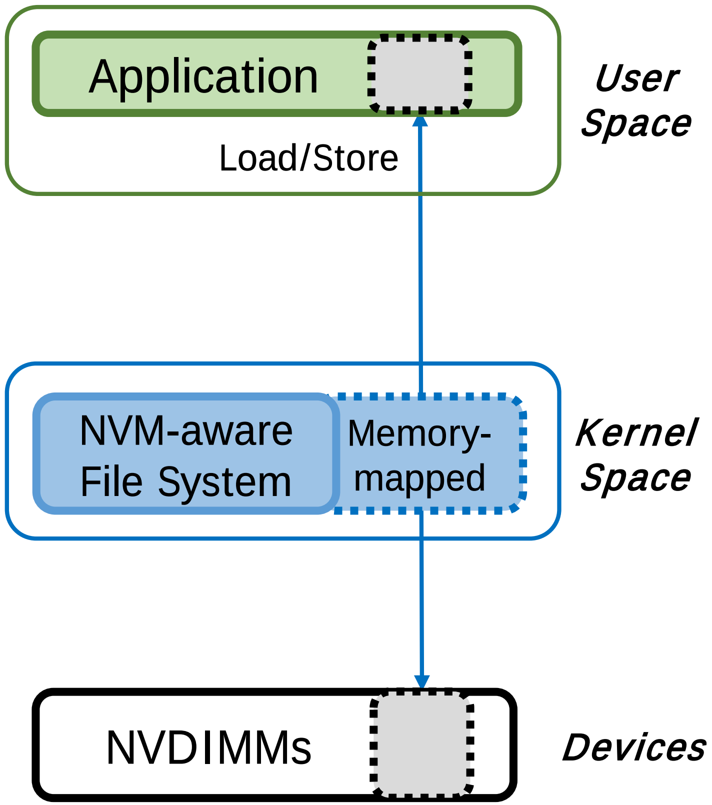
# Efficient Hardware-assisted Logging with Asynchronous and Direct Update for Persistent Memory

**Jungi Jeong**, Chang Hyun Park,
Jaehyuk Huh, and Seungryoul Maeng

KAIST School of Computing

International Symposium on Microarchitecture (MICRO) 2018

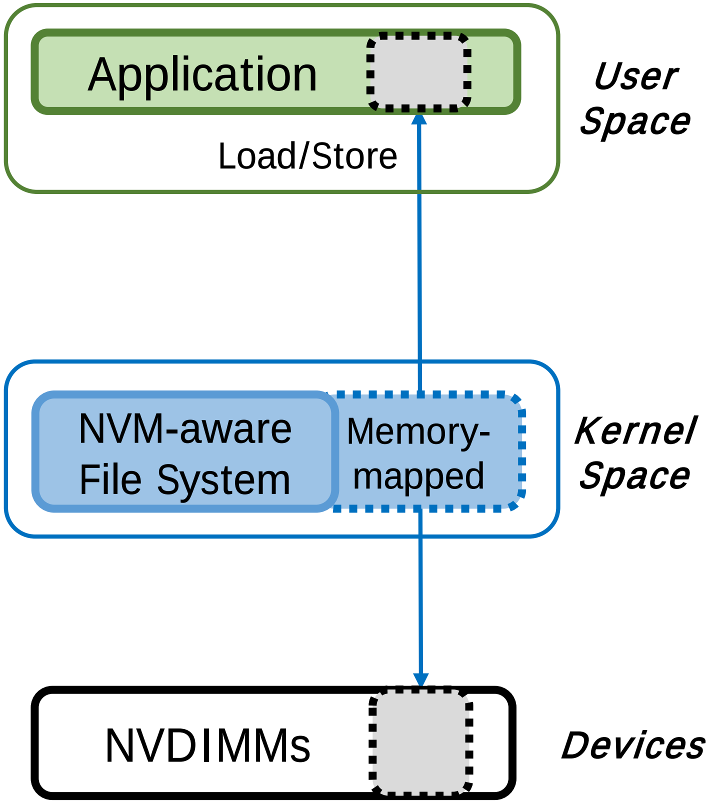
# Storage-Class Memory



# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

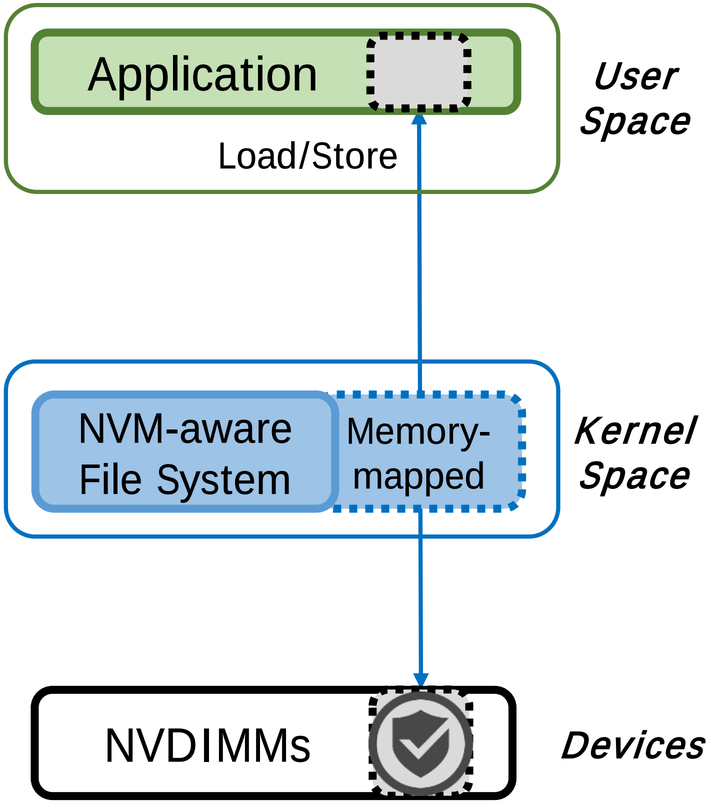
- Ex) Doubly linked-list insertion



# Storage-Class Memory

- Directly attached to the app's virtual address space
- Accessible through load/store instructions
- In-memory data persistency

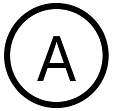
- Ex) Doubly linked-list insertion

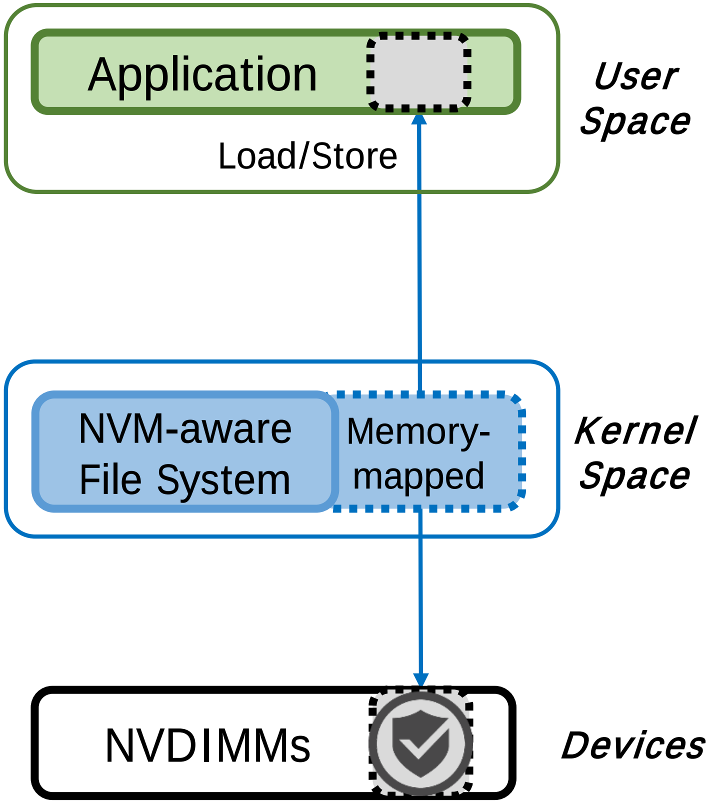


# Storage-Class Memory

- Directly attached to the app's virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion

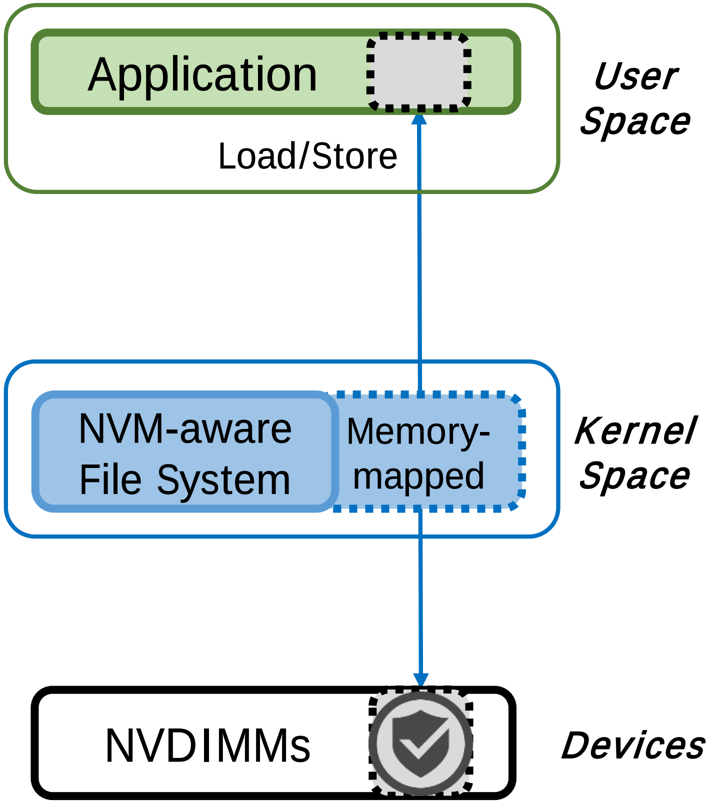




# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

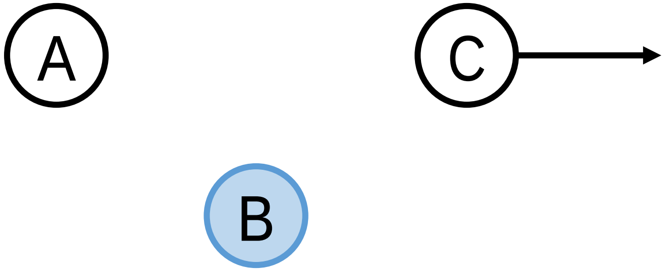
- Ex) Doubly linked-list insertion

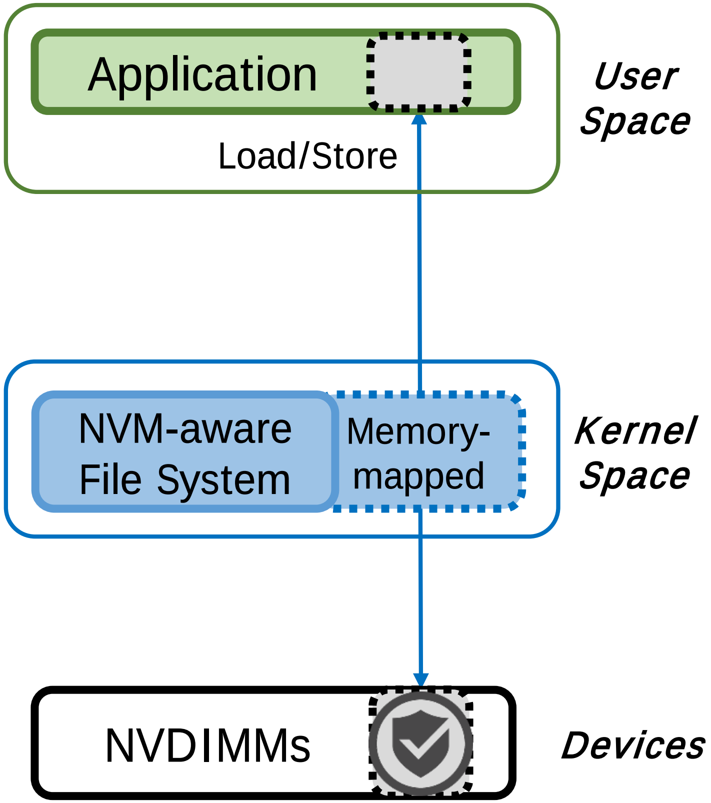


# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion

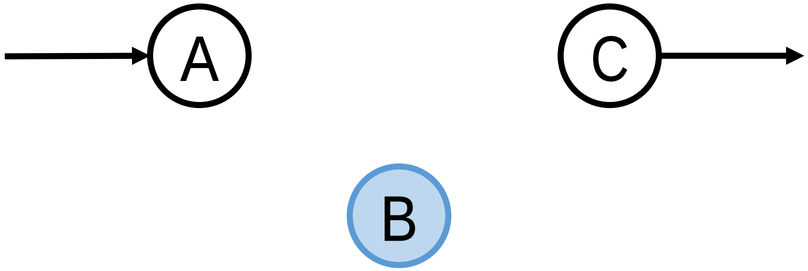


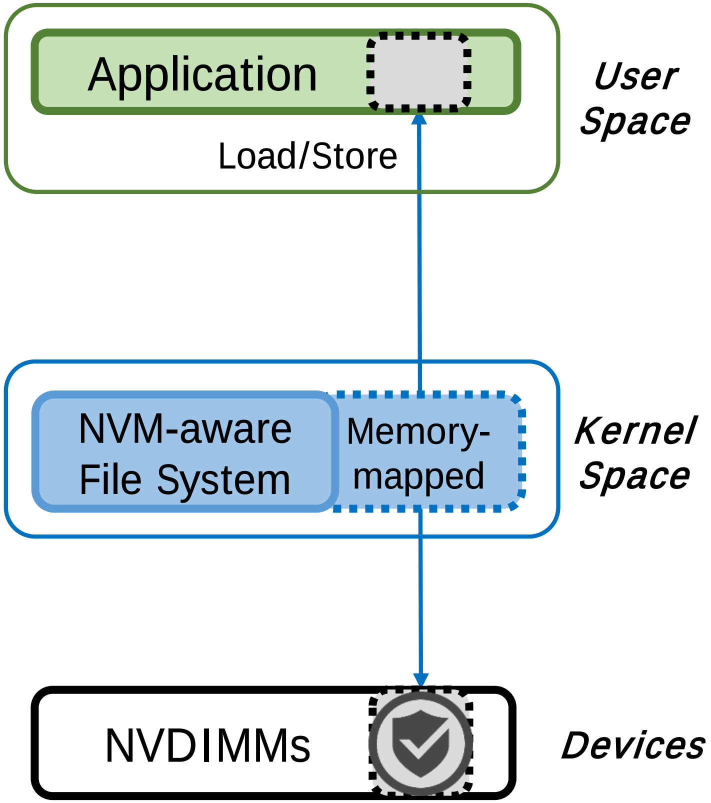


# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion

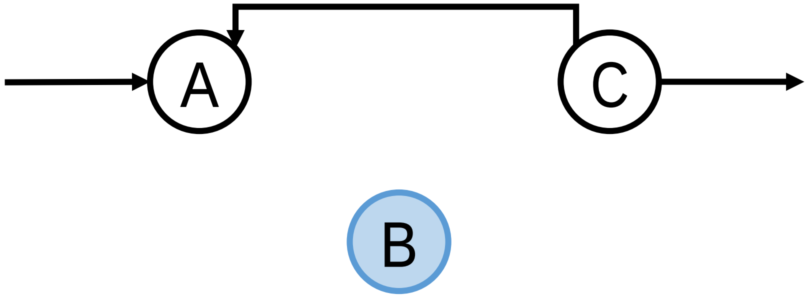


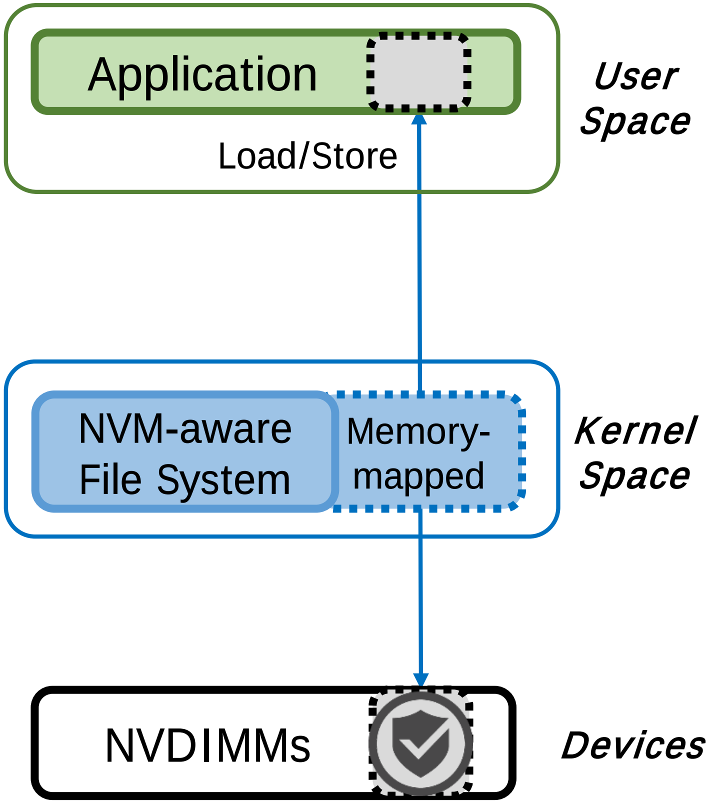


# Storage-Class Memory

- Directly attached to the app's virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion

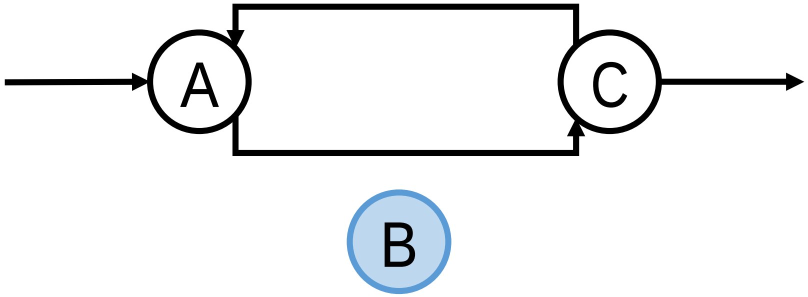




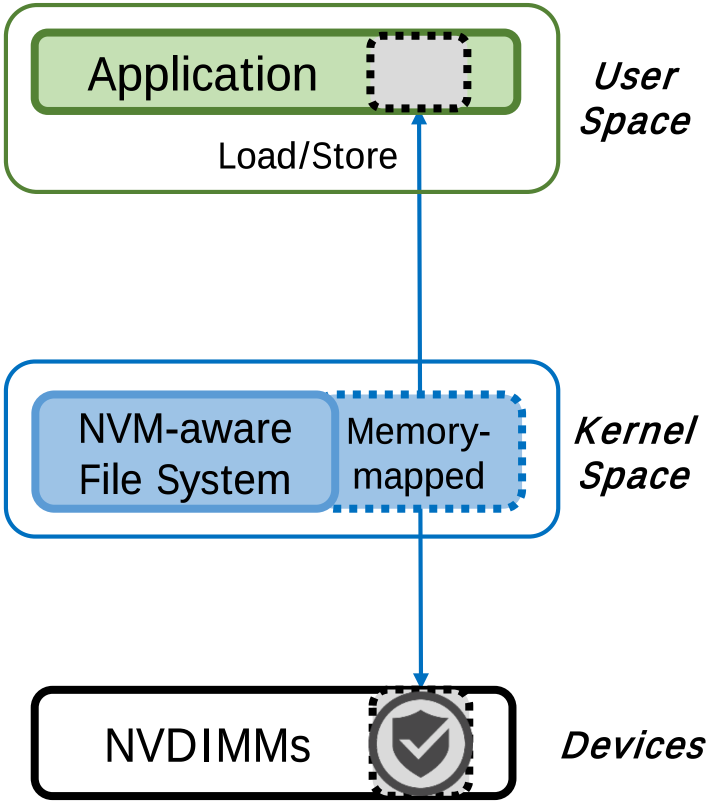
# Storage-Class Memory

- Directly attached to the app's virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



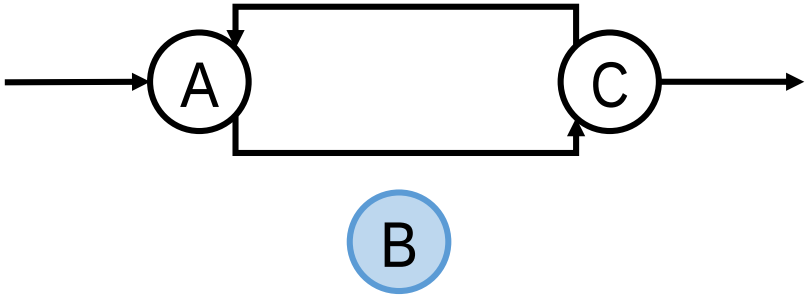
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



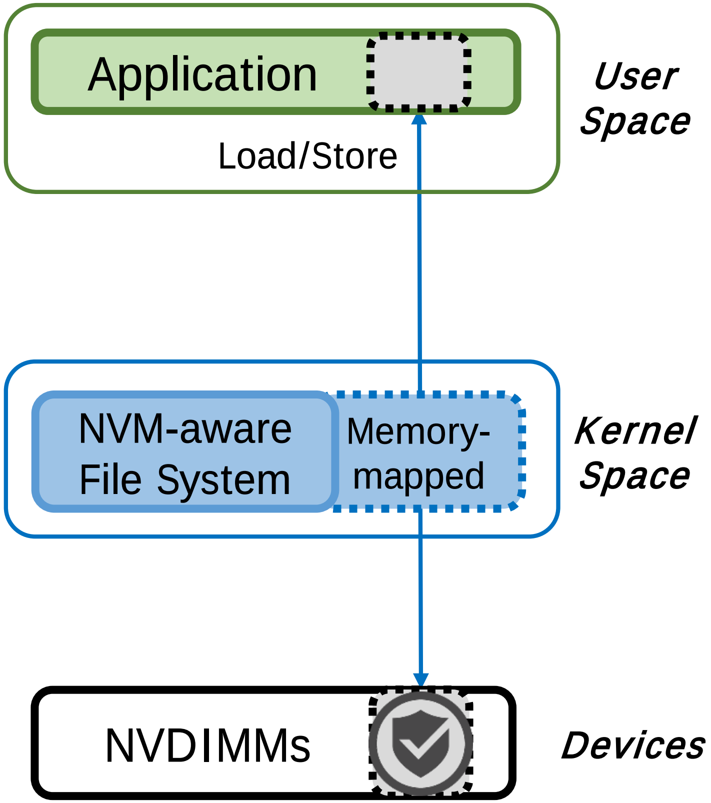
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



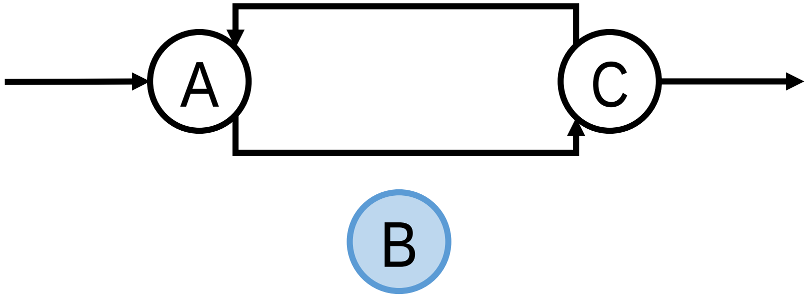
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



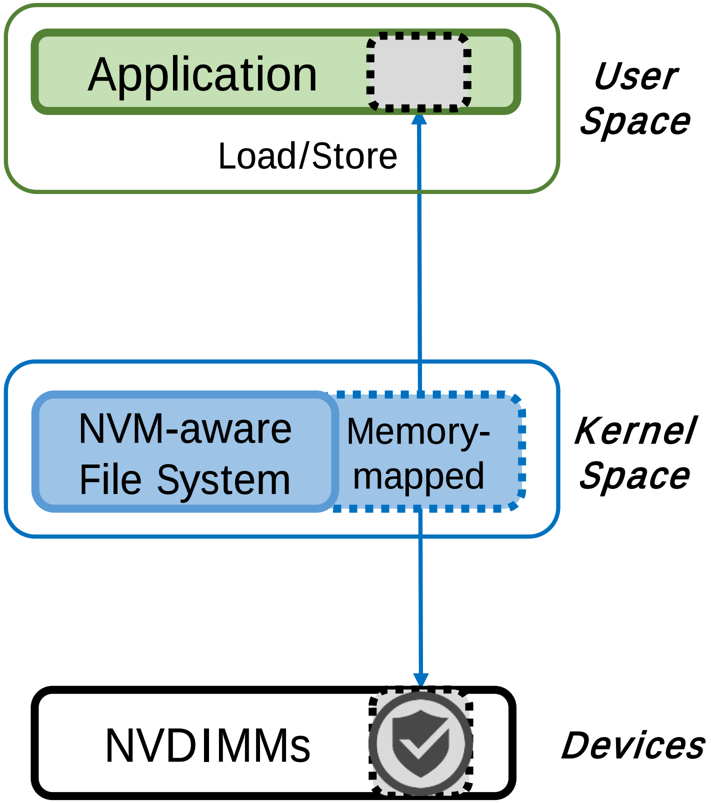
# Storage-Class Memory

- Directly attached to the app's virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



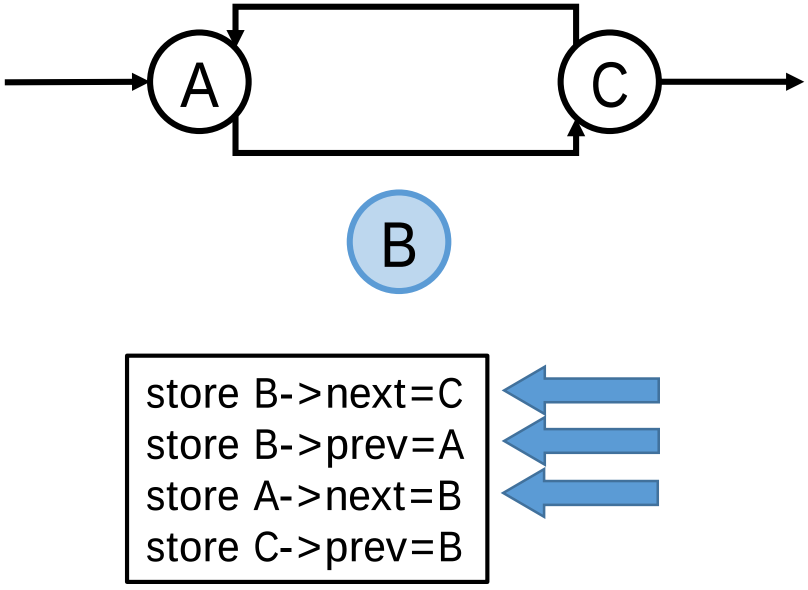
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



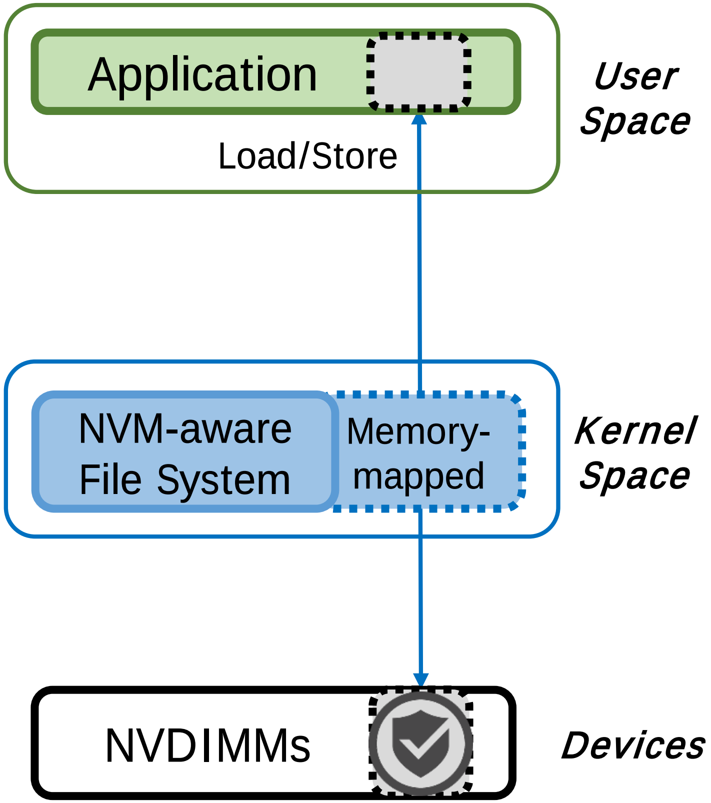
# Storage-Class Memory

- Directly attached to the app's virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



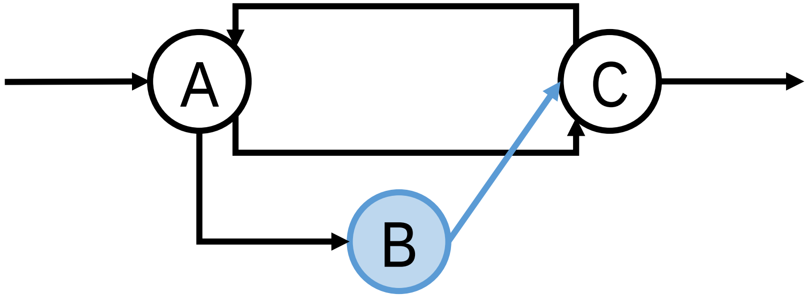
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



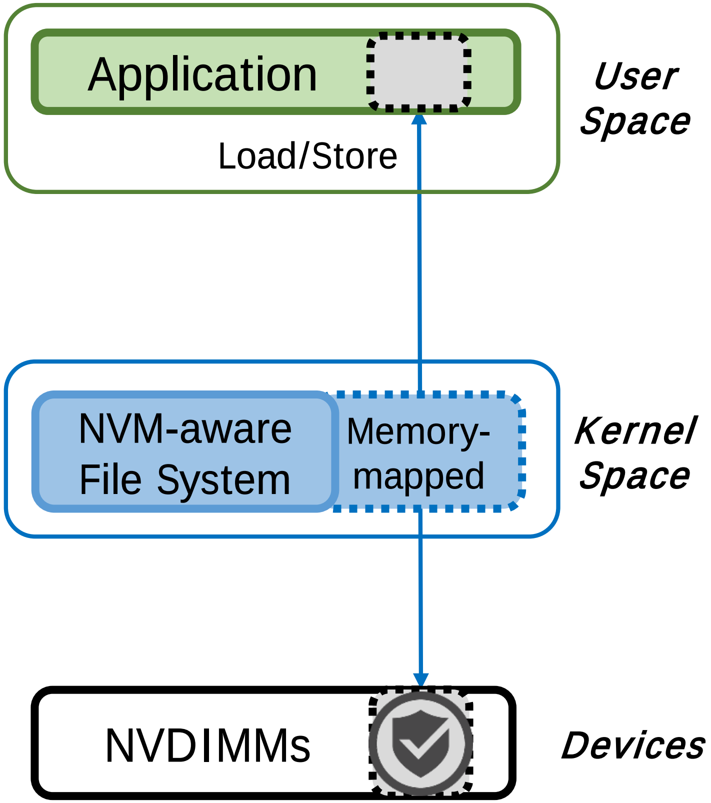
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



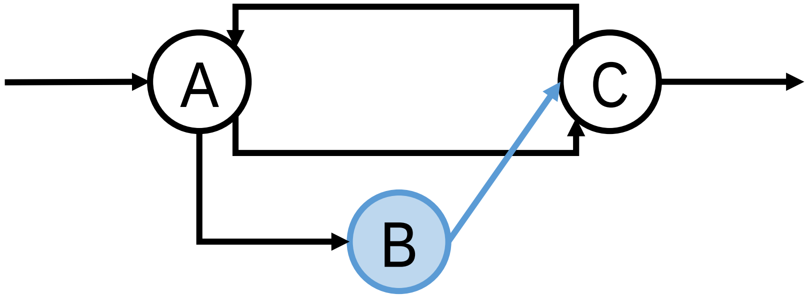
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B



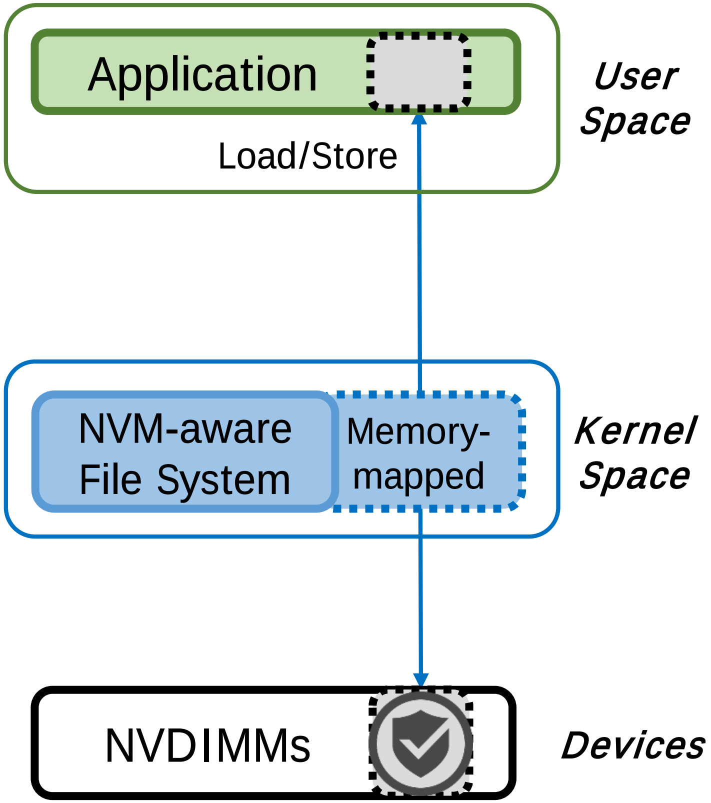
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



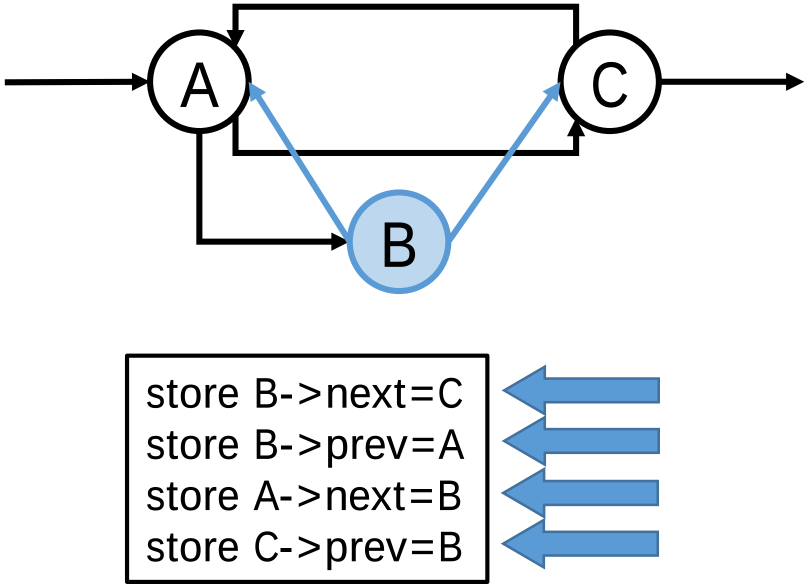
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B



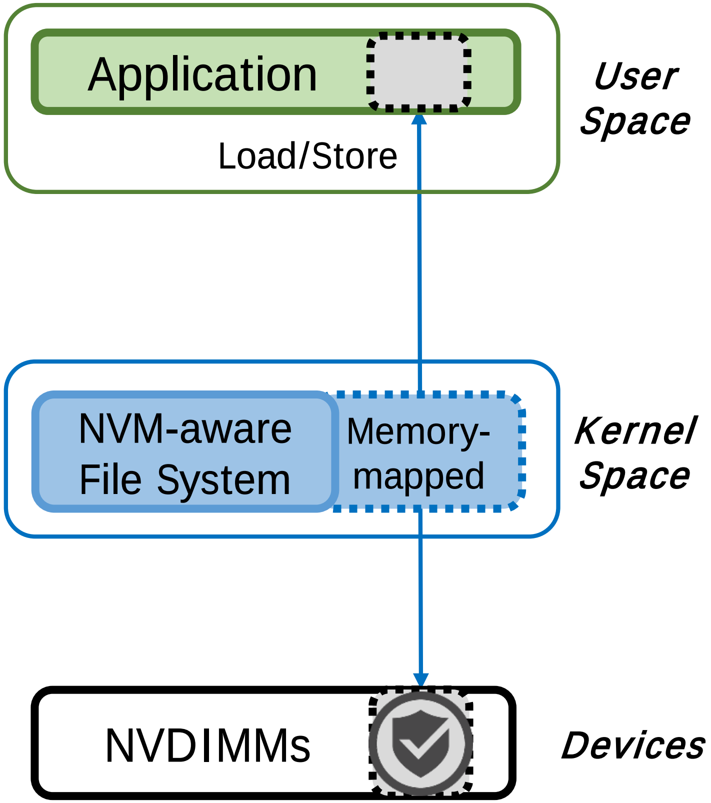
# Storage-Class Memory

- Directly attached to the app's virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



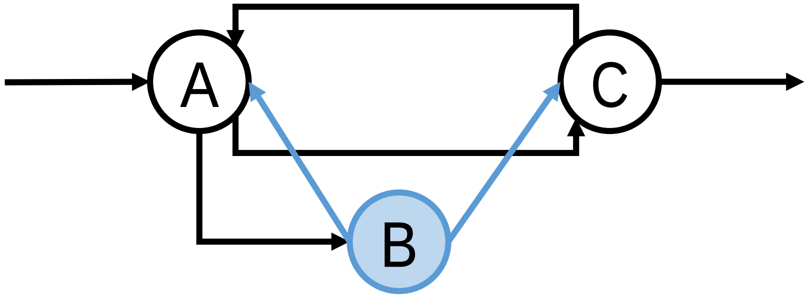
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



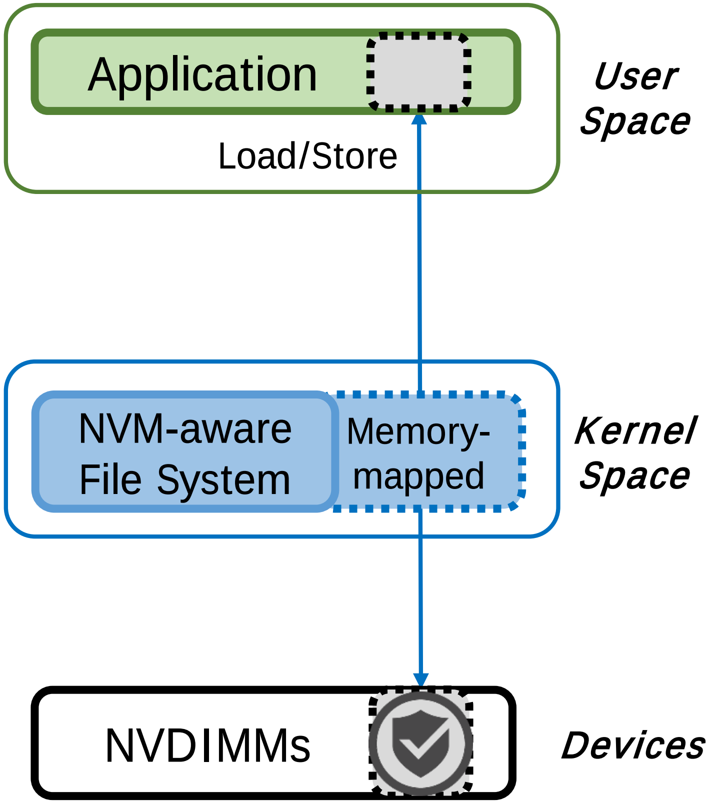
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



