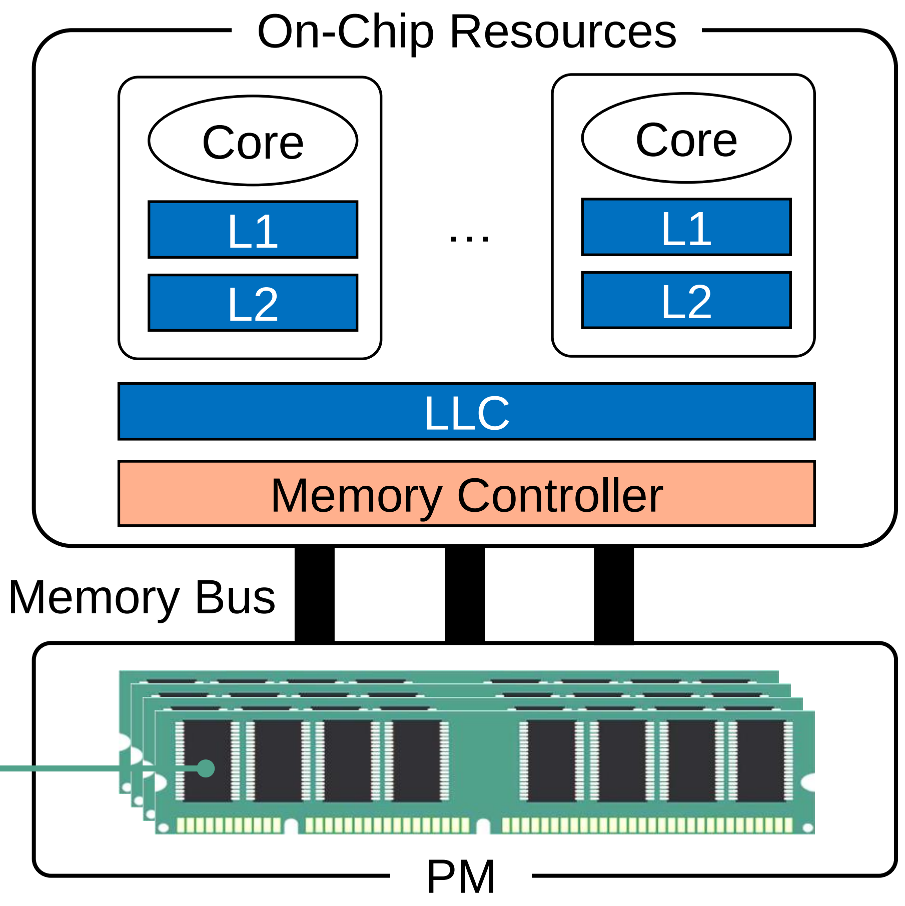
Scalable Crash Consistency for Secure Persistent Memory

Ming Zhang, Yu Hua, Xuan Li, Hao Xu

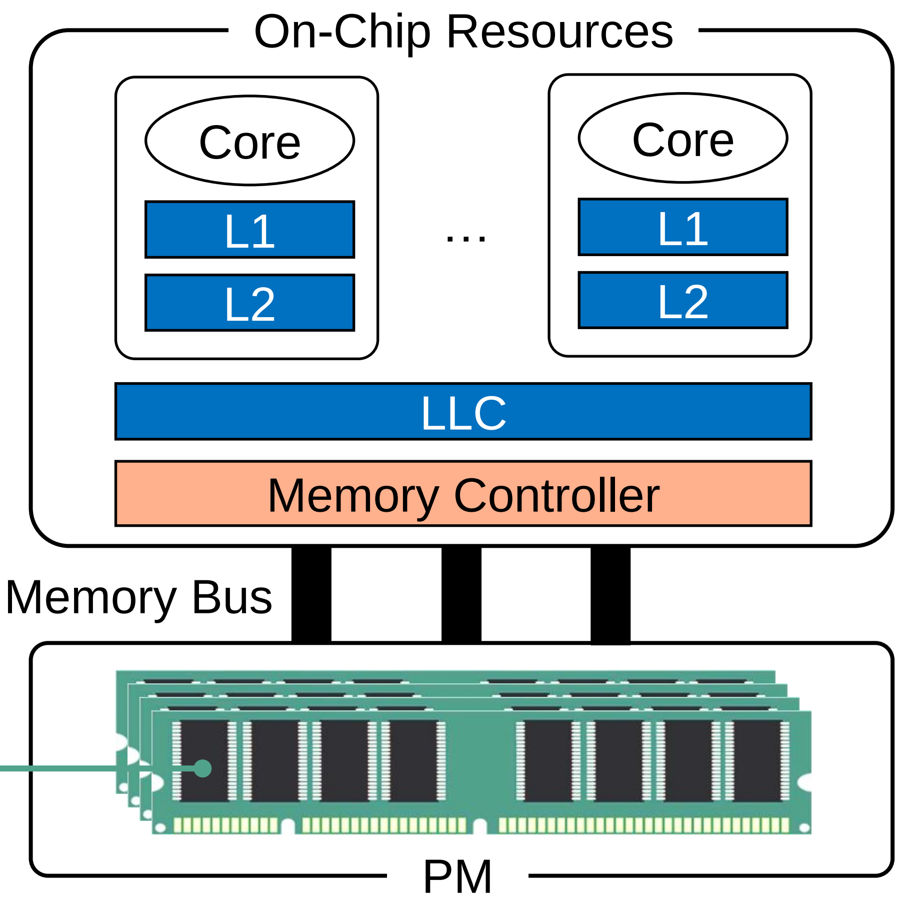
Huazhong University of Science and Technology, China

Persistent Memory (PM)



Non-volatility:
Data are not lost

Persistent Memory (PM)



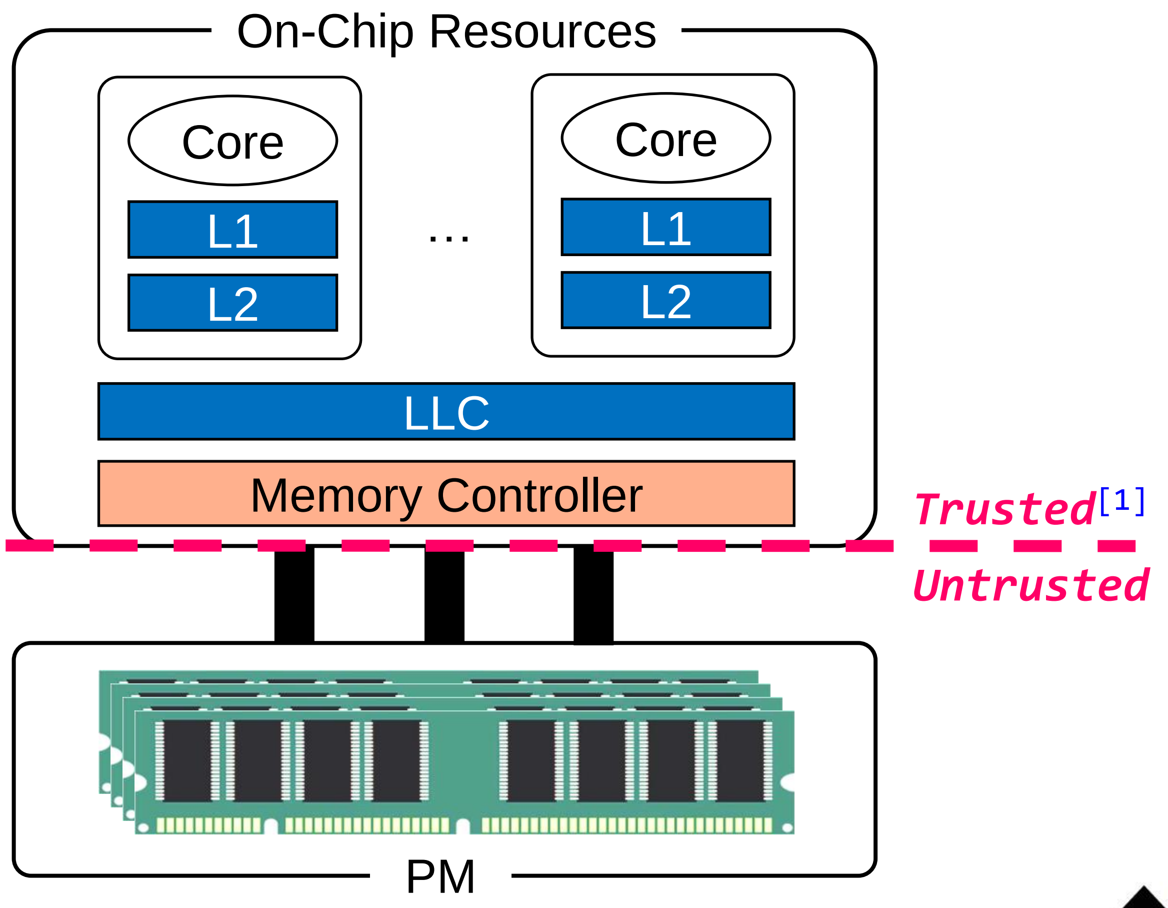
Requirements

① **Security**

② **Crash Consistency**

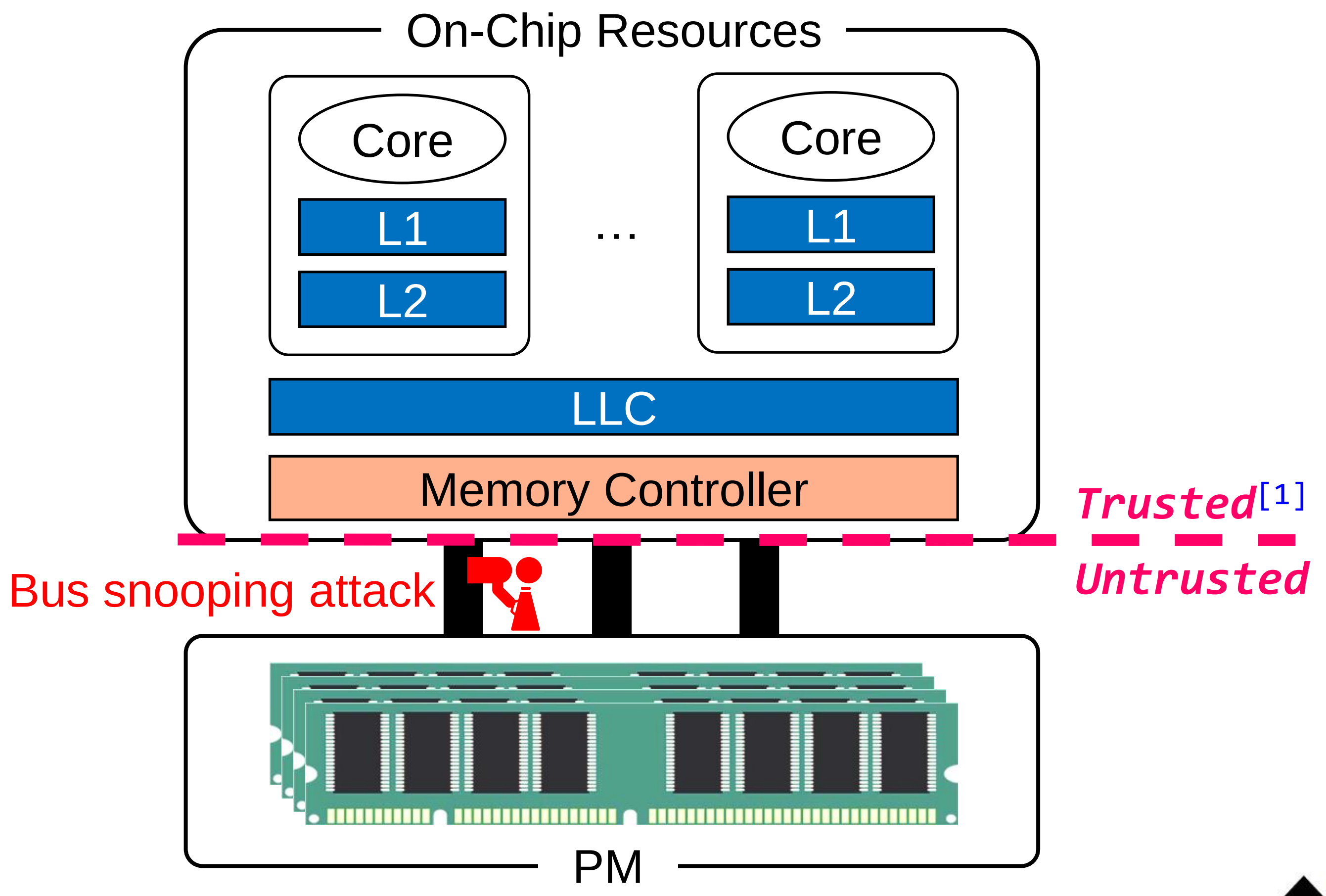
Non-volatility:
Data are not lost

Security for PM



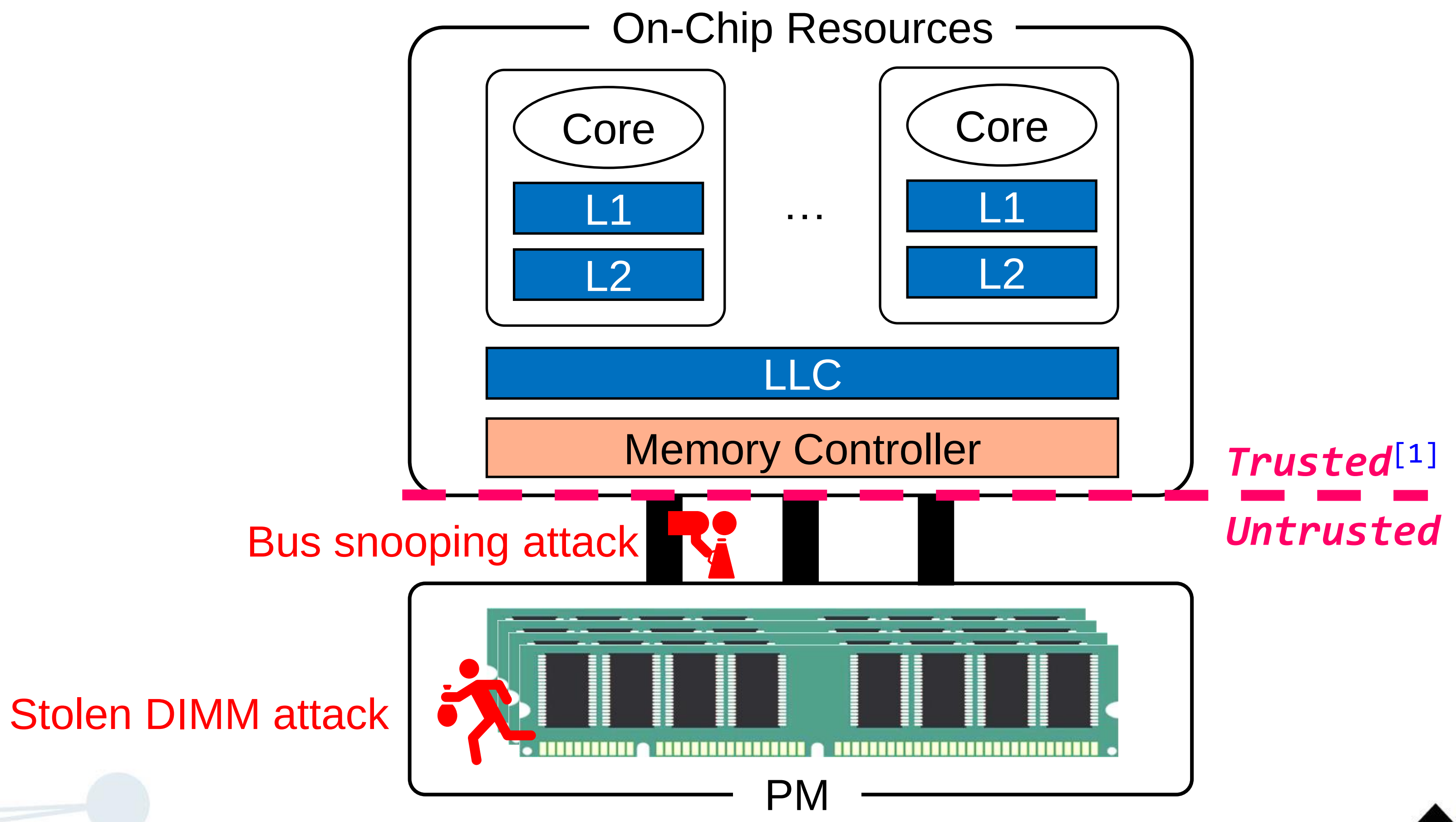
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



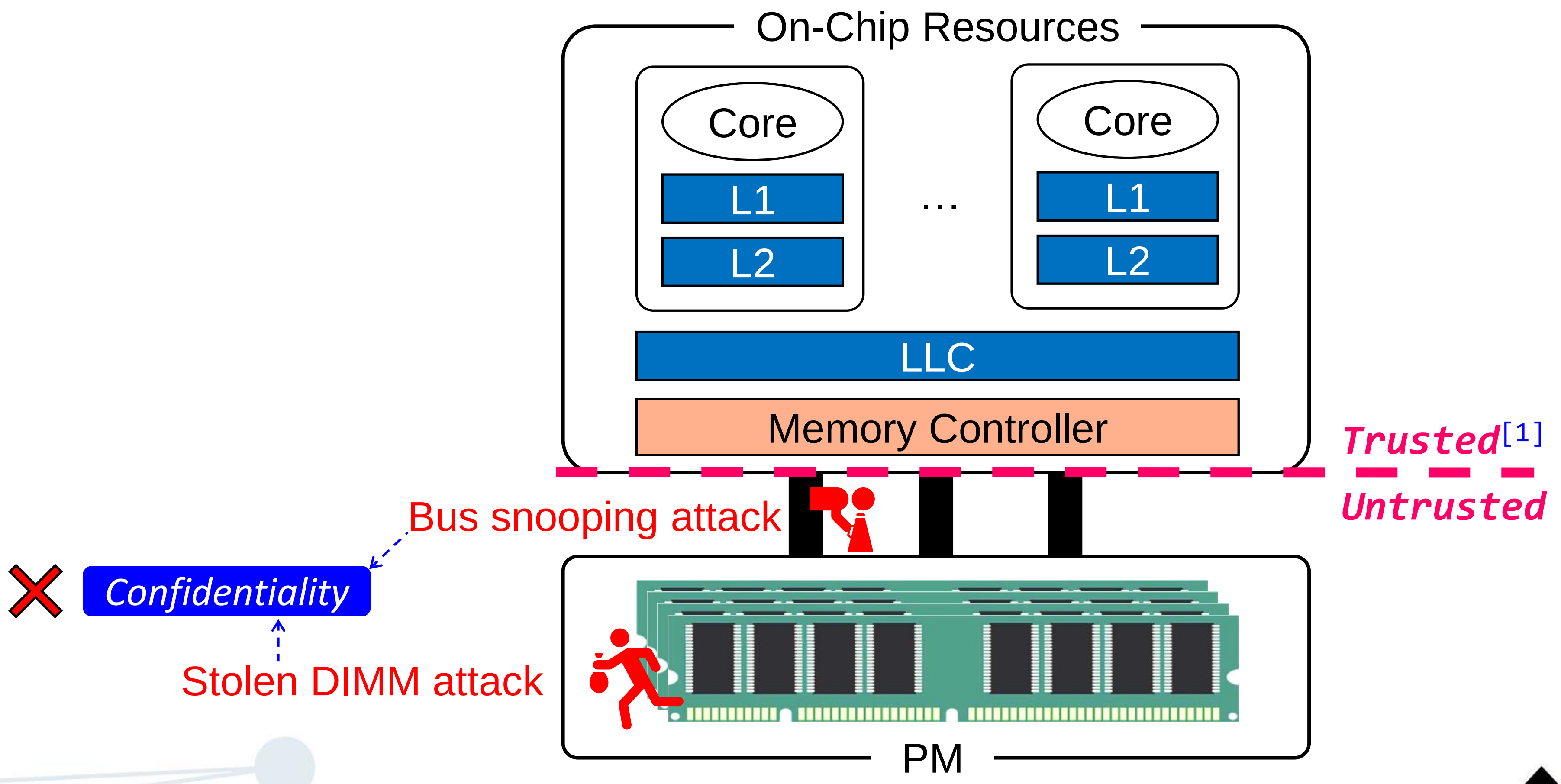
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM



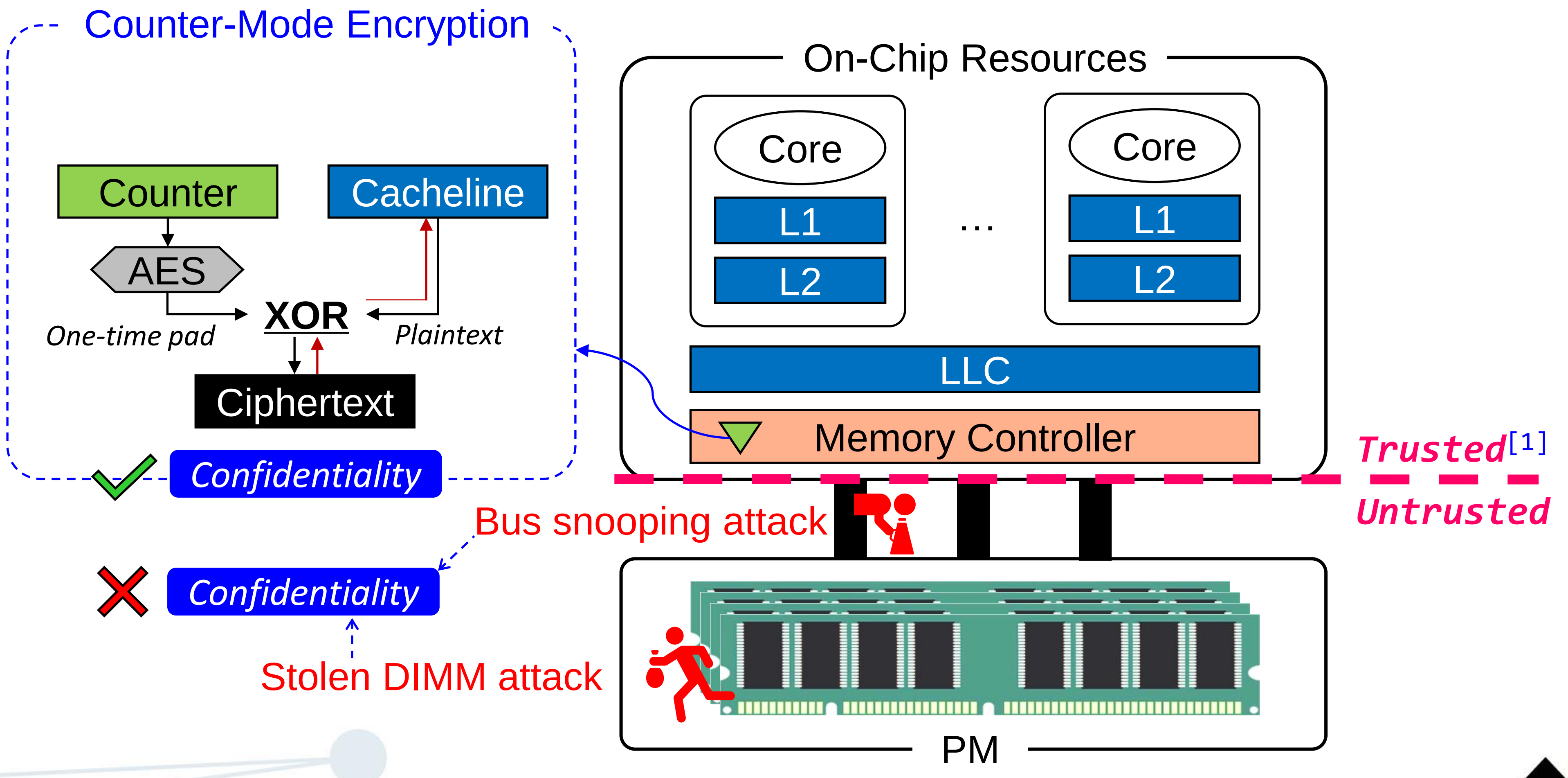
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

