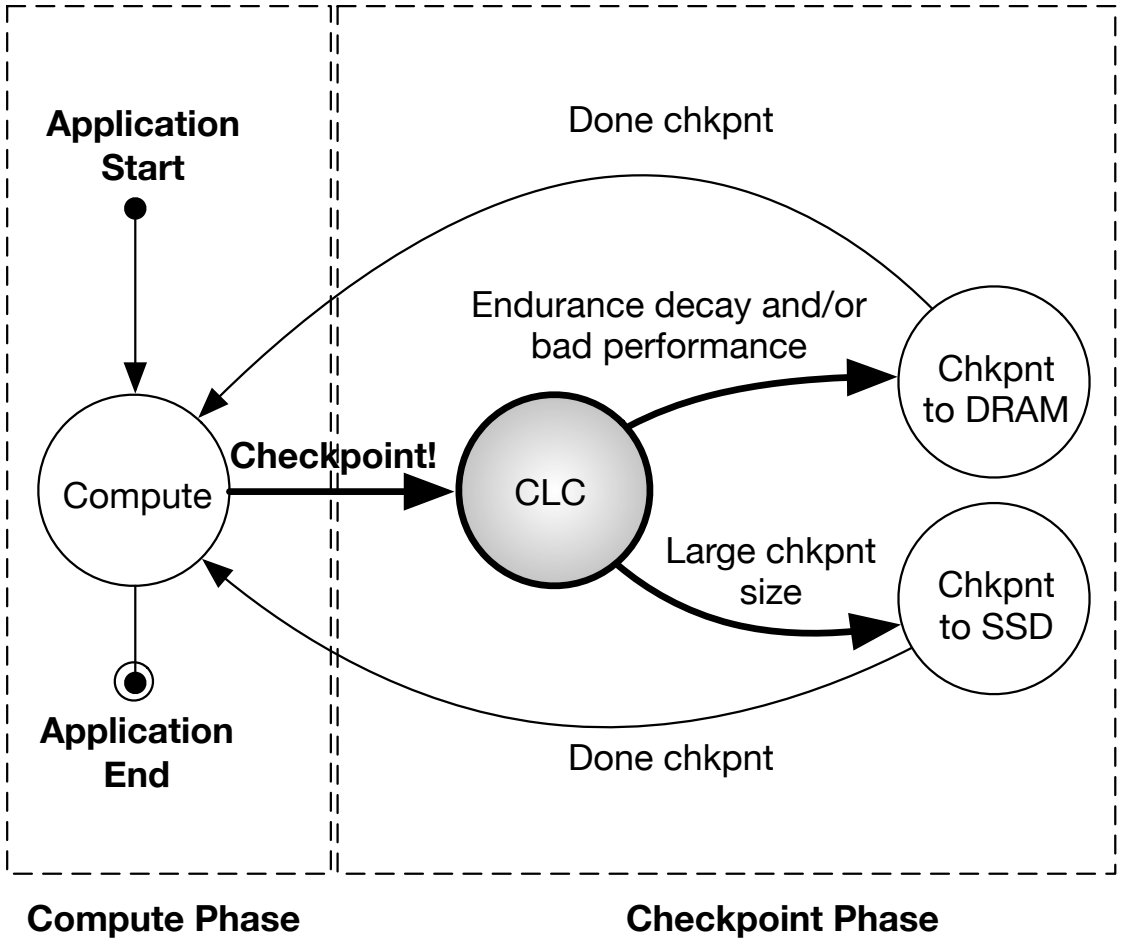
CLC disallows more than ~230 GB/hour

2. Performance Loss Estimation

Compare slowdown to user-set bound

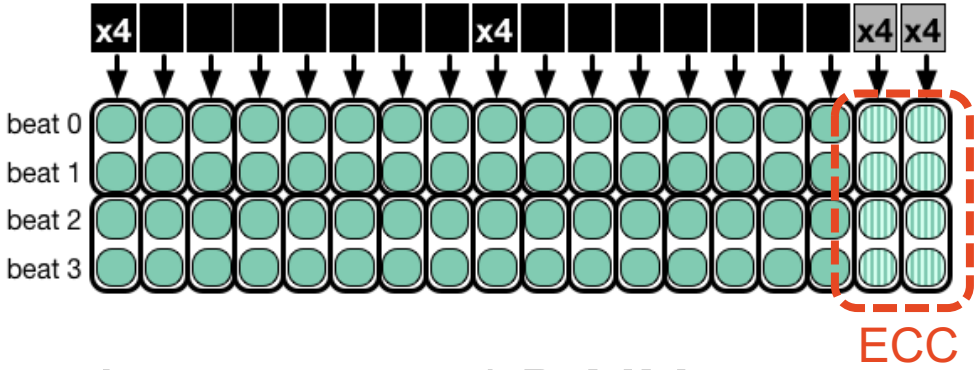
3. Checkpoint Size

Send large checkpoints always to SSD



Dual-ECC memory

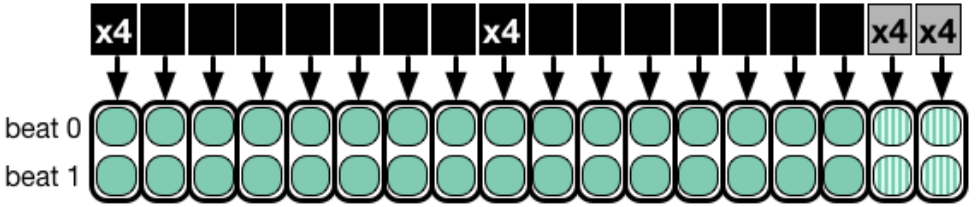
Normal ECC – protects regular data with low overhead



