INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2021/CHICAGO/USA
acm In-Cooperation
sighpc
AUGUST 9-12, 2021

CCAE: Crash-Consistency-Aware Encryption for Non-Volatile Memories

Mengya Lei, Fang Wang, Dan Feng, Fan Li, Xueliang Wei

Huazhong University of Science & Technology

Outline

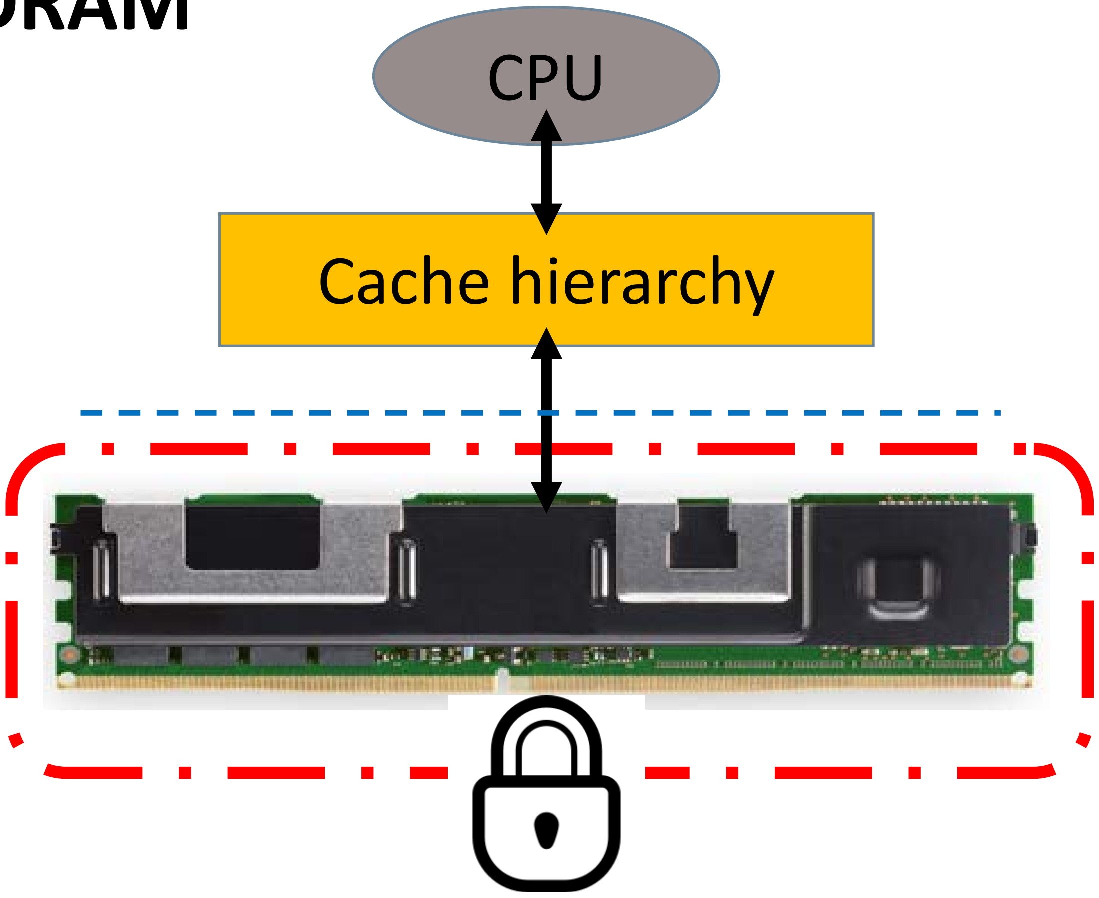
- 1 Background
- 2 CCAE
- 3 Experiment
- 4 Conclusion

Outline

- **1 Background**
- 2 CCAE
- 3 Experiment
- 4 Conclusion

Background : Non-Volatile Memory (NVM)

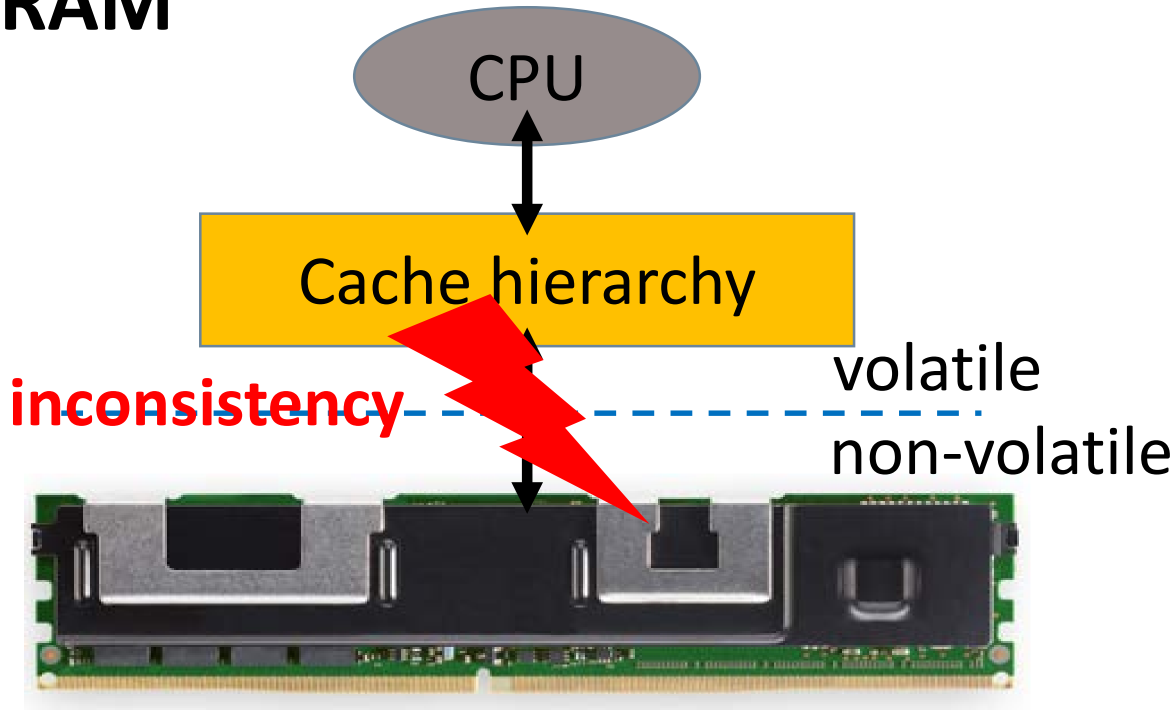
- NVM is promising alternative of DRAM
  - ✓ non-volatility
  - ✓ large capacity
  - ✓ fast speed + low power
  - ● limited write endurance
  - ● expensive write operations



- Challenges caused by persistency
  - ☹ data security

Background : Non-Volatile Memory (NVM)

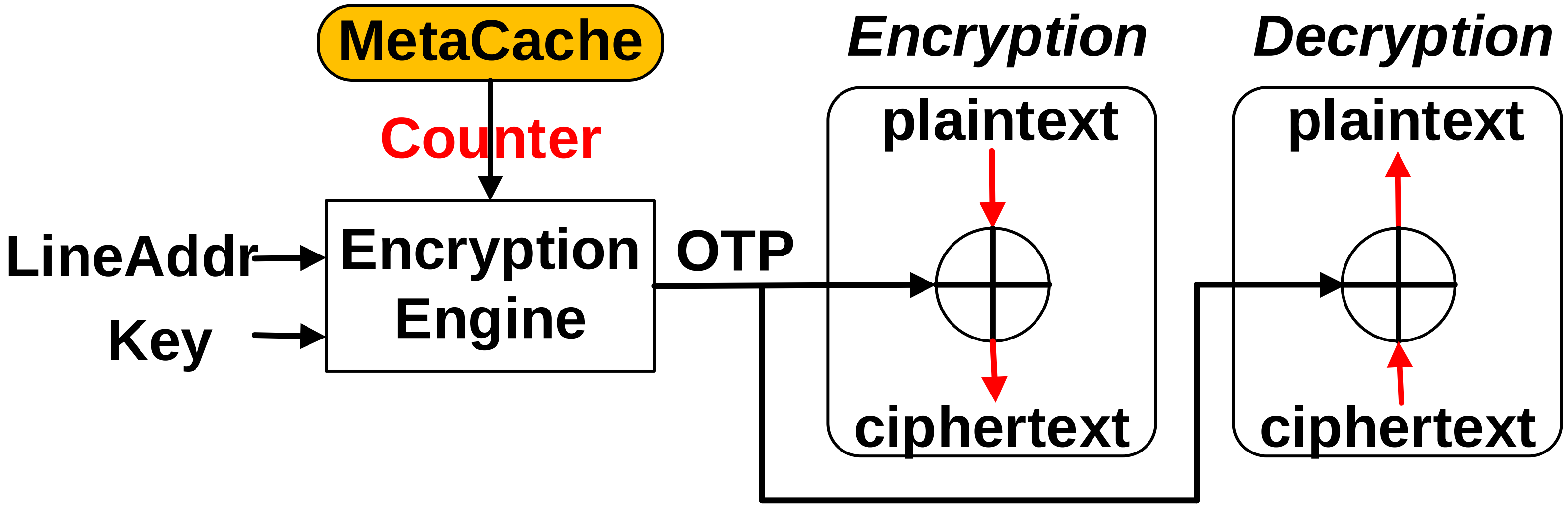
- **NVM is promising alternative of DRAM**
  - ✓ non-volatility
  - ✓ large capacity
  - ✓ fast speed + low power
  - ● limited write endurance
  - ● expensive write operations



- **Challenges caused by persistency**
  - ☹ data security
  - ☹ crash consistency

