



Architectural Support for Persistent Memory

NANDA Workshop

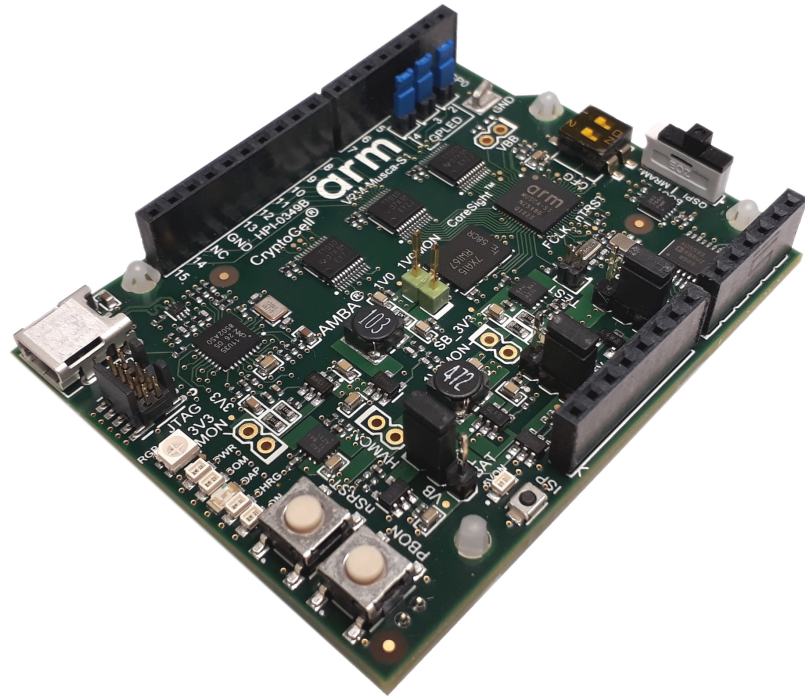
William Wang
6 September 2022

Executive Summary

- NVM uses
 - *More* memory (denser but slower) 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

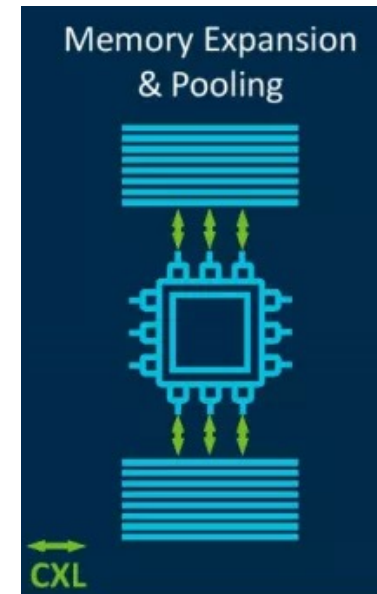
In Embedded, Client, and Infrastructure



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



Arm-based Nokia Asha Smartphone with Micron PCM in 2012



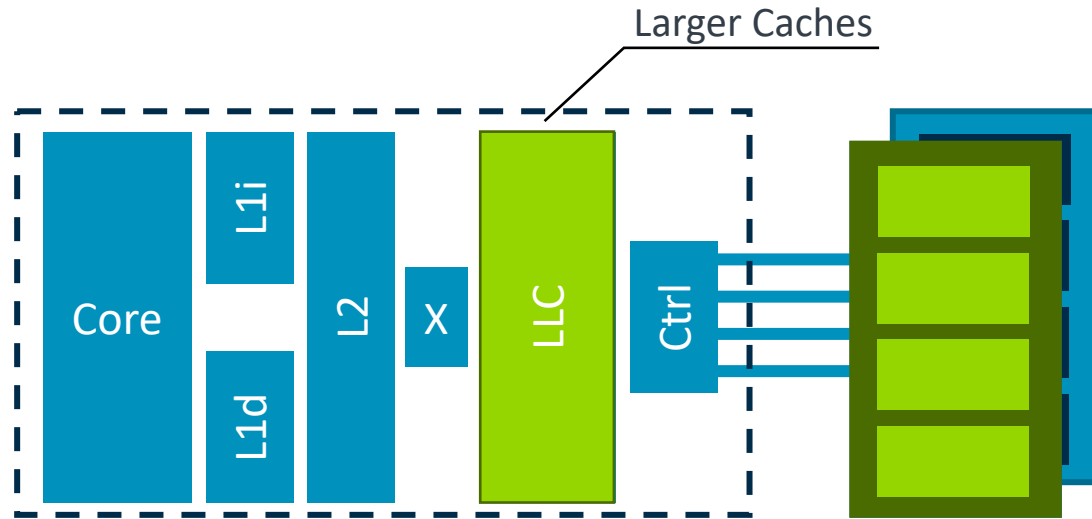
CXL Connected Persistent Memory in Infrastructure

Non-Volatile Memory Opportunities

On-chip Usage

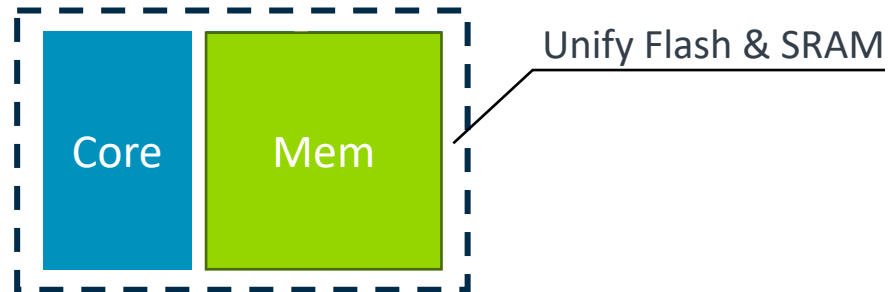
Off-chip Usage

Application-profile
(servers, phones, ..)

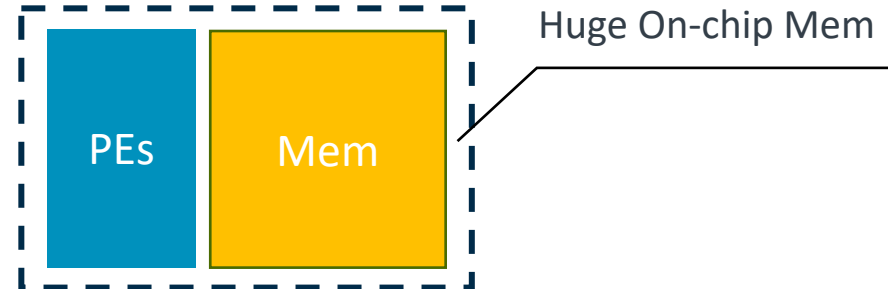


Larger & cheaper DRAM
Ultra-fast Storage
Converged Memory & Storage

Embedded-profile
(energy harvesters)



ASIC
(AI accelerators)