Security for PM

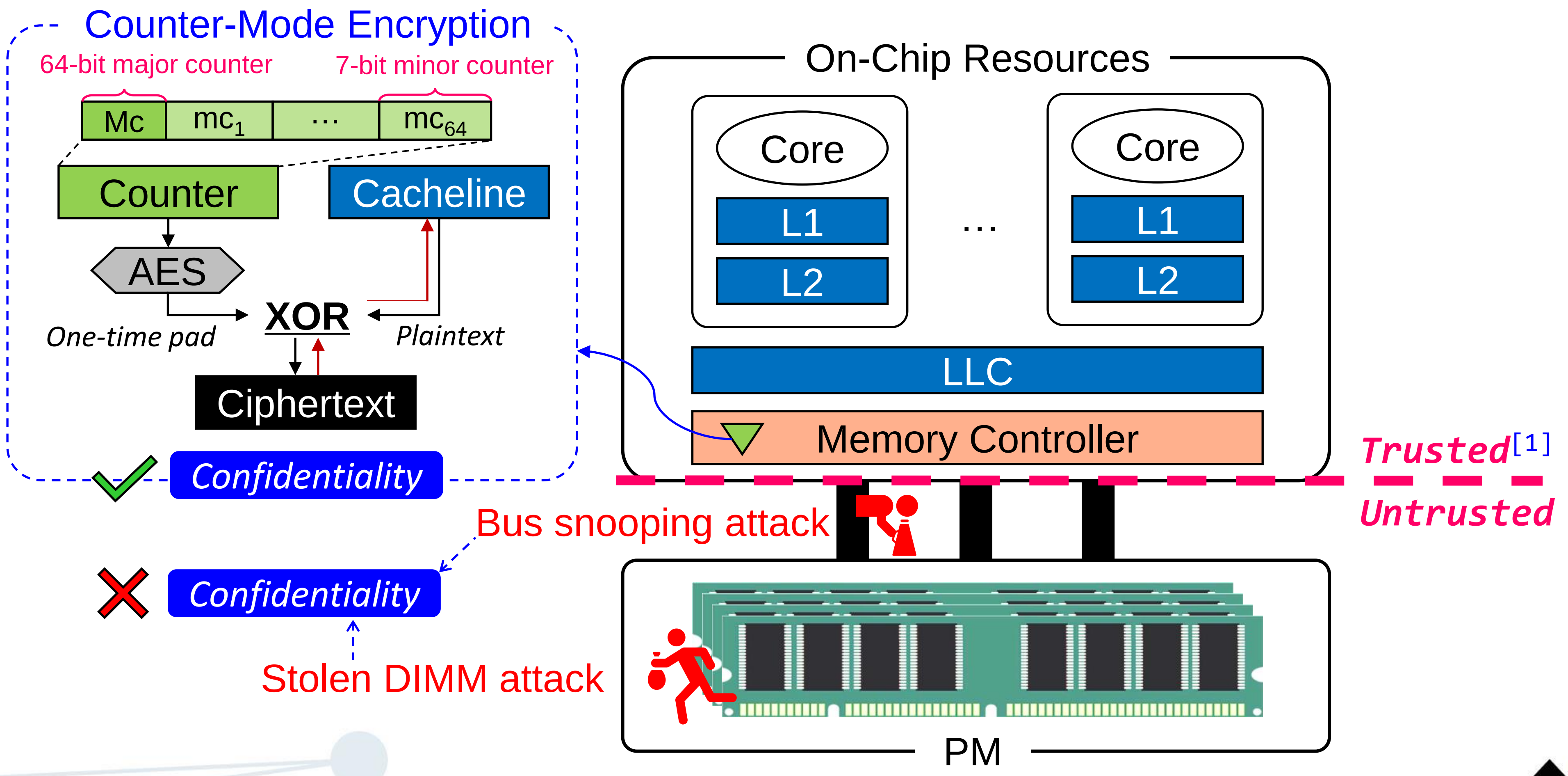


[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21

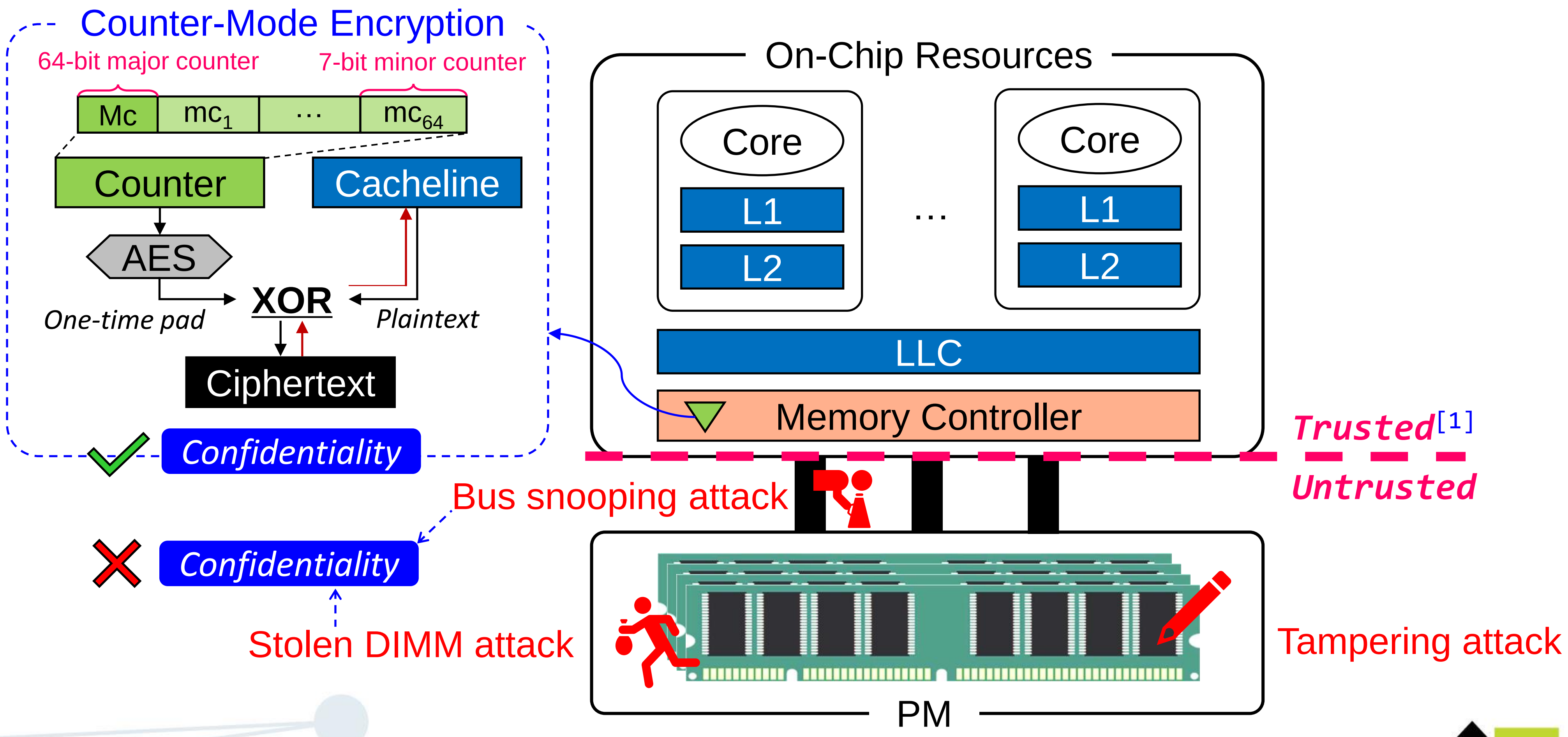
Security for PM



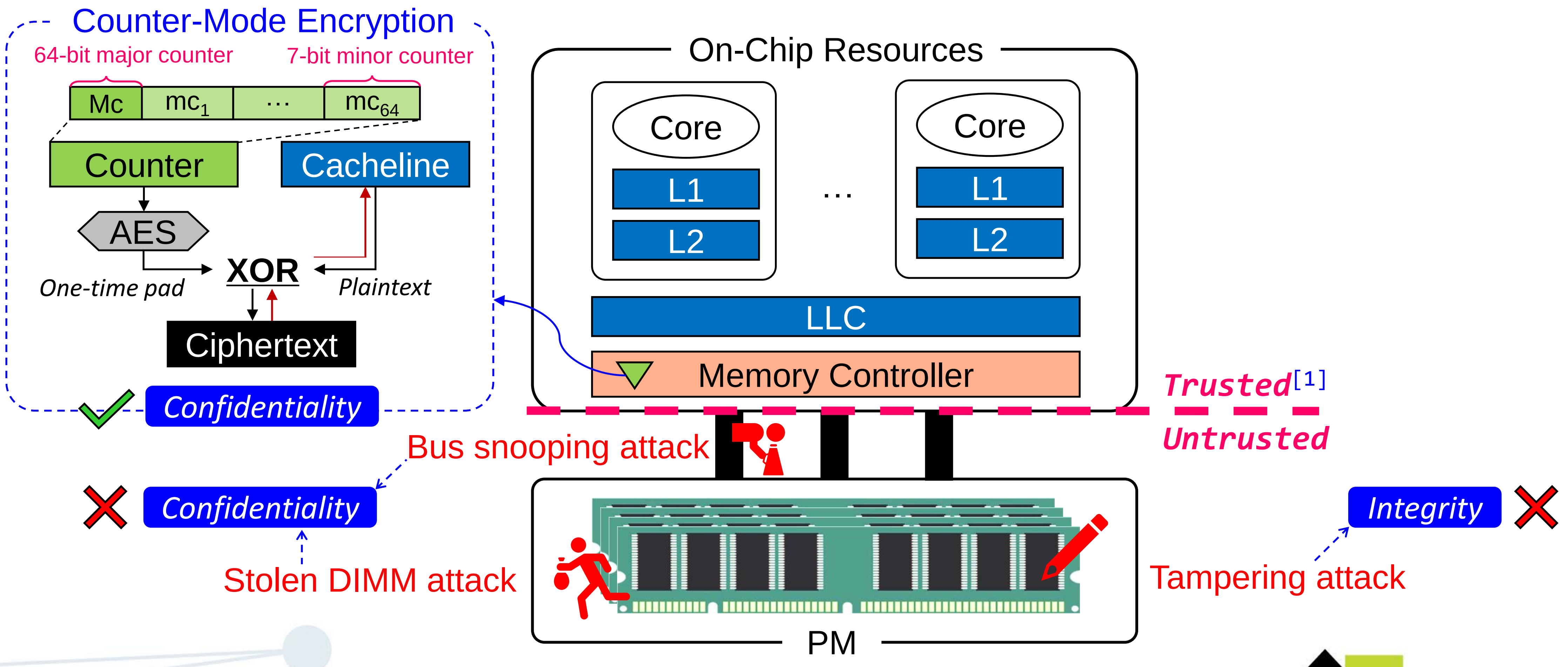
Security for PM



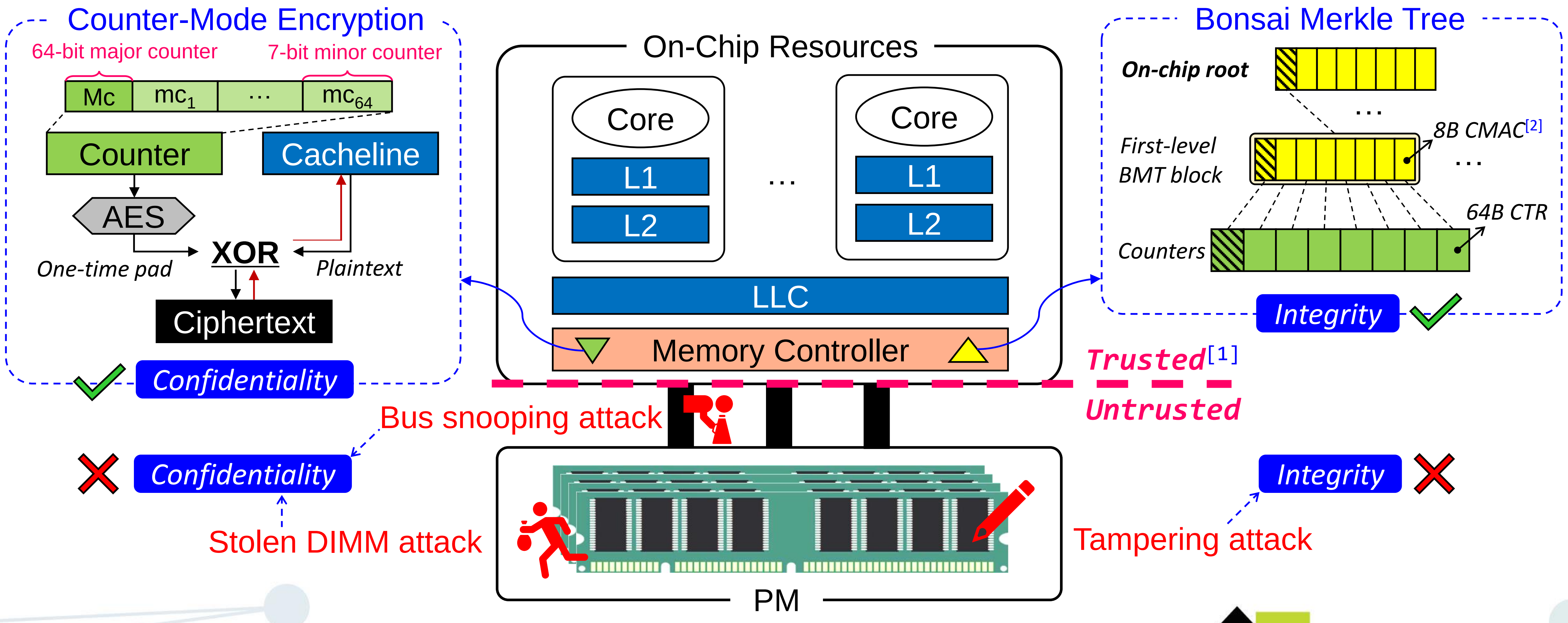
Security for PM



Security for PM

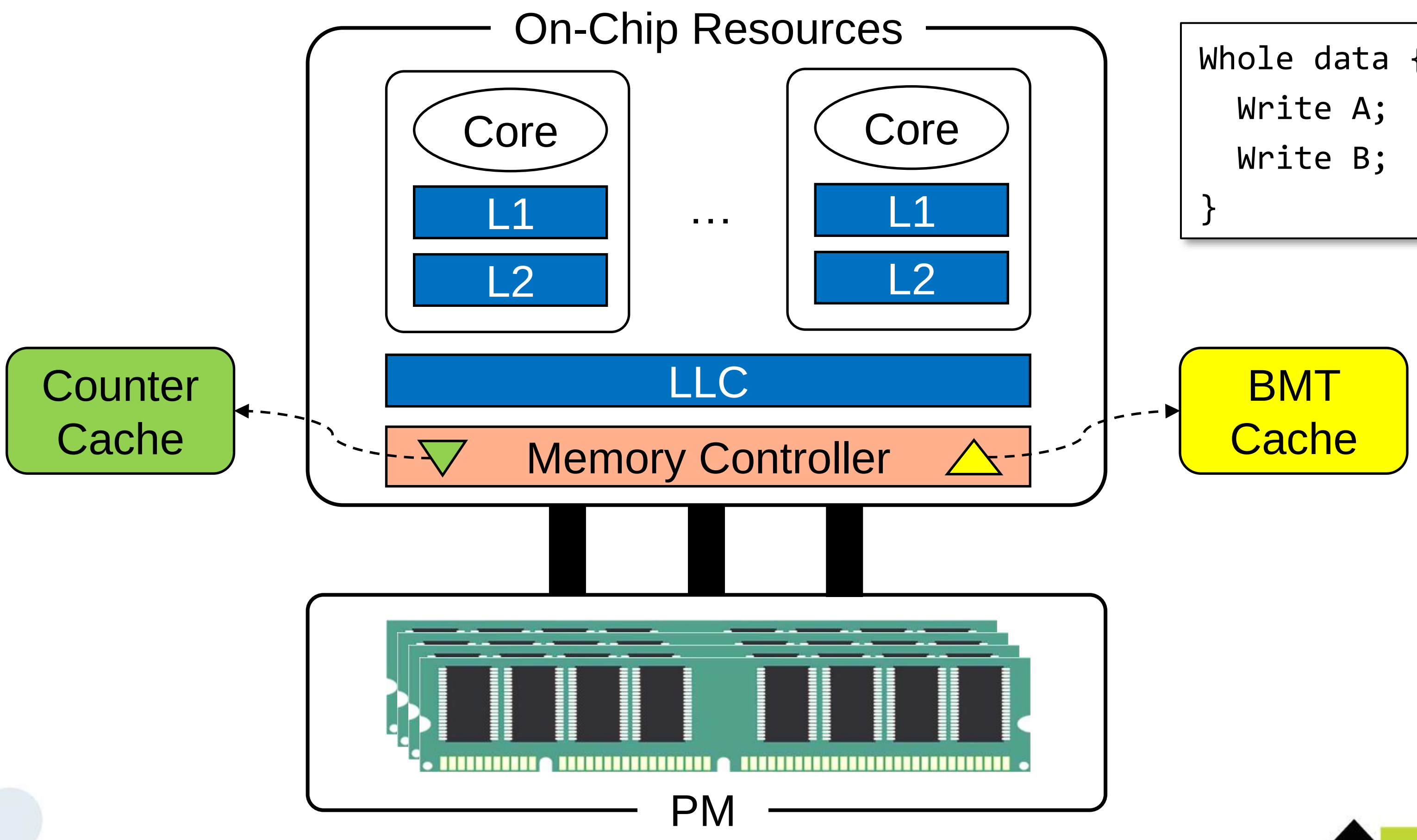


Security for PM



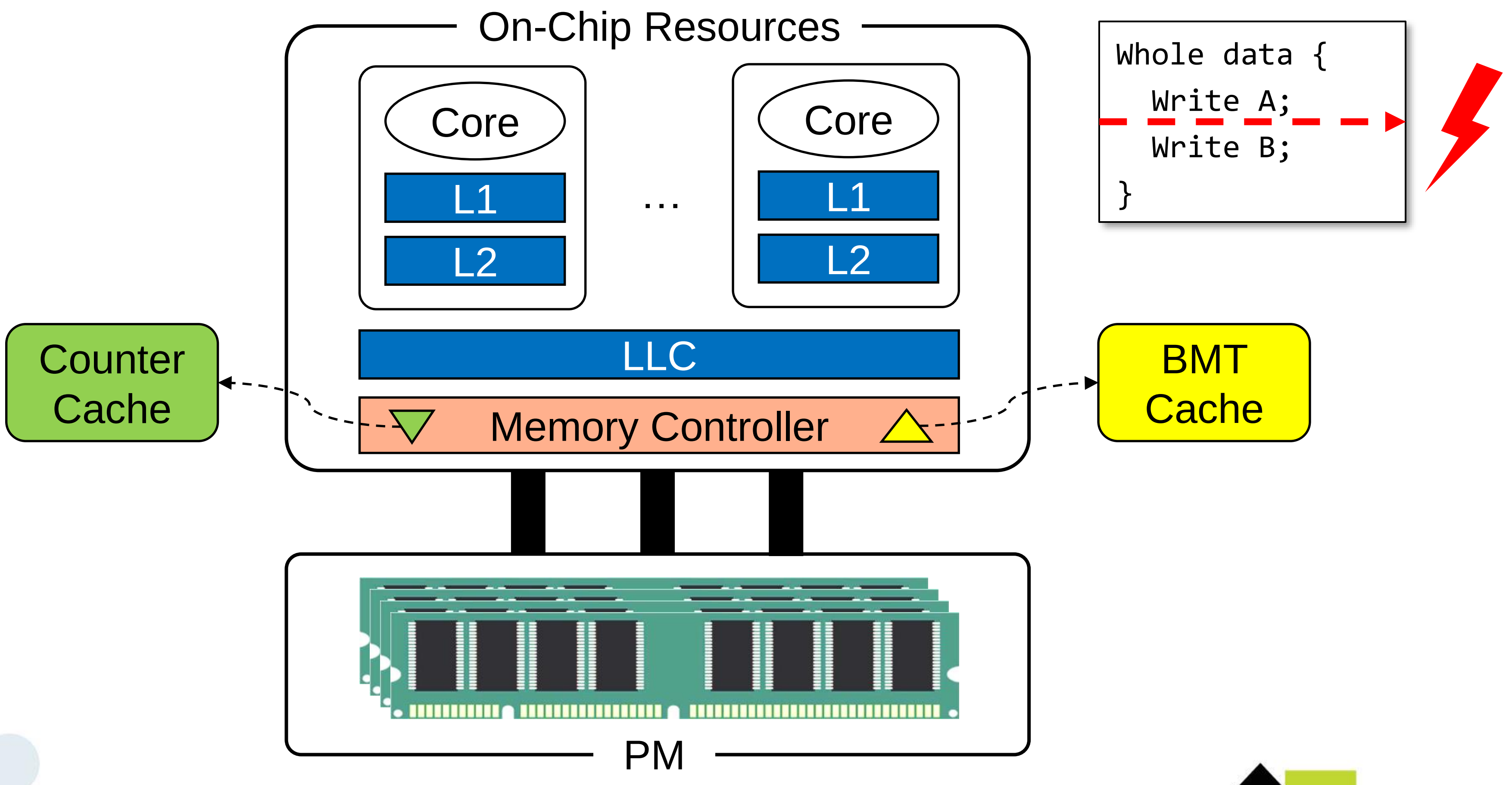
[1] SECRET@DAC'16, SCA@HPCA'18, SuperMem@MICRO'19, Bonsai Merkle Forests@MICRO'21
[2] Counter message authentication code

Crash Consistency for Secure PM



```
Whole data {
  Write A;
  Write B;
}
```

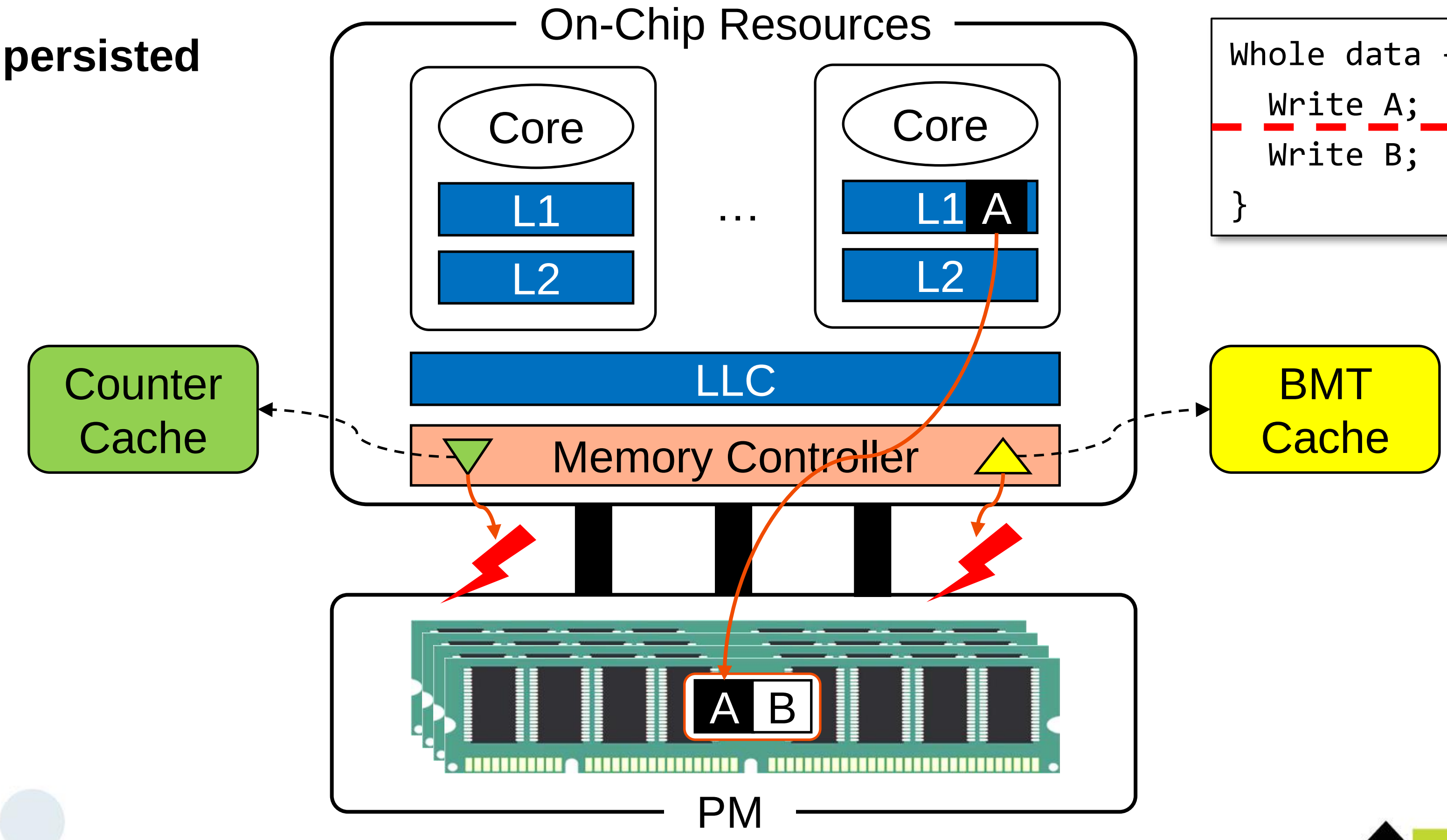
Crash Consistency for Secure PM

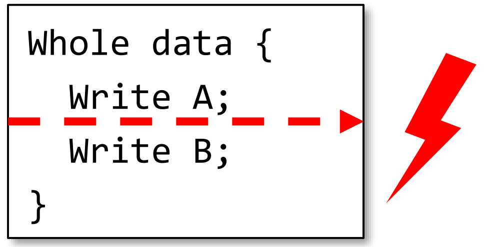


```
Whole data {
   Write A;
   Write B;
}
```

Crash Consistency for Secure PM

1) Only data is persisted

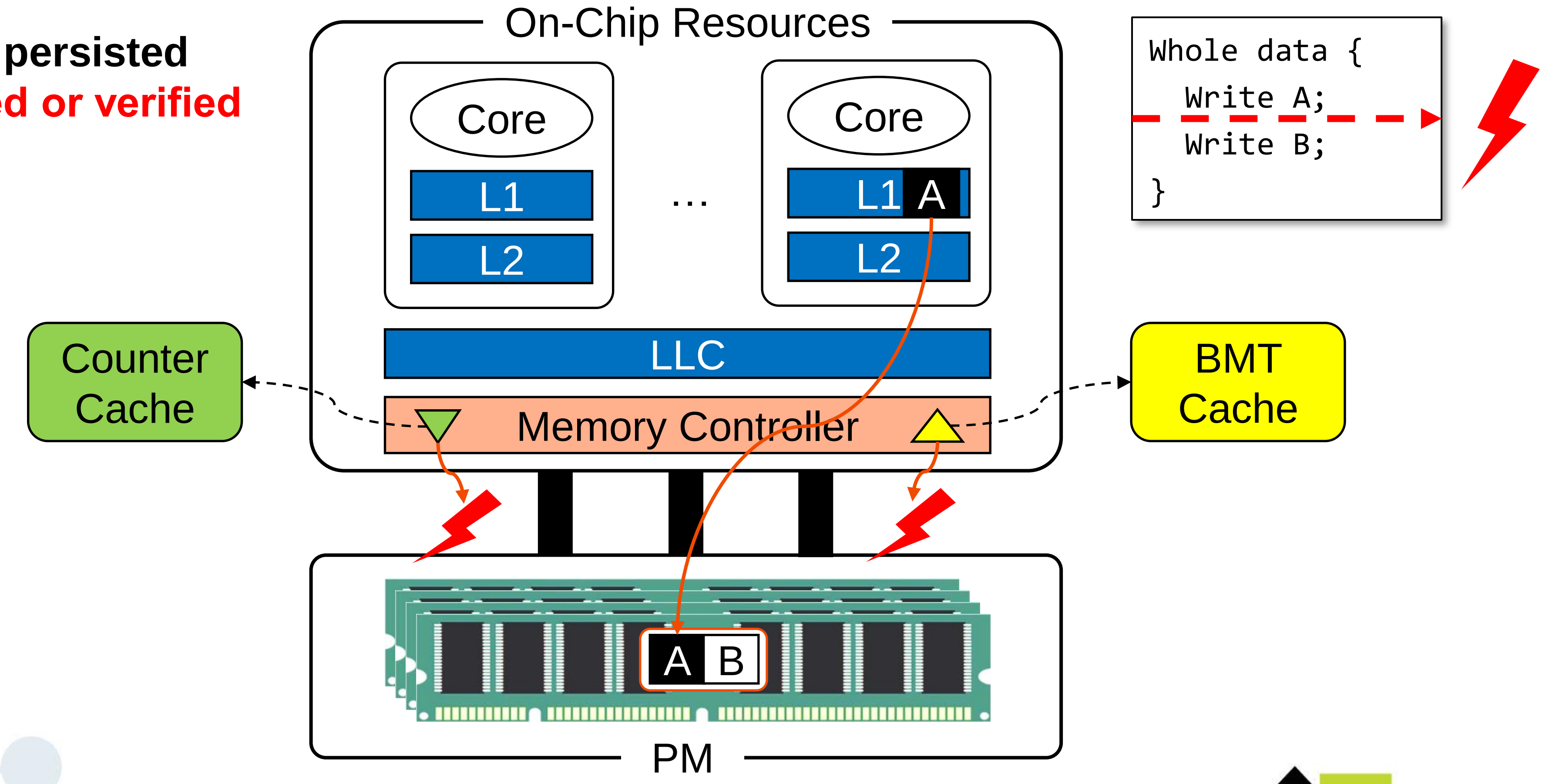




```
Whole data {
  Write A;
  Write B;
}
```

Crash Consistency for Secure PM

1) Only data is persisted
Can't be decrypted or verified



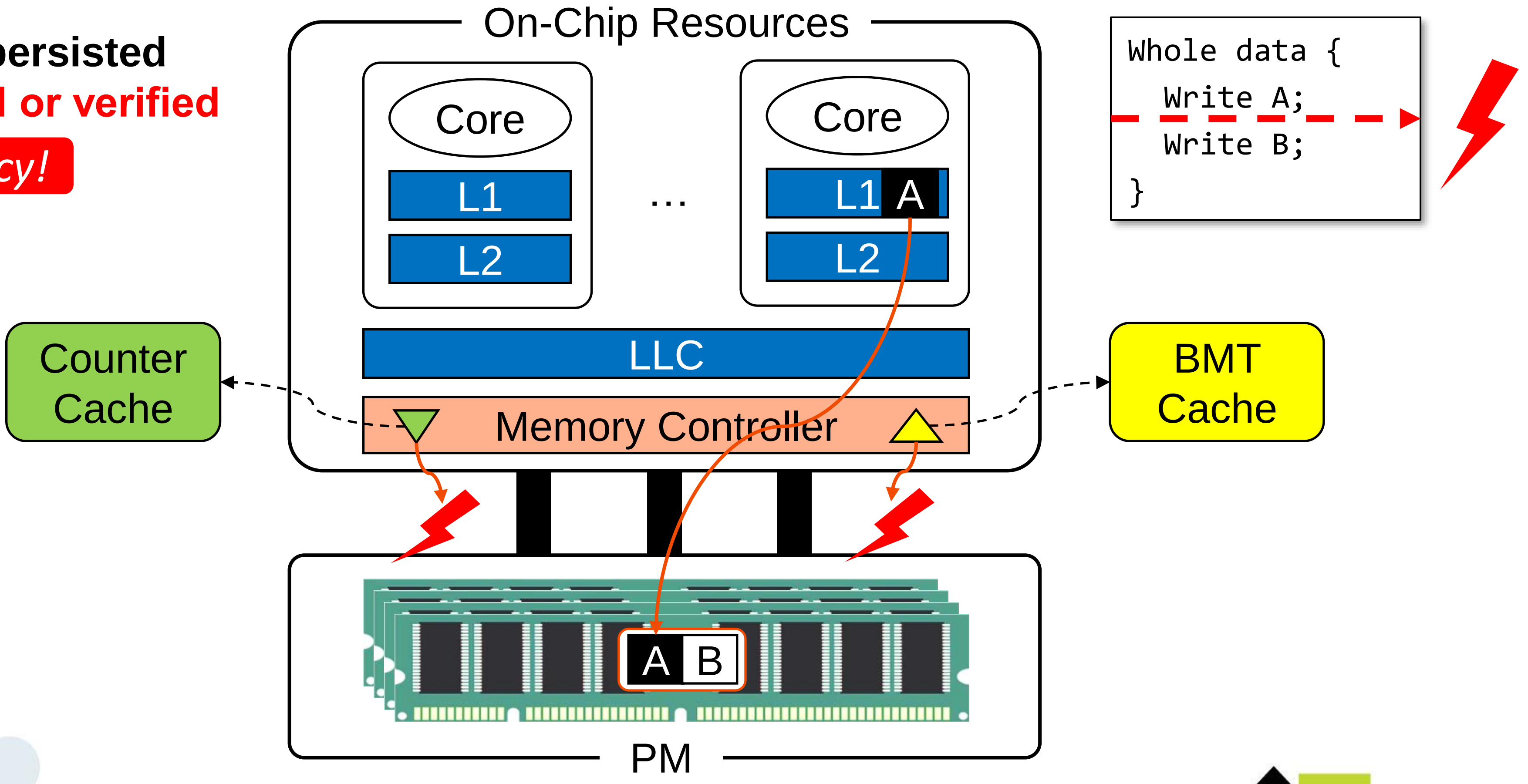
```
Whole data {
  Write A;
  Write B;
}
```