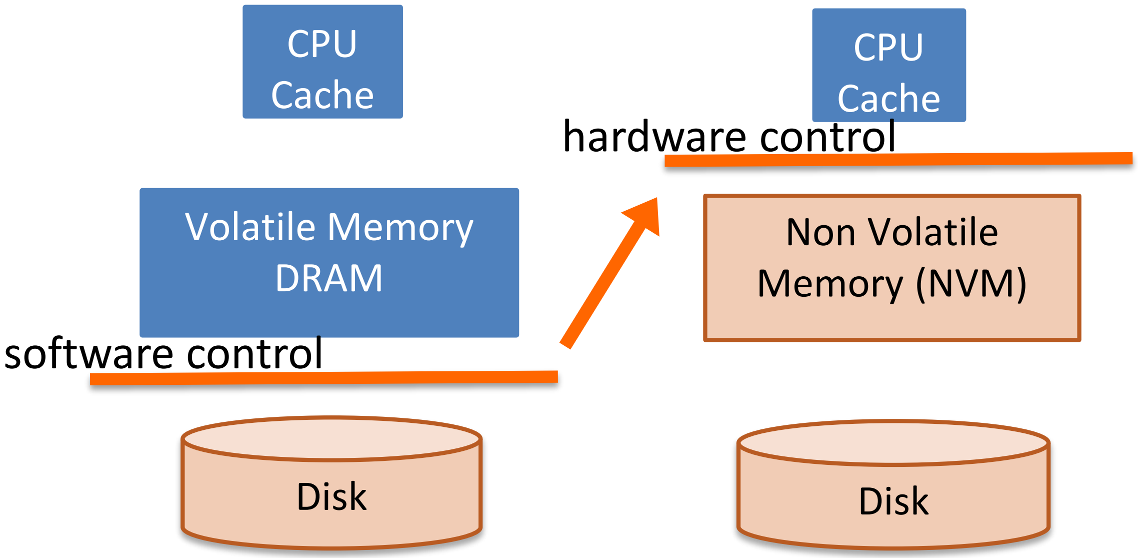
Background : Counter Mode Encryption (CME)

- **Data Security : Counter mode encryption (CME)**
  - ✓ hide the decryption latency
  - ✓ OTP never reuse (temporal & spatial uniqueness)
  - ✓ each data block (64B) has a counter
  - ✓ counter+1 each time the data is encrypted

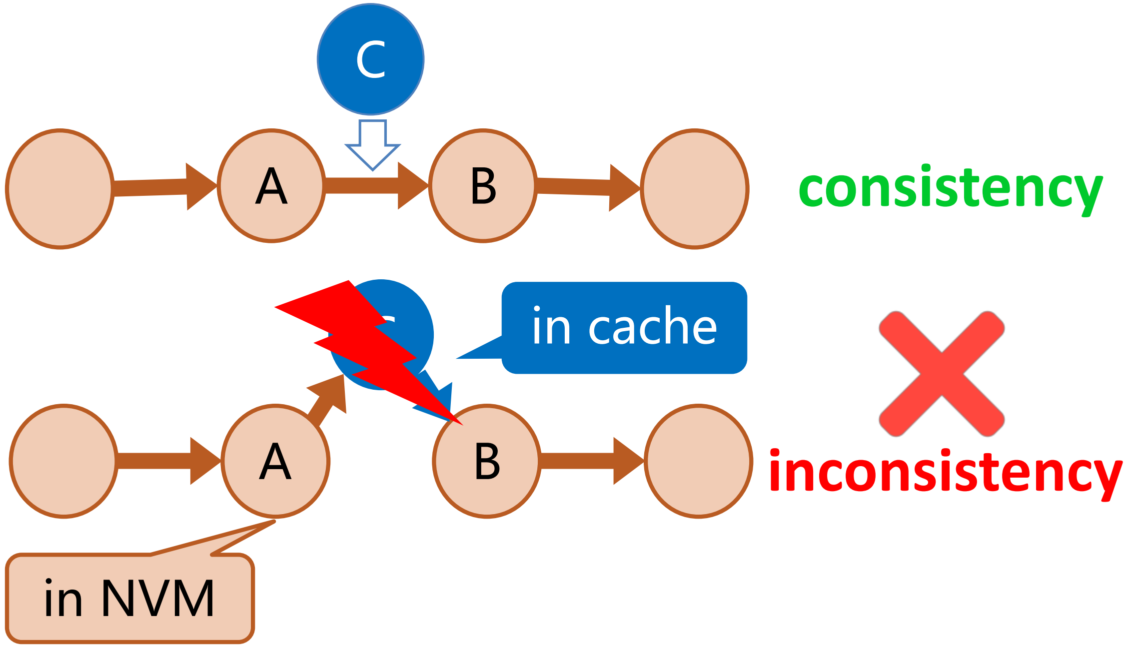


Background : Data Crash Consistency in NVM

- Volatile--non volatile gap between CPU & memory
- Inconsistency caused by partial update and out-of-order execution



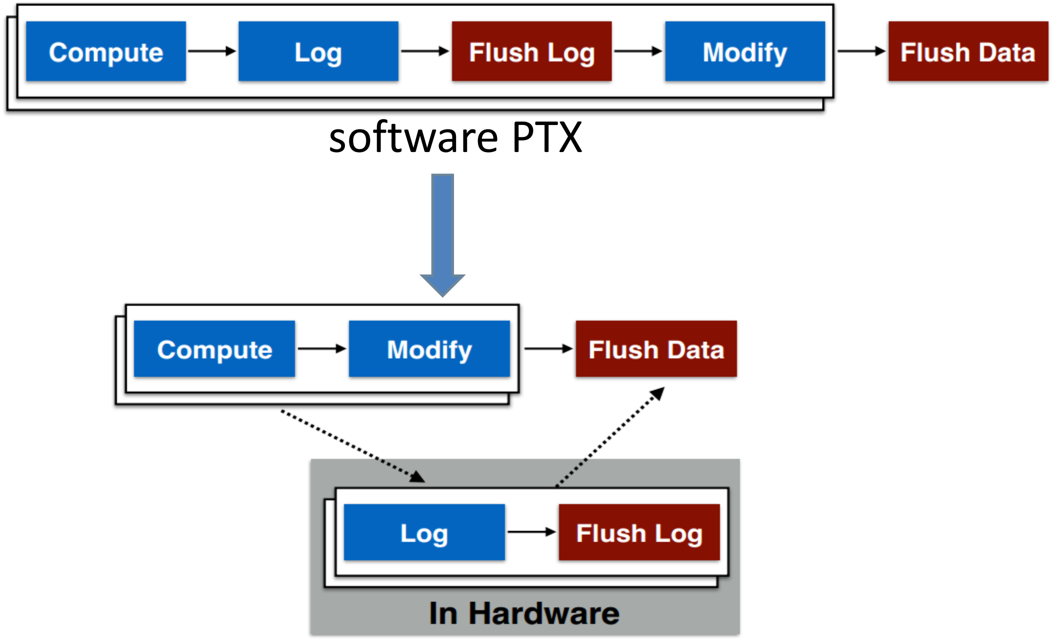
volatile--non volatile gap



Example: add a new node to a persistent linked list

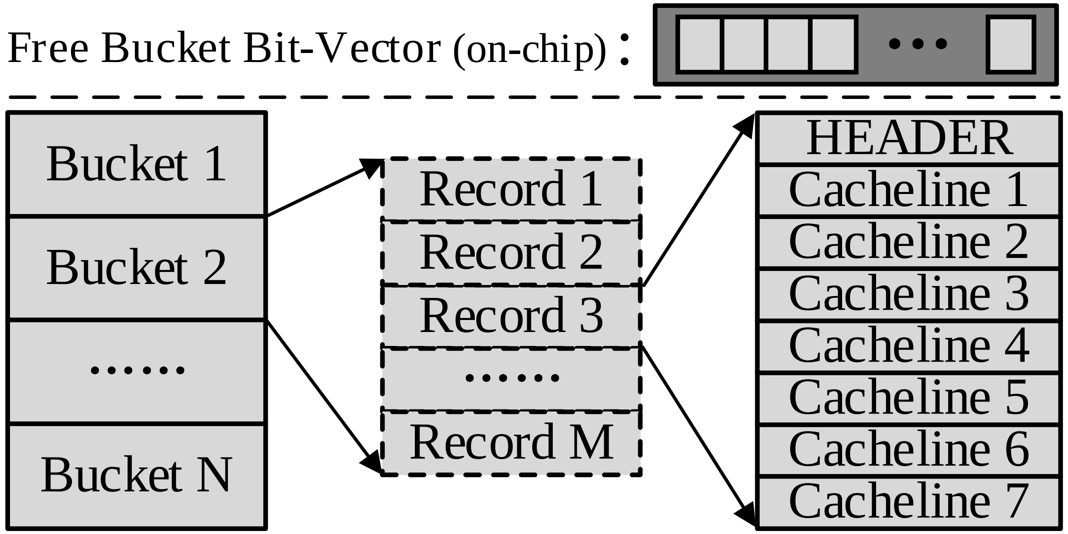
Background : Data Crash Consistency in NVM

- **Hardware persistent transaction： efficient order control + log management**
- **Example: ATOM [hpca17]: allocate log in Bucket granularity**



software PTX

hardware PTX



ATOM: Log Organization in NVM

Outline

- 1 Background
- **2 CCAE**
- 3 Experiment
- 4 Conclusion

New Problem: Counter Crash Consistency

- Consistency + Security → counter crash consistency
- Volatile Counter Cache → partial persisted
- Correct recovery

$$Ciphertext = OTP \oplus Plaintext, \quad OTP = En(key, addr|counter)$$

(a) Data Counter Time $\Rightarrow Plaintext_{new} \neq OTP_{stale} \oplus Ciphertext_{new}$

inconsistency

(b) Counter Data Time $\Rightarrow Plaintext_{new} \neq OTP_{new} \oplus Ciphertext_{stale}$

