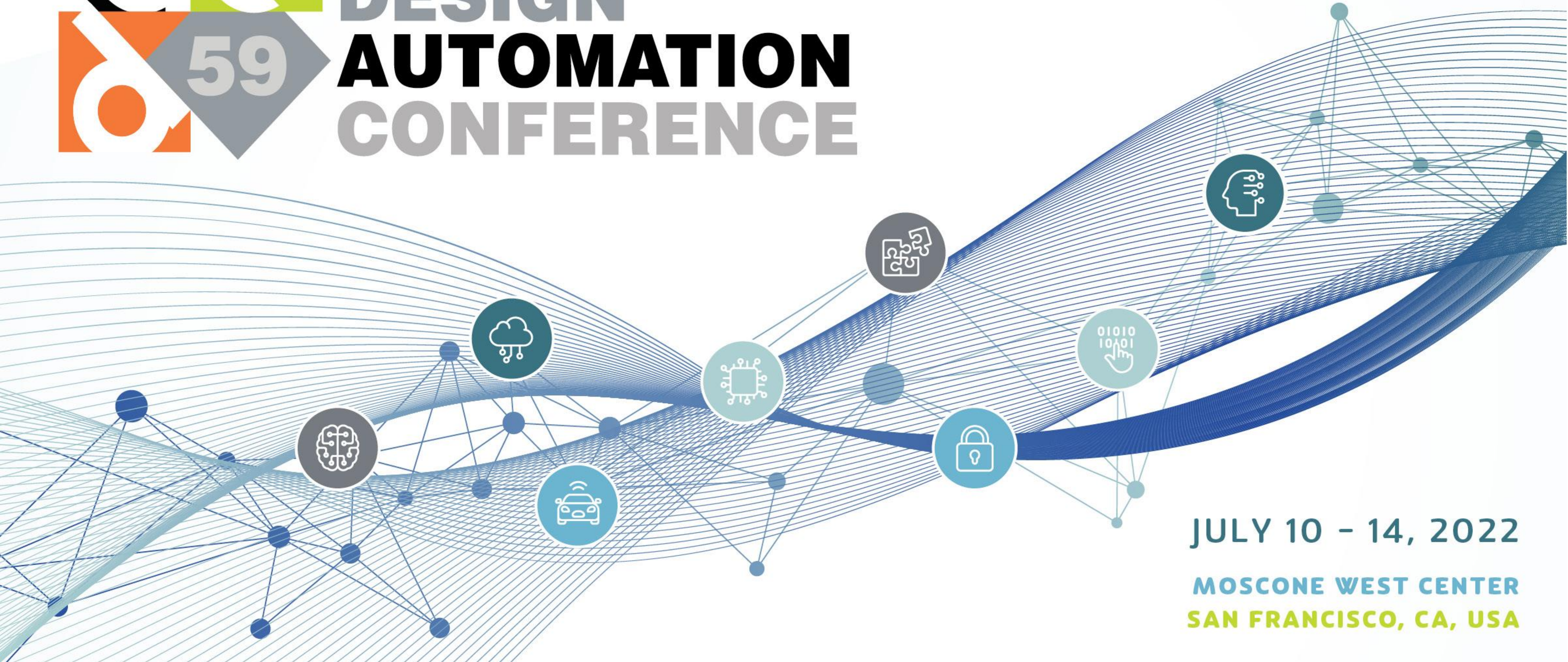
59

DESIGN AUTOMATION CONFERENCE



JULY 10 - 14, 2022

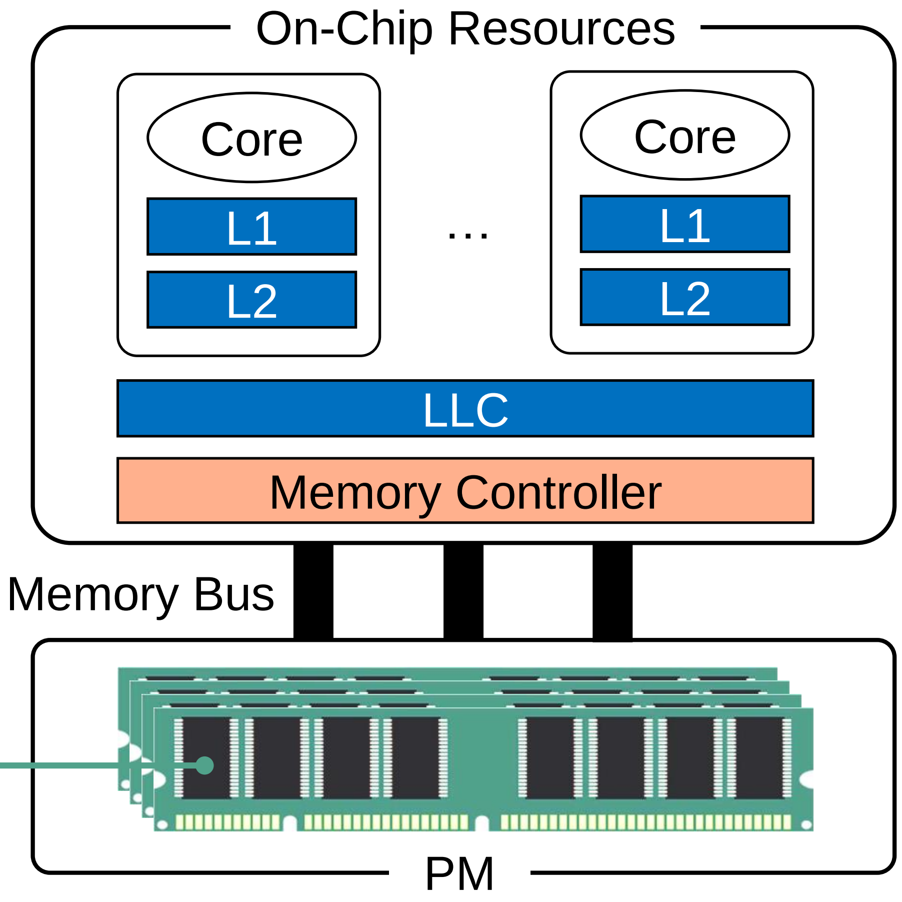
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Scalable Crash Consistency for Secure Persistent Memory

Ming Zhang, Yu Hua, Xuan Li, Hao Xu

Huazhong University of Science and Technology, China

Persistent Memory (PM)



Non-volatility:

Data are not lost

Persistent Memory (PM)

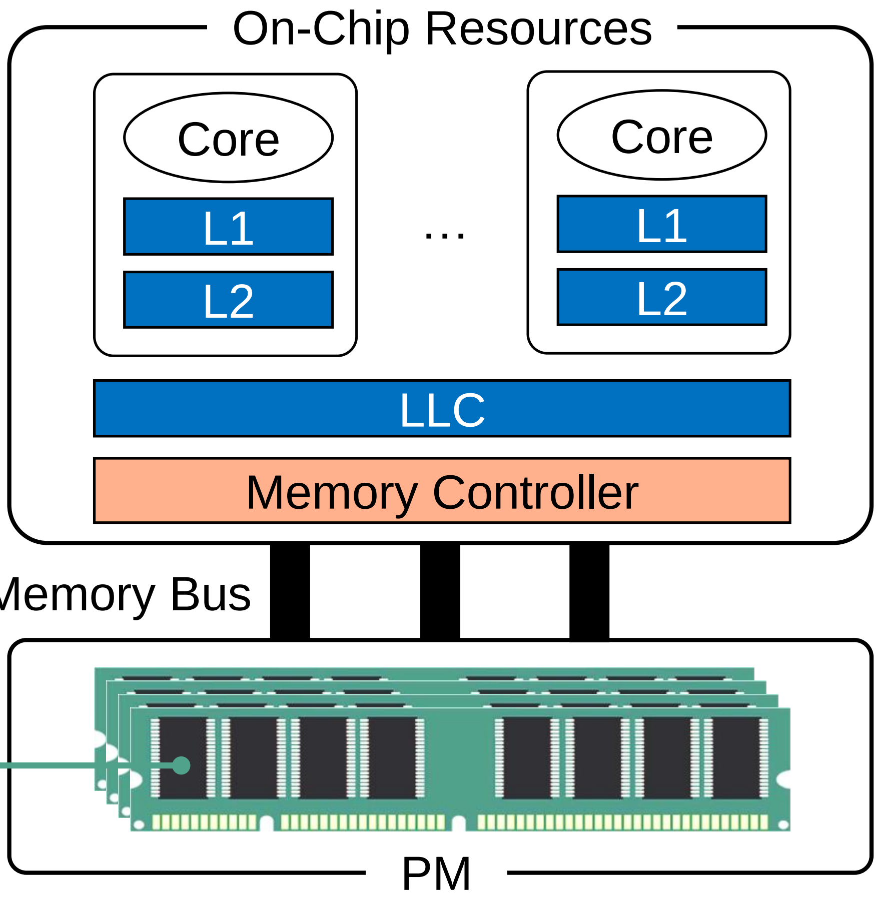
Requirements

① **Security**

② **Crash Consistency**

Non-volatility:

Data are not lost



On-Chip Resources

Core

L1

L2

…

Core

L1

L2

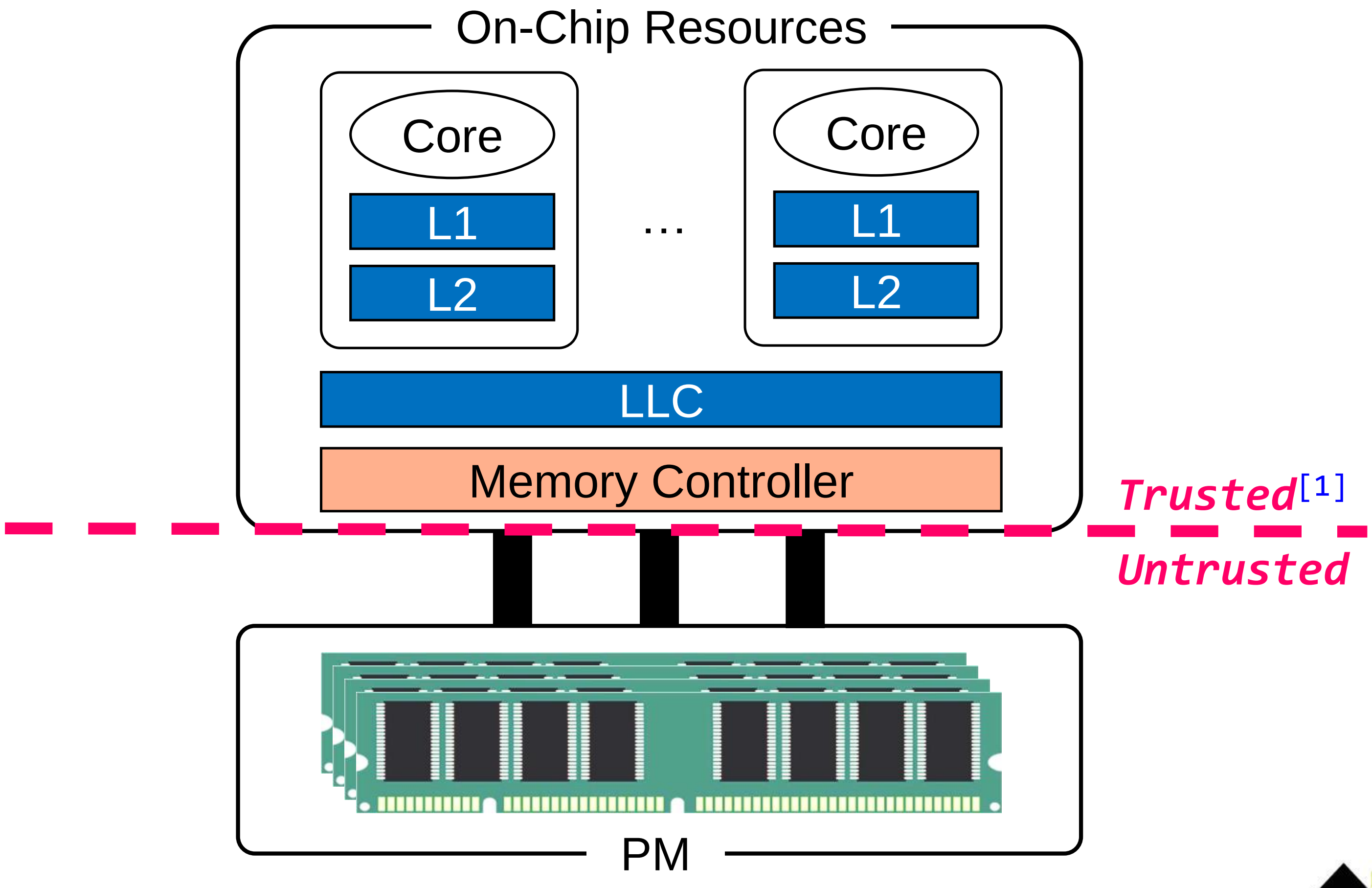
LLC

Memory Controller

Memory Bus

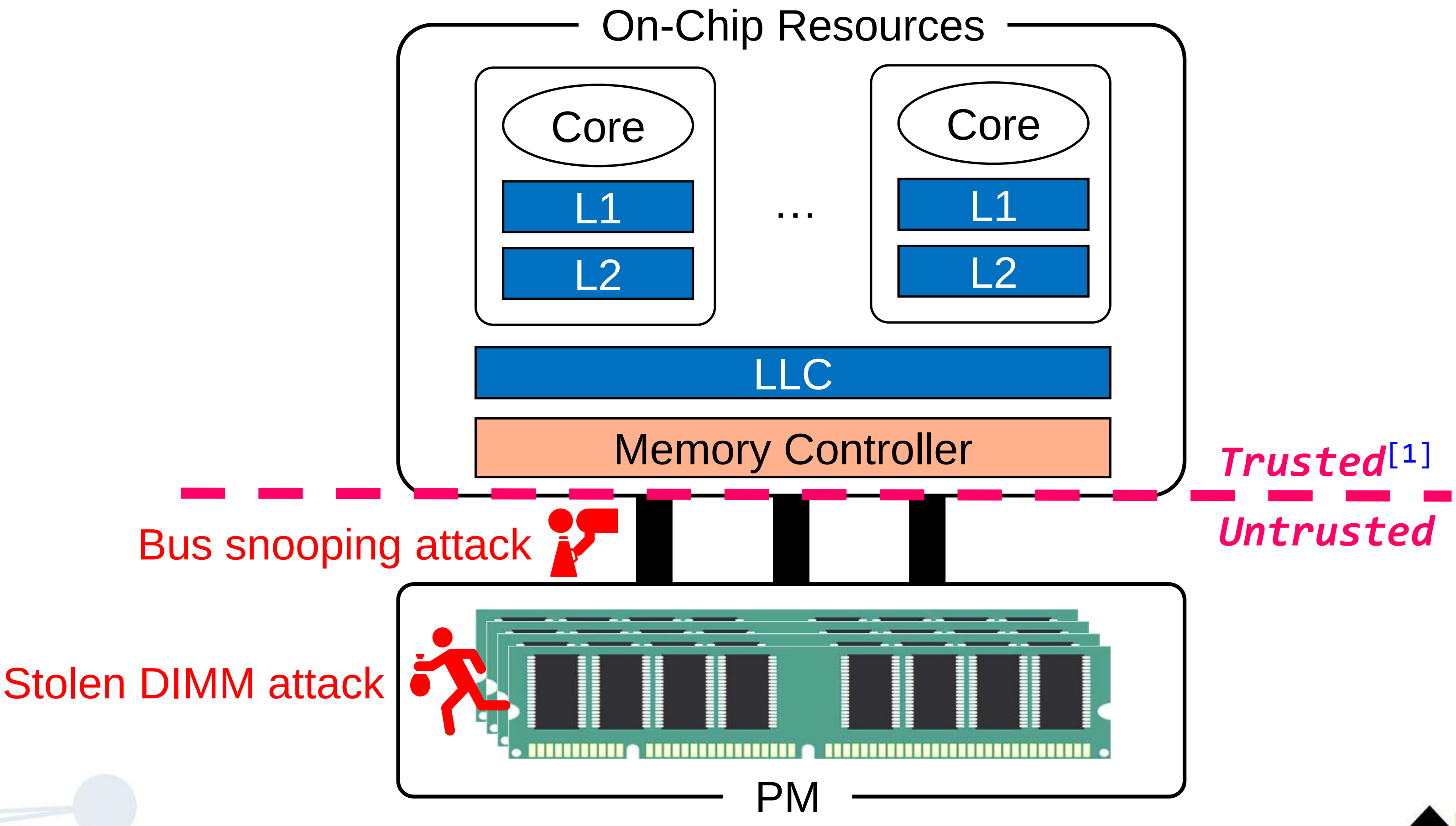
PM

Security for PM



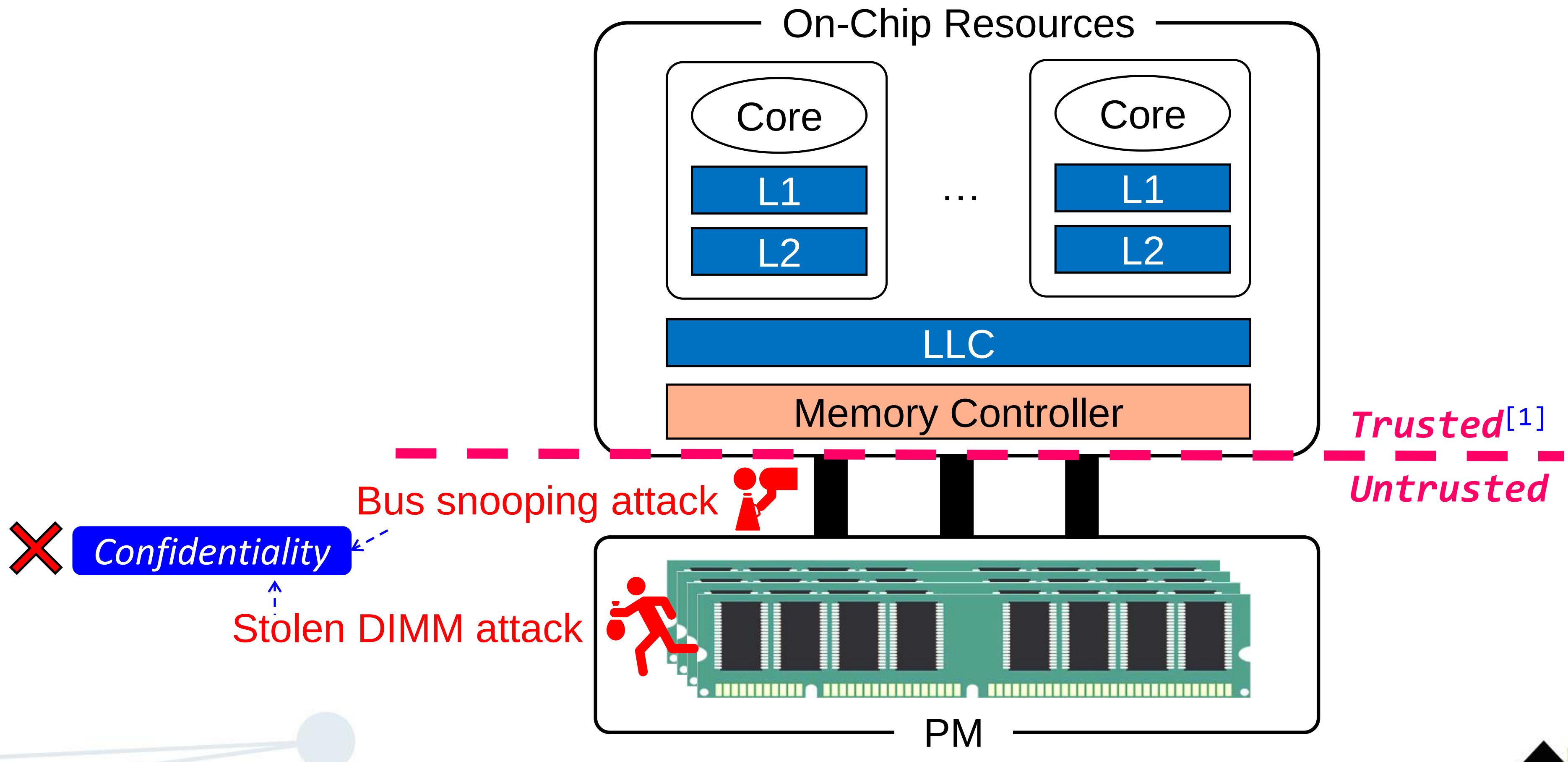
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



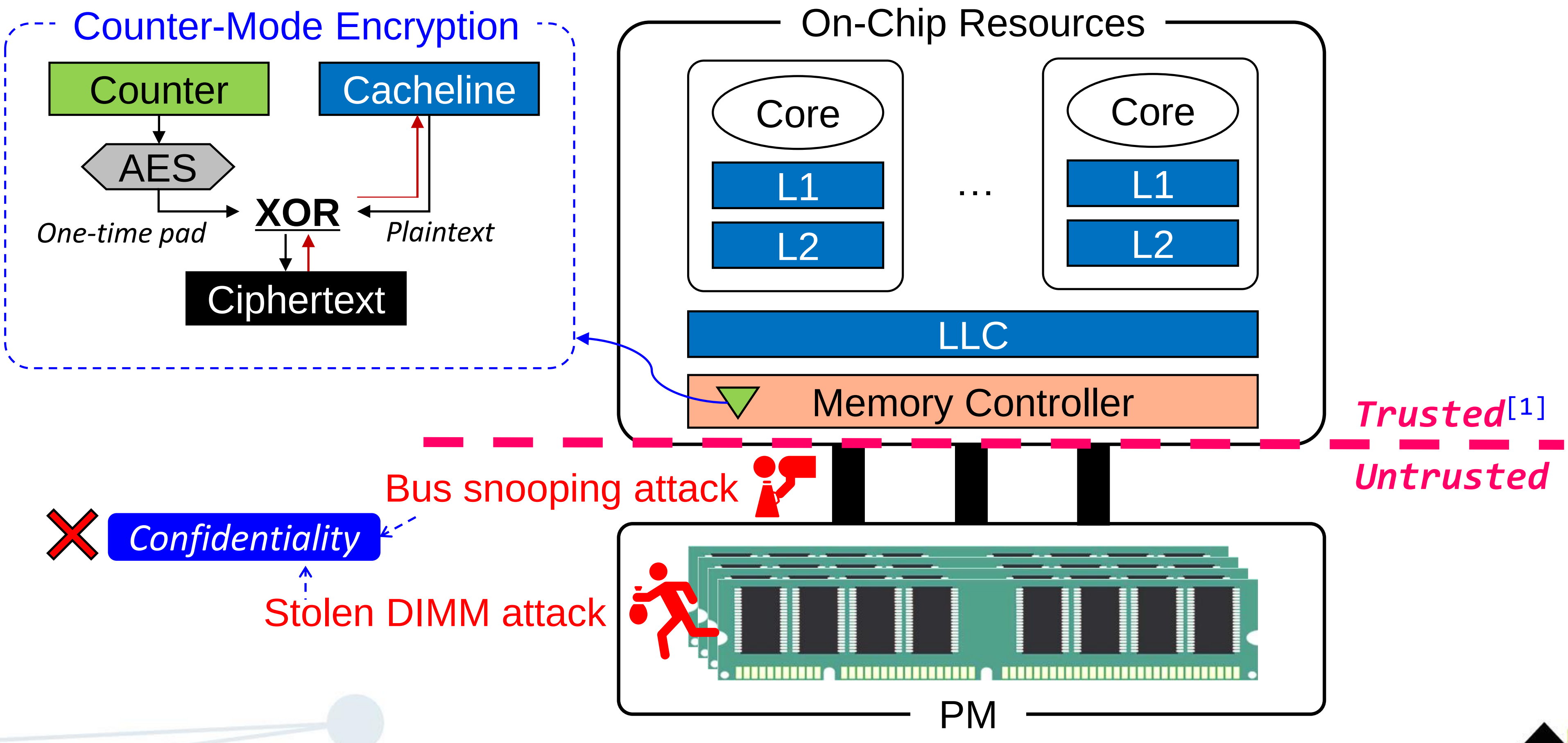
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



