Memory Hierarchy

COMPUTE SYSTEMS ORGANIZATION SPRING 2021

Flat Memory v/s Hierarchical Memory

- To this point in our study of systems, we have used a flat memory model wherein the access time for memory location (X) = access time for memory location (Y), X != Y.
- In a Hierarchical Memory design access time for memory location (X) != or = access time for memory location (Y), X != Y, depending on which memory X/Y is stored with respect to the hierarchy.

Mem1_Access_Time > Mem2_Access_Time > Mem3_Access_Time

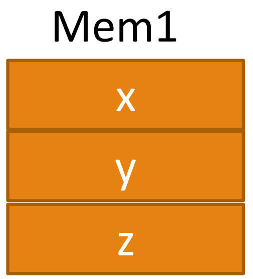


Fig1:
Flat Memory Storage

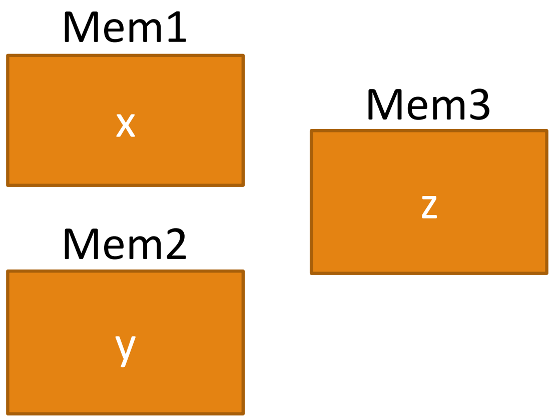
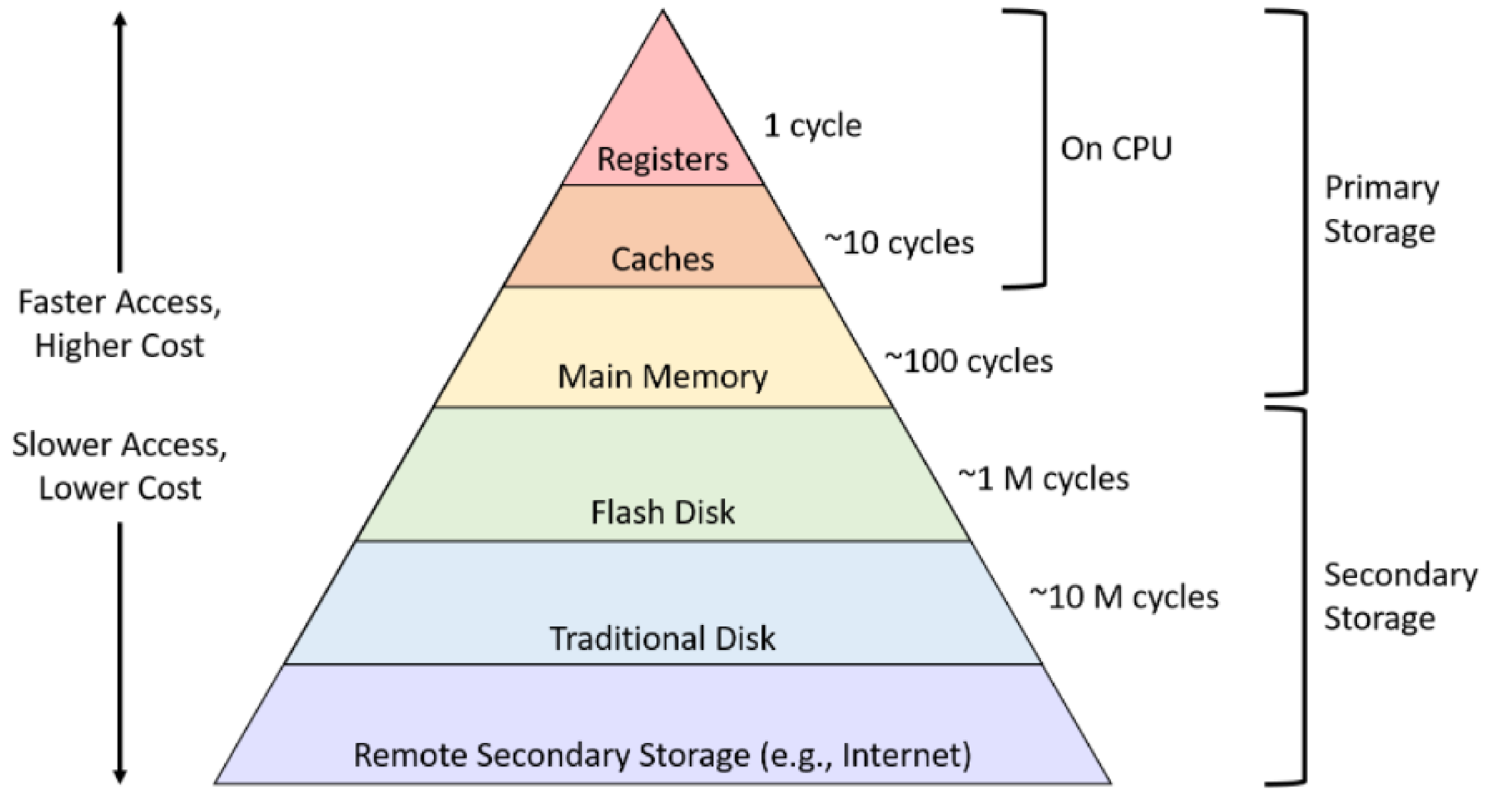


Fig2:
Hierarchical Memory Storage

```
void fun()
{
        loop:
                access(z)
        access(x)
        user_input_access(y)
}
```

Which Memory design leads to lower execution latency for fun() ?

