

The Arm logo, consisting of the lowercase letters "arm" in a white sans-serif font.

# Architectural Support for Persistent Memory

Dagstuhl Seminar 21462

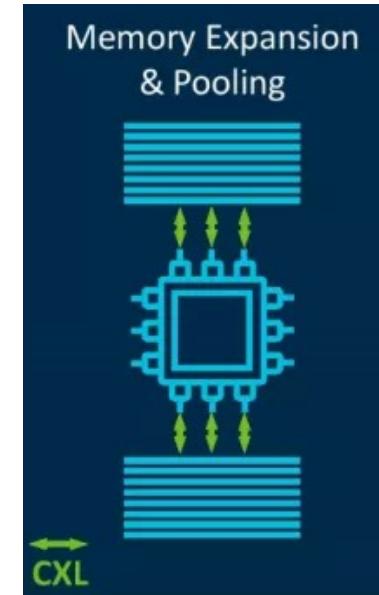
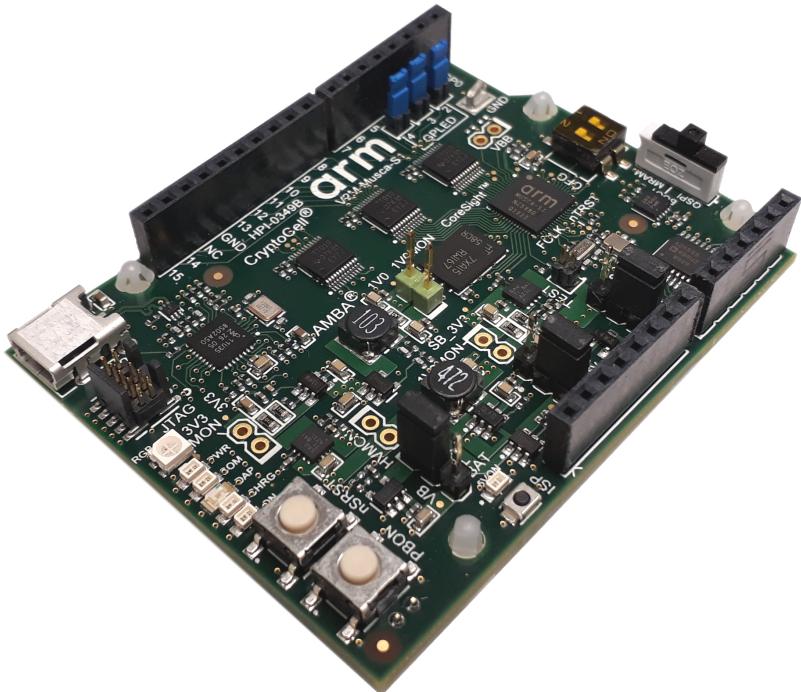
William Wang, Arm Research  
15 Nov 2021

# Executive Summary

- NVM uses
  - *More* memory (denser but slower, i.e., far memory) and *persistent* memory
- Persistent use -> software changes
  - Do we have sufficient support in the Arm architecture for programming persistent memory?
- Problems
  - Persist ordering across threads (concurrency on PM – locking, lock-free and TM)
  - Persist ordering within a thread (weak memory models)
- Solutions
  - Persistent transitive stores (for lock-free concurrency on PM and synchronization primitives)
  - Battery-backed buffers (for concurrency and performance, also sequential programs)
- Other challenges
  - Failure atomicity, persistent addressing

# NVM Augments SRAM, DRAM, NOR, and NAND

In Embedded, Client, and Infrastructure



Arm MUSCA-S1 Board with MRAM at 28nm in 2019

Arm-based Nokia Asha Smartphone with Micron PCM in 2012

CXL Connected Persistent Memory in Infrastructure

# Non-Volatile Memory Opportunities

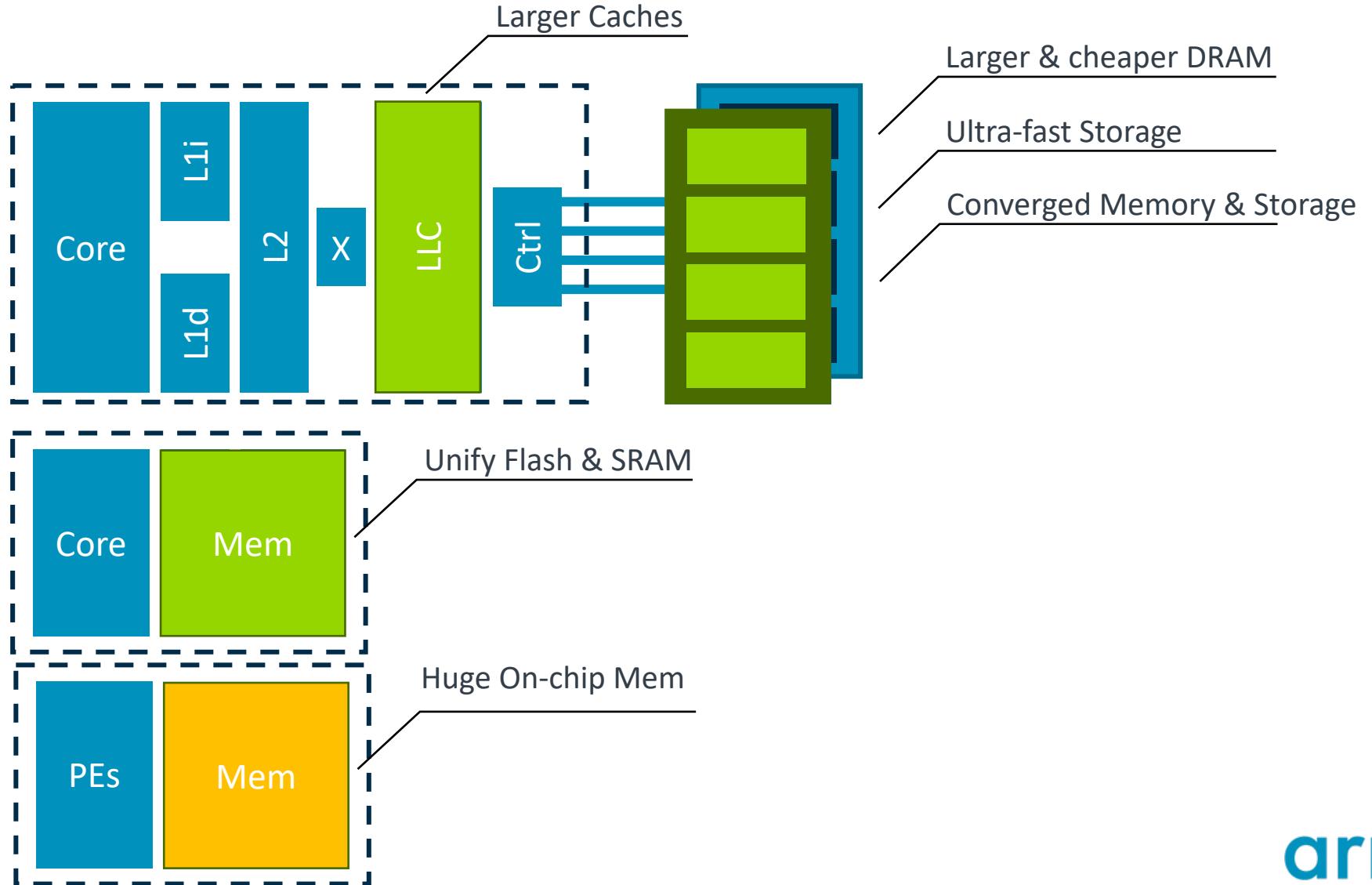
Application-profile  
(servers, phones, ..)

Embedded-profile  
(energy harvesters)

ASIC  
(AI accelerators)

On-chip Usage

Off-chip Usage



# Persistent Use

## Beyond 'More Memory'

- Byte addressable, denser than DRAM
- ***Today:*** new memory technologies offering density and cost improvements over DRAM
- ***Tomorrow:*** unlock performance through single memory for storage and compute

Today

**More Mem**

Denser than DRAM



**TCO/Capacity**

- Endurance
- Latency
- Volatility

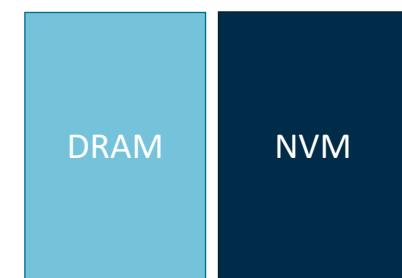
Unlock more perf out  
of cheaper memory



Tomorrow

**Persistent Mem**

Non-Volatile



**Persistency**

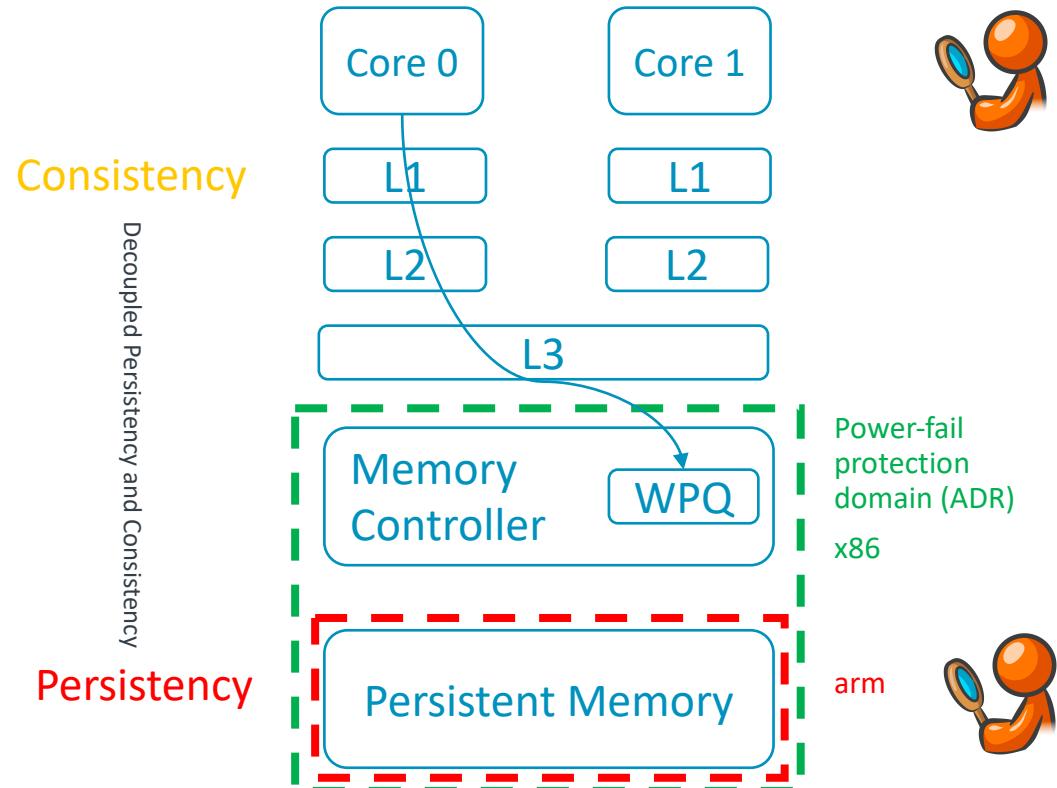
- Failure atomicity
- Persist ordering
- Persistent addressing
- Crash recovery
- Programming models
- ISA & uarch support



# Memory Persistency

Do we have sufficient support in the Arm ISA for  
programming persistent memory?

