Revised: 3/19/2021

Memory Architecture II

Michael C. Hackett
Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- Dependability
  - Parity Bits
  - Error Correction Code
- Virtual Machines
- Virtual Memory
  - Page Faults
  - Exceptions
- Rotational Hard Disks

Dependability

- A device (such as a hard drive) alternates between two states of service
- **Service Accomplishment**
  - The device is working (delivering service) as designed
- **Service Interruption**
  - The device is not working (not delivering service) as designed

- A **failure** causes the device to go from Service Accomplishment to Service Interruption
- A **restoration** causes the device to go from Service Interruption to Service Accomplishment

Dependability

- The measure of continuous, uninterrupted Service Accomplishment until the device experiences a failure its **reliability**
  - Two such measures of reliability are:
  - **Mean Time to Failure (MTTF)**
    - The average time it takes until a device fails
  - **Annual Failure Rate (AFR)**
    - The percentage of devices expected to fail in a year within a given MTTF

Dependability

- As an example, we'll say a server farm has 25,000 servers.
  - Each server has 3 hard disks (all the same model)
    - $3 \times 25000 = 75000\ disks$
  - The MTTF of these disks are said to be 1,000,000 hours

$$One\ year = 365\ days\ \times 24 \frac{hours}{day} = 8760\ hours$$

$$AFR = \frac{8760\ hours}{1000000\ hours} = .00876 = .876\%$$

- The AFR of each disk is 0.876%

$$75000\ disks\ \times .00876 = 657$$

- 657 disks are expected to fail annually
  - $\frac{657\ disks}{365\ days} = 1.8 \sim 2\ disks\ per\ day$

Dependability

- Service Interruption is measured by **Mean Time to Repair (MTTR)**
- The **Mean Time Between Failures (MTBF)** is the sum of the MTTF and MTTR
- Availability is the measure of Service Accomplishment (MTTF) with respect to the alternation between Service Accomplishment and Service Interruption (MTBF)

$$Availability = \frac{MTTF}{(MTTF + MTTR)}$$

Dependability

- Using the previous example, we'll say MTTR for each disk in the server farm is 2 hours.
  - Each server has 3 hard disks (all the same model)
    - $3 \times 25000 = 75000\ disks$
  - The MTTF of these disks are said to be 1,000,000 hours
  - 657 disks are expected to fail annually (calculated on a previous slide)

$$Availability = \frac{1000000\ hours}{(1000000\ hours + (2\ hours\ \times 657\ disks))} \sim .9986 \sim 99.86\%$$

Dependability

- MTTF can be increased through improving the quality of components or designing them to continue operating in the event of failures
  - **Fault avoidance**
    - Preventing faults in the design of the device/components
  - **Fault tolerance**
    - Using redundancy to continue functioning in the event of faults
  - **Fault forecasting**
    - Predicting faults so that devices/components may be replaced before a fault occurs

Parity Bits

- **Parity bits** are used as a way of validating that binary data makes it from source to destination intact.
  - A type of *error detection code*

- Before a binary word is sent through the system's data path (we'll use 8-bit words for this example), the total number of 1's in the word are counted.
  - If there are an even number of 1's, a 0 is added to the end of the number
  - If there are an odd number of 1's, a 1 is added to the end of the number

  - Either way, parity bits ensure an even number of 1's in the n+1 bit word.

  - If the received binary word (plus the parity bit) contain an odd number of 1's, it indicates there was a problem with the integrity of the data, and the word should be re-sent.

Parity Bits

- 01101100
  - Even number of 1's
  - 01101100**0**

Parity bit

Bit gets corrupted in transit

011**<u>1</u>**11000

Destination

011111000

- Not an even number of 1's
- Data must have been corrupted.

- 01001100
  - Odd number of 1's
  - 01001100**1**

Parity bit

Parity Bits

- Parity bits alone are only able to *detect* errors.

- It is possible that an error will not be detected:

- 01101100
  - Even number of 1's
  - 01101100**0**

Parity bit

Bits get corrupted in transit

011**1**110**1**0

Destination

011111010

- Even number of 1's
- No error detected

Error Correction Code

- **Error Correction Code** (ECC or Hamming Code, named after its creator Richard Hamming) uses multiple parity bits that can detect and *correct* errors in a binary word.

- A simple algorithm is followed to insert parity bits into the binary data.

