# Design and Implementation of Autocaching for CXLSSD

Heekwon Park(heekwon.p@Samsung.com), Jongmin Gim, Jaemin Jung, Adam Manzanares, Davidlohr Bueso, Javier Gonzalez, Yang Seok Ki

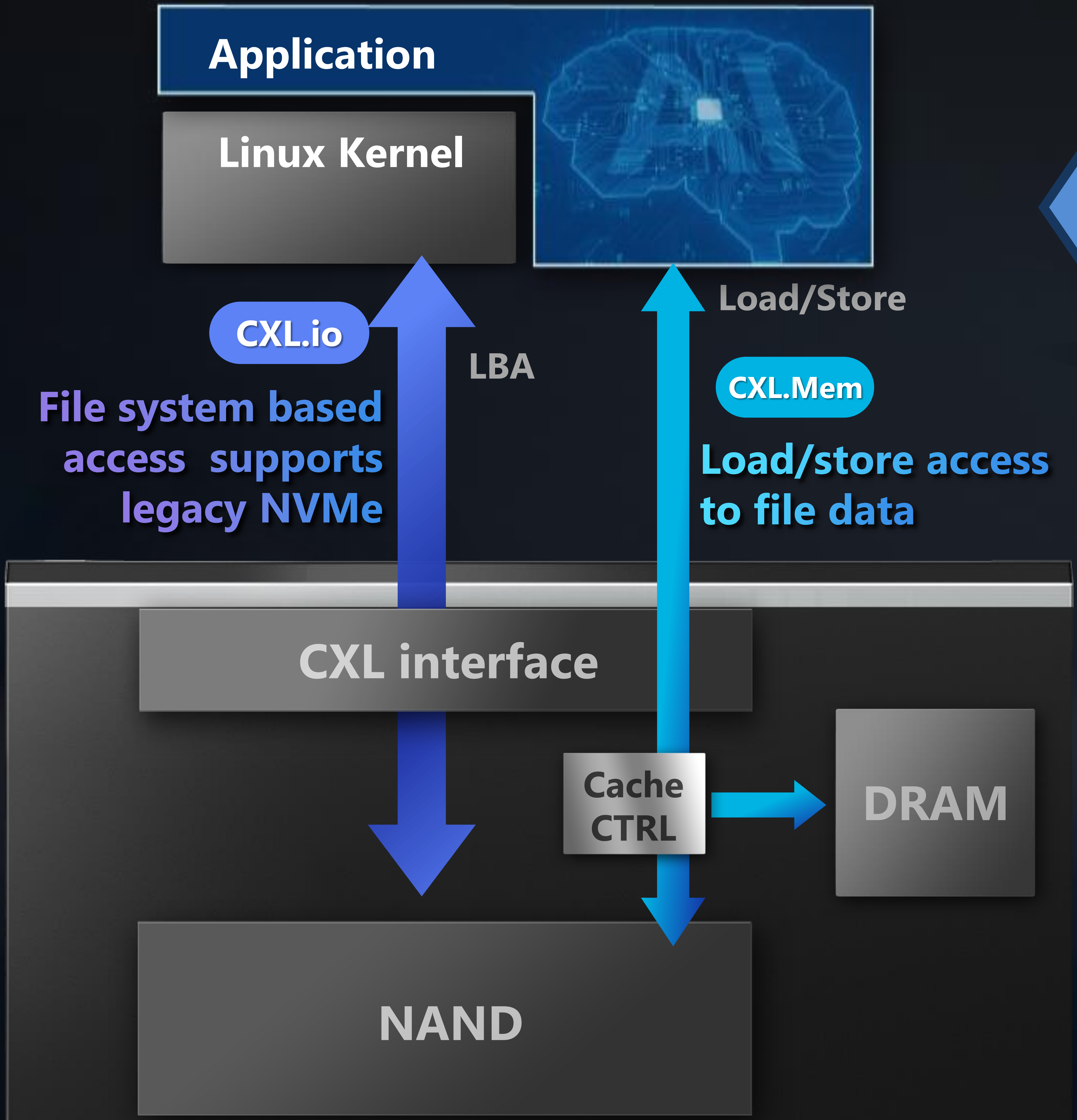
- Samsung Electronics -

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

# Agenda

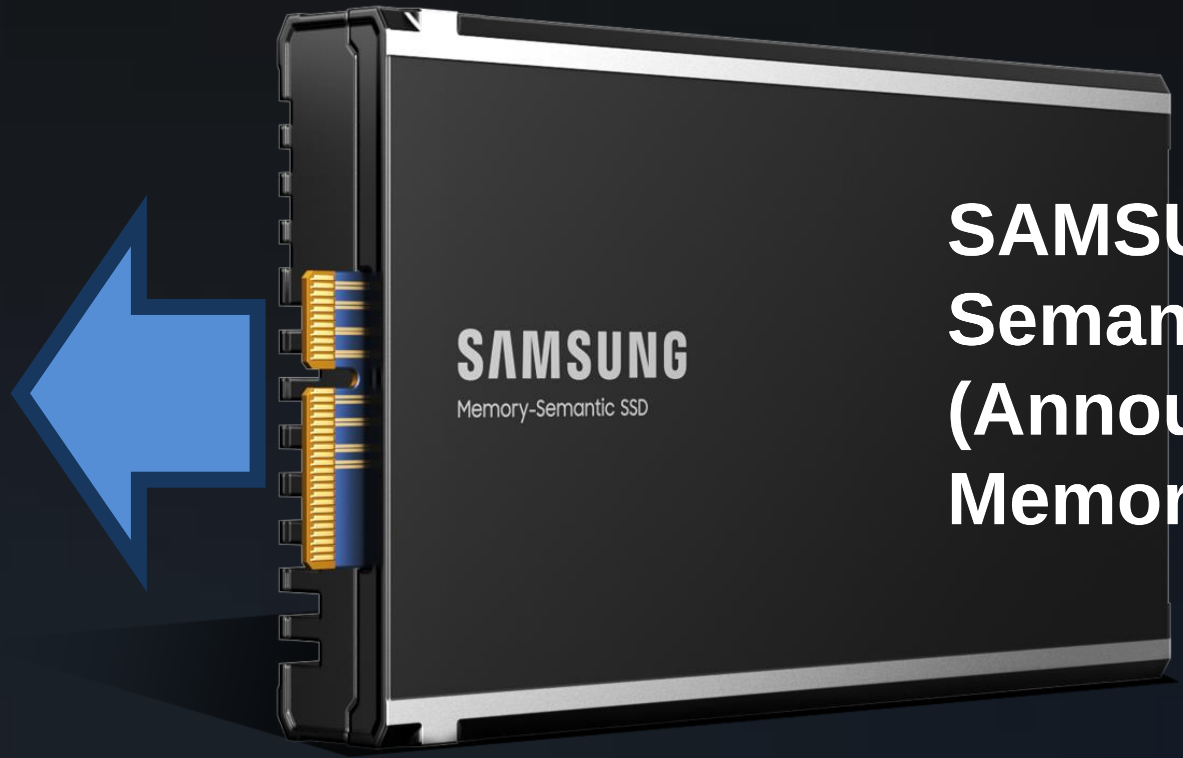
- Motivation
- Design of Autocaching
- Implementation details & Challenges
- Preliminary results
- TO-DO List

# Motivation: The advent of new types of storage device based on CXL



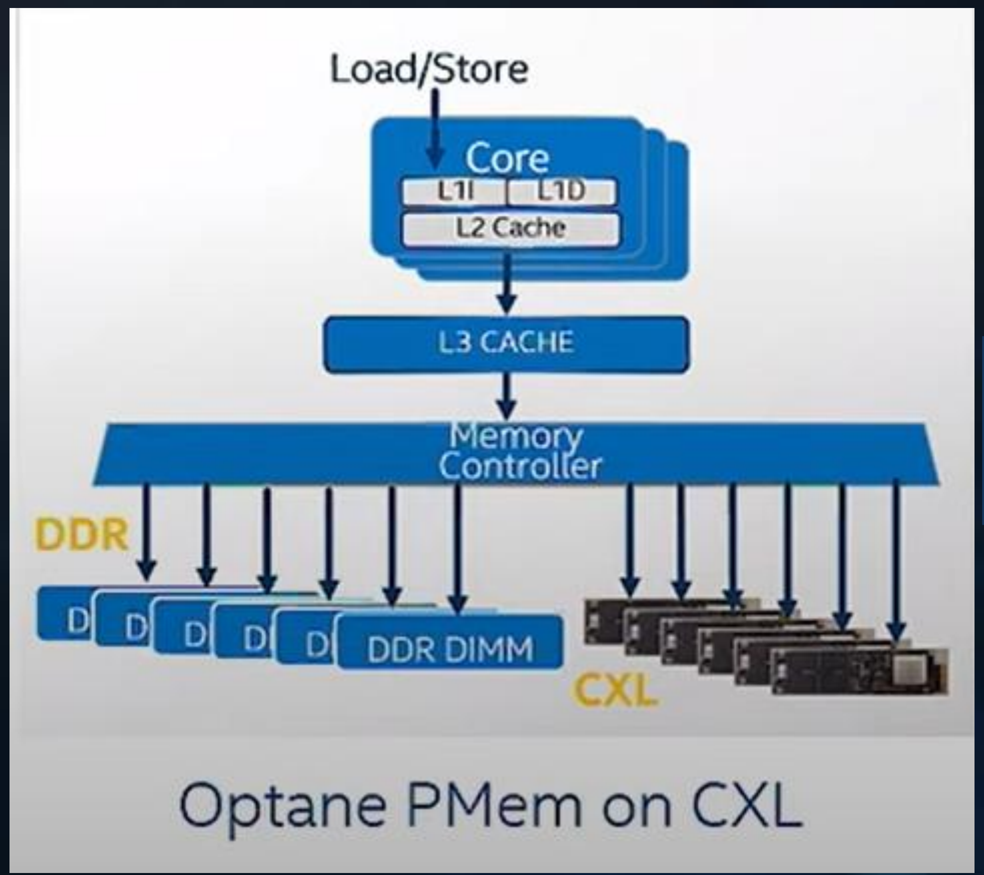
File system based access supports legacy NVMe

Load/store access to file data

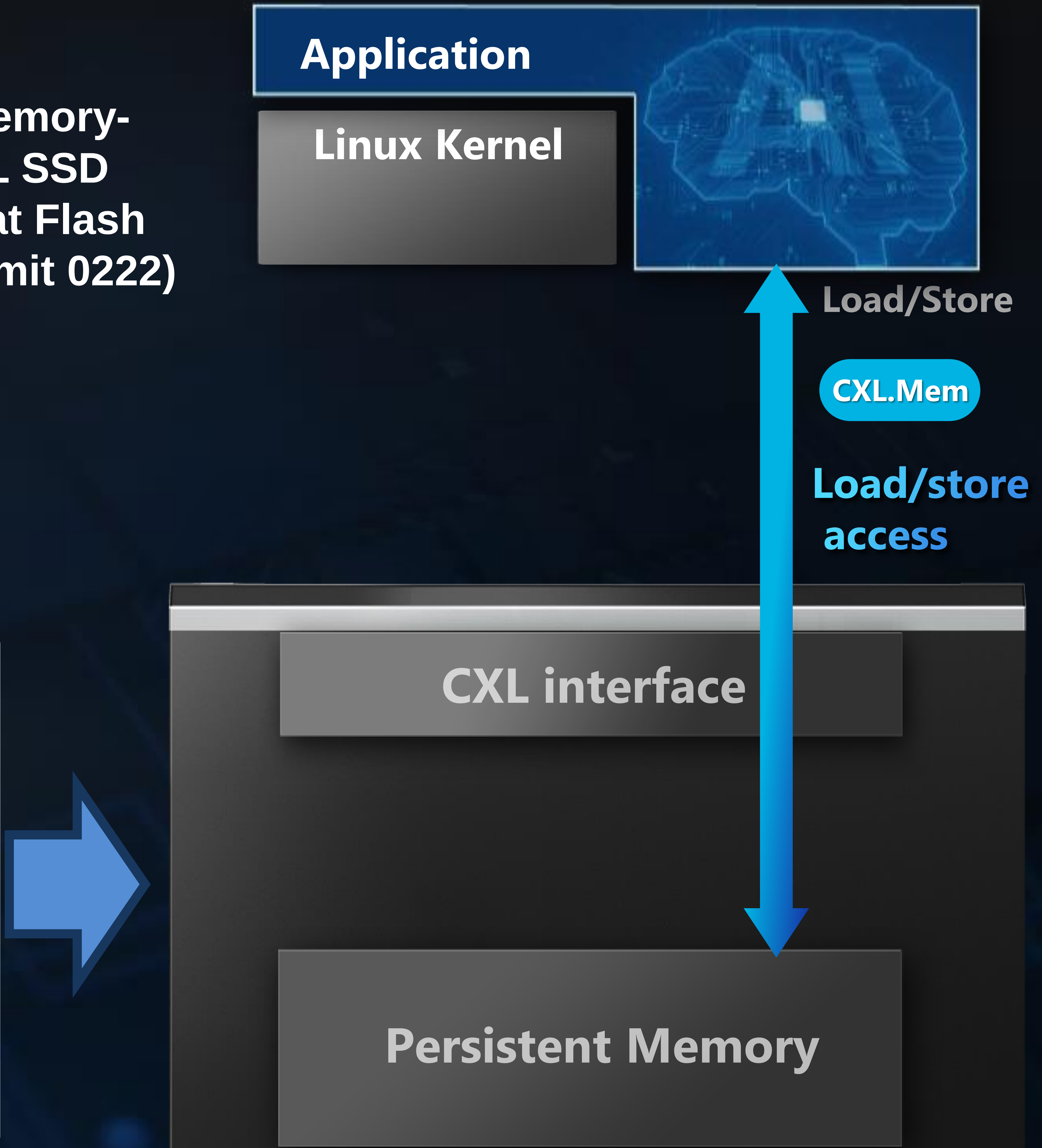


SAMSUNG Memory-Semantic CXL SSD (Announced at Flash Memory Summit 0222)

Intel CXL Optane (Proposed at Intel Tech Field Day 25(04/2022))

