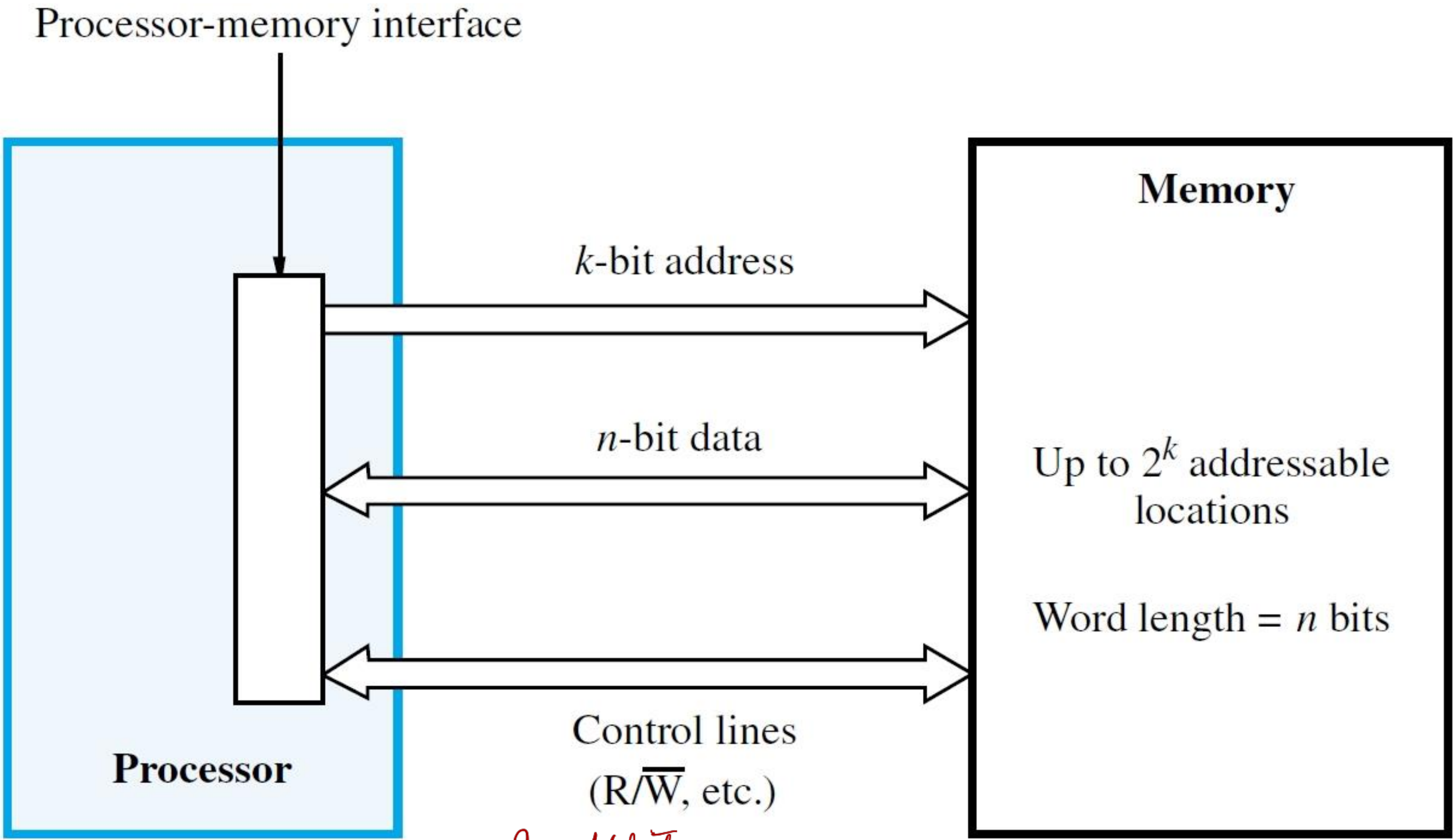
Chapter 8
Memory System

Chapter Outline

- Basic memory circuits & memory organization
- Memory technology
- Memory hierarchy
- Virtual memory

Basic Concepts

- Access provided by processor-memory interface
- *Address and data lines*, and also *control lines* for command (Read/Write), timing, data size
- *Memory access time* is time from initiation to completion of a word or byte transfer
- *Memory cycle time* is minimum time delay between initiation of successive transfers
- *Random-access memory (RAM)* means that access time is same, independent of location



Processor-memory interface

Processor

k-bit address

n-bit data

Control lines

($R/\overline{W}$, etc.)

Memory

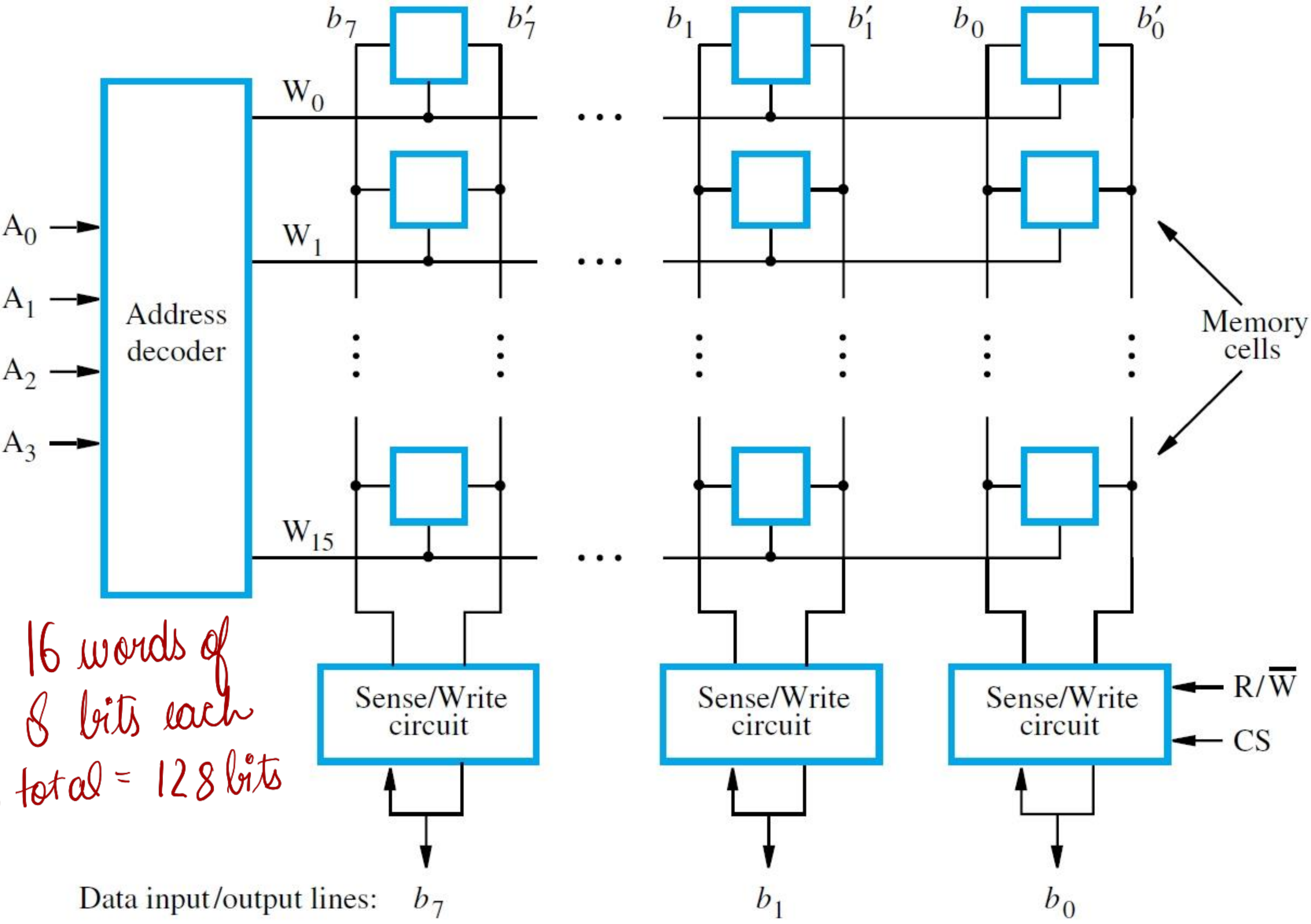
Up to 2^k addressable locations

Word length = *n* bits

Read/ Write

Semiconductor RAM Memories

- Memory chips have a common organization
- *Cells* holding single bits arranged in an array
- *Words* are rows; cells connected to *word lines* (cells per row ≠ bits per processor word)
- Cells in columns connect to *bit lines*
- Sense/Write circuits are interfaces between internal bit lines and data I/O pins of chip
- Typical control pin connections include Read/Write command and chip select (CS)



16 words of
8 bits each
∴ total = 128 bits

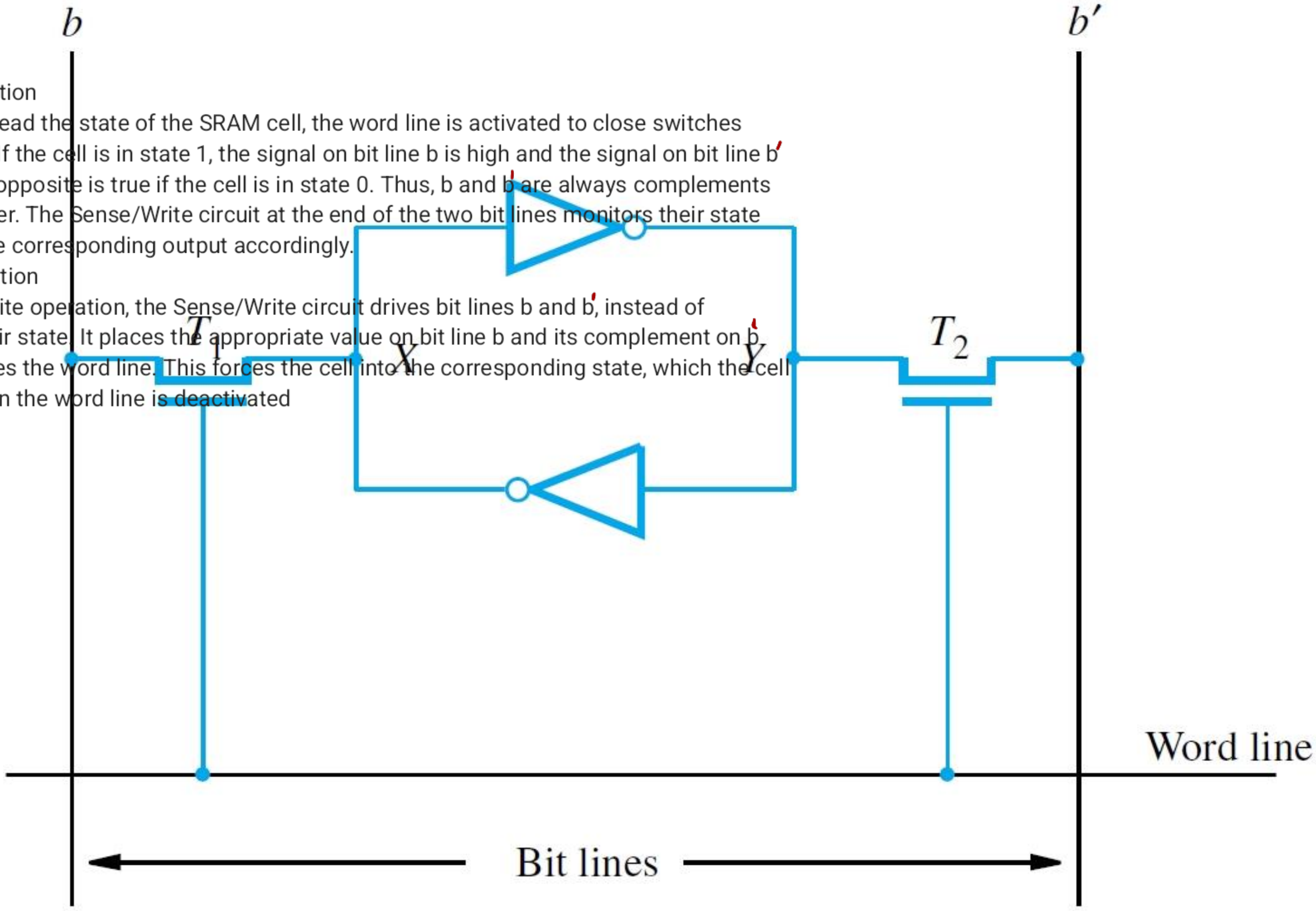
Internal Organization and Operation

- Example of 16-word × 8-bit memory chip has *decoder* to select word line from 4-bit address
- Two complementary bit lines for each data bit
- External source provides stable address bits, and asserts chip-select input with command
- For Read operation, Sense/Write circuits transfer data from selected row to I/O pins
- For Write operation, Sense/Write circuits transfer data from I/O pins to selected cells

