

# Memory Hierarchy

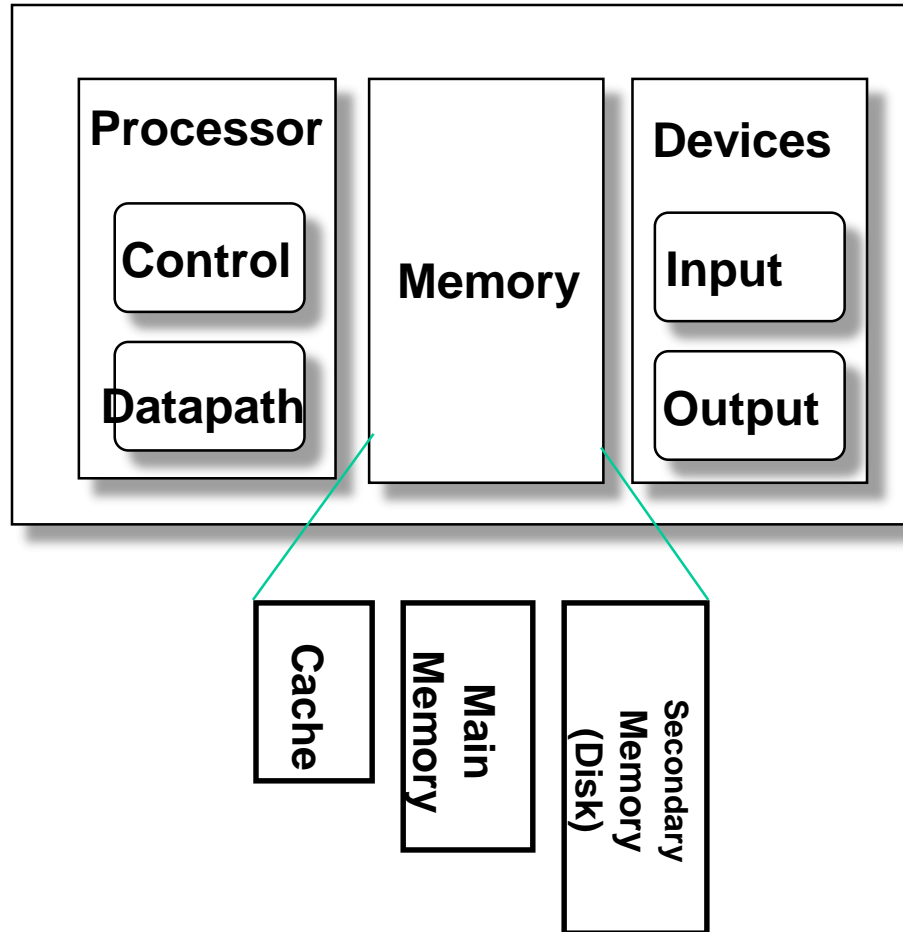
Y. K. Malaiya

Acknowledgements

Computer Architecture, Quantitative Approach - Hennessy,  
Patterson

Vishwani D. Agrawal

# Review: Major Components of a Computer



# Memory Hierarchy in LC-3

- **8 Registers (“general purpose”)**
  - Allocated by programmer/compiler
  - Fast access
- **Main memory**
  - $2^{16} \times 16$  (128K Bytes)
  - Directly address using 16 address bits.
- **LC-3 makes no mention of**
  - Secondary memory (disk)
  - Cache (between registers and Main memory)
- **Assembly language programmers know that information used most often should be kept in the registers.**

# Memory Hierarchy in Real Computers (1)

- **Registers**
  - **General purpose: 16-32**
  - **Also special registers for floating point, graphics coprocessors**
  - **Access time: within a clock cycle**
- **Small Cache memory: few MB**
  - **L1, and perhaps L2: on the same chip as CPU**
  - **Holds instructions and data likely to be used in near future.**
  - **There may be bigger on/off-chip L3 cache.**
  - **Cache access time: a few clock cycles**
  - **Managed by hardware. Serves as a “cache” for frequently used Main Memory contents.**

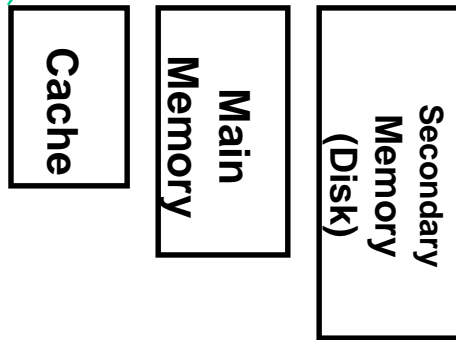
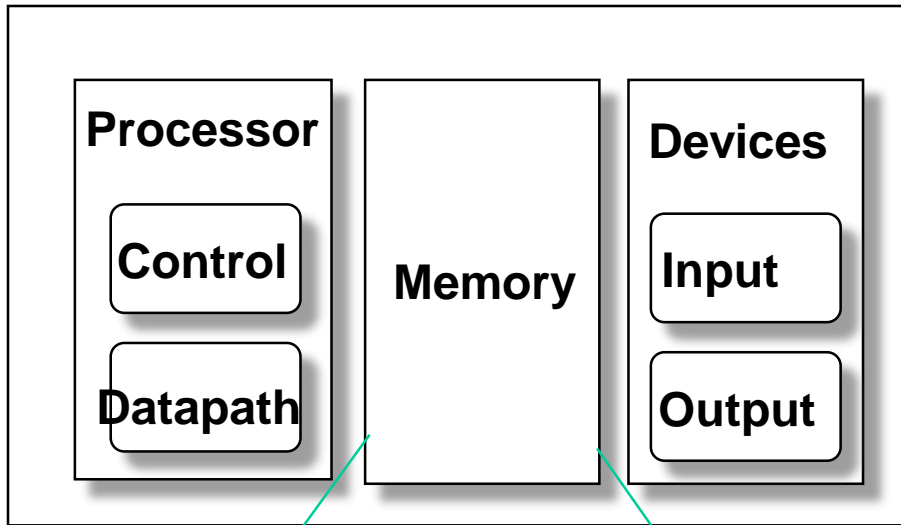
# Memory Hierarchy in Real Computers (2)

- **Main memory**
  - **Good sized:** perhaps  $2^{32} \times 8 = 4 \text{ GB}$ , perhaps more
  - **Access time:** 100-1000 clock cycles
  - **However a large fraction of all memory accesses, perhaps 95%, result in a **cache hit**, making effective memory access *appear* much faster.**
  - **Addresses often translated using Frame Tables, which allows each process to have a dedicated address space.**
  - **Virtual Memory:** Some of the information supposedly in the Main Memory may actually be on the disk, which is brought in when needed. That make Virtual Memory bigger than actual physical main Memory. Handled by the **Operating System**.