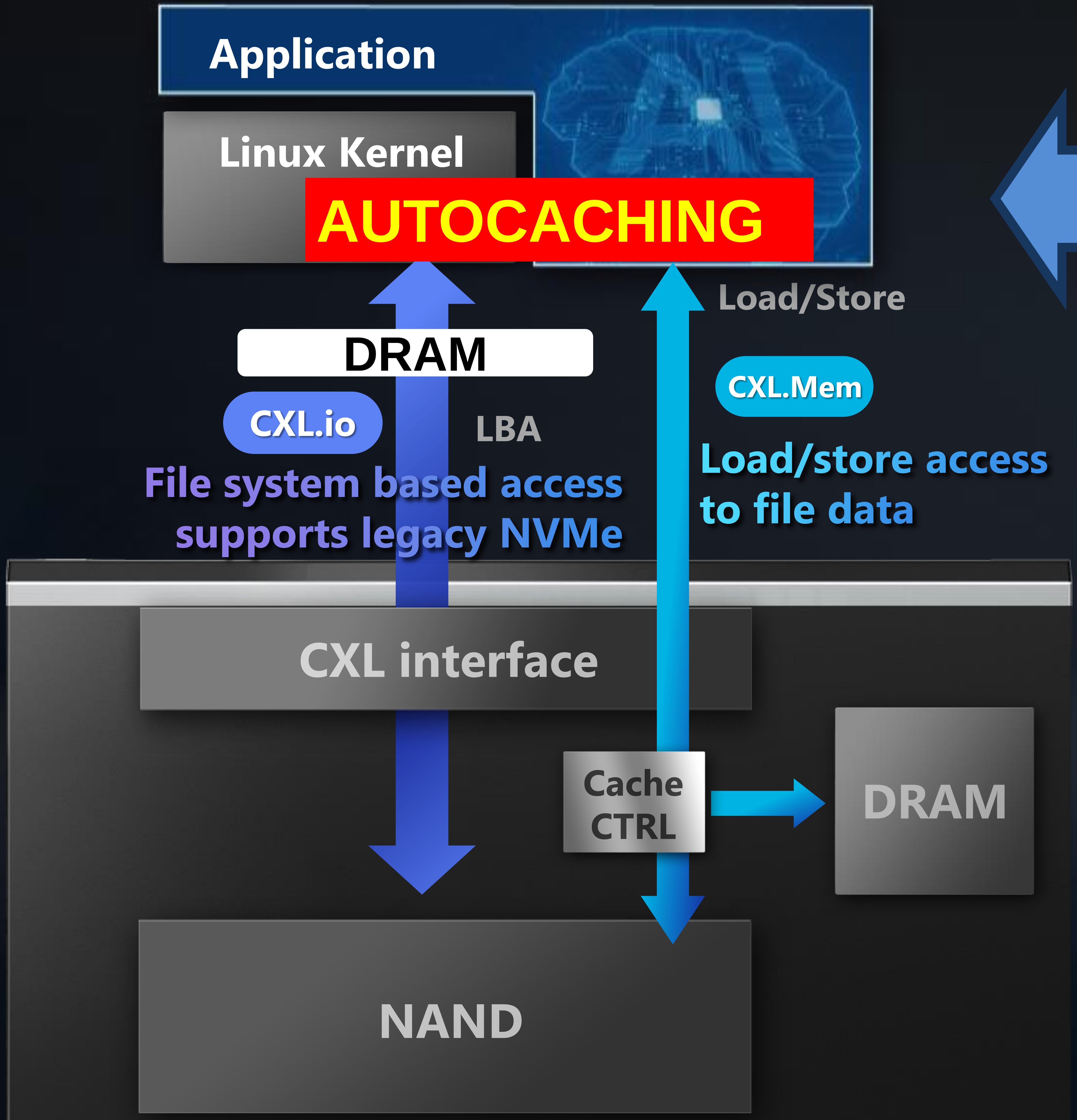


Optane PMem on CXL

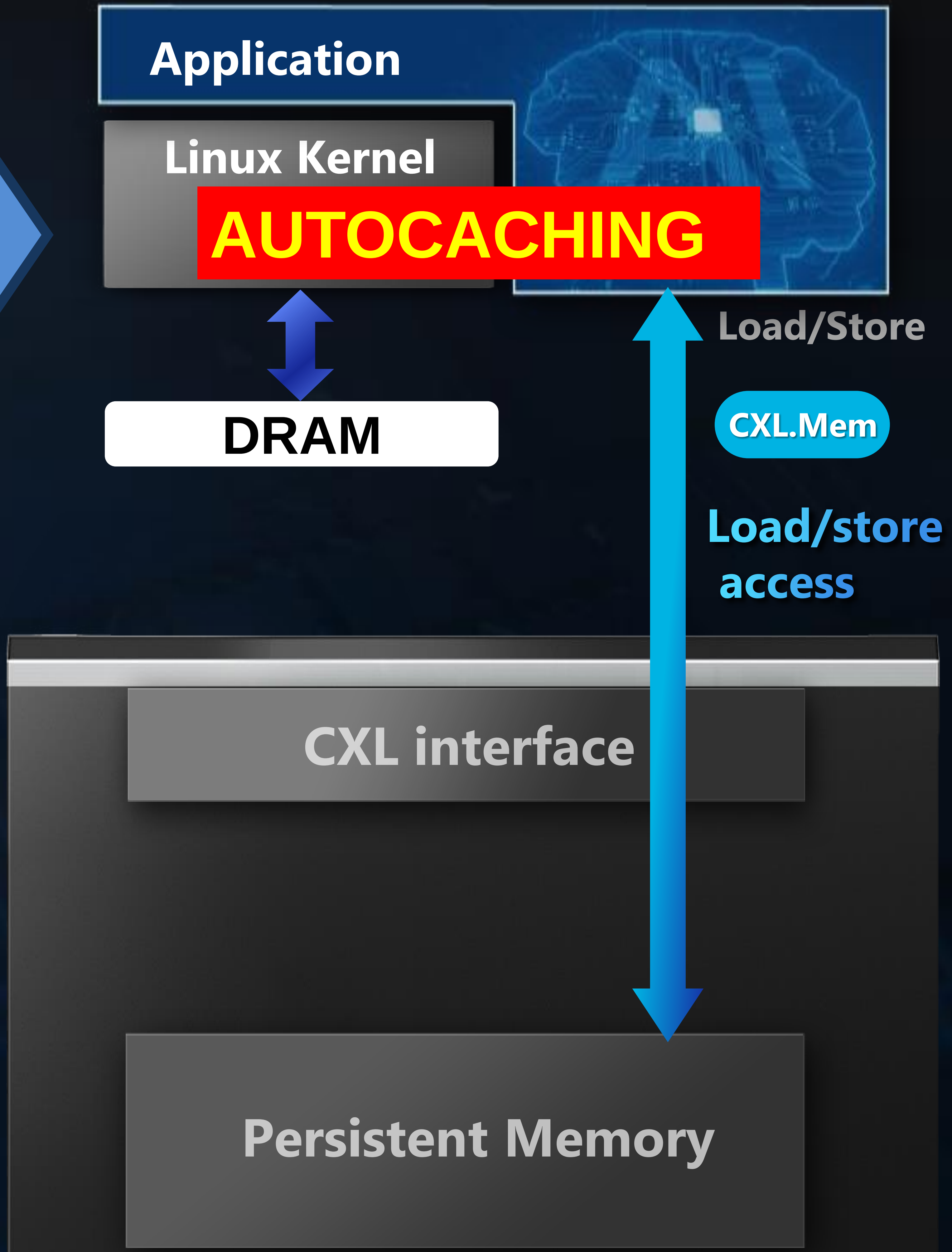


Load/store access

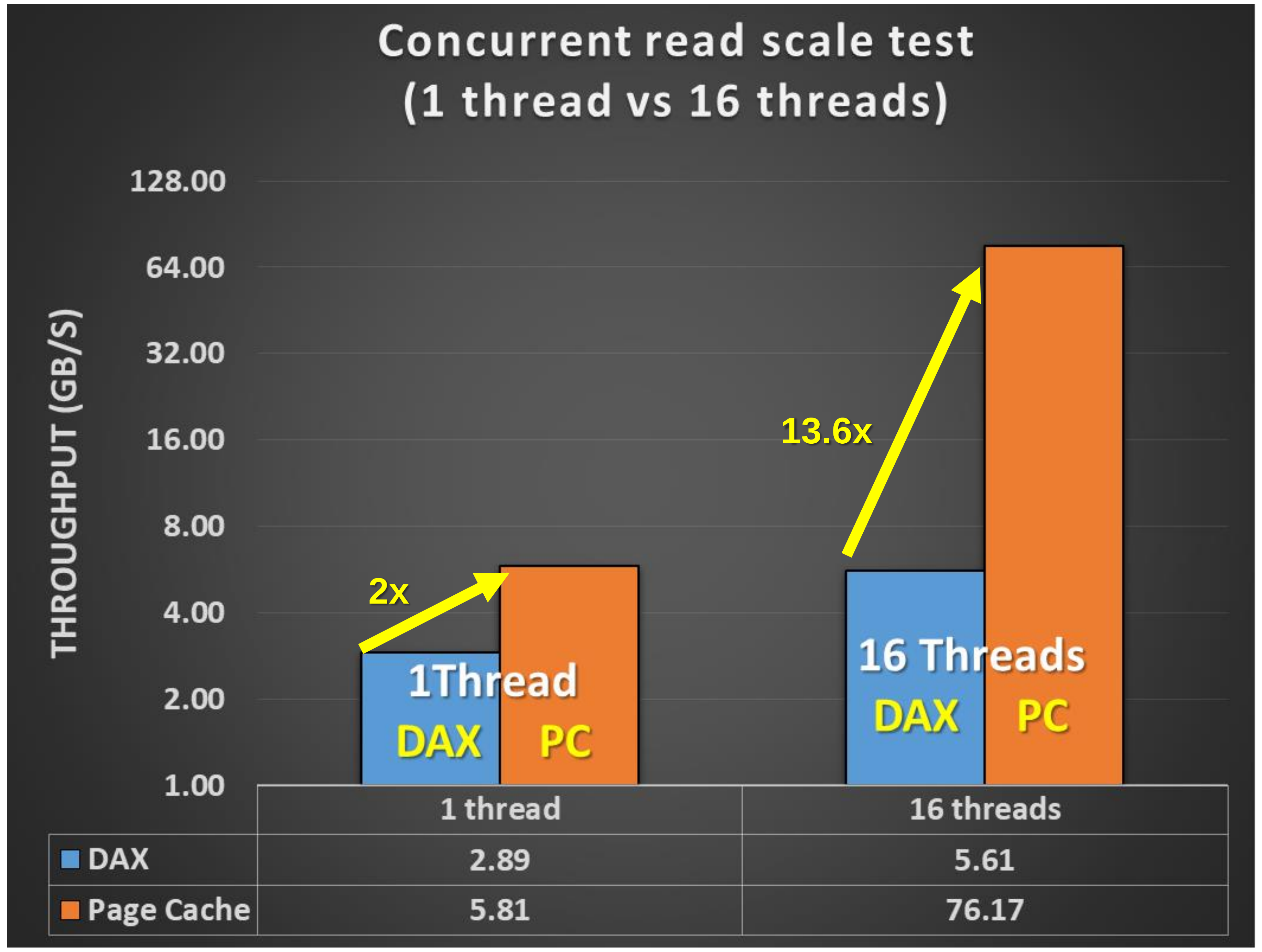
# Motivation: The advent of new types of storage device based on CXL



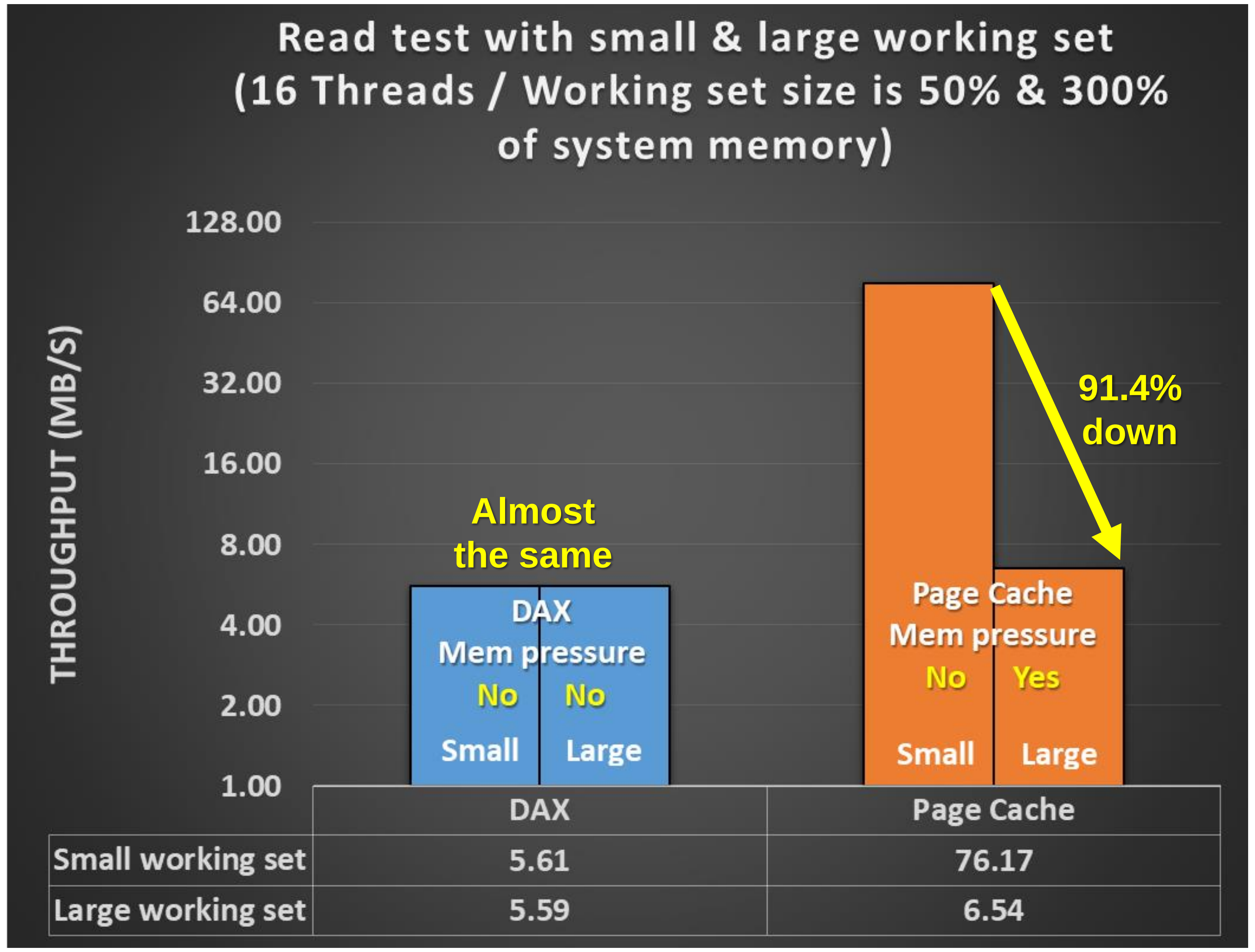
Autocaching can handle I/O efficiently for both architectures. by leveraging DRAM and Device Memory simultaneously



# Motivation: Read Test


Concurrent read scale test
(1 thread vs 16 threads)

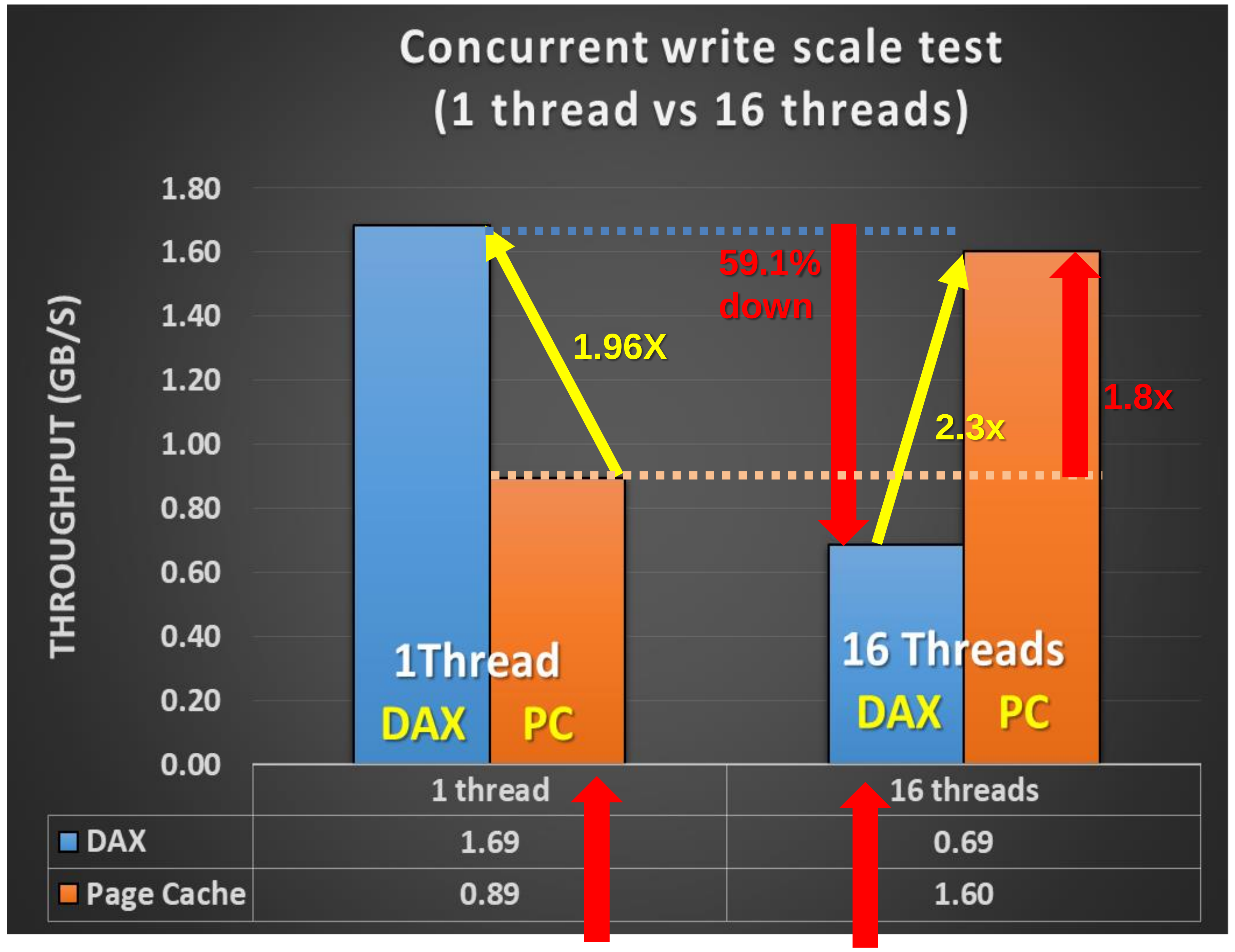
| | 1 thread | 16 threads |
|---|---|---|
| 🟦 DAX | 2.89 | 5.61 |
| 🟧 Page Cache | 5.81 | 76.17 |


Read test with small & large working set
(16 Threads / Working set size is 50% & 300% of system memory)

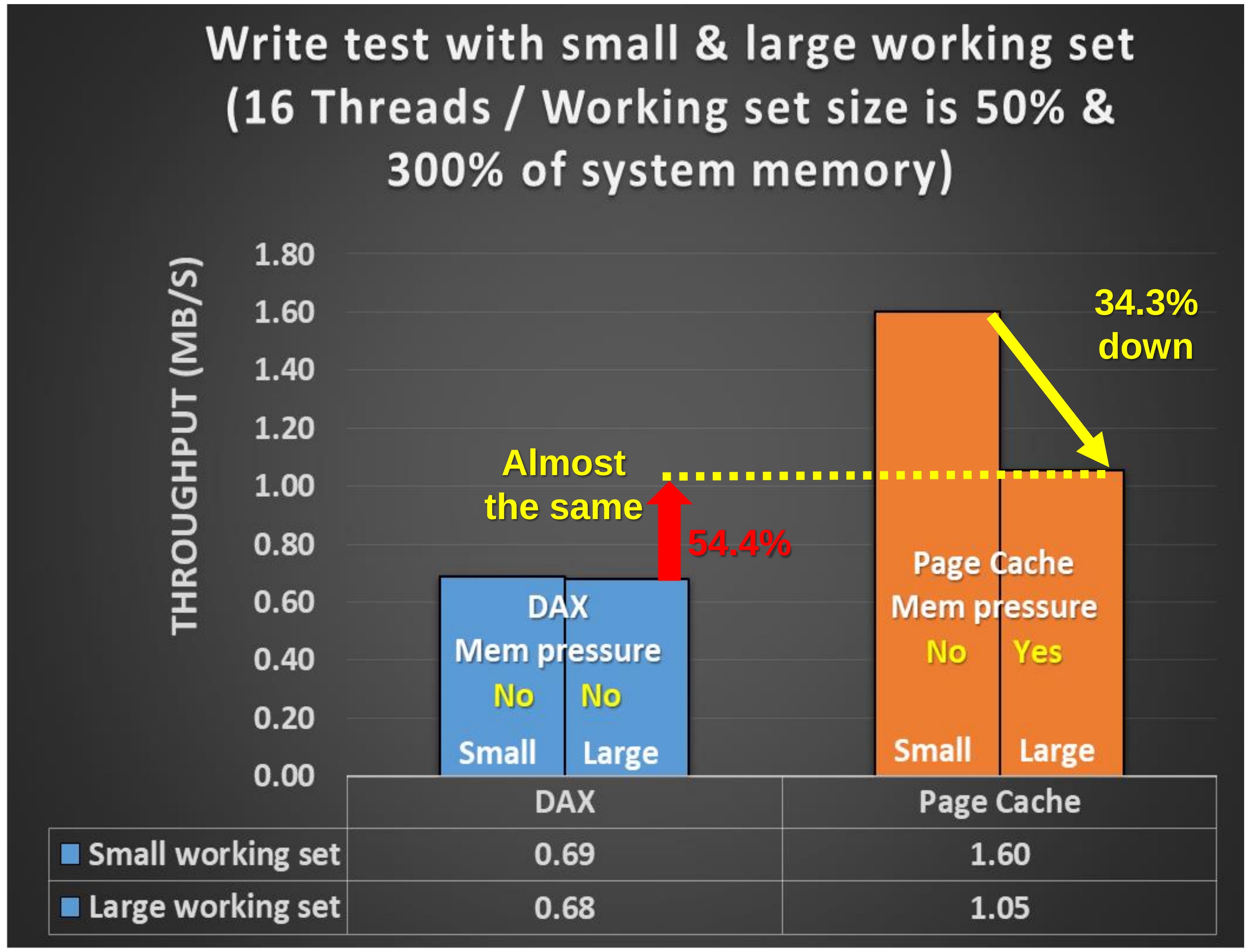
| | DAX | Page Cache |
|---|---|---|
| Small working set | 5.61 | 76.17 |
| Large working set | 5.59 | 6.54 |

- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Read
- Block size: 4KB
- Direct is 0(no msync after write)

# Motivation: Write Test



| | 1 thread | 16 threads |
|---|---|---|
| 🟦 DAX | 1.69 | 0.69 |
| 🟧 Page Cache | 0.89 | 1.60 |



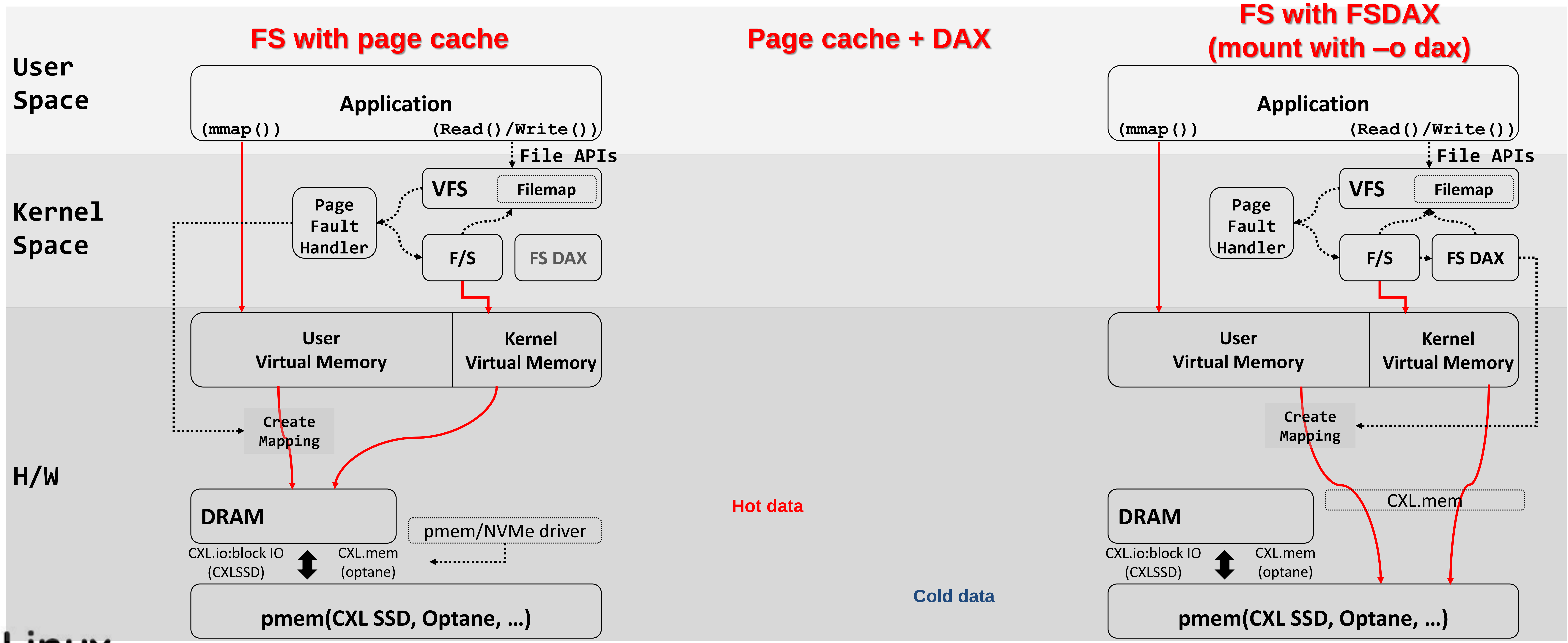
| | DAX | Page Cache |
|---|---|---|
| 🟦 Small working set | 0.69 | 1.60 |
| 🟦 Large working set | 0.68 | 1.05 |

**Flush daemon continues to write-back dirty pages to device and interferes application(lock page and cache pollution)**

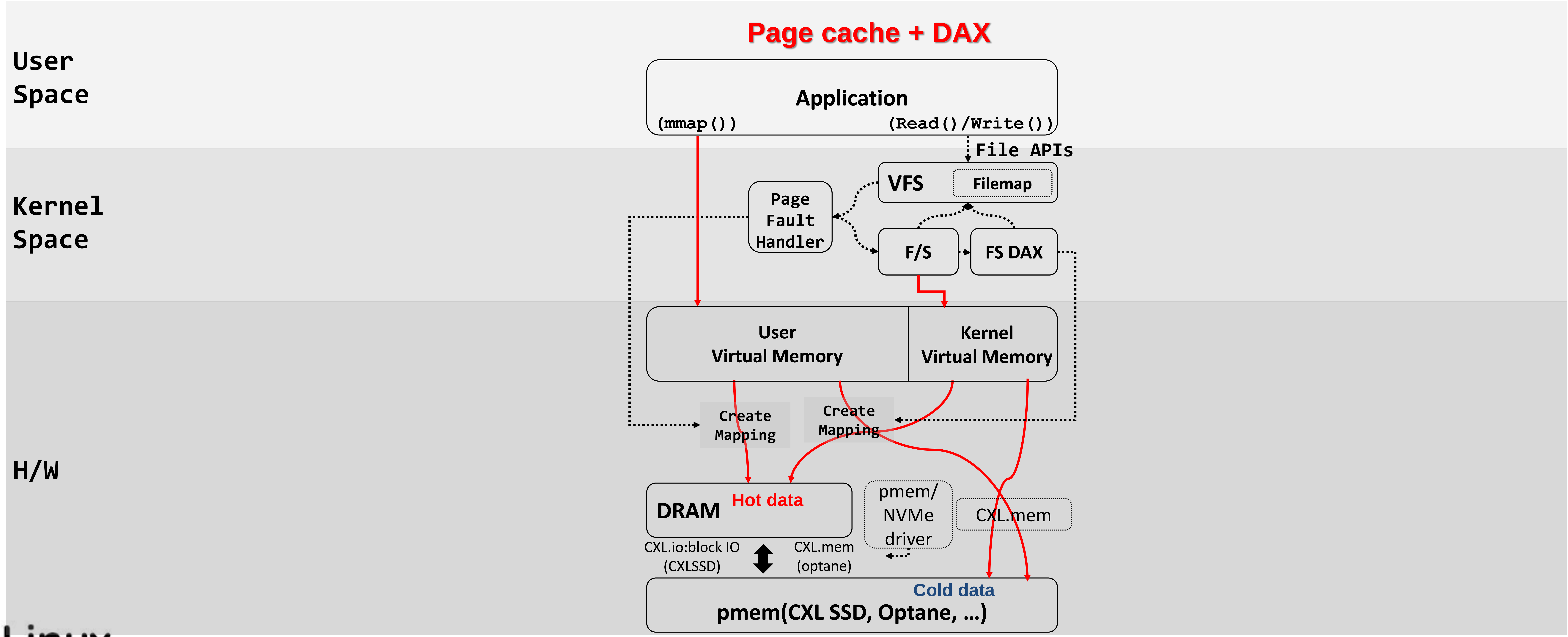
**Optane could not support concurrent write efficiently**

- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Write
- Block size: 4MB
- Direct is 0(no msync after write)

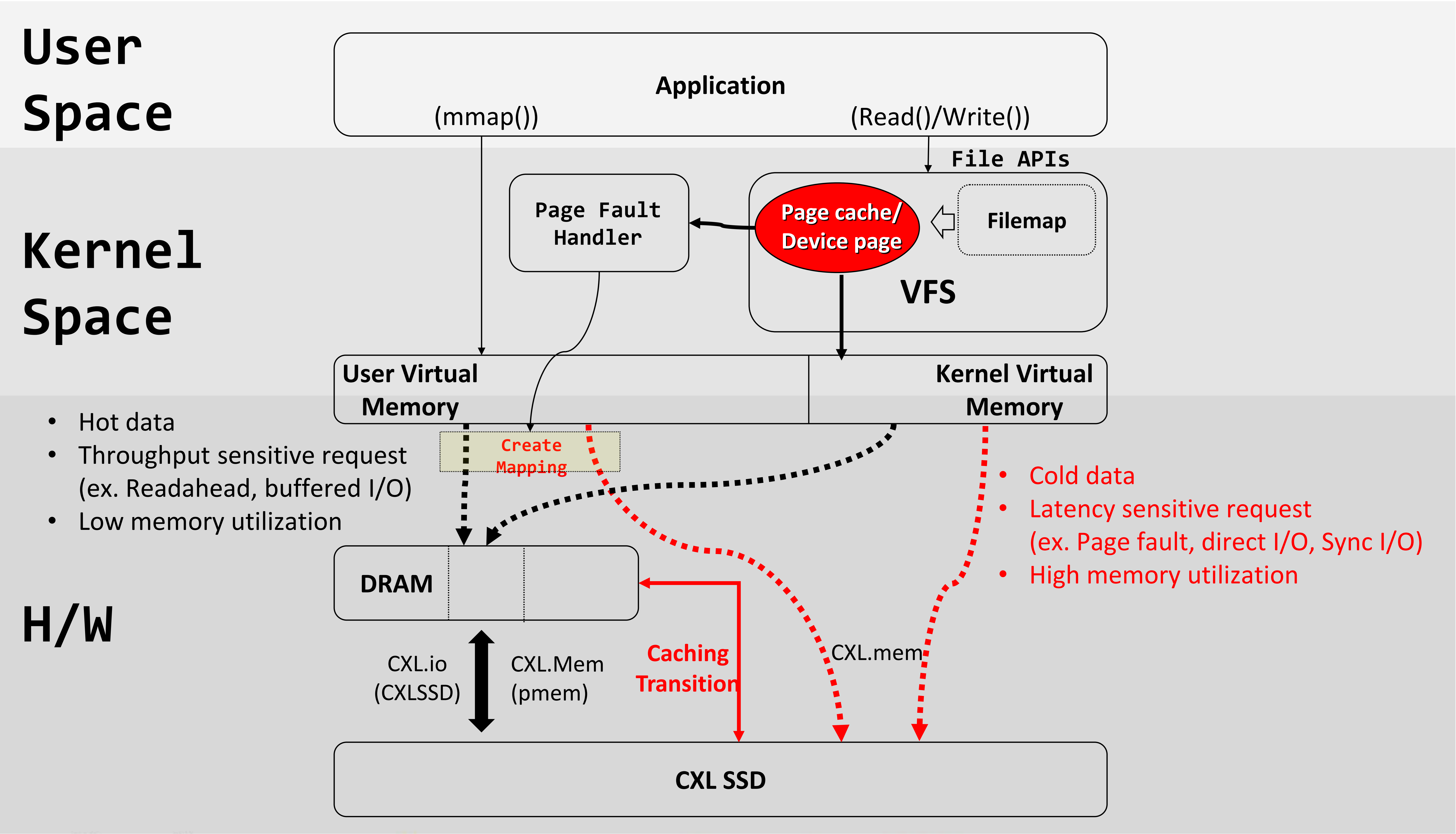
# Motivation: Linux Kernel Internal



# Motivation: Linux Kernel Internal



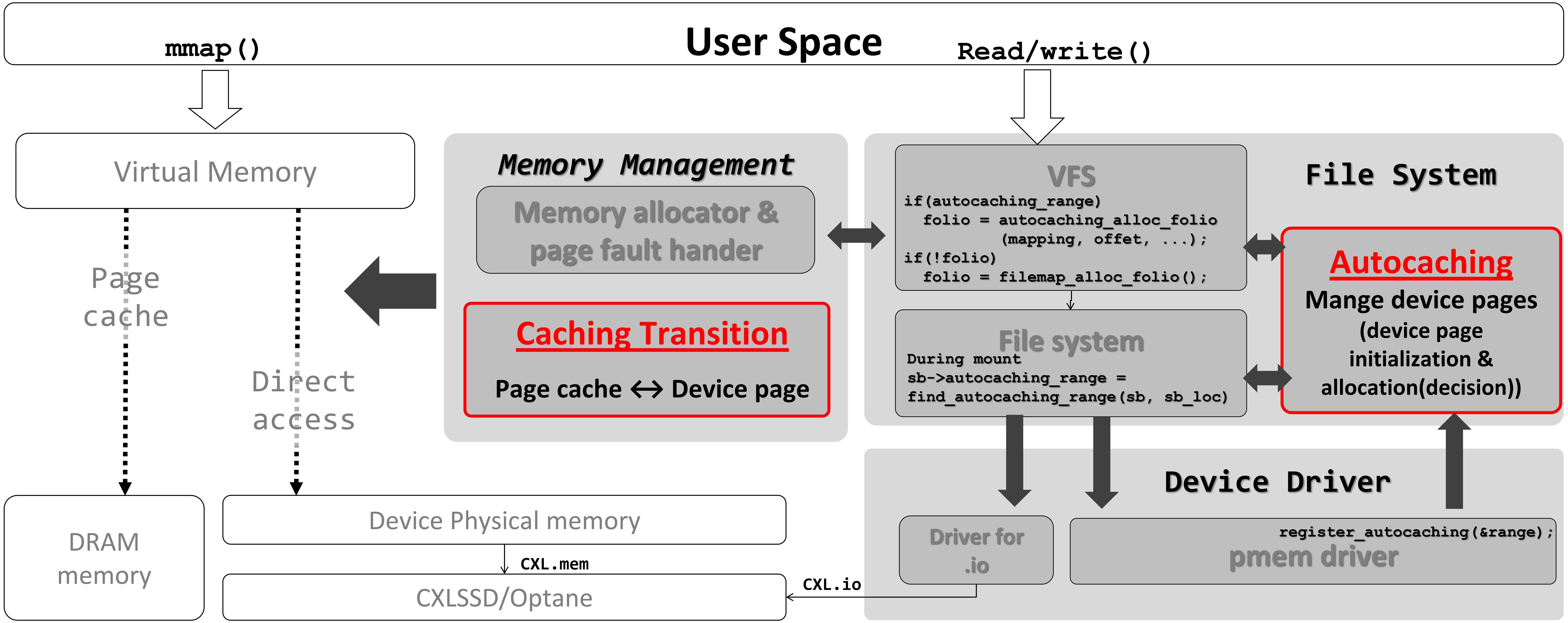
# Design of Autocaching



- Hot data
- Throughput sensitive request (ex. Readahead, buffered I/O)
- Low memory utilization

- Cold data
- Latency sensitive request (ex. Page fault, direct I/O, Sync I/O)
- High memory utilization

# Implementation of Autocaching



User Space

`mmap()`

`Read/write()`

Virtual Memory

Page cache

Direct access

*Memory Management*

Memory allocator & page fault hander

**Caching Transition**

Page cache ↔ Device page

VFS

```
if(autocaching_range)
  folio = autocaching_alloc_folio
          (mapping, offet, ...);
if(!folio)
  folio = filemap_alloc_folio();
```

File System

**Autocaching**

Mange device pages (device page initialization & allocation(decision))

File system

```
During mount
sb->autocaching_range =
find_autocaching_range(sb, sb_loc)
```

Device Driver

Driver for .io

```
register_autocaching(&range);
```

pmem driver

DRAM memory

Device Physical memory

CXL.mem

CXLSSD/Optane

CXL.io

# Implementation of Autocaching

- Challenges

1. Space overhead of device page structure
   - device page can waste precious DRAM memory
2. Cache transition
   - Detect warm page
3. Allocation Policy
   - Allocation policy to determine whether to allocate page cache or device pages

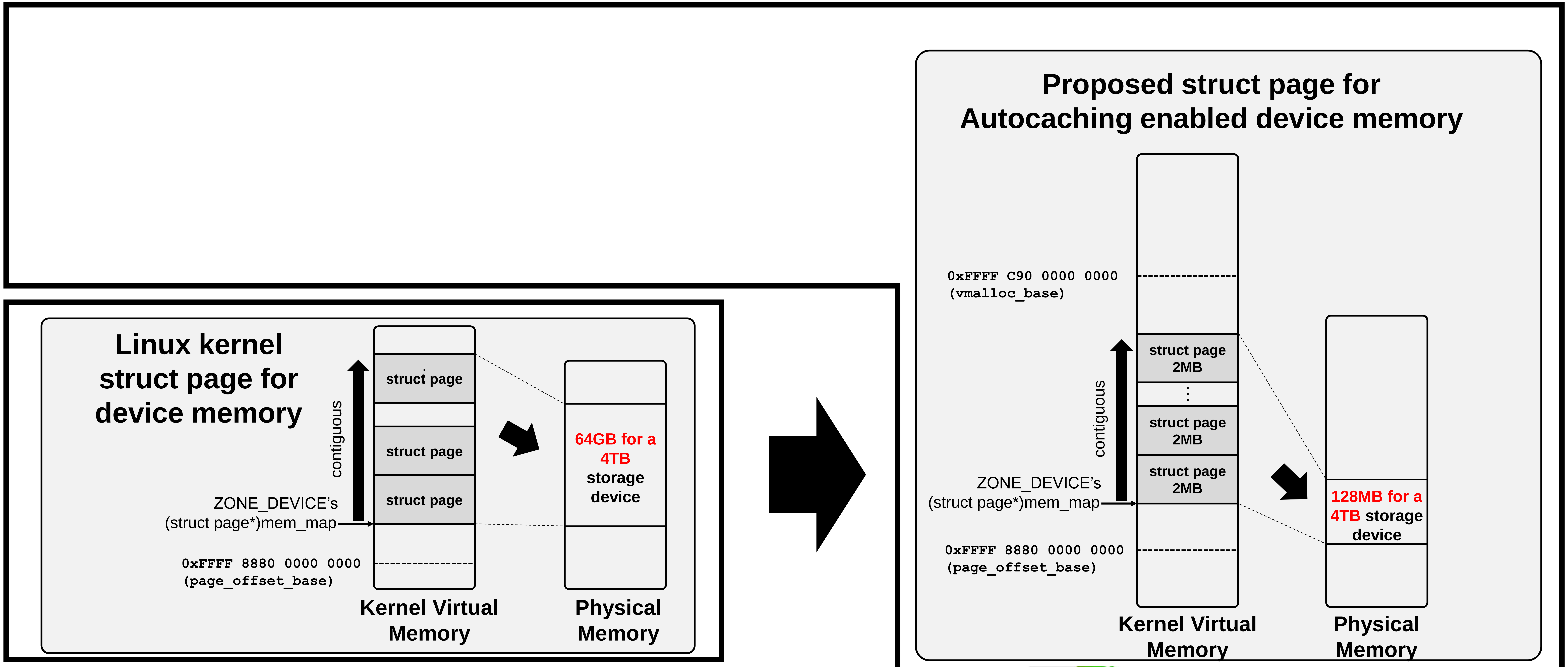
# Implementation of Autocaching
# - Space Overhead of device struct pages -

- Unlike ZONE_NORMAL pages, the ZONE_DEVICE pages may not be actively used if the corresponding blocks are not mapped to a file or the file has not been opened yet.
  - Memory allocator allocates ZONE_NORMAL pages using page struct.
    - Inactive dram page’s struct pages are actively used.
  - File system allocates blocks using the block map
    - Inactive device page’s struct pages are not used at all.

- Even though page structures are not used at all, system should reserve memory for all struct pages belonging to ZONE_DEVICE
  - **struct page size is 64B per 4KB ∴ Need 64GB for 4TB Storage**
  - Waste of system memory
    - The system reserves 64GB memory for device struct page on booting(or pmem device initialization), and the reserved region can not be used for any other purpose, even if CXL storage memory is not actively used.

# Implementation of Autocaching
## - Space Overhead of device struct pages -

- **Dynamic memory allocation for device memory struct page**



**Linux kernel struct page for device memory**

contiguous

struct page

struct page

struct page

ZONE_DEVICE's
(struct page*)mem_map

`0xFFFF 8880 0000 0000`
`(page_offset_base)`

64GB for a 4TB storage device

Kernel Virtual Memory

Physical Memory

**Proposed struct page for Autocaching enabled device memory**

`0xFFFF C90 0000 0000`
`(vmalloc_base)`

struct page 2MB

struct page 2MB

struct page 2MB

contiguous

ZONE_DEVICE's
(struct page*)mem_map

`0xFFFF 8880 0000 0000`
`(page_offset_base)`

128MB for a 4TB storage device

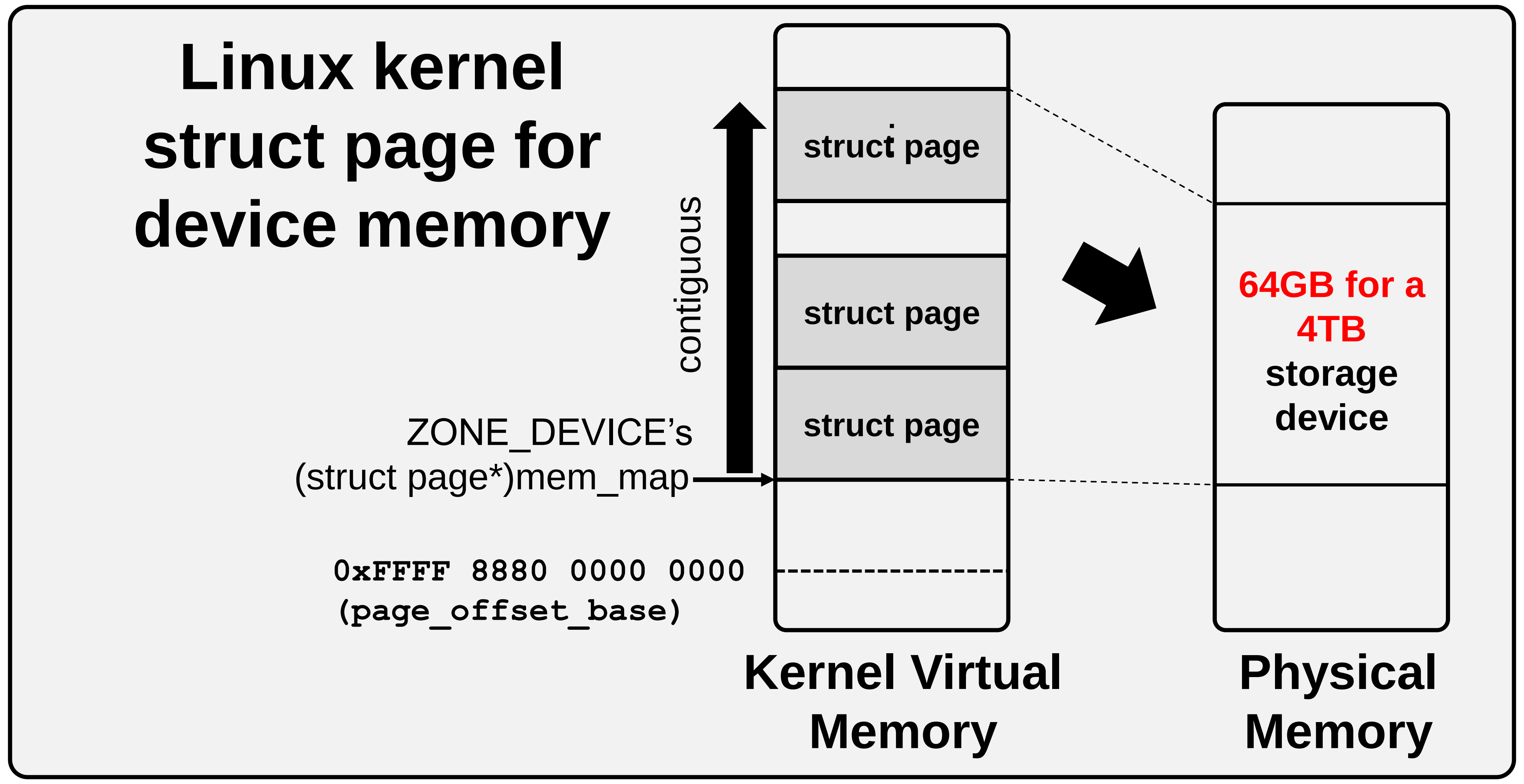
Kernel Virtual Memory

Physical Memory

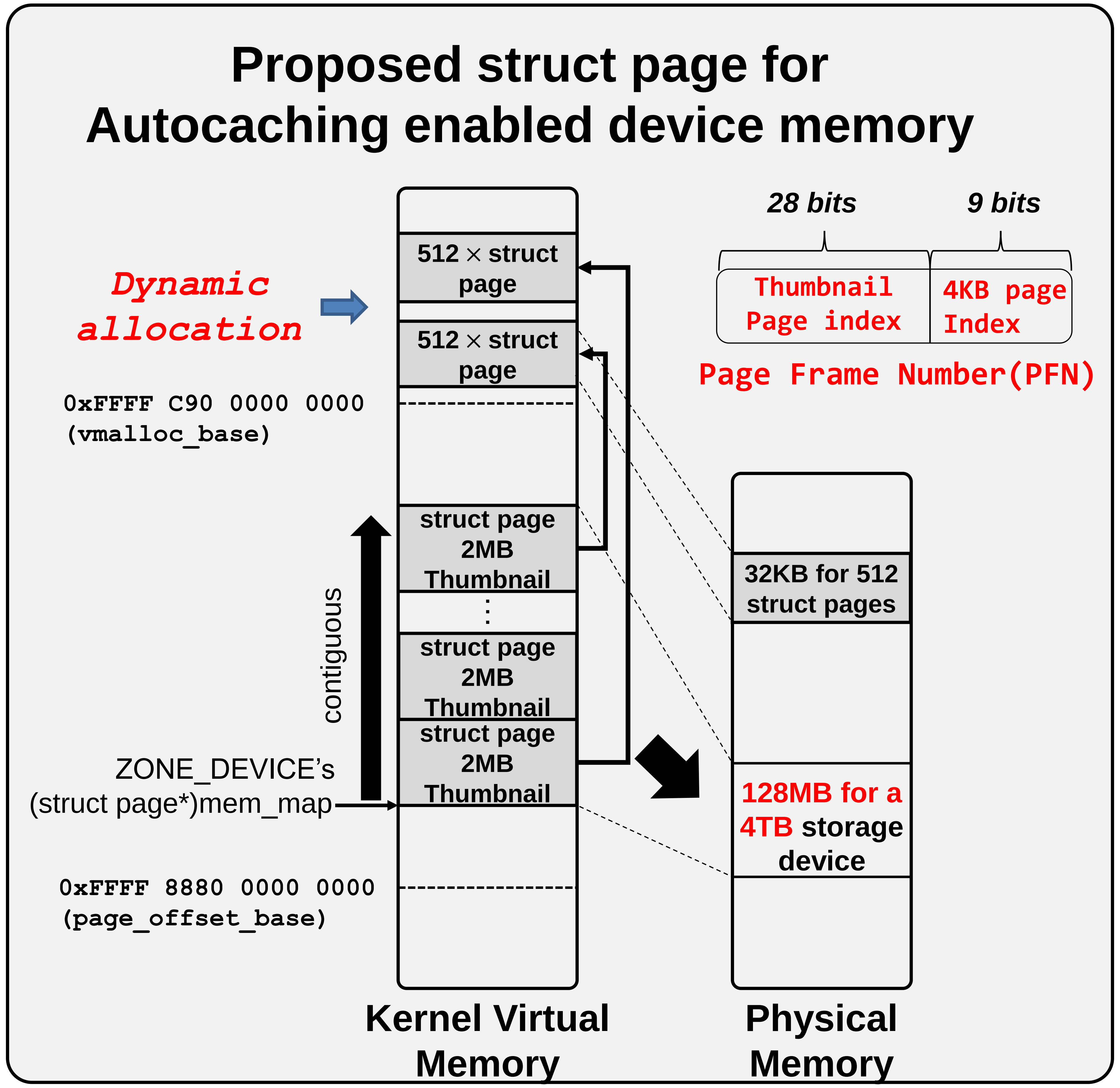
# Implementation of Autocaching
## - Space Overhead of device struct pages -

- **Dynamic memory allocation for device memory struct page**

```
Struct page{/*For thumbnail page*/
   flag & (1<<PG_thumbnail)
   union{
      struct{
         struct page* pages;
         spinlock_t lock;
        int count;
      }
```



