Lecture 18b:

I/O

Introduction to Computer Architecture
UC Davis EEC 170, Fall 2019

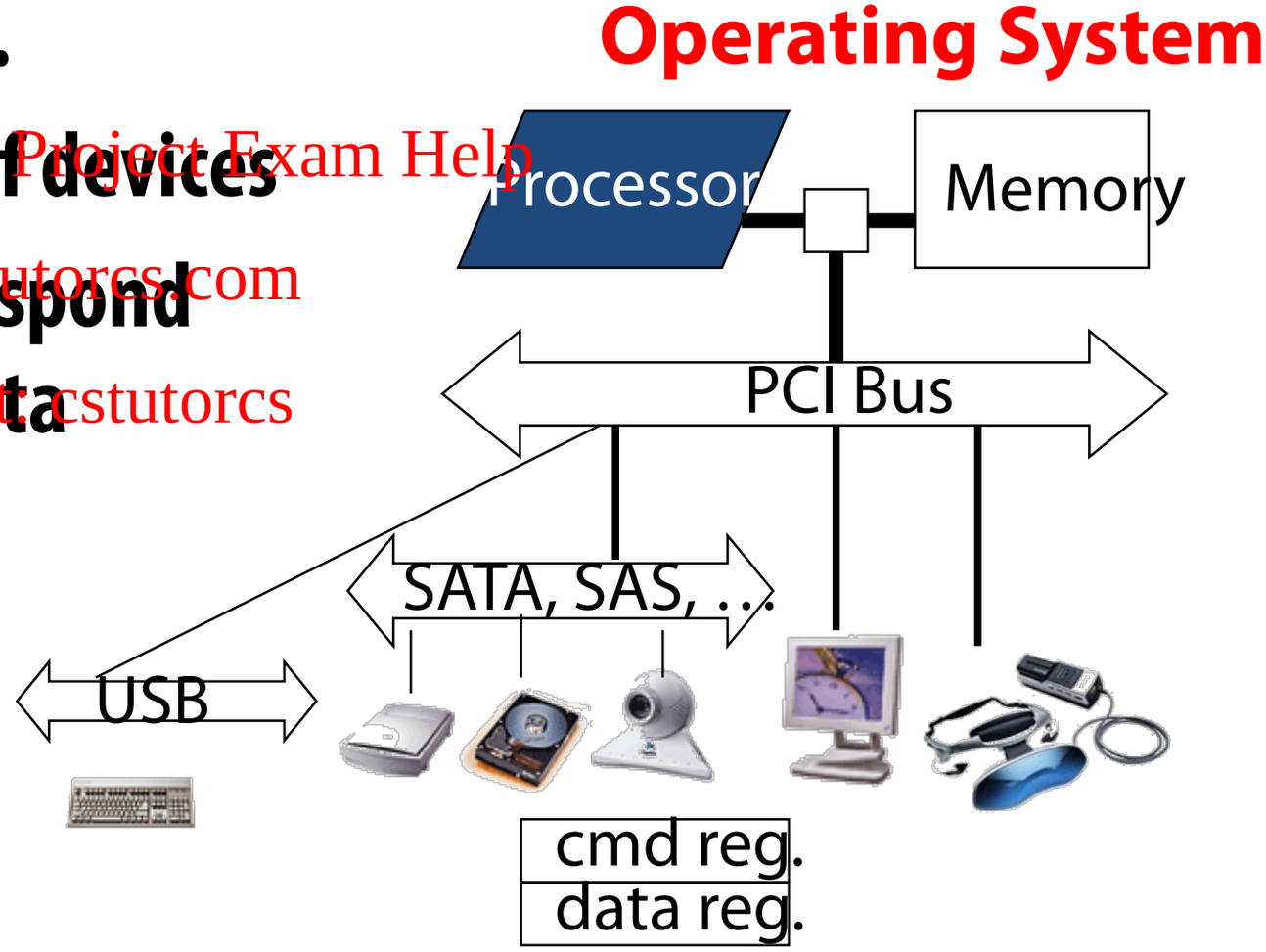
Lecture derived from Randy Katz, UC Berkeley

Agenda

- **Devices and I/O**
- Polling
- Interrupts
- ~~OS Boot Sequence~~
- ~~Multiprogramming/time-sharing~~

How to Interact with Devices?

- Assume a program running on a CPU. How does it interact with the outside world?
- Need I/O interface for Keyboards, Network, Mouse, Screen, etc.
  - Connect to many types of devices
  - Control these devices, respond to them, and transfer data
  - Present them to user programs so they are useful



Instruction Set Architecture for I/O

- What must the processor do for I/O?
  - Input: read a sequence of bytes
  - Output: write a sequence of bytes
- Interface options

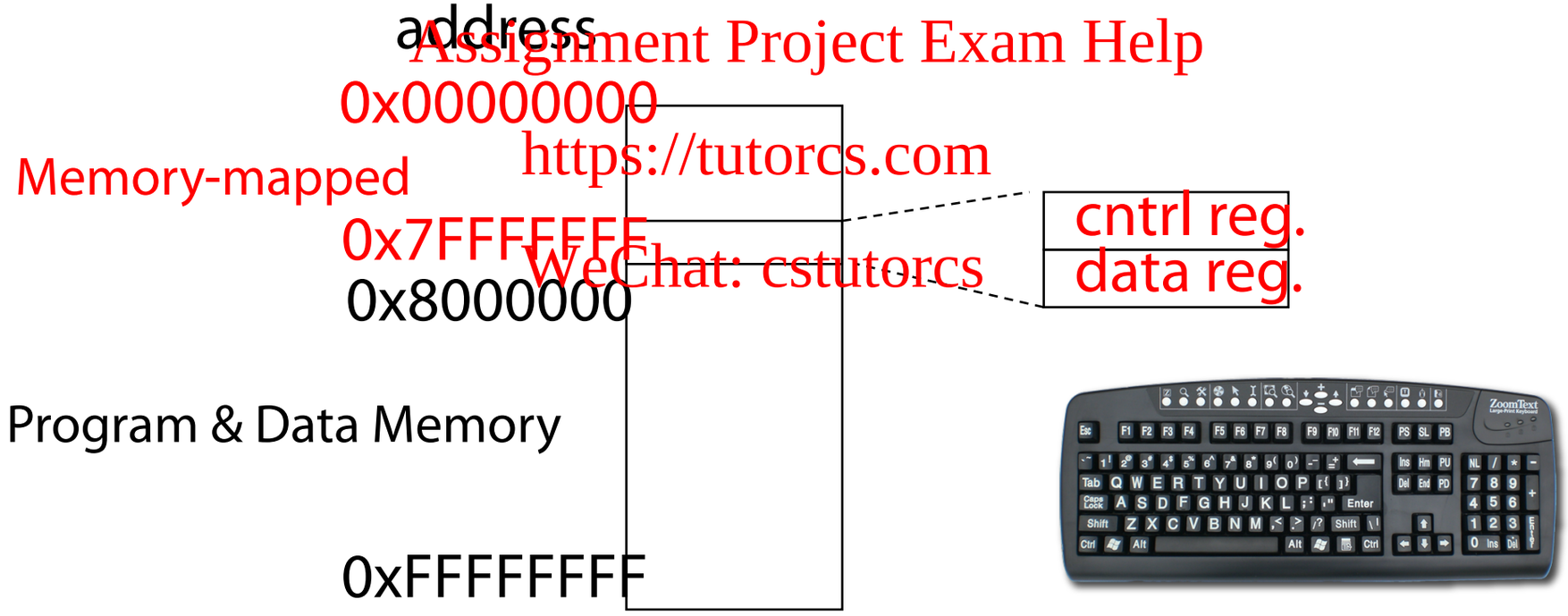
a) Special input/output instructions & hardware