Crash Consistency for Secure PM

1) Only data is persisted

Can't be decrypted or verified

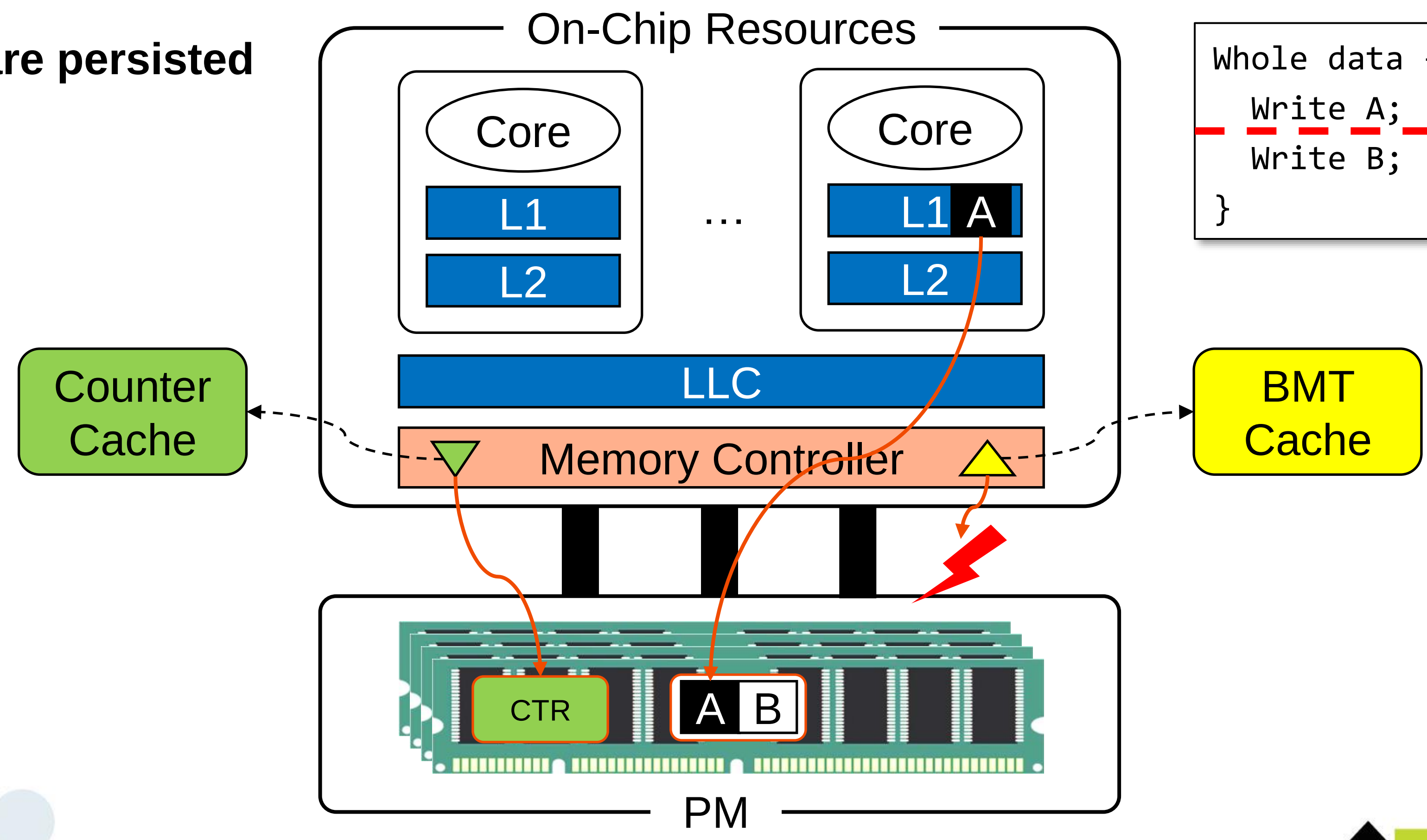
Inconsistency!

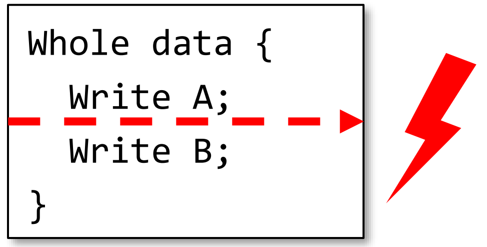


```
Whole data {
   Write A;
   Write B;
}
```

Crash Consistency for Secure PM

2) Data + counter are persisted



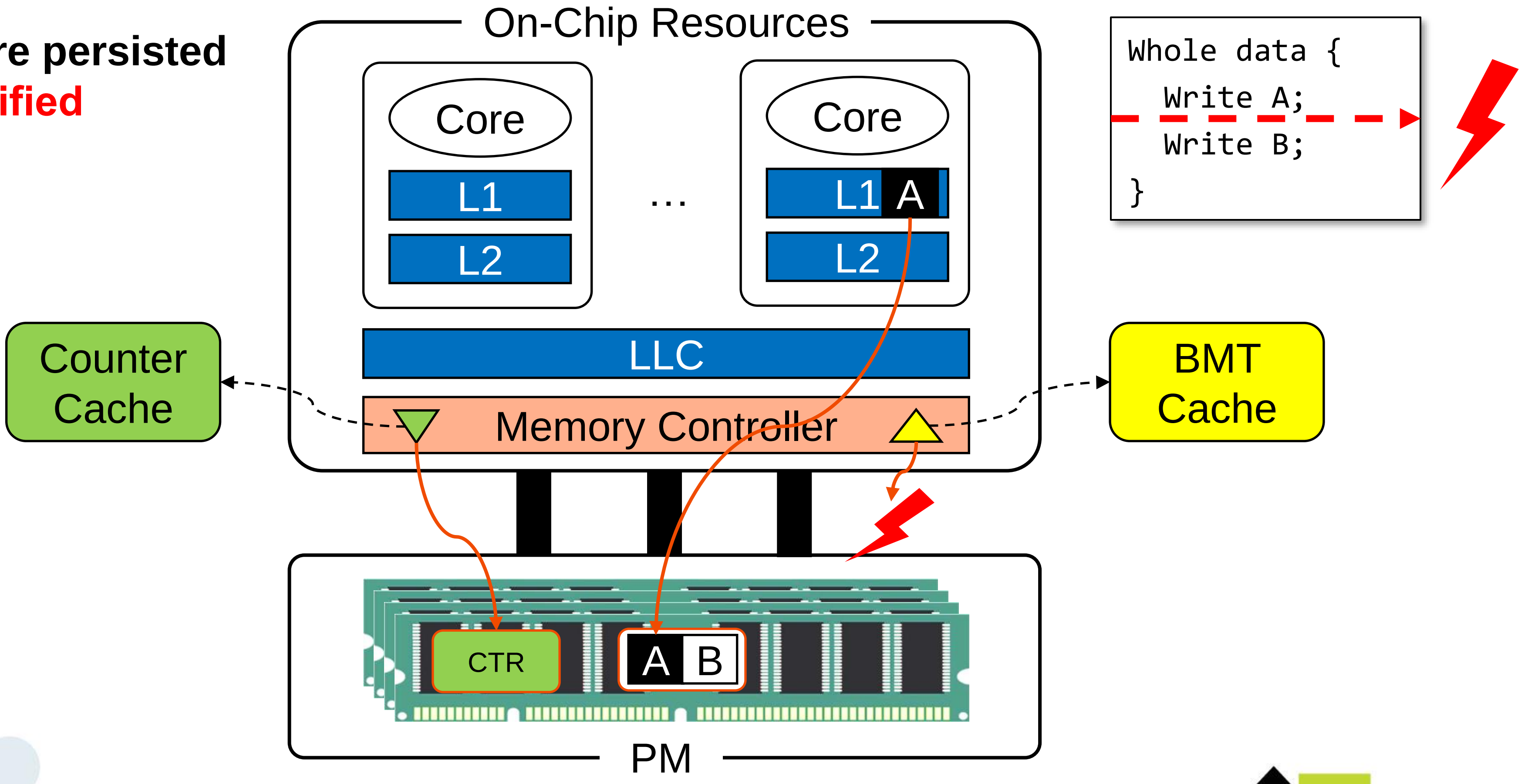


```
Whole data {
   Write A;
   Write B;
}
```

Crash Consistency for Secure PM

2) Data + counter are persisted

Can't be verified



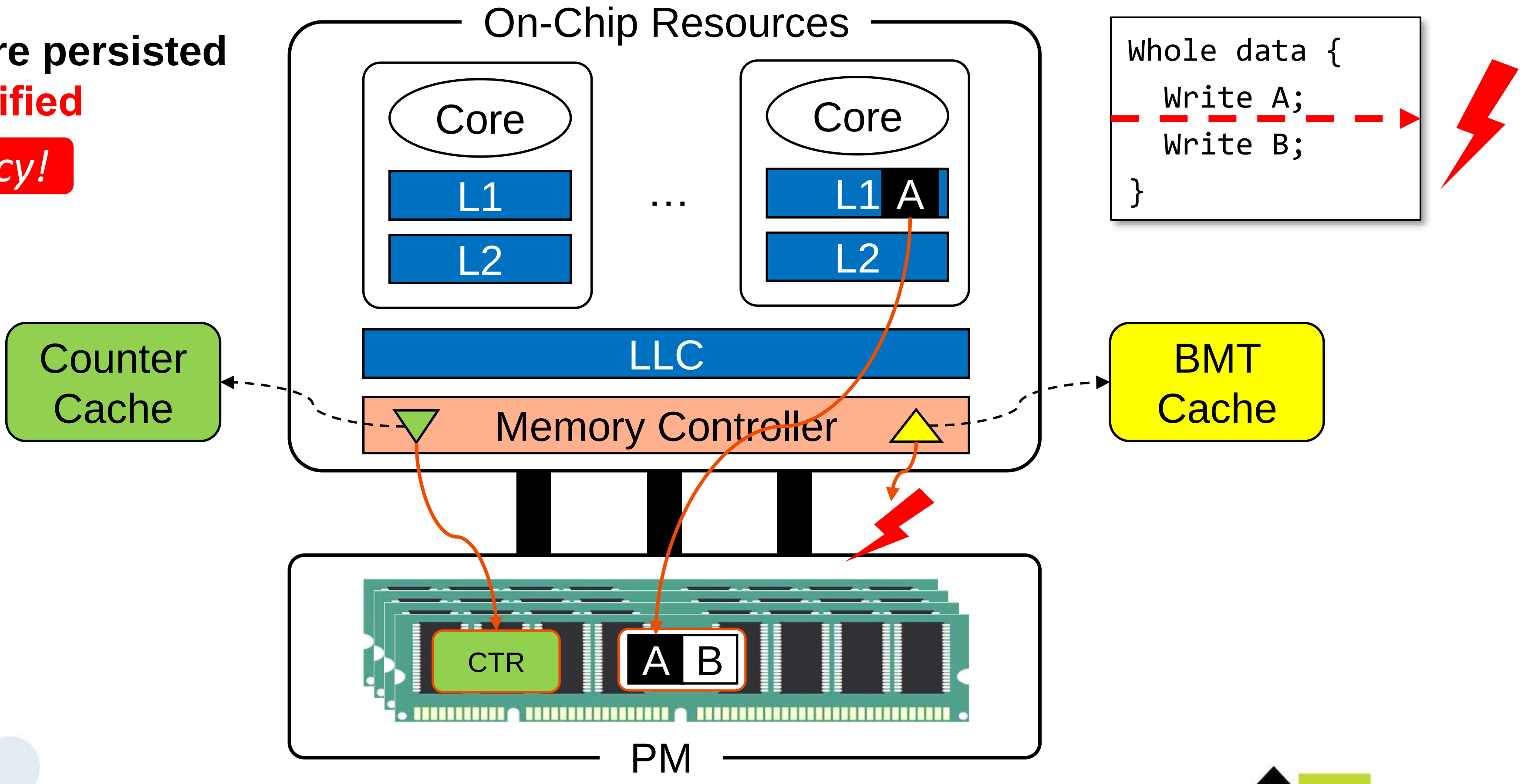
```
Whole data {
  Write A;
  Write B;
}
```

Crash Consistency for Secure PM

2) Data + counter are persisted

Can't be verified

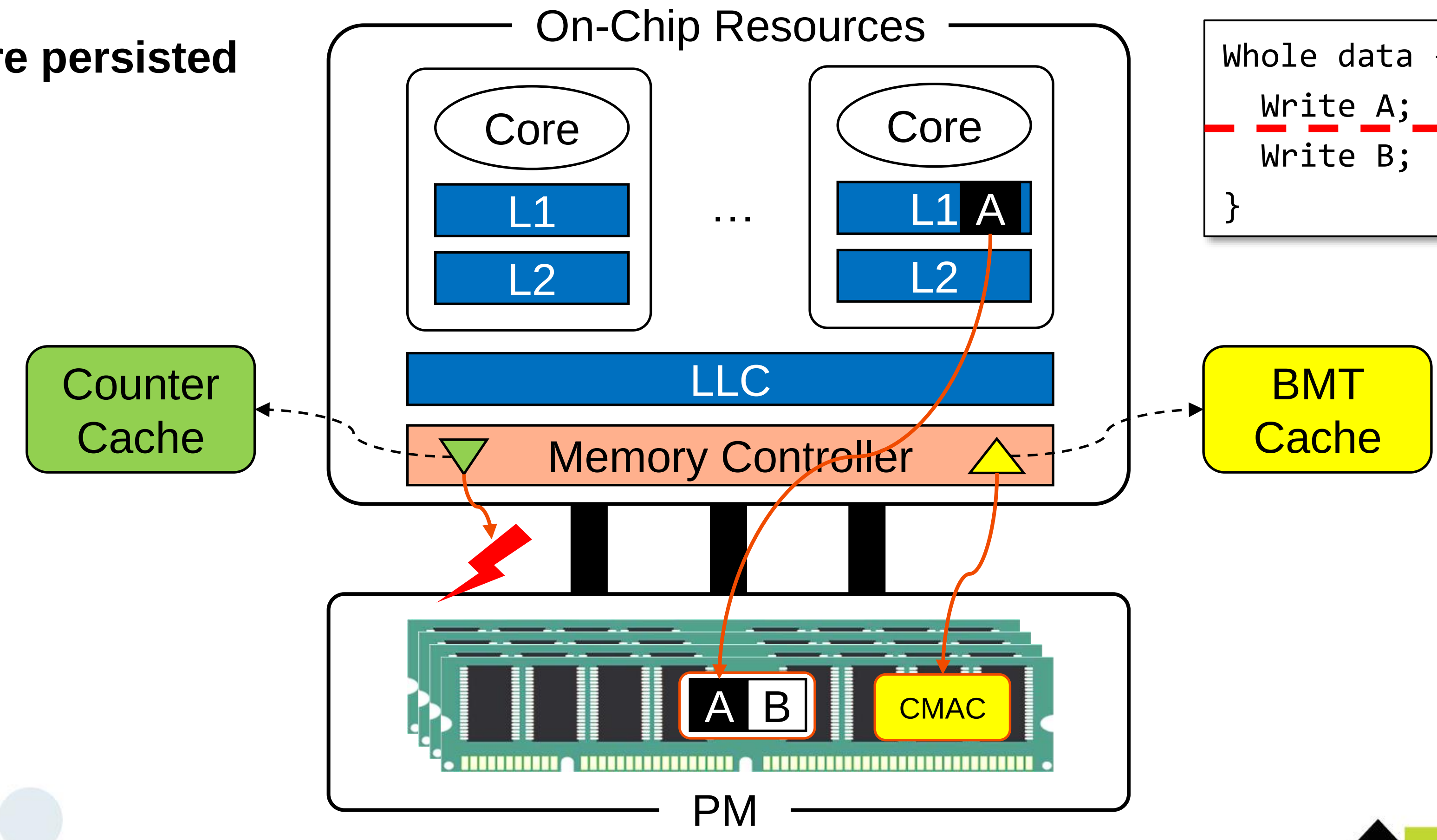
Inconsistency!



```
Whole data {
   Write A;
   Write B;
}
```

Crash Consistency for Secure PM

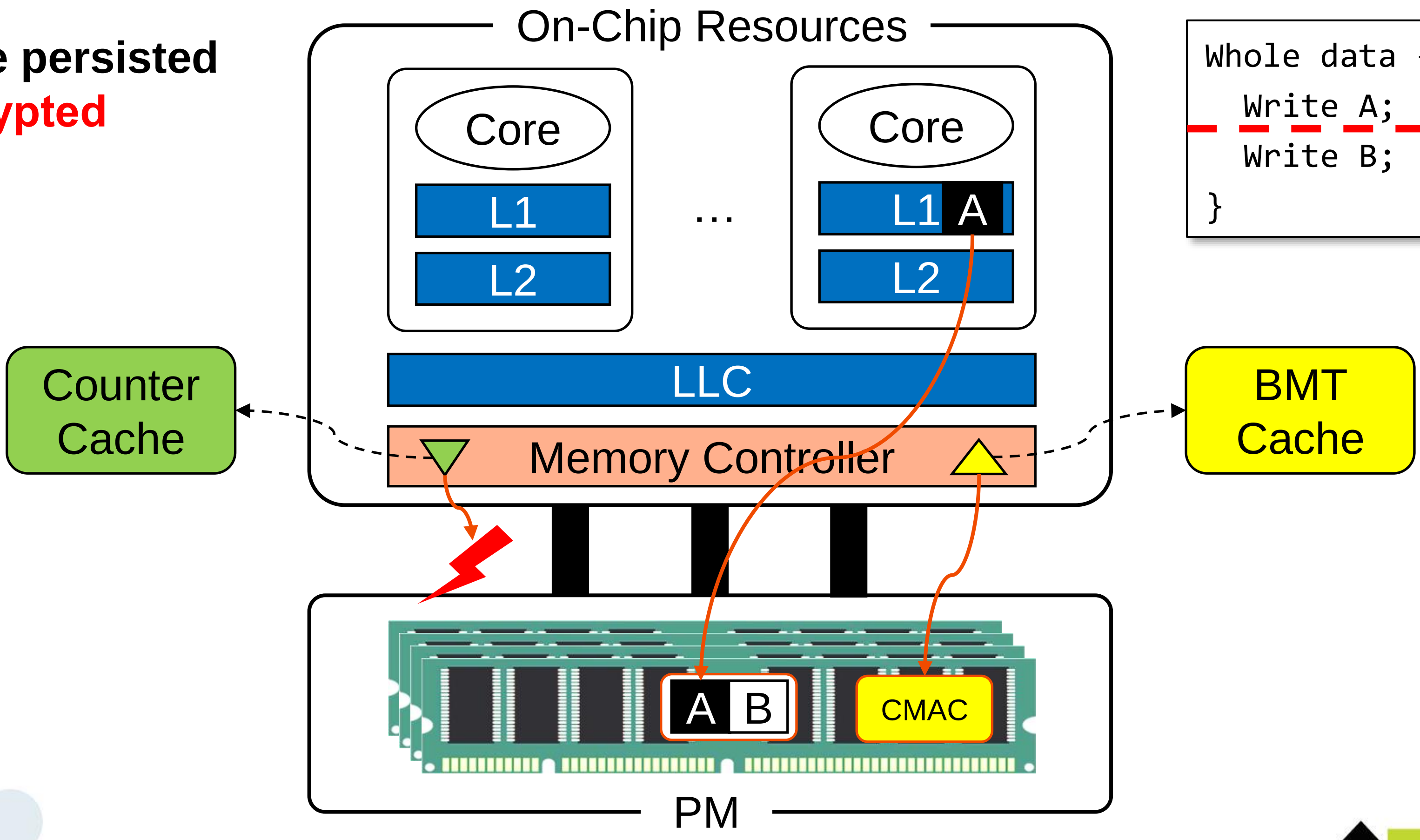
3) Data + CMAC are persisted

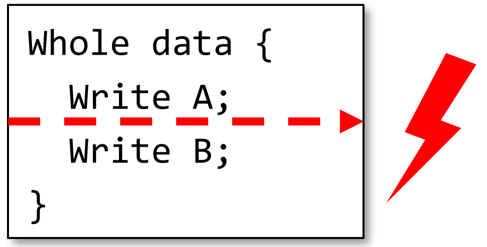


```
Whole data {
  Write A;
  Write B;
}
```

Crash Consistency for Secure PM

3) Data + CMAC are persisted
Can't be decrypted





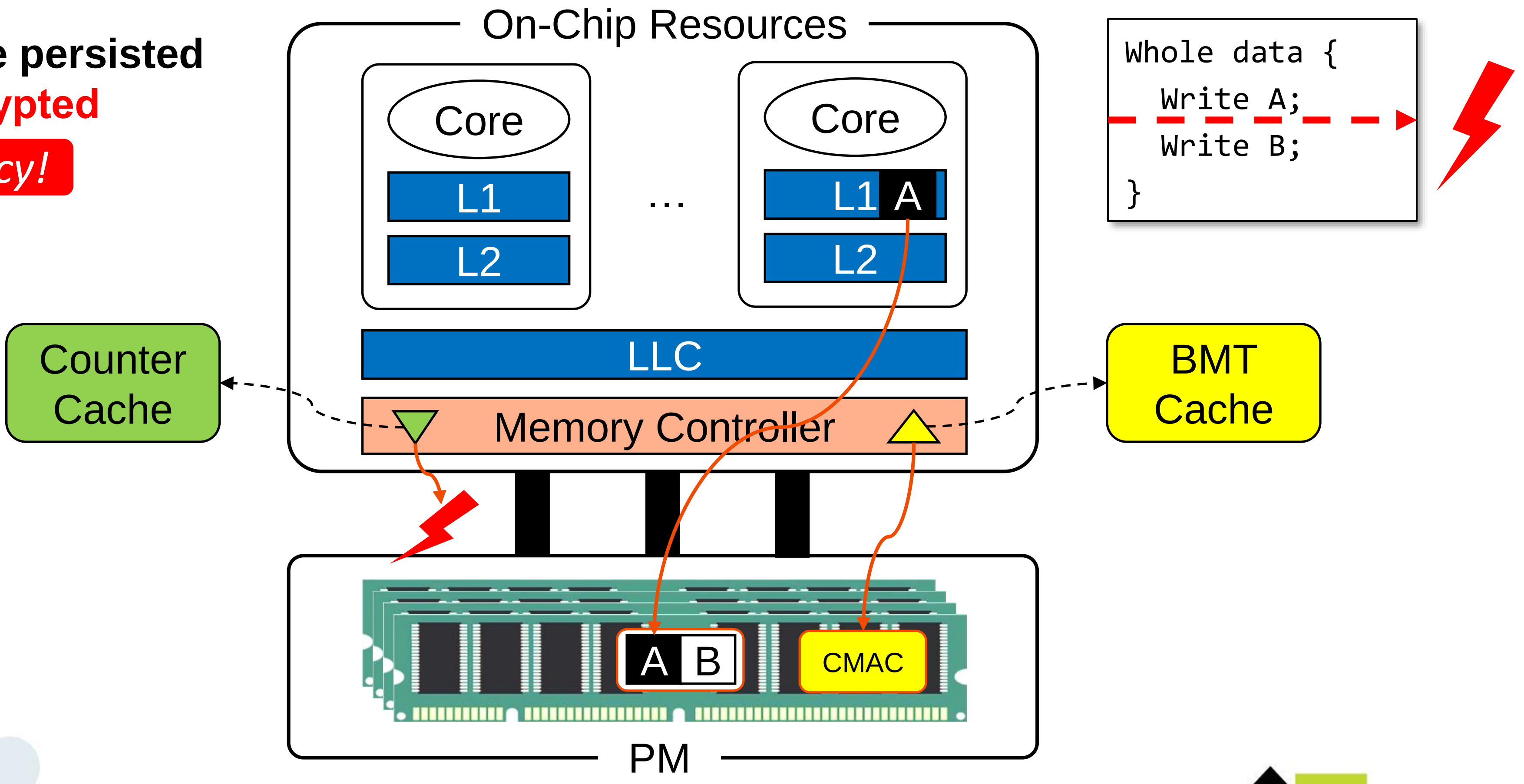
```
Whole data {
  Write A;
  Write B;
}
```

Crash Consistency for Secure PM

3) Data + CMAC are persisted

Can't be decrypted

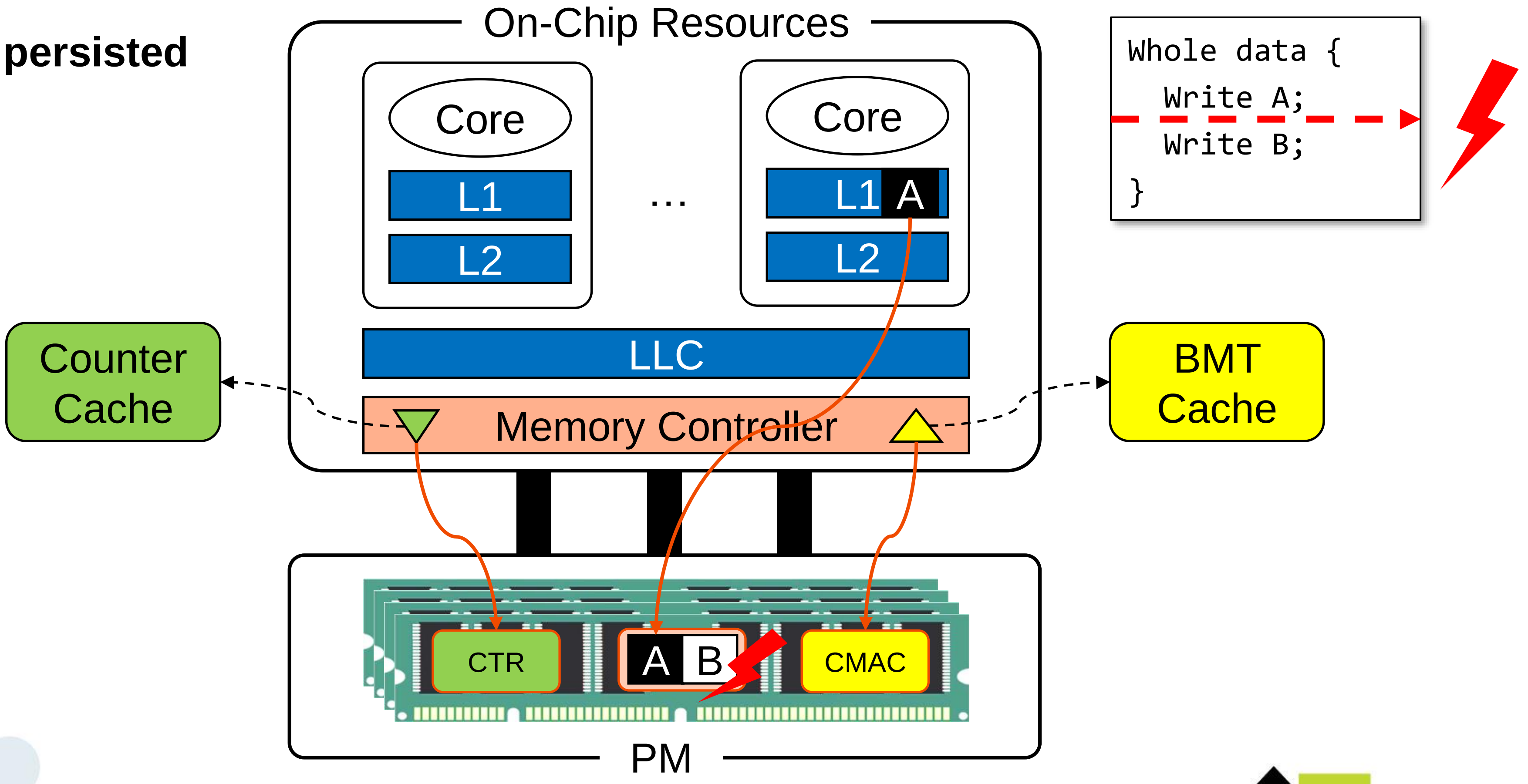
Inconsistency!



```
Whole data {
   Write A;
   Write B;
}
```

