Memory System Organization & Architecture

MODULE 4

Contents

- Memory systems hierarchy
- Main memory organization: Types of Main memory, memory interleaving and its characteristics and performance
- Cache memories: address mapping-line size-replacement and policies- coherence
- Virtual memory systems- TLB-
- Reliability of memory systems- error detecting and error correcting systems.

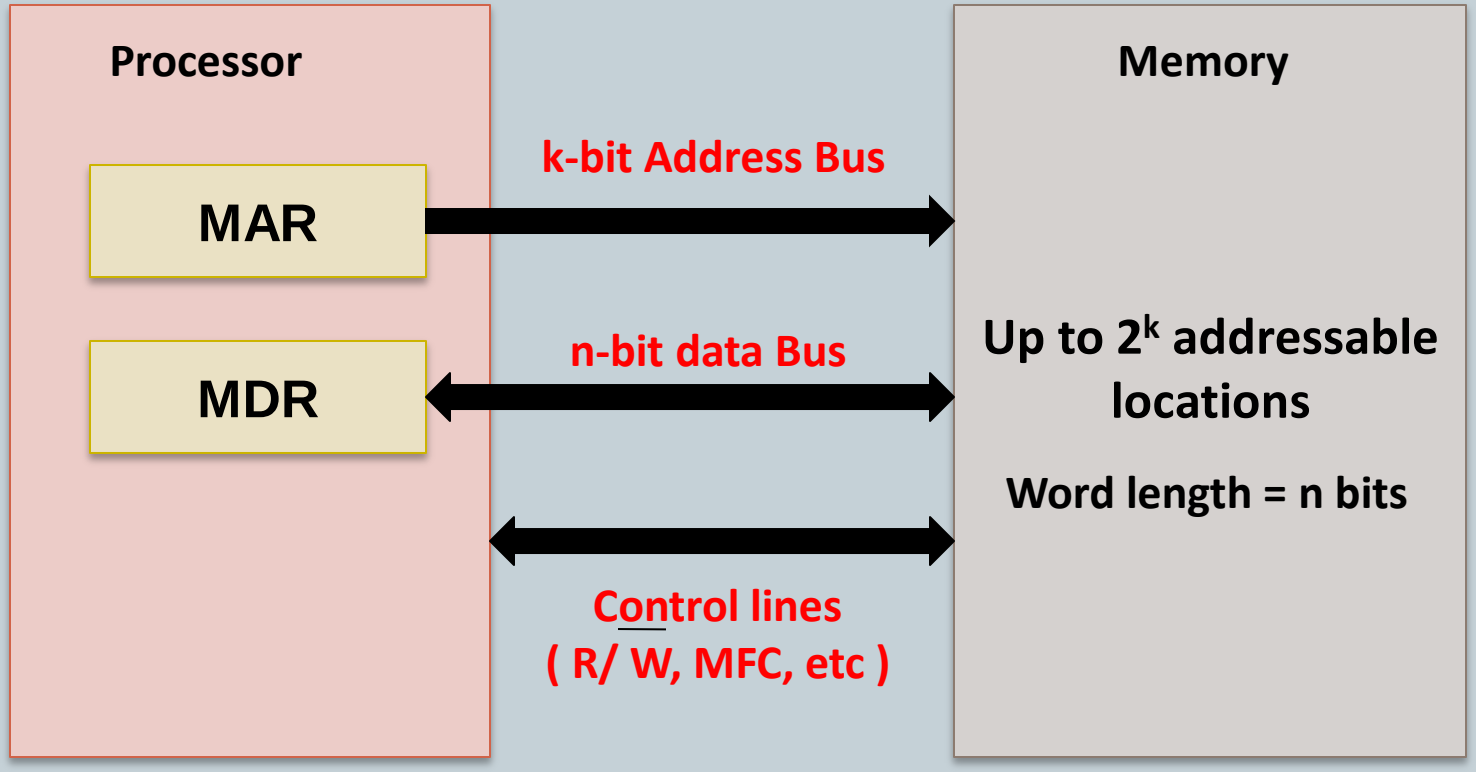
Outline

- Memory Cells – SRAM & DRAM
- Internal Organization of Memory Chip
- Organization of a SRAM memory unit
- Organization of a DRAM based Main Memory unit
- Error detection and correction memories
- Memory Hierarchy
- Cache Memory organization
- Virtual memory system design

Memory

- It’s all about storing bits – binary digits
- Vacuum tubes, CRT, Drums, Disks and ICs
- Issues – size, cost and speed
- Semiconductor memories (Chips)

Basic Concepts



Connection of Memory to the Processor

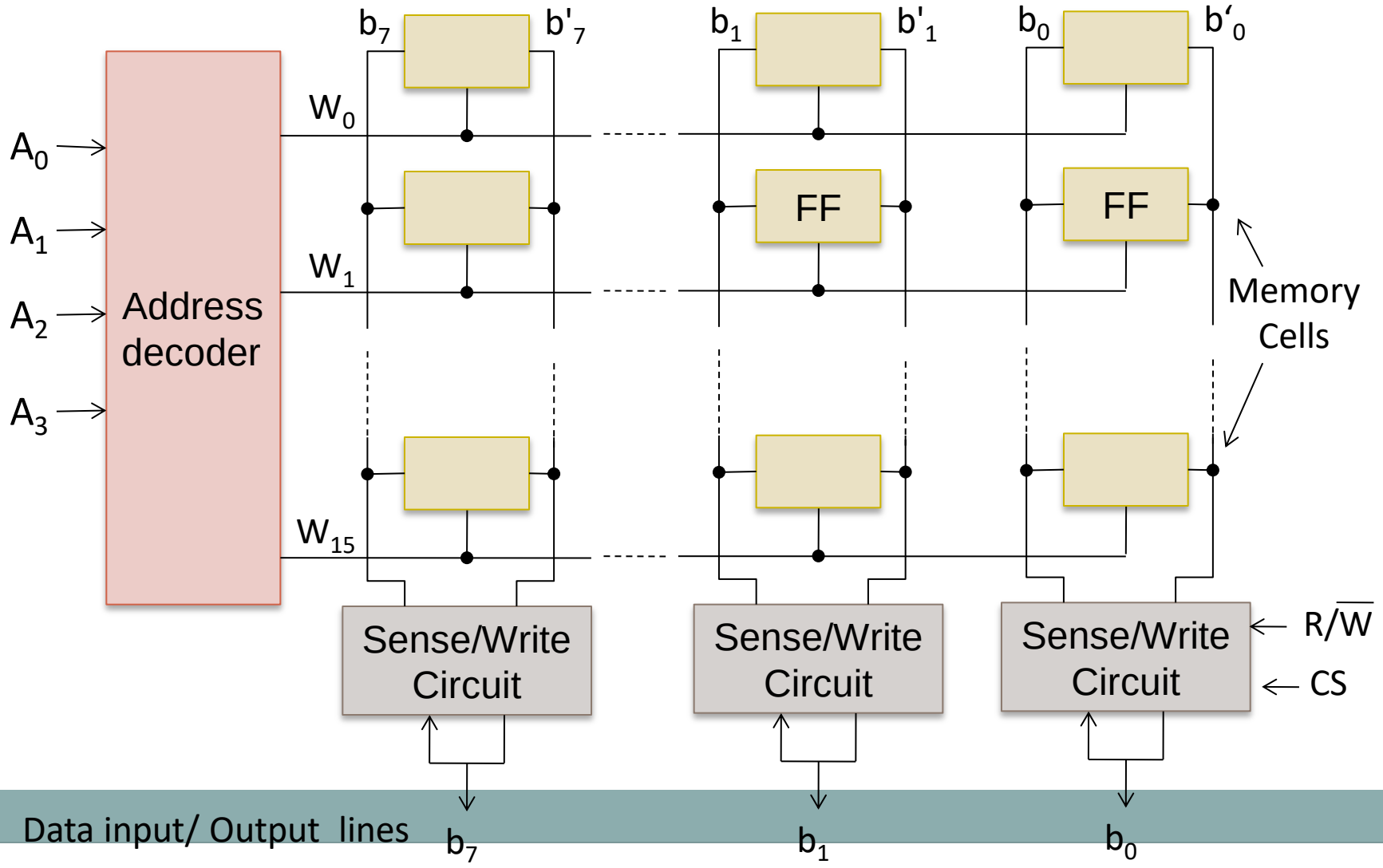
Terminology

- *Memory Access Time*: Time between read and the MFC (Memory Function Complete) signal.
- *Memory Cycle Time*: The minimum time delay required between the initiation of two successive memory operations.
- *Random Access Memory (RAM)*: Any location can be accessed for a read or write operation in some fixed amount of time that is independent of the location's address.

Semiconductor RAM Memories

- Semiconductor memories are available in a wide range of speeds.
- Their cycle time range from 100ns to less than 10ns.
- Obviously the speed of the processor depends on the speed of the memory

Internal Organization of Memory Cells

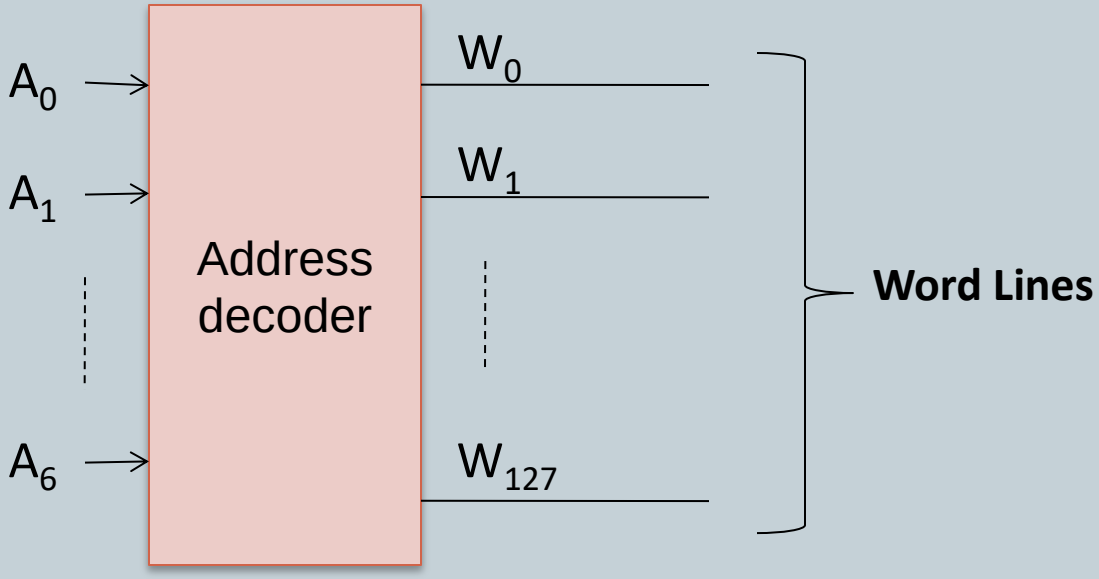


Internal Organization of Memory Cells

- Above Example: 128 bit chip
  - It have 4 address lines and 8 data lines
  - Using 4 address lines ($A_3A_2A_1A_0$) 16 word signals ($W_{15}...W_1W_0$) are generated with help of decoder 4:16 decoder
  - 16 words of each 8 bits (= 16 x 8 bits = 16 bytes)
  - For any given input address lines ($A_3A_2A_1A_0$) only one of the word lines from W_{15} ... W_0 will be active at a time.
  - Which in turn enable the complete row of the memory cells connected to W_i bit
  - Using R/W and CS signal one can enable the Sense/Write Circuits of the all bits ranging from b_7 to b_0

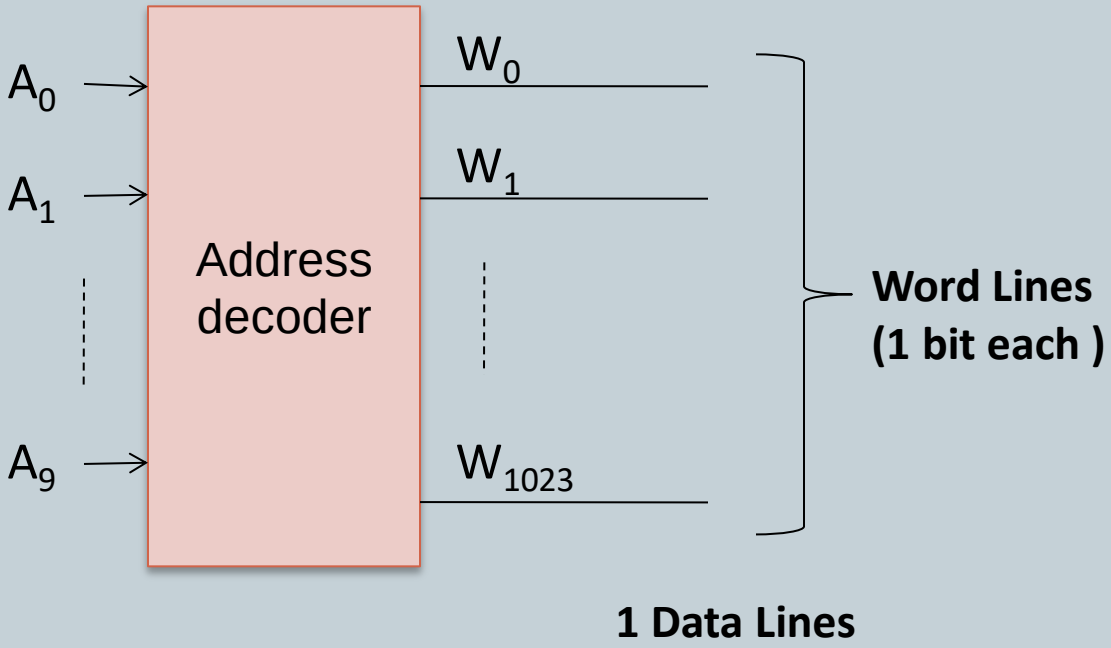
Chip with 1024 Memory Cells

- 1024 bits can be planned as 128 x 8 memory
- Total address lines required $128 = 2^N \rightarrow N = \log_2 128 = 7$, it means it should have A6 .. A_1A_0
- It means 128 x 8 bits (Similar to 16 x 8 bits)

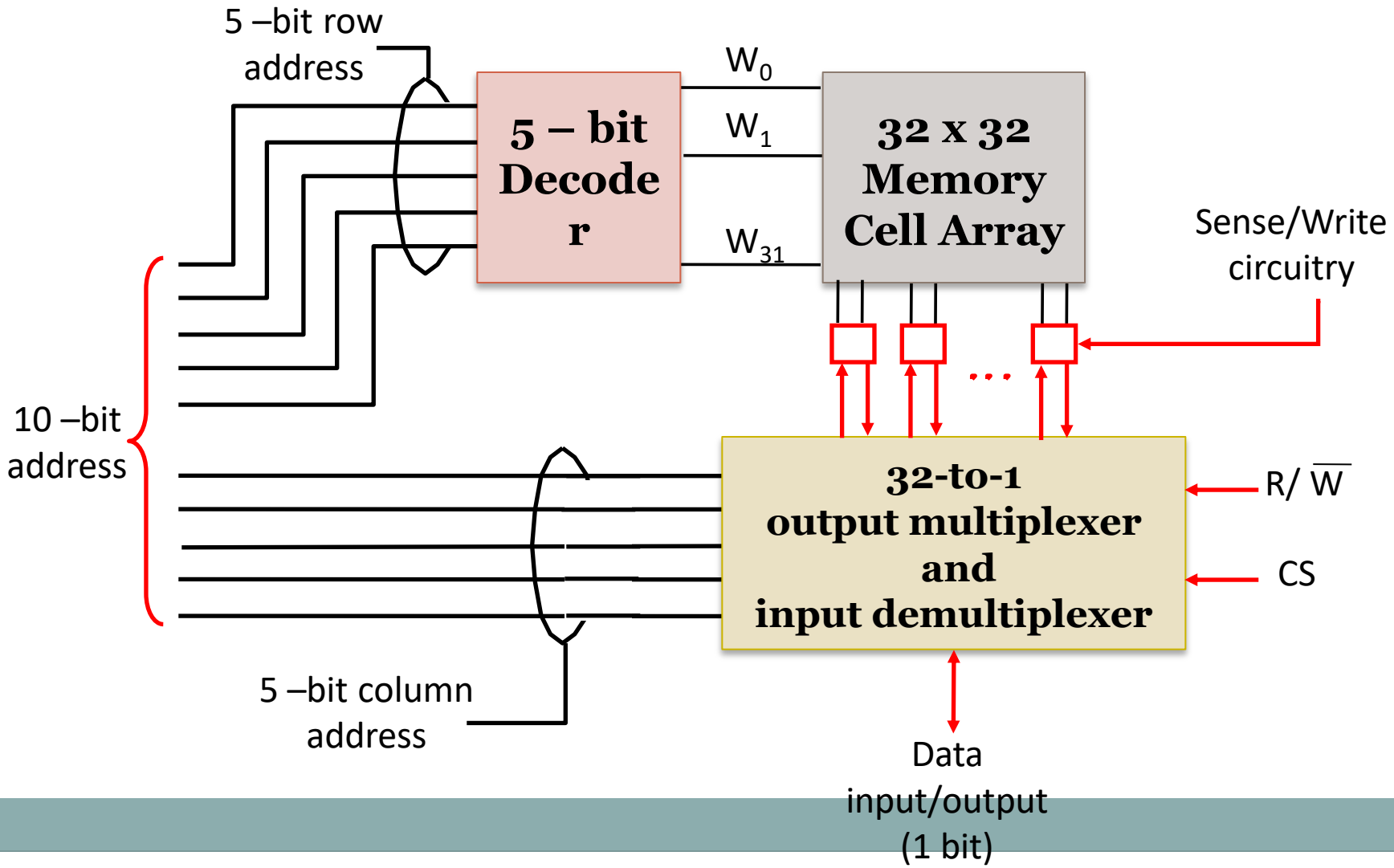


Chip with 1024 Memory Cells

- Or it could be 1024 x 1



Organization of a 1K × 1 memory chip

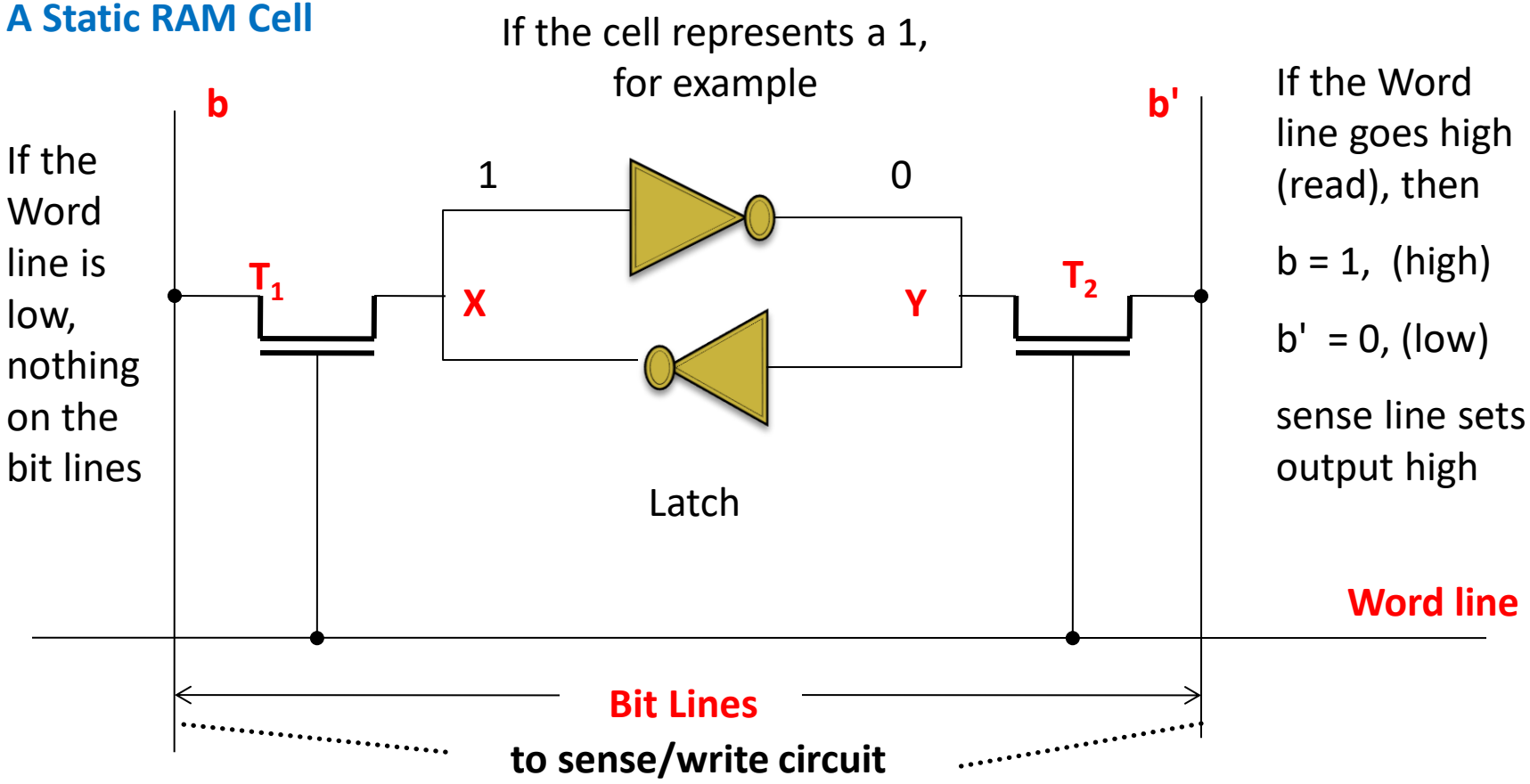


Static Memory

- Static: retains its state (content) as long as power is applied
  - Data will be erased from the cell if powered off (volatile)
- SRAM: Static RAM
  - Fast
  - Expensive

Semiconductor RAM Memories

A Static RAM Cell



- If the Word line goes high (read) ***b = 1, b' = 0***
- To write (say 0), put **b = 0**, **b' = 1**, set Word line high, latch changes

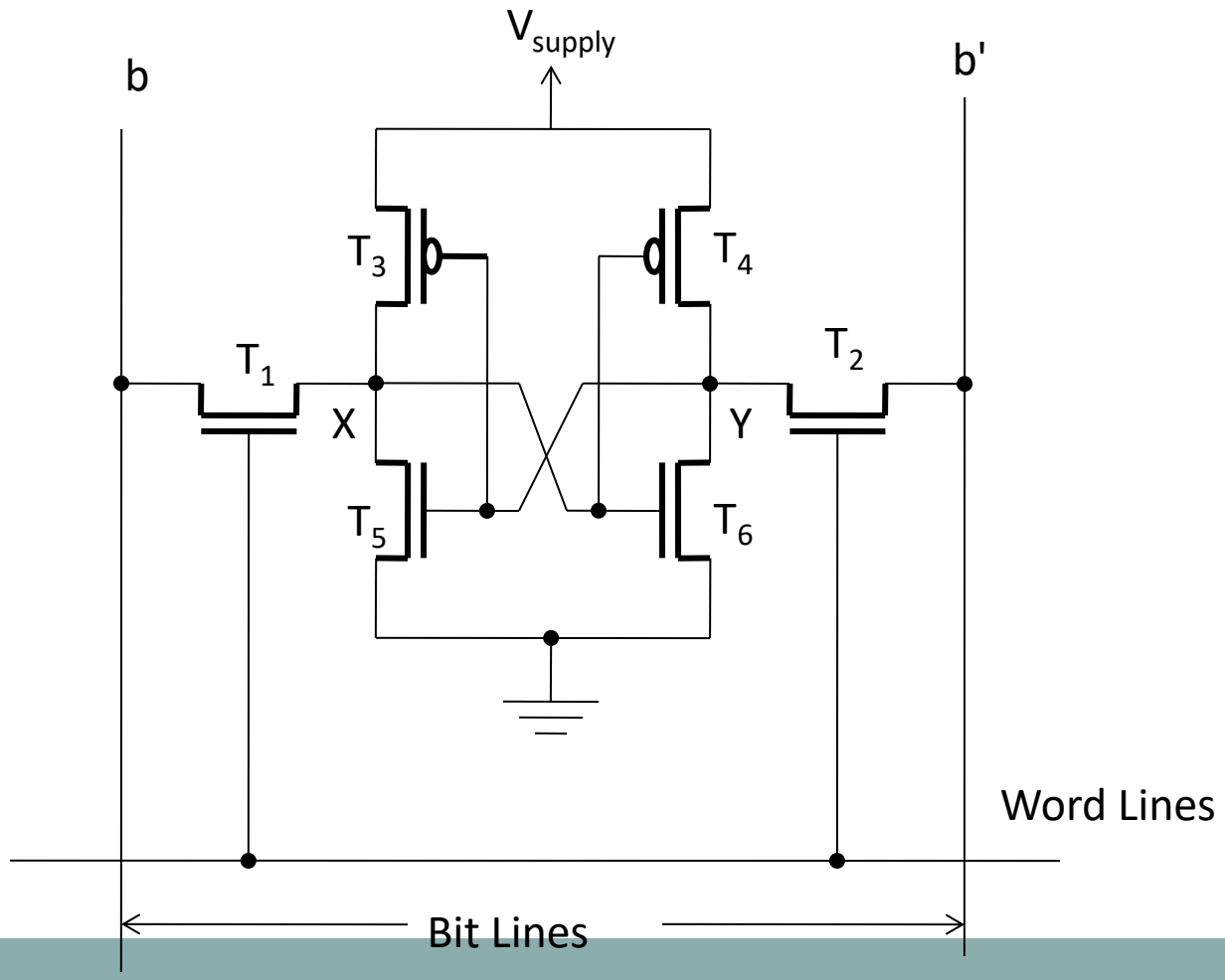
SRAM cells

- Two inverters are cross connected to form a latch
- The latch is connected to two bit lines by transistors T_1 and T_2.
- These transistors acts as switches that can be opened or closed under the control of word line
- When word line at ground level, the transistors are turned off and the latch retains its state
- Example:
  - Let us assume the cell is in state 1 if the logic value at point X is 1 and at point Y is 0.
  - This state is maintained as long as the signal on the word line is at ground level

SRAM Cells

- Read operation
  - The word line is activated to close the switches T1 and T2
  - If the cell 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.
  - Sense/write circuits at the end of the bit lines monitor the state of b and b' and set the output accordingly
- Write operation
  - The state of the cell is set by placing the appropriate value on bit line b and its complement on b'.
    - And then activating the word line
  - This forces the cell into the corresponding state.
  - The required signals on the bit lines are generated by the sense/write circuit.

