Strand Persistency

Vaibhav Gogte, William Wang[$], Stephan Diestelhorst[$],
Peter M. Chen, Satish Narayanasamy, Thomas F. Wenisch

UNIVERSITY OF MICHIGAN

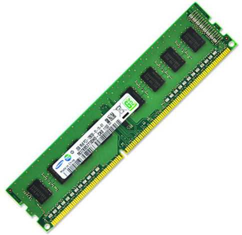
arm[$]

NVMW

03/12/2019

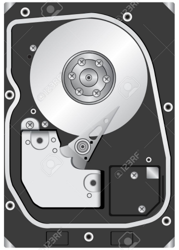
Promise of persistent memory (PM)

Performance



+

Density



+

Non-volatility

Promise of persistent memory (PM)

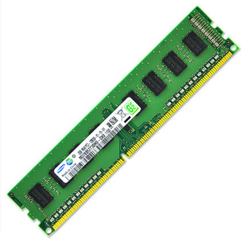
Performance

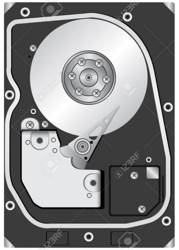
+

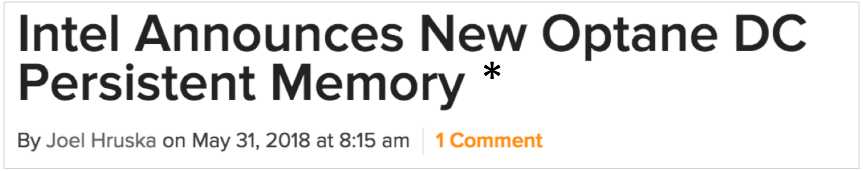
Density

+

Non-volatility







Intel Announces New Optane DC Persistent Memory *

By Joel Hruska on May 31, 2018 at 8:15 am | 1 Comment

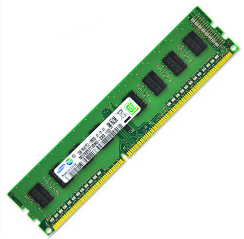
“Optane DC Persistent Memory will be offered in packages of up to 512GB per stick.”

“… expanding memory per CPU socket to as much as 3TB.”

* Source: www.extremetech.com

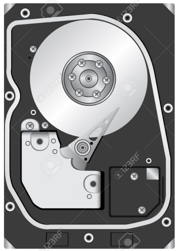
Promise of persistent memory (PM)

Performance



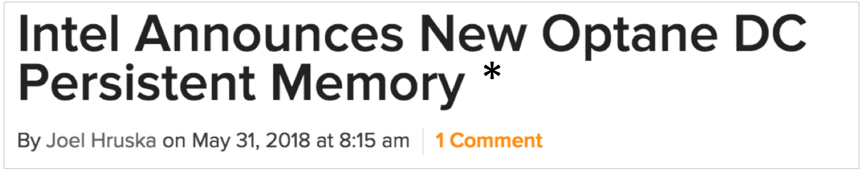
+

Density



+

Non-volatility



Intel Announces New Optane DC Persistent Memory *

By Joel Hruska on May 31, 2018 at 8:15 am | 1 Comment

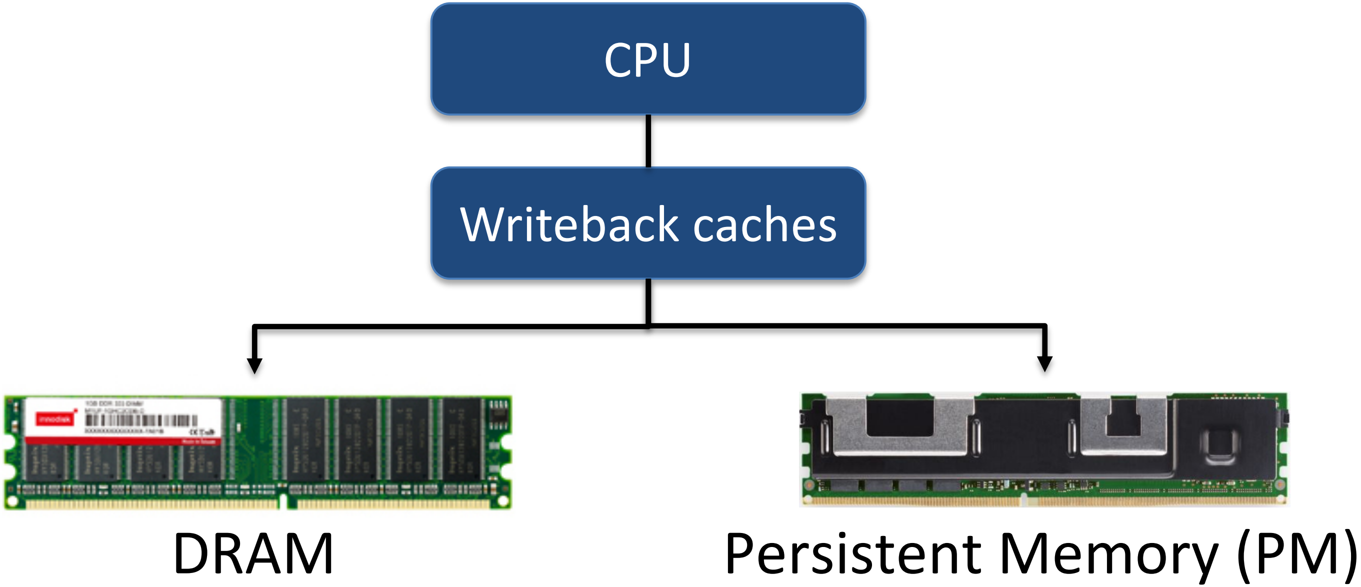
"Optane DC Persistent Memory will be offered in packages of up to 512GB per stick."

"… expanding memory per CPU socket to as much as 3TB."

* Source: www.extremetech.com

Byte-addressable, load-store interface to durable storage

Persistent memory system



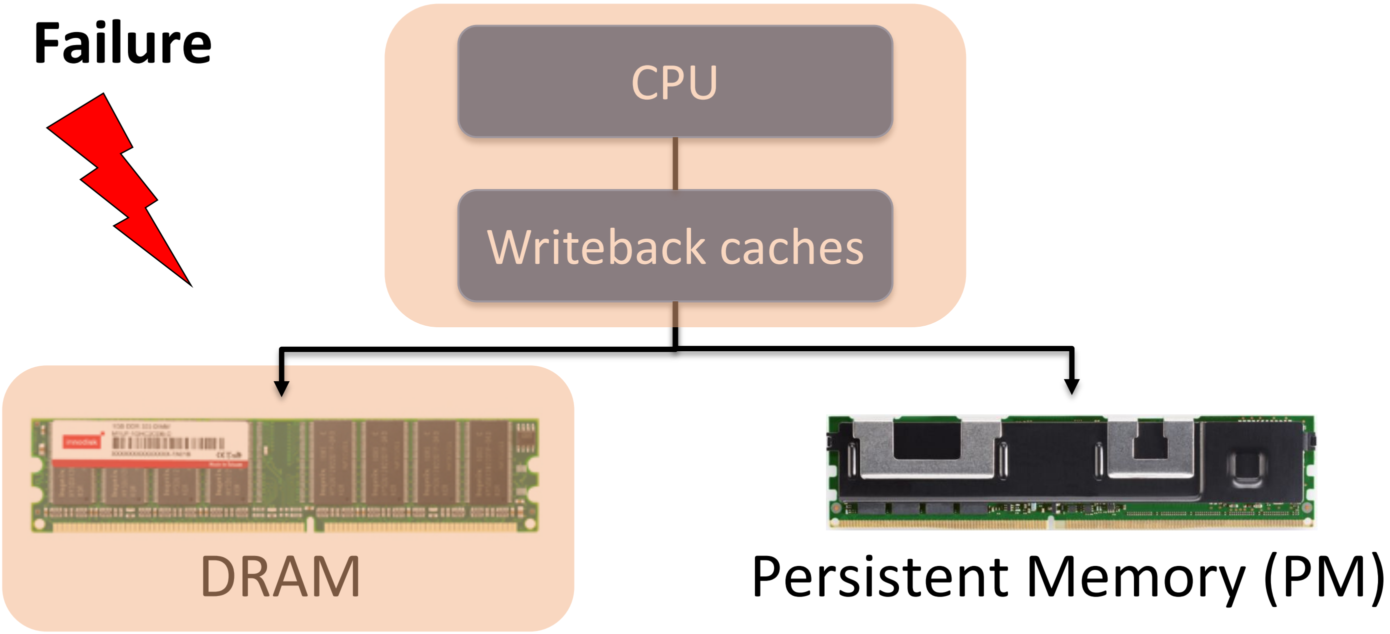
CPU

Writeback caches

DRAM

Persistent Memory (PM)

