Tair: A Challenge of Performance in Intel® Optane™ DC Persistent Memory

路青霖

炸酱面组

Team with 李凯 胡伟东

Tair[1]: Challenge of Performance

- High TPS in hotspot items (tens of millions)
  - Hotspot hashing mechanism
  - Limited in **Single node**
- Intel® Optane™ DC Persistent Memory Module
  - Performance[2]

Performance		**PMEM**	**DRAM**
Latency	Load	305ns/**169ns(Seq)**	81ns
	Store	**94ns**	86ns
Bandwidth	Load	6.6GB/s	
(MT, AVX-512)	Store	2.3GB/s	

阿里云 | intel

Tair: Challenge of Performance

- **Key-Value Engine**
  - Single node with **8G** DRAM / **64G AEP** (Optane DC PMEM)

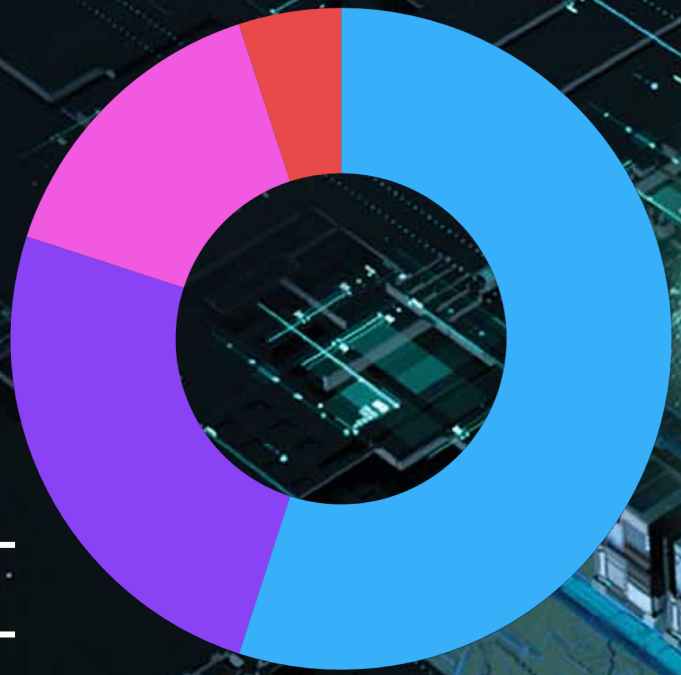
Intel(R) Xeon(R) Platinum 8269CY	**26** Cores (**54** Threads) SSE4.2、AVX2、AVX-512
DRAM	**8GB**
Intel(R) Optane(R) DC PMEM	**64GB**

  - Persistence
  - Recovery

阿里云 | intel

Tair: Challenge of Performance

- Judgment
- Data
  - Key (**16B**)
  - Value (**80~1024B**, Max **50GB**)

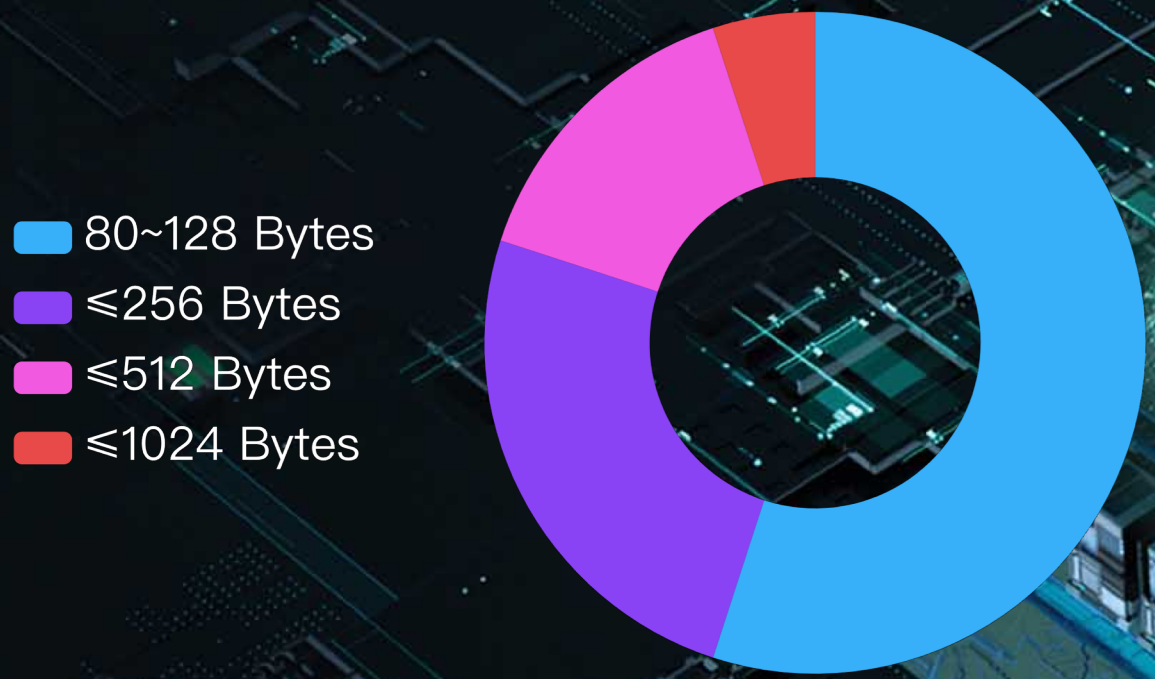


Correctness	Concurrent Get & Set (**16** Threads)
Persistence	Crash-Recover
Performance	**24M** Set (**16** Threads)
	24M Set (**25%**) + Get (**75%**, hotspot **75%**)

阿里云 | intel

Tair: Challenge of Performance

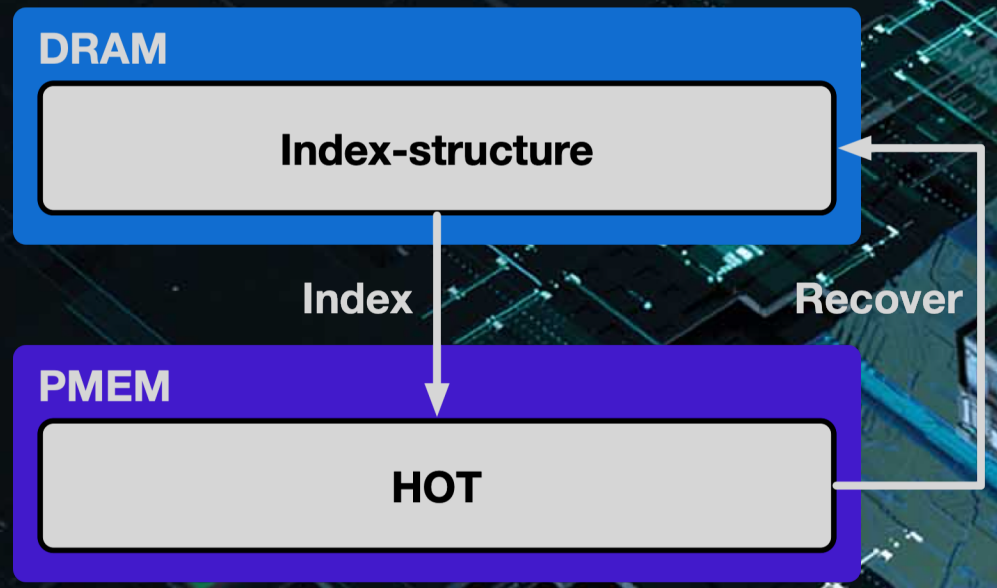
- **Index** under multi-threads
  - Concurrent **consistency**
  - Upper limit
- **Variable** value length
  - **80~1024** Bytes
  - Persistence
  - Collect & reuse (**25G**)
- Recovery
  - Flush and update
- Unleash potential performance of DC PMEM



