PMShifter: Enabling Persistent Memory Fluidness in Linux

Theodore Michailidis, Steven Swanson, Jishen Zhao

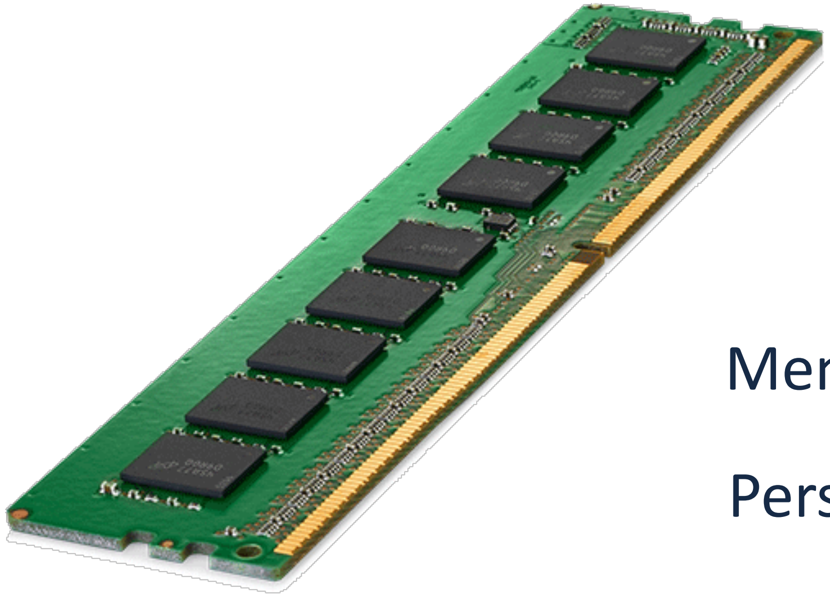
UC San Diego

System and Architecture Lab on Scalability, Reliability and Energy-Efficiency

Department of Computer Science & Engineering

University of California, San Diego

Memory costs in datacenters



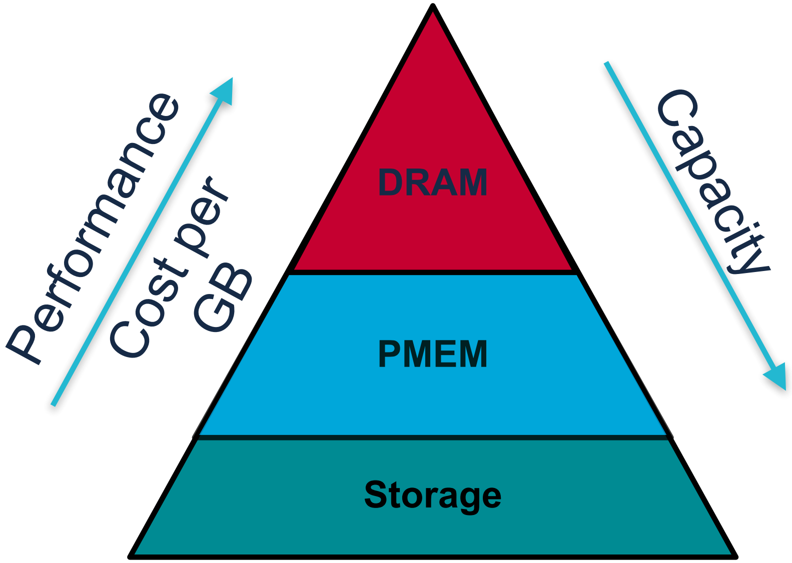
Memory accounts for 40-50% of modern data center costs

Persistent Memory (PMEM) can help alleviate these costs

STA
BLE

Persistent Memory in storage stack

- **2x higher** latency
- **7x lower** bandwidth
- **8x larger** capacity
- **Fraction of the cost**



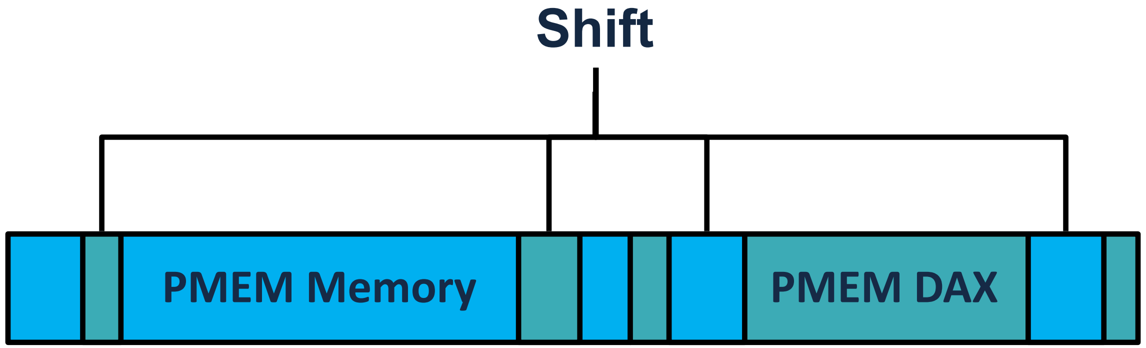
Persistent Memory Operation Modes

PMEM Memory

PMEM DAX

Memory and DAX at the same time

Persistent Memory Operation Modes



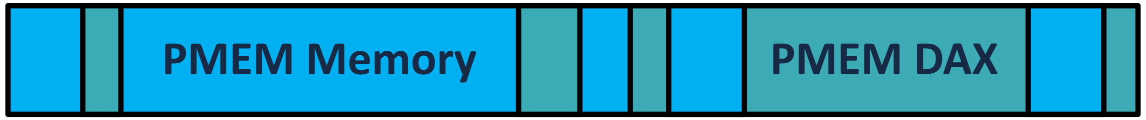
Division through command line

Access to machine

Impractical!

Wasteful!

Our idea: PMShifter



Key Idea: Dynamic shifting to utilize unused PMEM DAX chunks

PMShifter

Enables **dynamic PMEM shifting**

Proposes **associated memory management fixes**

- **Accelerates** compaction and migration
- **Alleviates** fragmentation
- **Fixes** PMEM issues in NUMA

The rest of the talk

- Background & Motivation
- PMShifter
- Evaluation
- Conclusion

STΔ
BLE

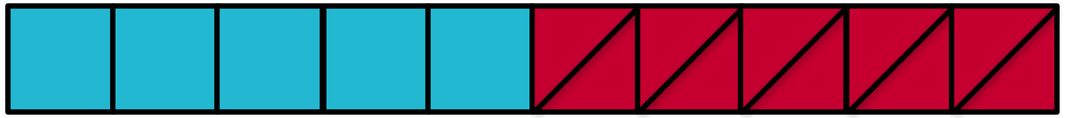
Overview

- Background & Motivation
- PMShifter
- Evaluation
- Conclusion

Memory Hot(un)plug

- Increasing/decreasing size of available memory during runtime (e.g. faulty DIMMs, capacity on demand) on a region granularity (e.g. 2GB)
- PMShifter uses memory hot(un)plugging to shift between memory and DAX
- Memory regions are hot(un)plugged by **onlining/offlining** pages within

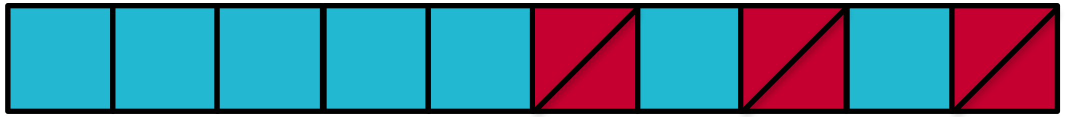
Memory Hot(un)plug



🟦 Online page

▨ Offline page

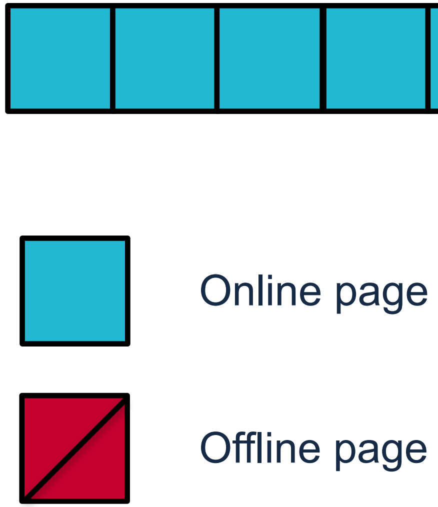
Memory Hot(un)plug



🟦 Online page

🟥 Offline page

Memory Hot(un)plug

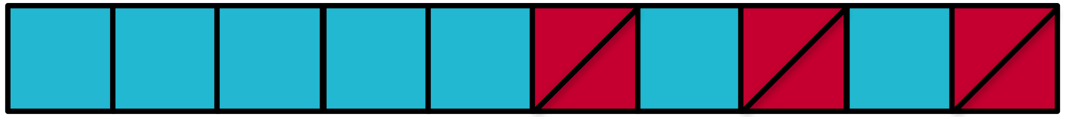


Onlining or offlining free page:

- Update page metadata

Memory Hot(un)plug

Goal: Minimize offlining allocated pages



🟦 Online page

▨ Offline page

Offlining an allocated page:

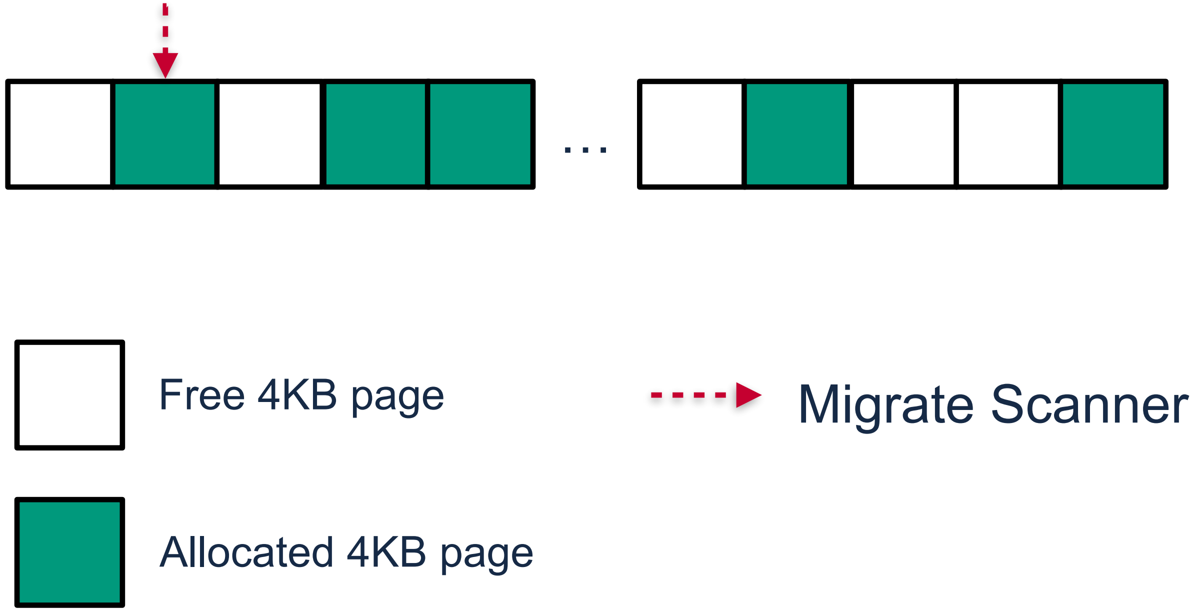
- Allocate a new page
- Copy contents
- TLB entries invalidated
- Update page metadata

