McGill

Computer Organization

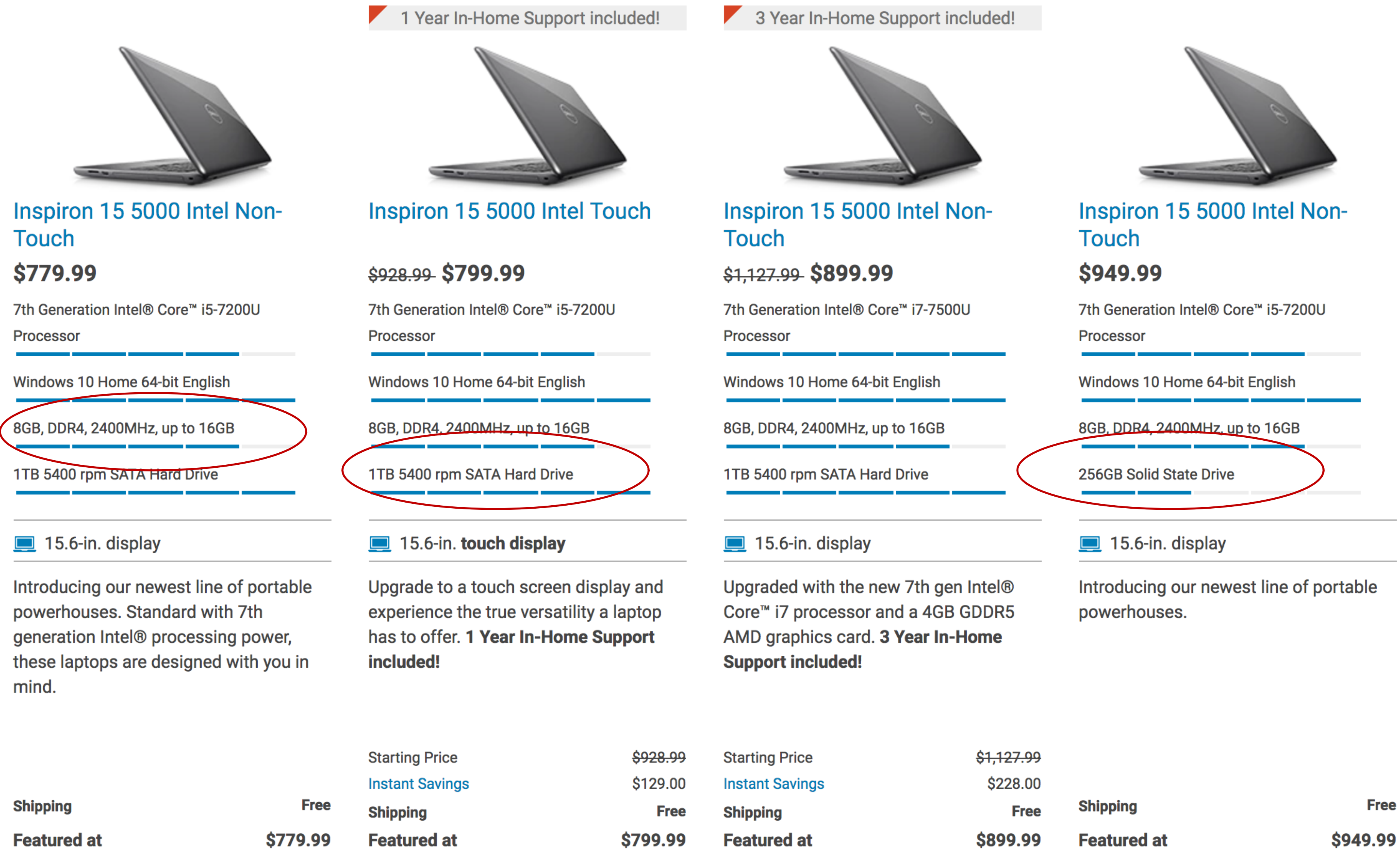
Memory

ECSE 324

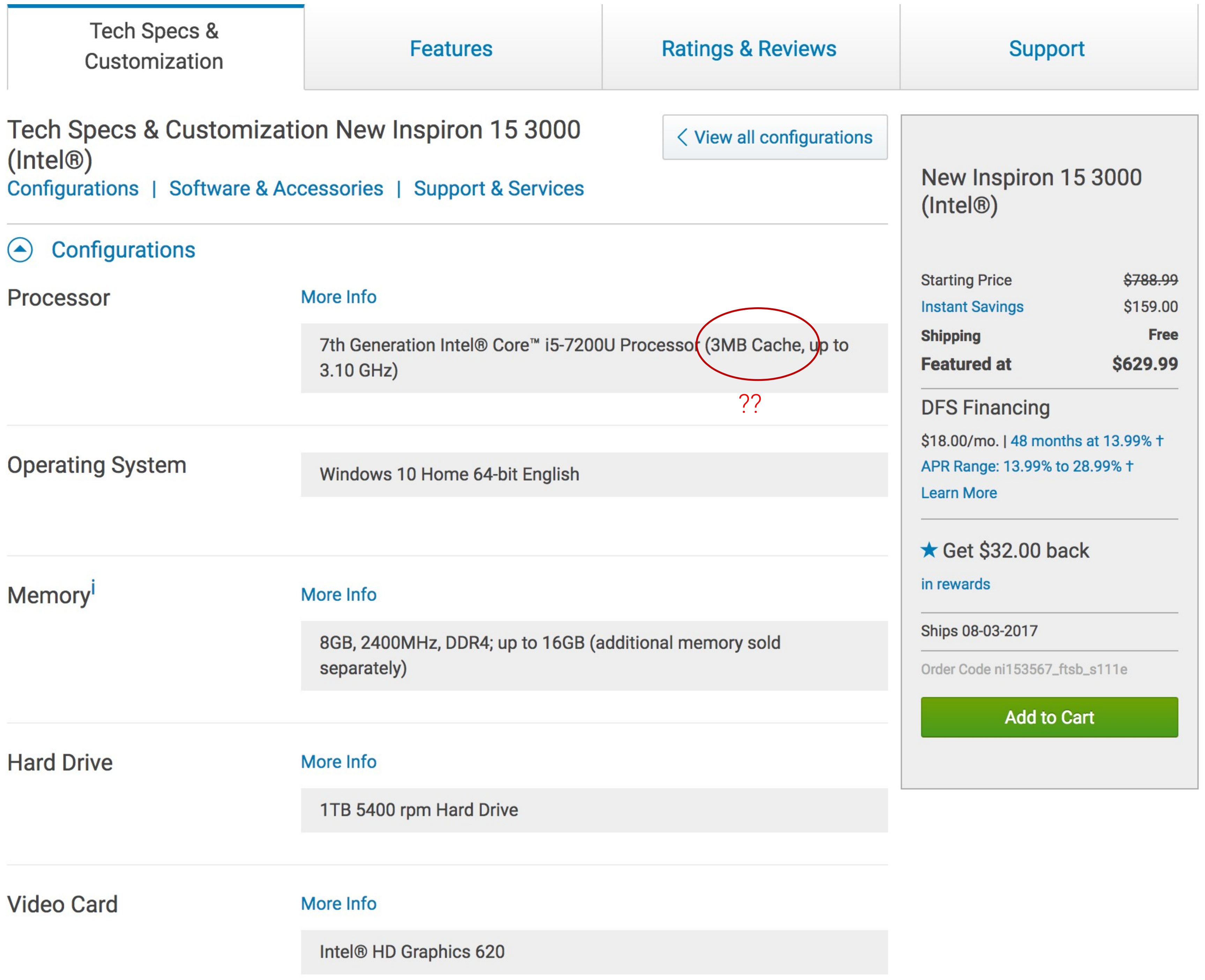
Fall 2020

Prof. Christophe Dubach

Original slides from Prof. Warren Gross. Some material from Hamacher, Vranesic, Zaky, and Manjikian, "Computer Organization and Embedded Systems, 6th ed", 2012, McGraw Hill and "Introduction to the ARM Processor using Altera Toolchain".



1	2	3	4
	1 Year In-Home Support included!	3 Year In-Home Support included!	
Inspiron 15 5000 Intel Non-Touch	Inspiron 15 5000 Intel Touch	Inspiron 15 5000 Intel Non-Touch	Inspiron 15 5000 Intel Non-Touch
$779.99	~~$928.99~~ **$799.99**	~~$1,127.99~~ **$899.99**	**$949.99**
7th Generation Intel® Core™ i5-7200U Processor	7th Generation Intel® Core™ i5-7200U Processor	7th Generation Intel® Core™ i7-7500U Processor	7th Generation Intel® Core™ i5-7200U Processor
Windows 10 Home 64-bit English	Windows 10 Home 64-bit English	Windows 10 Home 64-bit English	Windows 10 Home 64-bit English
8GB, DDR4, 2400MHz, up to 16GB	8GB, DDR4, 2400MHz, up to 16GB	8GB, DDR4, 2400MHz, up to 16GB	8GB, DDR4, 2400MHz, up to 16GB
1TB 5400 rpm SATA Hard Drive	1TB 5400 rpm SATA Hard Drive	1TB 5400 rpm SATA Hard Drive	256GB Solid State Drive
15.6-in. display	15.6-in. **touch display**	15.6-in. display	15.6-in. display
Introducing our newest line of portable powerhouses. Standard with 7th generation Intel® processing power, these laptops are designed with you in mind.	Upgrade to a touch screen display and experience the true versatility a laptop has to offer. **1 Year In-Home Support included!**	Upgraded with the new 7th gen Intel® Core™ i7 processor and a 4GB GDDR5 AMD graphics card. **3 Year In-Home Support included!**	Introducing our newest line of portable powerhouses.
	Starting Price ~~$928.99~~	Starting Price ~~$1,127.99~~	
	Instant Savings $129.00	Instant Savings $228.00	
Shipping **Free**	**Shipping** **Free**	**Shipping** **Free**	**Shipping** **Free**
Featured at **$779.99**	**Featured at** **$799.99**	**Featured at** **$899.99**	**Featured at** **$949.99**



Tech Specs & Customization | Features | Ratings & Reviews | Support

Tech Specs & Customization New Inspiron 15 3000 (Intel®)

Configurations | Software & Accessories | Support & Services

‹ View all configurations

Configurations

Processor	More Info 7th Generation Intel® Core™ i5-7200U Processor (3MB Cache, up to 3.10 GHz)
Operating System	Windows 10 Home 64-bit English
Memory[i]	More Info 8GB, 2400MHz, DDR4; up to 16GB (additional memory sold separately)
Hard Drive	More Info 1TB 5400 rpm Hard Drive
Video Card	More Info Intel® HD Graphics 620

??

New Inspiron 15 3000 (Intel®)

Starting Price	~~$788.99~~
Instant Savings	$159.00
Shipping	**Free**
Featured at	**$629.99**

DFS Financing

$18.00/mo. | 48 months at 13.99% †

APR Range: 13.99% to 28.99% †

Learn More

★ Get $32.00 back

in rewards

Ships 08-03-2017

Order Code ni153567_ftsb_s111e

Models

Wi-Fi

Wi-Fi + Cellular

Capacity[1]

Wi-Fi	Wi-Fi + Cellular
32GB	32GB
128GB	128GB

capacity?

Size and Weight[2]

Wi-Fi	Wi-Fi + Cellular
Height: 240 mm (9.4 inches)	**Height:** 240 mm (9.4 inches)
Width: 169.5 mm (6.6 inches)	**Width:** 169.5 mm (6.6 inches)

What is the difference between memory and storage?

How do they interact?

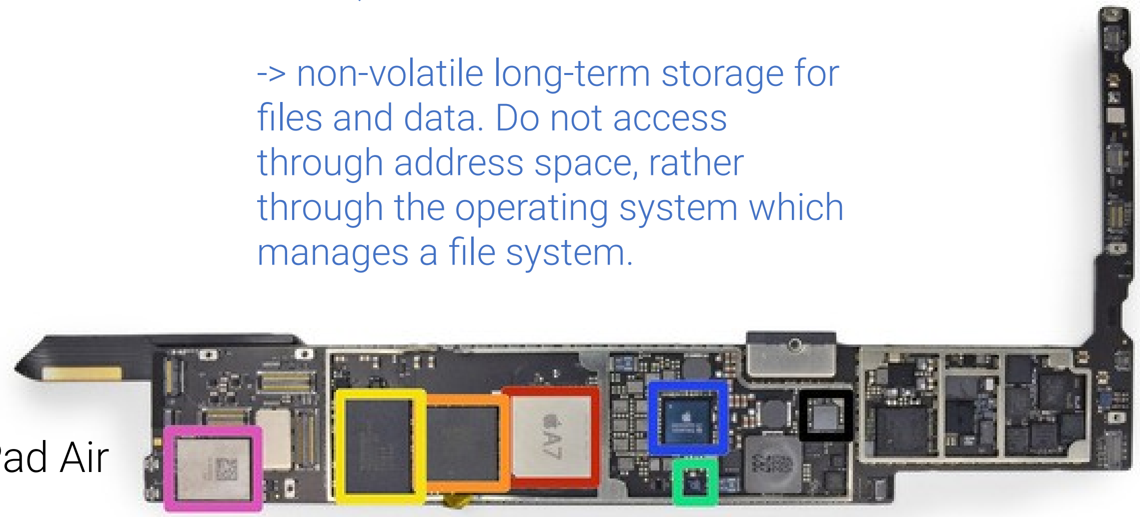
What are the different technologies used to implement memory?

"Memory", "Main Memory", "RAM",...
Volatile working memory (loses its contents when the machine is off) – addressable

"Storage", "Capacity (marketing term!)", "Hard disk", "Hard drive", "drive", "solid-state drive (SSD)", "flash",...

-> non-volatile long-term storage for files and data. Do not access through address space, rather through the operating system which manages a file system.

iPad Air



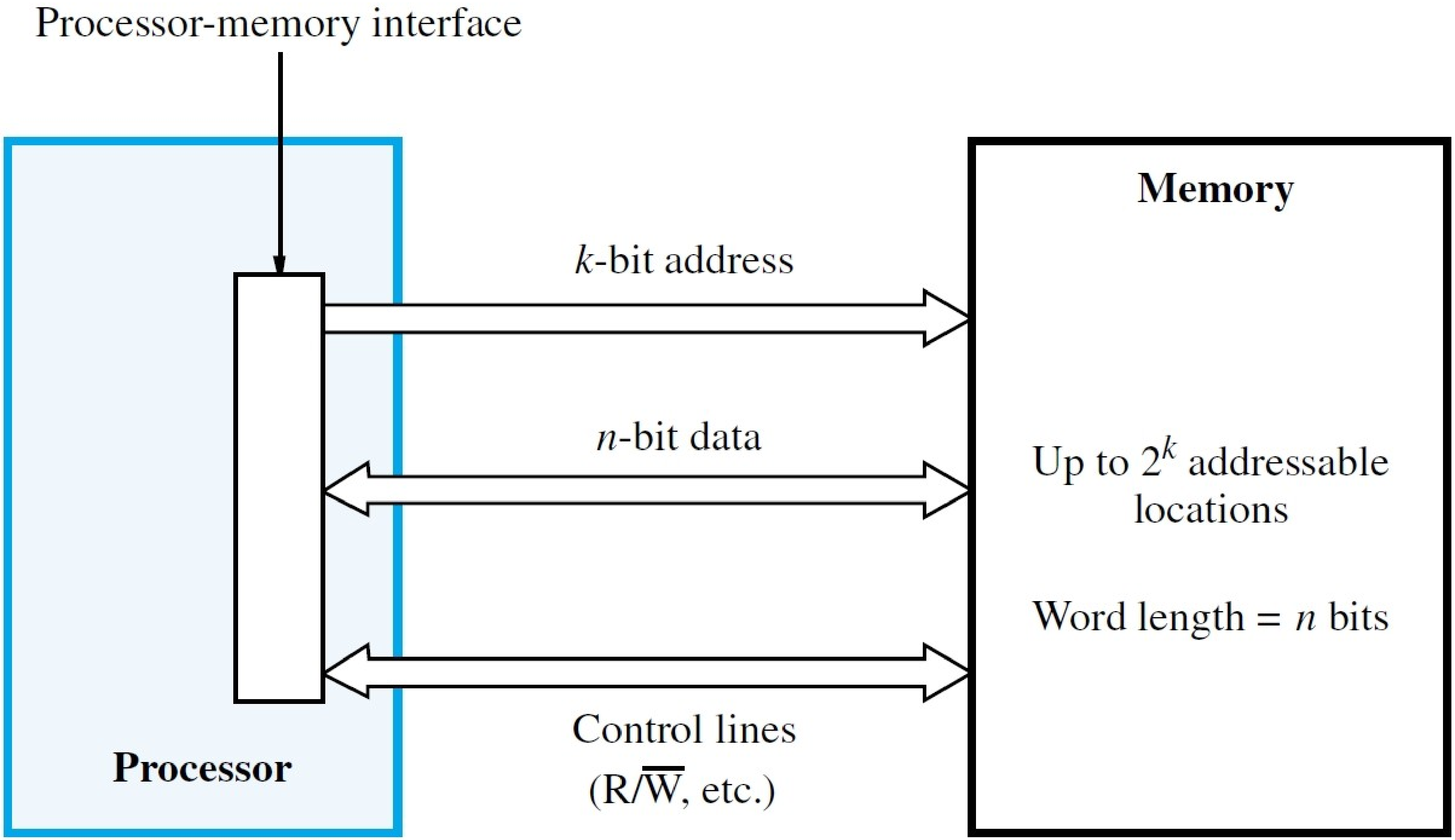
Elpida 1 GB LPDDR3 SDRAM - "memory"

Toshiba 16 GB NAND Flash - "storage"

Memory Technology

Textbook 8.1, 8.2, 8.3

Abstraction



RAM

- *Memory access time*

time from initiation to completion of a word or byte transfer

- *Memory cycle time*

minimum time delay between initiation of successive transfers

Random-access memory (RAM)

means that access time is same, independent of location

Semiconductor RAM

- Organized as an array of *cells*, each storing one bit
- Each row of the array stores one *memory word*

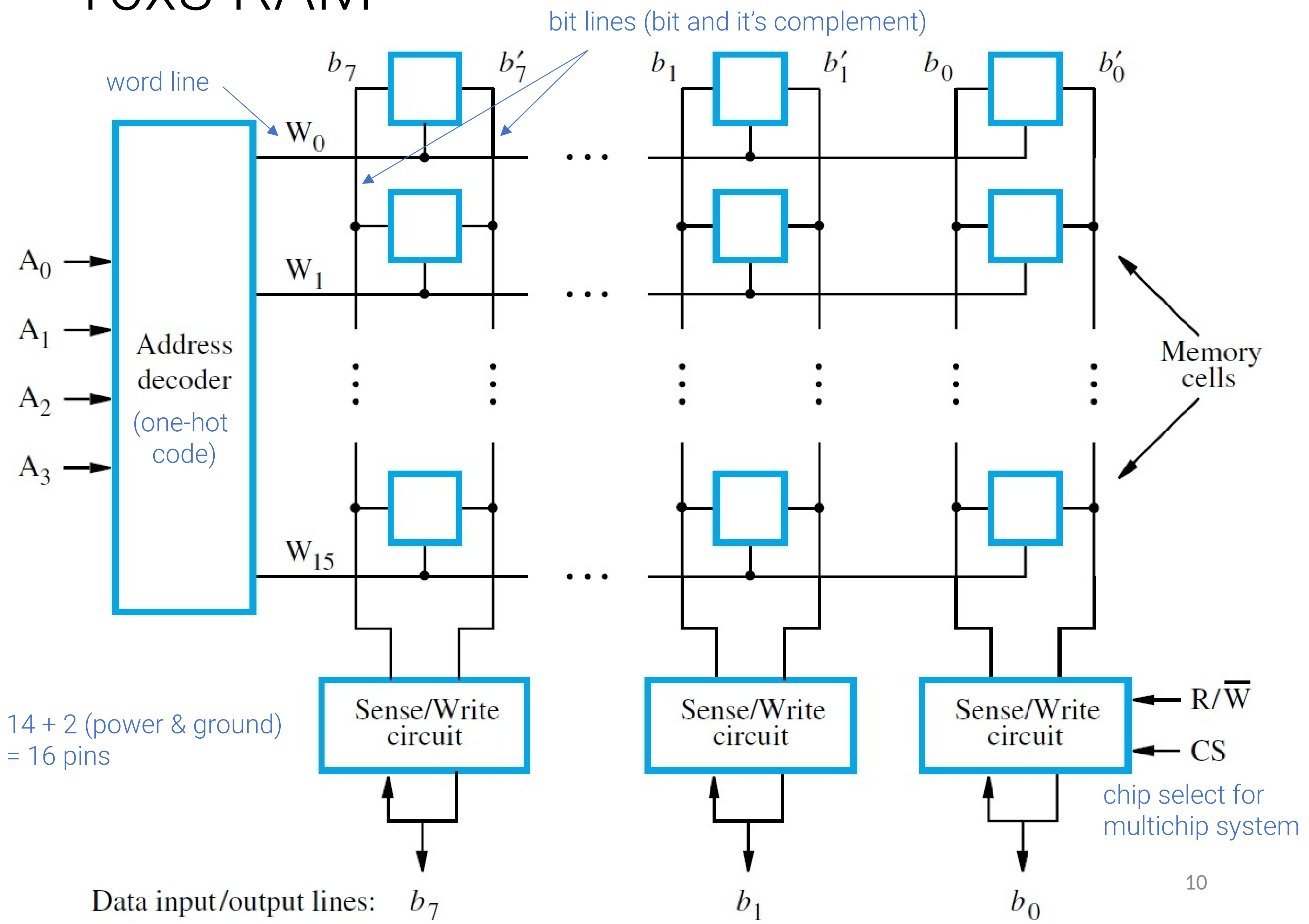
  !!! Memory word might be different from processor word !!!