Persistent memory system



Failure

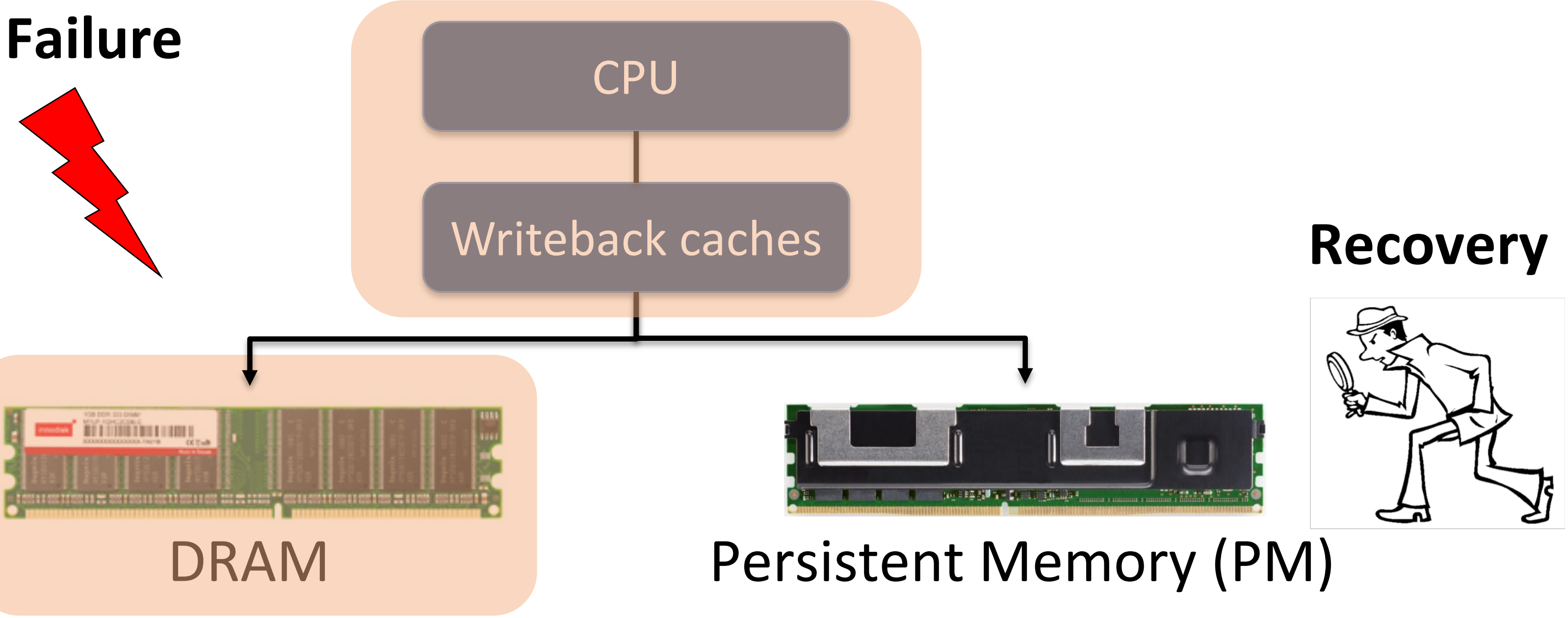
CPU

Writeback caches

DRAM

Persistent Memory (PM)

Persistent memory system



Failure

CPU

Writeback caches

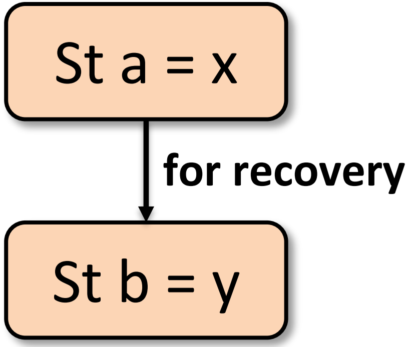
Recovery

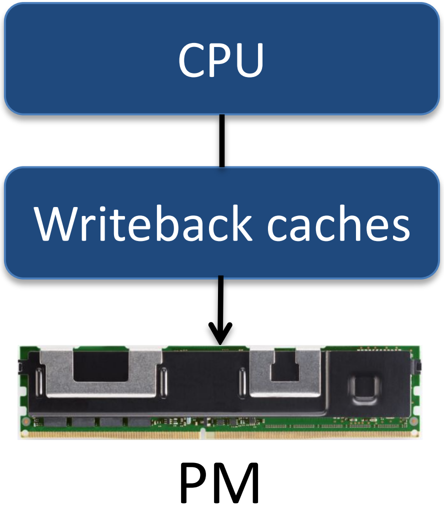
DRAM

Persistent Memory (PM)

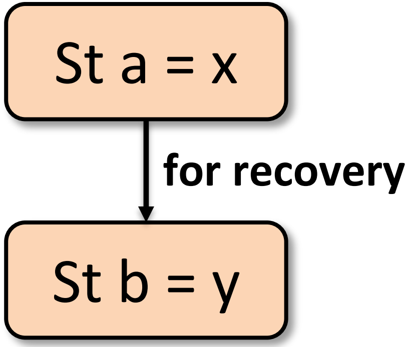
Recovery can inspect PM data-structures to restore system to a consistent state