Memory Compaction in Linux

- The process for **tackling memory fragmentation**, a key memory management issue
- A fragmented memory can **increase** allocation latency by **~3x**
- Based on an **iterative 3-step** process

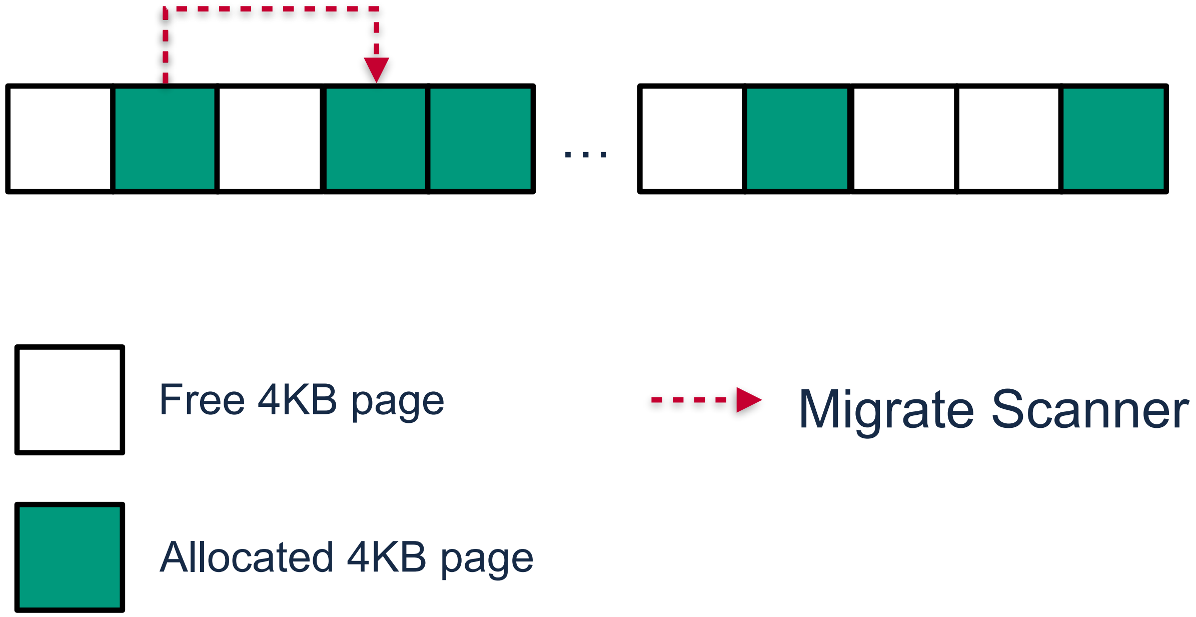
Memory Compaction in Linux

Step 1: Gather allocated pages from **start** of 2MB block (**start** of address space)



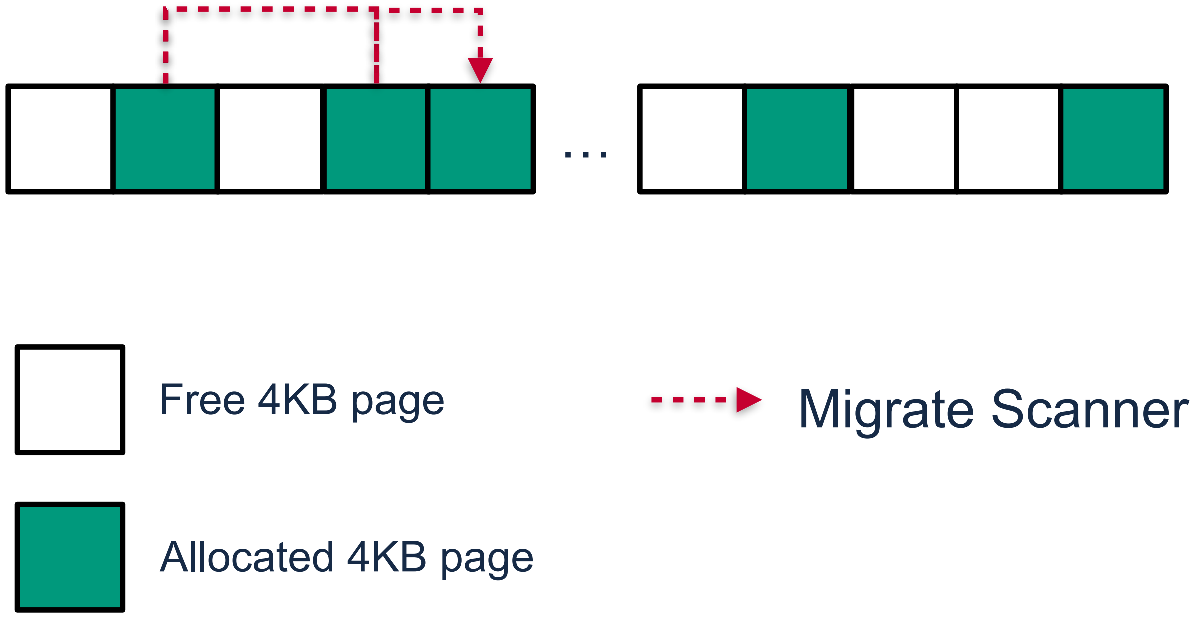
Memory Compaction in Linux

Step 1: Gather allocated pages from **start** of 2MB block (**start** of address space)



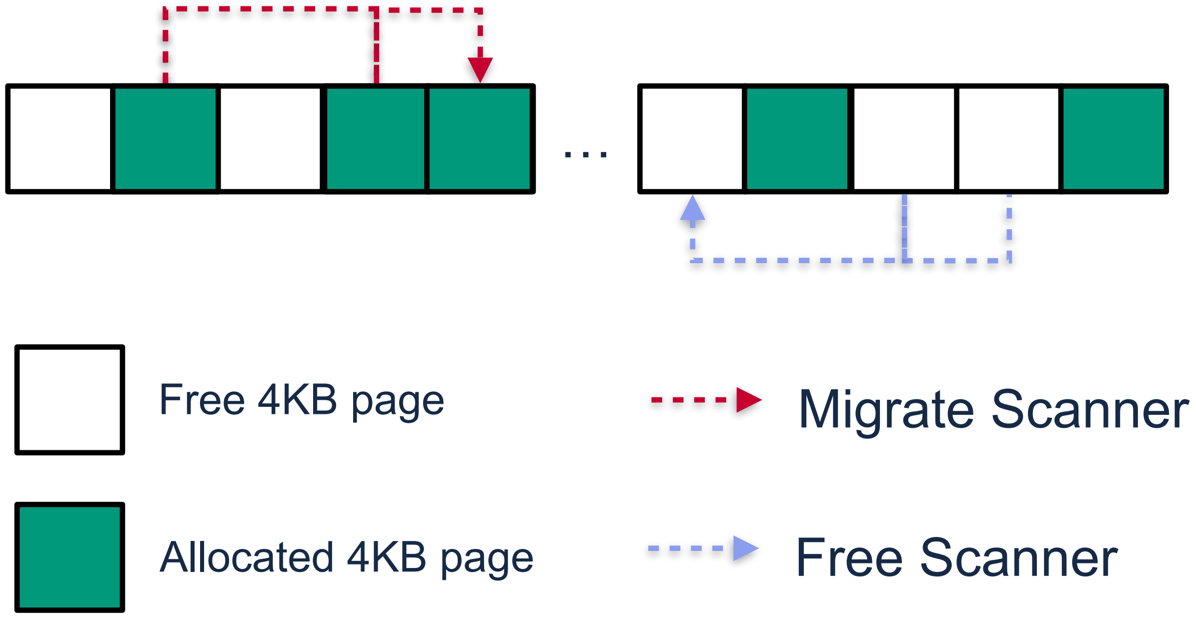
Memory Compaction in Linux

Step 1: Gather allocated pages from **start** of 2MB block (**start** of address space)



Memory Compaction in Linux

Step 2: Scan for free pages from **end** of 2MB block (**end** of address space)



□ Free 4KB page

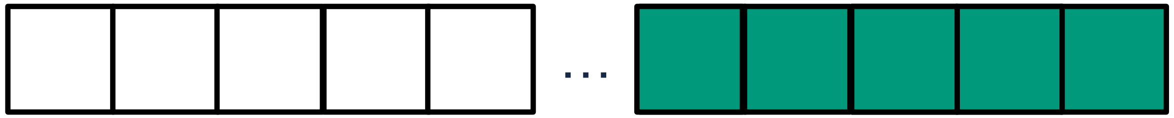
🟩 Allocated 4KB page

Migrate Scanner

Free Scanner

Memory Compaction in Linux

Step 3: Migrate Pages



□ Free 4KB page

🟩 Allocated 4KB page

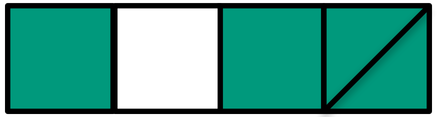
Memory Compaction in Linux

- **2MB** block granularity
  - Migrate scanner start from **start** of address space
  - Free scanner starts from **end** of address space
- Pages in migrate and free lists are ***invisible***

- Compaction **threshold**
  - Next compaction run, scanners continue from last stop
  - Position reset when scanners meet

