Lecture 02

Memory Hierarchy & I/O Architecture

Memory Hierarchy · Hit Ratio · Cache Design · DMA · Multicore

SENG 21213 · Computer Architecture & Operating Systems

Learning Objectives — By the end of this lecture you should be able to:

- 1. Apply the hit ratio formula to compute average memory access time for a two-level hierarchy
- 2. Describe the principles of temporal and spatial locality and explain how caches exploit them
- 3. Compare direct mapping, set-associative, and fully associative cache mapping functions
- 4. Distinguish programmed I/O, interrupt-driven I/O, and DMA by CPU involvement and interrupt count
- 5. Trace the 7-step DMA transfer process from CPU programming the DMA to completion interrupt
- 6. Explain SMP and multicore architectures and identify cache coherence as a key design challenge

Section 1 · The Memory Hierarchy

📖 Stallings Reference

- Chapter 1: Section 1.5 — Memory Hierarchy
- Key figures: 1.13 (the memory hierarchy pyramid), 1.14 (hit ratio and average access time)

Computer systems require memory that is simultaneously fast, large, and inexpensive. No single technology satisfies all three requirements. The solution is the memory hierarchy: multiple levels of storage, each with different speed, size, and cost characteristics, arranged so that the most-used data is in the fastest level.

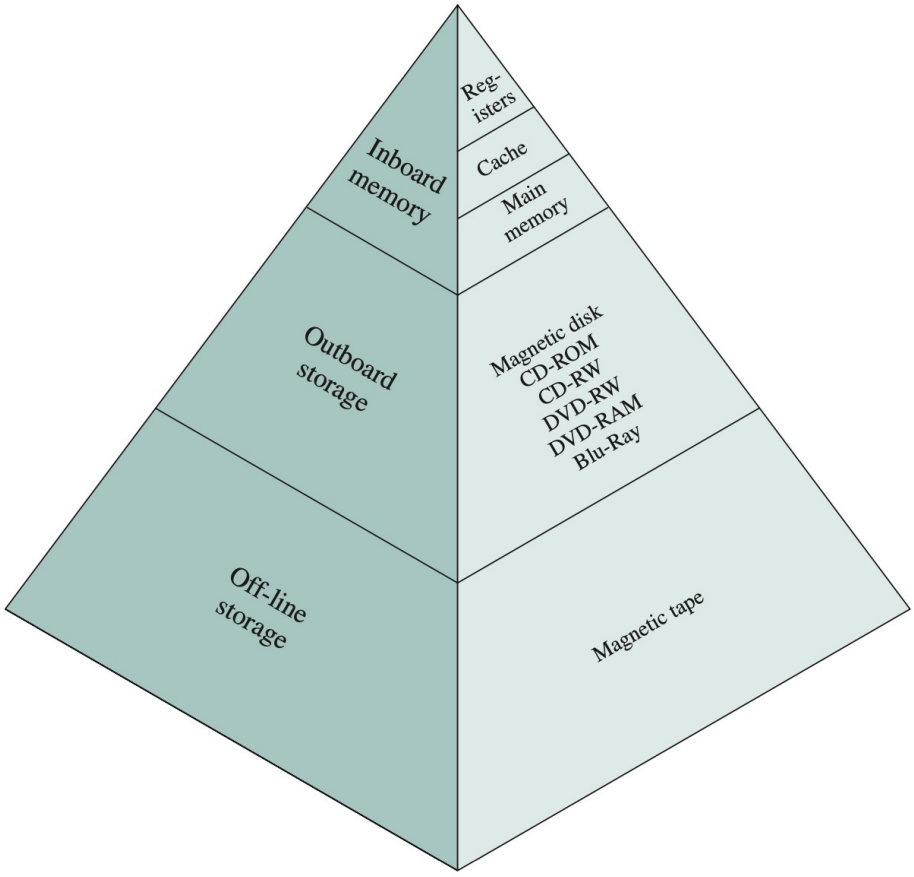

Figure: The memory hierarchy: speed decreases and capacity increases moving down each level (Stallings Fig. 1.13)

1.1 Memory Hierarchy Trade-offs

Parameter	Trend Moving Up the Hierarchy	Implication
Speed	Faster access time	The CPU can get data with less waiting
Capacity	Smaller storage size	Less data fits at any given level
Cost per bit	More expensive	Higher levels cost more to provision
Access frequency	More frequent accesses	Locality ensures the fast levels are used most
Technology	SRAM → DRAM → Flash/SSD → HDD → Tape	Each step uses a different storage mechanism

1.2 Two-Level Memory and the Hit Ratio Formula

Consider the simplest memory hierarchy: a cache (level 1) and main memory (level 2). Every memory reference first checks the cache:

- Cache hit: The requested word is found in cache. Access time = T_1 (cache access time).