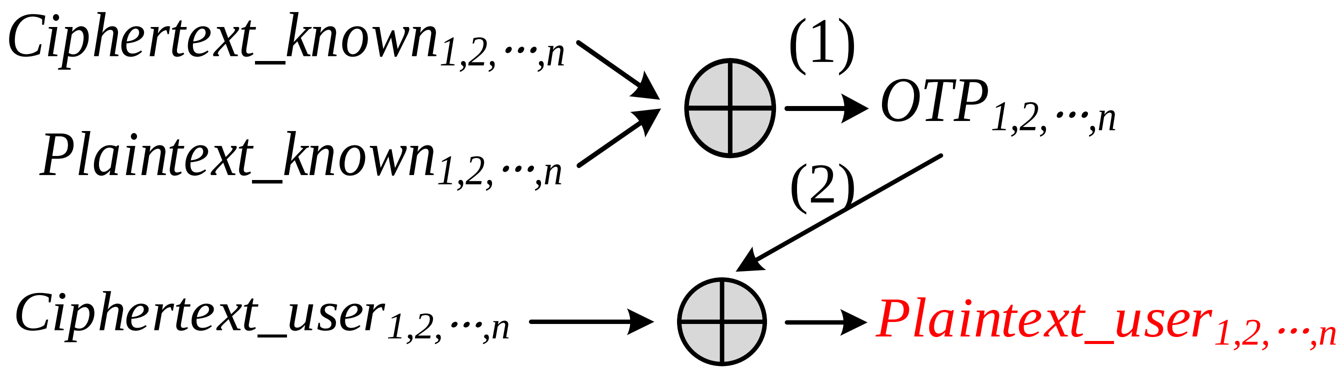
inconsistency

(c) Counter Data Time $\Rightarrow Plaintext_{new} = OTP_{new} \oplus Ciphertext_{new}$

consistency

New Problem: Challenges

- **Can't use data crash consistency mechanism**
  - new security requirement: **OTP can not reuse**



The known-plaintext attack caused by OTP reuse

- **Expensive NVM writes** (each data has a counter)

Existing Solutions & Our goals

➢ **Exiting Solutions fail to efficiently guarantee counter crash consistency**

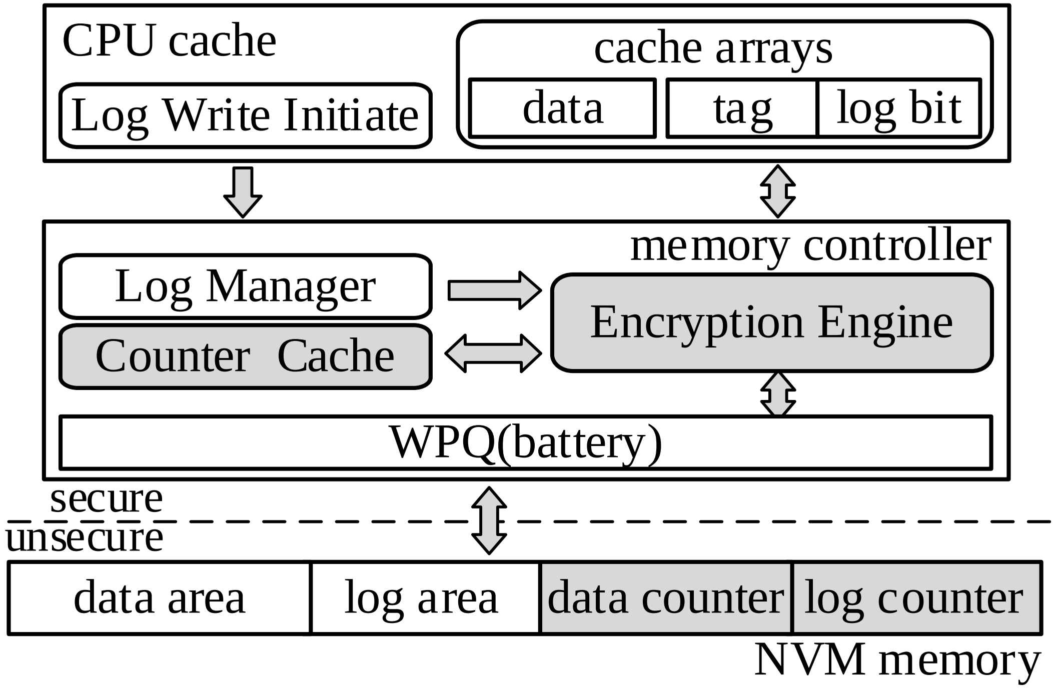
- lack of integration of data characteristics
- OTP reuse or high recovery time or long execution time

	Security	**Scalability**	**Recovery time**	**Effectiveness**	**Aware of data features**
Counter-Atomic [Ye et al., HPCA18]	×	×	√	√	√/×
Osiris [Ye et al., MICRO18]	√	√/×	√/×	√	×
SuperMem [Zuo et al.,MICRO19]	√	√	√	√/×	√/×
Our Solution CCAE	√	√	√	√	√

CCAE: highlight and architecture

➢ **CCAE : efficient Crash-Consistency-Aware Encryption for NVM**

- Two key observations of data characteristic in existing data crash consistency
- Shared counter optimization (SCO) for log encryption
- Delayed counter persistency (DCP) for data encryption
- Implement based on ATOM [HPCA'17]

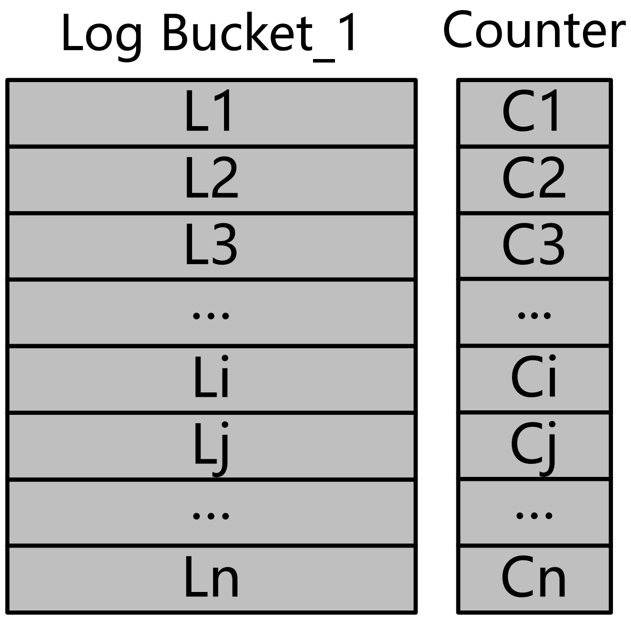


CCAE: Shared counter optimization (SCO)

➢SCO : For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:   Allocate Bucket_1
        St L1
        St L2
        St L3
        …
        St Li
        Reclaim Bucket_1
TX_B:   Allocate Bucket_1
        St L1
        St L2
        Reclaim Bucket_1
```



CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:     Allocate Bucket_1
       →  St L1
          St L2
          St L3
          …
          St Li
          Reclaim Bucket_1
TX_B:     Allocate Bucket_1
          St L1
          St L2
          Reclaim Bucket_1
```

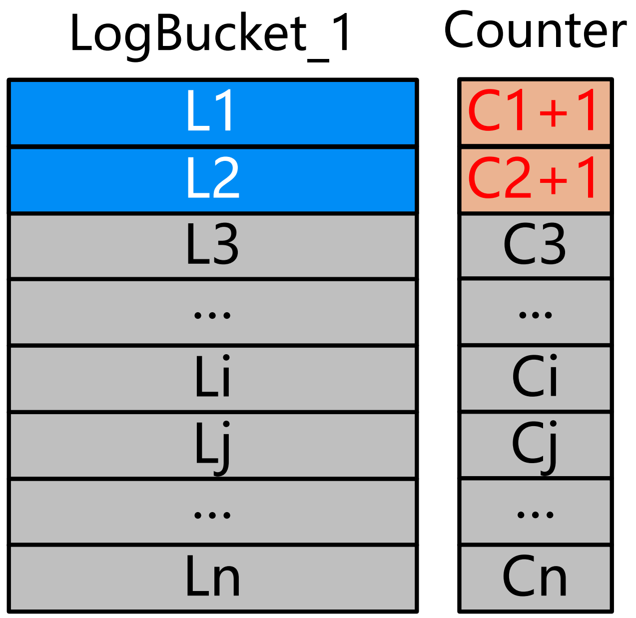
LogBucket_1	Counter
L1	C1+1
L2	C2
L3	C3
…	…
Li	Ci
Lj	Cj
…	…
Ln	Cn

CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:   Allocate Bucket_1
        St L1
   →    St L2
        St L3
        …
        St Li
        Reclaim Bucket_1
TX_B:   Allocate Bucket_1
        St L1
        St L2
        Reclaim Bucket_1
```

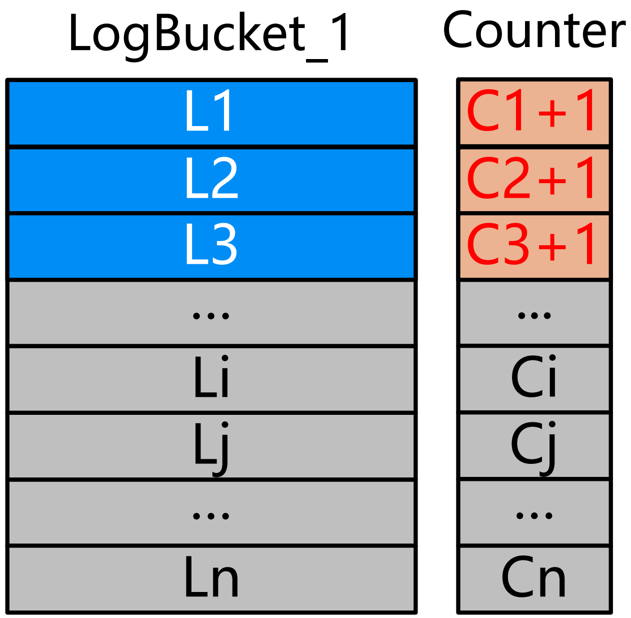


CCAE: Shared counter optimization (SCO)

➢SCO : For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:     Allocate Bucket_1
          St L1
          St L2
       →  St L3
          …
          St Li
          Reclaim Bucket_1
TX_B:     Allocate Bucket_1
          St L1
          St L2
          Reclaim Bucket_1
```