Memory Compaction Pathologies in Linux

1. Unmovable pages lead to **wasted cycles**



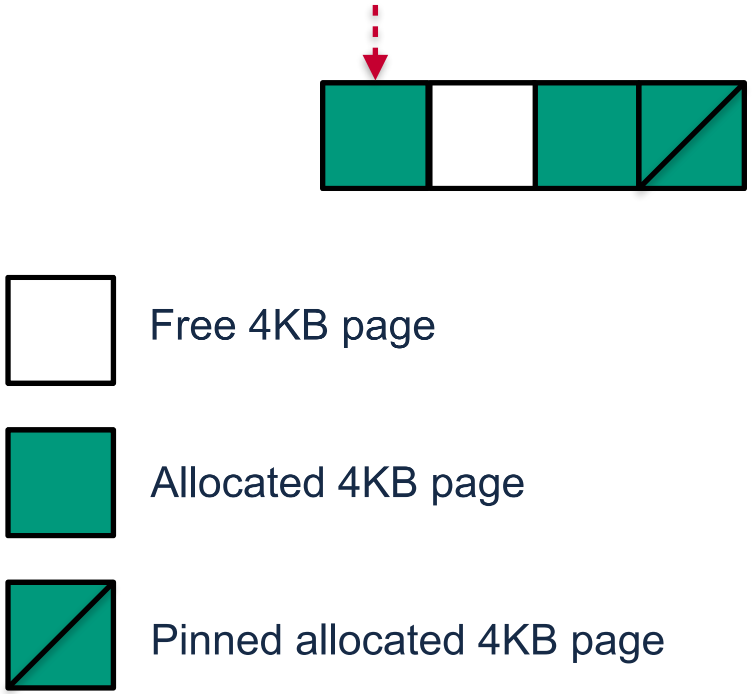
□ Free 4KB page

🟩 Allocated 4KB page

▨ Pinned allocated 4KB page

Memory Compaction Pathologies in Linux

1. Unmovable pages lead to **wasted cycles**



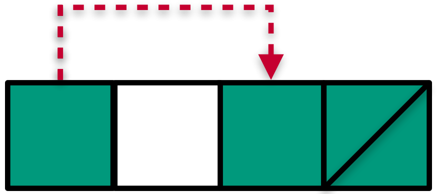
□ Free 4KB page

🟩 Allocated 4KB page

▨ Pinned allocated 4KB page

Memory Compaction Pathologies in Linux

1. Unmovable pages lead to **wasted cycles**



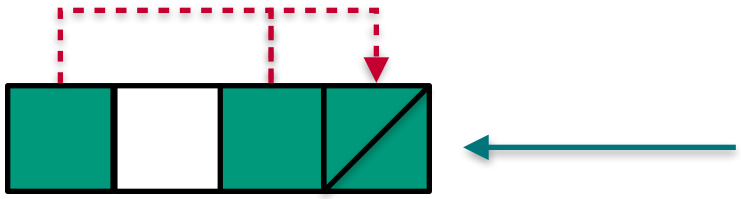
□ Free 4KB page

🟩 Allocated 4KB page

▨ Pinned allocated 4KB page

Memory Compaction Pathologies in Linux

1. Unmovable pages lead to **wasted cycles**



This will revert all work

□ Free 4KB page

🟩 Allocated 4KB page

▨ Pinned allocated 4KB page

Memory Compaction Pathologies in Linux

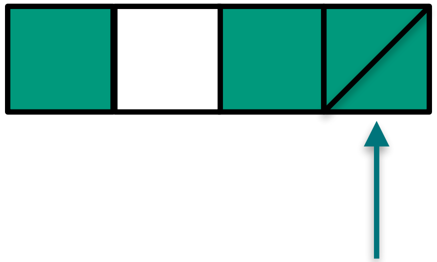
2. Free scanner **skips:**

- ≥ 2MB blocks (2MB and 4MB blocks are the biggest blocks)
- Small blocks that cannot accommodate all pages from migration list

Forces **preemptive scanners meet** and **creates mixed space**

Memory Compaction Pathologies in Linux

3. **Unfair** page skip



Page **will be excluded** from future compaction runs

PMEM Page Migration

Crucial operation in hybrid memories

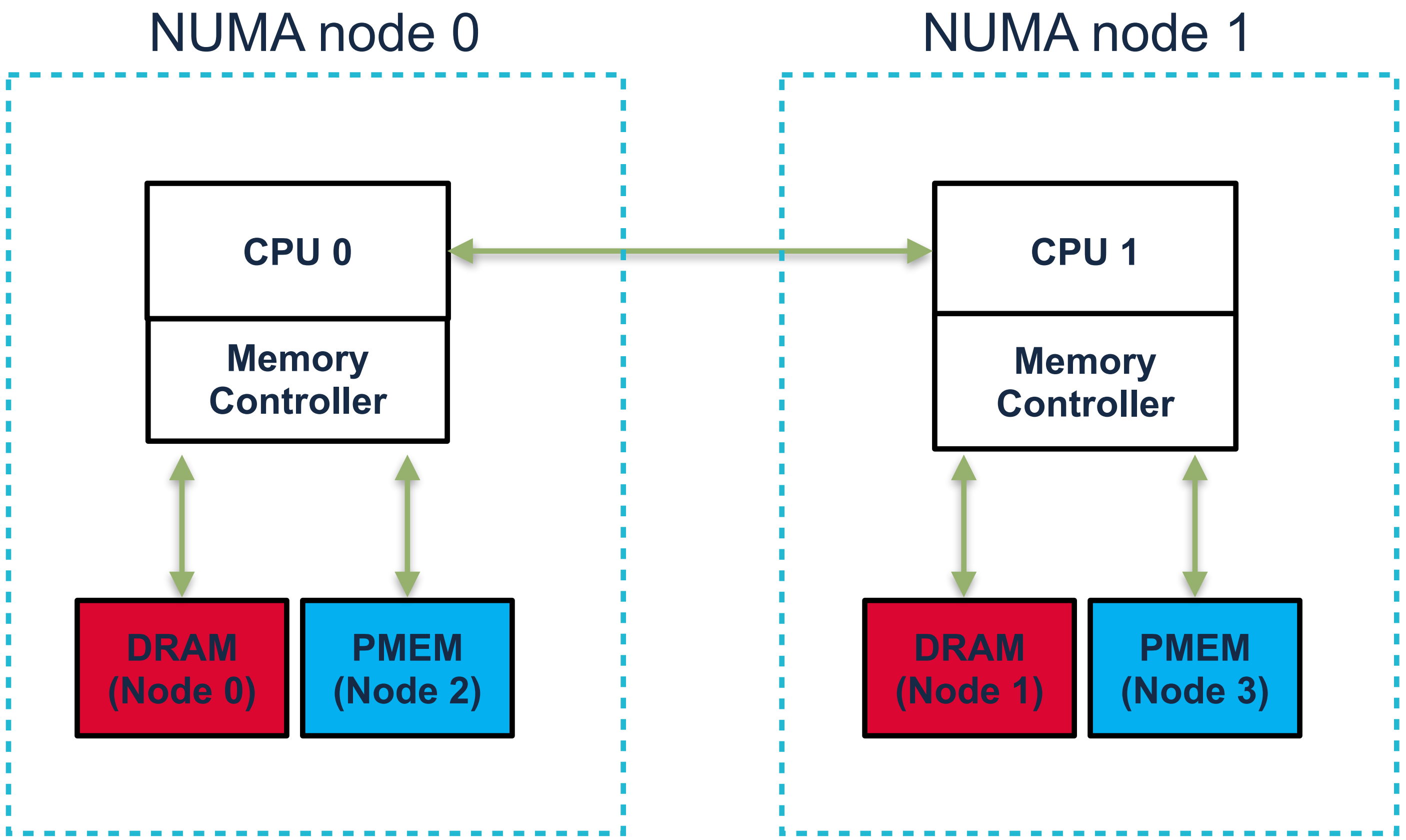
Process:

- Allocate a new page in the target memory
- Copy contents
- Free the old memory/TLB/metadata update

Main Linux allocator

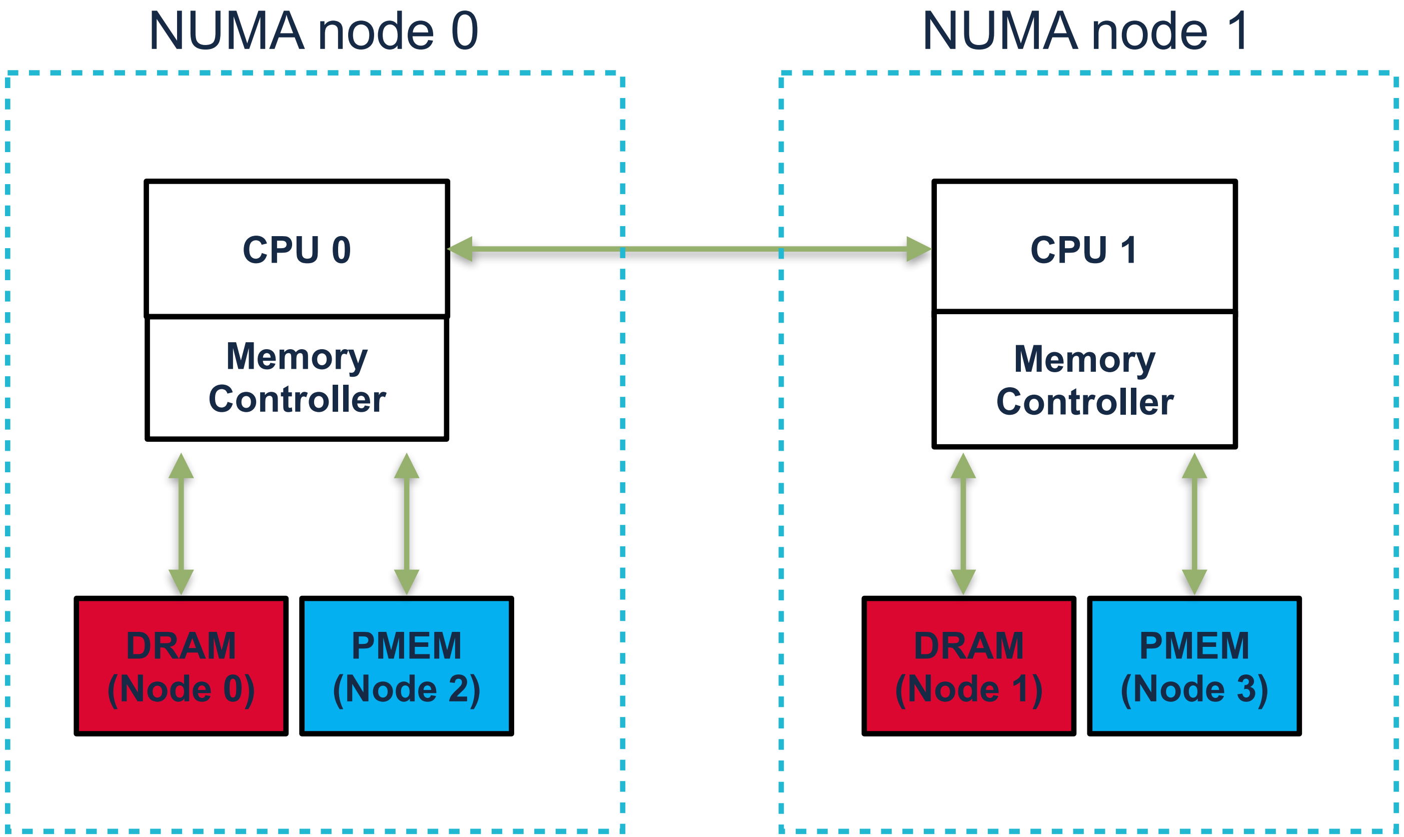
- Centralized, performance critical component
- Free page is closer to start of address space, will be migrated again from the compactor

Non-Uniform Memory Access (NUMA)



STABLE

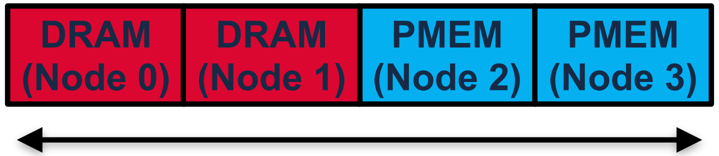
Non-Uniform Memory Access (NUMA)



1. Accessing a local memory is faster than accessing a remote.
2. Accessing a remote DRAM is faster than accessing a remote PMEM.

