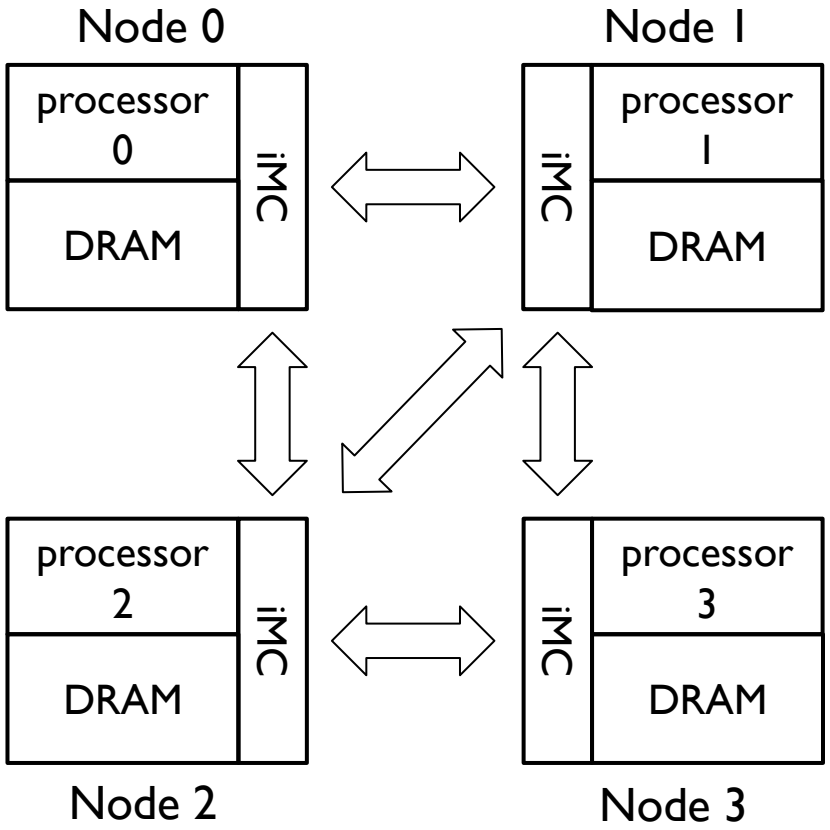
Nomad: Non-Exclusive Memory Tiering via Transactional Page Migration

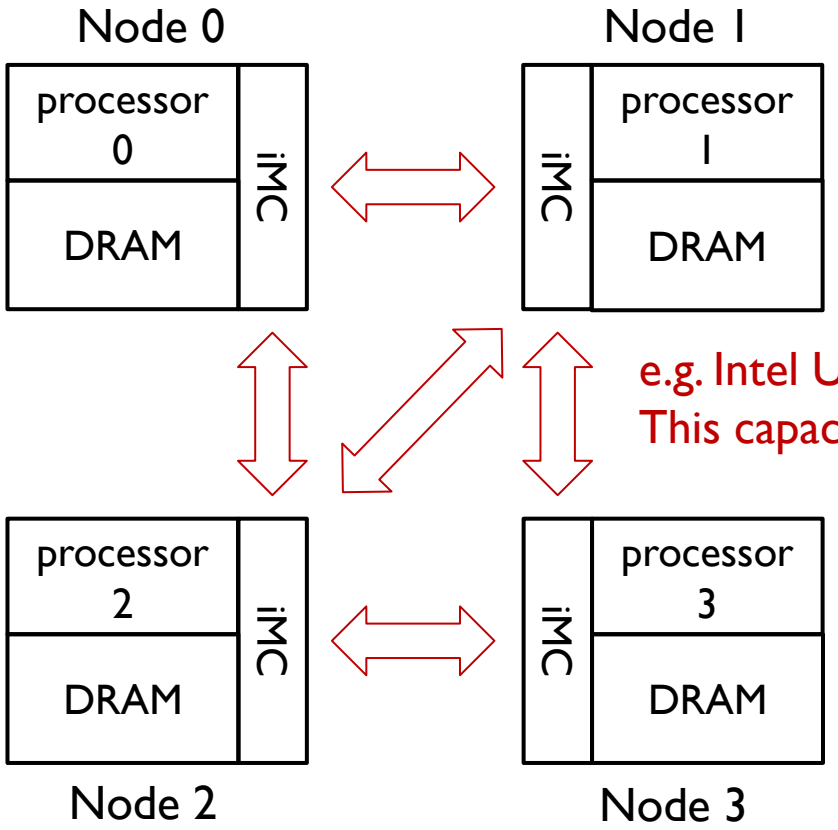
Speaker: Jiahao Li

Authors: Lingfeng Xiang, Zhen Lin, Weishu Deng, Hui Lu, Jia Rao, Yifan Yuan and Ren Wang

NUMA Systems



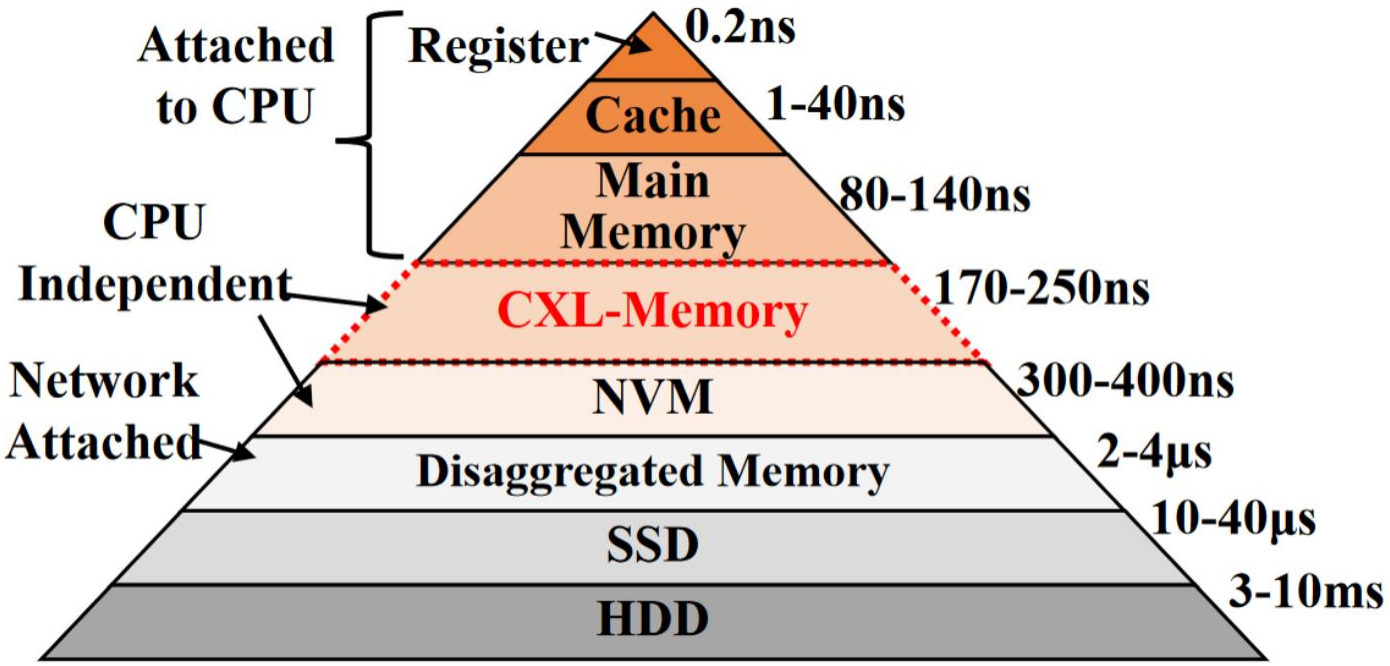
NUMA Systems



e.g. Intel UPI supports ≈ 22.4 GB/s for each direction.
This capacity is small compared to DRAM's bandwidth.

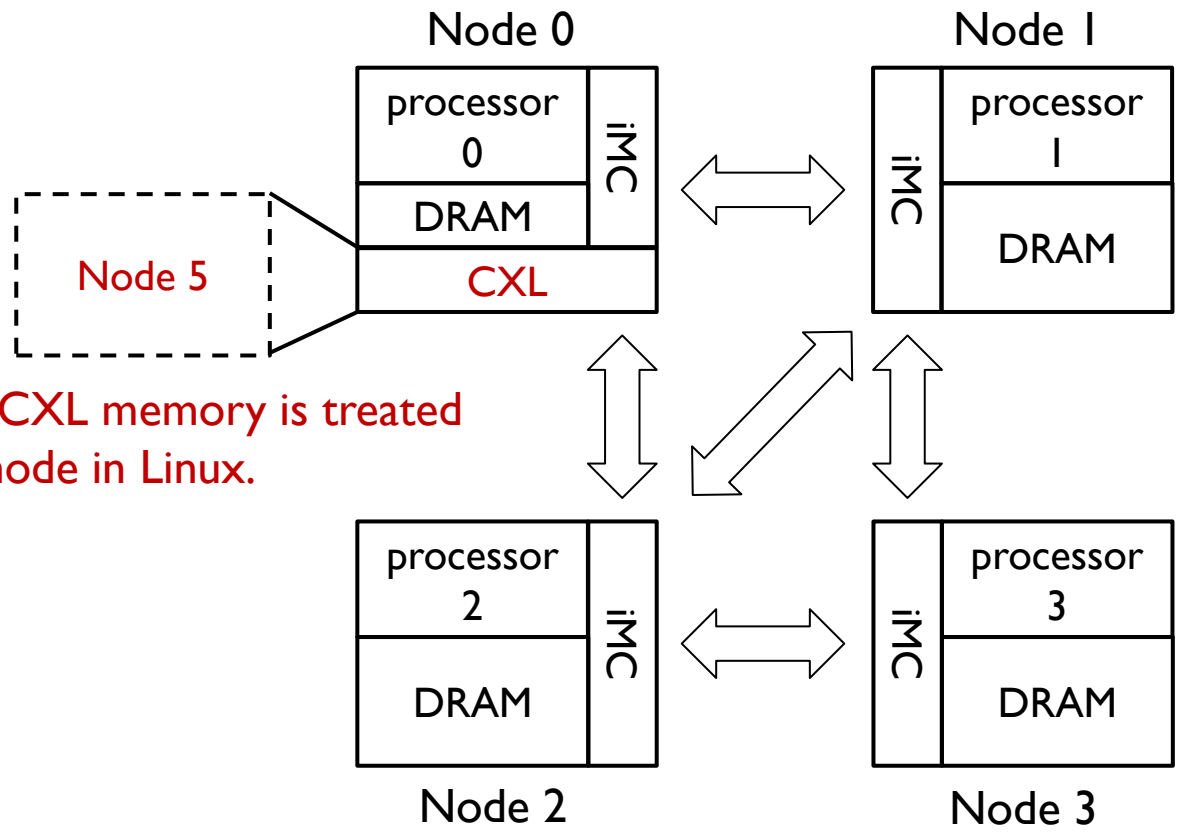
1. Nodes are connected through inter-connections.
2. Accessing remote memory is more costly than accessing local memory.

New Memory Hierarchy



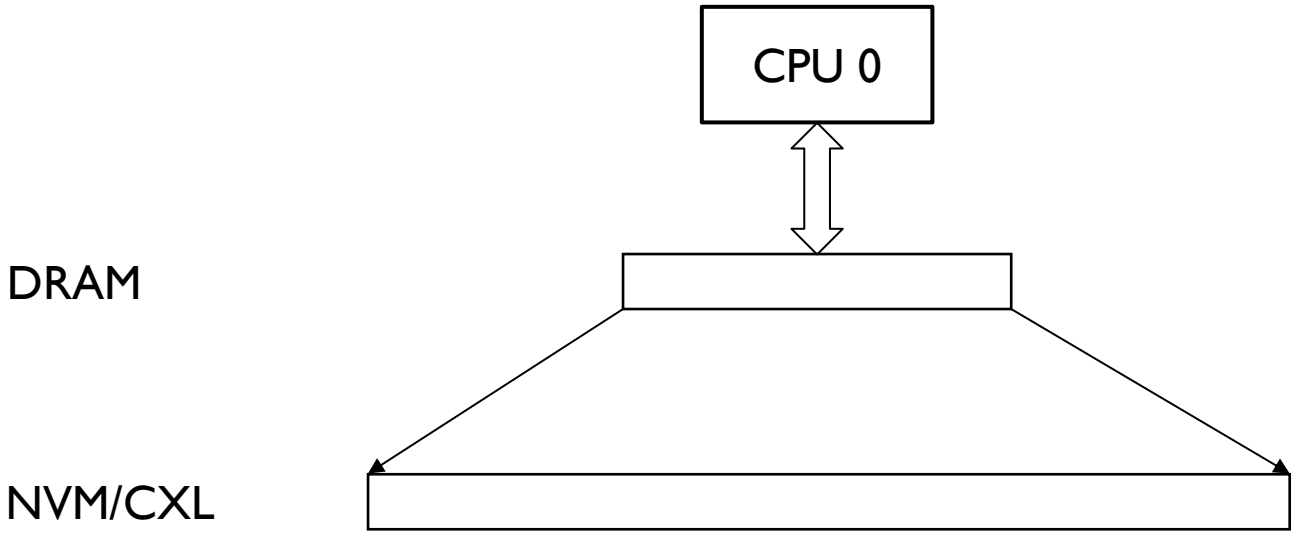
CXL enables more versatile memory hierarchy