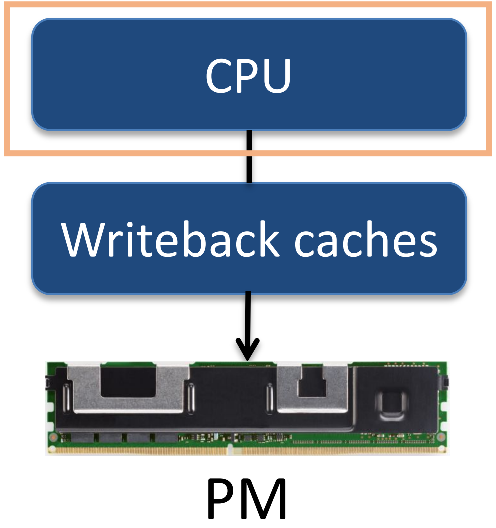
Recovery requires PM access ordering





Recovery requires PM access ordering





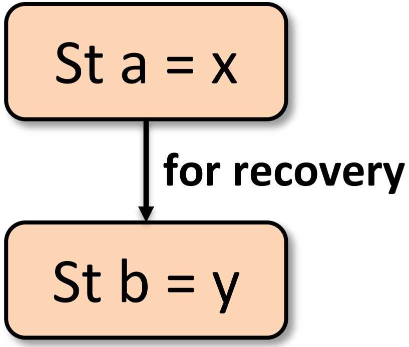
Consistency model

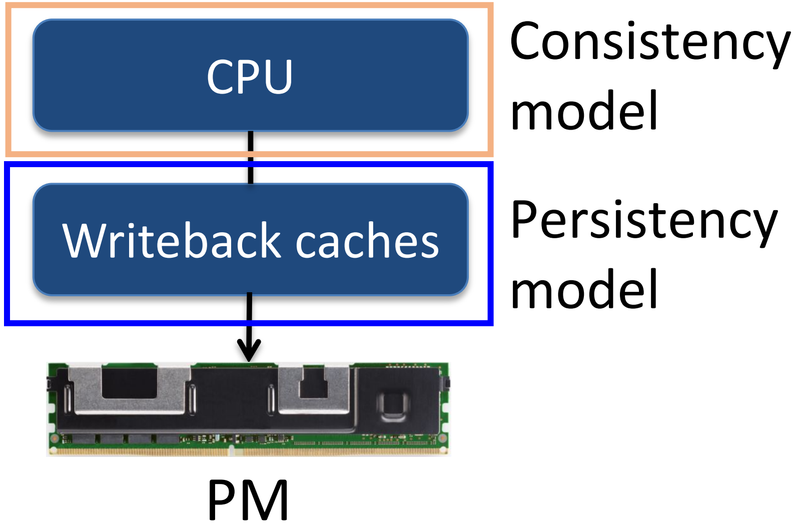
Intel x86 primitives

St a = x

St b = y

Recovery requires PM access ordering





Intel x86 primitives

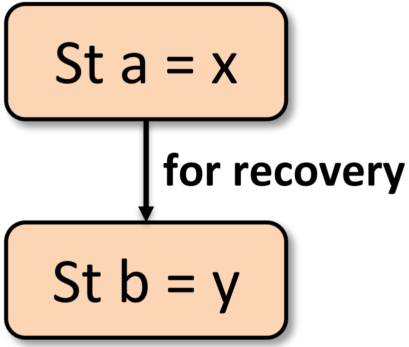
St a = x

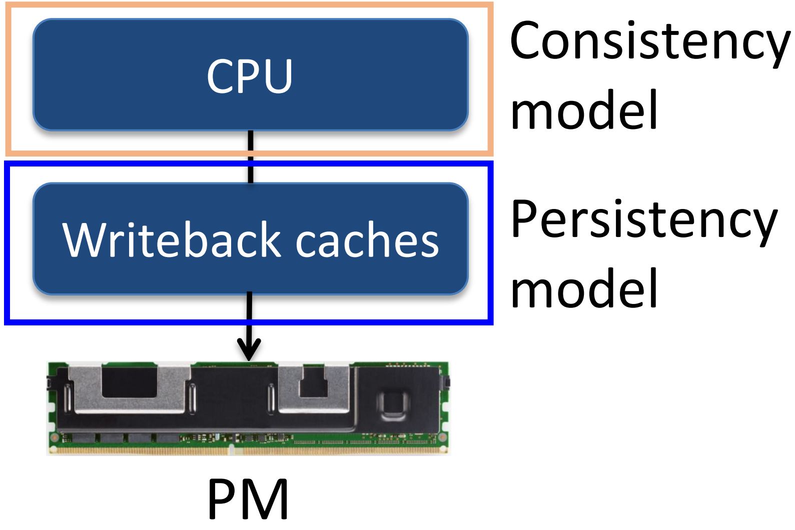
CLWB(a)

St b = y

CLWB(b)

Recovery requires PM access ordering





Intel x86 primitives

St a = x

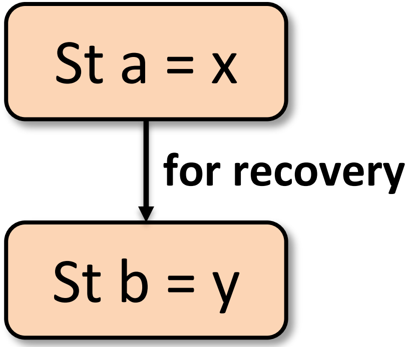
CLWB(a)

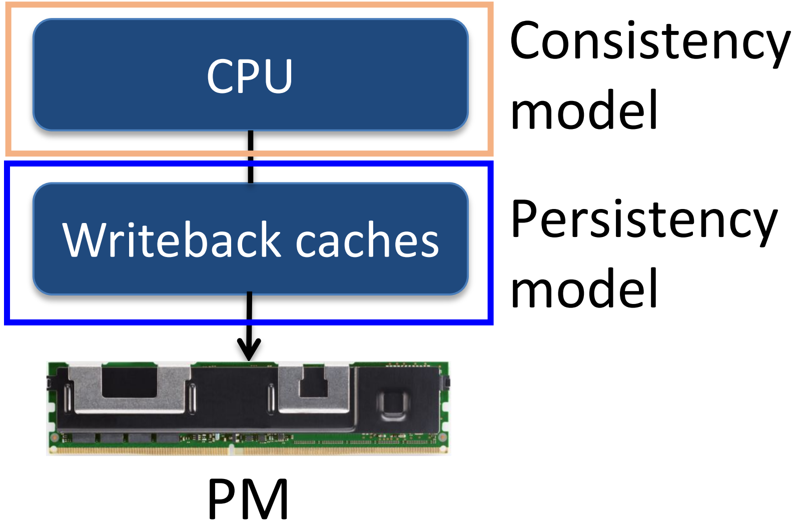
SFENCE

St b = y

CLWB(b)

Recovery requires PM access ordering





Intel x86 primitives

St a = x

CLWB(a)

SFENCE

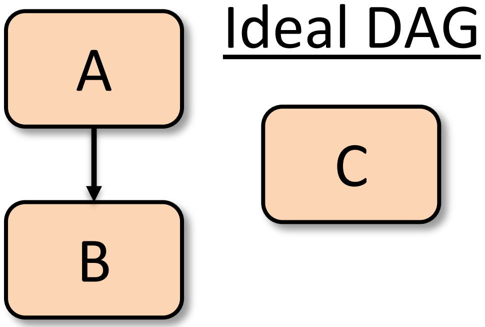
St b = y

CLWB(b)

Hardware systems provide primitives to express *persist* order to PM

Hardware imposes overly strict constraints

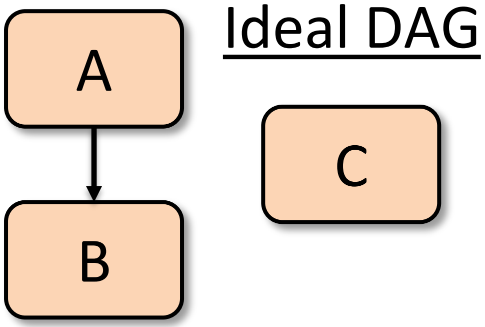
St A = 1; CLWB (A)
St B = 2; CLWB (B)
St C = 3; CLWB (C)

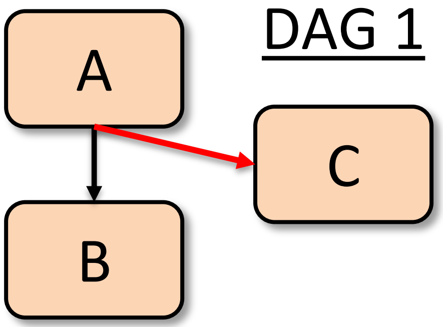


Hardware imposes overly strict constraints

St A = 1; CLWB (A)
St B = 2; CLWB (B)
St C = 3; CLWB (C)

St A = 1; CLWB (A)
SFENCE
St B = 2; CLWB (B)
St C = 3; CLWB (C)



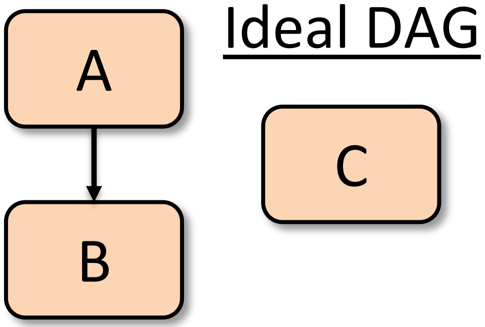


Hardware imposes overly strict constraints

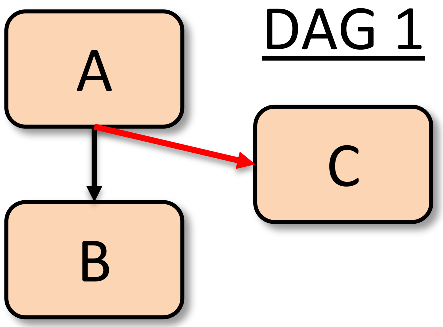
St A = 1; CLWB (A)
St B = 2; CLWB (B)
St C = 3; CLWB (C)

St A = 1; CLWB (A)
SFENCE
St B = 2; CLWB (B)
St C = 3; CLWB (C)

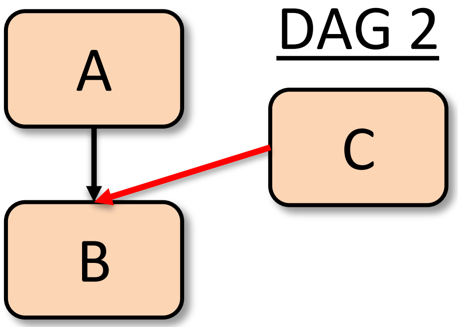
St A = 1 ; CLWB (A)
St C = 3; CLWB (C)
SFENCE
St B = 2; CLWB (B)