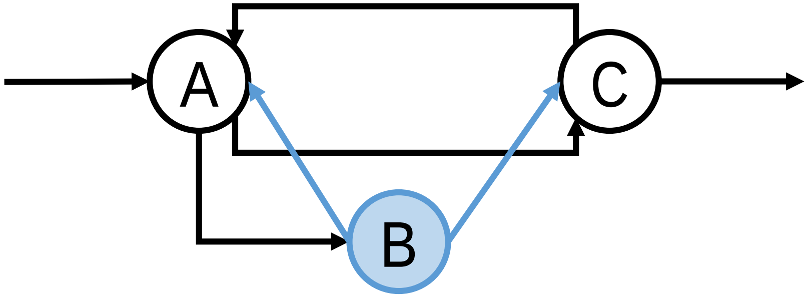
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



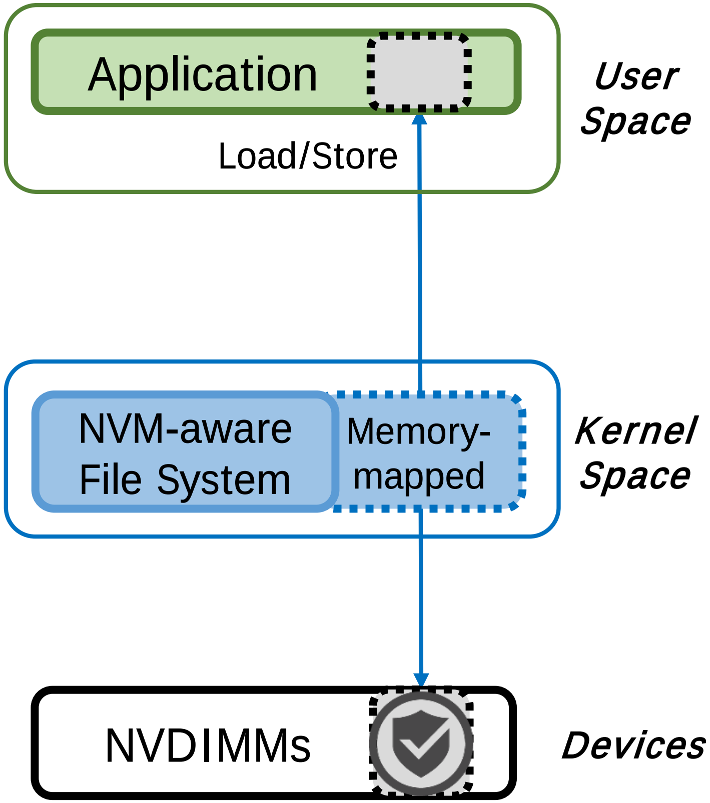
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



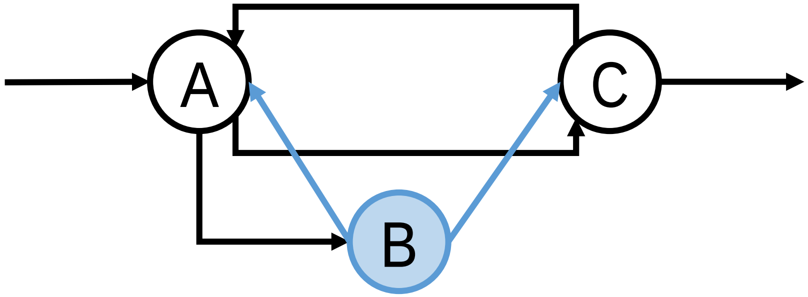
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



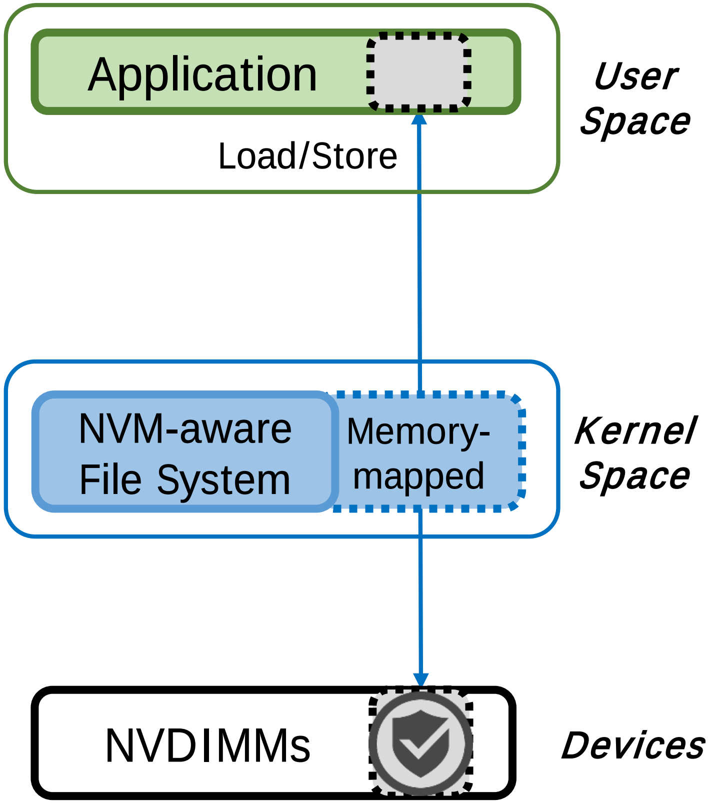
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



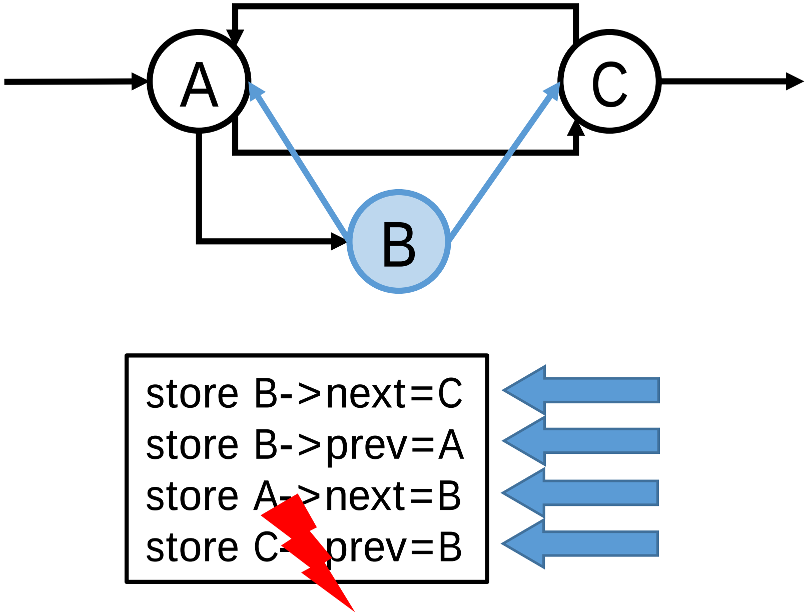
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



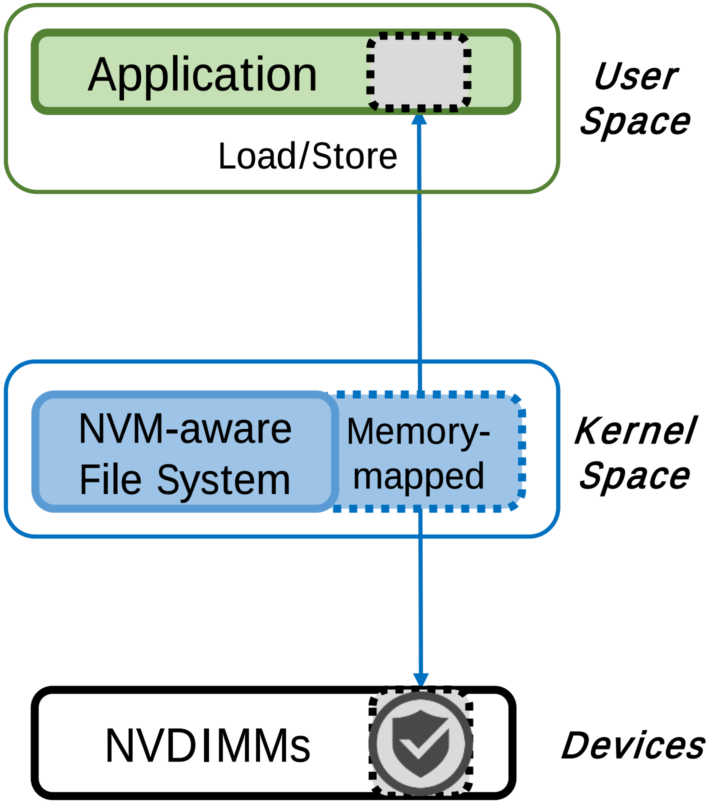
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



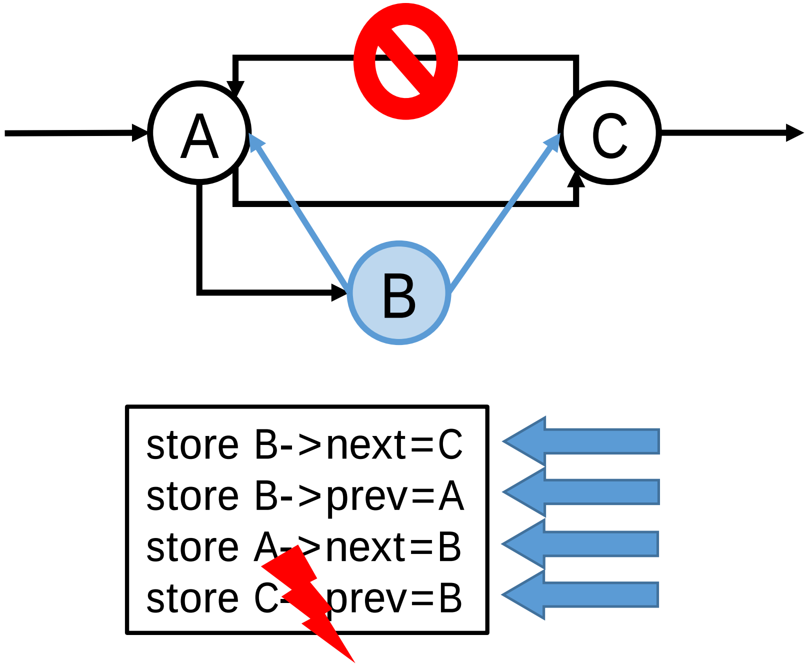
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```



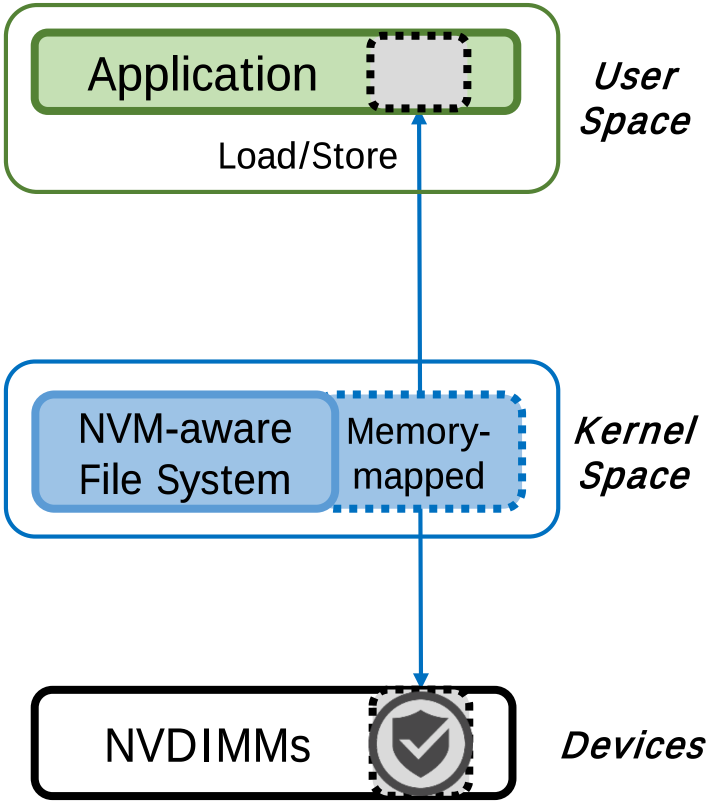
# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion



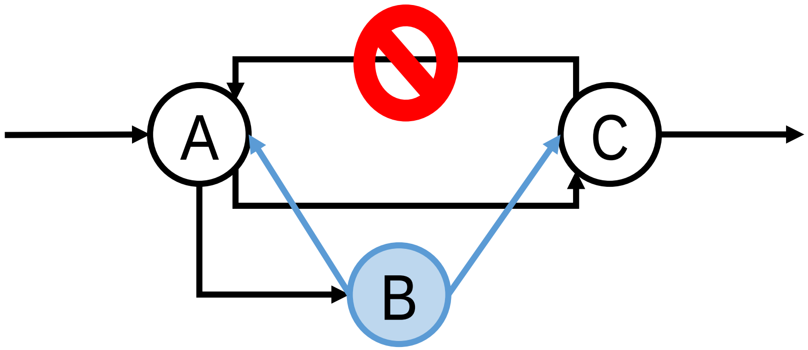
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

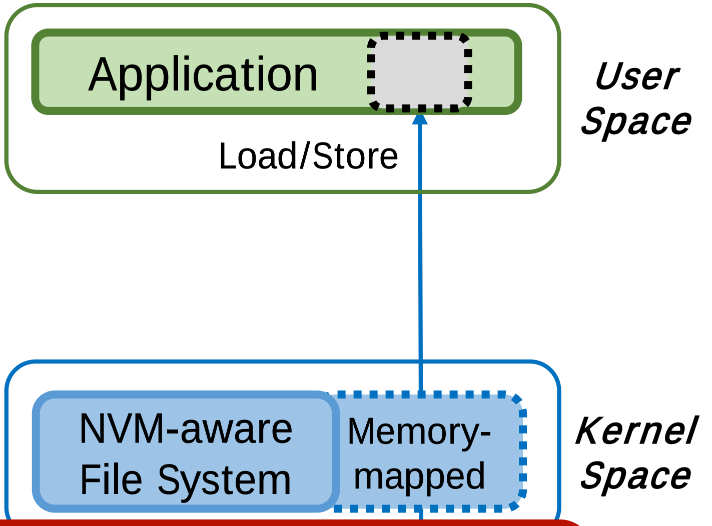


# Storage-Class Memory

- Directly attached to the app’s virtual address space
- Accessible through load/store instructions
- In-memory data persistency

- Ex) Doubly linked-list insertion





New requirement:
Supporting **Crash-Consistency** of NVM stores

store C->prev=B

NVDIMMs
Devices

# Atomic Durability through Logging

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software

Durability with **cache-flush**

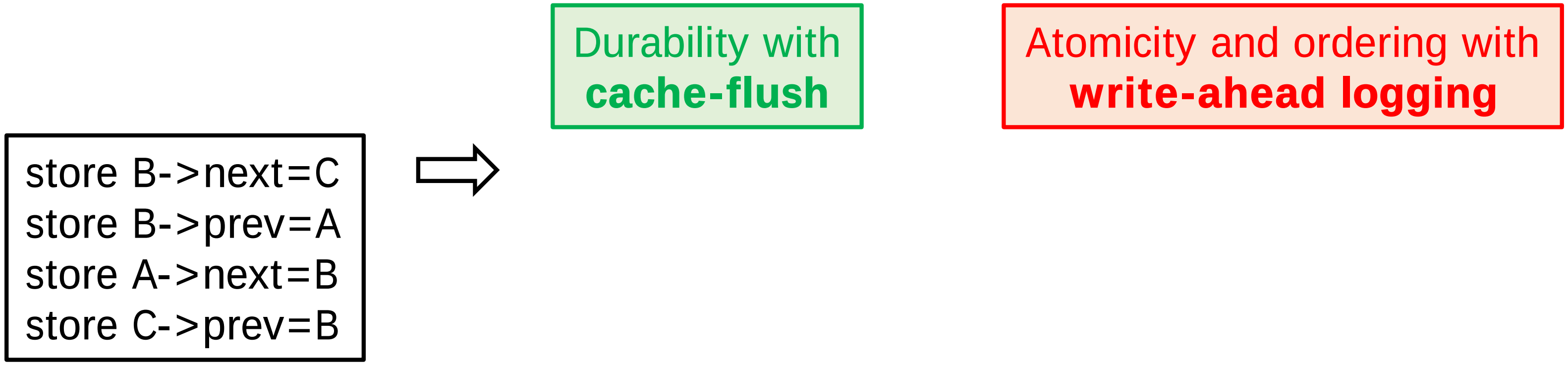
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

⇨

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

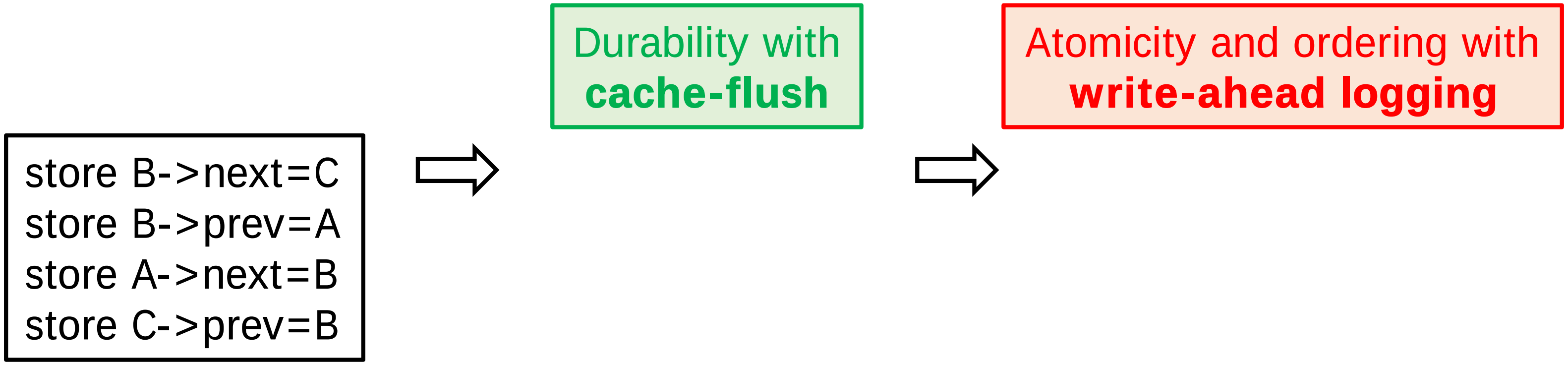
Durability with **cache-flush**

Atomicity and ordering with **write-ahead logging**

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

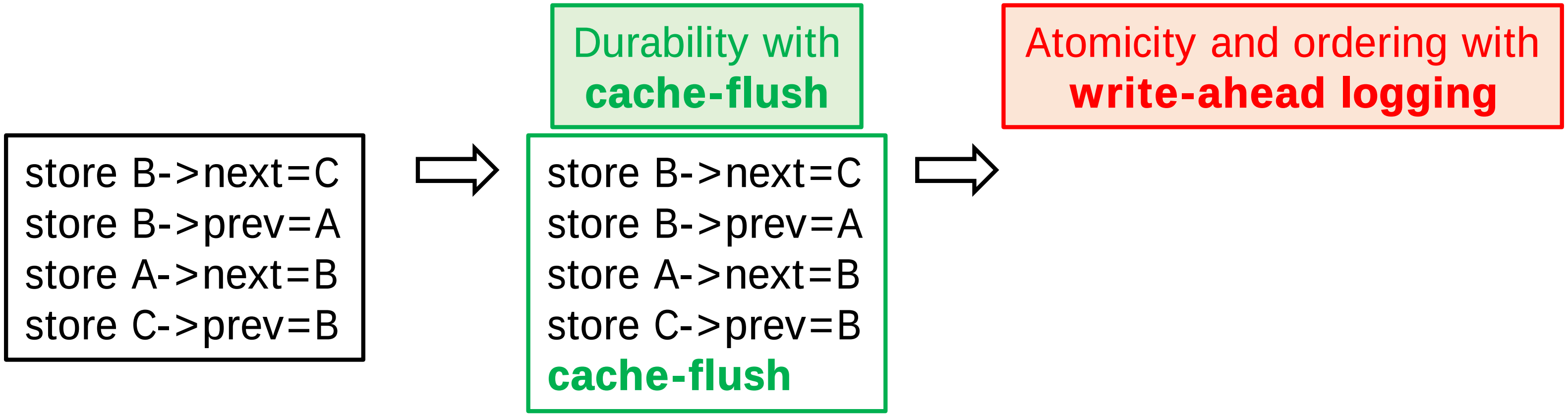
Durability with **cache-flush**

Atomicity and ordering with **write-ahead logging**

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

Durability with **cache-flush**

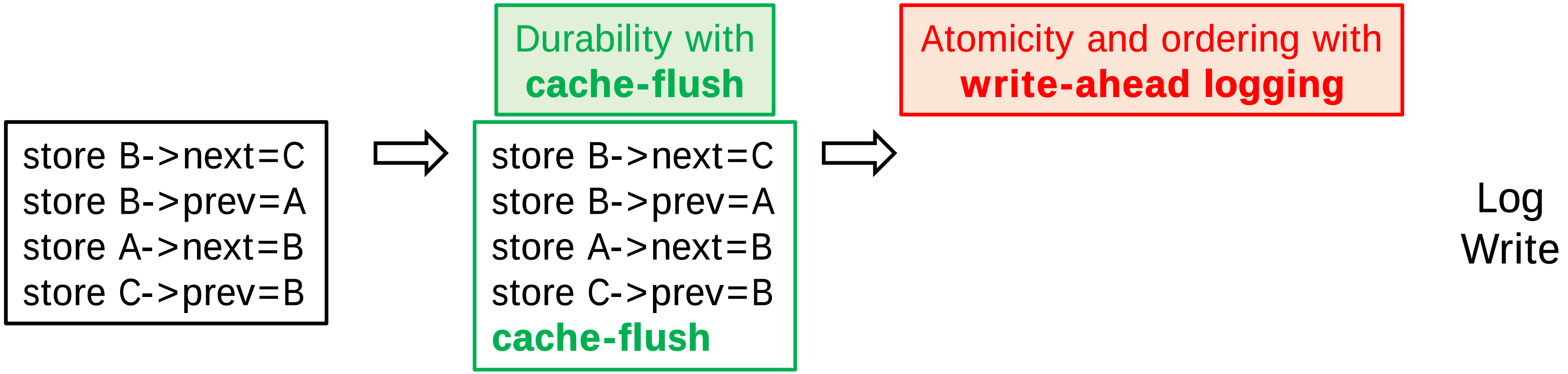
```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

Atomicity and ordering with **write-ahead logging**

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



Durability with **cache-flush**

Atomicity and ordering with **write-ahead logging**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

⇨

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```
**cache-flush**

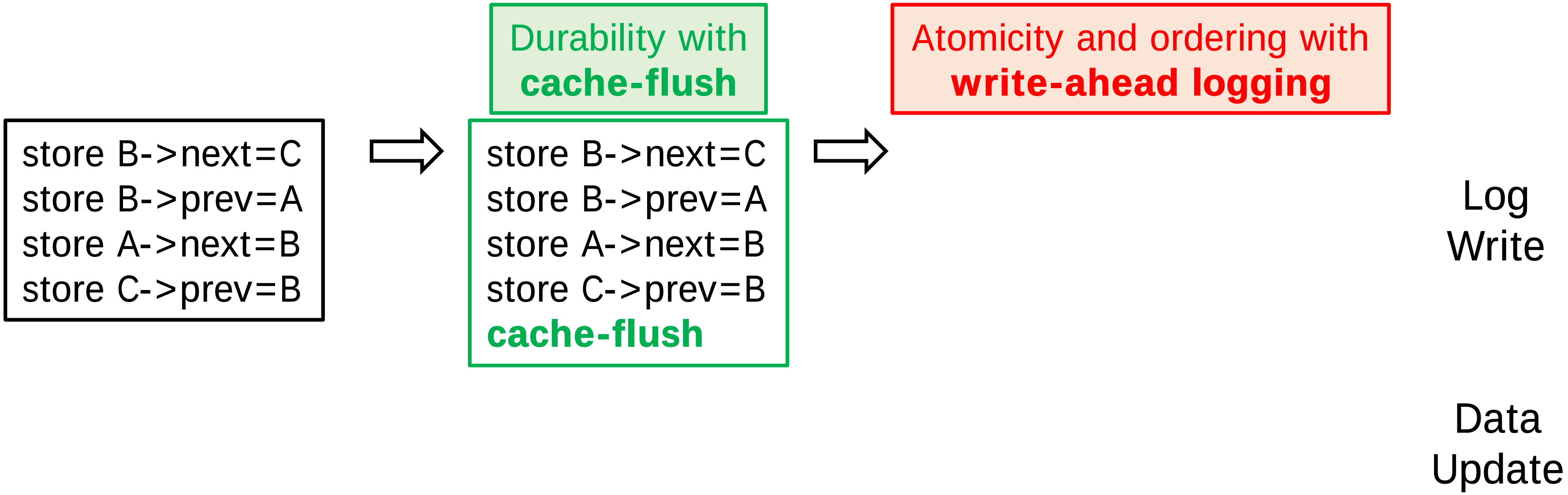
⇨

Log Write

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

Durability with **cache-flush**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

Atomicity and ordering with **write-ahead logging**

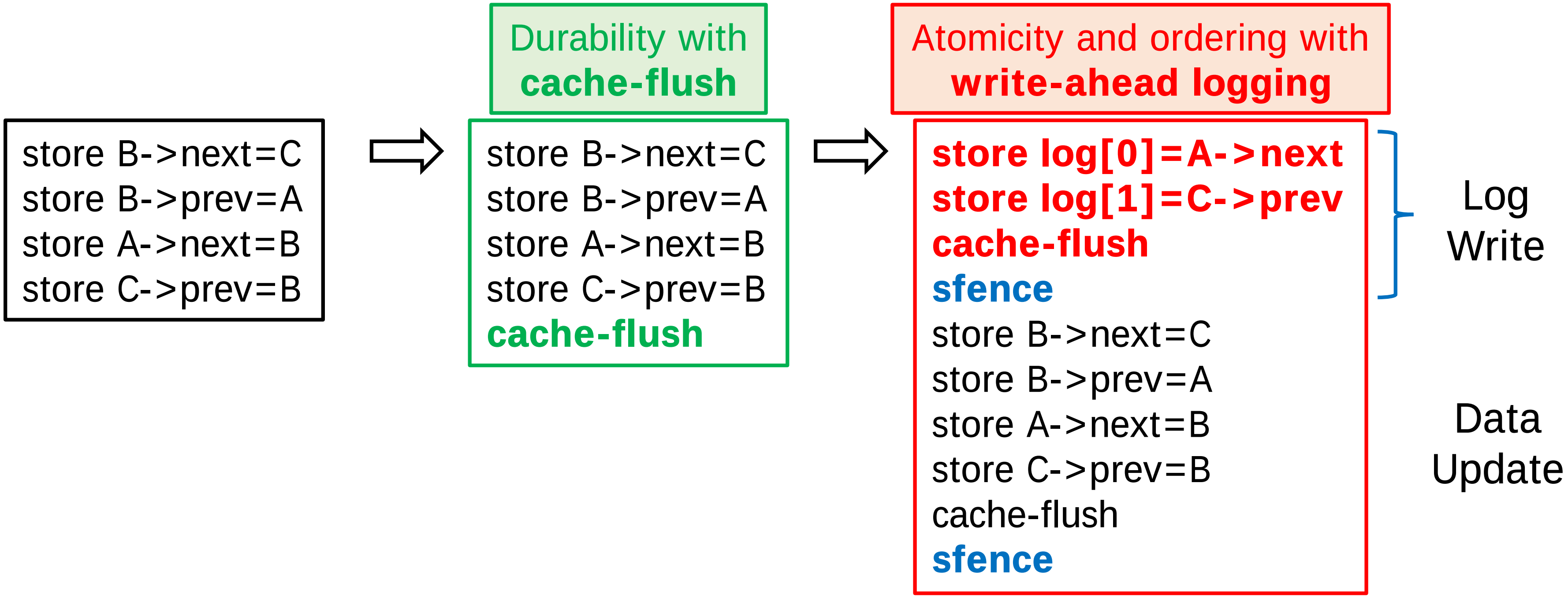
Log Write

Data Update

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

Durability with **cache-flush**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

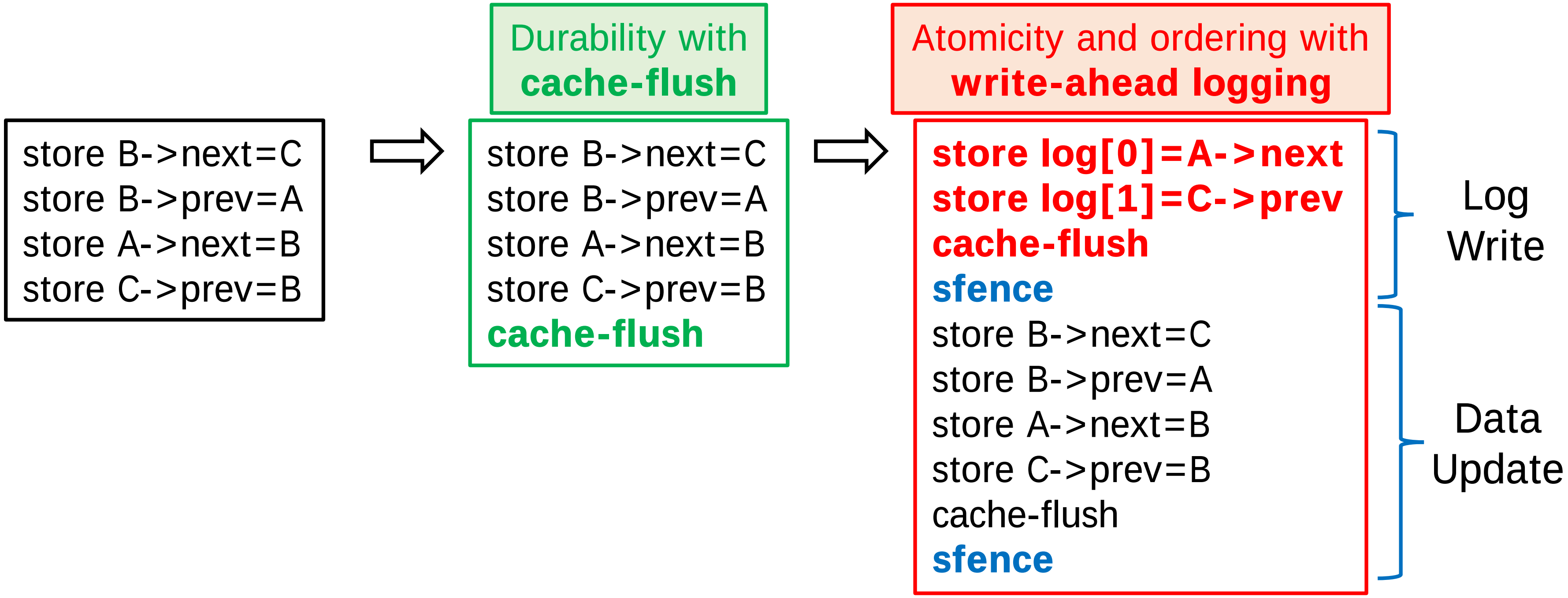
Atomicity and ordering with **write-ahead logging**

```
store log[0]=A->next
store log[1]=C->prev
cache-flush
sfence
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
sfence
```

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

Durability with **cache-flush**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

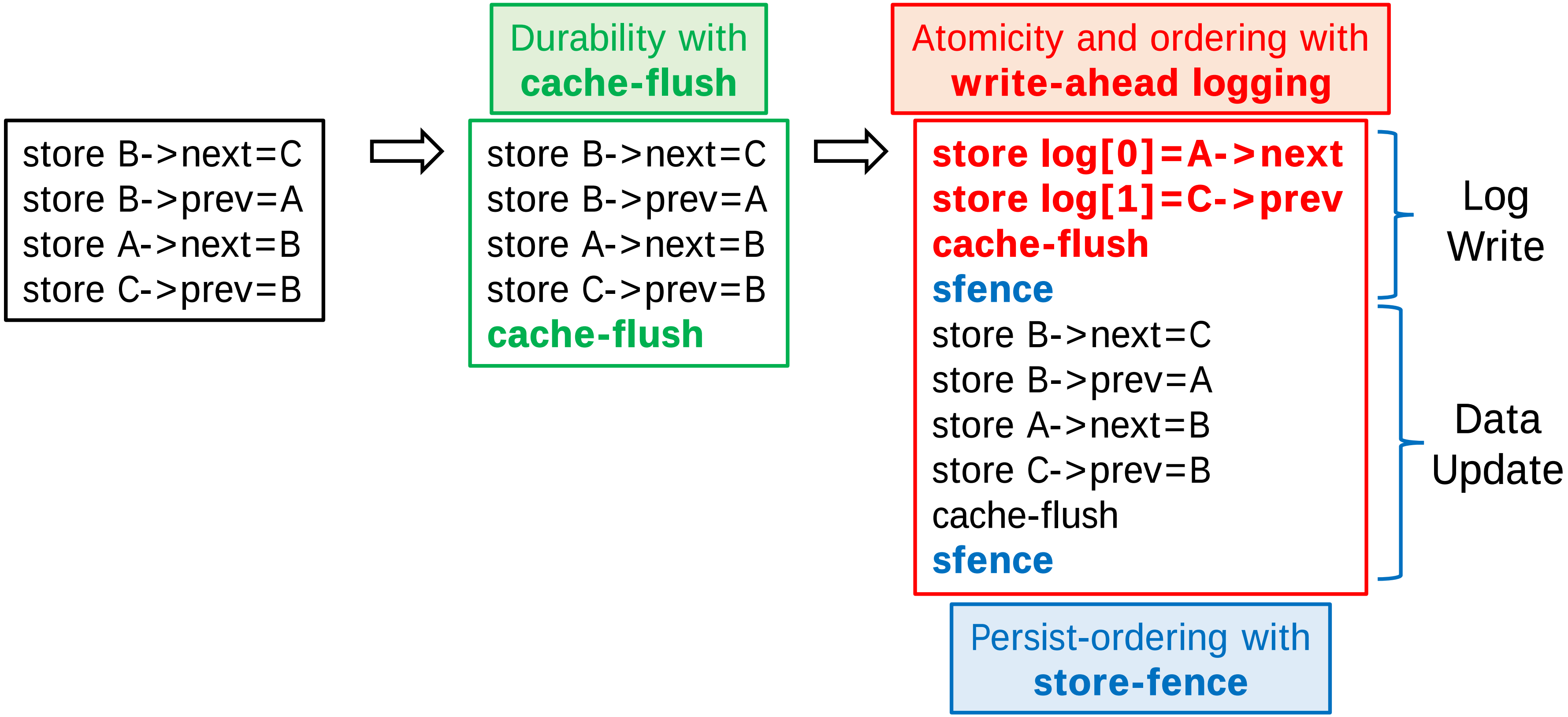
Atomicity and ordering with **write-ahead logging**

```
store log[0]=A->next
store log[1]=C->prev
cache-flush
sfence
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
sfence
```

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

Durability with **cache-flush**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

Atomicity and ordering with **write-ahead logging**

```
store log[0]=A->next
store log[1]=C->prev
cache-flush
sfence
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
sfence
```

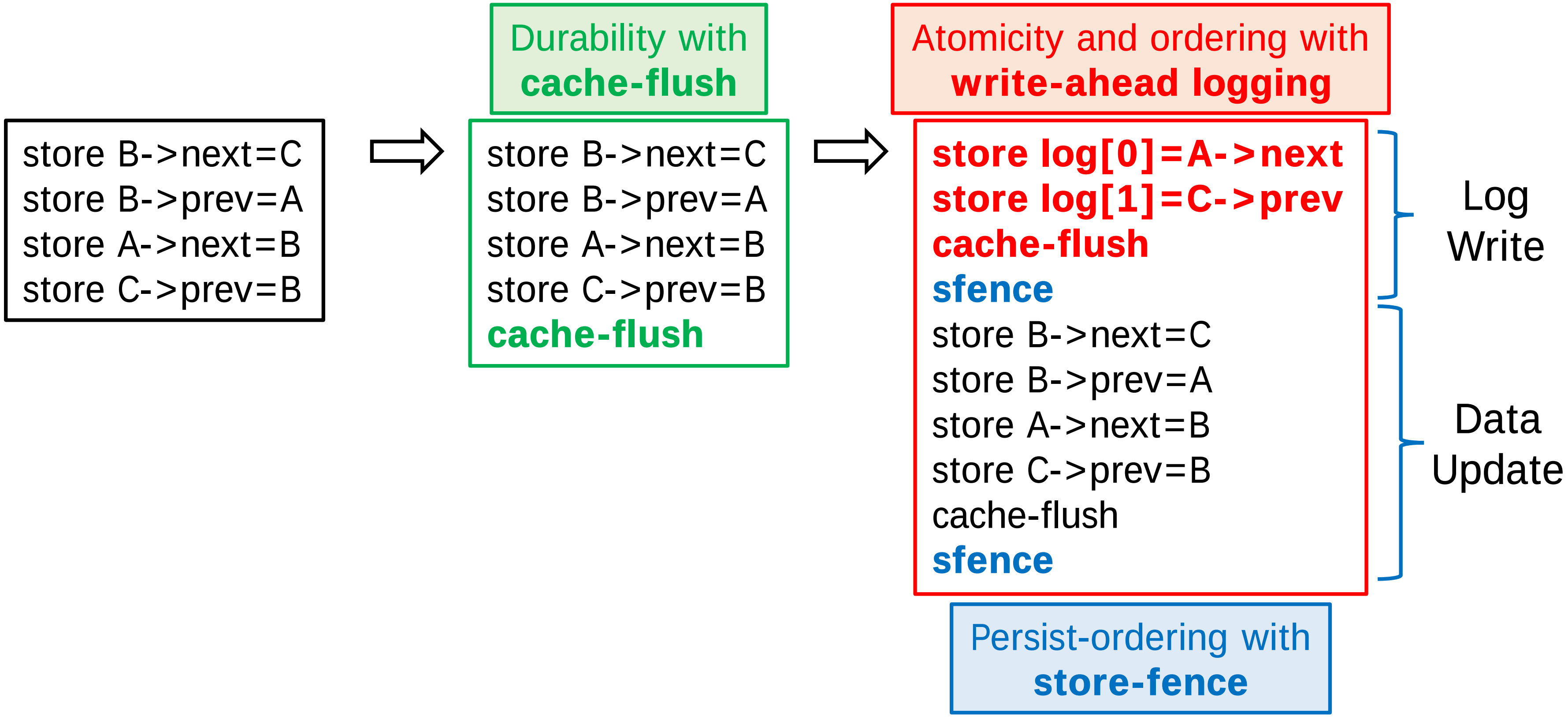
Log Write

Data Update

Persist-ordering with **store-fence**

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing
- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

Durability with **cache-flush**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

Atomicity and ordering with **write-ahead logging**

