Chapter 6

Memory Organization

Outline

- Introduction
- Computer Memory System Overview
  - Characteristics of Memory Systems
  - Memory Hierarchy
- Cache Memory Principles
  - Elements of Cache Design -Cache Addresses
  - Cache Size -Mapping Function
  - Direct Mapping -Associative Mapping
  - Set-associative mapping -Replacement Algorithms
  - Write Policy -Line Size
  - Number of caches -Multilevel caches
  - Unified versus split caches

Introduction

A computer has a **hierarchy** of memory subsystems:

- some internal to the system
  - *i.e.* directly accessible by the processor;
- some external
  - accessible via an I/O module;

Computer Memory System Overview

Classification of memory systems according to their key characteristics:

Location
- Internal (e.g., processor registers, cache, main memory)
- External (e.g., optical disks, magnetic disks, tapes)

Capacity
- Number of words
- Number of bytes

Unit of Transfer
- Word
- Block

Access Method
- Sequential
- Direct
- Random
- Associative

Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization
- Memory modules

Location: either internal or external to the processor.

- Forms of internal memory:
  - registers;
  - cache;
  - and others;
- Forms of external memory:
  - disk;
  - magnetic tape (too old... =P);
  - devices that are accessible to the processor via I/O controllers.

Capacity: amount of information the memory is capable of holding.

- Typically expressed in terms of bytes (1 byte = 8 bits) or **words**;
- A word represents each addressable block of the memory
  - common word lengths are 8, 16, and 32 bits;
- External memory capacity is typically expressed in terms of bytes;

U**nity of transfer:** number of bytes read / written into memory at a time.

- Need not equal a word or an addressable unit;
- Also possible to transfer **blocks**:
  - Much larger units than a word;
  - Used in external memory...
  - External memory is slow...
  - **Idea:** minimize number of acesses, optimize amount of data transfer;

Access Method: How are the units of memory accessed?

- **Sequential Method:** Memory is organized into units of data, called records.
  - Access must be made in a specific linear sequence;
  - Stored addressing information is used to assist in the retrieval process.
  - A shared read-write head is used;
  - The head must be moved from its one location to the another;
  - Passing and rejecting each intermediate record;
  - Highly variable times.

Sequential Method Example: Magnetic Tape

Access Method: How are the units of memory accessed?

- **Direct Access Memory:**
  - Involves a shared read-write mechanism;
  - Individual records have a unique address;
  - Requires accessing general record vicinity plus sequential searching, counting, or waiting to reach the final location;
  - Access time is also variable;

Magnetic Disk

Access Method: How are the units of memory accessed?

- **Random Access:** Each addressable location in memory has a unique,
  - physically wired-in addressing mechanism.
  - Constant time;
  - independent of the sequence of prior accesses;
  - Any location can be selected at random and directly accessed;
  - Main memory and some cache systems are random access.

Access Method: How are the units of memory accessed?

- **Associative:** RAM that enables one to make a comparison of desired bit
  - locations within a word for a specified match
  - a word is retrieved based on a portion of its contents rather than its address;
  - retrieval time is constant independent of location or prior access patterns
  - *E.g.:* cache, neural networks.

Performance:

- **Access time (latency):**
  - For RAM: time to perform a read or write operation;
  - For Non-RAM: time to position the read-write head at desired location;
- **Memory cycle time:** Primarily applied to RAM:
  - Access time + additional time required before a second access;
  - Required for electrical signals to be terminated/regenerated;
  - Concerns the system bus.

Performance

- **Transfer time:** Rate at which data can be transferred in / out of memory;
  - For RAM: 1/cycle time
  - For Non-RAM: $Tn = TA + n/R$

  where:
  - *Tn*: Average time to read or write n bits;
  - *TA*: Average access time;
  - *n*: Number of bits
  - *R*: Transfer rate, in bits per second (bps)

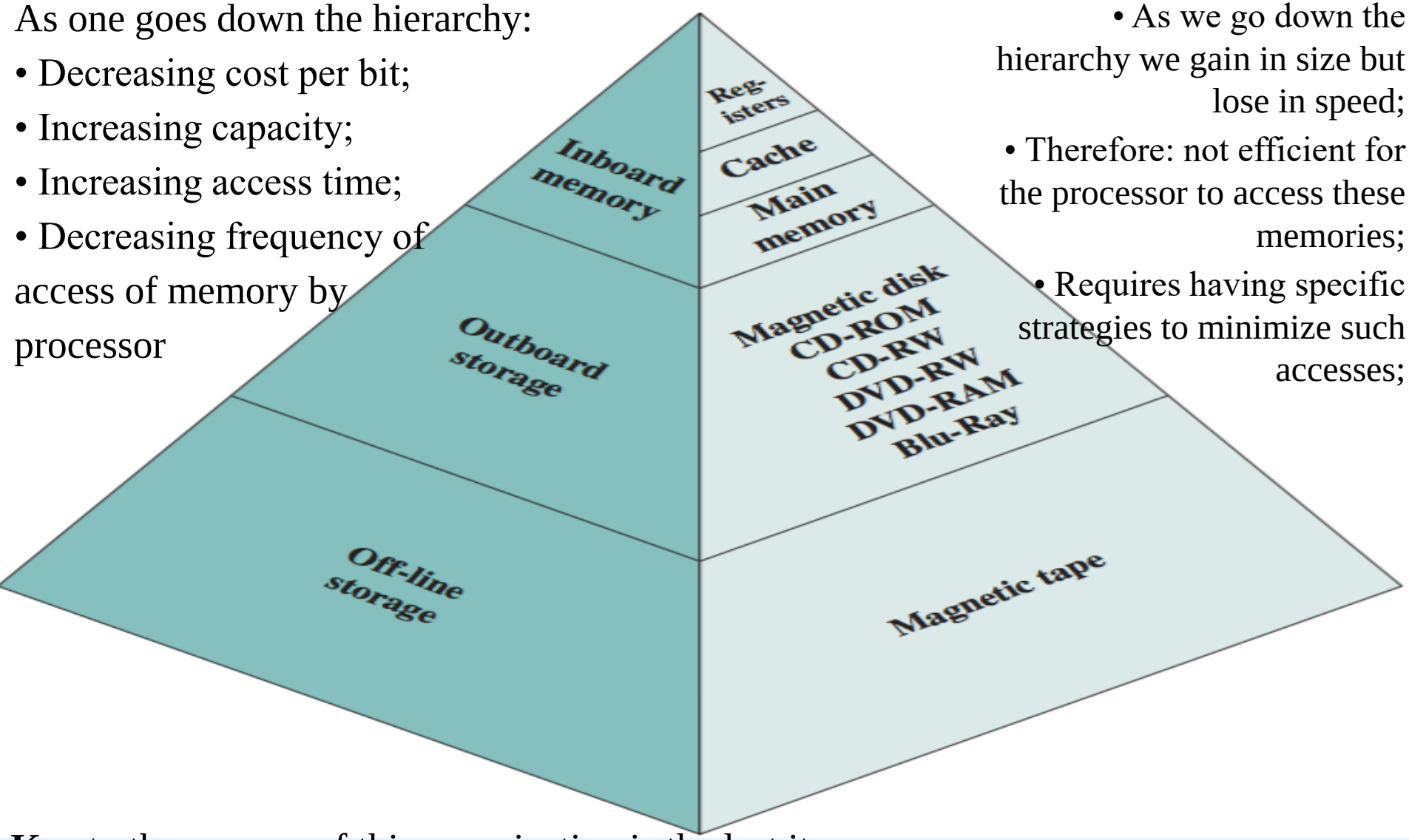
Physical characteristics:

- **Volatile:** information decays naturally or is lost when powered off;
- **Nonvolatile:** information remains without deterioration until changed:
  - no electrical power is needed to retain information.;
  - *E.g.:* Magnetic-surface memories are nonvolatile;
  - Semiconductor memory (memory on integrated circuits) may be either volatile or nonvolatile.

Memory Hierarchy

As one goes down the hierarchy:

- Decreasing cost per bit;
- Increasing capacity;
- Increasing access time;
- Decreasing frequency of access of memory by processor



- As we go down the hierarchy we gain in size but lose in speed;
- Therefore: not efficient for the processor to access these memories;
- Requires having specific strategies to minimize such accesses;

Key to the success of this organization is the last item:

- Decreasing frequency of access of memory by processor.

Memory Hierarchy

How can we develop strategies to minimize these accesses?

- **Space and Time locality of reference principle:**
  - **Space:** if we access a memory location, close by addresses will very likely be accessed;
  - **Time:** if we access a memory location, we will very likely access it again;

Example….

Suppose that the processor has access to two levels of memory:

- **Level 1 - *L*1:** contains 1000 words and has an access time of 0.01μ*s*;
- **Level 2 - *L*2:** contains 100,000 words and has an access time of 0.1μ*s*.

Assume that:

- If word ∈ *L*1, then the processor accesses it directly;
- If word ∈ *L*2, then word is transferred to *L*1 and then accessed by the processor.
- *H* define the fraction of all memory accesses that are found L1;
- *T*1 is the access time to L1 0.01μ*s* ; and *T*2 is the access time to L2 0.1μ*s*

Example ….

- Textual description of the previous :
- For high percentages of $L1$ access, the average total access time is much closer to that of $L1$ than that of $L2$;
- Now lets consider the following scenario:
- Suppose 95% of the memory accesses are found in $L1$. Average time to access a word is:
- $(0.95)(0.01\mu s) + (0.05)(0.01\mu s + 0.1\mu s) = 0.0095 + 0.0055 = 0.015\mu s$
- Average access time is much closer to $0.01\mu s$ than to $0.1\mu s$, as desired.