Ideal DAG



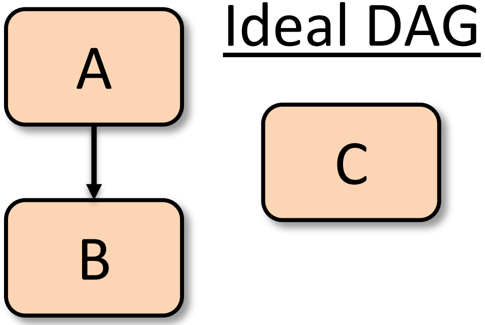
DAG 1



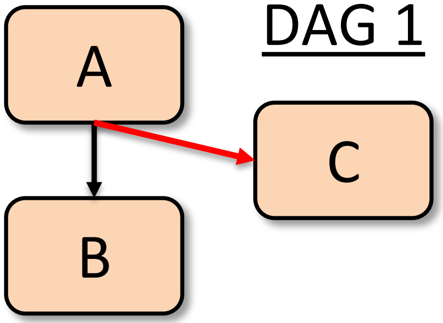
DAG 2

Hardware imposes overly strict constraints

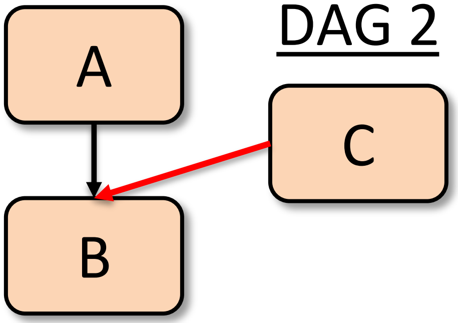
St A = 1; CLWB (A)
St B = 2; CLWB (B)
St C = 3; CLWB (C)


Ideal DAG

St A = 1; CLWB (A)
SFENCE
St B = 2; CLWB (B)
St C = 3; CLWB (C)


DAG 1

St A = 1 ; CLWB (A)
St C = 3; CLWB (C)
SFENCE
St B = 2; CLWB (B)


DAG 2

Primitives in existing hardware systems overconstrain PM accesses

Contributions

- Employ ***strand persistency*** [Pelley14]
  - Hardware ISA primitives to specify precise ordering constraints
- Comprises two primitives: **PersistBarrier** and **NewStrand**
  - Can encode an arbitrary DAG
- Map language-level persistency models to ISA level primitives
  - Leverage strand persistency to build persistency models efficiently

Contributions

- Employ ***strand persistency*** [Pelley14]
  - Hardware ISA primitives to specify precise ordering constraints
- Comprises two primitives: **PersistBarrier** and **NewStrand**
  - Can encode an arbitrary DAG
- Map language-level persistency models to ISA level primitives
  - Leverage strand persistency to build persistency models efficiently

Strand persistency improves perf. of language persistency models by 21.4% (avg.)

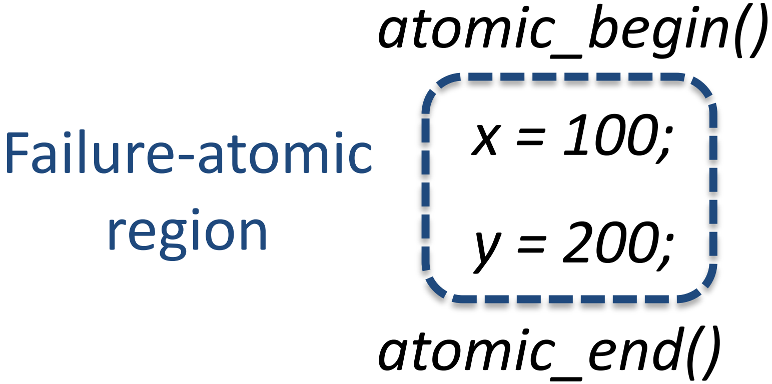
Outline

- Contributions
- Example: Failure atomicity
- Existing hardware primitives
- Strand persistency
- Evaluation

Example: Failure atomicity

<u>Failure-atomicity</u>:

Which group of stores persist atomically?



atomic_begin()

Failure-atomic region

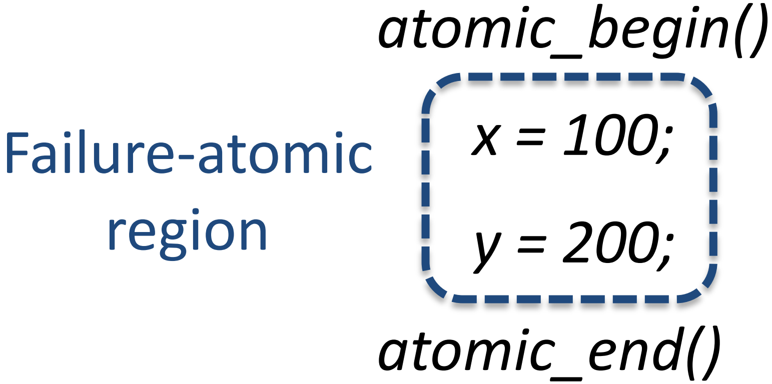
x = 100;

y = 200;

atomic_end()

Example: Failure atomicity

<u>Failure-atomicity</u>:
Which group of stores persist atomically?



atomic_begin()

Failure-atomic region

x = 100;

y = 200;

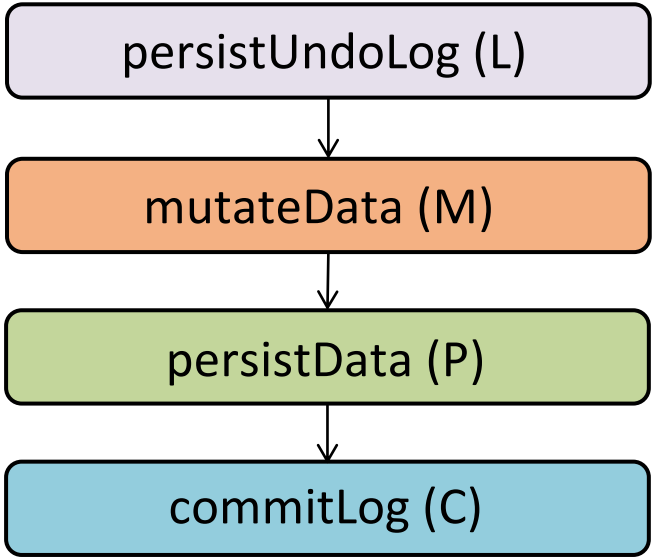
atomic_end()

Failure-atomicity limits state that recovery can observe after failure

Undo-logging for failure atomicity

Init: x = 0; y = 0

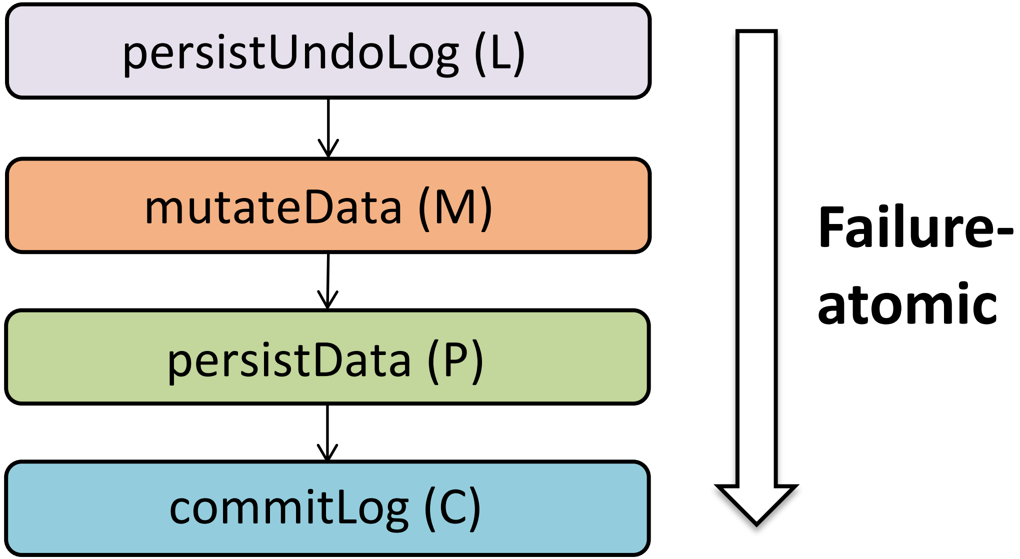
```
atomic_begin()
    x = 1;
    y = 2;
atomic_end()
```



