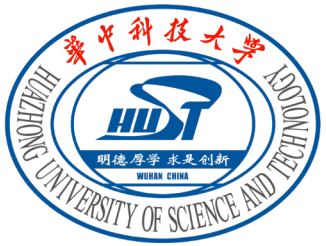
Efficiently Detecting Concurrency Bugs in Persistent Memory Programs

Zhangyu Chen, Yu Hua, Yongle Zhang*, Luochangqi Ding

Huazhong University of Science and Technology

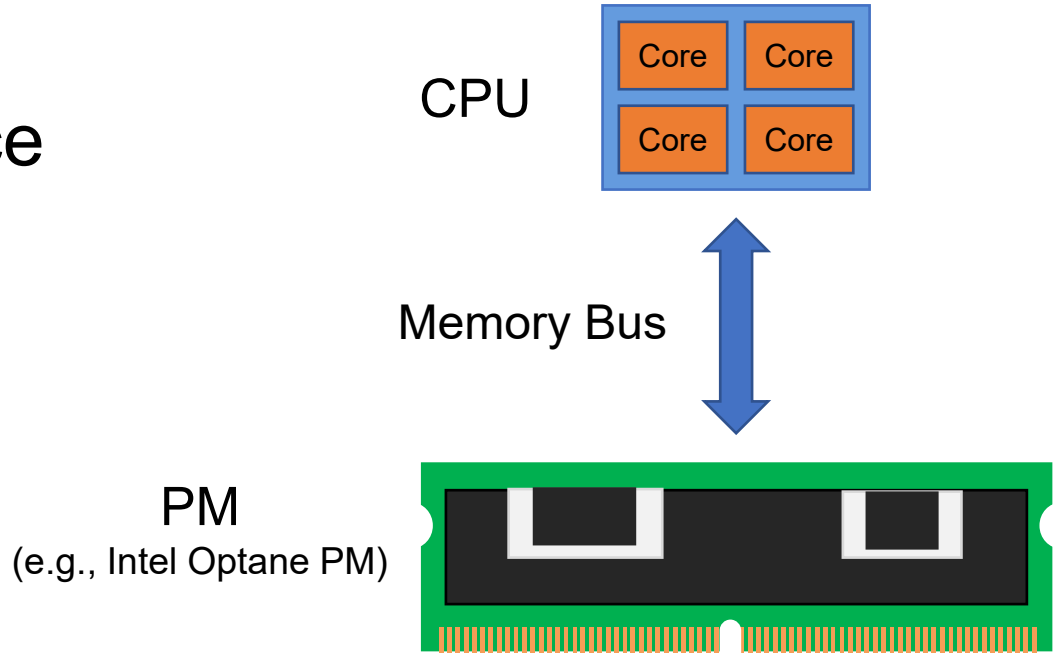
**Purdue University*

ASPLOS 2022



Persistent Memory (PM)

- PM characteristics
  - DRAM-comparable performance
  - TB-scale capacity
  - Non-volatility
  - Byte-addressability

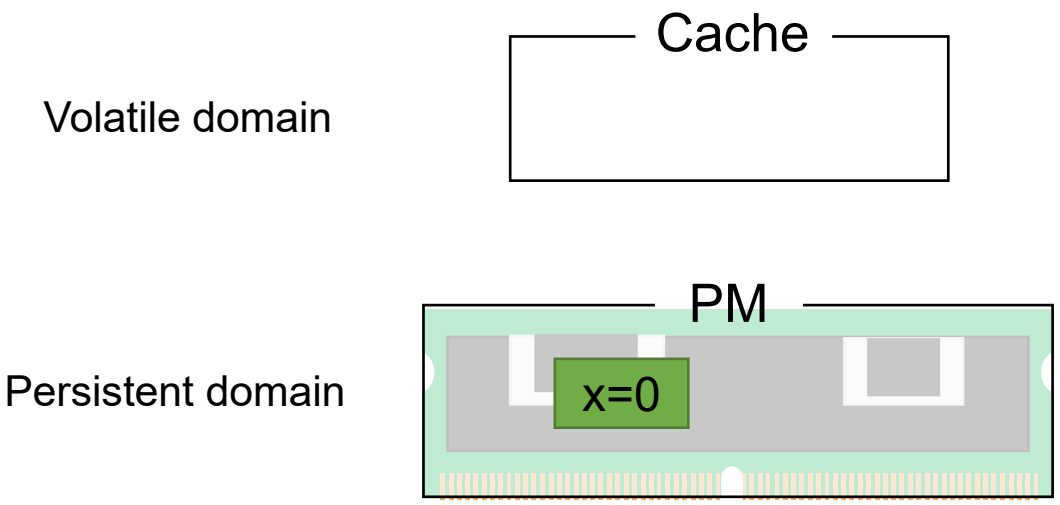


- New opportunities for memory systems
  - Lower cost/GB than DRAM
  - Instant recovery from PM

PM Programming

- PM programming is non-trivial
  - Volatile CPU caches

```
x = A;
```

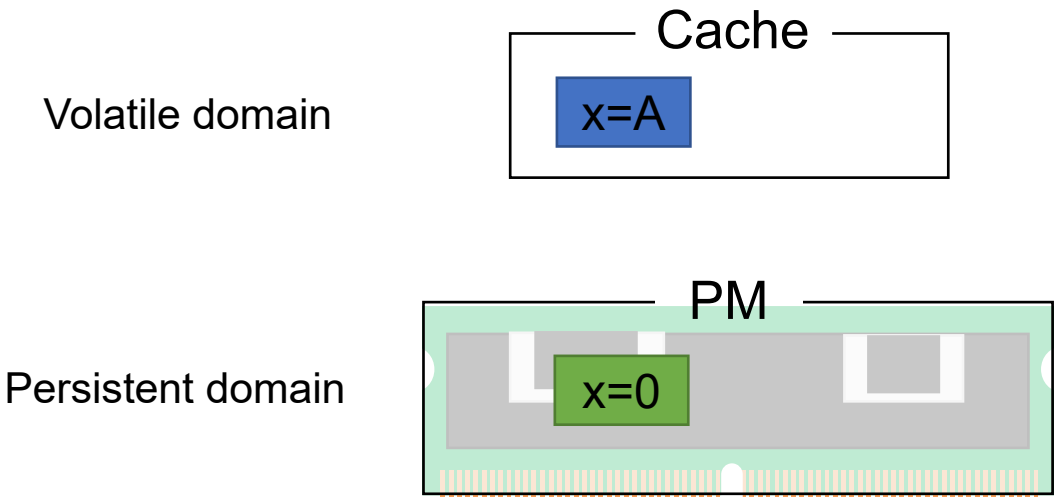


** Assume x = 0 initially*

PM Programming

- PM programming is non-trivial
  - Volatile CPU caches

```
x = A;
```

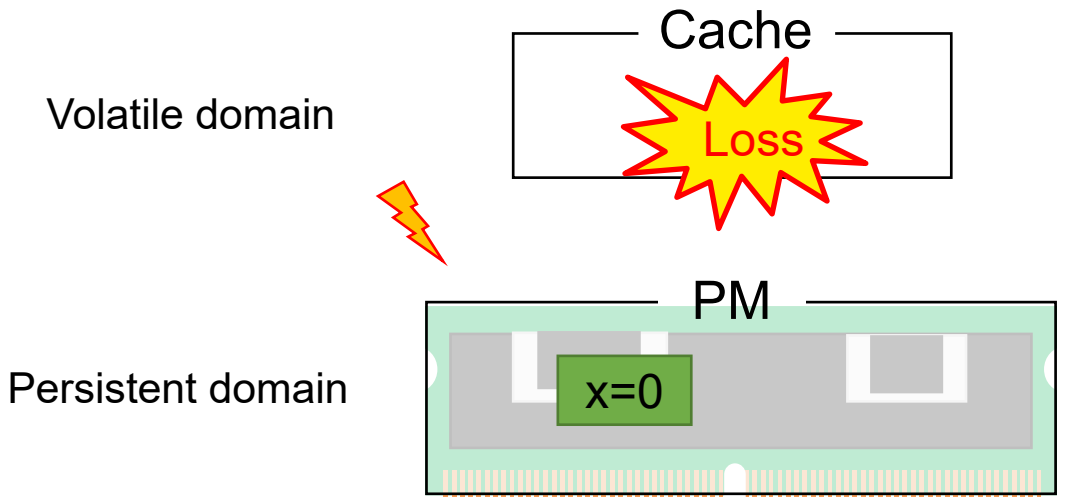


** Assume x = 0 initially*

PM Programming

- PM programming is non-trivial
  - Volatile CPU caches

```
x = A;
```



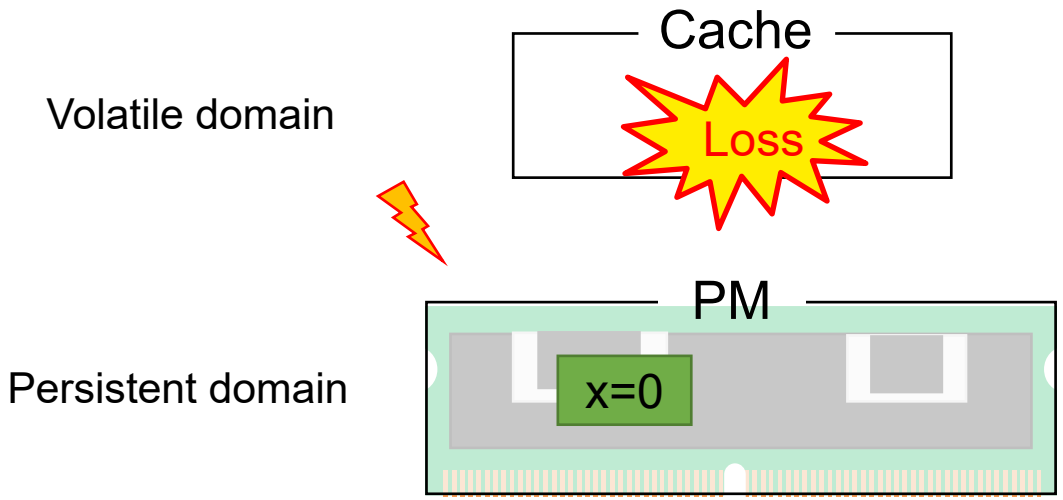
** Assume x = 0 initially*

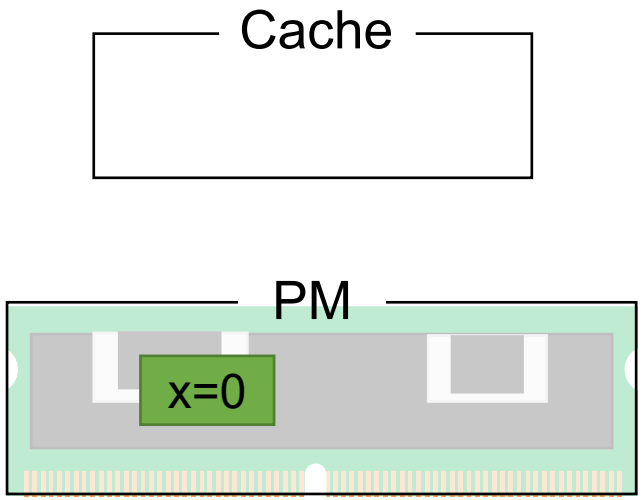
PM Programming

- PM programming is non-trivial
  - Volatile CPU caches
- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)

```
x = A;
```

```
x = A;
clwb &x;
```





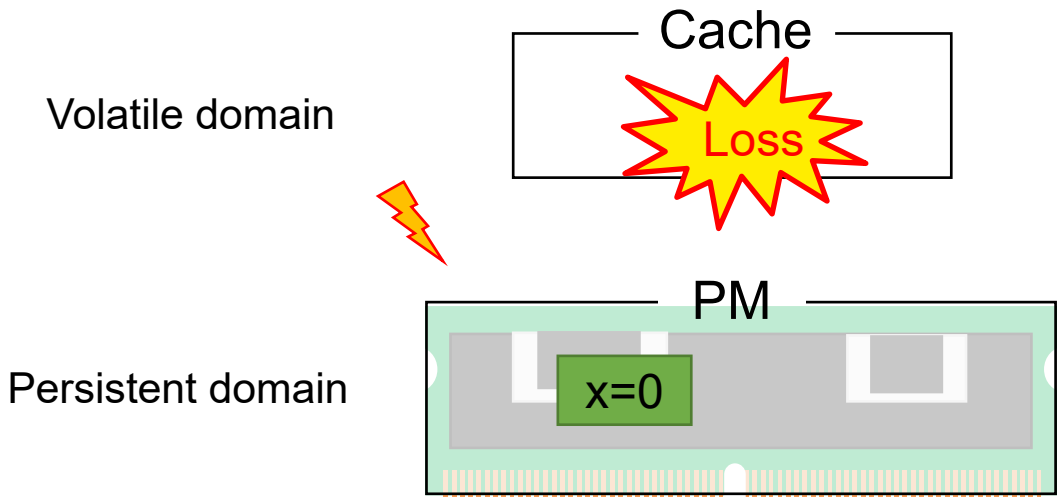
** Assume x = 0 initially*

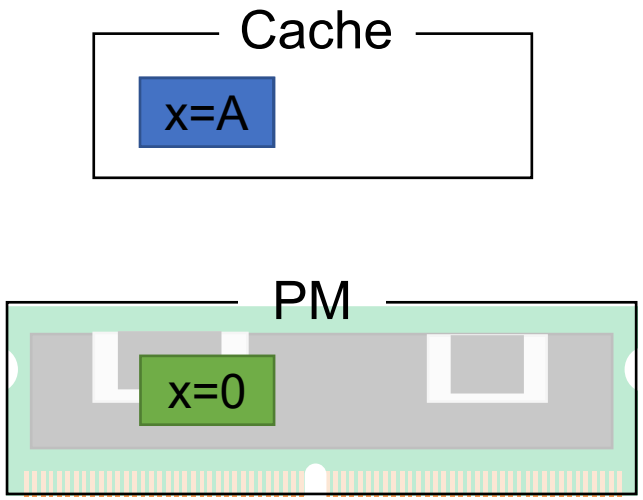
PM Programming

- PM programming is non-trivial
  - Volatile CPU caches
- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)

```
x = A;
```

```
x = A;
clwb &x;
```



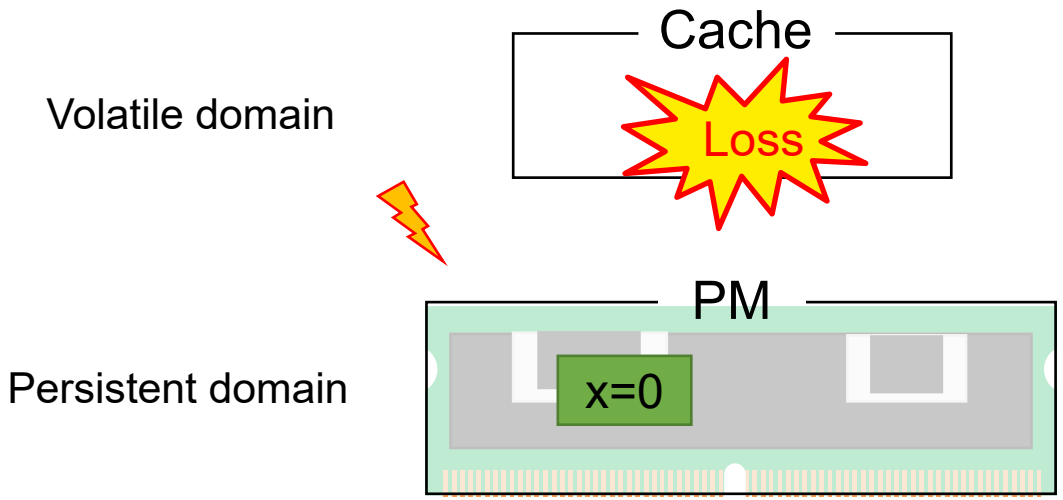


** Assume x = 0 initially*

PM Programming

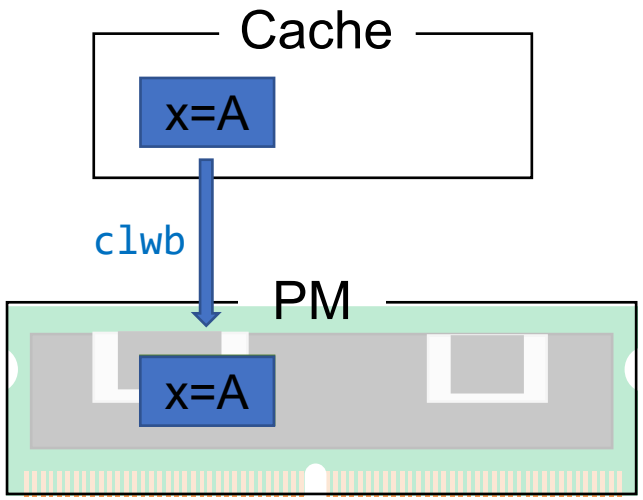
- PM programming is non-trivial
  - Volatile CPU caches
- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)

```
x = A;
```



** Assume x = 0 initially*

```
x = A;
clwb &x;
```



PM Programming

- PM programming is non-trivial
  - Volatile CPU caches
  - Persistency reordering

```
x = A;
clwb &x;
y = x;
clwb &y;
```

- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)

PM Programming

- PM programming is non-trivial
  - Volatile CPU caches
  - Persistency reordering

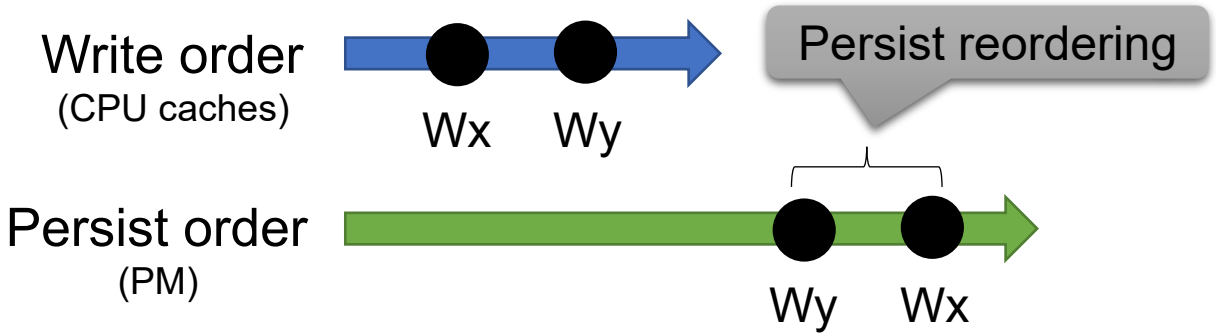
```
x = A;
clwb &x;
y = x;
clwb &y;
```

- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)