Static RAMs and CMOS Cell

- Static memories need power to retain state; usually have short access time (few nano secs.)
- A *static RAM cell* in a chip consists of two cross-connected inverters to form a *latch*
- Chip implementation typically uses *CMOS* cell whose advantage is low power consumption
- Two transistors controlled by word line act as switches between the cell and the bit lines
- To write, bit lines driven with desired data

Word line = Ground , transistor = OFF, latch retains its state



Read Operation

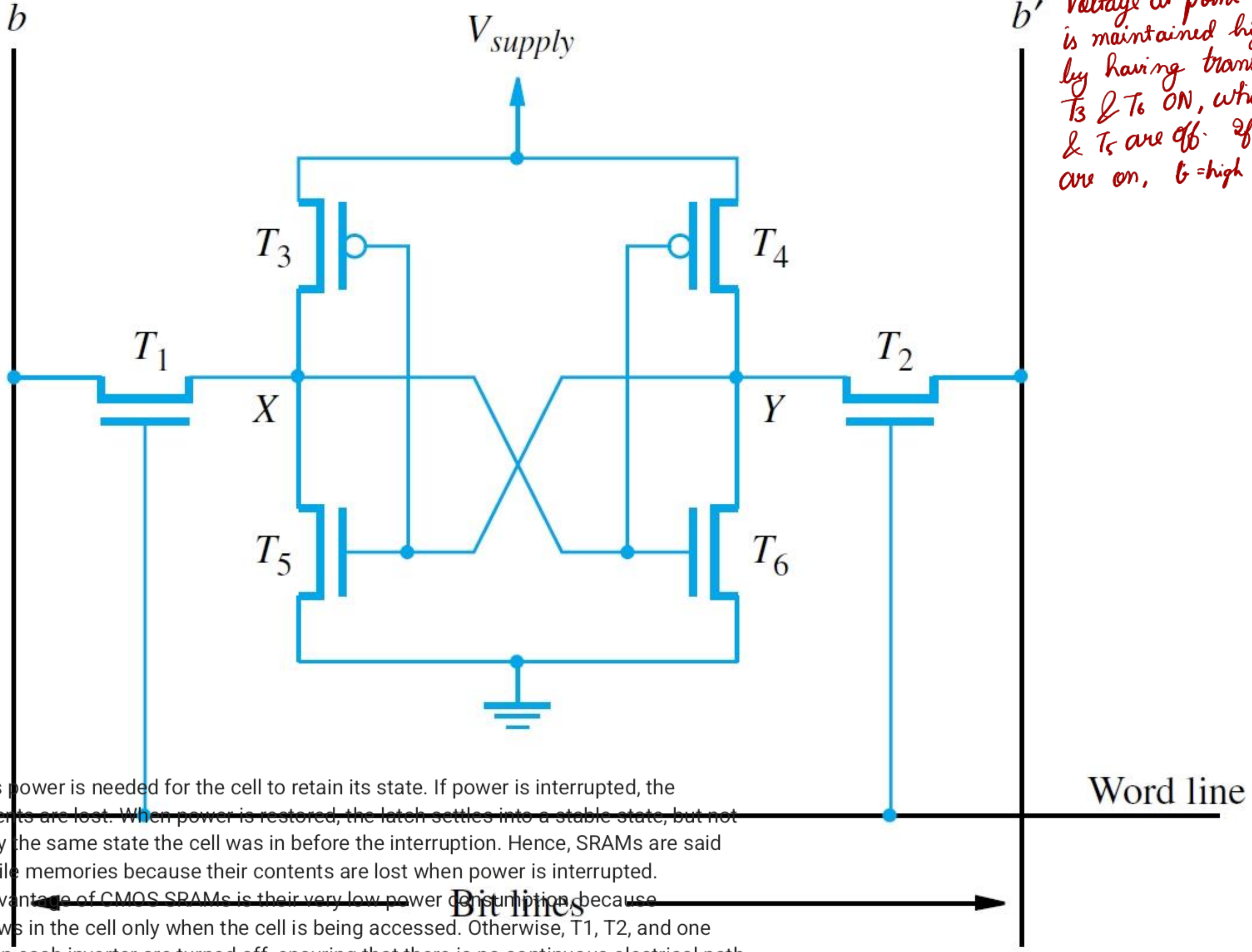
In order to read the state of the SRAM cell, the word line is activated to close switches T1 and T2. If the cell is in state 1, the signal on bit line b is high and the signal on bit line b′ is low. The opposite is true if the cell is in state 0. Thus, b and b′ are always complements of each other. The Sense/Write circuit at the end of the two bit lines monitors their state and sets the corresponding output accordingly.

Write Operation

During a Write operation, the Sense/Write circuit drives bit lines b and b′, instead of sensing their state. It places the appropriate value on bit line b and its complement on b′ and activates the word line. This forces the cell into the corresponding state, which the cell retains when the word line is deactivated

(T3, T5) ~ (T4, T6) inverters

In state 1, the voltage at point X is maintained high by having transistors T3 & T6 ON, while T4 & T5 are off. If T1 T2 are on, b = high b' = low

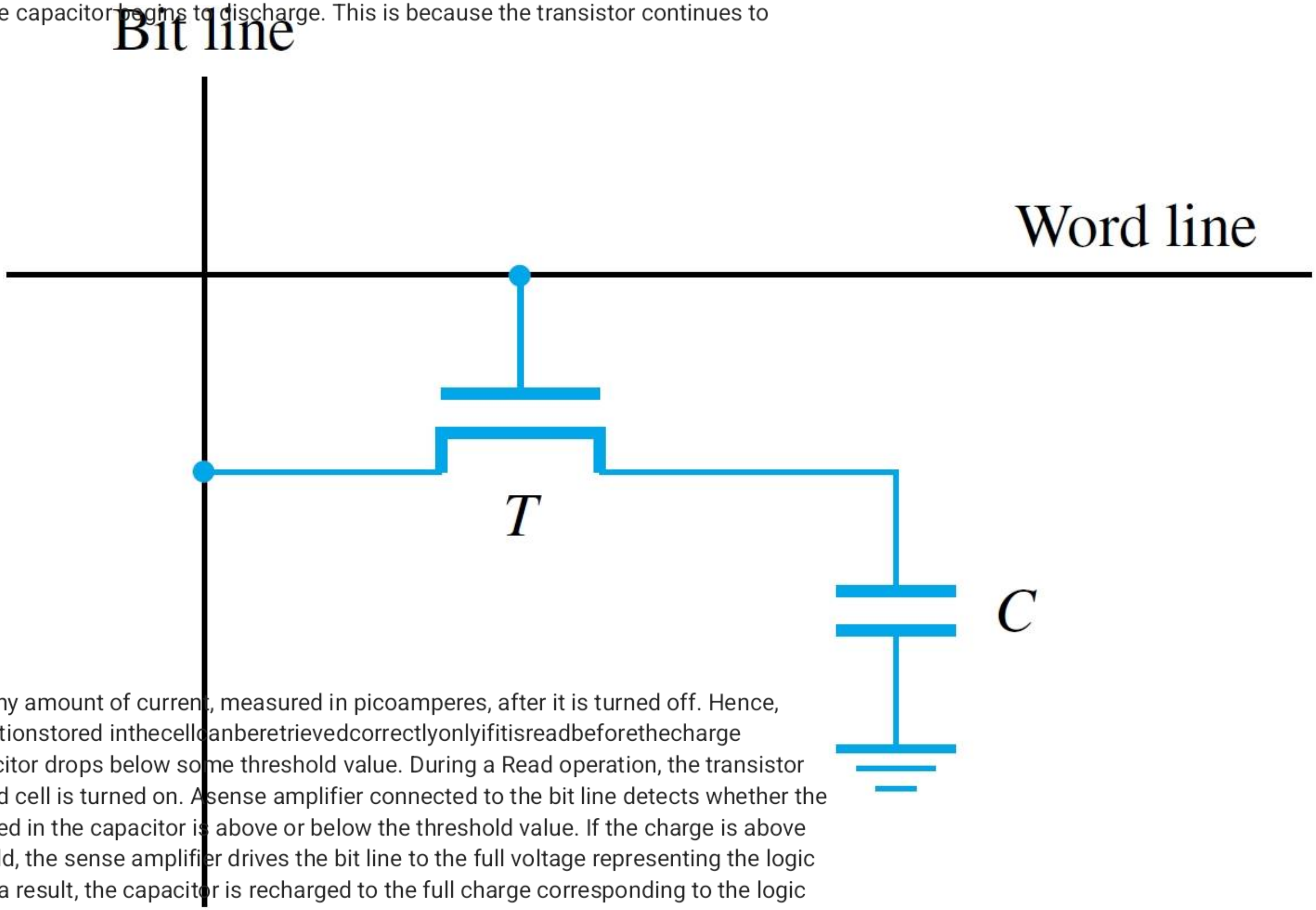


Continuous power is needed for the cell to retain its state. If power is interrupted, the cell's contents are lost. When power is restored, the latch settles into a stable state, but not necessarily the same state the cell was in before the interruption. Hence, SRAMs are said to be volatile memories because their contents are lost when power is interrupted. Amajor advantage of CMOS SRAMs is their very low power consumption, because current flows in the cell only when the cell is being accessed. Otherwise, T1, T2, and one transistor in each inverter are turned off, ensuring that there is no continuous electrical path between Vsupply and ground.

Dynamic RAMs

- Static RAMs have short access times, but need several transistors per cell, so density is lower
- Dynamic RAMs are simpler for higher density and lower cost, but access times are longer
- Density/cost advantages outweigh slowness
- Dynamic RAMs are widely used in computers
- Cell consists of a transistor and a capacitor
- State is presence/absence of capacitor charge
- Charge leaks away and must be *refreshed*

To store information in this cell, transistor T is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor.



After the transistor is turned off, the charge remains stored in the capacitor, but not for long. The capacitor begins to discharge. This is because the transistor continues to conduct a tiny amount of current, measured in picoamperes, after it is turned off. Hence, the informationstored inthecellcanberetrievedcorrectlyonlyifitisreadbeforethecharge in the capacitor drops below some threshold value. During a Read operation, the transistor in a selected cell is turned on. A sense amplifier connected to the bit line detects whether the charge stored in the capacitor is above or below the threshold value. If the charge is above the threshold, the sense amplifier drives the bit line to the full voltage representing the logic value 1. As a result, the capacitor is recharged to the full charge corresponding to the logic value 1. If the sense amplifier detects that the charge in the capacitor is below the threshold value, it pulls the bit line to ground level to discharge the capacitor fully. Thus, reading the contents of a cell automatically refreshes its contents. Since the word line is common to all cells in a row, all cells in a selected row are read and refreshed at the same time.

