MICRO-46, December 11, 2013

Kiln: Closing the Performance Gap Between Systems With and Without Persistence Support

Jishen Zhao	*Penn State Univ.*
Sheng Li	*HP Labs*
Doe Hyun Yoon	*IBM Research*
Yuan Xie	*Penn State Univ./AMD Research*
Norm Jouppi	*Google*

New Design Opportunity with NVRAMs

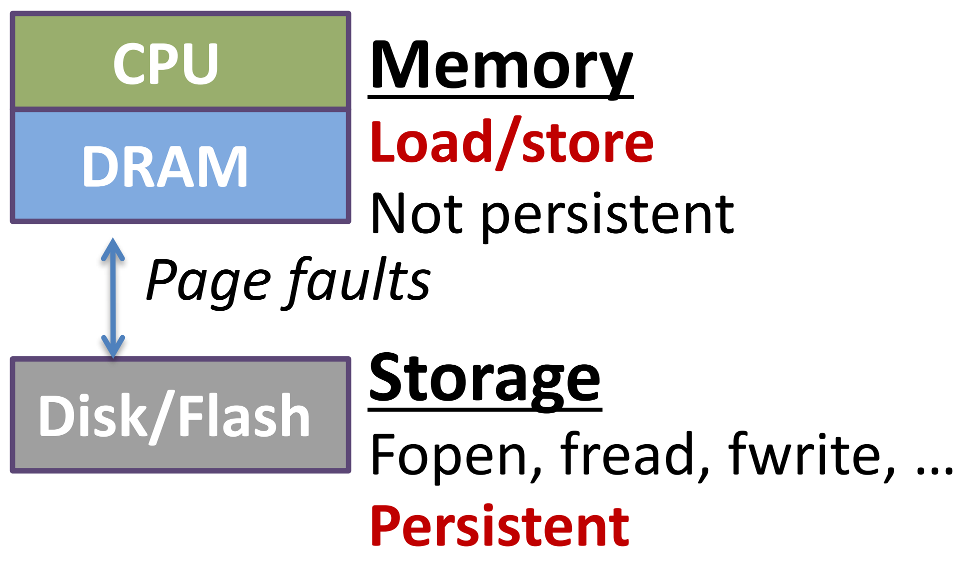
New Design Opportunity with NVRAMs

Memory

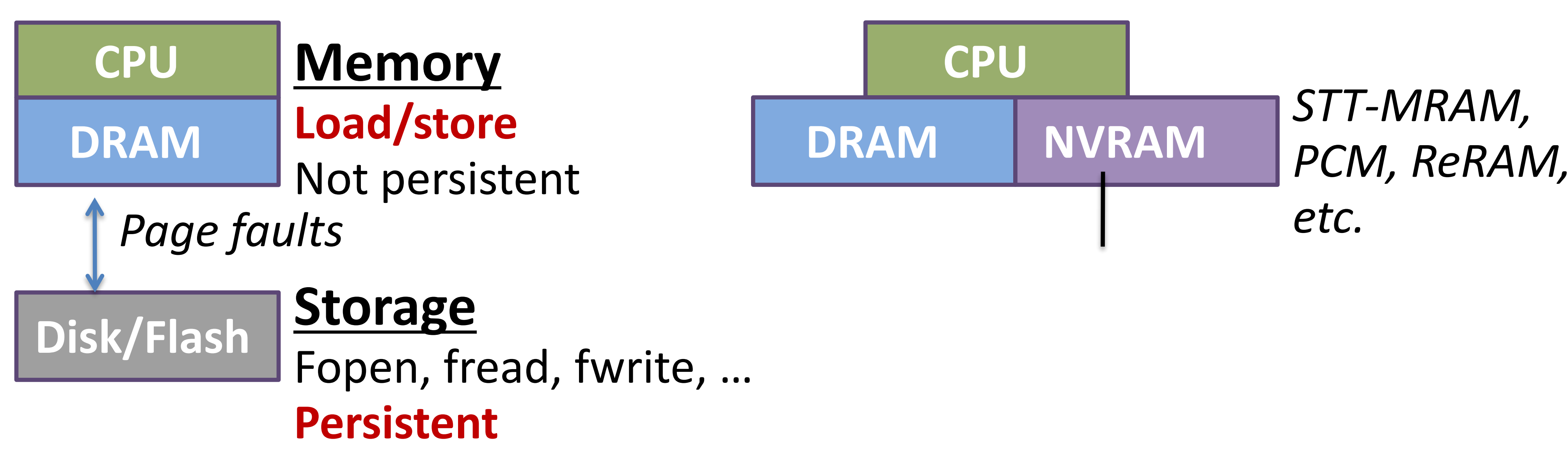
Load/store

Not persistent

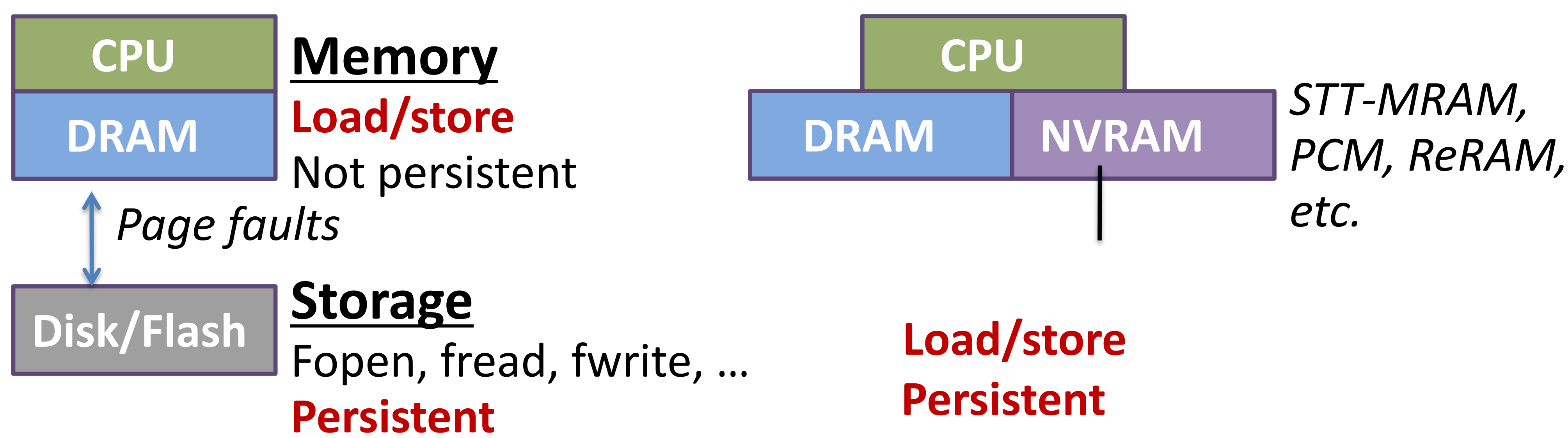
New Design Opportunity with NVRAMs



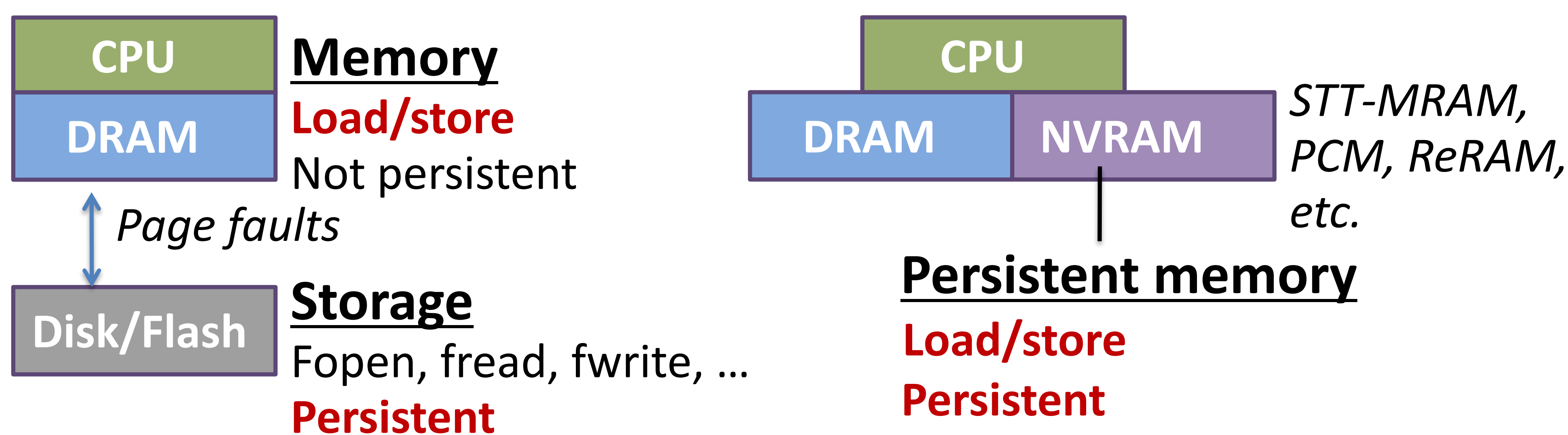
New Design Opportunity with NVRAMs



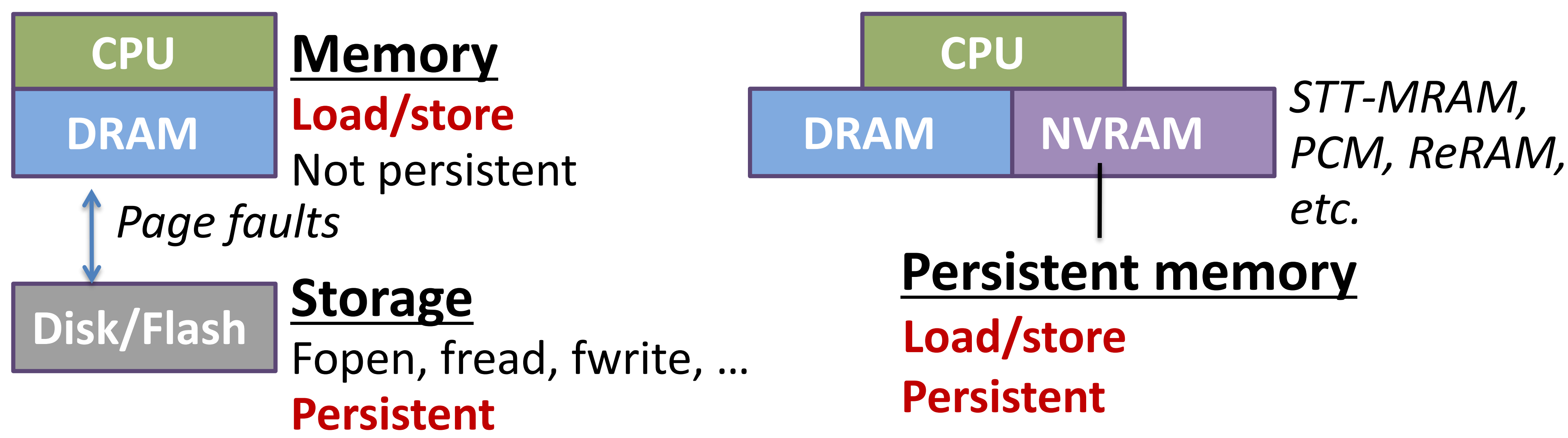
New Design Opportunity with NVRAMs



New Design Opportunity with NVRAMs

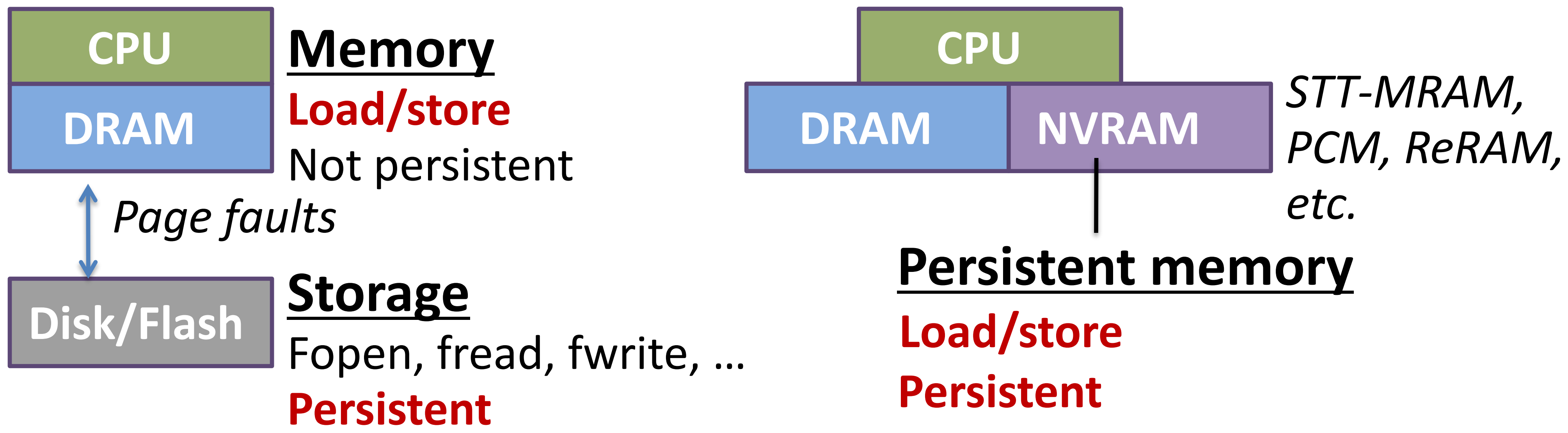


New Design Opportunity with NVRAMs



- Allow in-memory data structures to become permanent immediately

New Design Opportunity with NVRAMs



- Allow in-memory data structures to become permanent immediately
- Demonstrated 32x speedup compared with using storage devices
  [Condit+ SOSP'09, Volos+ ASPLOS'11, Coburn+ ASPLOS'11, Venkataraman+ FAST'11]

Why Can't We Use Existing Systems on Persistent Memory?

Adapted from Katelin Bailey's description at INFLOW'13

Why Can't We Use Existing Systems on Persistent Memory?

- Can't just use an in-memory system – ***No persistence support***
  - In-place updates to nonvolatile media may be interrupted without completion in the event of power loss or system crashes
  - Interrupted writes could leave **partially overwritten data** or **missing references**

Adapted from Katelin Bailey's description at INFLOW'13

Why Can't We Use Existing Systems on Persistent Memory?

- Can't just use an in-memory system – ***No persistence support***
  - In-place updates to nonvolatile media may be interrupted without completion in the event of power loss or system crashes
  - Interrupted writes could leave **partially overwritten data** or **missing references**

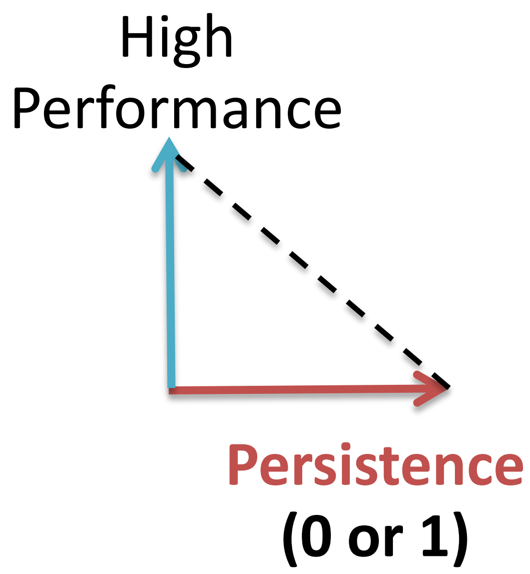
- Cant' just use a storage system – ***Not optimized for memory***
  - **Overhead** from database or file system interfaces, which assume and are optimized for slow, block-addressable devices
  - Not optimized for fast, byte-addressable memory with a load/store interface