b) Memory mapped I/O

  - Portion of address space dedicated to I/O
  - I/O device registers there (no memory)
  - Use normal load/store instructions, e.g. `lw/sw`
  - Very common, used by RISC-V

Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices



Processor-I/O Speed Mismatch

- 1 GHz microprocessor I/O throughput:
  - 4 Gi-B/s (`lw`/`sw`)
  - Typical I/O data rates:
    - 10 B/s (keyboard)
    - 100 Ki-B/s (Bluetooth)
    - 60 Mi-B/s (USB 2)
    - 100 Mi-B/s (Wifi, depends on standard)
    - 125 Mi-B/s (G-bit Ethernet)
    - 550 Mi-B/s (cutting edge SSD)
    - 1.25 Gi-B/s (USB 3.1 Gen 2)
    - 6.4 GiB/s (DDR3 DRAM)
  - These are peak rates – actual throughput is lower
- Common I/O devices neither deliver nor accept data matching processor speed

Agenda

- Devices and I/O
- **Polling**
- Interrupts
- ~~OS Boot Sequence~~
- ~~Multiprogramming/time-sharing~~

Processor Checks Status before Acting

- Device registers generally serve two functions:
  - Control Register, says it's OK to read/write (I/O ready) [think of a flagman on a road]
  - Data Register, contains data
- Processor reads from Control Register in loop
  - Waiting for device to set Ready bit in Control reg (0 → 1)
  - Indicates "data available" or "ready to accept data"
- Processor then loads from (input) or writes to (output) data register
  - I/O device resets control register bit (1 → 0)
- Procedure called "Polling"

I/O Example (Polling)

- **Input: Read from keyboard into `a0`**

```
                lui   t0,0x7ffff #7ffff000 (io addr)
Waitloop:       lw    t1,0(t0)   #read control
                andi  t1,t1,0x1  #ready bit
                beq   t1,zero,Waitloop
                lw    a0,4(t0)   #data
```


- **Output: Write to display from `a1`**


```
                lui   t0,0x7ffff #7ffff0000
Waitloop:       lw    t1,8($t0)  #write control
                andi  t1,t1,0x1  #ready bit
                beq   t1,zero,Waitloop
                sw    a1,12(t0)  #data
```


"Ready" bit is from processor's point of view!

Cost of Polling?

- Assume for a processor with
  - 1 GHz clock rate
  - Taking 400 clock cycles for a polling operation
    - Call polling routine
    - Check device (e.g., keyboard or wifi input available)
    - Return
  - What's the percentage of processor time spent polling?
- Example:
  - Mouse
  - Poll 30 times per second
    - Set by requirement not to miss any mouse motion (which would lead to choppy motion of the cursor on the screen)

Peer Instruction

Hard disk: transfers data in 16-Byte chunks and can transfer at 16 MB/second. No transfer can be missed. What percentage of processor time is spent in polling (assume 1 GHz clock)?

- **2%**
- **4%**
- **20%**
- **40%**

What is the Alternative to Polling?

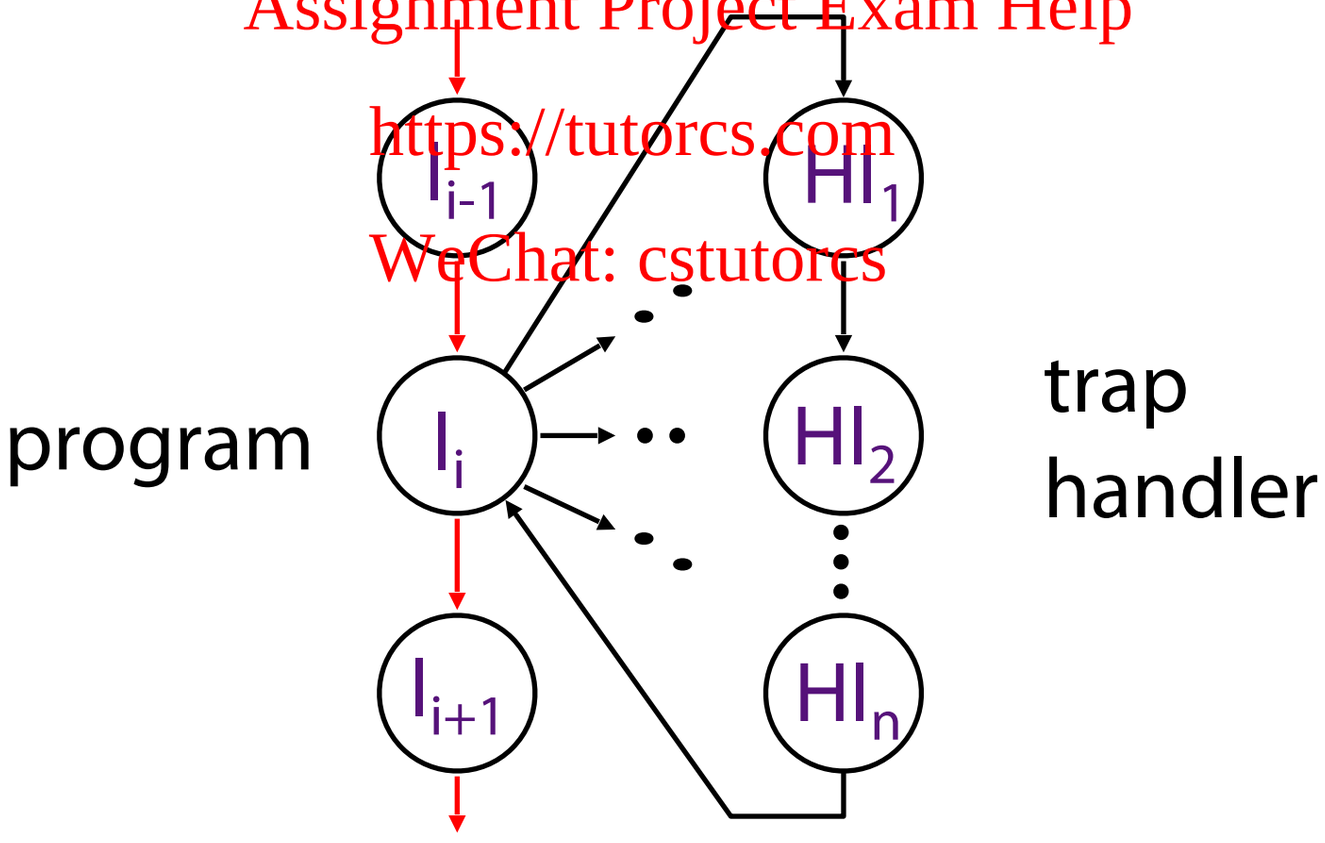
- Polling wastes processor resources
- Akin to waiting at the door for guests to show up
  - What about a bell?
- Computer lingo for bell:
  - **Interrupt**
  - Occurs when I/O is ready or needs attention
    - Interrupt current program
    - Transfer control to special code **"interrupt handler"**

