Memory

Eri Prasetyo Wibowo

http://eri.staff.gunadarma.ac.id

Semiconductor

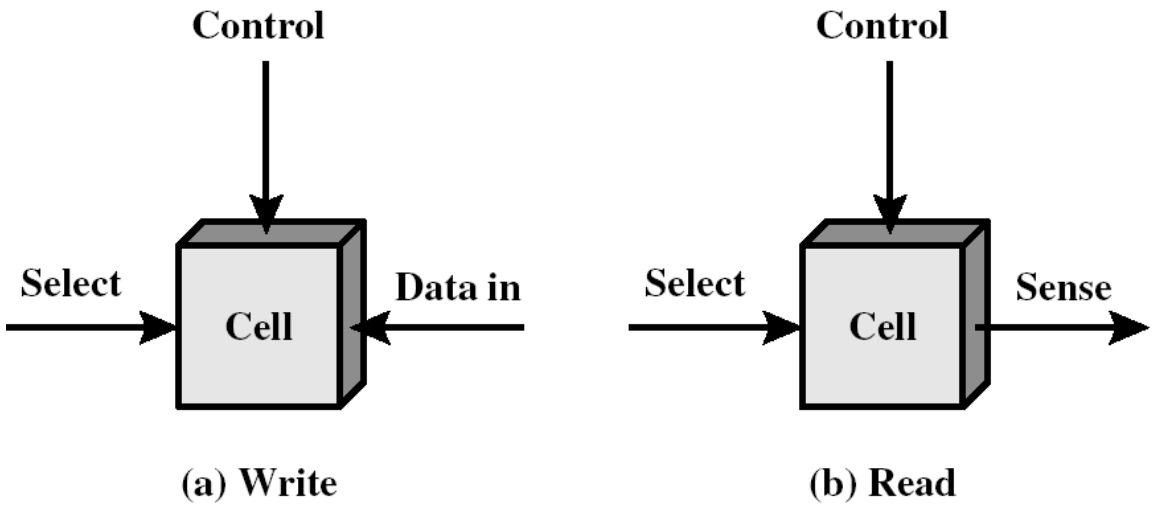
Table 5.1 Semiconductor Memory Types

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)				
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level	Electrically	
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

Semiconductor Memory

RAM

- Misnamed as all semiconductor memory is random access
- Read/Write
- Volatile
- Temporary storage
- Static or dynamic



Two basic types devised:

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Static RAM faster operation than dynamic RAM (around 10 times faster) but requires more internal components (see next).

Hence DRAM has more memory cells in chip (around 4-8 times capacity).

Usually main memory is dynamic and cache memory usually static.

Random-Access Memory (RAM)

Key features

- **RAM adalah sebagai kemasan chip.**
- **Dasar penyimpanan unit adalah sel (satu bit per sel).**
- **Beberapa chip RAM membentuk memori**

Static RAM (SRAM)

- **Setiap sel menyimpan bit dengan enam transistor-sirkuit.**
- **Nilai tetap tentu, selama ini disimpan daya.**
- **Relatif kebal untuk gangguan listrik seperti kebisingan.**
- **Lebih cepat dan lebih mahal daripada DRAM.**

Dynamic RAM (DRAM)

- **Setiap sel menyimpan bit dengan kapasitor dan transistor.**
- **Nilai harus refresh setiap 10-100 ms.**
- **Sensitif terhadap gangguan.**
- **Lambat dan lebih murah dibandingkan SRAM.**

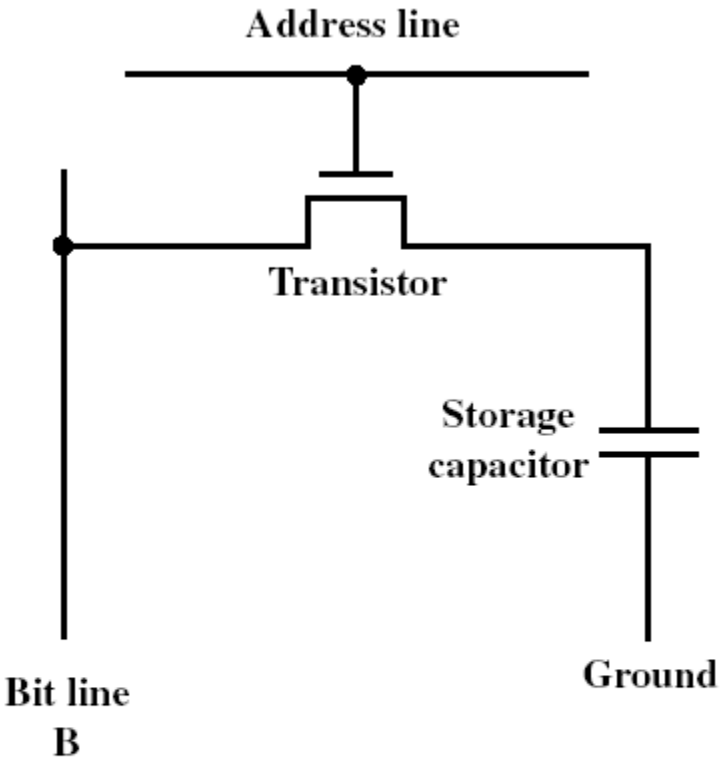
SRAM vs DRAM summary

	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers

Dynamic RAM

Bits stored as charge in capacitors
Charges leak
Need refreshing even when powered
Simpler construction
Smaller per bit
Less expensive
Need refresh circuits
Slower
Main memory
Essentially analogue

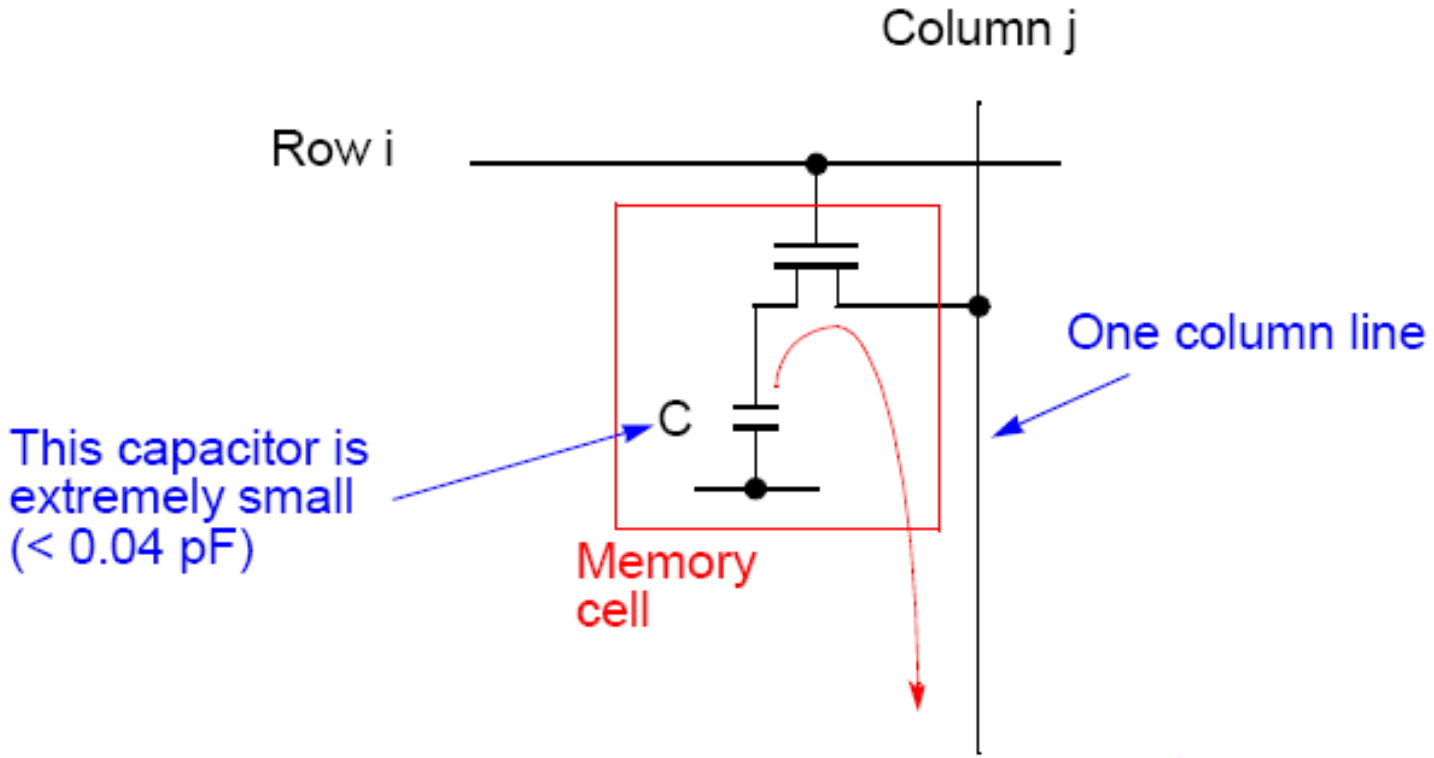
- Level of charge determines value



(a) Dynamic RAM (DRAM) cell

Dynamic RAM cell circuit

Each memory cell uses a capacitor to store charge. Two states, charged and not charged represent the two binary values 0 and 1. Less components, higher memory density. Usual design:



Because charge will decay, memory cell must be refreshed periodically, typically every 2-4 ms. Refresh consists of reading data and rewriting it. Usually done automatically within chip.

DRAM Operation

Address line active when bit read or written

- Transistor switch closed (current flows)

Write

- Voltage to bit line
  - **High for 1 low for 0**
- Then signal address line
  - **Transfers charge to capacitor**

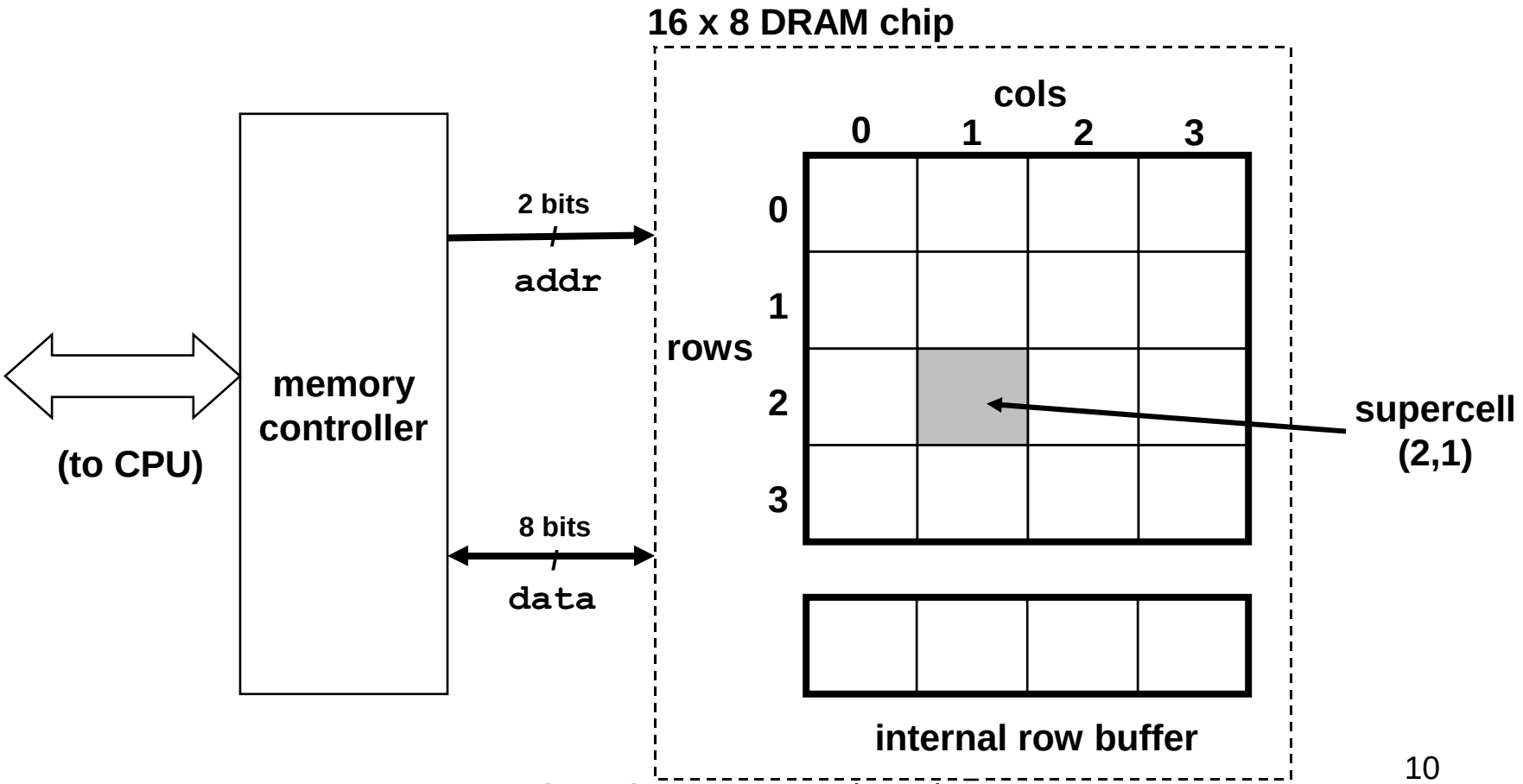
Read

- Address line selected
  - **transistor turns on**
- Charge from capacitor fed via bit line to sense amplifier
  - **Compares with reference value to determine 0 or 1**
- Capacitor charge must be restored

Conventional DRAM organization

d x w DRAM:

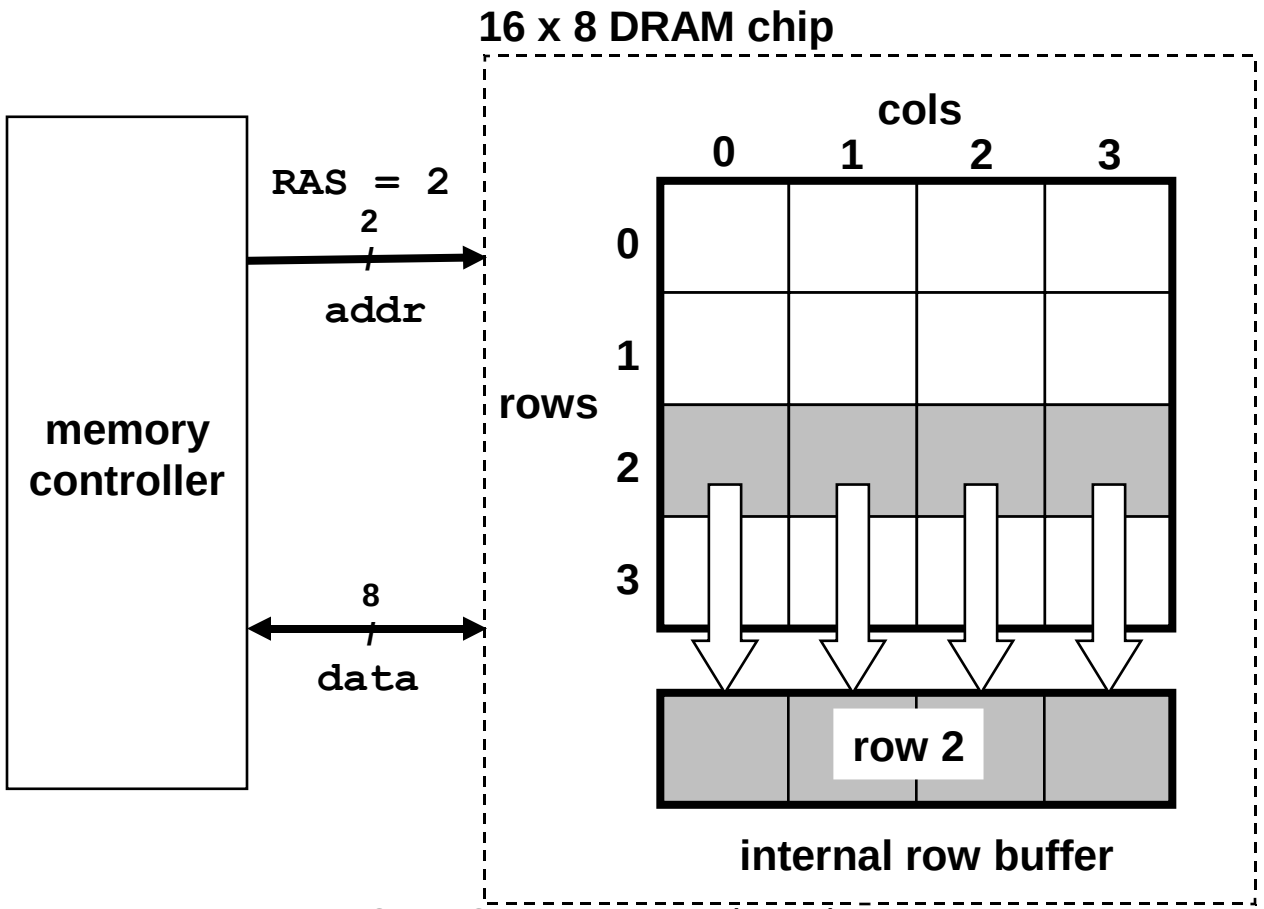
- dw total bits organized as *d* supercells of size *w* bits



Reading DRAM supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

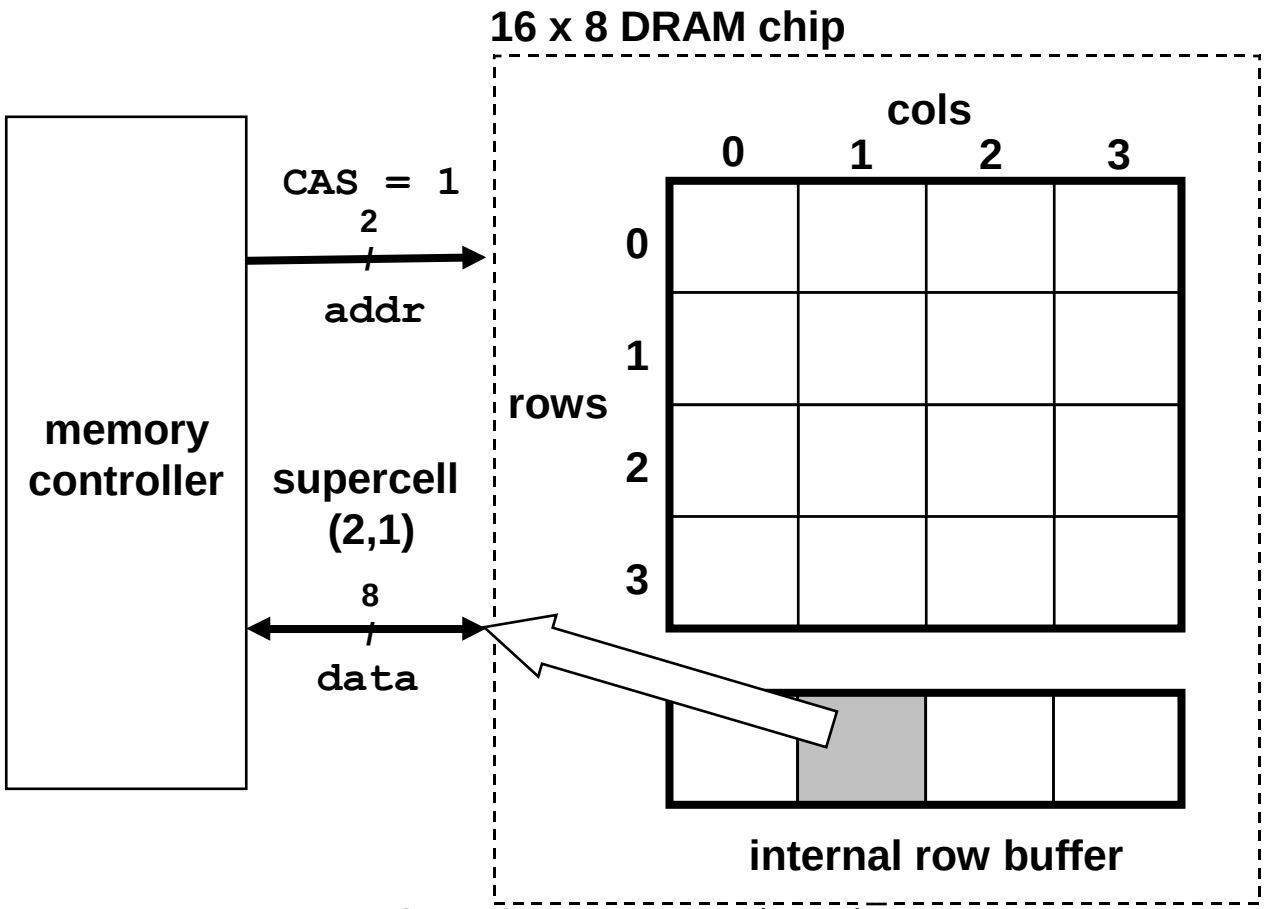
Step 1(b): Row 2 copied from DRAM array to row buffer.