# Memory Hierarchy in Real Computers (3)

- **Secondary Memory (Disk: real or solid-state)**
  - **HD**: magnetic hard disk, **SSD**: solid-state device that works like a disk
  - **Access time**: Each memory access can take as long as perhaps a million CPU clock cycles! Delays involve seek time by read/write head and rotational delay. SSDs are somewhat faster.
  - **System design attempts to minimize the number of disk accesses.**
  - **Capacity**: 1 or more TB (1 TB = 1,000 GB)
  - **A disk can fail. Backup is often used. Cloud or external drive may be used for backup.**

# Review: Major Components of a Computer



| Level             | Access time | Size                  | Cost/GB |
|-------------------|-------------|-----------------------|---------|
| Registers         | 0.25 ns     | 48+<br>64,128,<br>32b | -       |
| Cache<br>L1,L2,L3 | 0.5 ns      | 5MB                   | 125     |
| Memory            | 1000 ns     | 8GB                   | 4.0     |
| SSD               | 100K ns     |                       | 0.25    |
| Disk              | 1000K ns    | 1 TB                  | 0.02    |

# **The Memory Hierarchy: Key facts and ideas (1)**

- **Programs keep getting bigger exponentially.**
- **Memory cost /bit**
  - **Faster technologies are expensive, slower are cheaper. Different by orders of magnitude**
  - **With time storage density goes up driving per bit cost down.**
- **Locality in program execution**
  - **Code/data used recently will likely be needed soon.**
  - **Code/data that is near the one recently used, will likely be needed soon.**



## **The principle of caching: Key facts and ideas (2)**

- **Information that is likely to be needed soon, keep it where is fast to access it.**
  - Recently used information
  - Information in the vicinity of what was recently used
- **Caching**
  - A very general concept
  - Keep info likely to be needed soon close.
- **Examples of caches:**
  - Browser cache
  - Information cached in disk drives
  - Info in main memory brought in from disk (“Virtual Memory”)
  - Info in cache memory brought in from main memory

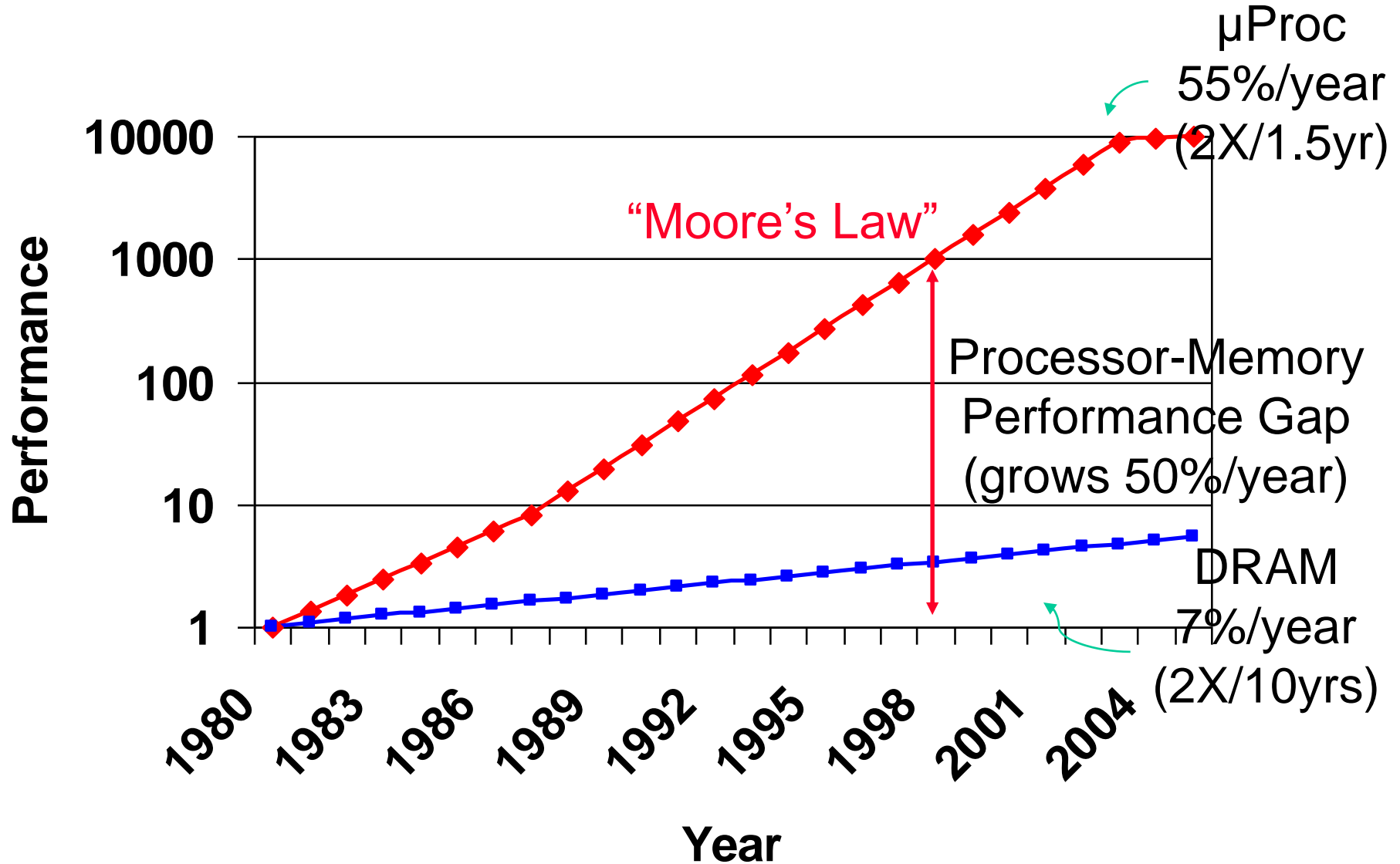
## **Block management: Key facts and ideas (3)**

- **Level x storage is divided into blocks**
  - **Cache: a few words**
  - **Memory: a few KB or MB**
- **Level x can only hold a limited number of blocks**
  - **Hit: when information sought is there**
  - **Miss: block needs to be fetched from the higher level  $x+1$**
- **Block replacement strategy**
  - **A block must be freed to the incoming block**
  - **Which block to be replaced**
  - **Perhaps one that has not been used for a file, and thus not likely to be needed soon**

# Trends: Moore's law and DRAM speed

- **Moore's law**
  - Number of transistors on a CPU chip **double every 1.5 years**, i.e. increase each year by a factor of 55%.
  - CPU performance also rises proportionately.
- **DRAM chips are also getting faster**
  - But at 7% a year , or doubling **every 10 years**.
- **Processor-Memory performance gap:**
  - Relative to the CPU, memory gets slower and slower.
  - The gap is filled by one or more levels of cache, which are faster than Main Memory (often DRAM chips), although slower than CPU.