```
store log[0]=A->next
store log[1]=C->prev
cache-flush
sfence
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
sfence
```

Log Write

Data Update

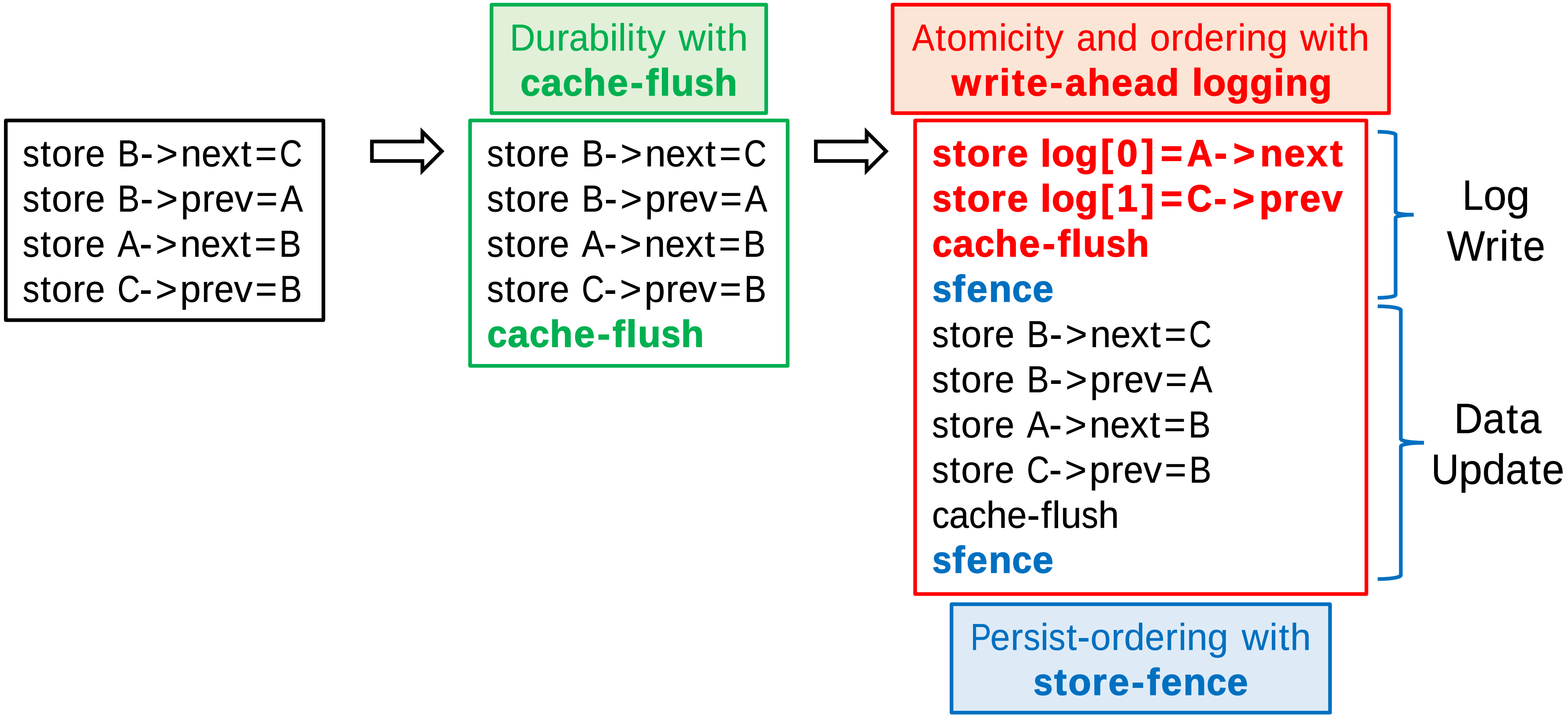
Persist-ordering with **store-fence**

2018-10-23

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

⇨

Durability with **cache-flush**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

⇨

Atomicity and ordering with **write-ahead logging**

```
store log[0]=A->next
store log[1]=C->prev
cache-flush
sfence
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
sfence
```

Log Write

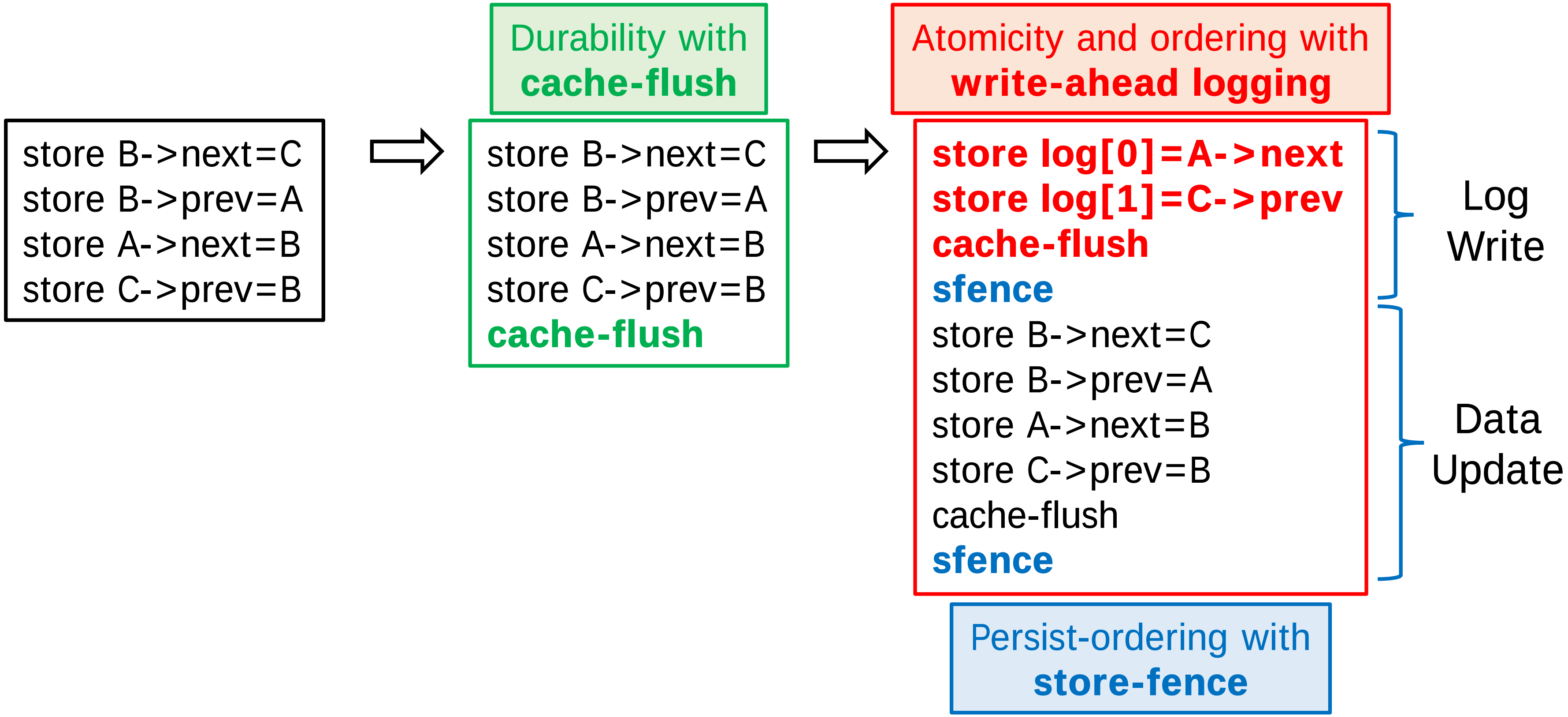
Data Update

Persist-ordering with **store-fence**

# Atomic Durability through Logging

- Transaction
  - All stores in a transaction become durable all together or nothing

- Ex) Atomic durability in software



```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
```

⇨

Durability with **cache-flush**

```
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
```

⇨

Atomicity and ordering with **write-ahead logging**

```
store log[0]=A->next
store log[1]=C->prev
cache-flush
sfence
store B->next=C
store B->prev=A
store A->next=B
store C->prev=B
cache-flush
sfence
```

Persist-ordering with **store-fence**

# HW-assisted Logging

# HW-assisted Logging

- Simple programming model

- HW is responsible for 1) log-write and 2) data update

- Advantages over software-logging
    - Fine-grained ordering & less CPU cycles

# HW-assisted Logging

- Simple programming model

```
Transaction_begin()
store B->next=C
...
store C->prev=B
Transaction_end()
```

- HW is responsible for 1) log-write and 2) data update

- Advantages over software-logging
  - Fine-grained ordering & less CPU cycles

# HW-assisted Logging

- Simple programming model

```
store B->next=C
...
store C->prev=B
```

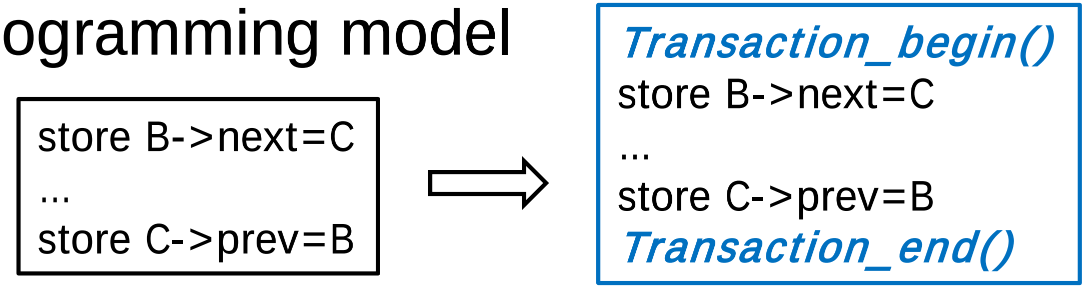
```
Transaction_begin()
store B->next=C
...
store C->prev=B
Transaction_end()
```

- HW is responsible for 1) log-write and 2) data update

- Advantages over software-logging
  - Fine-grained ordering & less CPU cycles

# HW-assisted Logging

- Simple programming model



```
store B->next=C
...
store C->prev=B
```

```
Transaction_begin()
store B->next=C
...
store C->prev=B
Transaction_end()
```

- HW is responsible for 1) <u>log-write</u> and 2) <u>data update</u>

- Advantages over software-logging
  - Fine-grained ordering & less CPU cycles

# HW-assisted Logging

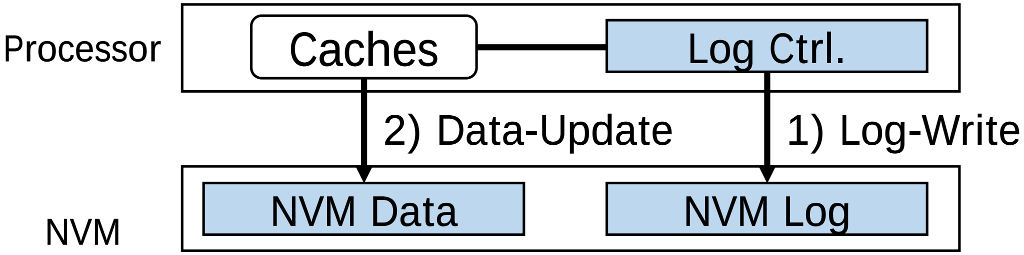
- Simple programming model

```
store B->next=C
…
store C->prev=B
```

⇒

```
Transaction_begin()
store B->next=C
…
store C->prev=B
Transaction_end()
```

- HW is responsible for 1) log-write and 2) data update



- Advantages over software-logging
  - Fine-grained ordering & less CPU cycles

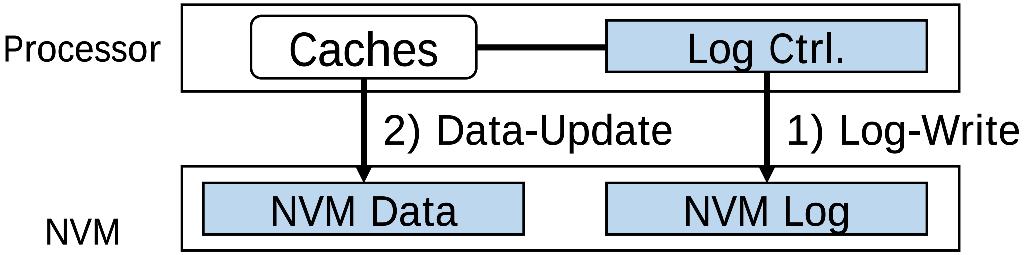
# HW-assisted Logging

- Simple programming model

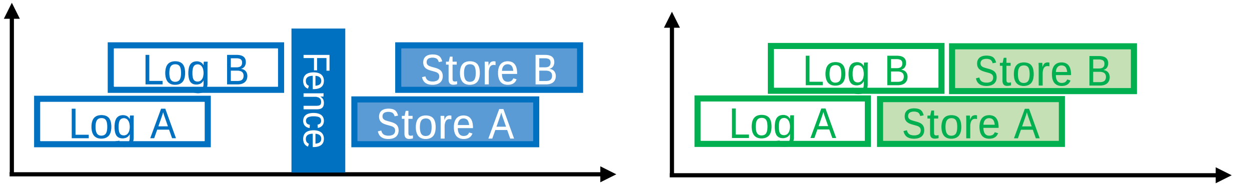
```
store B->next=C
...
store C->prev=B
```

