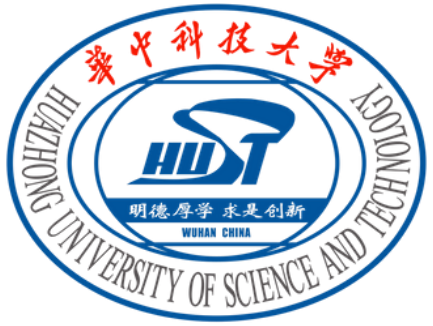


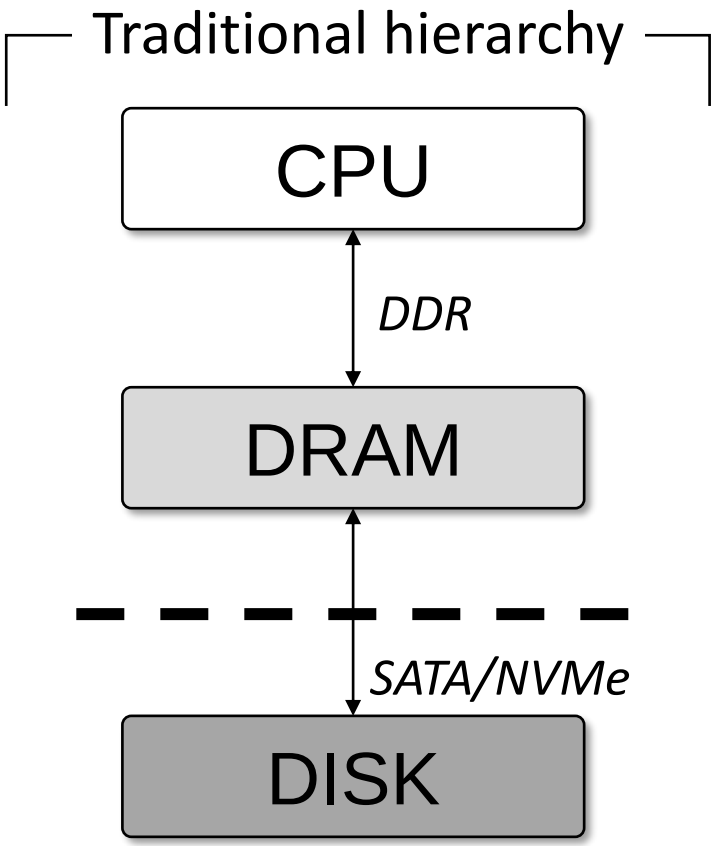
Silo: Speculative Hardware Logging for Atomic Durability in Persistent Memory

Ming Zhang, Yu Hua

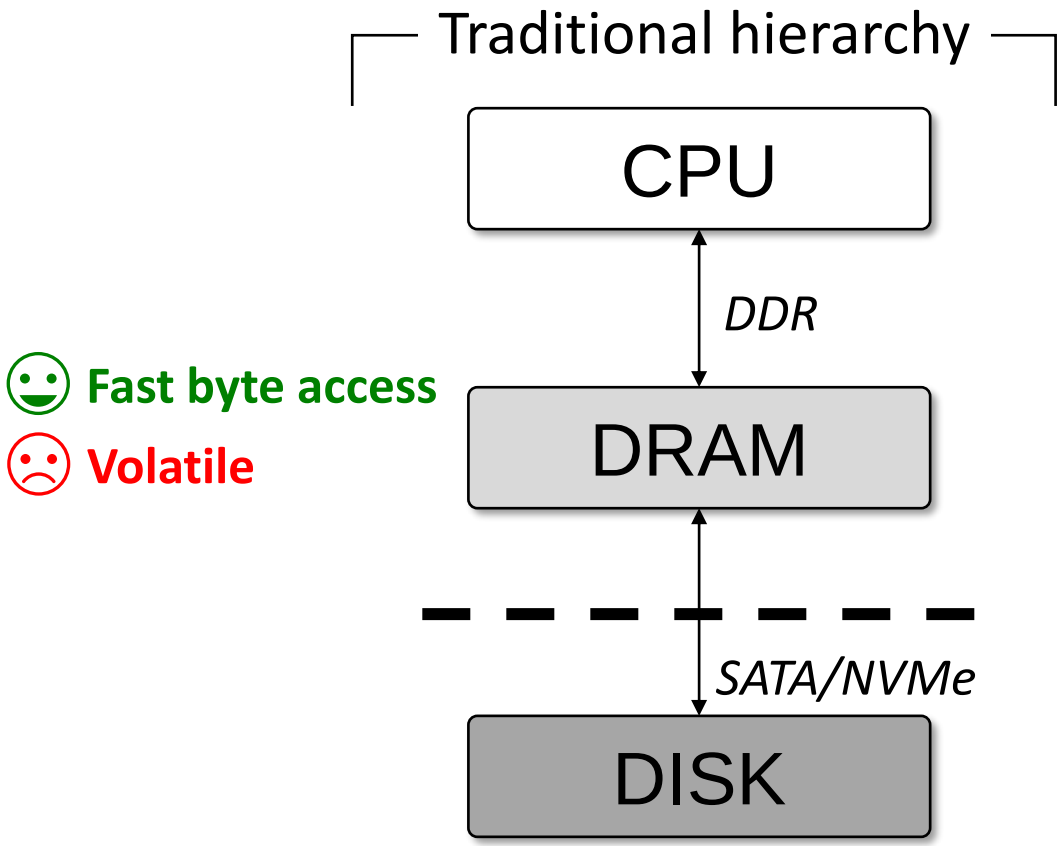
Huazhong University of Science and Technology, China

29th IEEE International Symposium on High-Performance Computer Architecture (**HPCA**), 2023

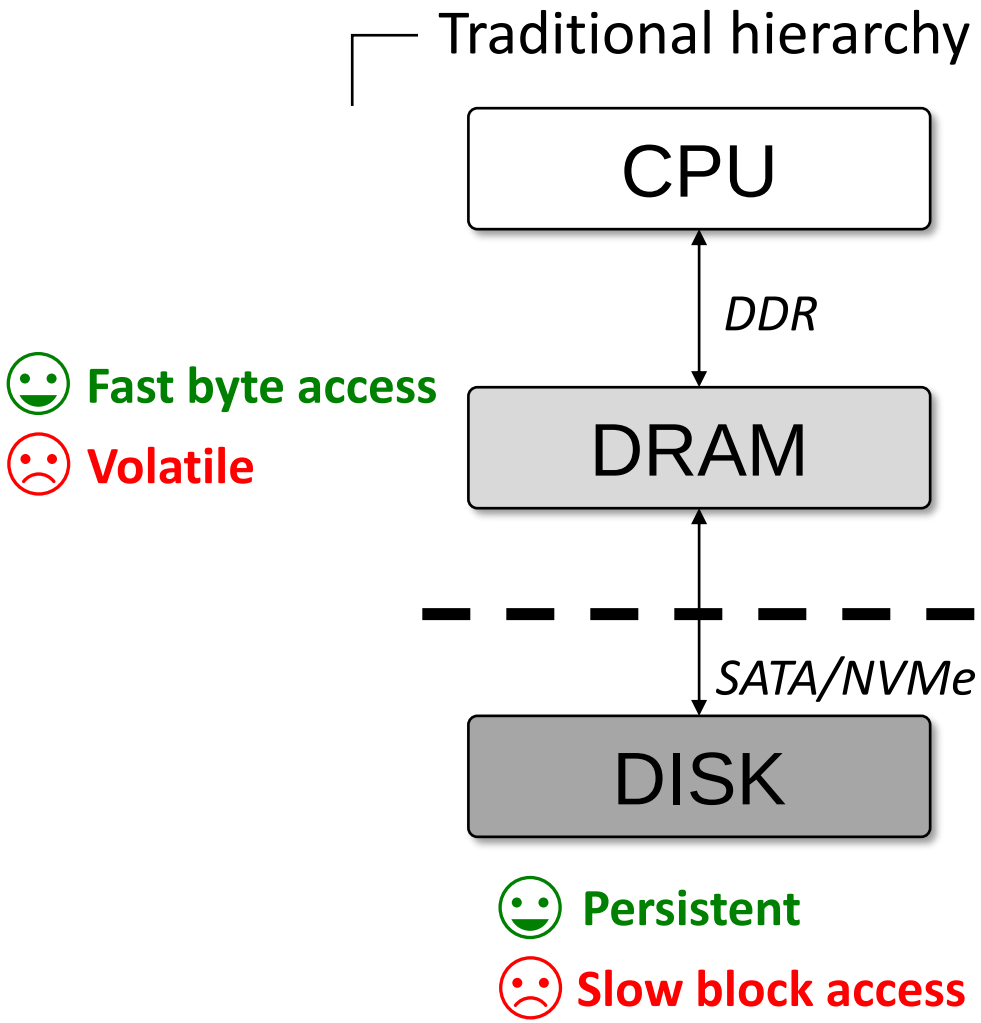
Persistent Memory (PM)



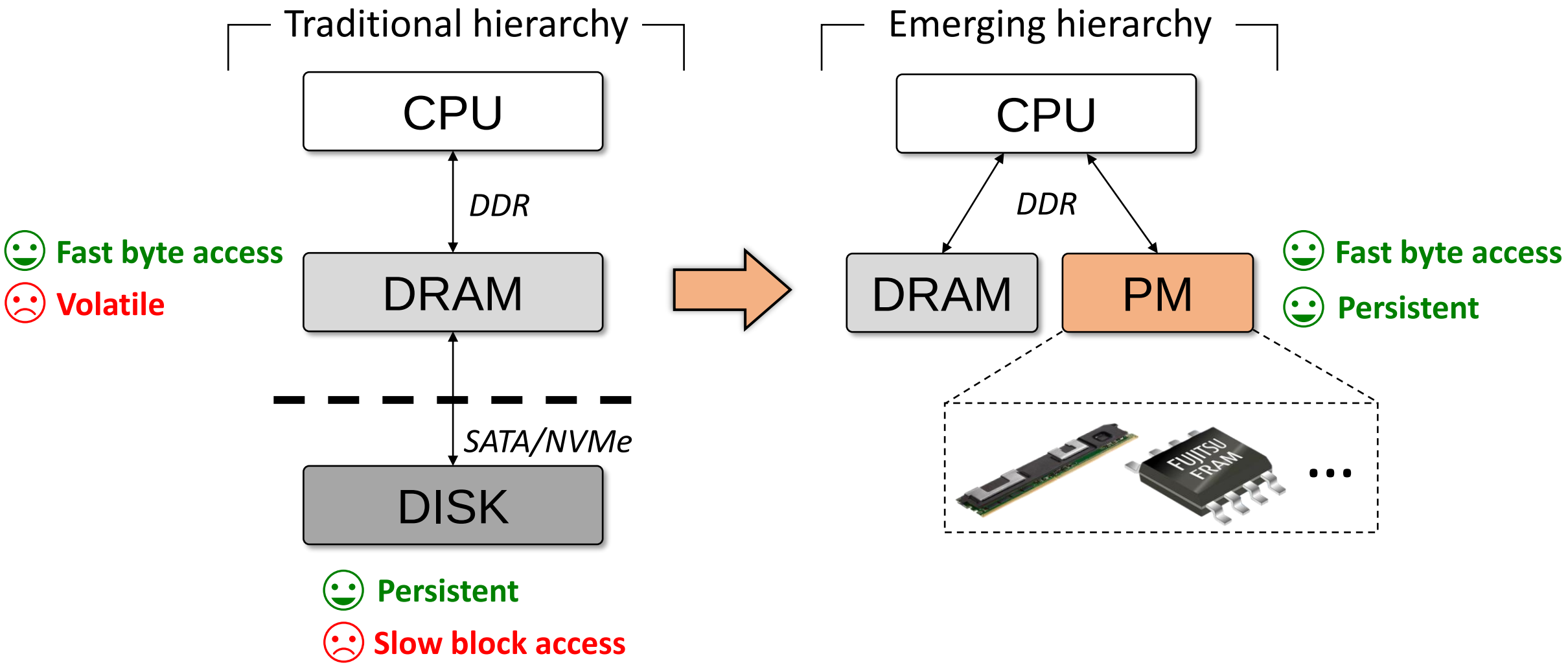
Persistent Memory (PM)



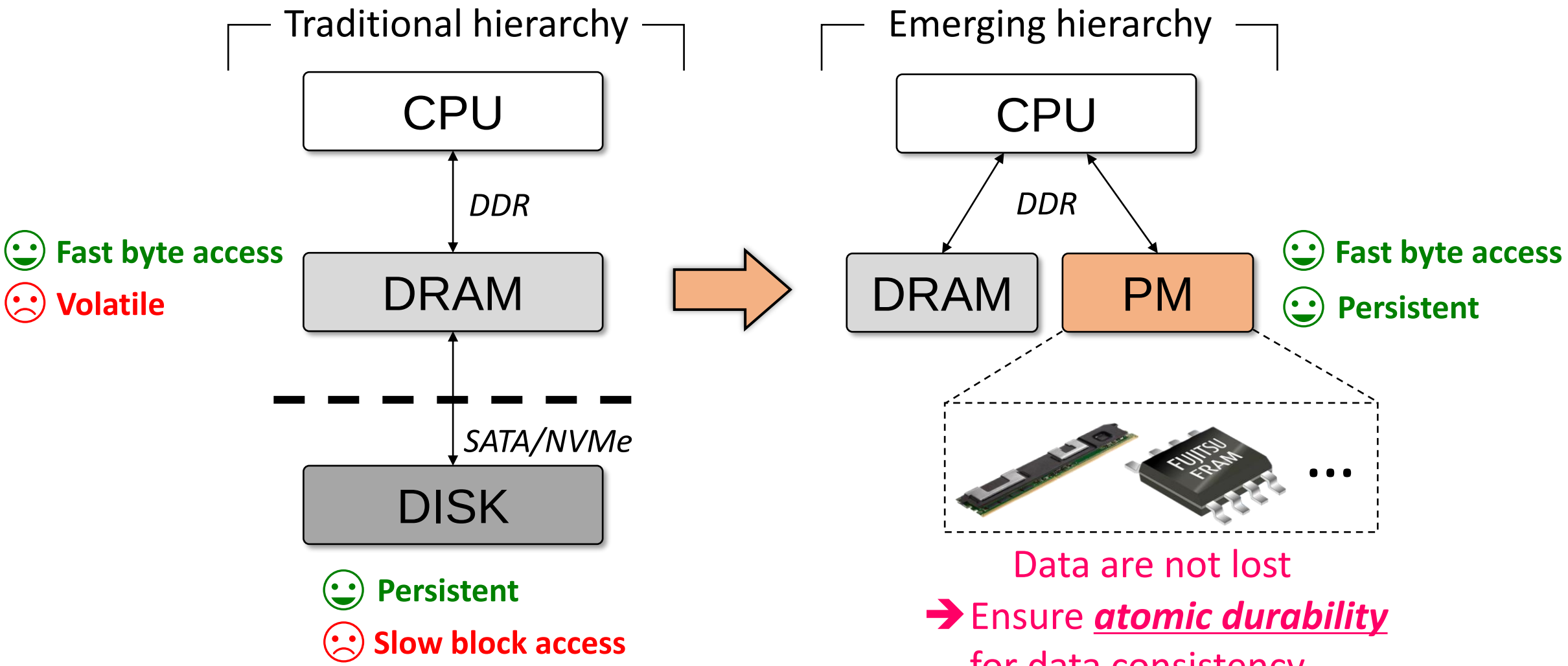
Persistent Memory (PM)



Persistent Memory (PM)



Persistent Memory (PM)



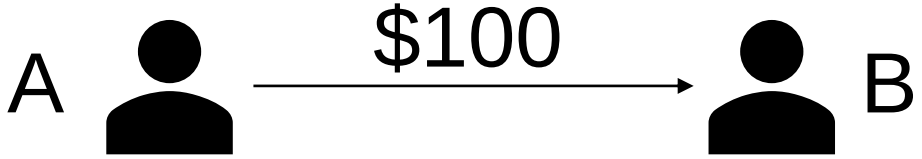
Atomic Durability

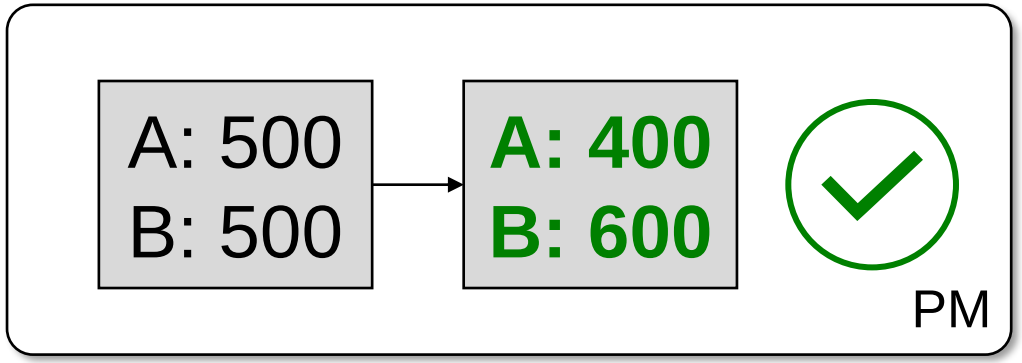
- A group of updates are written to PM in an ***all or nothing*** manner
- Current 64-bit CPUs only support 8B atomic write[1-3]

[1] Fast&Fair@FAST'18 [2] Level Hashing@OSDI'18 [3] Recipe@SOSP'19

Atomic Durability

- A group of updates are written to PM in an ***all or nothing*** manner
- Current 64-bit CPUs only support 8B atomic write[1-3]

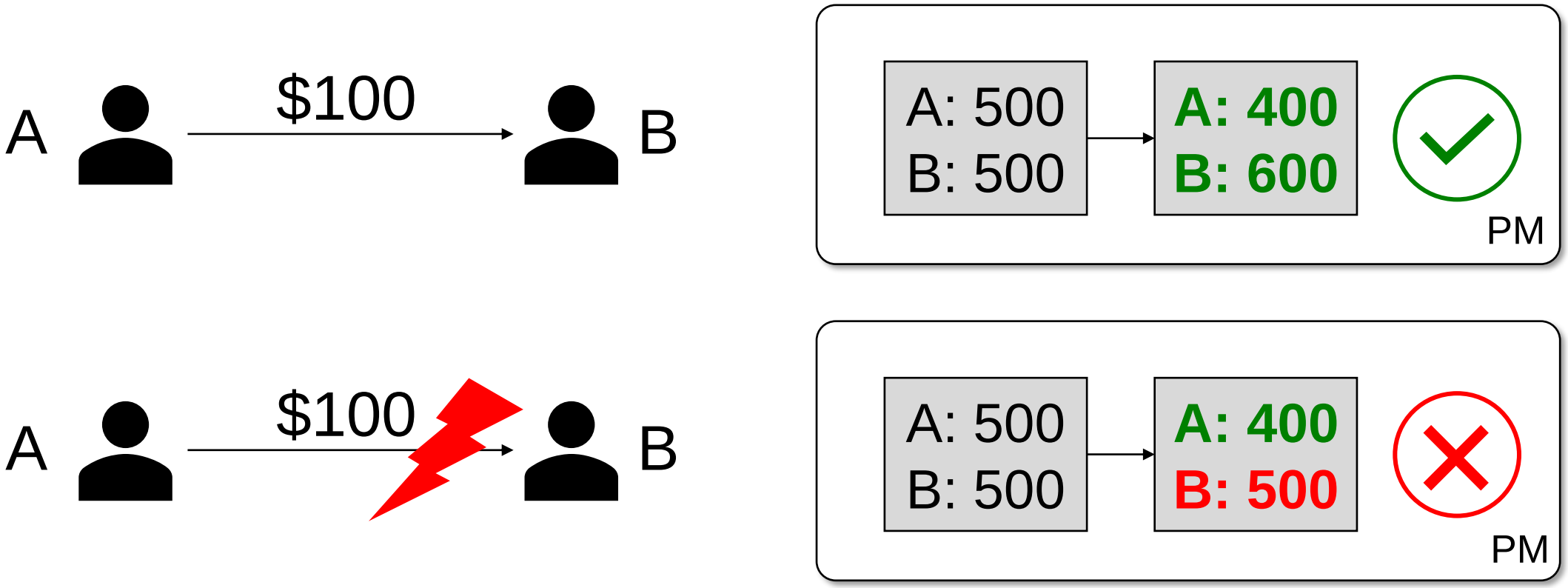




[1] Fast&Fair@FAST'18 [2] Level Hashing@OSDI'18 [3] Recipe@SOSP'19

Atomic Durability

- A group of updates are written to PM in an ***all or nothing*** manner
- Current 64-bit CPUs only support 8B atomic write[1-3]

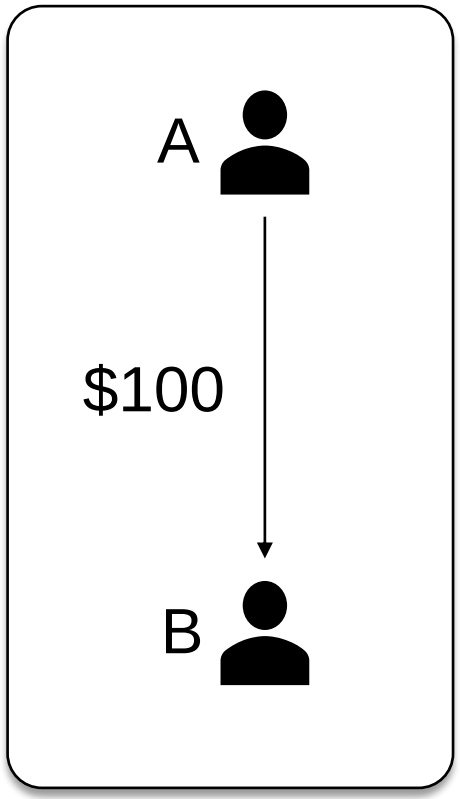


Partial updates!

[1] Fast&Fair@FAST'18 [2] Level Hashing@OSDI'18 [3] Recipe@SOSP'19

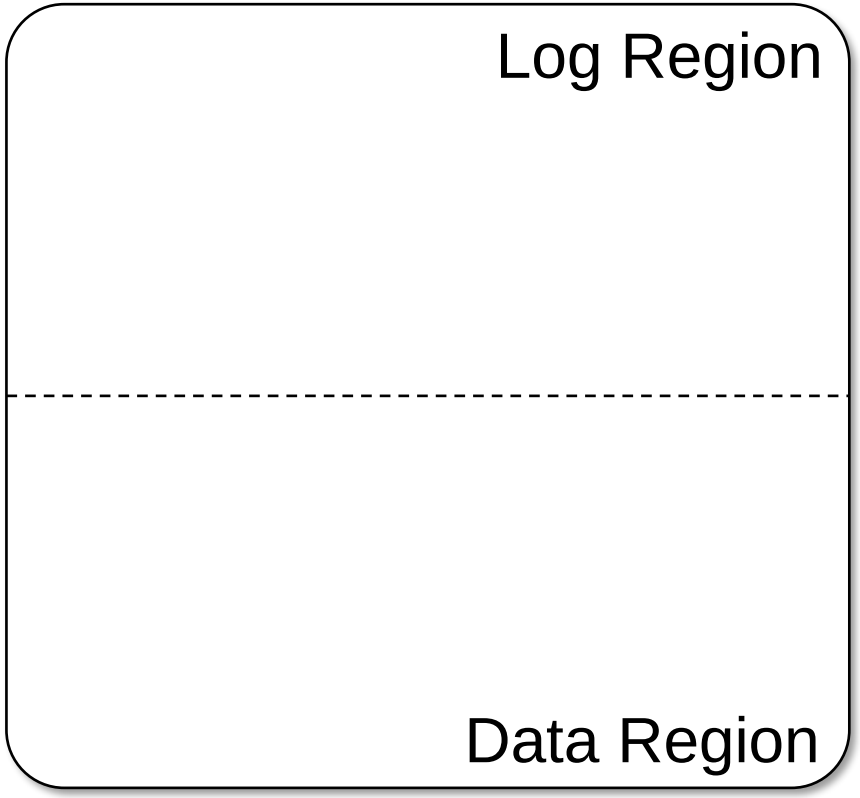
Write-Ahead Logging in Transaction

- Back-up data before updates to ensure atomic durability



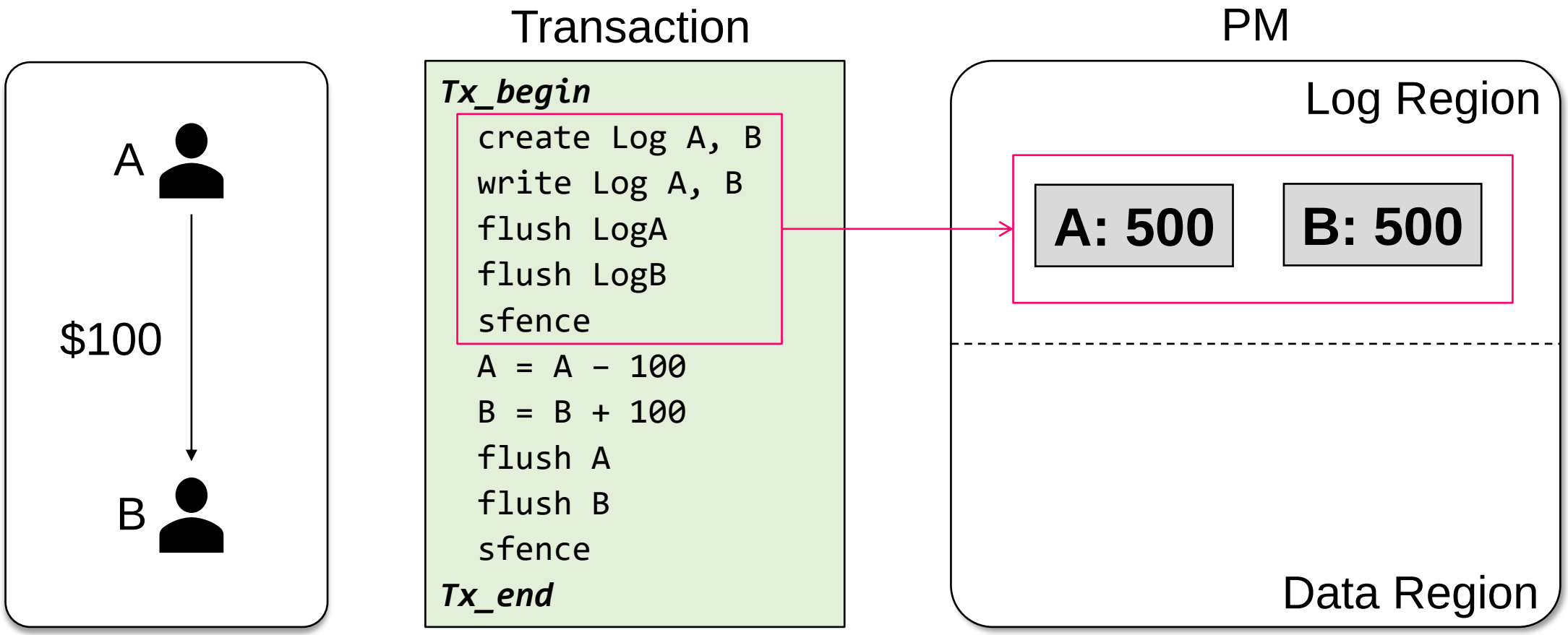
Transaction

```
Tx_begin
  create Log A, B
  write Log A, B
  flush LogA
  flush LogB
  sfence
  A = A – 100
  B = B + 100
  flush A
  flush B
  sfence
Tx_end
```