Undo-logging for failure atomicity

Init: x = 0; y = 0

```
atomic_begin()
    x = 1;
    y = 2;
atomic_end()
```

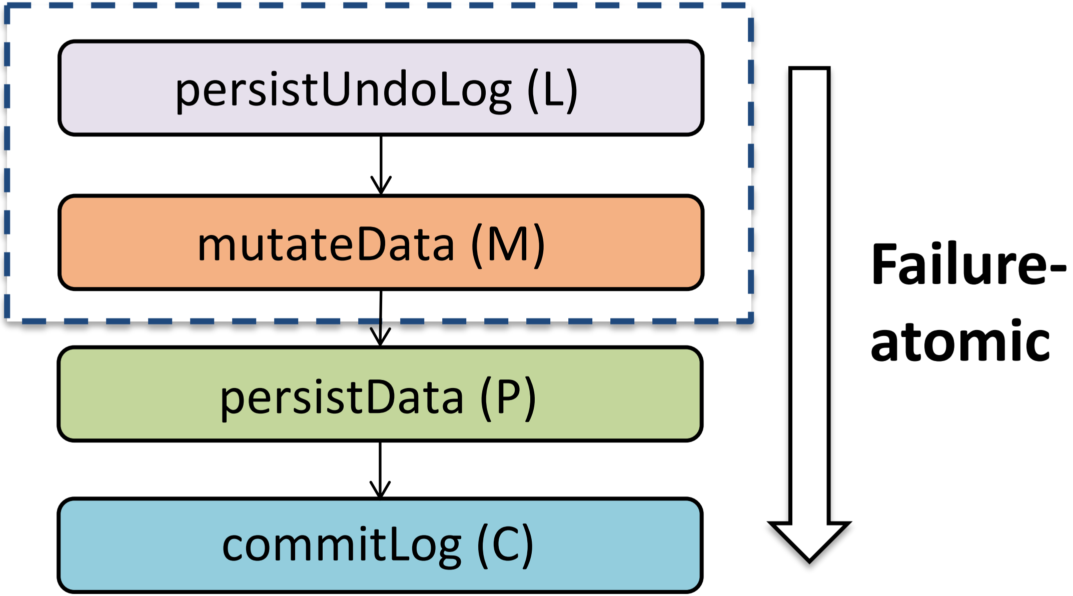


Undo logging steps ordered to ensure failure-atomicity

Undo-logging for failure atomicity

Init: x = 0; y = 0

```
atomic_begin()
    x = 1;
    y = 2;
atomic_end()
```

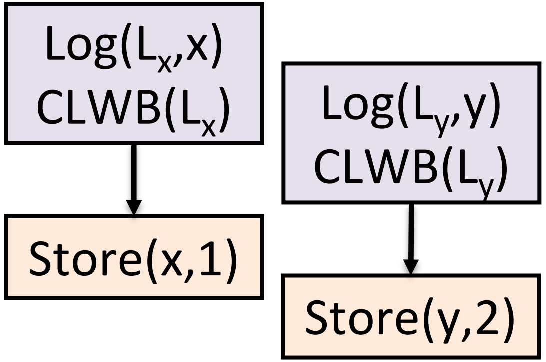


Undo logging steps ordered to ensure failure-atomicity

Hardware imposes stricter constraints

Ideal ordering

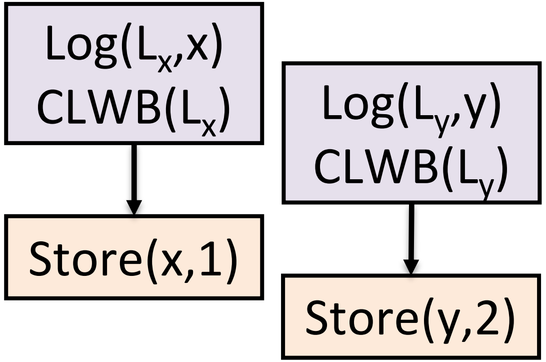
```
atomic_begin()
      x = 1;
      y = 2;
atomic_end()
```



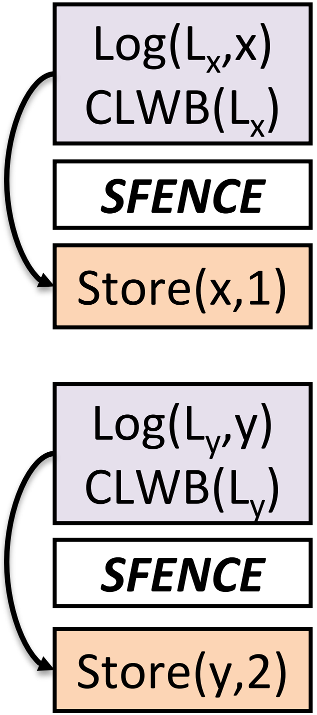
Hardware imposes stricter constraints

```
atomic_begin()
      x = 1;
      y = 2;
atomic_end()
```