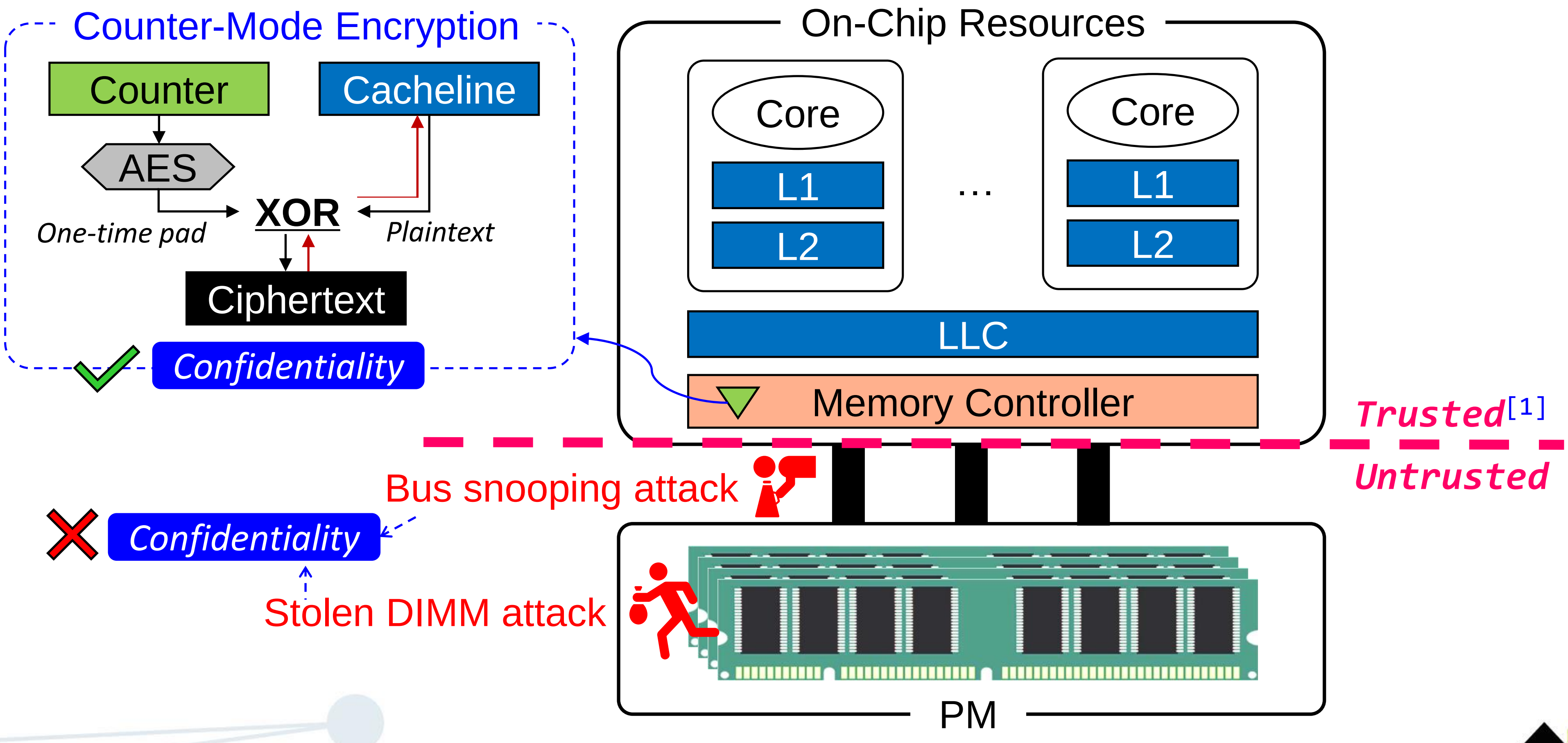
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



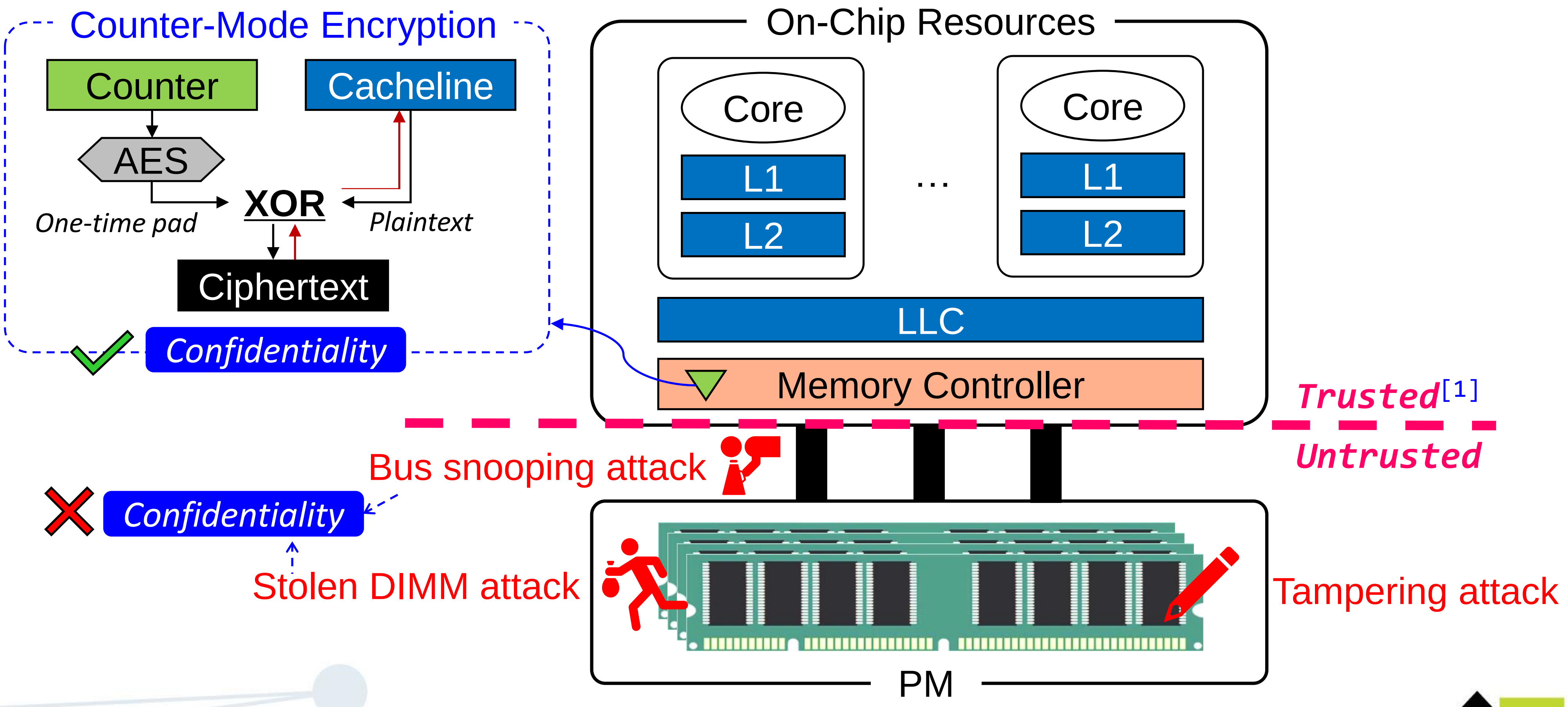
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



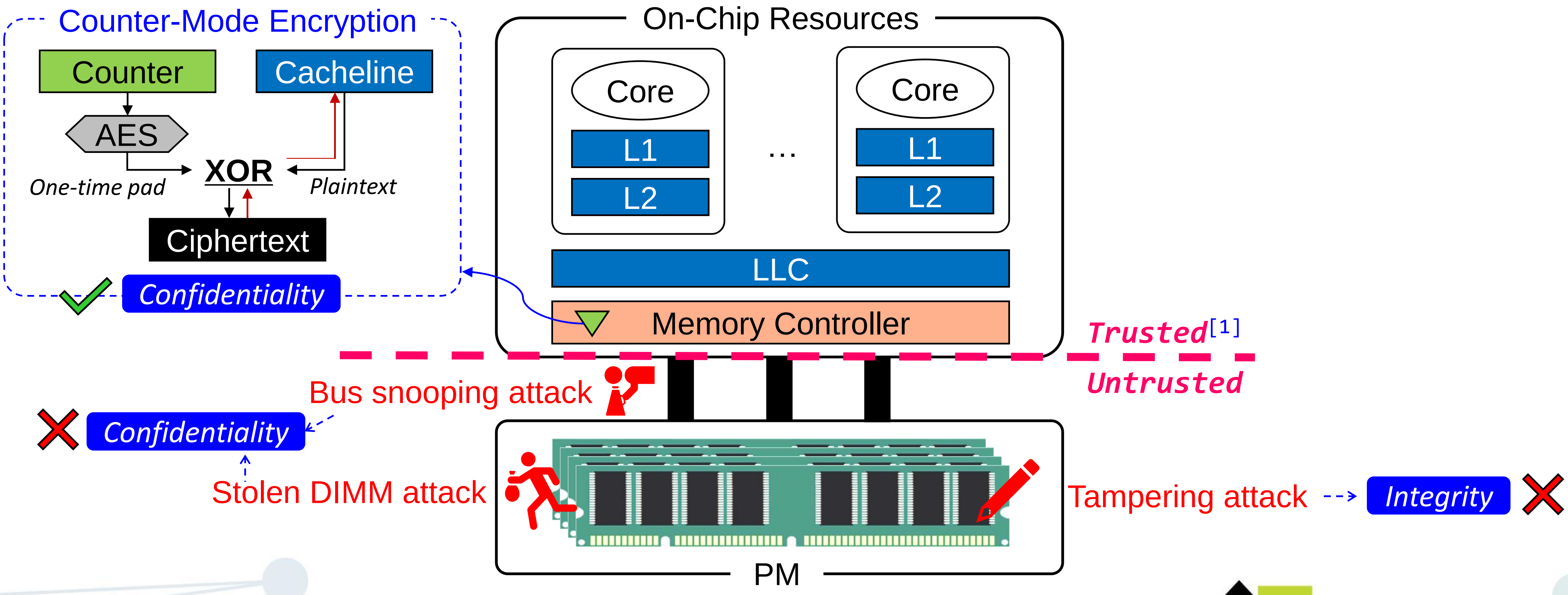
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



Counter-Mode Encryption

Counter

Cacheline

AES

XOR

One-time pad

Plaintext

Ciphertext

✓ *Confidentiality*

On-Chip Resources

Core

L1

L2

…

Core

L1

L2

LLC

Memory Controller

Trusted[1]

Untrusted

Bus snooping attack

✗ *Confidentiality*

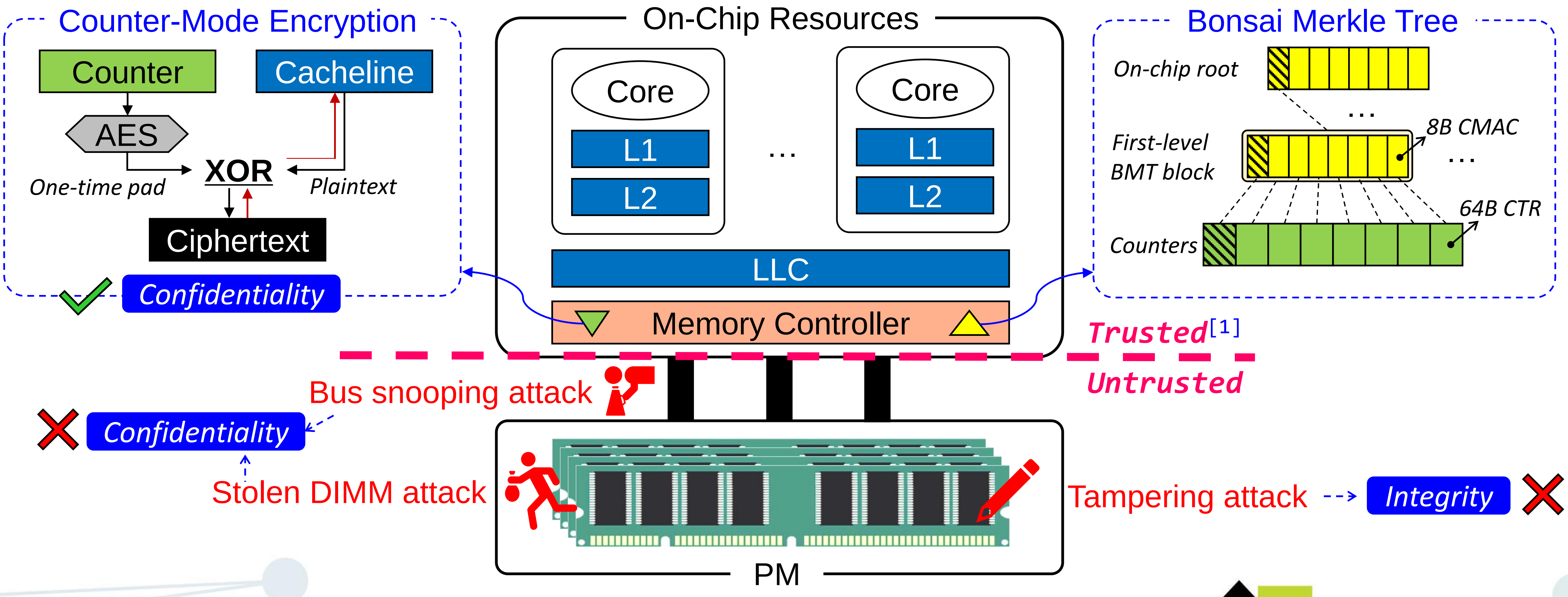
Stolen DIMM attack

Tampering attack --> *Integrity* ✗

PM

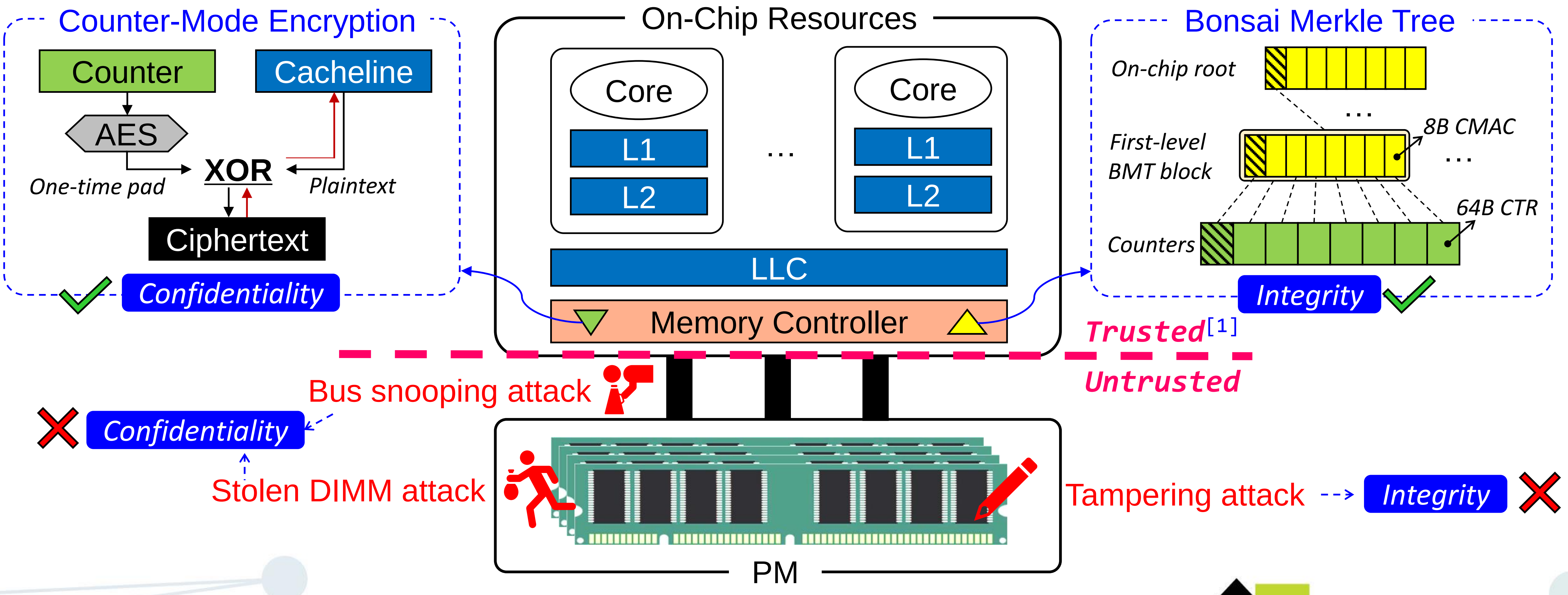
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



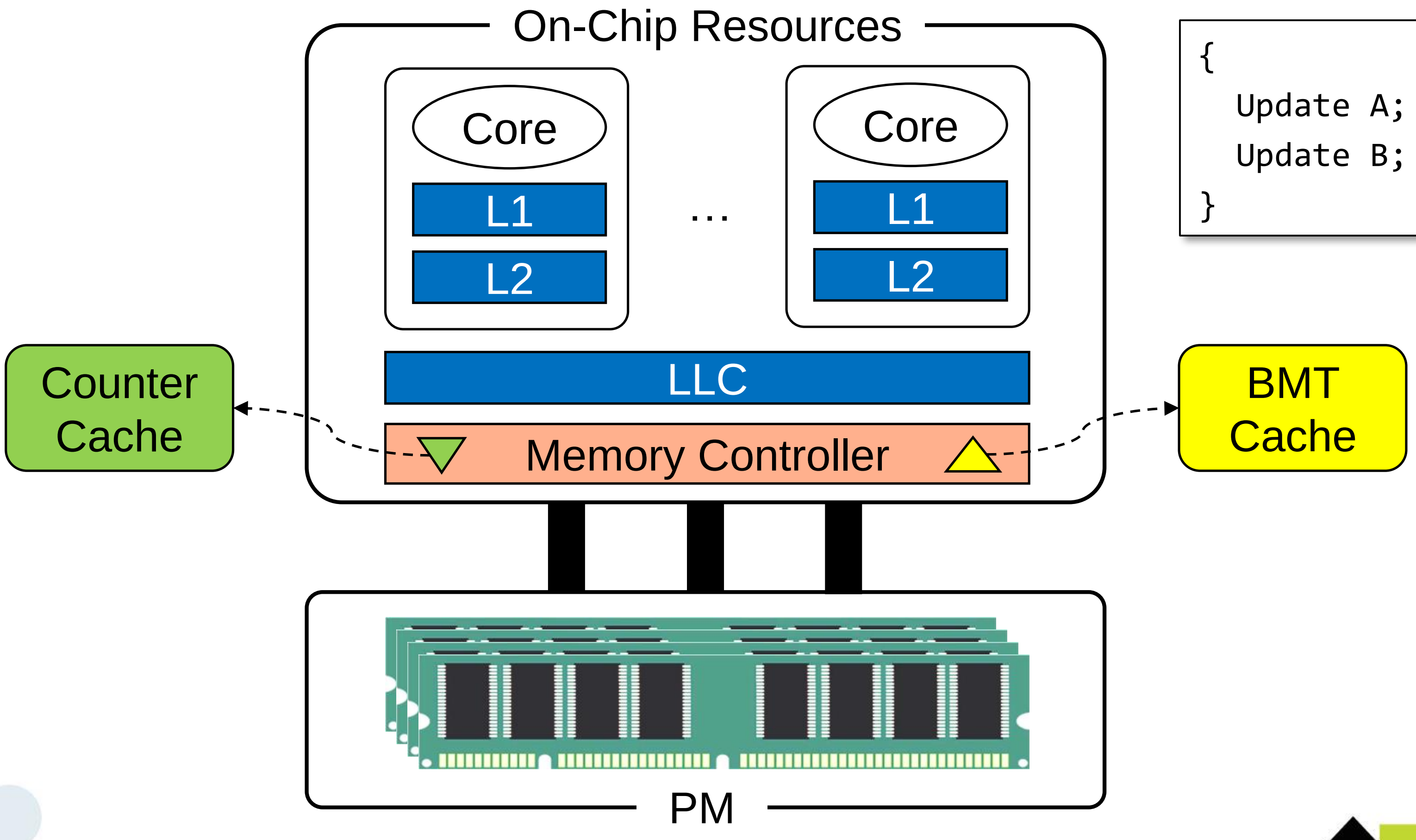
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



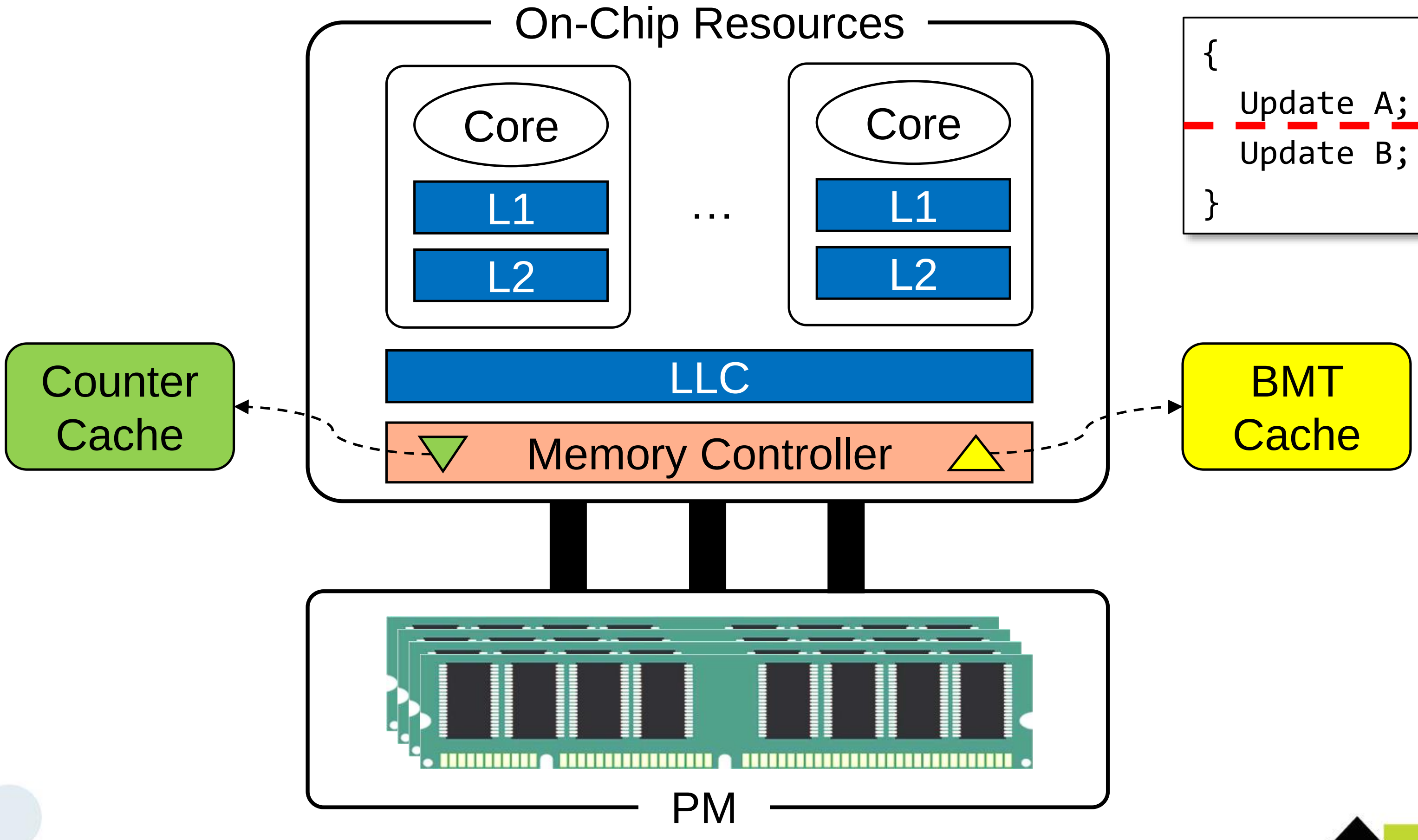
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

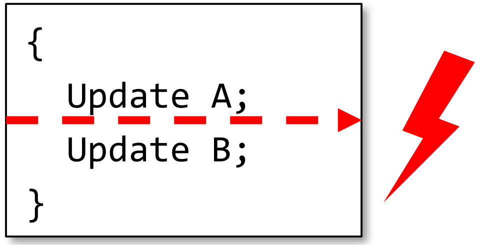
Crash Consistency for Secure PM



```
{
  Update A;
  Update B;
}
```