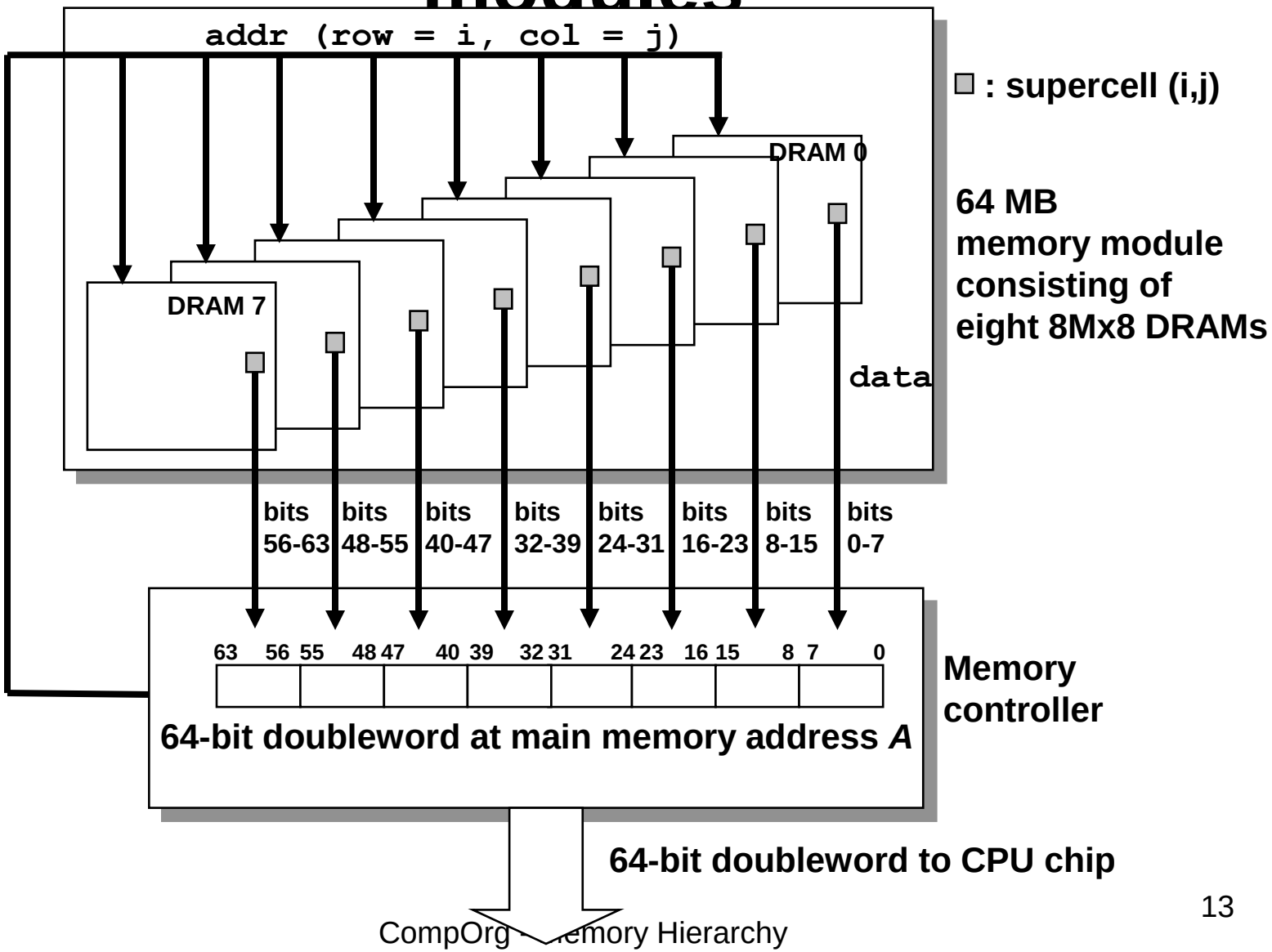
Reading DRAM supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



Memory modules



Enhanced DRAMs

All enhanced DRAMs are built around the conventional DRAM core.

- **Fast page mode DRAM (FPM DRAM)**
  - Access contents of row with [RAS, CAS, CAS, CAS, CAS] instead of [(RAS,CAS), (RAS,CAS), (RAS,CAS), (RAS,CAS)].
- **Extended data out DRAM (EDO DRAM)**
  - Enhanced FPM DRAM with more closely spaced CAS signals.
- **Synchronous DRAM (SDRAM)**
  - Driven with rising clock edge instead of asynchronous control signals.
- **Double data-rate synchronous DRAM (DDR SDRAM)**
  - Enhancement of SDRAM that uses both clock edges as control signals.
- **Video RAM (VRAM)**
  - Like FPM DRAM, but output is produced by shifting row buffer
  - Dual ported (allows concurrent reads and writes)

Static RAM

Bits stored as on/off switches

No charges to leak

No refreshing needed when powered

More complex construction

Larger per bit

More expensive

Does not need refresh circuits

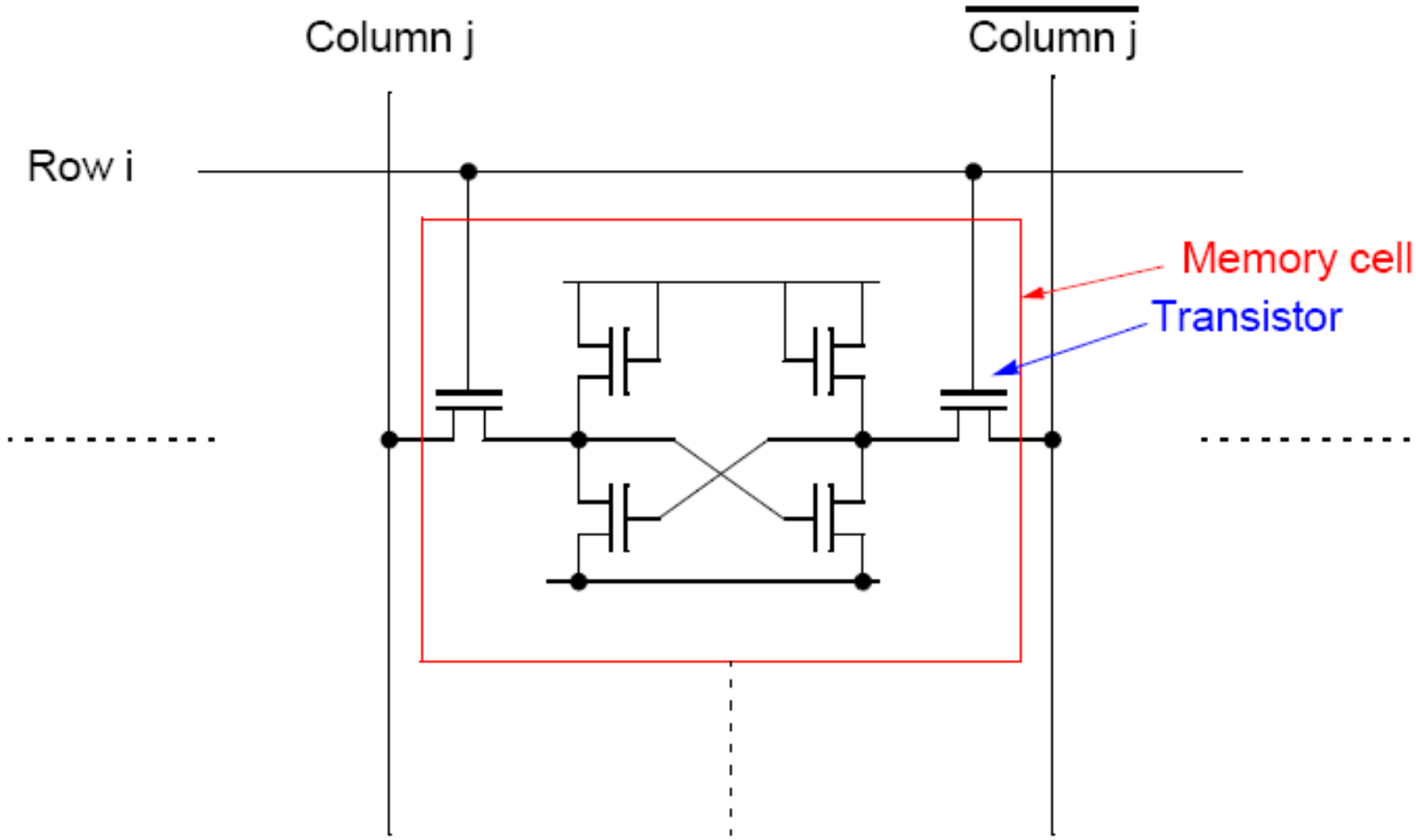
Faster

Cache

Digital

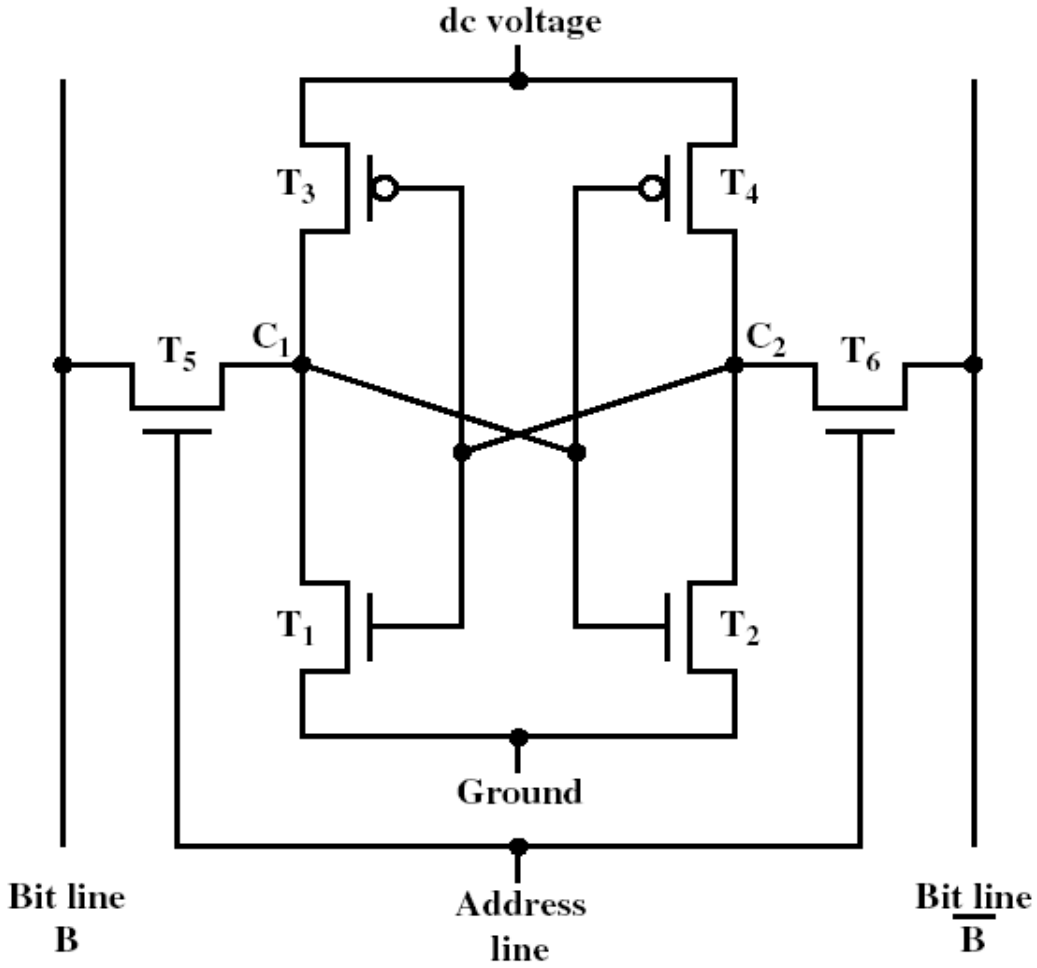
- Uses flip-flops

Static RAM Memory Cell Circuit



Design requires 4 or 6 transistors and two columns select lines per column.
Do not worry about its operation. Key point is number of components.

Stat



(b) Static RAM (SRAM) cell

Static RAM Operation

Transistor arrangement gives stable logic state

State 1

- C_1 high, C_2 low
- T_1 T_4 off, T_2 T_3 on

State 0

- C_2 high, C_1 low
- T_2 T_3 off, T_1 T_4 on

Address line transistors T_5 T_6 is switch

Write – apply value to B & compliment to B

Read – value is on line B

SRAM vs DRAM

Both volatile

- Power needed to preserve data

Dynamic cell

- Simpler to build, smaller
- More dense
- Less expensive
- Needs refresh
- Larger memory units

Static

- Faster
- Cache

Nonvolatile memories

DRAM and SRAM are *volatile* memories

- **Lose information if powered off.**

***Nonvolatile memories* retain value even if powered off.**

- **Generic name is read-only memory (ROM).**
- **Misleading because some ROMs can be read and modified.**

Types of ROMs

- **Programmable ROM (PROM)**
- **Eraseable programmable ROM (EPROM)**
- **Electrically eraseable PROM (EEPROM)**
- **Flash memory**

Firmware

- **Program stored in a ROM**
  - Boot time code, BIOS (basic input/ouput system)
  - graphics cards, disk controllers.

Read Only Memory (ROM)

Permanent storage

- Nonvolatile

Microprogramming (see later)

Library subroutines

Systems programs (BIOS)

Function tables

Types of ROM

Written during manufacture

- Very expensive for small runs

Programmable (once)

- PROM
- Needs special equipment to program

Read “mostly”

- Erasable Programmable (EPROM)
  - **Erased by UV**
- Electrically Erasable (EEPROM)
  - **Takes much longer to write than read**
- Flash memory
  - **Erase whole memory electrically**

Organisation in detail

A 16Mbit chip can be organised as 1M of 16 bit words

A bit per chip system has 16 lots of 1Mbit chip with bit 1 of each word in chip 1 and so on

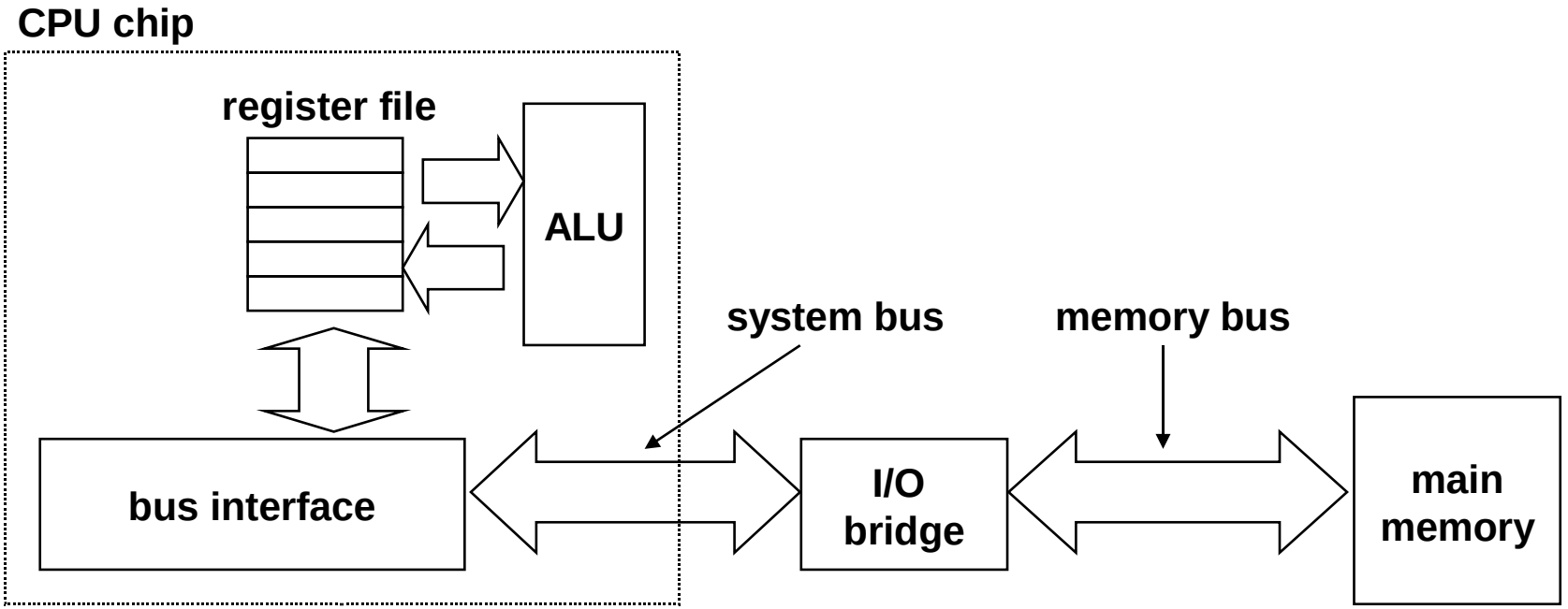
A 16Mbit chip can be organised as a 2048 x 2048 x 4bit array