NUMA Systems



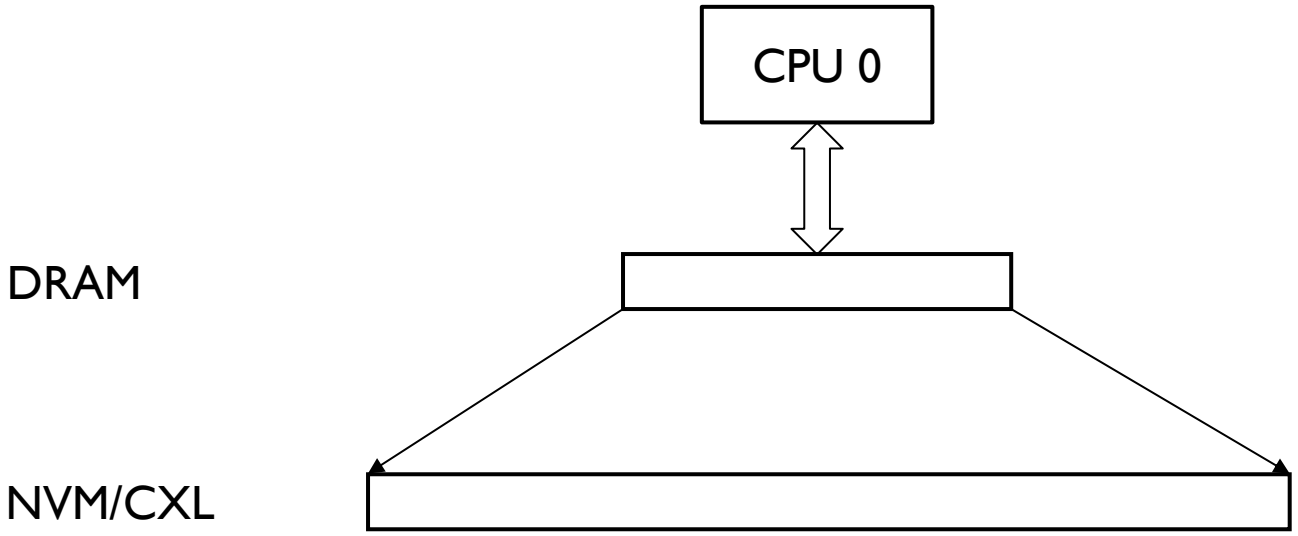
Different from DRAM, CXL memory is treated as a CPU-less NUMA node in Linux.

Memory Tiering – Hardware Approach



DRAM becomes "L4" cache and NVM/CXL becomes main memory.

Memory Tiering – Hardware Approach



DRAM becomes "L4" cache and NVM/CXL becomes main memory.

Pros:

1. Transparent.
2. Small granularity (cacheline).

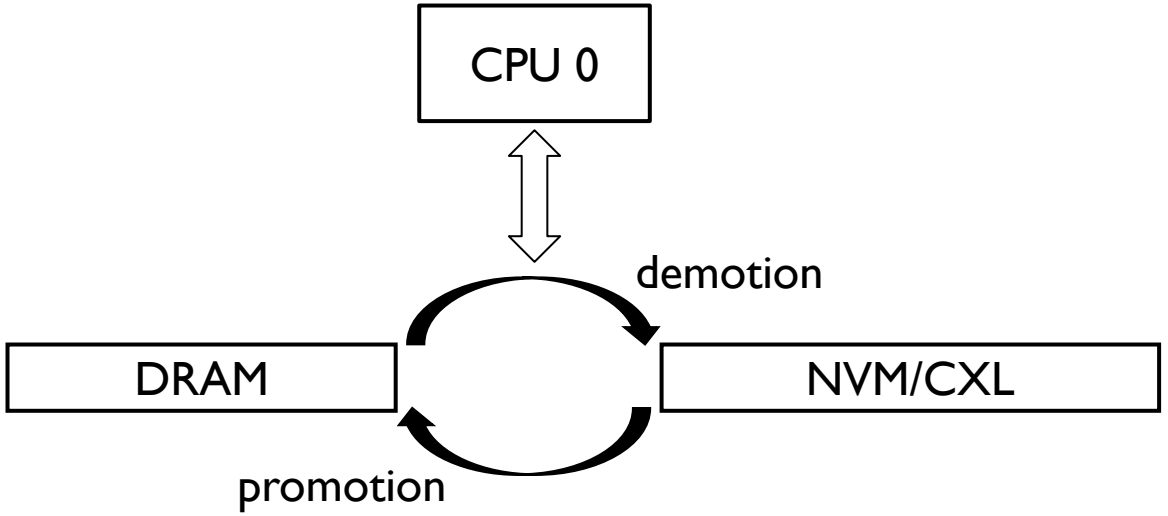
Cons:

1. Hard-coded
2. Prone to cache conflicts. [Johnny Cache OSDI'23]

Memory Tiering – Software Approach

Page migration is used in exclusive memory tiering to speedup memory access:

1. Migrate hot remote pages to application's local node.
2. Migrate cold pages to slower tier to make space for hot pages.



Memory Tiering – Challenges

1. Which page to promote/demote?
2. Where to place the migrated pages?
3. How to migrate pages efficiently?

Memory Tiering – Hotness Tracking

These methods are used to tracking page accesses:

1. Page faults (Linux NUMA balancing)
   - Periodically make part of application's address space (256MB) inaccessible, page's hotness is indicated by whether page faults are triggered.
   - Faulted pages are considered hot and will be migrated to fast tier.

Memory Tiering – Hotness Tracking

These methods are used to tracking page accesses:

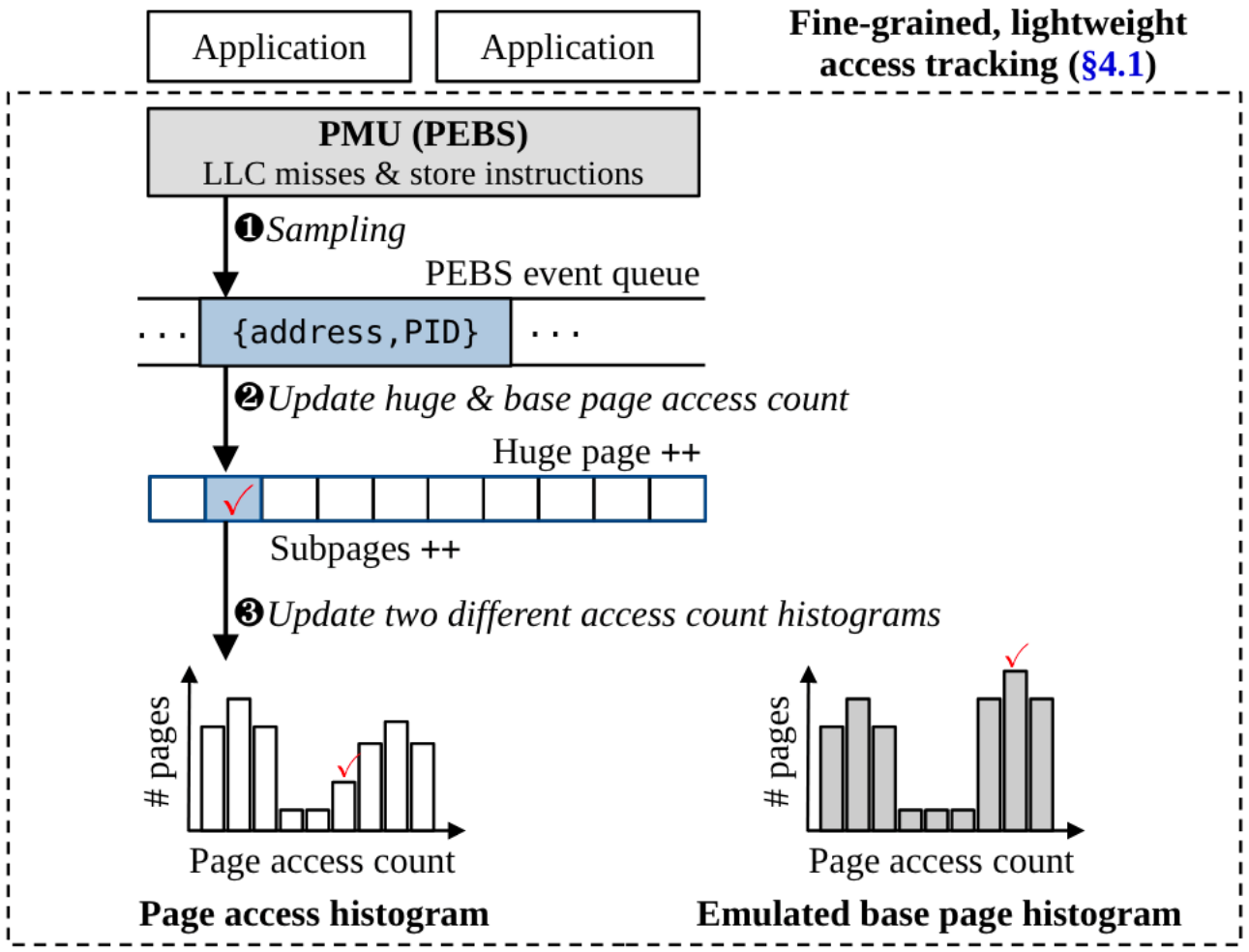
1. Page faults (Linux NUMA balancing)
2. Page Table Scanning (Linux kswapd)
   - Kernel maintains active and inactive LRU lists for each NUMA node.
   - Periodically scan all PTEs' access bits. If a PTE's access bit is set, the corresponding page will be moved to the active LRU.
   - Pages in inactive list are considered cold. When the memory is nearly full, they will be swapped out to swap space.

Memory Tiering – Hotness Tracking

These methods are used to tracking page accesses:

1. Page faults (Linux NUMA balancing)
2. Page Table Scanning (Linux kswapd)
3. Processor Event-based Sampling (HeMem SOSP'21)
   - Intel processor support event-based sampling to sample page access through certain events (e.g. MEM_LOAD_L3_MISS_RETIRED.LOCAL_DRAM).
   - Pages with larger access count are considered hot.

Memory Tiering – Hotness Tracking



Memory Tiering – TPP

Transparent Page Placement (TPP) is SOTA tiering approach in Linux:

1. Lightweight reclamation
   - Migration instead of swapping

Memory Tiering – TPP

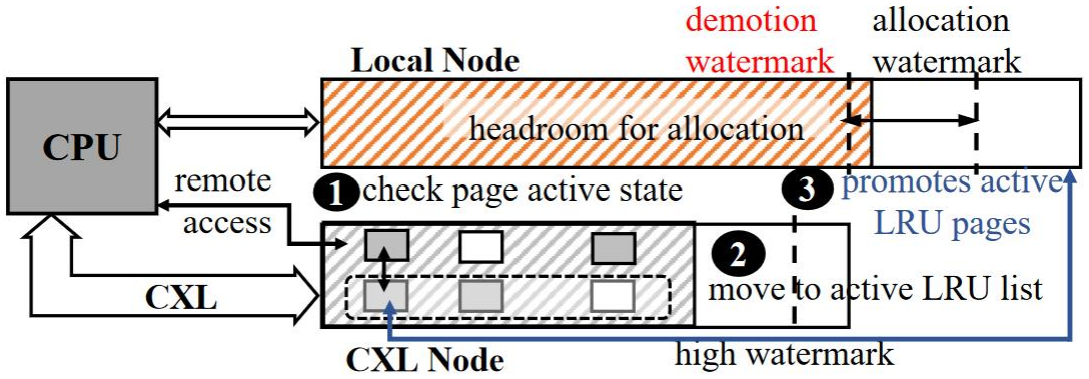
Transparent Page Placement (TPP) is SOTA tiering approach in Linux:

1. Lightweight reclamation
2. Decoupling allocation and reclamation

Memory Tiering – TPP

Transparent Page Placement (TPP) is SOTA tiering approach in Linux:

1. Lightweight reclamation
2. Decoupling allocation and reclamation
3. New promotion strategy



Memory Tiering – TPP

Transparent Page Placement (TPP) is SOTA tiering approach in Linux:

1. Lightweight reclamation
2. Decoupling allocation and reclamation
3. New promotion strategy
4. Page type-aware allocation

Page Migration is Complex

Page migration procedure:

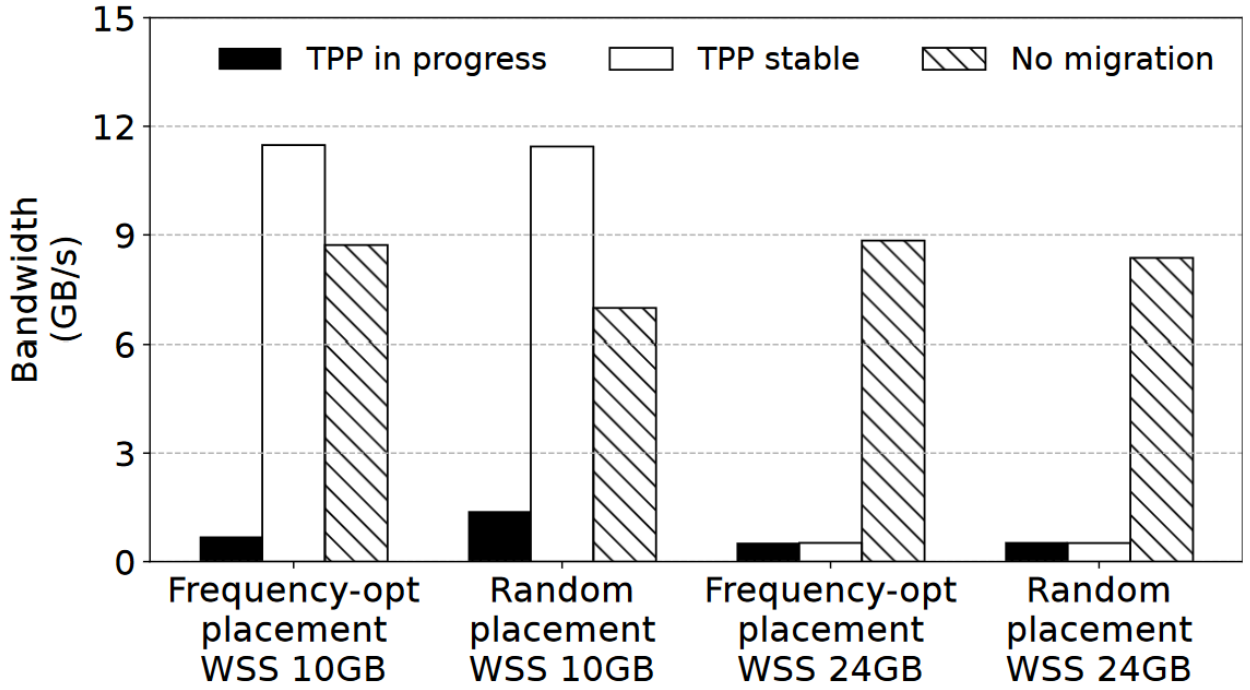
1. The system must trap to the kernel.
2. Lock the PTE of the migrating page and unmap it from page table.
3. TLB shootdown.
4. Copy the content of the page.
5. Remap the PTE.