The Memory Pyramid



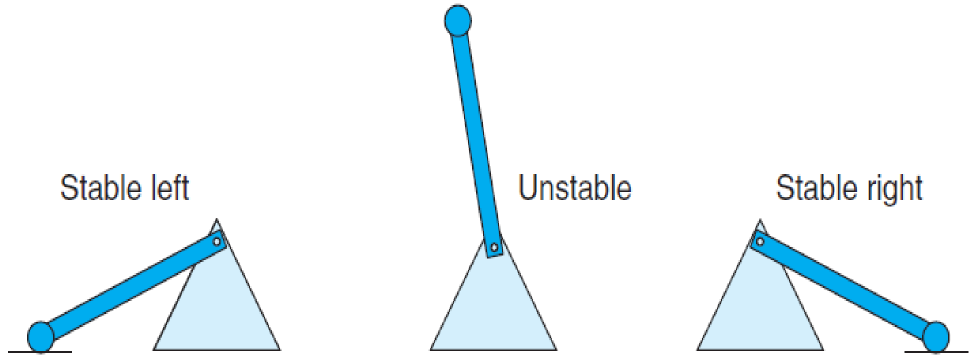
The Memory Hierarchy

Different Memory Technologies are used to implement different layers in the memory hierarchy

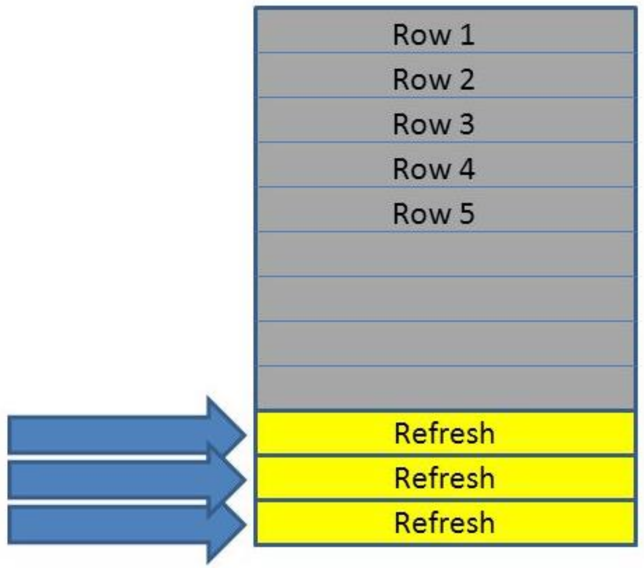
Level	Technology
Register	Hardware Register
Cache	SRAM
Main Mem	DRAM
Flash Disk	SDD/EEPROM
Traditional Disk	HDD
Remote	Web Server

SRAM and DRAM

- SRAM stores each bit in a *bistable* memory cell. Each cell is implemented with a six-transistor circuit that can stay indefinitely in either of two different voltage configurations, or *states*.
- DRAM stores each bit as charge on a capacitor. The memory system must periodically refresh every bit of DRAM memory by reading it out and then rewriting it.

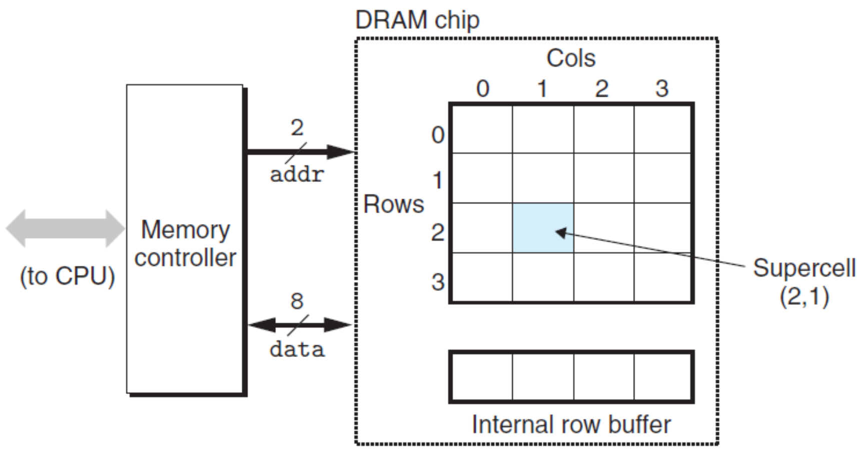


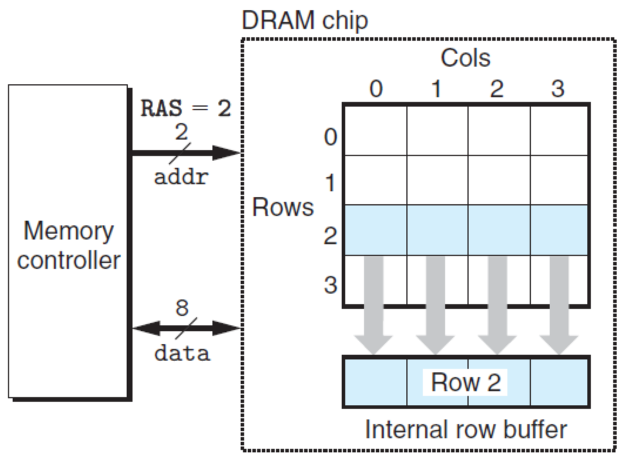
	Transistors per bit	Relative access time	Persistent?	Sensitive?	Relative cost	Applications
SRAM	6	1×	Yes	No	100×	Cache memory
DRAM	1	10×	No	Yes	1×	Main mem, frame buffers



