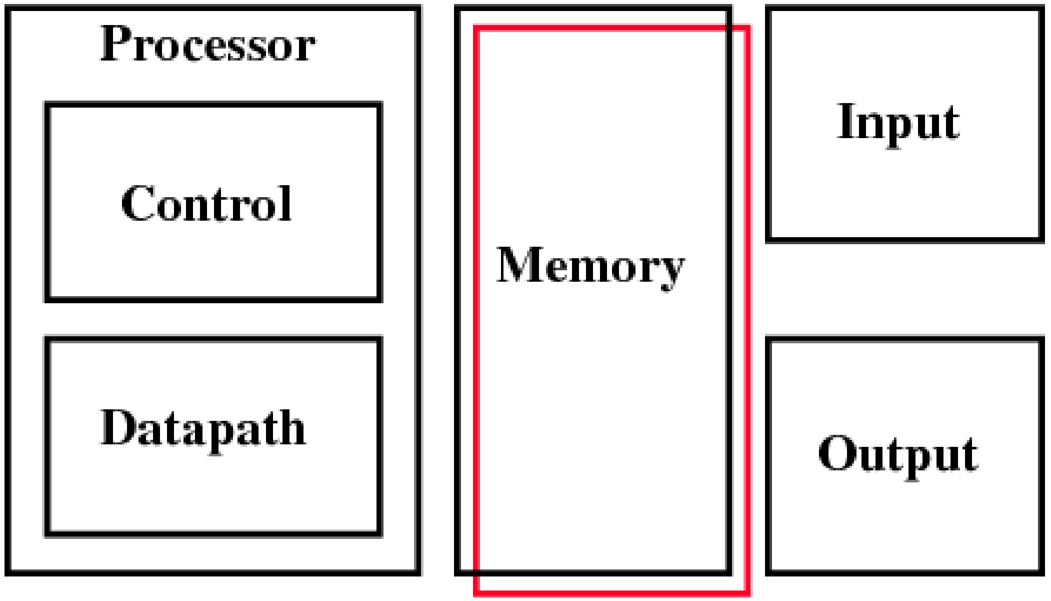
COMPUTER SYSTEMS ORGANIZATION

Memory Hierarchies -- Spring 2010 -- IIIT-H -- Suresh Purini

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer



- Today's Topic: Memory System

Source: Dave Patterson

Desired Features for Memory

Memory should be

- Large – So that programs with huge memory footprint can be executed.
- Fast – Why?

However, in real world these two are contradictory requirements!

Random Access Memory (RAM)

Main Memory – DRAM
Cache – SRAM

- Main memory can be logically viewed as an array of bytes
- In RAMs, time taken to access any byte of memory is exactly same.
  - Compare this against Disks, Magnetic Tapes, CD-Rom
- Technologies for RAM
  - Static RAM (SRAM)
    - Low density, high power, expensive, fast
    - Static: content will last “forever”
  - Dynamic RAM (DRAM)
    - High density, low power, cheap, slow
    - Dynamic: need to be “refreshed” regularly

Random Access Memory (RAM) Technology

- Why do computer designers need to know about RAM technology?
  - Processor performance is usually limited by memory bandwidth
  - As IC densities increase, lots of memory will fit on processor chip
  - Tailor on-chip memory to specific needs
    - Instruction cache
    - Data cache
    - Write buffer
- What makes RAM different from a bunch of flip-flops?
  - Density: RAM is much more denser

Technology Trends

	Capacity	Speed
Logic:	2x in 3 years	2x in 3 years
DRAM:	4x in 3 years	1.4x in 10 years
Disk:	2x in 3 years	1.4x in 10 years

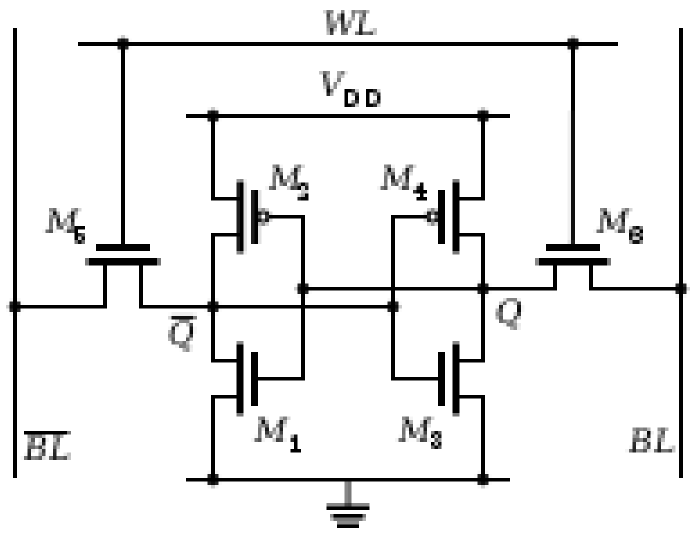
DRAM

Year	Size	Cycle Time
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns

Static RAM

- Low density, high power, expensive, fast
- Static: content will last “forever”