Crash Consistency for Secure PM

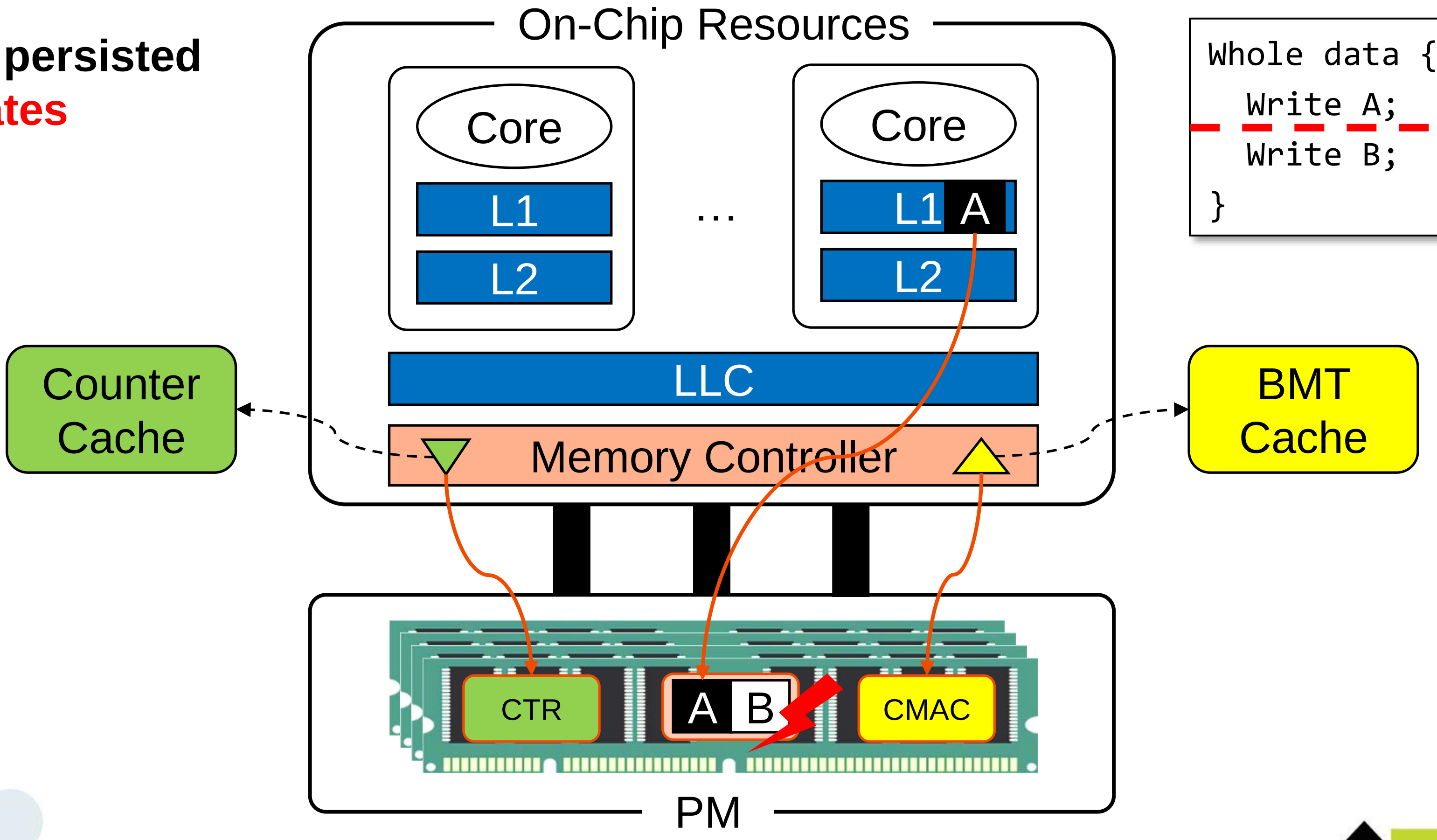
4) Part of data are persisted

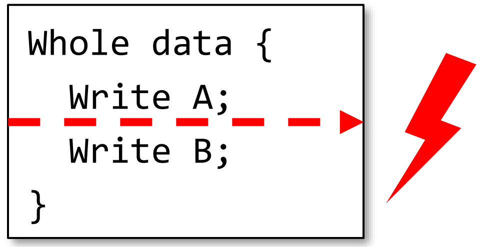


```
Whole data {
   Write A;
   Write B;
}
```

Crash Consistency for Secure PM

4) Part of data are persisted
Partial updates





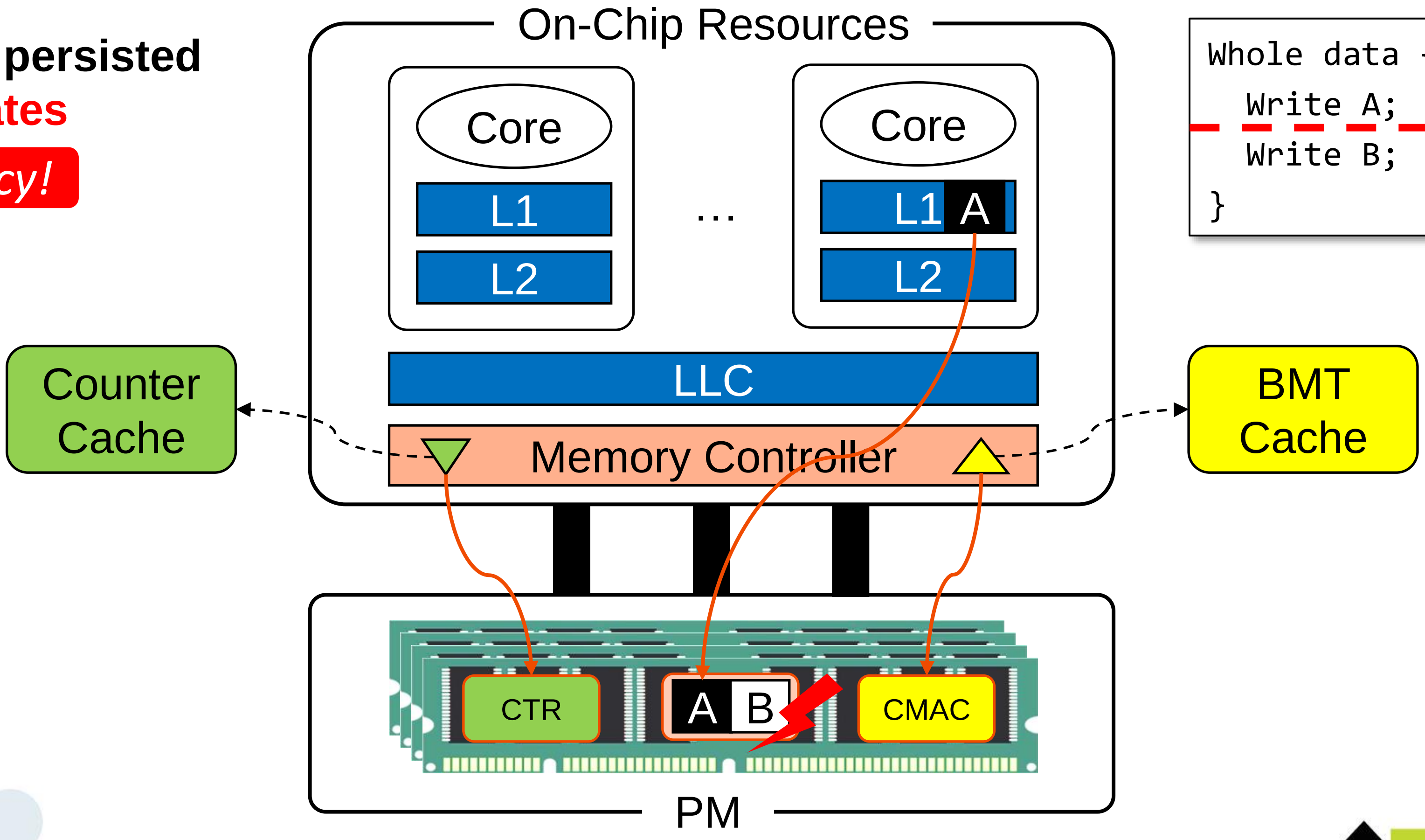
```
Whole data {
  Write A;
  Write B;
}
```

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



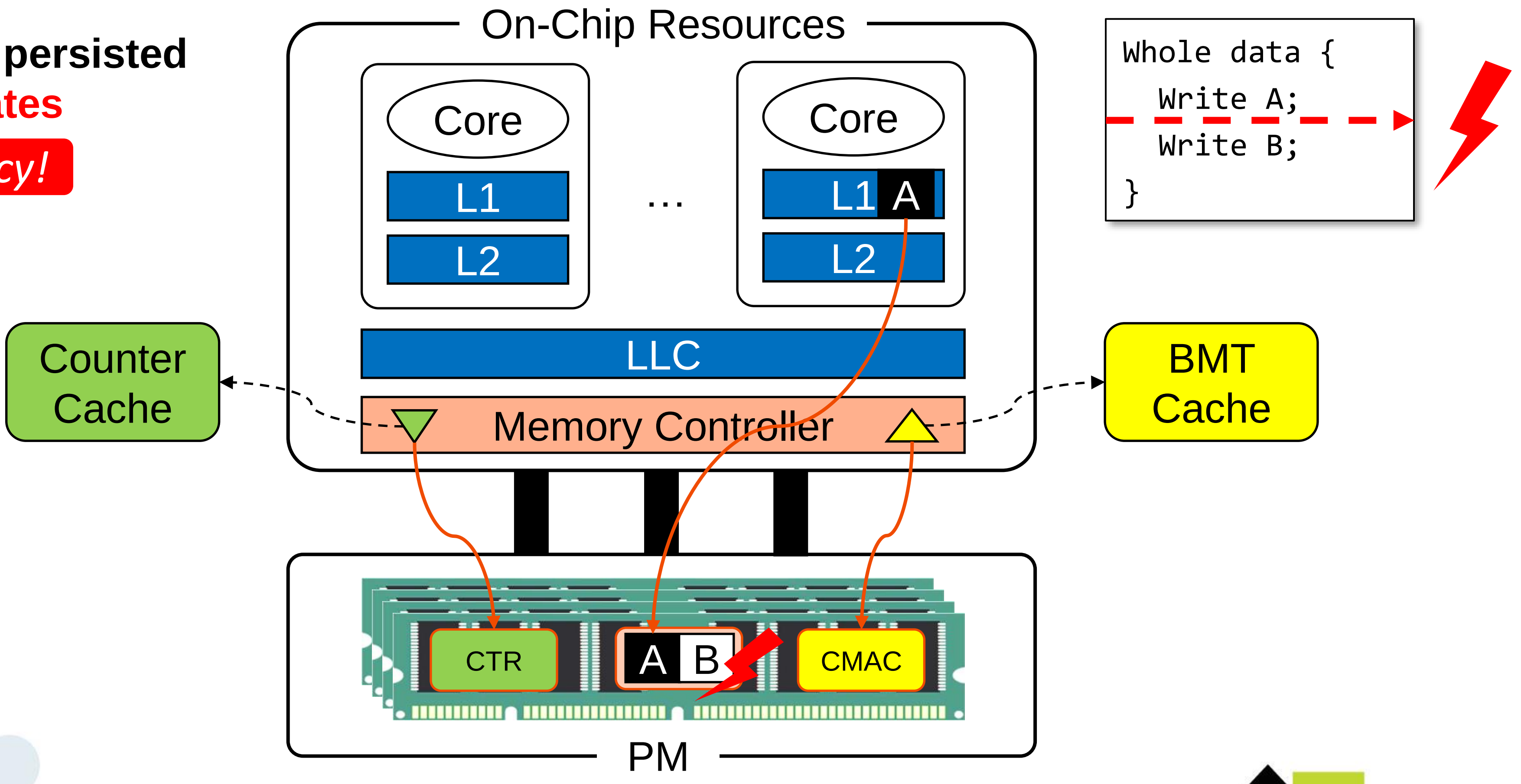
```
Whole data {
  Write A;
  Write B;
}
```

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



```
Whole data {
  Write A;
  Write B;
}
```

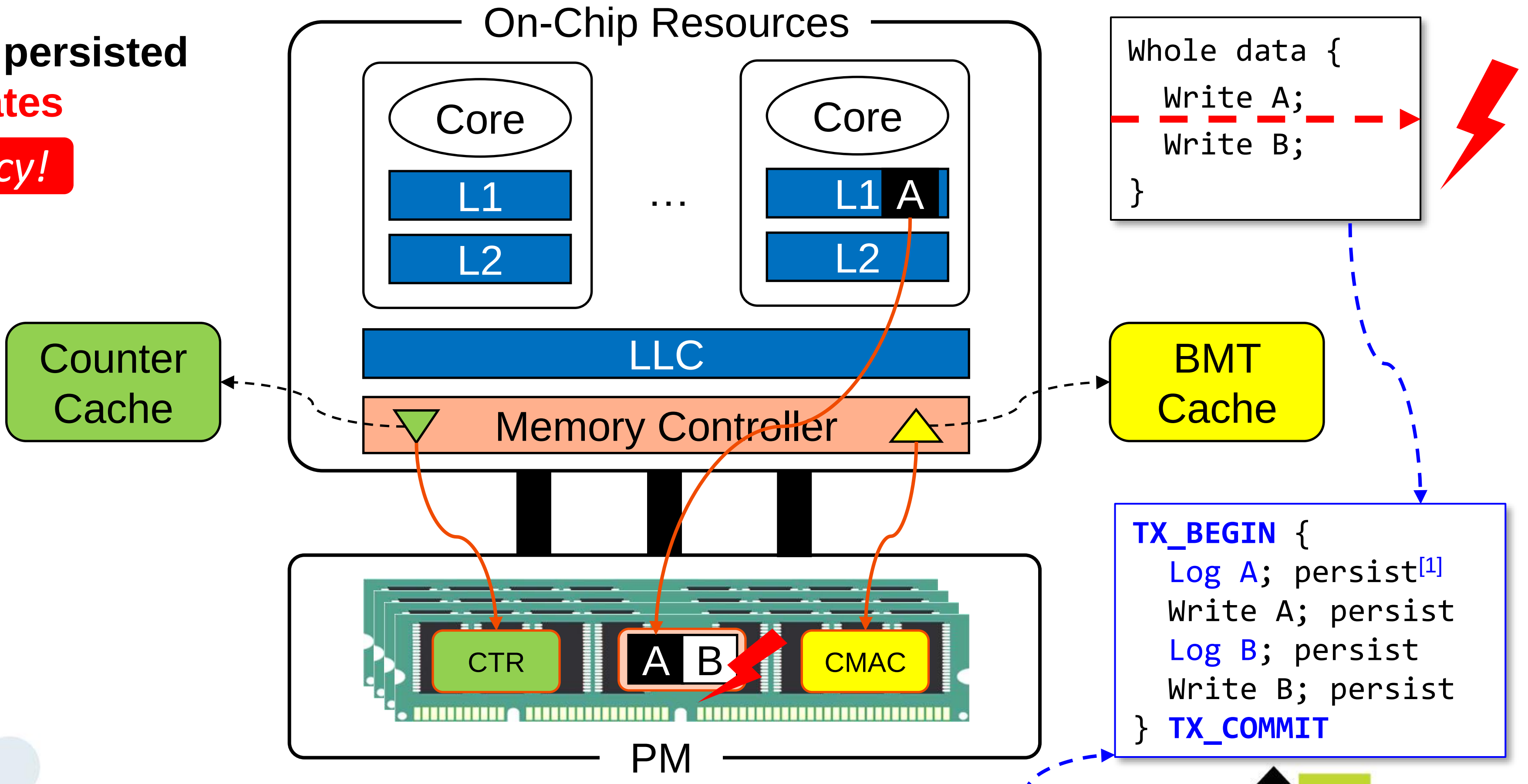
Guarantee failure-atomicity for
- ✓ A group of data
- ✓ Data + counter + CMAC

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



```
Whole data {
  Write A;
  Write B;
}
```

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

Durable Transaction with write-ahead logging

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

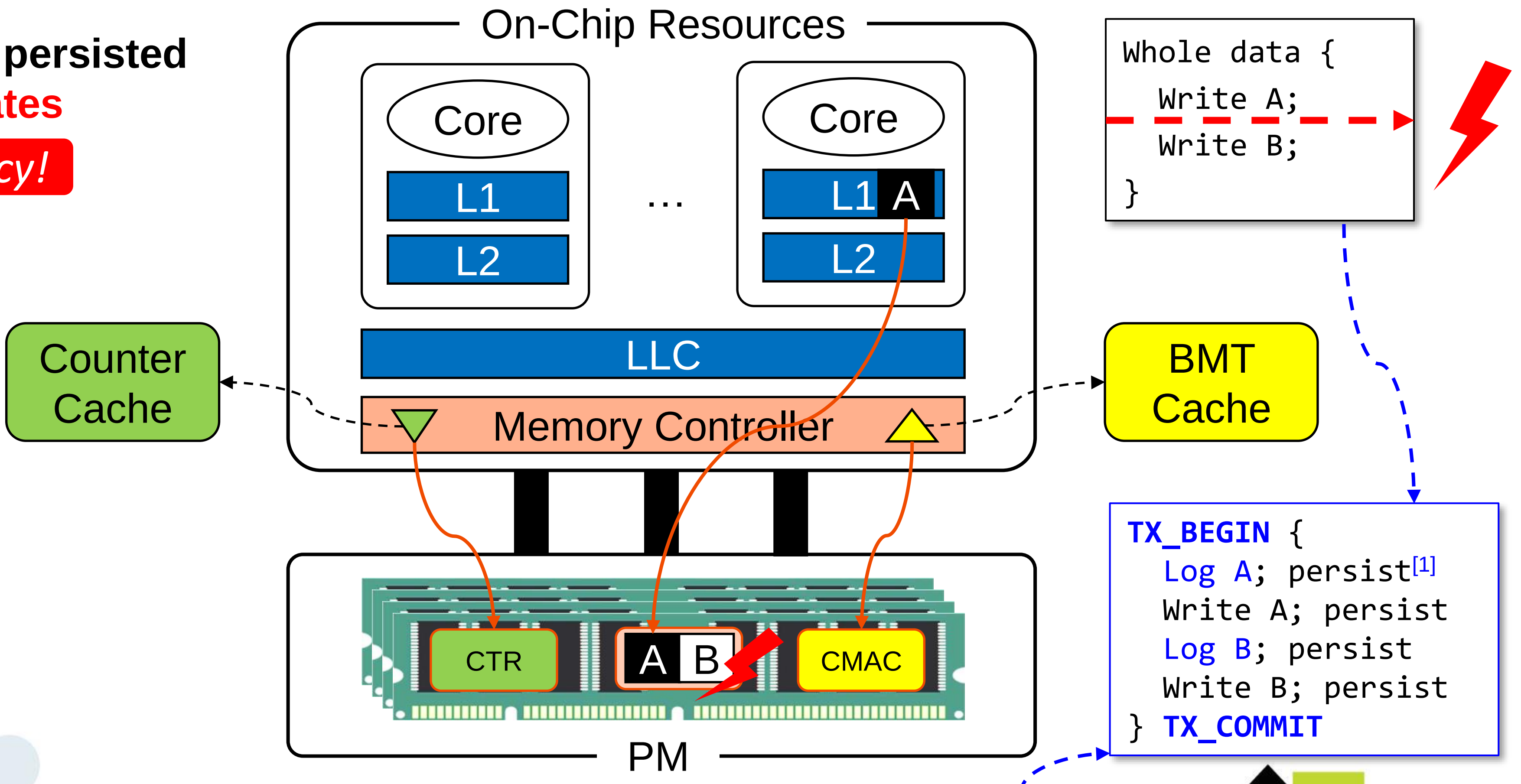
[1] Persist instruction sequence, e.g., clwb + sfence