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



In-mem DBs, then other apps

- Data analytics
- ML training
- Finance
- HPC

Tomorrow

Persistent Mem

Non-Volatile



Persistency

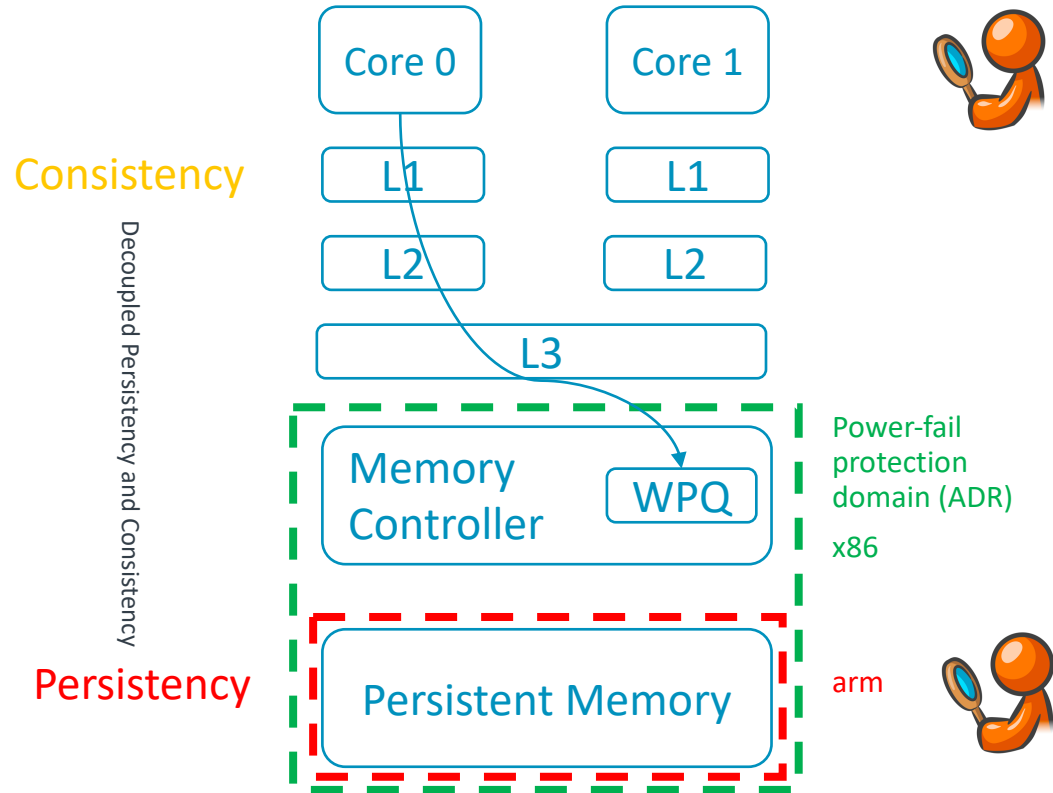
- Failure atomicity
- Persist ordering
- Persistent addressing
- Crash recovery
- Programming models
- ISA & uarch support



Memory Persistency

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

System Assumption



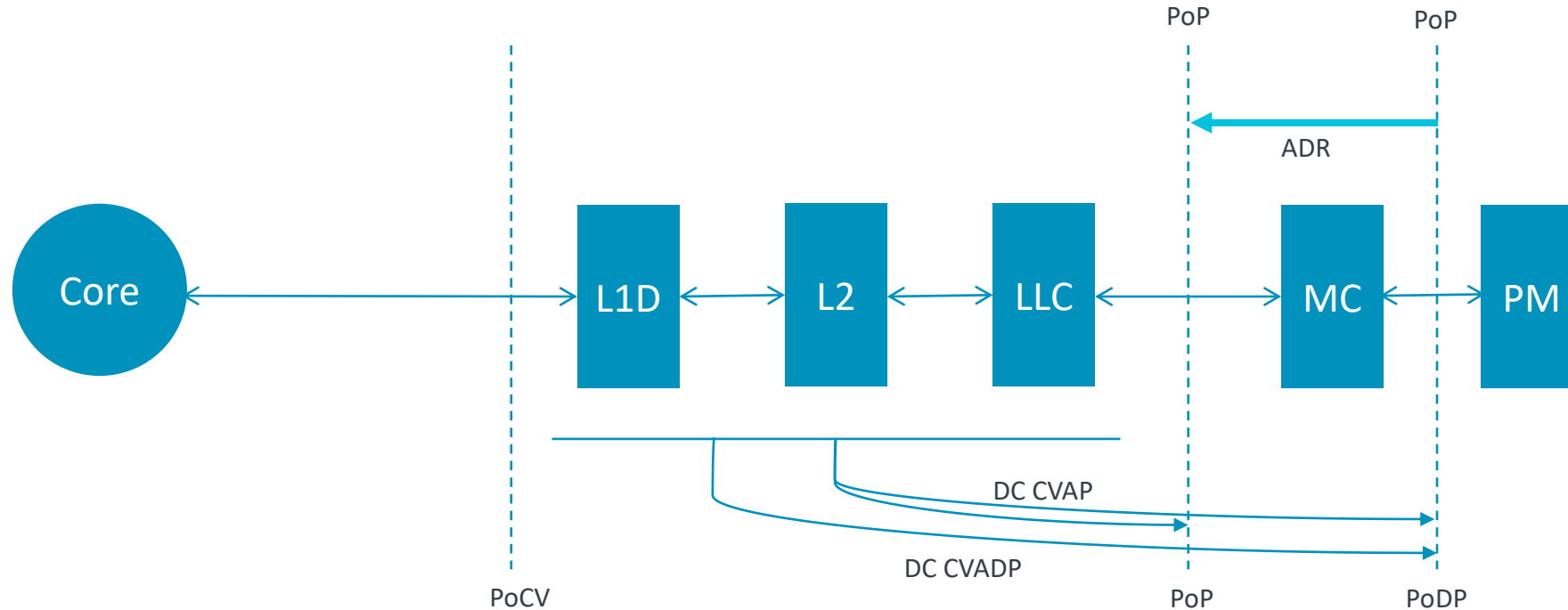
- Point of Persistence (PoP) at the persistent memory module or the memory controller WPQ
 - Contents in the power-fail protection domain will be saved upon power failure
- Caches and cores are still in the volatile domain
 - Contents will be lost upon power failure
- Persistency < Consistency (behind)
 - 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

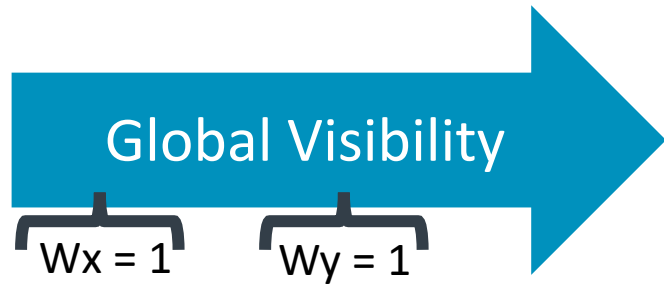
DMB: Data Memory Barrier



Global Visibility Order

PO
STR W0,[X]
STR W1,[Y]

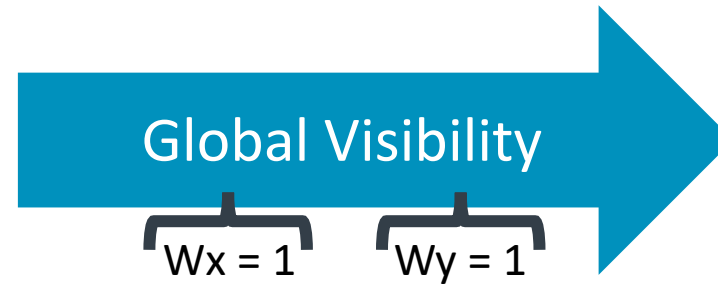
Thread 0
a: Wx = 1
po ↓
b: Wy = 1



time

PO
STR W0,[X]
DMB.ST
STR W1,[Y]

Thread 0
a: Wx = 1
dmb ↓
b: Wy = 1



time

DMB: Data Memory Barrier
DMB.ST: Store barrier



View of the NVM: Persist Order

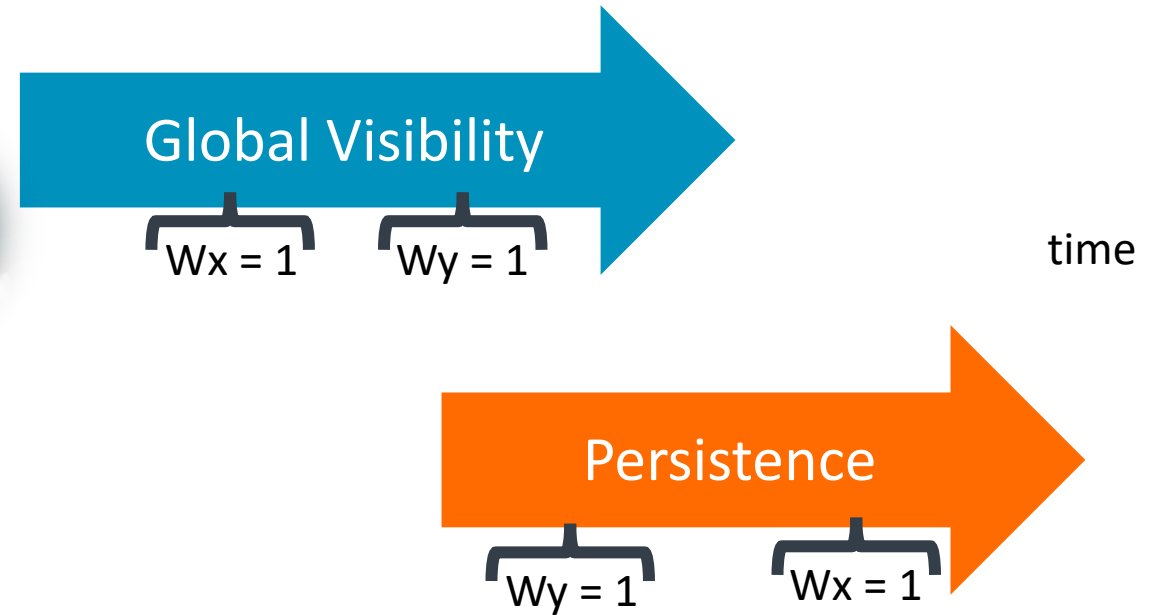
PO
STR W0,[X]
STR W1,[Y]

