

A Case for Hardware-Based Demand Paging

**Gyusun Lee^{1*}, Wenjing Jin^{2*}, Wonsuk Song¹, Jeonghun Gong², Jonghyun Bae²
Tae Jun Ham², Jae W. Lee² and Jinkyu Jeong¹**



Sungkyunkwan University (SKKU)¹

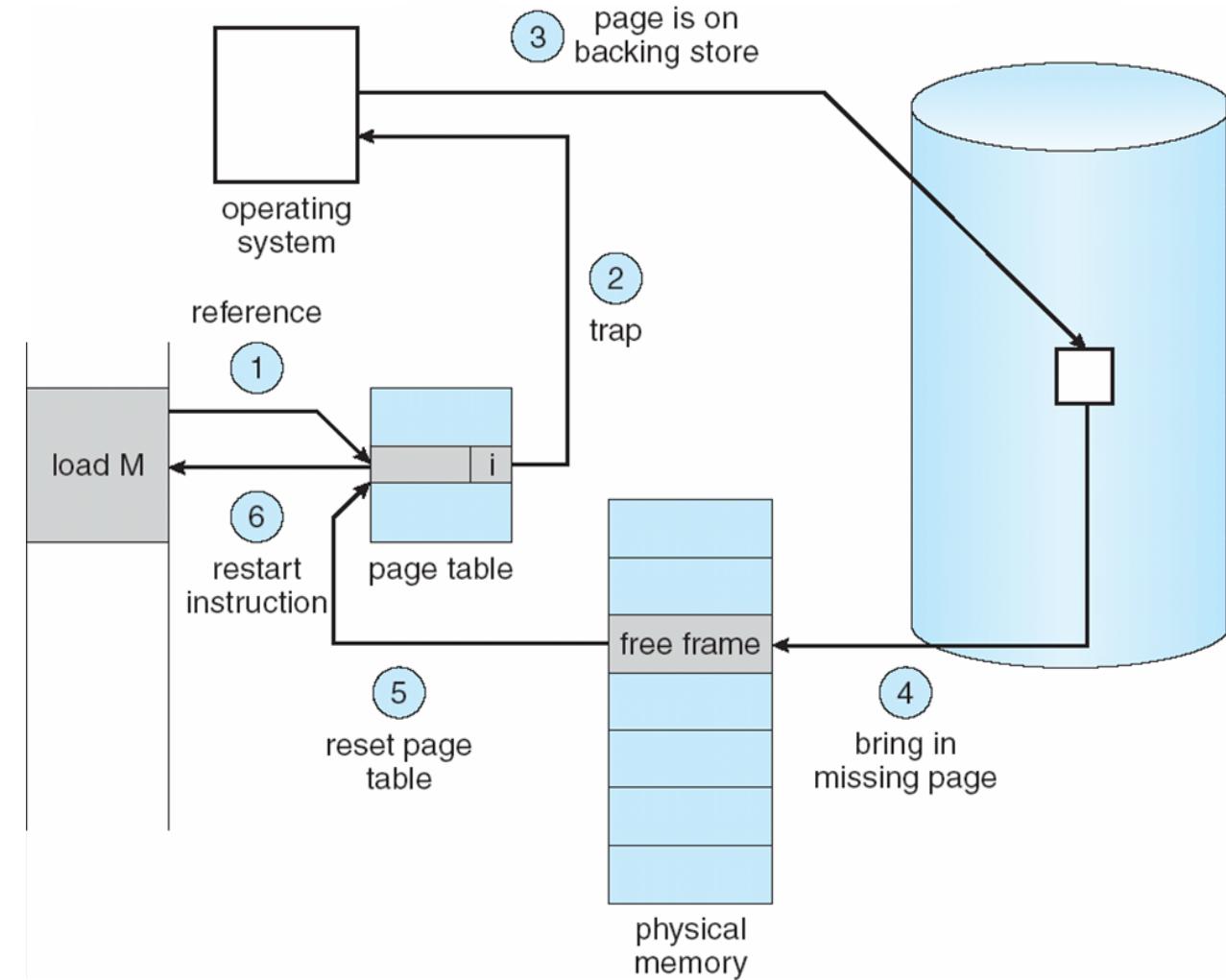


Seoul National University (SNU)²

* Equally contributed

Demand Paging

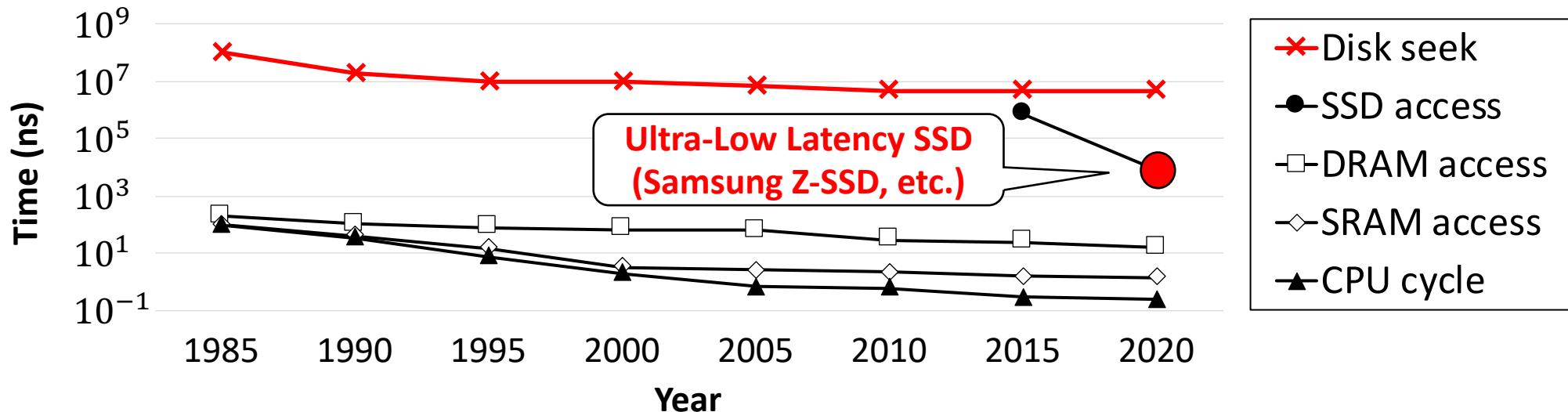
- Core mechanism of virtual memory
 - Memory as a cache of backing storage
 - Prevalent from mobile to cloud computing
- Benefits
 - Fewer I/Os needed
 - Less memory needed
 - Faster response
 - Better support for multiple processes



Source: A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts, 9th Edition, John Wiley & Sons, Inc. 2014.

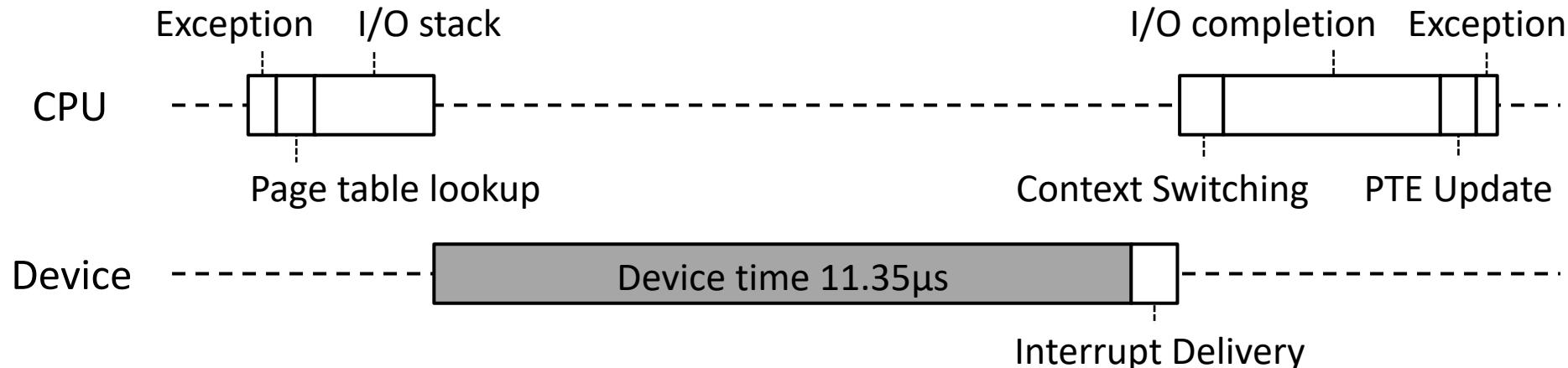
Bottleneck Shift in Demand Paging

Storage Performance Trends



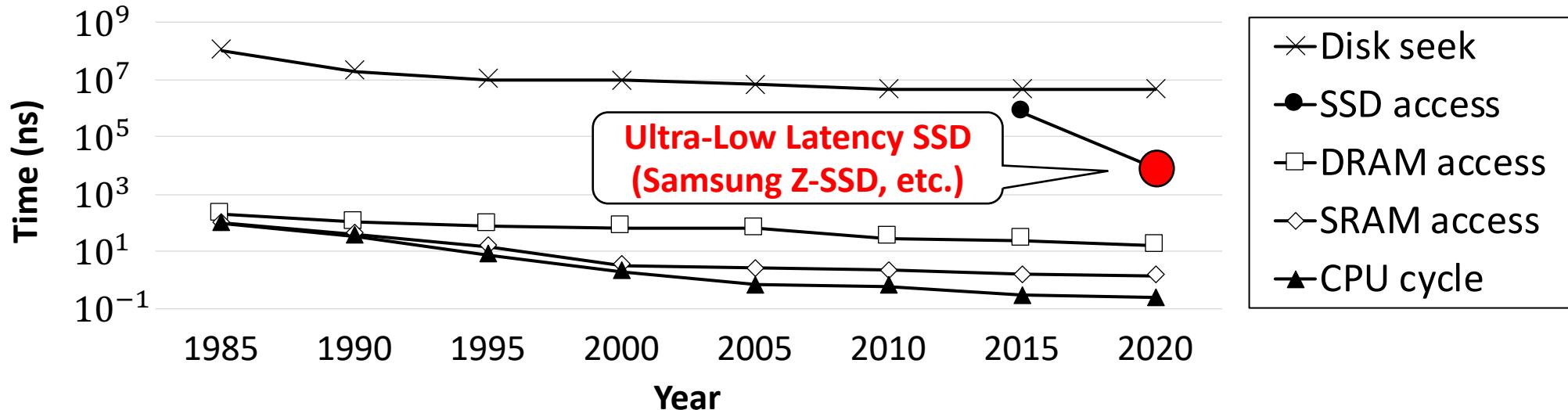
Source: R. E. Bryant and D. R. O'Hallaron, Computer Systems: A Programmer's Perspective, Second Edition, Pearson Education, Inc., 2015

Breakdown of Page Fault Handling



Bottleneck Shift in Demand Paging

Storage Performance Trends



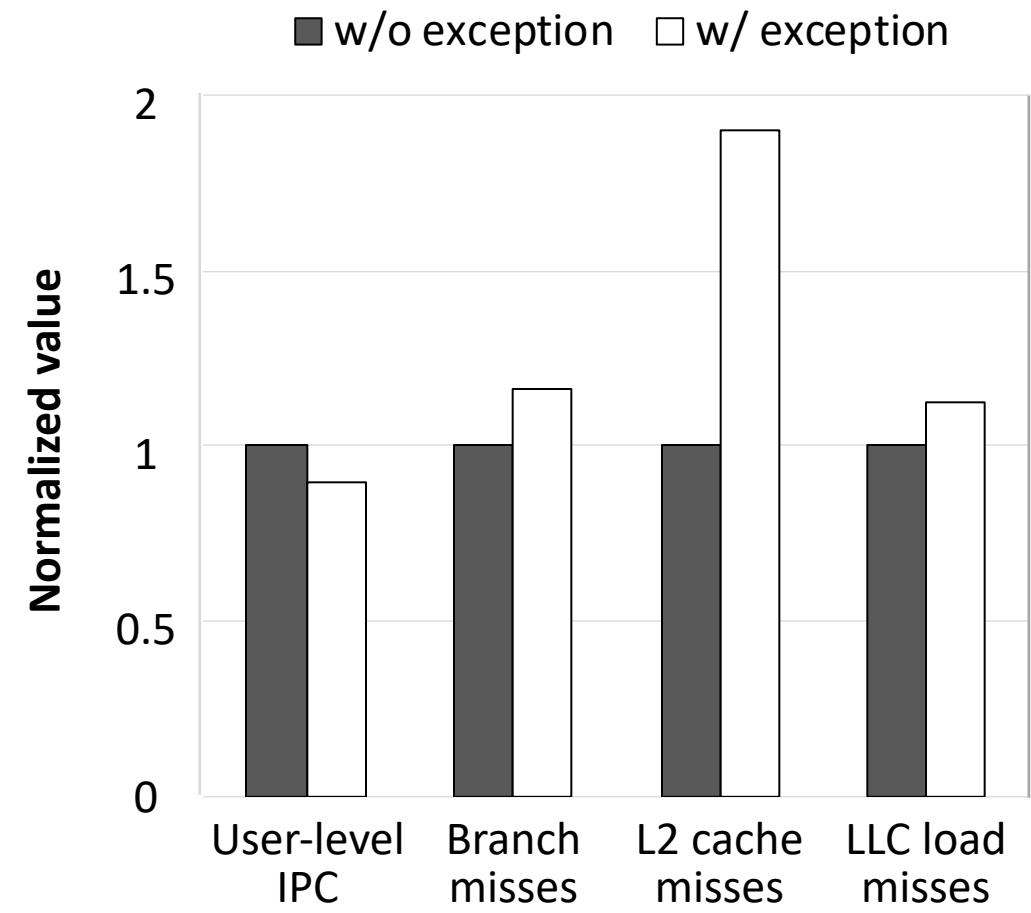
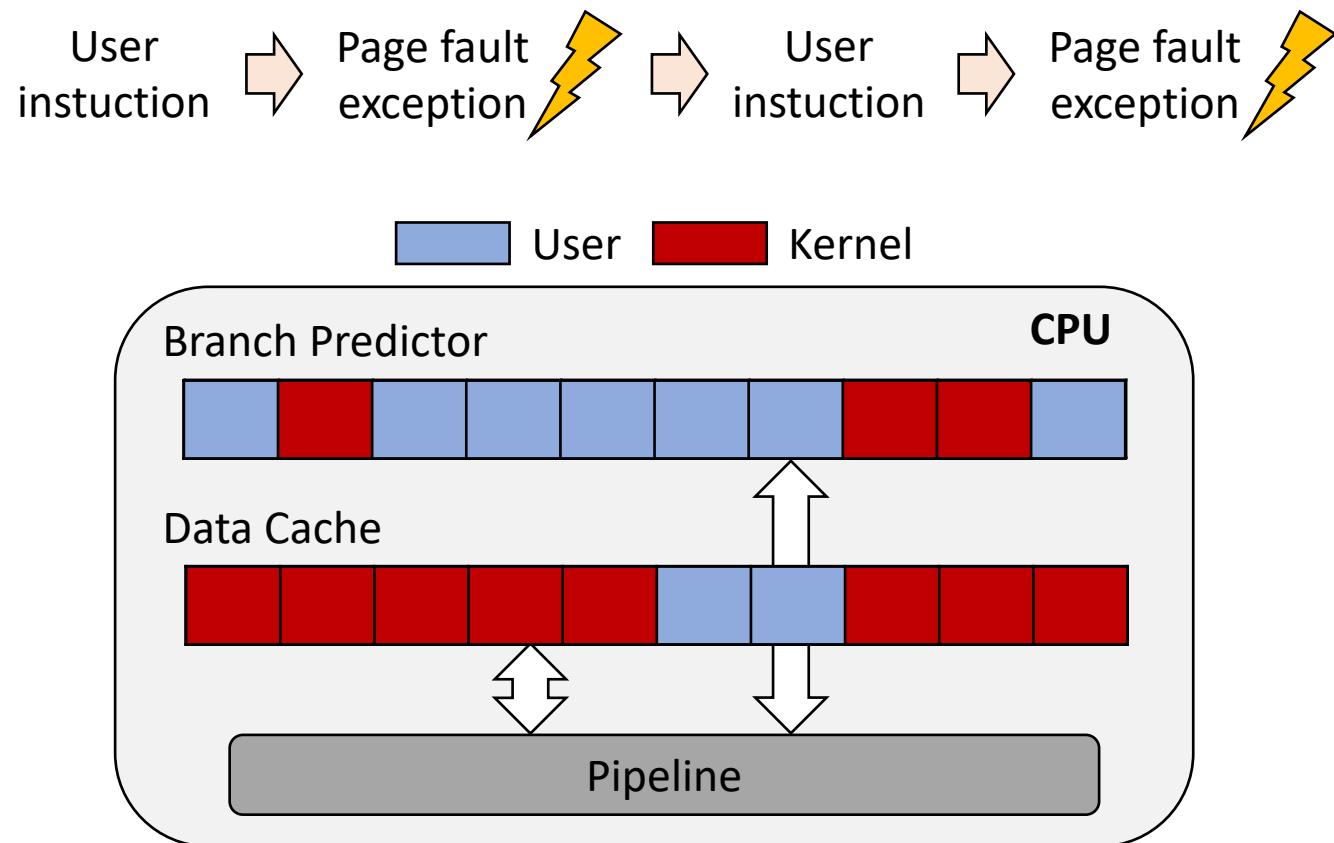
Source: R. E. Bryant and D. R. O'Hallaron, Computer Systems: A Programmer's Perspective, Second Edition, Pearson Education, Inc., 2015