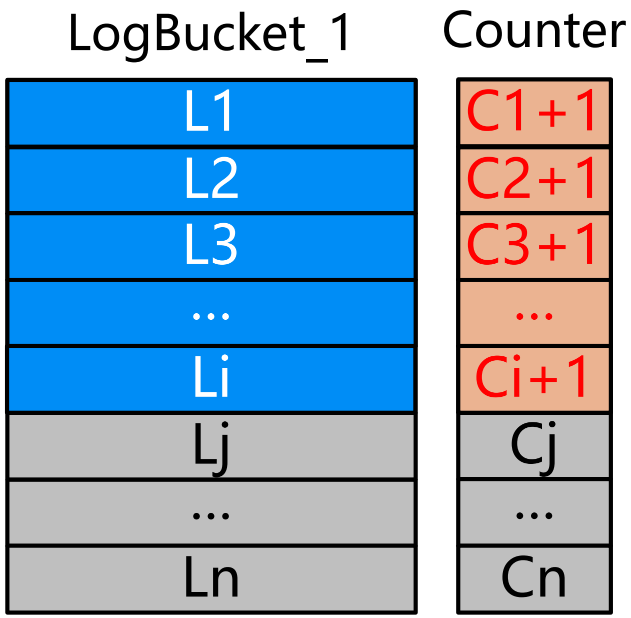


CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:     Allocate Bucket_1
          St L1
          St L2
          St L3
          …
      ➡   St Li
          Reclaim Bucket_1
TX_B:     Allocate Bucket_1
          St L1
          St L2
          Reclaim Bucket_1
```

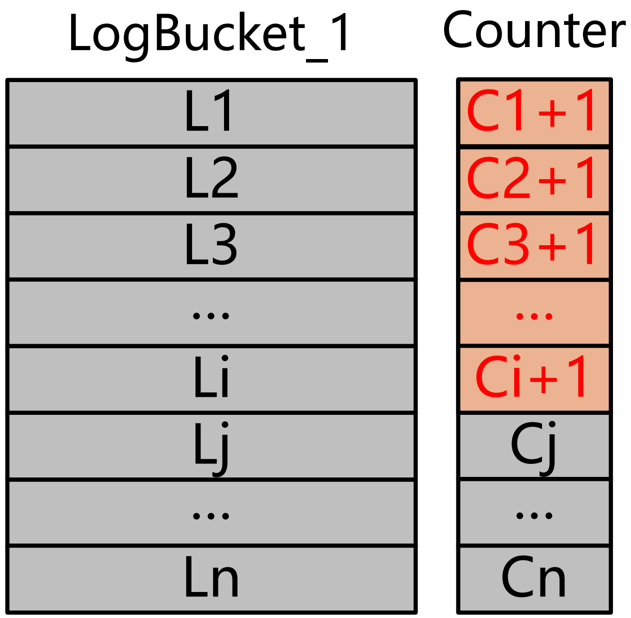


CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:      Allocate Bucket_1
           St L1
           St L2
           St L3
           …
           St Li
        →  Reclaim Bucket_1
TX_B:      Allocate Bucket_1
           St L1
           St L2
           Reclaim Bucket_1
```

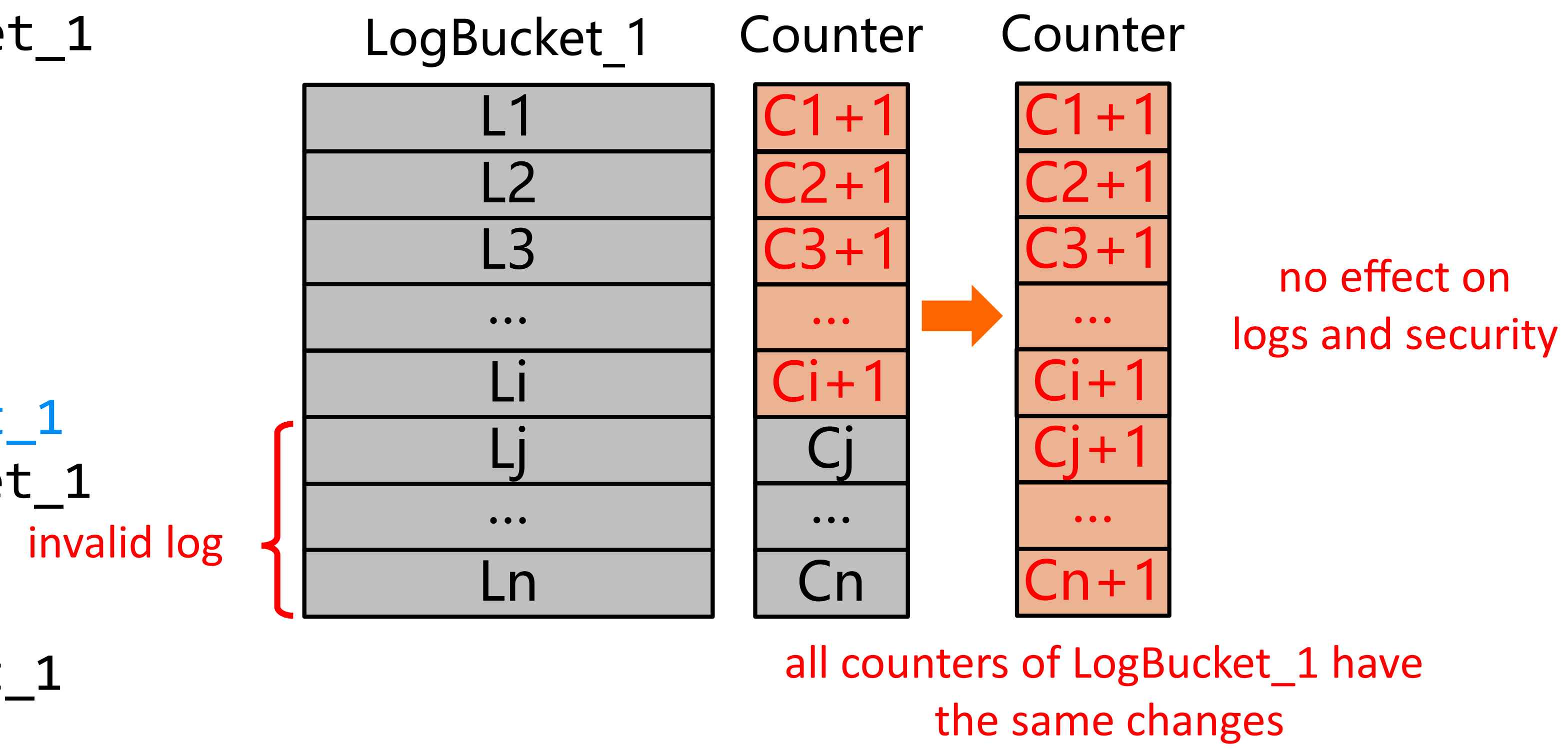


CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:    Allocate Bucket_1
         St L1
         St L2
         St L3
         …
         St Li
     ➡   Reclaim Bucket_1
TX_B:    Allocate Bucket_1
         St L1
         St L2
         Reclaim Bucket_1
```



CCAE: Shared counter optimization (SCO)

➢SCO : For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:      Allocate Bucket_1
           St L1
           St L2
           St L3
           …
           St Li
           Reclaim Bucket_1
TX_B: ---> Allocate Bucket_1
           St L1
           St L2
           Reclaim Bucket_1
```

LogBucket_1	Counter
L1	C1+1
L2	C2+1
L3	C3+1
…	…
Li	Ci+1
Lj	Cj+1
…	…
Ln	Cn+1

CCAE: Shared counter optimization (SCO)

➢SCO : For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:   Allocate Bucket_1
        St L1
        St L2
        St L3
        …
        St Li
        Reclaim Bucket_1
TX_B:   Allocate Bucket_1
   →    St L1
        St L2
        Reclaim Bucket_1
```

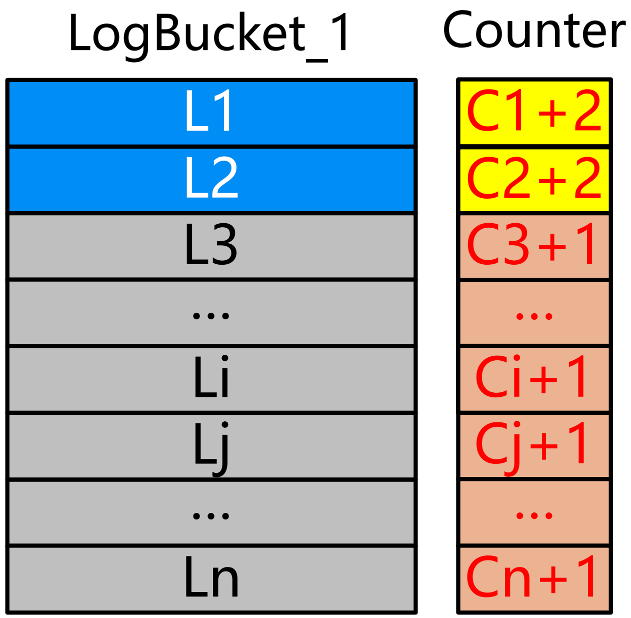
LogBucket_1	Counter
L1	C1+2
L2	C2+1
L3	C3+1
…	…
Li	Ci+1
Lj	Cj+1
…	…
Ln	Cn+1

CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:     Allocate Bucket_1
          St L1
          St L2
          St L3
          …
          St Li
          Reclaim Bucket_1
TX_B:     Allocate Bucket_1
          St L1
      ➔   St L2
          Reclaim Bucket_1
```



CCAE: Shared counter optimization (SCO)