E.g. consider a 16x8 RAM with a 8-bit wordsize and 16 words.

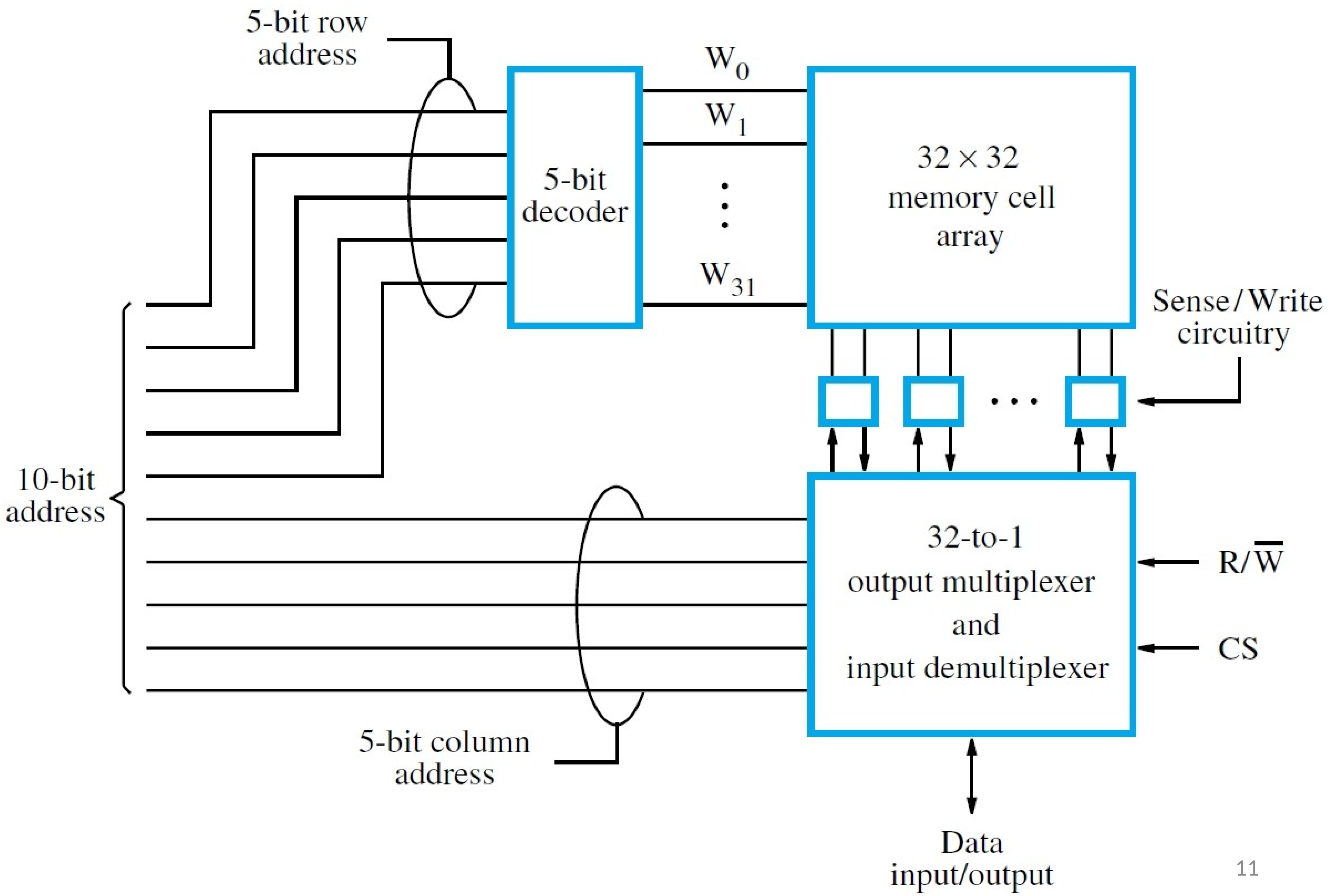
- How many bits does this memory store ?

- How many bits are needed for the memory address?

16x8 RAM

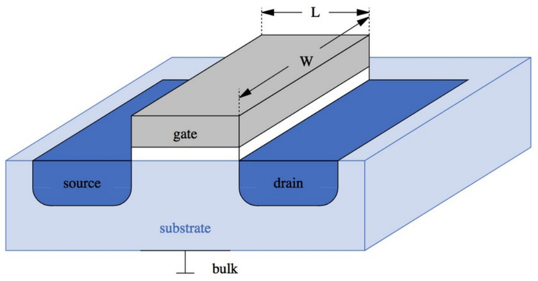

bit lines (bit and it's complement)
word line
b_7
b_7'
b_1
b_1'
b_0
b_0'
W_0
W_1
W_{15}
A_0
A_1
A_2
A_3
Address decoder
(one-hot code)
Memory cells
Sense/Write circuit
Sense/Write circuit
Sense/Write circuit
$R/\overline{W}$
CS
chip select for multichip system
14 + 2 (power & ground) = 16 pins
Data input/output lines: b_7 b_1 b_0

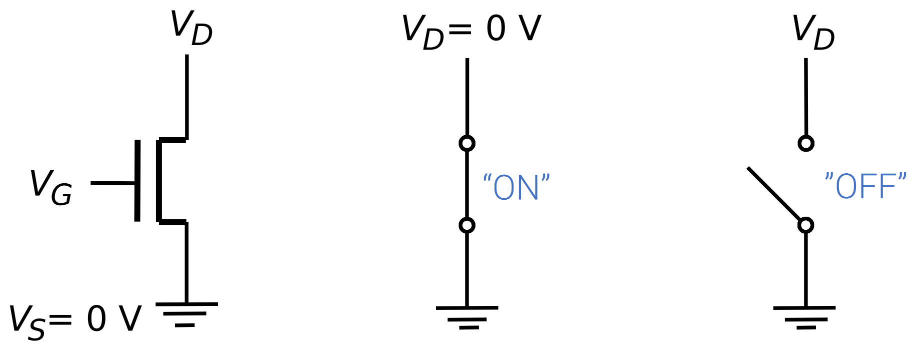
1024x1 RAM


5-bit row address
5-bit decoder
W_0
W_1
W_{31}
32×32 memory cell array
Sense/Write circuitry
10-bit address
32-to-1 output multiplexer and input demultiplexer
$R/\overline{W}$
CS
5-bit column address
Data input/output

SRAM

- *Static* RAM: retains contents as long as power is applied, but *volatile* – if power is removed the contents are destroyed.
- Fast (access time of a few ns), but expensive – each cell require 6 transistors to store a bit.
- As a result, SRAMs are limited to how large they can be – typically at most a few Mbits.
- Use to implement "cache" memory, but not the main memory

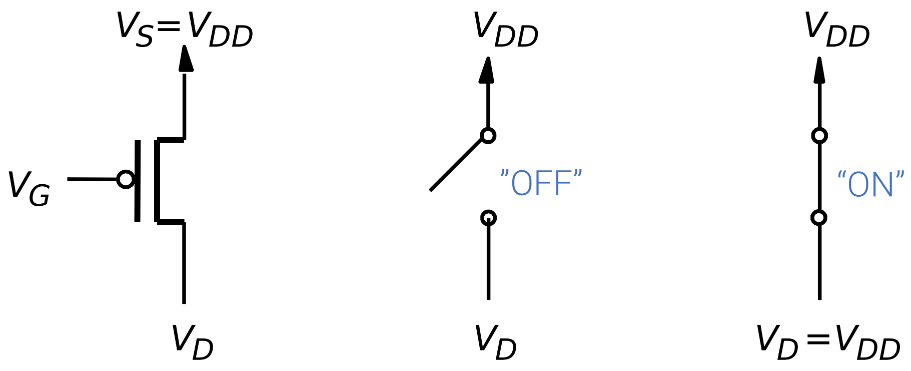




Closed switch when $V_G = V_D$

Open switch when $V_G = 0$ V

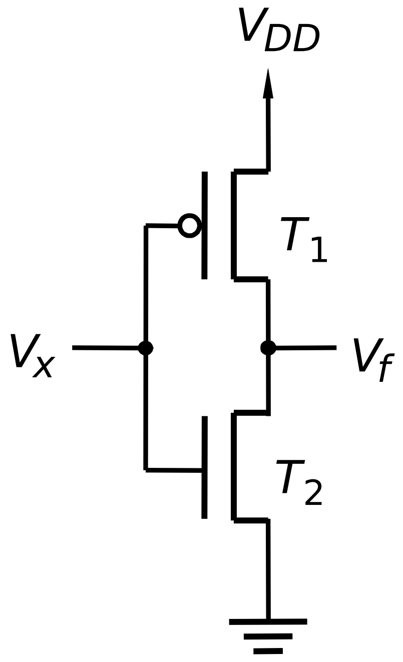
NMOS transistor



Open switch when $V_G = V_{DD}$

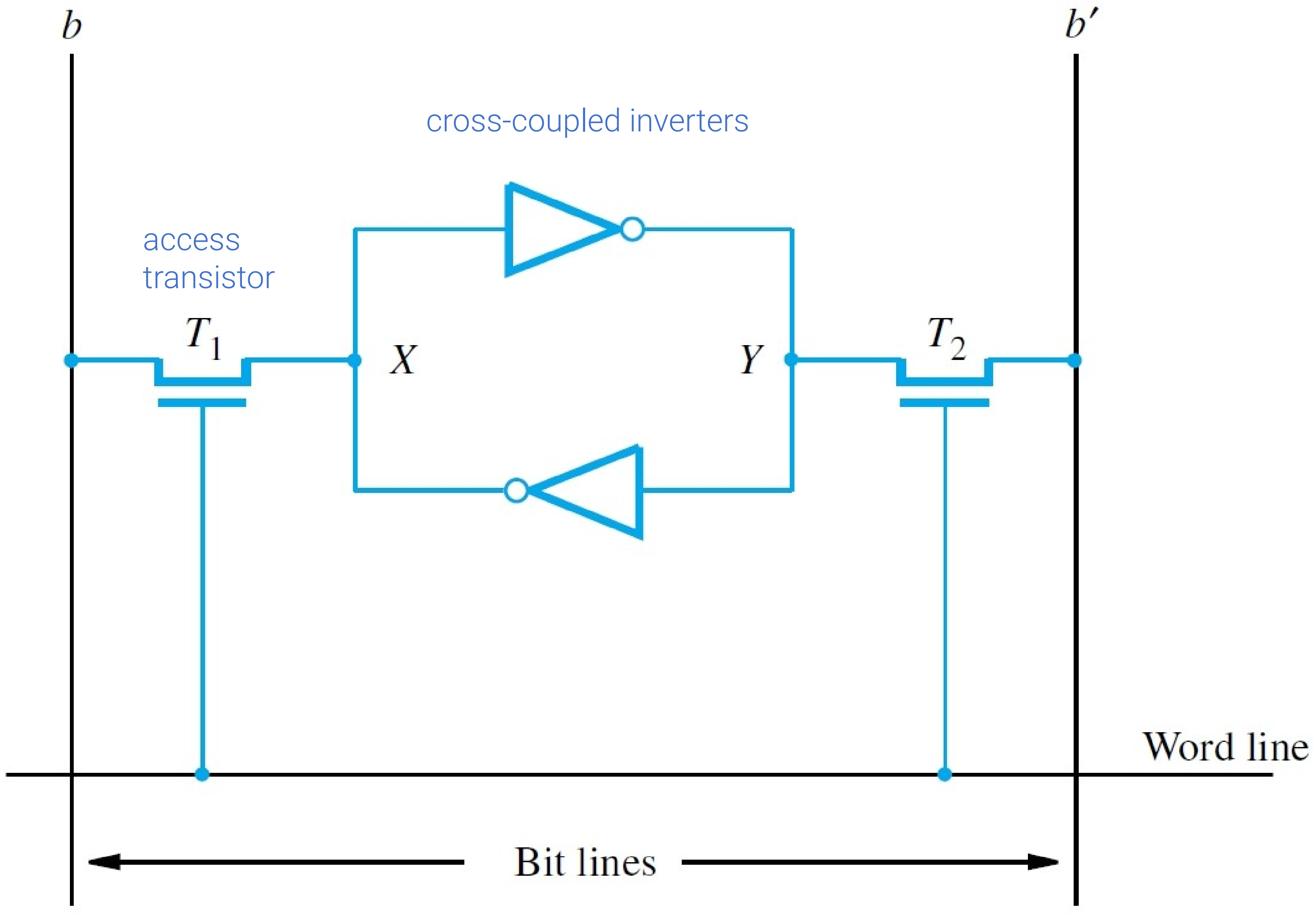
Closed switch when $V_G = 0$ V

PMOS transistor

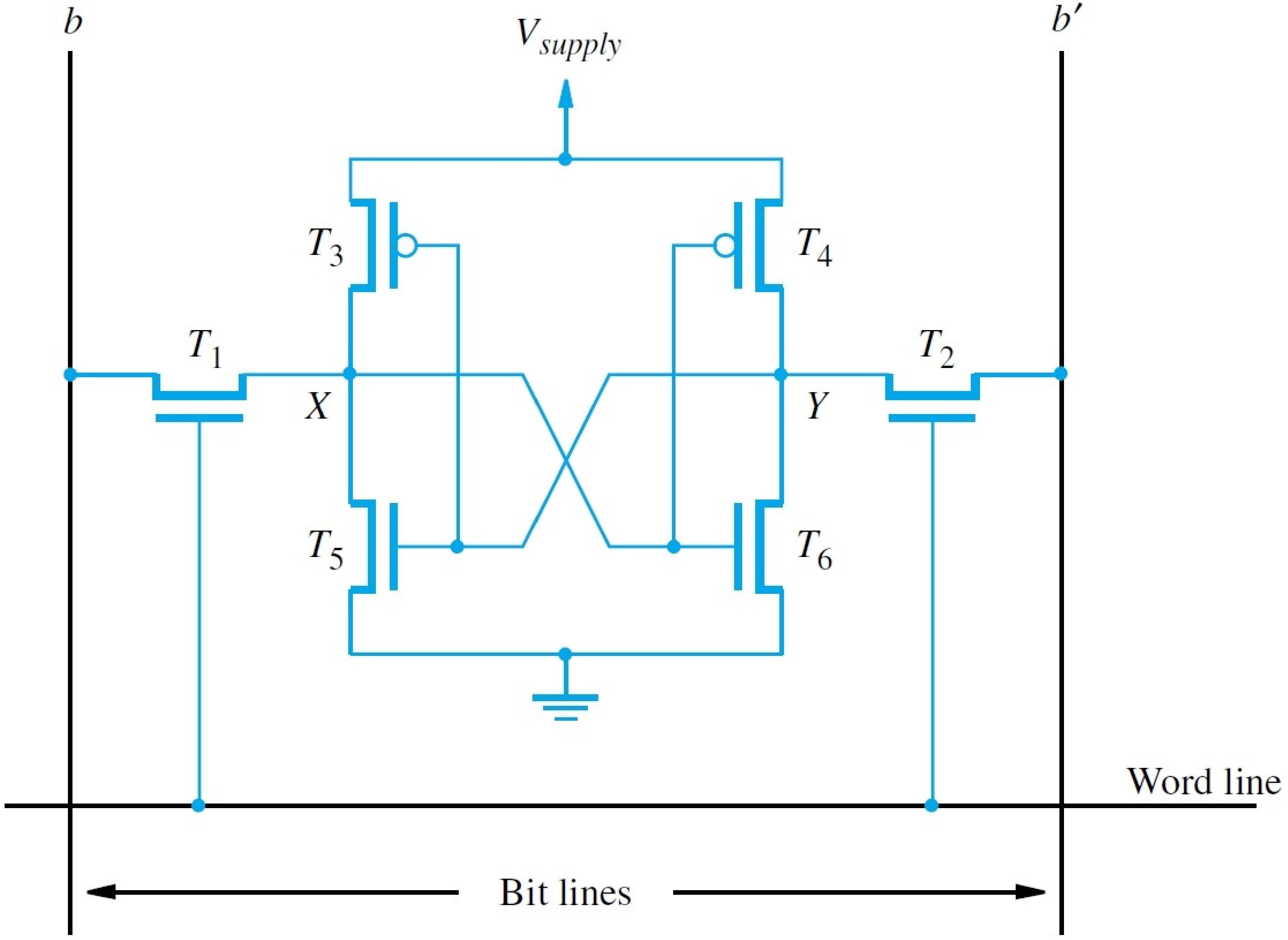


x	T_1 T_2	f
0	on off	1
1	off on	0

CMOS realization of a NOT gate


b
b′
cross-coupled inverters
access transistor
T_1
T_2
X
Y
Word line
Bit lines

SRAM Cell

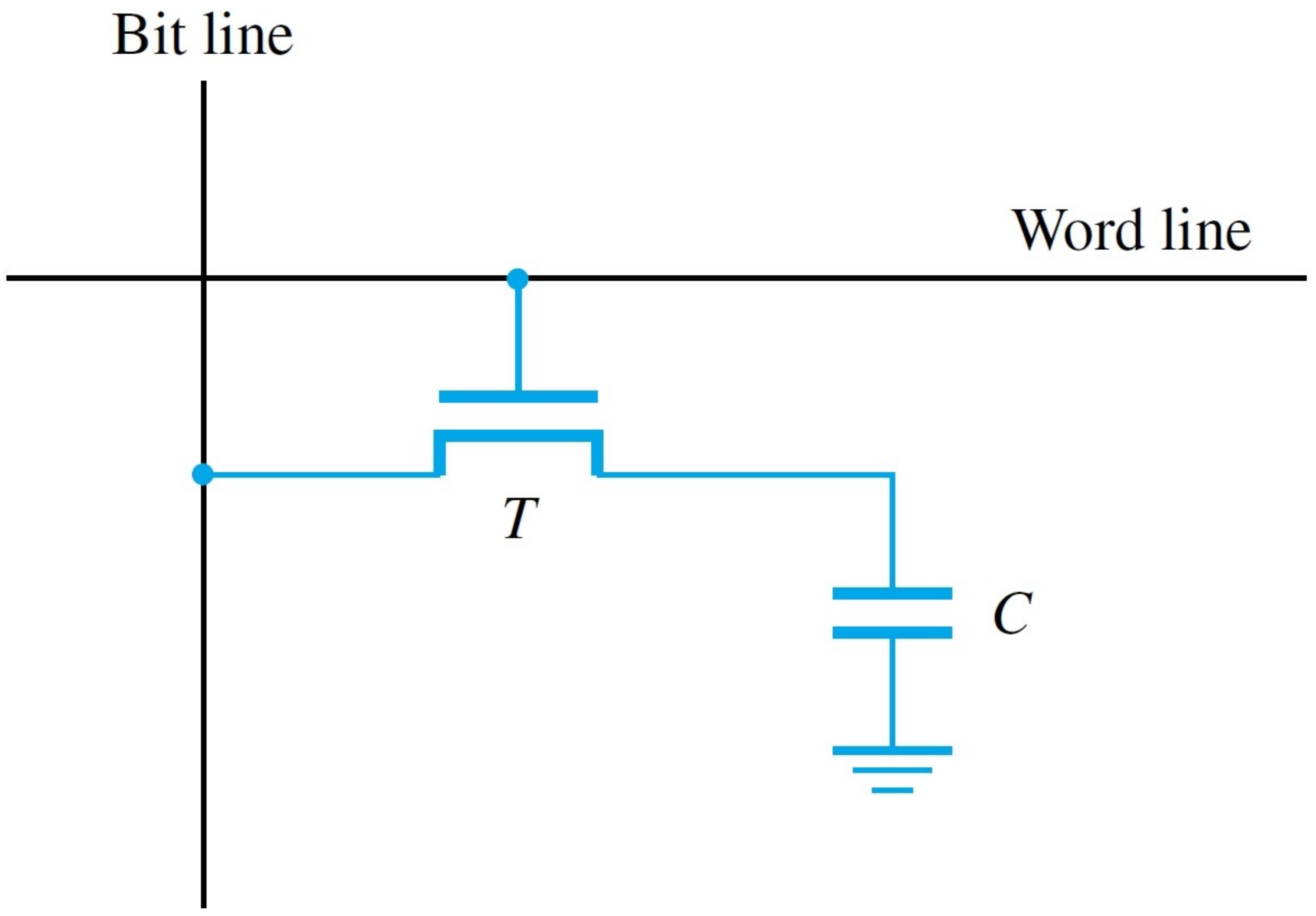


CMOS 6T SRAM Cell

DRAM

- *Dynamic* RAM: Retains contents for only a few tens of milliseconds and must be periodically “refreshed” to maintain the contents for longer periods.
- Slower than SRAM, but more dense (less expensive) – the DRAM cell is simpler than the SRAM cell.
- Can implement DRAMs with large capacity (~gigabits)
- Used to implement main memory

DRAM Cell



State is presence ("1") or absence ("0") of charge in capacitor

A small amount of current flows through transistor even when it is OFF, resulting in a leak of the stored charge

Reading DRAM

- Read: Sense amplifier connected to the bit line detects if the charge in the capacitor is above a threshold
- If above threshold, the sense amplifier drives the bit line to full voltage (“1”) and as a result the capacitor is recharged to full voltage (“1”)
- If below threshold, the sense amplifier pulls the bit line down to ground (“0”) and the capacitor is discharged fully (“0”)

→ reading a DRAM cell refreshes its contents (an entire row is read and refreshed at the same time)

To refresh the entire DRAM each row in the DRAM must be periodically read – done by an external memory controller

Refresh overhead

- Assume that each row needs to be refreshed every 64 ms, the minimum time between two row accesses is 50 ns and that all rows are refreshed in 8192 cycles
- Read/write operations have to be delayed until refresh is finished. What is the refresh overhead?

