GPHash: An Efficient Hash Index for GPU with Byte-Granularity Persistent Memory

<u>Menglei Chen</u>, Yu Hua, Zhangyu Chen, Ming Zhang, Gen Dong

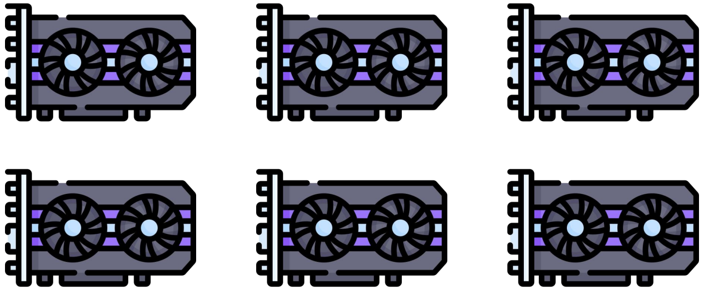
Huazhong University of Science and Technology, China

23rd USENIX Conference on File and Storage Technologies (**FAST**), 2025

Various Applications Are Powered by GPUs

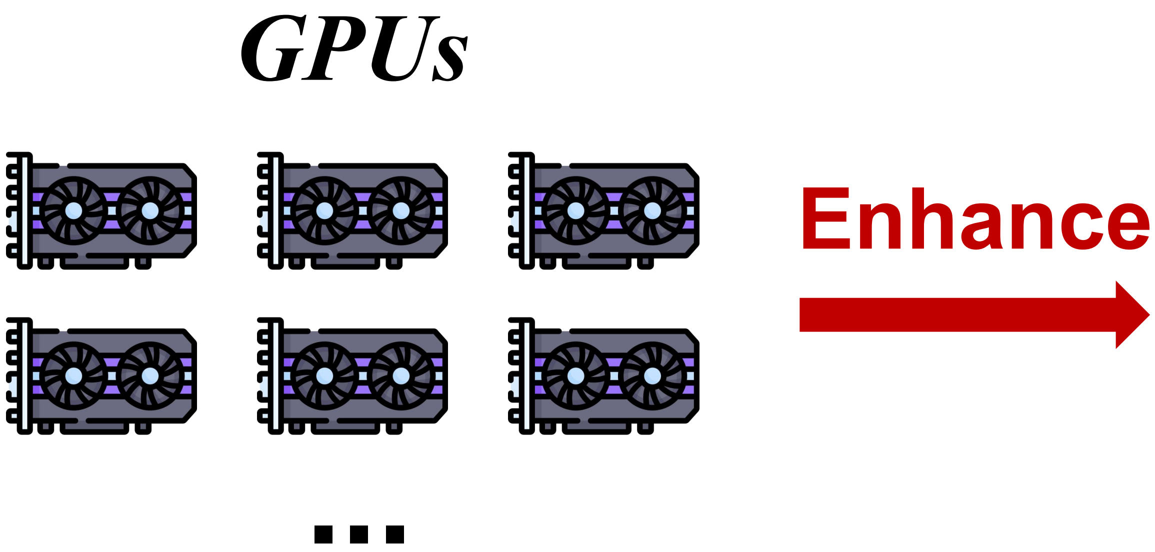
Various Applications Are Powered by GPUs

GPUs



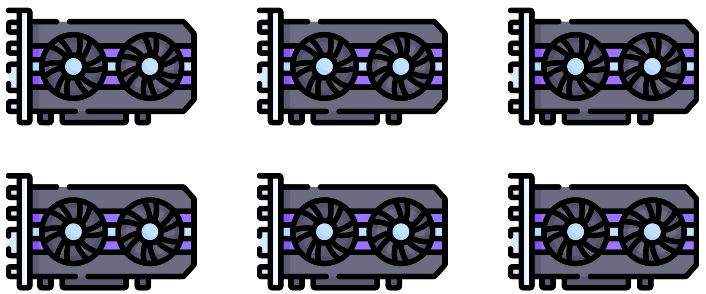
...

Various Applications Are Powered by GPUs



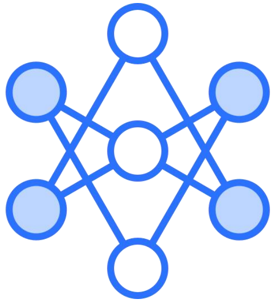
Various Applications Are Powered by GPUs

GPUs

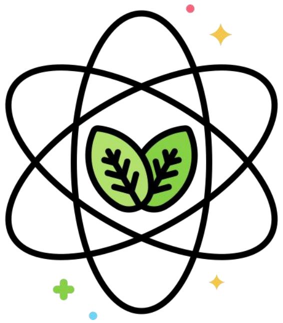


...

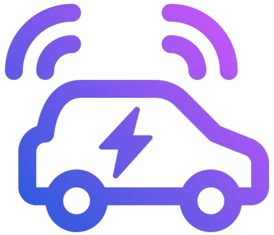
Enhance



Deep Neural Network



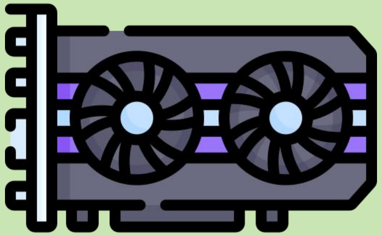
Scientific Computing



Autonomous driving

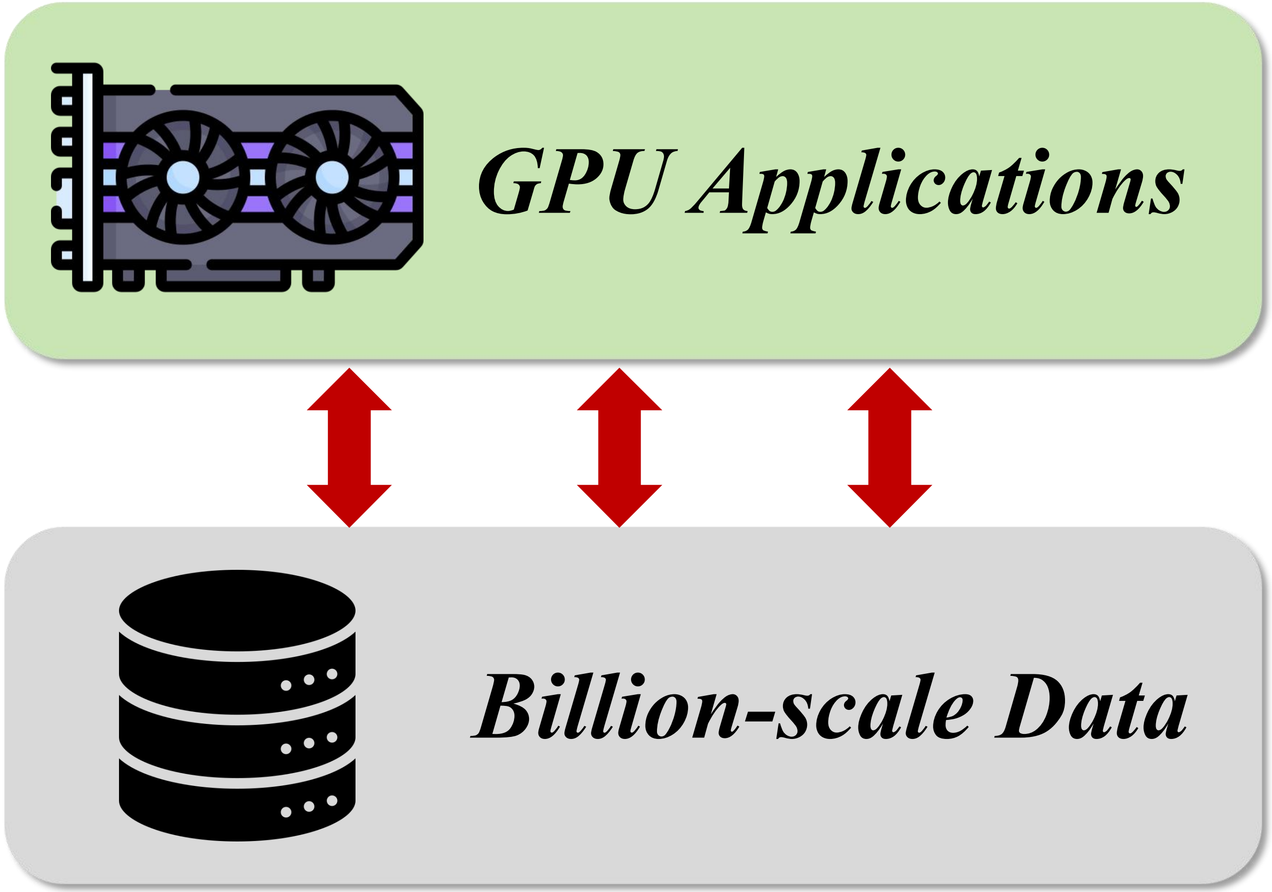
Unprecedented Data Scale of GPU App.

Unprecedented Data Scale of GPU App.

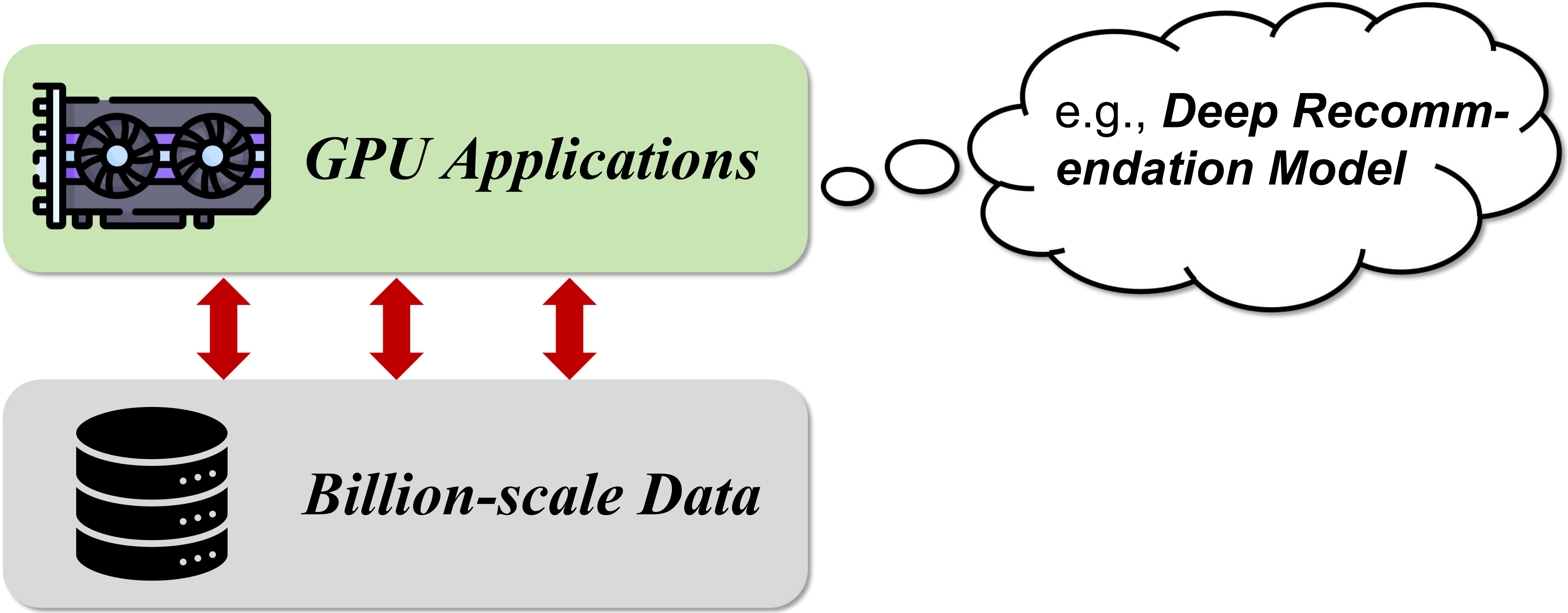


GPU Applications

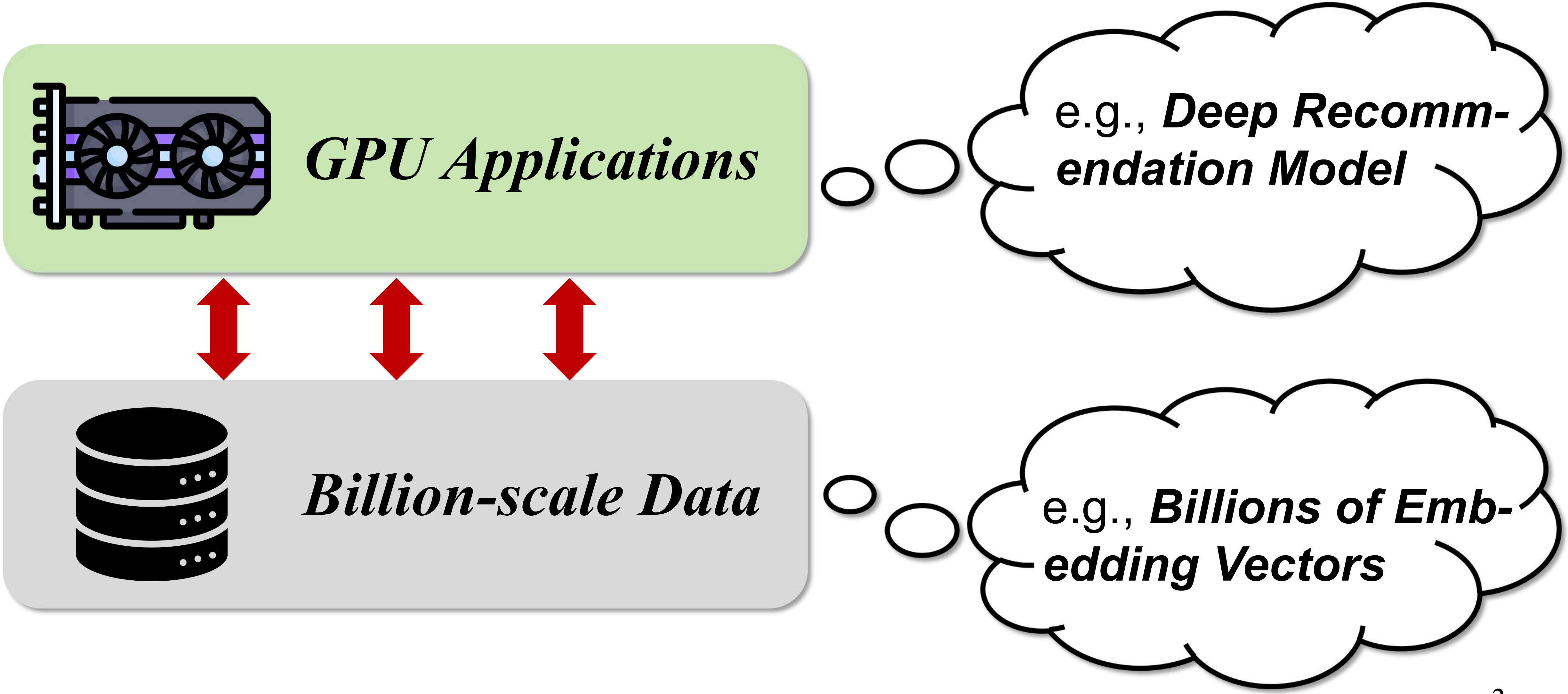
Unprecedented Data Scale of GPU App.



Unprecedented Data Scale of GPU App.



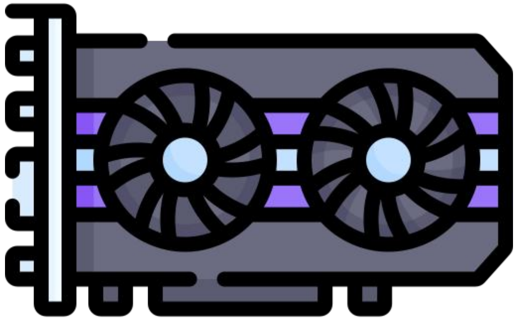
Unprecedented Data Scale of GPU App.



Existing Data Management of GPU App.

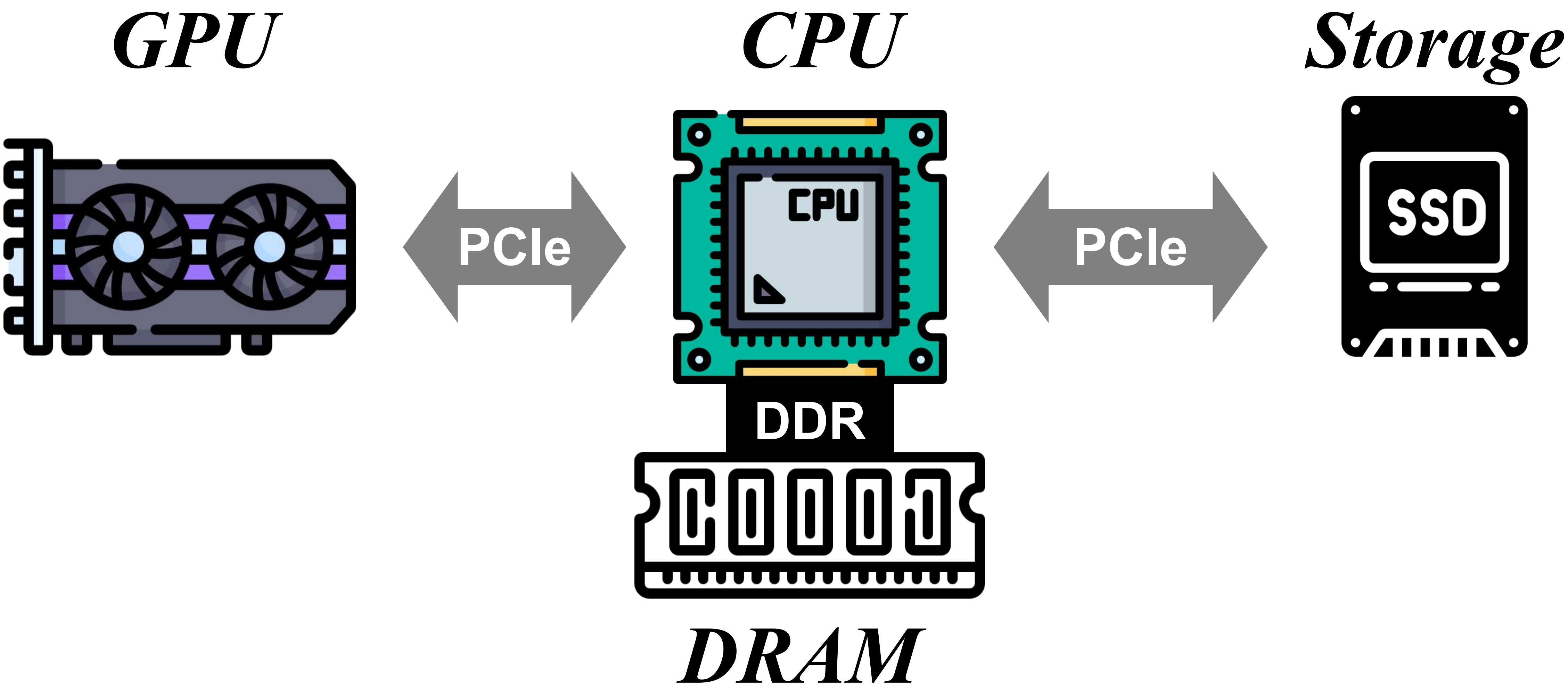
Existing Data Management of GPU App.

GPU

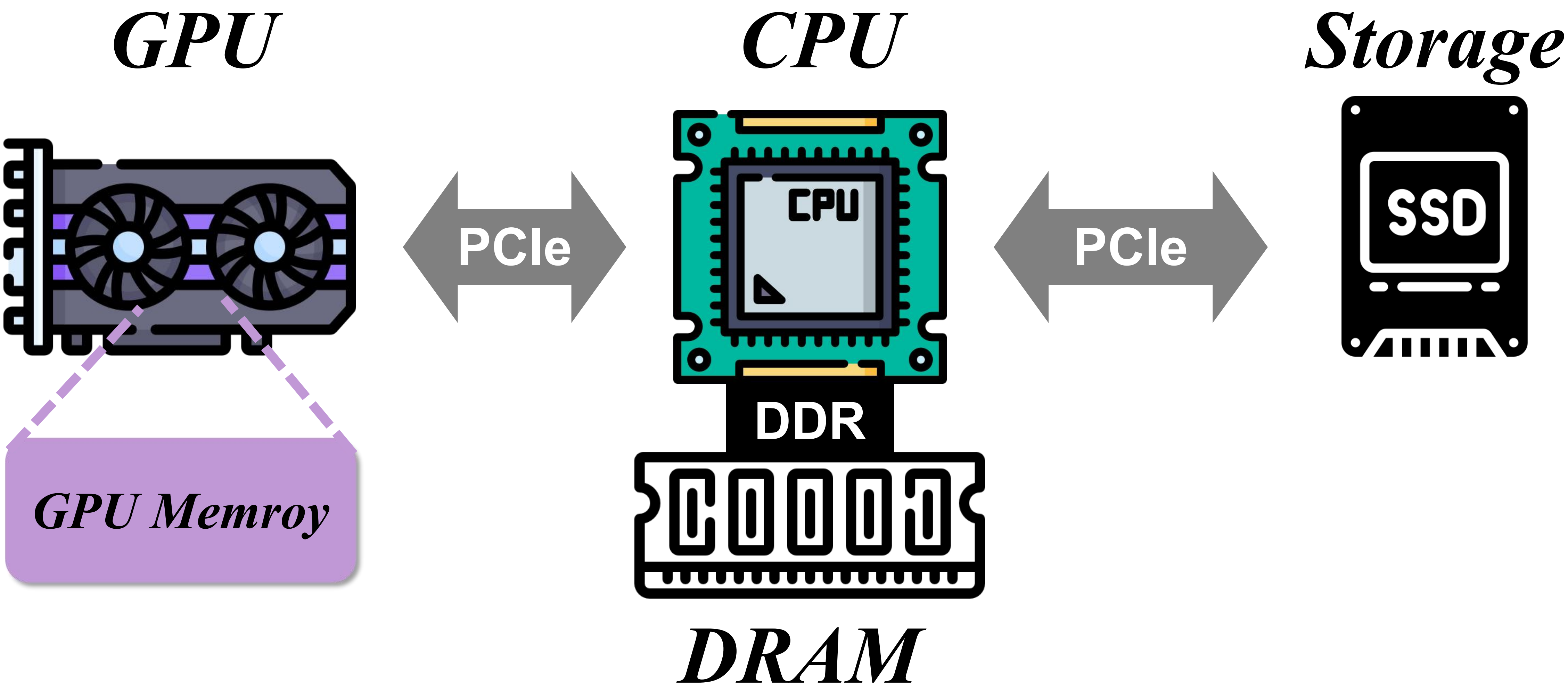


Storage

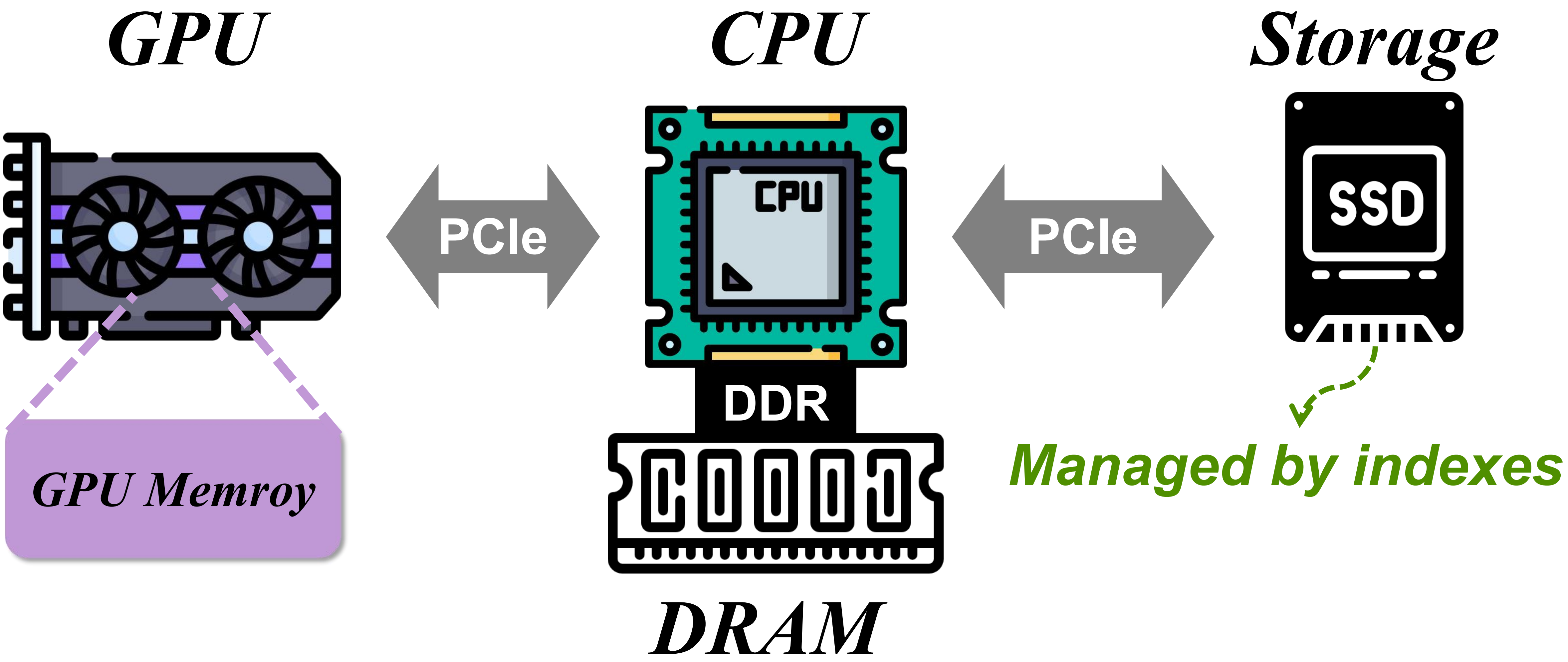
Existing Data Management of GPU App.



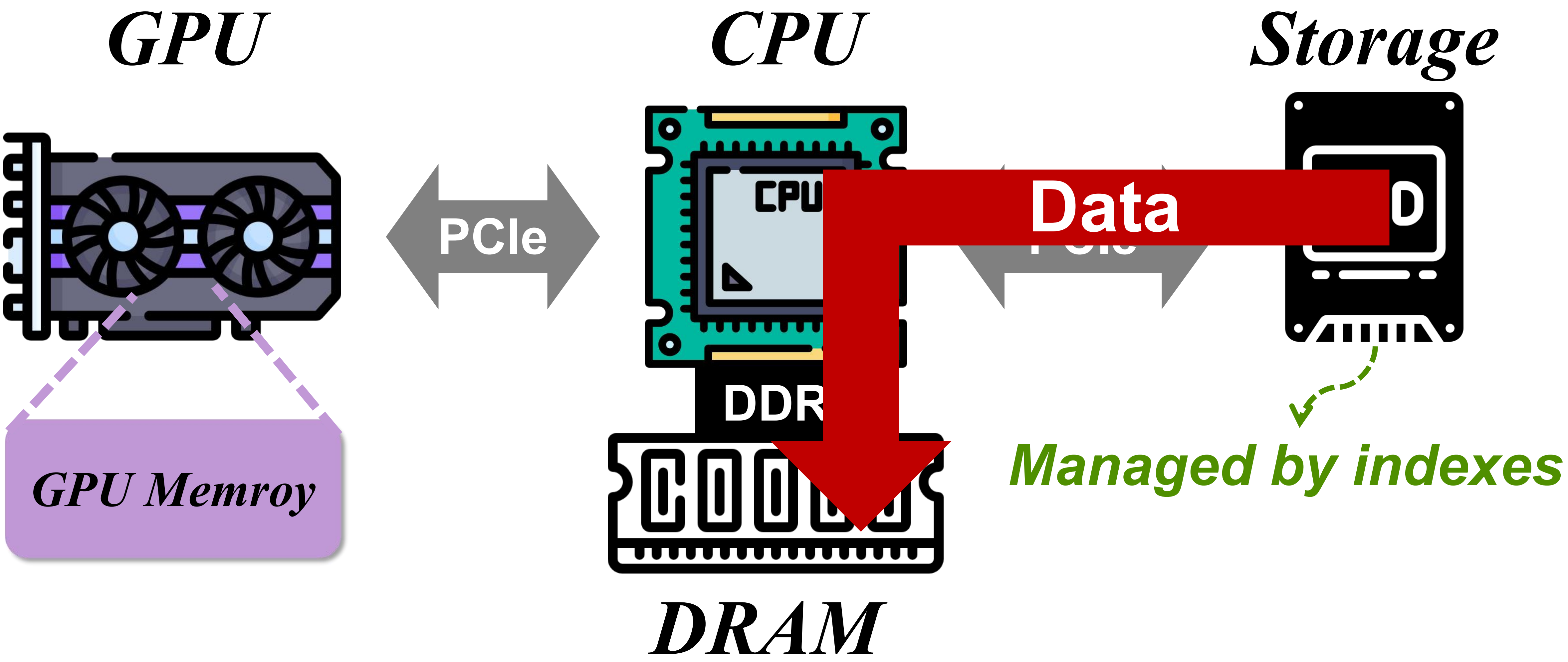
Existing Data Management of GPU App.



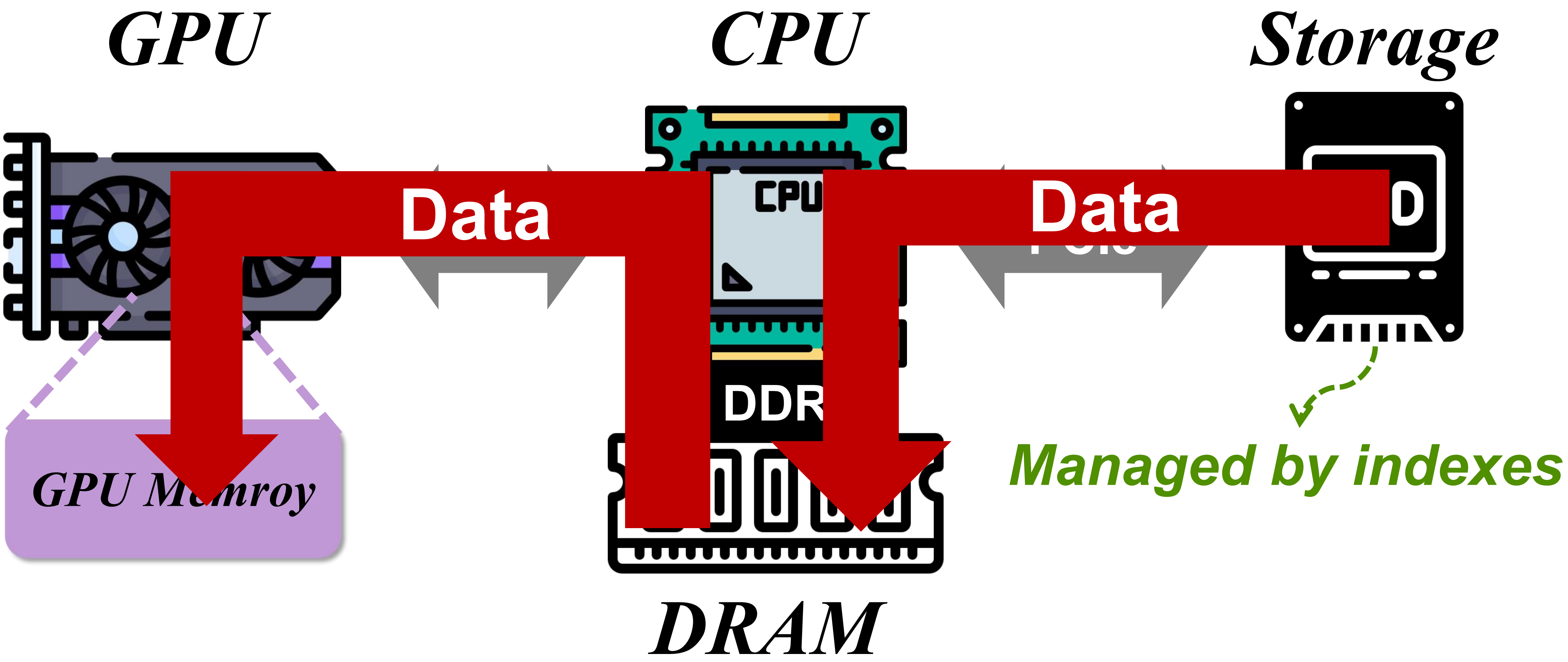
Existing Data Management of GPU App.



Existing Data Management of GPU App.



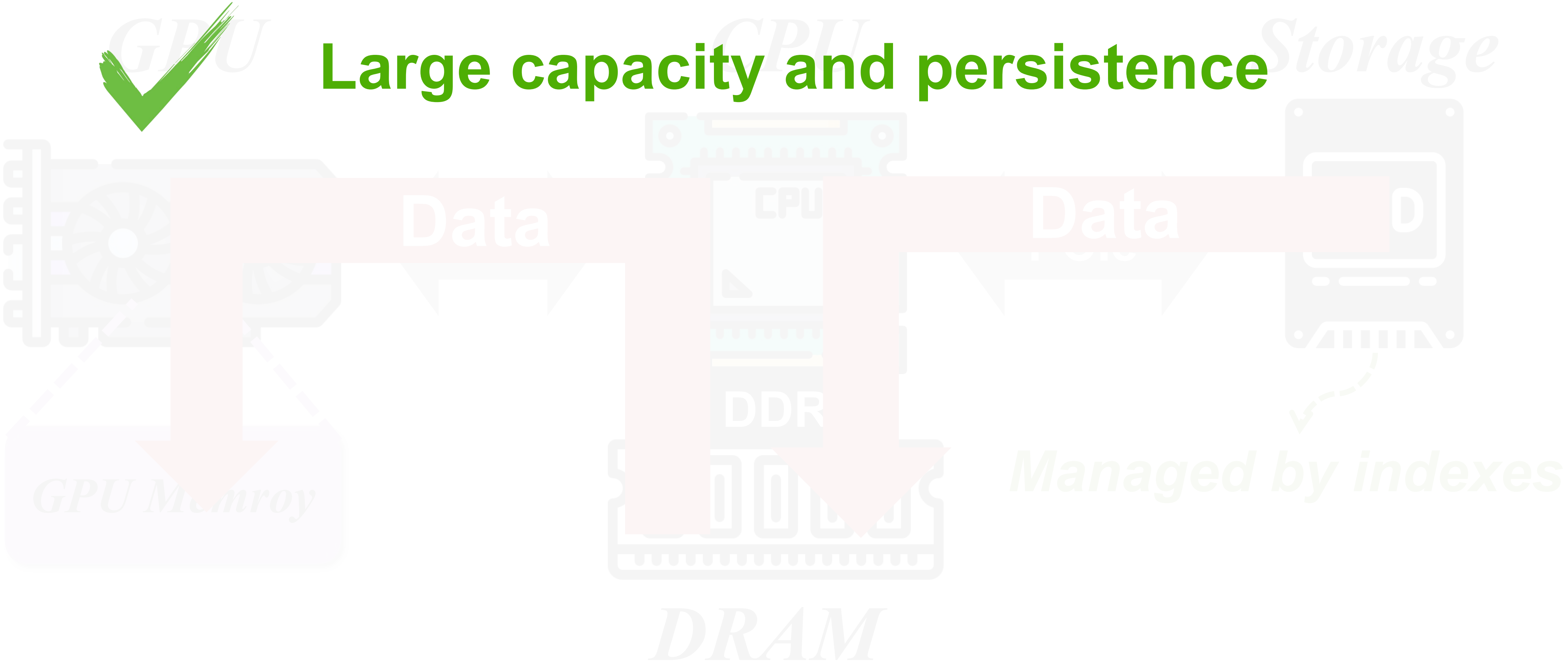
Existing Data Management of GPU App.



Existing Data Management of GPU App.

Existing Data Management of GPU App.

✓ Large capacity and persistence



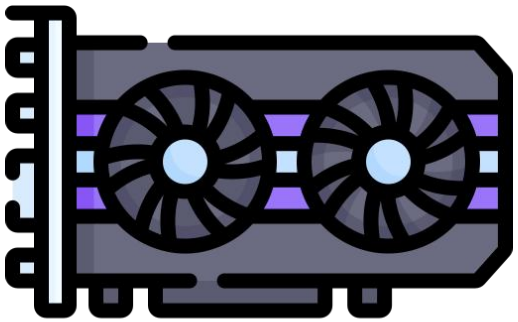
Existing Data Management of GPU App.