➢SCO : For log encryption

- append characteristic of log writing in NVM durable transactions
- During an allocation-recycling phase of a bucket, counters will only increase by one or remain unchanged

```
TX_A:   Allocate Bucket_1
        St L1
        St L2
        St L3
        …
        St Li
        Reclaim Bucket_1
TX_B:   Allocate Bucket_1
        St L1
        St L2
    ➡   Reclaim Bucket_1
```

LogBucket_1	Counter
L1	C1+2
L2	C2+2
L3	C3+2
…	…
Li	Ci+2
Lj	Cj+2
…	…
Ln	Cn+2

	C1	C2	…	Ci	Cj	…	Cn
Initial:	0	0	…	0	0	…	0
1st allocated:	1	1	…	1	1	…	1
2nd allocated:	2	2	…	2	2	…	2
……							
xth allocated:	x	x	…	x	x	…	x

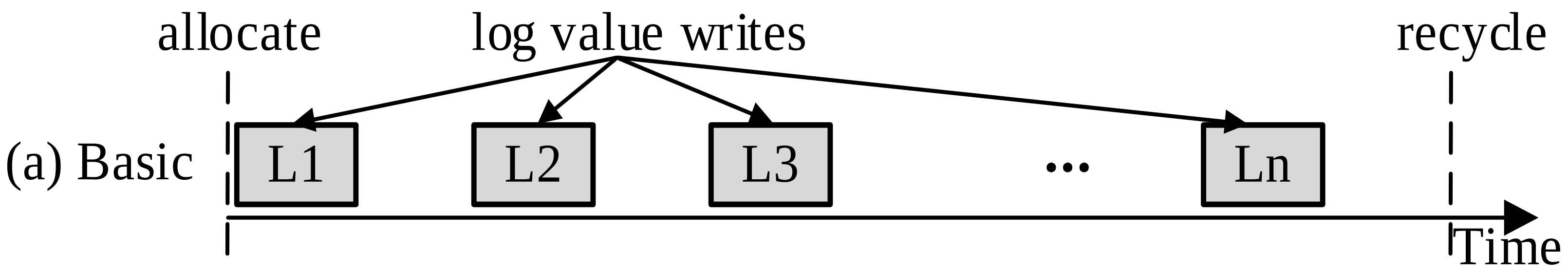
Observation: counters for logs in a bucket have the same values!

CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- all counters for cachelines in a bucket have the same values
- log cachelines in a bucket shared a bucket_counter
- strict persist for consistency. Since there are only a few bucket_counters and updates

(a log bucket has 2^16 log cachelines) (please refer to the paper for the details of record_header)

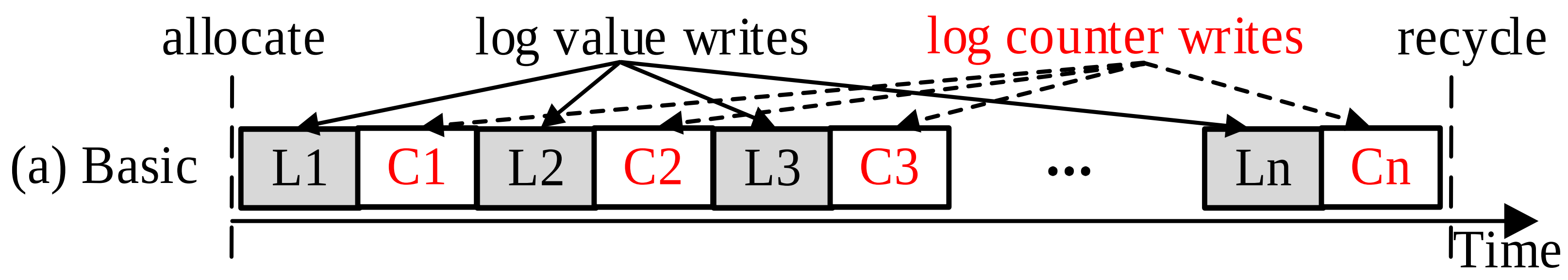


CCAE: Shared counter optimization (SCO)

➢SCO : For log encryption

- all counters for cachelines in a bucket have the same values
- log cachelines in a bucket shared a bucket_counter
- strict persist for consistency. Since there are only a few bucket_counters and updates

(a log bucket has 2^16 log cachelines) (please refer to the paper for the details of record_header)

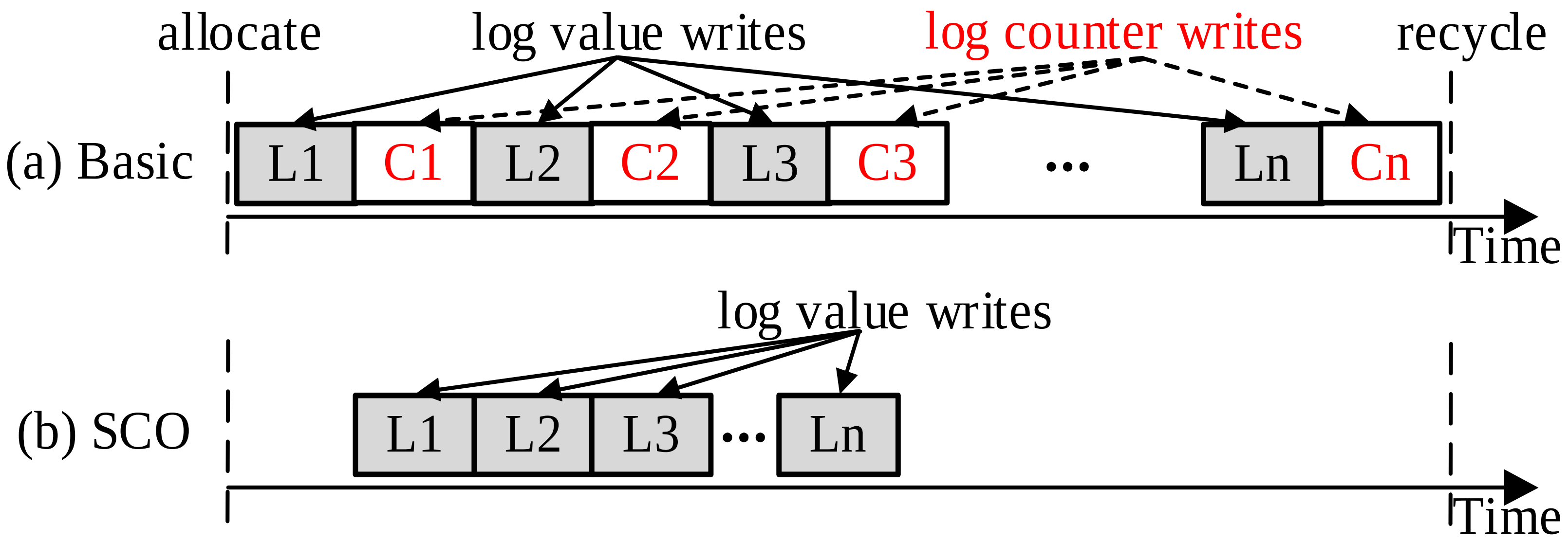


CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- all counters for cachelines in a bucket have the same values
- log cachelines in a bucket shared a bucket_counter
- strict persist for consistency. Since there are only a few bucket_counters and updates

(a log bucket has 2^16 log cachelines) (please refer to the paper for the details of record_header)

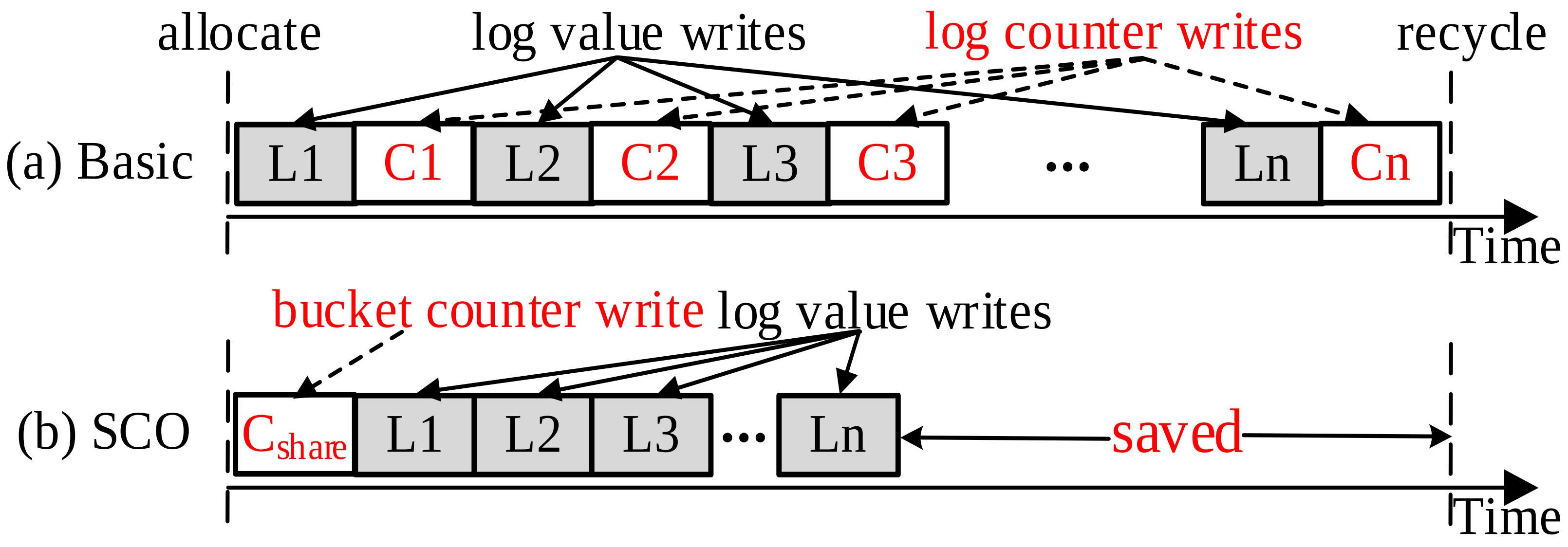


CCAE: Shared counter optimization (SCO)