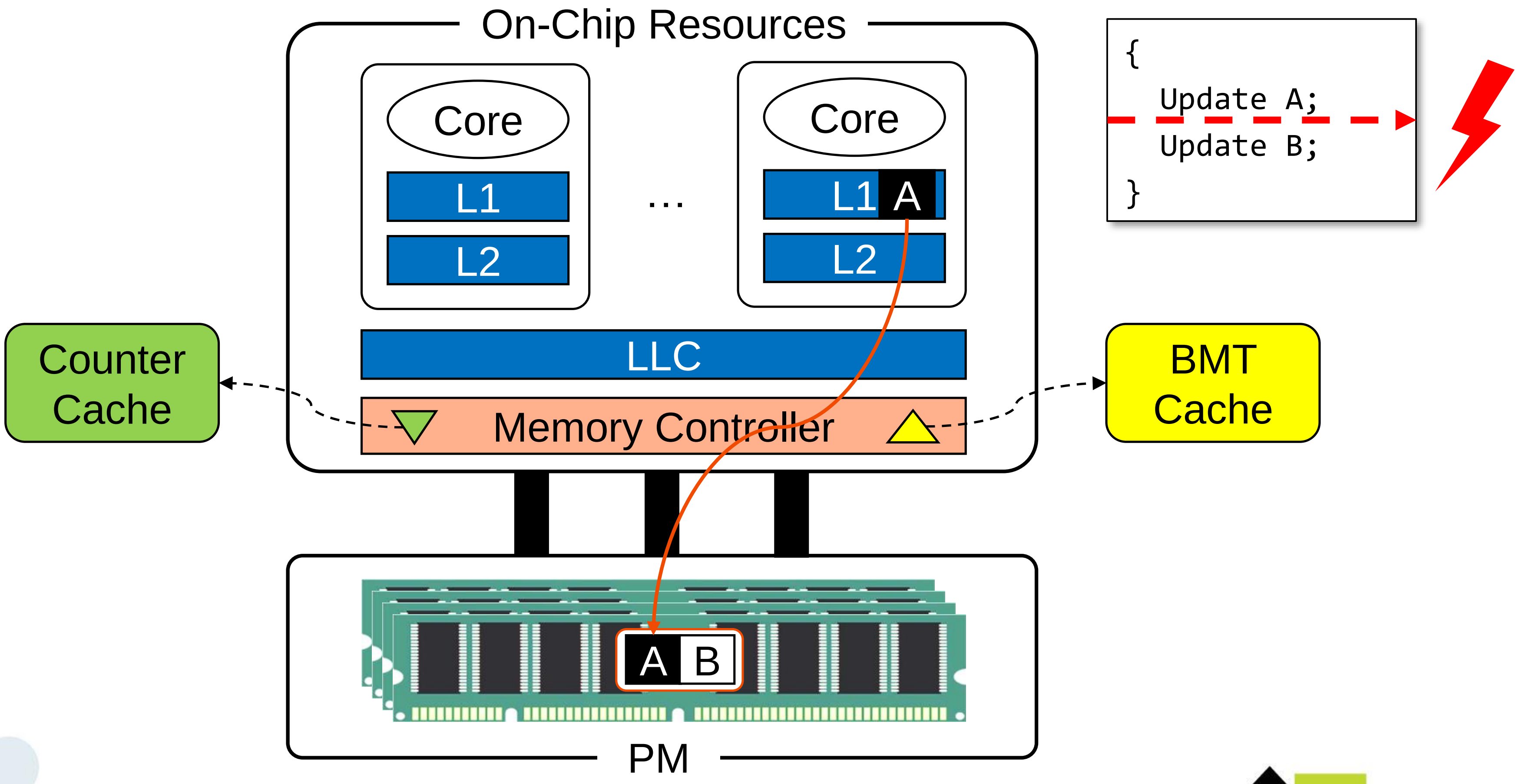
Crash Consistency for Secure PM





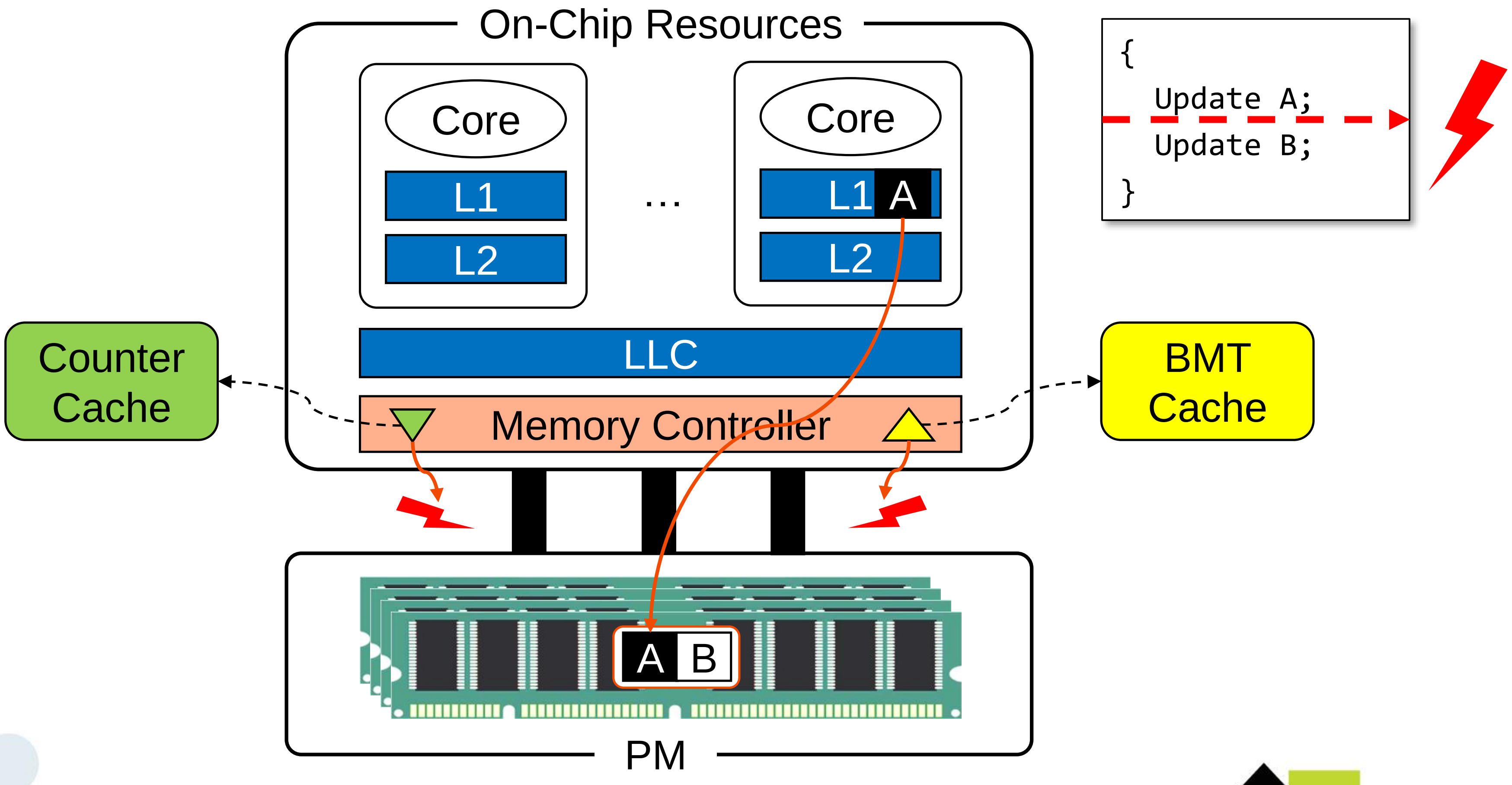
```
{
   Update A;
   Update B;
}
```

Crash Consistency for Secure PM



```
{
  Update A;
  Update B;
}
```

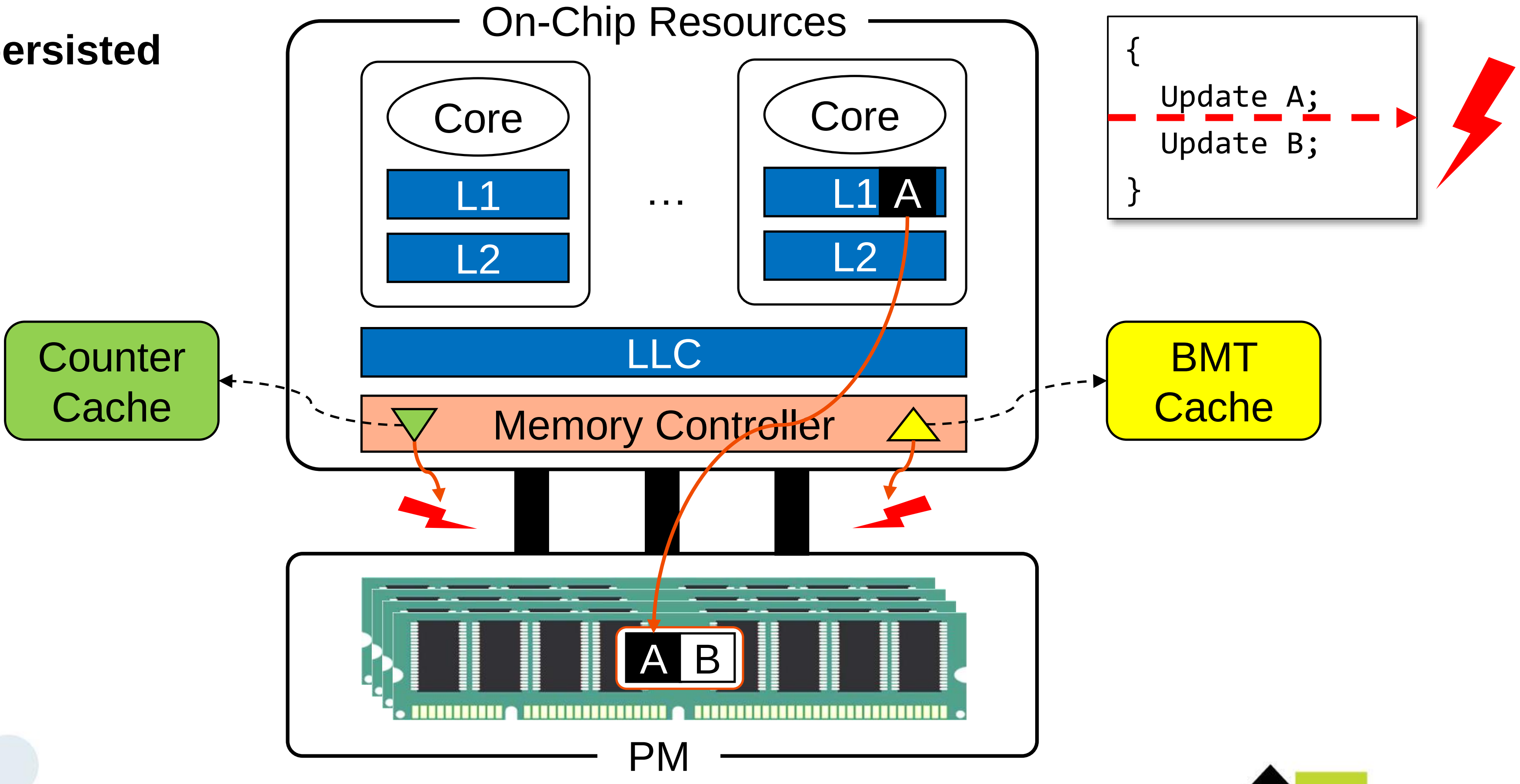
Crash Consistency for Secure PM



```
{
  Update A;
  Update B;
}
```

Crash Consistency for Secure PM

1) Only data is persisted

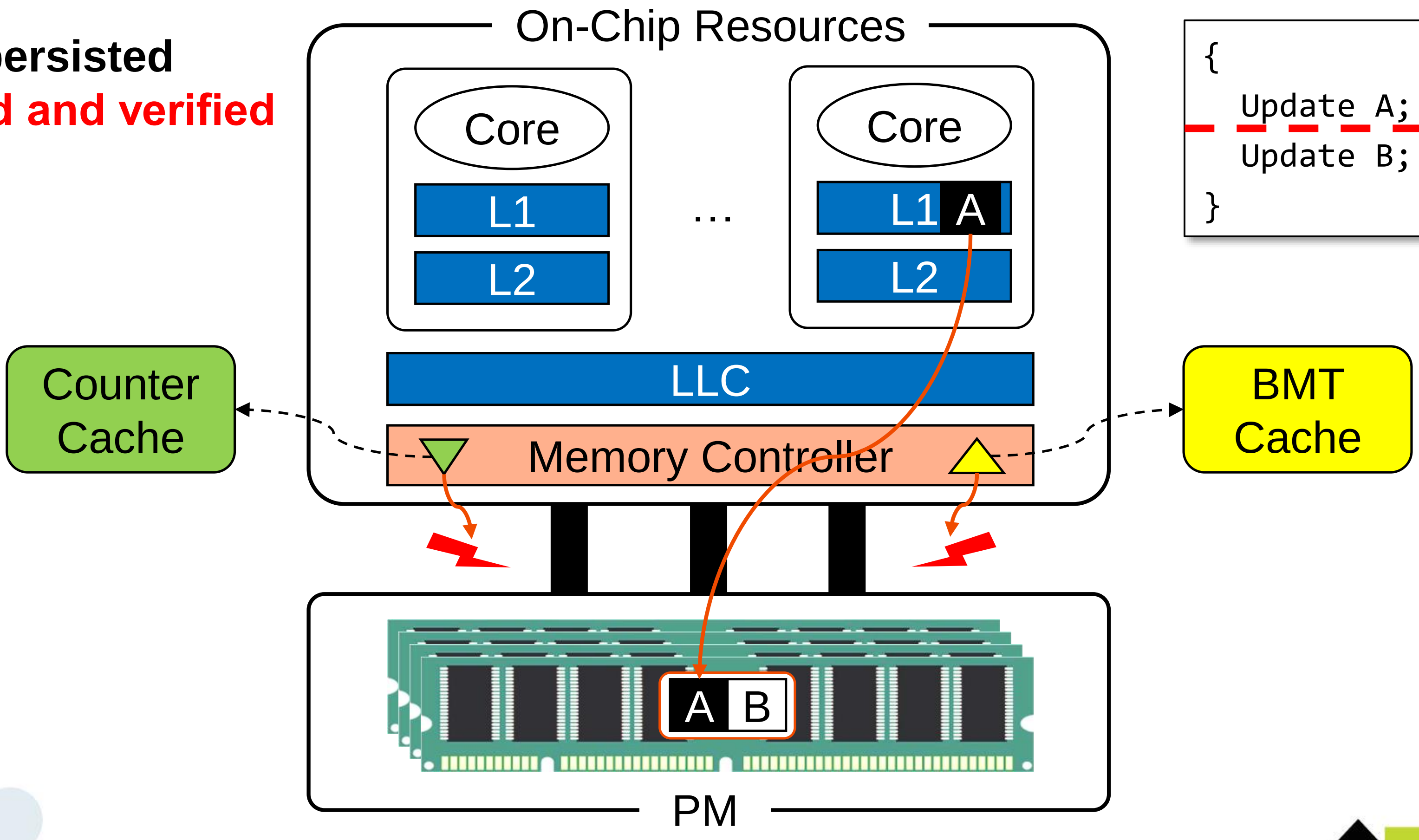


```
{
  Update A;
  Update B;
}
```

Crash Consistency for Secure PM

1) Only data is persisted

Can’t be decrypted and verified



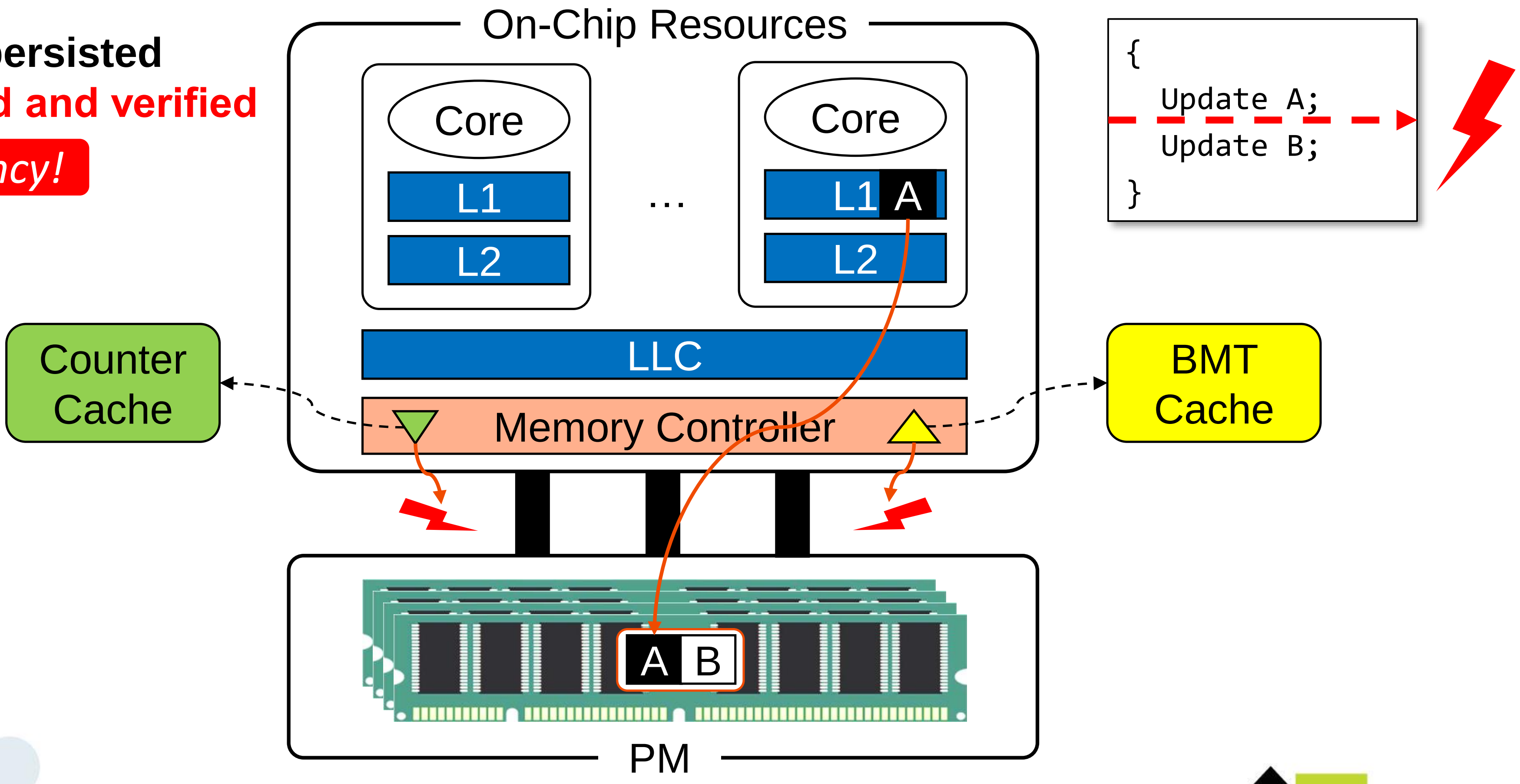
```
{
   Update A;
   Update B;
}
```

Crash Consistency for Secure PM

1) Only data is persisted

Can't be decrypted and verified

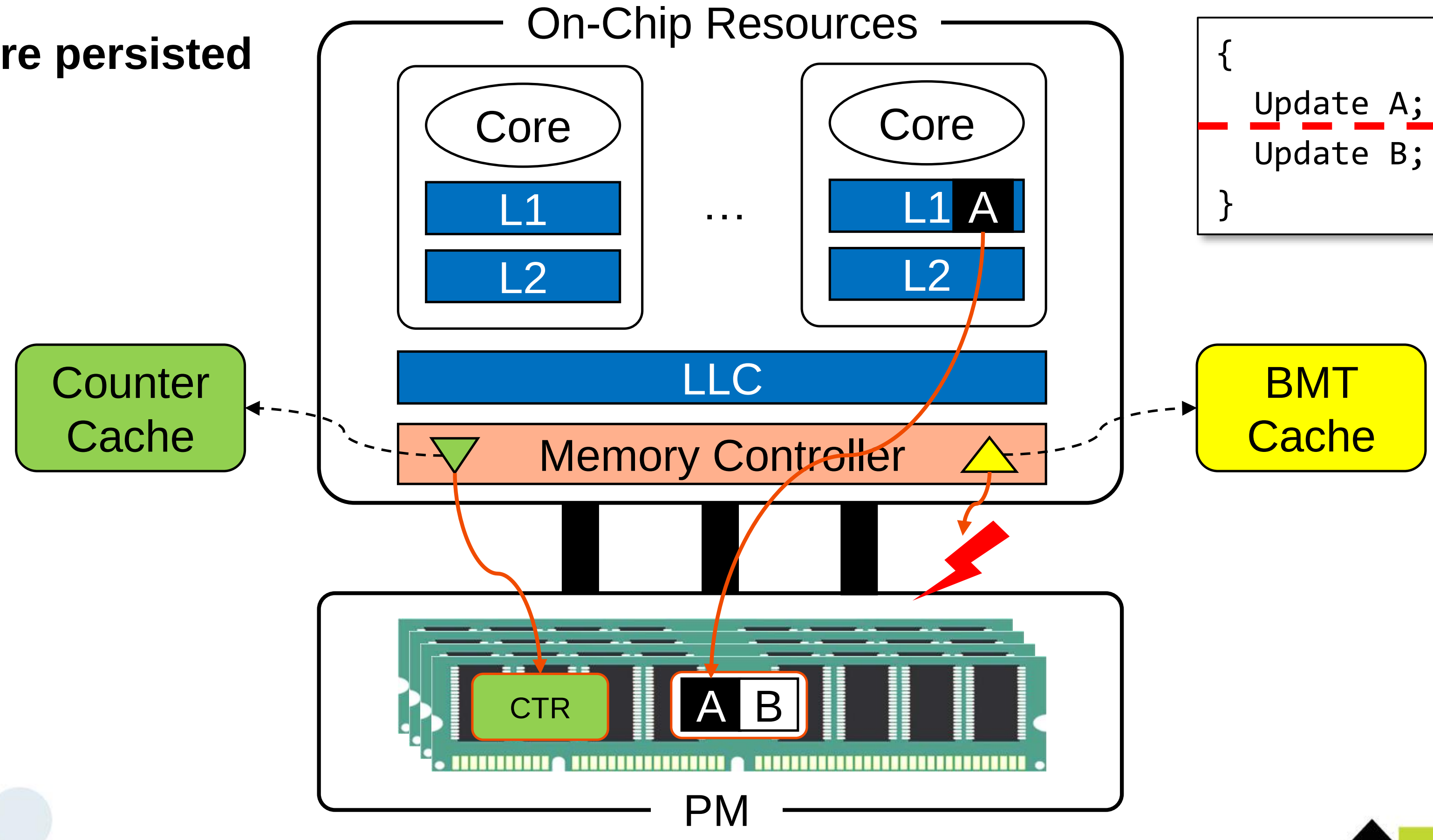
Inconsistency!