Adapted from Katelin Bailey's description at INFLOW'13

Focus of Our Work

Persistence

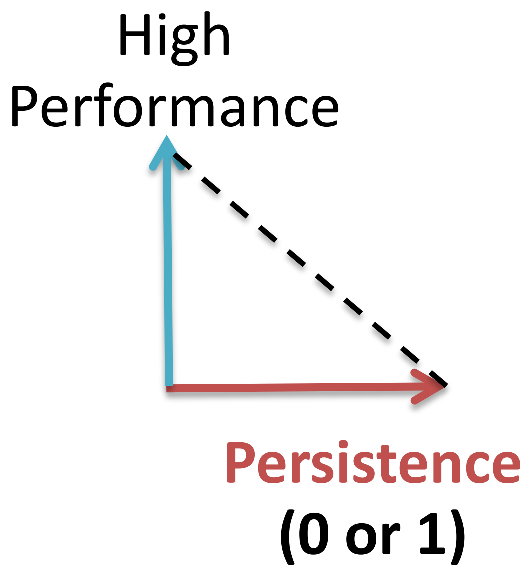
Performance



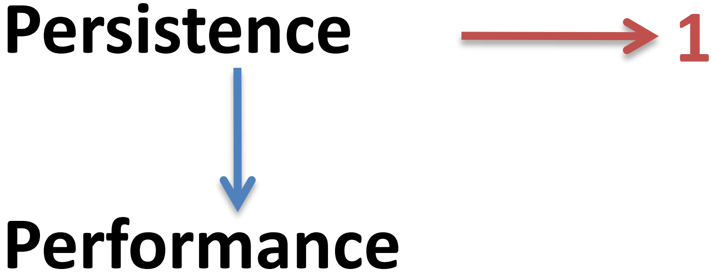
Focus of Our Work

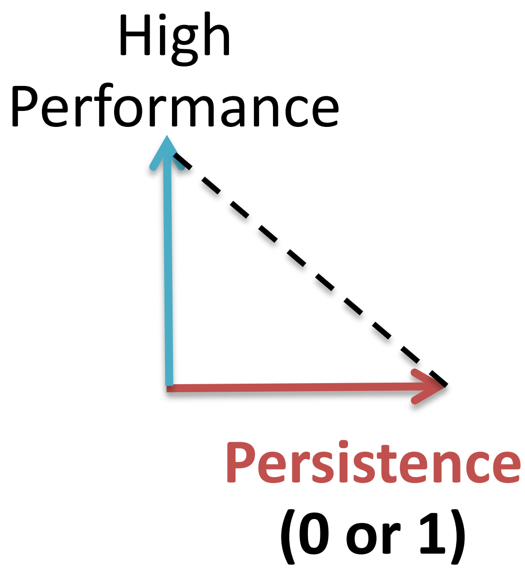
Persistence → **1**

Performance

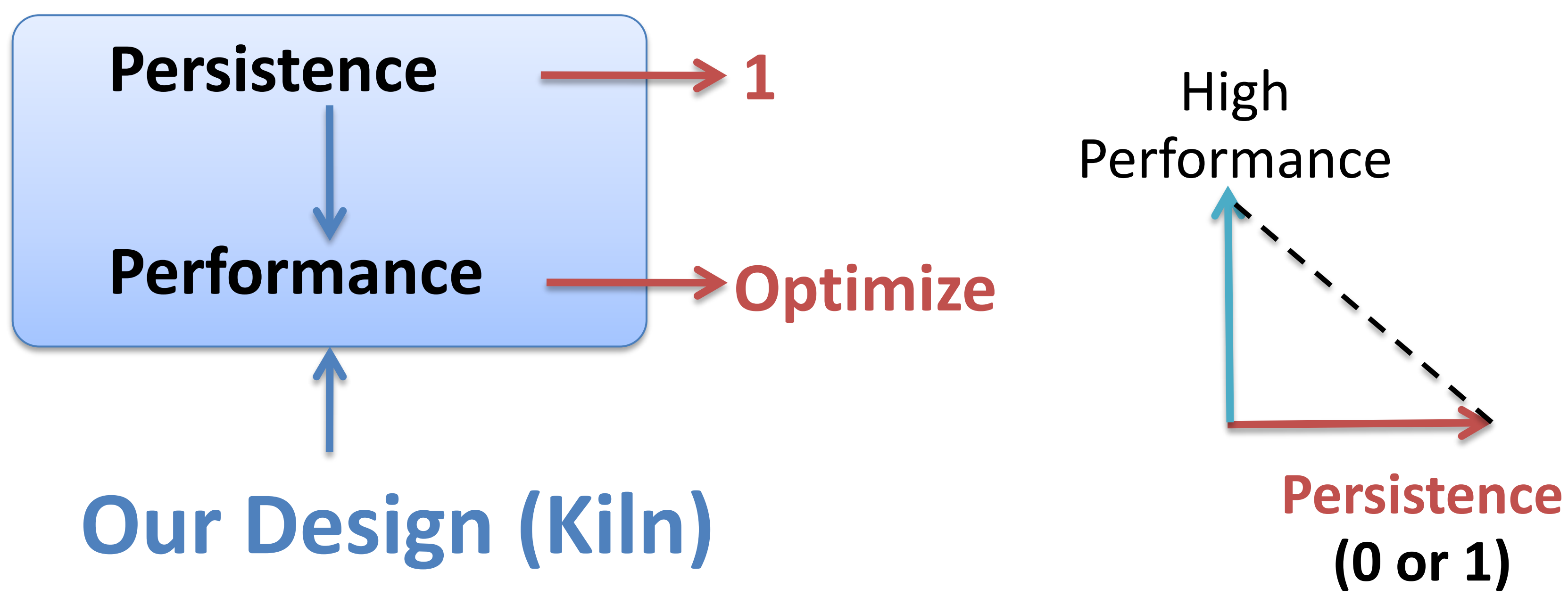


Focus of Our Work

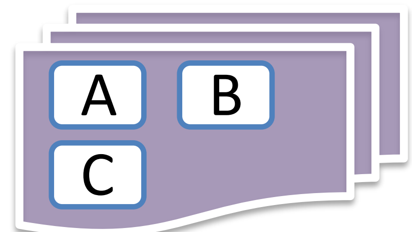




Focus of Our Work

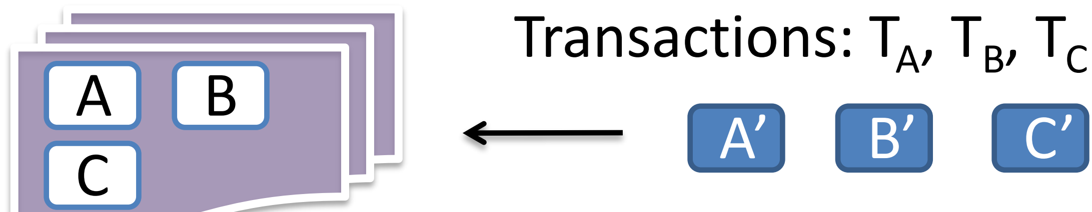


How to Support Persistence in a Memory



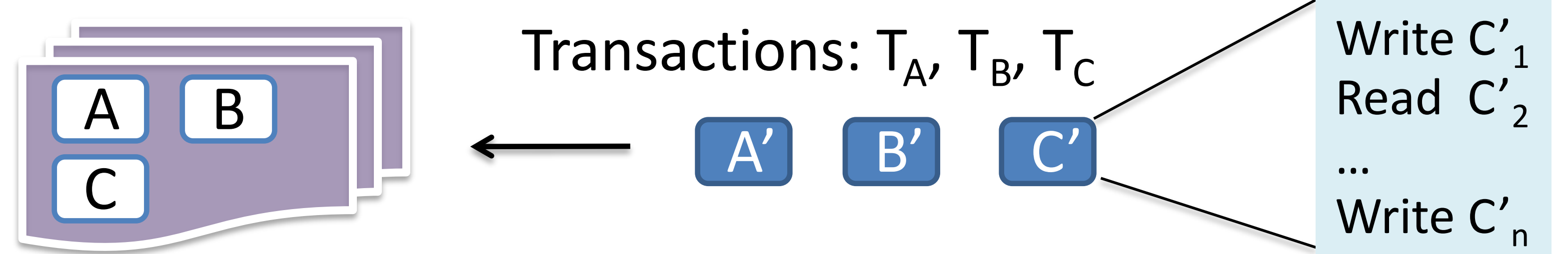
A database or a file system

How to Support Persistence in a Memory

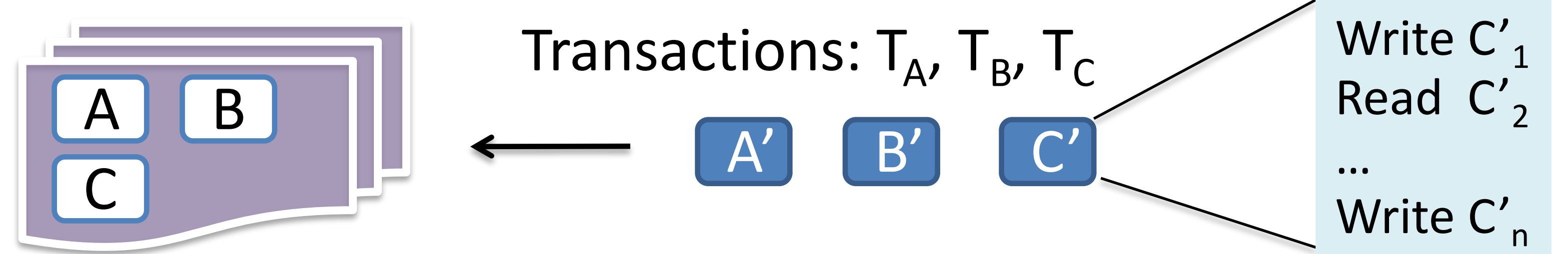


A database or a file system

How to Support Persistence in a Memory



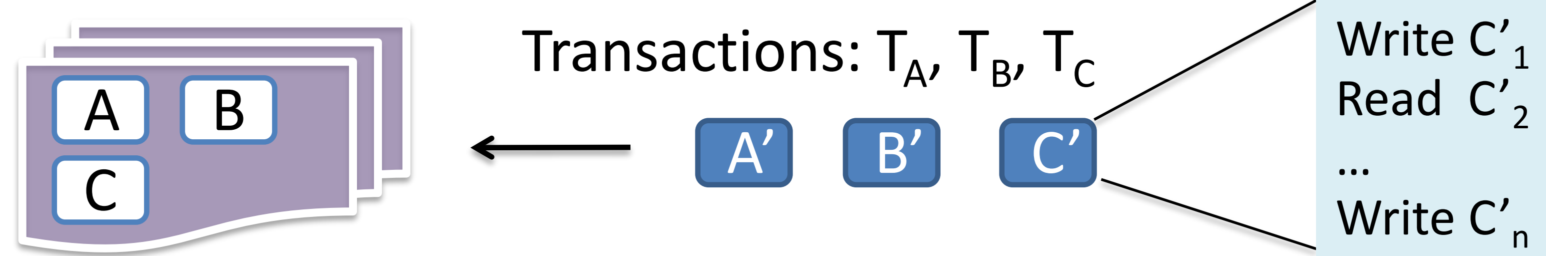
How to Support Persistence in a Memory



A database or a file system

ACID properties -- Guarantee transactions are processed reliably

How to Support Persistence in a Memory

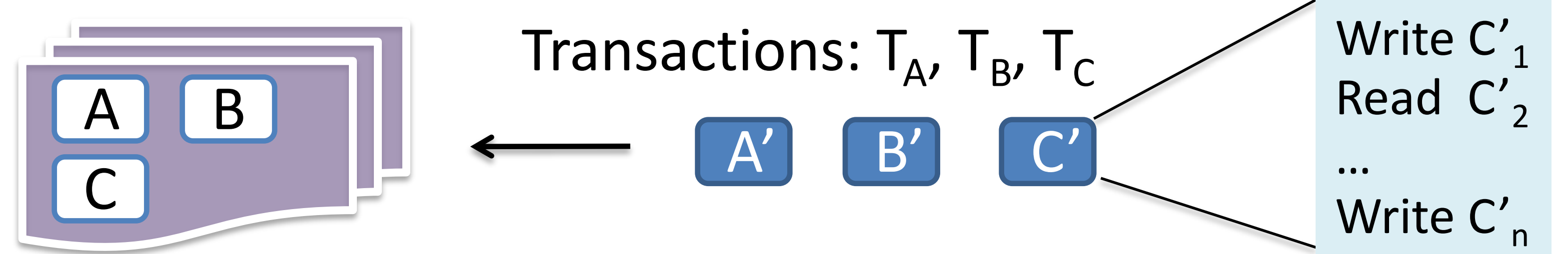


A database or a file system

ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"

How to Support Persistence in a Memory

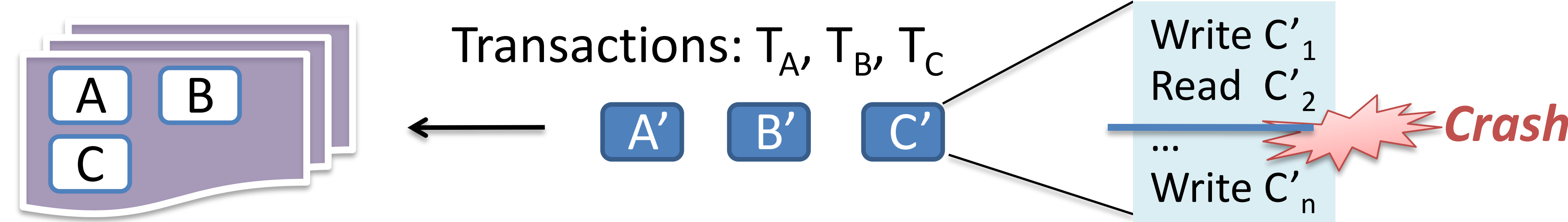


Can contain multiple write requests

ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"

How to Support Persistence in a Memory

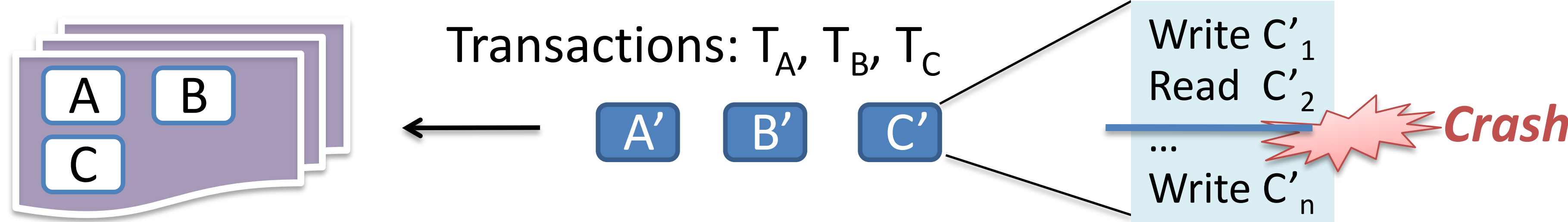


Can contain multiple write requests

ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"

How to Support Persistence in a Memory



A database or a file system

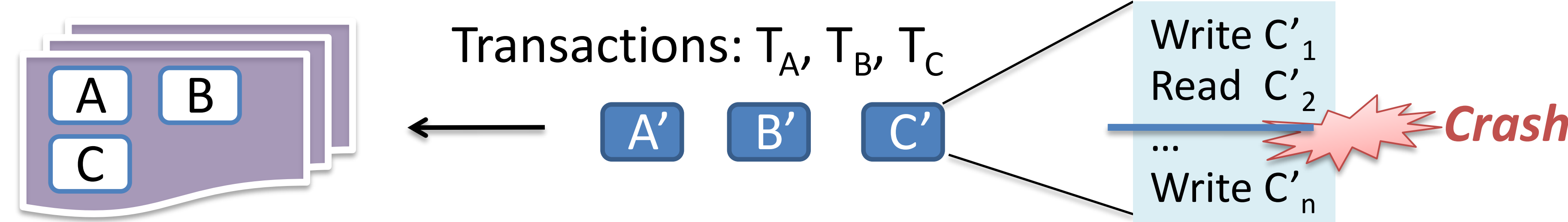
Corrupted by a partially completed transaction!

Can contain multiple write requests

ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is “all or nothing”

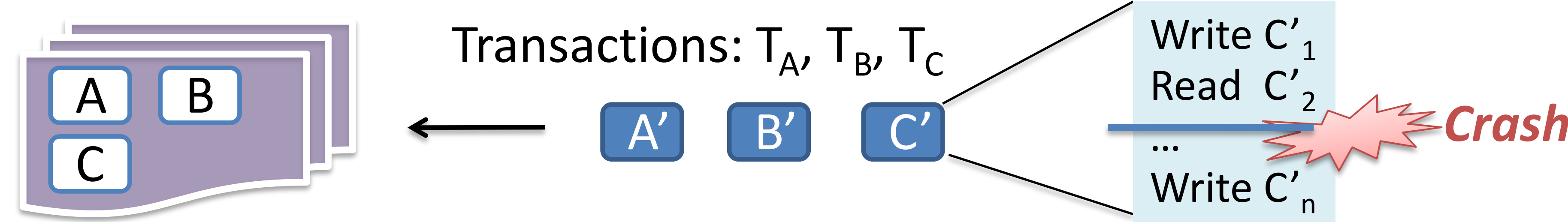
How to Support Persistence in a Memory



ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"
- **C**onsistency: Take data from one consistent state to another

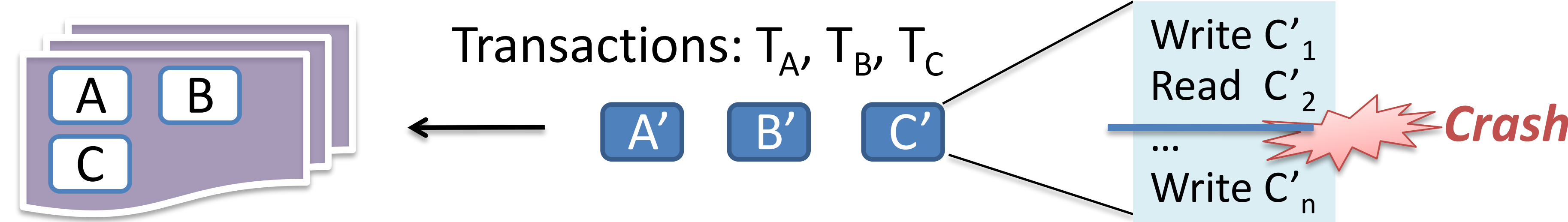
How to Support Persistence in a Memory



ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"
- **C**onsistency: Take data from one consistent state to another
- **I**solation: Concurrent transactions appears to be one after another