```
Transaction_begin()
store B->next=C
...
store C->prev=B
Transaction_end()
```

- HW is responsible for 1) log-write and 2) data update



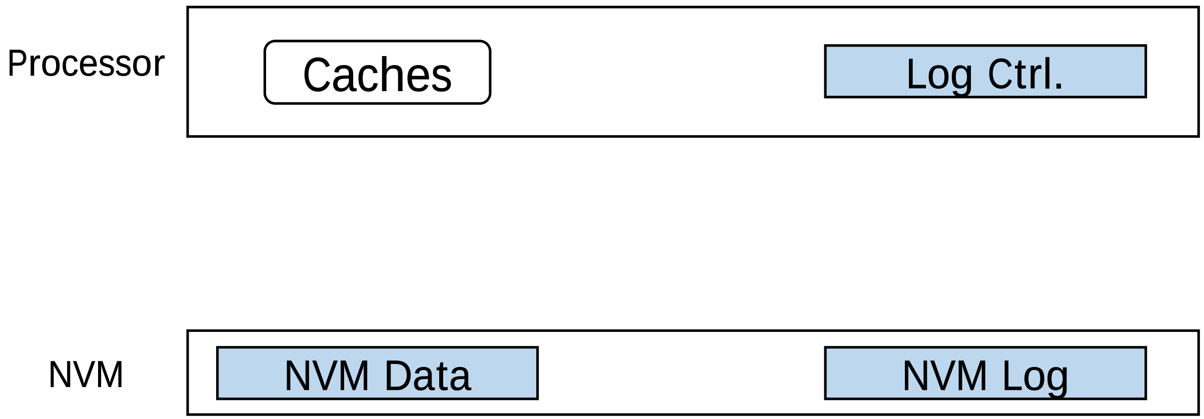
- Advantages over software-logging
  - Fine-grained ordering & less CPU cycles



(a) Ordering with SW

(b) Ordering with HW

# Past Proposal: Undo-based HW-Logging

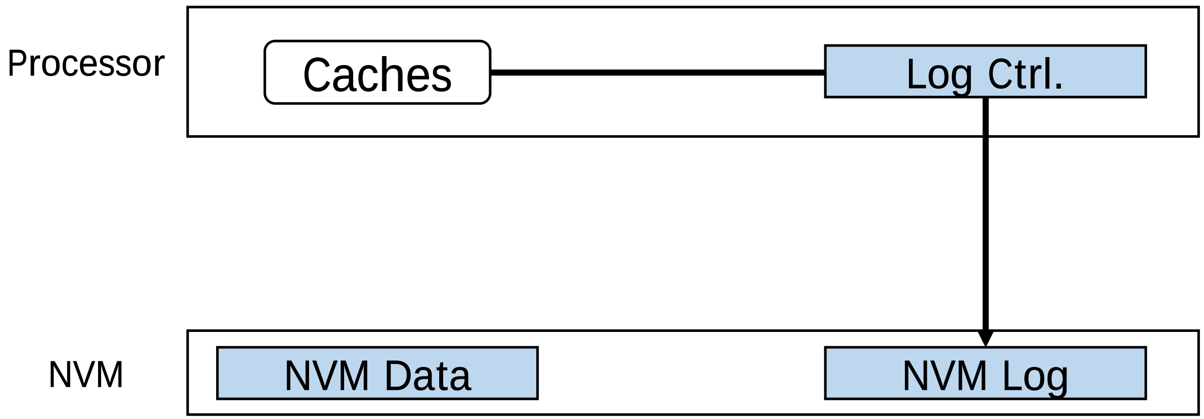


- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs
- Update data in NVM **before commit**

➔ **Synchronous data-update**

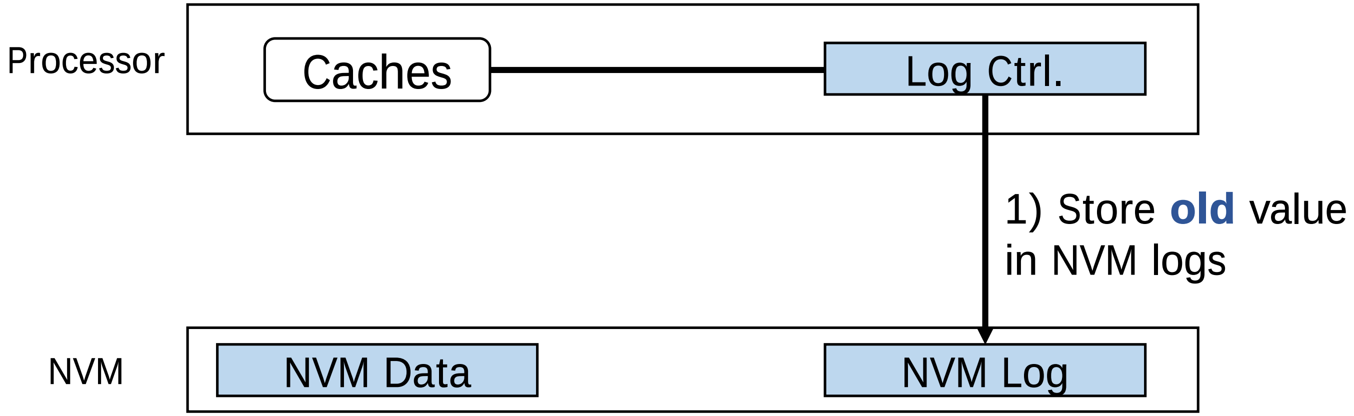


- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs
- Update data in NVM **before commit**

➔ **Synchronous data-update**

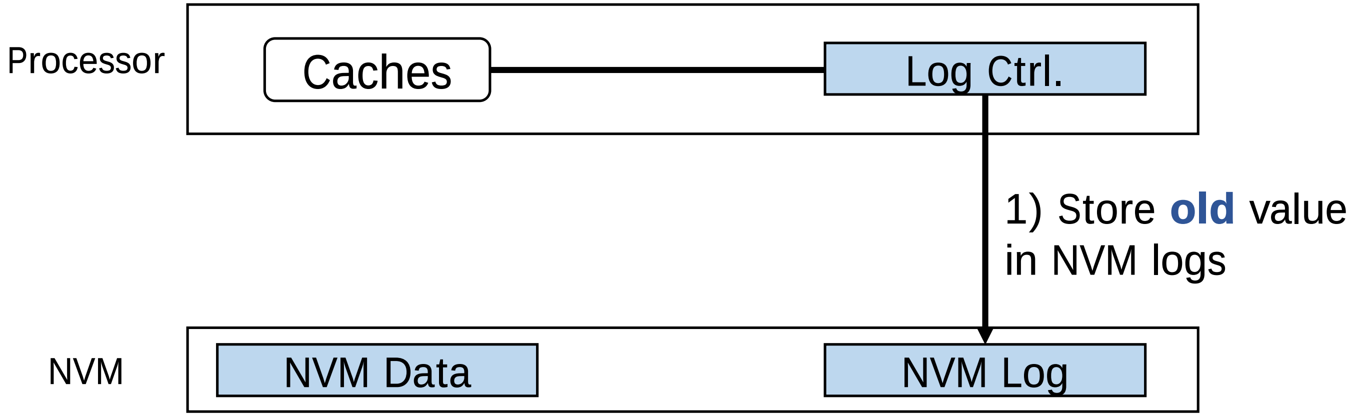


- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs
- Update data in NVM **before commit**

➔ **Synchronous data-update**

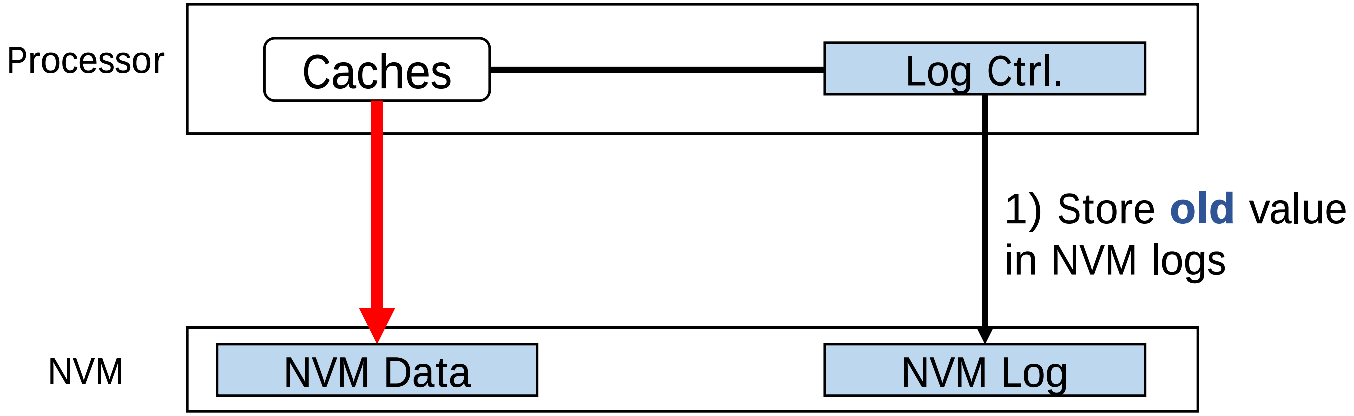


- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs Addr
- Update data in NVM **before commit**

➔ **Synchronous data-update**



- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

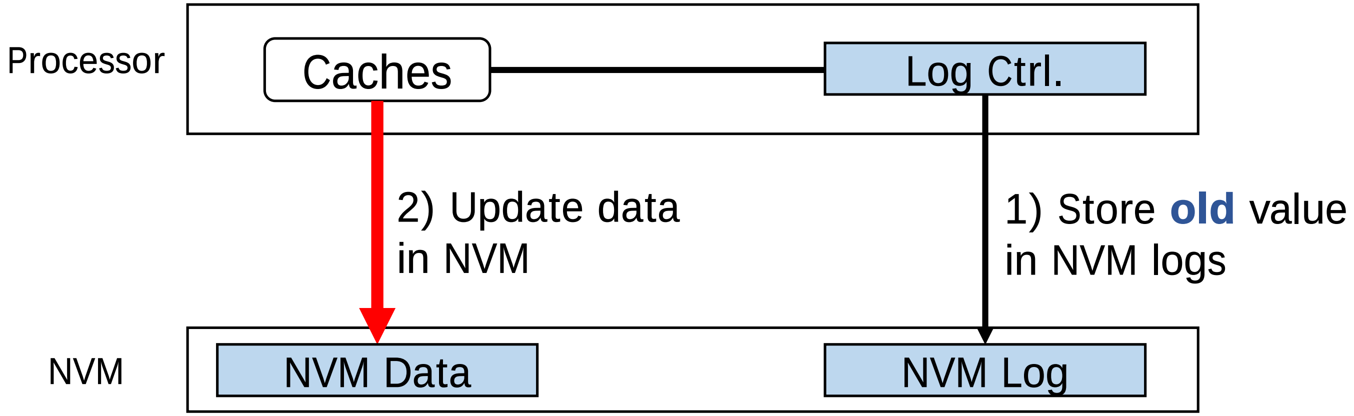
# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs

| Addr | **Old** Value |
|---|---|

- Update data in NVM **before commit**

➔ **Synchronous data-update**



- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

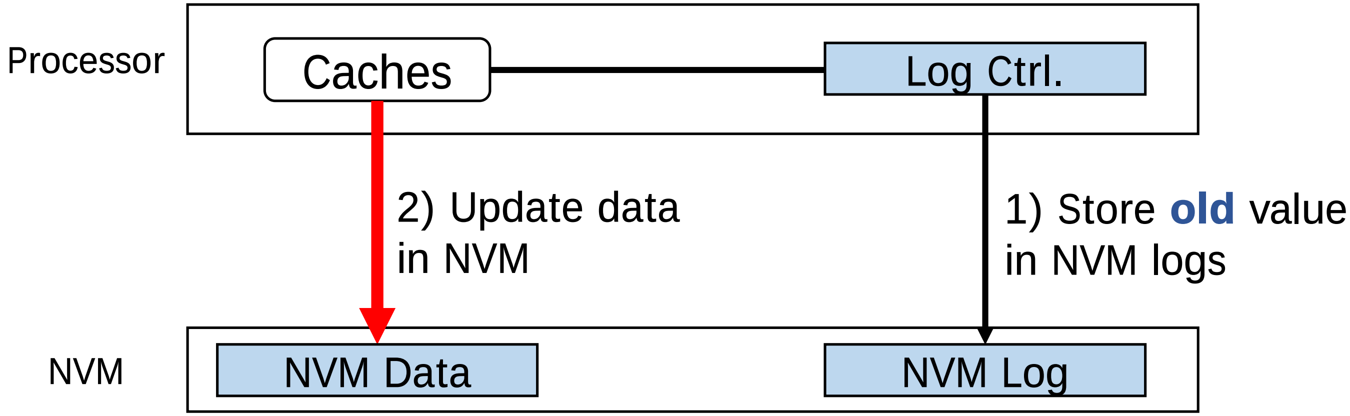
# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs

| | |
|---|---|
| Addr | **Old** Value |

- Update data in NVM **before commit**

➔ **Synchronous data-update**



**Long critical path** due to synchronous data-update

- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

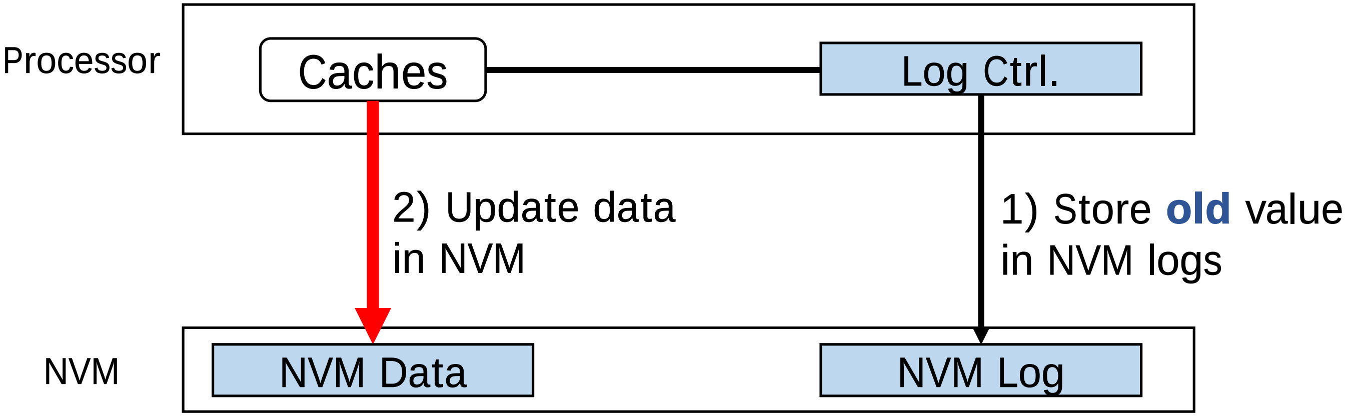
# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs

| | |
|---|---|
| Addr | **Old** Value |

- Update data in NVM **before commit**

➔ **Synchronous data-update**



**Long critical path** due to synchronous data-update

- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

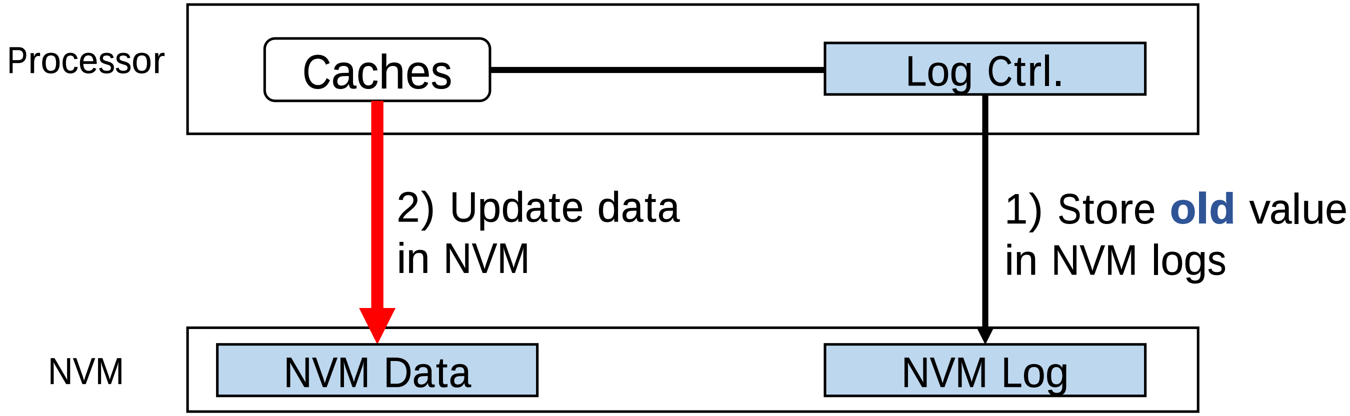
# Past Proposal: Undo-based HW-Logging

- Store **old** value in logs

| | |
|---|---|
| Addr | **Old** Value |

- Update data in NVM **before commit**

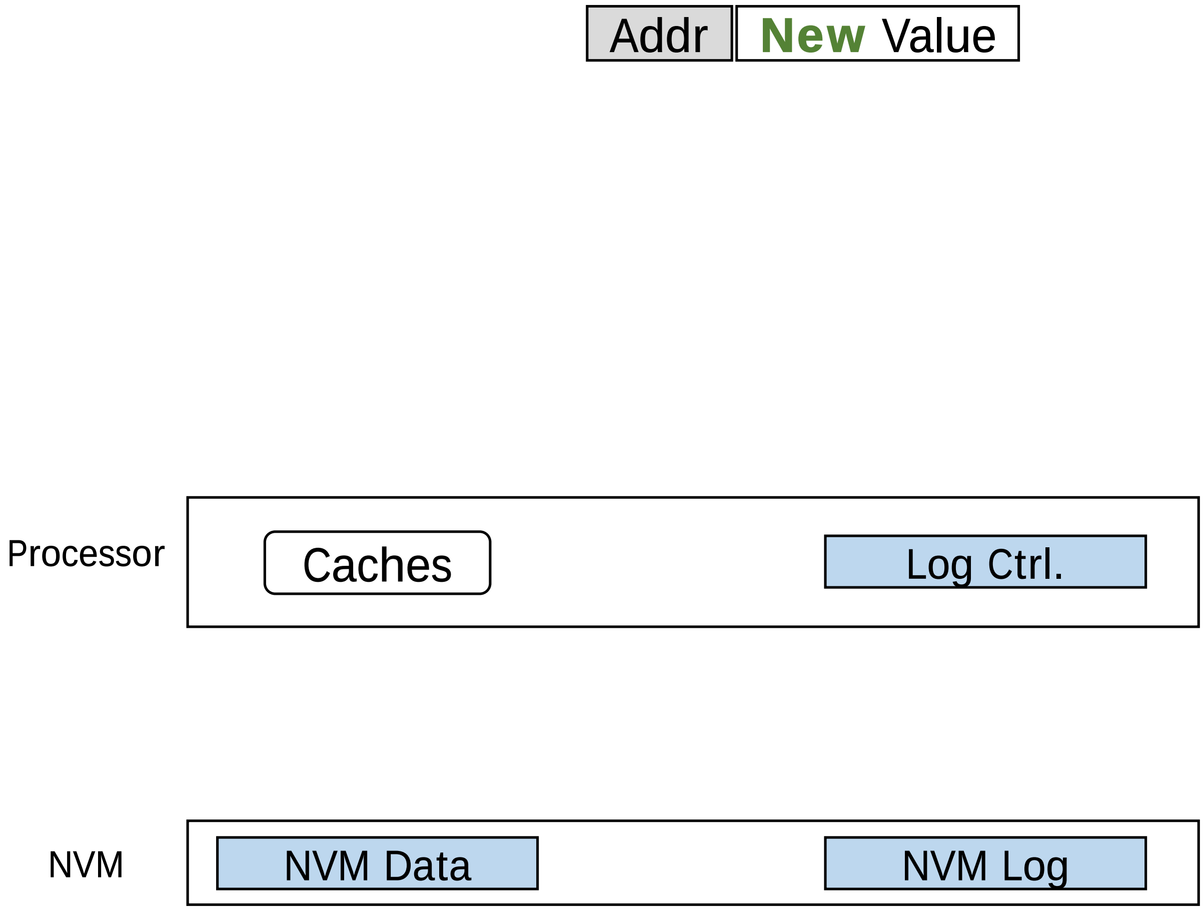
➔ **Synchronous data-update**



**Long critical path** due to synchronous data-update

- A. Joshi et al. HPCA 2017.
- S. Shin et al. ISCA 2017.

# Past Proposal: Redo-based HW-Logging



Addr **New** Value

Processor

Caches

Log Ctrl.

NVM

NVM Data

NVM Log

- K. Doshi et al. HPCA 2016.

# Past Proposal: Redo-based HW-Logging

- Store **new** value in logs

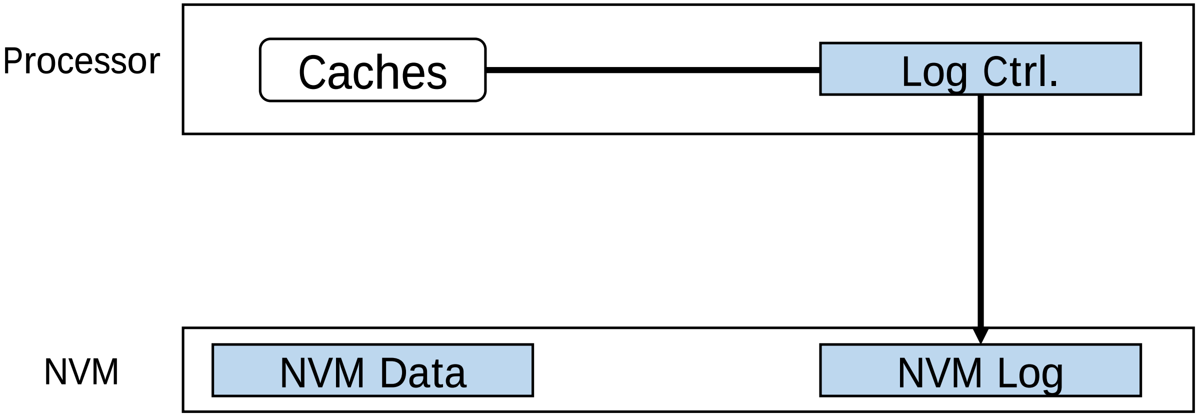
| Addr | **New** Value |
|---|---|

- Update data in NVM **after commit**

➔ Asynchronous data-update

- However, update by reading log entries from NVM

➔ **Indirect data-update**



- K. Doshi et al. HPCA 2016.

# Past Proposal: Redo-based HW-Logging

- Store **new** value in logs

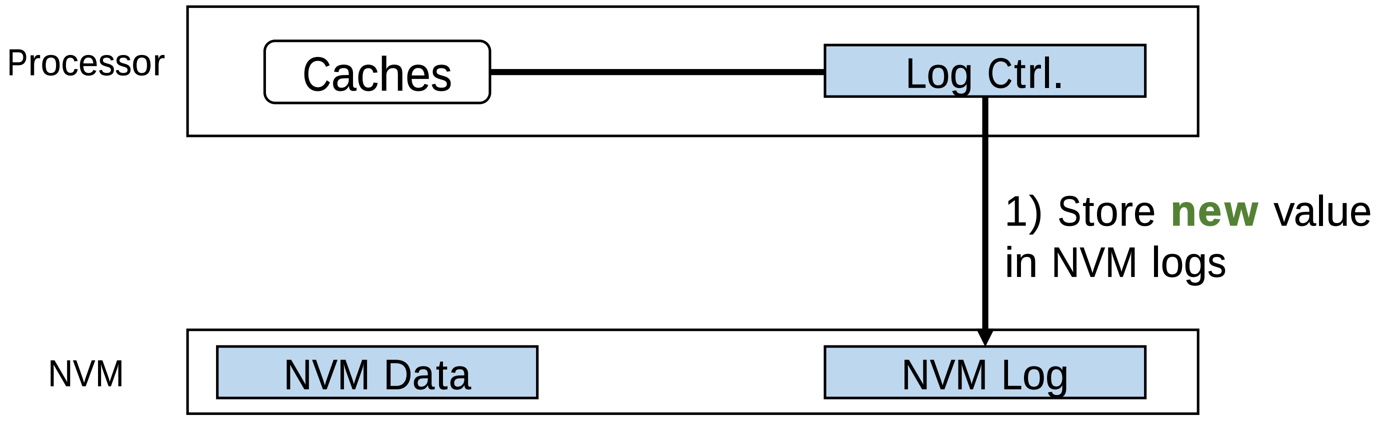
| Addr | **New** Value |
|---|---|

- Update data in NVM **after commit**

➔ Asynchronous data-update

- However, update by reading log entries from NVM

➔ **Indirect data-update**



1) Store **new** value in NVM logs

- K. Doshi et al. HPCA 2016.

# Past Proposal: Redo-based HW-Logging

- Store **new** value in logs

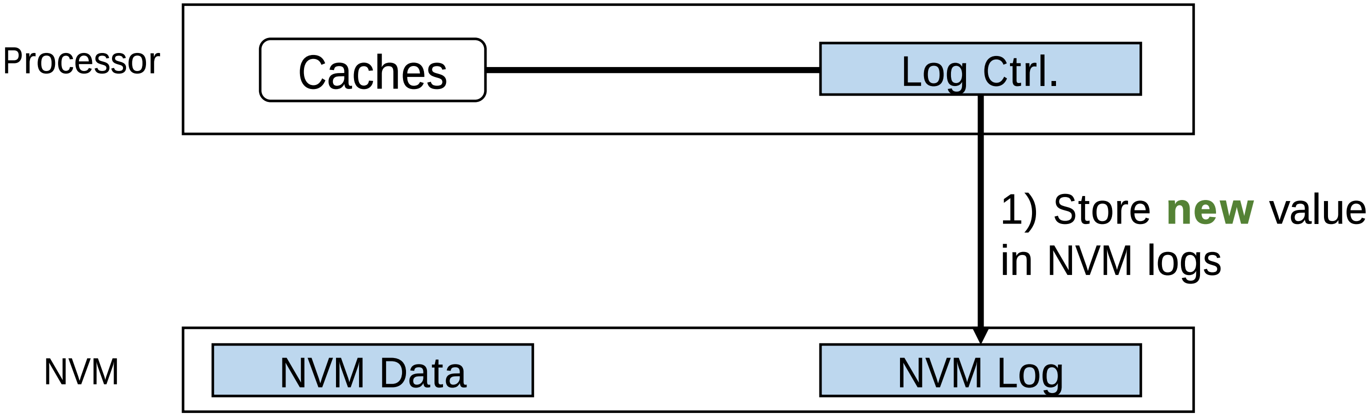
| Addr | **New** Value |
|---|---|

- Update data in NVM **after commit**

➔ Asynchronous data-update

- However, update by reading log entries from NVM

➔ **Indirect data-update**



- K. Doshi et al. HPCA 2016.

# Past Proposal: Redo-based HW-Logging

- Store **new** value in logs

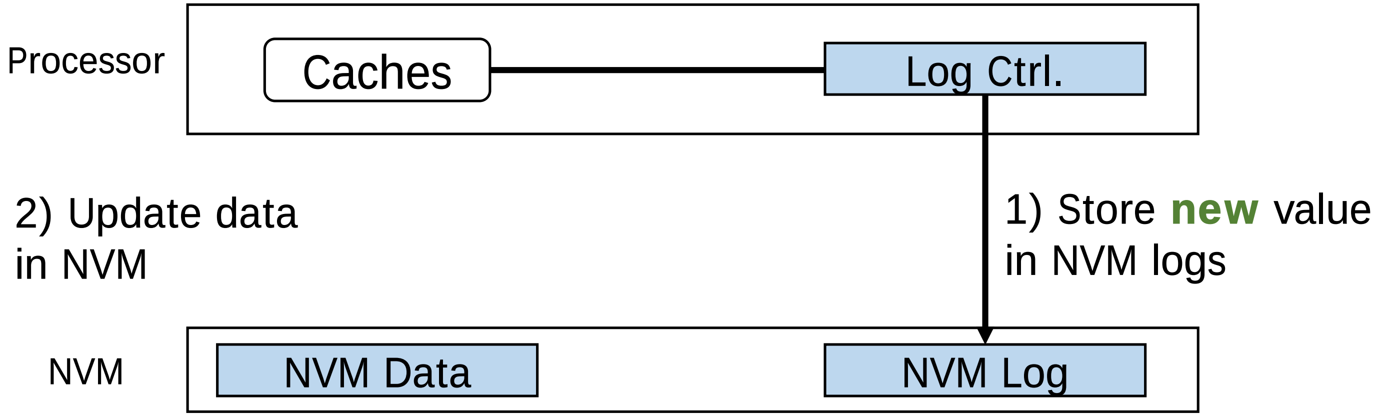
| | |
|---|---|
| Addr | **New** Value |

- Update data in NVM **after commit**

  ➔ Asynchronous data-update

- However, update by reading log entries from NVM

  ➔ **Indirect data-update**



- K. Doshi et al. HPCA 2016.

# Past Proposal: Redo-based HW-Logging

- Store **new** value in logs

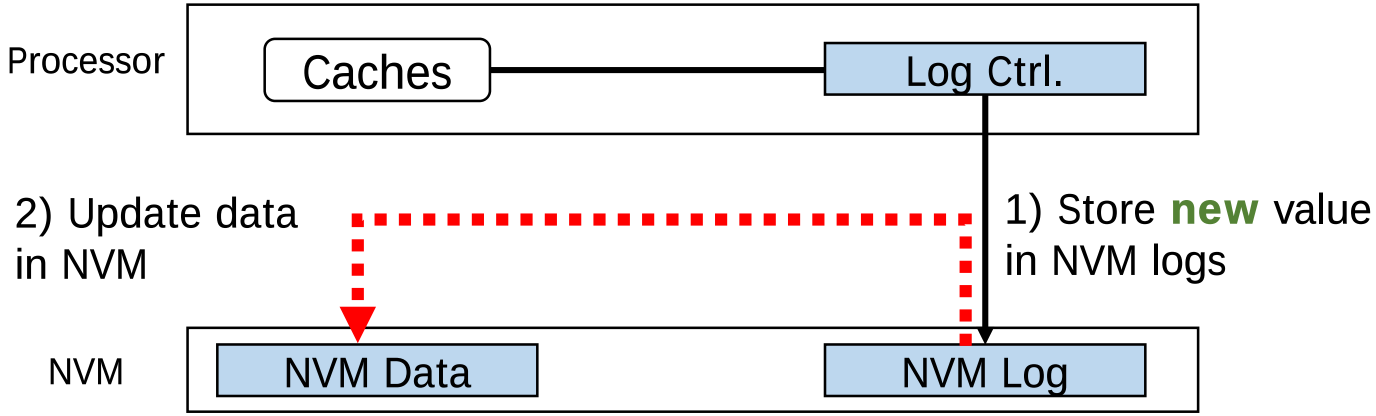
| | |
|---|---|
| Addr | **New** Value |

- Update data in NVM **after commit**

➔ Asynchronous data-update

- However, update by reading log entries from NVM

➔ **Indirect data-update**



- K. Doshi et al. HPCA 2016.

# Past Proposal: Redo-based HW-Logging

- Store **new** value in logs

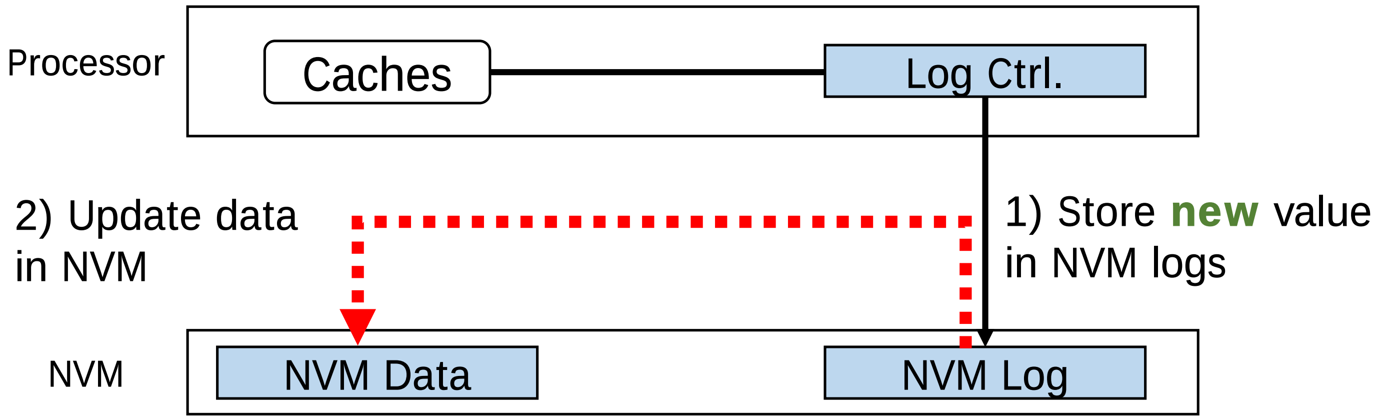
| Addr | **New** Value |
|---|---|

- Update data in NVM **after commit**

➔ Asynchronous data-update

- However, update by reading log entries from NVM

➔ **Indirect data-update**

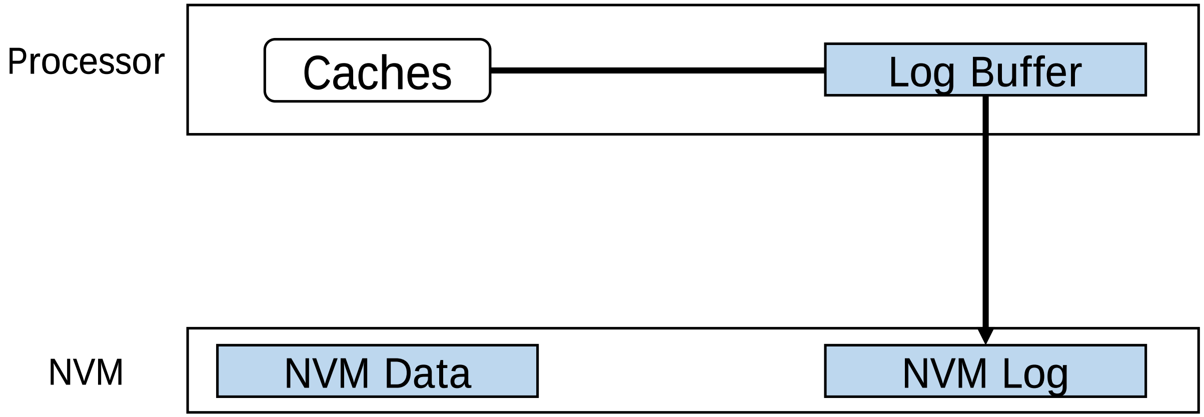


**Wastes extra NVM bandwidth** for reading logs from NVM

- K. Doshi et al. HPCA 2016.

# Past Proposal: Undo-Redo HW-Logging

| | | |
|---|---|---|
| Addr | **Old** Value | **New** Value |



- M. Ogleari et al. HPCA 2018.

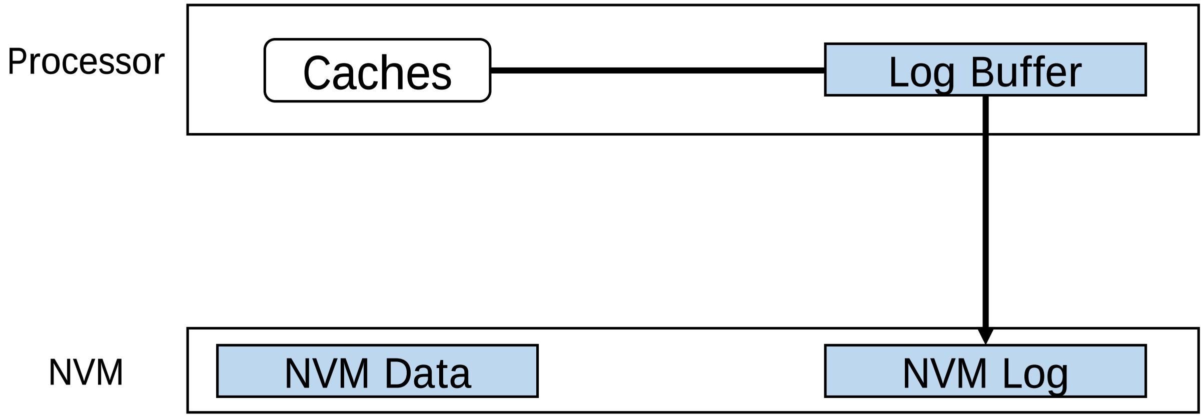
# Past Proposal: Undo-Redo HW-Logging

- Store **both** old and new value in logs

| | | |
|---|---|---|
| Addr | **Old** Value | **New** Value |

➔ **Larger log sizes**