Ideal ordering

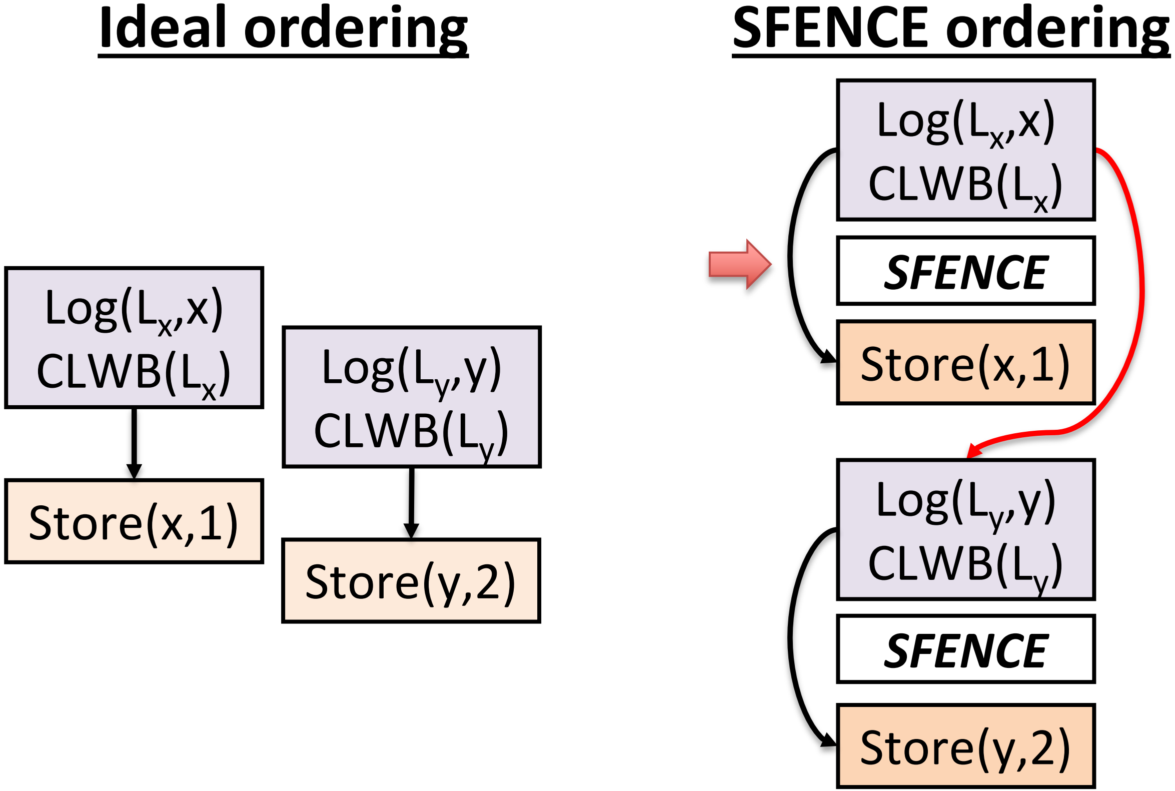


SFENCE ordering



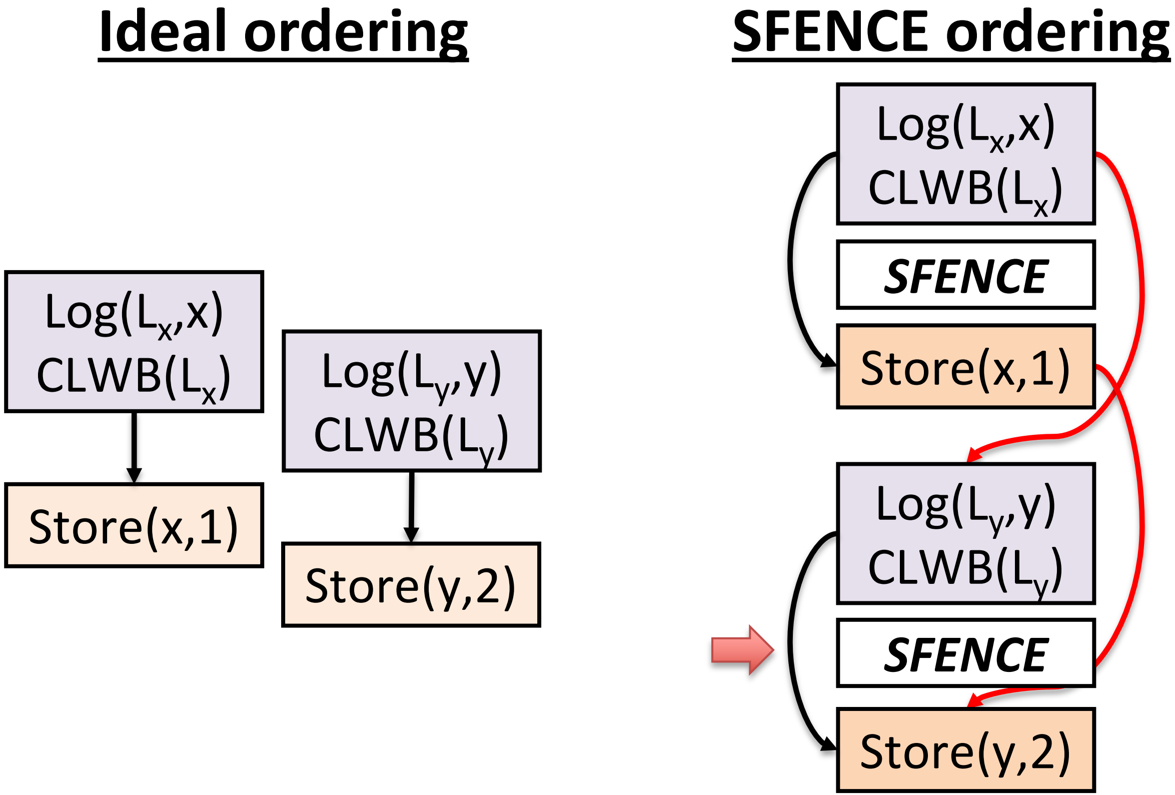
Hardware imposes stricter constraints

```
atomic_begin()
      x = 1;
      y = 2;
atomic_end()
```



Hardware imposes stricter constraints

```
atomic_begin()
    x = 1;
    y = 2;
atomic_end()
```



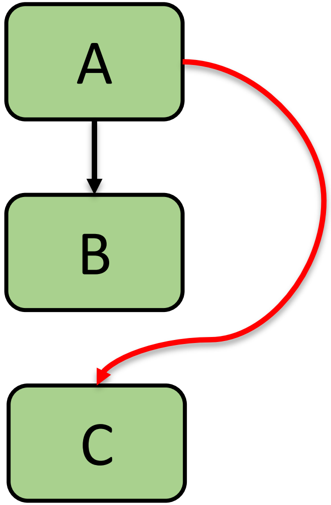
Strand persistency enables persist concurrency

- Provides primitives to express precise persist order

Persist A

Persist B

Persist C



Strand persistency enables persist concurrency

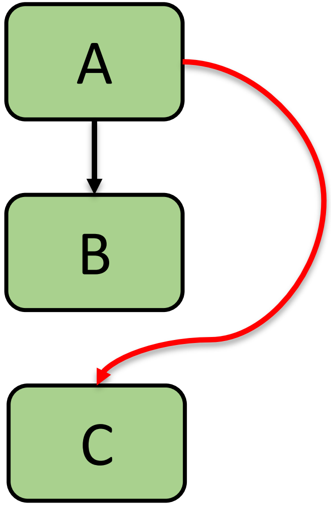
- Provides primitives to express precise persist order

Persist A

Orders persists within a thread ← *PersistBarrier*

Persist B

Persist C



Strand persistency enables persist concurrency

- Provides primitives to express precise persist order

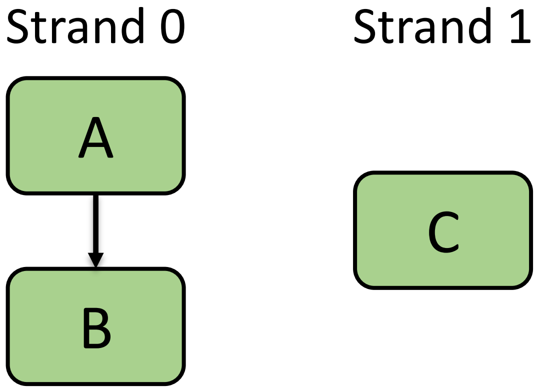
Persist A

Orders persists within a thread ← *PersistBarrier*

Persist B

Initiates new stream of persists ← *NewStrand*

Persist C



Strand persistency enables persist concurrency

- Provides primitives to express precise persist order

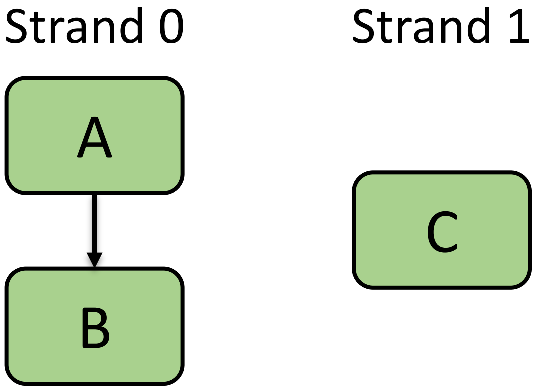
Persist A

Orders persists within a ~~thread~~ **strand** ← *PersistBarrier*

Persist B

Initiates new stream of persists ← *NewStrand*

Persist C



Strand persistency enables persist concurrency

- Provides primitives to express precise persist order

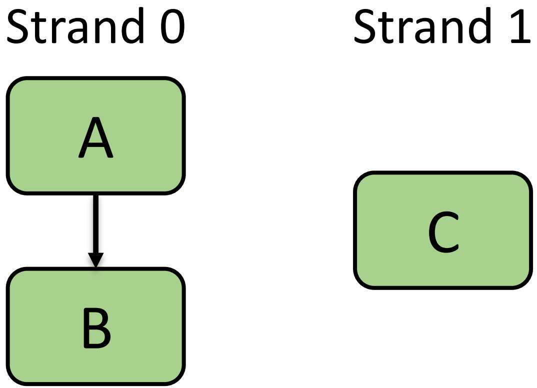
Persist A

Orders persists within a ~~thread~~ strand ← *PersistBarrier*

Persist B

Initiates new stream of persists ← *NewStrand*

Persist C



Persists on different strands can be issued concurrently to PM

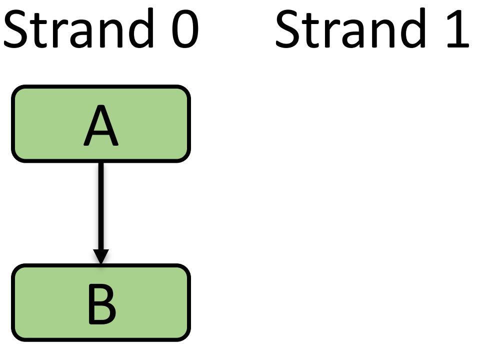
What if ordering is needed across strands?

- Conflicting accesses establish persist order across strands

Persist A

PersistBarrier

Persist B



What if ordering is needed across strands?

- Conflicting accesses establish persist order across strands

Persist A

PersistBarrier

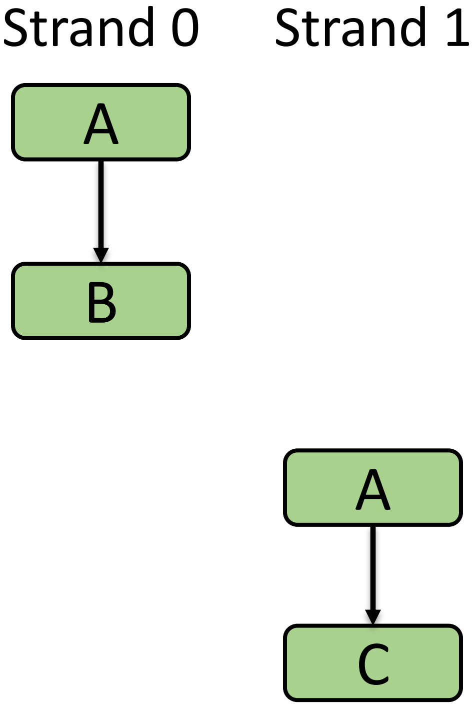
Persist B

NewStrand

Persist A

PersistBarrier

Persist C



What if ordering is needed across strands?