PM Programming

- PM programming is non-trivial
  - Volatile CPU caches
  - Persistency reordering

```
x = A;
clwb &x;
y = x;
clwb &y;
```

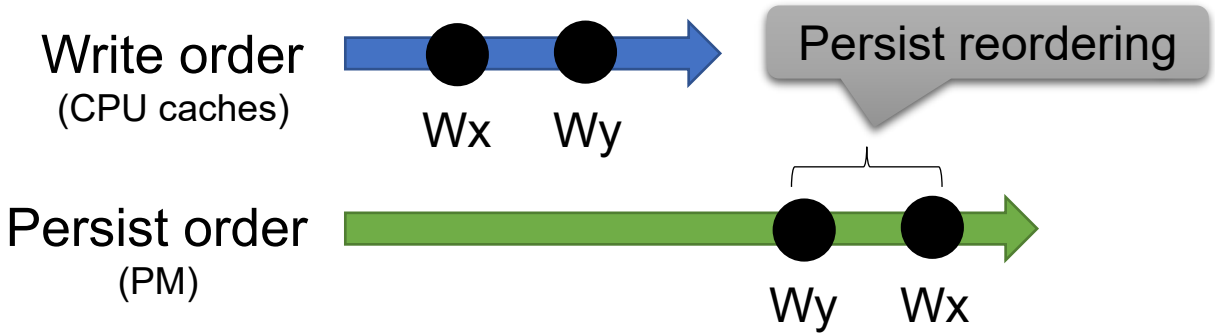


- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)

PM Programming

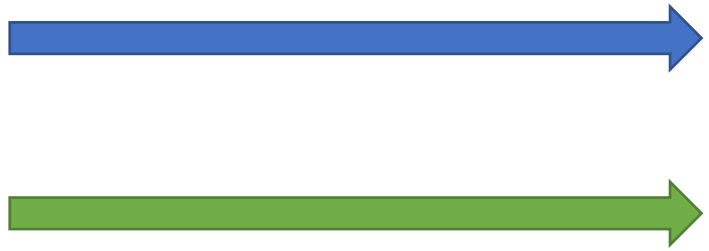
- PM programming is non-trivial
  - Volatile CPU caches
  - Persistency reordering

```
x = A;
clwb &x;
y = x;
clwb &y;
```



- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)
  - Fence for ordering (e.g., `sfence` from x86)

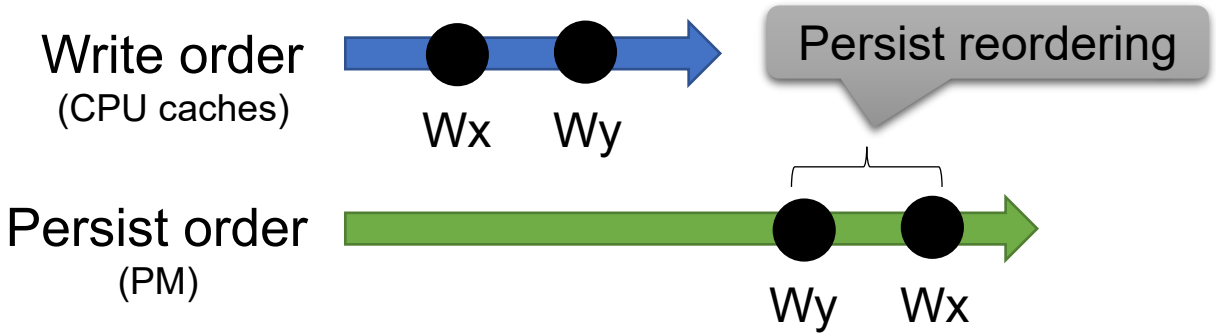
```
x = A;
clwb &x;

y = x;
clwb &y;
```



PM Programming

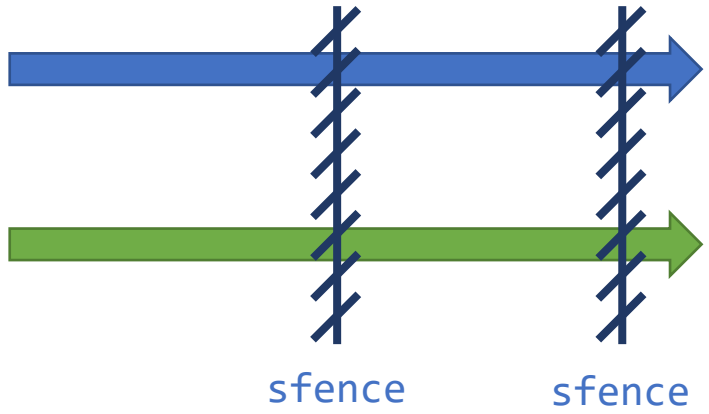
- PM programming is non-trivial
  - Volatile CPU caches
  - Persistency reordering

```
x = A;
clwb &x;
y = x;
clwb &y;
```



- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)
  - Fence for ordering (e.g., `sfence` from x86)

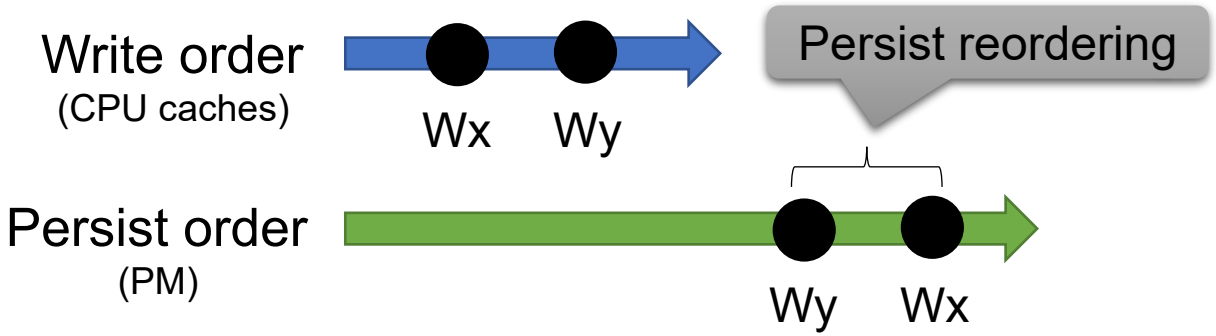
```
x = A;
clwb &x;
sfence;
y = x;
clwb &y;
sfence;
```



PM Programming

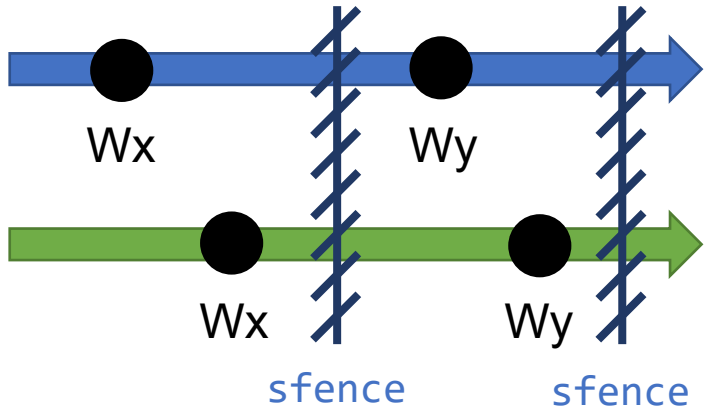
- PM programming is non-trivial
  - Volatile CPU caches
  - Persistency reordering

```
x = A;
clwb &x;
y = x;
clwb &y;
```



- Architectural support for PM
  - Flush for durability (e.g., `clwb` from x86)
  - Fence for ordering (e.g., `sfence` from x86)

```
x = A;
clwb &x;
sfence;
y = x;
clwb &y;
sfence;
```



PM Crash-Consistency Bugs

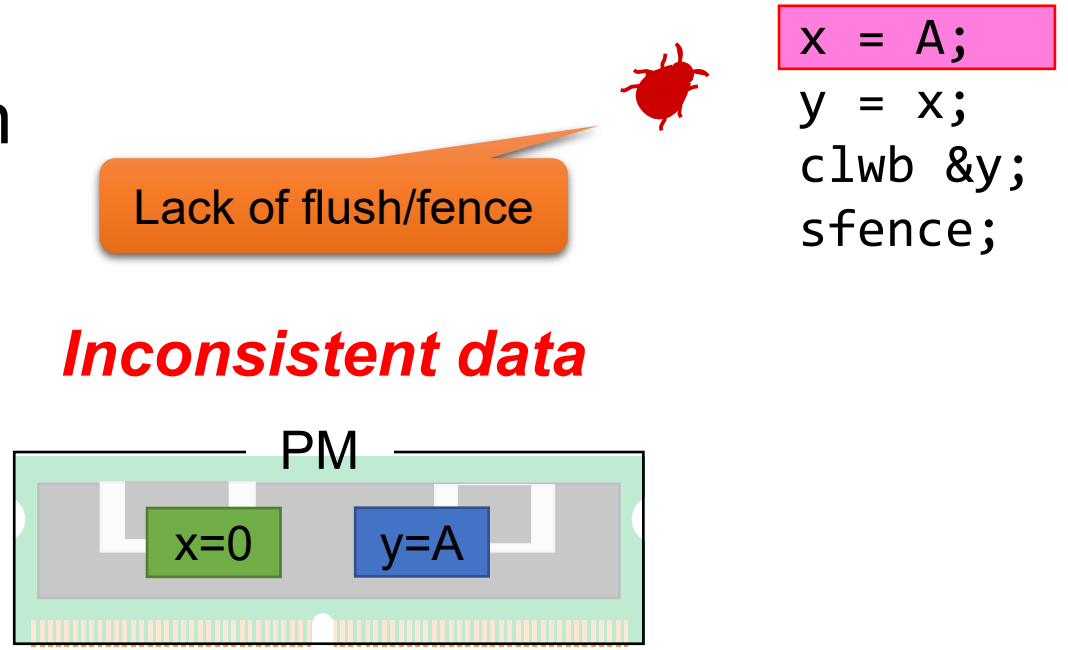
- Correctness bugs
  - Causing correctness violation
  - Patterns:
    - Missing flush/fence
    - …

```
x = A;
y = x;
clwb &y;
sfence;
```

PM Crash-Consistency Bugs

- Correctness bugs
  - Causing correctness violation
  - Patterns:
    - Missing flush/fence
    - …

** Assume x = 0, y = 0 initially*



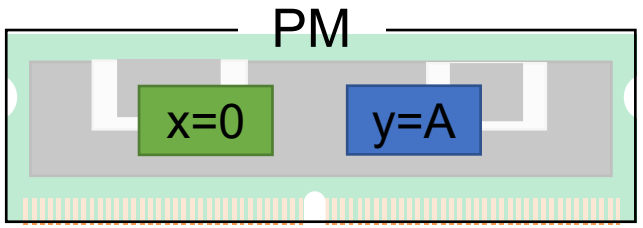
PM Crash-Consistency Bugs

- Correctness bugs
  - Causing correctness violation
  - Patterns:
    - Missing flush/fence
    - …

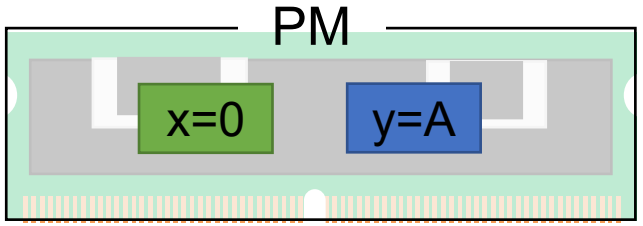
```
x = A;
y = x;
clwb &y;
sfence;
```

Lack of flush/fence

Inconsistent data



Restart



** Assume x = 0, y = 0 initially*

Bugs still exist after restarts!

PM Crash-Consistency Bugs

- Correctness bugs
  - Causing correctness violation
  - Patterns:
    - Missing flush/fence
    - …

Lack of flush/fence

Inconsistent data

```
x = A;
y = x;
clwb &y;
sfence;
```

- Performance bugs
  - Causing performance degradation
  - Patterns:
    - Extra flush/fence
    - …

```
x = A;
clwb &x;
sfence;
clwb &x;
y = x;
clwb &y;
sfence;
```

PM Crash-Consistency Bugs

- Correctness bugs
  - Causing correctness violation
  - Patterns:
    - Missing flush/fence
    - …

Lack of flush/fence

```
x = A;
y = x;
clwb &y;
sfence;
```

Inconsistent data

- Performance bugs
  - Causing performance degradation
  - Patterns:
    - Extra flush/fence
    - …

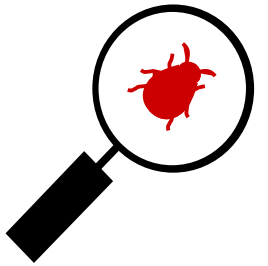
Extra flush

```
x = A;
clwb &x;
sfence;
clwb &x;
y = x;
clwb &y;
sfence;
```

Unnecessary stall

Existing Automatic PM Debugging Tools

- Correctness bugs
  - Missing flush/fence
    - AGAMOTTO [OSDI '20]
    - PMDebugger [ASPLOS '21]
    - …



```
x = A;
y = x;
clwb &y;
sfence;
```

Are PM writes followed by flush/fence?

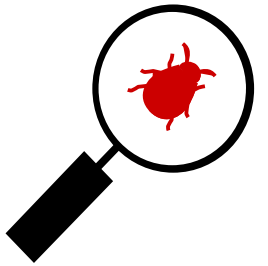
- Performance bugs
  - Extra flush/fence
    - AGAMOTTO, PMDebugger, …

```
x = A;
clwb &x;
sfence;
clwb &x;
y = x;
clwb &y;
sfence;
```

Are flushes/fences necessary?

Existing Automatic PM Debugging Tools

- Correctness bugs
  - Missing flush/fence
    - AGAMOTTO [OSDI '20]
    - PMDebugger [ASPLOS '21]
    - …
  - Other patterns
    - Cross-failure race: XFDetector [ASPLOS '20]
    - Application-specific bugs: WITCHER [SOSP '21]
    - ...
- Performance bugs
  - Extra flush/fence
    - AGAMOTTO, PMDebugger, …



```
x = A;
y = x;
clwb &y;
sfence;
```

Are PM writes followed by flush/fence?

```
x = A;
clwb &x;
sfence;
clwb &x;
y = x;
clwb &y;
sfence;
```

Are flushes/fences necessary?

Existing Automatic PM Debugging Tools

- Correctness bugs
  - Missing flush/fence
    - AGAMOTTO [OSDI '20]
    - PMDebugger [ASPLOS '21]
    - …
  - Other patterns
    - ...
- Performance bugs
  - Extra flush/fence
    - AGAMOTTO, PMDebugger, …

```
x = A;
y = x;
clwb &y;
sfence;
```

Are PM writes followed by flush/fence?

Lack of considerations for concurrent executions!

```
clwb &x;
y = x;
clwb &y;
sfence;
```

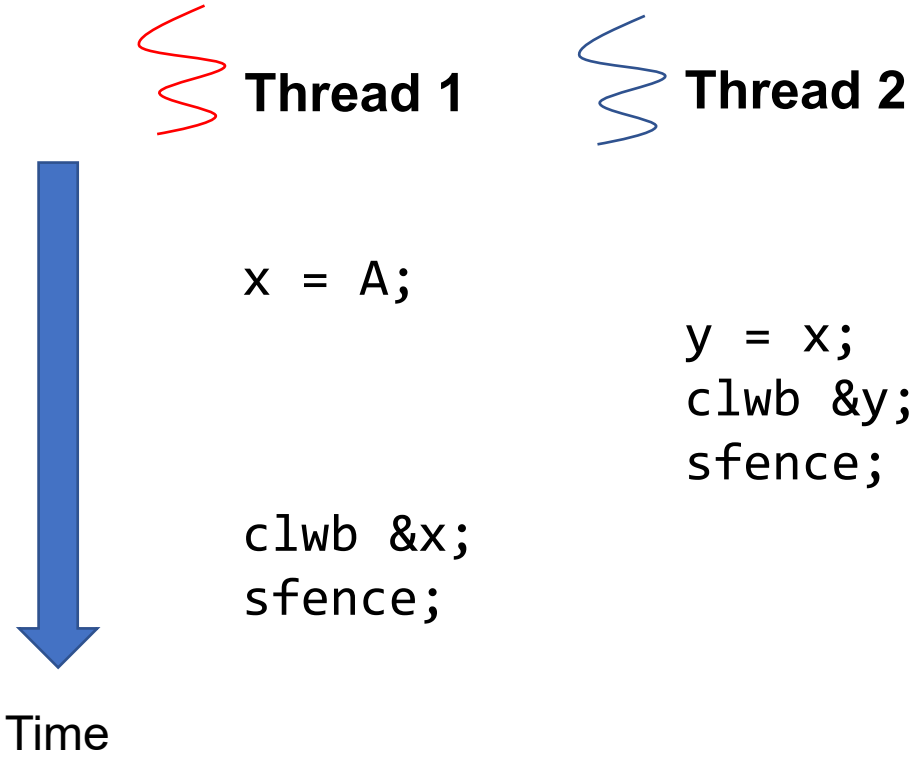
Are flushes/fences necessary?