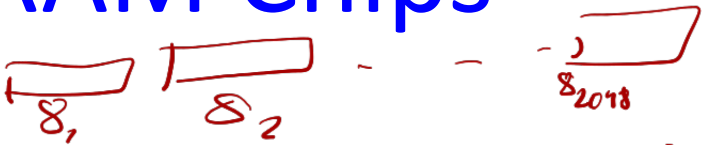
Dynamic RAM Chip Operation

- Reflects general principles of chip operation
- For Read, charge from cells in selected row is checked by *sense amplifiers* on bit lines
- 1 or 0 if charge is above or below threshold
- Action of sensing the bit lines also causes refresh of charge in all cells of selected row
- For Write, access the row and drive bit lines to alter amount of charge in subset of cells
- Refresh rows periodically to maintain charge

1/0

More on Dynamic RAM Chips

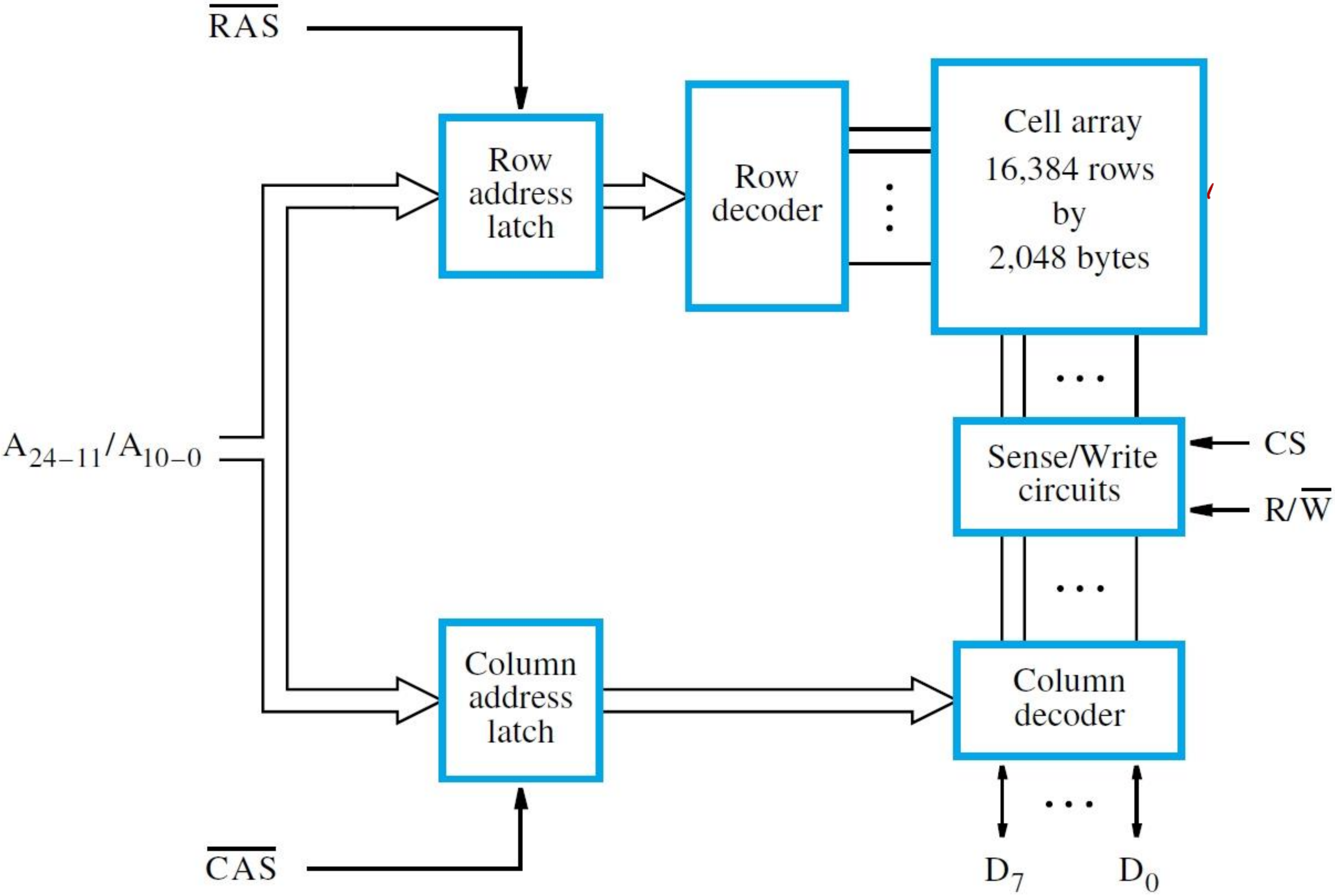
no. of words × length of 1 word →



- Consider 32M × 8 chip with 16K × 16K array
- 16,384 cells per row organized as 2048 bytes

2048 ↓ 2^{11}

- 14 bits to select row, 11 bits for byte in row
- Use *multiplexing* of row/column on same pins
- Row/column *address latches* capture bits
- Row/column *address strobe signals* for timing (asserted low with row/column address bits)
- Asynchronous DRAMs: delay-based access, external controller refreshes rows periodically



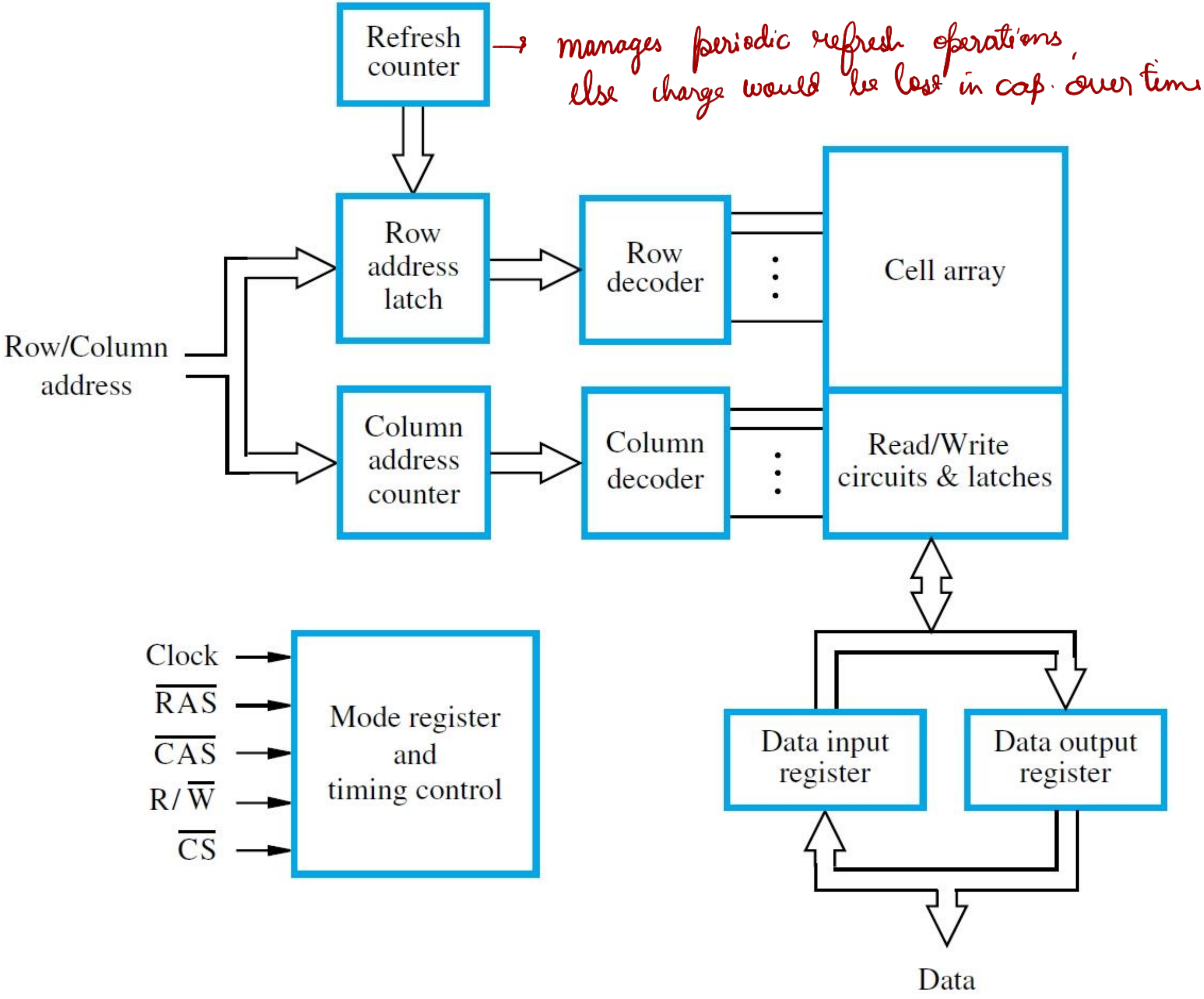
$\overline{RAS}$
Row address latch
Row decoder
Cell array 16,384 rows by 2,048 bytes
A_{24-11}/A_{10-0}
Sense/Write circuits
CS
$R/\overline{W}$
Column address latch
Column decoder
$\overline{CAS}$
D_7
D_0

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

- In early 1990s, DRAM technology enhanced by including clock signal with other chip pins
- More circuitry also added for enhancements
- These chips are called synchronous DRAMs
- Sense amplifiers still have latching capability
- Additional benefits from internal buffering and availability of synchronizing clock signal
- Internal row counter enables built-in refresh instead of relying on external controller



Refresh counter

→ manages periodic refresh operations, else charge would be lost in cap. over time

Row address latch

Row decoder

Cell array

Row/Column address

Column address counter

Column decoder

Read/Write circuits & latches

Clock

$\overline{\text{RAS}}$

$\overline{\text{CAS}}$

R/ $\overline{\text{W}}$

$\overline{\text{CS}}$

Mode register and timing control

Data input register

Data output register

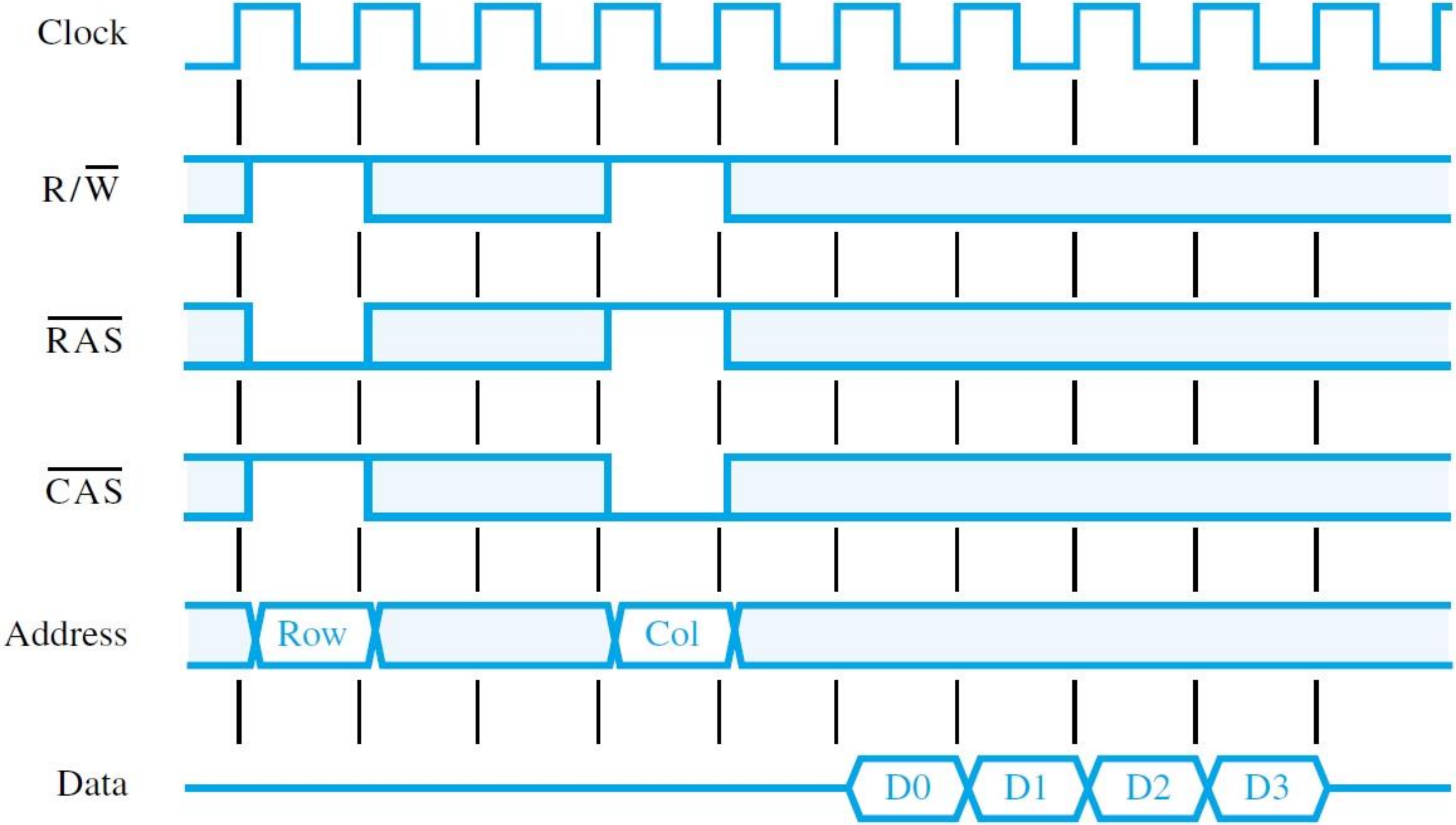
Data

SDRAM Features

- Synchronous DRAM (SDRAM) chips include *data registers* as well as address latches
- New access operation can be initiated while data are transferred to or from these registers
- Also have more sophisticated control circuitry
- SDRAM chips require power-up configuration
- Memory controller initializes *mode* register
- Used to specify *burst length* for block transfers and also to set delays for control of timing