Strategy to minimize accesses should be:

- organize data across the hierarchy such that
- percentage of accesses to lower levels is substantially less than that of upper levels
- *I.e. L*2 memory contains all program instructions and data:
- Data that is currently being used should be place in *L*1;
- Eventually, data in *L*1 will have to be swapped back to *L*2 to make room for new data;
- On average, most references will be to data contained in *L*1.

This principle can be applied across more than two levels of memory:

- Processor registers:
  - fastest, smallest, and most expensive type of memory
- Followed immediately by the cache:
  - Stages data movement between registers and main memory;
  - Improves perfomance;
  - Is not usually visible to the processor;
  - Is not usually visible to the programmer.
- Followed by main memory:
  - principal internal memory system of the computer;
  - Each location has a unique address.

Cache Memory Principles

- Cache memory is designed to combine:
- memory access time of expensive, high-speed memory combined with the large memory size of less expensive, lower-speed memory.
- large and slow memory together with a smaller, faster memory;
- the cache contains a copy of portions of main memory.

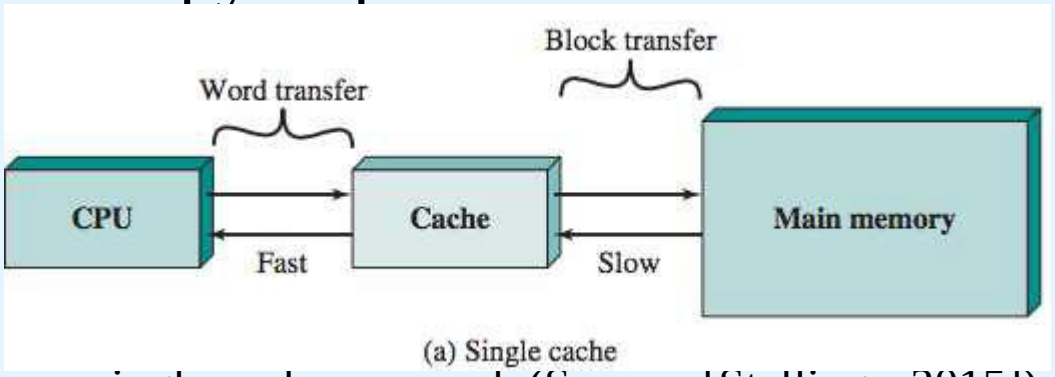


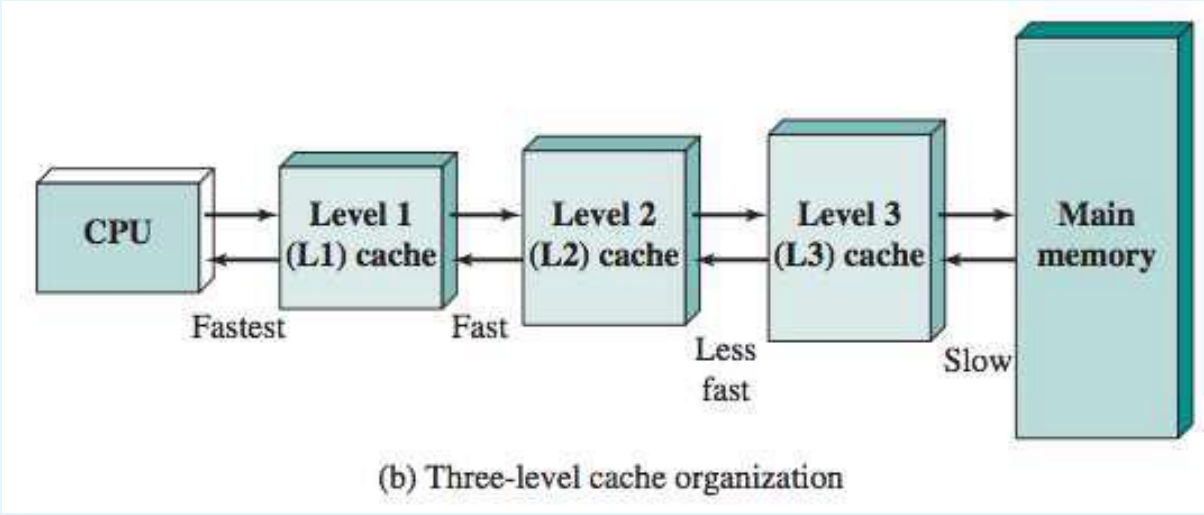
Figure: Cache and main memory - single cache approach (Source: [Stallings, 2015])

Conti…..

- When the processor attempts to read a word of memory: a check is made to determine if the word is in the cache;
- If so, the word is delivered to the processor.
- If the word is not in cache:
  - a block of main memory is read into the cache;
  - then the word is delivered to the processor.
- Because of the locality of reference principle:
- when a block of data is fetched into the cache;
  - it is likely that there will be future references to that same memory location;

What if we introduce multiple levels of cache?

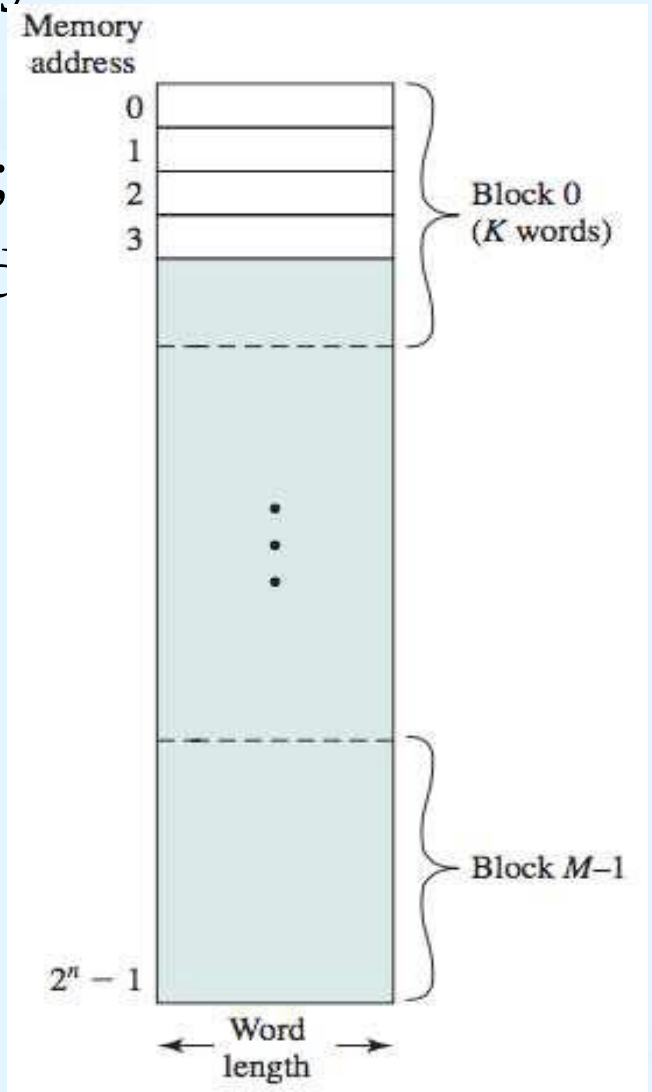
- L2 cache is slower and typically larger than the L1 cache
- L3 cache is slower and typically larger than the L2 cache.



(b) Three-level cache organization

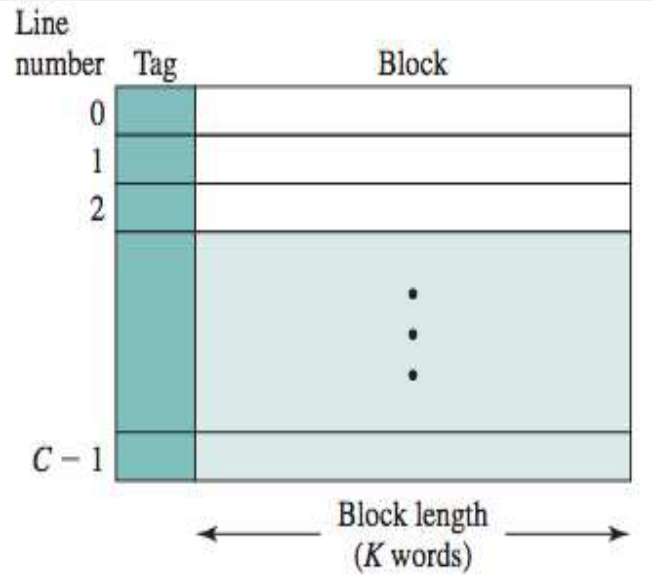
Main memory:

- the structure of the main-memory system
  - Consists of 2^n addressable words;
  - Each word has a unique *n*-bit address;
  - Memory consists of a number of fixed words each;
  - There are $M = 2^n/K$ blocks;
  - Where:
    - M is total number of blocks,
    - n is the number of bits is a word
    - K is the number of words in a block