# Processor-Memory Performance Gap

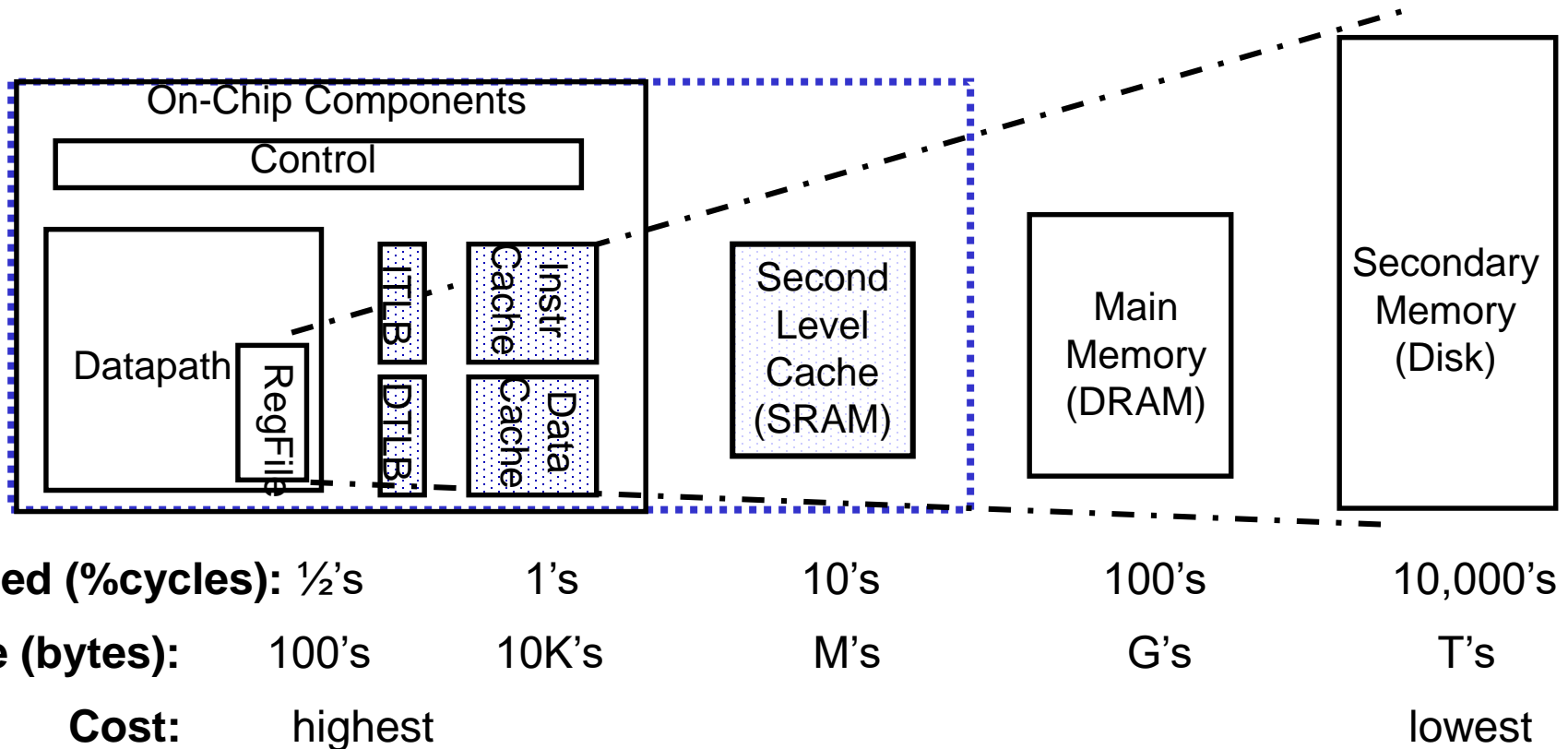


# The Memory Hierarchy Goal

- **Fact: Large memories are slow and fast memories are small**
- **How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?**
  - **With hierarchy**
  - **With parallelism**

# A Typical Memory Hierarchy

- Take advantage of the **principle of locality** to present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology



# Principle of Locality

- **Programs access a small proportion of their address space at any time**
- **Temporal locality**
  - **Items accessed recently are likely to be accessed again soon**
  - **e.g., instructions in a loop, induction variables**
- **Spatial locality**
  - **Items near those accessed recently are likely to be accessed soon**
  - **E.g., sequential instruction access, array data**

# Taking Advantage of Locality

- Memory hierarchy
- Store everything on **disk**
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - **Main memory**
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - **Cache memory attached to CPU**



