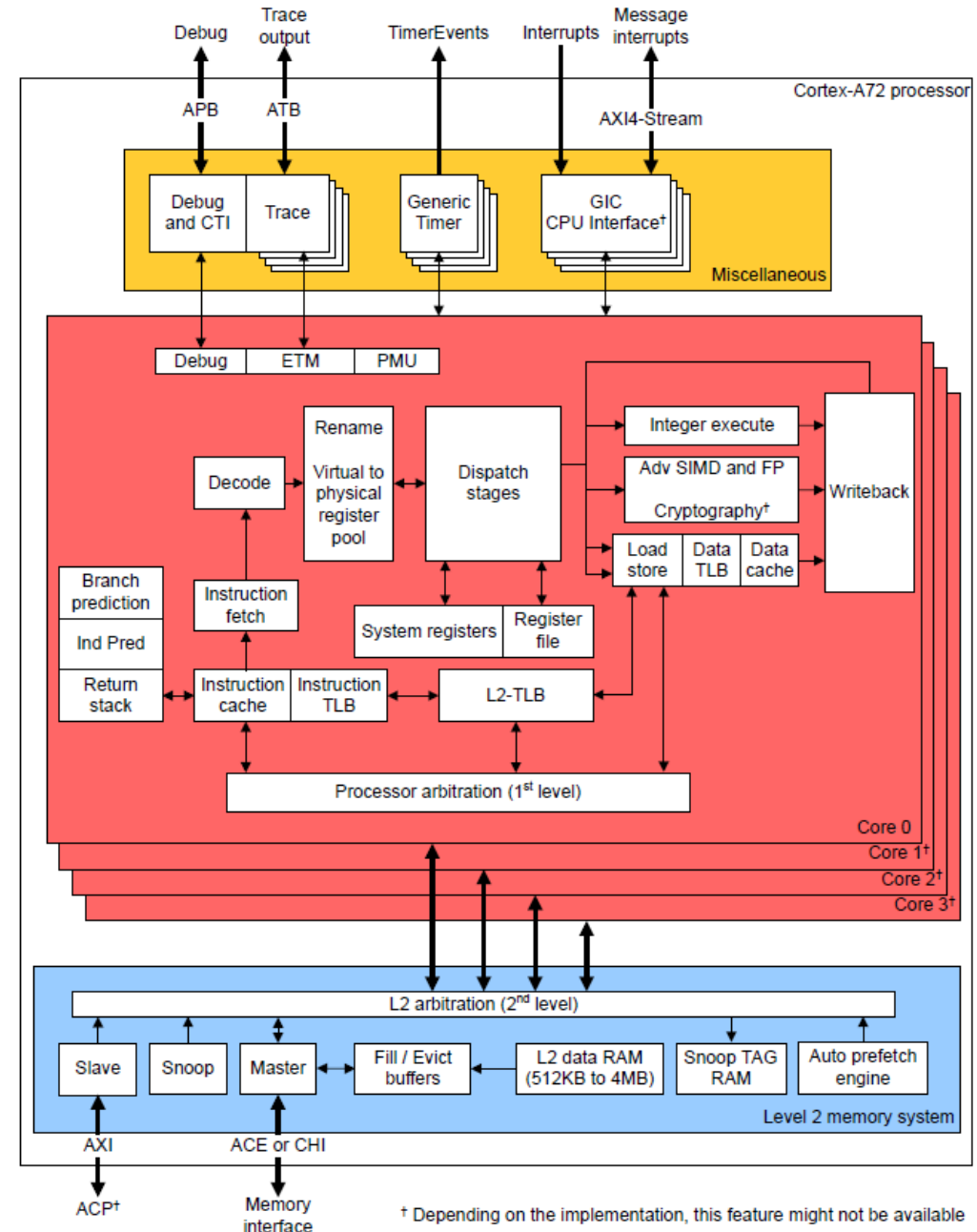


# Cortex-A72 Memory System

# Cortex-A72 Overview

- 4 cores, each with
  - Memory Management Unit
  - TRM Chapter 5
  - Per-core L1 caches
  - 48 kB instruction cache
  - 32 kB data cache
  - TRM Chapter 6
- Shared L2 cache
  - Unified I+D cache
  - 512 kB – 4 MB. RPi 4's BCM2711 has 1 MB.
  - TRM Chapter 7



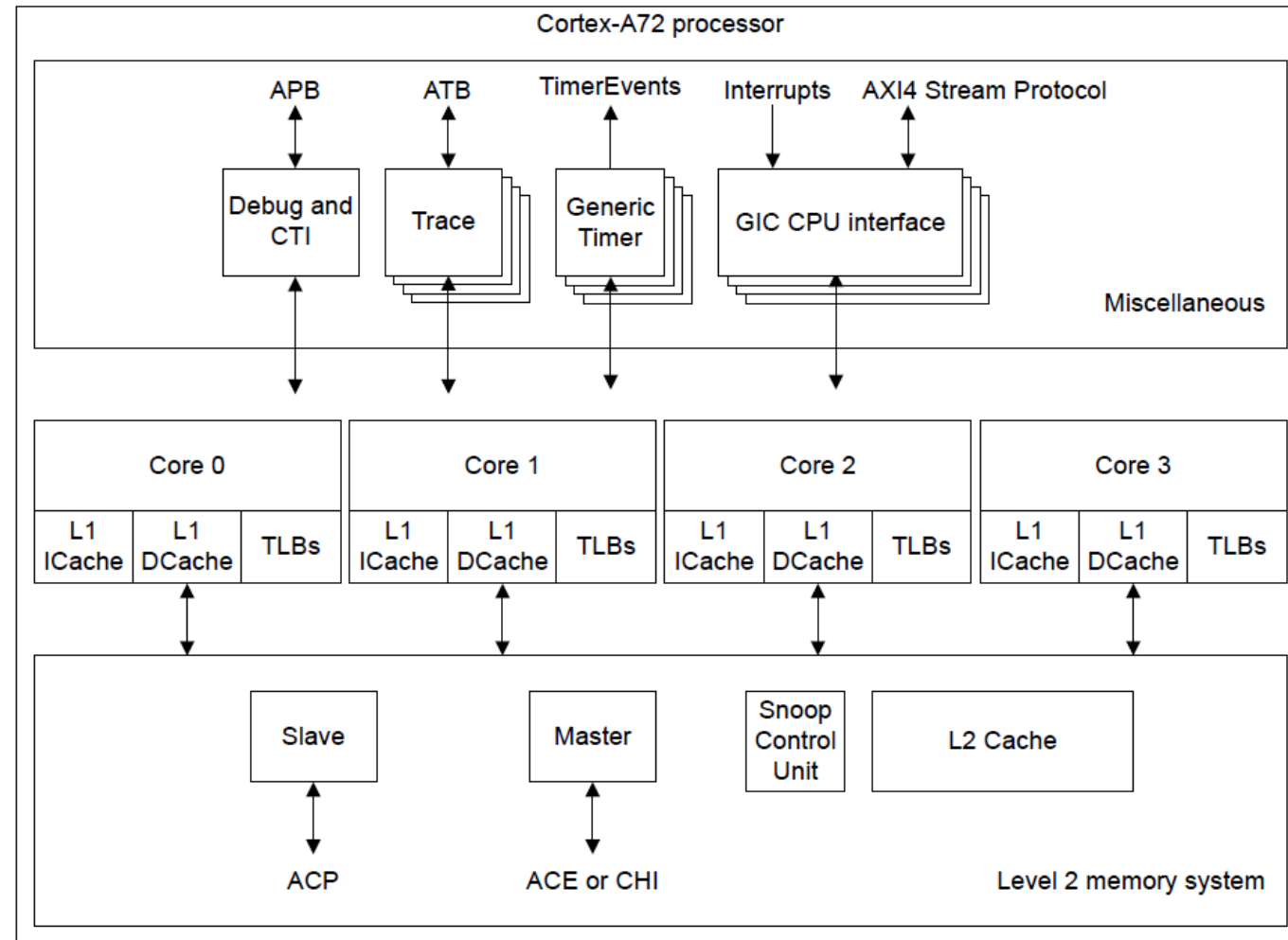
# LI Memory System

## ■ LI Instruction Cache

- 48 KB 3-way set-associative
- 64 byte line length
- 16 byte output path: 4 instructions wide
- Parity protection per halfword
- Physically indexed, physically tagged (PIPT)
- LRU replacement