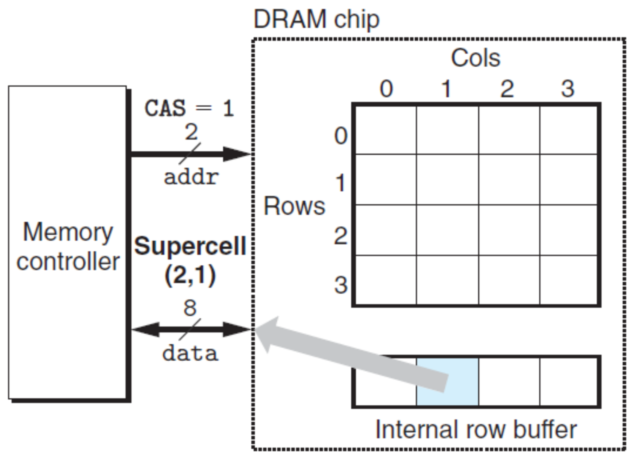
DRAM Structure and Design

- The cells (bits) in a DRAM chip are partitioned into *d supercells*, each consisting of *w* DRAM cells. A *d* × *w* DRAM stores a total of *dw* bits of information.
- The supercells are organized as a rectangular array with *R* rows and *C* columns, where *R x C* = *d*. Each supercell has an address of the form *(i, j)*, where *i* denotes the row, and *j* denotes the column.





(a) Select row 2 (RAS request).

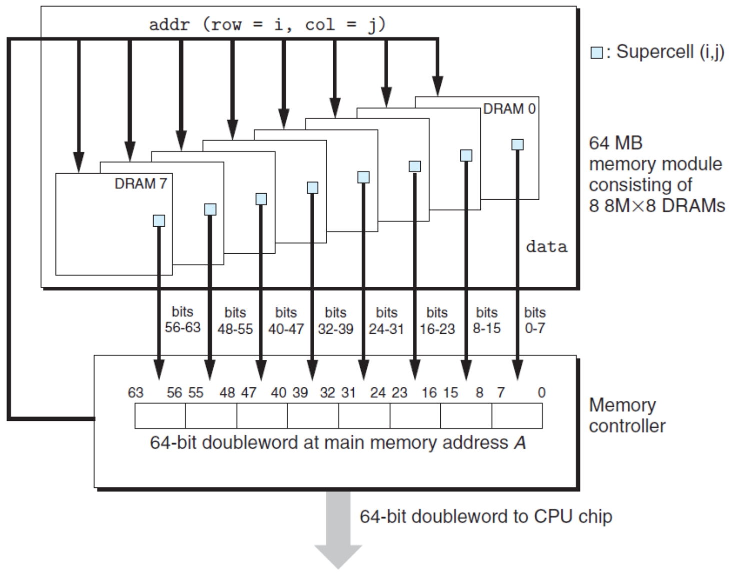


(b) Select column 1 (CAS request).

Worked Out Example

- Figure shows the basic idea of a memory module. The example module stores a total of 64 MB (megabytes) using eight 64-Mbit 8*M* × 8 DRAM chips, numbered 0 to 7. Each supercell stores 1 byte of *main memory*, and each 64-bit doubleword1 at byte address *A* in main memory is represented by the eight supercells whose corresponding supercell address is *(i, j)*.

- To retrieve a 64-bit doubleword at memory address *A*, the memory controller converts *A* to a supercell address *(i, j)* and sends it to the memory module, which then broadcasts *i* and *j* to each DRAM. In response, each DRAM outputs the 8-bit contents of its *(i, j)* supercell.



2 Problems

1. A main memory unit with a capacity of 4 megabytes is built using 1M × 1-bit DRAM chips. Each DRAM chip has 1K rows of cells with 1K cells in each row. The time taken for a single refresh operation is 100 nanoseconds. The time required to perform one refresh operation on all the cells in the memory unit is

2. Let *r* be the number of rows in a DRAM array, *c* the number of columns, *br* the number of bits needed to address the rows, and *bc* the number of bits needed to address the columns. For a 128 ×8-bit DRAM, determine the power-of-two array dimensions that minimize max*(br, bc)*, the maximum number of bits needed to address the rows or columns of the array.

Main Memory Access

- Data flows back and forth between the processor and the DRAM main memory over shared electrical conduits called *buses*. Each transfer of data between the CPU and memory is accomplished with a series of steps called a *bus transaction*.
- A *read transaction* transfers data from the main memory to the CPU. A *write transaction* transfers data from the CPU to the main memory.