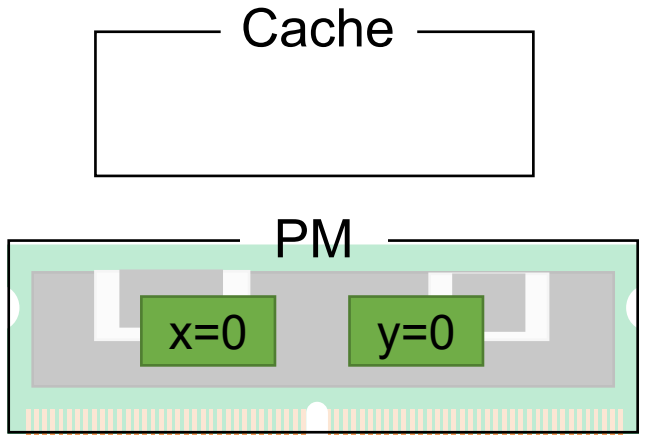
Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance

Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance
- PM-specific concurrency bugs exist

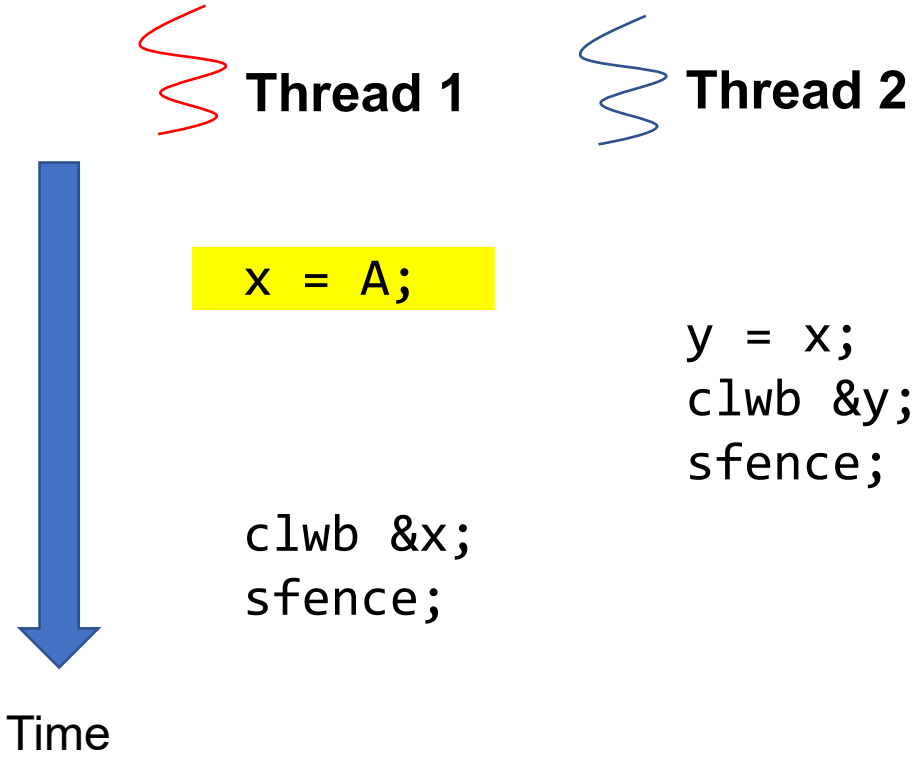


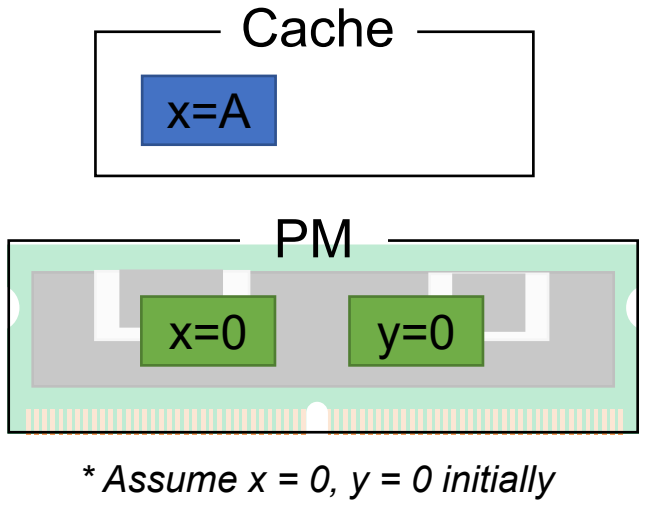


** Assume x = 0, y = 0 initially*

Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance
- PM-specific concurrency bugs exist

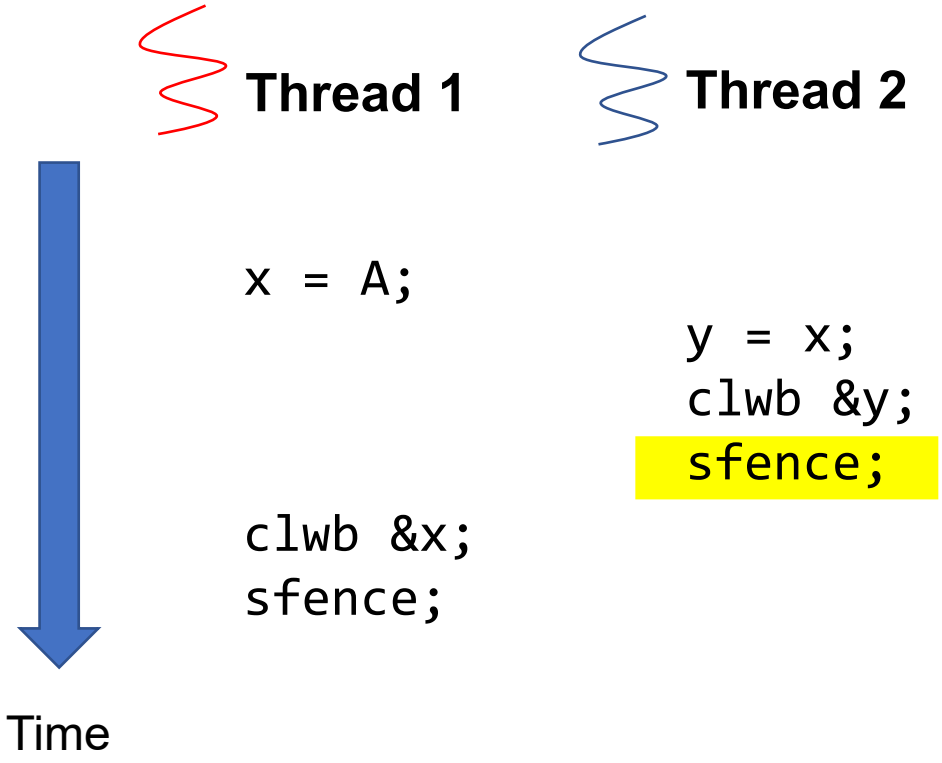


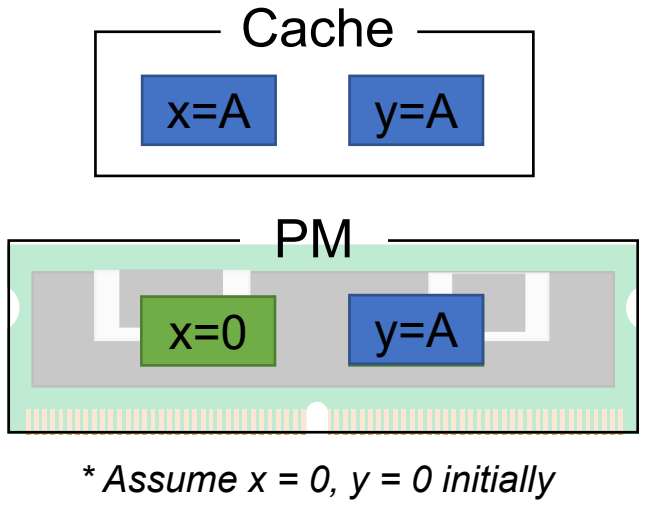


** Assume x = 0, y = 0 initially*

Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance
- PM-specific concurrency bugs exist

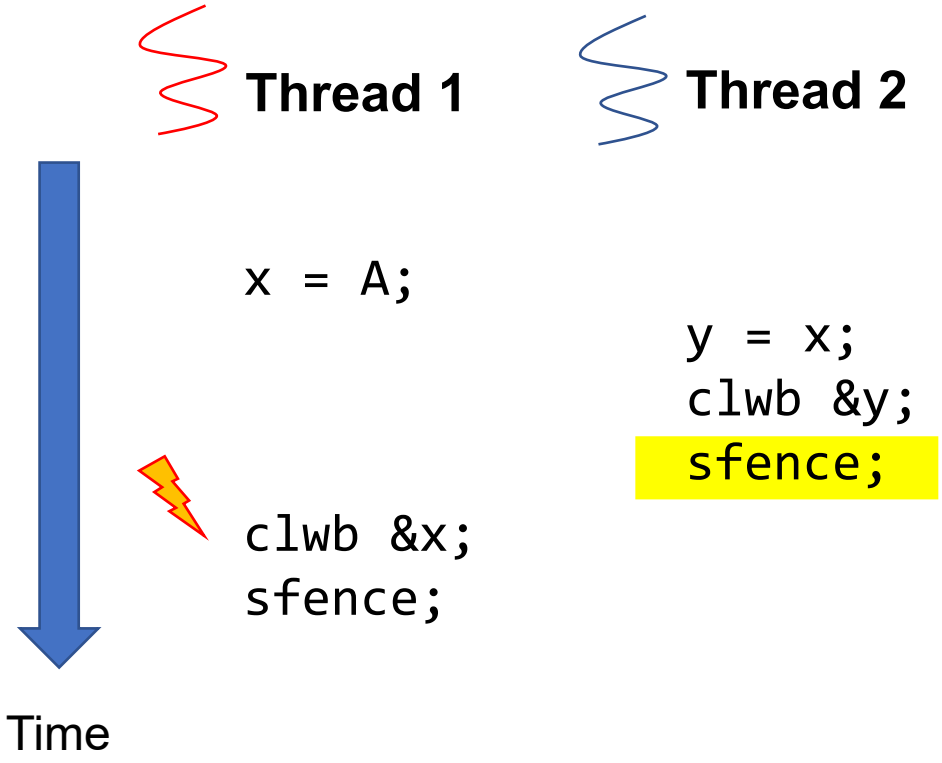


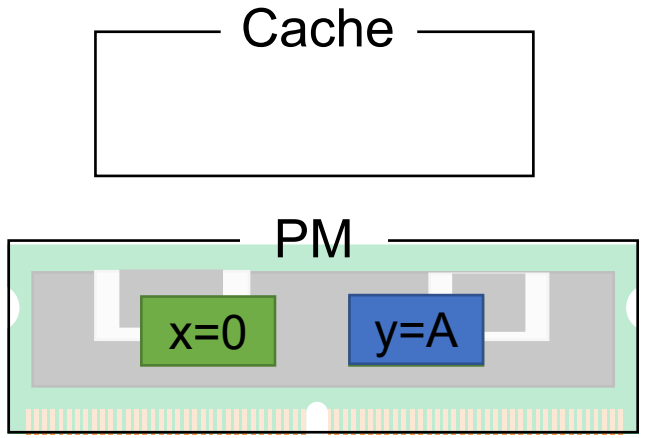


** Assume x = 0, y = 0 initially*

Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance
- PM-specific concurrency bugs exist

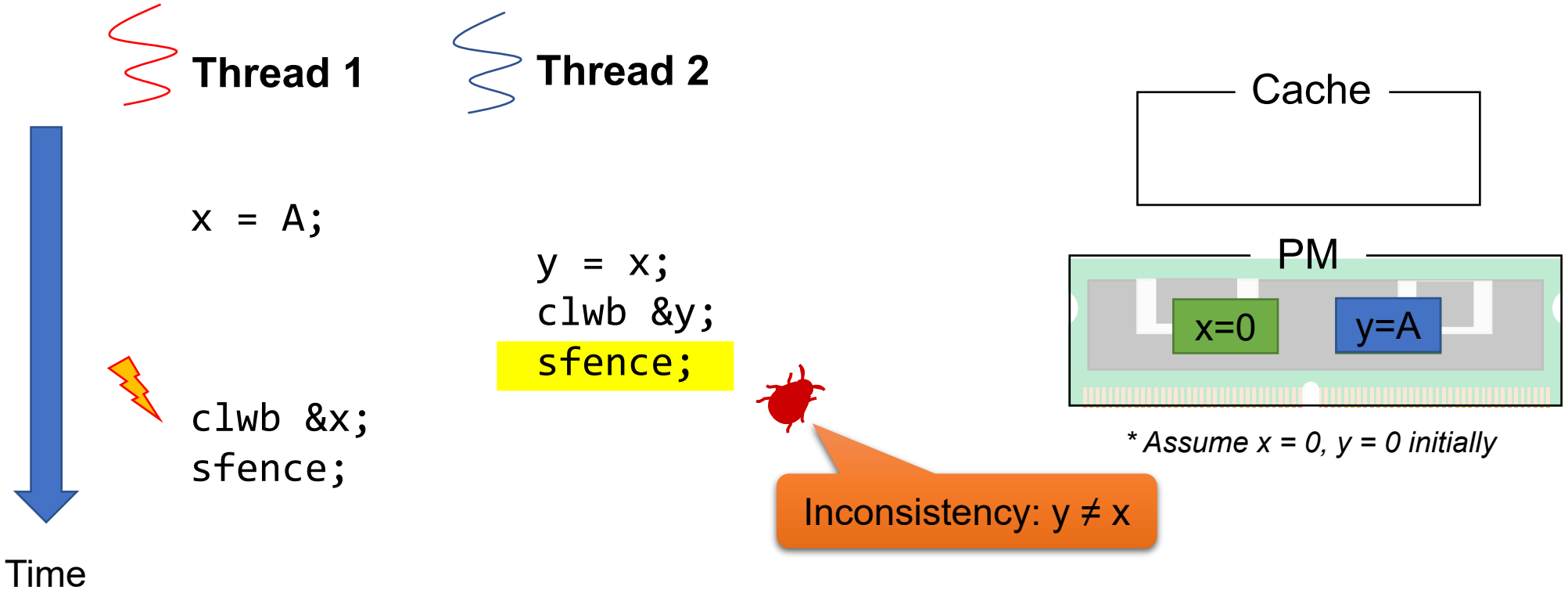




** Assume x = 0, y = 0 initially*

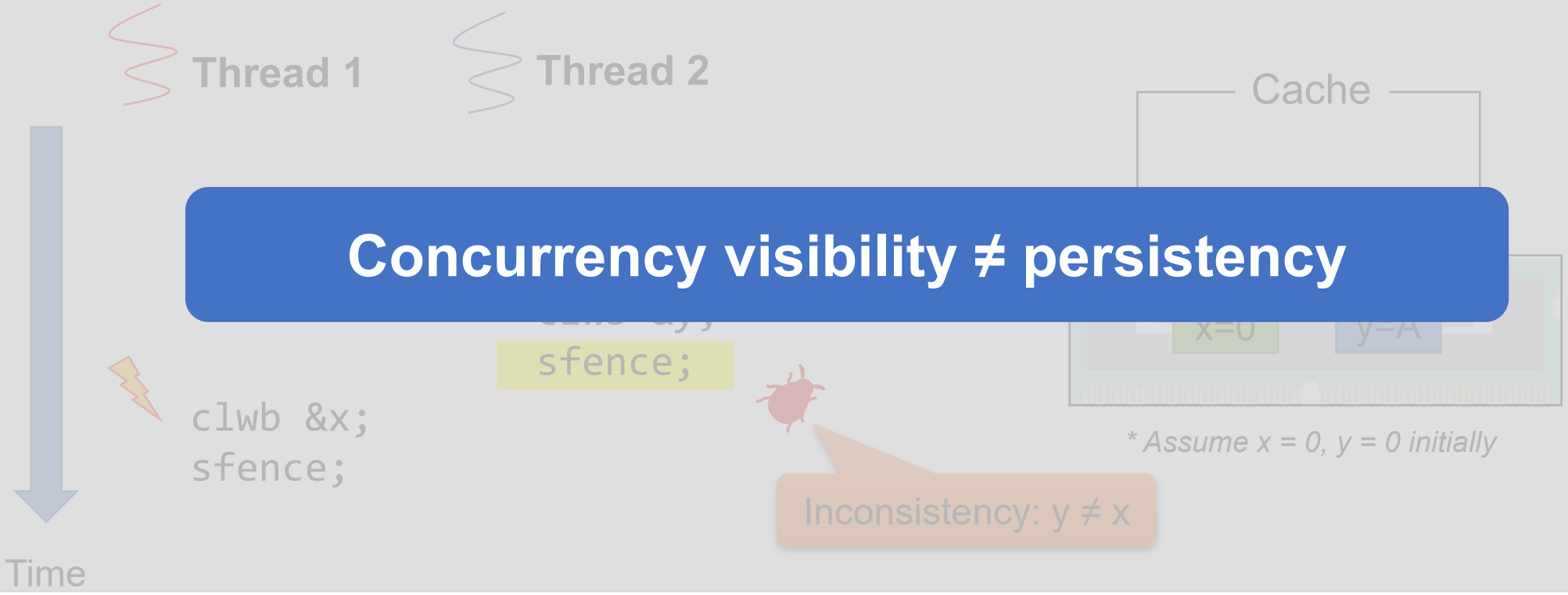
Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance
- PM-specific concurrency bugs exist



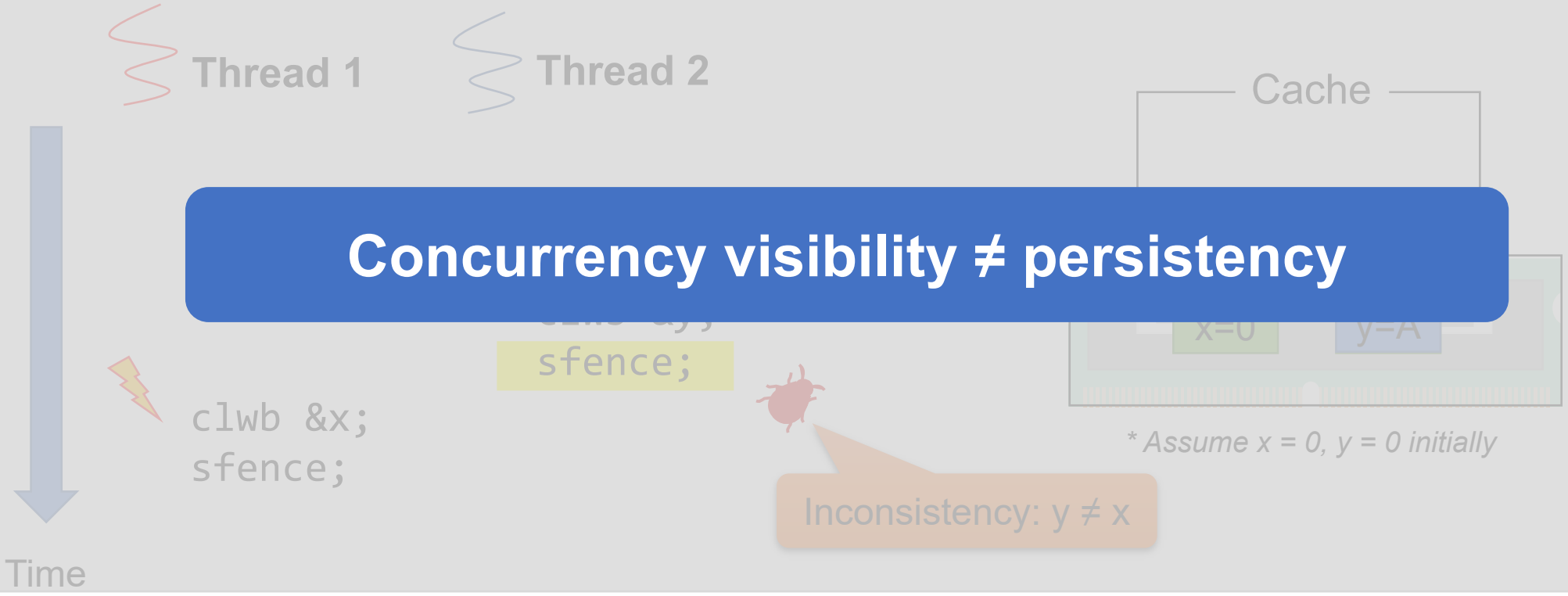
Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance
- PM-specific concurrency bugs exist



Crash-Inconsistency in Concurrent Executions

- Concurrency is important to PM system performance
- PM-specific concurrency bugs exist



No PM debugging tools detect buggy thread interleavings!