- SRAM stores each bit in a bistable memory cell
- Each cell is implemented with a six transistor circuit



- The above circuit can stay in one of two possible states indefinitely (as long as the circuit is powered)

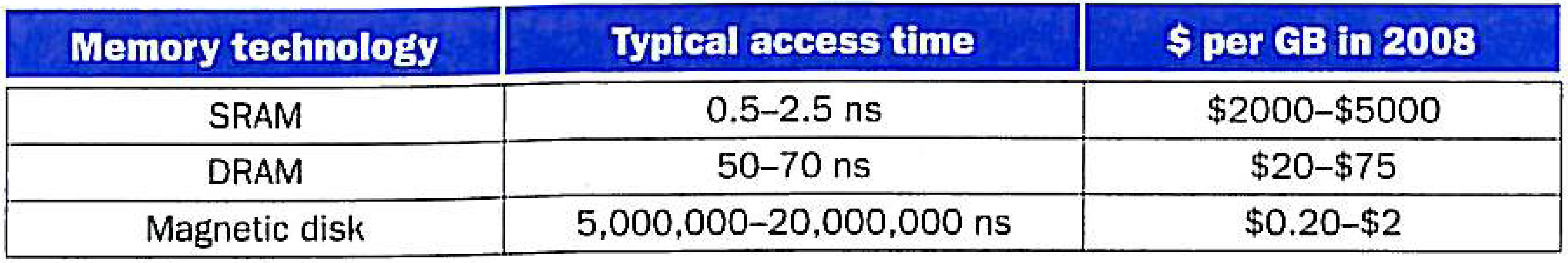
Dynamic RAM (DRAM)

- DRAM stores each bit as a charge on a capacitor.
- DRAM storage can be made very dense
- DRAM cell will lose its charge within a time period of around 10 to 100 milliseconds
- Memory system should be periodically refreshed by reading every bit of memory and writing it back

    - High density, low power, cheap, slow
    - Dynamic: need to be "refreshed" regularly

SRAM vs DRAM

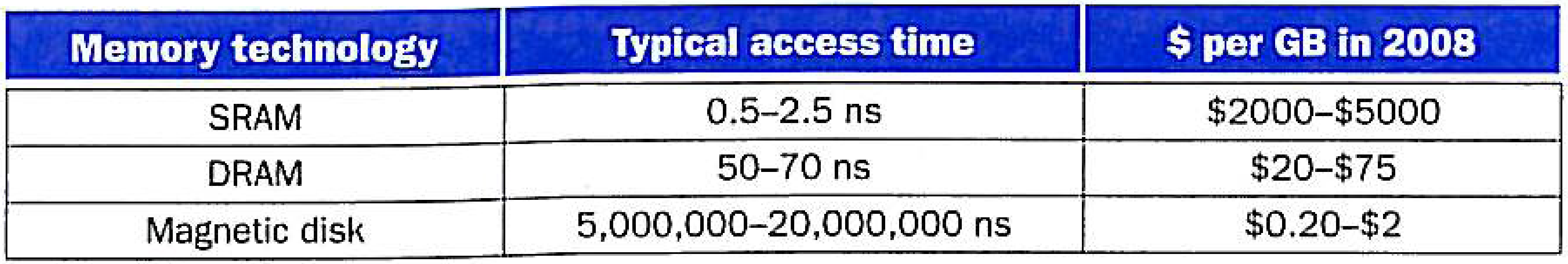
	Transistors per bit	Relative access time	Persistent?	Sensitive?	Relative cost	Applications
SRAM	6	1X	Yes	No	100X	Cache memory
DRAM	1	10X	No	Yes	1X	Main mem, frame buffers



Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5–2.5 ns	$2000–$5000
DRAM	50–70 ns	$20–$75
Magnetic disk	5,000,000–20,000,000 ns	$0.20–$2

The Illusion of Fast and Large Memories

- SRAMs – For small but fast Cache Memories
- DRAMs – For large but slow Main Memories
- Design Goal for Memory Hierarchy: Give the illusion of large and fast memory
- Key Idea: Exploit temporal and spatial locality of the programs to achieve the illusion.



Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5–2.5 ns	$2000–$5000
DRAM	50–70 ns	$20–$75
Magnetic disk	5,000,000–20,000,000 ns	$0.20–$2

Principle of Locality of Reference

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

Principle of Locality

Function sumvec exhibits

- Good temporal locality w.r.t variable sum
- Good spatial locality w.r.t variable v (Stride-1 reference pattern)
- Spatial locality decreases for Stride-k reference patterns as k increases

- Hey how about locality of instruction fetches?
- Loops exhibit good temporal and spatial locality

```
int sumvec(int v[N])
{
    int i, sum = 0;

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

(a)

N = 8

Address Contents	0 v_0	4 v_1	8 v_2	12 v_3	16 v_4	20 v_5	24 v_6	28 v_7
Access order	1	2	3	4	5	6	7	8

Principle of Locality

- Does the function sumarrayrows exhibit good spatial locality?
- What is the Stride for the array variable a?

```
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)

M = 2
N = 3

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

Principle of Locality

- Does the function sumarraycolumns exhibit good spatial locality?
- What is the Stride for the array variable a?

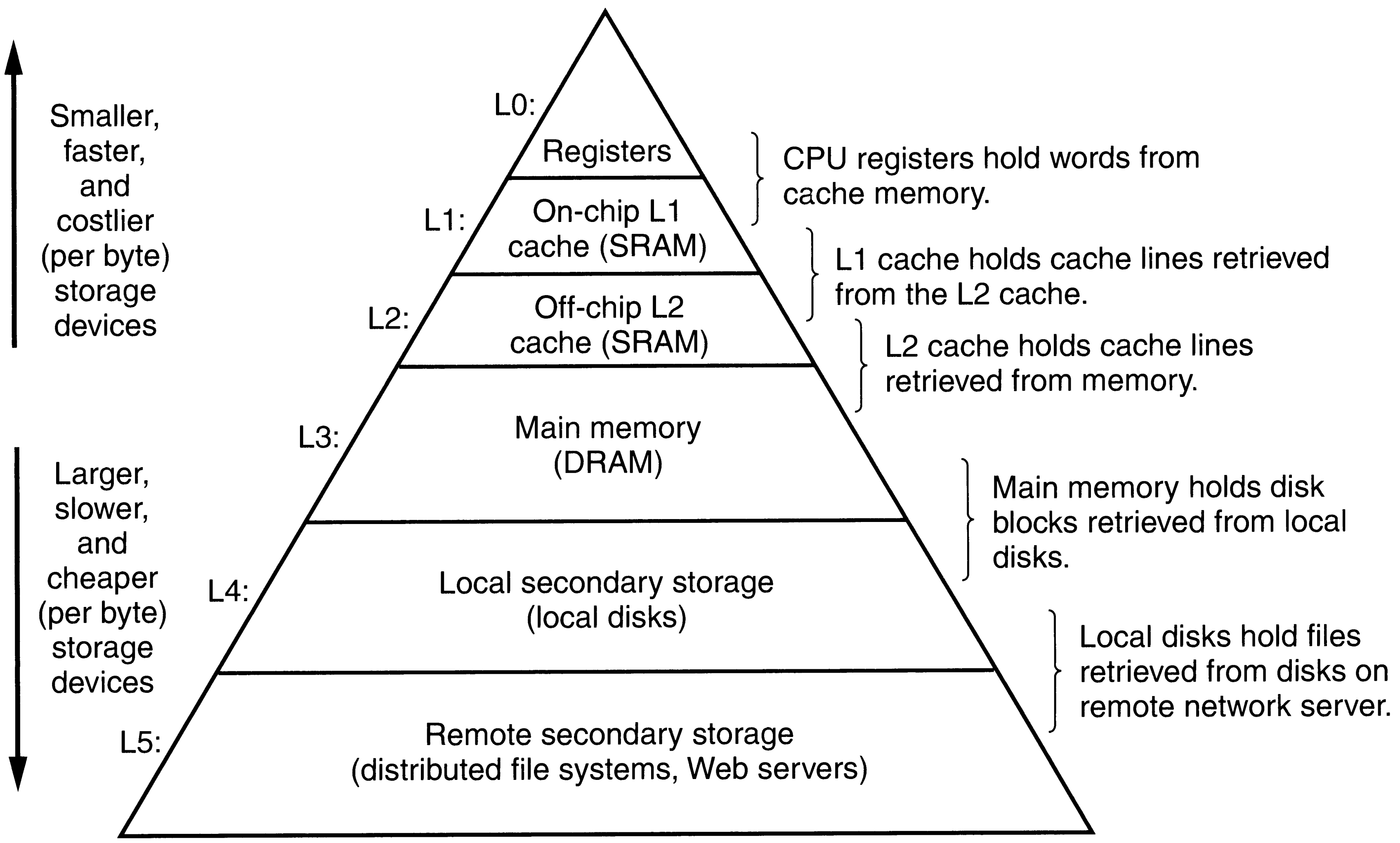
```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

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

(a)

M = 2
N = 3

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

Memory Hierarchy

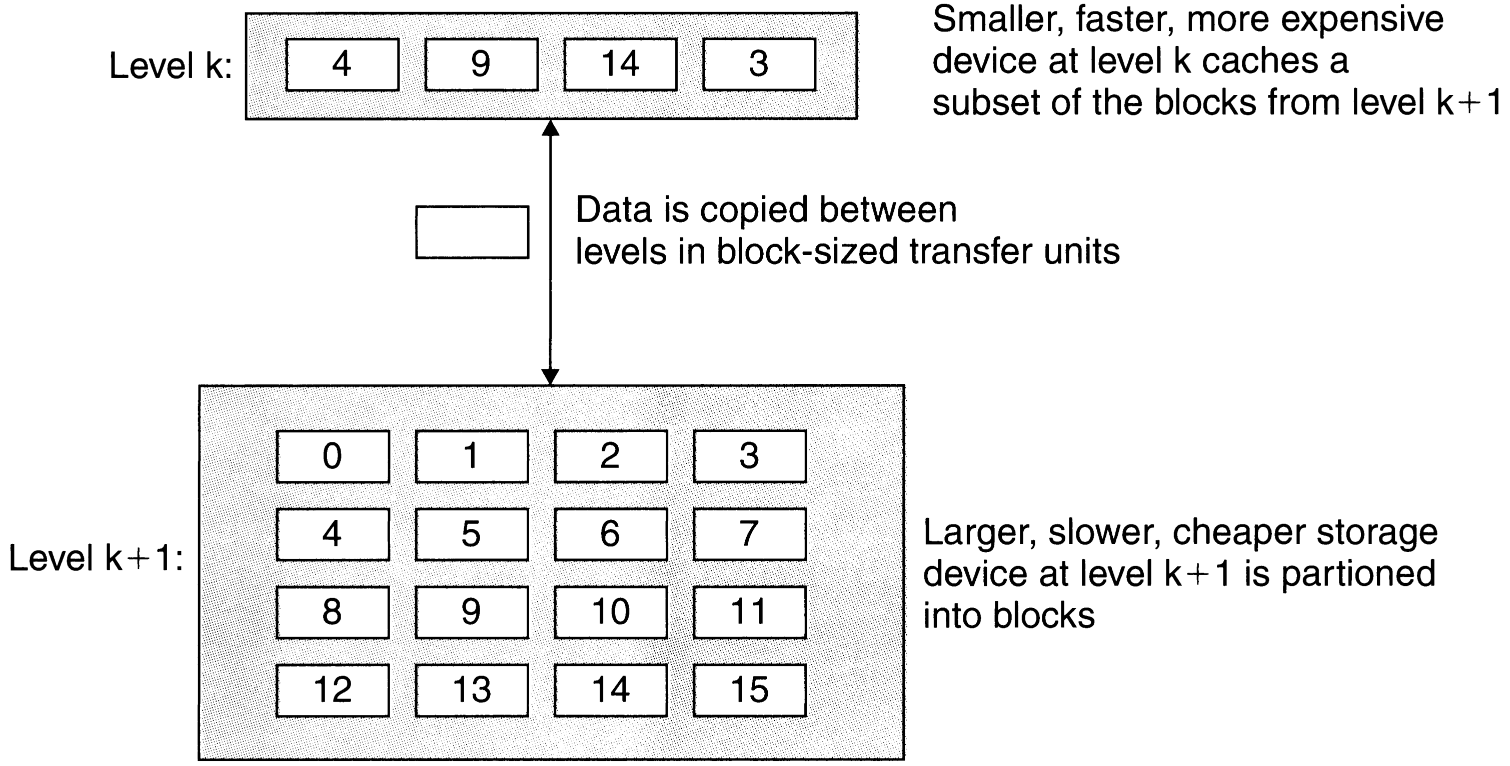

Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Registers
L1:
On-chip L1 cache (SRAM)
L2:
Off-chip L2 cache (SRAM)
L3:
Main memory (DRAM)
L4:
Local secondary storage (local disks)
L5:
Remote secondary storage (distributed file systems, Web servers)
CPU registers hold words from cache memory.
L1 cache holds cache lines retrieved from the L2 cache.
L2 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 server.