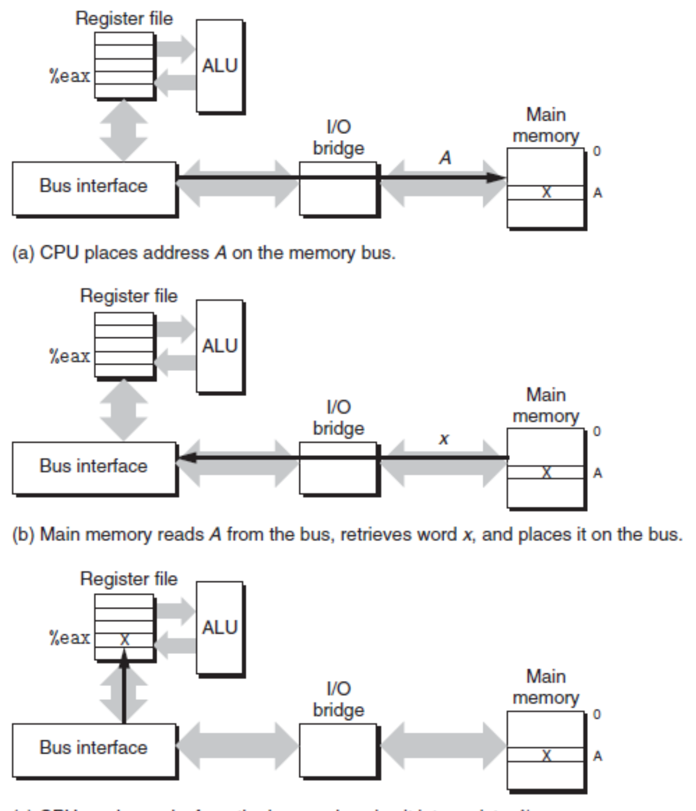

Register file
%eax
ALU
Bus interface
I/O bridge
Main memory
A
0
x
(a) CPU places address A on the memory bus.
(b) Main memory reads A from the bus, retrieves word x, and places it on the bus.
(c) CPU reads word x from the bus, and copies it into register %eax.

Figure 6.7 **Memory read transaction for a load operation:** `movl A,%eax`.

Disk Storage

- *Disks* are workhorse storage devices that hold enormous amounts of data, on the order of hundreds to thousands of gigabytes, as opposed to the hundreds or thousands of megabytes in a RAM-based memory. [https://www.youtube.com/watch?v=oEORcCQ62nQ&list=WL&index=219]

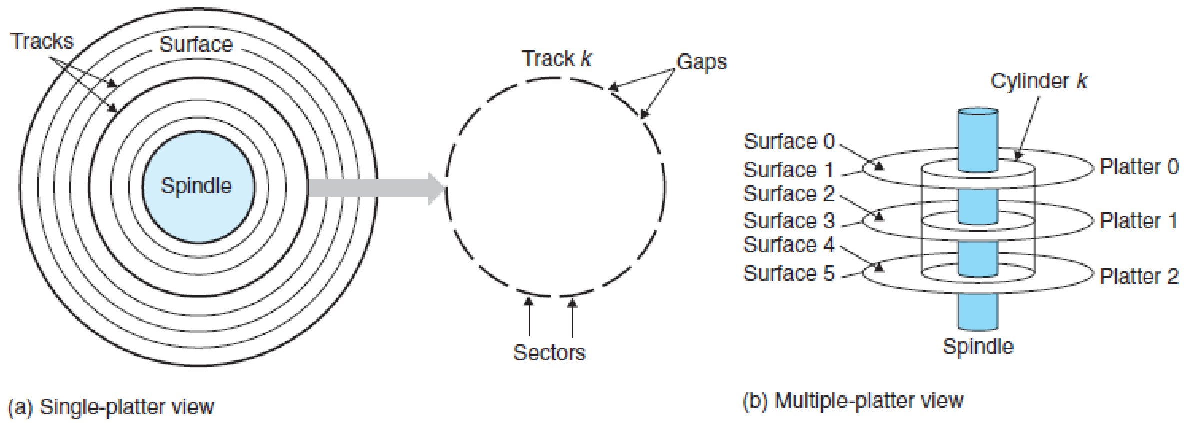


Figure 6.9 **Disk geometry.**

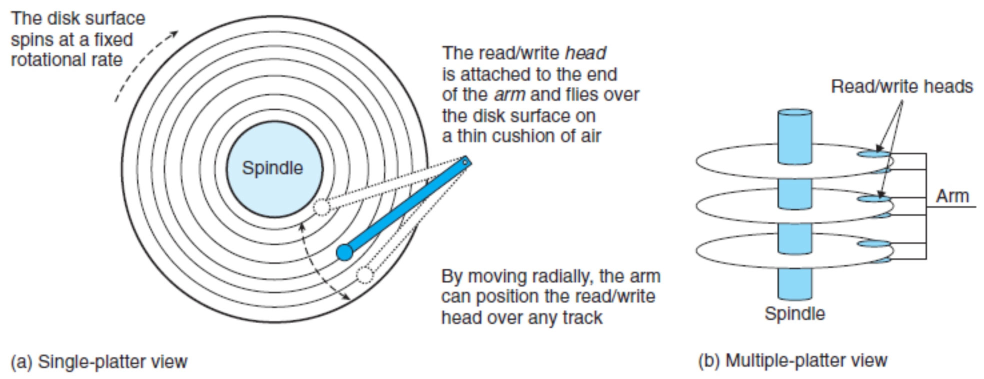

The disk surface spins at a fixed rotational rate
Spindle
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air
By moving radially, the arm can position the read/write head over any track
(a) Single-platter view
Read/write heads
Arm
Spindle
(b) Multiple-platter view

Figure 6.10 **Disk dynamics.**

$$\text{Disk capacity} = \frac{\#\text{ bytes}}{\text{sector}} \times \frac{\text{average }\#\text{ sectors}}{\text{track}} \times \frac{\#\text{ tracks}}{\text{surface}} \times \frac{\#\text{ surfaces}}{\text{platter}} \times \frac{\#\text{ platters}}{\text{disk}}$$

Disk Access

- Seek Time (TS) - To read the contents of some target sector, the arm first positions the head over the track that contains the target sector. The time required to move the arm is called the *seek time*.
- Average Rotational Latency (ARL) – Average Time to reach a sector on a track.