Breakdown of Page Fault Handling



Indirect Cost of Demand Paging

- Architecture resource pollution (branch mispredictions, cache misses) by page faults



Why Hardware Doesn't Handle Page Faults?

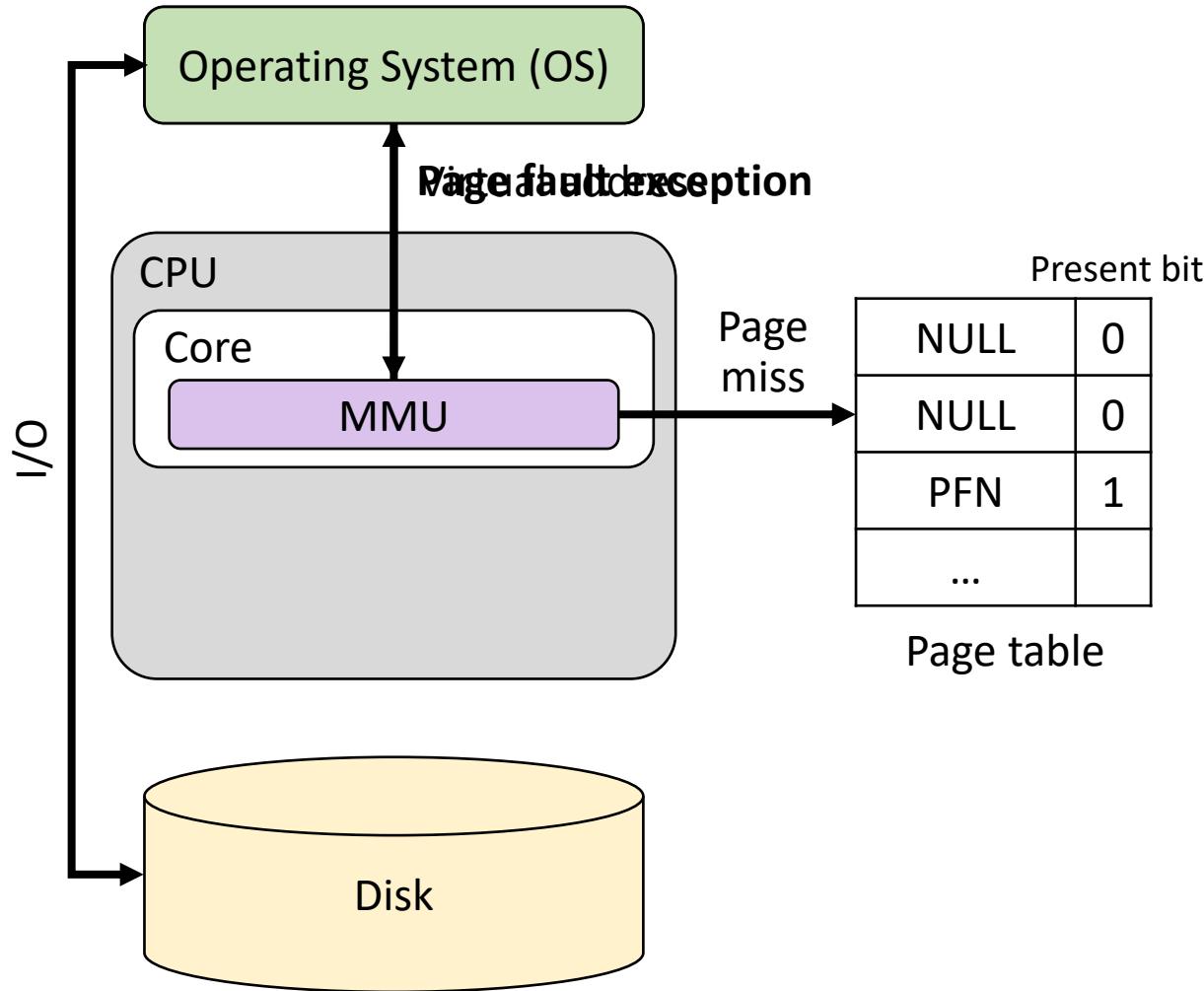
ASIDE: WHY HARDWARE DOESN'T HANDLE PAGE FAULTS

We know from our experience with the TLB that hardware designers are loathe to trust the OS to do much of anything. So why do they trust the OS to handle a page fault? There are a few main reasons. First, page faults to disk are slow; even if the OS takes a long time to handle a fault, executing tons of instructions, the disk operation itself is traditionally so slow that the extra overheads of running software are minimal. Second, to be able to handle a page fault, the hardware would have to understand swap space, how to issue I/Os to the disk, and a lot of other details which it currently doesn't know much about. Thus, for both reasons of performance and simplicity, the OS handles page faults, and even hardware types can be happy.

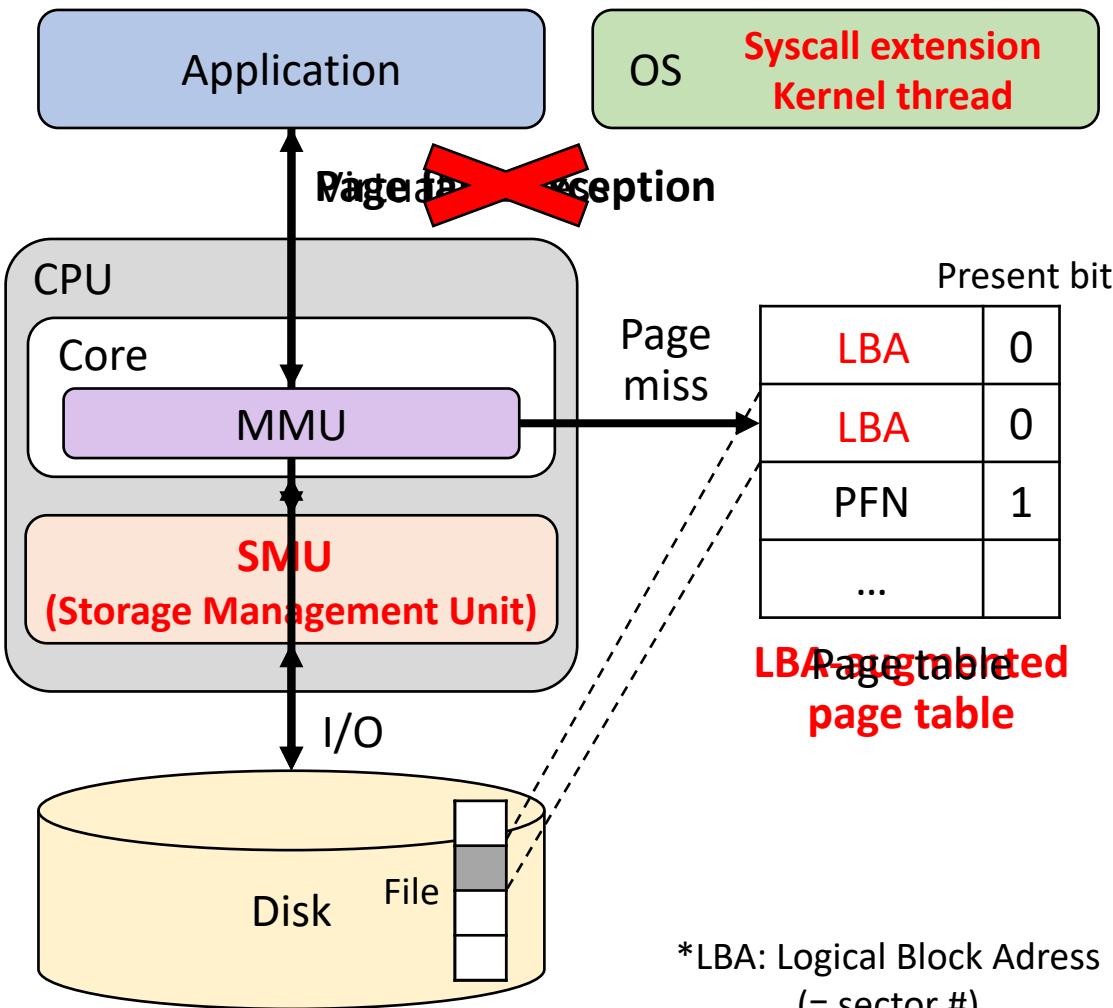
Andrea C. Arpaci-Dusseau

Source: Arpaci-Dusseau, Remzi H., Arpaci-Dusseau, Andrea C. *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC, 2018.

Operating System-Based Demand Paging (OSDP)



Hardware-Based Demand Paging (HWDP)



■ Challenges

- How to make a CPU understand storage layout?
 - **LBA*-augmented page table**
 - How to make a CPU control a storage I/O device?
 - **Storage Management Unit**
 - How to handle remaining page fault operations in the OS kernel?
 - **OS extensions (syscall extension, kernel thread...)**

■ Benefits

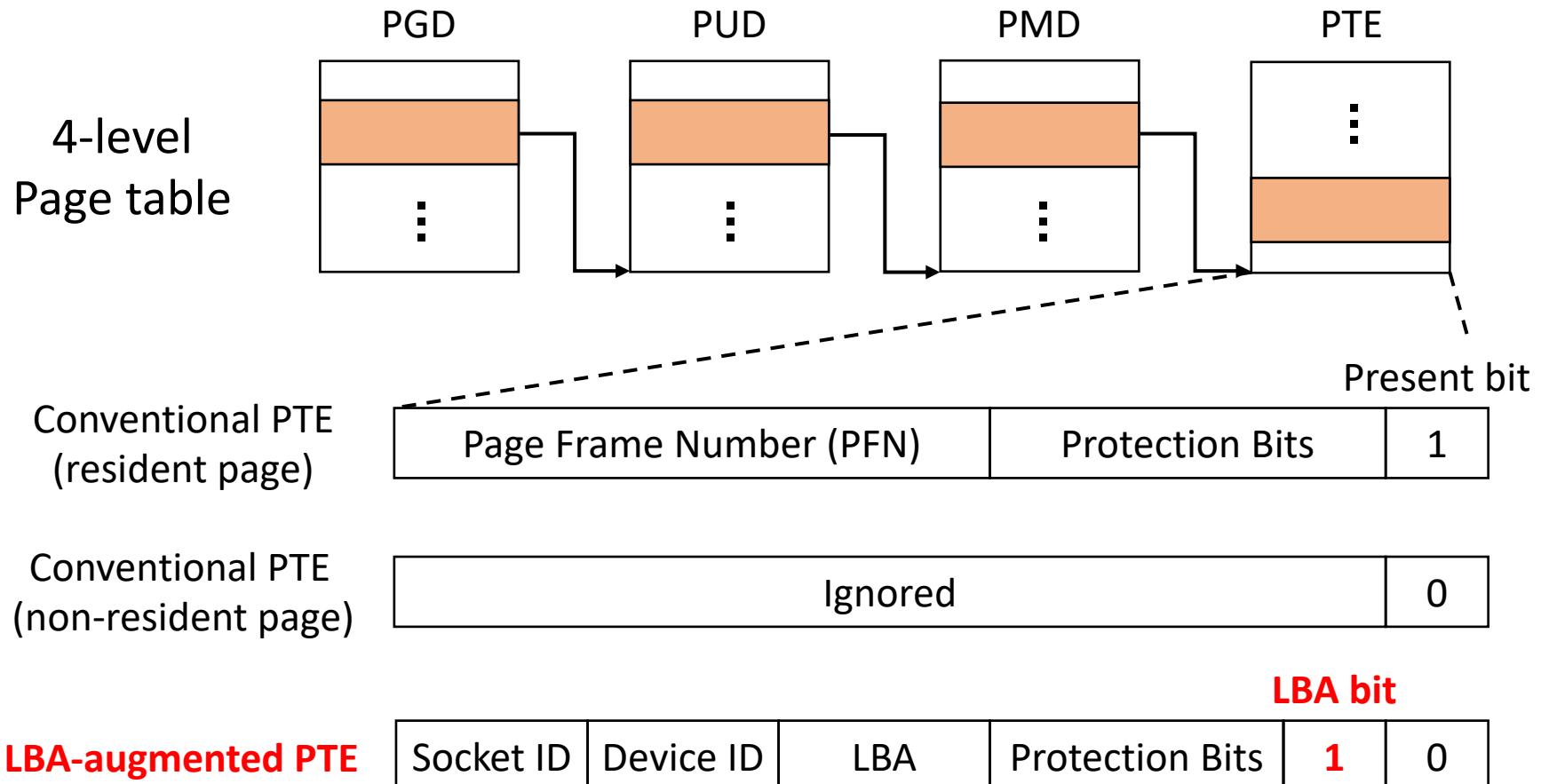
- Eliminates OS intervention (no exception occurs)
 - Reduces demand paging latency
 - Mitigates architecture resource pollution