- Cache miss: The word is not in cache. The block is fetched from main memory. Access time = T_1 + T_2 (cache check + main memory access).

★ Hit Ratio Formula

- $T_{avg} = H \times T_1 + (1 - H) \times (T_1 + T_2) = T_1 + (1 - H) \times T_2$
- H = hit ratio = fraction of memory accesses served by the cache
- T_1 = cache access time; T_2 = main memory access time
- Example: H = 0.95, T_1 = 0.1 µs, T_2 = 1.0 µs
- $T_{avg} = 0.1 + (1 - 0.95) \times 1.0 = 0.1 + 0.05 = 0.150$ µs
- Without cache: T_2 = 1.0 µs. Speedup = 1.0 / 0.150 = 6.67×

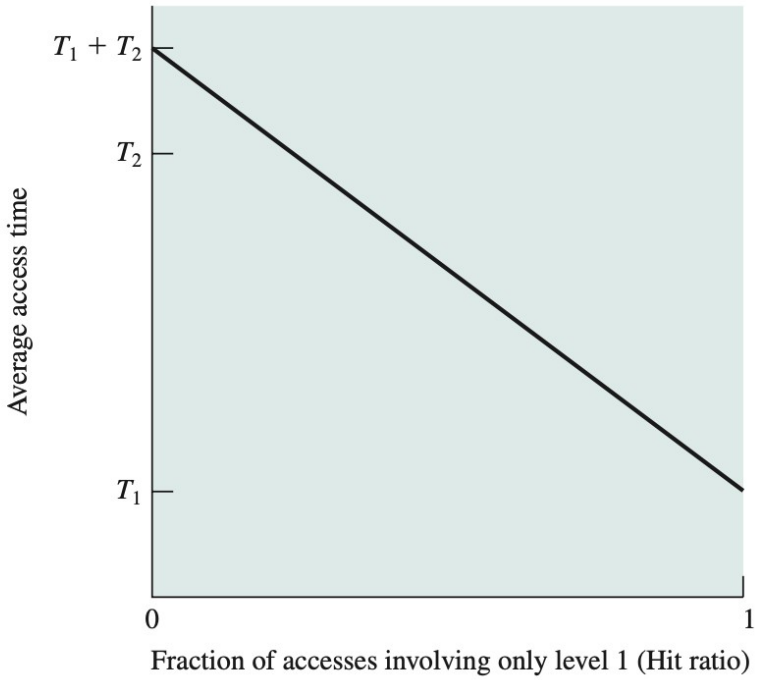

Figure: Two-level memory model — hit path (cache) and miss path (main memory) (Stallings Fig. 1.14)

Section 2 · Cache Memory — Design and Operation

📖 Stallings Reference

- Chapter 4: Cache Memory — Section 4.1 to 4.3
- Mapping functions: direct, set-associative, and fully associative

Cache memory is a small, fast memory that holds copies of recently used blocks from main memory. Its effectiveness relies on two fundamental principles of locality:

◆ Principles of Locality

- Temporal locality: A memory location that was accessed recently is likely to be accessed again soon. (Example: a loop variable read 10,000 times during a tight loop.)
- Spatial locality: If a location at address X is accessed, locations near X (X+1, X+2, …) are likely to be accessed soon. (Example: sequentially reading array elements.)
- Caches exploit both: temporal by keeping recently accessed items; spatial by loading whole cache lines (blocks of contiguous bytes).

$$(0.95)\,(0.1\,\mu s)\;+\;(0.05)\,(0.1\,\mu s\;+\;1\,\mu s)\;=\;0.095\;+\;0.055\;=\;0.15\,\mu s$$

Figure: Cache memory positioned between the CPU and main memory in the memory hierarchy (Stallings Ch. 4)