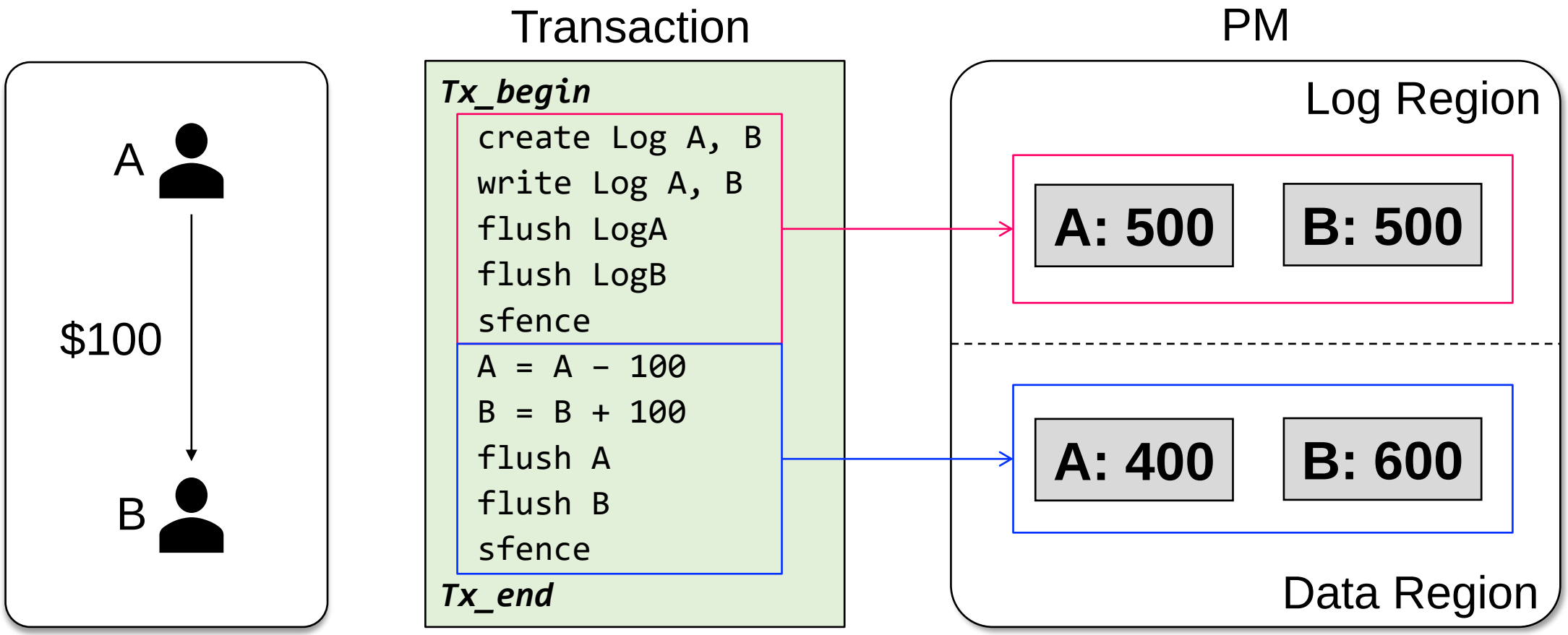
Write-Ahead Logging in Transaction

- Back-up data before updates to ensure atomic durability



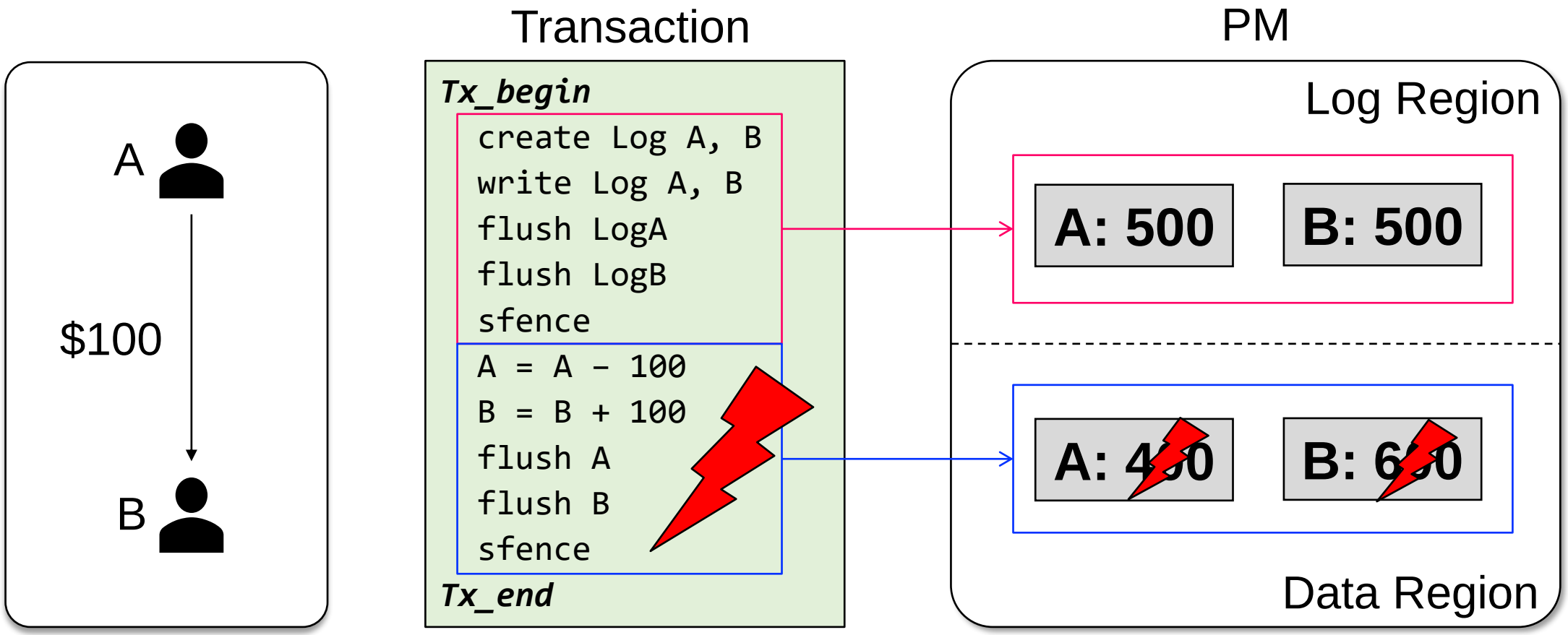
Write-Ahead Logging in Transaction

- Back-up data before updates to ensure atomic durability



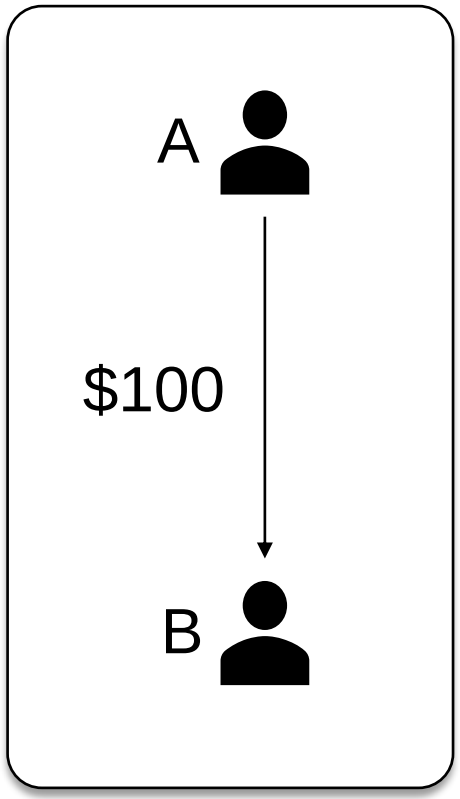
Write-Ahead Logging in Transaction

- Back-up data before updates to ensure atomic durability



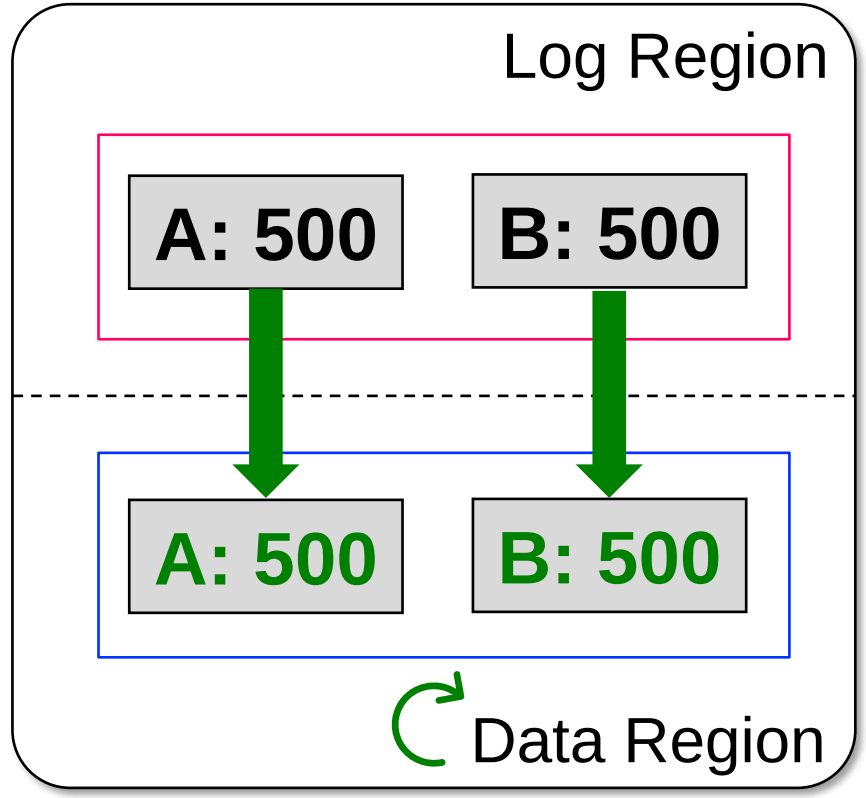
Write-Ahead Logging in Transaction

- Back-up data before updates to ensure atomic durability



Transaction

```
Tx_begin
  create Log A, B
  write Log A, B
  flush LogA
  flush LogB
  sfence
  A = A – 100
  B = B + 100
  flush A
  flush B
  sfence
Tx_end
```



Hardware Logging

Hardware Logging

Software Logging

```
Tx_begin
  create Log
  write Log
  flush Log
  sfence
  write data
  flush data
  sfence
Tx_end
```

Log operations exist on the critical path

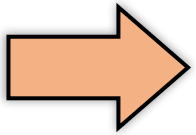
Throughput decreases by up to **70%**[1]

[1] ATOM@HPCA'17

Hardware Logging

Software Logging

```
Tx_begin
  create Log
  write Log
  flush Log
  sfence
  write data
  flush data
  sfence
Tx_end
```



Hardware Logging

```
Tx_begin
  write data
Tx_end
```

Log operations exist on the critical path

Throughput decreases by up to **70%**[1]

[1] ATOM@HPCA'17

Hardware Logging

Software Logging

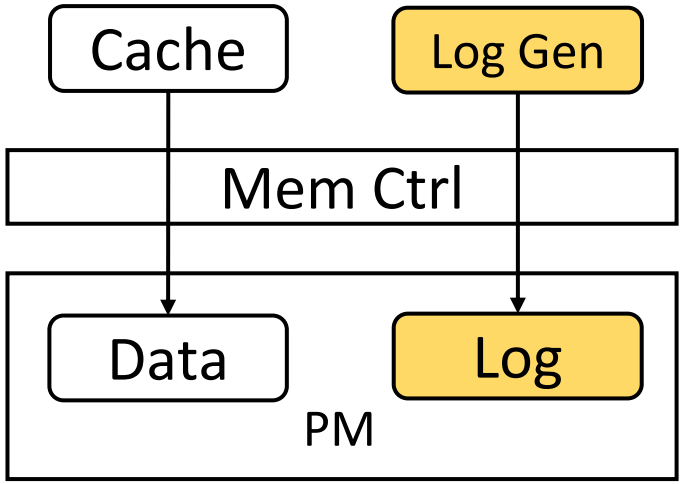
```
Tx_begin
  create Log
  write Log
  flush Log
  sfence
  write data
  flush data
  sfence
Tx_end
```

Log operations exist on the critical path

Throughput decreases by up to **70%**[1]

Hardware Logging

```
Tx_begin
  write data
Tx_end
```



Offload logging operations to hardware

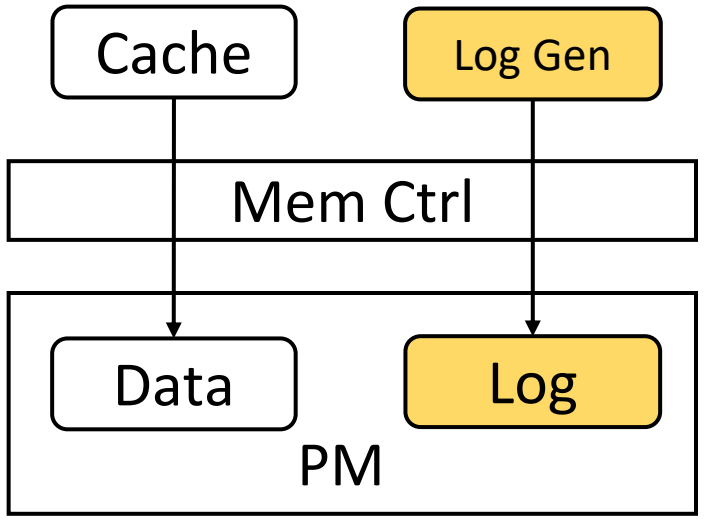
✓ Better performance

✓ Easy programming

[1] ATOM@HPCA'17

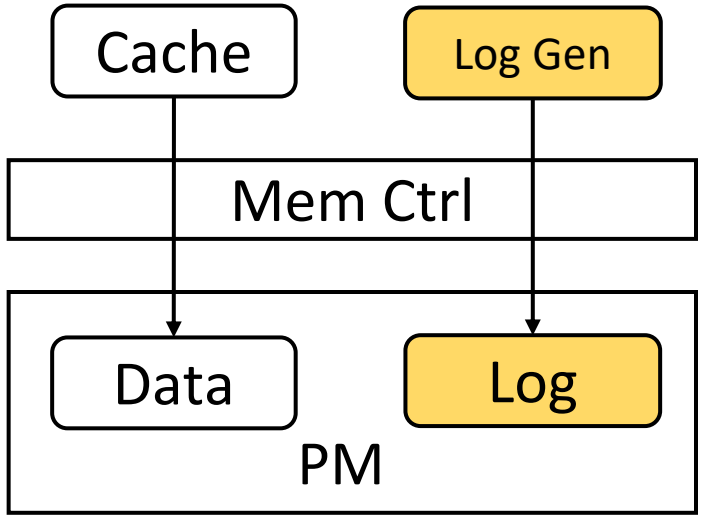
State-of-The-Art

State-of-The-Art

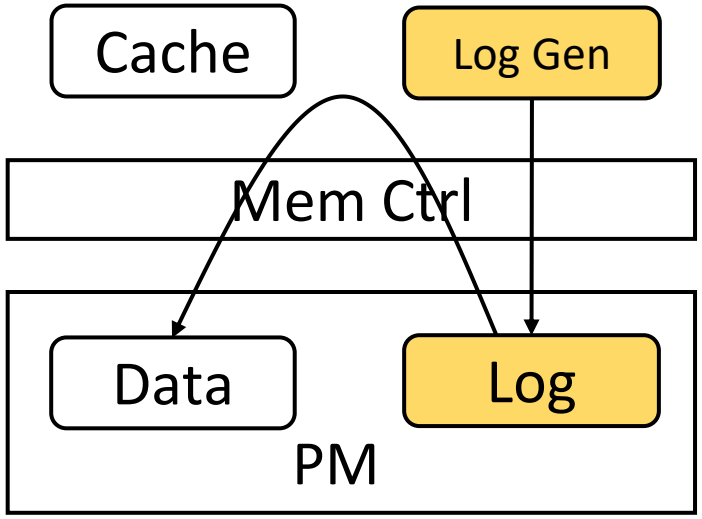


ATOM@HPCA'17
FWB@HPCA'18
MorLog@ISCA'20
ASAP@ISCA'22

State-of-The-Art

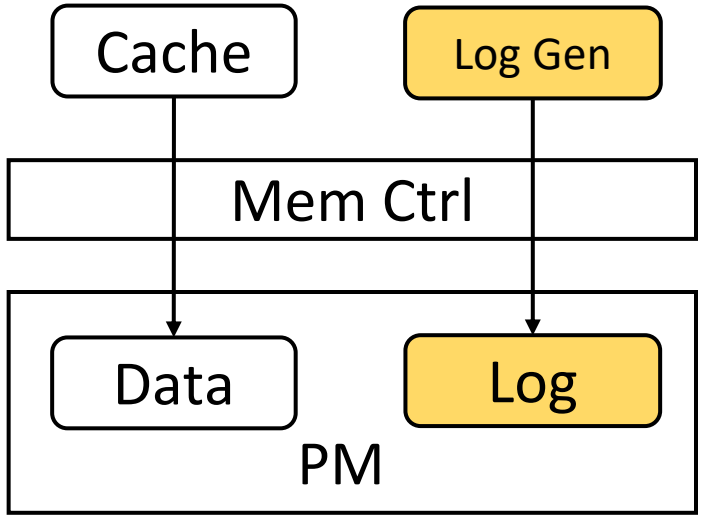


ATOM@HPCA'17
FWB@HPCA'18
MorLog@ISCA'20
ASAP@ISCA'22

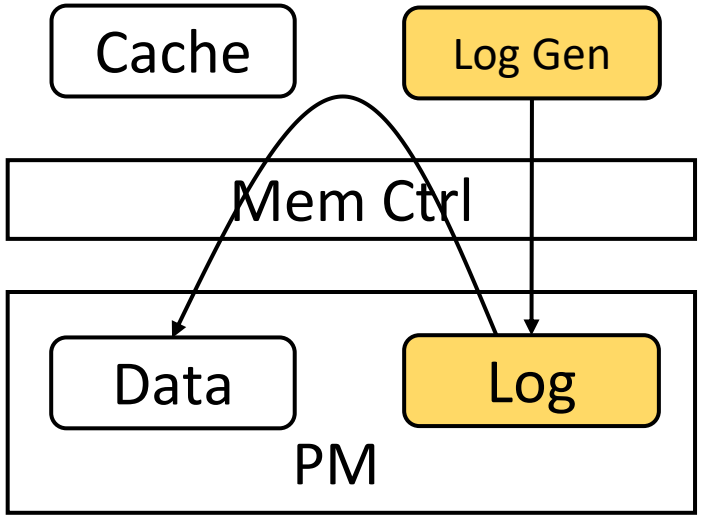


WrAP@HPCA'16

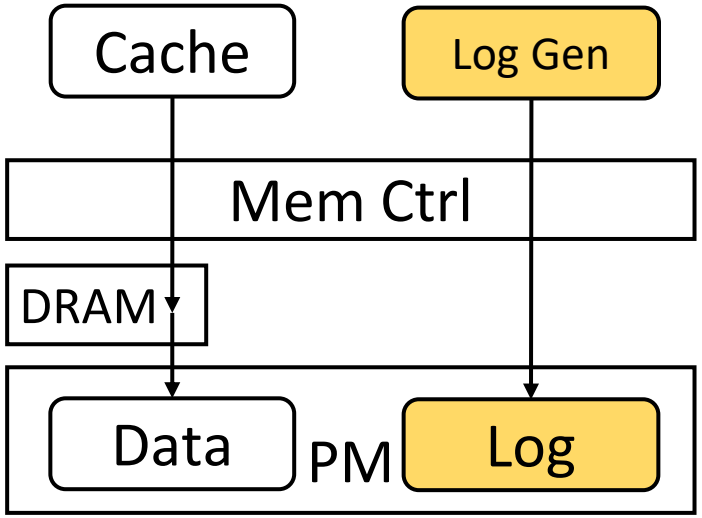
State-of-The-Art



ATOM@HPCA'17
FWB@HPCA'18
MorLog@ISCA'20
ASAP@ISCA'22

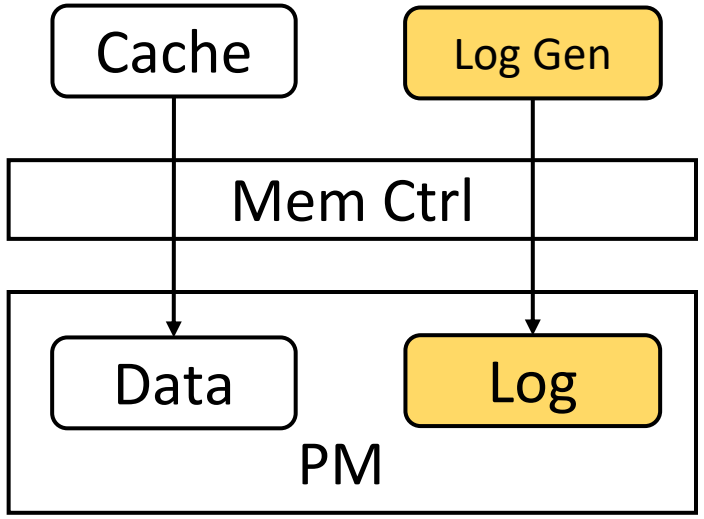


WrAP@HPCA'16

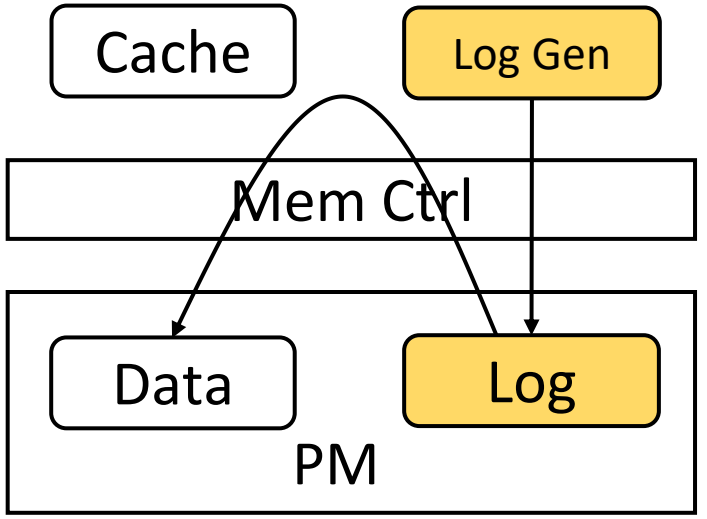


ReDU@MICRO'18

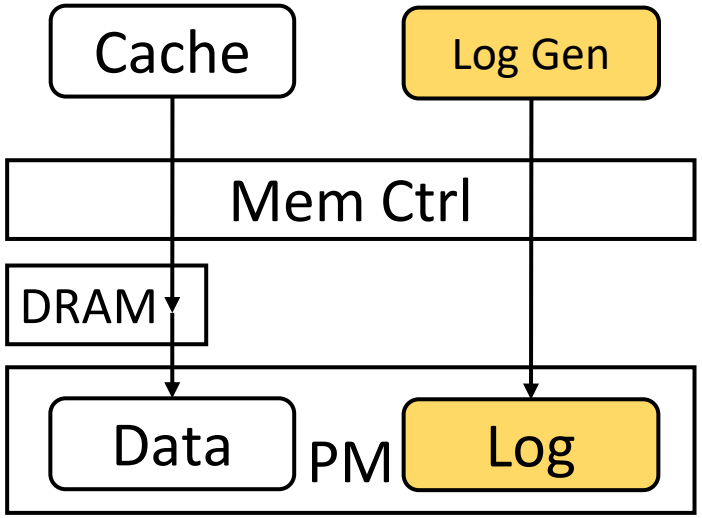
State-of-The-Art



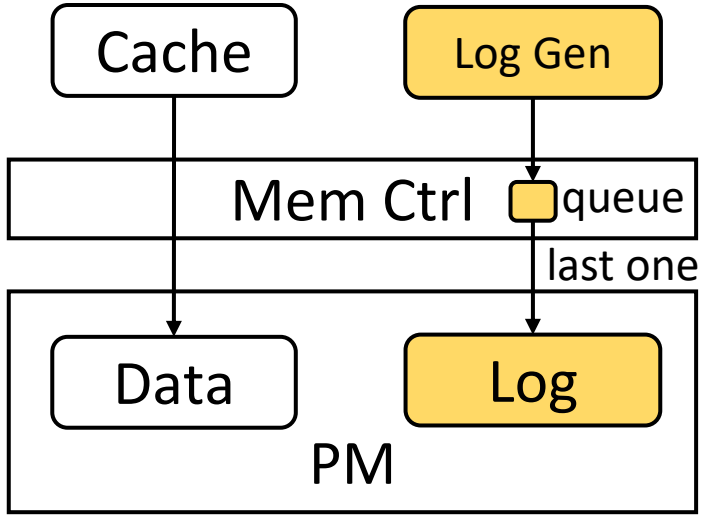
ATOM@HPCA'17
FWB@HPCA'18
MorLog@ISCA'20
ASAP@ISCA'22



WrAP@HPCA'16



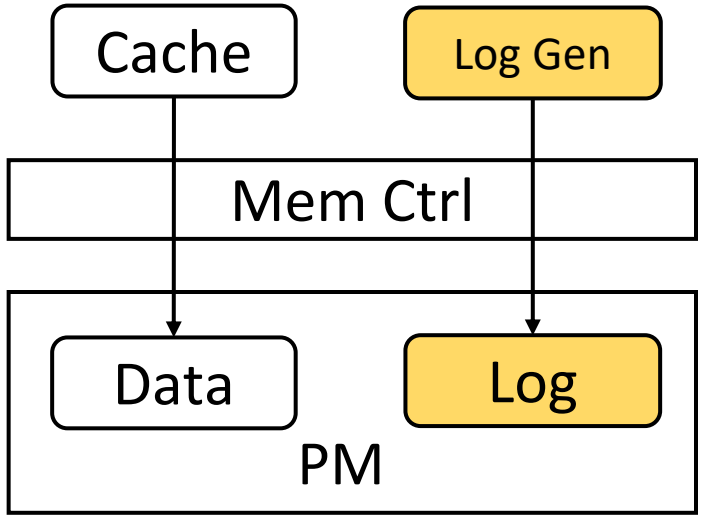
ReDU@MICRO'18



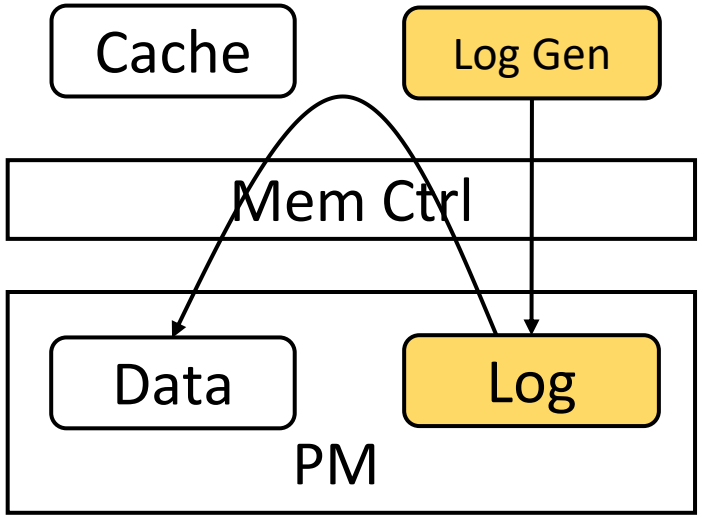
Proteus@MICRO'17

State-of-The-Art

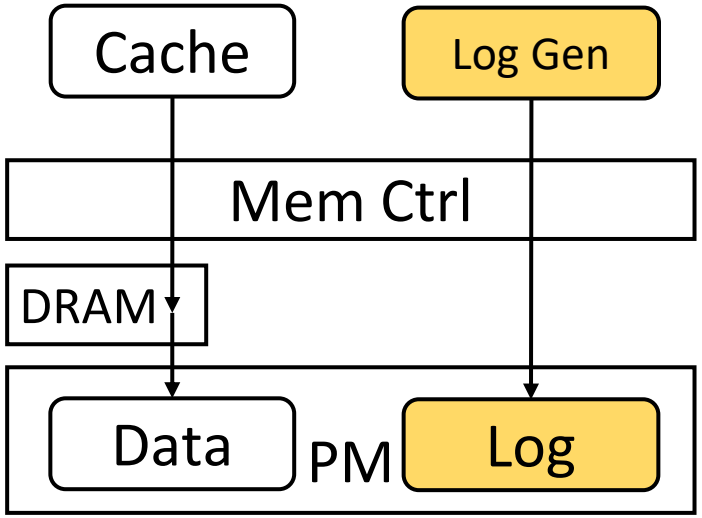
Log as backup



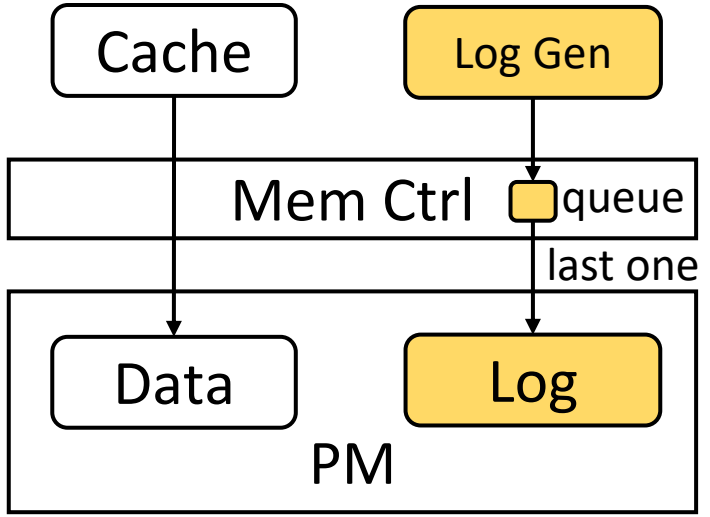
ATOM@HPCA'17
FWB@HPCA'18
MorLog@ISCA'20
ASAP@ISCA'22