Basic Principle of Caching in the Memory Hierarchy

Focus of this lecture:

Level k: on-chip cache memory (SRAM)

Level k+1: Main memory (DRAM)


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 partioned into blocks

What is a Cache Hit and Cache Miss?

Let us say

- Cache size – 128 bytes (2^7 bytes)
- Cache line/block size – 32 bytes
- Main Memory 512 bytes (2^9 bytes)
- Main memory is also logically divided into 2^4 blocks each of size 32 bytes

When the processor accesses a word of memory

Cache hit: If the block containing the word is available in the Cache

Cache Miss: If the block containing the word is not available in Cache

Question: How to search the cache for the block containing the word of memory the processor is accessing?

What happens on a Cache Miss?

- The block containing the word of memory has to be fetched into the cache.
- We may have to overwrite an existing block if the cache is already full. This is called replacing or evicting the block.
- Question: Which block should we evict? (Cache's Replacement policy)
  - Random replacement policy, Least Recently Used (LRU) replacement policy, FIFO, Least Frequently Used (LFU), ….

Direct-Mapped Cache Organization

Main Memory = 16 32-byte blocks
Cache = 4 32-byte blocks

Block No	Main Memory
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

Cache Structure

Index	Valid Bit	Tag	Data
00	N		
01	N		
10	N		
11	N		

Memory Address

Tag	Index	Byte Offset

Block K of memory is mapped to the block K mod 4 of cache.

Direct Mapped Cache Organization

- What is the Hit rate for the following loop assuming the cache is initially empty? Assume the array A is allocated memory starting at address 0. Also assume the variable sum is located in a register.
- Hit Rate: Percentage of memory references served from the cache
- Miss Rate: 1 – Hit Rate

