12. Persistent Memory

Special Topics in Computer Systems:
Modern Storage Systems
(IC820-01)

Instructor:
Prof. Sungjin Lee (sungjin.lee@dgist.ac.kr)

Outline

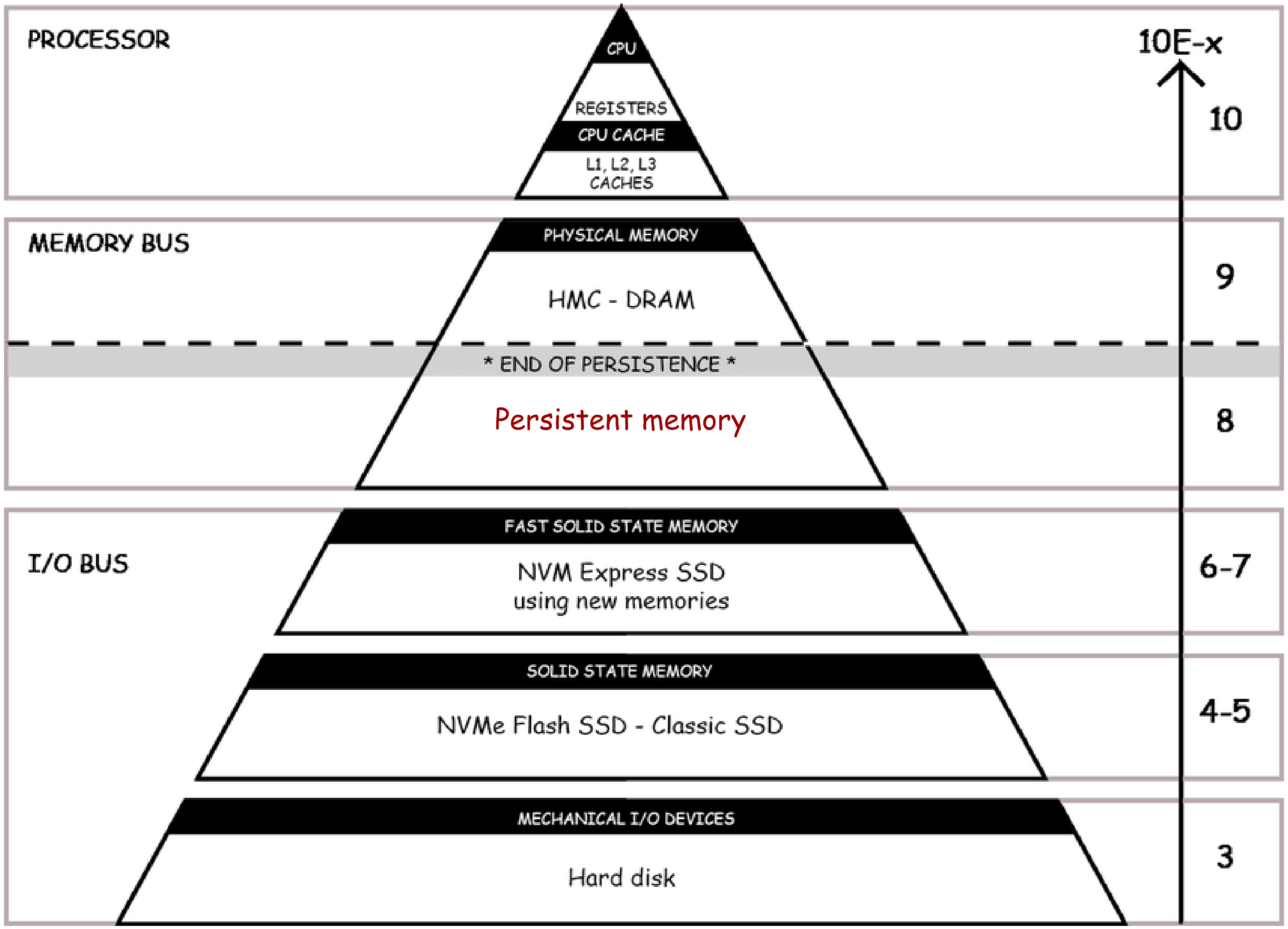
- What is Persistent Memory
- Memory Mode
- App Direct Mode

Persistent Memory

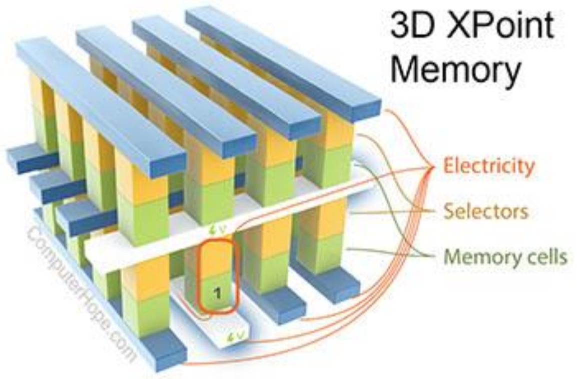
- **Byte addressable stable memory devices**
  - ***Phase Change (PCM)***
    - Crystalline material heated to two different phases
  - ***Spin Torque Transfer (STT-RAM, MRAM)***
    - Magnetic spin captured in ferrous element
  - ***ReRAM***
    - Resistive RAM

- **Persistent memory will fit a niche between DRAM and NAND**
  - Power advantages over DRAM because there is no “refresh” cycle necessary to preserve logic value
  - Cost disadvantages over NAND because block-addressing makes NAND more compact

System Hierarchy with Persistent Memory



Intel’s Optane (or 3D X-Point)



- **New Technology from Intel/Micron**
  - Similar to Phase Change and Resistive Memories
  - Bit addressable
  - No Erase requirement
- **‘~1000 faster’ than NAND**
  - But still 3x to 5x slower than DRAM
- **Process has a trajectory to catch NAND density**
  - This is a big deal, first parts are ***128, 256, and 512 GB***!
- **Much improved durability**
  - However, even at 100 million write cycles, you could wear it out in seconds w/out wear leveling
- **Products are memory (DIMM) or storage devices (PCIe)**

Intel's Optane (or 3D X-Point) (Cont.)

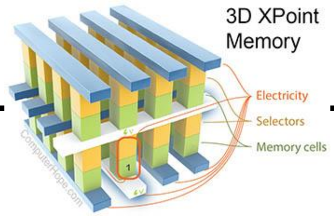
- Optane^{TM} DC Persistent Memory Module (Optane DC PMM)