- Reduces number of address pins
  - **Multiplex row address and column address**
  - **11 pins to address (2^{11}=2048)**
  - **Adding one more pin doubles range of values so x4 capacity (2^{12} x4 Capacity with 2^{11})**

Bus structure connecting CPU and memory

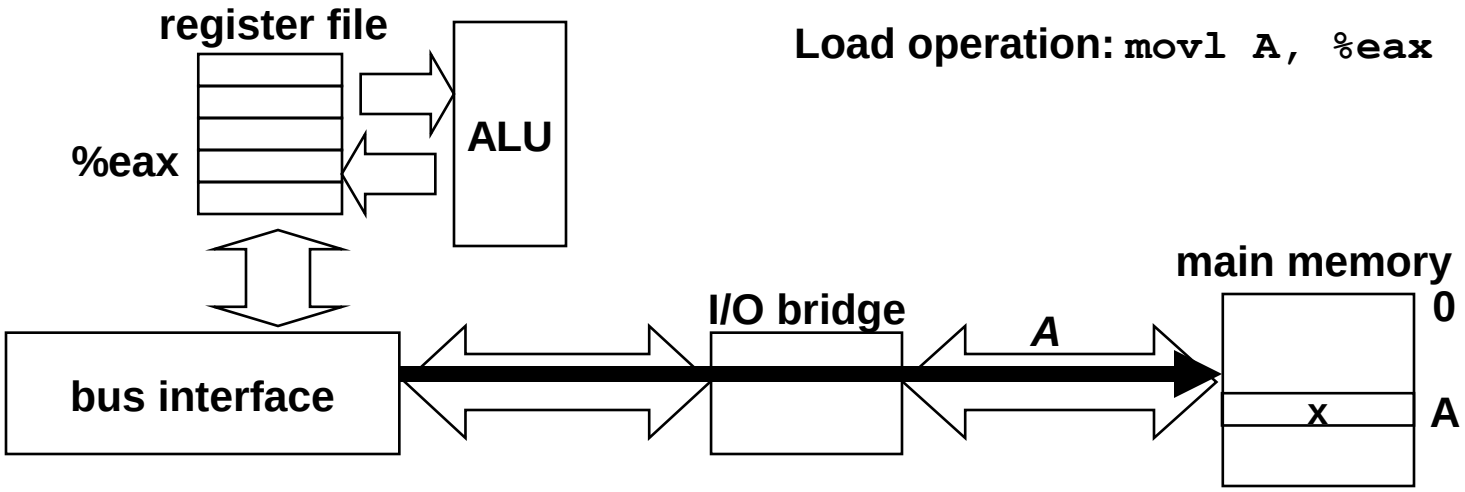
A *bus* is a collection of parallel wires that carry address, data, and control signals.

Buses are typically shared by multiple devices.



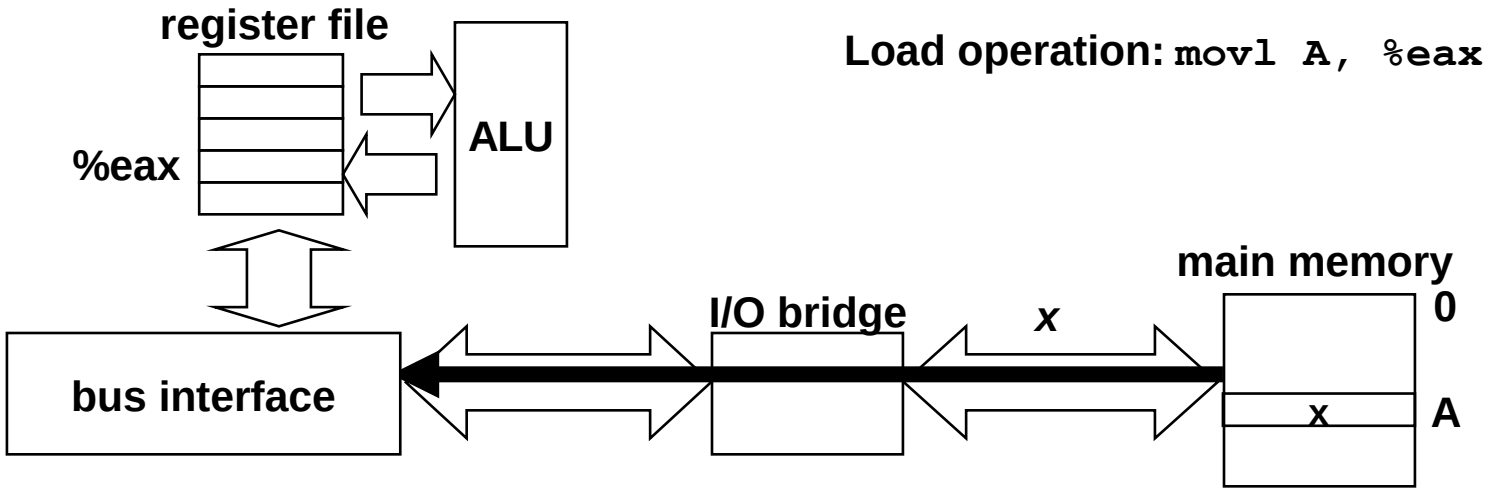
Memory read transaction (1)

CPU places address `A` on the memory bus.



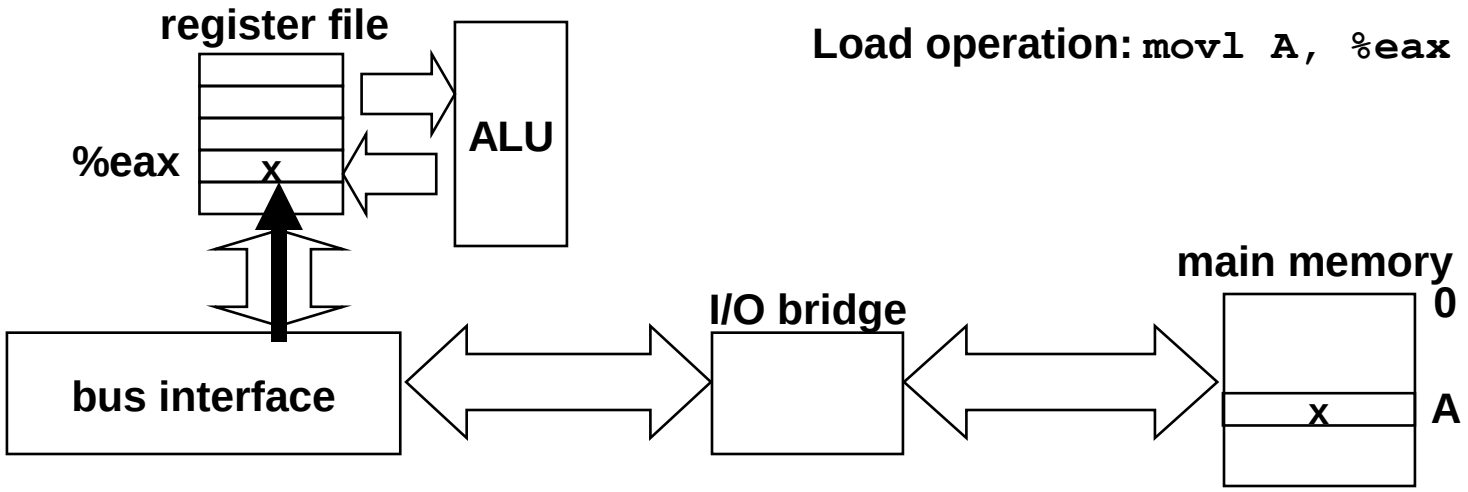
Memory read transaction (2)

Main memory reads `A` from the memory bus, retreives word `x`, and places it on the bus.



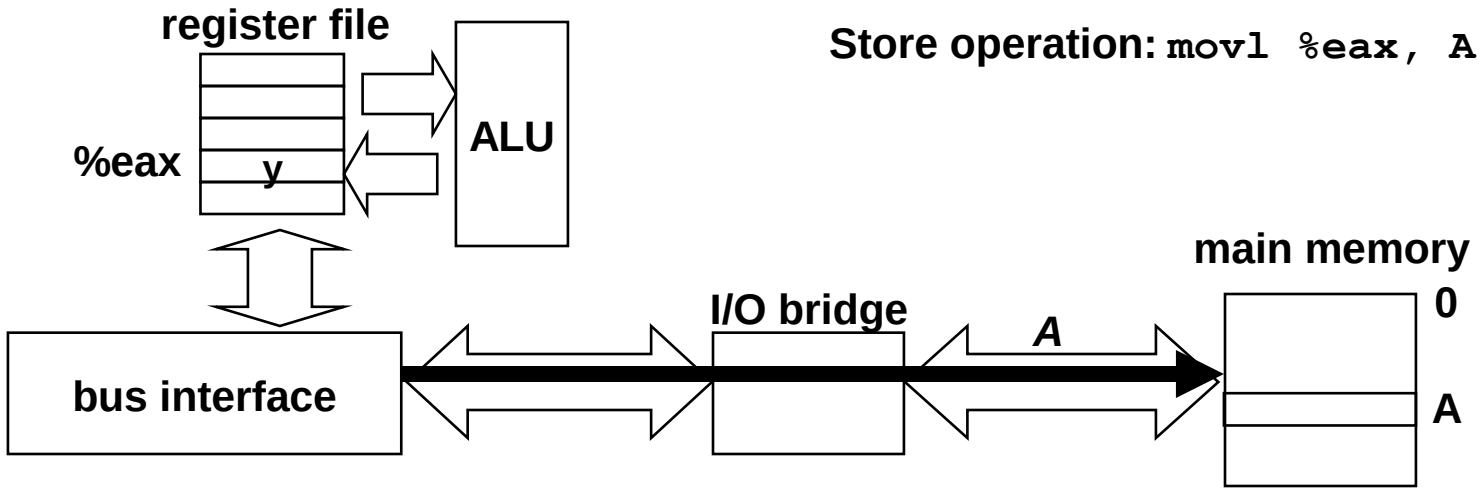
Memory read transaction (3)

CPU read word `x` from the bus and copies it into register `%eax`.



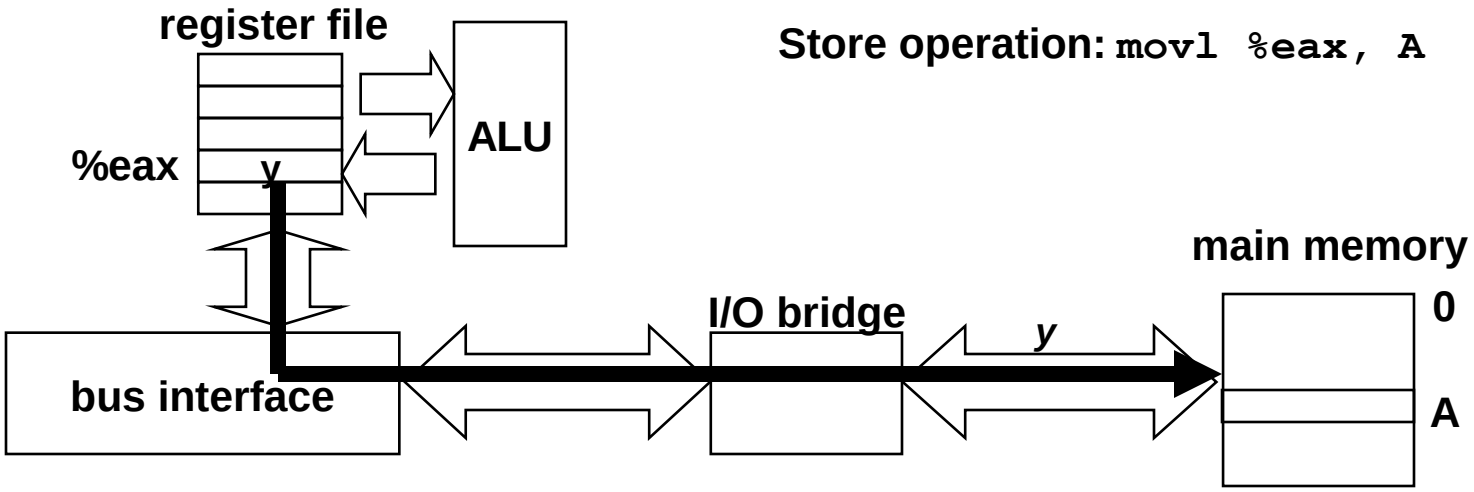
Memory write transaction (1)

CPU places address `A` on the bus. Main memory reads it and waits for the corresponding data word to arrive.



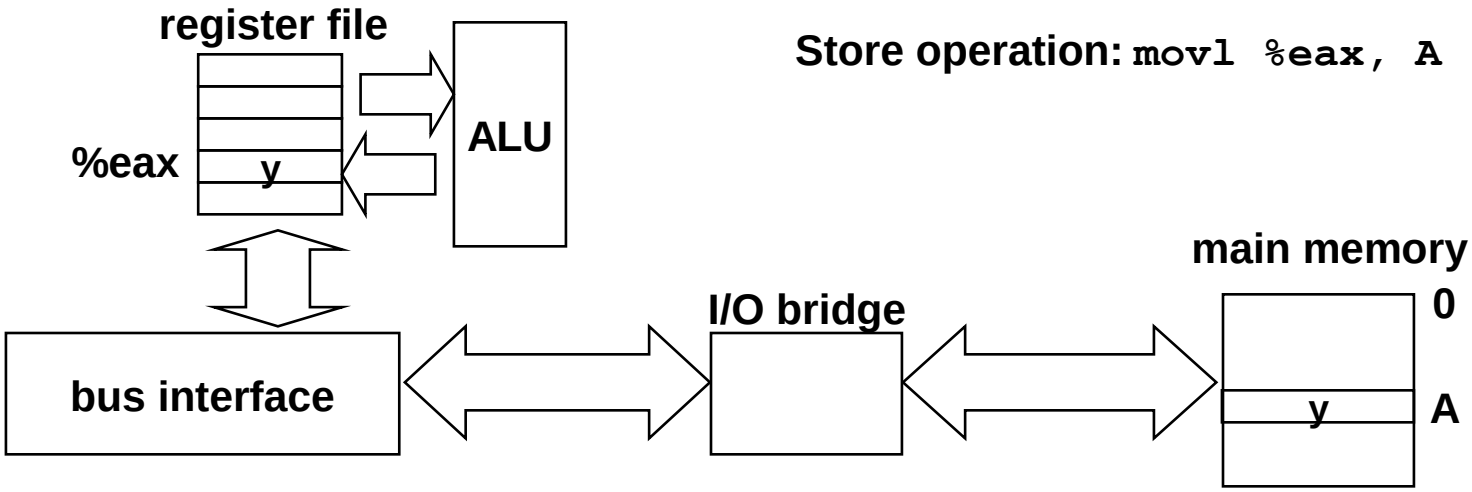
Memory write transaction (2)

CPU places data word `y` on the bus.



Memory write transaction (3)

Main memory read data word `y` from the bus and stores it at address `A`.

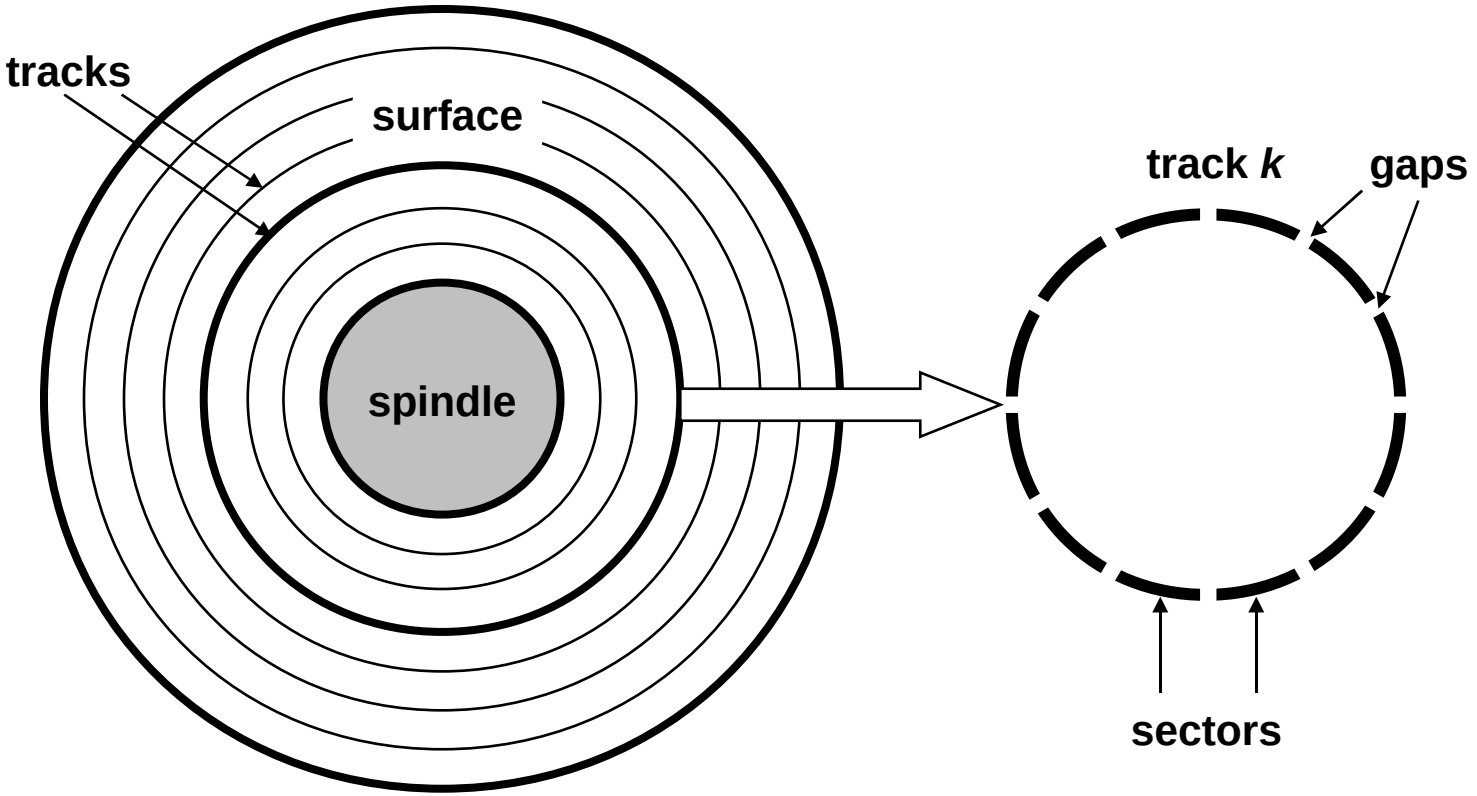


Disk geometry

Disks **consist of** ***platters*****, each with two** ***surfaces*****.**

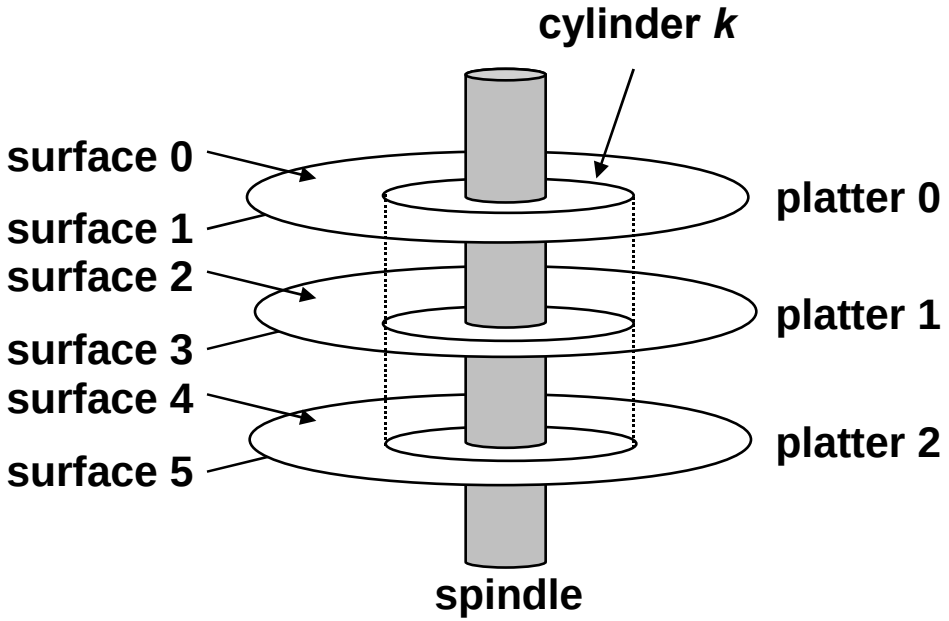
Each ***surface*** **consists of concentric rings called** ***tracks*****.**