CMOS Memory Cell



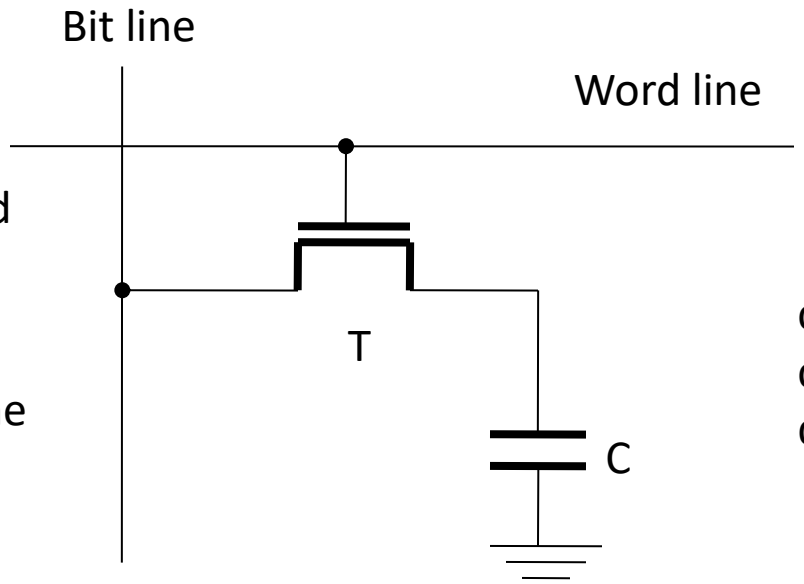
CMOS SRAM

- Volatile
- Low power consumption – no current flows except when being accessed
- Fast – access times of a few nanoseconds
- Expensive (6 Transistors per cell)

Dynamic RAM (DRAM)

- Simple cells, higher density
- 1 million to 16 million bits or more per chip
- Less expensive
- But DRAM cells do not retain their state
- Must be refreshed periodically

DRAM Cell



capacitor is charged to write a 1

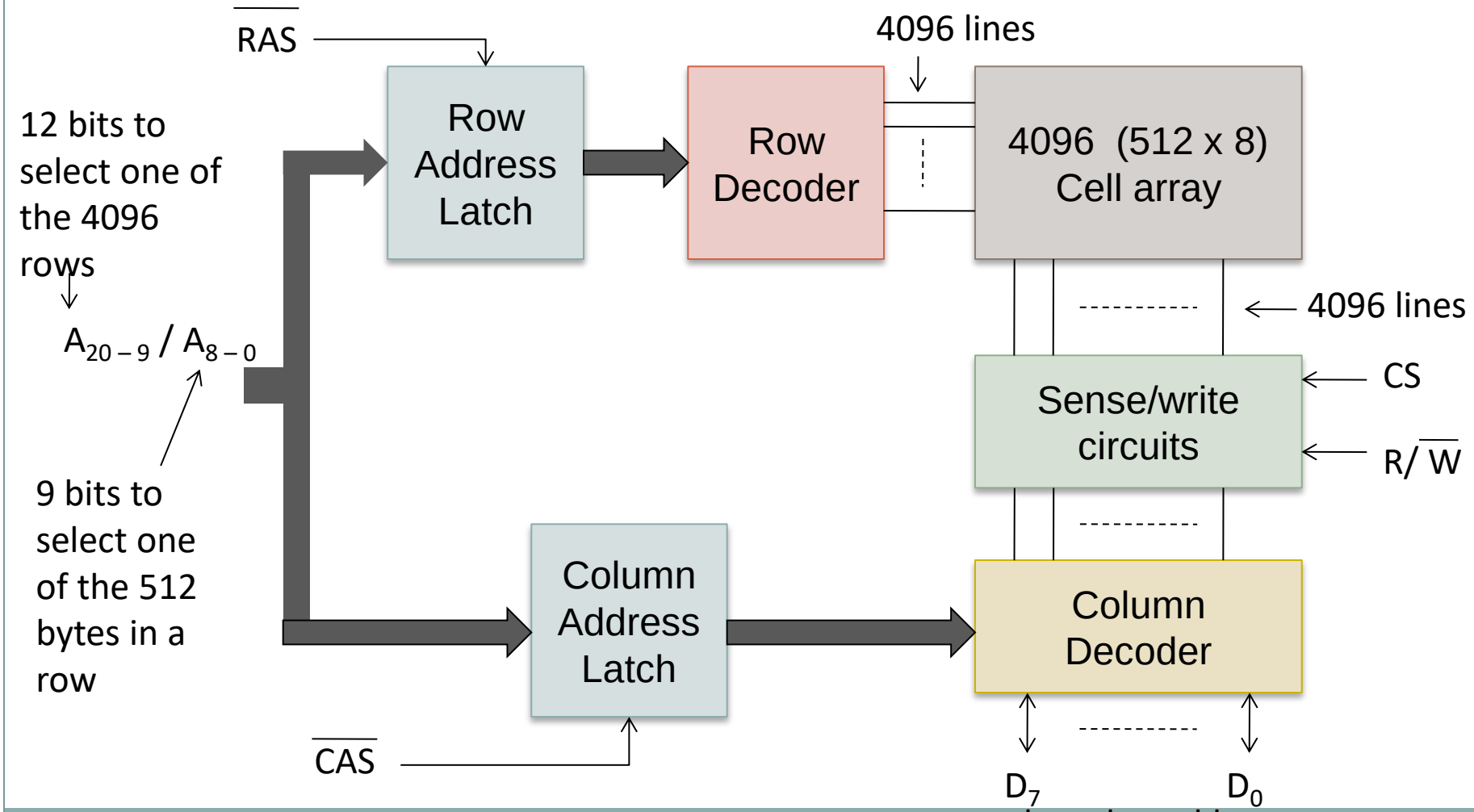
(voltage applied to Word line and to the Bit line)

charge on the capacitor will discharge over time

A single transistor dynamic memory cell

- If a Read detects a voltage on the capacitor above the “threshold, ” it “sees” a 1, and drives the bit line to full voltage and recharges the capacitor
- If a Read detects a voltage on the capacitor below the “threshold, ” it “sees” a 0, and drives the bit line to ground and fully discharges the capacitor
- Refreshes whenever read Refresh circuit will periodically read all cells

Internal Organization of DRAM



Internal Organization of a 2M x 8 dynamic memory chip

DRAM

- Possible to leave row selected (all 512 bytes on sense lines)
- Then rapidly retrieve successive bytes by changing column addresses
- Result is a “fast page mode” for “blocks” or “pages” of bytes where appropriate (such as cache loading, disk transfer)
- Or, *synchronous* DRAM, SDRAM

Synchronous DRAM

- Can operate in different modes
- “Burst” modes of different lengths
- Can transfer “blocks” of data on single Read or Write

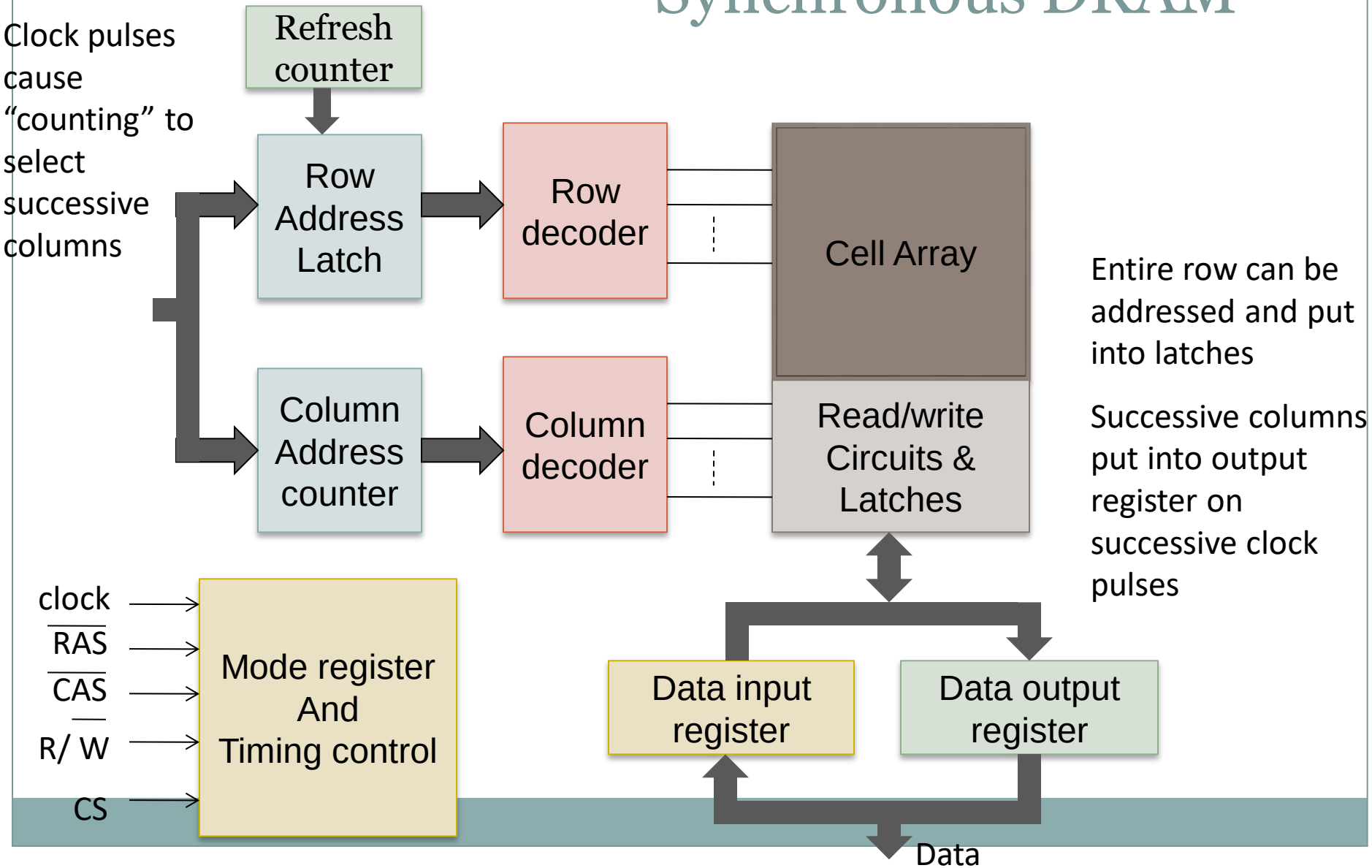
Latency and Bandwidth

- The speed and efficiency of data transfers among memory, processor, and disk have a large impact on the performance of a computer system.
- Memory latency – the amount of time it takes to transfer a word of data to or from the memory.
- Memory bandwidth – the number of bits or bytes that can be transferred in one second. It is used to measure how much time is needed to transfer an entire block of data.
- Bandwidth is not determined solely by memory. It is the product of the rate at which data are transferred (and accessed) and the width of the data bus.

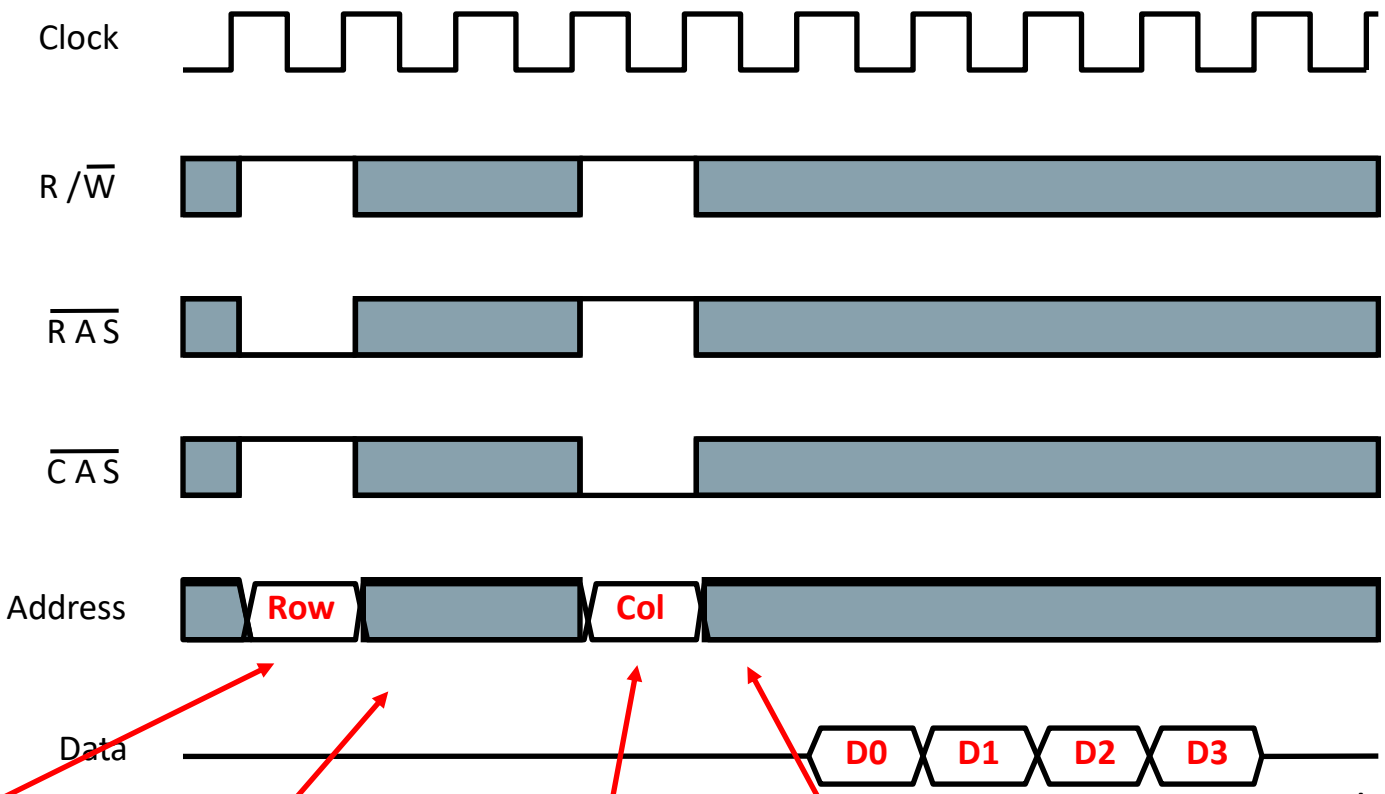
DDR SDRAM

- Double-Data-Rate SDRAM
- Standard SDRAM performs all actions on the rising edge of the clock signal.
- DDR SDRAM accesses the cell array in the same way, but transfers the data on both edges of the clock.
- The cell array is organized in two banks. Each can be accessed separately.
- DDR SDRAMs and standard SDRAMs are most efficiently used in applications where block transfers are prevalent.

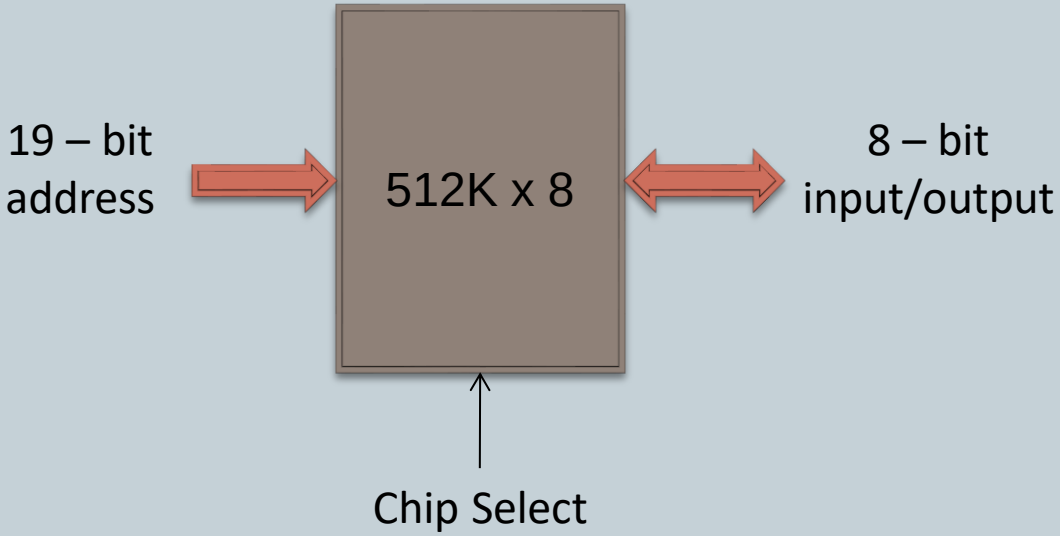
Synchronous DRAM



Burst Read of length 4 in an SDRAM



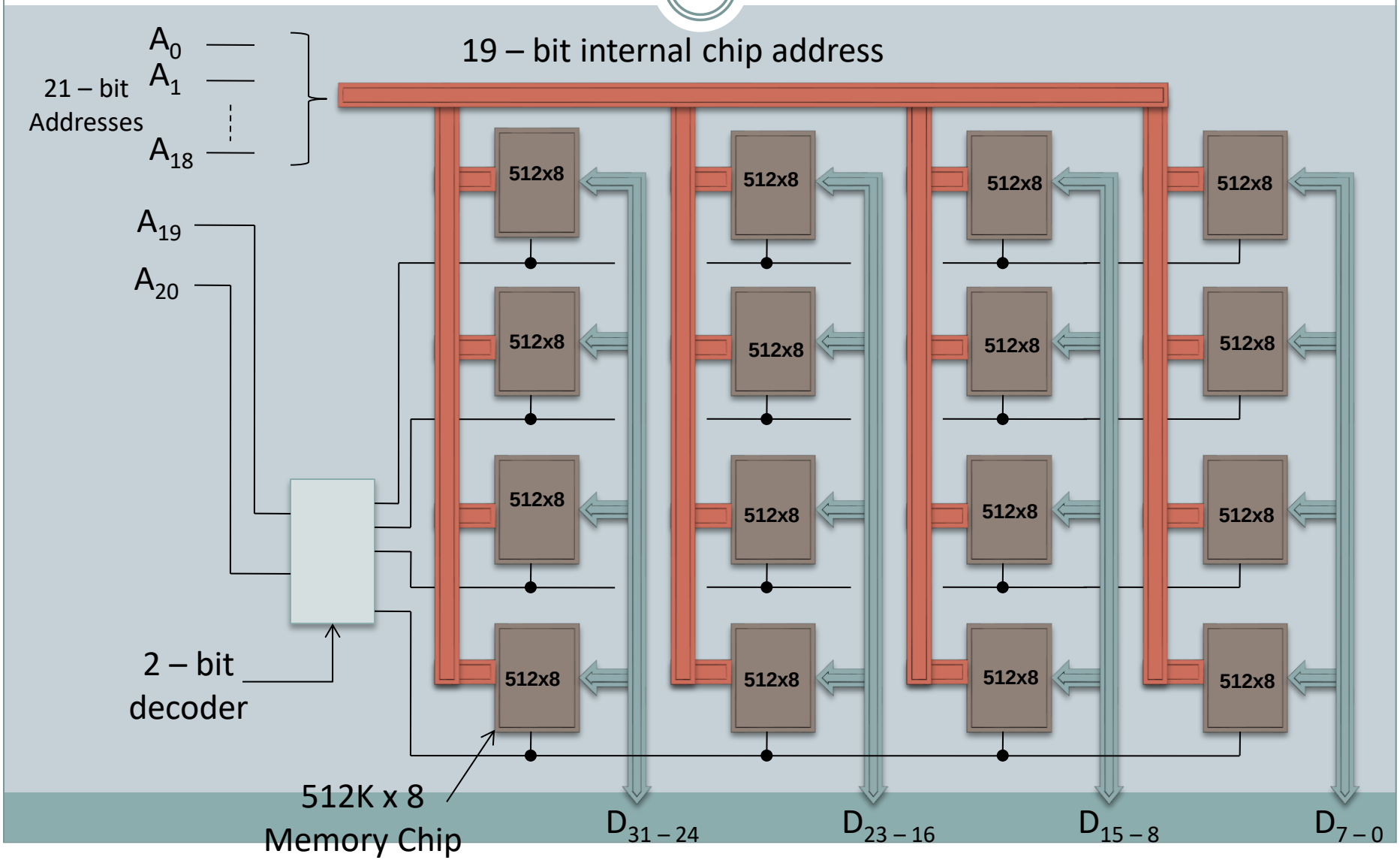
Larger Memories Using Multiple Chips



512K x 8 Memory Chip

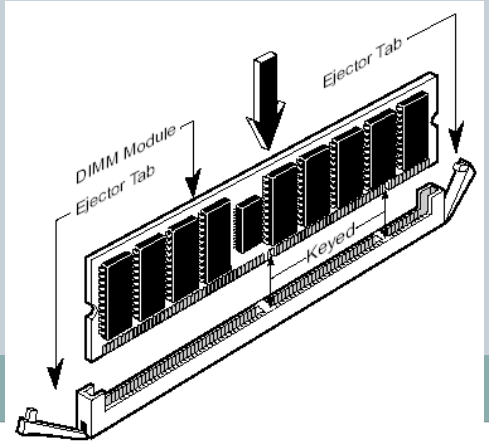
- Organization of a 2M x 32 memory module using 512K x 8 static memory chips
- 2M x 32 ➔ 2 x 2^{10} x 2^{10} x 32 ➔ 2^{21} x 32
- No. of 512K x 8 required = (2^{21} x 32) /(2^{9} x 2^{10} x 8)= 16 nos.