Outline

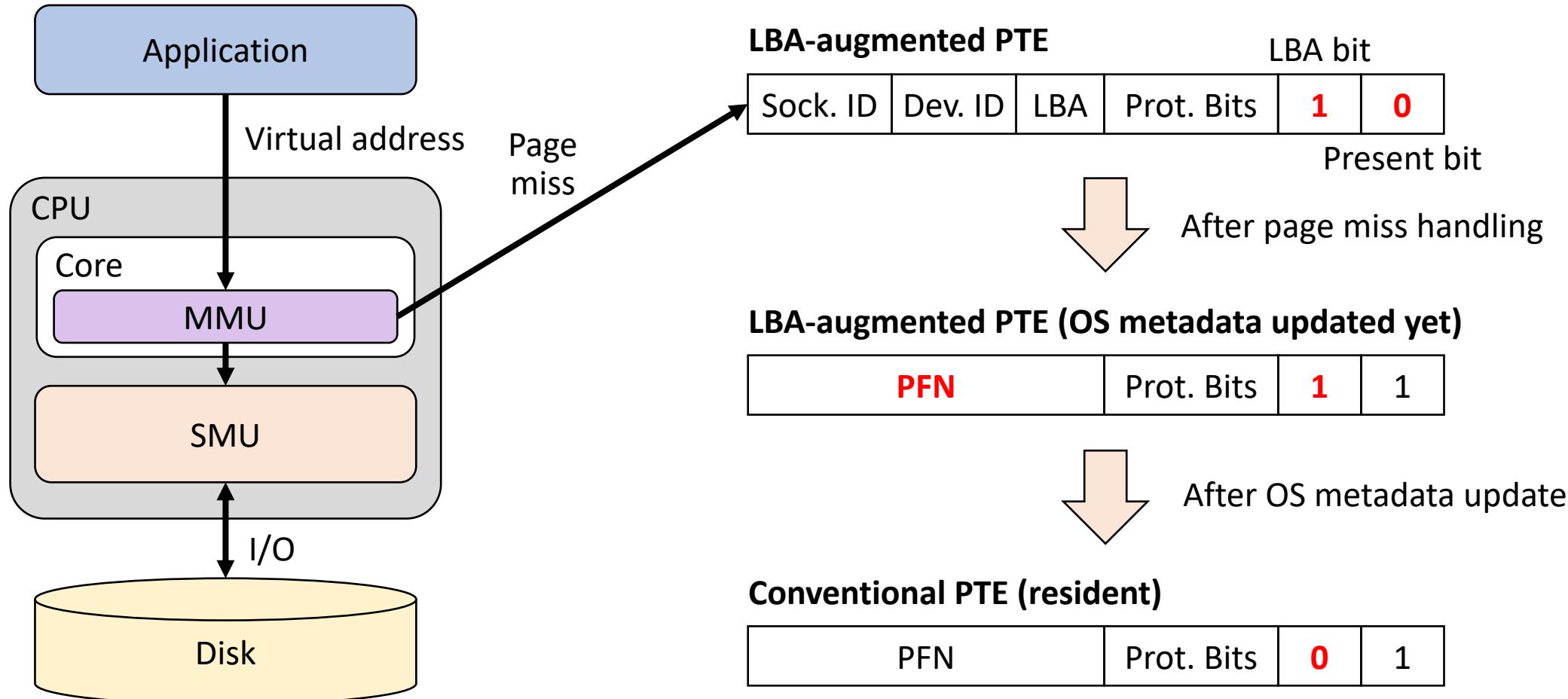
- Overview
- Architectural extensions
 - LBA-augmented page table
 - Storage Management Unit (SMU)
- OS supports
 - Fast file mmap()
 - Updating OS metadata for HWDP
 - Management of free page queue
- Evaluation
- Conclusion

LBA-augmented Page Table

- CPU understand storage layout through LBA-augmented page table

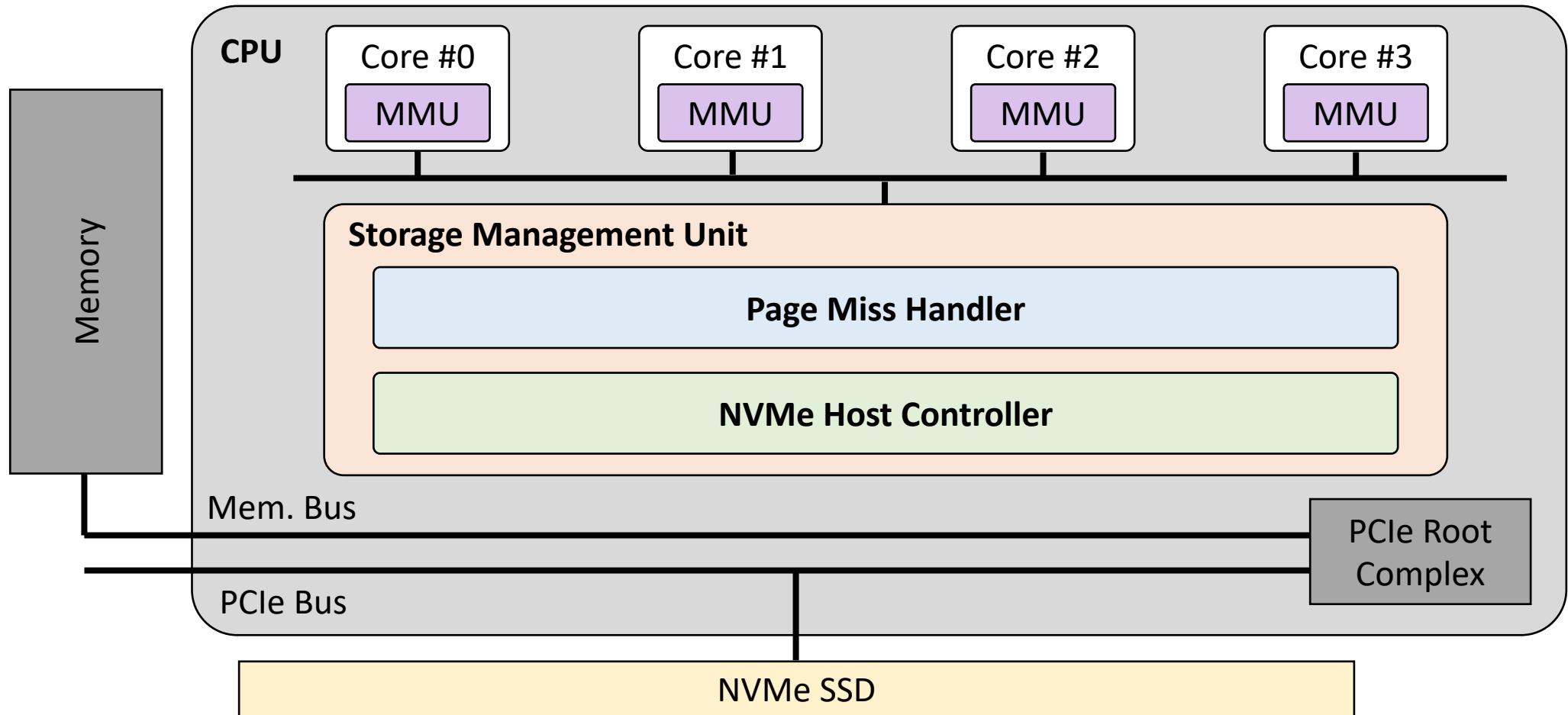


HWDP with LBA-augmented Page Table

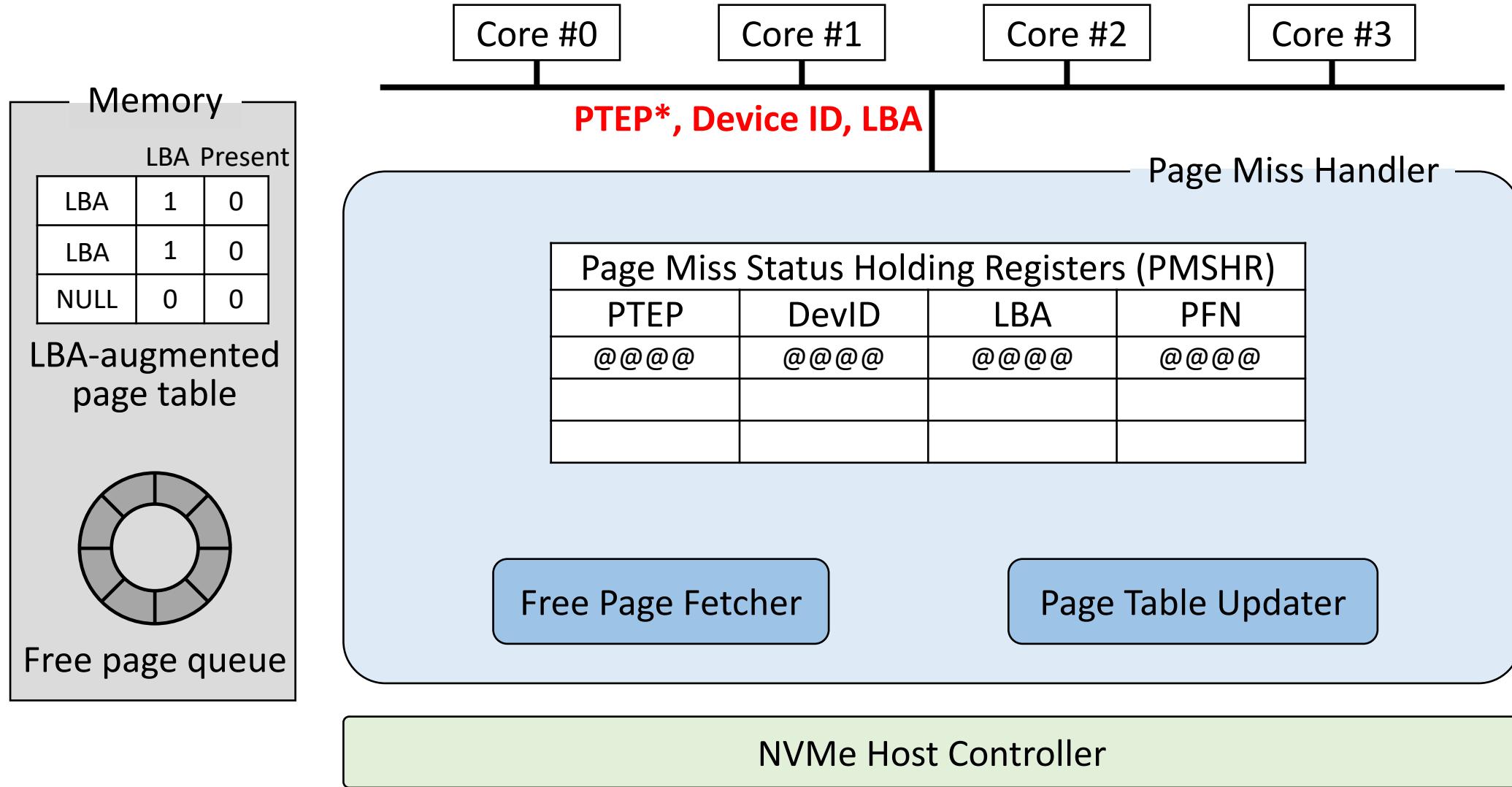


Storage Management Unit (SMU)

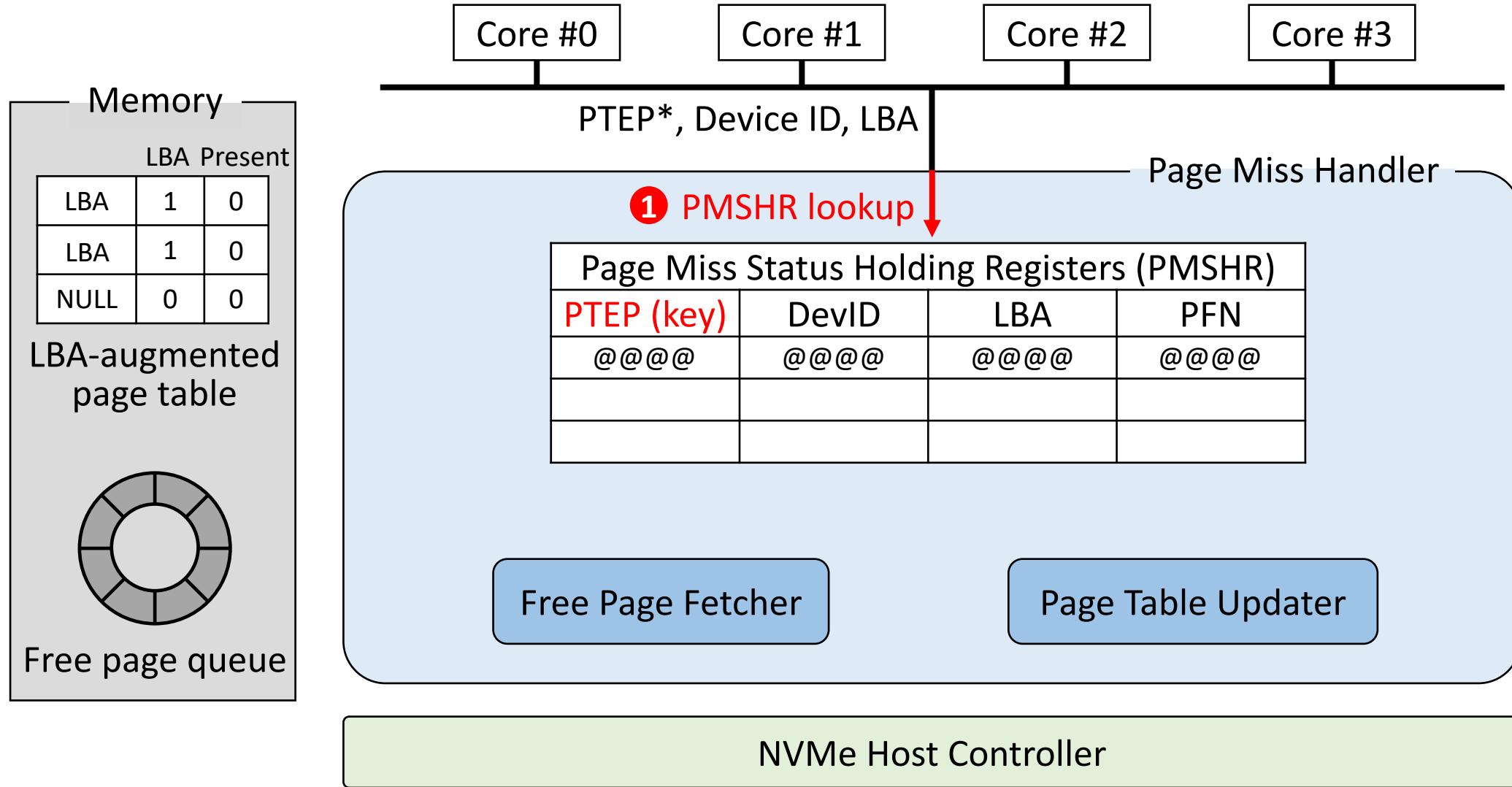
- Additional architectural component to handle page miss in hardware