**Linux kernel struct page for device memory**

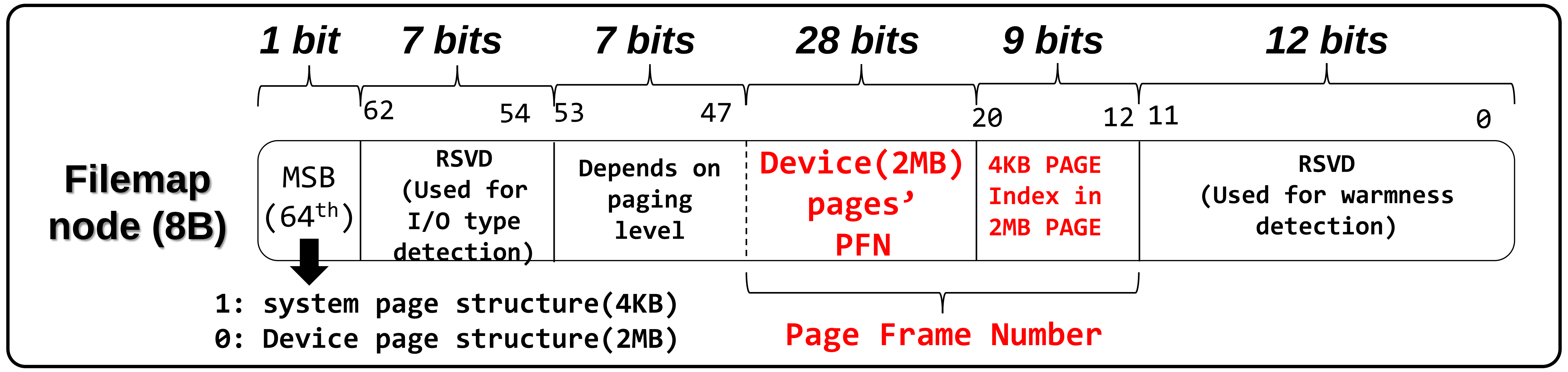


**Proposed struct page for Autocaching enabled device memory**

# Implementation of Autocaching
## - Space Overhead of device struct pages -

- Dynamic memory allocation for device memory struct page

### Filemap(xarray) Node



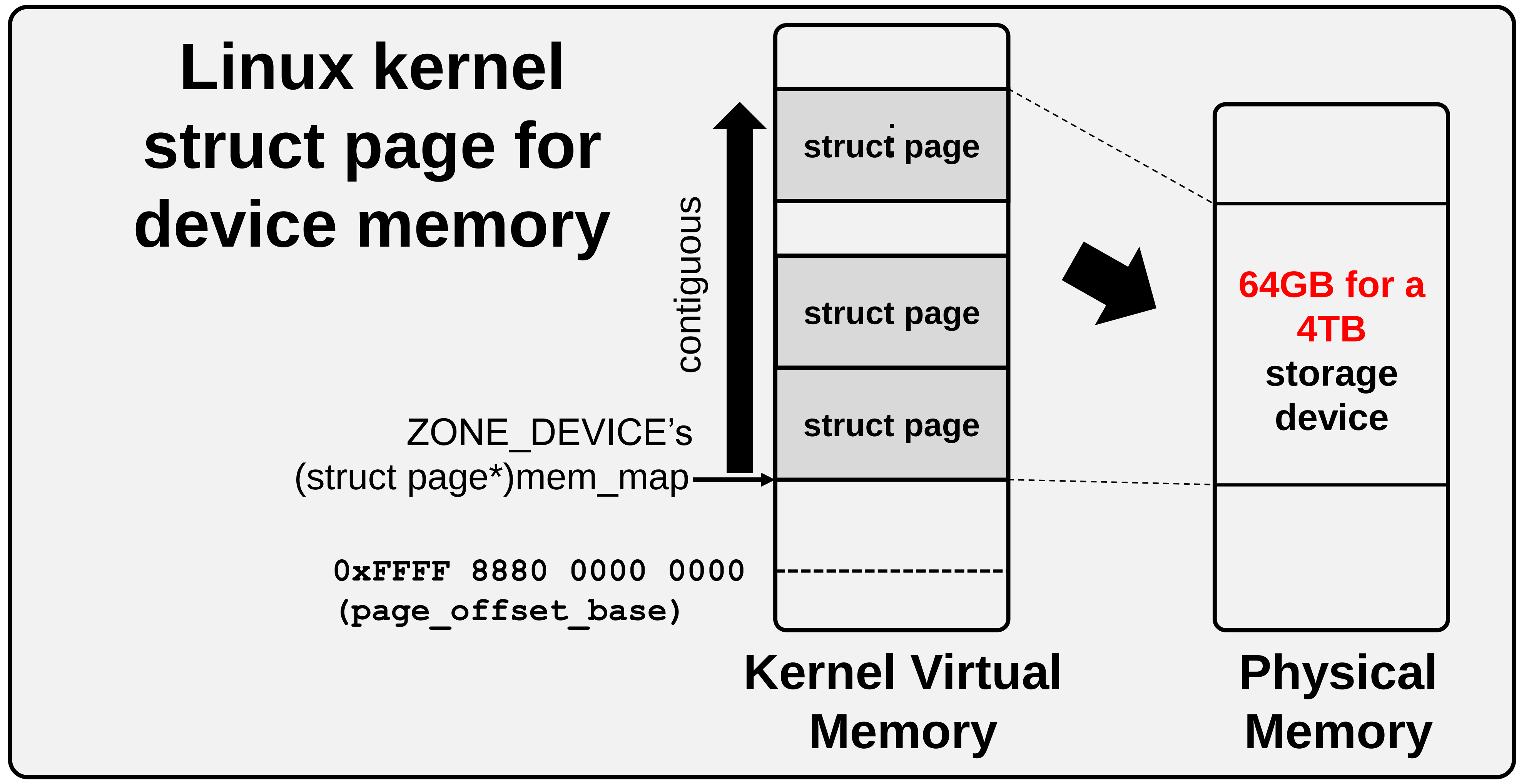
Filemap node (8B)

| 1 bit | 7 bits | 7 bits | 28 bits | 9 bits | 12 bits |
|---|---|---|---|---|---|
| | 62 … 54 | 53 … 47 | | 20 … 12 | 11 … 0 |
| MSB (64th) | RSVD (Used for I/O type detection) | Depends on paging level | Device(2MB) pages' PFN | 4KB PAGE Index in 2MB PAGE | RSVD (Used for warmness detection) |

1: system page structure(4KB)
0: Device page structure(2MB)

Page Frame Number

### Linux kernel struct page for device memory



contiguous

struct page

struct page

struct page

ZONE_DEVICE's (struct page*)mem_map

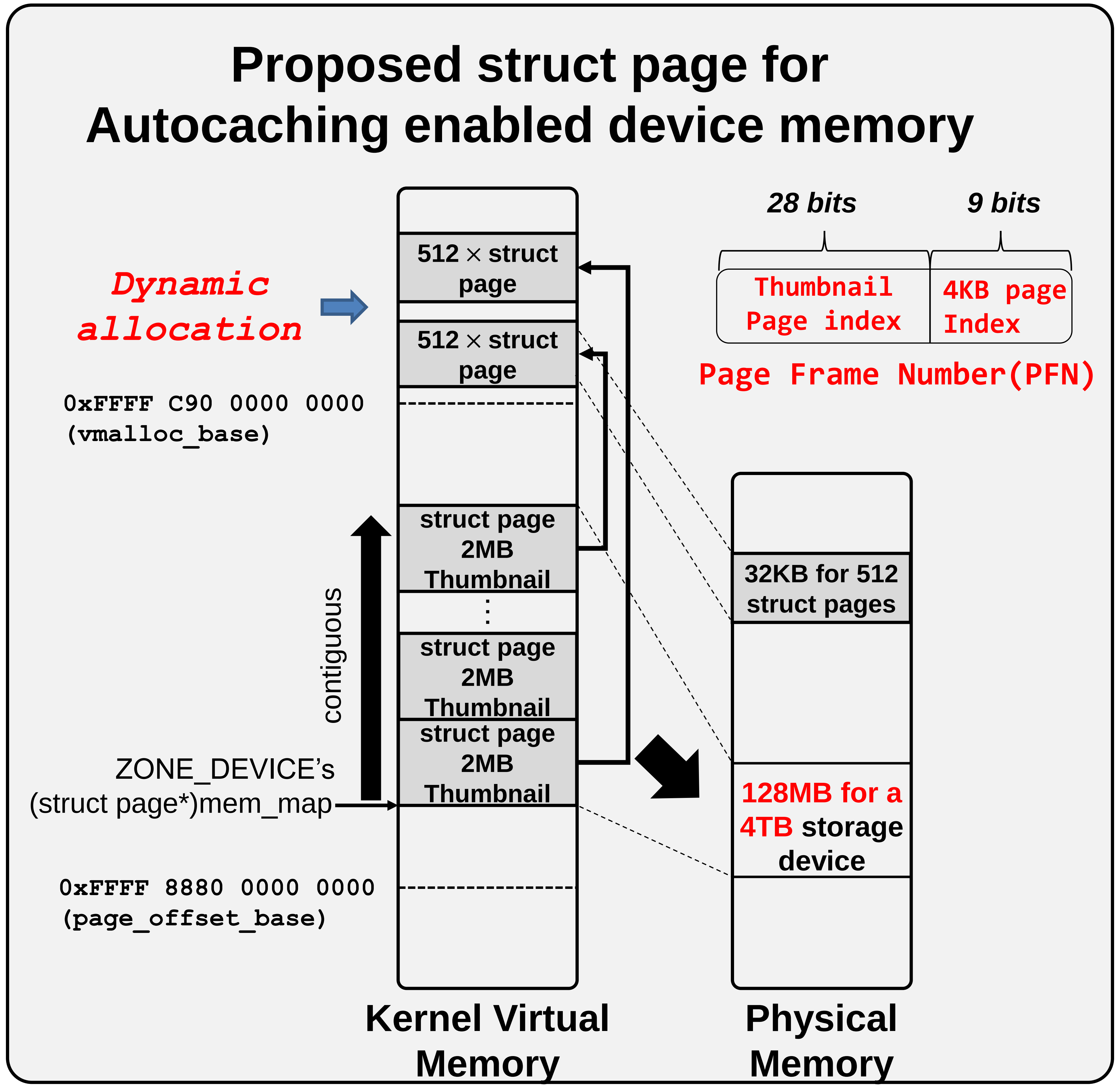
0xFFFF 8880 0000 0000 (page_offset_base)

Kernel Virtual Memory

64GB for a 4TB storage device

Physical Memory

### Proposed struct page for Autocaching enabled device memory



Dynamic allocation

512 × struct page

512 × struct page

0xFFFF C90 0000 0000 (vmalloc_base)

contiguous

struct page 2MB Thumbnail

struct page 2MB Thumbnail

struct page 2MB Thumbnail

ZONE_DEVICE's (struct page*)mem_map

0xFFFF 8880 0000 0000 (page_offset_base)

Kernel Virtual Memory

| 28 bits | 9 bits |
|---|---|
| Thumbnail Page index | 4KB page Index |

Page Frame Number(PFN)

32KB for 512 struct pages

128MB for a 4TB storage device

Physical Memory

# Implementation of Autocaching
## - Space Overhead of device struct pages -

- Dynamic memory allocation of struct page
  - Address conversion macros

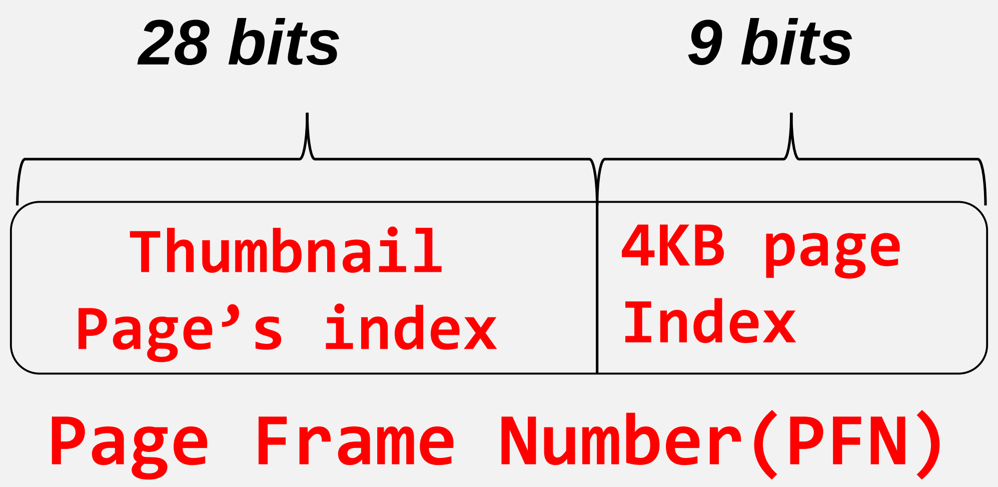
### Address conversion macros

```
page_to_pfn(page){
    page – mem_map;
}

pfn_to_page(pfn){
    mem_map + pfn;
}
```

Linux kernel struct page

### Address conversion macros

```
page_to_pfn(page)
 1. page->pfn;(add field)
 2. use rmap
    (filemap node’s PFN)
 3. Reserve VM for all
    struct pages in vmalloc
    area (to use struct page
    offset for PFN)

pfn_to_page(pfn)
  (mem_map + (pfn >> \
   PG_THM_SHIFT))->\
   pages)+ (pfn & \
   ~PG_THM_MASK);
```



*28 bits* *9 bits*

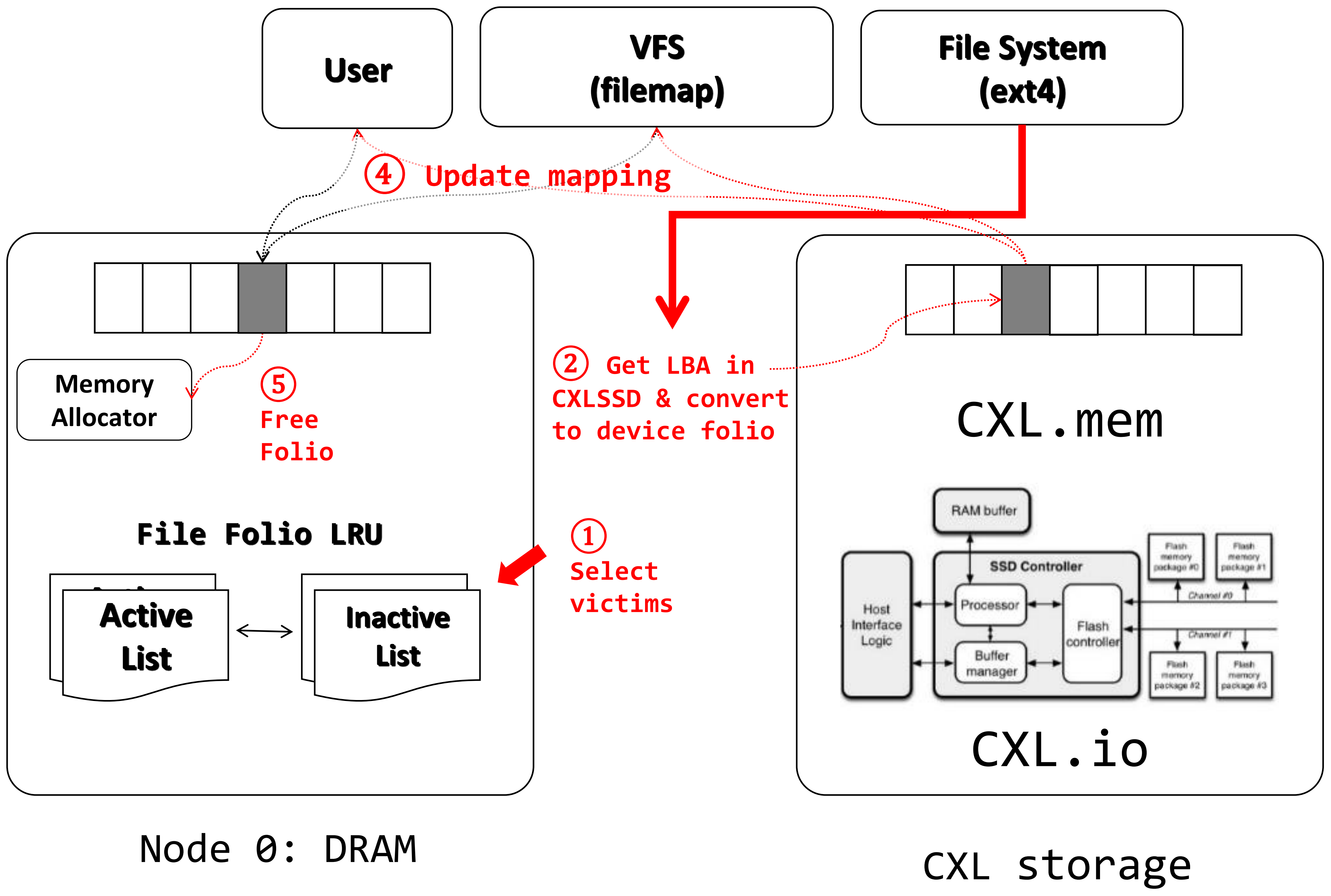
Thumbnail Page’s index | 4KB page Index

Page Frame Number(PFN)

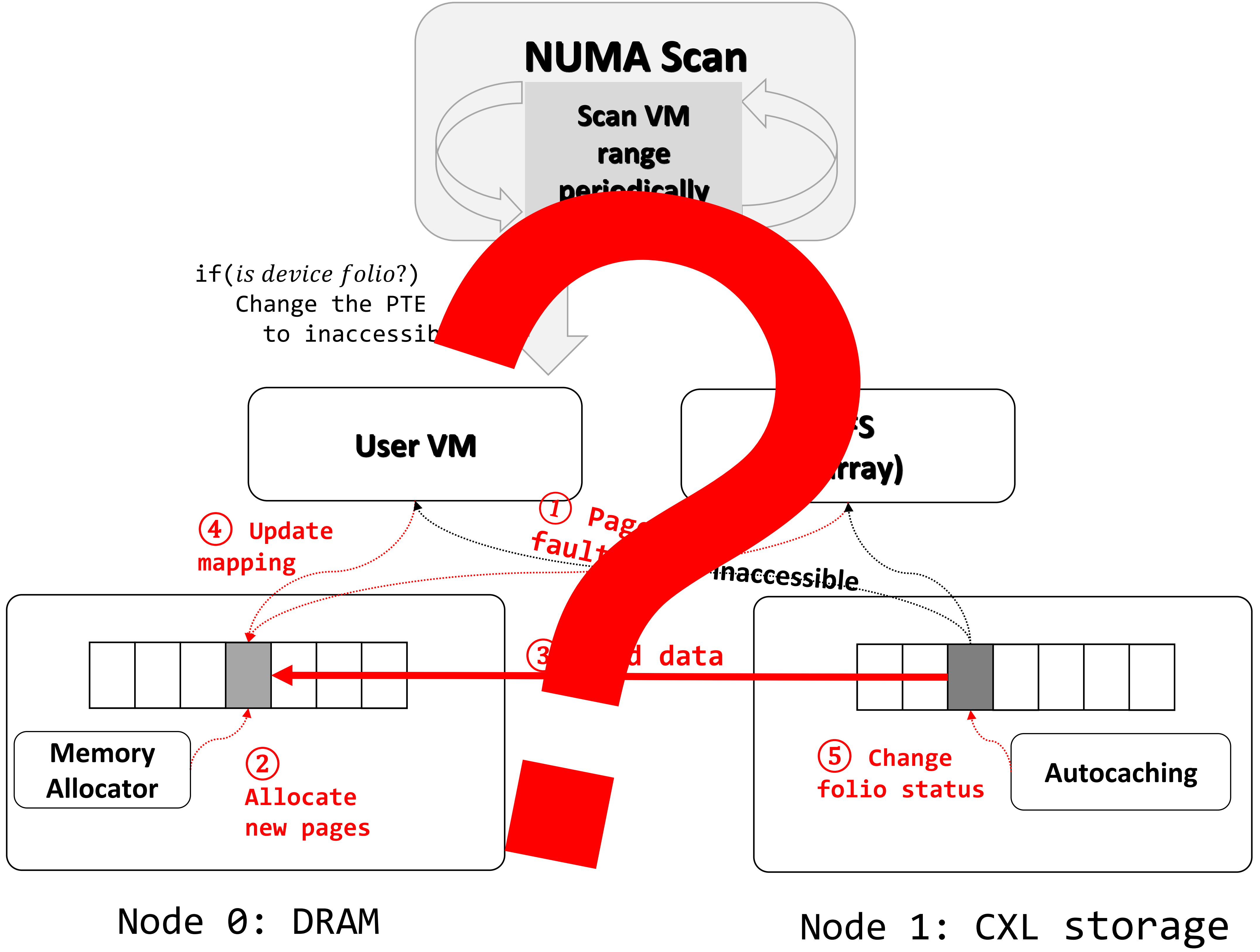
```
#define PG_THM_SHIFT 0x9
#define PG_THM_MASK  0x1ff
Struct page{
   flag & (1<<PG_thm)
   union{
    struct{
       struct page* pages;
       spinlock_t lock;
}}}
```