- Conflicting accesses establish persist order across strands

Persist A

PersistBarrier

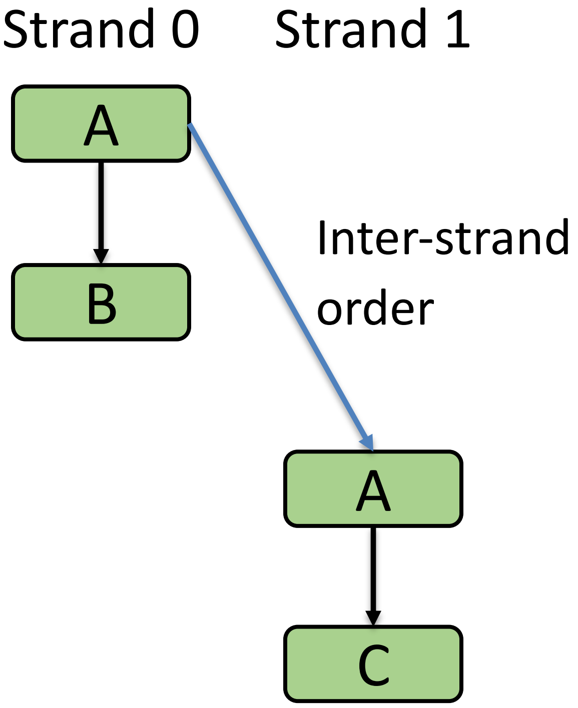
Persist B

NewStrand

Persist A

PersistBarrier

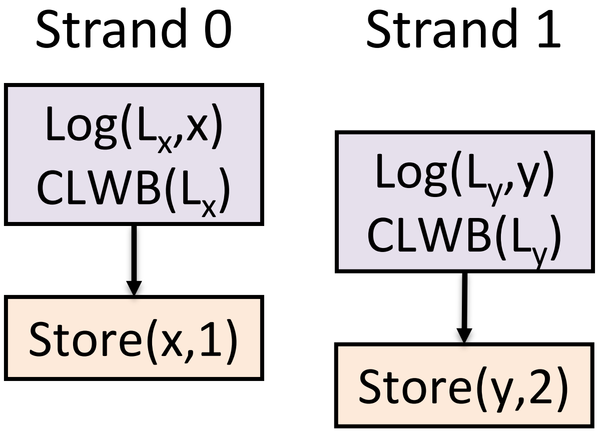
Persist C



Logging using strand persistency

```
atomic_begin()
    x = 1;
    y = 2;
atomic_end()
```

Log(L_x,x)
CLWB(L_x)
PersistBarrier
Store(x,1)
NewStrand
Log(L_y,y)
CLWB(L_y)
PersistBarrier
Store(y,2)



Logging using strand persistency

atomic_begin()

x = 1;

y = 2;

atomic_end()

Log(L_x,x)
CLWB(L_x)

PersistBarrier

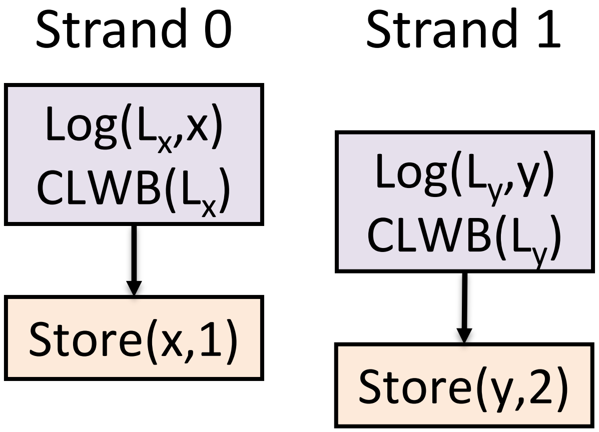
Store(x,1)

NewStrand

Log(L_y,y)
CLWB(L_y)

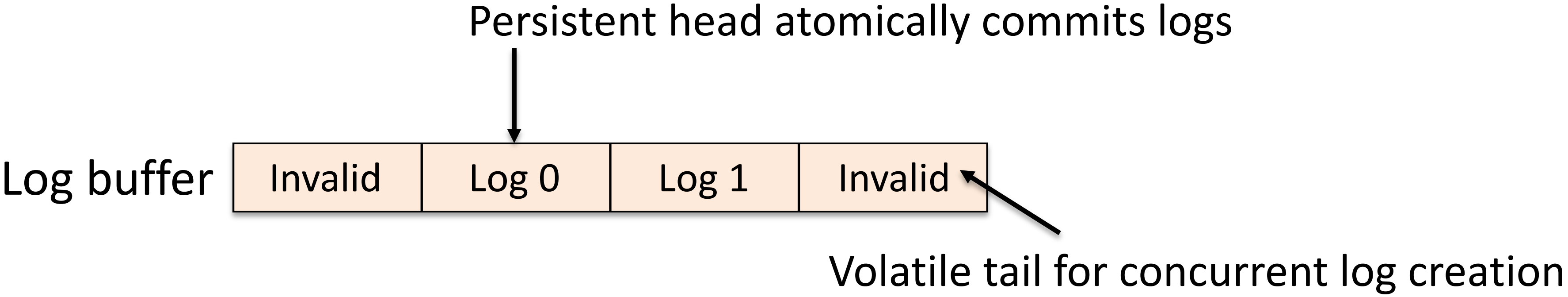
PersistBarrier

Store(y,2)



Need to implement log buffer that can manage concurrent log updates

Log space under strand persistency



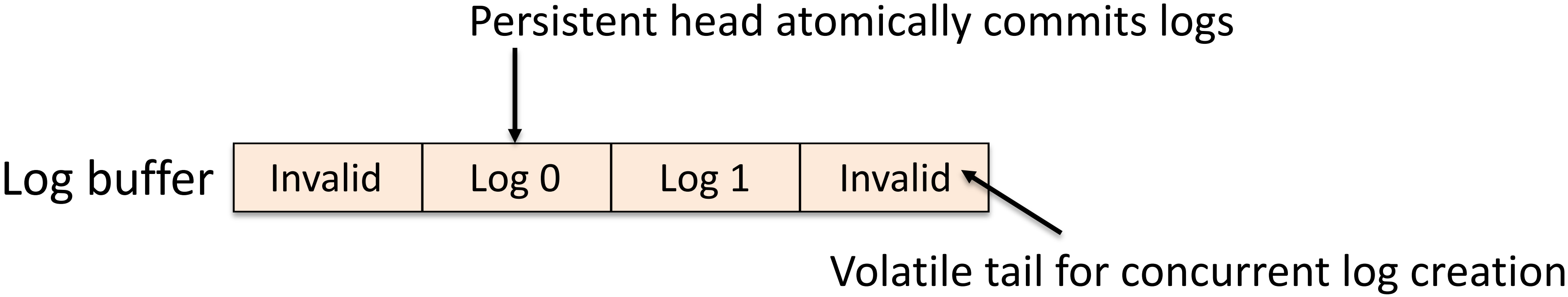
Persistent head atomically commits logs

Log buffer

Invalid	Log 0	Log 1	Invalid

Volatile tail for concurrent log creation

Log space under strand persistency



Persistent head atomically commits logs

Log buffer

Invalid	Log 0	Log 1	Invalid

Volatile tail for concurrent log creation

- Failure exposes log write reorderings
  - Identify valid logs in case of failure
  - Record order of log creation
  - Recovery rolls back partial updates using valid logs

More details in the paper

Language persistency models to ISA primitives

Hardware ISA

ISA primitives: PersistBarrier and NewStrand

Language persistency models to ISA primitives

Compiler — Logging impl. that map to hardware primitives

Hardware ISA — ISA primitives: PersistBarrier and NewStrand

Language persistency models to ISA primitives

High-level languages	Failure atomicity for language-level persistency models
Compiler	Logging impl. that map to hardware primitives
Hardware ISA	ISA primitives: PersistBarrier and NewStrand

Evaluation: Language-level persistency models

```
L1.lock();
    x -= 100;
    y += 100;
    L2.lock();
        a -= 100;
        b += 100;
    L2.unlock();
L1.unlock();
```

ATLAS [Chakrabarti14]

- Failure-atomic outermost critical sections

Evaluation: Language-level persistency models

```
L1.lock();
    x -= 100;
    y += 100;
    L2.lock();
        a -= 100;
        b += 100;
    L2.unlock();
L1.unlock();
```

ATLAS [Chakrabarti14]

- Failure-atomic outermost critical sections

Coupled-SFR [Gogte18]