# System Assumption



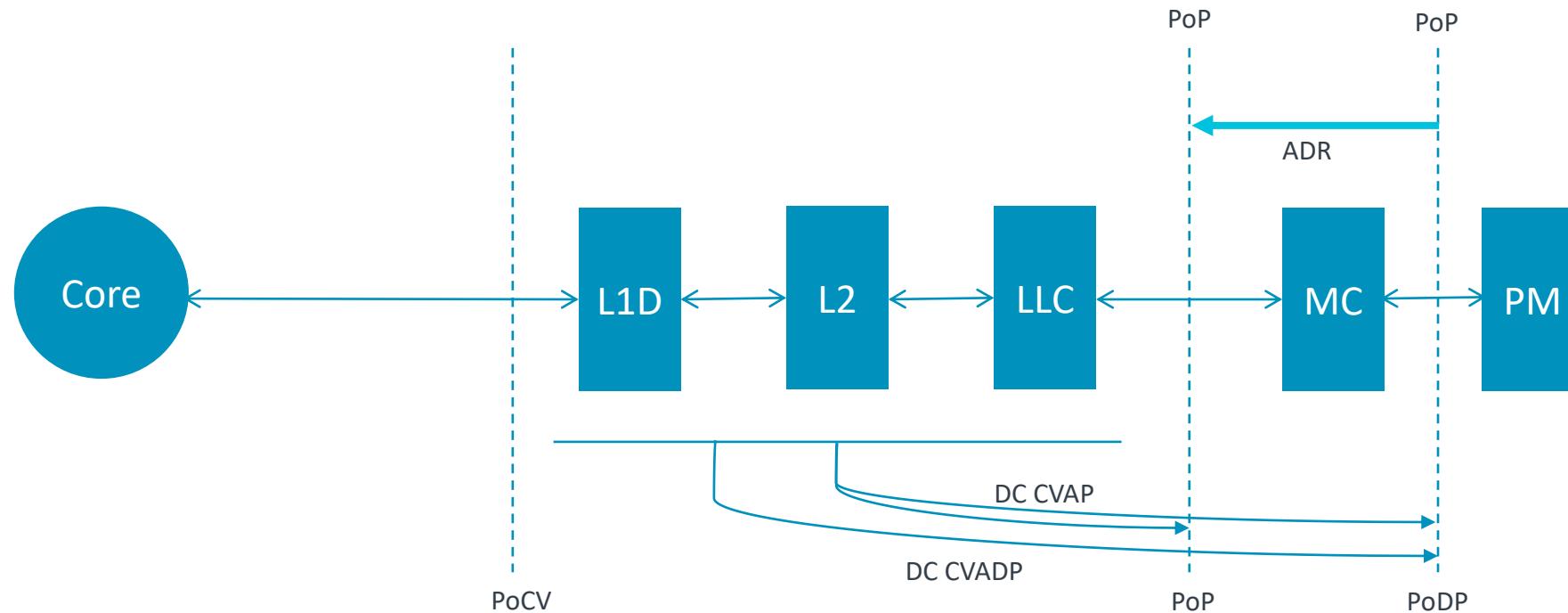
- Point of Persistence (PoP) at the persistent memory module or the memory controller WPQ
  - Contents in the power-fail protection domain will be saved upon power failure
- Caches and cores are still in the volatile domain
  - Contents will be lost upon power failure
- Persistency < Consistency (behind)
  - Stores need to be drained from volatile caches to PoP explicitly by software to sync persistency w. consistency

PoP: Point of Persistence

ADR : Asynchronous DRAM Refresh

WPQ: Write Pending Queue

# Architectural Support to Sync Visibility & Persistency



DC CVAP in Armv8.2-A and DC CVADP in Armv8.5-A

Barrier (DSB) to guarantee completion of DC CVA[D]P cache maintenance operations

Barrier (DMB) to order DC CVA[D]P cache maintenance operations

PoCV: Point of Concurrent Visibility

PoP: Point of Persistence

PoDP: Point of Deep Persistence

ADR : Asynchronous DRAM Refresh

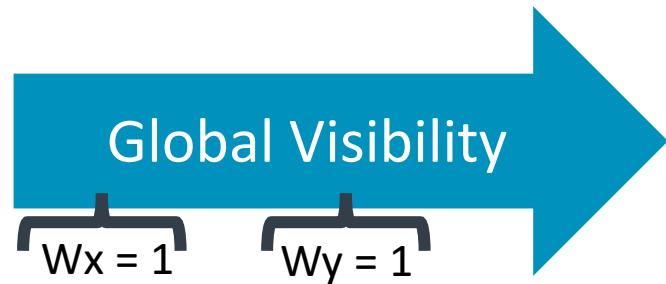
DSB: Data Synchronization Barrier

DMB: Data Memory Barrier

# Global Visibility Order

P0  
STR W0,[X1]  
STR W2,[X3]

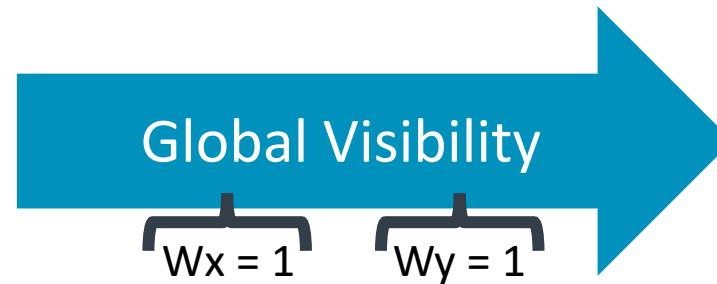
Thread 0  
a: Wx = 1  
 $\downarrow_{po}$   
b: Wy = 1



time

P0  
STR W0,[X1]  
**DMB.ST**  
STR W2,[X3]

Thread 0  
a: Wx = 1  
 $\downarrow_{dmb}$   
b: Wy = 1



time

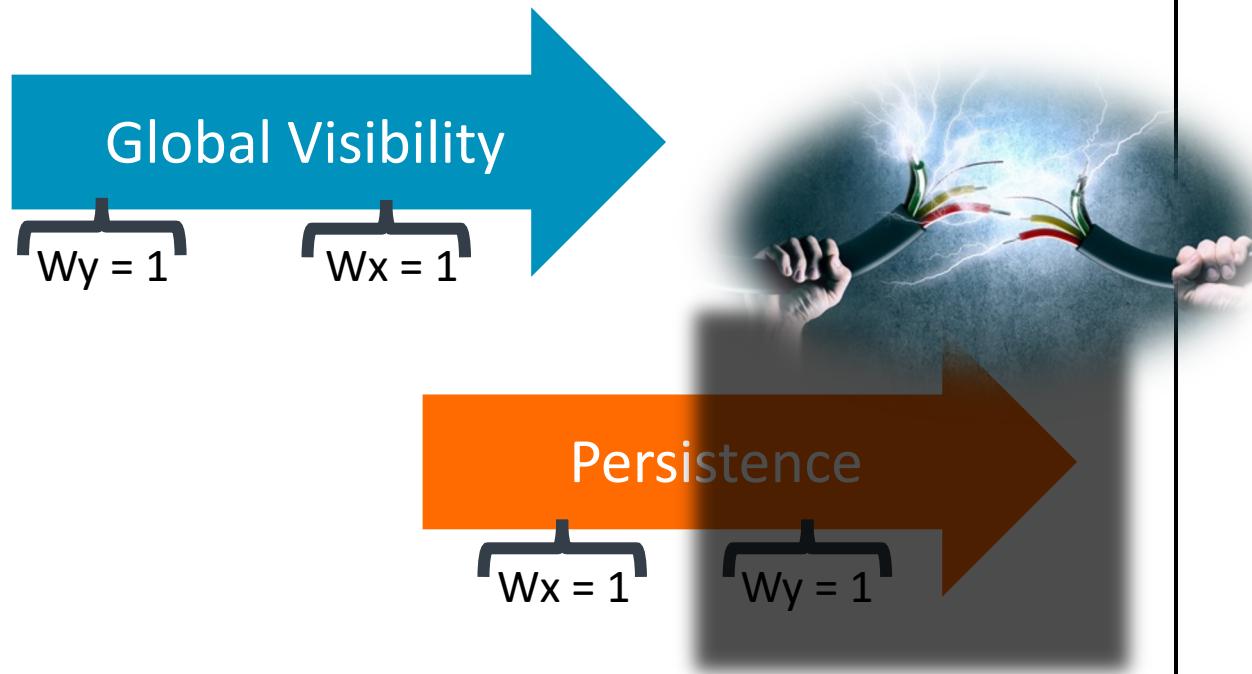
DMB: Data Memory Barrier

DMB.ST: Store barrier

# View of the NVM: Persist Order

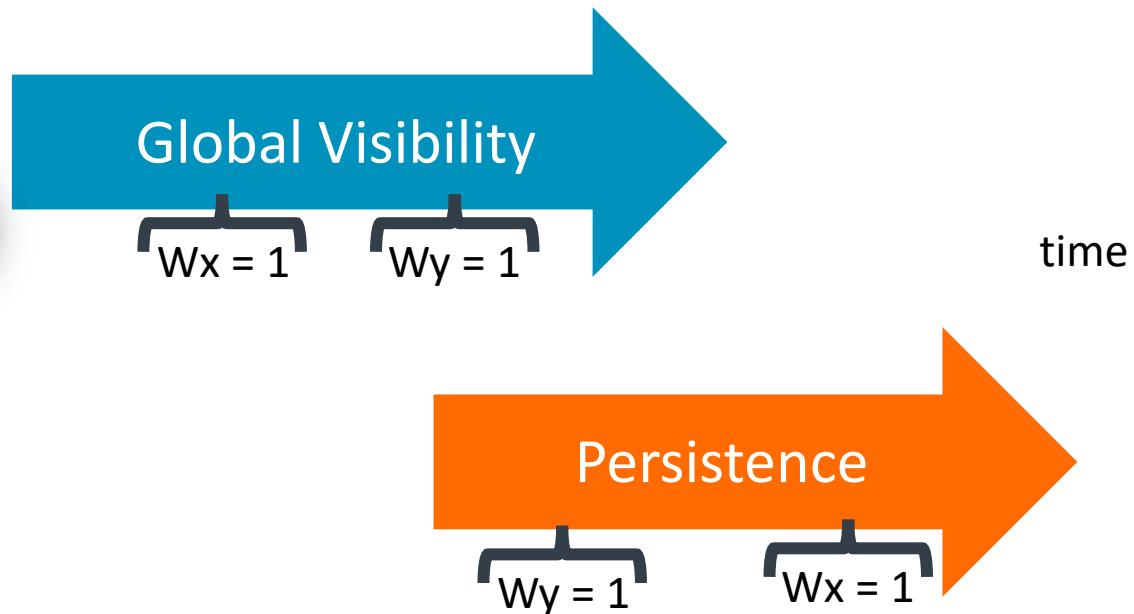
P0  
STR W0,[X1]  
STR W2,[X3]

Thread 0  
a:  $Wx = 1$   
 $\downarrow po$   
b:  $Wy = 1$



P0  
STR W0,[X1]  
**DMB.ST**  
STR W2,[X3]

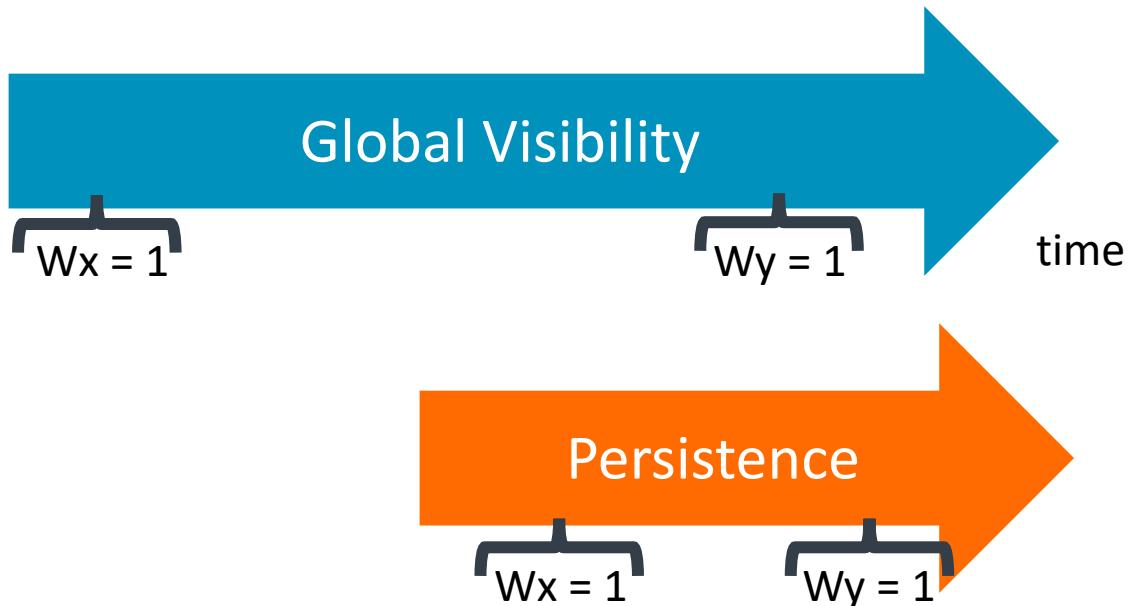
Thread 0  
a:  $Wx = 1$   
 $\downarrow dmb$   
b:  $Wy = 1$