Proposed struct page for device memory

# Implementation of Autocaching
# - Caching Transition -



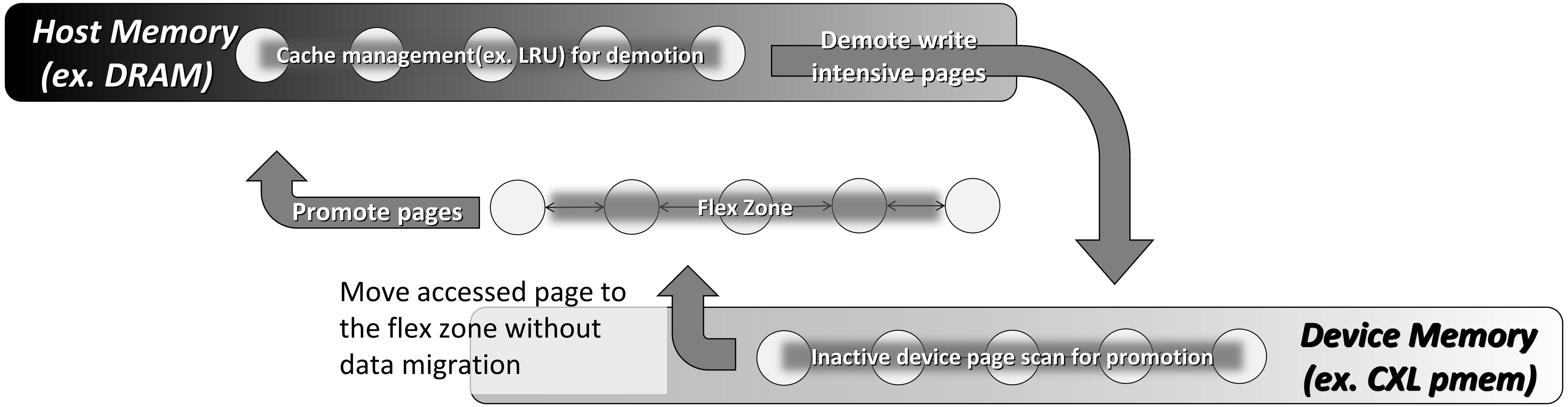
**Caching Transition: Demotion**
**(Cache invalidation & Direct Mapping)**



**Caching Transition: Promotion**

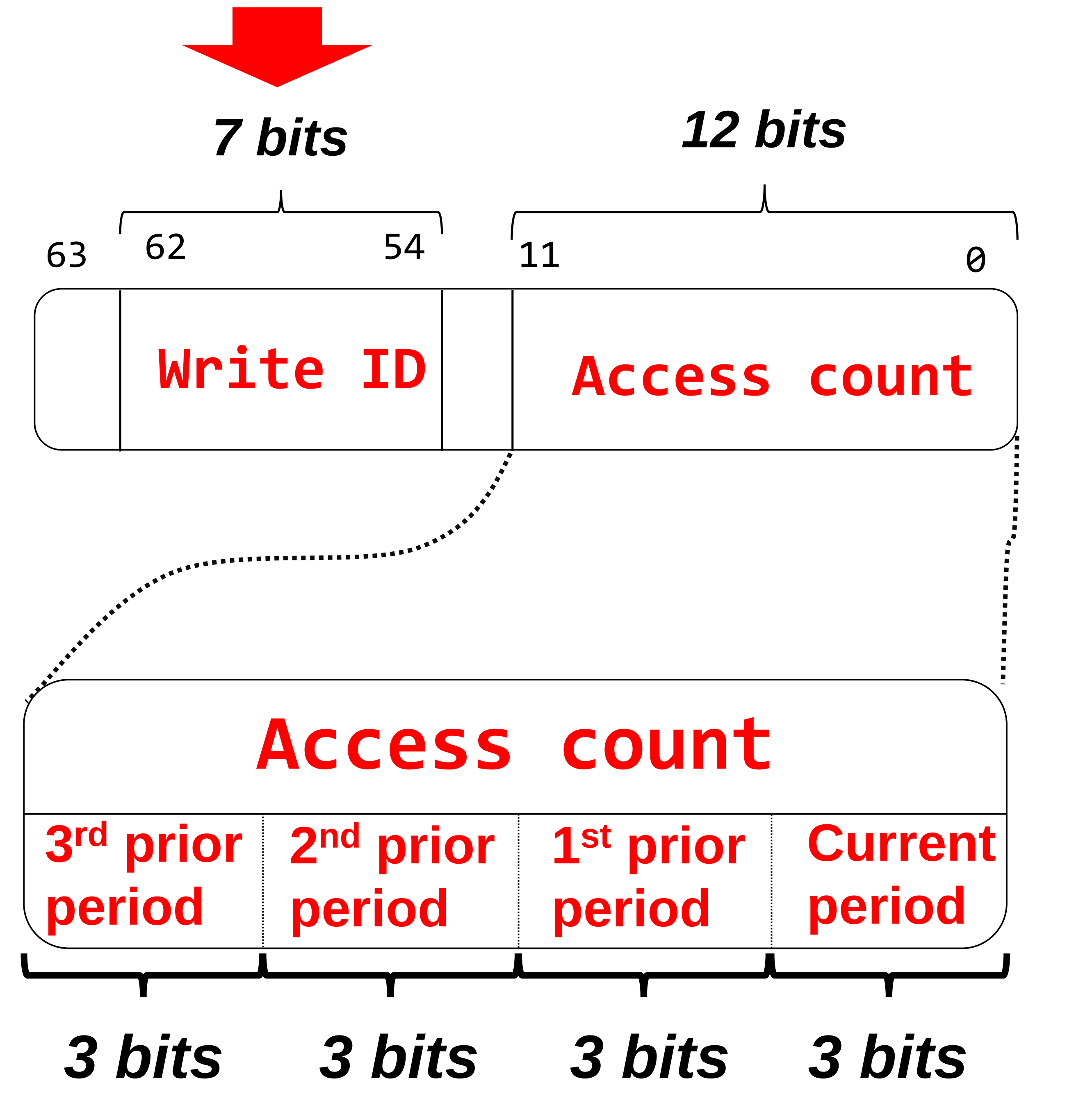
# Implementation of Autocaching
# - Caching Transition -

- Flex zone
  - A promotion daemon virtually promotes accessed pages to flex zone and keeps watching whether the page is hot or warm.
  - A flex zone has a limited number of pages.
    - If ping-pong happens, increase the size of flex zone.
    - If the flex zone is stable, decrease the size of flex zone.
  - Keep warm(ping-pong) pages in flex zone.
    - Warm page detection based on IO type(read or write), access cycle, access frequency, and recently accessed pages


*Host Memory*
*(ex. DRAM)*
Cache management(ex. LRU) for demotion
Demote write intensive pages
Promote pages
Flex Zone
Move accessed page to the flex zone without data migration
Inactive device page scan for promotion
*Device Memory*
*(ex. CXL pmem)*

## Filemap Node for CXL device page

To detect concurrent write, If PTE's dirty bit is set or write(), store write ID(Hash(tid))



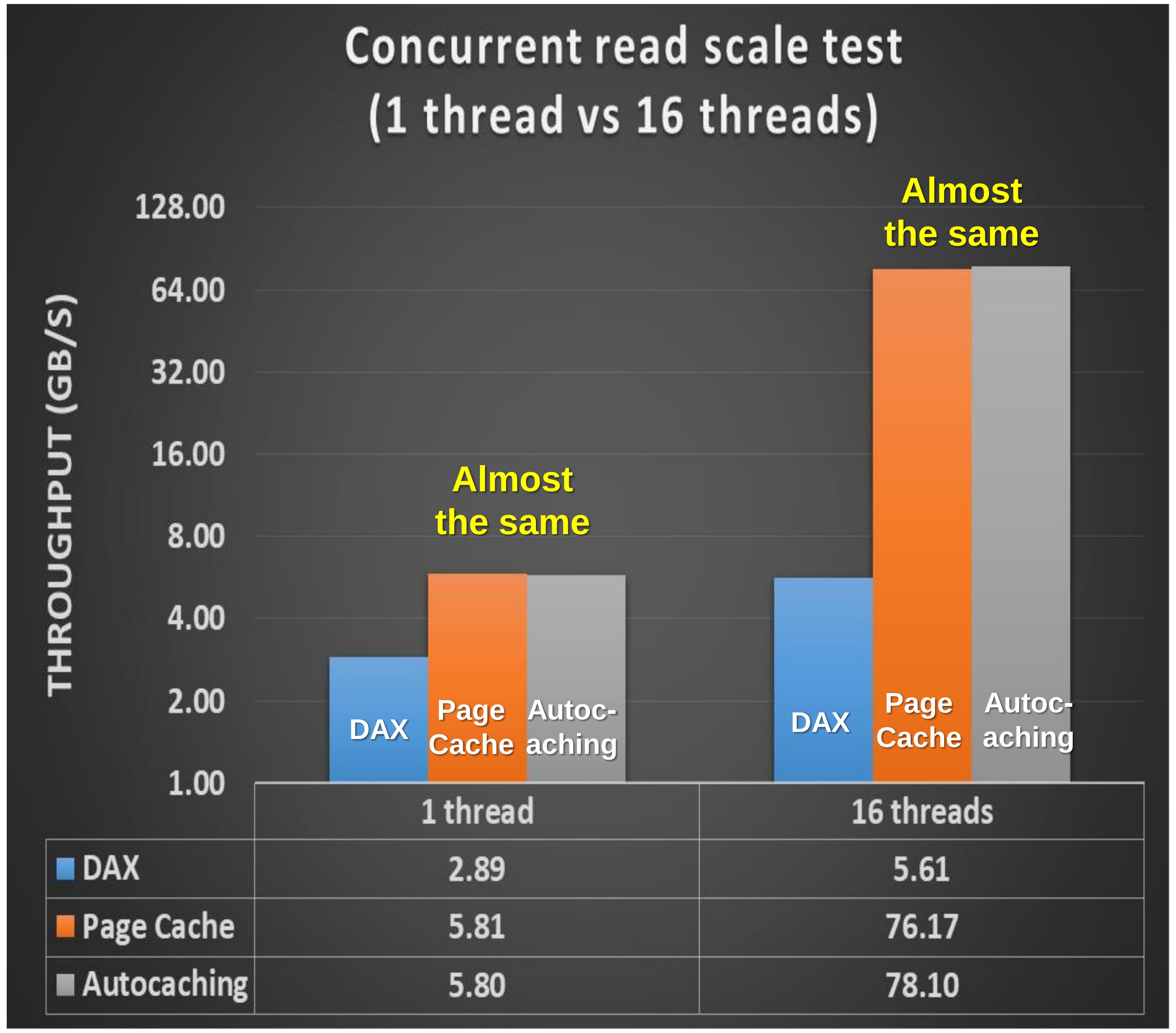
**(Periodically check and left-shift by 3)**

# Implementation of Autocaching
# - Allocation Policy -

- Allocation policy
  - Consider device characteristics
    - CXLSSD(dual interface, CXL.io & CXL.mem) & Optane(Support .mem only)
    - Provides policy control parameters to user space so that different devices can have different policies.
  - Use DRAM
    - File metadata
    - Inline data(store small size of data in inode)
    - New file write or append to support delayed allocation
  - Use device memory
    - sync I/O, Direct I/O

| | metadata | Read operation | Write operation |
|---|---|---|---|
| memory usage < 80% | DRAM | Page cache | • Balance Page cache(50%) and Device page(50%)<br>• Allocate page cache for concurrent write |
| 80% < memory usage < 97%<br>Decrease the DRAM allocation rate. | | Reduce page cache allocation ratio | • Device page<br>• Allocate page cache for concurrent write |
| 97% < memory usage<br>before kswapd starting to reclaim | | • Device page only<br>• Allocate page cache for concurrent write | |

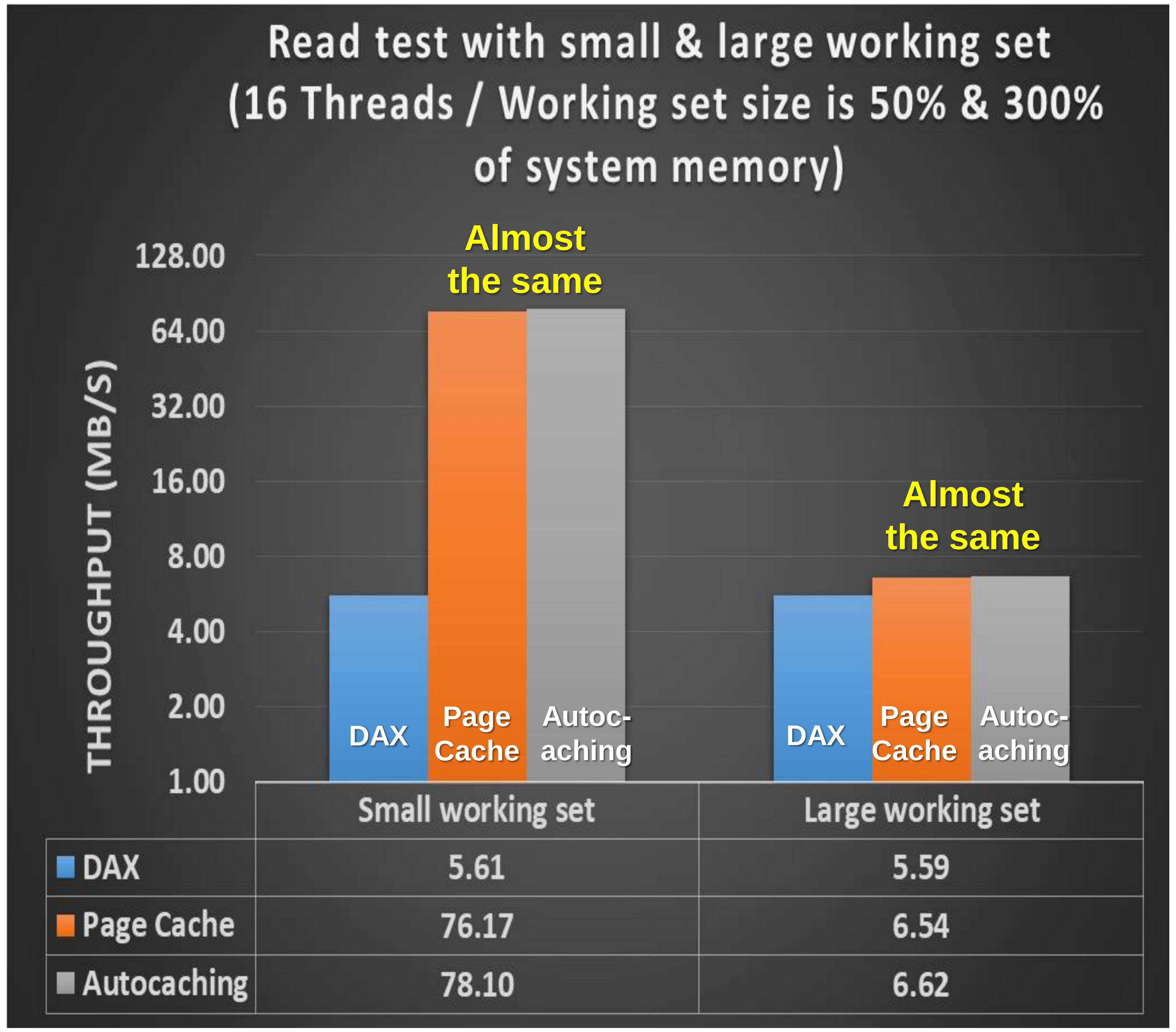
# Preliminary results(Read)


Concurrent read scale test
(1 thread vs 16 threads)

Almost the same

Almost the same

| | 1 thread | 16 threads |
|---|---|---|
| 🟦 DAX | 2.89 | 5.61 |
| 🟧 Page Cache | 5.81 | 76.17 |
| ⬜ Autocaching | 5.80 | 78.10 |


Read test with small & large working set
(16 Threads / Working set size is 50% & 300% of system memory)

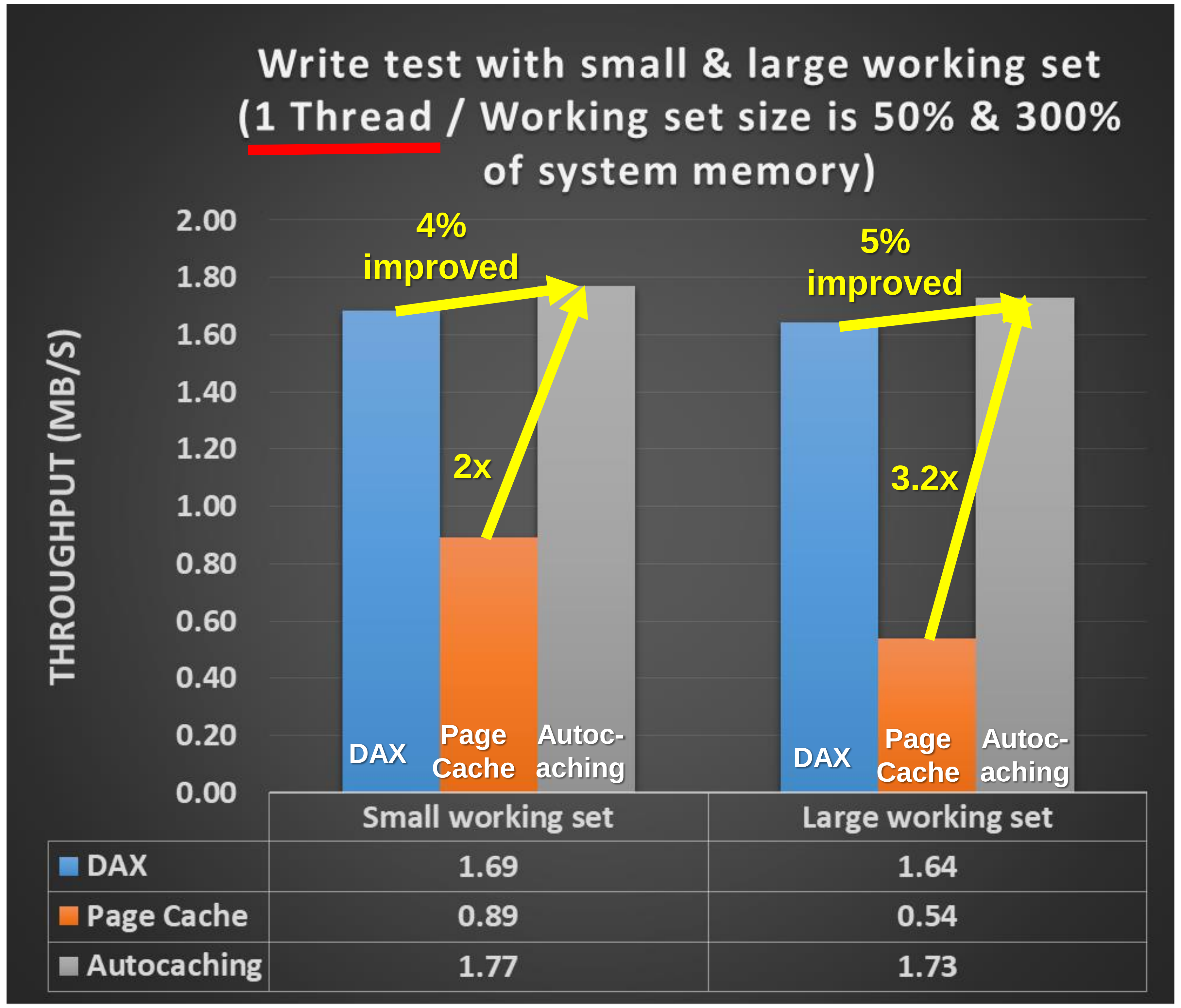
Almost the same

Almost the same

| | Small working set | Large working set |
|---|---|---|
| 🟦 DAX | 5.61 | 5.59 |
| 🟧 Page Cache | 76.17 | 6.54 |
| ⬜ Autocaching | 78.10 | 6.62 |

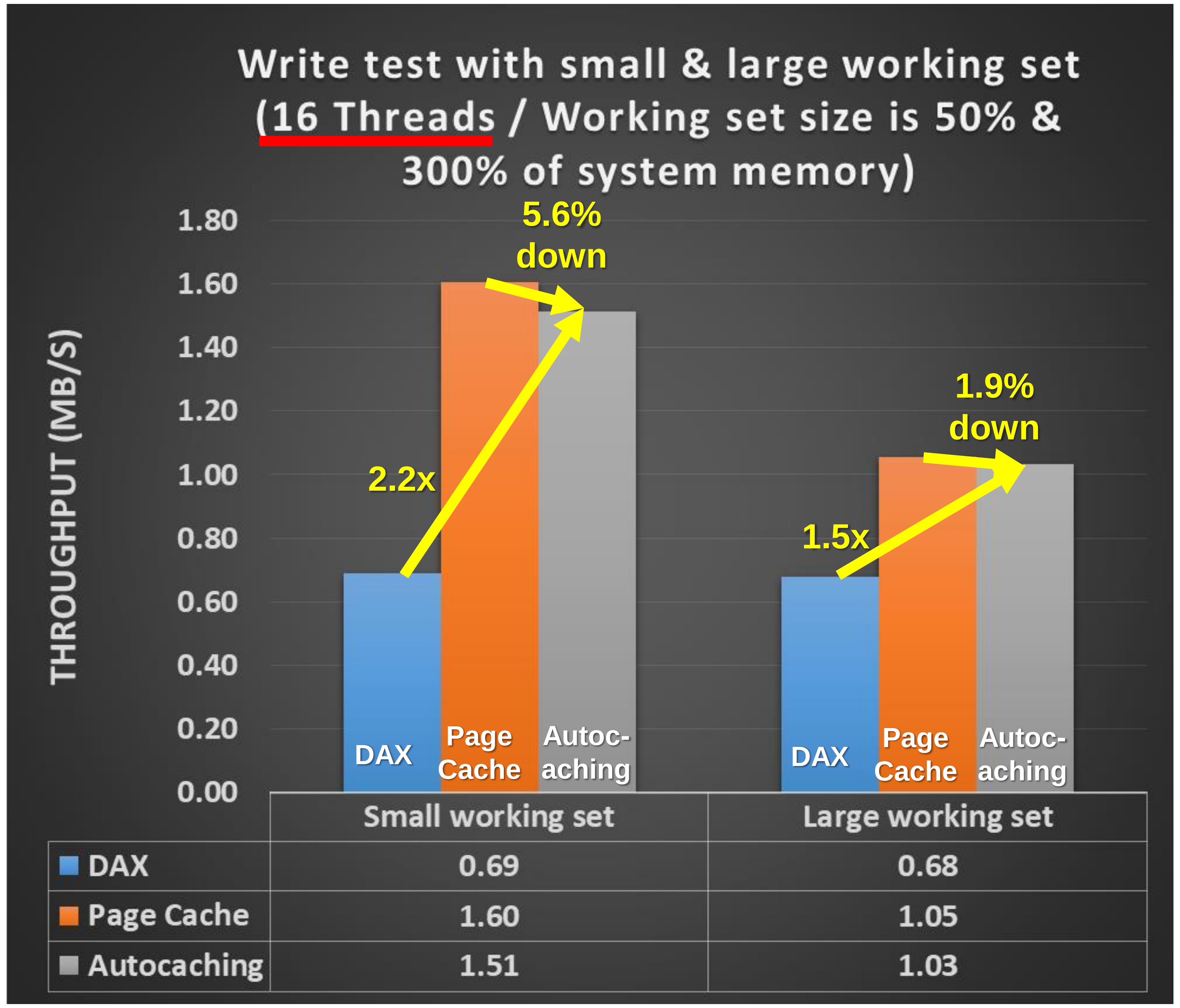
- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Read
- Block size: 4KB
- Direct is 0(no msync after write)