Memory attributes

Performance comparable to DRAM at low latencies

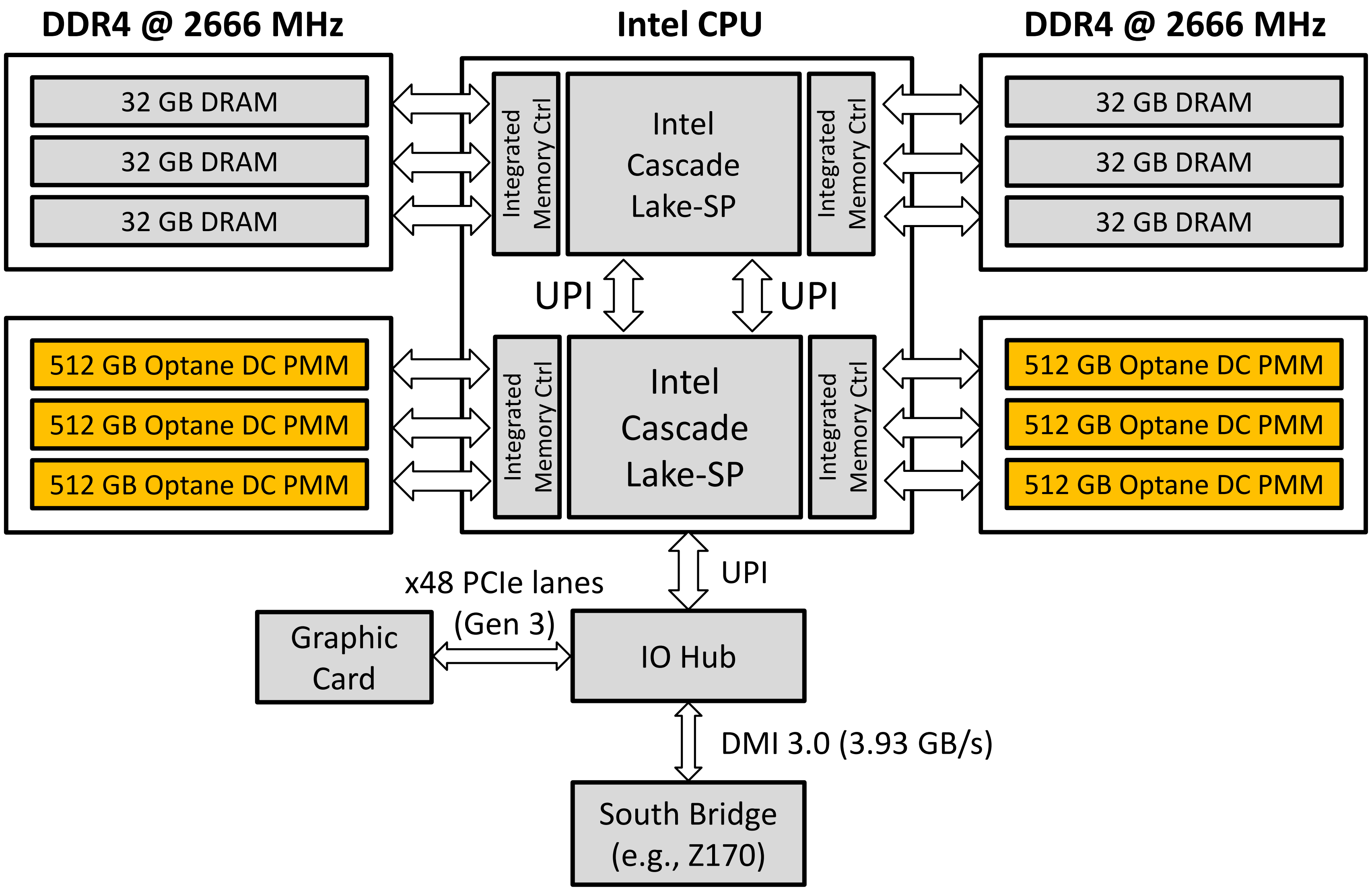
Storage attributes

Data persistence with Higher capacity than DRAM

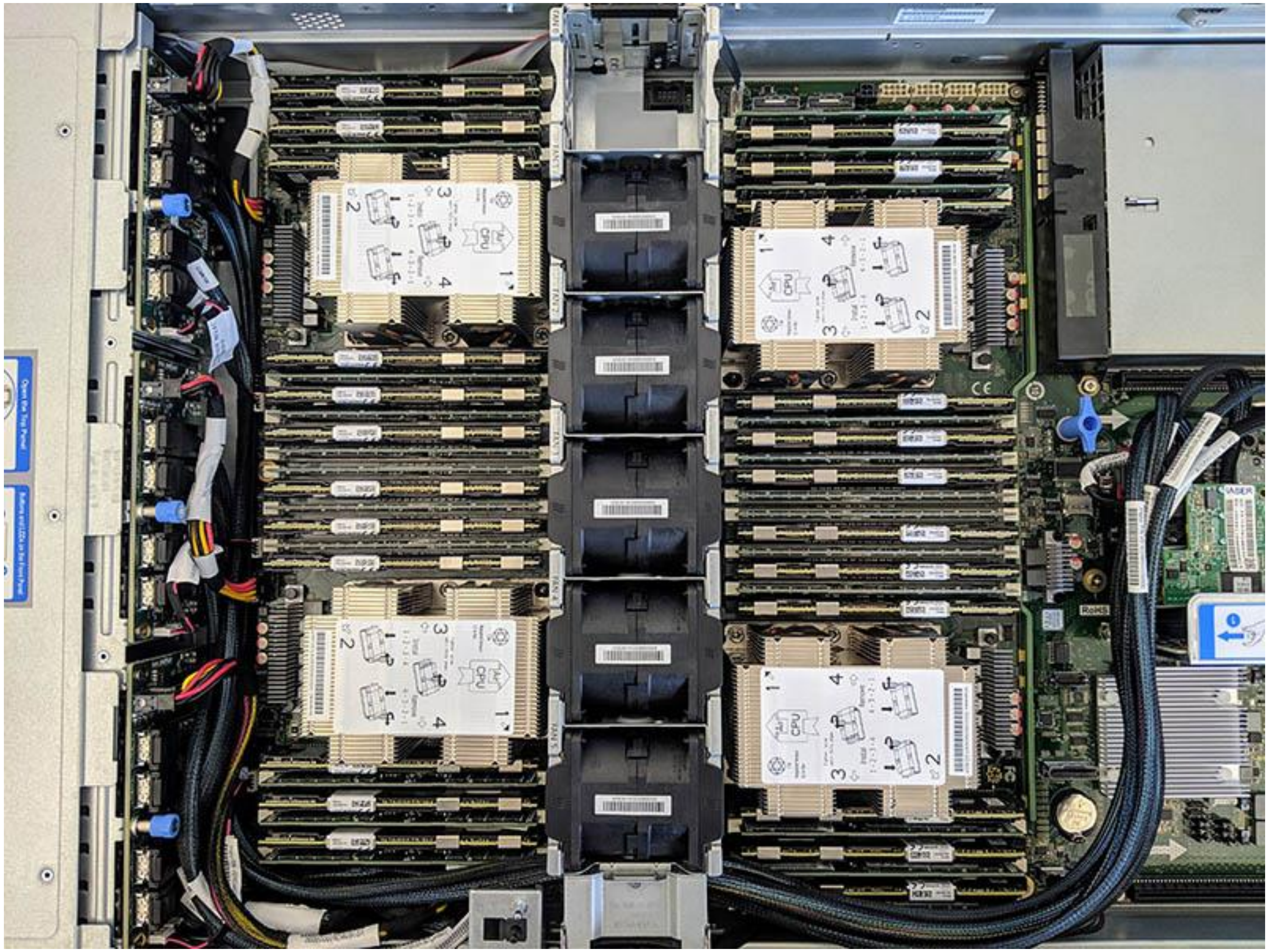


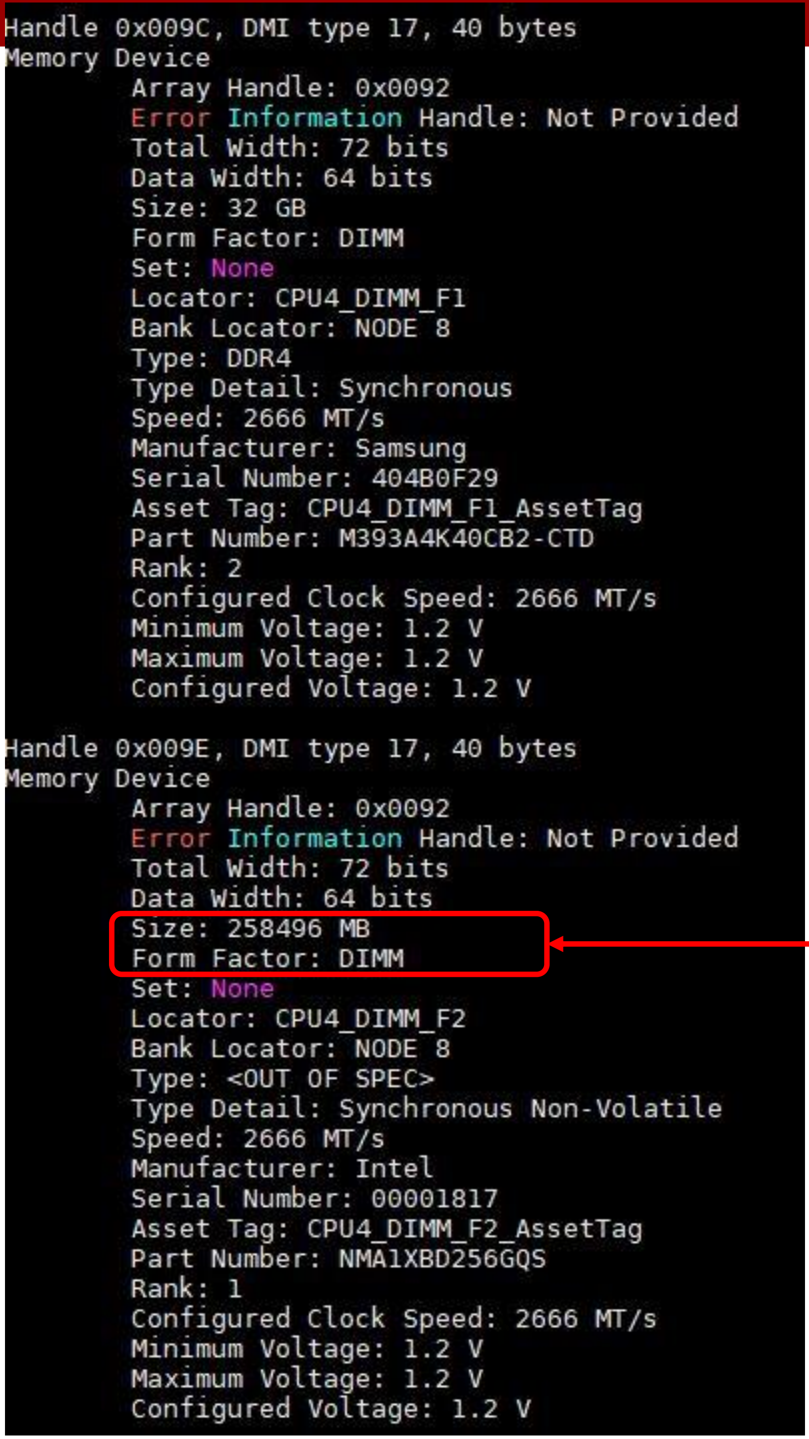
	DRAM	Optane DC PMM (6x)
Read Latency	81 ns	306 ns
Write Latency	86 ns	94 ns
Read Throughput	120 GB/s	39.4 GB/s
Write Throughput	80 GB/s	13.9 GB/s