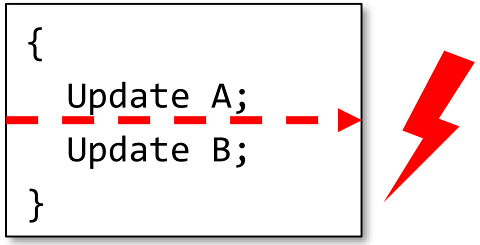


```
{
  Update A;
  Update B;
}
```

Crash Consistency for Secure PM

2) Data + counter are persisted



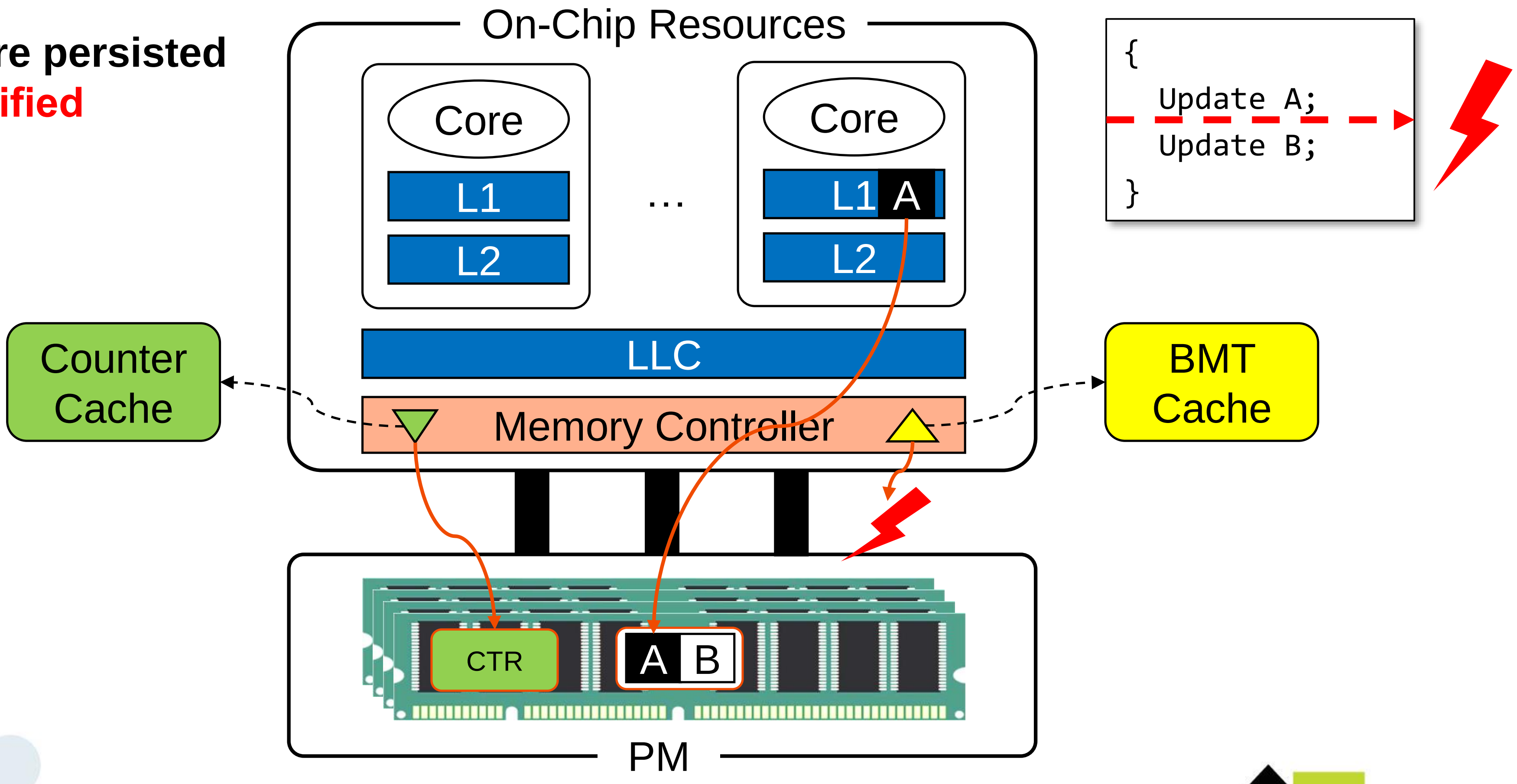


```
{
   Update A;
   Update B;
}
```

Crash Consistency for Secure PM

2) Data + counter are persisted

Can’t be verified



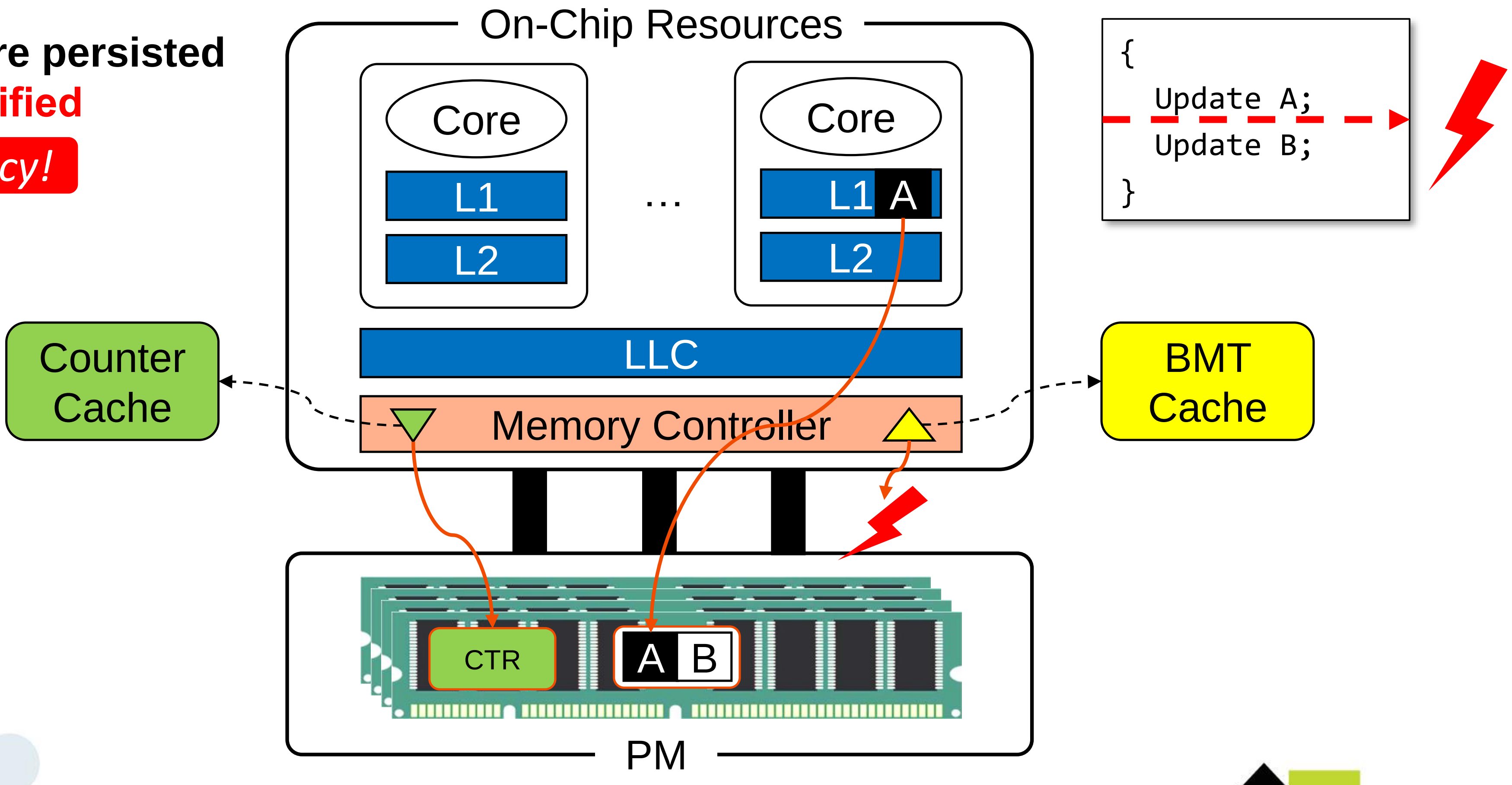
```
{
   Update A;
   Update B;
}
```

Crash Consistency for Secure PM

2) Data + counter are persisted

Can't be verified

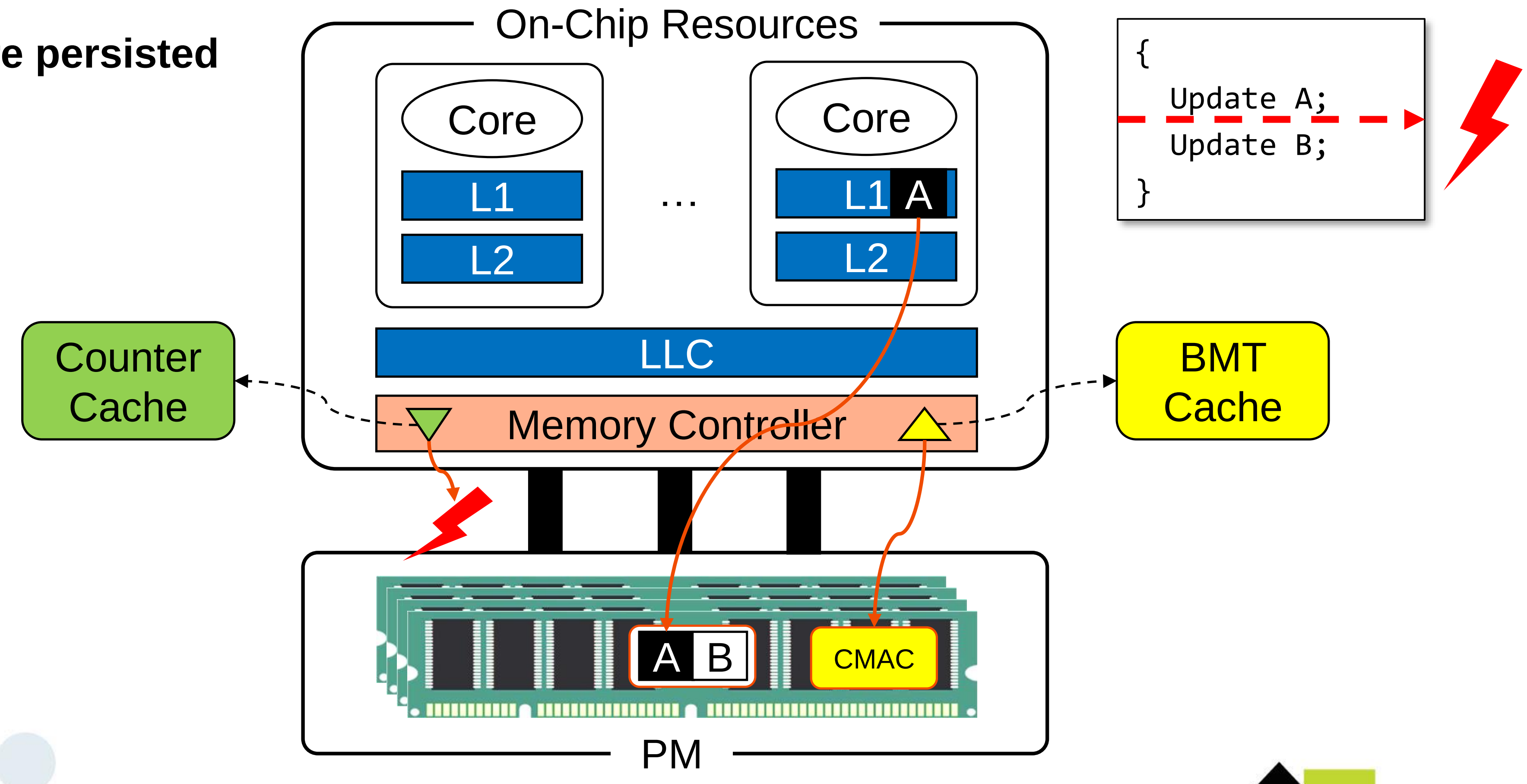
Inconsistency!



```
{
   Update A;
   Update B;
}
```

Crash Consistency for Secure PM

3) Data + CMAC are persisted

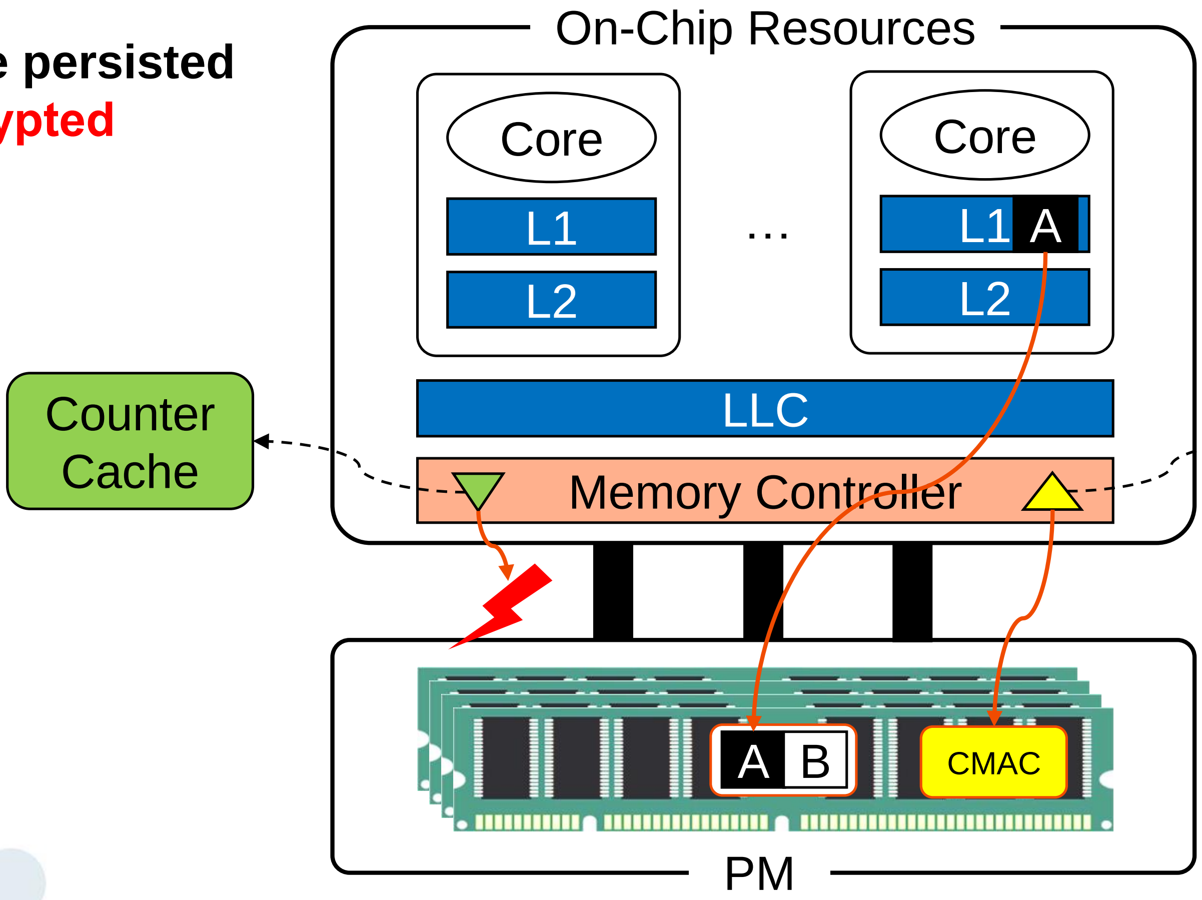


```
{
  Update A;
  Update B;
}
```

Crash Consistency for Secure PM

3) Data + CMAC are persisted

Can’t be decrypted



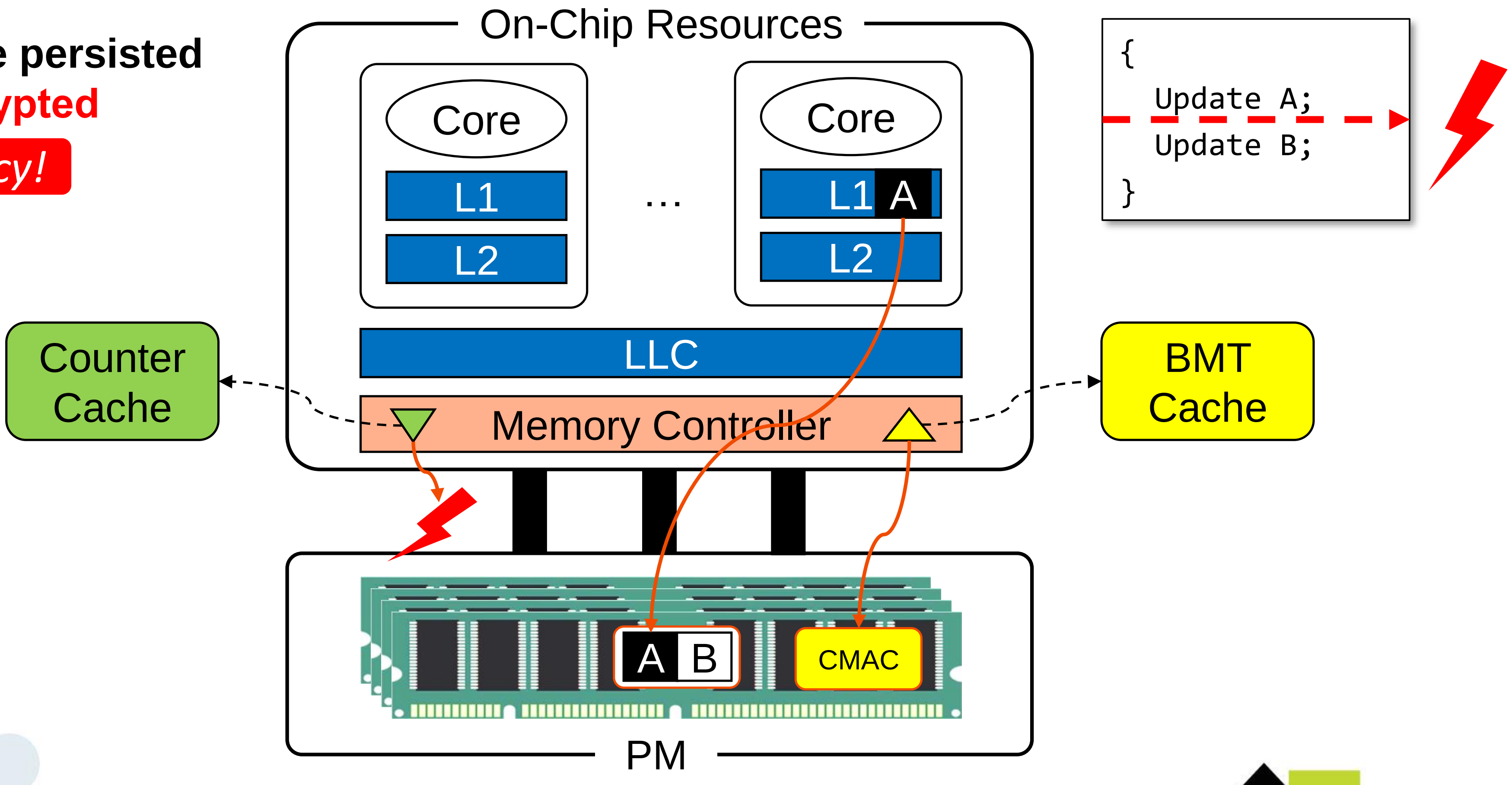
```
{
   Update A;
   Update B;
}
```

Crash Consistency for Secure PM

3) Data + CMAC are persisted

Can't be decrypted

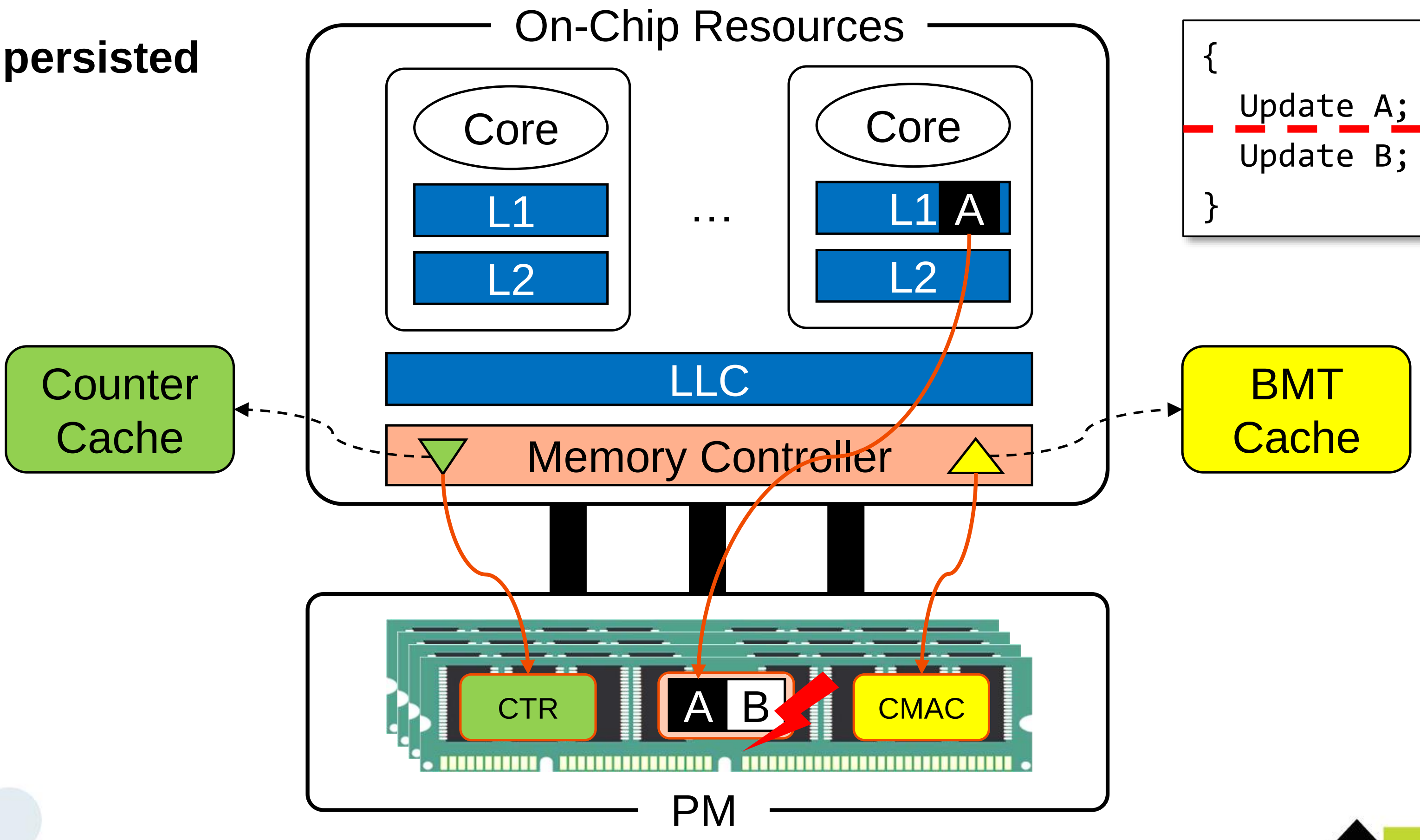
Inconsistency!



```
{
  Update A;
  Update B;
}
```