Thread 0
a: Wx = 1
po ↓
b: Wy = 1

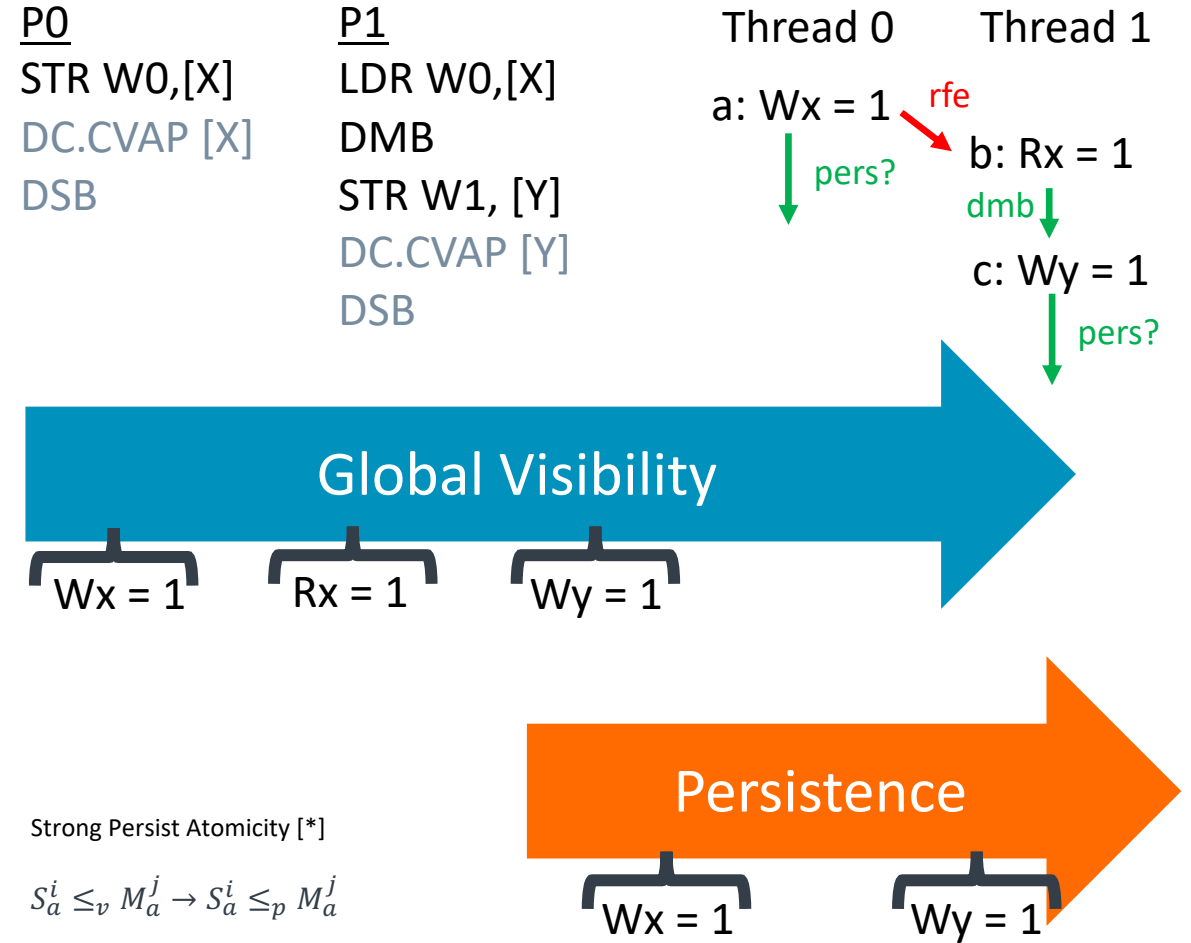
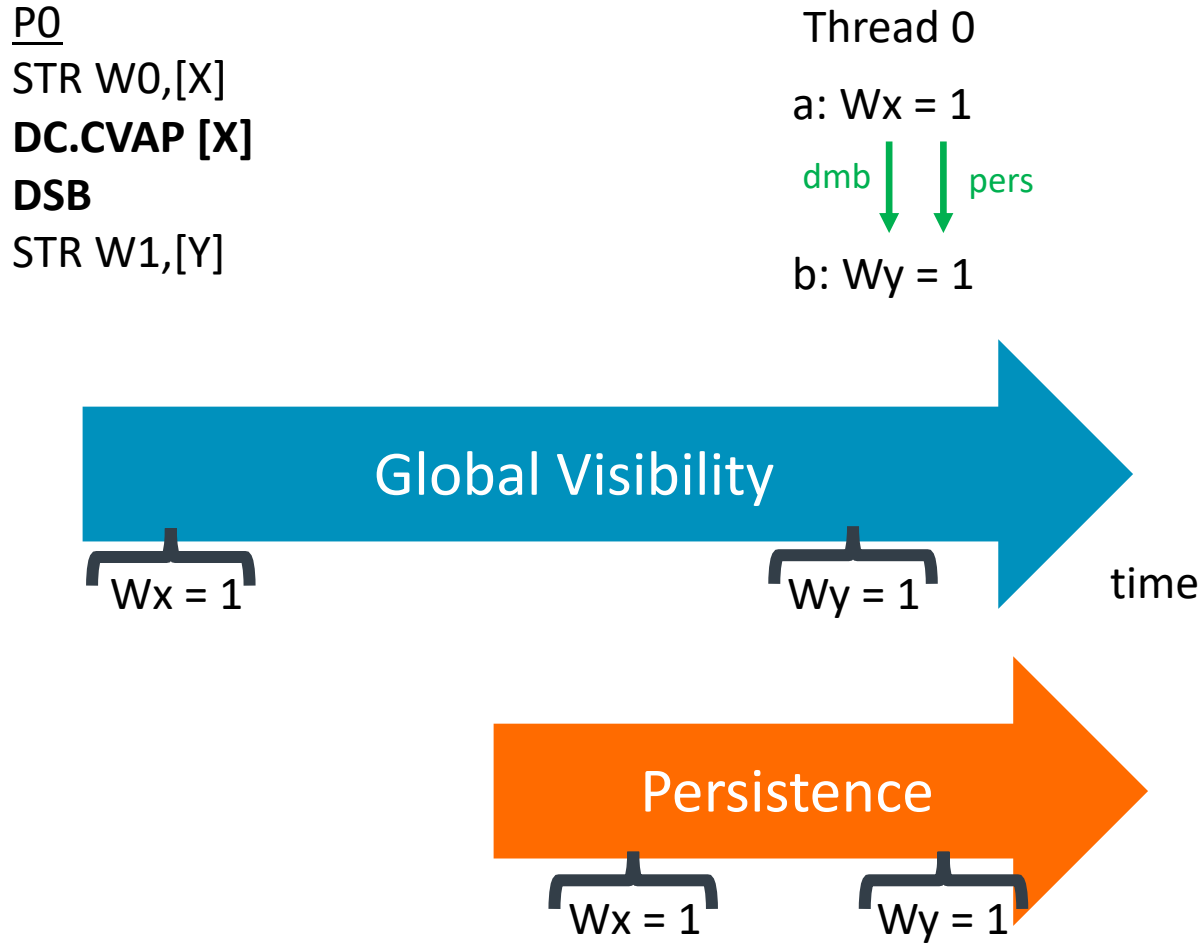