Agenda

- Devices and I/O
- Polling
- **Interrupts**
- ~~OS Boot Sequence~~
- ~~Multiprogramming/time-sharing~~

Traps/Interrupts/Exceptions: altering the normal flow of control

- An external or internal event that needs to be processed – by another program – the OS. The event is often unexpected from original program's point of view.



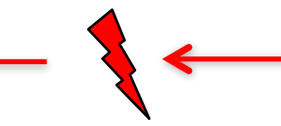
Interrupt-Driven I/O

Handler Execution
Stack Frame
Stack Frame
Stack Frame

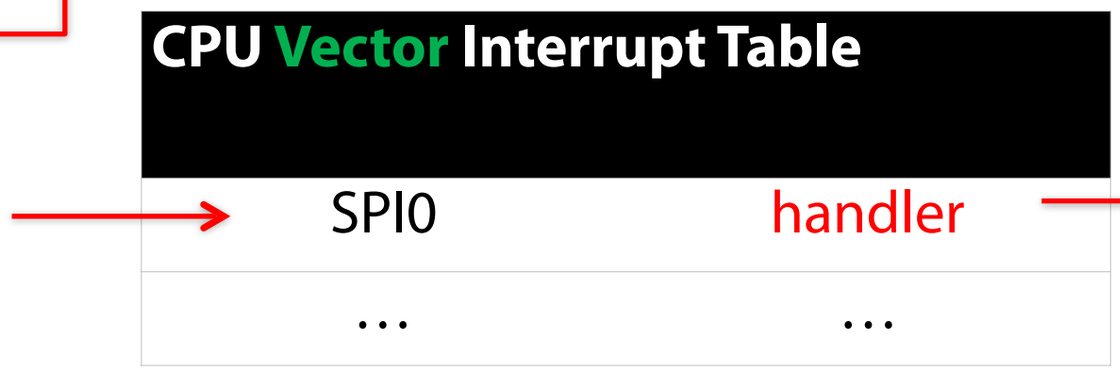
1. Incoming interrupt suspends instruction stream
2. Looks up the vector (function address) of a handler in an interrupt vector table stored within the CPU
3. Perform a jal to the handler (save PC in *special* MEPC* register)
4. Handler run on current stack and returns on finish (thread doesn't notice that a handler was run)


```
handler:   save registers
           decode interrupt cause
           handle interrupt
             clear interrupt
           restore registers
           mret
```

```
Label: sll   t1,s3,2
       add   t1,t1,s5
       lw    t1,0(t1)
       or    s1,s1,t1
       add   s3,s3,s4
       bne   s3,s2,Label
```



Interrupt (SPI0)



*MEPC: Machine Exception Program Counter

Terminology

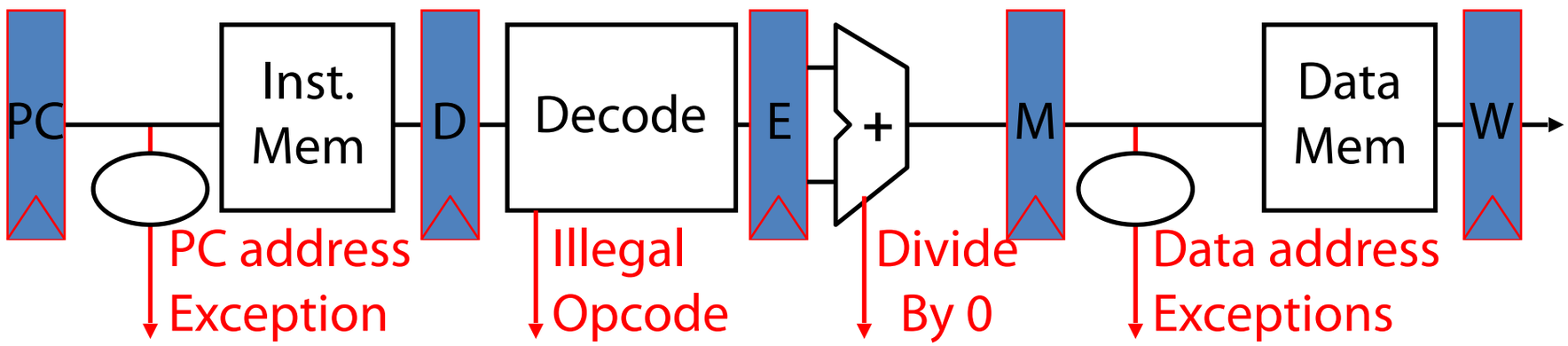
In this class (other definitions in use elsewhere):