Larger Memories Using Multiple Chips



Larger Memories Using Multiple Chips

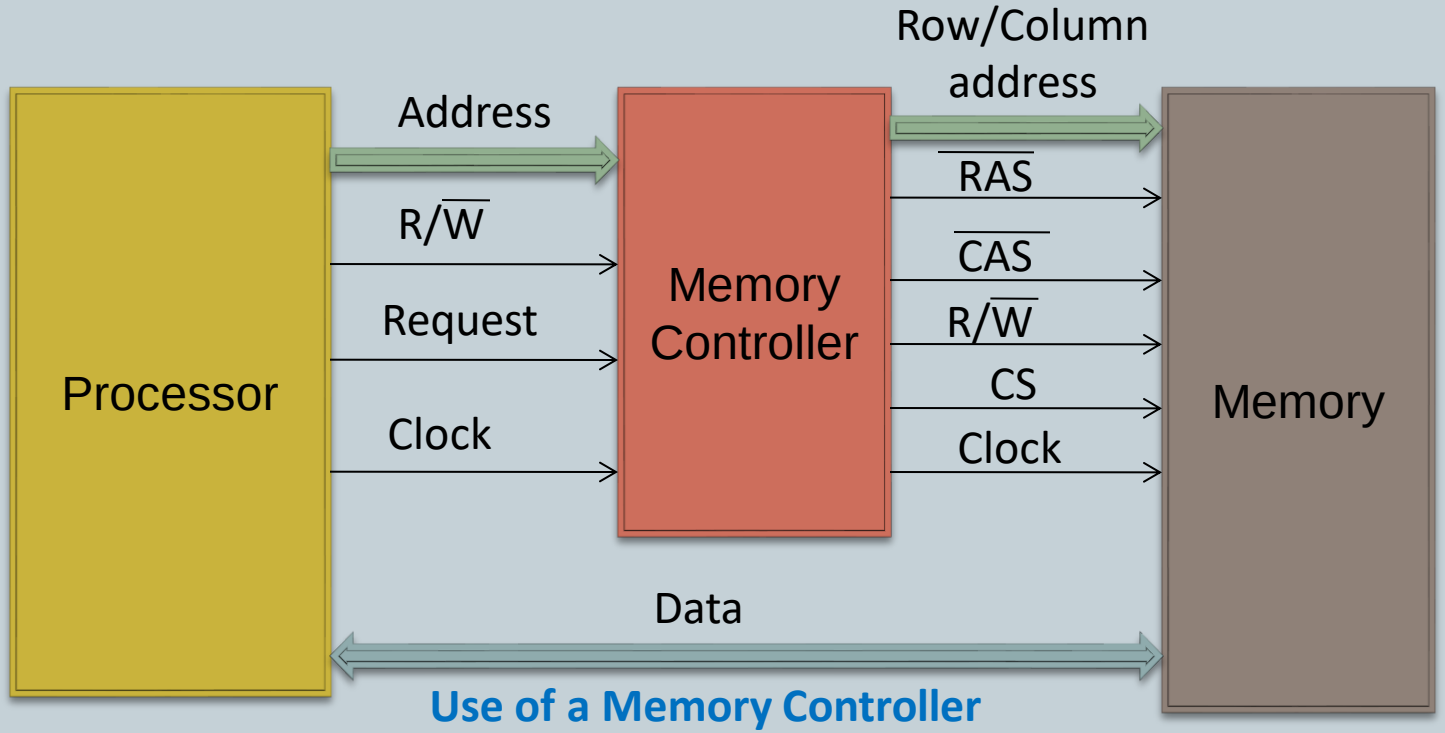
- Assembly of several memory chips on a separate small board (PCB) that plugs vertically into a single socket on the motherboard called as Memory Modules
  - SIMMs – Single In-line Memory Modules
  - DIMMs – Dual In-line Memory Modules



Memory System Considerations

- The choice of a RAM chip for a given application depends on several factors
  - Cost,
  - Speed
  - Power dissipation and
  - Size of the chip
- Static RAMs are generally used only when very fast operation is the primary requirement
  - They are used mostly in cache memories
- Dynamic RAMs are the predominant choice for implementing computer main memories

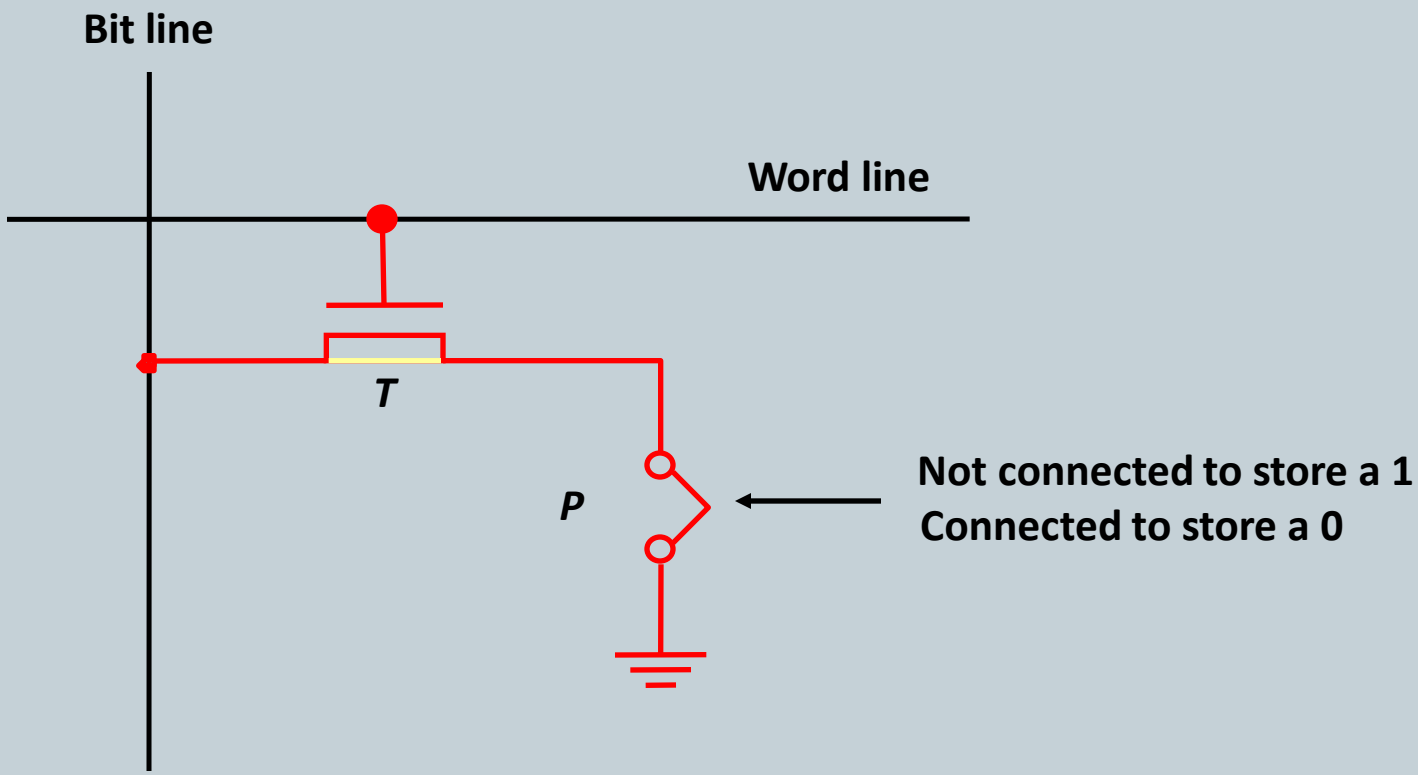
Memory Controller



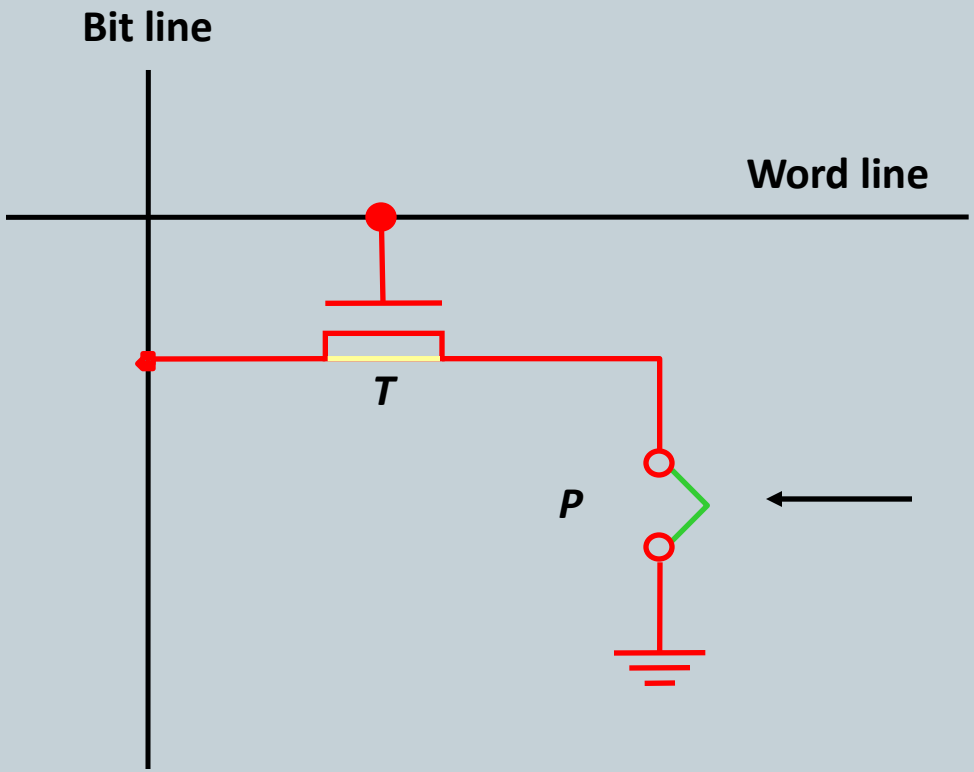
Use of a Memory Controller

- Memory controller provides the refresh control if not done on the chip
  - Refreshing typically once every 64ms. At a cost of .2ms
  - Less than .4% overhead

ROM : Read-Only-Memory



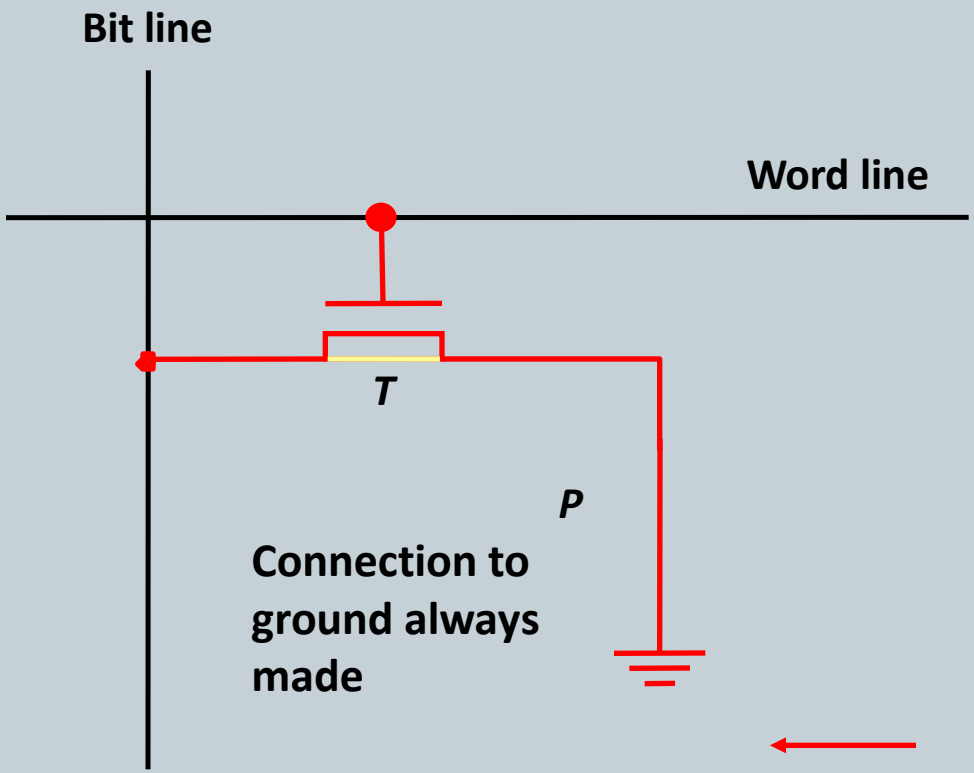
PROM: Programmable Read Only Memory



Manufactured connected (storing 0), but the connection is a "fuse" and can be burned out with a high current to change it to a 1

A PROM cell.

EPROM: Erasable Read Only Memory



Transistor can have a charge put into it that causes it to remain permanently open (programmed to be a 1)

Can be erased with ultraviolet light

An EPROM cell.

Flash Memory

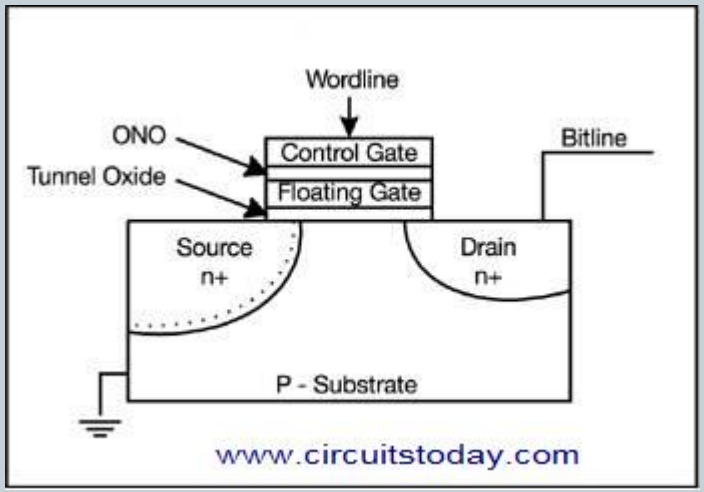
- Flash memory is a type of electrically erasable programmable read-only memory [EEPROM] chip that can be used for the transfer and permanent storage of digital data.
- It is actually a long-term persistent storage computer storage device that can be electrically erased and reprogrammed.

Working of Flash Memory

- The information is stored as an array of floating gate transistors called cells.
- Each of these cells can store only one bit of information at a time.
- The design of each memory cell is somewhat similar to that of a MOSFET.
- The only difference is that this cell has two gates.
- The gate on top is called the control gate [CG], and the bottom one is called the floating gate [FG].
- FG may be made of conductive materials like poly silicon or can also be non-conductive.
- These two gates are separated from each other by a thin oxide layer.

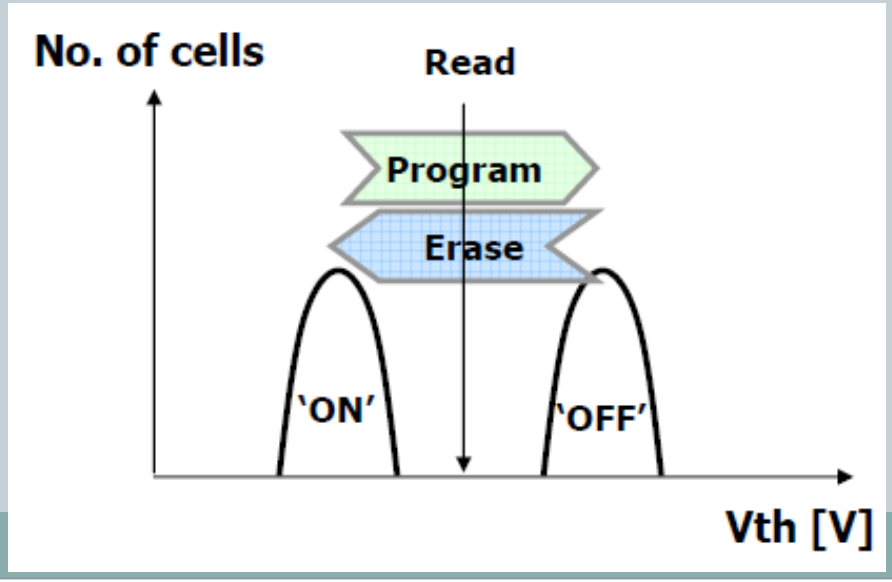
Working of Flash Memory

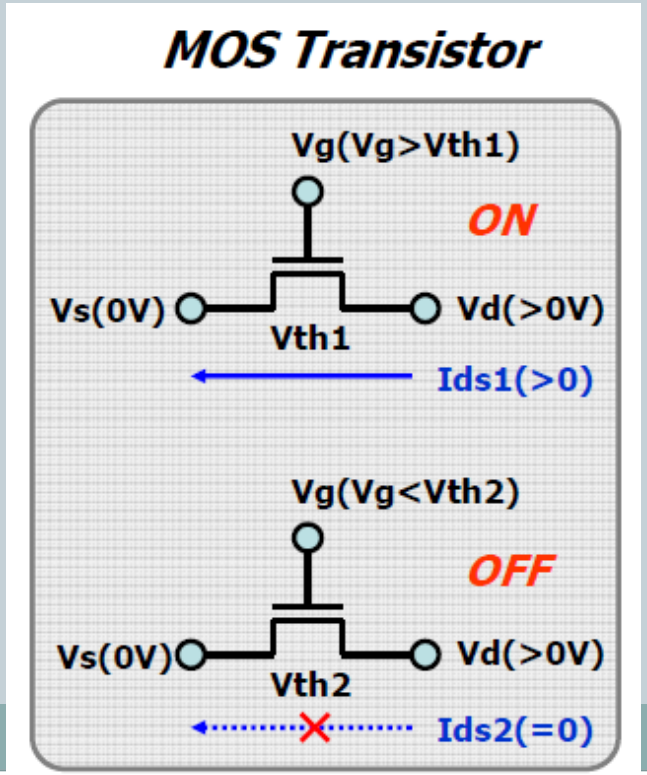
- The FG can come in contact with the word line only through CG.
- When that particular link is closed, the cell will have a value ‘1’.
- Inorder to change the value to ‘0’, a process called tunnelling has to be done.
- This charge cancels the electric field from CG, and thus causes to modify the threshold voltage [V] of the cell.



Flash Cell

- Write & read binary data to a flash cell
  - data ‘0’ ‘OFF’ state (program)
  - data ‘1’ ‘ON’ state (erase) Vt



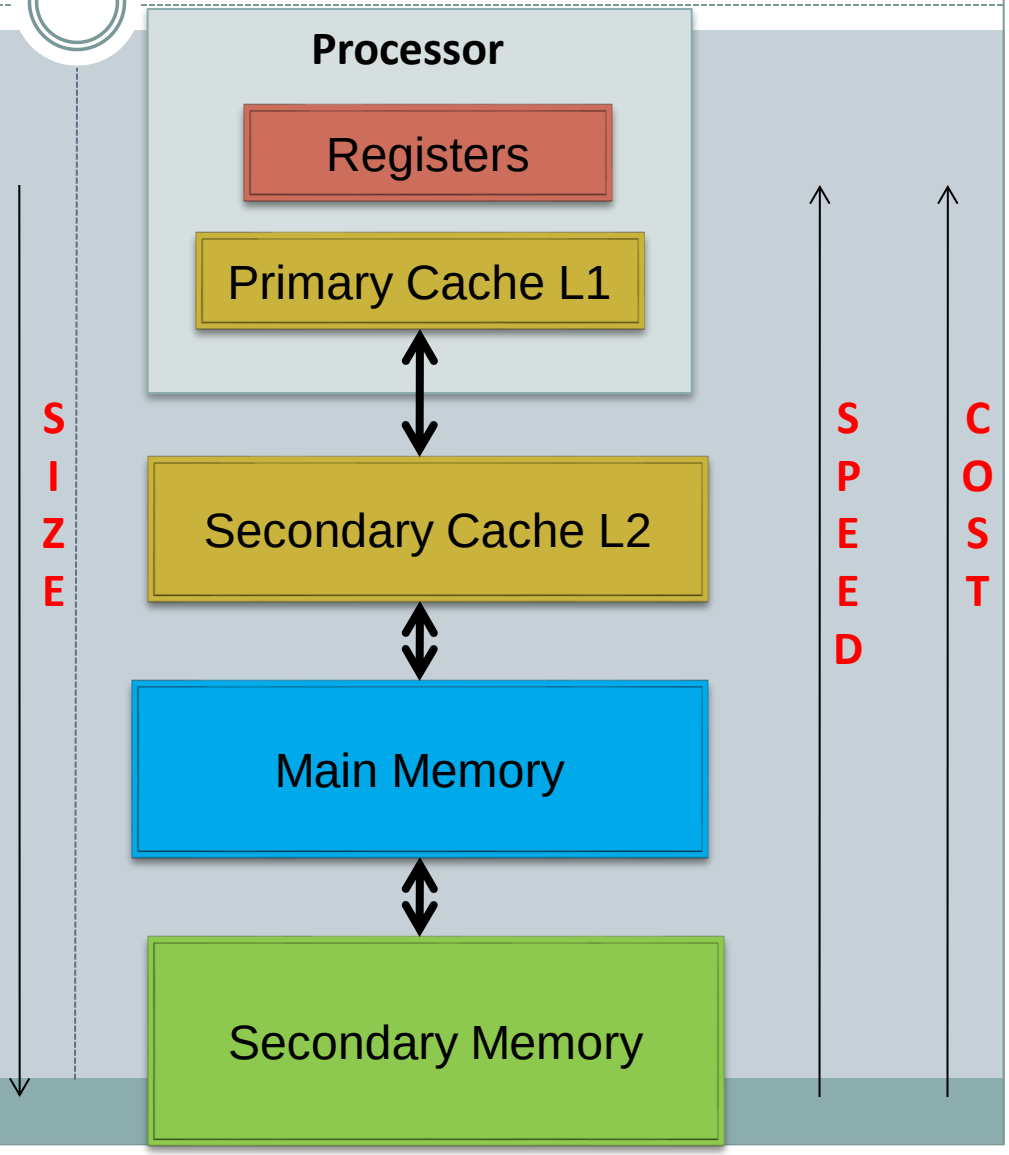


EEPROM & Flash Memory

- EEPROM: Electrically Erasable PROM
  - Cells erasable selectively
  - while EPROM, erase all (using UV Light)
- Flash Memory
  - Similar to EEPROM--each cell a single transistor with a “trapped” charge
  - Read individual cells, write in blocks
  - Greater density, low power consumption, small, cheap
  - Can substitute for disks (up to a gigabyte?)
  - higher cost, but portable

Memory Hierarchy

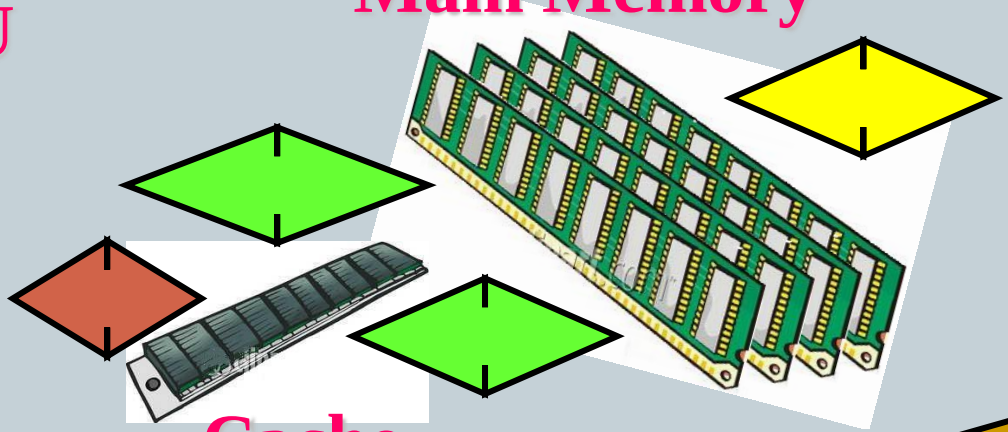
- Memory hierarchy is to obtain the highest possible access speed
- while minimizing the total cost of the memory system



Memory Hierarchy

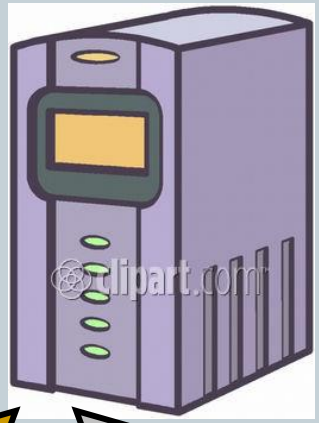
CPU

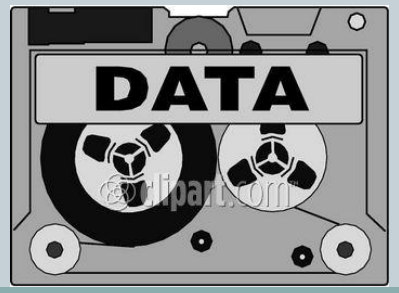
Main Memory



Cache

I/O Processor





Magnetic Disks

Cache Memories

- Main memory (still) slow in comparison to processor speed
- Main memory constrained by packaging, electronic characteristics and costs
- Cache memory on the processor chip typically ten times faster than main memory

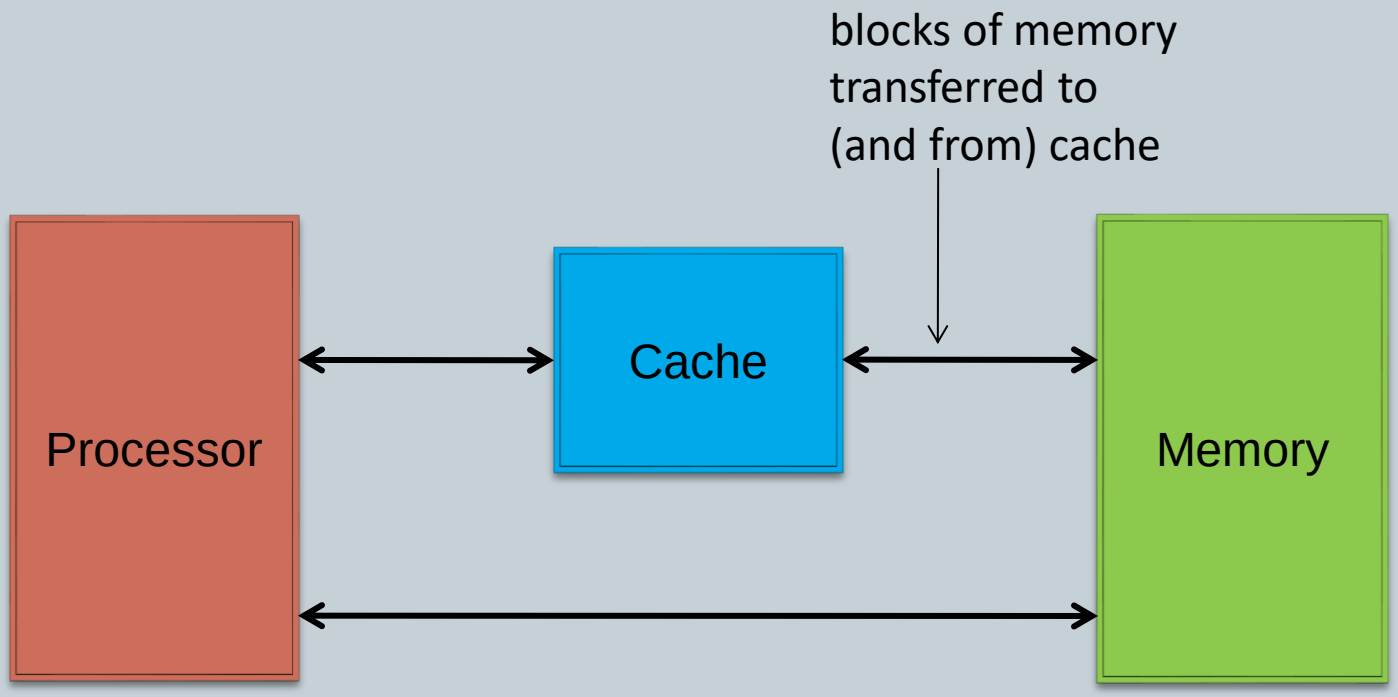
Locality Reference

- Programs tend to spend their time “focused” on particular groups of instructions
  - Loops
  - Frequently called procedures
- “Localized” areas of programs executed repeatedly during some time period
- Much (most?) of program not accessed during some time period

Locality Reference

- Temporal
  - Recently executed instruction likely to repeat soon
  - When first accessed, move to cache where it will be when referenced again
- Spatial
  - Instructions near an executed instruction likely to be executed soon
  - When fetching an instruction from memory, move its neighbors into cache as well