Error Correction Code

1. Number the bits starting at 1 from the left side:

1 2 3 4 5 6 7 8
0 1 1 1 1 1 0 1

2. Mark the positions that are powers of 2 (1, 2, 4, 8, 16, and so on) as the parity bits

1 **2** 3 **4** 5 6 7 **8**
0 1 1 1 1 1 0 1

Error Correction Code

3. All bit positions that are not powers of two will be used for the actual data:

1	**2**	3	**4**	5	6	7	**8**
0	1	1	1	1	1	0	1

→

1	2	3	4	5	6	7	8	9	10	11	12
_	_	0	_	1	1	1	_	1	1	0	1

Error Correction Code

4. The position of a parity bit determines which bits it checks:
   - Bit 1 checks positions 1, 3, 5, 7, 9, and so on
   - Bit 2 checks positions 2, 3, 6, 7, 10, 11, 14, 15, and so on
   - Bit 4 checks positions 4-7, 12-15, 20-23, and so on
   - Bit 8 checks positions 8-15, 24-31, 40-47, and so on

   - Each bit of the actual data is checked by two or more parity bits

Error Correction Code

- Bit 1 checks positions 1, 3, 5, 7, 9, and so on:

1	2	3	4	5	6	7	8	9	10	11	12
_	_	**0**	_	**1**	1	**1**	_	**1**	1	**0**	1

- Odd number of 1’s

1	2	3	4	5	6	7	8	9	10	11	12
1	_	0	_	1	1	1	_	1	1	0	1

Error Correction Code

- Bit 2 checks positions 2, 3, 6, 7, 10, 11, 14, 15, and so on :

1 2 3 4 5 6 7 8 9 10 11 12

1 _ **0** _ 1 **1 1** _ 1 **1 0** 1

- Odd number of 1’s

1 2 3 4 5 6 7 8 9 10 11 12

1 **1** 0 _ 1 1 1 _ 1 1 0 1

Error Correction Code

- Bit 4 checks positions 4-7, 12-15, 20-23, and so on :

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	_	**1**	**1**	**1**	_	1	1	0	**1**

- Even number of 1's

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	**0**	1	1	1	_	1	1	0	1

Error Correction Code

- Bit 8 checks positions 8-15, 24-31, 40-47, and so on :

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	0	1	1	1	_	**1**	**1**	**0**	**1**

- Odd number of 1’s

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	0	1	1	1	**1**	1	1	0	1

Error Correction Code

- We'll say these bits are received with an error:

1 1 0 0 1 1 1 1 1 1 0 1

1 1 0 0 <u>0</u> 1 1 1 1 1 0 1

Error Correction Code

- Bit 1 gets checked (positions 1, 3, 5, 7, 9, and so on):
  - Odd parity; **Error detected**

1 1 **0** 0 <u>**0**</u> 1 **1** 1 **1** 1 **0** 1

- Bit 2 gets checked (positions 2, 3, 6, 7, 10, 11, 14, 15, and so on):
  - Even parity; **No error detected**

1 **1** **0** 0 <u>0</u> **1** **1** 1 1 **1** **0** 1

Error Correction Code

- Bit 4 gets checked (positions 4-7, 12-15, 20-23, and so on):
  - Odd parity; **Error detected**

1 1 0 **0** <u>**0**</u> **1** **1** 1 1 1 0 **1**

- Bit 8 gets checked (positions 8-15, 24-31, 40-47, and so on):
  - Even parity; **No error detected**

1 1 0 0 <u>0</u> 1 1 **1** **1** **1** **0** **1**

Error Correction Code

- **Bit 1; Error detected** 1 1 0 0 0 1 1 1 1 1 1 1
- Bit 2; No error detected 1 1 0 0 0 1 1 1 1 1 1 1
- **Bit 4; Error detected** 1 1 0 0 0 1 1 1 1 1 1 1
- Bit 8; No error detected 1 1 0 0 0 1 1 1 1 1 1 1

- **Bit 1 + Bit 4 = Bit 5**
  - **Bit 5 is wrong**

1 1 0 0 1 1 1 1 1 1 1 1

Virtual Machines

- A **virtual machine** (VM) is a software-based computer that emulates all major functions of a physical computer system.
- *Process virtual machines* are VMs that only run a single program
  - Like the Java Virtual Machine runs Java programs, but not an entire operating system with emulated hardware.
- *System virtual machines* emulate an entire computer system.
  - Software like VMware and VirtualBox are System VMs

Virtual Machines

- A *hypervisor* (or virtual machine manager) is software that manages virtual machines

- The underlying physical computer that the virtual machine runs on is called the *host*.