- Update data in NVM **after commit**



- M. Ogleari et al. HPCA 2018.

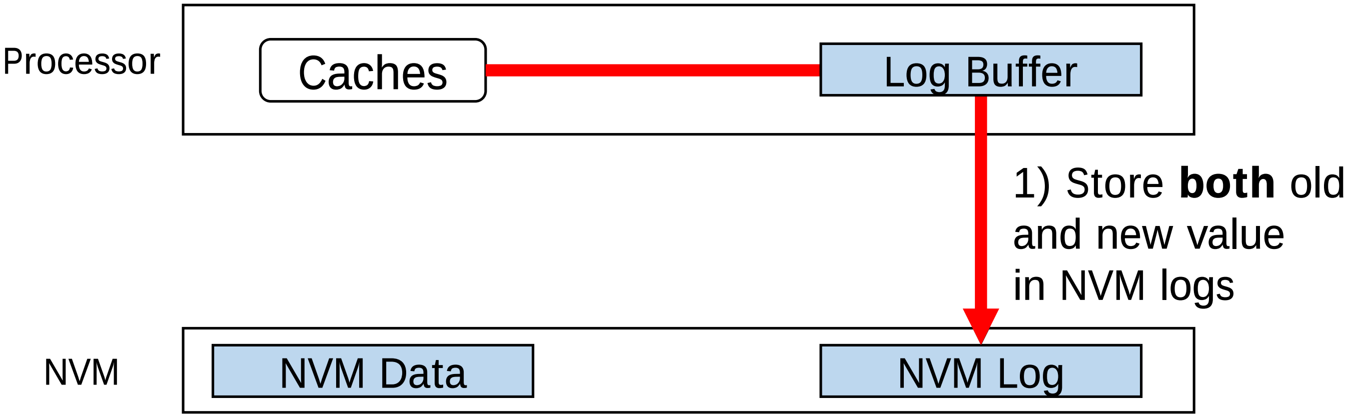
# Past Proposal: Undo-Redo HW-Logging

- Store **both** old and new value in logs

| | | |
|---|---|---|
| Addr | **Old** Value | **New** Value |

➔ **Larger log sizes**

- Update data in NVM **after commit**



1) Store **both** old and new value in NVM logs

Requires **more NVM writes** for storing logs in NVM

- M. Ogleari et al. HPCA 2018.

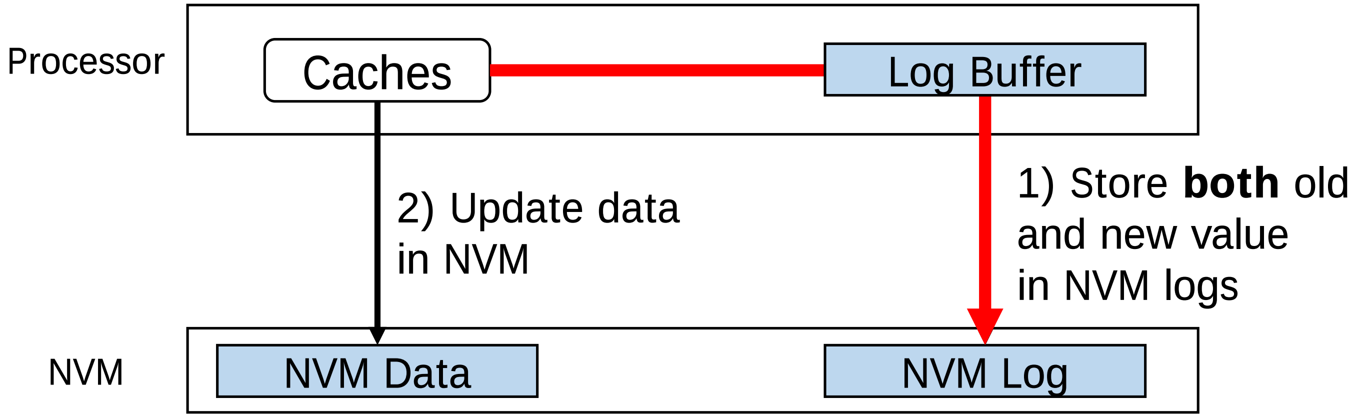
# Past Proposal: Undo-Redo HW-Logging

- Store **both** old and new value in logs

| Addr | **Old** Value | **New** Value |
|---|---|---|

→ **Larger log sizes**

- Update data in NVM **after commit**



Requires **more NVM writes** for storing logs in NVM

- M. Ogleari et al. HPCA 2018.

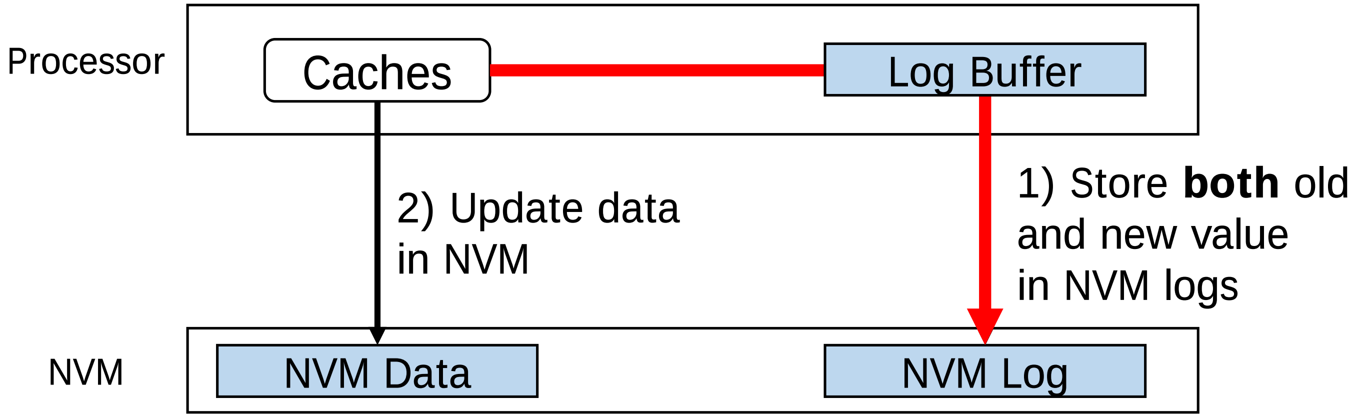
# Past Proposal: Undo-Redo HW-Logging

- Store **both** old and new value in logs

| Addr | **Old** Value | **New** Value |
|---|---|---|

→ **Larger log sizes**

- Update data in NVM **after commit**



Requires **more NVM writes** for storing logs in NVM

- M. Ogleari et al. HPCA 2018.

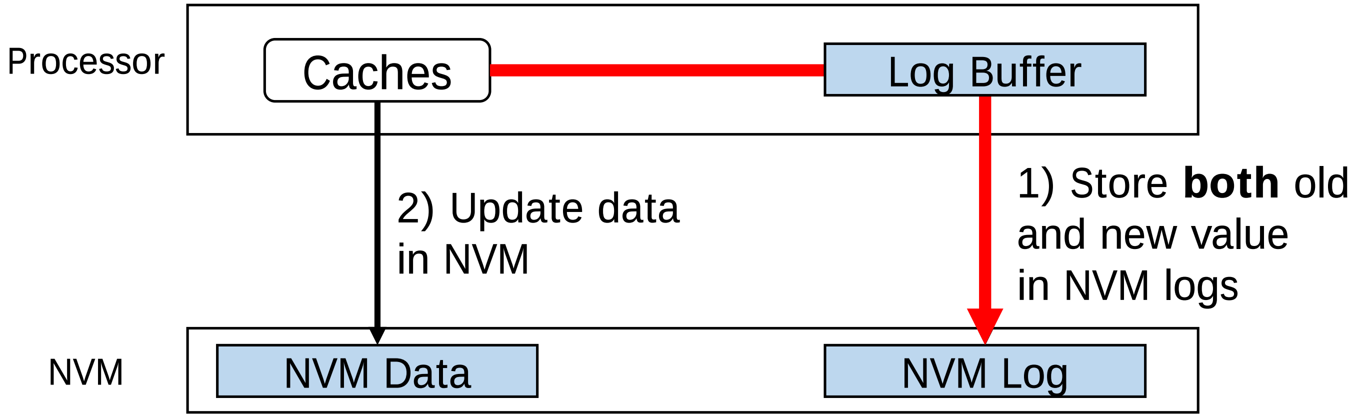
# Past Proposal: Undo-Redo HW-Logging

- Store **both** old and new value in logs

| Addr | **Old** Value | **New** Value |
|---|---|---|

→ **Larger log sizes**

- Update data in NVM **after commit**



Requires **more NVM writes** for storing logs in NVM

- M. Ogleari et al. HPCA 2018.

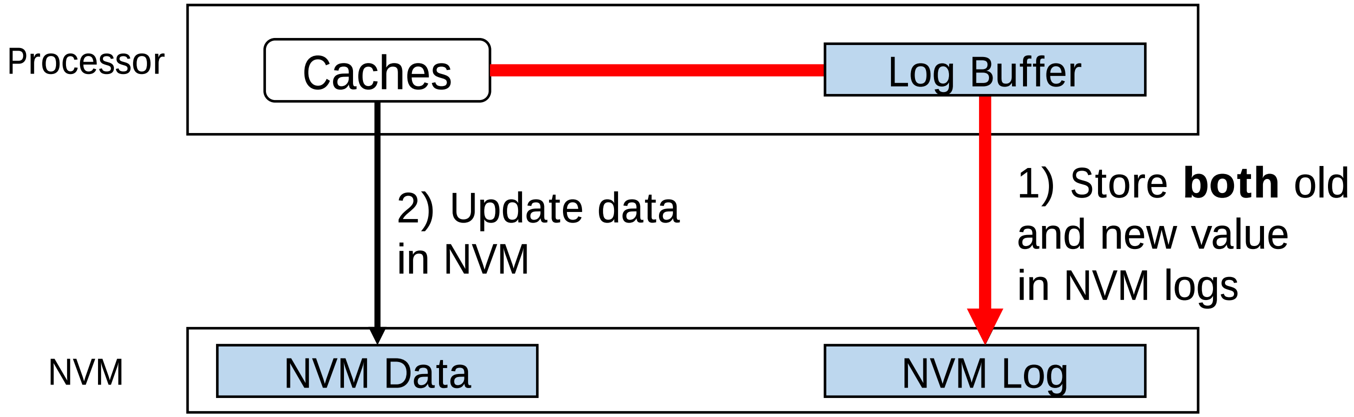
# Past Proposal: Undo-Redo HW-Logging

- Store **both** old and new value in logs

| Addr | **Old** Value | **New** Value |
|---|---|---|

➔ **Larger log sizes**

- Update data in NVM **after commit**



1) Store **both** old and new value in NVM logs

2) Update data in NVM

Requires **more NVM writes** for storing logs in NVM

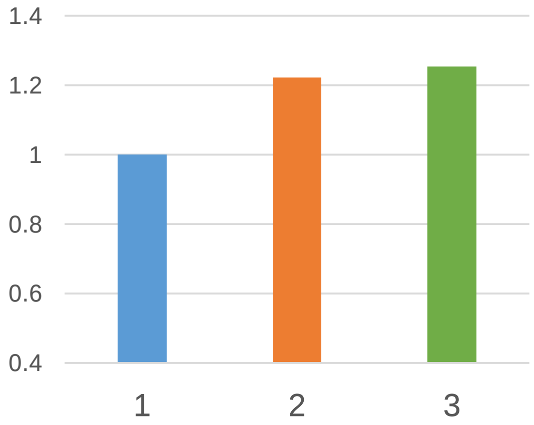
- M. Ogleari et al. HPCA 2018.

# Past Proposals: Summary

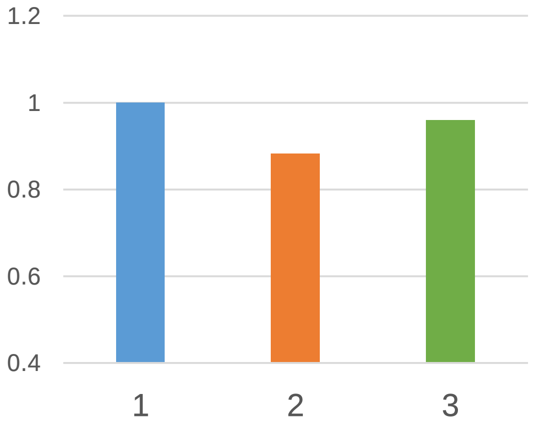
| | **Log-Write** | **Data-Update** | | **Drawback** |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

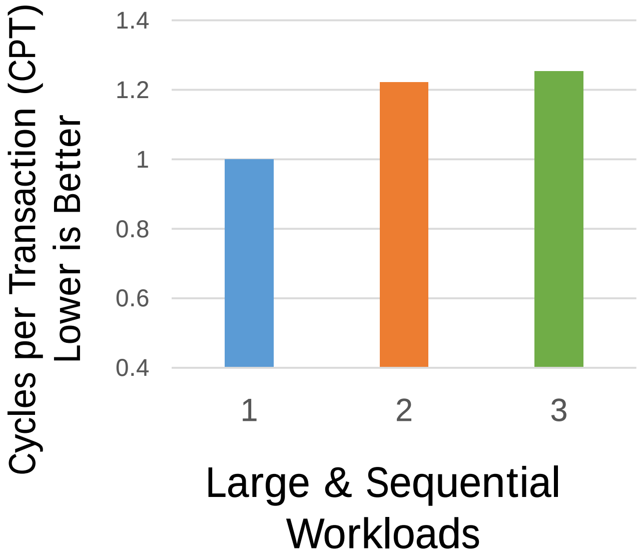


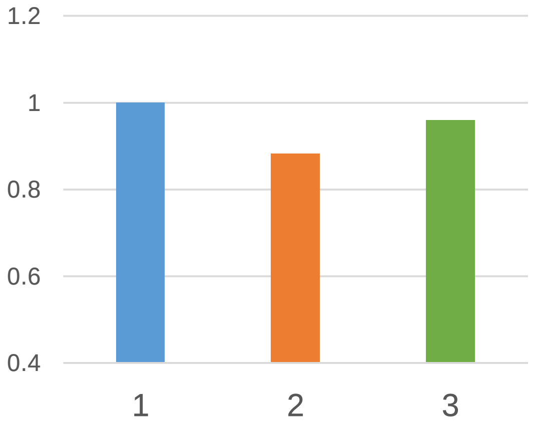
Large & Sequential Workloads



# Past Proposals: Summary

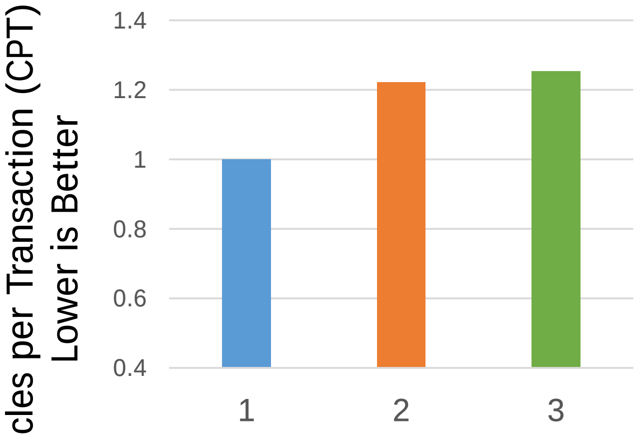
| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |



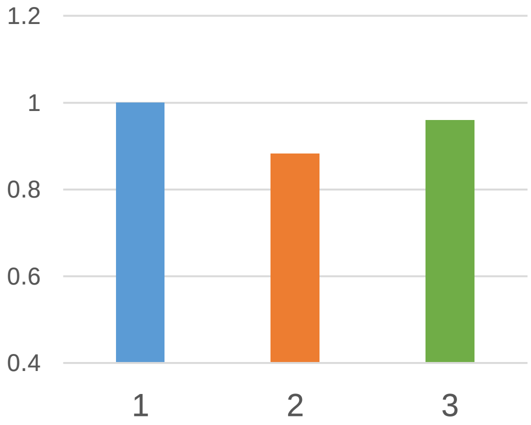


# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |



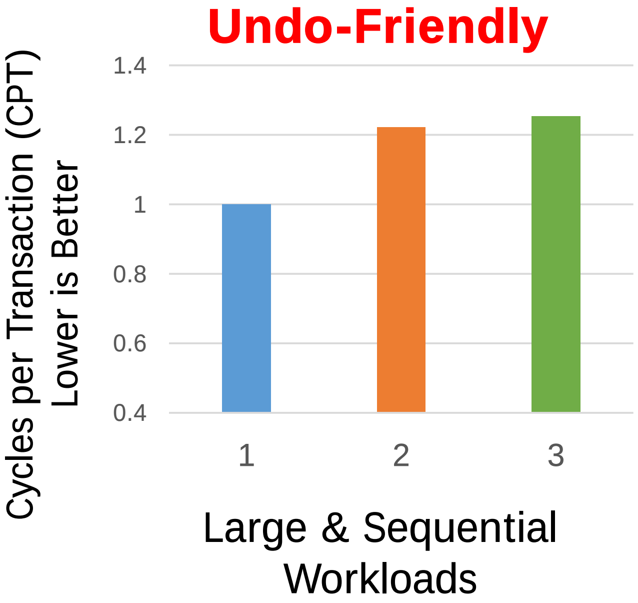
Large & Sequential Workloads

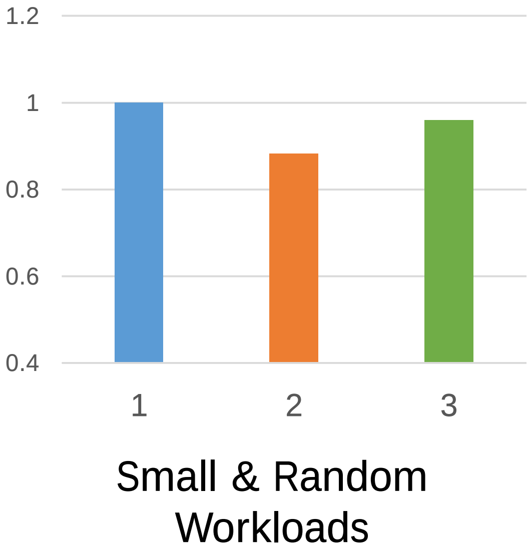


Small & Random Workloads

# Past Proposals: Summary

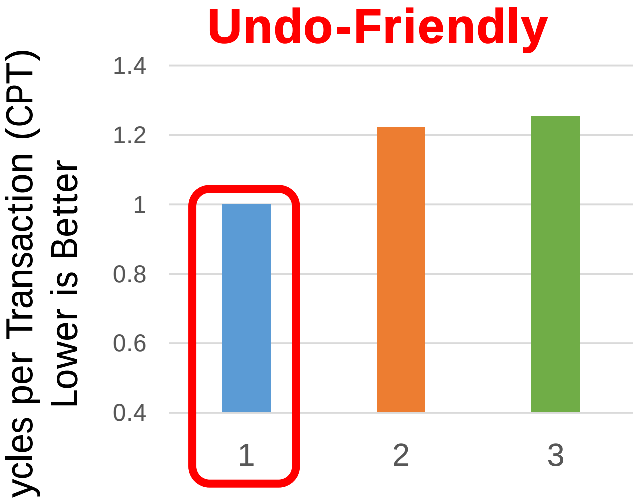
| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |



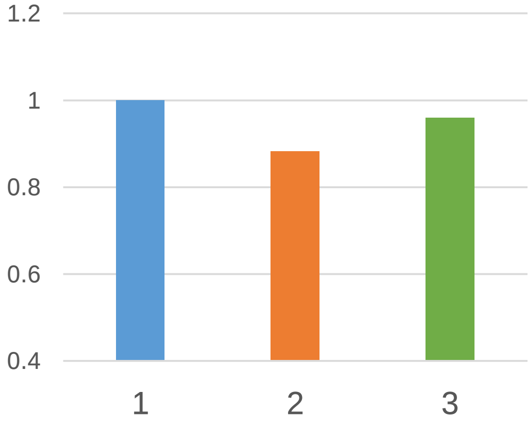


# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |



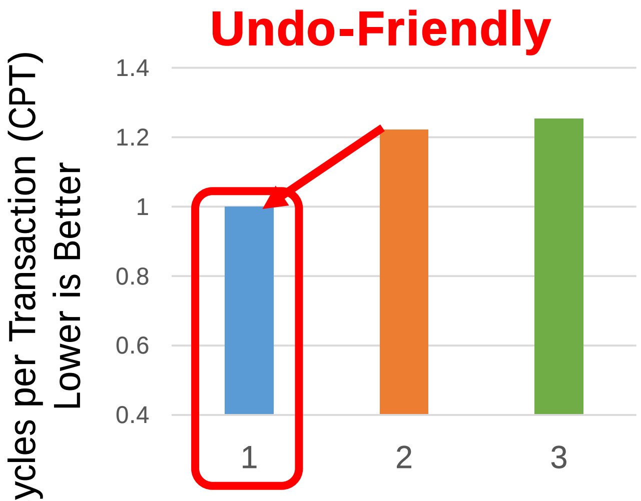
Large & Sequential Workloads



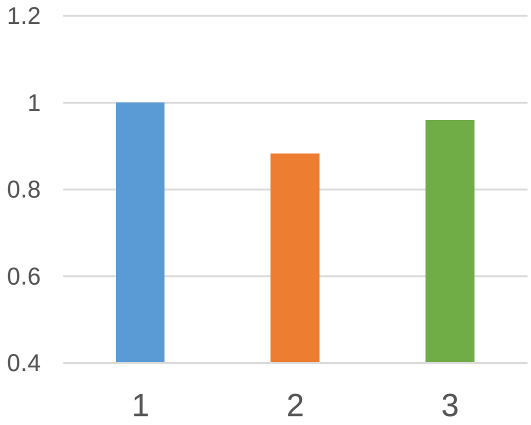
Small & Random Workloads

# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |



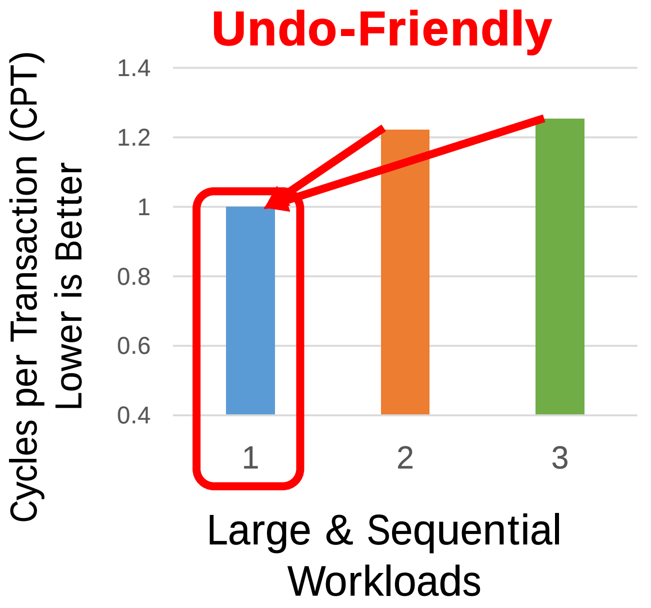
Large & Sequential Workloads

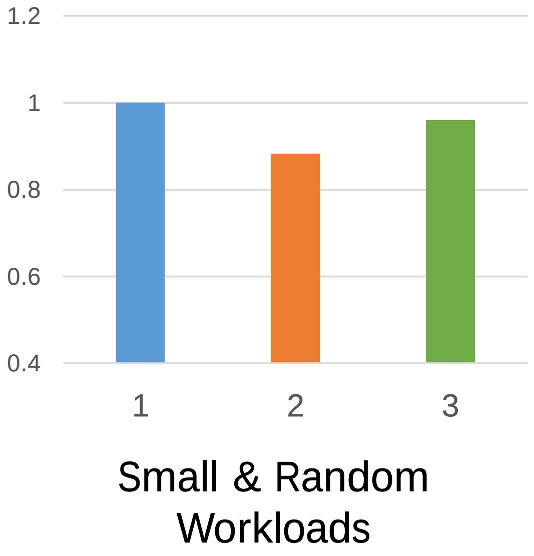


Small & Random Workloads

# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

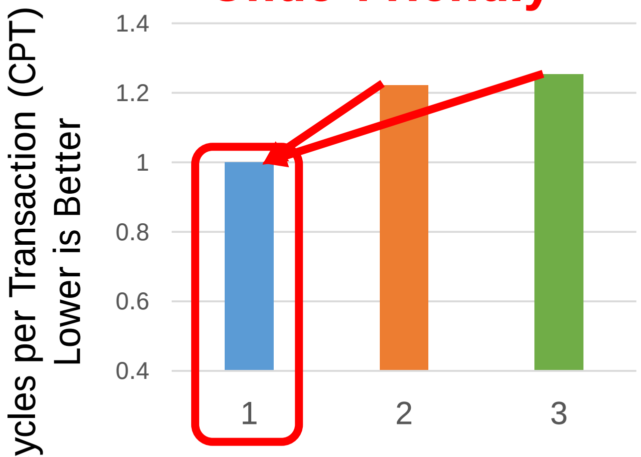




# Past Proposals: Summary

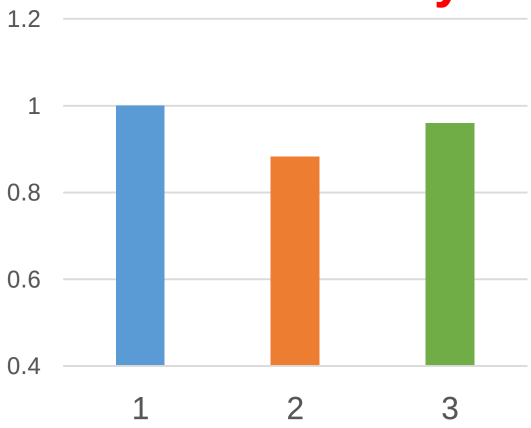
| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

**Undo-Friendly**



Large & Sequential Workloads

**Redo-Friendly**

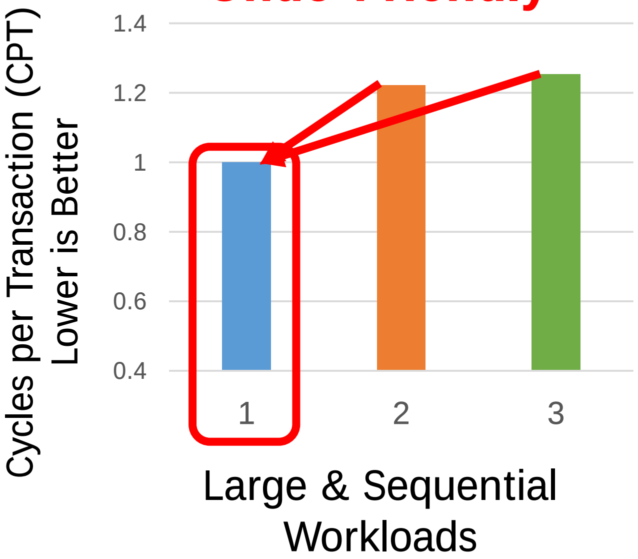


Small & Random Workloads

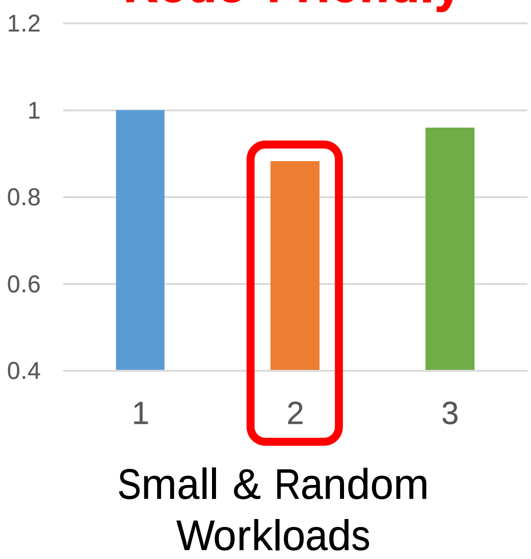
# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

**Undo-Friendly**

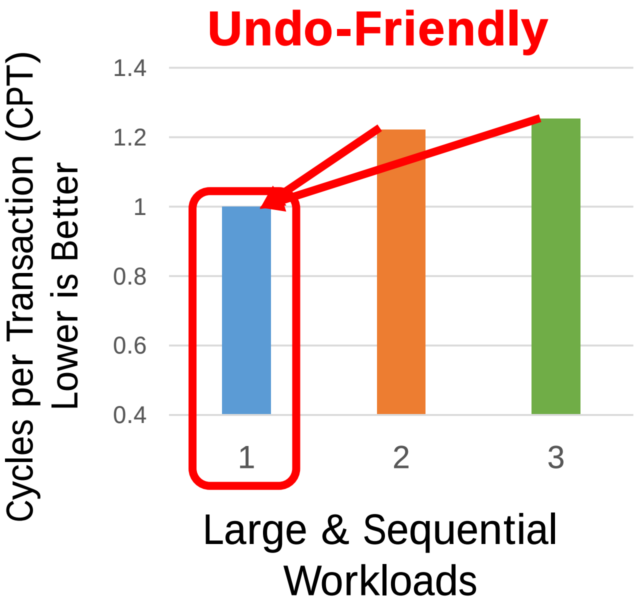


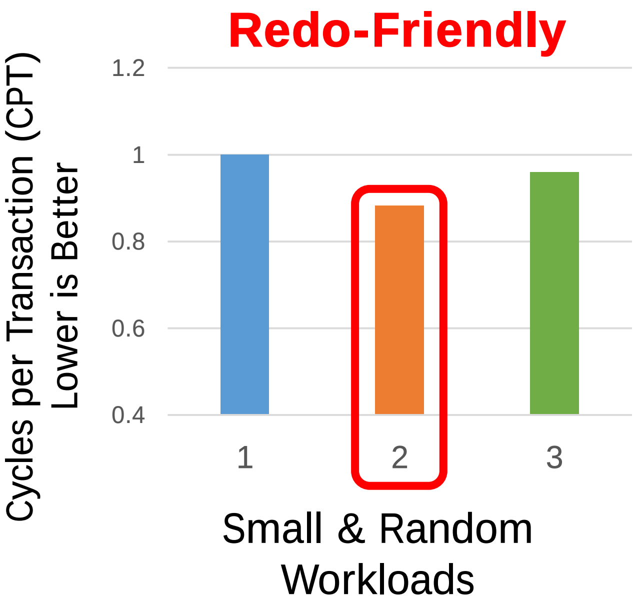
**Redo-Friendly**



# Past Proposals: Summary

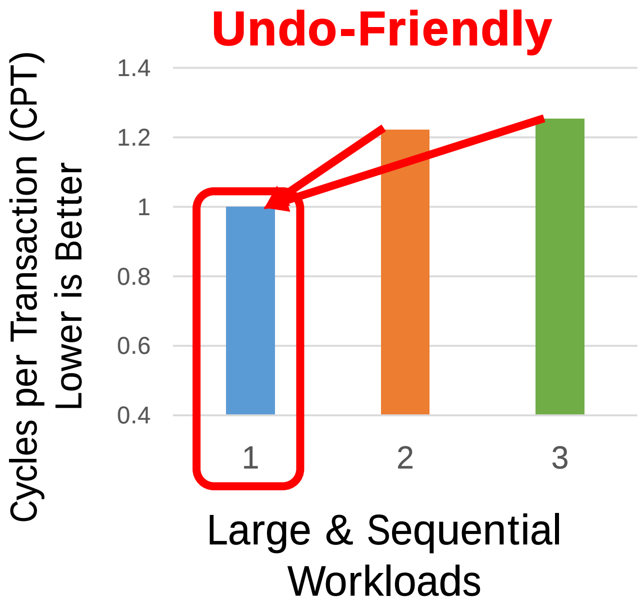
| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

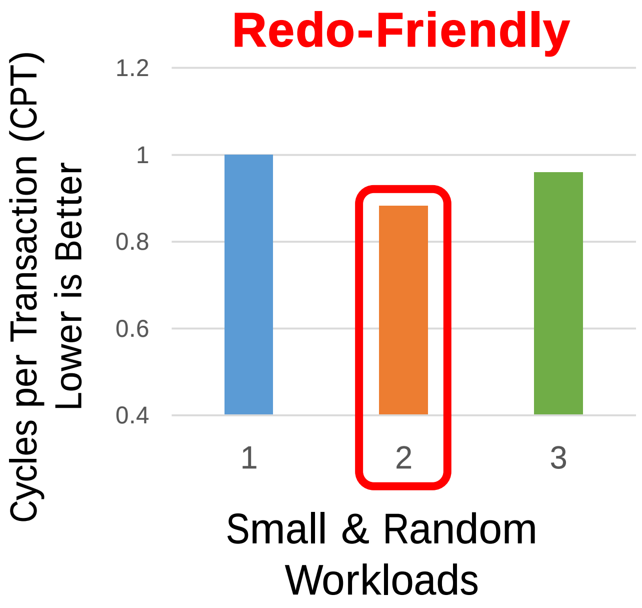




# Past Proposals: Summary

| | **Log-Write** | **Data-Update** | | **Drawback** |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

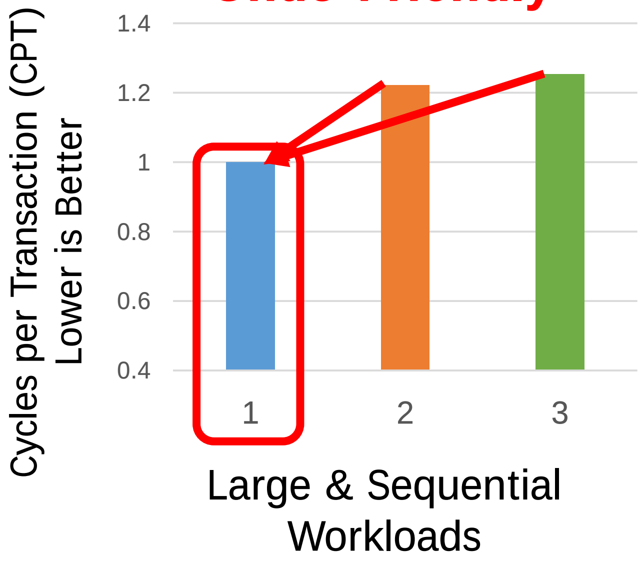

**Undo-Friendly**


**Redo-Friendly**

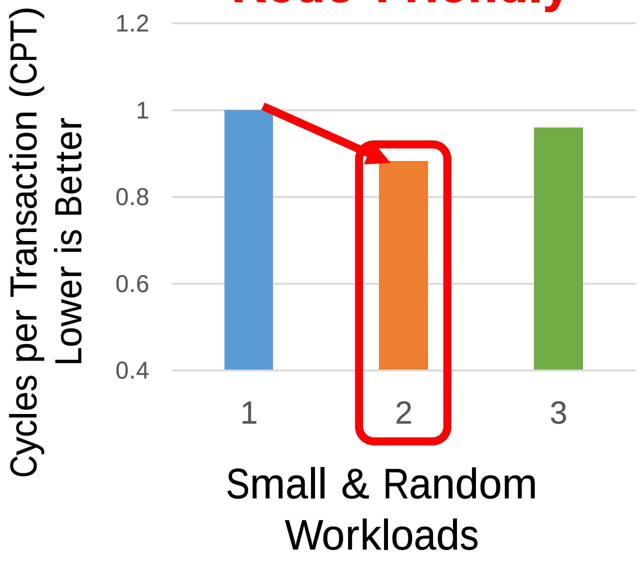
# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

**Undo-Friendly**

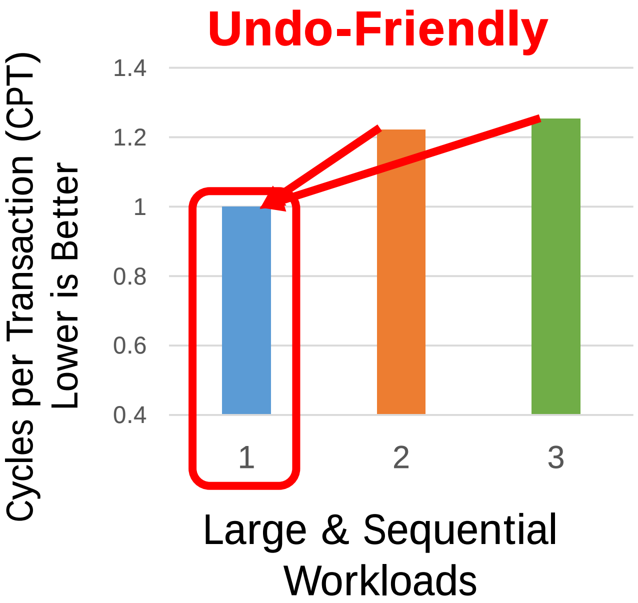


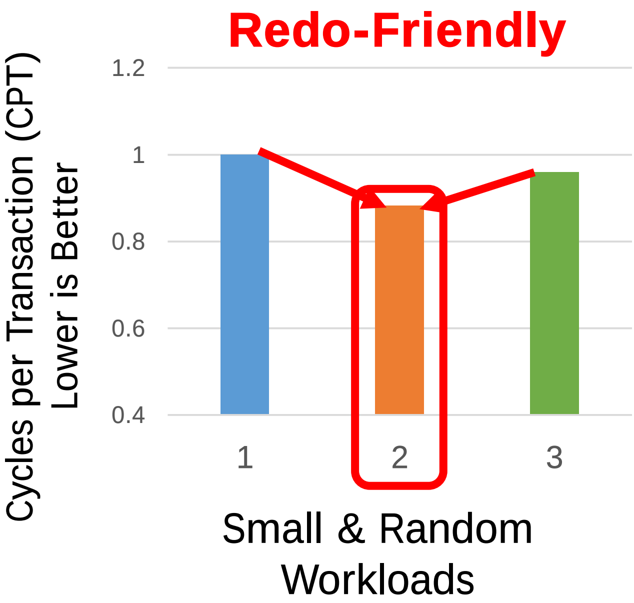
**Redo-Friendly**



# Past Proposals: Summary

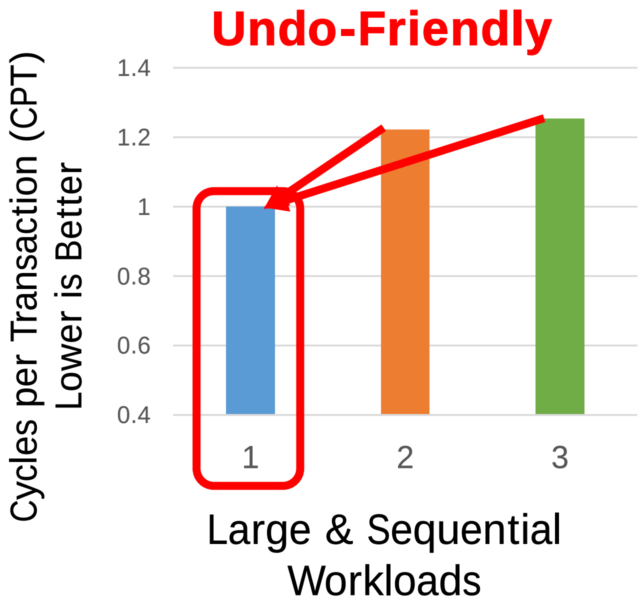
| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

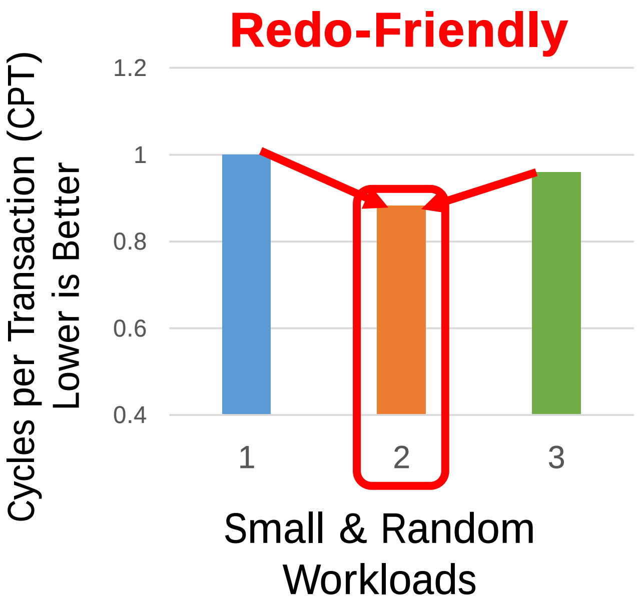




# Past Proposals: Summary

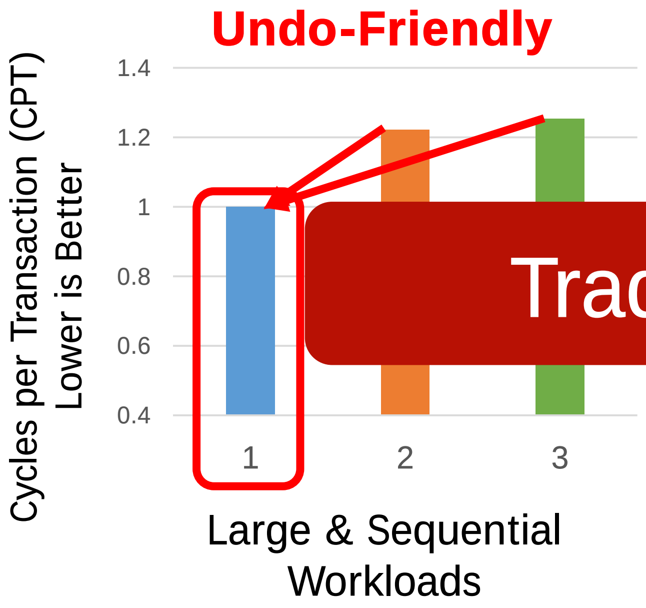
| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |

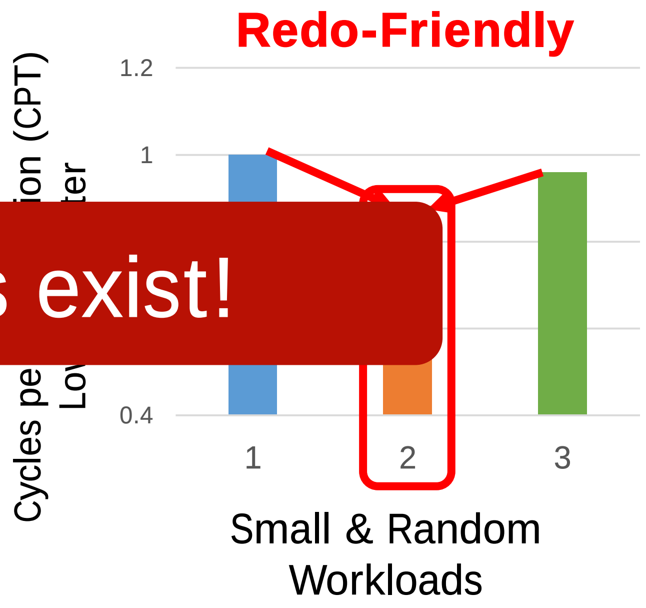

**Undo-Friendly**


**Redo-Friendly**

# Past Proposals: Summary

| | Log-Write | Data-Update | | Drawback |
|---|---|---|---|---|
| ATOM [HPCA 2017] | Undo | Direct | **Synchronous** | **Long Critical Path** |
| Proteus [ISCA 2017] | Undo | Direct | **Synchronous** | |
| Wrap [HPCA 2016] | Redo | **Indirect** | Asynchronous | **Waste NVM Bandwidth** |