256 Mb Asynchronous DRAM organization (32M x 8)

row address strobe (applied first, generated by external memory controller)

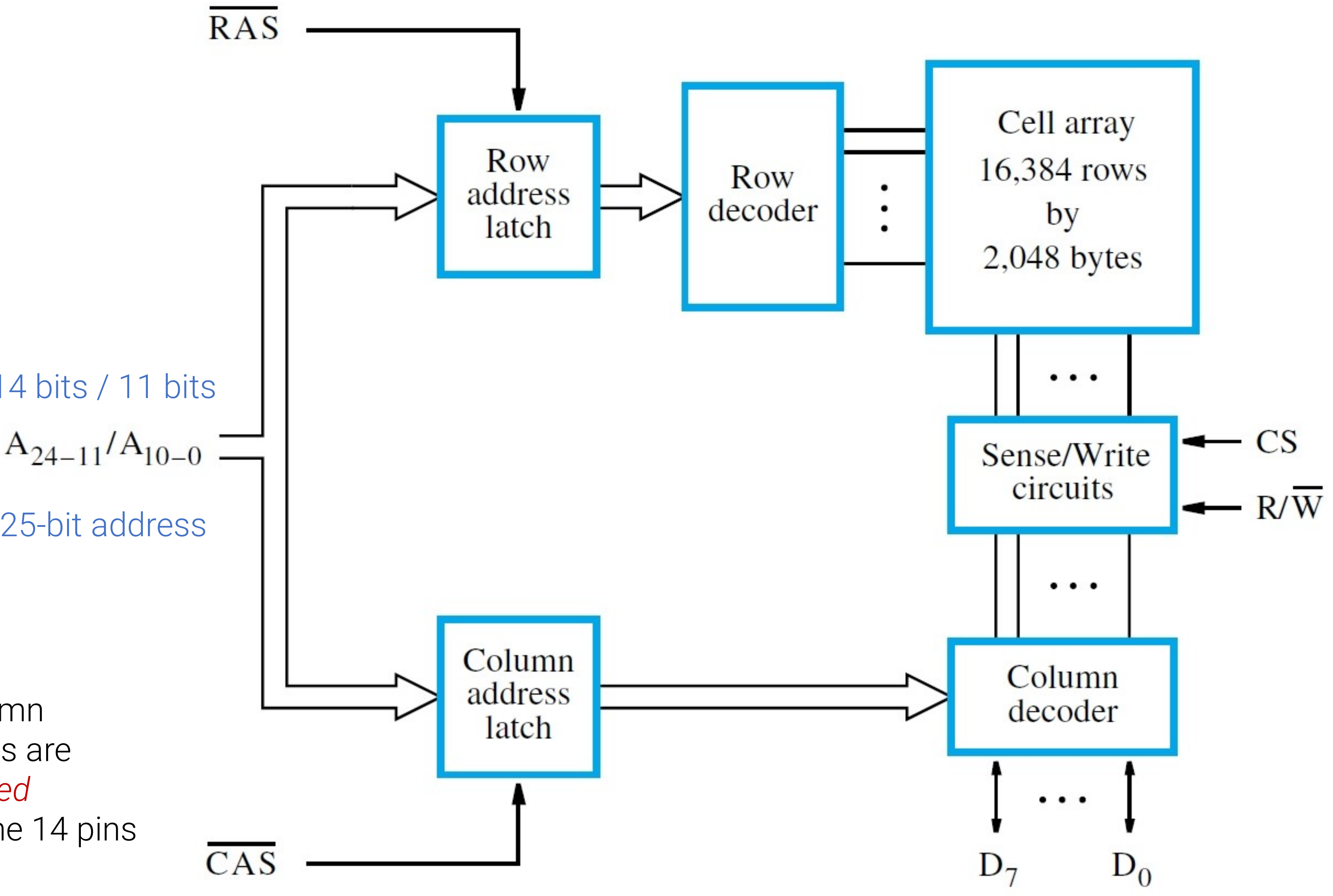
$32\text{ Mb} = 32 \times 2^{20} = 2^{25}$

14 bits / 11 bits

25-bit address

row/column addresses are *multiplexed* onto same 14 pins

column address strobe (bar indicates active low)

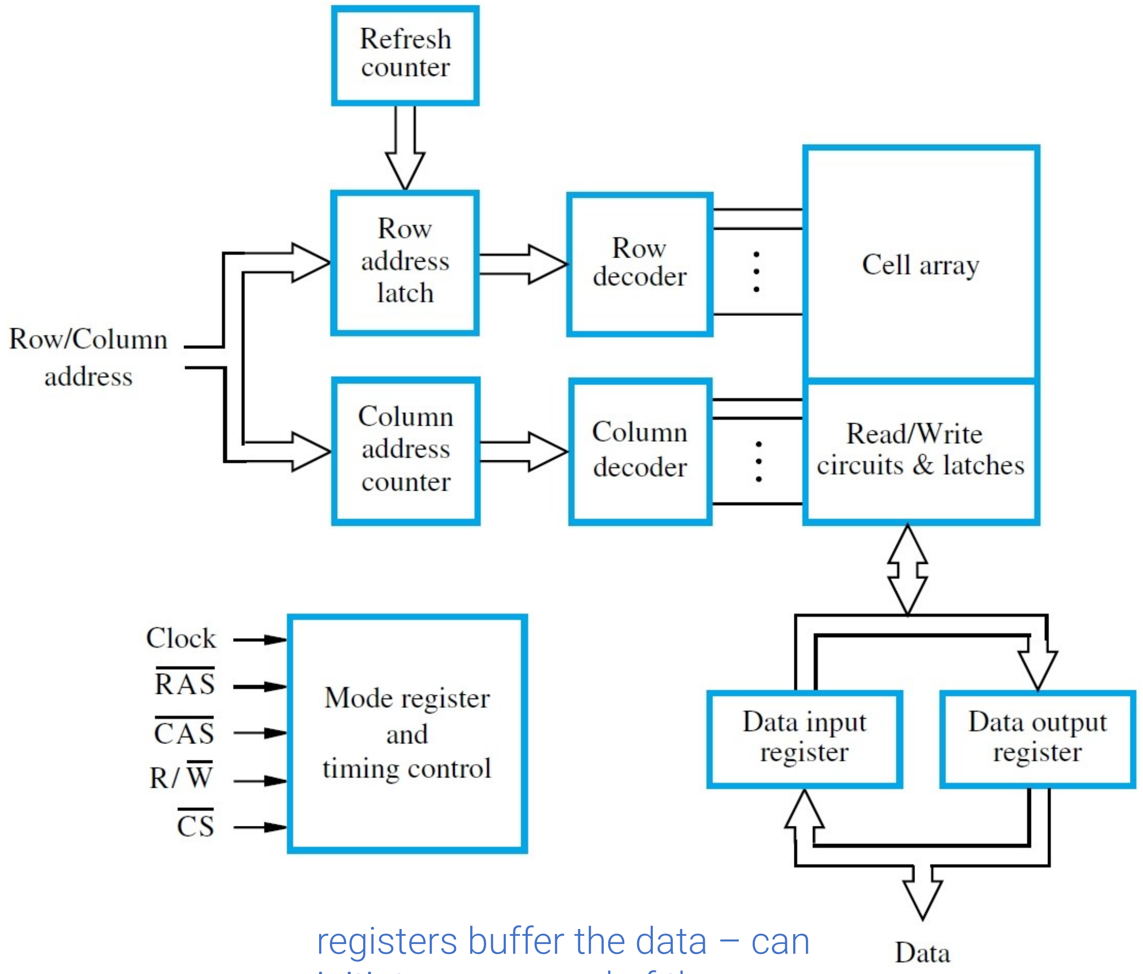


Fast Page Mode

- In preceding example, all 16,384 cells in a row are accessed (and also refreshed as a result)
- But only 8 bits of data are actually transferred for each full row/column addressing sequence
- For more efficient access to data in same row, *latches* in sense amplifiers hold cell contents
- For consecutive data, just assert CAS signal and increment column address in same row
- This fast page mode is useful in *block transfers*

Synchronous DRAMs

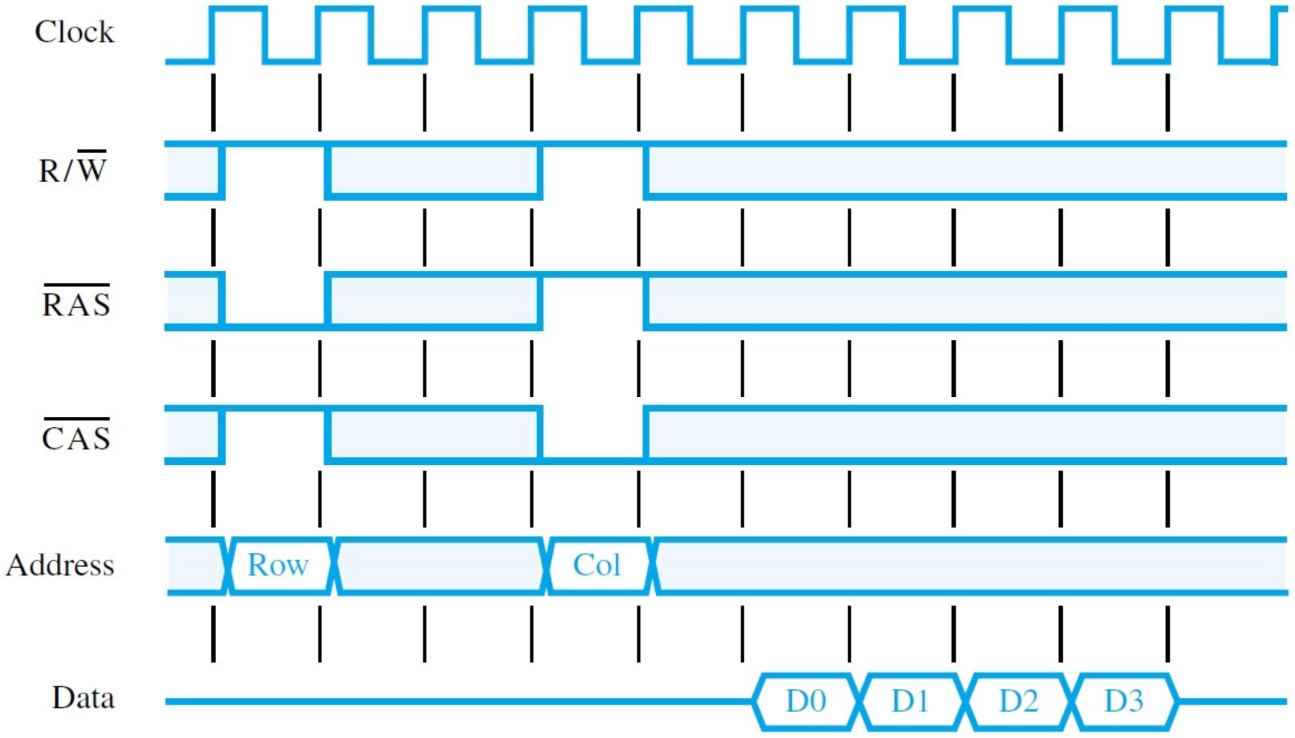
- Modern synchronous DRAM (SDRAM) uses a clock to generate internal timing signals (e.g. CAS and RAS)
- Memory controller is integrated on-chip (built-in refresh circuit)
- "dynamic" nature of the chip is invisible to the user



registers buffer the data – can initiate a new read of the array while reading out the previous word from the register

Efficient Block Transfers

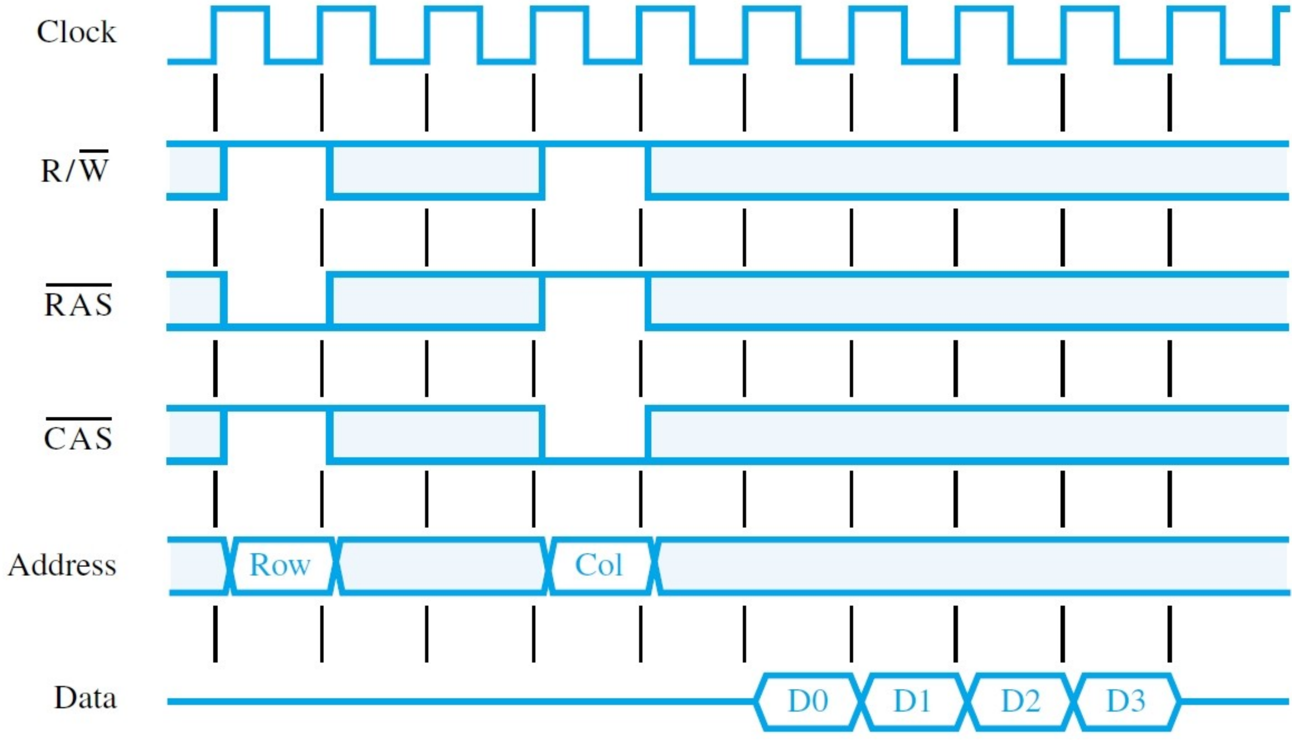
- Synchronous DRAM reduces delay by having CAS assertion *once* for initial column address
- SDRAM circuitry increments column counter and transfers consecutive data automatically
- Burst length determines number of transfers



Burst length of 4, `RAS` delay of 2 cycles, `CAS` delay of 1 cycle

Memory latency and bandwidth

- Memory *latency* (ns) is the time for the first word of a block transfer to appear on the data lines
- The time between subsequent words is much shorter than the time needed to transfer the first word
- The memory *bandwidth* (number of bits or bytes transferred per second) is a useful performance measure for a SDRAM



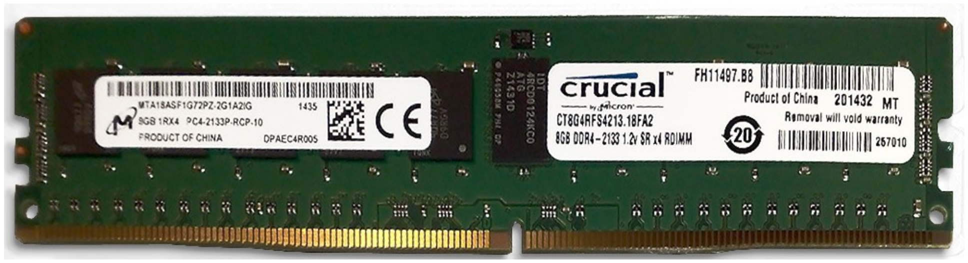
Latency is 5 cycles. If the clock is 500 MHz, the latency is 5 * 1/500e6 = 10 ns