➢SCO： For log encryption

- all counters for cachelines in a bucket have the same values
- log cachelines in a bucket shared a bucket_counter
- strict persist for consistency. Since there are only a few bucket_counters and updates

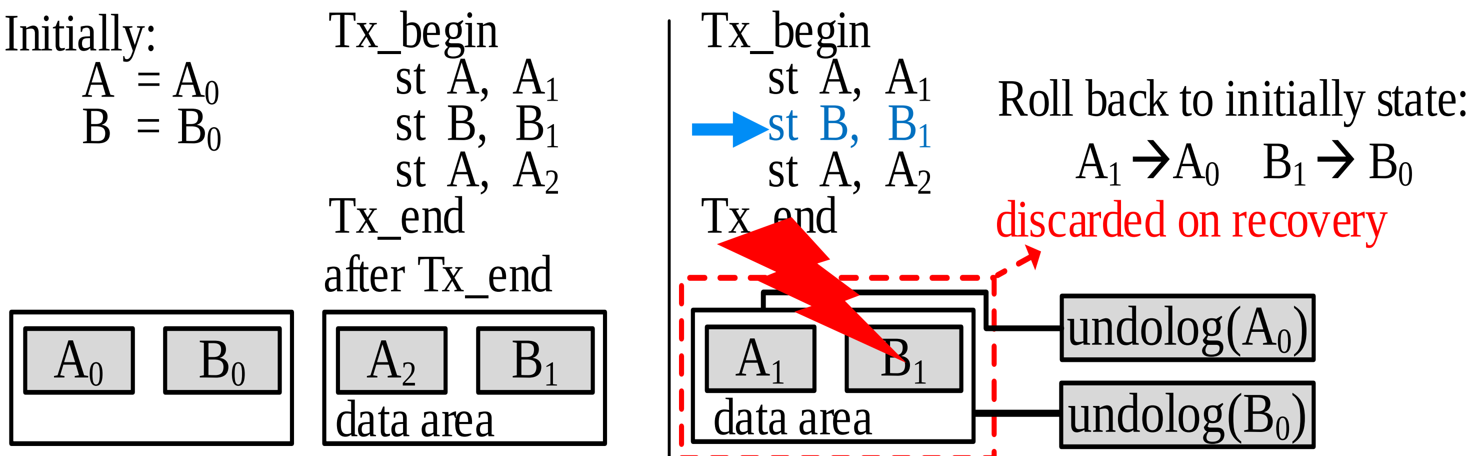
(a log bucket has 2^16 log cachelines) (please refer to the paper for the details of record_header)



CCAE: Delayed Counter Persistency (DCP)

➢DCP : For data encryption

- uncommitted transactions will be discarded
- no need to restore the latest values of the corresponding counters
- restore the counters to the newer value before crash to avoid OTP reuse → **period-write**



(a) Initially and committed states

(b) crash happens before commit

```
St A , Ctr_A++
if Ctr_A % N == 0?
    perist Ctr_A
else
    update A in cache
……
if Tx about to end?
    persist all working ctr
```

2 period-write

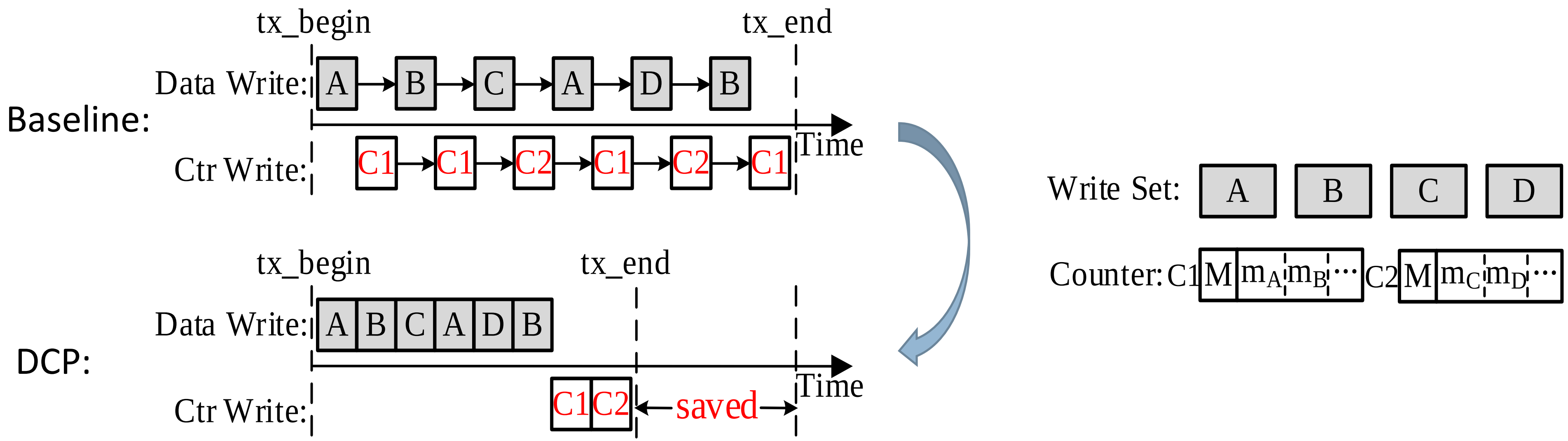
N→write limit

1 delayed persist

CCAE: Delayed Counter Persistency (DCP)

➢DCP： For data encryption

- uncommitted transactions will be discarded
- no need to restore the latest values of the corresponding counters
- restore the counters to the the newer value before crash to avoid OTP reuse → **period-write**



Since write-limit N is set to 32, few data blocks can be written more than N times, so we did not draw the counter write caused by write-limit solution. (but this part is included in the experiment)

CCAE: System Recovery

- **Log Counter recovery**
  - direct read
- **Data Counter recovery**
  - Ctr_stale + write_limit
- **Data recovery**
  - undo log
- **Counter Recovery time**
  - negligible

Algorithm 1: Recovery Process of CCAE

Input: *write_limit*

// 1 Find uncommitted transactions
Read log metadata;
Find uncommitted transactions *uncommit_TX*;
// 2 Recover uncommitted transactions
for *all* TX_i *in uncommit_TX* **do**
    // 2.1 read and decrypt undo log
    for *all bucket in log_bucket* **do**
        Bucket_Ctr ← Read shared bucket counter;
        Read log headers (counter in them) and decrypt;
        Read log values and decrypt them with *Bucket_Ctr* ;
    end for
    // 2.2 recover data counter
    for *all Ctr in data_counter* **do**
        Ctr_stale ← Read counter in NVM;
        Ctr_recovery = *Ctr_stale* + *write_limit*;
    end for
    // 2.3 recover data
    Roll back the data with undo log and *Ctr_recovery* .
end for

Outline

- 1 Background
- 2 CCAE
- **3 Experiment**
- 4 Conclusion

CCAE: Performance Evaluation

- **Model secure NVM using GEM5 simulator**

Table 1: System Configurations

Processor	
CPU	4-core, 1GHz, X86-64, out-of-order
L1 Cache	private, 4 cycles, 32KB, 8-way, 64B block
L2 Cache	private, 12 cycles, 512KB, 8-way, 64B block
L3 Cache	shared, 28 cycles, 8MB, 16-way, 64B block
DDR-based PCM Main Memory	
Capacity	16GB PCM
Latency	36ns row hit 100/300ns read/write row conflict
Energy	0.93 (1.02) pJ/bit row buffer read (write) 2.47 (16.82) pJ/bit PCM array read (write)
Security Parameters	
En/decryption	40ns AES
Counter Cache	256KB

Table 2: Evaluated Workloads

Workload	Description
B-Tree	Insert/delete nodes in a b-tree
Hash	Insert/delete entries in a hash table
Queue	Insert/delete entries in a queue
RB-Tree	Insert/delete nodes in a rb-tree
SDG	Insert/delete edges in a scalable graph
SPS	Random swap entries in an array

CCAE :Experiment Results

SP： the original solution   Osiris: one state-of-the-art solution

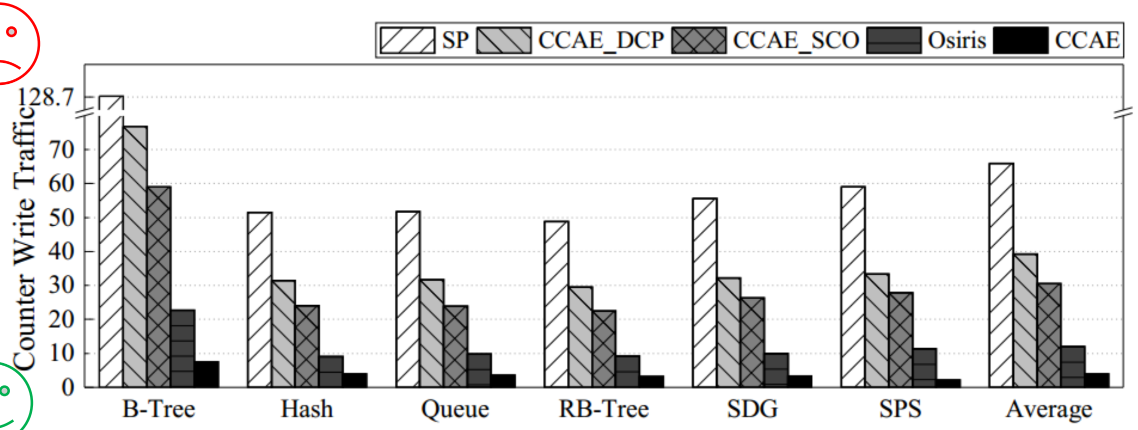


Figure 13: Counter write traffic of different systems.

Compared to SP/Osiris[micro18], the **NVM writes cause by counters** in CCAE are reduced 65x/67% , while the **system execution time** is reduced by 53%/14% , The **NVM energy** is reduced by 96%/35% .