# Enforcing Persist Order

P0  
STR W0,[X1]  
**DC.CVAP [X1]**  
**DSB**  
STR W2,[X3]

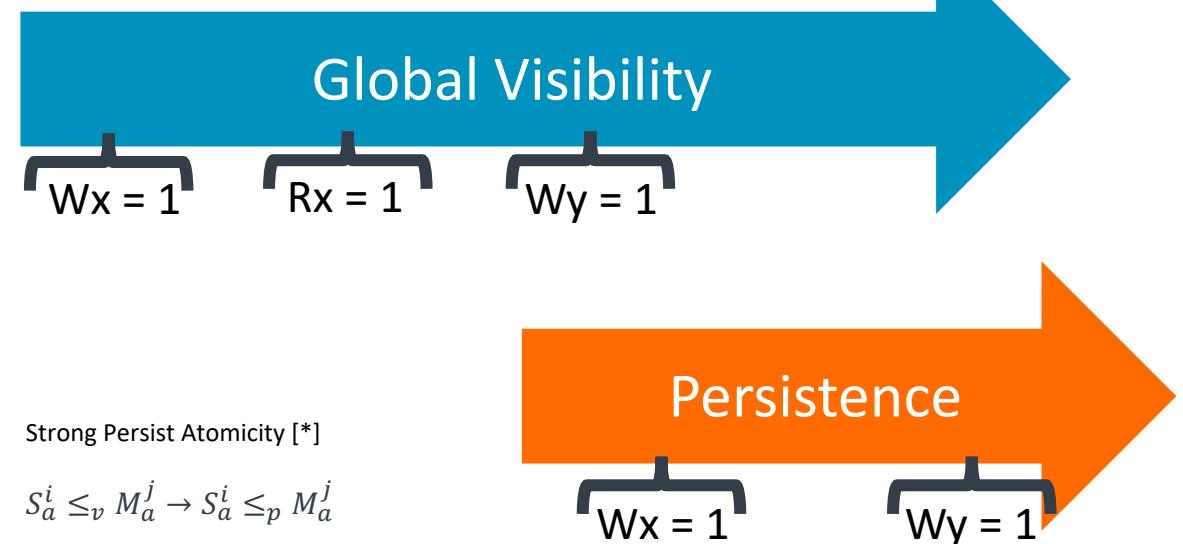
Thread 0  
a:  $W_x = 1$   
dmb ↓ pers ↓  
b:  $W_y = 1$



P0  
STR W0,[X1]  
**DC.CVAP [X1]**  
DSB

P1  
LDR W0,[X1]  
DMB  
STR W2, [X3]  
**DC.CVAP [X3]**  
DSB

Thread 0  
a:  $W_x = 1$   
pers? ↓ rfe  
Thread 1  
b:  $R_x = 1$   
dmb ↓  
c:  $W_y = 1$   
pers?



Strong Persist Atomicity [\*]

$$S_a^i \leq_v M_a^j \rightarrow S_a^i \leq_p M_a^j$$

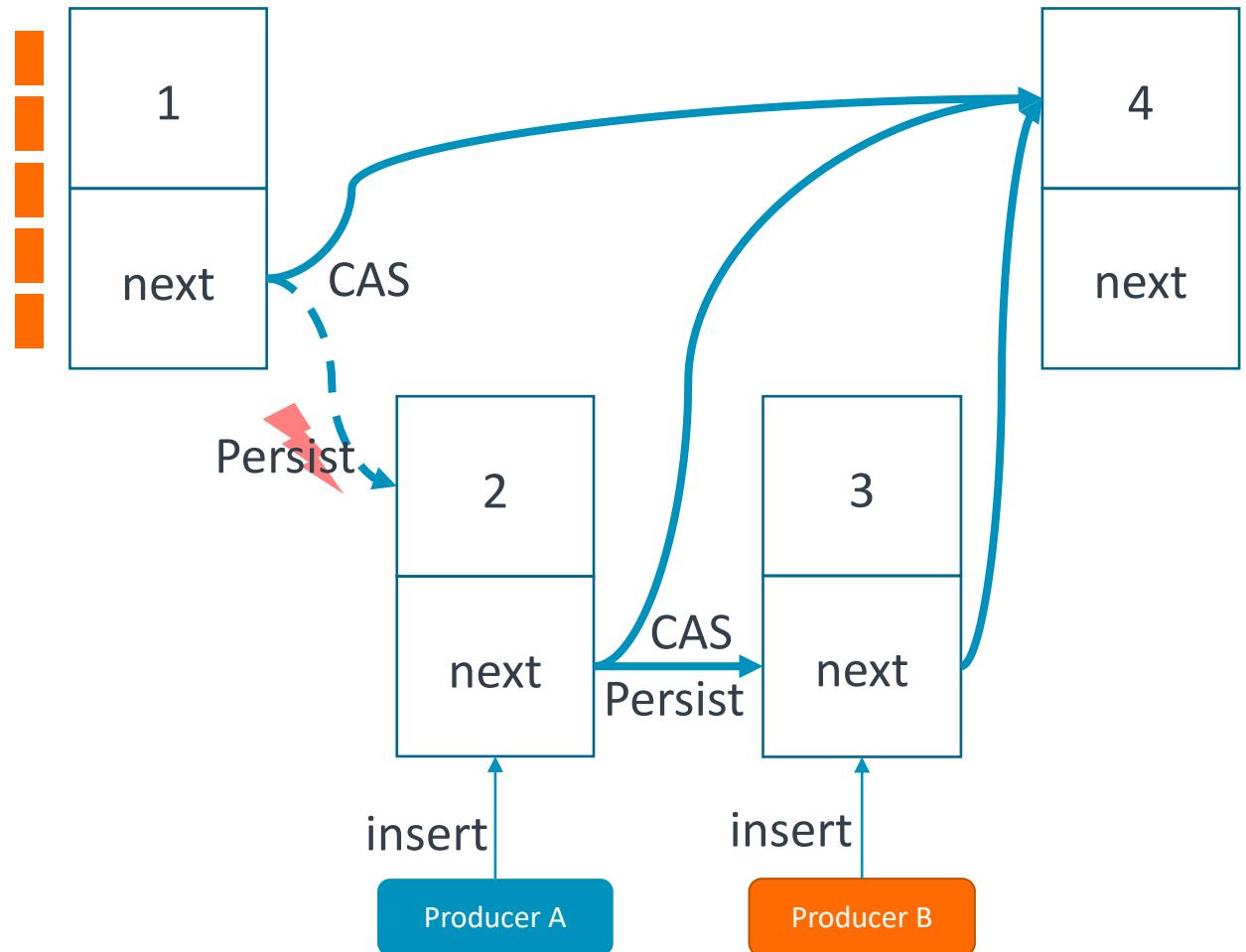
$$M_a^i \leq_v S_a^j \rightarrow M_a^i \leq_p S_a^j$$

[\*] Ref: Memory Persistency, ISCA'14

# Challenge: Data Loss In Concurrent Linked List

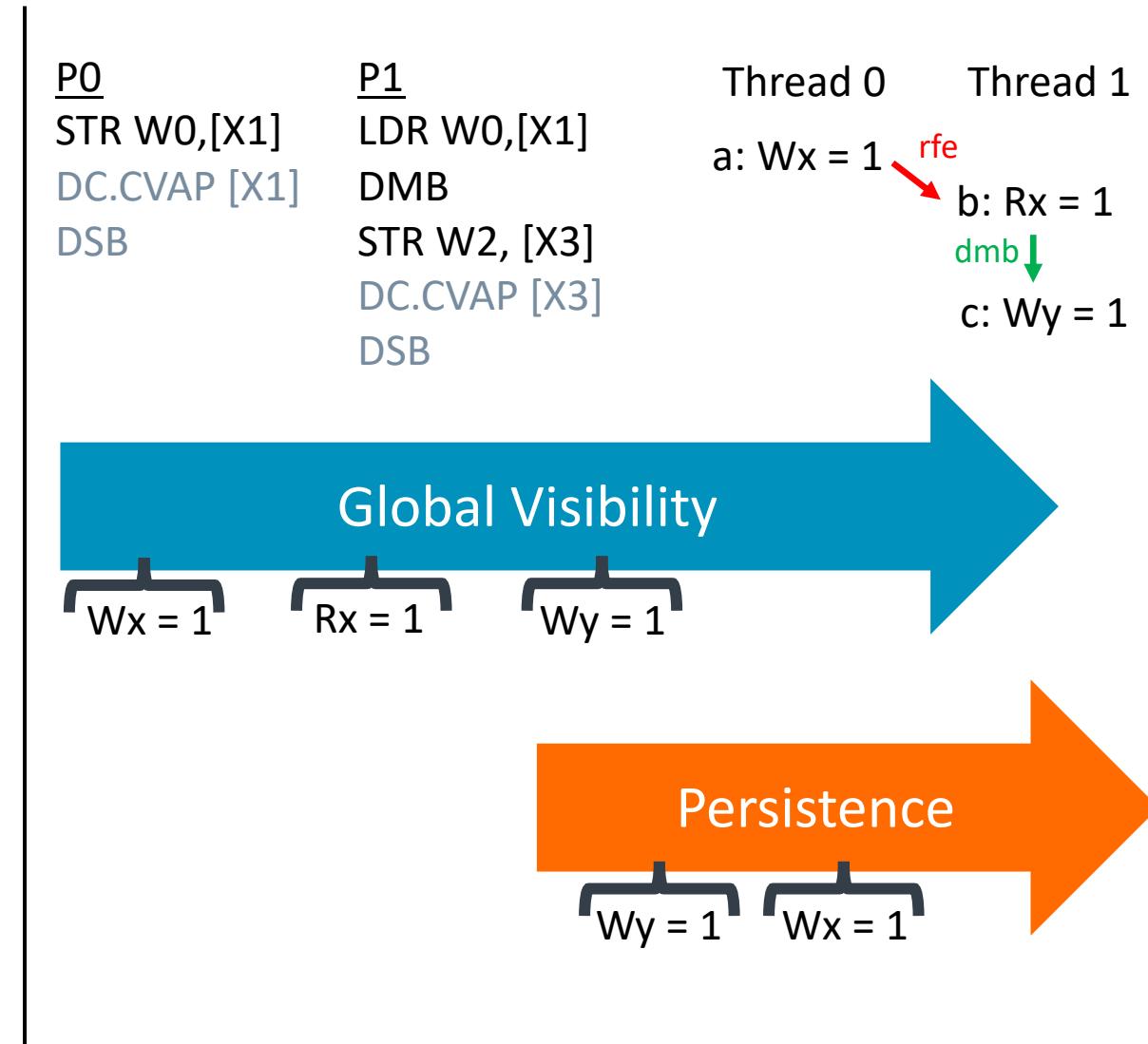
```
1. if(CAS(&last->next, next, node)) {  
2.   Persist(&last->next);  
3.   DSB  
4. }
```

- Producer B observes A's updates, but cannot / does not enforce the persists
- *The inter-thread “read of non-persistent write ” problem*