PO
STR W0,[X]
DMB.ST
STR W1,[Y]

Thread 0
a: Wx = 1
dmb ↓
b: Wy = 1



Enforcing Persist Order



Strong Persist Atomicity [*]

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

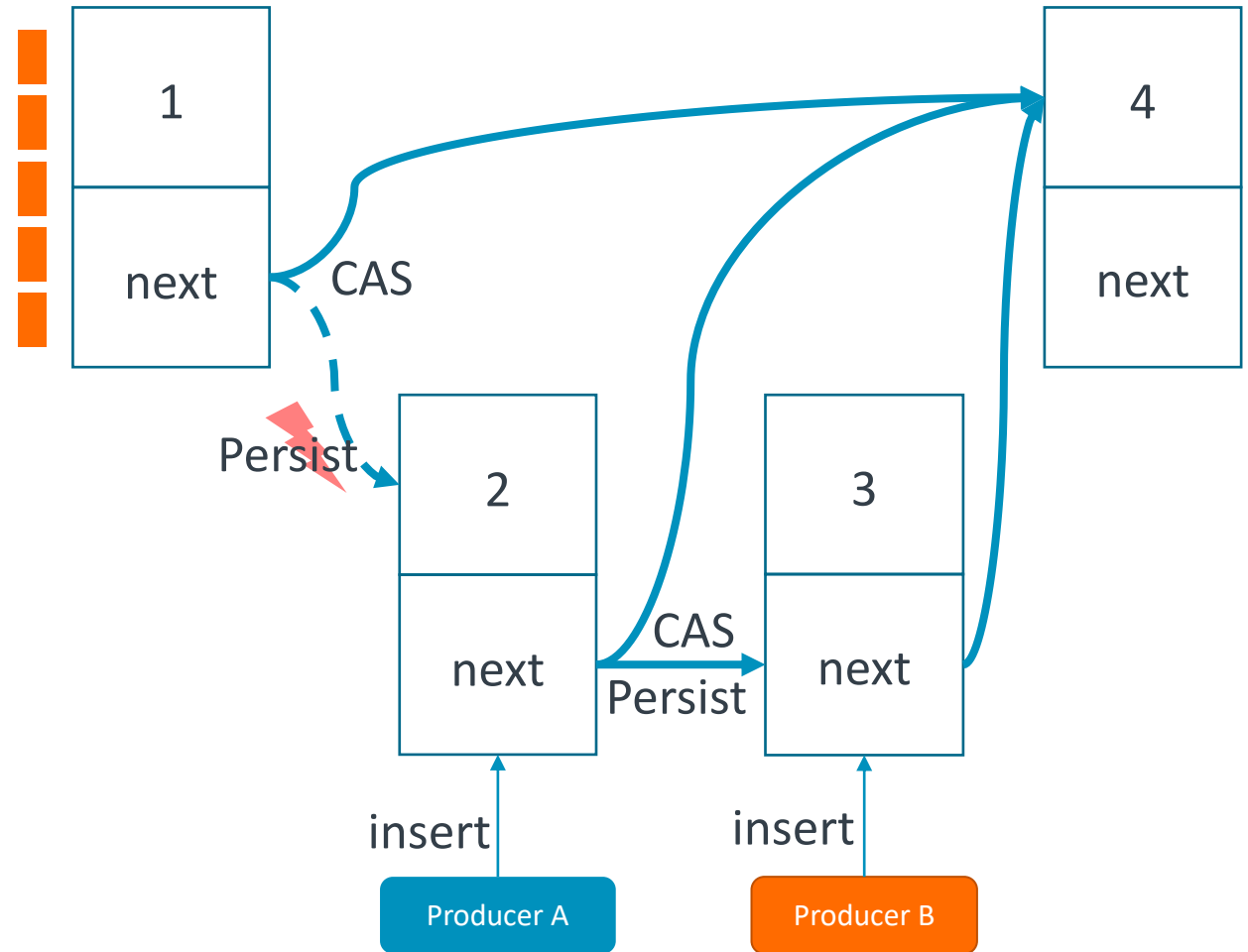
$$M_a^i \leq_v S_a^j \rightarrow M_a^i \leq_p S_a^j$$

[*] Ref: Memory Persistency, ISCA'14

Challenge: Data Loss In Concurrent Linked List

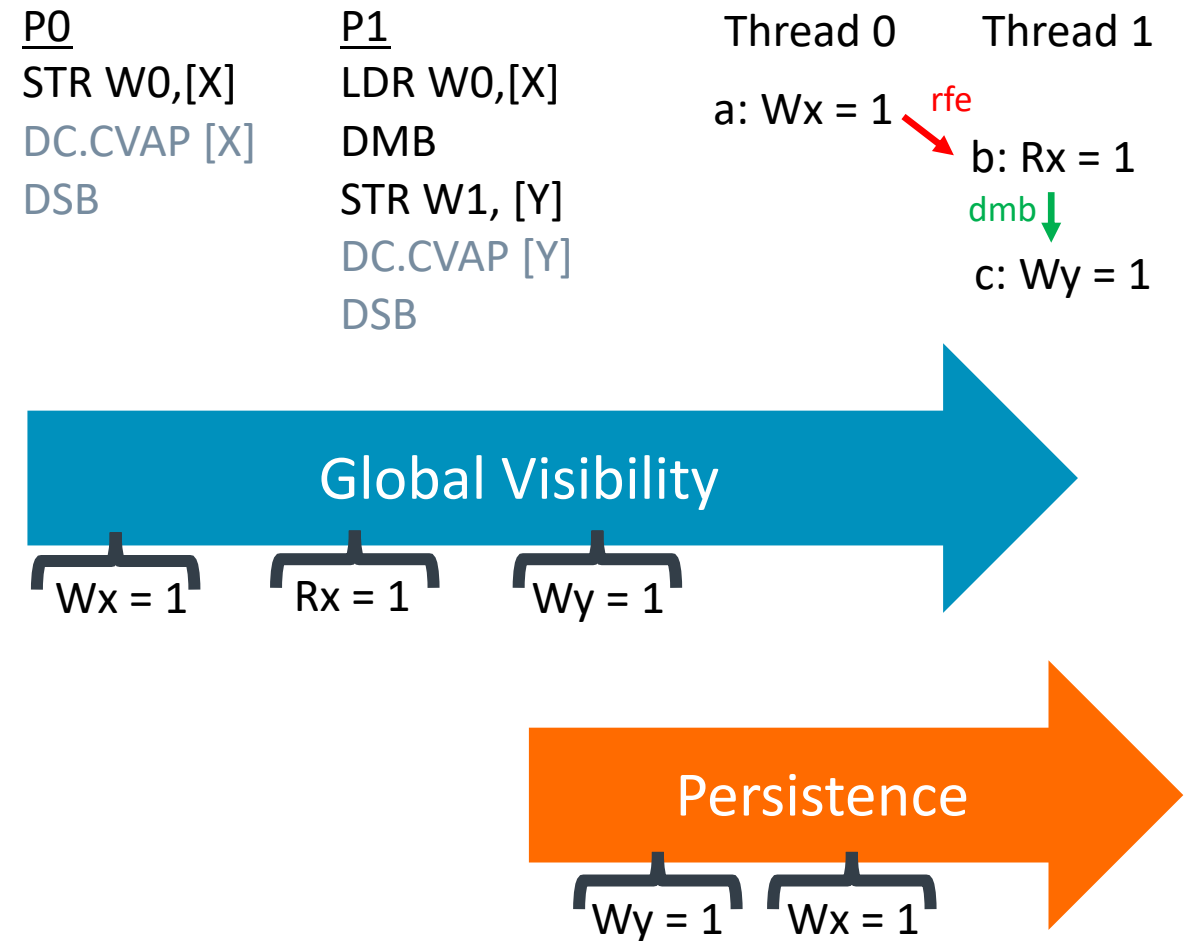
```
1. if(CAS(&last->next, next, node)) {
2.   Persist(&last->next);
3.   DSB
4. }
```