Storage Bandwidth Hierarchy



Intel Xeon Scalable with 6TB of Optane DC



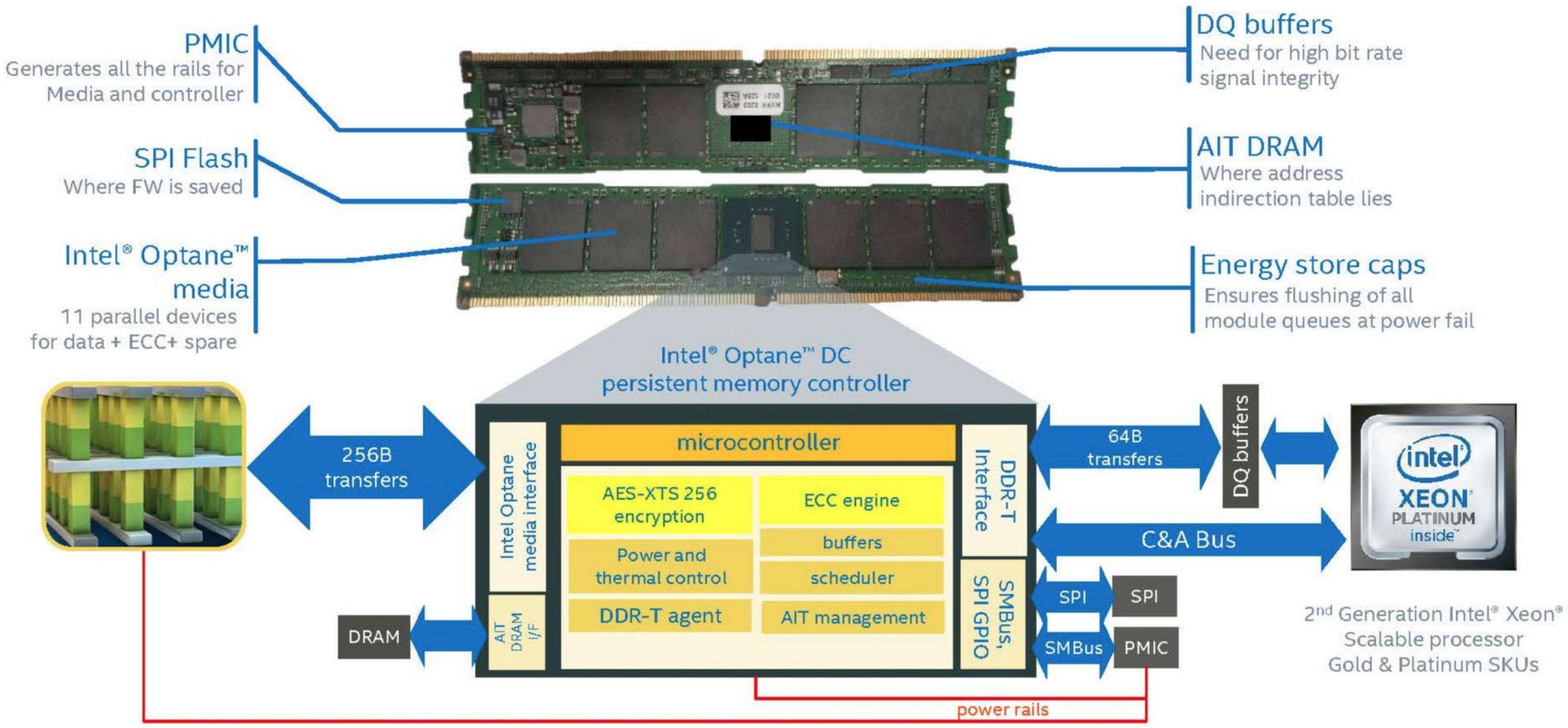


```
Handle 0x009C, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0092
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 32 GB
        Form Factor: DIMM
        Set: None
        Locator: CPU4_DIMM_F1
        Bank Locator: NODE 8
        Type: DDR4
        Type Detail: Synchronous
        Speed: 2666 MT/s
        Manufacturer: Samsung
        Serial Number: 404B0F29
        Asset Tag: CPU4_DIMM_F1_AssetTag
        Part Number: M393A4K40CB2-CTD
        Rank: 2
        Configured Clock Speed: 2666 MT/s
        Minimum Voltage: 1.2 V
        Maximum Voltage: 1.2 V
        Configured Voltage: 1.2 V

Handle 0x009E, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0092
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 258496 MB
        Form Factor: DIMM
        Set: None
        Locator: CPU4_DIMM_F2
        Bank Locator: NODE 8
        Type: <OUT OF SPEC>
        Type Detail: Synchronous Non-Volatile
        Speed: 2666 MT/s
        Manufacturer: Intel
        Serial Number: 00001817
        Asset Tag: CPU4_DIMM_F2_AssetTag
        Part Number: NMA1XBD256GQS
        Rank: 1
        Configured Clock Speed: 2666 MT/s
        Minimum Voltage: 1.2 V
        Maximum Voltage: 1.2 V
        Configured Voltage: 1.2 V
```

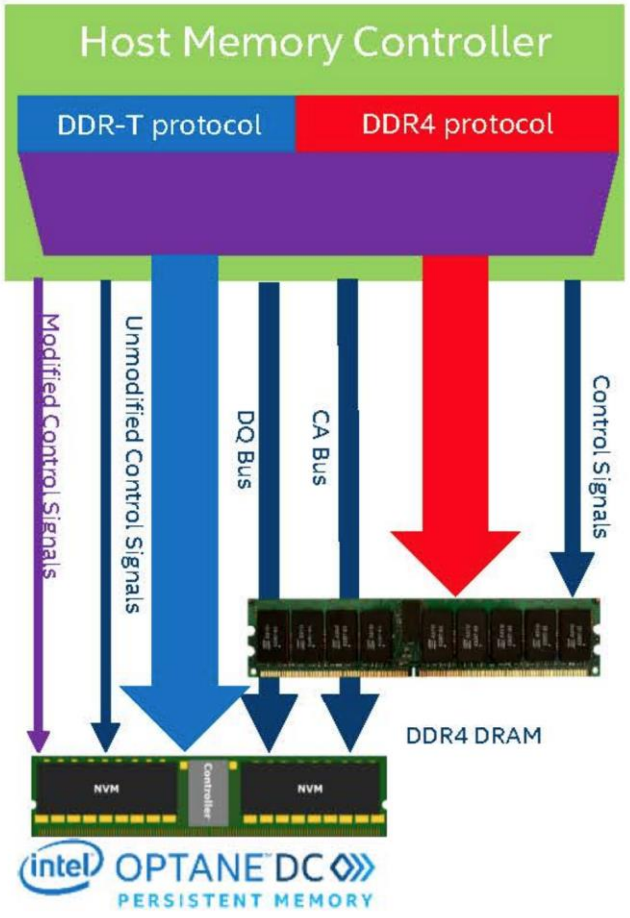
CPU recognizes it
as **256 GB DIMM DRAM**
(Memory mode)

Optane Architecture


PMIC
Generates all the rails for Media and controller
SPI Flash
Where FW is saved
Intel® Optane™ media
11 parallel devices for data + ECC+ spare
DQ buffers
Need for high bit rate signal integrity
AIT DRAM
Where address indirection table lies
Energy store caps
Ensures flushing of all module queues at power fail
[redacted]
Intel® Optane™ DC persistent memory controller
256B transfers
Intel Optane media interface
AIT DRAM I/F
microcontroller
AES-XTS 256 encryption
ECC engine
buffers
Power and thermal control
scheduler
DDR-T agent
AIT management
DDR-T Interface
SMBus, SPI GPIO
DRAM
64B transfers
DQ buffers
C&A Bus
SPI
SPI
SMBus
PMIC
power rails
intel
XEON
PLATINUM
inside
2nd Generation Intel® Xeon® Scalable processor Gold & Platinum SKUs

DDR-T Protocol

- **DDR protocol on top of electrical/mechanical interface for DDR4**
  - DDR DRAM operate synchronously which are not suitable for Optane media
- **DDR-T allow for *asynchronous* cmd/data timing**
  - Controller uses request/grant scheme to communicate with host controller
  - Data bus direction and timing controlled by host
  - Command packet per request sent from host to Optane DC PMM
- **Transaction can be re-ordered**
- **64B cache line access granularity (similar to DDR4)**



Address Indirection Table (AIT)

- **Optane DC PMM media has limited write endurance like NAND flash**
  - Internal address translation is necessary for wear-leveling and bad-block management

- ***Address Indirection Table*** **(AIT)**
  - Translate the DIMM physical address to an internal Optane DC media device address (similar to what FTLs do in SSDs)
  - The AIT table resides in Optane media, though on-DIMM DRAM keeps a copy of the AIT entries
  - The granularity of Optane DC media is 256 bytes, while a cache line size is 64 bytes
    - Write amplification occurs – smaller stores are handled as read-modify-write operations by the controller

Outline

- What is Persistent Memory
- Memory Mode
- App Direct Mode

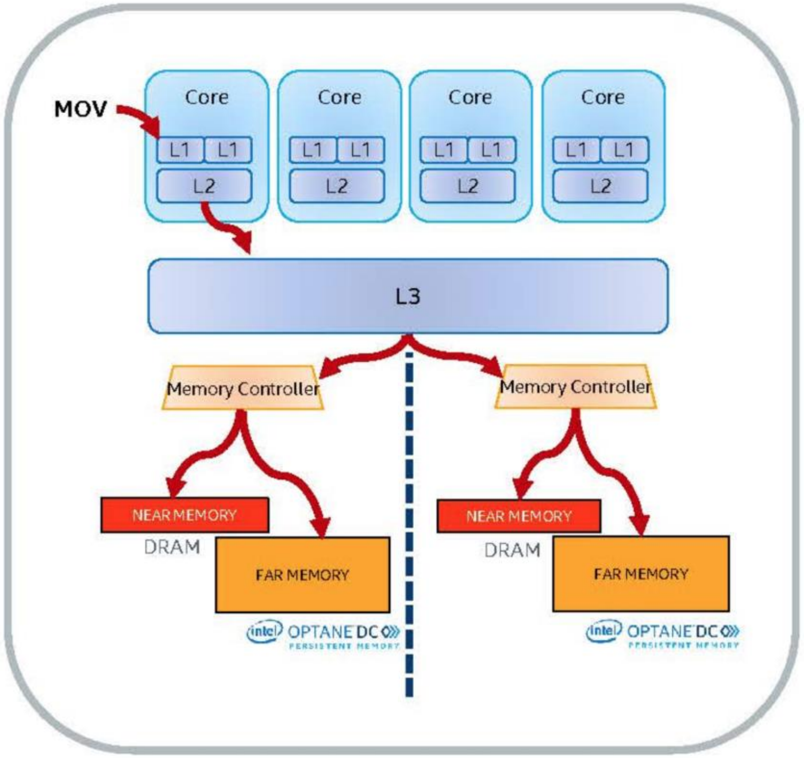
Two Modes of Optane DC PMM

- 1. ***Memory Mode***
  - DDR4 DIMMs operate as caches for slower Optane DC PMM
    - Direct mapped write-back cache with 64B cache lines
  - Optane DC PMMs are exposed as a large "*volatile*" memory
  - Do not have persistence

- 2. ***App Direct Mode***
  - Optane DC PMMs are directly exposed to the CPU and OS as "persistent" storage
  - Both Optane DC PMMs and DDR4 DIMMs are visible to OS as memory devices
  - Optane DC PMM is exposed as configurable regions on contiguously-addressed ranges