Overview Design

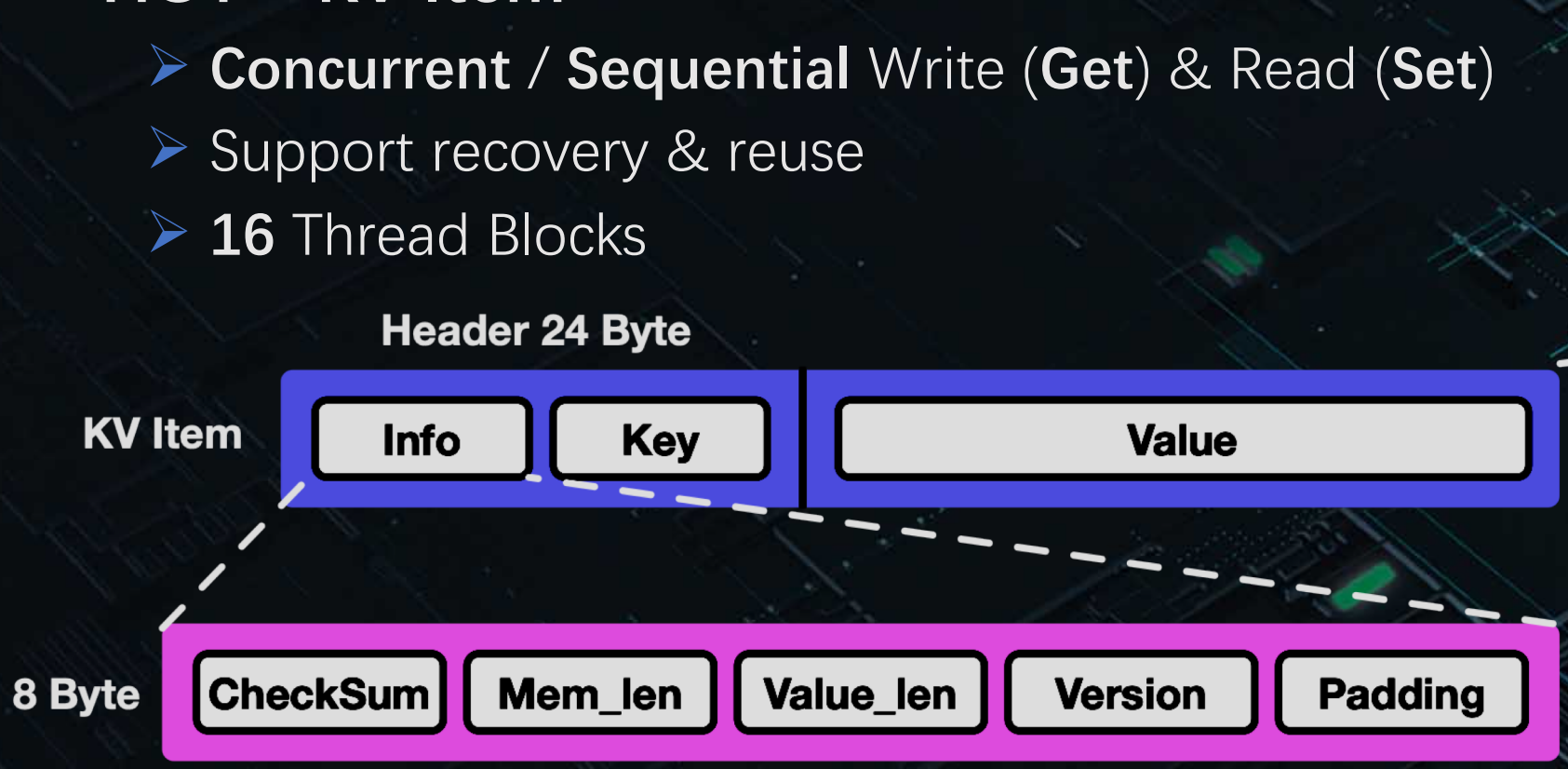
- What does KV-Engine do?
  - $f: Key \rightarrow Value$
- Heap-organized Table (HOT)[3]
  - **Unsorted**
- In-memory Index-structure
  - **Speedup** searching in HOT

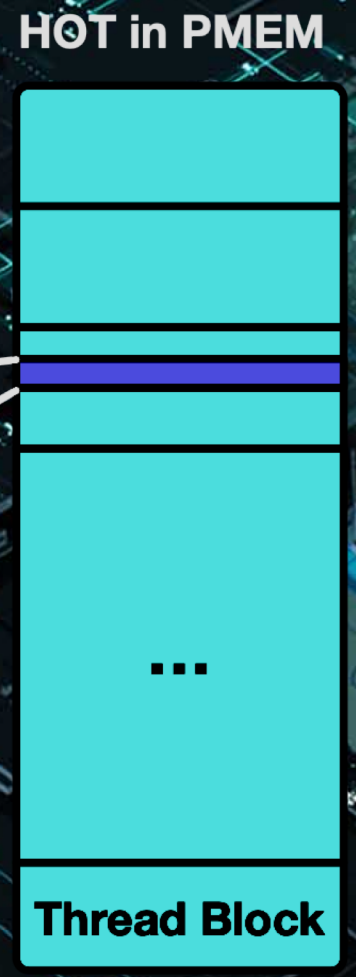
$$\begin{cases} g: Key \rightarrow Index \\ h: Index \rightarrow Value \end{cases} \Longrightarrow f(Key) = h(g(Key))$$



Solution

- **HOT – KV Item**
  - **Concurrent / Sequential** Write (**Get**) & Read (**Set**)
  - Support recovery & reuse
  - **16** Thread Blocks

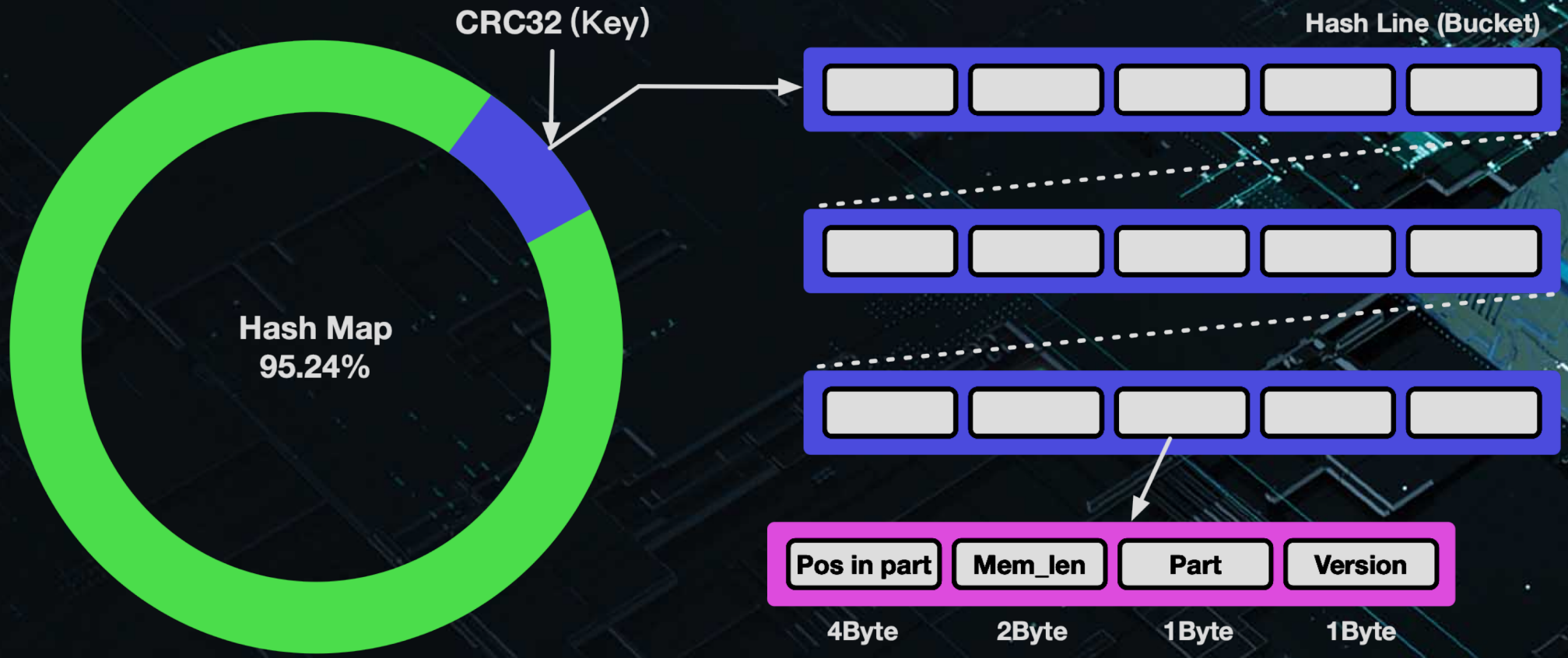




Solution

- Index-structure - Hash map
  - $g: Key \rightarrow Index$

Redundancy 5%, $\alpha = \frac{N_{KV}}{N * N_{line}} = \frac{229037390}{48097859 * 5} = 95.24\%$



Solution

- **Hash Map - Hash Line**
  - Hash Line X aligned by Cache Line (**64Byte**) C:

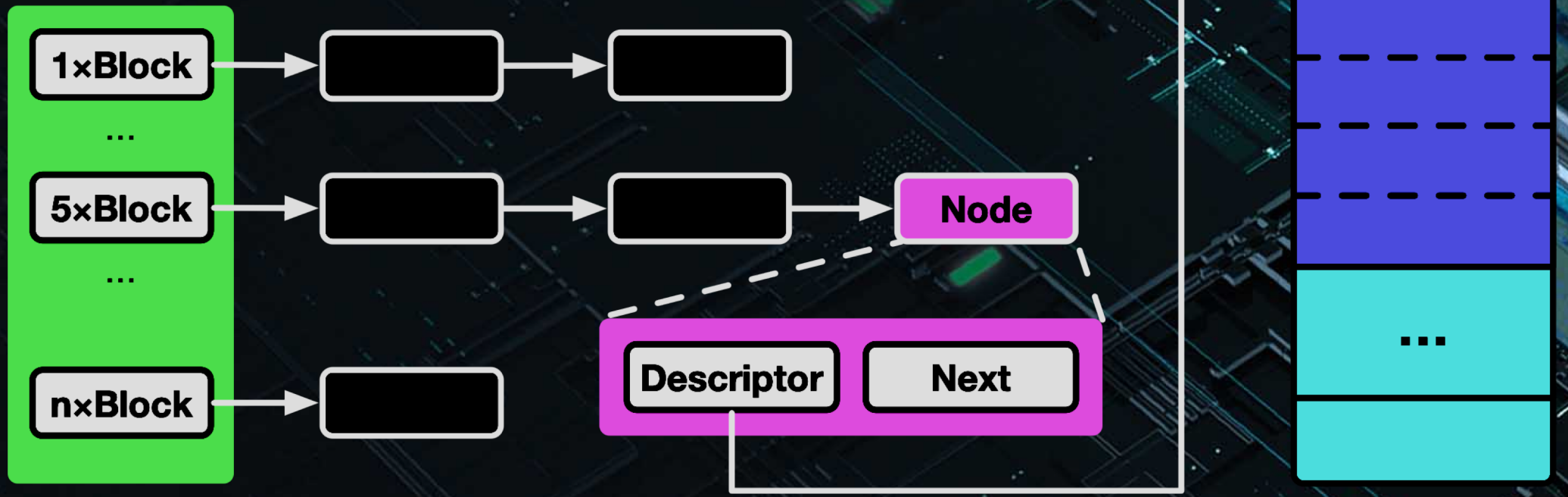
$$
\begin{aligned}
minimize \quad & 64C - (16+8)X - 1 \\
subject\ to \quad & 64C - (16+8)X \geq 1 \\
& 255 \geq X \geq 0 \\
& X \text{ is an integer} \\
& C \geq 0 \\
& C \text{ is an integer} \\
=> \ & X = 5, C = 2
\end{aligned}
$$

```
/* 128 Byte total */
HashLine {
  /* 80 Byte */
  Key     keys[5];
  /* 40 Byte */
  uint64 vals[5];
  /* 8 Byte */
  uint8   count;
  uint8   padding[7];
```

  - **Lock-free** Insert

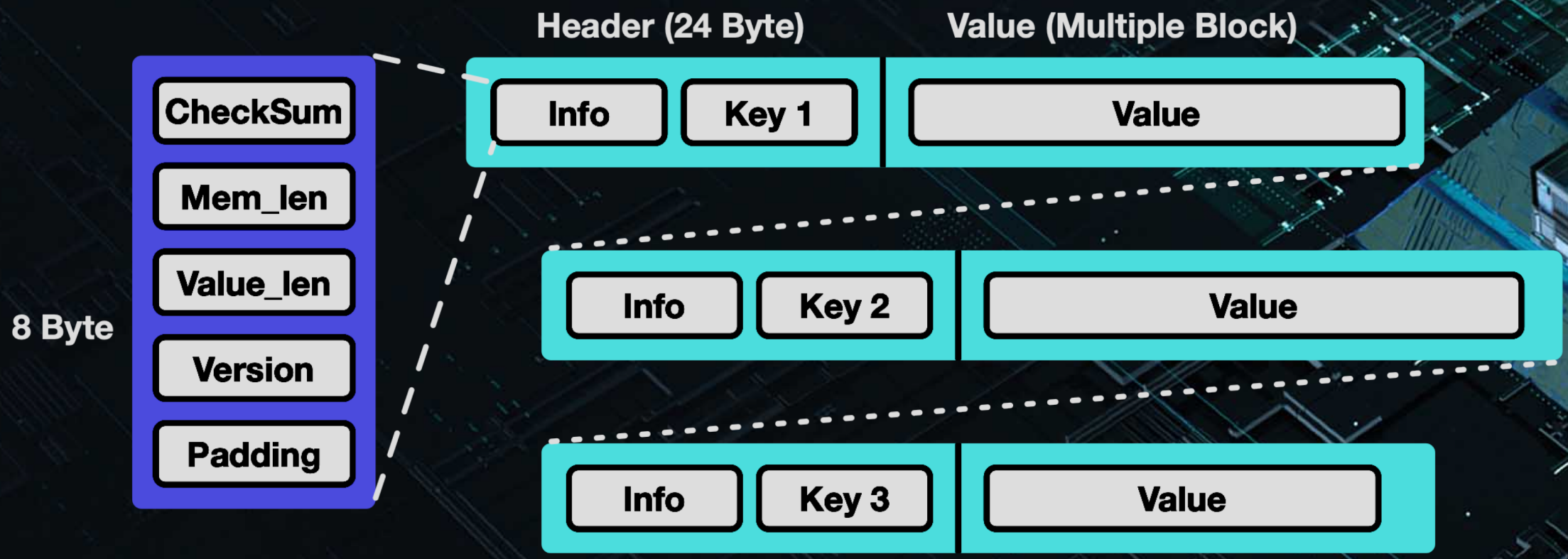
Solution

- **Allocator (Thread independent)**
  - Aligned (**32 Byte**) memory block
  - **Free-list (**Page Arena)



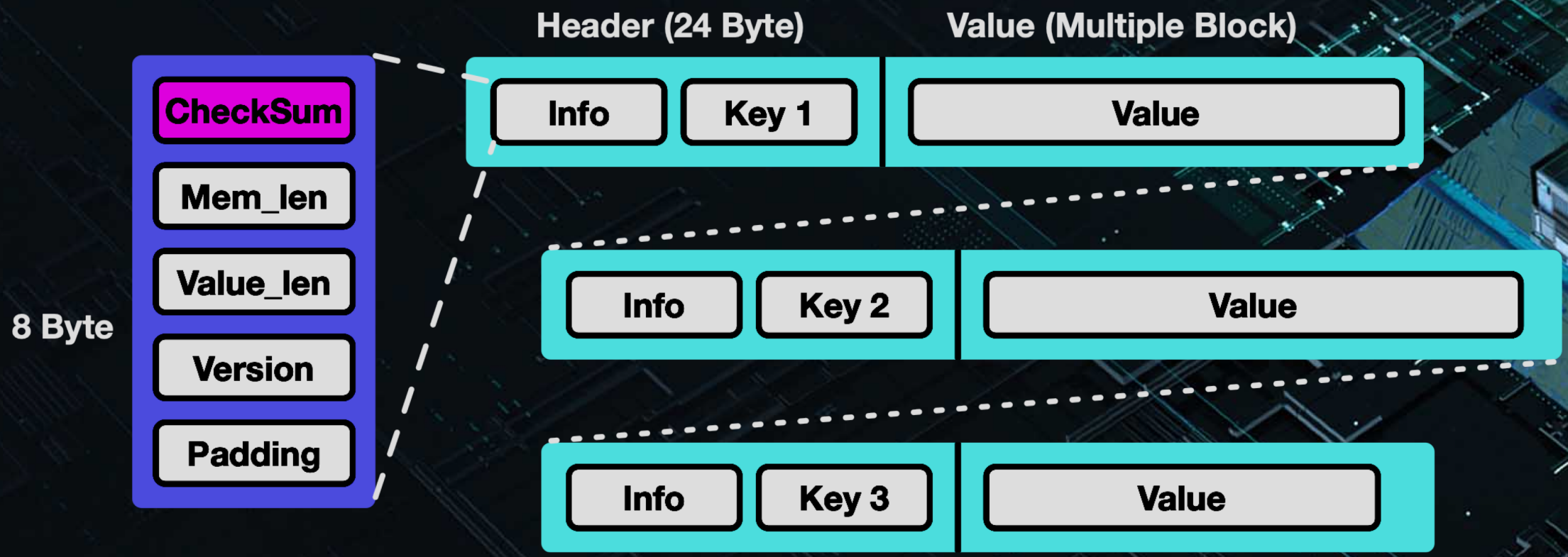
Solution

- **Recovery**
  - Consistence
  - **Atomic** Write Unit (**8 Byte**)[3]



Solution

- **Recovery**
  - Consistence
  - **Atomic** Write Unit (**8 Byte**)[3]



Optimization

- False sharing
  - ThreadLocal - allocator offset & size
- Warm up
  - Page fault
  - pmem_memset
  - <160s
- Get thread ID
  - Individual
  - Sequential

Score: 55

Optimization

- **False sharing**
  - ThreadLocal - allocator offset & size
- Warm up
  - Page fault
  - pmem_memset
  - <160s
- Get thread ID
  - Individual
  - Sequential

Score: 55

Optimization

- False sharing
  - ThreadLocal - allocator offset & size
- **Warm up**
  - Page fault
  - pmem_memset
  - <160s
- Get thread ID
  - Individual
  - Sequential

Score: 44

Optimization

- False sharing
  - ThreadLocal - allocator offset & size
- Warm up
  - Page fault
  - pmem_memset
  - <160s
- **Get thread ID**
  - Individual
  - Sequential