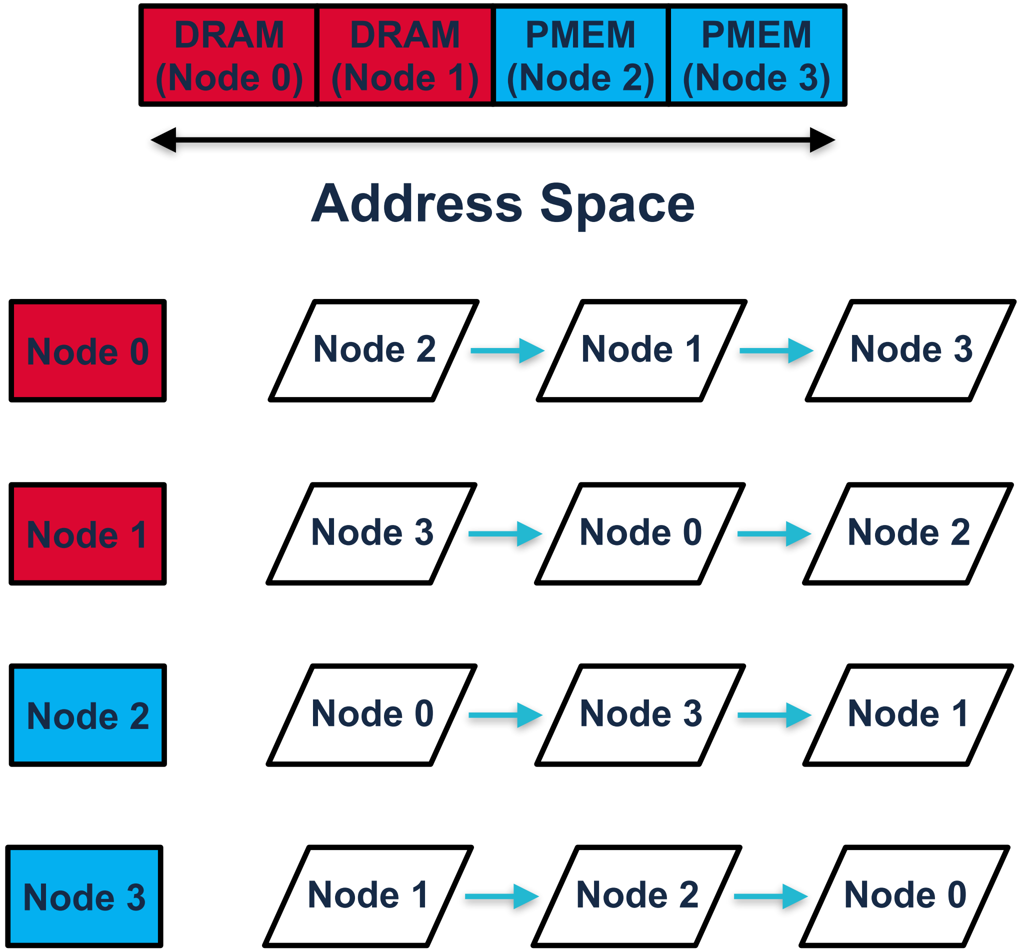
STABLE

Non-Uniform Memory Access (NUMA)



Address Space

Non-Uniform Memory Access (NUMA)



DRAM (Node 0) | DRAM (Node 1) | PMEM (Node 2) | PMEM (Node 3)

Address Space

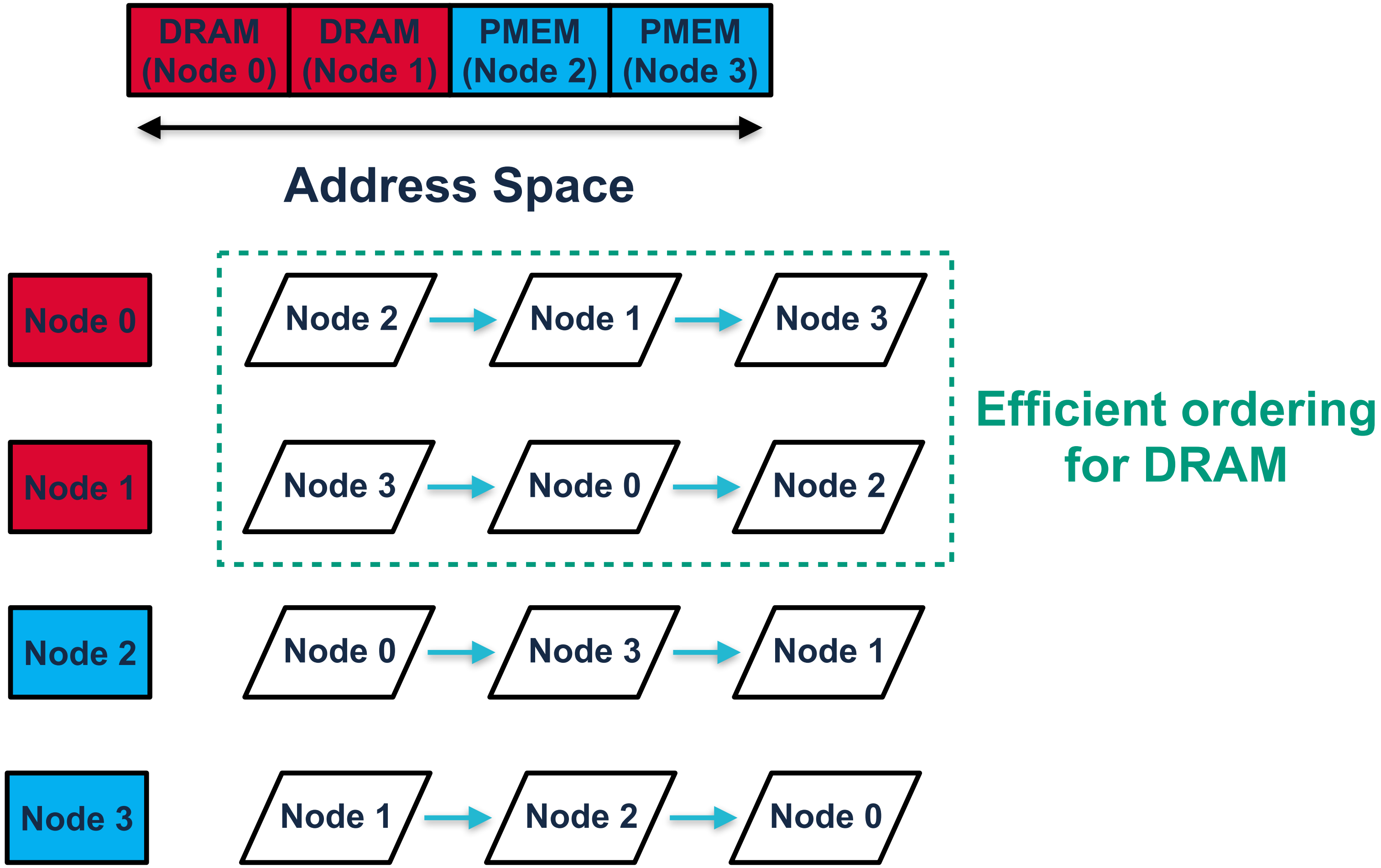
Node 0: Node 2 → Node 1 → Node 3

Node 1: Node 3 → Node 0 → Node 2

Node 2: Node 0 → Node 3 → Node 1

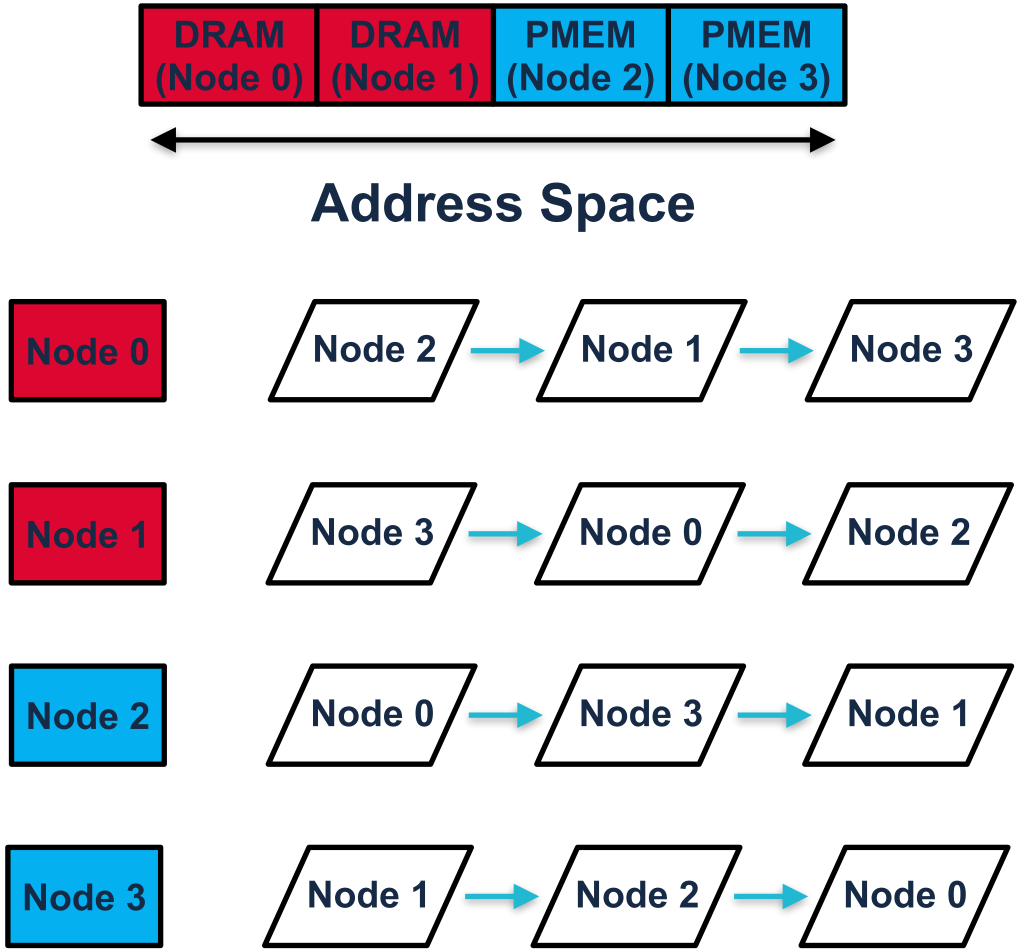
Node 3: Node 1 → Node 2 → Node 0

Non-Uniform Memory Access (NUMA) ordering



STA
BLE

Non-Uniform Memory Access (NUMA)