```
for( i = 0; i < 128; ++i)
    sum = sum + A[i] ;
```

Three types of Cache Misses

- Compulsory or Cold Start Misses
- Conflict Miss
- Capacity Miss

Conflict Misses and Direct Mapped Cache Organization

- What is the Hit rate for the following loop assuming the cache is initially empty? Assume the array A is allocated memory starting at address 0. Also assume the variable sum is located in a register.

```
for( i = 0; i < 128; i = i + 32)
    sum = sum + A[i] ;
for( i = 0; i < 128; i = i + 32)
    A[i] = ~A[i] ;
```

Hmm! Unnecessary Cache Misses in the second for loop. Conflict Misses? How can we solve this problem.

Conflict Misses and Direct Mapped Cache Organization

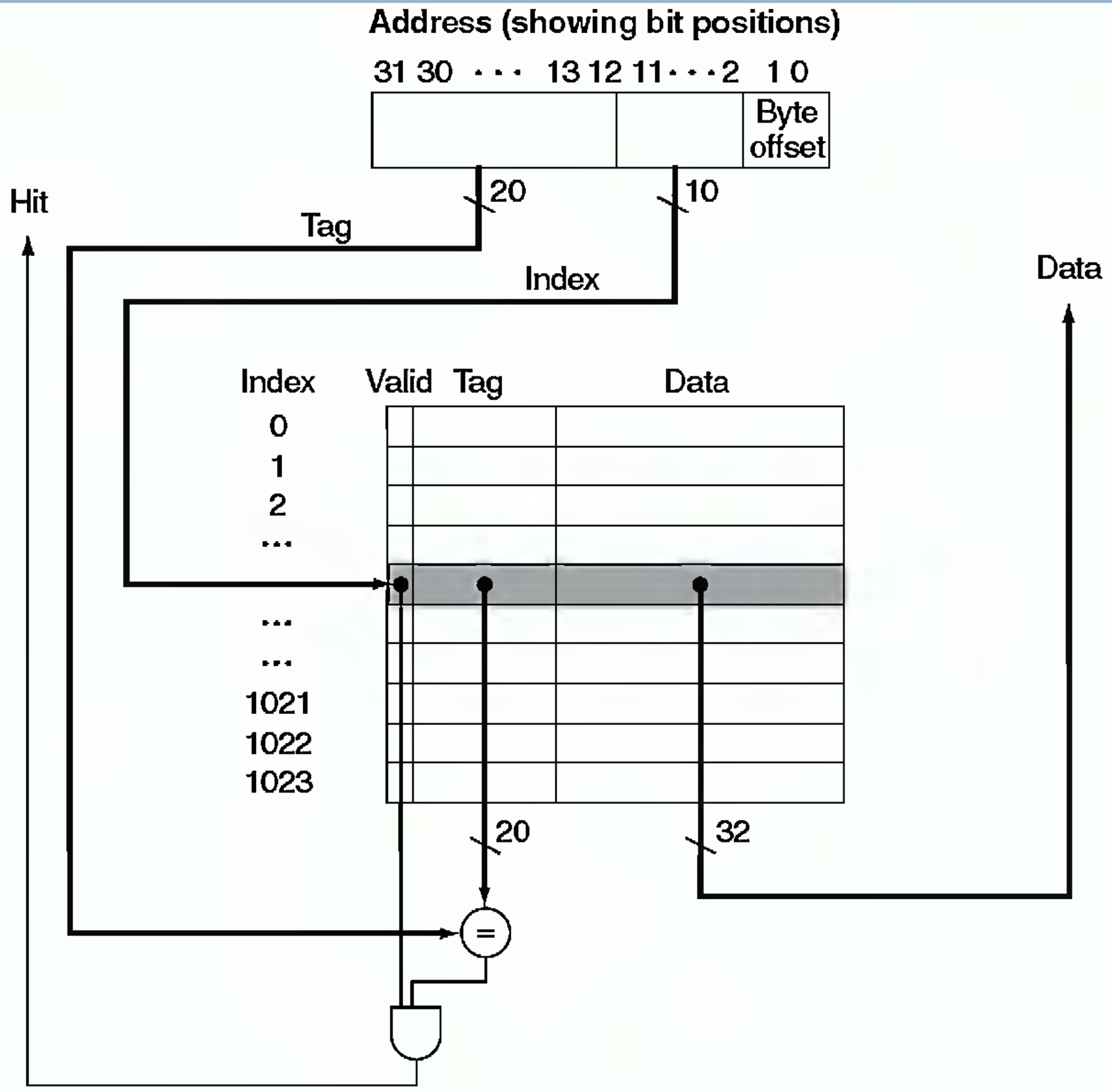
- What is the Hit rate for the following loop assuming the cache is initially empty? Assume array x is allocated memory starting at address 0x0 and array y is allocated memory starting at address 0b010000000.
- Variables sum and i are located in registers