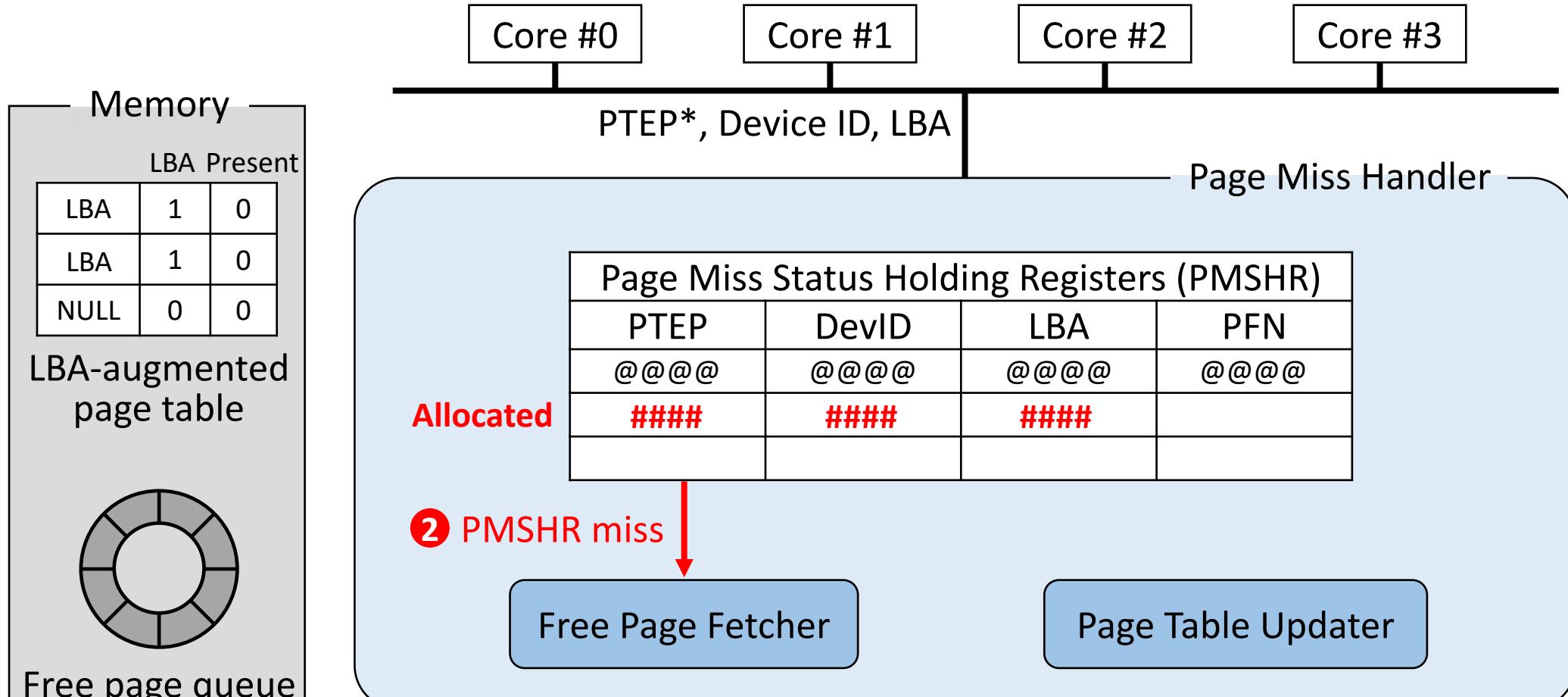
SMU - Page Miss Handler



SMU - Page Miss Handler



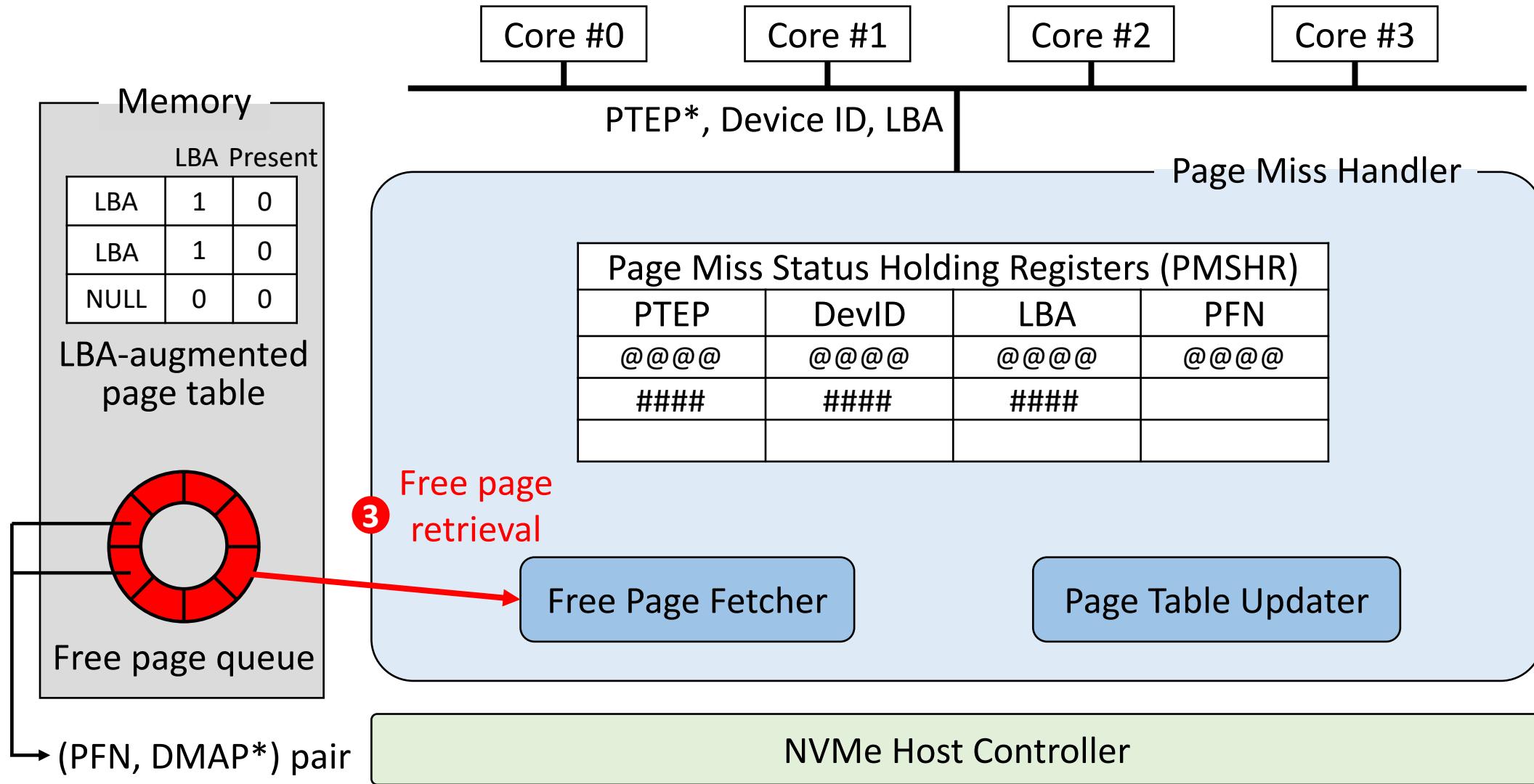
SMU - Page Miss Handler



NVMe Host Controller

*PTEP: PTE address

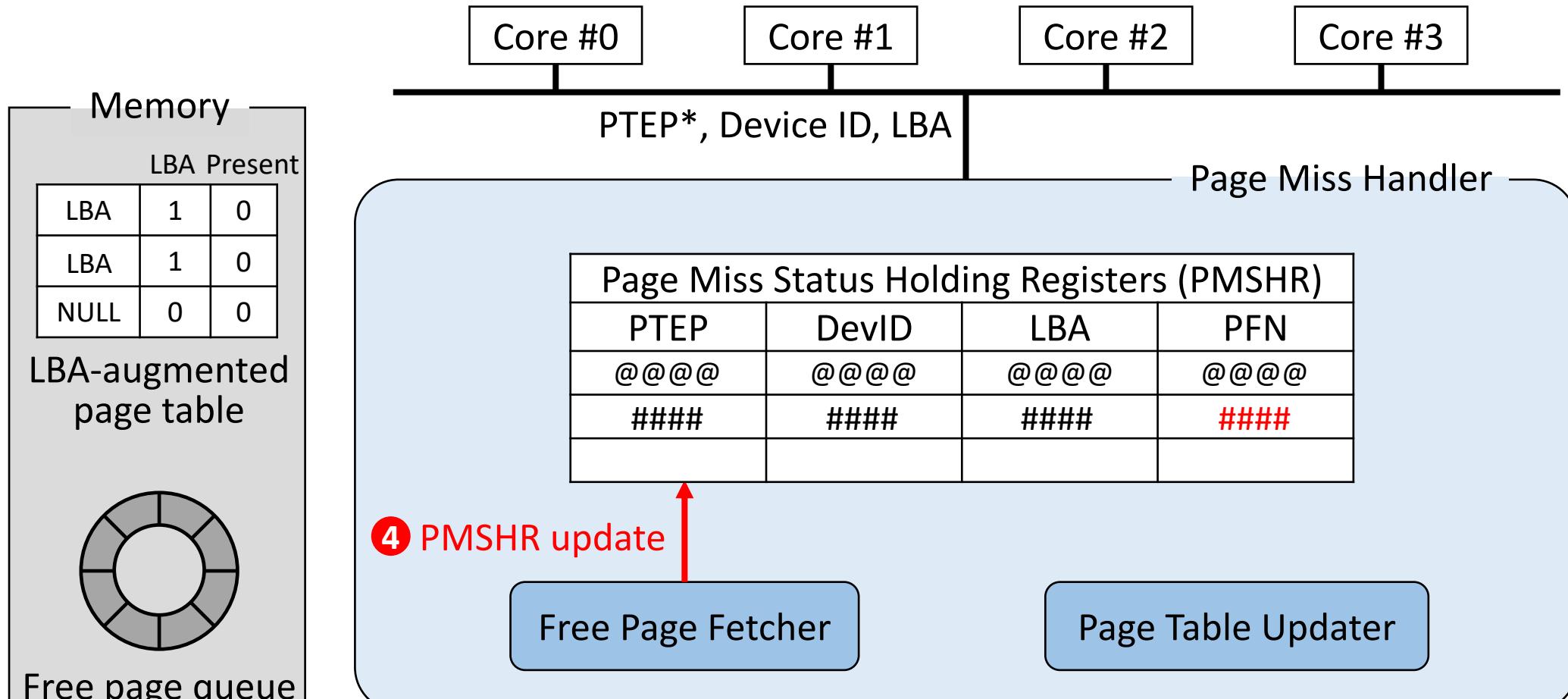
SMU - Page Miss Handler



*PTEP: PTE address

*DMAP: DMA address

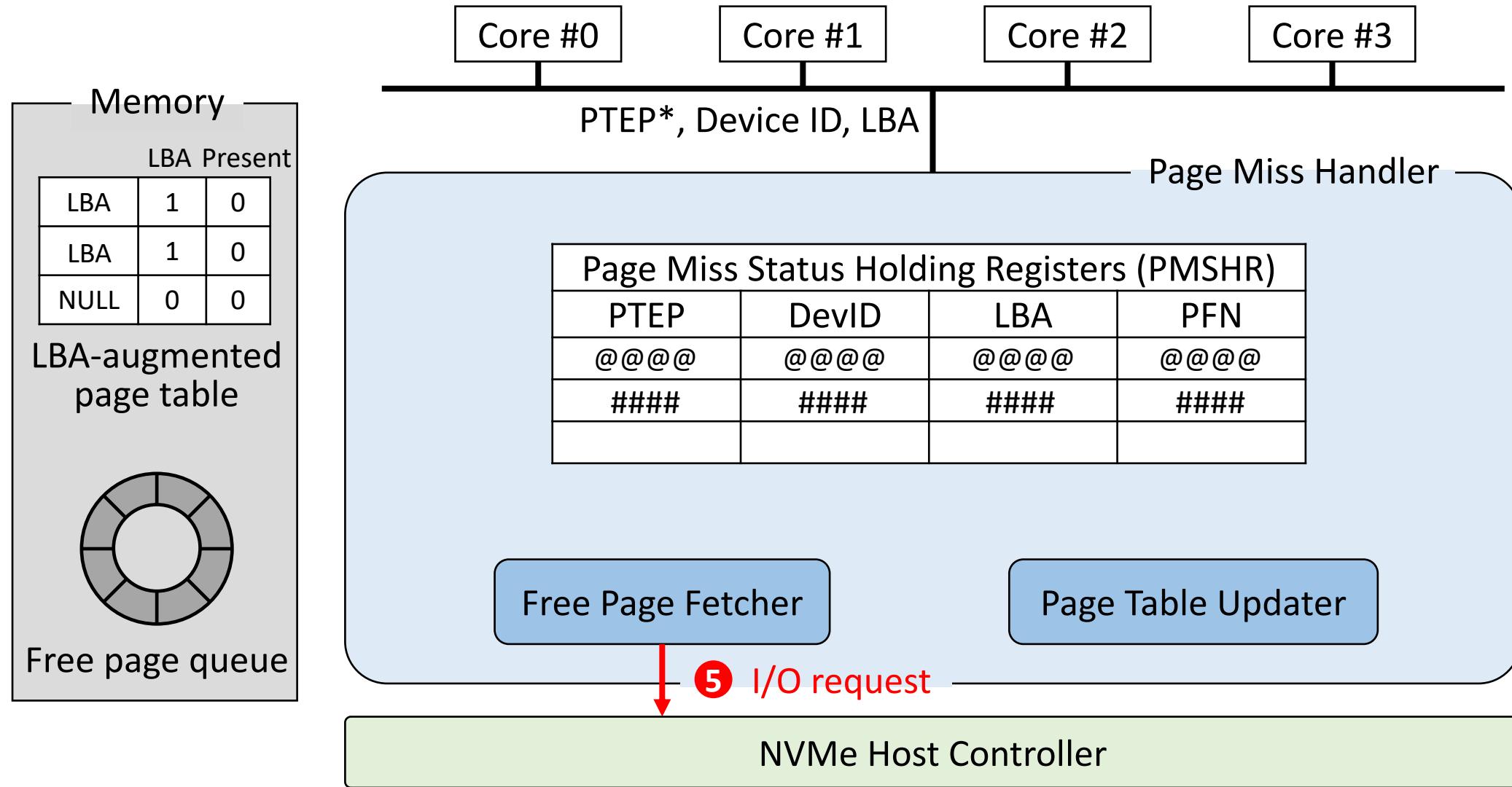
SMU - Page Miss Handler



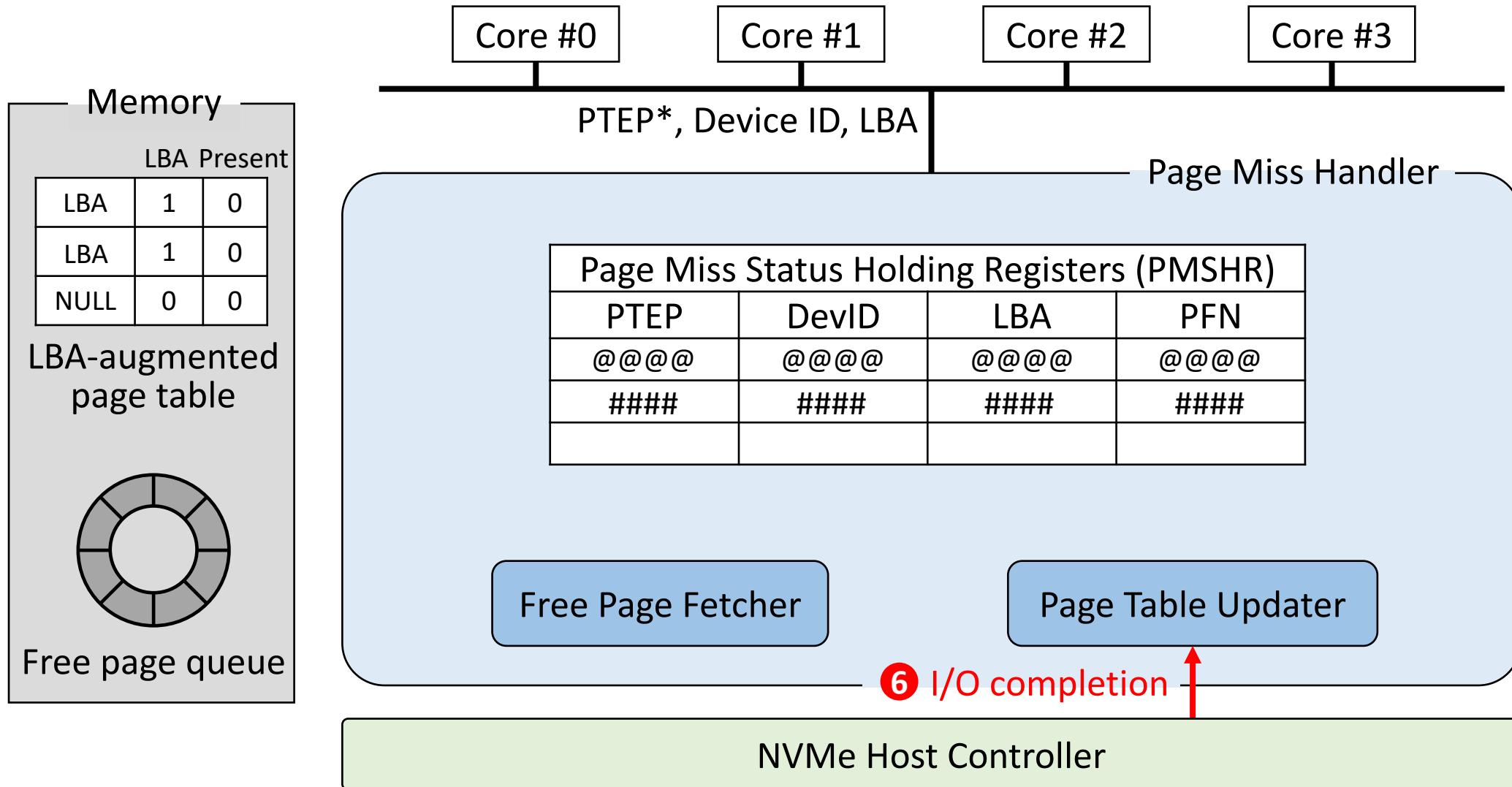
NVMe Host Controller

*PTEP: PTE address

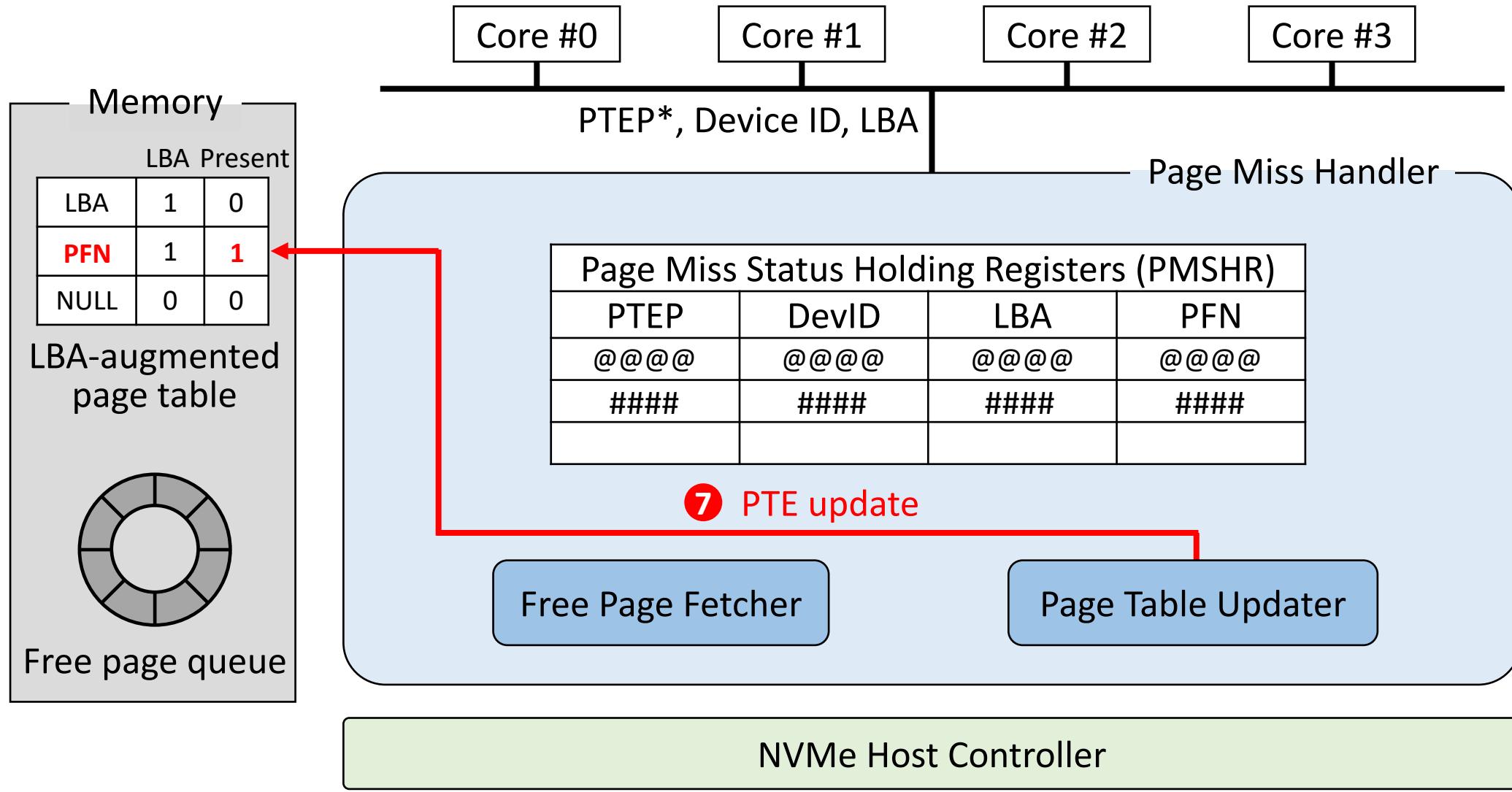