Virtual Machines

- Virtual machines have become significant in that they:
  - Allow multiple “machines” to run on (and share) one physical server

  - These machines might have entirely different software and operating system configurations
    - One physical server, many environments to test software on

  - Virtual machines can run on the same server but are isolated from each other.
    - If one is compromised, it doesn’t necessarily compromise the entire server.

Virtual Machines

- Many instruction sets were not created with virtualization in mind.
  - X86, MIPS, and ARM, to name a few

- The virtual machine cannot execute ISA-level instructions on the host system if the ISA does not support virtualization.
  - The virtualized system can only interact with virtualized resources

Virtual Machines

- Many instruction sets were not created with virtualization in mind.
  - X86, MIPS, and ARM, to name a few

- The virtual machine cannot execute ISA-level instructions on the host system if the ISA does not support virtualization.
  - The virtualized system can only interact with virtualized resources

Virtual Memory

- **Virtual memory** is when the system's main memory gets used as cache for secondary storage devices (i.e., hard drives)

- Virtual memory allows for programs to only use a portion of main memory.
  - The program's address space is a range of virtual memory addresses available to the program.

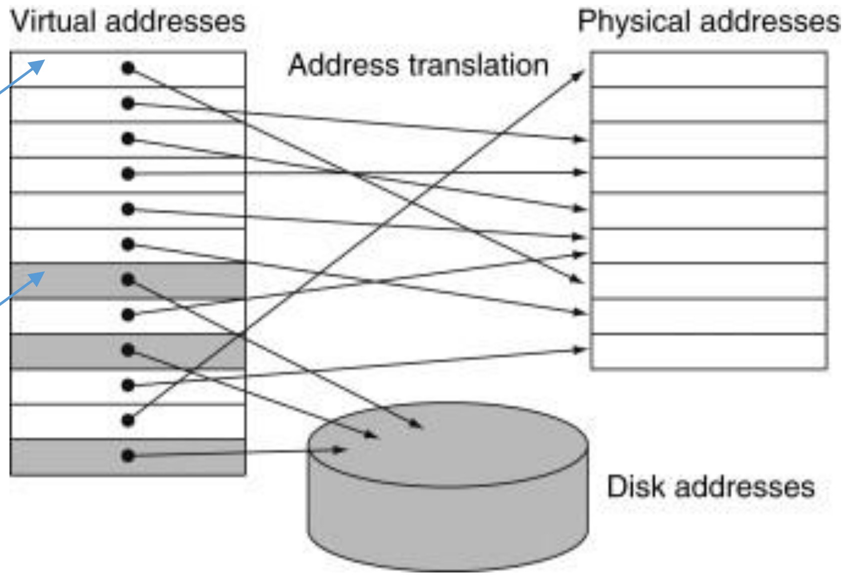
- Virtual memory translates the virtual addresses to real, physical addresses of main memory
  - Called *address mapping* or *address translation*

Virtual Memory

- Virtual memory also expands the capacity of main memory.
  - Data that can't fit in main memory are stored in secondary storage devices.