The ARL is half this value.

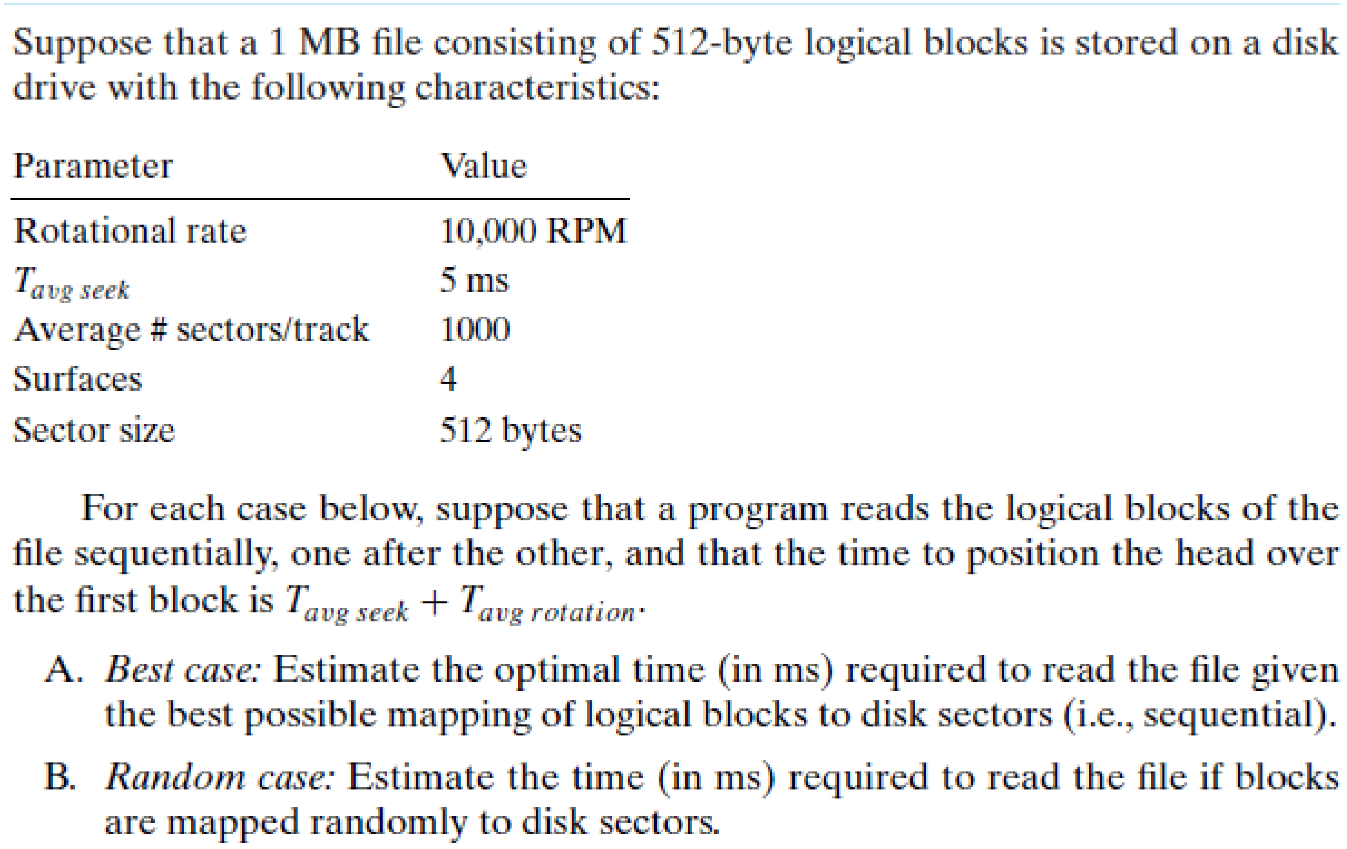
$$T_{max\ rotation} = \frac{1}{\text{RPM}} \times \frac{60\ \text{secs}}{1\ \text{min}}$$

- Transfer time (TR) – How fast can the disk read a sector.

$$T_{avg\ transfer} = \frac{1}{\text{RPM}} \times \frac{1}{(\text{average \# sectors/track})} \times \frac{60\ \text{secs}}{1\ \text{min}}$$

Disk Latency = TS + ARL + TR

Problem



Suppose that a 1 MB file consisting of 512-byte logical blocks is stored on a disk drive with the following characteristics:

Parameter	Value
Rotational rate	10,000 RPM
$T_{avg\ seek}$	5 ms
Average # sectors/track	1000
Surfaces	4
Sector size	512 bytes

For each case below, suppose that a program reads the logical blocks of the file sequentially, one after the other, and that the time to position the head over the first block is $T_{avg\ seek} + T_{avg\ rotation}$.