- 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



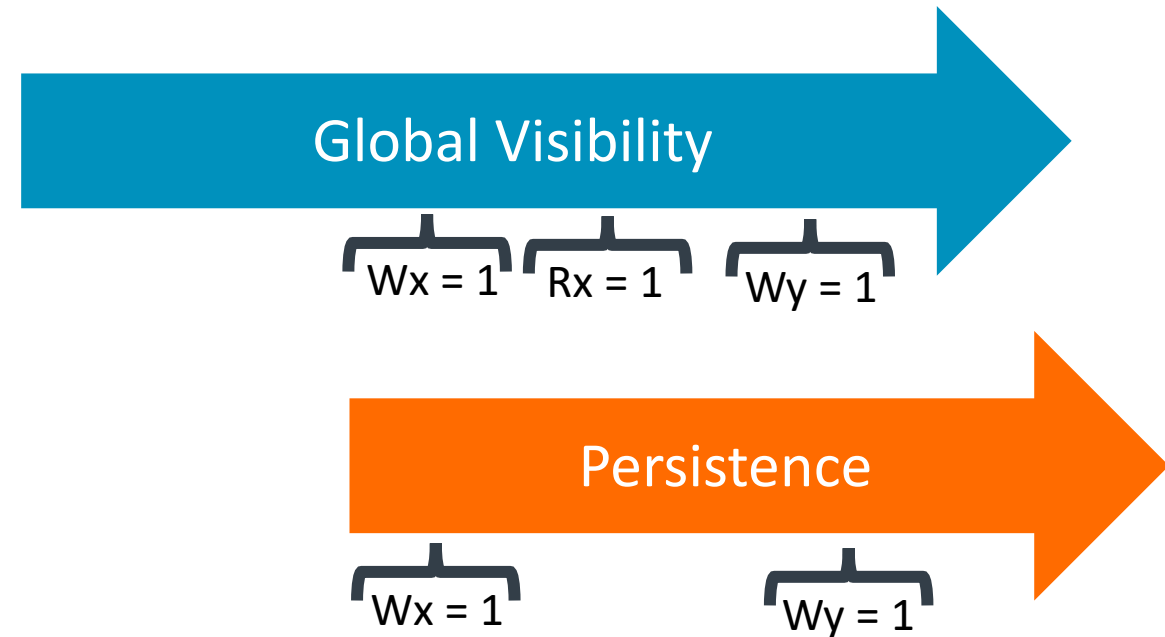
Solution

- 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
 - The persist operation can be done lazily till data is requested across threads

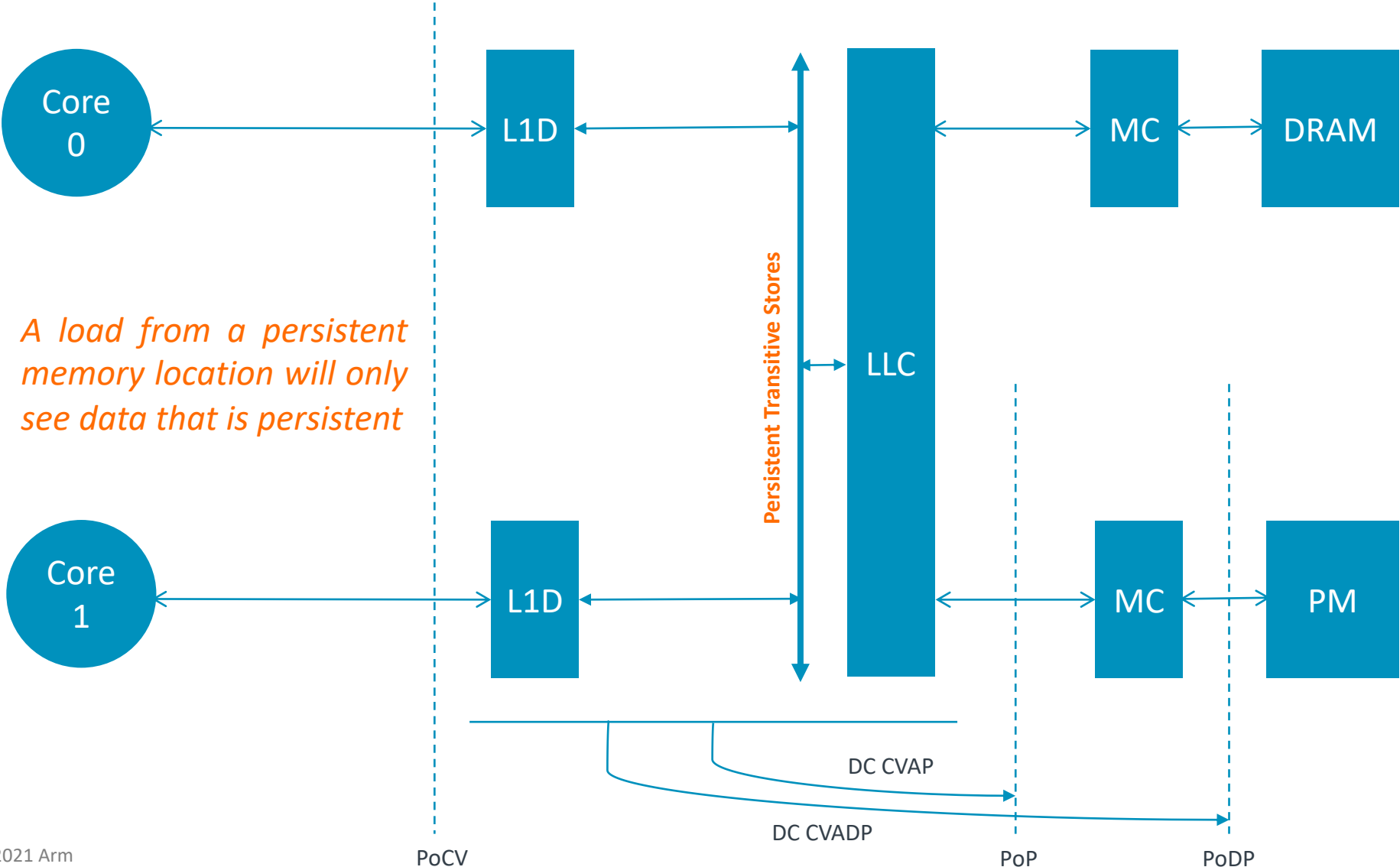
P0
 STR W0,[X]
 DC.CVAP [X]
 DSB

P1
 LDR W0,[X]
 DMB
 STR W1, [Y]
 DC.CVAP [Y]
 DSB

Thread 0 Thread 1
 a: Wx = 1 b: Rx = 1
 ↓ rfe ↓ dmb
 c: Wy = 1



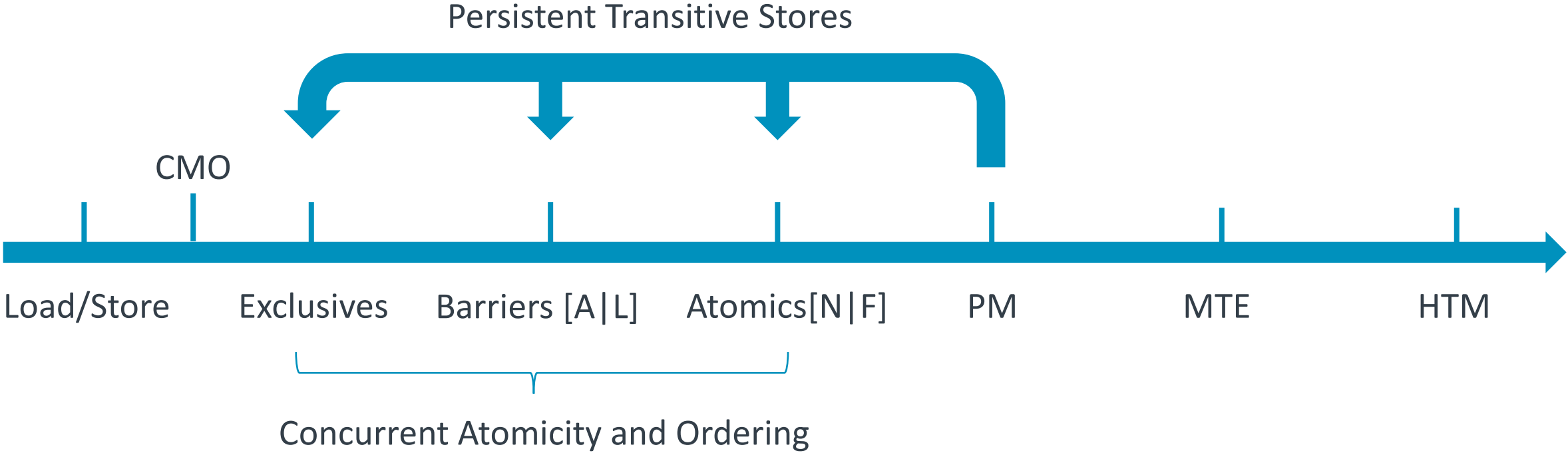
Persistent Transitive Stores to Synchronize Visibility & Persistency