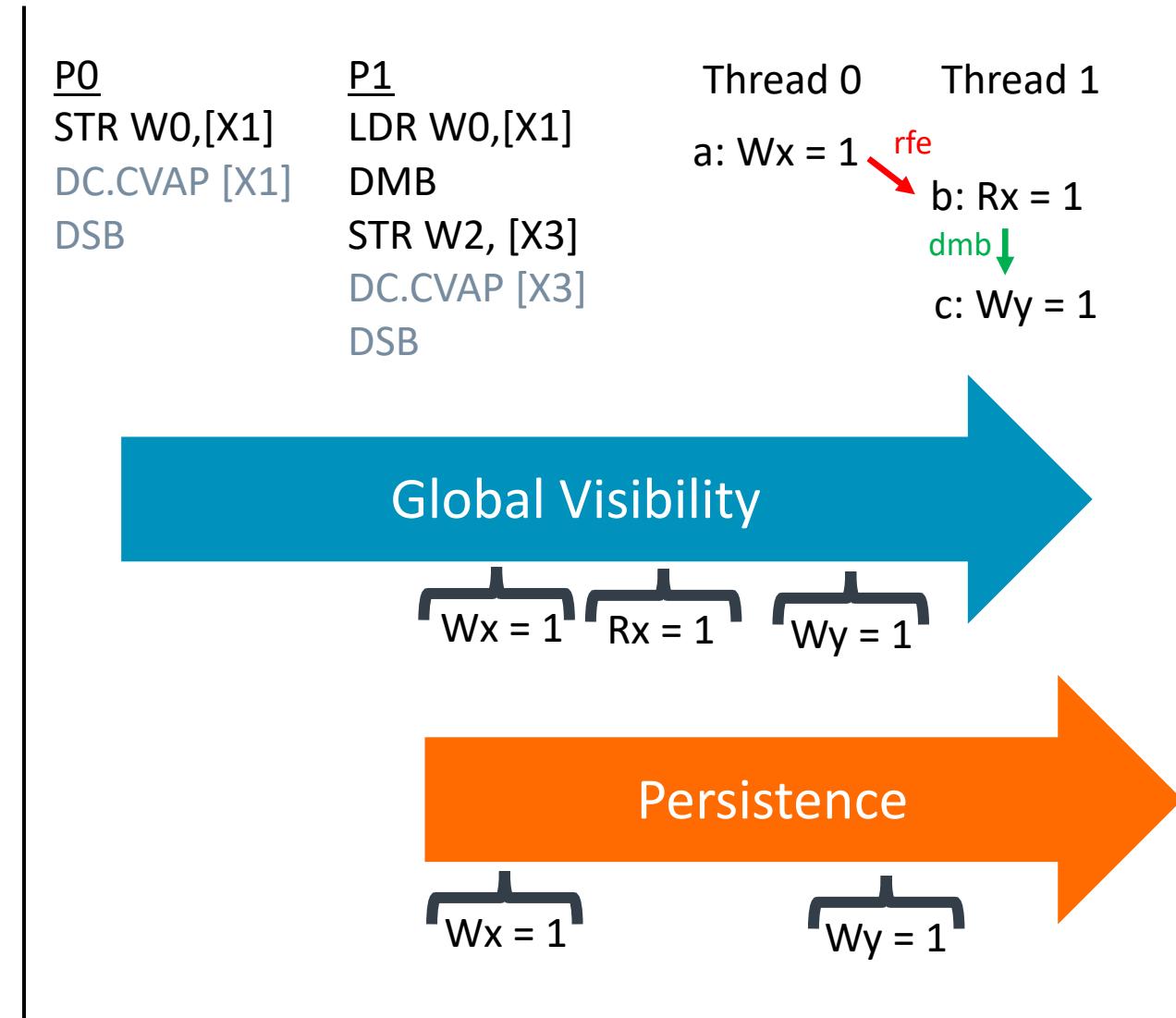
# Solution

- Basic idea: delay consumer's persist operation until producer's persist operation is done
- Various arch options
  - Delay producer's visibility until persistence is done



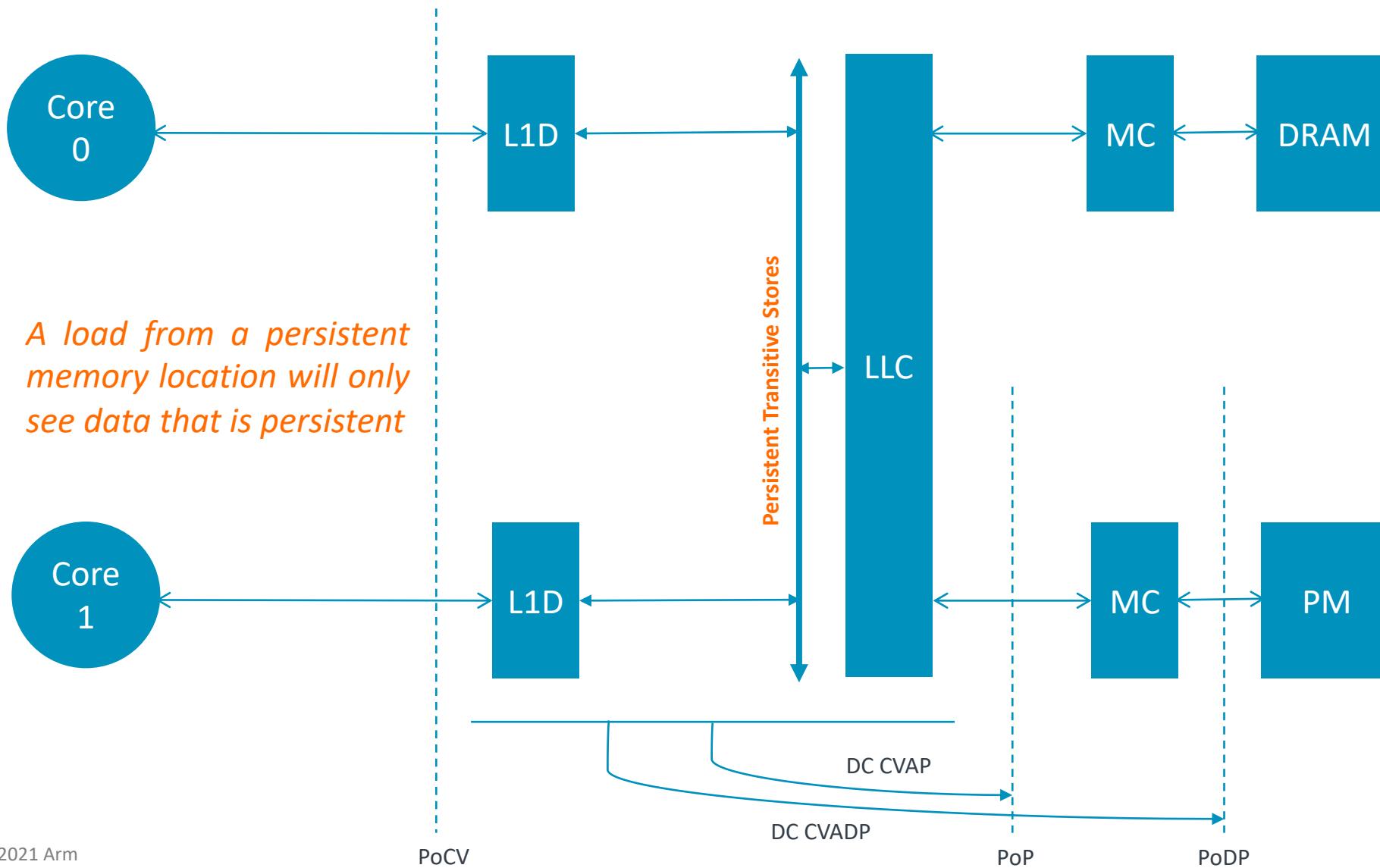
# Solution

- Basic idea: delay consumer's persist operation until producer's persist operation is done
- Various arch options
  - Delay producer's visibility until persistence is done
- New instructions for combining persist and store for synchronizing stores



More info: Persistent Atomics for Implementing Durable Lock-Free Data Structures for Non-Volatile Memory, SPAA'19

# Persistent Transitive Stores to Synchronize Visibility & Persistency



# In Software

- Readers persist all locations read -> bloat, slow
- Tell reader to persist / to wait
- Single out-of-band location -> scalability??
- Multiple out-of-band locations -> hello mini-STM ?!?
- Borrow payload -> steals payload bits

P0

produce(P)

CAS(X ,P, ..)

persist(X)

P1

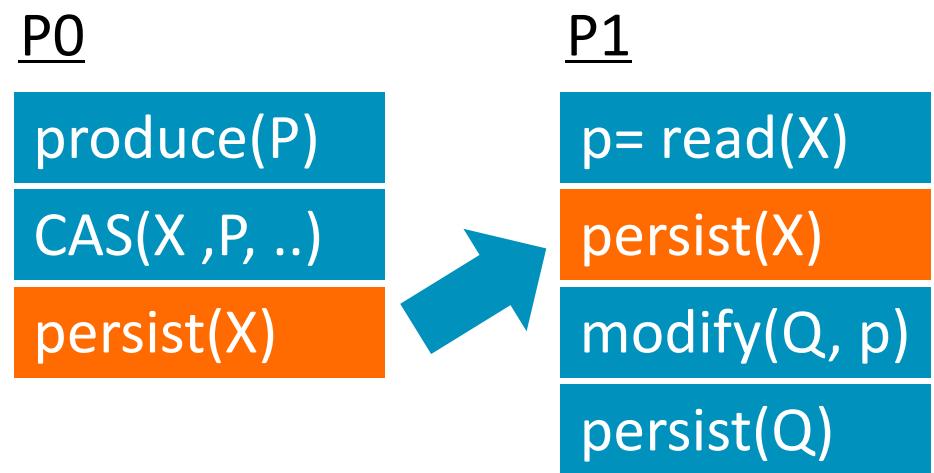
p= read(X)

modify(Q, p)

persist(Q)

# In Software

- Readers persist all locations read -> bloat, slow
- Tell reader to persist / to wait
- Single out-of-band location -> scalability??
- Multiple out-of-band locations -> hello mini-STM ?!?
- Borrow payload -> steals payload bits

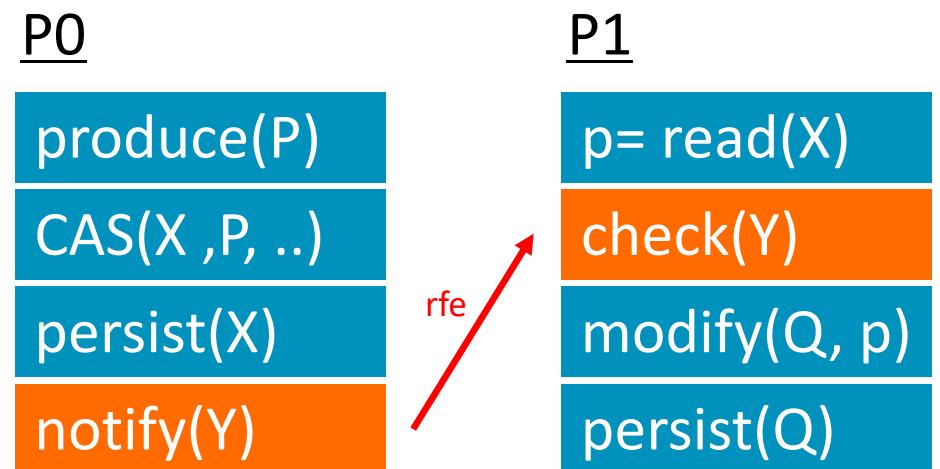


# In Software

- Readers persist all locations read -> bloat, slow
- Tell reader to persist / to wait
- Single out-of-band location -> scalability??
- Multiple out-of-band locations -> hello mini-STM ?!?
- Borrow payload -> steals payload bits

*"There is always a software way around the problem if you are aware of it, but that is not a reliable solution. The best solution is to design processors so that a load from a persistent memory location will only see data that is persistent."*