Data stored in main memory

Data stored on the hard disk



Virtual Memory

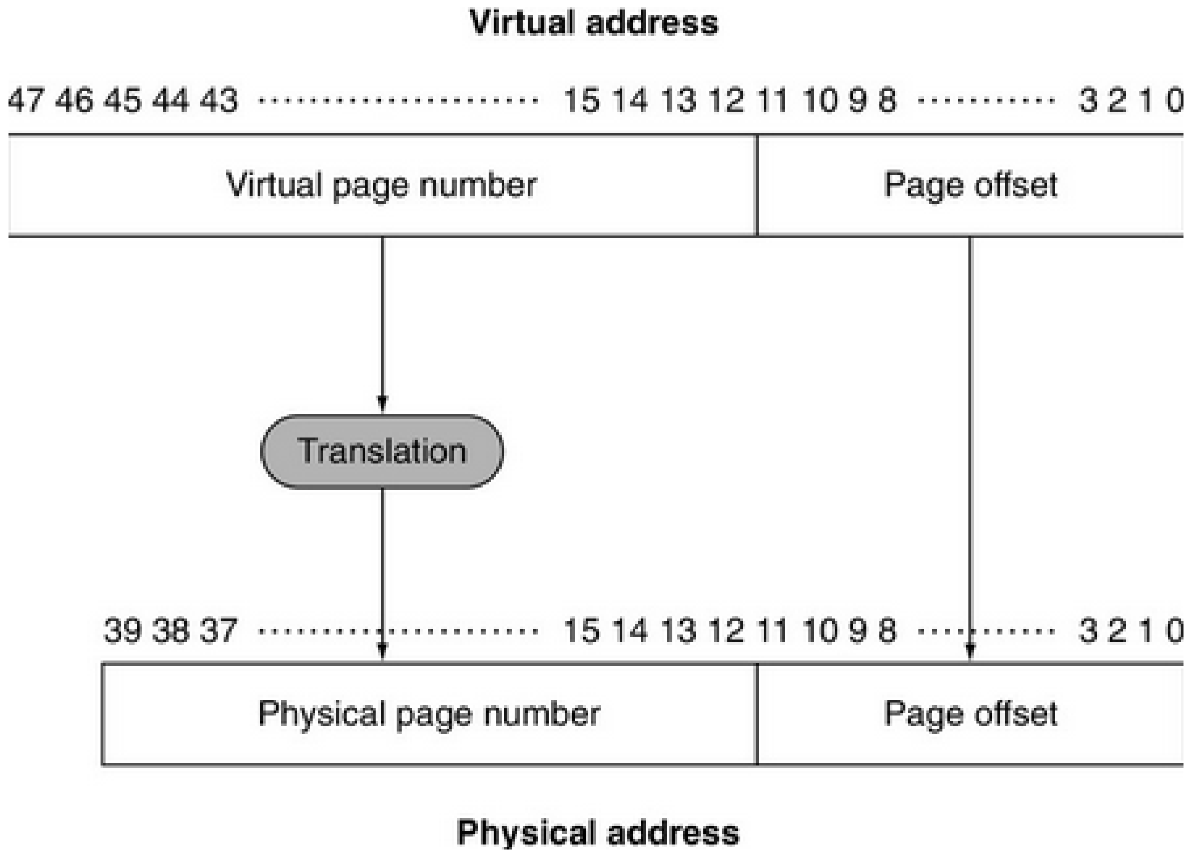
- Virtual memory and caches operate on similar principles.
- Blocks of virtual memory are referred to as *pages.*
- Like when cache memory experiences a cache miss, virtual memory experiences a *page fault* when the page sought is not present.

Virtual Memory

- Virtual memory addresses are split into two sections: the virtual page number and page offset

- The virtual page number is translated to a physical page number
  - The physical page number will be the upper portion of a physical address
  - The page offset remains unchanged and is the lower portion of a physical address.