Summary of Debugging Tools

➢ Targets and platforms

	Sequential Bugs	Concurrency Bugs
DRAM	• KLEE [OSDI '08] • LLVM sanitizers: AddressSanitizer [ATC '12], MemorySanitizer [CGO '15], … • …	• AVIO [ASPLOS '06] • TSVD [SOSP '19] • Krace [S&P '20] • Kard [ASPLOS '21] • …
PM	• XFDetector [ASPLOS '20] • AGAMOTTO [OSDI '20] • PMDebugger [ASPLOS '21] • WITCHER [SOSP '21] • …	Our approach: PMRace

Challenges for PM Concurrency Bug Detection

- Exponential interleaving search space
  - Exponential growth rate with respect to instructions

Thread 1

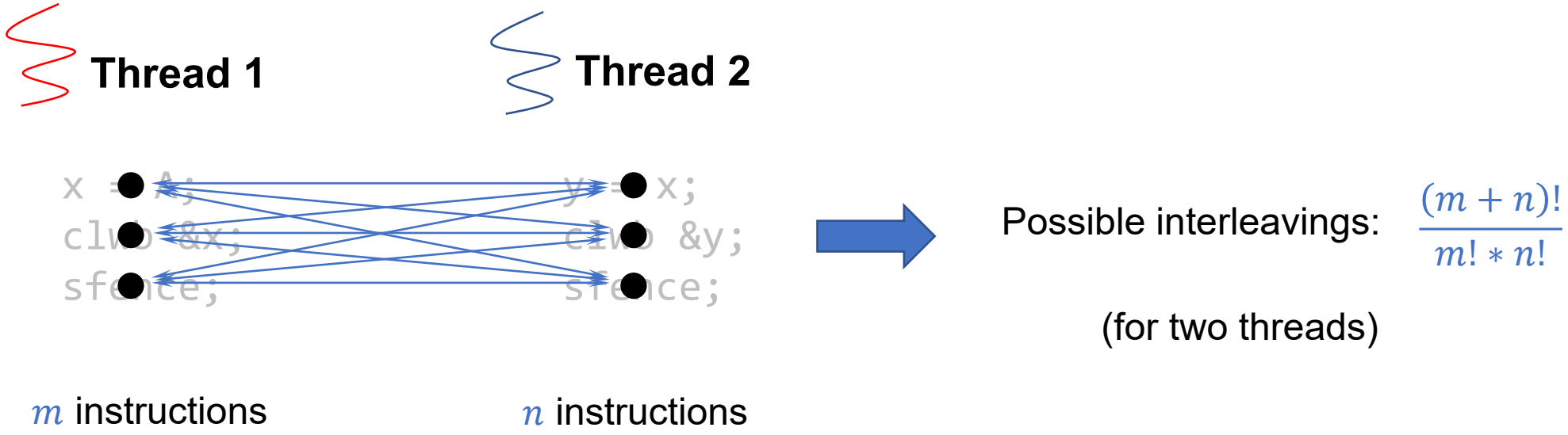
```
x = A;
clwb &x;
sfence;
```

Thread 2

```
y = x;
clwb &y;
sfence;
```

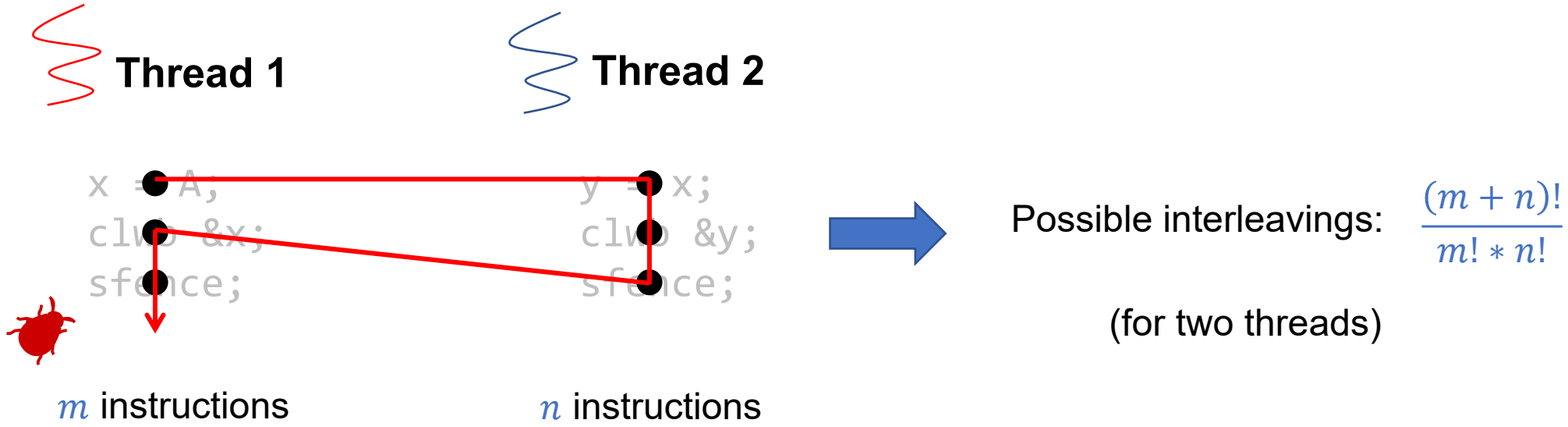
Challenges for PM Concurrency Bug Detection

- Exponential interleaving search space
  - Exponential growth rate with respect to instructions



Challenges for PM Concurrency Bug Detection

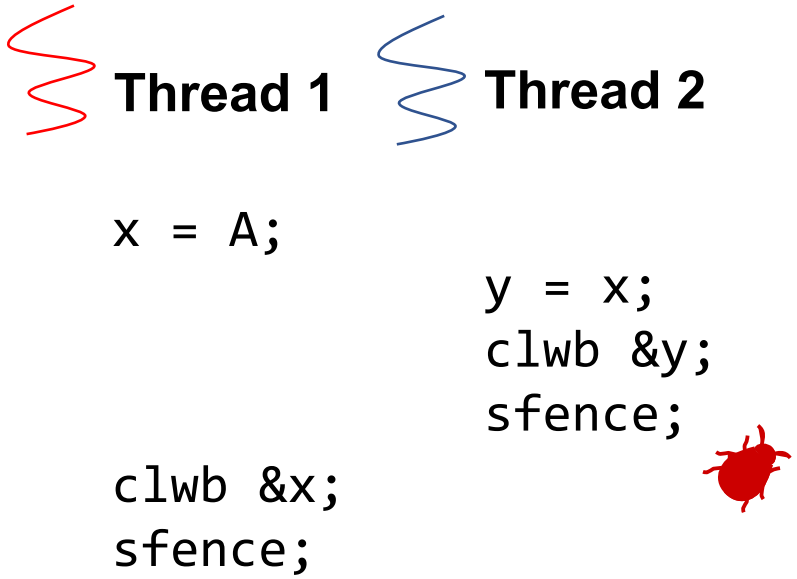
- Exponential interleaving search space
  - Exponential growth rate with respect to instructions



Challenges for PM Concurrency Bug Detection

- Exponential interleaving search space

- False positive
  - Definition: a detected bug is not a true bug
  - Reasons: inaccurate checkers, application-specific recovery…




```
Recover() {
  // Assume x = 0, y = 0 initially
  if (y != 0 && y != x) {
    // Handle inconsistent x and y
  }
}
```

An example of custom recovery code

Our Approach: PMRace

- Two new PM concurrency bug patterns
  - *PM Inter-thread Inconsistency* and *PM Synchronization Inconsistency*
- A fuzzer for PM concurrency bugs
  - Exponential interleaving: PM-aware coverage-guided fuzzing
  - False positive: Post-failure validation
- Found 14 bugs in 5 concurrent PM programs

The Two Bug Patterns

- PM Inter-thread Inconsistency
  - Durable side effects (e.g., PM writes) based on non-persisted data written by other threads

Thread 1

Thread 2

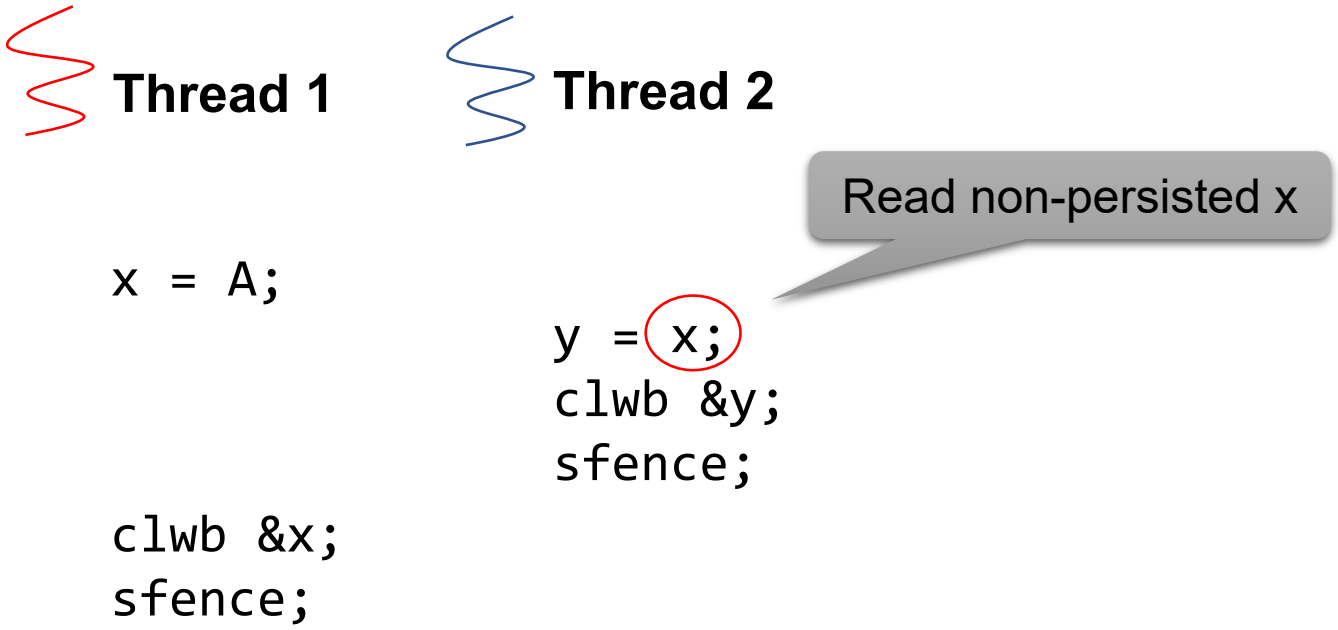
```
x = A;
```

```
clwb &x;
sfence;
```

```
y = x;
clwb &y;
sfence;
```

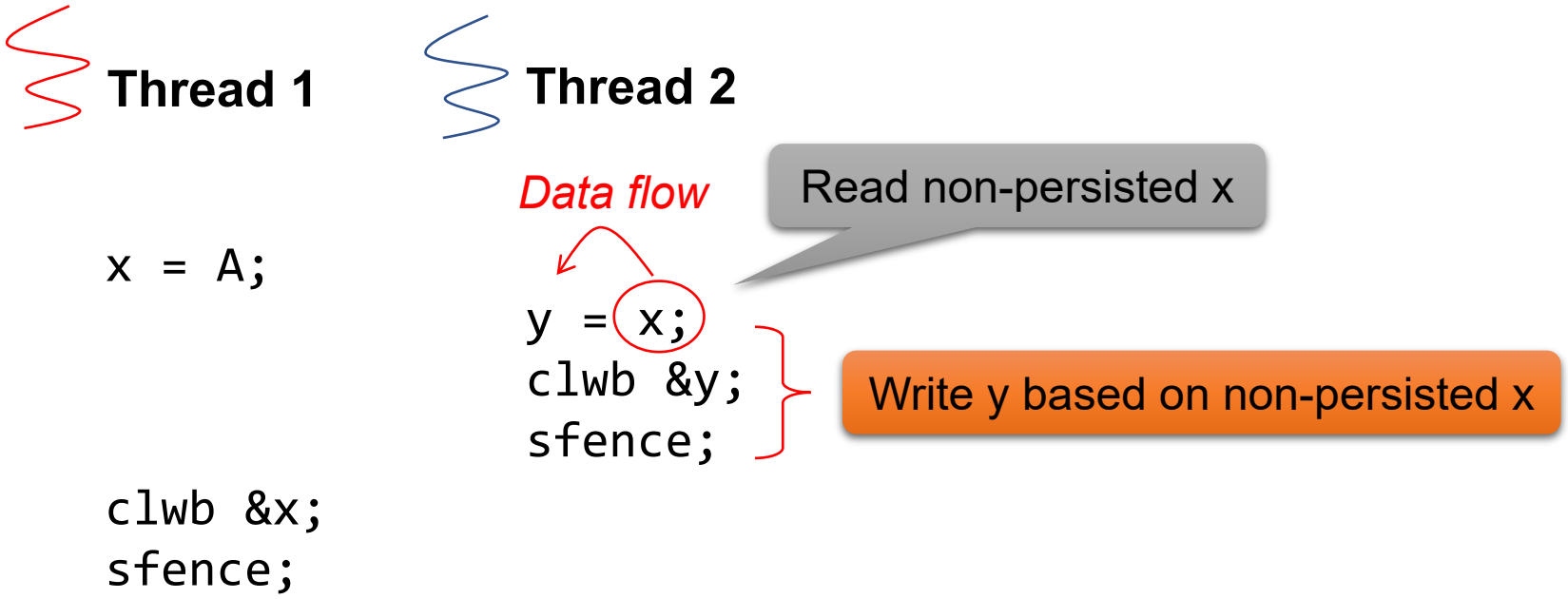
The Two Bug Patterns

- PM Inter-thread Inconsistency
  - Durable side effects (e.g., PM writes) based on non-persisted data written by other threads



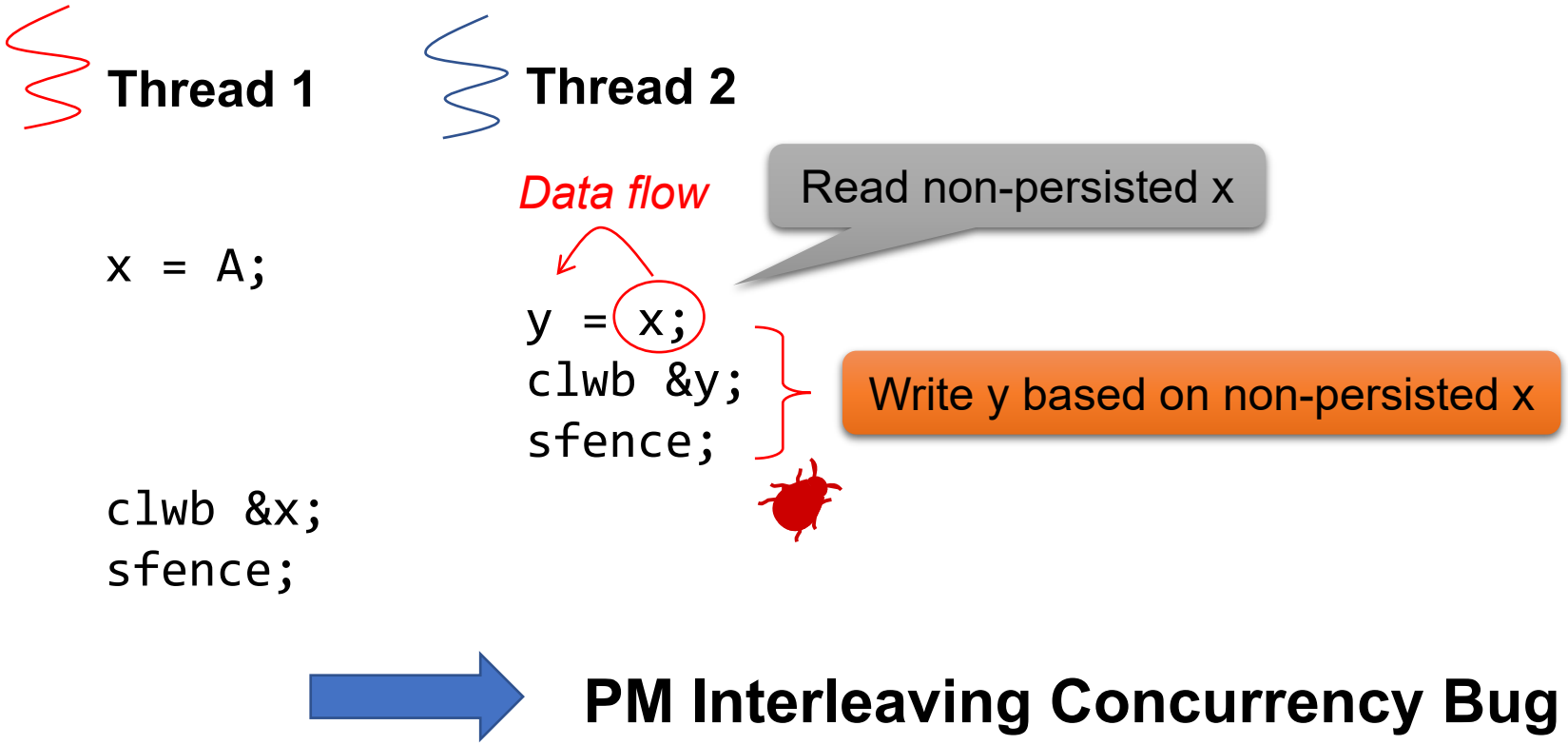
The Two Bug Patterns

- PM Inter-thread Inconsistency
  - Durable side effects (e.g., PM writes) based on non-persisted data written by other threads