# Preliminary results(Write)



Write test with small & large working set (1 Thread / Working set size is 50% & 300% of system memory)

| | Small working set | Large working set |
|---|---|---|
| ■ DAX | 1.69 | 1.64 |
| ■ Page Cache | 0.89 | 0.54 |
| ■ Autocaching | 1.77 | 1.73 |



Write test with small & large working set (16 Threads / Working set size is 50% & 300% of system memory)

| | Small working set | Large working set |
|---|---|---|
| ■ DAX | 0.69 | 0.68 |
| ■ Page Cache | 1.60 | 1.05 |
| ■ Autocaching | 1.51 | 1.03 |

- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Write
- Block size: 4MB
- Direct is 0(no msync after write)

**Autocaching shows little bit better performance than DAX because it uses DAX and page cache at the same time(Interference by flush daemons is reduced)**

**Autocaching shows a little bit worse performance than page cache because Autocaching uses some device memory allocated before detecting concurrent write.**

# DONE and TO-DO list

- DONE
  - Framework
    - Initialization
    - Device page allocation
    - F/S & VFS changes to support Autocaching
    - Simple Policy

- TO-DO
  - More test with CXL storage devices for accurate and general policy
  - Dynamic struct page allocation
  - Cache transition
  - Alleviate concurrent write issue for Optane
  - Awareness of flush daemon
  - Submit rfc by Q4 for review

# Linux Plumbers Conference

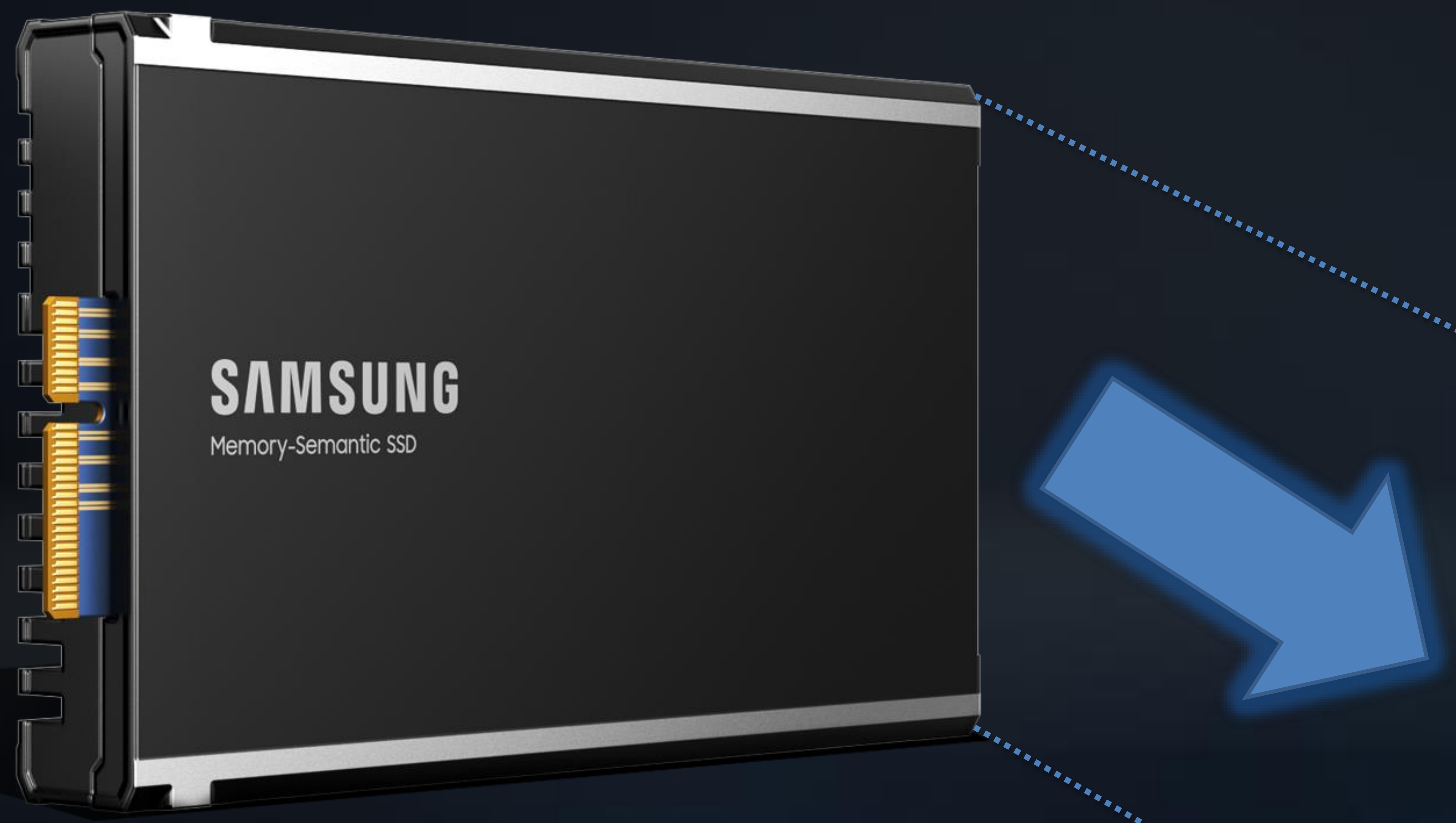
Dublin, Ireland September 12-14, 2022

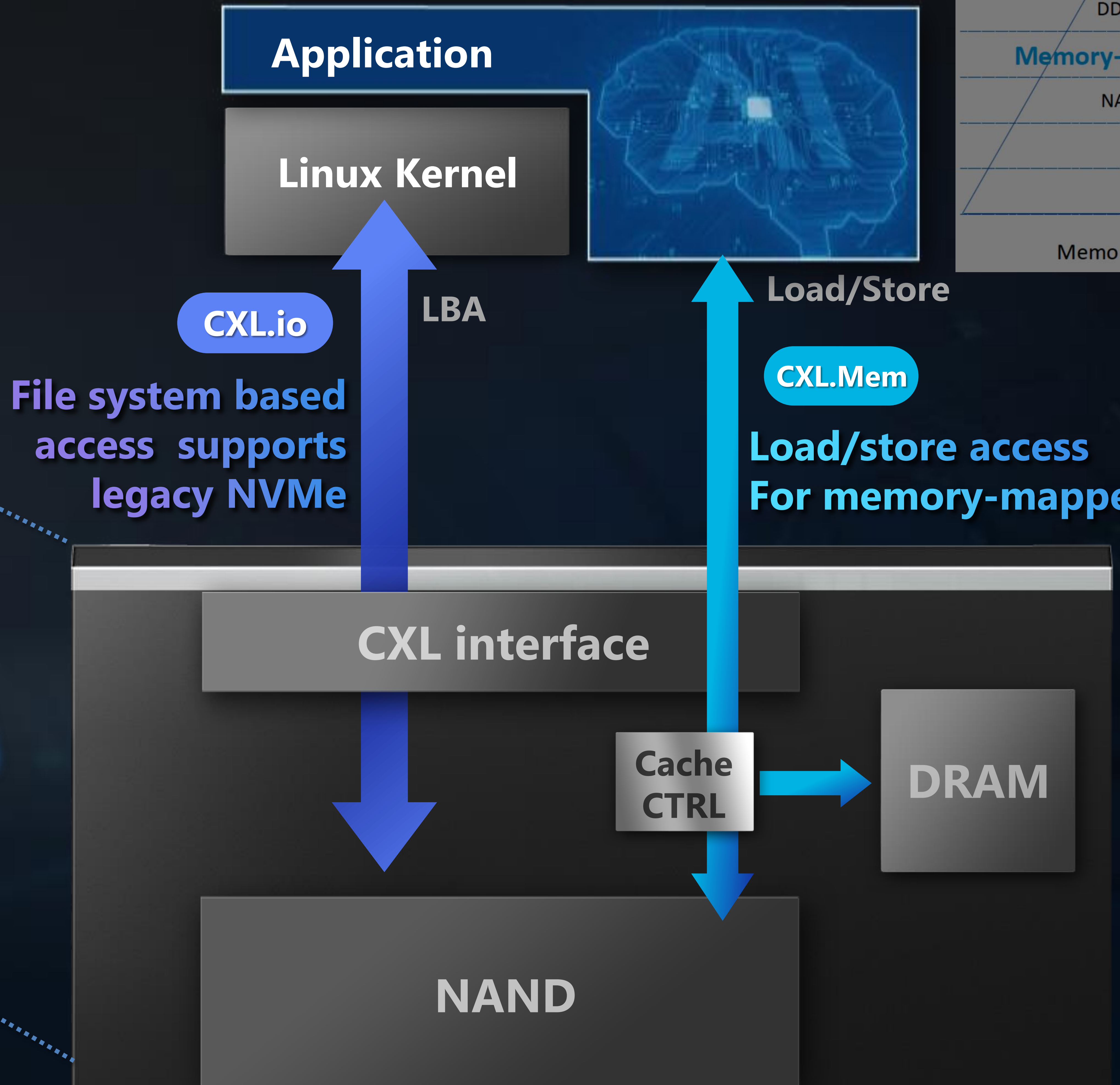
# reference

# Motivation: The advent of new types of storage device based on CXL

Dual mode support

Memory-Semantic CXLSSD



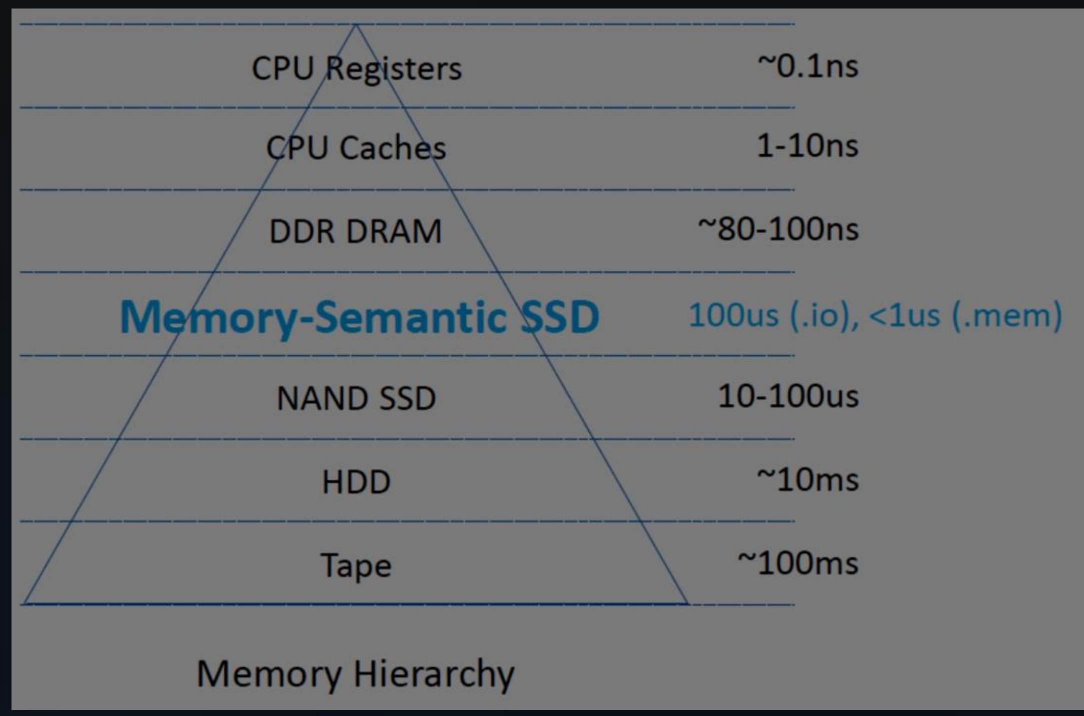


CXL.io

File system based access supports legacy NVMe

CXL.Mem

Load/store access For memory-mapped files

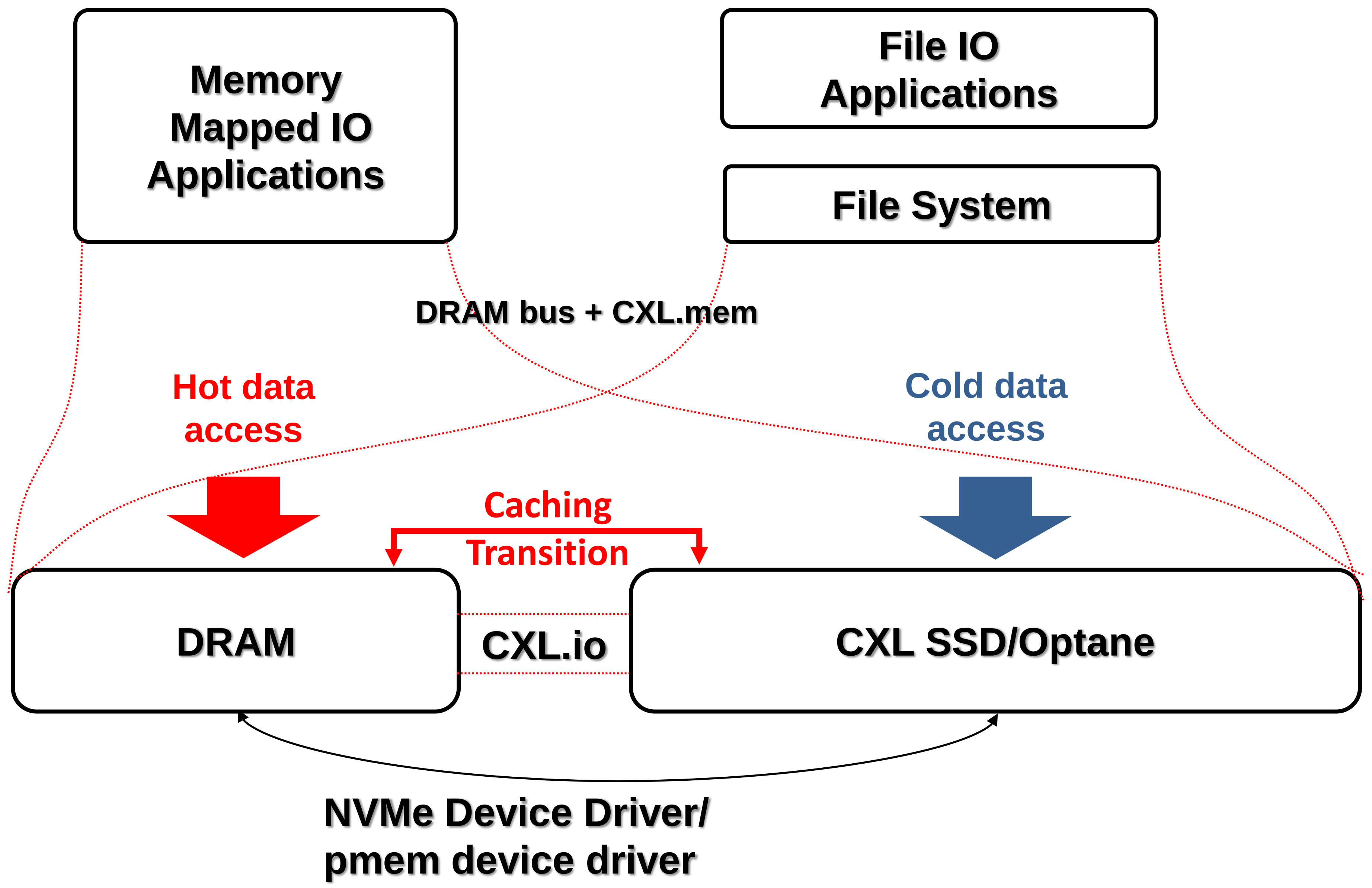


| | |
|---|---|
| CPU Registers | ~0.1ns |
| CPU Caches | 1-10ns |
| DDR DRAM | ~80-100ns |
| Memory-Semantic SSD | 100us (.io), <1us (.mem) |
| NAND SSD | 10-100us |
| HDD | ~10ms |
| Tape | ~100ms |

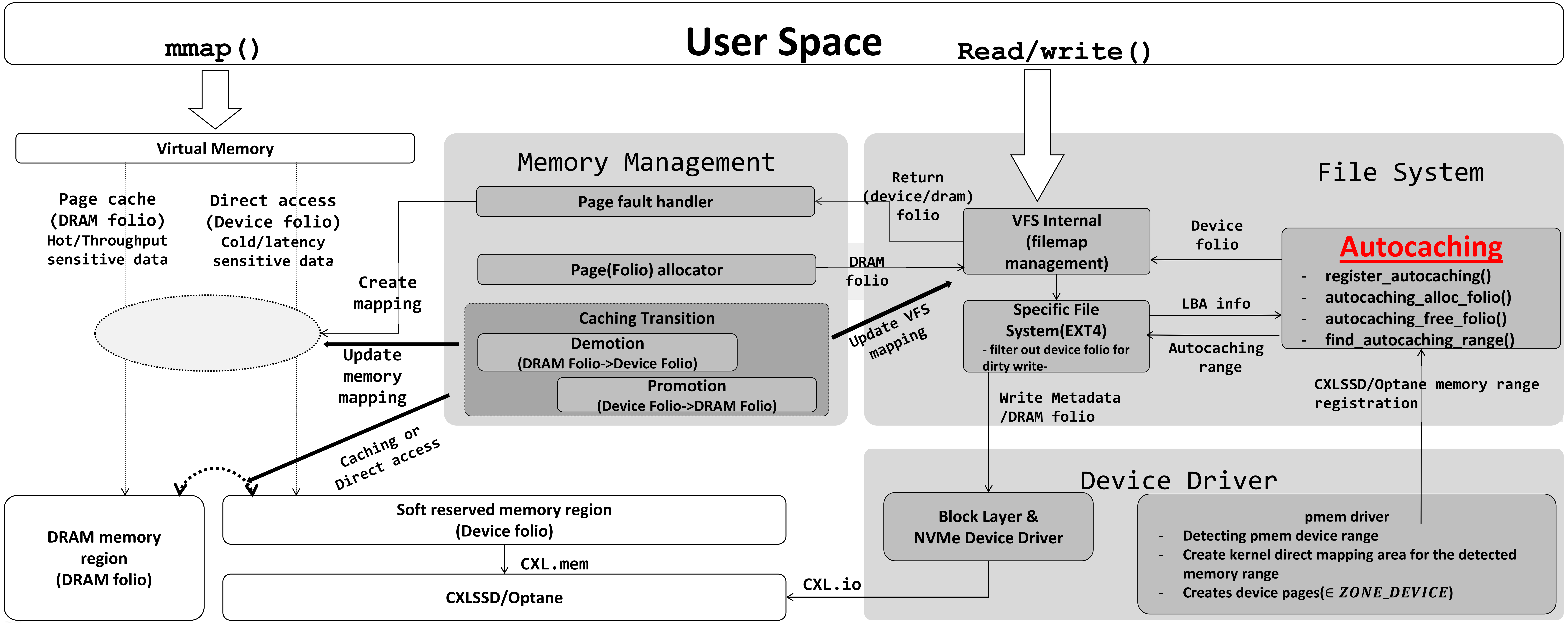
Memory Hierarchy

# What is Autocaching?

- Use page cache for hot data
- Use direct access for cold data



# Implementation of Autocaching



User Space

mmap()

Read/write()

Virtual Memory

Page cache
(DRAM folio)
Hot/Throughput
sensitive data

Direct access
(Device folio)
Cold/latency
sensitive data

Create
mapping

Update
memory
mapping

Caching or
Direct access

Memory Management

Page fault handler

Page(Folio) allocator

Caching Transition

Demotion
(DRAM Folio->Device Folio)

Promotion
(Device Folio->DRAM Folio)

Return
(device/dram)
folio

DRAM
folio

Update VFS
mapping

File System

VFS Internal
(filemap
management)

Specific File
System(EXT4)
- filter out device folio for
dirty write-

Device
folio

LBA info

Autocaching
range

**Autocaching**

- register_autocaching()
- autocaching_alloc_folio()
- autocaching_free_folio()
- find_autocaching_range()

CXLSSD/Optane memory range
registration

Write Metadata
/DRAM folio

Device Driver

Block Layer &
NVMe Device Driver

pmem driver

- Detecting pmem device range
- Create kernel direct mapping area for the detected memory range
- Creates device pages($\in \mathit{ZONE\_DEVICE}$)

DRAM memory
region
(DRAM folio)