2.1 Cache Design Parameters

Parameter	Options	Impact
Cache size	Typically 16 KB – 32 MB depending on level	Larger → higher hit ratio; slower access and higher cost
Block (line) size	Typically 32 – 128 bytes	Larger → exploits spatial locality more; but larger miss penalty
Mapping function	Direct, Set-associative, Fully associative	See detailed comparison below
Replacement algorithm	LRU, Clock, Random, FIFO	LRU performs best for typical programs
Write policy	Write-through or Write-back	Write-back reduces bus traffic; needs dirty-bit tracking
Number of levels	L1 (4-64 KB, 1-4 cycles), L2 (256KB-4MB, 10-15 cycles), L3 (4-32MB, 30-50 cycles)	Multiple levels balance speed and capacity

2.2 Mapping Functions

The mapping function determines which cache line(s) a main memory block can be stored in:

Mapping	How It Works	Conflict Miss Risk	Hardware Cost
Direct	Memory block i maps to cache line i mod N (N = cache lines). Exactly one possible location.	High — two frequently used blocks can compete for the same line	Low — one comparator
Set-associative (k-way)	Cache divided into S sets of k lines. Block i maps to set i mod S; within the set any of the k lines can hold it.	Moderate — k blocks can share each set	Medium — k comparators per set
Fully associative	A block can go into any cache line. No fixed mapping.	Low — conflict misses eliminated	High — one comparator per cache line; expensive for large caches

⚠ Common Mistake

- Direct mapping and fully associative are the two extremes. Set-associative is the industry standard — a 2-way or 4-way set-associative cache captures most of the benefit of full associativity at a fraction of the hardware cost. Virtually all modern CPU caches are set-associative.

Section 3 · I/O Architecture — Three Techniques

📖 Stallings Reference

- Chapter 1: Section 1.6 — I/O Communication Techniques
- Chapter 7: I/O Management — Programmed I/O, Interrupt-Driven I/O, DMA

Three distinct techniques exist for managing I/O transfers, progressively reducing CPU involvement in the data movement:

3.1 Programmed I/O (Polling)

The CPU directly controls every aspect of the I/O operation and spins in a polling loop until the device signals completion. The CPU cannot do anything else during this time.

```
while (status_register != DONE) ;  /* spin — check status port */
data = data_register;               /* read data when device ready */
```

- Pro: Simple to implement; no interrupt hardware needed.
- Con: CPU utilisation = 0% for useful work during the entire I/O wait — completely wasteful.
- Use case: Only for very fast I/O where the wait is shorter than interrupt overhead (e.g. reading a few bytes from a fast local SSD in embedded systems).

3.2 Interrupt-Driven I/O

The CPU issues an I/O command and immediately returns to executing the current program. When the device completes one unit of data transfer, it raises an interrupt, and the CPU runs the ISR to handle the data.

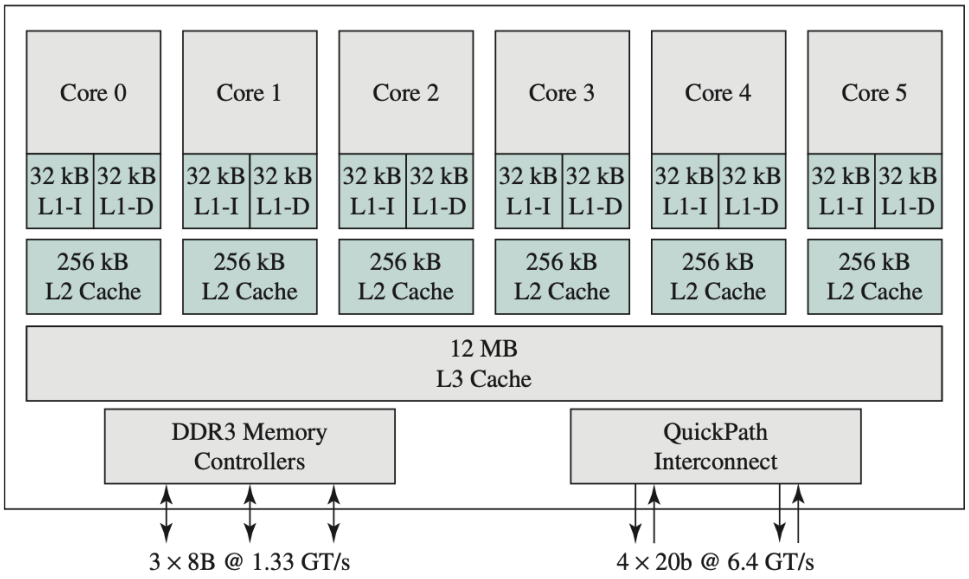

Figure: Comparison of I/O techniques: programmed I/O, interrupt-driven I/O, and DMA (Stallings Fig. 1.17)

- Pro: CPU is free to run other code while I/O proceeds.
- Con: One interrupt per byte or word transferred — for a 1 MB file at 1 byte per interrupt = 1,048,576 interrupts. Huge overhead.
- Use case: Slow devices with infrequent transfers (keyboard, mouse, serial port).