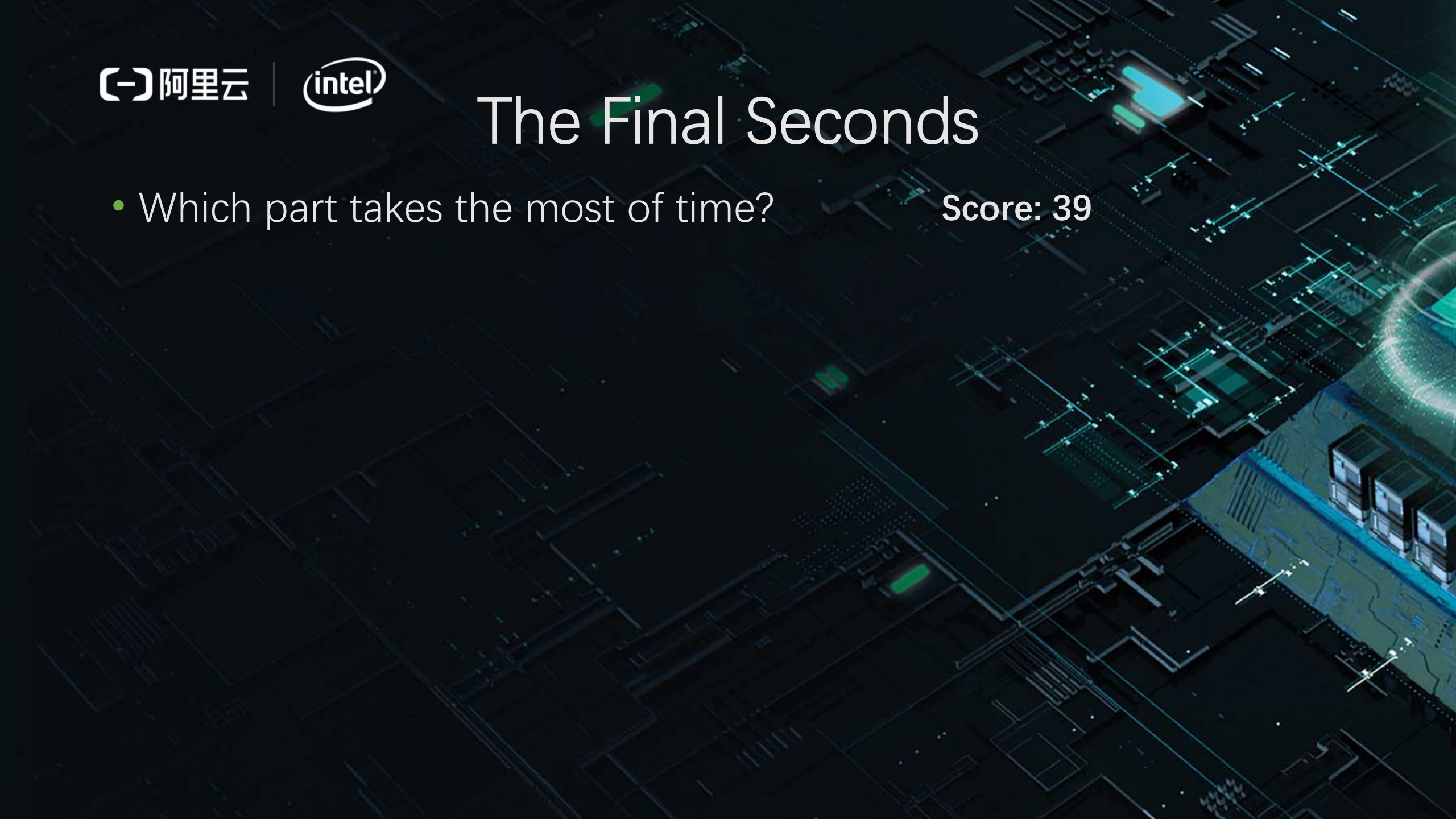
Score: 39

```
int64_t gettno() {
  static __thread int64_t tno = -1;
  static int64_t tno_seq = 0;
  if (UNLIKELY(-1 == tno)) {
    tno = __sync_fetch_and_add(&tno_seq, 1);
  }
  return tno;
}
```

The Final Seconds

- Which part takes the most of time?

Score: 39



The Final Seconds

- Which part takes the most of time?

Score: 39

- **Hash Map**
  - Get
  - Occupy
  - Replace
- Heap Table
- Allocator

```
hash = CRC32(key);
for (i = 0; i < HASH_LINE_CNT; ++i) {

  /* Locate hash line */
  index = (hash + i) % HASH_LINE_CNT;

  /* find key in current hash line */
  hash_line[index].find(key);
  ...
}
```

Mod operation

Keys Comparing

The Final Seconds

- Which part takes the most of time?

Score: 38

  - **Hash Map**
    - Get
    - Occupy
    - Replace
  - Heap Table
  - Allocator

```
index = CRC32(key) % HASH_LINE_CNT;
for (i = 0; i < HASH_LINE_CNT; ++i) {

  /* find key in current hash line */
  hash_line[index].find(key);
  ...

  /* next hash line */
  index ++;
  index = UNLIKELY(index == HASH_LINE_CNT) ? 0 : index;

  /* prefetch next keys to L0 cache */
  _mm_prefetch(hash_line[index].keys, _MM_HINT_T0);
}
```

The Final Seconds

- Which part takes the most of time?

Score: 38

- Hash Map
- **Heap Table**
  - Get
  - Put
- Allocator

- **MEMCPY**
- **CRC** of KV Item

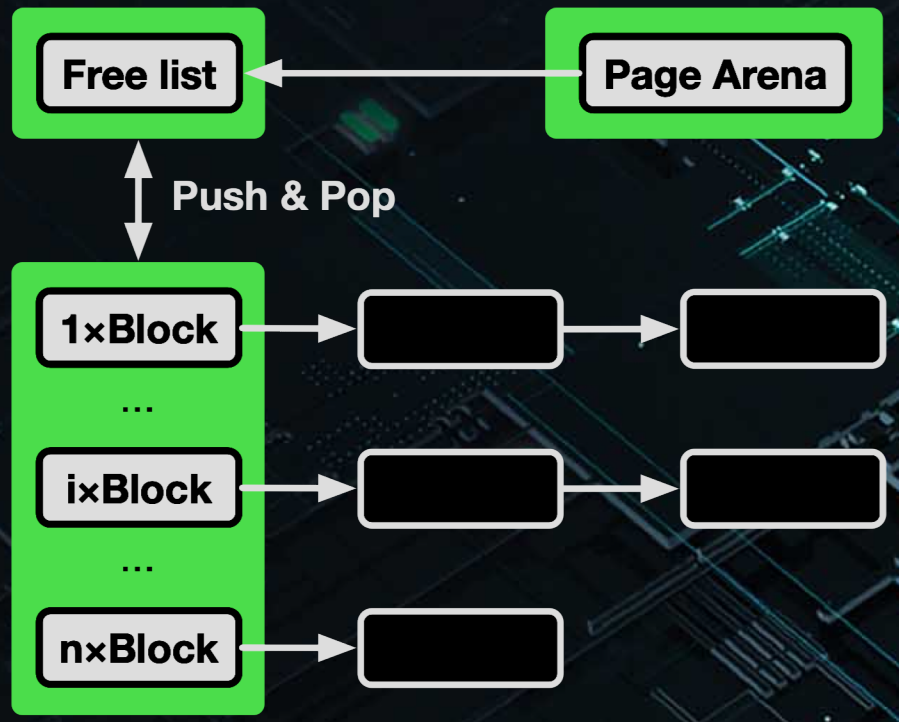
- **rte_memcpy**
- **XXHASH3[4] (AVX2)**

Hash Name	Width	Large Data Bandwidth	Small Data Velocity
XXH3	**64**	**59.4 GB/s**	**133.1**
XXH128	128	57.9 GB/s	118.1
CRC32C (hw)	32	13.0 GB/s	57.9

[4] Collet, Yann. "xxHash: Extremely Fast Hash Algorithm." (2016).