- ✓ Large capacity and persistence
- ✗ High overhead for data transfer
- ✗ Extra CPU consumption

Direct Data Access from GPU

Direct Data Access from GPU

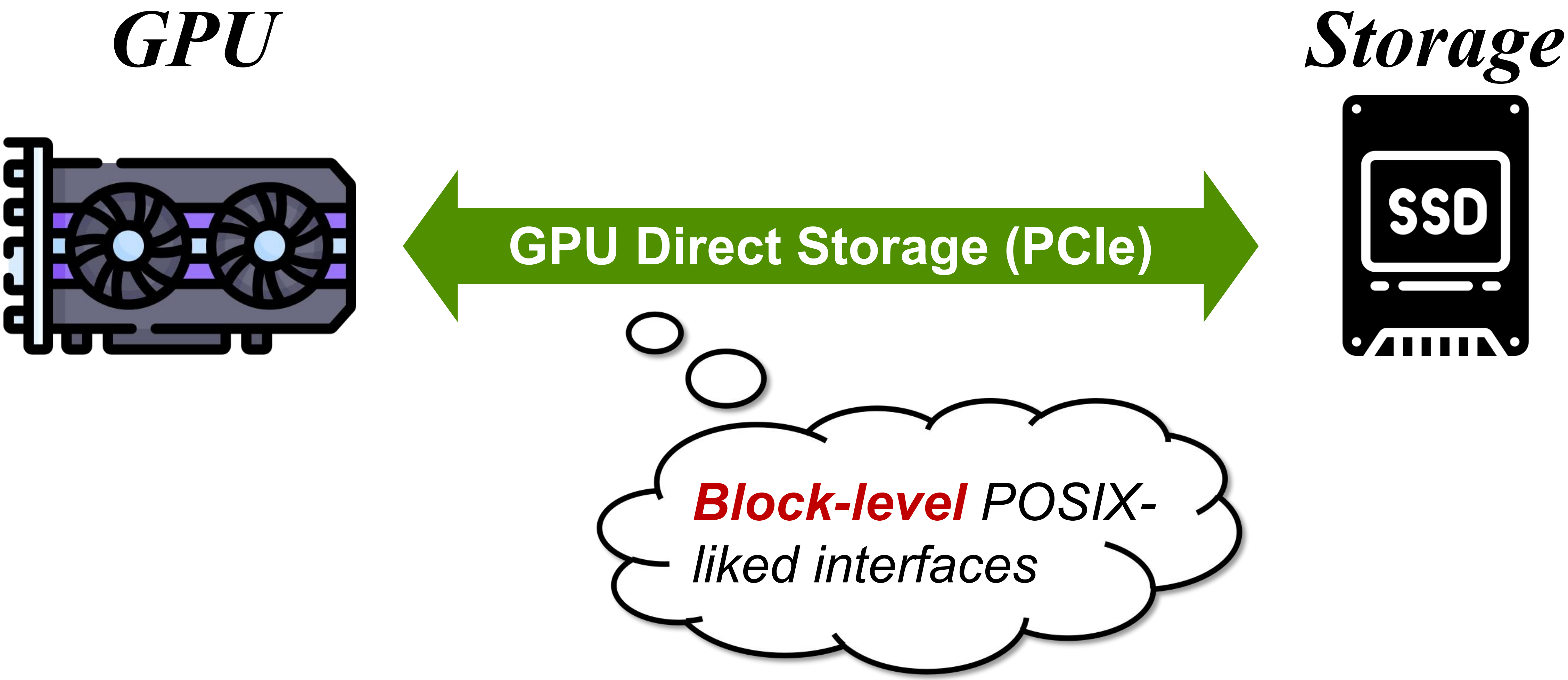
GPU



Storage

Direct Data Access from GPU

Direct Data Access from GPU



Direct Data Access from GPU

Direct Data Access from GPU

- ✓ Large capacity and persistence
- ✓ Cost-efficient data transfer

Direct Data Access from GPU

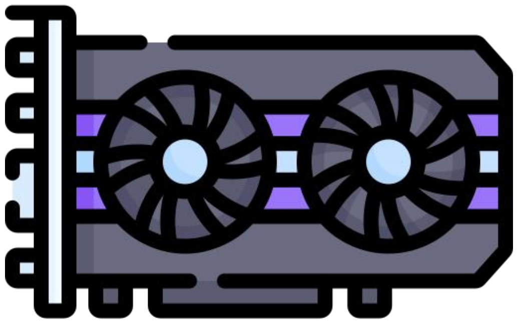
- ✓ Large capacity and persistence
- ✓ Cost-efficient data transfer
- ✗ Hard to program data structure
- ✗ Transfer extraneous data

GPU with Persistent Memory (GPM)

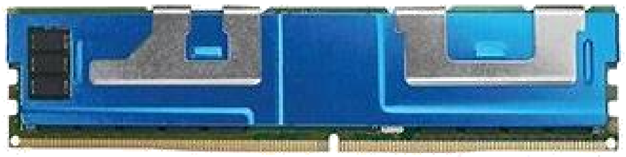
[1] GPM: Leveraging Persistent Memory from a GPU [ASPLOS' 22]
[2] Scoped Buffered Persistency Model for GPUs [ASPLOS' 23]

GPU with Persistent Memory (GPM)

GPU

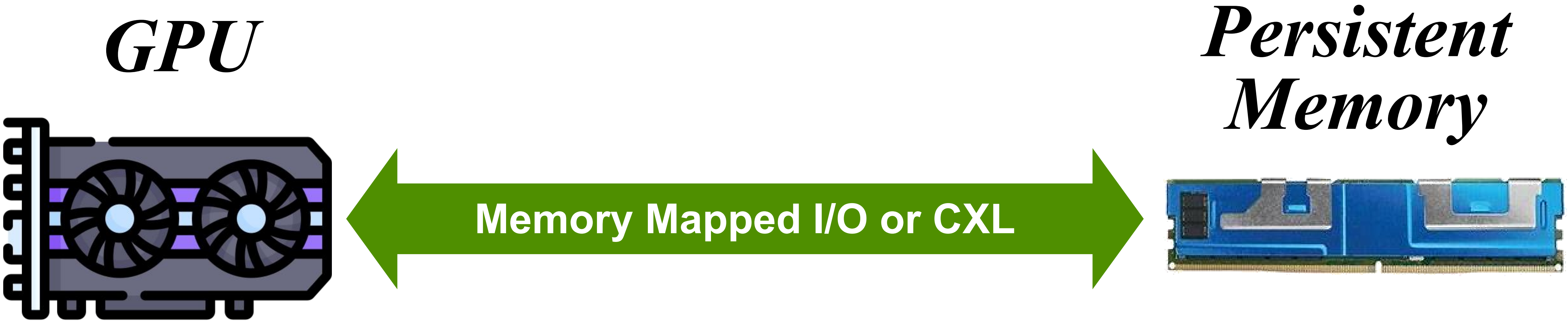


Persistent Memory



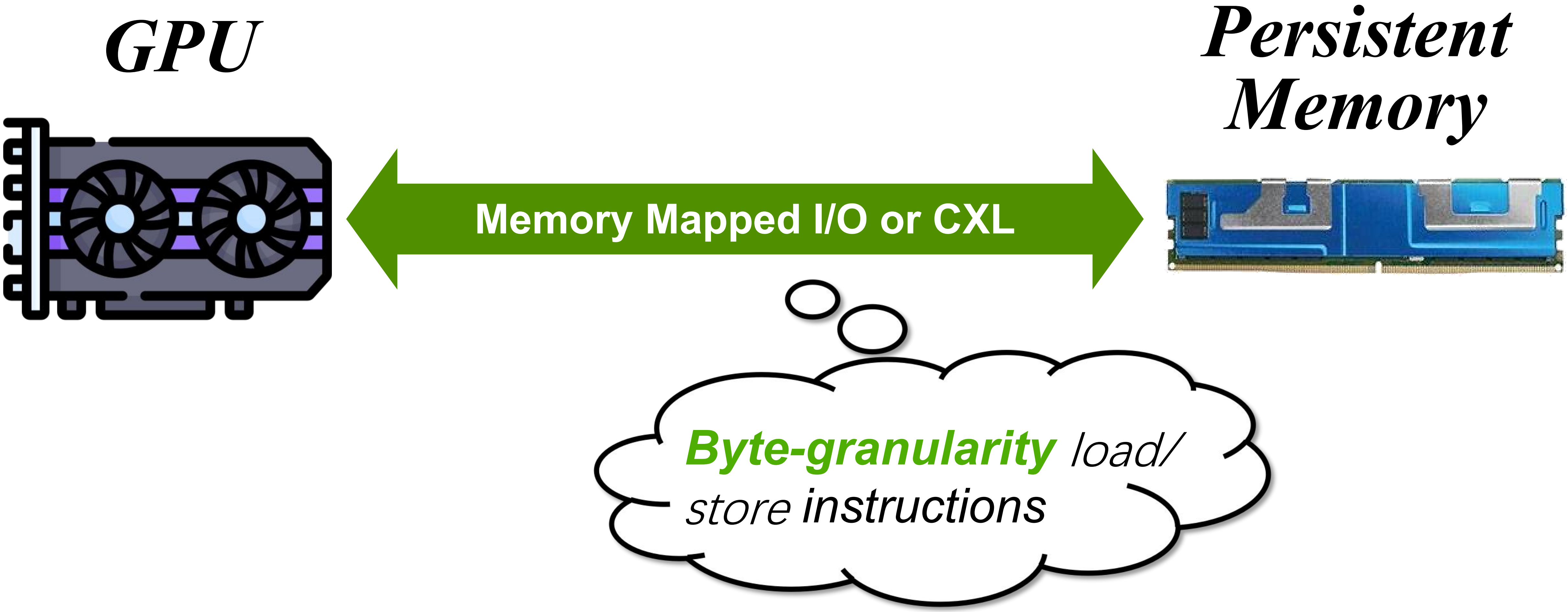
[1] GPM: Leveraging Persistent Memory from a GPU [ASPLOS' 22]
[2] Scoped Buffered Persistency Model for GPUs [ASPLOS' 23]

GPU with Persistent Memory (GPM)



[1] GPM: Leveraging Persistent Memory from a GPU [ASPLOS' 22]
[2] Scoped Buffered Persistency Model for GPUs [ASPLOS' 23]

GPU with Persistent Memory (GPM)



[1] GPM: Leveraging Persistent Memory from a GPU [ASPLOS' 22]
[2] Scoped Buffered Persistency Model for GPUs [ASPLOS' 23]

GPU with Persistent Memory (GPM)

[1] GPM: Leveraging Persistent Memory from a GPU [ASPLOS' 22]
[2] Scoped Buffered Persistency Model for GPUs [ASPLOS' 23]

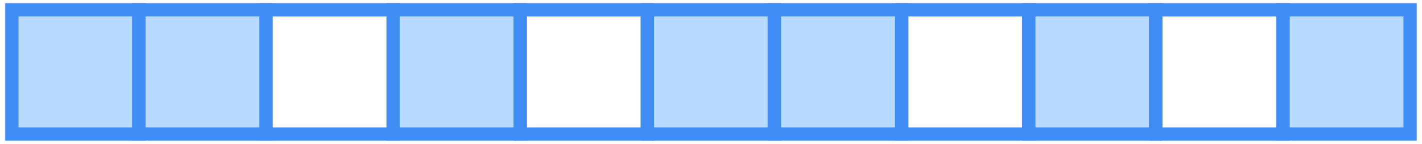
GPU with Persistent Memory (GPM)

- ✓ Large capacity and persistence
- ✓ Cost-efficient and fine-grained data transfer
- ✓ Easy to program data structure

[1] GPM: Leveraging Persistent Memory from a GPU [ASPLOS' 22]
[2] Scoped Buffered Persistency Model for GPUs [ASPLOS' 23]

Hash Index

Hash Index



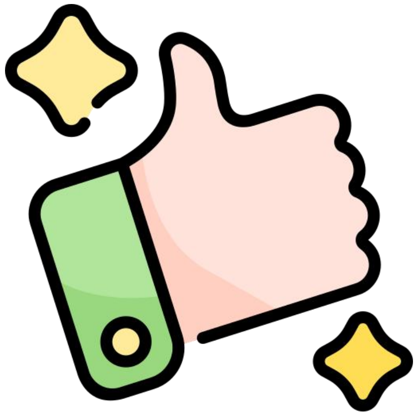
Hash Index

Hash Index

- ✓ Constant-scale point query
- ✓ Good for parallel access

Hash Index

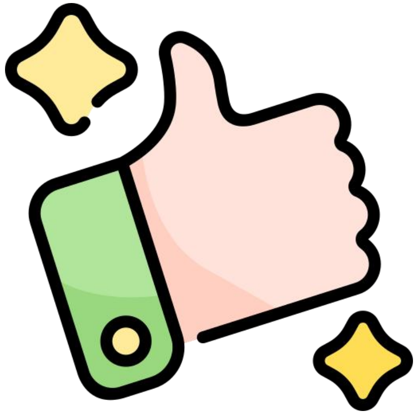
GPM Hash Index



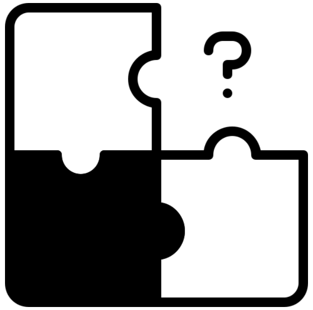
- ✓ Constant-scale point query
- ✓ Good for parallel access

Hash Index

GPM Hash Index



- Constant-scale point query
- Good for parallel access



However, it is non-trivial to implement an efficient GPM hash index

Challenge 1: **Agnostic to GPU Execution Manner**

Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

Challenge 1: Agnostic to GPU Execution Manner

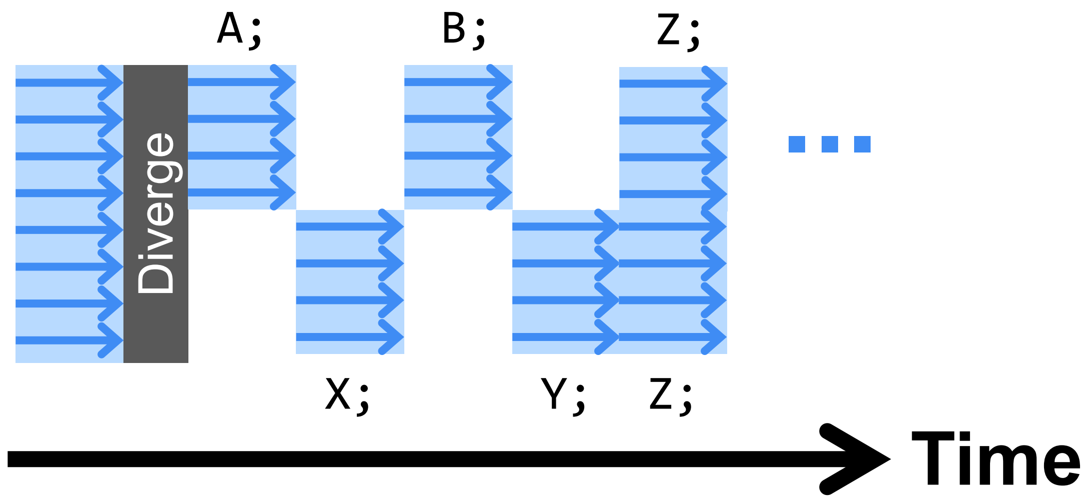
Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```



Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```


Coalesced memory accesses

Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

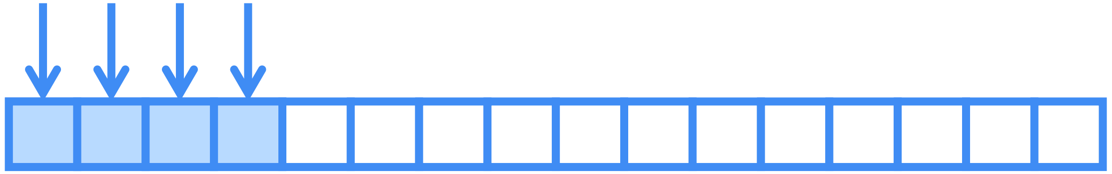

Coalesced memory accesses

Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```


Coalesced memory accesses

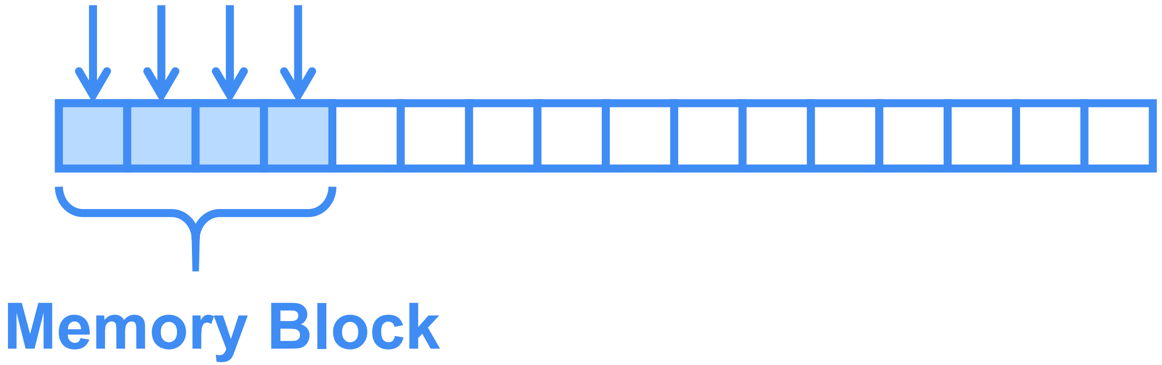


Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```


Coalesced memory accesses

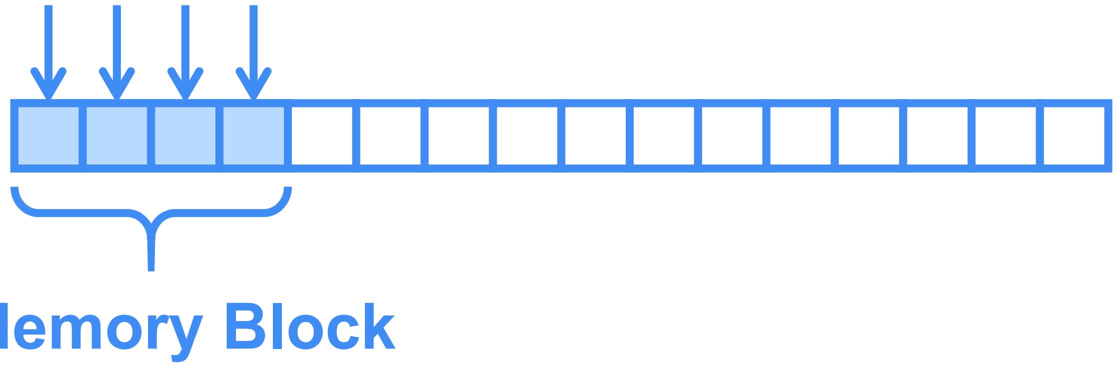


Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```


Coalesced memory accesses



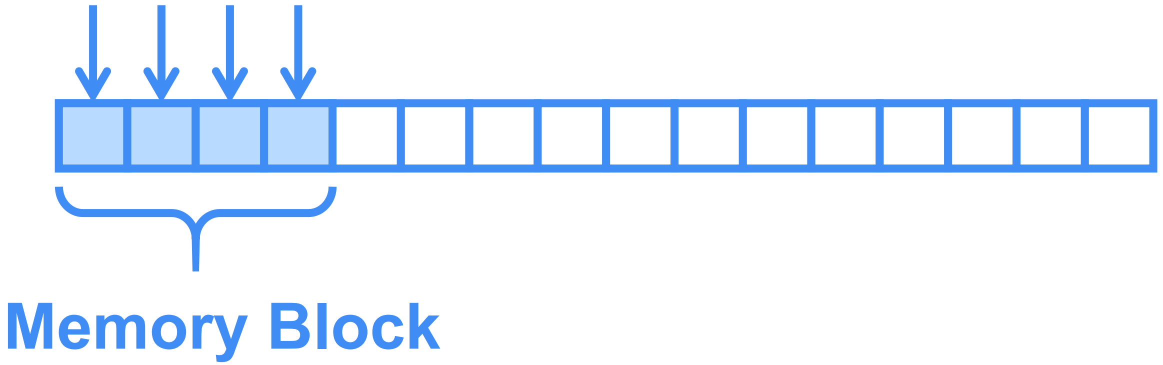
Uncoalesced memory accesses

Challenge 1: Agnostic to GPU Execution Manner

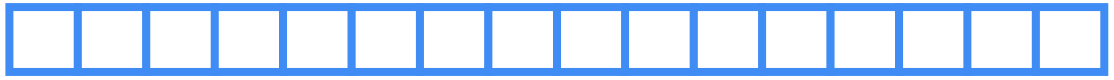
Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```


Coalesced memory accesses



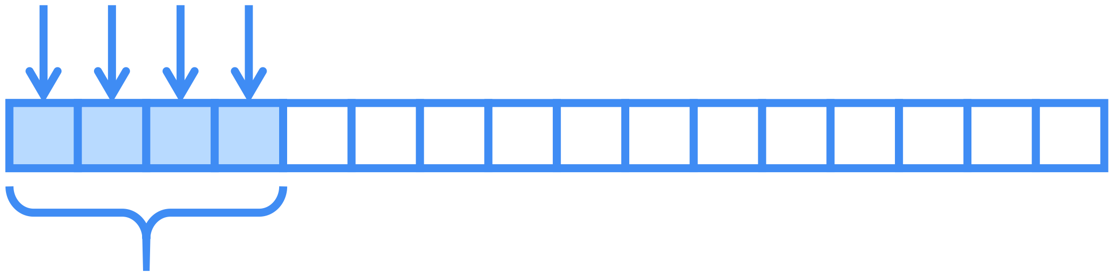
Uncoalesced memory accesses



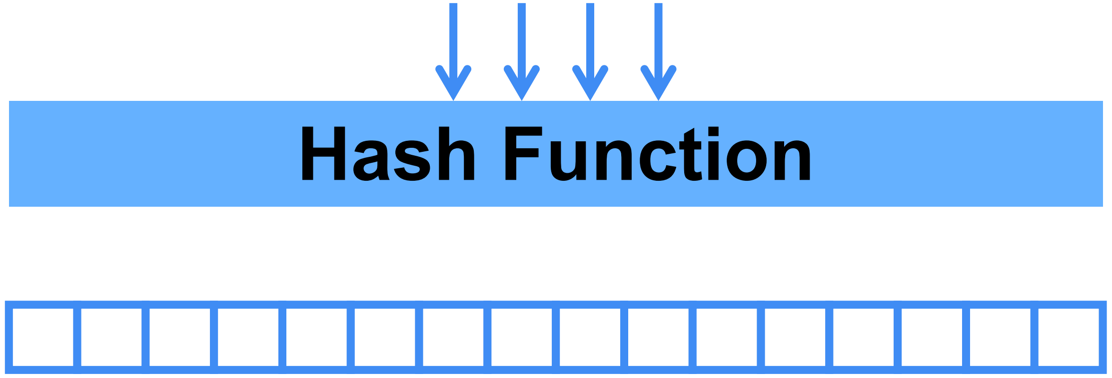
Challenge 1: Agnostic to GPU Execution Manner