Each ***track*** **consists of** ***sectors*** **separated by** ***gaps*****.**



Disk geometry (muliple-platter view)

Aligned tracks form a *cylinder.*



Disk capacity

Capacity: **maximum number of bits that can be stored.**

- **Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^6.**

Capacity is determined by these technology factors:

- ***Recording density (bits/in):*** **number of bits that can be squeezed into a 1 inch segment of a track.**
- ***Track density (tracks/in):*** **number of tracks that can be squeezed into a 1 inch radial segment.**
- ***Areal density (bits/in^2):*** **product of recording and track density.**

Modern disks partition tracks into disjoint subsets called *recording zones*

- **Each track in a zone has the same number of sectors, determined by the circumference of innermost track.**
- **Each zone has a different number of sectors/track**

Computing disk capacity

Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)

Example:

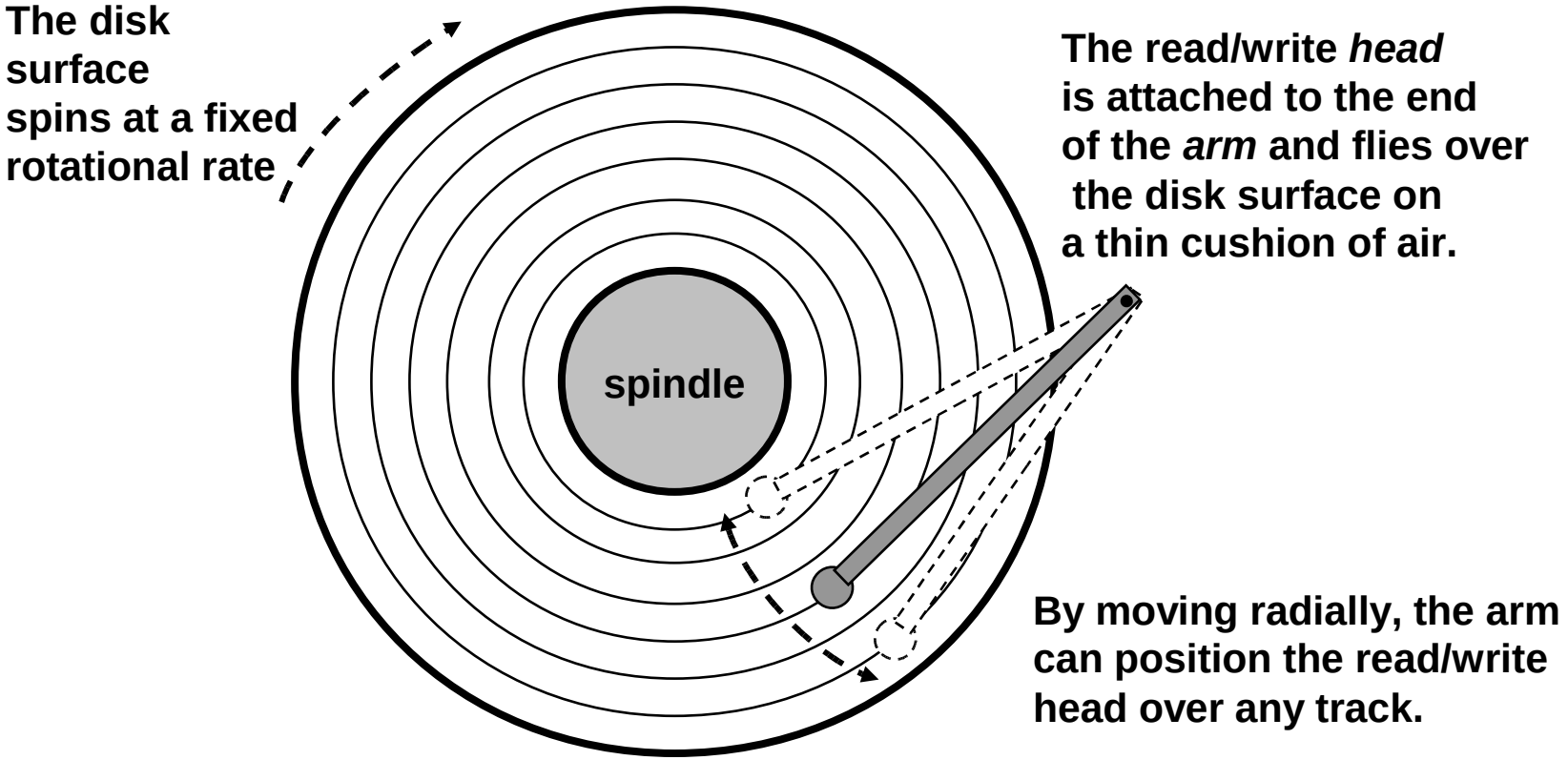
- **512 bytes/sector**
- **300 sectors/track (on average)**
- **20,000 tracks/surface**
- **2 surfaces/platter**
- **5 platters/disk**

Capacity = 512 x 300 x 20000 x 2 x 5

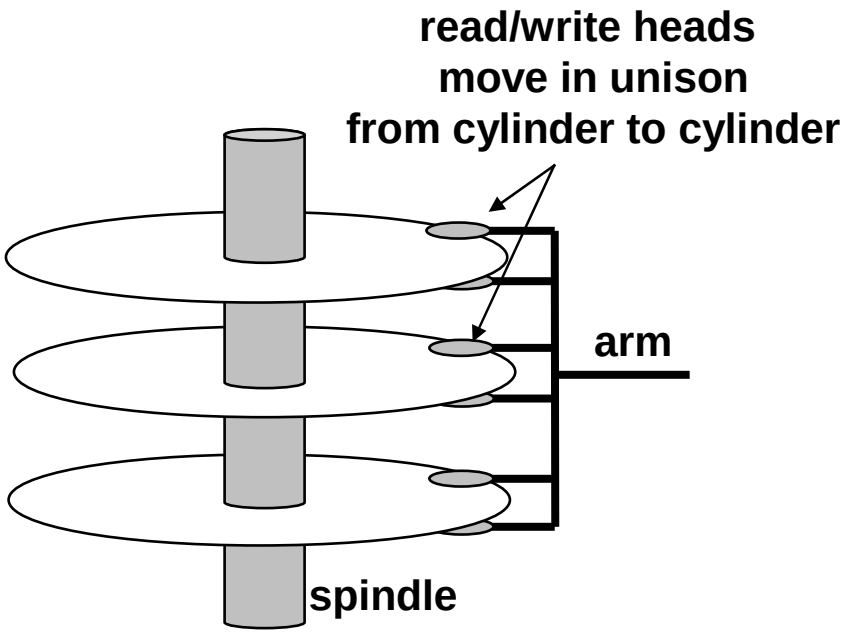
= 30,720,000,000

= 30.72 GB

Disk operation (single-platter view)



Disk operation (multi-platter view)



Disk access time

Average time to access some target sector approximated by :

- $T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$

Seek time

- **Time to position heads over cylinder containing target sector.**
- **Typical $T_{avg\ seek}$ = 9 ms**

Rotational latency

- **Time waiting for first bit of target sector to pass under r/w head.**
- **$T_{avg\ rotation}$ = 1/2 x 1/RPMs x 60 sec/1 min**

Transfer time

- **Time to read the bits in the target sector.**
- **$T_{avg\ transfer}$ = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min.**

Disk access time example

Given:

- **Rotational rate = 7,200 RPM**
- **Average seek time = 9 ms.**
- **Avg # sectors/track = 400.**

Derived:

- **$T_{avg\ rotation}$ = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.**
- **$T_{avg\ transfer}$ = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms**
- **T_{access} = 9 ms + 4 ms + 0.02 ms**

Important points:

- **Access time dominated by seek time and rotational latency.**
- **First bit in a sector is the most expensive, the rest are free.**
- **SRAM access time is about 4ns/doubleword, DRAM about 60 ns**
  - Disk is about 40,000 times slower than SRAM,
  - 2,500 times slower then DRAM.

Logical disk blocks

Modern disks present a simpler abstract view of the complex sector geometry:

- **The set of available sectors is modeled as a sequence of b-sized *logical blocks* (0, 1, 2, ...)**

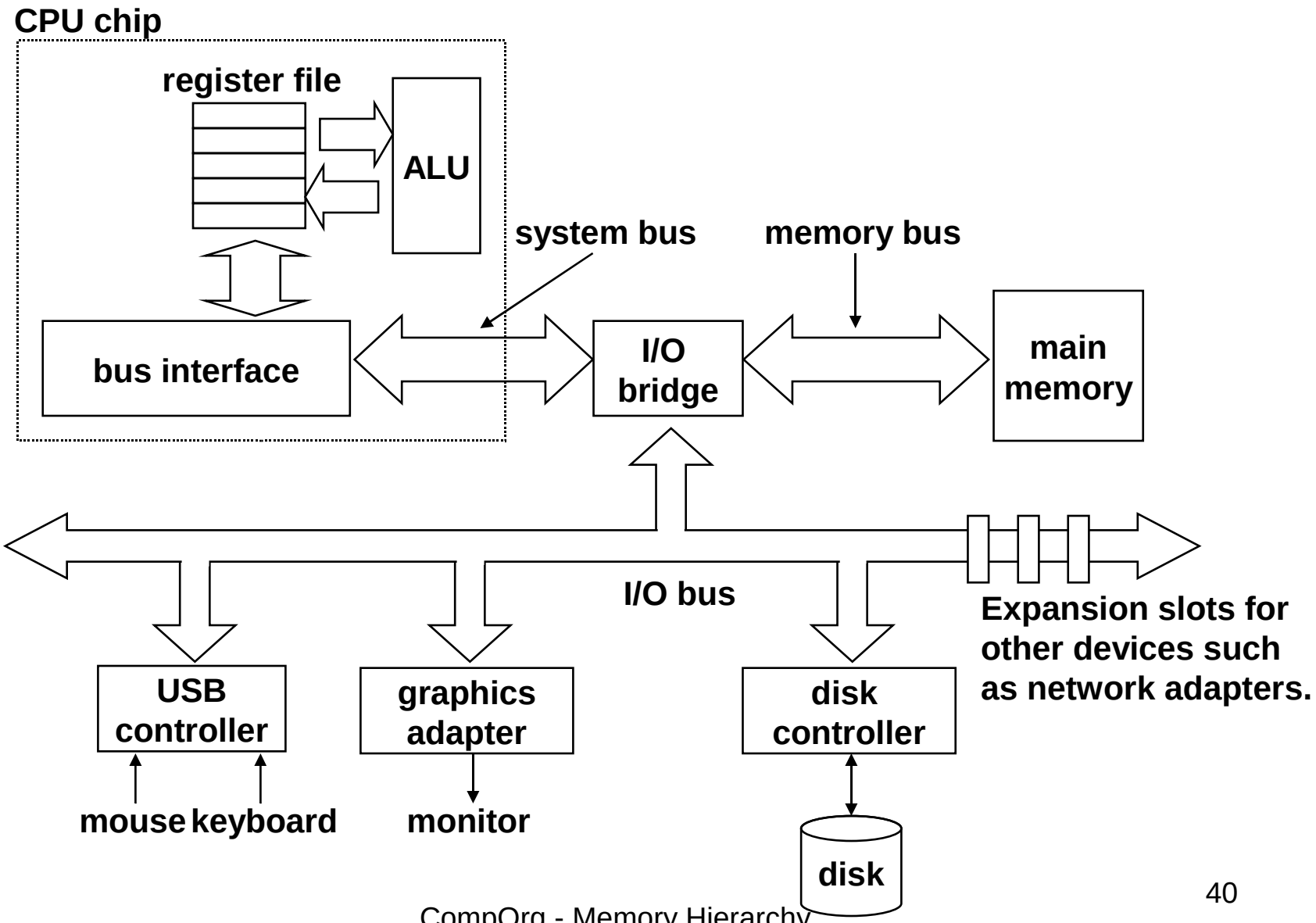
Mapping between logical blocks and actual (physical) sectors

- **Maintained by hardware/firmware device called disk controller.**
- **Converts requests for logical blocks into (surface,track,sector) triples.**

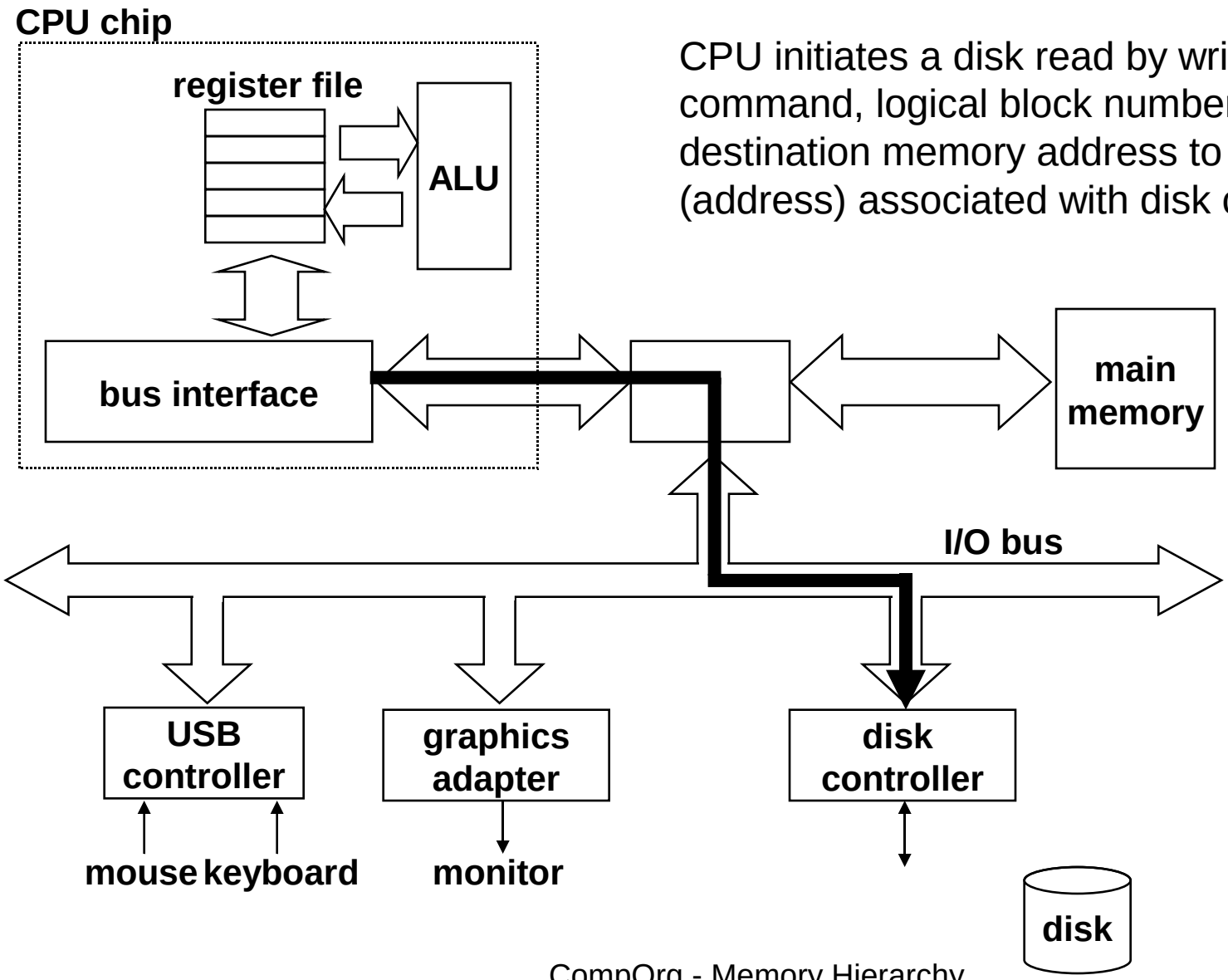
Allows controller to set aside spare cylinders for each zone.