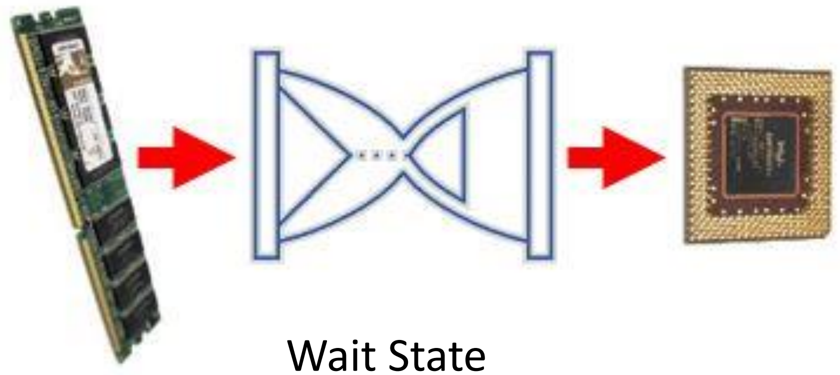
A. *Best case:* Estimate the optimal time (in ms) required to read the file given the best possible mapping of logical blocks to disk sectors (i.e., sequential).

B. *Random case:* Estimate the time (in ms) required to read the file if blocks are mapped randomly to disk sectors.

Memory Pyramid Revisited

There are two aspects to Memory Pyramid.

- Different storage technologies have widely different access times. Faster technologies cost more per byte than slower ones and have less capacity. The gap between CPU and main memory speed is widening.
- Well-written programs tend to exhibit good locality.



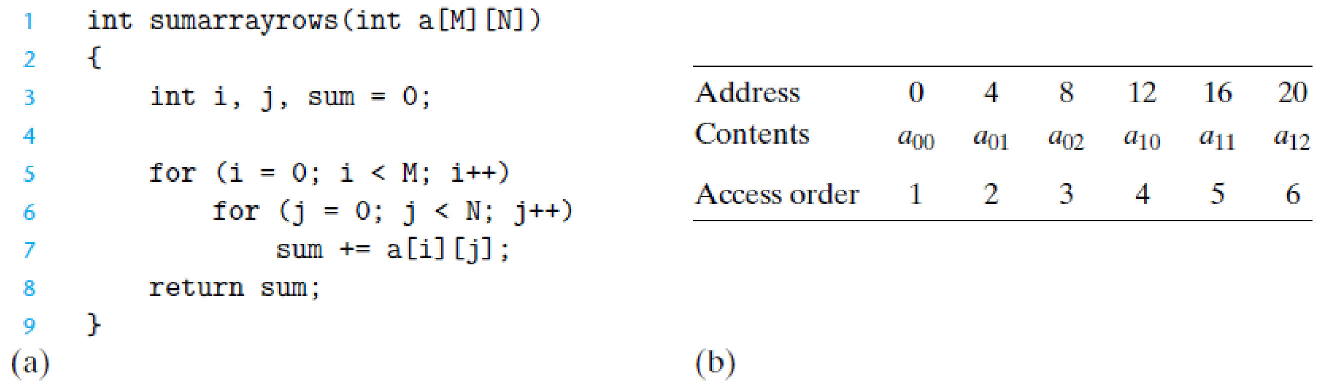
Wait State

Consider a 33MHz CPU based system.
What is the number of wait states required if it is interfaced with a 60ns memory?
Assume a maximum of 10ns delay for additional circuitry like buffering and decoding. [ISRO 2014]

Locality To Reduce Wait States.

- Well-written computer programs tend to exhibit good *locality*.
- they tend to reference data items that are near other recently referenced data items, or that were recently referenced themselves.