# Memory Hierarchy Levels

- **Block: unit of copying**

  - May be multiple words

- **If accessed data is present in upper level**

  - **Hit: access satisfied by upper level**

    - **Hit ratio: hits/accesses**

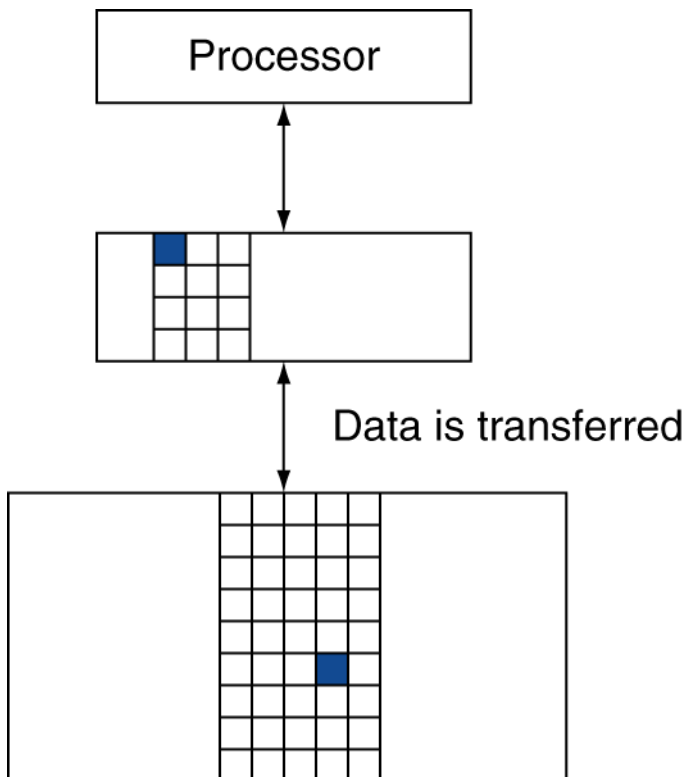
- **If accessed data is absent**

  - **Miss: block copied from lower level**

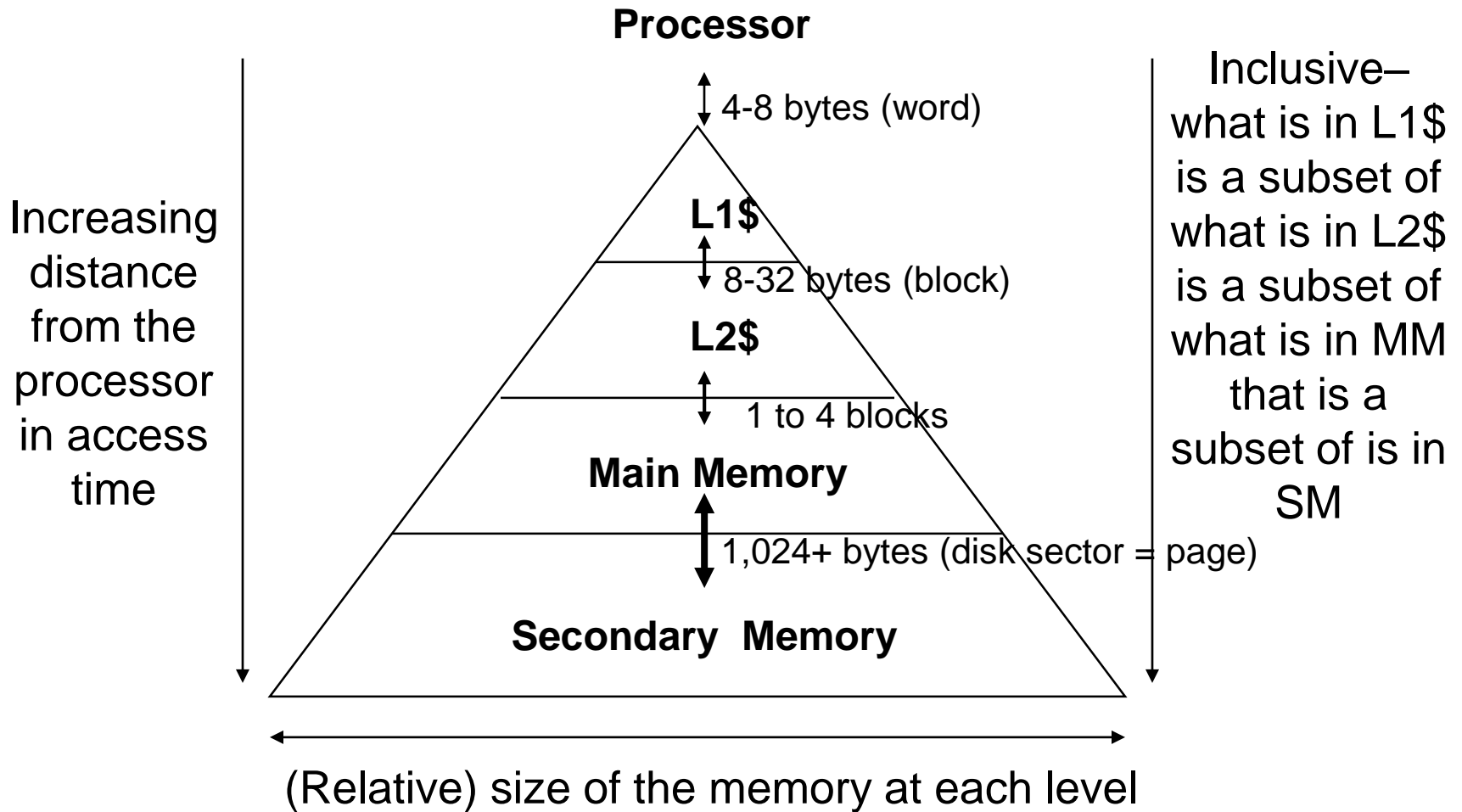
    - **Time taken: miss penalty**

    - **Miss ratio: misses/accesses**  
**= 1 – hit ratio**

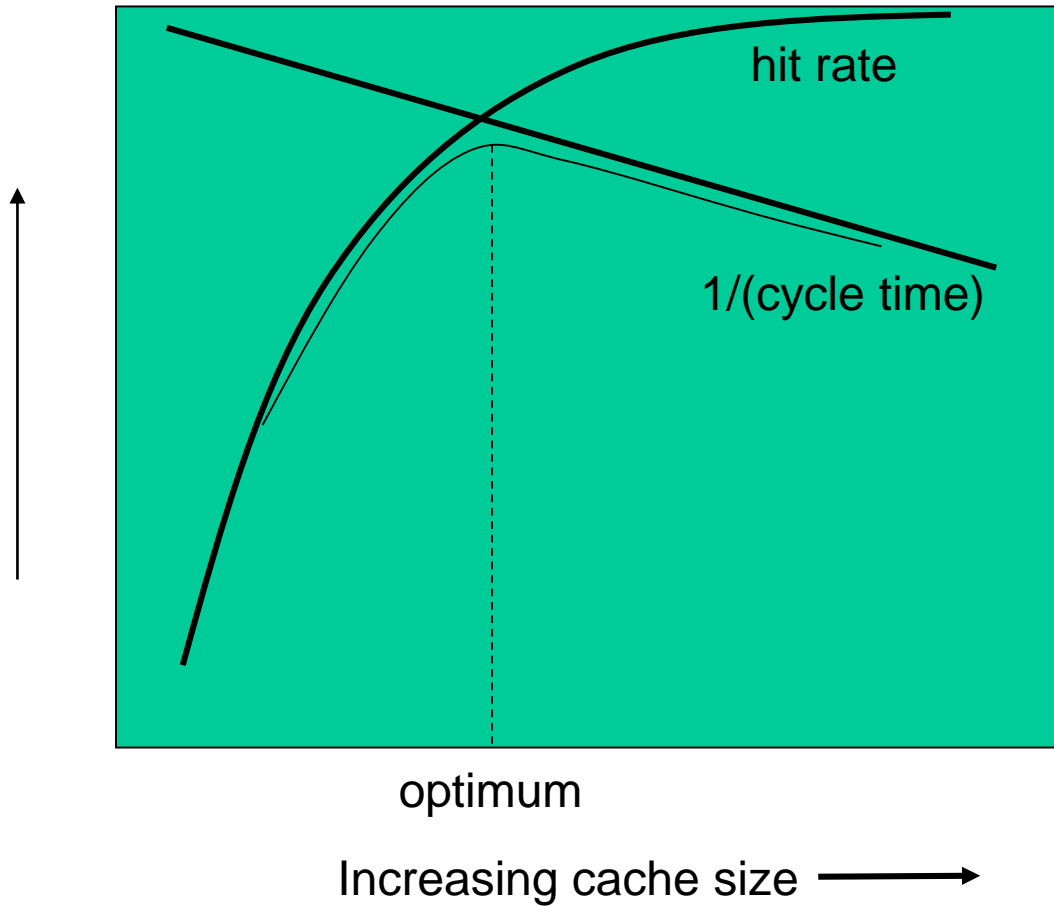
  - **Then accessed data supplied from upper level**



# Characteristics of the Memory Hierarchy



# Cache Size vs average access time



A larger cache has a better hit rate, but, higher access time

# Cache Memory

## ● Cache memory

- The level of the memory hierarchy closest to the CPU

## ● Given accesses $X_1, \dots, X_{n-1}, X_n$

|           |
|-----------|
| $X_4$     |
| $X_1$     |
| $X_{n-2}$ |
|           |
| $X_{n-1}$ |
| $X_2$     |
|           |
| $X_3$     |

a. Before the reference to  $X_n$

|           |
|-----------|
| $X_4$     |
| $X_1$     |
| $X_{n-2}$ |
|           |
| $X_{n-1}$ |
| $X_2$     |
| $X_n$     |
| $X_3$     |

b. After the reference to  $X_n$

- How do we know if the data is present?
  - include address as tag.
- Where is it?
  - Associative search for the tag: fast since it is implemented in hardware.