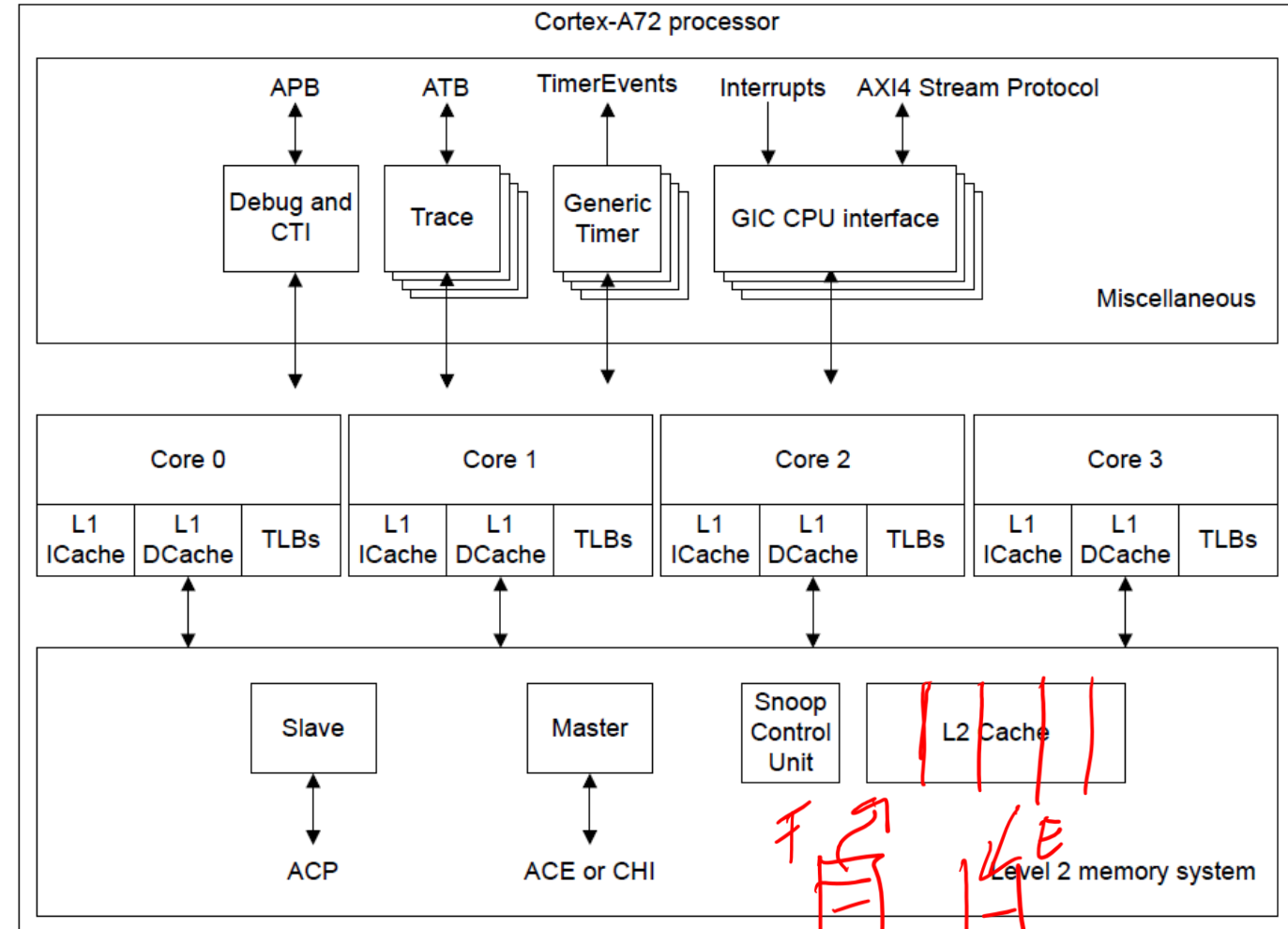
## ■ LI Data Cache

- 32 KB 2-way set associative
- 64 byte line length
- ECC per word
- PIPT, LRU
- Hardware prefetcher
- Normal memory load requests: out-of-order, speculative, non-blocking
- Device memory load requests: non-speculative, non-blocking



# L2 Memory System

- L2 Memory System
- Unified Cache
  - 512 KB – 4 MB
    - RPi 4's BCM2711 has 1 MB
  - 16-way set-associative
  - 64 byte line length
  - Physically indexed, physically tagged
  - Banked pipeline structures
  - Programmable pseudo-LRU or pseudo-random replacement
  - 20 – 28 Fill/Eviction Queues



# Cache Prefetcher

- Hardware prefetcher for L1 data cache, L2 cache
  - Load/Store unit handles prefetch generation
  - On L2 instruction fetch, fetch consecutive cache lines (configurable for 0, 1, 2 or 3 prefetches)
  - On L2 table walk descriptor access, fetch next cache line
  - On prefetch request, forward read data before line is allocated
- <https://developer.arm.com/documentation/100095/0003/Level-1-Memory-System>

# Branch Prediction (Program Flow Prediction)

- Predicted A64 instructions
  - Conditional branches
  - Unconditional branches
  - Indirect branches
- Static predictor
  - Unconditional branch predicted taken
  - Unconditional BL immediate (call) predicted taken, return address pushed
  - Unconditional return branches predicted taken, target popped from return address stack
- Dynamic predictor
  - Branch Target Buffer
  - 2-level global history-based direction predictor
- Indirect branch predictor
  - Stores branch address, state to predict target
- Return stack
  - Pushes address from X30 on BL or BLR, pops address on RET
  - Exception returns not predicted



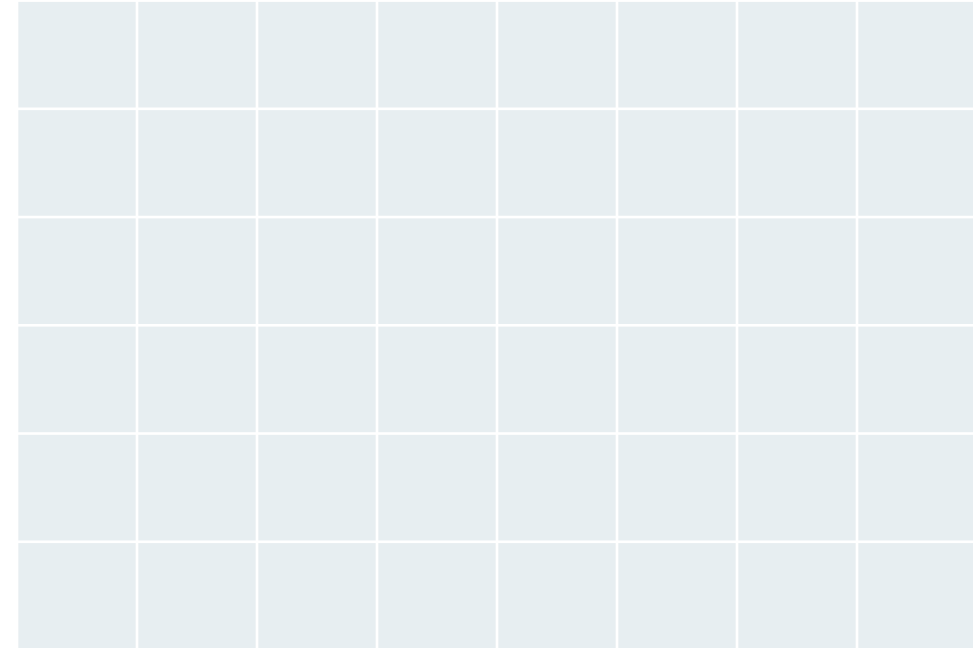
# Memory System Performance

# Memory Access Performance

```
initialize(number_of_elements) ;
a72MeasureDataAccessEvents() ;

start_clock() ;
peStartCounting() ;
for ( ; iterations > 0 ; iterations--) {
    for (CacheLine *p = listHead ; p != NULL ;
        p = p->nextLine)
        ;
}
peStopCounting() ;

print_clock_time(stdout, get_clock_time()) ;
a72PrintDataAccessEvents(stdout) ;
```