Crash Consistency for Secure PM

4) Part of data are persisted

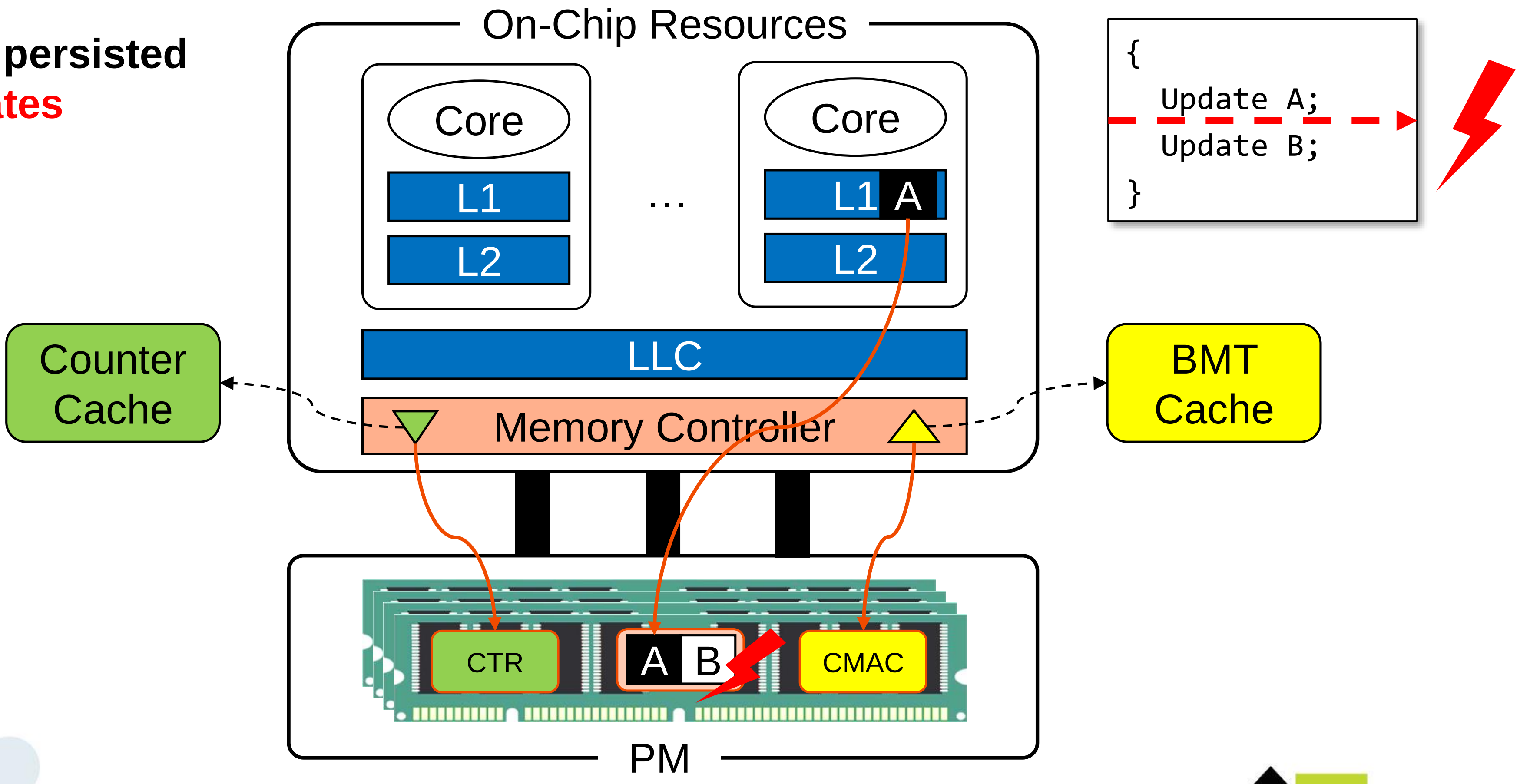


```
{
  Update A;
  Update B;
}
```

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates



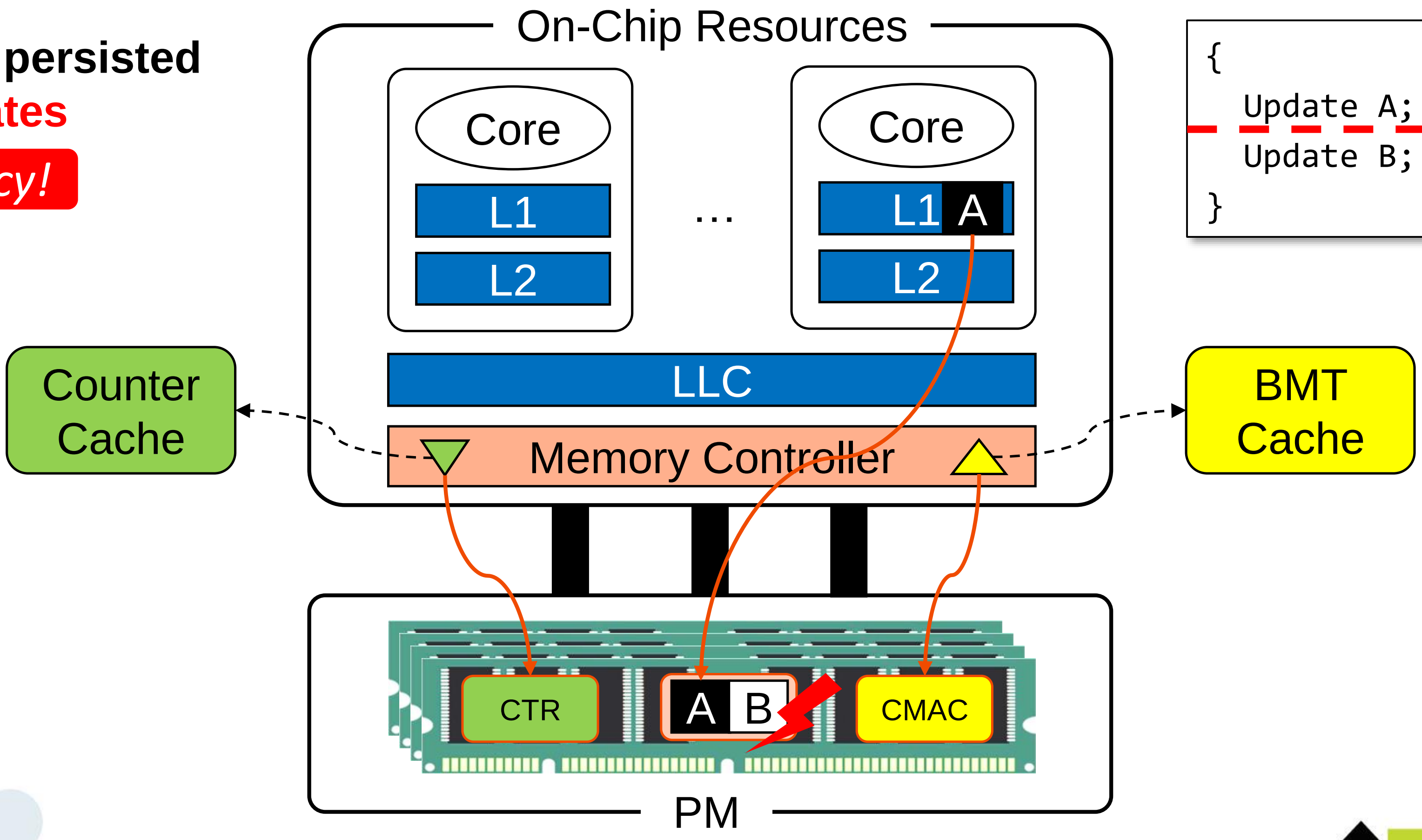
```
{
   Update A;
   Update B;
}
```

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



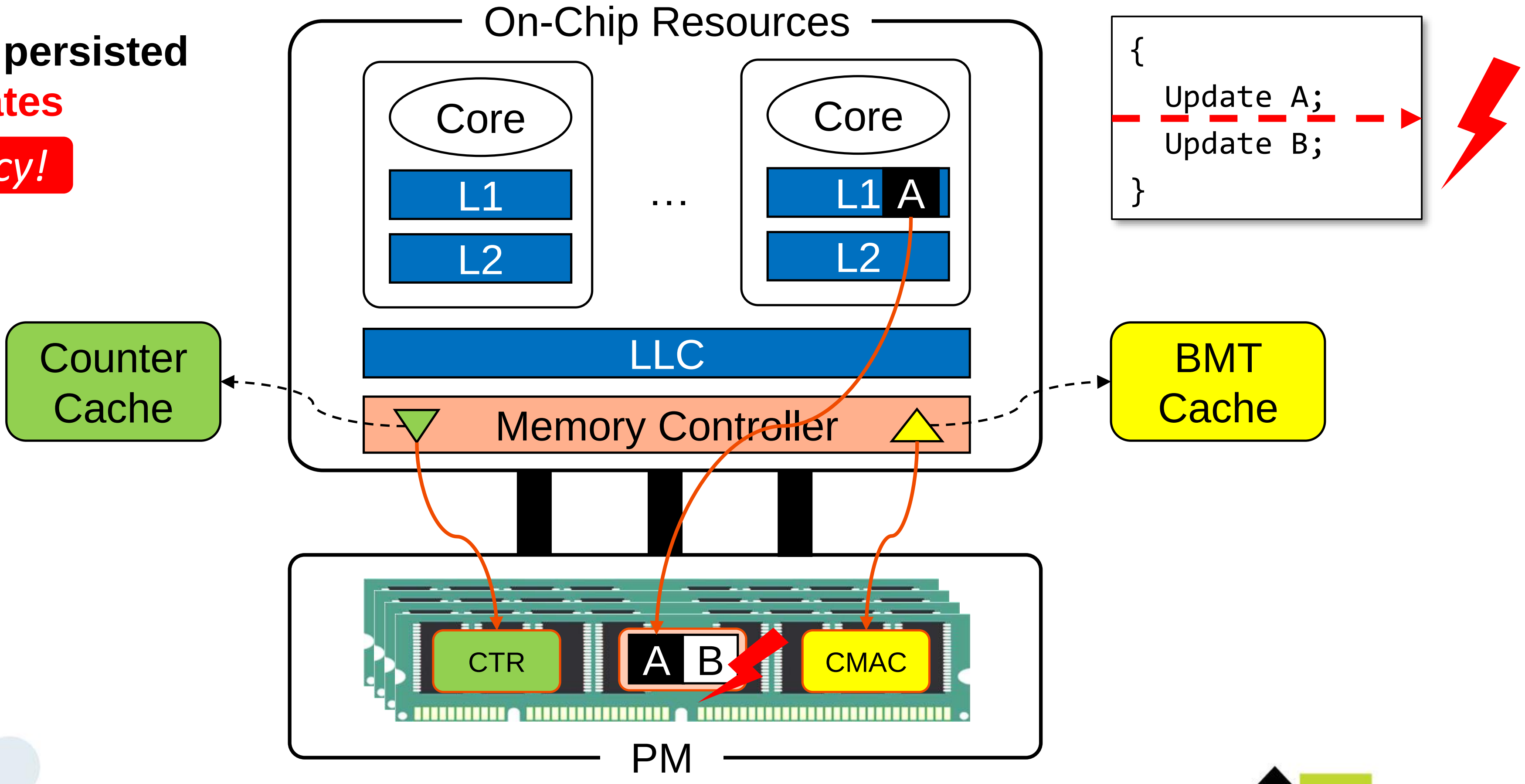
```
{
  Update A;
  Update B;
}
```

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



```
{
  Update A;
  Update B;
}
```

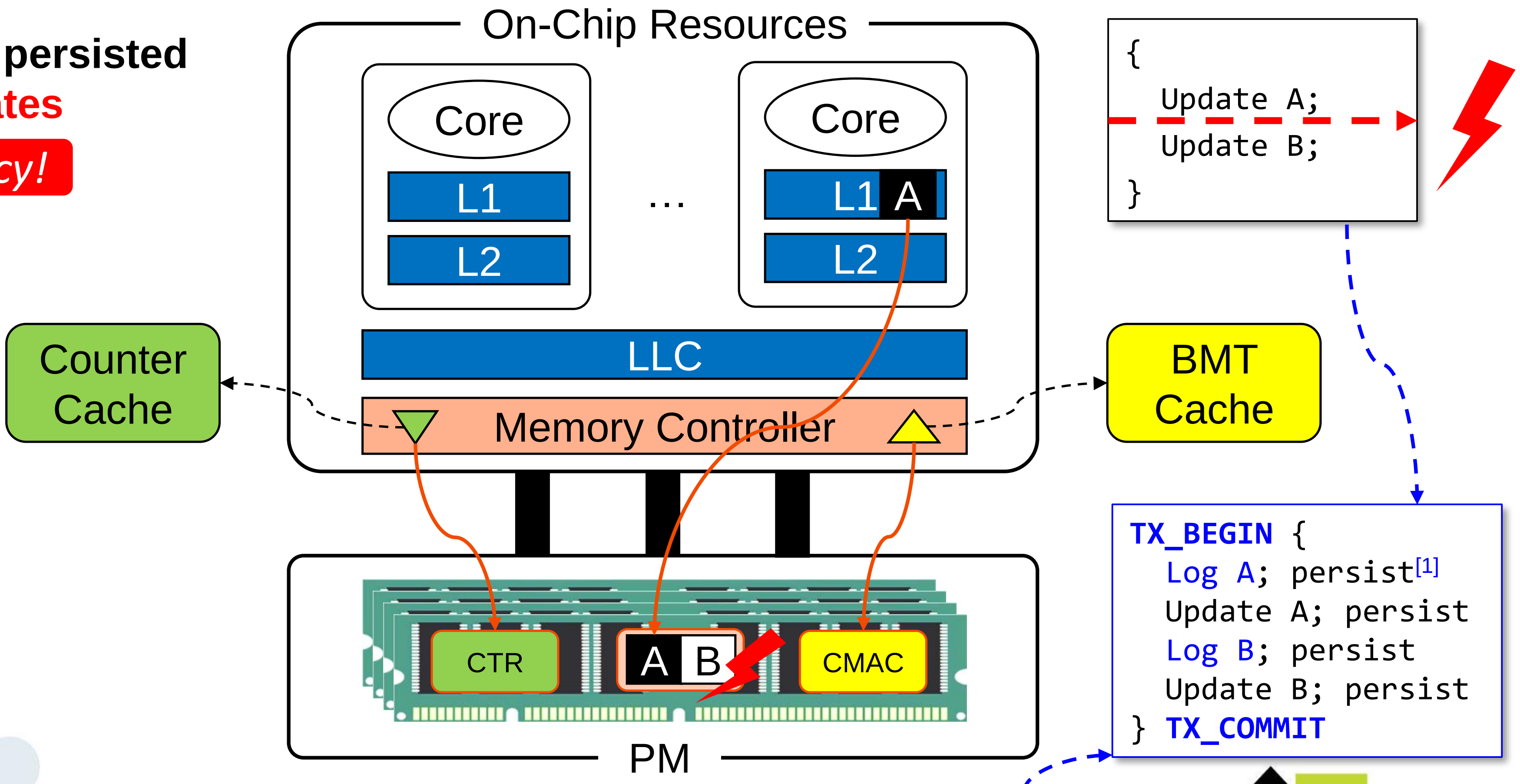
Guarantee failure-atomicity for
- ✓ A group of data
- ✓ Data + counter + CMAC

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



```
{
  Update A;
  Update B;
}
```

```
TX_BEGIN {
  Log A; persist[1]
  Update A; persist
  Log B; persist
  Update B; persist
} TX_COMMIT
```

Durable Transaction with write-ahead logging

Guarantee failure-atomicity for
- ✓ A group of data
- ✓ Data + counter + CMAC

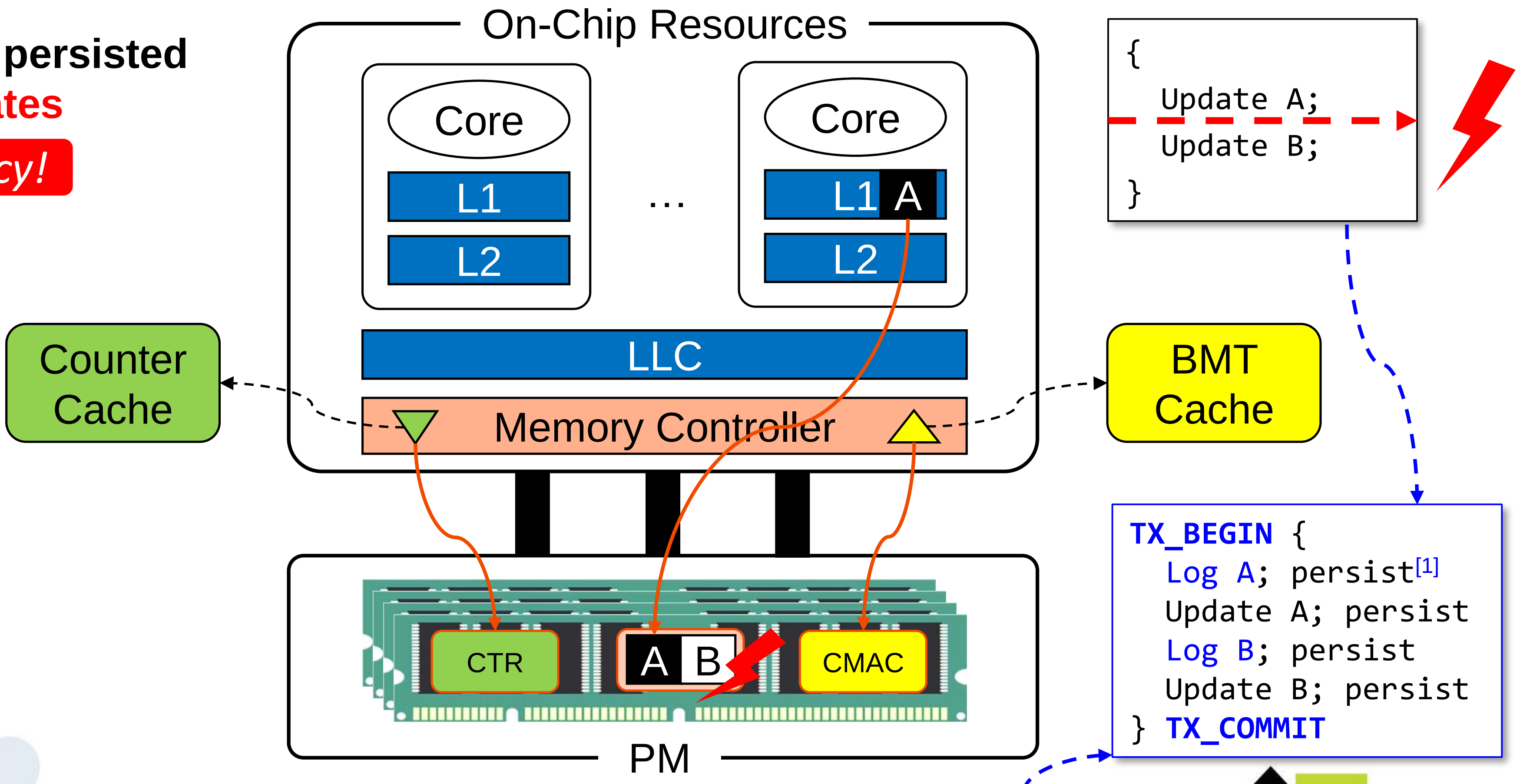
[1] Persist instruction sequence, e.g., clwb + sfence

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



```
{
  Update A;
  Update B;
}
```

```
TX_BEGIN {
  Log A; persist[1]
  Update A; persist
  Log B; persist
  Update B; persist
} TX_COMMIT
```

Durable Transaction with write-ahead logging

Guarantee failure-atomicity for [
- ✓ A group of data
- ✓ Data + counter + CMAC **?**

[1] Persist instruction sequence, e.g., clwb + sfence

State-of-The-Art

Design	Confidentiality	Integrity	Atomicity for a group of updates	Atomicity of data and its security metadata
SCA@HPCA’18	✓	✘	✓	Data + Counter
SuperMem@MICRO’19	✓	✘	✓	Data + Counter

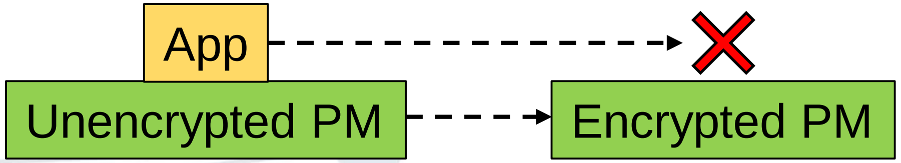
State-of-The-Art

Design	Confidentiality	Integrity	Atomicity for a group of updates	Atomicity of data and its security metadata
SCA@HPCA'18	✓	✗	✓	Data + Counter
SuperMem@MICRO'19	✓	✗	✓	Data + Counter

SCA@HPCA'18

- Write-back counter cache
- New primitives required
  - CounterAtomicity
  - counter_cache_writeback()

➔ Limited portability

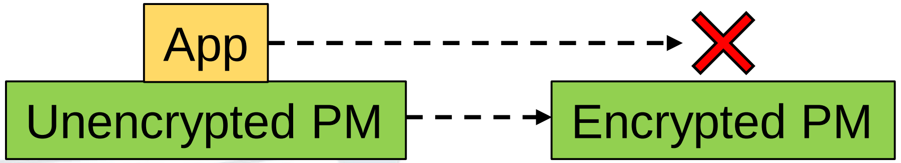


State-of-The-Art

Design	Confidentiality	Integrity	Atomicity for a group of updates	Atomicity of data and its security metadata
SCA@HPCA'18	✓	✗	✓	Data + Counter
SuperMem@MICRO'19	✓	✗	✓	Data + Counter

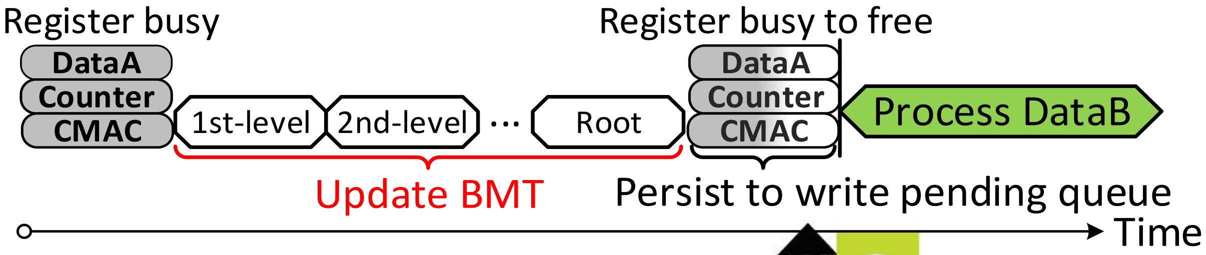
SCA@HPCA'18

- ➢ Write-back counter cache
- ➢ New primitives required
  - CounterAtomicity
  - counter_cache_writeback()
- ➔ Limited portability



SuperMem@MICRO'19

- ➢ Write-through counter cache
  - Application transparent → Good portability
- ➢ A register appends <data+counter> to write queue
- ➔ Limited scalability

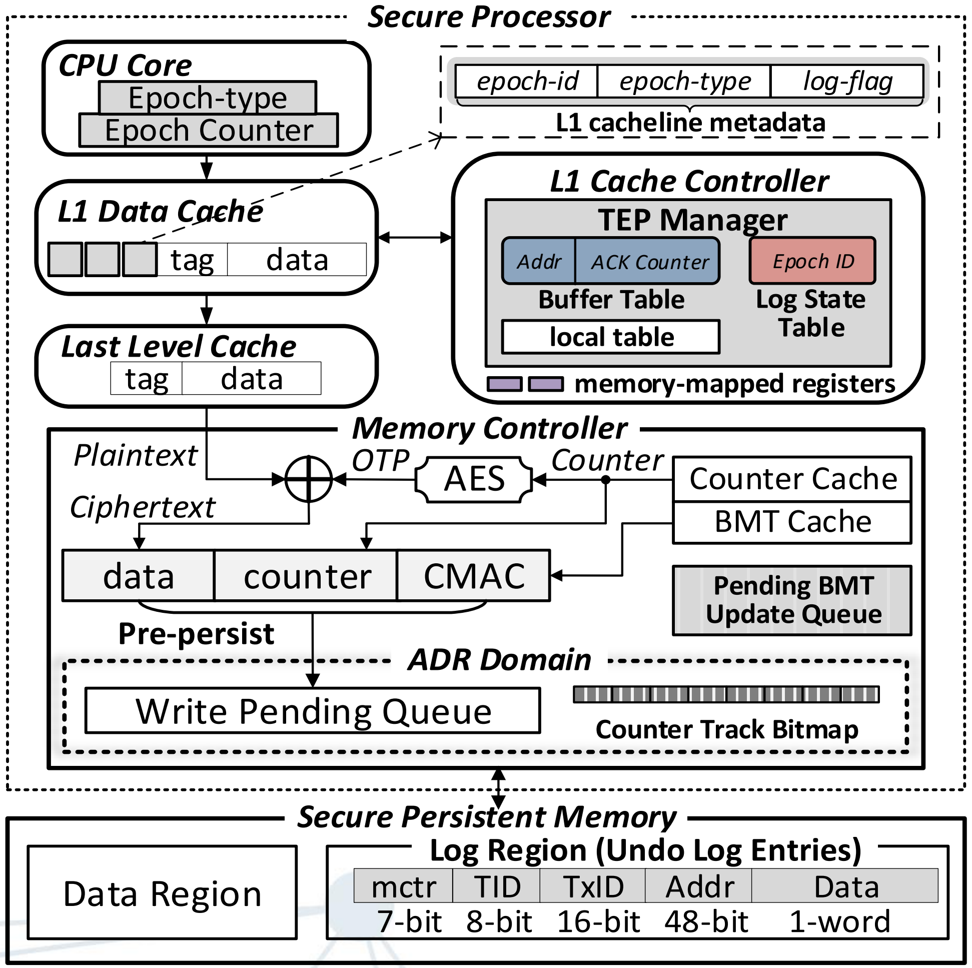


Secon: Security and crash consistency for PM

- Goal

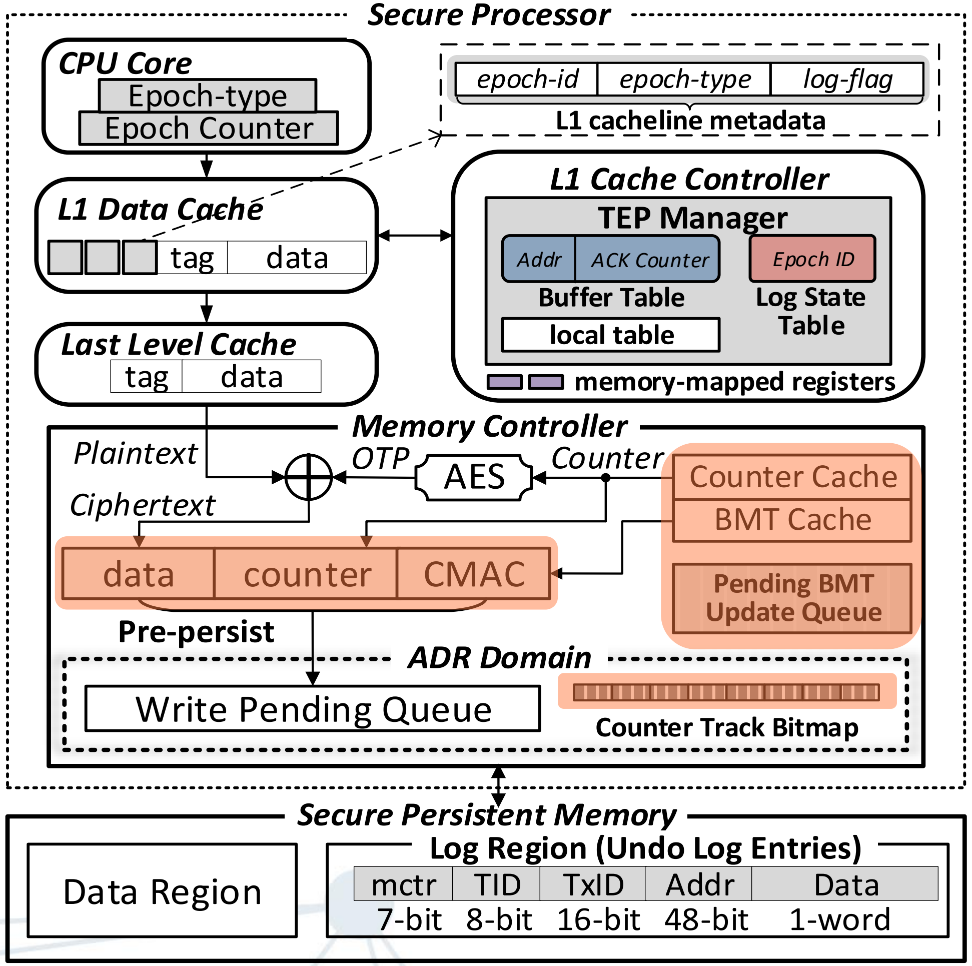
Design	Confidentiality	Integrity	Atomicity for a group of updates	Atomicity of data and its security metadata
SCA@HPCA'18	✓	✗	✓	Data + Counter
SuperMem@MICRO'19	✓	✗	✓	Data + Counter
Our Secon	✓	✓	✓	**Data + Counter + CMAC**