- **Accounts for the difference in “formatted capacity” and “maximum capacity”.**

Bus structure connecting I/O and CPU

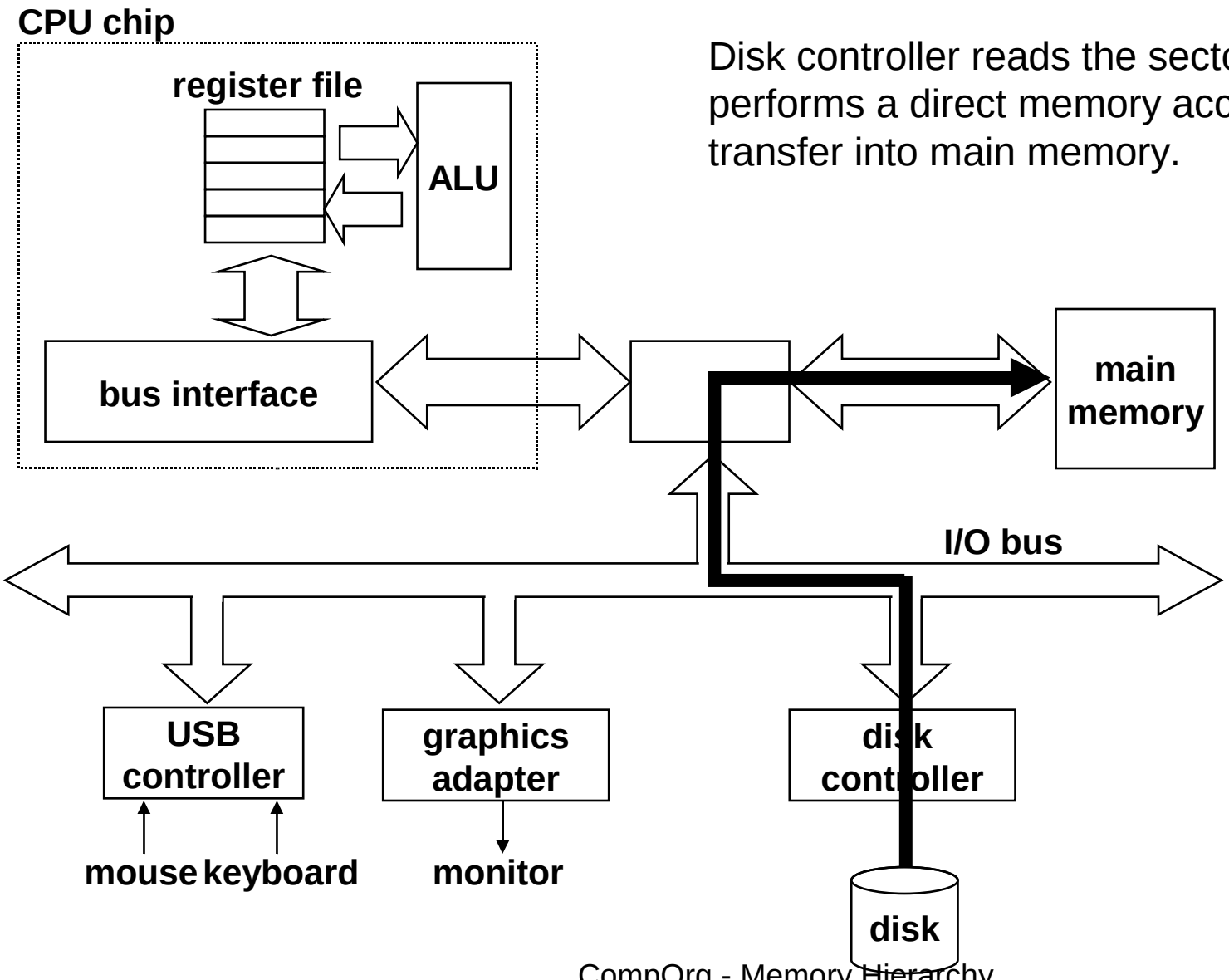


Reading a disk sector (1)



CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

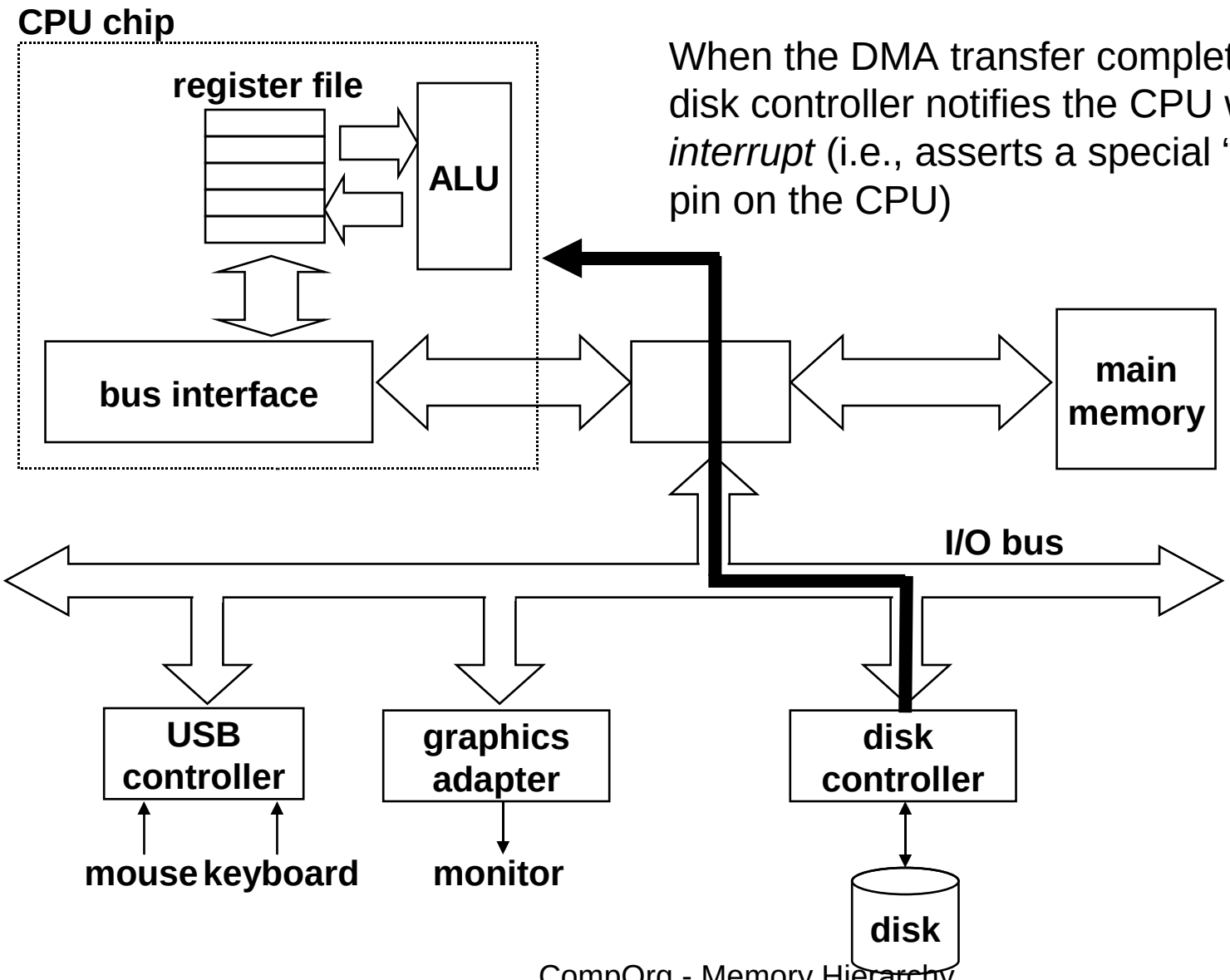
Reading a disk sector (2)



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Reading a disk sector (3)

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special “interrupt” pin on the CPU)



Storage trends

SRAM

metric	1980	1985	1990	1995	2000	*2000:1980*
$/MB	19,200	2,900	320	256	100	*190*
access (ns)	300	150	35	15	2	*100*

DRAM

metric	1980	1985	1990	1995	2000	*2000:1980*
$/MB	8,000	880	100	30	1	*8,000*
access (ns)	375	200	100	70	60	*6*
typical size(MB)	0.064	0.256	4	16	64	*1,000*

Disk

metric	1980	1985	1990	1995	2000	*2000:1980*
$/MB	500	100	8	0.30	0.05	*10,000*
access (ms)	87	75	28	10	8	*11*
typical size(MB)	1	10	160	1,000	9,000	*9,000*

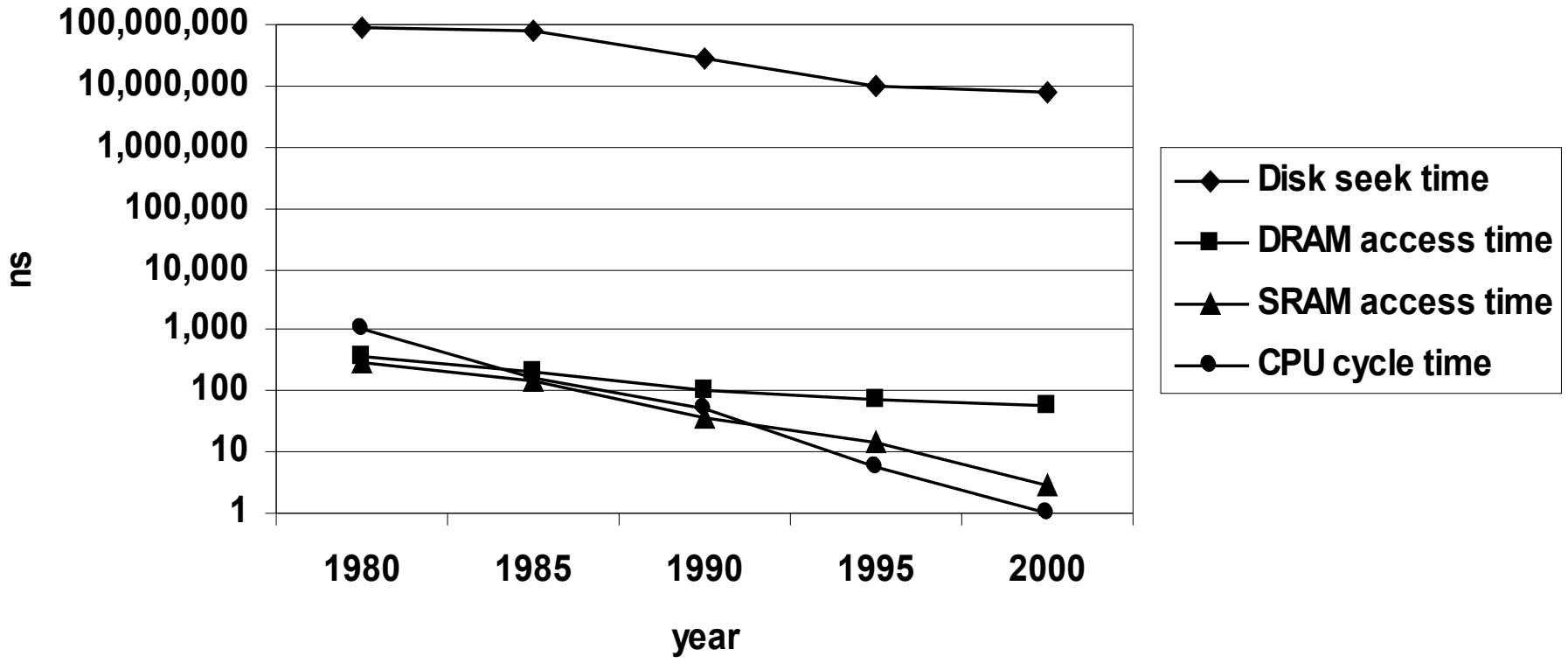
(Culled from back issues of Byte and PC Magazine)

CPU clock rates

	1980	1985	1990	1995	2000	*2000:1980*
processor	**8080**	**286**	**386**	**Pent**	**P-III**	
clock rate(MHz)	**1**	**6**	**20**	**150**	**750**	**750**
cycle time(ns)	**1,000**	**166**	**50**	**6**	**1.6**	**750**

The CPU-Memory Gap

The increasing gap between DRAM, disk, and CPU speeds.



Memory hierarchies

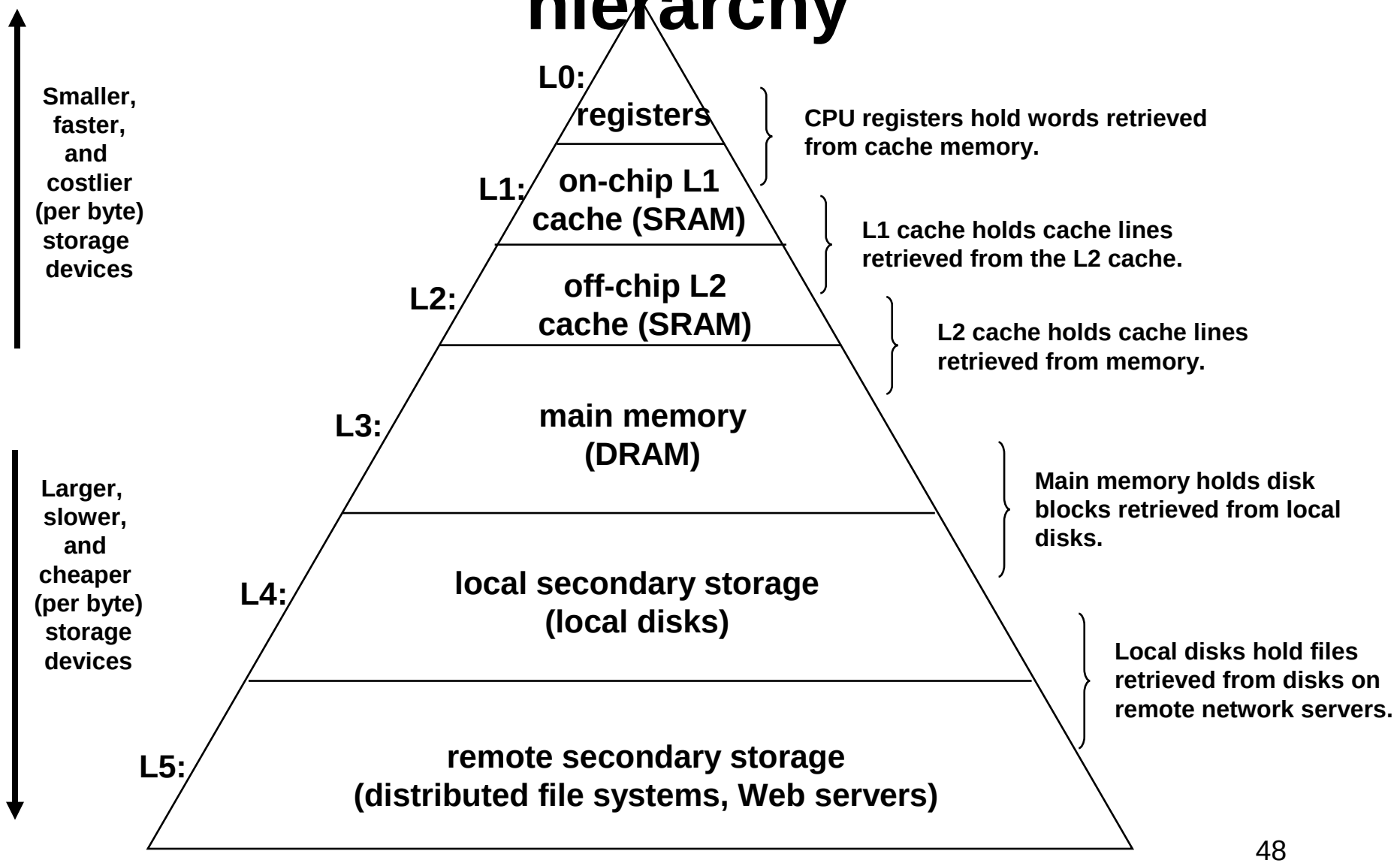
Some fundamental and enduring properties of hardware and software:

- **Fast storage technologies cost more per byte and have less capacity.**
- **The gap between CPU and main memory speed is widening.**
- **Well-written programs tend to exhibit good locality.**

These fundamental properties complement each other beautifully.

Suggest an approach for organizing memory and storage systems known as a “memory hierarchy”.

An example memory hierarchy



Caches

Cache: **A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.**

Fundamental idea of a memory hierarchy:

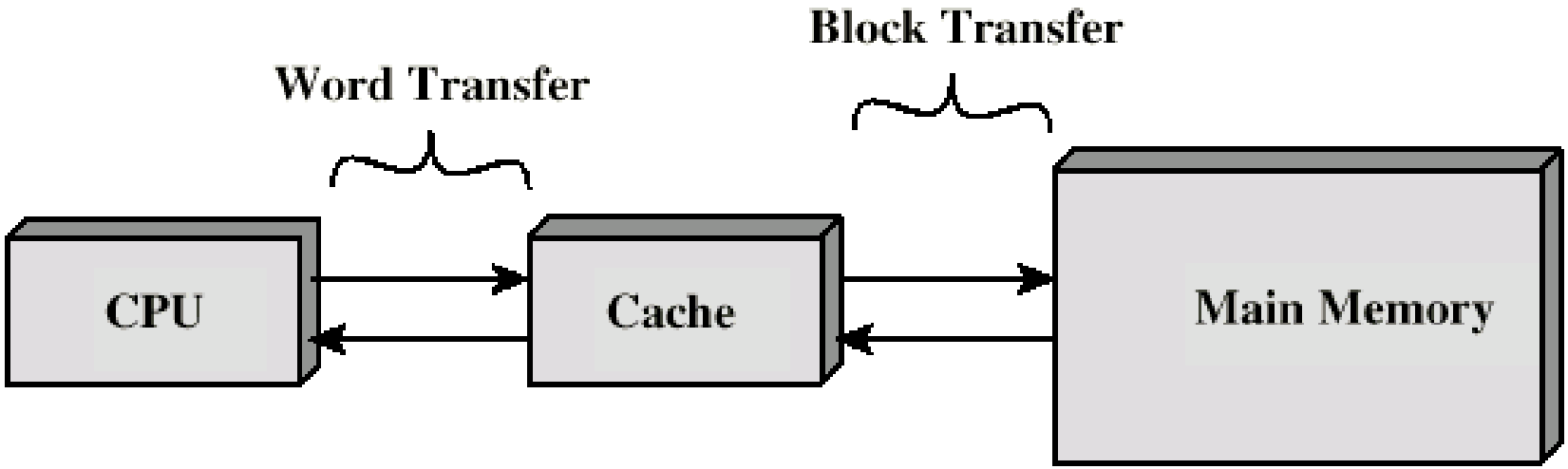
- **For each *k*, the faster, smaller device at level *k* serves as a cache for the larger, slower device at level *k+1*.**

Why do memory hierarchies work?