Warp Divergence

Coalesced memory accesses



Uncoalesced memory accesses

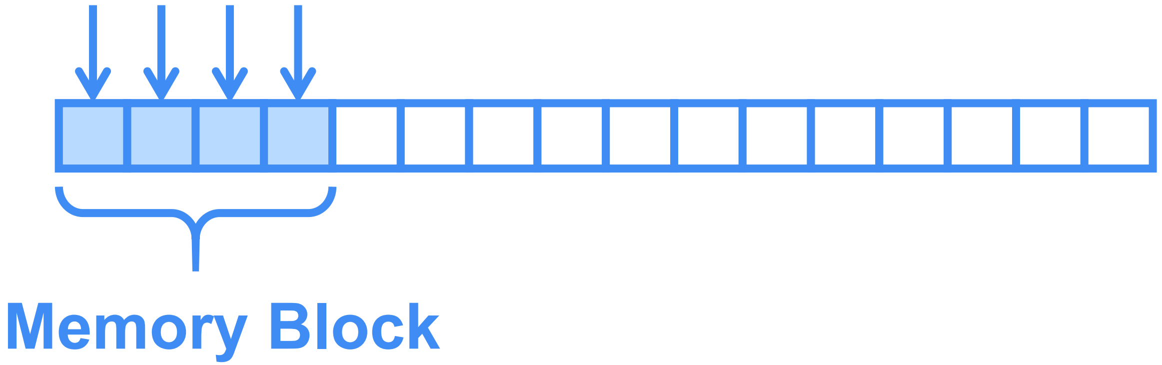


Challenge 1: Agnostic to GPU Execution Manner

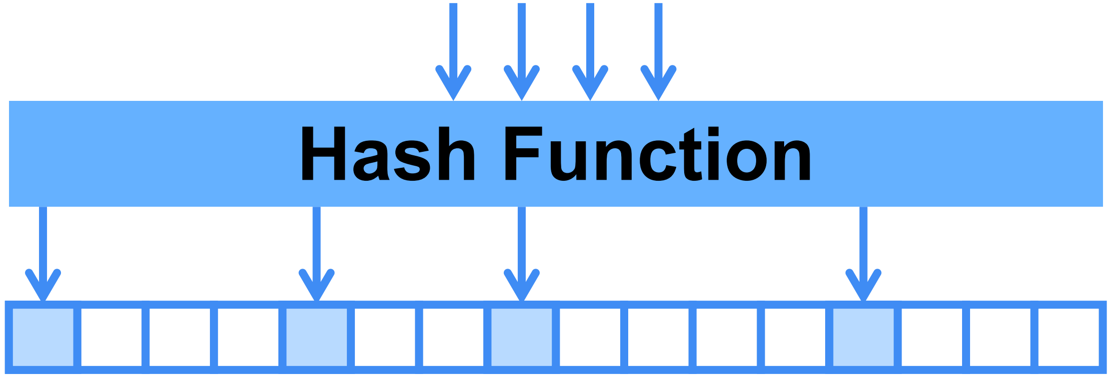
Warp Divergence

```
if (thread_id < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```


Coalesced memory accesses



Uncoalesced memory accesses



Challenge 1: **Agnostic to GPU Execution Manner**

Severe ***warp divergence*** and ***uncoalesced memory accesses*** lead to **Performance Degradation**

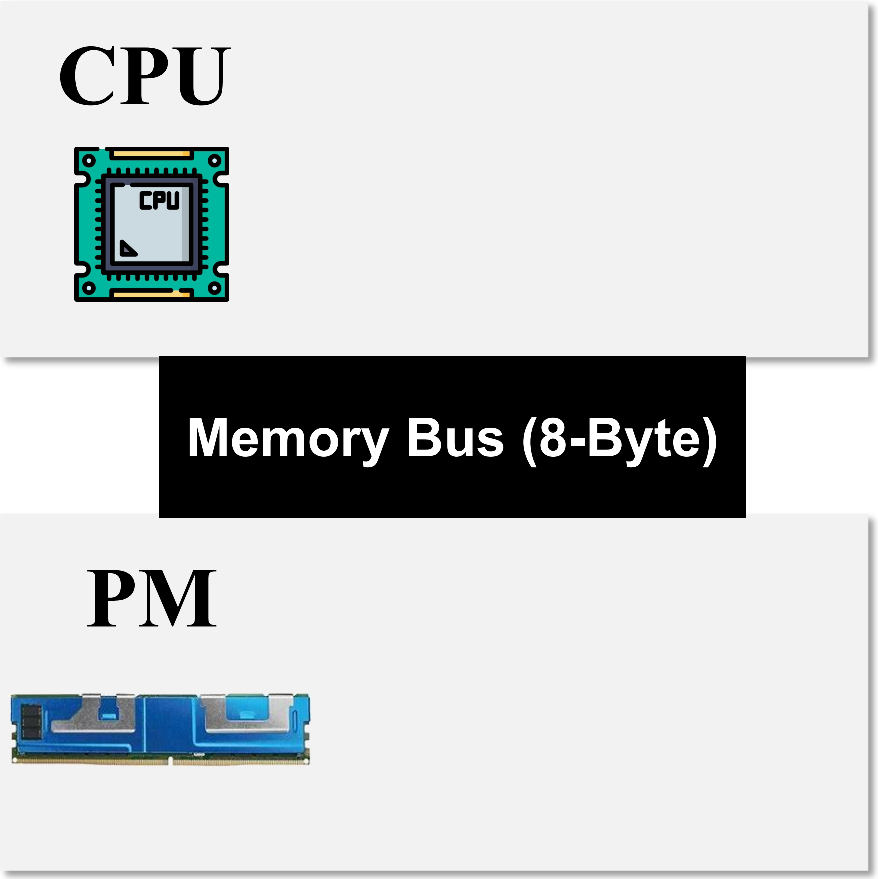
Challenge 2: Ensure Crash Consistency

Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee

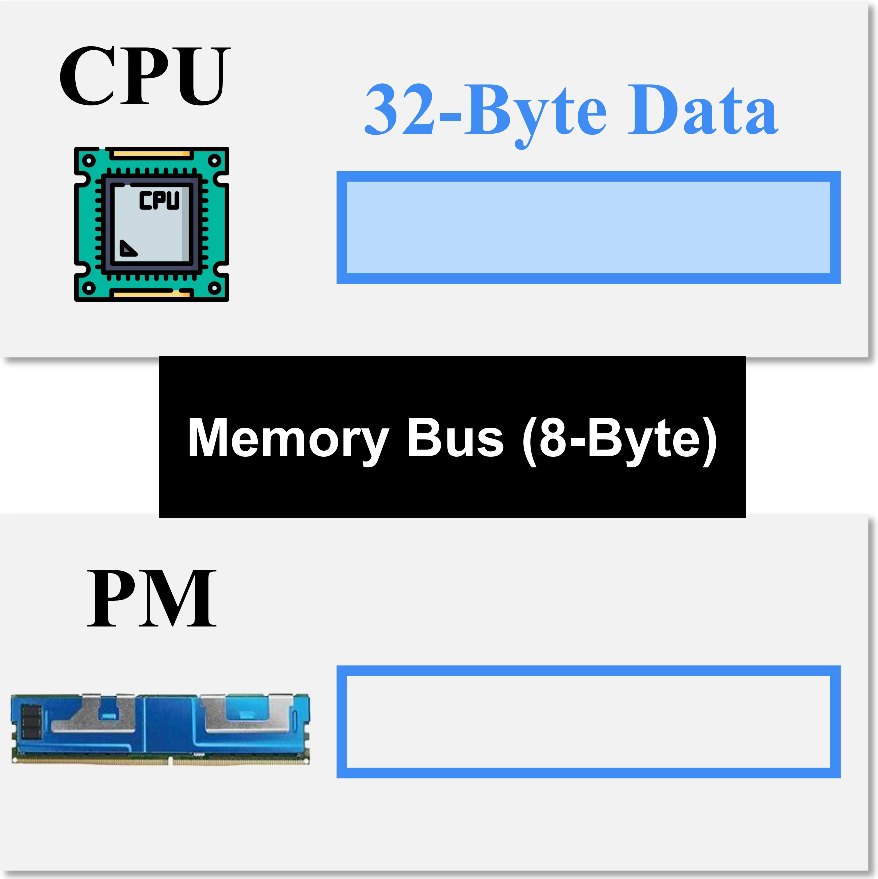
Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee



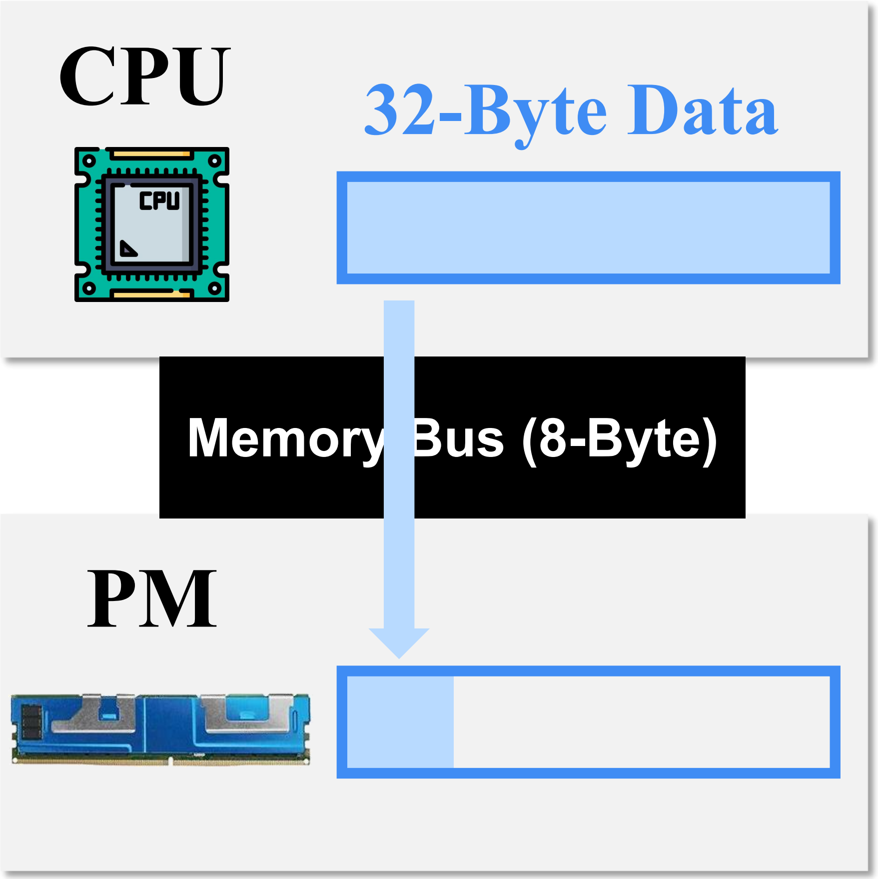
Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee



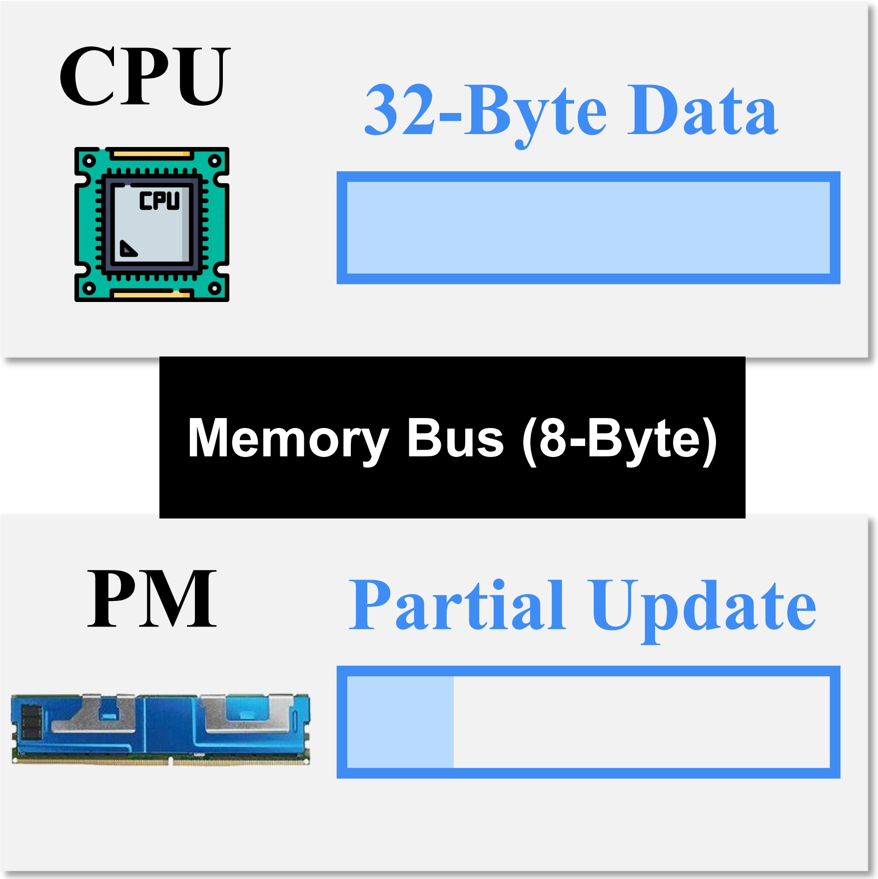
Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee



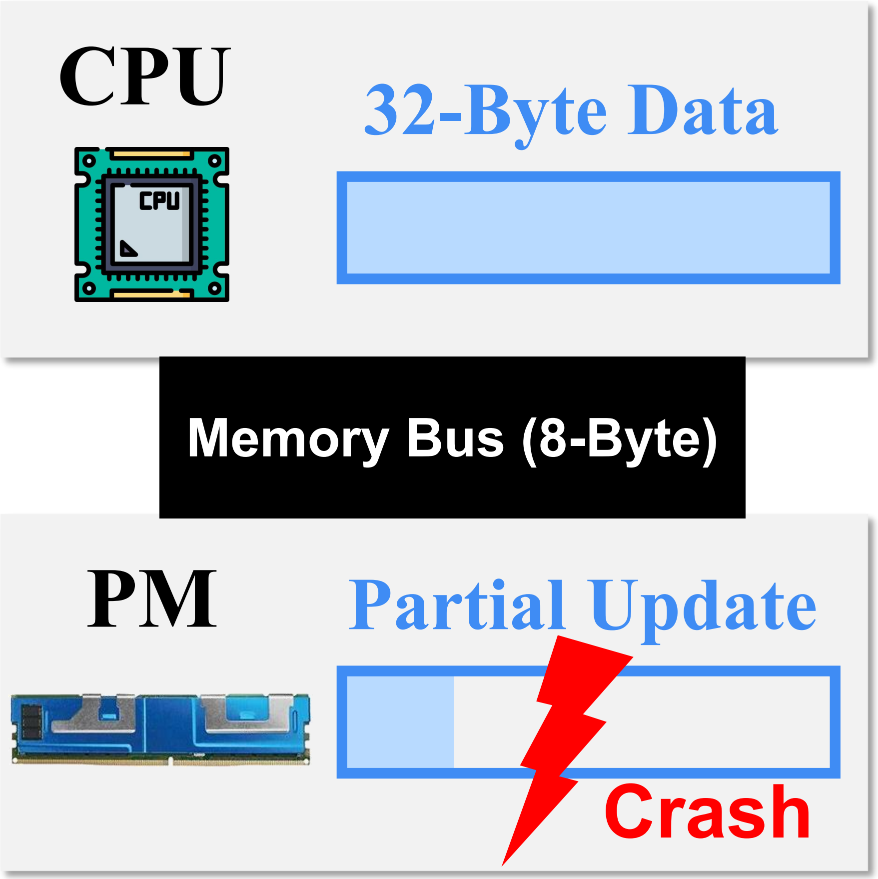
Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee



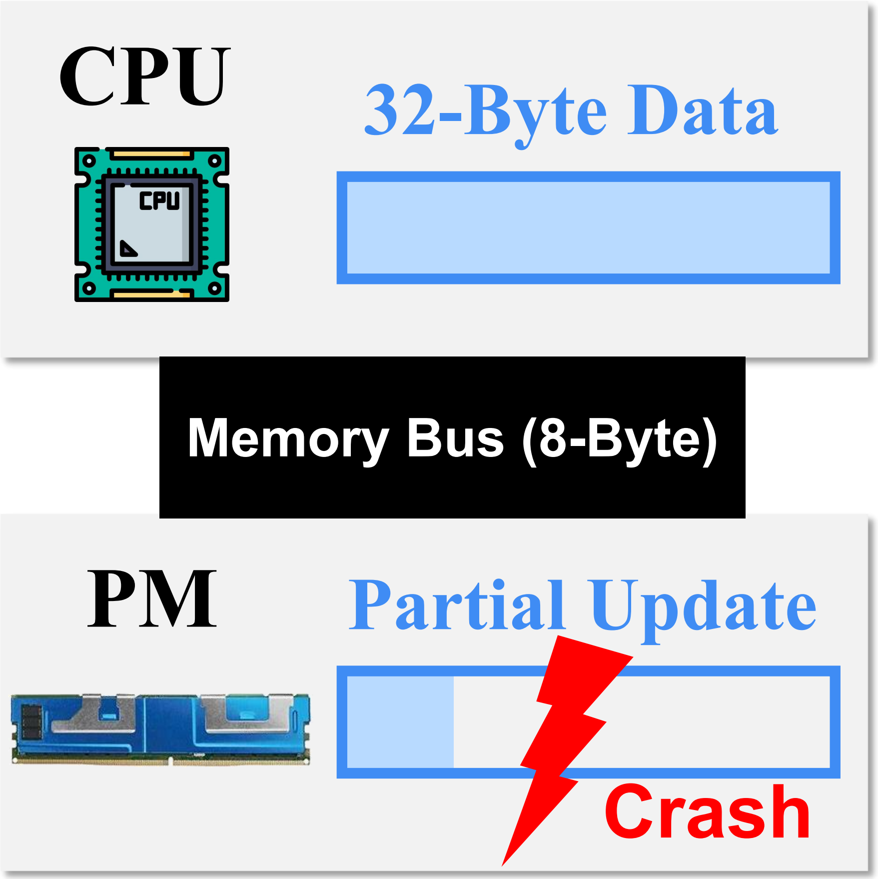
Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee



Challenge 2: Ensure Crash Consistency

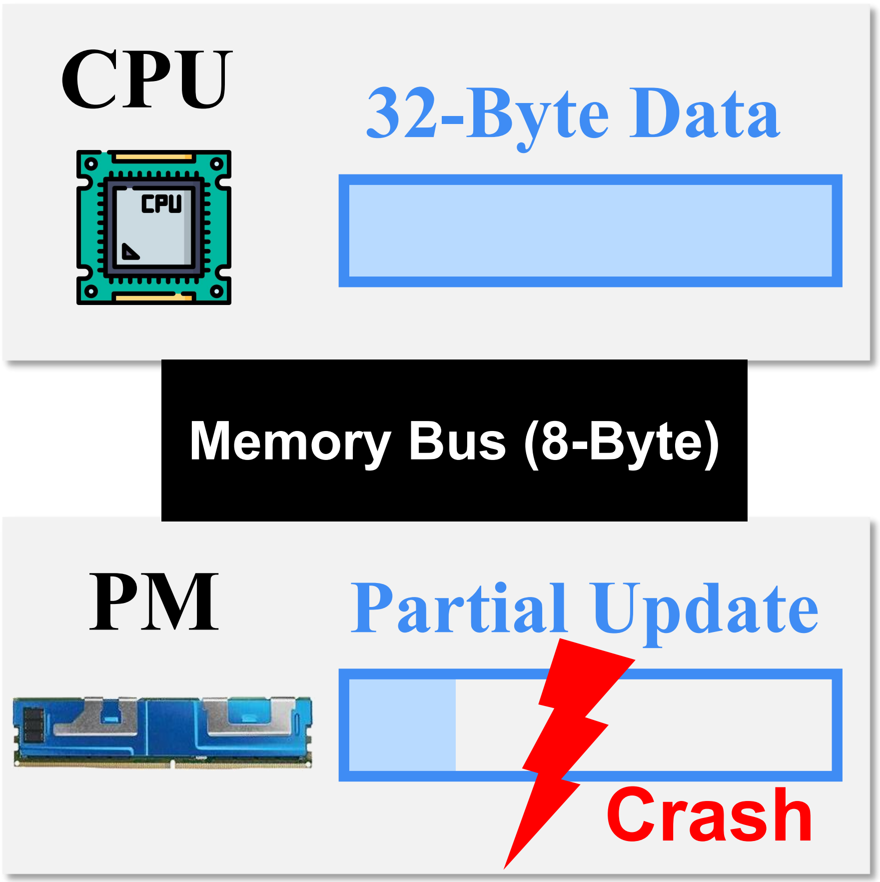
Without Consistency Guarantee



Data Inconsistency!

Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee

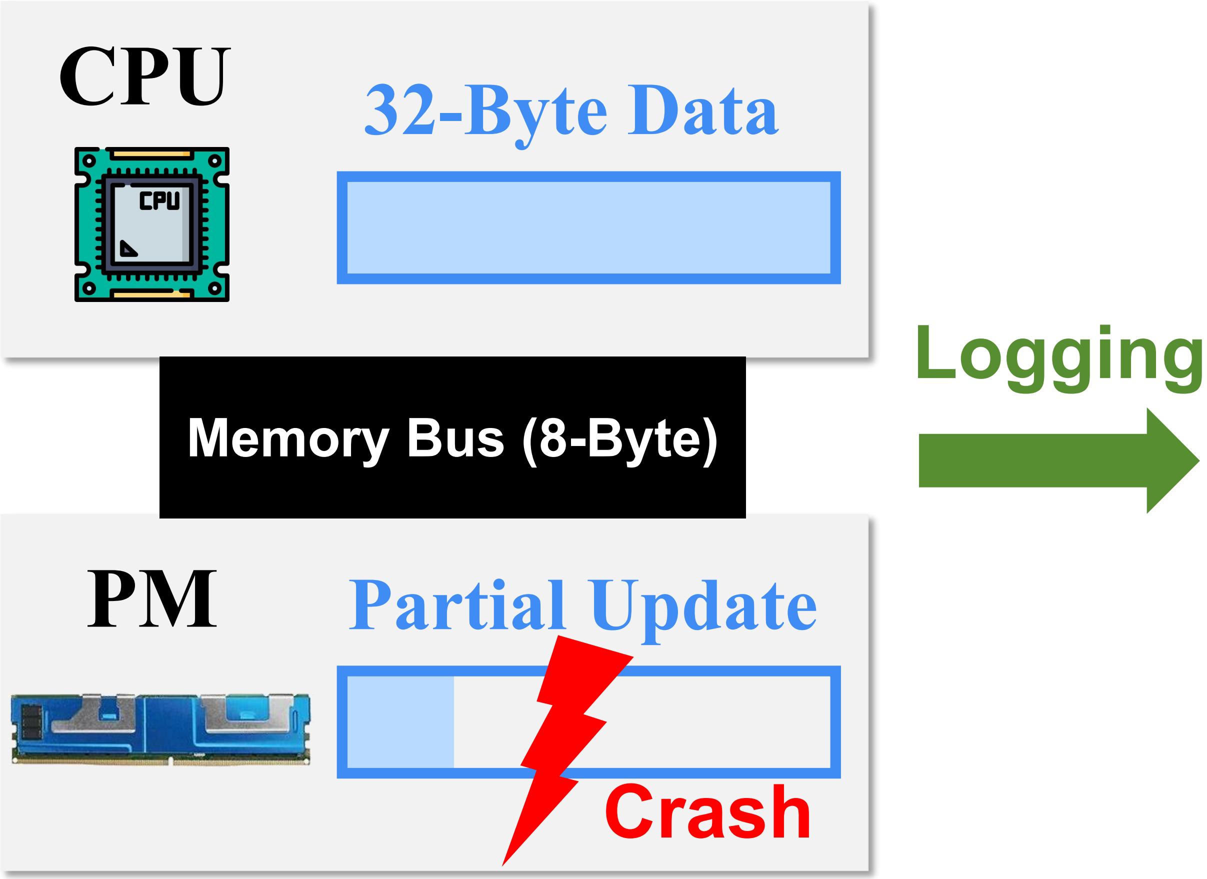


Data Inconsistency!

With Consistency Guarantee

Challenge 2: Ensure Crash Consistency

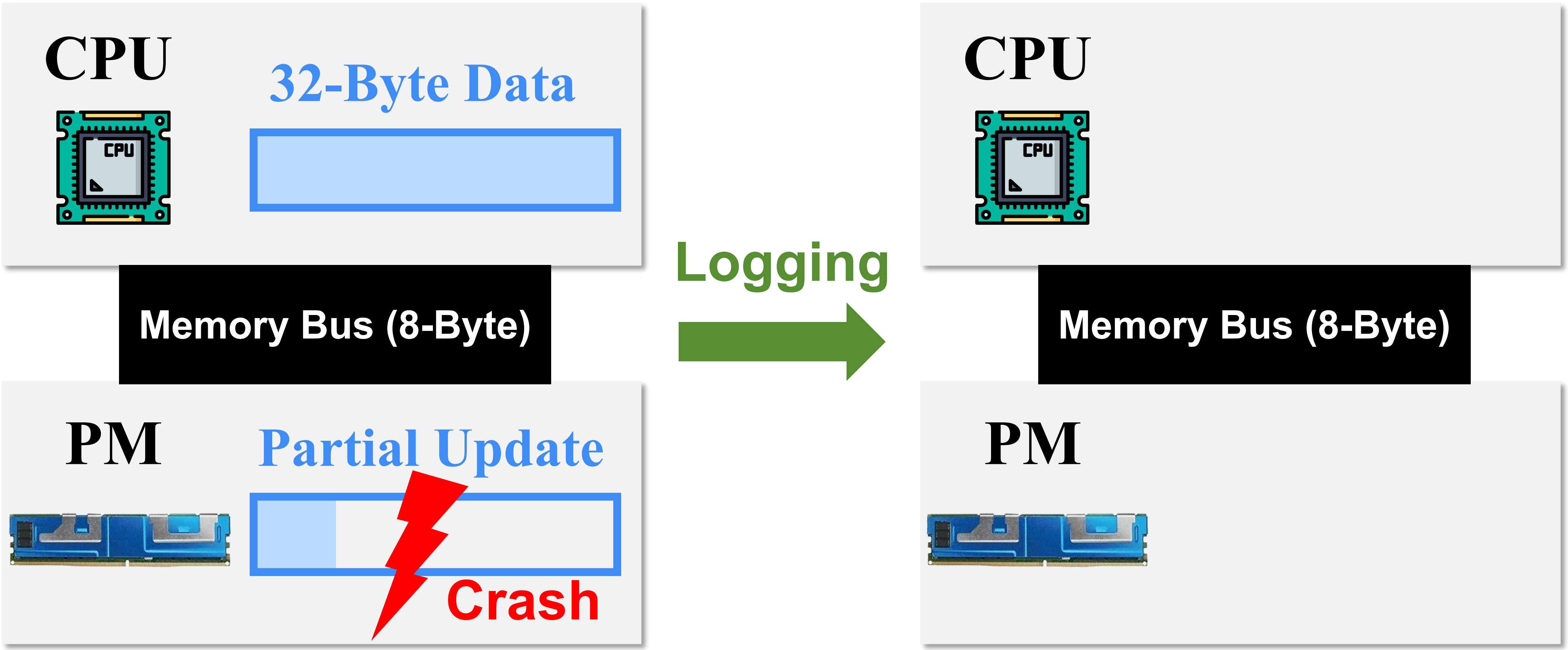
Without Consistency Guarantee **With Consistency Guarantee**



Data Inconsistency!

Challenge 2: Ensure Crash Consistency

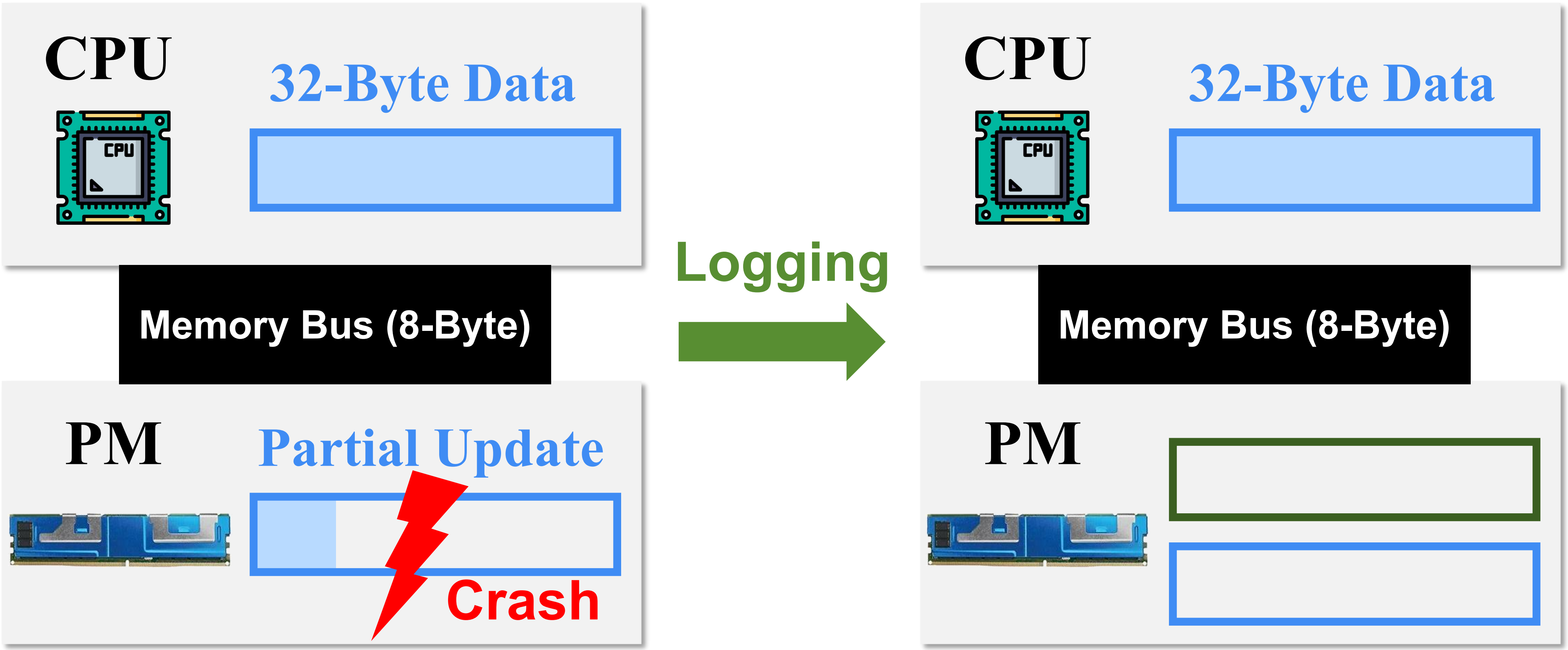
Without Consistency Guarantee | With Consistency Guarantee



Data Inconsistency!

Challenge 2: Ensure Crash Consistency

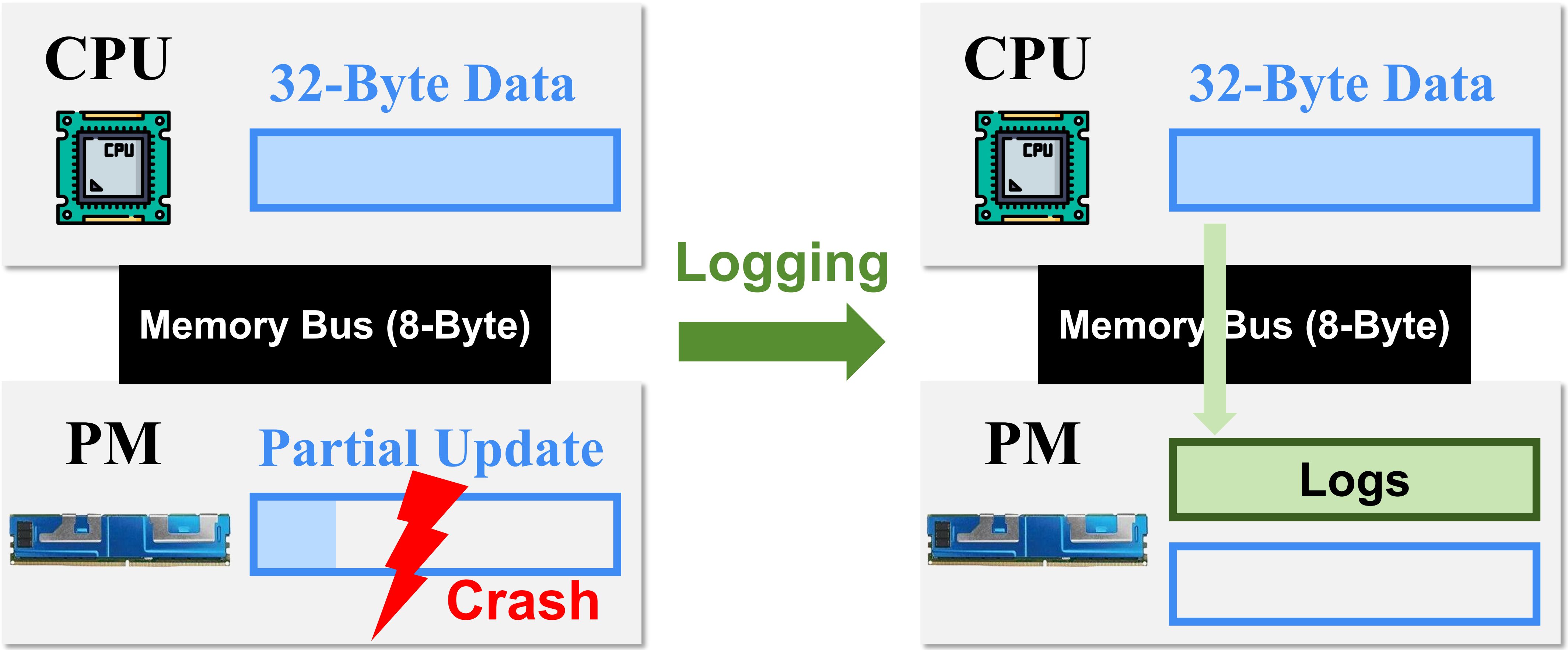
Without Consistency Guarantee **With Consistency Guarantee**



Data Inconsistency!

Challenge 2: Ensure Crash Consistency

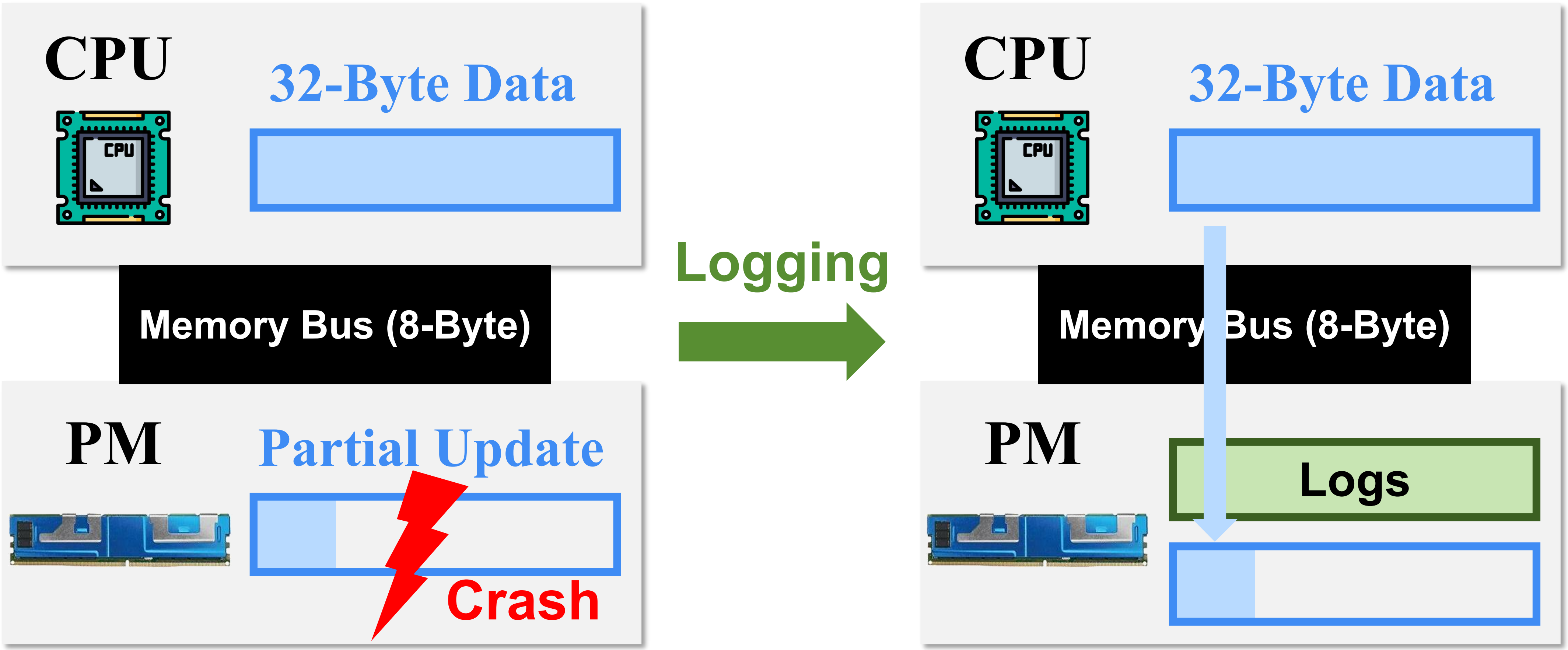
Without Consistency Guarantee | **With Consistency Guarantee**



Data Inconsistency!

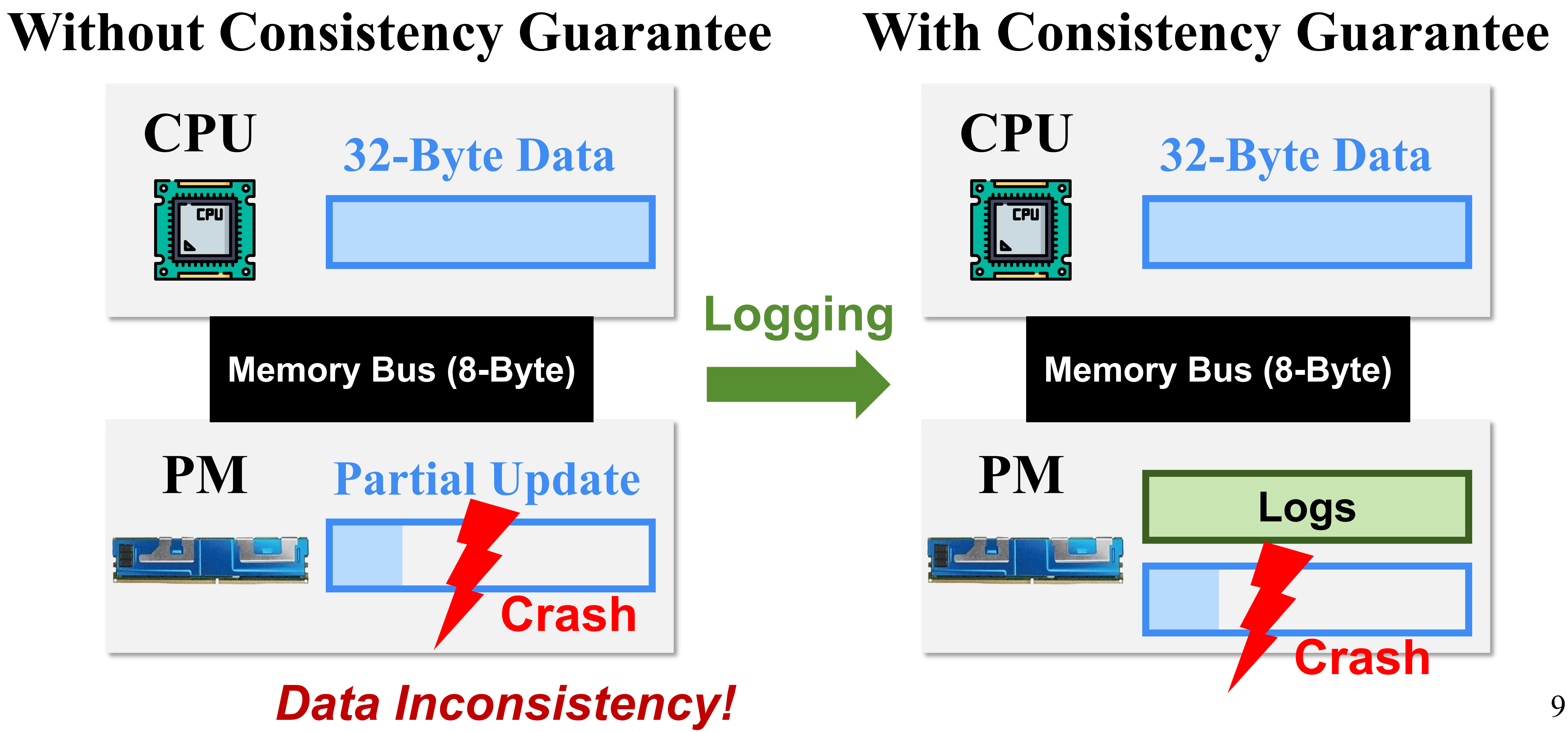
Challenge 2: Ensure Crash Consistency

Without Consistency Guarantee | **With Consistency Guarantee**

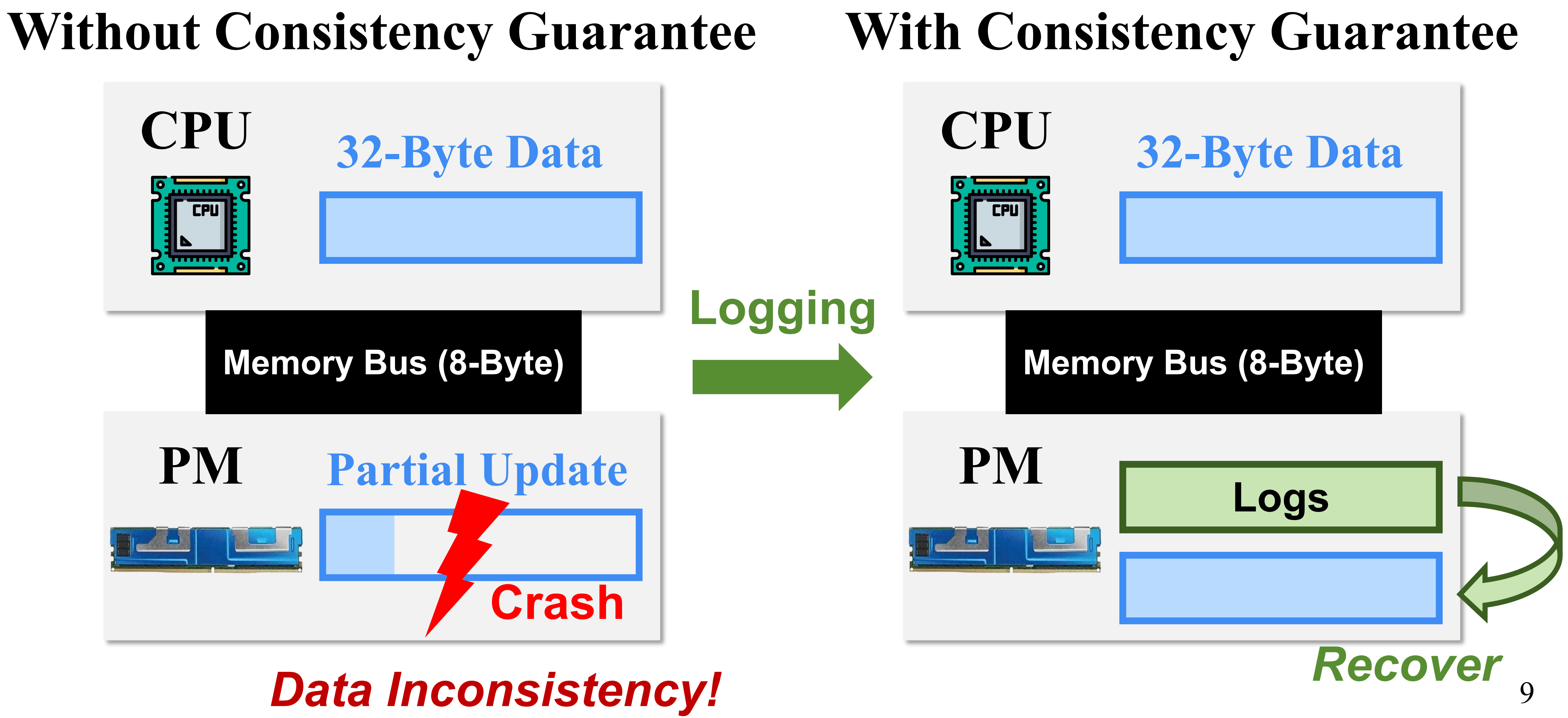


Data Inconsistency!

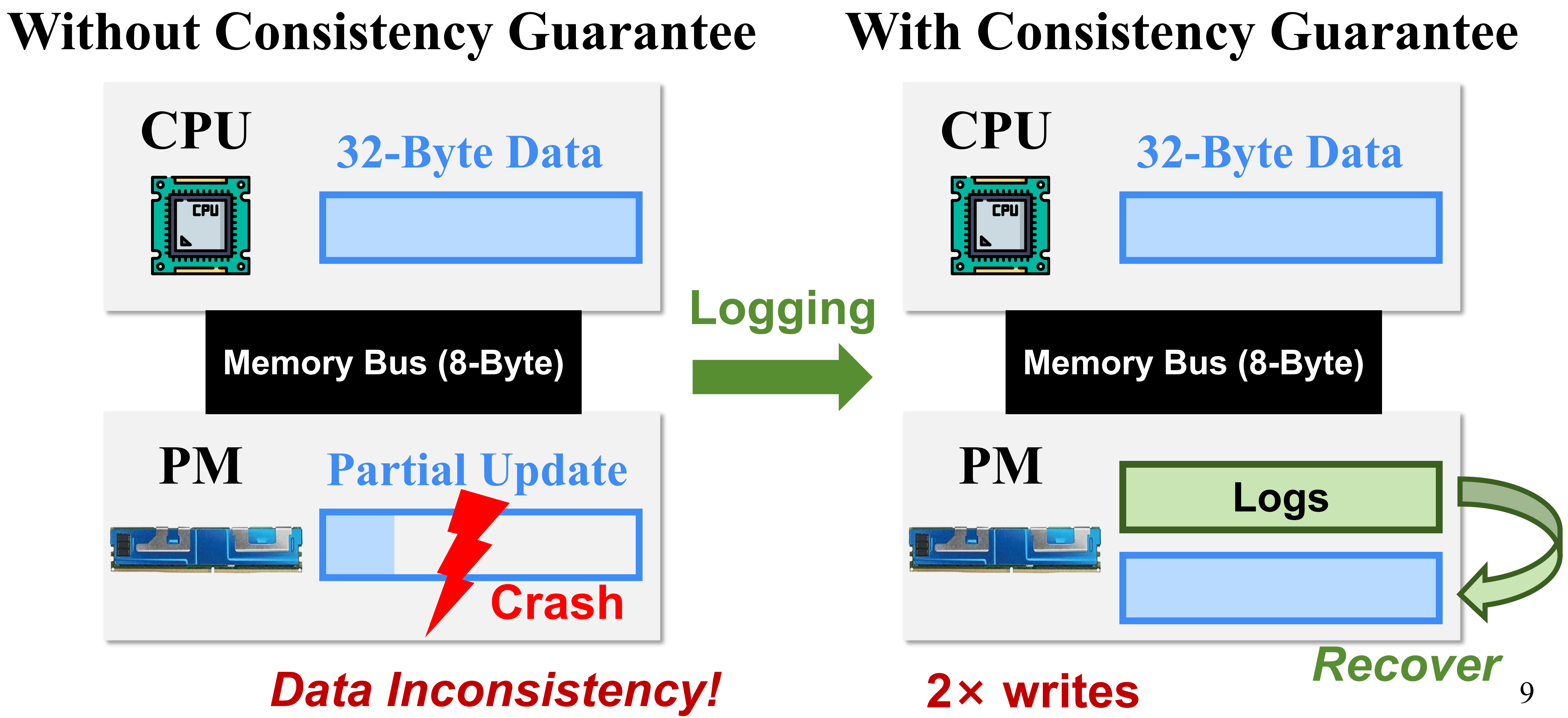
Challenge 2: Ensure Crash Consistency



Challenge 2: Ensure Crash Consistency



Challenge 2: Ensure Crash Consistency



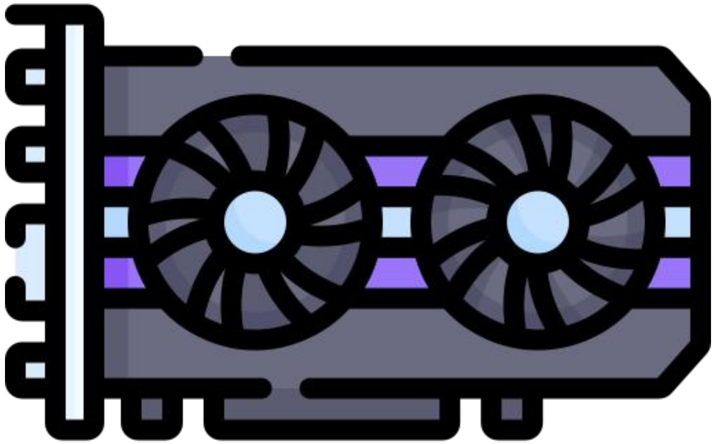
Challenge 2: Ensure Crash Consistency

Crash consistency guarantee introduces **High Overhead**

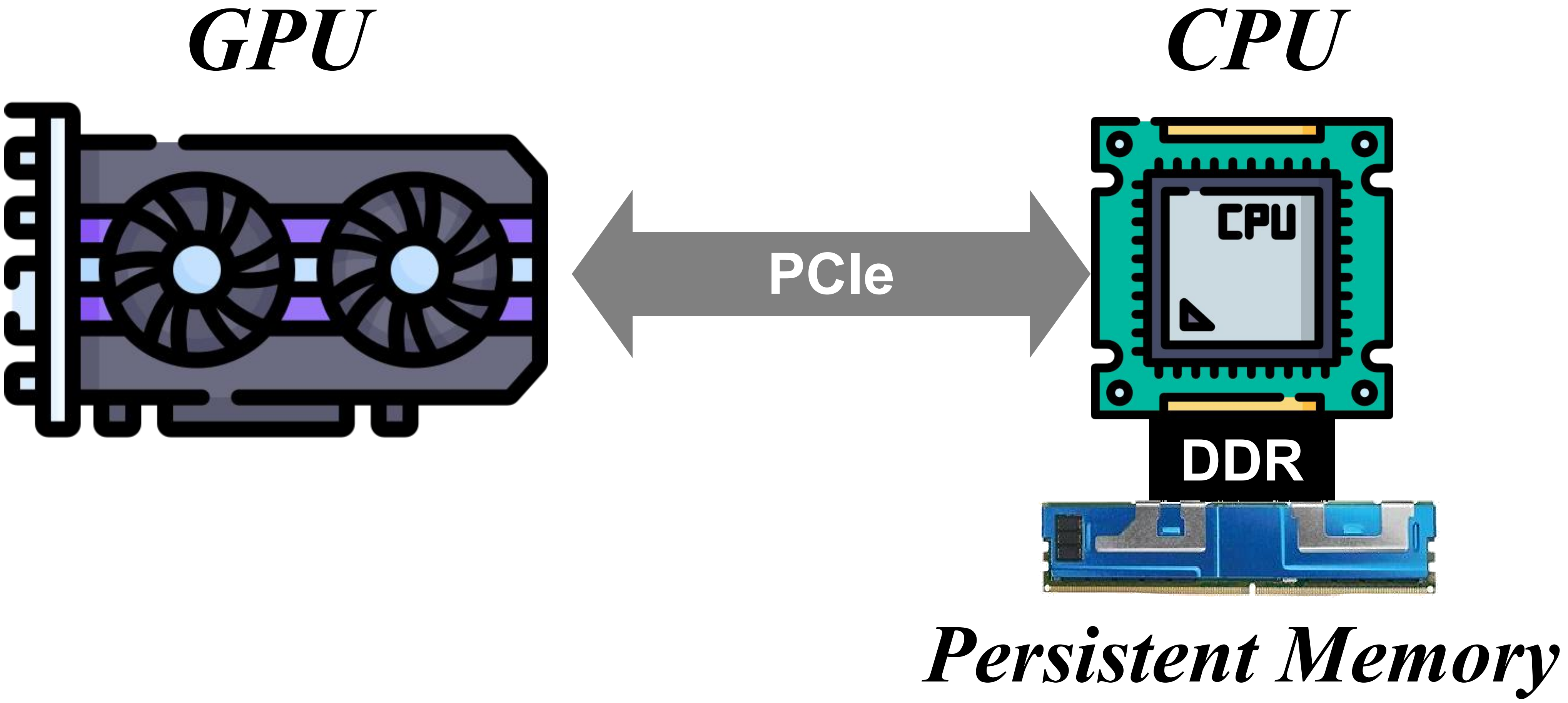
Challenge 3: Huge Bandwidth Gap

Challenge 3: Huge Bandwidth Gap

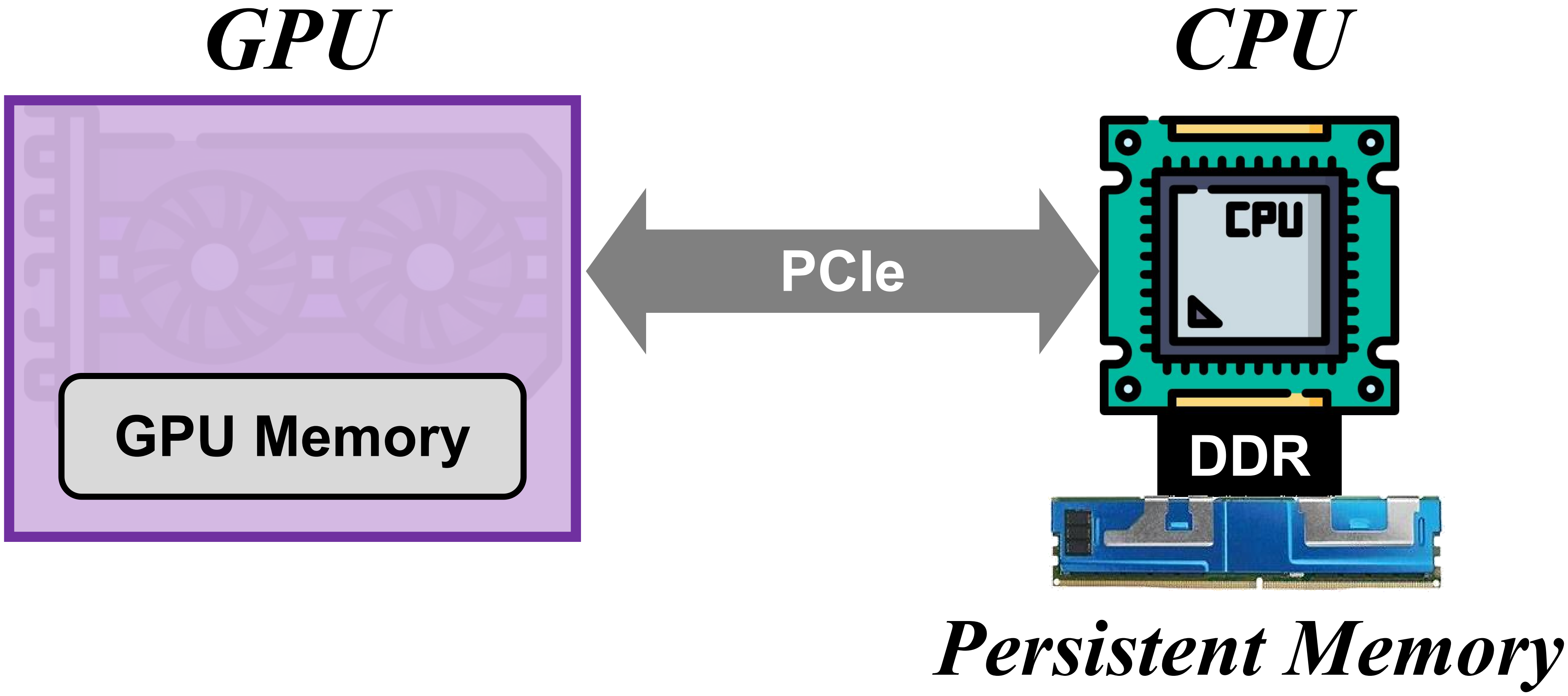
GPU



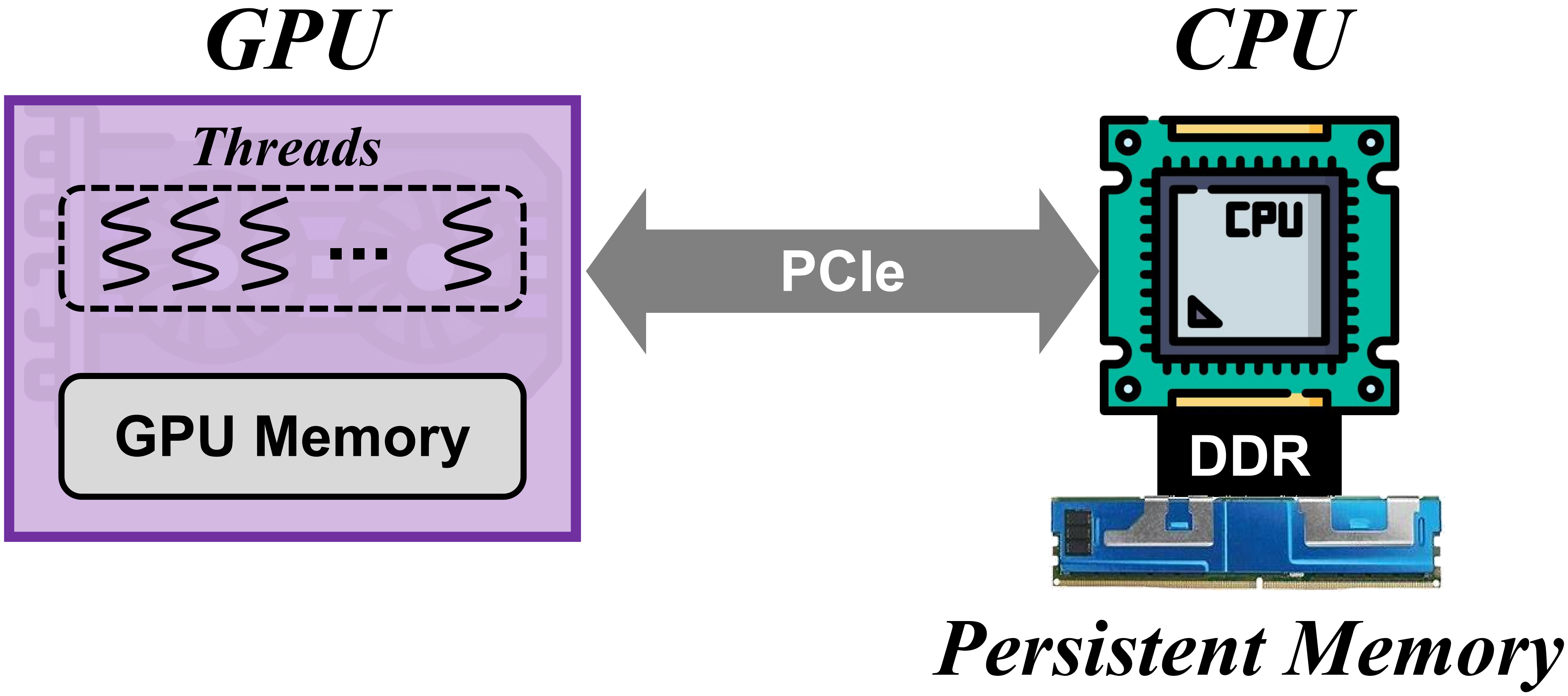
Challenge 3: Huge Bandwidth Gap



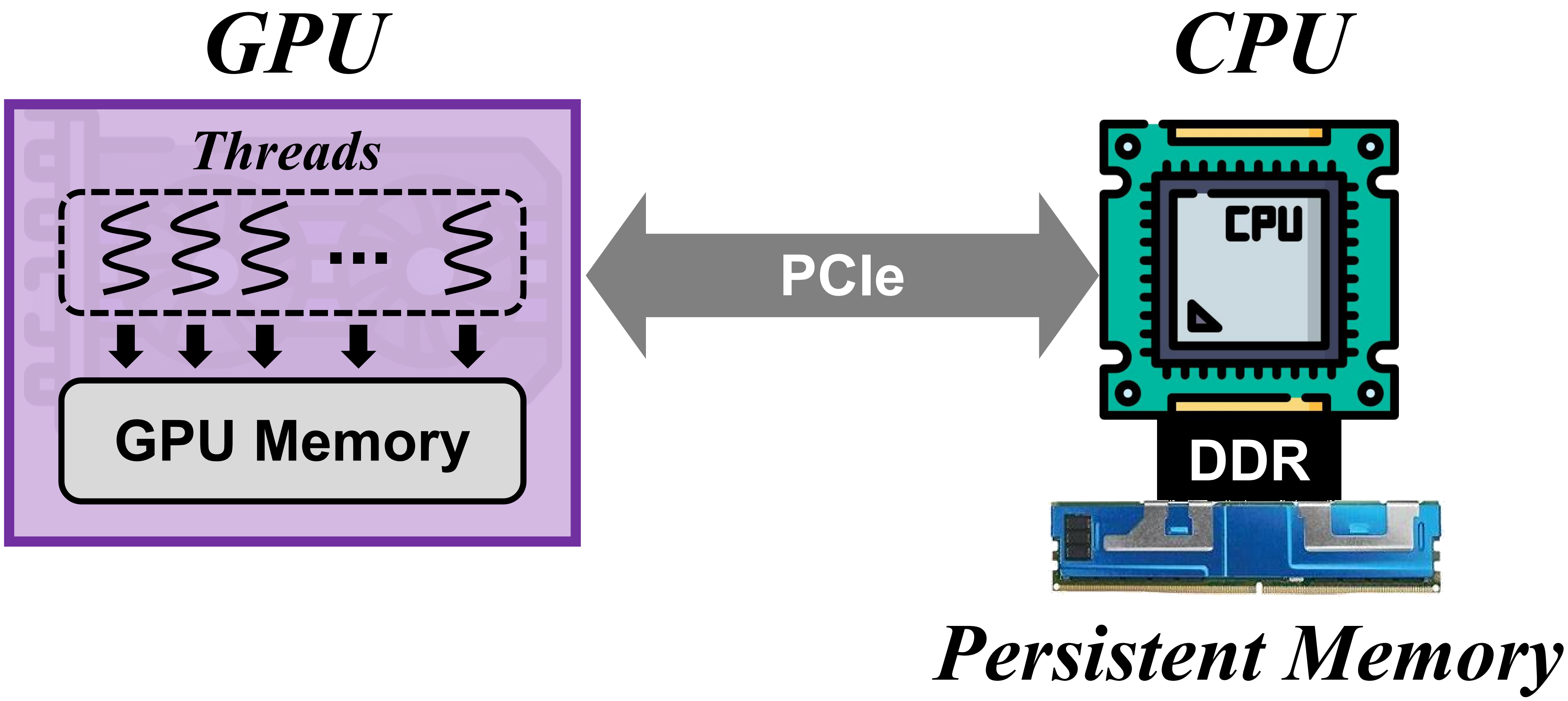
Challenge 3: Huge Bandwidth Gap



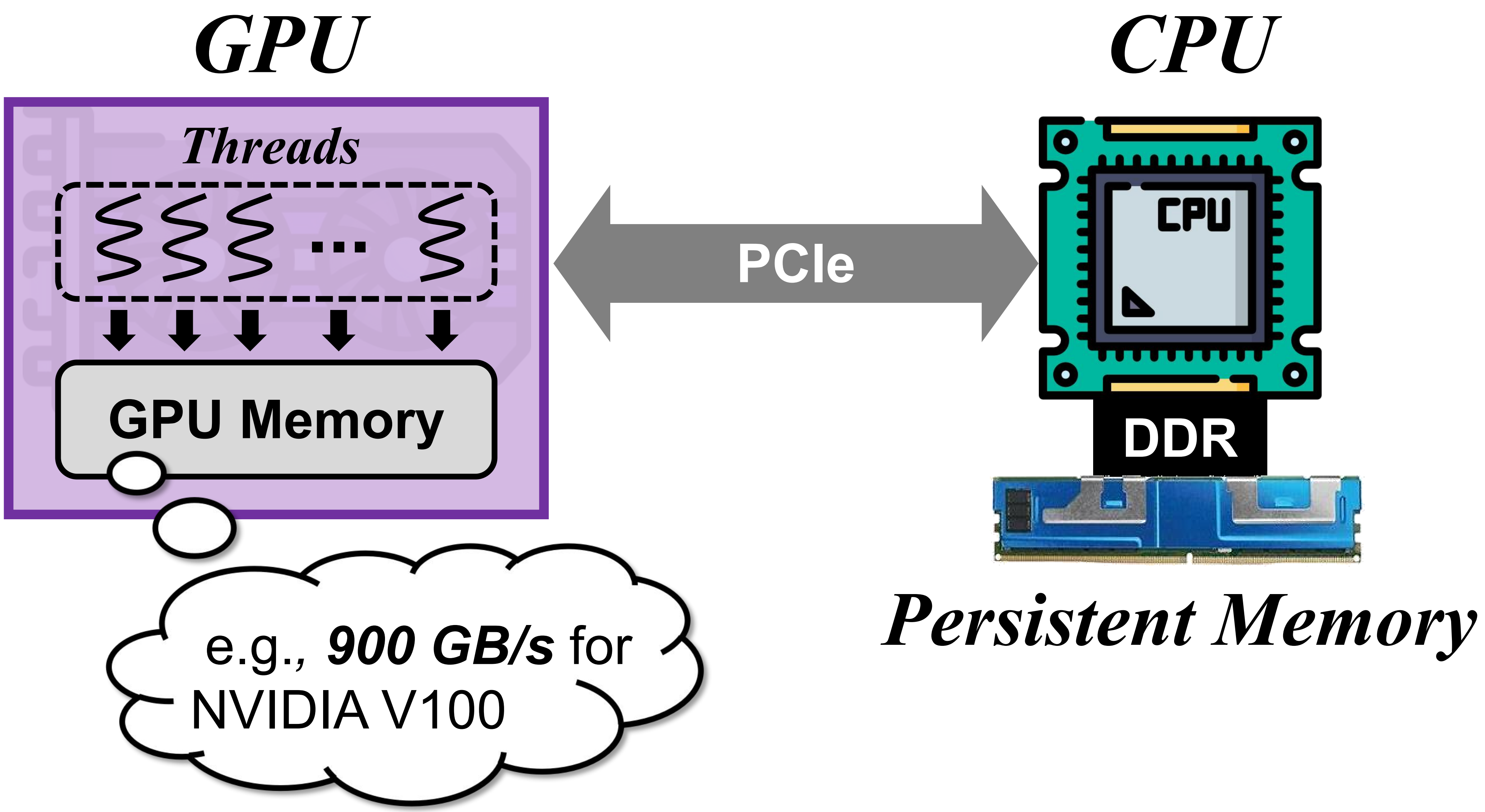
Challenge 3: Huge Bandwidth Gap



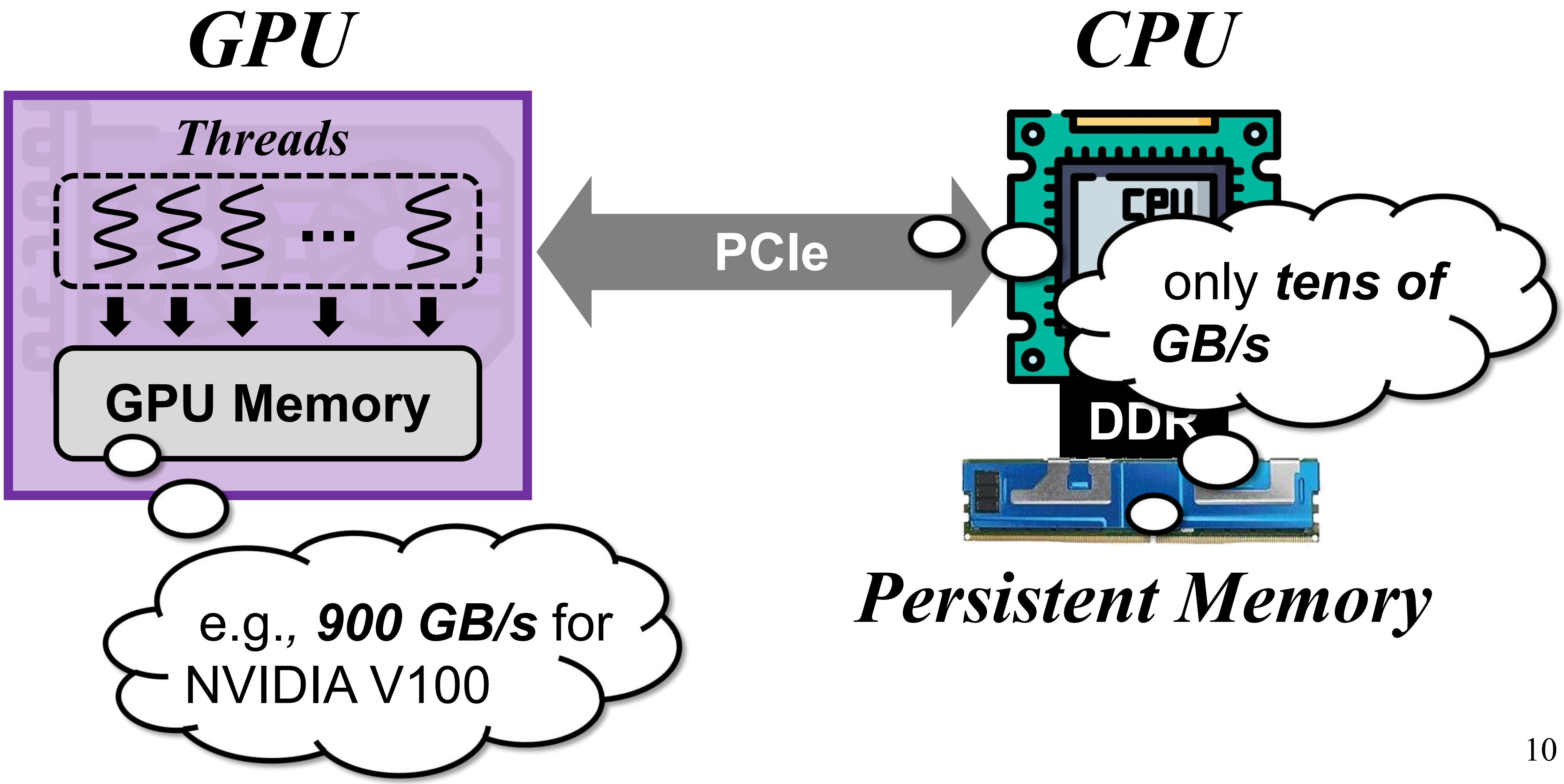
Challenge 3: Huge Bandwidth Gap