Use of a Cache Memory

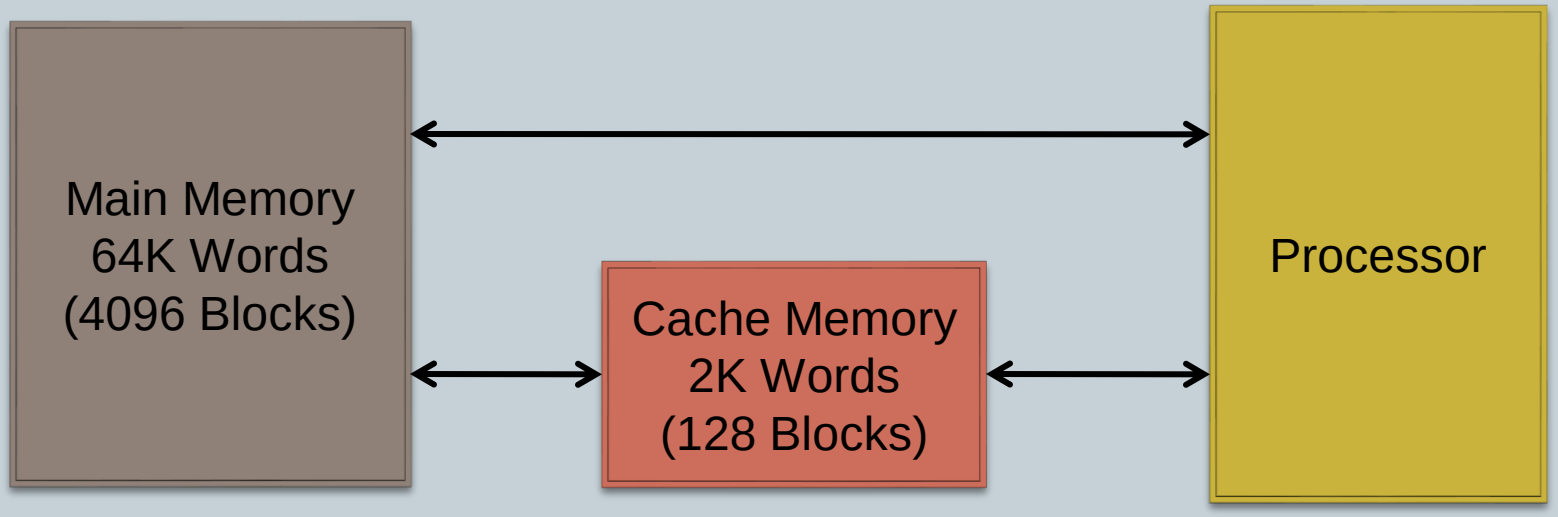


- Processor accesses instructions and data in the cache if there (is a “hit”), if not (is a “miss”)

Cache Memory

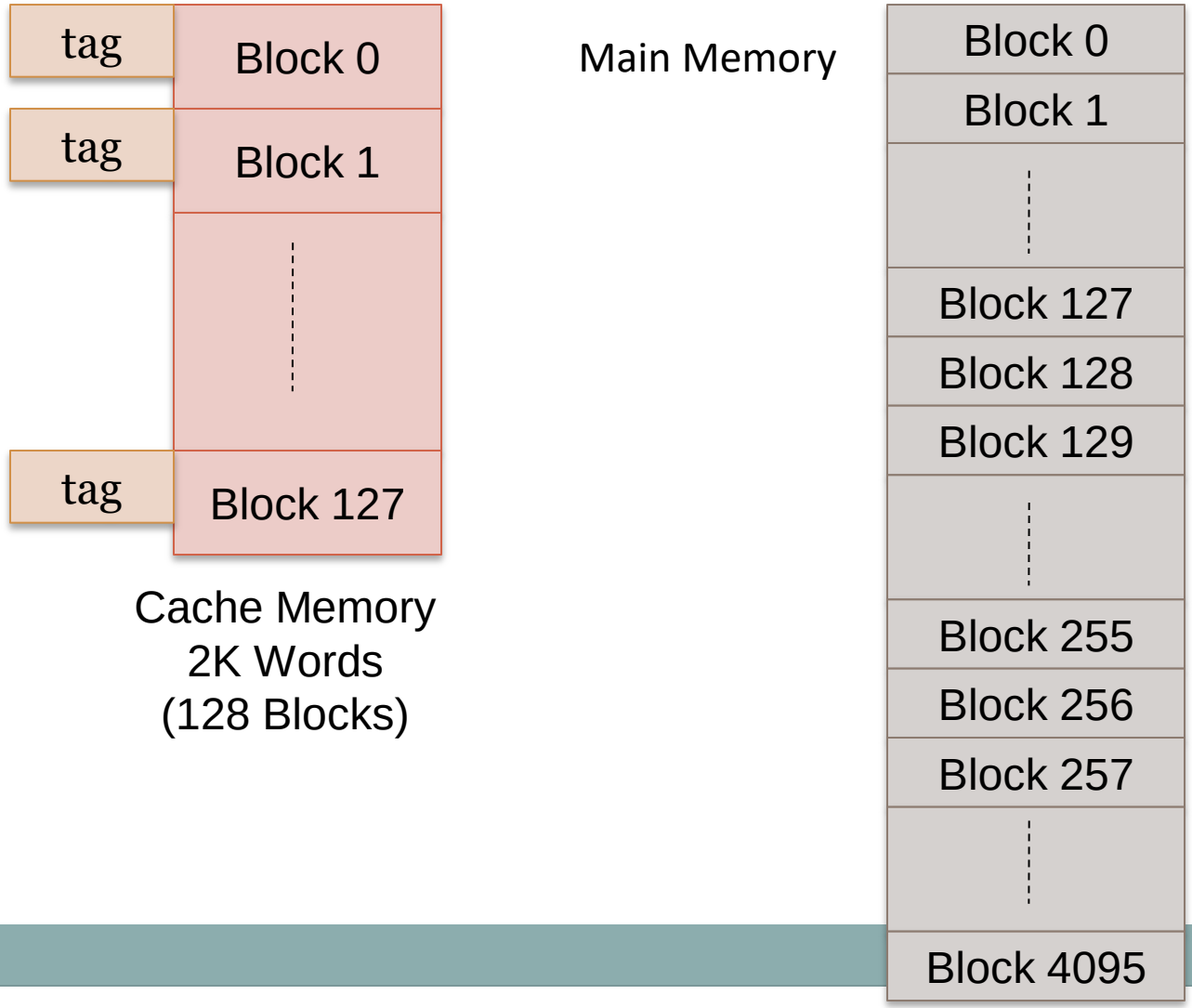
- Lets consider a **cache** consisting of 128 blocks of 16 words each, for total 2048 (2K)words
  - 128 x 16 words = 2048 words
  - Assume each block size is 16 words
- Lets consider the **main memory** is addressable by a 16-bit address.
  - 16 bit address ➔ 64 K words can be stored ➔ 4 K blocks can be stored (since each block size is 16 words)
- Main memory blocks can be modeled many way on to the cache
  - Cache is Content Addressable Memory (CAM)

Cache Memory



- Each block consists of 16 words

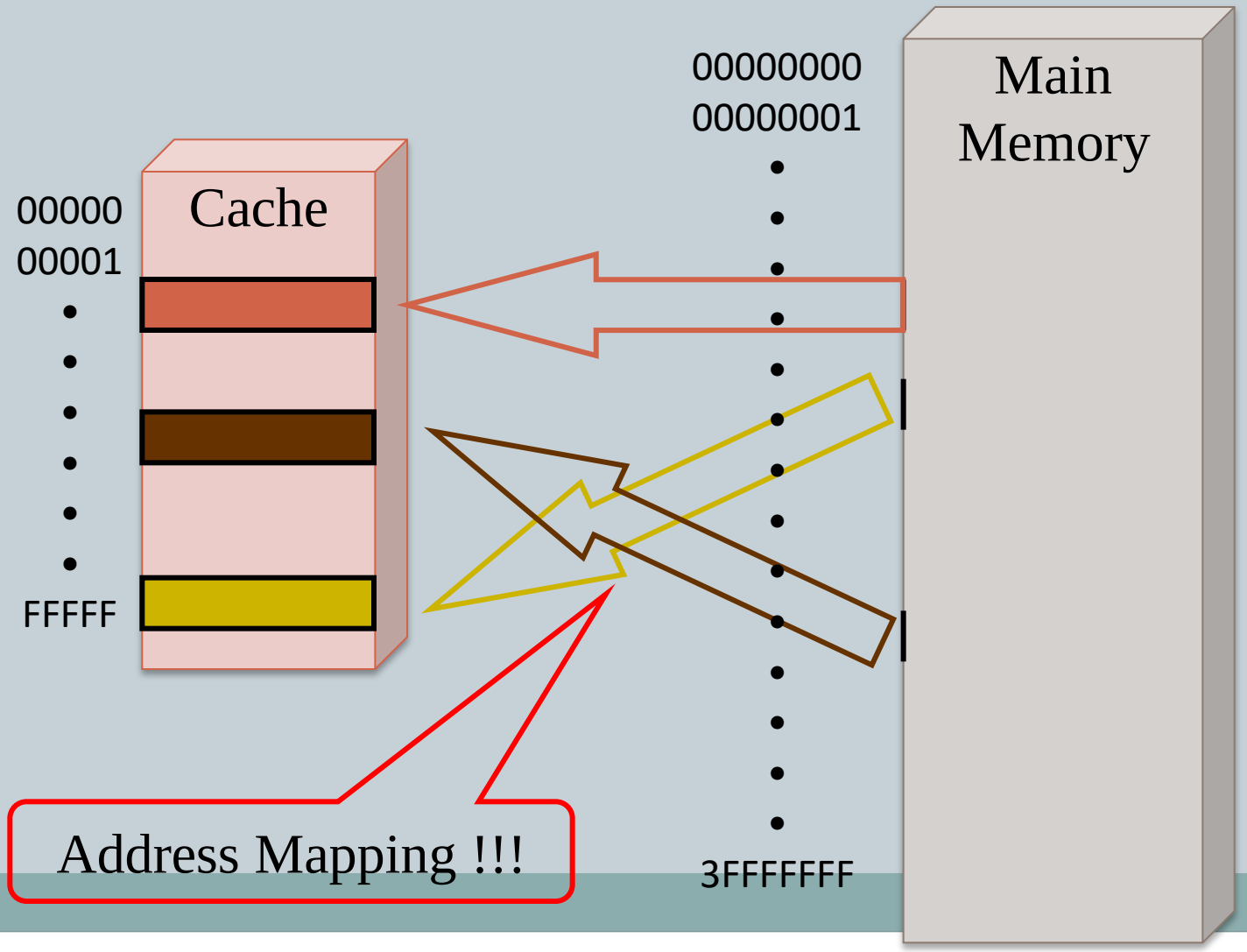
Cache and Main Memory



Cache Management

- Mapping
  - Determination of where in the cache the blocks (cache lines) of main memory are to be placed
- Replacement
  - Determination of when to replace a block in cache with another block of main memory
- Coherency
  - Assurance that no problems arise from cache version differing from main memory version

Cache Mapping



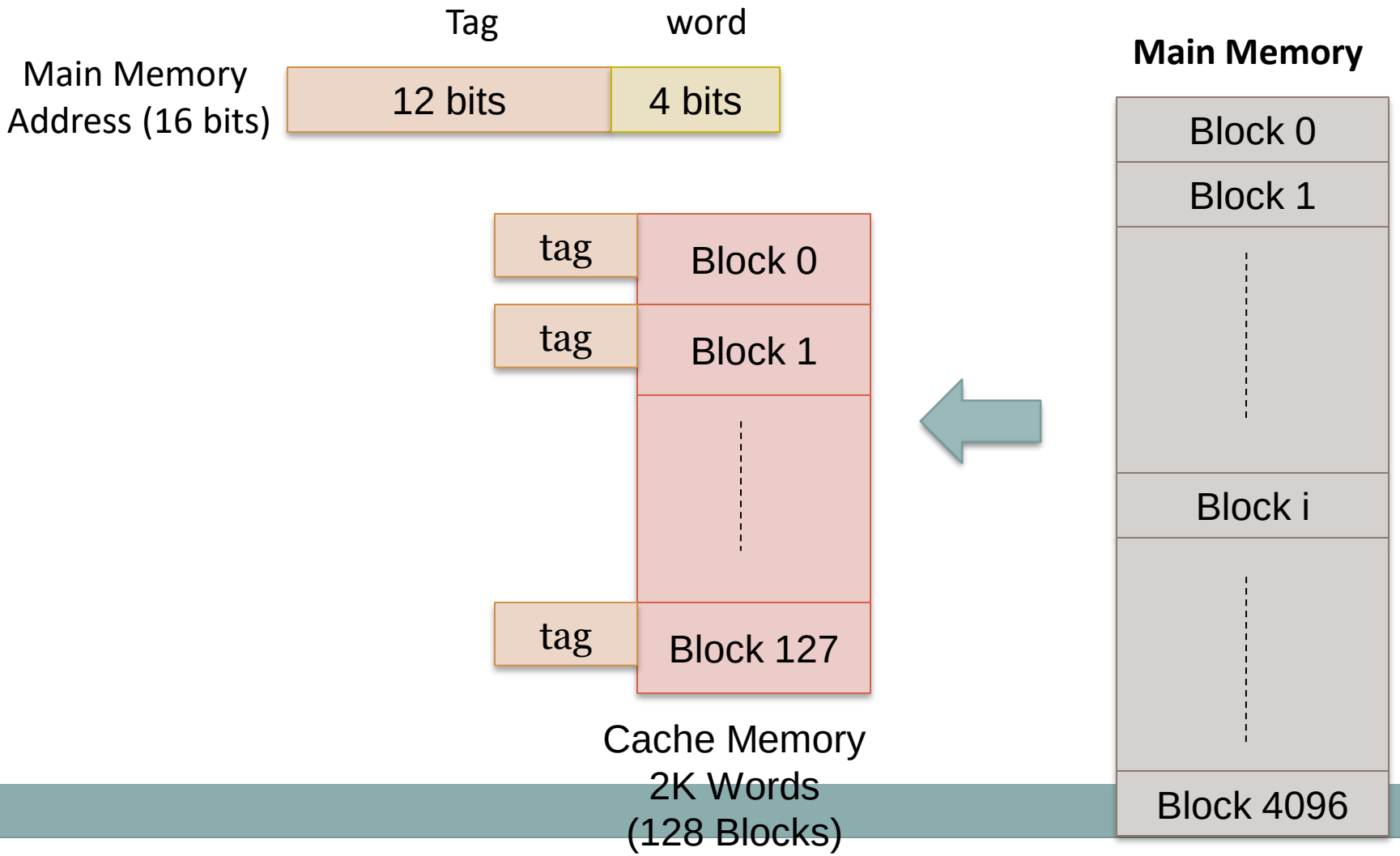
Memory and Cache mapping

- Mapping function
  - Specification of correspondence between main memory blocks and cache blocks
    - Associative mapping
    - Direct Mapping
    - Set- associative mapping

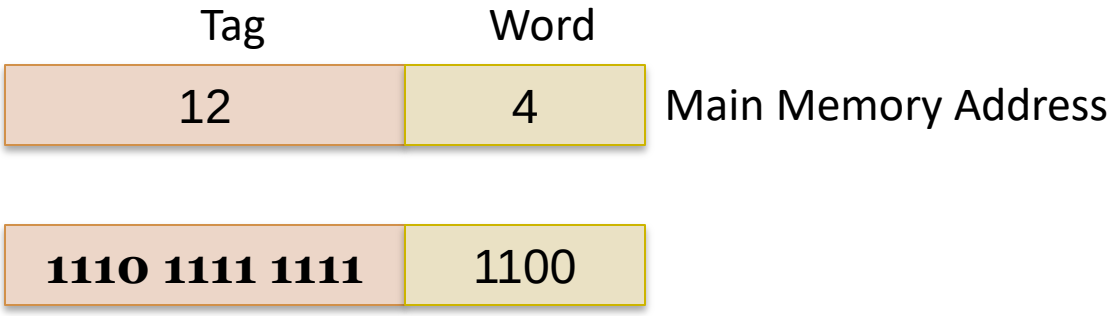
Associative Mapping

- Any block location in cache can store any block in memory
  - Most flexible
- Mapping table is implemented in an associative memory
  - Fast, very expensive
- Mapping table stores both address and the content of the memory word