Crash Consistency for Secure PM

4) Part of data are persisted

Partial updates

Inconsistency!



```
Whole data {
  Write A;
  Write B;
}
```

```
TX_BEGIN {
  Log A; persist[1]
  Write A; persist
  Log B; persist
  Write 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

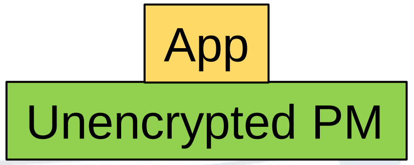
State-of-The-Art

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



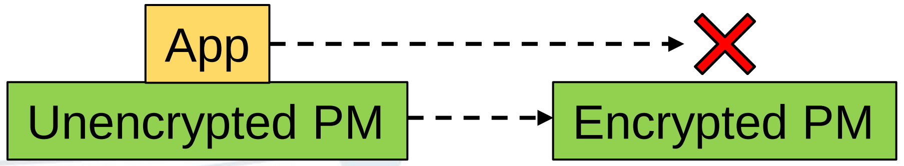
State-of-The-Art

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



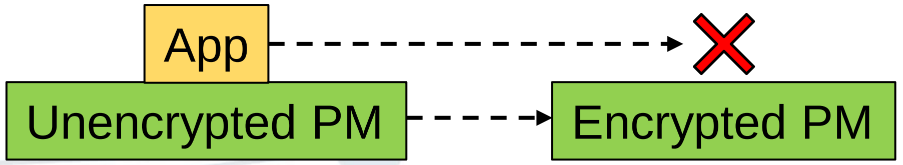
State-of-The-Art

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
- A register appends <data+counter> to write queue
  - Application transparent → Good portability

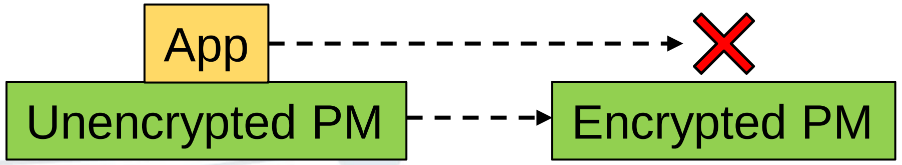
State-of-The-Art

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()

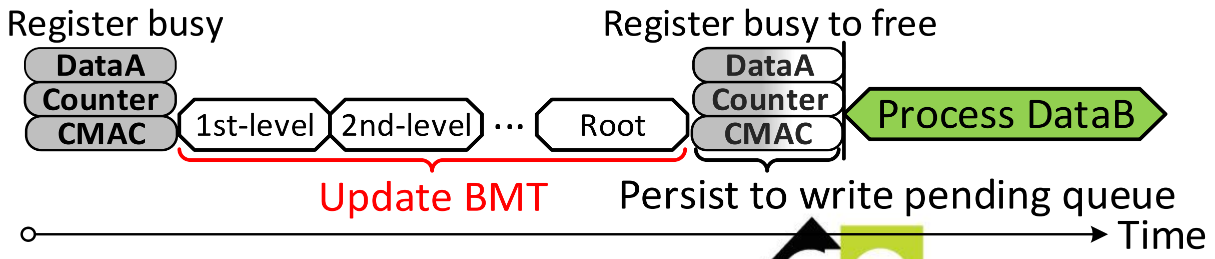
➔ Limited portability



SuperMem@MICRO'19

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

➔ Limited scalability

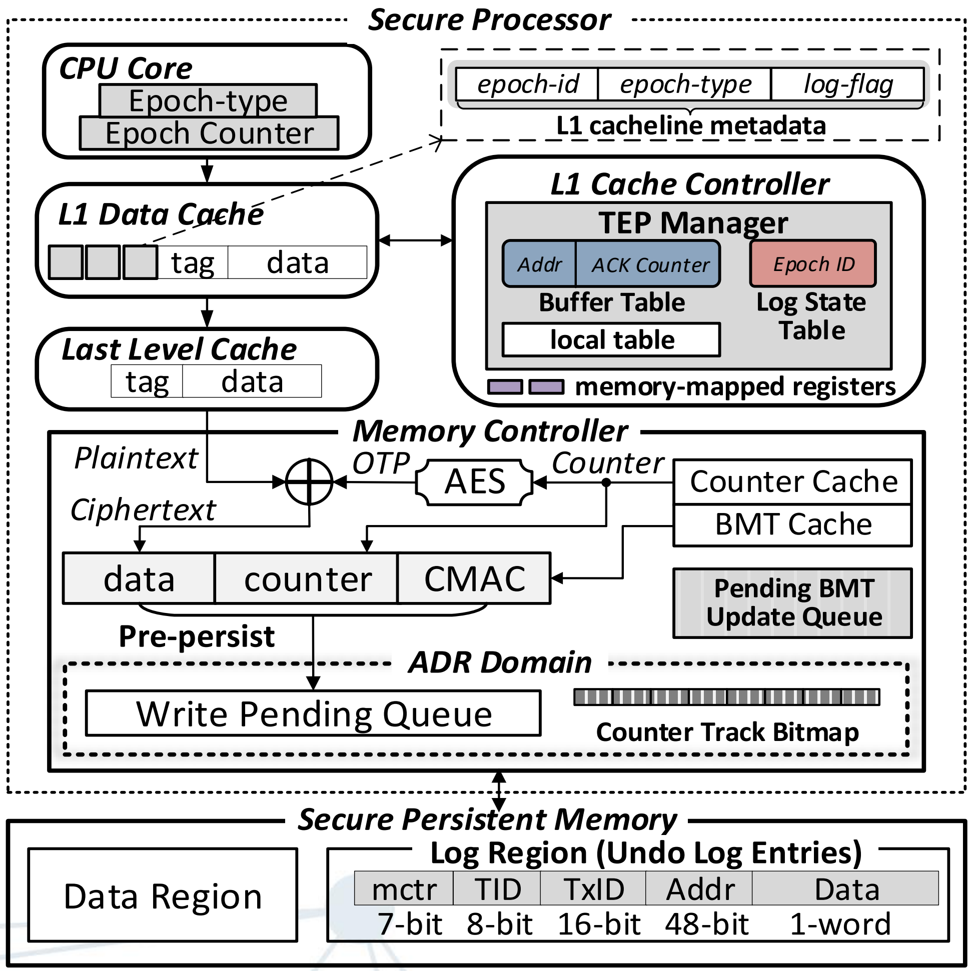


Secon

- Security and crash consistency for PM
- Goal

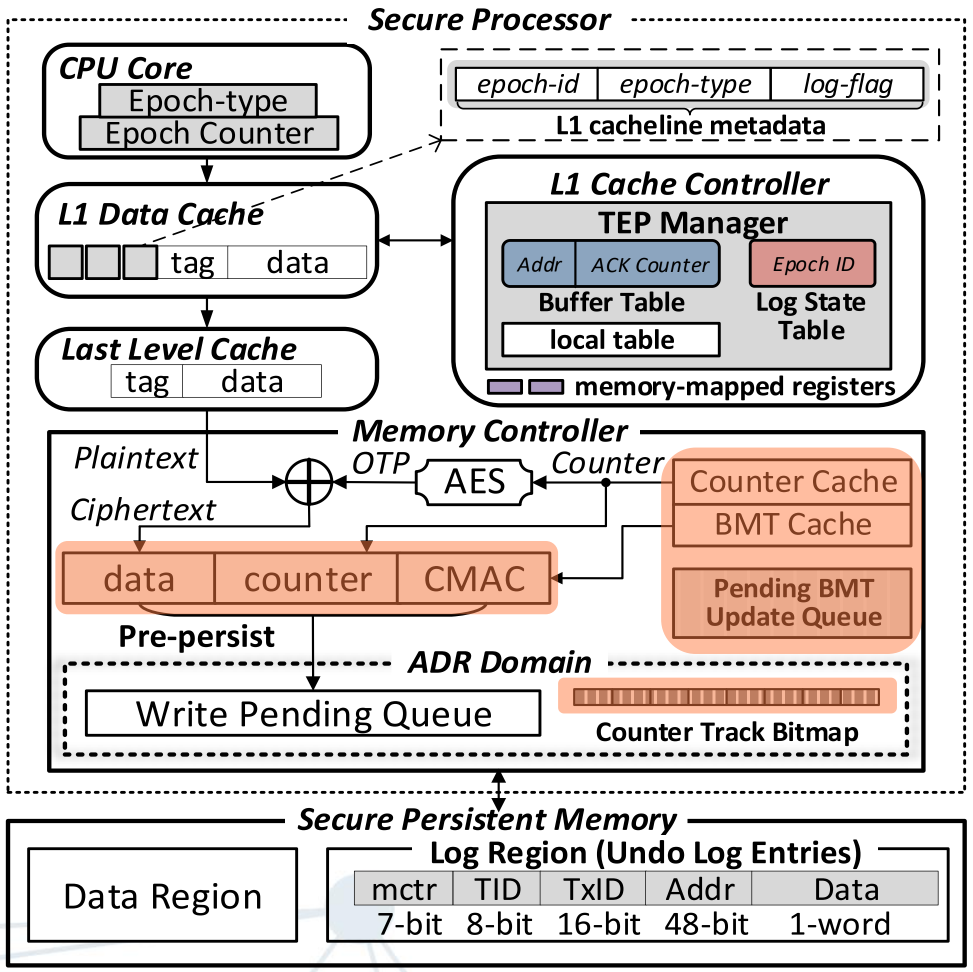
Design	Confidentiality	Integrity	Atomicity for a group of updates	Atomicity of data and its security metadata
SCA@HPCA'18	✓	✗	✓	Data + Counter
SuperMem@MICRO'19	✓	✗	✓	Data + Counter
Our Secon	✓	✓	✓	**Data + Counter + CMAC**

Architecture



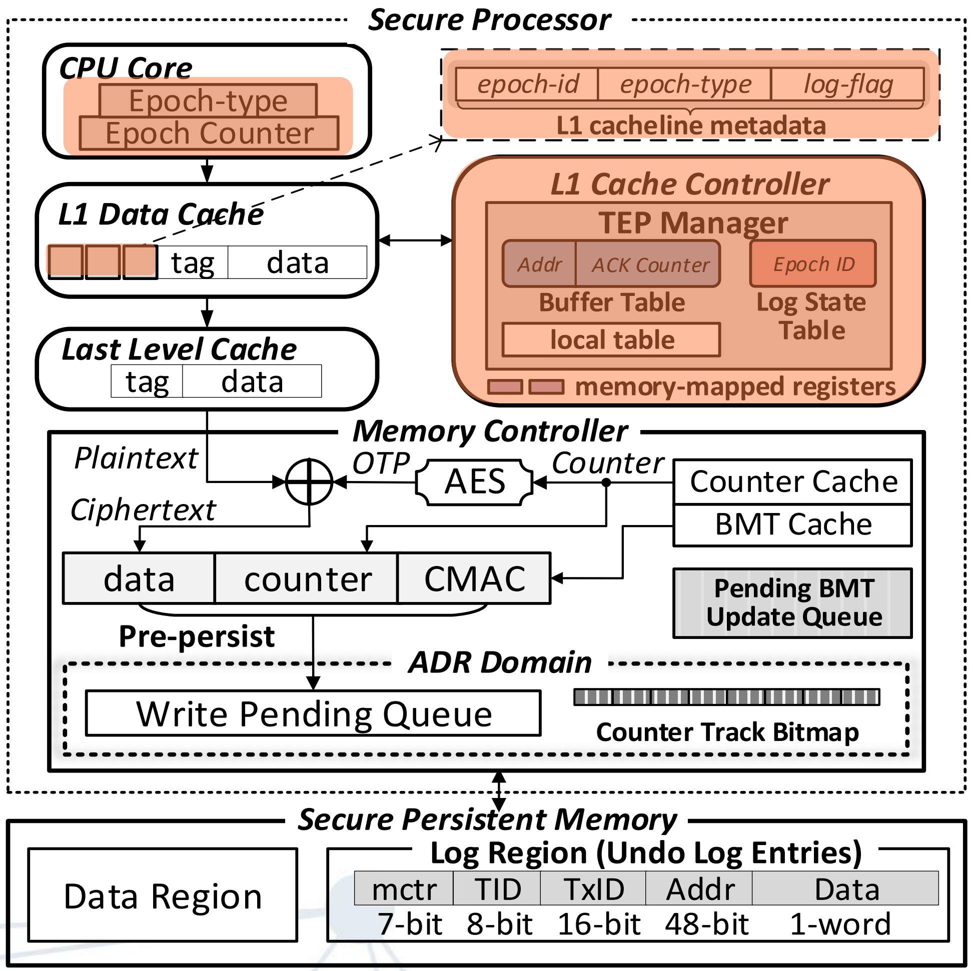
- **Scalable write-through security metadata cache**
  - Move BMT update to the background
- **Transaction-specific epoch persistency model**
  - Minimize ordering constraints between logs and data
- **Security metadata write-reduction schemes**
  - Mitigate the writes caused by counters and CMACs

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

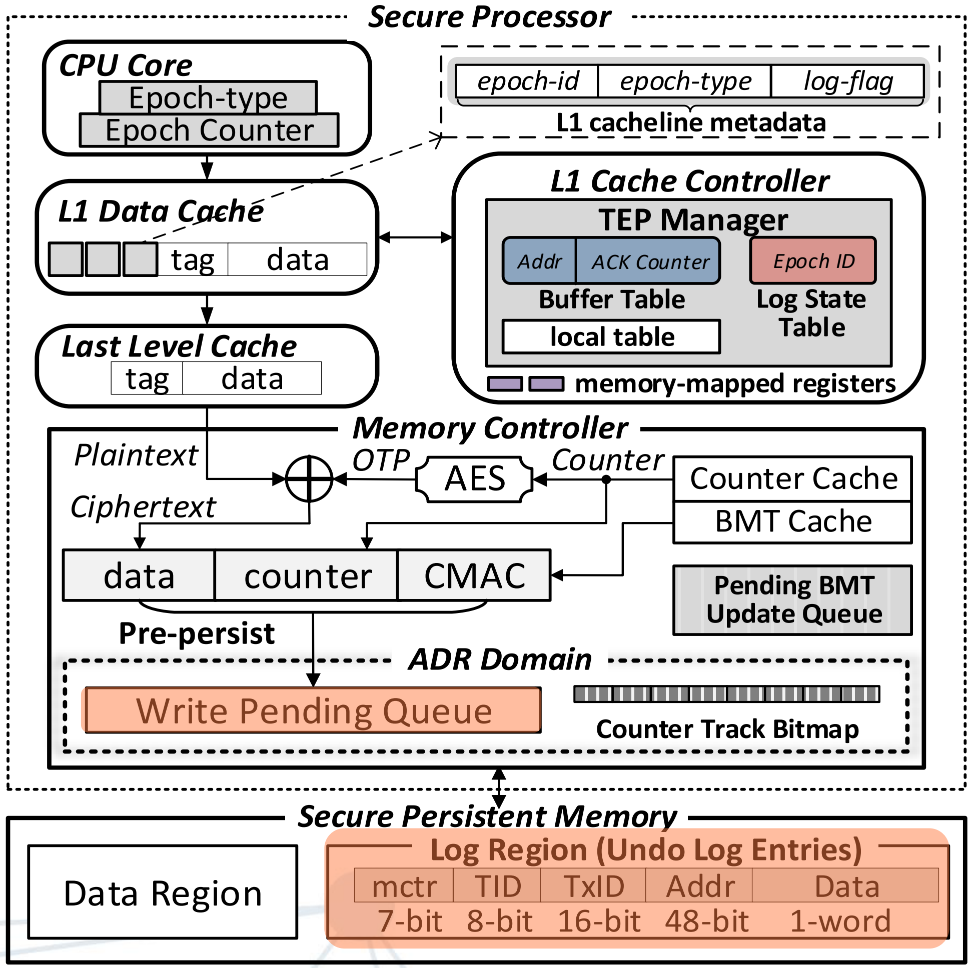
Architecture



- 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

DESIGN AUTOMATION CONFERENCE 59

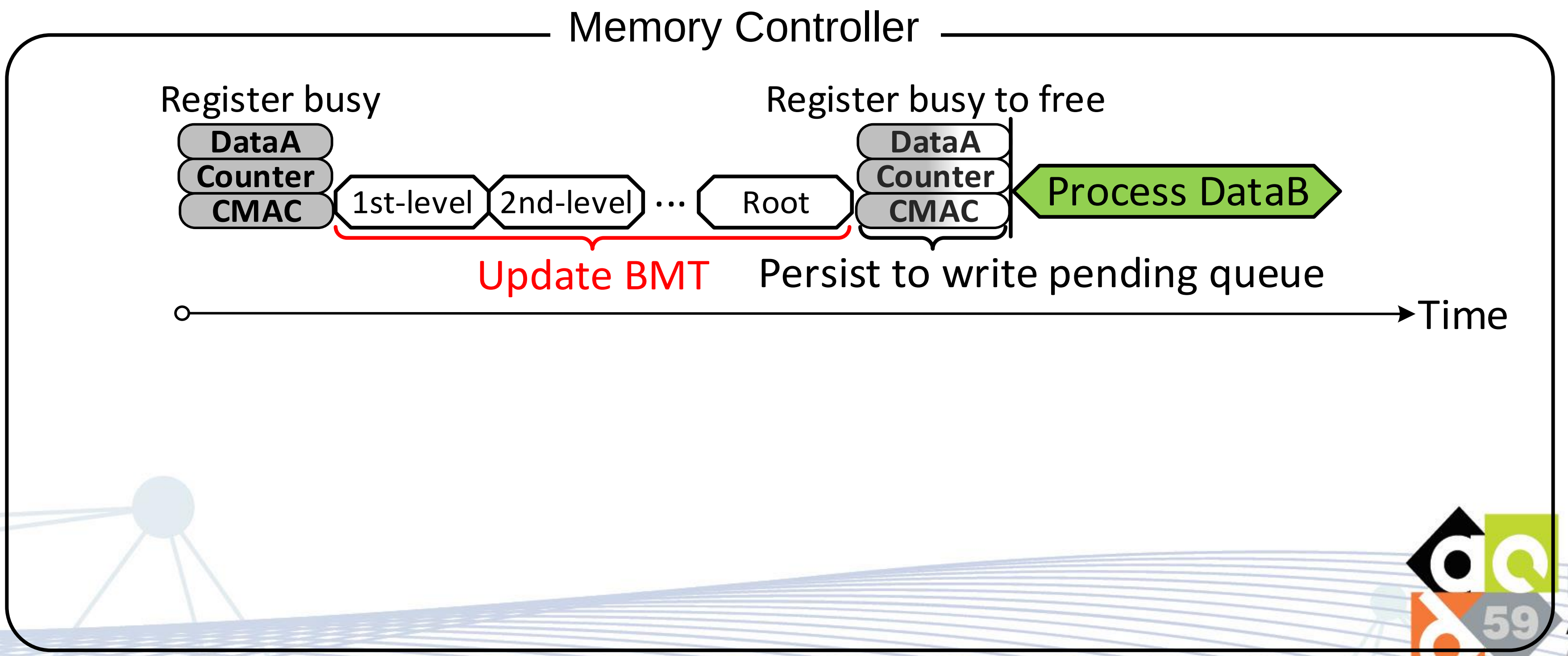
Architecture



- 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

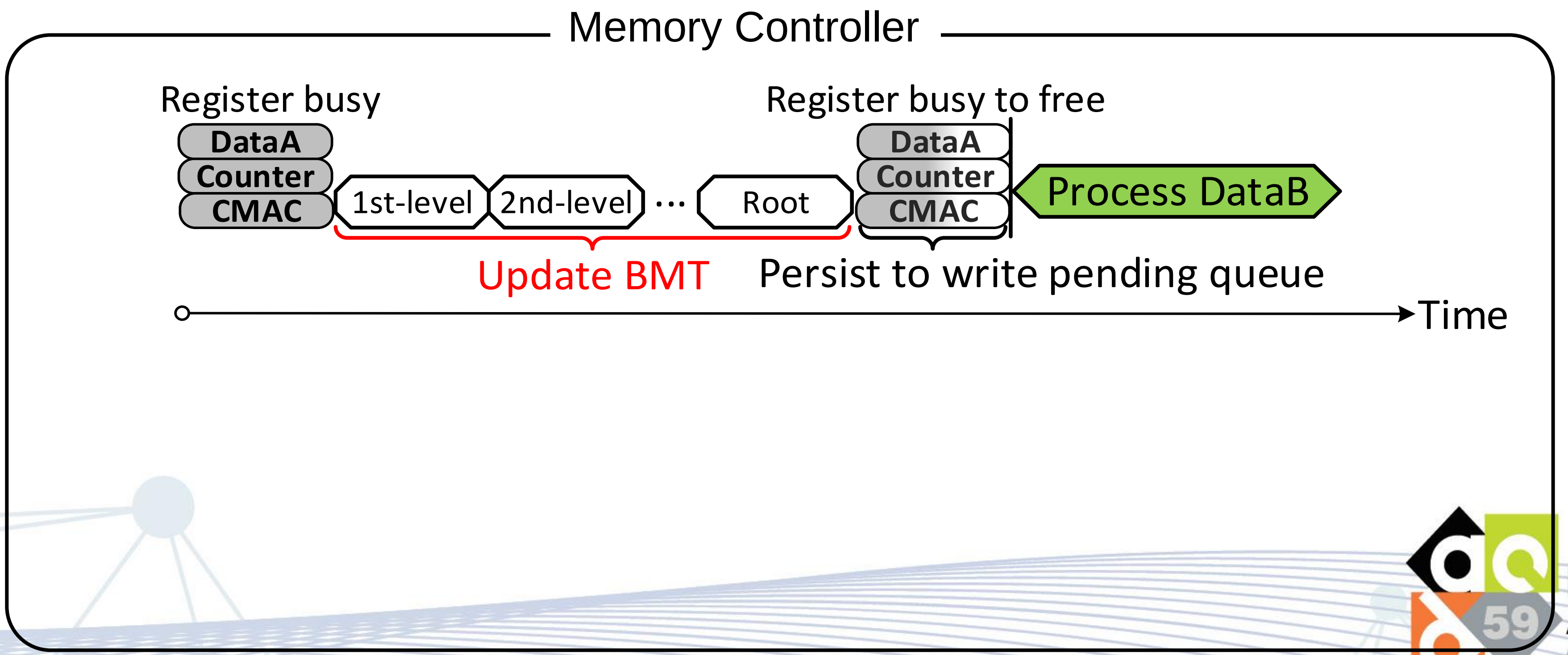
DESIGN AUTOMATION CONFERENCE 59

Scalable Write-Through Security Metadata Cache



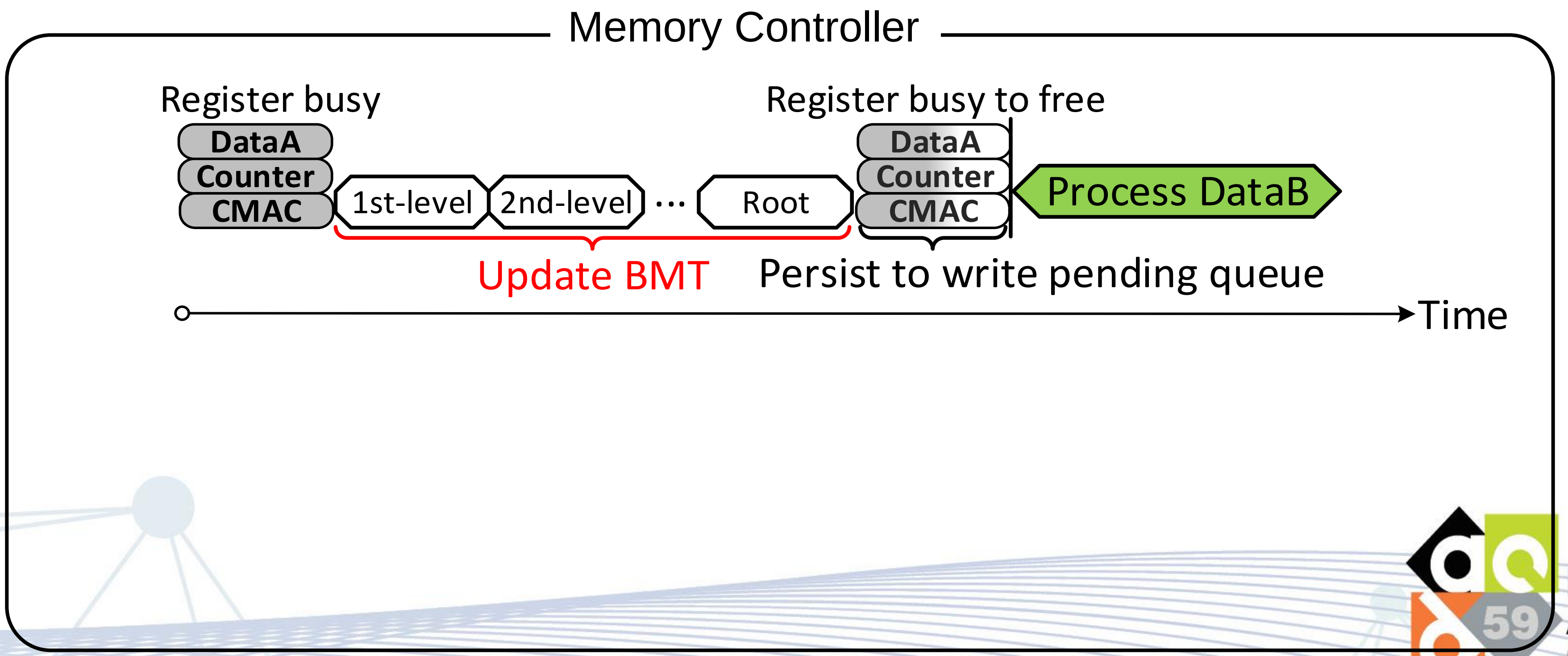
Scalable Write-Through Security Metadata Cache

- **Observation:** PM always has a consistent copy of data by logging



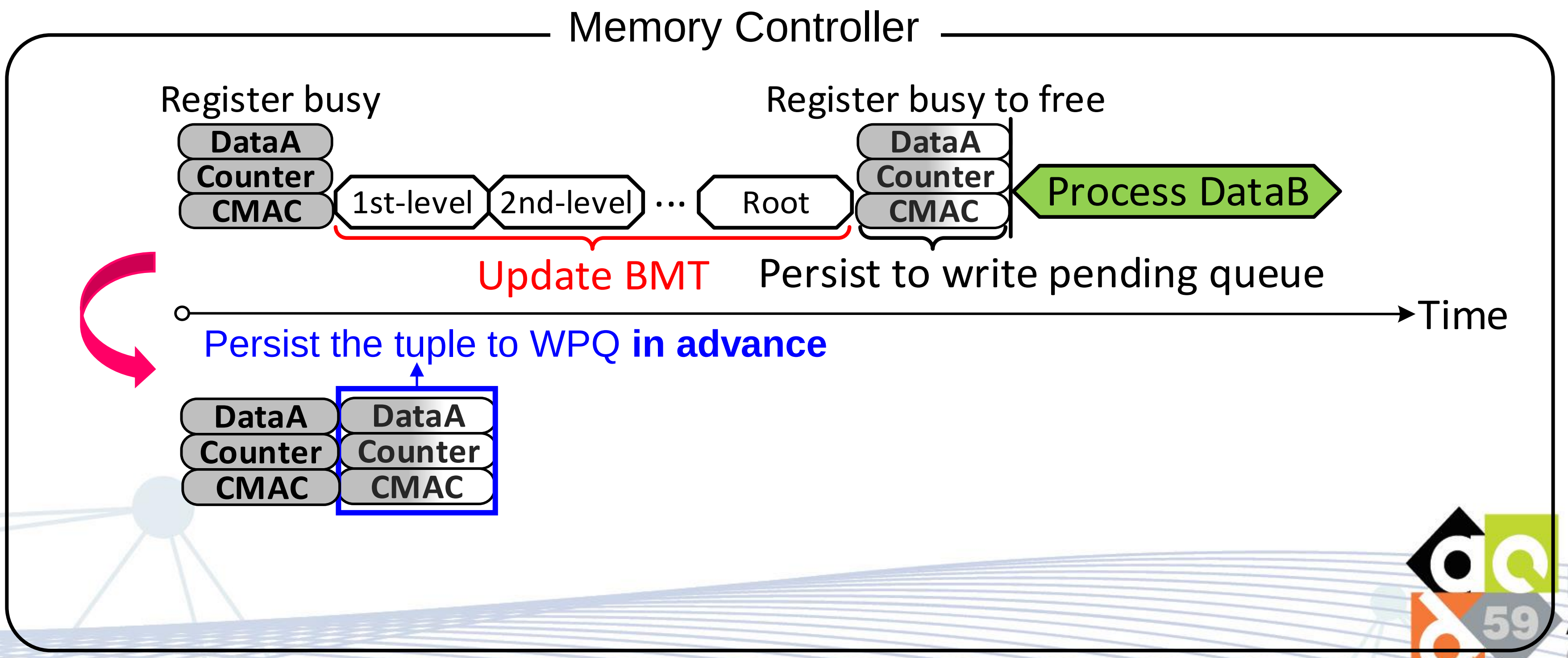
Scalable Write-Through Security Metadata Cache

- **Observation:** PM always has a consistent copy of data by logging
  - In the log region or data region



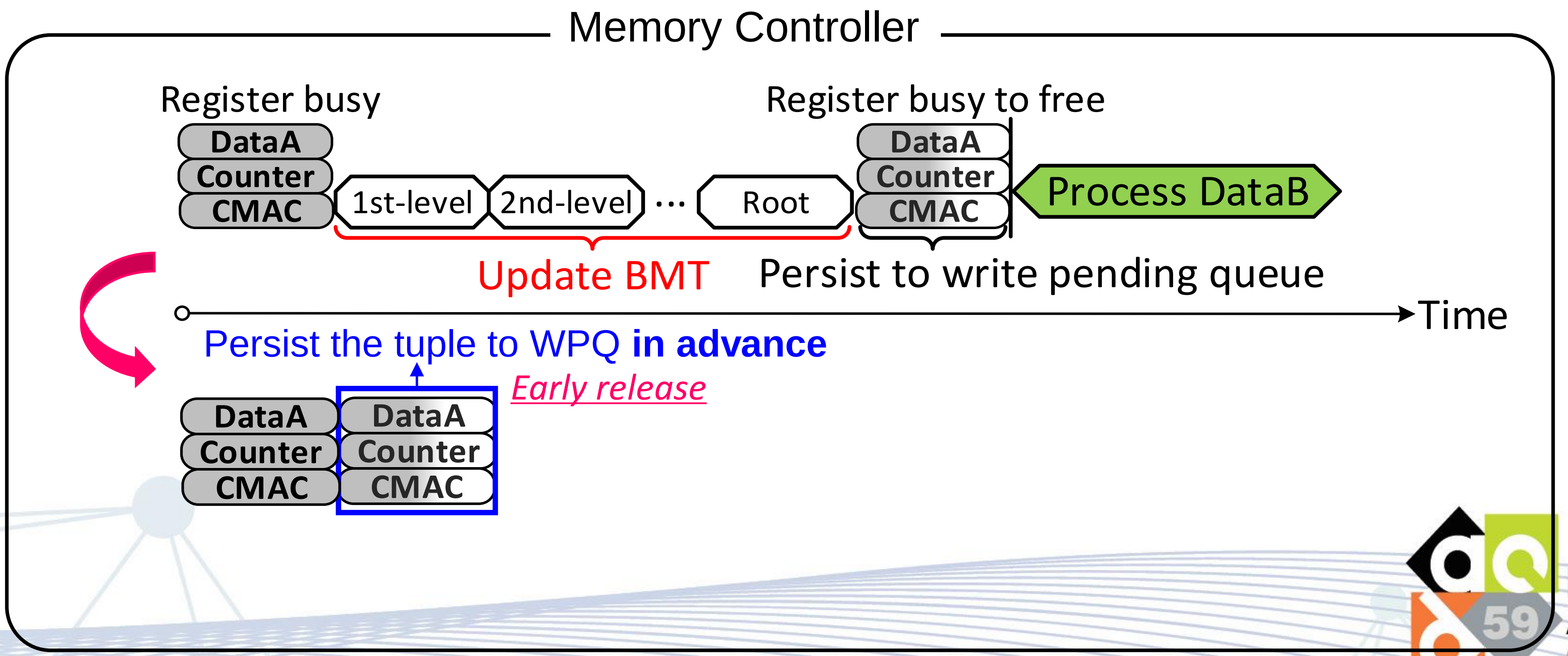
Scalable Write-Through Security Metadata Cache

- **Observation:** PM always has a consistent copy of data by logging
  - In the log region or data region
- Persist the tuple of <data, counter, CMAC> **in advance**



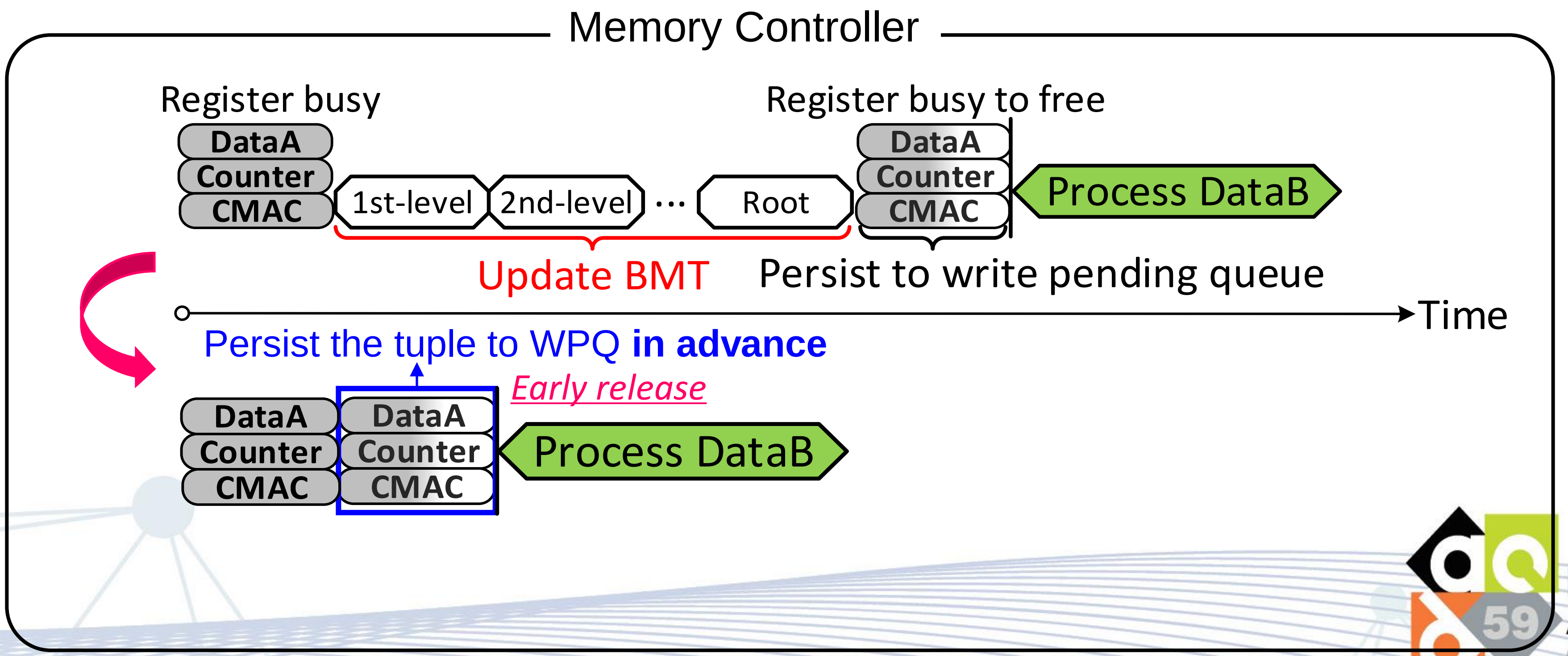
Scalable Write-Through Security Metadata Cache

- **Observation:** PM always has a consistent copy of data by logging
  - In the log region or data region
- Persist the tuple of <data, counter, CMAC> **in advance**
  - Release the register early to process the next independent write request



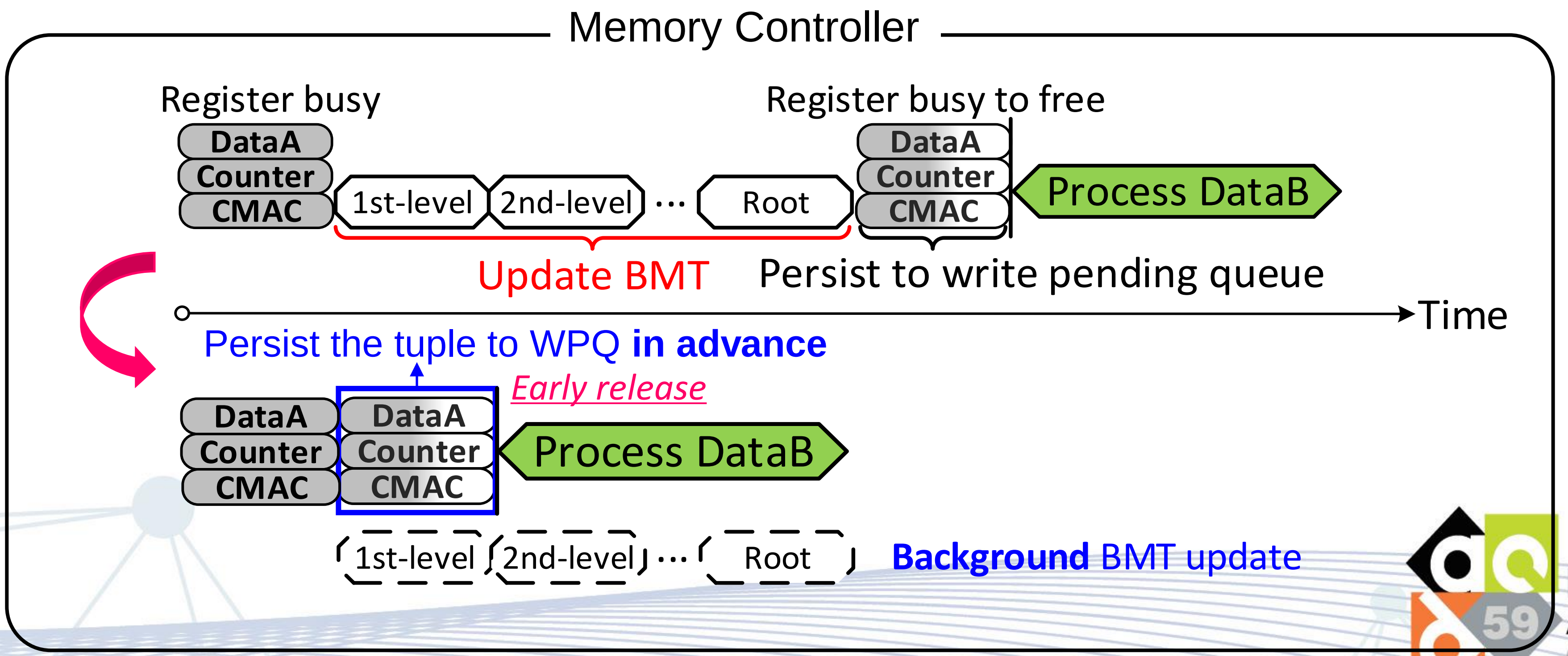
Scalable Write-Through Security Metadata Cache

- **Observation:** PM always has a consistent copy of data by logging
  - In the log region or data region
- Persist the tuple of <data, counter, CMAC> **in advance**
  - Release the register early to process the next independent write request



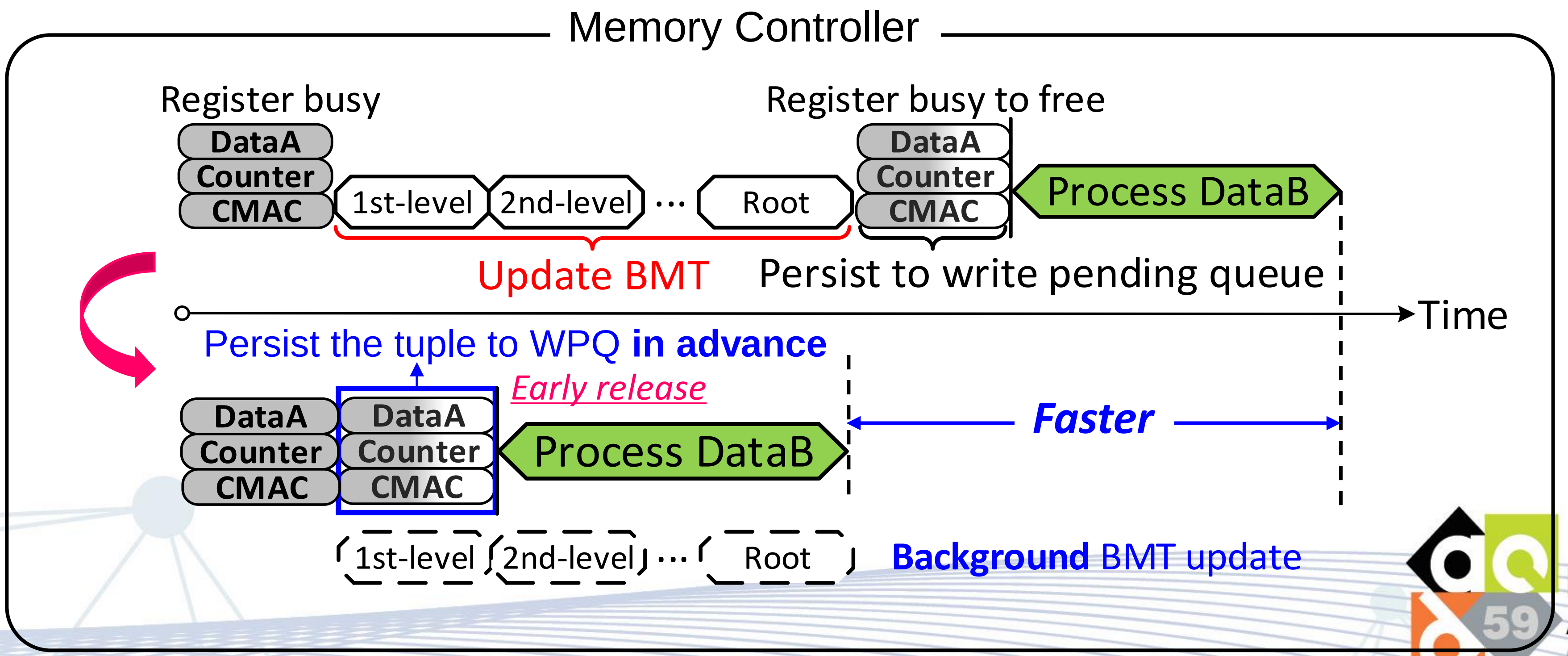
Scalable Write-Through Security Metadata Cache

- **Observation:** PM always has a consistent copy of data by logging
  - In the log region or data region
- Persist the tuple of <data, counter, CMAC> **in advance**
  - Release the register early to process the next independent write request
  - Move BMT update to the background



Scalable Write-Through Security Metadata Cache

- **Observation:** PM always has a consistent copy of data by logging
  - In the log region or data region
- Persist the tuple of <data, counter, CMAC> **in advance**
  - Release the register early to process the next independent write request
  - Move BMT update to the background



Scalable Write-Through Security Metadata Cache

- Guarantee the consistency between on-chip BMT root and off-chip counters after a crash

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) – which CMAC is updated

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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which 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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which counter is updated

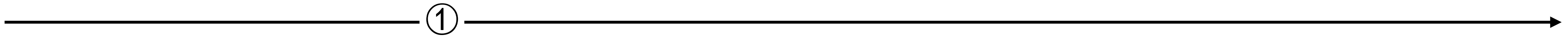
[Example] The counter and CMAC of current write request are respectively **mc_1** and **$CMAC_1$**

[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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which counter is updated

[Example] The counter and CMAC of current write request are respectively **mc_1** and **$CMAC_1$**

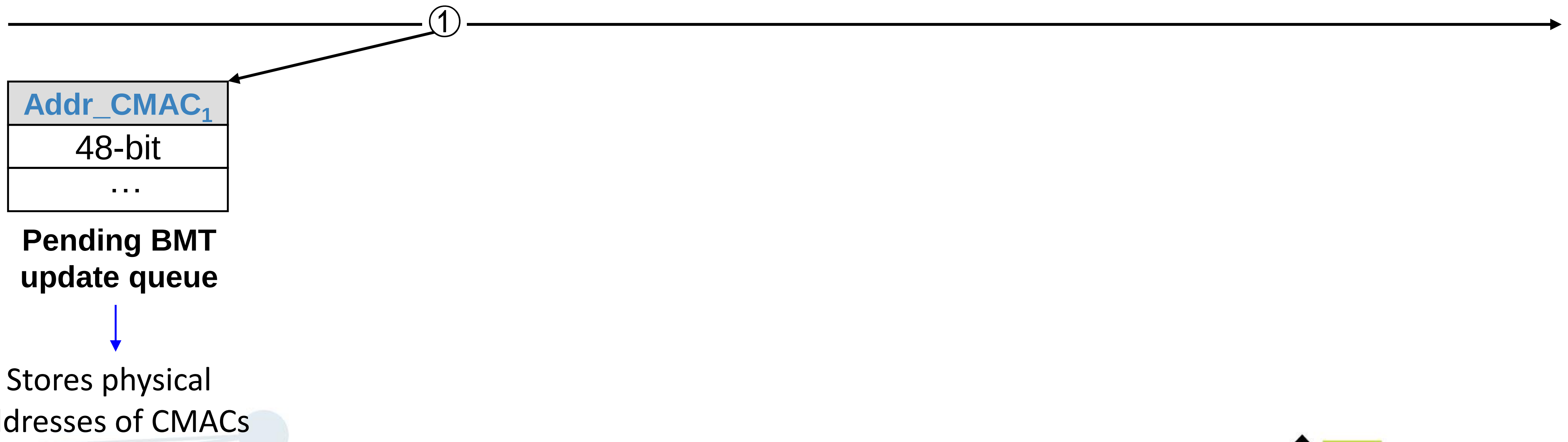


[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which counter is updated

[Example] The counter and CMAC of current write request are respectively **mc_1** and **$CMAC_1$**

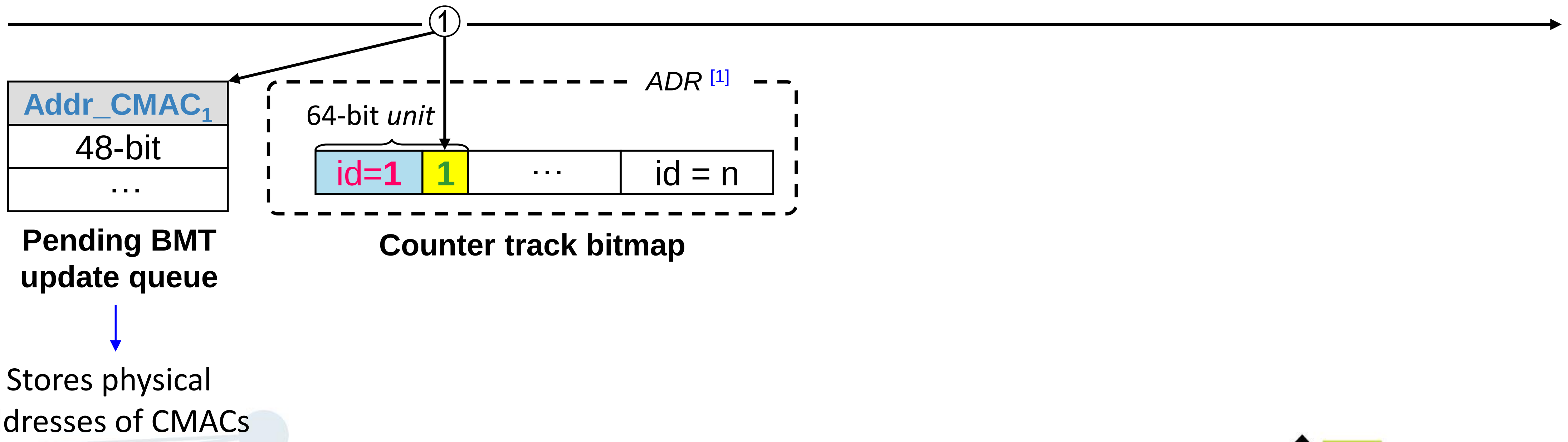


[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which counter is updated

[Example] The counter and CMAC of current write request are respectively **mc_1** and **$CMAC_1$**

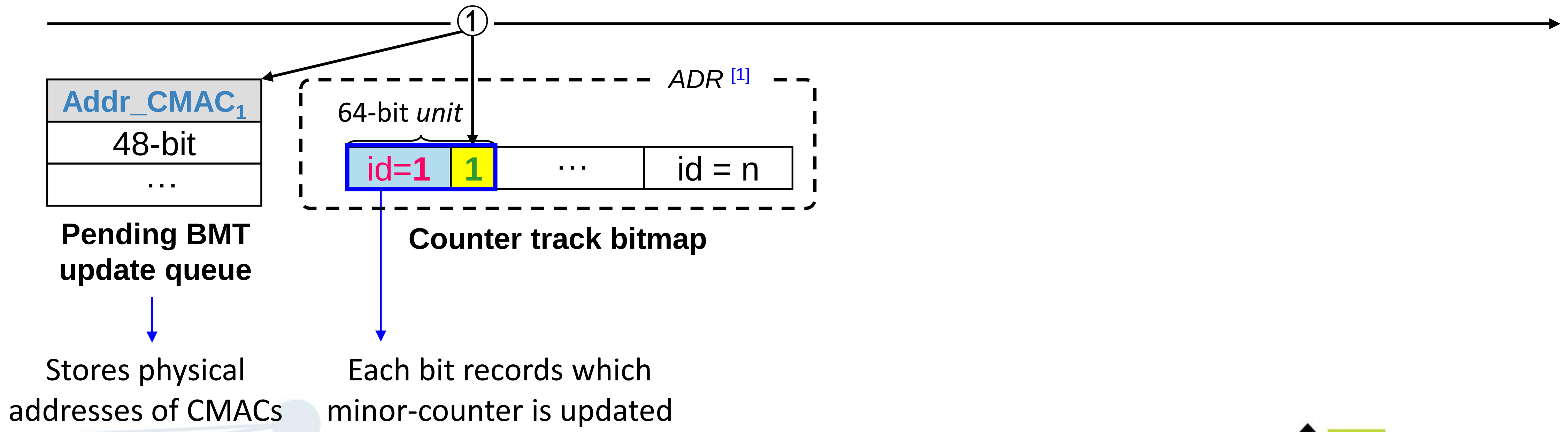


[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which counter is updated

[Example] The counter and CMAC of current write request are respectively **$\mathbf{mc_1}$** and **$\mathbf{CMAC_1}$**

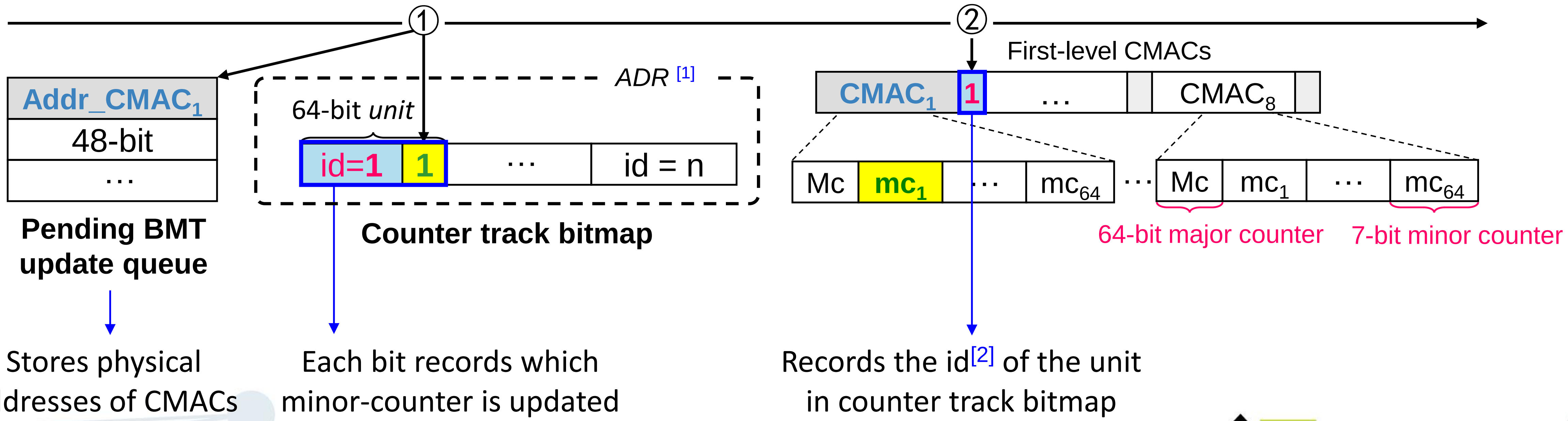


[1] The **A**synchronous **D**RAM **R**efresh domain, in which the internal data survive a crash or power failure

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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which counter is updated

[Example] The counter and CMAC of current write request are respectively **mc_1** and **$CMAC_1$**

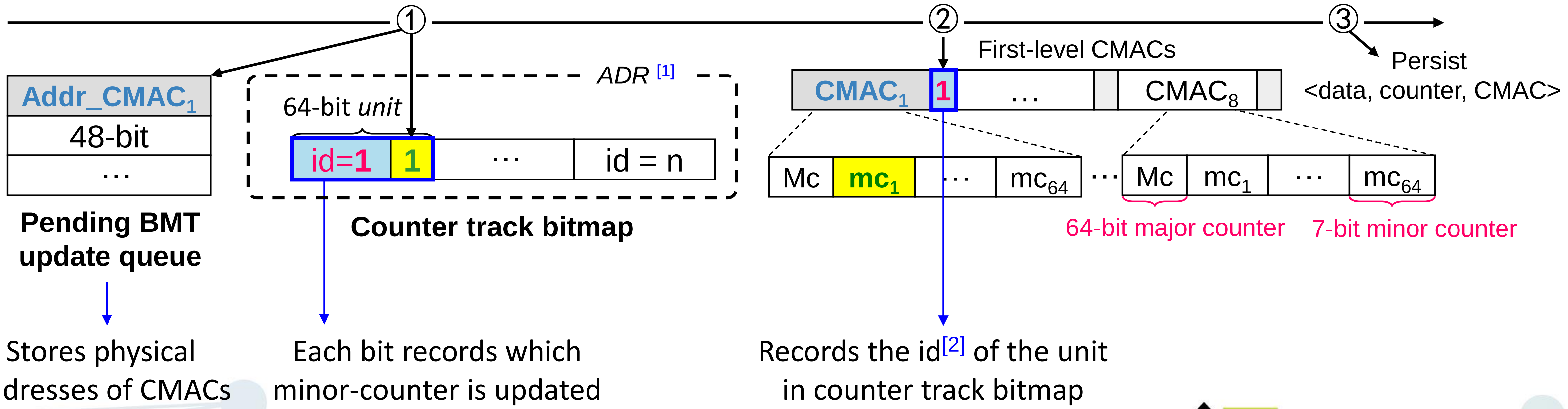


[1] The Asynchronous DRAM Refresh domain, in which the internal data survive a crash or power failure
[2] Only using 54 bits of a 64-bit CMAC is sufficiently secure (Morphable Counters@MICRO'18)

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) – which CMAC is updated
  - Counter track bitmap (In ADR[1] of MC) – which counter is updated

[Example] The counter and CMAC of current write request are respectively **mc_1** and **$CMAC_1$**



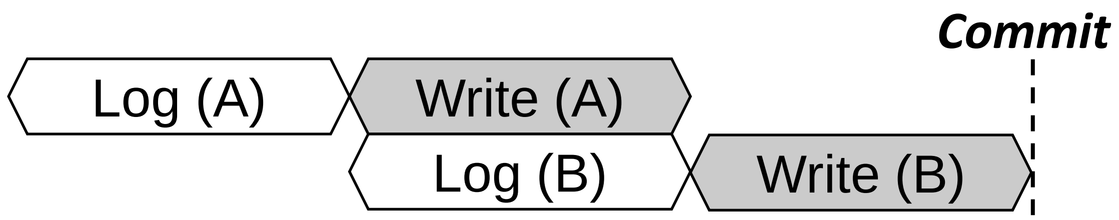
[1] The **A**synchronous **DR**AM **R**efresh domain, in which the internal data survive a crash or power failure
[2] Only using 54 bits of a 64-bit CMAC is sufficiently secure (Morphable Counters@MICRO'18)

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]



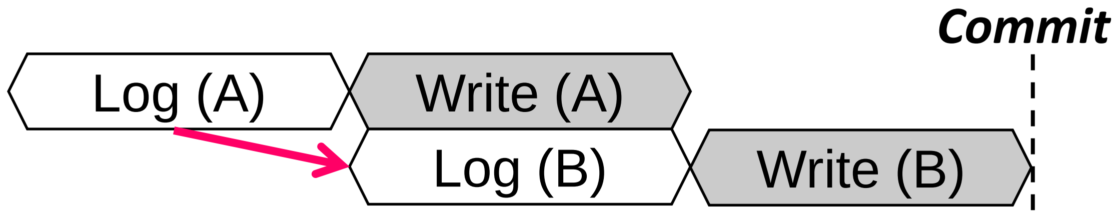
[1] A transaction without pre-defined write set

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

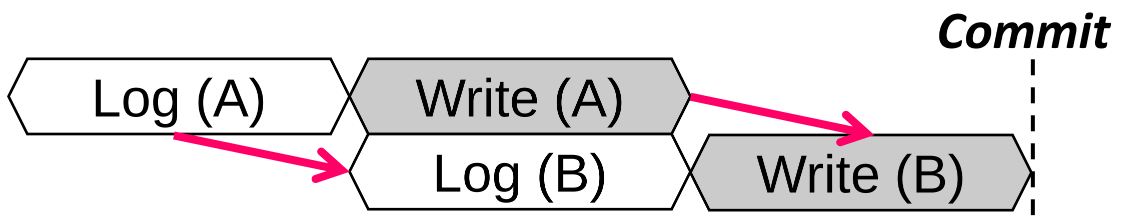
[1] A transaction without pre-defined write set

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

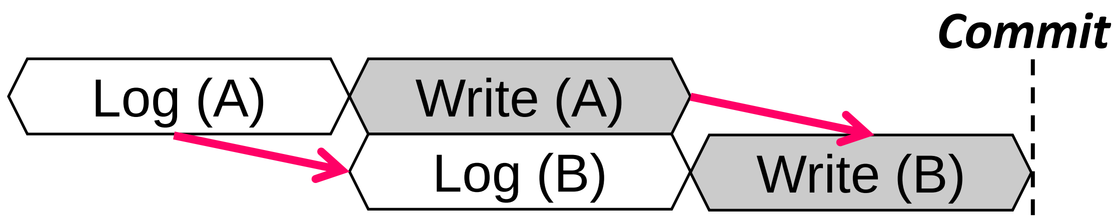
[1] A transaction without pre-defined write set

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

DESIGN AUTOMATION CONFERENCE 59

Transaction-Specific Epoch Persistency Model

Epoch Persistency Model [1]

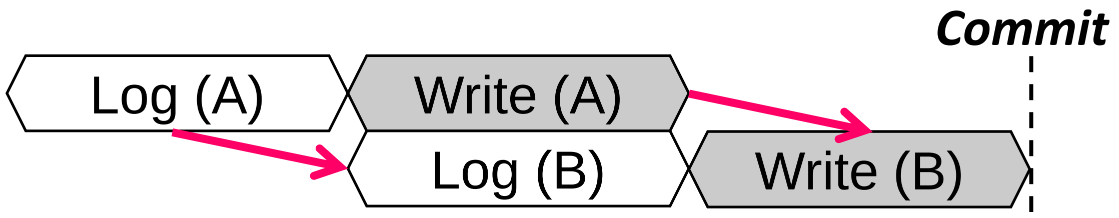
```
TX_BEGIN {
    Log (A)          } epoch 1
    clwb (LogA)
    sfence
    Write (A)        } epoch 2
    clwb (A)