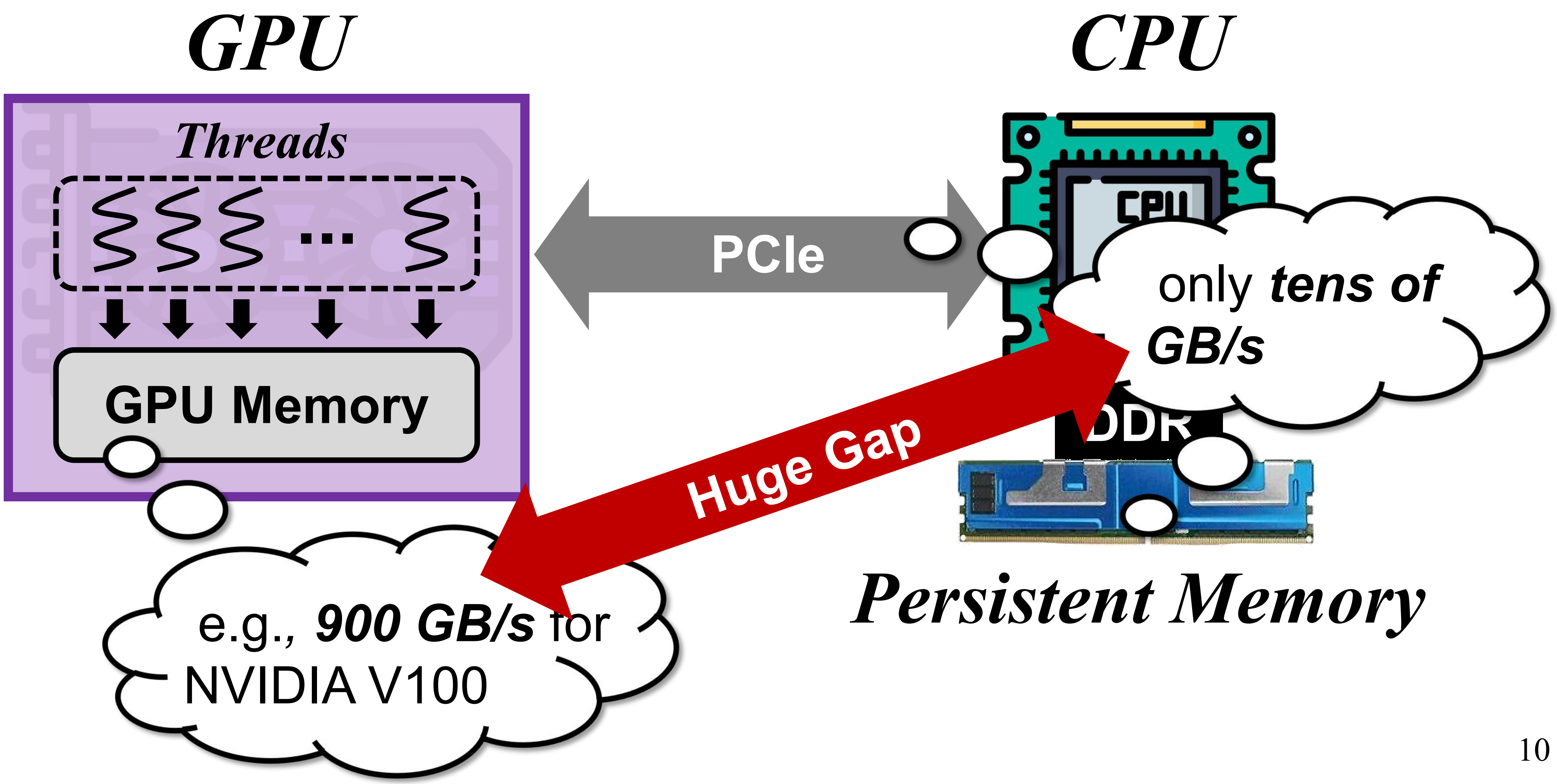
Challenge 3: Huge Bandwidth Gap



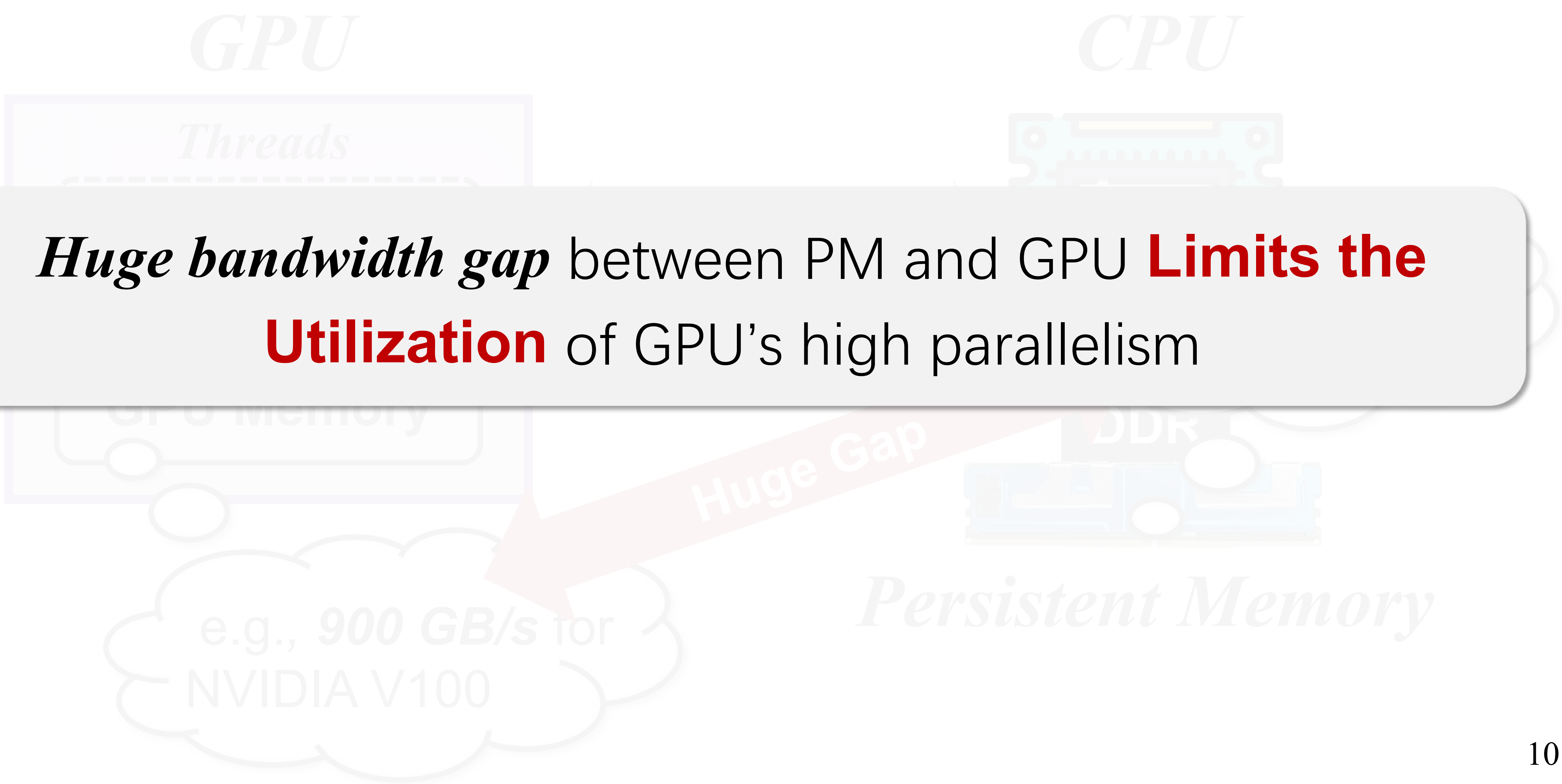
Challenge 3: Huge Bandwidth Gap



Challenge 3: Huge Bandwidth Gap



Challenge 3: **Huge Bandwidth Gap**



Huge bandwidth gap between PM and GPU **Limits the Utilization** of GPU's high parallelism

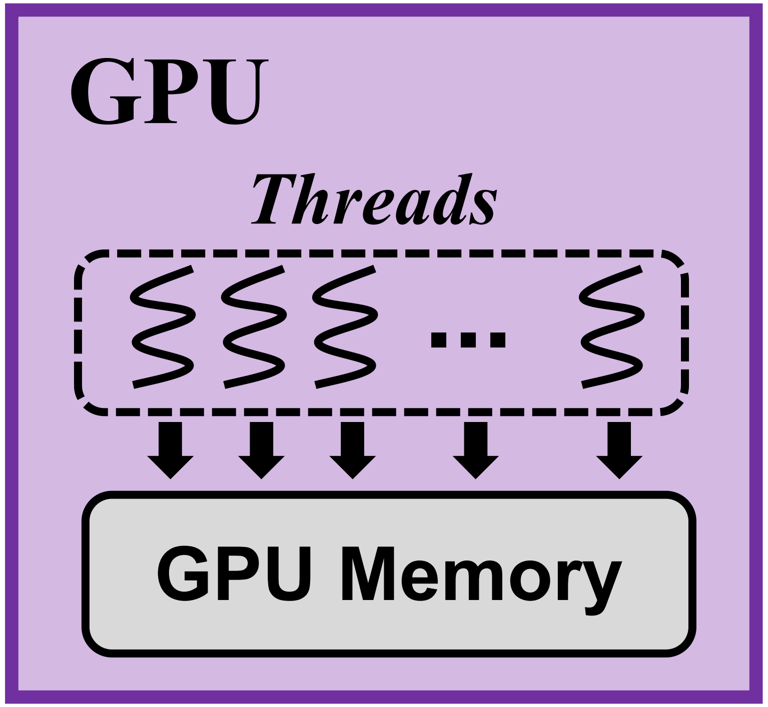
Our Solution: GPHash

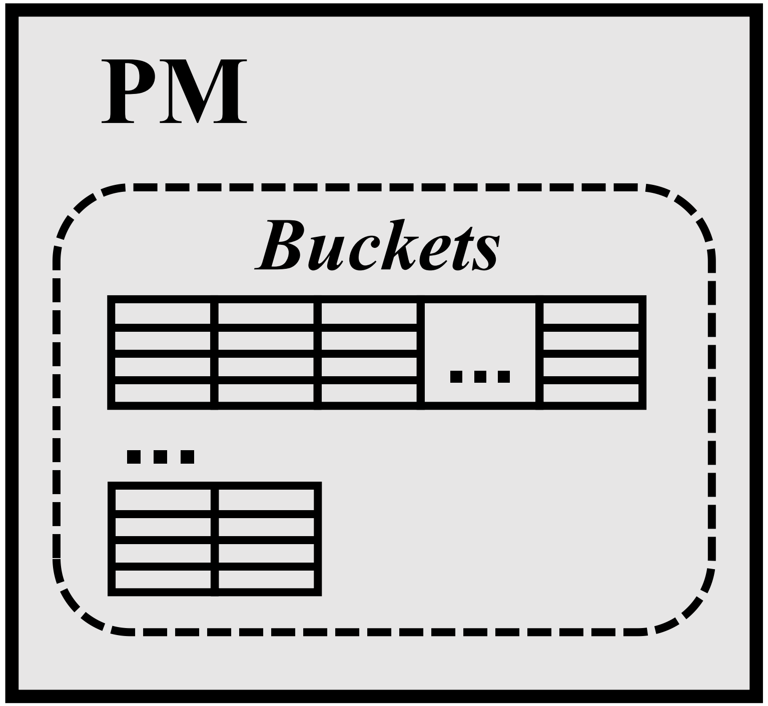
Our Solution: GPHash

- **GPHash:** ***An Efficient Hash Index for GPM Systems***

Our Solution: GPHash

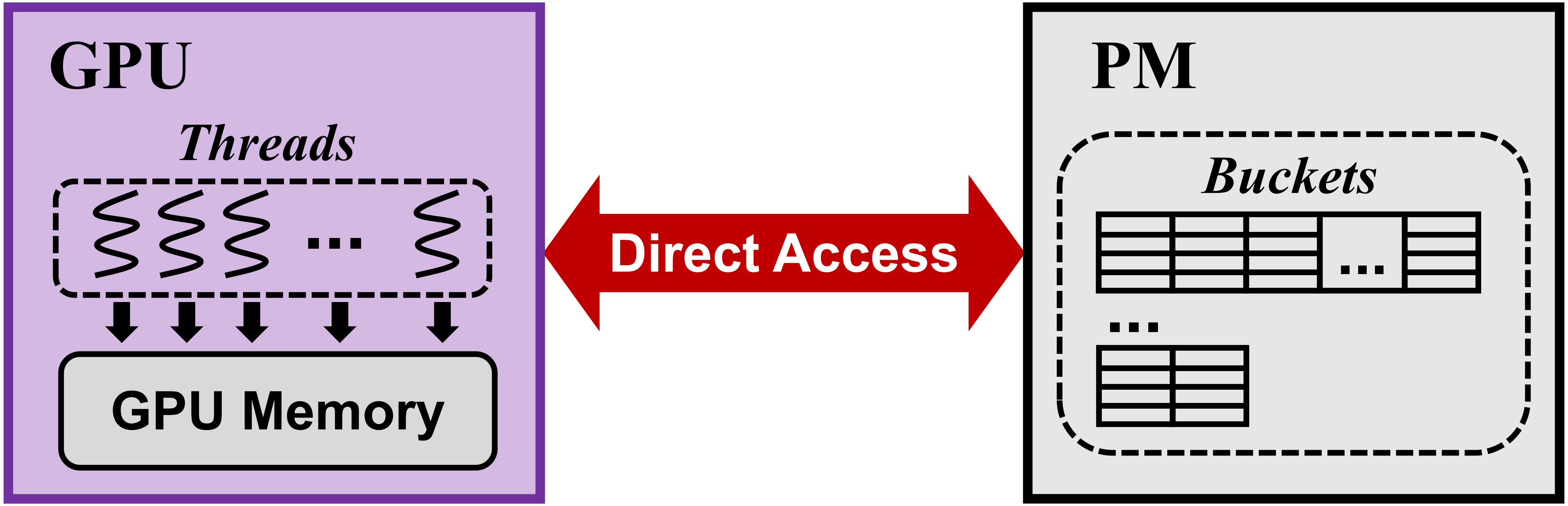
- **GPHash:** ***An Efficient Hash Index for GPM Systems***





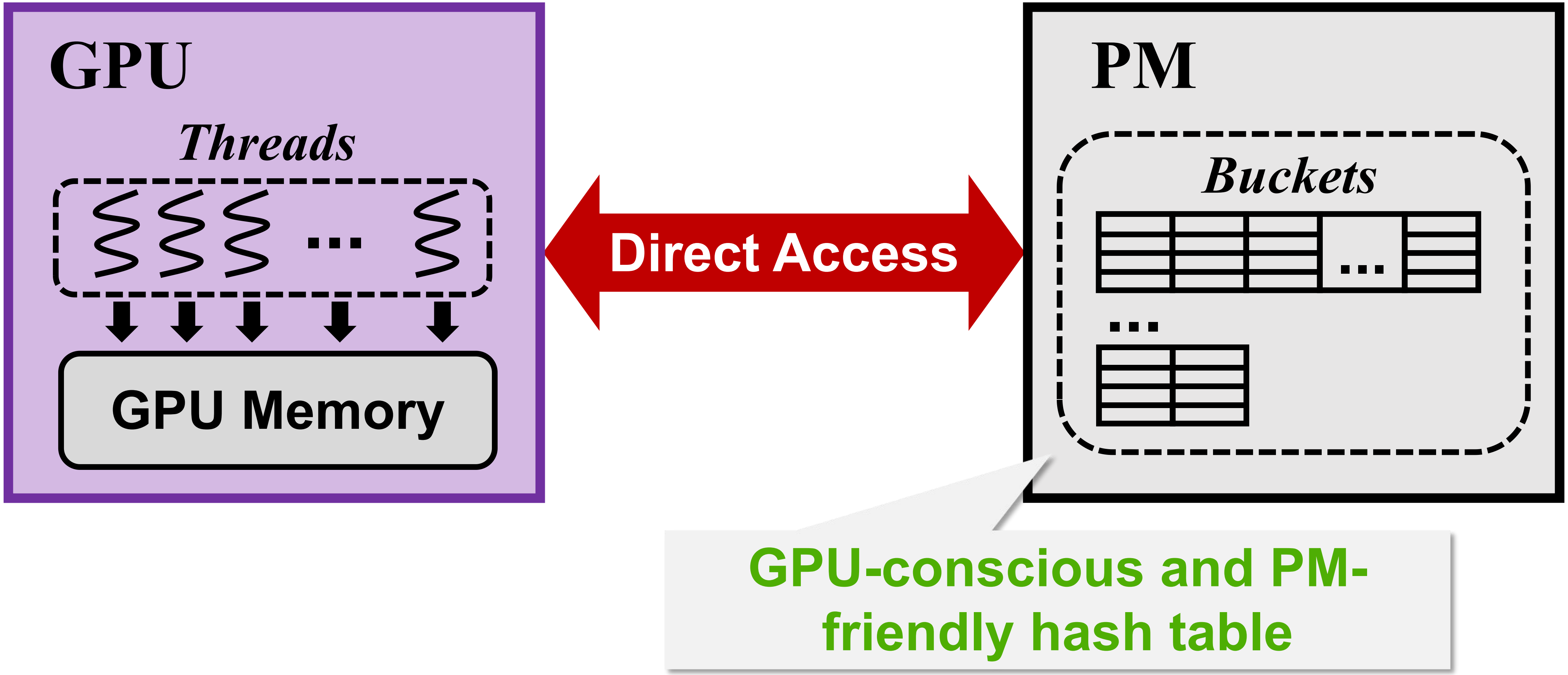
Our Solution: GPHash

➢ **GPHash:** ***An Efficient Hash Index for GPM Systems***



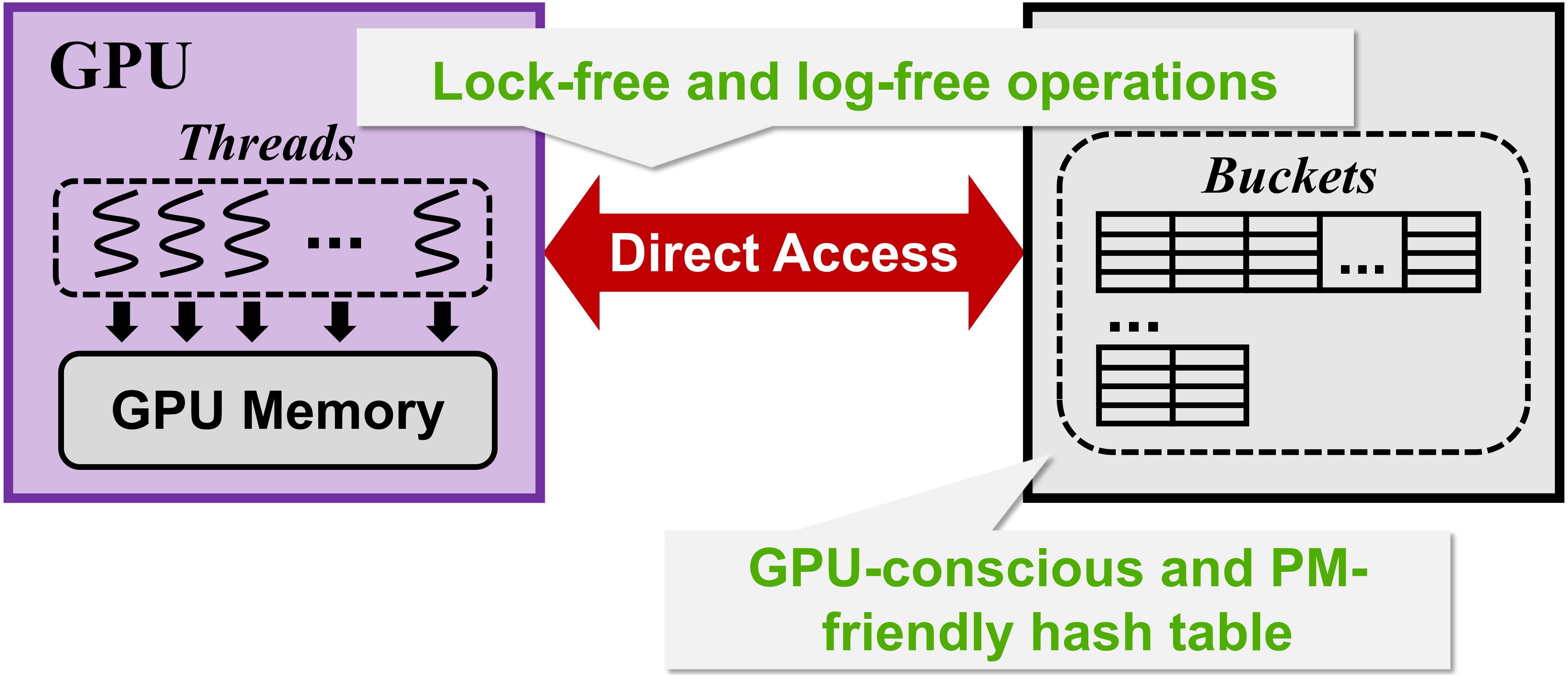
Our Solution: GPHash

- **GPHash:** ***An Efficient Hash Index for GPM Systems***



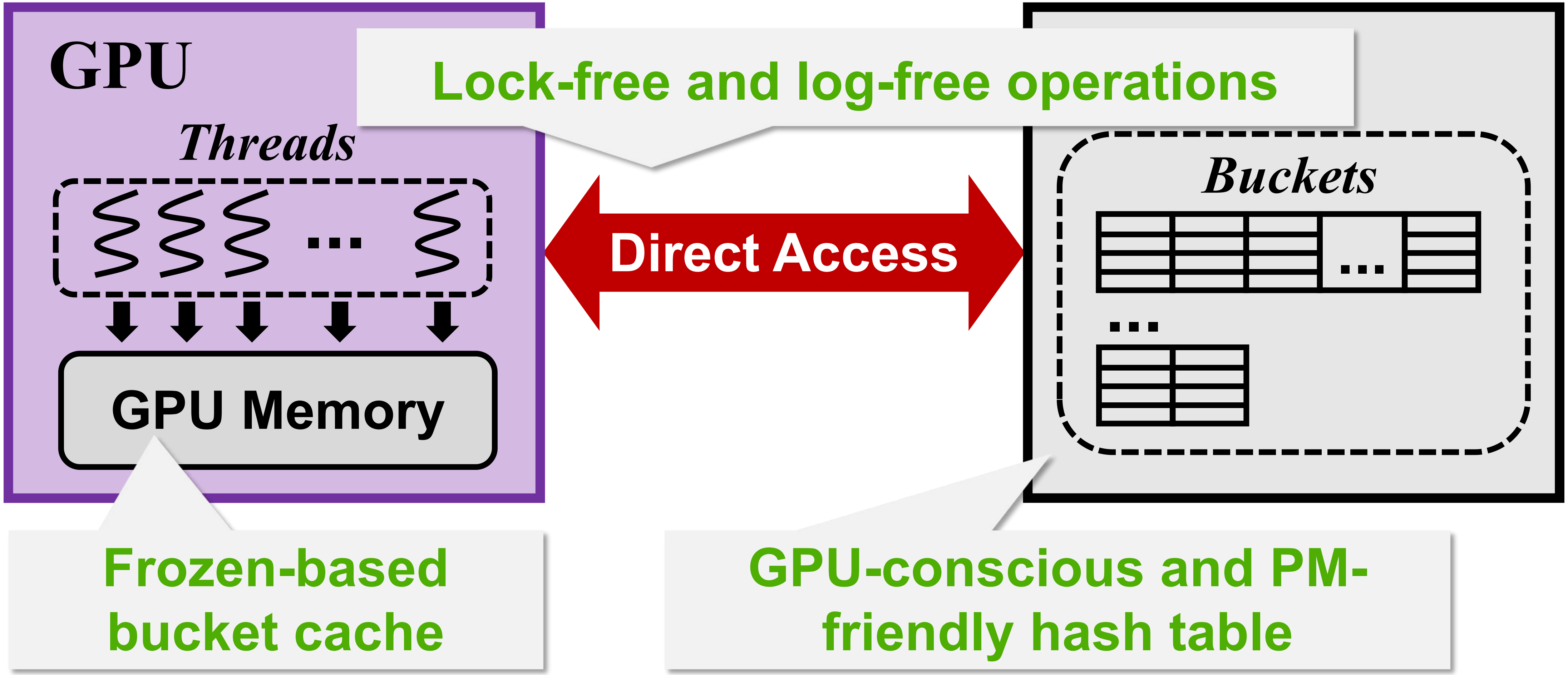
Our Solution: GPHash

➢ **GPHash:** ***An Efficient Hash Index for GPM Systems***



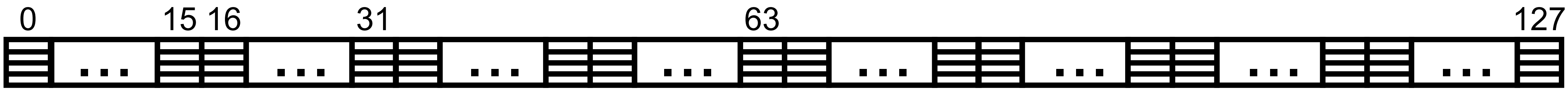
Our Solution: GPHash

➢ **GPHash:** ***An Efficient Hash Index for GPM Systems***

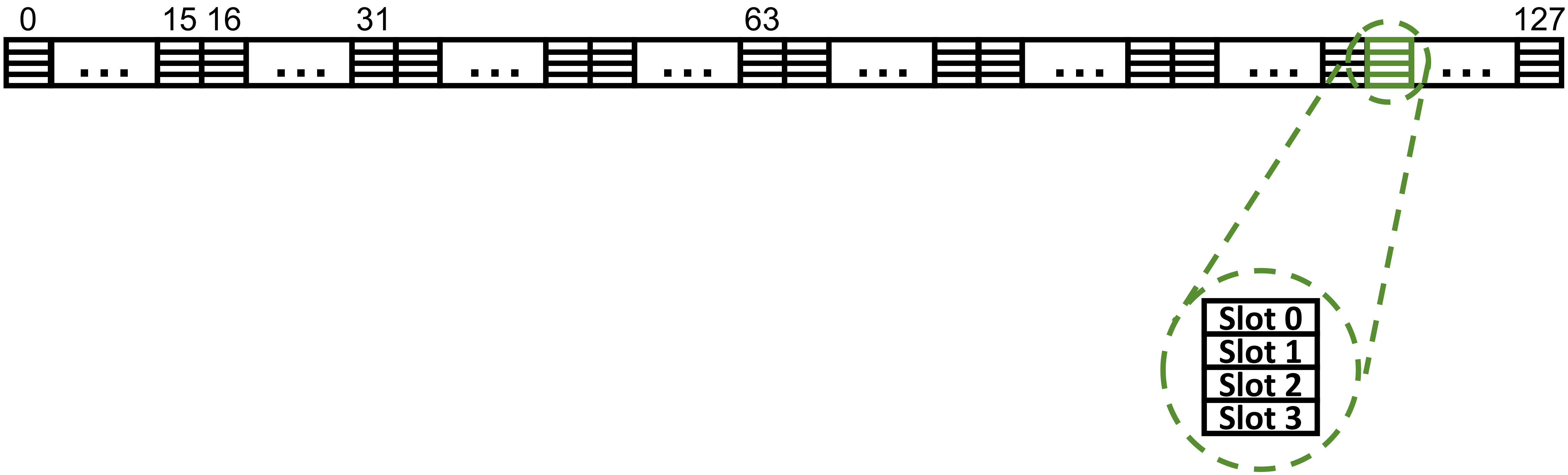


GPU-Conscious and PM-Friendly Hash Table

GPU-Conscious and PM-Friendly Hash Table

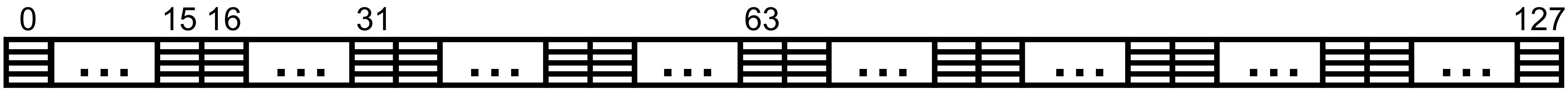


GPU-Conscious and PM-Friendly Hash Table

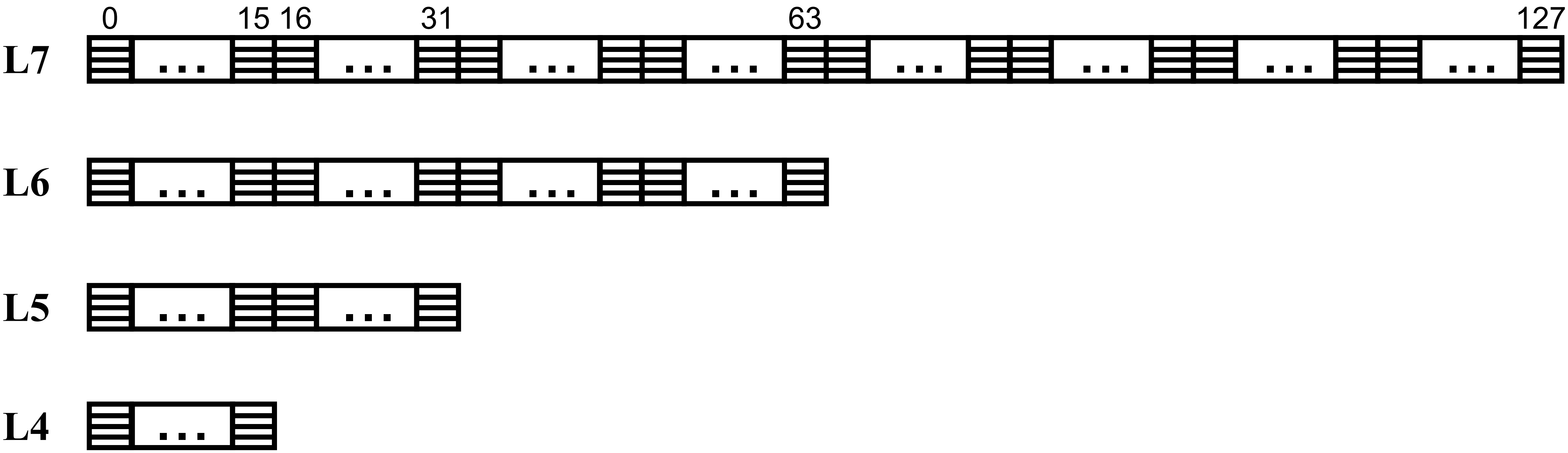


- *Slot Associativity*

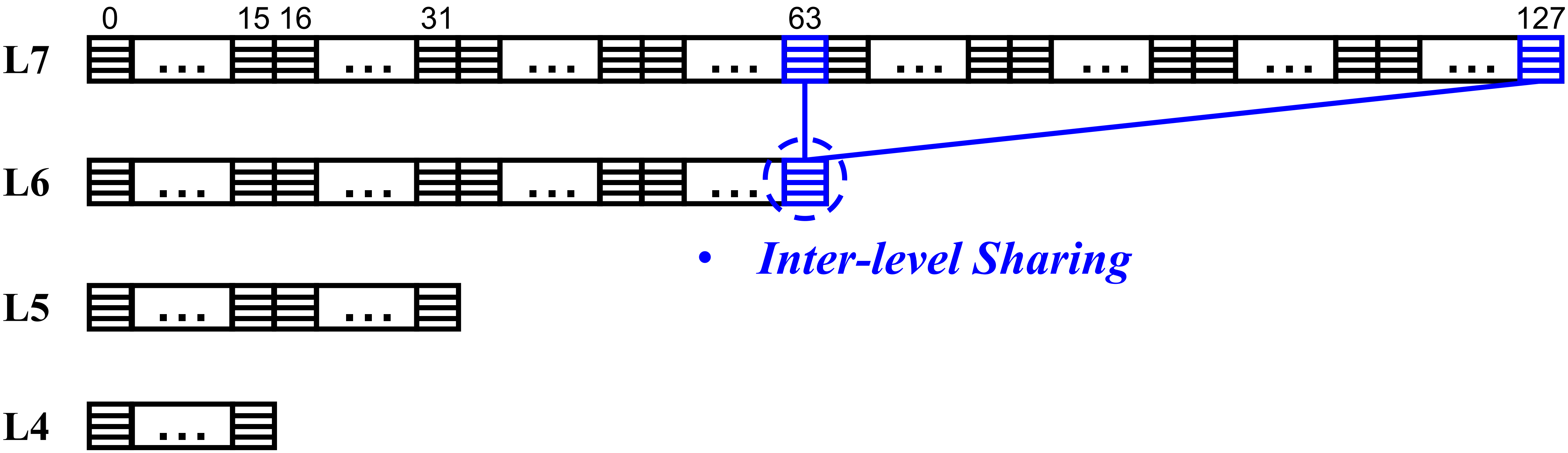
GPU-Conscious and PM-Friendly Hash Table



GPU-Conscious and PM-Friendly Hash Table

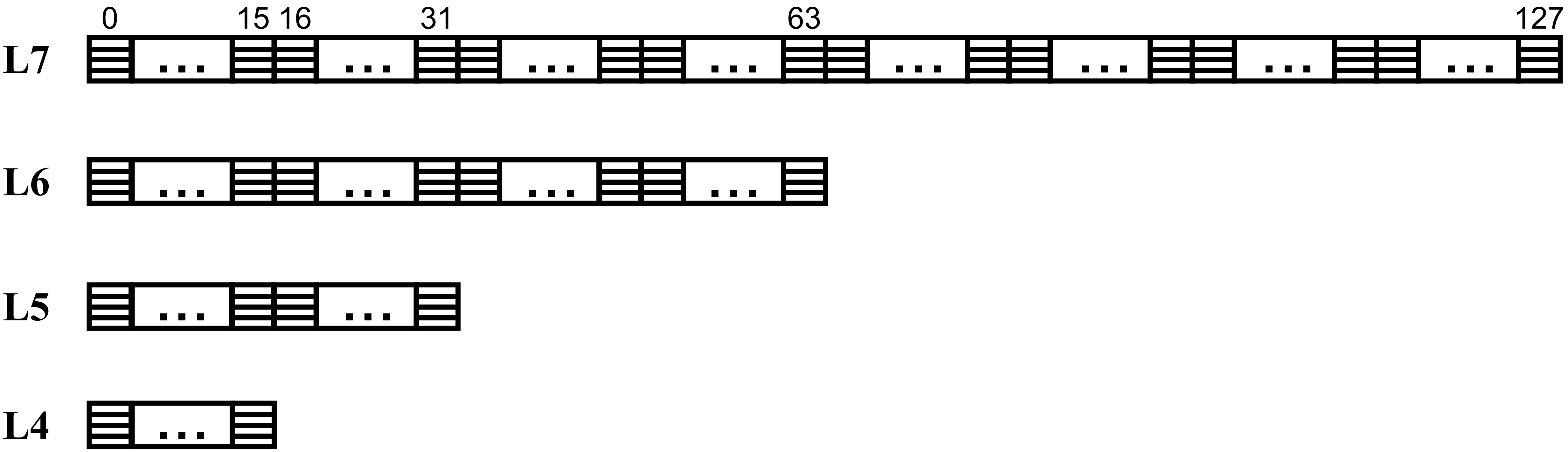


GPU-Conscious and PM-Friendly Hash Table

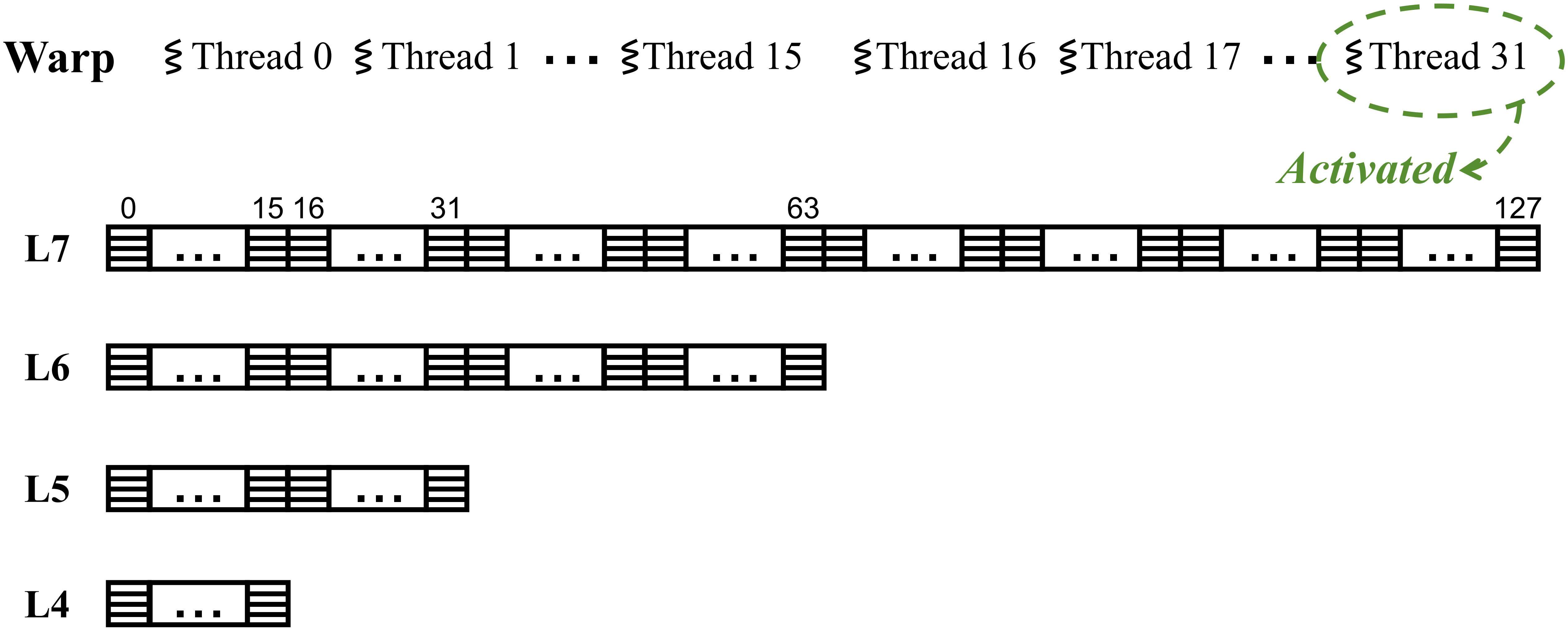


- ***Inter-level Sharing***

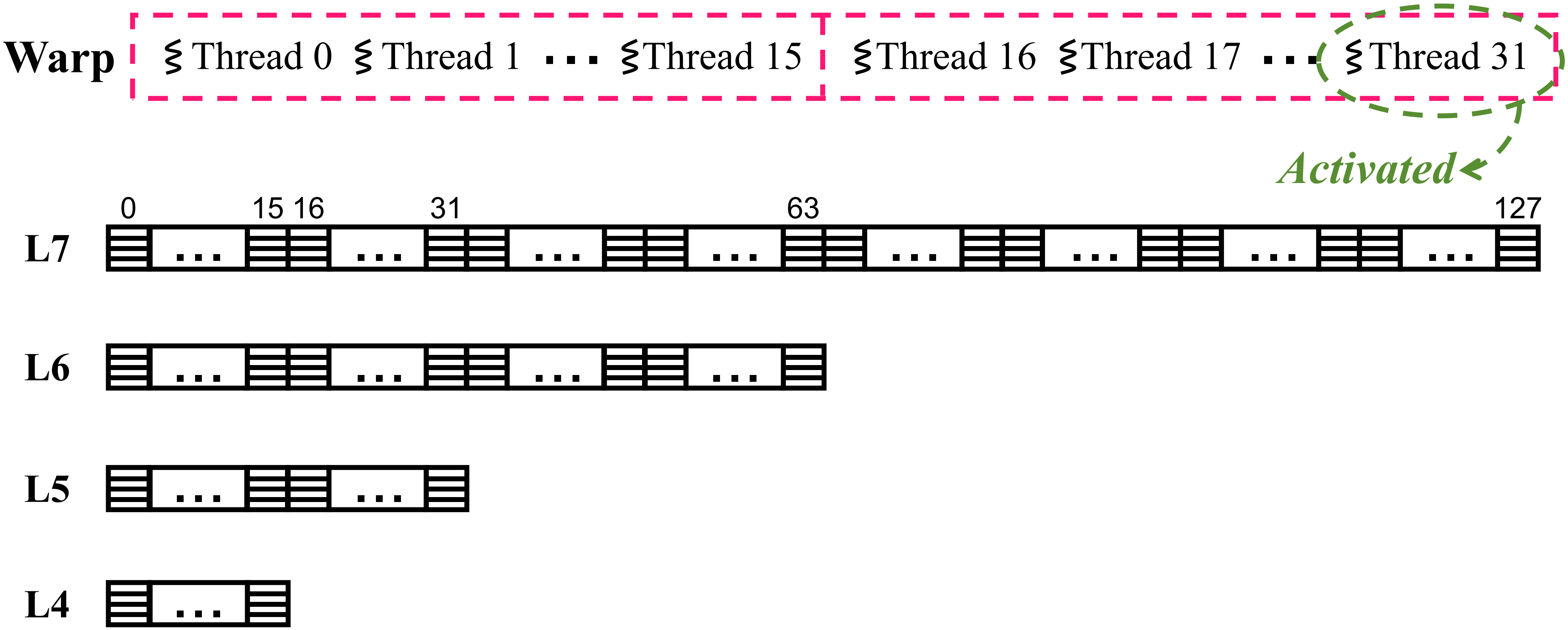
GPU-Conscious and PM-Friendly Hash Table



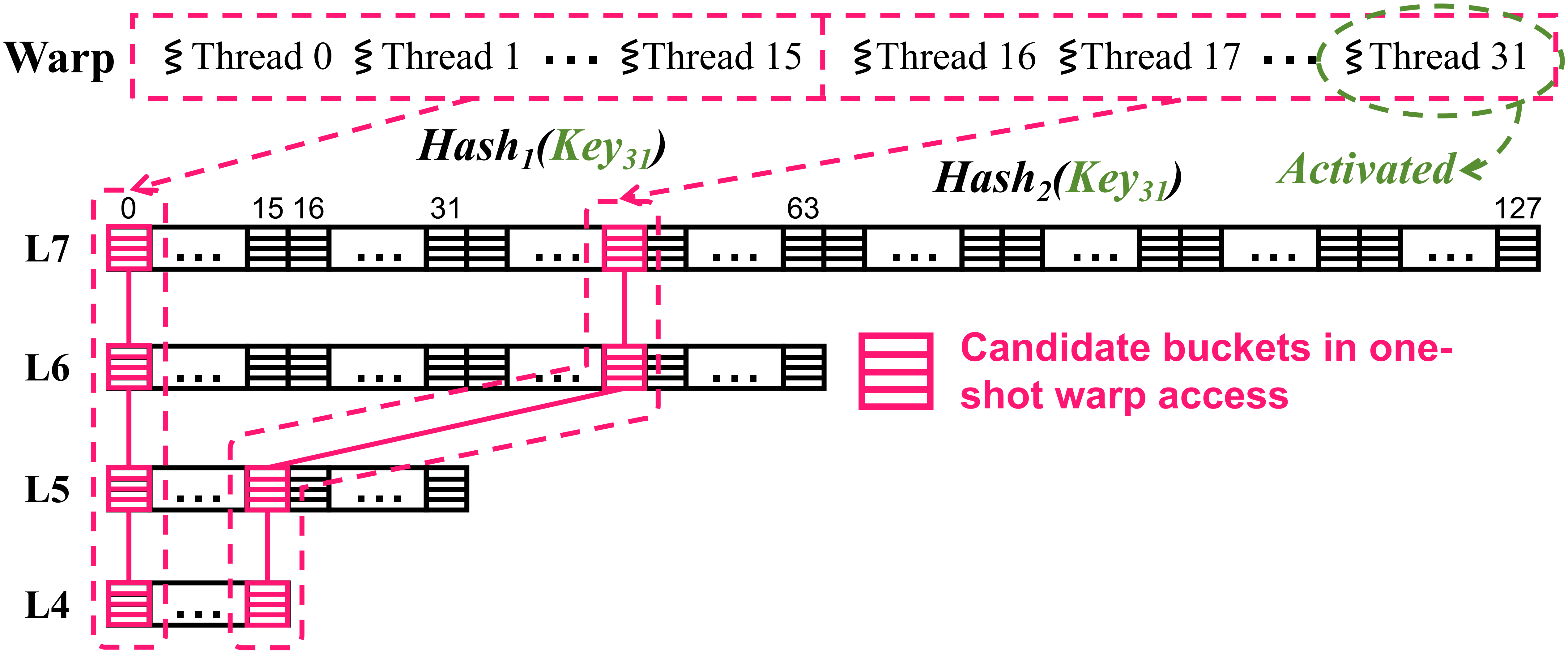
GPU-Conscious and PM-Friendly Hash Table



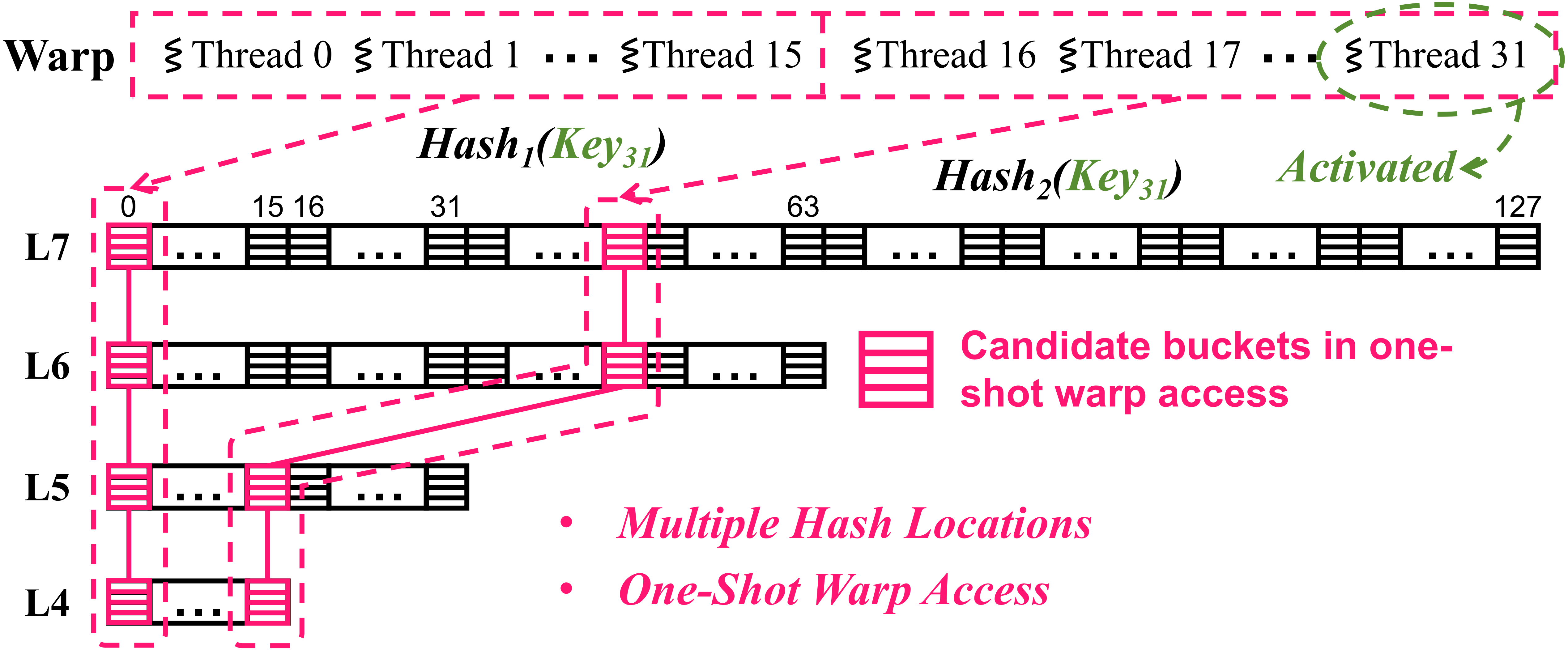
GPU-Conscious and PM-Friendly Hash Table



GPU-Conscious and PM-Friendly Hash Table

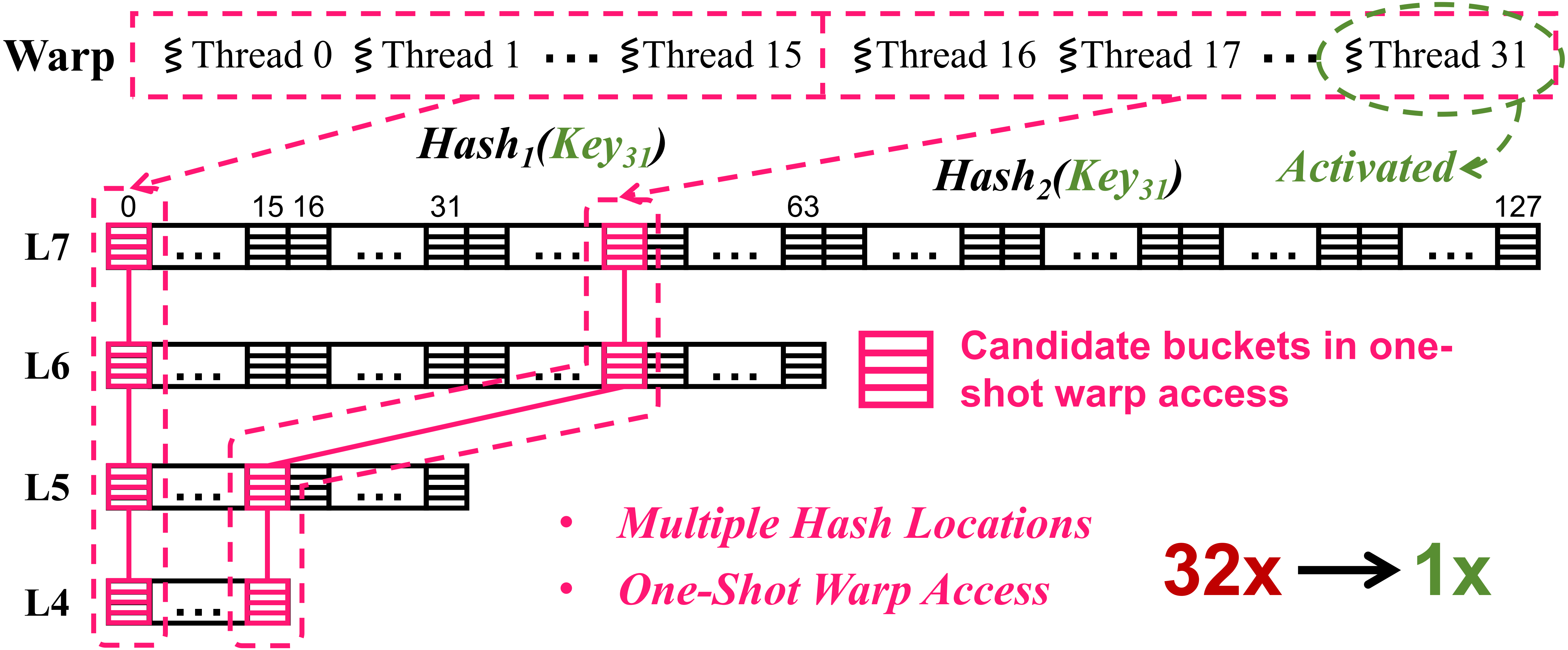


GPU-Conscious and PM-Friendly Hash Table



- *Multiple Hash Locations*
- *One-Shot Warp Access*

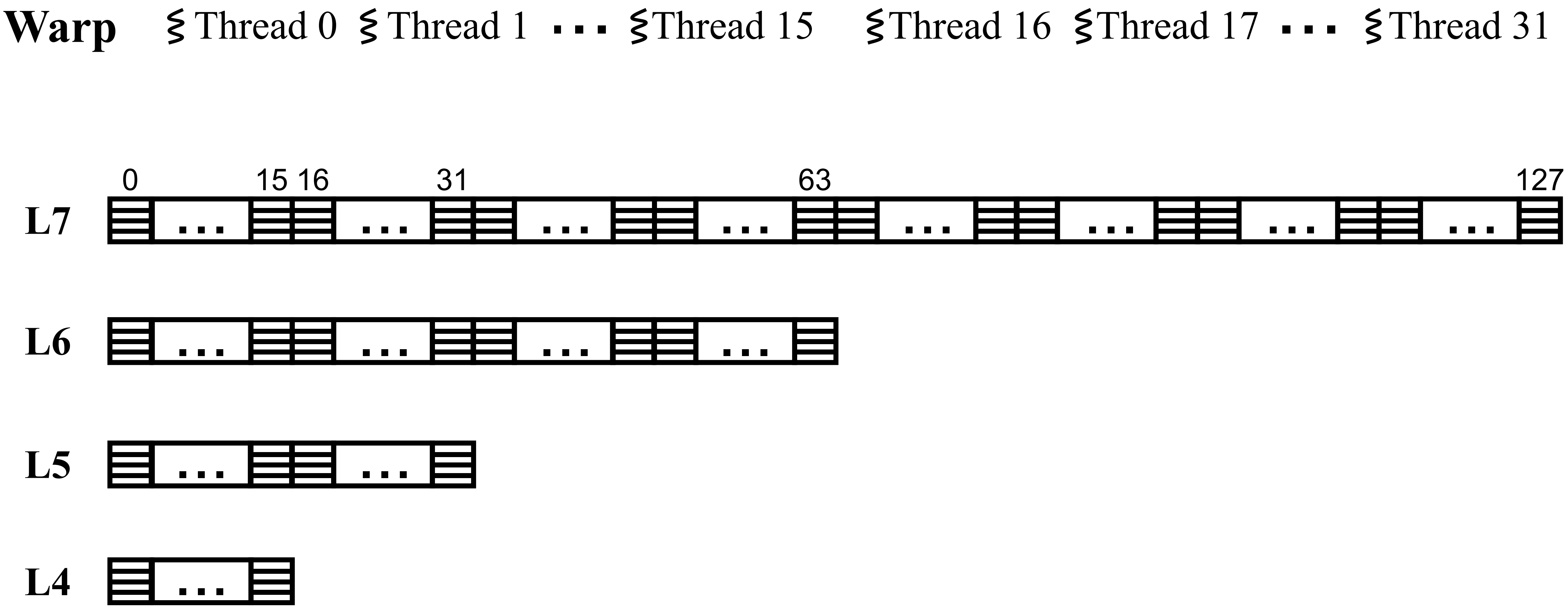
GPU-Conscious and PM-Friendly Hash Table



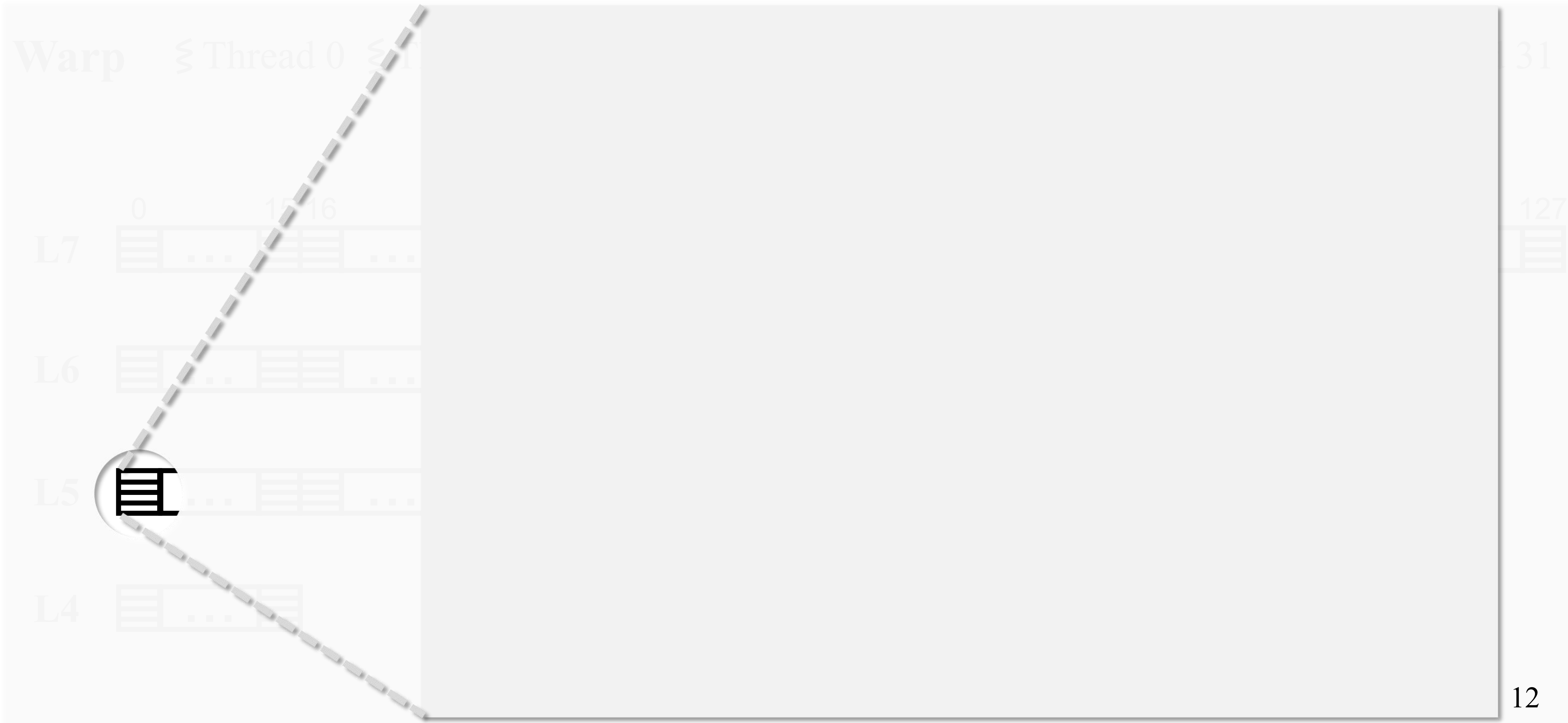
- *Multiple Hash Locations*
- *One-Shot Warp Access*

32x → 1x

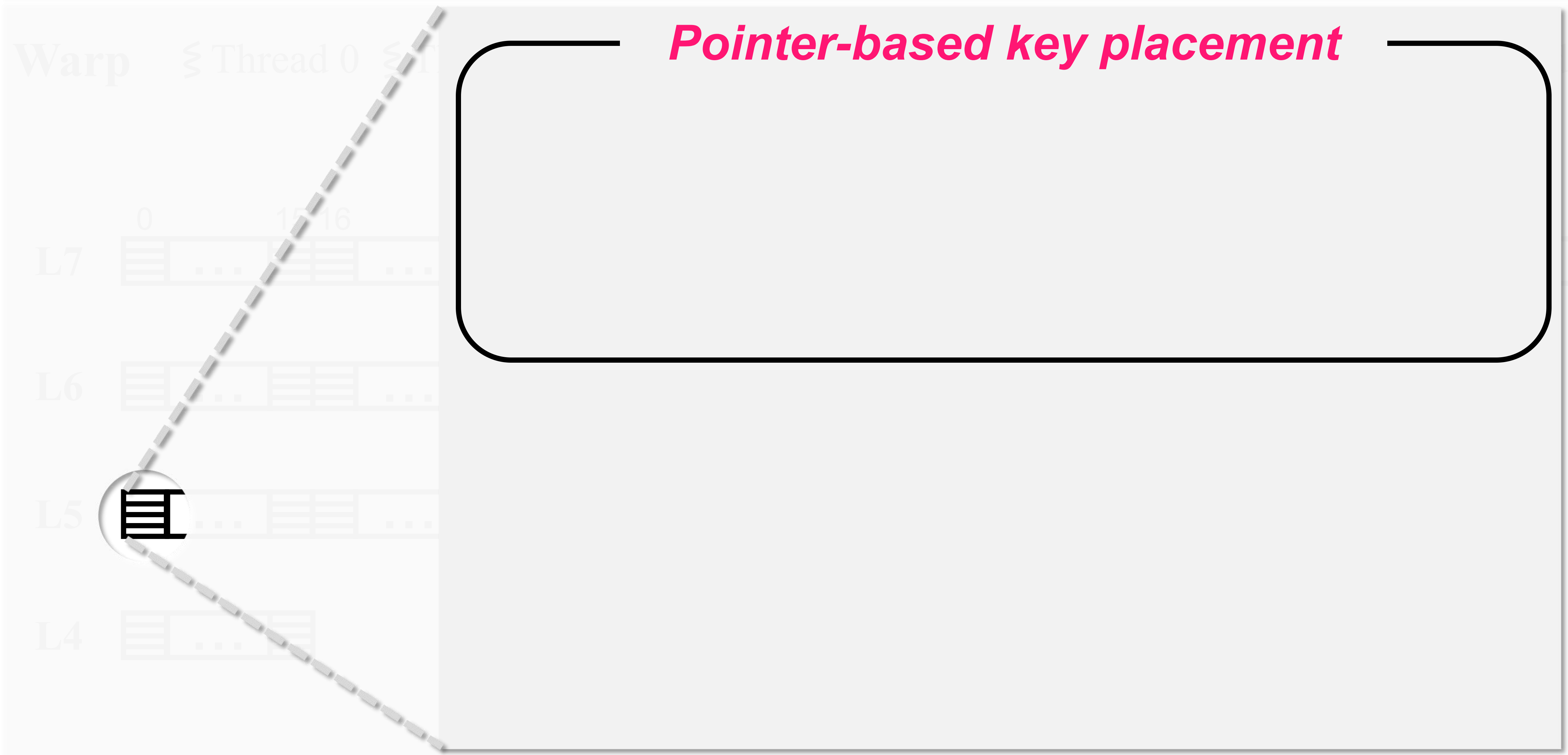
GPU-Conscious and PM-Friendly Hash Table



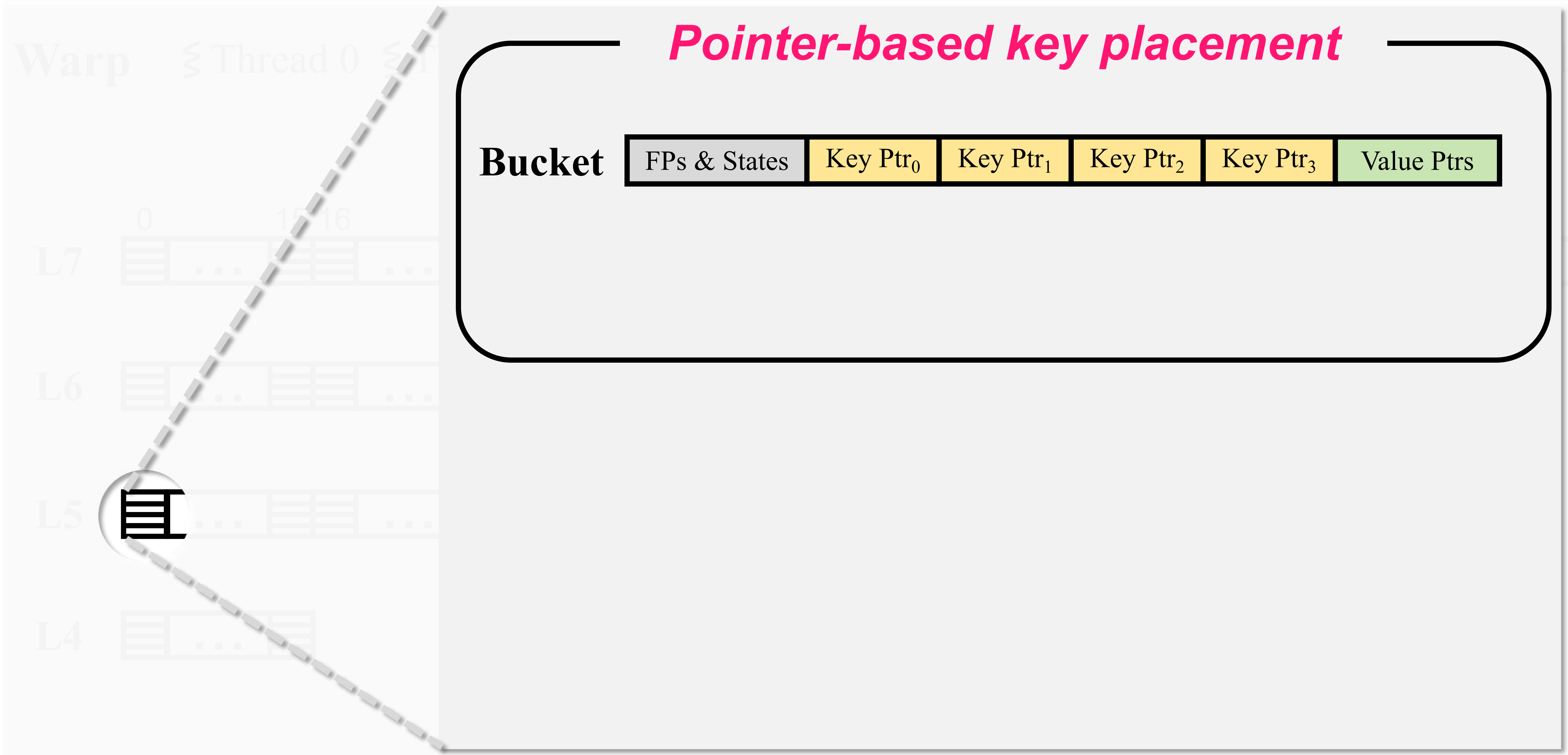
GPU-Conscious and PM-Friendly Hash Table



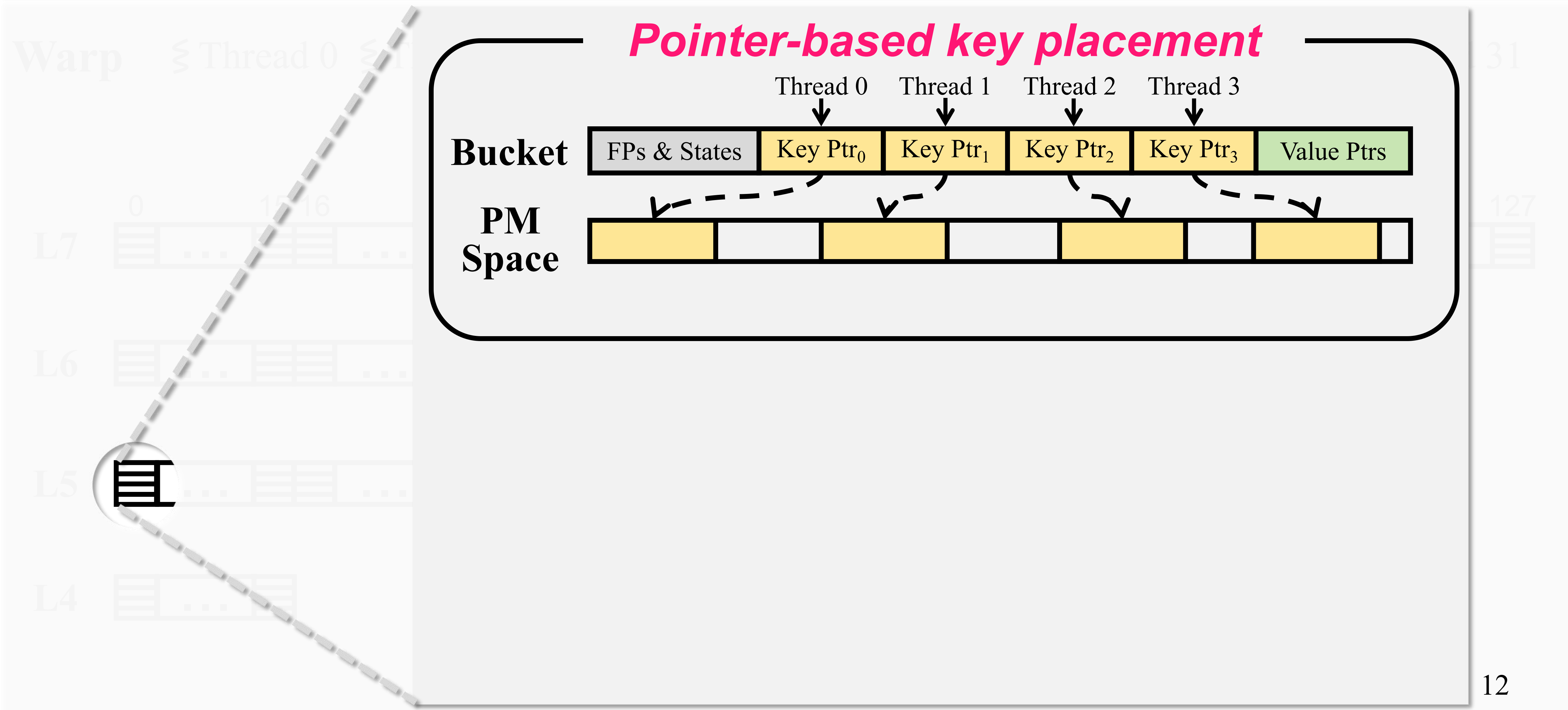
GPU-Conscious and PM-Friendly Hash Table



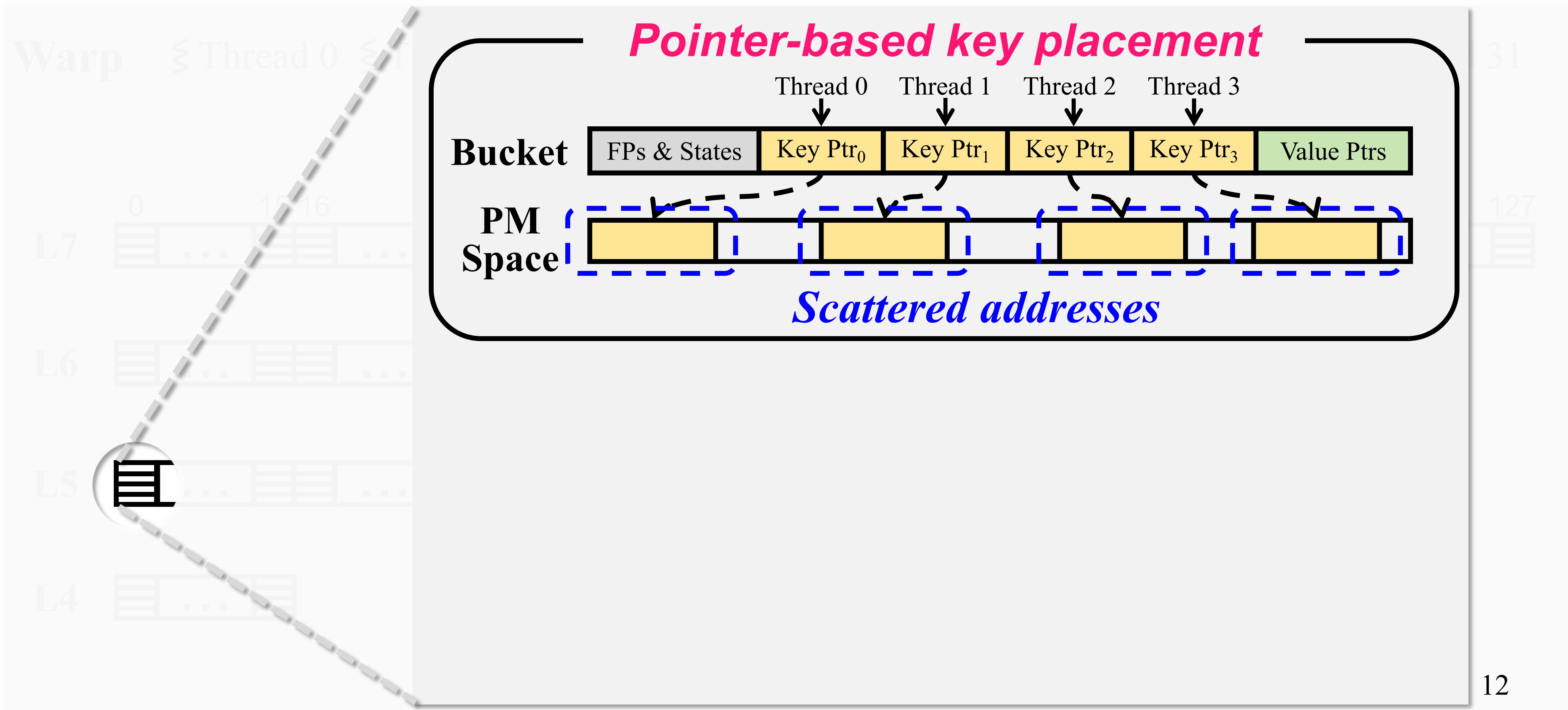
GPU-Conscious and PM-Friendly Hash Table



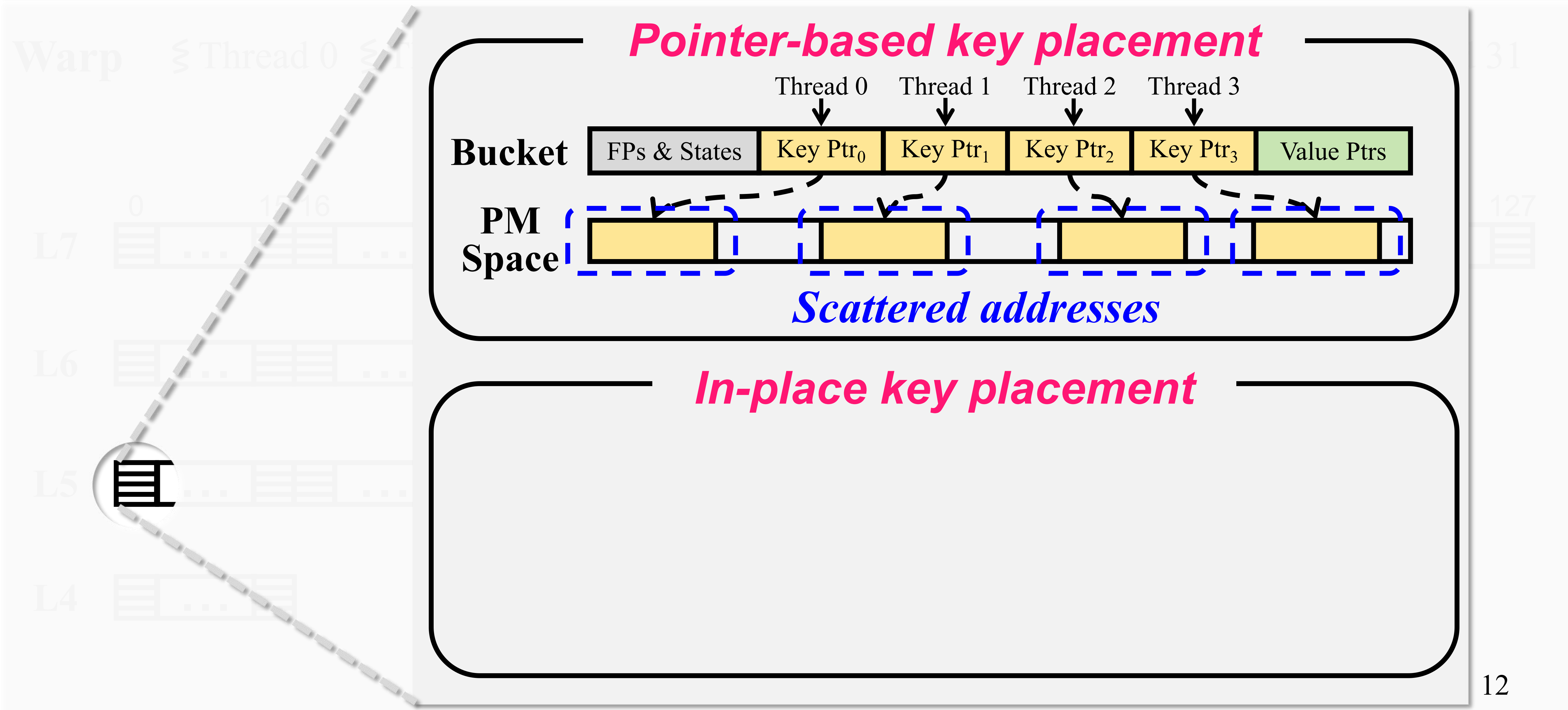
GPU-Conscious and PM-Friendly Hash Table



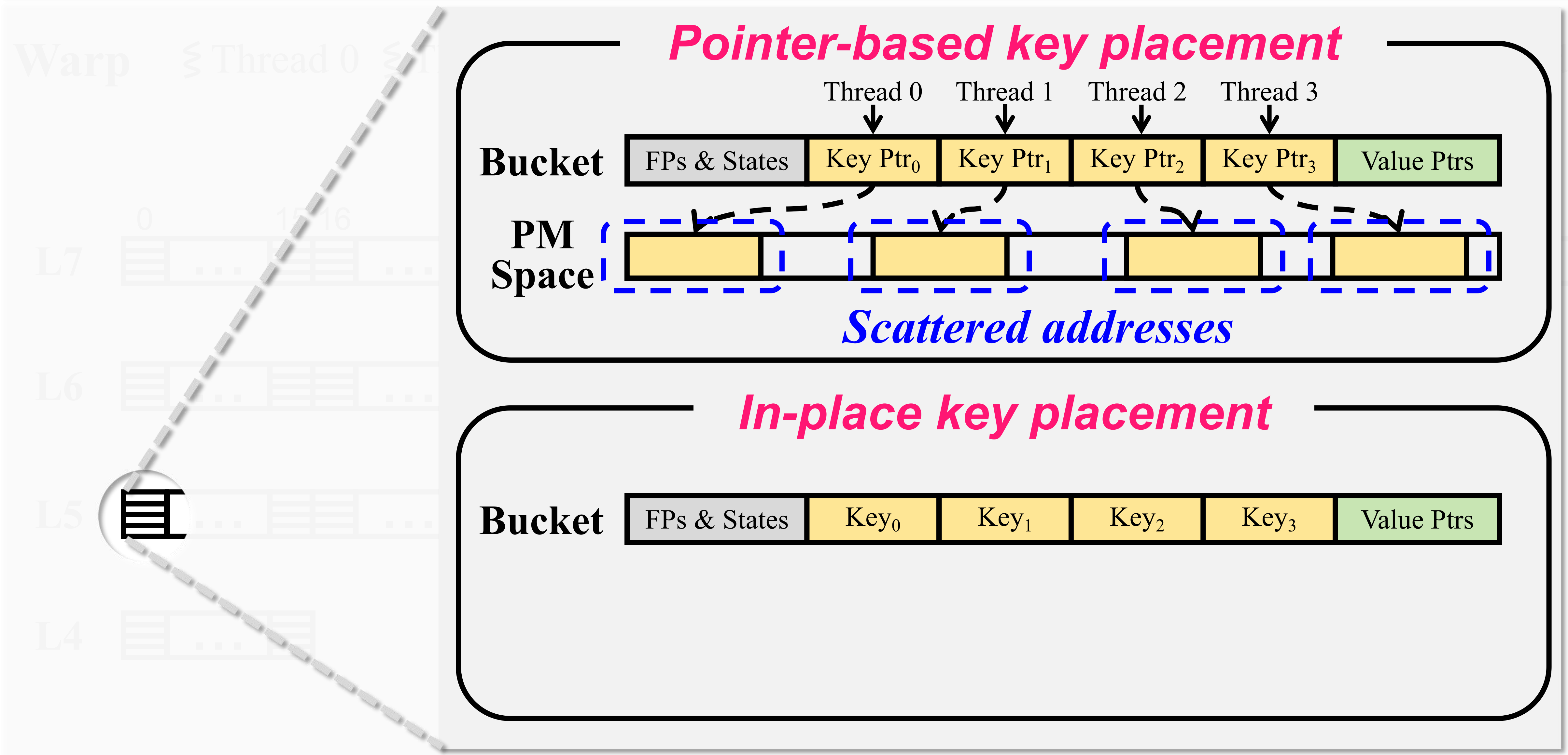
GPU-Conscious and PM-Friendly Hash Table



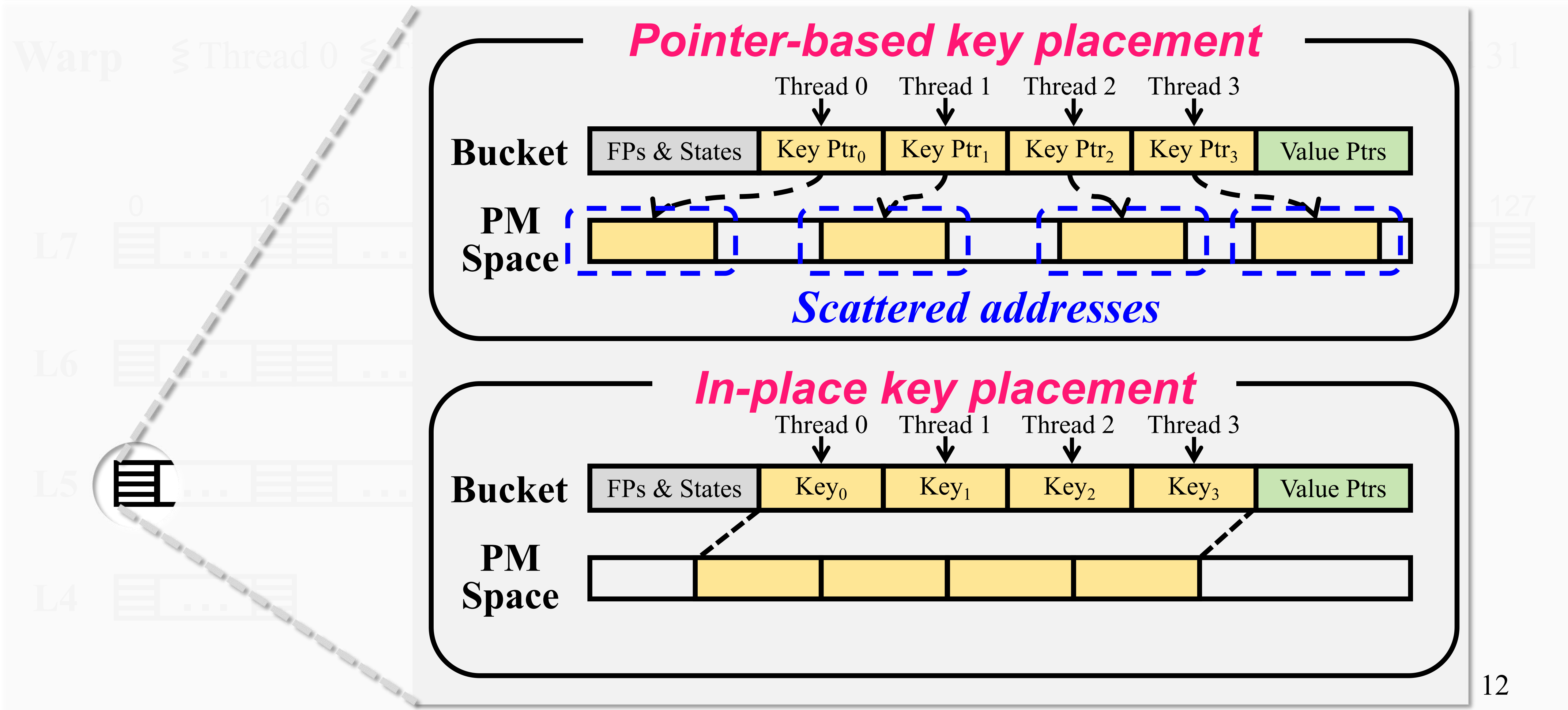
GPU-Conscious and PM-Friendly Hash Table



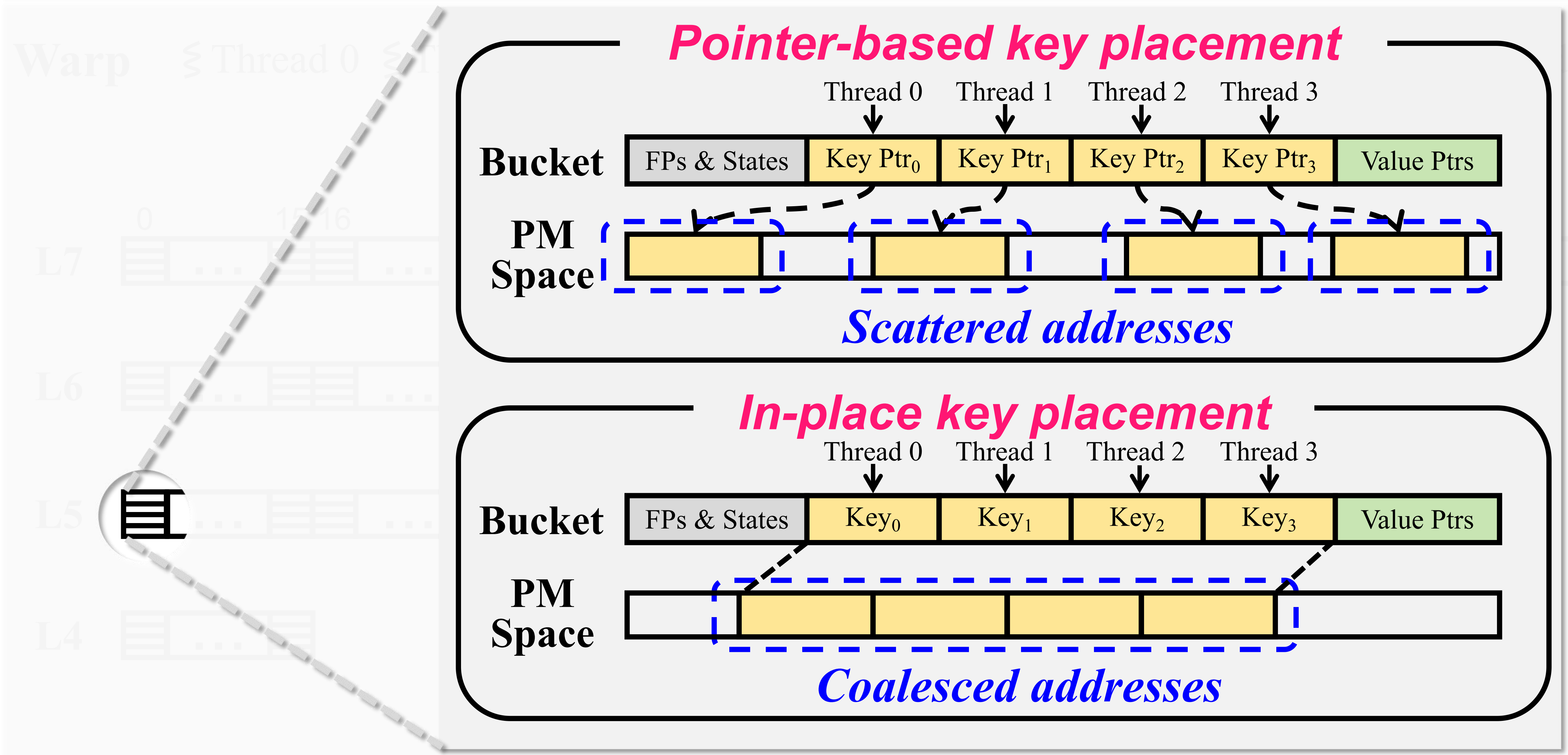
GPU-Conscious and PM-Friendly Hash Table



GPU-Conscious and PM-Friendly Hash Table