The Two Bug Patterns

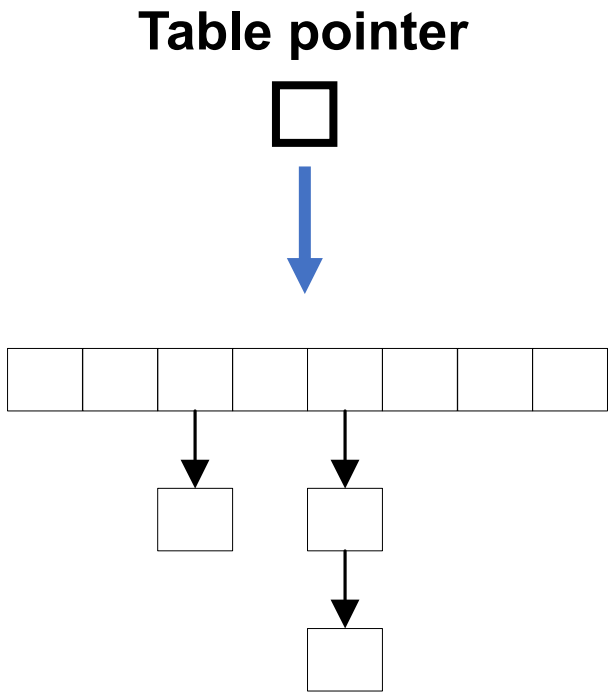
- PM Inter-thread Inconsistency
  - Durable side effects (e.g., PM writes) based on non-persisted data written by other threads



PM Interleaving Concurrency Bug

A PM Inter-thread Inconsistency in P-CLHT

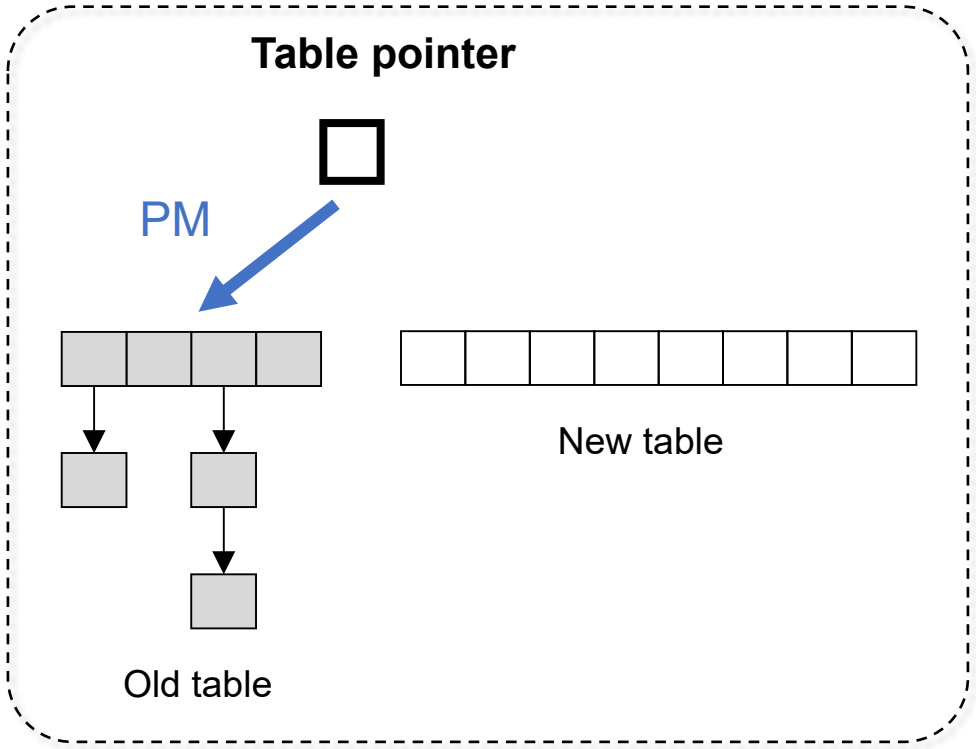
- P-CLHT from RECIPE [SOSP '19]
  - A chained hash table for PM
  - Lock-free read and bucket-grained locks for write



A PM Inter-thread Inconsistency in P-CLHT

Thread 1: ht_resize_pes

```
// Swap the hash table pointer for resizing
SWAP_U64(h->ht_off, pmemobj_oid(ht_new).off);
```




```
clwb(&h->ht_off); sfence();
```

Thread 2: ht_put

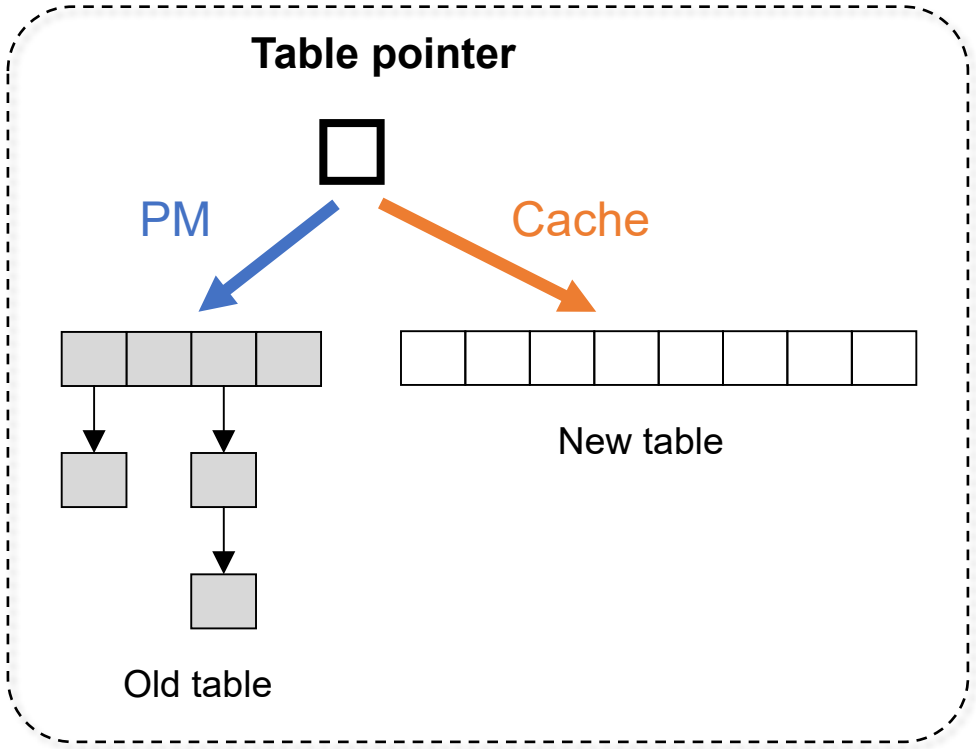
```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();
```

** The code is simplified for presentation*

A PM Inter-thread Inconsistency in P-CLHT

Thread 1: ht_resize_pes

```
// Swap the hash table pointer for resizing
SWAP_U64(h->ht_off, pmemobj_oid(ht_new).off);
```




```
clwb(&h->ht_off); sfence();
```

Thread 2: ht_put

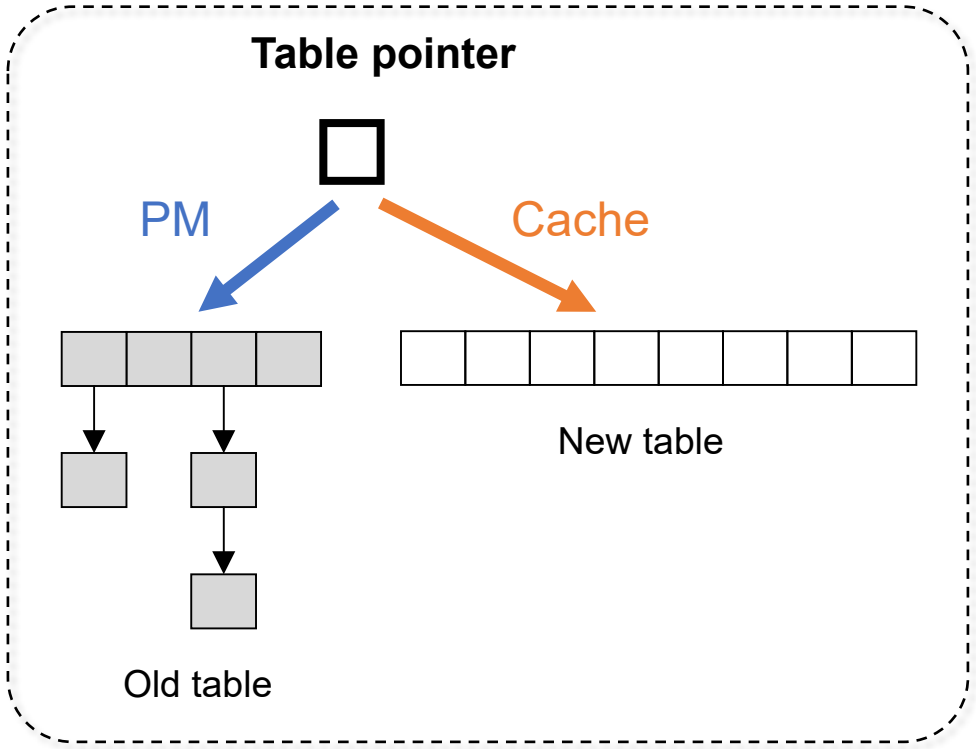
```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();
```

** The code is simplified for presentation*

A PM Inter-thread Inconsistency in P-CLHT

Thread 1: ht_resize_pes

```
// Swap the hash table pointer for resizing
SWAP_U64(h->ht_off, pmemobj_oid(ht_new).off);
```




```
clwb(&h->ht_off); sfence();
```

Thread 2: ht_put

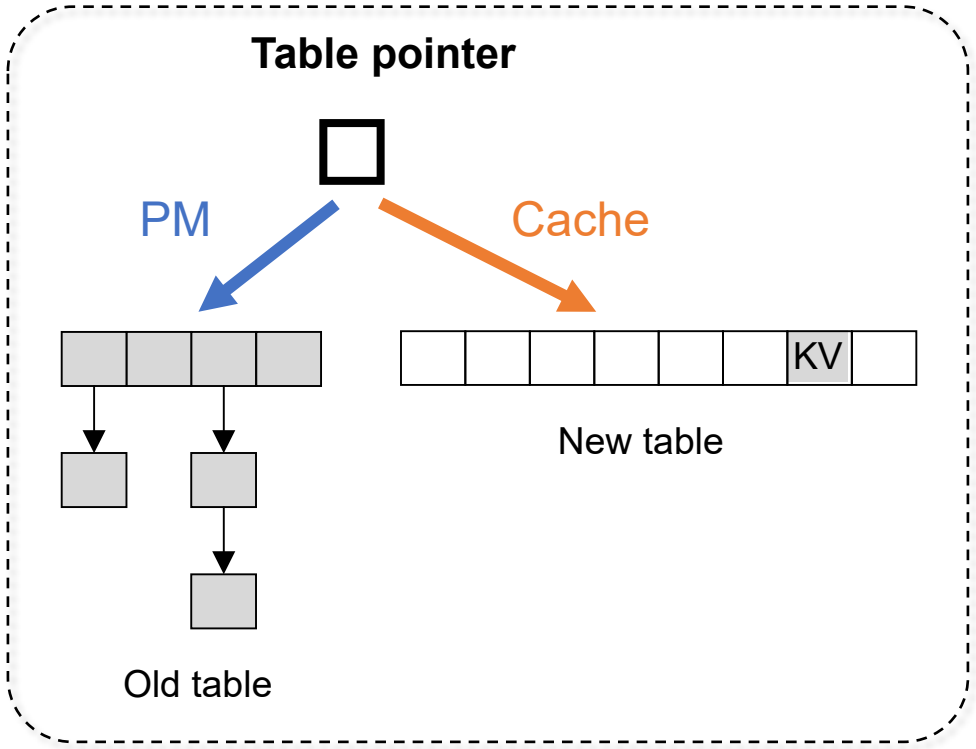
```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();
```

** The code is simplified for presentation*

A PM Inter-thread Inconsistency in P-CLHT

Thread 1: ht_resize_pes

```
// Swap the hash table pointer for resizing
SWAP_U64(h->ht_off, pmemobj_oid(ht_new).off);
```




```
clwb(&h->ht_off); sfence();
```

Thread 2: ht_put

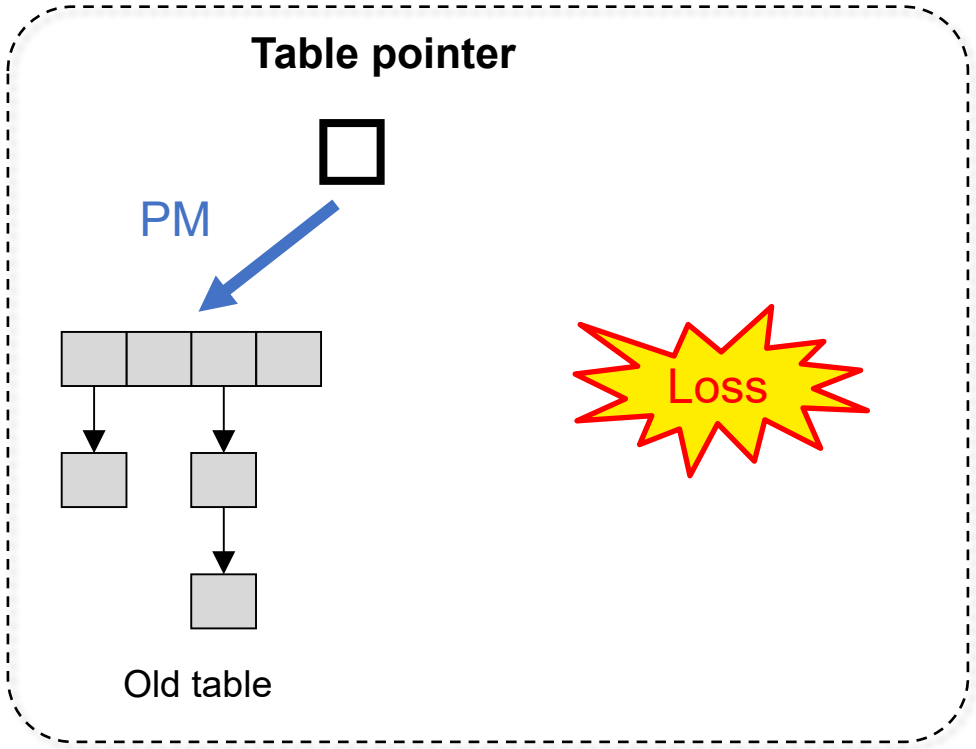
```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();
```

** The code is simplified for presentation*

A PM Inter-thread Inconsistency in P-CLHT

Thread 1: ht_resize_pes

```
// Swap the hash table pointer for resizing
SWAP_U64(h->ht_off, pmemobj_oid(ht_new).off);
```




```
clwb(&h->ht_off); sfence();
```

Thread 2: ht_put

```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();
```

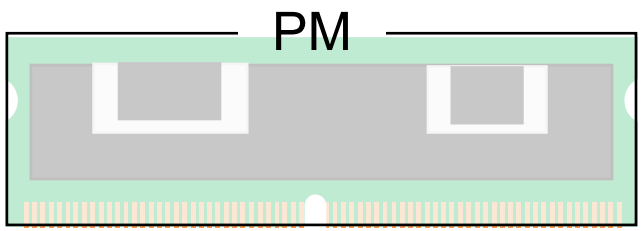
** The code is simplified for presentation*

The Two Bug Patterns

- PM Inter-thread Inconsistency
  - Durable side effects (e.g., PM writes) based on non-persisted data written by other threads

- PM Synchronization Inconsistency
  - Unreleased synchronization data after restarts

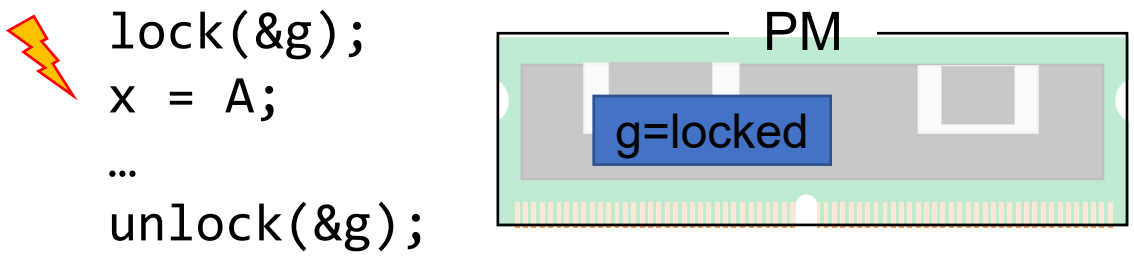
```
lock(&g);
x = A;
…
unlock(&g);
```



The Two Bug Patterns

- PM Inter-thread Inconsistency
  - Durable side effects (e.g., PM writes) based on non-persisted data written by other threads