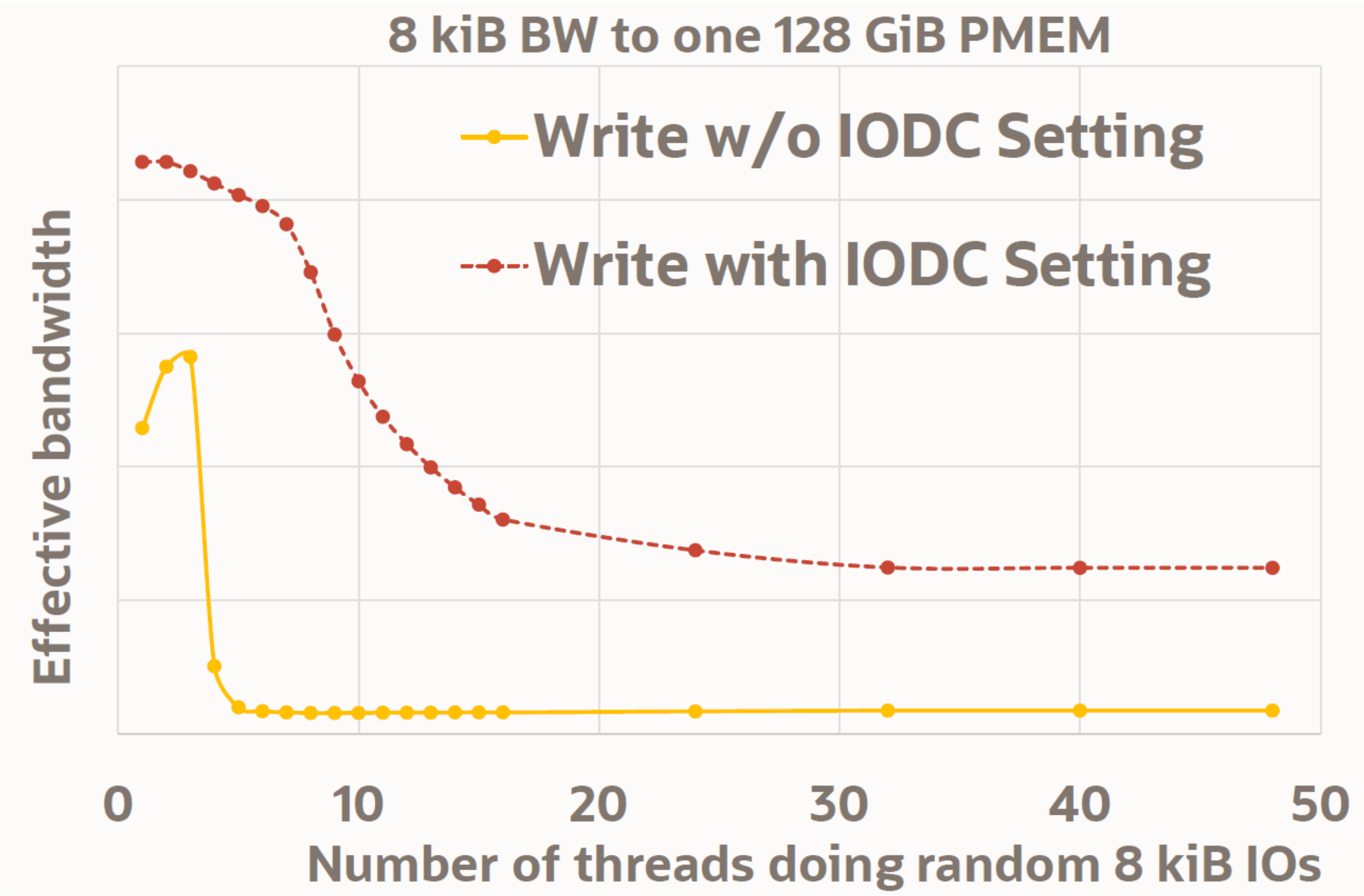
Soft reserved memory region
(Device folio)

CXL.mem

CXLSSD/Optane

CXL.io

# Write scales poorly with concurrent writes (from Oracle)

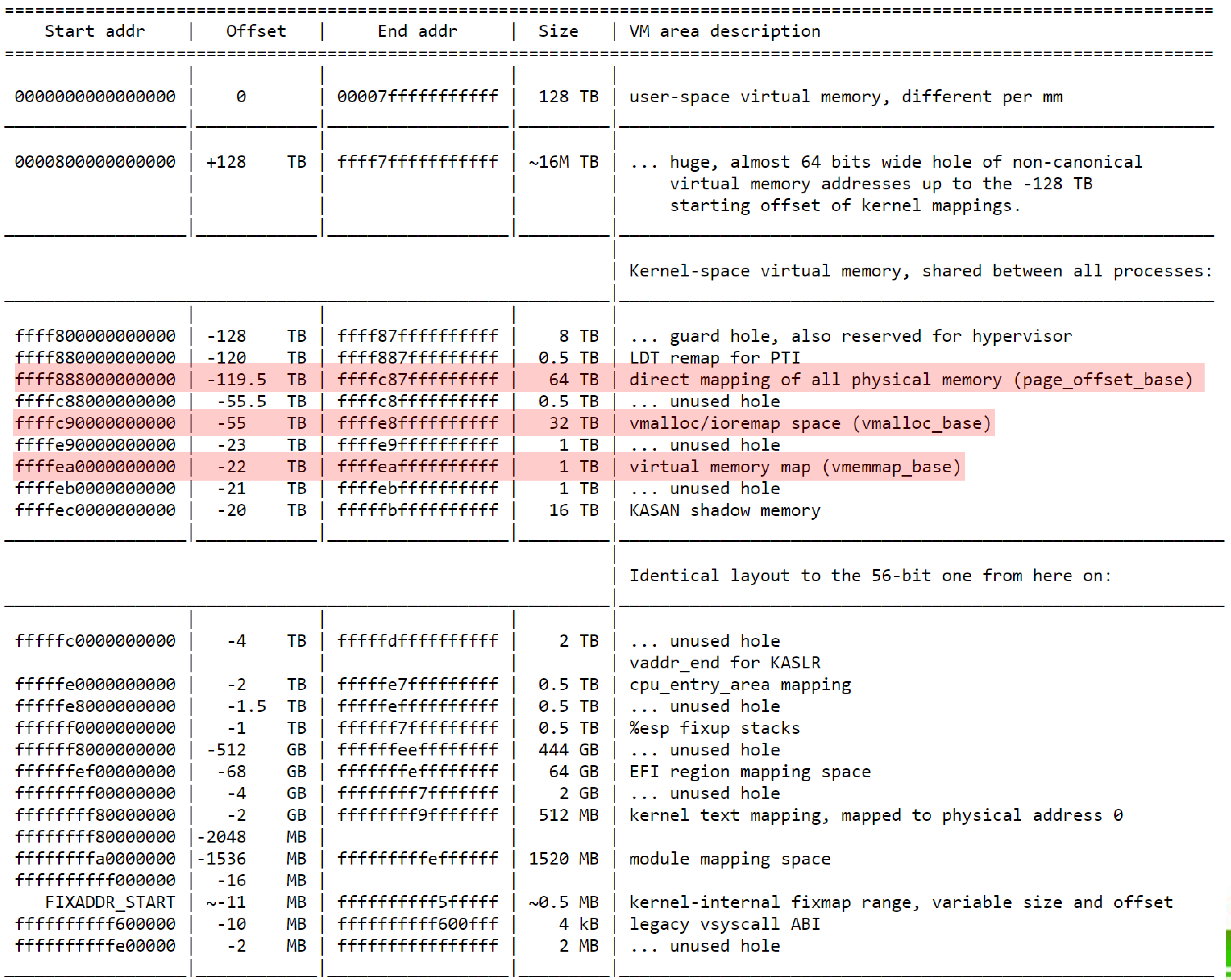


ORACLE

Under the Hood of an Exadata Transaction

How do we harness the power of Persistent Memory?

# Virtual Memory Map(Documentation/x86/x86_64/mm.rst)

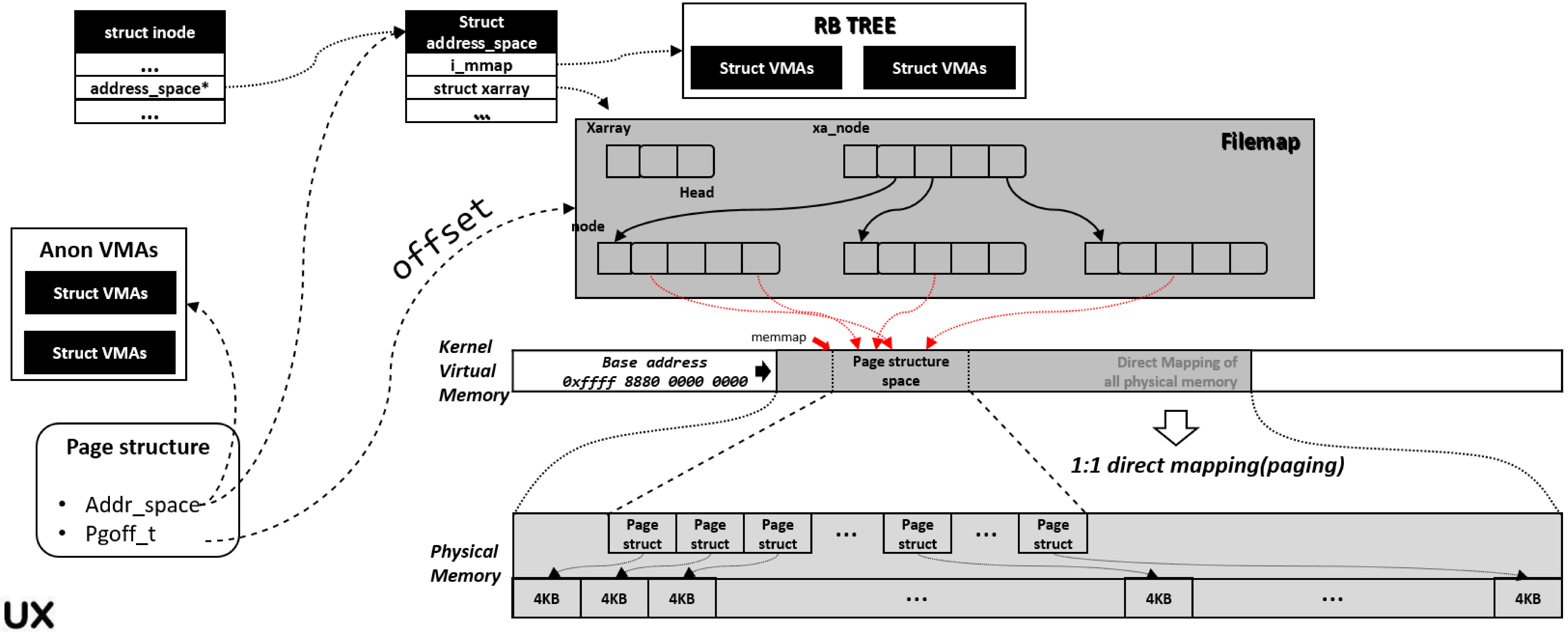


```
========================================================================================================================
    Start addr    |   Offset   |     End addr     |  Size   | VM area description
========================================================================================================================
                  |            |                  |         |
 0000000000000000 |    0       | 00007fffffffffff |  128 TB | user-space virtual memory, different per mm
__________________|____________|__________________|_________|___________________________________________________________
                  |            |                  |         |
 0000800000000000 | +128    TB | ffff7fffffffffff | ~16M TB | ... huge, almost 64 bits wide hole of non-canonical
                  |            |                  |         |     virtual memory addresses up to the -128 TB
                  |            |                  |         |     starting offset of kernel mappings.
__________________|____________|__________________|_________|___________________________________________________________
                                                            |
                                                            | Kernel-space virtual memory, shared between all processes:
____________________________________________________________|___________________________________________________________
                  |            |                  |         |
 ffff800000000000 | -128    TB | ffff87ffffffffff |    8 TB | ... guard hole, also reserved for hypervisor
 ffff880000000000 | -120    TB | ffff887fffffffff |  0.5 TB | LDT remap for PTI
 ffff888000000000 | -119.5  TB | ffffc87fffffffff |   64 TB | direct mapping of all physical memory (page_offset_base)
 ffffc88000000000 |  -55.5  TB | ffffc8ffffffffff |  0.5 TB | ... unused hole
 ffffc90000000000 |  -55    TB | ffffe8ffffffffff |   32 TB | vmalloc/ioremap space (vmalloc_base)
 ffffe90000000000 |  -23    TB | ffffe9ffffffffff |    1 TB | ... unused hole
 ffffea0000000000 |  -22    TB | ffffeaffffffffff |    1 TB | virtual memory map (vmemmap_base)
 ffffeb0000000000 |  -21    TB | ffffebffffffffff |    1 TB | ... unused hole
 ffffec0000000000 |  -20    TB | fffffbffffffffff |   16 TB | KASAN shadow memory
__________________|____________|__________________|_________|____________________________________________________________
                                                            |
                                                            | Identical layout to the 56-bit one from here on:
____________________________________________________________|____________________________________________________________
                  |            |                  |         |
 fffffc0000000000 |   -4    TB | fffffdffffffffff |    2 TB | ... unused hole
                  |            |                  |         | vaddr_end for KASLR
 fffffe0000000000 |   -2    TB | fffffe7fffffffff |  0.5 TB | cpu_entry_area mapping
 fffffe8000000000 |   -1.5  TB | fffffeffffffffff |  0.5 TB | ... unused hole
 ffffff0000000000 |   -1    TB | ffffff7fffffffff |  0.5 TB | %esp fixup stacks
 ffffff8000000000 | -512    GB | ffffffeeffffffff |  444 GB | ... unused hole
 ffffffef00000000 |  -68    GB | fffffffeffffffff |   64 GB | EFI region mapping space
 ffffffff00000000 |   -4    GB | ffffffff7fffffff |    2 GB | ... unused hole
 ffffffff80000000 |   -2    GB | ffffffff9fffffff |  512 MB | kernel text mapping, mapped to physical address 0
 ffffffff80000000 |-2048    MB |                  |         |
 ffffffffa0000000 |-1536    MB | fffffffffeffffff | 1520 MB | module mapping space
 ffffffffff000000 |  -16    MB |                  |         |
    FIXADDR_START | ~-11    MB | ffffffffff5fffff | ~0.5 MB | kernel-internal fixmap range, variable size and offset
 ffffffffff600000 |  -10    MB | ffffffffff600fff |    4 kB | legacy vsyscall ABI
 ffffffffffe00000 |   -2    MB | ffffffffffffffff |    2 MB | ... unused hole
__________________|____________|__________________|_________|___________________________________________________________
```

12-14, 2022

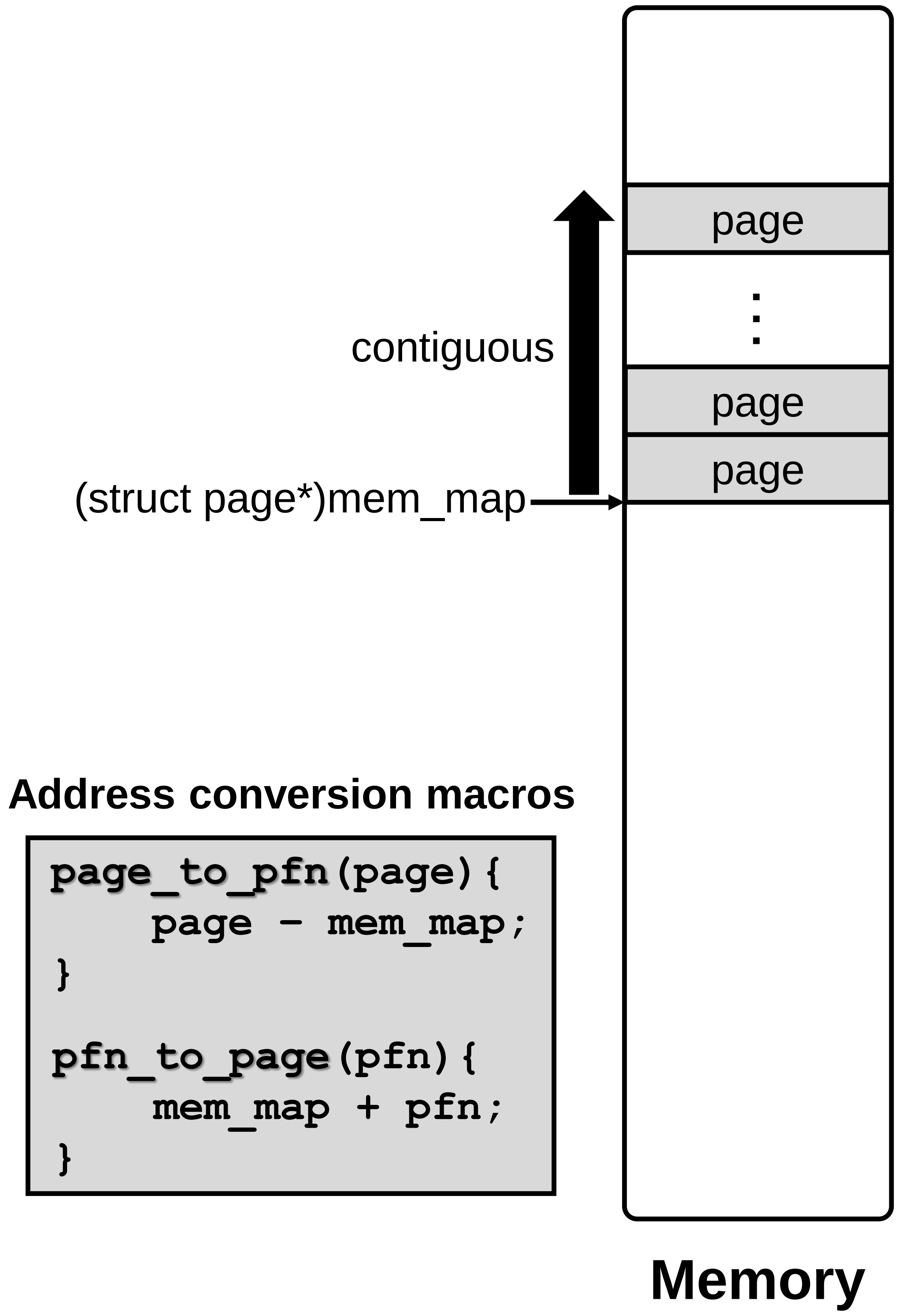
# Implementation of Autocaching

- Page structure is used for reverse mapping for caching transition



# Implementation of Autocaching

- **Dynamic memory allocation of struct page**
  - Only active files use struct page
    - Does not used for block allocation(free page management)
      - File system allocates blocks using its block allocator.
  - Challenge
    - Address conversion: page_to_pfn(), pfn_to_page()
      - Dynamically allocated pages may not be in contiguous memory.
    - VFS manages file in 4KB unit(nodes in filemap(xarray) of VFS).