GPU-Conscious and PM-Friendly Hash Table



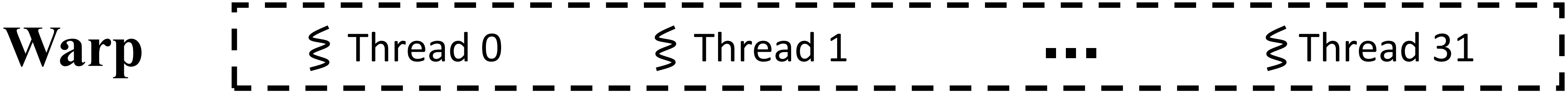
GPU-Conscious and PM-Friendly Hash Table

GPU-Conscious and PM-Friendly Hash Table

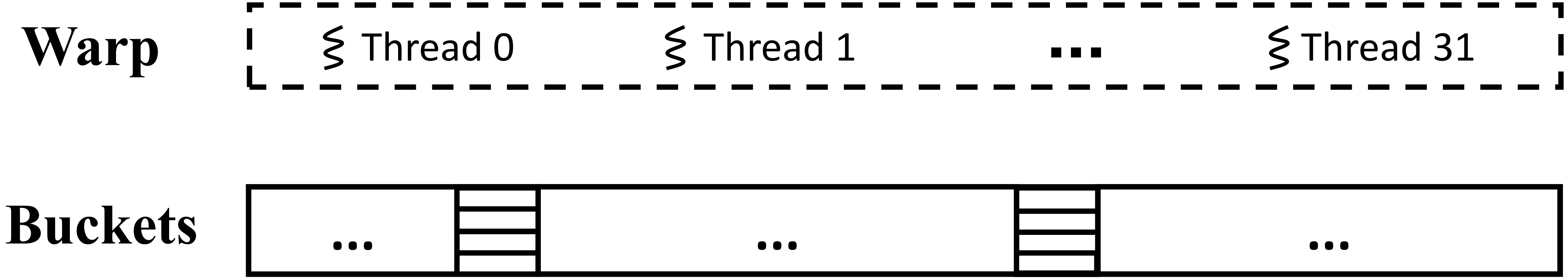
- **GPU-friendly:** *minimize warp divergence and uncoalesced memory accesses*
- **Write-optimized:** *each insertion only involves a constant number of buckets without any data movement*
- **Memory-efficient:** *achieve a high load factor that is up to 92%*

Lock-Free and Log-Free Insertion

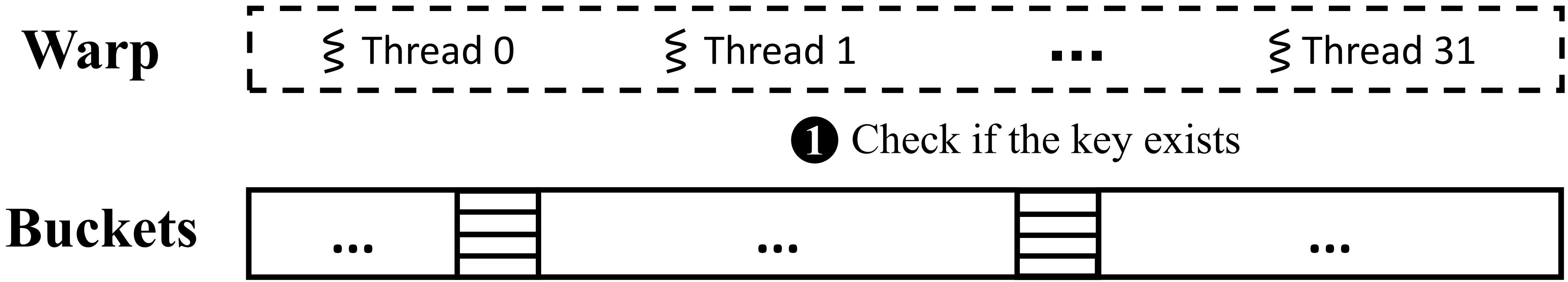
Lock-Free and Log-Free Insertion



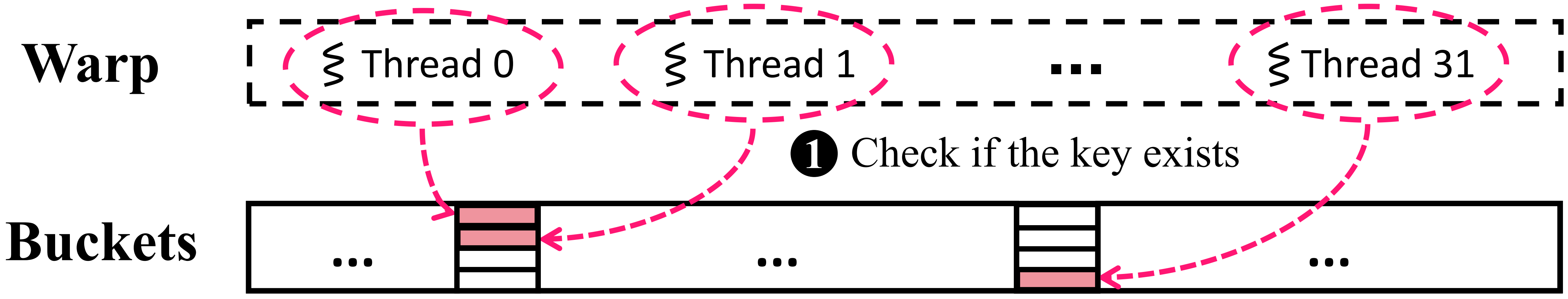
Lock-Free and Log-Free Insertion



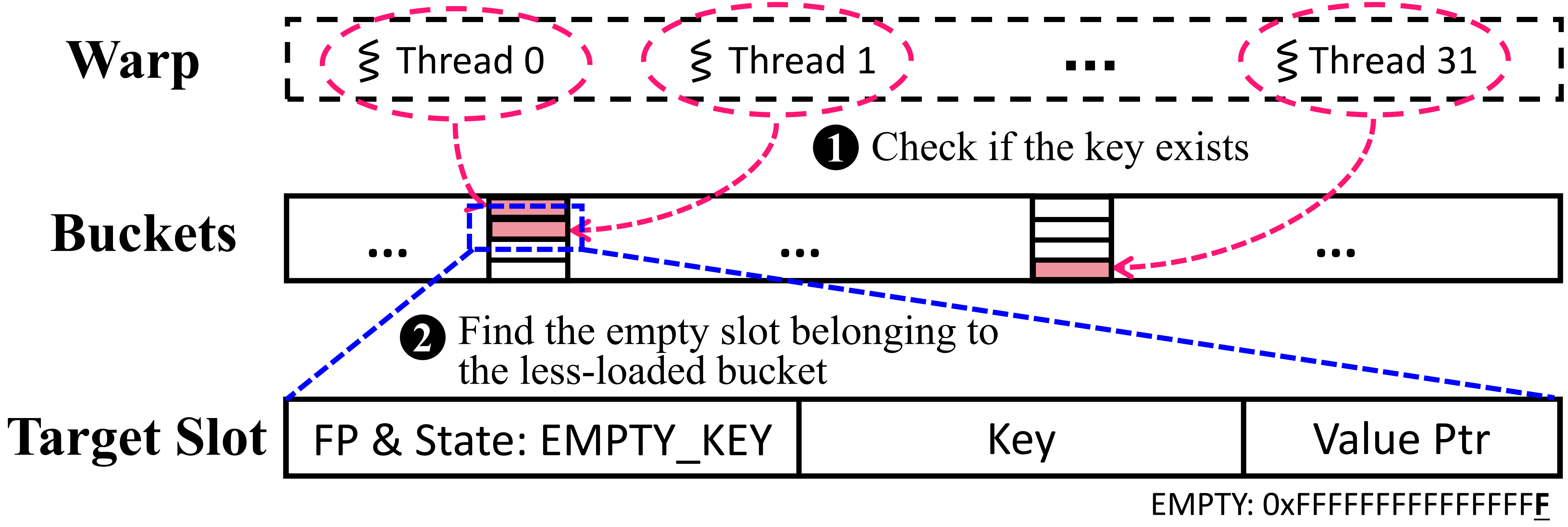
Lock-Free and Log-Free Insertion



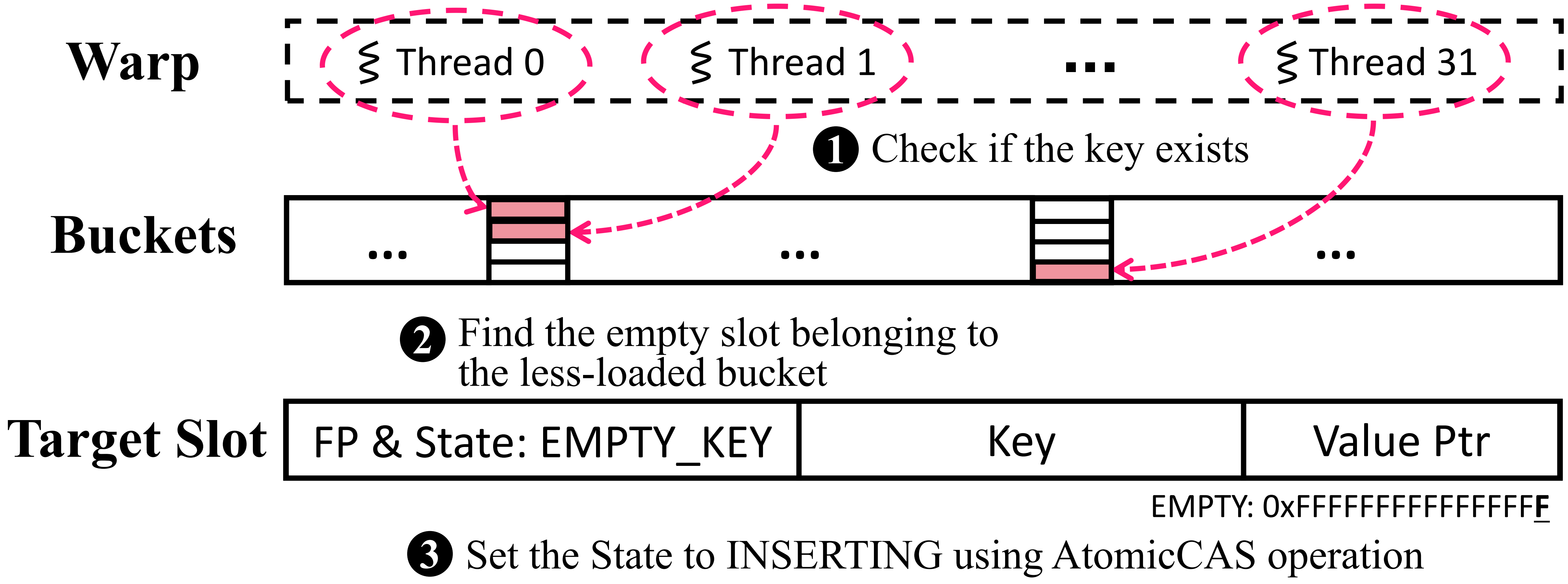
Lock-Free and Log-Free Insertion



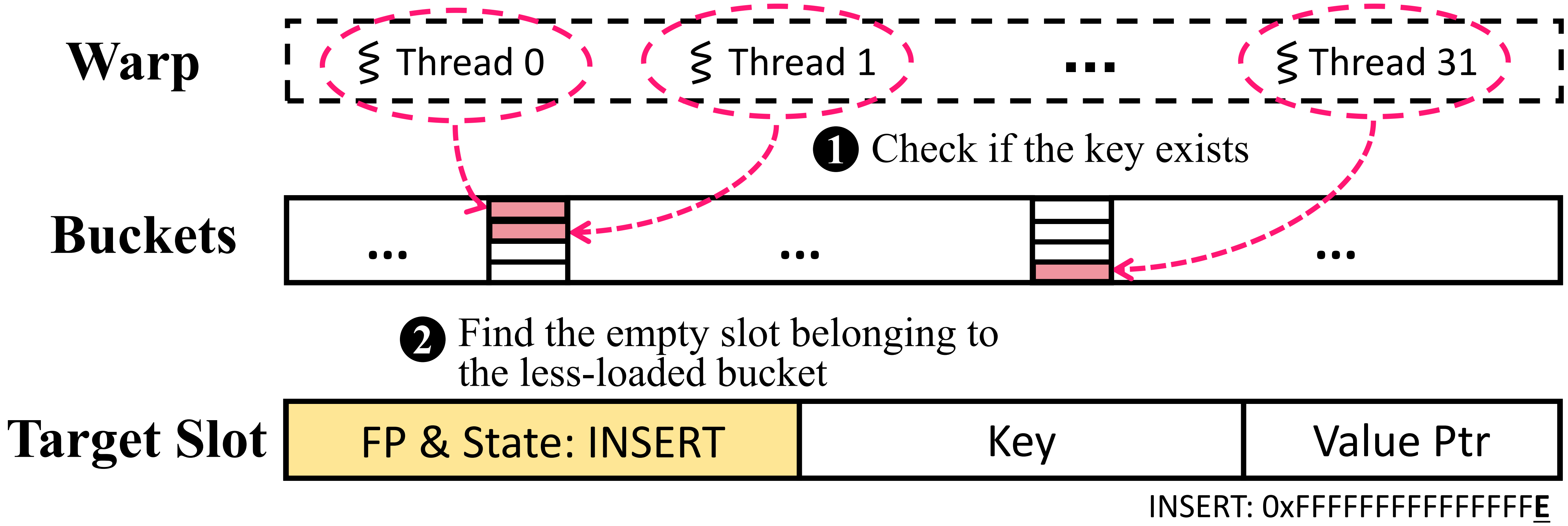
Lock-Free and Log-Free Insertion



Lock-Free and Log-Free Insertion

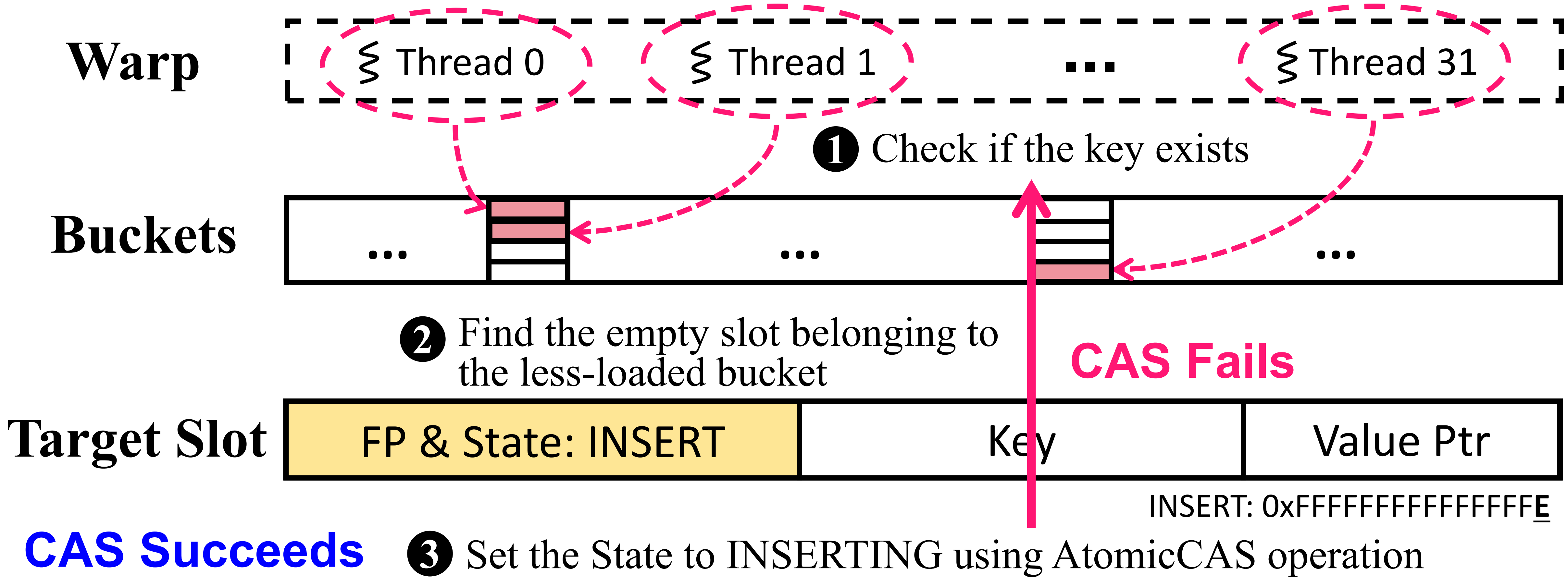


Lock-Free and Log-Free Insertion

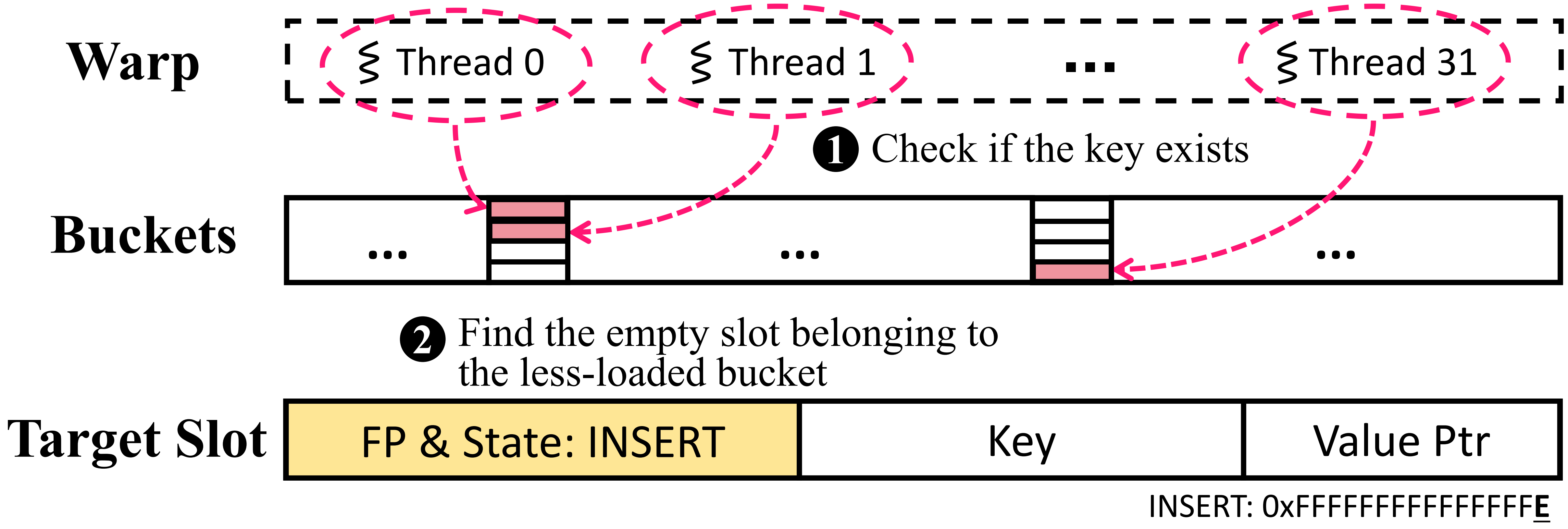


CAS Succeeds ❸ Set the State to INSERTING using AtomicCAS operation

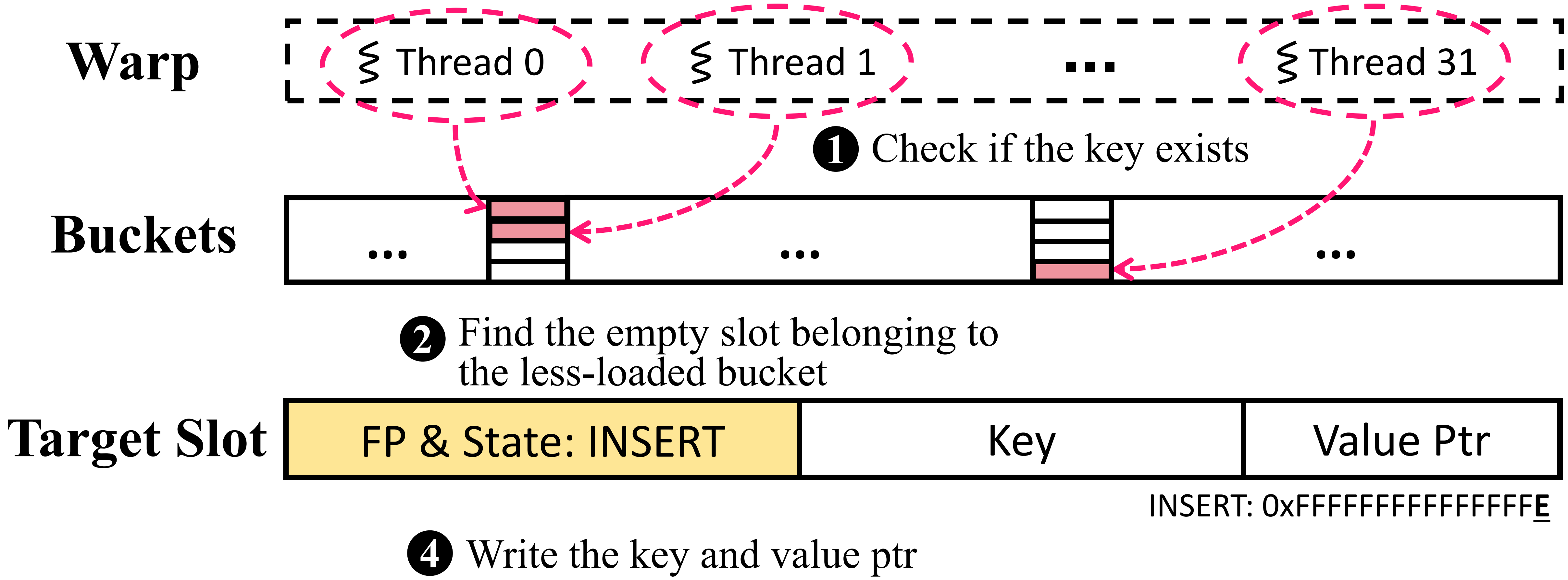
Lock-Free and Log-Free Insertion



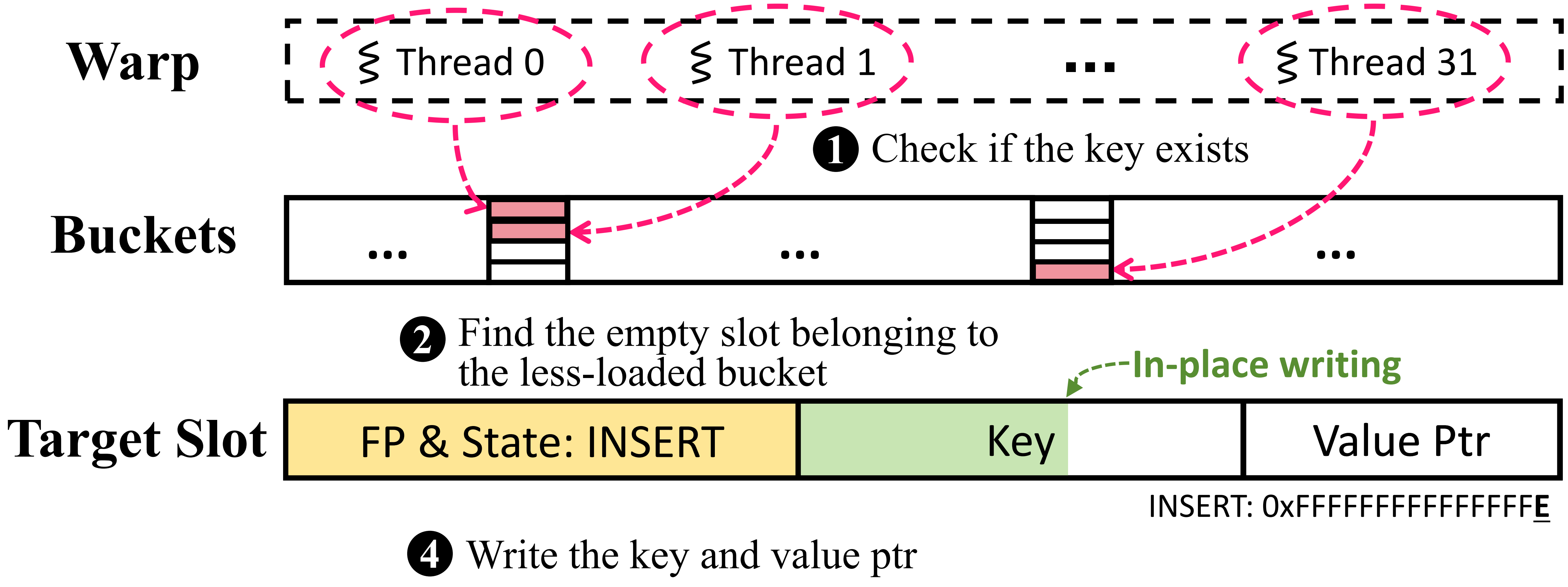
Lock-Free and Log-Free Insertion



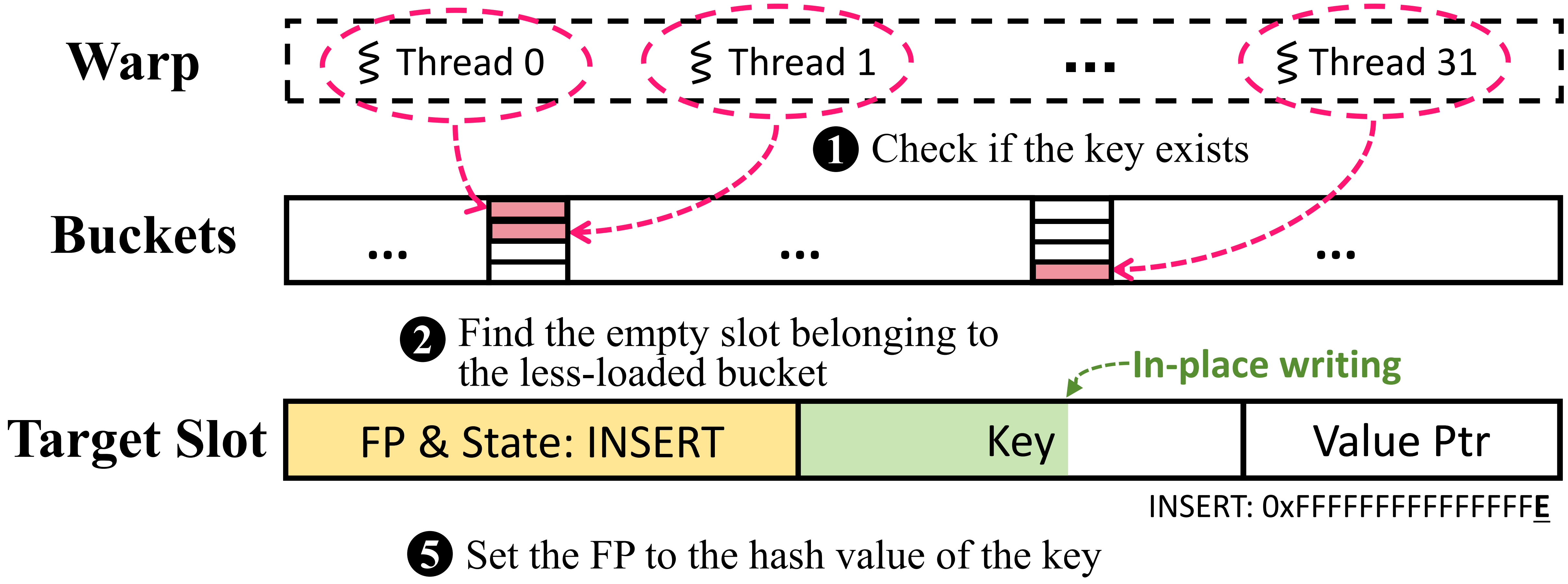
Lock-Free and Log-Free Insertion



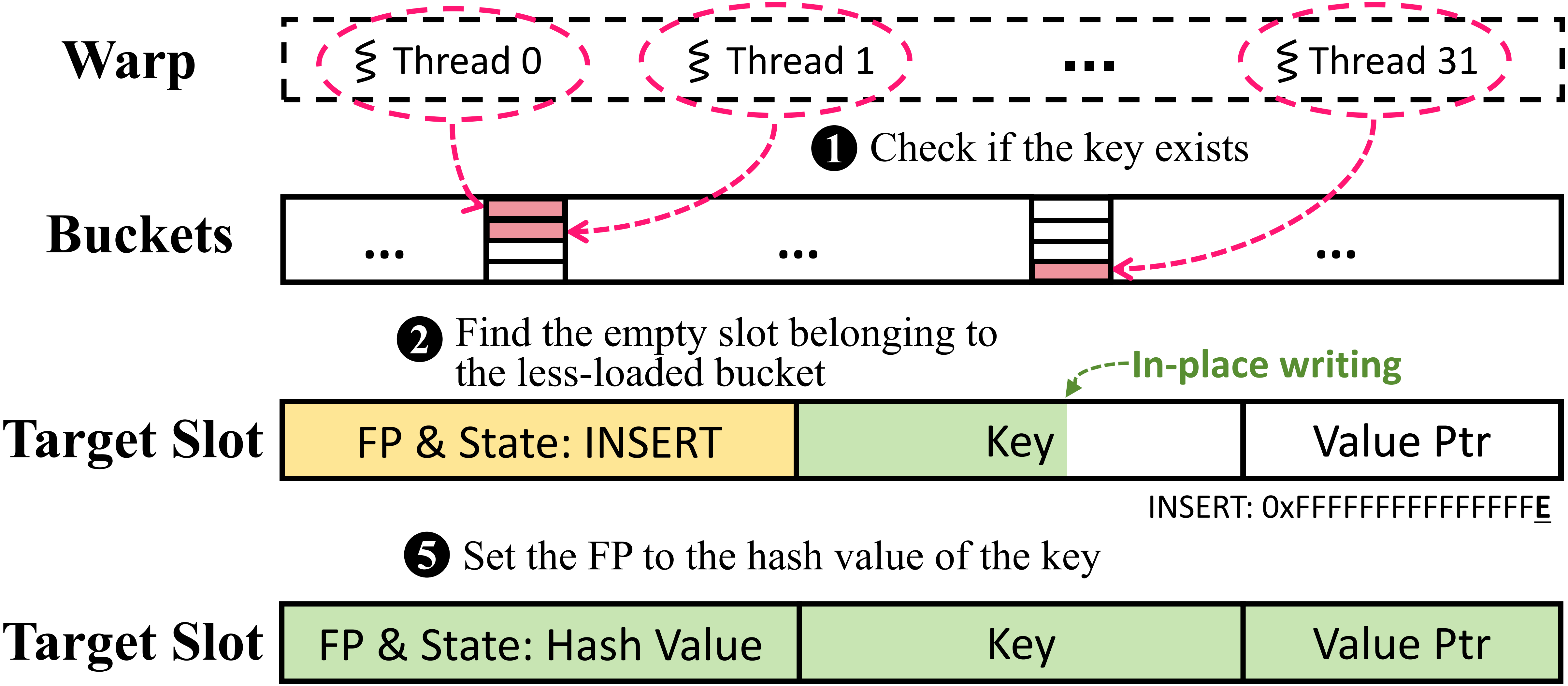
Lock-Free and Log-Free Insertion



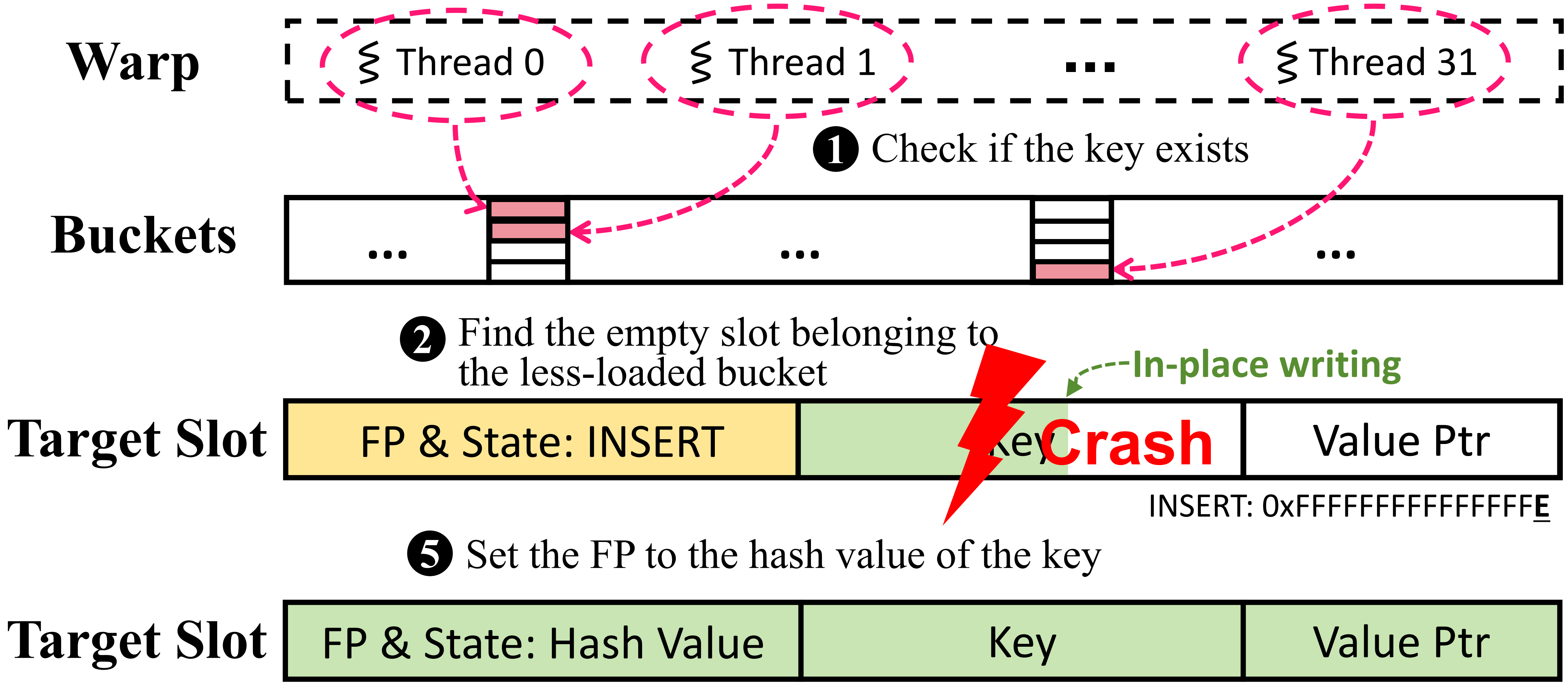
Lock-Free and Log-Free Insertion



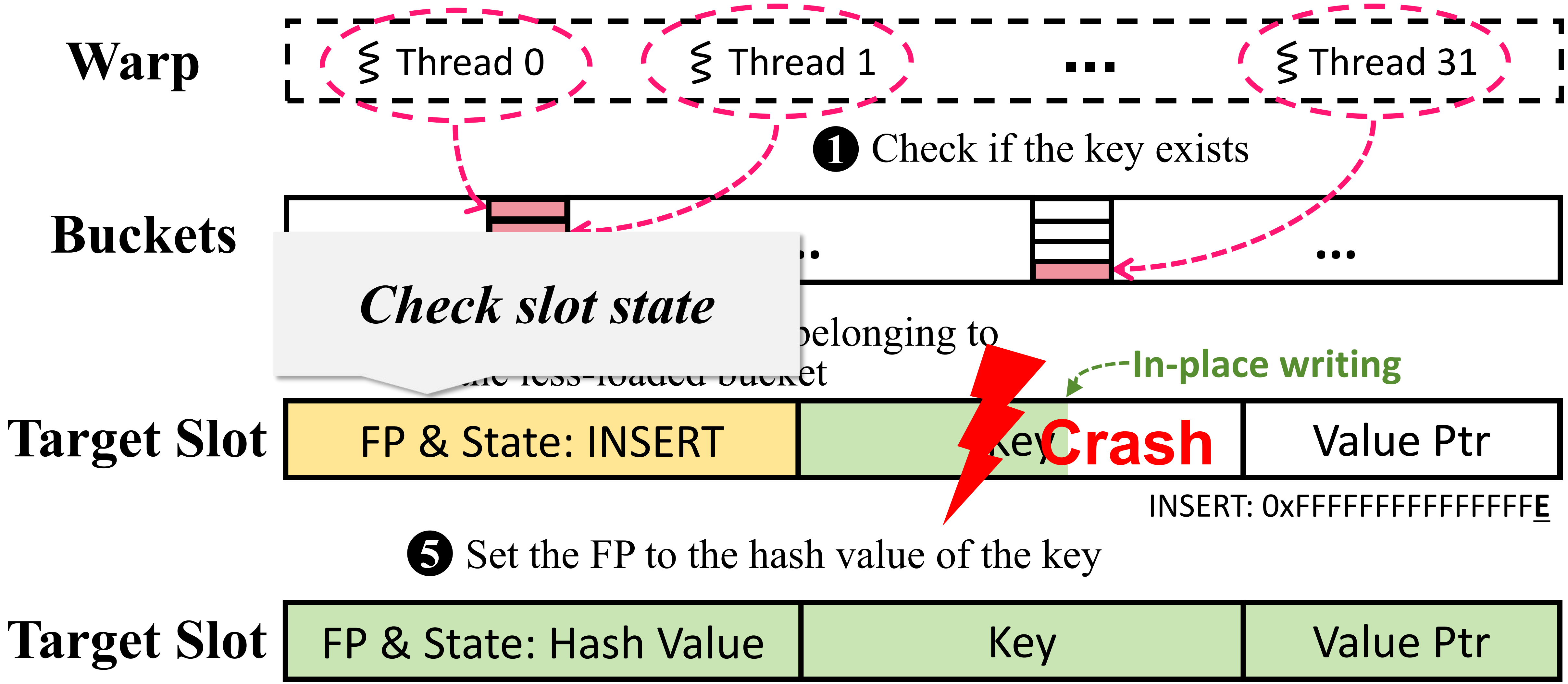
Lock-Free and Log-Free Insertion



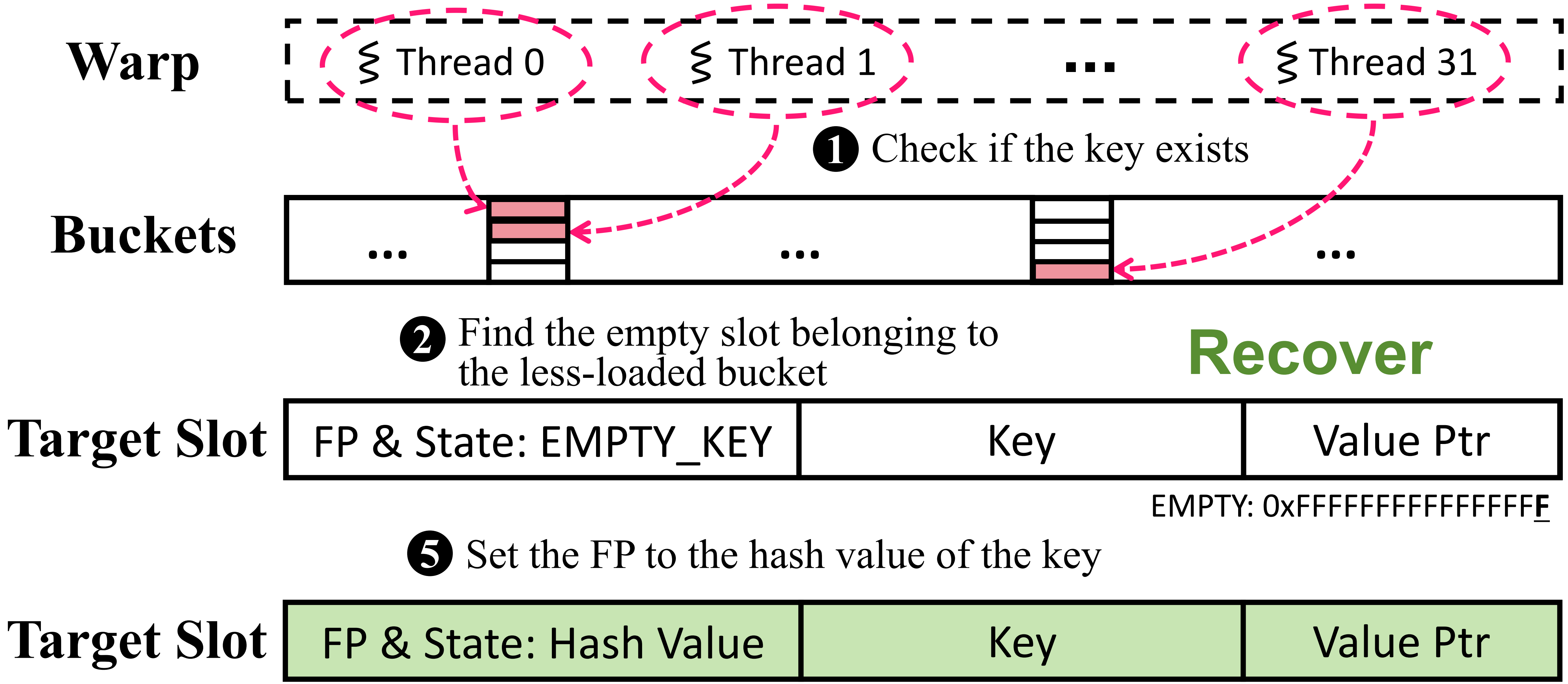
Lock-Free and Log-Free Insertion



Lock-Free and Log-Free Insertion



Lock-Free and Log-Free Insertion



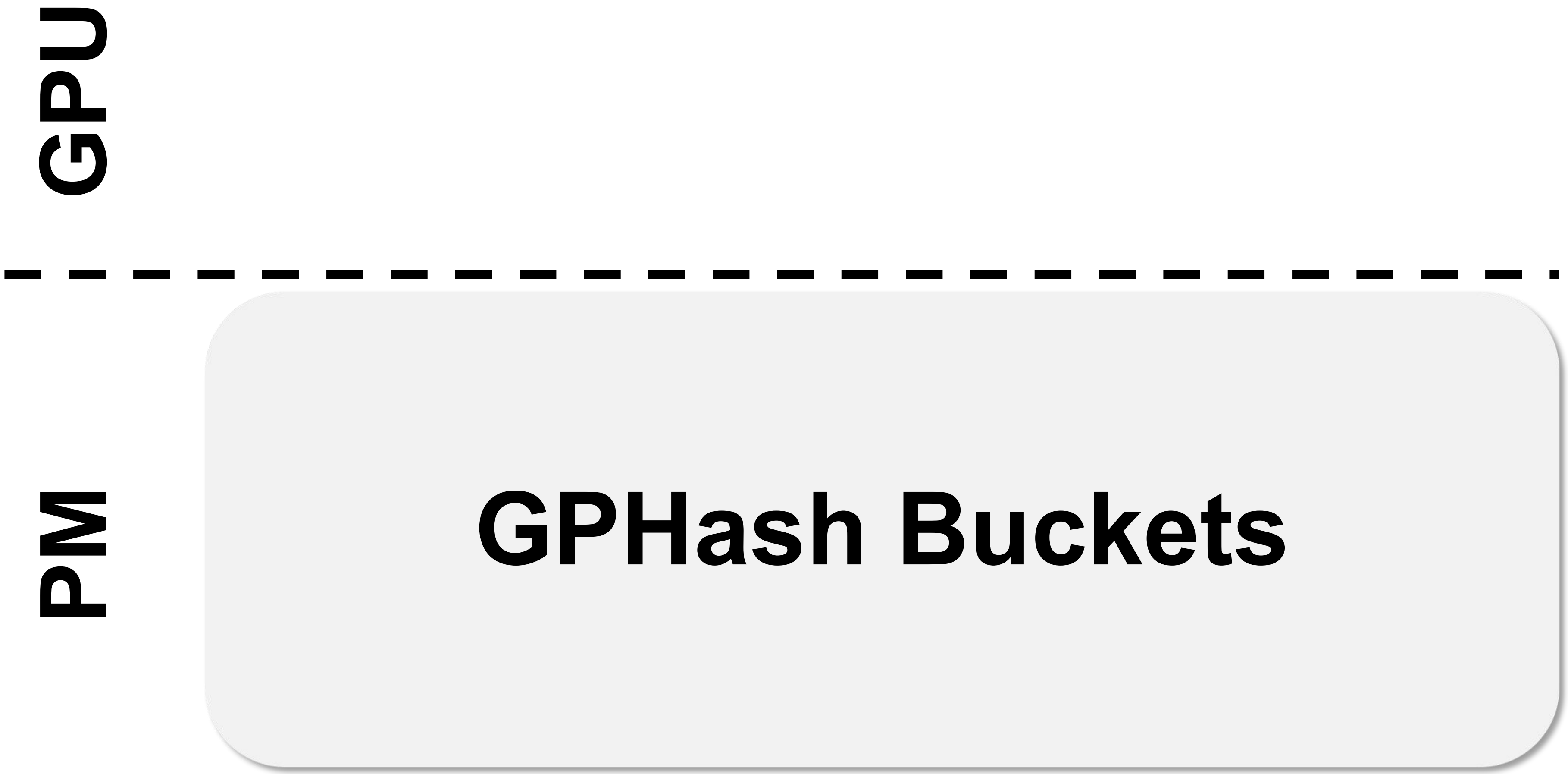
Frozen-Based Bucket Cache

Frozen-Based Bucket Cache

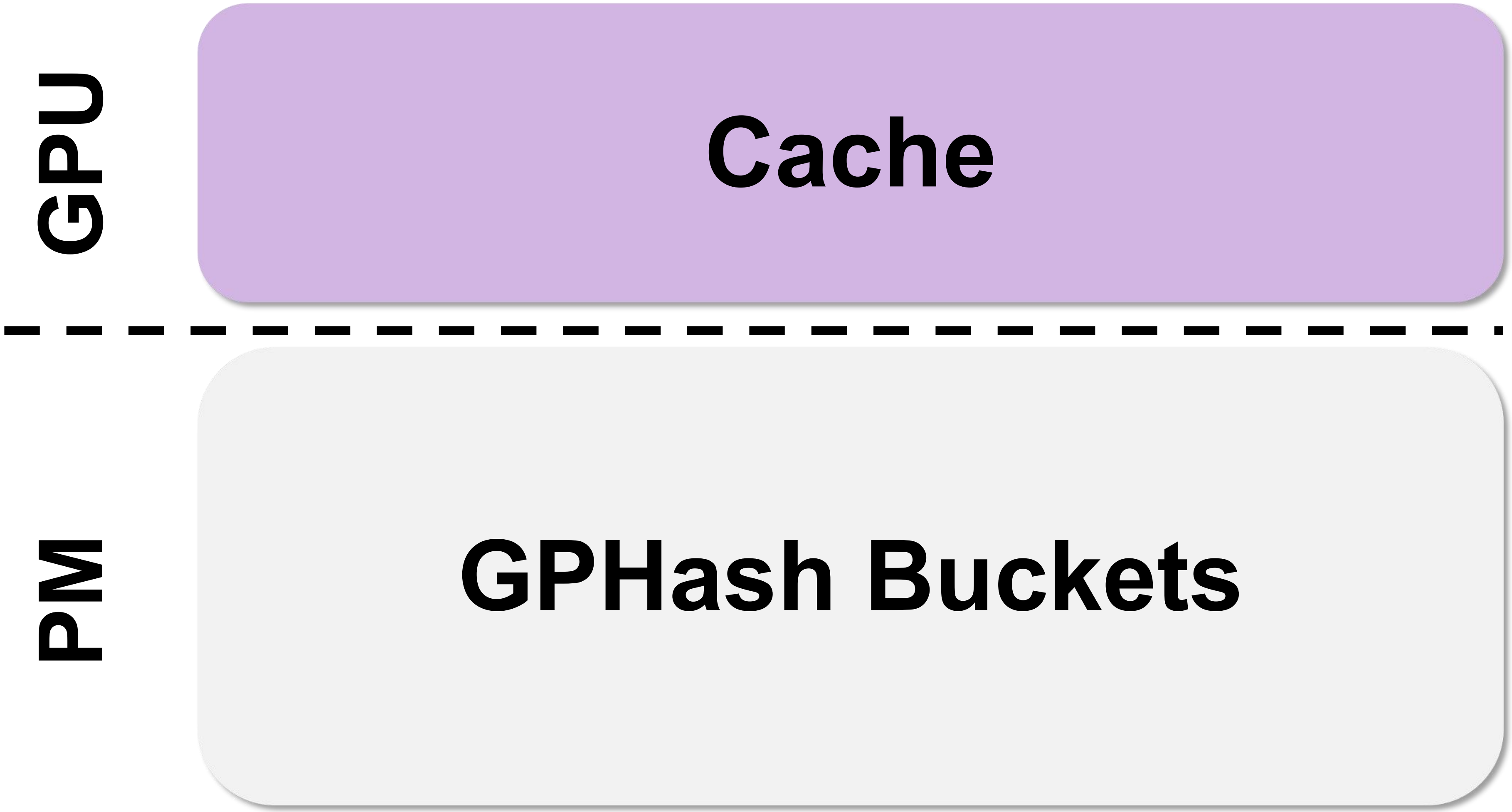
GPU

PM

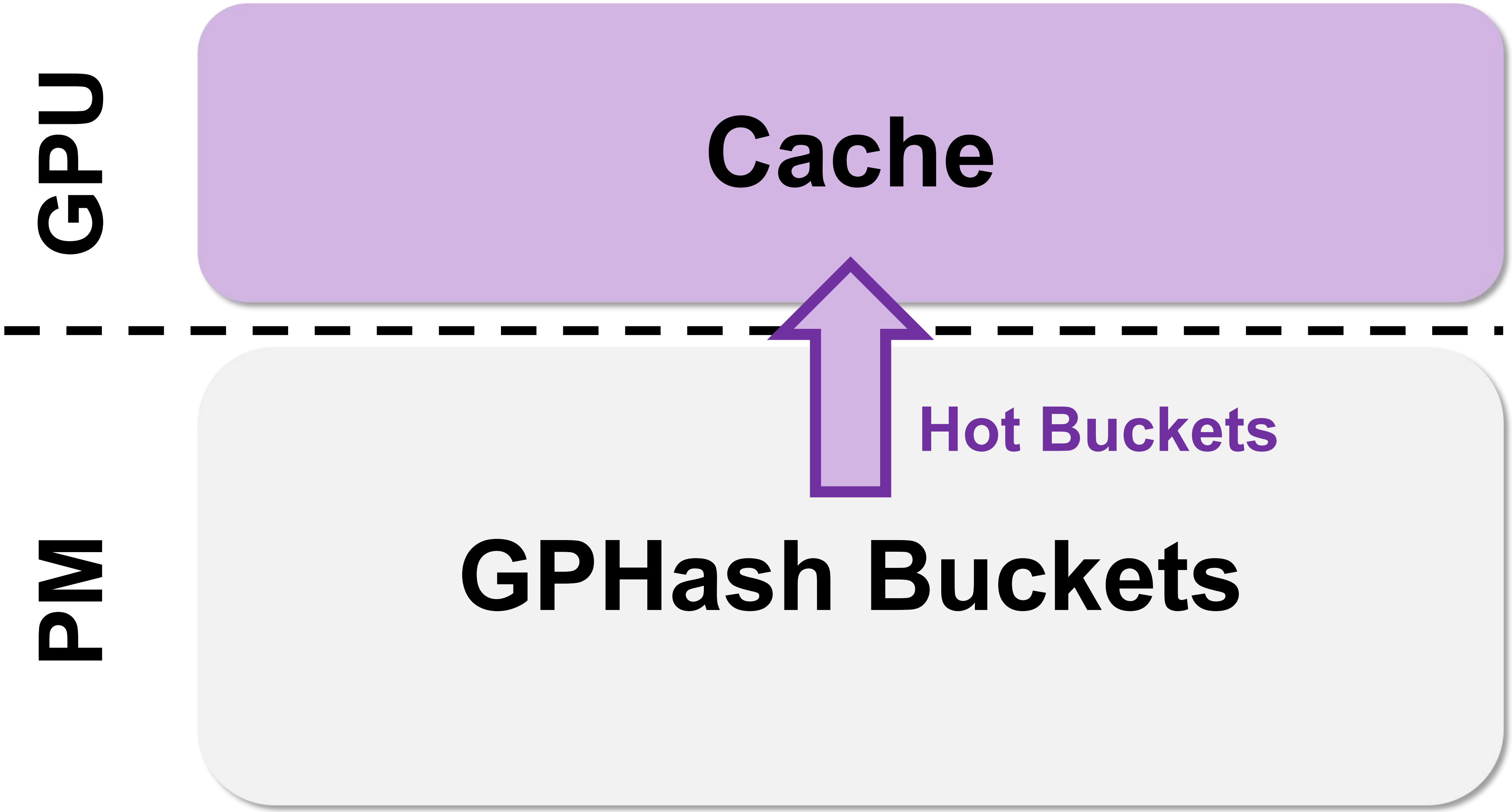
Frozen-Based Bucket Cache



Frozen-Based Bucket Cache



Frozen-Based Bucket Cache

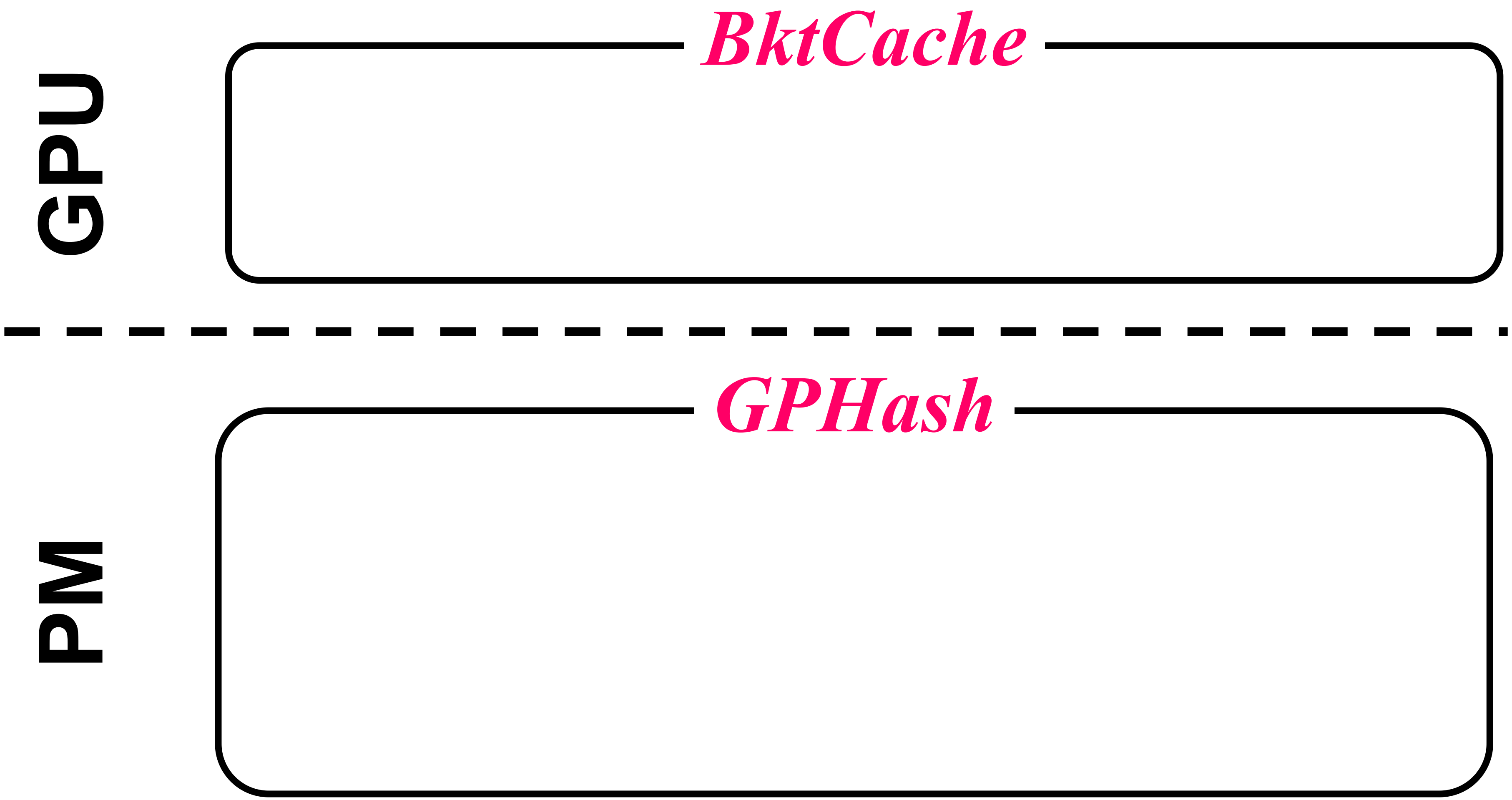


Frozen-Based Bucket Cache

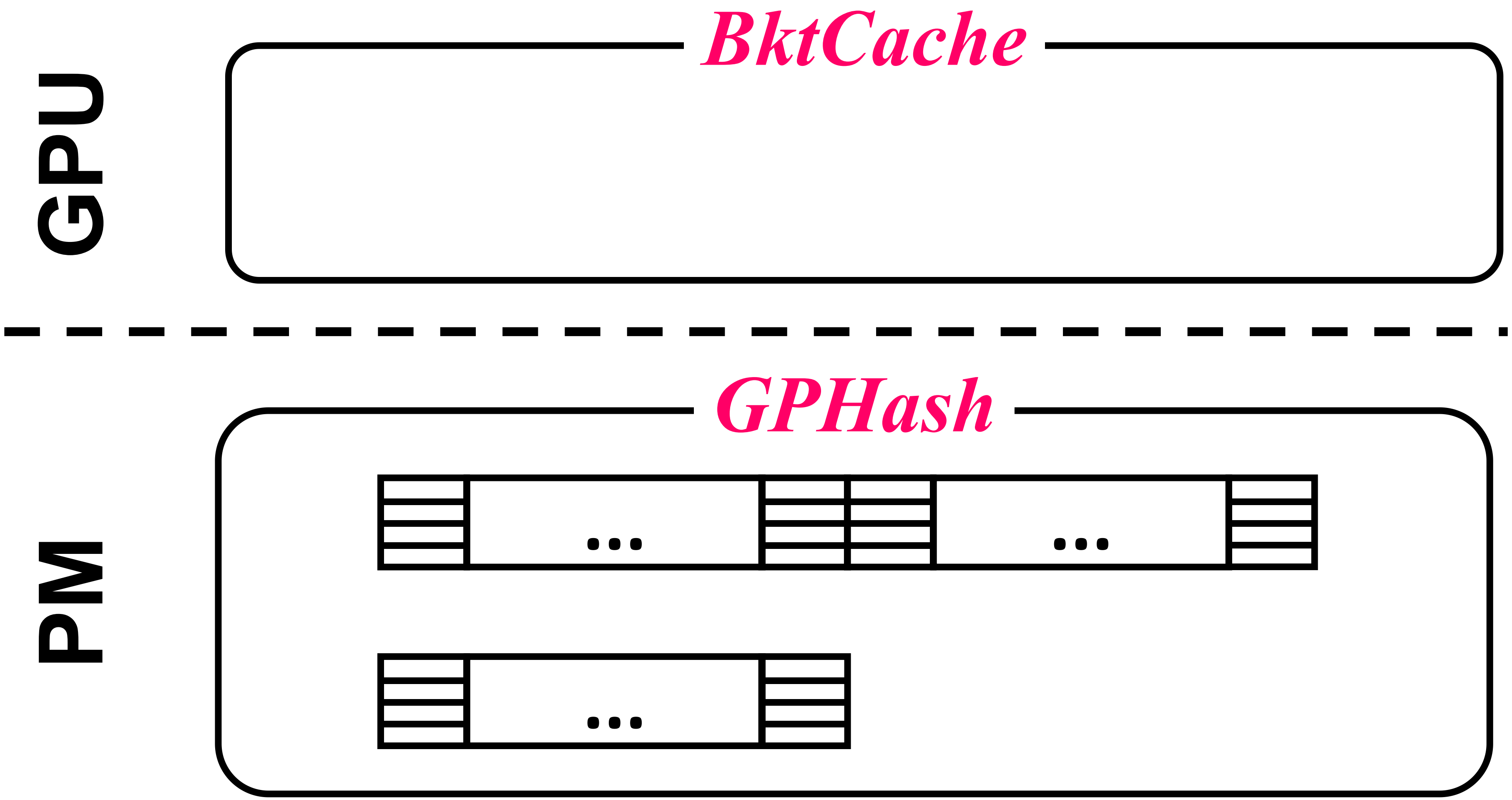
GPU

PM

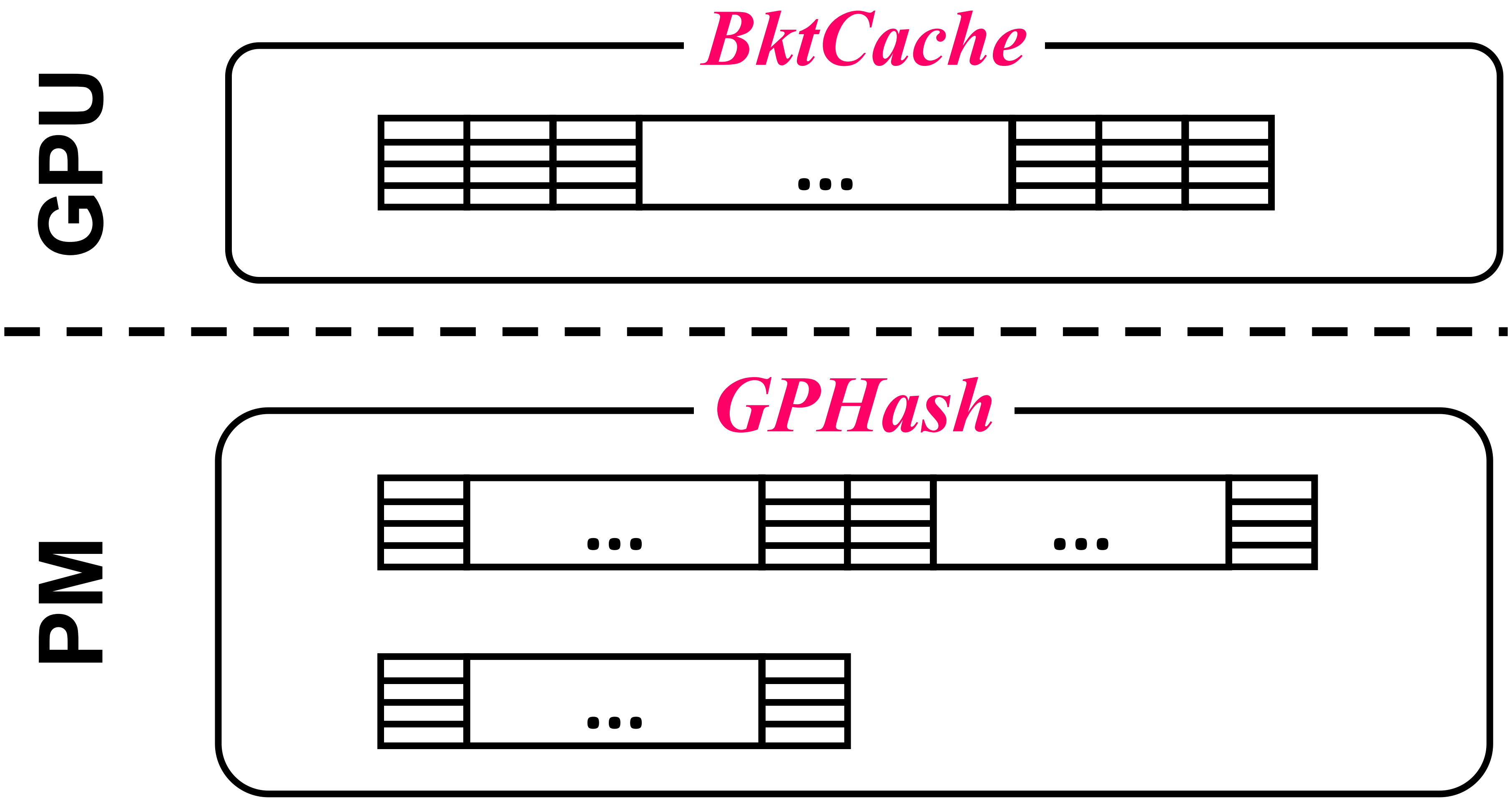
Frozen-Based Bucket Cache



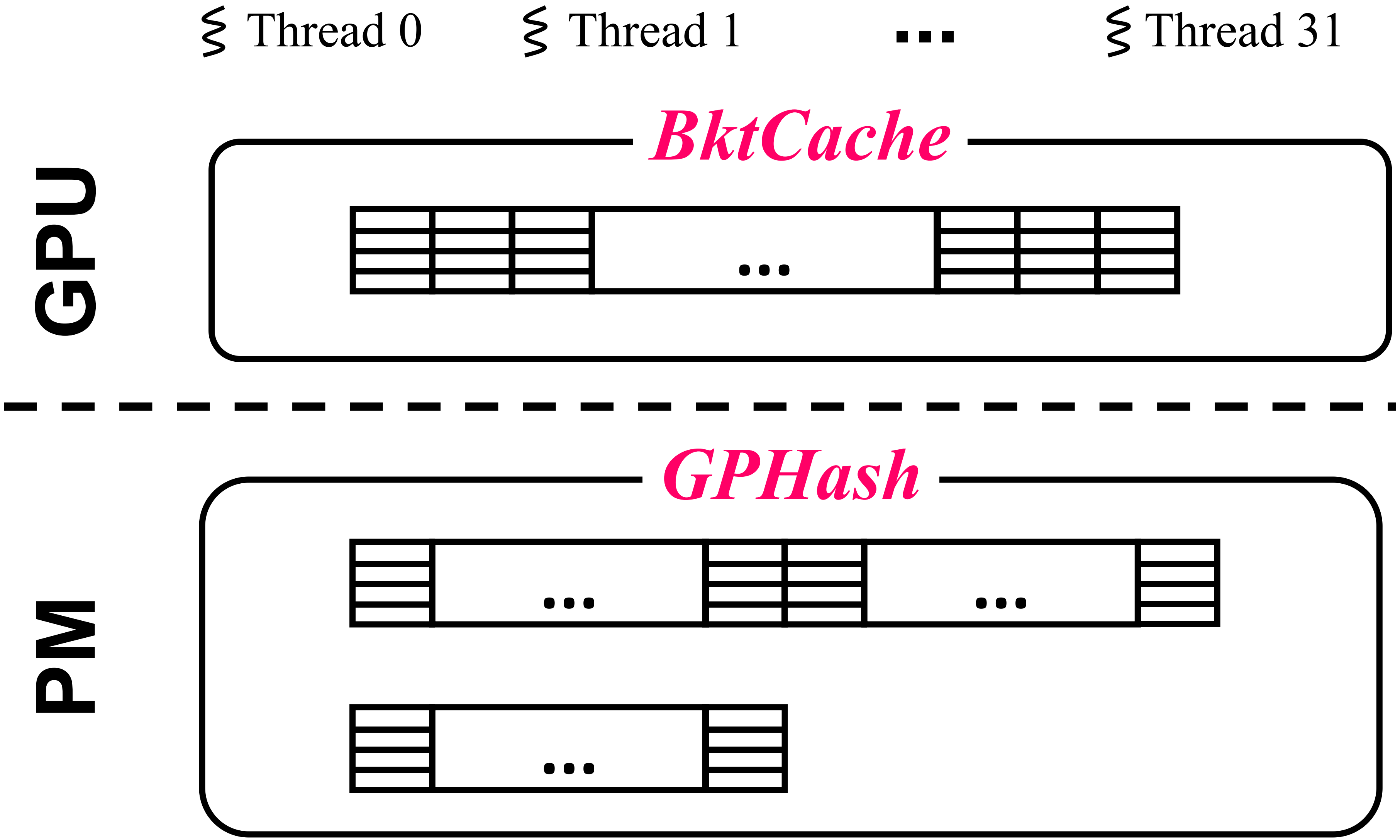
Frozen-Based Bucket Cache



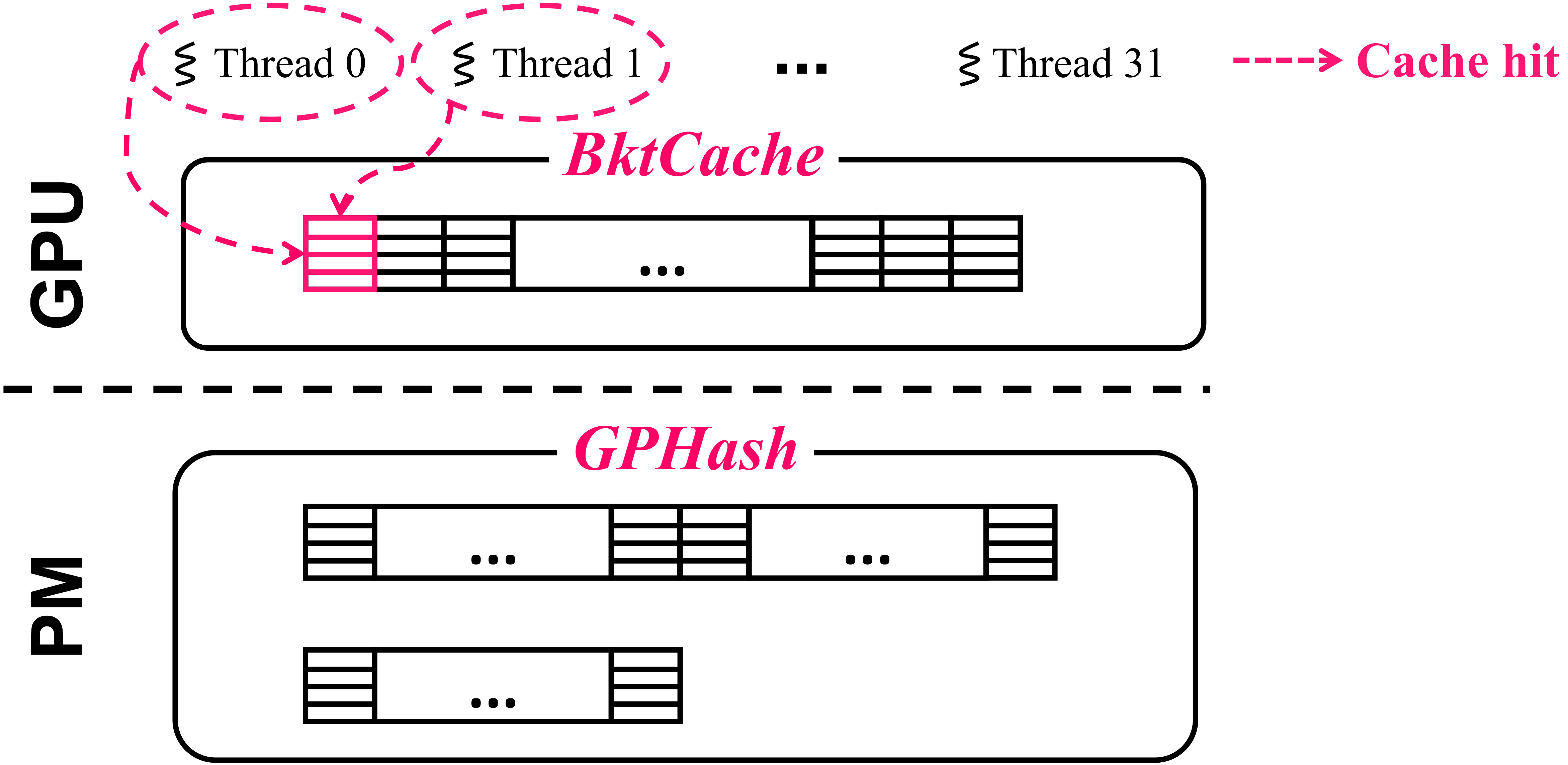
Frozen-Based Bucket Cache



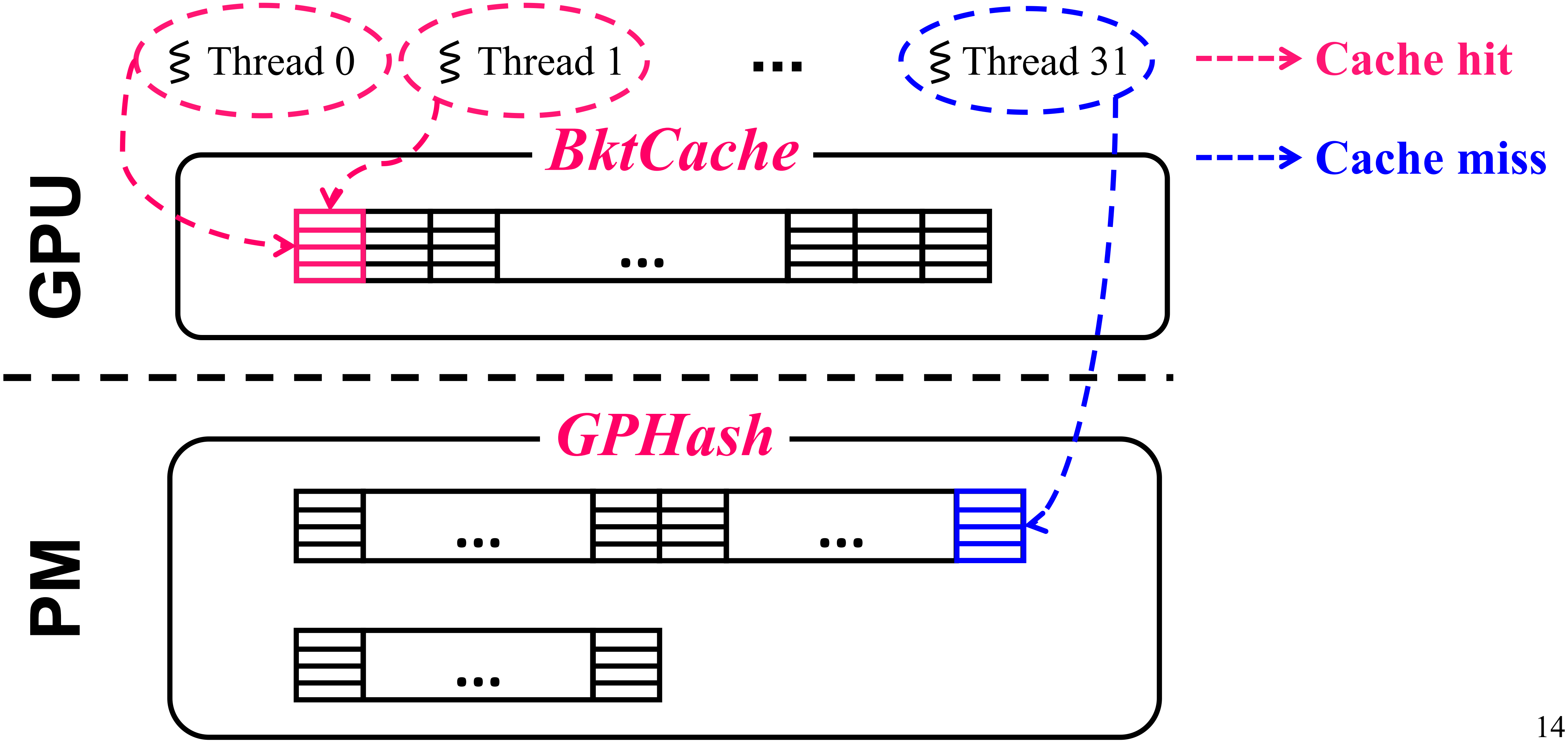
Frozen-Based Bucket Cache



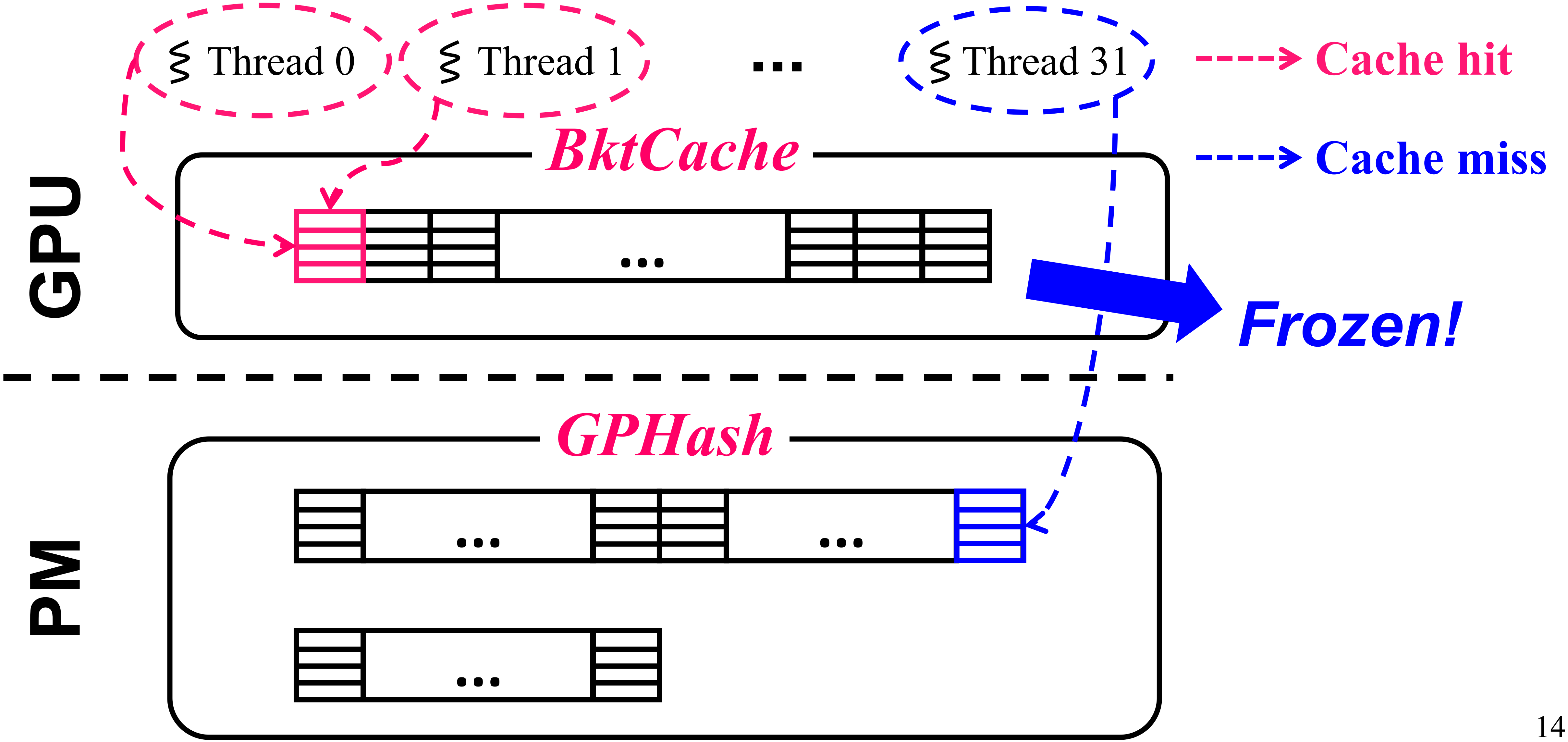
Frozen-Based Bucket Cache



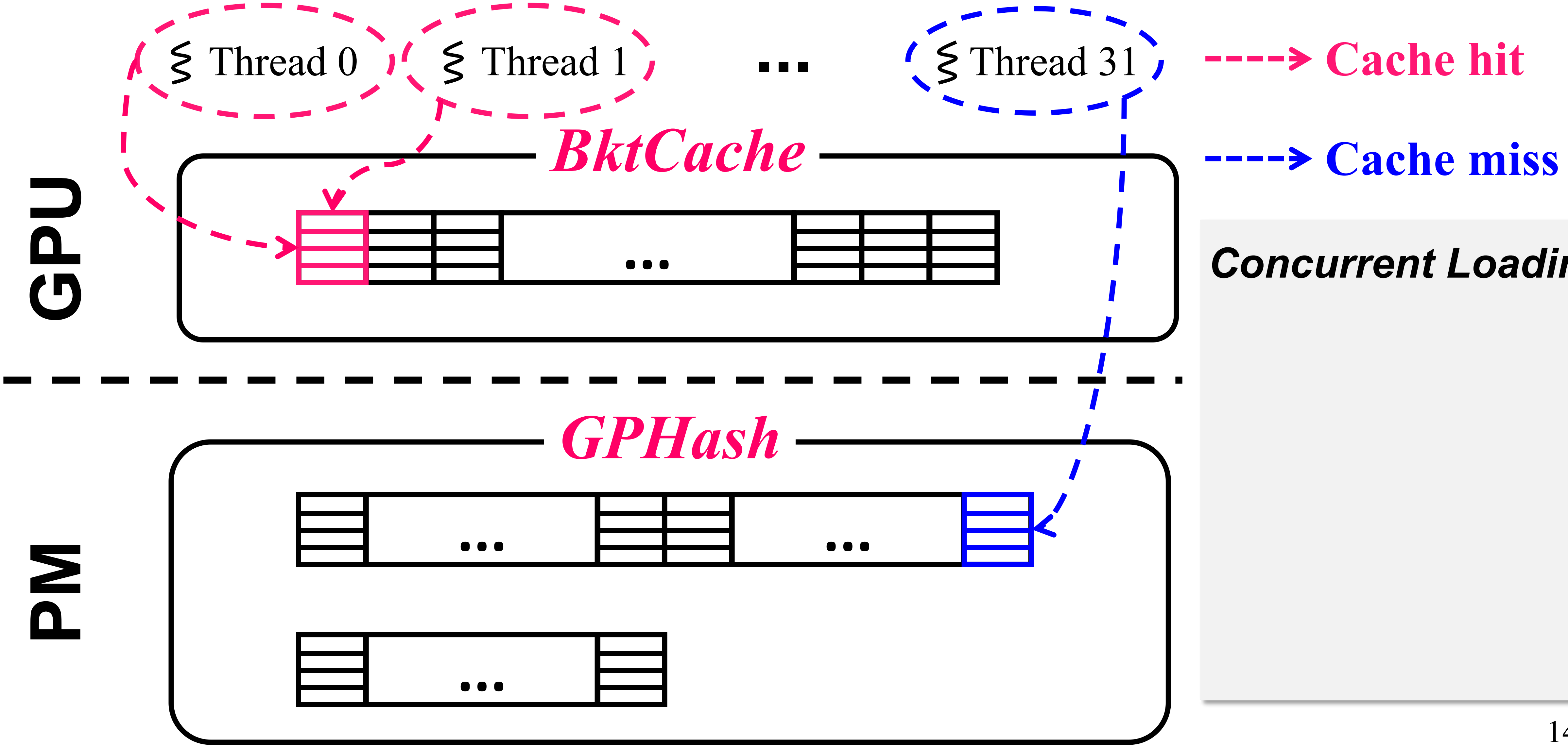
Frozen-Based Bucket Cache



Frozen-Based Bucket Cache

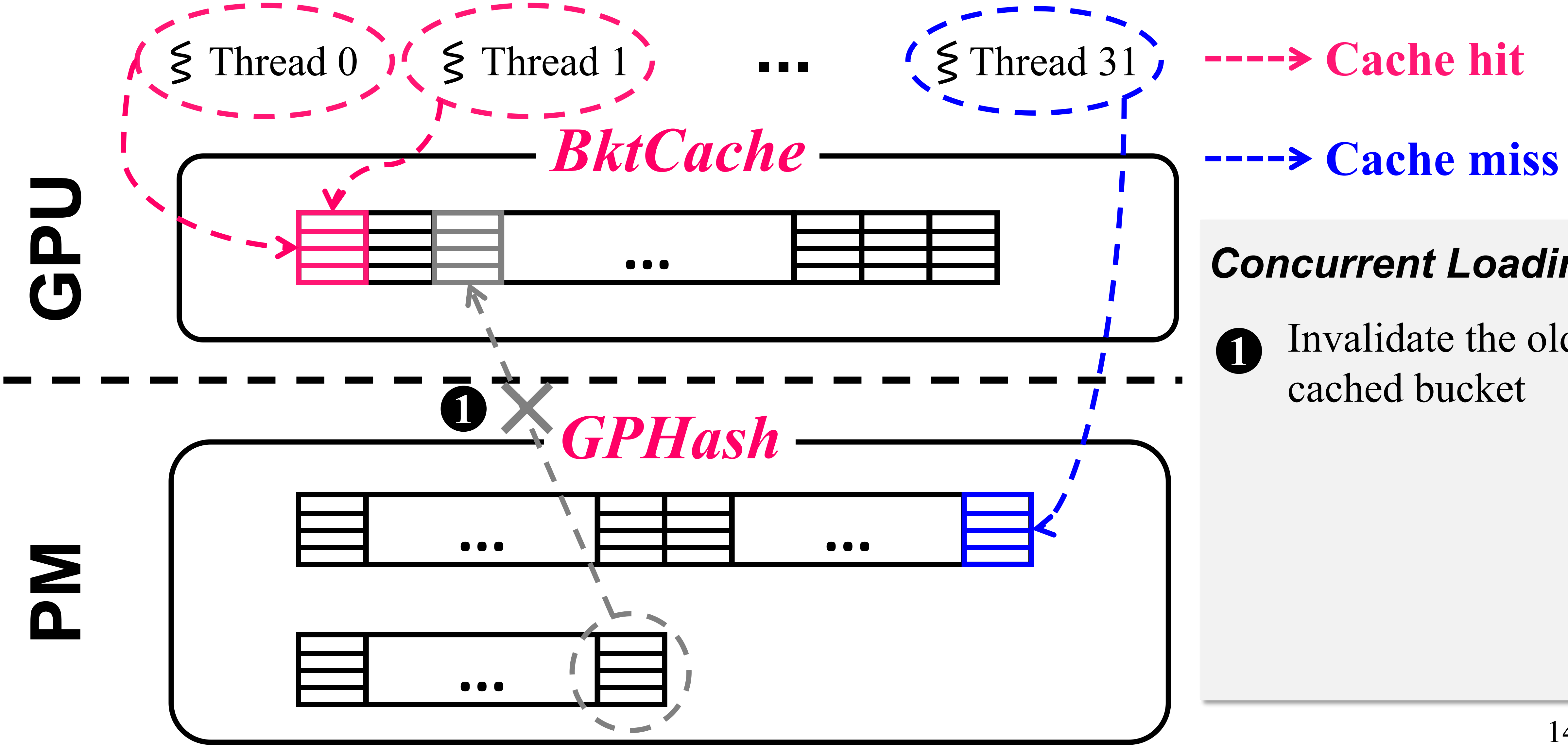


Frozen-Based Bucket Cache



Concurrent Loading

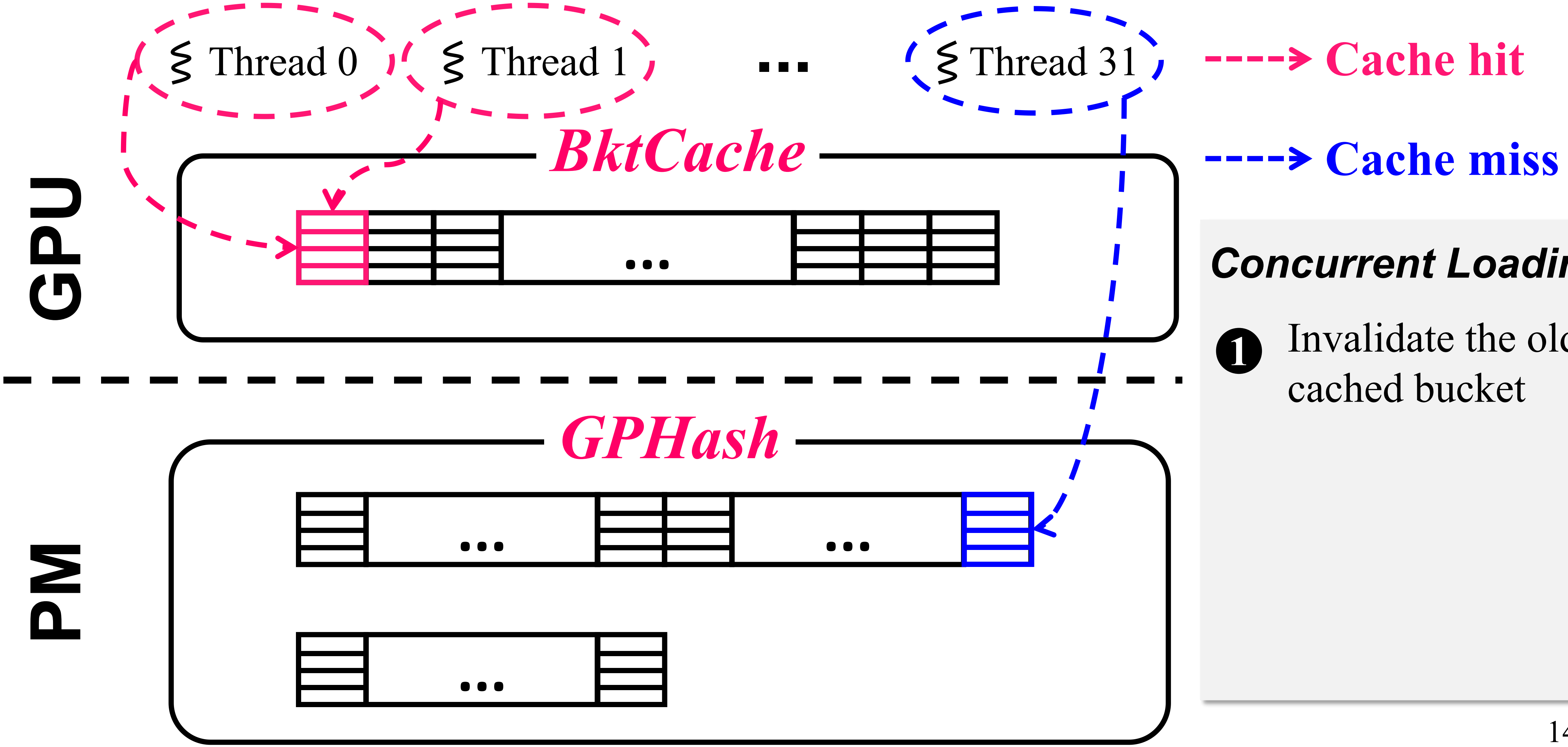
Frozen-Based Bucket Cache



Concurrent Loading

❶ Invalidate the old cached bucket

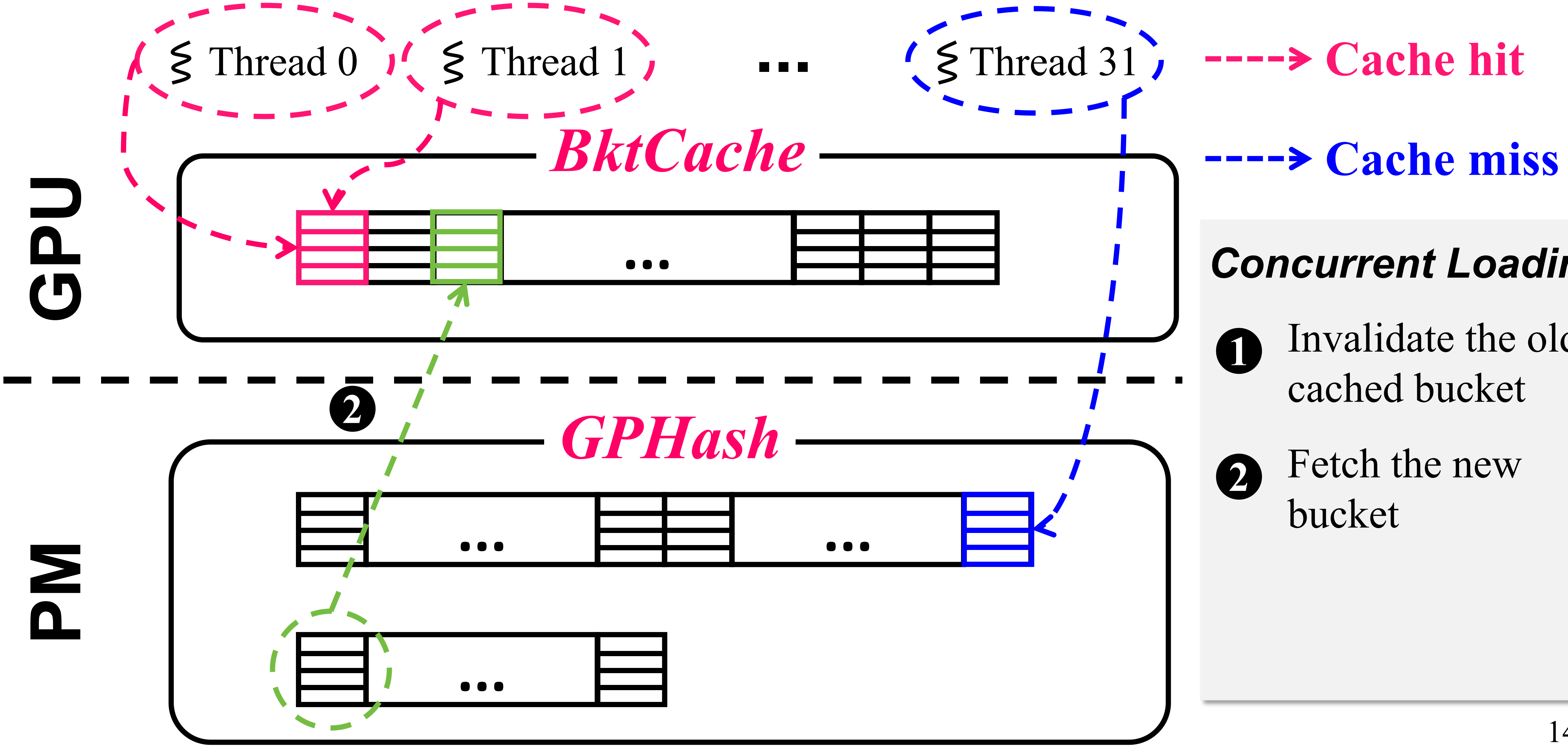
Frozen-Based Bucket Cache



Concurrent Loading

❶ Invalidate the old cached bucket

Frozen-Based Bucket Cache

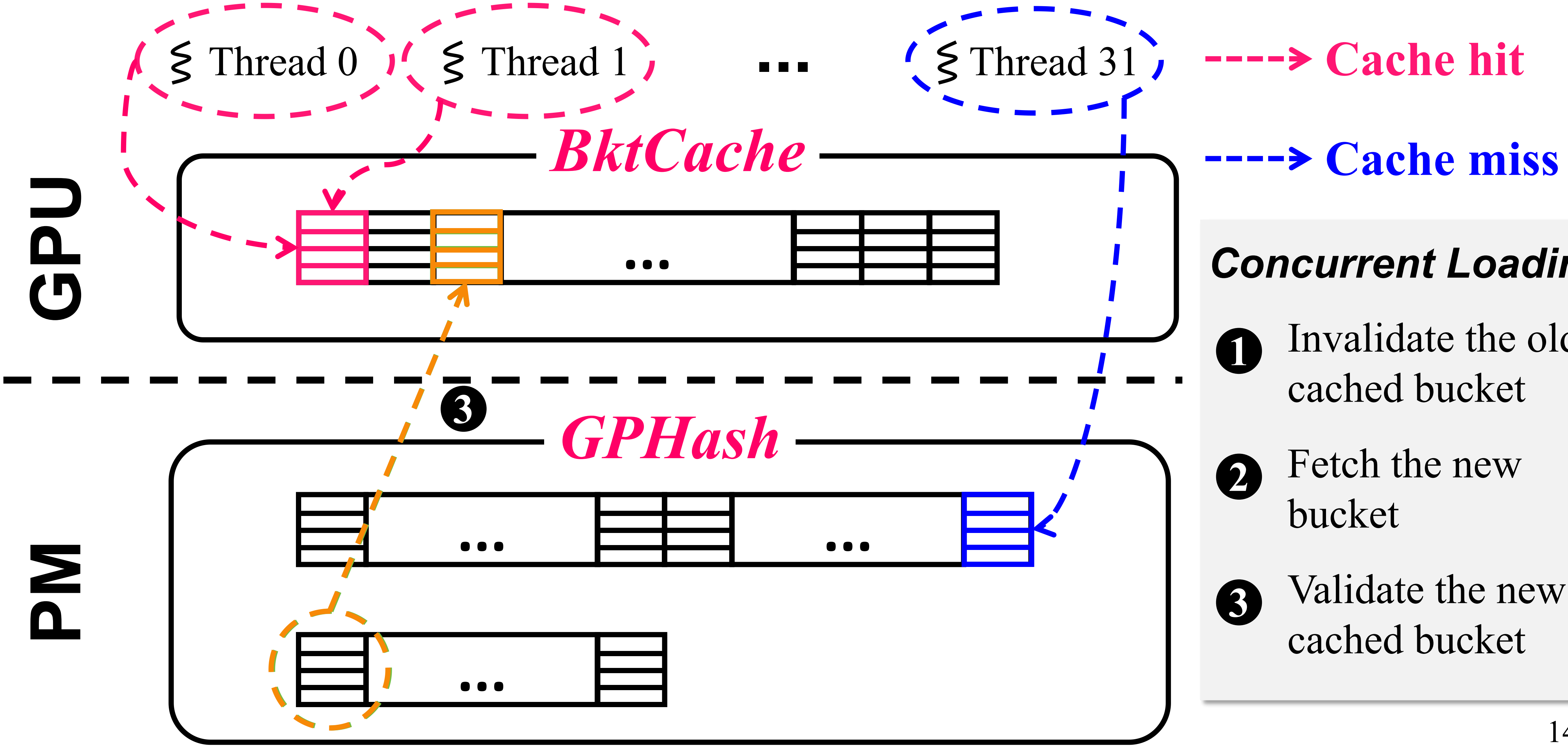


Concurrent Loading

❶ Invalidate the old cached bucket

❷ Fetch the new bucket

Frozen-Based Bucket Cache