    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

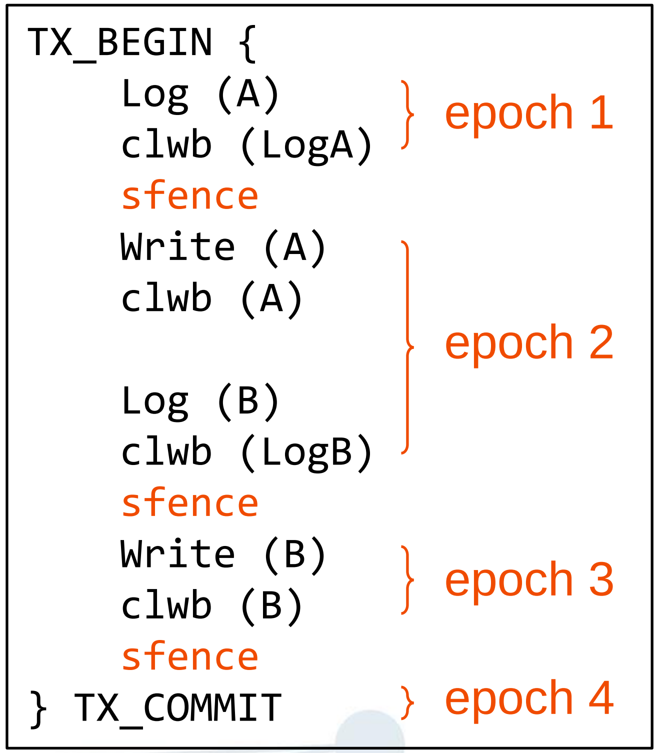


[1] Memory persistency@ISCA'14

[2] A transaction with pre-defined write set

Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model



```
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

Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model