The Final Seconds

- Which part takes the most of time? **Score: 36**
  - Hash Map
  - Heap Table
  - **Allocator**

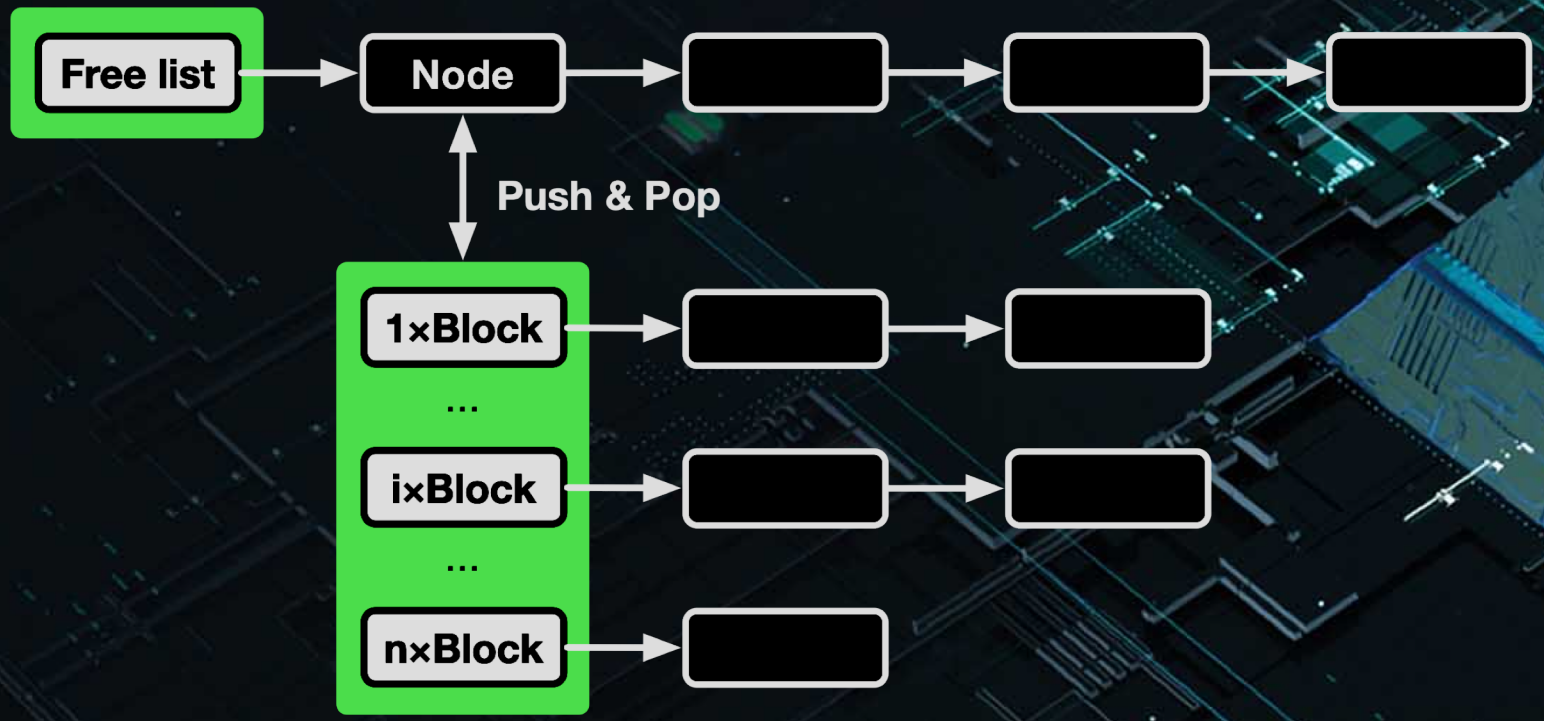


The Final Seconds

- Which part takes the most of time?

Score: 35.5

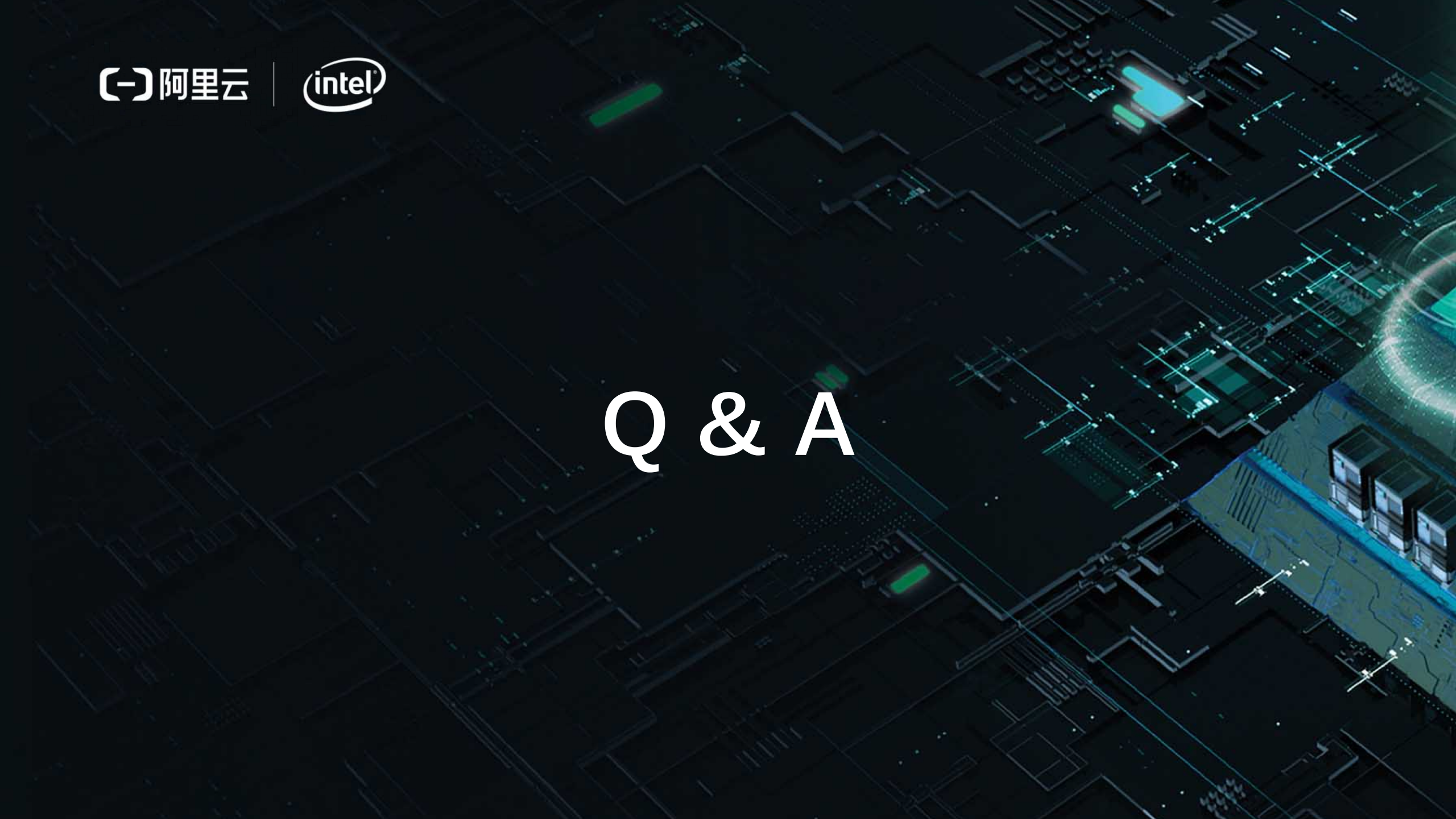
  - Hash Map
  - Heap Table
  - **Allocator**



Conclusion

- Hash Map (DRAM) + HOT (DC PMEM)
  - Thread blocks
  - Allocator
- Log & Profiling
- Value Cache
  - **Simplify** or not?
- 16 Threads => Thread pool
- **NOT** relay on the distribution of data

Score: 35.5 (25.5 + 10)


阿里云
intel
Q & A

Reference

[1] Duo long, Dao an, et al. 2017. TAIR, A distributed key-value storage system developed by Alibaba Group. https://github.com/alibaba/tair.

[2] Izraelevitz, Joseph, et al. "Basic performance measurements of the intel optane DC persistent memory module." *arXiv preprint* arXiv:1903.05714 (2019).

[3] Islam, A. A. R., Narayanan, A., York, C., & Dai, D. A Performance Study of Optane Persistent Memory: From Indexing Data Structures' Perspective.

[4] Collet, Yann. "xxHash: Extremely Fast Hash Algorithm." (2016).

[5] Chen, Jiqiang, et al. "HotRing: A Hotspot-Aware In-Memory Key-Value Store." *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20).* 2020.

Reference

[1] Duo long, Dao an, et al. 2017. TAIR, A distributed key-value storage system developed by Alibaba Group. https://github.com/alibaba/tair.

[2] Izraelevitz, Joseph, et al. "Basic performance measurements of the intel optane DC persistent memory module." *arXiv preprint* arXiv:1903.05714 (2019).