How to Support Persistence in a Memory

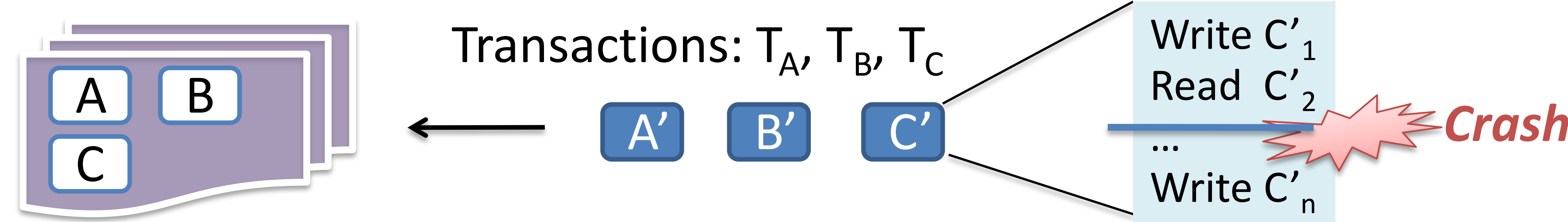


Can contain multiple write requests

ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"
- **C**onsistency: Take data from one consistent state to another
- **I**solation: Concurrent transactions appears to be one after another
- **D**urability: Changes to data will remain across system boots

How to Support Persistence in a Memory

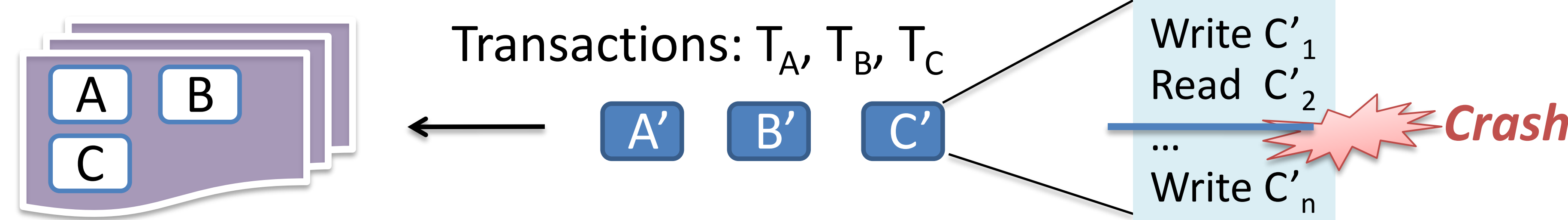


ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"
- **C**onsistency: Take data from one consistent state to another
- **I**solation: Concurrent transactions appears to be one after another
- **D**urability: Changes to data will remain across

Nonvolatility

How to Support Persistence in a Memory



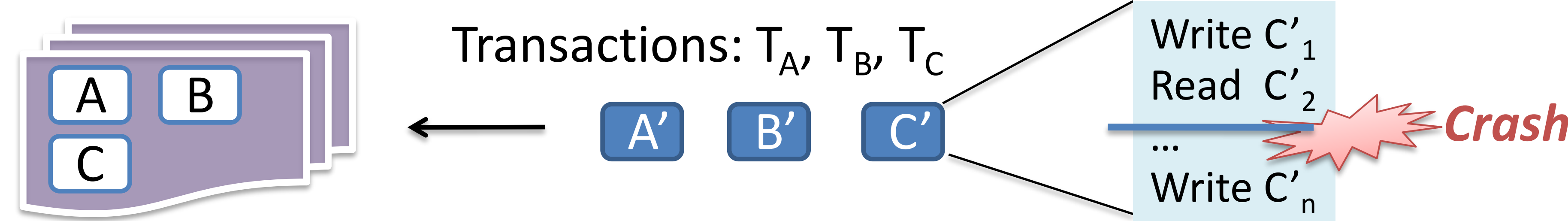
ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"
- **C**onsistency: Take data from one consistent state to another
- **I**solation: Concurrent transactions appears to
- **D**urability: Changes to data will remain across

Concurrency Control

Nonvolatility

How to Support Persistence in a Memory



ACID properties -- Guarantee transactions are processed reliably

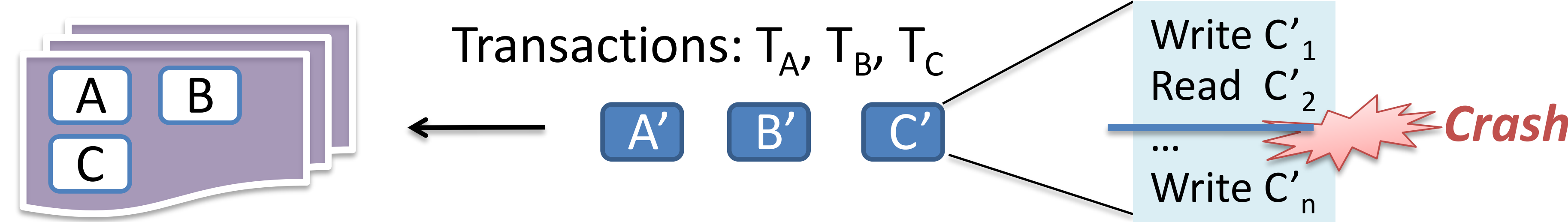
- **A**tomicity: A transaction is "all or nothing"
- **C**onsistency: Take data from one consistent s
- **I**solation: Concurrent transactions appears to
- **D**urability: Changes to data will remain across

Ordering of Writes

Concurrency Control

Nonvolatility

How to Support Persistence in a Memory



ACID properties -- Guarantee transactions are processed reliably

- **A**tomicity: A transaction is "all or nothing"
- **C**onsistency: Take data from one consistent s
- **I**solation: Concurrent transactions appears tc
- **D**urability: Changes to data will remain across

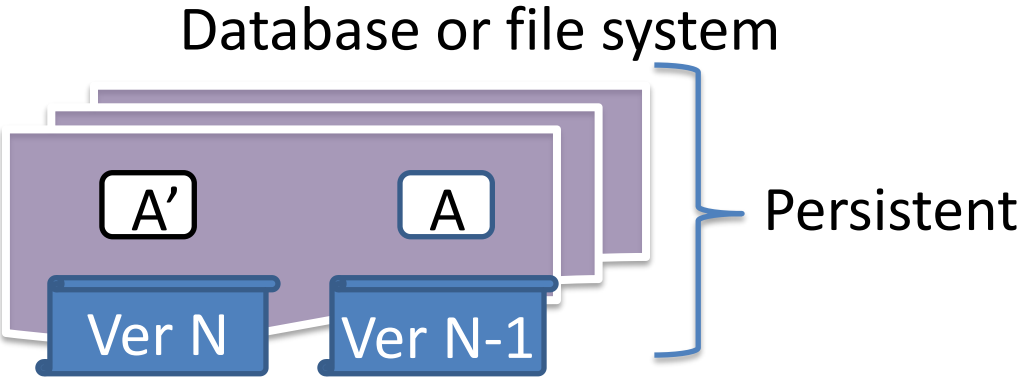
Multiversioning

Ordering of Writes

Concurrency Control

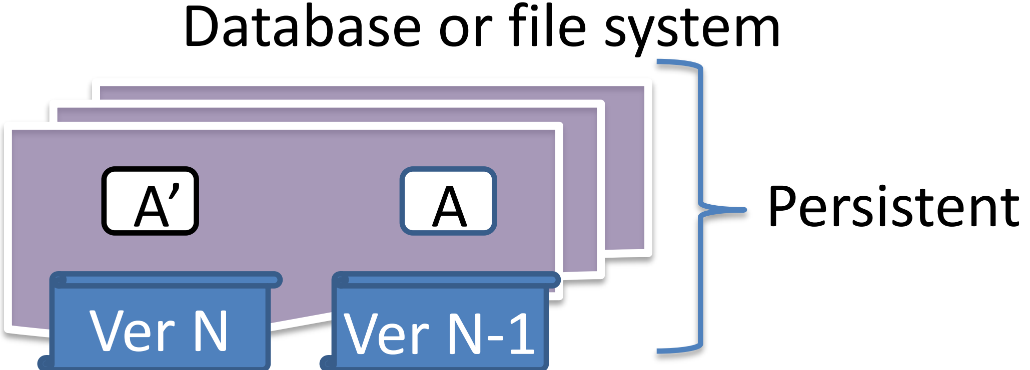
Nonvolatility

Multiversioning



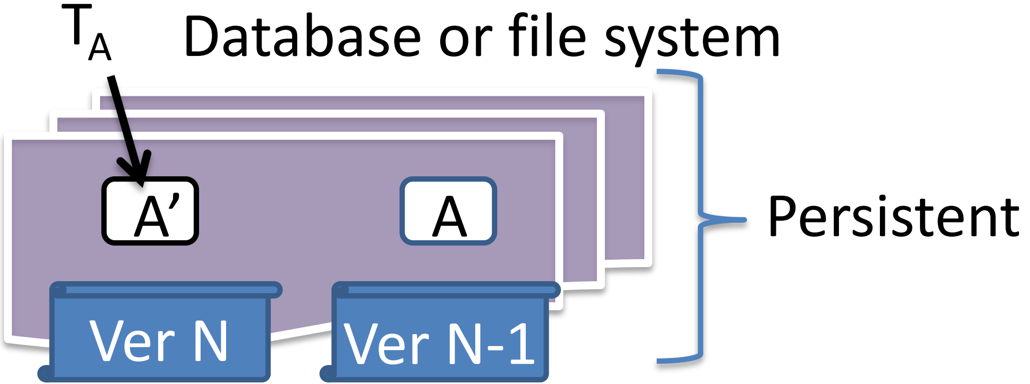
- At least two versions

Multiversioning



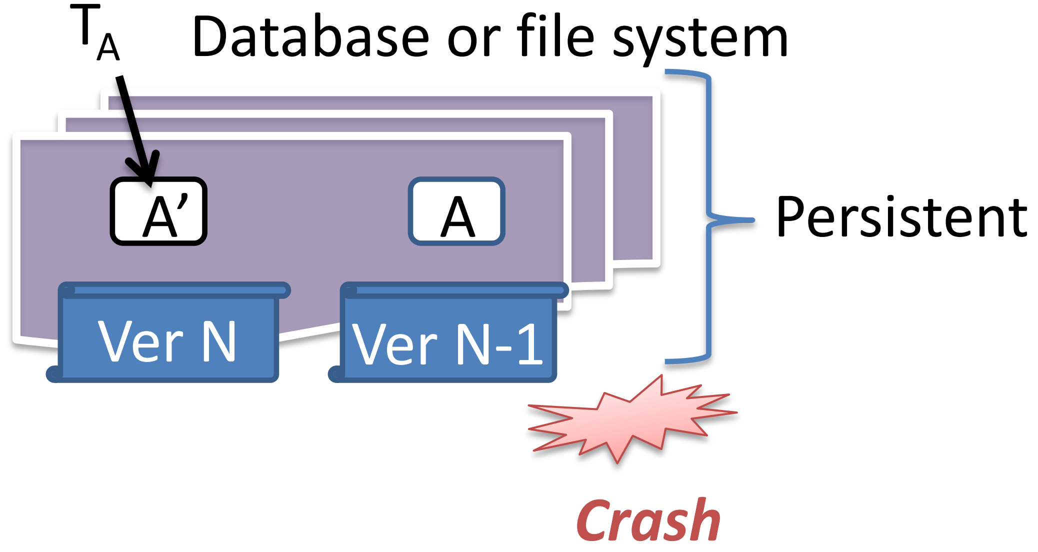
- At least two versions
- Update one version at a time

Multiversioning



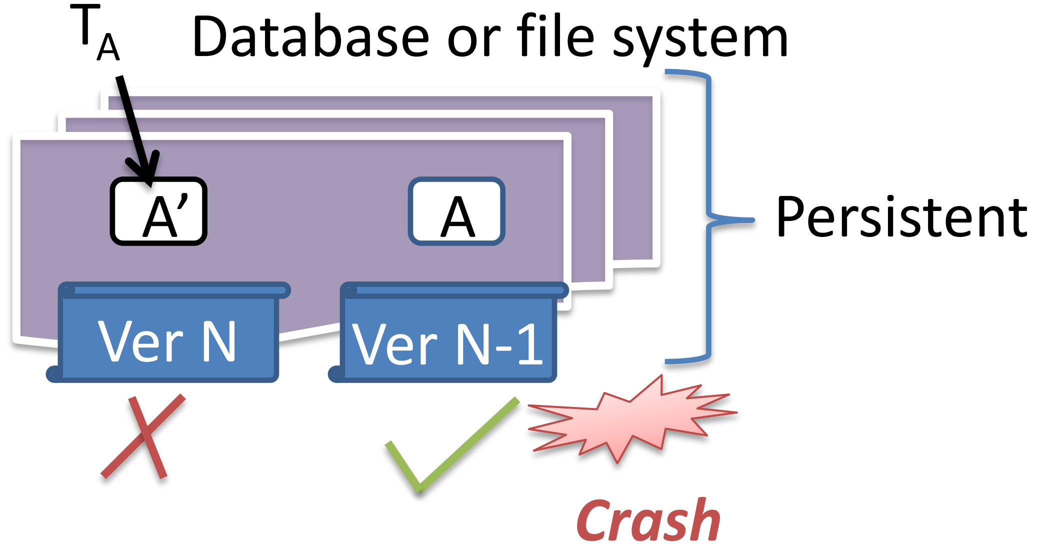
- At least two versions
- Update one version at a time
- Leave the other version intact

Multiversioning



- At least two versions
- Update one version at a time
- Leave the other version intact

Multiversioning



- At least two versions
- Update one version at a time
- Leave the other version intact

Persistence Support of Our Design

Persistence Support of Our Design

ACID maintained in different manners

Persistence Support of Our Design

ACID maintained in different manners

- ACI in an integrated mechanism: e.g., transactional memory

Persistence Support of Our Design

ACID maintained in different manners

- ACI in an integrated mechanism: e.g., transactional memory
- Separate A and D from ACID:
  e.g., failure-atomic msync() [Park+ Eurosys'13]

Persistence Support of Our Design

ACID maintained in different manners

- ACI in an integrated mechanism: e.g., transactional memory
- Separate A and D from ACID:
  e.g., failure-atomic msync() [Park+ Eurosys'13]

Our design

Persistence Support of Our Design

ACID maintained in different manners

- ACI in an integrated mechanism: e.g., transactional memory
- Separate A and D from ACID:
  e.g., failure-atomic msync() [Park+ Eurosys'13]

Our design

- **Hardware** maintains A and D
- **Hardware** preserves the consistency semantics defined by software
  - Ordering of writes defined by software

Persistence Support of Our Design

ACID maintained in different manners

- ACI in an integrated mechanism: e.g., transactional memory
- Separate A and D from ACID:
  e.g., failure-atomic msync() [Park+ Eurosys'13]

Our design

- **Hardware** maintains A and D
- **Hardware** preserves the consistency semantics defined by software
  - Ordering of writes defined by software
- **Software** maintains isolation with concurrency control mechanisms

Persistence Support of Our Design

ACID maintained in different manners

- ACI in an integrated mechanism: e.g., transactional memory
- Separate A and D from ACID:
  e.g., failure-atomic msync() [Park+ Eurosys'13]

Our design

- **Hardware** maintains A and D
- **Hardware** preserves the consistency semantics defined by software
  - Ordering of writes defined by software
- **Software** maintains isolation with concurrency control mechanisms

Hardware portion of persistent memory design effectively maintains multiversioning and preserves ordering

Overheads
of multiversioning and ordering by software methods

Overheads
of multiversioning and ordering by software methods

How to optimize
persistent memory design with hardware support

Multiversioning and Ordering by Software

Design strategy: take a storage system mechanism, optimize its performance

Multiversioning and Ordering by Software

Design strategy: take a storage system mechanism, optimize its performance

Processor

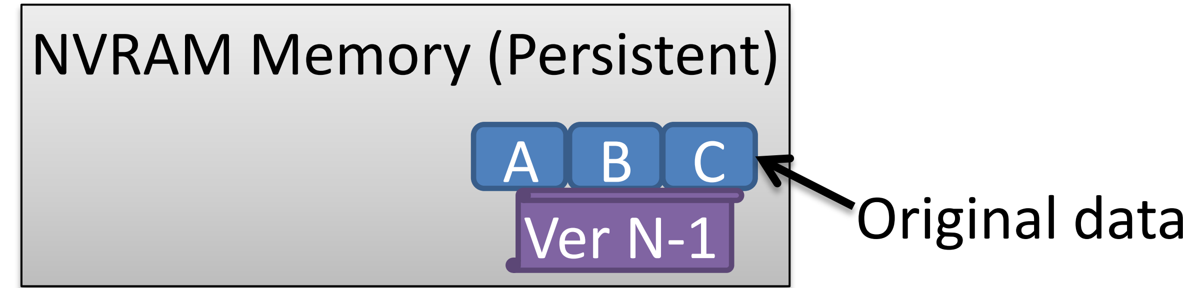
NVRAM Memory (Persistent)

- **Multiversioning Updates do not overwrite original data**
  - Logging [Volos+ ASPLOS'11, Coburn+ ASPLOS'11]
  - Copy-on-write [Condit+ SOSP'09, Venkataraman+ FAST'11]

  (Atomic 8-byte writes [Condit+ SOSP'09, Volos+ ASPLOS'11, Coburn+ ASPLOS'11])