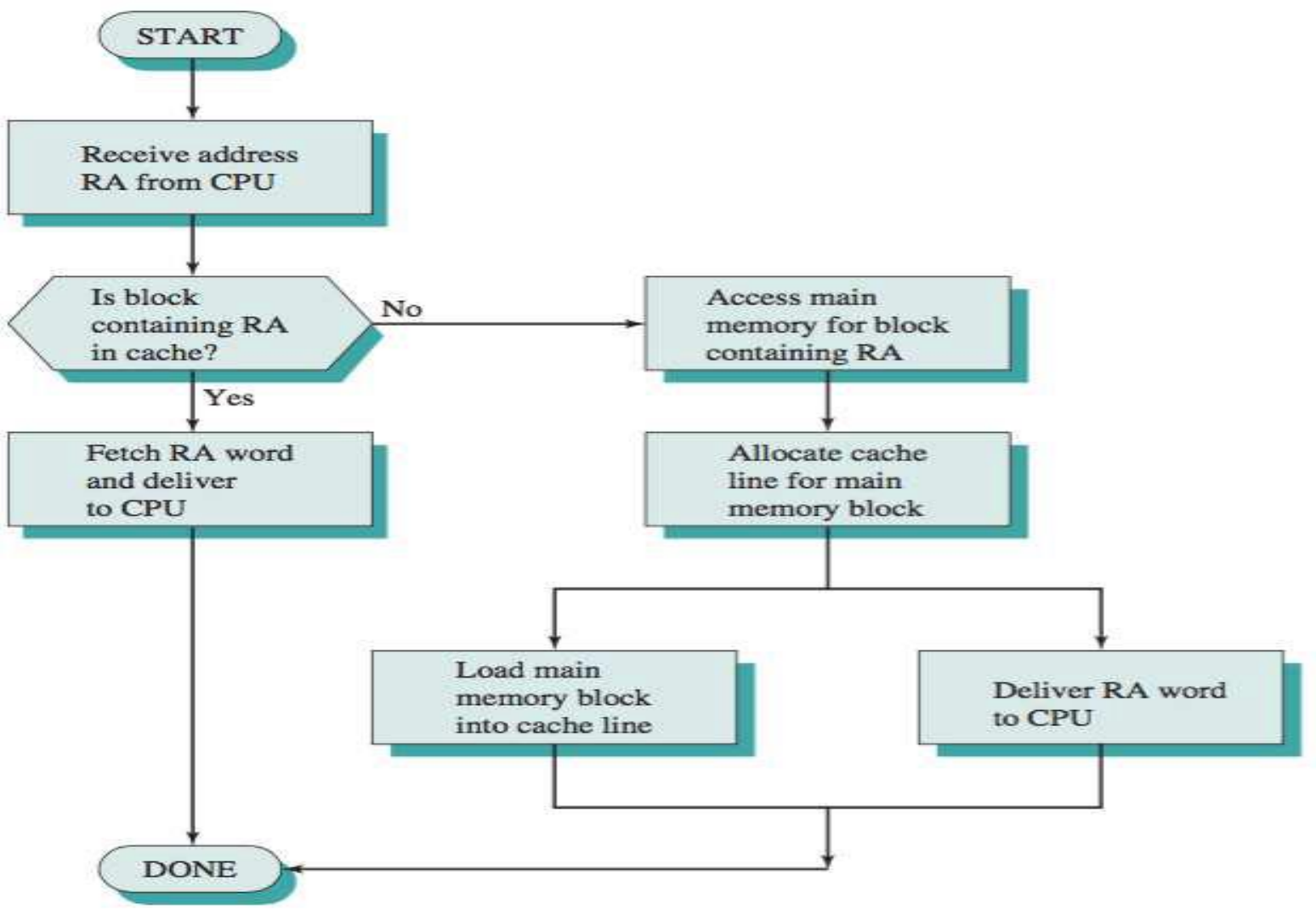


Conti…

- the structure of the cache system
  - Consisting of *m* blocks, called lines;
  - Each line contains *K* words;
  - $m \ll M$
  - Each line also includes control bits:
  - Not shown in the figure;
  - If a word in a block of memory is read:
  - block is transferred to a cache line; because $m \ll M$, lines: cannot permanently store a block.
  - need to identify the block stored;
  - info stored in the tag field;



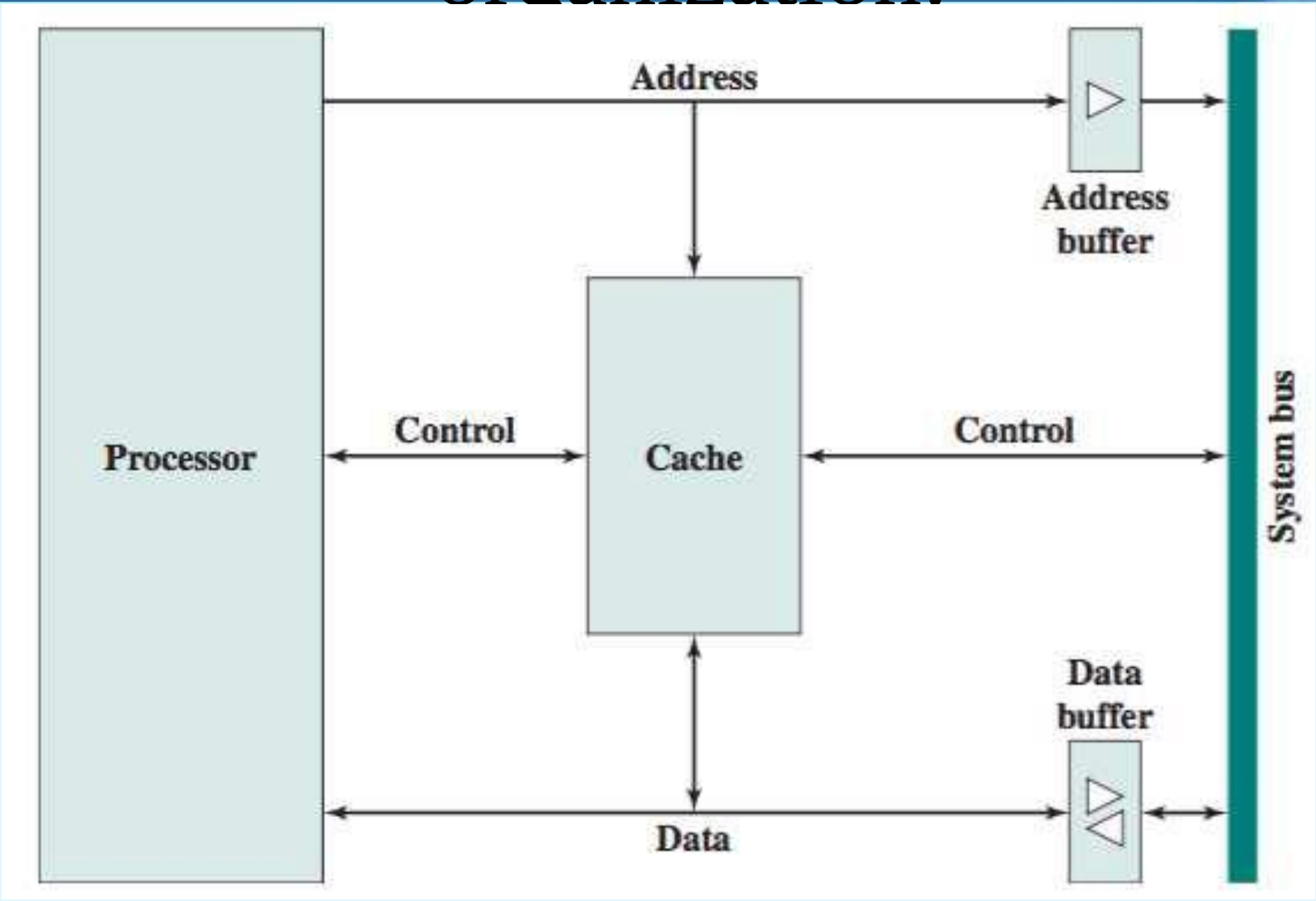
Set of operations that need to be performed for a read operation issued by the processor



Cache Read operation:

- Processor generates **read address (RA)** of word to be read;
- If the word ∈ cache, it is delivered to the processor;
- Otherwise:
  - Block containing that word is loaded into the cache from RAM;
  - Word is delivered to the processor;
  - These last two operations occurring in parallel.

Typical contemporary cache organization:



Cont…

- In this organization the **cache:**
- connects to the processor via data, control, and address lines; the data and address lines also attach to data and address buffers:
- which attach to a system bus.....from which main memory is reached.
- What do you think happens when a word is **in** cache? When a **cache hit** occurs (word is in cache):
  - • the data and address buffers are disabled;
  - • communication is only between processor and cache;
  - • no system bus traffic.

Cont…

- What do you think happens when a word is **not in** cache?
  - When a **cache miss** occurs (word is not in cache):
  - the desired address is loaded onto the system bus;
  - the data are returned through the data buffer...to both the cache and the processor

Elements of Cache Design: Cache architectures can be classified according to key elements:

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

Line Size

Number of Caches
- Single or two level
- Unified or split

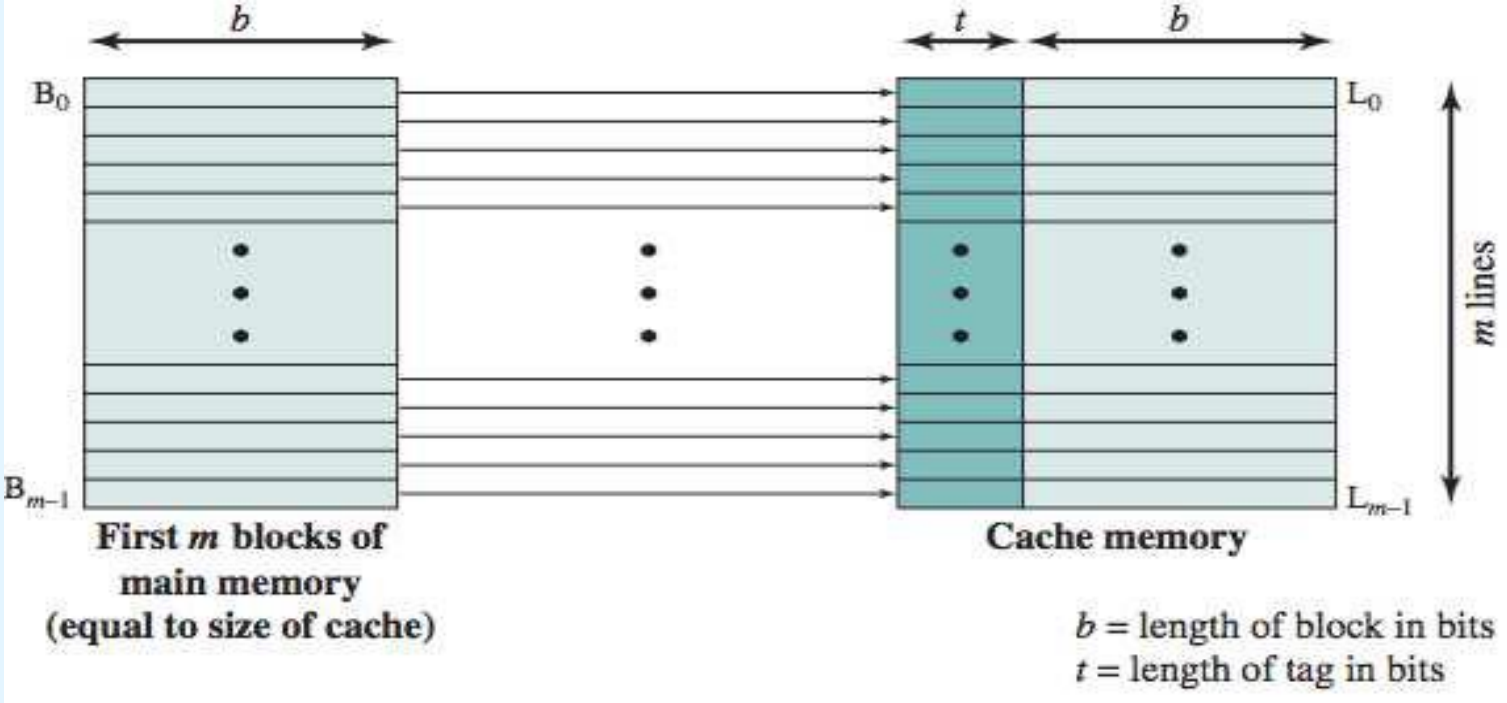
Cache memory Organization

Mapping Function

- Recall that there are fewer cache lines than main memory blocks
- How should one map main memory blocks into cache lines?
- Three techniques can be used for mapping blocks into cache lines:
  - direct;
  - associative;
  - set associative

Direct Mapping

- Maps each block of main memory into only one possible cache line as: $i = j \bmod m$
- where: i = cache line number; j = main memory block number; and m = number of lines in the cache



Conti…

- Previous picture shows mapping of main memory blocks into cache:
- First *m* main memory blocks map into each line of the cache;
- Next *m* blocks of main memory map in the following manner:
  - *Bm* maps into line *L*0 of cache;
  - *Bm*+1 maps into line *L*1;
  - and so on...
- Modulo operation implies repetitive structure;

Conti….

- With direct mapping blocks are assigned to lines as follows:

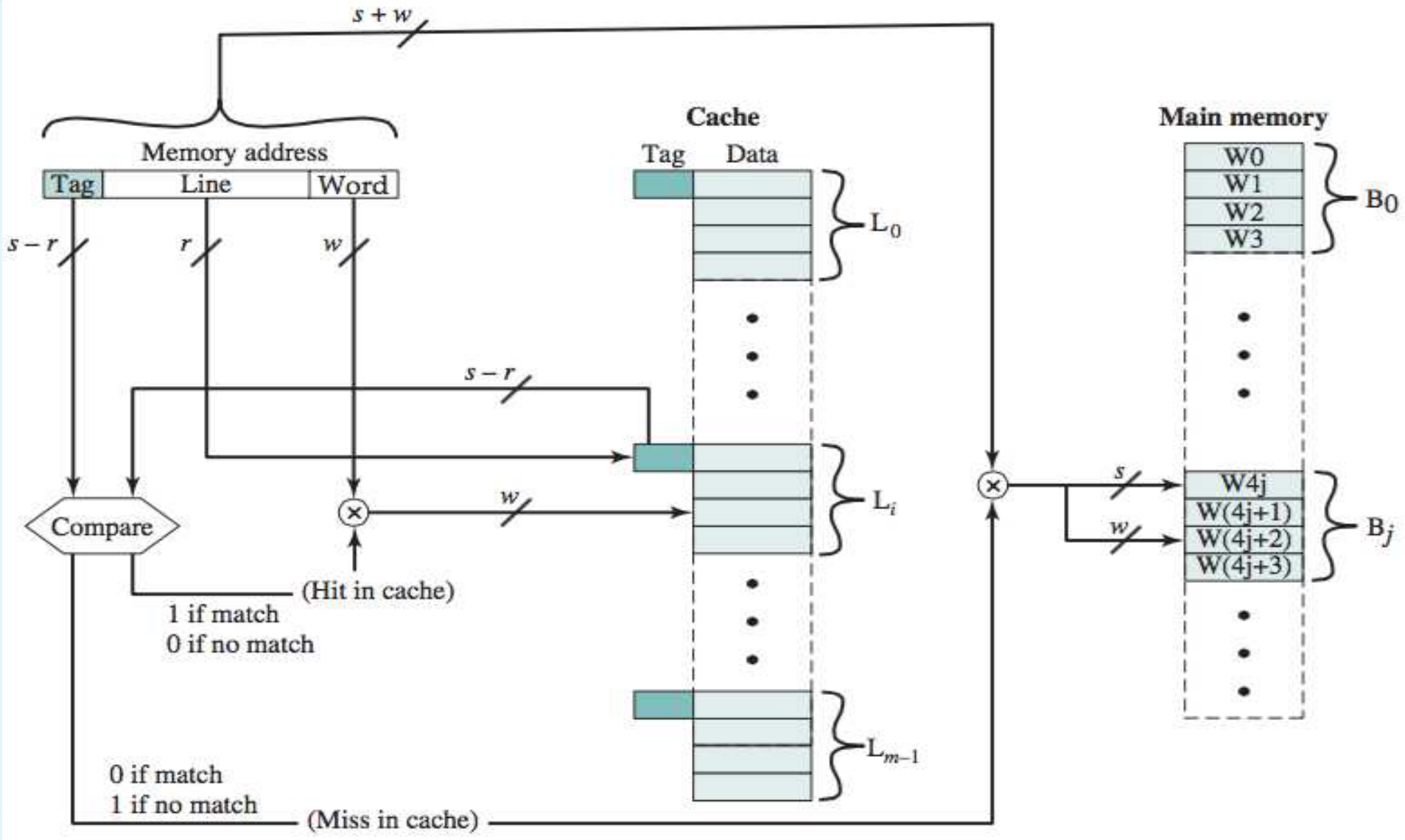
Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

- Over time:
  - each line can have a different main memory block;
  - we need the ability to distinguish between these;
  - the most significant s - r bits (tag) serve this purpose.

Conti

- Each main **memory address** ($s + w$ bits) can be viewed as:
- **Offset** (w bits): identifies a word within a block of main memory;
- **Line** (r bits): specify one of the 2^r cache lines;
- **Tag** ($s - r$ bits): to distinguish blocks that are mapped to the same line:
- Note that:
  - s allows addressing of all blocks in main memory... whilst the number of available cache lines is much smaller than 2^s
  - Tag fields identifies the block that is currently mapped to a cache line;

To determine whether a block is in the cache:



To determine whether a block is in the cache:

1 Use the line field of the memory address to index the cache line;

2 Compare the tag from the memory address with the line tag;

- **1** If **both match**, then **Cache Hit:**

  1 Use the line field of the memory address to index the cache line;

  2 Retrieve the corresponding word from the cache line;

- **2** If **both do not match**, then **Cache Miss:**

  1 Use the line field of the memory address to index the cache line;

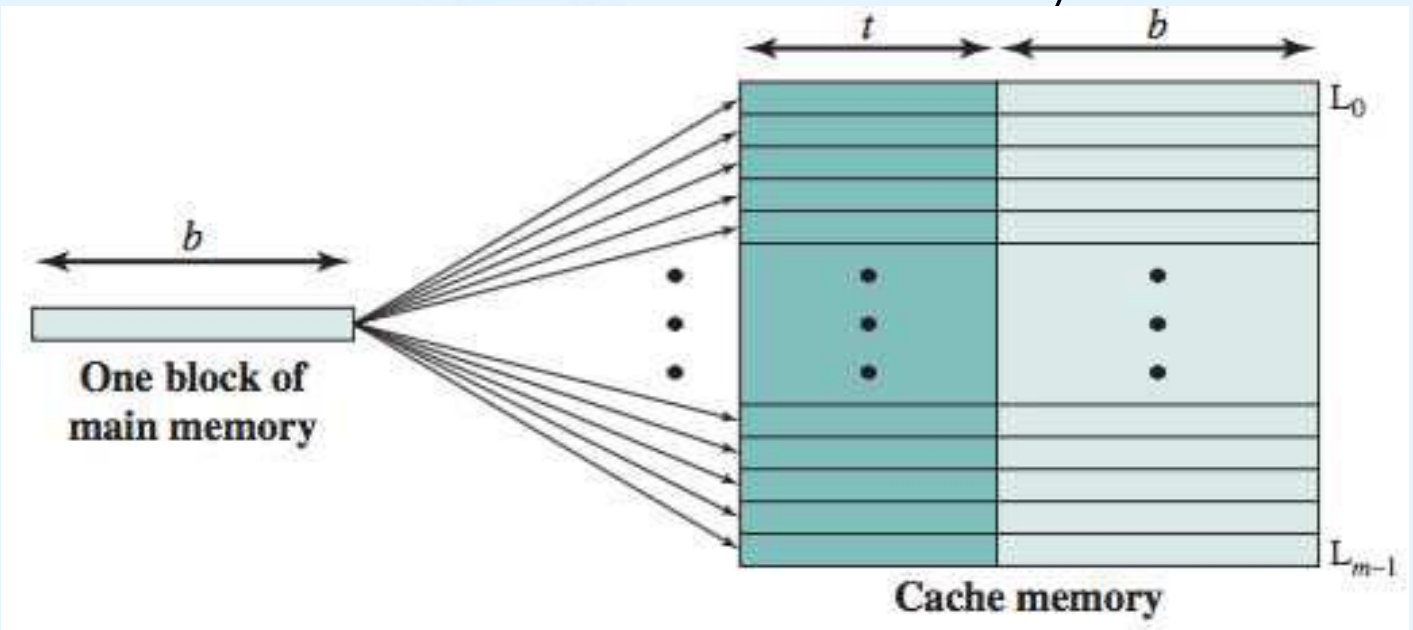
  2 Update the cache line (word + tag);

Direct mapping technique:

- **Advantage:** simple and inexpensive to implement;
- **Disadvantage:** there is a fixed cache location for any given block;
- if a program happens to reference words repeatedly from two different blocks that map into the same line;
- then the blocks will be continually swapped in the cache;
- hit ratio will be low (a.k.a. **thrashing**).