- Failure-atomic synchronization-free regions

Evaluation: Language-level persistency models

```
L1.lock();
        x -= 100;
        y += 100;
        L2.lock();
                a -= 100;
                b += 100;
        L2.unlock();
L1.unlock();
```

ATLAS [Chakrabarti14]

- Failure-atomic outermost critical sections

Coupled-SFR [Gogte18]

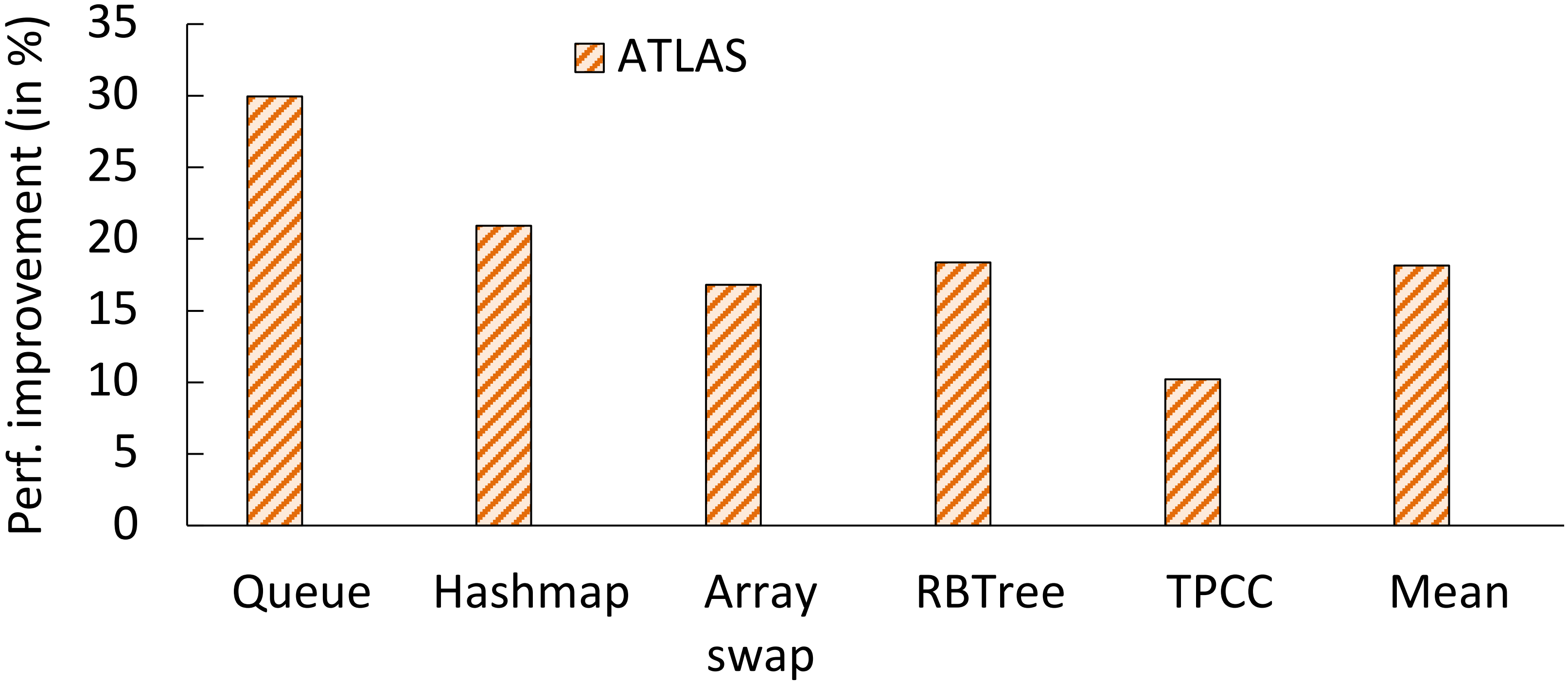
- Failure-atomic synchronization-free regions

Integrate our logging mechanisms with ATLAS and Coupled-SFR

Methodology

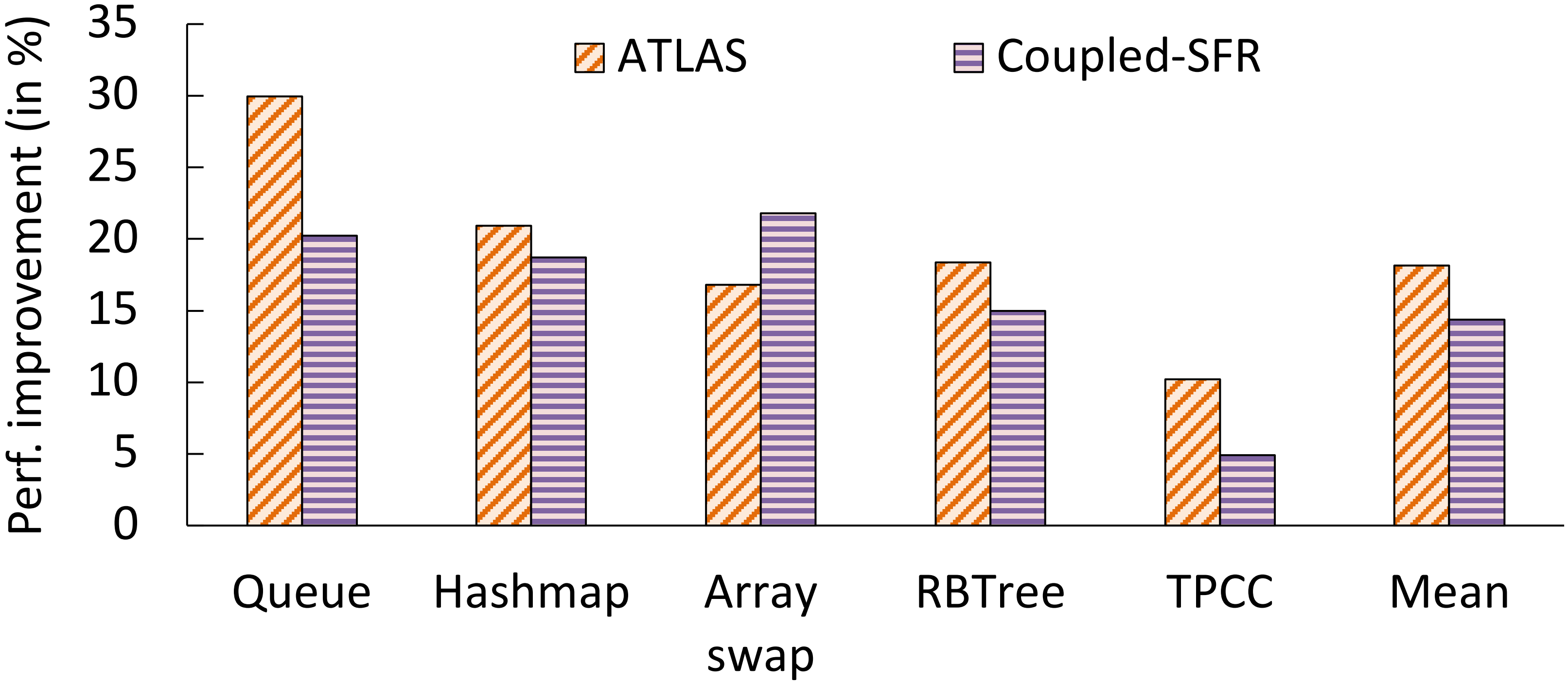
- Gem5 simulator
- Workloads: **write intensive micro-benchmarks**
  - **Queue:** insert/delete entries in a queue
  - **Hashmap**: update values in persistent hash table
  - **Array swaps:** random swaps of array elements
  - **RBTree**: insert/delete entries in red-black tree
  - **TPCC:** new order transaction from TPCC

Performance evaluation



Improves performance of ATLAS by up to 29.9% (18.2% avg.)

Performance evaluation



Improves performance of Coupled-SFR by up to 34.5% (21.4% avg.)

Conclusion

- Strand persistency to precisely order persists
- Two primitives: **PersistBarrier** and **NewStrand**
  - Work together to relax ordering constraints in undo logging
- Evaluation using language-level persistency models
- Performance improvement of up to 34.5%

Strand Persistency

Vaibhav Gogte, William Wang[$], Stephan Diestelhorst[$],
Peter M. Chen, Satish Narayanasamy, Thomas F. Wenisch

UNIVERSITY OF MICHIGAN

arm[$]

NVMW
03/12/2019