Remaining three words transferred at one word every 2 ns

Double-Data-Rate (DDR) SDRAM

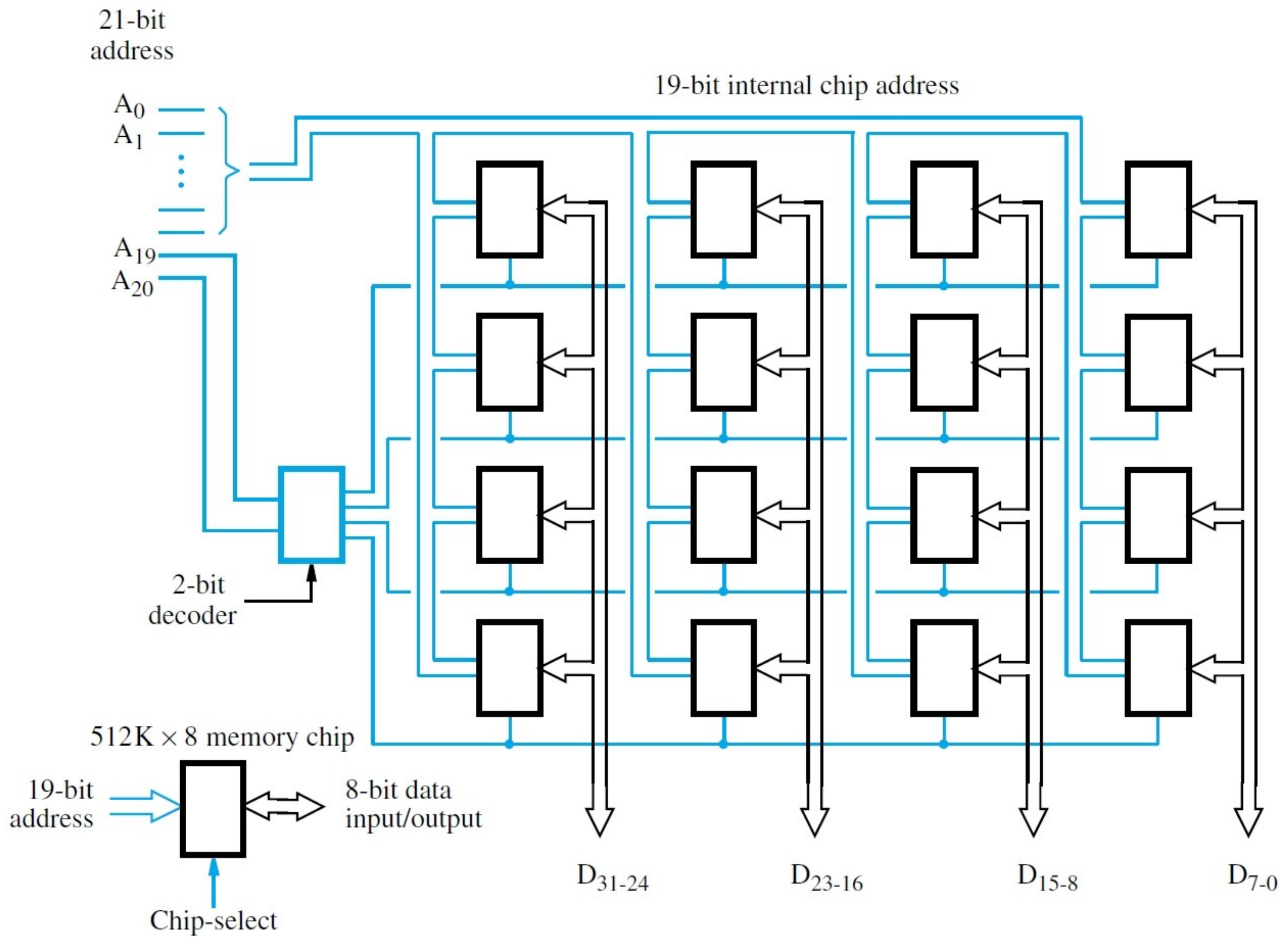
- Modern SDRAMs use both rising and falling edges of the clock (“Double data rate”)

e.g. DDR4 has a clock of 2133 MHz and can support up to 2400 MTransfers / second


crucial
CT8G4RFS4213.18FA2
8GB DDR4-2133 1.2v SR x4 RDIMM
Product of China
Removal will void warranty
MTA18ASF1G72PZ-2G1A2IG
8GB 1RX4 PC4-2133P-RCP-10
PRODUCT OF CHINA

8 Gbyte DDR4-2133 ECC 1.2 V RDIMM
(registered dual-inline memory module)

Multi-chip memories (e.g. 2M x 32 SRAM)

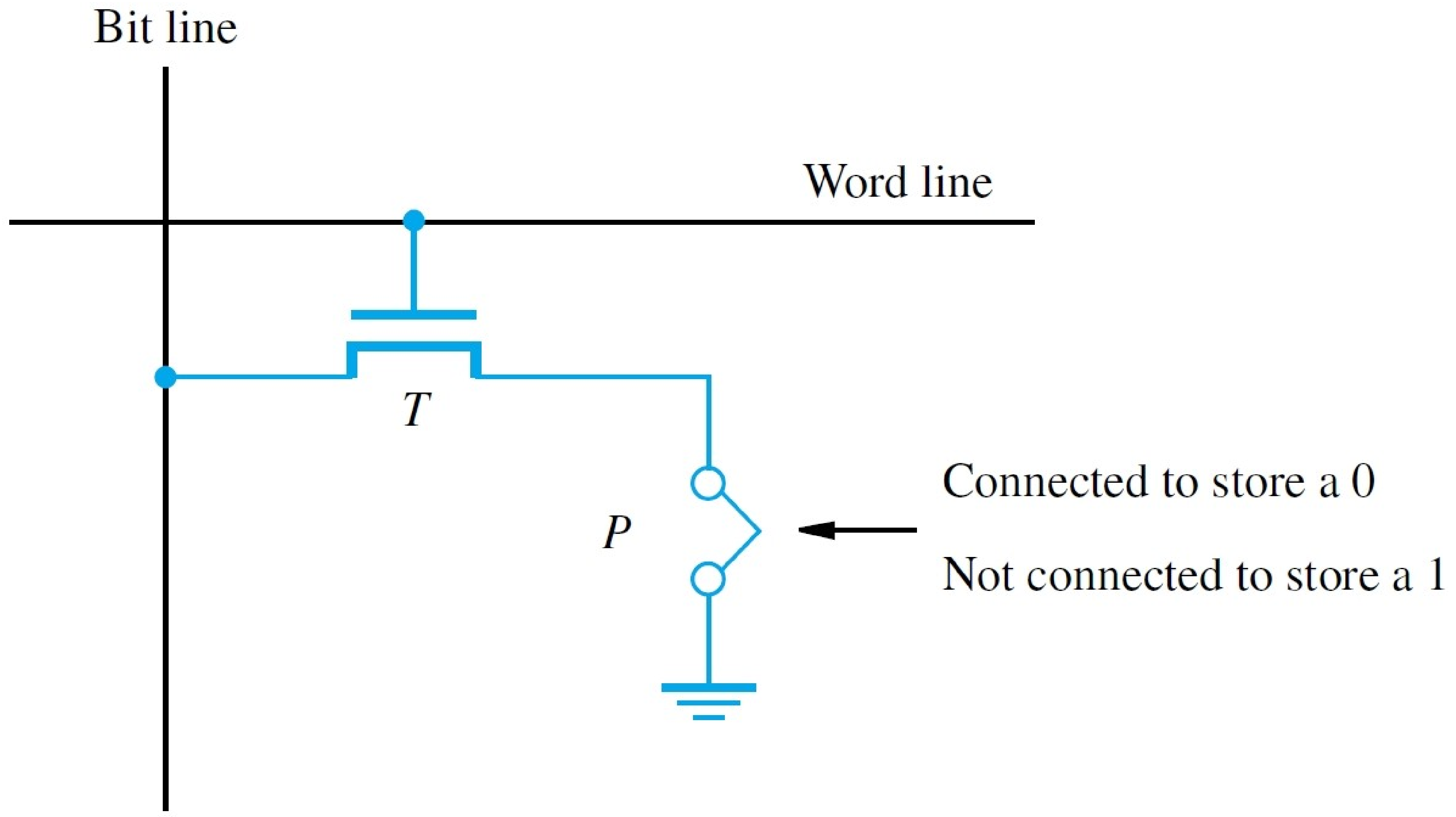


Non-volatile memories

- Non-volatile memories retain their contents even when the power is removed.
- Slower than volatile memories and special procedure for writes
- Suitable for implementing long-term storage
  - e.g. Solid-State Disk (SSD)

Read-only-memory (ROM)

contents written only *once*, at the time of manufacture



PROM, EPROM, and EEPROM

- Cells of a *programmable ROM (PROM)* chip may be written after the time of manufacture
- A fuse is burned out with a high current pulse
- An *erasable programmable ROM (EPROM)* uses a special transistor instead of a fuse
- Injecting charge allows transistor to turn on
- Erasure requires UV light to remove all charge
- An *electrically erasable ROM (EEPROM)* can have individual cells erased with chip in place

Flash Memory

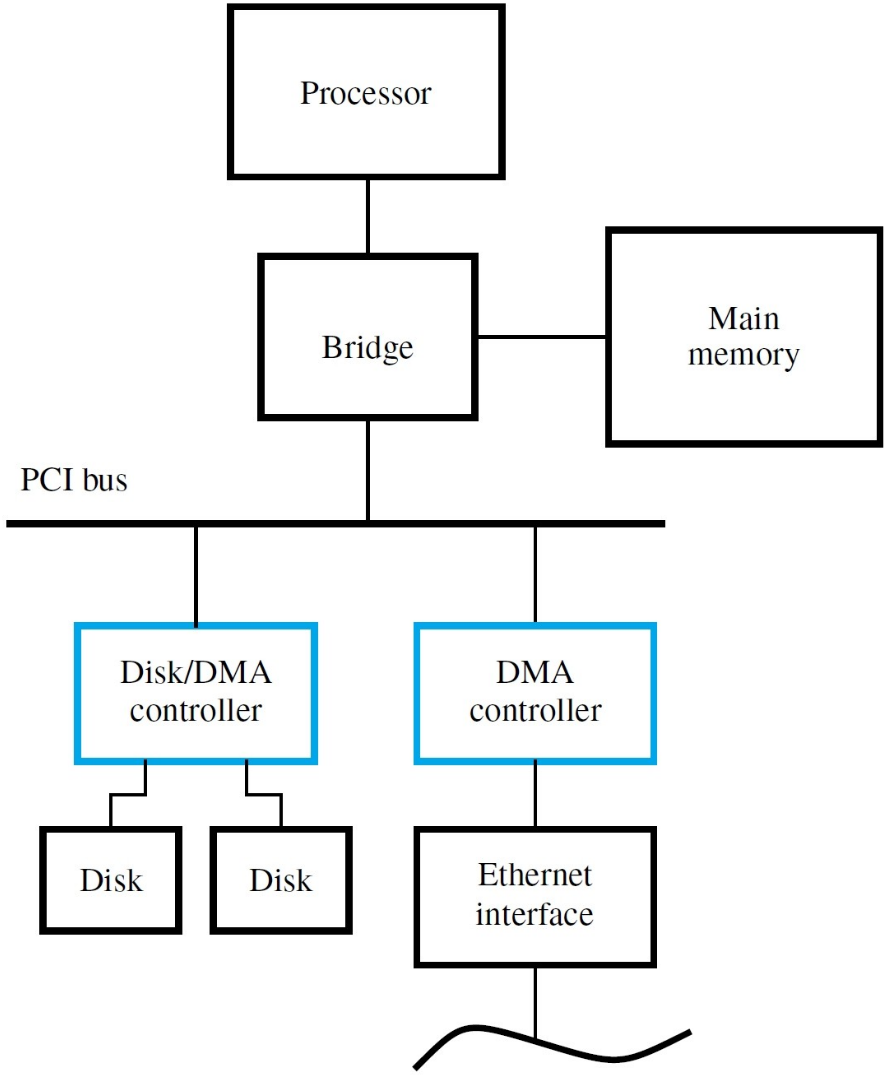
- High-density, low-power and low-cost
- For higher density, Flash cells are designed to be erased in larger blocks, not individually
- Writing individual cells requires reading block, erasing block, then writing block with changes
- Flash cells can only be written a certain number of times – wear levelling distributes writes to avoid wearing out one part of the memory before others
- Widely used in cell phones, digital cameras, and solid-state drives (e.g., USB memory keys)

Direct Memory Access (DMA)

Textbook 8.4

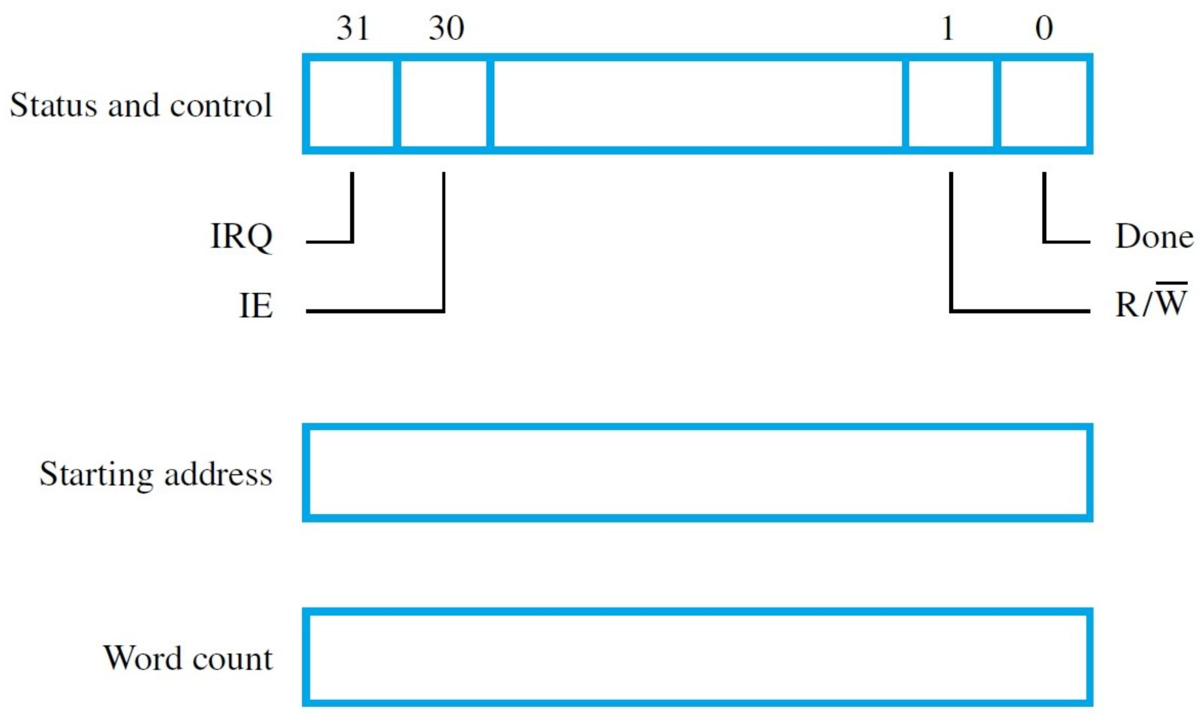
Direct Memory Access

- CPU Overhead for block transfers between I/O and memory is high because each transfer involves only a single word or a single byte (Load/Store instructions plus other instructions to calculate addresses)
- Solutions: A direct memory access (DMA) controller manages the transfer of larger blocks of data between memory & I/O devices.
- CPU initiates transfer, which is managed by the DMA unit without further CPU involvement



DMA Controller

- *DMA controller* is shared, or in each I/O device
- Keeps track of progress with address counter
- Processor initiates DMA controller activity after writing information to special registers (starting address, count, Read/Write, etc.)
- Processor interrupt used to signal completion



Caches

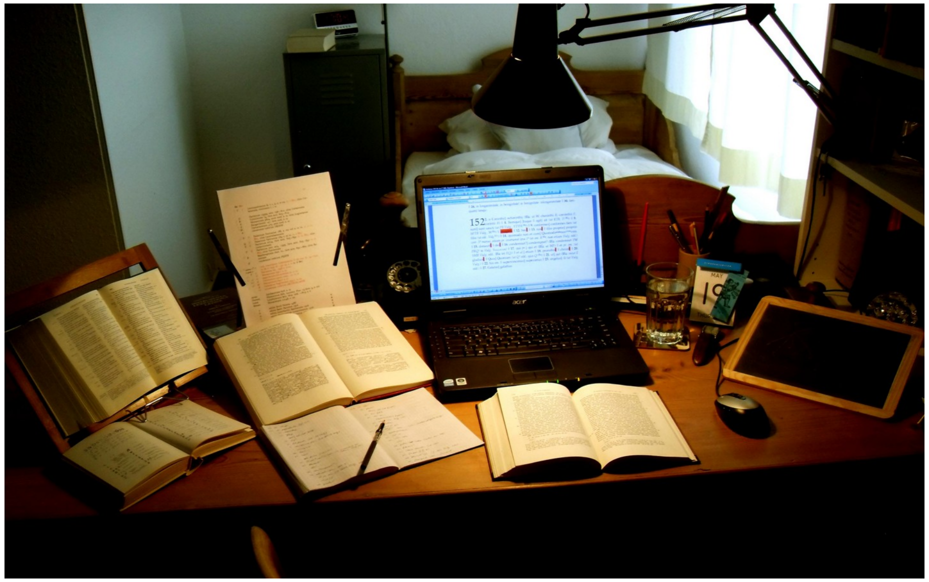
Textbook 8.5, 8.6

The problem

- Want very large memory that is very fast
  - DRAM can be large, but is slow
  - SRAM can be fast, but not large
- Solution: use both DRAM and SRAM in a way that the processor thinks it has a single large memory that is fast
- The solution should be transparent to the programmer

Library: large, slow access

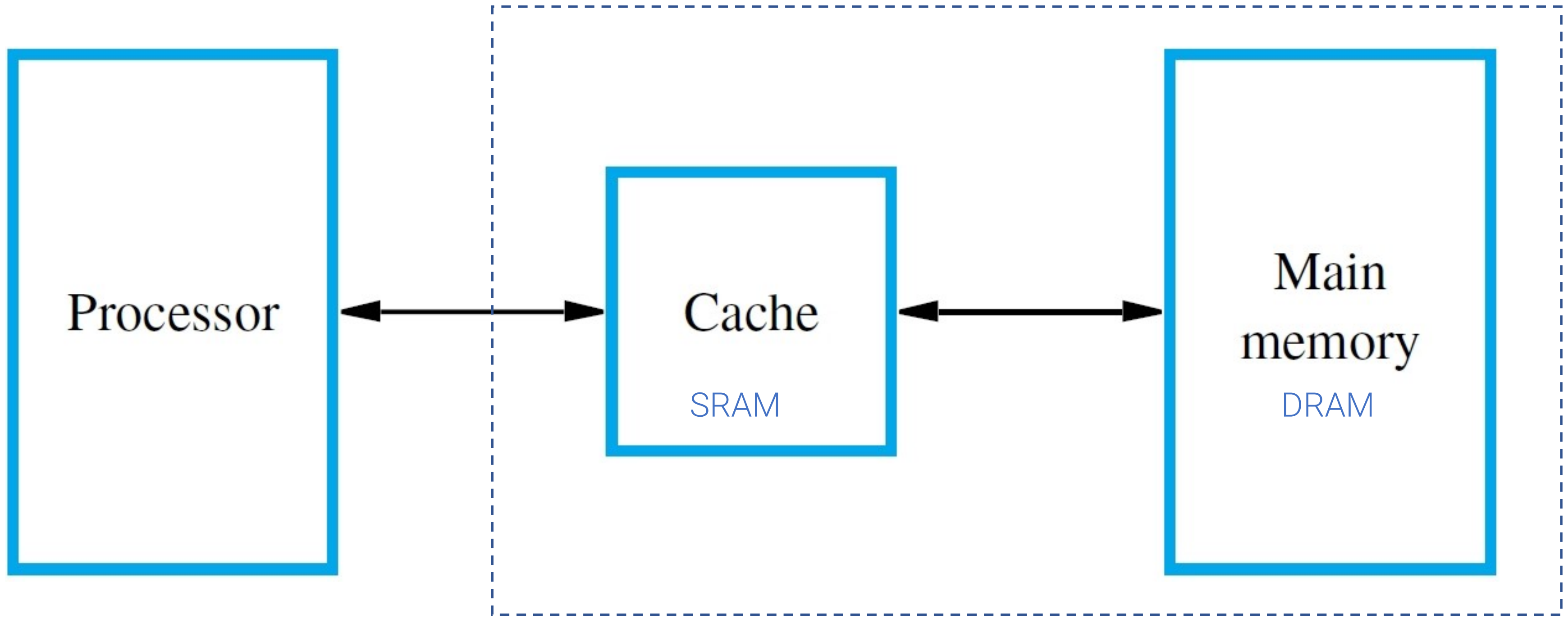
Desk: small, fast access



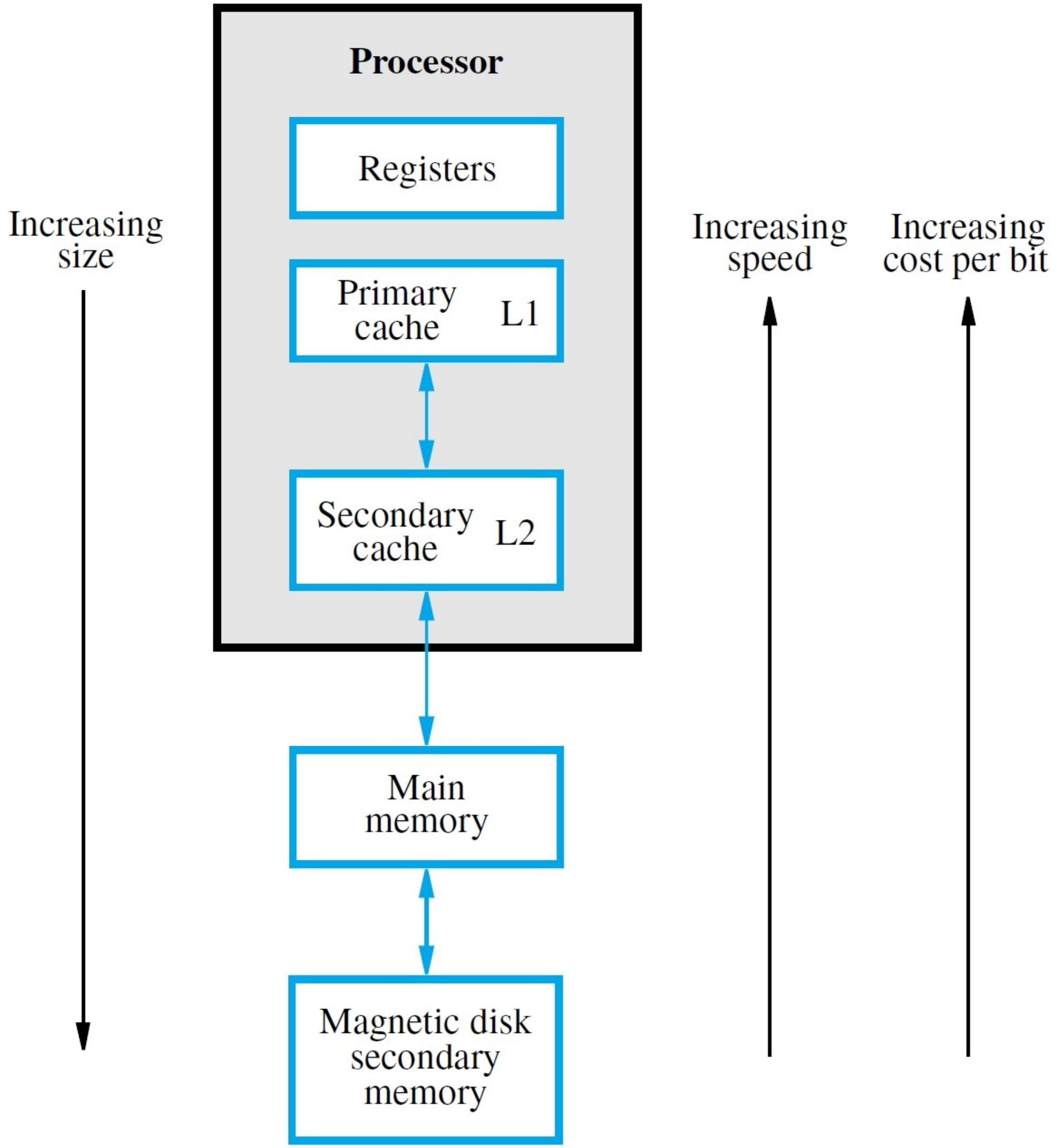
Unlimited Amounts of Fast Memory?