Multiversioning and Ordering by Software

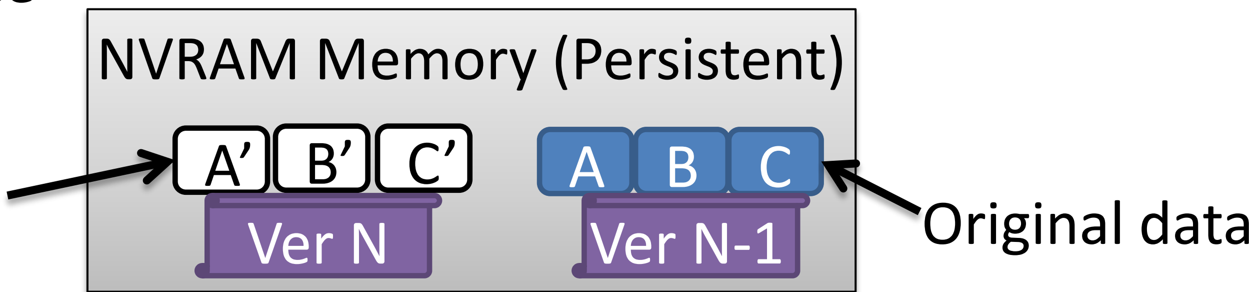
Design strategy: take a storage system mechanism, optimize its performance



- Multiversioning **Updates do not overwrite original data**
  - Logging [Volos+ ASPLOS'11, Coburn+ ASPLOS'11]
  - Copy-on-write [Condit+ SOSP'09, Venkataraman+ FAST'11]
  - (Atomic 8-byte writes [Condit+ SOSP'09, Volos+ ASPLOS'11, Coburn+ ASPLOS'11])

Multiversioning and Ordering by Software

Design strategy: take a storage system mechanism, optimize its performance



Logs, temporary buffers, or newly created versions

Original data

- **Multiversioning Updates do not overwrite original data**
  - Logging [Volos+ ASPLOS'11, Coburn+ ASPLOS'11]
  - Copy-on-write [Condit+ SOSP'09, Venkataraman+ FAST'11]
  - (Atomic 8-byte writes [Condit+ SOSP'09, Volos+ ASPLOS'11, Coburn+ ASPLOS'11])

Multiversioning and Ordering by Software

Design strategy: take a storage system mechanism, optimize its performance

Processor

NVRAM Memory (Persistent)

A' B' C'
Ver N

A B C
Ver N-1

Cache flush, memory fence, epoch barriers

Logs, temporary buffers, or newly created versions

Original data

- **Multiversioning** **Updates do not overwrite original data**
  - Logging [Volos+ ASPLOS'11, Coburn+ ASPLOS'11]
  - Copy-on-write [Condit+ SOSP'09, Venkataraman+ FAST'11]

  (Atomic 8-byte writes [Condit+ SOSP'09, Volos+ ASPLOS'11, Coburn+ ASPLOS'11])
- **Ordering** **Restrict the ordering of writes arriving at memory**
  - Software employs cache flush, memory fence, or uncacheable writes [Volos+ ASPLOS'11, Venkataraman+ FAST'11]
  - Software sets epoch barriers, hardware manages epoch-based writes [Condit+ SOSP'09, Coburn+ ASPLOS'11]

The Cost of Software Maintained Persistence

Persistent Memory

- Updates do not overwrite original data
- Restrict the ordering of writes crossing the memory bus

The Cost of Software Maintained Persistence

Persistent Memory

- Updates do not overwrite original data
- Restrict the ordering of writes crossing the memory bus

Native (Use NVRAM, but no persistence)

The Cost of Software Maintained Persistence

Persistent Memory

- Updates do not overwrite original data
- Restrict the ordering of writes crossing the memory bus

Native (Use NVRAM, but no persistence)

- Updates directly overwrite original data
- Writes reordered by cache writebacks/memory controllers

The Cost of Software Maintained Persistence

Persistent Memory

- Updates do not overwrite original data
- Restrict the ordering of writes crossing the memory bus

Native (Use NVRAM, but no persistence)

- Updates directly overwrite original data
- Writes reordered by cache writebacks/memory controllers

Ideal Performance

The Cost of Software Maintained Persistence

Persistent Memory

- Updates do not overwrite original data
- Restrict the ordering of writes crossing the memory bus

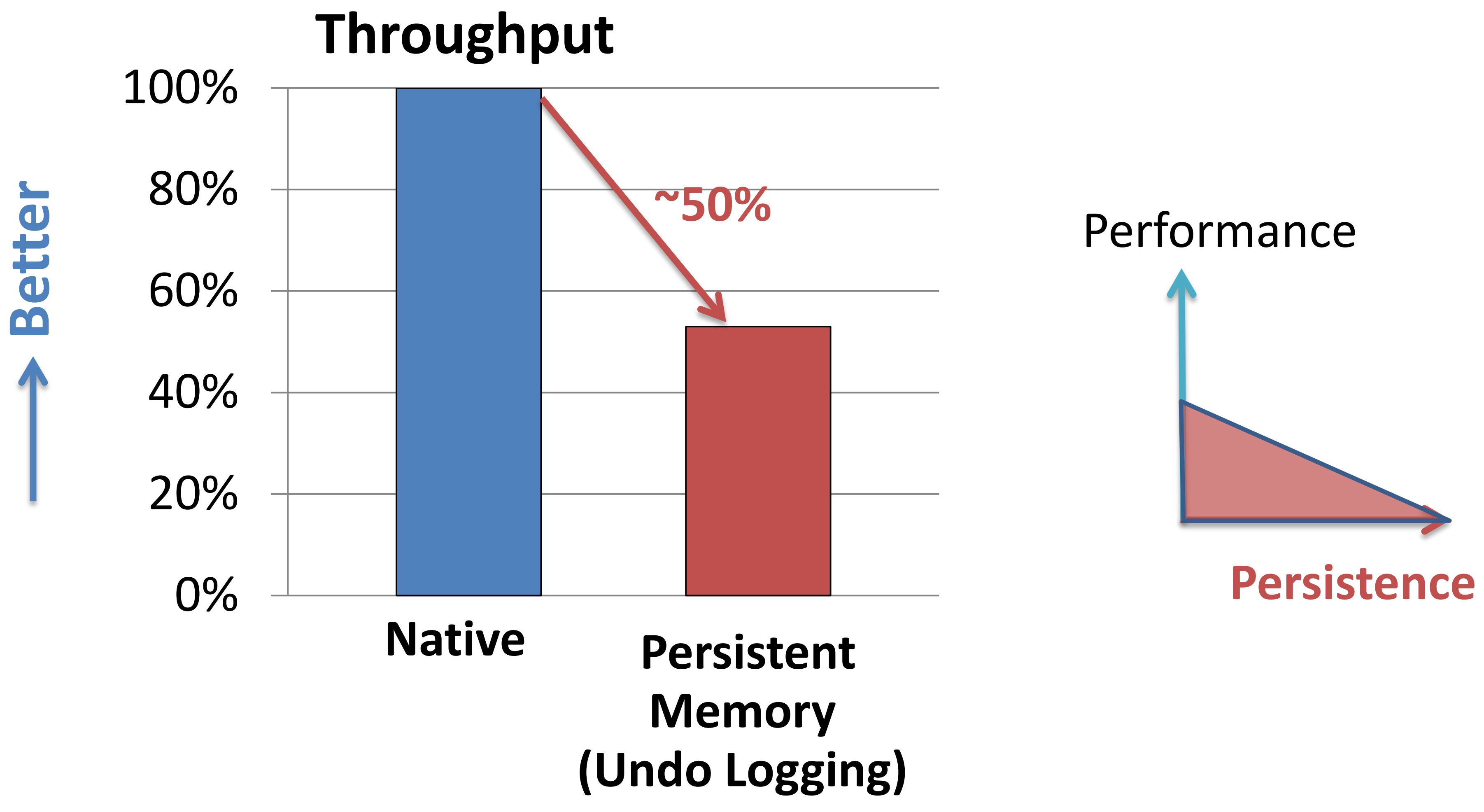
Native (Use NVRAM, but no persistence)

- Updates directly overwrite original data
- Writes reordered by cache writebacks/memory controllers

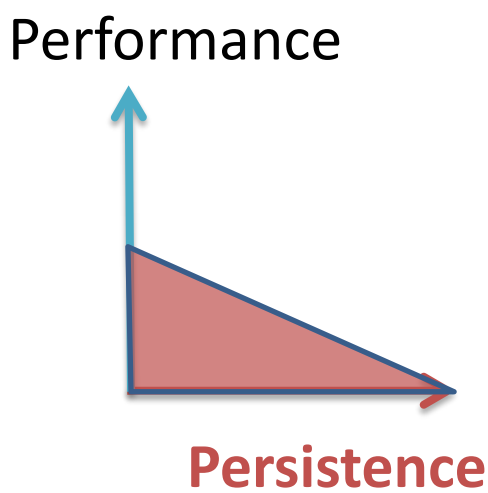
Ideal Performance

Benchmarks: inserts/deletes to B+ tree, hash table, red-black tree, array, and scalable large graph

The Cost of Software Maintained Persistence

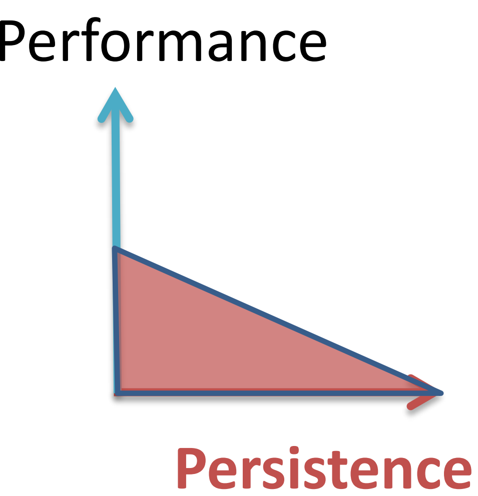


Reasons for the Degradation



Reasons for the Degradation

- **Explicitly duplicate data in memory**
  - By executing logging or copy-on-write instructions

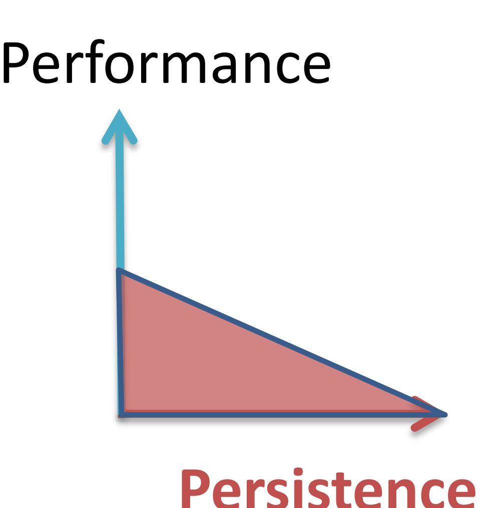


Reasons for the Degradation

- **Explicitly duplicate data in memory**
  - By executing logging or copy-on-write instructions

```
while( true ) {
  transaction_begin { // Transaction X
    read Xa, Xb, Xc;
    do some processing;
    log(X0);
    log(X1);
    write X0, X1;
    read Xd;
    do some other processing;
    log(X2)
    write X2;
  } transaction_end;
}
```

Software logging functions



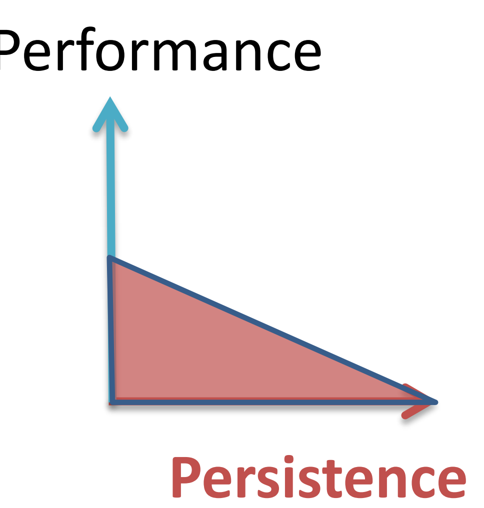
Reasons for the Degradation

- **Explicitly duplicate data in memory**
  - By executing logging or copy-on-write instructions

```
while( true ) {
  transaction_begin { // Transaction X
    read Xa, Xb, Xc;
    do some processing;
    log(X0);
    log(X1);
    write X0, X1;
    read Xd;
    do some other processing;
    log(X2)
    write X2;
  } transaction_end;
}
```

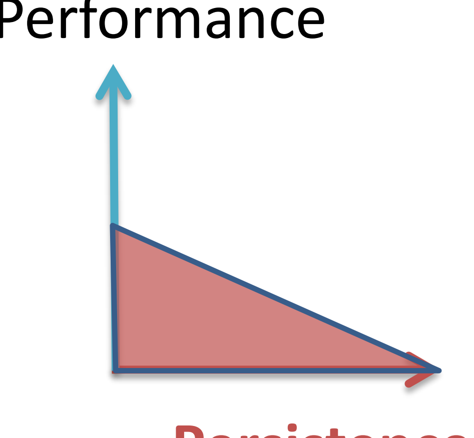
Software logging functions

In-place updates



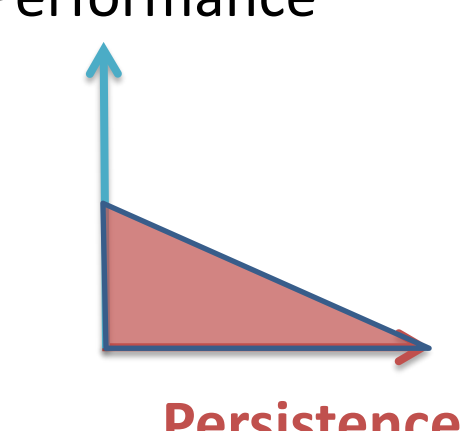
Reasons for the Degradation

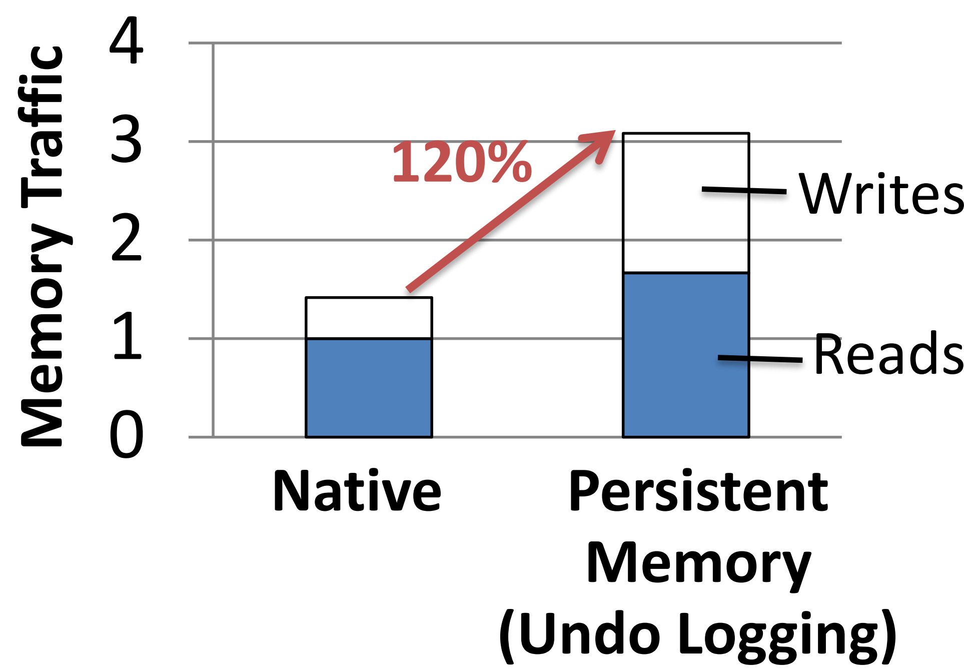
- **Explicitly duplicate data in memory**
  - By executing logging or copy-on-write instructions
- **Two memory stores for one data update**
  - Increase memory activities
  - Saturate memory bandwidth



Reasons for the Degradation

- **Explicitly duplicate data in memory**
  - By executing logging or copy-on-write instructions
- **Two memory stores for one data update**
  - Increase memory activities
  - Saturate memory bandwidth

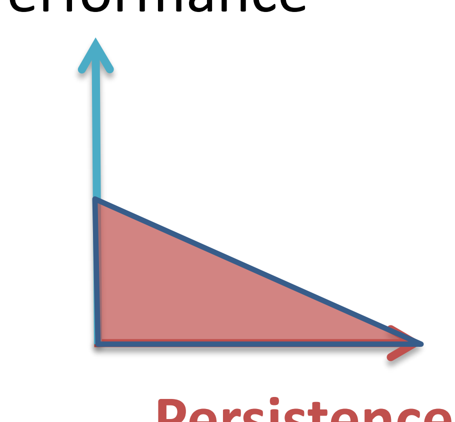




Reasons for the Degradation

- **Explicitly duplicate data in memory**
  - By executing logging or copy-on-write instructions
- **Two memory stores for one data update**
  - Increase memory activities
  - Saturate memory bandwidth
- **In-order updates**
  - Forced all the way down to main memory
  - Cancel out the effect of processor's reordering optimizations