Virtual Memory



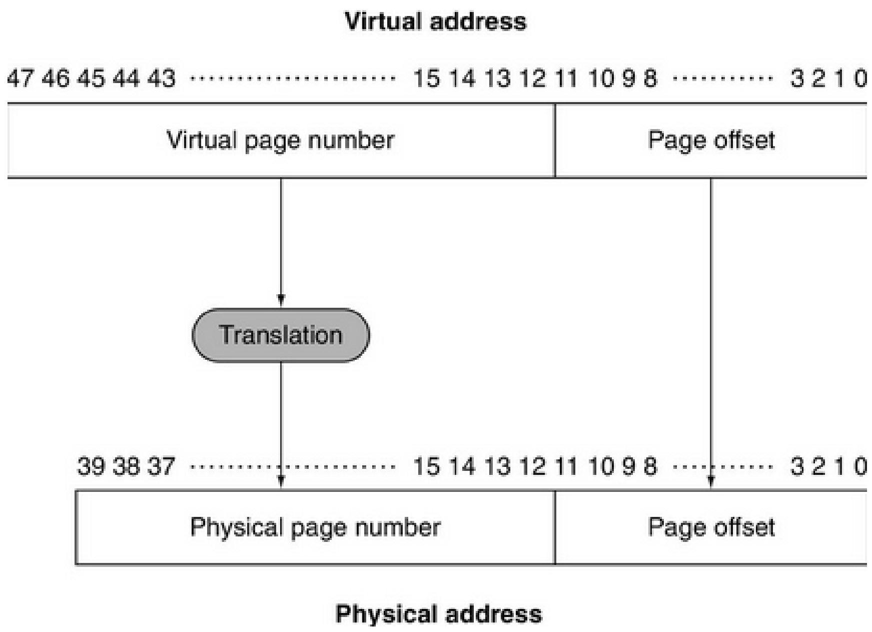
Virtual Memory

$$Page\ Size = 2^{Number\ of\ Offset\ Bits}$$

$$Physical\ Pages = 2^{(Physical\ address\ length - Number\ of\ Offset\ Bits)}$$

$$Address\ Space = 2^{(Address\ length)}$$

Virtual Memory



- $Page\ Size = 2^{12} = 4\ KiB$
- $Physical\ Pages = 2^{40-12} = 2^{28}$
- Virtual Memory
  - $Address\ Space = 2^{48} = 256\ TiB$
- Physical Memory
  - $Address\ Space = 2^{40} = 1\ TiB$

Virtual Memory

- Page faults to the hard disk are extremely time consuming.
  - By comparison, a page fault to main memory is about 100,000 times faster
- Pages should be large enough to compensate for high access times.
  - 4KiB to 16KiB are typical
- Optimizing page placement reduces the frequency of page faults.

Virtual Memory

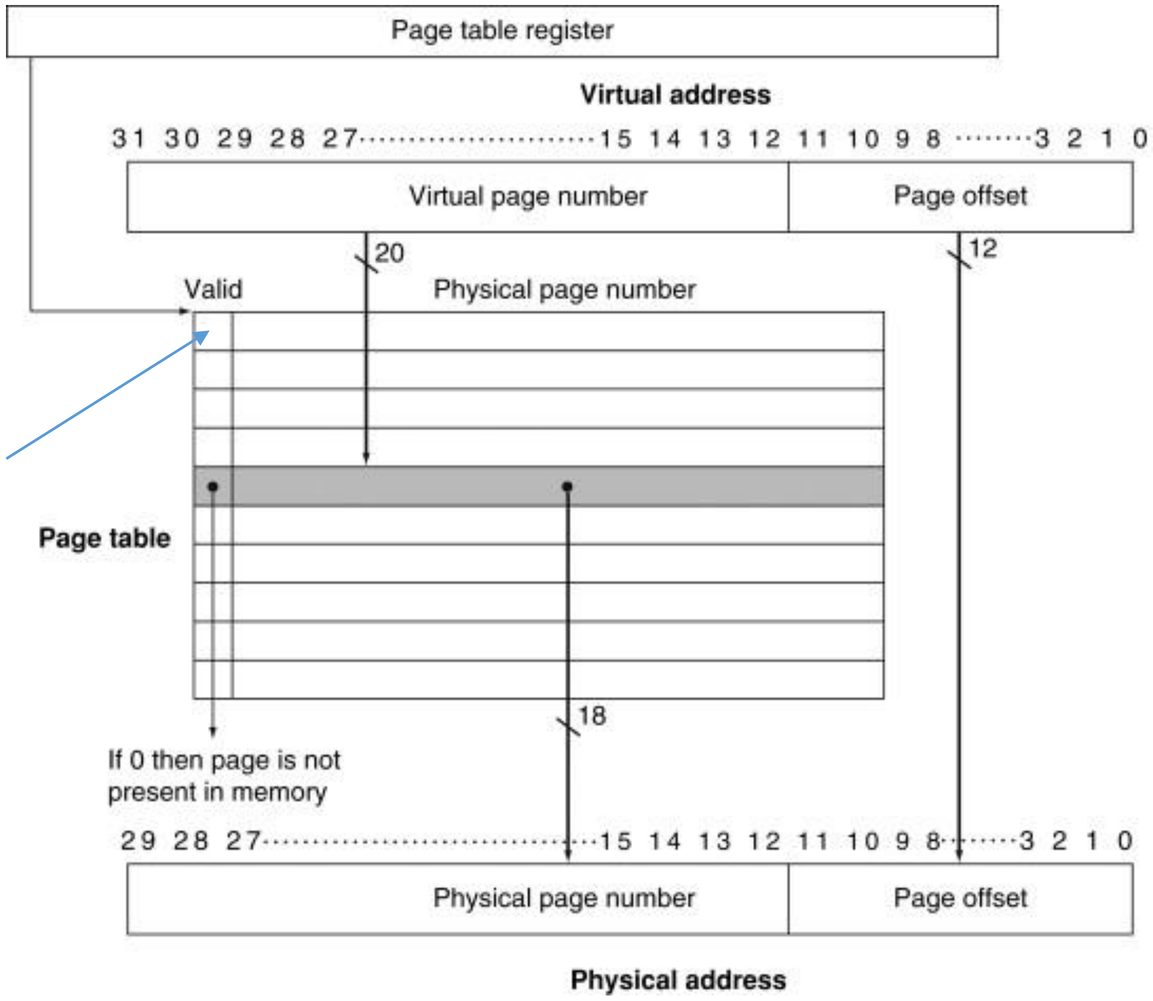
- Virtual memory systems use a *page table* to keep track of the virtual-to-physical memory translations.
  - Usually indexed by page number
  - Each table entry contains the physical page number for a virtual page number, provided the page is in memory