```
float dotprod(int x[8], int y[8])
{
  int sum = 0, i ;
  for( i = 0; i < 8; ++i )
    sum = sum + x[i] * y[i] ;
  return sum;
}
```

Direct Mapped Cache Organization

By looking at this picture can you guess ….

1. Memory size
2. Block size
3. No of blocks in main memory
4. How many distinct main memory blocks can map to the same cache block?



Fully Associative Cache Organization

Main Memory = 16 32-byte blocks
Cache = 4 32-byte blocks

Block No	Main Memory
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

Cache Structure

Valid Bit	Tag	Data
N		
N		
N		
N		

Memory Address

Tag	Byte Offset

Block K of memory can be placed anywhere in Cache.

Conflict Misses and Fully Associative Cache Organization

- What is the Hit rate for the following loop assuming the cache is initially empty? Assume the array A is allocated memory starting at address 0. Also assume the variable sum is located in a register.

```
for( i = 0; i < 128; i = i + 32)
    sum = sum + A[i] ;
for( i = 0; i < 128; i = i + 32)
    A[i] = ~A[i] ;
```

No Conflict Misses at all!

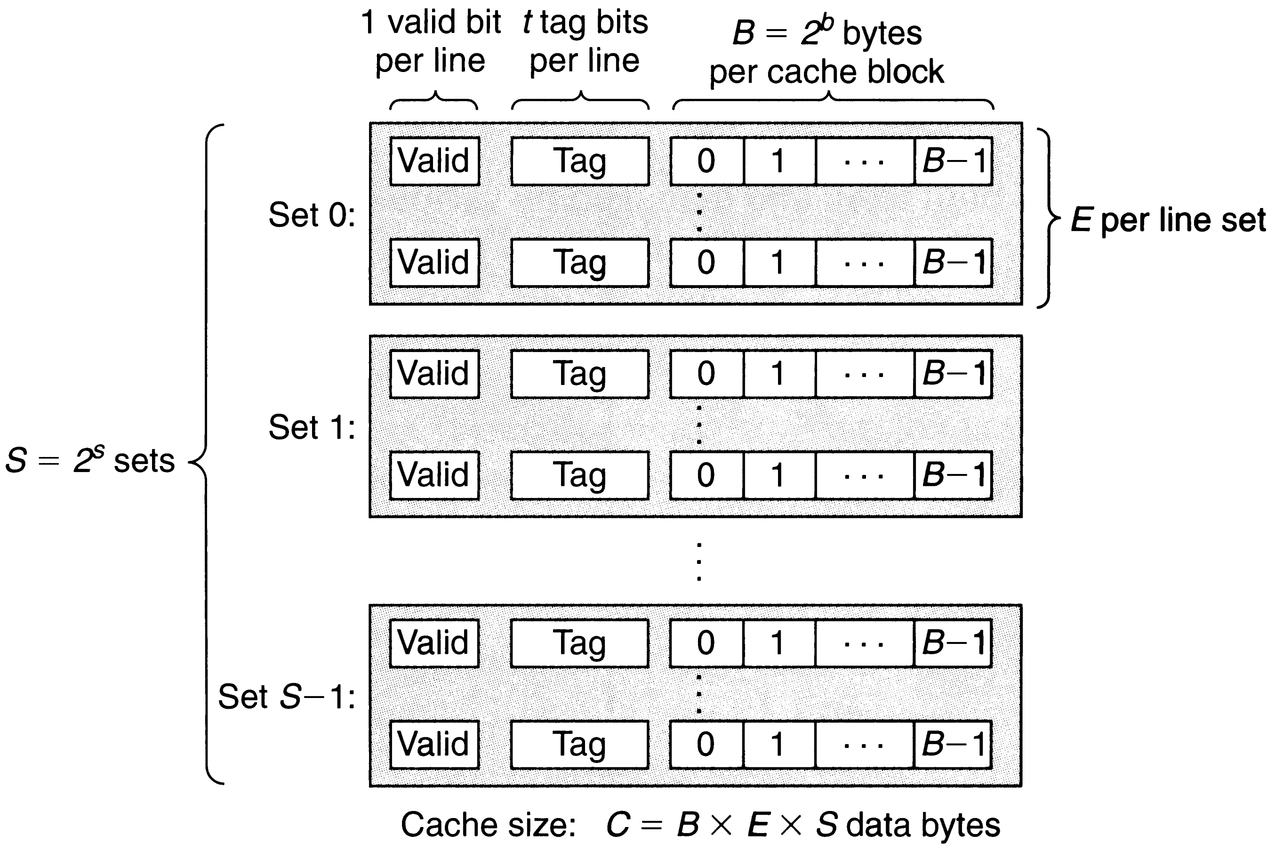
In Real Life: It is extremely expensive to build fast Tag Lookup hardware.

Fully Associative Cache Organization

- What is the Hit rate for the following loop assuming the cache is initially empty? Assume array x is allocated memory starting at address 0x0 and array y is allocated memory starting at address 0x010000000.
- Variables sum and i are located in registers

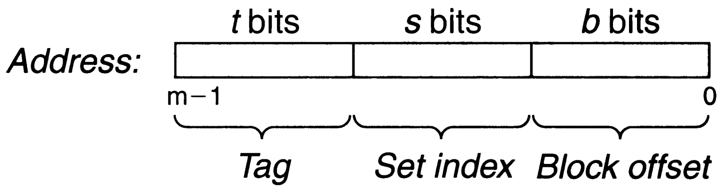
```
float dotprod(int x[8], int y[8])
{
  int sum = 0, i ;
  for( i = 0; i < 8; ++i )
    sum = sum + x[i] * y[i] ;
  return sum;
}
```

Set Associative Cache Organization



Cache size: $C = B \times E \times S$ data bytes

(a)

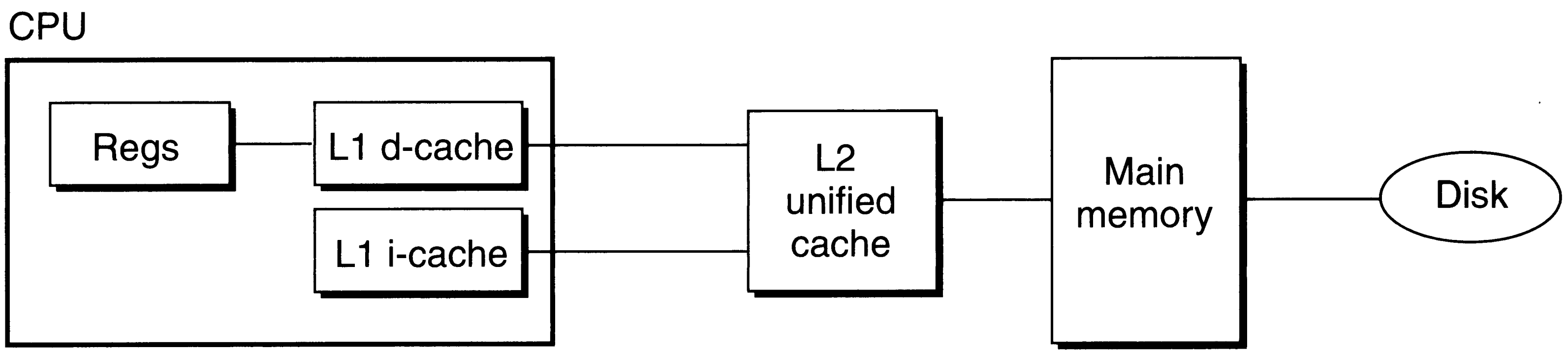


Set Associative Cache Organization

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

Typical Multi-Level Cache Organization



Cache type	Associativity (E)	Block size (B)	Sets (S)	Cache size (C)
On-chip L1 i-cache	4	32 B	128	16 KB
On-chip L1 d-cache	4	32 B	128	16 KB
Off-chip L2 unified cache	4	32 B	1024–16384	128 KB–2 MB