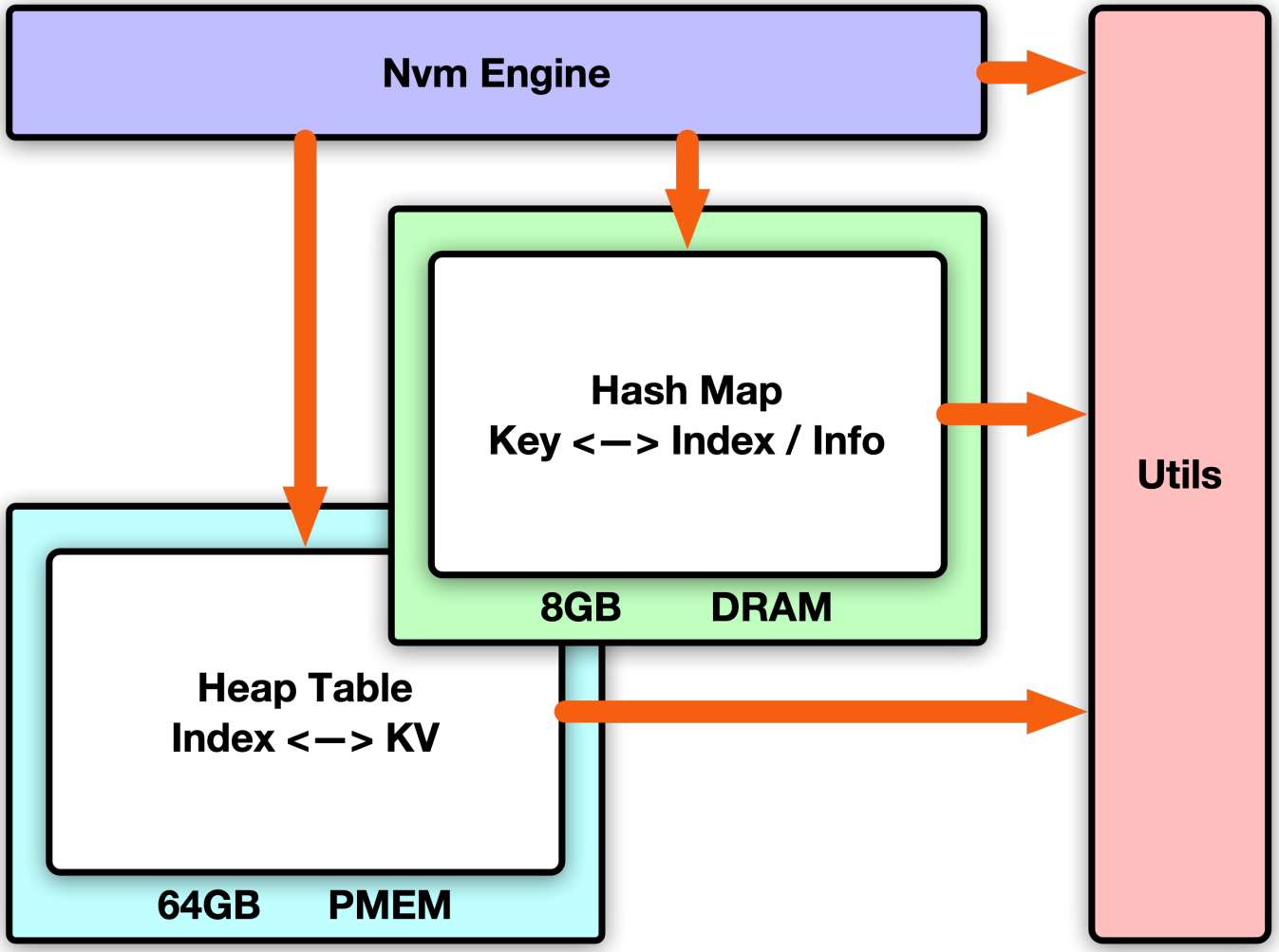
[3] Islam, A. A. R., Narayanan, A., York, C., & Dai, D. A Performance Study of Optane Persistent Memory: From Indexing Data Structures' Perspective.

[4] Collet, Yann. "xxHash: Extremely Fast Hash Algorithm." (2016).

[5] Chen, Jiqiang, et al. "HotRing: A Hotspot-Aware In-Memory Key-Value Store." *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20).* 2020.

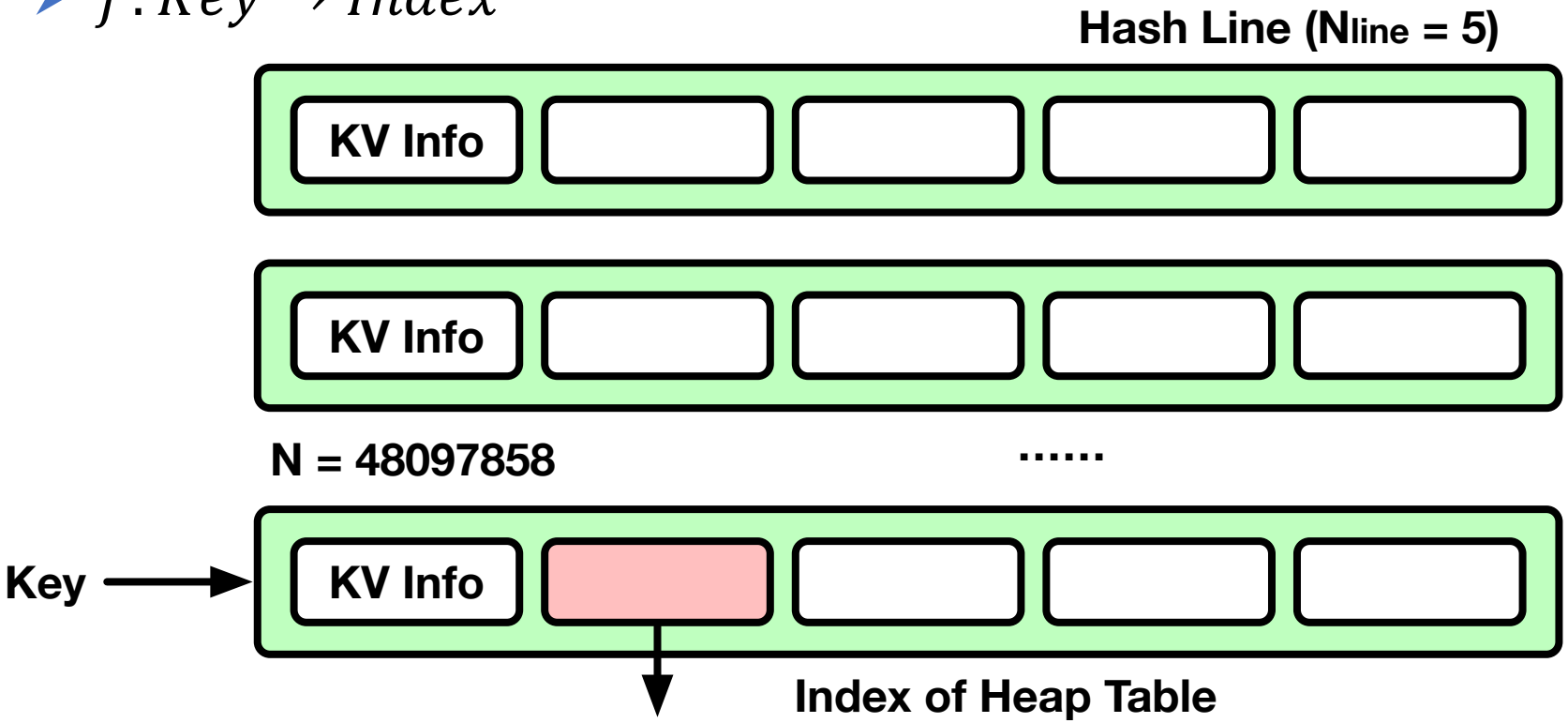
Solution

- **NvmEngine**
  - **Hash Map (5.73GB)**
    - Key to Index
  - **Heap Table (64GB)**
    - Index to KV
    - Recovery
  - **Utils**
    - Log Module
    - Profiling Module
    - System Call



Solution

- **Hash Map**
  - $f: Key \rightarrow Index$



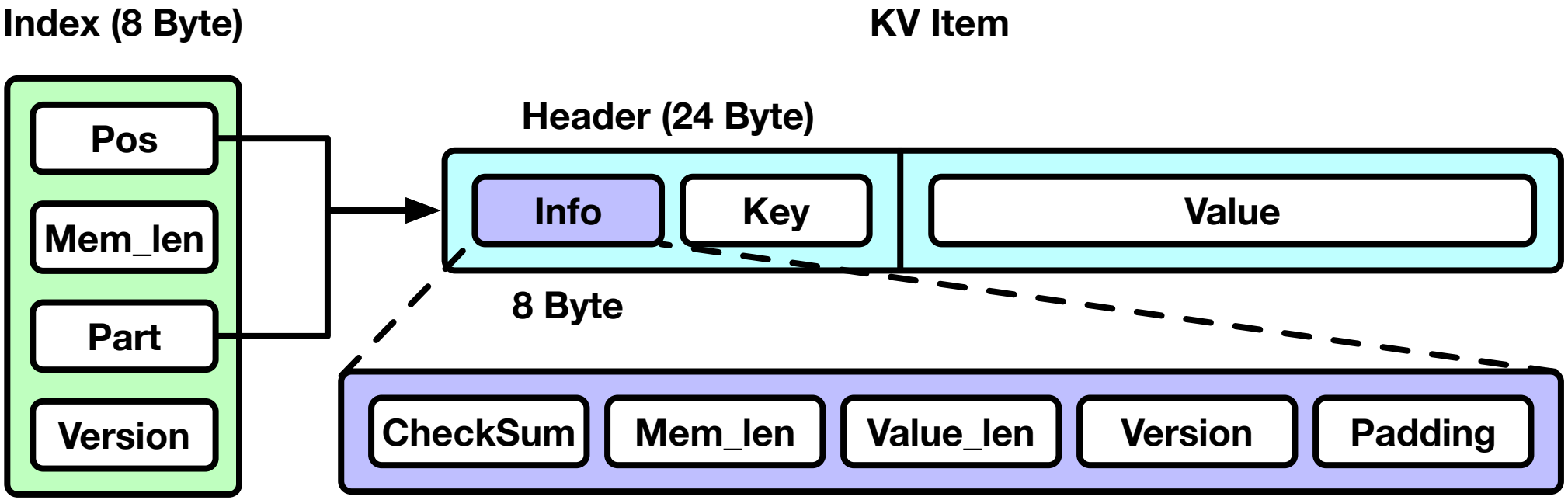
- CRC32 (SSE4.2)
- **Linear Probe**
- **Key Compare**

```
Key:
  uint64 k[2];
```