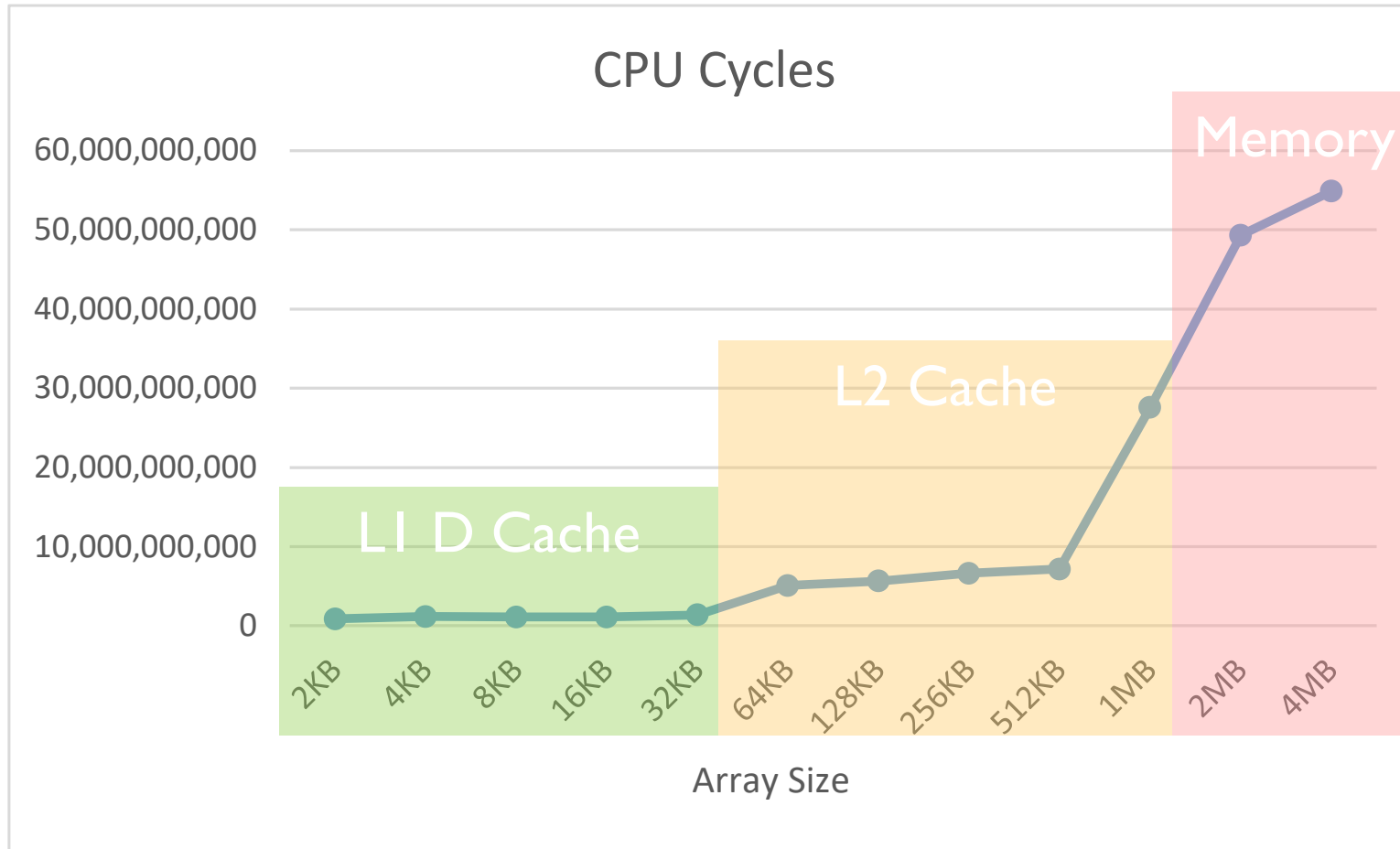


- From <http://sandsoftwaresound.net/arm-cortex-a72-tuning-memory-access/>
- Use pointer-chasing loop code
  - Use linked list, step through with pointer p
  - Test p. If not NULL, load p with p->nextLine.
  - Pseudorandom nextLine locations should eliminate spatial & temporal locality to make cache ineffective

- Measure execution time, PMU events
  - Each list element is 64 bytes long (takes exactly one cache line)
  - Array size and iteration counts modified so program always accesses same number of memory locations
    - # elements \* # iterations = constant = 268,435,456