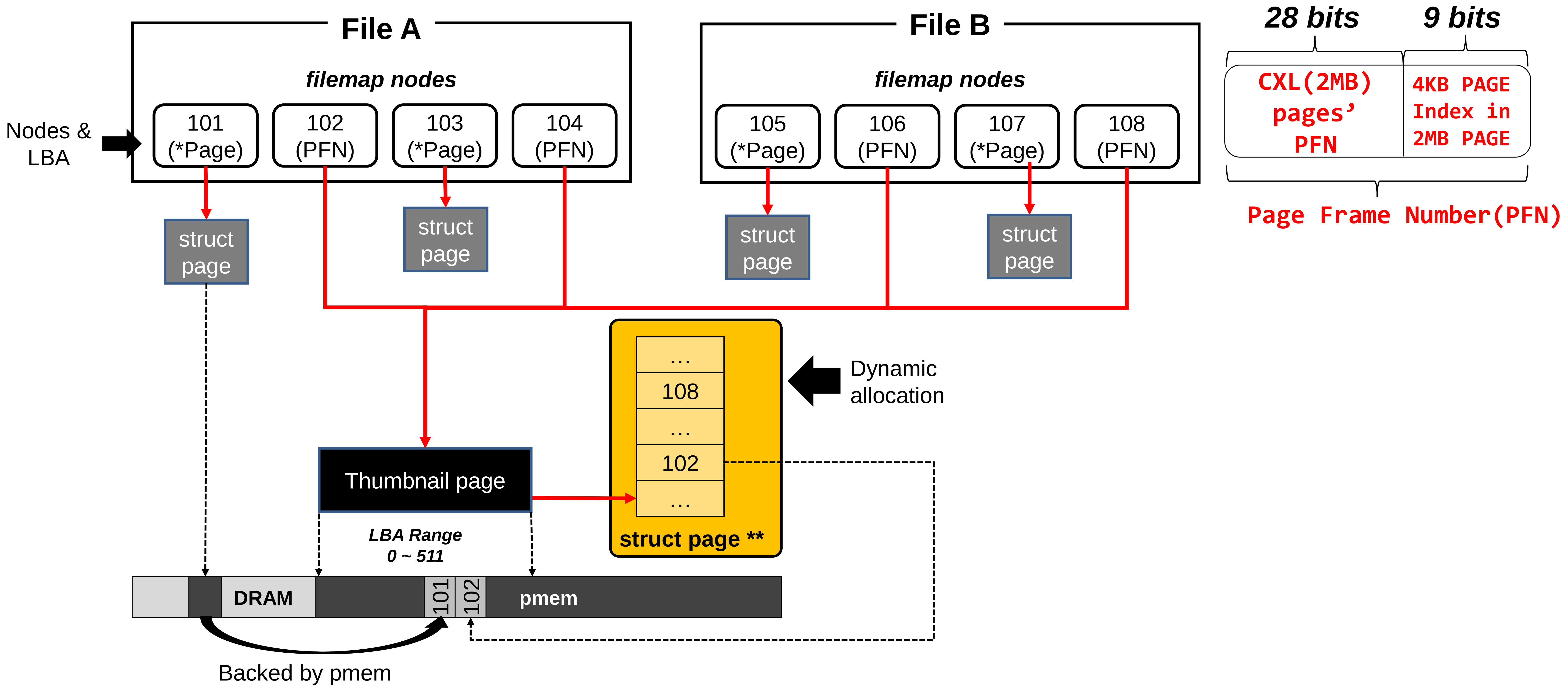


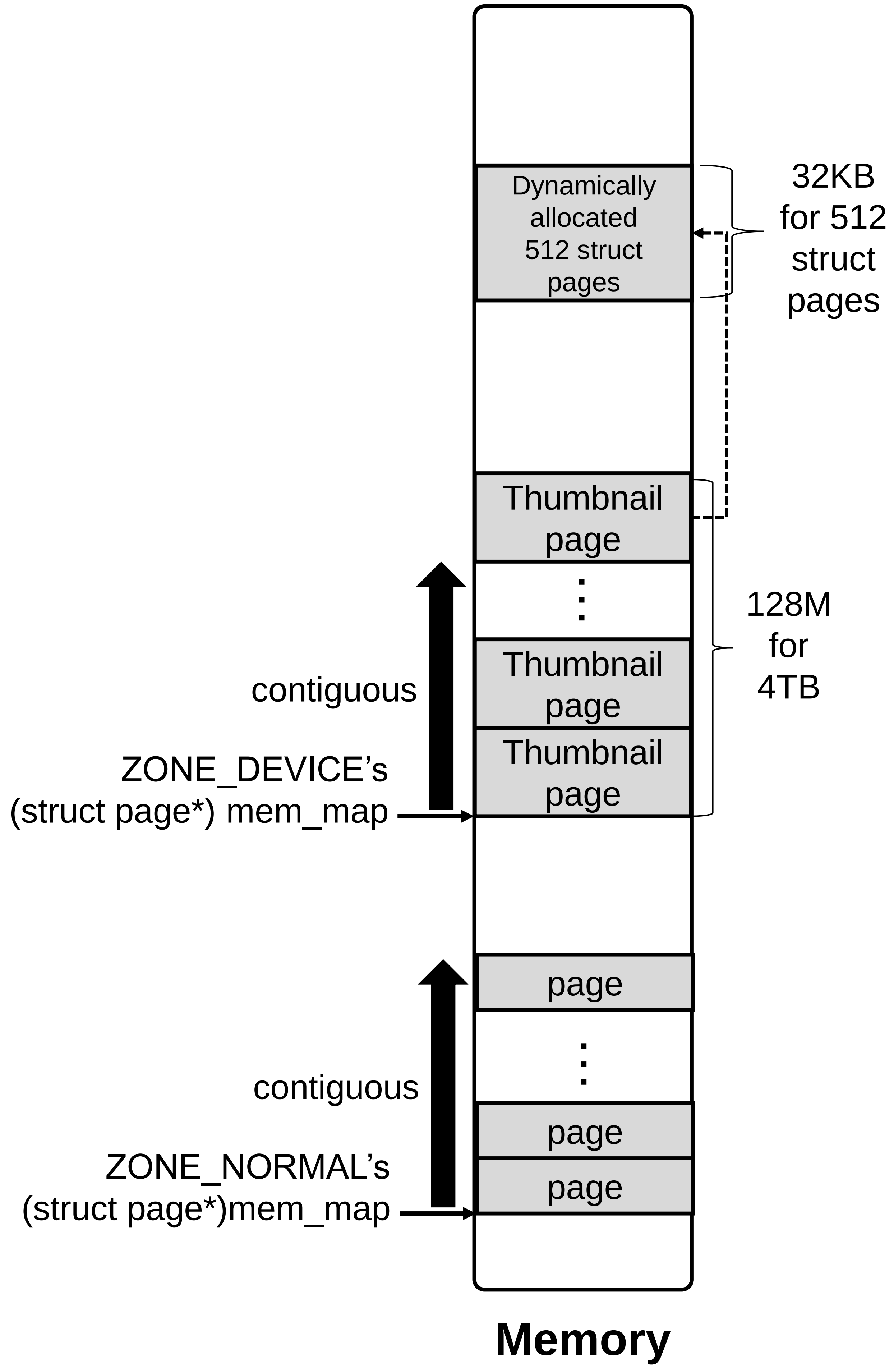
**Address conversion macros**

```
page_to_pfn(page){
    page – mem_map;
}

pfn_to_page(pfn){
    mem_map + pfn;
}
```

# Implementation of Autocaching

- **`Thumbnail struct page`**
  - Dynamic memory allocation of struct page
  - Reserve memory for cxl_pages(40KB for 4TB)

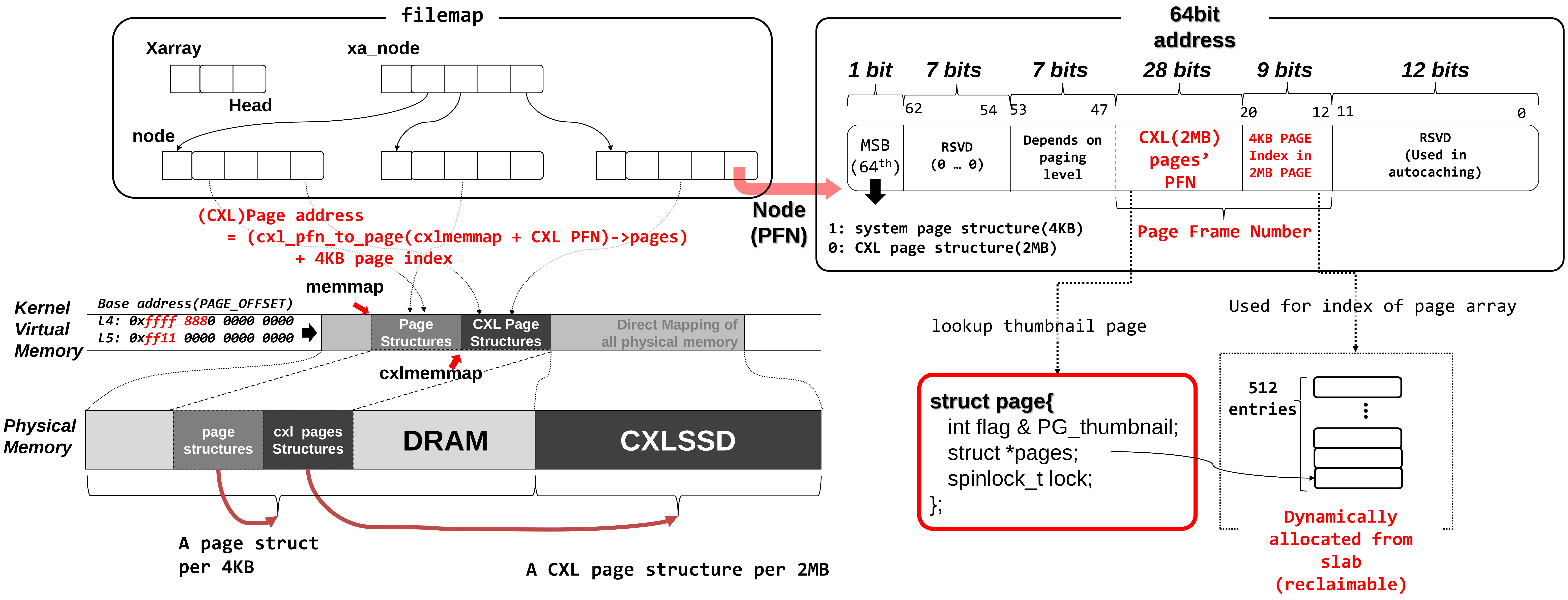




Dynamically allocated 512 struct pages
32KB for 512 struct pages
Thumbnail page
⋮
Thumbnail page
Thumbnail page
128M for 4TB
contiguous
ZONE_DEVICE's (struct page*) mem_map
page
⋮
page
page
contiguous
ZONE_NORMAL's (struct page*)mem_map
Memory

# Implementation of Autocaching

- **`Thumbnail struct page`** in detail



filemap

Xarray

Head

xa_node

node

(CXL)Page address
= (cxl_pfn_to_page(cxlmemmap + CXL PFN)->pages)
+ 4KB page index

Node (PFN)

64bit address

| 1 bit | 7 bits | 7 bits | 28 bits | 9 bits | 12 bits |
|---|---|---|---|---|---|
| | 62 54 | 53 47 | | 20 12 | 11 0 |
| MSB ($64^{th}$) | RSVD (0 … 0) | Depends on paging level | CXL(2MB) pages' PFN | 4KB PAGE Index in 2MB PAGE | RSVD (Used in autocaching) |

Page Frame Number

1: system page structure(4KB)
0: CXL page structure(2MB)

lookup thumbnail page

Used for index of page array

memmap

*Kernel Virtual Memory*

*Base address(PAGE_OFFSET)*
*L4: 0xffff 8880 0000 0000*
*L5: 0xff11 0000 0000 0000*

Page Structures

CXL Page Structures

Direct Mapping of all physical memory

cxlmemmap

*Physical Memory*

page structures

cxl_pages Structures

DRAM

CXLSSD

A page struct
per 4KB

A CXL page structure per 2MB

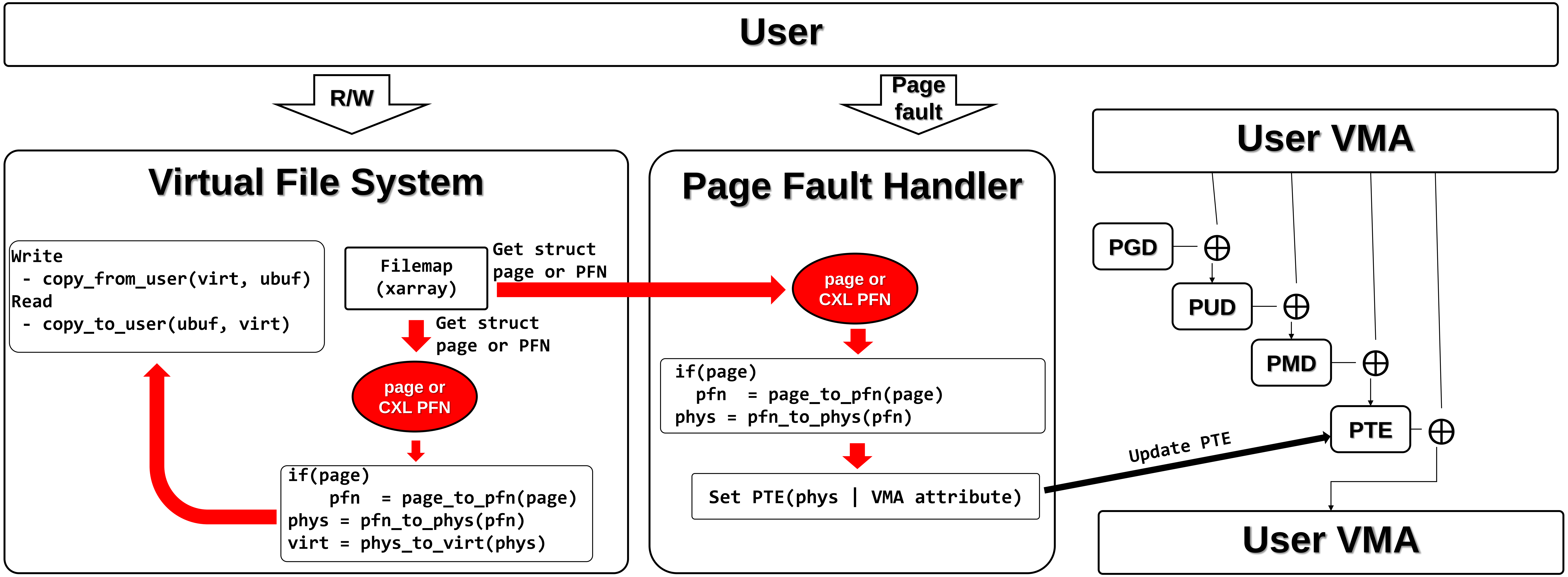
```
struct page{
    int flag & PG_thumbnail;
    struct *pages;
    spinlock_t lock;
};
```

512 entries

Dynamically allocated from slab (reclaimable)

# Implementation of Autocaching

- Address conversion from CXL page to physical address



User

R/W

Page fault

Virtual File System

```
Write
 - copy_from_user(virt, ubuf)
Read
 - copy_to_user(ubuf, virt)
```

Filemap (xarray)

Get struct page or PFN

Get struct page or PFN

page or CXL PFN

```
if(page)
    pfn  = page_to_pfn(page)
phys = pfn_to_phys(pfn)
virt = phys_to_virt(phys)
```

Page Fault Handler

page or CXL PFN

```
if(page)
  pfn  = page_to_pfn(page)
phys = pfn_to_phys(pfn)
```

```
Set PTE(phys | VMA attribute)
```

Update PTE

User VMA

PGD ⊕

PUD ⊕

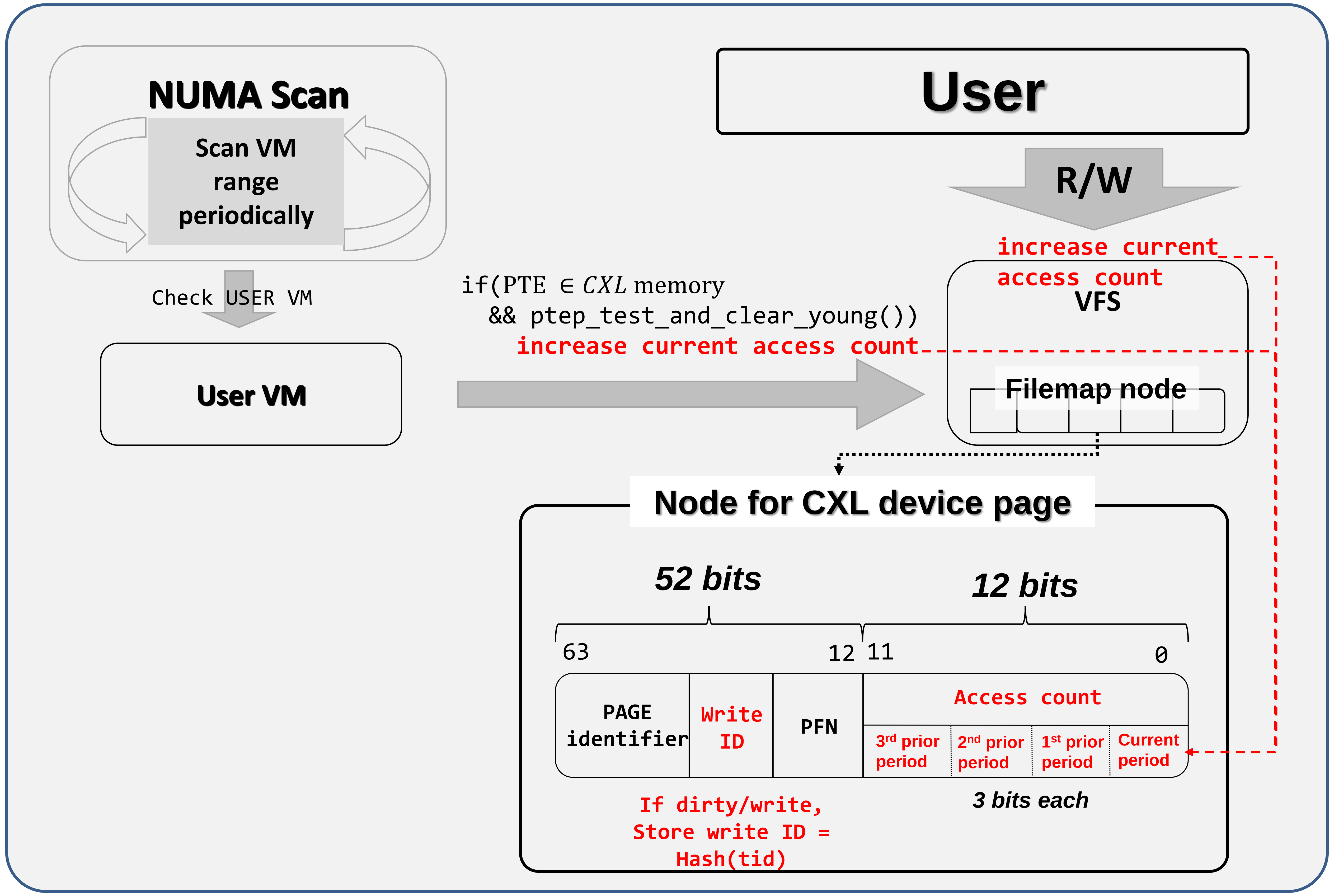
PMD ⊕

PTE ⊕

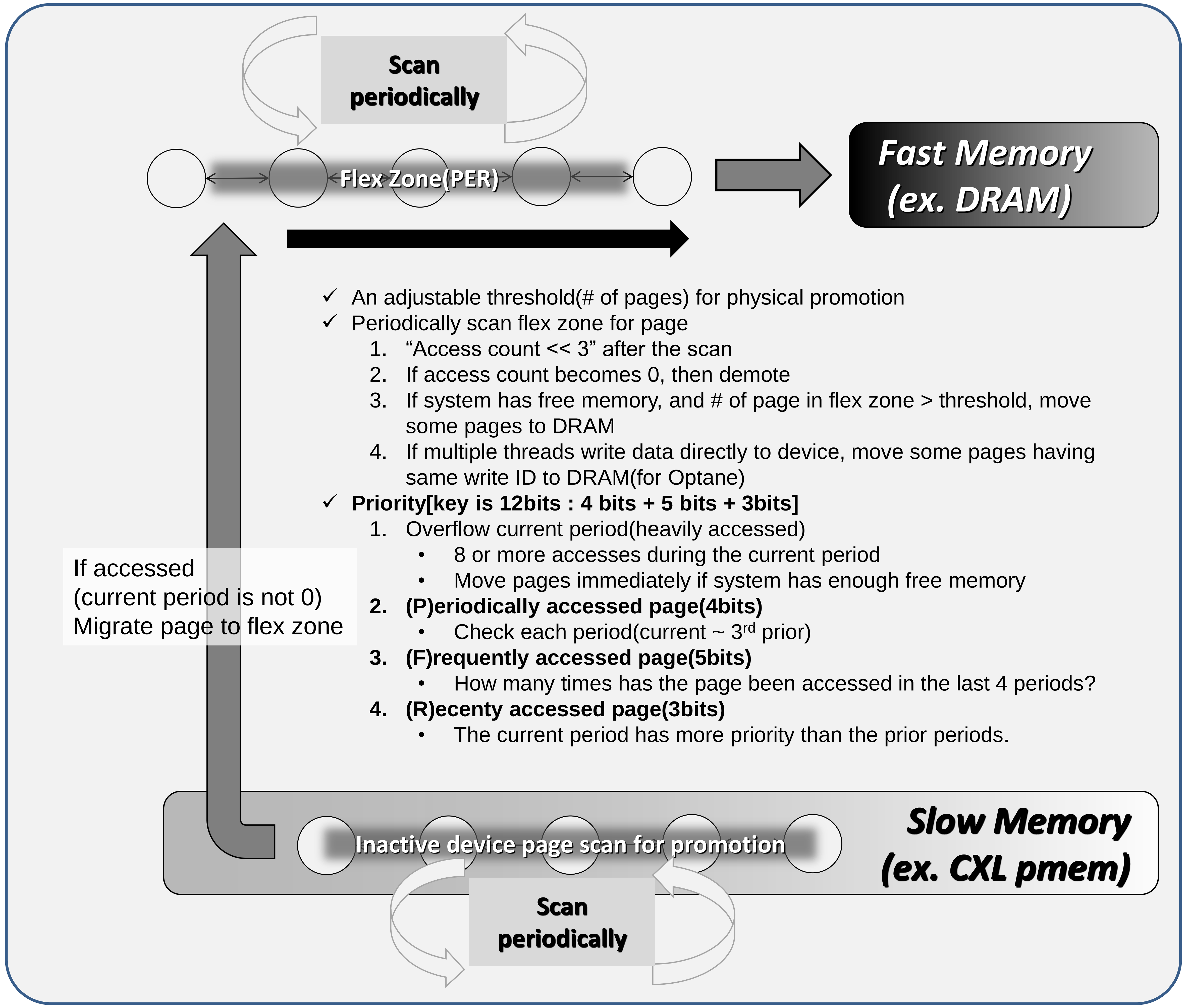
User VMA

# Implementation of Autocaching

- Promotion



Step1. Increasing Access count


Scan periodically
Flex Zone(PER)
Fast Memory (ex. DRAM)
If accessed
(current period is not 0)
Migrate page to flex zone
Inactive device page scan for promotion
Slow Memory (ex. CXL pmem)
Scan periodically

- ✓ An adjustable threshold(# of pages) for physical promotion
- ✓ Periodically scan flex zone for page
  1. "Access count << 3" after the scan
  2. If access count becomes 0, then demote
  3. If system has free memory, and # of page in flex zone > threshold, move some pages to DRAM
  4. If multiple threads write data directly to device, move some pages having same write ID to DRAM(for Optane)
- ✓ **Priority[key is 12bits : 4 bits + 5 bits + 3bits]**
  1. Overflow current period(heavily accessed)
     - 8 or more accesses during the current period
     - Move pages immediately if system has enough free memory
  2. **(P)eriodically accessed page(4bits)**
     - Check each period(current ~ 3rd prior)
  3. **(F)requently accessed page(5bits)**
     - How many times has the page been accessed in the last 4 periods?
  4. **(R)ecenty accessed page(3bits)**
     - The current period has more priority than the prior periods.

Step 2. Promotion based on access count