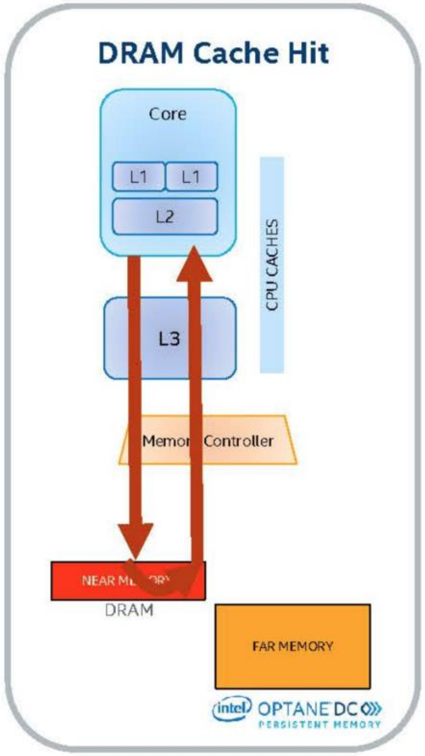
Memory Mode Details

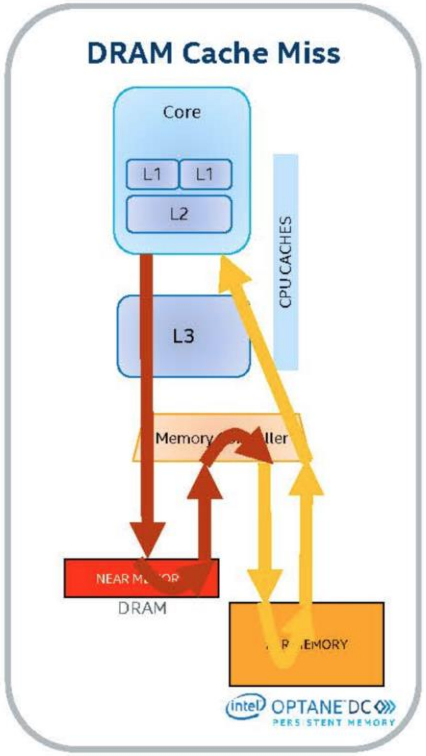
- No software/application changes required
- To mimic traditional memory, data is "volatile"
- DRAM is "*near memory*"
  - Used as a write-back cache
  - Managed by host memory controller
  - Within the same host memory controller, not across
- Optane DC PMM is "*far memory*"
  - Managed by host software
- Ratio of far/near memory (PMEM/DRAM) can vary



Memory Mode Details (Cont.)

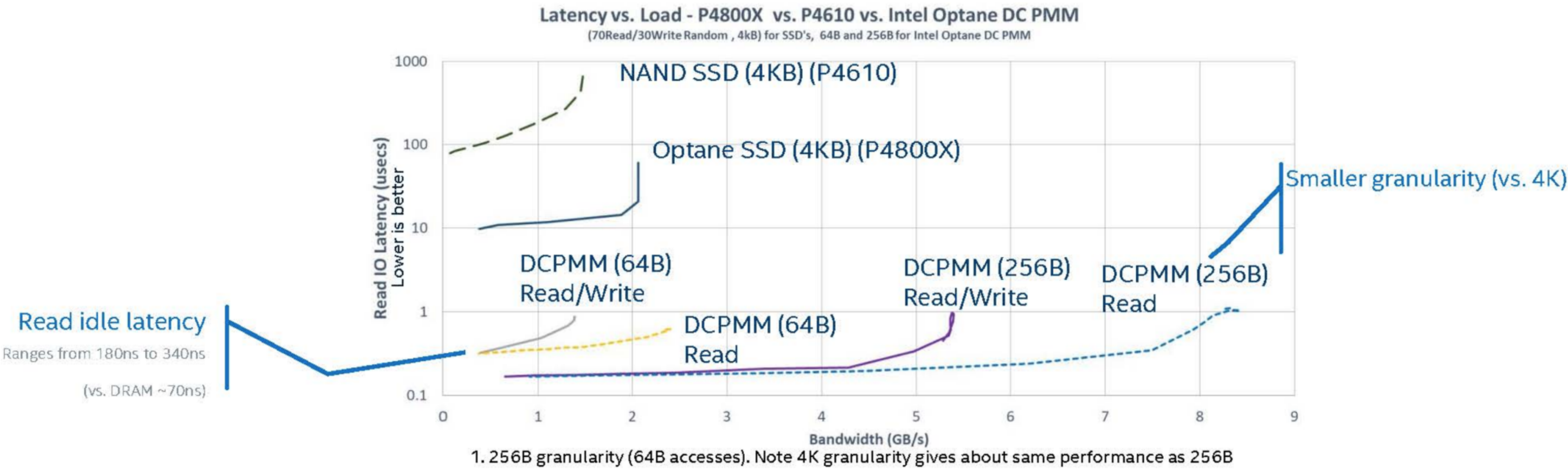
- **Good locality means near-DRAM performance**
  - Cache hit: latency same as DRAM
  - Cache miss: latency DRAM + Optane DC persistent memory
- **Performance varies by workload**
  - Best workloads have the following traits:
    - Good locality for high DRAM cache hit rate
    - Low memory bandwidth demand
  - Other factors:
    - # of reads > # of writes





Memory Mode Performance

- Order of magnitude lower latency than SSD
- 2x read/write bandwidth vs disk, with one module, more with multiple modules

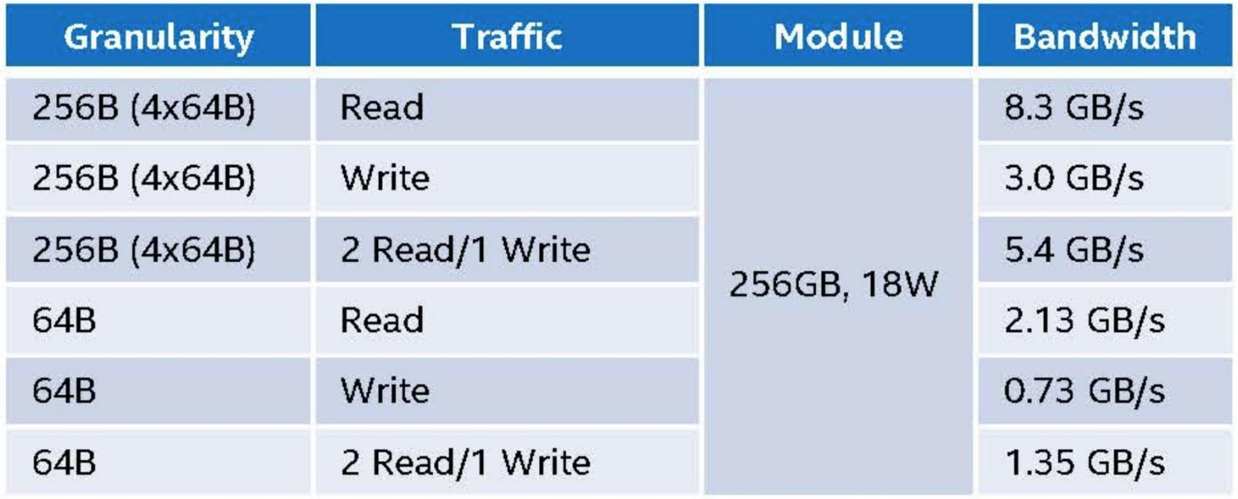


1. 256B granularity (64B accesses). Note 4K granularity gives about same performance as 256B

Memory Mode Performance (Cont.)

- **Optane DC PMM is programmable for different power limits for power/performance optimization**
  - 12W – 18W, in 0.25 watt granularity – for example: 12.25W, 14.75W, 18W
  - Higher power settings give best performance

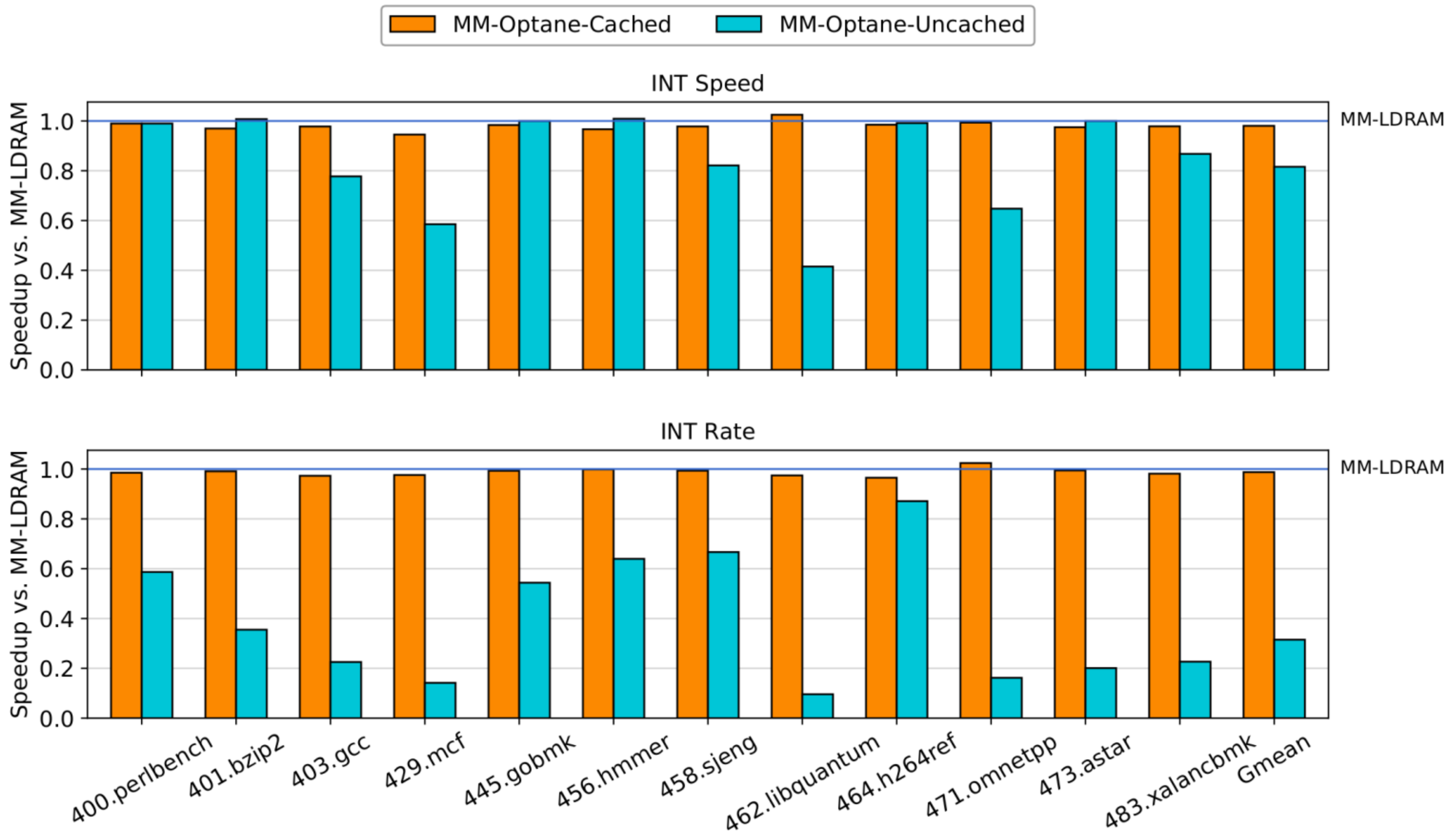
- **Performance varies based on traffic pattern**
  - Contiguous 4 cache lines (256B) vs. single random cache line (64B)
  - Reads vs. writes