Performance



Persistence

Overheads
of multiversioning and ordering by software methods

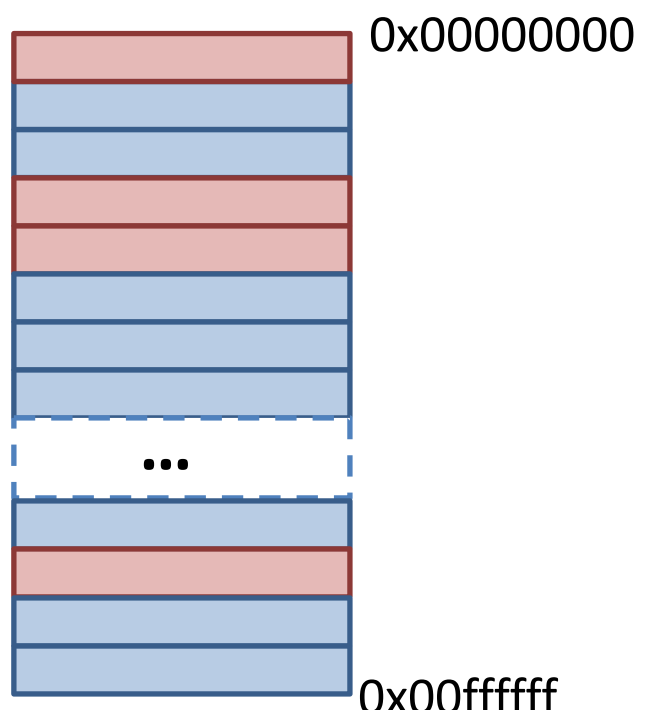
How to optimize
persistent memory design with hardware support

What Can be Leveraged From Hardware

What Can be Leveraged From Hardware

Software's view of memory system

- A flat address space
- Pages



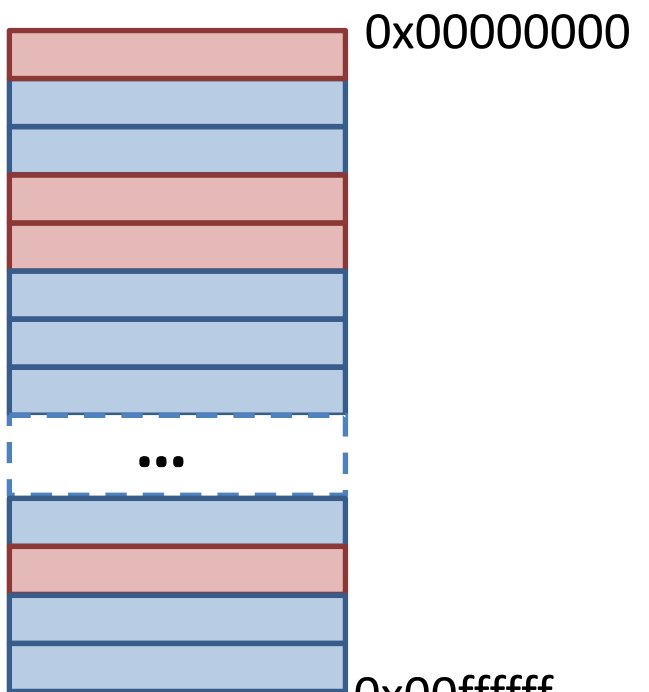
Page belonging to process

Page not belonging to process

What Can be Leveraged From Hardware

Software's view of memory system

- A flat address space
- Pages



Page belonging to process

Page not belonging to process

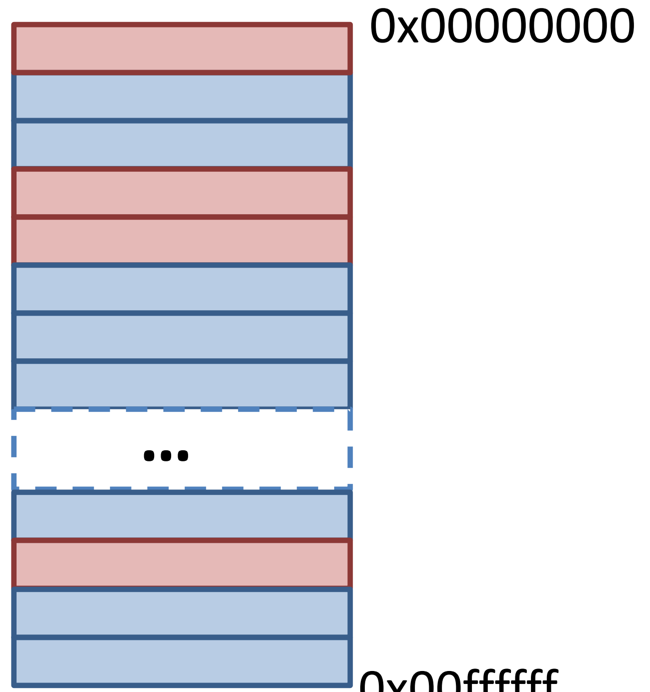
Hardware's view of memory system

- Not flat, a hierarchy
- Cache lines

What Can be Leveraged From Hardware

Software's view of memory system

- A flat address space
- Pages

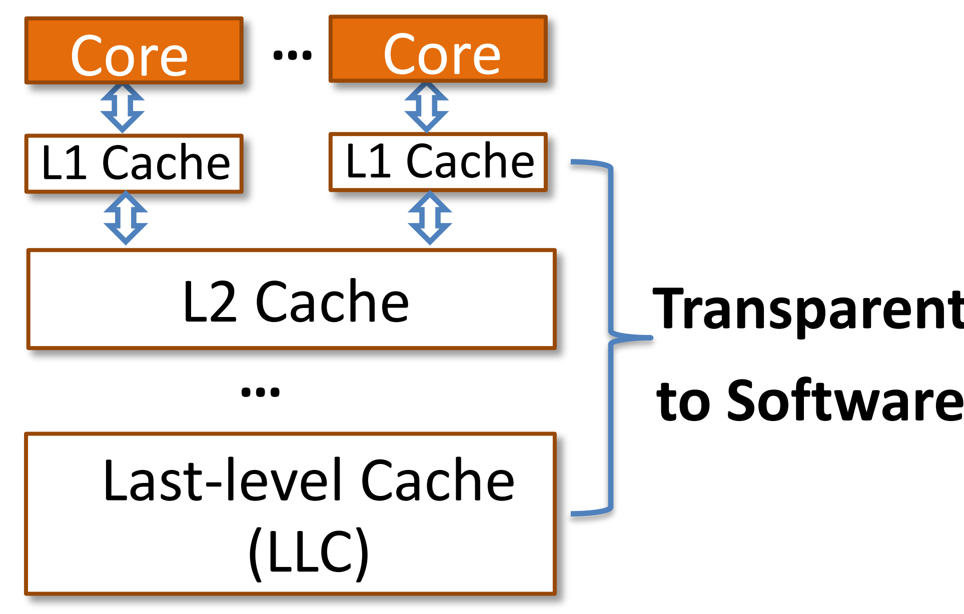


Page belonging to process

Page not belonging to process

Hardware's view of memory system

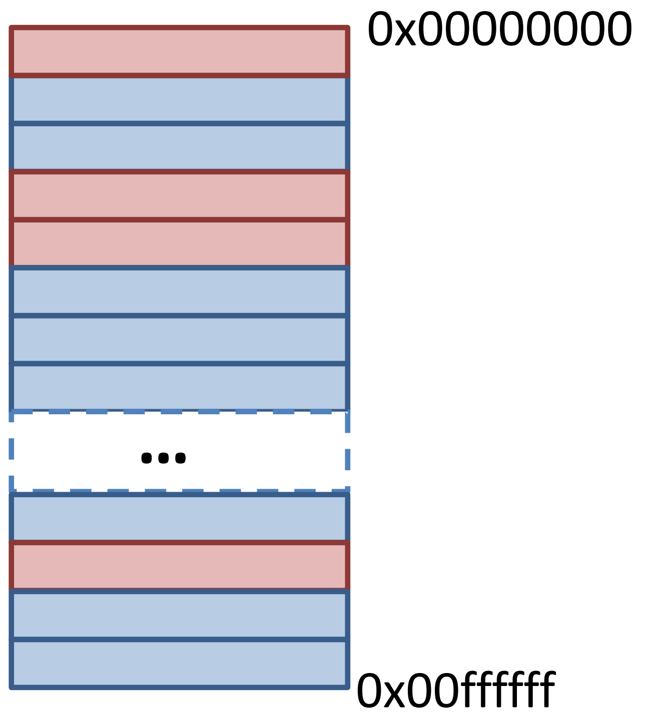
- Not flat, a hierarchy
- Cache lines



What Can be Leveraged From Hardware

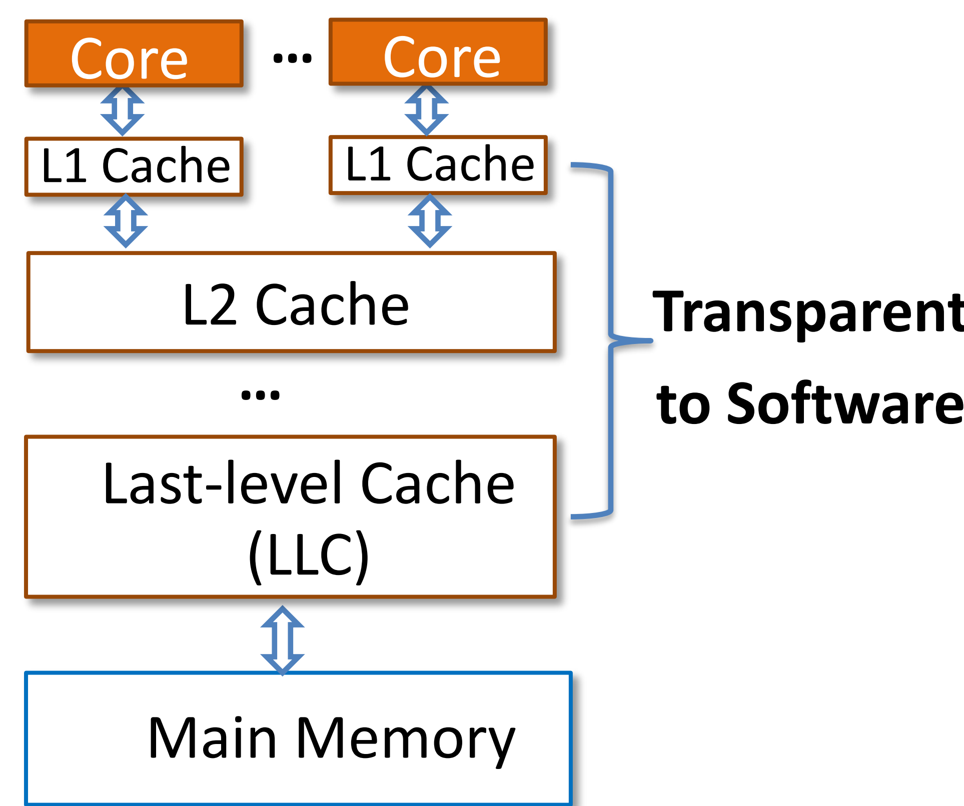
Software's view of memory system

- A flat address space
- Pages



Hardware's view of memory system

- Not flat, a hierarchy
- Cache lines



Multiversioning:
Leveraging Caching for In-place Updates

Multiversioning: Leveraging Caching for In-place Updates

- **How does a write-back cache work?**

Multiversioning: Leveraging Caching for In-place Updates

- **How does a write-back cache work?**
  - A processor writes a value
  - Write to L1 cache only
  - Old values remain in lower levels
  - Until the new value gets evicted

Multiversioning: Leveraging Caching for In-place Updates

- **How does a write-back cache work?**
  - A processor writes a value
  - Write to L1 cache only
  - Old values remain in lower levels
  - Until the new value gets evicted
- **A multiversioned system by nature**

Multiversioning: Leveraging Caching for In-place Updates

- **How does a write-back cache work?**
  - A processor writes a value
  - Write to L1 cache only
  - Old values remain in lower levels
  - Until the new value gets evicted
- **A multiversioned system by nature**

A Multiversioned Persistent Memory Hierarchy

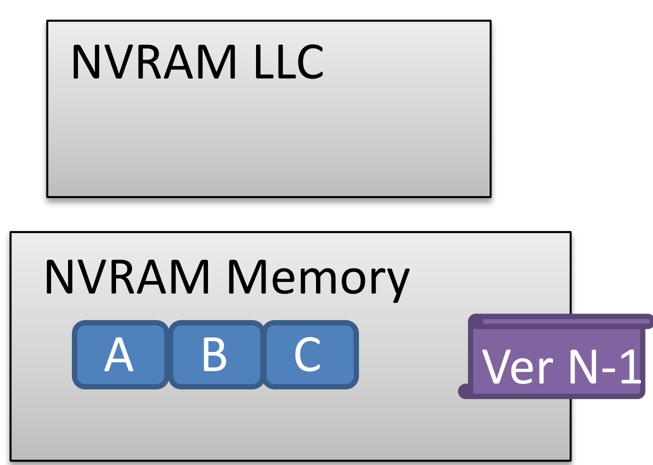
NVRAM LLC

NVRAM Memory

Multiversioning: Leveraging Caching for In-place Updates

- **How does a write-back cache work?**
  - A processor writes a value
  - Write to L1 cache only
  - Old values remain in lower levels
  - Until the new value gets evicted
- **A multiversioned system by nature**

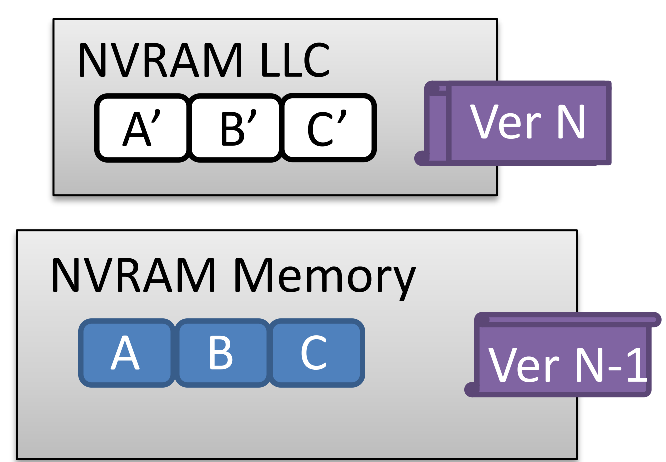
A Multiversioned Persistent Memory Hierarchy



Multiversioning:
Leveraging Caching for In-place Updates

- **How does a write-back cache work?**
  - A processor writes a value
  - Write to L1 cache only
  - Old values remain in lower levels
  - Until the new value gets evicted
- **A multiversioned system by nature**

A Multiversioned Persistent Memory Hierarchy



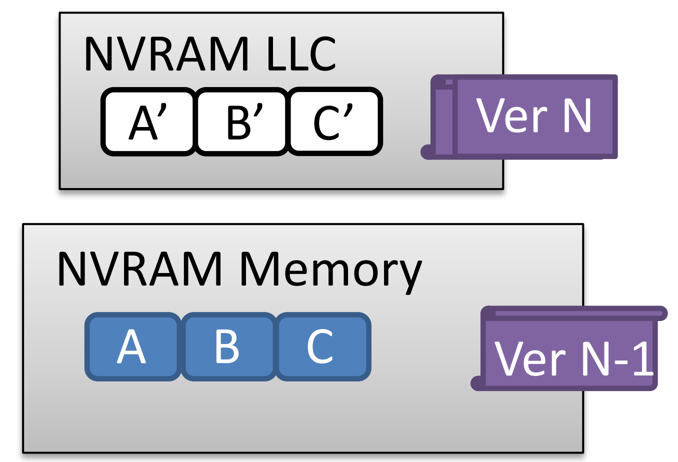
Multiversioning: Leveraging Caching for In-place Updates

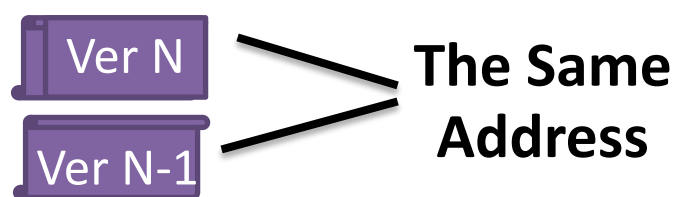
- **How does a write-back cache work?**
  - A processor writes a value
  - Write to L1 cache only
  - Old values remain in lower levels
  - Until the new value gets evicted
- **A multiversioned system by nature**
  - Allow updates to directly overwrite original data
  - No need for logging or copy-on-write

Native

- Updates overwrite original data

A Multiversioned Persistent Memory Hierarchy





The Same Address

Preserving Write Ordering: Out-of-order Writes and In-order Commits

Preserving Write Ordering: Out-of-order Writes and In-order Commits

- **Out-of-order writes to NVRAM LLC**
  - hardware remembers the committing state of each cache line in NVRAM LLC

Preserving Write Ordering: Out-of-order Writes and In-order Commits

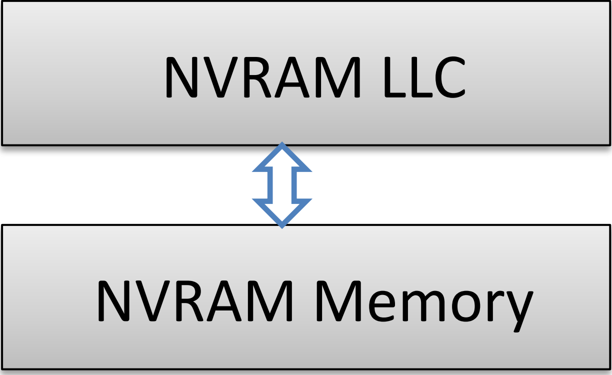
- **Out-of-order writes to NVRAM LLC**
  - hardware remembers the committing state of each cache line in NVRAM LLC
- **In-order commits of transactions**