  - Redundancy **5%**, $\alpha = \frac{N_{KV}}{N * N_{line}} = \frac{229037390}{48097859 * 5} = \mathbf{95.24\%}$

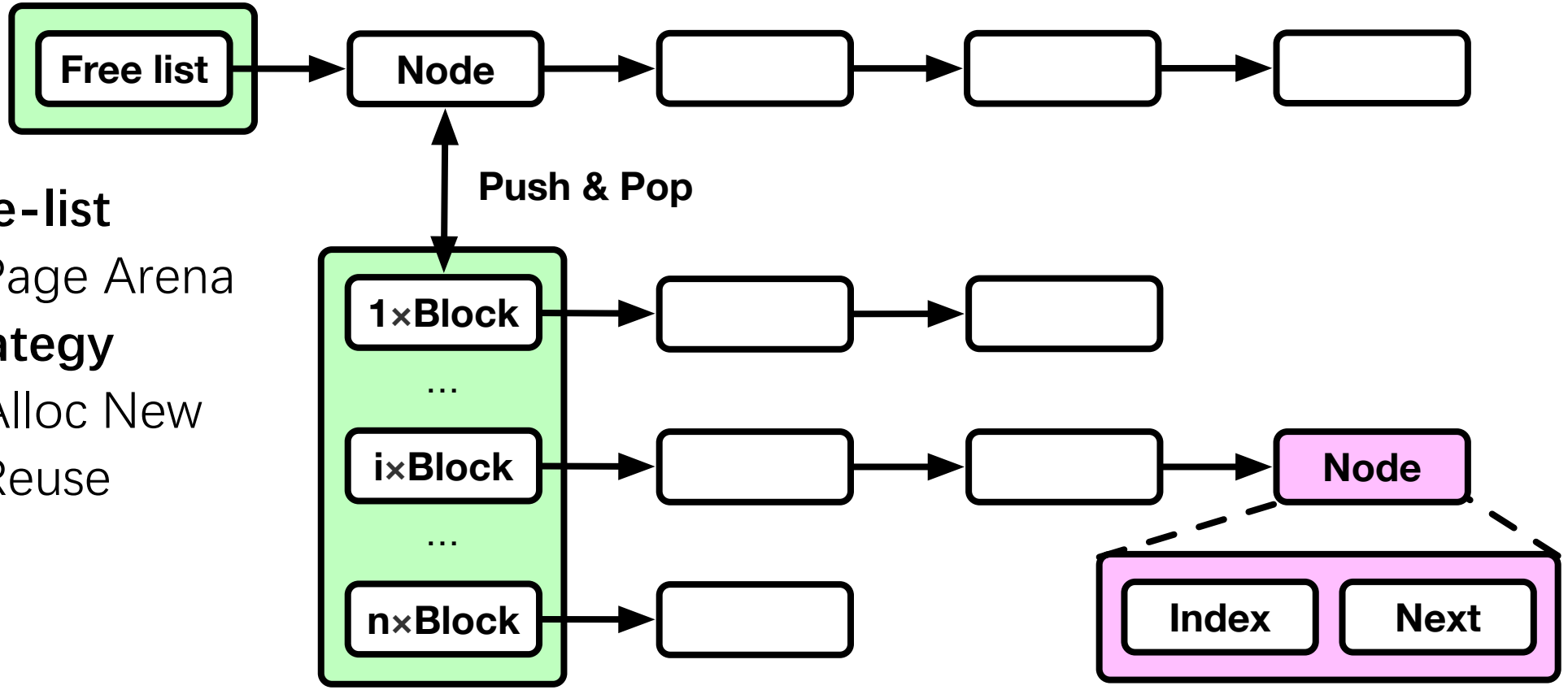
Solution

- **Heap Table**
  - **Concurrent** / **Sequential** Write (**Get**) & Read (**Set**)
  - Recovery
  - Memory Reuse (**Allocator**)



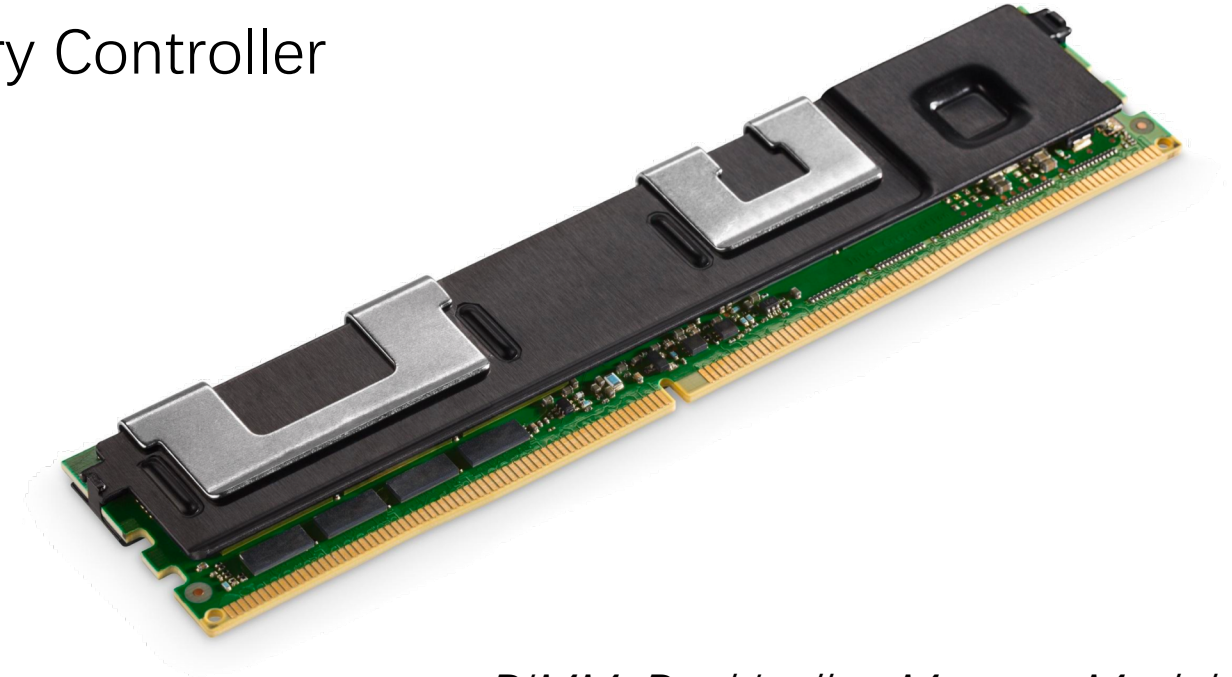
Solution

- **Allocator (Thread independent)**
  - Aligned (**32 Byte**) memory block
  - **Free-list**
    - Page Arena
  - **Strategy**
    - Alloc New
    - Reuse



What exactly is Optane DC PMEM?