3.3 Direct Memory Access (DMA)

A dedicated DMA controller handles an entire block of data transfer between a device and memory, without CPU involvement for each byte. The CPU is only interrupted once — at the start (to set up) and once at the end (to signal completion).

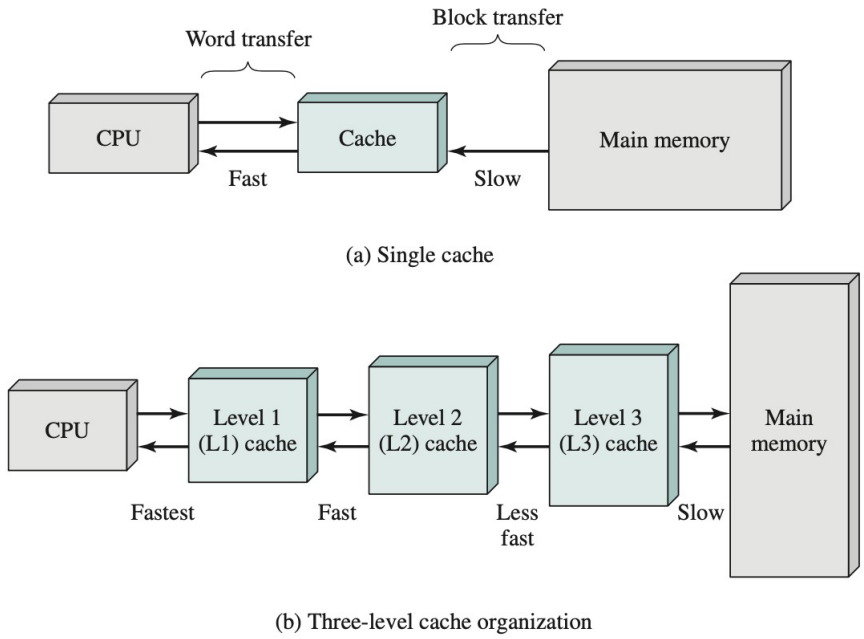

Figure: DMA controller architecture — DMA moves data between device and memory using cycle stealing (Stallings Fig. 1.18)

DMA transfer process (7 steps):

1. CPU programs the DMA controller: source/destination address, byte count, transfer direction, device ID.
2. CPU issues the DMA GO command and returns to executing user/OS code immediately.
3. DMA controller requests the bus from the bus arbiter ('cycle stealing' — takes the bus for one transfer cycle at a time).
4. DMA transfers one word between the I/O device and main memory using MAR/MBR equivalents internal to the DMA.
5. DMA releases the bus for one cycle so the CPU can execute (if needed), then steals another cycle for the next transfer.
6. Steps 3–5 repeat until the byte count reaches zero.
7. DMA asserts an interrupt to signal completion. The brief ISR updates bookkeeping (bytes transferred, next buffer pointer).

Technique	CPU Involvement	Interrupts per MB	Best For
Programmed I/O	100% — CPU polls every byte	0 (no interrupts — just busy-wait)	Very fast, tiny transfers
Interrupt-Driven I/O	ISR runs per byte/word	1,000,000 per MB (at 1 byte/interrupt)	Slow devices, sporadic transfers
DMA	ISR runs once per block	1 per block (regardless of size)	High-speed bulk transfer (disk, NIC, GPU)

Section 4 · Multiprocessor and Multicore Systems

📖 Stallings Reference

- Chapter 1: Section 1.7 — Multicore and Other Organizations
- Example: Intel Core i7-990X — 6 cores, each with L1/L2, shared L3

4.1 Symmetric Multiprocessing (SMP)

In an SMP system, multiple fully functional processors (each with its own registers and cache) share a single main memory and run a single OS instance. All processors have equal access to all memory and I/O devices.