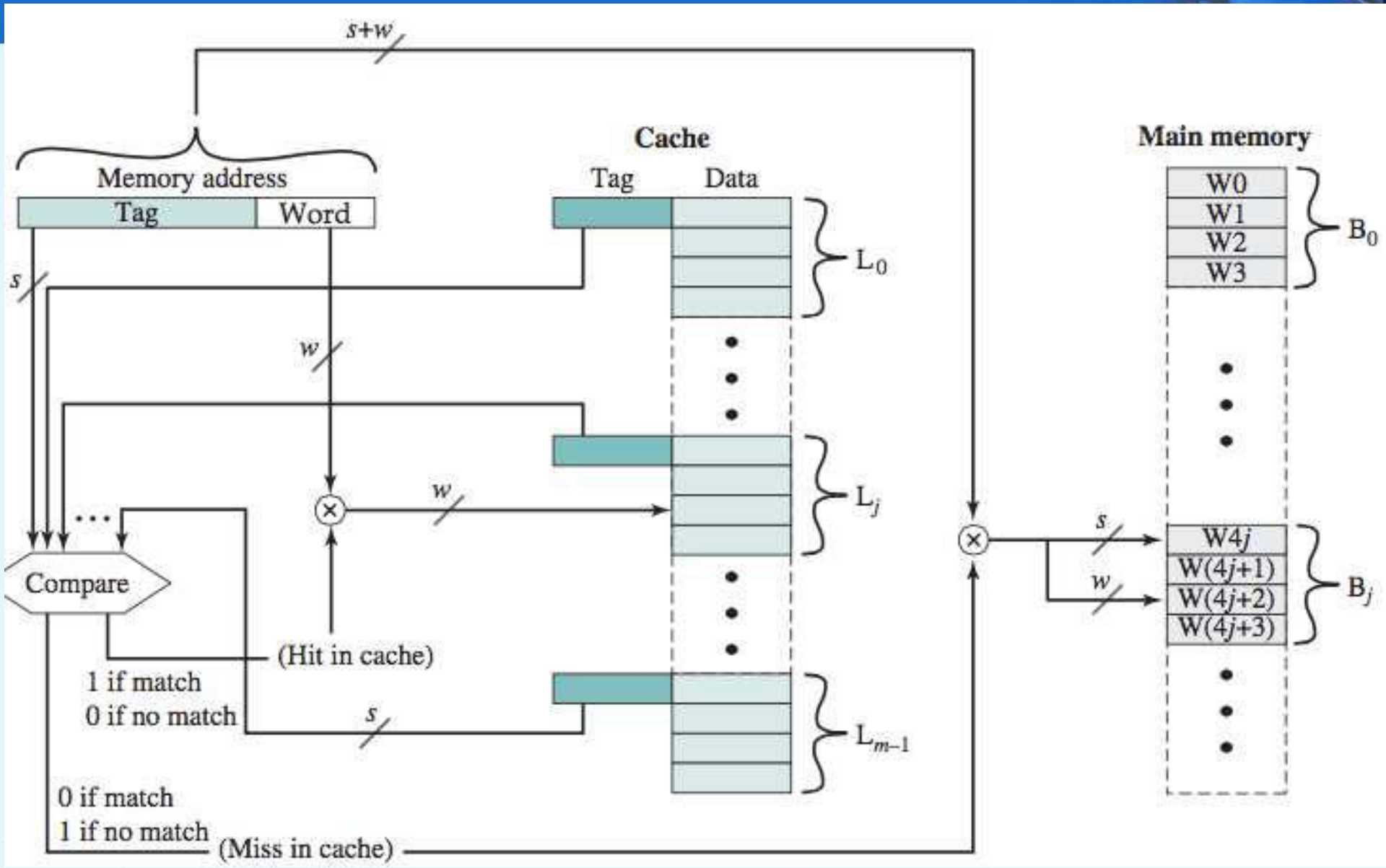
Associative Mapping

- Overcomes the disadvantage of direct mapping by:
- permitting each block to be loaded into any cache line:



  - Cache interprets a memory address as a **Tag** and a **Word** field:
  - **Tag:** (*s* bits) uniquely identifies a block of main memory;
  - **Word:** (*w* bits) uniquely identifies a word within a block;

Fully associative cache organization



To determine whether a block is in the cache:

- Simultaneously **compare** every line's tag for a match:
- If a **match exists**, then **Cache Hit:**

  1 Use the tag field of the memory address to index the cache line;

  2 Retrieve the corresponding word from the cache line;
- If a **match does not exist**, then **Cache Miss:**

  1 Choose a cache line. How?

  2 Update the cache line (word + tag);

Advantage and Disadvantage of Associative mapping technique:

- **Advantage:** flexibility as to which block to replace when a new block is read into the cache;
- **Disadvantage:** complex circuitry required to examine the tags of all cache lines in parallel.

Can you see any way of improving this scheme?

- **Perform less comparisons**
  - Instead of comparing the tag against all lines
  - compare only against a subset of the cache lines.
  - Welcome to set-associative mapping =)

Set-associative mapping

Combination of direct and associative approaches:

- Cache consists of a number of sets, each consisting of a number of lines.
- From **direct mapping:**
  - each block can only be mapped into a single set;
  - *I.e.* Block *Bj* always maps to set *j*;
  - Done in a modulo way =)
- From **associative mapping:**
  - each block can be mapped into any cache line of a certain set.

Conti

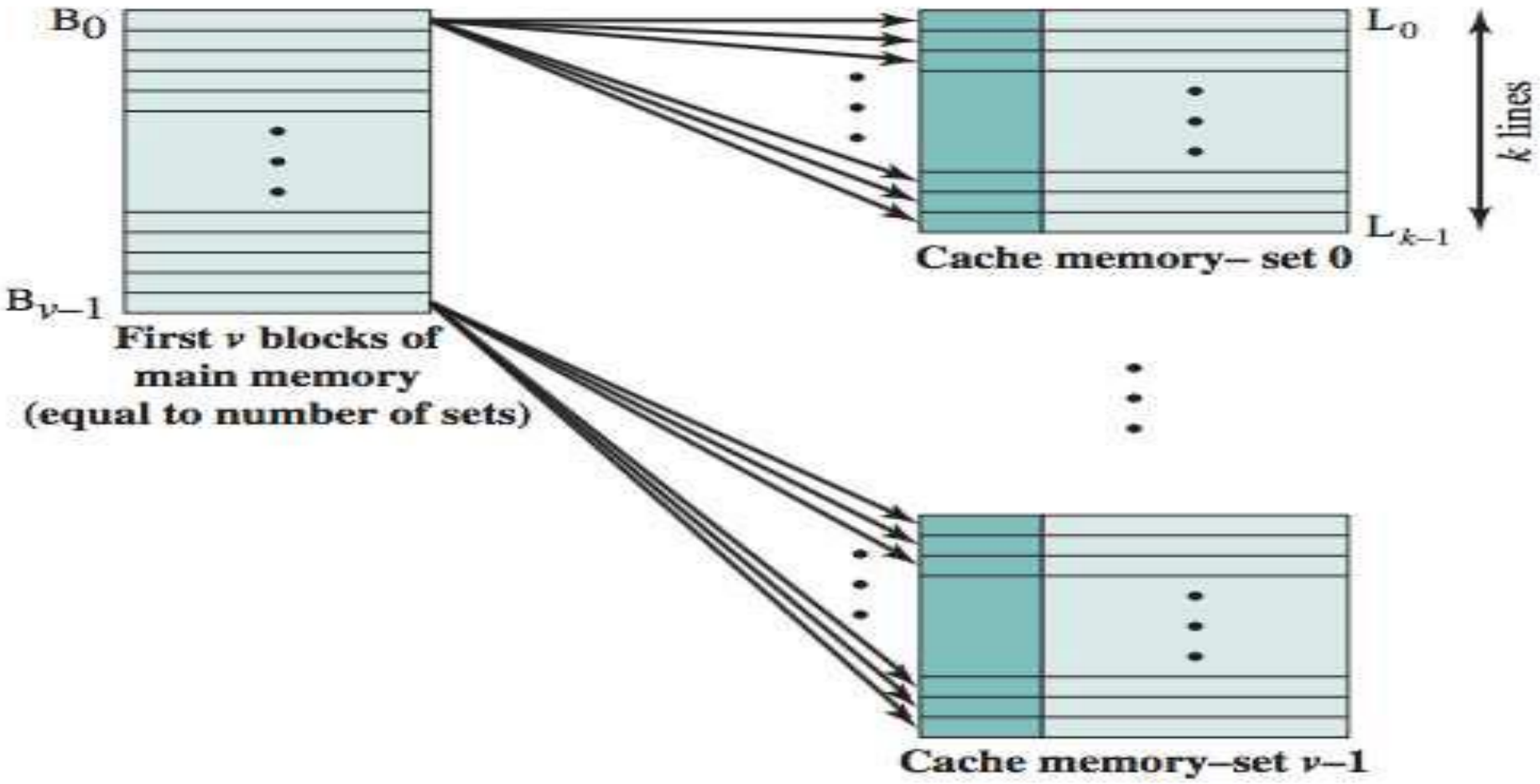
The relationships are:

- $m = v \times k$
- $i = j \bmod v$

where:

- i = cache set number;
- j = main memory block number;
- m = number of lines in the cache;
- v = number of sets;
- k = number of lines in each set

Associative Mapped



Idea:

- 1 memory block!1 single set, but to any row of that set.
- can be physically implemented as *v* associative caches

Conti

Cache interprets a memory address as a **Tag**, a **Set** and a **Word** field:

- **Set:** identifies a set (d bits, $v = 2d$ sets);
- **Tag:** used in conjunction with the set bits to identify a block ($s - d$ bits);
- **Word:** identifies a word within a block;

To determine whether a block is in the cache:

1 Determine the **set** through the set fields;

2 Compare address tag simultaneously with all cache line tags;

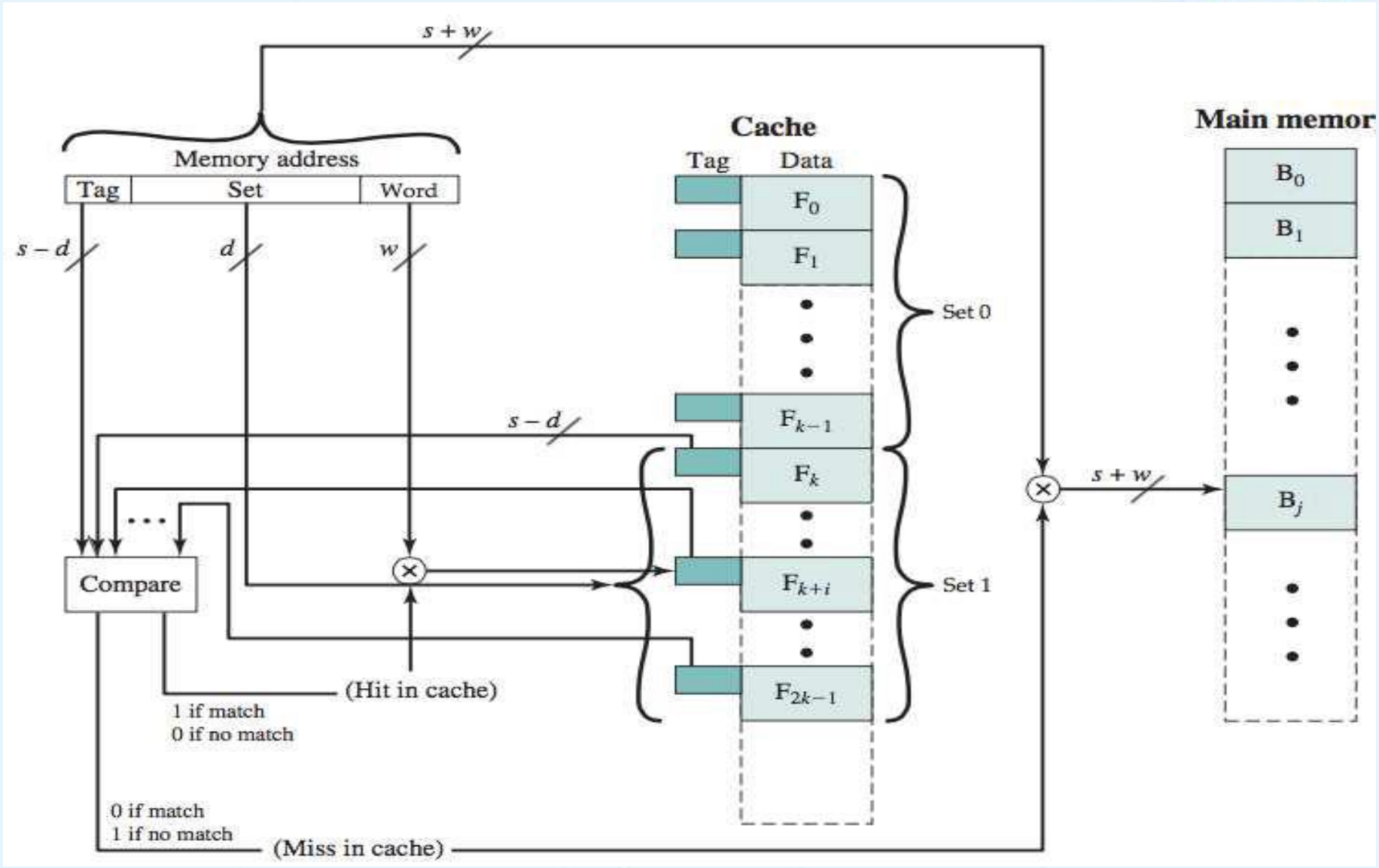
3 If a **match exists**, then **Cache Hit:**

- **1** Retrieve the corresponding word from the cache line;

4 If a **match does not exist**, then **Cache Miss:**

- **1** Choose a cache line within the set. How?
- **2** Update the cache line (word + tag);

K -Way Set Associative Cache Organization



Number of caches

Recent computer systems:

- use multiple caches;

This design issue covers the following topics

- number of cache levels;
- also, the use of unified versus split caches;

Lets have a look at the details of each one of these...

Multilevel Caches

As logic density increased:

- became possible to have a cache on the same chip as the processor:
  - reduces the processor's external bus activity;
  - therefore improving performance;
- when the requested instruction or data is found in the on-chip cache:
  - bus access is eliminated;
  - because of the short data paths internal to the processor:
    - cache accesses will be faster than even zero-wait state bus cycles.
    - Furthermore, during this period the bus is free to support other transfers.

Conti

With the continued shrinkage of processor components:

- processors now incorporate a second cache level (L2) or more:
- savings depend on the hit rates in both the L1 and L2 caches.
  - In general: use of a second-level cache does improve performance;
  - However, multilevel caches complicate design issues:
    - size;
    - replacement algorithms;
    - write policy;

Unified versus split caches

In recent computer systems:

- it has become common to **split** the cache into two:
  - Instruction cache;
  - Data cache;
- both exist at the same level:
  - typically as two L1 caches:
  - When the processor attempts to fetch:
    - an instruction from main memory, it first consults the instruction L1 cache,
    - data from main memory, it first consults the data L1 cache.

Two potential advantages of a **unified cache:**

- Higher hit rate than split caches:
  - automatically load balancing between instruction and data fetches, *i.e.:*
  - if an execution pattern involves more instruction fetches than data fetches...the cache will tend to fill up with instructions;
  - if an execution pattern involves relatively more data fetches...the cache will tend to fill up with data;
- Only one cache needs to be designed and implemented.

Key advantage of the **split cache** design:

- eliminates competition for the cache between
  - Instruction fetch/decode/execution stages...
  - and the load / store data stages;
- Important in any design that relies on the **pipelining of instructions:**
  - fetch instructions ahead of time
  - thus filling a pipeline with instructions to be executed.

Conti

With a **unified** instruction / data cache:

- Data / instructions will be stored in a single location;
- Pipelining:
  - allows for multiples stages of the instruction cycle to be executed simultaneously.
- Because there is a single cache:
  - The executions of multiple stages cannot be performed;
  - Performance bottleneck;

Split cache structure overcomes this difficulty.

Semiconductor Main Memory

Memory is a collection of **cells** with the following properties

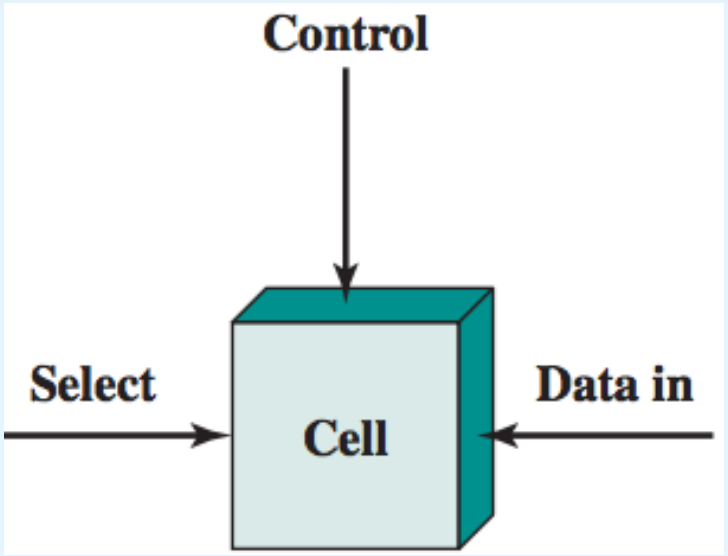
- They exhibit two stable states, used to represent binary 1 and 0;
- They are capable of being written into (at least once), to set the state;
- They are capable of being read to sense the state;
- Access time is the same regardless of the location;

Conti

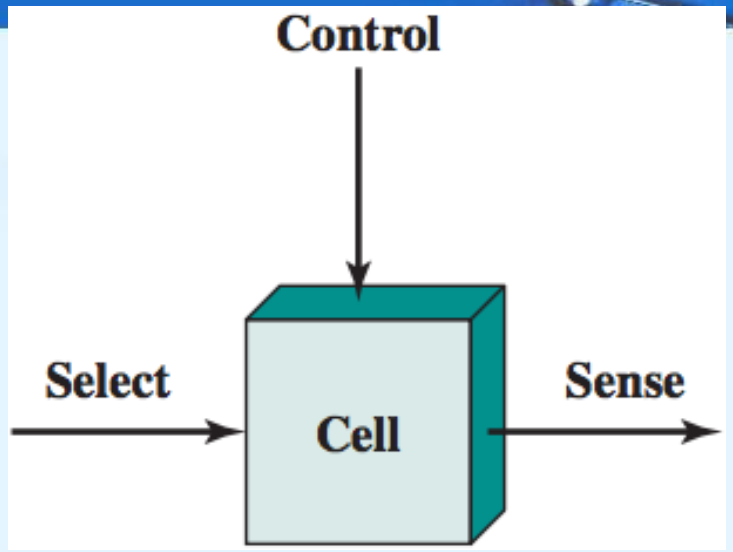
Cell has three terminals capable of carrying an electrical signal:

- **select:** selects a memory cell for a read / write operation;
- **control:** indicates whether a read / write operation is being performed;
- **read/write:**
  - For writing, terminal sets the state of the cell to 1 or 0.
  - For reading, terminal is used for output of the cell's state.