Associative Mapping

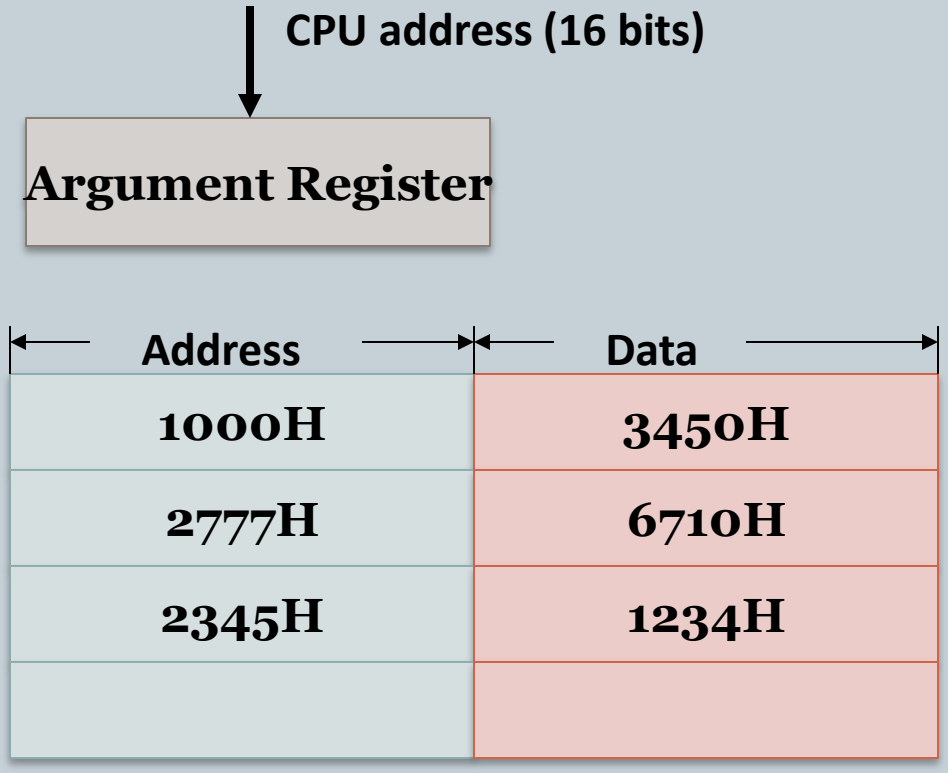


Associative Mapping

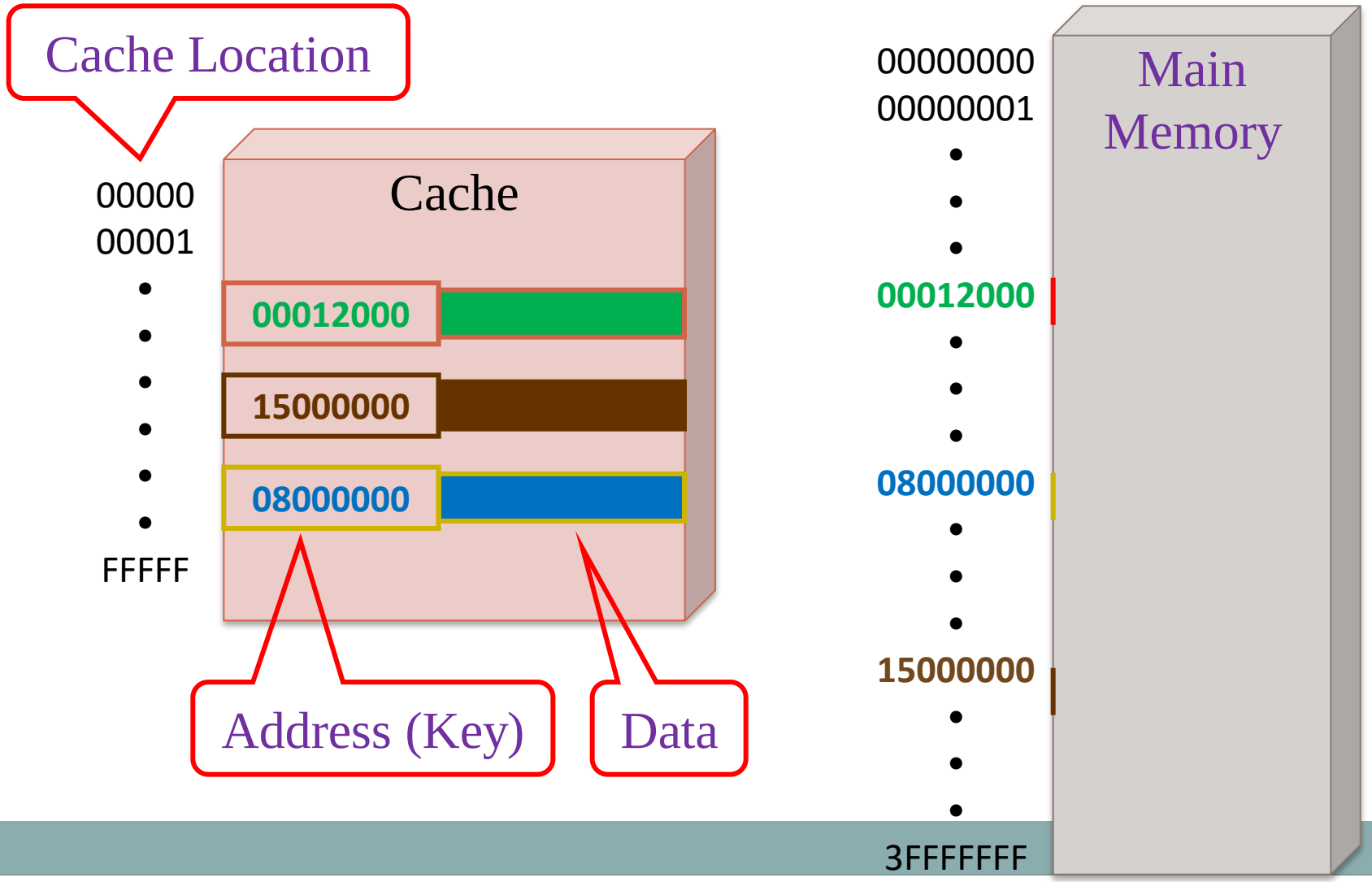


- Tag: 111011111111
- Word:1100=12, the 12th word of a block in the cache

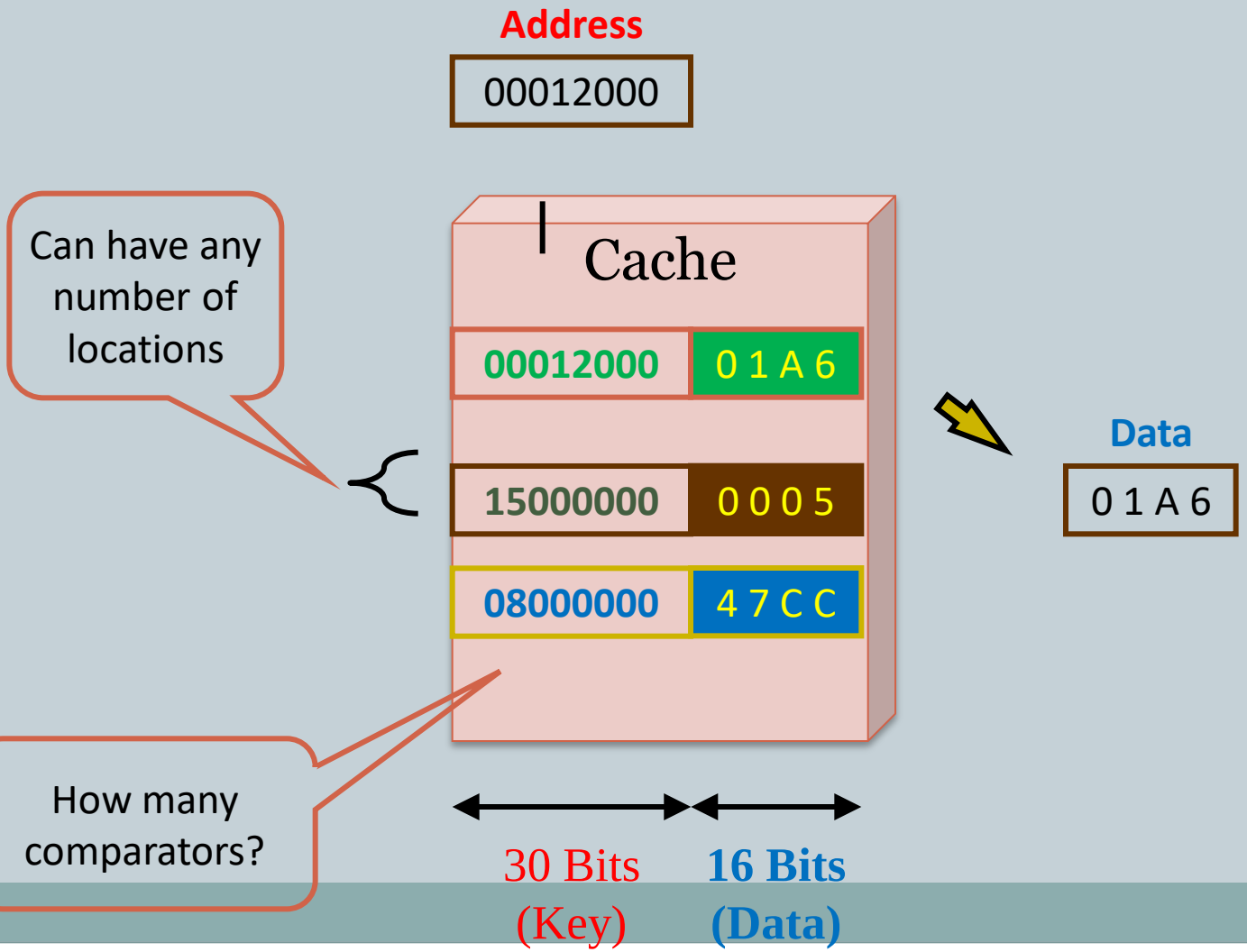
Associative Mapping Example



Associative Mapping



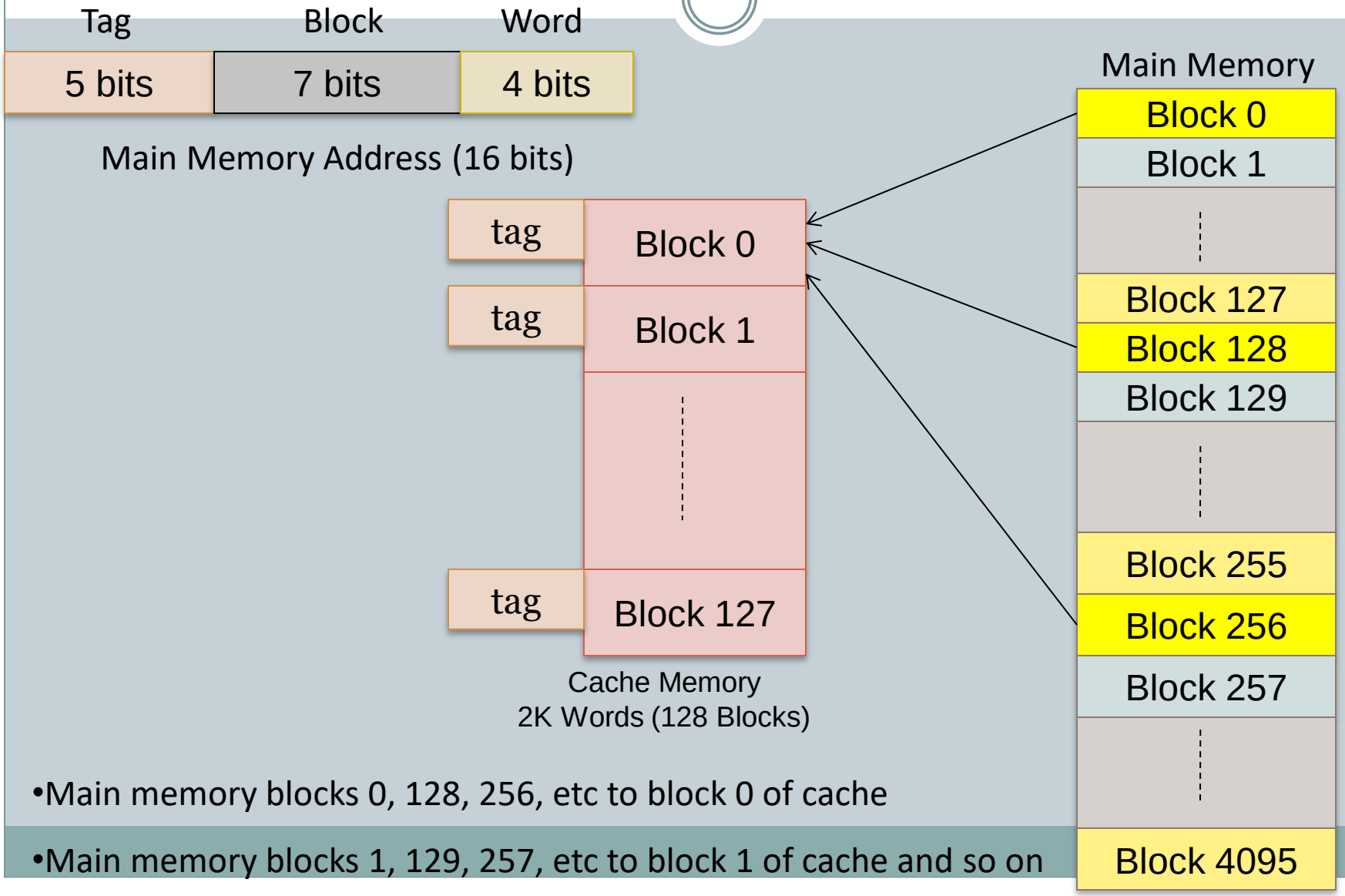
Associative Mapping



Direct Mapping

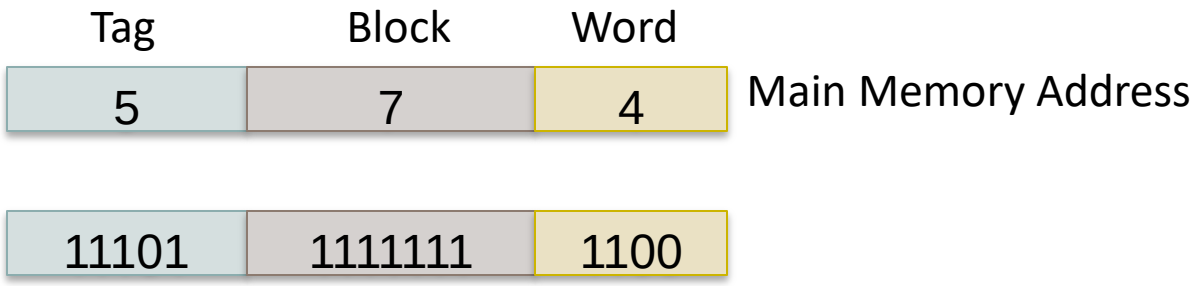
- Each memory block has only one place to load in Cache
- n-bit addresses are used to access main memory and k-bit index is used to access the cache
- Mapping function : j modulo 128 where 128 is the block size in cache

Direct Mapping



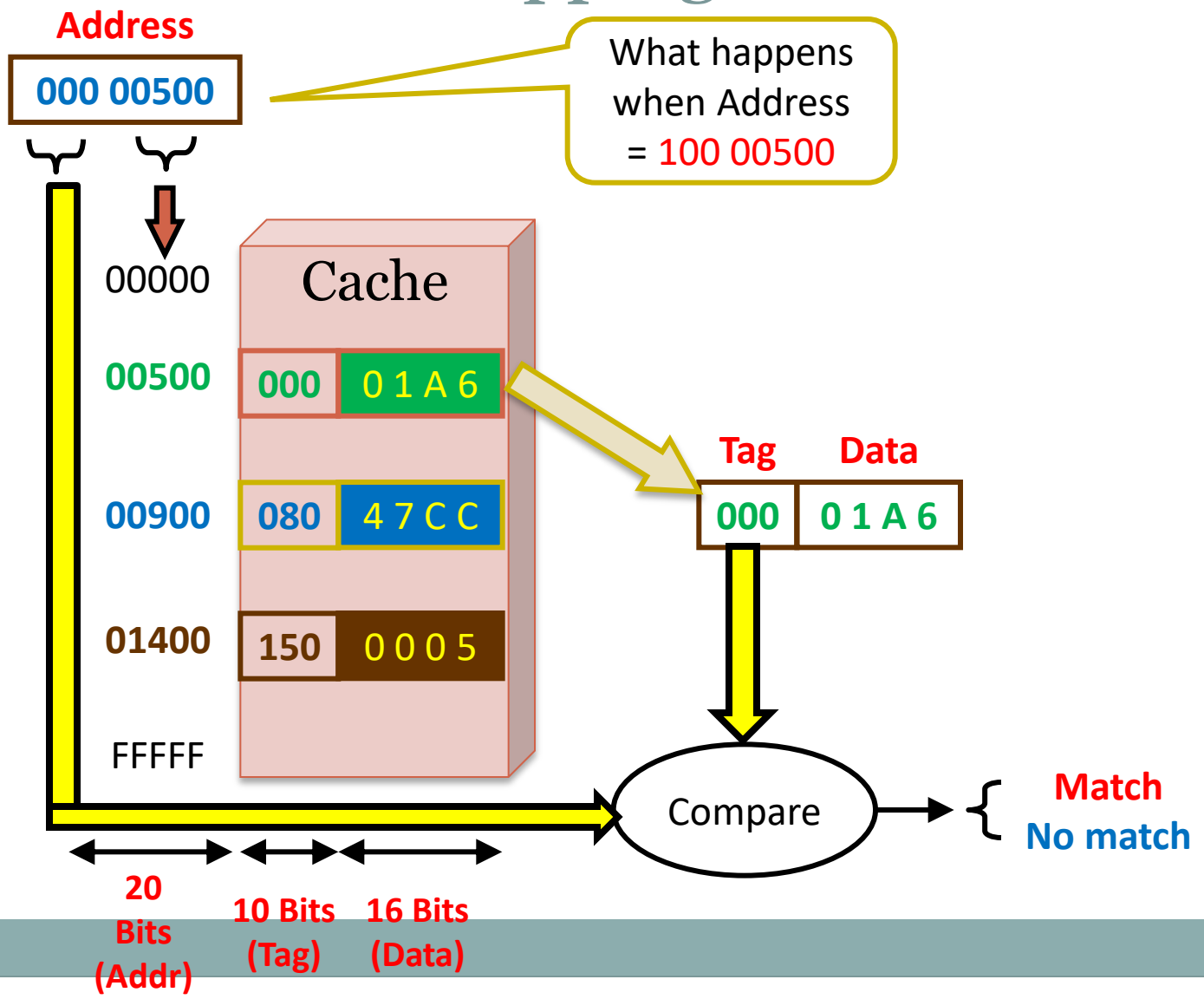
- Main memory blocks 0, 128, 256, etc to block 0 of cache
- Main memory blocks 1, 129, 257, etc to block 1 of cache and so on

Direct Mapping

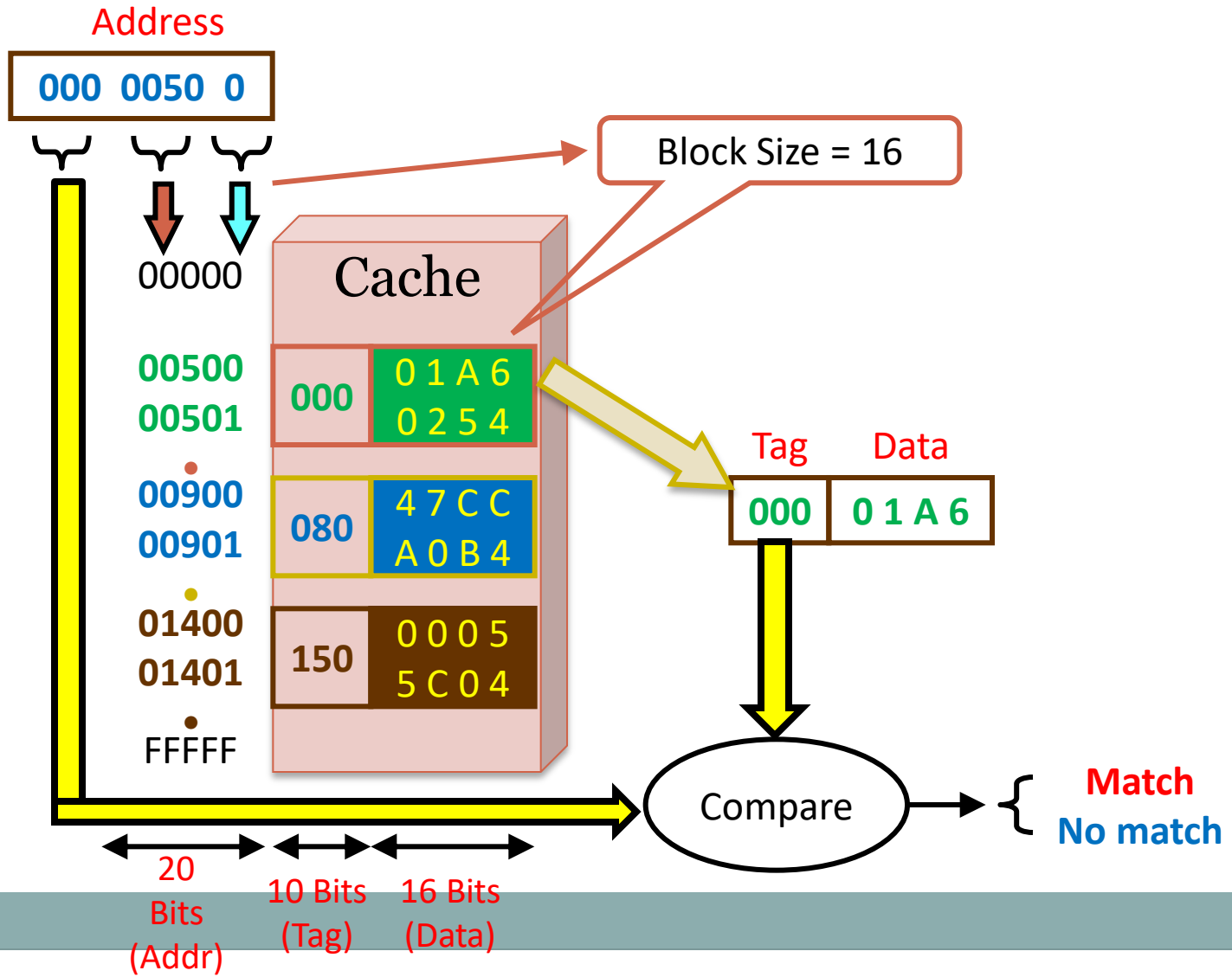


- Tag: 11101
- Block: 1111111=127, in the 127th block of the cache
- Word:1100=12, the 12th word of the 127th block in the cache

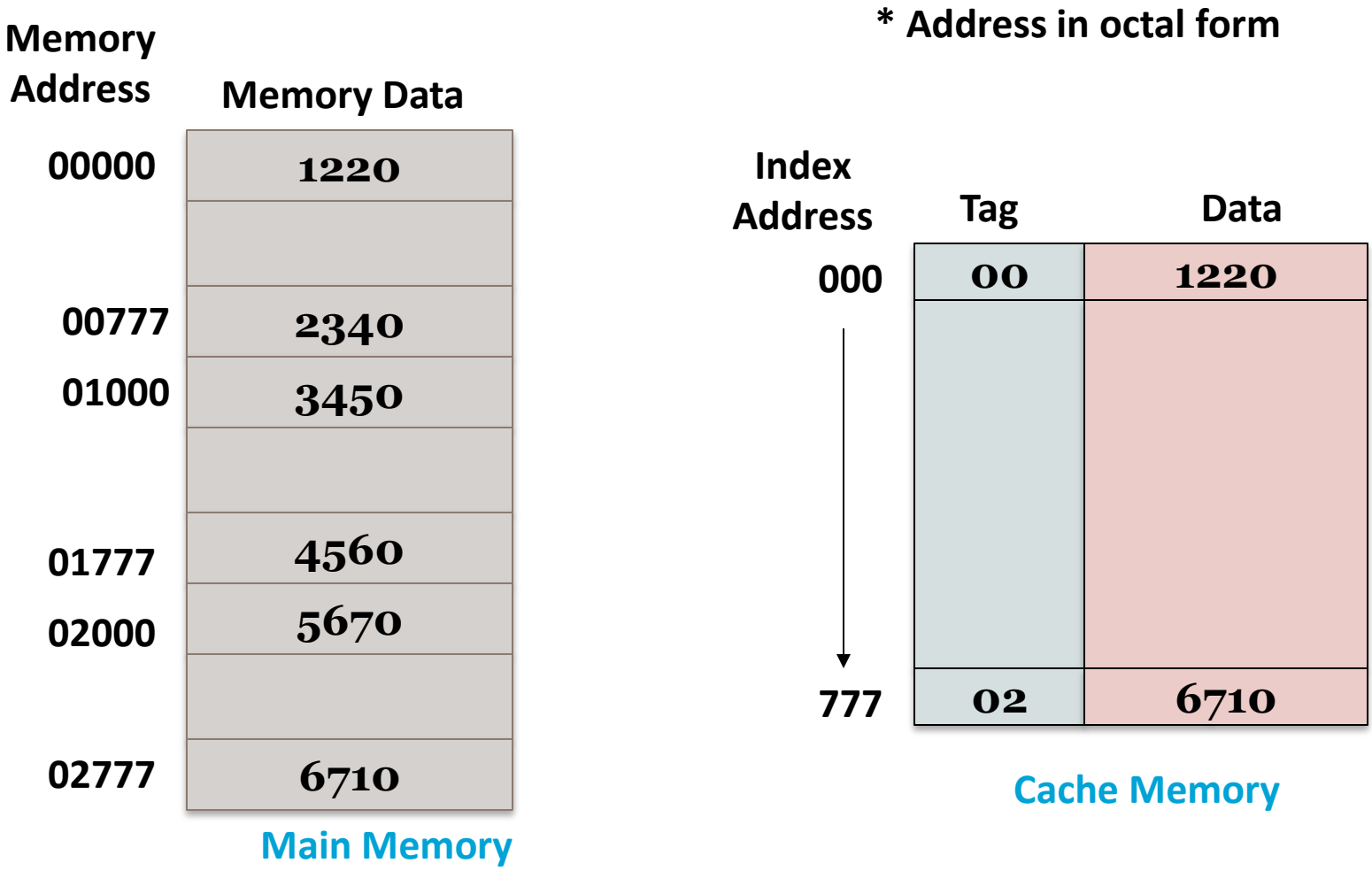
Direct Mapping



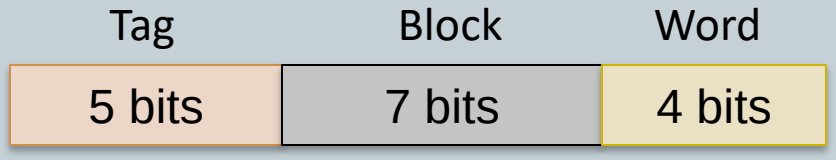
Direct Mapping with Blocks



Direct Mapped Cache Example



Direct Mapping with Block Example



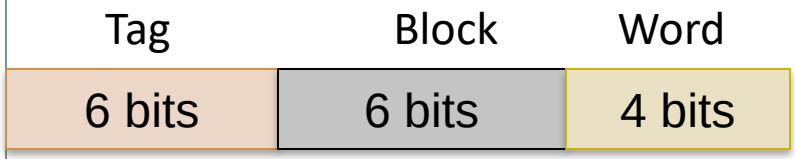
Index

	Index	Tag	Data
Block 0	000	01	3450
	00F	01	6578
Block 1	010		
	01F		
Block 127	7F0	02	
	7FF	02	6710

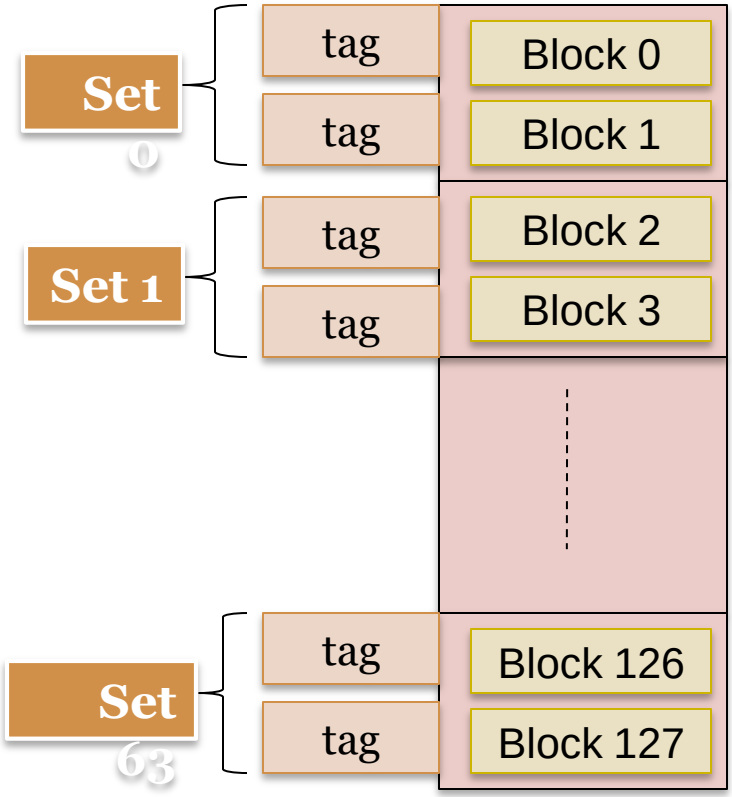
Set Associative Mapping

- A combination of direct and associative mapping techniques can be used
- Blocks of the cache are grouped into sets and the mapping allows a block of the main memory to reside in any block of a specific set

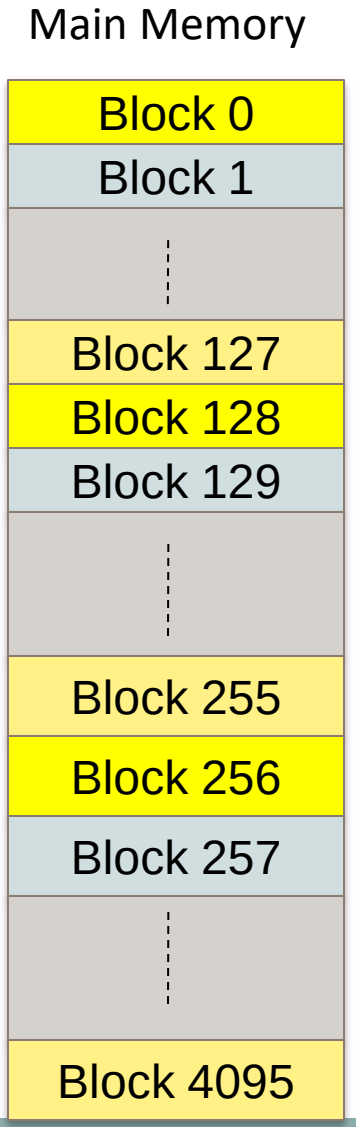
Set Assocoative Mapping Example



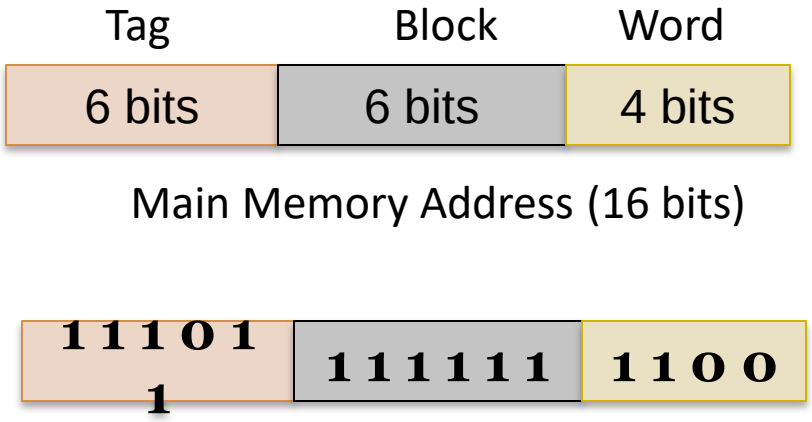
Main Memory Address (16 bits)



Cache Memory 2K Words (128 Blocks)



Set-Associative Mapping



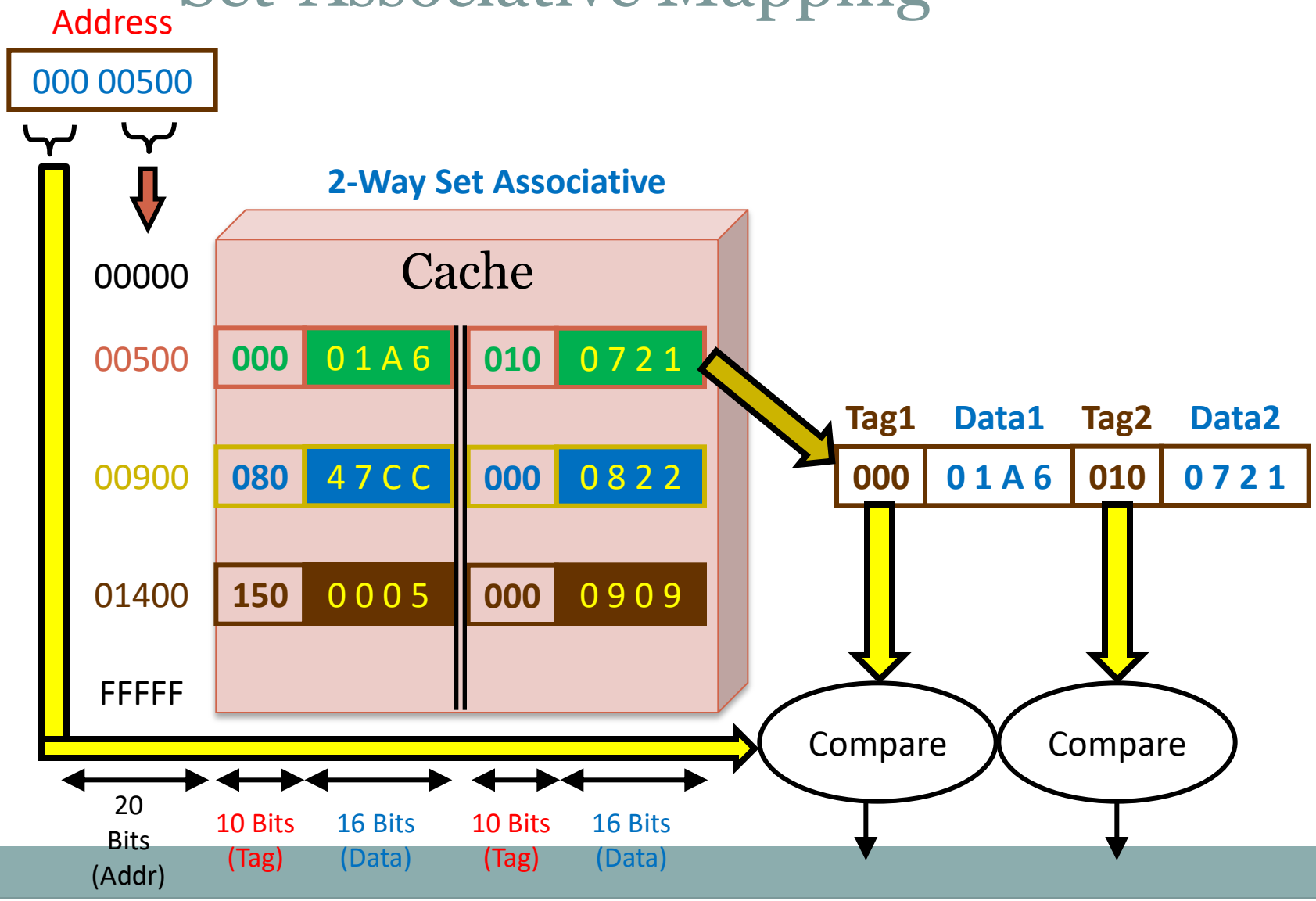
- Tag: 111011
- Set: 111111=63, in the 63th set of the cache
- Word:1100=12, the 12th word of the 63th set in the cache