Concurrent Loading

1. Invalidate the old cached bucket
2. Fetch the new bucket
3. Validate the new cached bucket

More Details

More Details

- Warp-cooperative Execution Manner
- More Index Operations
- Bucket Caching Granularity
- …

More Details

- Warp-cooperative Execution Manner
- More Index Operations
- Bucket Caching Granularity
- …

More Details

- ➢ Warp-cooperative Execution Manner
- ➢ More Index Operations
- ➢ Bucket Caching Granularity
- ➢ ...

work [78], GPHash determines the *valid* item of a key. Given multiple items of the same key, the valid item is the one having the maximal level number, the minimal bucket number, and the minimal slot number. When finding duplicates, GPHash keeps the valid item and deletes other duplicates.

Concurrency correctness. When threads concurrently perform the search and the IDU (i.e., insertion/deletion/update) operations with the same key, the readers may return the partial-updated value, which violates the concurrency correctness. To ensure concurrency correctness while providing high performance, GPHash follows the *"no lost key"* concurrent correctness condition akin to prior schemes [38, 78]. Specifically, when threads concurrently perform the search and the update operations, the search operations return either the old or the new values instead of partial-updated values. When a search and a deletion run in parallel, the search operation returns either the value or no-key statement.

Crash consistency guarantee. When directly managing data in persistent memory, a crash would interrupt the ongoing index operations, which can lead to persistent partial updates for keys and values. Such data inconsistency causes data loss and unpredictable errors. To guarantee data consistency in the presence of crashes, GPHash uses CAS primitive and the slot state to achieve log-free operations with negligible overhead.

3.2.3 Lock-Free and Log-Free Operations

We introduce the details of lock-free and log-free operations. Here, we focus on operations of the fixed-length large keys whose sizes are larger than 8 bytes, while the operations of fixed-length small keys (i.e., ≤ 8 bytes) and variable-length keys can be implemented in a similar way using the CAS primitive. We use system-scoped threadfence [46] to order the persists for the correct consistency guarantee.

Insertion. Figure 3 illustrates the lock-free and log-free insertions. First, GPHash obtains the fingerprints and the keys of all candidate slots of the activated key with one-shot warp access. GPHash then checks if the key exists by comparing these keys with the activated key, while leveraging the fingerprints for fast comparison. If the activated key does not exist, GPHash finds the empty slots, i.e., the slots whose states are `EMPTY`[2]. If there are several empty slots, GPHash inserts the activated key into the slot belonging to the less-loaded bucket. After deciding the target slot for insertion, the activated thread uses CAS primitive to atomically change the slot state (i.e., fingerprint region) from `EMPTY` to `INSERT`. If the CAS fails, meaning that the slot is changed by another thread, GPHash re-executes the insertion from the beginning. If CAS succeeds, the activated thread writes the item into the target slot. Finally, the activated thread sets the fingerprint region of the target slot to the hash value of the activated key.