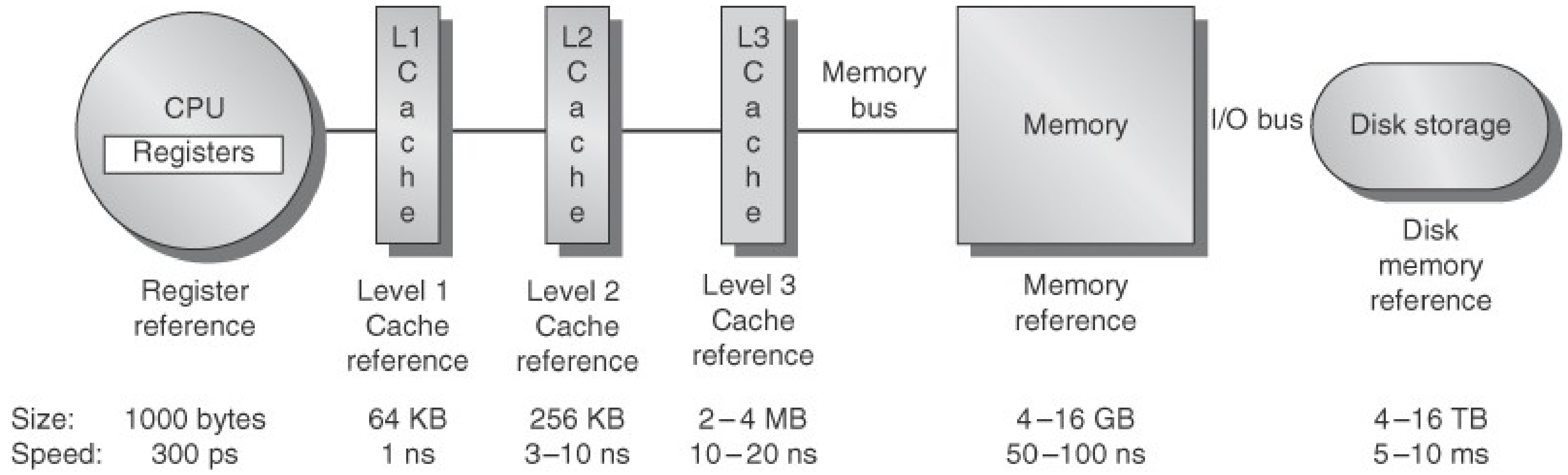
“Ideally one would desire an indefinitely large memory capacity such that any particular…word would be immediately available…We are…forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.”

A. W. Burks, H. H. Goldstine, and J. von Neumann,
“Preliminary Discussion of the Logical Design of an electronic computing instrument”, 1946

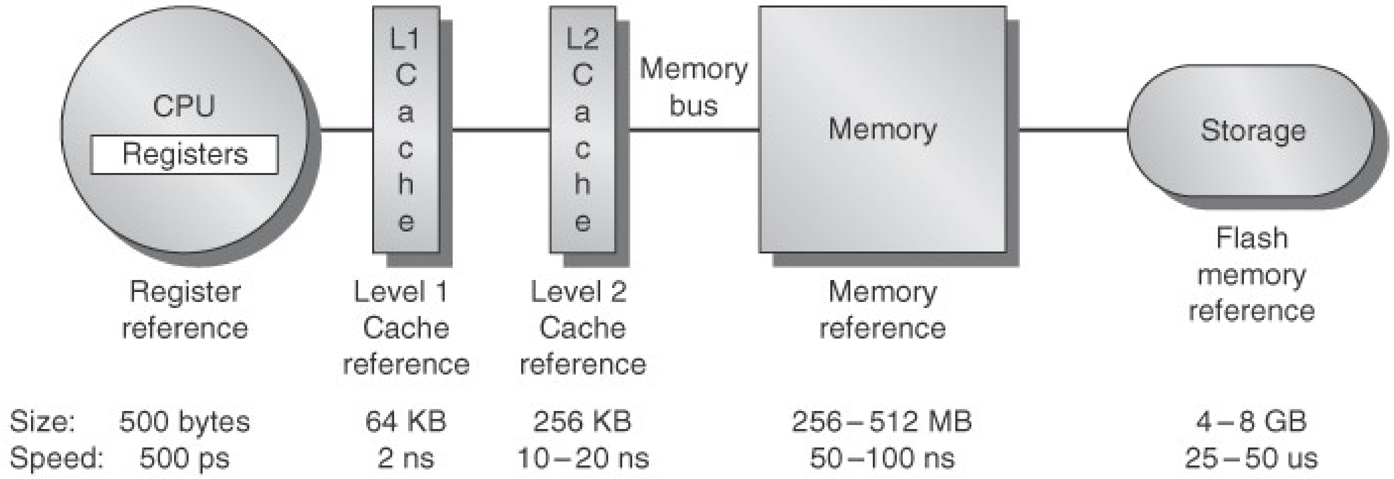


- Keep a copy of frequently used data in the small cache memory (fast) so that if it is needed again it is quickly accessible without going to the large main memory (slow)
- Specialized hardware manages the movement of data between the main memory and the cache
- Transparent to the programmer (load/stores use memory addresses as usual).
- Why should software engineers care about this if it is transparent? As we will see, it works well most of the time, for most programs, but sometimes, having knowledge of how caches work will help you write better programs.


Processor
Registers
Primary cache L1
Secondary cache L2
Main memory
Magnetic disk secondary memory
Increasing size
Increasing speed
Increasing cost per bit



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

So why does this work?

- This only works because humans write programs with structure.
- If you look at a trace of memory addresses issued by the processor running typical programs, you see patterns
- This is called the “principle of locality” and is the reason caches work.

Temporal locality

Recently accessed items are likely to be accessed again soon—loop, reuse

Spatial locality

Items with nearby addresses tend to be referenced nearby in time—code without branching, arrays

store *blocks* of multiple words in the cache

Cache basics

- Processor requests an item (instruction fetch, load)
- If it is found in the cache: hit
  - deliver the desired item to the processor
- If it is not found in the cache: miss
  - copy the block from main memory into the cache and then deliver the item to the processor

Hit and miss rate

- For a cache to make sense, most accesses to memory have to hit the cache. It is not uncommon for caches to have a hit rate of > 95%

```
hit rate = # cache hits / # memory accesses

miss rate = 1 - hit rate
```

Valid bit

- Each block has a valid bit, initialized to 0 upon startup to indicate the block is “empty” – set to 1 when a block is copied to the cache
- For a hit, valid bit must be 1
- Stale Data:
  - e.g. DMA transfer: Disk → Memory
    cache may contain *stale* data from memory, so valid bits are cleared to 0 for those blocks

Where to put blocks in the cache?

Main memory is divided into *blocks* (a.k.a cache lines), each consisting of several consecutive data elements (*e.g.* bytes)

Block in main memory must be transferred to the cache after a miss

- The *mapping function* determines the location
  - Some mapping functions are simple, and some are more complex but have higher performance, i.e. result in a higher hit rate

Direct Mapping

- Every memory block maps to a single cache block
- `n` = # blocks in cache

```
memory block j → cache block (j mod n)
```

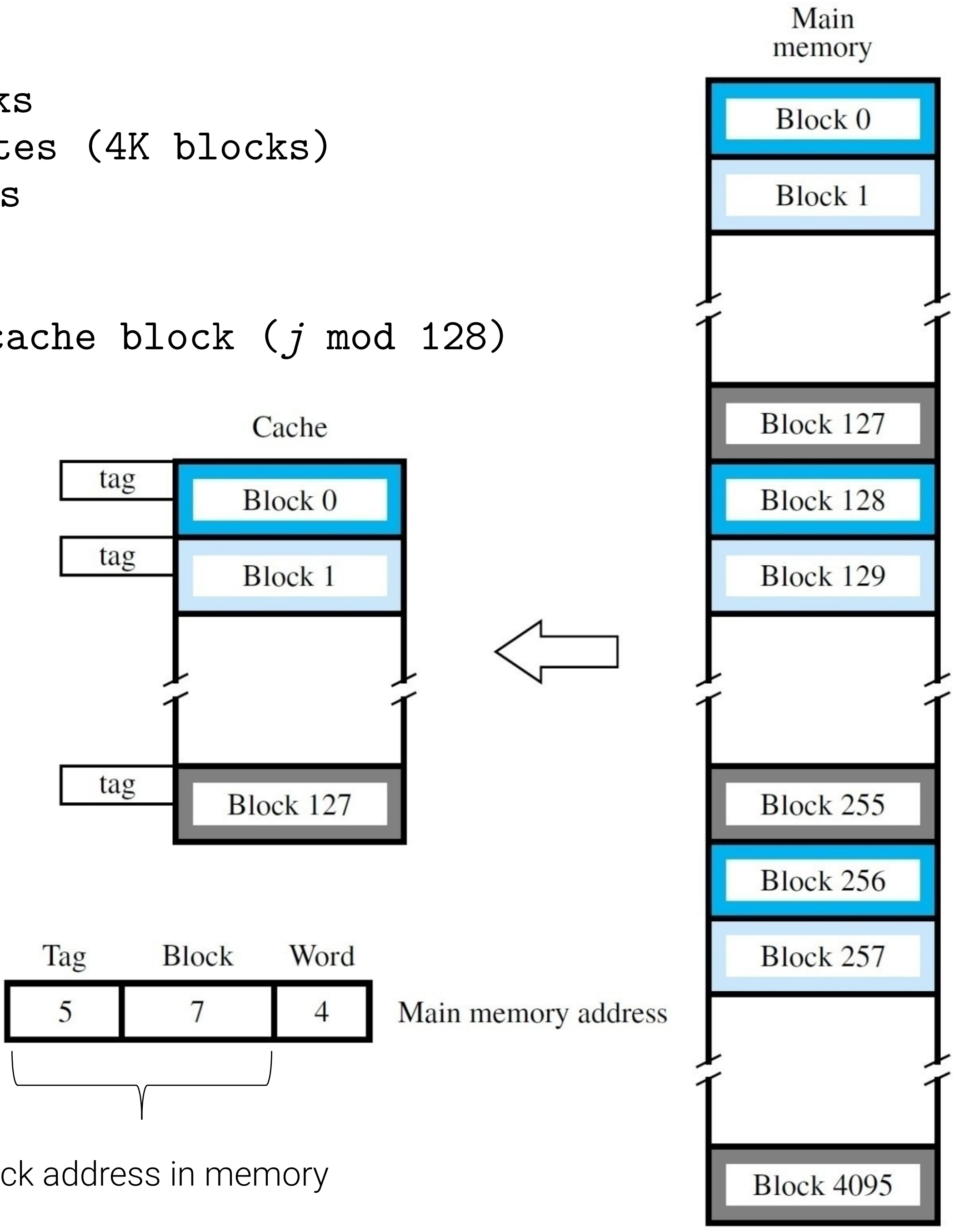
✓ Simplest approach: uses a fixed mapping

Multiple blocks may contend for same location

- New block always overwrites previous block
- If you have multiple frequently accessed blocks that kick eachother out of the cache, you will have many cache misses and suffer the penalty of having to go to main memory frequently

block size = `16 words`
Cache size `n = 128 blocks`
Main memory has `64K bytes (4K blocks)`
Memory address is `16-bits`

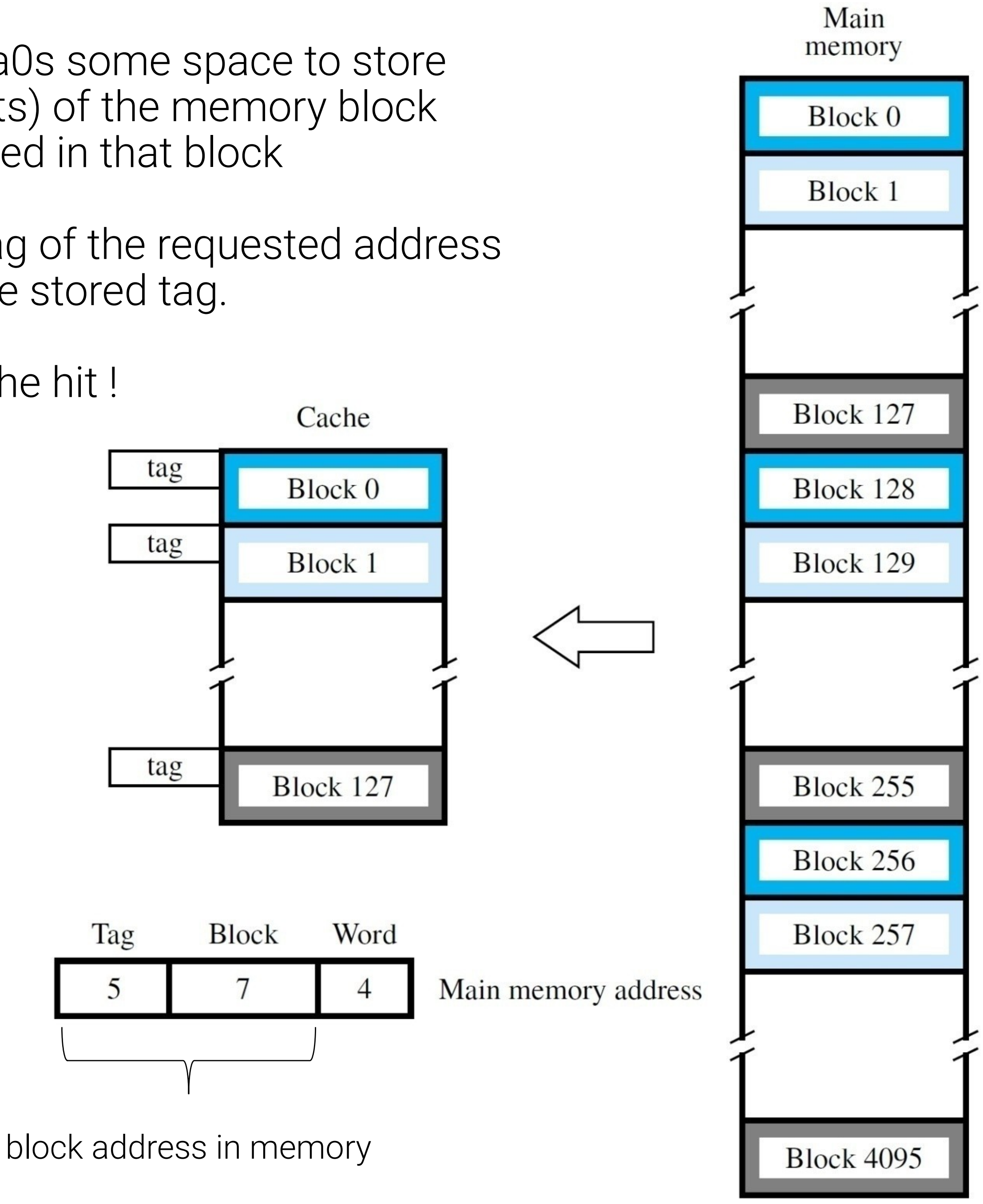
memory block $j \rightarrow$ cache block (j mod 128)



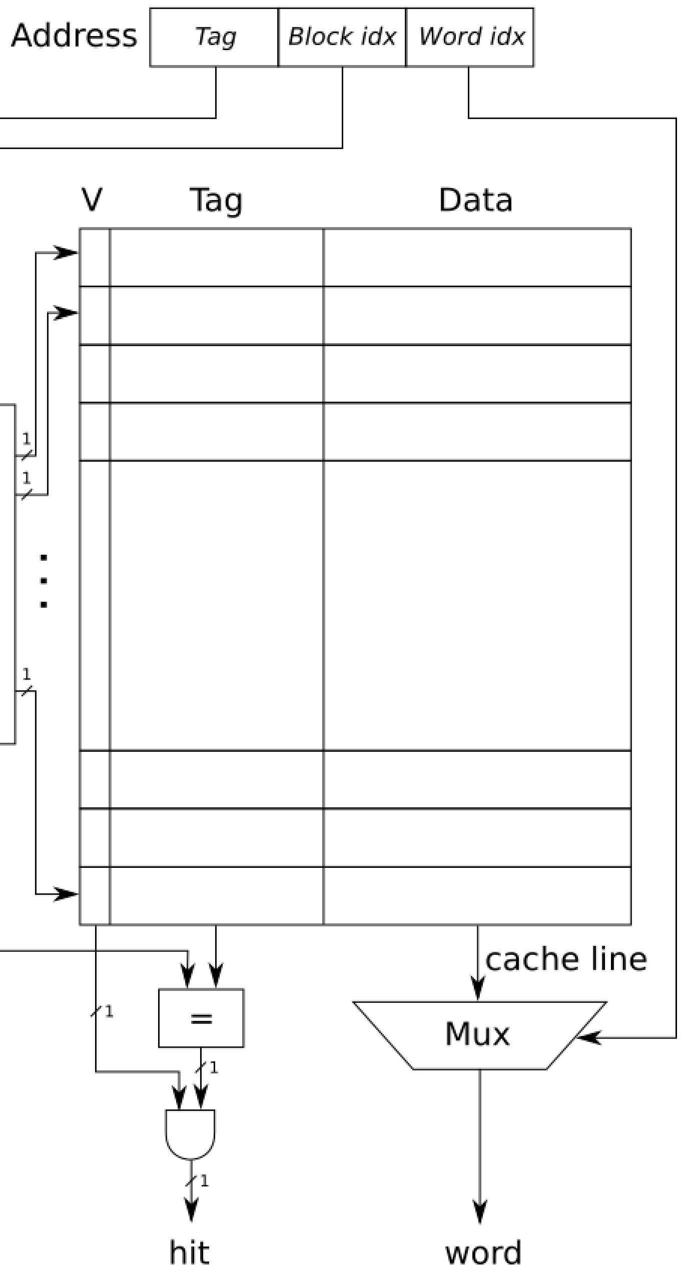
- Each cache block ha0s some space to store the “tag” (upper 5 bits) of the memory block that is currently stored in that block

- On an access, the tag of the requested address is compared with the stored tag.