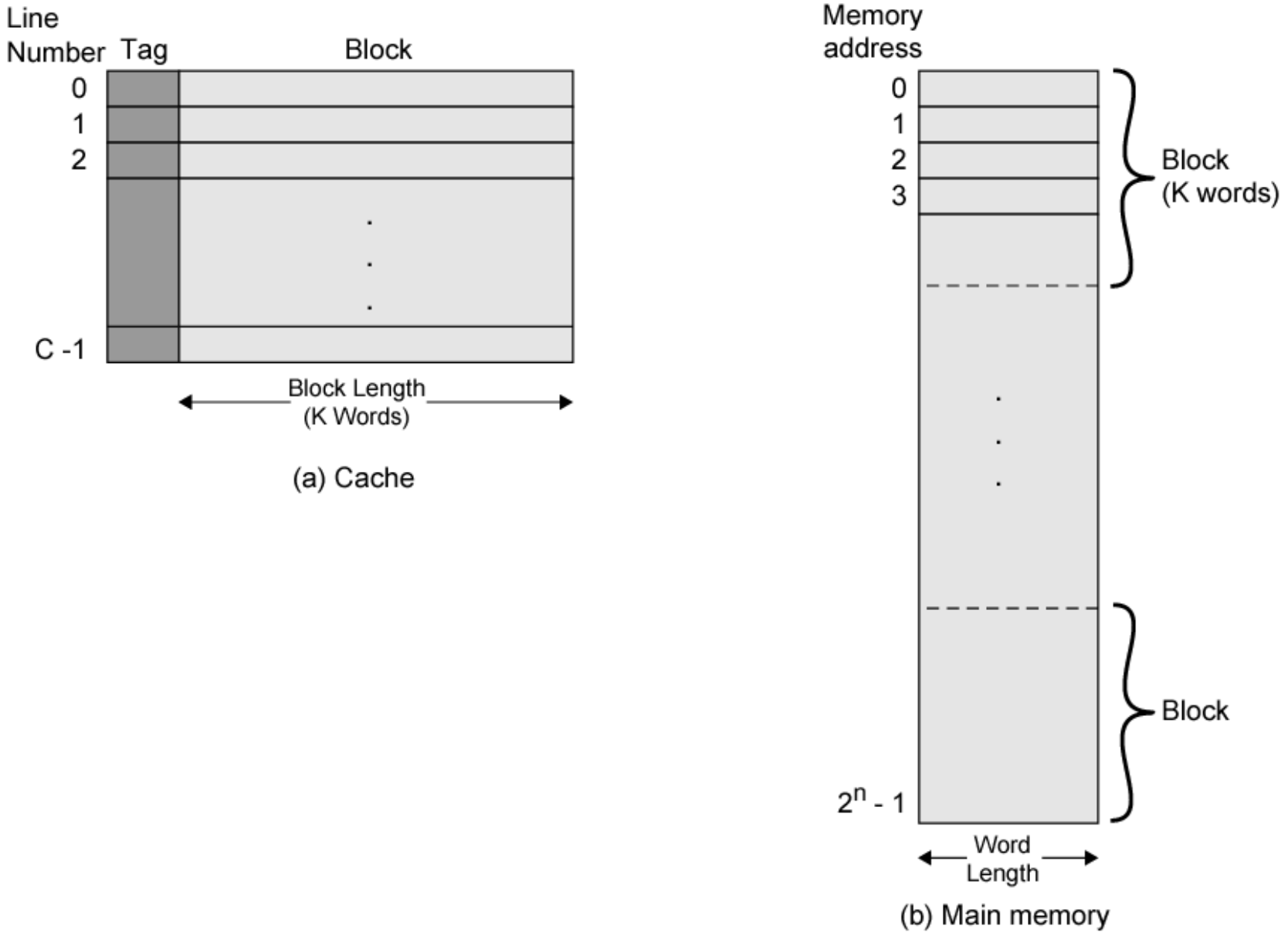
- **Programs tend to access the data at level k more often than they access the data at level k+1.**
- **Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.**
- **Net effect: A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.**

Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module



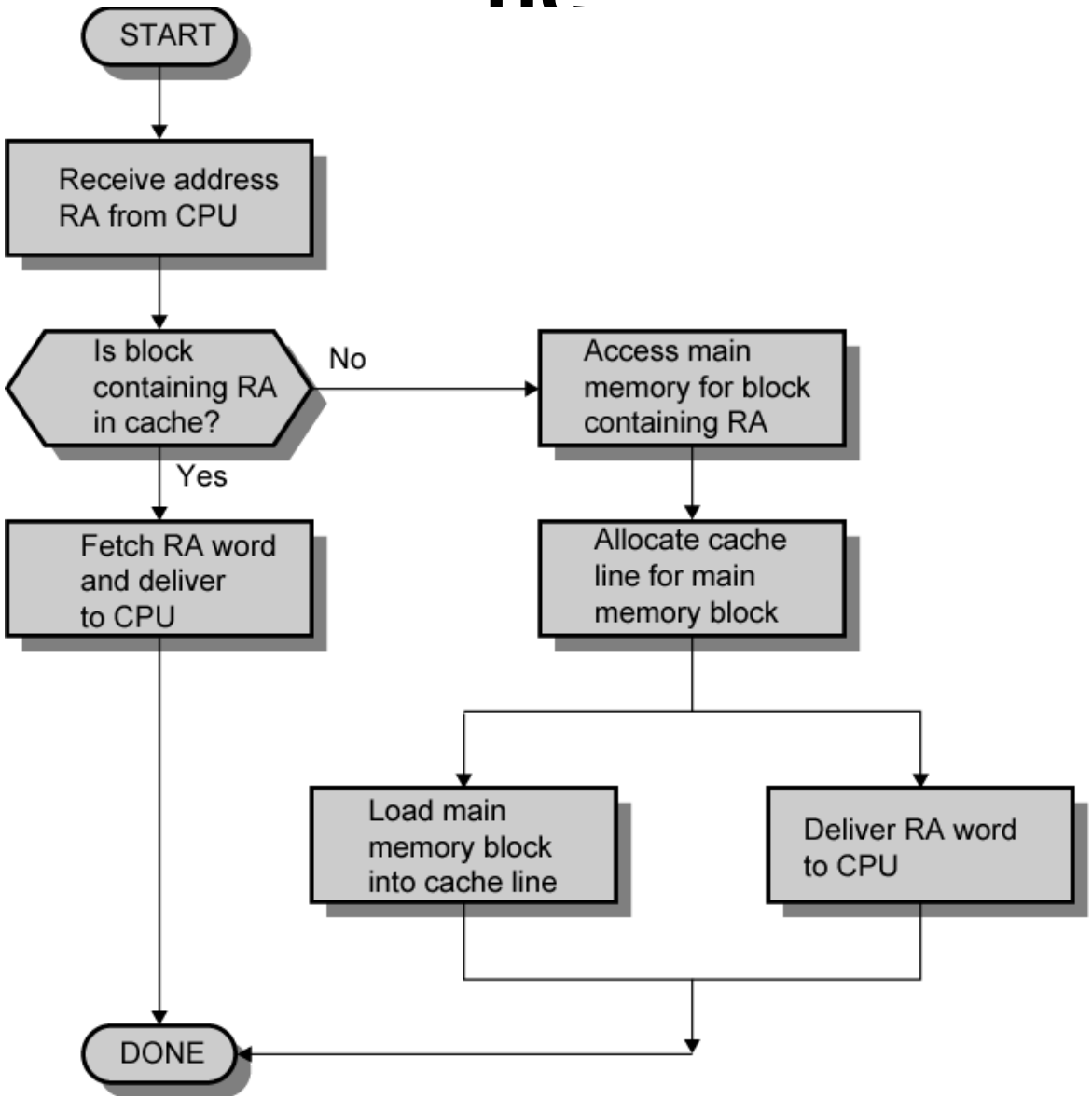
Cac
he/


Line
Number
Tag
Block
0
1
2
C -1
Block Length
(K Words)
(a) Cache
Memory
address
0
1
2
3
Block
(K words)
Block
$2^n - 1$
Word
Length
(b) Main memory

Cache Operation – Overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cac
he


START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Size does matter

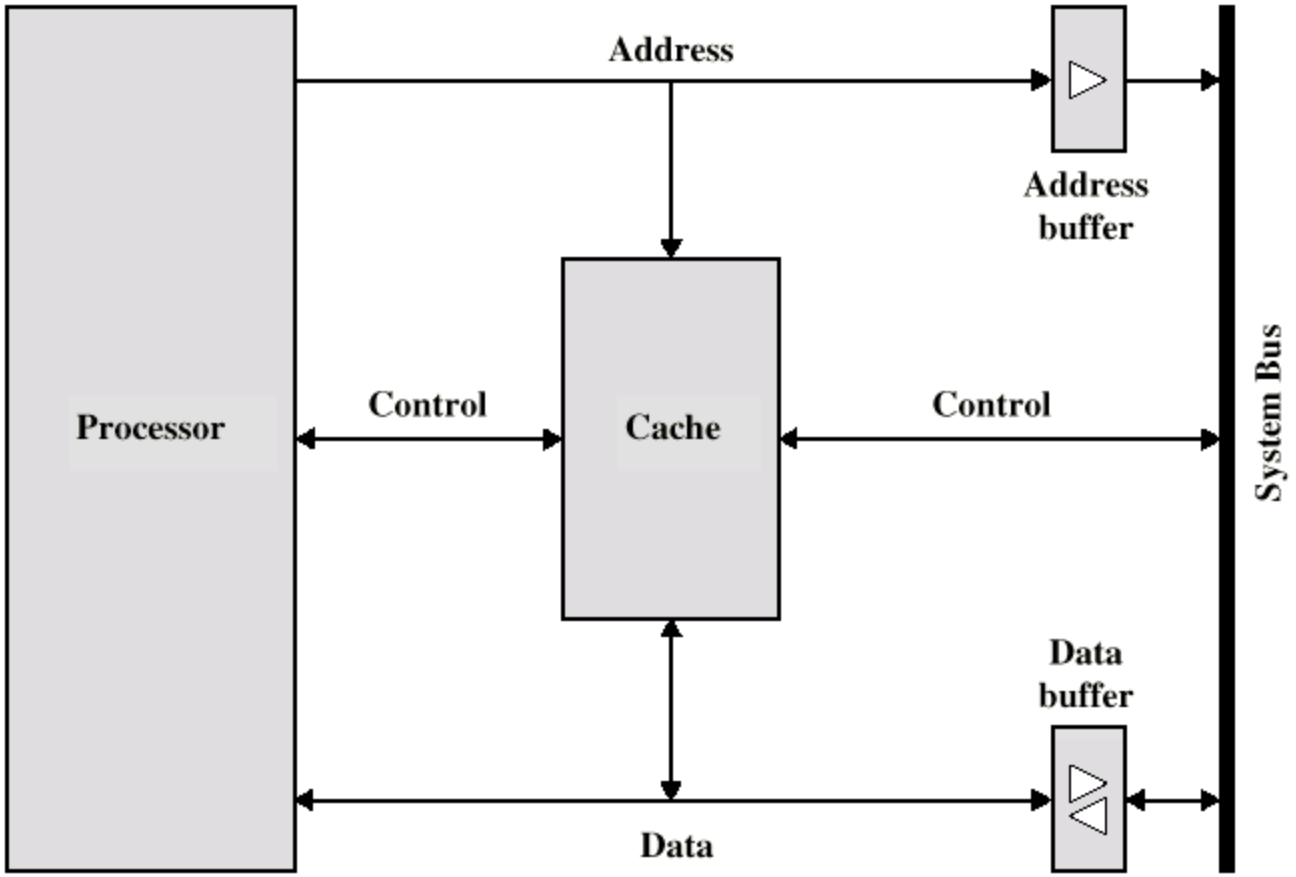
Cost

- **More cache is expensive**

Speed

- **More cache is faster (up to a point)**
- **Checking cache for data takes time**

Typical Cache



Comparison of Cache Sizes

Processor	Type	Year of Introduction	L1 cache[a]	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—

Mapping Function

Cache of 64kByte

Cache block of 4 bytes

- **i.e. cache is 16k (2^{14}) lines of 4 bytes**

16MBytes main memory

24 bit address

- **(2^{24}=16M)**

Direct Mapping

- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
- Least Significant w bits identify unique word
- Most Significant s bits specify one memory block
- The MSBs are split into a cache line field r and a tag of s-r (most significant)

Direct Mapping Address Structure

Tag s-r	Line or Slot r	Word w
8	14	2

24 bit address

2 bit word identifier (4 byte block)

22 bit block identifier

- **8 bit tag (=22-14)**
- **14 bit slot or line**

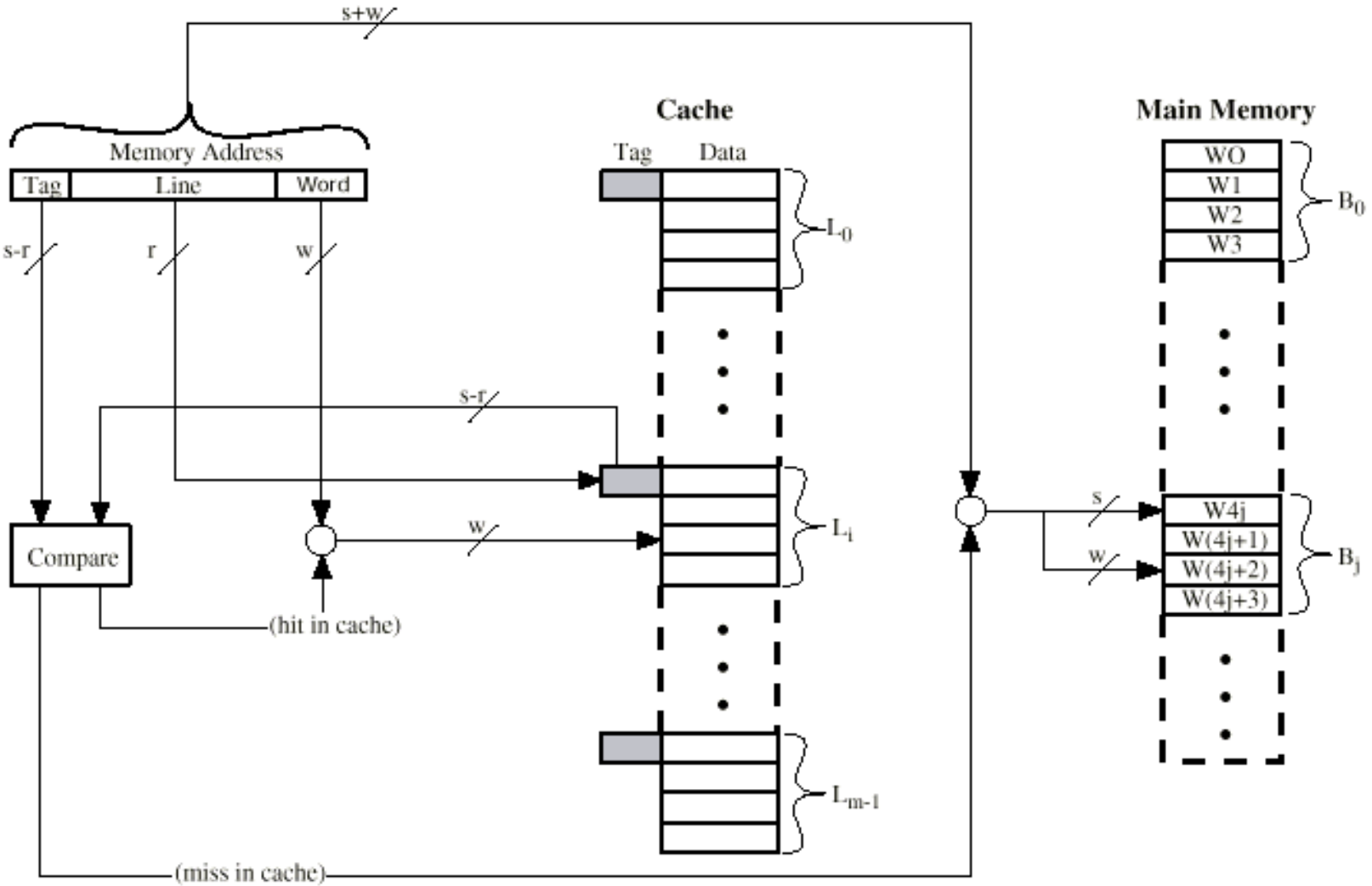
No two blocks in the same line have the same Tag field

Check contents of cache by finding line and checking Tag

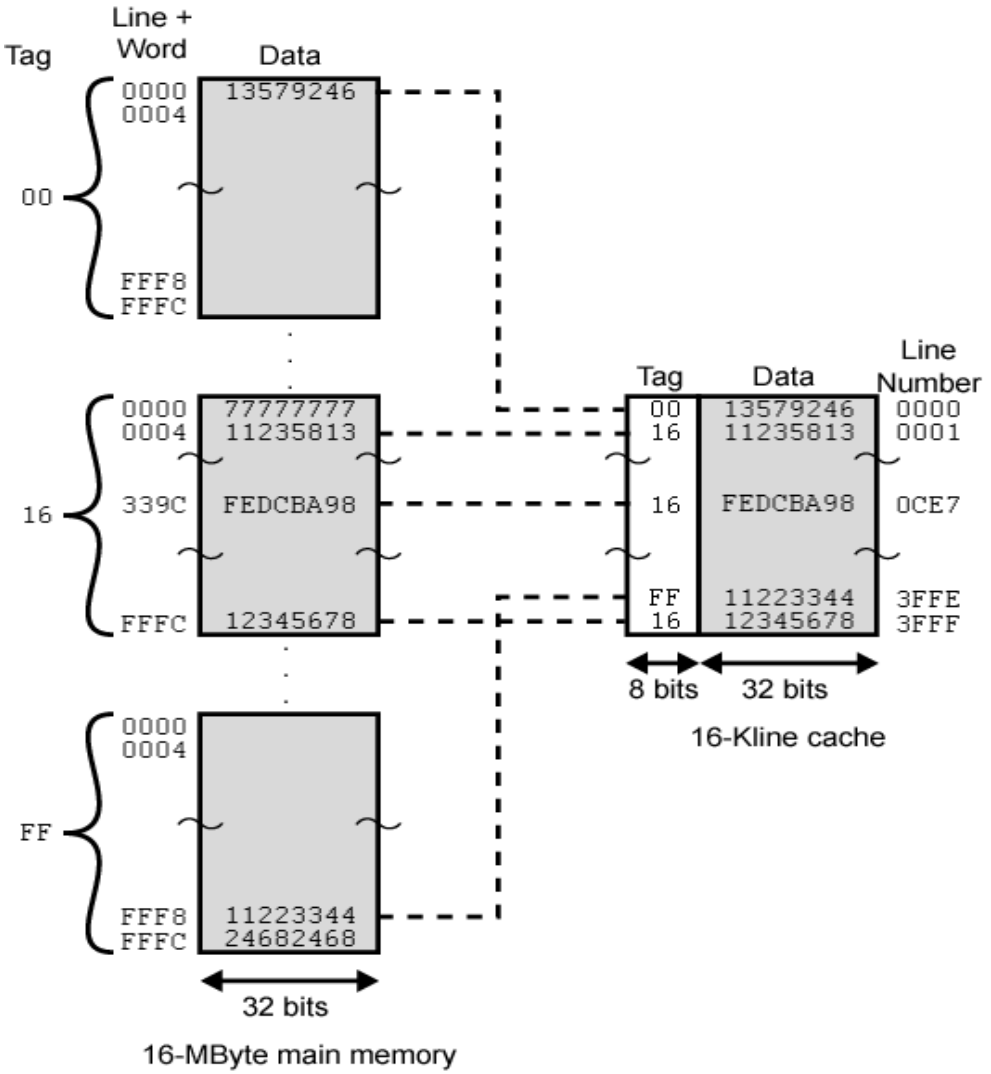
Direct Mapping Cache Line Table

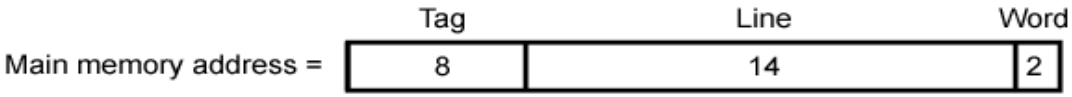
Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2s-m
1	1,m+1, 2m+1…2s-m+1
m-1	m-1, 2m-1, 3m-1…2s-1