- Mario Wolczko and Bill Bridge (Oracle)

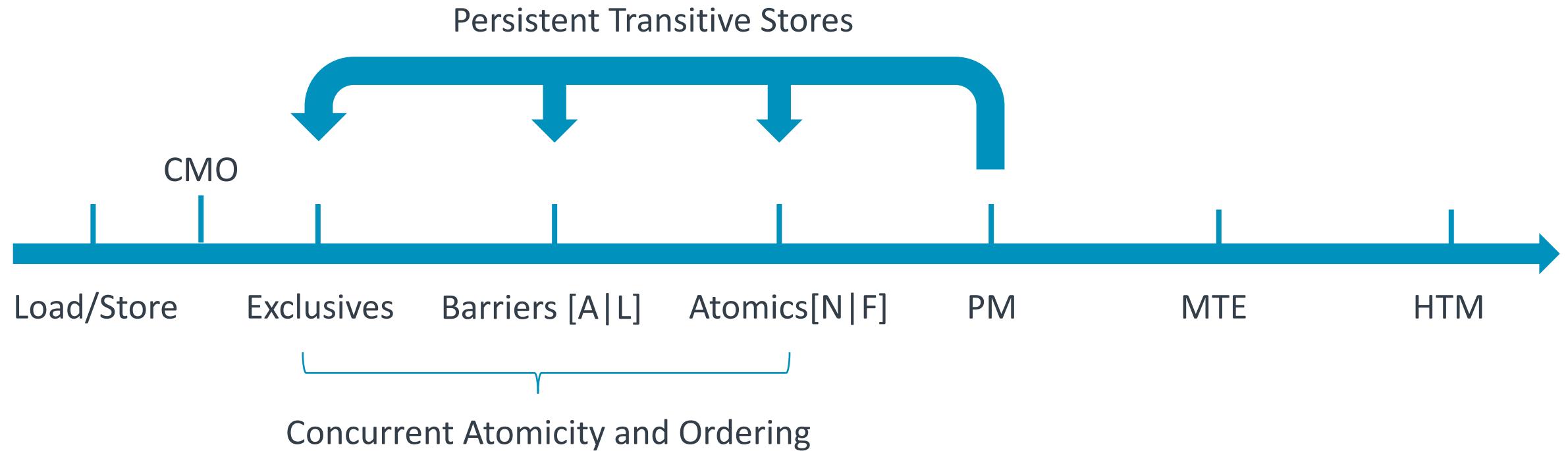


Source: <https://medium.com/@mwolczko/non-volatile-memory-and-java-part-2-c15954c04e11>

# Summary: Persistent Transitive Stores

- Persistent memory introduces a new level of reasoning
- Arm ISA extensions for flushing to *point of (deep) persistence*: DC CVA[D]P
  - Armv8.2-A DC CVAP, Armv8.5-A DC CVADP
- Simple persist operations do not allow transitive ordering of persists
- Tricky case closing store of lock-free section
- Extending the ISA (and µarch) to synchronize *visibility and persist* orders

# Architectural Support for Memory



# Use Cases for Persistent Atomics

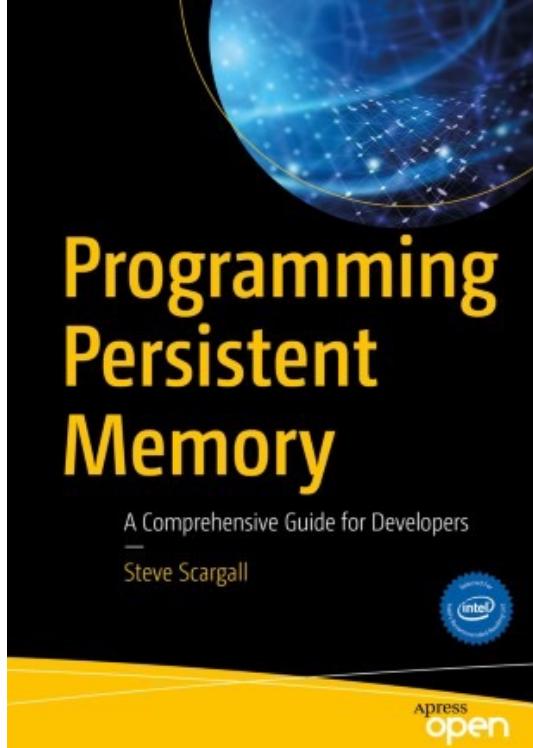
- Lock-free data structures for filesystems, databases, k-v stores, and caching tiers

Data Structures	Example Implementations	Applications
B+ Trees	BZTree, and Crab-tree, Masstree, noveLSM, FAST-FAIR B+-Tree, WORT, FPTree, NV-tree, WB+-Tree, B+-Tree, CDDS B-Tree	<b>Filesystems and databases:</b> Microsoft Hekaton, HANA, Timesten, SQLite, LevelDB/RocksDB/Cassandra (LSM Tree), NOVA, ext4-DAX
Hashmaps	NVC-hashmap, CCEH, LevelHashing, Dali, PFHT	<b>Key-value stores:</b> Redis, Memcached, Pelikan, Tair
Queues	LogQueue	<b>Persistent log queues:</b> Oracle DB, SQL server
Skiplists	NV-skiplist	<b>Databases and KV stores:</b> MemSQL

- Synchronization primitives in languages, libraries, runtimes and compilers for PM

Software Stacks	Synchronization Primitives	Examples
Applications	Locks, lock-free atomics, STM	MySQL, Tomcat, Nginx (sync intensive)
Runtimes	<ul style="list-style-type: none"><li>Interpret language functions to runtime builtin implementations</li><li>Concurrent GC in runtime implementations</li></ul>	<ul style="list-style-type: none"><li><i>Synchronized</i> in Java to intrinsic lock or monitor lock</li><li>v8, OpenJDK, go-runtime</li></ul>
Kernels	spinlock, ticket spinlock, mcs queued spinlock, clh queued spinlock mutex, semaphore, reader-writer lock, read-copy-update	Linux kernel
Languages	Locks and atomics: Java, C11/C++, C#, Golang, JS, NodeJS, WASM; TM: C/C++	<i>Synchronized</i> in Java/C++, <i>lock</i> in C#
Libraries	mutexes, semaphores	pthreads, Windows threads
Compilers	atomics in languages get mapped to compiler builtin implementations	GCC __atomic_Builtins, LLVM __atomic_
ISA	PCAS[A L], PSWP[A L], P[LD ST]ADD[A L]	Persistent atomics

# Concurrency on Persistency Memory : It's Complicated



“ We also explain that atomic operations cannot be used inside a [PMDK] transaction while building lock-free algorithms without transactions. ***This is a very complicated task if your platform does not support eADR.***”

Source: Programming Persistent Memory (Steve Scargall)  
[https://link.springer.com/chapter/10.1007/978-1-4842-4932-1\\_14](https://link.springer.com/chapter/10.1007/978-1-4842-4932-1_14)



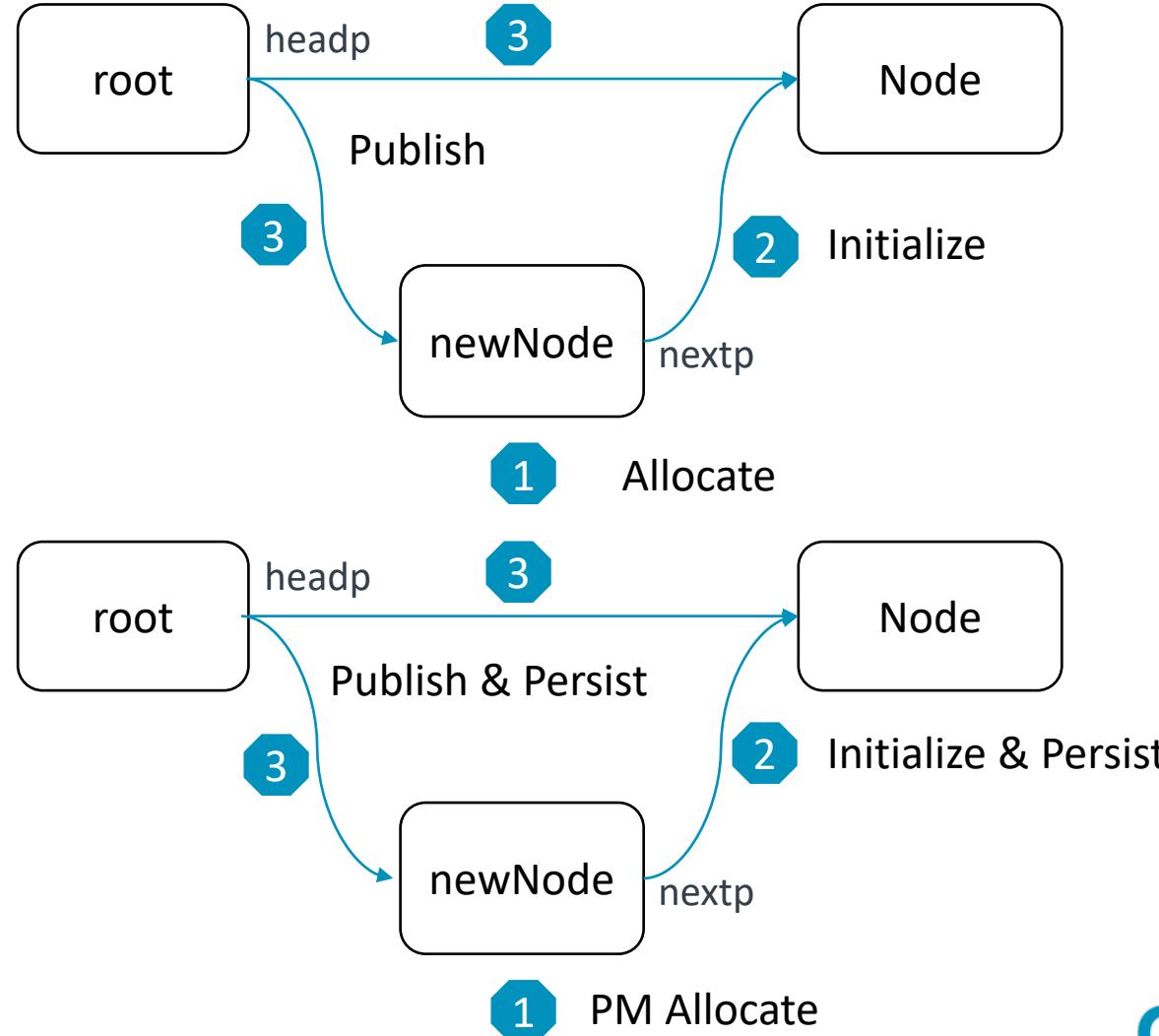
# Memory Consistency

Why should sequential application developers care about memory consistency?