- Each program has its own page table that maps the virtual address space to physical memory addresses.

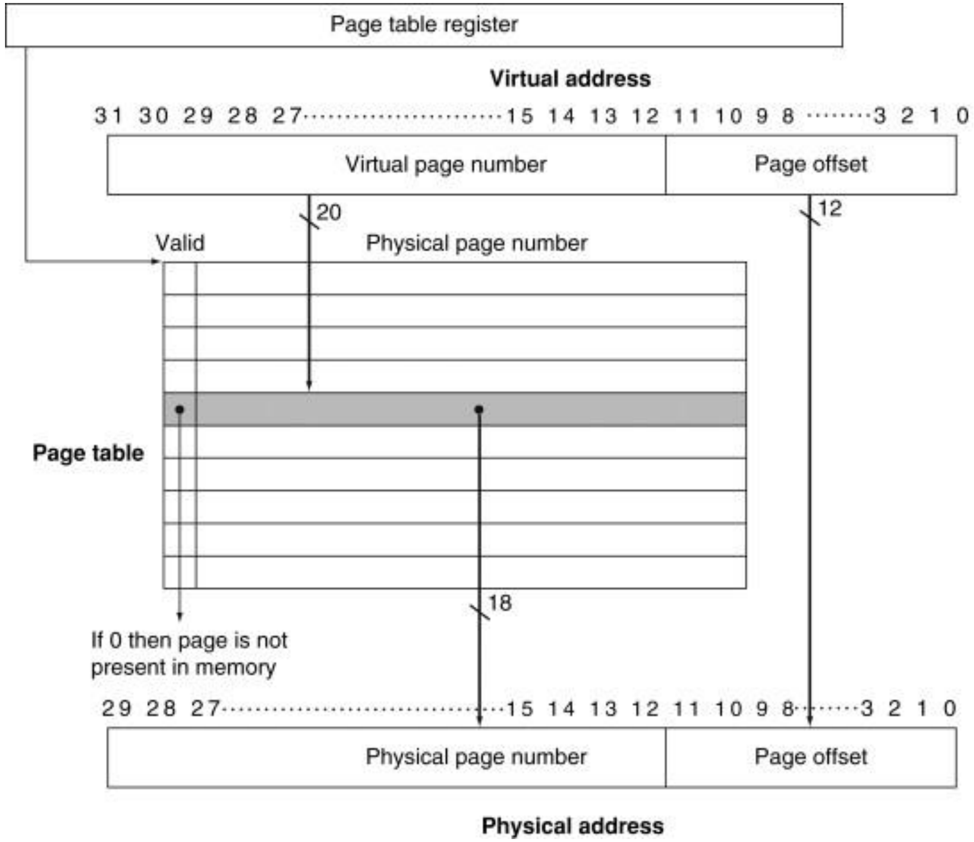
- The *page table register* points to the start of the page table

Virtual Memory

The valid bit is like valid/dirty bits in cache memory.
0 will indicate a page fault.