- **Interrupt – caused by an event *external* to current running program**
  - E.g., key press, disk I/O
  - Asynchronous to current program
    - Can handle interrupt on any convenient instruction
    - "Whenever it's convenient, just don't wait too long"
- **Exception – caused by some event *during* execution of one instruction of current running program**
  - E.g., divide by zero, bus error, illegal instruction
  - Synchronous
    - Must handle exception *precisely* on instruction that causes exception
    - "Drop whatever you are doing and act now"
- **Trap – action of servicing interrupt or exception by hardware jump to "interrupt or trap handler" code**

Precise Traps

- ***Trap handler's view of machine state is that every instruction prior to the trapped one (e.g., overflow) has completed, and no instruction after the trap has executed.***
- **Implies that handler can return from an interrupt by restoring user registers and jumping back to interrupted instruction**
  - **Interrupt handler software doesn't need to understand the pipeline of the machine, or what program was doing!**
  - **More complex to handle trap caused by an exception than interrupt**
- **Providing precise traps is tricky in a pipelined superscalar out-of-order processor!**
  - **But a requirement, e.g., for**
    - **Virtual memory to function properly (see next lecture)**

Trap Handling in 5-Stage Pipeline



- Exceptions are handled like *pipeline hazards*

- Complete execution of instructions before exception occurred
- Flush instructions currently in pipeline (i.e., convert to nops or "bubbles")
- Optionally store exception cause in status register
  - Indicate type of exception
  - Note: several exceptions can occur in a single clock cycle!
- Transfer execution to trap handler

Trap Pipeline Diagram

	time t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: DIV	IF_1	ID_1	EX_1	MEM_1	-		*divide by zero!*		
(I_2) 100: XOR		IF_2	ID_2	EX_2	-				
(I_3) 104: SUB			IF_3	ID_3	-	-	-		
(I_4) 108: ADD				IF_4	-	-	-	-	
(I_5) Trap Handler code*					IF_5	ID_5	EX_5	MEM_5	WB_5

*MEPC = 100 (instruction following offending DIV)

Review: I/O

- "Memory mapped I/O": Device control/data registers mapped to CPU address space
- CPU synchronizes with I/O device:
  - Polling
  - Interrupts
- "Programmed I/O":
  - CPU execs lw/sw instructions for all data movement to/from devices
  - CPU spends time doing two things:
    1. Getting data from device to main memory
    2. Using data to compute

Reality Check!

- "Memory mapped I/O": Device control/data registers mapped to CPU address space
- CPU synchronizes with I/O device:
  - Polling
  - Interrupts
- ~~"Programmed I/O":~~ **DMA**
  - ~~CPU execs lw/sw instructions for all data movement to/from devices~~
  - CPU spends time doing ~~2 things~~:
    1. ~~Getting data from device to main memory~~
    2. Using data to compute

Outline

- Direct Memory Access
- Review: Disks
- Networking
- Storage Attachment Evolution
- Rack Scale Memory
- And in Conclusion …

Outline

- **Direct Memory Access**
- Disks
- Networking
- Storage Attachment Evolution
- Rack Scale Memory
- And in Conclusion . . .

What's Wrong with Programmed I/O?

- Not ideal because …
  1. CPU has to execute all transfers, could be doing other work
  2. Device speeds don't align well with CPU speeds
  3. Energy cost of using beefy general-purpose CPU where simpler hardware would suffice

- Until now CPU has sole control of main memory

- 5% of CPU cycles on Google Servers spent in memcpy() and memmove() library routines!*

- * Kanev et al., "Profiling a warehouse-scale computer," ICSA 2015, June 2015, Portland, OR.

Direct Memory Access (DMA)

- Allows I/O devices to directly read/write main memory
- New Hardware: the DMA Engine
- DMA engine contains registers written by CPU:
  - Memory address to place data
  - # of bytes
  - I/O device #, direction of transfer
  - unit of transfer, amount to transfer per burst