SMU - Page Miss Handler



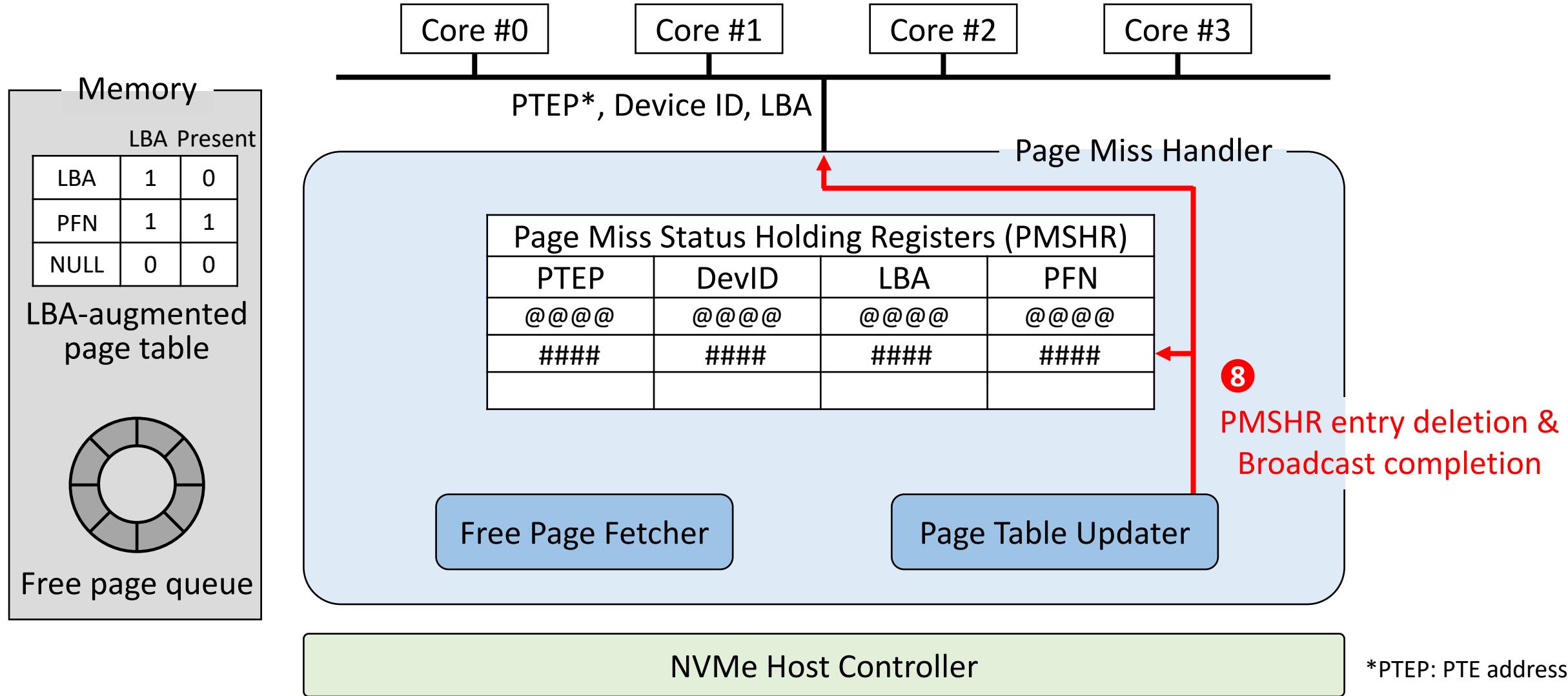
SMU - Page Miss Handler



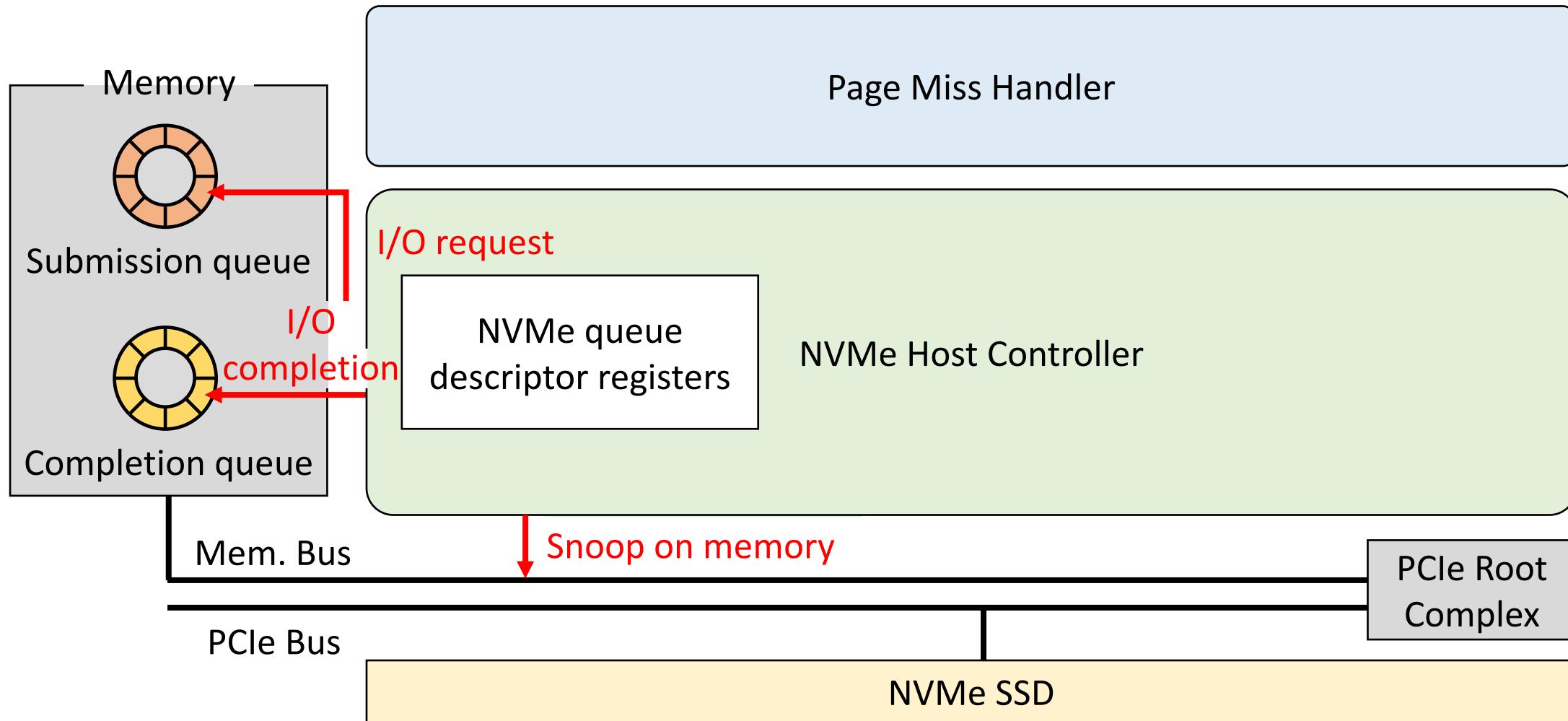
SMU - Page Miss Handler



SMU - Page Miss Handler



SMU - NVMe Host Controller

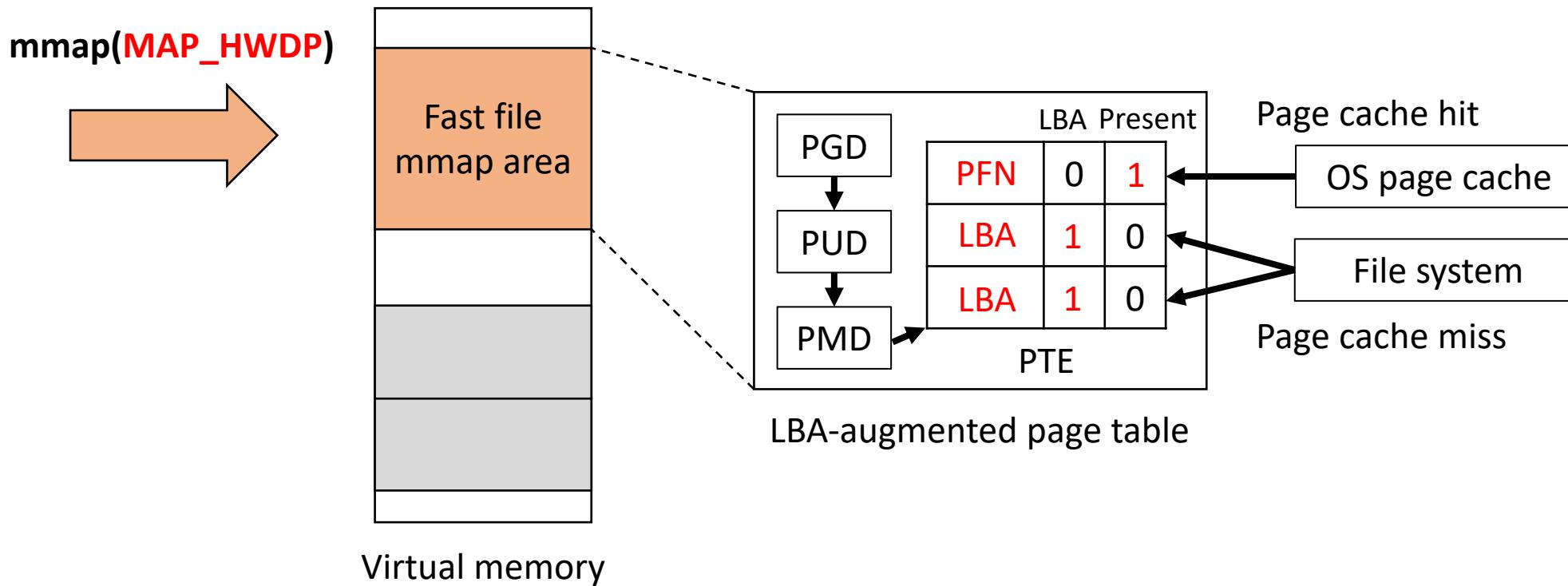


Outline

- Overview
- Architectural extensions
 - LBA-augmented page table
 - Storage Management Unit (SMU)
- OS supports
 - Fast file mmap()
 - Updating OS metadata for HWDP
 - Management of free page queue
- Evaluation
- Conclusion