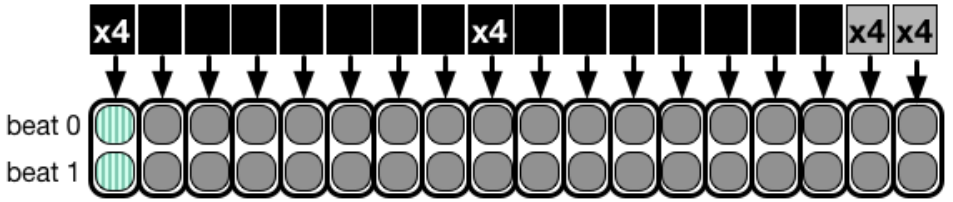
- Access one x4 DIMM
- RS(36,32)
- 32 data + 4 ECC symbols
  - i. 2 symbol correction
  - ii. 4 symbol detection
  - iii. 1 symbol correction, 3 symbol detection

Our choice. Detection focused, not chipkill

Strong ECC– protects checkpoint data with Chipkill-level strength

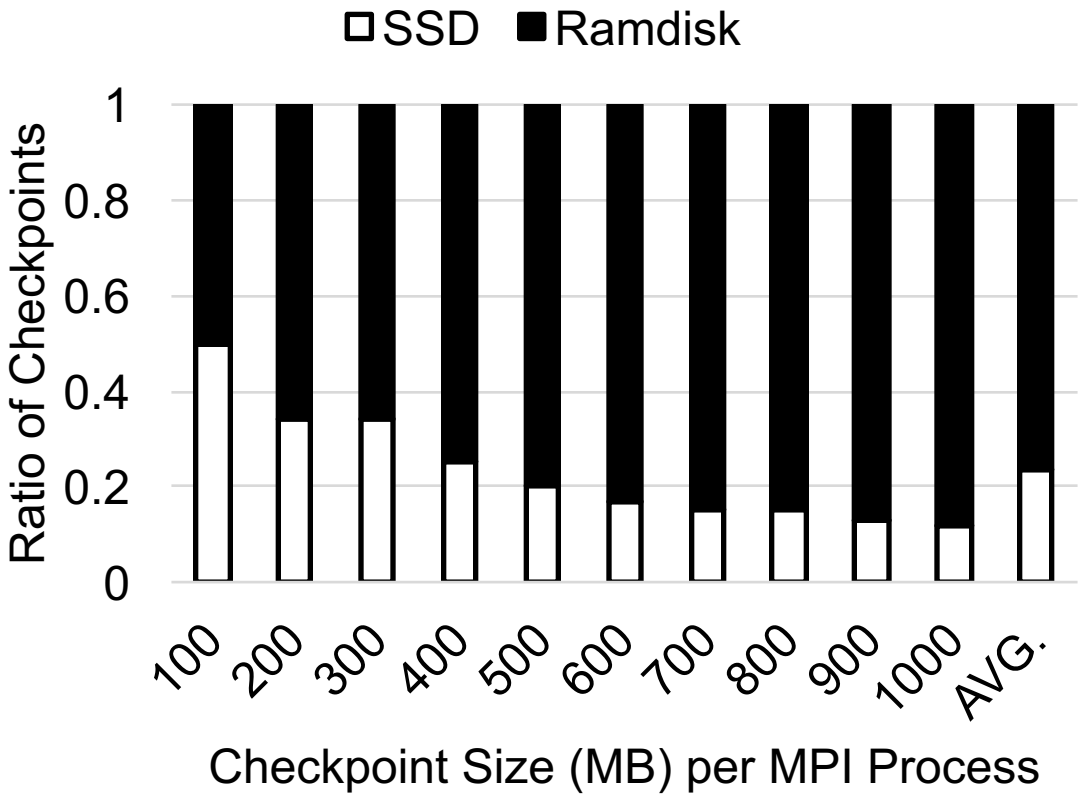


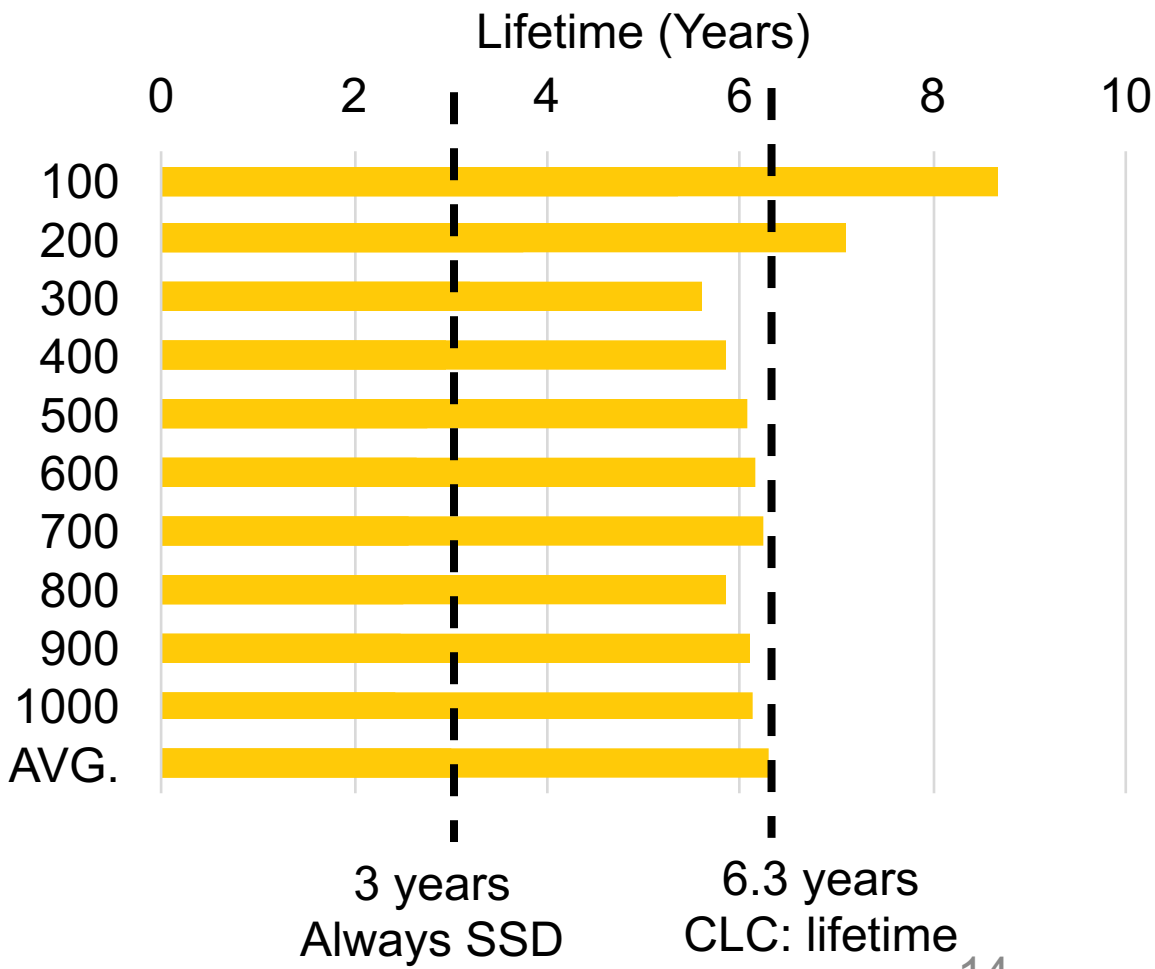
- Access one x4 DIMM
  - 16 data + 2 ECC symbols
  - RS(18,16) detects up to 2 failed chips



- IFF failure exists, trigger 2nd access
  - Retrieve 1 more parity symbol
  - RS(19,16) corrects 1 symbol