Page Migration – TPP

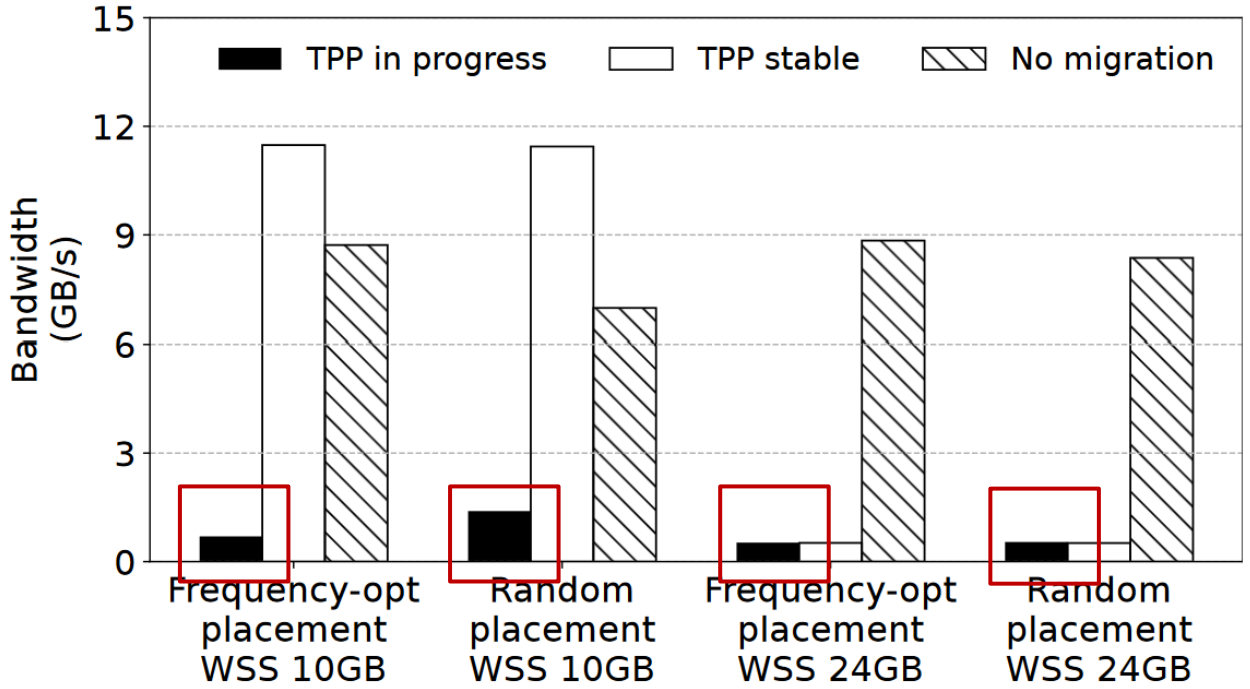
An experiment to demonstrate TPP problems:

- Workload: memory access with Zipfian distribution
- Fast tier size: 16GB
- Working set size (WSS): 10GB / 24GB
- Allocation policy: frequency-opt / random, pre-allocating 10GB in fast tier to simulate memory usage of existing applications

Page Migration – TPP



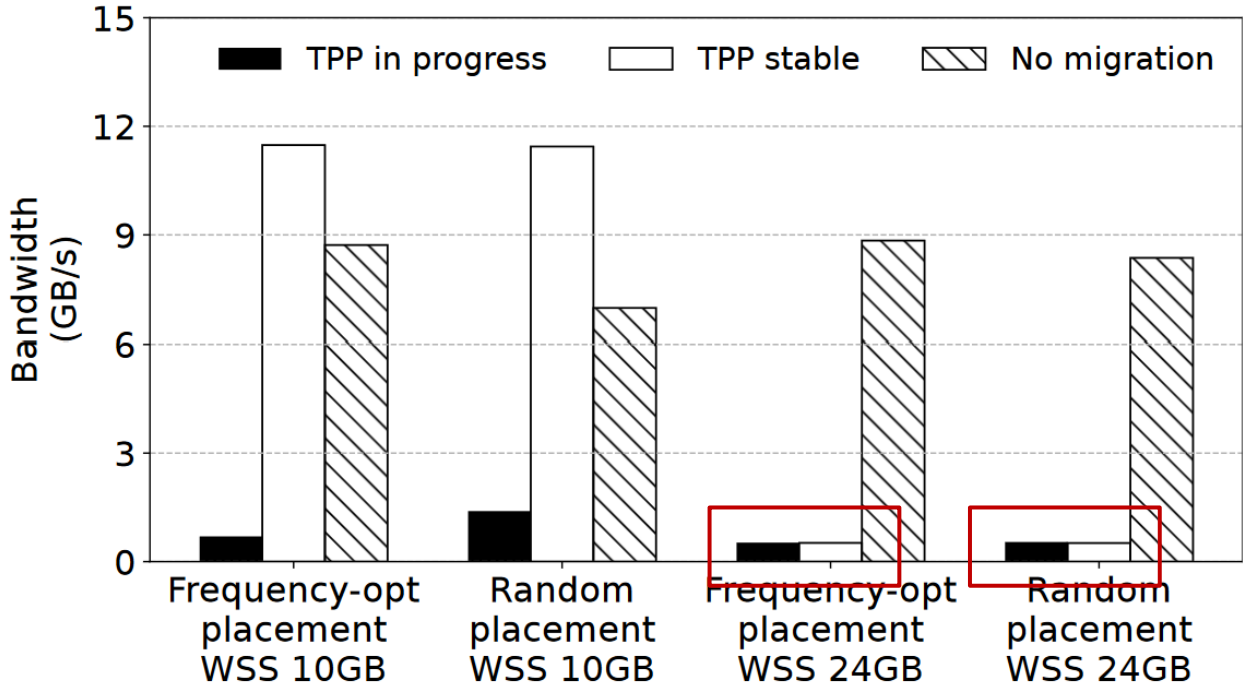
Page Migration – TPP



Two problems in TPP:

1. Migration limit the application's memory access bandwidth.

Page Migration – TPP

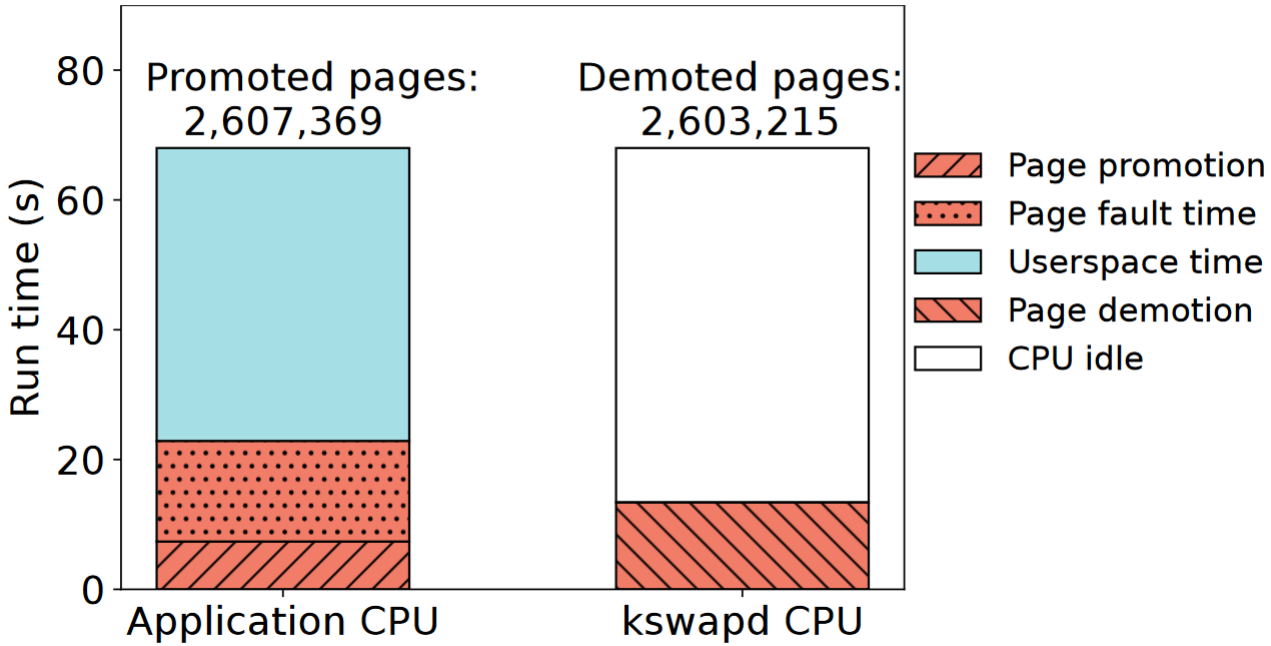


Two problems in TPP:

1. Migration limit the application's memory access bandwidth.
2. When WSS exceeds the capacity of the fast tier, TPP enters memory thrashing.

Page Migration – TPP

In TPP, promotion is synchronous



Synchronous promotion causes TPP performance degradation during migration

NOMAD Overview

Goals:

- Enable the CPU to freely access both fast and slow memory
- Move page migration off the critical path of users' data access

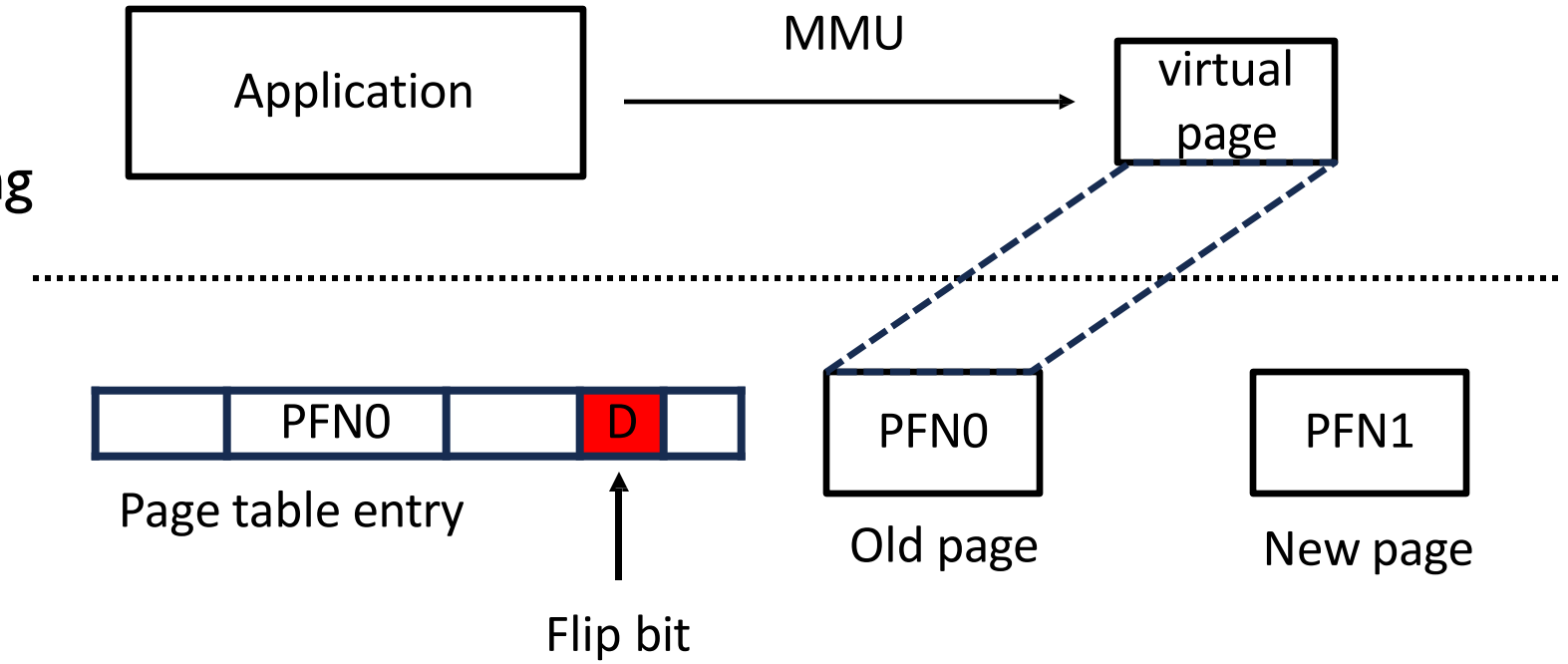
NOMAD Approaches:

- Transactional page migration
- Non-exclusive tiering via page shadowing

Design Overview

Key ideas:

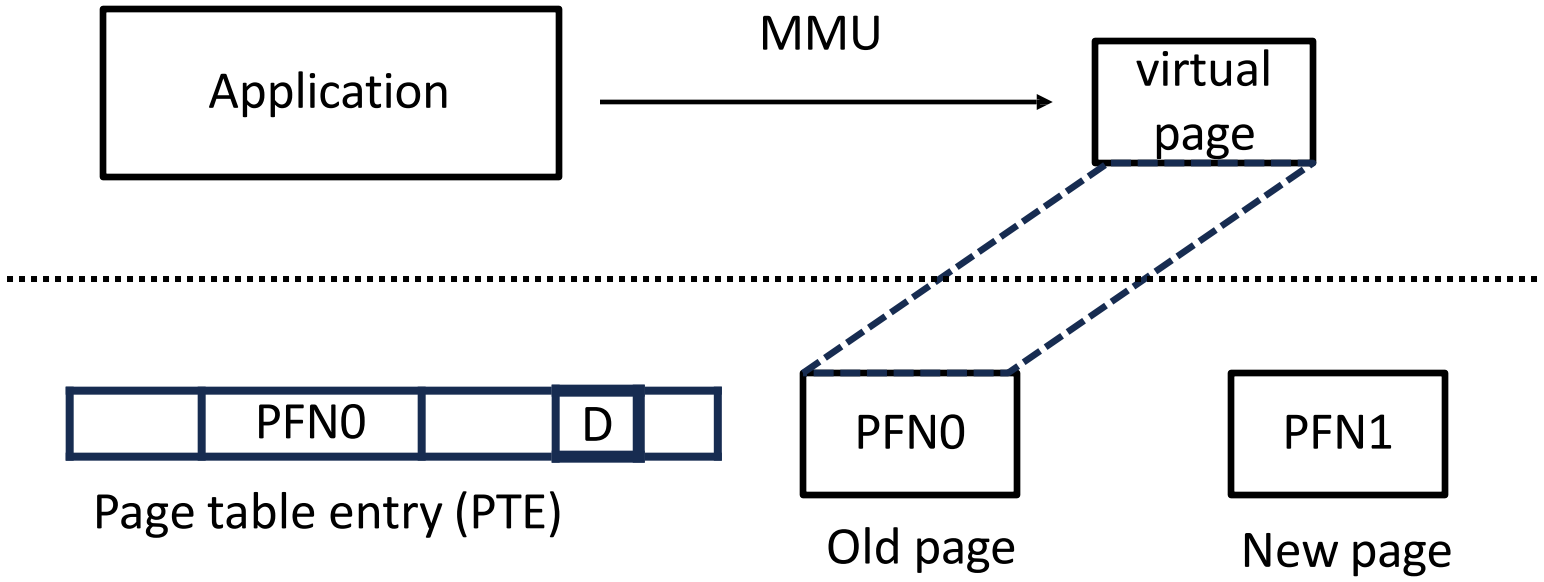
- Keep pages accessible during migration
- Invalidate the migration if pages are dirtied



Transaction Page Migration (TPM)

Major steps:

1. Clear the dirty bit in PTE

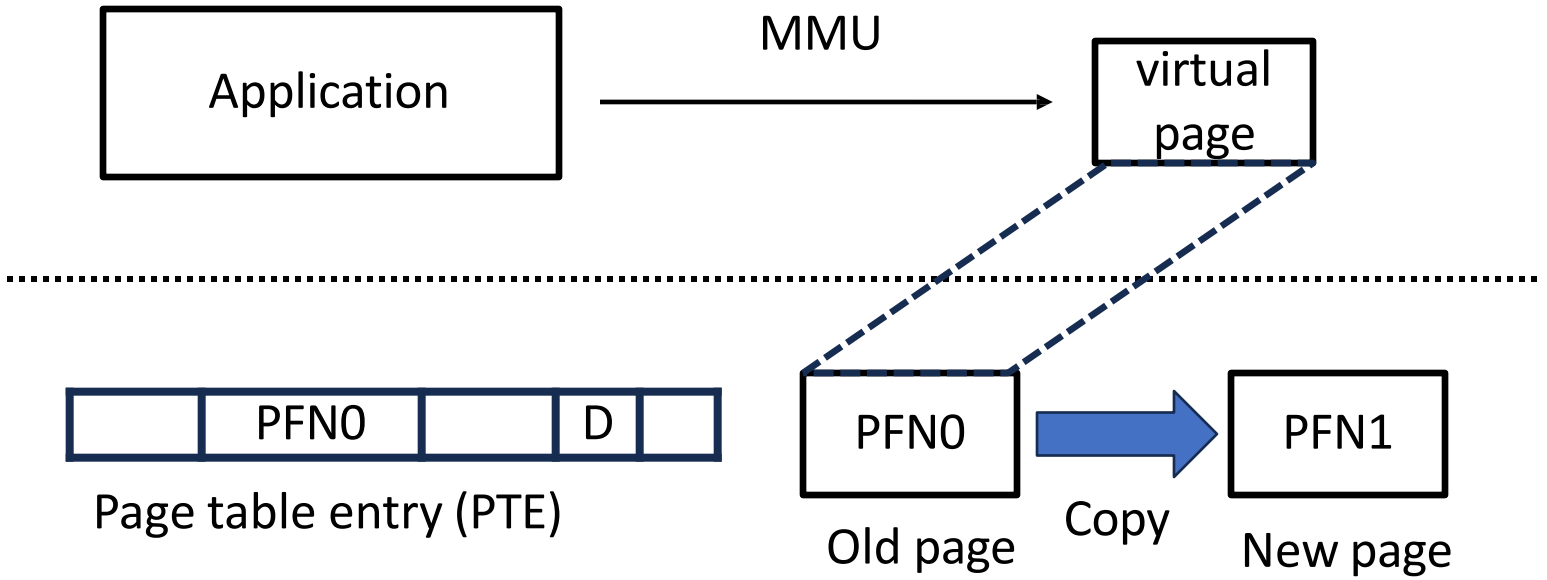


Issue a TLB shootdown.
The page remains accessible

Transaction Page Migration (TPM)

Major steps:

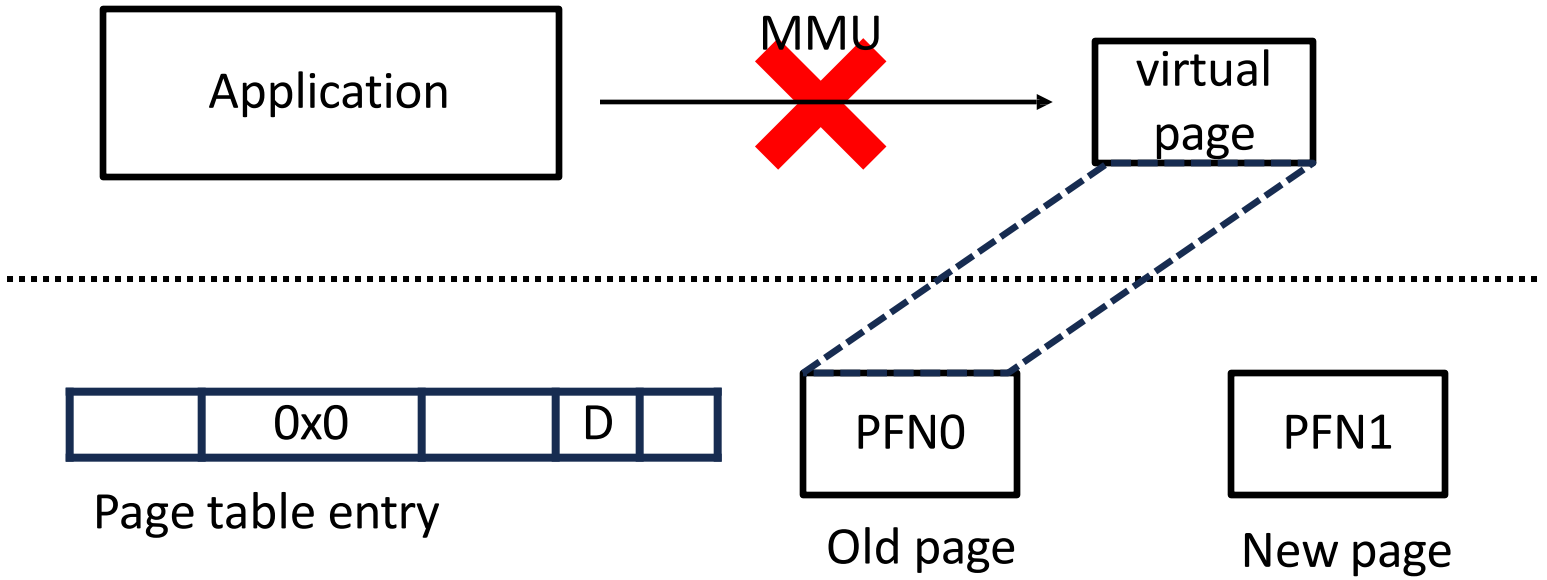
1. Clear the dirty bit in PTE
2. Copy page



Transaction Page Migration (TPM)

Major steps:

1. Clear the dirty bit in PTE
2. Copy page
3. Unmap page



Issue second TLB shootdown.
The page becomes inaccessible

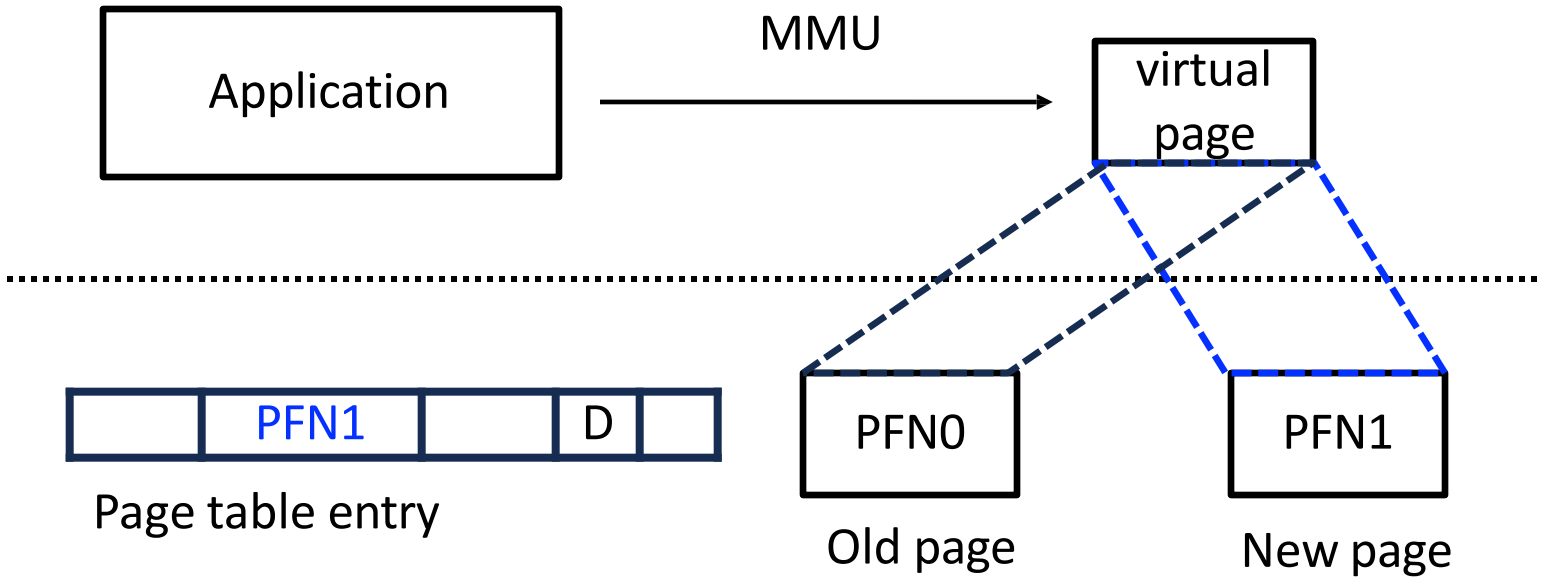
Transaction Page Migration (TPM)

Major steps:

1. Clear the dirty bit in PTE
2. Copy page
3. Unmap page

If D bit is clean:

4. Map the page to destination



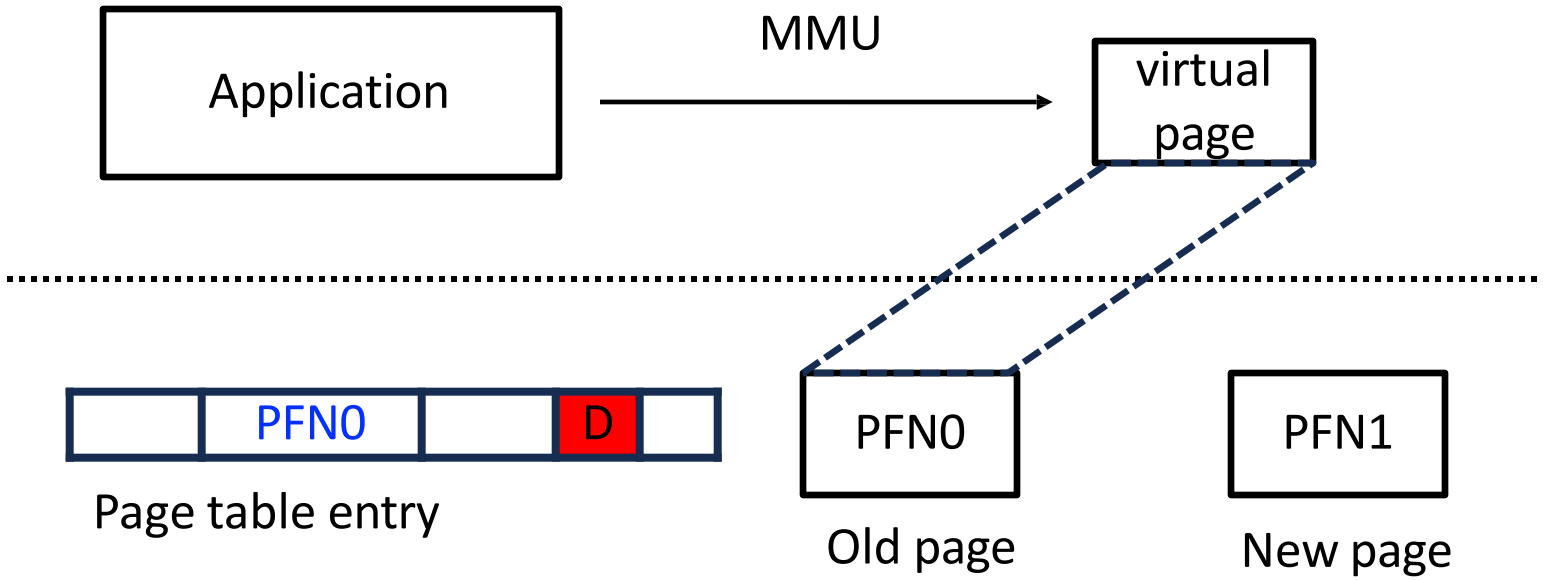
Transaction Page Migration (TPM)

Major steps:

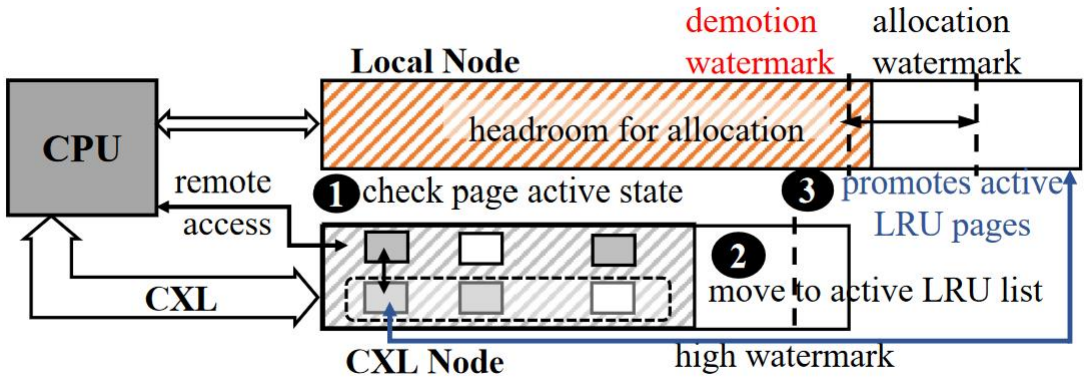
1. Clear the dirty bit in PTE
2. Copy page
3. Unmap page

If D bit is dirty:

4. Map the page to source and abort migration



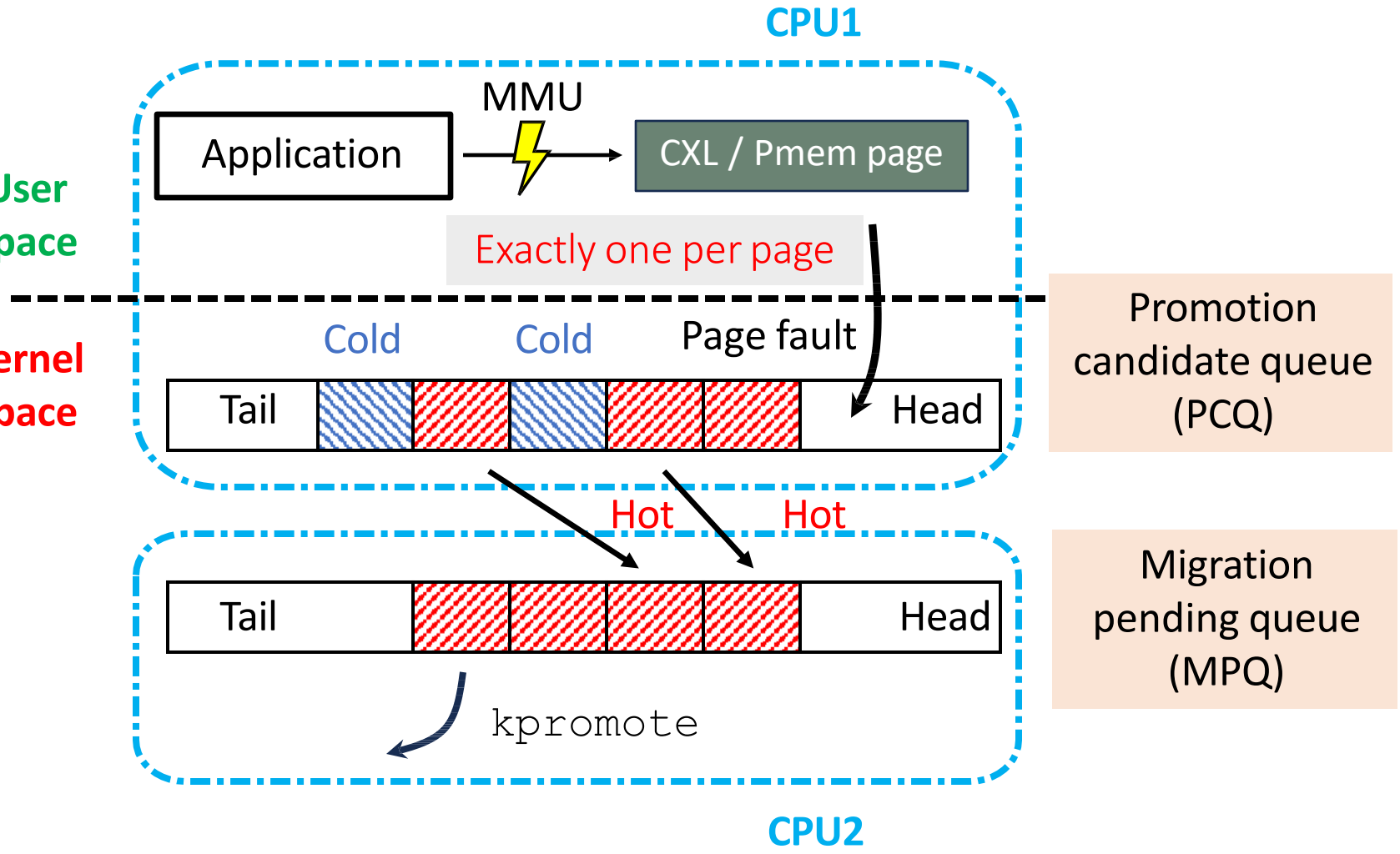
Minimizing Page Faults



Linux batches (up to 15) LRU movements to reduce queue management overhead. In the worst case, migrating one page may generate as many as 15 page faults.

Minimizing Page Faults

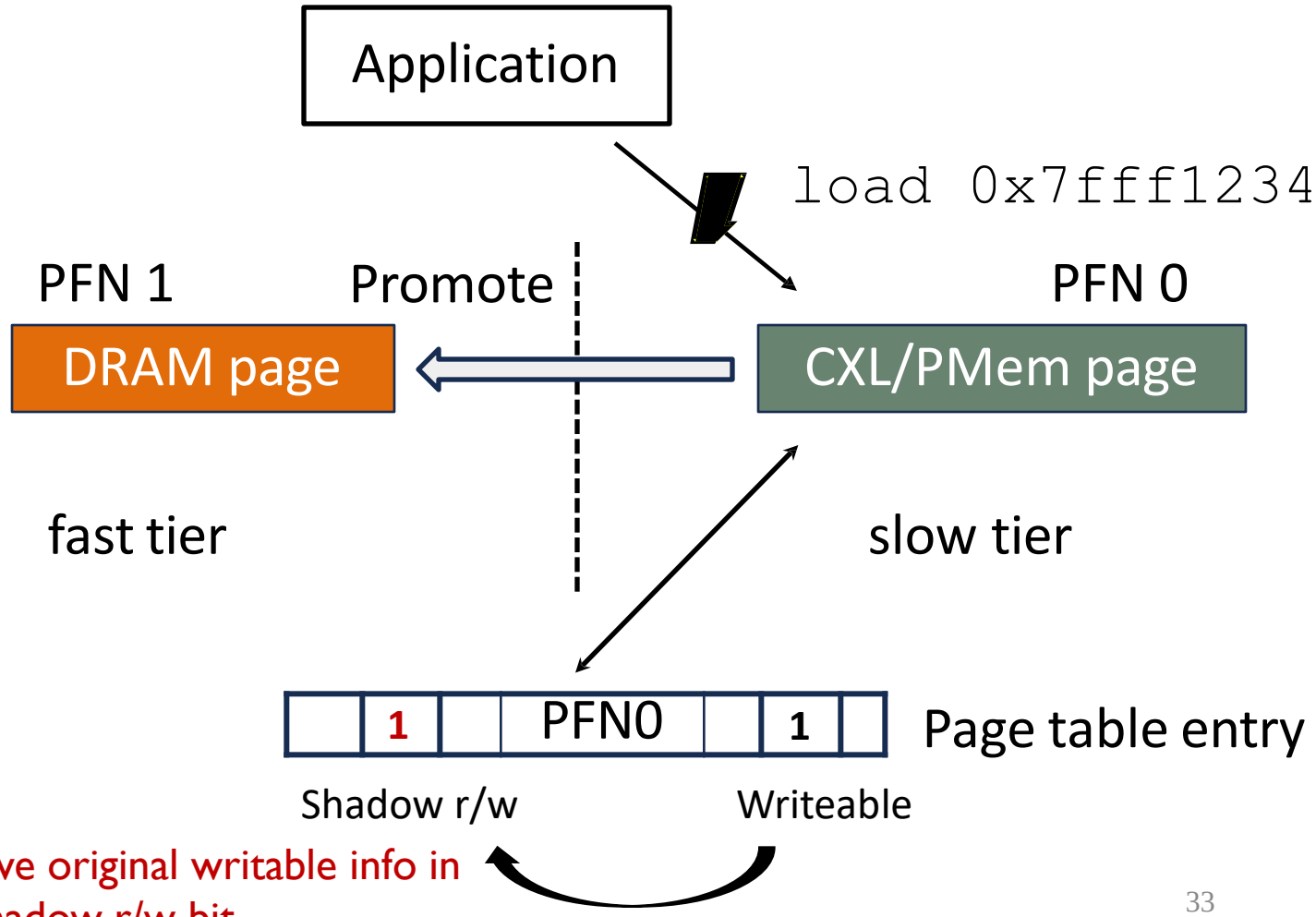
- NOMAD introduces PCQ to record pages that faulted but not in the active list.
- Scan PTEs of pages in PCQ on every fault to decide whether promote it to MPQ
- `kpromote` asynchronously performs TPM on pages in MPQ