WrAP@HPCA'16



ReDU@MICRO'18



Proteus@MICRO'17

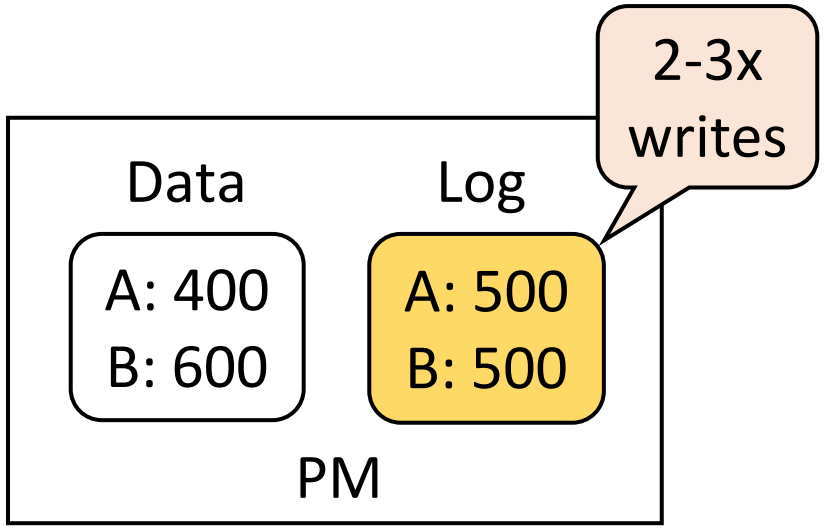
Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

Challenges

Heavy Writes

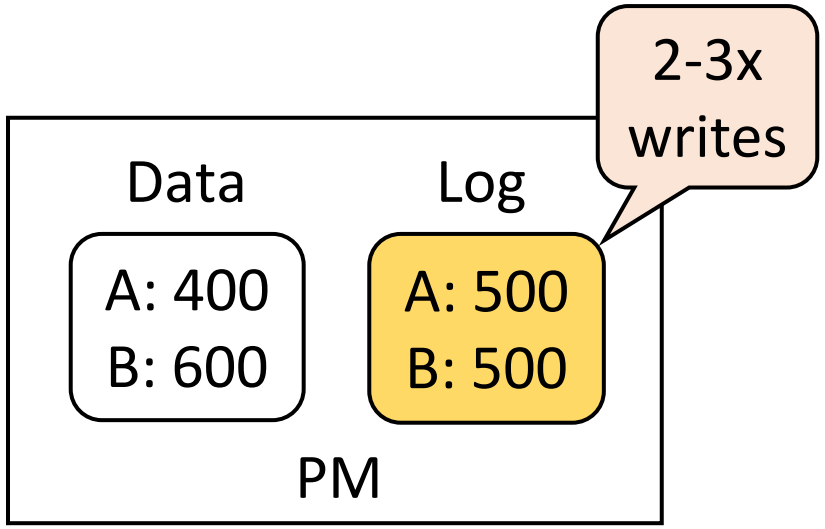
```
Tx_begin
  write A
  write B
Tx_end
```



Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

Heavy Writes

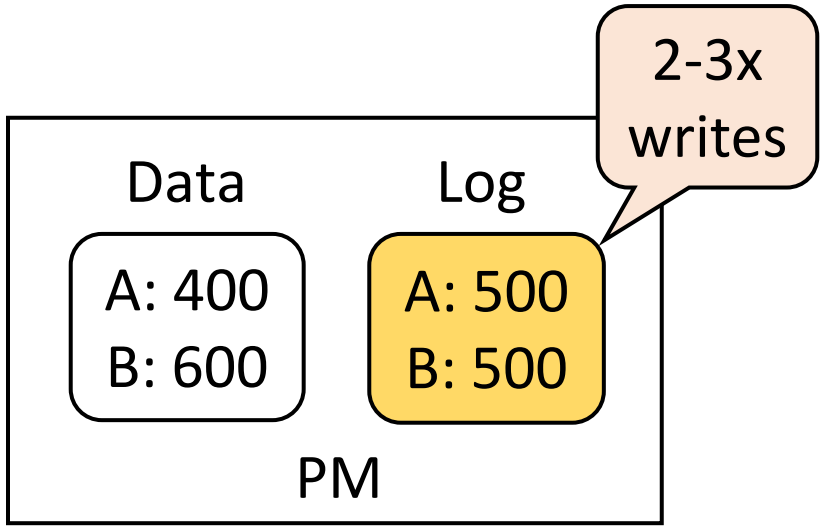


Logging supports to recover data from a system crash, but increases the write traffic

Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

Heavy Writes



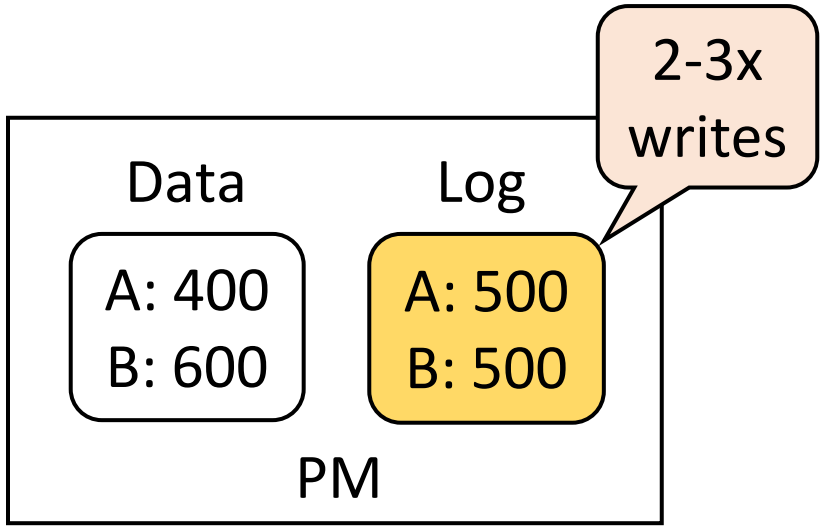
Logging supports to recover data from a system crash, but increases the write traffic

➔ Exacerbate PM endurance

Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

Heavy Writes



Logging supports to recover data from a system crash, but increases the write traffic

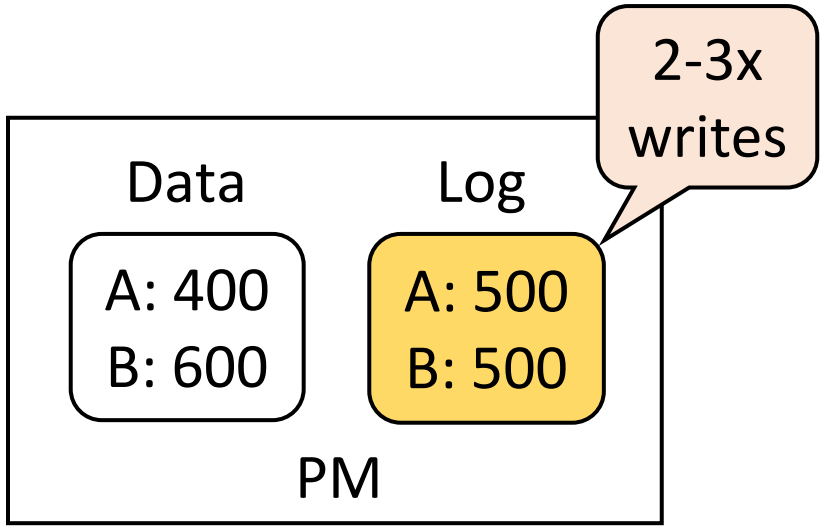
➔ Exacerbate PM endurance

Ordering Constraints

Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

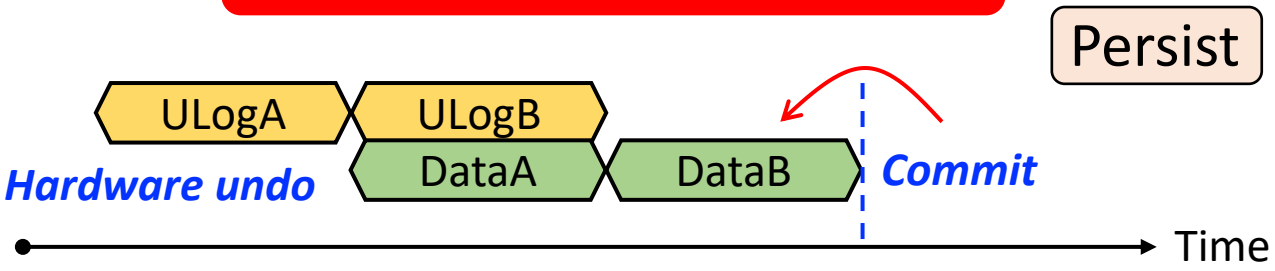
Heavy Writes



Logging supports to recover data from a system crash, but increases the write traffic

➔ Exacerbate PM endurance

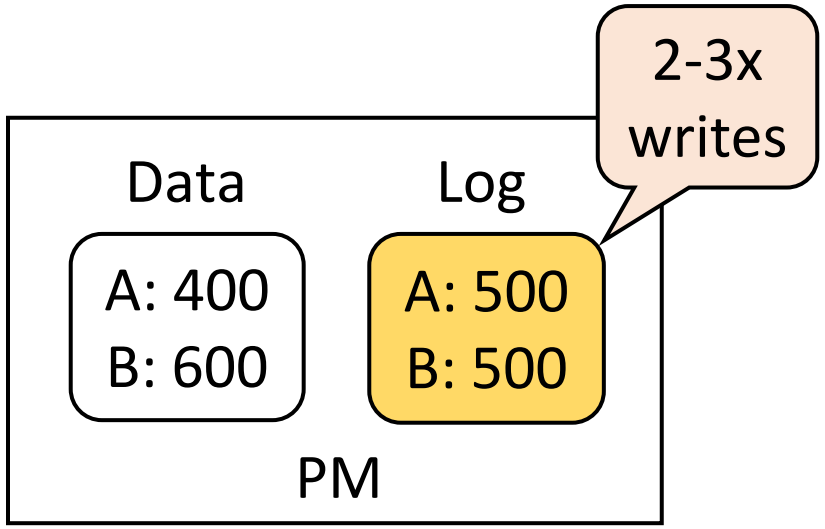
Ordering Constraints



Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

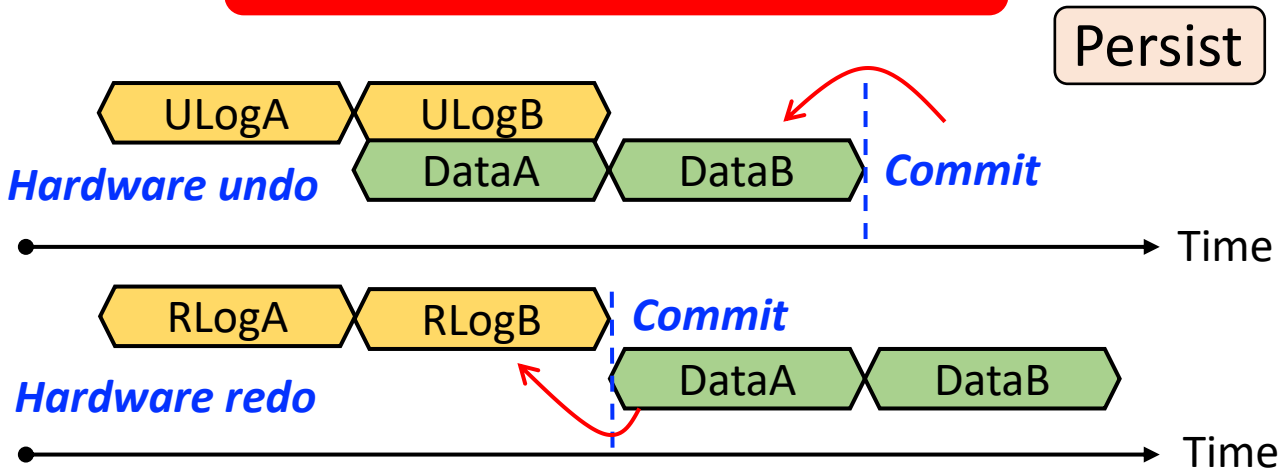
Heavy Writes



Logging supports to recover data from a system crash, but increases the write traffic

➔ Exacerbate PM endurance

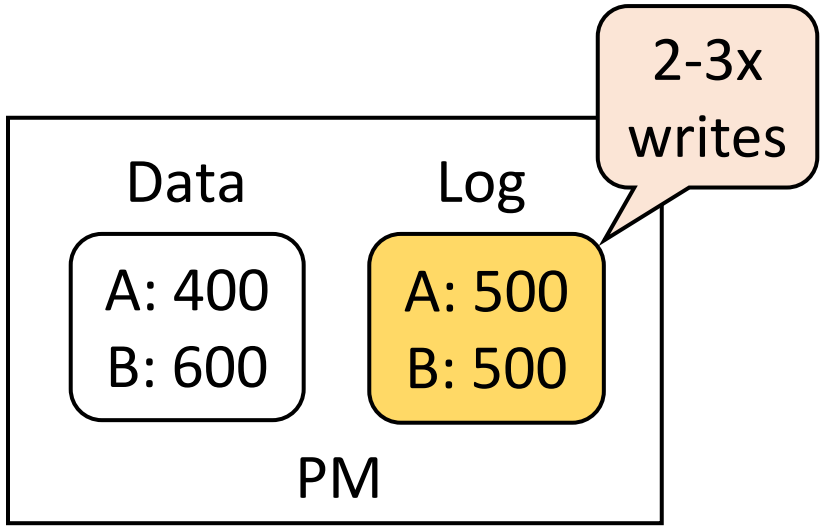
Ordering Constraints



Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

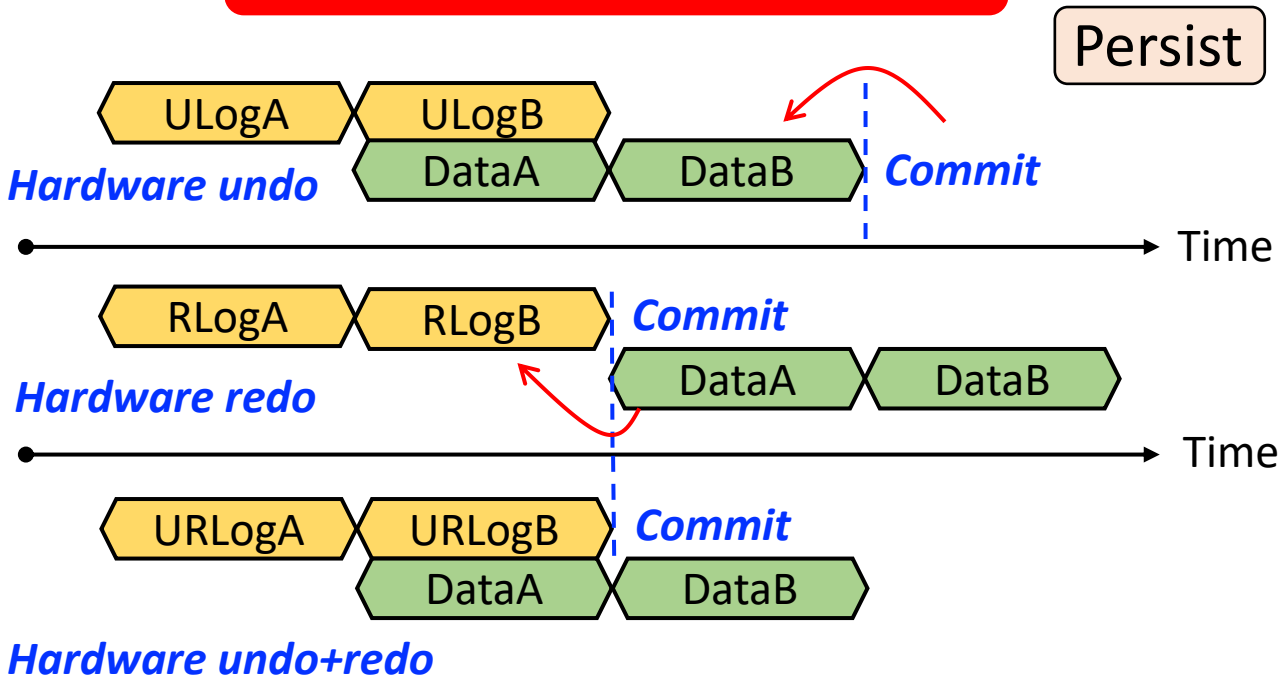
Heavy Writes



Logging supports to recover data from a system crash, but increases the write traffic

➔ Exacerbate PM endurance

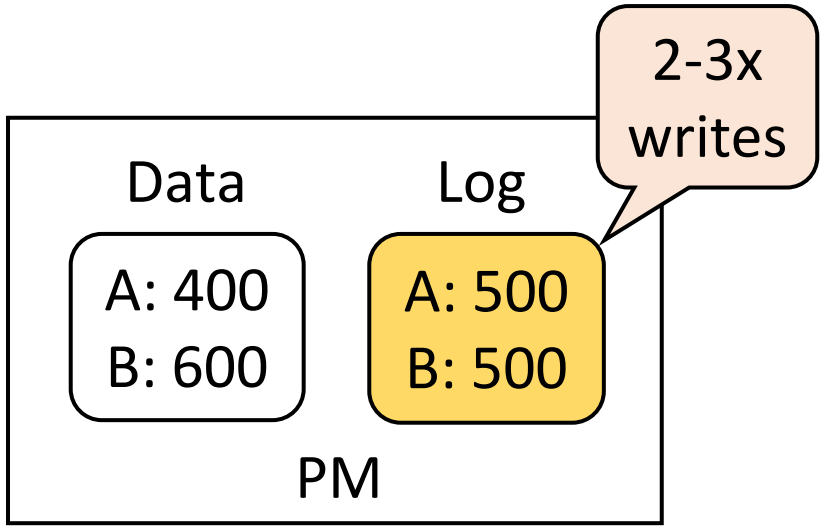
Ordering Constraints



Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

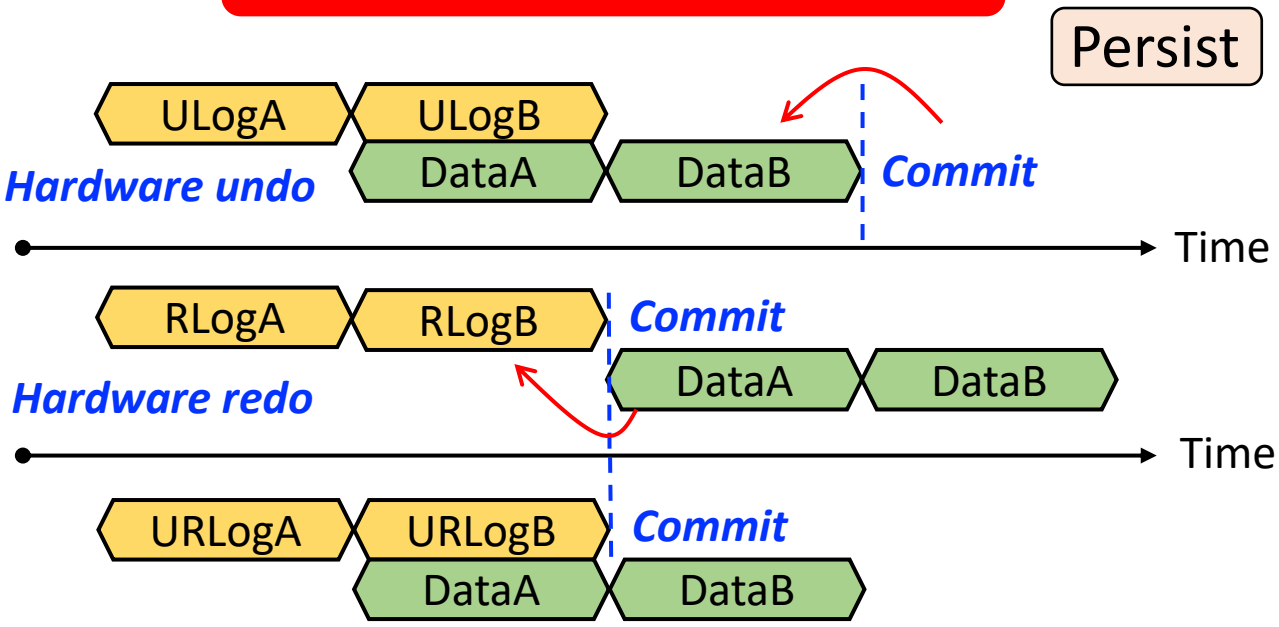
Heavy Writes



Logging supports to recover data from a system crash, but increases the write traffic

➔ Exacerbate PM endurance

Ordering Constraints



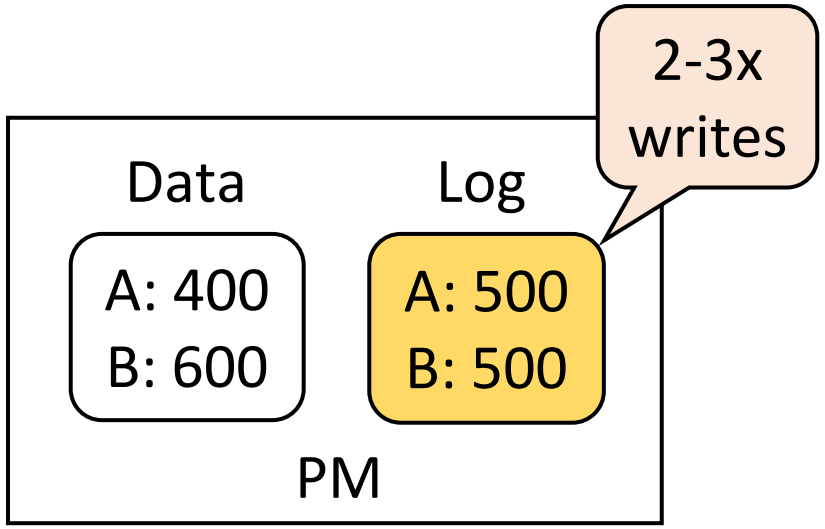
- **FWB**[1] writes logs to PM before the updated data for each write
- **MorLog**[2] flushes logs to PM before commit to ensure durability

[1] FWB@HPCA'18 [2] MorLog@ISCA'20

Challenges

```
Tx_begin
  write A
  write B
Tx_end
```

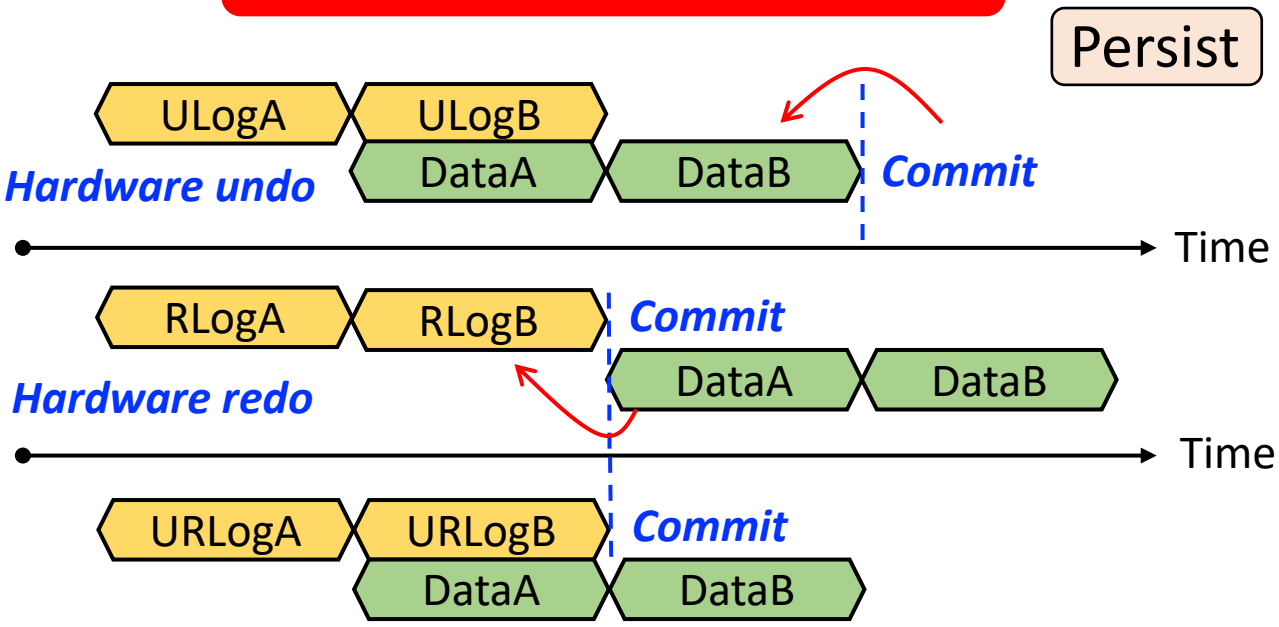
Heavy Writes



Logging supports to recover data from a system crash, but increases the write traffic

➔ Exacerbate PM endurance

Ordering Constraints



- **FWB**[1] writes logs to PM before the updated data for each write
- **MorLog**[2] flushes logs to PM before commit to ensure durability

➔ Increase latency

[1] FWB@HPCA'18 [2] MorLog@ISCA'20

Key Ideas

Key Ideas

➢ ***Speculative Logging***

- Crash is rare for a single machine[1-2]

➔ Do not conservatively write logs to PM in common cases (no failures)

➔ Only write logs to PM in rare cases (e.g., crashes) to guarantee atomic durability

[1] Lazy Persistency@ISCA'18 [2] LAD@MICRO'19

Key Ideas

- ***Speculative Logging***
  - Crash is rare for a single machine[1-2]

  ➔ Do not conservatively write logs to PM in common cases (no failures)

  ➔ Only write logs to PM in rare cases (e.g., crashes) to guarantee atomic durability
- ***Log as Data***
  - Logs are able to record the new data