Granularity	Traffic	Module	Bandwidth
256B (4x64B)	Read	256GB, 18W	8.3 GB/s
256B (4x64B)	Write		3.0 GB/s
256B (4x64B)	2 Read/1 Write		5.4 GB/s
64B	Read		2.13 GB/s
64B	Write		0.73 GB/s
64B	2 Read/1 Write		1.35 GB/s

Memory Mode Performance (Cont.)

- **Performance with SPEC CPU 2006 and 2017**



(a)

🟧 MM-Optane-Cached 🟦 MM-LDRAM ■ MM-Optane-Uncached

Outline

- What is Persistent Memory
- Memory Mode
- App Direct Mode

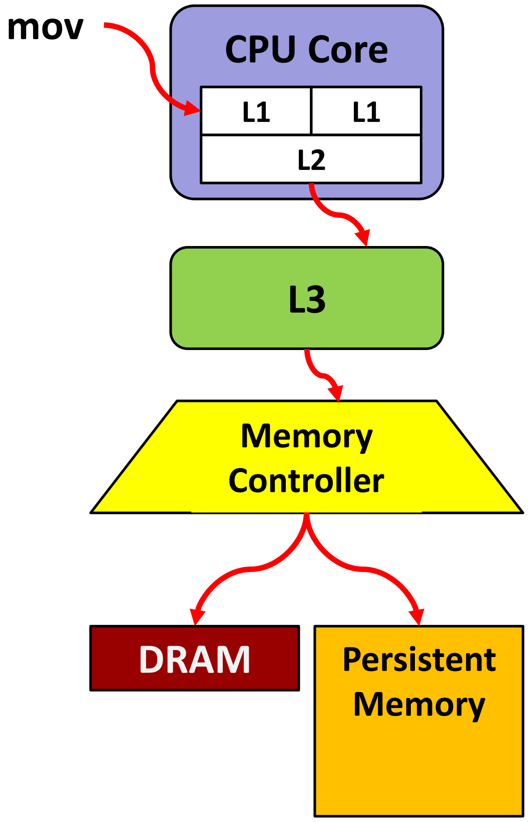
Two Modes of Optane DC PMM

- 1. ***Memory Mode***
  - DDR4 DIMMs operate as caches for slower Optane DC PMM
    - Direct mapped write-back cache with 64B cache lines
  - Optane DC PMMs are exposed as a large "*volatile*" memory
  - Do not have persistence

- 2. ***App Direct Mode***
  - Optane DC PMMs are directly exposed to the CPU and OS as "persistent" storage
  - Both Optane DC PMMs and DDR4 DIMMs are visible to OS as memory devices
  - Optane DC PMM is exposed as configurable regions on contiguously-addressed ranges

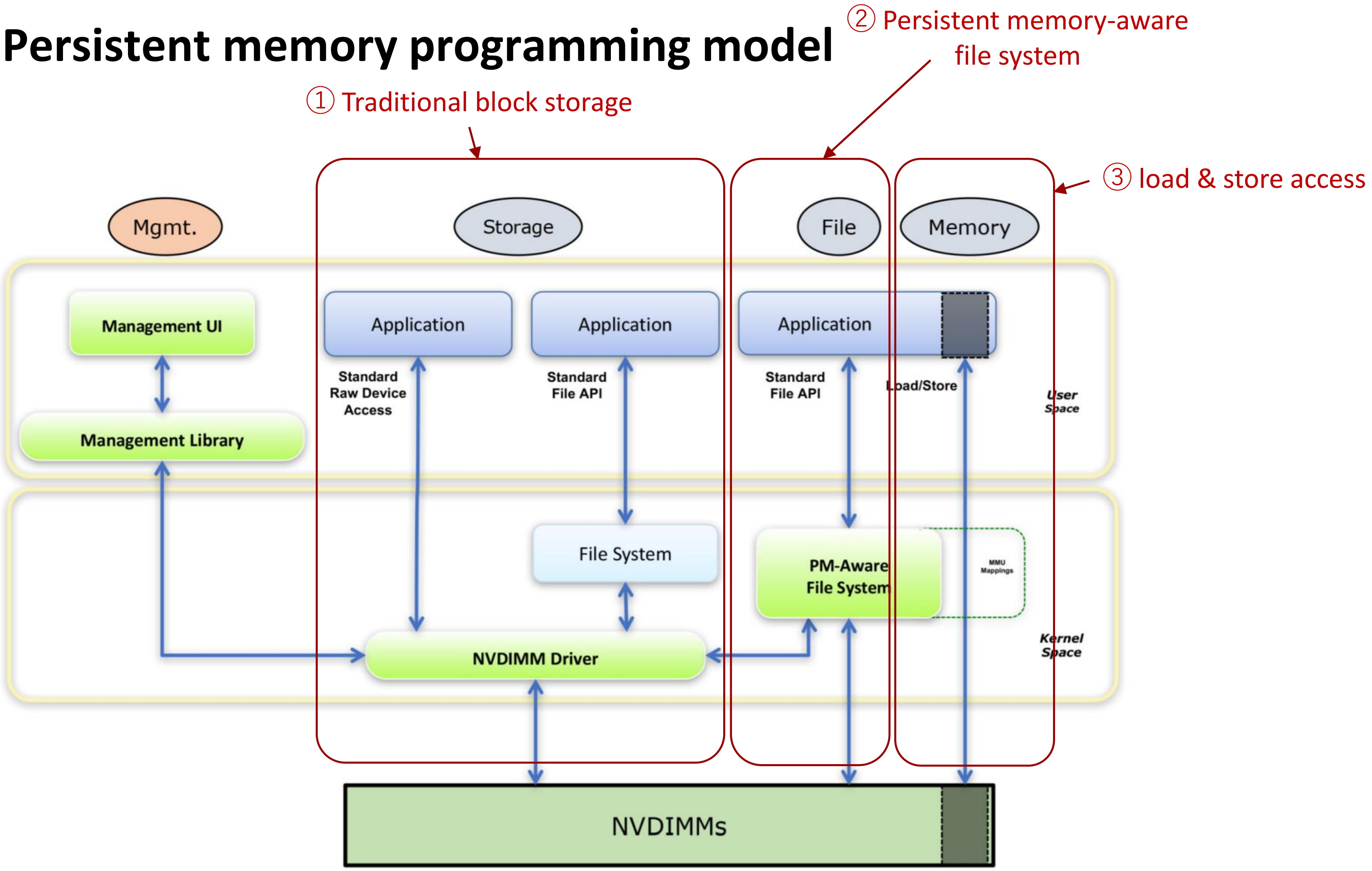
App Direct Mode Details

- **PMEM-aware software/application required**
  - Adds a new tier between DRAM and block storage
  - Industry open standard programming model (e.g., PMDK)
- **In-place persistence**
  - No paging, context switching, interrupts, nor kernel code executes
- **Byte addressable like memory**
  - Load/store access, no page caching
- **Cache coherent**
- **Ability to do DMA & RDMA**
  - No copy to DRAM for data transfer to block storage