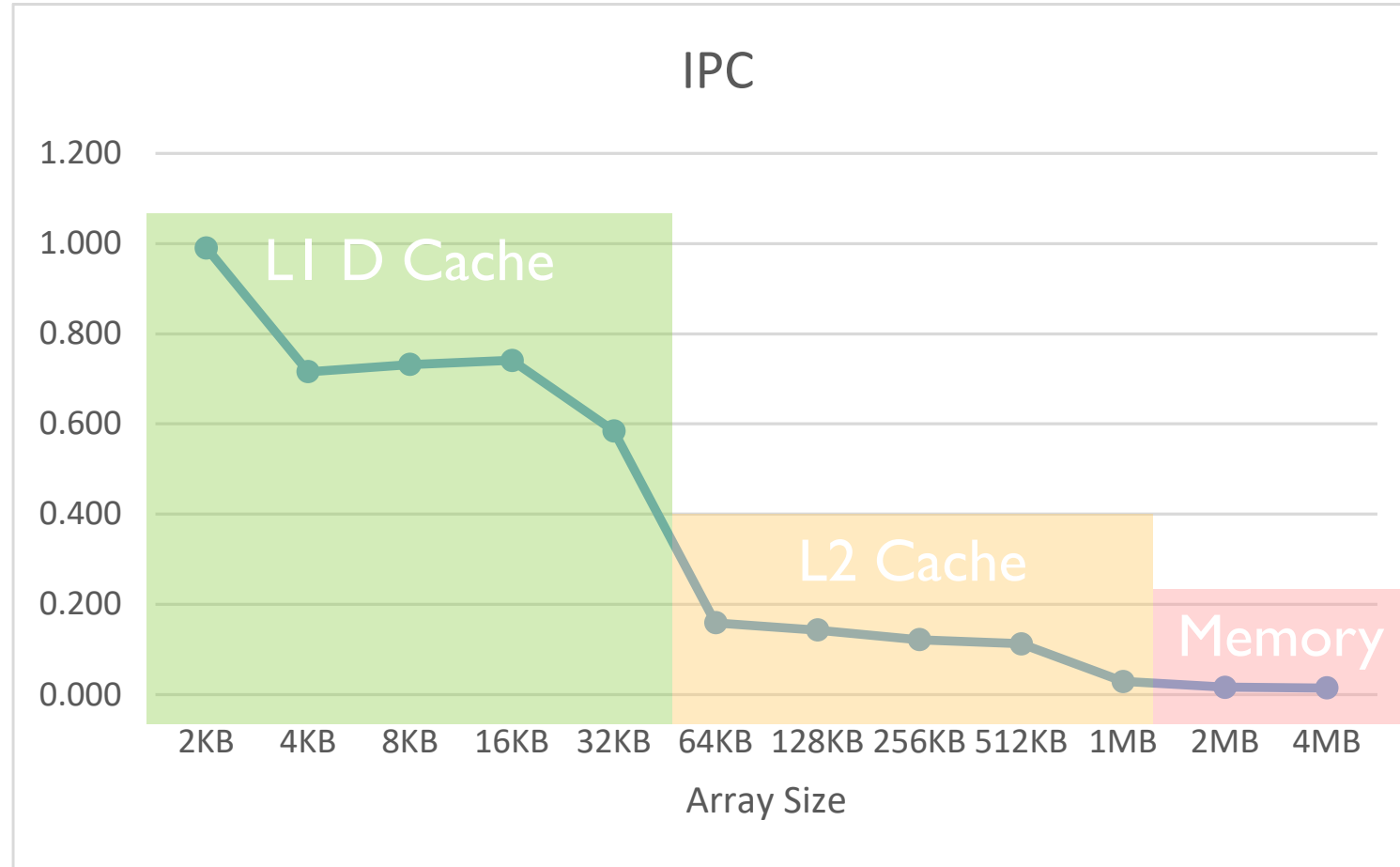


# CPU Cycles for Program Execution



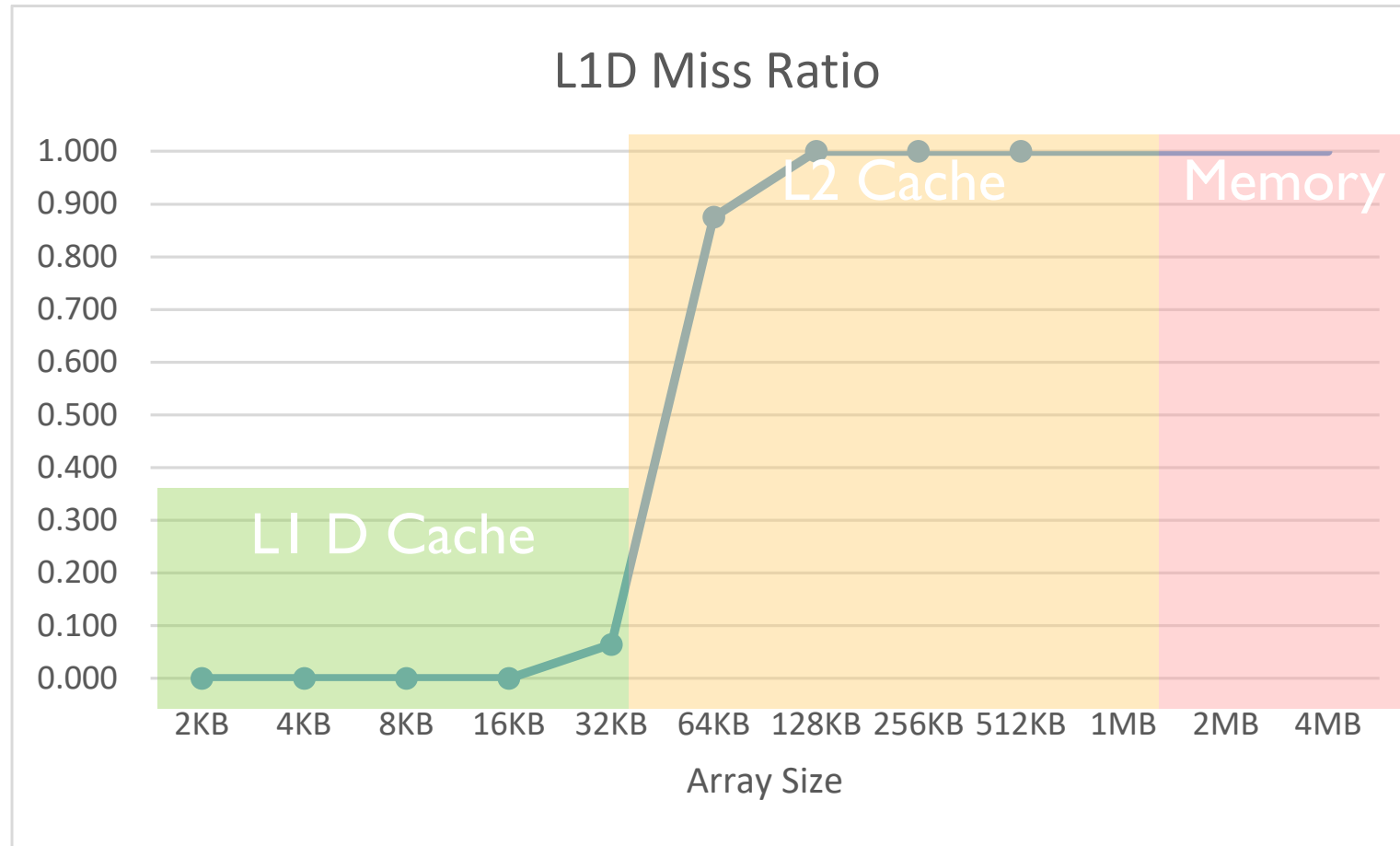
- From <http://sandsoftwaresound.net/arm-cortex-a72-tuning-memory-access/>

# Average Instruction per Cycle (IPC)



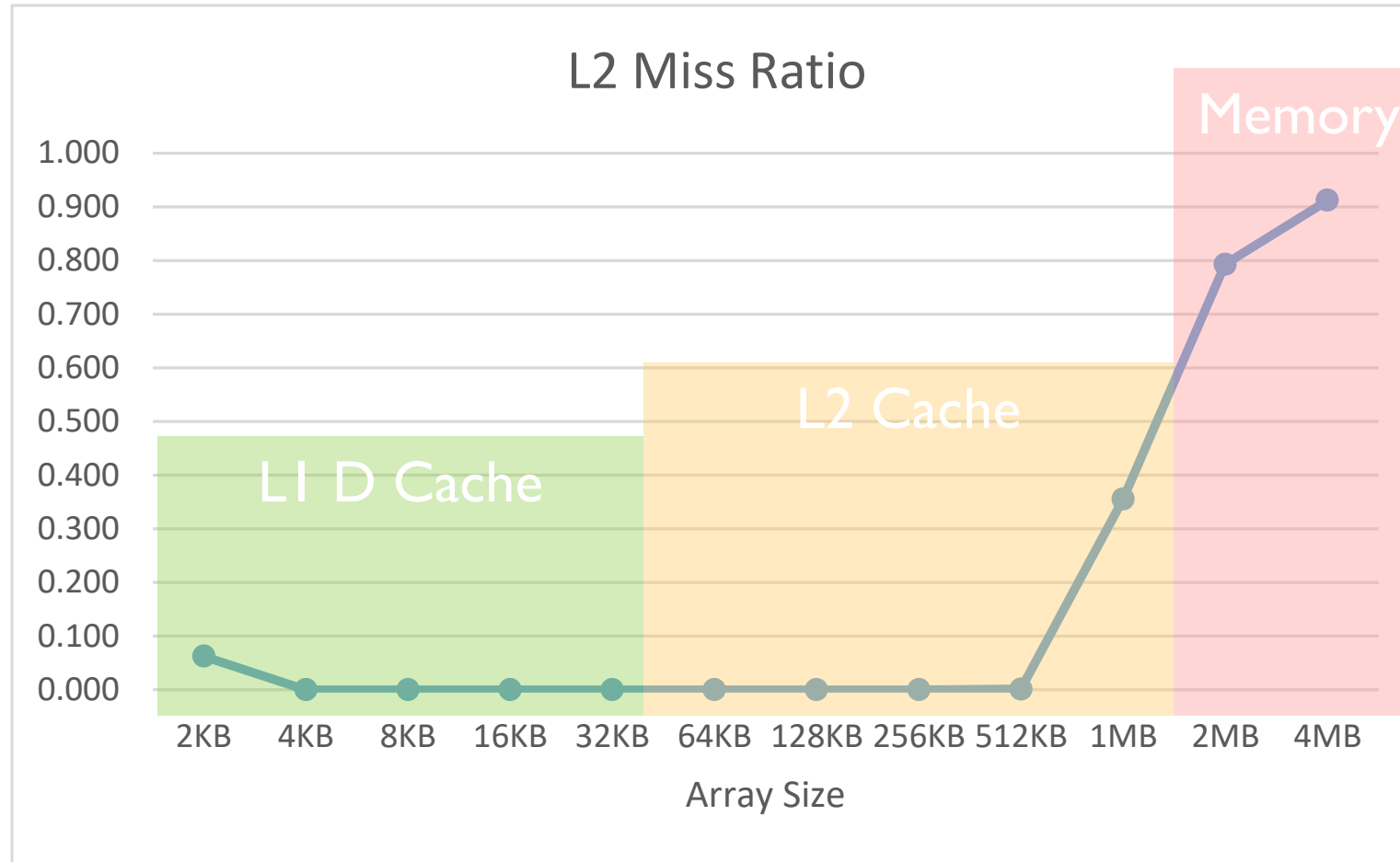
- From <http://sandsoftwaresound.net/arm-cortex-a72-tuning-memory-access/>

# LI D-Cache Miss Ratio



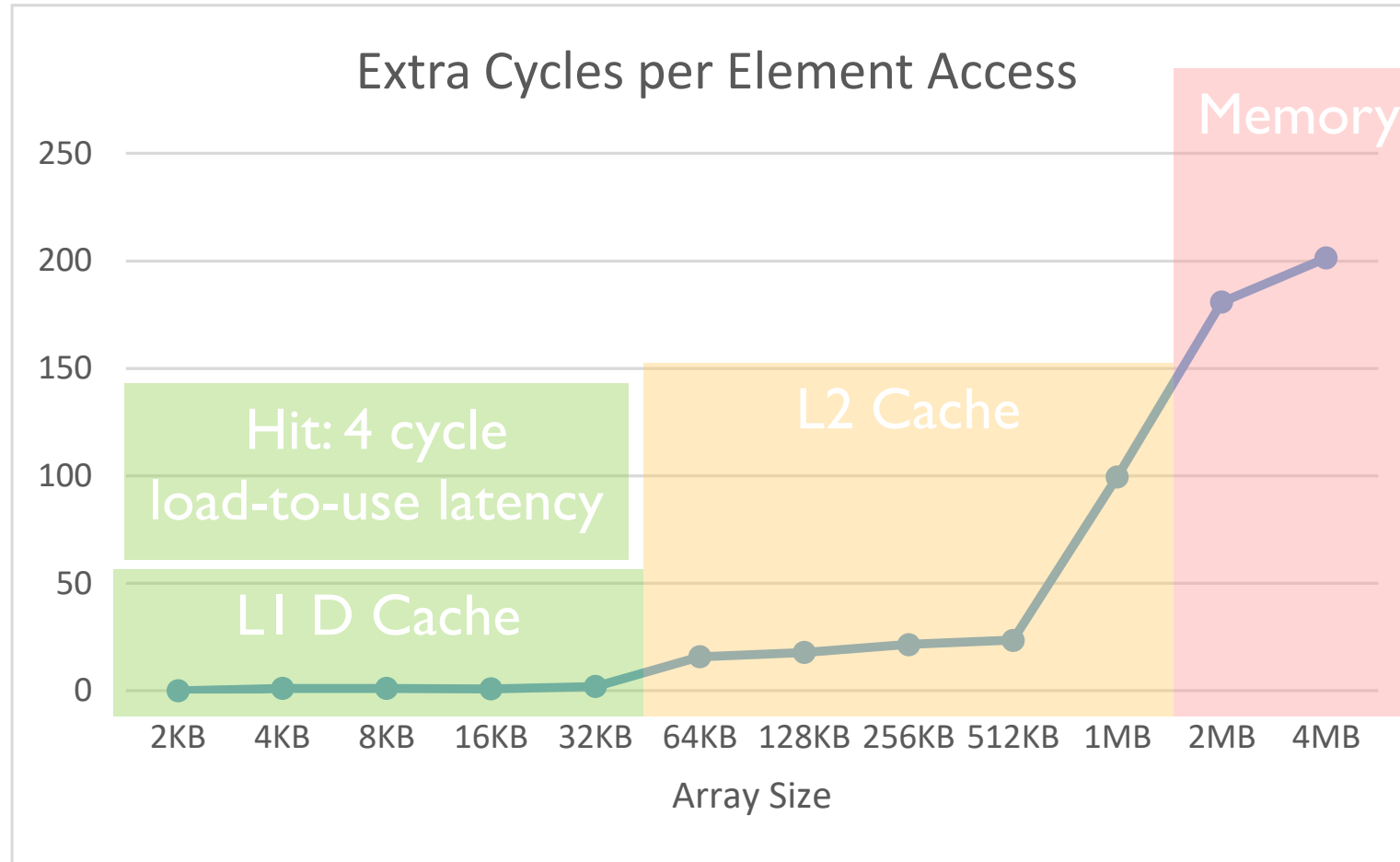
- 32 kB LI Data Cache
- From <http://sandsoftwaresound.net/arm-cortex-a72-tuning-memory-access/>