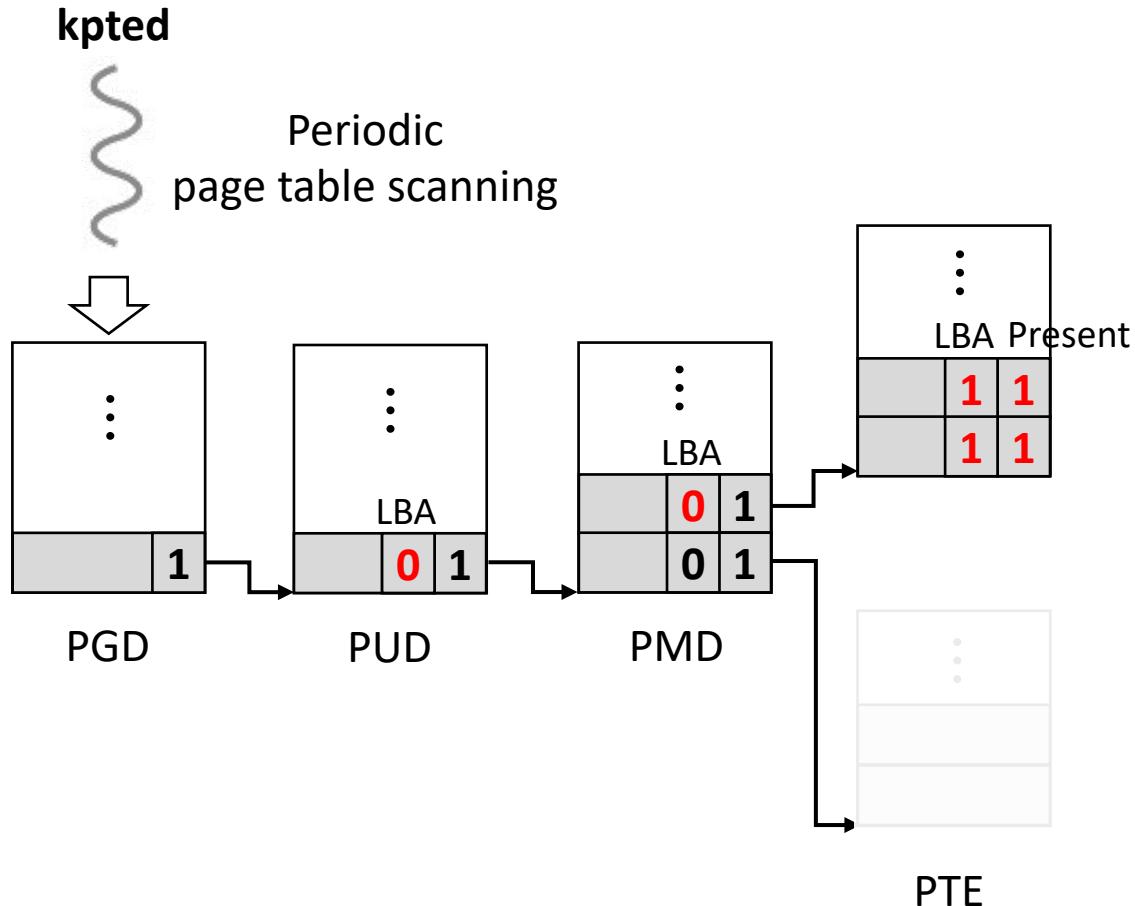
Fast File mmap()

- A new interface to utilize HWDP
 - Allocate a region of memory-mapped file to use HWDP
 - Update associated metadata lazily
 - Currently, only for files that are not shared across multiple processes



Updating OS Metadata for HWDP

- Introduces a kernel thread (*kpted*) to update OS-managed metadata in background



```
if (PUD→LBA bit) {  
    Clears PUD→LBA bit;  
} Goes to next-level (PMD);  
}
```

```
if (PMD→LBA bit) {  
    Clears PMD→LBA bit;  
} Goes to next-level (PTE);  
}
```

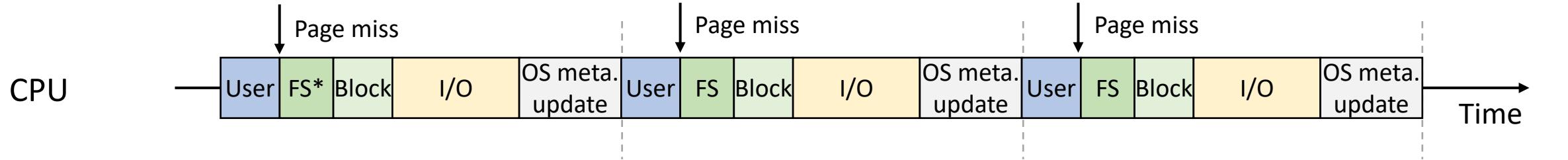
```
if (PTE→LBA bit && PTE→present bit) {  
    Updates OS metadata; (inserts page to LRU list ..)  
    Clears PTE→LBA bit;  
}
```

Management of Free Page Queue

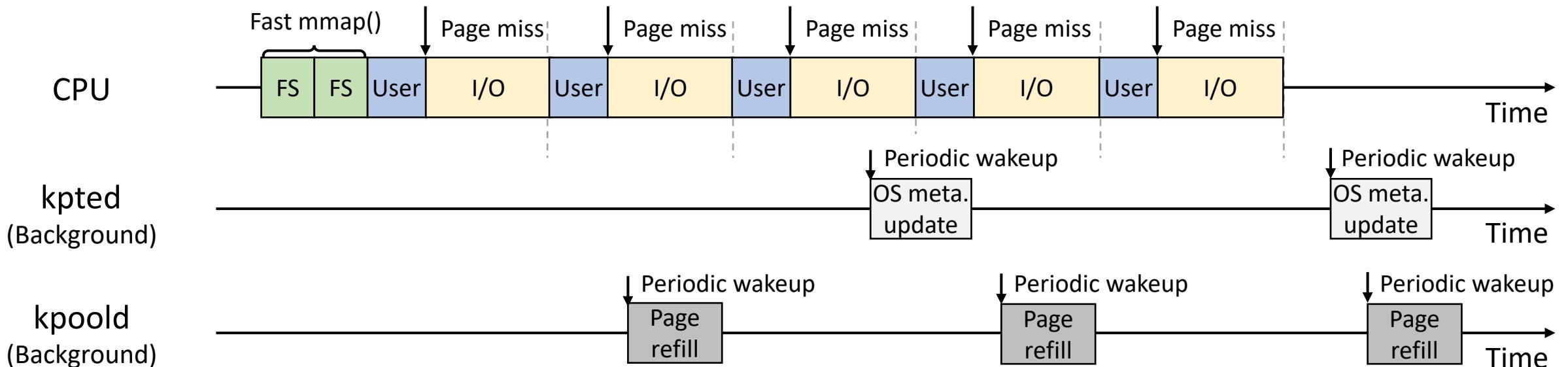
- Synchronous free page refill
 - SMU raises a page fault exception when it fails to allocate a free page
 - OS page fault handler handles the page fault and refills the free page queue
- Asynchronous free page refill
 - A background kernel thread (*kpoold*) periodically refills free pages
 - 44-78% reduction of synchronous page refills (in YCSB workloads)

Summary - Demand Paging Comparison

OS-based Demand Paging (OSDP)



Hardware-based Demand Paging (HWDP)



Outline

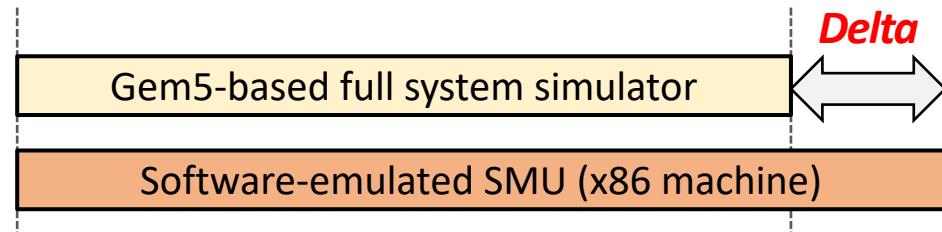
- Overview
- Architectural extensions
 - LBA-augmented page table
 - Storage Management Unit (SMU)
- OS supports
 - Fast file mmap()
 - Updating OS metadata for HWDP
 - Management of free page queue
- Evaluation
- Conclusion

Evaluation

■ Experimental Setup

- Micro-architecture-level evaluation
 - Gem5-based full system simulator integrated with the SSD model
- End-to-end system level evaluation
 - Implementing HWDP on real x86 machine (software-emulated SMU)

■ Delta measurement of single page miss handling



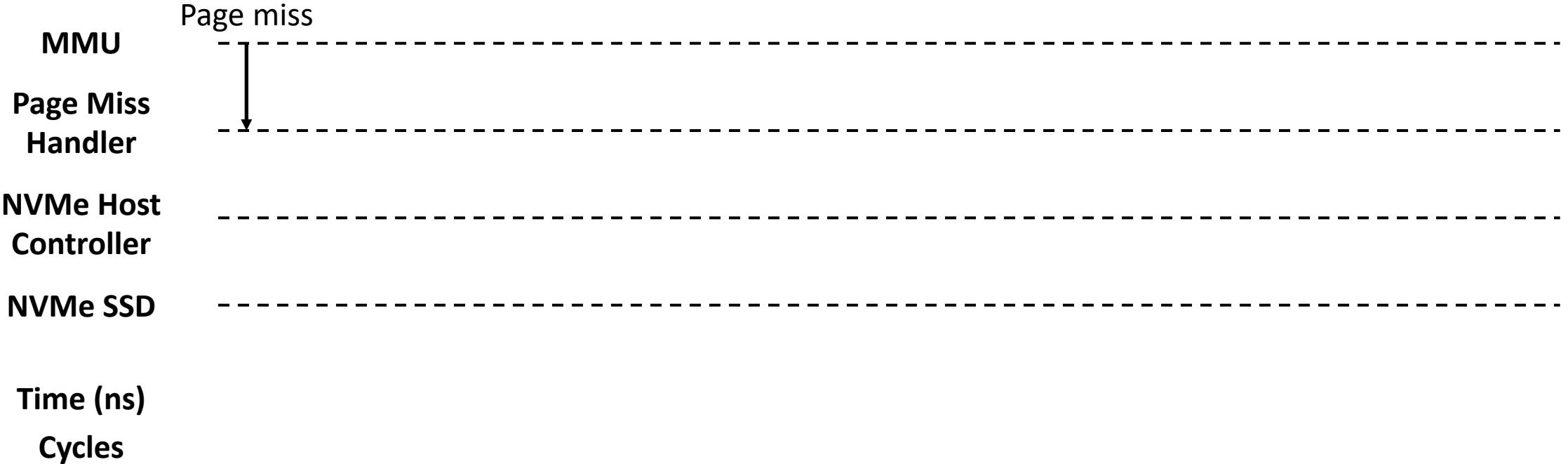
■ HWDP performance estimation (Ops/sec)

$$\frac{\text{\# of operations}}{\text{\# of workload}} \times (\text{Execution time} - \text{\# of page misses} * \Delta)$$

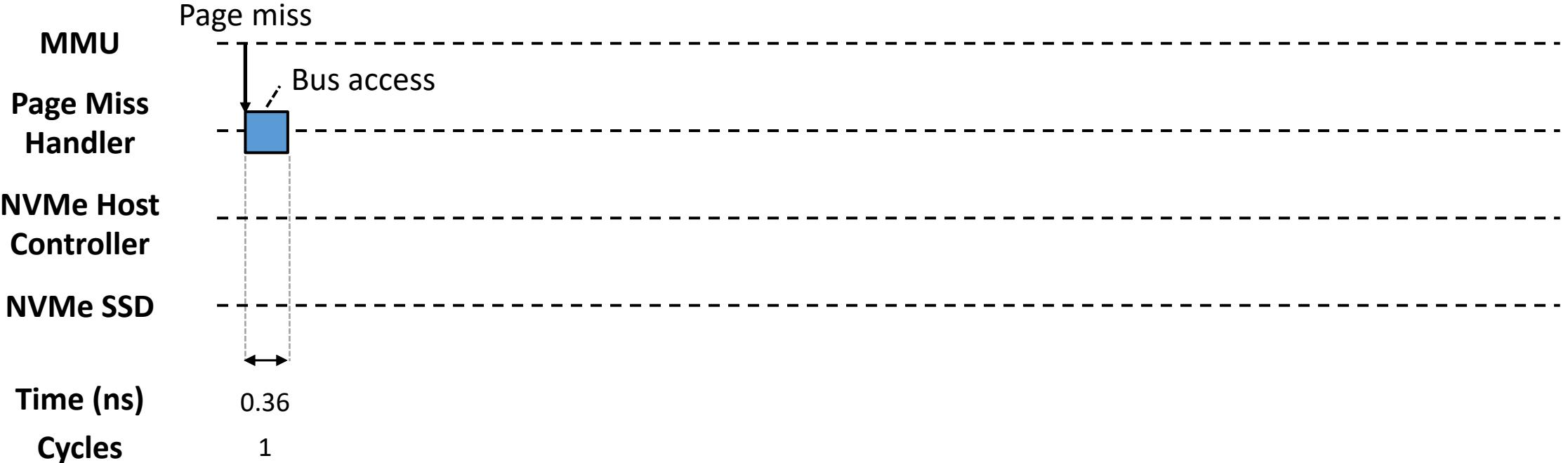
■ x86 machine configuration

Server	Dell R730
OS	Ubuntu 16.04.6
Base kernel	Linux 4.9.30
CPU	Intel Xeon E5-2640v3 2.6GHz 8 physical cores (Hyperthreading on)
Memory	DDR4 32GB
Storage devices	Samsung SZ985 800GB Z-SSD