Operation of a DMA Transfer

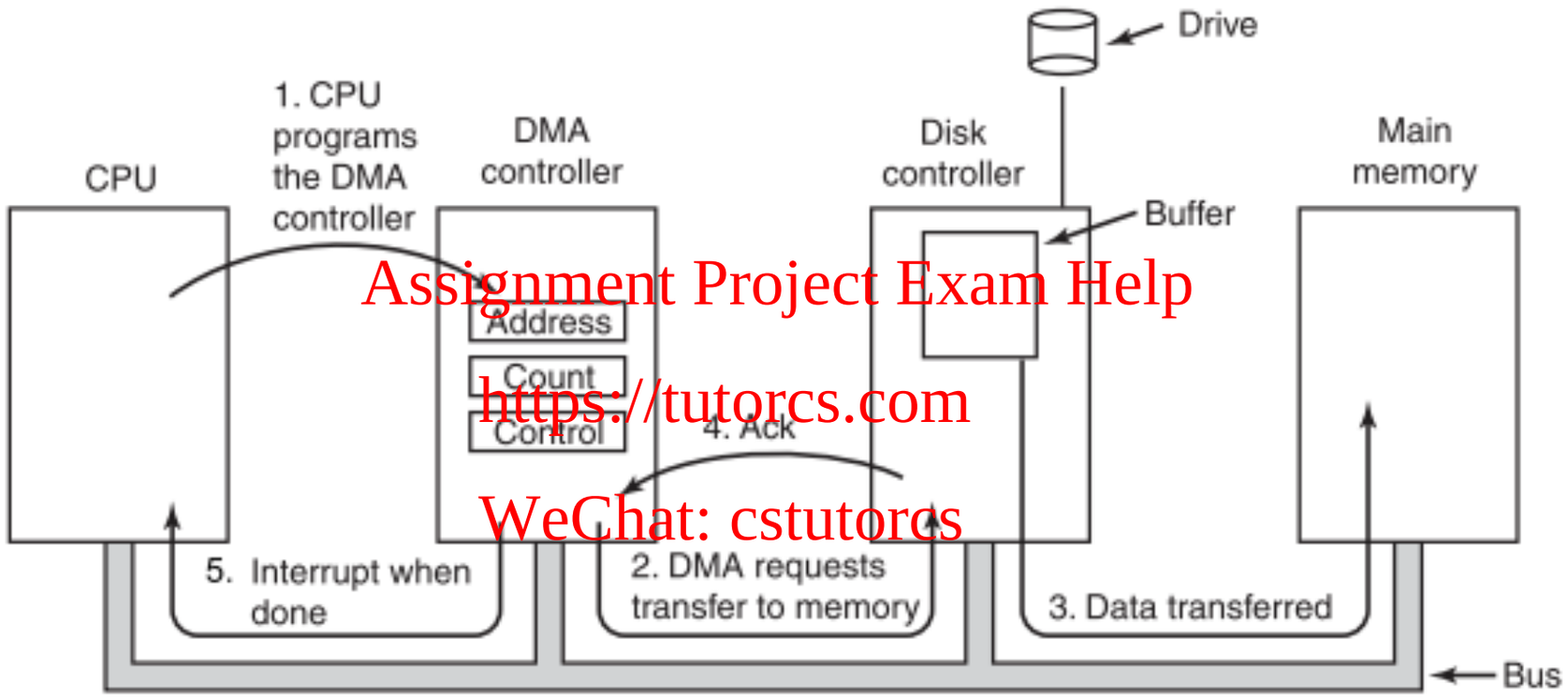


Figure 5-4. Operation of a DMA transfer.

[From Section 5.1.4 Direct Memory Access in *Modern Operating Systems* by Andrew S. Tanenbaum, Herbert Bos, 2014]

DMA: Incoming Data

1. Receive interrupt from device
2. CPU takes interrupt, begins transfer
   - Instructs DMA engine/device to place data @ certain address
3. Device/DMA engine handle the transfer
   - CPU is free to execute other things
4. Upon completion, Device/DMA engine interrupt the CPU again

DMA: Outgoing Data

1. CPU decides to initiate transfer, confirms that external device is ready
2. CPU begins transfer
   - Instructs DMA engine/device that data is available @ certain address
3. Device/DMA engine handle the transfer
   - CPU is free to execute other things
4. Device/DMA engine interrupt the CPU again to signal completion

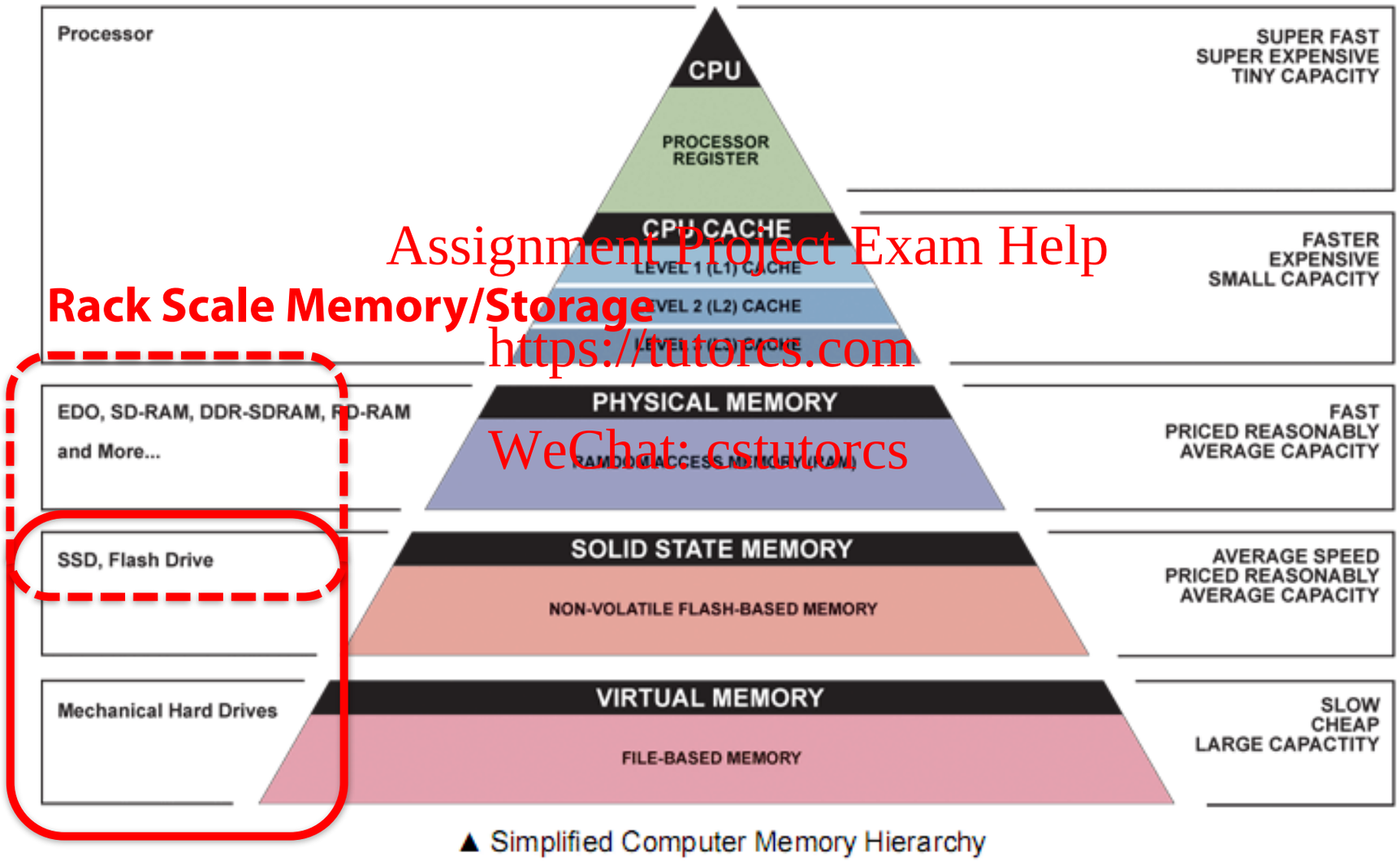
DMA: Some New Problems

- Where in the memory hierarchy do we plug in the DMA engine?
  Two extremes:
  - Between L1$ and CPU:
    - Pro: Free coherency
    - Con: Trash the CPU's working set with transferred data
  - Between Last-level cache and main memory:
    - Pro: Don't mess with caches
    - Con: Need to explicitly manage coherency