- If they match -> cache hit !



Direct mapped



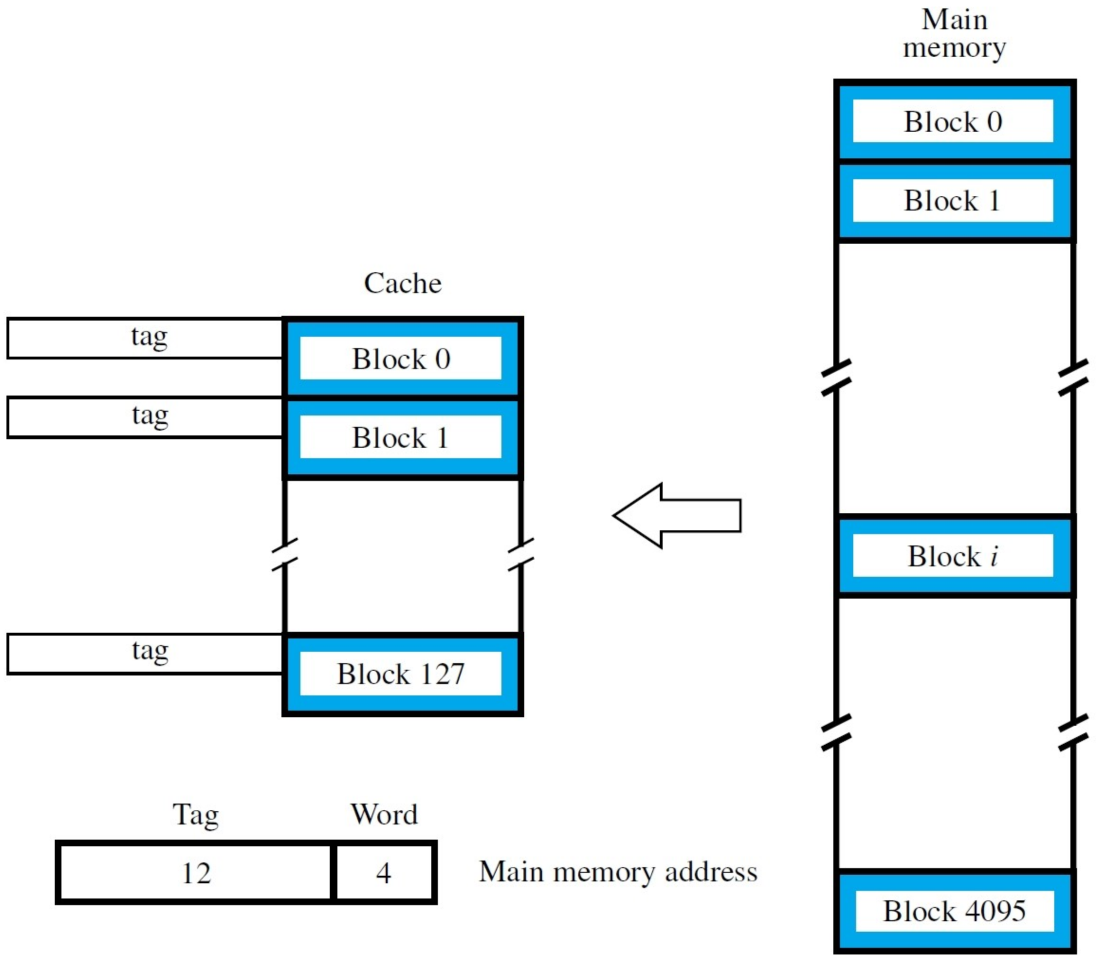
Fully Associative Mapping

- The most flexible mapping: a main memory block can be placed into *any* cache block

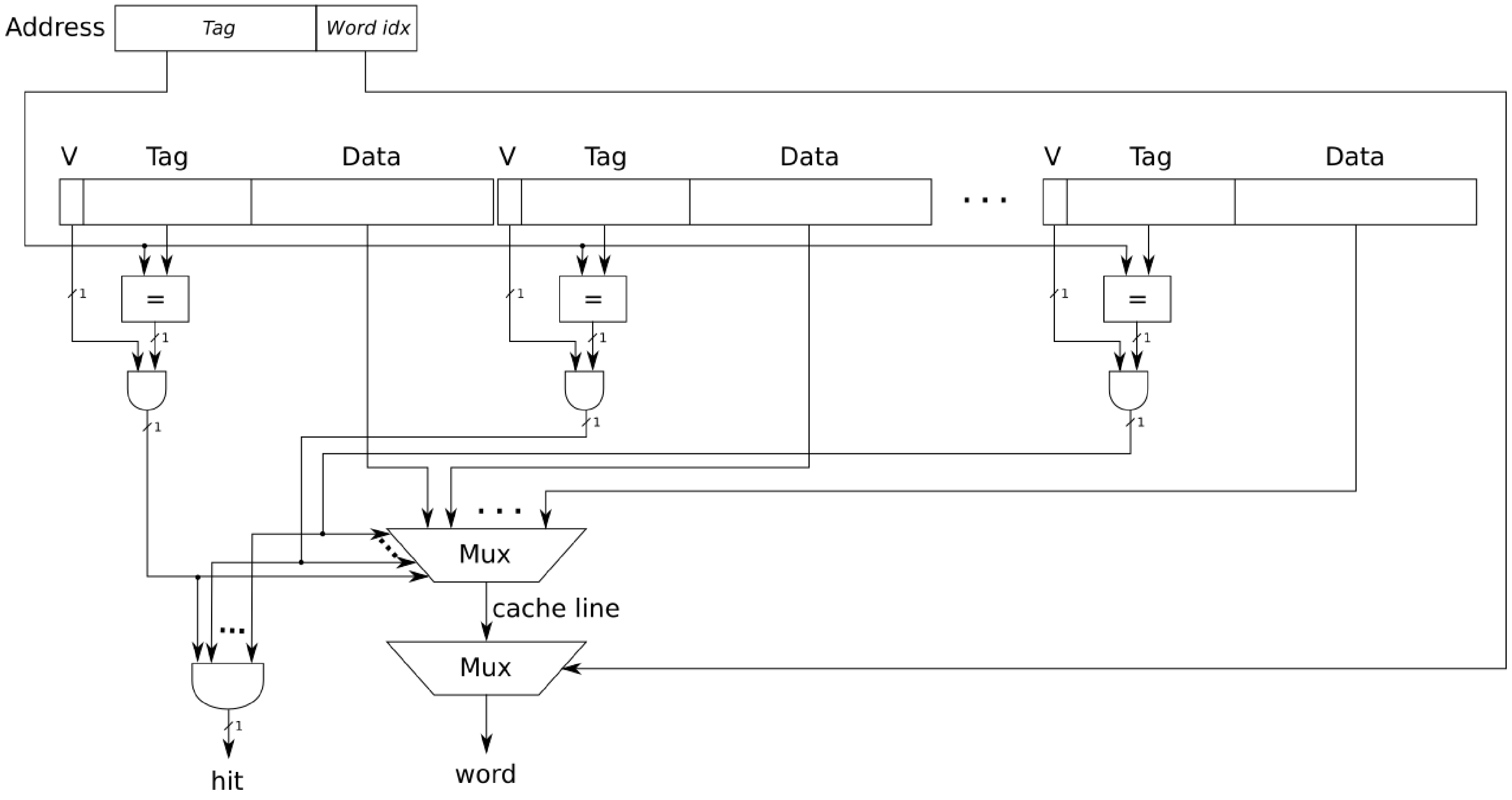
- A block is only ejected from the cache if it is full

- The entire block address is the tag

- Check if a block is in the cache by doing an *associative search* of ALL the cache tags in parallel – complex !



Fully associative

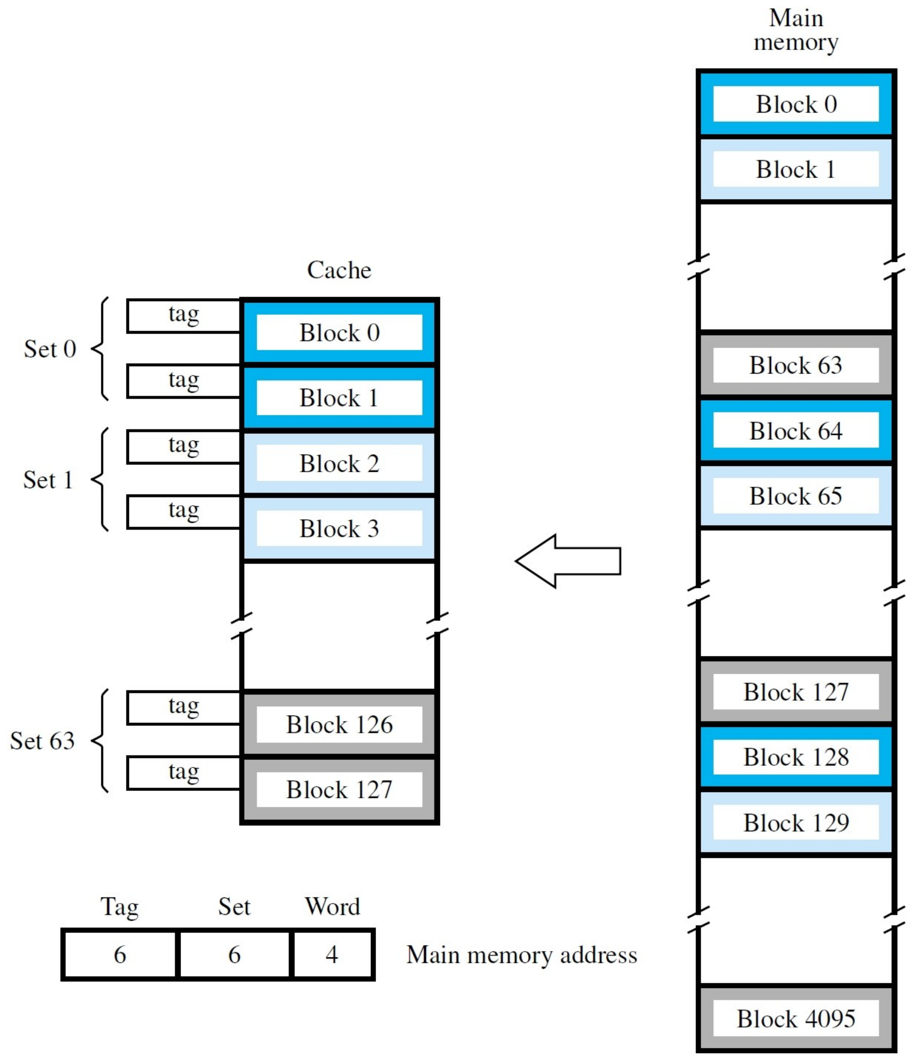


Set-Associative Mapping

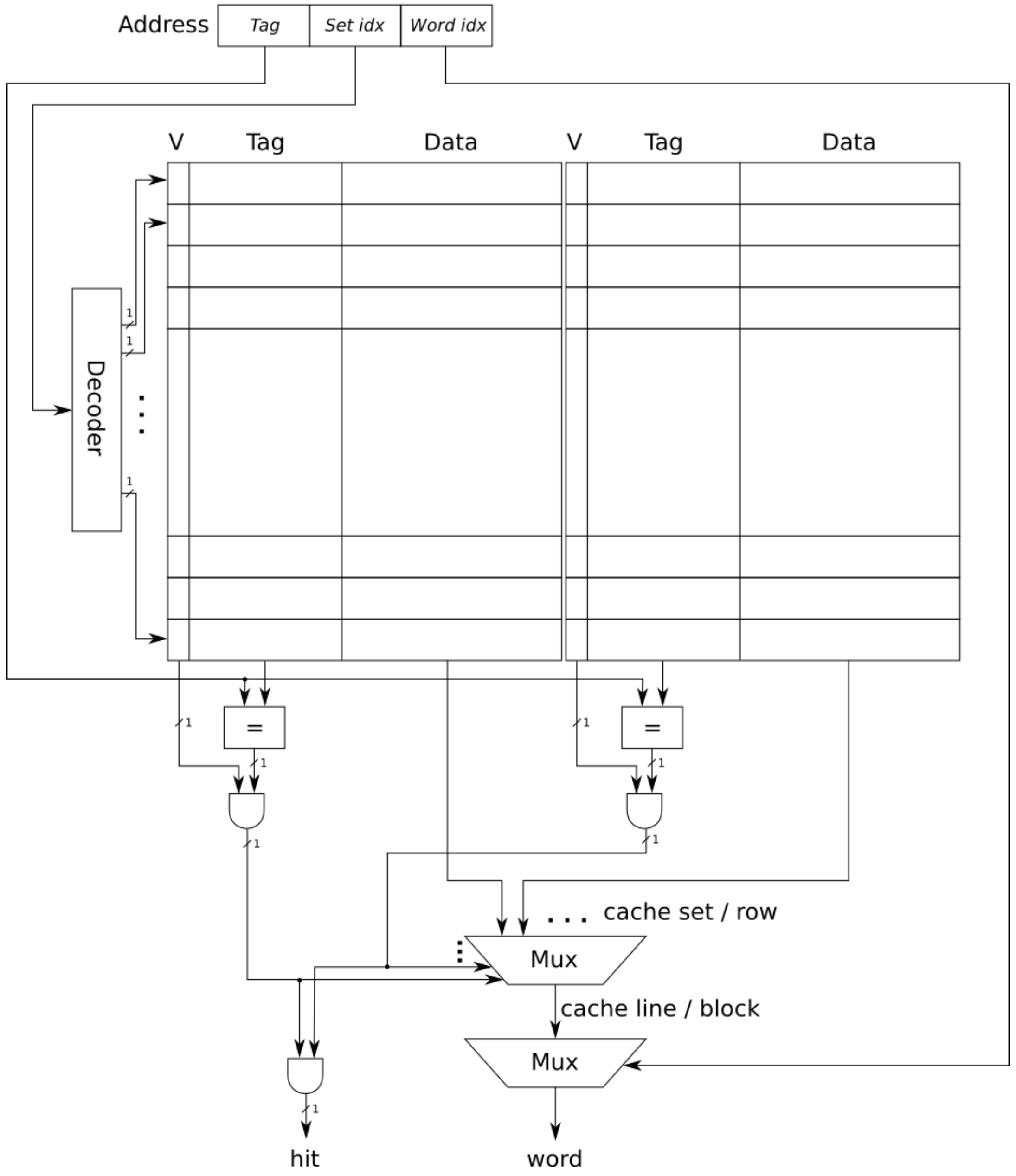
- *k-way* set-associative cache: Group blocks of cache into *sets* of k *blocks*
- *Direct mapping of a memory block to a specific set - any block in the set can be used*

```
memory block j → cache set (j mod 64)
```

- Associative search involves only tags in a set (k = 2, 4, 8)
- Direct-mapped : 1-way
  Associative : n-way



Set-associative



Replacement policies

- Replacement is trivial for direct mapping,
  but need a method for associative mapping
- *least-recently-used (LRU)* algorithm
    - Requires specialized hardware to track accesses to cache blocks in a set
- Another replacement policy is to remove the “oldest” block in the set
- Random replacement works surprisingly well

Writes to Cache

- *Write hit*:
  - *Write-through*: write to both the cache and the main memory
  - *Write-back*: only write to the cache. Update the main memory only when that cache block is removed from the cache to make room for another block. A *dirty bit* (or modified bit) is set to indicate the cache block has been modified and is no longer identical to the block in main memory
- *Write miss*:
  - If write-through is being used, then write directly to the main memory on a write miss
  - If write-back is being used, first copy the block containing the addressed word into the cache, and then write the new word in the cache block
- Flushing the cache in case of Write-Back policy
  - Store modified blocks from cache to memory using dirty bit information

Cache example

- A 4x10 array of 16-bit numbers is stored in an array A in column order.
- Normalize the elements of the first row of A with respect to the average value of the elements in the row

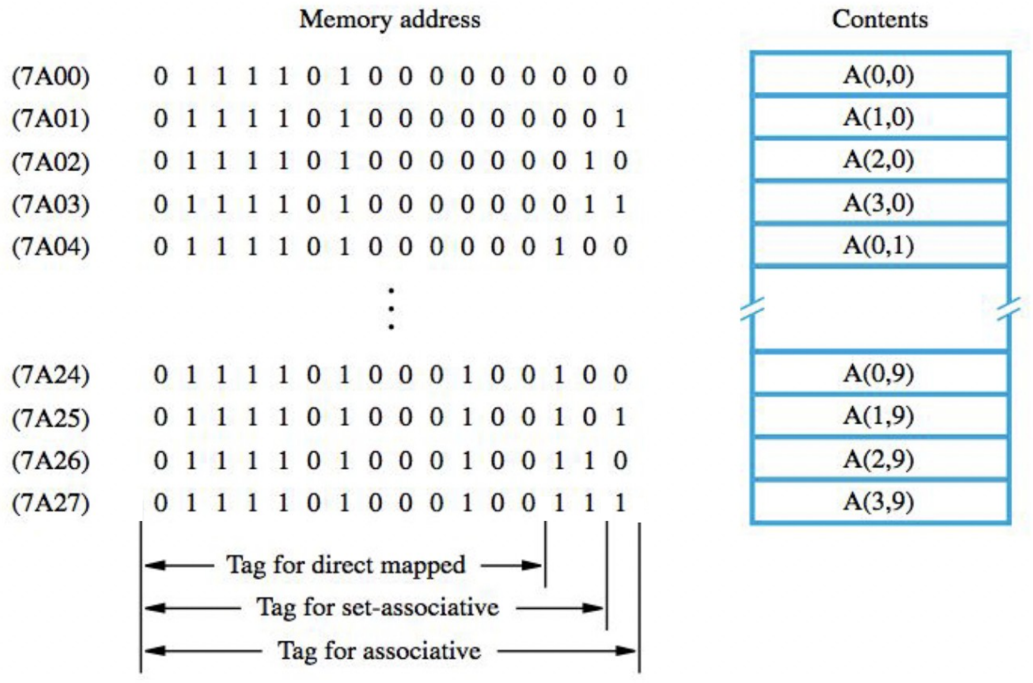
$$A(0,i) \leftarrow \frac{A(0,i)}{\left(\sum_{j=0}^{9} A(0,j)\right)/10} \qquad \text{for } i = 0, 1, \ldots, 9$$

	Memory address	Contents
(7A00)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	A(0,0)
(7A01)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1	A(1,0)
(7A02)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0	A(2,0)
(7A03)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1	A(3,0)
(7A04)	0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0	A(0,1)
	⋮	
(7A24)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0	A(0,9)
(7A25)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 1	A(1,9)
(7A26)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0	A(2,9)
(7A27)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 1	A(3,9)

Cache example

```
SUM := 0
for j := 0 to 9 do
        SUM := SUM + A(0,j)
end
AVG := SUM/10
for i := 9 downto 0 do
        A(0,i) := A(0,i)/AVG
end
```