```
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

- ***Paired epoch:*** Two adjacent epochs are paired

Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model

```
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

- ***Paired epoch:*** Two adjacent epochs are paired
    - Writes in one pair are persisted in epoch order

Transaction-Specific Epoch Persistency Model

Our Transaction-specific Epoch Persistency Model

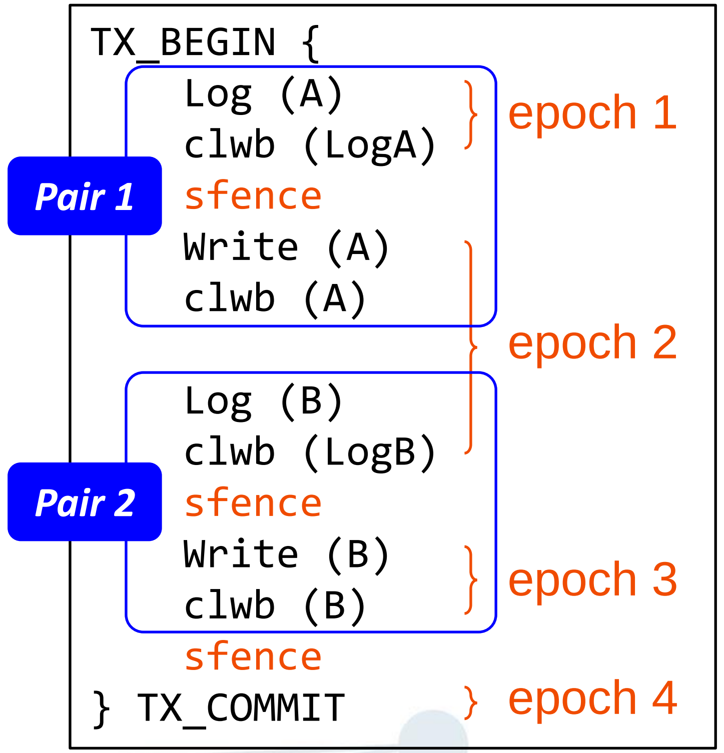
```
TX_BEGIN {
    Log (A)           } epoch 1
    clwb (LogA)
    sfence
    Write (A)         
    clwb (A)
                      } epoch 2
    Log (B)
    clwb (LogB)
    sfence
    Write (B)         } epoch 3
    clwb (B)
    sfence
} TX_COMMIT           } epoch 4
```

A dynamic transaction

- ***Paired epoch:*** 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



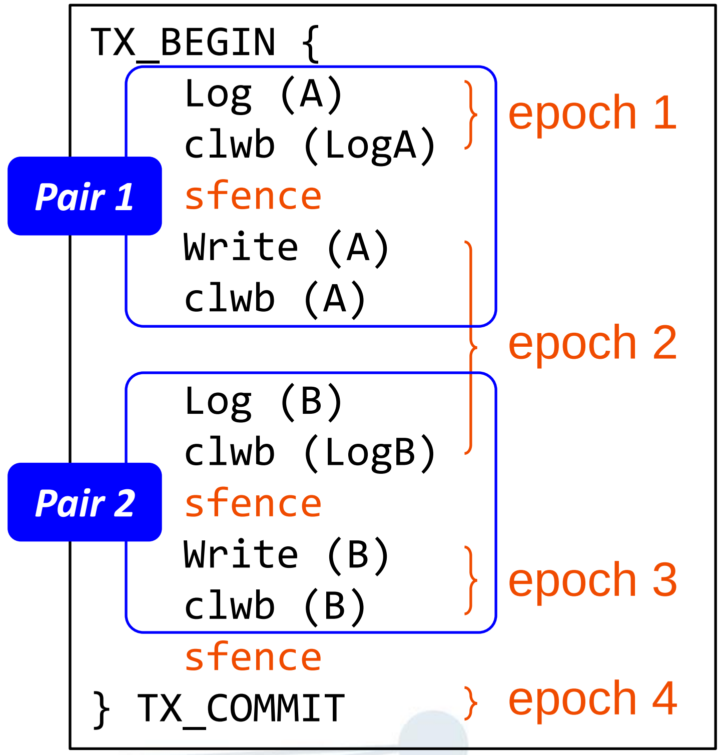
```
TX_BEGIN {
    Log (A)
    clwb (LogA)
    sfence
    Write (A)
    clwb (A)

    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

Transaction-Specific Epoch Persistency Model

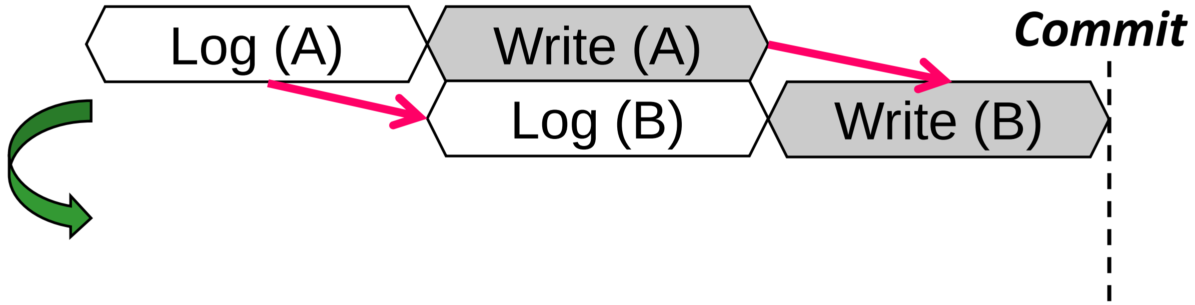
Our Transaction-specific Epoch Persistency Model



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

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

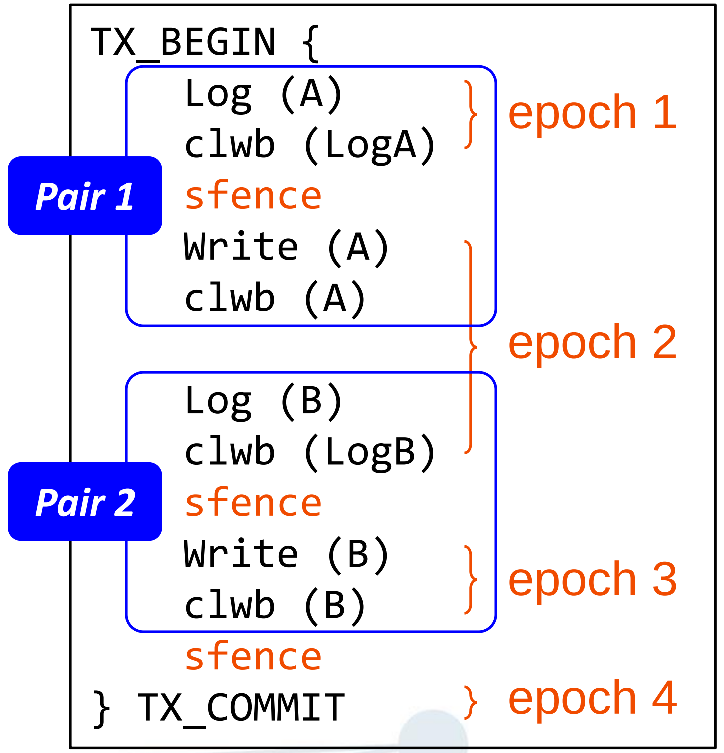
A dynamic transaction

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



Transaction-Specific Epoch Persistency Model

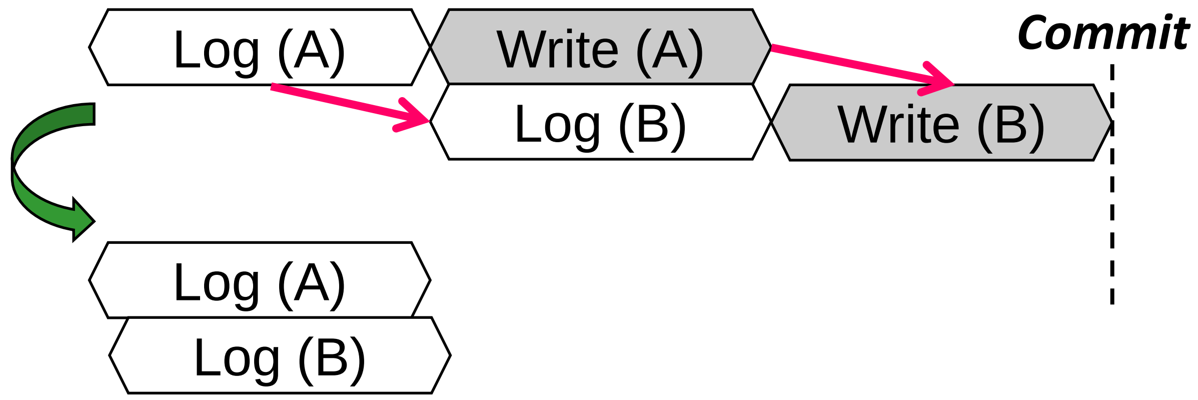
Our Transaction-specific Epoch Persistency Model



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

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

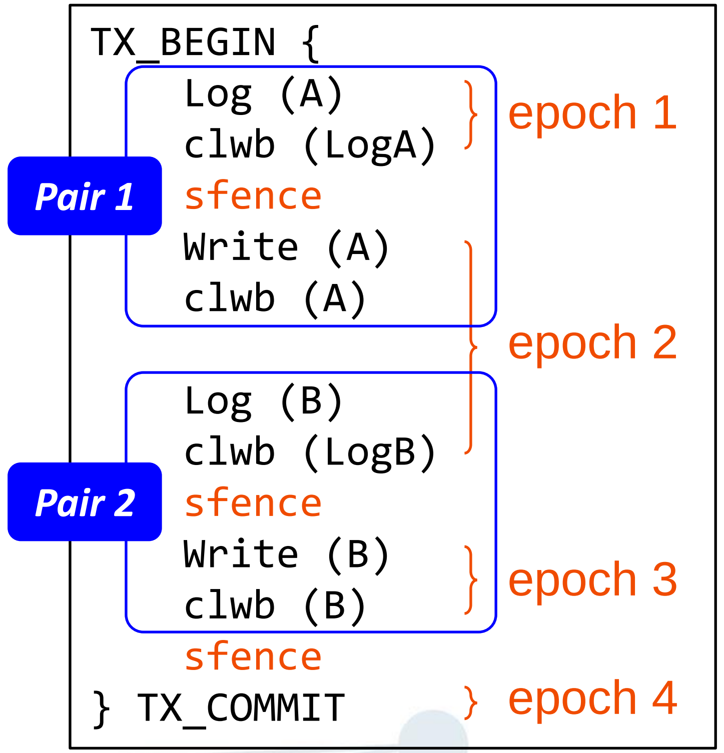
A dynamic transaction

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



Transaction-Specific Epoch Persistency Model

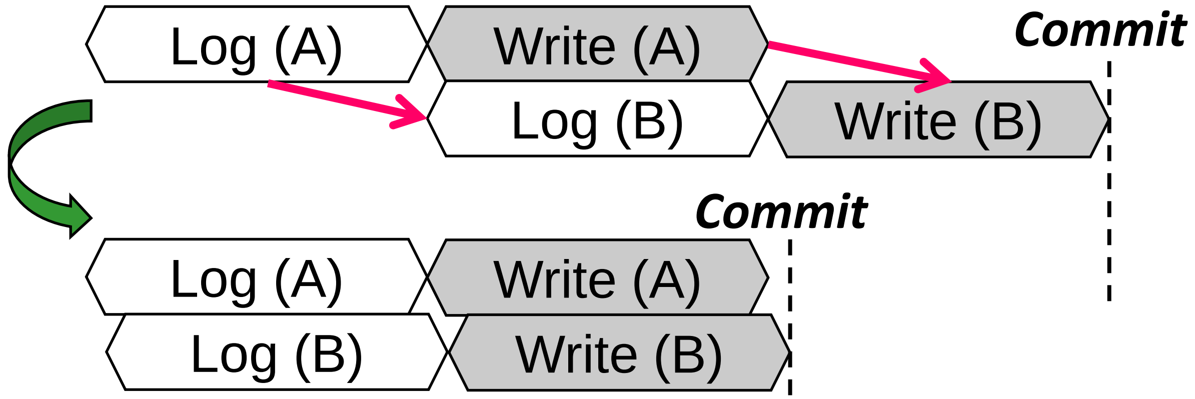
Our Transaction-specific Epoch Persistency Model



```
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
```

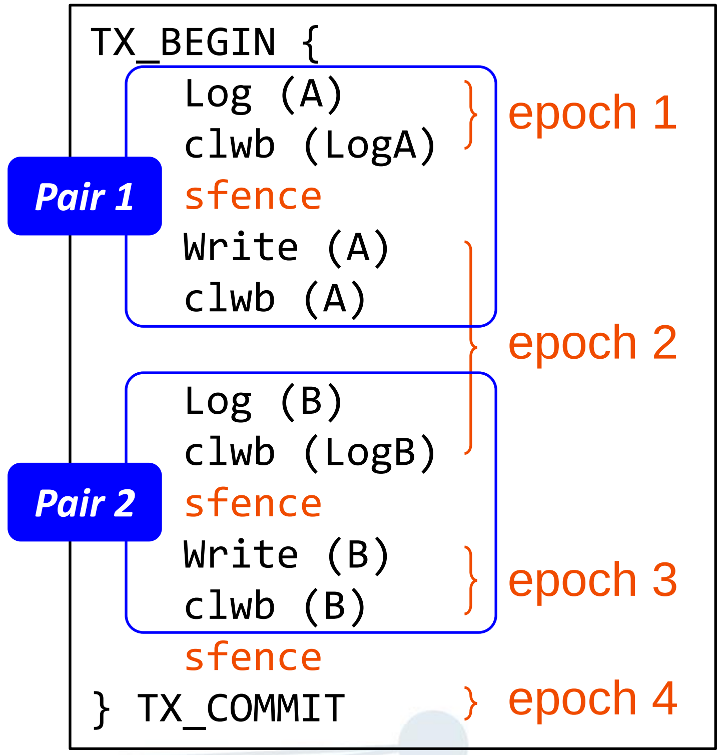
A dynamic transaction

- ***Paired epoch:*** 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

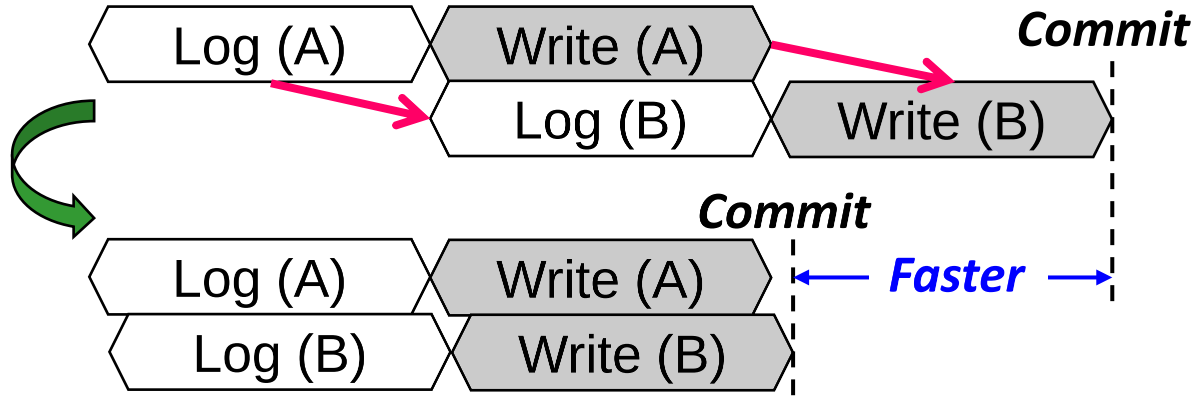
Our Transaction-specific Epoch Persistency Model



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

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

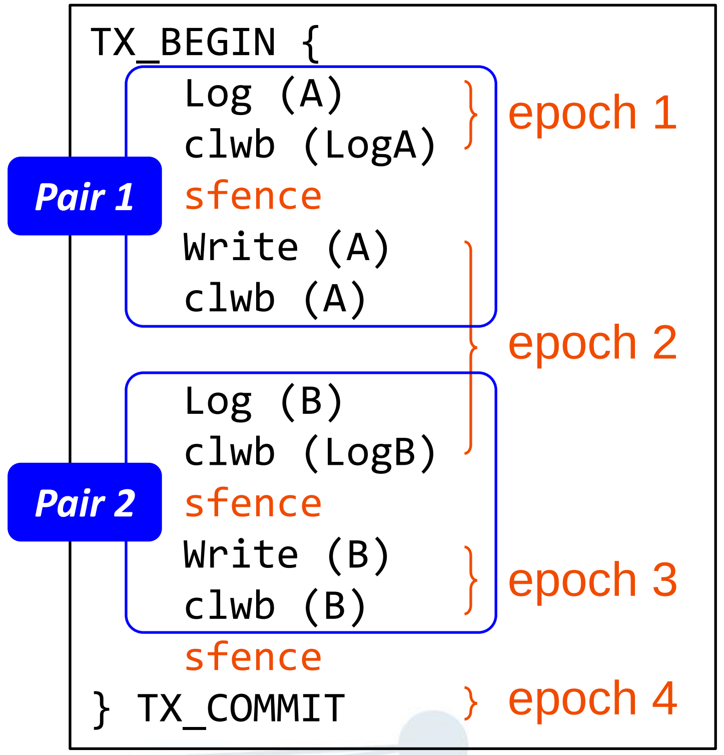
A dynamic transaction

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



Transaction-Specific Epoch Persistency Model

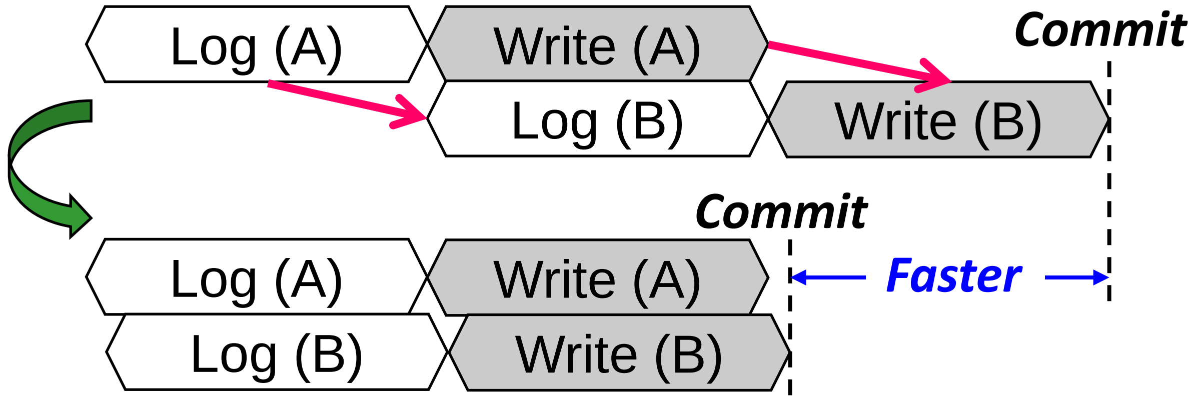
Our Transaction-specific Epoch Persistency Model



```
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

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

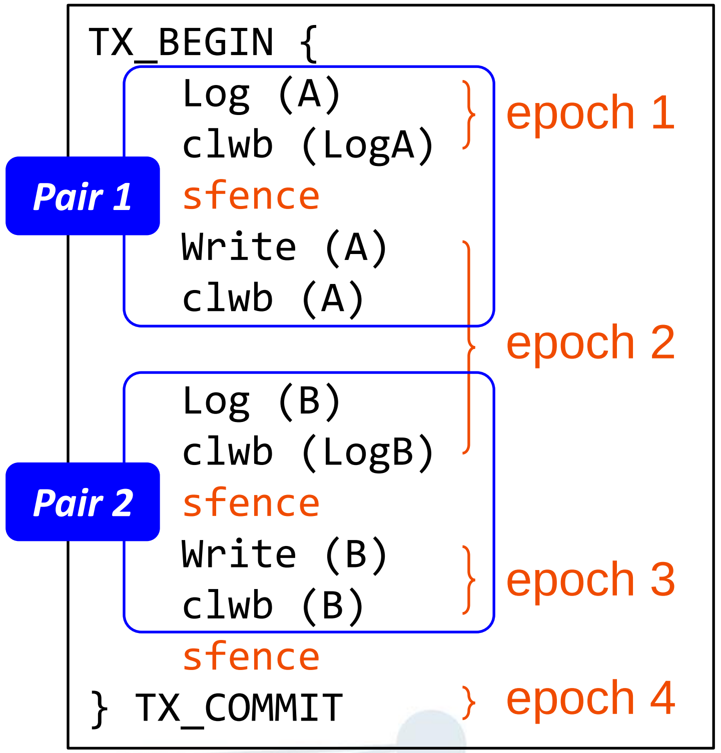


➔ Efficient in both static and dynamic transactions

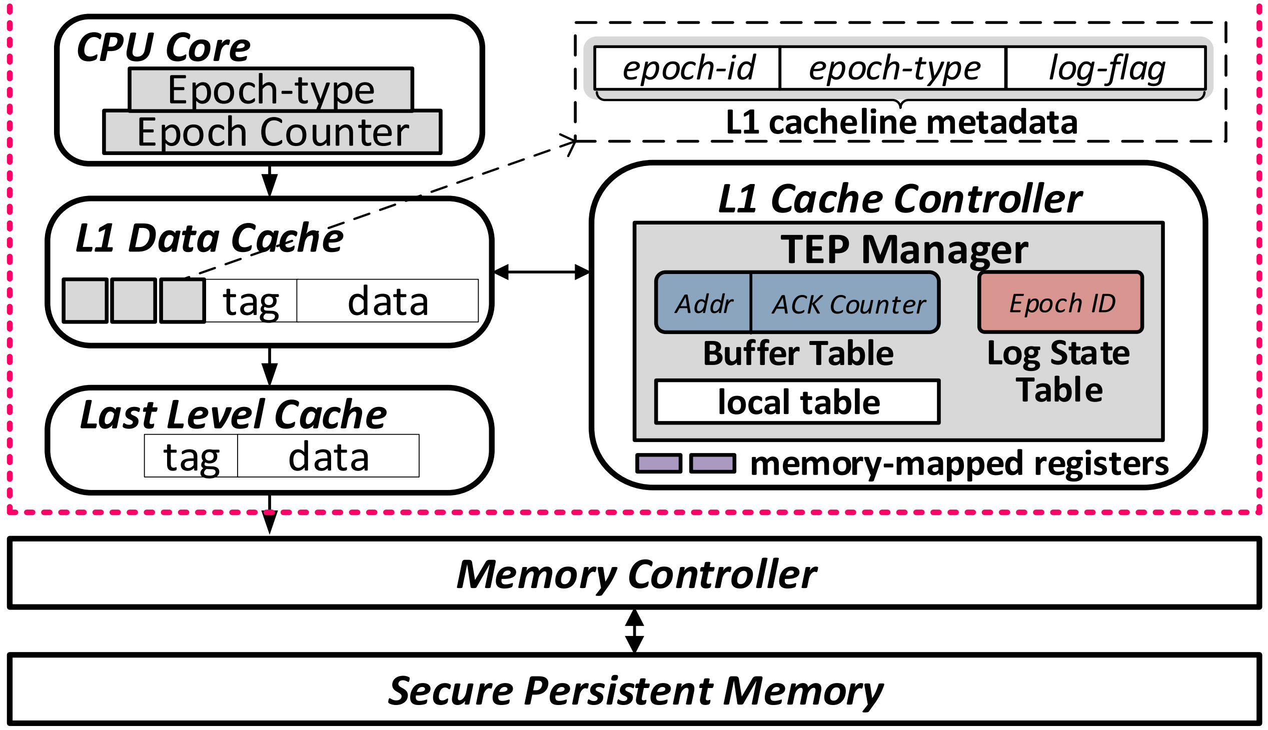
➔ Minimize ordering constraints

Transaction-Specific Epoch Persistency Model

Implementations


```
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

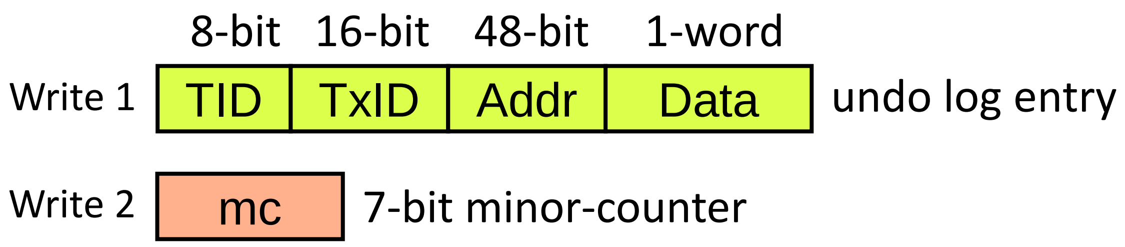
Security Metadata Write-Reduction Schemes

Co-locate log and counter

Security Metadata Write-Reduction Schemes

Co-locate log and counter

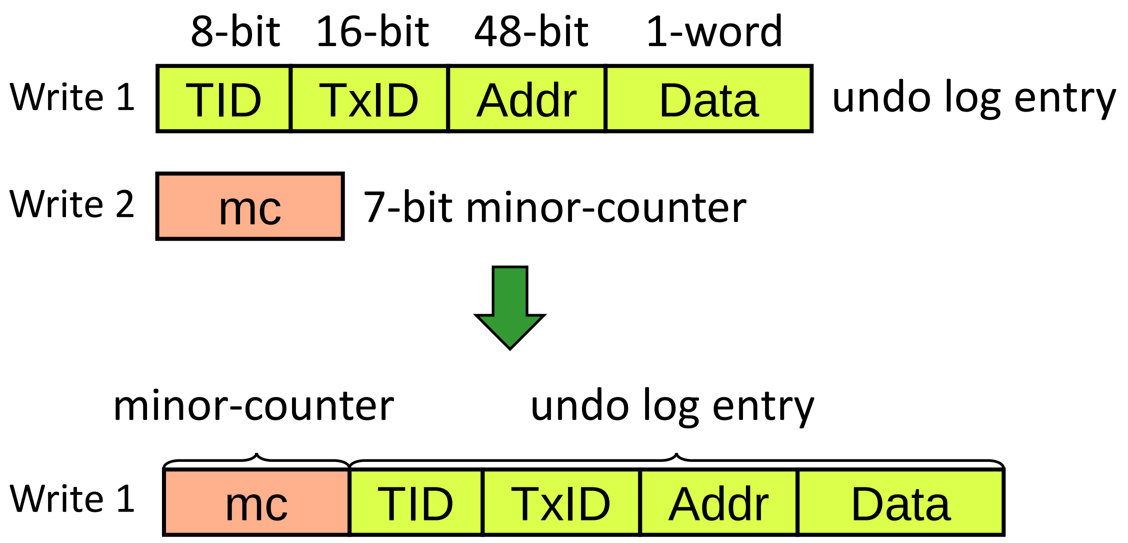
When writing data to PM



Security Metadata Write-Reduction Schemes

Co-locate log and counter

When writing data to PM

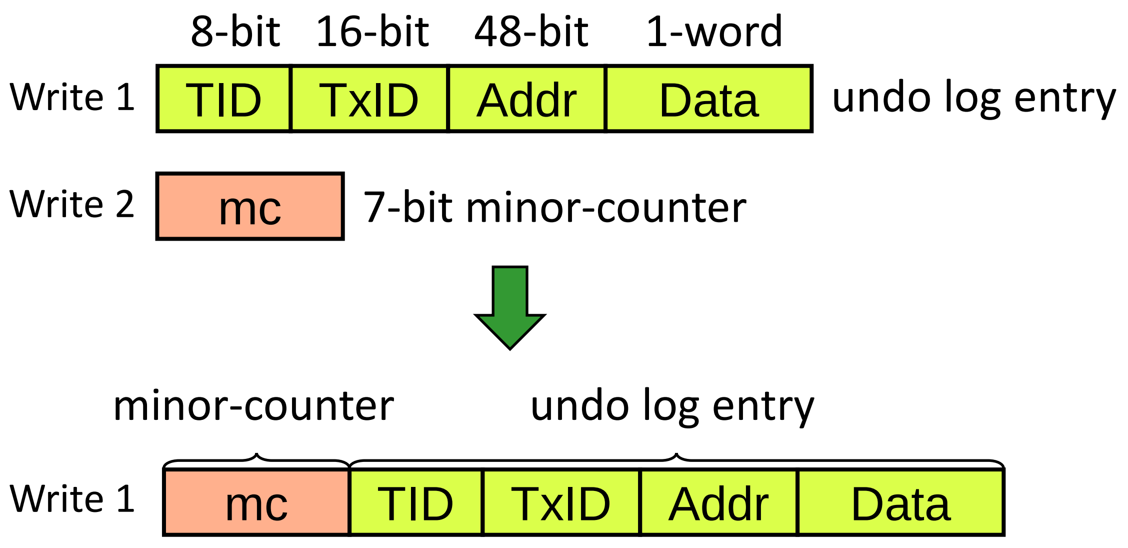


Write a minor-counter together with a log entry

Security Metadata Write-Reduction Schemes

Co-locate log and counter

When writing data to PM



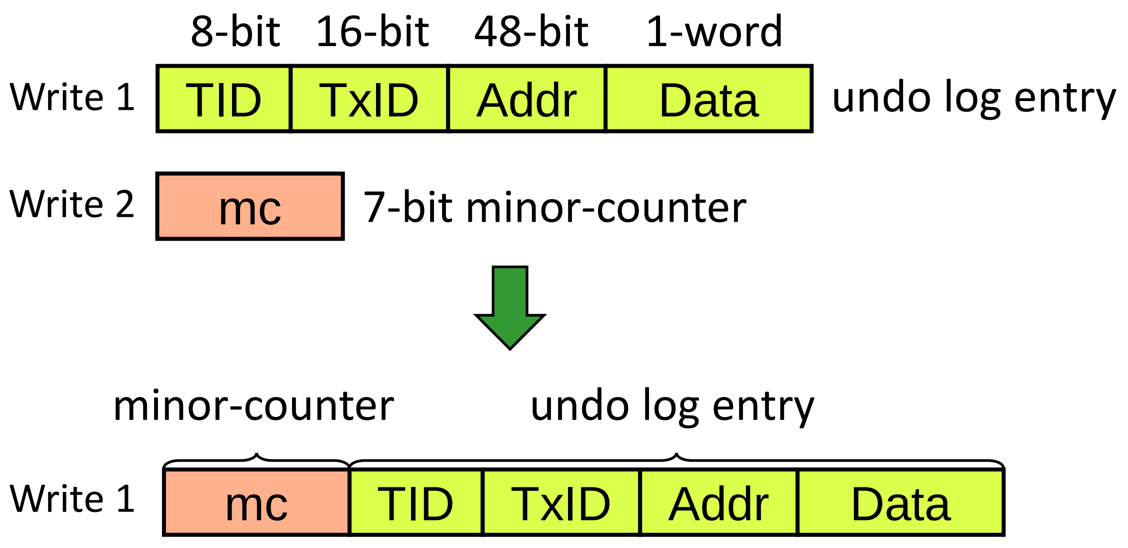
Write a minor-counter together with a log entry