# L2 Cache Miss Ratio



- 1 MB L2 cache
- From <http://sandsoftwaresound.net/arm-cortex-a72-tuning-memory-access/>

# Extra Cycles per Element Access



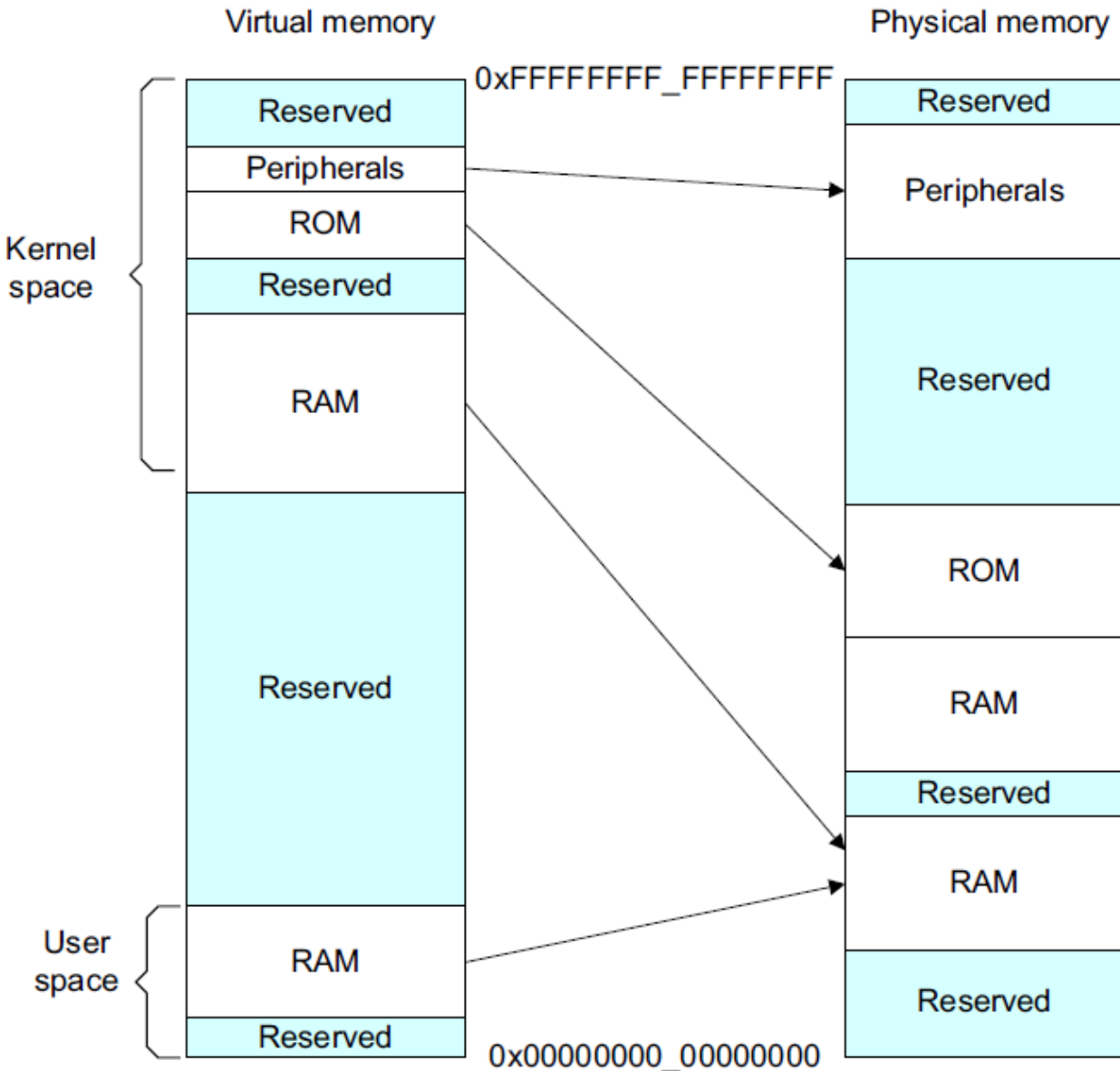
- $(\text{Total cycles for array size } X - \text{total cycles for array size 2 kB}) / \text{number of array accesses}$
- From <http://sandsoftwaresound.net/arm-cortex-a72-tuning-memory-access/> and <http://sandsoftwaresound.net/arm-cortex-a72-execution-and-load-store/>

# Test Parameters

#Elements	Iterations	Array Size	Mem Level
-----	-----	-----	-----
32	8388608	2KB	L1D cache
64	4194304	4KB	L1D cache
128	2097152	8KB	L1D cache
256	1048576	16KB	L1D cache
512	524288	32KB	L1D cache
1024	262144	64KB	L2 cache
2048	131072	128KB	L2 cache
4096	65536	256KB	L2 cache
8192	32768	512KB	L2 cache
16384	16384	1MB	L2 cache
32768	8192	2MB	RAM
65536	4096	4MB	RAM