PoCV: Point of Concurrent Visibility
 PoP: Point of Persistence
 PoDP: Point of Deep Persistence



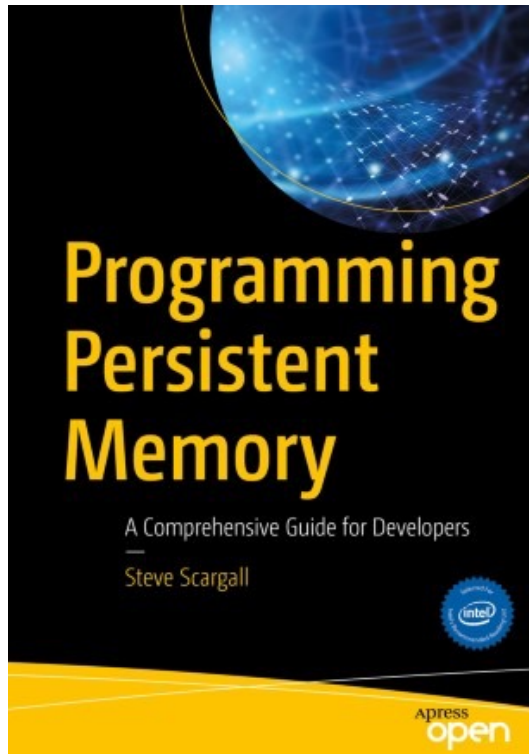
Architectural Support for Memory: Persistent Transitive Stores



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 μ arch) to synchronize *visibility and persist* orders

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



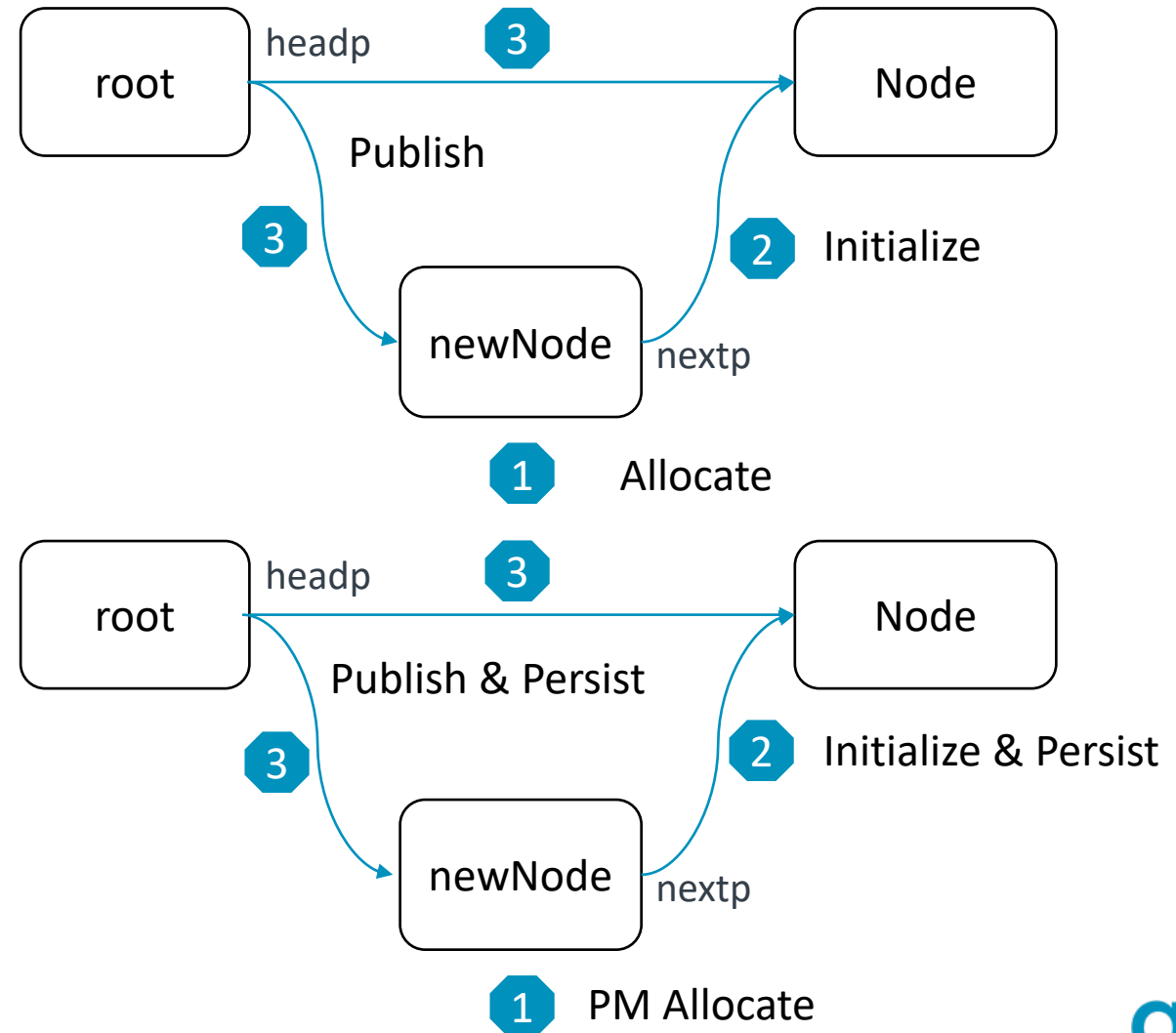
Memory Consistency

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 = calloc(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 }
```

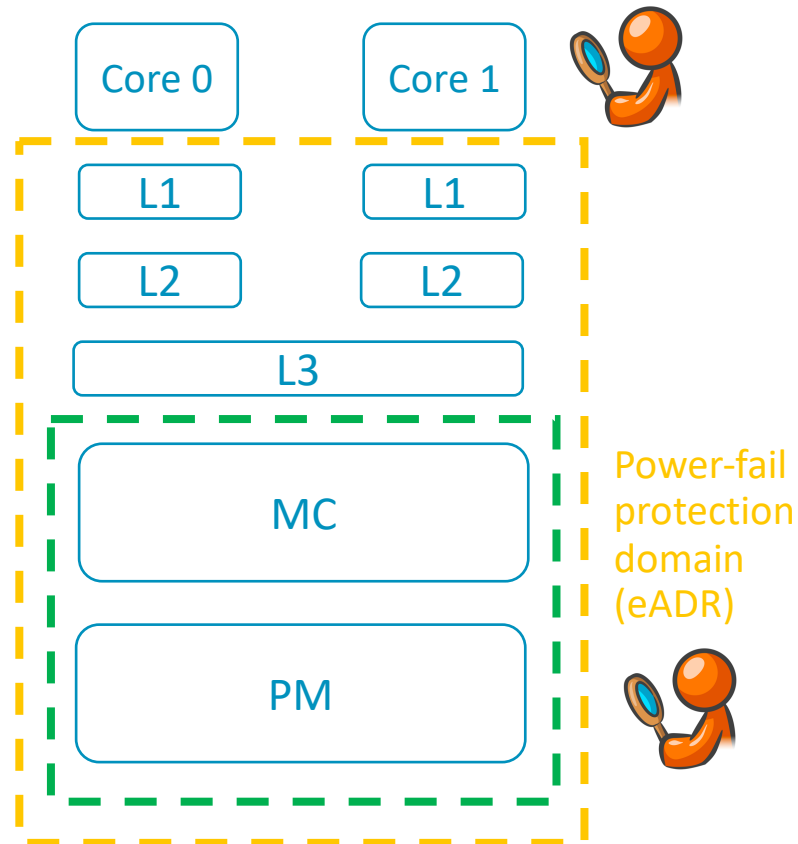
```
2 void
3 addNode(struct root *rootp, int data)
4 {
5     struct node *newnodep;
6     if((newnodep = pm_calloc(1,
7         sizeof(struct node))) == NULL)
8         fatal("out of memory");
9     newnodep->data = data;
10    newnodep->nextp = rootp->headp;
11    pm_flush(newnodep,
12        sizeof(struct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