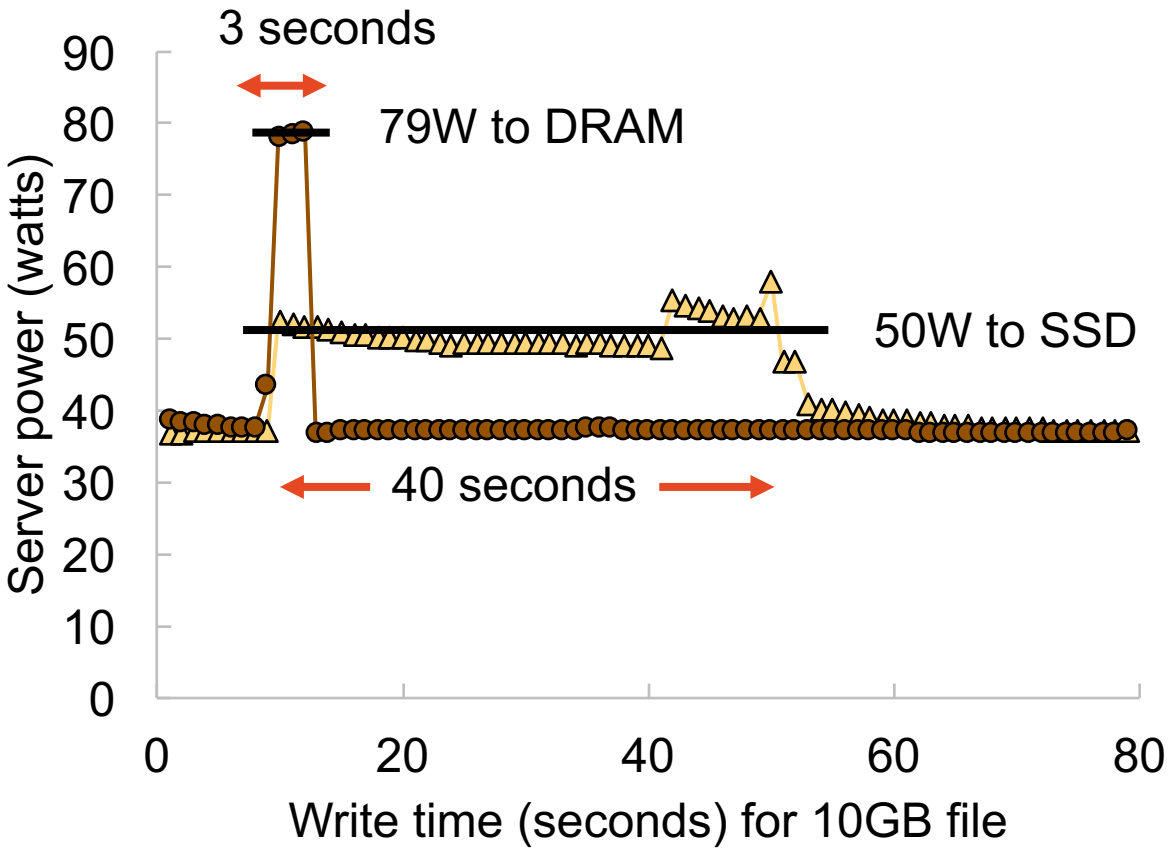
CLC improves SSD lifetime



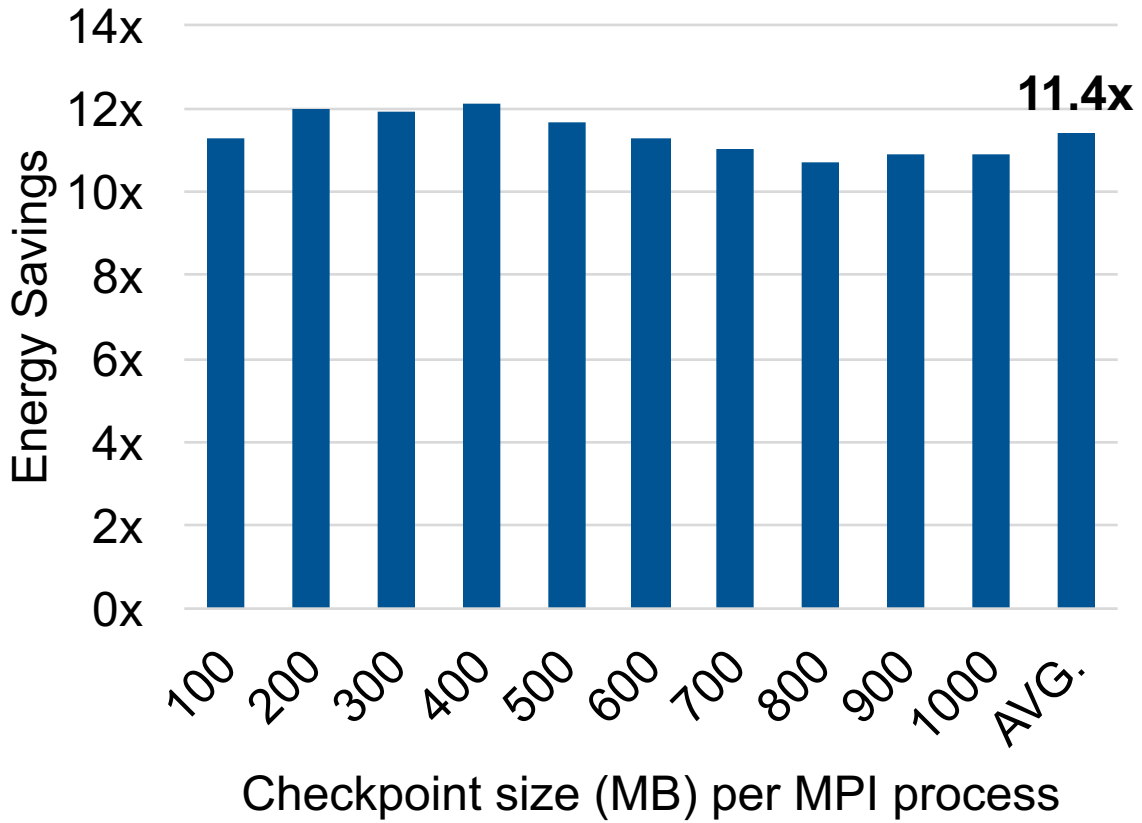


Microbenchmark:
64 MPI processes on 8 nodes
Checkpoints every 5 seconds, for 100 iterations