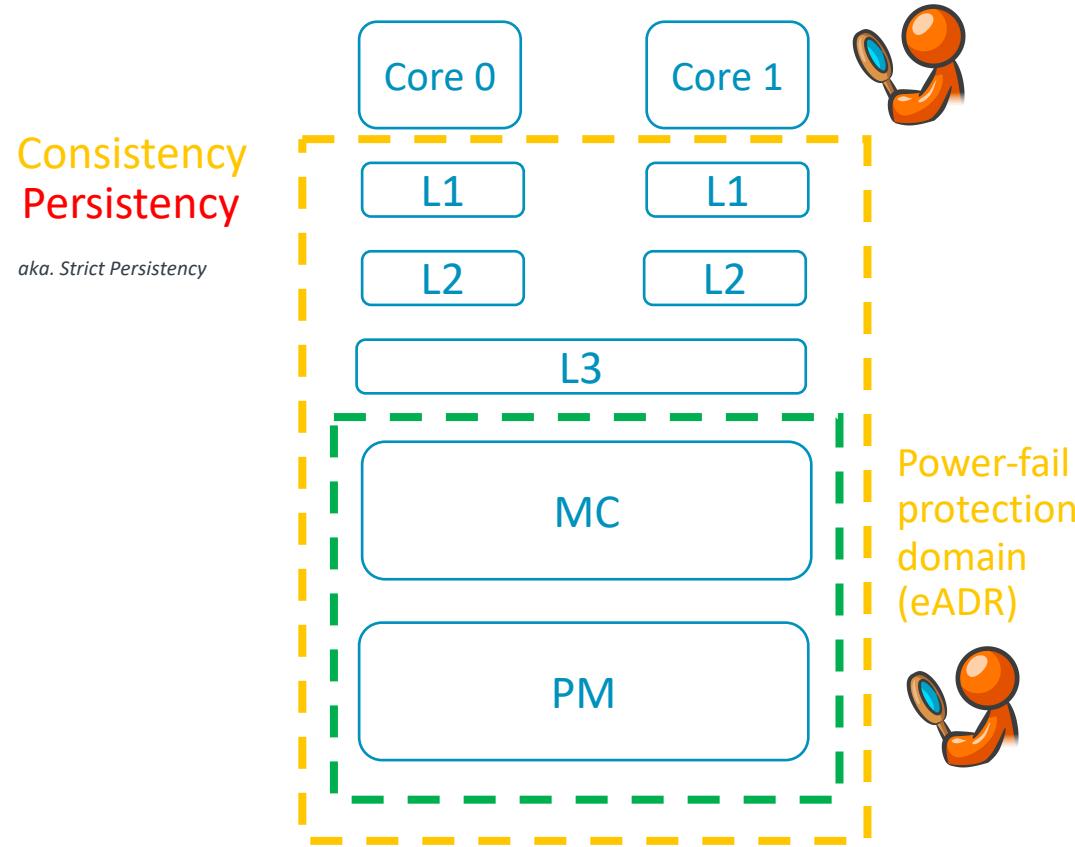
# Example: Adding a Node to a Linked List

```
1 // Add a node to a linked list
2 void
3 addnode(struct root *rootp, int data)
4 {
5     struct node *newnodep;
6     if ((newnodep = malloc(1,
7         sizeof(struct node))) == NULL)
8         fatal("out of memory");
9     newnodep->data = data;
10    newnodep->nextp = rootp->headp;
11    rootp->headp = newnodep;
12 }
```

```
2 void
3 addNode(struct root *rootp, int data)
4 {
5     struct node *newnodep;
6     if((newnodep = pm_malloc(1,
7         sizeof(struct node))) == NULL)
8         fatal("out of memory");
9     newnodep->data = data;
10    newnodep->nextp = rootp->headp;
11    pm_flush(newnodep,
12        sizeof(struct node));
13    pm_fence();
14    rootp->headp=newnodep;
15    pm_flush(newnodep,
16        sizeof(struct node));
17    pm_fence();
18 }
```



# eADR Simplifies Persistent Programming, but Not Sufficient



- CPU cache hierarchy in the power-fail protection domain (PoP)
  - Contents will be saved upon power failure
- Persistency == Consistency
  - Concurrent programs ✓
  - Is that sufficient for sequential programs?
- Globally visible stores in the cache hierarchy will be persistent too
  - No need to DC CVAP
  - No need to use barriers?
    - No, simple sequential programs need to reason about memory consistency

Note: eADR power-fail protection domain can differ due to inclusiveness of the cache hierarchy

# Arm's Weak Memory Model: W->W Reordering Allowed

P0  
str A=1  
str flag=0

P1  
while(flag==1){};  
print A

P1 can read a stale copy of A, as **str flag=0** can be made globally visible before **str A=1**.

Use **DMB.st** (or **stlr**) between the two stores on P0 to serialize the two stores.

```
1 // Add a node to a linked list
2 void
3 addnode(struct root *rootp, int data)
4 {
5     struct node *newnodep;
6     if ((newnodep = malloc(1,
7         sizeof(struct node))) == NULL)
8         fatal("out of memory");
9     newnodep->data = data;
10    newnodep->nextp = rootp->headp;
11    DMB.ST
12    rootp->headp = newnodep;
13    DMB.ST
14 }
```

Can we remove both persist and fences?

Even though caches are in the PoP, no need to **PERSIST**, but **FENCES** are still needed.

Non-TSO needs the first **DMB.ST** to prevent store reordering.

TSO & non-TSO may need the second **DMB.ST** for global visibility due to store buffering.

# Enforcing Failure Atomicity in Language-Level Persistency Models

## Undo logging for failure atomicity

Bank balance transfer example

```
FASE
{
    Store A;
    Store B;
}
```

Failure atomicity w. Armv8.2-A

```
FASE
{
    STORE log-A;
    DCCVAP log-A;
    DMB;
    STORE A;
    DCCVAP A;

    STORE log-B;
    DCCVAP log-B;
    DMB;
    STORE B;
    DCCVAP B;
    DSB;
}
```

Failure atomicity with eADR on Arm

```
FASE
{
    ST/NP log-A;
    STLR A;

    ST/NP log-B;
    STLR B;
}
```

Failure atomicity with eADR+ on Arm

```
FASE
{
    STORE log-A;
    STORE A;
    STORE log-B;
    STORE B;
}
```

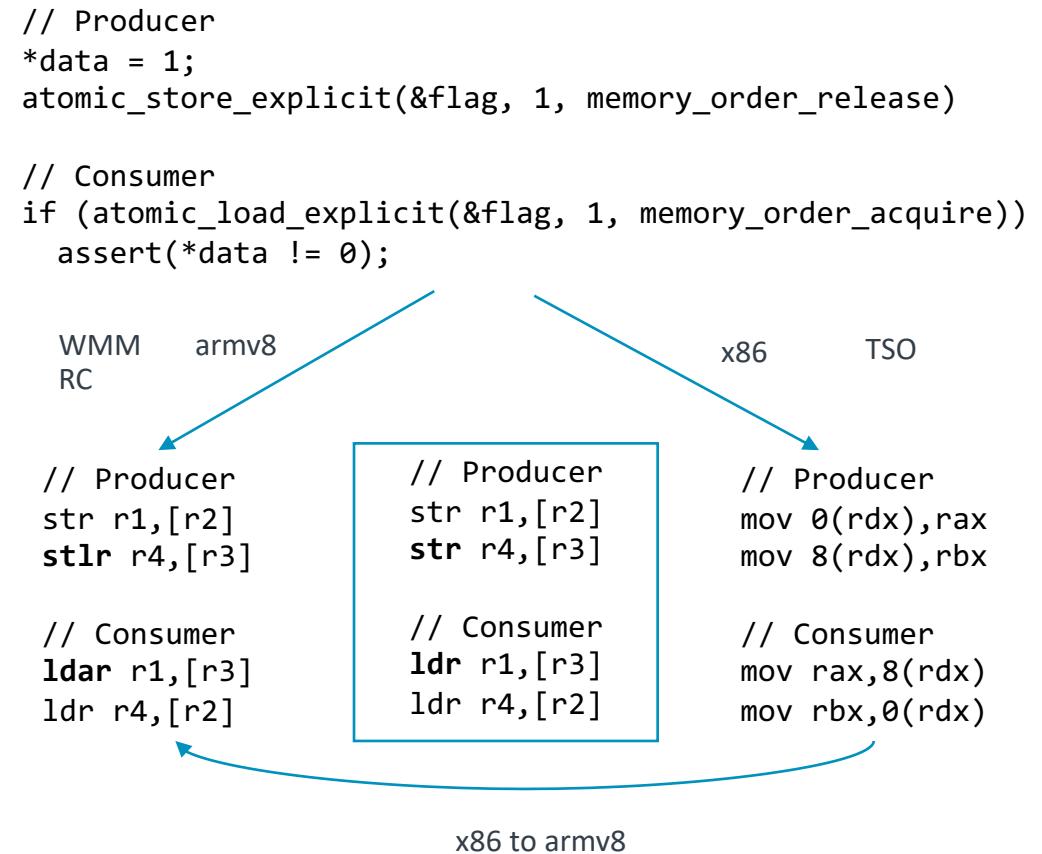
Despite compilers can instrument, barriers are expensive.

Can we remove barriers?

# Software Porting from TSO to WMM

Barriers are hard to get right

- DBT (x86->Arm)
  - Need to add fences (STLR/LDAR, DMB)
    - Hard problem to identify all cases, if not overusing
- Applications porting from TSO -> WMM
  - Recompile, if w. language-level consistency model
  - Add fences (STLR/LDAR, DMB), if not
    - Tedious, easy to overuse or underuse barriers
- Silicon can support TSO and WMM
  - Set a register to get TSO dynamically
  - So the code in the middle would run okay

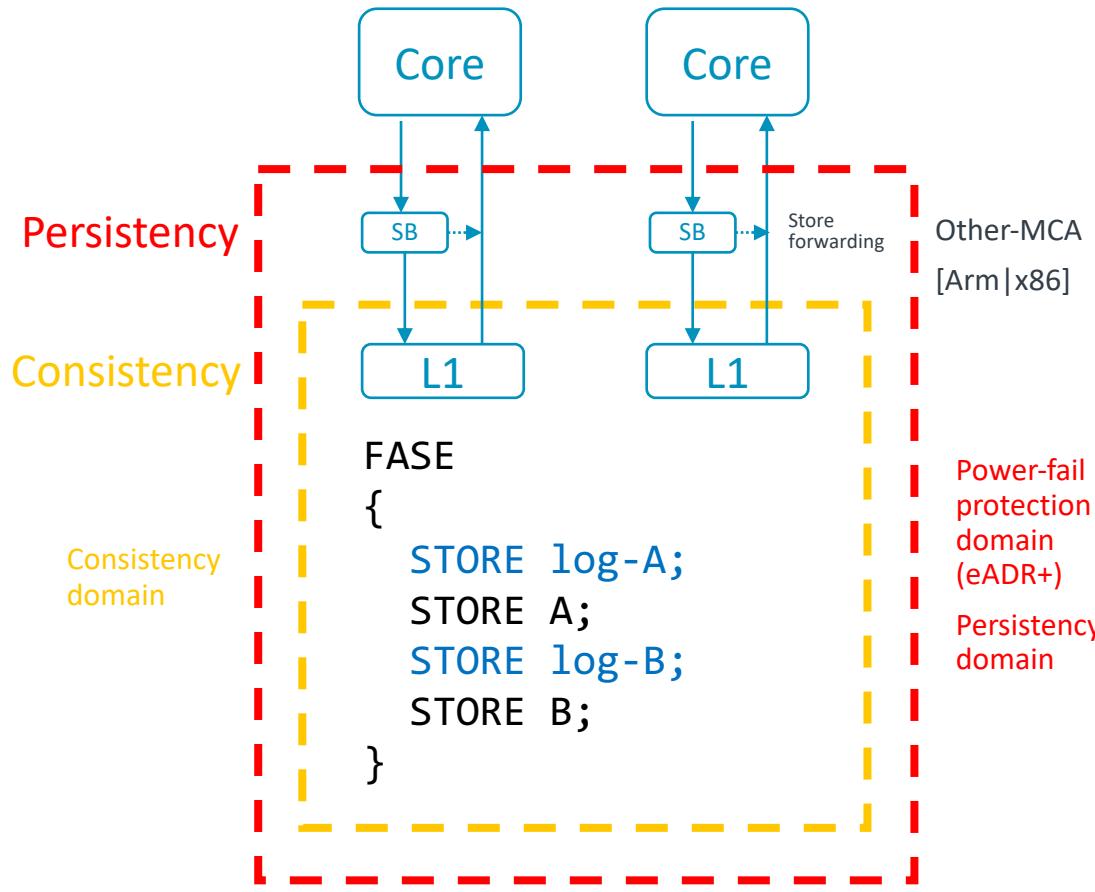


A compiler targeting either architecture directly would produce correct code. However, binary translation that does not account for differences in consistency models would lead to the invalid outcome becoming observable!

DBT needs to insert fences, otherwise tricky bugs get introduced.  
Or, processors support TSO as well.