The insertion can easily recover from crashes. There are two cases of a slot after crashes. (1) The slot state is `INSERT`, indicating that the slot is under insertion (i.e., writing a new item) before crashes. In this case, the slot may be broken, and thus we need to clear the slot and set the slot state to `EMPTY`. (2) The slot state is not `INSERT`, meaning that the slot is empty or contains an unbroken item. In this case, we do not need to do anything since the slot is already in a valid state.

[2] We reserve two 8-byte values in the fingerprint value range, i.e., `EMPTY` and `INSERT`, to indicate the slot is empty or under insertion.

Figure 3: The illustration of lock-free and log-free insertion *(using the logical structure of a slot for easy understanding).*

Deletion. For deletion operation, GPHash first locates the target items whose keys are equal to the activated key, including duplicate items. Similar to insertion, the activated thread is responsible for atomically deleting all these items by using the CAS primitive to set the slot states to `EMPTY`. Thanks to the atomicity of the CAS primitive, the deletion does not introduce any invalid slot state in the presence of crashes.

Update. For the update operation in GPHash, after locating the target slot and deleting the other duplicates, the activated thread atomically changes the value pointer to point to the new value via the CAS primitive. GPHash writes the new value to the pre-allocated space before updating the value pointer. After crashes, the value pointer either points to the old value or the new one, both of which are unbroken.

Search. Since GPHash takes advantage of the atomicity of the CAS primitive to perform the IDU operations, the lock-free search operation can be easily implemented. After locating all slots whose keys are equal to the activated key, the activated thread reads the value that is pointed by the value pointer of the valid slot. If the activated key does not exist, the thread returns a no-key statement. Based on the above introduction to other operations, the search operation can be proved to meet the *"no lost key"* concurrent correctness condition.

Resizing. As the load factor increases, more hash collisions will occur in hash indexes, which results in performance degradation and insertion failure. Thanks to the one-shot warp access, GPHash does not suffer from performance degradation caused by more hash collisions. However, GPHash still needs to handle insertion failure to avoid item loss. If failing to find an empty slot to insert a new item, GPHash has to resize. Specifically, GPHash first allocates a new level as the new top one. GPHash then leverages thousands of GPU threads to scan the bottom level in parallel and rehashes the items. Each rehashing operation consists of reading the item in the