Set Associative Mapping Example

- Each memory block has a set of locations in the cache to load
- Set associative mapping cache with set size of two

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Set-Associative Mapping



Replacement Algorithms

- Difficult to determine which blocks to kick out
- Least Recently Used (LRU) block
- The cache controller tracks references to all blocks as computation proceeds
- Increase / clear track counters when a hit/miss occurs

Replacement Algorithms

- For Associative & Set-Associative Cache
  - Which location should be emptied when the cache is full and a miss occurs?
  - First In First Out (FIFO)
  - Least Recently Used (LRU)
- Distinguish an *Empty* location from a *Full* one
  - Valid Bit

Cache Write

- Write Through
  - When writing into memory
    - If hit both cache and Main memory is written in parallel
    - If Miss, Memory is written
      - For a read miss, missing block may be overloaded onto a cache block
  - Memory is always updated
    - Important when CPU and DMA I/O are both executing
    - Slow – due to the memory access time

Cache Write

- Write Back (Update Cache only)
  - When writing into Memory
    - If hit, only cache is written
    - If miss, missing block is brought to cache and write into cache
    - For a read miss, candidate block must be written back to the memory
  - Memory is not up-to-dated, i.e. the same item in cache and memory may have different value
  - Mark cache block “dirty” or “modified”
  - Copy it back to main memory when another block needs the cache space

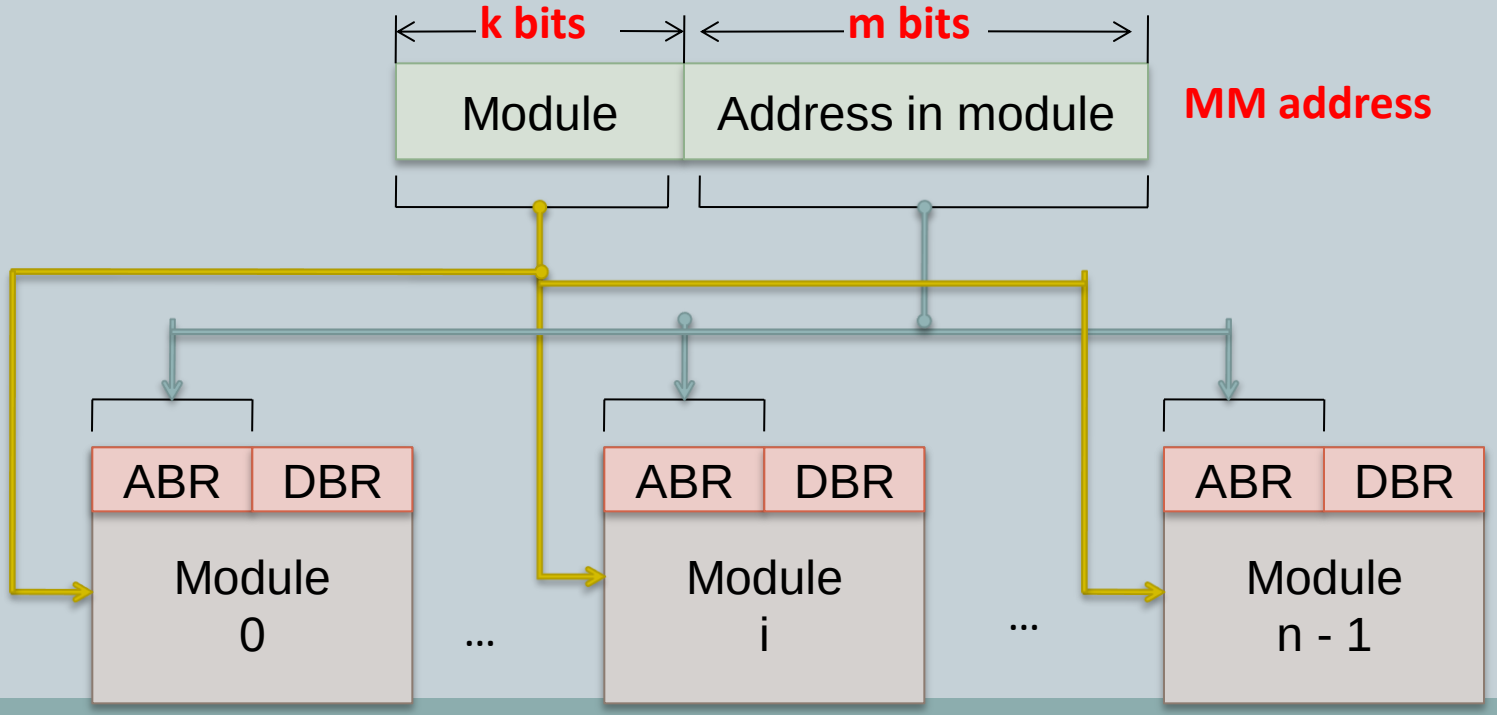
Performance Considerations

Overview

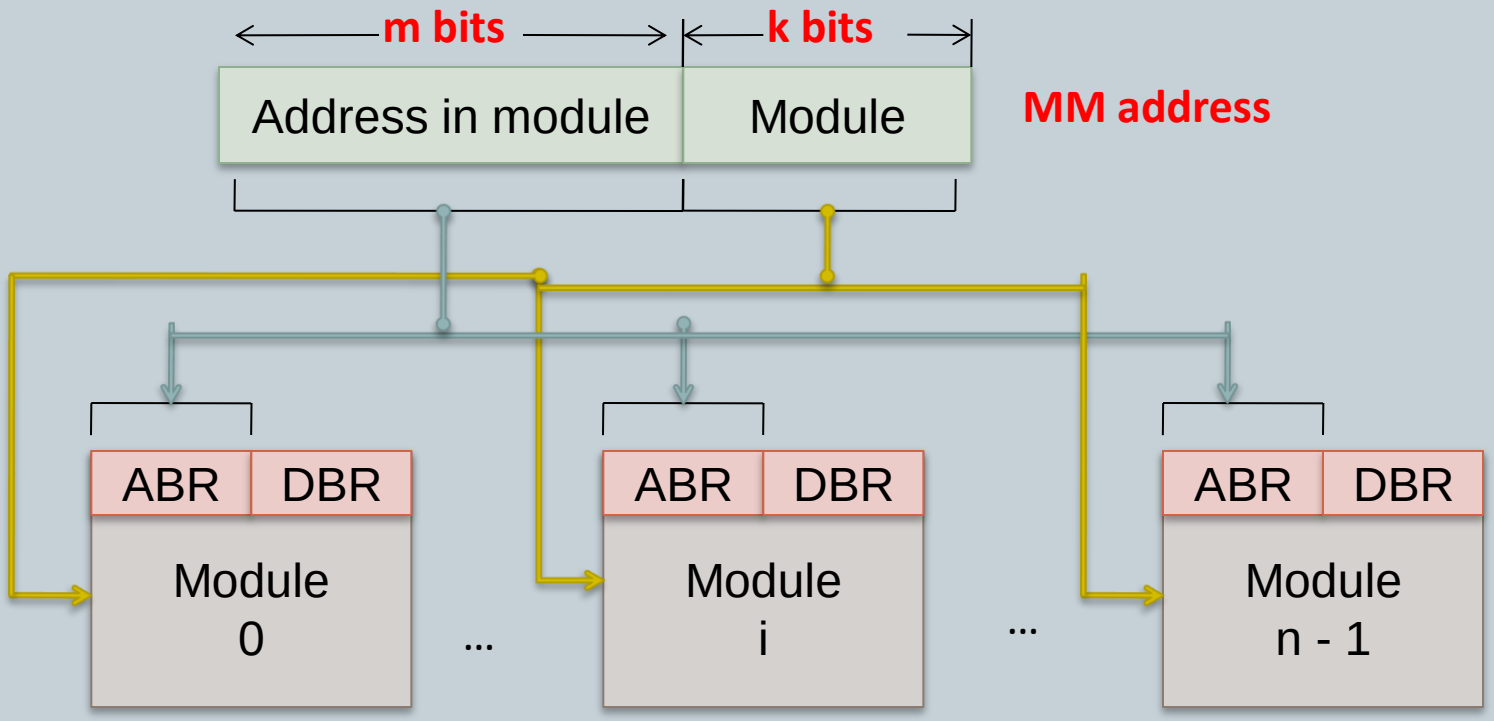
- Two key factors: performance and cost
- Price/performance ratio
- Performance depends on how fast machine instructions can be brought into the processor for execution and how fast they can be executed.
- For memory hierarchy, it is beneficial if transfers to and from the faster units can be done at a rate equal to that of the faster unit.
- This is not possible if both the slow and the fast units are accessed in the same manner.
  - However, it can be achieved when parallelism is used in the organizations of the slower unit (interleaved Organization).

Interleaving

- If the main memory is structured as a collection of physically separated modules, each with its own ABR (Address buffer register) and DBR(Data buffer register), memory access operations may proceed in more than one module at the same time



Interleaving



Consecutive words in consecutive modules

Hit Rate and Miss Penalty

- The success rate in accessing information at various levels of the memory hierarchy – hit rate / miss rate.
- Ideally, the entire memory hierarchy would appear to the processor as a single memory unit that has the access time of a cache on the processor chip and the size of a magnetic disk – depends on the hit rate (>>0.9).
- A miss causes extra time needed to bring the desired information into the cache.

Hit Rate and Miss Penalty (cont.)

- $T_{ave} = hC + (1-h)\,M$
  - $T_{ave:}$ average access time experienced by the processor
  - h: hit rate
  - M: miss penalty, the time to access information in the main memory
  - C: the time to access information in the cache

How to Improve Hit Rate?

- Use larger cache – increased cost
- Increase the block size while keeping the total cache size constant.
- However, if the block size is too large, some items may not be referenced before the block is replaced – miss penalty increases.

Caches on the Processor Chip

- On chip vs. off chip?
- Two separate caches for instructions and data, respectively
- Single cache for both
- Which one has better hit rate? -- Single cache
- What's the advantage of separating caches? – parallelism, better performance
- Level 1 and Level 2 caches

  L1 cache – faster and smaller. Access more than one word simultaneously and let the processor use them one at a time.

  L2 cache – slower and larger.
- How about the average access time?

  Average access time: $t_{ave} = h_1C_1 + (1-h_1)h_2C_2 + (1-h_1)(1-h_2)M$

  where h is the hit rate, C is the time to access information in cache, M is the time to access information in main memory.

Other Enhancements

- Write buffer – processor doesn't need to wait for the memory write to be completed
- Prefetching – prefetch the data into the cache before they are needed
- Lockup-Free cache – A cache that support multiple outstanding misses is called lockup-free. To keep track of all outstanding misses, cache makes use of processor registers

Performance of cache

- Memory Access
  - All the memory accesses are directed first to cache
  - If the word is in the cache; access cache provide it to CPU
  - If the word is not in cache; bring a block including that word to replace a block now in cache

Virtual Memories

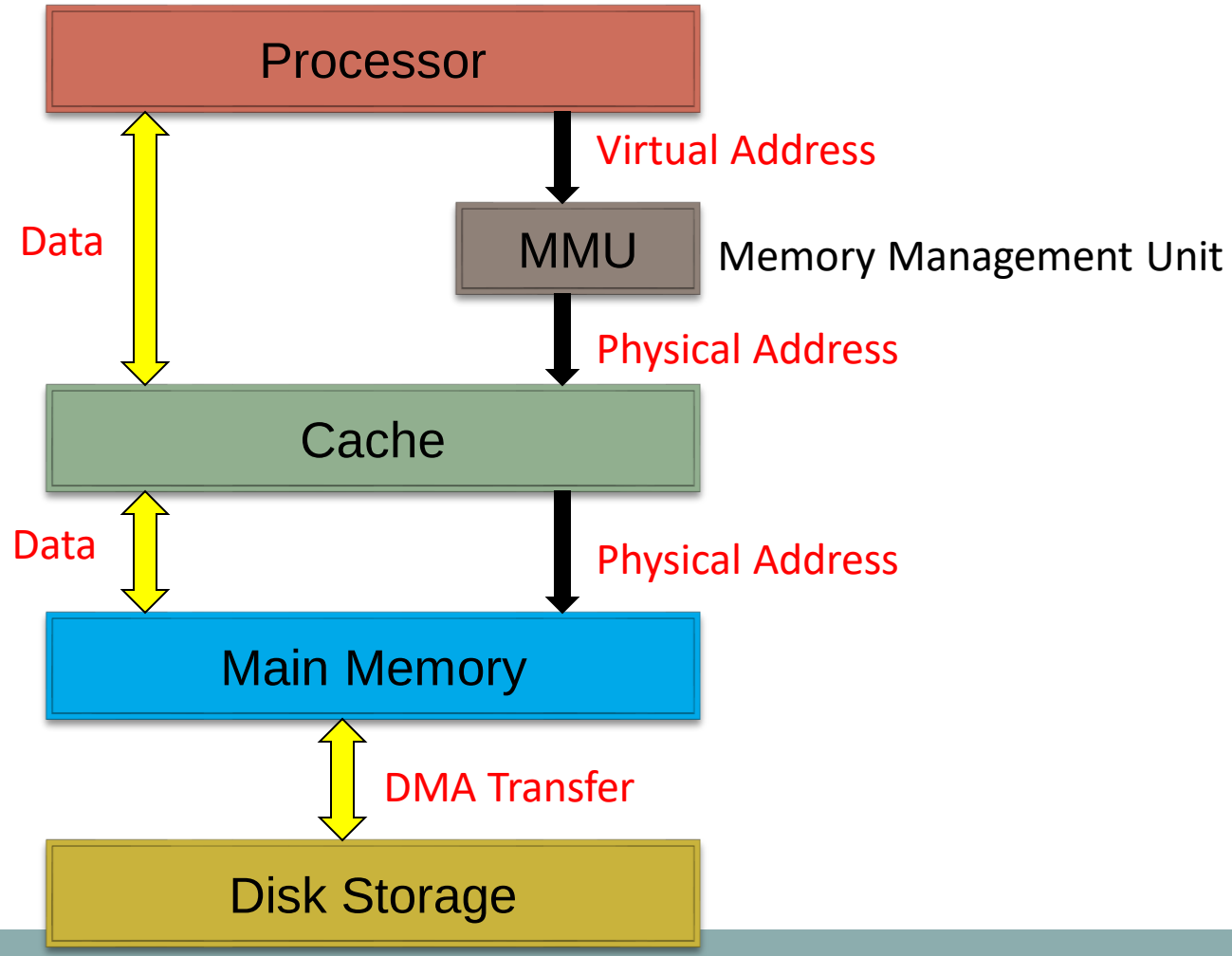
Introduction

- Physical main memory is not as large as the address space spanned by an address issued by the processor.

  2^{32} = 4 GB, 2^{64} = …

- When a program does not completely fit into the main memory, the parts of it not currently being executed are stored on secondary storage devices.
- Techniques that automatically move program and data blocks into the physical main memory when they are required for execution are called virtual-memory techniques.
- Virtual addresses will be translated into physical addresses.

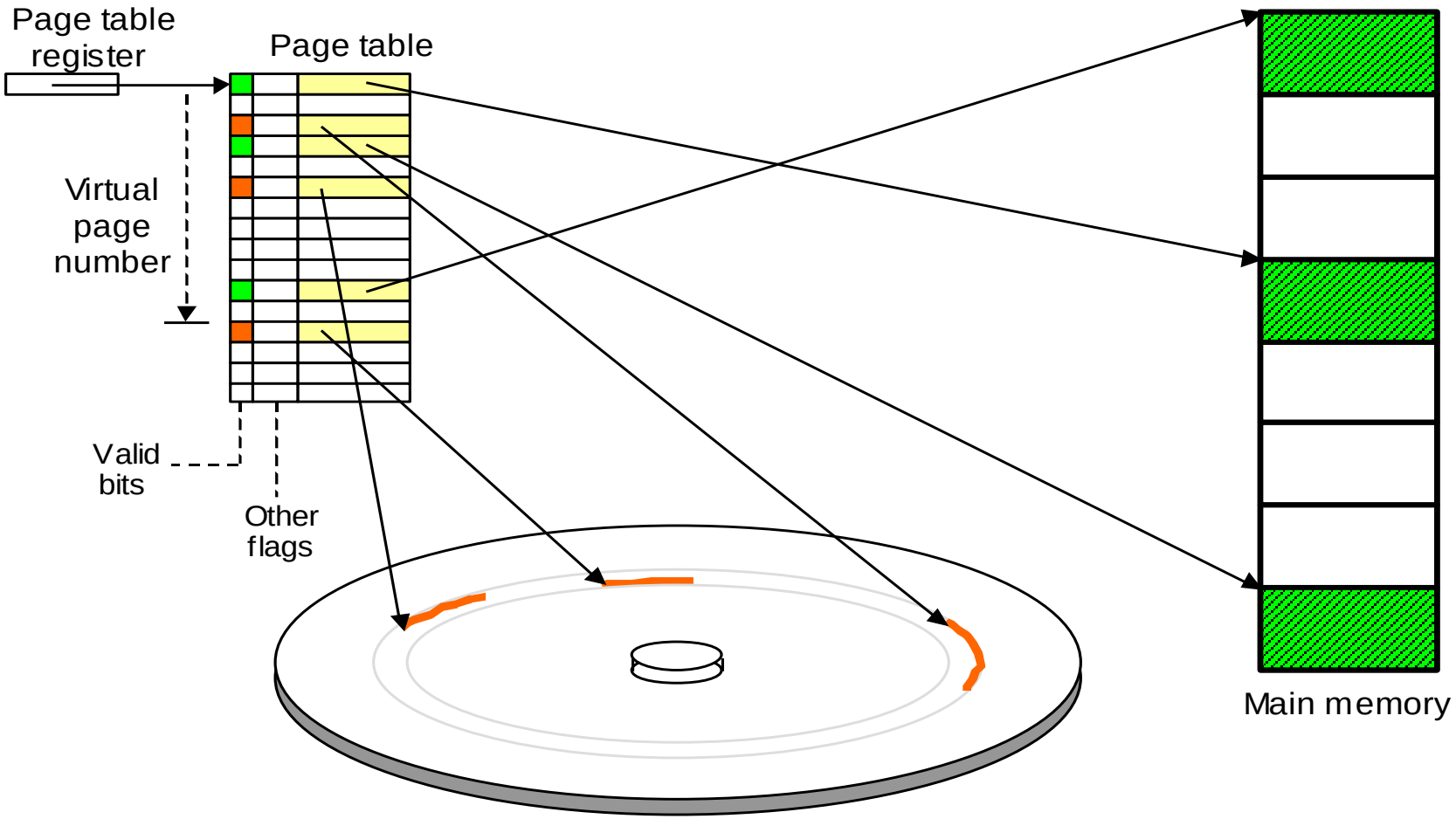
Virtual Memory Organization



Address Translation

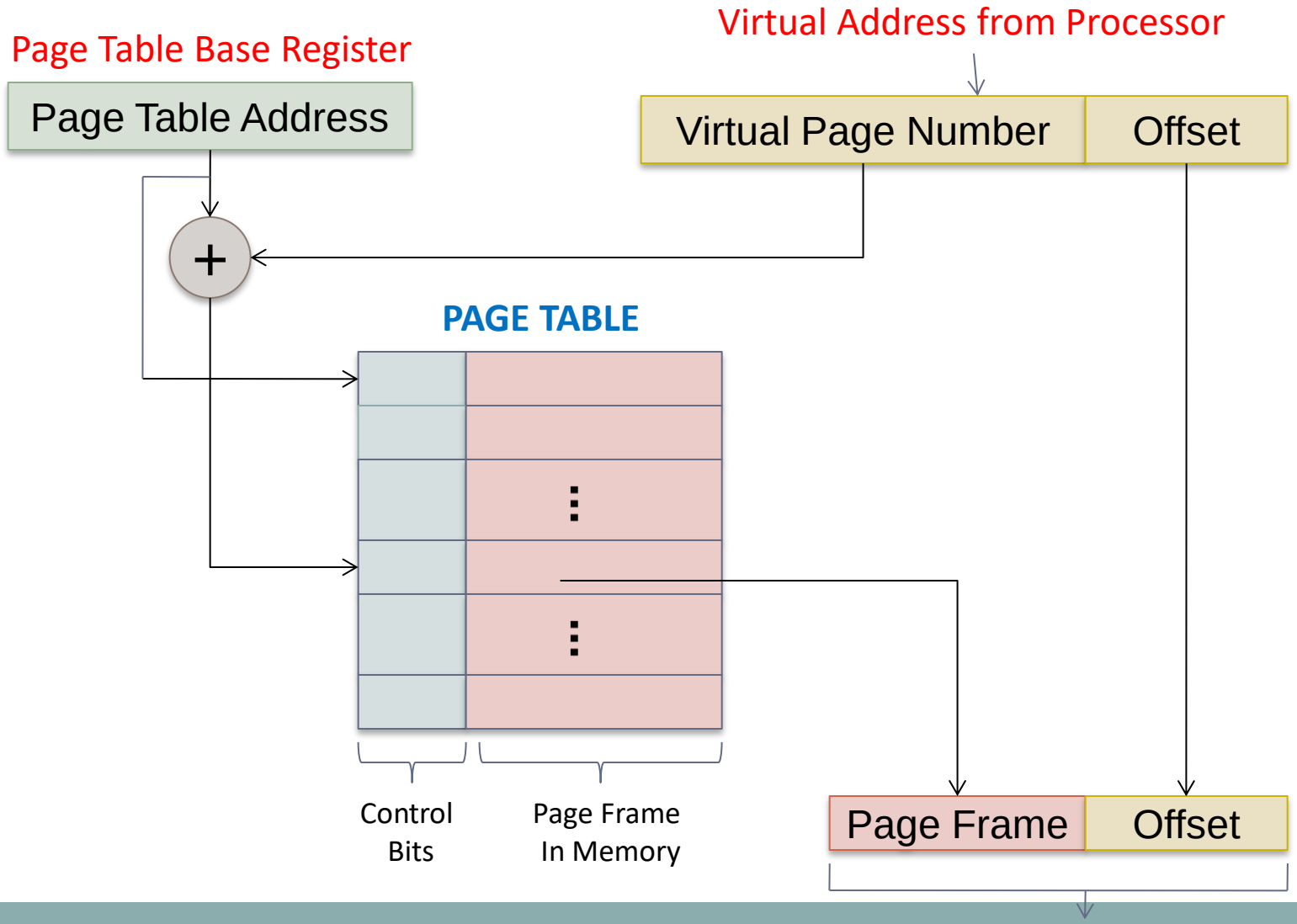
- All programs and data are composed of fixed-length units called pages
  - Each of which consists of a block of words that occupy contiguous locations in the main memory.
- Page cannot be too small or too large.
- The virtual memory mechanism bridges the size and speed gaps between the main memory and secondary storage
  - similar to cache.

Page Tables and Address Translation



The role of page table in the virtual-to-physical address translation process.