Preserving Write Ordering: Out-of-order Writes and In-order Commits

- **Out-of-order writes to NVRAM LLC**
  - hardware remembers the committing state of each cache line in NVRAM LLC
- **In-order commits of transactions**
  - Example: T_A before T_B

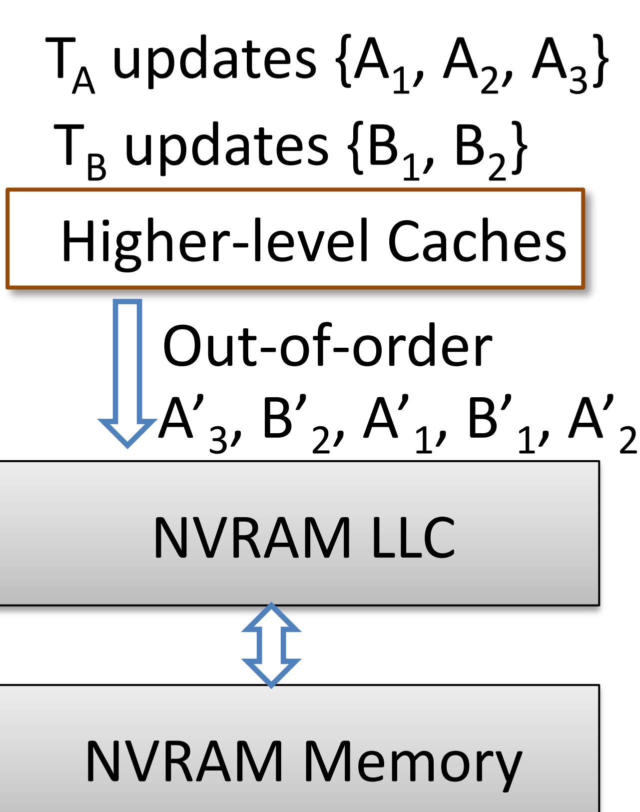
T_A updates $\{A_1, A_2, A_3\}$

T_B updates $\{B_1, B_2\}$



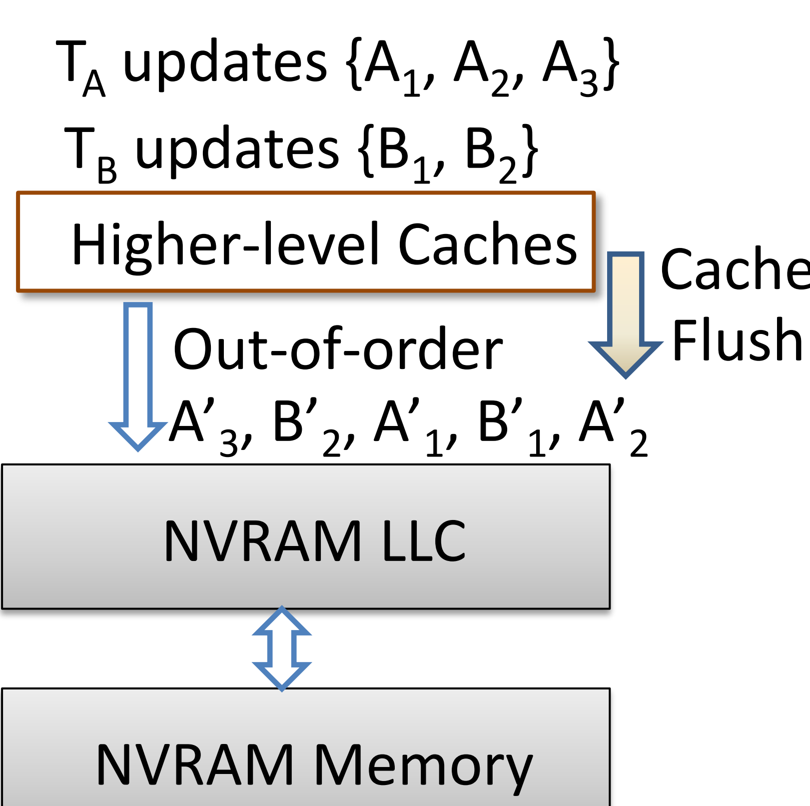
Preserving Write Ordering: Out-of-order Writes and In-order Commits

- **Out-of-order writes to NVRAM LLC**
  - hardware remembers the committing state of each cache line in NVRAM LLC
- **In-order commits of transactions**
  - Example: T_A before T_B
  - T_B will not commit until A'_2 arrives in NVRAM LLC



Preserving Write Ordering: Out-of-order Writes and In-order Commits

- **Out-of-order writes to NVRAM LLC**
  - hardware remembers the committing state of each cache line in NVRAM LLC
- **In-order commits of transactions**
  - Example: T_A before T_B
  - T_B will not commit until A'_2 arrives in NVRAM LLC
- **Commit a transaction**
  - Flush higher-level caches -- very fast
  - Change cache line states in NVRAM LLC



Code Examples
Before and After Using Our Design

Code Examples
Before and After Using Our Design

Before

```
while( true ) {
  transaction_begin { // Transaction X
    read Xa, Xb, Xc;
    do some processing;
    log(X0);
    log(X1);
    write X0, X1;
    read Xd;
    do some other processing;
    log(X2)
    write X2;
  } transaction_end;
}
```

Software logging functions

In-place updates

Code Examples
Before and After Using Our Design

Before

```
while( true ) {
  transaction_begin { // Transaction X
    read Xa, Xb, Xc;
    do some processing;
    log(X0);
    log(X1);
    write X0, X1;
    read Xd
    do some
    log(X2)
    write X
  } transact
}
```

Software logging functions

In-place updates

After

```
while( true ) {
  persistent{ // Transaction X
    read Xa, Xb, Xc;
    do some processing;
    write X0, X1;
    read Xd;
    do some other processing;
    write X2;
  }
}
```

In-place updates

Native

- Updates directly overwrite original data
- Reordered memory requests

Native

- Updates directly overwrite original data
- Reordered memory requests

Persistence →

Conventional Design

- Updates do not overwrite original data
- Restrict ordering of writes crossing the memory bus

Native

- Updates directly overwrite original data
- Reordered memory requests

Persistence →

Conventional Design

- Updates do not overwrite original data
- Restrict ordering of writes crossing the memory bus

Leverage hardware support for performance optimizations →

Our Design

- Updates directly overwrite original data
- Restrict ordering of writes arriving at LLC

Native

- Updates directly overwrite original data
- Reordered memory requests

Persistence →

Conventional Design

- Updates do not overwrite original data
- Restrict ordering of writes crossing the memory bus

Leverage hardware support for performance optimizations

Our Design

- Updates directly overwrite original data
- Restrict ordering of writes arriving at LLC

Address implementation challenges

Native
- Updates directly overwrite original data
- Reordered memory requests

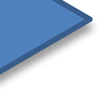
Persistence →

Conventional Design
- Updates do not overwrite original data
- Restrict ordering of writes crossing the memory bus

Leverage hardware support for performance optimizations

Our Design
- Updates directly overwrite original data
- Restrict ordering of writes arriving at LLC

Multiversioning, really?

Address implementation challenges 

A Hardware Memory Barrier

- **Why**
  - Prevents early eviction of uncommitted transactions
  - Avoid violating atomicity

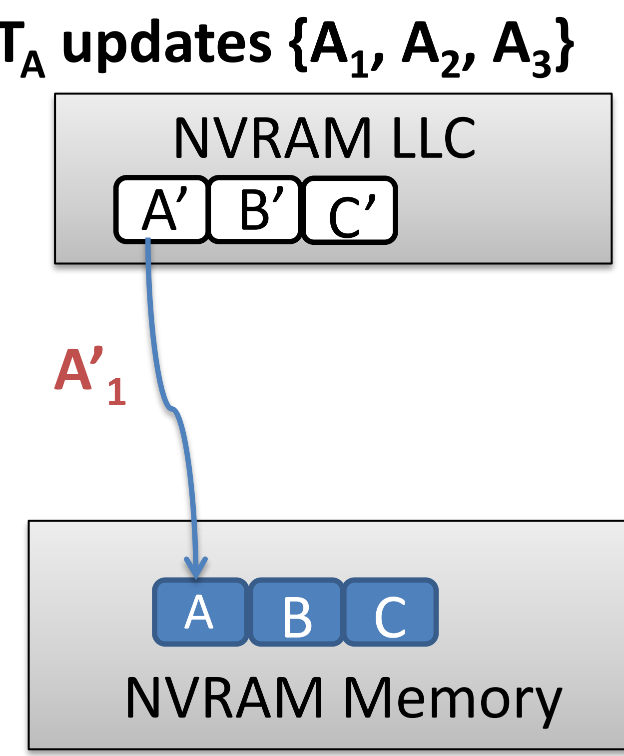
T_A updates {A_1, A_2, A_3}

NVRAM LLC

A’ B’ C’

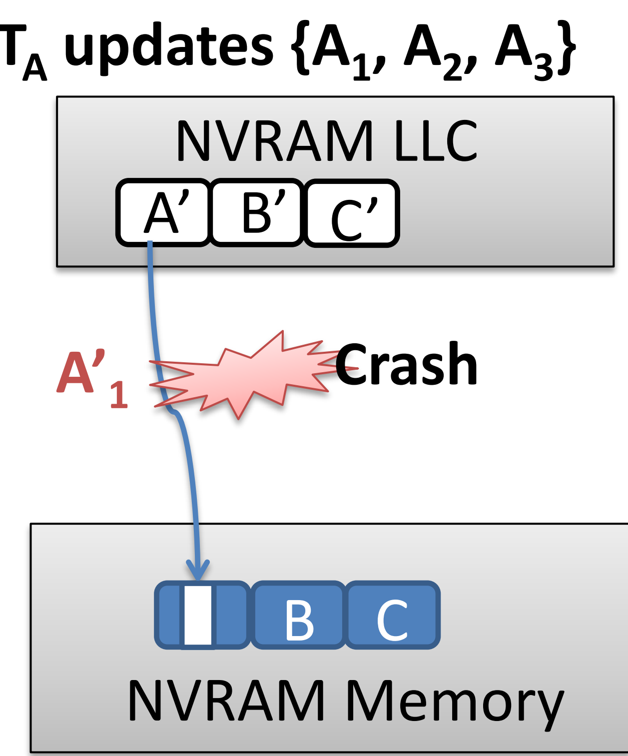
A Hardware Memory Barrier

- **Why**
  - Prevents early eviction of uncommitted transactions
  - Avoid violating atomicity



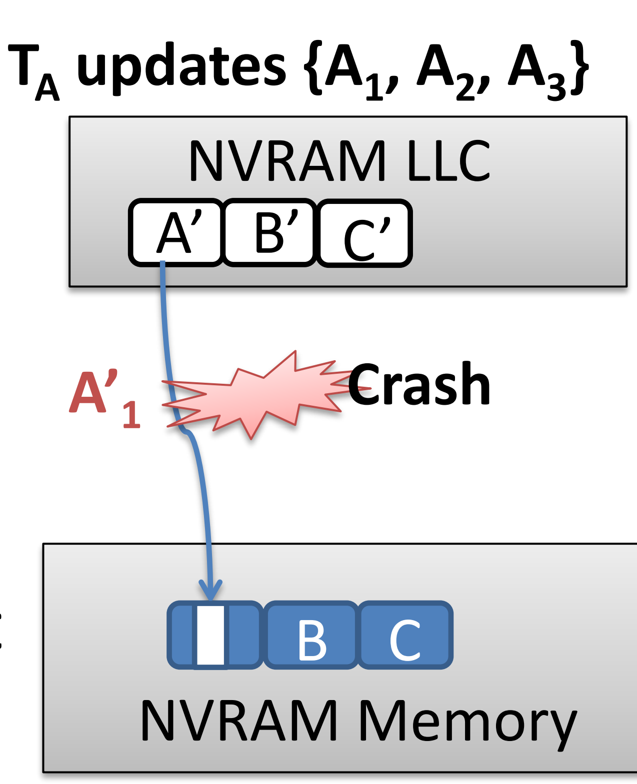
A Hardware Memory Barrier

- **Why**
  - Prevents early eviction of uncommitted transactions
  - Avoid violating atomicity



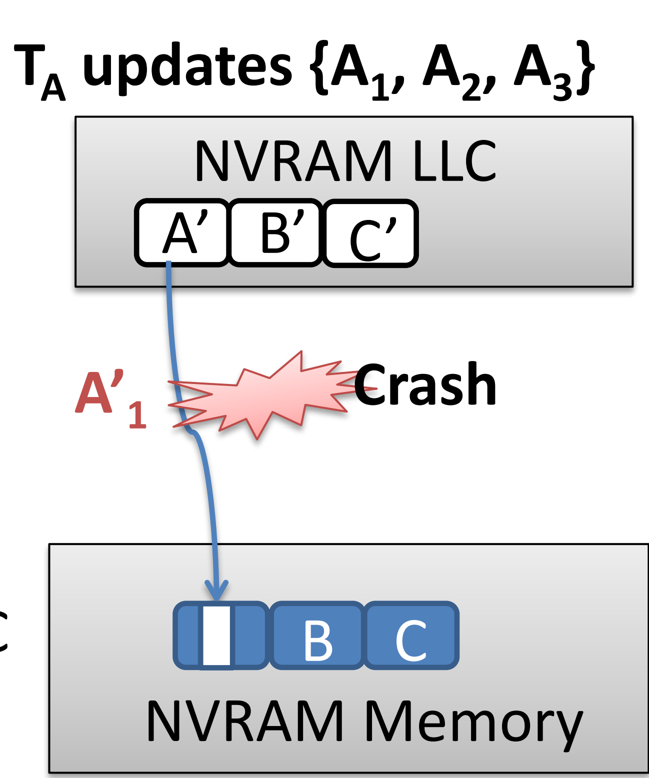
A Hardware Memory Barrier

- **Why**
  - Prevents early eviction of uncommitted transactions
  - Avoid violating atomicity
- **How -- Rule of thumb**
  - Keep uncommitted transactions in NVRAM LLC
- **Implementation**
  - Selective replacement policy in NVRAM LLC
  - Add transactions' committing information to traditional replacement policies



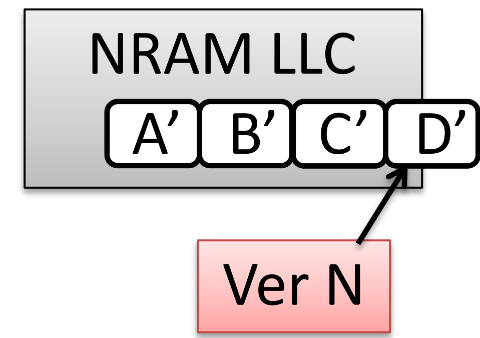
A Hardware Memory Barrier

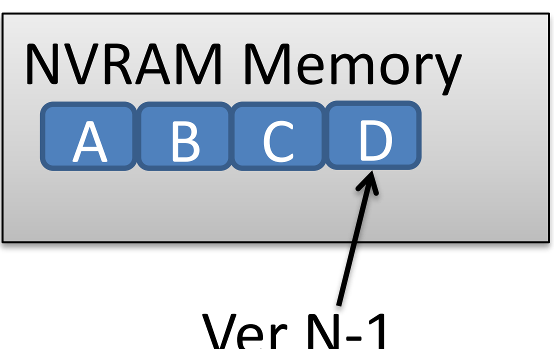
- **Why**
  - Prevents early eviction of uncommitted transactions
  - Avoid violating atomicity
- **How -- Rule of thumb**
  - Keep uncommitted transactions in NVRAM LLC
- **Implementation**
  - Selective replacement policy in NVRAM LLC
  - Add transactions' committing information to traditional replacement policies
- **What if**
  - NVRAM LLC overflows



Addressing NVRAM LLC Overflow

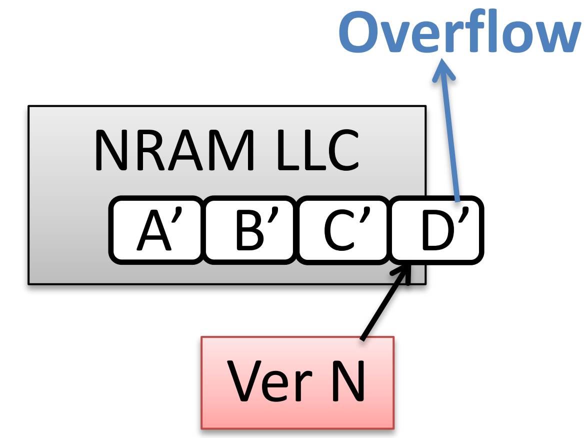
- **Detect overflows by hardware**
  - Continuously monitor transactions' forward progress
  - Set time out limit to prevent deadlocks
- **Provide a fall-back path**
  (Copy-on-write with the overflowed transactions)
  - Hardware notifies OS by interrupts
  - OS allocates temporary data buffers in NVRAM memory