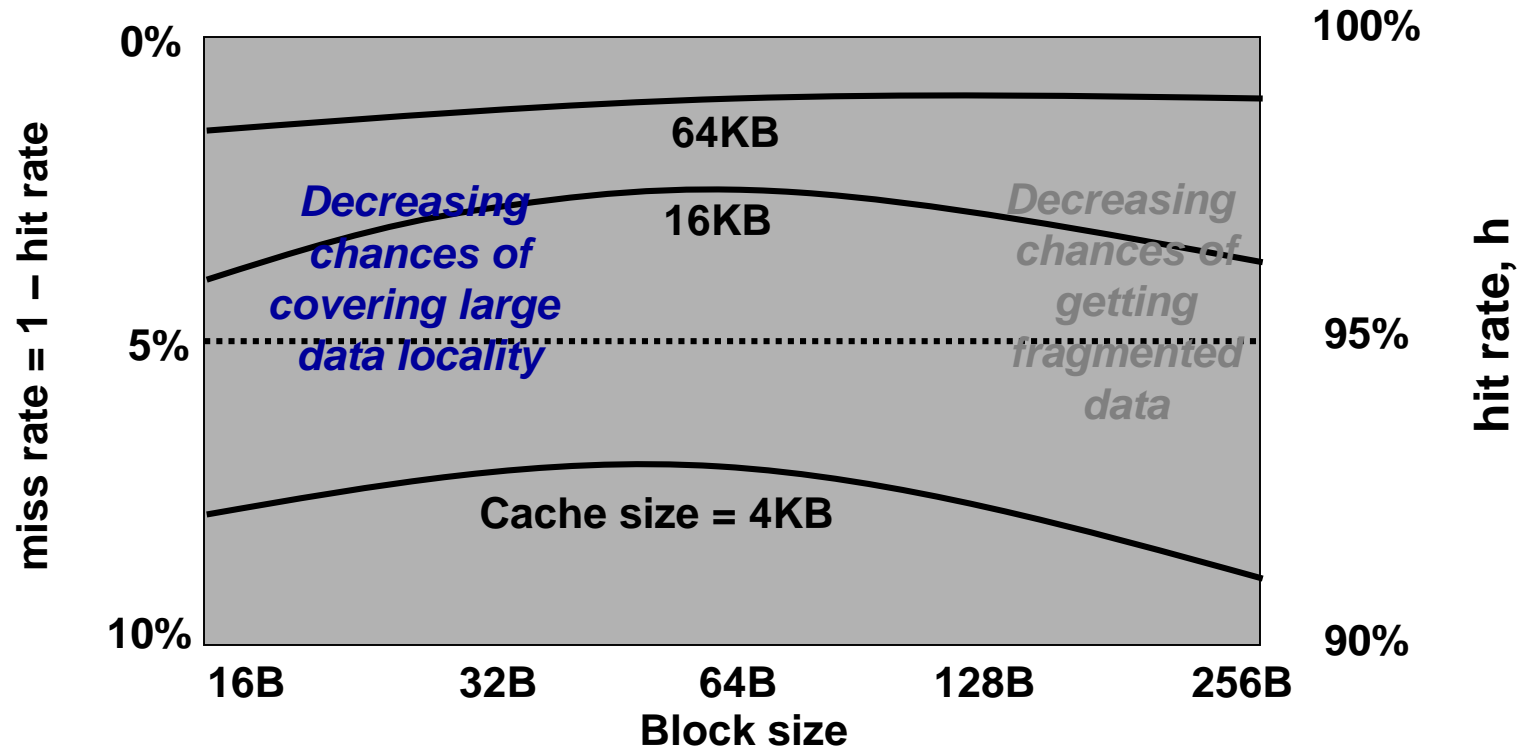
# Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks  $\Rightarrow$  fewer of them
    - More competition  $\Rightarrow$  increased miss rate
  - Larger blocks  $\Rightarrow$  *pollution*
- Minimizing miss penalty
  - Loading *critical-word-first* in a block can help

Dirty block: block that has been written into. Must be saved before replacement.

# Increasing Hit Rate

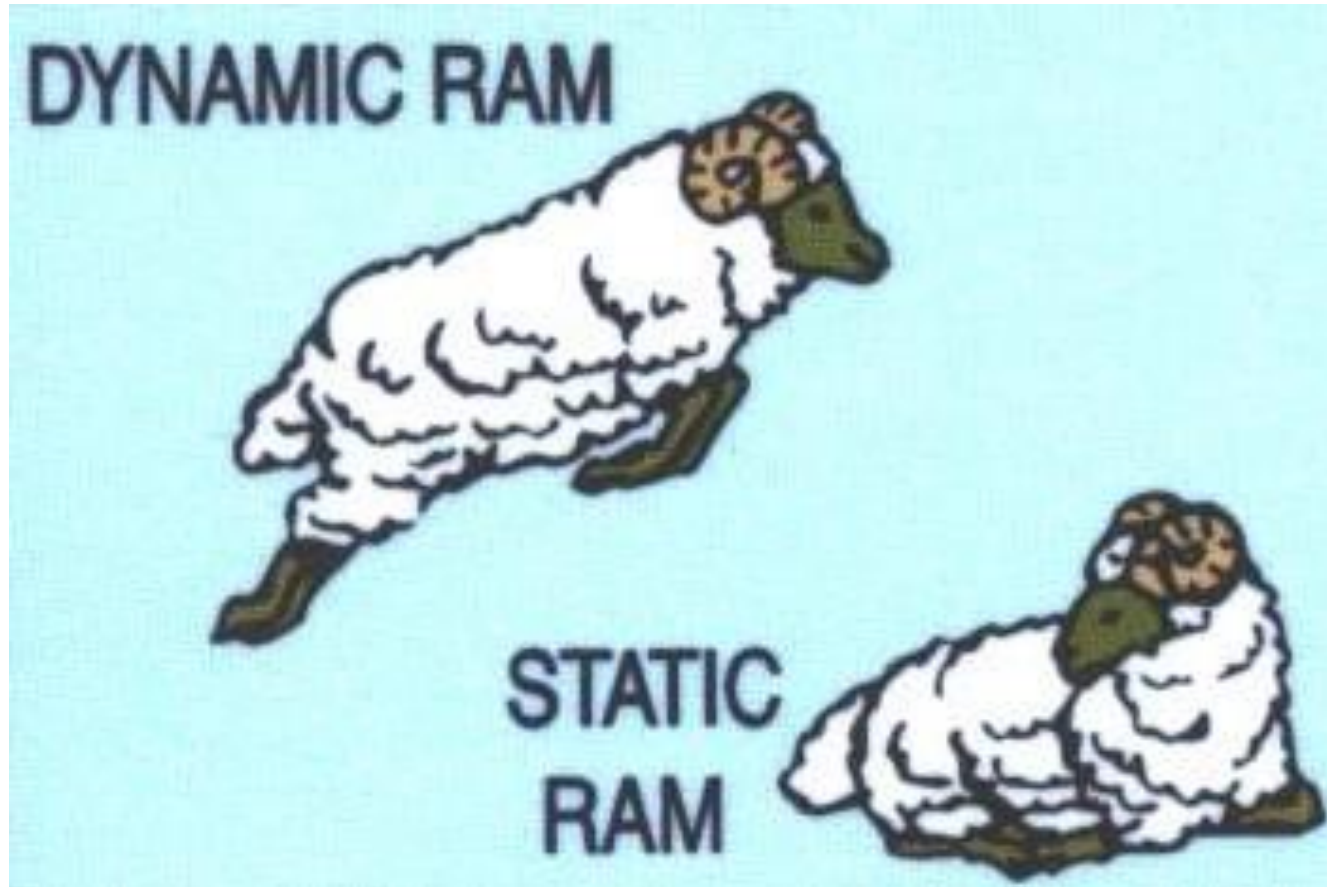
- Hit rate increases with cache size.
- Hit rate mildly depends on block size.



# Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

# Static vs Dynamic RAMs





# Random Access Memory (RAM)

Address bits



Address  
decoder



Memory  
cell  
array

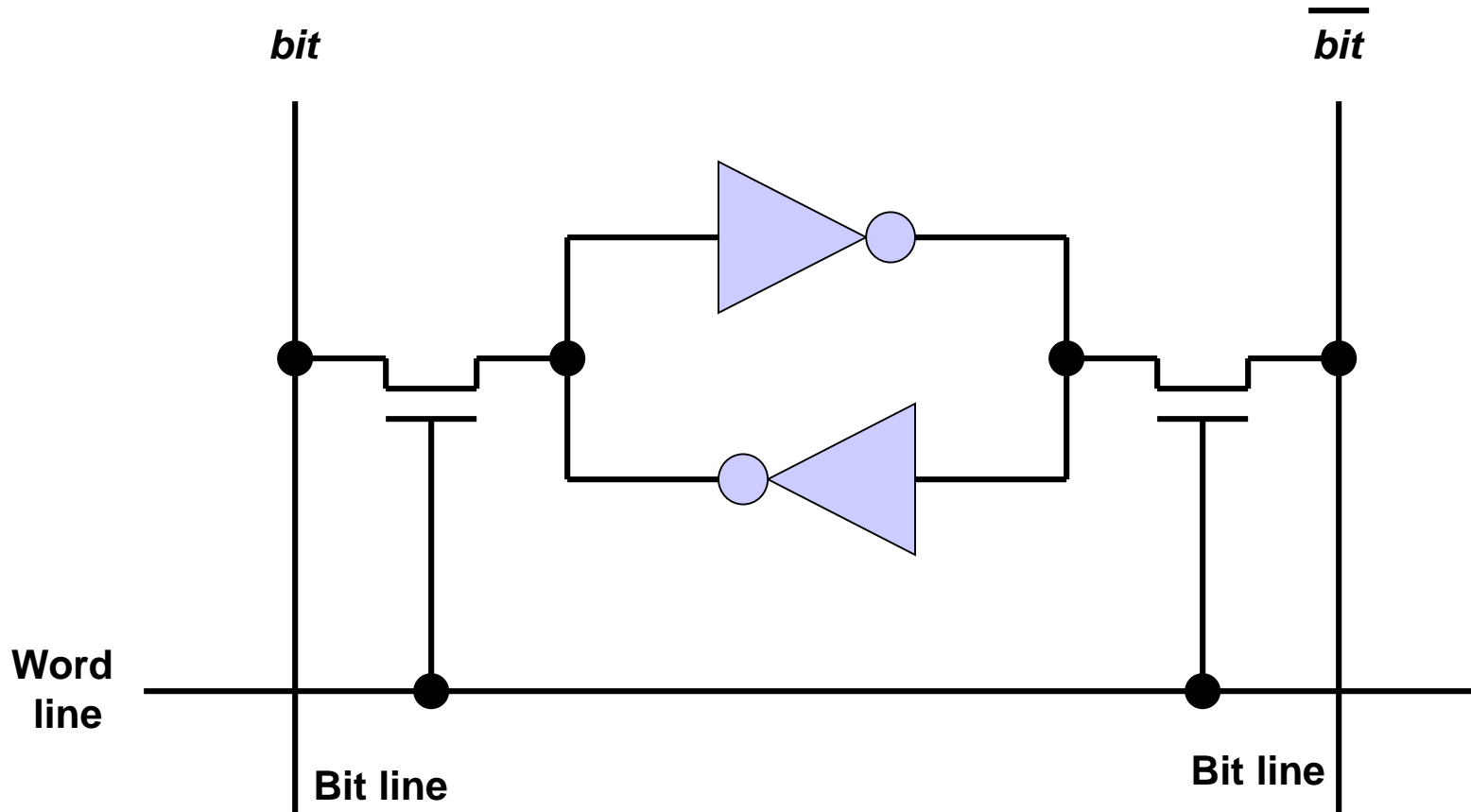


Read/write  
circuits



Data bits

# Six-Transistor SRAM Cell

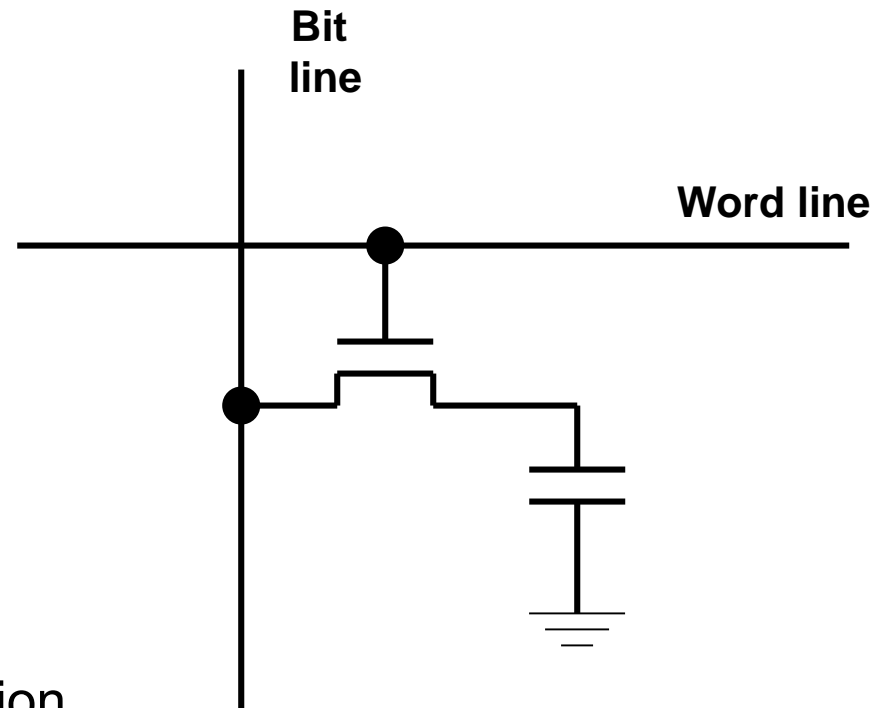


Note the feedback used to store a bit of information

# Dynamic RAM (DRAM) Cell



“Single-transistor DRAM cell”  
Robert Dennard’s 1967 invention



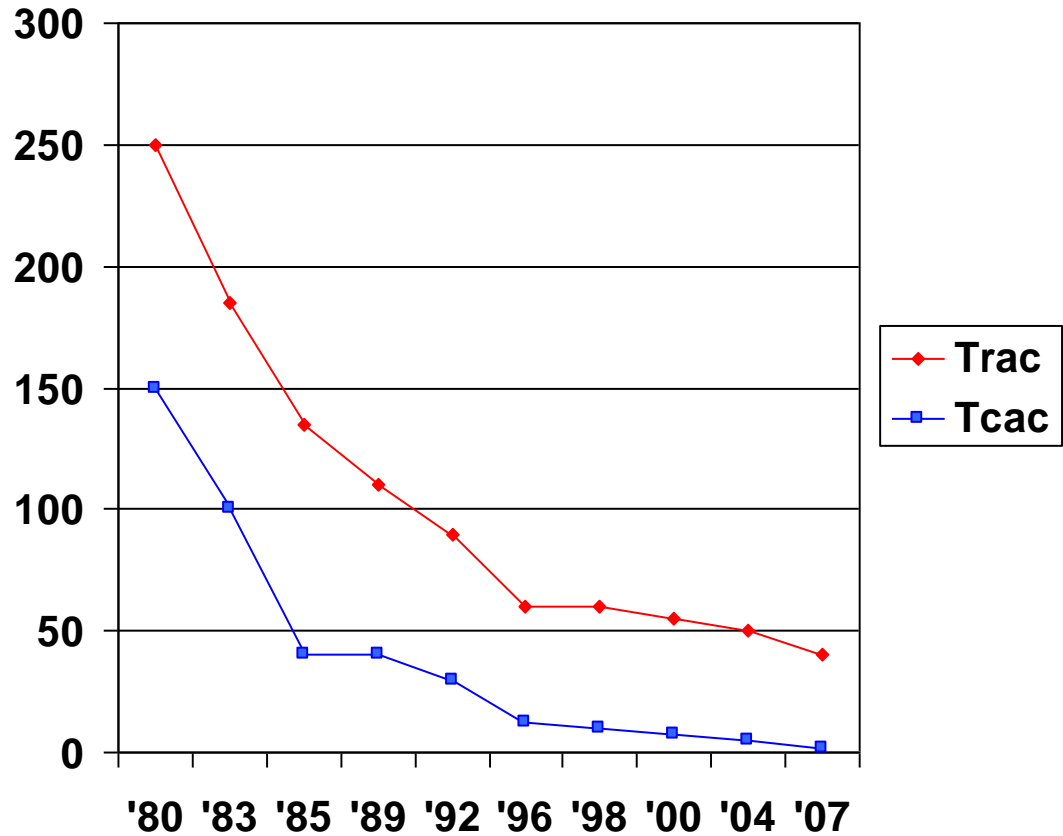
Charge across the capacitor store a bit. Charge keep leaking slowly, hence need to refresh charge Time to time.