# Extending Power-fail Protection to Store Buffers

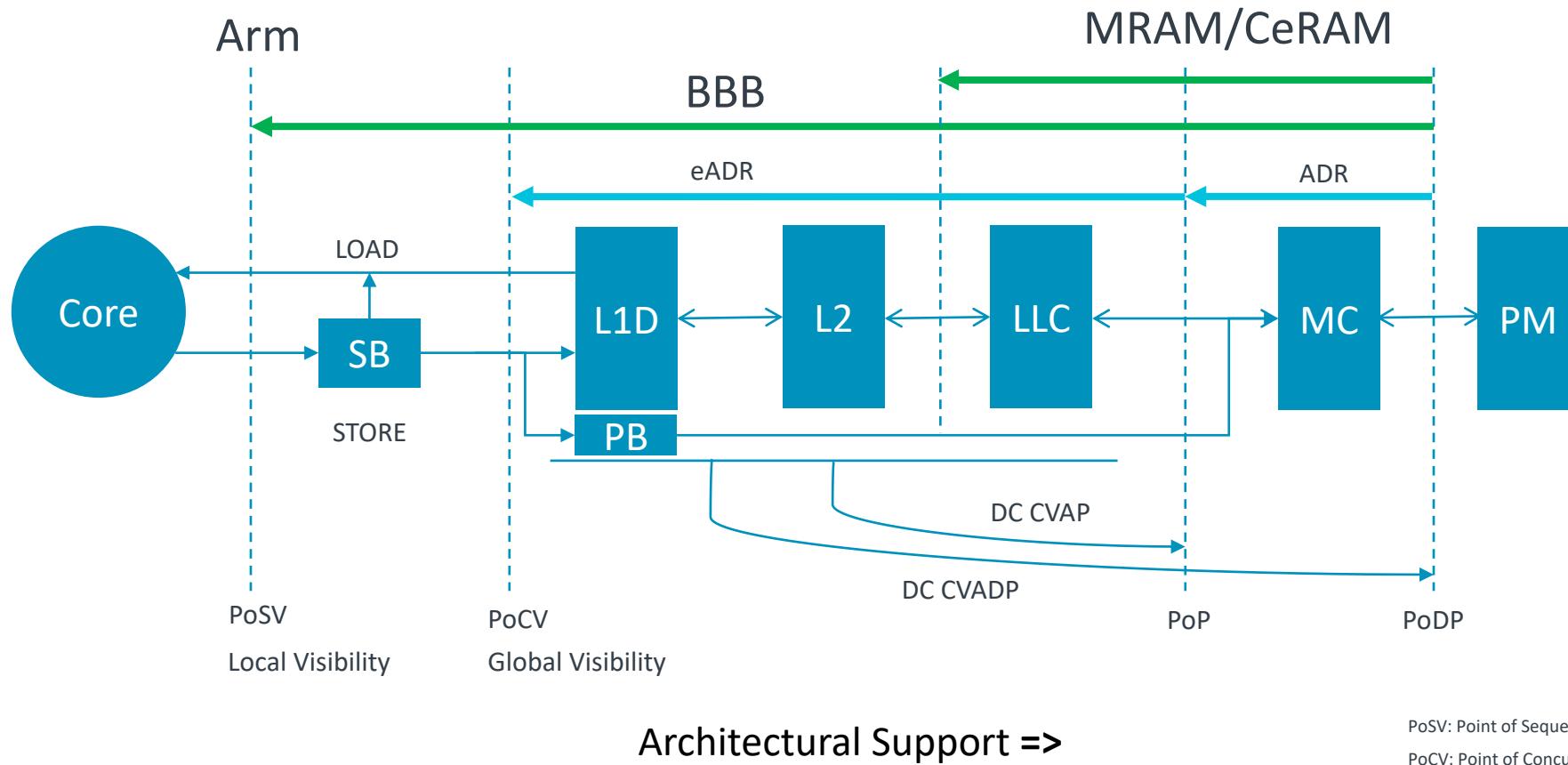


Note: For simplicity of illustration, store buffer may include other buffers on the store path in between core and L1D, e.g., merge buffer

- CPU store buffers in the power-fail protection domain (PoP) too
  - Contents will be saved to PoP
- Stores are executed OoO but committed in order
  - No need to order w. barriers explicitly
- **Consistency == Persistence**
  - Concurrent programs ✓
- **Persistence > Consistency (ahead)**
  - Persistence at SB
    - WMM stores get persisted in order, despite can be made visible OoO, barriers would have already been needed for concurrency so okay.
  - Sequential programs continue to execute correctly without CPU barriers
    - Language support may be needed to prevent compiler reordering

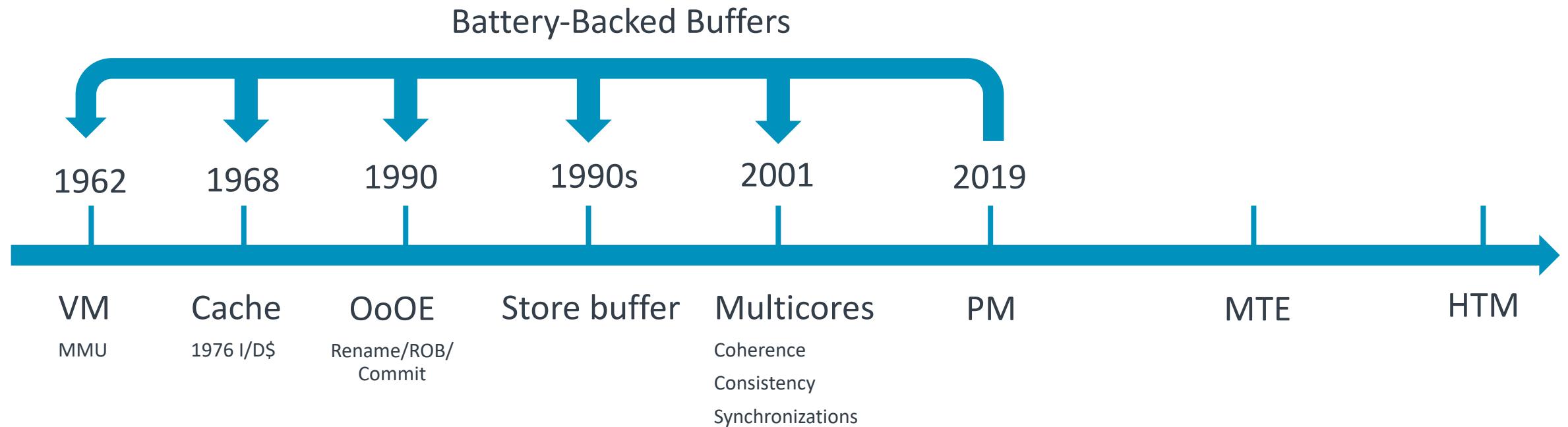
# Microarchitectural Support to Sync Visibility & Persistency: BBB

<= Microarchitectural Support



PoSV: Point of Sequential/Local Visibility  
PoCV: Point of Concurrent/Global Visibility  
PoP: Point of Persistence  
PoDP: Point of Deep Persistence  
BBB: Battery-Backed Buffers

# CPU Microarchitectural Support for Memory



# Summary: Battery-Backed Buffers

- Battery-backed buffers, instead of the on-chip cache hierarchy
  - Reduce energy, by two orders of magnitude vs. eADR
  - Improve performance and simplify programming vs. v8.2
    - both DC CVAP and DSB can be eliminated
- Sequential persistency, in addition to strict persistency
  - Persistency == Consistency (strict)
  - Relaxed -> Strict (eADR) -> Sequential (BBB)

Programmability	Sequential Programs		Concurrent Programs	
	DC CVAP	DSB	DC CVAP	DSB
eADR	✓		✓	✓
BBB	✓	✓	✓	✓

Total Energy Cost	Mobile Class	Server Class
eADR	46.5 mJ ( <b>317X</b> of BBB)	550 mJ ( <b>709X</b> of BBB)
BBB [32 entries]	145 μJ	775 μJ

More BBB µarch details in HPCA'21

[\*] <https://community.arm.com/developer/research/b/articles/posts/simplifying-persistent-programming-with-microarchitectural-support>

arm

Summary

# Summary

- Problems
  - Persist ordering across threads
  - Persist ordering within a thread
- Solutions [\*]
  - Persistent transitive stores
  - Battery-backed buffers
- Other challenges

	Persistent transitive stores	Battery-backed buffers
<b>Performance</b>		
Improvement	Small	Big
<b>Programmability</b>		
Concurrency	Yes	Yes
Failure atomicity	No	No
Persist ordering	Yes	Yes
Persistent addressing	No	No
Persistent MM	No	No
Portability	High	Low
<b>Implementation</b>		
ISA architecture	Yes	No
System architecture	No	Yes
Microarchitecture	Yes	Yes
Interconnect	Yes	No
Operating System	No	Yes
Compiler& toolchain	Yes	No

Persist Ordering	Sequential Programs		Concurrent Programs	
	DC CVAP	DSB	DC CVAP	DSB
Persistent transitive stores			✓	✓
Battery-backed buffers	✓	✓	✓	✓

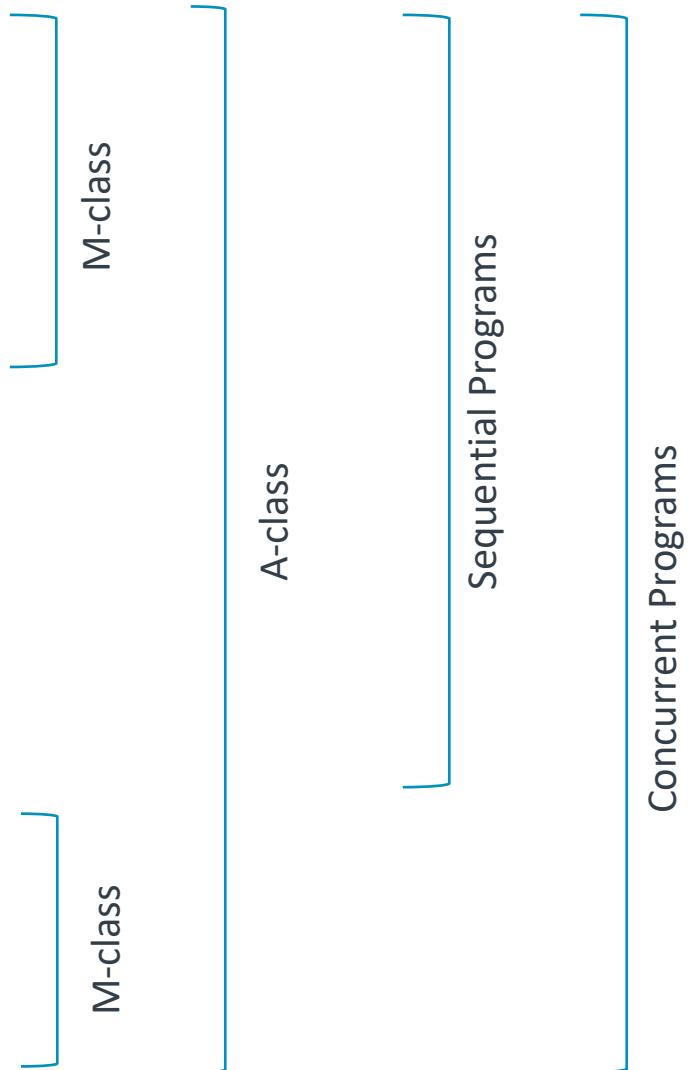
[\*] The solutions are proposals rather than committed Arm architectural features at this stage