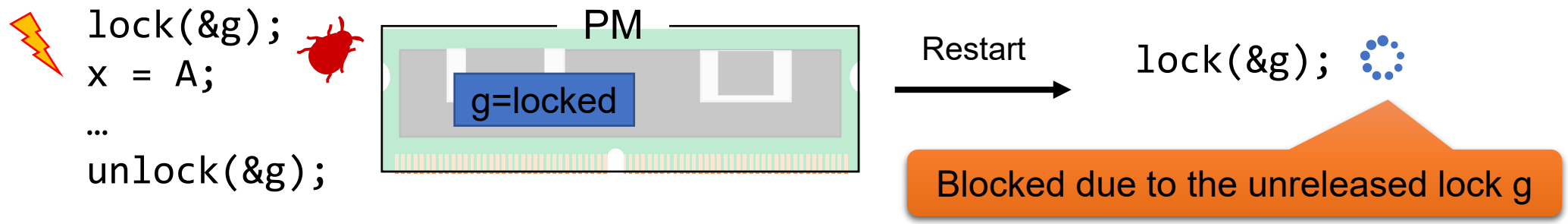
- PM Synchronization Inconsistency
  - Unreleased synchronization data after restarts



The Two Bug Patterns

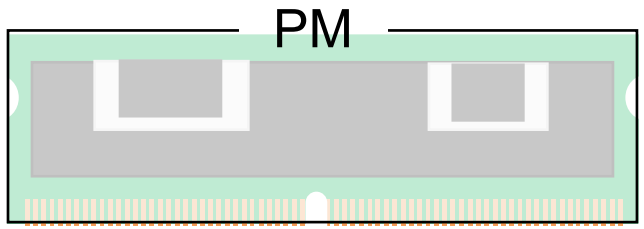
- PM Inter-thread Inconsistency
  - Durable side effects (e.g., PM writes) based on non-persisted data written by other threads

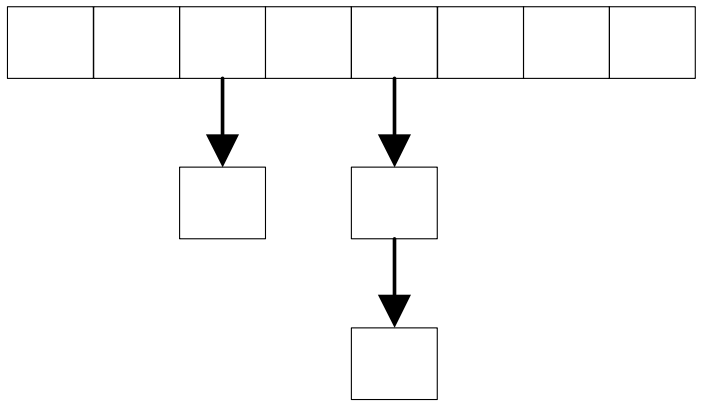
- PM Synchronization Inconsistency
  - Unreleased synchronization data after restarts



PM Execution Context Bug

A PM Synchronization Inconsistency in P-CLHT



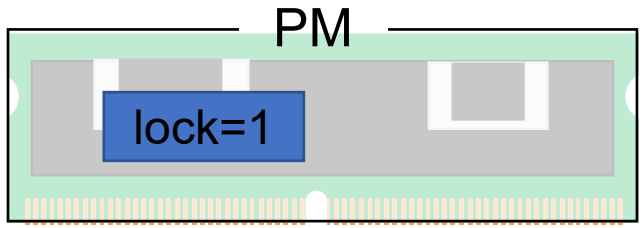


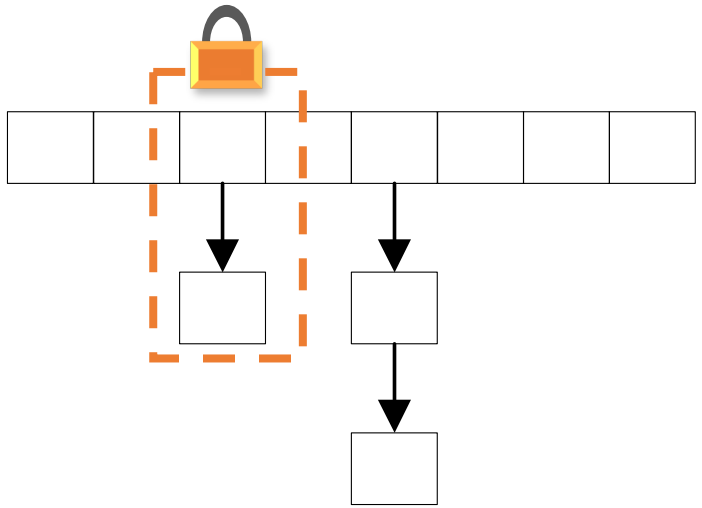
Thread x: ht_put

```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();

LOCK_RLS(lock);
```

** The code is simplified for presentation*

A PM Synchronization Inconsistency in P-CLHT



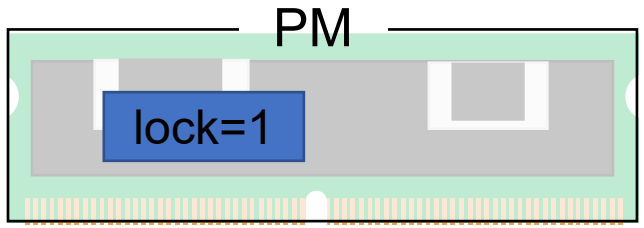


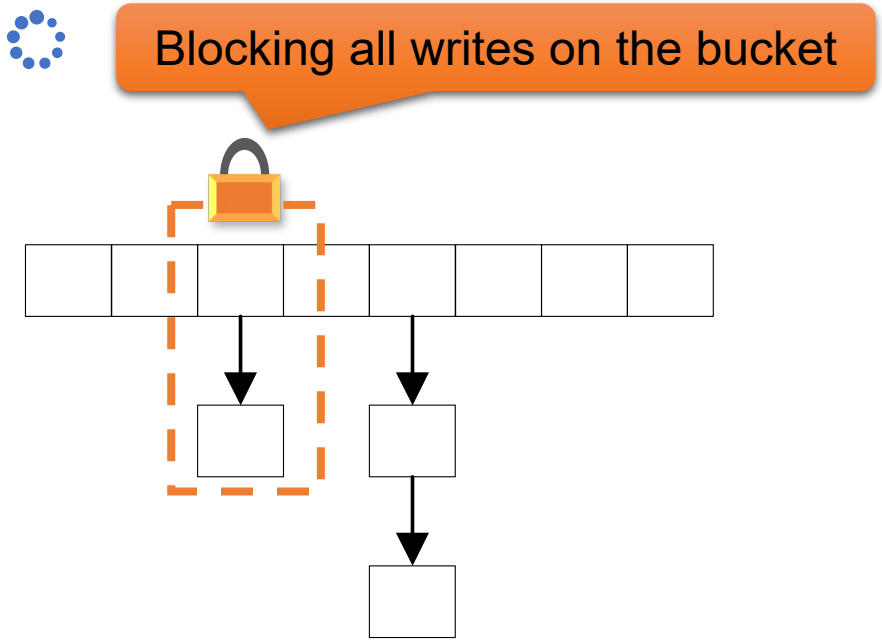
Thread x: ht_put

```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();

LOCK_RLS(lock);
```

** The code is simplified for presentation*

A PM Synchronization Inconsistency in P-CLHT



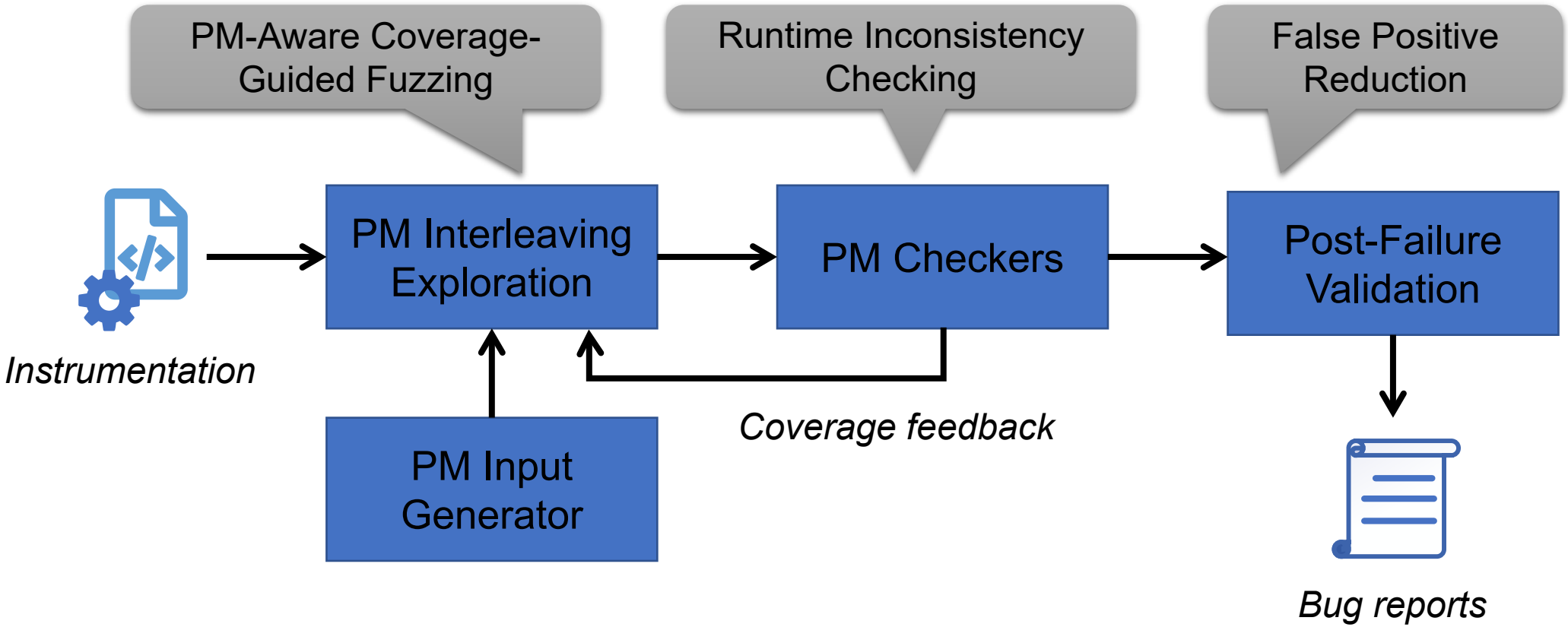


Thread x: ht_put

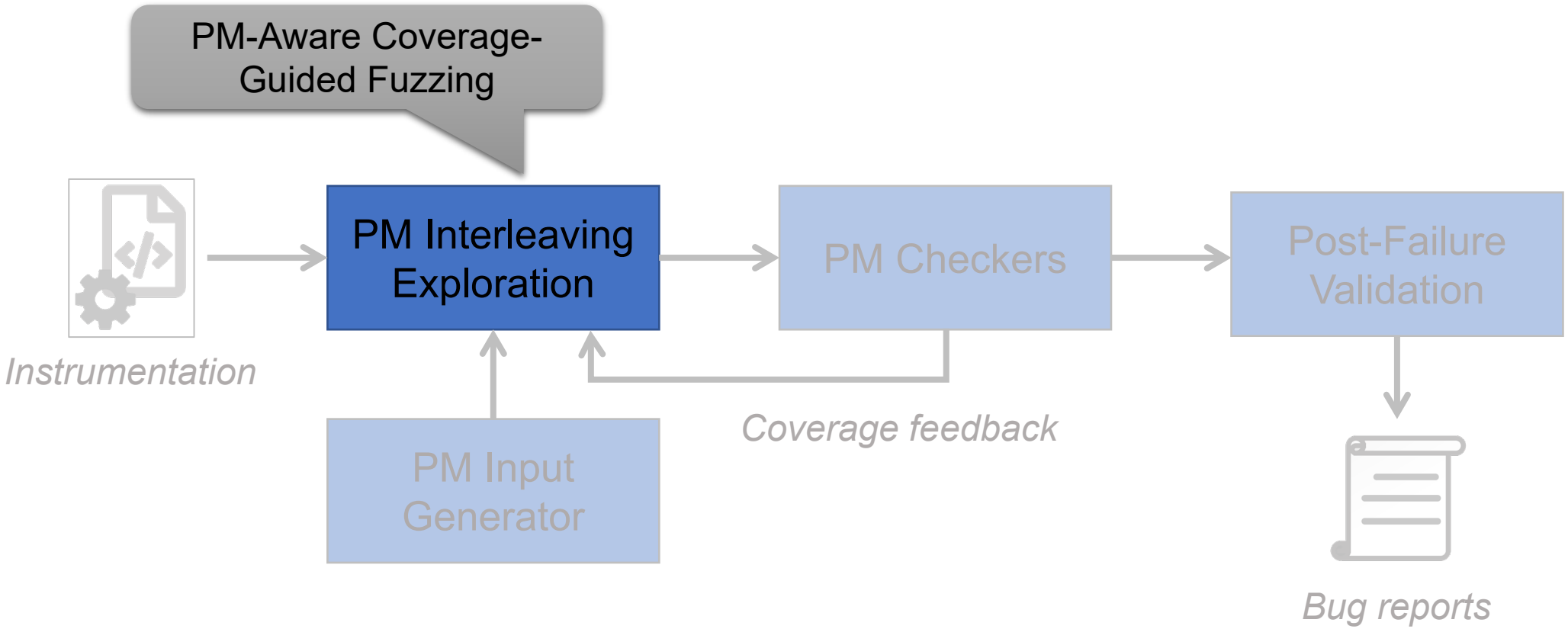
```
// Insert a key-value item
hashtable = clht_ptr_from_off(h->ht_off);
bin = clht_hash(hashtable, key)
bucket = clht_ptr_from_off(hashtable->table_off) + bin;

// Acquire the bucket lock
lock = &bucket->lock;
while (!LOCK_ACQ(lock, hashtable))
{
  ...
}

// Find an empty slot in the bucket
bucket->val[j] = val;
...
clwb(&bucket->val[j]); sfence();
movnt64(&bucket->key[j], key); sfence();

LOCK_RLS(lock);
```

** The code is simplified for presentation*

PMRace Overview

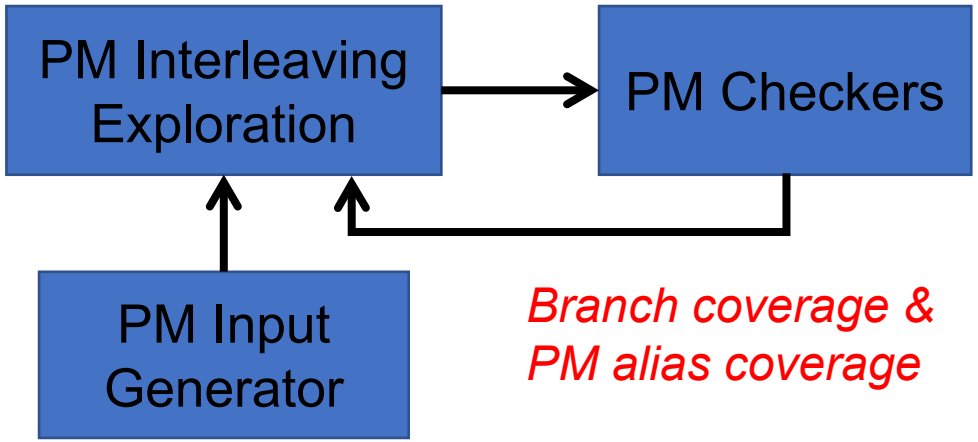


PMRace Overview



PM Alias (Pair) Coverage

- Recording concurrent PM accesses to the same address
- Guiding fuzzing to test “new” interleavings



PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data

Thread 1

```
x = A;
clwb &x;
sfence;
```

Thread 2

```
y = x;
clwb &y;
sfence;
```

PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
    - PM accesses to shared data
    - A priority queue of PM access

Thread 1

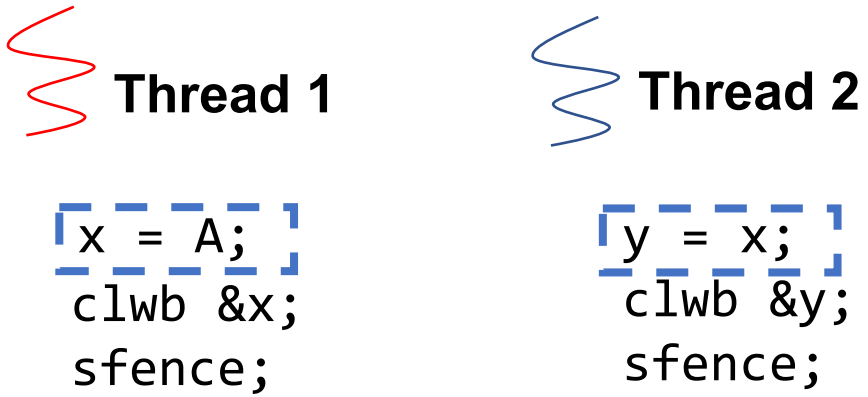
```
x = A;
clwb &x;
sfence;
```

Thread 2

```
y = x;
clwb &y;
sfence;
```

PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
    - PM accesses to shared data
    - A priority queue of PM access



PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
  - Step 2: scheduling a group of alias PM accesses (to the same address)
    - `cond_wait` before PM reads
    - `cond_signal` after corresponding PM writes and before flushes
    - Pitfalls and solutions… (refer to the paper)

Thread 1

```
x = A;
clwb &x;
sfence;
```

Thread 2

```
y = x;
clwb &y;
sfence;
```

PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
  - Step 2: scheduling a group of alias PM accesses (to the same address)
    - `cond_wait` before PM reads
    - `cond_signal` after corresponding PM writes and before flushes
    - Pitfalls and solutions… (refer to the paper)