  ➔ Use on-chip logs to in-place update the PM data after commit in common cases

[1] Lazy Persistency@ISCA'18 [2] LAD@MICRO'19

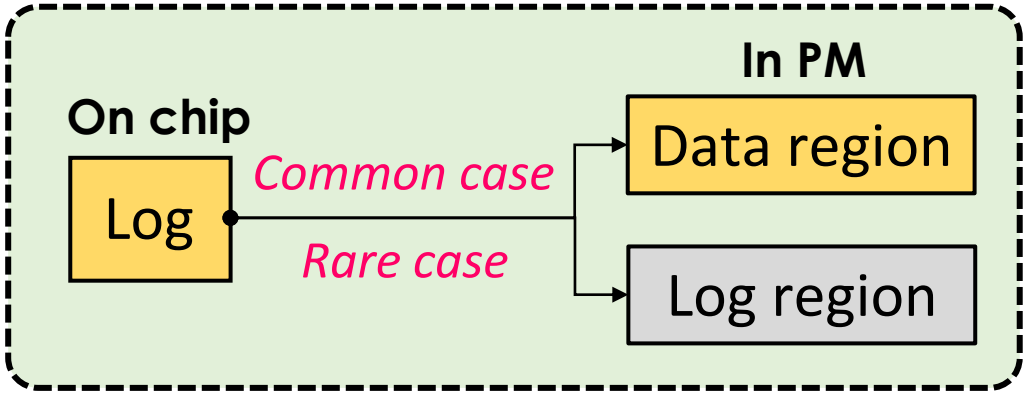
Key Ideas

- ***Speculative Logging***
  - Crash is rare for a single machine[1-2]

  ➔ Do not conservatively write logs to PM in common cases (no failures)

  ➔ Only write logs to PM in rare cases (e.g., crashes) to guarantee atomic durability
- ***Log as Data***
  - Logs are able to record the new data

  ➔ Use on-chip logs to in-place update the PM data after commit in common cases



[1] Lazy Persistency@ISCA'18 [2] LAD@MICRO'19

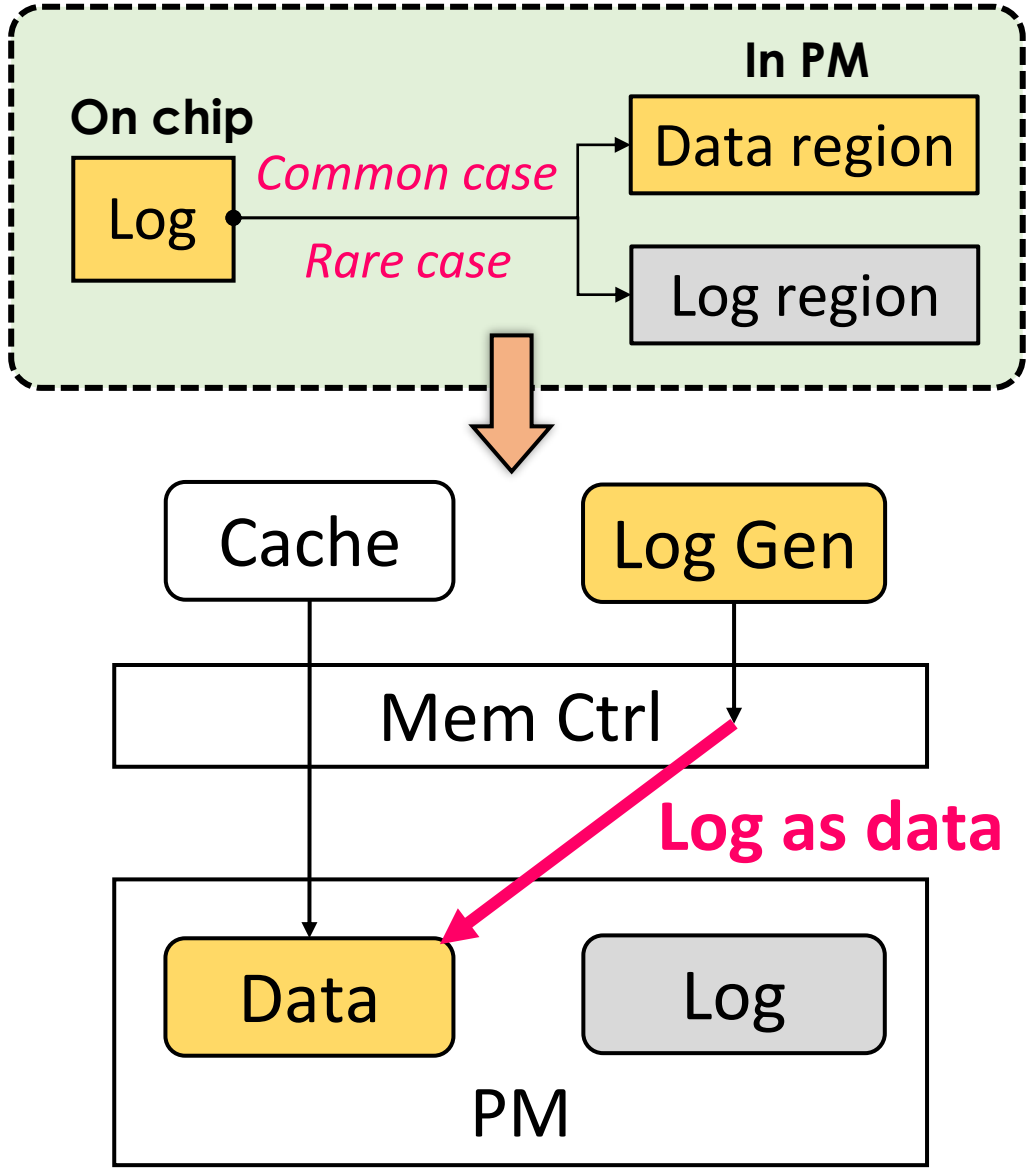
Key Ideas

- ***Speculative Logging***
  - Crash is rare for a single machine[1-2]

  ➔ Do not conservatively write logs to PM in common cases (no failures)

  ➔ Only write logs to PM in rare cases (e.g., crashes) to guarantee atomic durability
- ***Log as Data***
  - Logs are able to record the new data

  ➔ Use on-chip logs to in-place update the PM data after commit in common cases



[1] Lazy Persistency@ISCA'18 [2] LAD@MICRO'19

Key Ideas

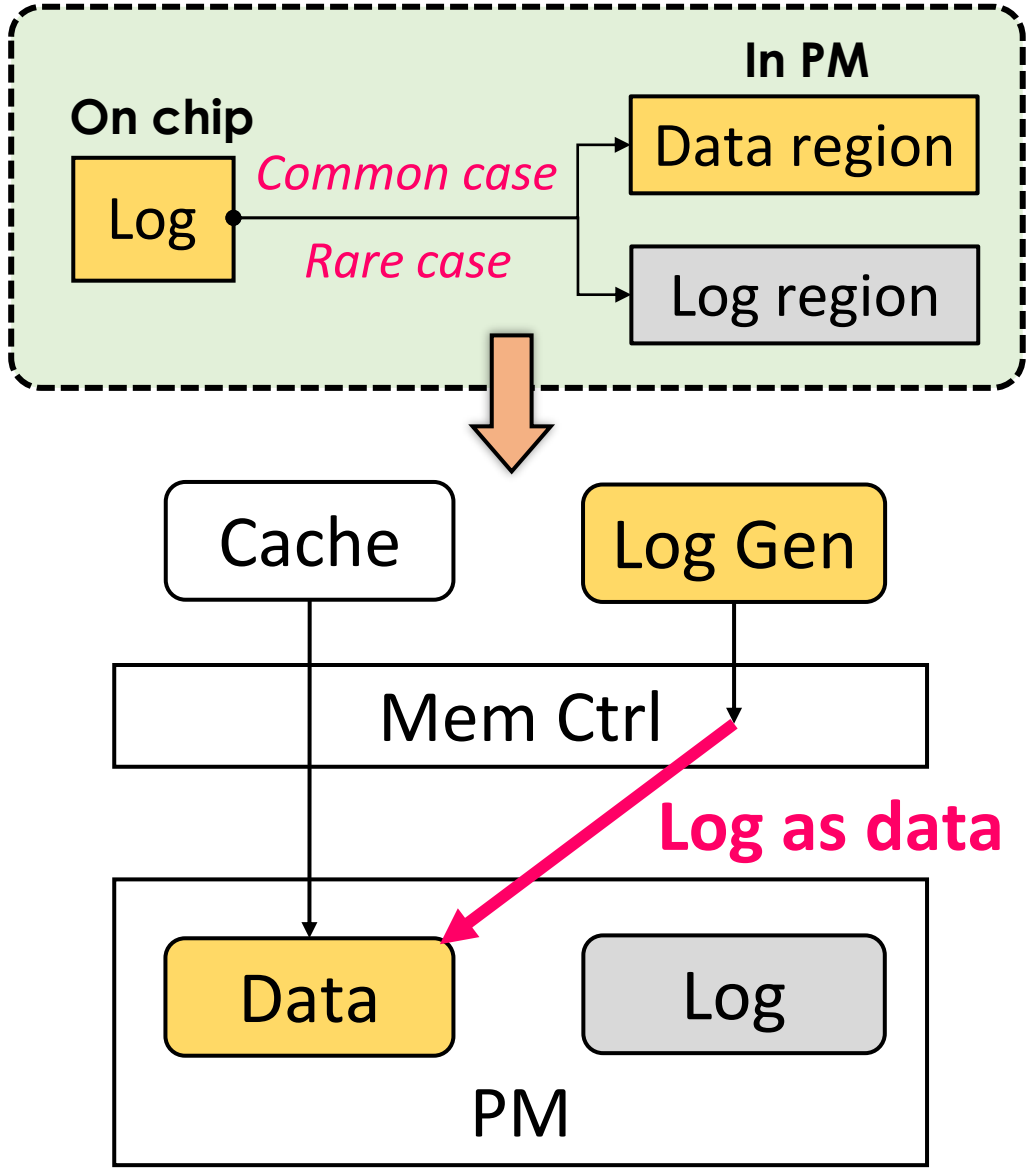
- ***Speculative Logging***
  - Crash is rare for a single machine[1-2]

  ➔ Do not conservatively write logs to PM in common cases (no failures)

  ➔ Only write logs to PM in rare cases (e.g., crashes) to guarantee atomic durability
- ***Log as Data***
  - Logs are able to record the new data

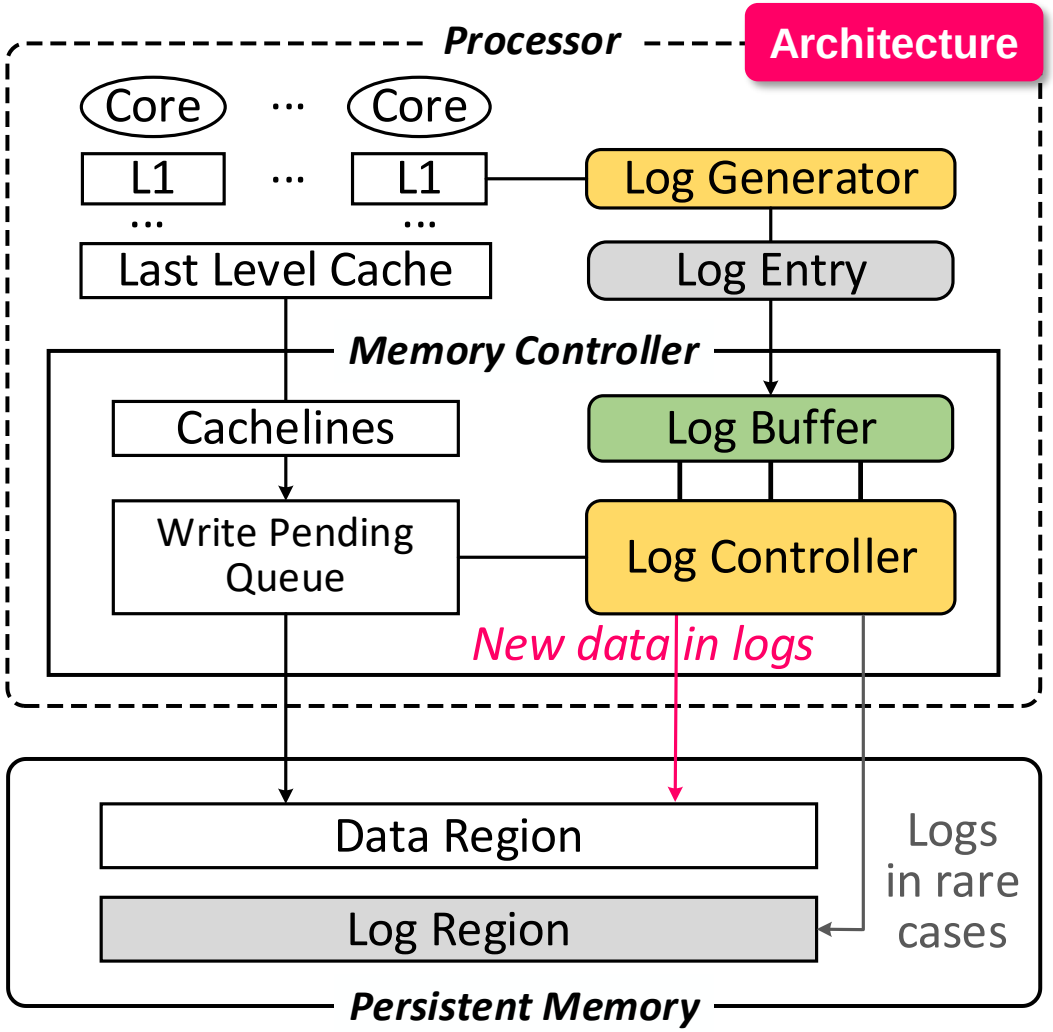
  ➔ Use on-chip logs to in-place update the PM data after commit in common cases

Make the common case fast and guarantee recoverability

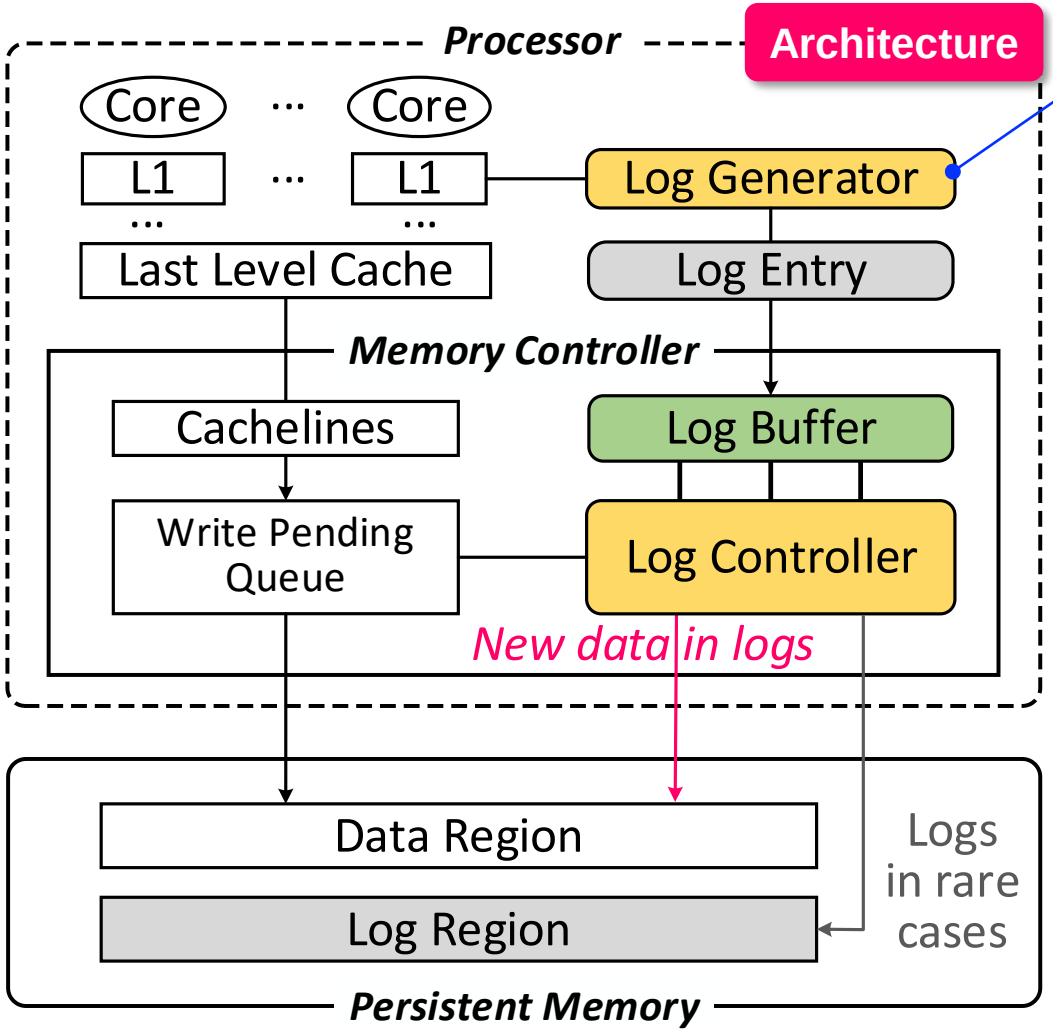


[1] Lazy Persistency@ISCA'18 [2] LAD@MICRO'19

Silo: Speculative Hardware Logging

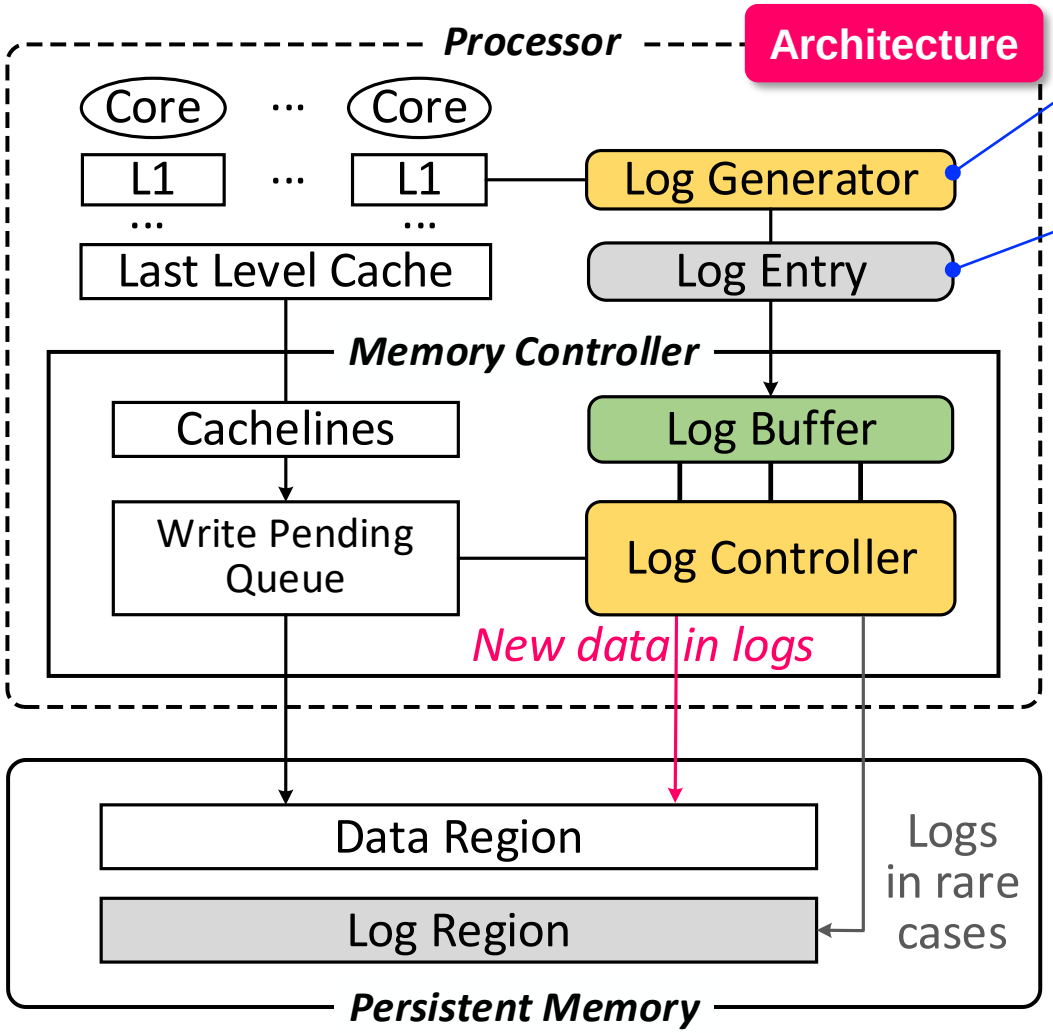


Silo: Speculative Hardware Logging

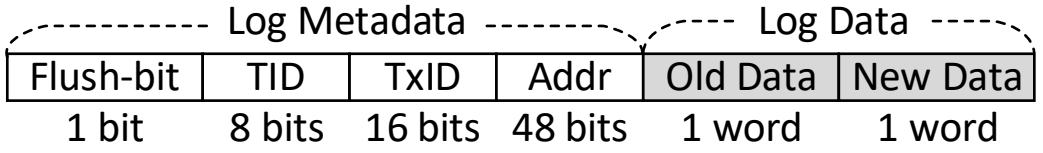


Monitor L1D updates during txns and generate log entries

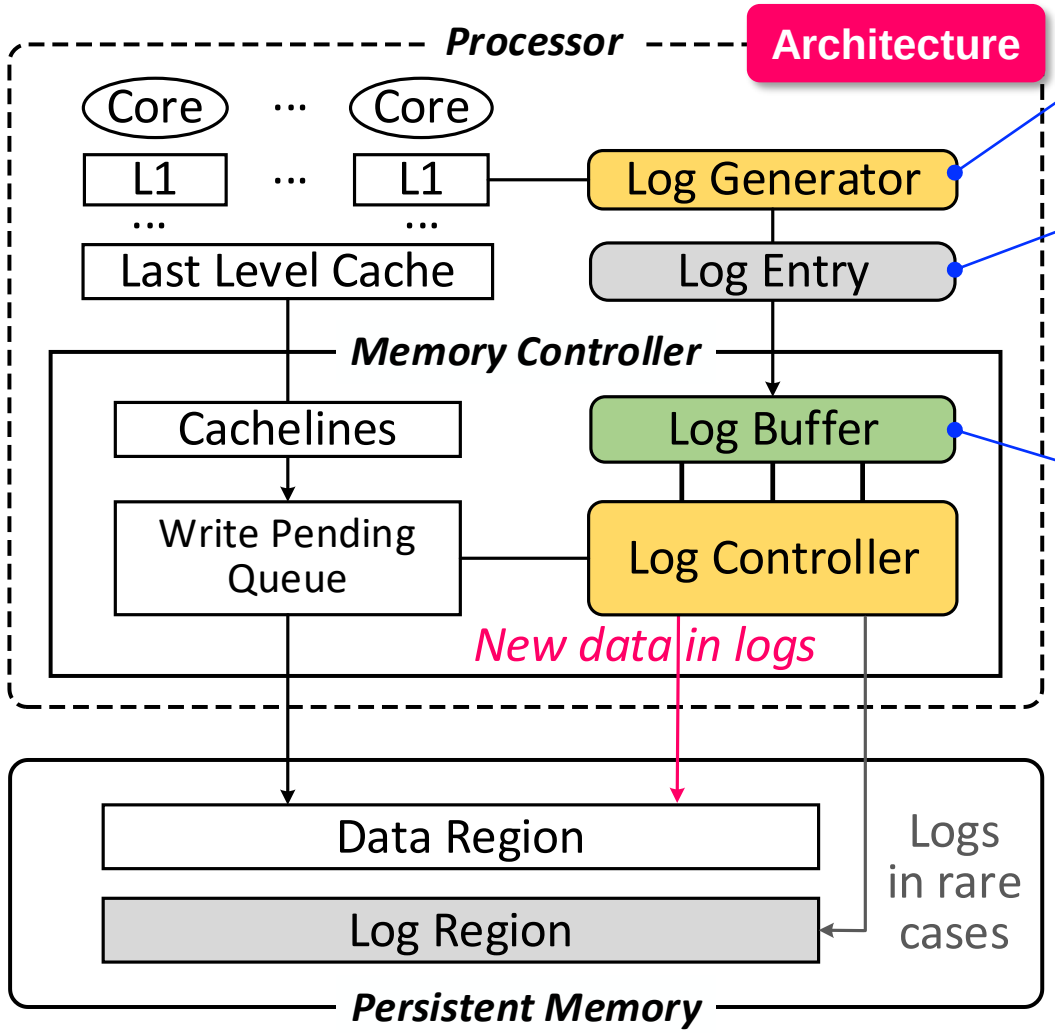
Silo: Speculative Hardware Logging



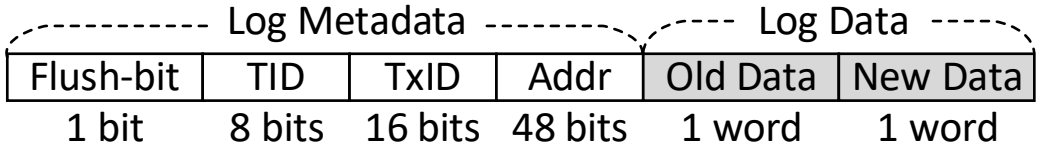
Monitor L1D updates during txns and generate log entries

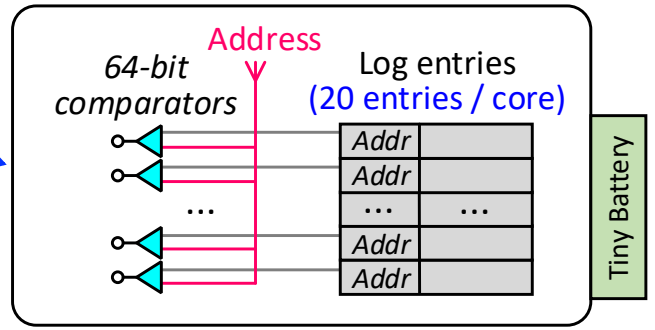


Silo: Speculative Hardware Logging



Monitor L1D updates during txns and generate log entries



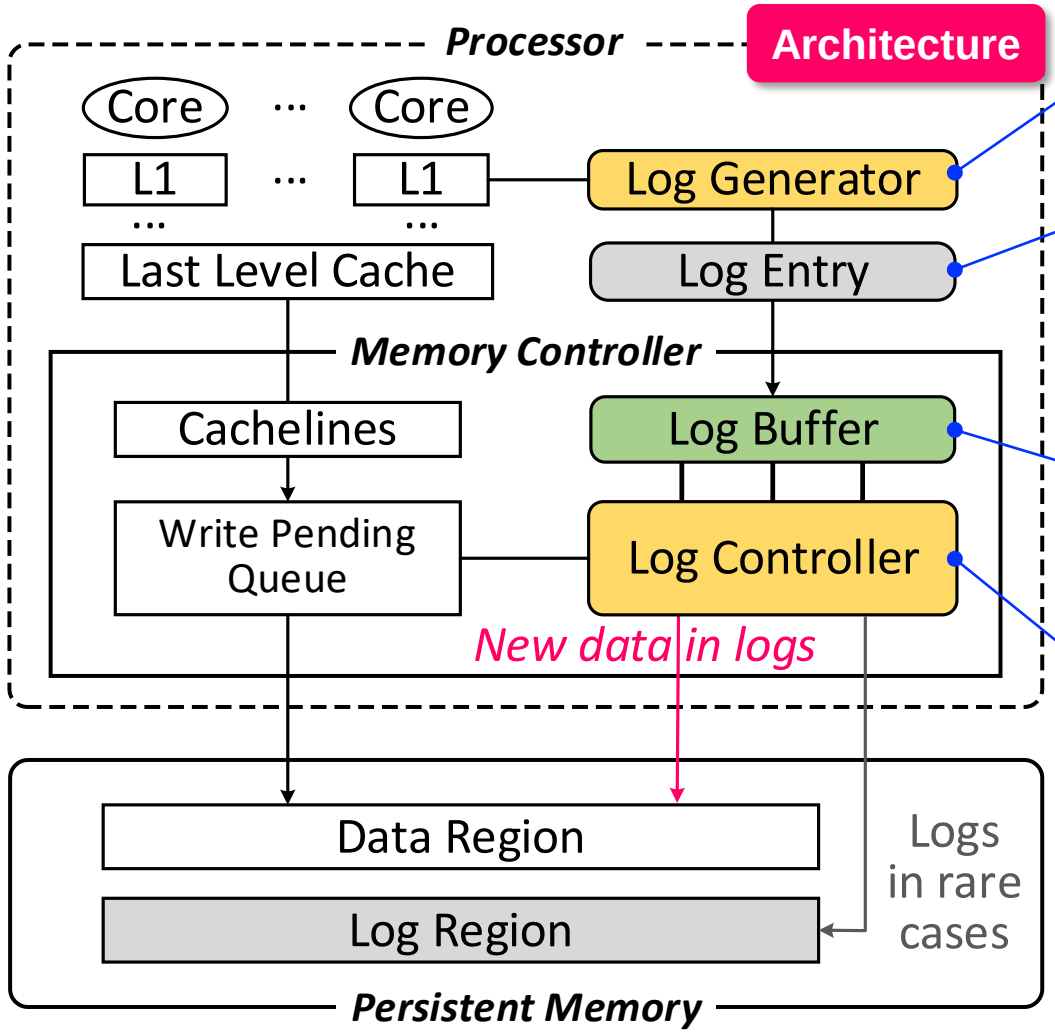


Hold the log entries of one txn

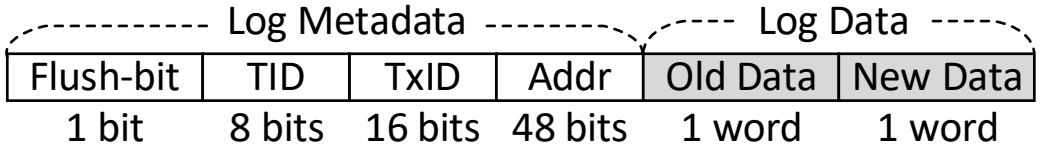
- Small: 680B per core. The write sets of txns in OLTP applications are typically small[1-5]
- Battery-backed: A tiny battery (e.g., 2.125 x 10^{-4}mm^3 of Li* per core) supplies energy on crashes