Page Shadowing

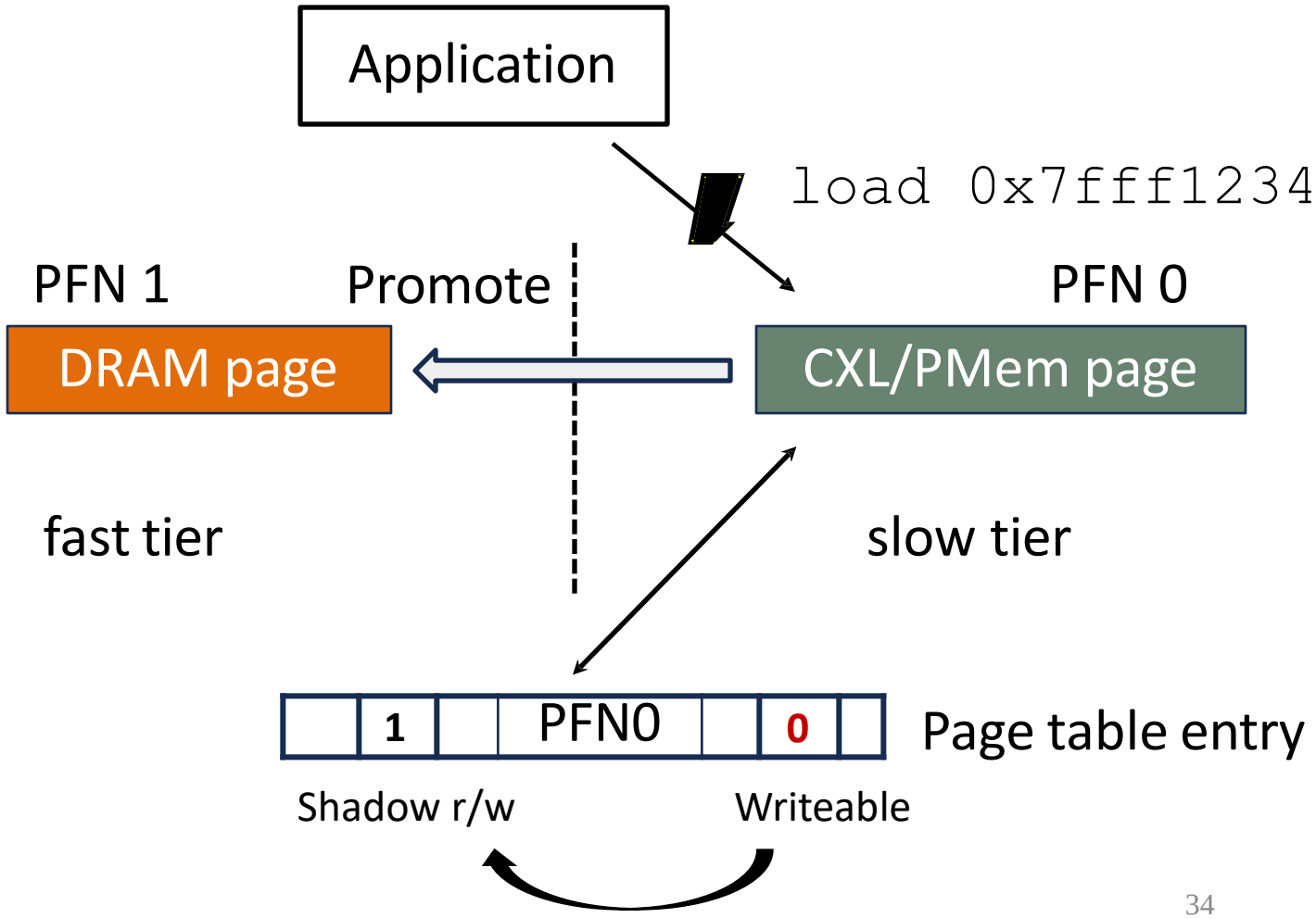
Key ideas:

Keep a shadow copy of a page promoted from slow tier to fast tier.



Page Shadowing

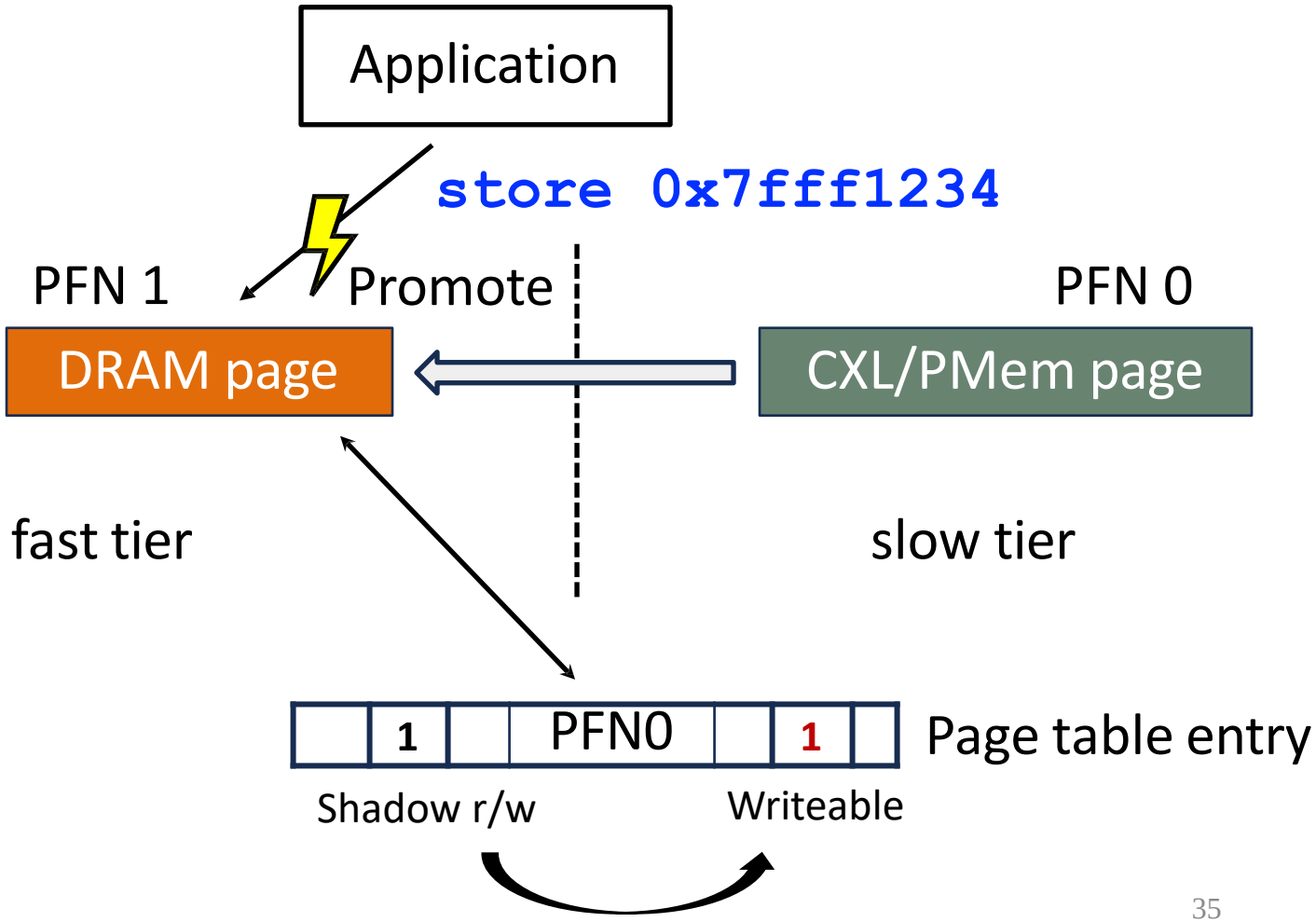
NOMAD marks all master pages as read-only



Page Shadowing

Restore writable bit on page writes:

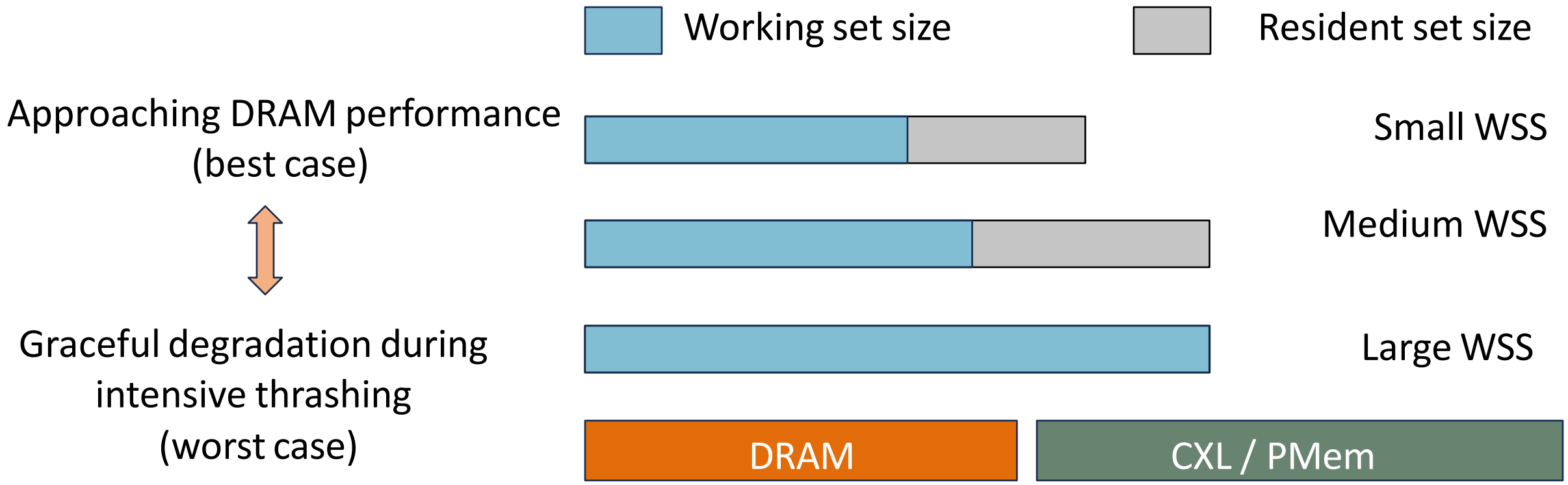
- No overhead for read-only pages
- Single page fault for writable pages



Testbeds

- Intel CPU + 16 GB Intel FPGA-based CXL memory (CXL-FPGA)
- Intel CPU + 6 x 256GB Optane PMem (PMem)
- AMD CPU + 4 x 256GB Micron CXL memory (CXL-Product)
- DRAM: 16 GB