Thread 1

```
x = A;
clwb &x;
sfence;
```

Thread 2

```
cond_wait(&m);
y = x;
clwb &y;
sfence;
```

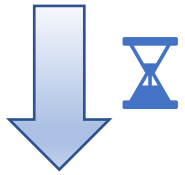
PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
  - Step 2: scheduling a group of alias PM accesses (to the same address)
    - `cond_wait` before PM reads
    - `cond_signal` after corresponding PM writes and before flushes
    - Pitfalls and solutions… (refer to the paper)

Thread 1

Thread 2

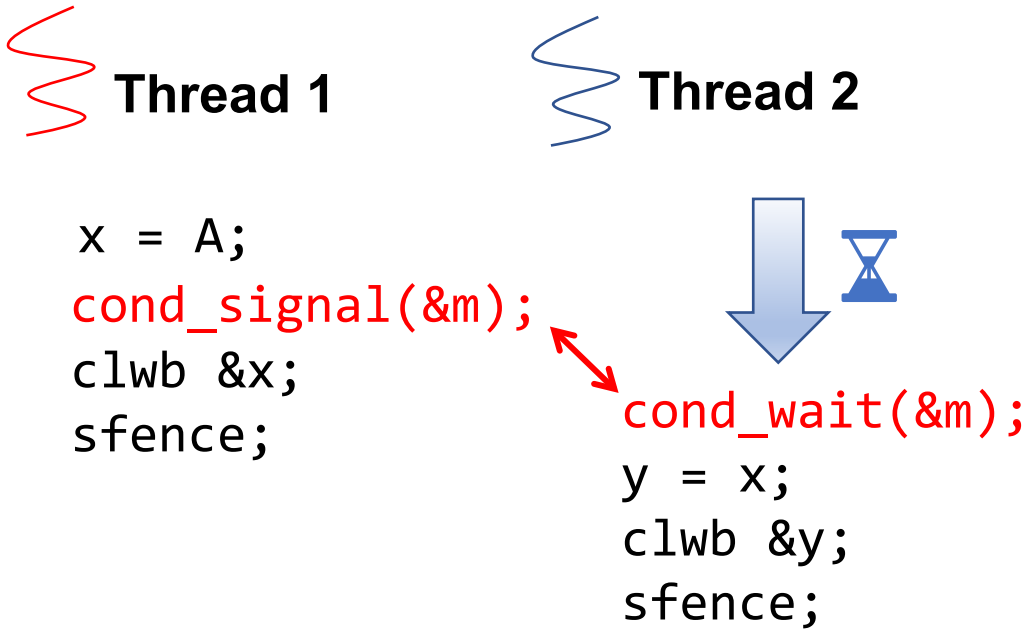
```
x = A;
clwb &x;
sfence;
```



```
cond_wait(&m);
y = x;
clwb &y;
sfence;
```

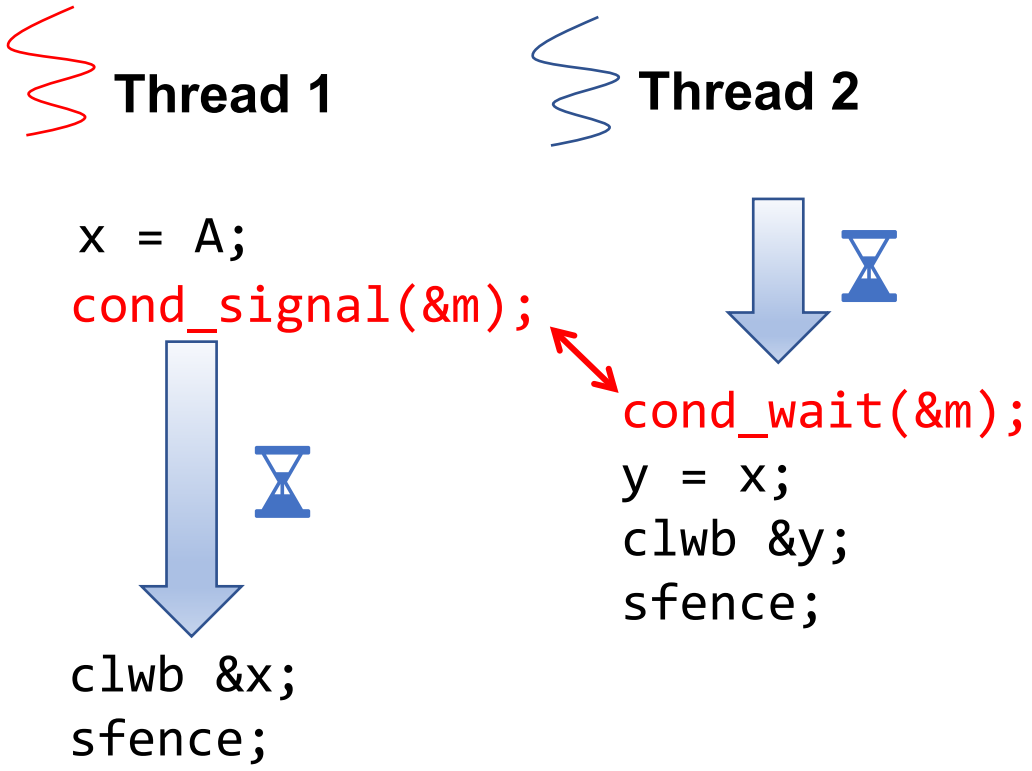
PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
  - Step 2: scheduling a group of alias PM accesses (to the same address)
    - `cond_wait` before PM reads
    - `cond_signal` after corresponding PM writes and before flushes
    - Pitfalls and solutions… (refer to the paper)



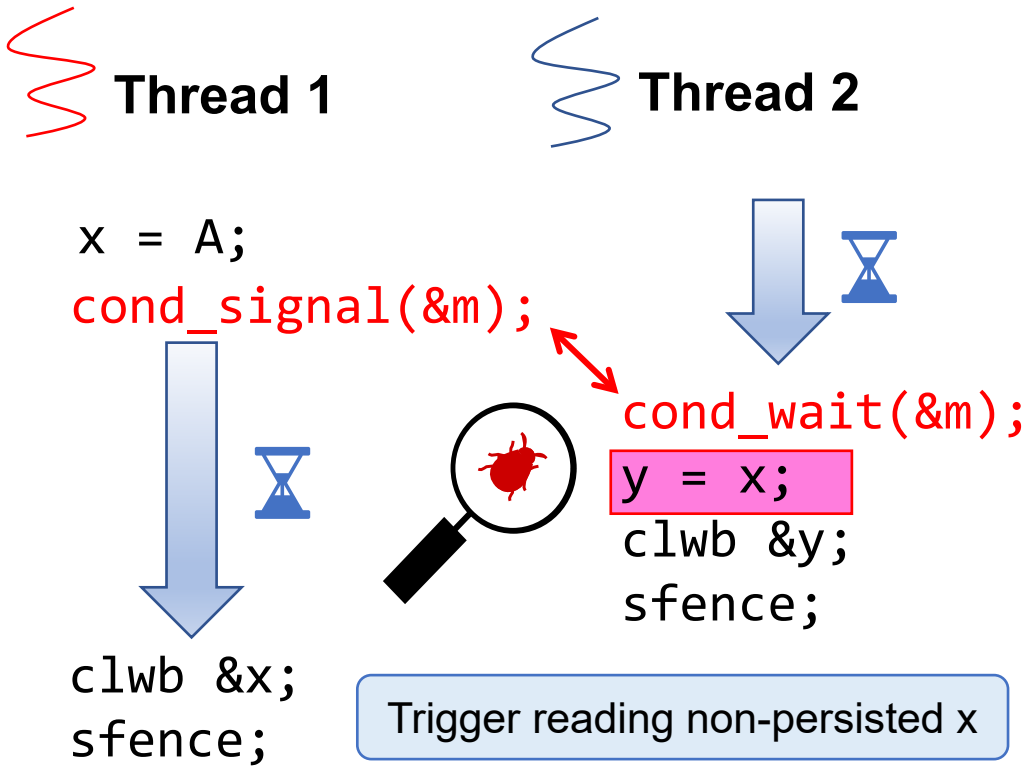
PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
  - Step 2: scheduling a group of alias PM accesses (to the same address)
    - `cond_wait` before PM reads
    - `cond_signal` after corresponding PM writes and before flushes
    - Pitfalls and solutions… (refer to the paper)

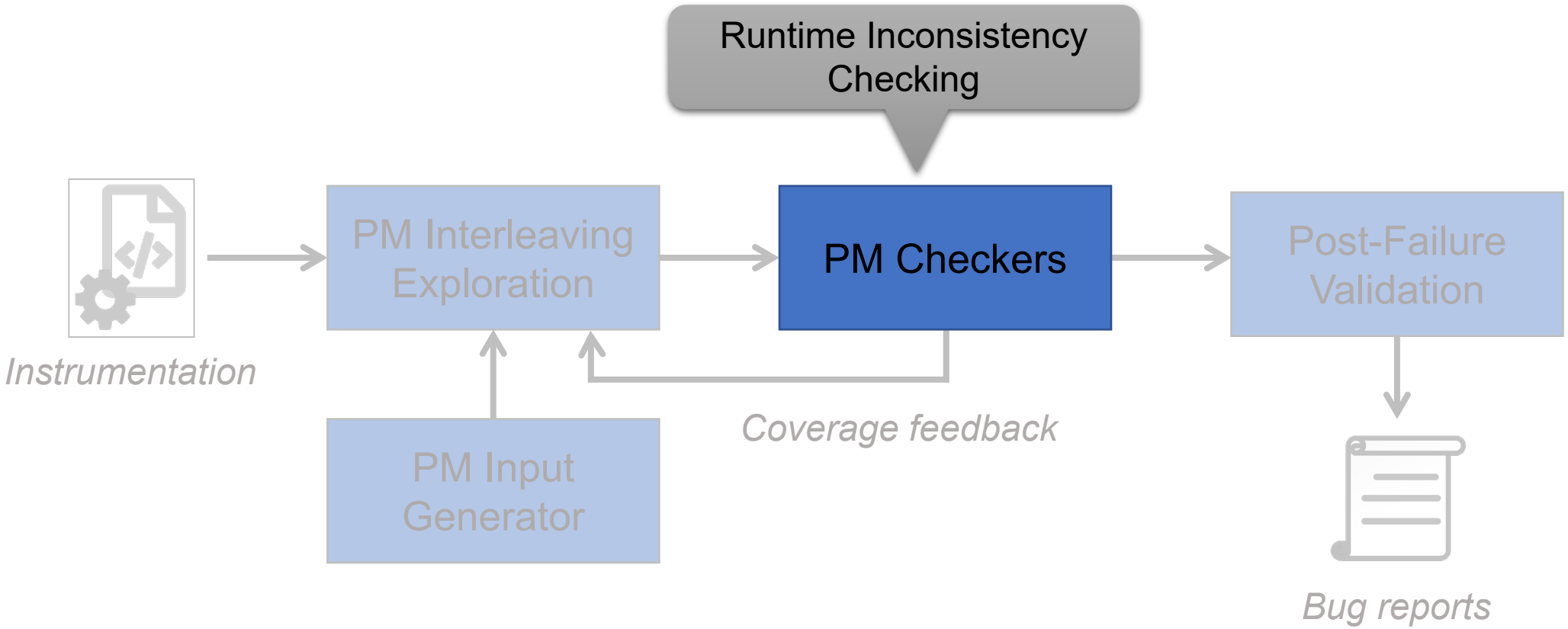


PM-Aware Interleaving Exploration

- Exploration scheme
  - Driving the execution towards reading non-persisted data
  - Step 1: preemption point selection
  - Step 2: scheduling a group of alias PM accesses (to the same address)
    - `cond_wait` before PM reads
    - `cond_signal` after corresponding PM writes and before flushes
    - Pitfalls and solutions… (refer to the paper)

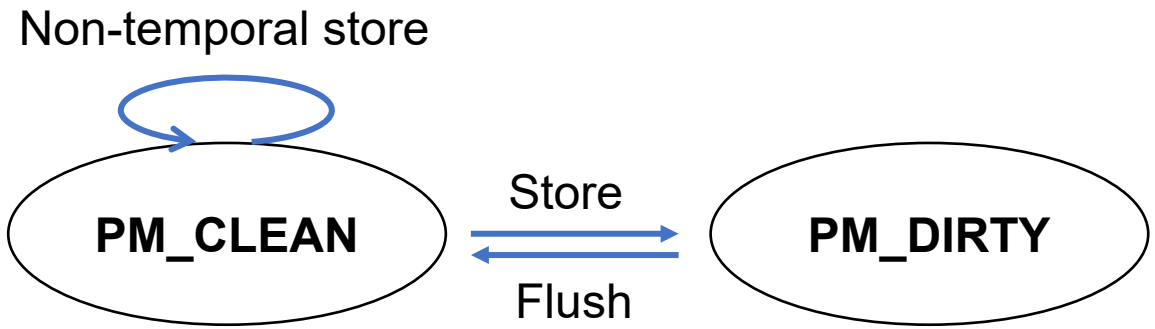


PMRace Overview



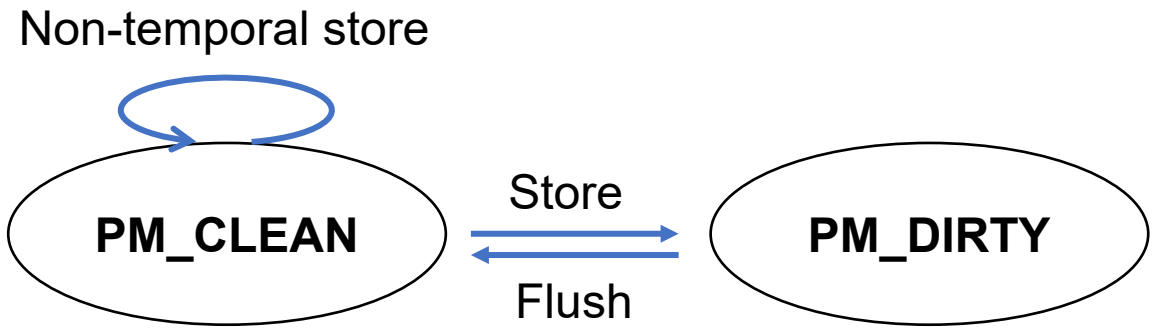
PM Inconsistency Checkers

- Persistency state tracking

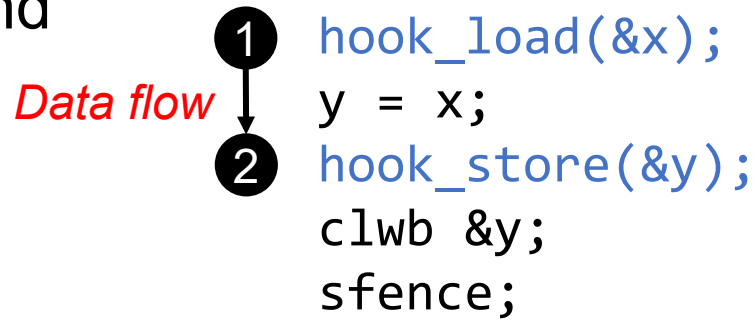


PM Inconsistency Checkers

- Persistency state tracking

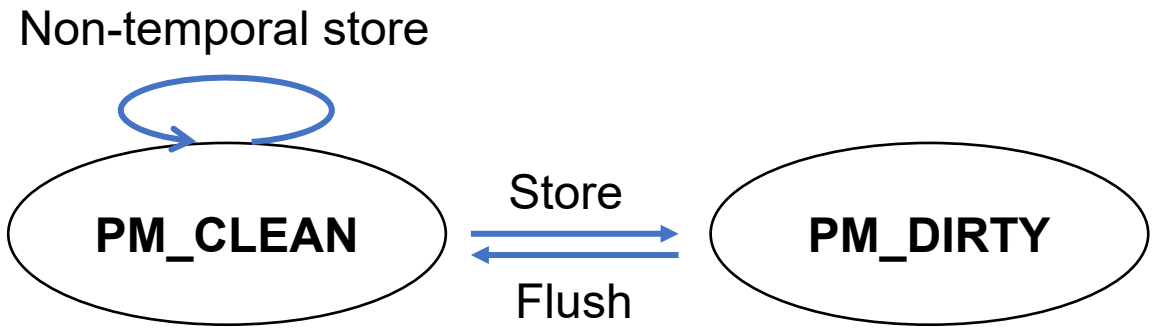


- Runtime PM checkers
  - *PM Inter-thread Inconsistency* when
    1. Reading non-persisted data (PM_DIRTY) and
    2. causing durable side effects (PM writes)