Outline

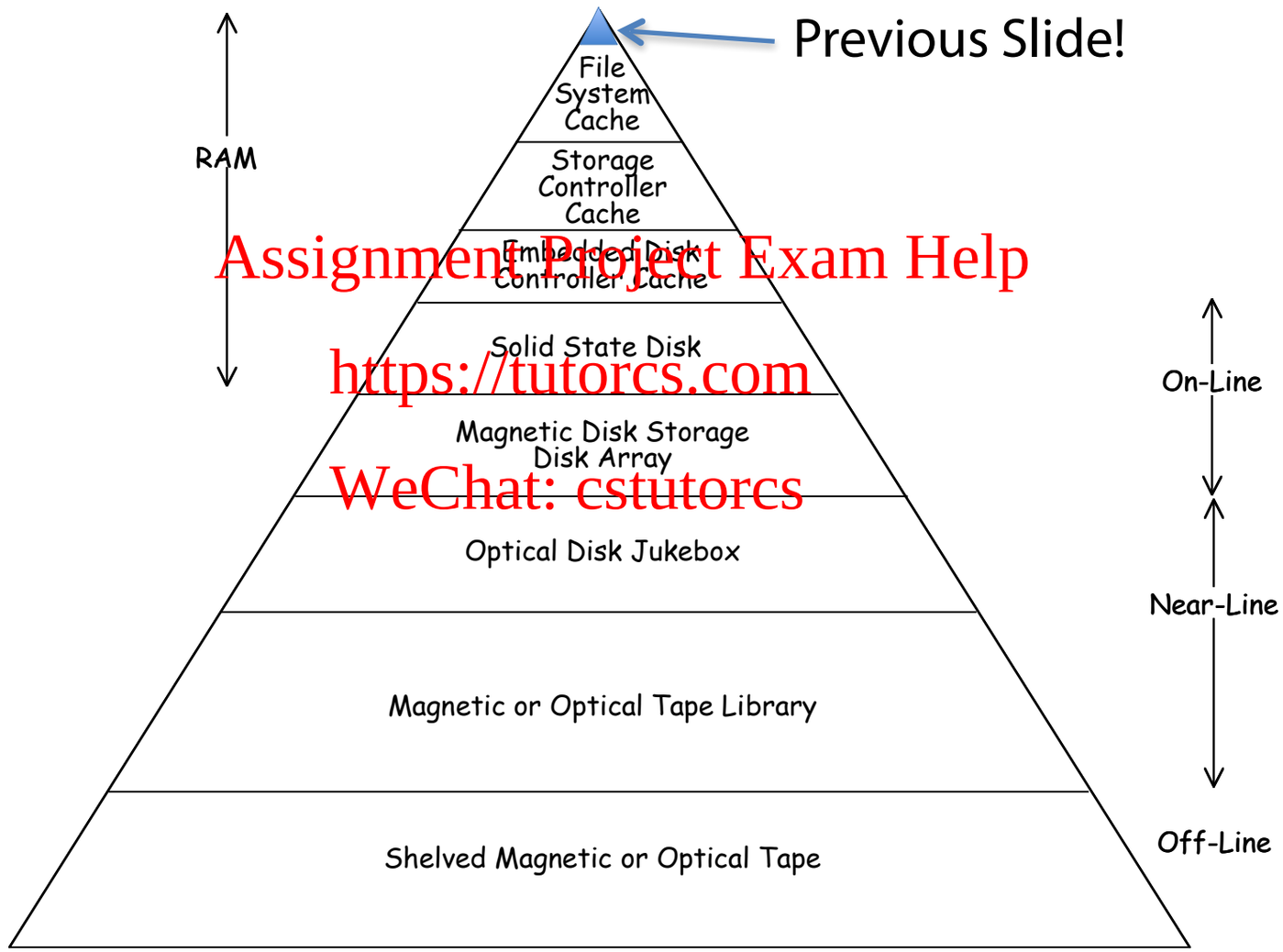
- Direct Memory Access
- **Disks**
- Networking
- And in Conclusion …

Computer Memory Hierarchy: One of our "Great Ideas"