| FWB [HPCA 2018] | **UndoRedo** | Direct | Asynchronous | **More Log Write** |





Trade-offs exist!

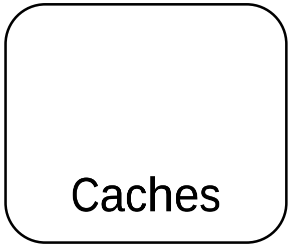
# Design Goal & Challenges

# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches

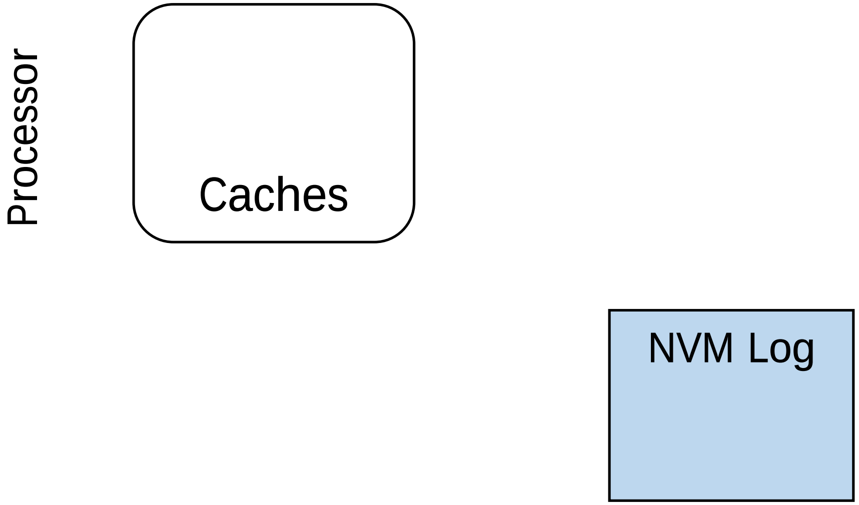
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



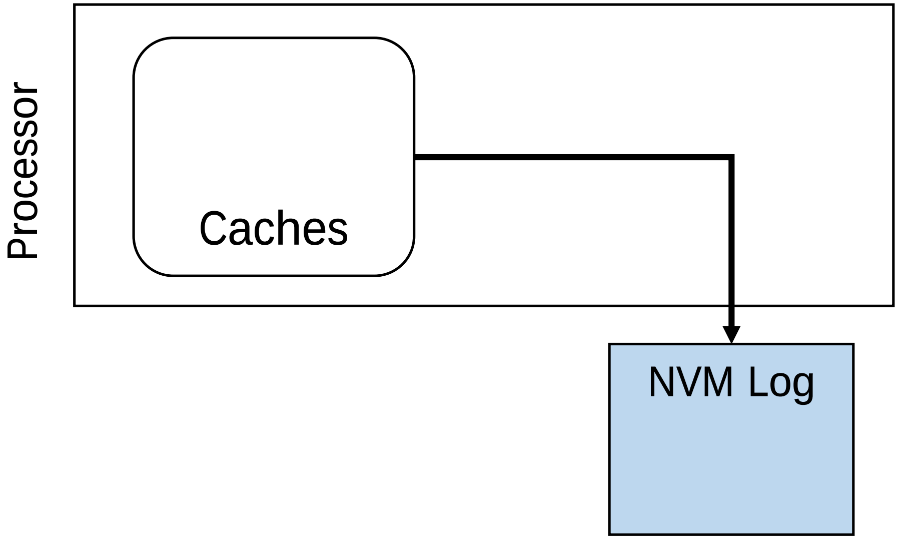
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



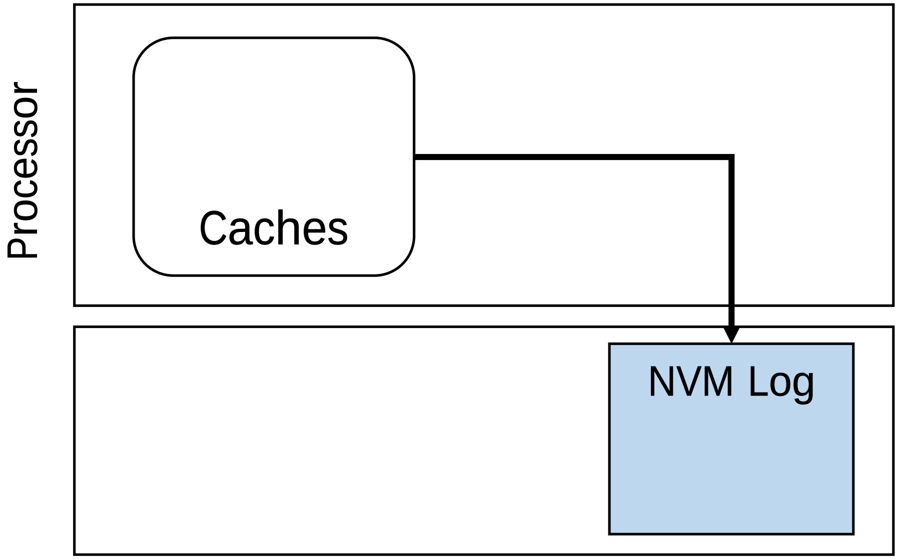
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



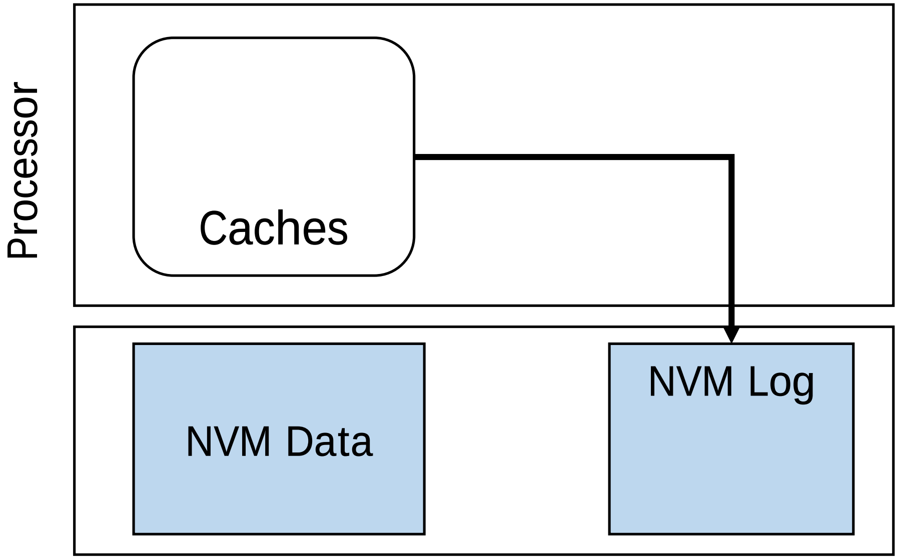
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



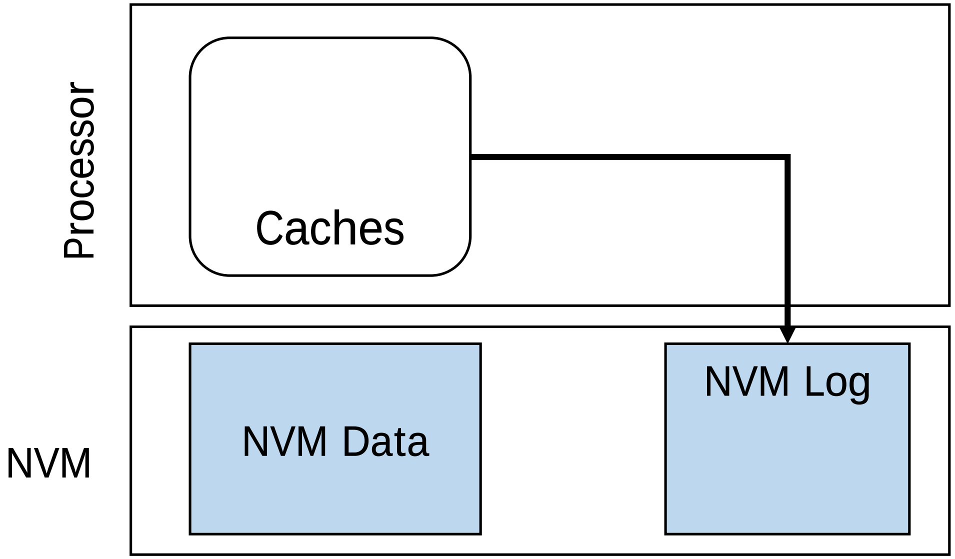
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



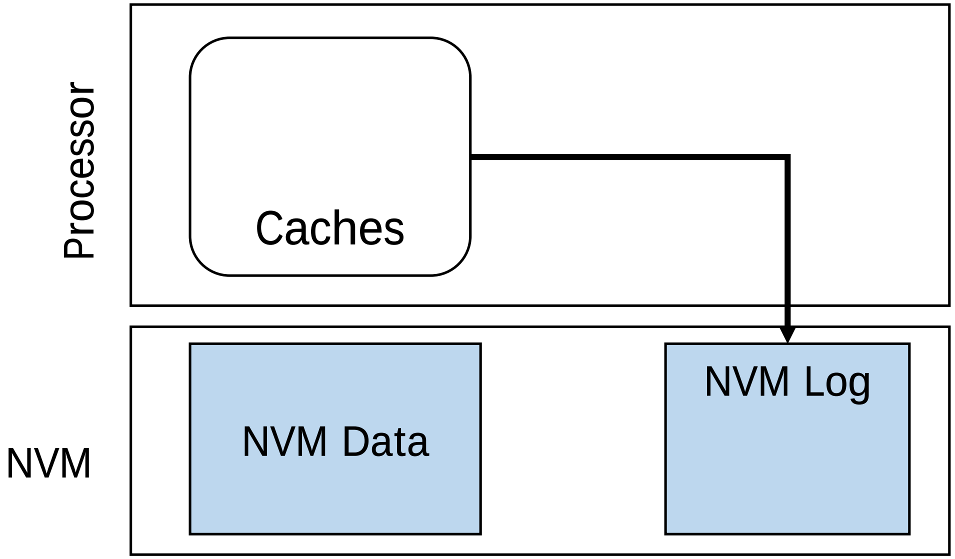
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
    - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
    - Eviction of uncommitted changes from volatile CPU caches



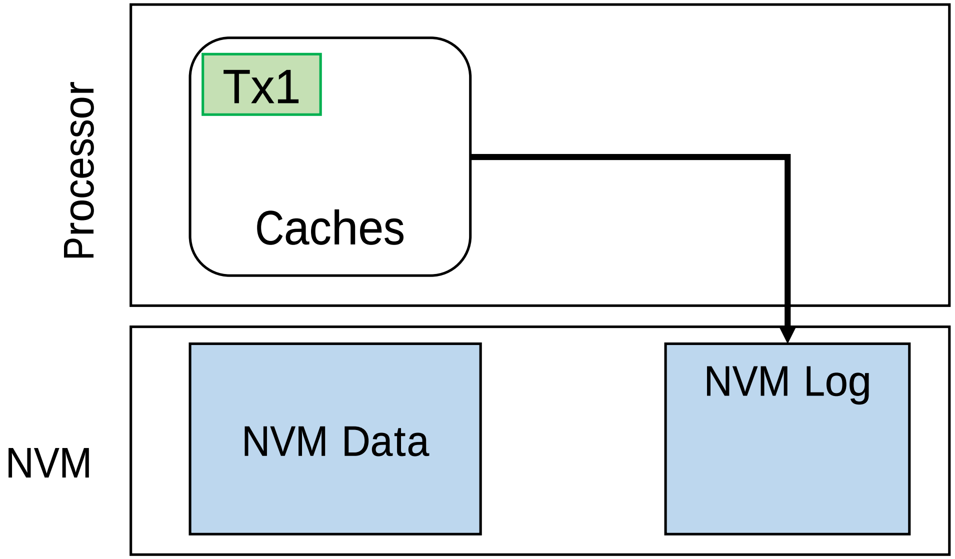
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



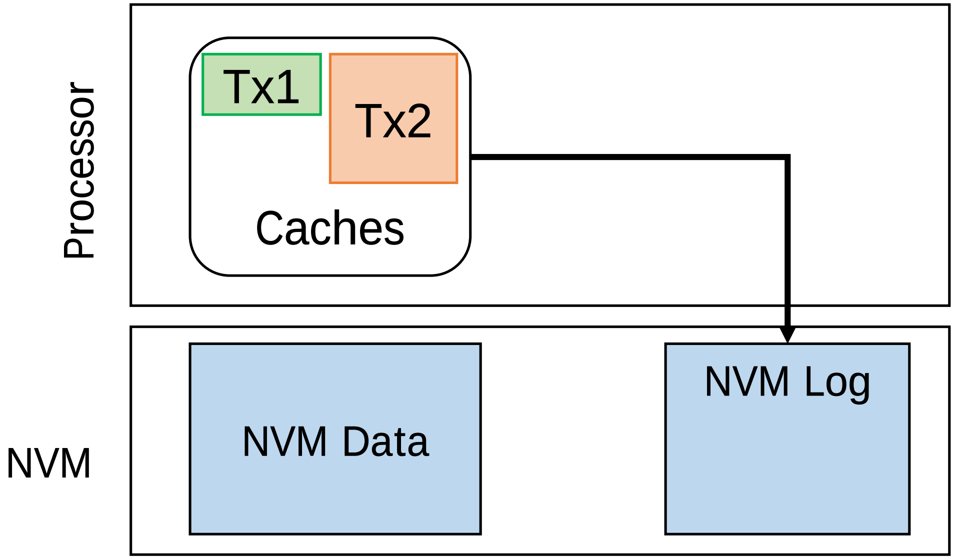
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



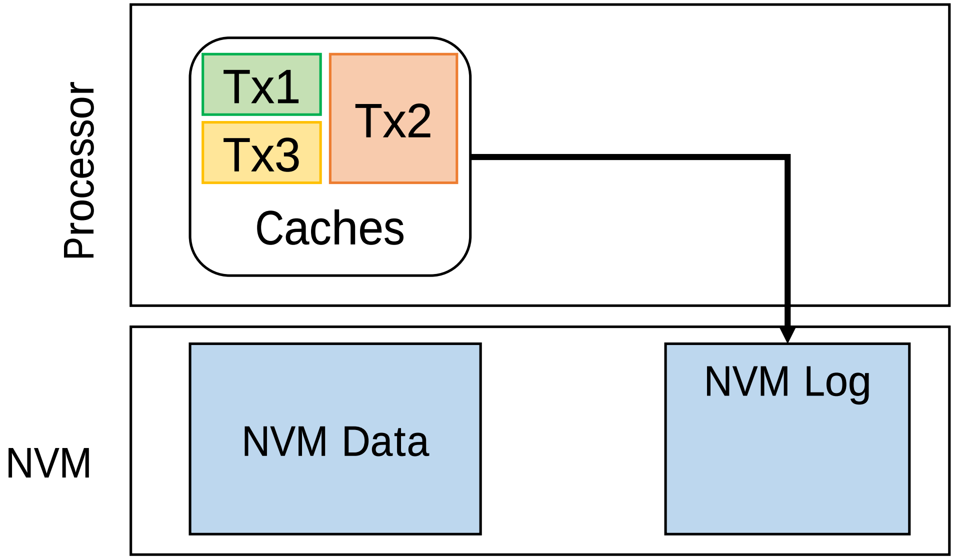
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



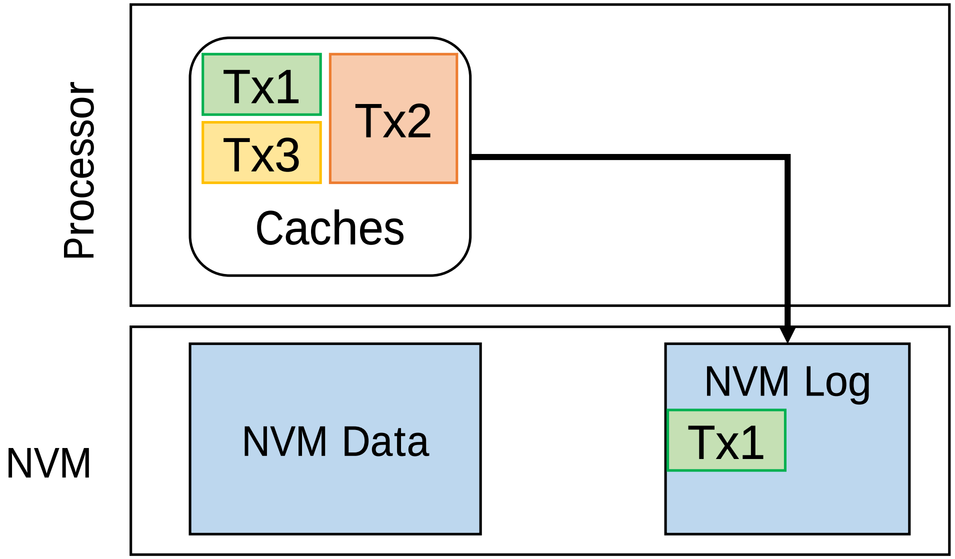
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



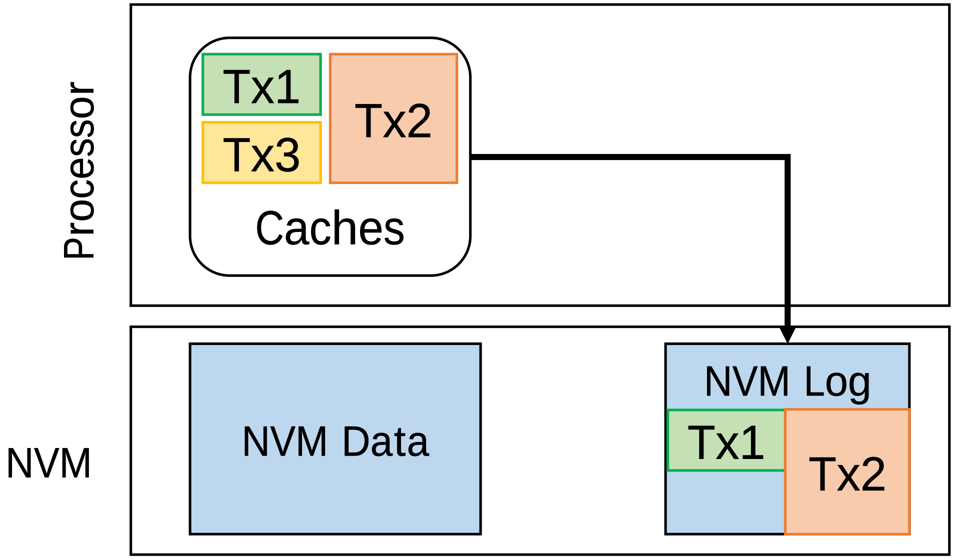
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



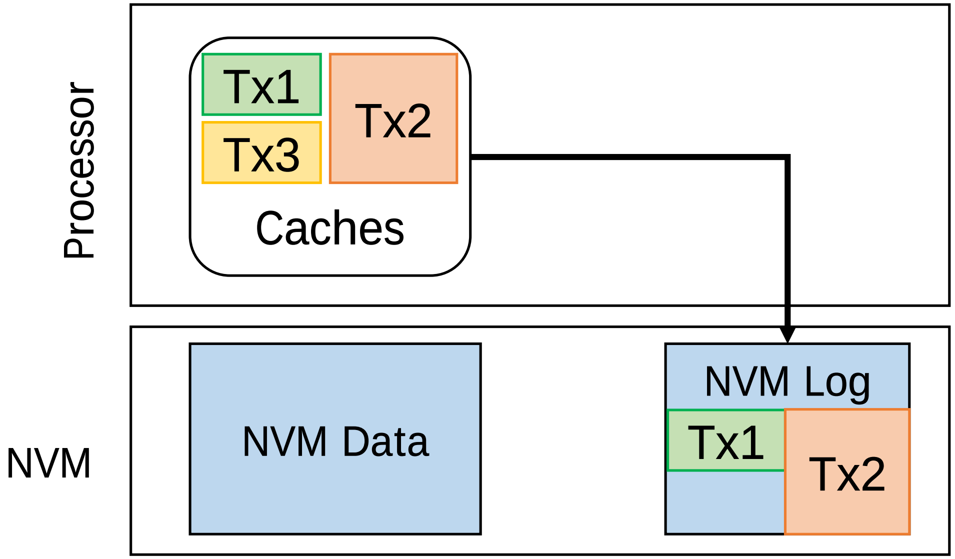
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



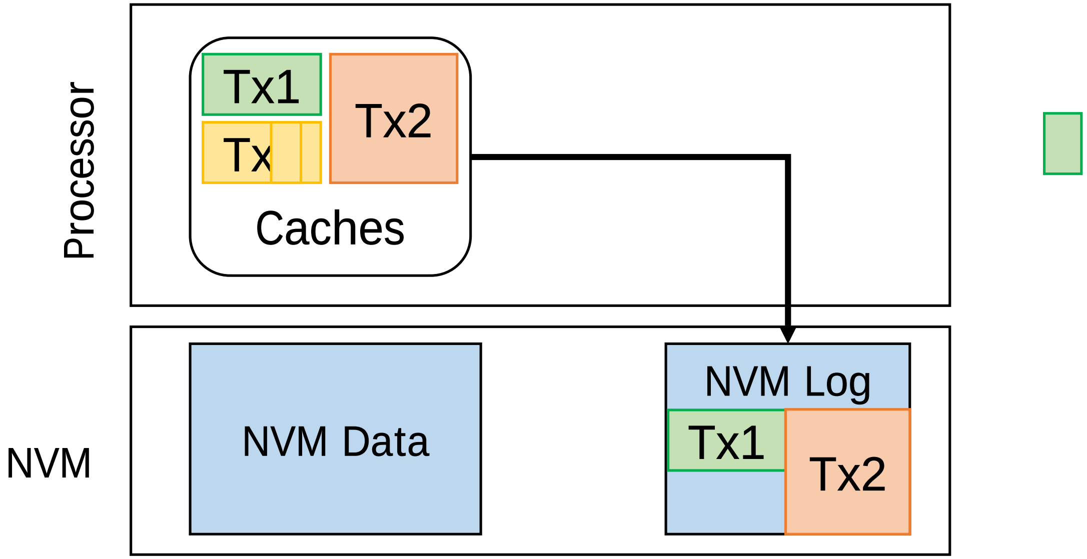
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



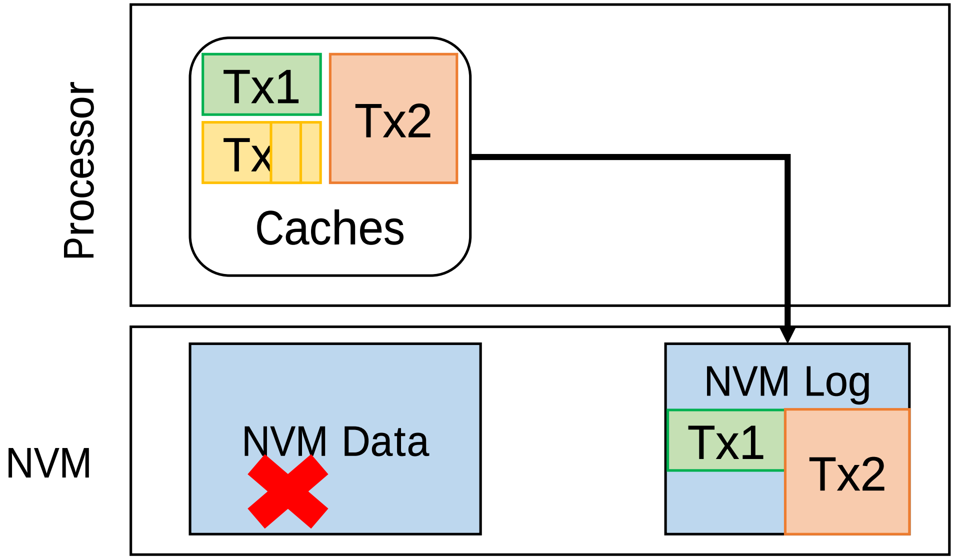
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



# Design Goal & Challenges

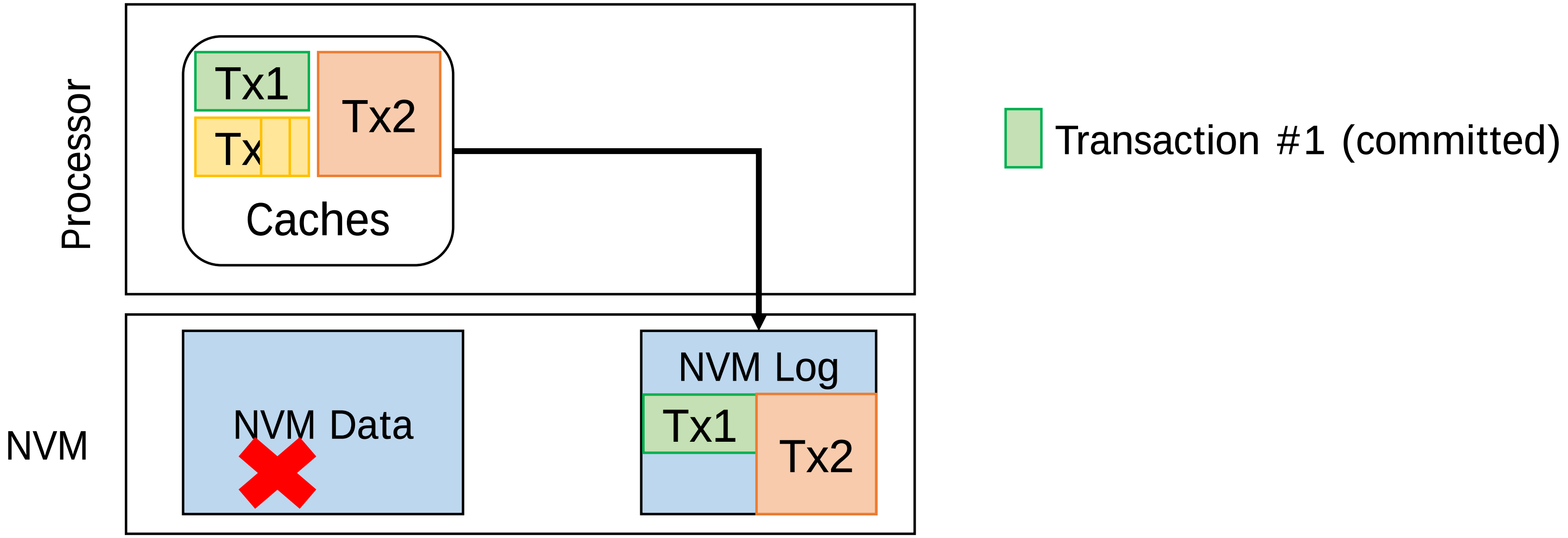
- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



🟩 Transaction #1

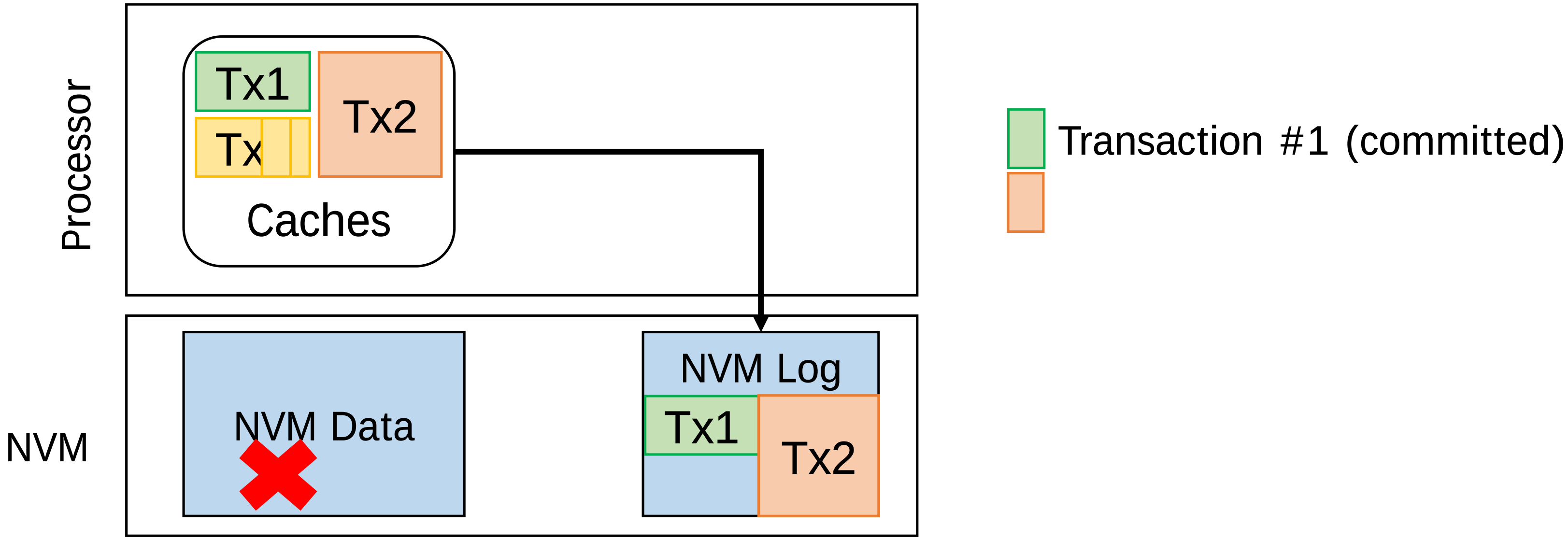
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



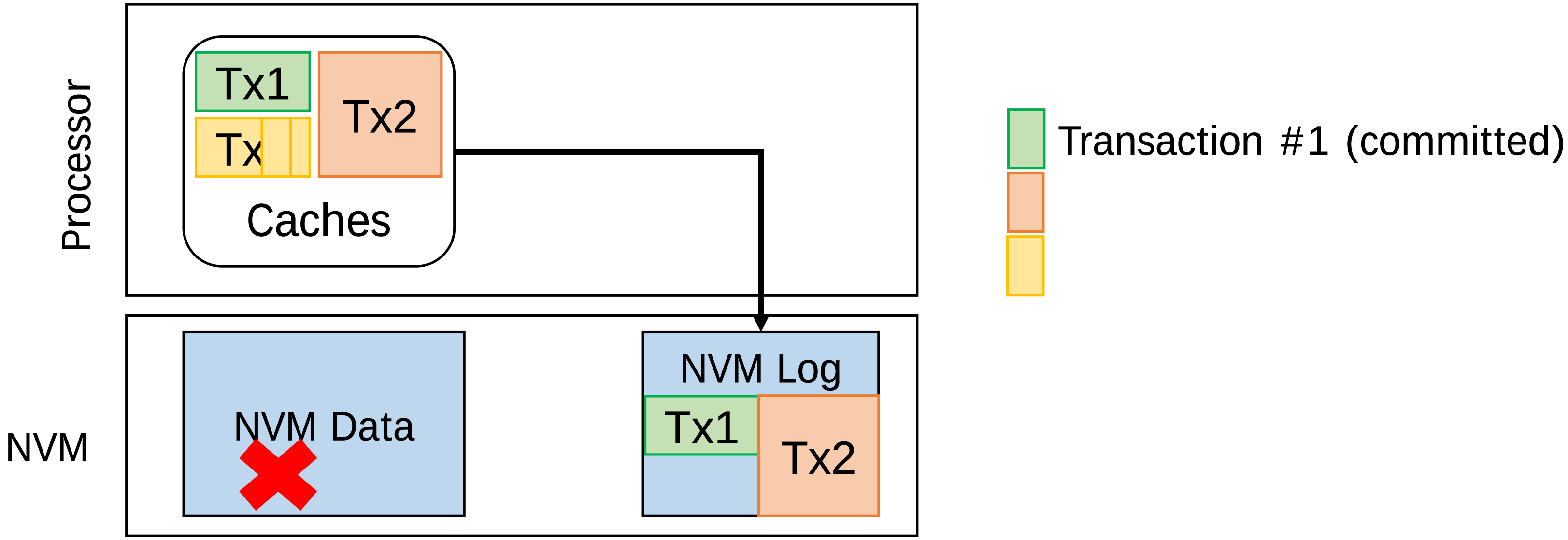
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



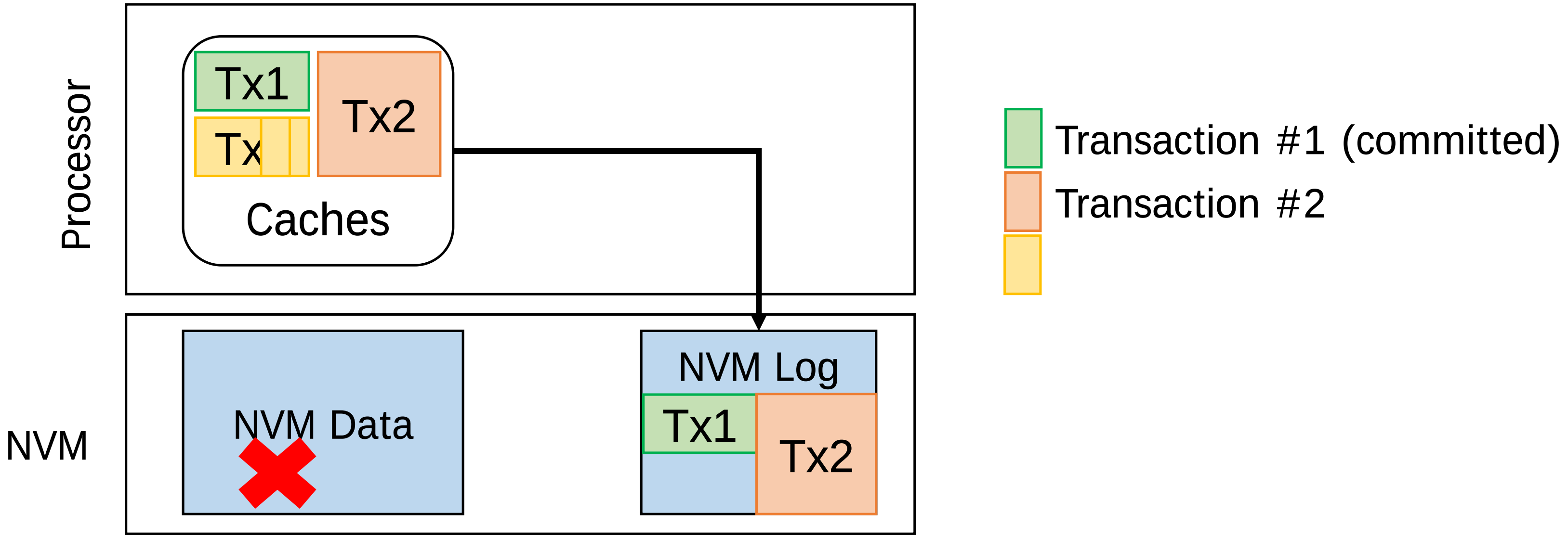
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



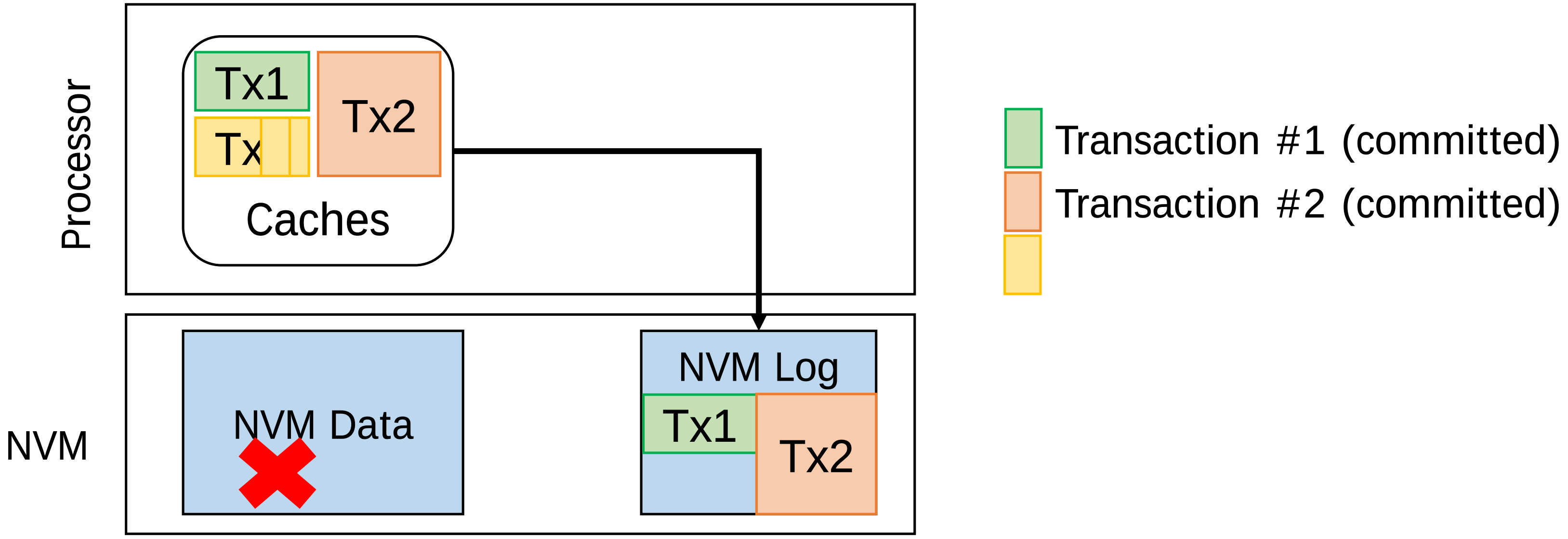
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



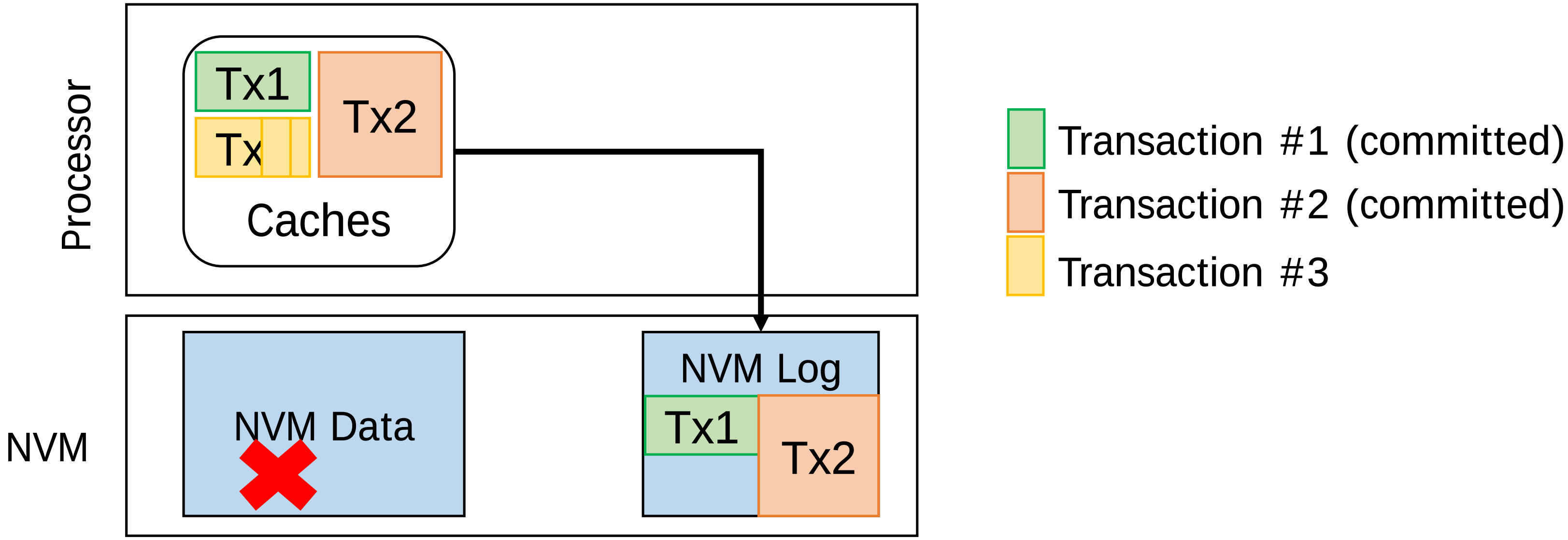
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



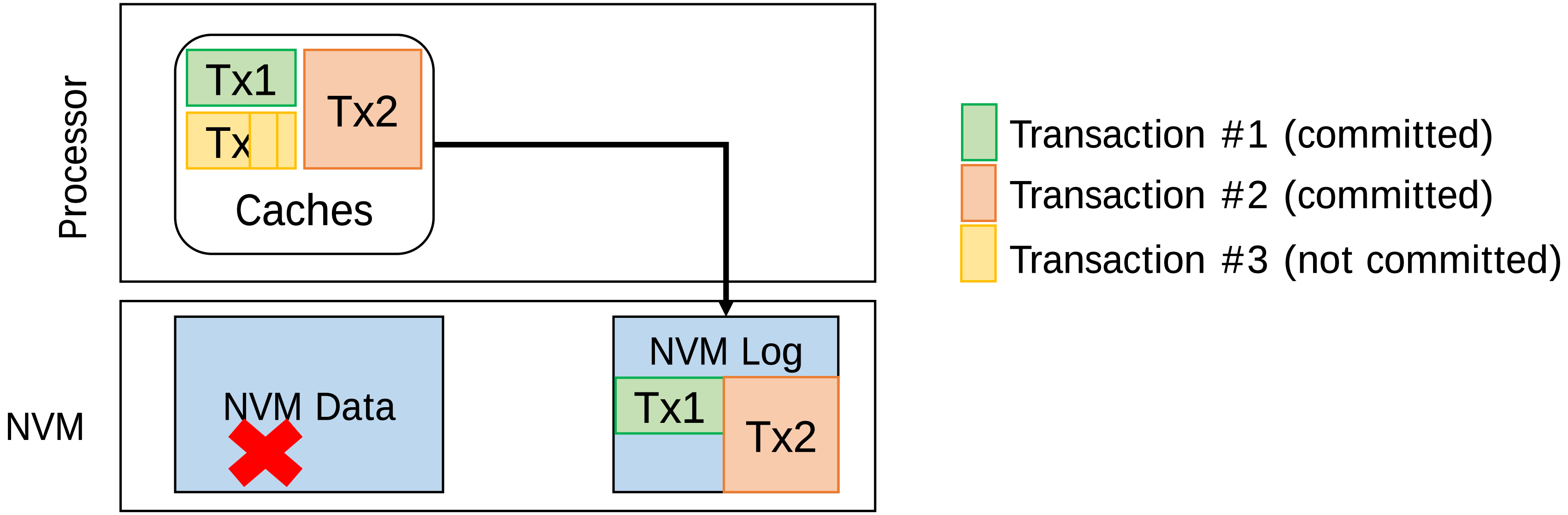
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



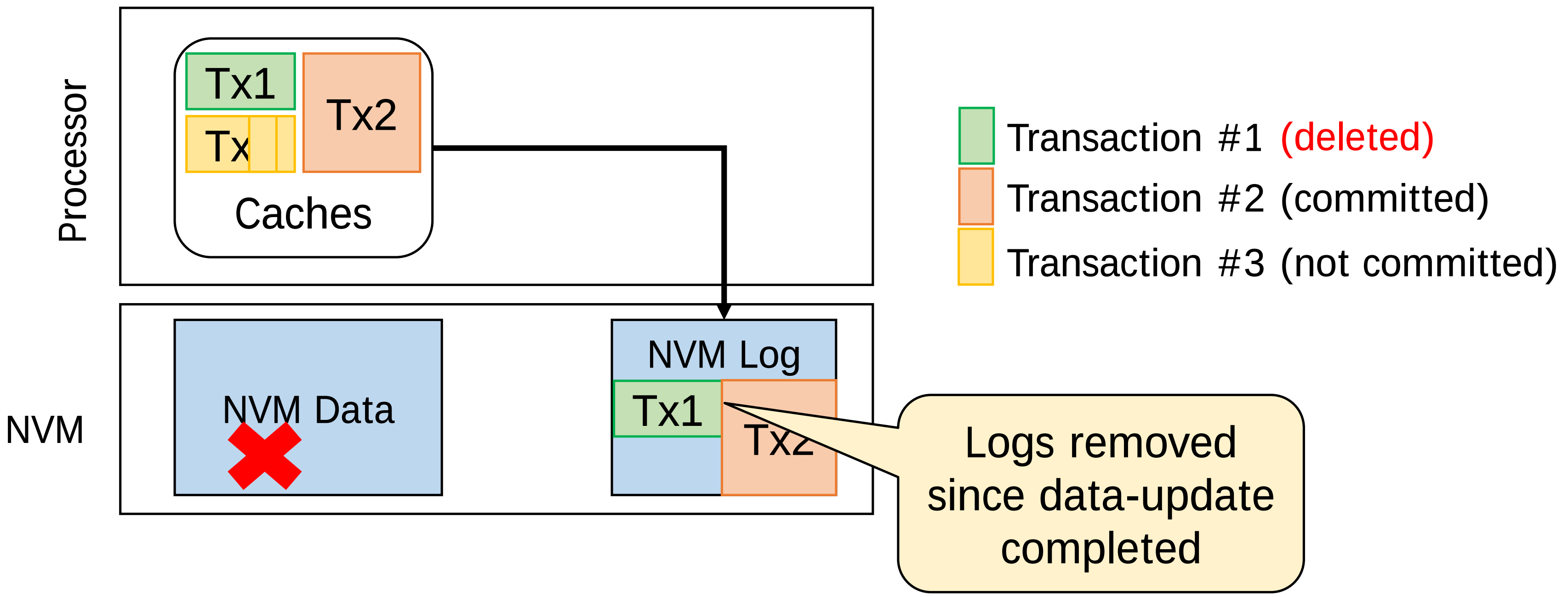
# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches


Processor
Tx1
Tx2
Tx
Caches
NVM
NVM Data
NVM Log
Tx1
Tx2
Transaction #1 (committed)
Transaction #2 (committed)
Transaction #3 (not committed)

# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches

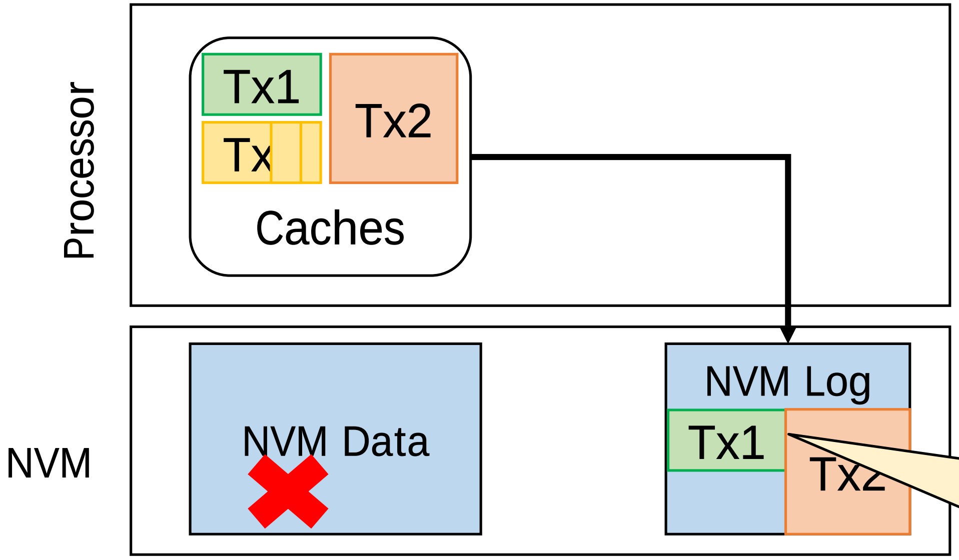


🟩 Transaction #1 (deleted)
🟧 Transaction #2 (committed)
🟨 Transaction #3 (not committed)

Logs removed since data-update completed

# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches

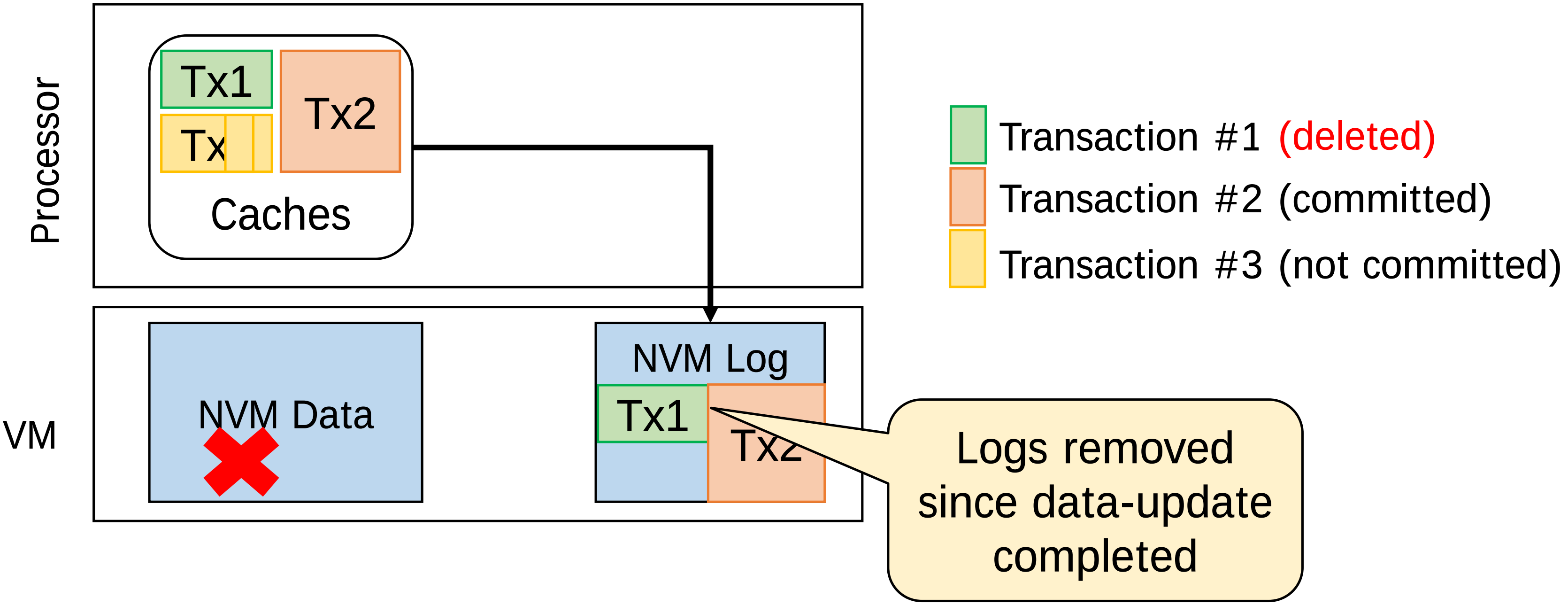


🟩 Transaction #1 (deleted)
🟧 Transaction #2 (committed)
🟨 Transaction #3 (not committed)

Logs removed since data-update completed

# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



🟩 Transaction #1 (deleted)

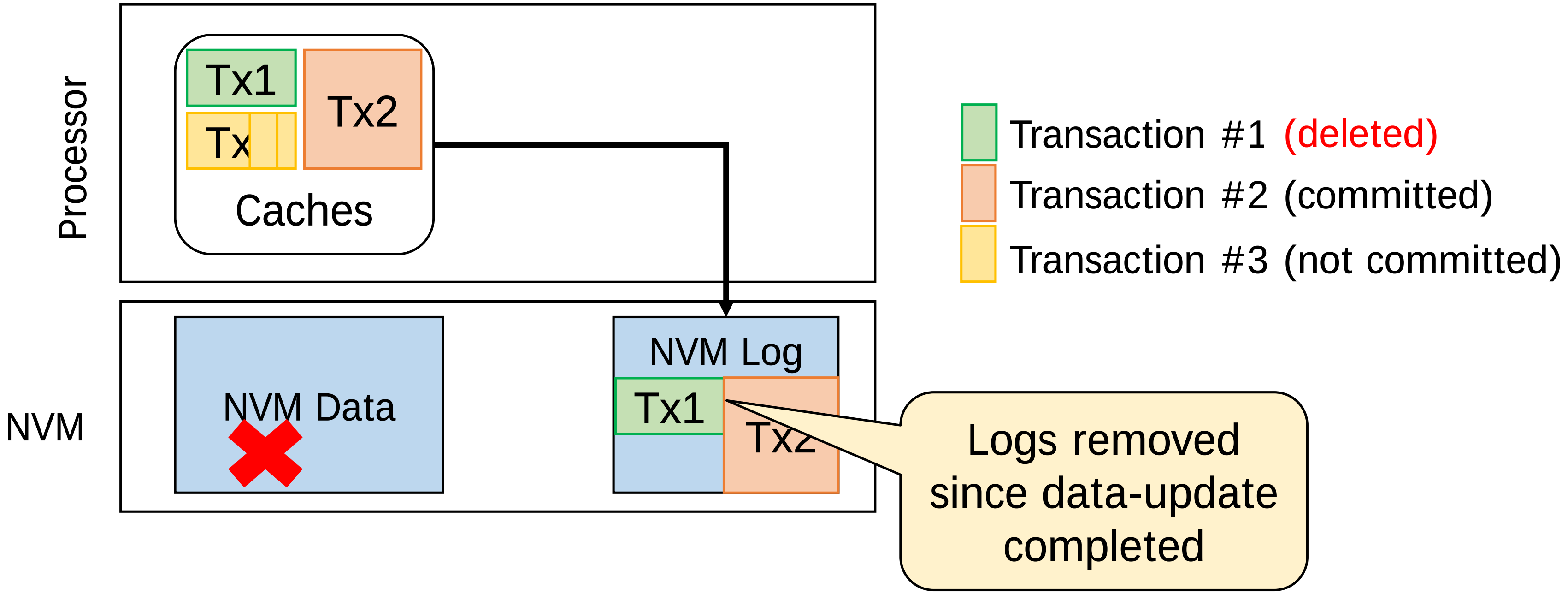
🟧 Transaction #2 (committed)

🟨 Transaction #3 (not committed)

Logs removed since data-update completed

# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches

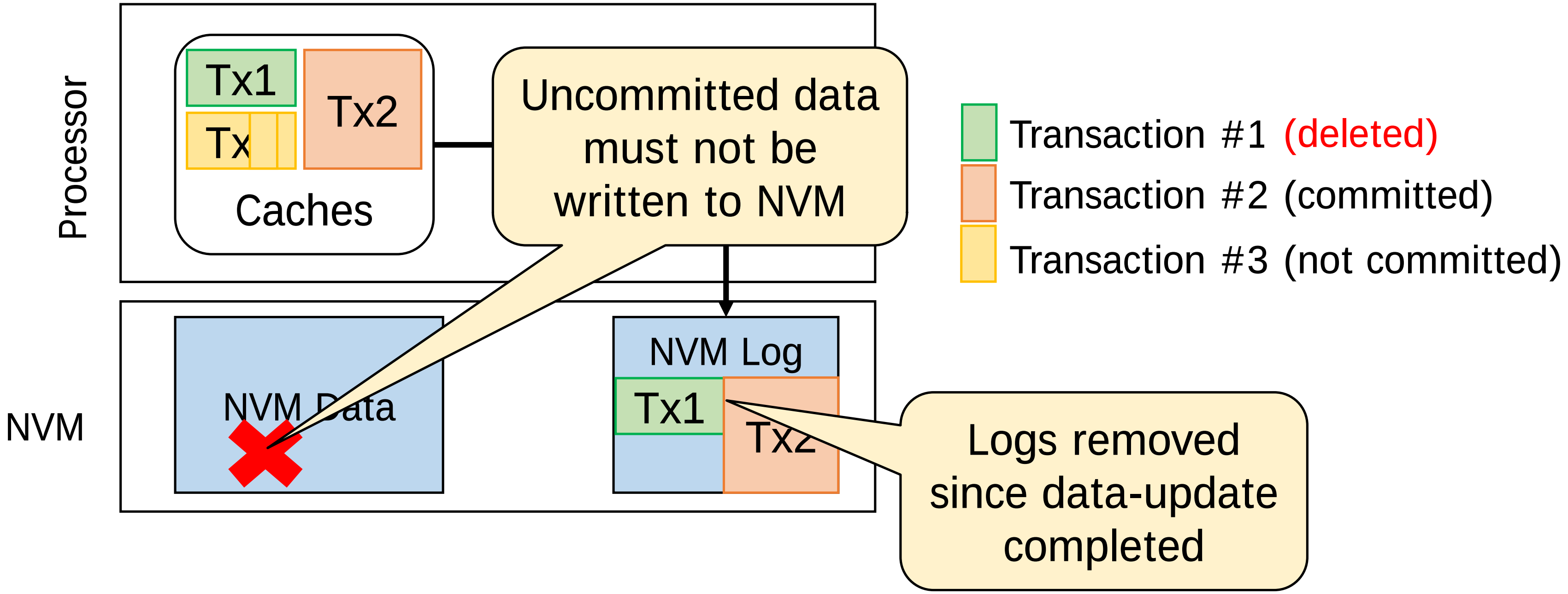


🟩 Transaction #1 (deleted)
🟧 Transaction #2 (committed)
🟨 Transaction #3 (not committed)

Logs removed since data-update completed

# Design Goal & Challenges

- Redo log with asynchronous & direct update to NVM
- Challenge #1: tracking write-sets of previous transactions
  - Without data update, logs keep growing
- Challenge #2: handling an early-eviction
  - Eviction of uncommitted changes from volatile CPU caches



Uncommitted data must not be written to NVM

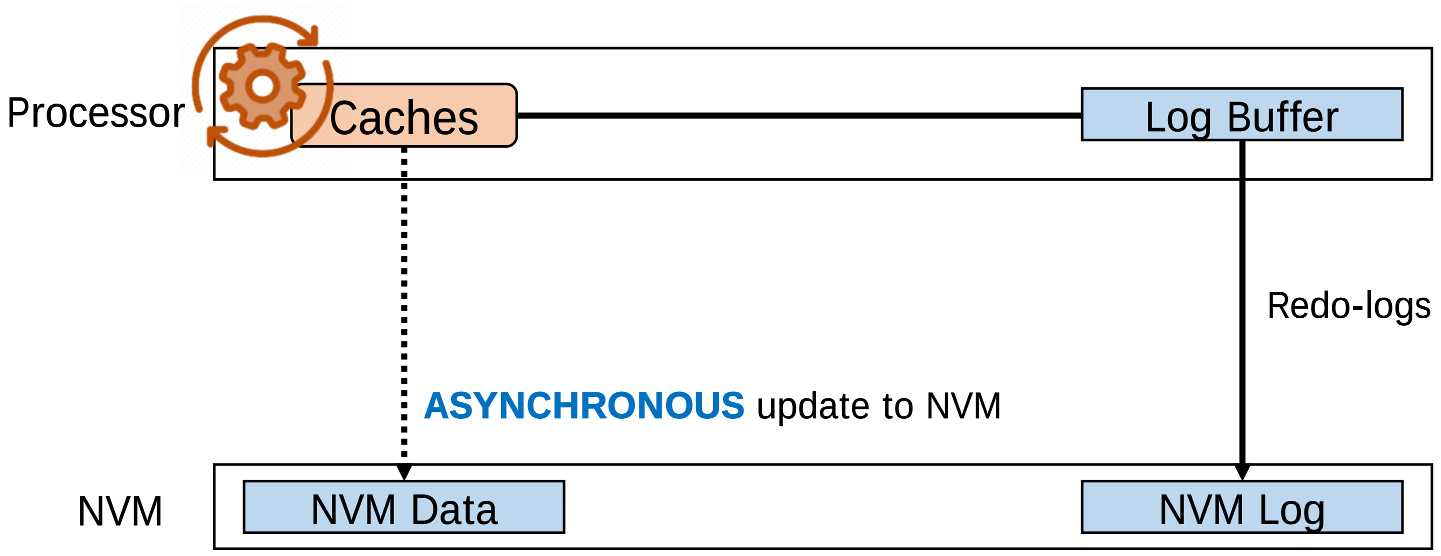
Logs removed since data-update completed

🟩 Transaction #1 (deleted)

🟧 Transaction #2 (committed)

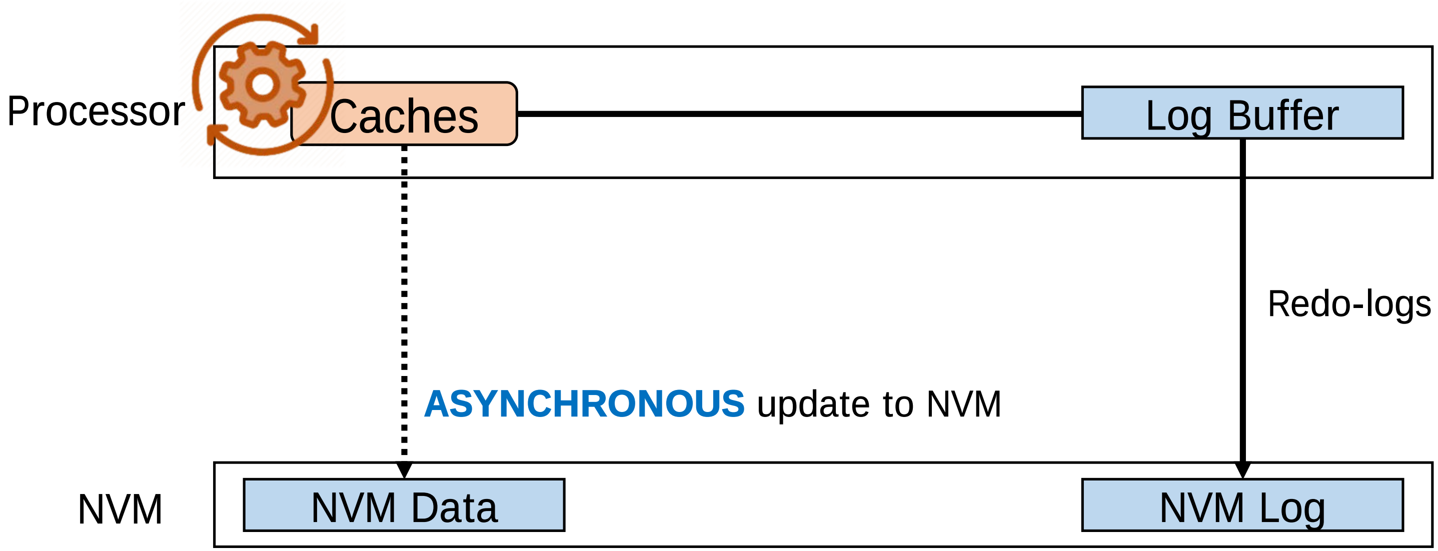
🟨 Transaction #3 (not committed)

# Naïve Solution: On-chip Cache Extension



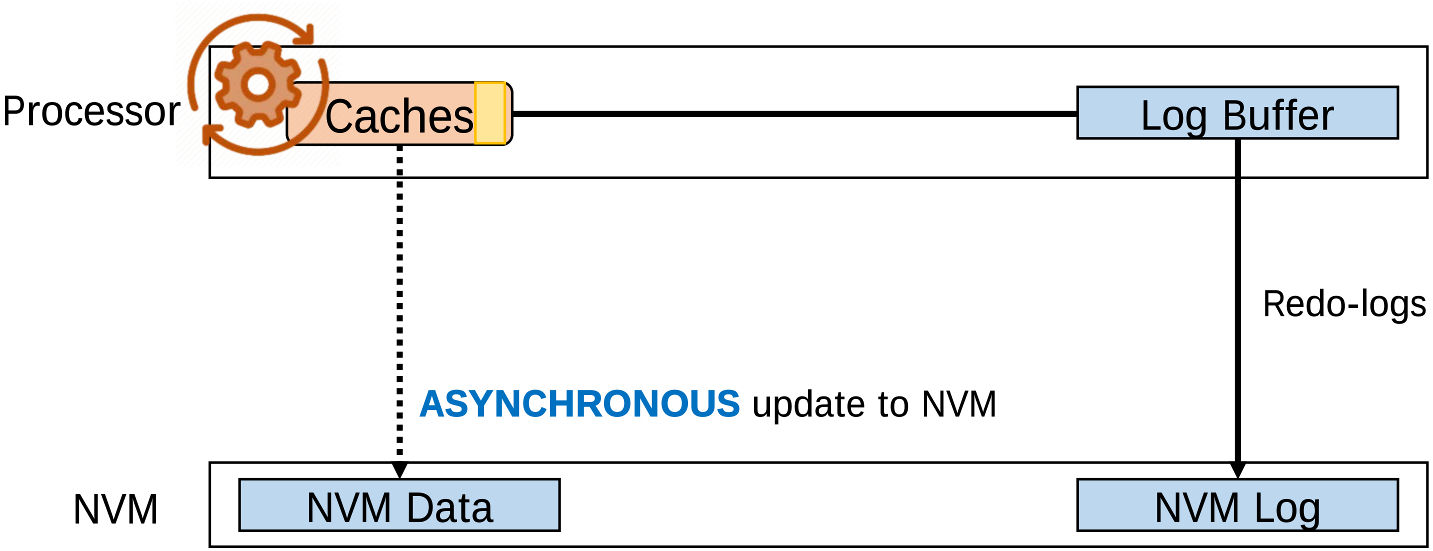
# Naïve Solution: On-chip Cache Extension

- Additional storages to store multiple write-sets
  - E.g., to store all physical address, scan the entire cache hierarchy
- Cache replacement policy to be aware of transactions
  - E.g., evict non-transactional cache blocks first
  - Has to discard the cache block if overflow
    ➔ Need to search log area for read access
    ➔ Need indirect data update



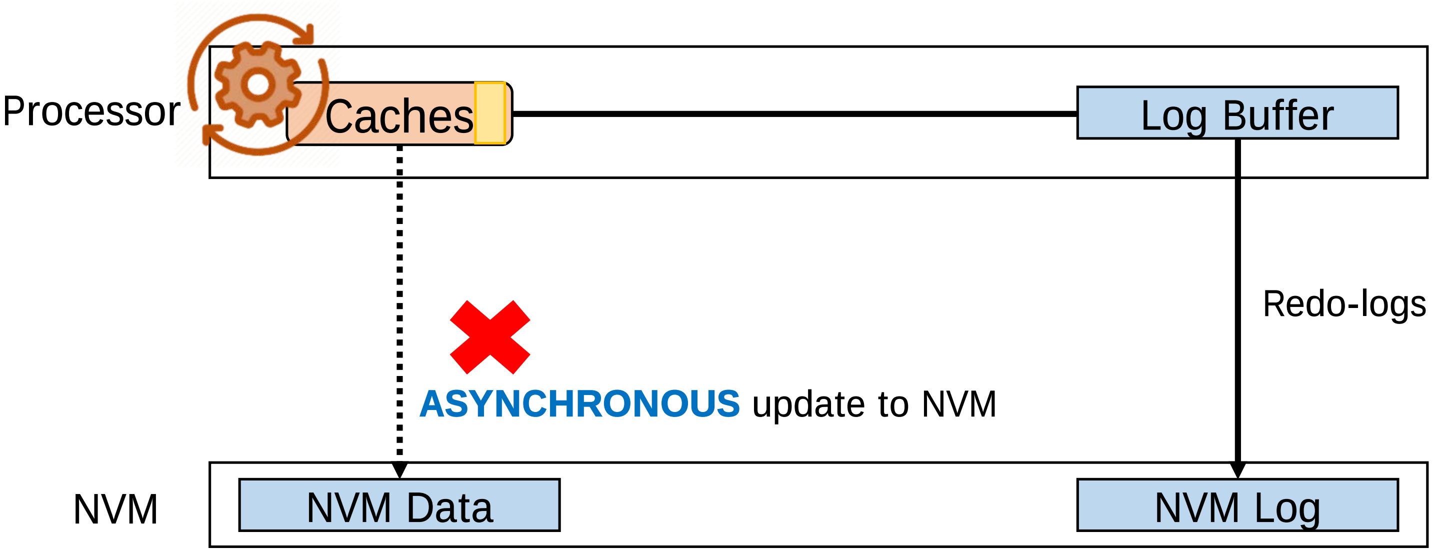
# Naïve Solution: On-chip Cache Extension

- Additional storages to store multiple write-sets
  - E.g., to store all physical address, scan the entire cache hierarchy
- Cache replacement policy to be aware of transactions
  - E.g., evict non-transactional cache blocks first