Hybrid checkpointing saves energy

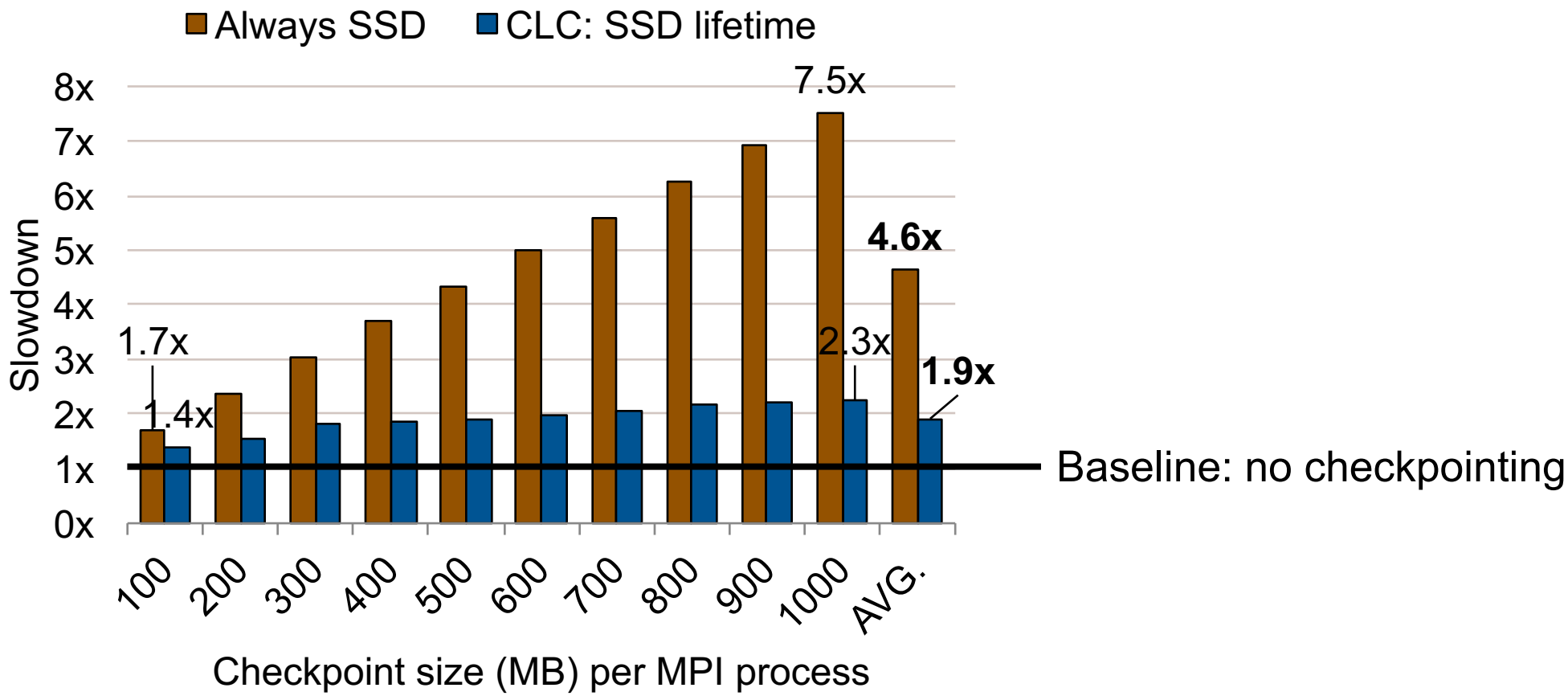


DRAM consumes more power, but for a shorter time

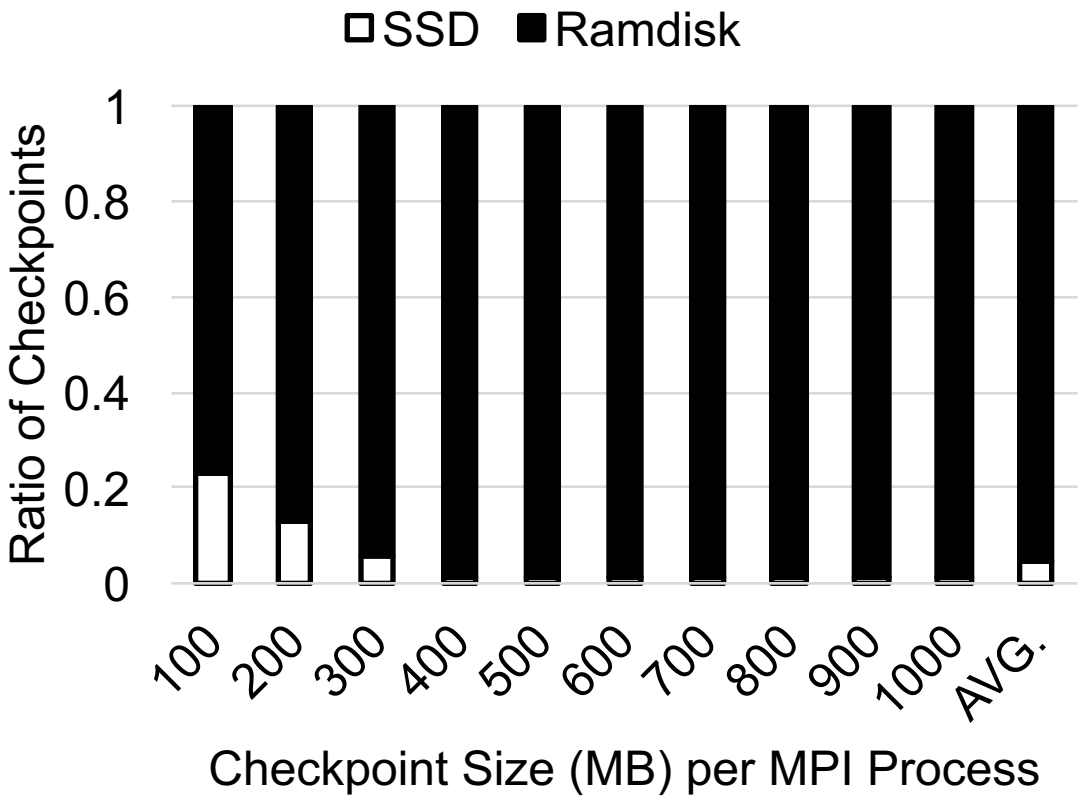


Hence, selective DRAM checkpointing saves energy

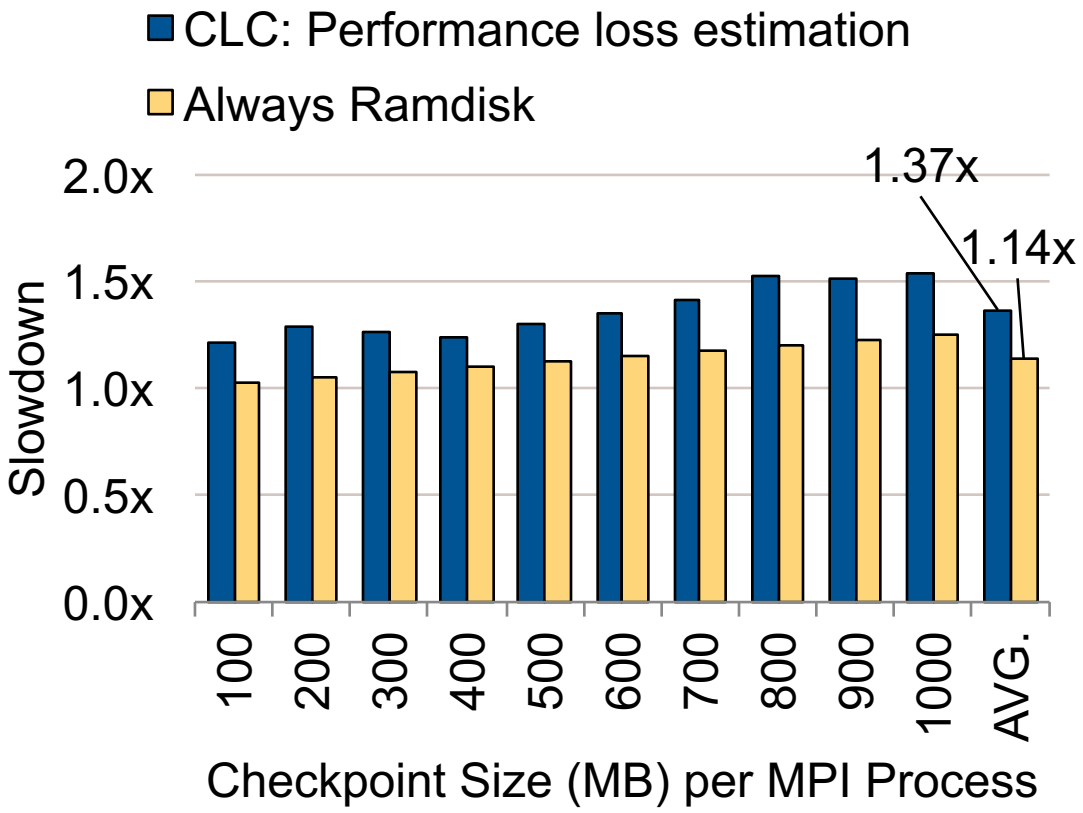