DRAM (Node 0) | DRAM (Node 1) | PMEM (Node 2) | PMEM (Node 3)

Address Space

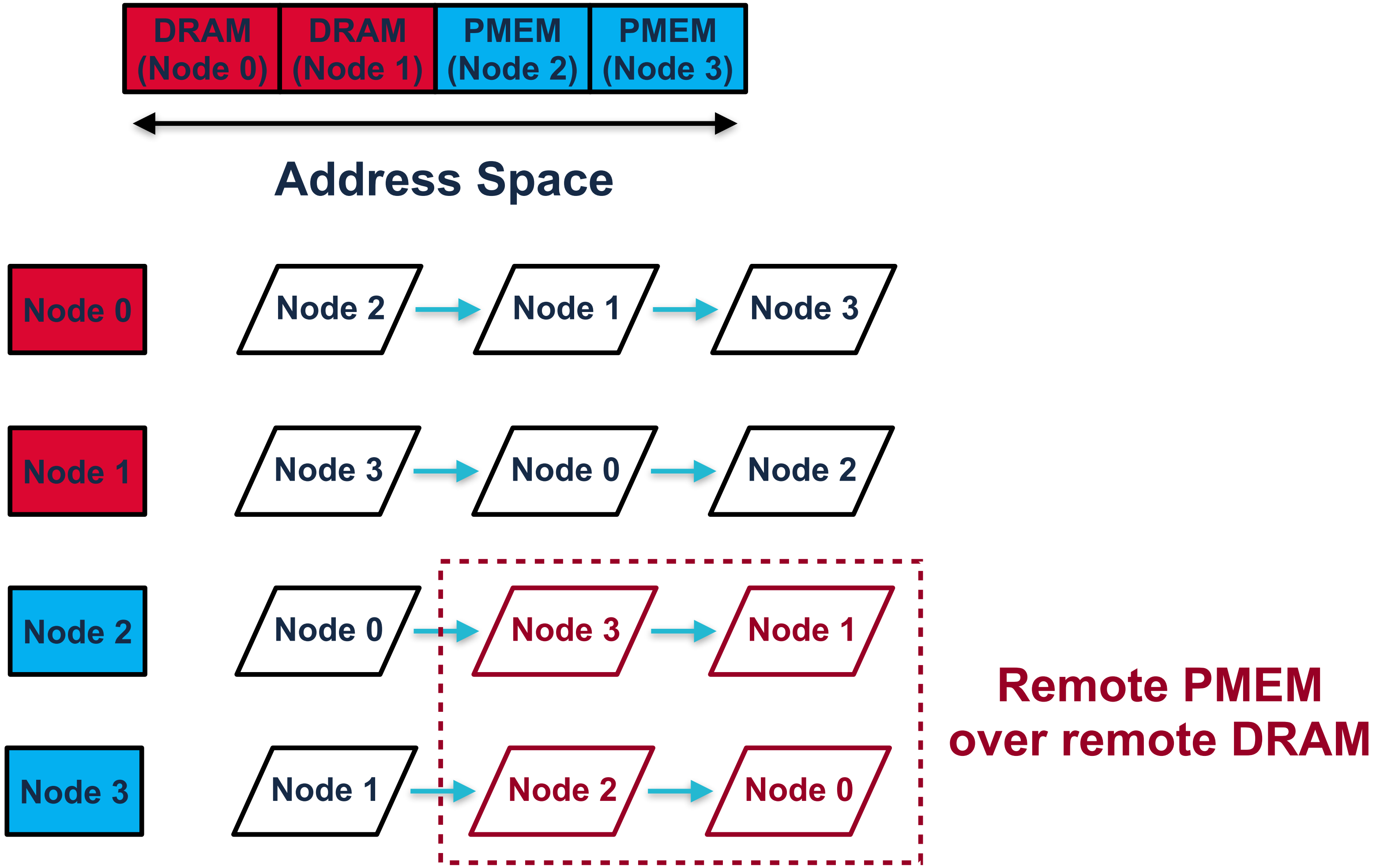
Node 0: Node 2 → Node 1 → Node 3

Node 1: Node 3 → Node 0 → Node 2

Node 2: Node 0 → Node 3 → Node 1

Node 3: Node 1 → Node 2 → Node 0

NUMA issue with PMEM



STΔ BLE

Overview

- Background & Motivation
- PMShifter
- Evaluation
- Conclusion

PMShifter compaction

Same 3-step logic with Linux for compatibility

PMShifter compaction

Same 3-step logic with Linux for compatibility

Linux	PMShifter
Block-to-block logic	**In bulk** operation
2MB block	**4MB** block
Wasted cycles due to unmovable pages	**Skip unmovable pages** in O(1)
Free scanner **skips ≥** 2MB and small blocks	Free scanner **uses** all blocks
Unfair page skip	Page state **not maintained**

PMShifter compactors

Different goals in DRAM and PMEM

- In DRAM maximize biggest free blocks
- In PMEM keep the start of the address space clean

The intuition behind this relates to shifting

Combined DRAM compaction and PMEM migration

During compaction, fill half of the migrate list with the hottest pages from PMEM

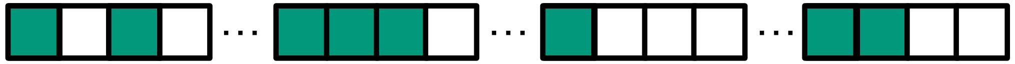
- Increasing the total amount of migrated pages, increases throughput[1]
- Avoid pressure in Linux allocator
- Accurate page placement

[1] Yan et al. “Nimble Page Management for Tiered Memory Systems” ASPLOS 2019

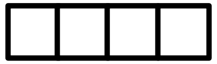
STA
BLE

Combined DRAM compaction and PMEM migration

Step 1: DRAM pages from topN **less loaded** 4MB blocks

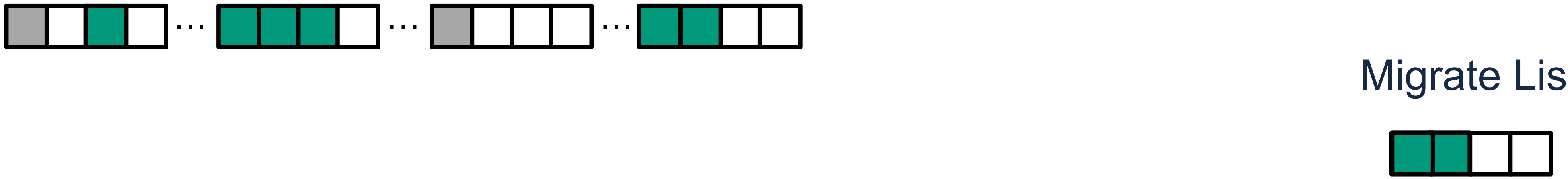


Migrate List

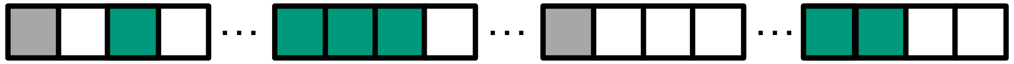


Combined DRAM compaction and PMEM migration

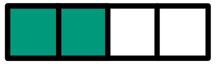
Step 1: DRAM pages from **less** topN 4MB blocks



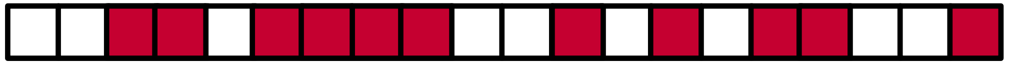
Combined DRAM compaction and PMEM migration