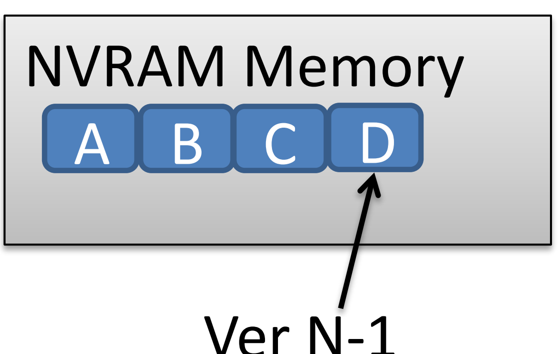




Addressing NVRAM LLC Overflow

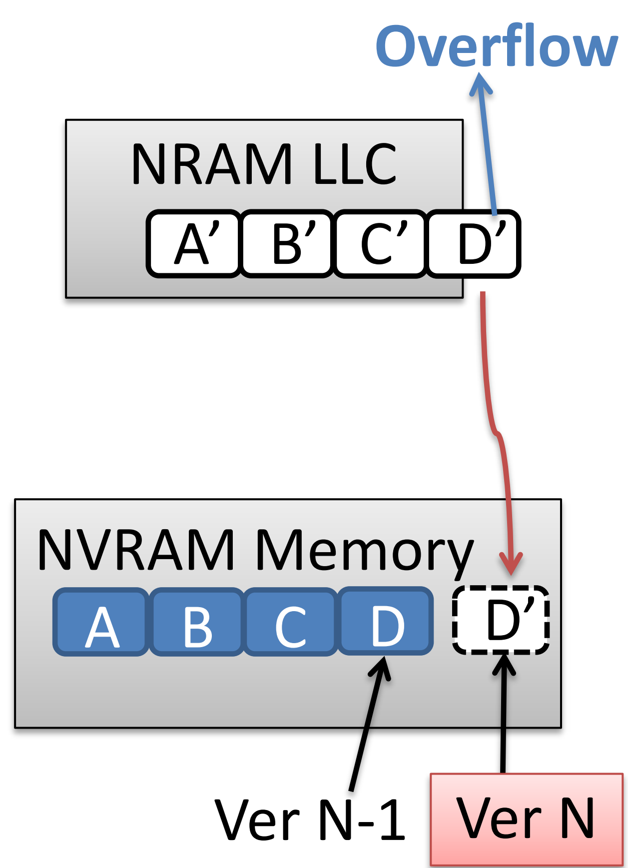
- **Detect overflows by hardware**
  - Continuously monitor transactions' forward progress
  - Set time out limit to prevent deadlocks
- **Provide a fall-back path**
  (Copy-on-write with the overflowed transactions)
  - Hardware notifies OS by interrupts
  - OS allocates temporary data buffers in NVRAM memory





Addressing NVRAM LLC Overflow

- **Detect overflows by hardware**
  - Continuously monitor transactions' forward progress
  - Set time out limit to prevent deadlocks
- **Provide a fall-back path**
  (Copy-on-write with the overflowed transactions)
  - Hardware notifies OS by interrupts
  - OS allocates temporary data buffers in NVRAM memory



Experimental Setup

- Simulator: McSimA+ [Ahn+, ISPASS’13] (modified)
- Configuration
  - Eight-core processor, 16 threads
  - Private caches: L1/L2, SRAM, 64KB/256KB per core
  - Shared last-level cache: L3, STT-MRAM, 64MB
  - Main memory: DRAM (2GB) + STT-MRAM (2GB)
- Benchmarks
  - Perform inserts/deletes to B+ tree, hash table, red-black tree, array, and scalable large graph data structures

Volatile vs. Nonvolatile LLC

Volatile vs. Nonvolatile LLC

No persistence

Volatile vs. Nonvolatile LLC

STT-MRAM L3: 64MB
SRAM based L3: 16MB

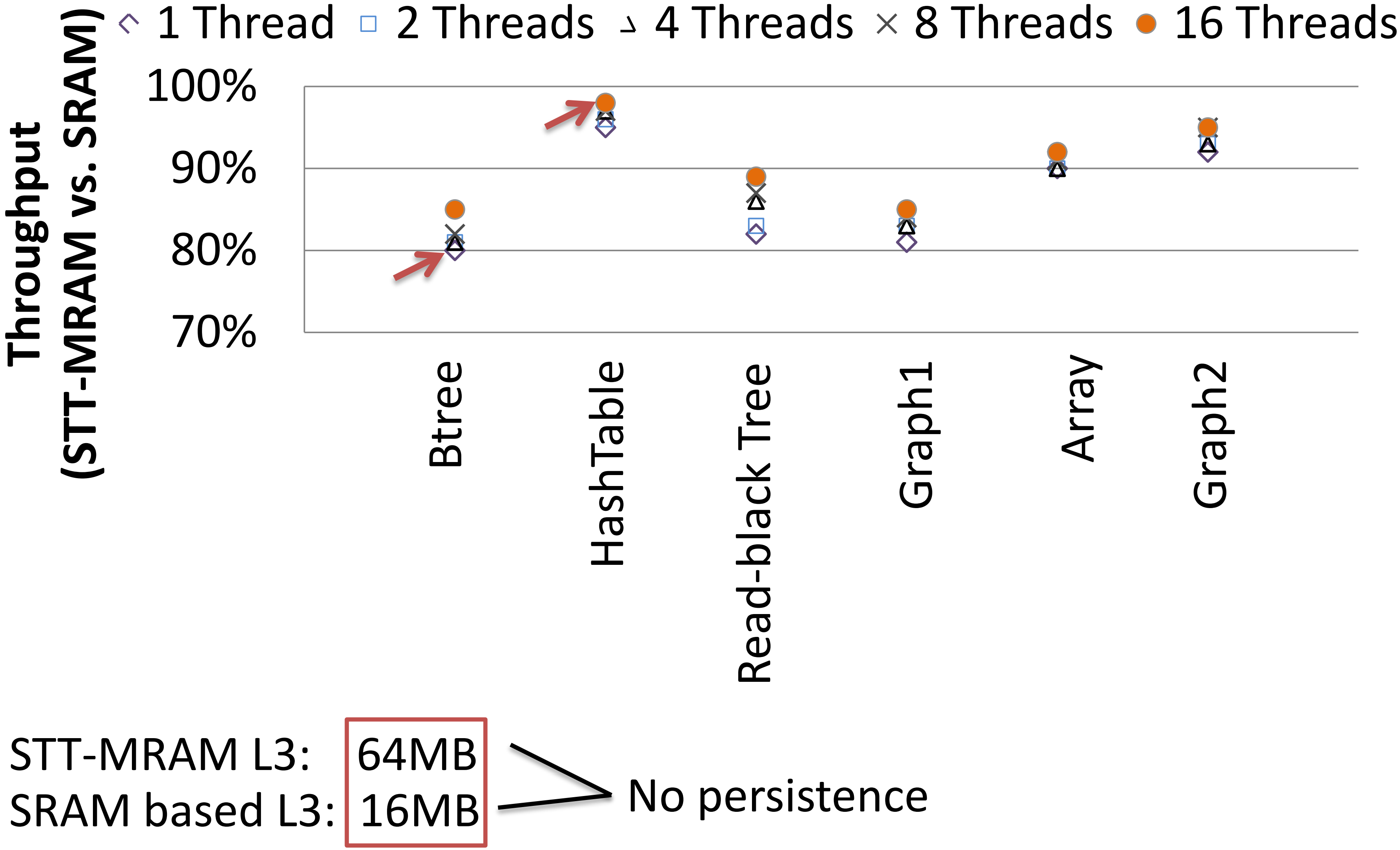
No persistence

Volatile vs. Nonvolatile LLC

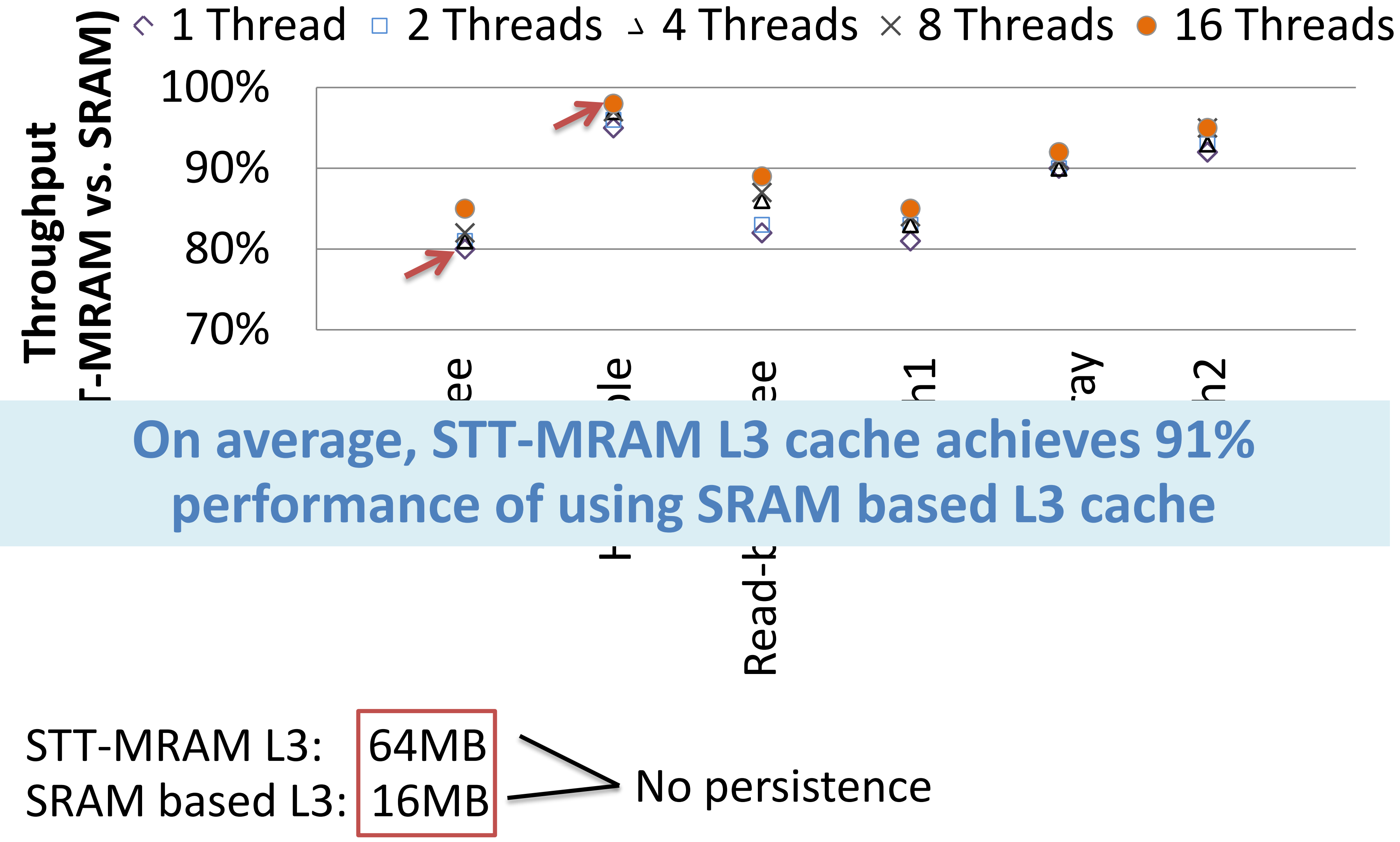
STT-MRAM L3: 64MB
SRAM based L3: 16MB

No persistence

Volatile vs. Nonvolatile LLC



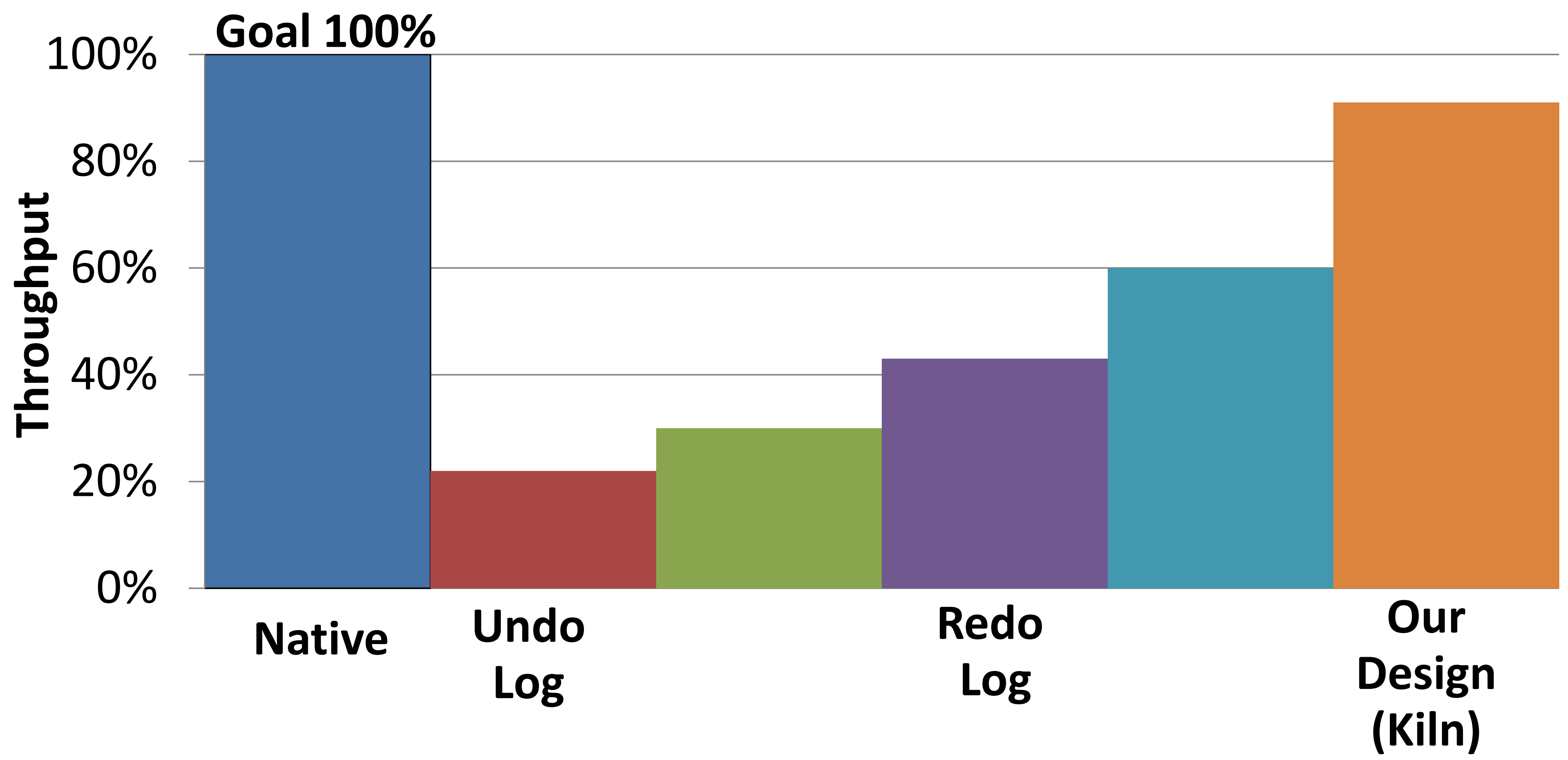
Volatile vs. Nonvolatile LLC



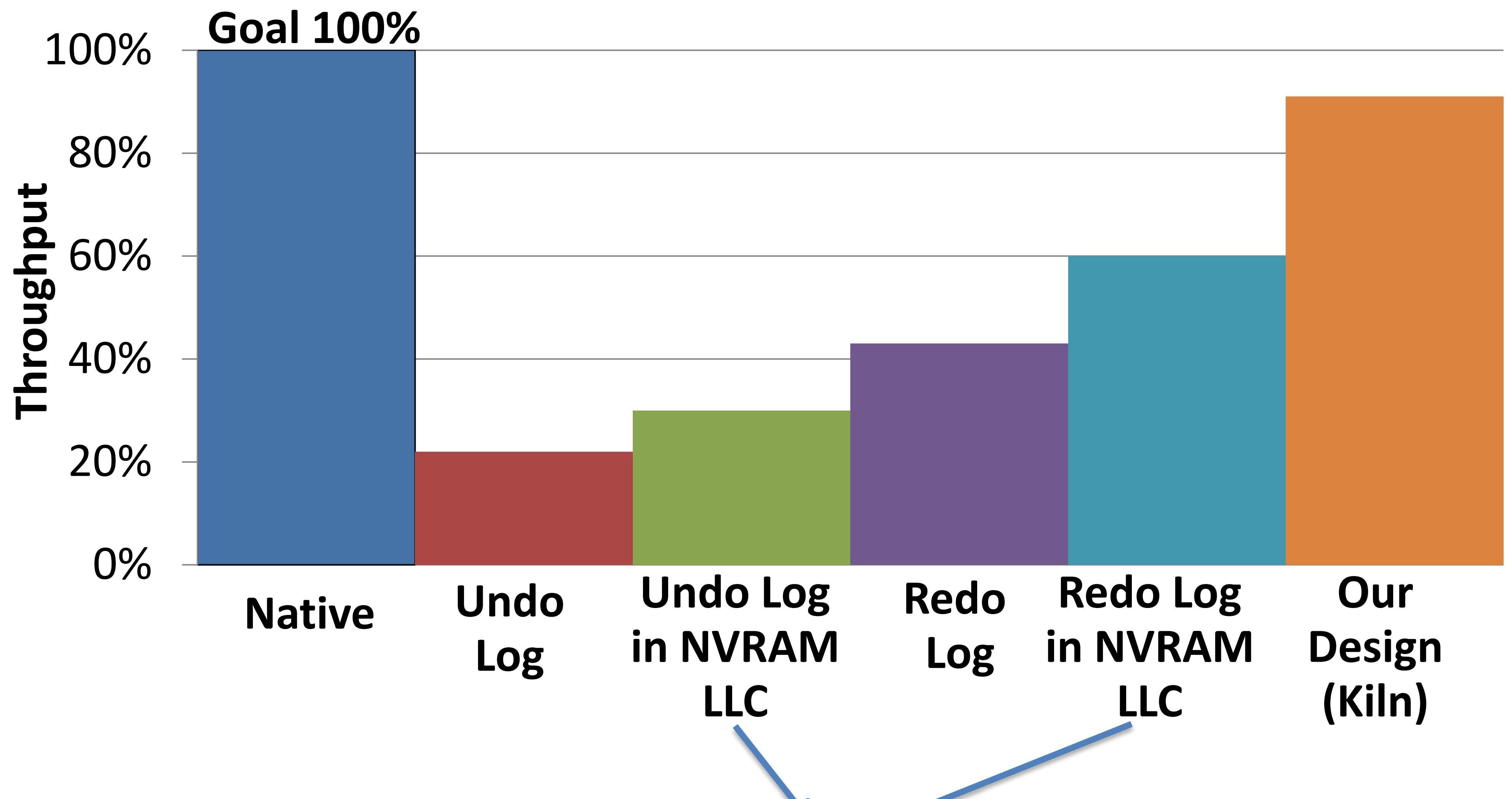
On average, STT-MRAM L3 cache achieves 91% performance of using SRAM based L3 cache