Hybrid checkpointing saves performance



CLC decreases performance loss



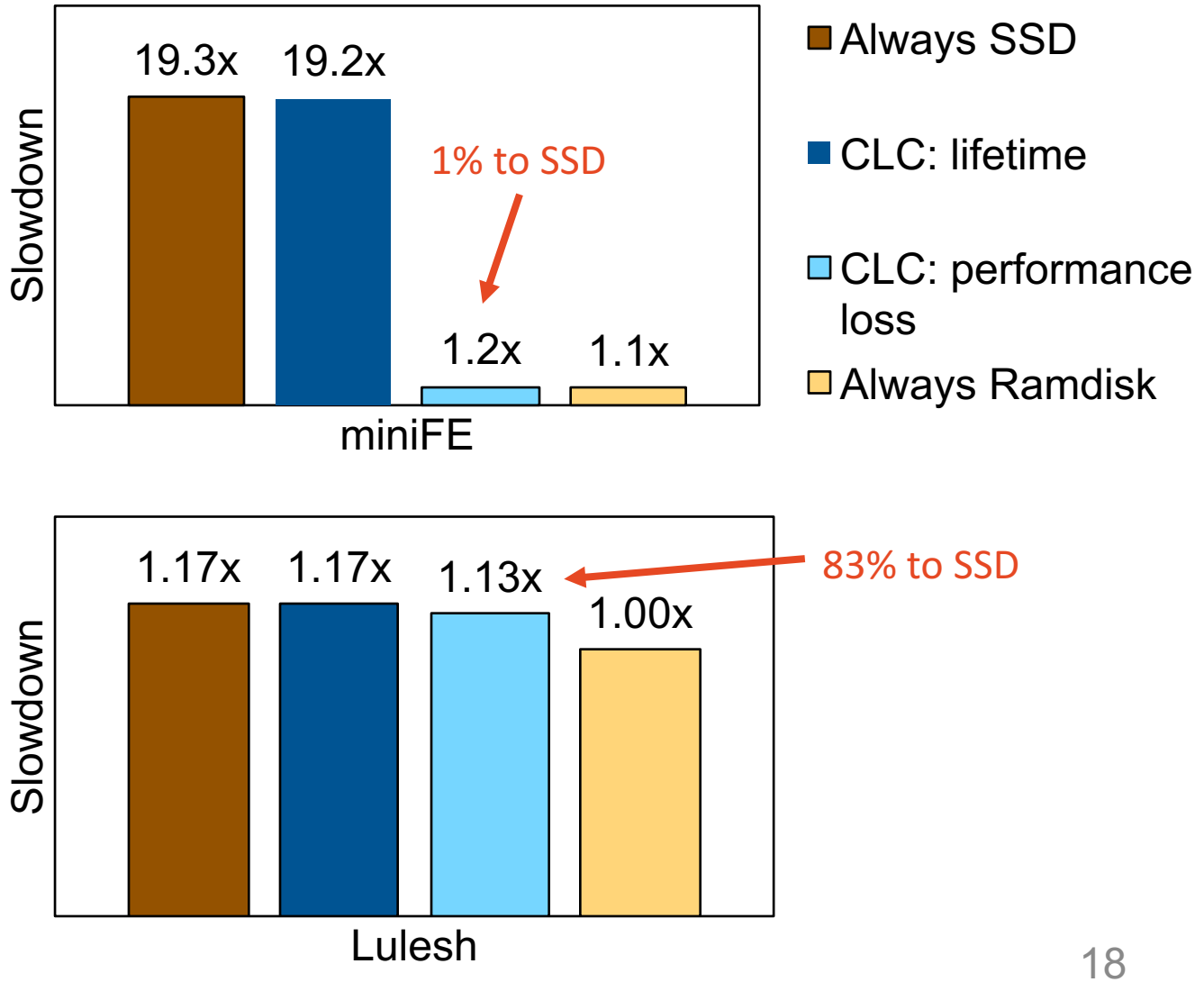
Fewer checkpoints are written to the SSD



Slowdown decreased to 1.37x
(1.4x accounting strong ECC overhead)

Application results

	miniFE	Lulesh
Parameters	528×512×768	45×45×45
Setup	64 MPI processes, 8 nodes 24GB/node	
Checkpoint sizes:		
1 MPI proc:	50 MB	8 MB
1 node:	400 MB	64 MB
App. Total:	3.1 GB	512 MB
Baseline runtime	236 sec.	74,470 sec.
Checkpoint behavior	once/iter, ~1 sec/iter, 200 iters,	once/iter, ~11 sec/iter, 6,499 iters