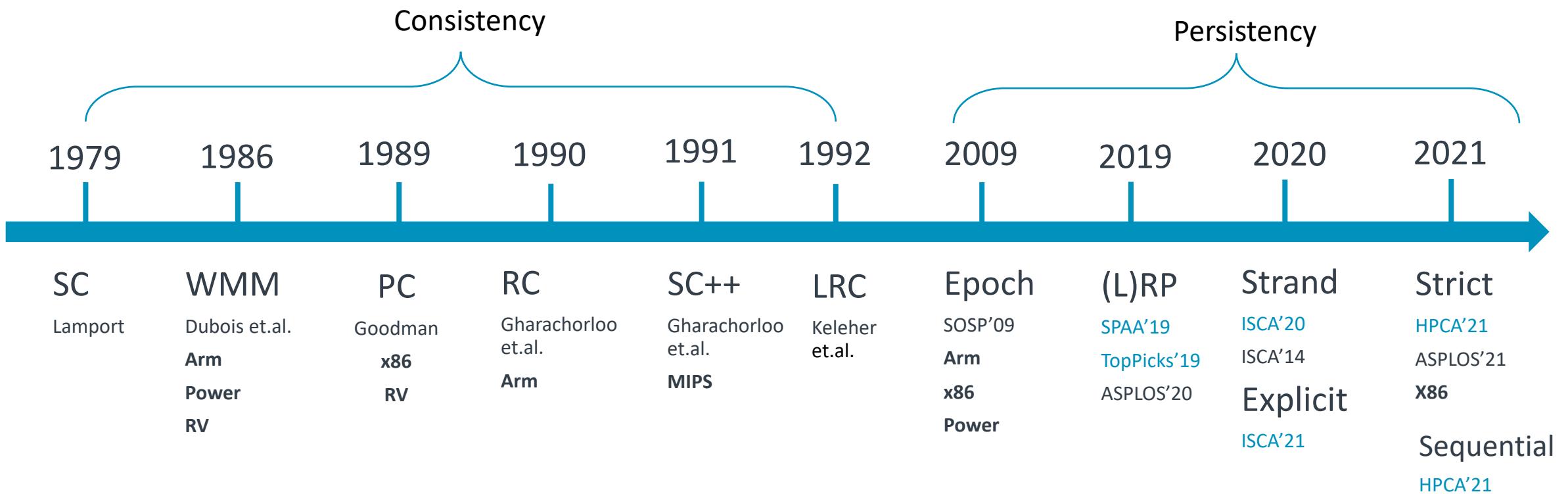
# Other Persistent Memory Programming Challenges

# Persistent Memory Programming Challenges

- Persist ordering
  - Relaxed & strict memory persistency models [arch & uarch]
- Failure atomicity
  - PSTM [sw]
  - HW logging [uarch & arch]
- Persistent addressing
  - Persistent pointers [sw & arch]
  - Pointer swizzling at crash recovery [sw]
- Persistent memory management
  - Metadata crash consistency, GC [sw]
- Concurrency
  - Persistent transitive stores [arch]
  - PHTM/PSTM [uarch/sw]
  - Locking [sw]



# Evolution of Memory Models: Consistency and Persistency



Note: PC differs w. x86-TSO on store atomicity but same ordering.



Thank You

Danke

Gracias

謝謝

ありがとう

Asante

Merci

감사합니다

ধন্যবাদ

Kiitos

شکرًا

ଧନ୍ୟବାଦ

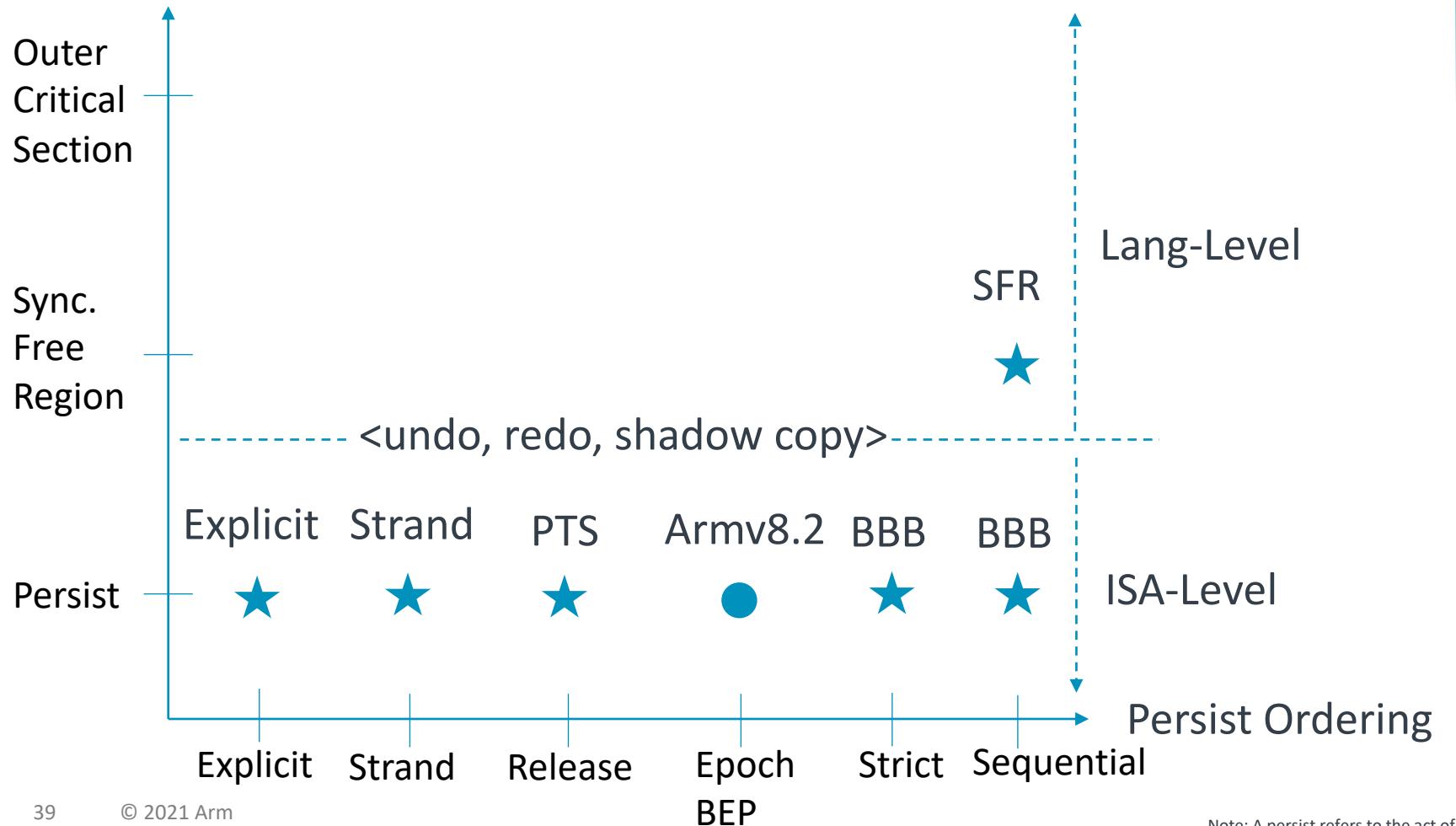
תודה

Thanks Richard Grisenthwaite, Nigel Stephens, Robert Dimond, Stuart Biles, Matt Horsnell, Thomas Grocott, Stephan Diestelhorst, Wendy Elsasser, David Weaver, Nikos Nikoleris, Andreas Sandberg, Joseph Yiu, Rod Crawford, Andrew Sloss, Mitch Ishihara, Dave Rodgman, Gustavo Petri, Jade Alglave, Will Deacon, Alex Waugh, Ola Liljedahl, Magnus Bruce, Stefano Ghiggini, Luca Nassi, Bobby Batacharia, Travis Walton, David Bull, Shidhartha Das, Shiyou Huang, Sivert Sliper, Prakash Ramrakhyani, Mohammad Alshboul, Mike Filippo, Gagan Gupta, Jay Lorch, Bret Toll, Ben Chaffin, Nagi Aboulenein, Guoyun Zhu, Jonathan Halliday, Hans-J. Boehm, Pedro Ramalhete, Virendra Marathe, and Mário Wolczko for their valuable feedback and insightful discussions. Thanks my collaborators Thomas Wenisch, Peter Chen, Satish Narayanasamy, Yan Solihin, James Tuck, Geoff Merrett, Alex Weddell, Boris Grot for very insightful discussions over the years.

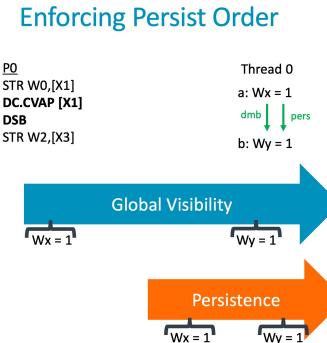
# Memory Persistency Model = Persist Ordering + Persist Atomicity

Memory Consistency Model = Instruction Ordering + Store Atomicity

## Persist Atomicity



## Persist Ordering



## Persist Atomicity

PO  
FASE  
{  
STR W0,[X1]  
STR W2,[X3]  
}

# Memory Persistence Models – by Arm Research & Co.

PUBLICATION	TITLE	LEVEL	PERSIST ORDERING	PERSIST ATOMICITY
IEEE Top Picks'19	<a href="#">Language Support For Memory Persistence</a>	Language	Acquire-Release Persistency	Persist
PLDI'18	<a href="#">Persistence For Synchronization-free Regions</a>	Language	Sequential Persistency	SFR
SPAA'19	<a href="#">Persistent Atomics For Implementing Durable Lock-free Data Structures For Non-volatile Memory</a>	ISA	(Lazy) Release Persistency	Persist
ISCA'20	<a href="#">Relaxed Persist Ordering Using Strand Persistence</a>	ISA	Strand Persistency	Persist
HPCA'21	<a href="#">BBB: Simplifying Persistent Programming Using Battery-backed Buffers</a> <a href="#">Execution Dependence Extension (EDE): ISA Support for Eliminating Fences</a>	ISA	Strict Persistency	Persist
ISCA'21		ISA	Explicit (Buffered) Epoch	Persist
Arm Arm	<a href="#">Armv8.2-A DC CVA[D]P with DMB and DSB</a>	ISA	Persistency	Persist

Note: A persist refers to the act of writing a store durably in persistent memory

# Languages Support for Persistent Memory

Language	Applications	Extensions	Failure atomicity	Addressing	Memory Management	Concurrency
C	Oracle DB, SAP HANA, MS Hekaton, Redis	NVM-Direct (Oracle), PMDK (Intel)	STM	Fat pointers	metadata crash consistent	Locking
C++	SQL Server	PMDK (Intel), STL containers, ATLAS(HPE), Avalon(CMU)	STM CS-based FASE	Fat pointers (P0773R0)	metadata crash consistent	Locking
Go	Kubernetes, Docker, Redis	Go-pmem (VMWare)	STM	Pointer swizzling at recovery	pnew, pmake, GC, heap metadata crash consistent	Locking
Java	Cassandra, ActiveMQ	OpenJDK JEP-352, mashona (RedHat), PCJ (Intel), PJama (Sun)				

More info: William Wang et.al. , Language Support for Memory Persistency, IEEE MICRO Top-picks 2019

# OS Support for PM - Persistent Memory Objects

Addressing Protection, Translation and Persistence Holistically, along w. PM & Capabilities

## Virtual Memory

### Pros

- Same VA to apps, ease of prog.
  - No overlays to move data from storage to memory
- Illusion of large PA to all apps
- VA >> PA w. demand paging

### Cons

- High translation overheads w. TB NVMM
- Page tables won't survive reboots
- Sharing across processes
- Page granularity protection, mapping and migration
- Size of memory no longer a problem

## Single-level Store

### Pros

- Objects survive reboots
- Relocation better than VM
- No MM <-> FS serialization/deserialization
- No fixed page granularity translations
- Inter-process sharing (PGAS)

### Cons?

- Distributed systems (naming)
- Performance? (TLB to VM)

## Example SAS OS

### Single Address Space OS (SAS OS)

- Multics (1965)
- IBM i [OS/400] (1988)
- Opal (1995)
- EROS (2002)
- Twizzler (2020)
- Aurora (2021) ...

*"VM was invented in a time of scarcity, is it still a good idea? We should rethink design decisions based on new realities. Some choices we made may be less relevant, and paths not taken might be more appropriate today."*

- Charles Thacker (Microsoft)

2009 Turing Award Speech at ISCA'2010