System Performance

System Performance

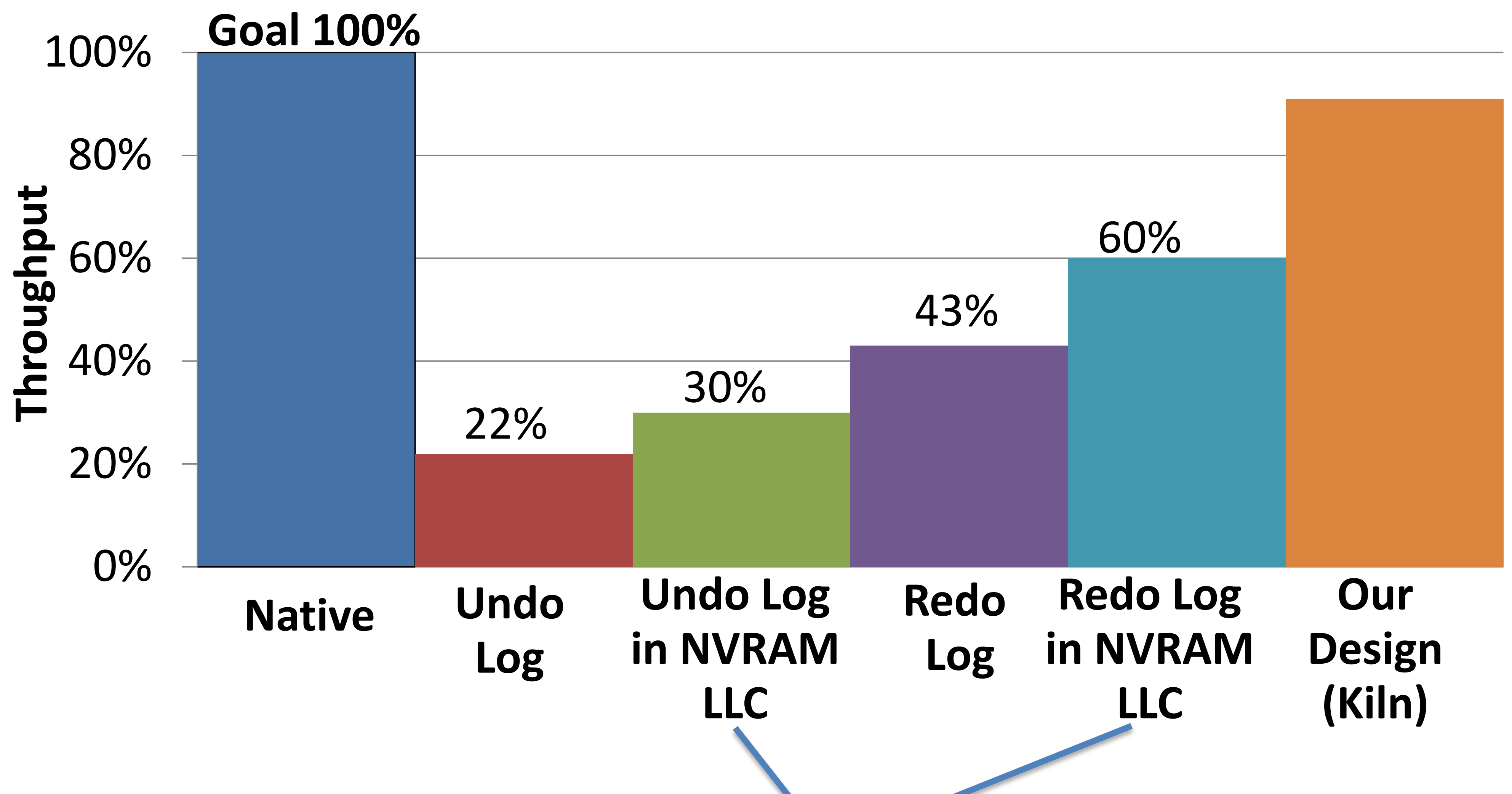


System Performance



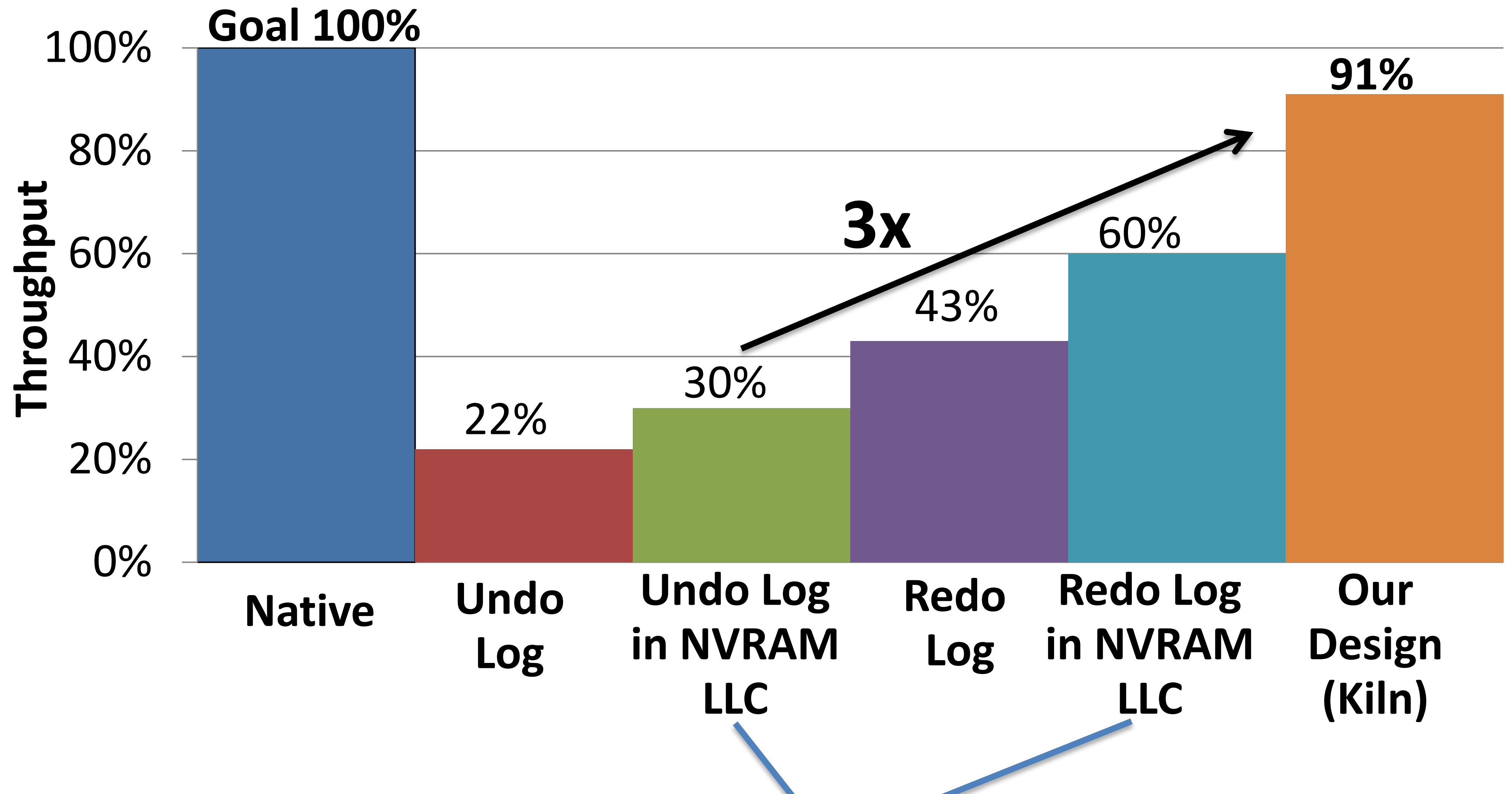
Log updates become persistent once they arrive at the NVRAM LLC

System Performance



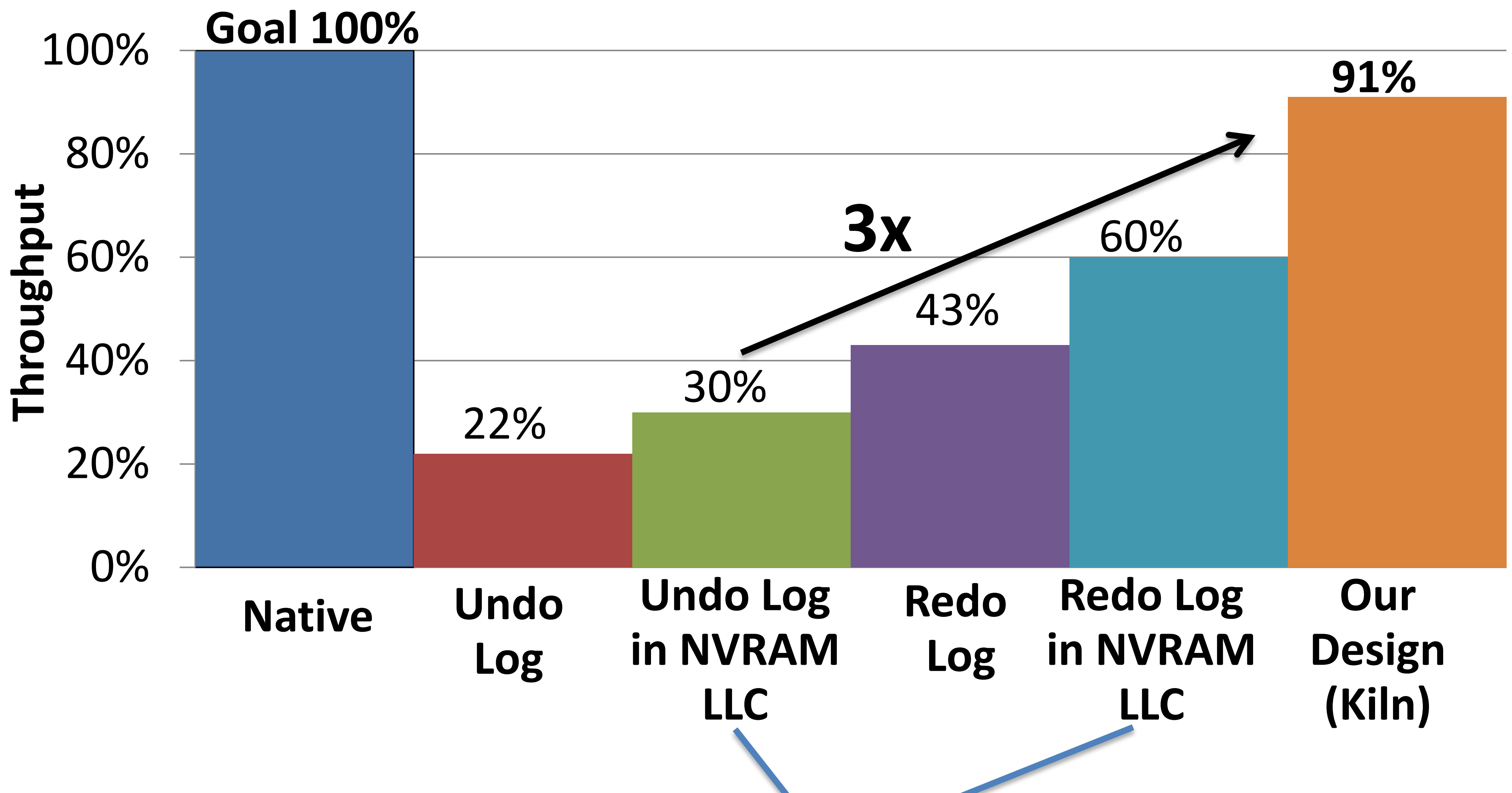
Log updates become persistent once they arrive at the NVRAM LLC

System Performance



Log updates become persistent once they arrive at the NVRAM LLC

System Performance



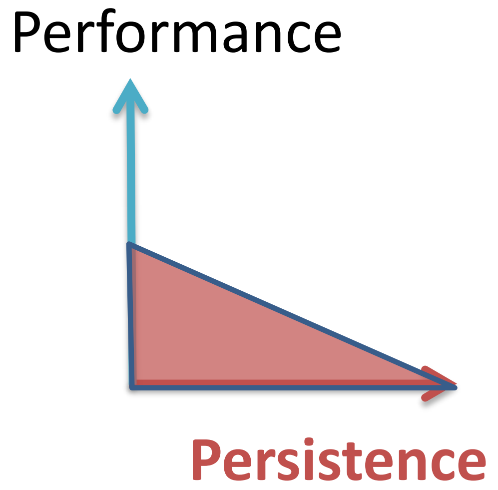
Log updates become persistent once they arrive at the NVRAM LLC

Other results: performance, power, sensitivity studies.

Summary

NVRAM → **DRAM Replacement**

NVRAM → **Persistent Memory (New design opportunity)**

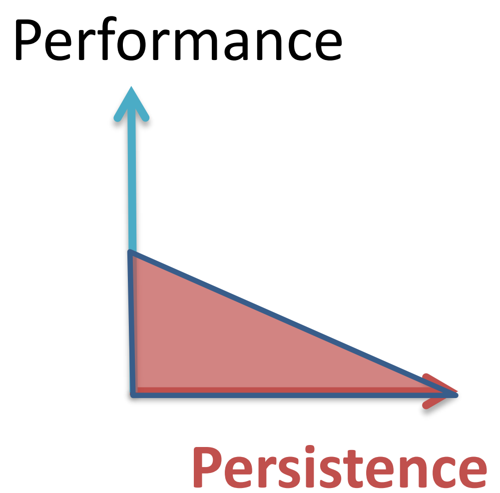


Summary

NVRAM → **DRAM Replacement**

NVRAM → **Persistent Memory (New design opportunity)**

- Proposed a hardware-based persistent memory design
- Leverage a multiversioned persistent memory hierarchy that directly overwrites original data

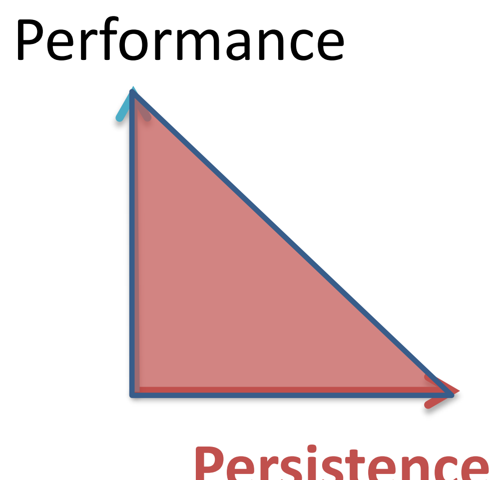


Summary

NVRAM → **DRAM Replacement**

NVRAM → **Persistent Memory (New design opportunity)**

- Proposed a hardware-based persistent memory design
- Leverage a multiversioned persistent memory hierarchy that directly overwrites original data
- Close the performance gap between persistent memory systems and the native system



Why Name It "Kiln"?

"Kiln" was once used by ancient Mesopotamians to bake the clay tablets with temporary scripts and turn them into permanent records. We name our persistent memory design Kiln, because it is analogous to persistent memory which turns volatile data into permanent records.

Thank You!

Kiln: Closing the Performance Gap Between Systems With and Without Persistence Support

Jishen Zhao	*Penn State Univ.*
Sheng Li	*HP Labs*
Doe Hyun Yoon	*IBM Research*
Yuan Xie	*Penn State Univ./AMD Research*
Norm Jouppi	*Google*

MICRO-46, December 11, 2013