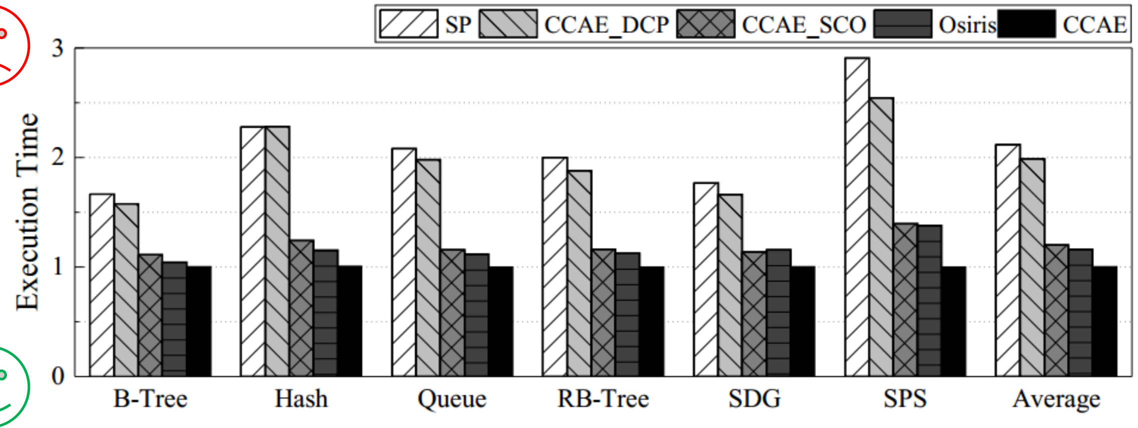


Figure 14: System execution time of different systems.

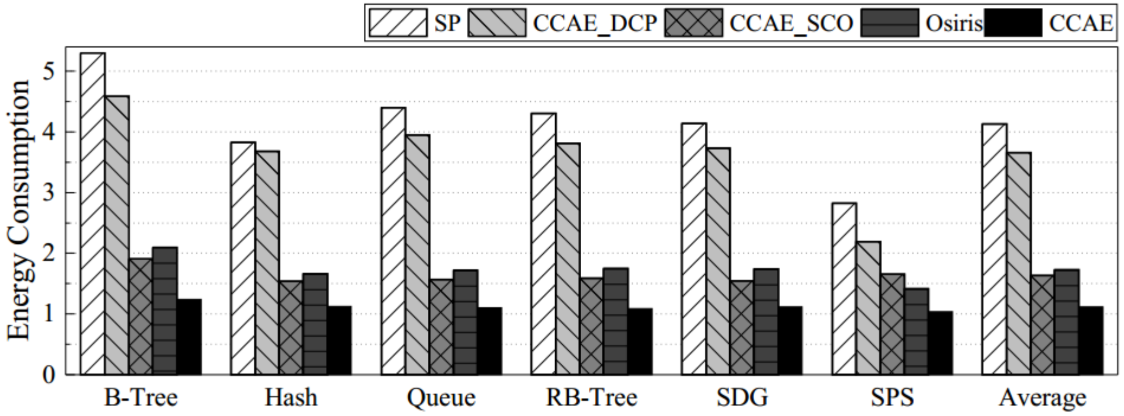


Figure 15: The energy consumption of different systems.

CCAE :Experiment Results

SP： the original solution Osiris: one state-of-the-art solution

Compared to SP/Osiris[micro18], the **NVM writes cause by counters** in CCAE are reduced **65x/67%** , while the **system execution time** is reduced by 53%/14% , The **NVM energy** is reduced by 96%/35% .

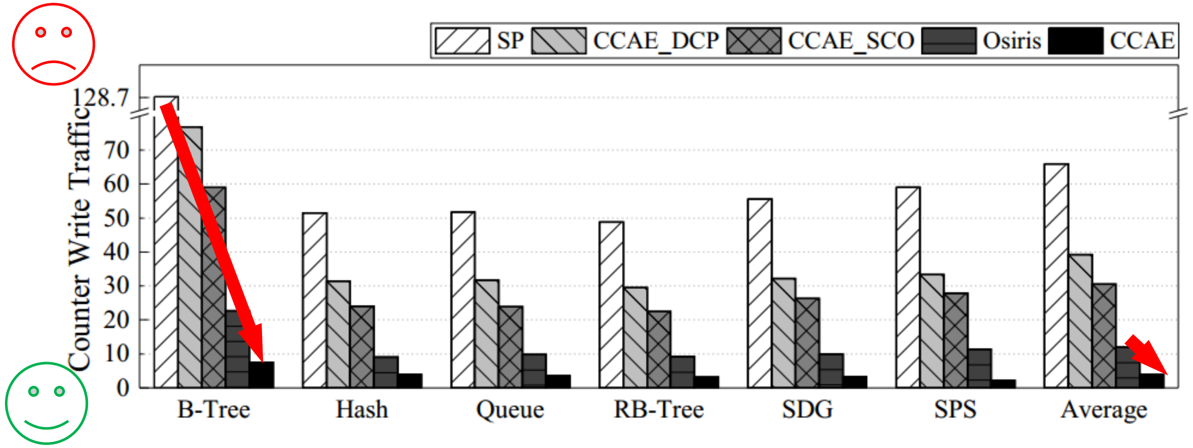


Figure 13: Counter write traffic of different systems.

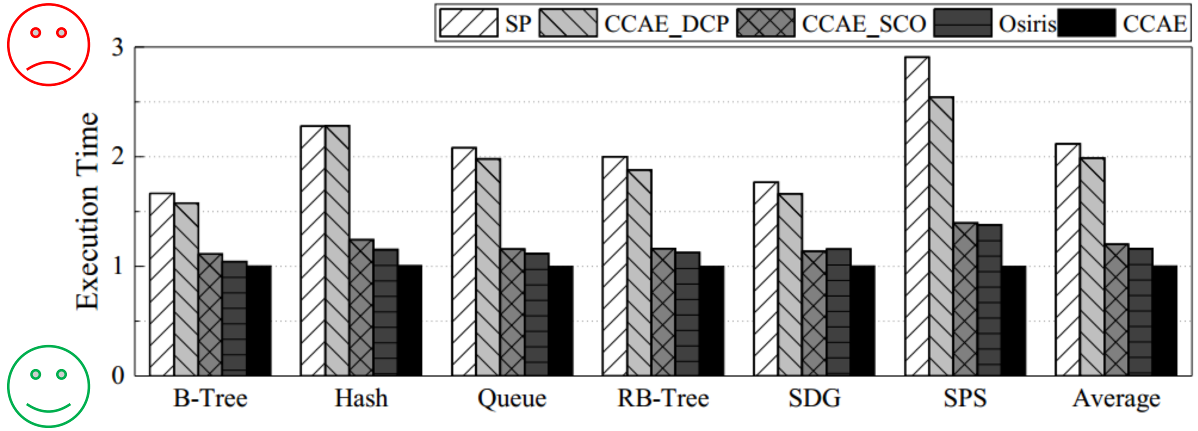


Figure 14: System execution time of different systems.

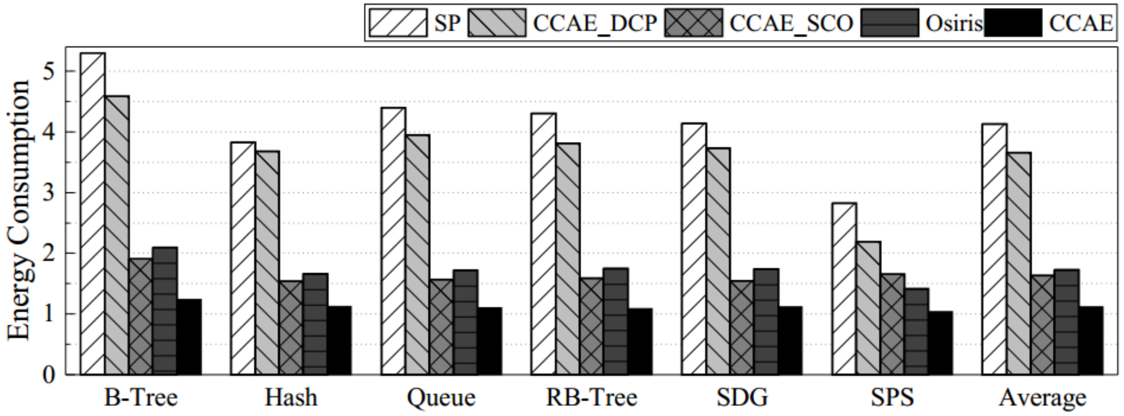


Figure 15: The energy consumption of different systems.

CCAE :Experiment Results

SP： the original solution   Osiris: one state-of-the-art solution

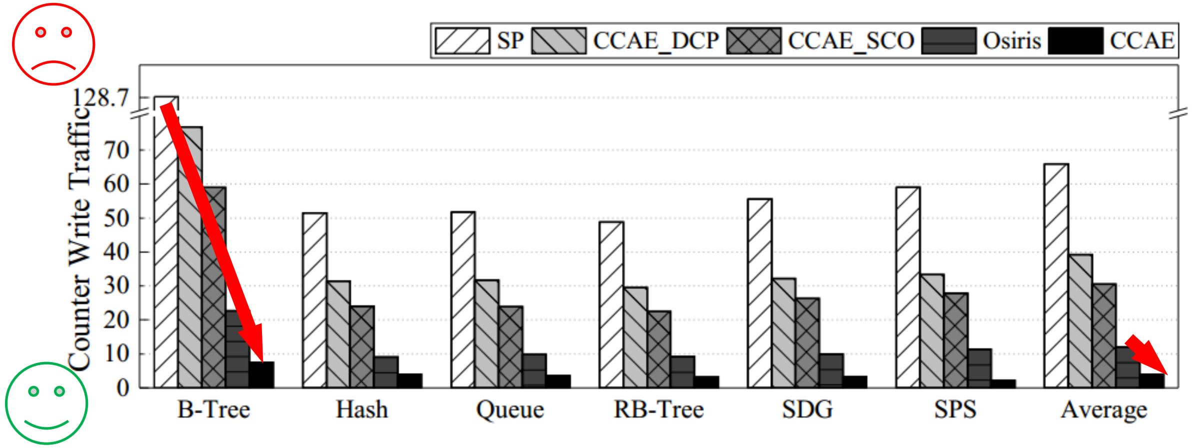


Figure 13: Counter write traffic of different systems.