Address Translation



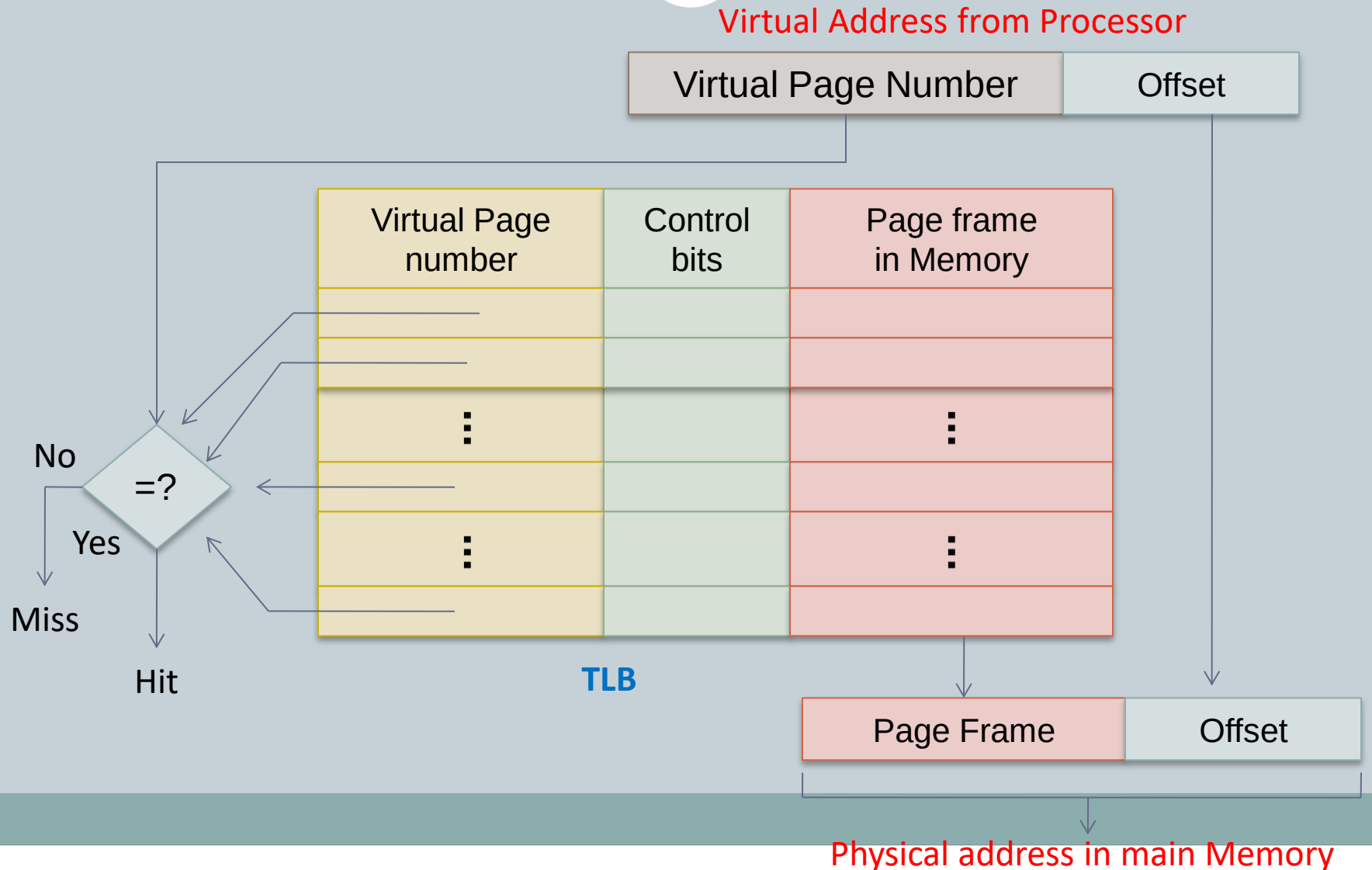
Address Translation

- The page table information is used by the MMU for every access, so it is supposed to be with the MMU.
- However, since MMU is on the processor chip and the page table is rather large, only small portion of it, which consists of the page table entries that correspond to the most recently accessed pages, can be accommodated within the MMU
- It is impossible to include a complete page table on this chip
- Therefore, page table is kept in the Main Memory
  - A copy of a small portion of the page table can be accommodated within the MMU

TLB

- This portion is consists of the page table entries that correspond to the most recently accessed pages
  - A small cache, usually called the Translation Lookaside Buffer (TLB) incorporated into the MMU for this purpose

Translation Lookaside Buffer (TLB)



Translation Lookaside Buffer (TLB)

- The contents of TLB must be coherent with the contents of page tables in the memory
- Translation procedure
- Page fault
  - An access request to the page that is not there in the Main Memory
- Page replacement
- Write-through is not suitable for virtual memory
- Locality of reference in virtual memory

Memory Management Requirements

- Multiple programs
- System space / user space
- Protection (supervisor / user state, privileged instructions)
- Shared pages appear in both user space and system space

Reliability of Memory Systems

- **Error Detecting Systems:**

  It is the process of encountering errors, resulting from operational deficiencies
or noise.

- **Error Detecting Code (EDC):**

  A code that is considered to detect an error in data,

Outline

- The need for error detection and correction
- How simple parity check can be used to detect error
- How two-dimensional parity check extends error detection capability
- How checksum is used to detect error
- How cyclic redundancy check works
- How Hamming code is used to correct error

Introduction

- Environmental interference and physical defects in the communication medium can cause random bit errors during data transmission.
- Error coding is a method of detecting and correcting these errors to ensure information is transferred intact from its source to its destination.

Introduction

- Different error coding schemes are chosen
  - depending on the types of errors expected,
  - the communication medium's expected error rate,
  - and whether or not data retransmission is possible.
- Faster processors and better communications technology make more complex coding schemes,
  - with better error detecting and correcting capabilities,
  - possible for smaller embedded systems,
  - allowing for more robust communications.

Introduction

- However, tradeoffs between bandwidth and coding overhead, coding complexity and allowable coding delay between transmissions, must be considered for each application.

Introduction

- Two basic strategies for dealing with errors
  - One way is
    - to include enough redundant information (extra bits are introduced into the data stream at the transmitter on a regular and logical basis)
    - along with each block of data sent to enable the receiver to deduce what the transmitted character must have been
  - Other way is
  - to include only enough redundancy to allow the receiver to deduce that error has occurred, but not which error has occurred and the receiver asks for a retransmission

Introduction

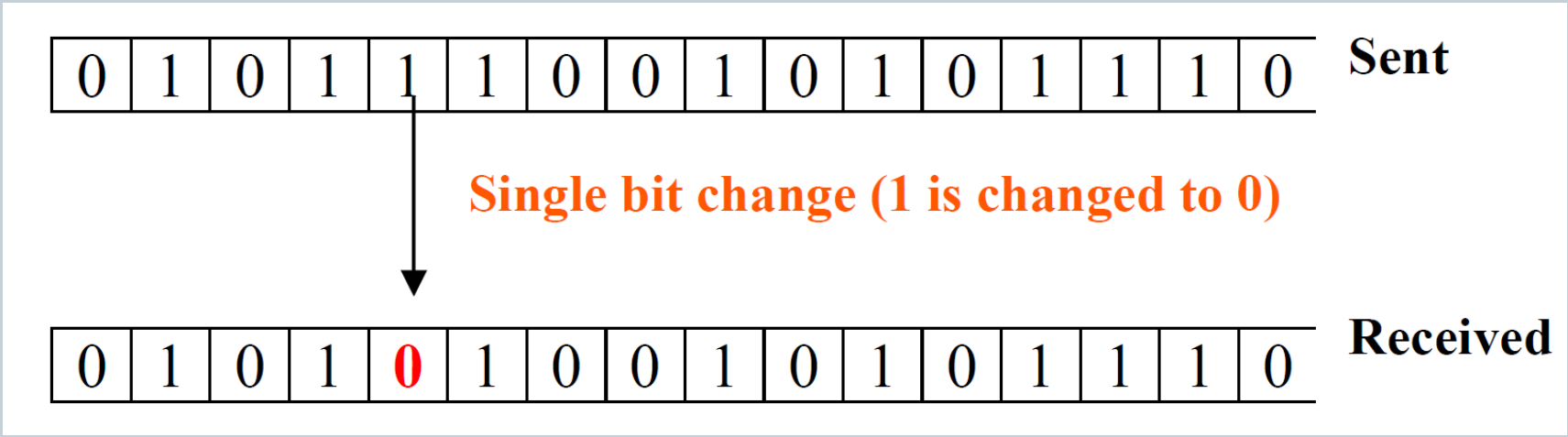
- The former strategy uses **Error-Correcting Codes**
- and latter uses **Error-detecting Codes**.

Types of errors

- Single-bit error
- Burst error
- interferences can change the timing and shape of the signal.
- If the signal is carrying binary encoded data, changes can alter the meaning of the data.

Single-bit Error

- The term single-bit error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1

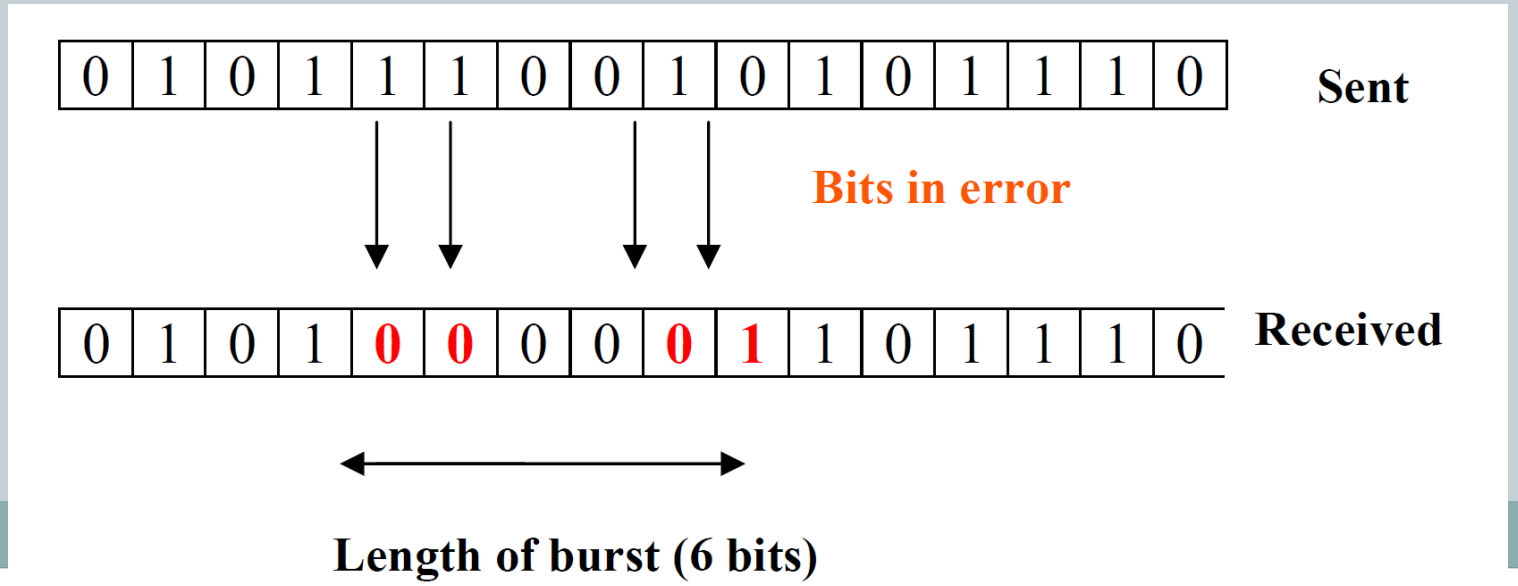


Single-bit Error

- Single bit errors are least likely type of errors in serial data transmission.
- To see why, imagine a sender sends data at 10 Mbps.
- This means that each bit lasts only for 0.1 µs (micro-second).
- For a single bit error to occur noise must have duration of only 0.1 µs (micro-second), which is very rare.
- However, a single-bit error can happen if we are having a parallel data transmission.
- For example, if 16 wires are used to send all 16 bits of a word at the same time and one of the wires is noisy, one bit is corrupted in each word.

Burst Error

- Two or more bits in the data unit have changed from 0 to 1 or vice-versa
- Doesn't means that error occurs in consecutive bits
- The length of the burst error is measured from the first corrupted bit to the last corrupted bit.
- Some bits in between may not be corrupted.



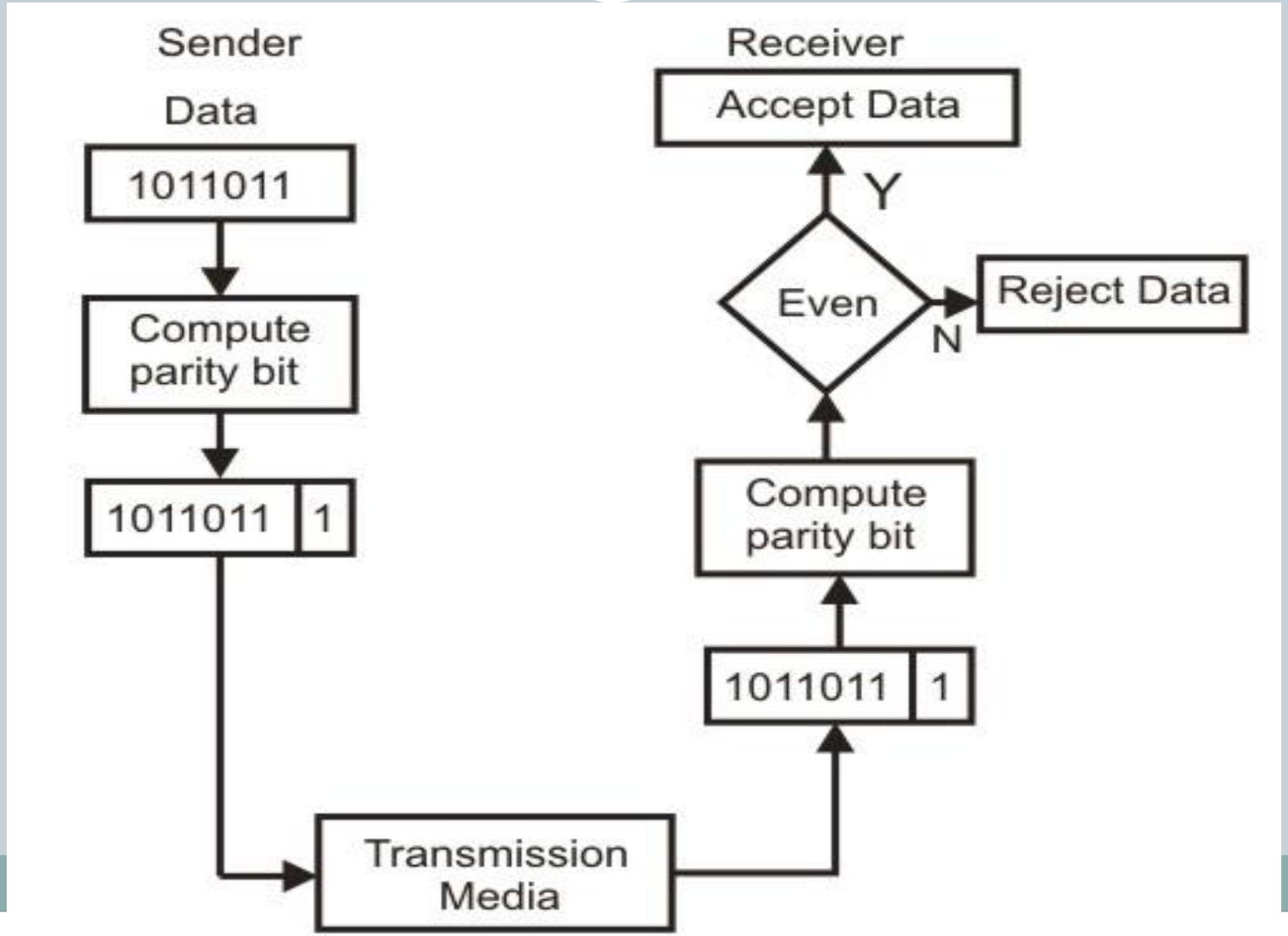
Burst Error

- Likely to happen in serial transmission.
- The duration of the noise is normally longer than the duration of a single bit, which means that the noise affects data; it affects a set of bits
- The number of bits affected depends on the data rate and duration of noise.

Error Detecting Codes

- Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors.
- Popular:
  - Simple Parity check
  - Two-dimensional Parity check
  - Checksum
  - Cyclic redundancy check

Simple Parity Checking or One-dimension Parity Check



Simple Parity Checking or One-dimension Parity Check

- common and least expensive mechanism for error-detection is the simple parity check
- redundant bit called **parity bit**, is appended to every data unit so that the number of 1s in the unit
- where a *parity bit* of 1 is added to the block if it contains an odd number of 1's and 0 is added if it contains an even number of 1's.
- At the receiving end the parity bit is computed from the received data bits and compared with the Sent parity bit

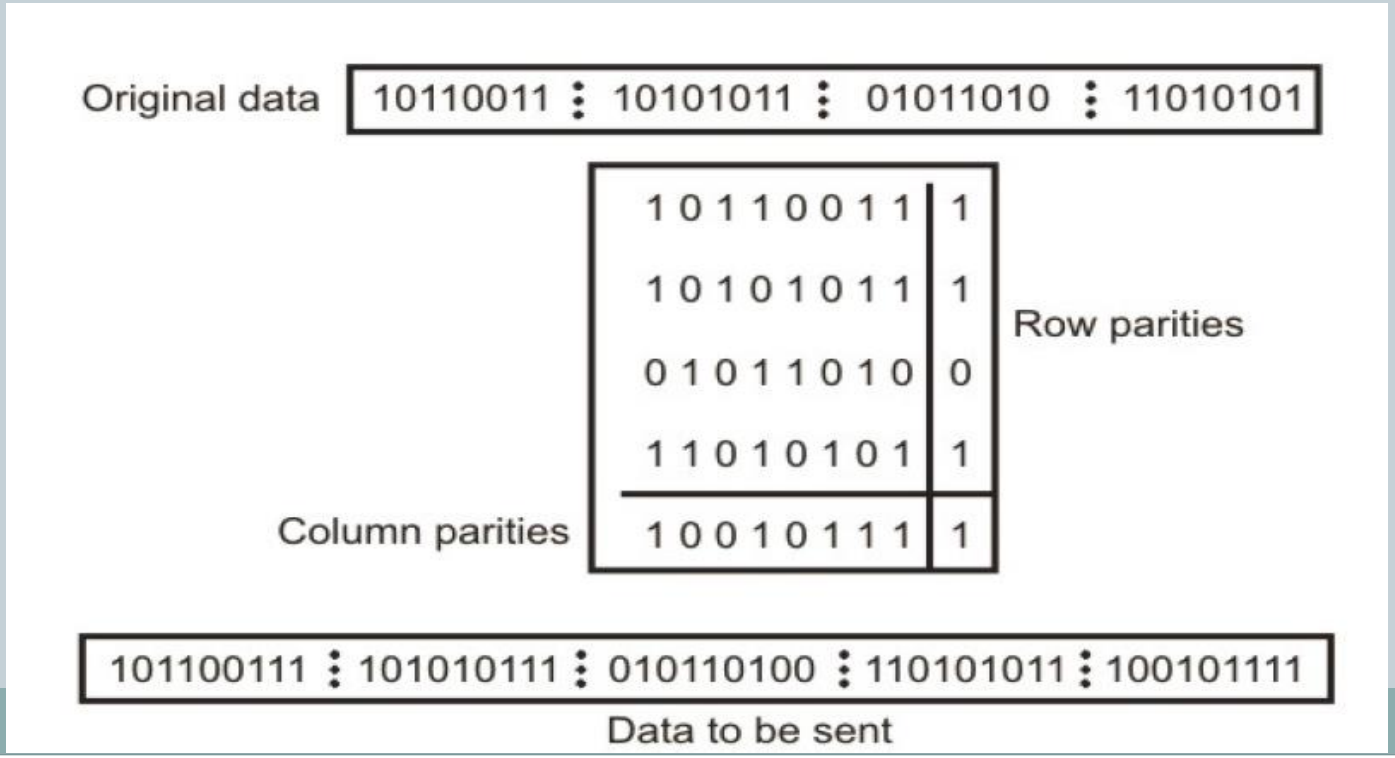
Possible 4-bit data words and corresponding code words

Decimal value	Data Block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101
3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

An observation of the table reveals that to move from one code word to another, at least two data bits should be changed.
Hence these set of code words are said to have a minimum distance (*hamming distance*) of 2
A single parity check code can detect only odd number of errors in a code word.

2D Parity Check

- Parity check bits are calculated for each row, which is equivalent to a simple parity check bit
- Parity check bits are also calculated for all columns then both are sent along with the data.



Check Sum

- the data is divided into k segments each of m bits
- In the sender's end the segments are added using 1's complement arithmetic to get the sum.
- The sum is complemented to get the checksum. The checksum segment is sent along with the data segments
- At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum.
- The sum is complemented.
- If the result is zero, the received data is accepted; otherwise discarded

Sender End and Receiver End

Example:

k=4, m=8

```
  10110011
  10101011
 ----------
  01011110
        1
 ----------
  01011111
  01011010
 ----------
  10111001
  11010101
 ----------
  10001110
        1
 ----------
  10001111
 ----------
  01110000
```

Sum : 10001111

Checksum 01110000

Example: Received data

```
  10110011
  10101011
 ----------
  01011110
        1
 ----------
  01011111
  01011010
 ----------
  10111001
  11010101
 ----------
  10001110
        1
 ----------
  10001111
  01110000
 ----------
  11111111
```

Sum: 11111111

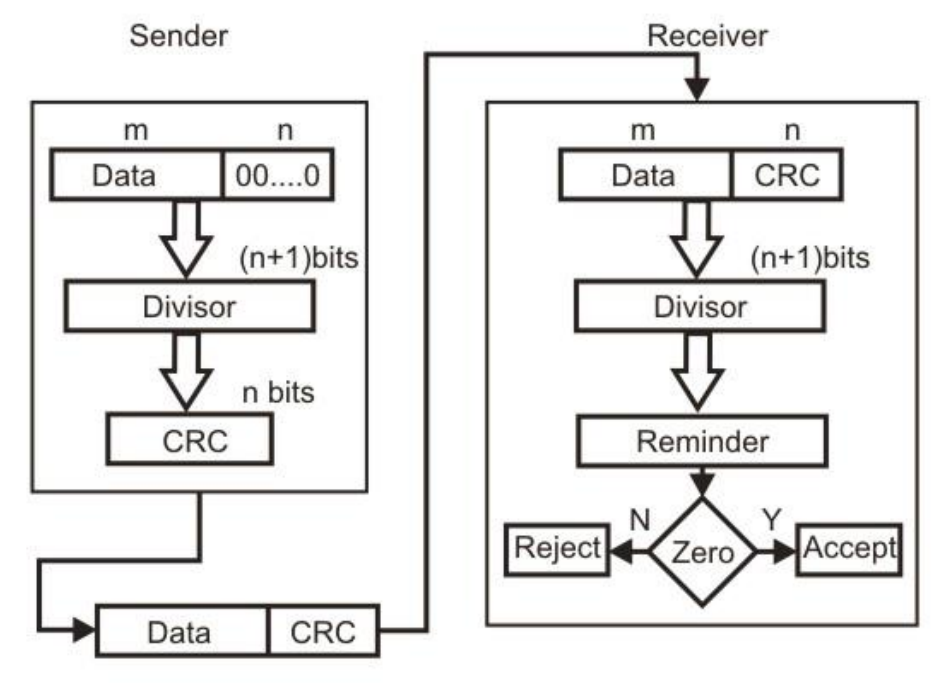
Complement = 00000000

Conclusion = Accept data

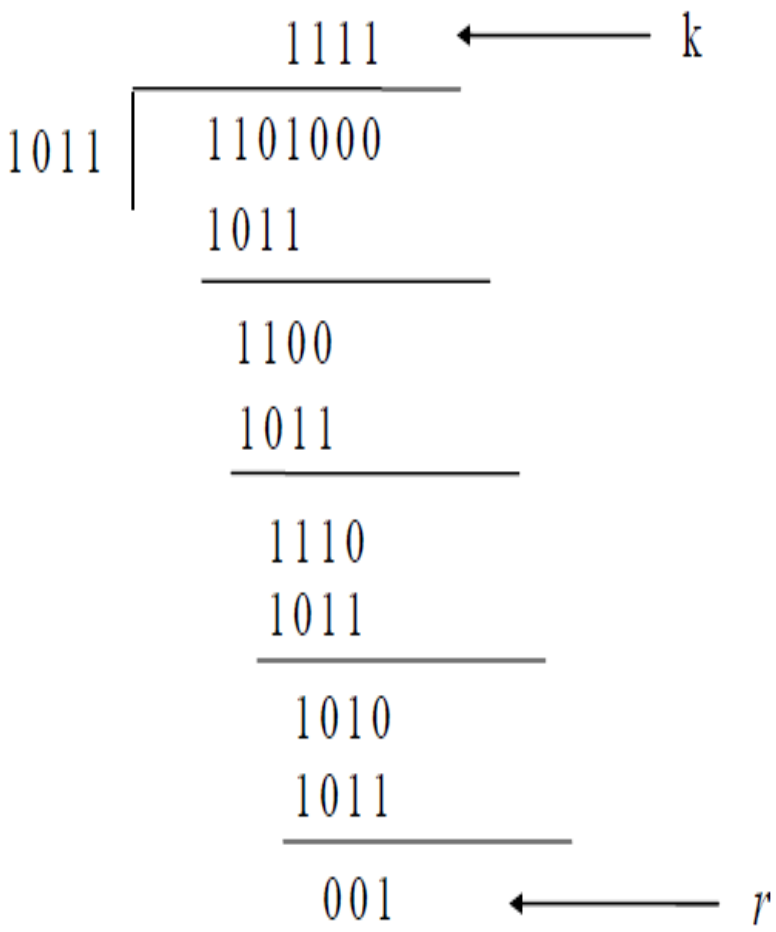
Cyclic Redundancy Checks

- Most powerful and easy to implement technique
- CRC is based on binary division
- a sequence of redundant bits, called **cyclic redundancy check bits**, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number.
- If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected

CRC


Sender
Receiver
m
n
Data
00....0
(n+1)bits
Divisor
n bits
CRC
Data
CRC
m
n
Data
CRC
(n+1)bits
Divisor
Reminder
N
Y
Reject
Zero
Accept

CRC Example



Consider the case where k=1101. Hence we have to divide 1101000 (i.e. *k* appended by 3 zeros) by 1011, which produces the remainder r=001, so that the bit frame *(k+r)* =1101001 is actually being transmitted through the communication channel.