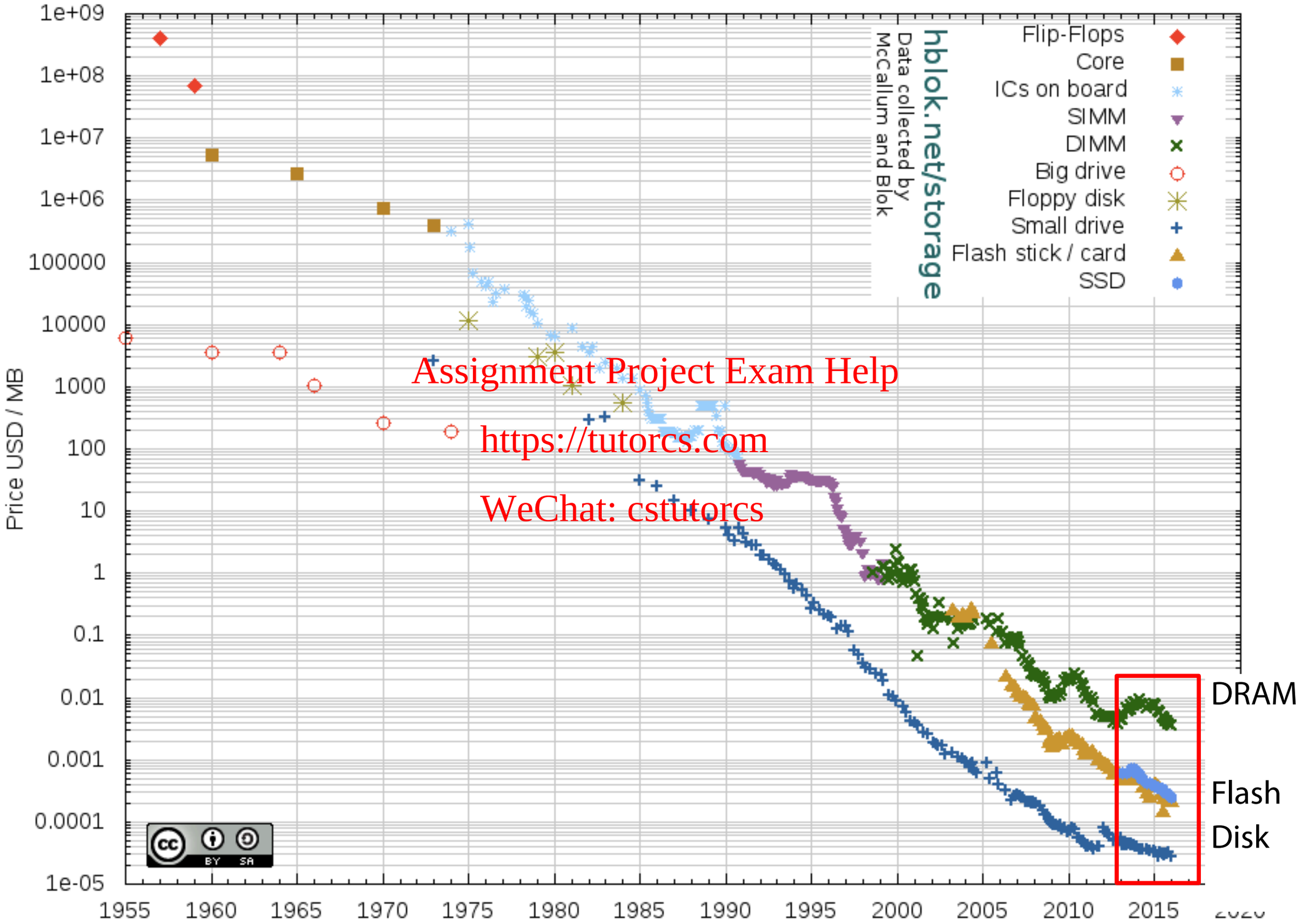


▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

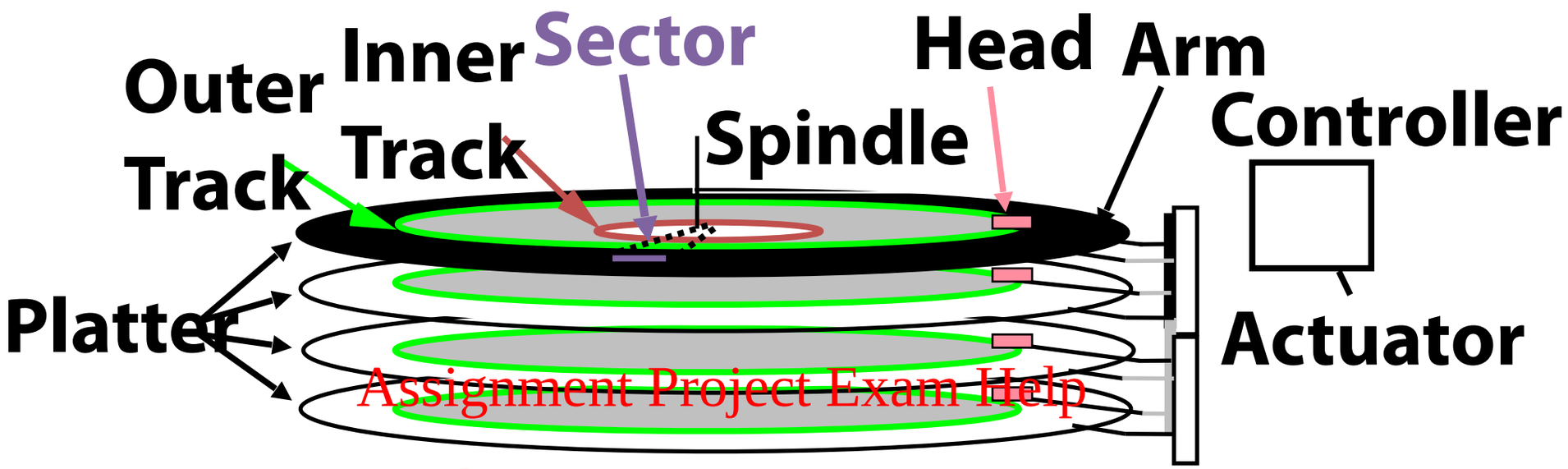
Storage-Centric View of the Memory Hierarchy



Historical Cost of Computer Memory and Storage



Disk Device Performance (1/2)



- Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  - Seek Time = time to position the head assembly at the proper cylinder
  - Rotation Time = time for the disk to rotate to the point where the first sectors of the block to access reach the head
  - Transfer Time = time taken by the sectors of the block and any gaps between them to rotate past the head

Disk Device Performance (2/2)

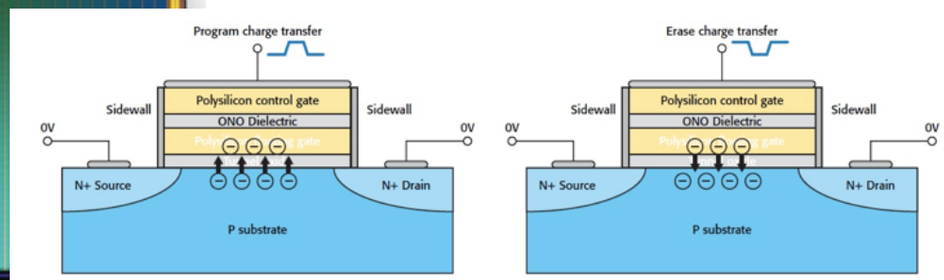
- Average values to plug into the formula:
- Rotation Time: Average distance of sector from head?
  - 1/2 time of a rotation
    - 7200 Revolutions Per Minute ⇒ 120 Rev/sec
    - 1 revolution = 1/120 sec ⇒ 8.33 milliseconds
    - 1/2 rotation (revolution) ⇒ 4.17 ms

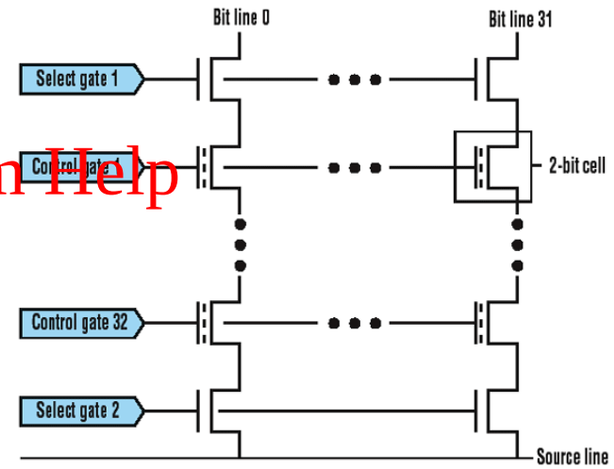
- Seek time: Average no. tracks to move arm?
  - Number of tracks/3 (see EEC 161 for the math)
  - Then, seek time = number of tracks moved × time to move across one track

But wait!