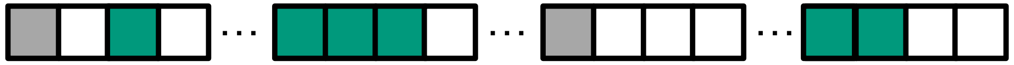
Migrate List



Step 2: Hot pages from PMEM

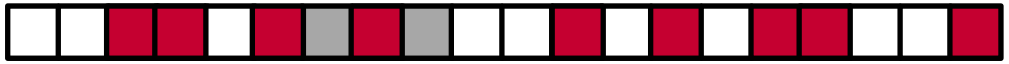


Combined DRAM compaction and PMEM migration



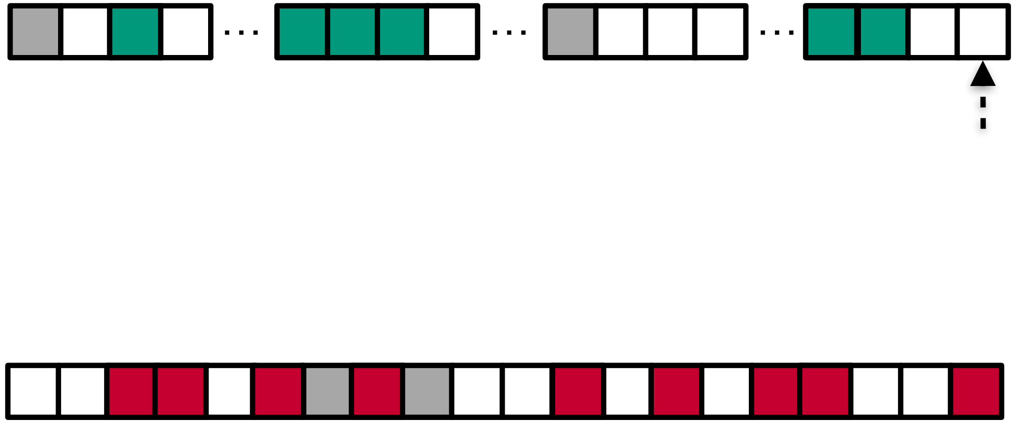
Migrate List

Step 2: Hot pages from PMEM



Combined DRAM compaction and PMEM migration

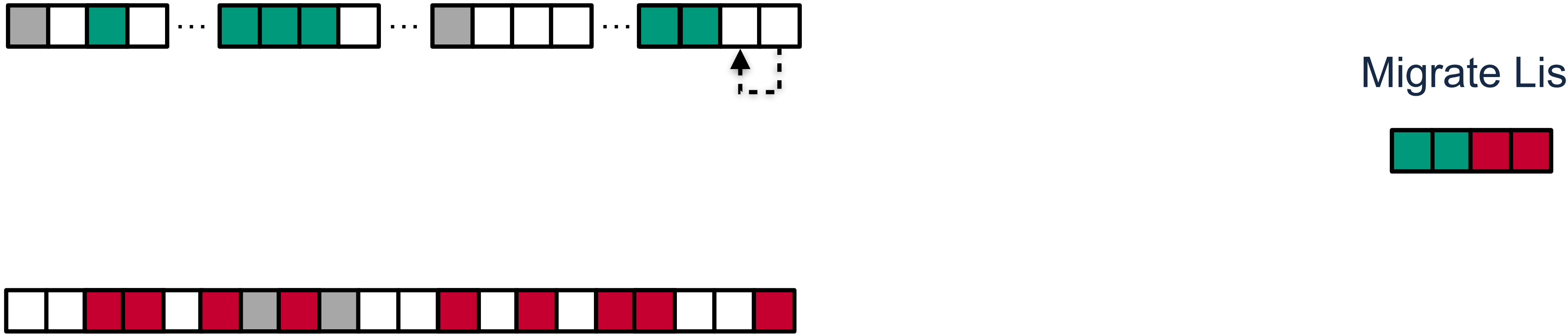
Step 3: Scan for free pages



Migrate List

Combined DRAM compaction and PMEM migration

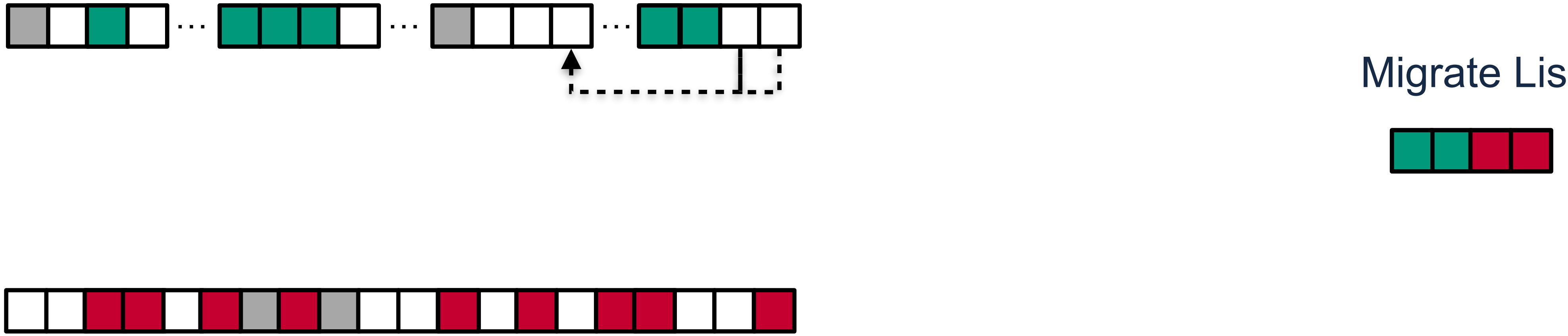
Step 3: Scan for free pages



Migrate List

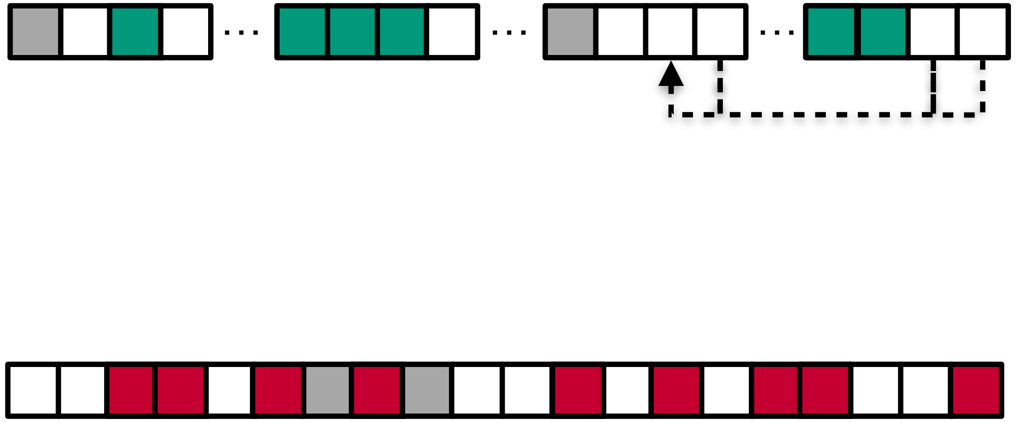
Combined DRAM compaction and PMEM migration

Step 3: Scan for free pages



Combined DRAM compaction and PMEM migration

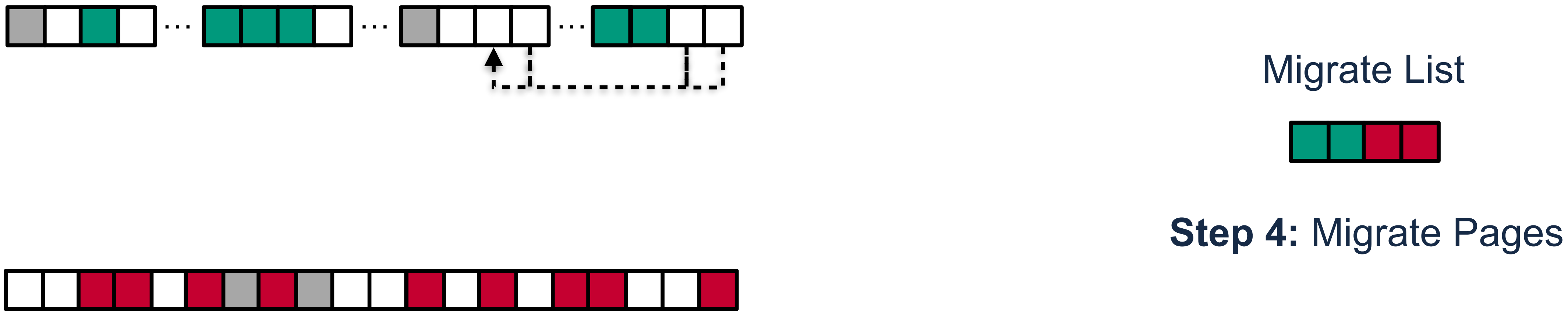
Step 3: Scan for free pages



Migrate List

STA
BLE

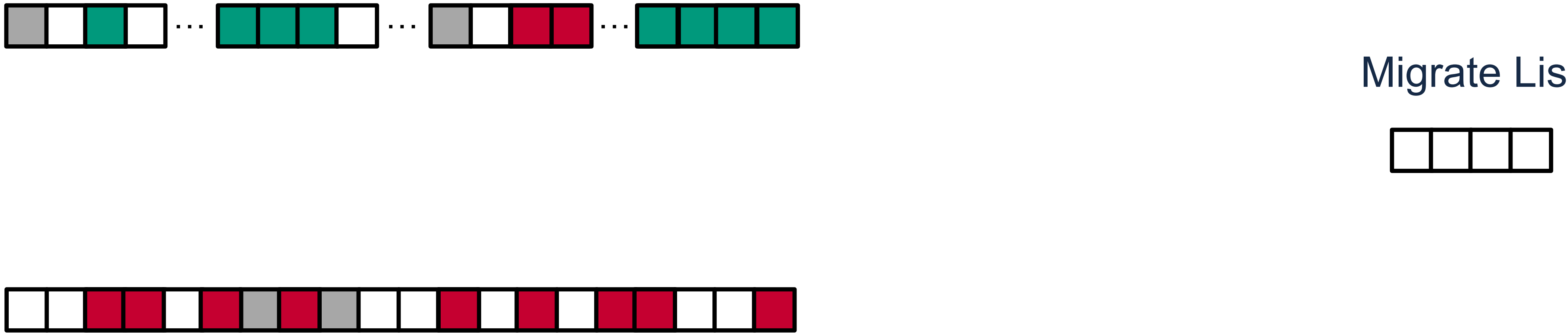
Combined DRAM compaction and PMEM migration



Migrate List

Step 4: Migrate Pages

Combined DRAM compaction and PMEM migration



Persistent Memory Shifting

PMEM shifting is a costly operation that should occur infrequently

Goal: Accurately predict if we need to acquire/release memory

↓

Use an adjusted version of the Exponential Moving Average

STA
BLE

Persistent Memory Shifting

Memory pressure at time t total free space

$$\boxed{MP_t} = \boxed{a} * \boxed{free\_space} + (1 - a) * MP_{t-1}$$

smoothing factor

If MPt > threshold, **increase free memory by 5x**

Persistent Memory Shifting

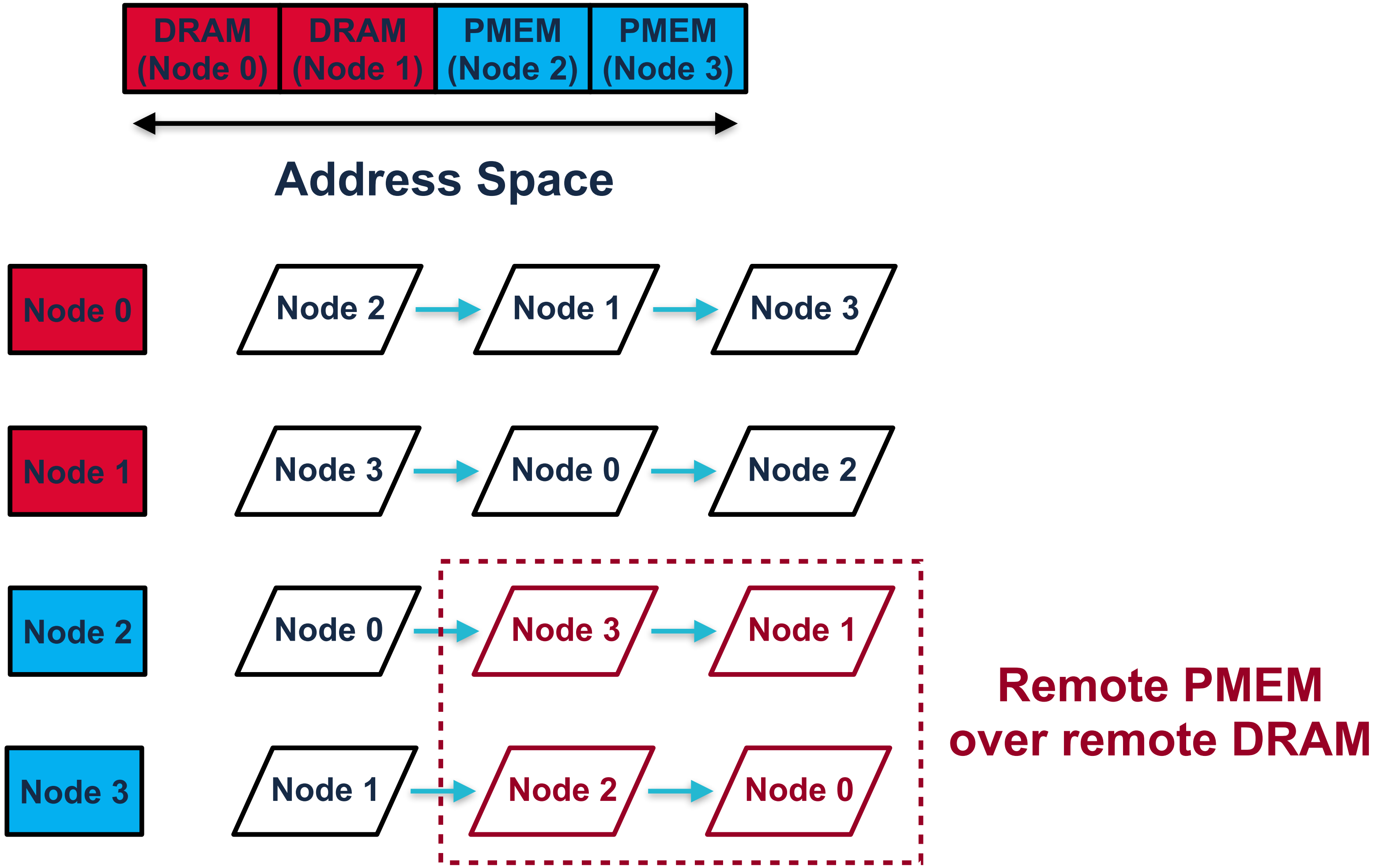
Which pages to shift, **to reduce the cost**?