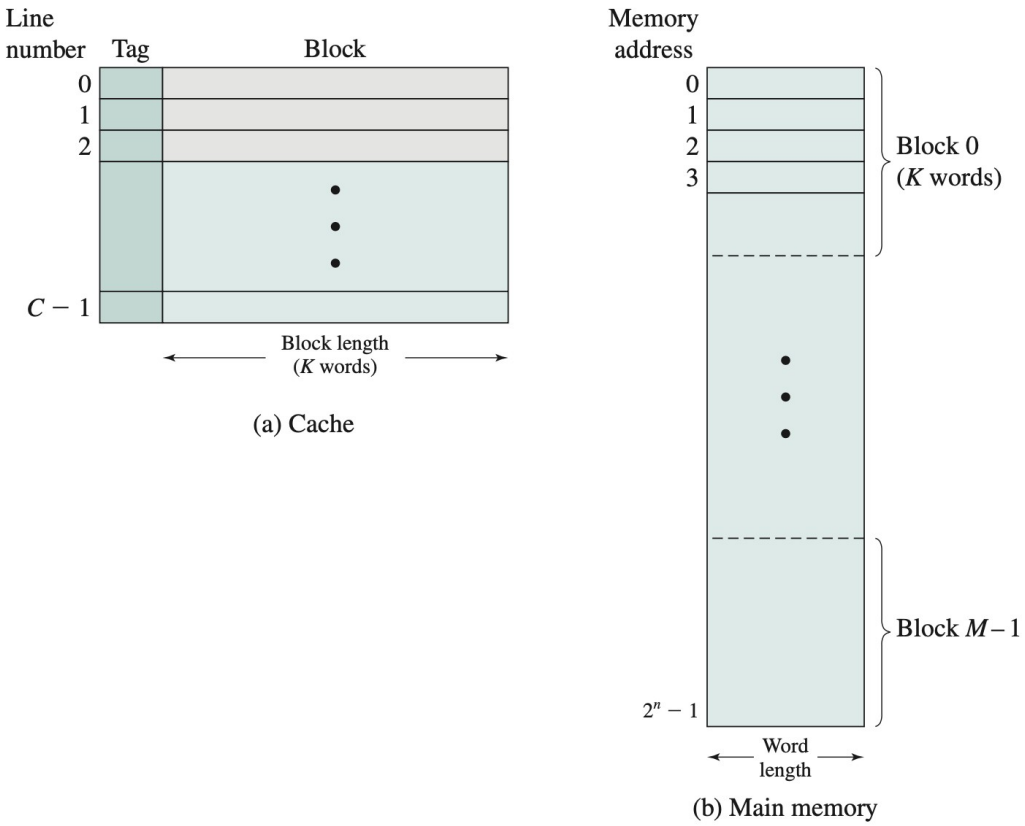

Figure: SMP architecture — multiple processors sharing memory and I/O via a common bus (Stallings Fig. 1.20)

SMP Advantage	Explanation
Performance	Multiple processes or threads execute in true parallel on separate processors.
Availability	Failure of one processor reduces performance but does not halt the system.
Incremental growth	Processors can be added to increase throughput without redesigning the system.
Scalability	Vendors offer product ranges with different processor counts from the same design.

4.2 Multicore Processors

A multicore processor integrates multiple processor cores onto a single chip, sharing on-chip L3 cache and memory controllers. This dramatically reduces inter-core communication latency compared to separate physical processors.