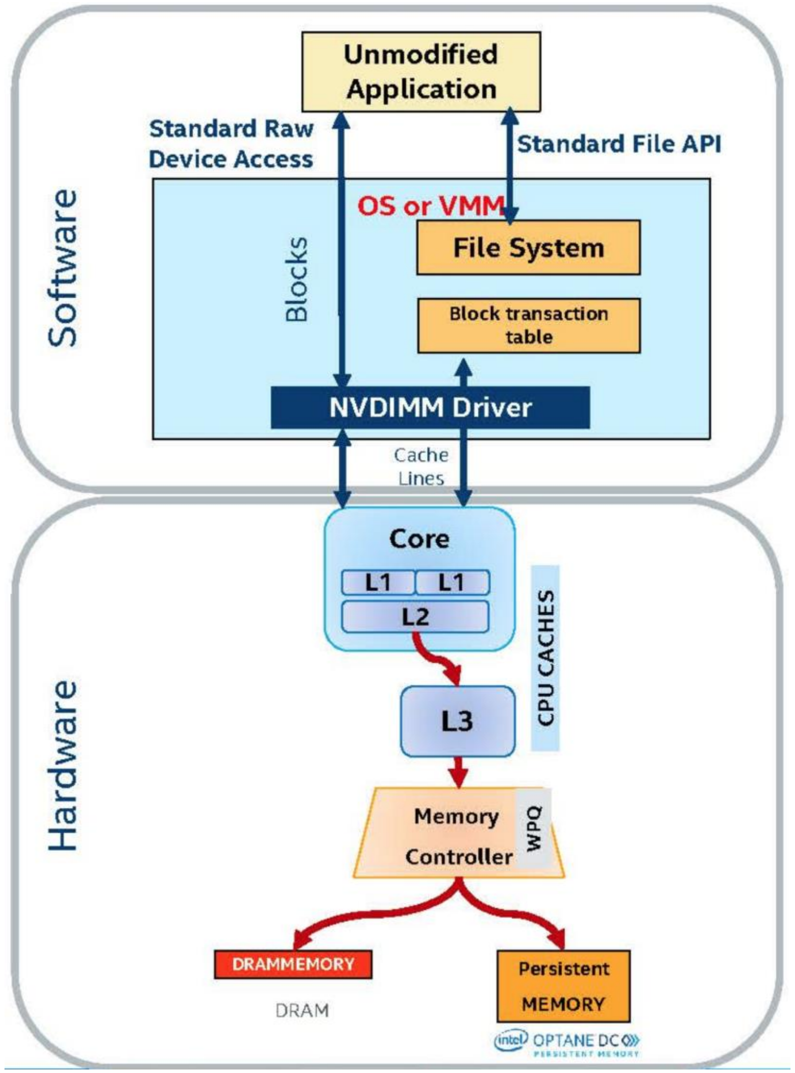
App Direct Mode Details (Cont.)

- **Persistent memory programming model**



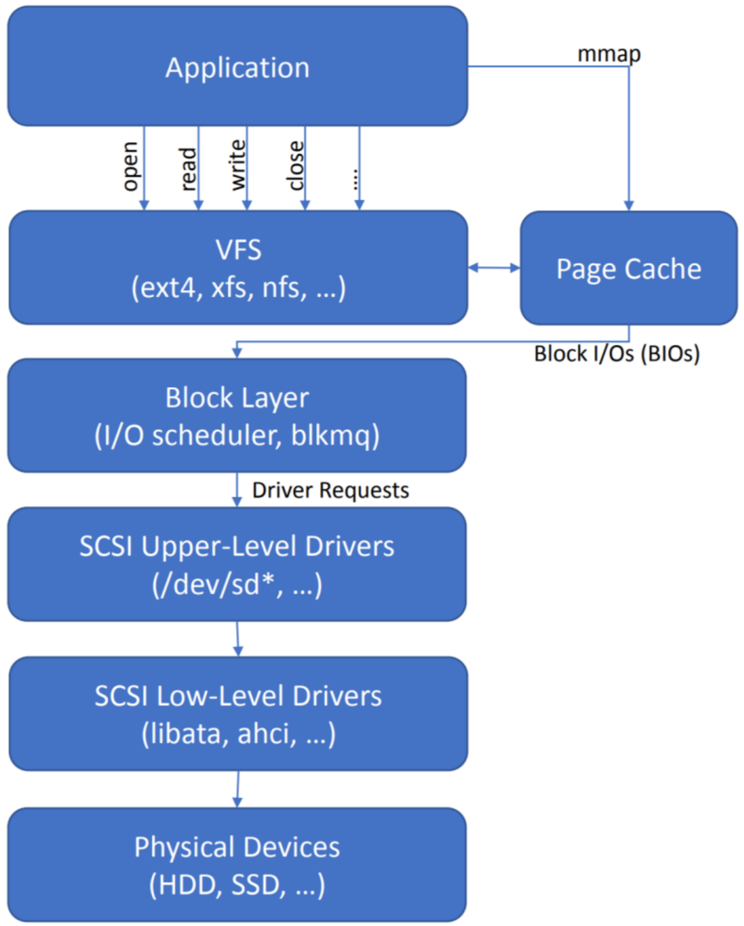
Traditional Block Storage

- **Operates in blocks like SSD/HDD**
  - Traditional read/write instructions
  - Work with existing file systems
- **NVDIMM driver required**
  - Support starting Linux kernel 4.2
- **Scalable capacity**
- **Higher endurance than SSDs**
- **High performance block storage**
  - Low latency, higher bandwidth, high IOPS

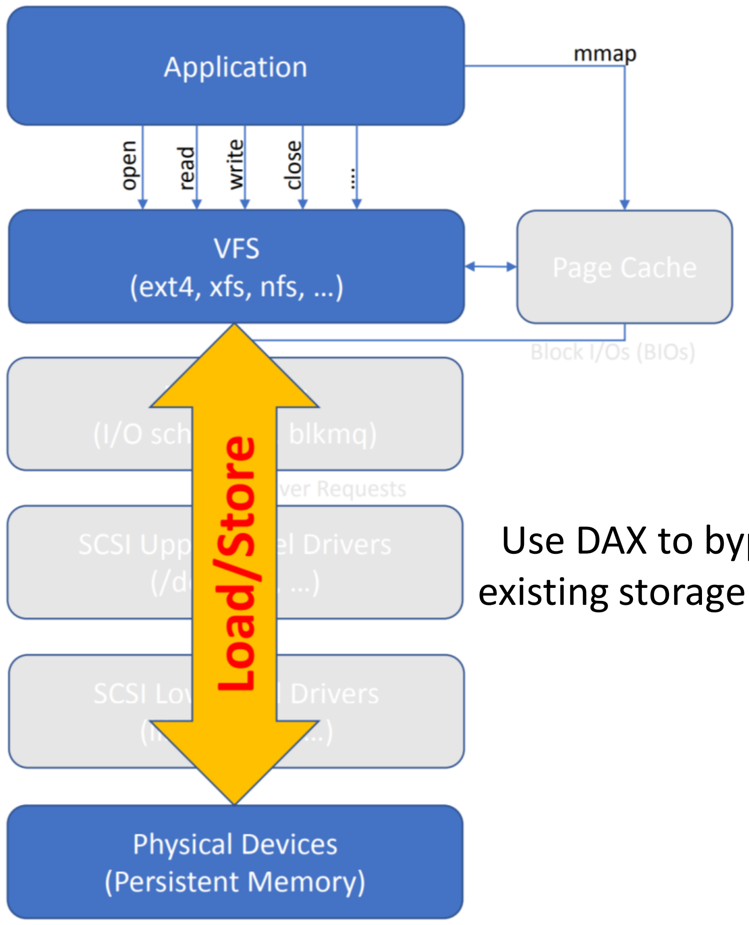


Persistent Memory-aware File System

- Traditional Storage Stack



- Persistent Memory-aware Stack

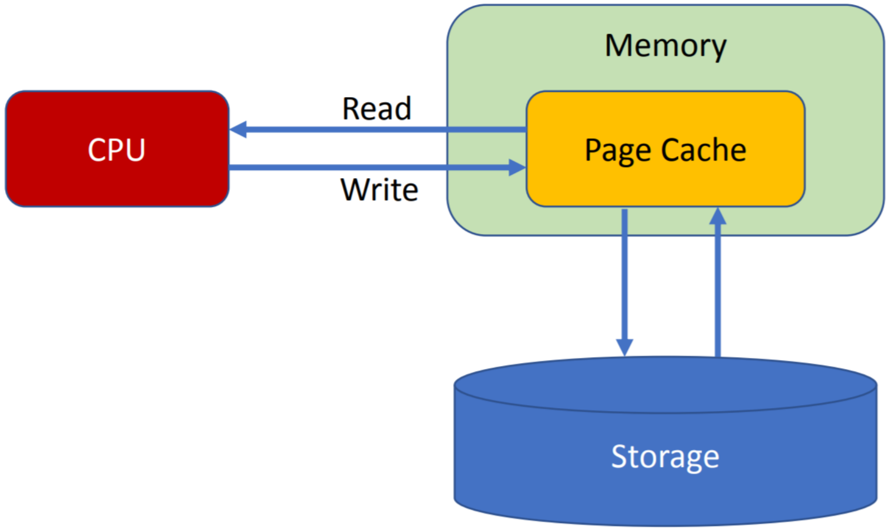


Use DAX to bypass existing storage stack

DAX-enabled File System

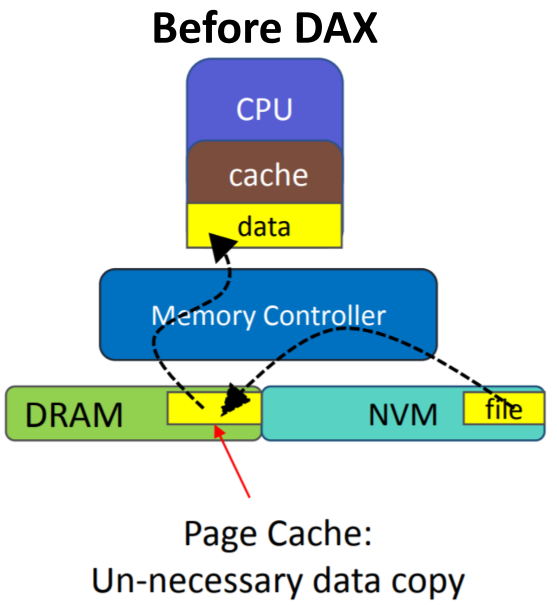
- **What is DAX (*Direct Access*)**
  - Allow file systems to directly access to persistent memory without using the system page cache
  - Added to Linux and Windows operating systems

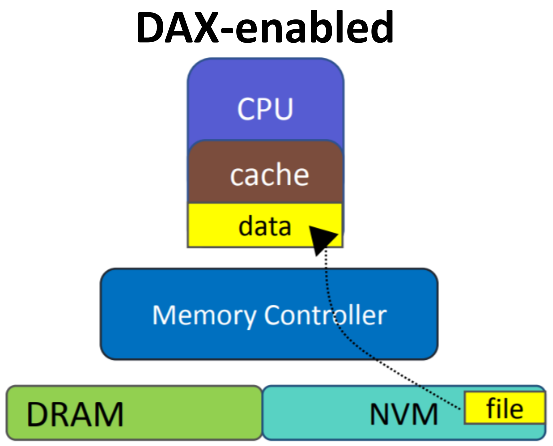
- **Why is DAX necessary?**
  - Page Cache
    - Mediate between fast memory and slow storage
  - Elegant design
    - Read-ahead
    - Write-back policy
    - …
  - Valid for decades until now,
    - Becoming a new bottleneck!