Direct Map


s+w
Memory Address
Tag
Line
Word
s-r
r
w
Cache
Tag
Data
L_0
L_i
L_{m-1}
s-r
w
Compare
(hit in cache)
(miss in cache)
Main Memory
WO
W1
W2
W3
B_0
s
w
W4j
W(4j+1)
W(4j+2)
W(4j+3)
B_j

Direct Mapping Example





Direct Mapping Summary

Address length = (s + w) bits

Number of addressable units = 2s+w words or bytes

Block size = line size = 2w words or bytes

Number of blocks in main memory = 2s+ w/2w = 2s

Number of lines in cache = m = 2r

Size of tag = (s – r) bits

Direct Mapping pros & cons

Simple

Inexpensive

Fixed location for given block

- **If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high**

Associative Mapping

A main memory block can load into any line of cache

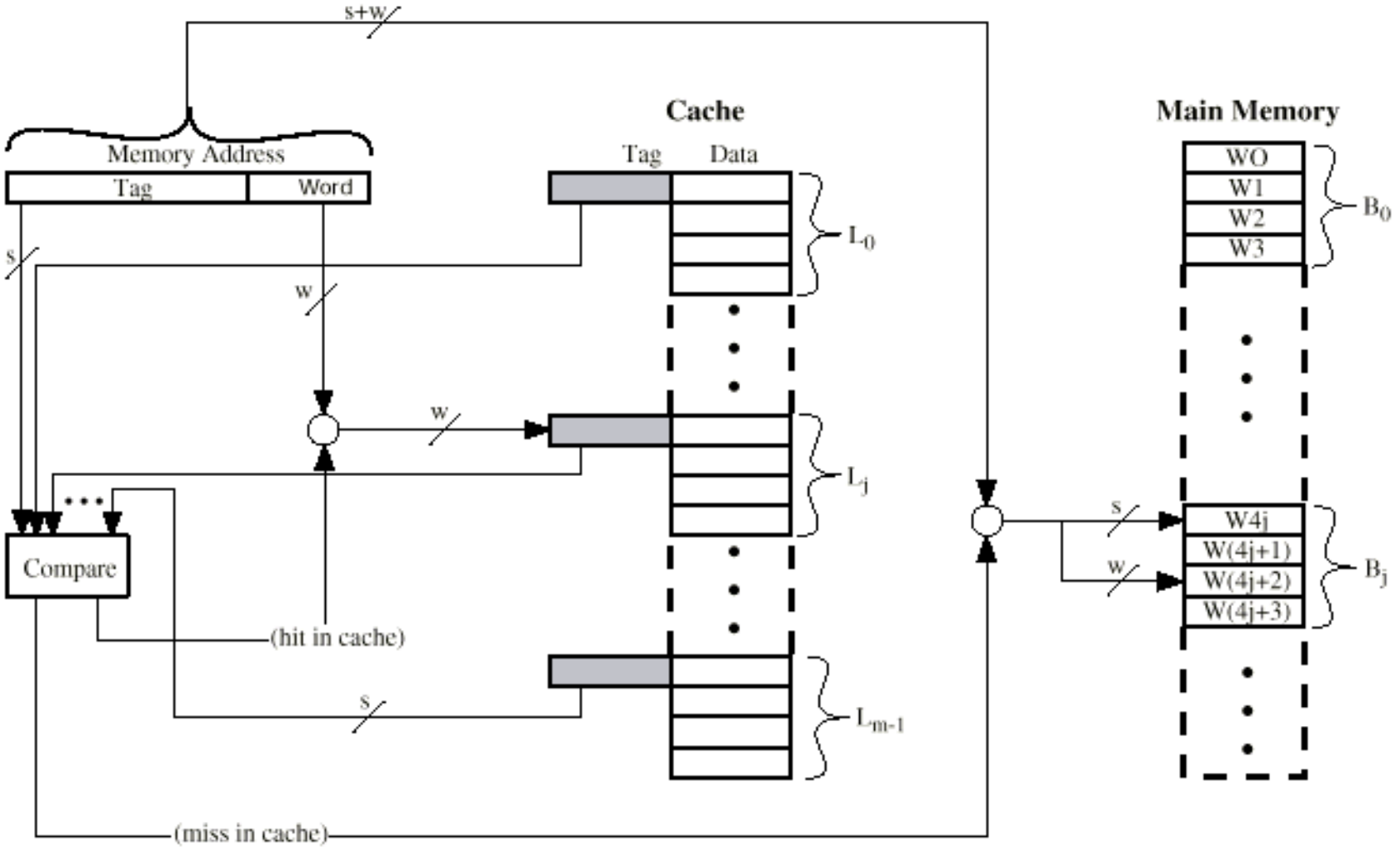
Memory address is interpreted as tag and word

Tag uniquely identifies block of memory

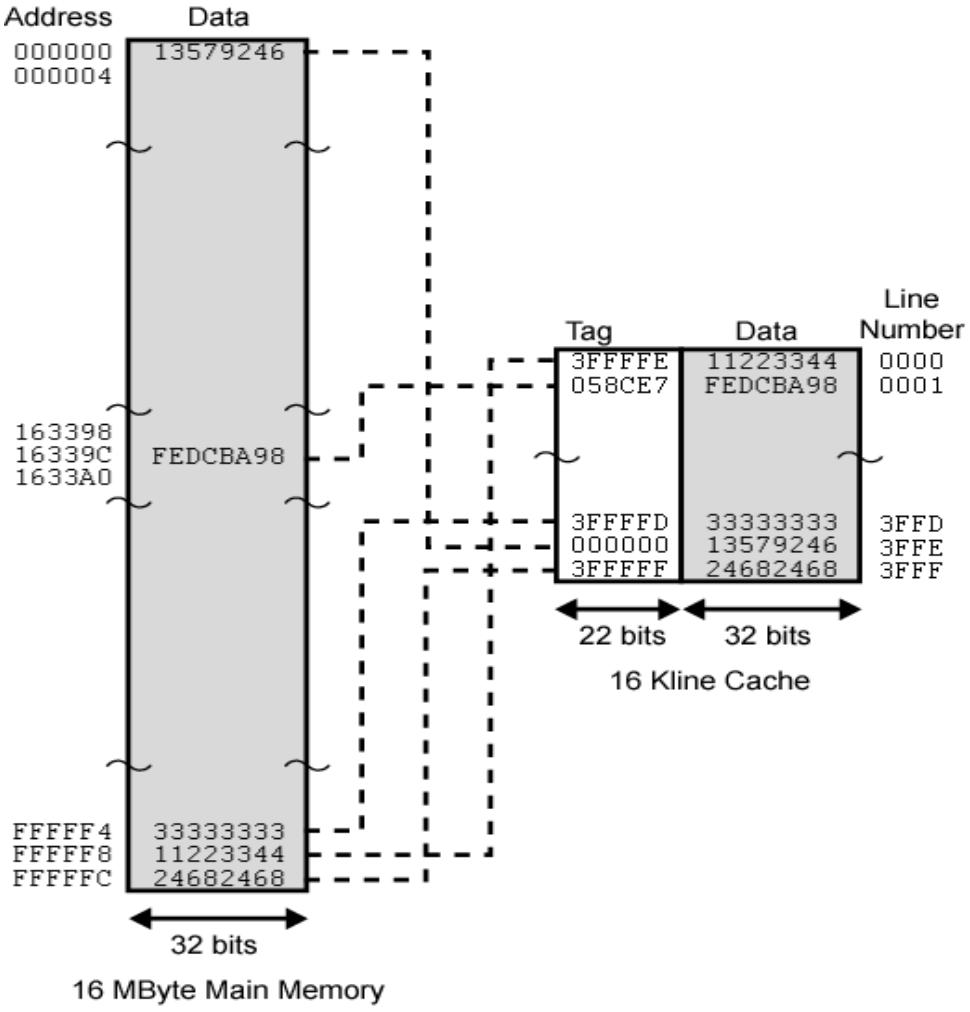
Every line's tag is examined for a match

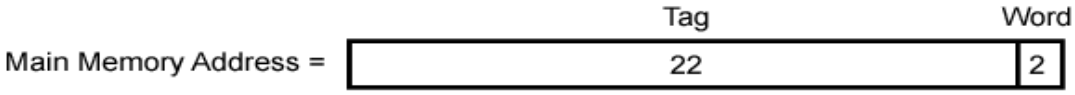
Cache searching gets expensive

Full
y
Ass


s+w
Memory Address
Tag
Word
s
w
Cache
Tag
Data
L_0
w
L_j
Compare
(hit in cache)
s
L_{m-1}
(miss in cache)
Main Memory
WO
W1
W2
W3
B_0
s
w
W4j
W(4j+1)
W(4j+2)
W(4j+3)
B_j

Assosiative Mapping Example





Associative Mapping Address Structure

22 bit tag stored with each 32 bit block of data

Compare tag field with tag entry in cache to check for hit

Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block

e.g.

	Address	Tag	Data	Cache line
•	FFFFFC	FFFFFC	24682468	3FFF

Associative Mapping Summary

Address length = (s + w) bits

Number of addressable units = 2s+w words or bytes

Block size = line size = 2w words or bytes

Number of blocks in main memory = 2s+ w/2w = 2s

Number of lines in cache = undetermined

Size of tag = s bits

Set Associative Mapping

Cache is divided into a number of sets

Each set contains a number of lines

A given block maps to any line in a given set

- e.g. Block B can be in any line of set i

e.g. 2 lines per set

- 2 way associative mapping
- A given block can be in one of 2 lines in only one set

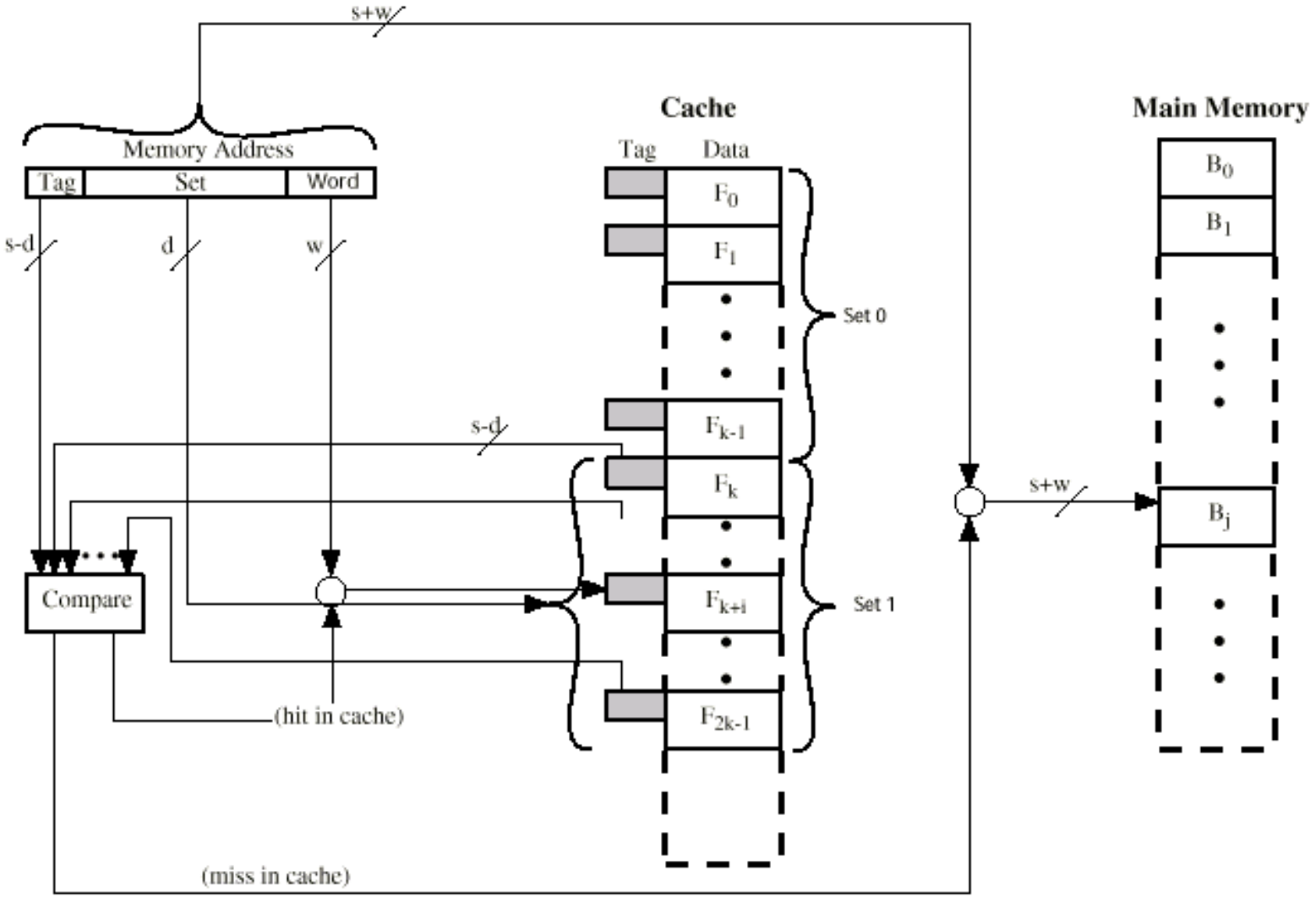
Set Associative Mapping Example

13 bit set number

Block number in main memory is modulo 2^{13}

000000, 00A000, 00B000, 00C000 … map to same set

Two Way Set Associative Cache Organization



Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

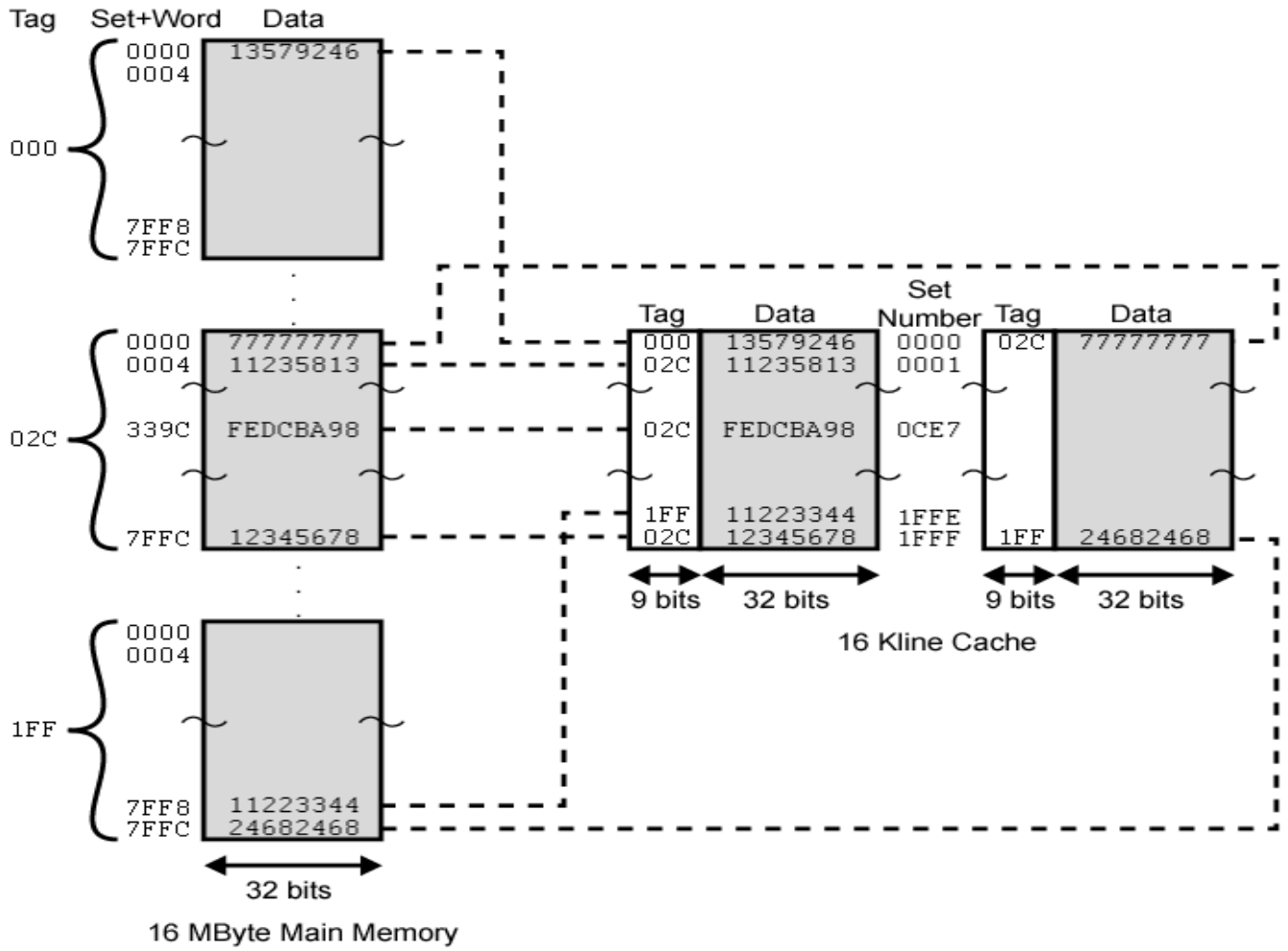
Use set field to determine cache set to look in

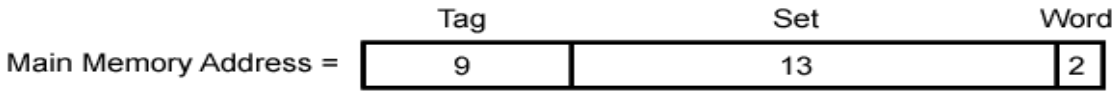
Compare tag field to see if we have a hit

e.g

Address	Tag	Data	Set number
1FF 7FFC	1FF	12345678	1FFF
001 7FFC	001	11223344	1FFF