# Advanced DRAM Organization

- **Bits in a DRAM are organized as a rectangular array**
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- **Double data rate (DDR) DRAM**
  - Transfer on rising and falling clock edges
- **Quad data rate (QDR) DRAM**
  - Separate DDR inputs and outputs

# DRAM Generations

| Year | Capacity | \$/GB     |
|------|----------|-----------|
| 1980 | 64Kbit   | \$1500000 |
| 1983 | 256Kbit  | \$500000  |
| 1985 | 1Mbit    | \$200000  |
| 1989 | 4Mbit    | \$50000   |
| 1992 | 16Mbit   | \$15000   |
| 1996 | 64Mbit   | \$10000   |
| 1998 | 128Mbit  | \$4000    |
| 2000 | 256Mbit  | \$1000    |
| 2004 | 512Mbit  | \$250     |
| 2007 | 1Gbit    | \$50      |



Row and column access times

## Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example: CPU with cache and main memory
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - $AMAT = 1 + 0.05 \times 20 = 2ns$ 
    - 2 cycles per instruction

## Performance Summary

- **When CPU performance increased**
  - Miss penalty becomes more significant
- **Can't neglect cache behavior when evaluating system performance**

# Multilevel Caches

- **Primary cache attached to CPU**
  - **Small, but fast**
- **Level-2 cache services misses from primary cache**
  - **Larger, slower, but still faster than main memory**
- **Main memory services L-2 cache misses**
- **Some systems now include L-3 cache**



# Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a **private virtual address** space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

# Virtual vs. Physical Address

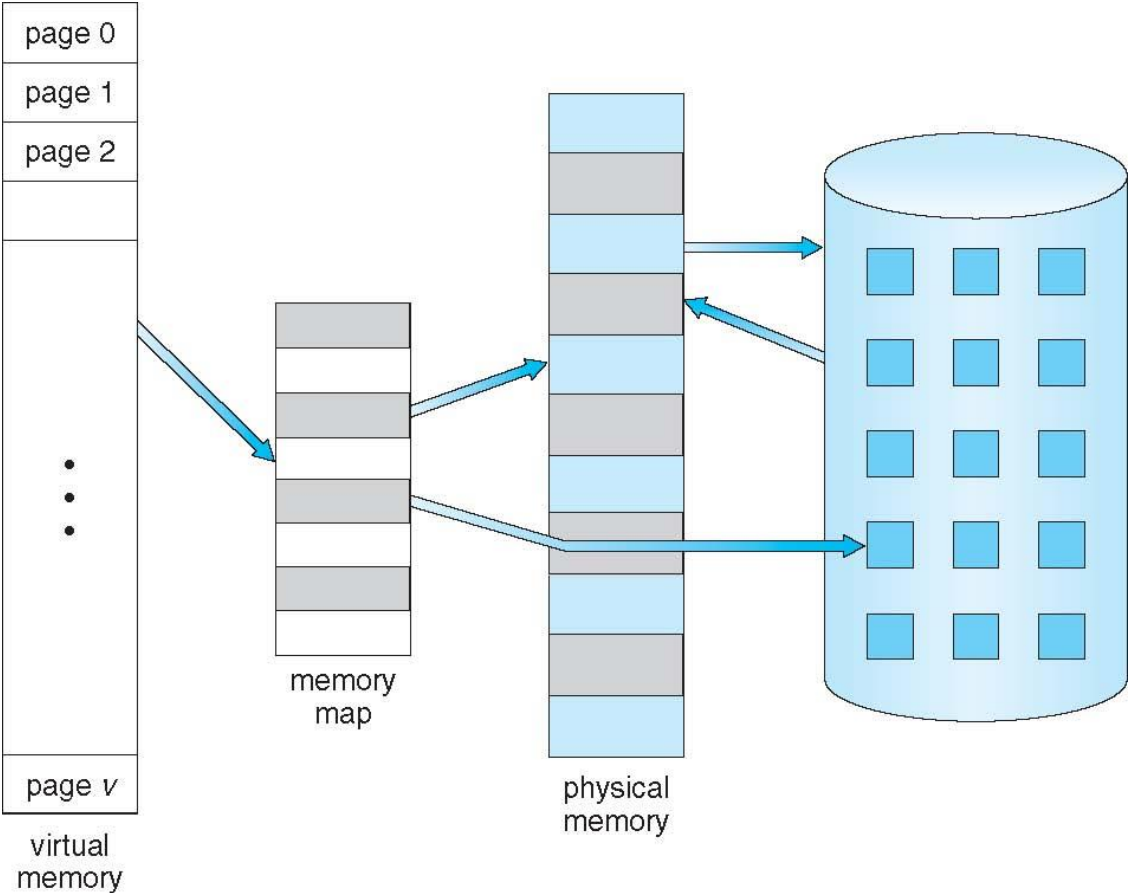
● **Processor assumes a certain memory addressing scheme:**

- A block of data is called a virtual page
- An address is called virtual (or logical) address

● **Main memory may have a different addressing scheme:**

- Real memory address is called a physical address, MMU translates virtual address to physical address
- Complete address translation table is large and must therefore reside in main memory
- MMU contains TLB (translation lookaside buffer), which is a small cache of the address translation table

# Virtual vs. Physical Address

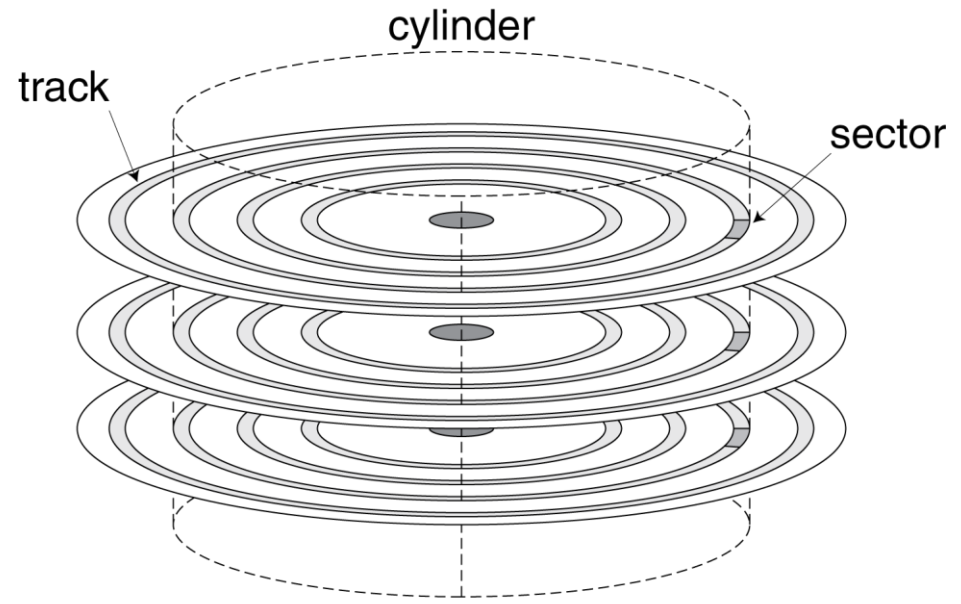


## Page Fault Penalty

- **On page fault, the page must be fetched from disk**
  - Takes millions of clock cycles
  - Handled by OS code
- **Try to minimize page fault rate**
  - Smart replacement algorithms