Compared to SP/Osiris[micro18], the **NVM writes cause by counters** in CCAE are reduced **65x/67%** , while the **system execution time** is reduced by **53%/14%** , The **NVM energy** is reduced by 96%/35% .

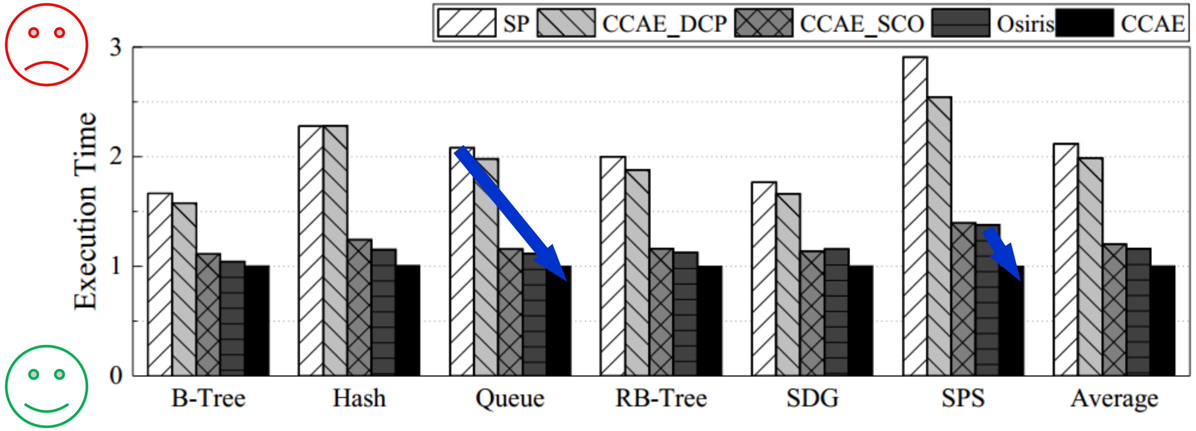


Figure 14: System execution time of different systems.

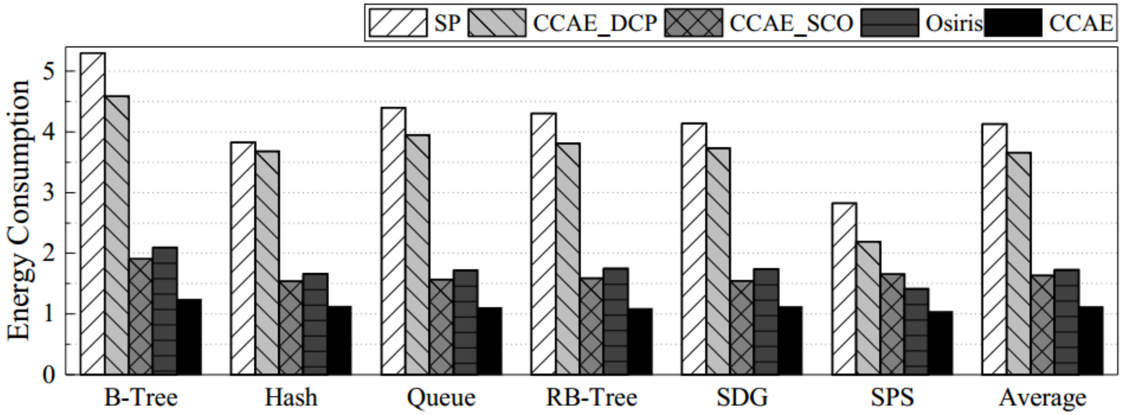


Figure 15: The energy consumption of different systems.

CCAE :Experiment Results

SP： the original solution   Osiris: one state-of-the-art solution

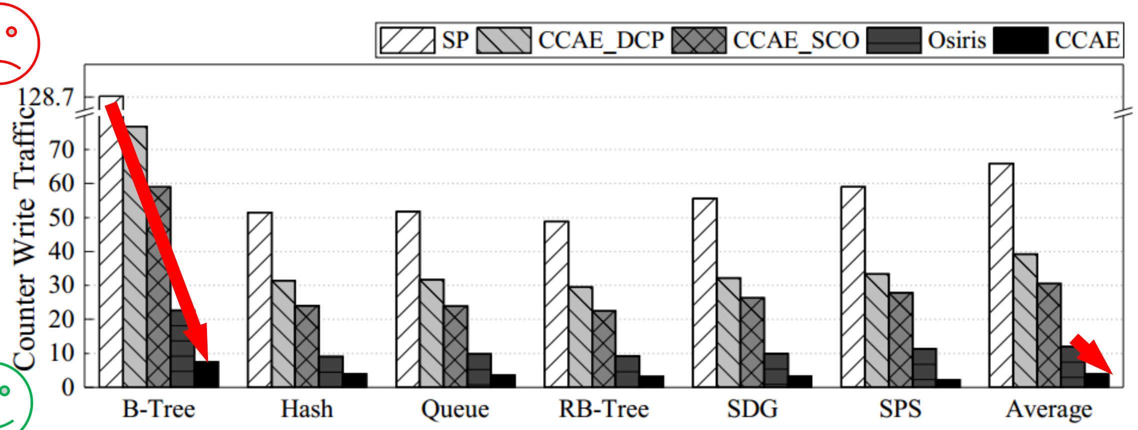


Figure 13: Counter write traffic of different systems.

Compared to SP/Osiris[micro18], the **NVM writes cause by counters** in CCAE are reduced **65x/67%** , while the **system execution time** is reduced by **53%/14%** , The **NVM energy** is reduced by **96%/35%**.

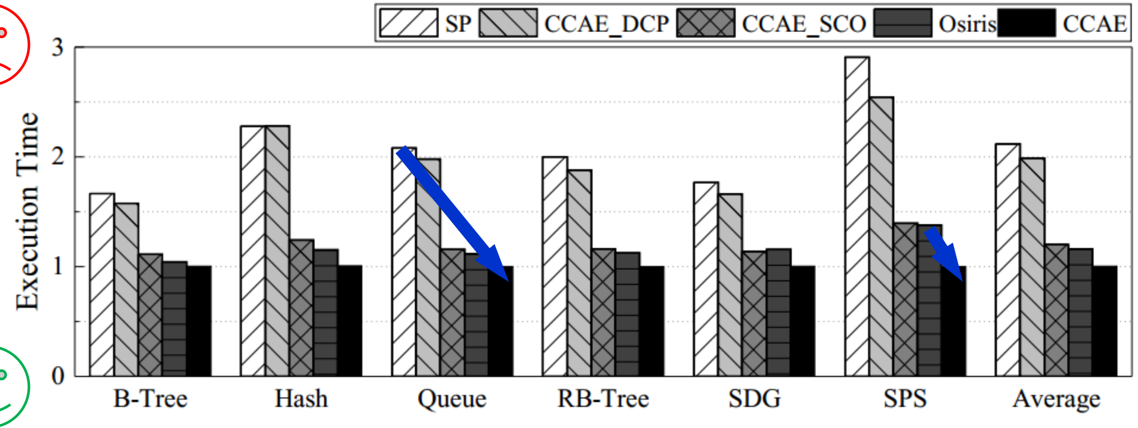


Figure 14: System execution time of different systems.

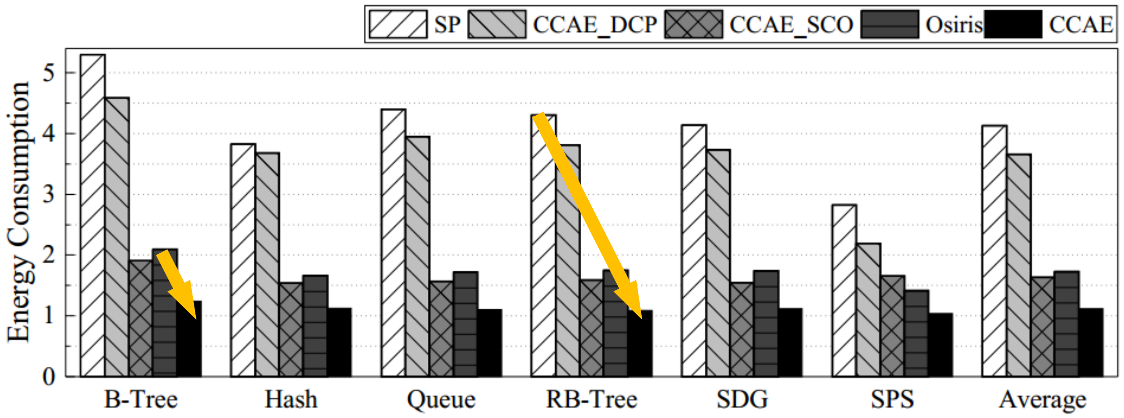


Figure 15: The energy consumption of different systems.

Outline

- 1 Background
- 2 CCAE
- 3 Experiment
- **4 Conclusion**

CCAE: Conclusion

- **Problem**
  - ✓ Existing counter crash consistency solutions ingore data features in NVM
- **CCAE enables efficient counter crash consistency**
  - Shared counter optimization (SCO) for log encryption
  - Delayed counter persistency (DCP) for data encryption
- **Results**
  - ✓ less NVM writes for counters: 65x/67% to SP/Osiris [Ye et al., MICRO18]
  - ✓ low system overhead: 53%/14% to SP/Osiris [Ye et al., MICRO18]
  - ✓ low NVM energy: 96%/35% to SP/Osiris [Ye et al., MICRO18]
  - ✓ fast recovery & high scalability

Thanks!