# Memory Management Unit in AArch64 State

# Virtual Memory

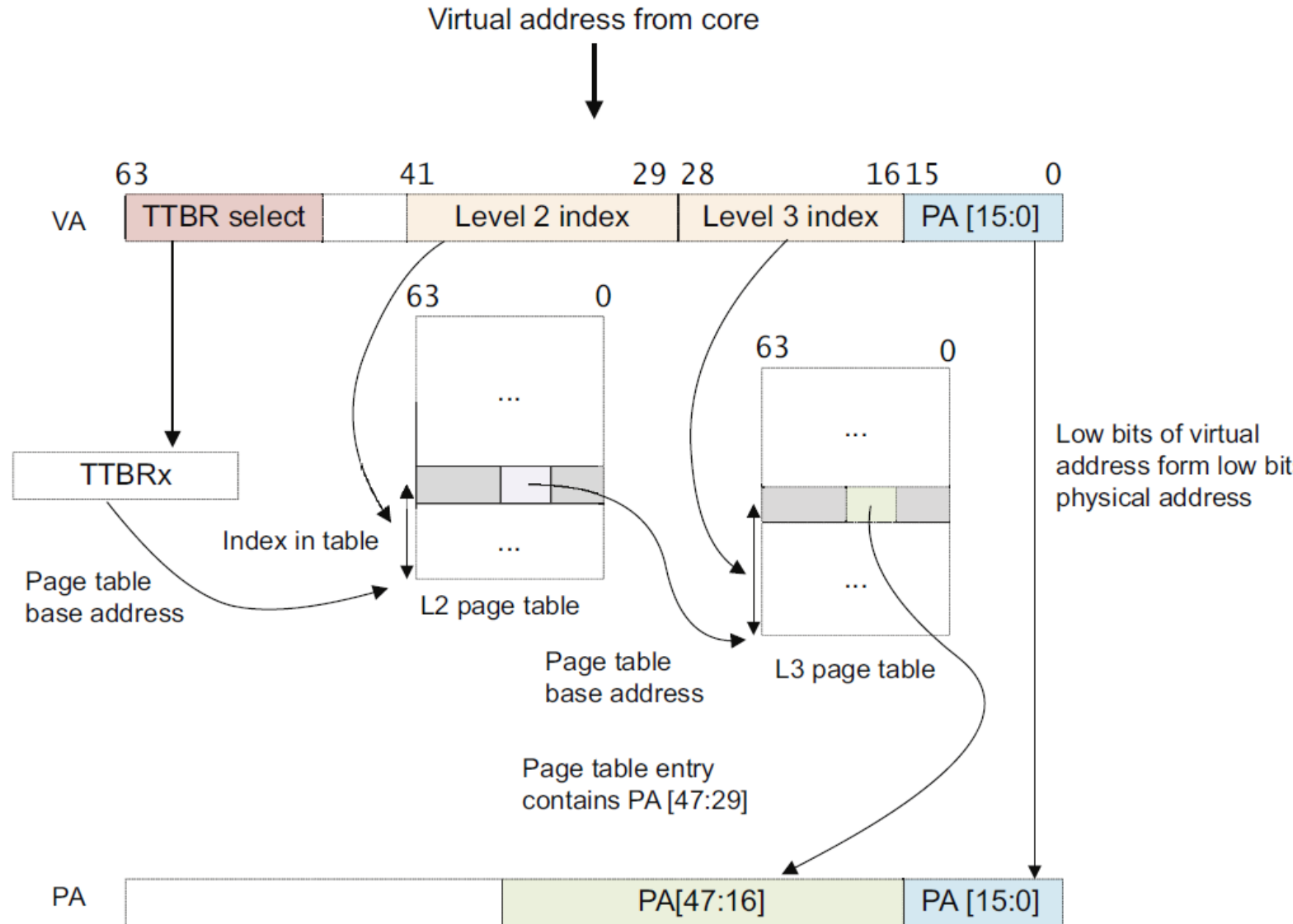


- Enables system to run programs as independent processes with private virtual memory space
  - Simplifies development: don't need to know memory map, its use by other programs
  - Isolates system and programs for protection
  - Allows programs to run even with fragmented physical memory
- Program sees virtual address space, translated by MMU to physical address
- MMU also controls each region's
  - Memory access permissions
  - Memory ordering
  - Cache policies



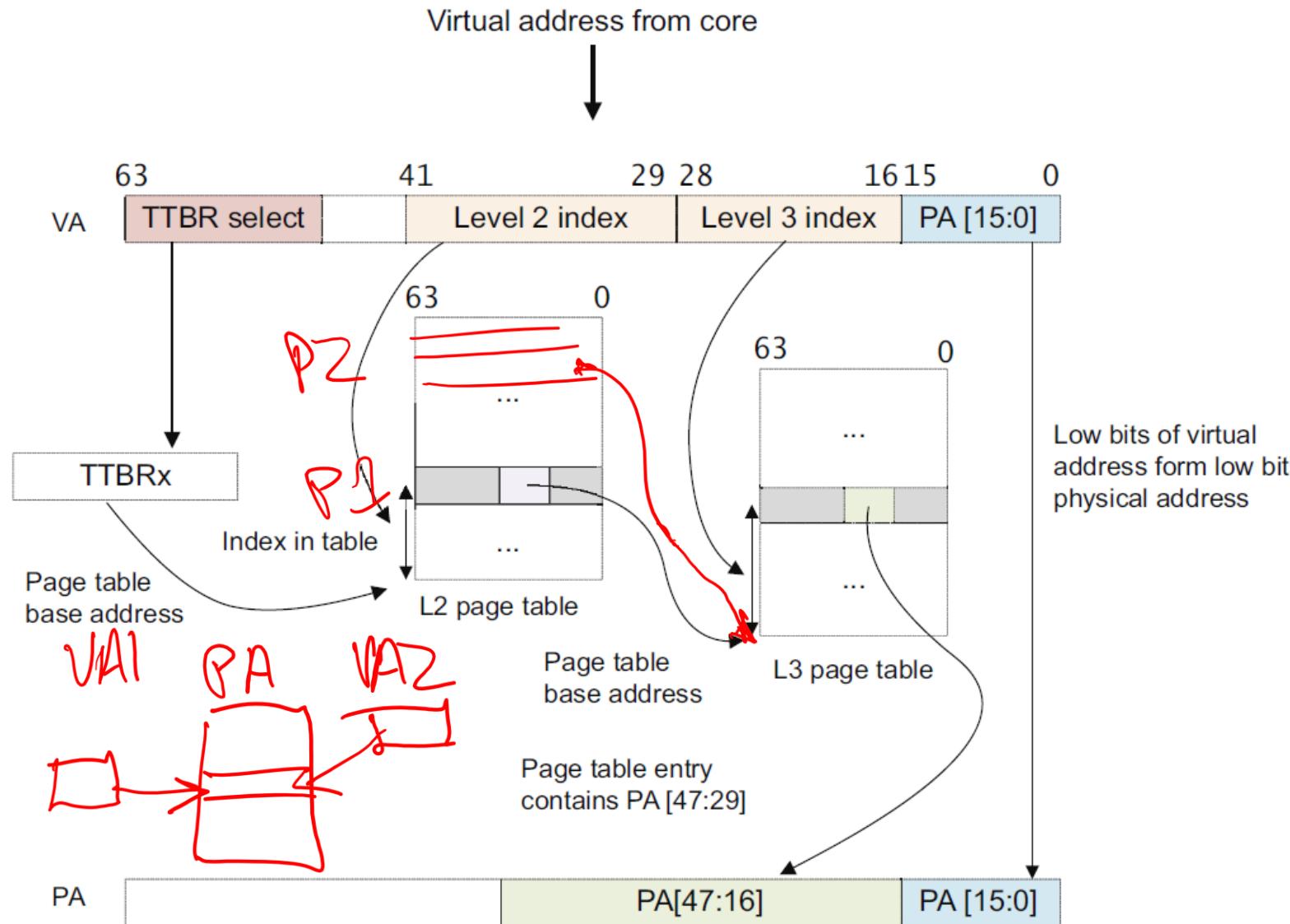
# Translation

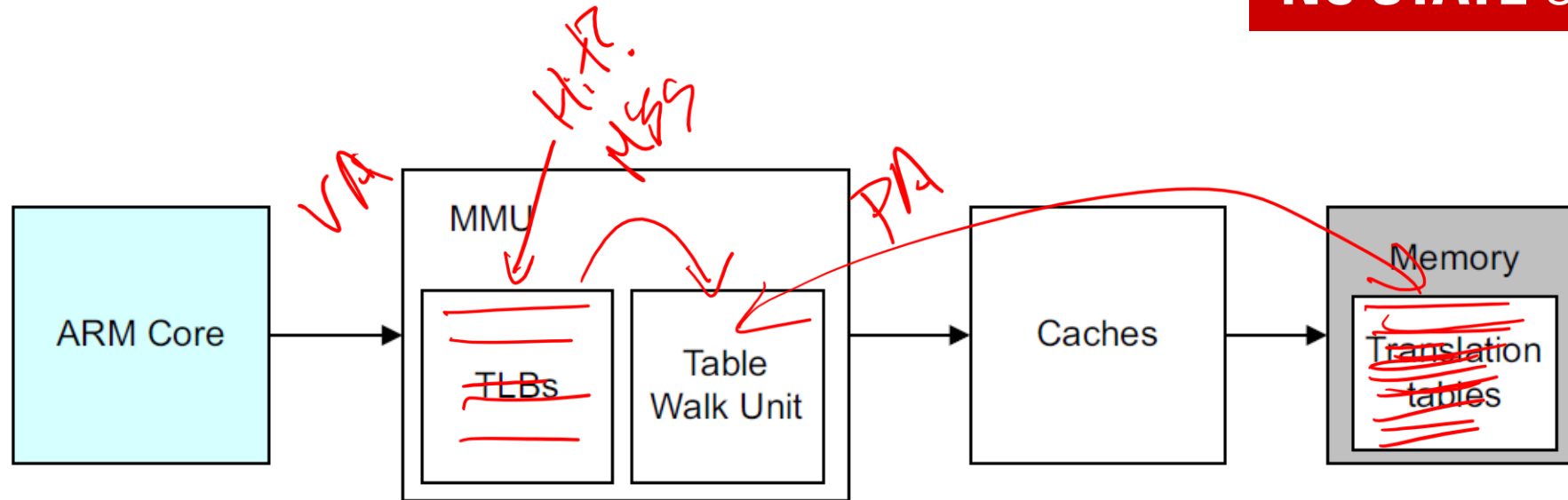
- 1. If  $VA[63:42] = 1$  then TTBR1 is used for the base address for the first page table. When  $VA[63:42] = 0$ , TTBR0 is used for the base address for the first page table.
- 2. The page table contains 8192 64-bit page table entries, and is indexed via  $VA[41:29]$ . The MMU reads the pertinent level 2 page table entry from the table.
- 3. The MMU checks the level 2 page table entry for validity and whether or not the requested memory access is allowed. Assuming it is valid, the memory access is allowed.
- 4. The level 2 page table entry refers to the address of the level 3 page table (it is a table descriptor).
- 5. Bits  $[47:16]$  are taken from the level 2 page table entry and form the base address of the level 3 page table.