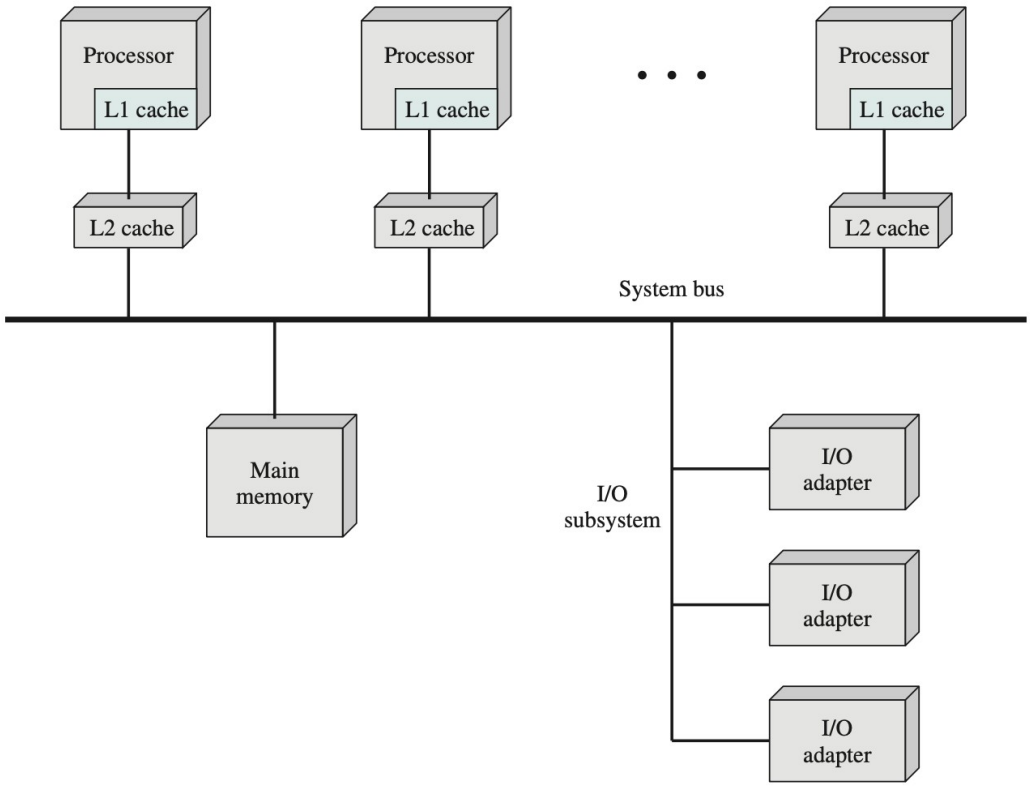

Figure: Intel Core i7 multicore die — 6 cores each with private L1/L2, shared L3 cache, memory controller (Stallings Fig. 1.22)

Attribute	Detail
Core count	Modern CPUs: 4–64 cores on one die (servers: up to 128 cores)
Cache hierarchy	L1 (32–64 KB, private, 1–4 cycle latency), L2 (256 KB–1 MB, private, 10–15 cycles), L3 (4–64 MB, shared, 30–50 cycles)
OS view	Each core appears as a separate logical CPU to the OS scheduler. A 6-core CPU = 6 logical CPUs (or 12 with Hyper-Threading)
Cache coherence	When Core 0 writes to address X, Core 1's cache may hold a stale copy. MESI protocol (Modified/Exclusive/Shared/Invalid) keeps caches coherent.