Two Way set Assosiative Mapping Example





Set Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2s+w words or bytes
- Block size = line size = 2w words or bytes
- Number of blocks in main memory = 2d
- Number of lines in set = k
- Number of sets = v = 2d
- Number of lines in cache = kv = k * 2d
- Size of tag = (s – d) bits

Replacement Algorithms (1) Direct mapping

No choice

Each block only maps to one line

Replace that line

Replacement Algorithms (2)
Associative & Set Associative

Hardware implemented algorithm (speed)

Least Recently used (LRU)

e.g. in 2 way set associative

- **Which of the 2 block is lru?**

First in first out (FIFO)

- **replace block that has been in cache longest**

Least frequently used

- **replace block which has had fewest hits**

Random

Write Policy

- Must not overwrite a cache block unless main memory is up to date
- Multiple CPUs may have individual caches
- I/O may address main memory directly

Write through

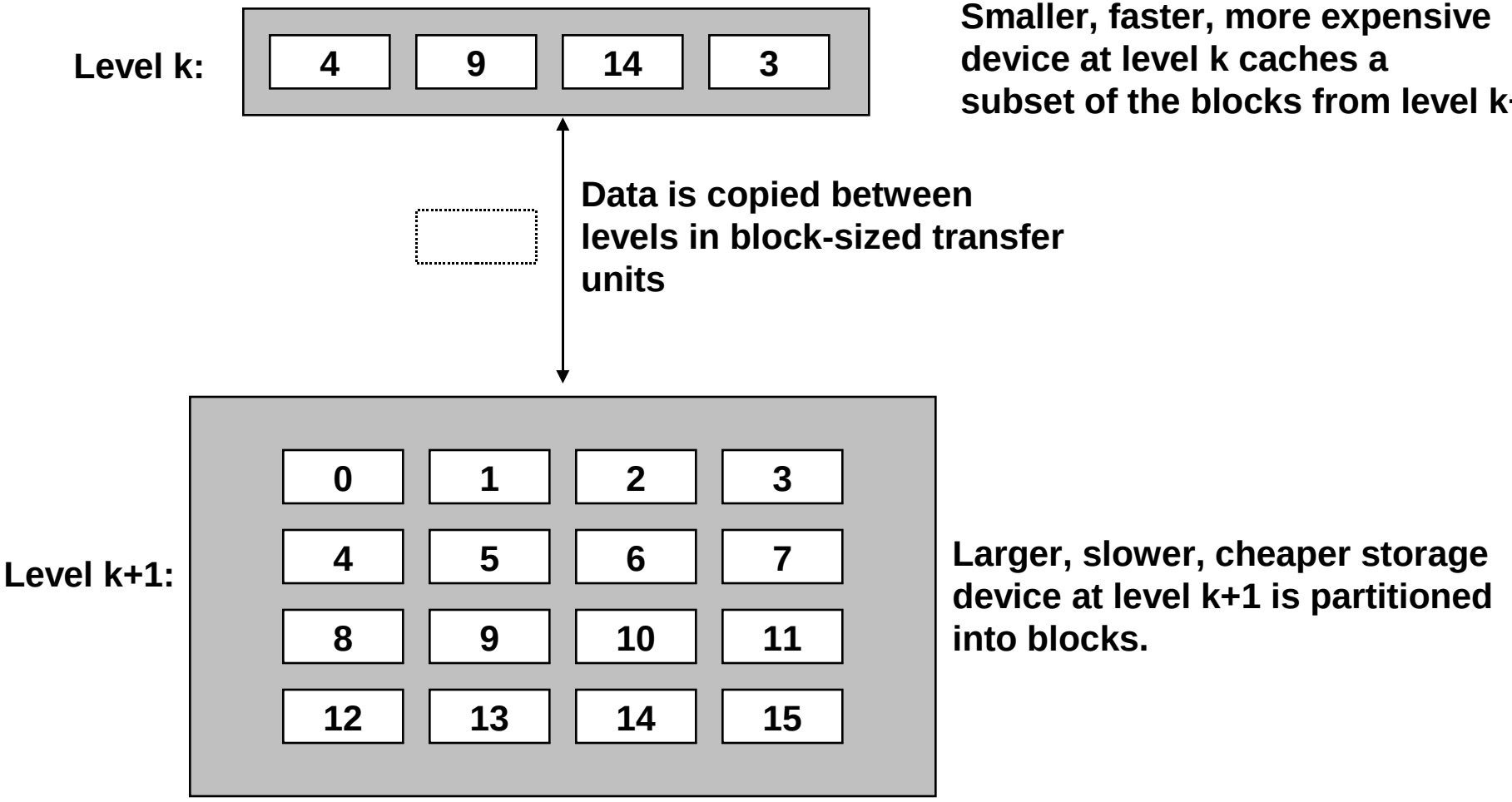
- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes

Remember bogus write through caches!

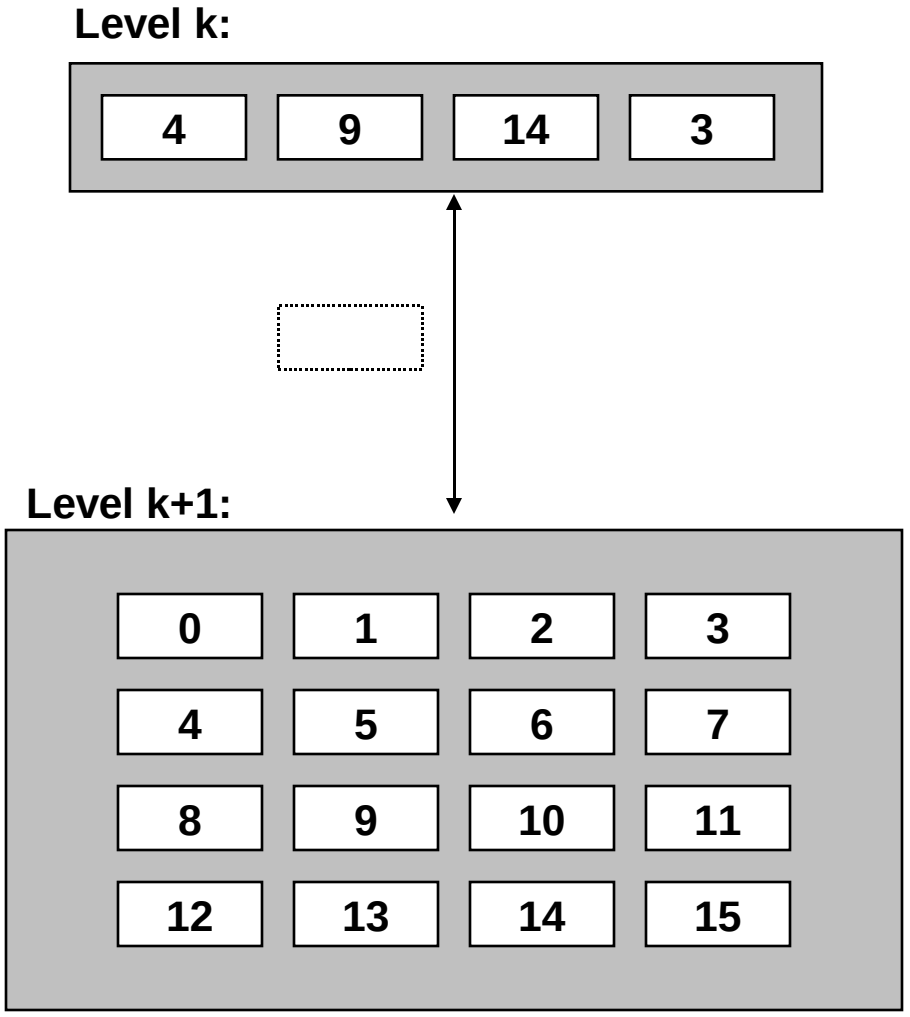
Write back

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- N.B. 15% of memory references are writes

Caching in a memory hierarchy



General caching concepts



Program needs object d, which is stored in some block b.

Cache hit

- **Program finds b in the cache at level k. E.g. block 14.**

Cache miss

- **b is not at level k, so level k cache must fetch it from level k+1. E.g. block 12.**
- **If level k cache is full, then some current block must be *replaced* (*evicted*).**
- **Which one? Determined by replacement policy. E.g. evict least recently used block.**

General caching concepts

Types of cache misses:

- **Cold (compulsary) miss**
  - Cold misses occur because the cache is empty.
- **Conflict miss**
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
  - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k+1.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
  - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

Examples of caching in the hierarchy

Cache Type	What Cached	Where Cached	Latency (cycles)	Managed By
Registers	4-byte word	CPU registers	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	32-byte block	On-Chip L1	1	Hardware
L2 cache	32-byte block	Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware+ OS
Buffer cache	Parts of files	Main memory	100	OS
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Pentium 4 Cache

80386 – no on chip cache

80486 – 8k using 16 byte lines and four way set associative organization

Pentium (all versions) – two on chip L1 caches

- **Data & instructions**

Pentium III – L3 cache added off chip

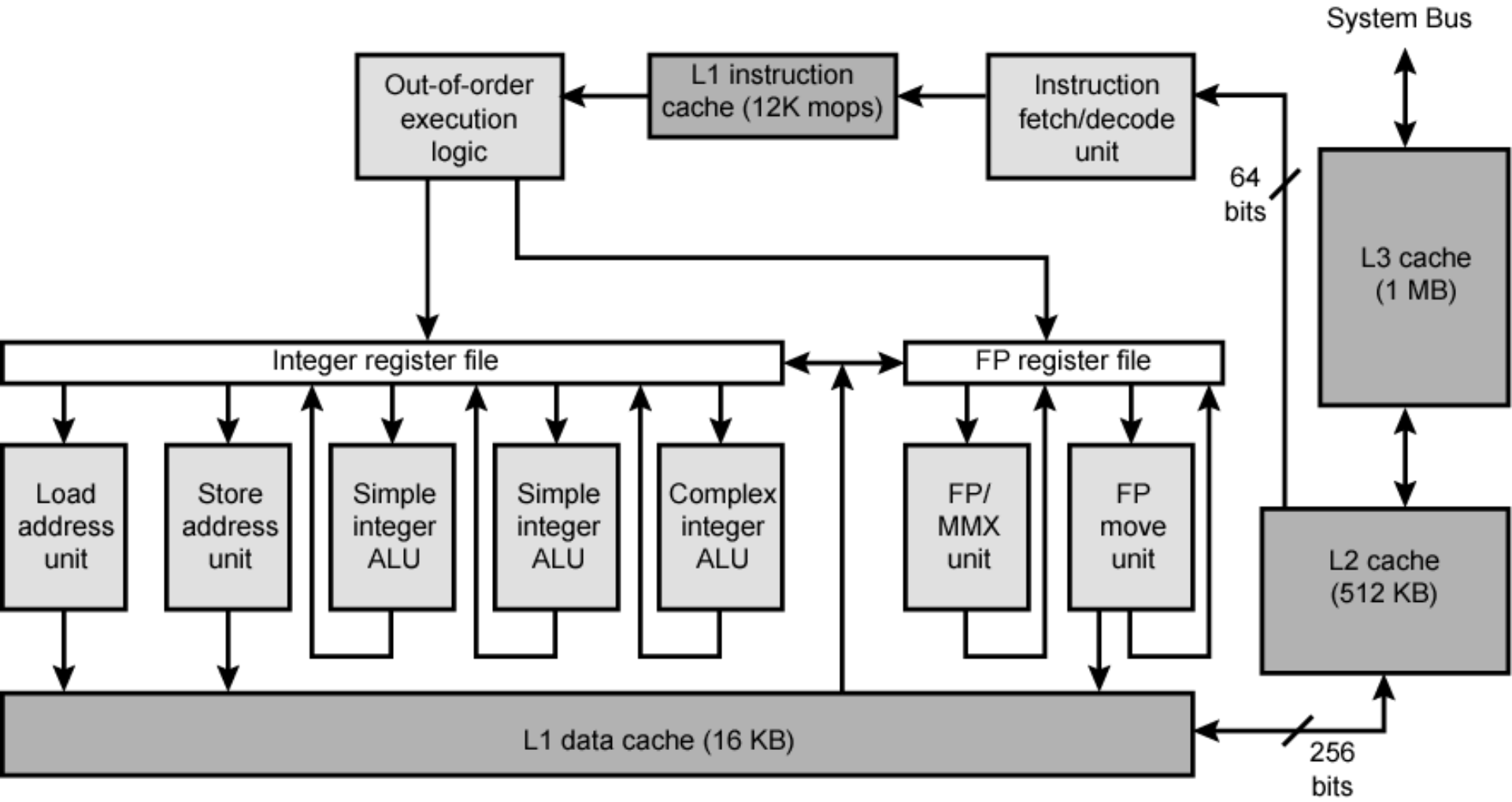
Pentium 4

- **L1 caches**
  - 8k bytes
  - 64 byte lines
  - four way set associative
- **L2 cache**
  - Feeding both L1 caches
  - 256k
  - 128 byte lines
  - 8 way set associative
- **L3 cache on chip**

Intel Cache Evolution

Problem	Solution	Processor on which feature first appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip	Add external L2 cache using faster technology than main memory	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Pentiu


System Bus
Out-of-order execution logic
L1 instruction cache (12K mops)
Instruction fetch/decode unit
64 bits
L3 cache (1 MB)
Integer register file
FP register file
Load address unit
Store address unit
Simple integer ALU
Simple integer ALU
Complex integer ALU
FP/ MMX unit
FP move unit
L2 cache (512 KB)
L1 data cache (16 KB)
256 bits

Pentium 4 Core Processor

Fetch/Decode Unit

- Fetches instructions from L2 cache
- Decode into micro-ops
- Store micro-ops in L1 cache

Out of order execution logic

- Schedules micro-ops
- Based on data dependence and resources
- May speculatively execute

Execution units

- Execute micro-ops
- Data from L1 cache
- Results in registers

Memory subsystem

- L2 cache and systems bus

Pentium 4 Design Reasoning

- Decodes instructions into RISC like micro-ops before L1 cache
- Micro-ops fixed length
  - Superscalar pipelining and scheduling
- Pentium instructions long & complex
- Performance improved by separating decoding from scheduling & pipelining
  - (More later – ch14)
- Data cache is write back
  - Can be configured to write through
- L1 cache controlled by 2 bits in register
  - CD = cache disable
  - NW = not write through
  - 2 instructions to invalidate (flush) cache and write back then invalidate
- L2 and L3 8-way set-associative
  - Line size 128 bytes

PowerPC Cache Organization

601 – single 32kb 8 way set associative

603 – 16kb (2 x 8kb) two way set associative

604 – 32kb

620 – 64kb

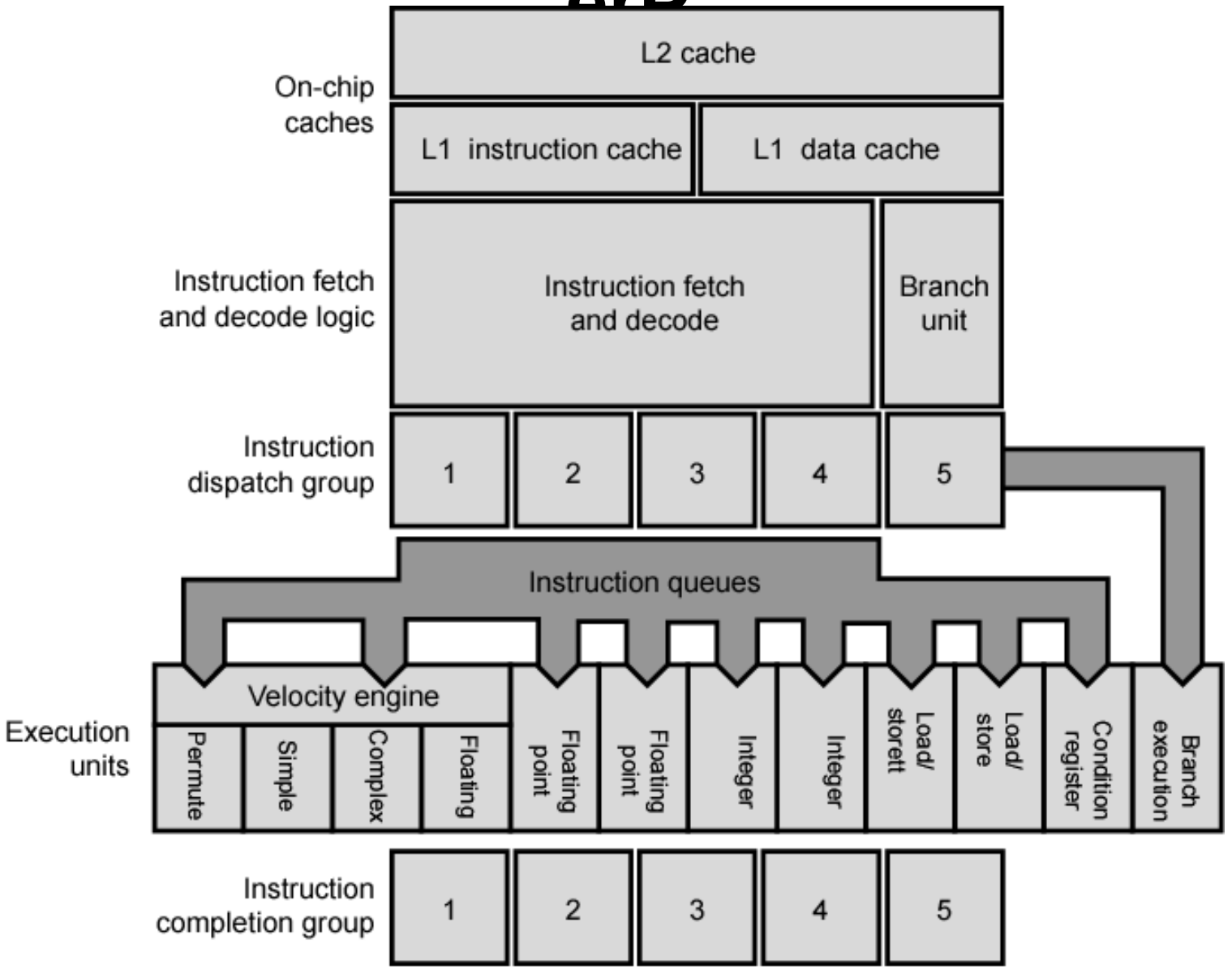
G3 & G4

- **64kb L1 cache**
  - 8 way set associative
- **256k, 512k or 1M L2 cache**
  - two way set associative

G5

- **32kB instruction cache**
- **64kB data cache**

PowerPC


On-chip caches
L2 cache
L1 instruction cache
L1 data cache
Instruction fetch and decode logic
Instruction fetch and decode
Branch unit
Instruction dispatch group
1
2
3
4
5
Instruction queues
Execution units
Velocity engine
Permute
Simple
Complex
Floating
Floating point
Floating point
Integer
Integer
Load/ storett
Load/ store
Condition register
Branch execution
Instruction completion group
1
2
3
4
5