Efficient Block Transfers

- Asynchronous DRAM incurs longer delay from CAS assertion for *each* column address
- 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
- Consider example with burst length of 4, RAS delay of 3 cycles, CAS delay of 2 cycles


Clock
R/$\overline{W}$
$\overline{RAS}$
$\overline{CAS}$
Address
Row
Col
Data
D0
D1
D2
D3

Double-Data-Rate (DDR) SDRAM

- Early SDRAMs transferred on rising clock edge
- Later enhanced to use rising and falling edges
- *Doubles* effective rate *after* RAS/CAS assertion
- Requires more complex clock/control circuitry
- Internal array access not significantly faster
- How can transfer rate be effectively doubled?
- *Interleave* consecutive data across two arrays
- Switch between arrays for each clock edge

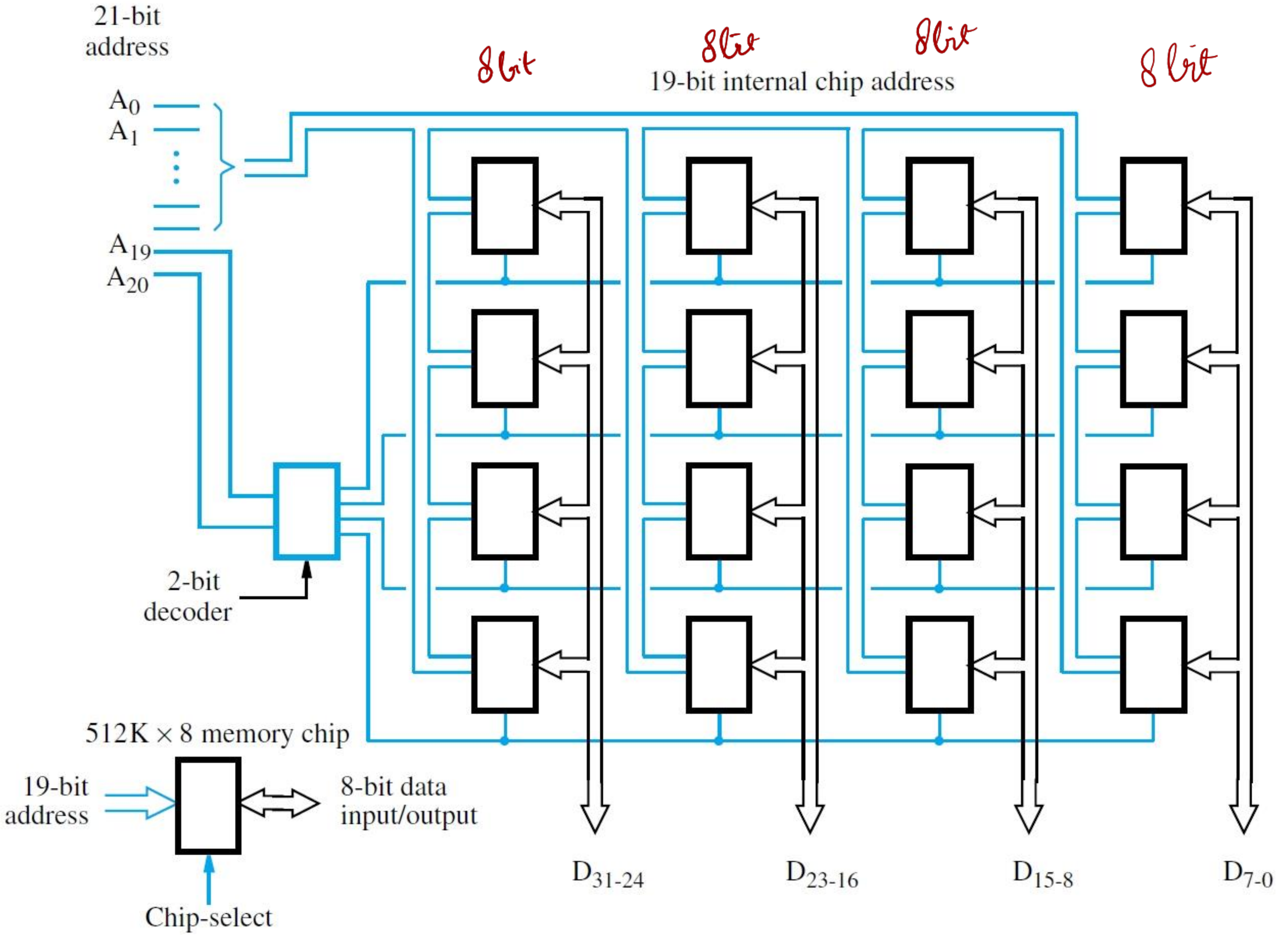
Structure of Larger Memories

- Internal chip organization has been discussed
- Larger memories combine multiple chips
- How are these chips connected together?
  - Consider an example based on static memory
  - Memory size is 2M words, each 32 bits in size
  - Implement with 512K × 8 static memory chips
  - 4 chips for 32 bits, 4 groups of 4 chips for 2M

$512K = 2^9 \times 2^{10} =$ 19 bits (internally)

+

2 bits (externally)


21-bit
address
8bit
8bit
8bit
8 bit
19-bit internal chip address
A0
A1
A19
A20
2-bit
decoder
512K × 8 memory chip
19-bit
address
8-bit data
input/output
Chip-select
D31-24
D23-16
D15-8
D7-0

Address Decoder and Tri-state Pins

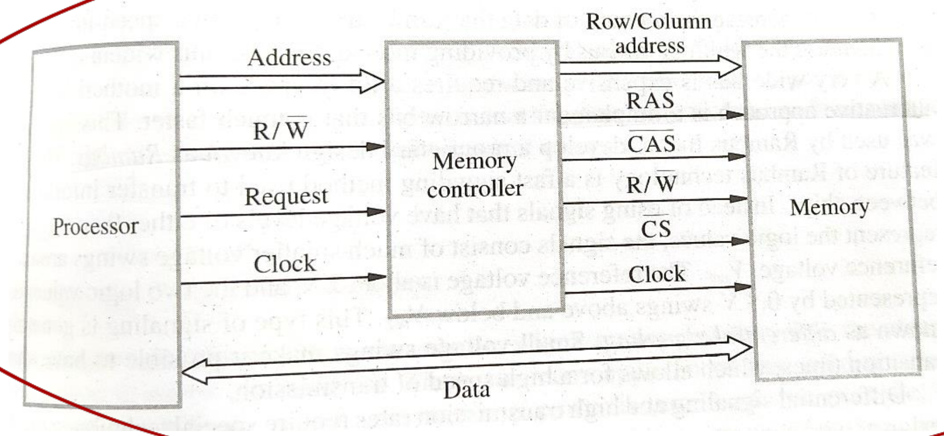
- 2M word-addressable memory needs 21 bits
- Each chip has only 19 address bits ($2^{19} = 512$K)
- Address bits A_{20} and A_{19} select one of 4 groups
- Outputs of 2-bit *decoder* drive chip-select pins
- Only the selected chips respond to request

- Shared data connections need *tri-state* circuits
- When a chip is not selected (i.e., CS input is 0), its I/O pins are *electrically disconnected*

Expandable Memory Systems

- DRAM chip capacity has grown over time with 2G bits/chip available now, and more in future
- Individual DRAM chips grouped into a module with aggregate capacity of 4 Gbytes or more
- Module socket interface is standardized – SIMM and DIMM
- Enables simple upgrades to increase capacity by replacing smaller modules with larger ones
- Printed-circuit board can have many sockets with common lines for address and data

Memory Controller



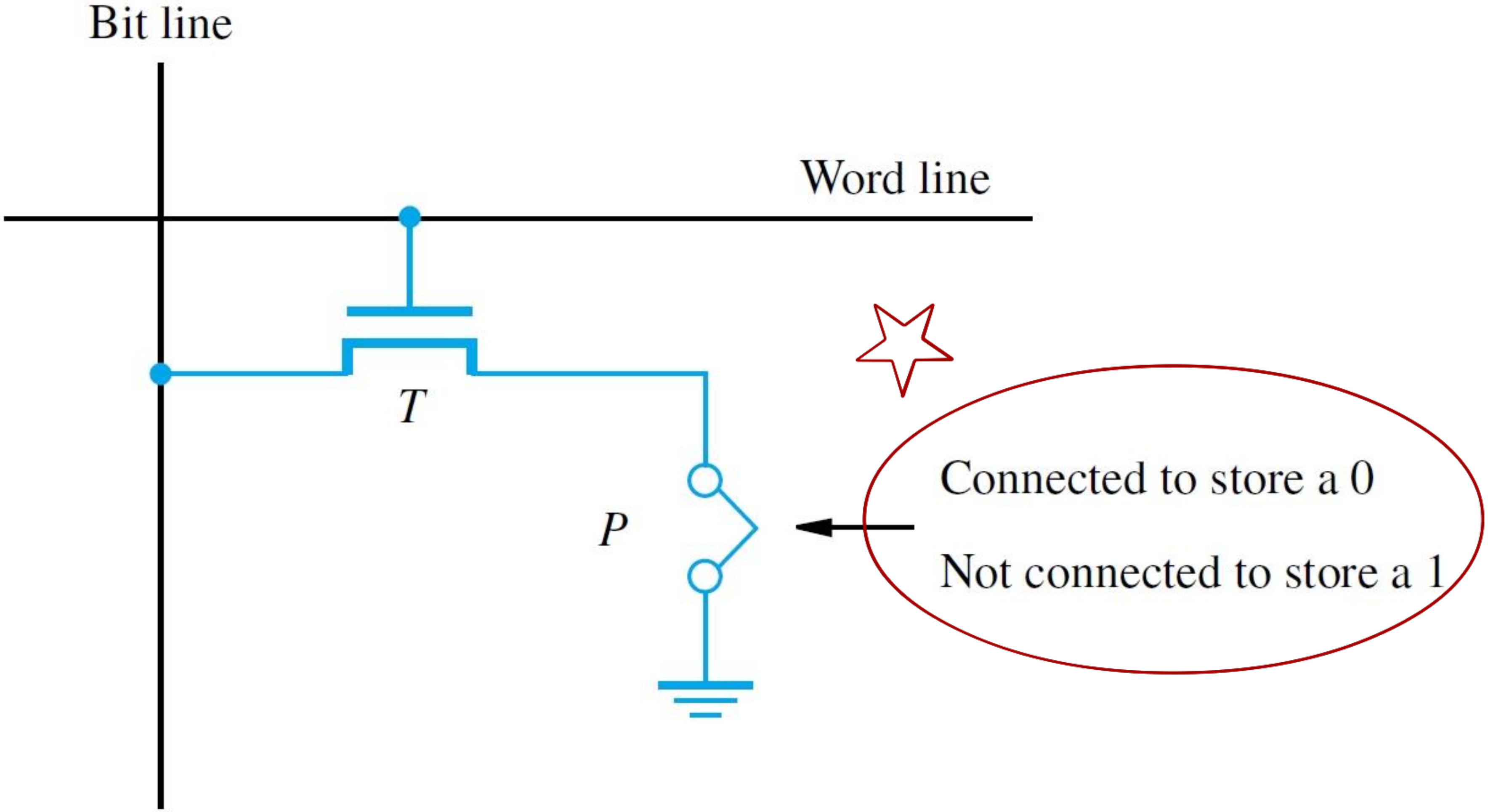
- Processor issues all address bits together, but DRAM chips need row/column multiplexing
- A *memory controller* handles this task and also asserts control signals for proper timing
- A large main memory is implemented using multiple DRAM modules sharing data lines
- Controller decodes high-order address bits to assert chip-select signal of only one module
- Other modules turn off their tri-state outputs

Read-only Memories

- Static and dynamic RAM chips are *volatile*, (information retained only when power is on)
- Some applications require information to be retained in memory chips when power is off
- Example: computers without disk drives
- Read-only memory (ROM) chips provide the nonvolatile storage for such applications
- Special writing process sets memory contents
- Read operation is similar to volatile memories