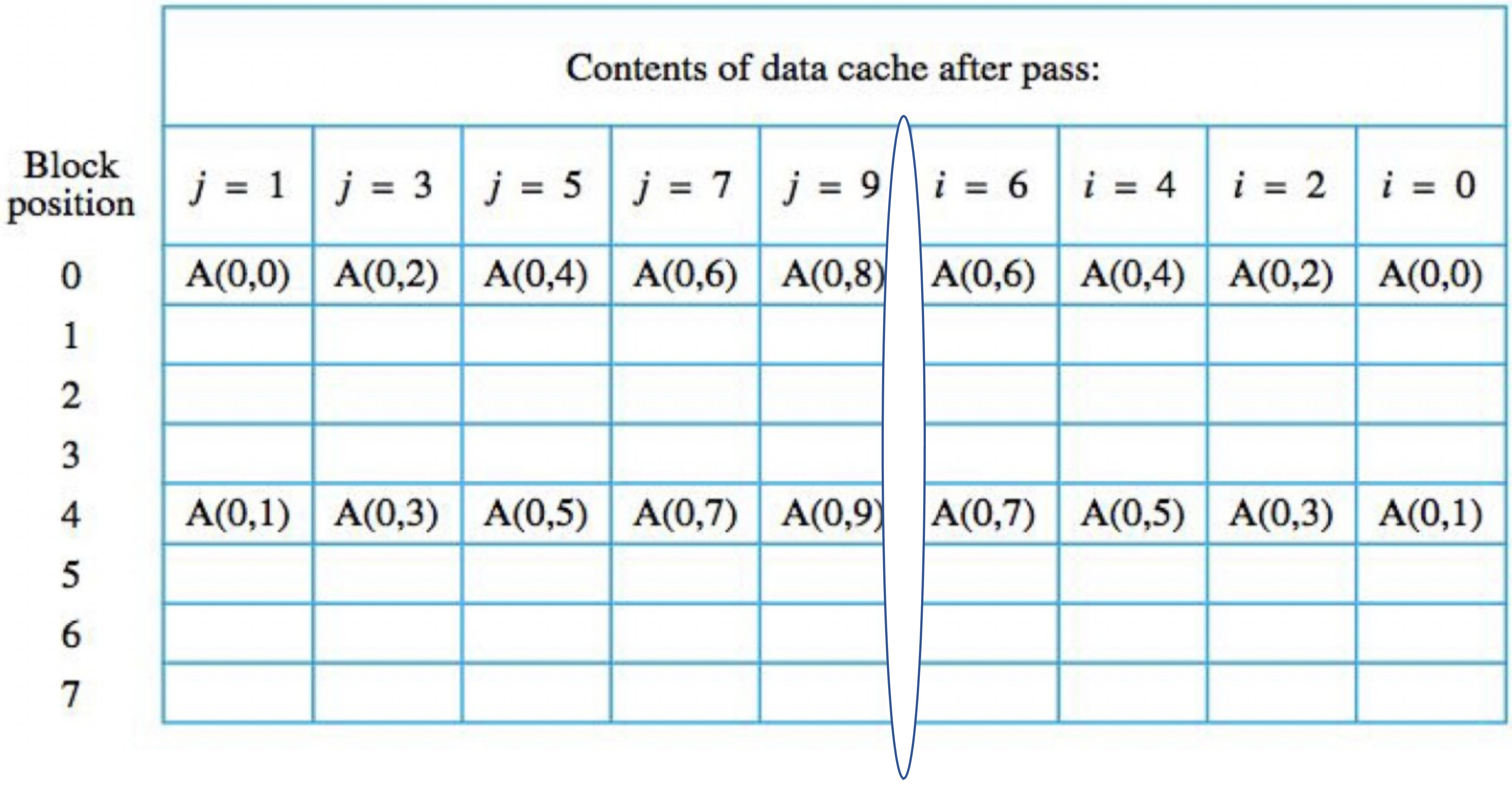
- Memory word size = 16 bits
- Word-addressable with 16-bit addresses
- Block size = one 16-bit word
- Cache size n = 8 blocks
- LRU replacement
- Consider direct mapped, associative and 4-way set-associative caches



	Memory address	Contents
(7A00)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	A(0,0)
(7A01)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1	A(1,0)
(7A02)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0	A(2,0)
(7A03)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1	A(3,0)
(7A04)	0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0	A(0,1)
	⋮	
(7A24)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0	A(0,9)
(7A25)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 1	A(1,9)
(7A26)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0	A(2,9)
(7A27)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 1	A(3,9)

Direct-mapped

only 2 hits!

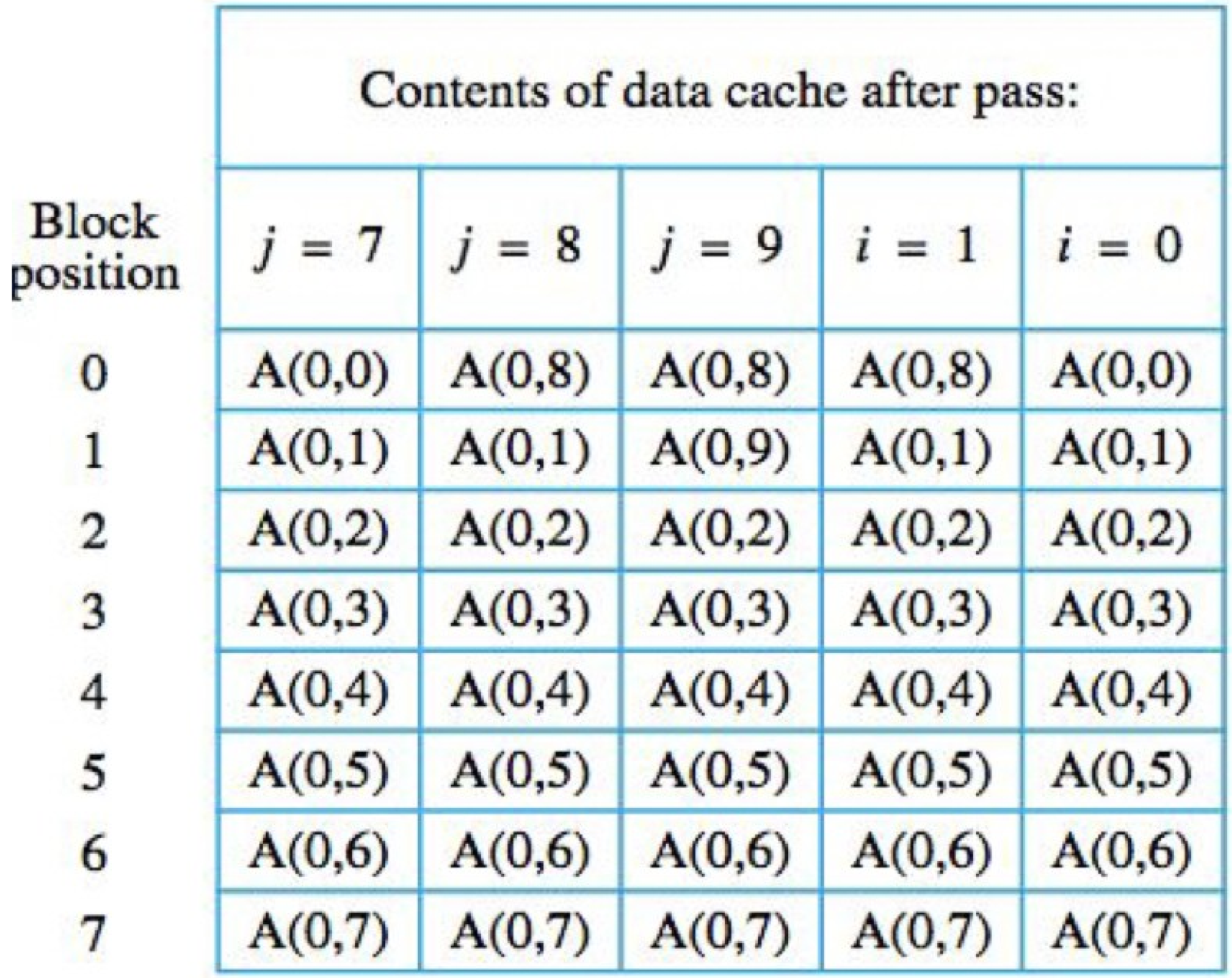


Block position	Contents of data cache after pass:								
	$j = 1$	$j = 3$	$j = 5$	$j = 7$	$j = 9$	$i = 6$	$i = 4$	$i = 2$	$i = 0$
0	A(0,0)	A(0,2)	A(0,4)	A(0,6)	A(0,8)	A(0,6)	A(0,4)	A(0,2)	A(0,0)
1									
2									
3									
4	A(0,1)	A(0,3)	A(0,5)	A(0,7)	A(0,9)	A(0,7)	A(0,5)	A(0,3)	A(0,1)
5									
6									
7									

Associative

only 2 misses !

reversing the order of the seconds loop helps hit the cache

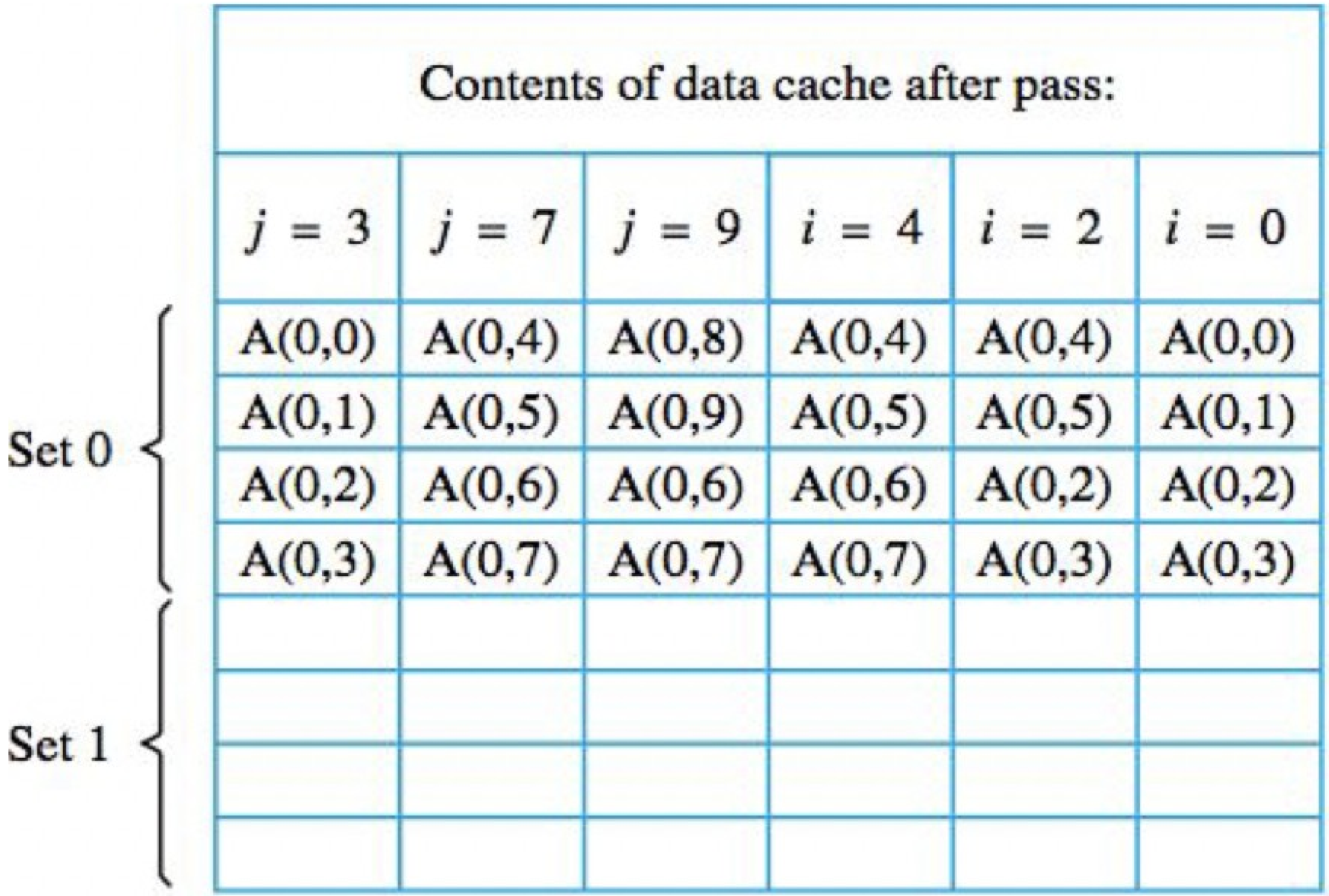


Block position	Contents of data cache after pass:				
	$j = 7$	$j = 8$	$j = 9$	$i = 1$	$i = 0$
0	A(0,0)	A(0,8)	A(0,8)	A(0,8)	A(0,0)
1	A(0,1)	A(0,1)	A(0,9)	A(0,1)	A(0,1)
2	A(0,2)	A(0,2)	A(0,2)	A(0,2)	A(0,2)
3	A(0,3)	A(0,3)	A(0,3)	A(0,3)	A(0,3)
4	A(0,4)	A(0,4)	A(0,4)	A(0,4)	A(0,4)
5	A(0,5)	A(0,5)	A(0,5)	A(0,5)	A(0,5)
6	A(0,6)	A(0,6)	A(0,6)	A(0,6)	A(0,6)
7	A(0,7)	A(0,7)	A(0,7)	A(0,7)	A(0,7)

Set-Associative

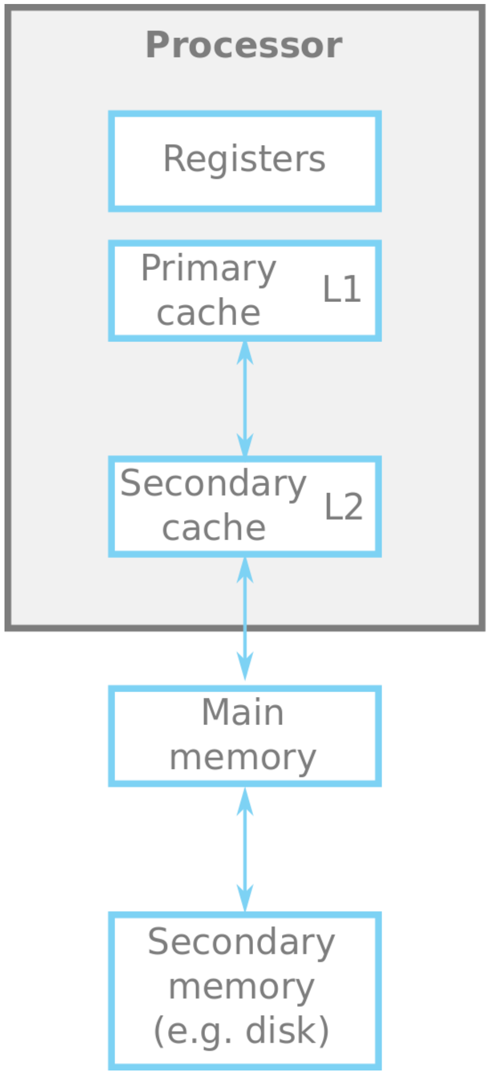
All desired blocks have even addresses – all map to Set 0

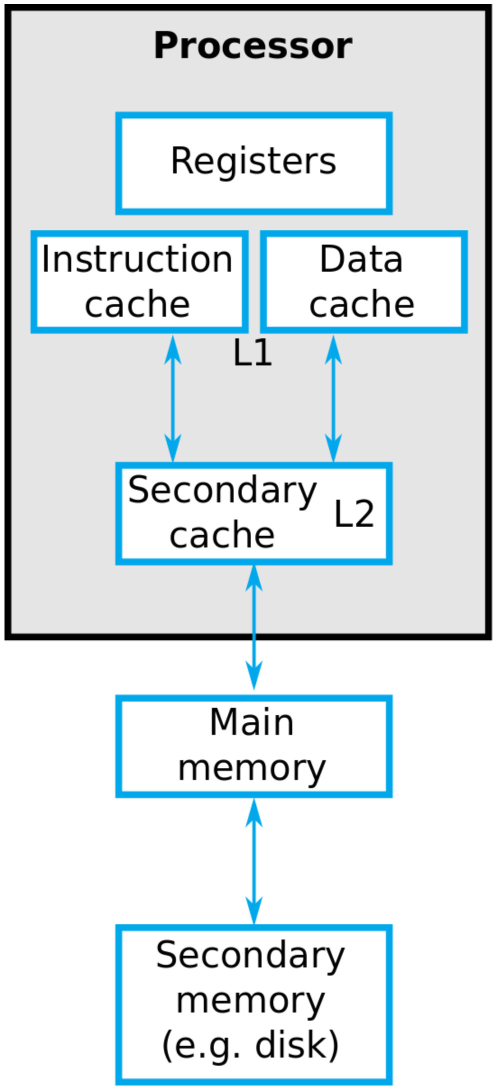
6 misses in the second loop



	Contents of data cache after pass:					
	$j = 3$	$j = 7$	$j = 9$	$i = 4$	$i = 2$	$i = 0$
Set 0	A(0,0)	A(0,4)	A(0,8)	A(0,4)	A(0,4)	A(0,0)
	A(0,1)	A(0,5)	A(0,9)	A(0,5)	A(0,5)	A(0,1)
	A(0,2)	A(0,6)	A(0,6)	A(0,6)	A(0,2)	A(0,2)
	A(0,3)	A(0,7)	A(0,7)	A(0,7)	A(0,3)	A(0,3)
Set 1						

Instruction & Data cache





Instructions and data are stored in same memory; however:

- Different access patterns
  - repetitions (e.g. loops)
  - linear sequences of instructions
- Instructions are read-only (mostly)
- L1 separated into data cache and instruction cache

Secondary storage technology

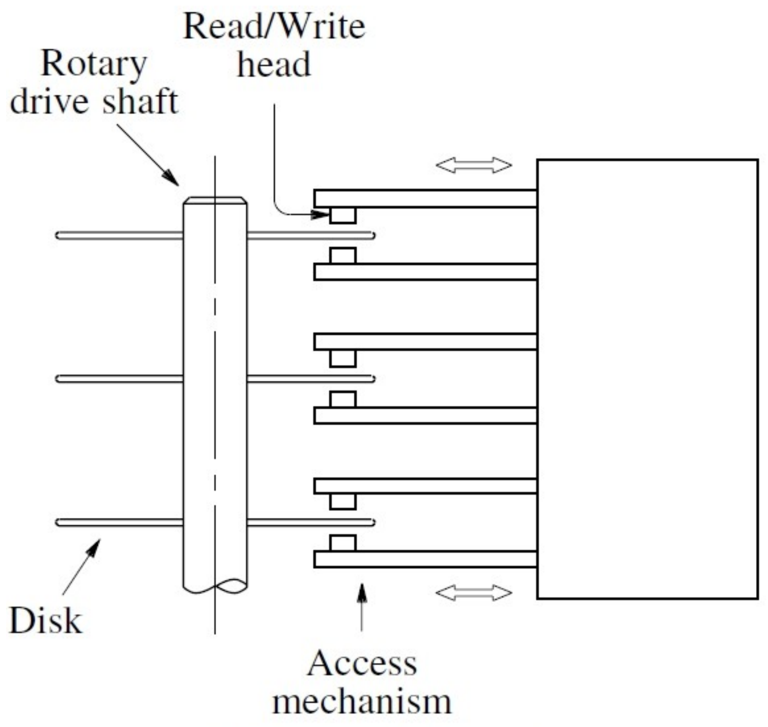
Textbook 8.10

Secondary storage

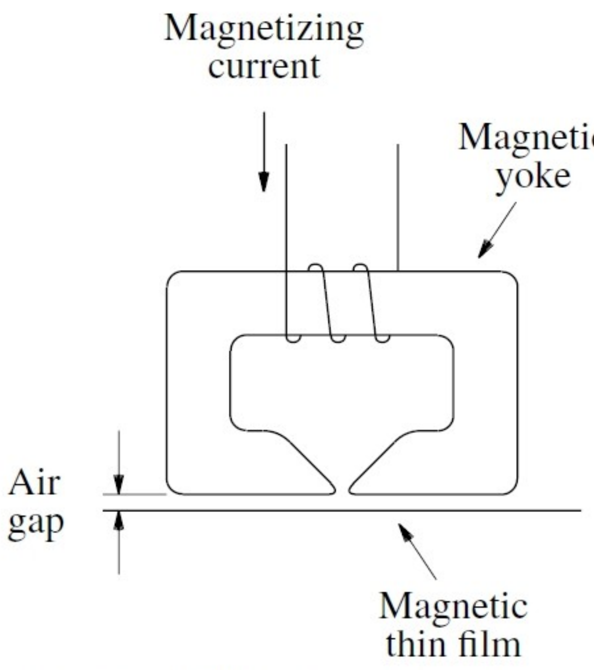
- Non-volatile long term storage
- Bottom of memory hierarchy → slow but large capacity
- Managed by the operating system
- Flash memory (SSD) is the technology used in phones, tablets, and some laptops
- Magnetic disks (hard drives) have lower cost / bit