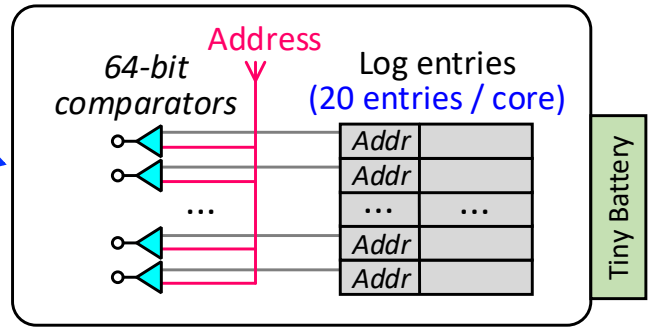
[1] ReDU@MICRO'18 [2] Deneva@VLDB'17 [3] Hybrid Index@SIGMOD'16 [4] FOEDUS@SIGMOD'15
[5] From Oracle (https://www.oracle.com/database/what-isoltp/) * Lithium thin-film battery

Silo: Speculative Hardware Logging



Monitor L1D updates during txns and generate log entries





Hold the log entries of one txn

- Small: 680B per core. The write sets of txns in OLTP applications are typically small[1-5]
- Battery-backed: A tiny battery (e.g., 2.125 x 10^{-4}mm^3 of Li* per core) supplies energy on crashes

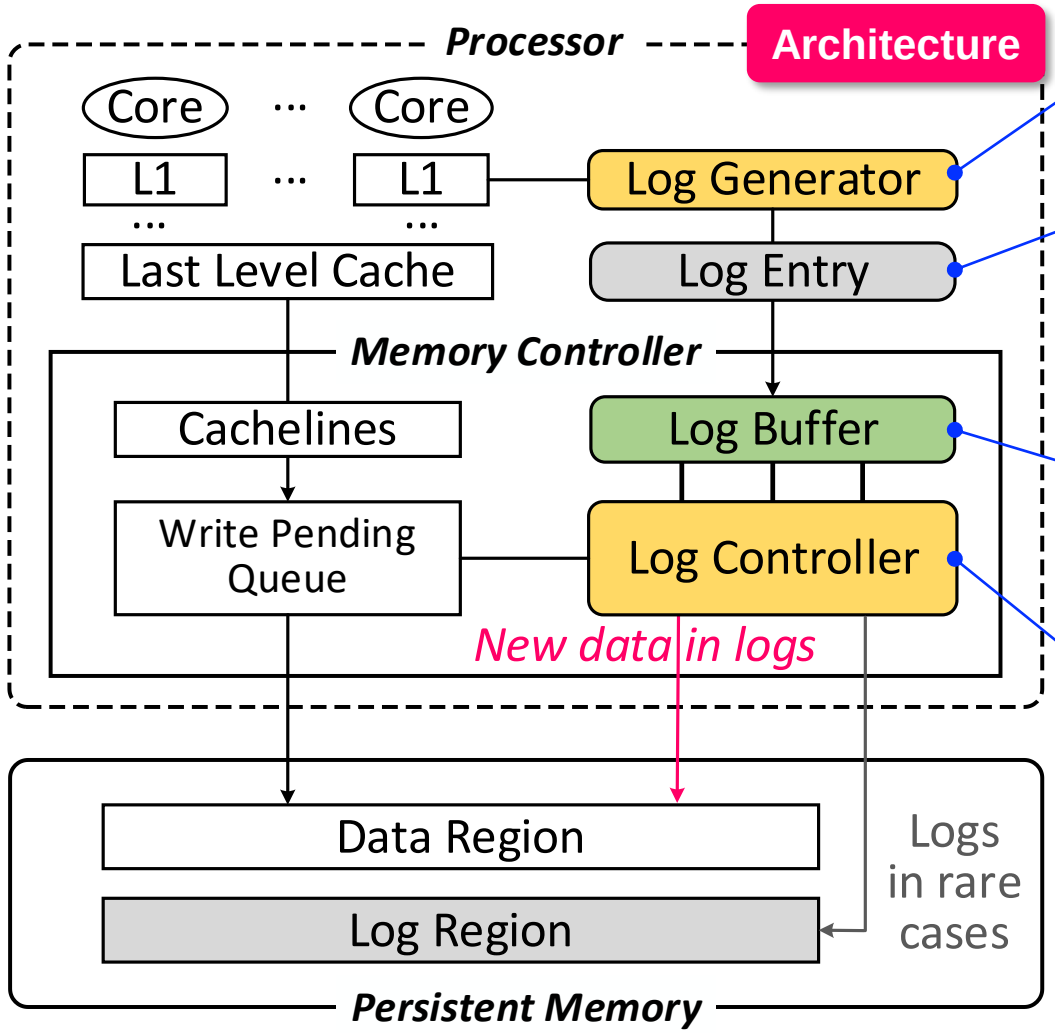
Schemes to manipulate logs

- Log reduction
- Log update
- Write coalescing
- Handle log overflow
- Selective log flush on crash

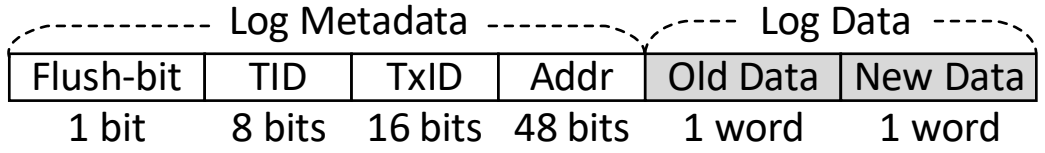
> Rare cases

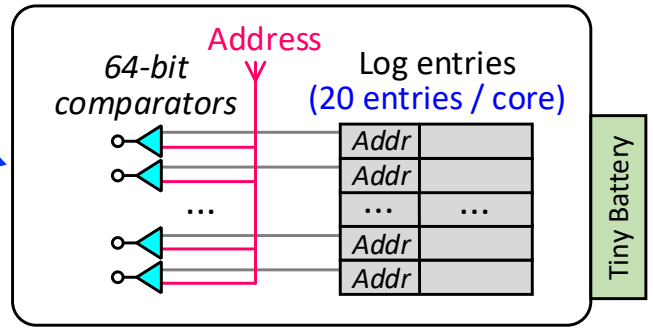
[1] ReDU@MICRO'18 [2] Deneva@VLDB'17 [3] Hybrid Index@SIGMOD'16 [4] FOEDUS@SIGMOD'15
[5] From Oracle (https://www.oracle.com/database/what-isoltp/) * Lithium thin-film battery

Silo: Speculative Hardware Logging



Monitor L1D updates during txns and generate log entries





Hold the log entries of one txn

- Small: 680B per core. The write sets of txns in OLTP applications are typically small[1-5]
- Battery-backed: A tiny battery (e.g., 2.125 x 10^{-4}mm^3 of Li* per core) supplies energy on crashes

Schemes to manipulate logs

- Log reduction
- Log update
- Write coalescing
- Handle log overflow
- Selective log flush on crash

> Rare cases

[1] ReDU@MICRO'18 [2] Deneva@VLDB'17 [3] Hybrid Index@SIGMOD'16 [4] FOEDUS@SIGMOD'15
[5] From Oracle (https://www.oracle.com/database/what-isoltp/) * Lithium thin-film battery

Log Reduction

- Reduce the size of on-chip log buffer based on write behaviors

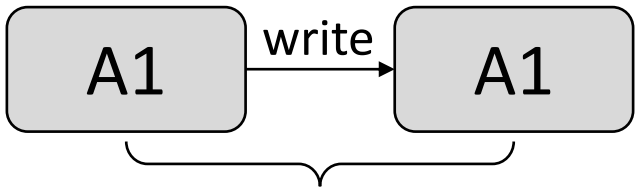
Log Reduction

➢ Reduce the size of on-chip log buffer based on write behaviors

Log Ignorance

A write does not modify the data

- E.g., copy and assignment[1]
- Old data == New data



Log generator ignores this write

- Does not produce log entry

[1] LLC Deduplication@ICS'14

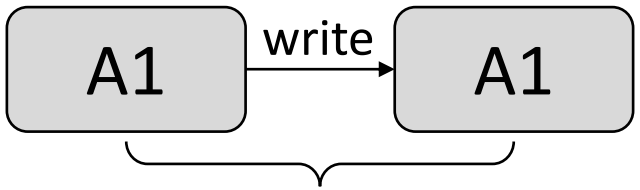
Log Reduction

➢ Reduce the size of on-chip log buffer based on write behaviors

Log Ignorance

A write does not modify the data

- E.g., copy and assignment[1]
- Old data == New data



Log generator ignores this write

- Does not produce log entry

Log Merging

Multiple writes modify the same data

- Temporal locality of programs
- Only the oldest and newest data are required

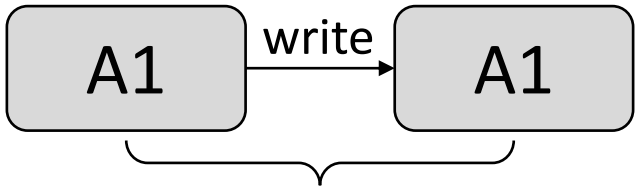
[1] LLC Deduplication@ICS'14

Log Reduction

➢ Reduce the size of on-chip log buffer based on write behaviors

Log Ignorance

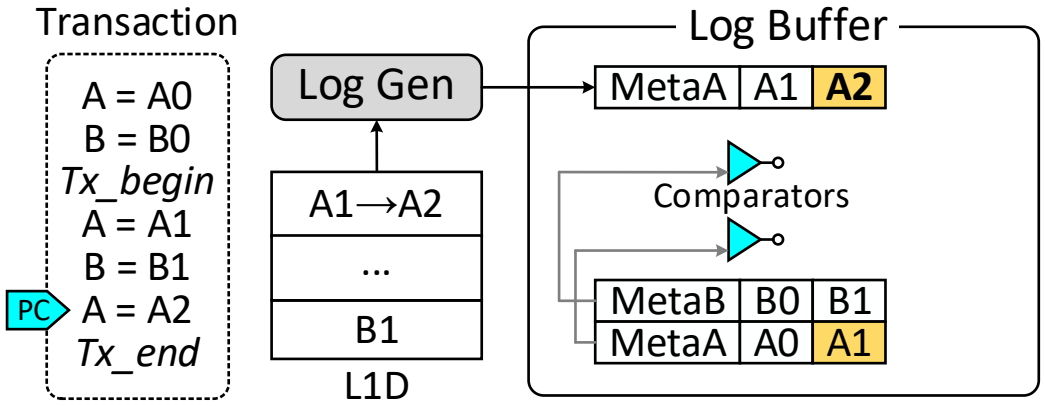
A write does not modify the data
- E.g., copy and assignment[1]
- Old data == New data



Log generator ignores this write
- Does not produce log entry

Log Merging

Multiple writes modify the same data
- Temporal locality of programs
- Only the oldest and newest data are required



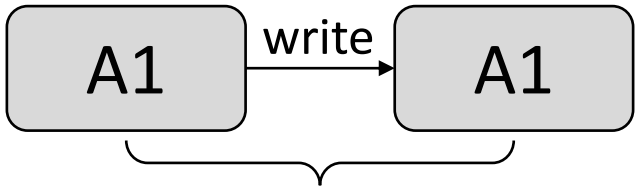
[1] LLC Deduplication@ICS'14

Log Reduction

➢ Reduce the size of on-chip log buffer based on write behaviors

Log Ignorance

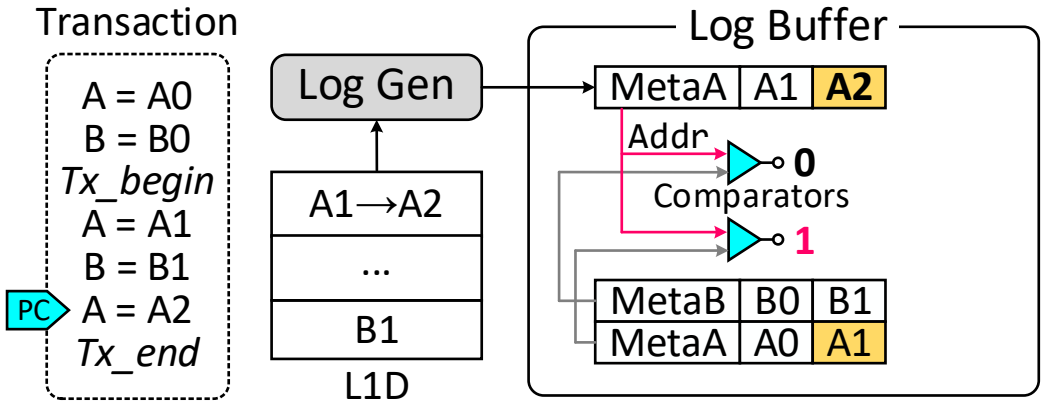
A write does not modify the data
- E.g., copy and assignment[1]
- Old data == New data



Log generator ignores this write
- Does not produce log entry

Log Merging

Multiple writes modify the same data
- Temporal locality of programs
- Only the oldest and newest data are required



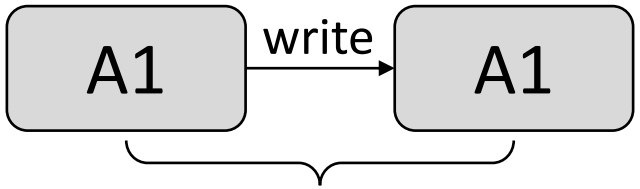
[1] LLC Deduplication@ICS'14

Log Reduction

➢ Reduce the size of on-chip log buffer based on write behaviors

Log Ignorance

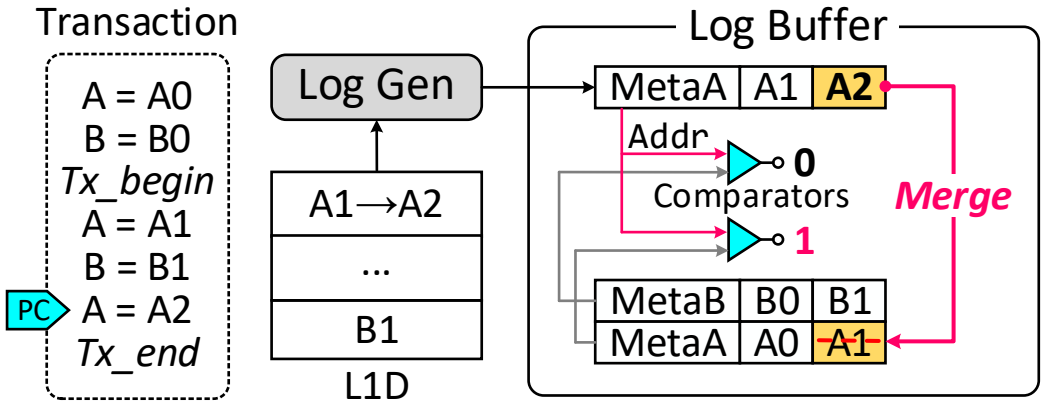
A write does not modify the data
- E.g., copy and assignment[1]
- Old data == New data



Log generator ignores this write
- Does not produce log entry

Log Merging

Multiple writes modify the same data
- Temporal locality of programs
- Only the oldest and newest data are required



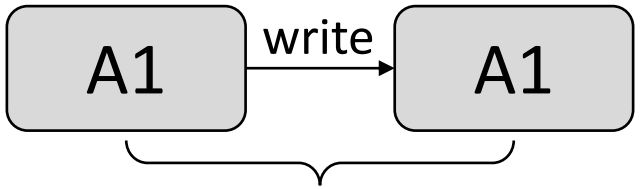
[1] LLC Deduplication@ICS'14

Log Reduction

➢ Reduce the size of on-chip log buffer based on write behaviors

Log Ignorance

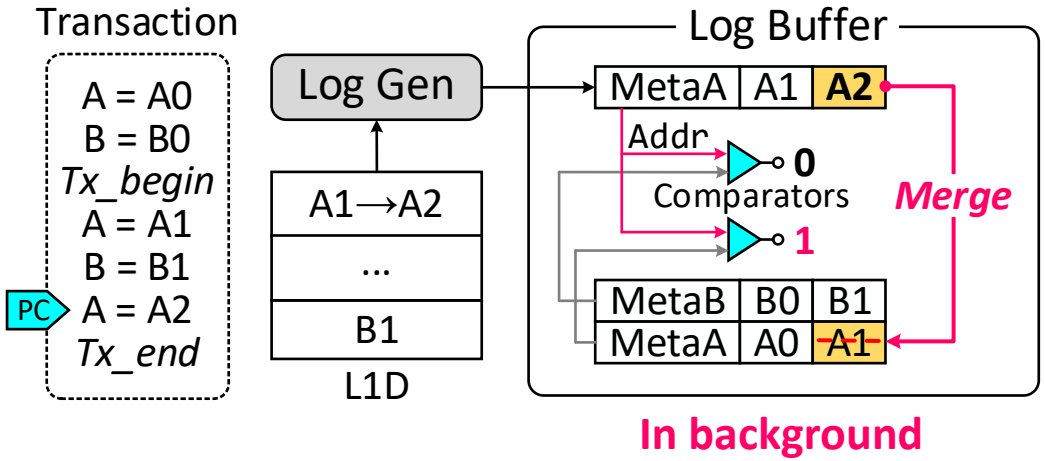
A write does not modify the data
- E.g., copy and assignment[1]
- Old data == New data



Log generator ignores this write
- Does not produce log entry

Log Merging

Multiple writes modify the same data
- Temporal locality of programs
- Only the oldest and newest data are required



[1] LLC Deduplication@ICS'14

Log Update

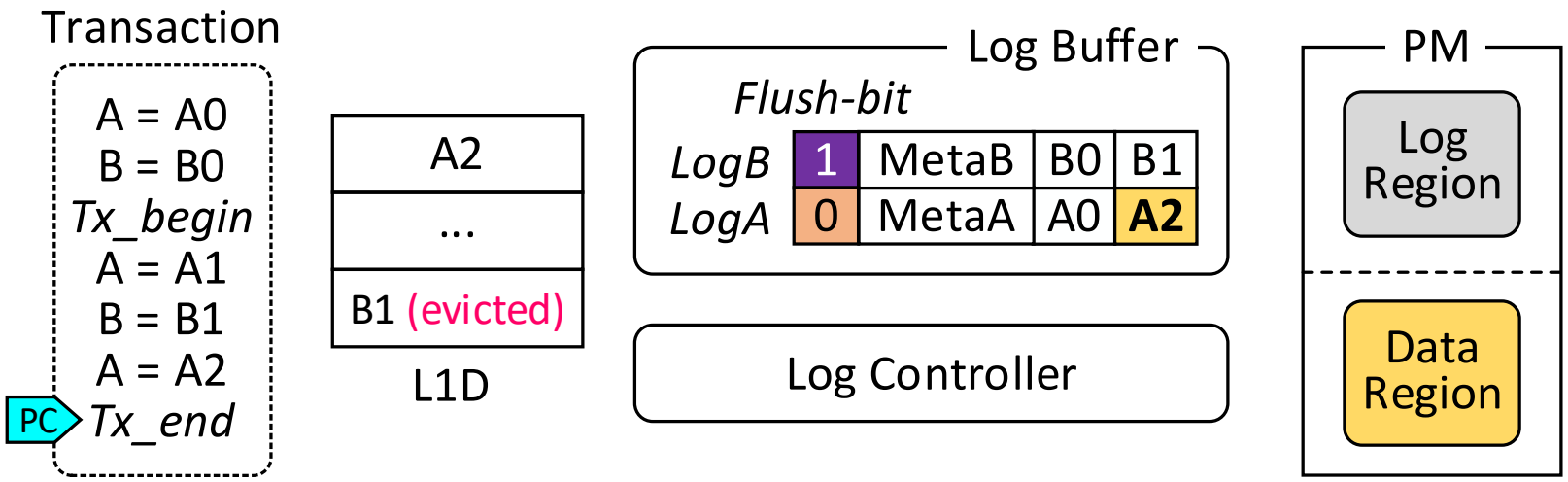
- Use the new data in on-chip logs to in-place update the data region

Log Update

- Use the new data in on-chip logs to in-place update the data region
- Not block cacheline evictions
  - Set the flush-bit to 1 to discard the log after commit if an updated cacheline is evicted

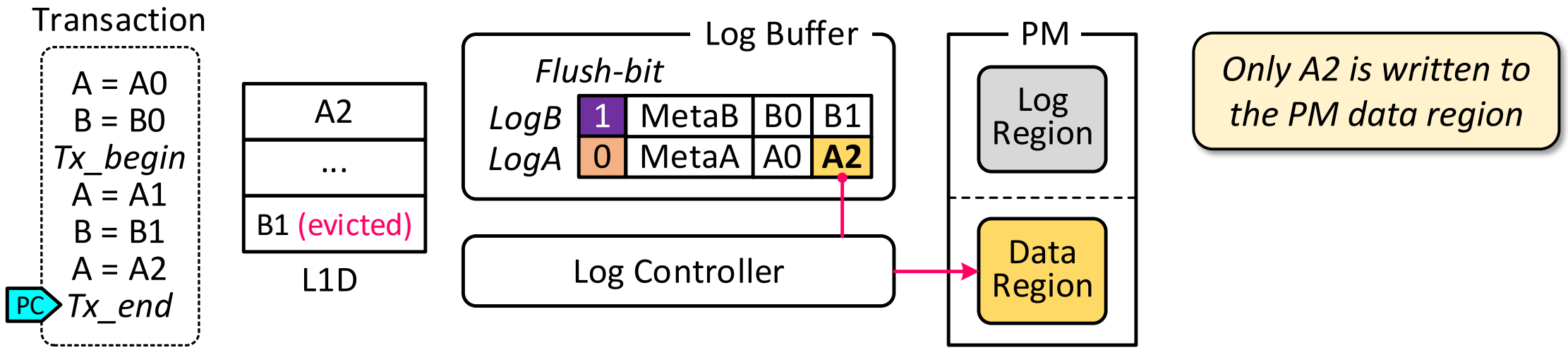
Log Update

- Use the new data in on-chip logs to in-place update the data region
- Not block cacheline evictions
  - Set the flush-bit to 1 to discard the log after commit if an updated cacheline is evicted



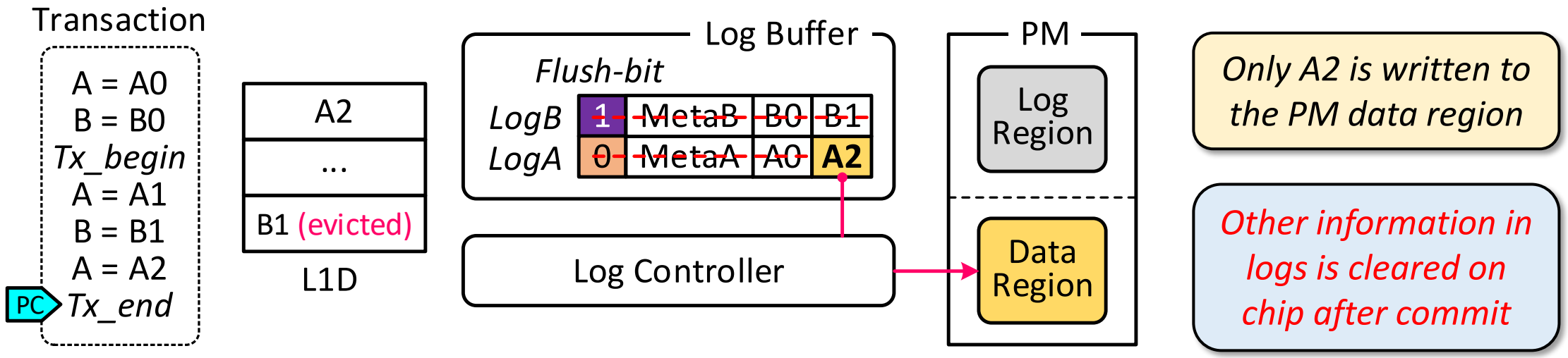
Log Update

- Use the new data in on-chip logs to in-place update the data region
- Not block cacheline evictions
  - Set the flush-bit to 1 to discard the log after commit if an updated cacheline is evicted



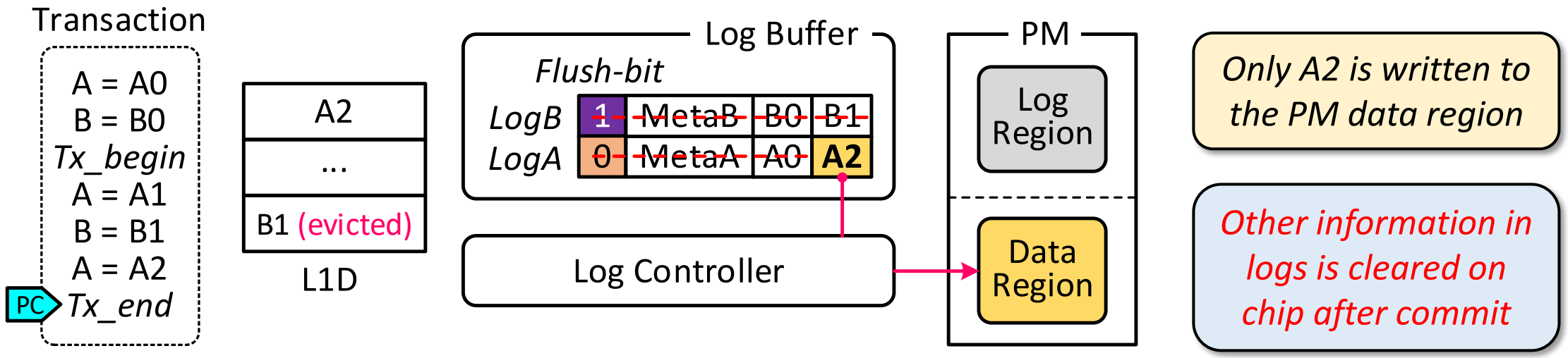
Log Update

- Use the new data in on-chip logs to in-place update the data region
- Not block cacheline evictions
  - Set the flush-bit to 1 to discard the log after commit if an updated cacheline is evicted