Microbenchmarks



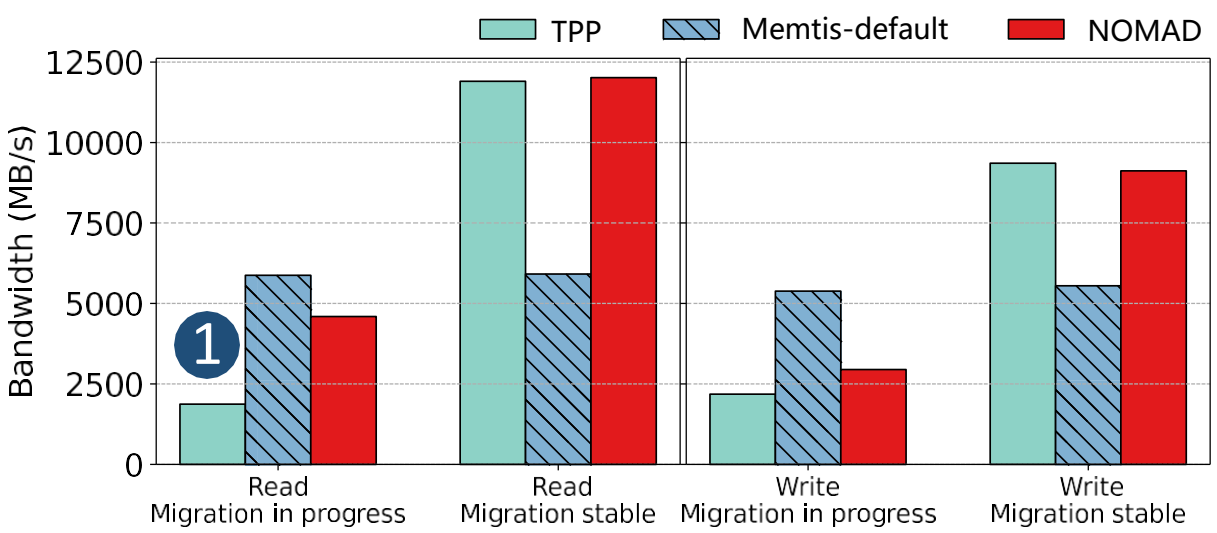
Migration in-progress

Migration stable

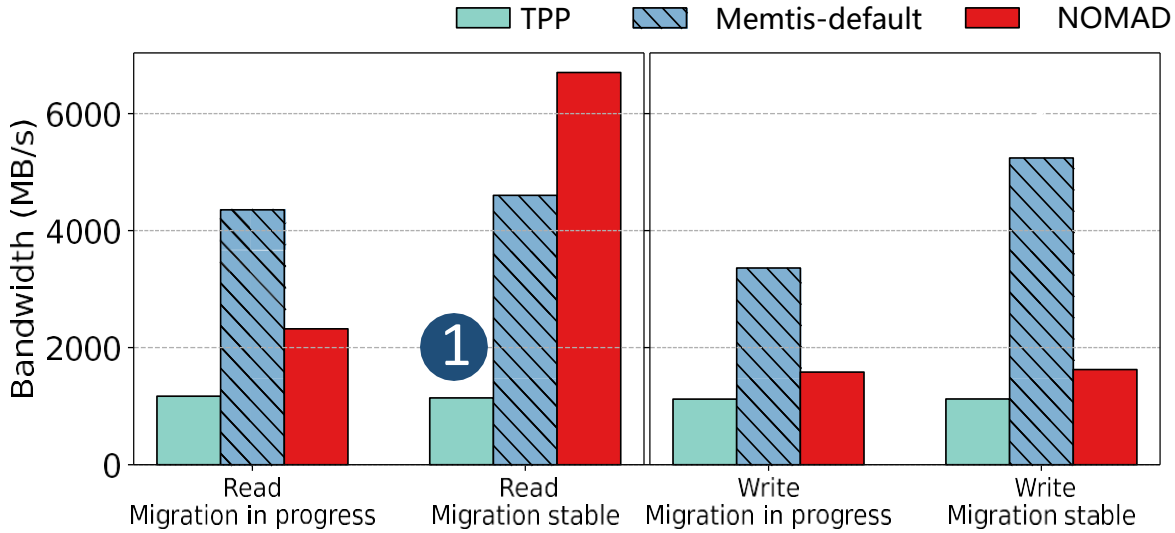
Microbenchmarks

USTC, CHINA ADSLAB

Testbed: CXL-FPGA



Small WSS



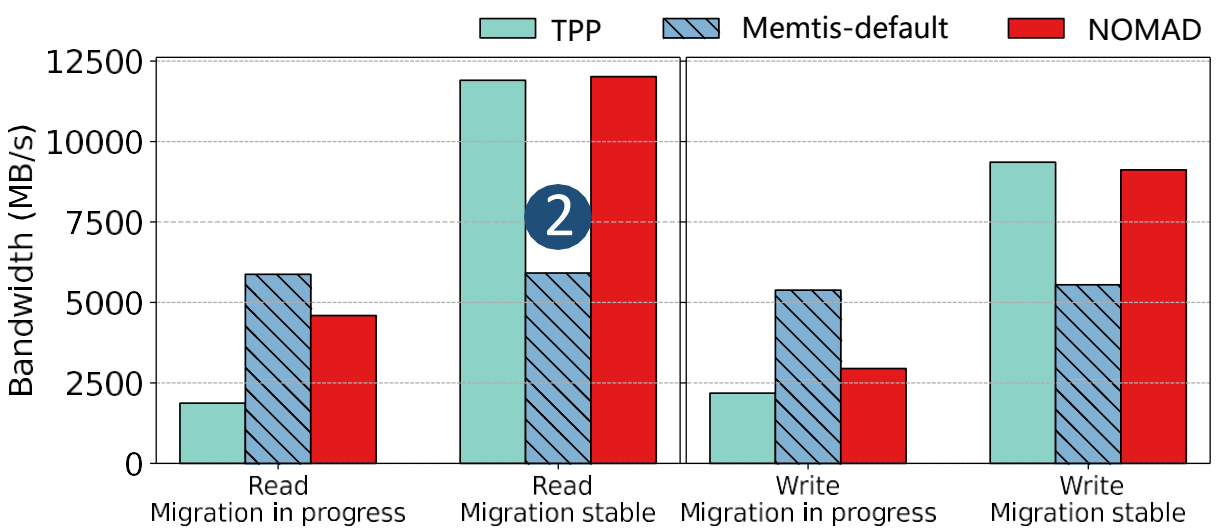
Large WSS

1. NOMAD significantly outperforms TPP during active migration and for large WSS

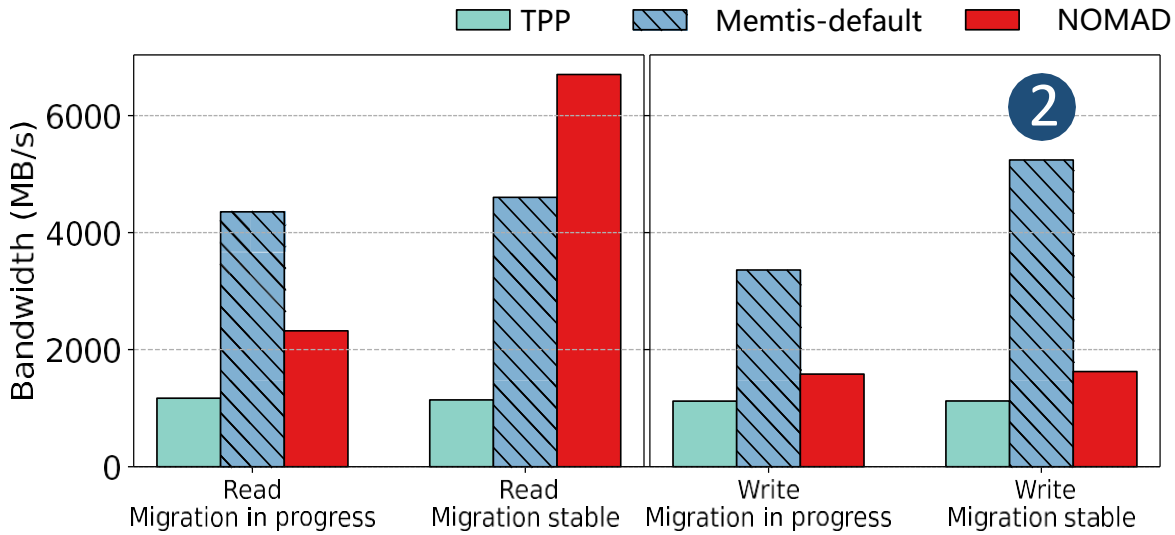
Microbenchmarks

USTC, CHINA ADSLAB

Testbed: CXL-FPGA



Small WSS



Large WSS

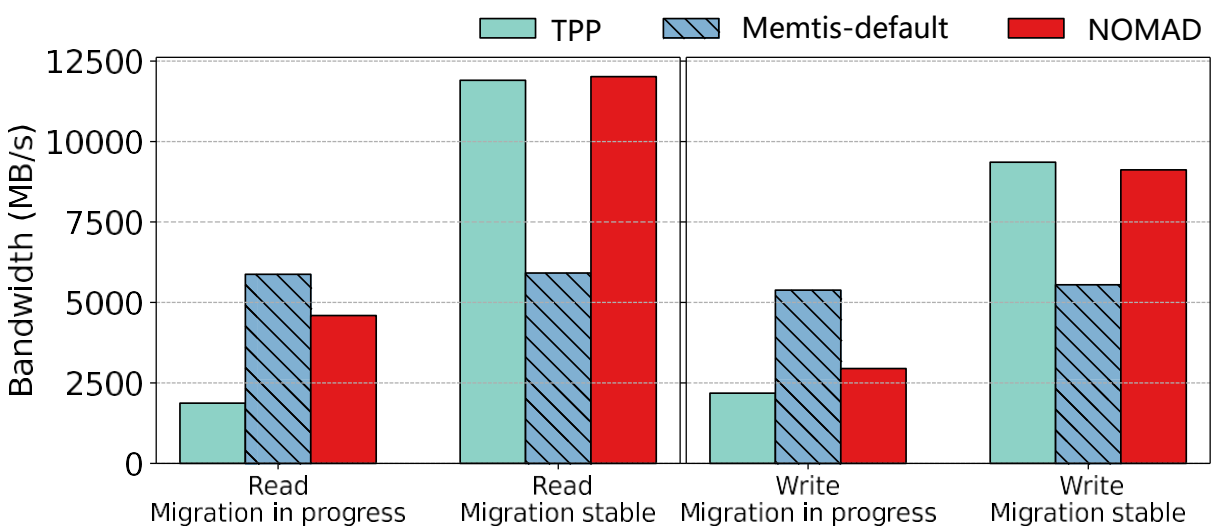
1. NOMAD significantly outperforms TPP during active migration and for large WSS

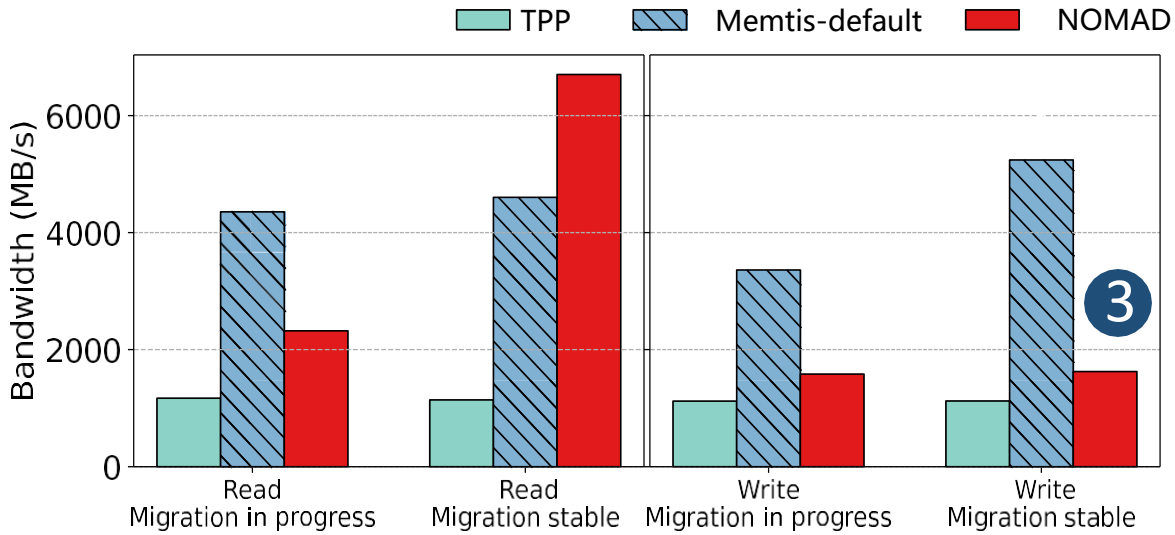
2. Sampling-based approach (Memtis) achieves stable performance during thrashing but fails to optimally place hot data in fast memory

Microbenchmarks

USTC, CHINA ADSLAB

Testbed: CXL-FPGA





Small WSS

Large WSS

1. NOMAD significantly outperforms TPP during active migration and for large WSS

2. Sampling-based approach (Memtis) achieves stable performance during thrashing but fails to optimally place hot data in fast memory

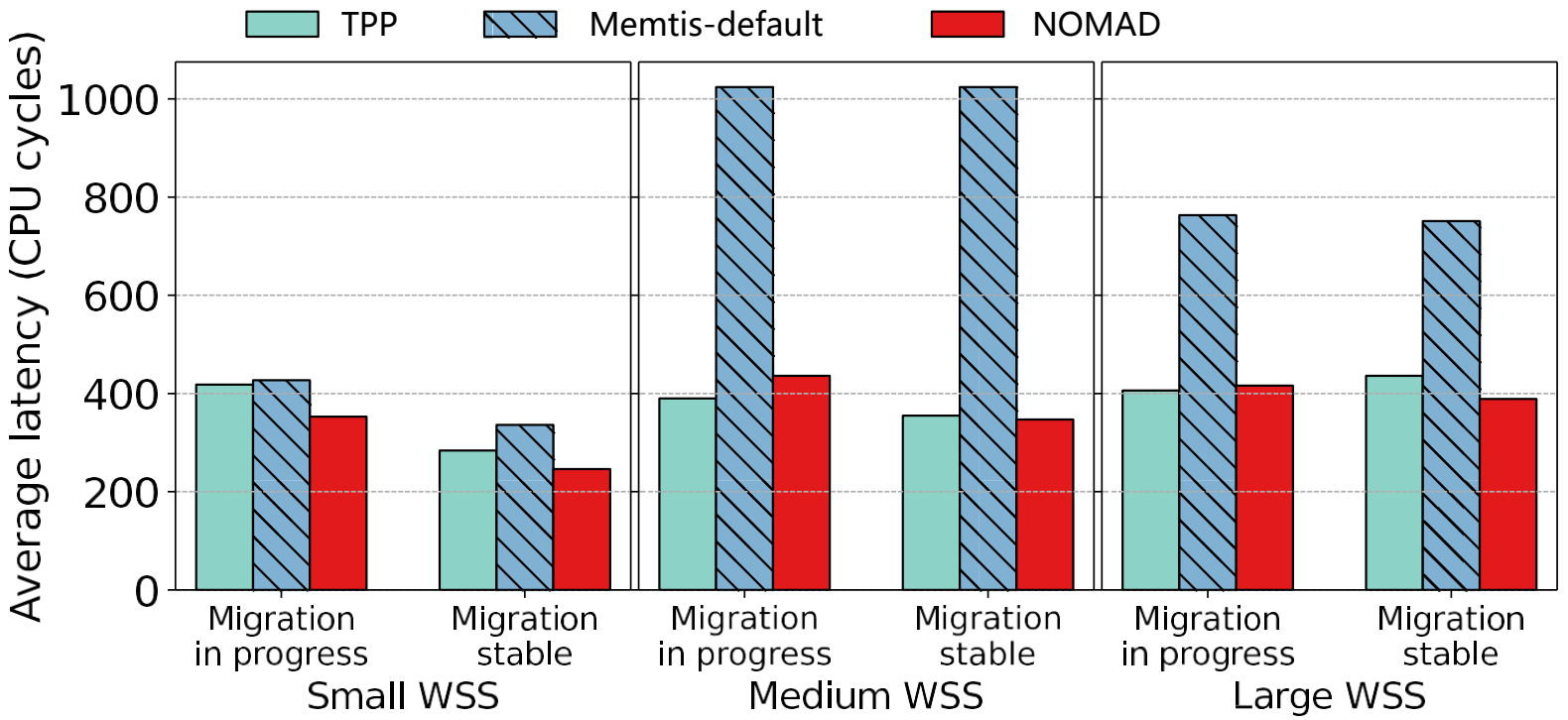
3. NOMAD is more effective for read-only workloads and suffers from migration abortions for write-intensive workloads

Microbenchmarks

USTC, CHINA ADSLAB

Testbed: PMem

Lower is better



Although the sampling-based approach maintains high throughput during thrashing thanks to a lack of migrations, its latency is sub-optimal, suggesting page migration is ineffective