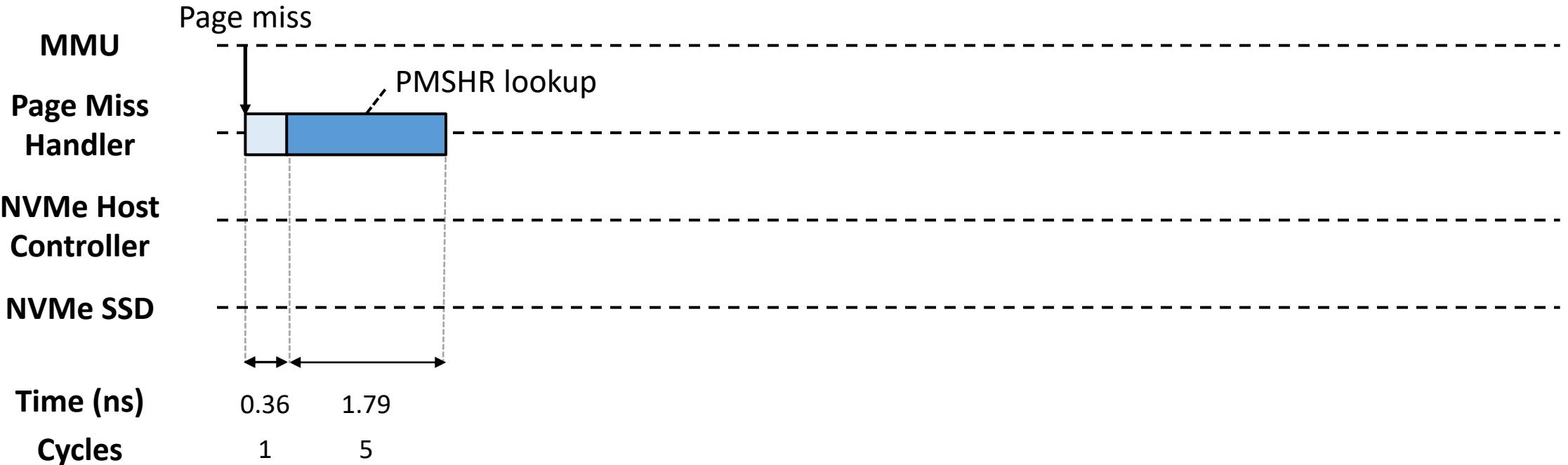
Page Miss Analysis



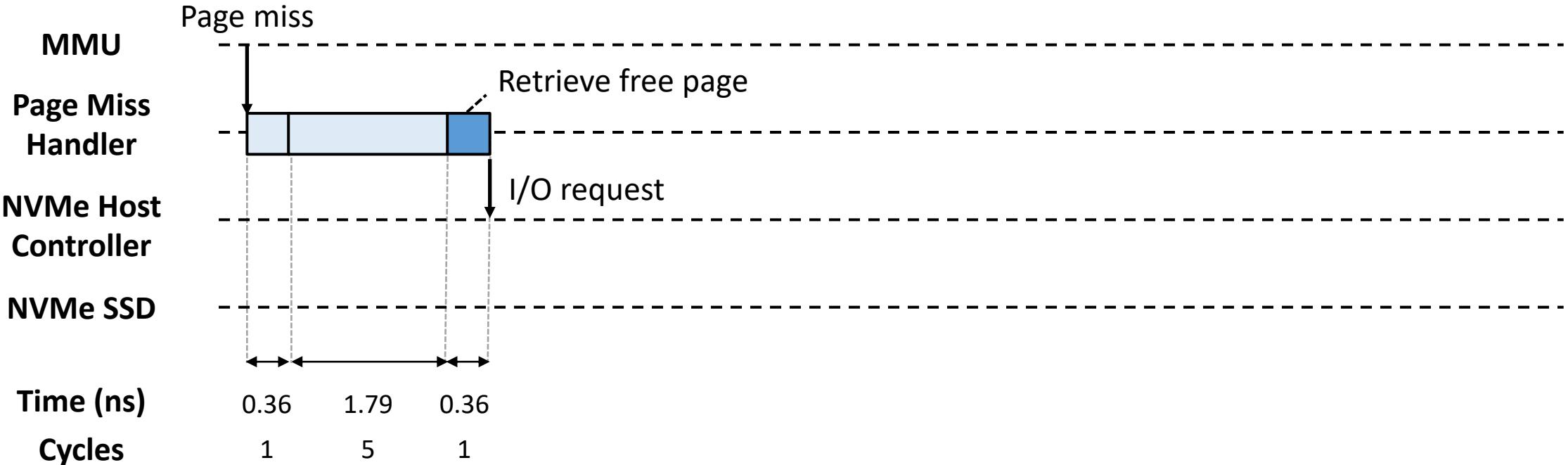
Page Miss Analysis



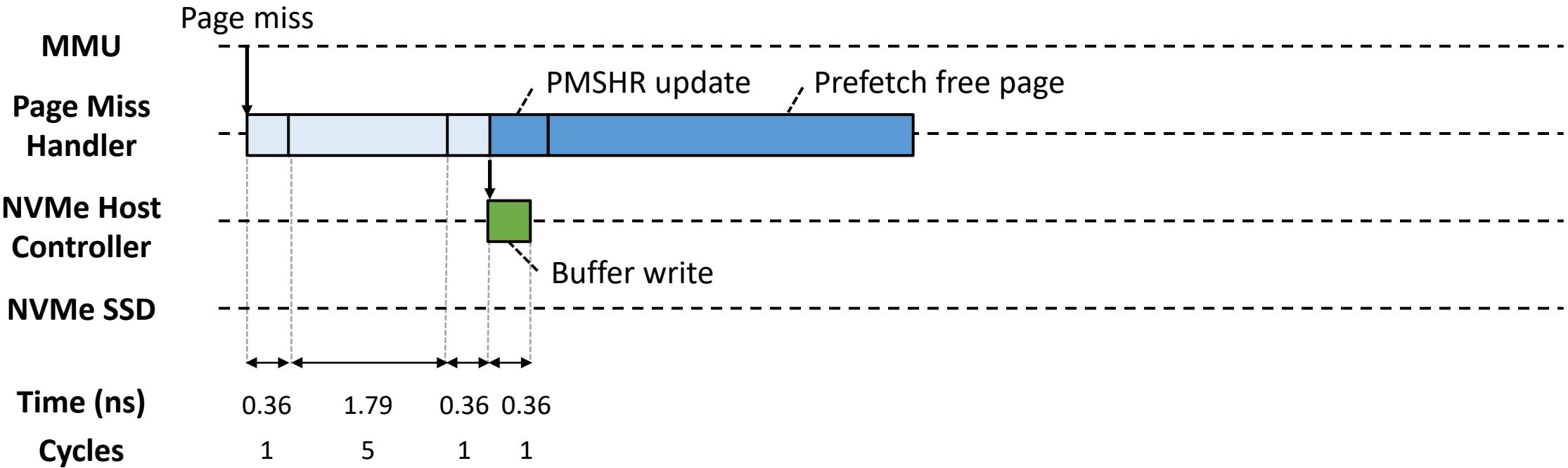
Page Miss Analysis



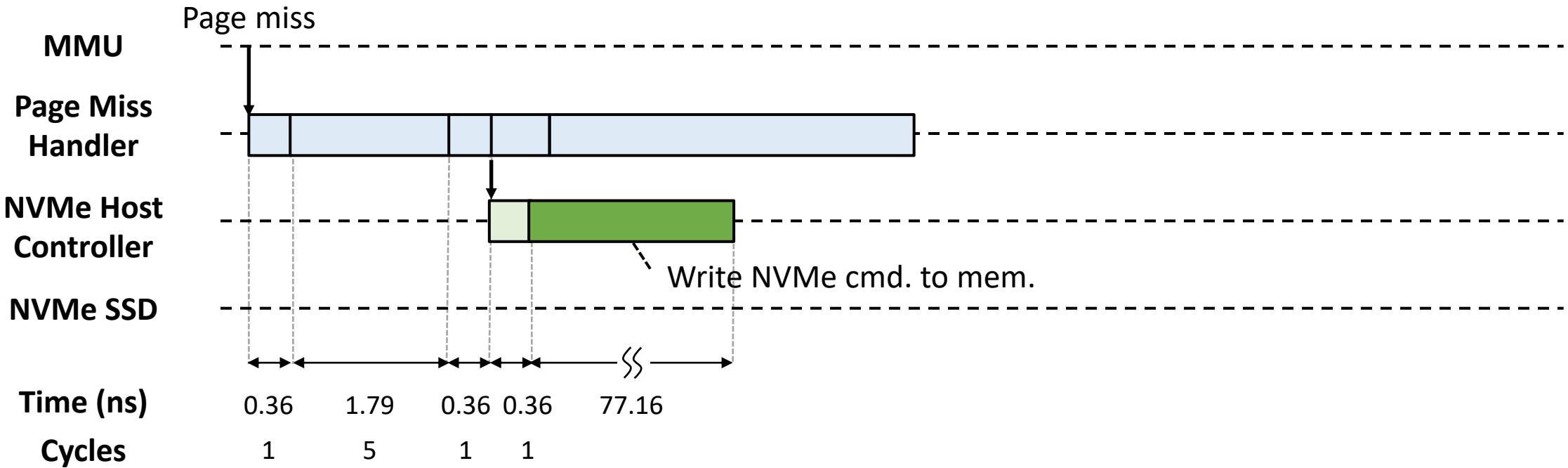
Page Miss Analysis



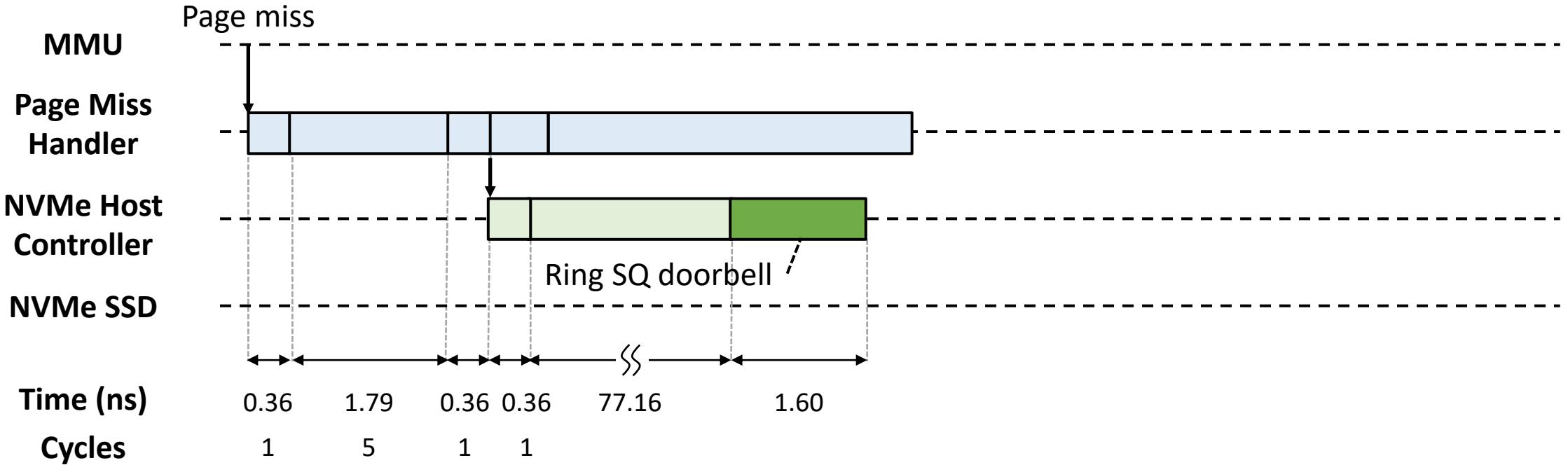
Page Miss Analysis



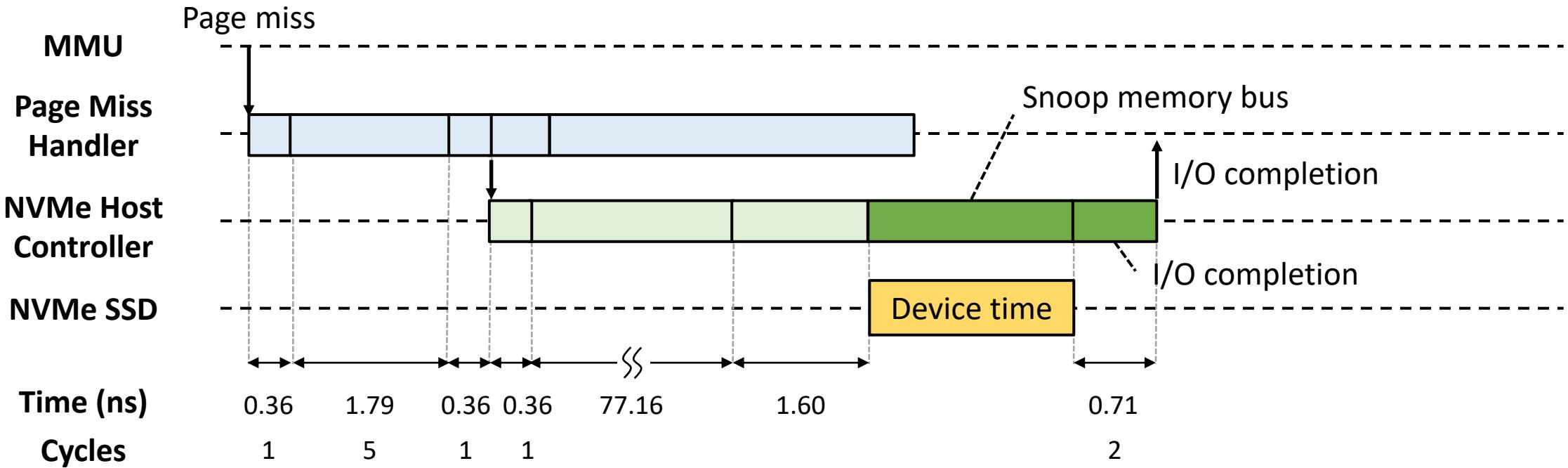
Page Miss Analysis



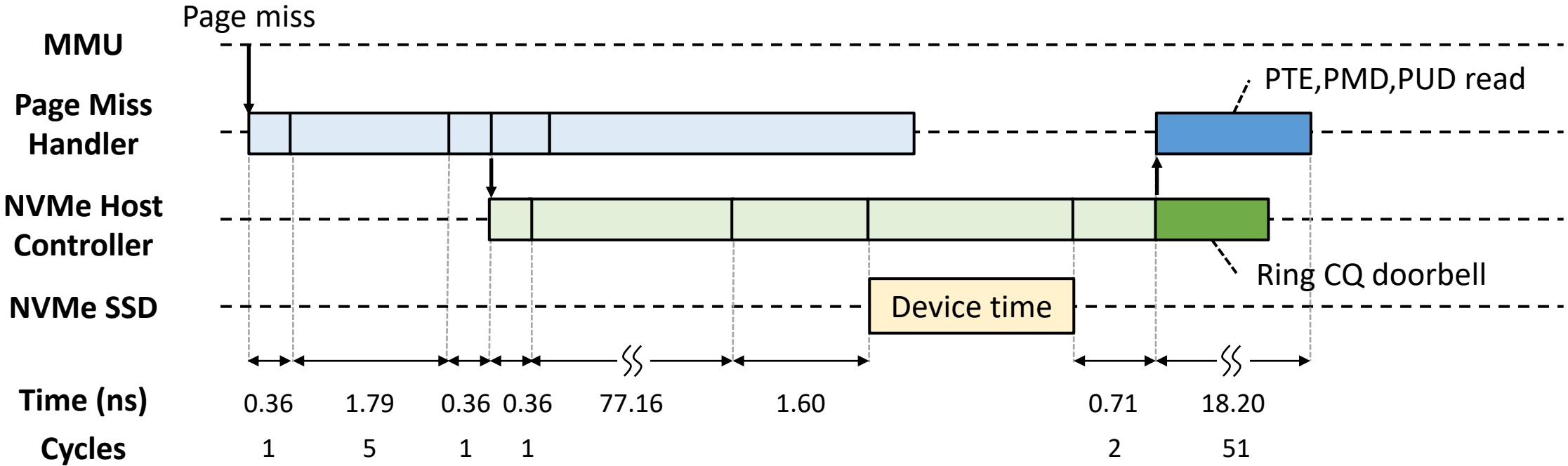
Page Miss Analysis



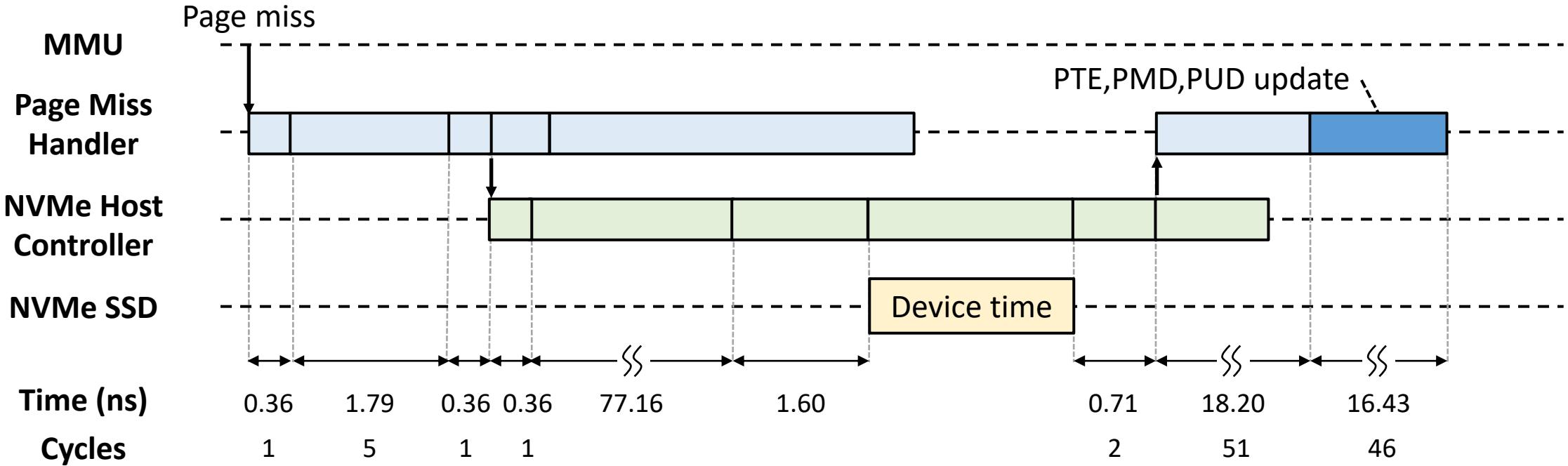
Page Miss Analysis



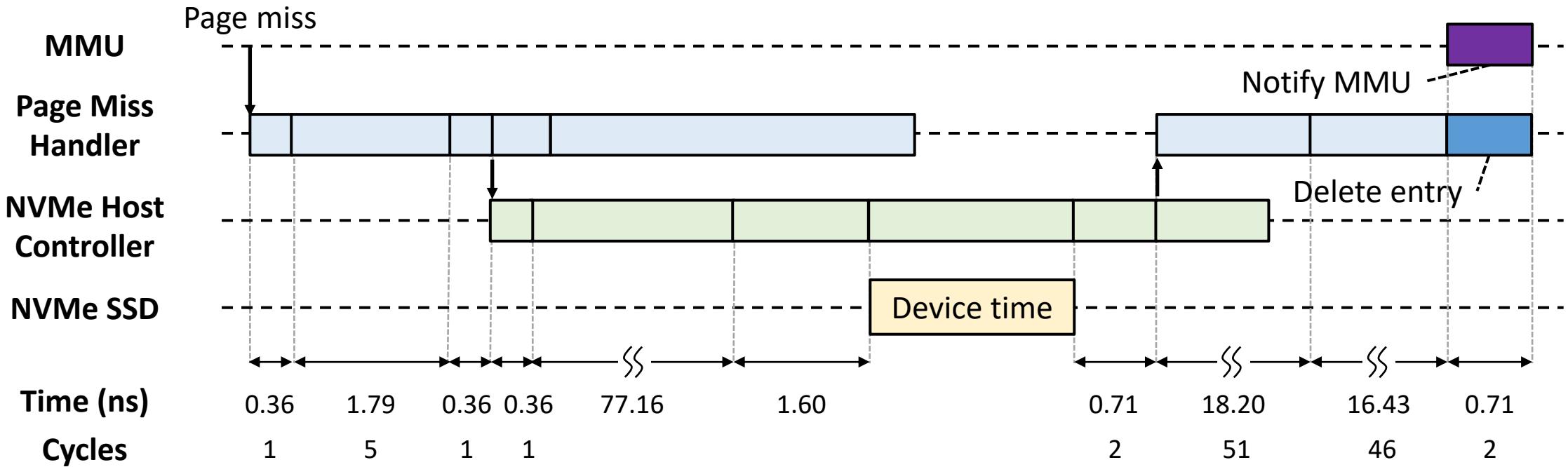
Page Miss Analysis



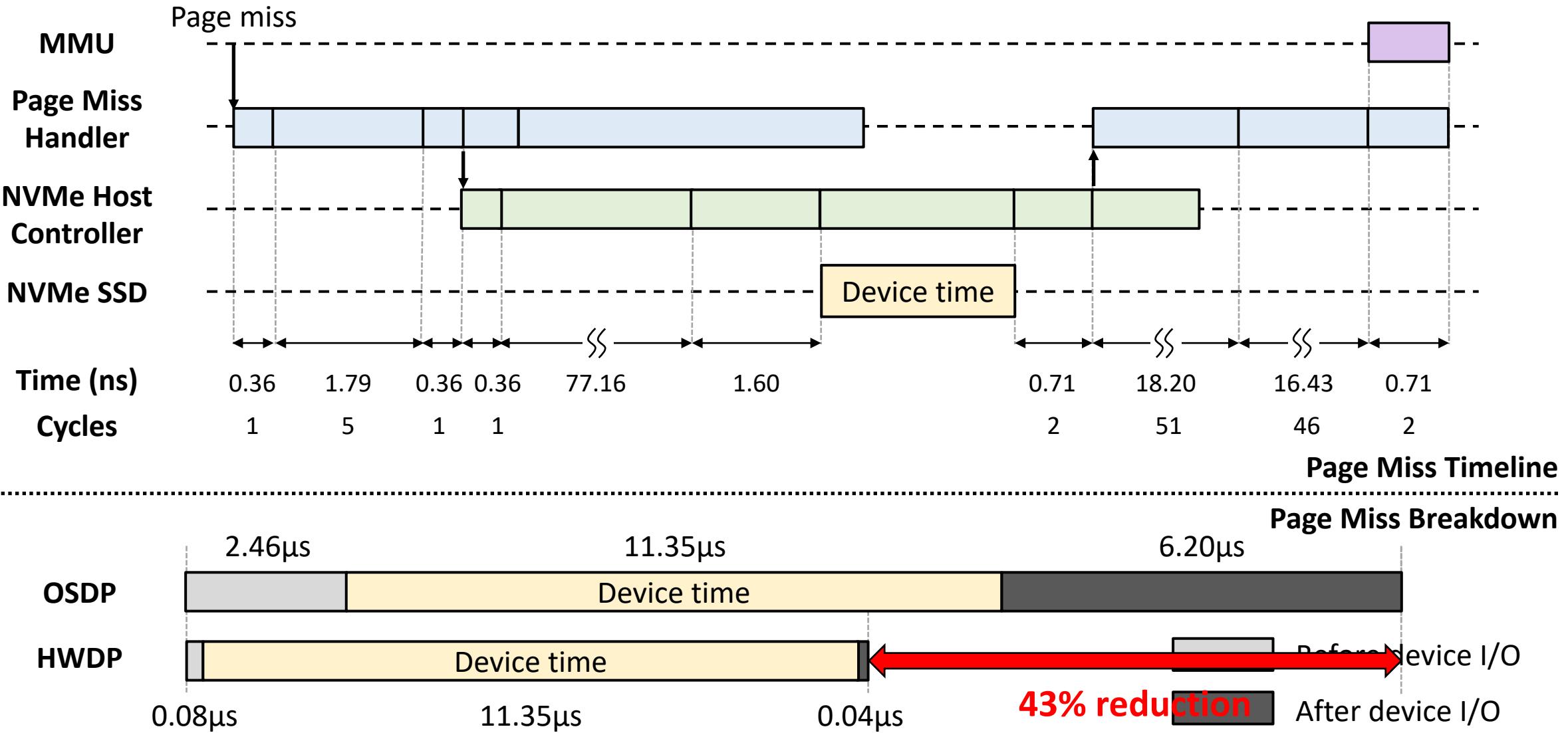
Page Miss Analysis



Page Miss Analysis

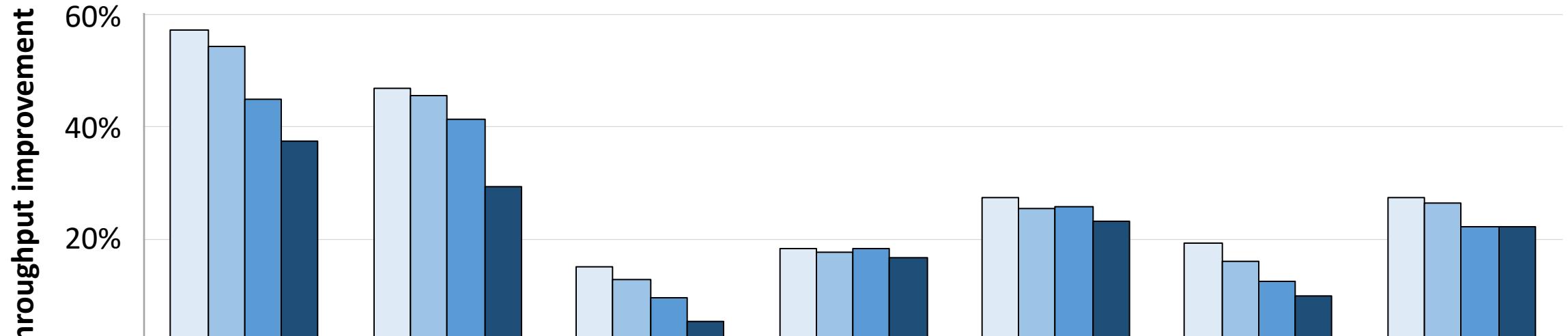


Page Miss Analysis