Architecture



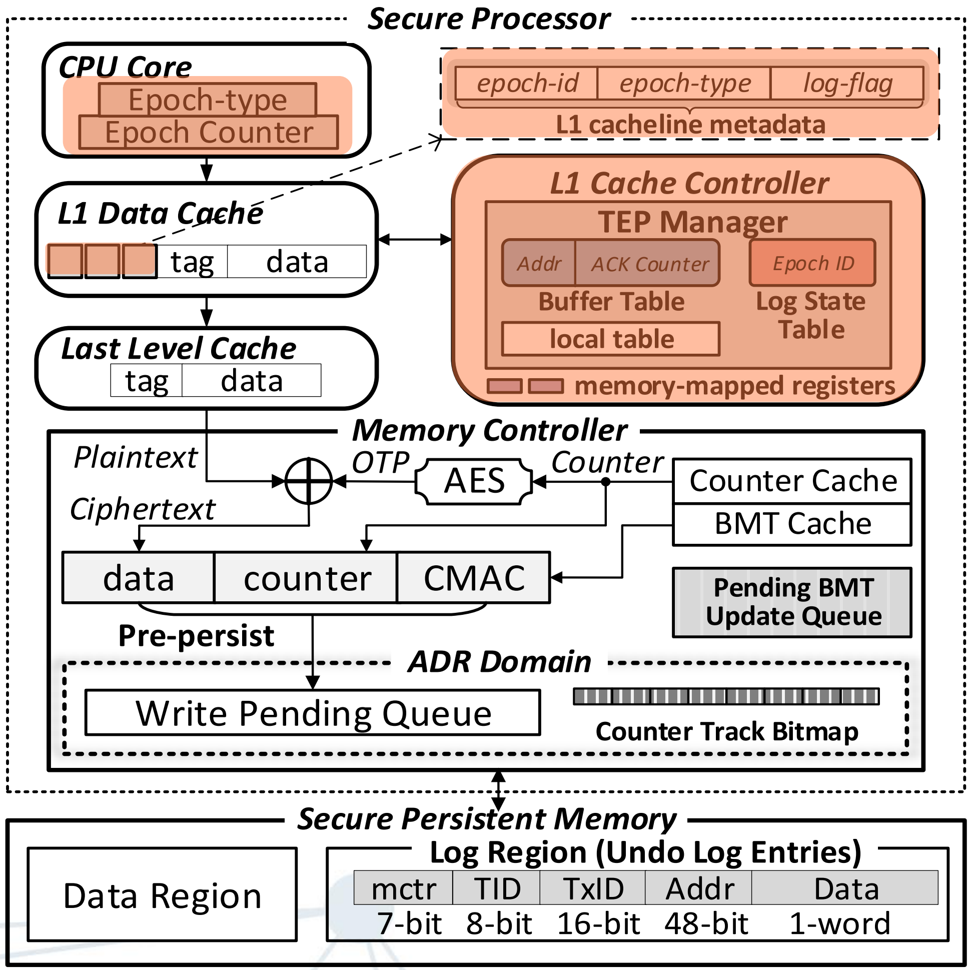
- **Scalable write-through security metadata cache**
  - Move BMT update to the background
- **Transaction-specific epoch persistency model**
  - Minimize ordering constraints between logs and data
- **Security metadata write-reduction schemes**
  - Mitigate the writes caused by counters and CMACs

Architecture



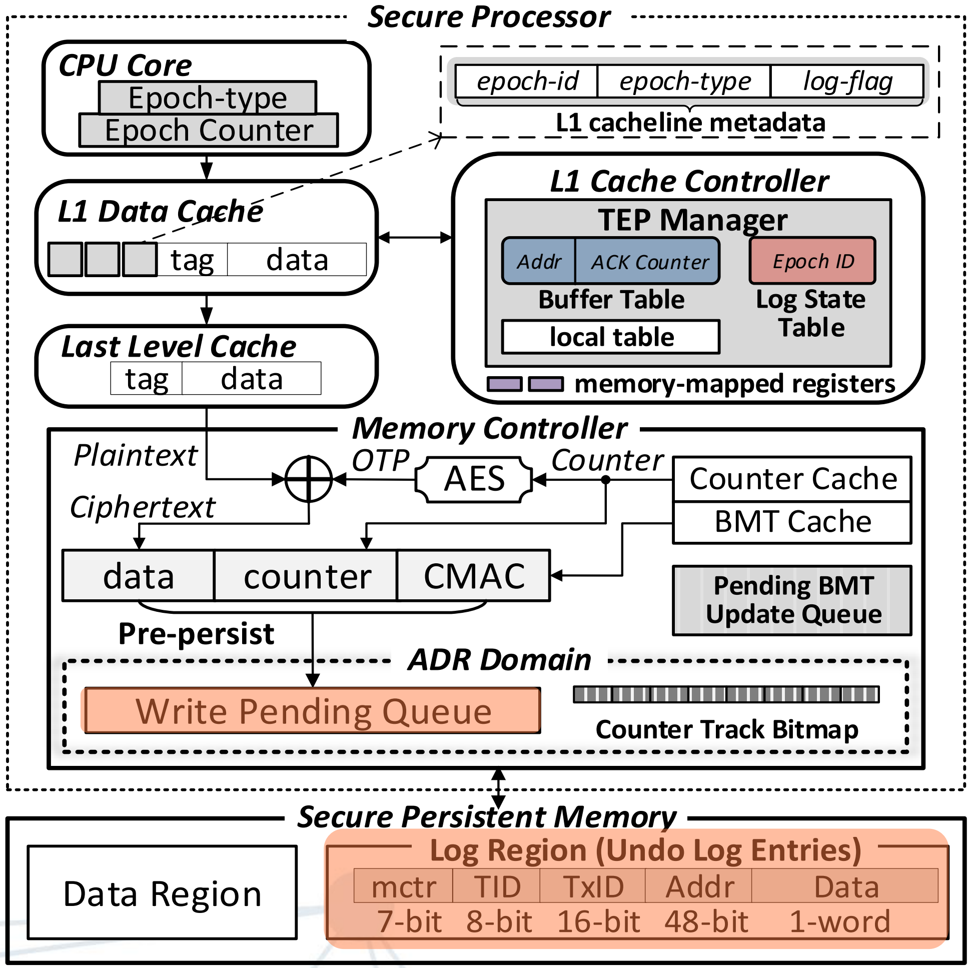
- **Scalable write-through security metadata cache**
  - Move BMT update to the background
- **Transaction-specific epoch persistency model**
  - Minimize ordering constraints between logs and data
- **Security metadata write-reduction schemes**
  - Mitigate the writes caused by counters and CMACs

Architecture



- Scalable write-through security metadata cache
  - Move BMT update to the background
- **Transaction-specific epoch persistency model**
  - Minimize ordering constraints between logs and data
- Security metadata write-reduction schemes
  - Mitigate the writes caused by counters and CMACs

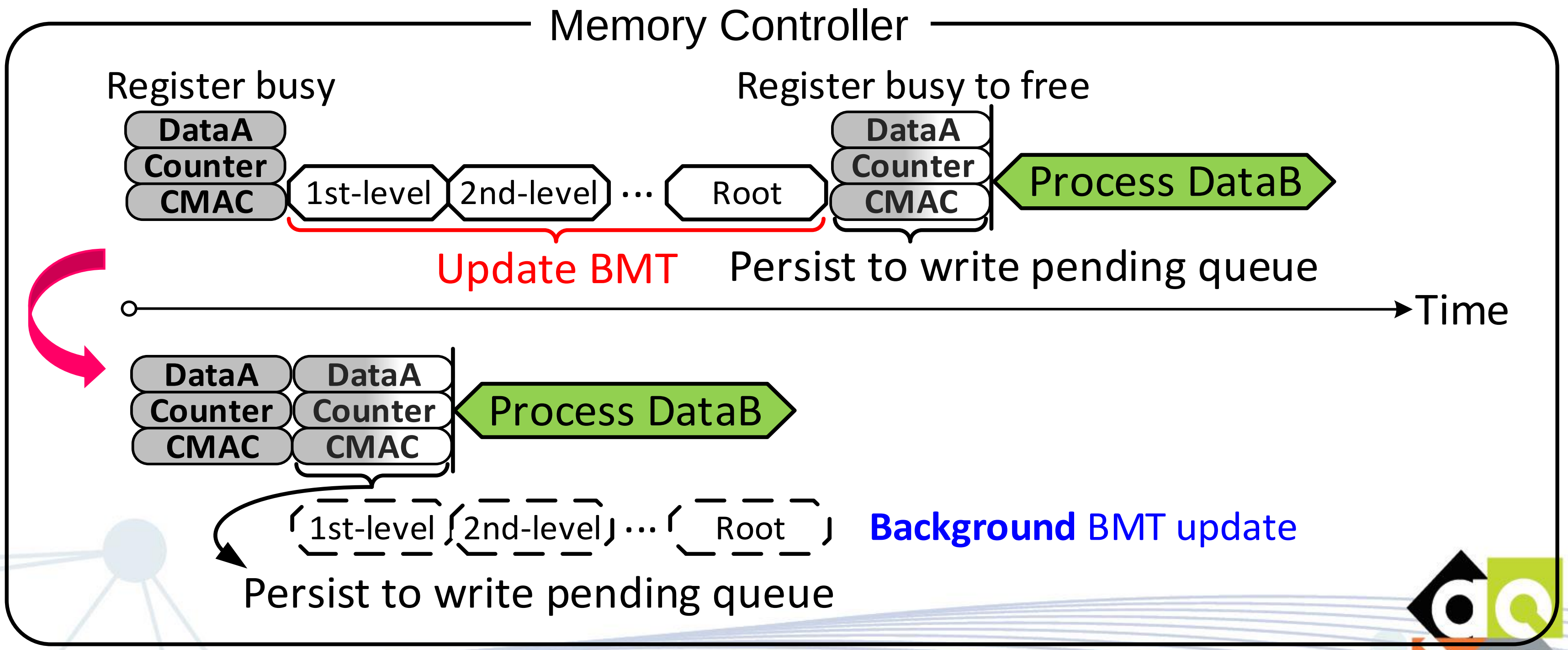
Architecture



- Scalable write-through security metadata cache
  - Move BMT update to the background
- Transaction-specific epoch persistency model
  - Minimize ordering constraints between logs and data
- **Security metadata write-reduction schemes**
  - Mitigate the writes caused by counters and CMACs

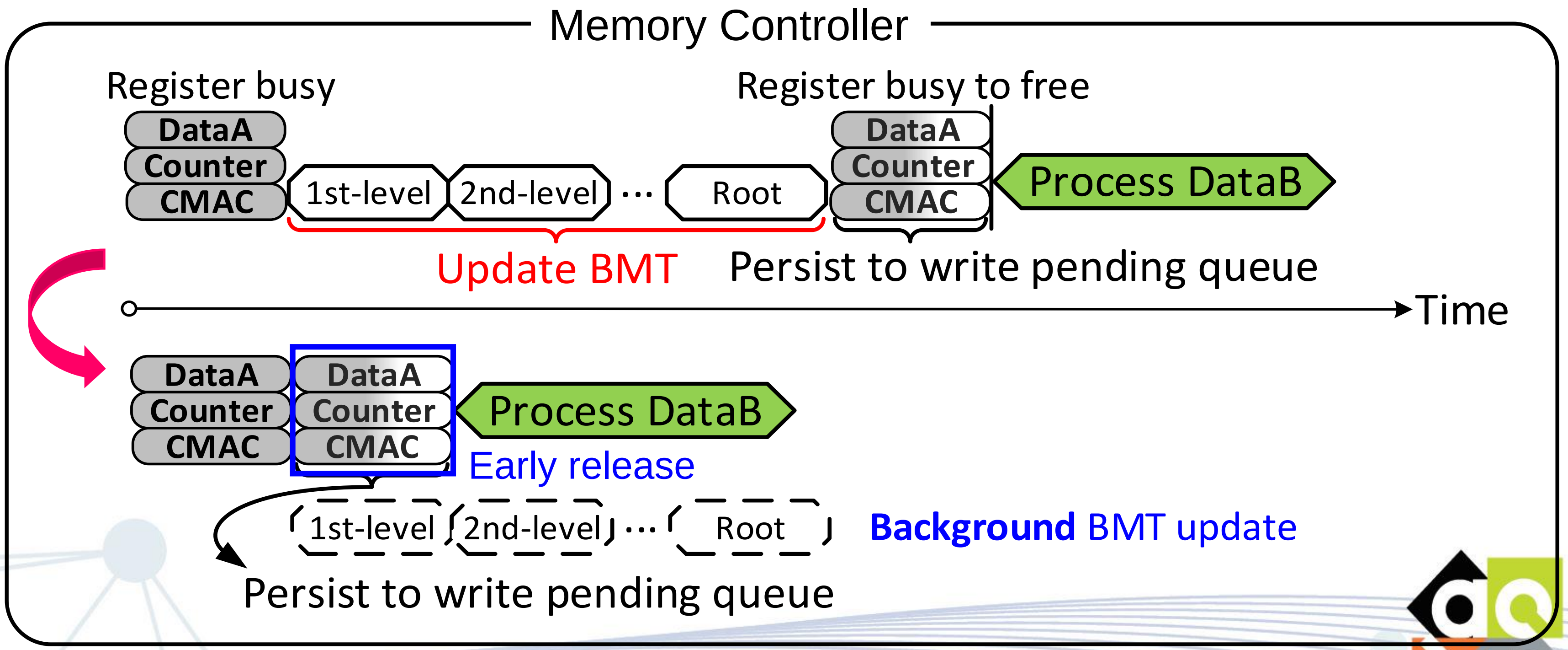
Scalable Write-Through Security Metadata Cache

- **Insight:** PM always has a consistent data view by logging
  - In the log region or data region
- Move BMT update to the background
  - Release the register early to process the next independent write request



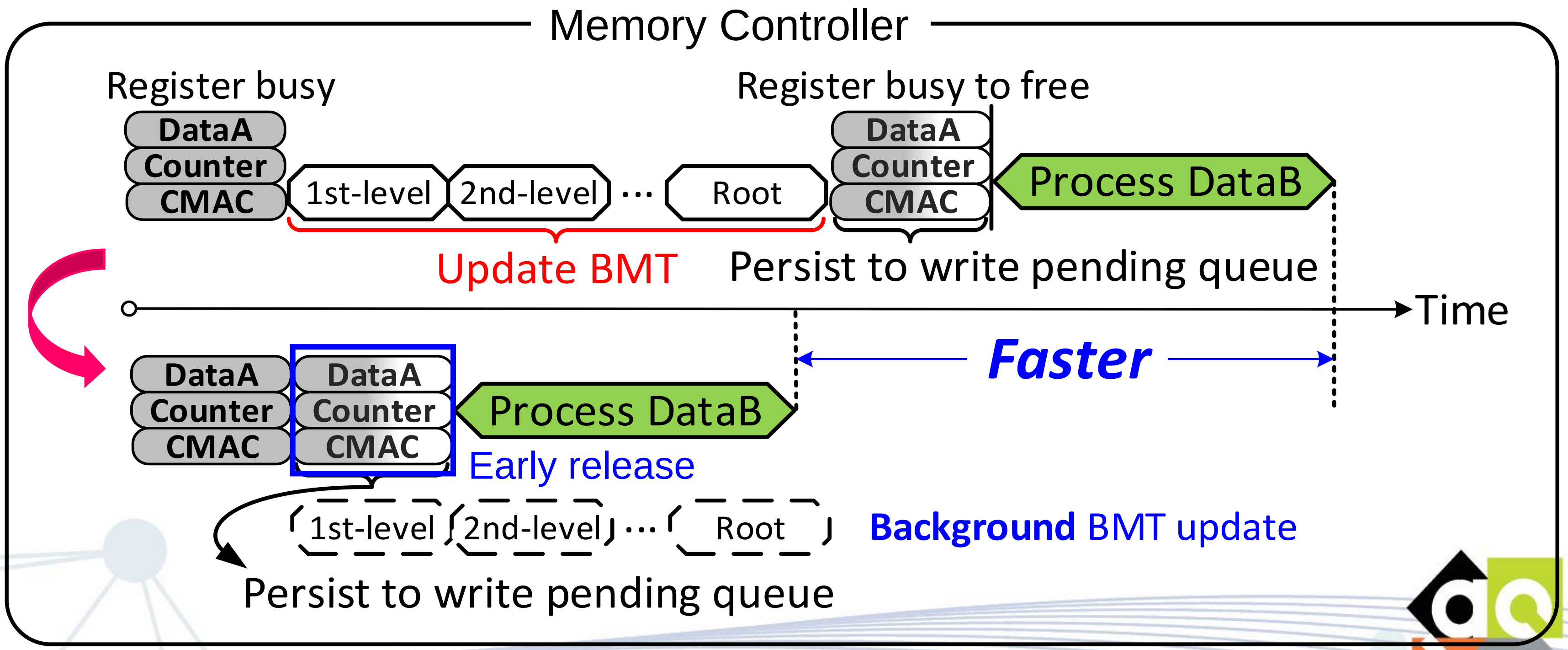
Scalable Write-Through Security Metadata Cache

- **Insight:** PM always has a consistent data view by logging
  - In the log region or data region
- Move BMT update to the background
  - Release the register early to process the next independent write request



Scalable Write-Through Security Metadata Cache

- **Insight:** PM always has a consistent data view by logging
  - In the log region or data region
- Move BMT update to the background
  - Release the register early to process the next independent write request



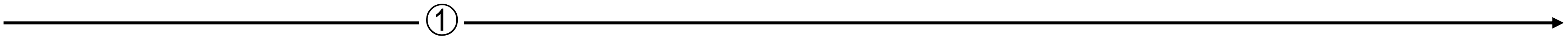
Scalable Write-Through Security Metadata Cache

- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
    - Pending BMT update queue (In MC)
    - Counter track bitmap (In ADR[1] of MC)

[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

Scalable Write-Through Security Metadata Cache

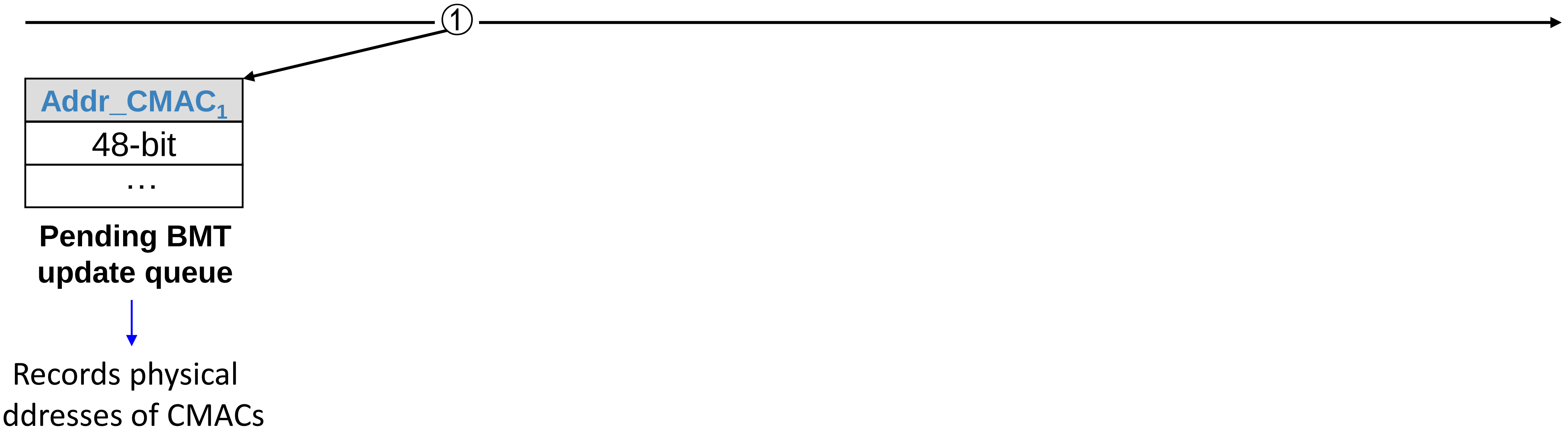
- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
  - Pending BMT update queue (In MC)
  - Counter track bitmap (In ADR[1] of MC)



[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

Scalable Write-Through Security Metadata Cache

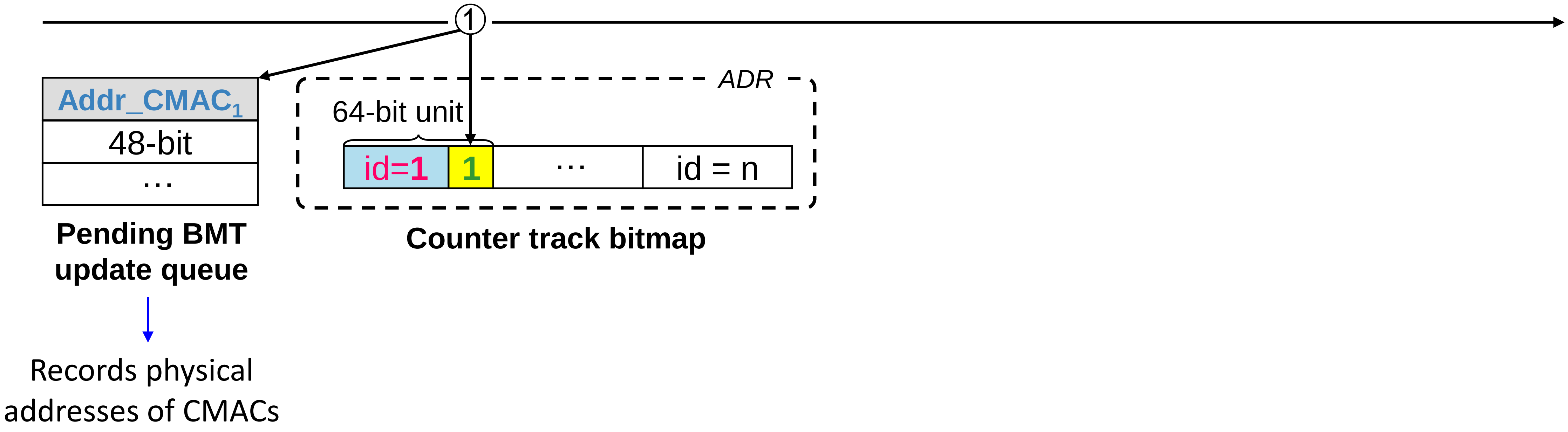
- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
  - Pending BMT update queue (In MC)
  - Counter track bitmap (In ADR[1] of MC)