Magnetic Hard Disks

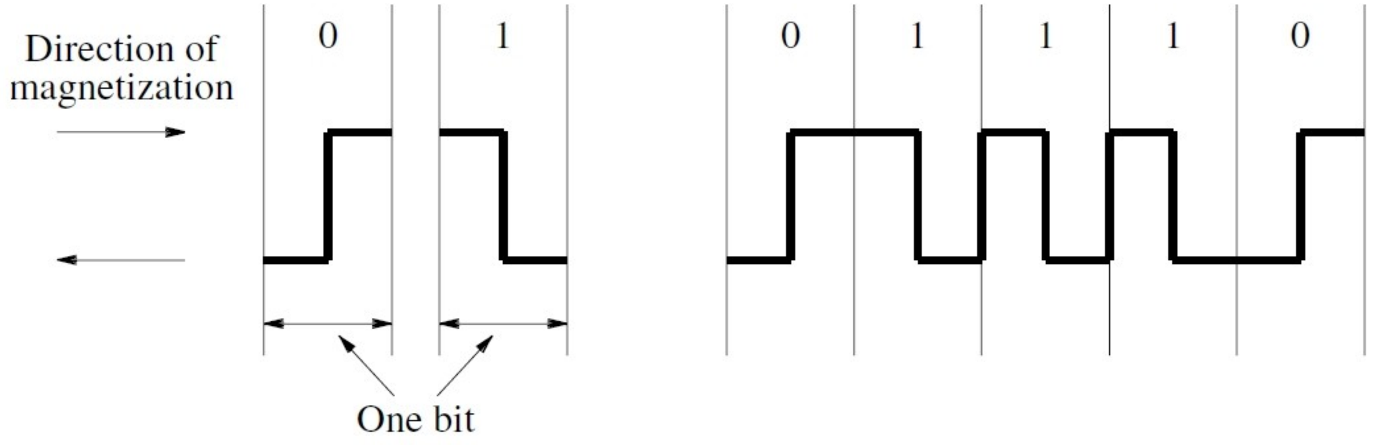
- One or more platters on a common spindle
- Platters are covered with thin magnetic film
- Platters rotate on spindle at constant rate
- Read/write heads in close proximity to surface can access data arranged in concentric tracks
- Magnetic yoke and magnetizing coil can change or sense polarity of areas on surface



(a) Mechanical structure



(b) Read/Write head detail

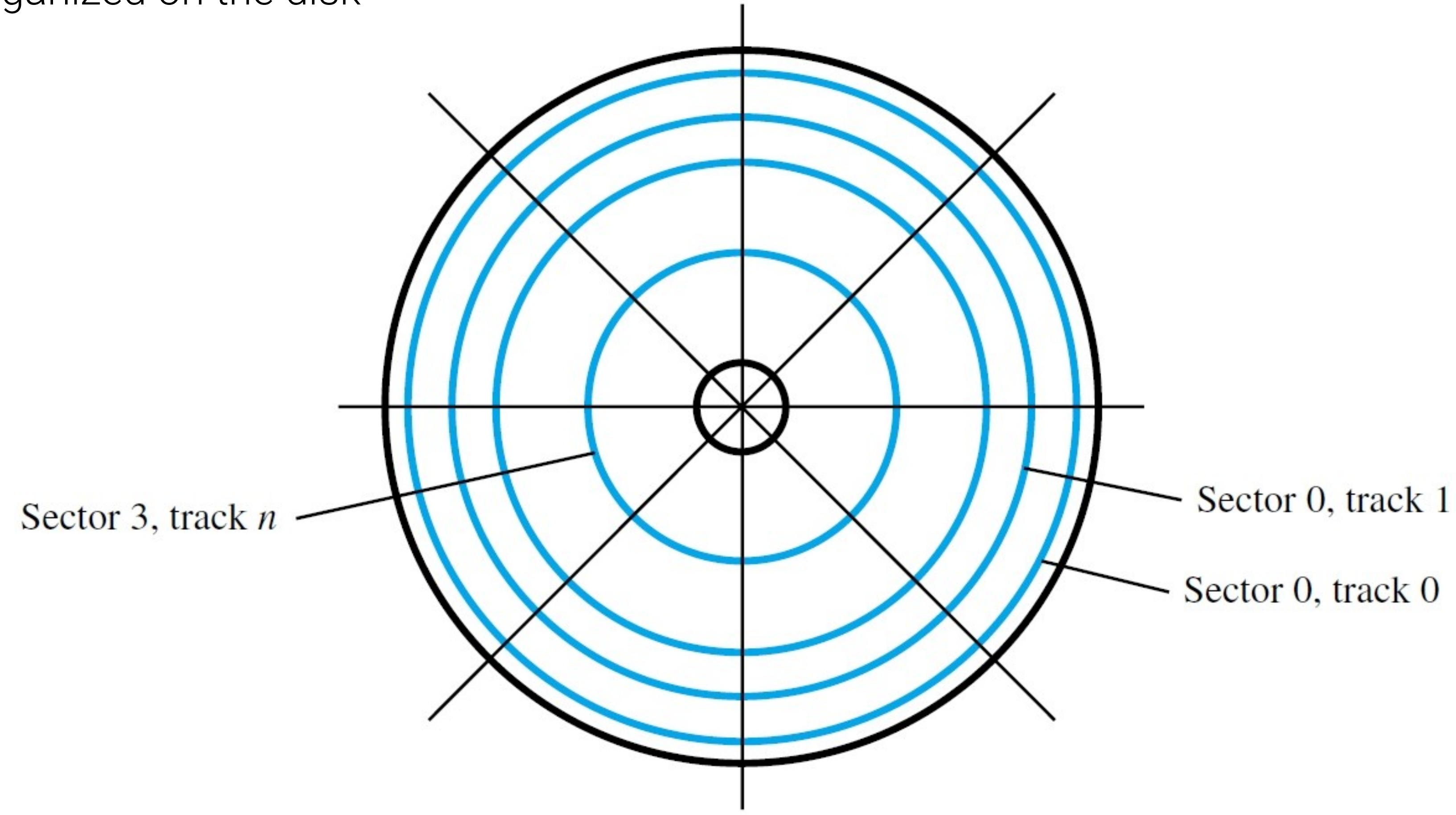


(c) Bit representation by phase encoding

- A *cylinder* is a logical set of tracks on a stack of disks that can be accessed without moving the read/write heads

- Formatting information includes track/sector markers and *error-correcting code* (ECC) information

- *Filesystem*: data structures that the O/S uses to keep track of files organized on the disk



Access time

- *Seek time* = time required to move the read/write head to the proper track. Depends on the initial position of the head.
    - Average values are 5 to 8 ms
- *Latency* (rotational delay) = time to read addressed sector after the head is positioned over the correct track.
    - On average the time for ½ a rotation of the disk

Access time = seek time + latency

- Flash access time is typically 35 to 100 μs (100x faster)

Virtual Memory

Textbook 8.8, 8.9

Virtual Memory

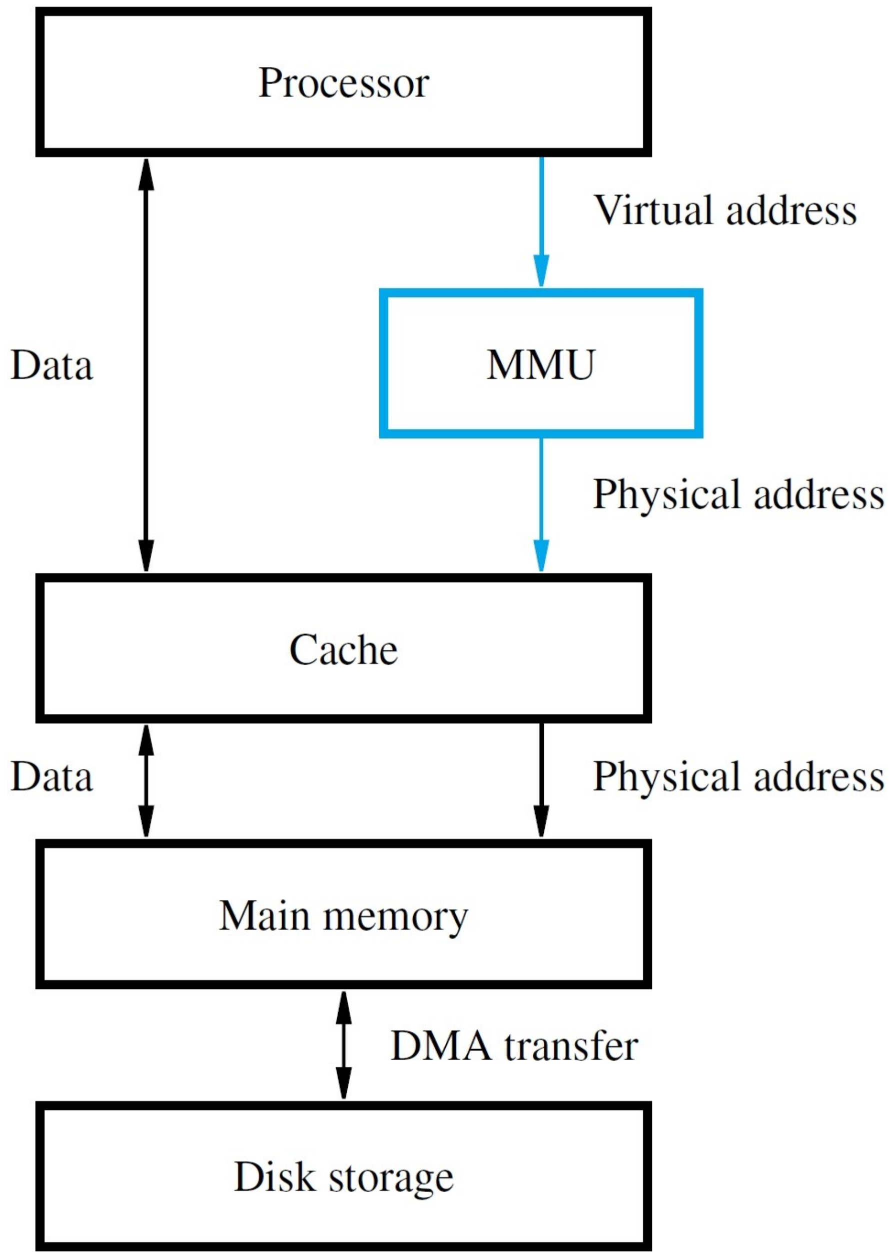
- Physical mem. Capacity ≤ address space size
- A large program or many active programs may not be entirely resident in the main memory
- Use secondary storage (flash or magnetic disk) to hold portions exceeding memory capacity (“swap file” or “page file”) – makes the RAM appear to be very large
- Virtual memory is the lowest tier of memory hierarchy
  - Magnetic disk (5 ms) is 5 orders of magnitude slower than SDRAM (15 ns)
  - It is important to manage virtual memory carefully to reduce number of disk accesses - managed in software (O/S)

Virtual Memory

- Programs written assuming full address space
- Processor issues *virtual address* *(logical address)*
- Must be translated into *physical address*
- Proceed with normal memory operation
  when addressed contents are in the memory
- When no current physical address exists, perform actions
  to place contents in memory
- The mapping is fully associative (reduce miss rate !)

Memory Management Unit

- Implementation of virtual memory relies on a *memory management unit (MMU)*
- Maintains virtual → physical address mapping to perform the necessary translation
- When no current physical address exists, MMU invokes operating system services
- Causes transfer of desired contents from disk to the main memory using DMA scheme
- MMU mapping information is also updated

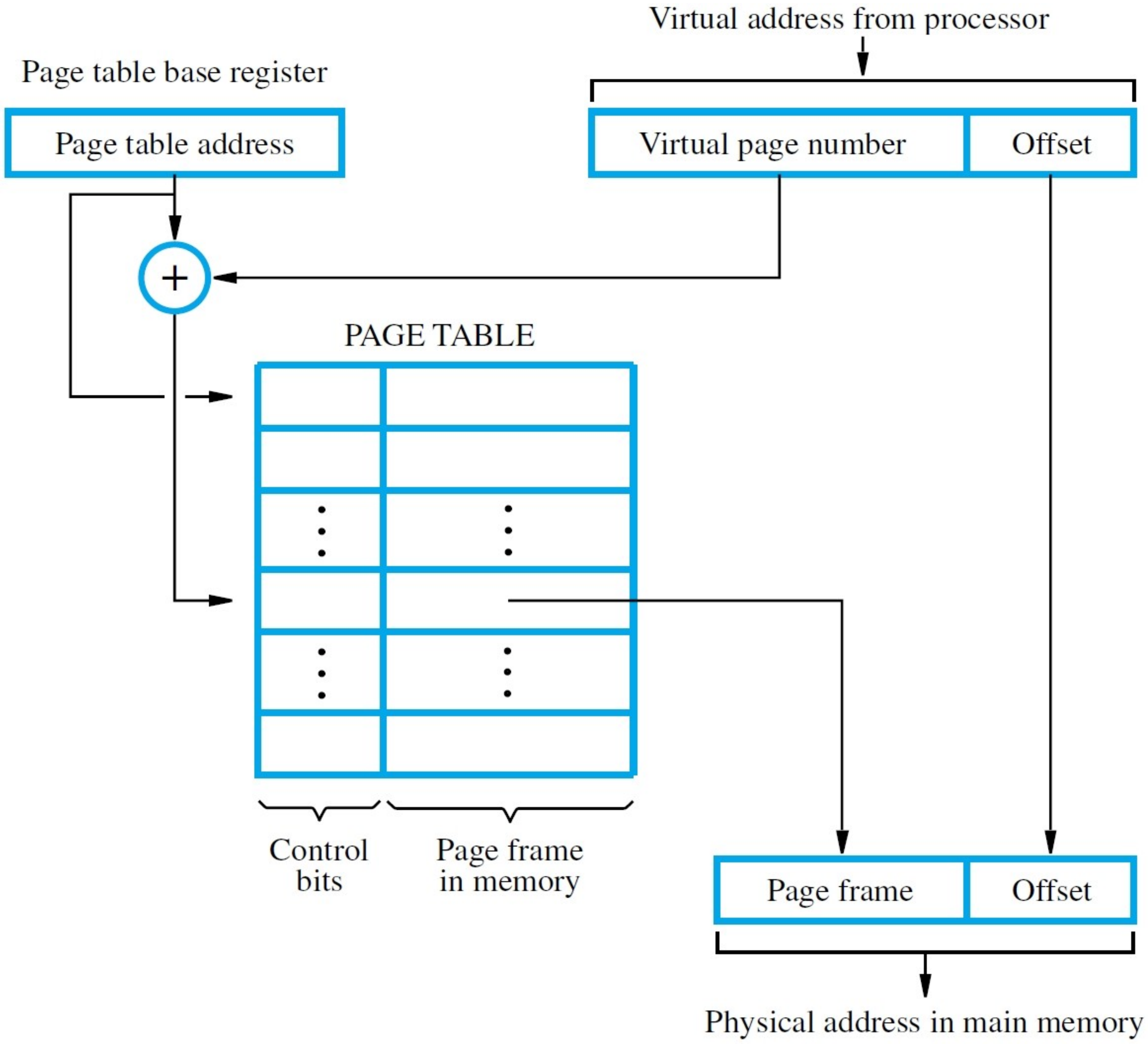

Processor
Virtual address
Data
MMU
Physical address
Cache
Data
Physical address
Main memory
DMA transfer
Disk storage

Address Translation

- Use fixed-length unit of *pages* (2K-16K bytes)
- Larger size than cache blocks
  - Disks have high access times, but bandwidths of several MB / s
- For translation, divide address bits into 2 fields
  - Lower bits give *offset* of word within page
  - Upper bits give *virtual page number (VPN)*
- Translation preserves offset bits, but causes VPN bits to be replaced with *page frame* bits
- *Page table* (stored in the main memory) provides information to perform translation

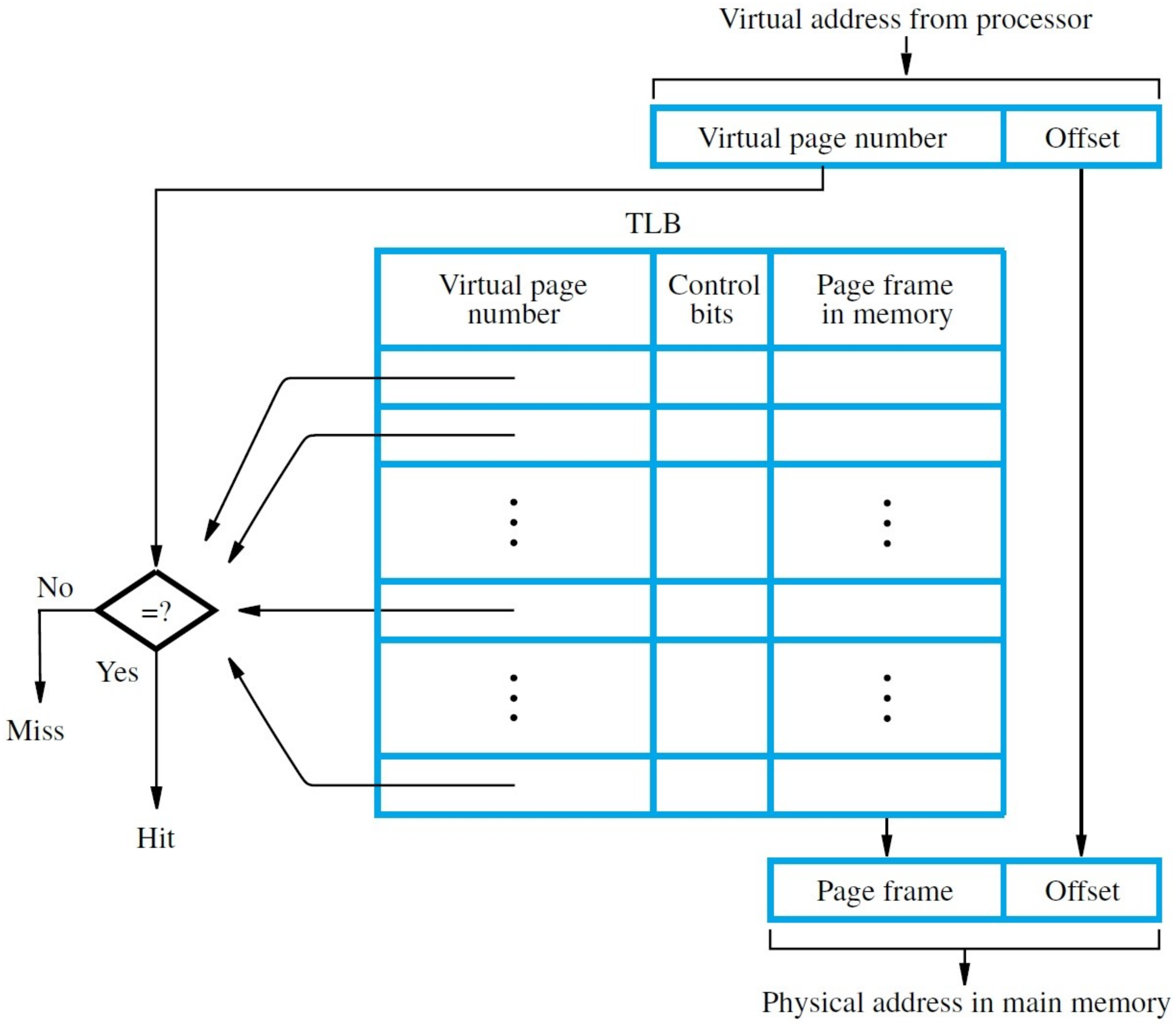
Page Table

- MMU must know location of page table
- *Page table base register* has starting address
- Adding VPN to base register contents gives location of corresponding entry about page
- If page is in memory, table gives frame bits
- Otherwise, table may indicate disk location
- Control bits for each entry include a valid bit and modified bit indicating needed write-back
- Also have bits for page read/write permissions


Virtual address from processor
Page table base register
Page table address
Virtual page number
Offset
+
PAGE TABLE
Control bits
Page frame in memory
Page frame
Offset
Physical address in main memory

Translation Lookaside Buffer

- MMU must perform lookup in page table for translation of every virtual address
- For large physical memory, MMU cannot hold entire page table with all of its information
- *Translation lookaside buffer (TLB)* in the MMU holds recently-accessed entries of page table
- Associative searches are performed on the TLB with virtual addresses to find matching entries
- If miss in TLB, access full table and update TLB


Virtual address from processor
Virtual page number
Offset
TLB
Virtual page number
Control bits
Page frame in memory
No
=?
Yes
Miss
Hit
Page frame
Offset
Physical address in main memory

Page Faults

- A *page fault* occurs when a virtual address has no corresponding physical address
- MMU raises an interrupt for operating system to place the containing page in the memory
- Operating system selects location using LRU, performing write-back if needed for old page
- Delay may be long, involving disk accesses, hence another program is selected to execute
- Suspended program restarts later when ready

What’s next

- In the next chapter we will look at how the hardware of the processor is implemented and how it executes instructions