Basic ROM Cell

- A *read-only memory (ROM)* has its contents written only *once,* at the time of manufacture
- The basic ROM cell in such a memory contains a single transistor switch for the bit line
- The other end of the bit line is connected to the power supply through a resistor
- If the transistor is connected to ground, bit line voltage is near zero, so cell stores a 0
- Otherwise, bit line voltage is high for a 1



Connected to store a 0

Not connected to store a 1

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

Direct Memory Access

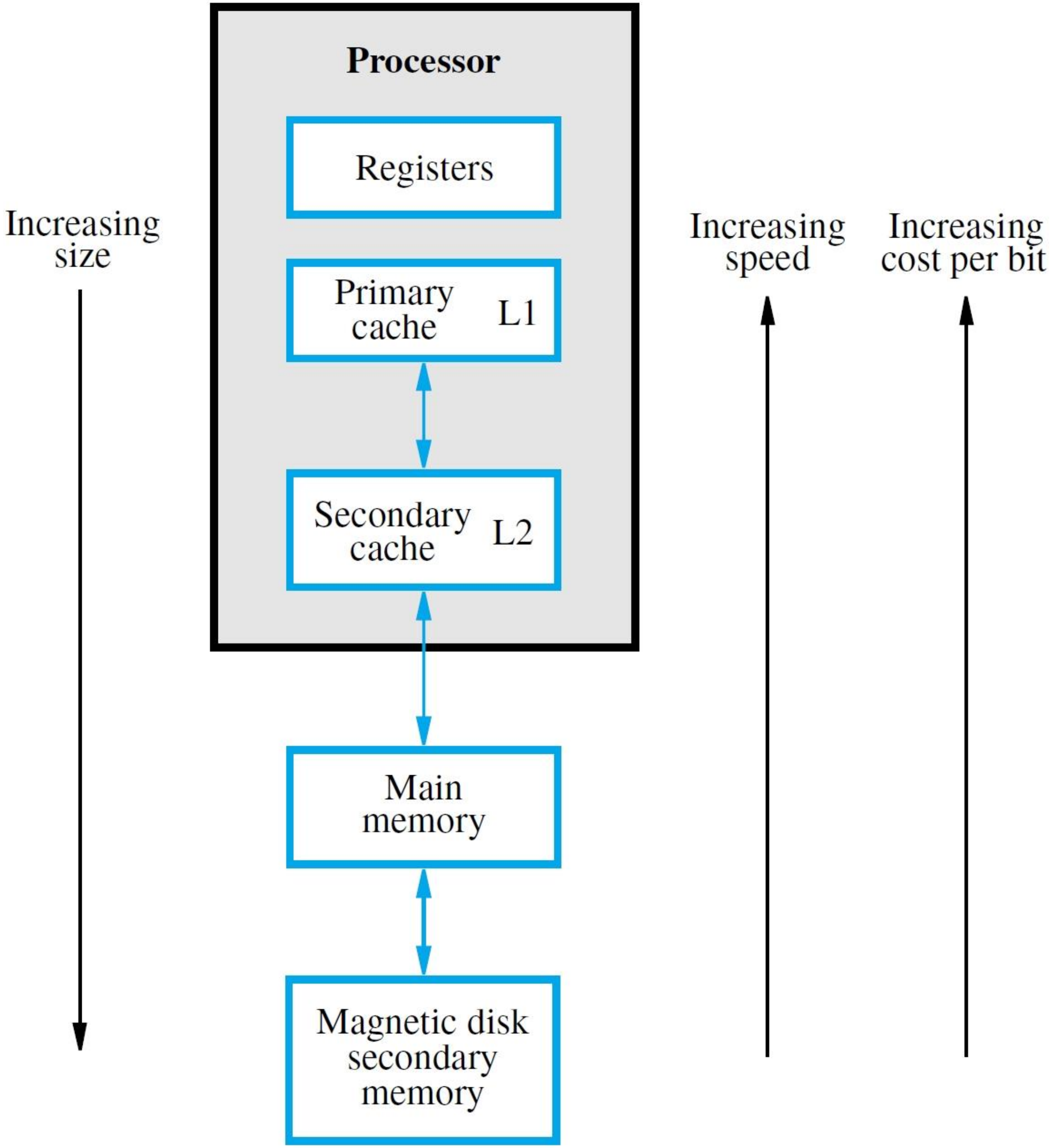
- Program-controlled I/O requires processor to intervene frequently for many data transfers
- Overhead is high because each transfer involves only a single word or a single byte
- Interrupt state-saving and operating system also introduce overheads for small data size
- Alternative: direct memory access (DMA)
- Special unit manages the transfer of larger *blocks* of data between memory & I/O devices

DMA Controller

- *DMA controller* is shared, or in each I/O device
- Performs individual memory accesses that would have been done by the processor
- 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
- DMA controller examples: disk and Ethernet

Memory Hierarchy

- Ideal memory is fast, large, and inexpensive
- Not feasible, so use memory hierarchy instead
- Exploits program behavior to make it *appear* as though memory is fast and large
- Recognizes speed/capacity/cost features of different memory technologies
- Fast static memories are closest to processor
- Slower dynamic memories for more capacity
- Slowest disk memory for even more capacity



Processor

Registers

Primary cache L1

Secondary cache L2

Main memory

Magnetic disk secondary memory

Increasing size

Increasing speed

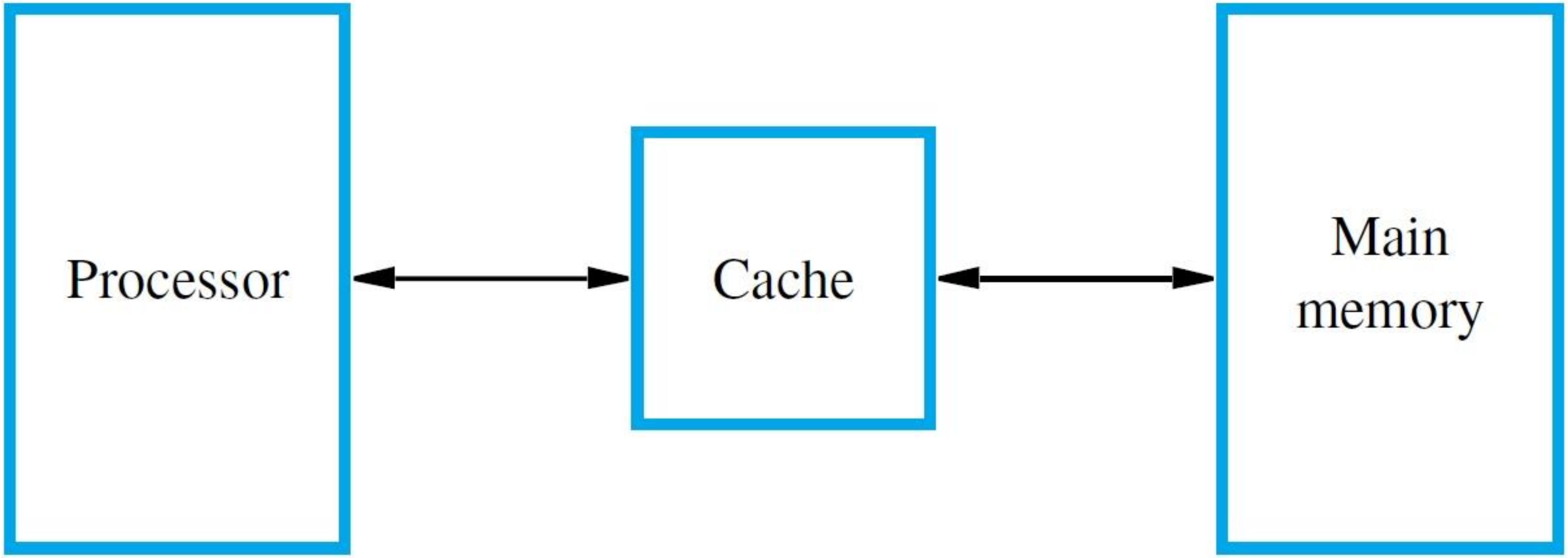
Increasing cost per bit

Memory Hierarchy

- Processor registers are fastest, but do not use the same address space as the memory
- Cache memory often consists of 2 (or 3) levels, and technology enables on-chip integration
- Holds copies of program instructions and data stored in the large external main memory
- For very large programs, or multiple programs active at the same time, need more storage
- Use disks to hold what exceeds main memory

Caches and Locality of Reference

- The cache is between processor and memory
- Makes large, slow main memory *appear* fast
- Effectiveness is based on locality of reference
- Typical program behavior involves executing instructions in loops and accessing array data
- Temporal locality: instructions/data that have been recently accessed are likely to be *again*
- Spatial locality: *nearby* instructions or data are likely to be accessed after current access


Processor
Cache
Main memory

More Cache Concepts

- To exploit spatial locality, transfer *cache block* with multiple adjacent words from memory
- Later accesses to nearby words are fast, provided that cache still contains the block
- *Mapping function* determines where a block from memory is to be located in the cache
- When cache is full, *replacement algorithm* determines which block to remove for space

Cache Operation

- Processor issues Read and Write requests as if it were accessing main memory directly
- But control circuitry first checks the cache
- If desired information is present in the cache, a *read* or *write hit* occurs
- For a read hit, main memory is not involved; the cache provides the desired information
- For a write hit, there are two approaches

Handling Cache Writes

- *Write-through protocol*: update cache & mem.
- *Write-back protocol*: only update the cache; memory updated later when block is replaced
- Write-back scheme needs *modified* or *dirty bit* to mark blocks that are updated in the cache
- If same location is written repeatedly, then write-back is much better than write-through
- Single memory update is often more efficient, even if writing back unchanged words

Handling Cache Misses

- If desired information is not present in cache, a *read* or *write miss* occurs
- For a read miss, the block with desired word is transferred from main memory to the cache
- For a write miss under write-through protocol, information is written to the main memory
- Under write-back protocol, first transfer block containing the addressed word into the cache
- Then overwrite specific word in cached block

Mapping Functions

- Block of consecutive words in main memory must be transferred to the cache after a miss
- The *mapping function* determines the location
- Study three different mapping functions
- Use small cache with 128 blocks of 16 words
- Use main memory with 64K words (4K blocks)
- *Word*-addressable memory, so 16-bit address