[1] The **A**synchronous **DR**AM **R**efresh domain, in which the internal data survive a crash or power failure

Scalable Write-Through Security Metadata Cache

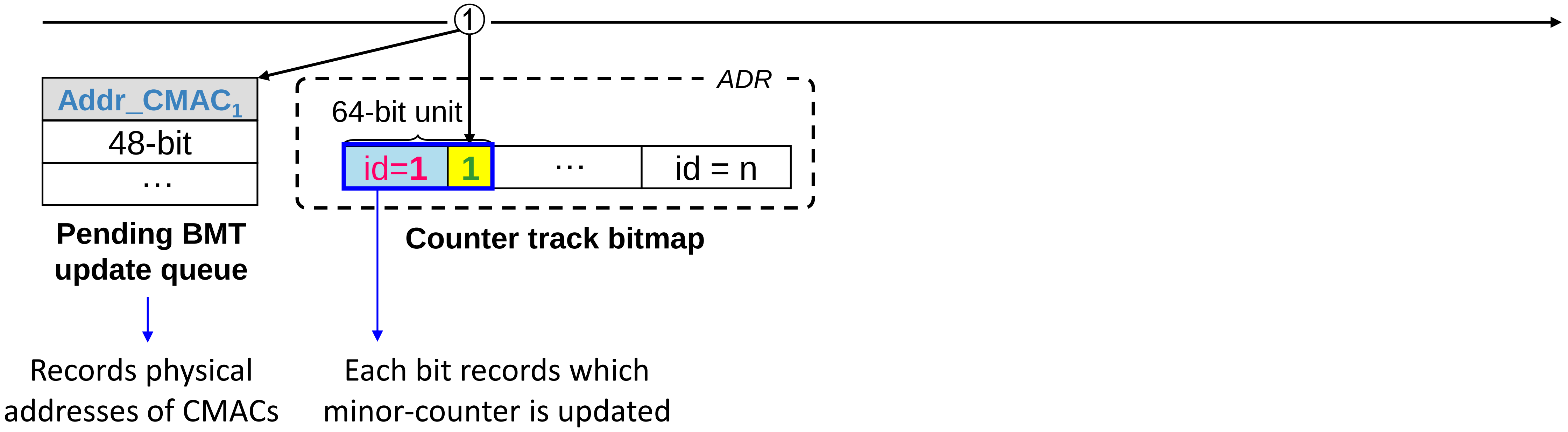
- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
  - Pending BMT update queue (In MC)
  - Counter track bitmap (In ADR[1] of MC)



[1] The Asynchronous DRAM Refresh domain, in which the internal data survive a crash or power failure

Scalable Write-Through Security Metadata Cache

- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
  - Pending BMT update queue (In MC)
  - Counter track bitmap (In ADR[1] of MC)



Pending BMT update queue

Counter track bitmap

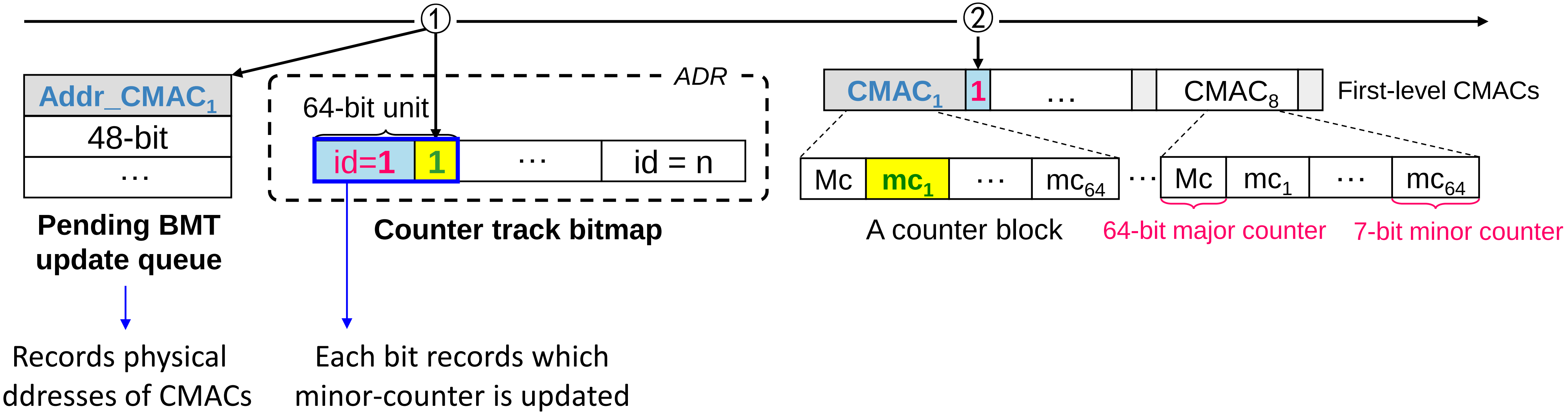
Records physical addresses of CMACs

Each bit records which minor-counter is updated

[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

Scalable Write-Through Security Metadata Cache

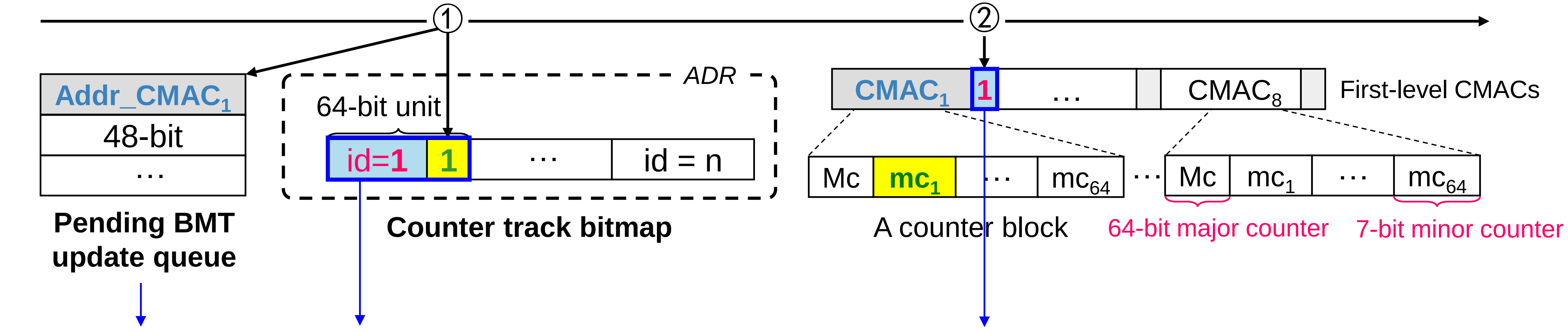
- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
  - Pending BMT update queue (In MC)
  - Counter track bitmap (In ADR[1] of MC)



[1] The Asynchronous DRAM Refresh domain, in which the internal data survive a crash or power failure

Scalable Write-Through Security Metadata Cache

- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
  - Pending BMT update queue (In MC)
  - Counter track bitmap (In ADR[1] of MC)



Pending BMT update queue

Counter track bitmap

A counter block

Records physical addresses of CMACs

Each bit records which minor-counter is updated

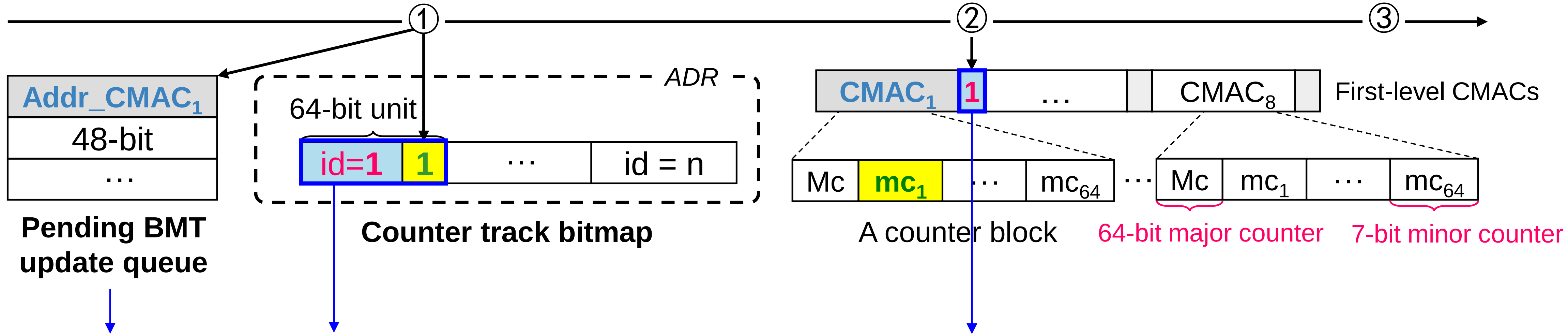
Records the id[2] of the unit in counter track bitmap

[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

[2] Only using 54 bits of a 64-bit CMAC is sufficiently secure (Morphable Counters@MICRO'18)

Scalable Write-Through Security Metadata Cache

- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash
  - Pending BMT update queue (In MC)
  - Counter track bitmap (In ADR[1] of MC)



Pending BMT update queue

Counter track bitmap

A counter block

Records physical addresses of CMACs

Each bit records which minor-counter is updated

Records the id[2] of the unit in counter track bitmap

[1] The **A**synchronous **DR**AM **R**efresh domain, in which the internal data survive a crash or power failure

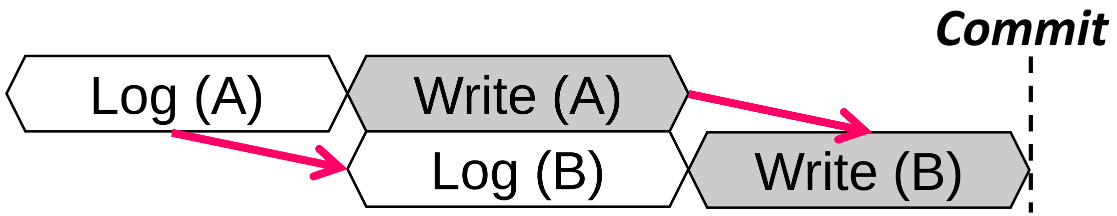
[2] Only using 54 bits of a 64-bit CMAC is sufficiently secure (Morphable Counters@MICRO'18)

Transaction-Specific Epoch Persistency Model

Unnecessary ordering constraints

```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

A dynamic transaction[1]



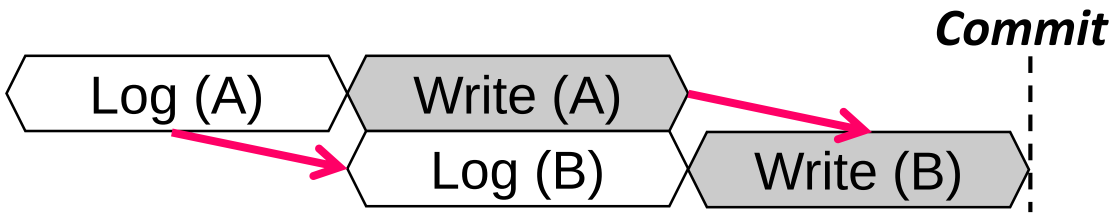
[1] A transaction without pre-defined write set

Transaction-Specific Epoch Persistency Model

Unnecessary ordering constraints

```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

A dynamic transaction[1]



- Log (A) and Log (B) are independent, but ordered

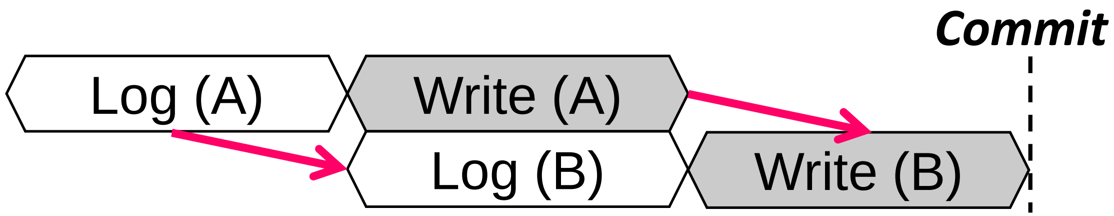
[1] A transaction without pre-defined write set

Transaction-Specific Epoch Persistency Model

Unnecessary ordering constraints

```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

A dynamic transaction[1]



- Log (A) and Log (B) are independent, but ordered
- Write (A) and Write (B) are independent, but ordered

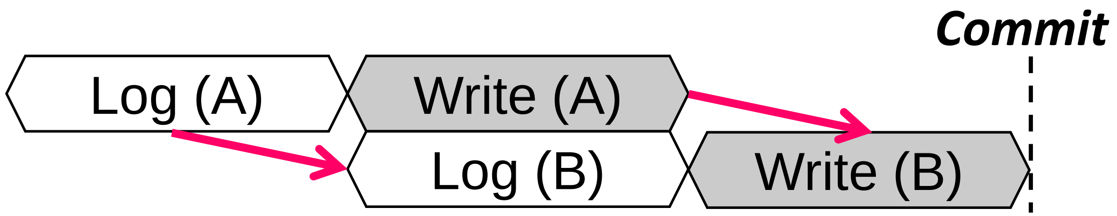
[1] A transaction without pre-defined write set

Transaction-Specific Epoch Persistency Model

Unnecessary ordering constraints

```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

A dynamic transaction[1]



- Log (A) and Log (B) are independent, but ordered
- Write (A) and Write (B) are independent, but ordered

➔ LogB (or DataB) waits for the BMT updates of LogA (or DataA)

[1] A transaction without pre-defined write set

Transaction-Specific Epoch Persistency Model

Epoch Persistency Model [1]

```
TX_BEGIN {
    Log (A)
    clwb (LogA)       epoch 1
    sfence
    Write (A)
    clwb (A)
                      epoch 2
    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)          epoch 3
    sfence
} TX_COMMIT           epoch 4
```

A dynamic transaction

- A program is divided by memory barrier (e.g., sfence)
  - All writes in one epoch are persisted w/o order
  - Different epochs are persisted in order

[1] Memory persistency@ISCA'14

Transaction-Specific Epoch Persistency Model

Epoch Persistency Model [1]

```
TX_BEGIN {
    Log (A)          } epoch 1
    clwb (LogA)      }
    sfence
    Write (A)        }
    clwb (A)         }
                     } epoch 2
    Log (B)          }
    clwb (LogB)      }
    sfence
    Write (B)        } epoch 3
    clwb (B)         }
    sfence
} TX_COMMIT          } epoch 4
```

A dynamic transaction

- A program is divided by memory barrier (e.g., sfence)
  - All writes in one epoch are persisted w/o order
  - Different epochs are persisted in order

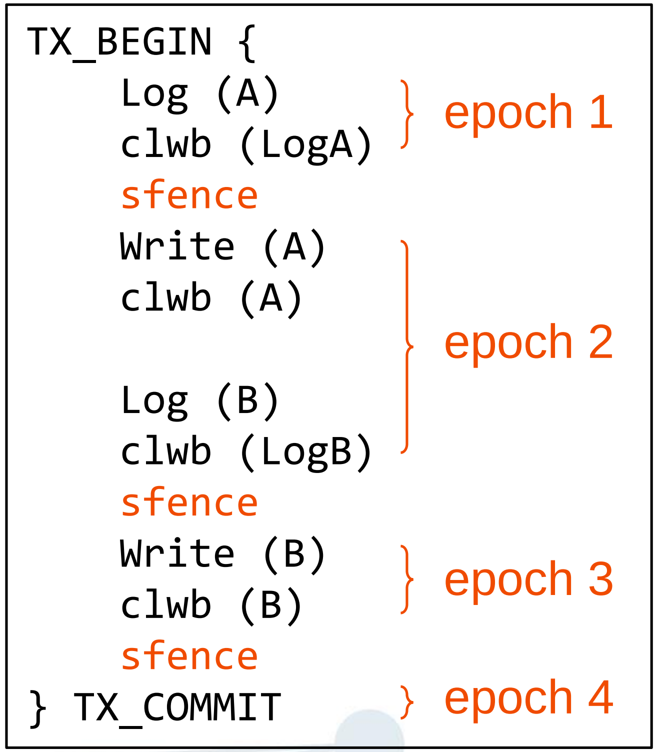
➔ Efficient in static transactions[2] since only one barrier is needed

[1] Memory persistency@ISCA'14

[2] A transaction with pre-defined write set

Transaction-Specific Epoch Persistency Model

Epoch Persistency Model [1]



```
TX_BEGIN {
    Log (A)          } epoch 1
    clwb (LogA)      }
    sfence
    Write (A)        }
    clwb (A)         }
                     } epoch 2
    Log (B)          }
    clwb (LogB)      }
    sfence
    Write (B)        } epoch 3
    clwb (B)         }
    sfence
} TX_COMMIT          } epoch 4
```

A dynamic transaction

- A program is divided by memory barrier (e.g., sfence)
  - All writes in one epoch are persisted w/o order
  - Different epochs are persisted in order

➔ Efficient in static transactions[2] since only one barrier is needed

➔ Inefficient in dynamic transactions due to many barriers

[1] Memory persistency@ISCA'14

[2] A transaction with pre-defined write set

Transaction-Specific Epoch Persistency Model

Epoch Persistency Model [1]

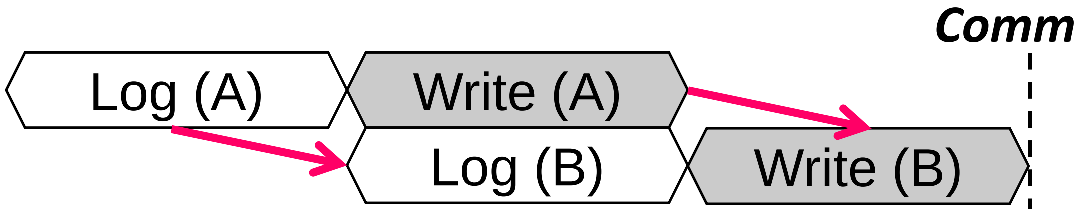
```
TX_BEGIN {
    Log (A)          } epoch 1
    clwb (LogA)      }
    sfence
    Write (A)        }
    clwb (A)         }
                     } epoch 2
    Log (B)          }
    clwb (LogB)      }
    sfence
    Write (B)        } epoch 3
    clwb (B)         }
    sfence
} TX_COMMIT          } epoch 4
```

A dynamic transaction

- A program is divided by memory barrier (e.g., sfence)
  - All writes in one epoch are persisted w/o order
  - Different epochs are persisted in order

➔ Efficient in static transactions[2] since only one barrier is needed

➔ Inefficient in dynamic transactions due to many barriers

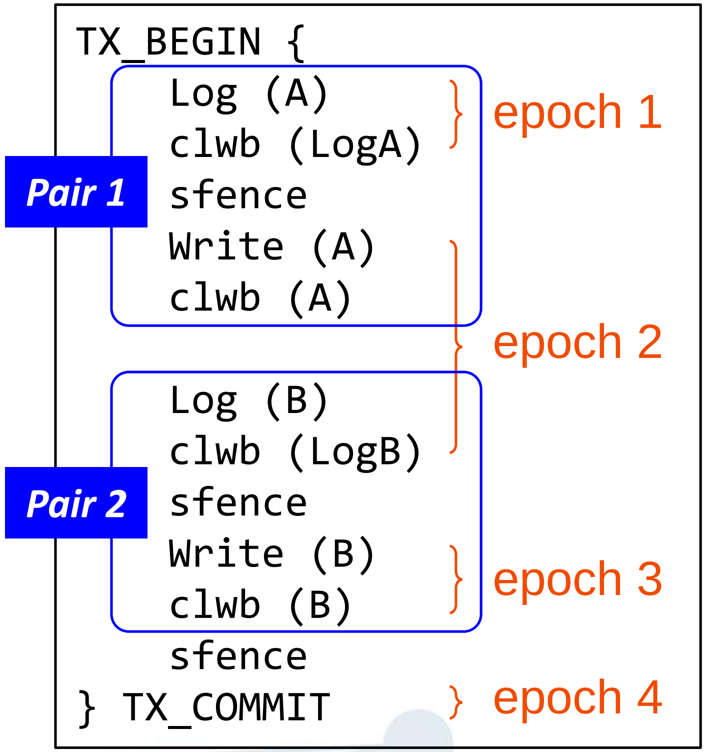


[1] Memory persistency@ISCA'14

[2] A transaction with pre-defined write set

Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model (TEP)



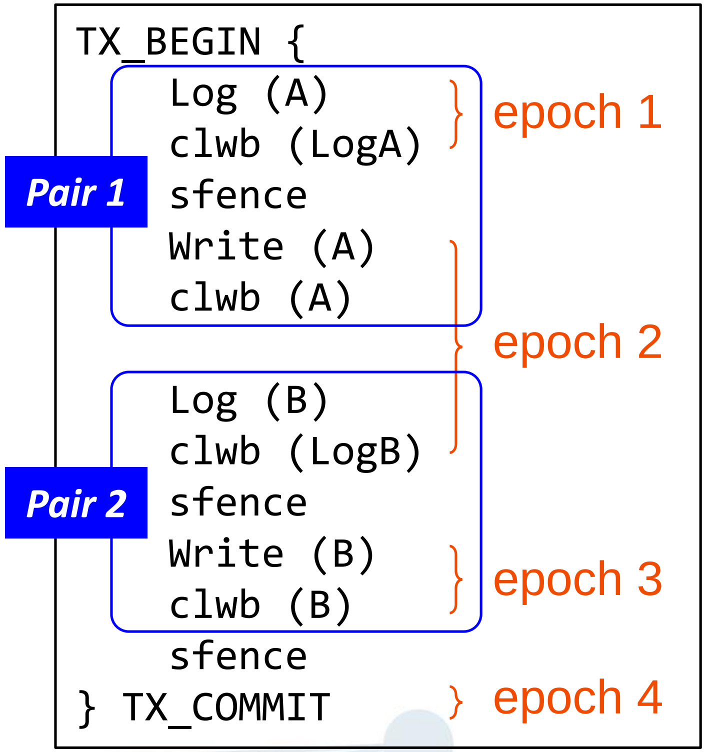
```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

A dynamic transaction

- ***<u>Paired epoch:</u>*** Two adjacent epochs are paired
  - Writes in one pair are persisted in epoch order
  - Different pairs are persisted w/o order

Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model (TEP)



```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

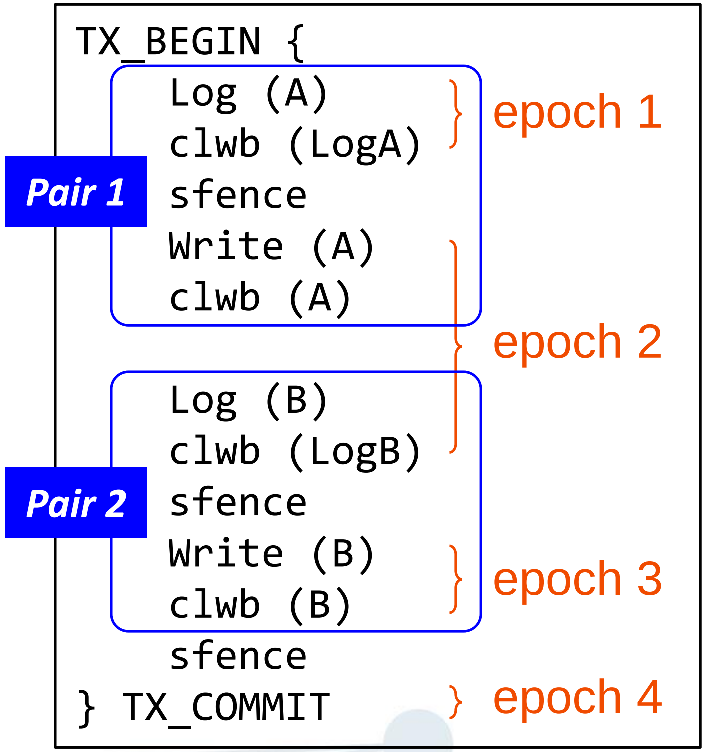
A dynamic transaction

- ***<u>Paired epoch:</u>*** Two adjacent epochs are paired
  - Writes in one pair are persisted in epoch order
  - Different pairs are persisted w/o order