- **Intel® Optane™ DC Persistent Memory Module**
  - Capacity & Persistence
  - **Byte**-granularity
  - DIMM* (DDR4) to CPU's Memory Controller
  - Memory Mode
    - DRAM Cache
  - **App-Direct Mode**
    - Persistent Device / Memory
    - File System
    - **Ext4 DAX**(Direct Access)
    - **User-space Persistent**

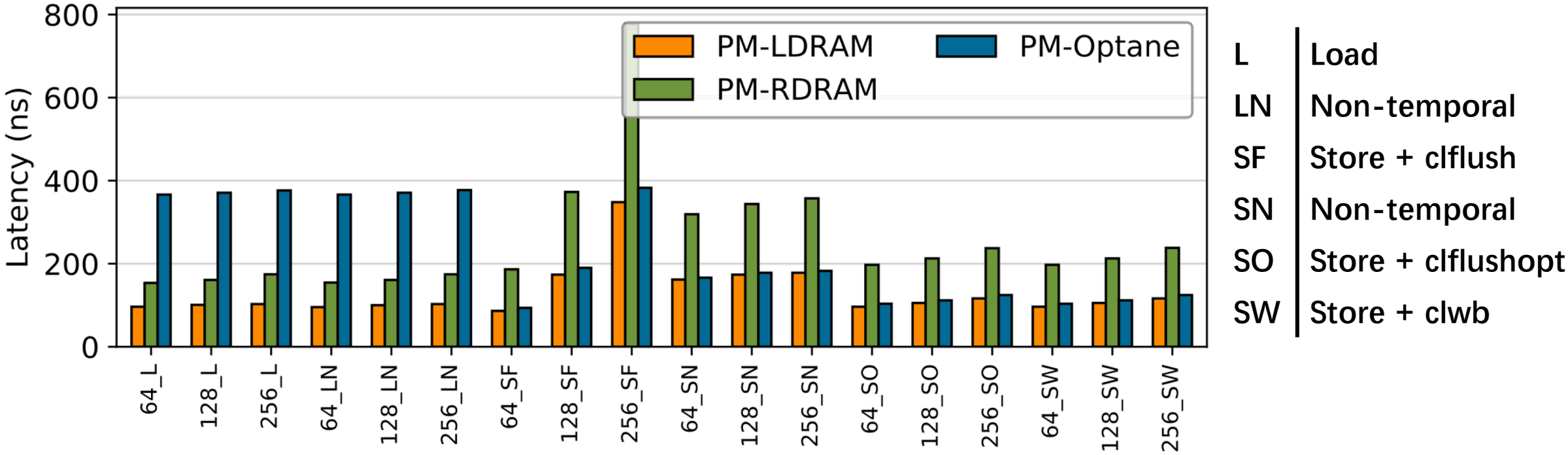


DIMM: Dual In-line Memory Module

[2] Izraelevitz, Joseph, et al. "Basic performance measurements of the intel optane DC persistent memory module." *arXiv preprint* arXiv:1903.05714 (2019).

Intel® Optane™ DC Persistent Memory Module

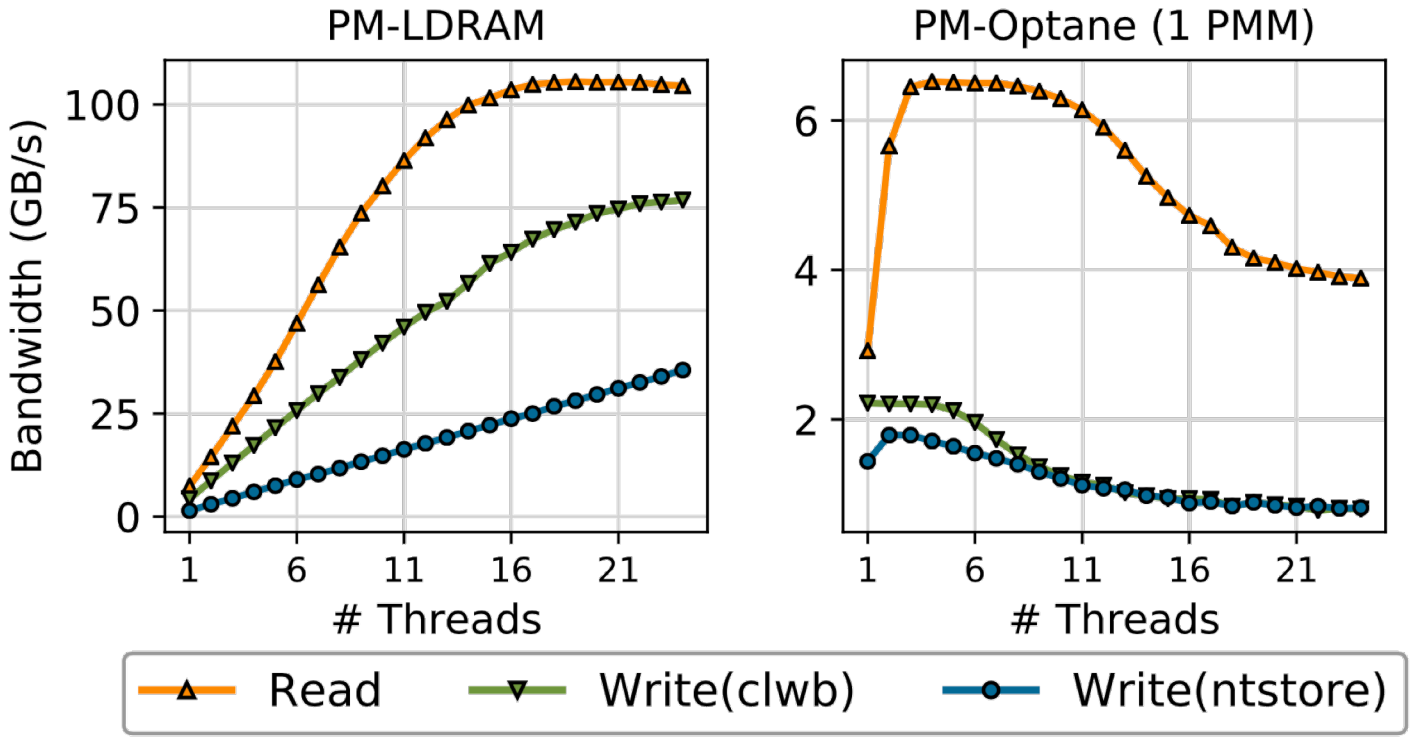
- Performance[2]
  - Load latency **305ns** (Rand) / **169ns** (Seq) DRAM **81ns** (Rand)
  - Store latency **94ns** DRAM **86ns**

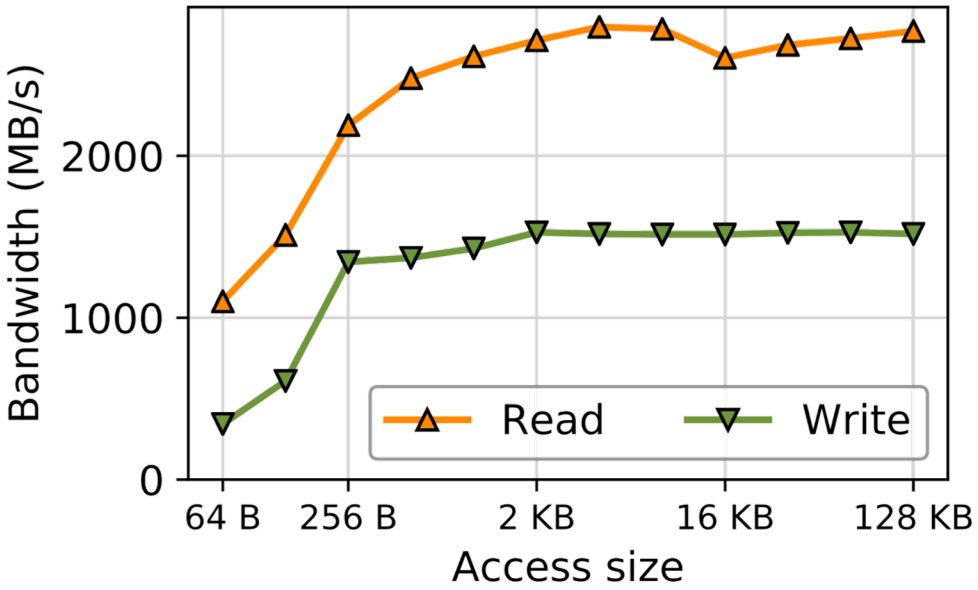


[2] Izraelevitz, Joseph, et al. "Basic performance measurements of the intel optane DC persistent memory module." *arXiv preprint* arXiv:1903.05714 (2019).

Intel® Optane™ DC Persistent Memory Module

- Performance[2]
  - Max Bandwidth **6.6 GB/s** Read, **2.3 GB/s** Write (Multi-thread, **AVX-512**)





[2] Izraelevitz, Joseph, et al. "Basic performance measurements of the intel optane DC persistent memory module." *arXiv preprint* arXiv:1903.05714 (2019).

xxHash

- Extremely Fast Hash Algorithm[4]
  - Completed the **SMHasher** test suite

Hash Name	Width	Large Data Bandwidth	Small Data Velocity	Comment
XXH3	**64**	**59.4 GB/s**	**133.1**	**AVX2 (optional)**
XXH128	128	57.9 GB/s	118.1	AVX2 (optional)
RAM Seq read	N/A	28.0 GB/s	N/A	*for reference*
CRC32C (hw)	**32**	**13.0 GB/s**	**57.9**	**SSE4.2**
Murmur3	32	3.9 GB/s	56.1	
FNV64	64	1.2 GB/s	62.7	Poor avalanche properties
SHA1	160	0.8 GB/s	5.6	Cryptographic but broken
MD5	128	0.6 GB/s	7.8	Cryptographic but broken

[4] Collet, Yann. "xxHash: Extremely Fast Hash Algorithm." (2016).