Consistency
Persistency

aka. Strict Persistency



- 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

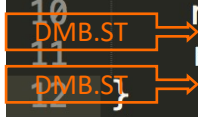
Arm's Weak Memory Model: W->W Reordering Allowed

```
P0          P1
str A=1     while(flag==1){};
str flag=0  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 = calloc(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 }
```



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;
}
```

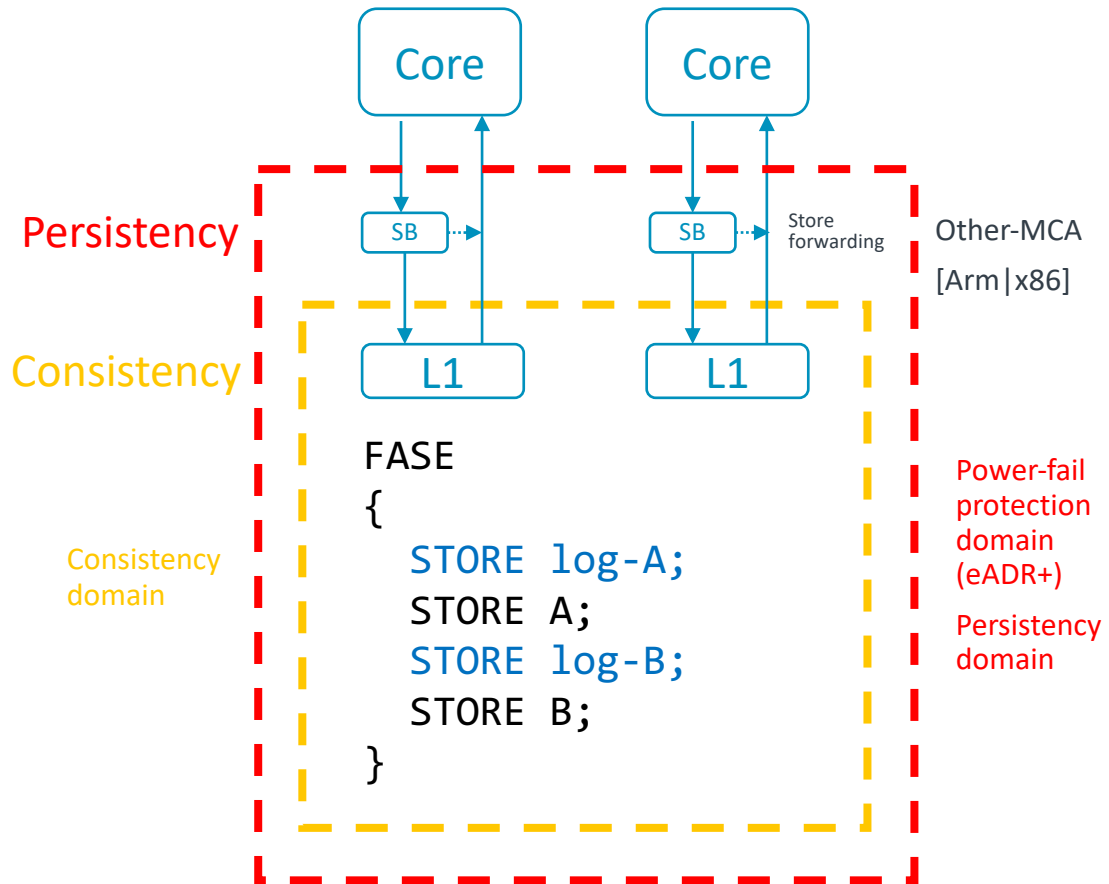
Despite compilers can instrument, barriers are expensive.

Can we remove barriers?

Failure atomicity with eADR+ on Arm

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


Extending Power-fail Protection to Store Buffers

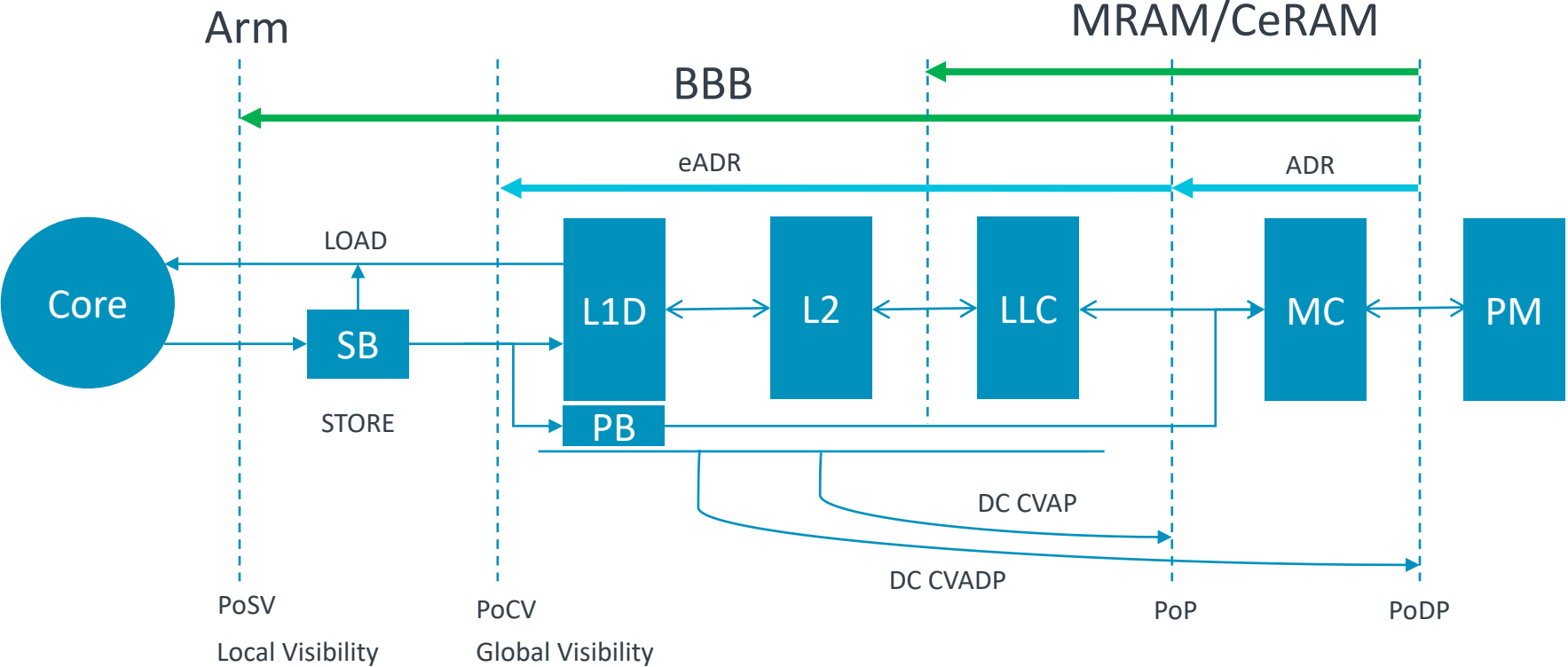


Note: For simplicity of illustration, store buffer may include other buffers on the store path in between core and L1D, e.g., merge buffer

- 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 == Persistency
 - Concurrent programs ✓
- Persistency > Consistency (ahead)
 - Persistency 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

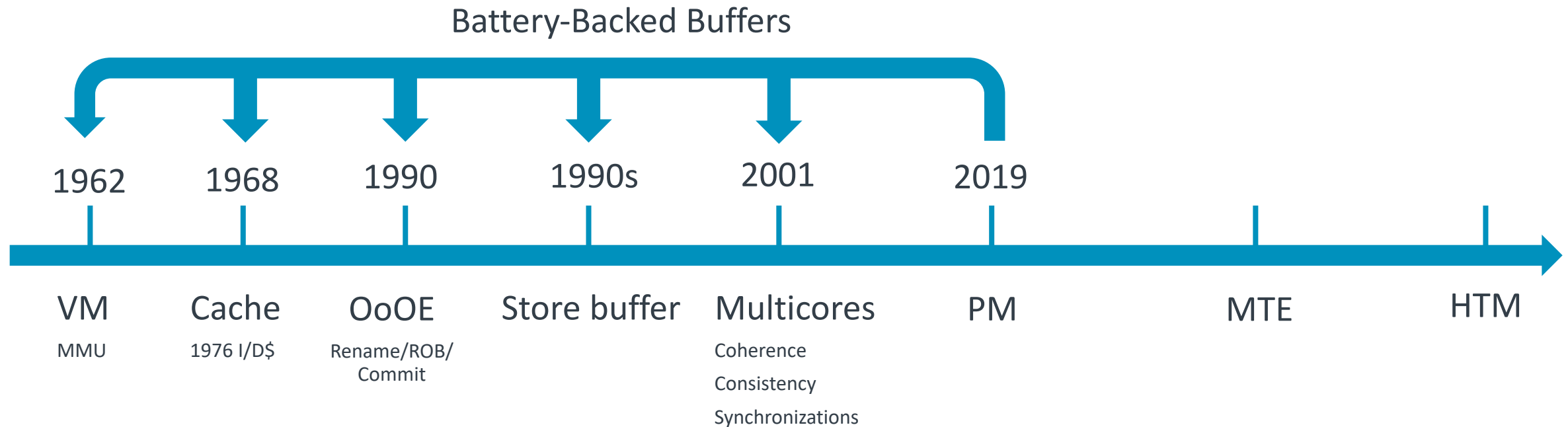


Architectural Support =>

- 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: BBB



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 (317X of BBB)	550 mJ (709X of BBB)