Virtual Memory



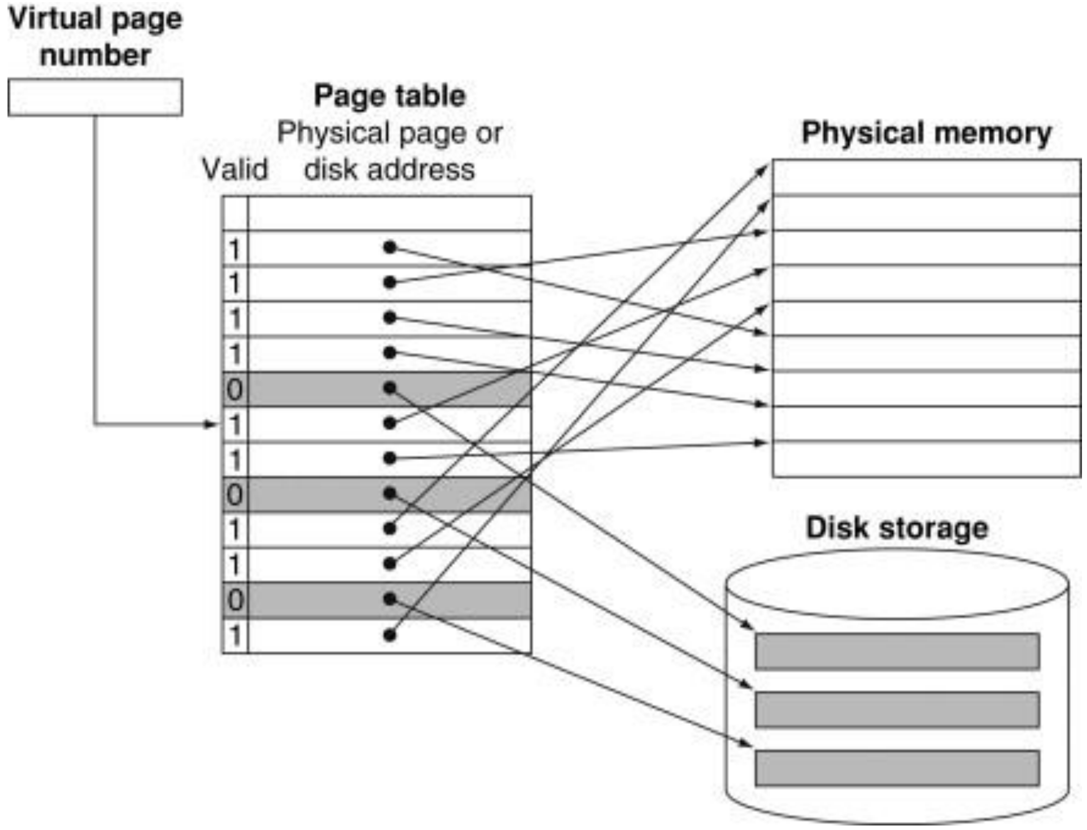
$$Page\ Table\ Entries = 2^{Virtual\ page\ number\ length} = 2^{20} = 1{,}048{,}576\ entries$$

Virtual Memory

- The virtual address alone does not indicate where the page is on a hard disk.
  - The location of each page on disk in the virtual address space must be kept track of.

- The operating system creates *swap space* on the disk for all the pages of a process when the process is created.
  - Also creates records of where each virtual page is stored on disk.

Virtual Memory



- If the valid bit is off, the page is on the hard disk
- The table of physical page addresses and disk page addresses, while logically one table, is stored in two separate data structures.
  - Dual tables are justified in part because we must keep the disk addresses of all the pages, even if they are currently in main memory