Evaluation

Evaluation

➢ Platform

- 1 V100 GPU + 768 GB Intel Optane DC PM (6 × 128 GB)

Evaluation

- **Platform**
  - 1 V100 GPU + 768 GB Intel Optane DC PM (6 × 128 GB)
- **Comparisons**
  - CPU-assisted approaches[1]: *Clevel[ATC'20] , Dash[VLDB'20], and SEPH[OSDI'23]*
  - GPM-enabled approaches: *Clevel-GPM and SlabHash[IPDPS'18]-GPM*

[1] For fair evaluation, we use PM to store the data and leverage PM hash indexes to manage the data in PM

Evaluation

➢ Platform

- 1 V100 GPU + 768 GB Intel Optane DC PM (6 × 128 GB)

➢ Comparisons

- CPU-assisted approaches[1]: *Clevel[ATC'20] , Dash[VLDB'20], and SEPH[OSDI'23]*
- GPM-enabled approaches: *Clevel-GPM and SlabHash[IPDPS'18]-GPM*

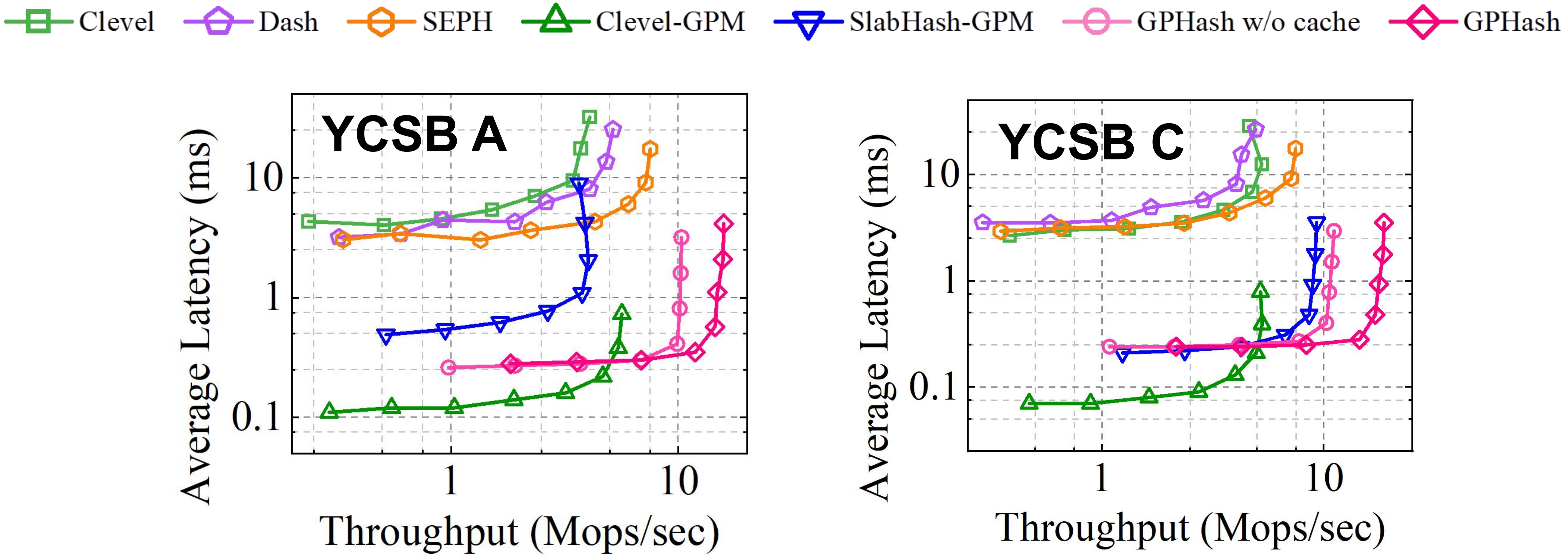
➢ Workloads

- YCSB workloads: *8-byte and 32-byte keys, 128-byte values*
- Real-world workloads: *DLRM and PageRank*

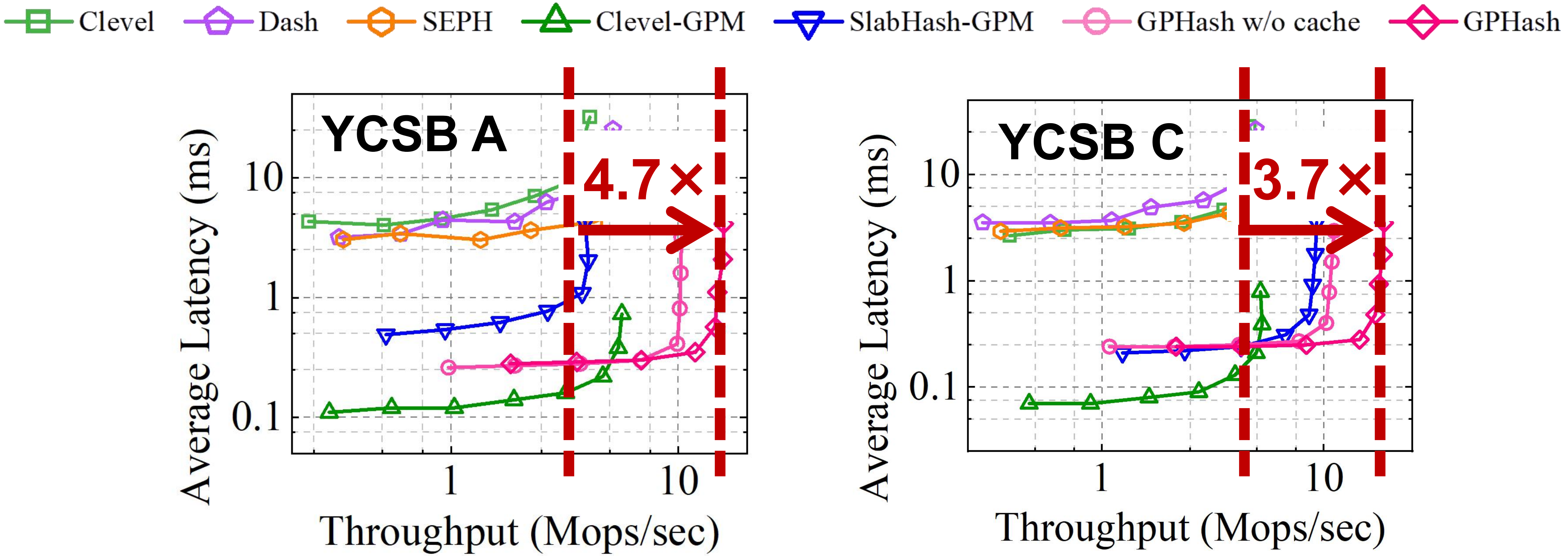
[1] For fair evaluation, we use PM to store the data and leverage PM hash indexes to manage the data in PM

End-to-End Performance

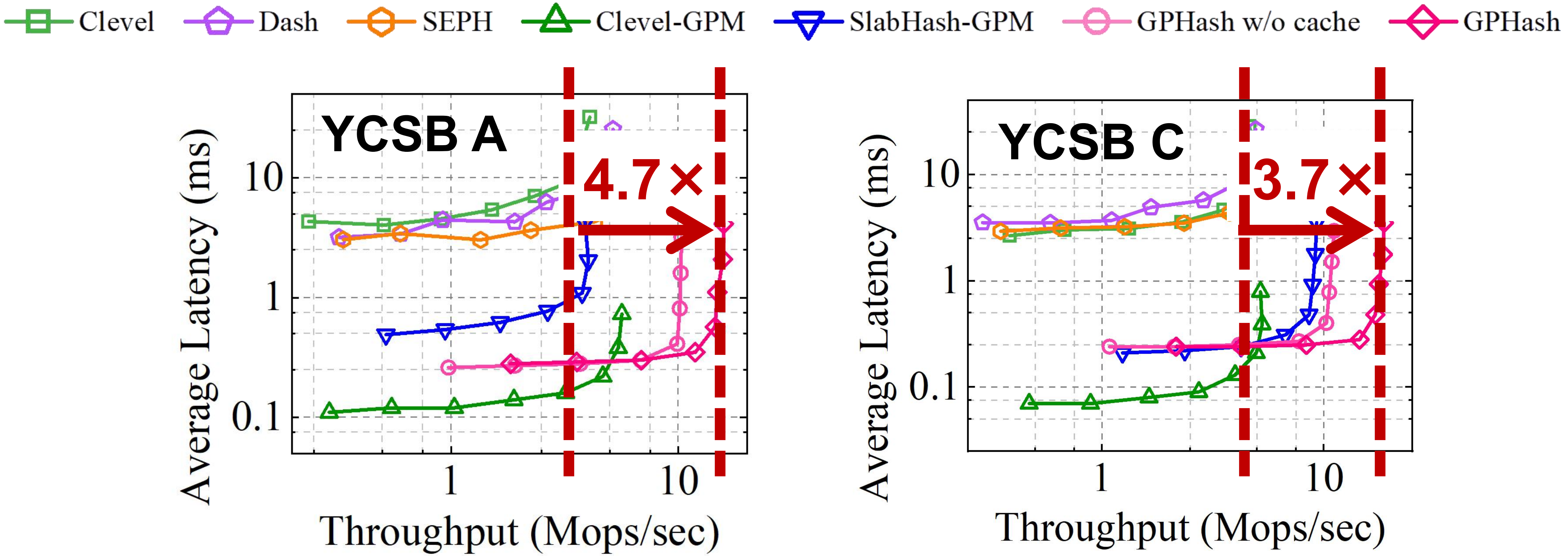
End-to-End Performance



End-to-End Performance



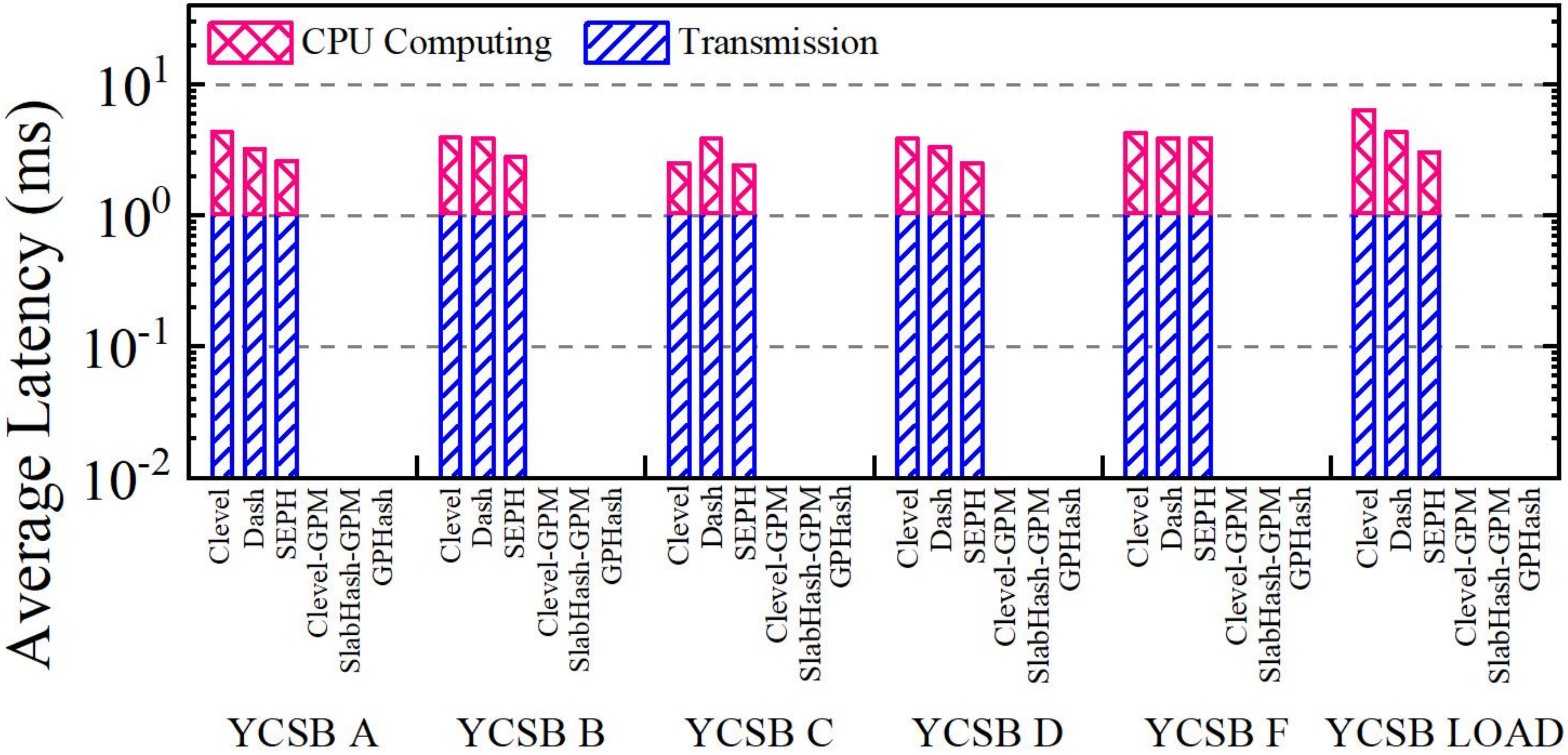
End-to-End Performance



GPHash improves the throughput by **1.9~6.3×**

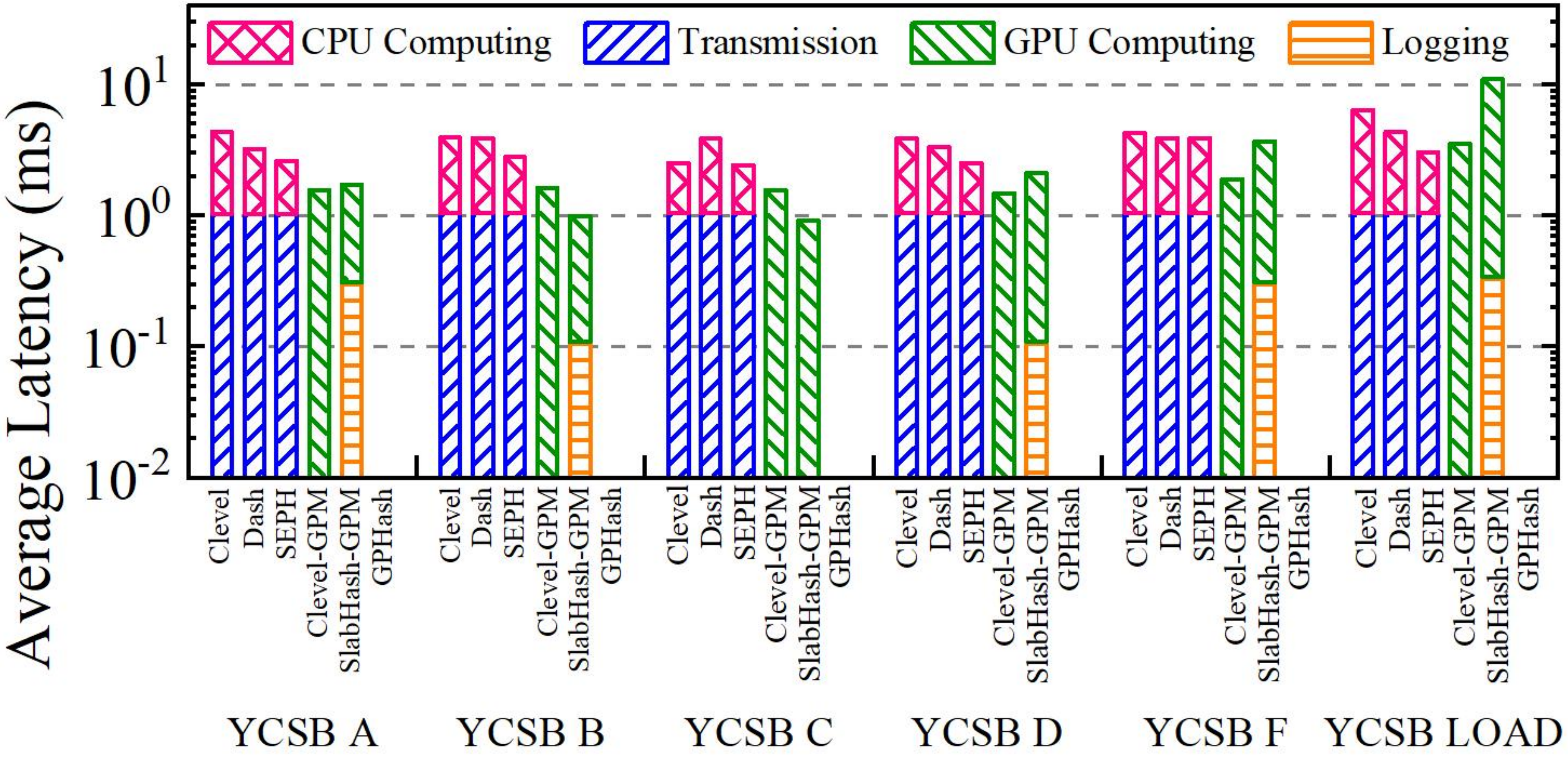
Latency Breakdown

Latency Breakdown



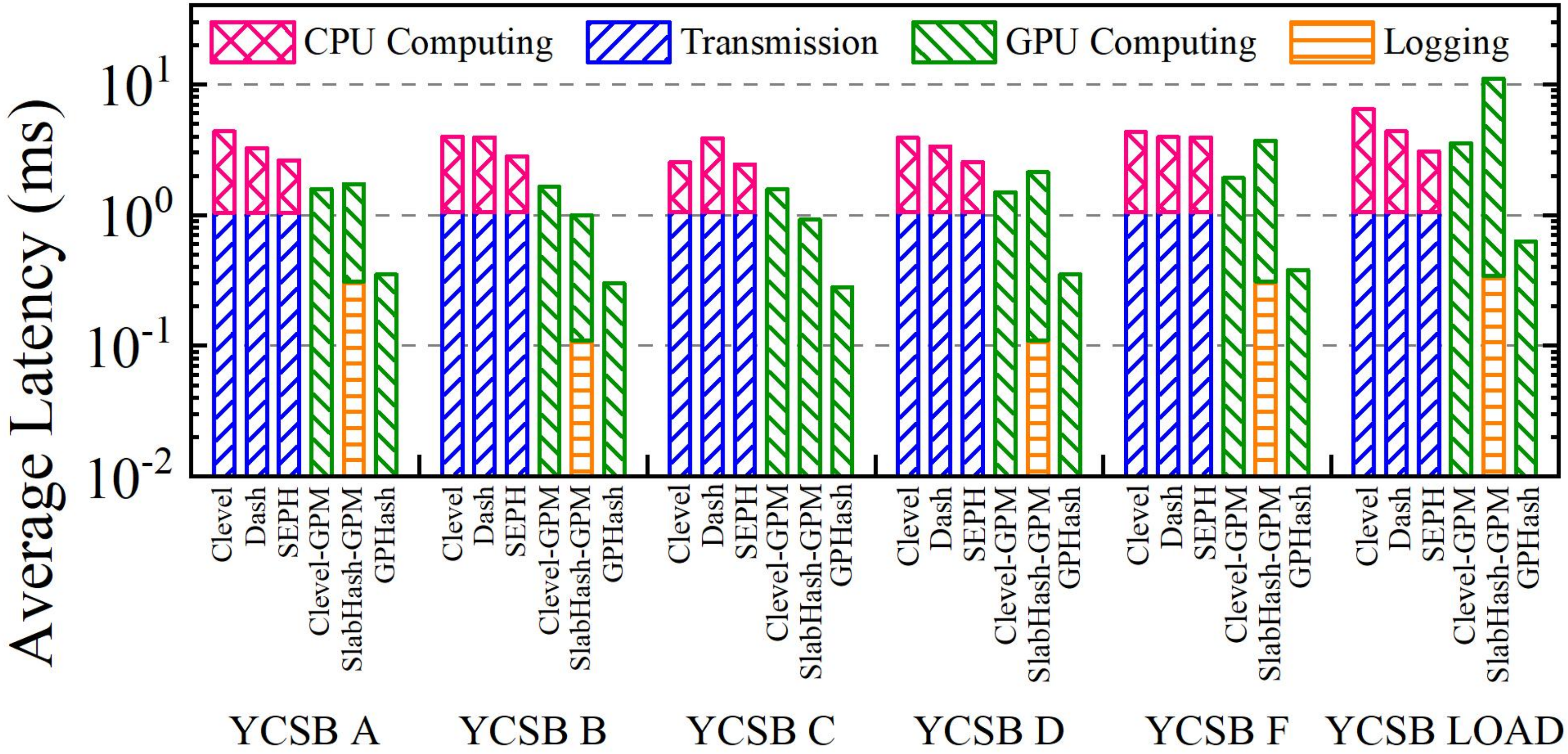
CPU-assisted approaches suffer from ***high transmission cost***

Latency Breakdown



Naive GPM-enabled approaches suffer from ***severe warp divergence*** and ***high-overhead consistency guarantee***

Latency Breakdown



GPHash fully leverages the ***high parallelism of GPU*** and provides a ***low-overhead consistency guarantee***

Conclusion

Conclusion

- Inefficient GPU data management on large-scale data
  - ***High overhead for data transfer***
  - ***Extra CPU consumption***

Conclusion

- Inefficient GPU data management on large-scale data
  - ***High overhead for data transfer***
  - ***Extra CPU consumption***
- **GPHash:** an efficient GPM-enabled hash index
  - ***GPU-conscious and PM-friendly hash table***
  - ***Lock-free and log-free operations***
  - ***Frozen-based bucket cache***

Conclusion

https://github.com/LighT-chenml/GPHash

chenml@hust.edu.cn

- Inefficient GPU data management on large-scale data
  - ***High overhead for data transfer***
  - ***Extra CPU consumption***
- **GPHash:** an efficient GPM-enabled hash index
  - ***GPU-conscious and PM-friendly hash table***
  - ***Lock-free and log-free operations***
  - ***Frozen-based bucket cache***

Conclusion

https://github.com/LighT-chenml/GPHash

chenml@hust.edu.cn

- Inefficient GPU data management on large-scale data
  - ***High overhead for data transfer***
  - ***Extra CPU consumption***
- **GPHash:** an efficient GPM-enabled hash index
  - ***GPU-conscious and PM-friendly hash table***
  - ***Lock-free and log-free operations***
  - ***Frozen-based bucket cache***

Thank you! Q&A