Conti



Memory cell write operation



Memory cell read operation

Example Consider, a memory with a capacity of 1K words of 16 bits each:

- Capacity:
  - $1K \times 16 = 16Kbits = 2Kbytes$;
- Each word has an address: • 0 to 1023
- When a word is read or written:
  - memory acts on all 16 bits;

Contents of a 1024 x 16 Memory

Memory Address		
Binary	Decimal	Memory Contents
0000000000	0	10110101 01011100
0000000001	1	10101011 10001001
0000000010	2	00001101 01000110
	.	.
	.	.
	.	.
	.	.
	.	.
1111111101	1021	10011101 00010101
1111111110	1022	00001101 00011110
1111111111	1023	11011110 00100100

Write operation steps:

1 Apply the binary address of the desired word to the address lines.

2 Apply the data bits that must be stored in memory to the data input lines.

3 Activate the Write control line.

Memory unit will then transfer the bits to that address.

Read operation steps:

1 Apply the binary address of the desired word to the address lines.

2 Activate the Read control line.

3 Memory unit will then transfer the bits to the data output lines.

Memory unit operation is controlled by an external device :

- CPU is synchronized by its own clock pulses;
- Memory, however, does not employ a clock:
  - control signals are employed for read / write.
    - **Access time -** time required for specifying an address and obtaining the word;
    - **Write time -** time required for specifying an address and storing the word;
  - CPU must provide the control signals synchronized with its clock:
    - This implies that:
      - read / write operations will take a certain number of clock periods;
      - What does this mean? Lets see with an example =)

Example

Assume:

- CPU with a clock frequency of 50 Mhz;
- Memory access time: 65*ns*;
- Memory write time: 75*ns*

- How many clock pulses do we need for read / write operations?

Conti…

- CPU with a clock frequency of 50 Mhz:
- How long does one clock pulse take? Any Ideas?
- $f = 1/p$
- $p = 1/f$
- $p = 1/50Mhz$
- $p = 1/50 \times 10^{6}hz$
- $p = 2 \times 10^{-8}s = 20ns$

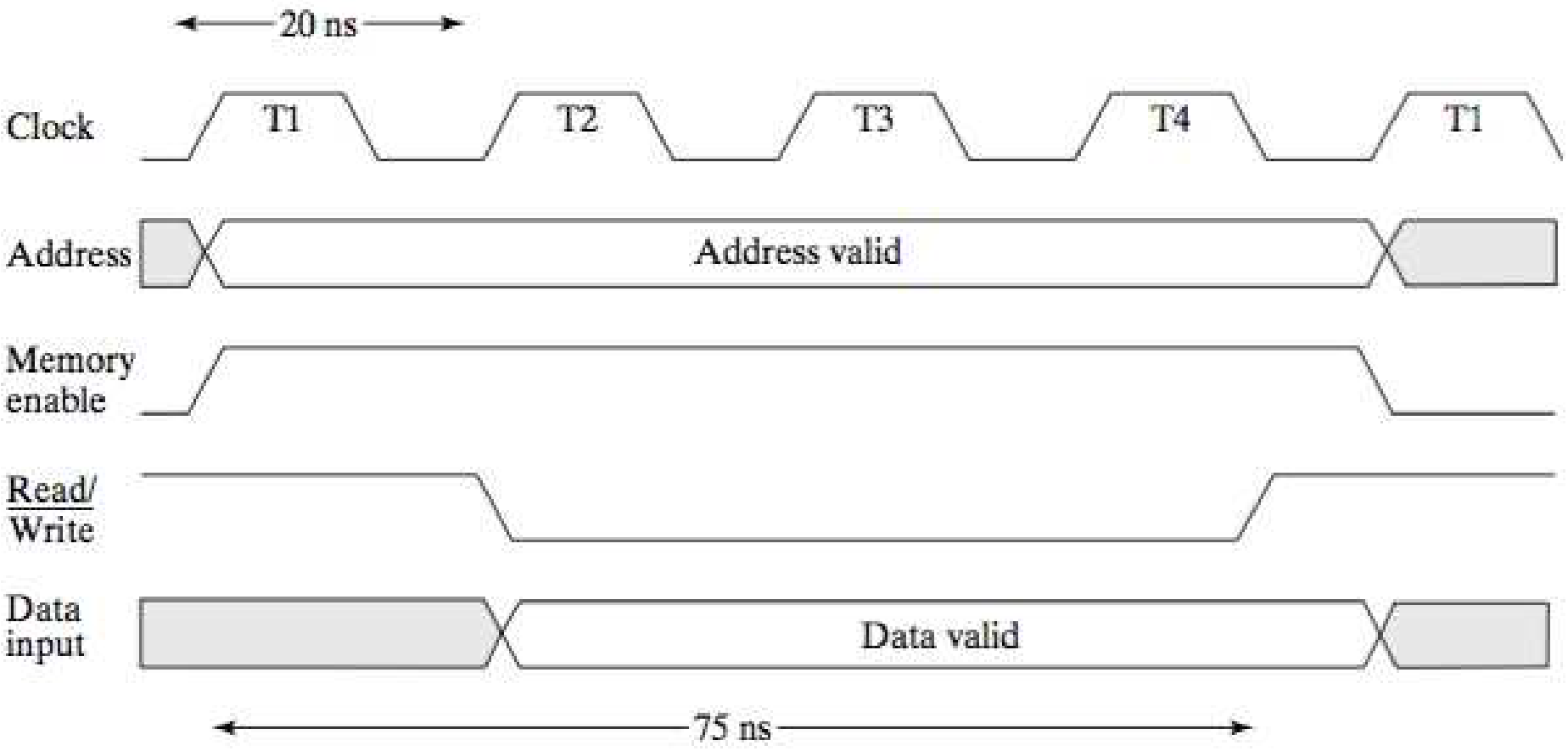
Conti

- CPU clock pulse: 20*ns*;
- Assume:
  - Memory access time: 65*ns*;
  - Memory write time: 75*ns*

How many clock pulses do we need for read / write operations?

- 4 clock pulses for read / write operations;

write operation



write operation:

1 CPU must provide the address (T1 pulse);

2 Memory enable is set (T1 pulse);

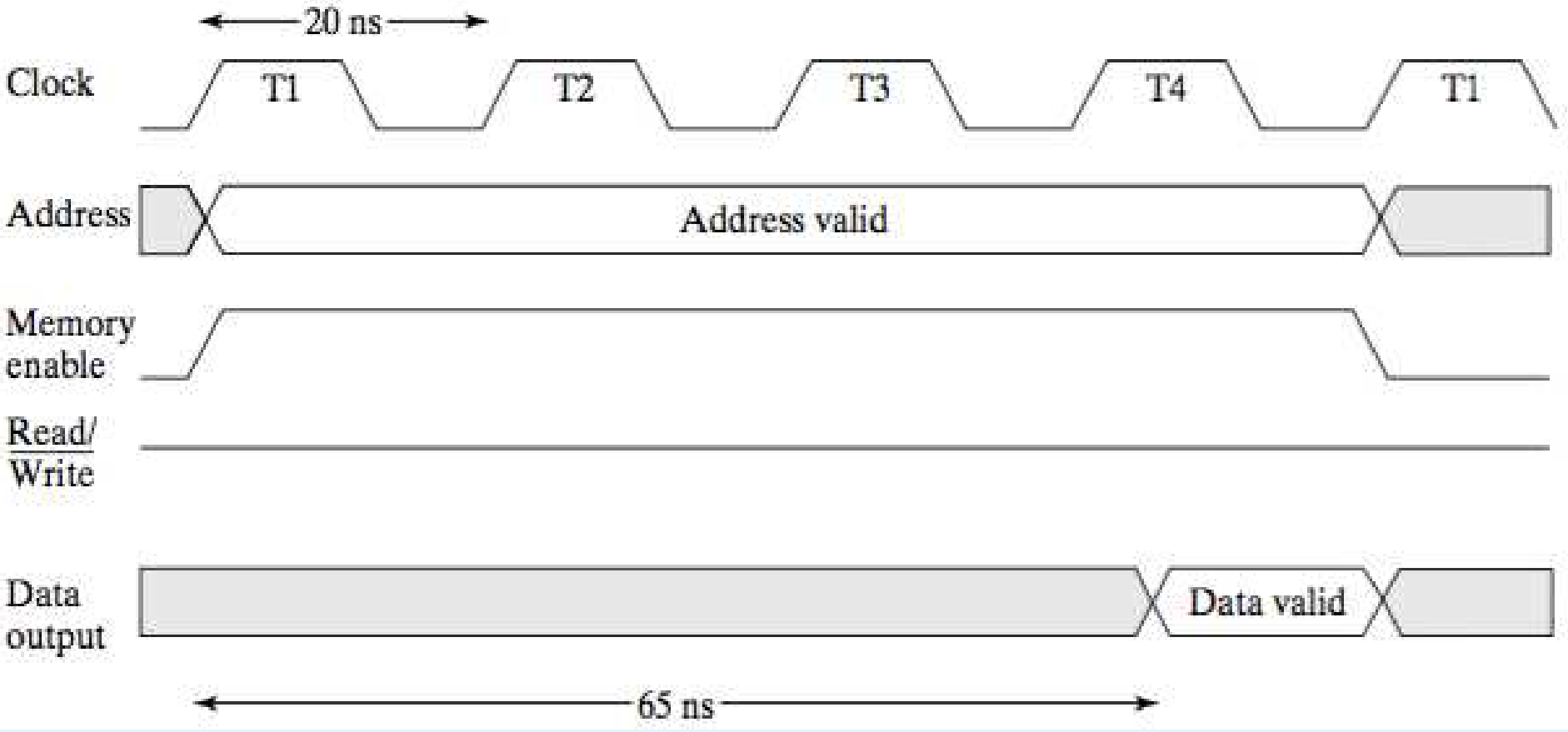
3 Data is supplied (T2 pulse);