16 bit address

128 blocks
16 words
each

4k blocks
64K words

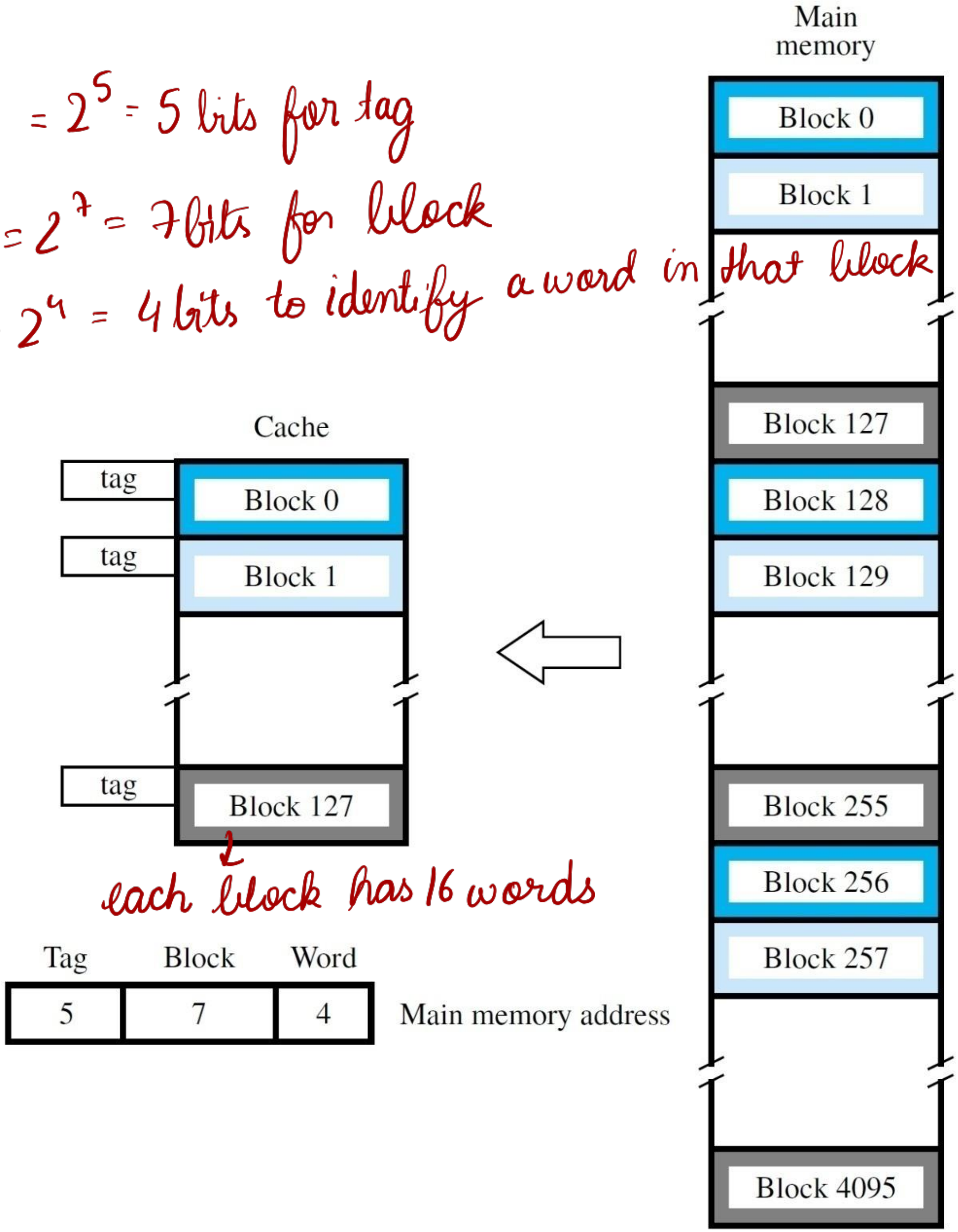
Direct Mapping

- Simplest approach uses a fixed mapping:
  memory block *j* → cache block (*j* mod 128)
- Only one unique location for each mem. block
- Two blocks may contend for same location
- New block always overwrites previous block
- Divide address into 3 fields: word, block, tag
- Block field determines location in cache
- Tag field from original address stored in cache
- Compared with later address for hit or miss

$\frac{4096}{128} = 32 = 2^5 = 5$ bits for tag

$128 = 2^7 = 7$ bits for block

$16 = 2^4 = 4$ bits to identify a word in that block



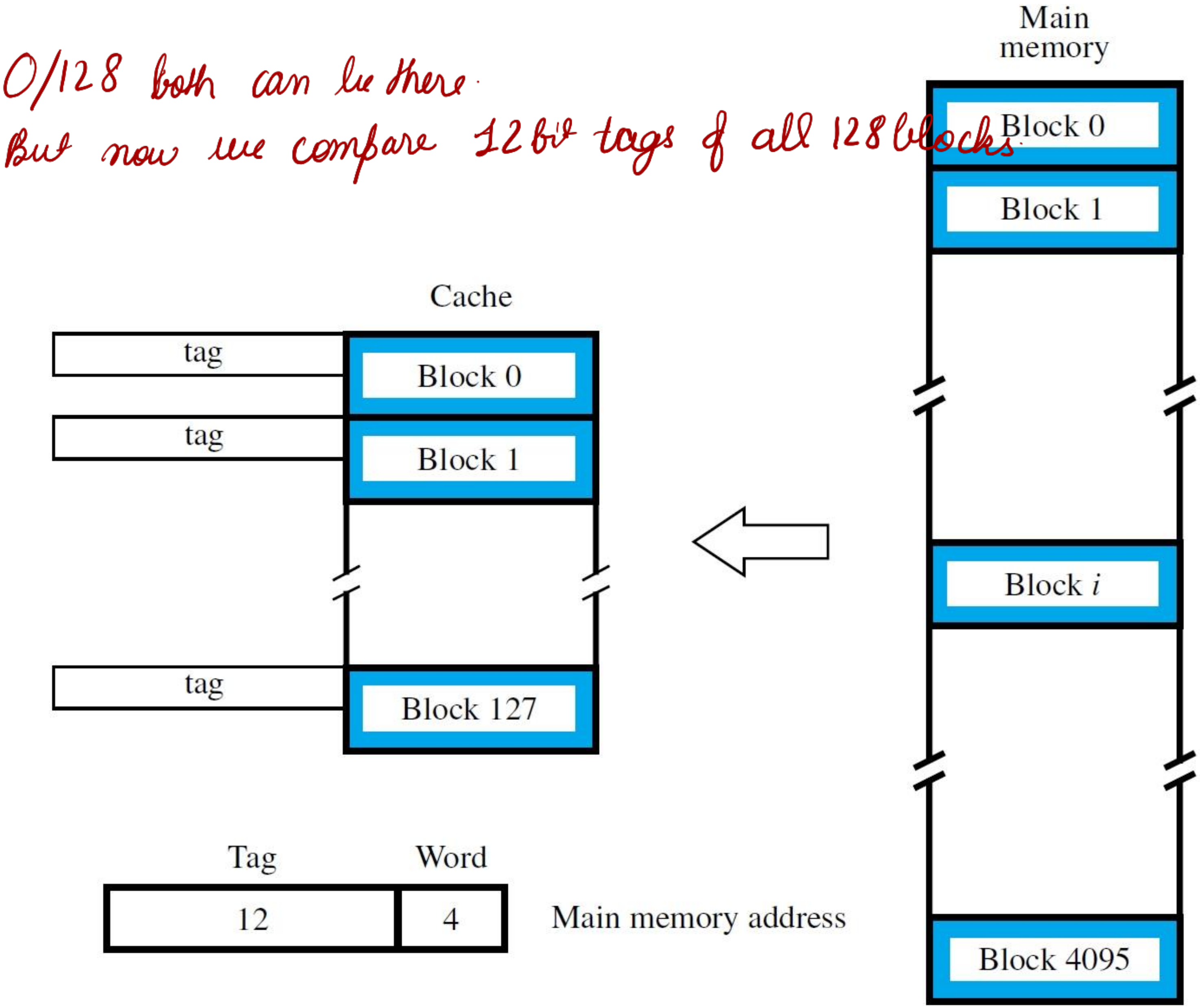
each block has 16 words

Associative Mapping

- Full flexibility: locate block anywhere in cache
- Block field of address no longer needs any bits
- Tag field is enlarged to encompass those bits
- Larger tag stored in cache with each block
- For hit/miss, compare all tags simultaneously in parallel against tag field of given address
- This *associative search* increases complexity
- Flexible mapping also requires appropriate replacement algorithm when cache is full

0/128 both can be there.

But now we compare 12 bit tags of all 128 blocks.



Set-Associative Mapping

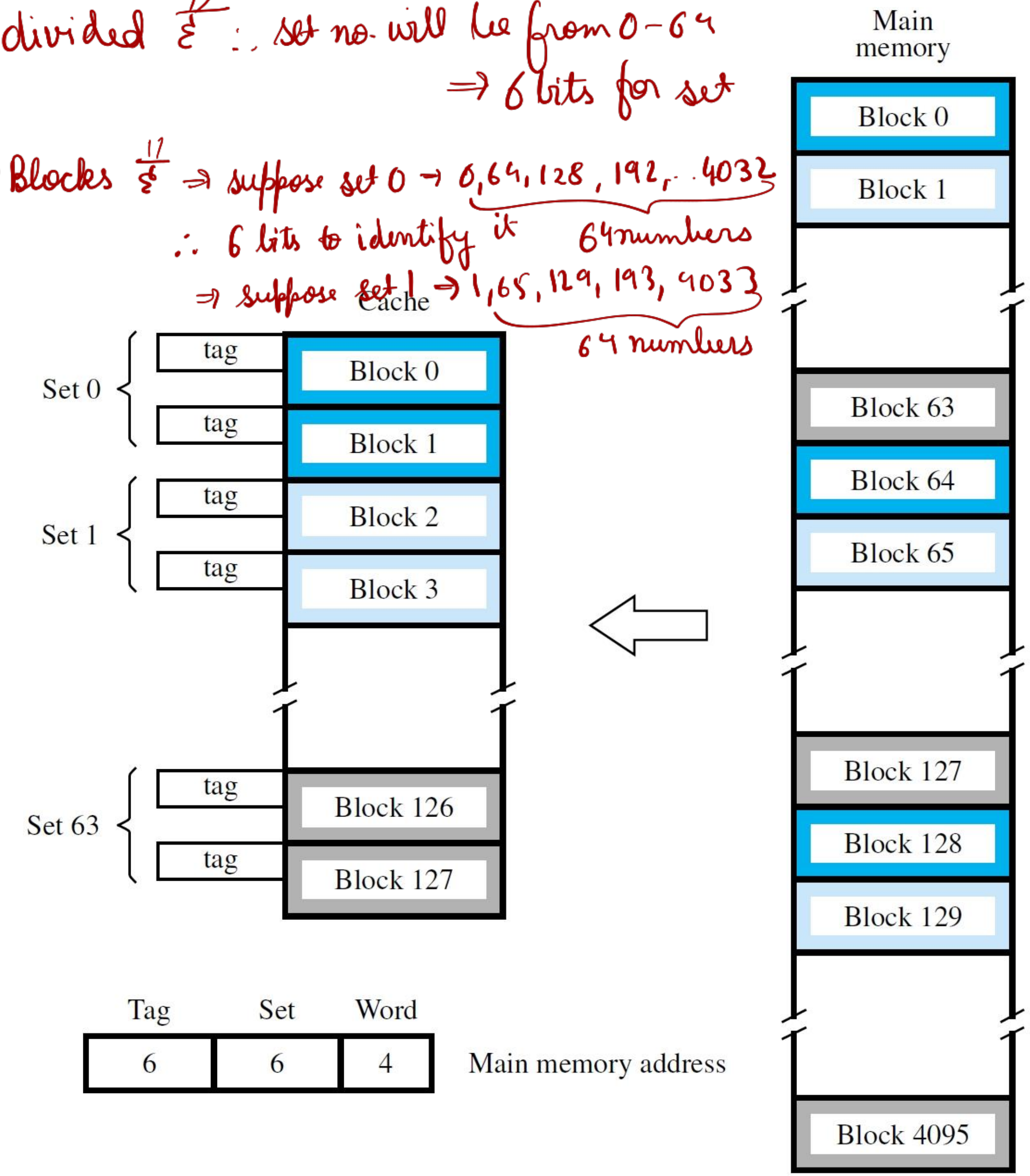
- Combination of direct & associative mapping
- Group blocks of cache into *sets*
- Block field bits map a block to a unique set
- But any block within a set may be used
- Associative search involves only tags in a set
- Replacement algorithm is only for blocks in set
- Reducing flexibility also reduces complexity
- *k* blocks/set → *k*-way set-associative cache
- Direct-mapped = 1-way; associative = all-way

64 sets में divided हैं ∴ set no. will be from 0-64
⇒ 6 bits for set

1 set में 2 Blocks हैं ⇒ suppose set 0 → $\underbrace{0, 64, 128, 192, \ldots 4032}_{64 \text{ numbers}}$

∴ 6 bits to identify it

⇒ suppose set 1 → $\underbrace{1, 65, 129, 193, 4033}_{64 \text{ numbers}}$



Stale Data

- Each block has a valid bit, initialized to 0
- No hit if valid bit is 0, even if tag match occurs
- Valid bit set to 1 when a block placed in cache
- Consider direct memory access with disk drive
- Disk → memory transfers: what about cache?
- Cache may contain *stale* data from memory, so valid bits are cleared to 0 for those blocks
- Memory → disk transfers: avoid stale data by *flushing* modified blocks from cache to mem.