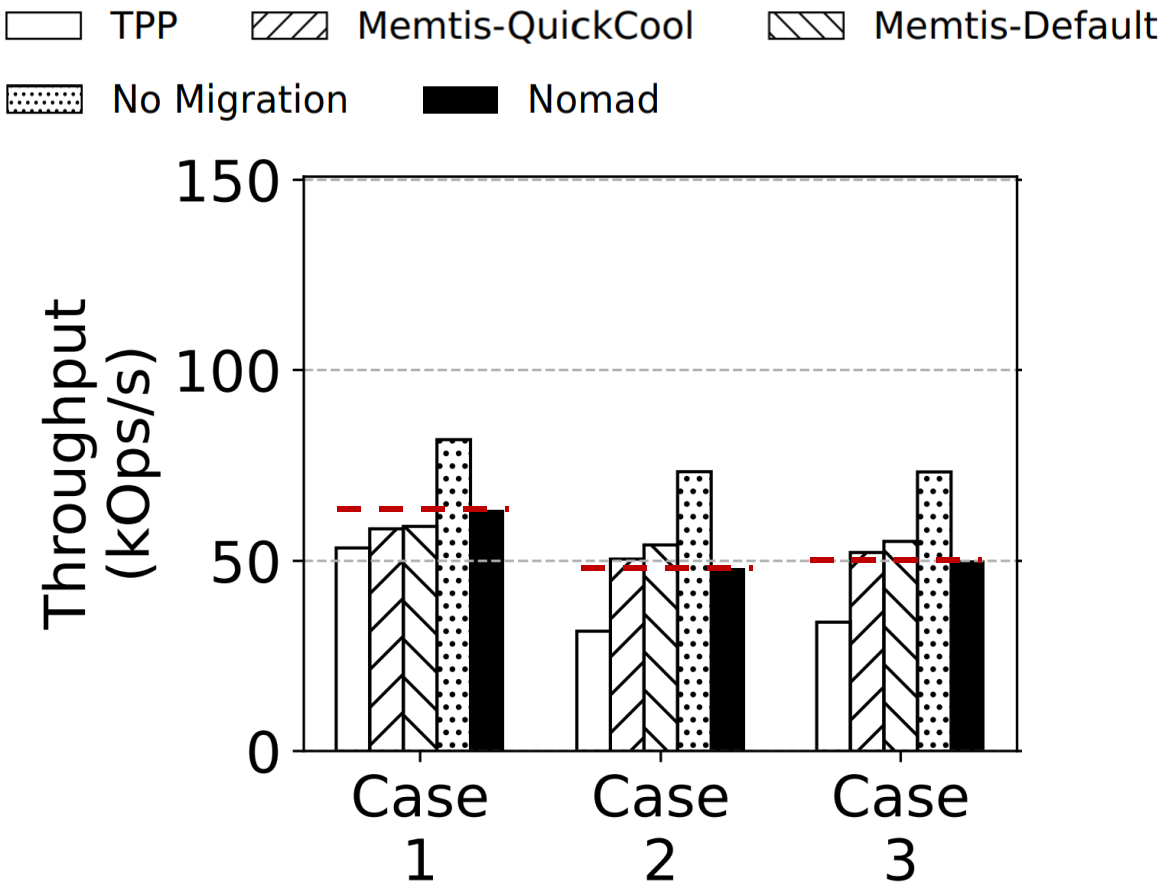
Real-world Applications - Redis

Testbed: CXL-FPGA

Case 1: 13GB RSS, demote pages

Case 2: 24GB RSS, demote pages

Case 3: 24GB RSS, no demotion



NOMAD outperforms TPP in all cases and outperforms Memtis when WSS fits in fast tier.

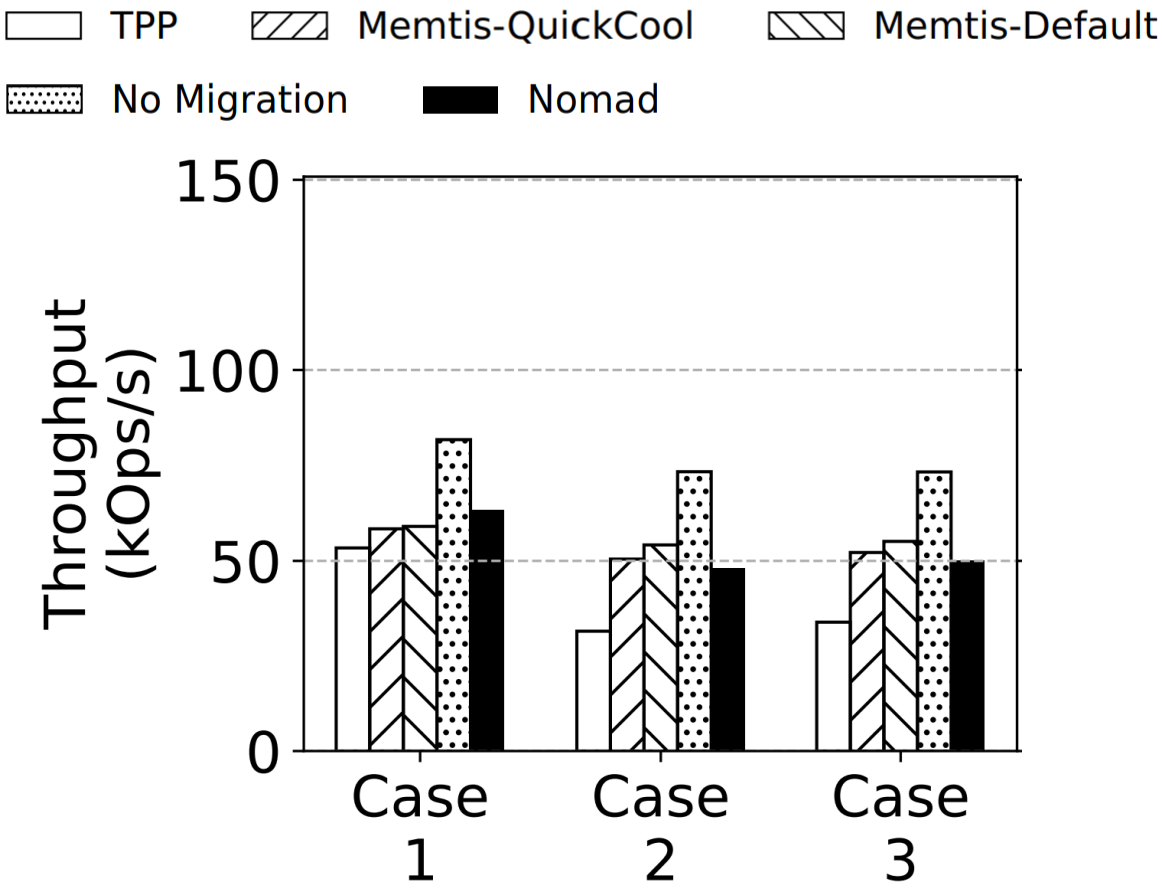
Real-world Applications - Redis

Testbed: CXL-FPGA

Case 1: 13GB RSS, demote pages

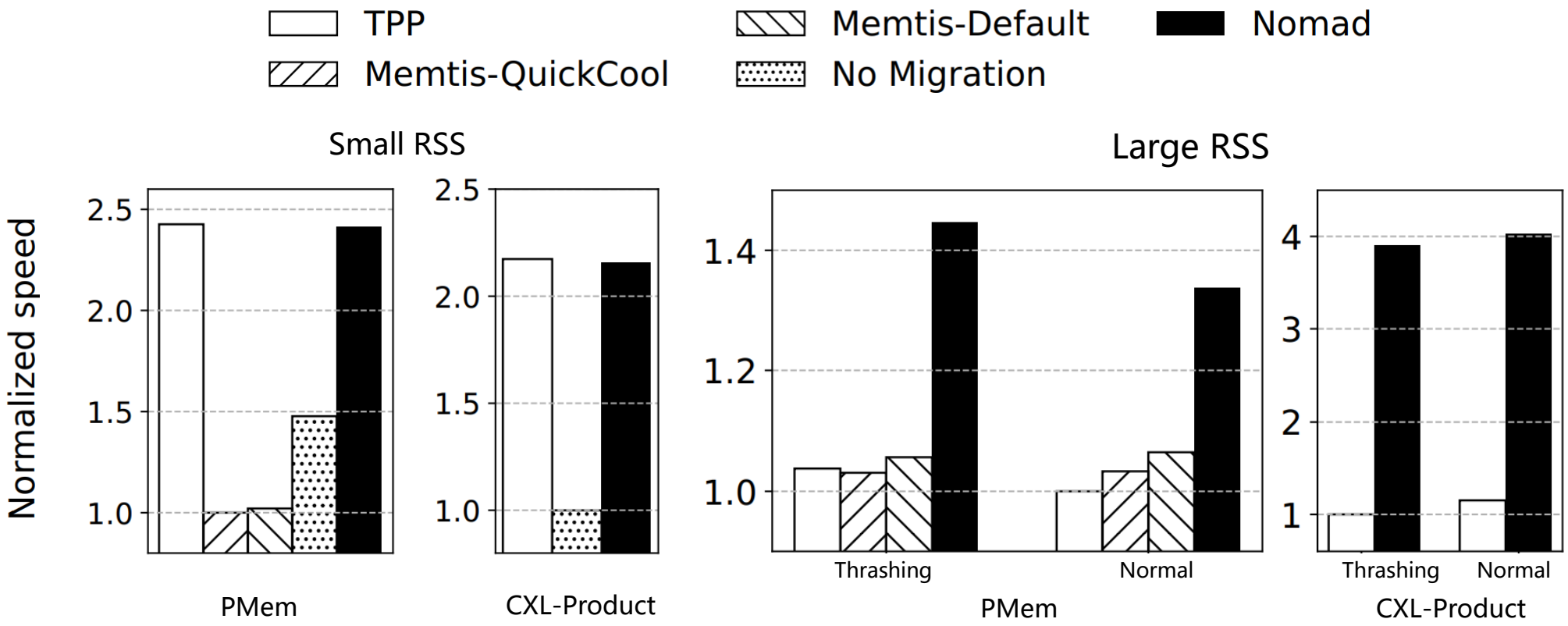
Case 2: 24GB RSS, demote pages

Case 3: 24GB RSS, no demotion



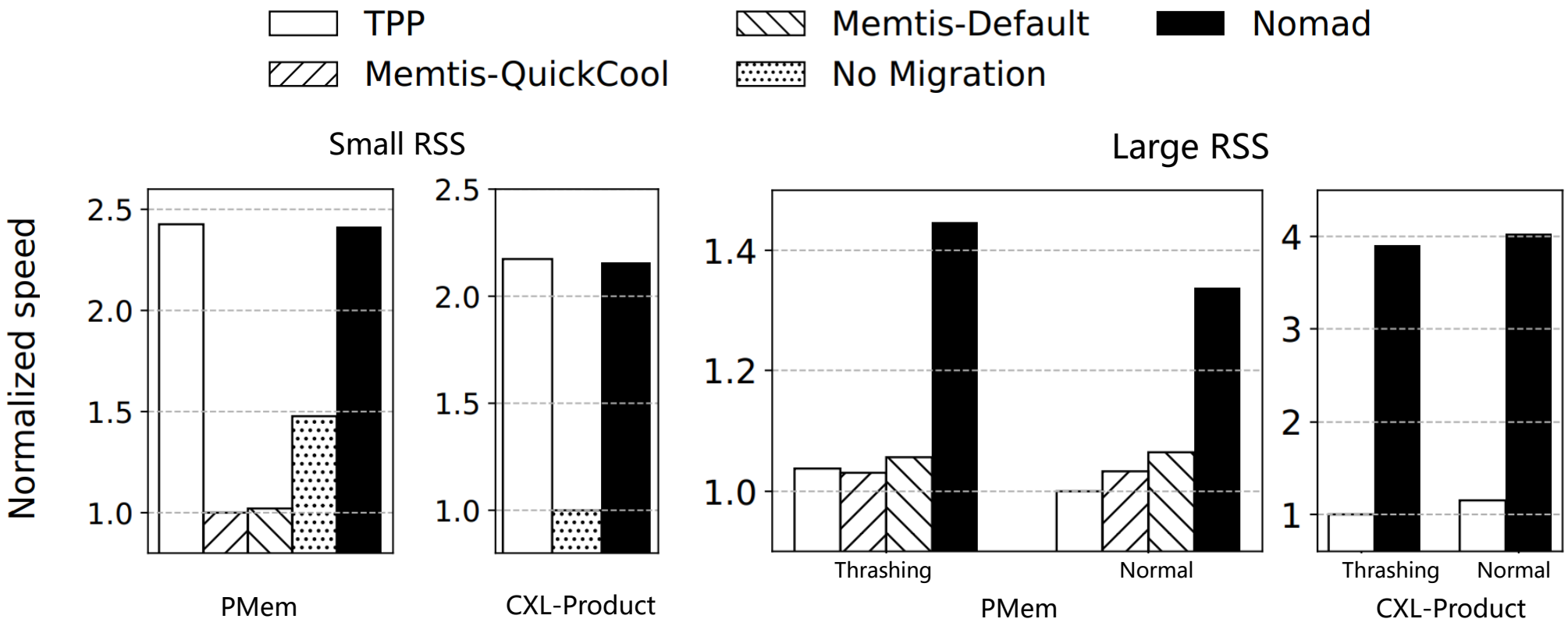
Page migration could incur nontrivial overhead, and a strategy to dynamically switch it on/off is needed.

Real-world Applications - liblinear



1. Page fault-based approaches outperform Memtis and no-migration for small RSS

Real-world Applications - liblinear



1. Page fault-based approaches outperform Memtis and no-migration for small RSS

2. TPP's performance significantly declines due to inefficient migration.

Conclusions

NOMAD is a tiered memory management mechanism that features:

- Transactional page migration
- Page shadowing
- Non-exclusive memory tiering

Results show that NOMAD is significantly more efficient than the state-of-the- art tiered memory management scheme in Linux but call for more research on memory tiering

- The optimal strategy to enable/disable page migrations under high memory pressure