Reminder: The PMEM compactor keeps the **start of address space clean**

↓

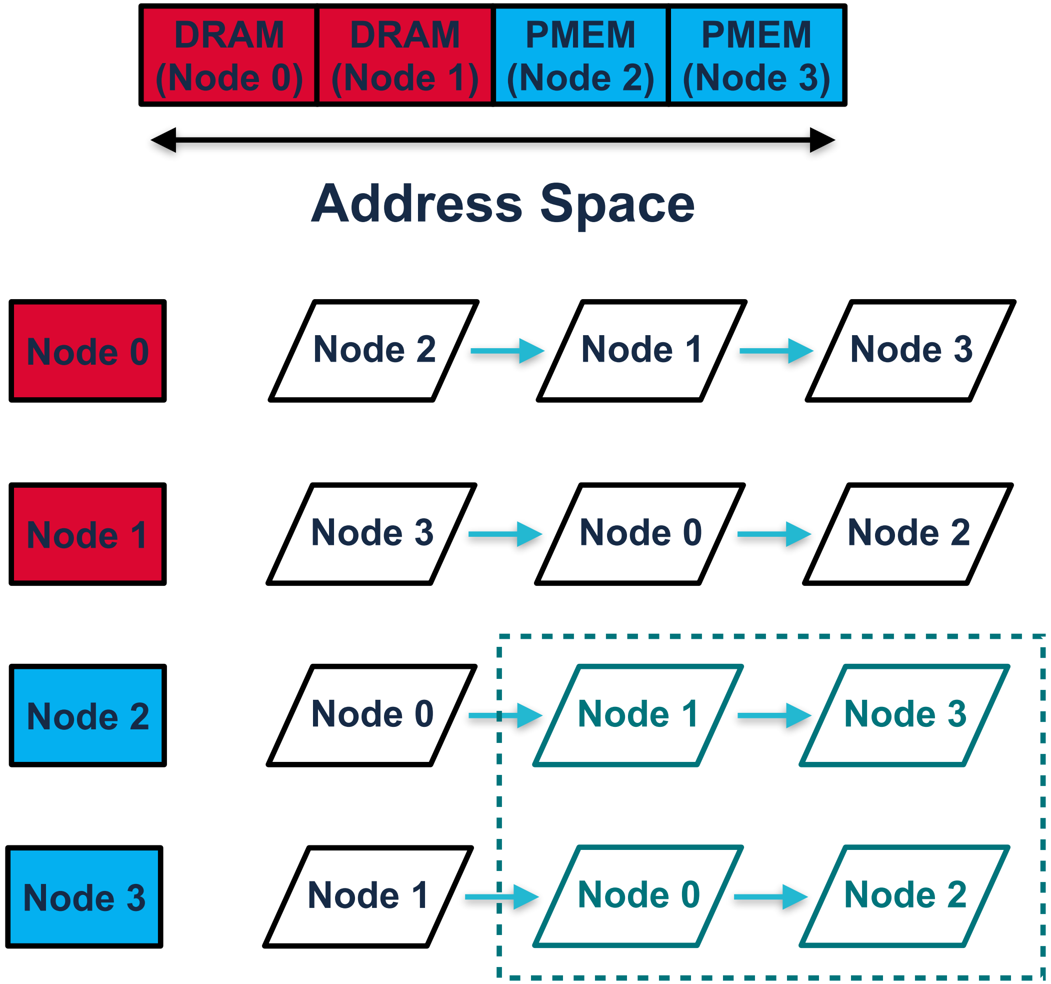
Shift the start of address space

Proposed fix for NUMA



STΔ
BLE

Proposed fix for NUMA



DRAM (Node 0) | DRAM (Node 1) | PMEM (Node 2) | PMEM (Node 3)

Address Space

Node 0: Node 2 → Node 1 → Node 3

Node 1: Node 3 → Node 0 → Node 2

Node 2: Node 0 → Node 1 → Node 3

Node 3: Node 1 → Node 0 → Node 2

Remote DRAM over remote PMEM

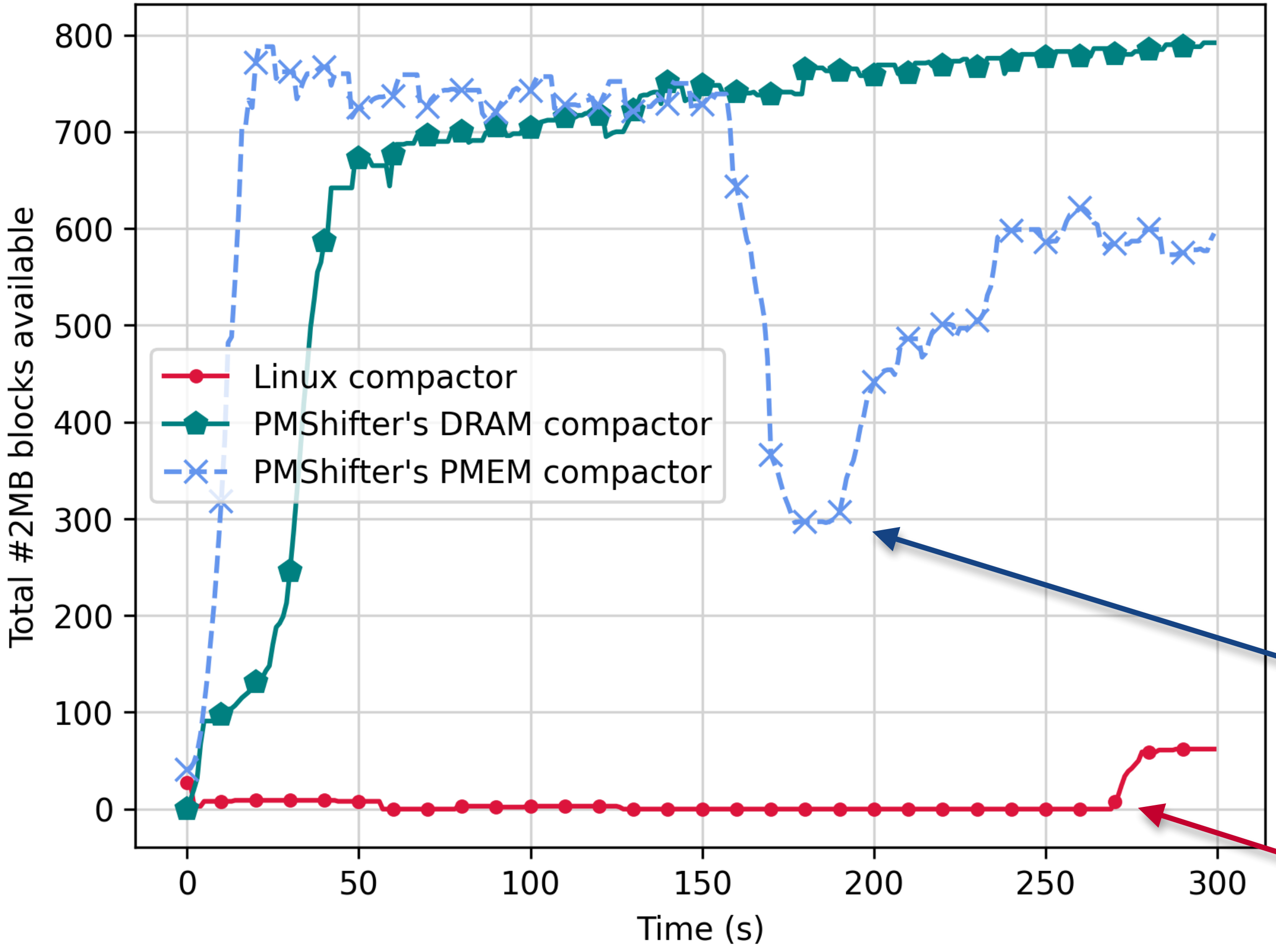
Overview

- Background & Motivation
- PMShifter
- Evaluation
- Conclusion

Evaluation

- Implemented PMShifter in Linux v5.6.19
- Evaluated with
  - Microbenchmarks
  - Redis
  - Galois