DAX-enabled File System (Cont.)

- **Use existing `mmap` semantics**
  - The persistent memory is directly mapped to the user space
- **In-place update**
  - All the data is directly written to the persistent memory without buffering in the page cache
- **True device performance**
- **File system implementation with DAX**
  - EXT4-DAX and NTFS-DAX





Is DAX all enough?

- **Simple answer: No**

- **More details?**
  - Crash consistency not guaranteed!

Crash Consistency

- **Definition**
  - “Ensure that the file system keeps the on-disk image in a reasonable state given that crashes can occur at arbitrary points in time.”

Crash!

```
strcpy(pmem, “Hello World!”);
```

✗ Hello

○ Hello World!

○ *Nothing*

How about using `fsync()` or `msync()`

- **Some might want to `fsync()` or `msync()` to ensure the changes are persistence**

```
strcpy(pmem, "Hello World!");
msync(pmem, 12, MS_SYNC);
```

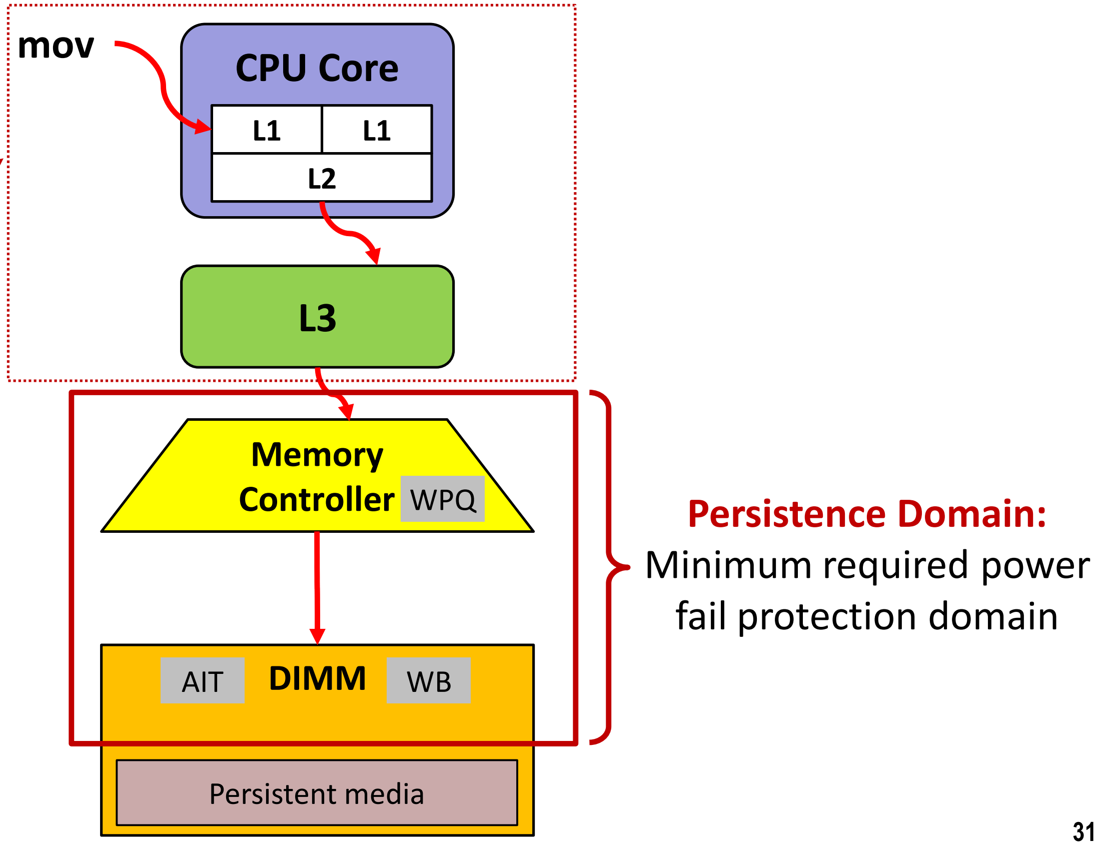
- **Do they work with the persistent memory?**
  - Answer: No
  - Why: `fsync()` and `msync()` are used to flush out dirty pages in the page cache to storage devices. But, with DAX, no page cache is used

Persistence Domain

- **Asynchronous DRAM refresh (ADR) (Intel's feature)**
  - Ensure the data is in a "safe" state during a power loss event or system crash
  - Triggers a hardware interrupt to the memory controller
  - The controller will *flush* the data buffers and place the DRAM in *self-refresh*
  - *It does not flush the CPU cache*

X86 caches are quite large;
Take more energy than the capacitors in a power supply can provide

Software makes sure that data is flushed to persistent domain using `CLFLUSH` or `CLWB`



Persistence Domain:
Minimum required power fail protection domain

X86 Cache Flush Instructions

- **Simply executing a store instruction is not enough to make data persistent**
  - The data may be sitting in the CPU caches and could be lost by a power failure
- **Additional cache flush instructions (e.g., CLFLUSHOPT and CLWB) are required to make the stores persistent**

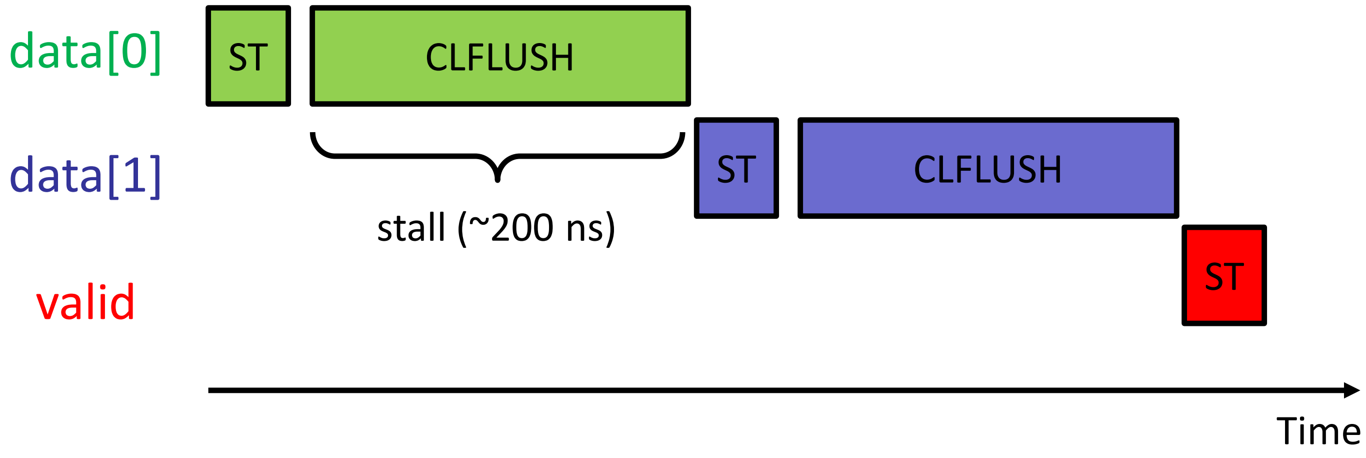
CLFLUSH	This instruction, supported in many generations of CPU, flushes a single cache line. Historically, this instruction is serialized, causing multiple CLFLUSH instructions to execute one after the other, without any concurrency.
CLFLUSHOPT	This instruction, newly introduced for persistent memory support, is like CLFLUSH but without the serialization
CLWB	Another newly introduced instruction, CLWB stands for cache line write back. The effect is the same as CLFLUSHOPT except that the cache line may remain valid in the cache, but no longer dirty
SFENCE	Ensure the flushes are complete before continuing

CFLUSH

- **Order writes by flushing cachelines via CLFLUSH**

```
STORE data[0] = 0xFOOD
STORE data[1] = 0xBEEF
CLFLUSH data[0]
CLFLUSH data[1]
STORE valid = 1  /* mark that data[0] and data[1] are valid */
```