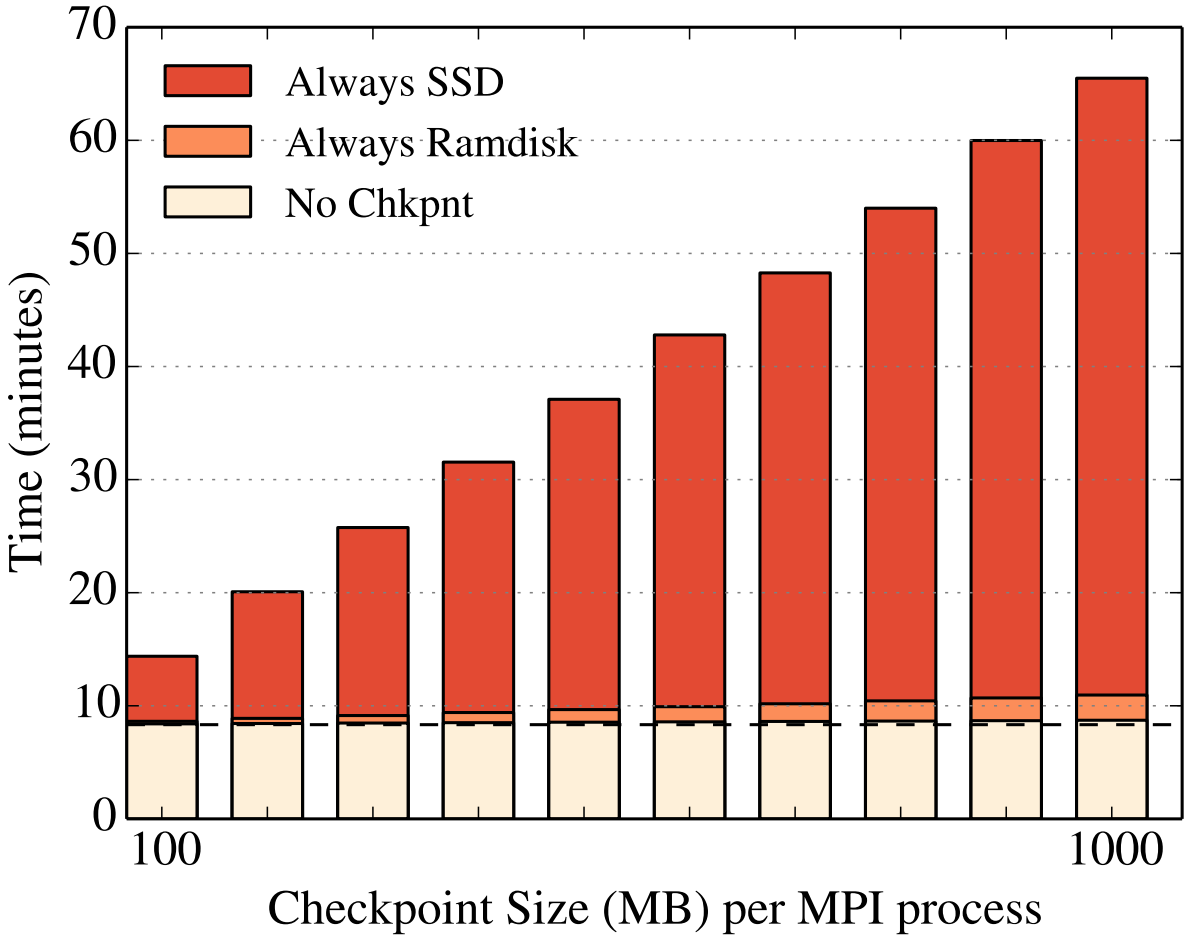


Summary of contributions

- Low-risk checkpointing framework using COTS memory devices
- Fast and reliable hybrid DRAM-SSD checkpointing
- Extends SSD lifetime with CLC
  - Minimizes checkpointing overhead
  - Saves energy
- Chipkill-level protection for checkpoints with dual-ECC memory