# Translation

- 6. Bits [28:16] of the VA are used to index the level 3 page table entry. The MMU reads the pertinent level 3 page table entry from the table.
- 7. The MMU checks the level 3 page table entry for validity and whether or not the requested memory access is allowed. Assuming it is valid, the memory access is allowed.
- 8. The level 3 page table entry refers to a 64KB page (it is a page descriptor).
- 9. Bits [47:16] are taken from the level 3 page table entry and used to form PA[47:16].
- 10. Because we have a 64KB page, VA[15:0] is taken to form PA[15:0].
- 11. The full PA[47:0] is returned, along with additional information from the page table entries.





- Coder, compiler, linker see virtual addresses
- MMU uses top of virtual address to index to find entry in translation table, indicating corresponding physical block
  - TLB holds recently-used page translations

# Cortex-A72 MMU



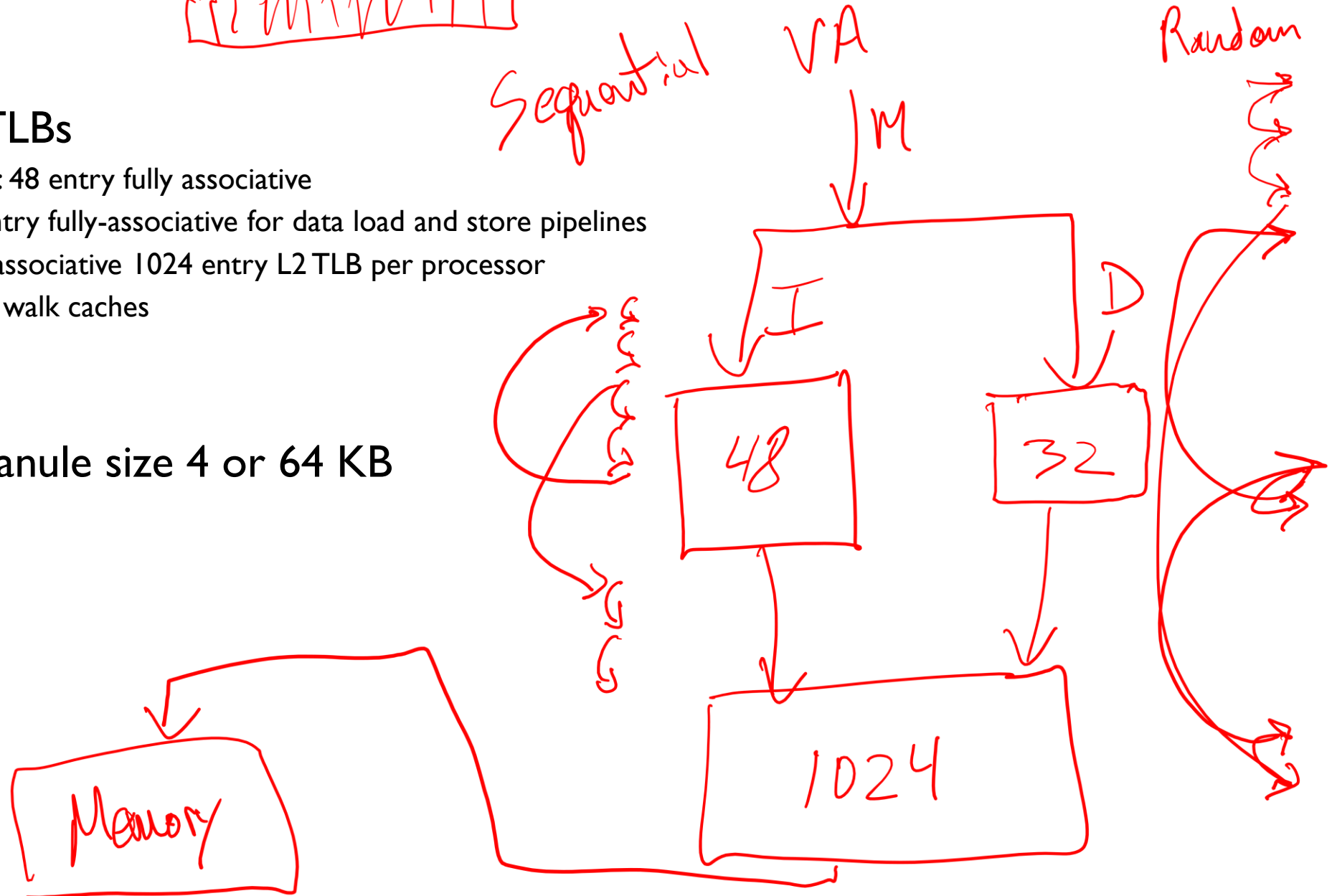
## Two layers of TLBs

- LI Instruction TLB: 48 entry fully associative
- LI Data TLB: 32 entry fully-associative for data load and store pipelines
- L2 TLB: 4-way set associative 1024 entry L2 TLB per processor
- Intermediate table walk caches

## TLB Entries

- ASID, VMID

## Translation Granule size 4 or 64 KB



# Instruction and Data Memory Ordering and Barriers

# Ordering

- Changing instruction, memory access execution order can often speed up a program
  - Parallel instruction issue
  - Out-of-order completion
  - Speculatively prefetch instructions after a branch
  - Starting long latency operations early
  - Merge small memory operations into one (more efficient because less overhead)
  - Multicore systems may migrate cache lines to maintain coherency
- Both software (software) and hardware may change instruction order

# Memory Types

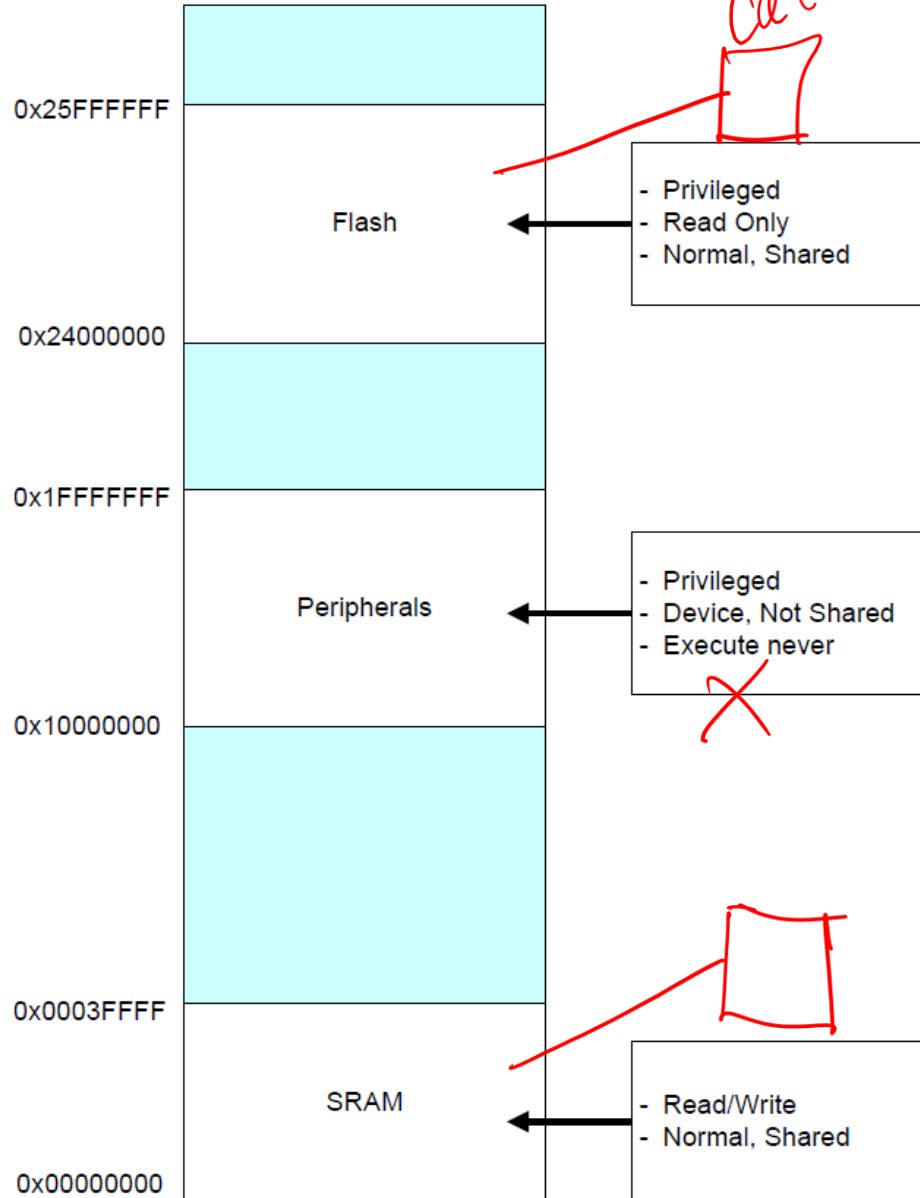
- Is it safe to...
  - Reorder the accesses?
  - Repeat an access?
  - Merge accesses?
- Two exclusive types defined for ARMv8-A
- Normal: all code and most data
  - Yes, yes, yes, assuming the addresses are independent. Is **weakly ordered**.
- Device: peripherals and other memories with possible side effects
  - Probably not, probably not, probably not.