- **CLFLUSH stalls the CPU pipeline and serializes execution**



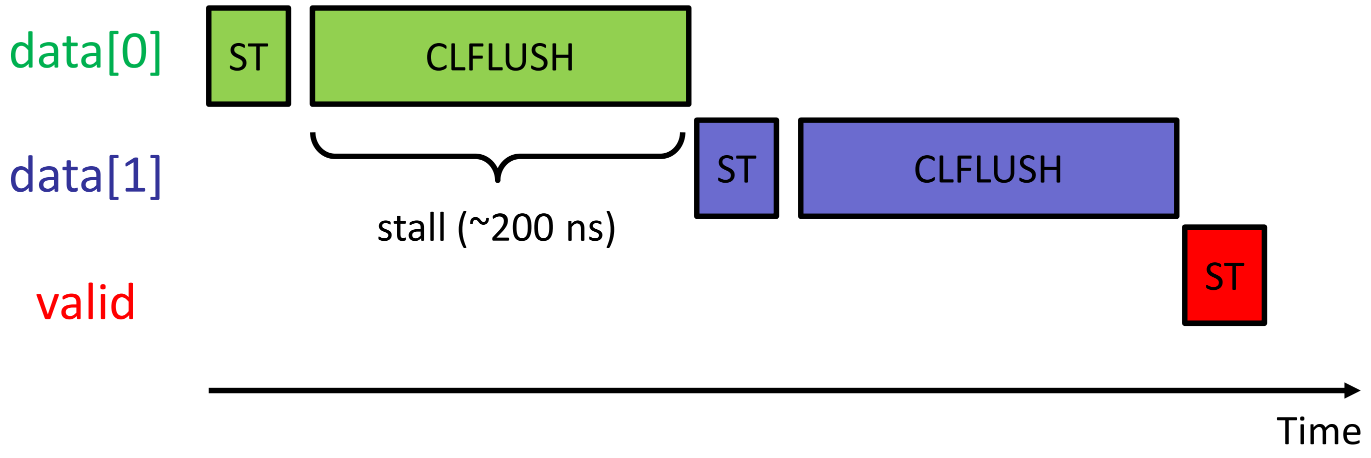
CLFLUSHOPT

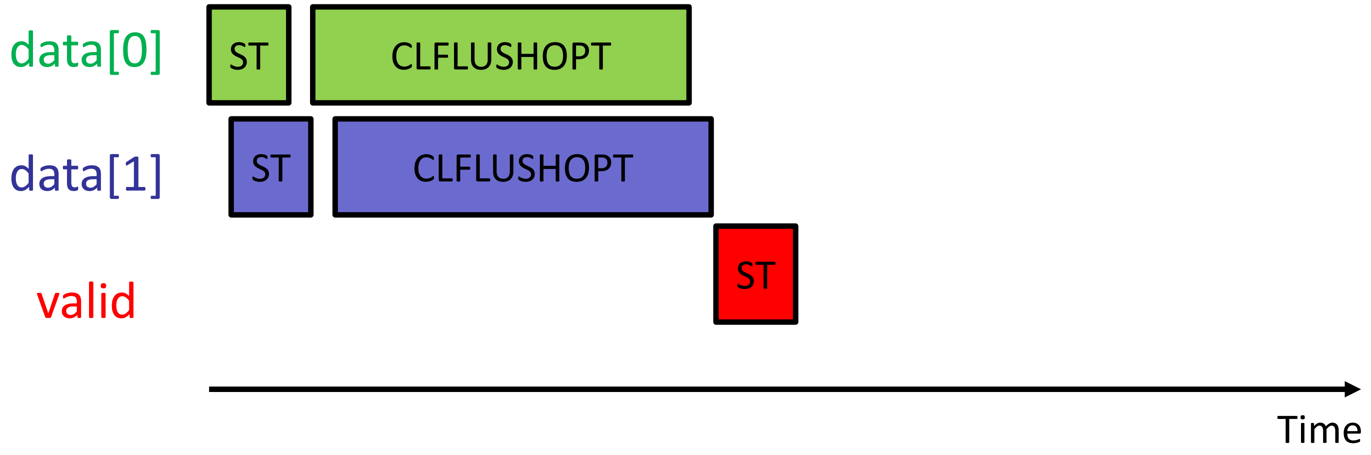
- Provides unordered version of CLFLUSH
- Supports efficient cache flushing

```
STORE data[0] = 0xFOOD
STORE data[1] = 0xBEEF            ← Implicit orderings
CLFLUSHOPT data[0]
CLFLUSHOPT data[1]
SFENCE                            ← Explicit ordering point
STORE valid = 1
```

CLFLUSHOPT (Cont.)

- Provides unordered version of CLFLUSH
- Supports efficient cache flushing



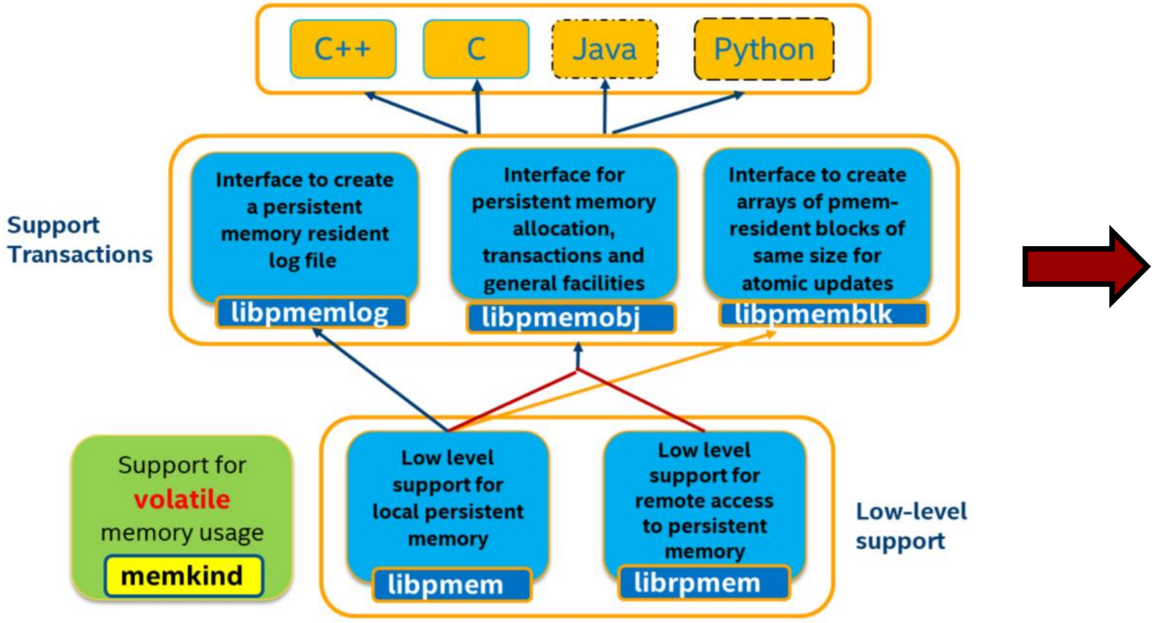


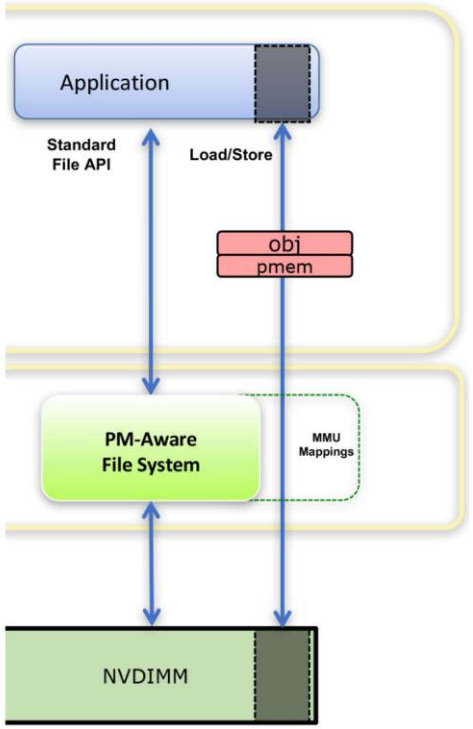
CLWB

- **Write backs modified data of a cacheline**
- **Does not invalidate the line from the cache**
  - Marks the line as non-modified
  - Improve cache hit ratios

The NVM Libraries

- A suite of six libraries for convenient use of persistent memory in user-space applications


C++
C
Java
Python
Support Transactions
Interface to create a persistent memory resident log file
libpmemlog
Interface for persistent memory allocation, transactions and general facilities
libpmemobj
Interface to create arrays of pmem-resident blocks of same size for atomic updates
libpmemblk
Support for volatile memory usage
memkind
Low level support for local persistent memory
libpmem
Low level support for remote access to persistent memory
librpmem
Low-level support



`libpmem`: Basic Persistence Support

- **`libpmem` provides low level persistent memory supports**
  - Open and allocate the persistent memory
  - Memory map the persistent memory to the user space
  - Detect which types of flush instructions are supported by the CPU
  - Performance-tuned routines for copying ranges of persistence memory
  - …

```c
	/* create a pmem file */
	if ((fd = open("/pmem-fs/myfile", O_CREAT|O_RDWR, 0666)) < 0) {
		perror("open");
		exit(1);
	}

	/* allocate the pmem */
	if ((errno = posix_fallocate(fd, 0, PMEM_LEN)) != 0) {
		perror("posix_fallocate");
		exit(1);
	}

	/* memory map it */
	if ((pmemaddr = pmem_map(fd)) == NULL) {
		perror("pmem_map");
		exit(1);
	}
	close(fd);
```

```c
	/* flush above strcpy to persistence */
	if (is_pmem)
		pmem_persist(pmemaddr, PMEM_LEN);
	else
		pmem_msync(pmemaddr, PMEM_LEN);
```

`libpmemobj`: General-Purpose Allocations and Transactions

- `libpmemobj` allows persistent memory objects to be allocated in a way that is power fail safe
- Allows making an arbitrary number of changes atomic by encompassing the changes in a transaction
- Multithread safe and optimized for multithread scalability

```
class pmem_queue {

        /* entry in the list */
        struct pmem_entry {
                persistent_ptr<pmem_entry> next;
                p<uint64_t> value;
        };
        /* … */
```

```
void
push(pool_base &pop, uint64_t value)
{
        transaction::exec_tx(pop, [&] {
                auto n = make_persistent<pmem_entry>();

                n->value = value;
                n->next = nullptr;

                if (head == nullptr) {
                        head = tail = n;
                } else {
                        tail->next = n;
                        tail = n;
                }
        });
}
```

`libpmemblk` and `libpmemlog`: Support for Specific Use Cases

- `libpmemblk`
  - Maintain a large array of persistent memory blocks, all the same size
  - This is useful when an application is managing a block cache
    - The block size provided by the library is flexible, supporting blocks 512-bytes and larger
- `libpmemlog`
  - For a specific use case where the application frequently appends to a private log file, one that is read rarely, like during crash recovery

`libmemkind`: The Volatile Use of Persistent Memory

- **Volatile use of persistent memory**
  - Places some data structures in persistent memory to avoid a large DRAM footprint, but doesn't really care that the memory is persistent

End of Chapter 12