LRU Replacement Algorithm

- Replacement is trivial for direct mapping, but need a method for associative mapping

recently used data will be used again

- Consider temporal locality of reference and use a *least-recently-used (LRU)* algorithm
- For *k*-way set associativity, each block in a set has a counter ranging from 0 to *k*–1
- Hitting on a cache block clears its counter value to 0; others originally lower in set are incremented → ensures that the least recently used block always has the highest counter value

oldest

LRU Replacement Algorithm

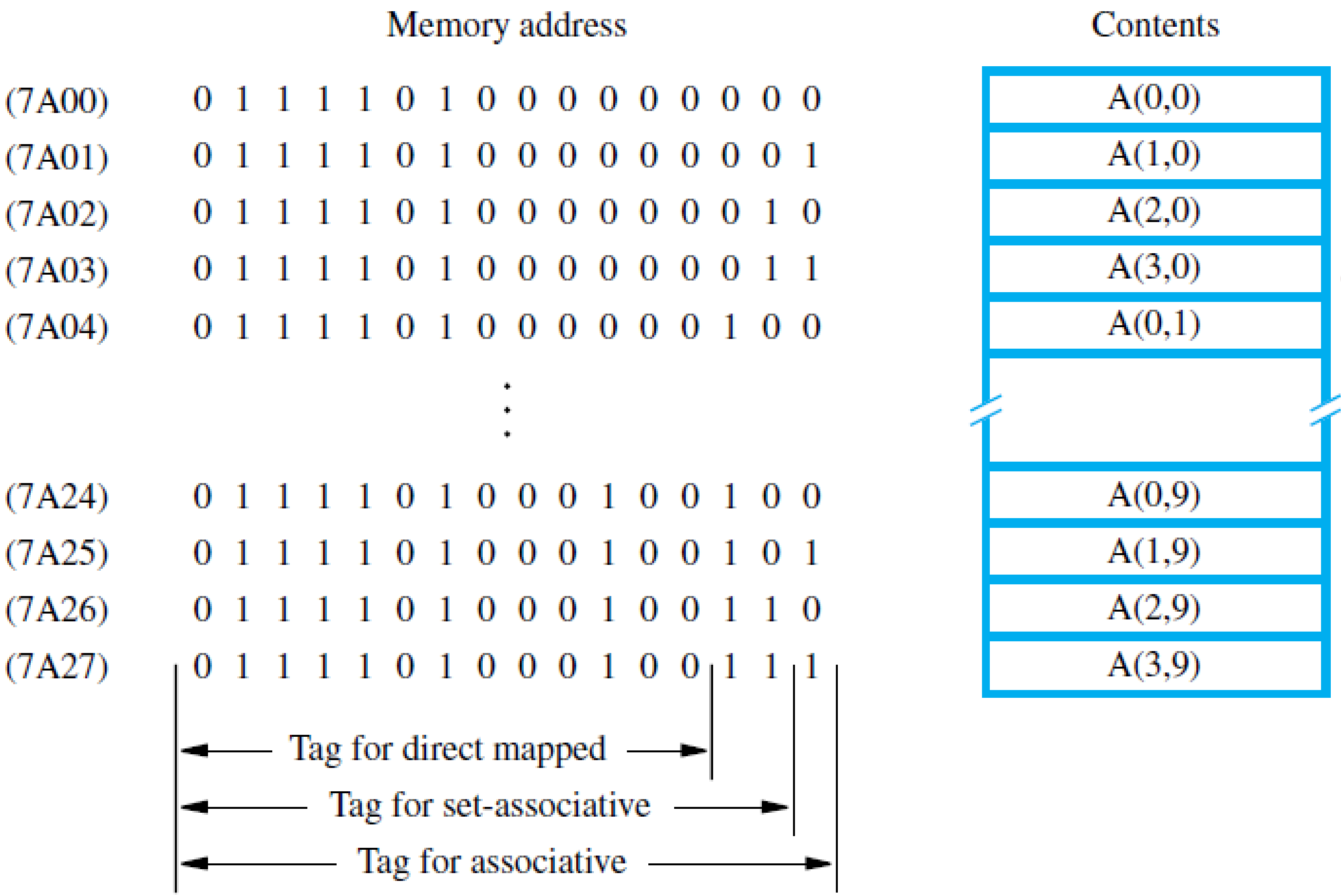
- If a *miss* occurs and the set is not full, the counter associated with the new block loaded from the main memory is set to 0, and the values of all other counters are increased by one.
- If set is full, replace the block with highest counter value, i.e., k-1. The new block is put in its place, and its counter is set to 0. Other block counters are incremented by one.