  - Has to discard the cache block if overflow
    ➔ Need to search log area for read access
    ➔ Need indirect data update

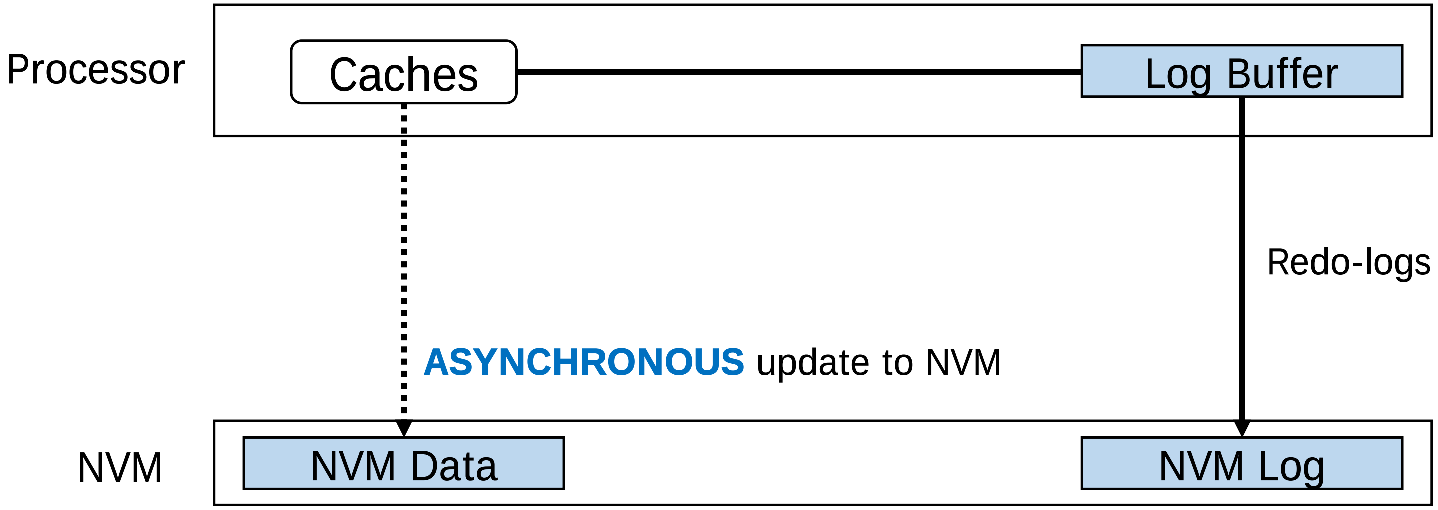


# Naïve Solution: On-chip Cache Extension

- Additional storages to store multiple write-sets
  - E.g., to store all physical address, scan the entire cache hierarchy
- Cache replacement policy to be aware of transactions
  - E.g., evict non-transactional cache blocks first
  - Has to discard the cache block if overflow
    ➔ Need to search log area for read access
    ➔ Need indirect data update

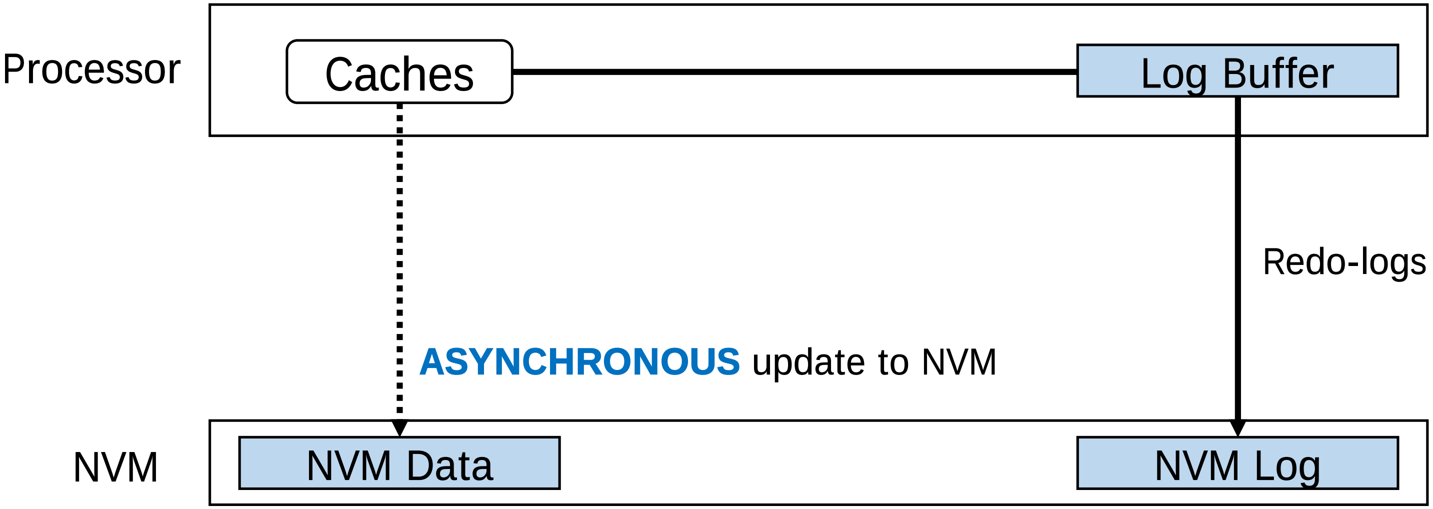


# **Re**do log with **D**irect **U**pdate (ReDU)



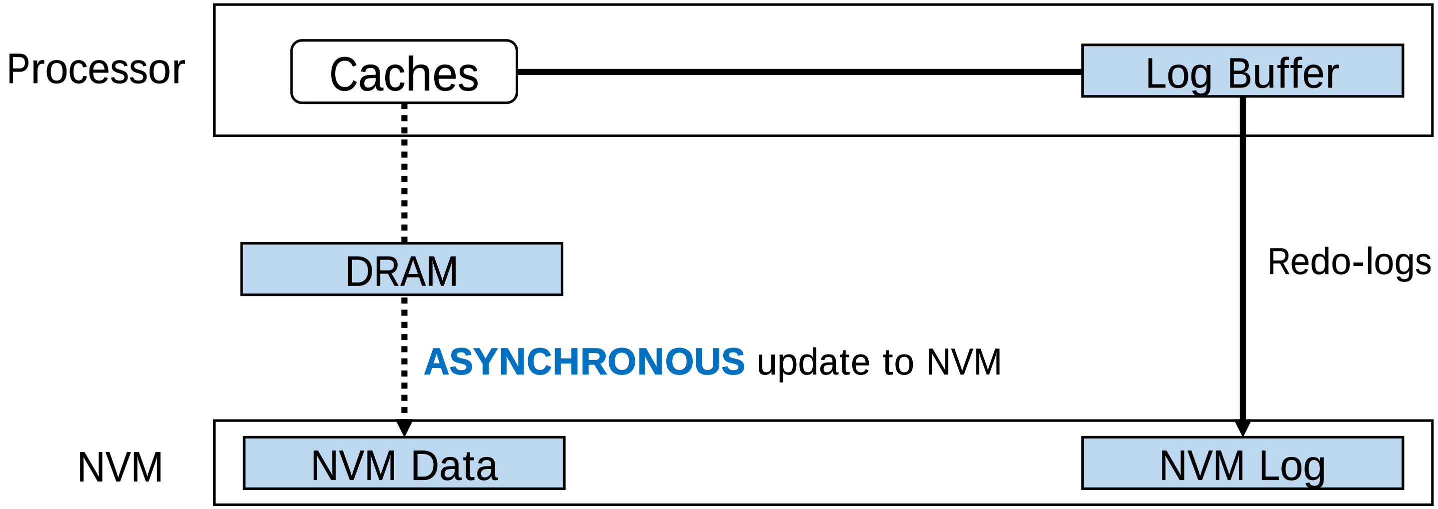
# **Re**do log with **D**irect **U**pdate (ReDU)

- Our approach: use DRAM for handling direct-update
  - Synchronous update to the FAST DRAM
  - Asynchronous update to the SLOW NVM



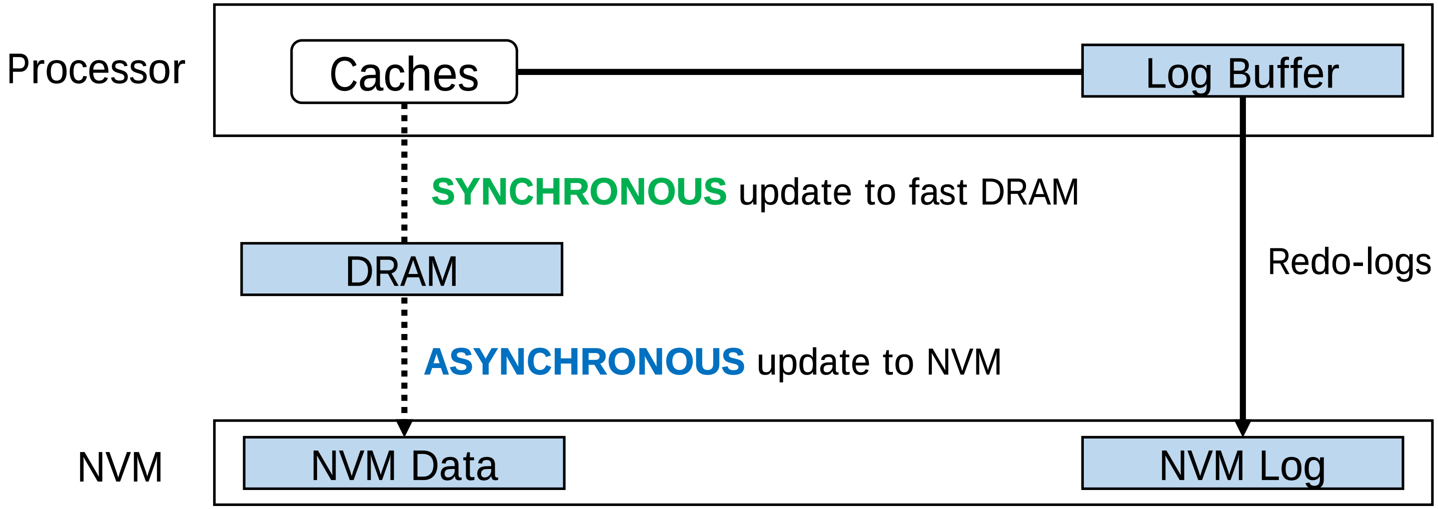
# **Re**do log with **D**irect **U**pdate (ReDU)

- Our approach: use DRAM for handling direct-update
  - Synchronous update to the FAST DRAM
  - Asynchronous update to the SLOW NVM



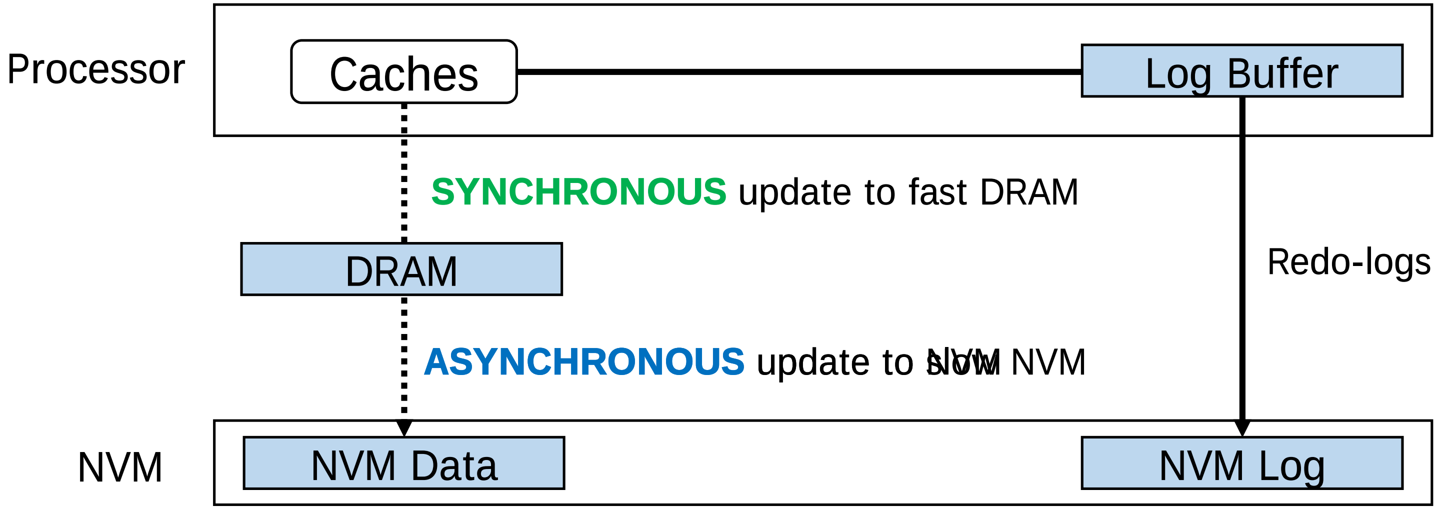
# **Re**do log with **D**irect **U**pdate (ReDU)

- Our approach: use DRAM for handling direct-update
  - Synchronous update to the FAST DRAM
  - Asynchronous update to the SLOW NVM

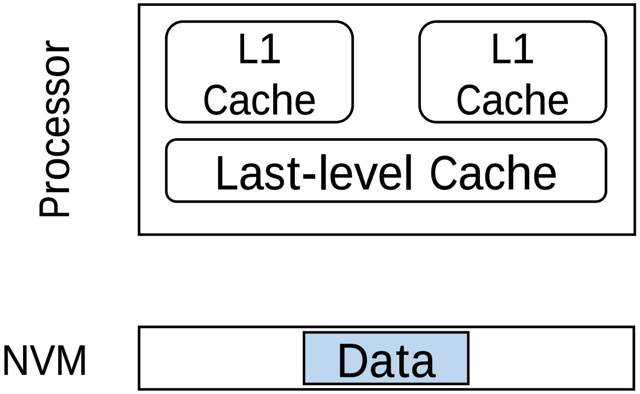


# **Re**do log with **D**irect **U**pdate (ReDU)

- Our approach: use DRAM for handling direct-update
  - Synchronous update to the FAST DRAM
  - Asynchronous update to the SLOW NVM


Processor
Caches
Log Buffer
SYNCHRONOUS update to fast DRAM
DRAM
Redo-logs
ASYNCHRONOUS update to slow NVM
NVM
NVM Data
NVM Log

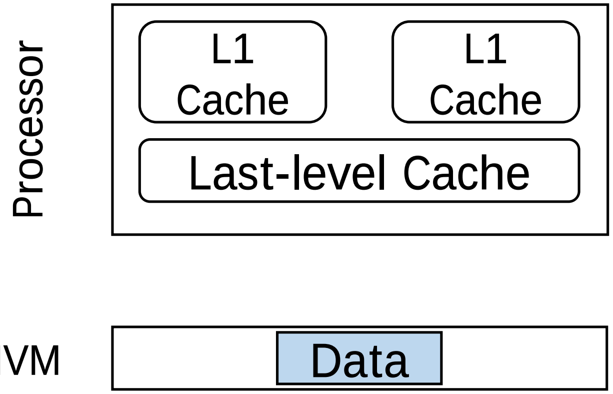
# ReDU – Direct-Update



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

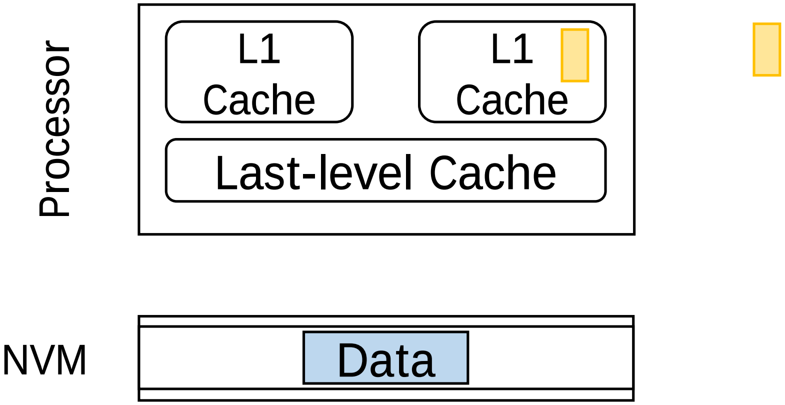
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

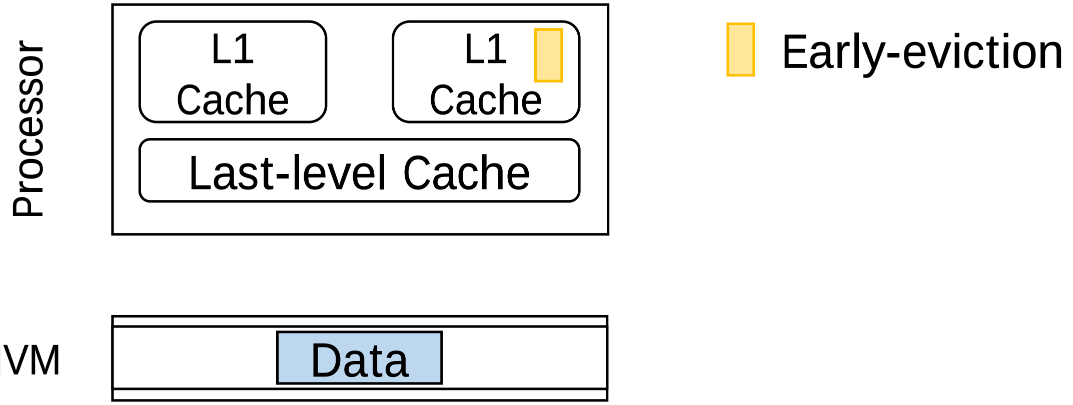
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

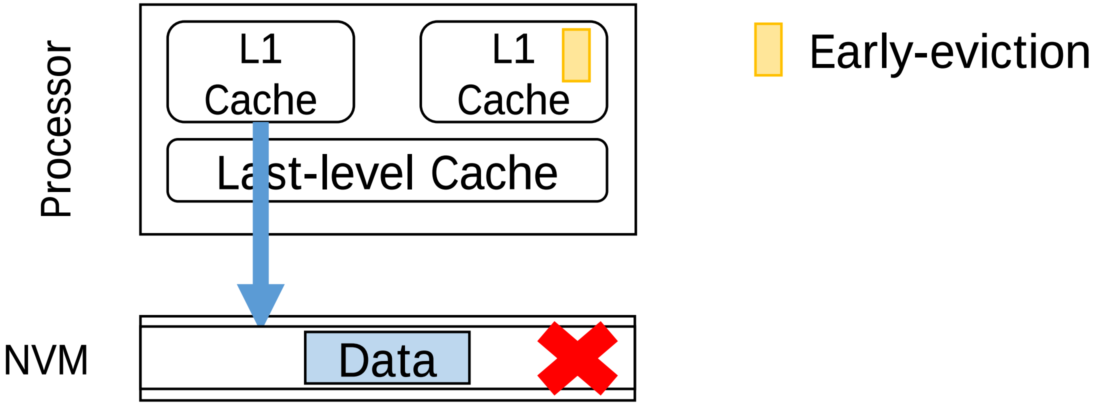
- DRAM cache stores:
  - "Early-evicted": modified cachelines evicted from L1 before commit
  - "On-commit-flushed": modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

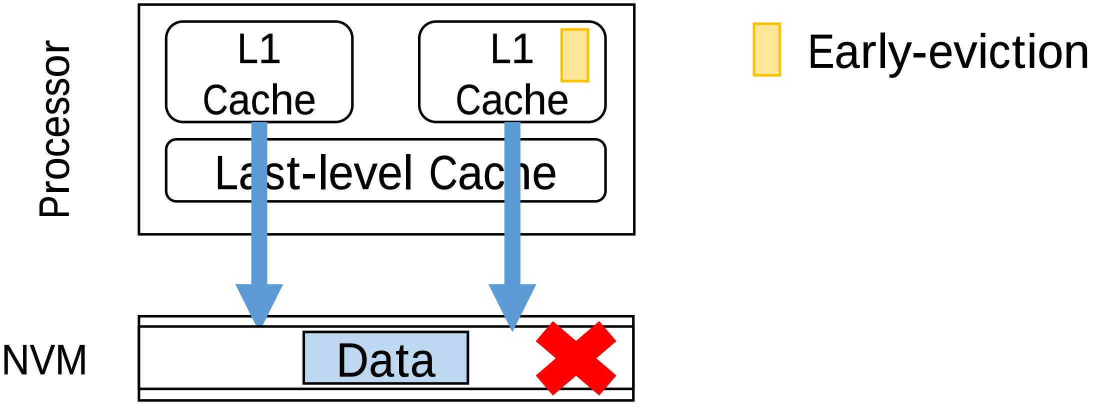
- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

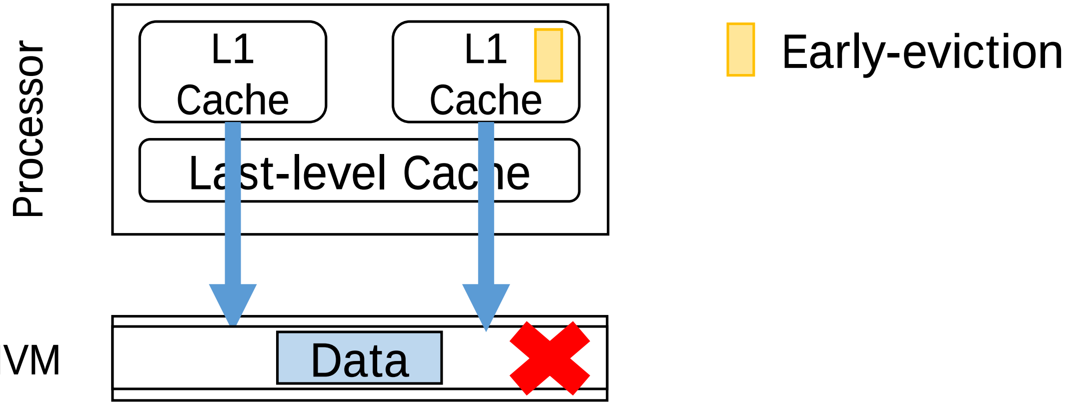
- Track the write-set within L1 cache
  - No on-chip cache modifications except L1
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

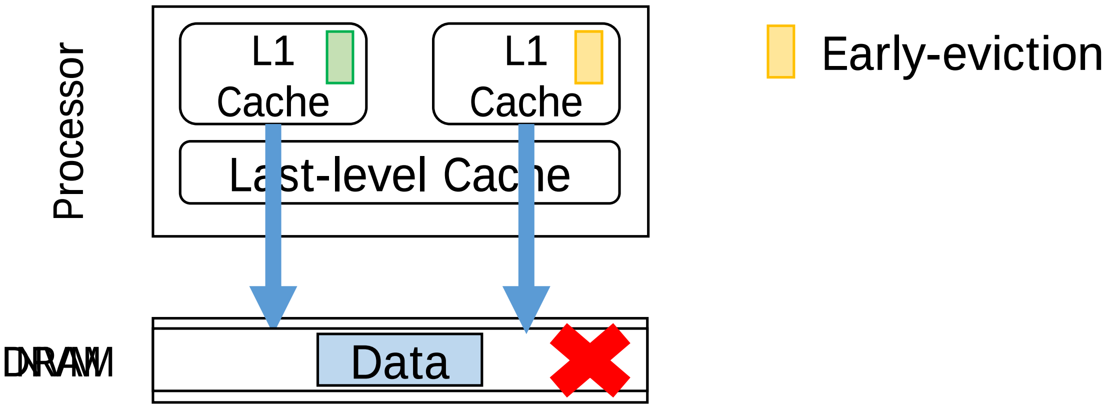
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache


Processor

L1 Cache

L1 Cache

Last-level Cache

DRAM

NVM

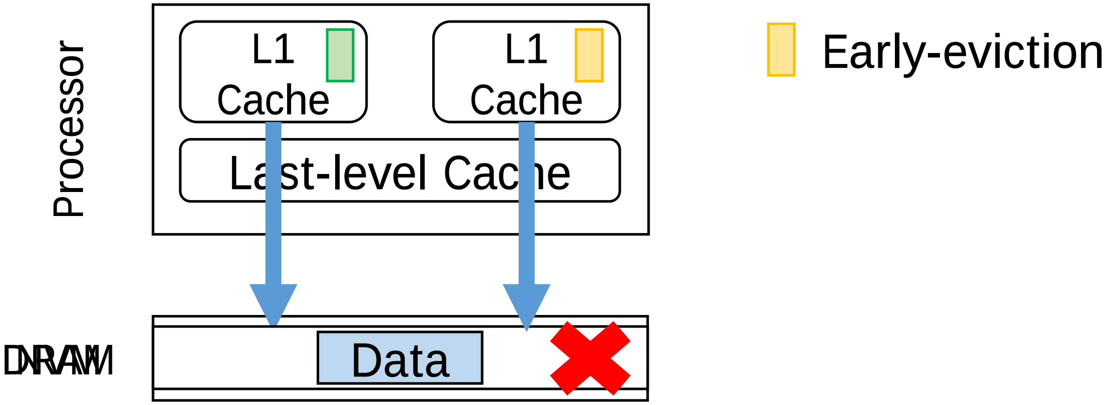
Data

🟨 Early-eviction

# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

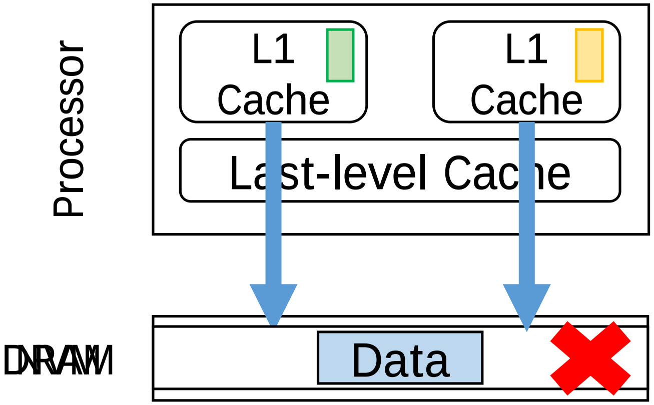
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 <u>before commit</u>
  - “On-commit-flushed”: modified cachelines in L1 <u>on commit</u>
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 <u>before commit</u>
  - “On-commit-flushed”: modified cachelines in L1 <u>on commit</u>
  - For both events, explicitly flush through the DRAM cache

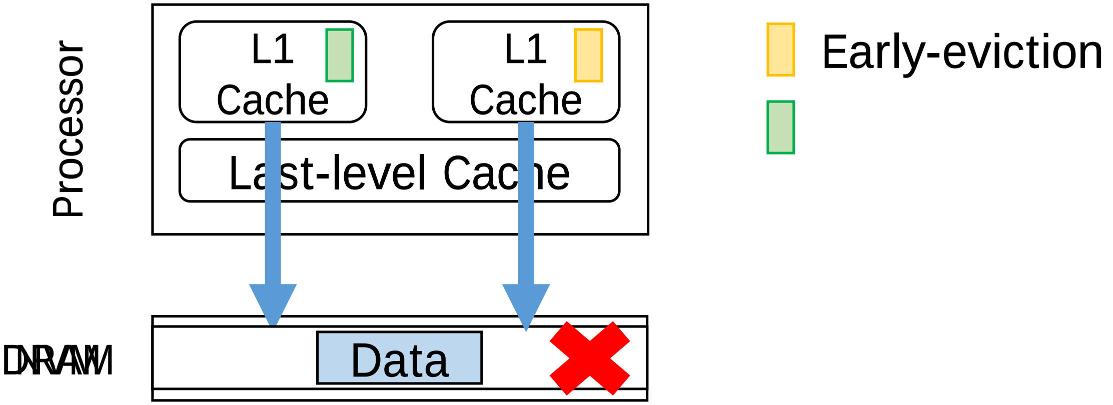


🟨 Early-eviction

# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

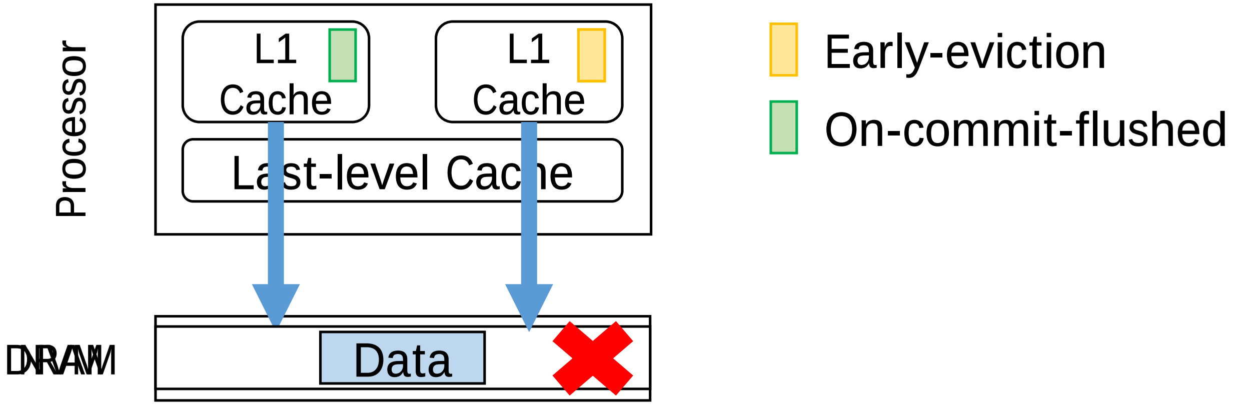
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 <u>before commit</u>
  - “On-commit-flushed”: modified cachelines in L1 <u>on commit</u>
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

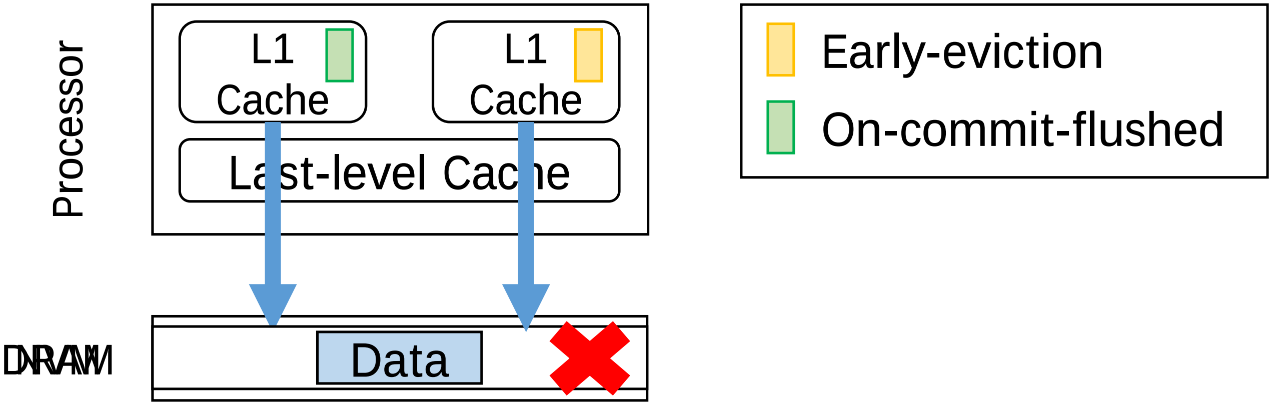
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 <u>before commit</u>
  - “On-commit-flushed”: modified cachelines in L1 <u>on commit</u>
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

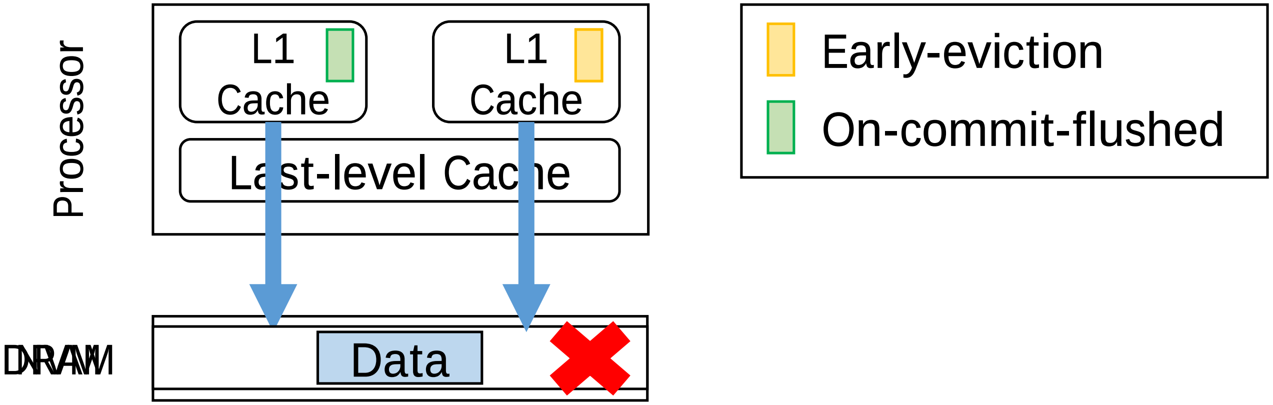
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

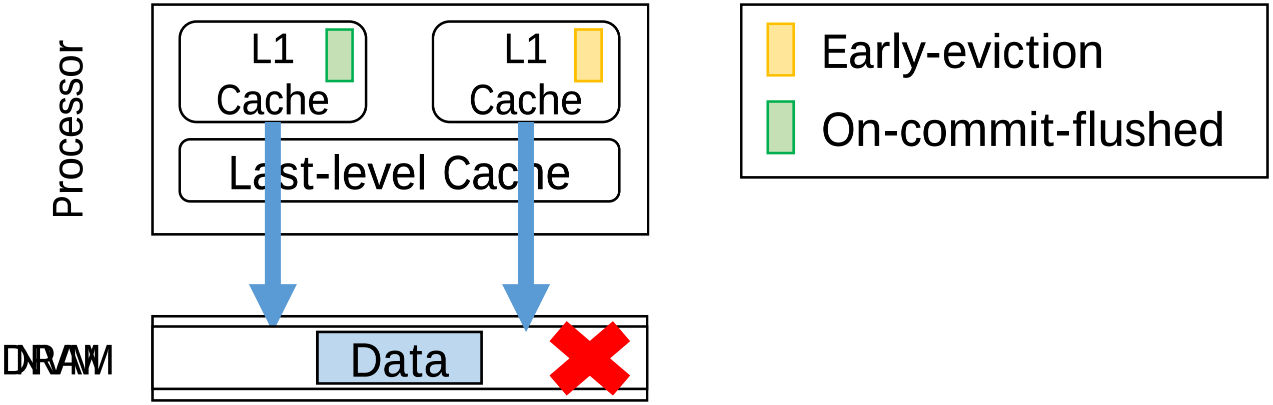
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 <u>before commit</u>
  - “On-commit-flushed”: modified cachelines in L1 <u>on commit</u>
  - For both events, explicitly flush through the DRAM cache



# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

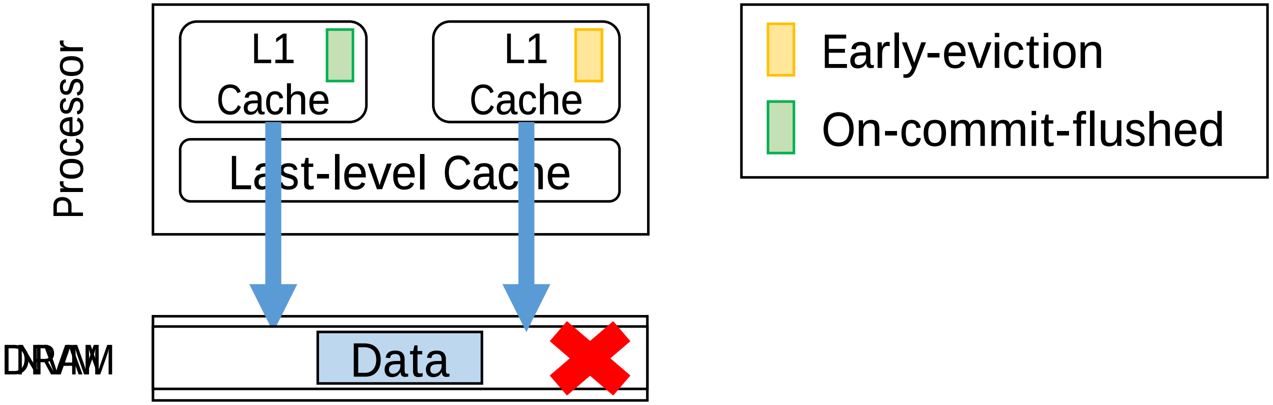
- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 before commit
  - “On-commit-flushed”: modified cachelines in L1 on commit
  - For both events, explicitly flush through the DRAM cache



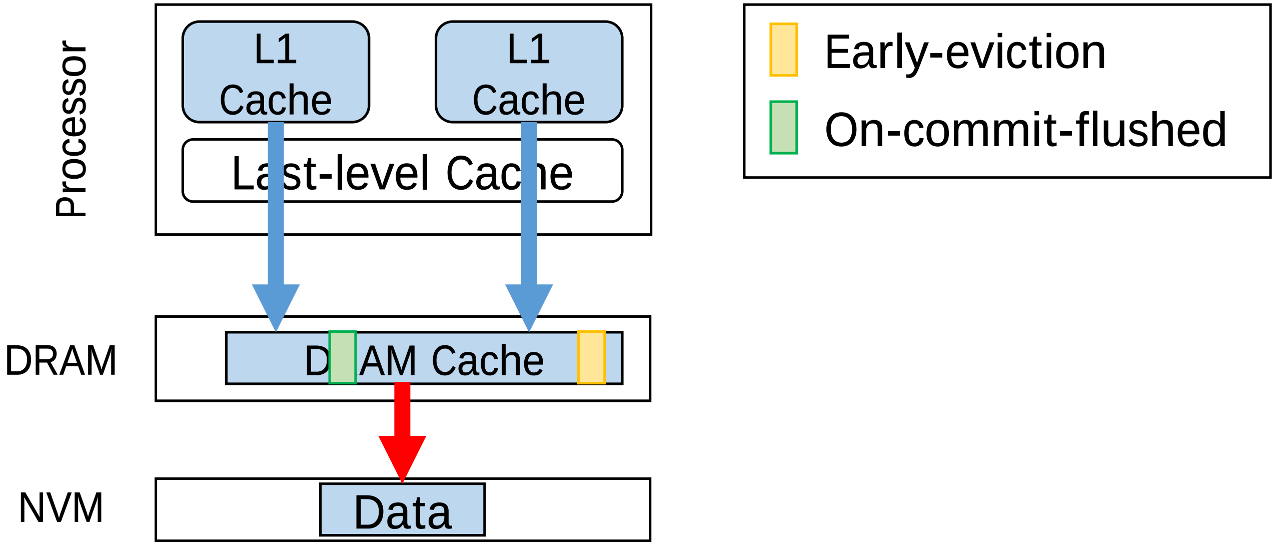
# ReDU – Direct-Update

- Track the write-set within L1 cache
  - No on-chip cache modifications except L1

- DRAM cache stores:
  - “Early-evicted”: modified cachelines evicted from L1 <u>before commit</u>
  - “On-commit-flushed”: modified cachelines in L1 <u>on commit</u>
  - For both events, explicitly flush through the DRAM cache

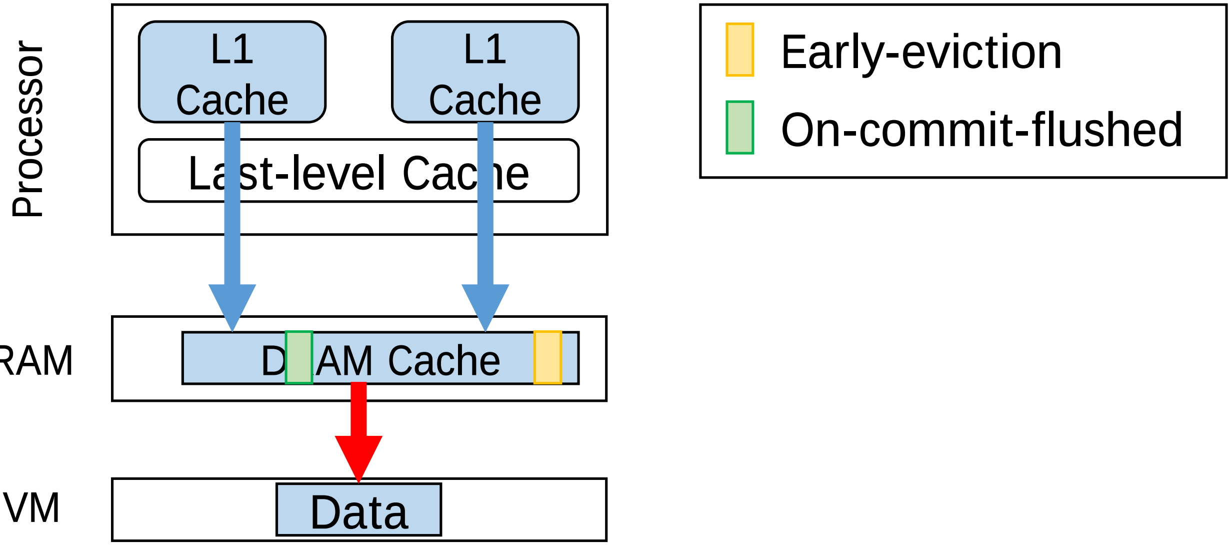


# ReDU – Direct-Update



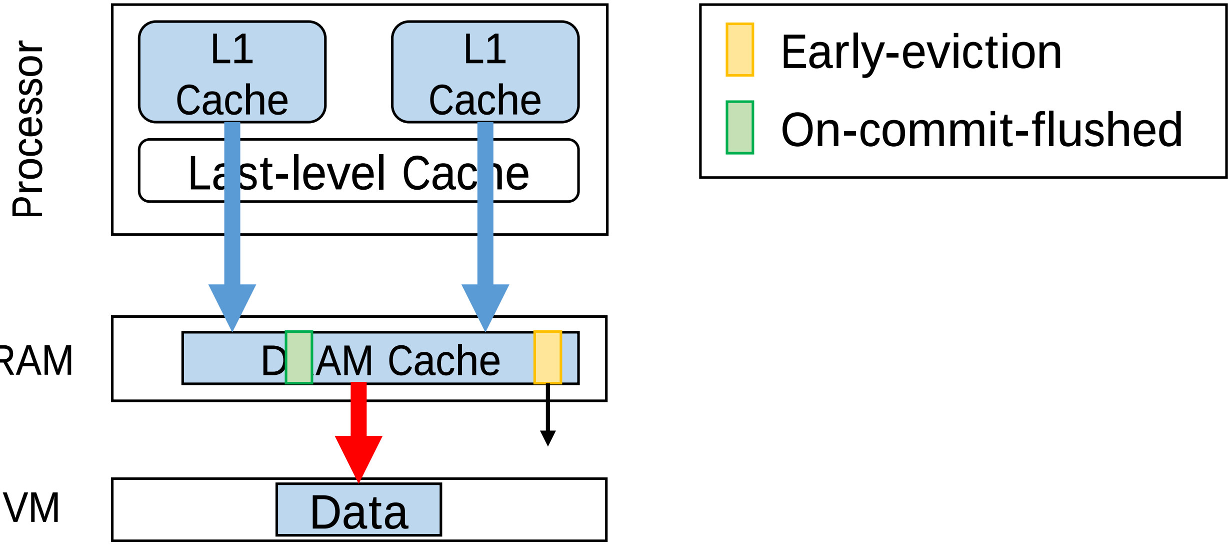
# ReDU – Direct-Update

- Update to NVM done asynchronously
- Only flush cachelines that belong to the committed transaction