Log Update

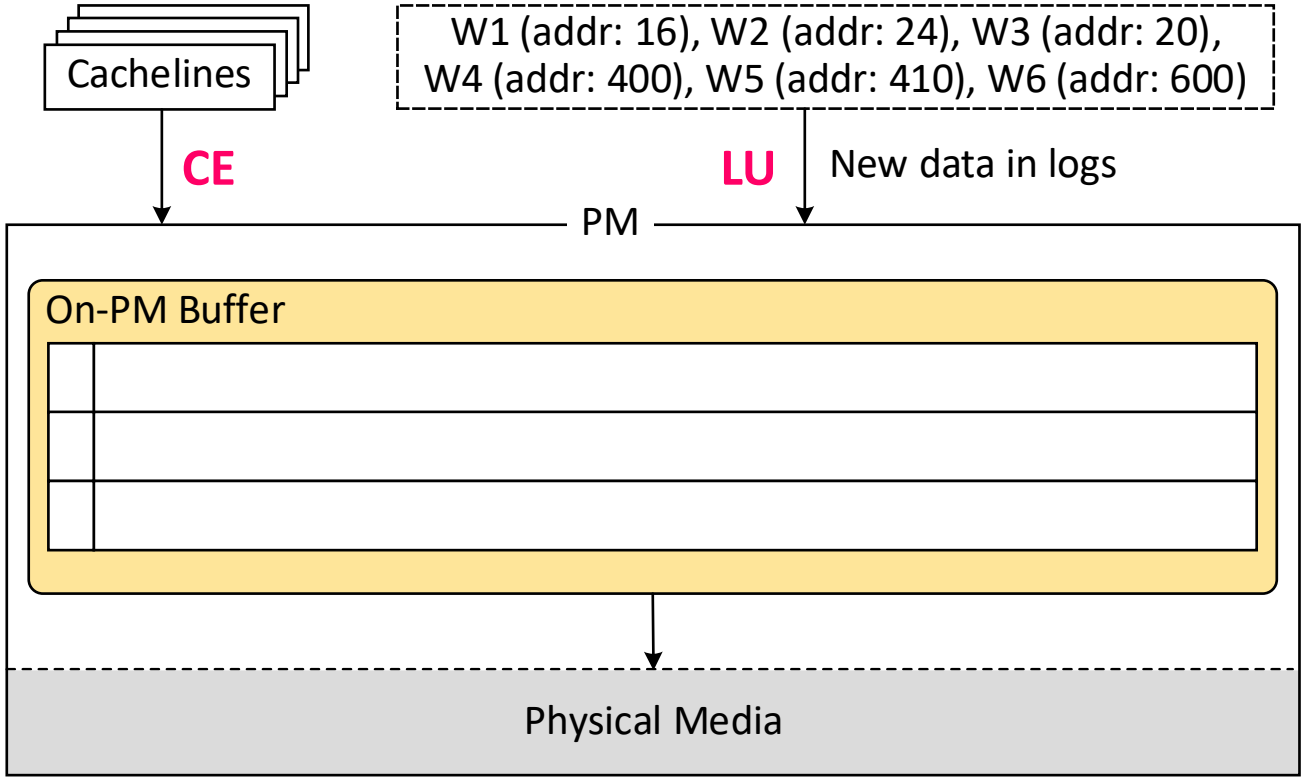
- Use the new data in on-chip logs to in-place update the data region
- Not block cacheline evictions
  - Set the flush-bit to 1 to discard the log after commit if an updated cacheline is evicted



- ***Benefits***
  - Write reduction: Don't write logs to PM in common cases
  - No ordering constraints: Don't wait for flushing logs (and cachelines) to the log (and data) regions

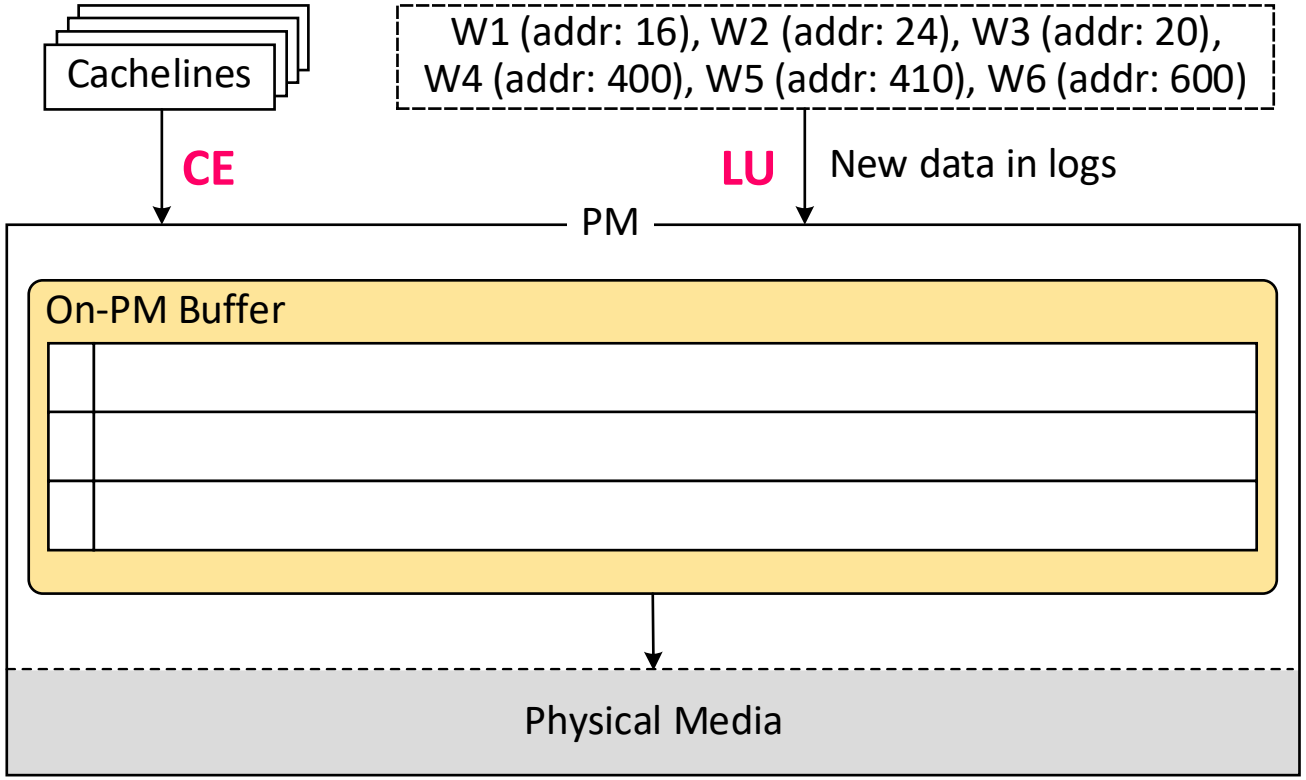
Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)



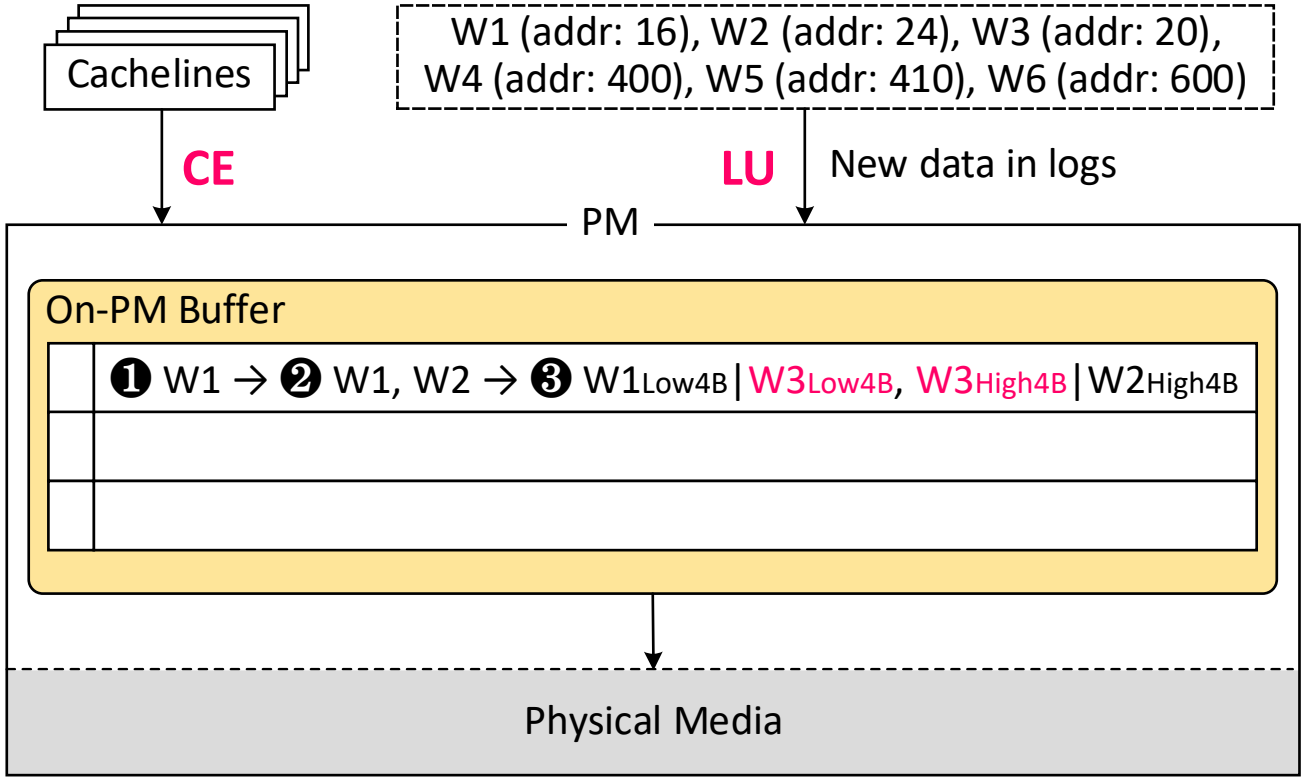
Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer



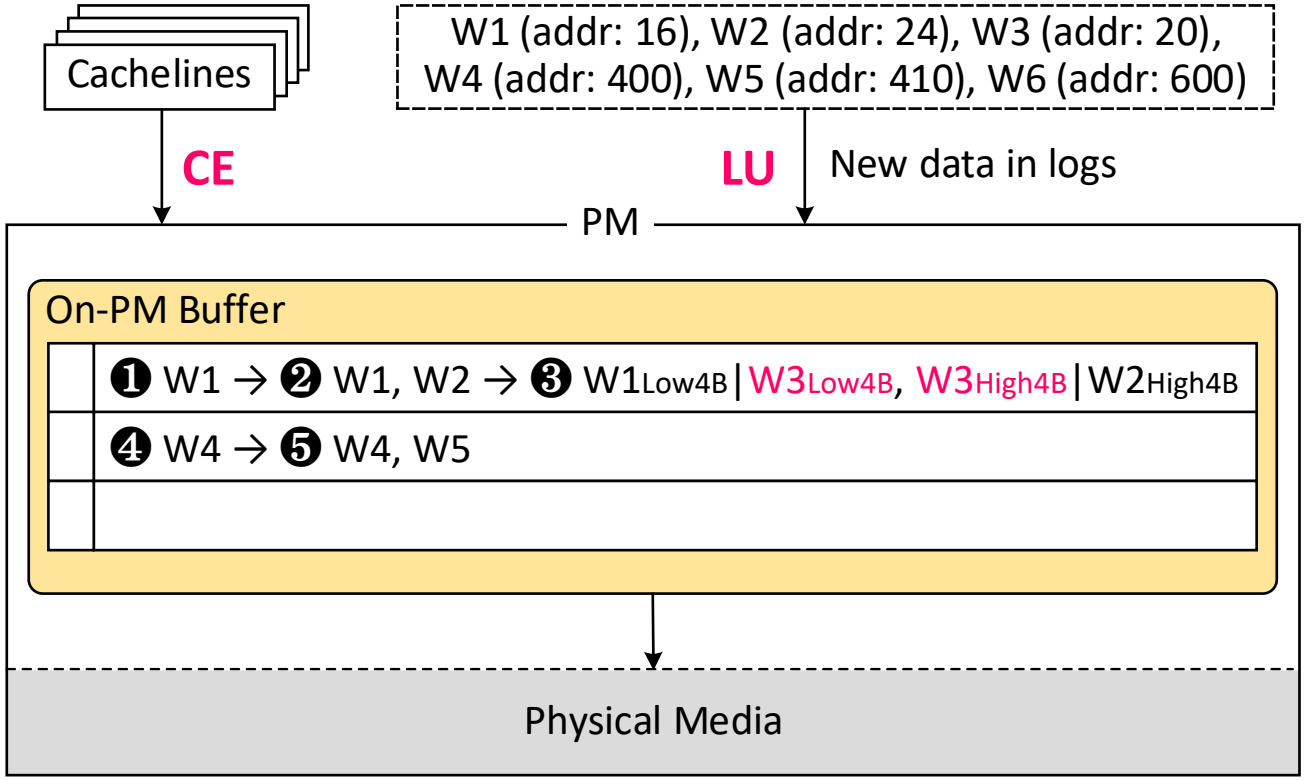
Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer
  - W1-W3 have overlapped bytes



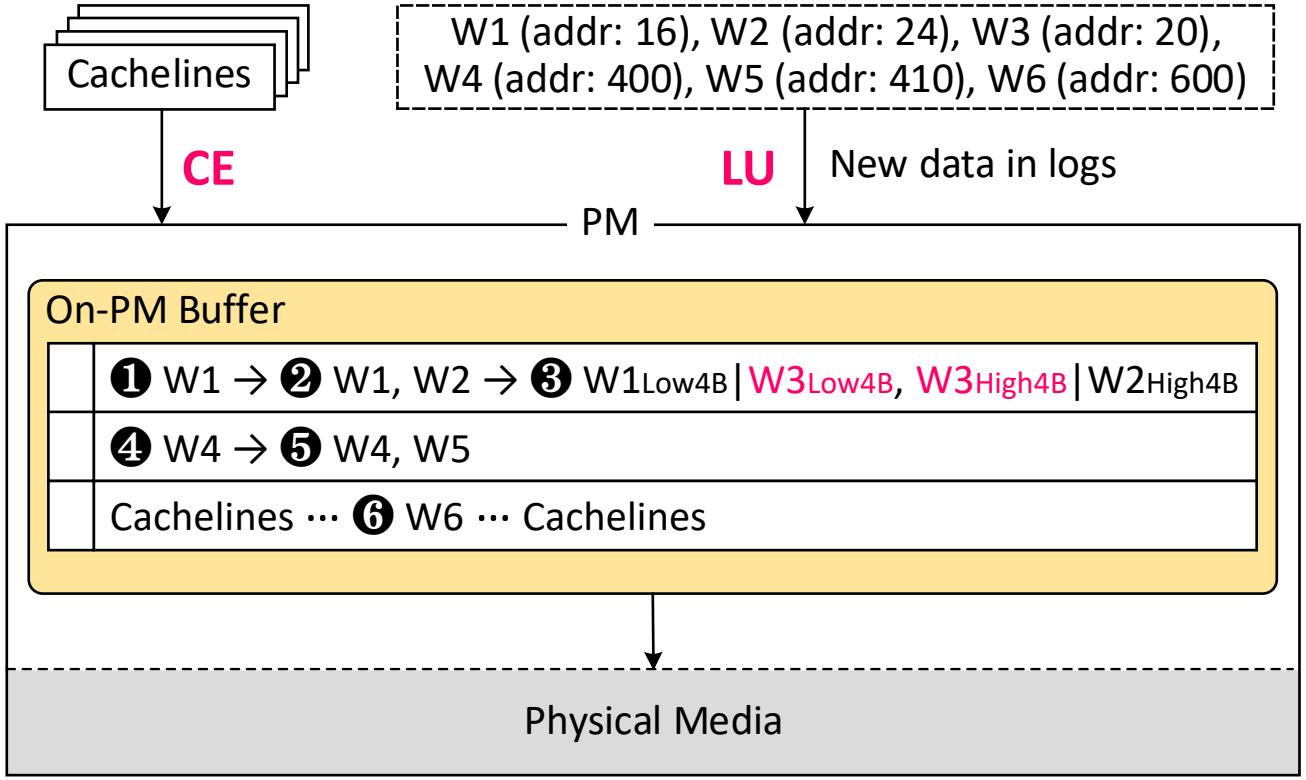
Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer
  - W1-W3 have overlapped bytes
  - W4-W5 are not overlapped



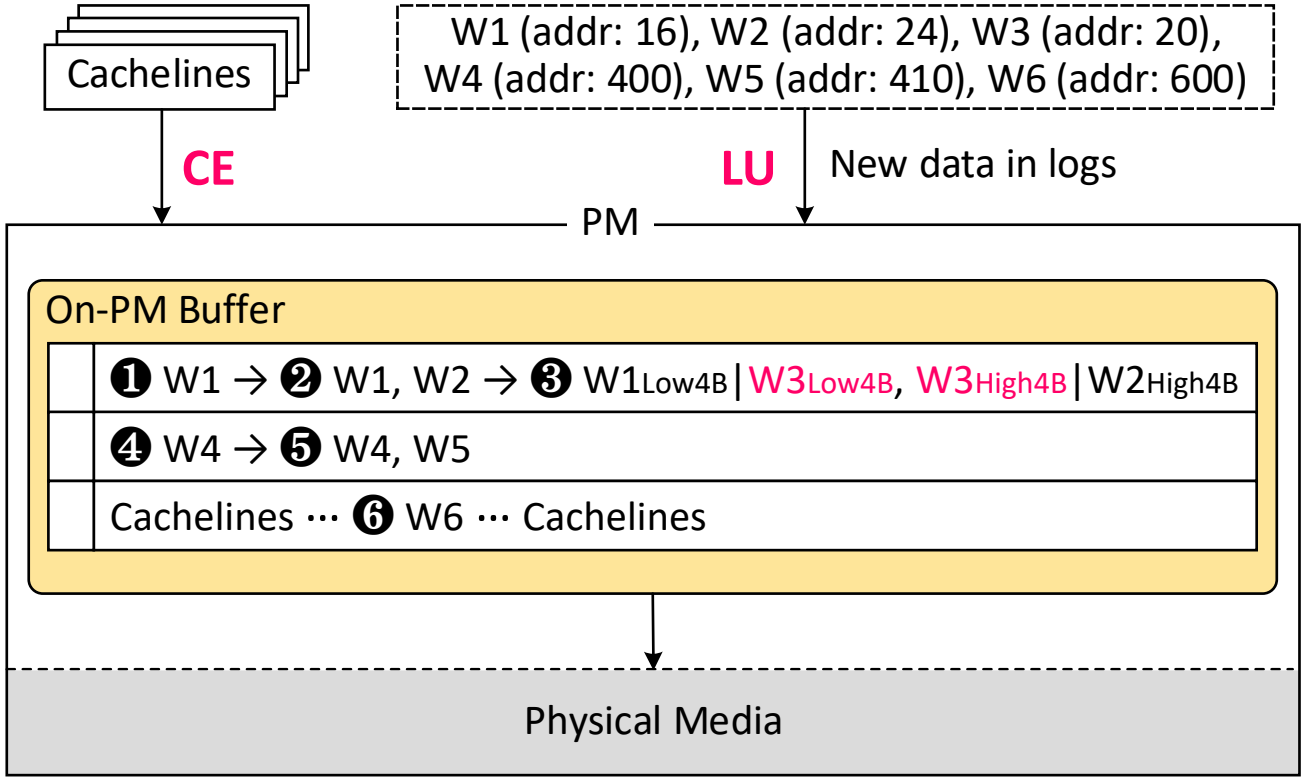
Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer
  - W1-W3 have overlapped bytes
  - W4-W5 are not overlapped
  - W6 is merged into cachelines



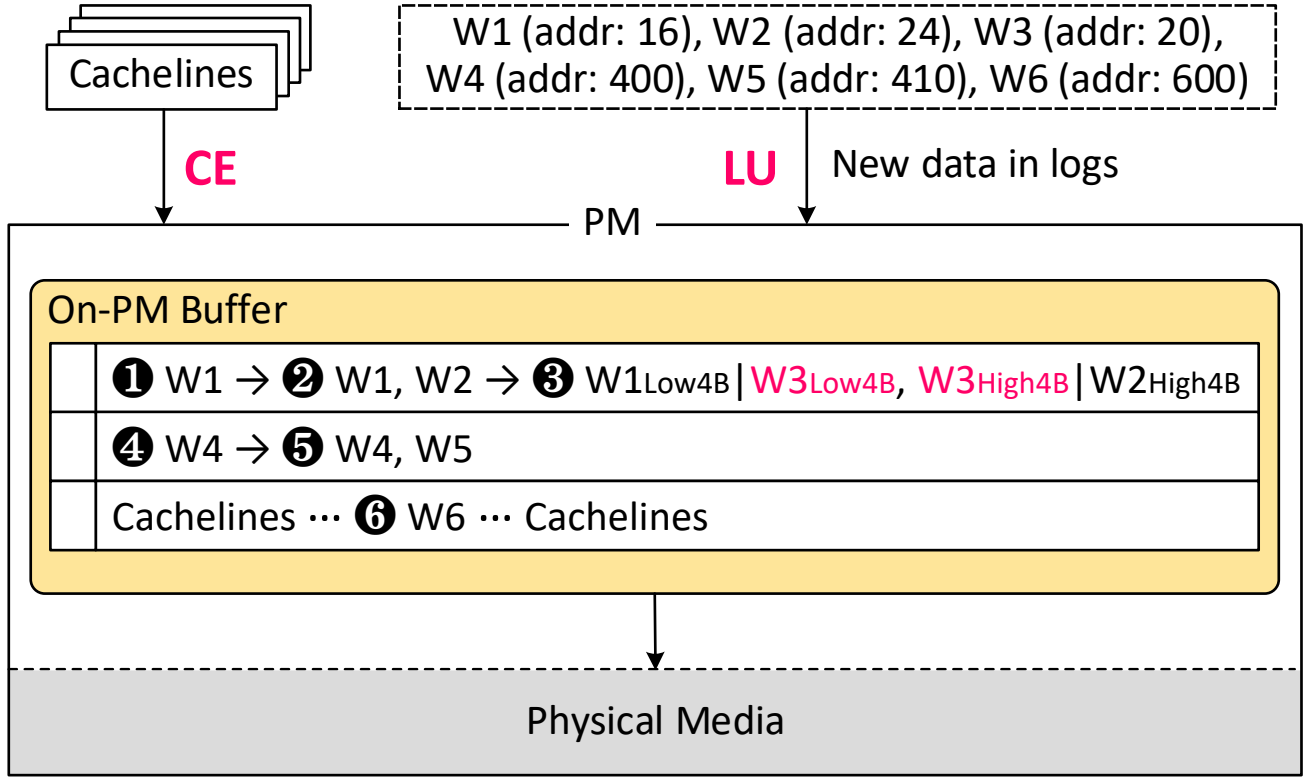
Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer
  - `W1-W3` have overlapped bytes
  - `W4-W5` are not overlapped
  - `W6` is merged into cachelines
- ***Correctness:*** No race risk



Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer
  - W1-W3 have overlapped bytes
  - W4-W5 are not overlapped
  - W6 is merged into cachelines
- ***Correctness:*** No race risk

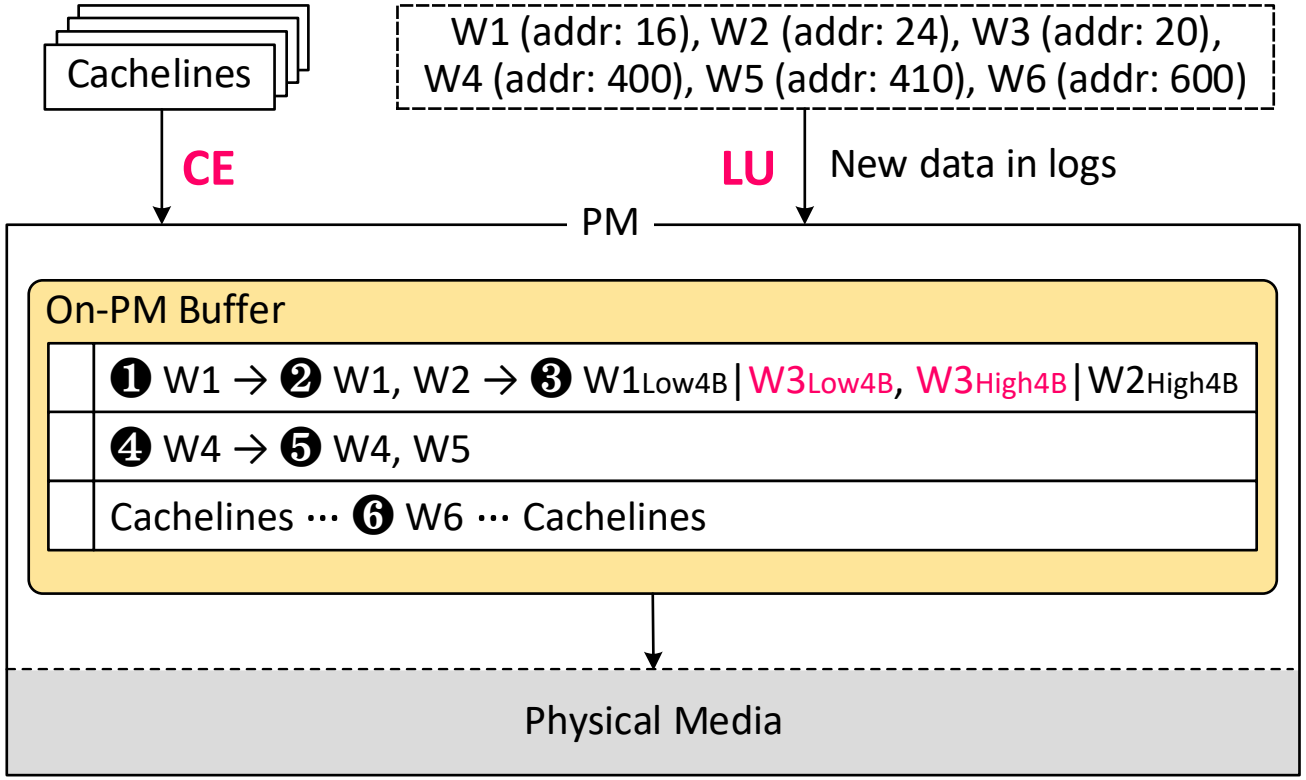


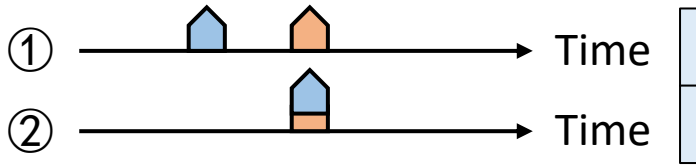
① Flush-bit in log is 1. CE updates the data region

LU (at txn commit)
CE

Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer
  - W1-W3 have overlapped bytes
  - W4-W5 are not overlapped
  - W6 is merged into cachelines
- ***Correctness:*** No race risk