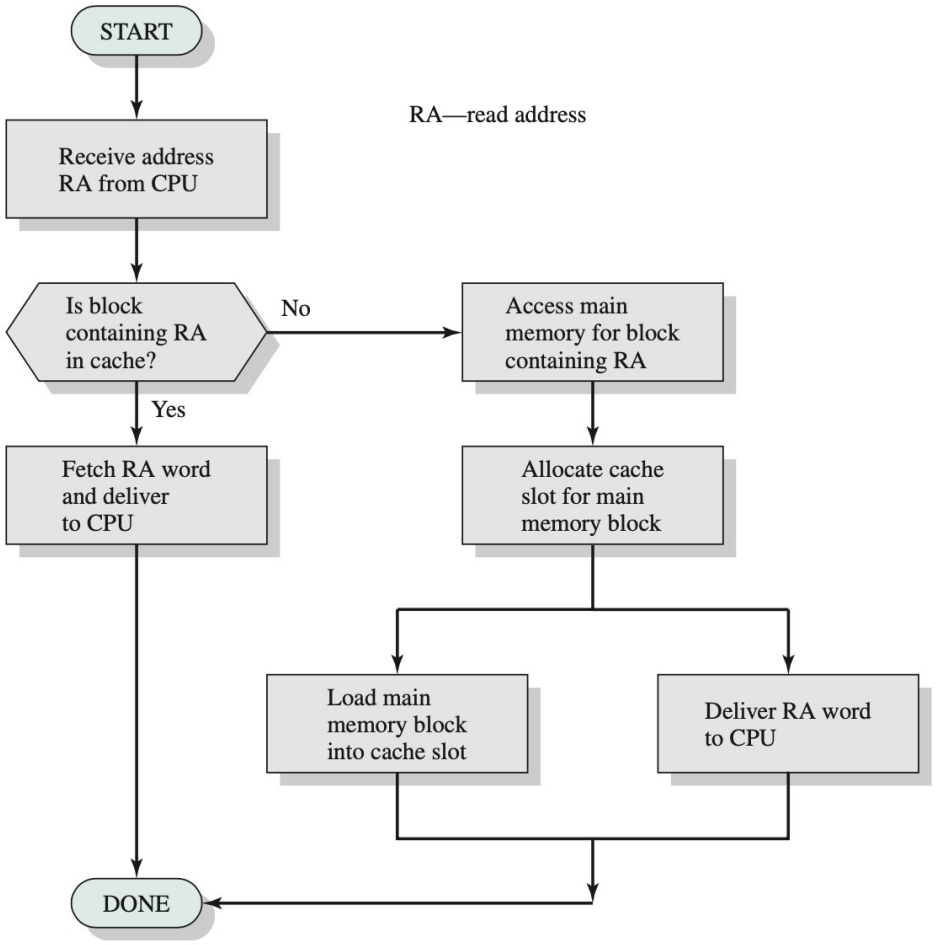

Figure: Memory hierarchy in a multicore system — per-core L1/L2 and shared L3 (Stallings Fig. 1.23)

Section 5 · Practice Exercises

✎ Exercise 2.1 [8 marks]

- A two-level memory system has T_1 = 0.5 ns (cache) and T_2 = 50 ns (main memory). The cache block size is 64 bytes.
- (a) [2] What hit ratio H is required to achieve T_avg = 2 ns? Show the working from the hit ratio formula.
- (b) [2] If H increases from 0.90 to 0.98, by what factor does T_avg improve?
- (c) [4] A program accesses addresses in the pattern: 100, 104, 108, 200, 100, 104, 108, 200, ... (repeating). With a cache block of 64 bytes, would accesses 100, 104, and 108 all hit the same cache block? Explain using block address = address DIV 64.

✎ Exercise 2.2 [10 marks]

- A direct-mapped cache has 512 lines. Main memory has 16 K (16,384) blocks of 64 bytes each. Memory addresses are 20 bits.
- (a) [2] How is a 20-bit memory address partitioned into tag, line index, and byte offset fields?

State the width of each field.

- (b) [3] To which cache line does main memory block 1,234 map? Which cache line does block 1,746 map to? Do they conflict?
- (c) [5] If a 4-way set-associative cache is used instead (still 512 total lines), how many sets are there? Redo part (b): which set do blocks 1,234 and 1,746 map to? Can both reside in cache simultaneously? Explain.

✎ Exercise 2.3 [12 marks]

- A disk transfer of 1 MB is performed using each I/O technique. The disk transfer rate is 100 MB/s. The CPU runs at 2 GHz. An interrupt requires 200 cycles to service.
- (a) [3] Programmed I/O (polling): Assume the CPU checks the status port every 10 ns. How many status checks are needed? What fraction of CPU time is consumed by polling during the transfer?
- (b) [3] Interrupt-driven I/O at 1 byte per interrupt: How many interrupts are generated? What is the total ISR overhead in CPU cycles? Express as a percentage of CPU time during the 10 ms transfer.
- (c) [3] DMA: Only 1 interrupt is generated. How many CPU cycles are consumed by the interrupt? Express as a percentage of CPU time during the 10 ms transfer.
- (d) [3] Based on your calculations, justify which technique is most appropriate for: (i) a keyboard input, (ii) a 4 GB database file copy, (iii) a hardware sensor sampling at 10 Hz.

✎ Exercise 2.4 [10 marks]

- A multicore processor has 4 cores. Each core has a private L1 cache (64 KB, 1 ns), a private L2 cache (512 KB, 5 ns), and all cores share an L3 cache (8 MB, 20 ns). Main memory access time is 80 ns.
- (a) [3] Core 0 reads address 0x1000. L1 miss, L2 miss, L3 hit. What is the total access time for this reference?
- (b) [3] Core 0 writes to address 0x2000 (L1 hit, write-back policy). Core 1 subsequently reads address 0x2000 (L1 miss). Describe the cache coherence problem and explain how the MESI protocol resolves it.
- (c) [4] Amdahl's Law: A program runs on this 4-core system. 20% of the code is serial (cannot be parallelised). What is the maximum speedup? What speedup is actually achieved if the parallelisable portion achieves perfect speedup on 4 cores? Show all working.