  - DRAM cache maintains the committed transaction IDs
- Various write-back policies are possible
  - E.g., Eager or LRU



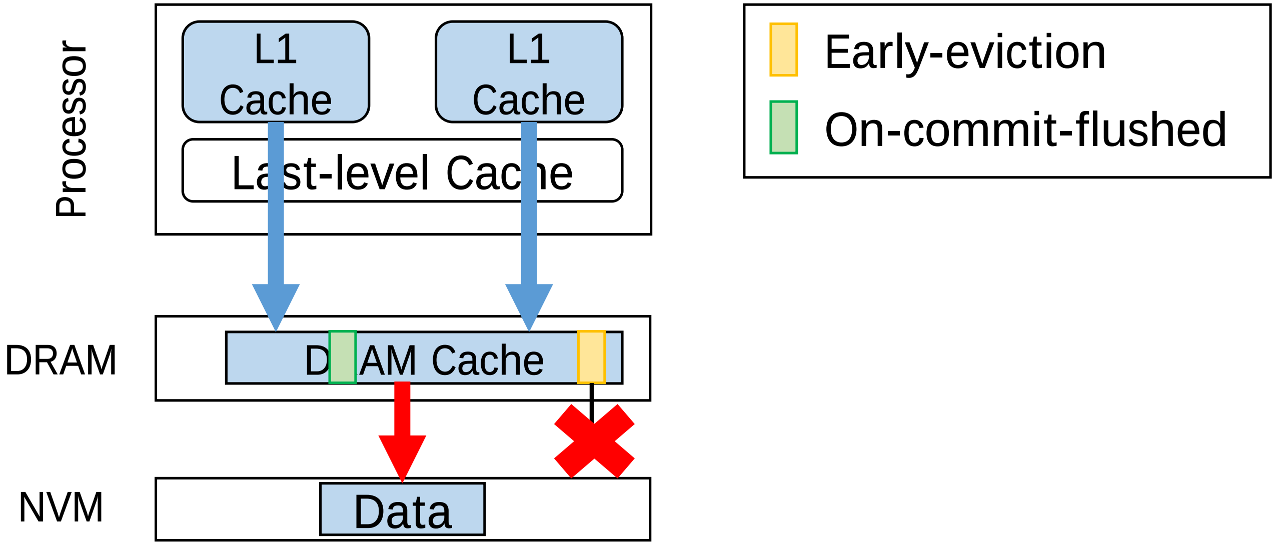
# ReDU – Direct-Update

- Update to NVM done asynchronously
- Only flush cachelines that belong to the committed transaction
  - DRAM cache maintains the committed transaction IDs
- Various write-back policies are possible
  - E.g., Eager or LRU



# ReDU – Direct-Update

- Update to NVM done asynchronously
- Only flush cachelines that belong to the committed transaction
  - DRAM cache maintains the committed transaction IDs
- Various write-back policies are possible
  - E.g., Eager or LRU



# More in the paper...

- Full design space exploration of HW logging
  - Log optimization #1: coalescing
  - Log optimization #2: packing
- Details of DRAM cache organization
  - Transaction Table and Offset Table
- Bloom filter-based HW-filter to reduce DRAM accesses
- Evaluation of LRU write-back policy of the DRAM cache
- Log management
- ...

# Methodology

- Gem5 simulator

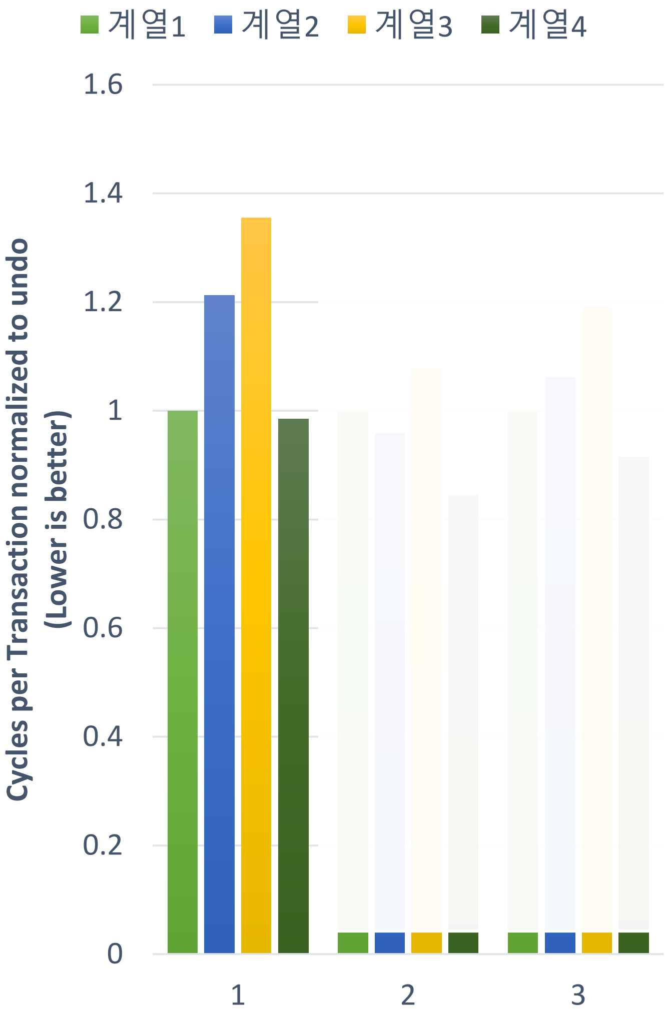
| | |
|---|---|
| Processor | OoO, 2GHz, x86 |
| L1 I/D cache | Private, 32KB, 8-way |
| L2 cache | Private, 256KB, 8-way |
| L3 cache | Shared, 8MB, 16-way |
| DRAM cache | 40MB (8MB meta + 32MB data) |
| NVM | Read: 50ns, write: 150ns |

- Benchmarks

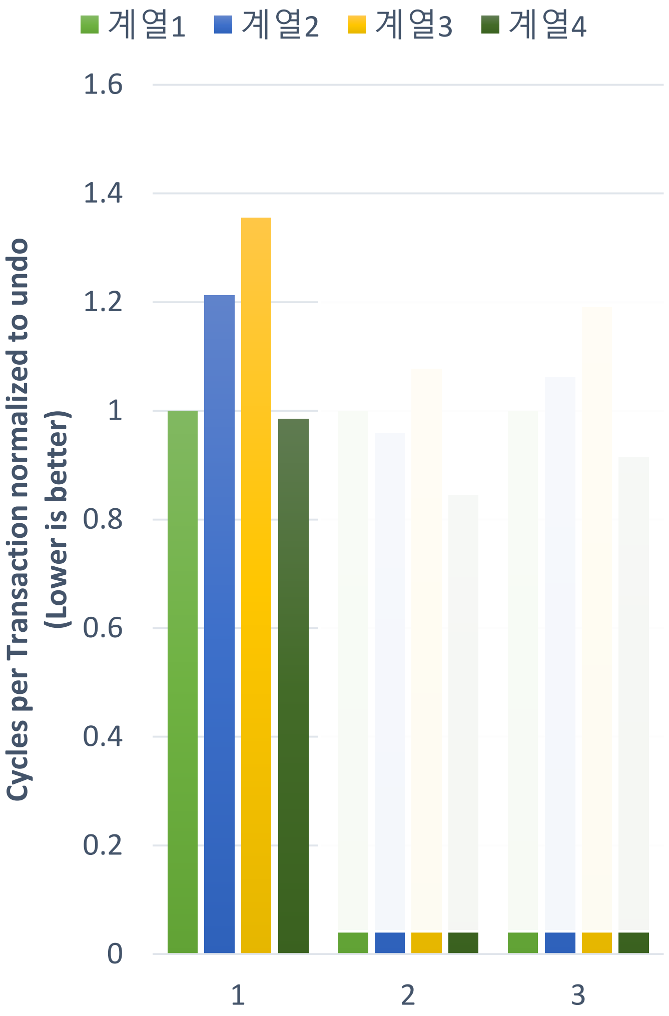
| | |
|---|---|
| Micro-bench | Vector, Swap |
| NVML | HashMap, B-Tree, RB-Tree |
| Macro-bench | YCSB, TPCC, ECHO |

- Comparing schemes
  - All equally include log optimizations (e.g., coalescing and packing)
  - UndoSync: undo log with synchronous commit
  - RedoIndirect: redo log with asynchronous but indirect update
  - Undo+Redo: undo+redo log with asynchronous & direct update
  - ReDU: our approach

# Evaluation – Transaction Throughput

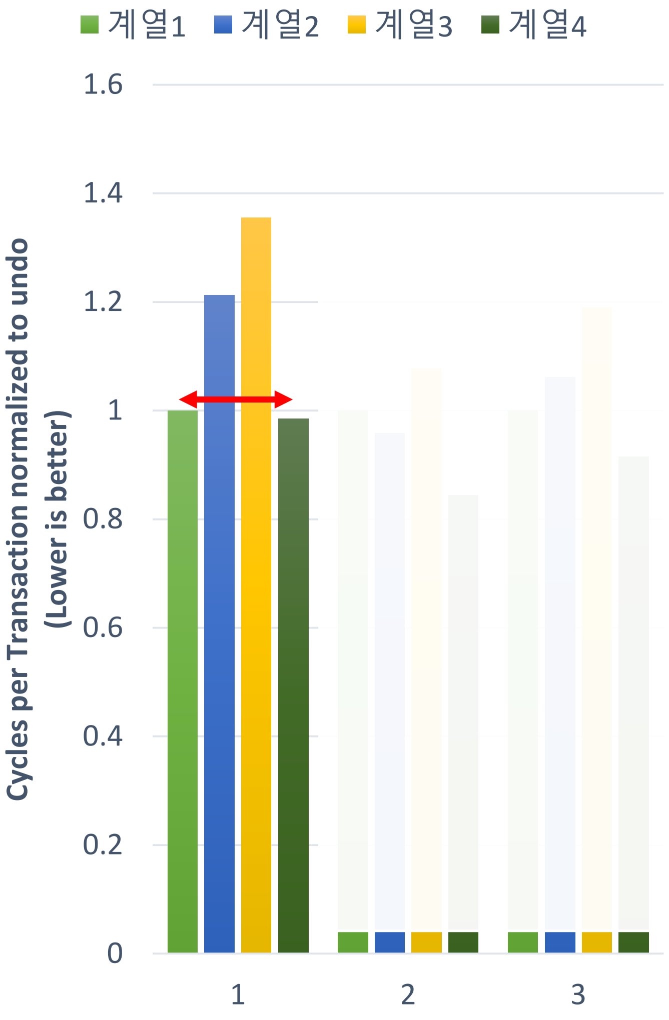


# Evaluation – Transaction Throughput



- Large & sequential workloads
  - Undo and ReDU perform similarly
    (same data path and NVM bandwidth saturated)
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs
- Small & Random workloads
- On average

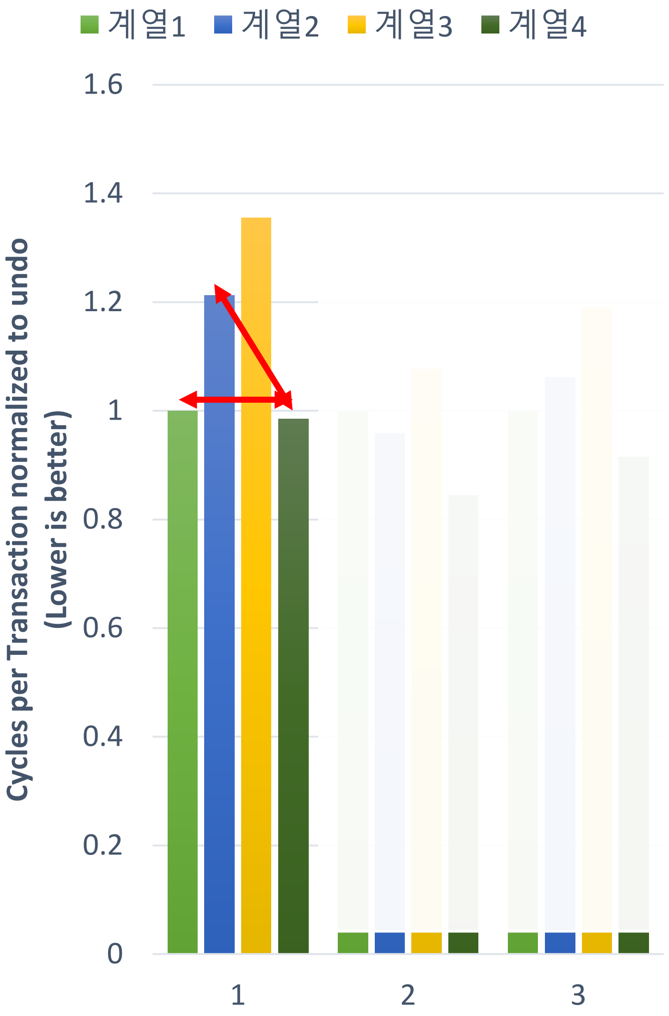
# Evaluation – Transaction Throughput



- Large & sequential workloads
  - Undo and ReDU perform similarly
    (same data path and NVM bandwidth saturated)
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs

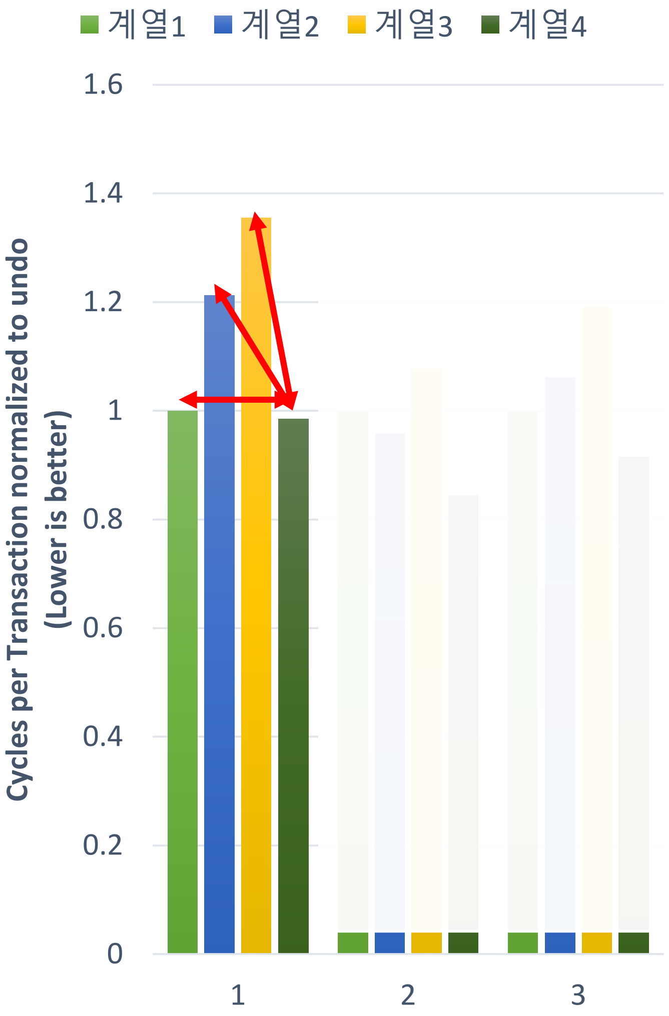
- Small & Random workloads
- On average

# Evaluation – Transaction Throughput



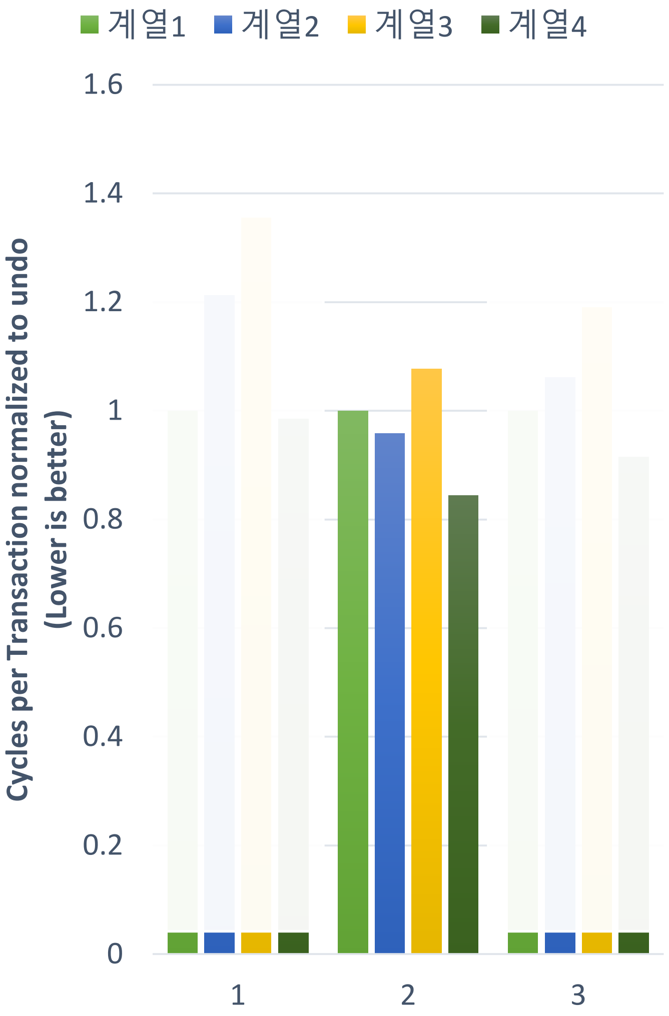
- Large & sequential workloads
  - Undo and ReDU perform similarly (same data path and NVM bandwidth saturated)
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs
- Small & Random workloads
- On average

# Evaluation – Transaction Throughput

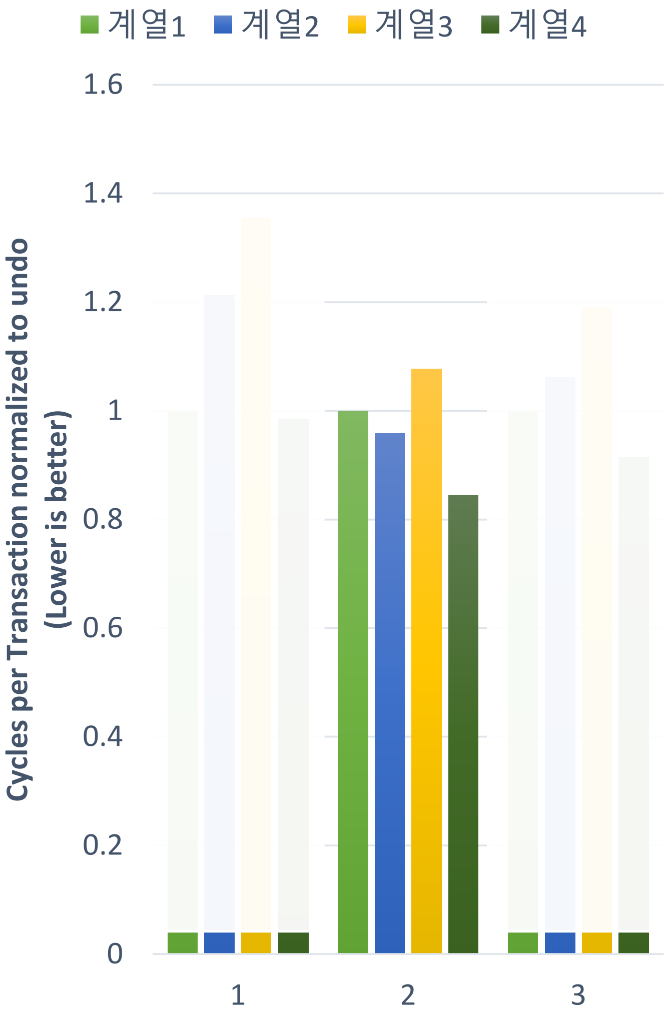


- Large & sequential workloads
  - Undo and ReDU perform similarly (same data path and NVM bandwidth saturated)
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs
- Small & Random workloads
- On average

# Evaluation – Transaction Throughput

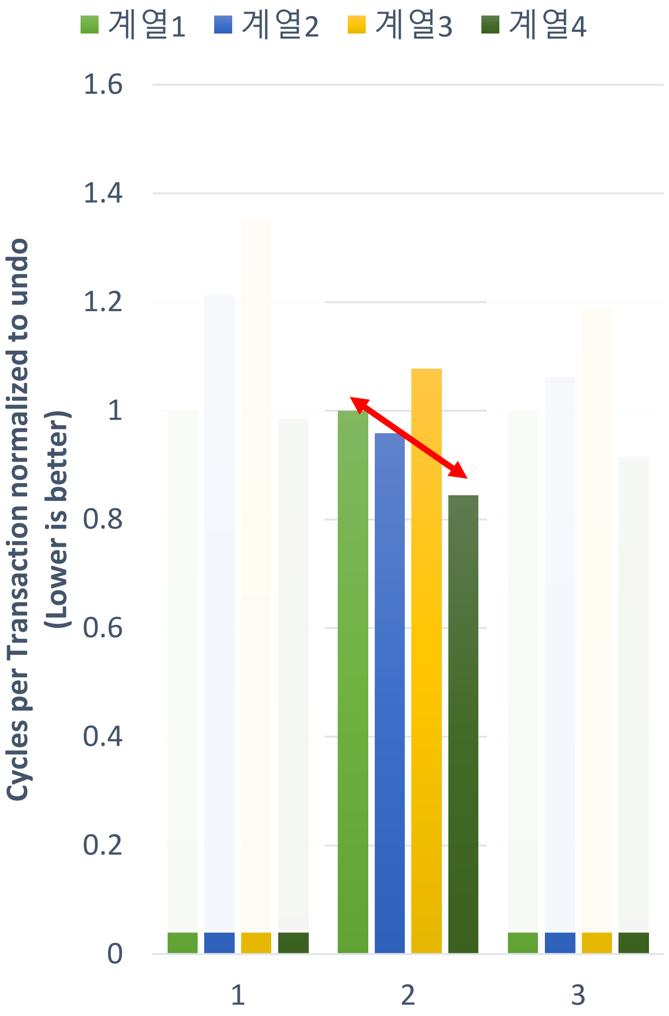


# Evaluation – Transaction Throughput



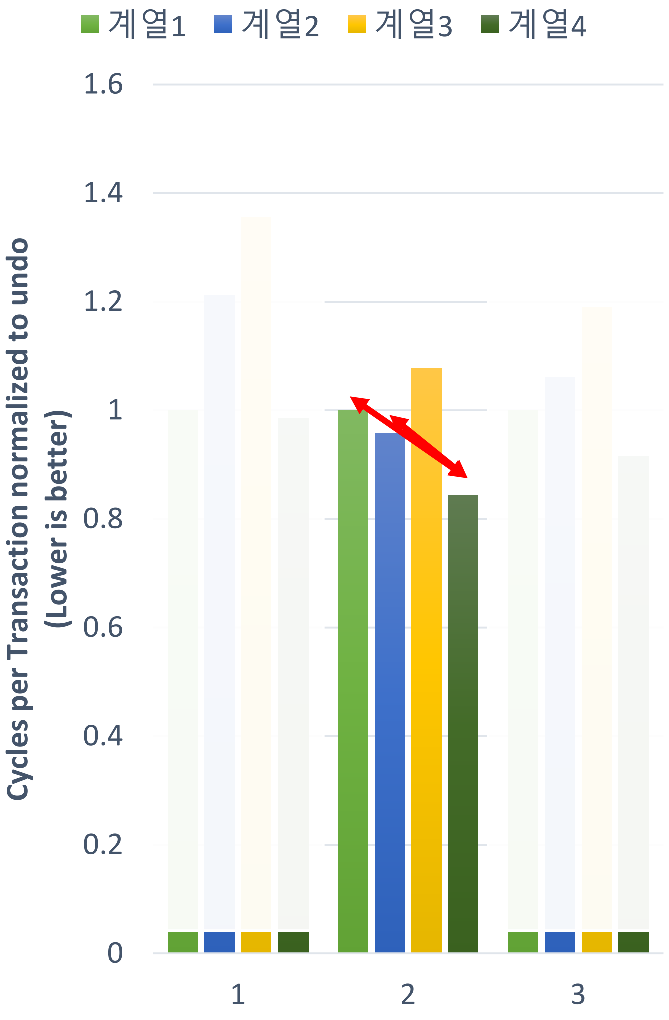
- Large & Sequential workloads
- Small & Random workloads
  - Undo waits synchronous commit
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs
- On average

# Evaluation – Transaction Throughput



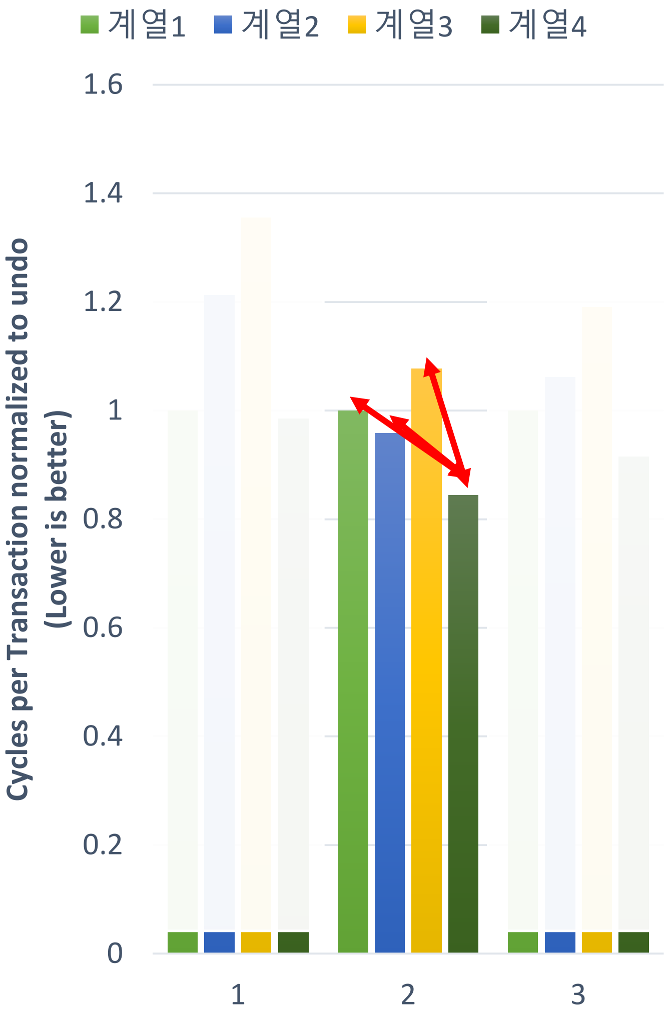
- Large & Sequential workloads
- Small & Random workloads
  - Undo waits synchronous commit
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs
- On average

# Evaluation – Transaction Throughput



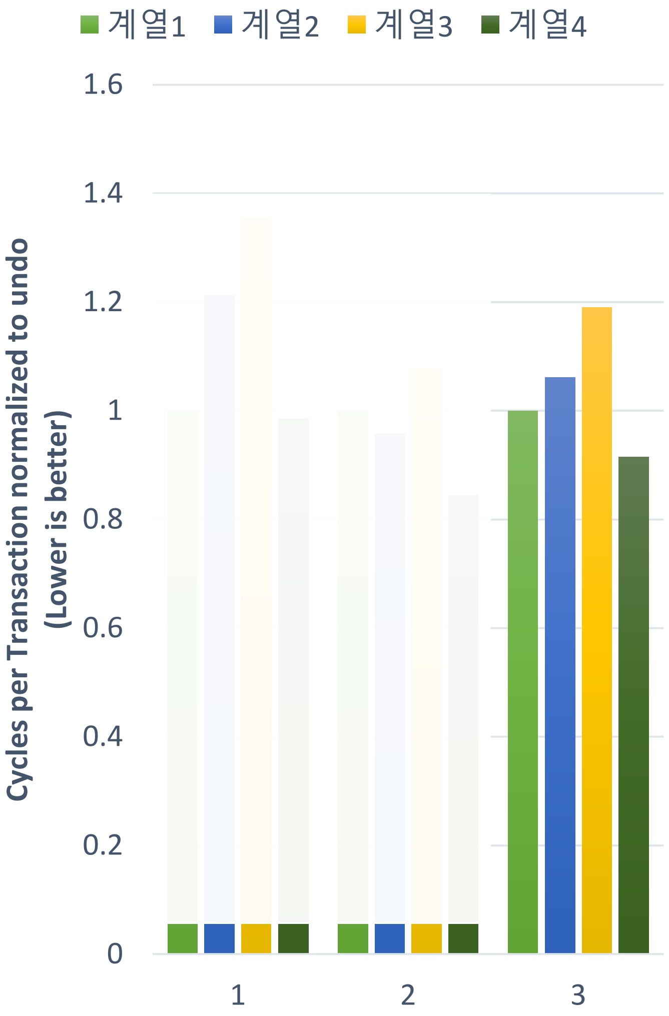
- Large & Sequential workloads
- Small & Random workloads
  - Undo waits synchronous commit
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs
- On average

# Evaluation – Transaction Throughput



- Large & Sequential workloads
- Small & Random workloads
  - Undo waits synchronous commit
  - Redo suffers from indirect update
  - UndoRedo requires double NVM writes for logs
- On average

# Evaluation – Transaction Throughput



- Large & Sequential workloads
- Small & Random workloads
- On average
  - Asynchronous update ➔ 9%
  - Direct update ➔ 16%
  - Small log size ➔ 30%

# Summary

- Problem: crash-consistency in storage-class memory
  - Atomicity and durability support for NVM writes
  - Existing hardware solutions exhibit trade-offs

- Solution: Redo log with Direct Updates (ReDU)
  - Redo-based log with optimizations
  - Synchronous update to the fast DRAM
  - Asynchronous update to the slow NVM

- Results: ReDU outperforms existing solutions in various workloads
  - Bringing DRAM into the atomicity and durability

# Efficient Hardware-assisted Logging with Asynchronous and Direct Update for Persistent Memory

**Jungi Jeong**, Chang Hyun Park,
Jaehyuk Huh, and Seungryoul Maeng

KAIST School of Computing

*International Symposium on Microarchitecture (MICRO) 2018*