# Device Memory Types

- Consider three properties
  - Gathering (G, nG)
    - Can multiple accesses be merged into one?
  - Reordering (R, nR)
    - Can access to same device be reordered?
  - Early Write Acknowledgement (E, nE)
    - Can intermediate buffer indicate write is done early?
- Range of device memory types
  - Device-nGnRnE: Most restrictive
  - Device-nGnRE
  - Device-nGRE
  - Device-GRE: Least restrictive



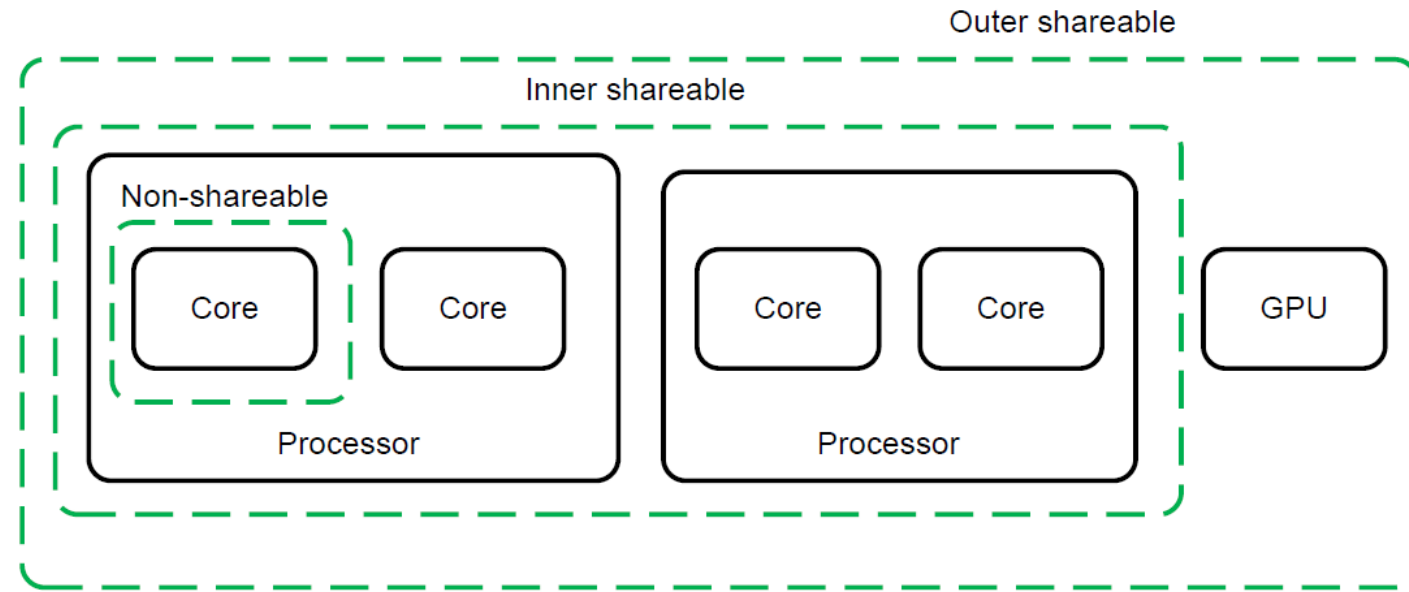
# Memory Attributes



- Attributes for memory regions
  - Access permissions for different privilege levels, memory type, cache policies, shareability
- Normal memory regions
  - May be cacheable
  - May be shareable with other cores
- Device memory regions
  - Always non-cacheable, outer-shareable

# Domains

- Architecture splits system into *domains* to reduce cache coherency traffic and power consumption
- Domain shareability options
  - Non-shareable: no other agents access the domain
  - Inner Shareable: shared by some other agents, but Inner Shareable regions do not affect each other
  - Outer Shareable: operation affecting Outer Shareable Domain affects all of its Inner Shareable Domains
  - Full system: Operation on full system affects all observers



# Barriers

- Barrier instruction forces access ordering and completion when it executes
  - Instruction Synchronization Barrier
  - Data Memory Barrier
  - Data Synchronization Barrier

# Barriers

## ■ Instruction Synchronization Barrier

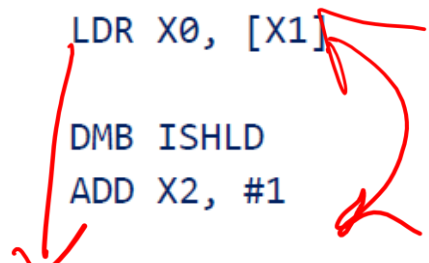
- Flushes pipeline, forces context-changing instructions to complete
- Forces following instructions to be re-fetched from memory (not cache)
- Example: Enable FPU and SIMD, then make sure they are enabled before letting any FPU or SIMD instructions execute

```
MRS X1, CPACR_EL1           // Copy contents of CPACR to X1
ORR X1, X1, #(0x3 << 20)    // Write to bit 20 of X1. (Enable FPU and SIMD)
MSR CPACR_EL1, X1           // Write contents of X1 to CPACR
ISB                          //
```

# Barriers

## ■ Data Memory Barrier

- Prevents reordering of data access instructions across this instruction.



The diagram illustrates how a Data Memory Barrier (DMB) prevents instruction reordering. It shows a sequence of instructions: LDR X0, [X1], DMB ISHLD, ADD X2, #1, and STR X3, [X4]. Red arrows indicate that without the barrier, the LDR and STR instructions could be reordered relative to the ADD instruction. The DMB ISHLD instruction acts as a barrier, ensuring that the LDR instruction is executed before the ADD and STR instructions.

```

LDR X0, [X1]           // Must be seen by the memory system before the
                        // STR below.

DMB ISHLD              // May be executed before or after the memory
                        // system sees LDR.

ADD X2, #1             // Must be seen by the memory system after the
                        // LDR above.

STR X3, [X4]

```

- Forces cache maintenance operations to complete

```

DC CSW, X5             // Data clean by Set/way
LDR x0, [X1]           // Effect of data cache clean might not be seen by
                        // this instruction

DMB ISH                // Effect of data cache clean are seen by this
LDR X2, [X3]           // instruction

```

# Barriers

- Data Synchronization Barrier

- Like DMB, but also prevents all following instructions from executing until DSB completes

```
DC ISW, X5           // operation must have completed before DSB can
                     // complete STR
STR X0, [X1]         // Access must have completed before DSB can complete
DSB ISH
ADD X2, X2, #3        // Cannot be executed until DSB completes
```

# DMB, DSB Parameter

<option>	Description	Ordered Accesses (before – after)	Shareability Domain
OSHL	Operation that waits only for loads to complete, and only to the outer shareable domain	Load – Load, Load – Store	Outer Shareable
OSHST	Operation that waits only for stores to complete, and only to the outer shareable domain.	Store – Store	
OSH	Operation only to the outer shareable domain.	Any – Any	
NSHL	Operation that waits only for loads to complete and only out to the point of unification.	Load – Load, Load – Store	Non-shareable
NSHST	Operation that waits only for stores to complete and only out to the point of unification.	Store – Store	
NSH	Operation only out to the point of unification.	Any – Any	
ISHL	Operation that waits only for loads to complete, and only to the Inner Shareable domain	Load – Load, Load – Store	Inner Shareable
ISHST	Operation that waits only for stores to complete, and only to the Inner Shareable domain.	Store – Store	
ISH	Operation only to the Inner Shareable domain.	Any – Any	
LD	Operation that waits only for loads to complete.	Load – Load, Load – Store	Full system
ST	Operation that waits only for stores to complete.	Store – Store	
SY	Full system operation. This is the default and can be omitted.	Any – Any	

<b>Load - Load/Store</b>	This means that the barrier requires all loads to complete before the barrier but does not require stores to complete. Both loads and stores that appear after the barrier in program order must wait for the barrier to complete.
<b>Store – Store</b>	This means that the barrier only affects store accesses and that loads can still be freely reordered around the barrier.
<b>Any – Any</b>	This means that both loads and stores must complete before the barrier. Both loads and stores that appear after the barrier in program order must wait for the barrier to complete.

# One-Way Barriers

- **Load-Acquire LDAR**
  - Guarantees loads and stores after the LDAR become visible after the LDAR completes
- **Store-Release STLR**
  - Guarantees loads and stores before the STLR become visible before the STLR