Degree of Spatial and Temporal Locality depends on the loop count and the stride value



```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

(a)

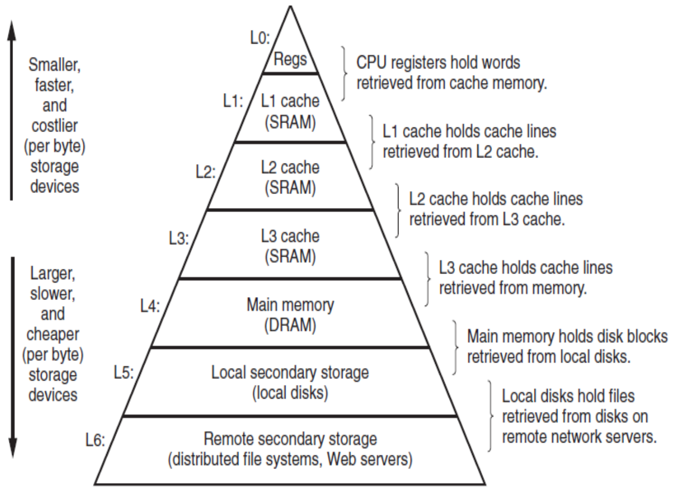
Address	0	4	8	12	16	20
Contents	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}
Access order	1	2	3	4	5	6

(b)

Figure 6.20 **(a) Another function with good locality. (b) Reference pattern for array a ($M = 2$, $N = 3$).** There is good spatial locality because the array is accessed in the same row-major order in which it is stored in memory.

Principle of Caching

- In general, a ***cache*** is a small, fast storage device that acts as a staging area for the data objects stored in a larger, slower device.
- The central idea of a memory hierarchy is that for each k, the faster and smaller storage device at level k serves as a cache for the larger and slower storage device at level k + 1.


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0: Regs
L1: L1 cache (SRAM)
L2: L2 cache (SRAM)
L3: L3 cache (SRAM)
L4: Main memory (DRAM)
L5: Local secondary storage (local disks)
L6: Remote secondary storage (distributed file systems, Web servers)
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from L2 cache.
L2 cache holds cache lines retrieved from L3 cache.
L3 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network servers.

Hit/Miss

- Cold miss
- Conflict miss
- Capacity miss

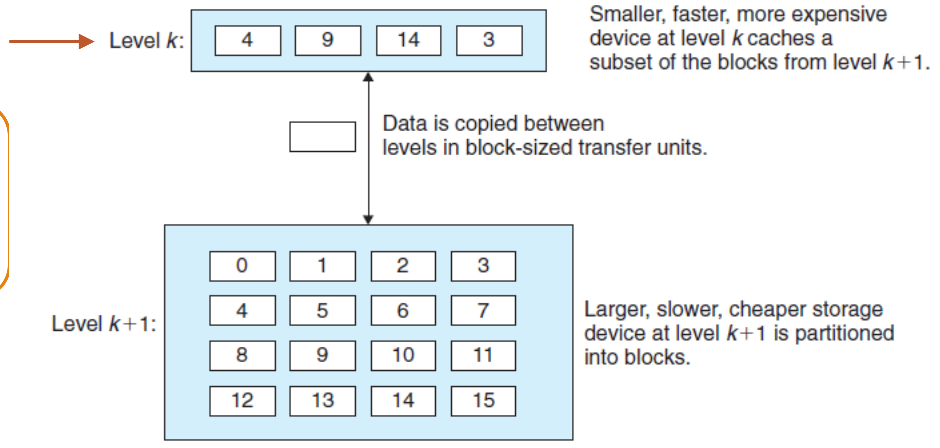
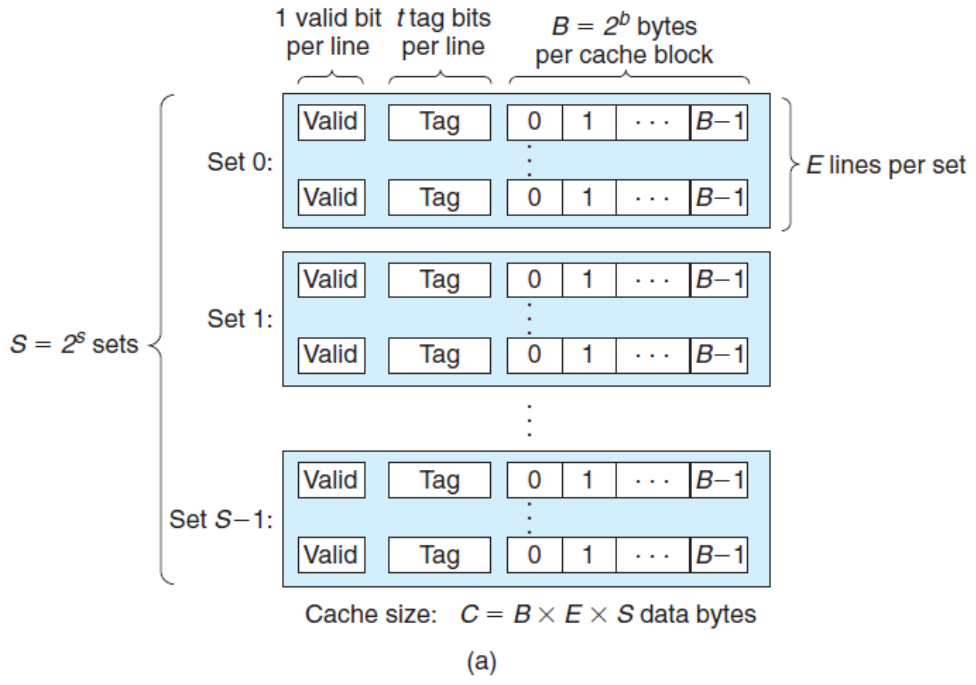
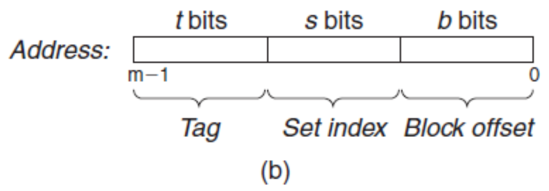

Level k:
4 9 14 3
Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1.
Data is copied between levels in block-sized transfer units.
Level k+1:
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.

Figure 6.24 The basic principle of caching in a memory hierarchy.

Cache Memory





Fundamental parameters	
Parameter	Description
$S = 2^s$	Number of sets
E	Number of lines per set
$B = 2^b$	Block size (bytes)
$m = \log_2(M)$	Number of physical (main memory) address bits

Derived quantities	
Parameter	Description
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of *set index bits*
$b = \log_2(B)$	Number of *block offset bits*
$t = m - (s + b)$	Number of *tag bits*
$C = B \times E \times S$	Cache size (bytes) not including overhead such as the valid and tag bits

Figure 6.28 **Summary of cache parameters.**

Worked out Example-1

- A block-set associative cache memory consists of 128 blocks divided into four block sets . The main memory consists of 16,384 blocks and each block contains 256 eight-bit words. How many bits are needed to represent the TAG, SET and WORD fields?

Worked out Example-2