Rotational Hard Disks

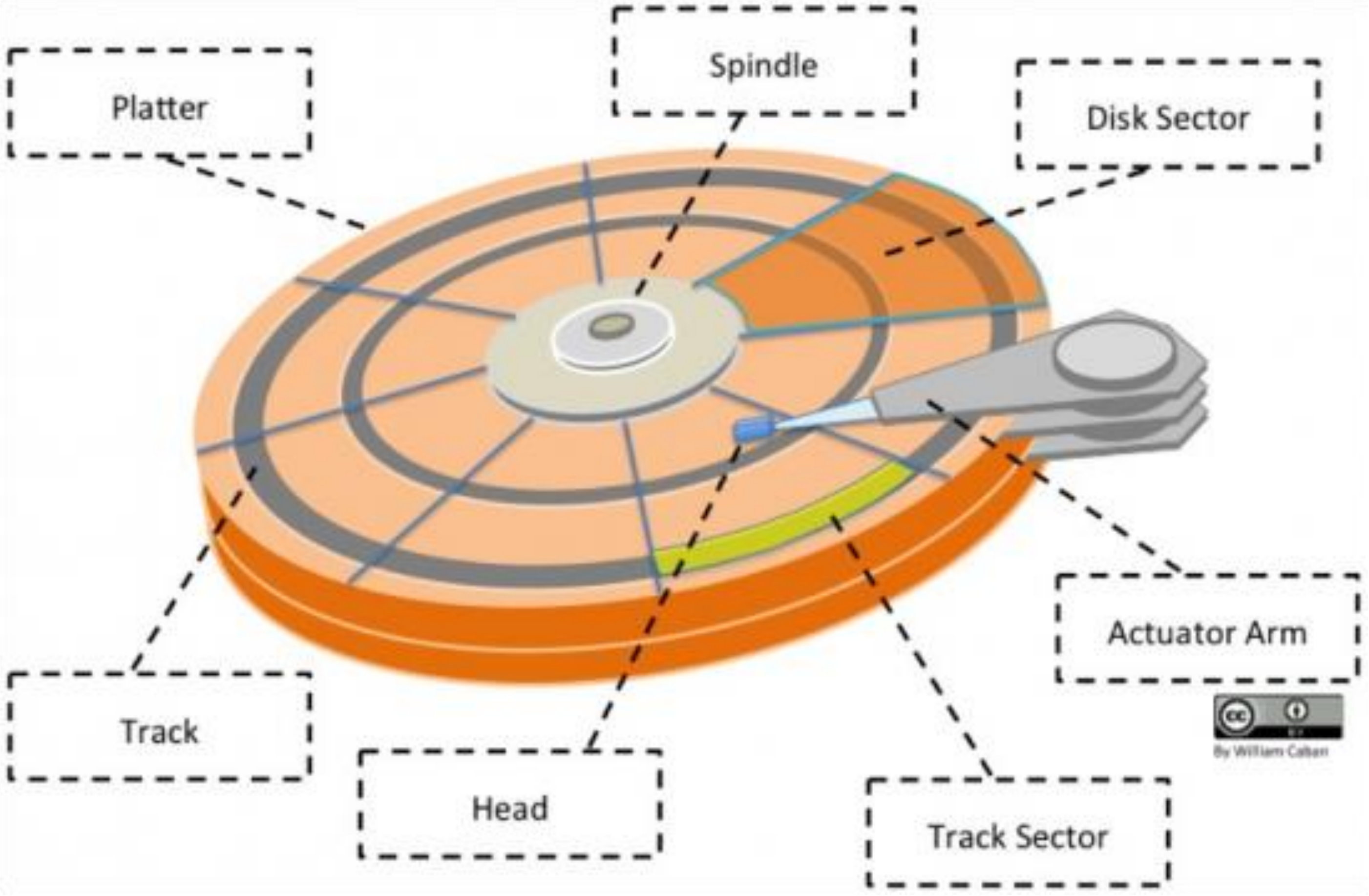
- Rotational hard disks consist of metal *platters* that rotate on a *spindle* at 5400 to 15,000 revolutions per minute.
  - The platters are covered with magnetic recording material on both sides.

- An *actuator arm* contains a small electromagnetic coil called a *head* is located just above the surface of each platter.
  - The actuator arm moves back and forth across the disk
  - The head reads and writes data to the disk.

Rotational Hard Disks

- The surface of a platter is divided into concentric circles called *tracks*.
- Platters are further divided into *sectors*, typically 4096 bytes in size.
  - A track sector is the smallest unit of storage that can be allocated by the disk.
  - For example, a file that is only 100 bytes will still take up one whole track sector; A 5000-byte file will need to use 2 track sectors.

Rotational Hard Disks



Rotational Hard Disks

- Hard disk performance is measured in a few different ways.

- Minimum seek time: The fastest the actuator can move a certain track
  - *Seeking* is the process of the actuator arm moving to a certain track
- Maximum seek time: The slowest the actuator can move a certain track
- Average seek time: Sum of all possible seek times divided by number of possible seeks

Rotational Hard Disks

- One the actuator arm's head reaches the right track, it must wait for the platter to spin to the correct sector
  - Call the *rotational latency*

- Average rotational latency is one half of a rotation divided by the rotations per minute.
- For a disk with 5400 RPMs:

$$Average\ rotational\ latency = \frac{0.5\ rotations}{5400\ RPMs}$$
$$= \frac{0.5\ rotations}{5400\ RPMs/60\,\frac{seconds}{minute}} = 0.0056\ seconds = 5.6\ ms$$