➔ Efficient in both static and dynamic transactions

Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model (TEP)

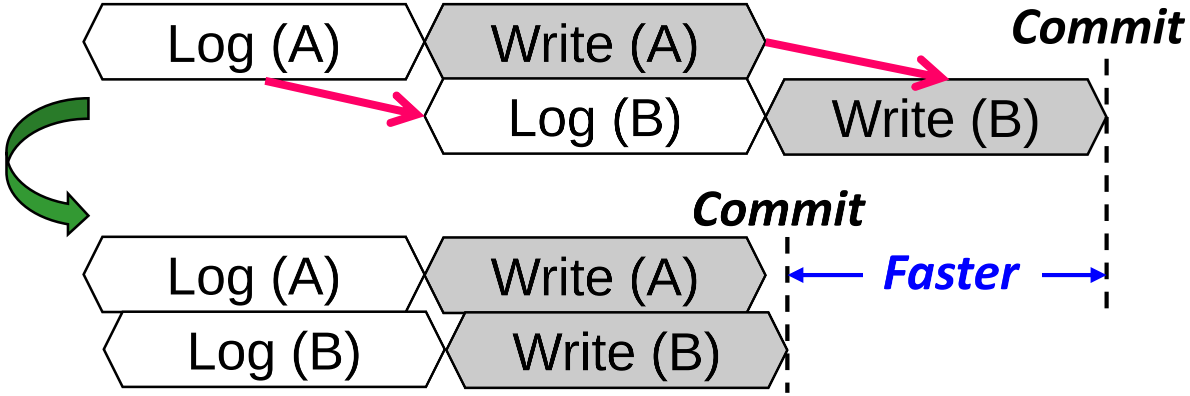


```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

A dynamic transaction

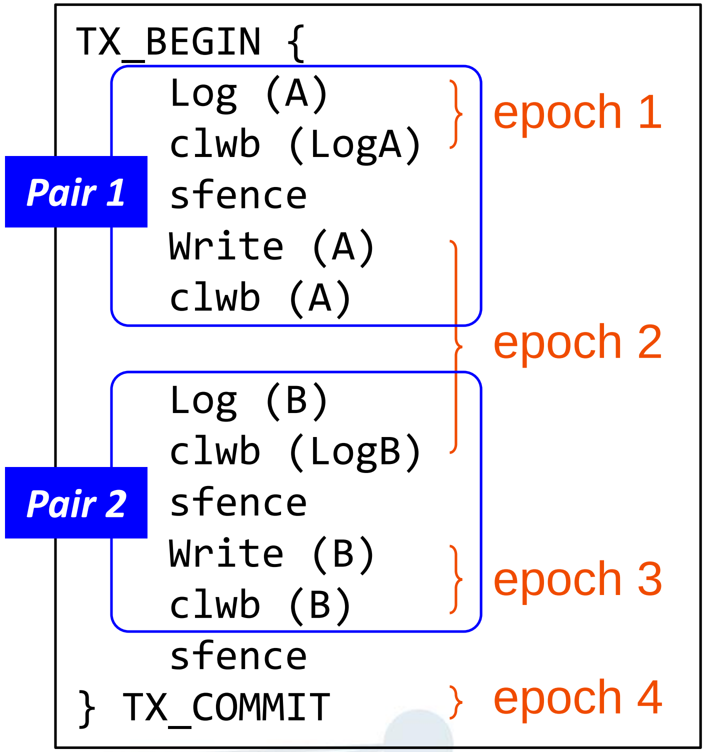
- ***Paired epoch:*** Two adjacent epochs are paired
    - Writes in one pair are persisted in epoch order
    - Different pairs are persisted w/o order

➔ Efficient in both static and dynamic transactions



Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model (TEP)

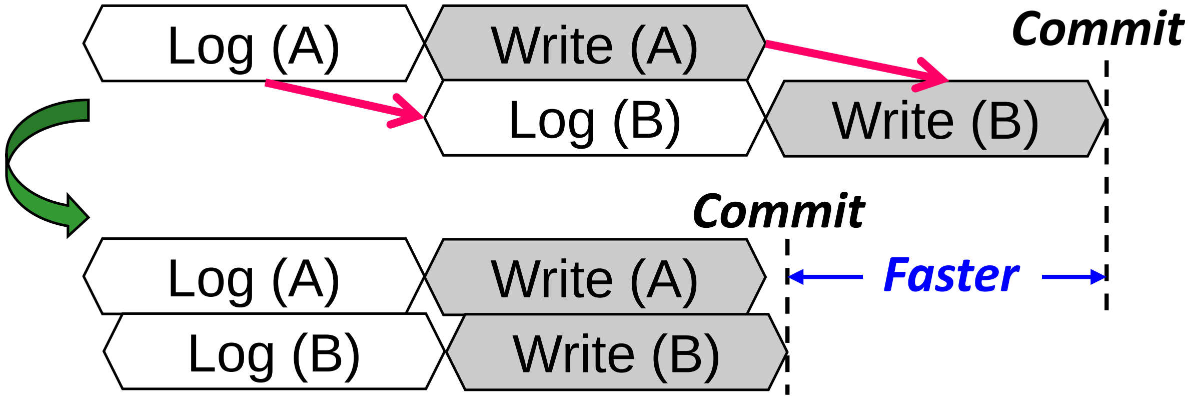


```
TX_BEGIN {
        Log (A)
        clwb (LogA)
        sfence
        Write (A)
        clwb (A)

        Log (B)
        clwb (LogB)
        sfence
        Write (B)
        clwb (B)
        sfence
} TX_COMMIT
```

A dynamic transaction

- ***Paired epoch:*** Two adjacent epochs are paired
  - Writes in one pair are persisted in epoch order
  - Different pairs are persisted w/o order

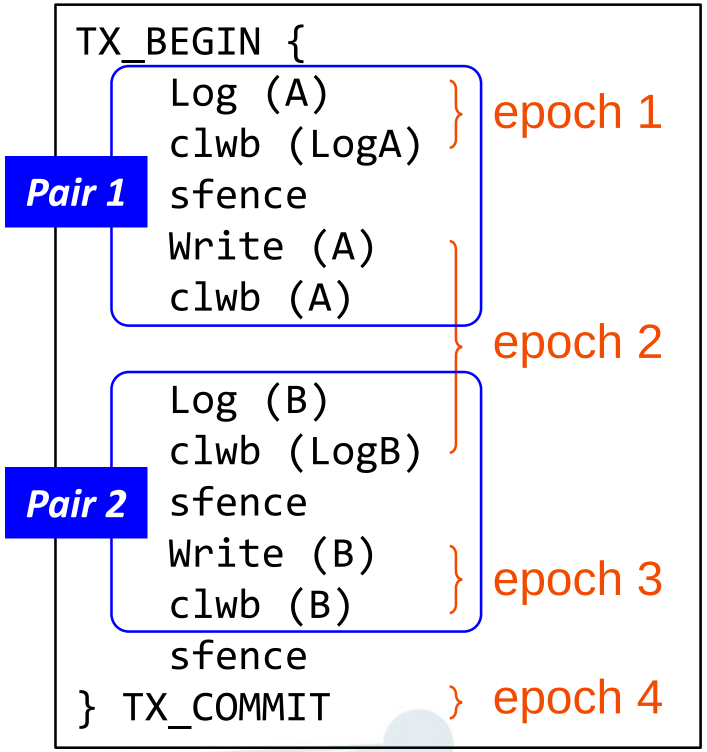
➔ Efficient in both static and dynamic transactions



➔ Minimize ordering constraints

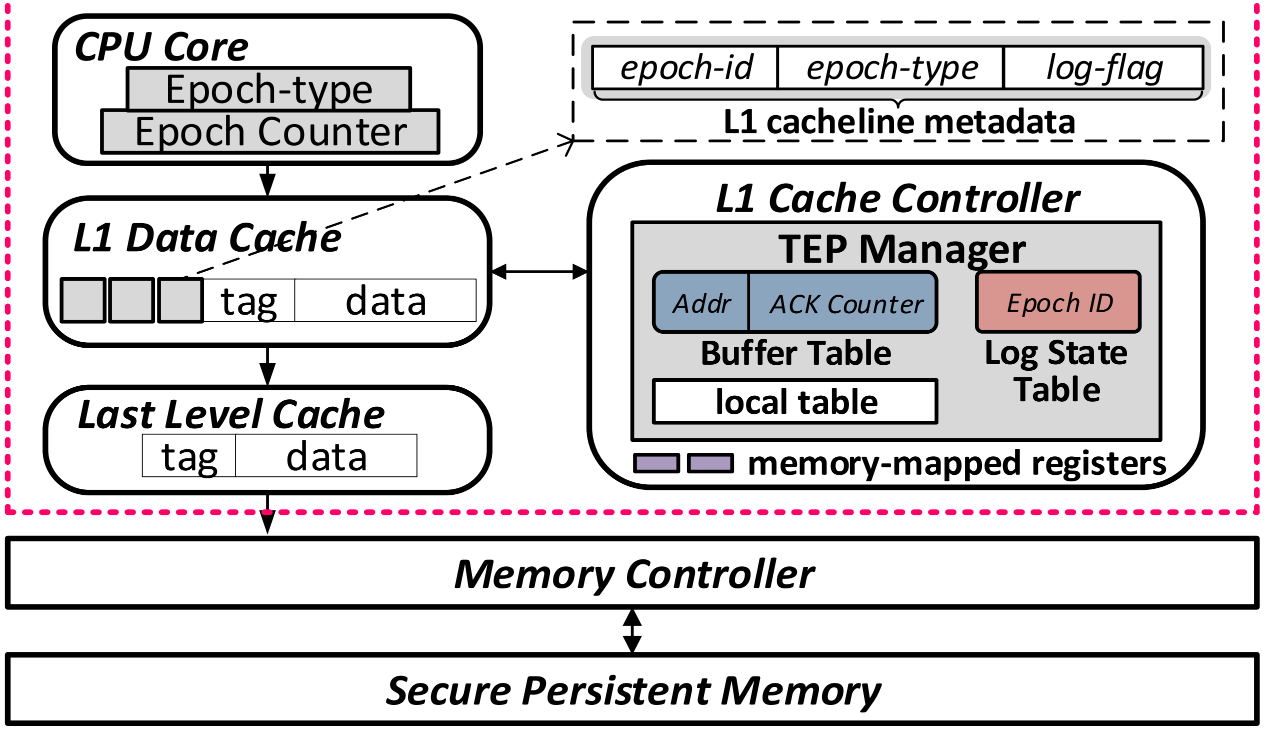
Transaction-Specific Epoch Persistency Model

Implementations of TEP



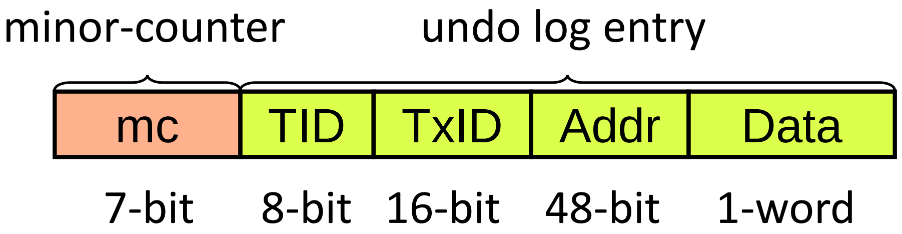
```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    Log (B)
    clwb (LogB)
    sfence
    Write (B)
    clwb (B)
    sfence
} TX_COMMIT
```

A dynamic transaction



Security Metadata Write-Reduction Schemes

Co-locate log and counter



minor-counter

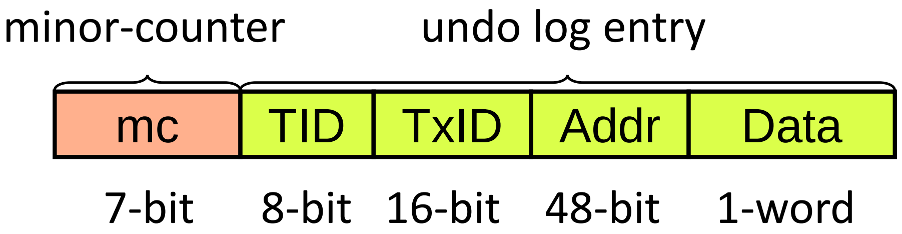
undo log entry

mc	TID	TxID	Addr	Data
7-bit	8-bit	16-bit	48-bit	1-word

Write a minor-counter together with a log entry

Security Metadata Write-Reduction Schemes

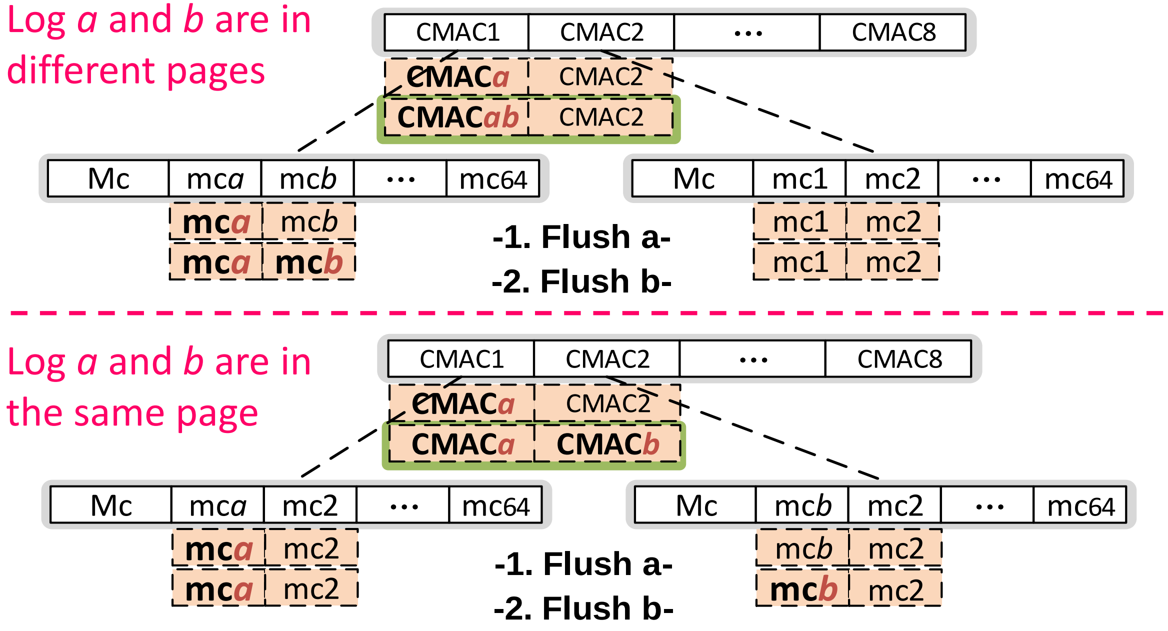
Co-locate log and counter



minor-counter	undo log entry			
mc	TID	TxID	Addr	Data
7-bit	8-bit	16-bit	48-bit	1-word

Write a minor-counter together with a log entry

Coalesce BMT blocks



Log *a* and *b* are in different pages

Log *a* and *b* are in the same page

Exploit the spatial locality to merge BMT writes

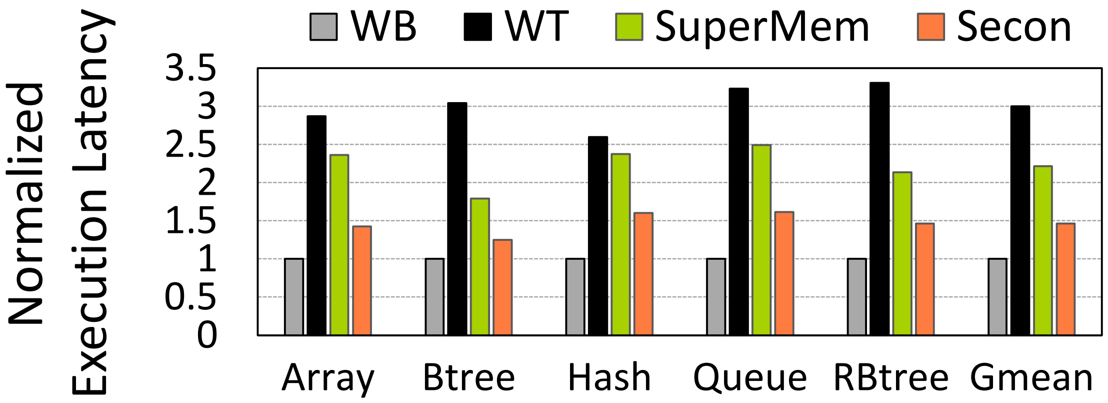
Performance Evaluation

- Model Secon using Gem5 and NVMain

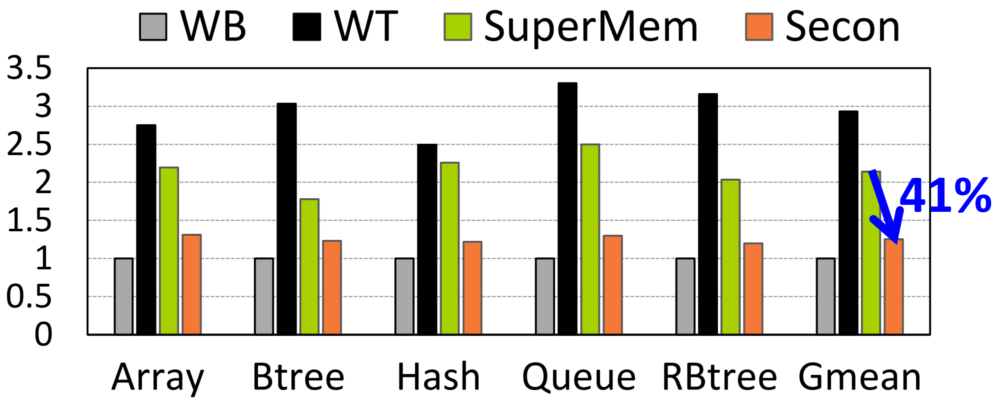
Design	Description
WB	An ideal write-back scheme
WT	A write-through scheme
SuperMem	A write-optimized write-through scheme using our BMT coalescing
Secon	Our proposed schemes

Benchmark	Description
Array	Swap two random entries in an array
Queue	Enqueue/dequeue random entries in a queue
Btree	Insert/delete random nodes in a B-tree
Hash	Insert/delete random items in a hash table
RBtree	Insert/delete random nodes in a red-black tree
YCSB	Cloud benchmark. 100% update
TPCC	OLTP benchmark. Use the New-Order transaction

Transaction Latency



1 operation per transaction

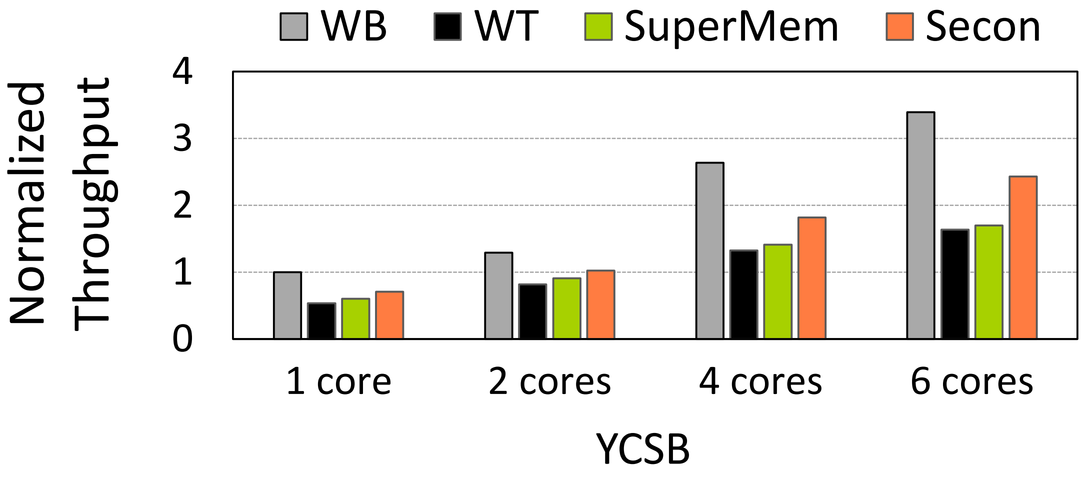


16 operations per transaction

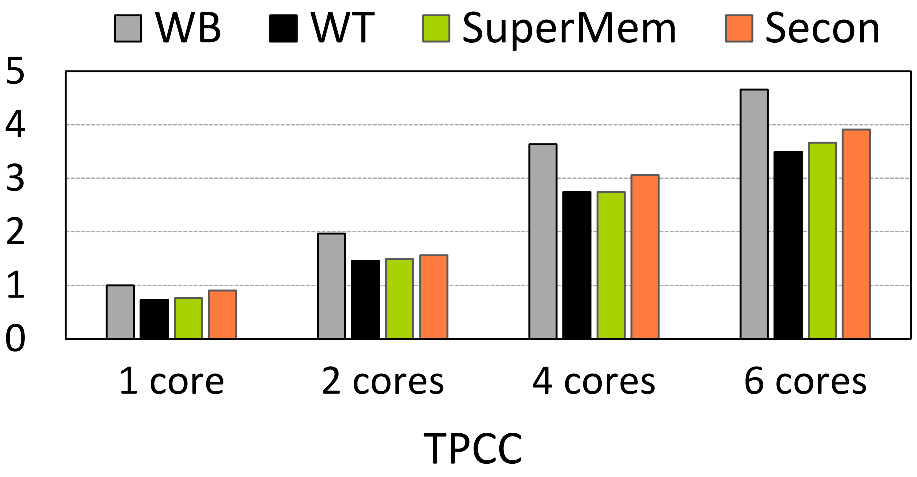
TEP improves by 14%

- Scalable security metadata cache
- Move BMT update to the background

Transaction Throughput

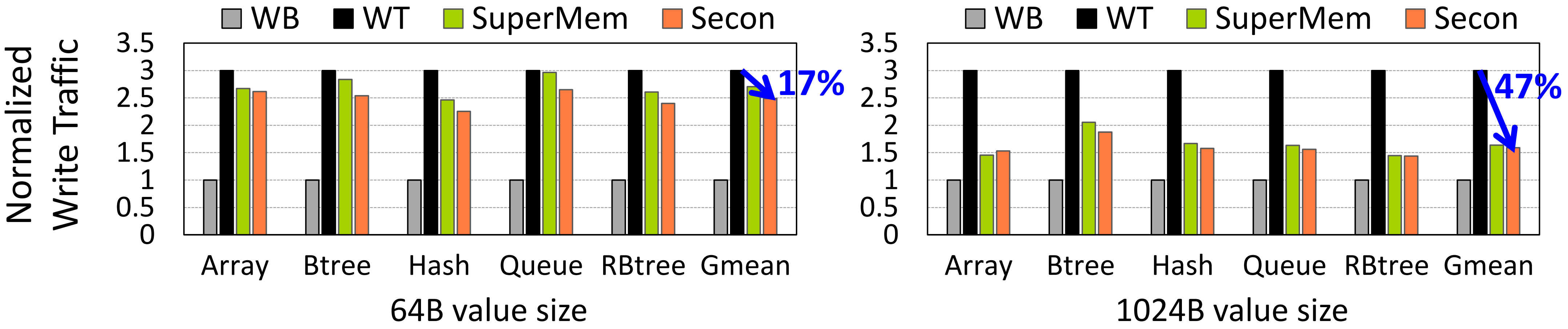


Improve throughput by
43% over SuperMem



Improve throughput by
19% over SuperMem

Write Traffic



- Reduce the number of writes
  - Log and counter co-locating
  - BMT block coalescing

Conclusion

- Security and crash consistency are important for persistent memory
- Existing approaches suffer from low scalability
- Our solution: **Secon**
  - Scalable write-through security metadata cache
    - Move BMT update to the background
  - Transaction-specific epoch persistency model
    - Minimize ordering constraints
  - Security metadata write-reduction schemes
    - Enhance endurance

59 DESIGN AUTOMATION CONFERENCE

Thanks! Q&A