Application Performance

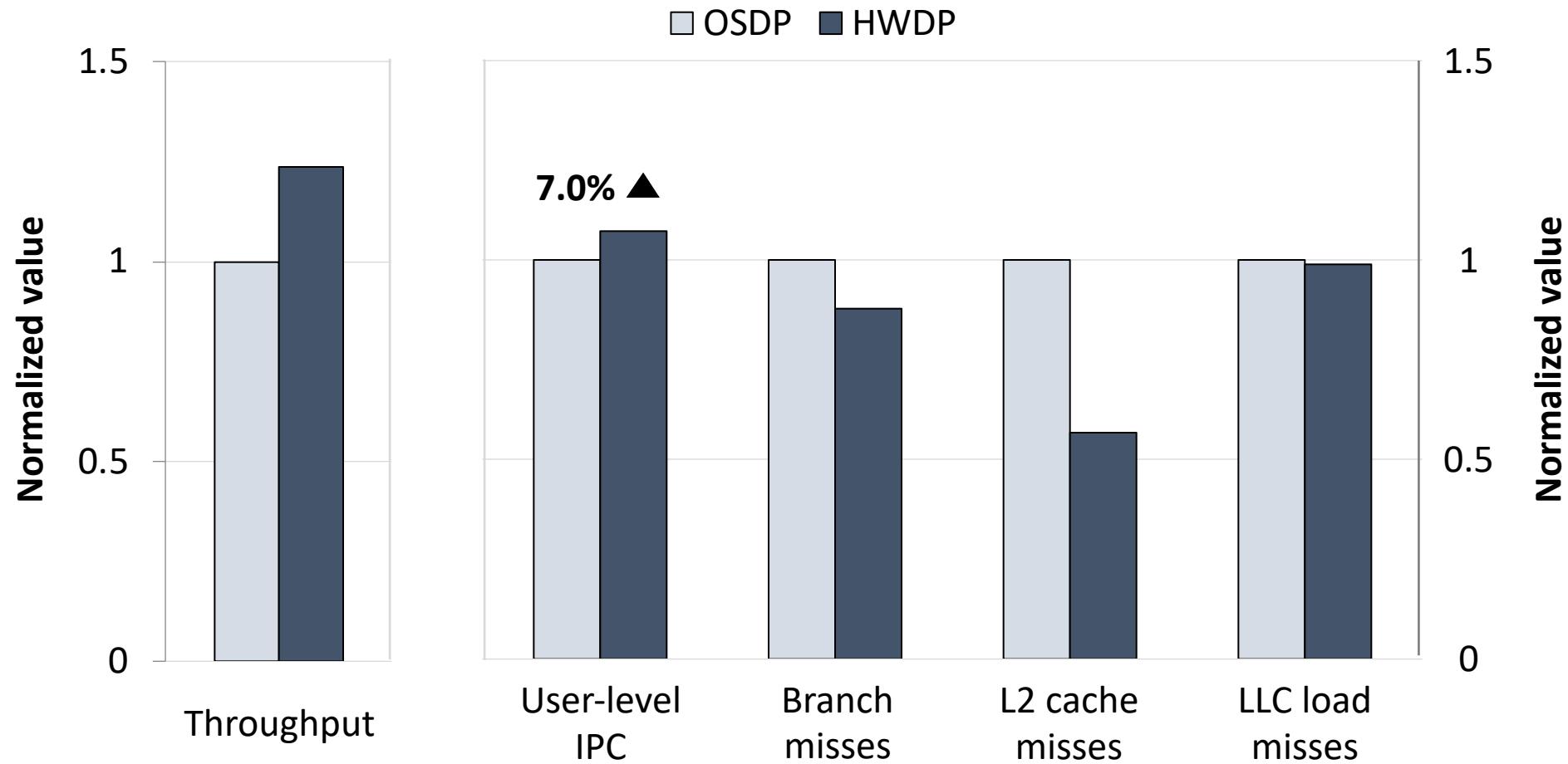
- Result of throughput improvement by HWDP over OSDP



Dataset / Workingset (unit)	64/16 (GB)		64/128 (GB)				
Access pattern	Uniform	Zipfian	Zipfian	Zipfian	Latest	Zipfian	
Read:Write	100:0	100:0	50:50	95:5	100:0	95:5	95:5

Architecture Resource Pollution

- Comparison of micro-architectural events in YCSB-C workload



Conclusion

- Scaling SSD performance demands new directions
- A case for hardware-based demand paging
 - New architectural extensions + OS supports
 - Page misses (mostly) handled in hardware
 - Fast demand paging performance + improved user-level IPC
 - Up to 27% throughput improvement in realistic workload (YCSB-C on RocksDB)

Q&A

- Thank you