Intel Pentium cache organization.

Average Memory Access Time

Average Memory Access Time =

Cache Memory Access Time * Hit Ratio + (1 – Hit Ratio)* Miss Penalty

Miss Penalty: (roughly) Time taken to access the main memory block and put it in the cache.

Technology Trends

Metric	1980	1985	1990	1995	2000	2000:1980
$/MB	19,200	2,900	320	256	100	190
Access (ns)	300	150	35	15	3	100

(a) SRAM trends

Metric	1980	1985	1990	1995	2000	2000:1980
$/MB	8,000	880	100	30	1	8,000
Access (ns)	375	200	100	70	60	6
Typical size (MB)	0.064	0.256	4	16	64	1,000

(b) DRAM trends

Metric	1980	1985	1990	1995	2000	2000:1980
$/MB	500	100	8	0.30	0.01	50,000
Seek time (ms)	87	75	28	10	8	11
Typical size (MB)	1	10	160	1,000	20,000	20,000

(c) Disk trends

Metric	1980	1985	1990	1995	2000	2000:1980
Intel CPU	8080	80286	80386	Pentium	P-III	—
CPU clock rate (MHz)	1	6	20	150	600	600
CPU cycle time (ns)	1,000	166	50	6	1.6	600

(d) CPU trends

Memory Wall

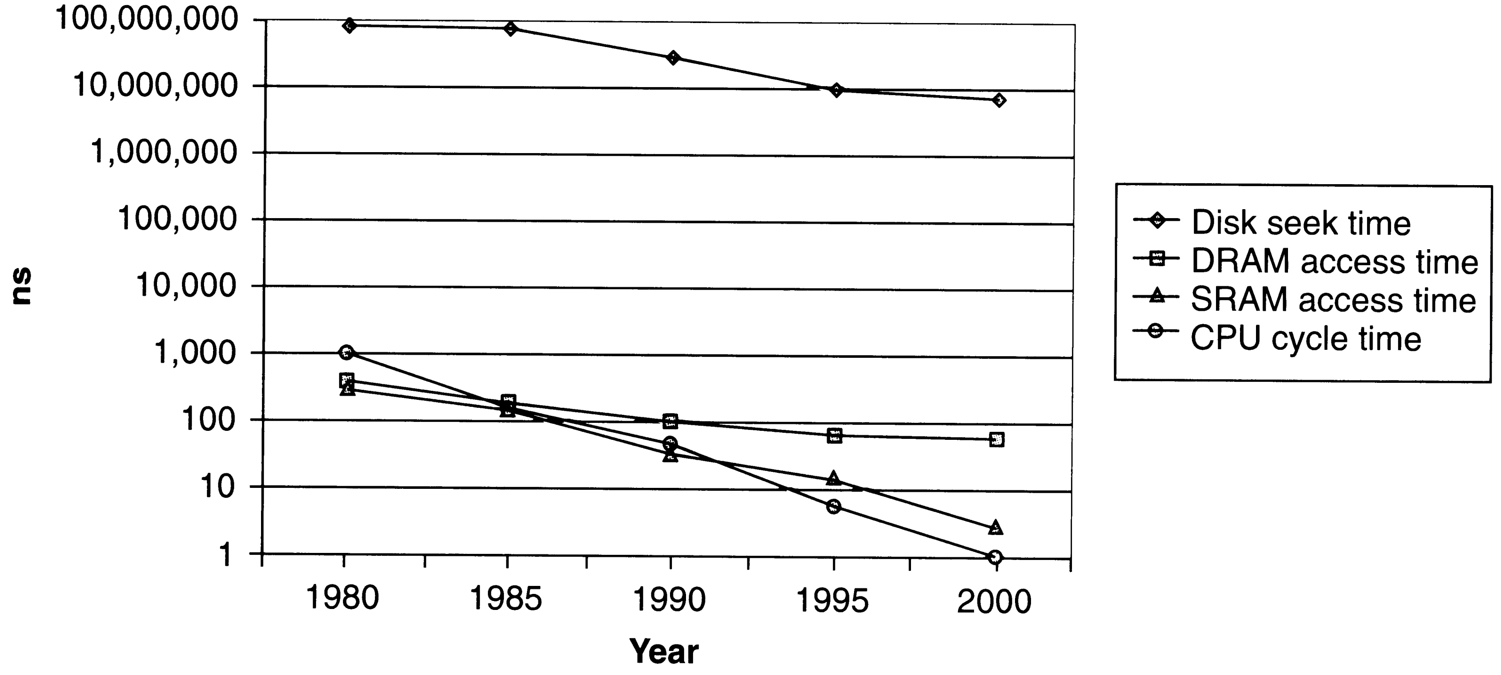


Figure 6.16 The increasing gap between DRAM, disk, and CPU speeds.