Illustrative Example

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

→ dividing by average of that row

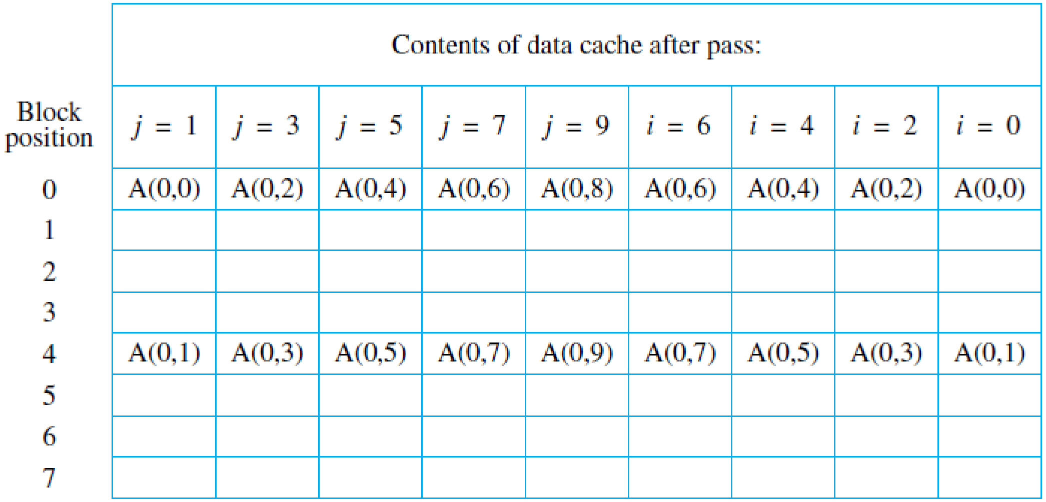
```
SUM := 0

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 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)

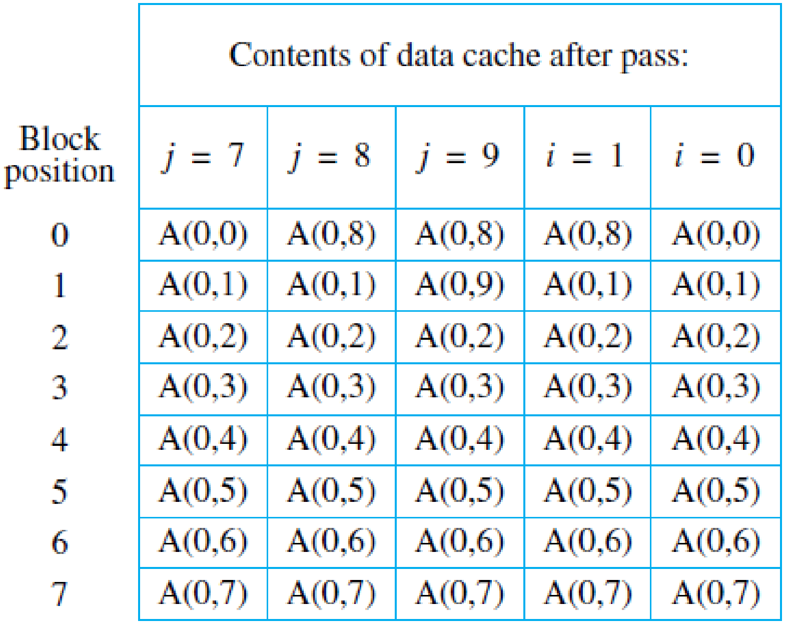
Q. Cache has space for only 8 blocks of data.
- Each block consists of 16 bit data.
- 16 bit address

Illustrative Example (contd.)



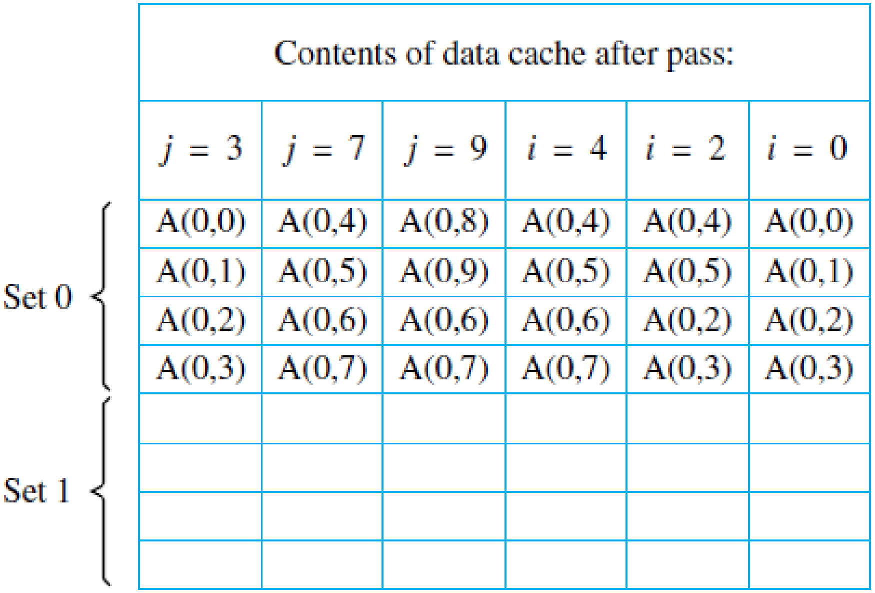
Block position	Contents of data cache after pass:								
	$j = 1$	$j = 3$	$j = 5$	$j = 7$	$j = 9$	$i = 6$	$i = 4$	$i = 2$	$i = 0$
0	A(0,0)	A(0,2)	A(0,4)	A(0,6)	A(0,8)	A(0,6)	A(0,4)	A(0,2)	A(0,0)
1									
2									
3									
4	A(0,1)	A(0,3)	A(0,5)	A(0,7)	A(0,9)	A(0,7)	A(0,5)	A(0,3)	A(0,1)
5					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

Contents of a direct-mapped data 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)

Contents of an associative-mapped data cache.



	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						

Contents of a set-associative-mapped data cache.

Performance Consideration

- Hit Rate and Miss Penalty
  - Number of hits as a fraction of all attempted accesses is called the *hit rate*
  - *miss rate* is the number of misses stated as a fraction of attempted accesses.
  - total access time seen by the processor when a miss occurs as the *miss penalty*
  - Average time experience by processor:
    - t_{avg}= hC + (1-h)M, where h is hit rate, C is cache access time and M is miss penalty

Performance Consideration (Contd.)

- Access times to the cache and main memory are τ and 10τ , respectively.
- When a cache miss occurs, a block of 8 words is transferred from the main memory to the cache.
- It takes 10τ to transfer the first word of the block, and the remaining 7 words are transferred at the rate of one word every τ seconds.
- Miss penalty also includes a delay of τ for the initial access to the cache, which misses, and another delay of τ to transfer the word to the processor after the block is loaded into the cache.
- Thus, the miss penalty in this computer is given by:

$$M = \underbrace{\tau}_{\text{delay to transfer word to processor}} + \underbrace{10\tau}_{\text{for 1st word}} + \underbrace{7\tau}_{\text{for next 7 words}} + \underbrace{\tau}_{\text{delay for initial access to cache}} = 19\tau$$

Performance Consideration (Contd.)

- Assume that 30 percent of the instructions in a typical program perform a Read or a Write operation
- It means that there are 130 memory accesses for every 100 instructions executed. (100 for fetching + 30% for data access)
- Assume that the hit rates in the cache are 0.95 for instructions and 0.9 for data.
- Assume further that the miss penalty is the same for both read and write accesses

Access time to cache = τ

to main memory = 10τ

Performance Consideration (Contd.)

- ✓ Time without cache/ Time with cache =

  $(130 \times 10\tau)/(100(0.95\tau + 0.05 \times 19\tau) + 30(0.9\tau + 0.1 \times 19\tau)) = 4.7.$

- ✓ Time for Real cache/ Time for Ideal cache =

  $(100(0.95\tau + 0.05 \times 19\tau) + 30(0.9\tau + 0.1 \times 19\tau))/(130\tau) = 2.1.$

- For two-level cache with L1 and L2,

  $t_{avg} = h_1 C_1 + (1 - h_1)(h_2 C_2 + (1-h_2)M)$ where

  h_1 is the hit rate for L1, h_2 is the hit rate for L2, C_1 is the time to access data in L1, C_2 is the miss penalty to transfer data from L2 to L1, M is the miss penalty to transfer data from main memory to L2.

Virtual Memory

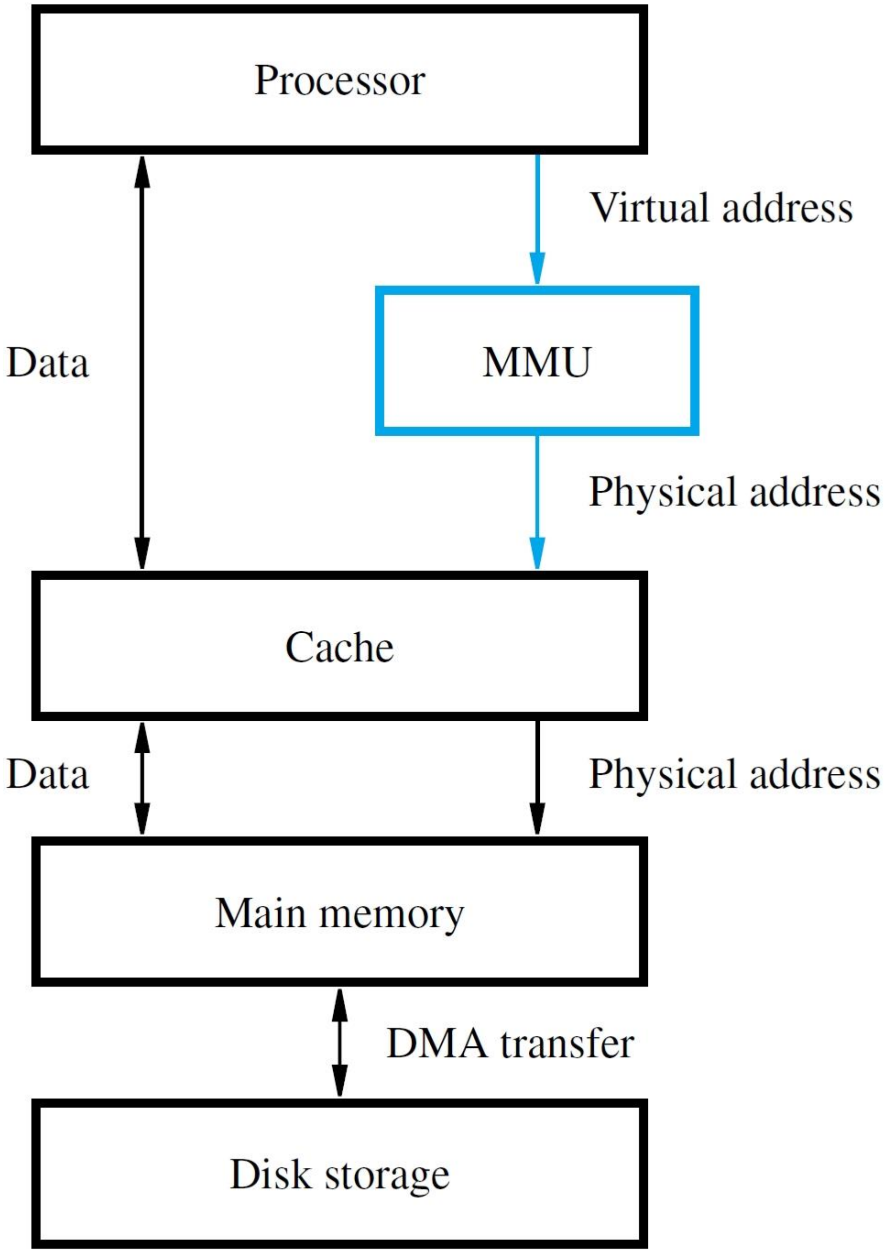
- Physical mem. capacity $\leq$ address space size
- A large program or many active programs may not be entirely resident in the main memory
- Use secondary storage (e.g., magnetic disk) to hold portions exceeding memory capacity
- Needed portions are *automatically* loaded into the memory, replacing other portions
- Programmers need not be aware of actions; virtual memory hides capacity limitations

Virtual Memory

- Programs written assuming full address space
- Processor issues *virtual* or *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
- System may select any physical address; no unique assignment for a virtual address

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

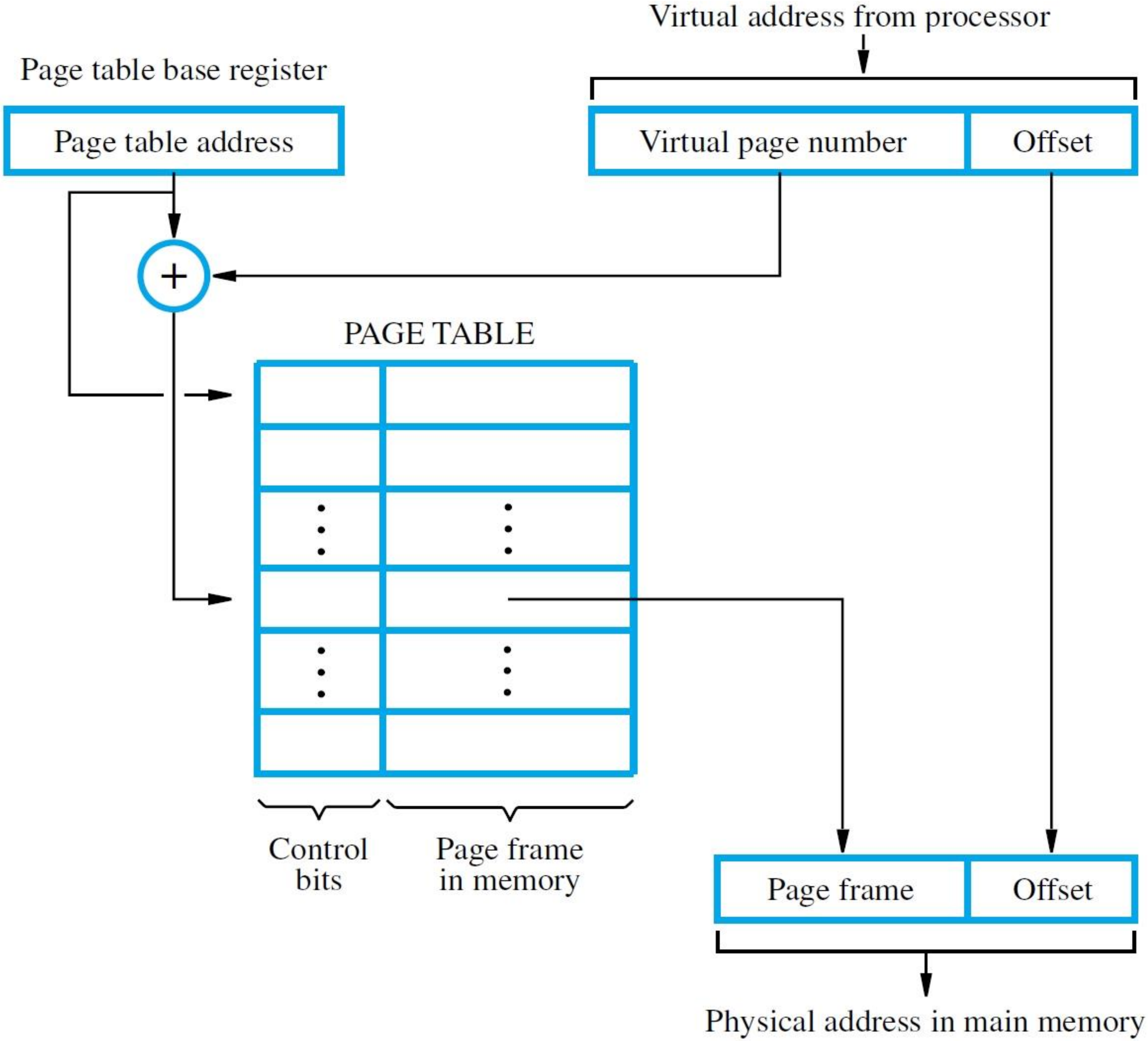

Processor
Virtual address
Data
MMU
Physical address
Cache
Data
Physical address
Main memory
DMA transfer
Disk storage

Address Translation

- Use fixed-length unit of *pages* (2K-16K bytes)
- Larger size than cache blocks due to slow disks
- 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

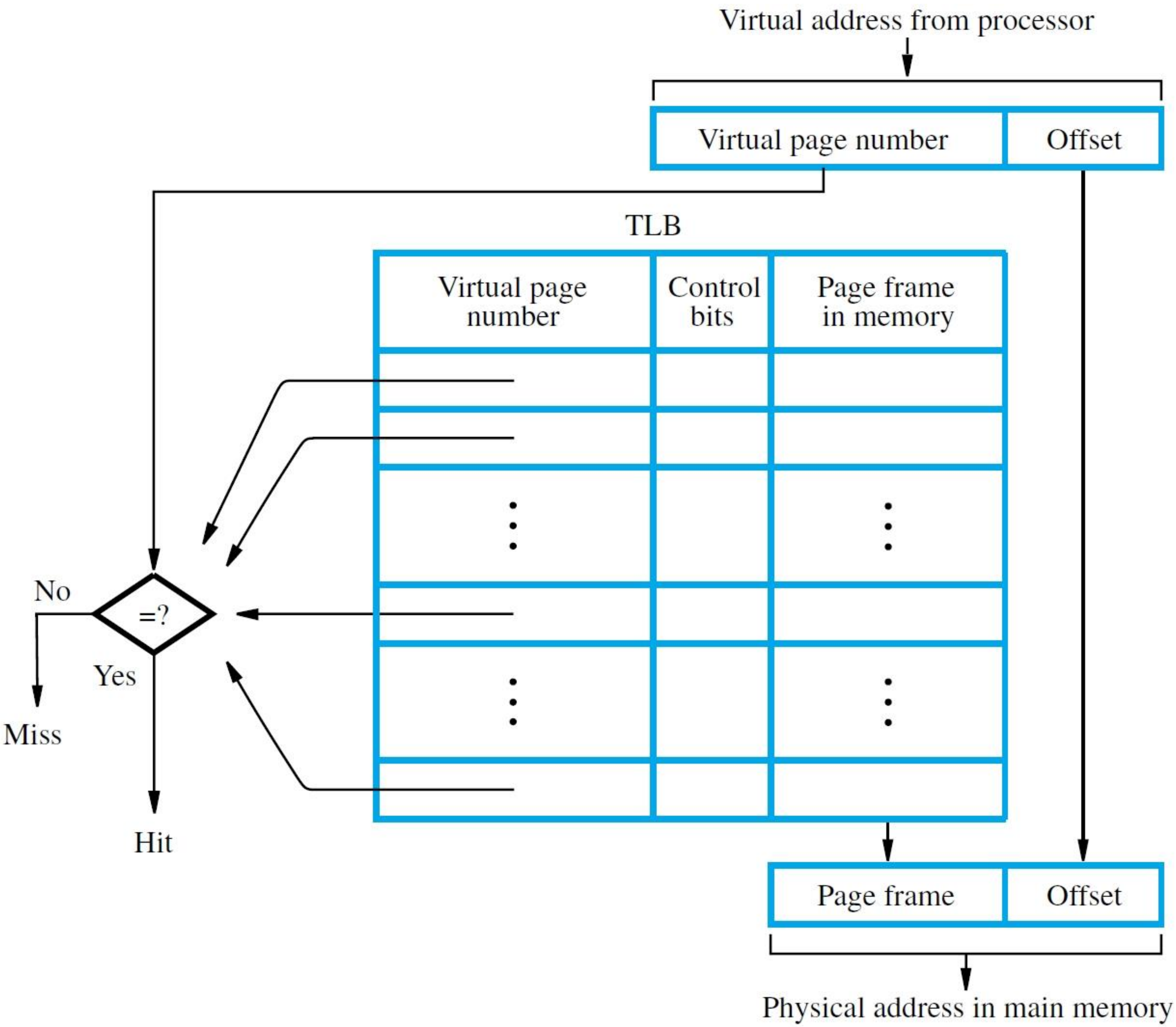
Page Table

- 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 copy-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 copy-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

Memory Management Requirements

- Virtual address space divided as system space and user space (one space per user)
- One page table per process
- MMU uses Page Table Base Register to determine where the corresponding page table is located
- By changing PTBR content, OS can switch from one space to another
- Processor runs in two modes – supervisor mode and user mode
- Privileged instructions can be run in supervisor mode only – e.g., update the PTBR. Thus user processes cannot overwrite each other's data
- For data sharing, pages may be shared between processes – with control bits set to read/write, etc.

Sections to Read (From Hamacher's Book)

- Chapter on Memory System
  - All sections and sub-sections except 8.3.5 (Flash Memory) 8.10 (Secondary Storage)