# Disk Storage

- Nonvolatile, rotating magnetic storage



# Disk Sectors and Access

## ● Each sector records

- Sector ID
- Data (512 bytes, 4096 bytes proposed)
- Error correcting code (ECC)
  - Used to hide defects and recording errors
- Synchronization fields and gaps

## ● Access to a sector involves

- Queuing delay if other accesses are pending
- Seek: move the heads
- Rotational latency
- Data transfer
- Controller overhead

# Disk Access: Example

## ● Given

- 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Find average read time

## ● Average read time

- 4ms seek time
  - +  $\frac{1}{2} / (15,000/60) = 2\text{ms}$  rotational latency
  - +  $512 / 100\text{MB/s} = 0.005\text{ms}$  transfer time
  - + 0.2ms controller delay
  - = 6.2ms

## ● If actual average seek time is 1ms

- Average read time = 3.2ms

# Disk Performance Issues

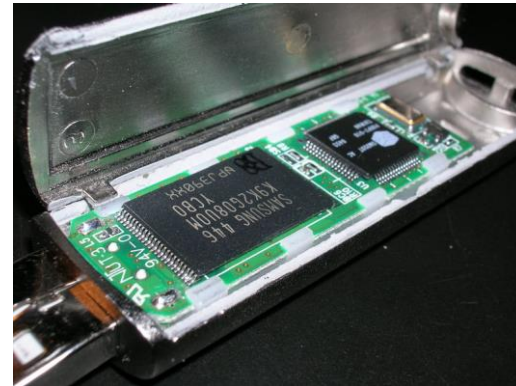
- **Manufacturers quote average seek time**
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- **Smart disk controller allocate physical sectors on disk**
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- **Disk drives include caches**
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay



# Flash Storage

## ● Nonvolatile semiconductor storage

- $100\times$  –  $1000\times$  faster than disk
- Smaller, lower power, more robust
- But more \$/GB (between disk and DRAM)



# Flash Types

- **NOR flash: bit cell like a NOR gate**
  - Random read/write access
  - Used for instruction memory in embedded systems
- **NAND flash: bit cell like a NAND gate**
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, ...
- **Flash bits wears out after 1000's of accesses**
  - ~~Not suitable for direct RAM or disk replacement~~
  - Wear leveling: remap data to less used blocks

# Memory Protection

● Different tasks can share parts of their virtual address spaces

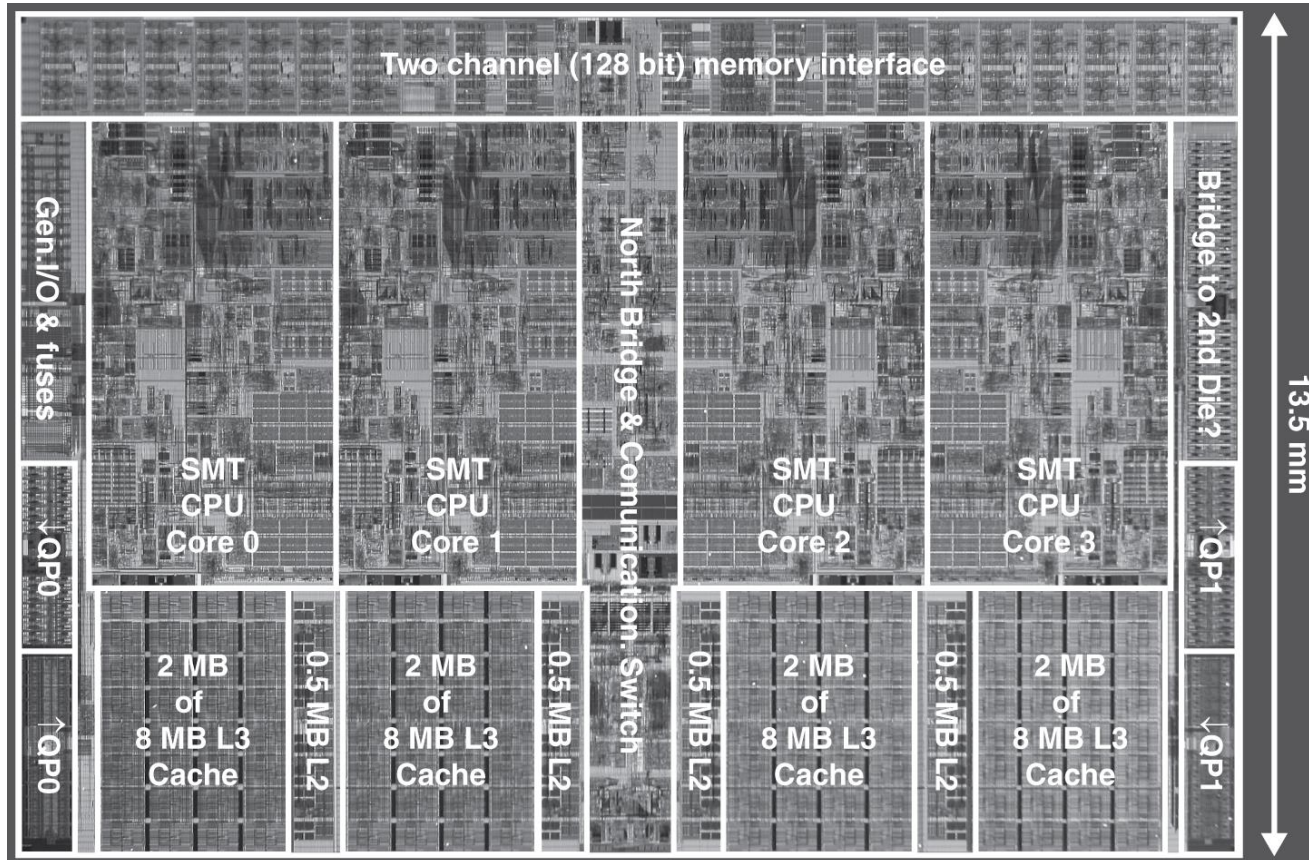
- But need to protect against errant access
- Requires OS assistance

● Hardware support for OS protection

- Privileged supervisor mode (aka kernel mode)
- Privileged instructions
- Page tables and other state information only accessible in supervisor mode
- System call exception (e.g., syscall in MIPS)

# Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

## 3-Level Cache Organization

|                             | Intel Nehalem   | AMD Opteron X4  |
|-----------------------------|---|---|
| L1 caches (per core)        | L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a<br>L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a | L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles<br>L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles |
| L2 unified cache (per core) | 256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a   | 512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a  |
| L3 unified cache (shared)   | 8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a   | 2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles  |

n/a: data not available

## Concluding Remarks

- **Fast memories are small, large memories are slow**
  - We really want fast, large memories ☹️
  - Caching gives this illusion 😊
- **Principle of locality**
  - Programs use a small part of their memory space frequently
- **Memory hierarchy**
  - L1 cache ↔ L2 cache ↔ ... ↔ DRAM memory  
↔ disk
- **Memory system design is critical for multiprocessors**