① Flush-bit in log is 1. CE updates the data region

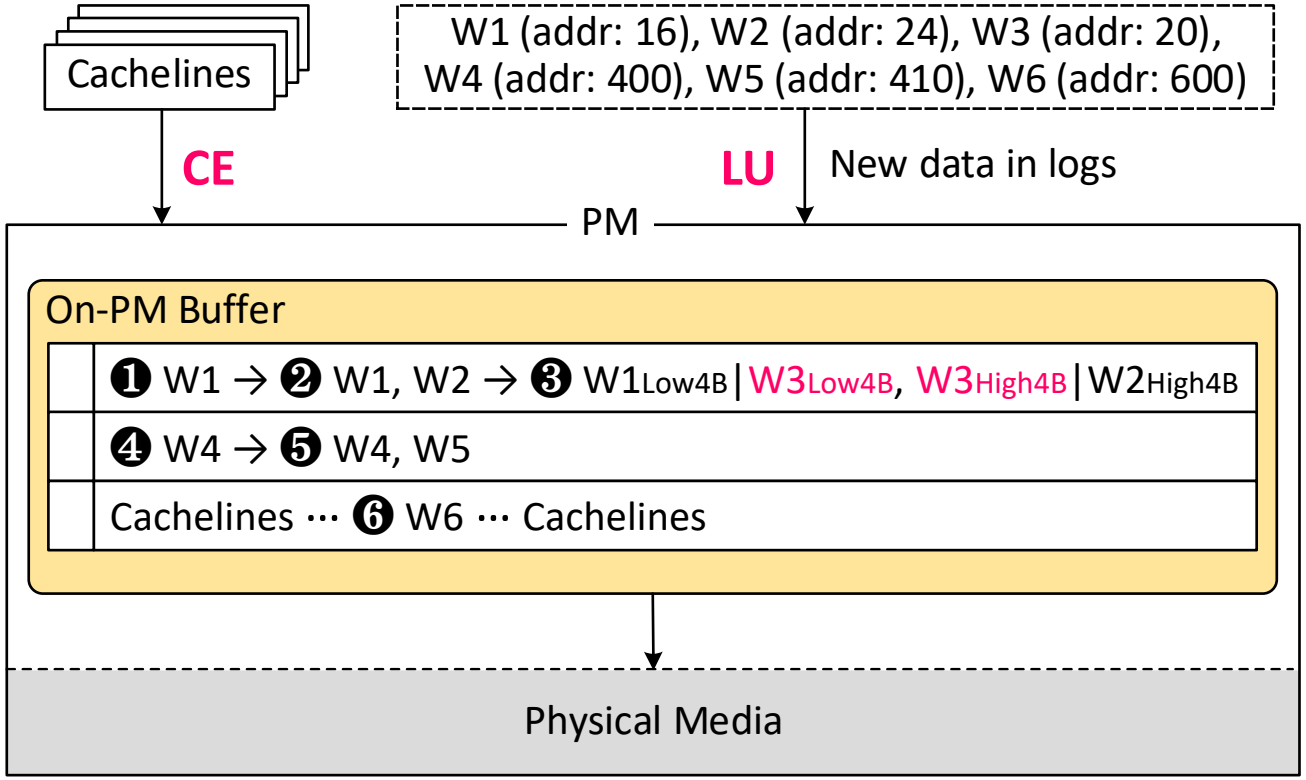
② LU and CE are coalesced to update the data region

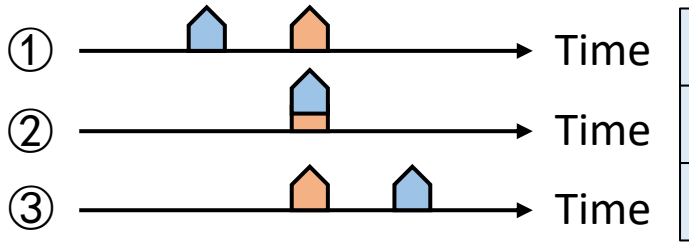
LU (at txn commit)

CE

Write Coalescing

- Silo allows two update paths
  - **8B: L**og in-place **U**pdates (**LU**)
  - **64B: C**acheline **E**victions (**CE**)
- LU and CE are coalesced in an on-PM buffer
  - W1-W3 have overlapped bytes
  - W4-W5 are not overlapped
  - W6 is merged into cachelines
- ***Correctness:*** No race risk





① Flush-bit in log is 1. CE updates the data region

② LU and CE are coalesced to update the data region

③ LU writes the data region. CE will not write twice*

LU (at txn commit)

CE

* By using bit-level write reduction schemes, e.g., DCW@ISCA'09

Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

Log Overflow

System Crash

Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity

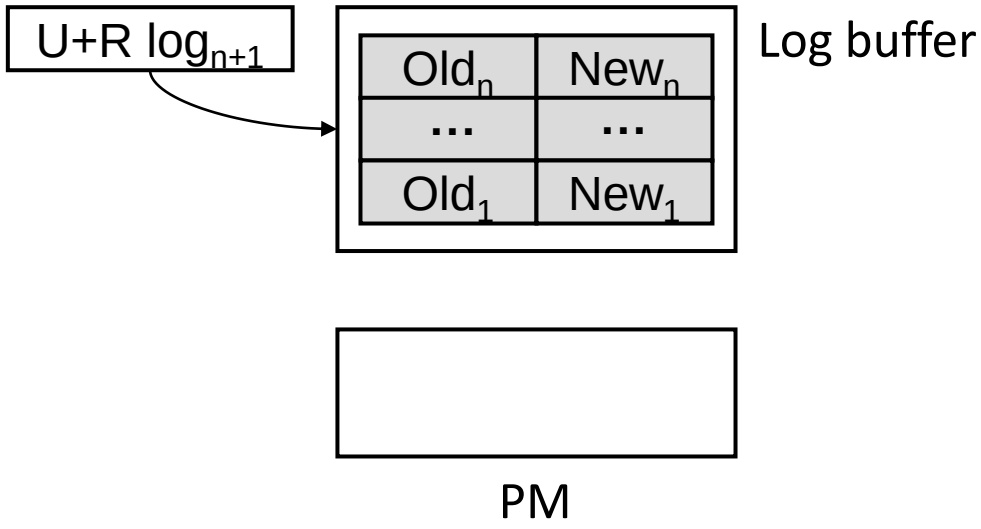
System Crash

Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



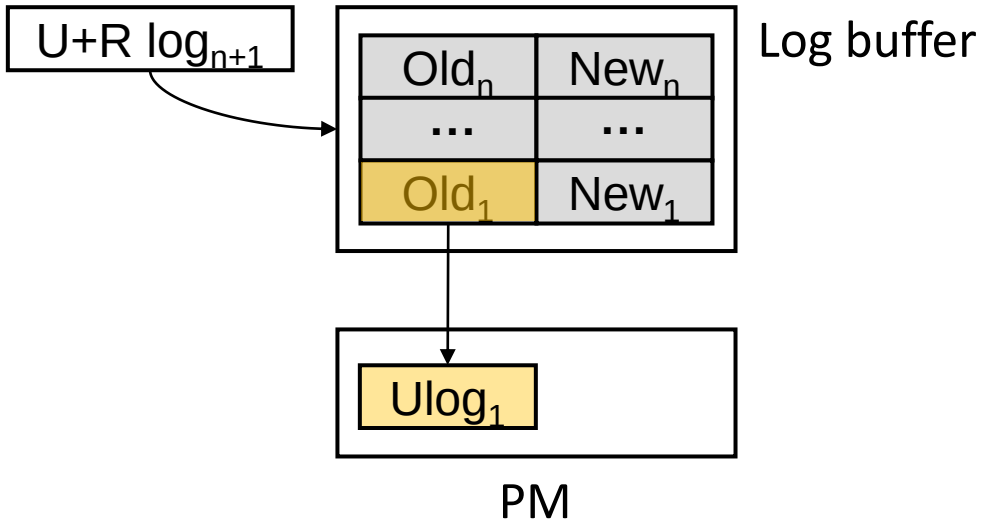
System Crash

Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



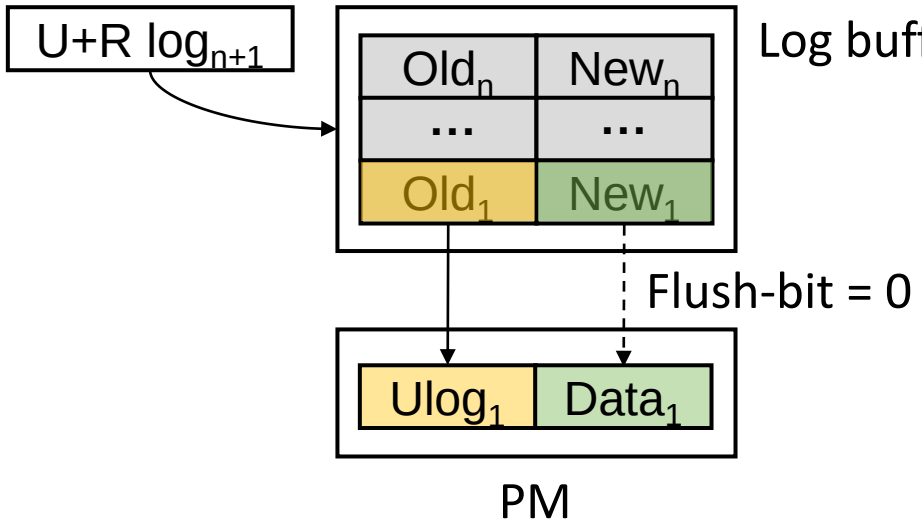
System Crash

Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



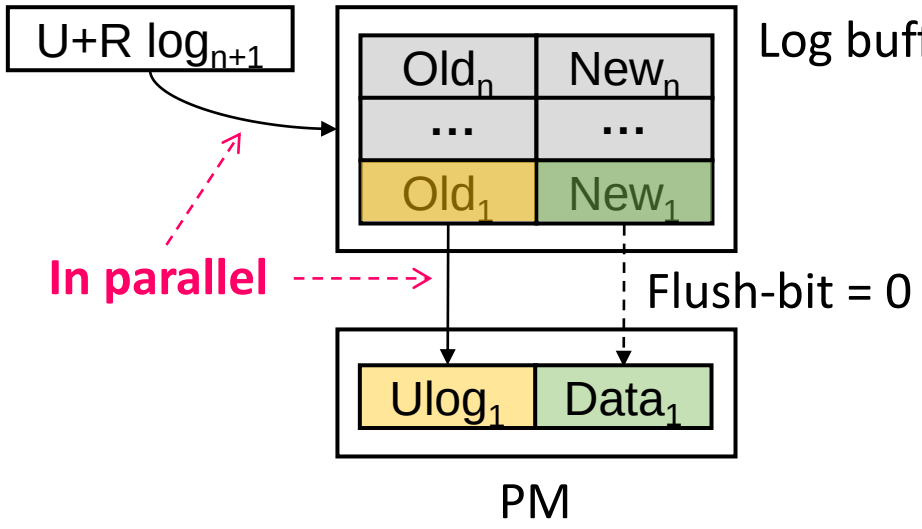
System Crash

Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



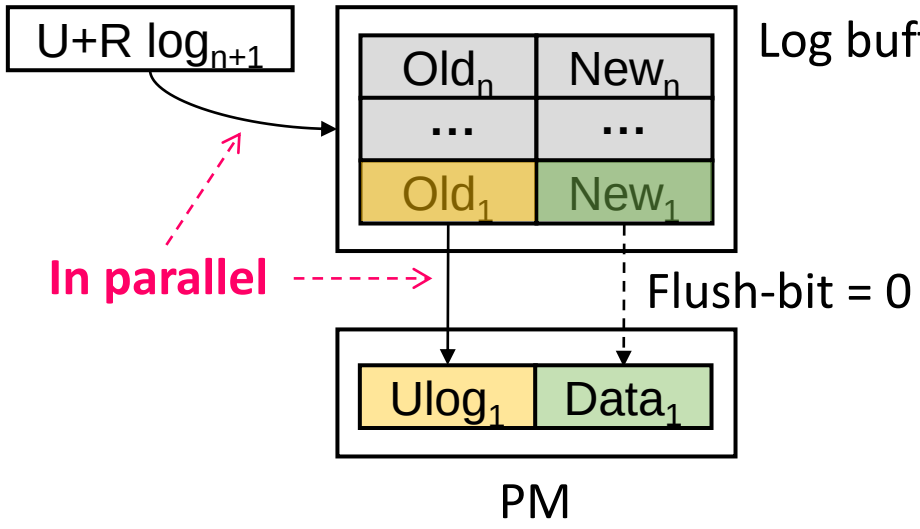
System Crash

Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

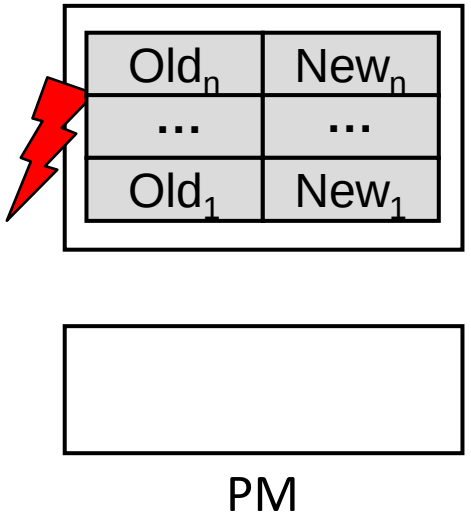
Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



System Crash

A system crash or power failure occurs → Selectively flush logs on demand

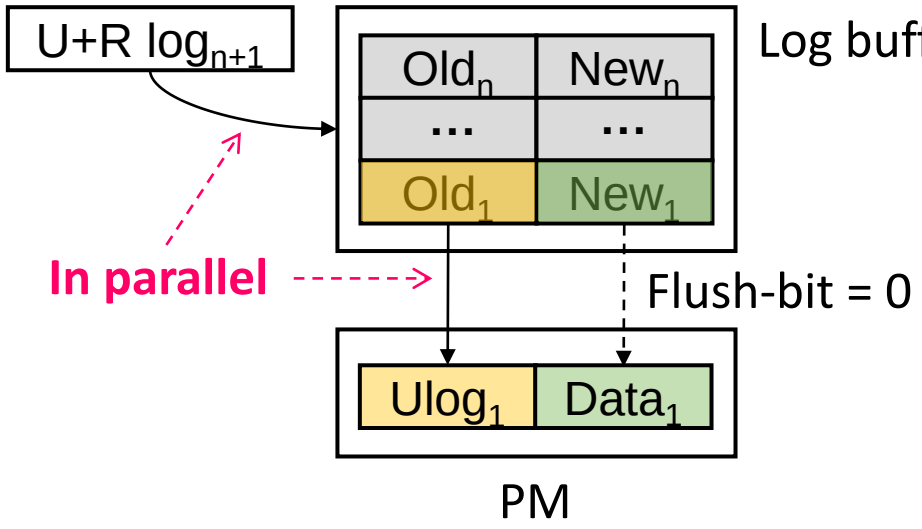


Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

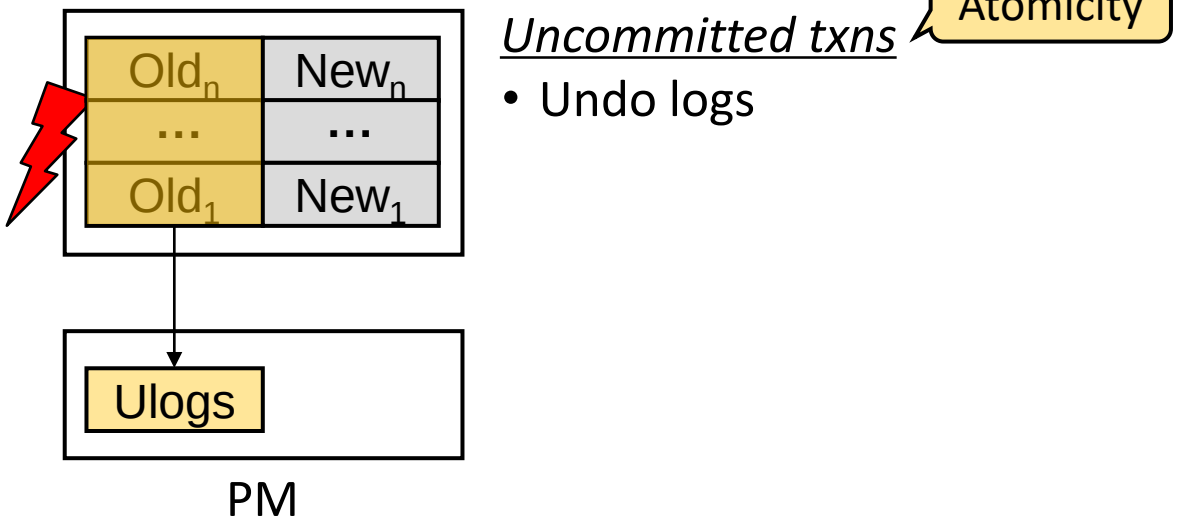
Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



System Crash

A system crash or power failure occurs → Selectively flush logs on demand

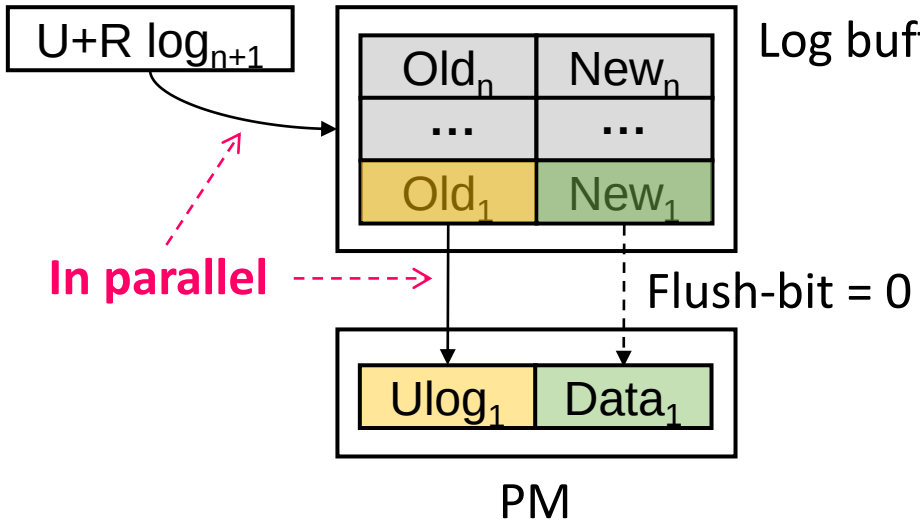


Rare Cases

- Silo writes logs to guarantee correctness in two rare cases

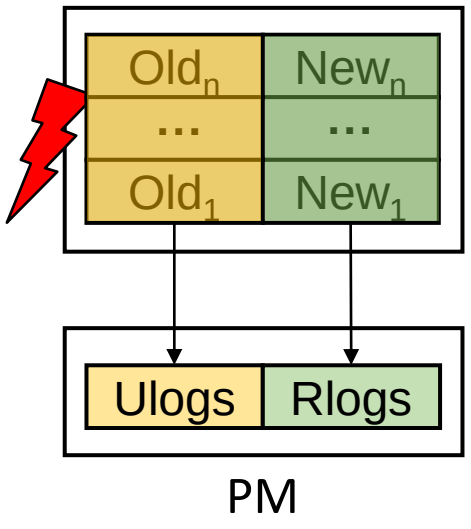
Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



System Crash

A system crash or power failure occurs → Selectively flush logs on demand



Uncommitted txns — Atomicity
- Undo logs

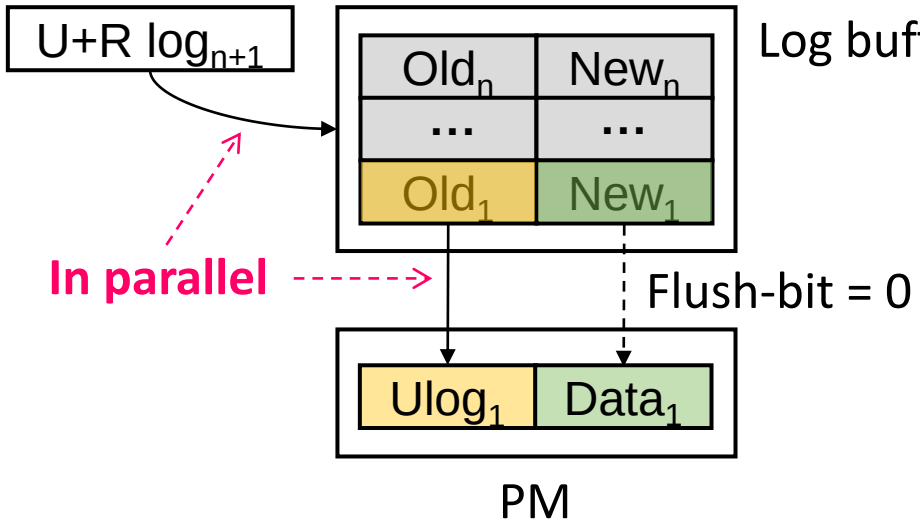
Committed txns — Durability
- Redo logs (flush-bit = 0)

Rare Cases

➢ Silo writes logs to guarantee correctness in two rare cases

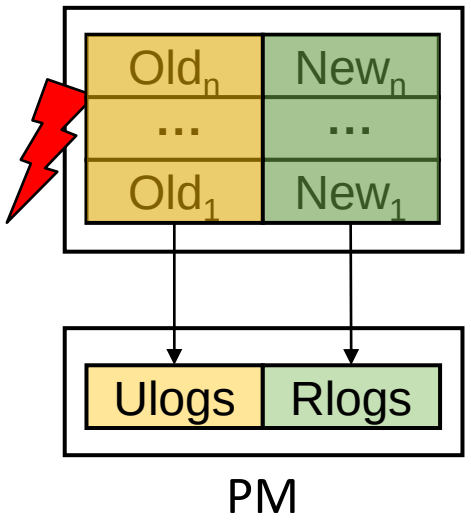
Log Overflow

The log buffer cannot hold all logs in one txn → Flush undo logs to ensure atomicity



System Crash

A system crash or power failure occurs → Selectively flush logs on demand



Uncommitted txns — Atomicity
- Undo logs
- Revoke

Committed txns — Durability
- Redo logs (flush-bit = 0)
- Replay

Recoverability

Evaluation

➢ ***Benchmarks***

- Micro-benchmarks
  - Array, Btree, Hash, Queue, RBtree
- Macro-benchmarks
  - TPCC, YCSB

➢ ***Comparisons***

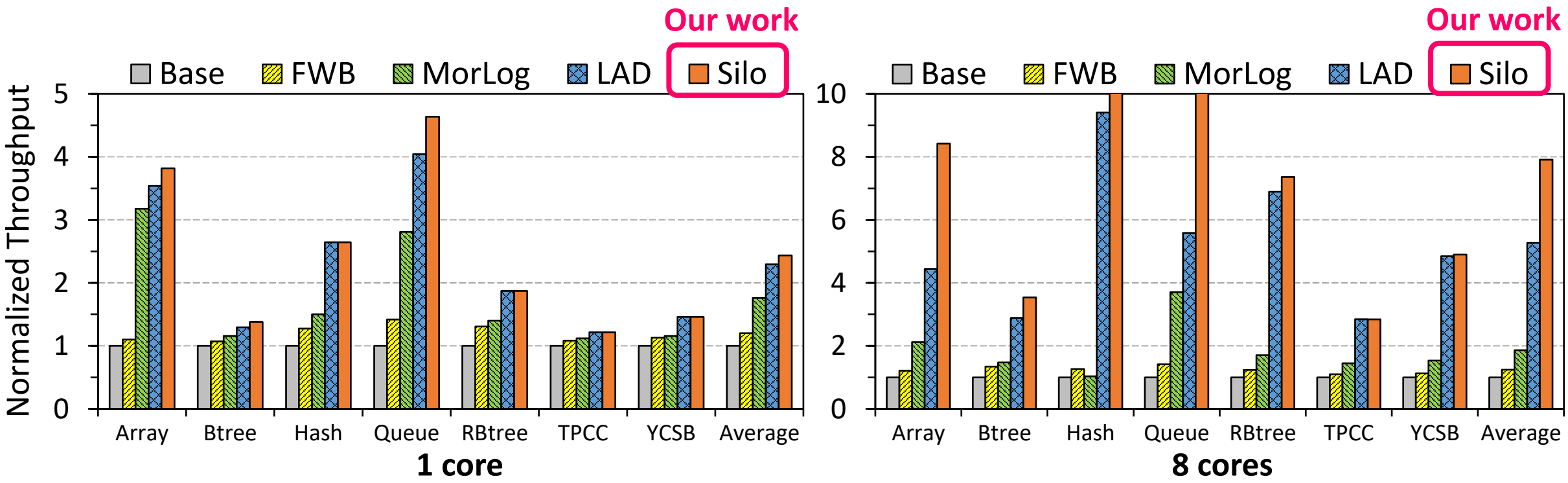
- **Base:** A hardware logging baseline
- **FWB**[2]**:** The hardware logging design of FWB
- **MorLog**[3]: The morphable hardware logging
- **LAD**[4]**:** The logless atomic durability design
- **Silo**: Our speculative logging design

Gem5 Simulation

Processor	
Cores	8 cores, x86-64, 2 GHz
L1 I/D	Private, 64B per line, 32KB, 8-way, 4 cycles
L2	Private, 64B per line, 256KB, 8-way, 12 cycles
LLC	Shared, 64B per line, 8MB, 16-way, 28 cycles
Mem Ctrl	FRFCFS, 64-entry queue in ADR domain
Log Buffer	680B per core, FIFO, 8 cycles, battery-backed
Persistent Memory	
Capacity	16GB phase-change memory
Latency	Read / Write: 50 / 150 ns[1]

[1] HOOP@ISCA'20 [2] FWB@HPCA'18 [3] MorLog@ISCA'20 [4] LAD@MICRO'19

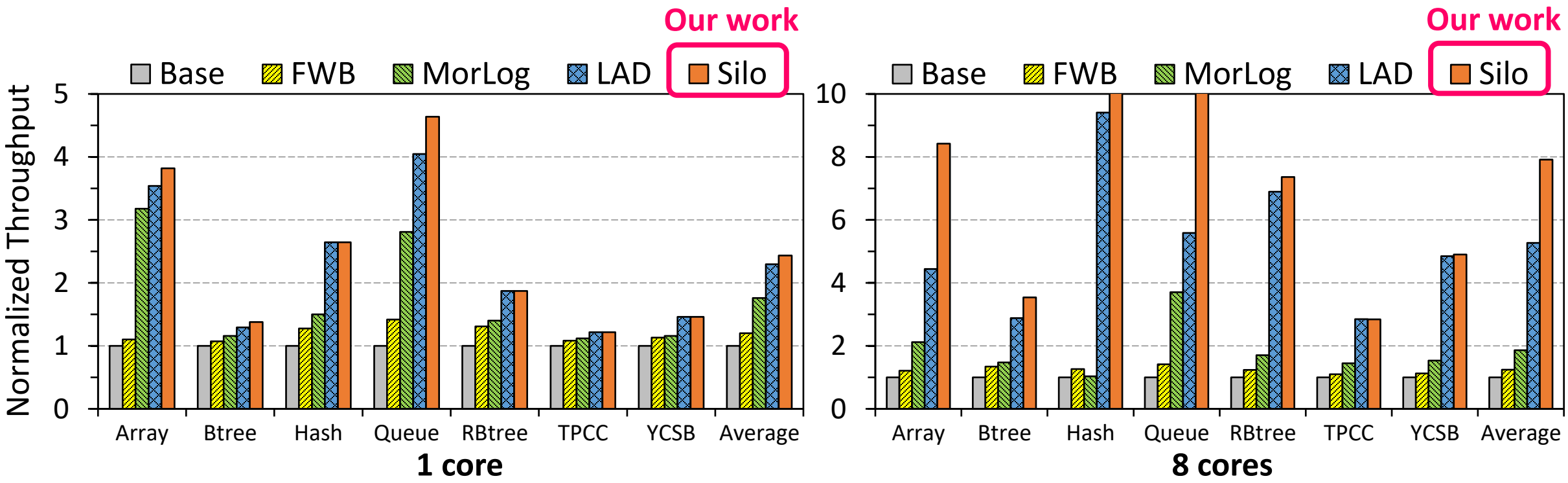
Transaction Throughput



Silo improves throughput	1 core	8 cores
Existing hardware logging designs	**1.4x**	**4.3x**
Existing hardware logless design (LAD)	**1.1x**	**1.5x**