At the receiving end, if the received number, i.e., 1101001 is divided by the same generator polynomial 1011 to get the remainder as 000, it can be assumed that the data is free of errors.

Performance of CRC

- CRC is a very effective error detection technique. If the divisor is chosen according to the previously mentioned rules, its performance can be summarized as follows:
  - CRC can detect all single-bit errors
  - CRC can detect all double-bit errors (three 1's)
  - CRC can detect any odd number of errors (X+1)
  - CRC can detect all burst errors of less than the degree of the polynomial.
  - CRC detects most of the larger burst errors with a high probability.
  - For example CRC-12 detects 99.97% of errors with a length 12 or more.

Reliability of Memory Systems: Continued…

- **Error Correcting Systems:**

  It is the process of discovery of errors, as well as the restoration of the original data.

- **Error Correction Code:**

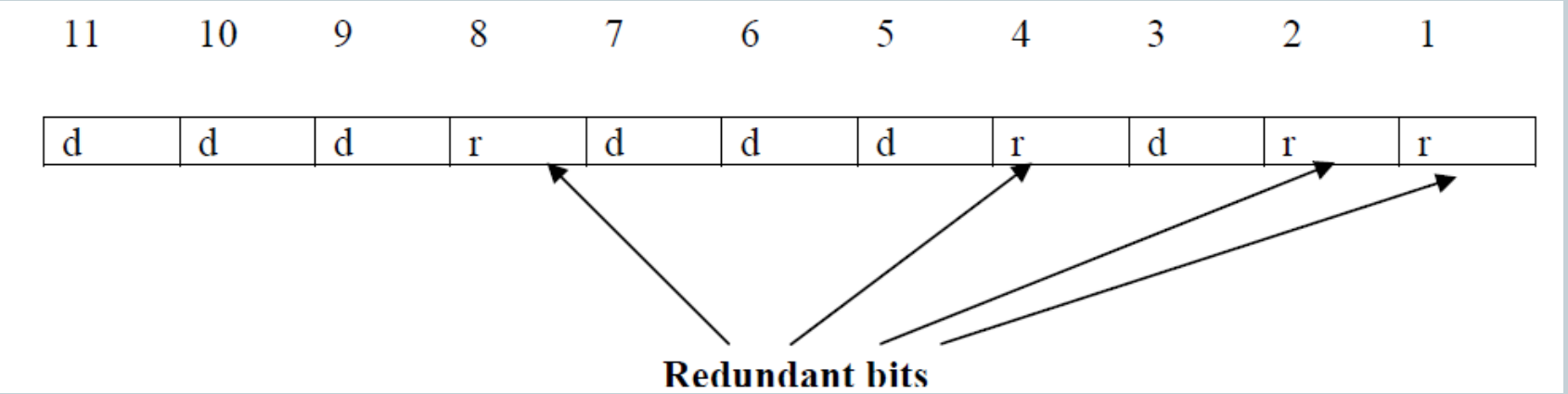
  If any error occurs, then code is used to detect and correct the errors.

Single-bit error correction

- For correcting an error one has to know the exact position of error, i.e. exactly which bit is in error
- To calculate the numbers of redundant bits (r) required to correct d data bits, let us find out the relationship between the two.
- So we have (d+r) as the total number of bits, which are to be transmitted; then r must be able to indicate at least d+r+1 different values.
- Of these, one value means no error, and remaining d+r values indicate error location of error in each of d+r locations.
- So, d+r+1 states must be distinguishable by r bits, and r bits can indicates 2r states.
- Hence, 2r must be greater than d+r+1.; **2r >= d+r+1**

Error Detection

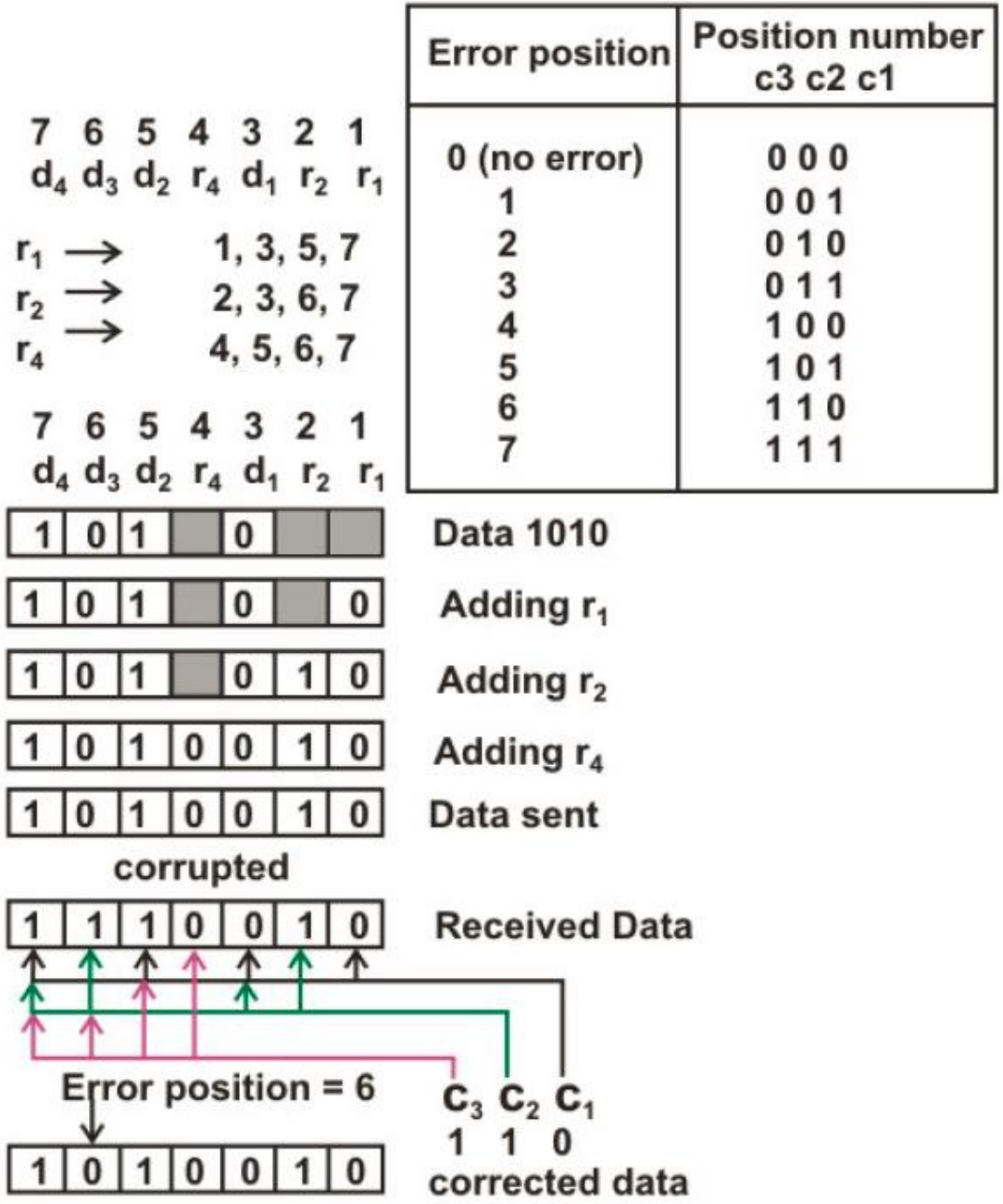
- if d is 7, then the smallest value of r that satisfies the above relation is 4. So the total bits, which are to be transmitted is 11 bits (d+r = 7+4 =11).



Positions of redundancy bits in hamming code

Error Detection

- Basic approach for error detection by using Hamming code is as follows:
  - To each group of m information bits k parity bits are added to form (m+k) bit code as shown in Fig. 3.2.8.
  - Location of each of the (m+k) digits is assigned a decimal value.
  - The k parity bits are placed in positions 1, 2, ..., 2k-1 positions.–K parity checks are performed on selected digits of each codeword.
  - At the receiving end the parity bits are recalculated. The decimal value of the k parity bits provides the bit-position in error, if any.



Error position	Position number c3 c2 c1
0 (no error)	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

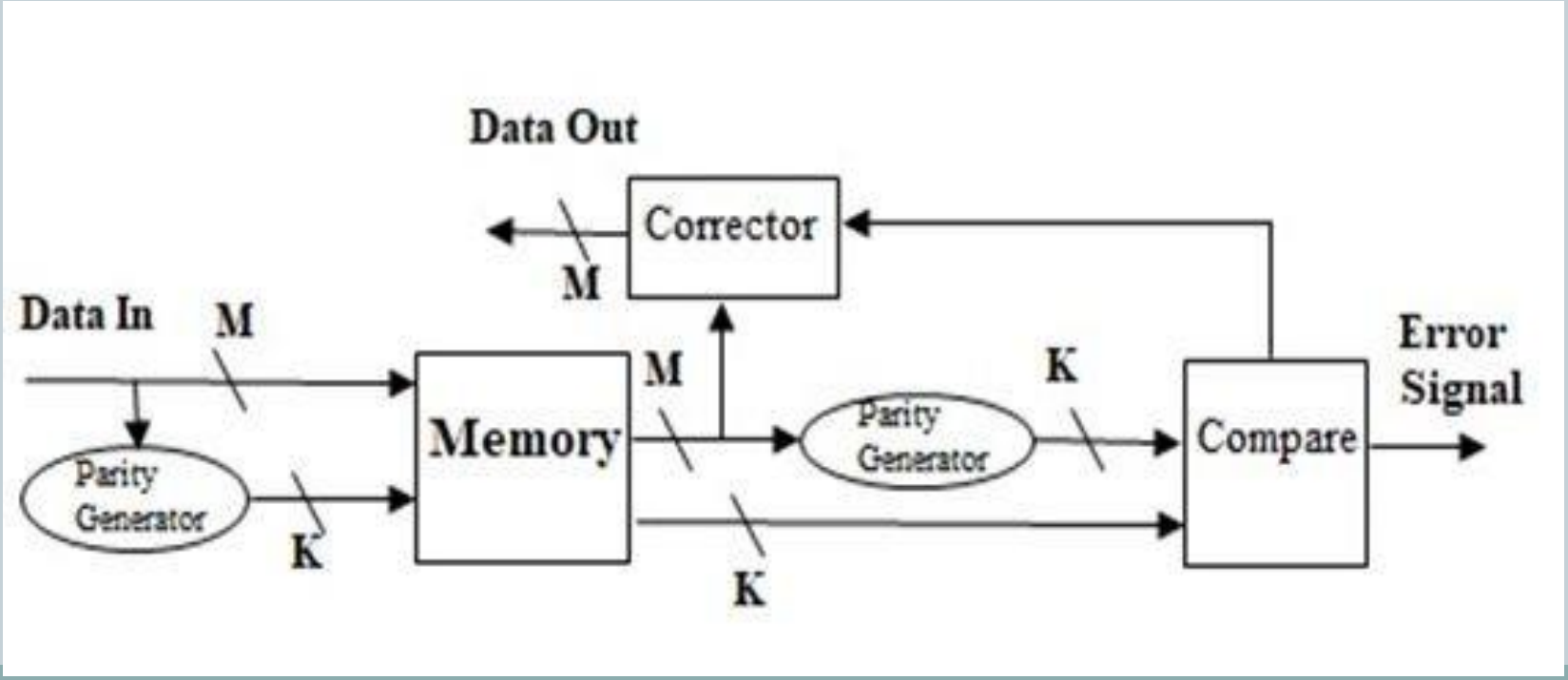
Use of Hamming code for error correction for a 4-bit data

Error Correction

- hamming code is used for correction for 4-bit numbers (d4d3d2d1) with the help of three redundant bits (r3r2r1).
- For the example data 1010, first r1 (0) is calculated considering the parity of the bit positions, 1, 3, 5 and 7.
- Then the parity bits r2 is calculated considering bit positions 2, 3, 6 and 7.
- Finally, the parity bits r4 is calculated considering bit positions 4, 5, 6 and 7 as shown.
- If any corruption occurs in any of the transmitted code 1010010, the bit position in error can be found out by calculating r3r2r1 at the receiving end.
  - For example, if the received code word is 1110010, the recalculated value of r3r2r1 is 110, which indicates that bit position in error is 6, the decimal value of 110.

Hamming Code Architecture

The following general architecture generates a Double Error Detection (DED) and Single-Error Correcting (SEC) code for any number of bits.



Hamming Code Operational Procedure

- Data are to be read into memory.
- A function f (Parity Generator), is applied on the data to yield a code.
- Parity code (K bits) and the data (M bits) are deposited in memory.
- While reading out the word, a new set of K parity code bits is produced from the M data bits.
- Finally, compare new parity code with old parity code bits.
- **Note:** For M data bits or K check bits single error correction hamming code must have **(2^K) - 1 >= M + K**

Arrangement of Hamming code

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Parity bit					P8				P4		P2	P1

Check (Parity) Bits Calculations

P1 = 3⊕5⊕7⊕9⊕11 (bit positions)

P2 = 3⊕6⊕7⊕10⊕11 (bit positions)

P4 = 5⊕6⊕7⊕12 (bit positions)

P8 = 9⊕10⊕11⊕12 (bit positions)

Example:

Generate hamming code (Even parity) for a 8-bit word – 00111001.

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Hamming Code	0	0	1	1	P8	1	0	0	P4	1	P2	P1

$P1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$P2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$P4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$

$P8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Example:

Detection of error using hamming code

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Hamming Code	0	0	1	1	P8=0	1	0	0	P4=1	0	P2=1	P1=1

P1 = 0⊕0⊕1⊕1⊕0 = 0

P2 = 0⊕0⊕1⊕1⊕0 =0

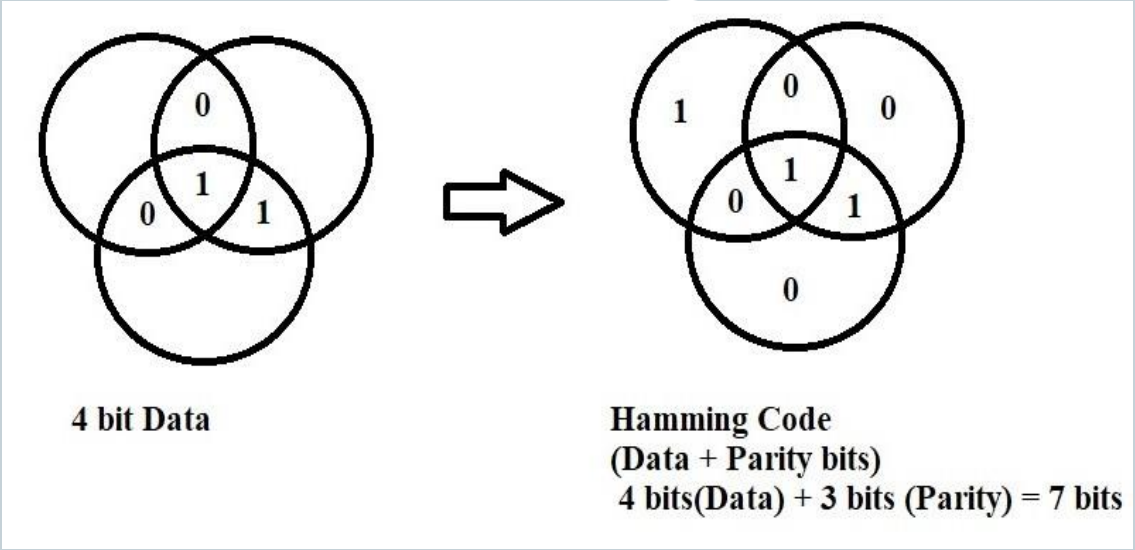
P4= 0⊕0⊕1⊕0 = 1

P8 = 1⊕1⊕0⊕0 = 0

P8	P4	P2	P1
0	1	1	1
⊕ 0	1	0	0
0	0	1	1

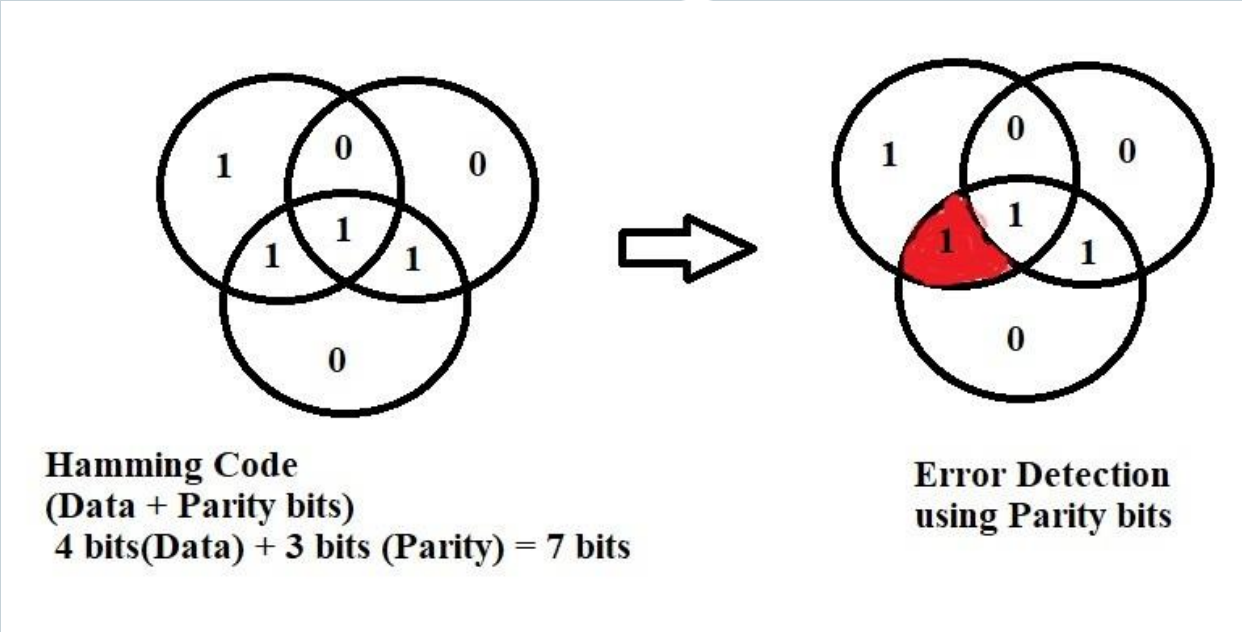
Hamming code technique illustrated by using Venn diagram

Hamming Encoding:



Hamming coding technique is illustrated by using Venn diagram

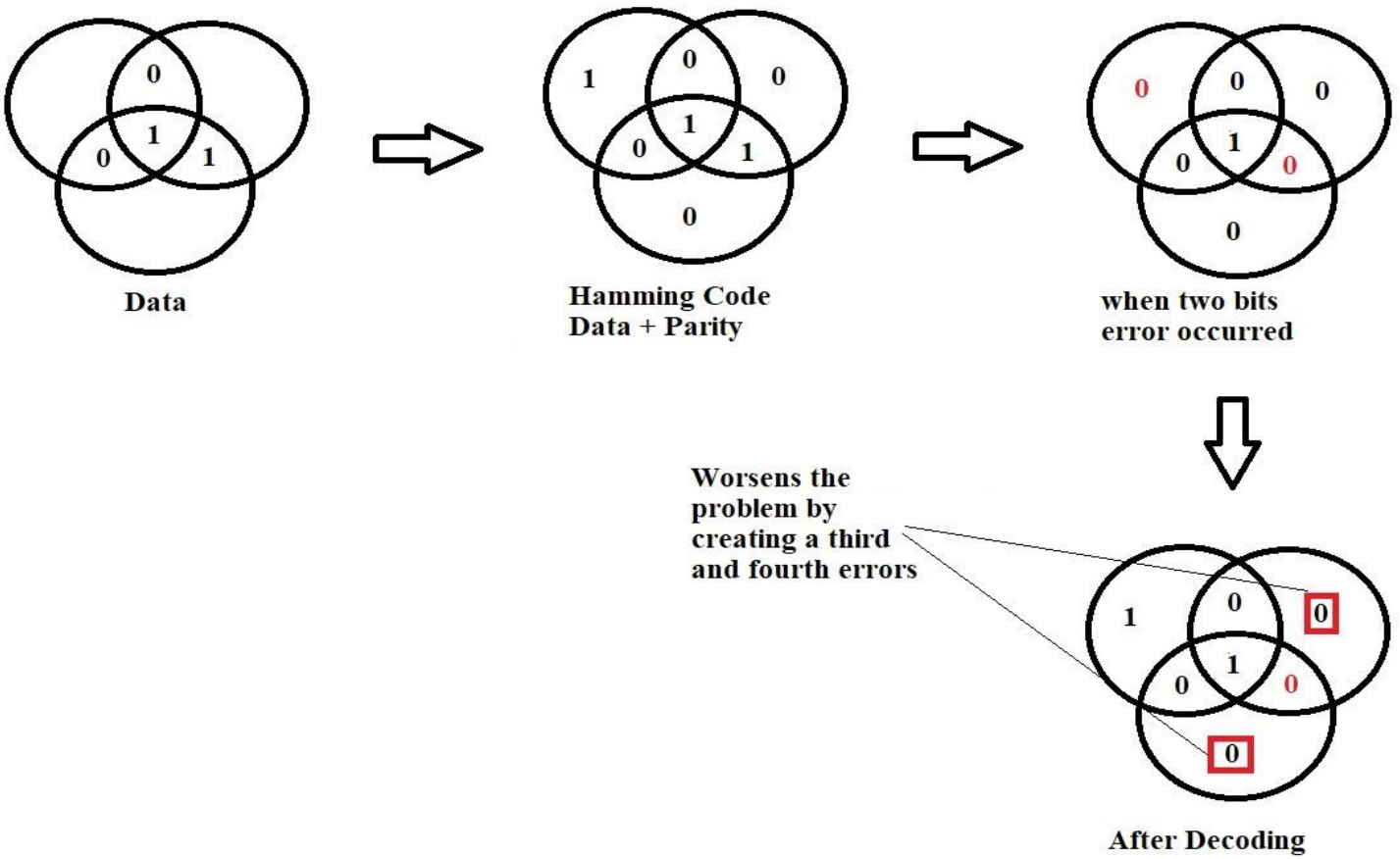
Error Detection:



Hamming Code
(Data + Parity bits)
4 bits(Data) + 3 bits (Parity) = 7 bits

Error Detection using Parity bits

Hamming SEC-DEC Code



Assignment:

1)Find the number odd parity or check bits needed when we have 2048 bit input word? Solution: input word = 2048 bits = 2^12 =12 bits.

2)Suppose an 8-bit input data is 11101010. Determine the odd parity or check bits?

3)Suppose an 8-bit word is 00111101, Determines the even parity bits, hamming code word and illustrate how to correct when a single bit error occurs?

Assignment: - Solutions

1)Find the number odd parity or check bits needed when we have 2048 bit input word? Solution: input word = 2048 bits = 2^12 =12 bits.

2)Suppose an 8-bit input data is 11101010. Determine the odd parity or check bits? Solution:

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Hamming Code	D12	D11	D10	D9	P8	D7	D6	D5	P4	D3	P2	P1
Input Data	1	1	1	0		1	0	1		0		

P1 = D3⊕D5⊕D7⊕D9⊕ D11 = 0⊕1⊕1⊕0⊕1 = 0

P2 = D3⊕D6⊕D7⊕D10⊕ D11 = 0⊕0⊕1⊕1⊕1 =0

P4= D5⊕D6⊕D7⊕ D12 = 1⊕0⊕1⊕1 = 0

P8 = D9⊕D10⊕D11⊕ D12 = 0⊕1⊕1⊕1 = 0

Assignment: - Solutions (Continued..)

3) Suppose an 8-bit word is 00111101, Determines the even parity bits, hamming code word and illustrate how to correct

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Hamming Code	D12	D11	D10	D9	P8	D7	D6	D5	P4	D3	P2	P1
Input Data	0	0	1	1		1	1	0		1		

P1 = D3⊕D5⊕D7⊕D9⊕ D11 = 1⊕0⊕1⊕1⊕0 = 1

P2 = D3⊕D6⊕D7⊕D10⊕D11 = 1⊕1⊕1⊕1⊕0 =0

P4= D5⊕D6⊕D7⊕D12 = 0⊕1⊕1⊕0 = 0

P8 = D9⊕D10⊕D11⊕D12 = 1⊕1⊕0⊕0 = 0

Supposed error occurred in fifth position that is D5 got 1 instead of 0 then calculate new even parity.

P1 = D3⊕D5⊕D7⊕D9⊕ D11 = 1⊕1⊕1⊕1⊕0 = 0

P2 = D3⊕D6⊕D7⊕D10⊕D11 = 1⊕1⊕1⊕1⊕0 =0

P4= D5⊕D6⊕D7⊕D12 = 1⊕1⊕1⊕0 = 1

P8 = D9⊕D10⊕D11⊕D12 = 1⊕1⊕0⊕0 = 0

Syndrome word

P8 P4 P2 P1

0 0 0 1

⊕ 0 1 0 0

0 1 0 1

Error at 5th location

References:

1. David A. Patterson and . John L. Hennessy "Computer Organization and Design-The Hardware/Software Interface" 5th edition, Morgan Kaufmann, 2011.

2. Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer organization, Mc Graw Hill, Fifth edition ,Reprint 2011.

3. W. Stallings, Computer organization and architecture, Prentice-Hall, 8th edition, 2009