Backup Slides

SSDs are slow



Microbenchmark

64 MPI processes on 8 nodes

checkpoints to local DRAM or SSD

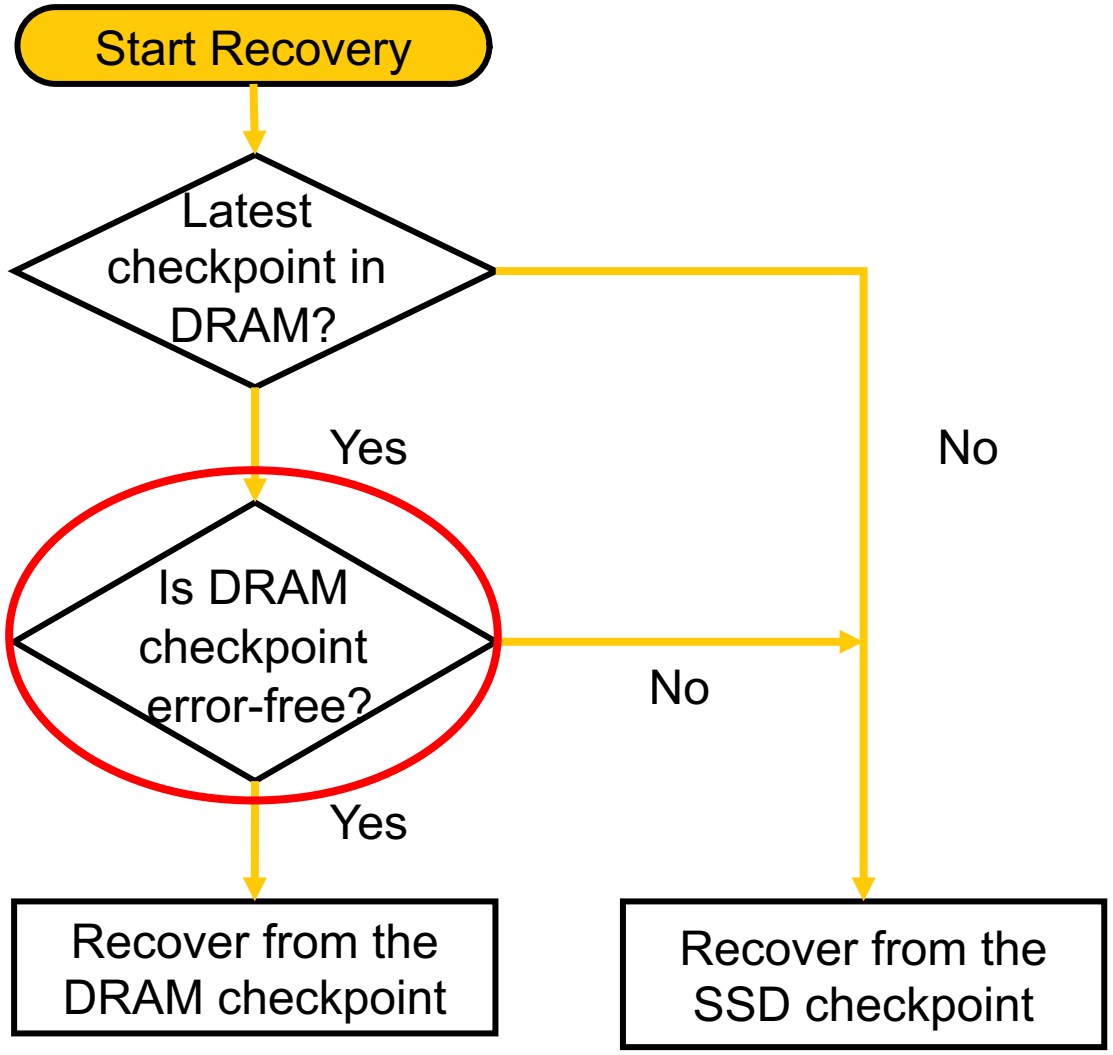
every 5 seconds, for 100 iterations

CLC

Algorithm 1 Checkpoint Location Controller (CLC)

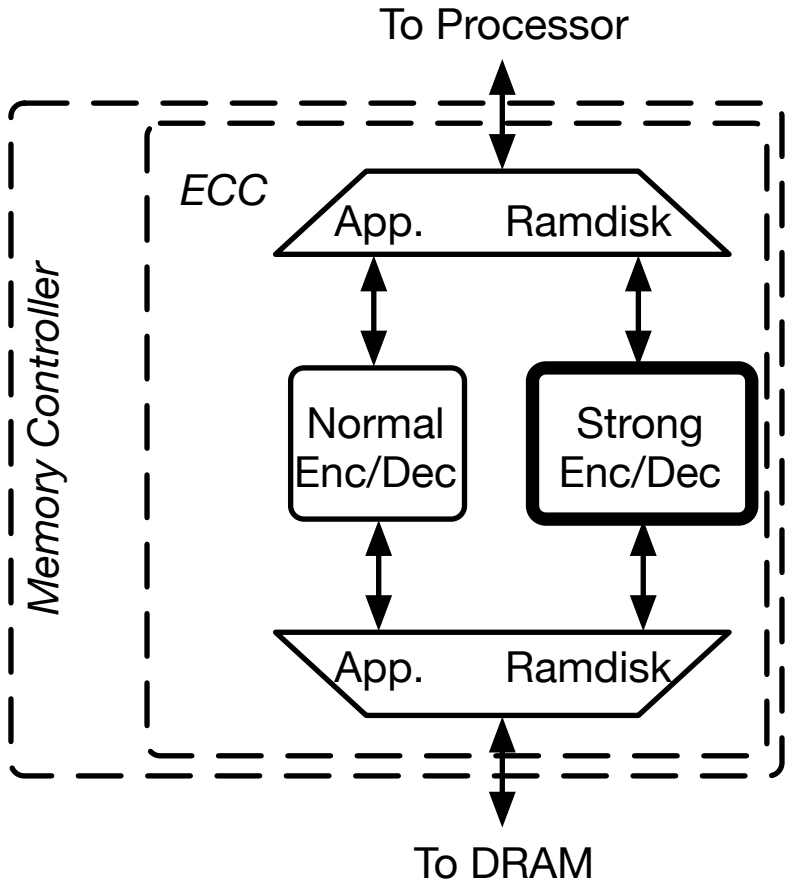
```
1: function CLC(D, r, i) ▷ Where D - data, r - MPI rank,
   i - chkpnt number
2:     L_estimated, L_expected = lifetimeEstimation()
3:     T_slowdown = performanceEstimation(T_chk, T_total)
4:     if L_estimated > L_expected then
5:         loc = SSD
6:         if T_slowdown > bound then
7:             loc = RAM
8:         end if
9:     else
10:        loc = RAM
11:    end if
12:    size_limit = TMPFS_SIZE/numMPIRanks
13:    if loc == RAM and sizeof(D) > size_limit then
14:        return
15:    end if
16:    writeCheckpoint(loc, D, r, i)
17:    update(T_chk)
18: end function
```

Check DRAM checkpoint for errors



Which is the best ECC to protect DRAM checkpoints?

Modification to the Memory Controller



Decoding latencies

Table 3: Synthesis results for proposed RS codes

	RS(36,32)	RS(18,16)	RS(19,16)
Syndrome Calculation	0.48ns (σ)	0.41ns (ρ)	0.41ns (ρ)
Single Symbol Correction & Double Symbol Detection	N/A	N/A	ρ + 0.47ns
Single Symbol Correction & Triple Symbol Detection	σ + 0.47ns	N/A	N/A