4 *Read*/*Write* signal set to 0 for write operation (T2 pulse):

- CPU waits for T2 to let the address signals stabilize:
- Otherwise: wrong address may be used!
- Signal must stay activated long enough for the operation to finish;

5 When T4 completes, write operation has ended with 5*ns* to spare

read operation:



read operation:

1 CPU must provide the address (T1 pulse);

2 Memory enable is set (T1 pulse);

3 *Read*/*Write* signal set to 1 for read operation (T2 pulse):

- Signal must stay activated long enough for the operation to finish;
- Selected word is placed onto the data output lines;

4 When T4 completes, write operation has ended with 15*ns* to spare

Major types of semiconductor memory

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

Random-access memory

- Randomly access any possible address;
- Possible both to read and write data;
- Volatile: requires a power supply;
- For a chip with *m* words with *n* bits per word:
  - Consists of an array with $m \times n$ binary storage cells
- *E.g.:* DRAM and SRAM.
  - Lets have a look at these =)

Dynamic RAM (DRAM)

Made with **cells** that store data as charge on capacitors:

- Capacitor charge presence or absence is interpreted as a binary 1 or 0;
- Capacitors have a natural tendency to discharge;
- Requires periodic charge refreshing to maintain data storage

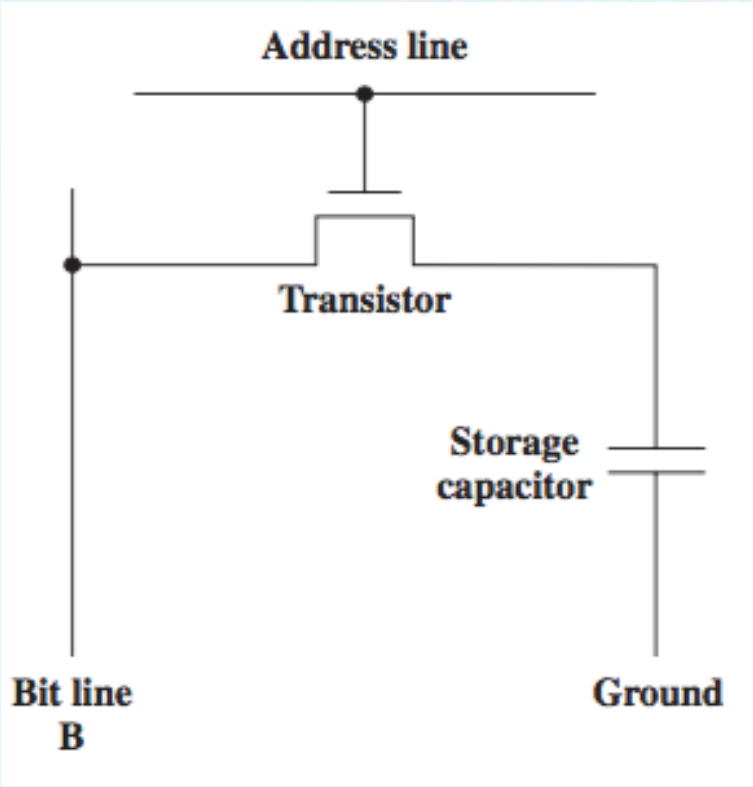
DRAM Cell components:

If voltage goes to the address line:

- **Transistor closes:**
- Current flows to capacitor;

If **no** voltage goes to the address line:

- **Transistor opens:**
- No current flows to capacitor;

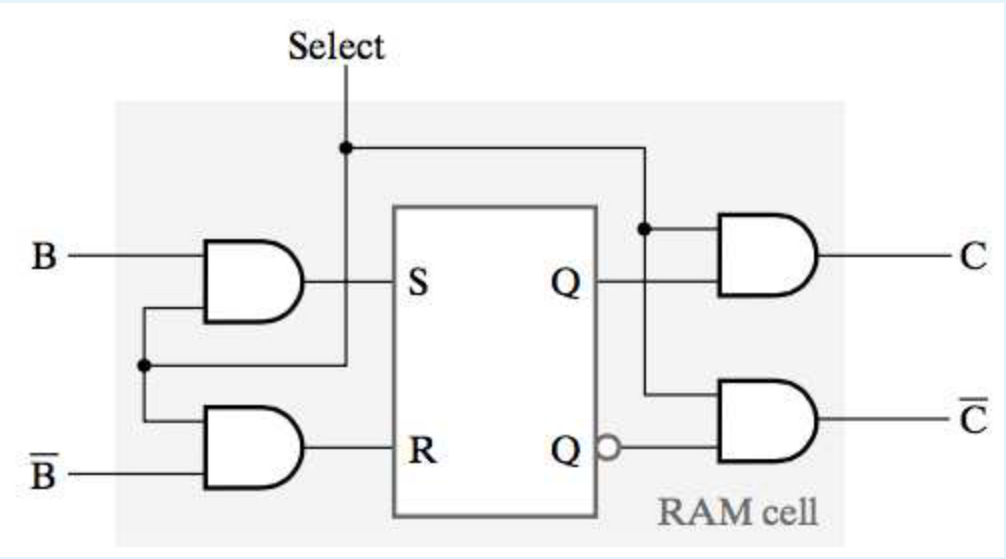


Static RAM (SRAM)

Binary values are stored using **SR flip-flop** configurations:

- Remember flip-flops? We saw them at the beginning of the semester...
- Same logic elements used in the processor registers;
- will hold its data as long as power is supplied to it.

Cell storage is modeled by an SR flip-flop:



SRAM Cell

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	—

SR Flip flop table

- Flip-flop **inputs** are enabled by a **select** (S) signal:
  - *S* = 0, stored content is held;
  - *S* = 1, stored content is determined by *B* and *B'*
- Flip-flop **outputs** are enabled by a **select** (S) signal:
  - *S* = 0, both *C* and *C'* are 0;
  - *S* = 1, *C* is the stored value and *C'* is its complement.

Pros / Cons of DRAM and SRAM

- Both are volatile: power must be continuously supplied to preserve the bit values;
- DRAMs: Periodically refresh capacitor's charge;
- SRAM: No need to periodically refresh;
- DRAM cell is simpler and smaller than a SRAM cell: (1 transistor, 1 capacitor) vs. SR flip-flop
- DRAM is denser (smaller cells = more cells per unit area) ;
- • Due to its simplicity: DRAM is cheaper than corresponding SRAM;

Conti…

- DRAM requires the supporting refresh circuitry (readout):
  - This is a fixed cost, does not increase with size =)
  - This cost is more than compensated by the smaller cost of DRAM cells;
  - **Thus, DRAMs tend to be favored for large memory requirements;**
- SRAMs are faster and more expensive than DRAMs
  - No need to refresh the circuitry after a readout;
- Because of these characteristics:
  - **DRAM:** used in main memory;& **SRAM:** used in cache;

Conti…

Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

Read-only memory (ROM)

- Contains a permanent pattern of data that cannot be changed;
  - Therefore:
    - Possible to read a ROM;
    - **Not possible to write data more than once;**
- Advantage:
  - Data / Program is permanently in main memory;
  - No need to load from a secondary storage device;
  - Original BIOS were stored in ROM chips.
- Nonvolatile: no power source is required;

Conti…

Not possible to write data more than once:

- Data is wired into the chip as part of the fabrication process
- Data insertion is expensive;
- No room for error:
  - If one bit is wrong, the whole batch of ROMs must be thrown out.

Different types of ROM

Programmable ROM (PROM):

- nonvolatile and may be written into only once;
- Writing may be performed at a later time than the original chip fabrication.
- Special equipment is required for the writing process;
- More expensive than ROM;

Erasable programmable read-only memory (EPROM):

- Can be altered multiple times;
- Entire memory contents need to be erased;
- Special equipment is required for the erasure process: Ultra-violet radiation;
- Slow erasure procedure (>20 minutes); & More expensive than PROM;

Conti…

Electrically erasable programmable read-only memory (EEPROM):

- Can be altered multiple times;
- Special equipment is required for the erasure process:
  - **No need to erase prior contents...only the byte or bytes addressed are updated.**
  - Ultra-violet radiation;
- Faster erasure procedure than EPROM;
- More expensive than EPROM;

Conti…

Flash memory:

- named because of the speed with which it can be reprogrammed;
- between EPROM and EEPROM in both cost and functionality:
  - Uses an electrical erasing technology;
  - Possible to erase blocks of memory;
  - Does not provide byte-level erasure;
- Technology used in pendrives;