- A computer has a 256 KB, 4-way set associative, write back data cache with block size of 32 bytes. The processor sends 32-bit addresses to the cache controller. Each cache tag directory entry contains in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit. What is the number of bits in the TAG field and the size of the TAG directory of the cache in bits.

Worked Out Example 3

In a k-way set associative cache, the cache is divided into v sets, each of which consists of k lines. The lines of a set are placed in sequence one after another. The lines in set s are sequenced before the lines in set (s+1). The main memory blocks are numbered 0 onwards. The main memory block numbered j must be mapped to any one of the cache lines from.

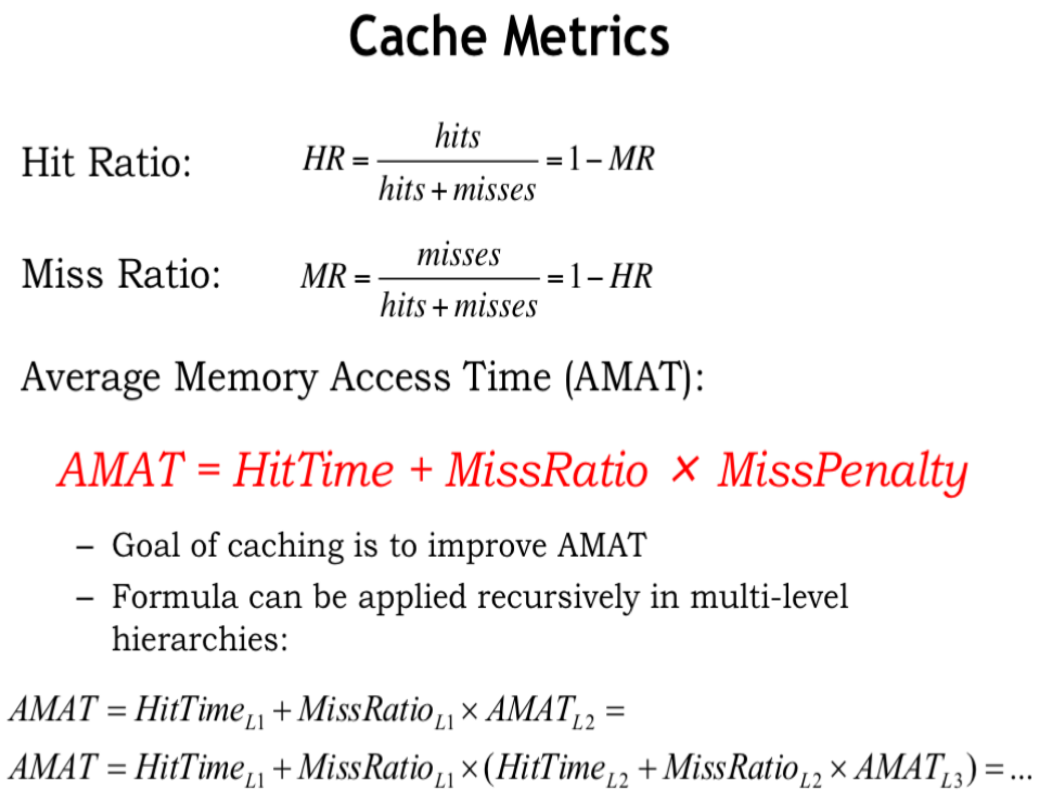
(A) (j mod v) * k to (j mod v) * k + (k-1)
(B) (j mod v) to (j mod v) + (k-1)
(C) (j mod k) to (j mod k) + (v-1)
(D) (j mod k) * v to (j mod k) * v + (v-1)

Worked out Example-4 Eviction Policy

- Here is a series of address references given as word addresses: 1, 4, 8, 5, 20, 17. Assuming a direct-mapped cache with 16 one-word blocks that is initially empty, label each reference in the list as a hit or a miss and show the final contents of the cache. #of blocks = 16.

- Most Recently used
- Least Recently Used
- Least Frequently Use

Cache Access Latency/AMAT



Cache Metrics

Hit Ratio: $HR = \frac{hits}{hits + misses} = 1 - MR$

Miss Ratio: $MR = \frac{misses}{hits + misses} = 1 - HR$

Average Memory Access Time (AMAT):

$$AMAT = HitTime + MissRatio \times MissPenalty$$

- Goal of caching is to improve AMAT
- Formula can be applied recursively in multi-level hierarchies:

$$AMAT = HitTime_{L1} + MissRatio_{L1} \times AMAT_{L2} =$$

$$AMAT = HitTime_{L1} + MissRatio_{L1} \times (HitTime_{L2} + MissRatio_{L2} \times AMAT_{L3}) = \ldots$$

- If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?AMAT = Hit time + Miss rate x Miss penalty = 4 + 0.05 x 100 = 9 ns.

- If replacing the cache with a 2-way set associative increases the hit rate to 97%, but increases the hit time to 5 ns, what is the new AMAT?AMAT = Hit time + Miss rate x Miss penalty = 5 + 0.03 x 100 = 8 ns

Problem 1

A direct mapped cache memory of 1 MB has a block size of 256 bytes. The cache has an access time of 3 ns and a hit rate of 94%. During a cache miss, it takes 20 ns to bring the first word of a block from the main memory, while each subsequent word takes 5 ns. The word size is 64 bits. The average memory access time in ns (round off to 1 decimal place) is ?

Cache Access Latency

- Consider a system with 2 level caches. Access times of Level 1 cache, Level 2 cache and main memory are 1 ns, 10ns, and 500 ns, respectively. The hit rates of Level 1 and Level 2 caches are 0.8 and 0.9, respectively. What is the average access time of the system ignoring the search time within the cache?