Transaction Throughput

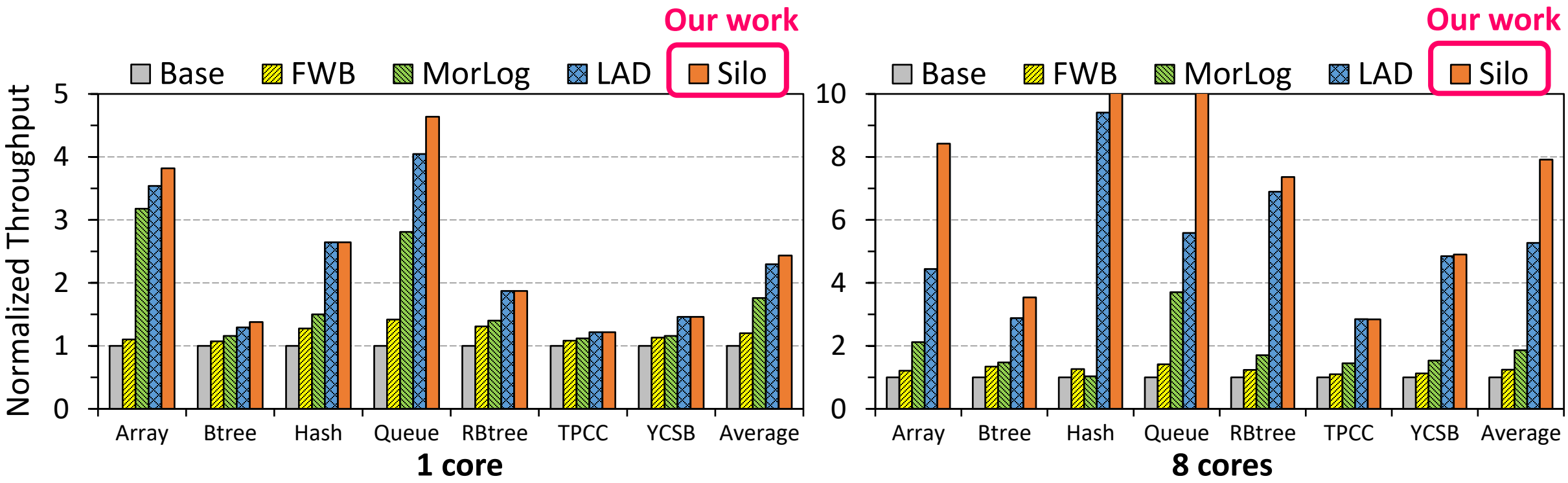
"Log as data": No ordering constraints
Do not wait to persist logs and cachelines



Silo improves throughput	1 core	8 cores
Existing hardware logging designs	**1.4x**	**4.3x**
Existing hardware logless design (LAD)	**1.1x**	**1.5x**

Transaction Throughput

"Log as data": No ordering constraints
Do not wait to persist logs and cachelines

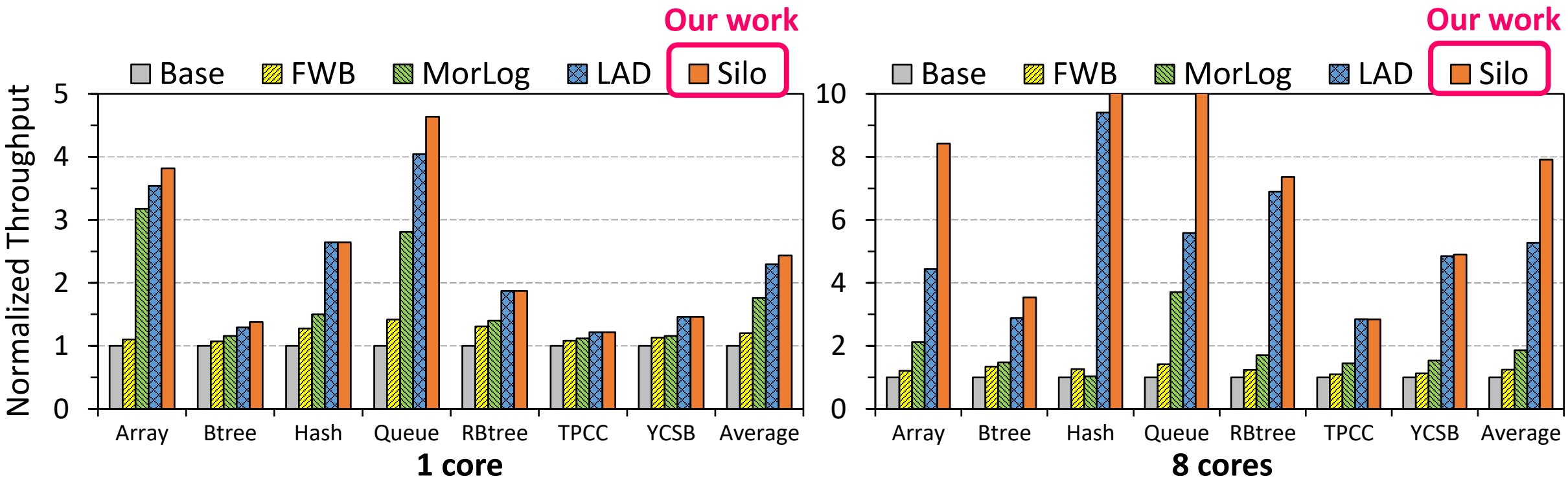


Wait to persist logs and cachelines

Silo improves throughput	1 core	8 cores
Existing hardware logging designs	**1.4x**	**4.3x**
Existing hardware logless design (LAD)	**1.1x**	**1.5x**

Transaction Throughput

"Log as data": No ordering constraints
Do not wait to persist logs and cachelines

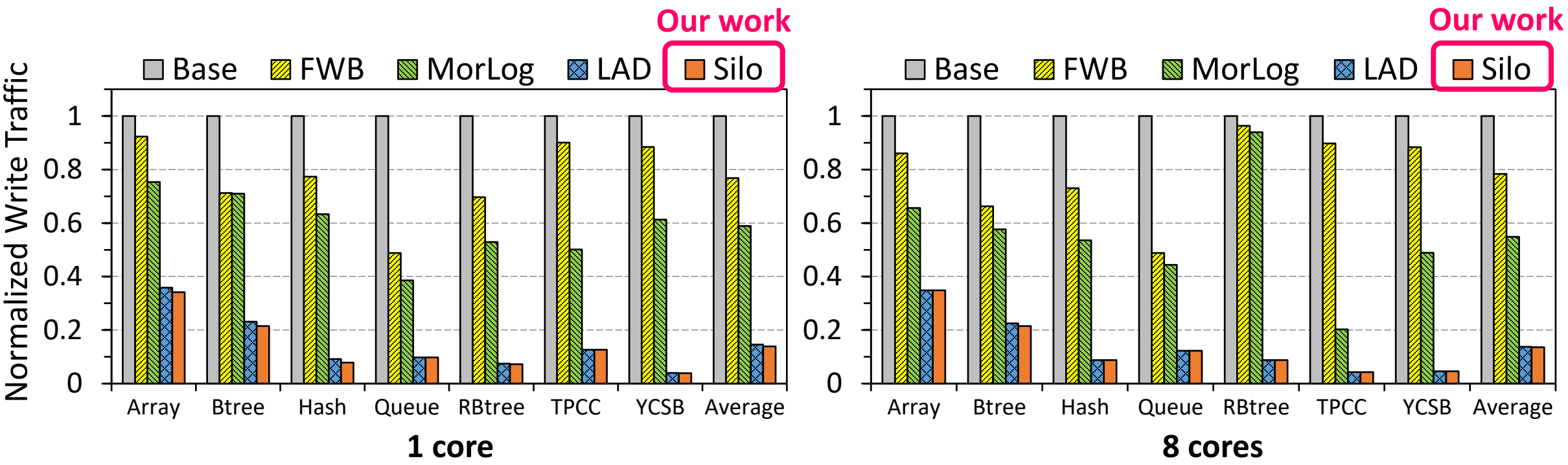


Wait to persist logs and cachelines

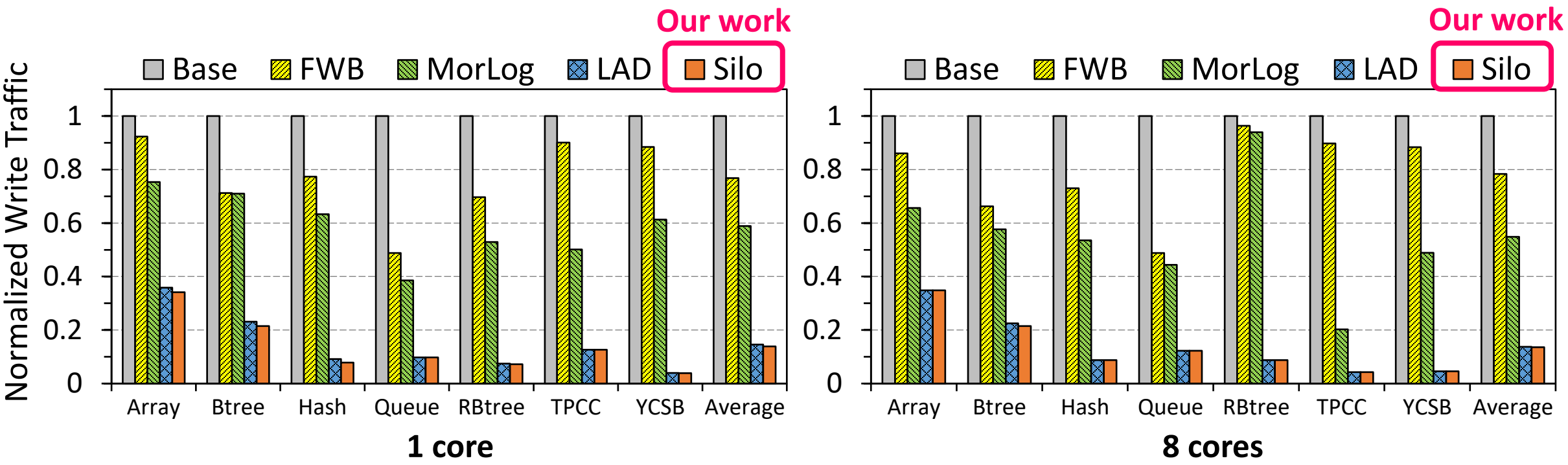
Wait to persist cachelines:
L1 → LLC → MC

Silo improves throughput	1 core	8 cores
Existing hardware logging designs	**1.4x**	**4.3x**
Existing hardware logless design (LAD)	**1.1x**	**1.5x**

Write Traffic



Write Traffic

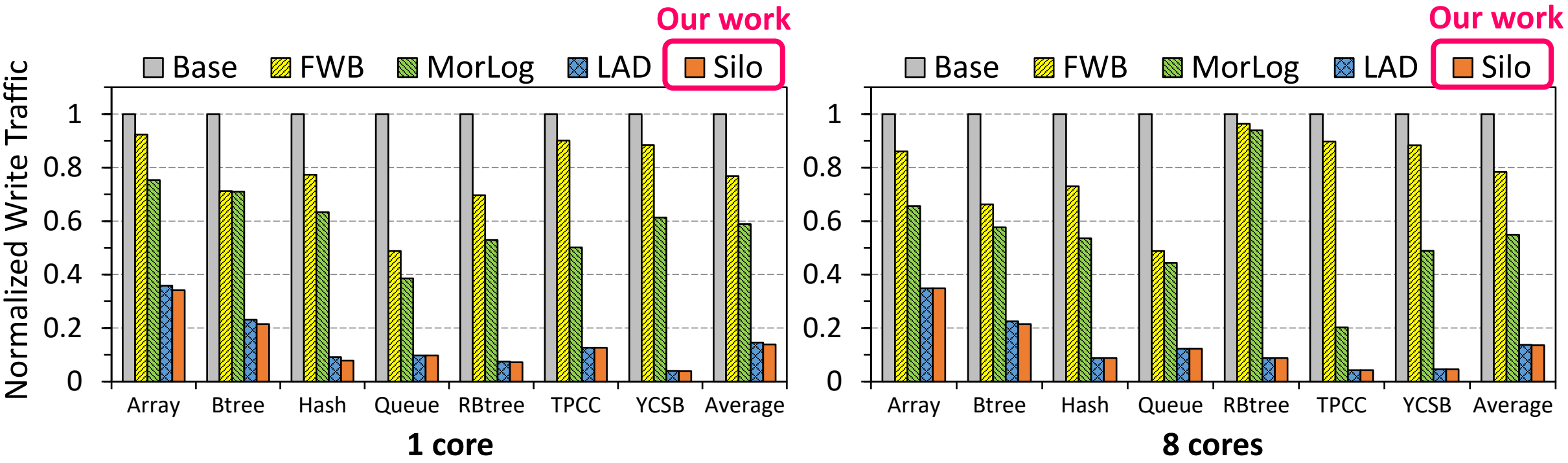


Logs are used to update data

Write logs and cachelines

Silo reduces write traffic by **76.5%** over existing hardware logging designs

Write Traffic



Logs are used to update data

Write logs and cachelines

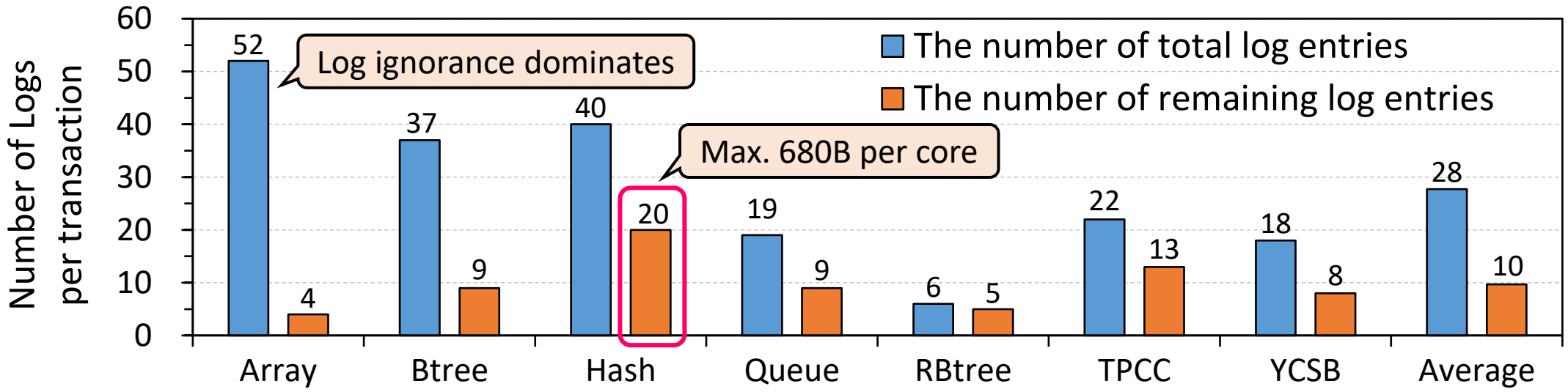
Silo reduces write traffic by **76.5%** over existing hardware logging designs

Write coalescing

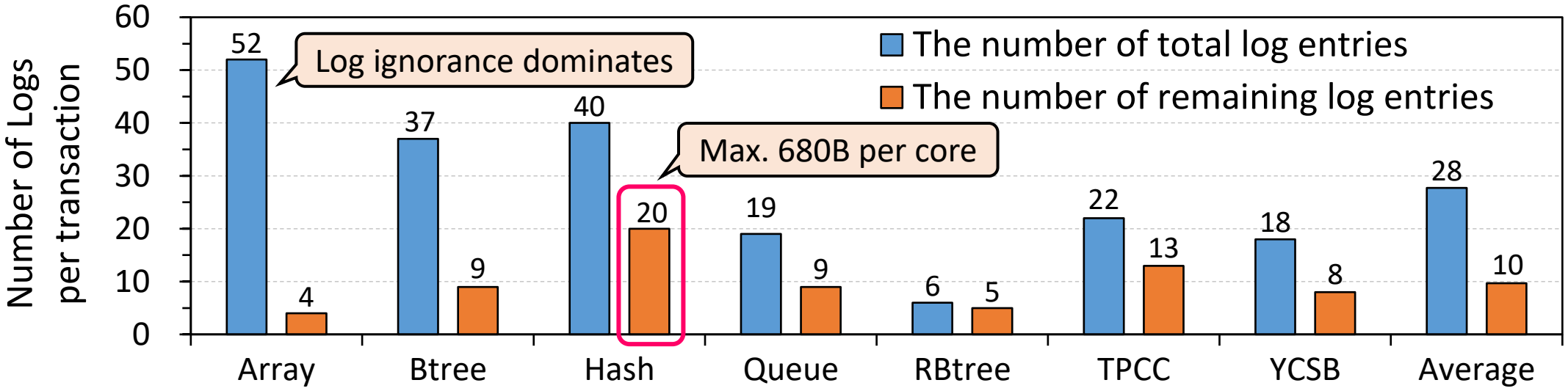
Silo exhibits approximate write traffic with LAD

Do not produce logs

Overhead of Log Buffer



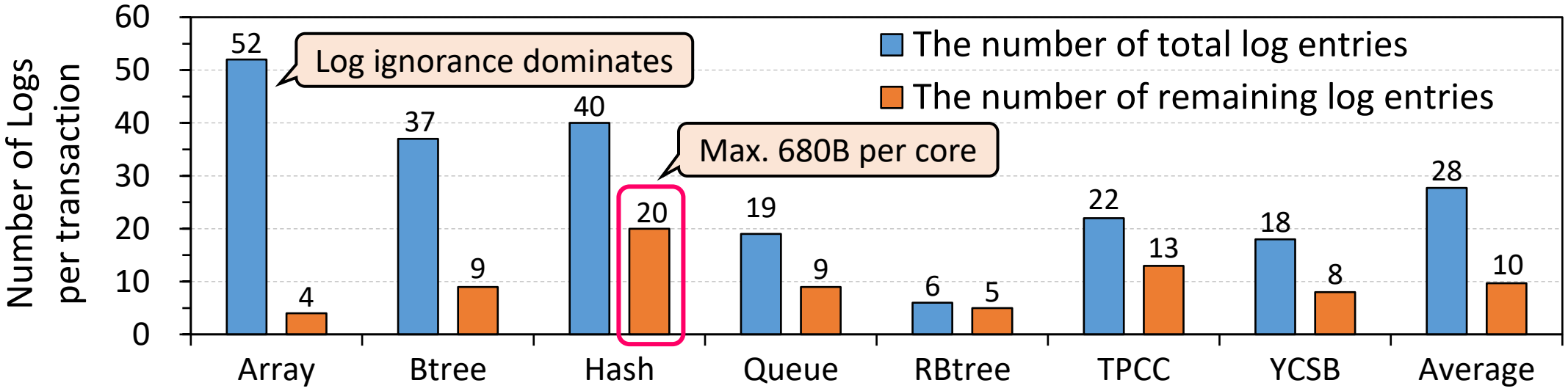
Overhead of Log Buffer



Battery consumption*	Intel's eADR	BBB@HPCA'21	Our Silo
Flush Size for 8 cores (KB)	10,496	16	5.3125
Flush Energy (μJ)	54,377	194	62
Supercapacitor (size: mm^3; area: mm^2)	151; 28.4	0.54; 0.66	0.17; 0.31
Lithium thin-film (size: mm^3; area: mm^2)	1.51; 1.32	0.0054; 0.031	0.0017; 0.014

* Based on the energy calculation model from BBB@HPCA'21

Overhead of Log Buffer



Battery consumption*	Intel's eADR	BBB@HPCA'21	Our Silo
Flush Size for 8 cores (KB)	10,496	16	
Flush Energy (µJ)	54,377	194	
Supercapacitor (size: mm^3; area: mm^2)	151; 28.4	0.54; 0.66	0.17; 0.31
Lithium thin-film (size: mm^3; area: mm^2)	1.51; 1.32	0.0054; 0.031	0.0017; 0.014

Smaller than
[eADR] **888.2x**; **91.6x**
[BBB] **3.2x**; **2.1x**

* Based on the energy calculation model from BBB@HPCA'21

More Results

- Handle large transactions
  - Log overflow occurs
  - Throughput decreases by only **7.4%**
- Change latency of log buffer
  - A 128-cycle log buffer only decreases the throughput by **3.3%** over an 8-cycle one

Find more details in our paper!

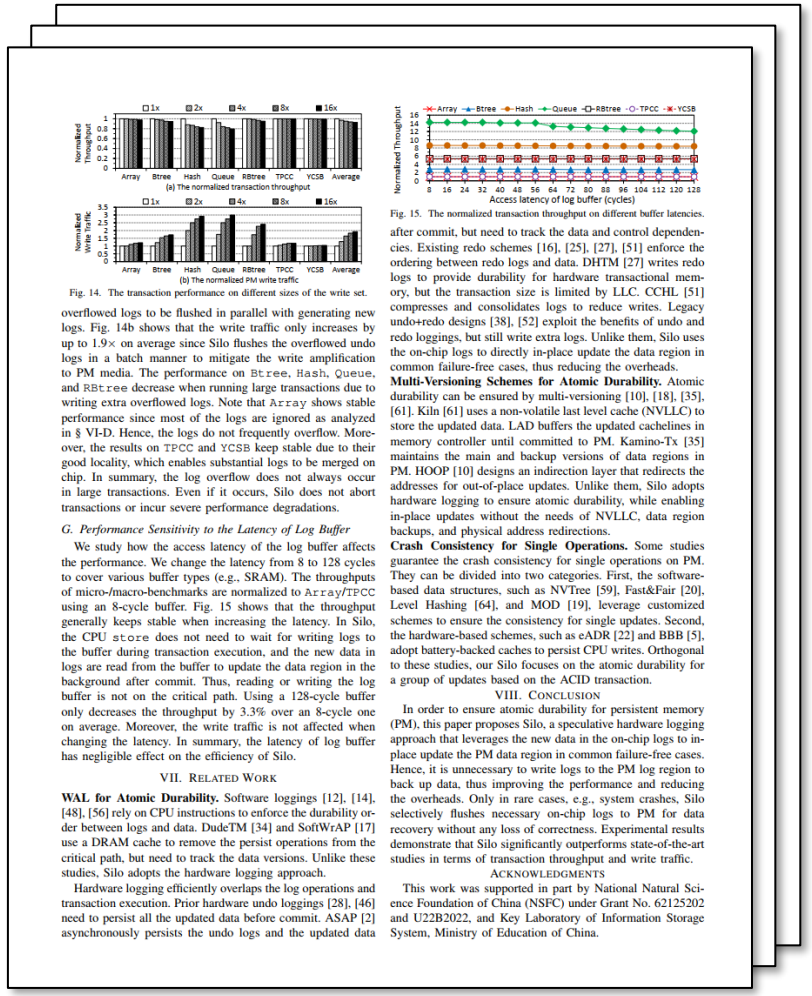


Fig. 14. The transaction performance on different sizes of the write set.

Fig. 15. The normalized transaction throughput on different buffer latencies.

overflowed logs to be flushed in parallel with generating new logs. Fig. 14b shows that the write traffic only increases by up to 1.9× on average since Silo flushes the overflowed undo logs in a batch manner to mitigate the write amplification to PM media. The performance on `Btree`, `Hash`, `Queue`, and `RBtree` decrease when running large transactions due to writing extra overflowed logs. Note that `Array` shows stable performance since most of the logs are ignored as analyzed in § VI-D. Hence, the logs do not frequently overflow. Moreover, the results on `TPCC` and `YCSB` keep stable due to their good locality, which enables substantial logs to be merged on chip. In summary, the log overflow does not always occur in large transactions. Even if it occurs, Silo does not abort transactions or incur severe performance degradations.

G. Performance Sensitivity to the Latency of Log Buffer

We study how the access latency of the log buffer affects the performance. We change the latency from 8 to 128 cycles to cover various buffer types (e.g., SRAM). The throughputs of micro-/macro-benchmarks are normalized to `Array`/`TPCC` using an 8-cycle buffer. Fig. 15 shows that the throughput generally keeps stable when increasing the latency. In Silo, the CPU `store` does not need to wait for writing logs to the buffer during transaction execution, and the new data in logs are read from the buffer to update the data region in the background after commit. Thus, reading or writing the log buffer is not on the critical path. Using a 128-cycle buffer only decreases the throughput by 3.3% over an 8-cycle one on average. Moreover, the write traffic is not affected when changing the latency. In summary, the latency of log buffer has negligible effect on the efficiency of Silo.

VII. Related Work

WAL for Atomic Durability. Software loggings [12], [14], [48], [56] rely on CPU instructions to enforce the durability order between logs and data. DudeTM [34] and SoftWrAP [17] use a DRAM cache to remove the persist operations from the critical path, but need to track the data versions. Unlike these studies, Silo adopts the hardware logging approach.

Hardware logging efficiently overlaps the log operations and transaction execution. Prior hardware undo loggings [28], [46] need to persist all the updated data before commit. ASAP [2] asynchronously persists the undo logs and the updated data after commit, but need to track the data and control dependencies. Existing redo schemes [16], [25], [27], [51] enforce the ordering between redo logs and data. DHTM [27] writes redo logs to provide durability for hardware transactional memory, but the transaction size is limited by LLC. CCHL [51] compresses and consolidates logs to reduce writes. Legacy undo+redo designs [38], [52] exploit the benefits of undo and redo loggings, but still write extra logs. Unlike them, Silo uses the on-chip logs to directly in-place update the data region in common failure-free cases, thus reducing the overheads.

Multi-Versioning Schemes for Atomic Durability. Atomic durability can be ensured by multi-versioning [10], [18], [35], [61]. Kiln [61] uses a non-volatile last level cache (NVLLC) to store the updated data. LAD buffers the updated cachelines in memory controller until committed to PM. Kamino-Tx [35] maintains the main and backup versions of data regions in PM. HOOP [10] designs an indirection layer that redirects the addresses for out-of-place updates. Unlike them, Silo adopts hardware logging to ensure atomic durability, while enabling in-place updates without the needs of NVLLC, data region backups, and physical address redirections.

Crash Consistency for Single Operations. Some studies guarantee the crash consistency for single operations on PM. They can be divided into two categories. First, the software-based data structures, such as NVTree [59], Fast&Fair [20], Level Hashing [64], and MOD [19], leverage customized schemes to ensure the consistency for single updates. Second, the hardware-based schemes, such as eADR [22] and BBB [5], adopt battery-backed caches to persist CPU writes. Orthogonal to these studies, our Silo focuses on the atomic durability for a group of updates based on the ACID transaction.

VIII. Conclusion

In order to ensure atomic durability for persistent memory (PM), this paper proposes Silo, a speculative hardware logging approach that leverages the new data in the on-chip logs to in-place update the PM data region in common failure-free cases. Hence, it is unnecessary to write logs to the PM log region to back up data, thus improving the performance and reducing the overheads. Only in rare cases, e.g., system crashes, Silo selectively flushes necessary on-chip logs to PM for data recovery without any loss of correctness. Experimental results demonstrate that Silo significantly outperforms state-of-the-art studies in terms of transaction throughput and write traffic.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 62125202 and U22B2022, and Key Laboratory of Information Storage System, Ministry of Education of China.

Conclusion

- Ensuring atomic durability becomes important for PM

Conclusion

- Ensuring atomic durability becomes important for PM
- Prior hardware logging studies: **Log as Backup**
  - Heavy writes to PM
  - Ordering constraints between persisting logs and data

Conclusion

- Ensuring atomic durability becomes important for PM
- Prior hardware logging studies: **Log as Backup**
  - Heavy writes to PM
  - Ordering constraints between persisting logs and data
- We propose a speculative logging design **Silo: Log as Data**
  - Use on-chip logs to in-place update data (*Make common case fast*)
  - Write logs to back up data in rare cases (*Guarantee recoverability*)

Conclusion

- Ensuring atomic durability becomes important for PM
- Prior hardware logging studies: **Log as Backup**
  - Heavy writes to PM
  - Ordering constraints between persisting logs and data
- We propose a speculative logging design **Silo: Log as Data**
  - Use on-chip logs to in-place update data (*Make common case fast*)
  - Write logs to back up data in rare cases (*Guarantee recoverability*)
- **Benefits**
  - Improve transaction throughput
  - Reduce write traffic to PM
  - Low hardware overhead

Thank you!