PM Inconsistency Checkers

- Persistency state tracking

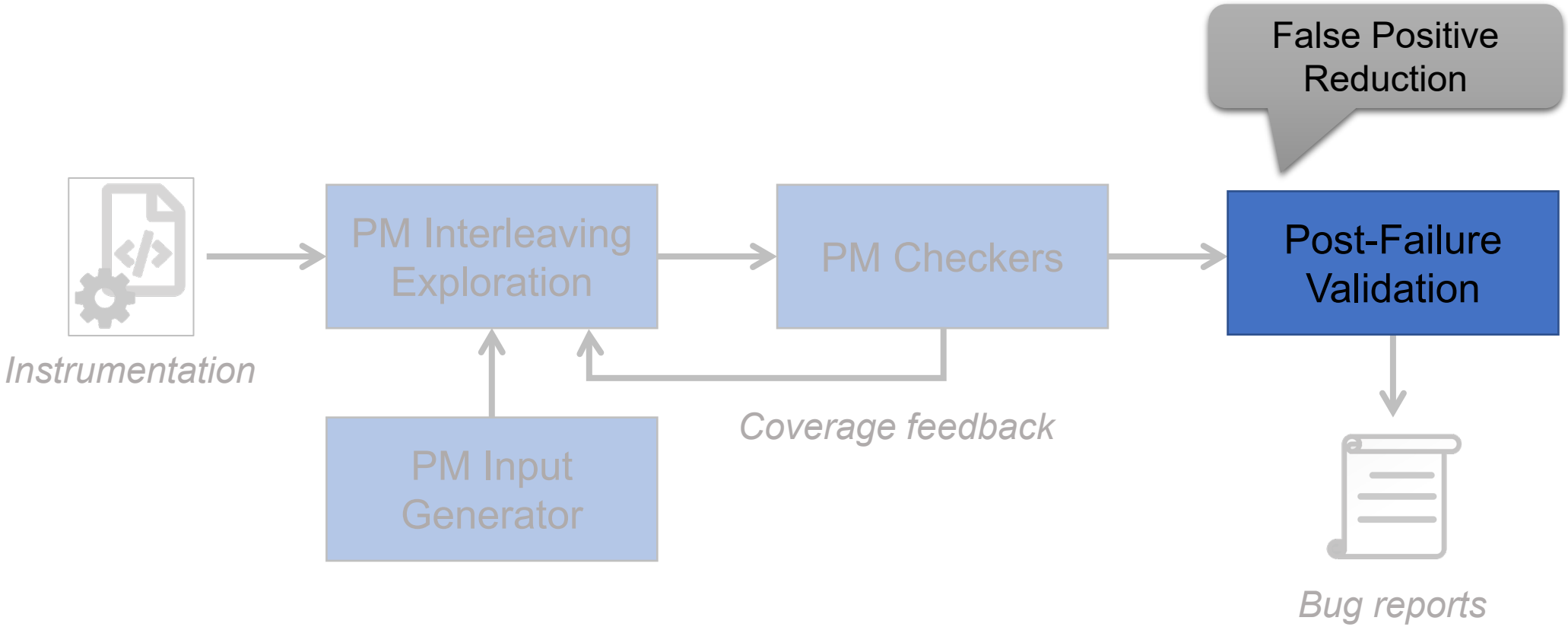


- Runtime PM checkers
  - *PM Inter-thread Inconsistency* when
    1. Reading non-persisted data (PM_DIRTY) and
    2. causing durable side effects (PM writes)
  - *PM Synchronization Inconsistency* when
    1. Updating annotated synchronization data

Data flow

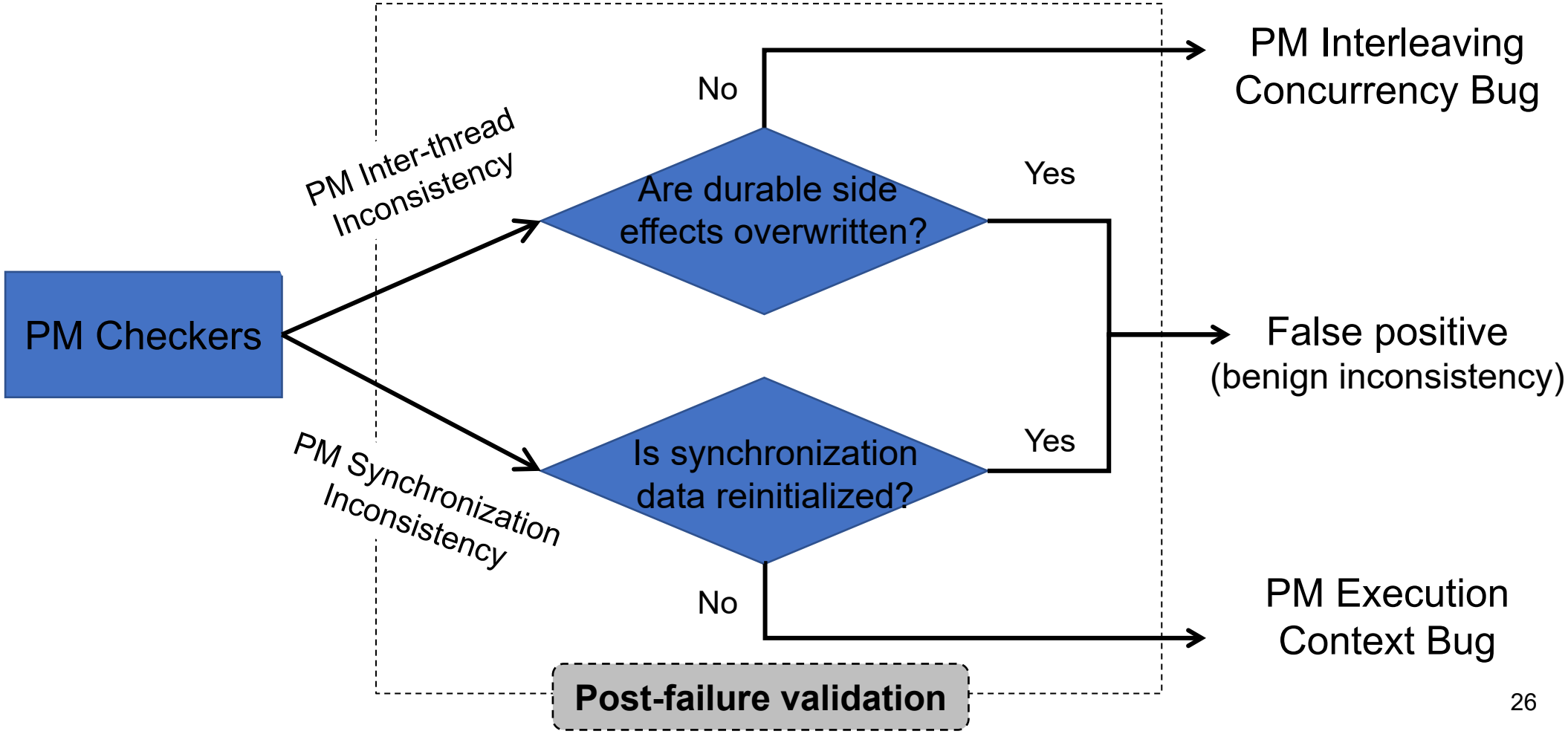
```
1  hook_load(&x);
   y = x;
2  hook_store(&y); 1
   clwb &y;
   sfence;
```

PMRace Overview



Post-Failure Validation

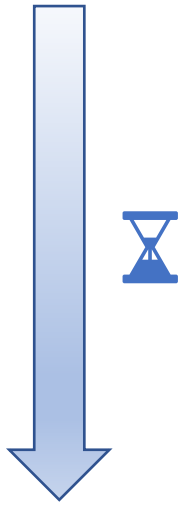
- To identify false positive (benign inconsistency)



An Example of Benign Inconsistency

Thread 1

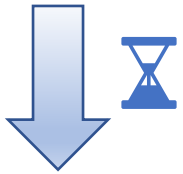
```
x = A;
hook_store(&x);
cond_signal(&m);
```



```
clwb &x;
sfence;
```

Thread 2

```
cond_wait(&m);
```



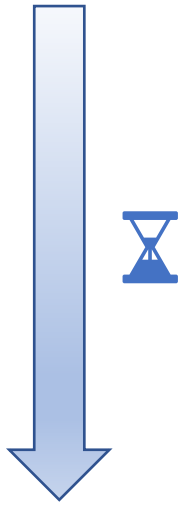
```
hook_load(&x);
y = x;
hook_store(&y);
clwb &y;
sfence;
```

```
Recover() {
  // Assume x = 0, y = 0 initially
  if (y != 0 && y != x) {
    // Handle inconsistent x and y
    x = 0;
    y = 0;
  }
}
```

An Example of Benign Inconsistency

Thread 1

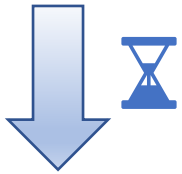
```
x = A;
hook_store(&x);
cond_signal(&m);
```



```
clwb &x;
sfence;
```

Thread 2

```
cond_wait(&m);
```



```
hook_load(&x);
y = x;
hook_store(&y);
clwb &y;
sfence;
```

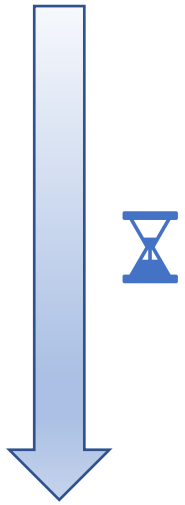
```
Recover() {
  // Assume x = 0, y = 0 initially
  if (y != 0 && y != x) {
    // Handle inconsistent x and y
    x = 0;
    y = 0;
  }
}
```

PM checker: PM Inter-thread Inconsistency

An Example of Benign Inconsistency

Thread 1

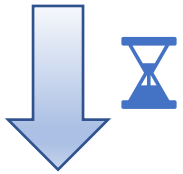
```
x = A;
hook_store(&x);
cond_signal(&m);
```



```
clwb &x;
sfence;
```

Thread 2

```
cond_wait(&m);
```



```
hook_load(&x);
y = x;
hook_store(&y);
clwb &y;
sfence;
```

```
Recover() {
  // Assume x = 0, y = 0 initially
  if (y != 0 && y != x) {
    // Handle inconsistent x and y
    x = 0;
    y = 0;
  }
}
```

Post-failure validation: benign inconsistency

PM checker: PM Inter-thread Inconsistency

Evaluation

- System configurations
  - Two 26-core Intel Xeon Gold 6230R CPUs
  - 1.5 TB Intel Optane PM 100 Series, 192 GB DRAM
- Tested 5 open-source concurrent PM programs based on PMDK

Systems	Scope	Concurrency
P-CLHT [SOSP '19]	Static hashing	Lock-based
Clevel Hashing [ATC '20]	PM-optimized hashing	Lock-free
CCEH [FAST '19]	Extendible hashing	Lock-based
FAST-FAIR [FAST '18]	B+-Tree	Lock-based
memcached-pmem	Key-value store	Lock-based

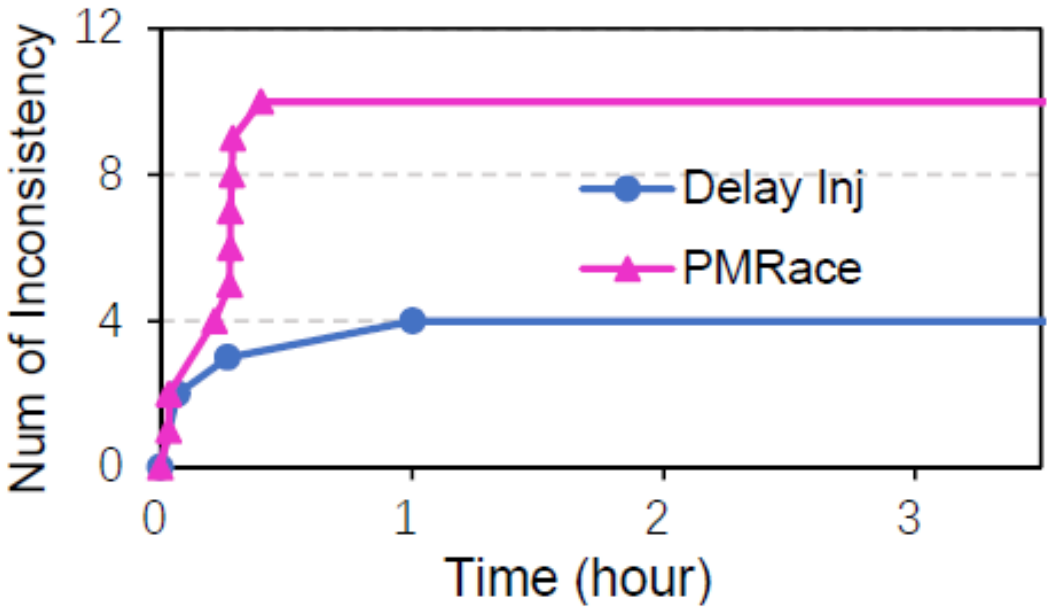
- Comparison
  - **PMRace**: our scheme
  - **Delay Inj**: PMRace with random delay injection for interleaving exploration

14 Bugs

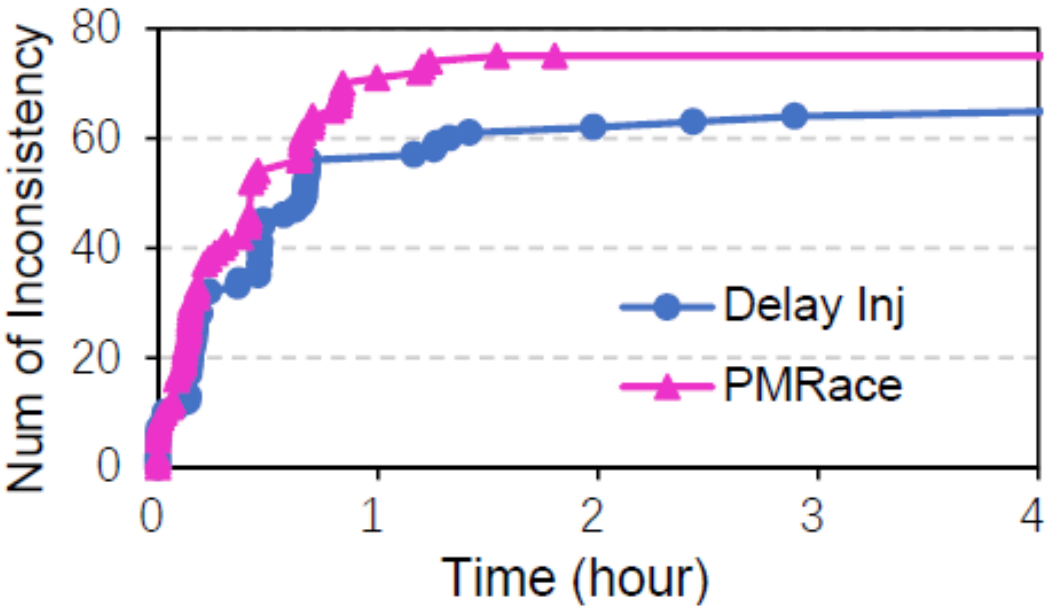
	#	Type	New	Description	Impact
P-CLHT	1	Inter	Y	read unflushed table pointer and insert items	data loss
	2	Sync	Y	do not initialize bucket locks after restarts	hang
	3	Intra	Y	read unflushed table pointer and perform GC	PM leakage
	4	Other	Y	read unflushed keys	redundant PM writes
	5	Other	Y	do not release bucket locks in update	hang
CCEH	6	Sync	Y	do not release segment locks after restarts	hang
	7	Intra	Y	read unflushed capacity and allocate segments	PM leakage
FAST-FAIR	8	Inter	Y	read unflushed pointer and insert data	data loss
memcached-pmem	9	Inter	Y	read unflushed value and write value	inconsistent data
	10	Inter	Y	read unflushed value and write value	inconsistent data
	11	Inter	N	read unflushed "prev" and write "slabs_clsid"	inconsistent data
	12	Inter	N	read unflushed "prev" and write "it_flags" or value	inconsistent index
	13	Inter	N	read unflushed "it_flags" and write value	inconsistent data
	14	Inter	N	read unflushed "slabs_clsid" and write "slabs_clsid" of others	inconsistent index

Inter: PM Inter-thread Inconsistency Intra: PM Intra-thread Inconsistency Sync: PM Synchronization Inconsistency

The Time to Identify PM Inter-thread Inconsistency



(1) P-CLHT



(2) FAST-FAIR

- PMRace efficiently triggers reading non-persisted data

Inconsistencies and False Positives

	PM Interleaving Concurrency Bug				PM Execution Context Bug		
	Inter-Cand	Inter	Filtered FP	Unique Bugs	Sync	Filtered FP	Unique Bugs
P-CLHT	35	10	0	1	4	3	1
clevel hashing	6	2	0	0	0	0	0
CCEH	15	0	0	0	1	0	1
FAST-FAIR	179	69	3	1	0	0	0
memcached-pmem	266	79	62	6	0	0	0
Total	501	160	65	8	5	3	2

Inter-Cand: PM Inter-thread Inconsistency Candidate
Filtered FP: Filtered false positives

Inconsistencies and False Positives

	PM Interleaving Concurrency Bug				PM Execution Context Bug		
	Inter-Cand	Inter	Filtered FP	Unique Bugs	Sync	Filtered FP	Unique Bugs
P-CLHT	35	10	0	1	4	3	1
clevel hashing	6	2	0	0	0	0	0
CCEH	15	0	0	0	1	0	1
FAST-FAIR	179	69	3	1	0	0	0
memcached-pmem	266	79	62	6	0	0	0
Total	501 →	160	65	8	5	3	2

Inter-Cand: PM Inter-thread Inconsistency Candidate
Filtered FP: Filtered false positives

- Durable side effects refine inconsistencies

Inconsistencies and False Positives

	PM Interleaving Concurrency Bug				PM Execution Context Bug		
	Inter-Cand	Inter	Filtered FP	Unique Bugs	Sync	Filtered FP	Unique Bugs
P-CLHT	35	10	0	1	4	3	1
clevel hashing	6	2	0	0	0	0	0
CCEH	15	0	0	0	1	0	1
FAST-FAIR	179	69	3	1	0	0	0
memcached-pmem	266	79	62	6	0	0	0
Total	501 →	160	65	8	5	3	2

Inter-Cand: PM Inter-thread Inconsistency Candidate
Filtered FP: Filtered false positives

- Durable side effects refine inconsistencies
- Post-failure validation reduces false positives

Inconsistencies and False Positives

	PM Interleaving Concurrency Bug				PM Execution Context Bug		
	Inter-Cand	Inter	Filtered FP	Unique Bugs	Sync	Filtered FP	Unique Bugs
P-CLHT	35	10	0	1	4	3	1
clevel hashing	6	2	0	0	0	0	0
CCEH	15	0	0	0	1	0	1
FAST-FAIR	179	69	3	1	0	0	0
memcached-pmem	266	79	62	6	0	0	0
Total	501 →	160	65	8	5	3	2

Inter-Cand: PM Inter-thread Inconsistency Candidate
Filtered FP: Filtered false positives

- Durable side effects refine inconsistencies
- Post-failure validation reduces false positives
- Limitation: false positives still exist due to lazy recovery mechanisms…

Conclusion

- PM-specific concurrency bugs are unexplored
- We identify two new PM concurrency bug patterns
- **PMRace**: the first tool to detect PM concurrency bugs
  - PM-aware coverage-guided fuzzing to accelerate interleaving exploration
  - Post-failure validation to reduce false positives
- Found 14 bugs in 5 concurrent PM programs
- Open-source at https://github.com/yhuacode/pmrace

Thanks!

Email: chenzy@hust.edu.cn

Homepage: https://chenzhangyu.github.io