Compaction performance



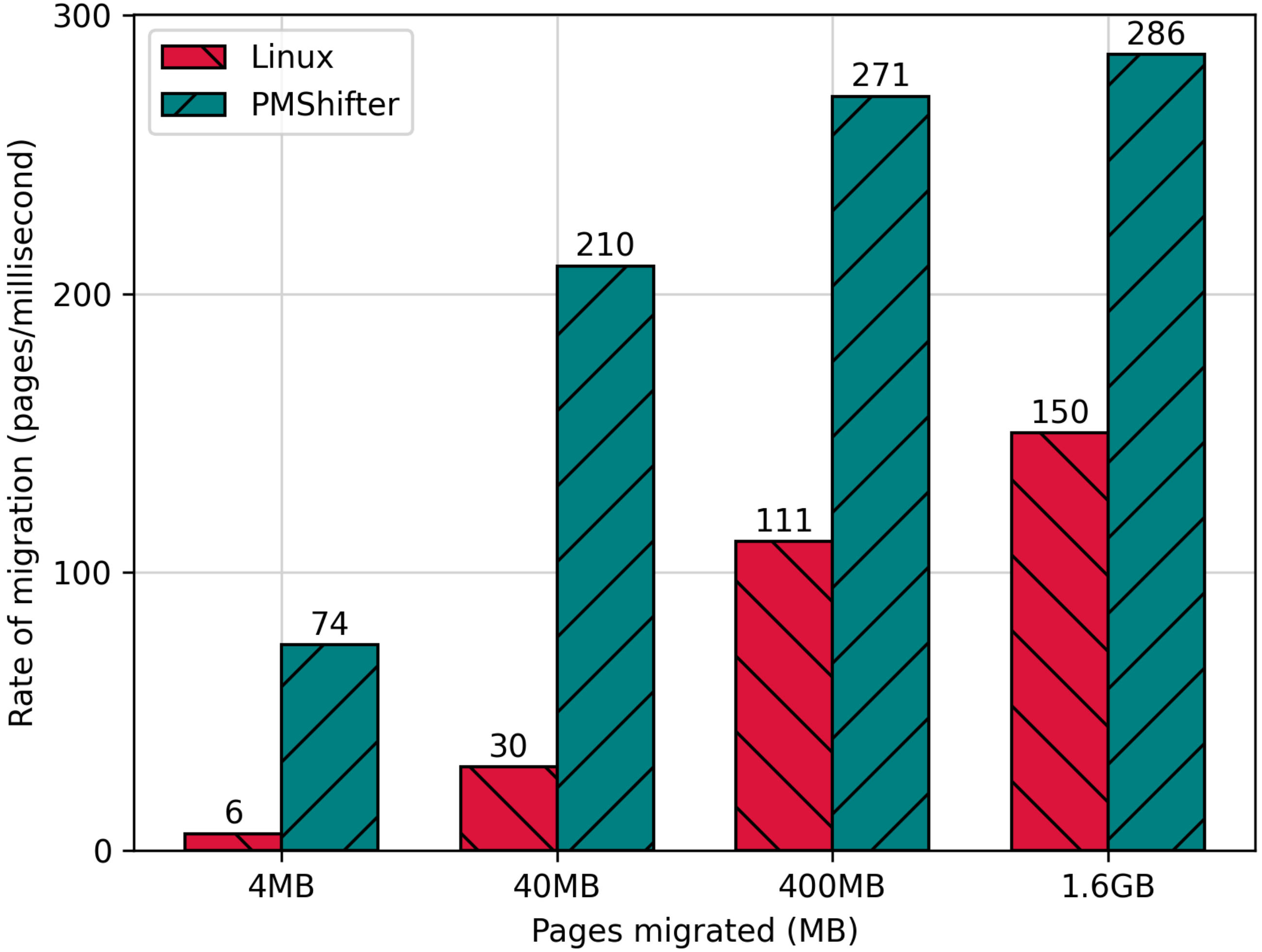
PMShifter achieves up to **12.77×** more 2MB clean contiguous blocks.

Mixed space effect

4.5 minutes to start recovering

STΔ BLE

Combined DRAM compaction & PMEM migration

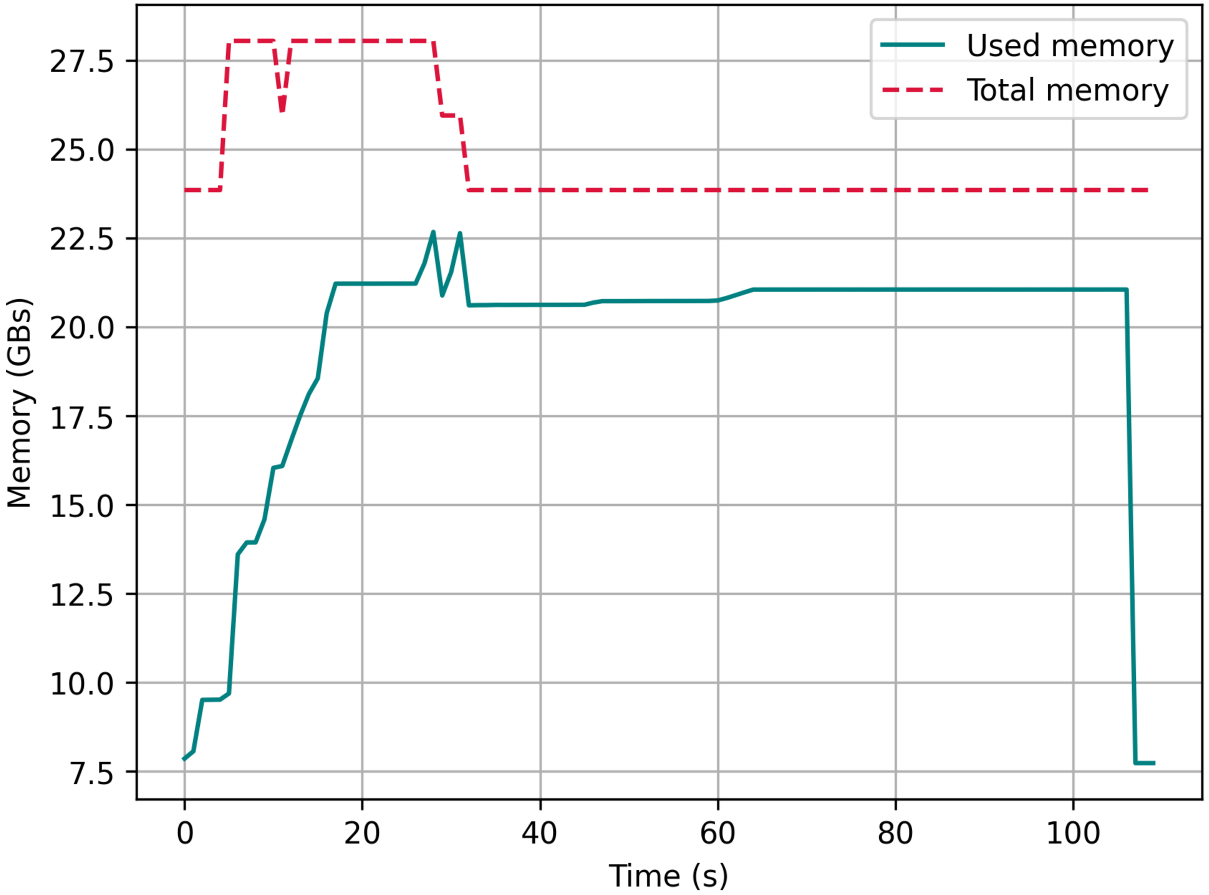


Average speedup: **5.88×**

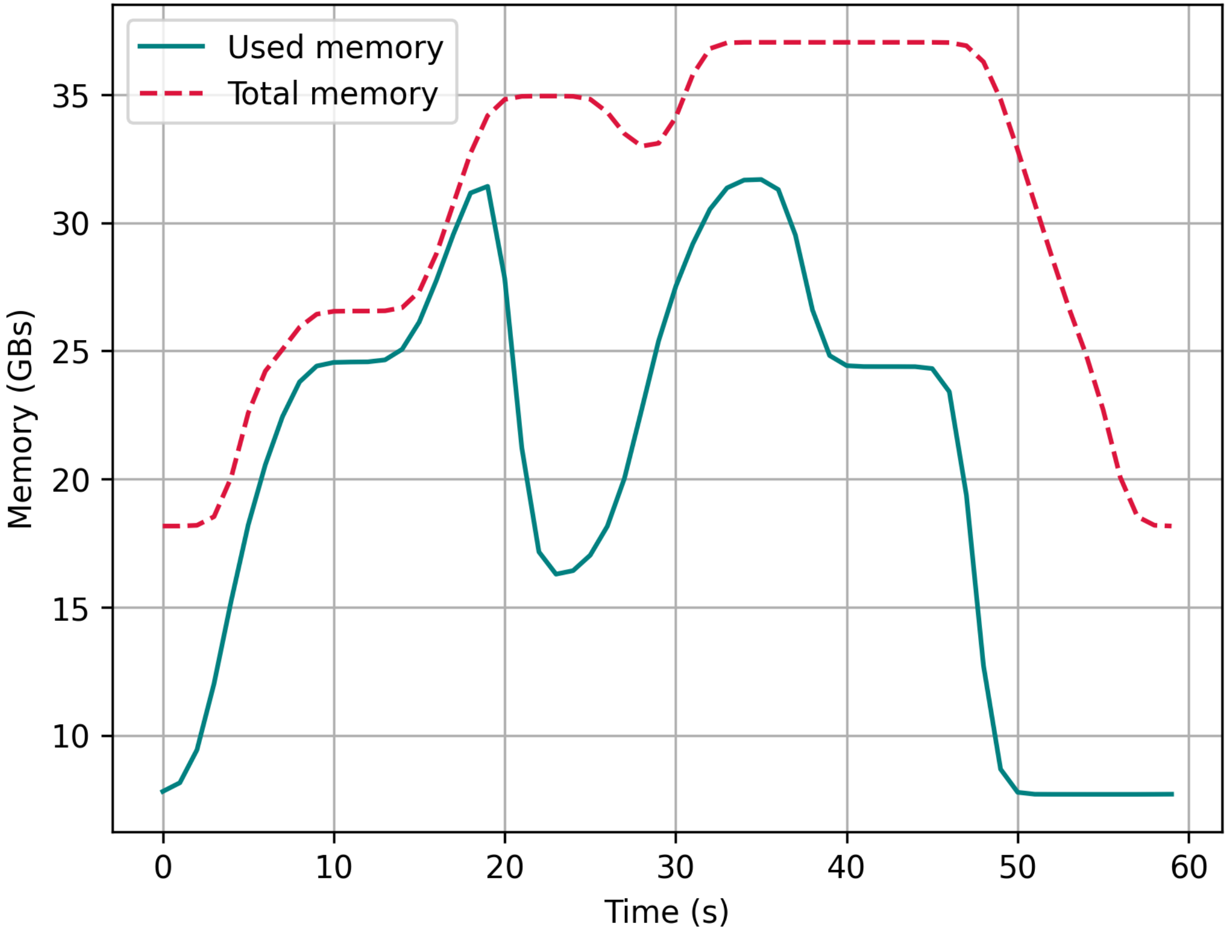
Failed migrations:
- Less than **0.0083%** for PMShifter
- Between **41.4%** and **49.9%** for Linux

STΔ
BLE

PMShifter elasticity



PMShifter elasticity



PMShifter is **elastic** and **proactive**

Overview

- Background & Motivation
- PMShifter
- Evaluation
- Conclusion

STA
BLE

Conclusion

Is **dynamic and elastic**

Proposes **associated memory management fixes**

- **Accelerates** page migration
- Significantly **improves** fragmentation
- **Fixes** PMEM issues in NUMA