BBB [32 entries]	145 µJ	775 µJ

More BBB µarch details in HPCA'21

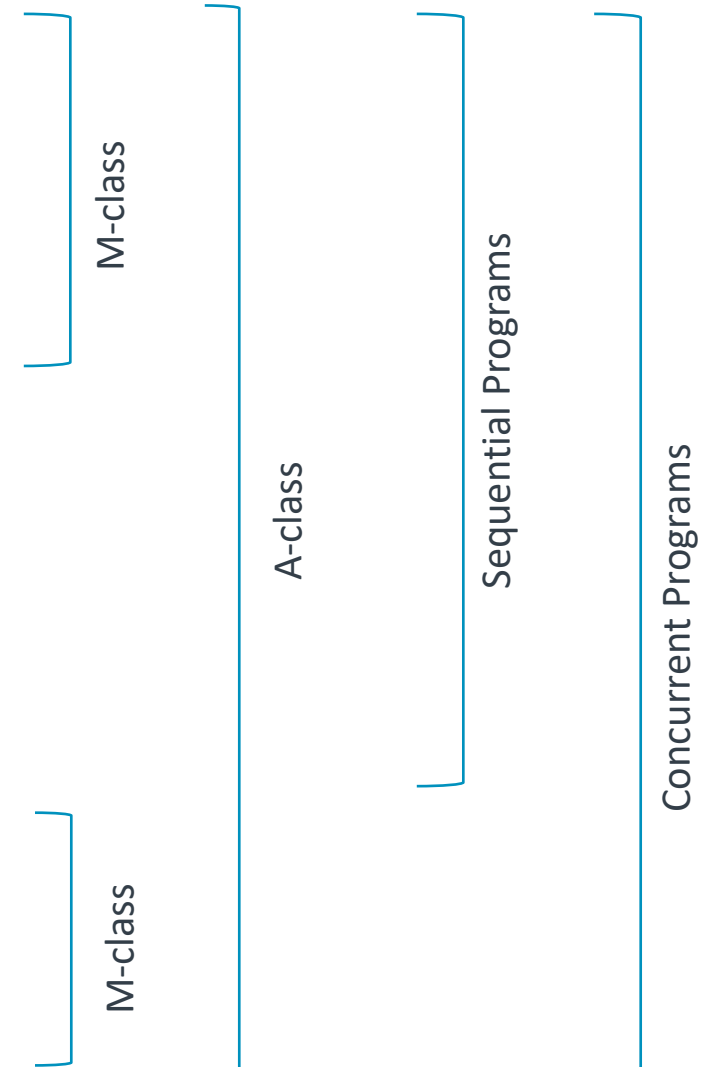
[*] More info : <https://community.arm.com/developer/research/b/articles/posts/simplifying-persistent-programming-with-microarchitectural-support>

arm

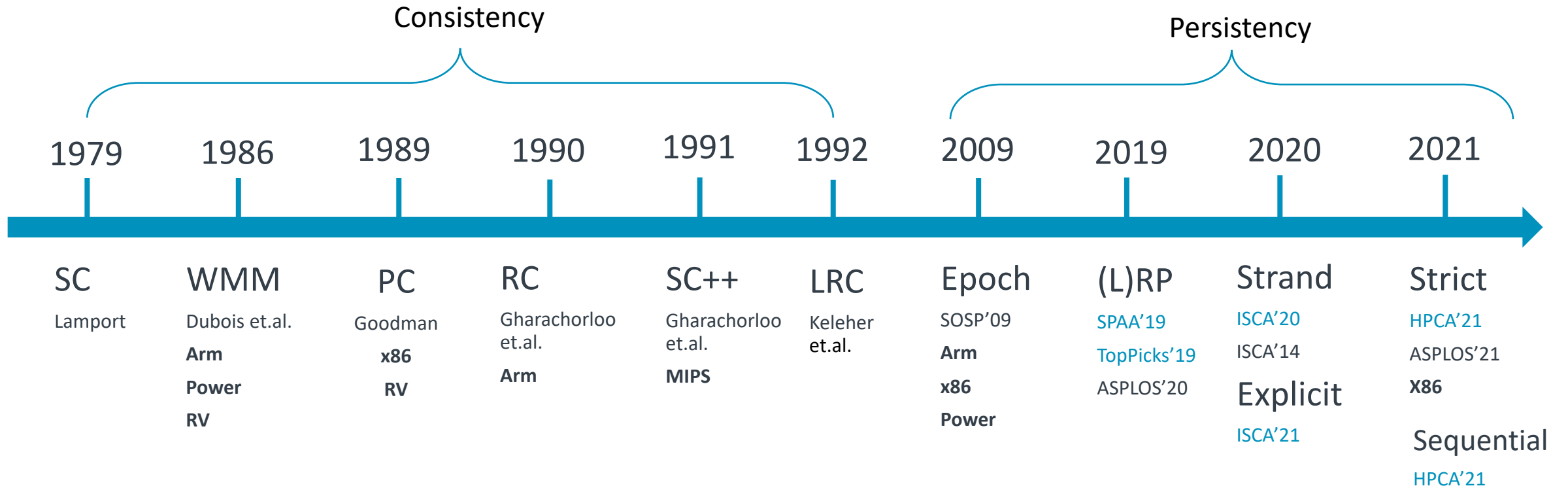
Other Persistent Memory Programming Challenges

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.

Memory Persistency Models – by Arm Research & Co.

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

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

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
Performance		
Improvement	Small	Big
Programmability		
Concurrency	Yes	Yes
Failure atomicity	No	No
Persist ordering	Yes	Yes
Persistent addressing	No	No
Persistent MM	No	No
Portability	High	Low
Implementation		
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

arm

Thank You

Danke

Gracias

谢谢

ありがとう

Asante

Merci

감사합니다

धन्यवाद

Kiitos

شكراً

ধন্যবাদ

תודה

Thanks Richard Grisenthwaite, Nigel Stephens, Robert Dimond, Stuart Biles, Matt Horsnell, Thomas Grocutt, 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, Shiyong 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 Mario 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.