Coalesce BMT blocks

Security Metadata Write-Reduction Schemes

Co-locate log and counter

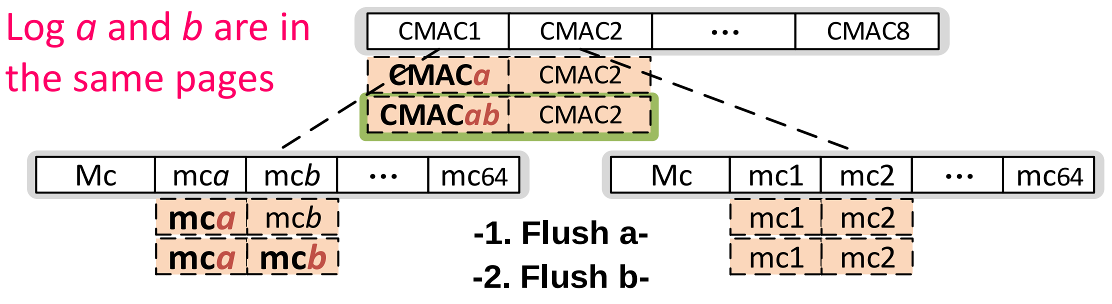
When writing data to PM



Write a minor-counter together with a log entry

Coalesce BMT blocks

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

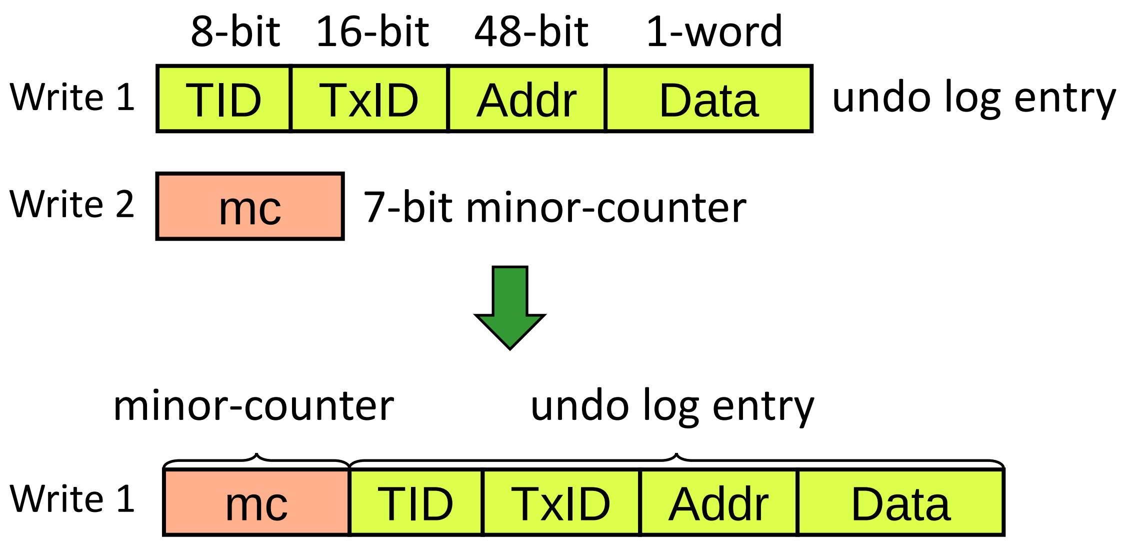


DESIGN AUTOMATION CONFERENCE

Security Metadata Write-Reduction Schemes

Co-locate log and counter

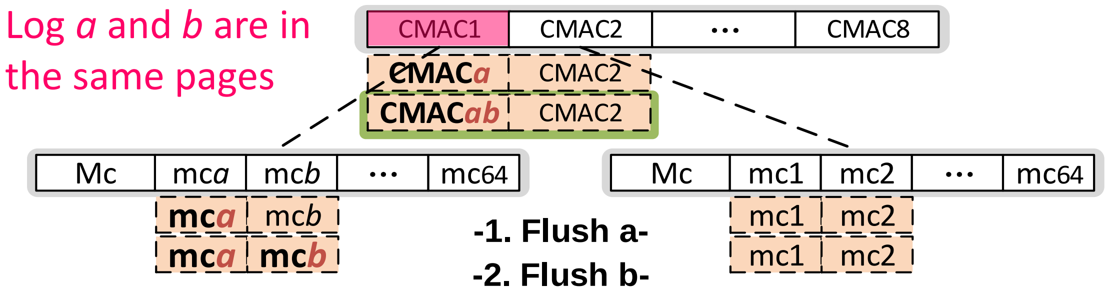
When writing data to PM



Write a minor-counter together with a log entry

Coalesce BMT blocks

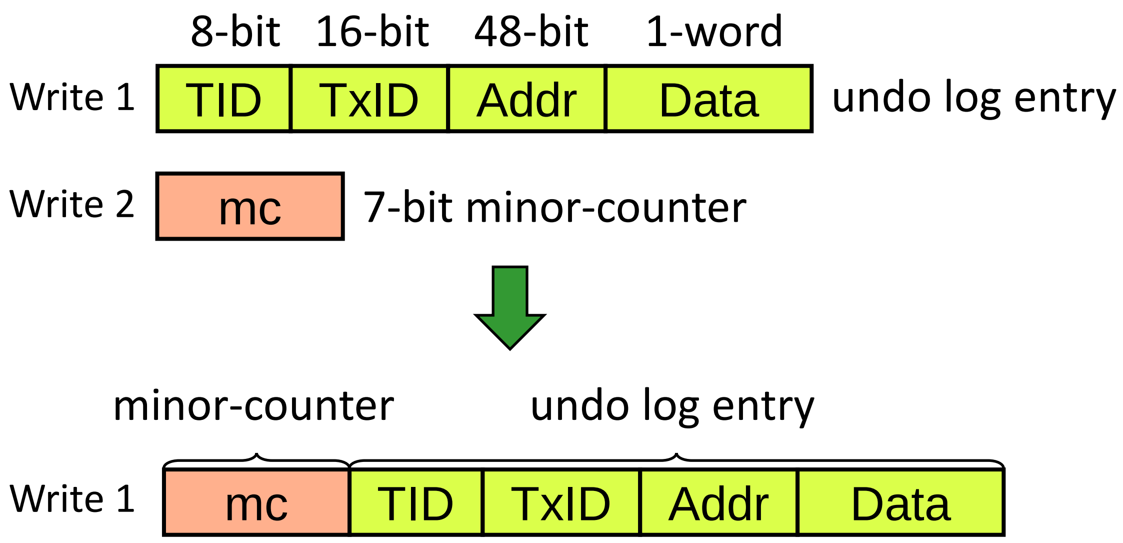
Log *a* and *b* are in the same pages



Security Metadata Write-Reduction Schemes

Co-locate log and counter

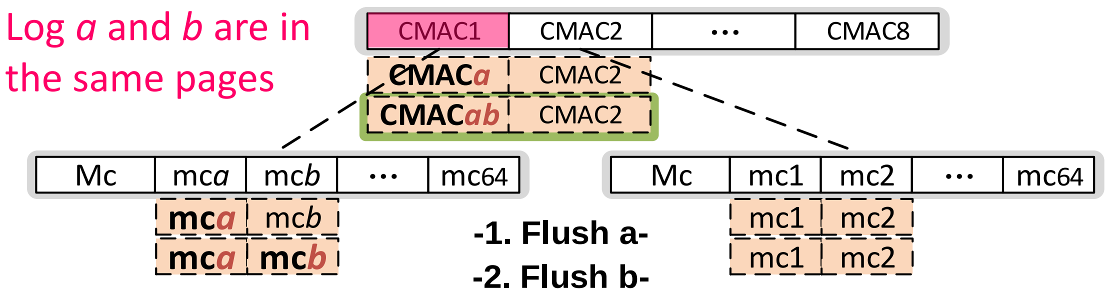
When writing data to PM



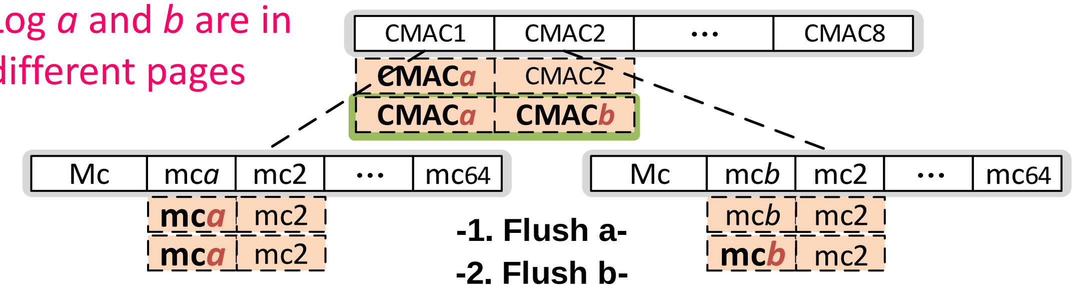
Write a minor-counter together with a log entry

Coalesce BMT blocks

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



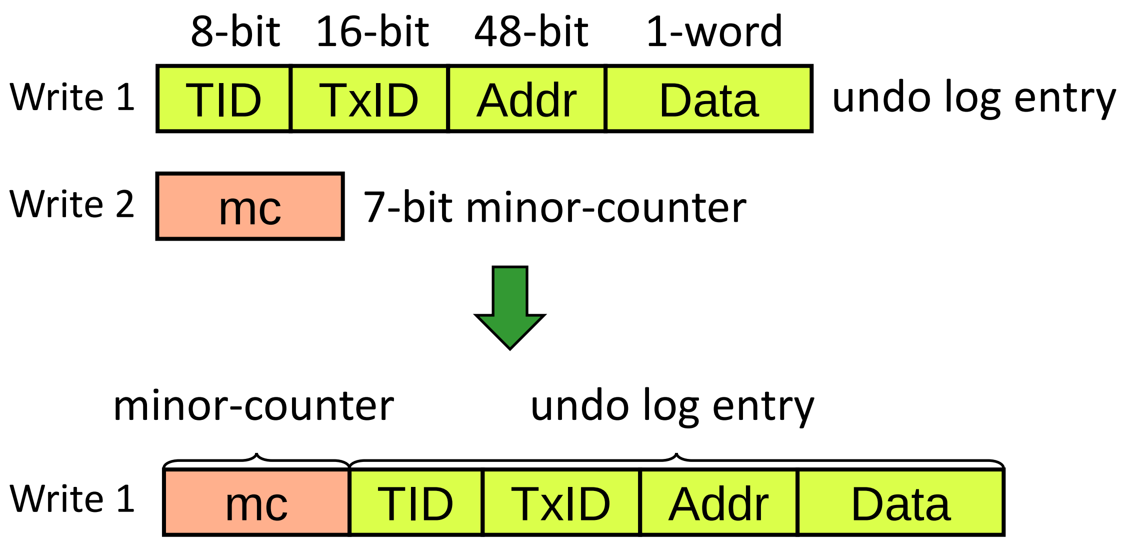
Log *a* and *b* are in different pages



Security Metadata Write-Reduction Schemes

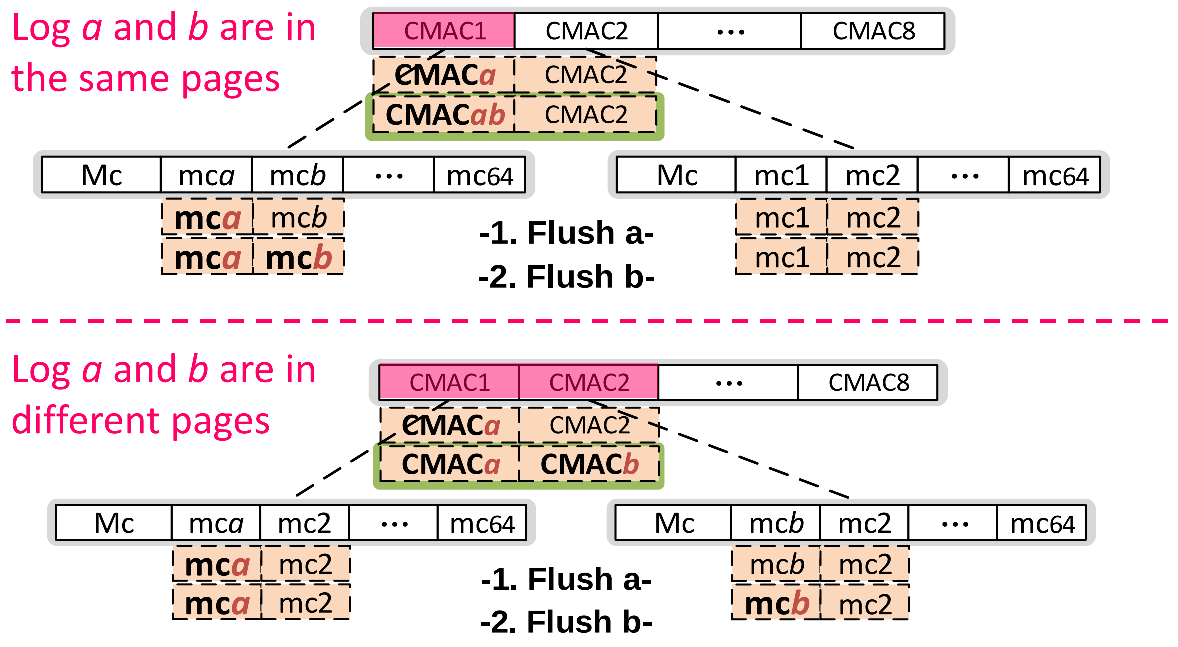
Co-locate log and counter

When writing data to PM



Write a minor-counter together with a log entry

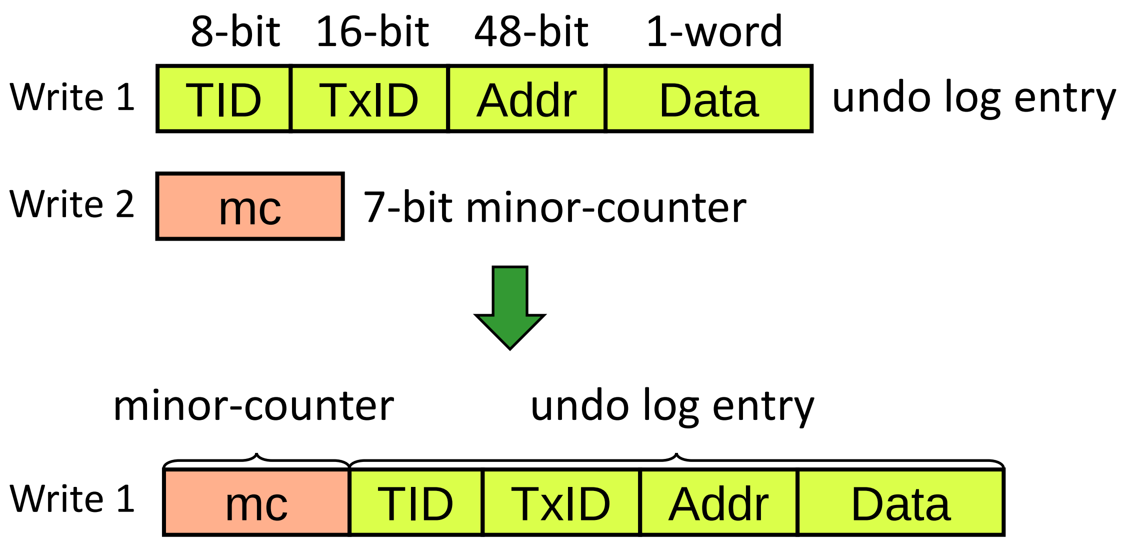
Coalesce BMT blocks



Security Metadata Write-Reduction Schemes

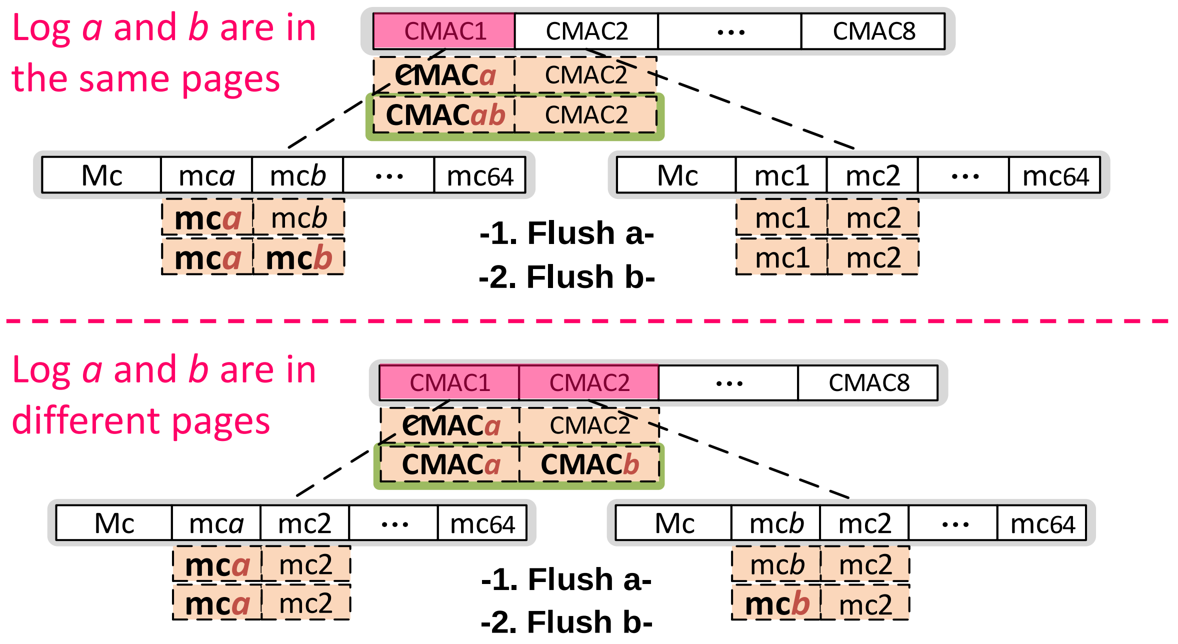
Co-locate log and counter

When writing data to PM



Write a minor-counter together with a log entry

Coalesce BMT blocks



Exploit the spatial locality to merge BMT writes

Performance Evaluation

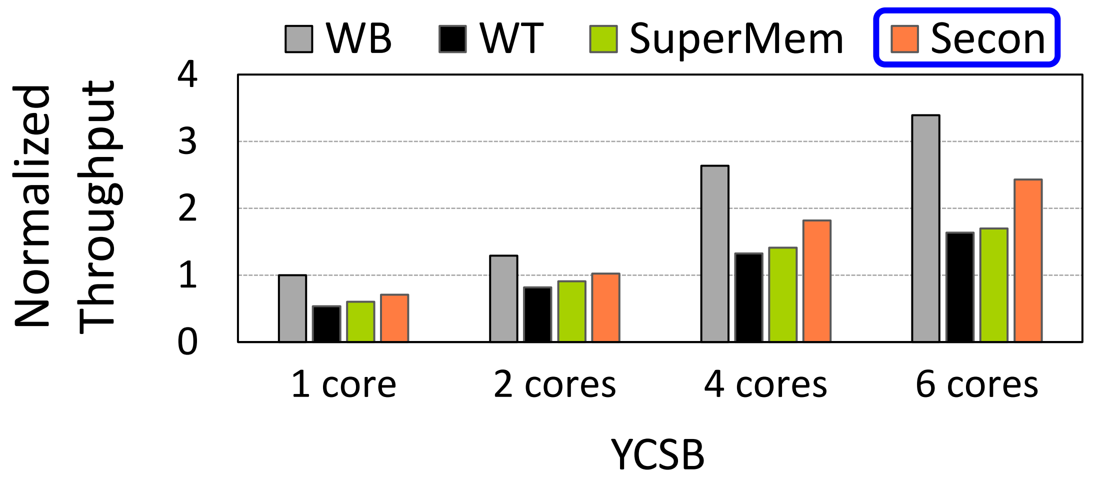
- Model Secon using Gem5 and NVMain

Design	Description
WB	An ideal write-back scheme
WT	A standard write-through scheme
SuperMem [MICRO'19]	A write-optimized write-through scheme using our BMT coalescing
Secon	**Our proposed schemes**

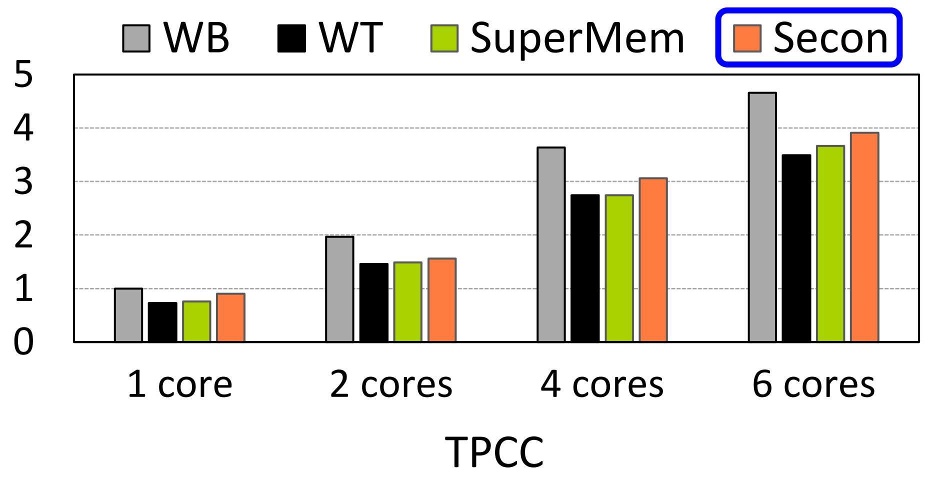
Benchmark	Description
Array	Swap two random entries in an array
Queue	Enqueue/dequeue random entries in a queue
Btree	Insert/delete random nodes in a B-tree
Hash	Insert/delete random items in a hash table
RBtree	Insert/delete random nodes in a red-black tree
YCSB	Cloud benchmark. 100% update
TPCC	OLTP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

Transaction Throughput

43% improvement over SuperMem

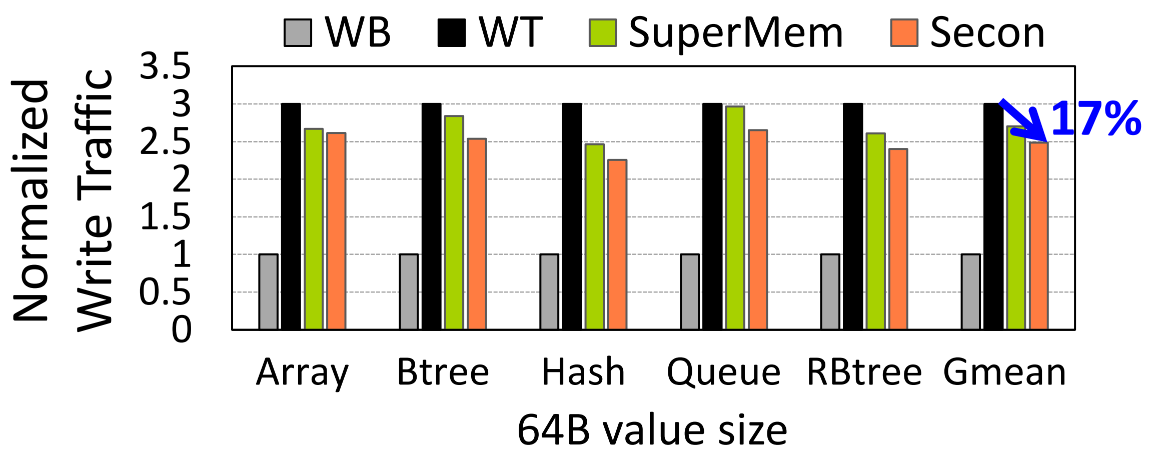


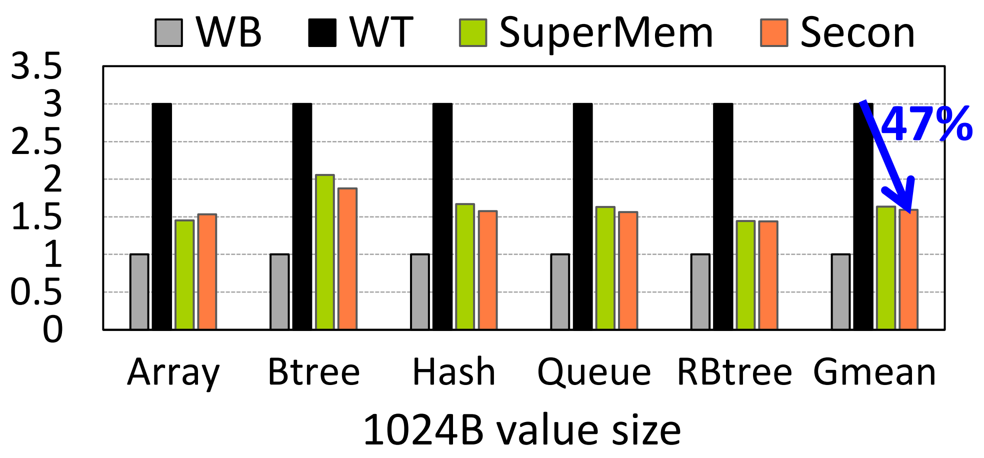
19% improvement over SuperMem



- Move BMT update to the background
- Eliminate unnecessary ordering constraints

Write Traffic





- 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

DESIGN AUTOMATION CONFERENCE

Thanks! Q&A