- Performance estimates are different in practice
- Modern disks have on-disk caches, which are hidden from the outside world
  - Generally, what limits real performance is the on-disk cache access time

Flash Memory / SSD Technology





In the basic functional block used in multilevel NAND flash memories, 32 rows of bit lines and 32 control-gate lines form a building block that's repeated many times to form the memory array. The select gate lines are used with the control gate lines to control access to the array.

2. Micron's triple-level cell (TLC) flash memory stores 3 bits of data in each transistor.

- NMOS transistor with an additional conductor between gate and source/drain which "traps" electrons. The presence/absence is a 1 or 0
- Memory cells can withstand a limited number of program-erase cycles. Controllers use a technique called wear leveling to distribute writes as evenly as possible across all the flash blocks in the SSD.

Outline

- Direct Memory Access
- Disks
- **Networking**
- Storage Attachment Evolution
- Rack Scale Memory
- And in Conclusion . . .

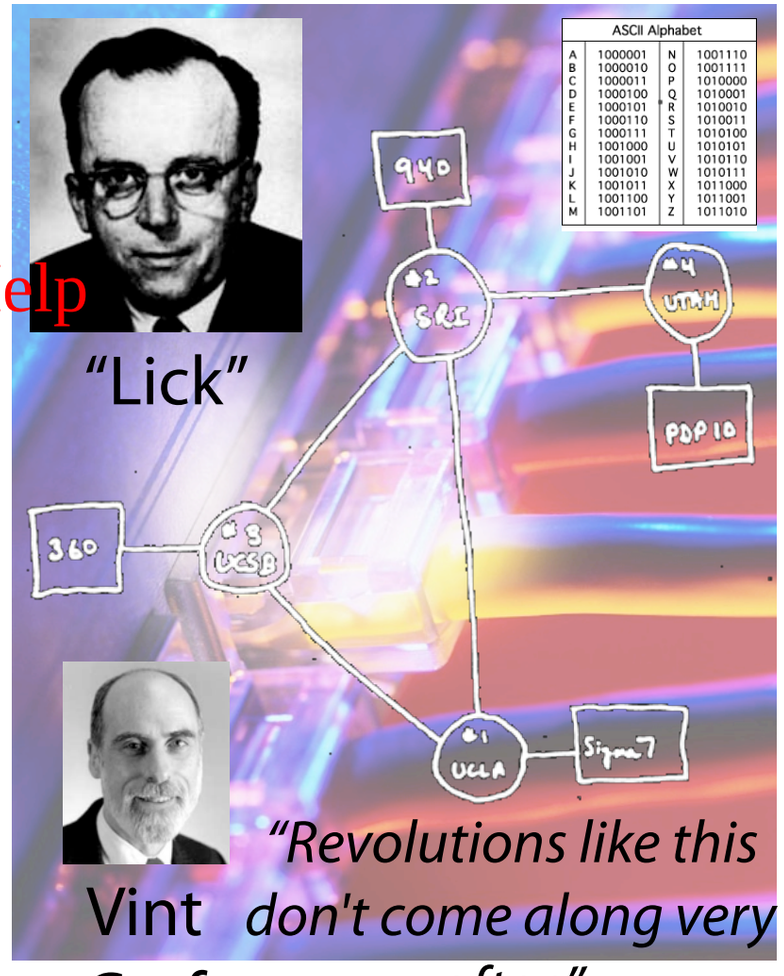
Networks: Talking to the Outside World

- Originally sharing I/O devices between computers
  - E.g., printers
- Then communicating between computers
  - E.g., file transfer protocol
- Then communicating between people
  - E.g., e-mail
- Then communicating between networks of computers
  - E.g., file sharing, www, …

The Internet (1962)

www.computerhistory.org/internet_history

- History
  - 1963: JCR Licklider, while at DoD's ARPA, writes a memo describing desire to connect the computers at various research universities: Stanford, Berkeley, UCLA…
  - 1969 : ARPA deploys 4 "nodes" @ UCLA, SRI, Utah, & UCSB
  - 1973 Robert Kahn & Vint Cerf invent TCP, now part of the Internet Protocol Suite
- Internet growth rates
  - Exponential since start!



www.greatachievements.org/?id=3736
en.wikipedia.org/wiki/Internet_Protocol_Suite

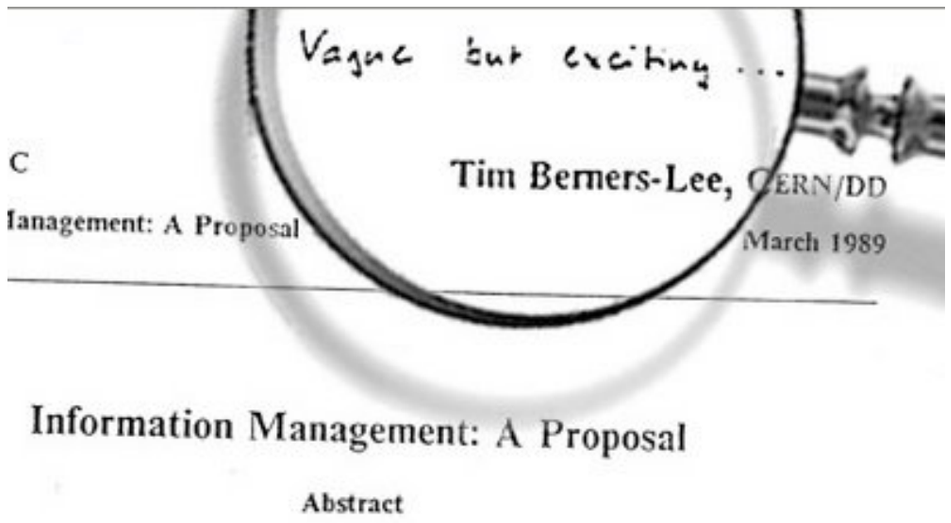
The World Wide Web (1989)

`en.wikipedia.org/wiki/History_of_the_World_Wide_Web`

- "System of interlinked hypertext documents on the Internet"
- History
  - 1945: Vannevar Bush describes hypertext system called "memex" in article
  - 1989: Sir Tim Berners-Lee proposed and implemented the first successful communication between a Hypertext Transfer Protocol (HTTP) client and server using the internet.
  - ~2000 Dot-com entrepreneurs rushed in, 2001 bubble burst
- Today : Access anywhere!

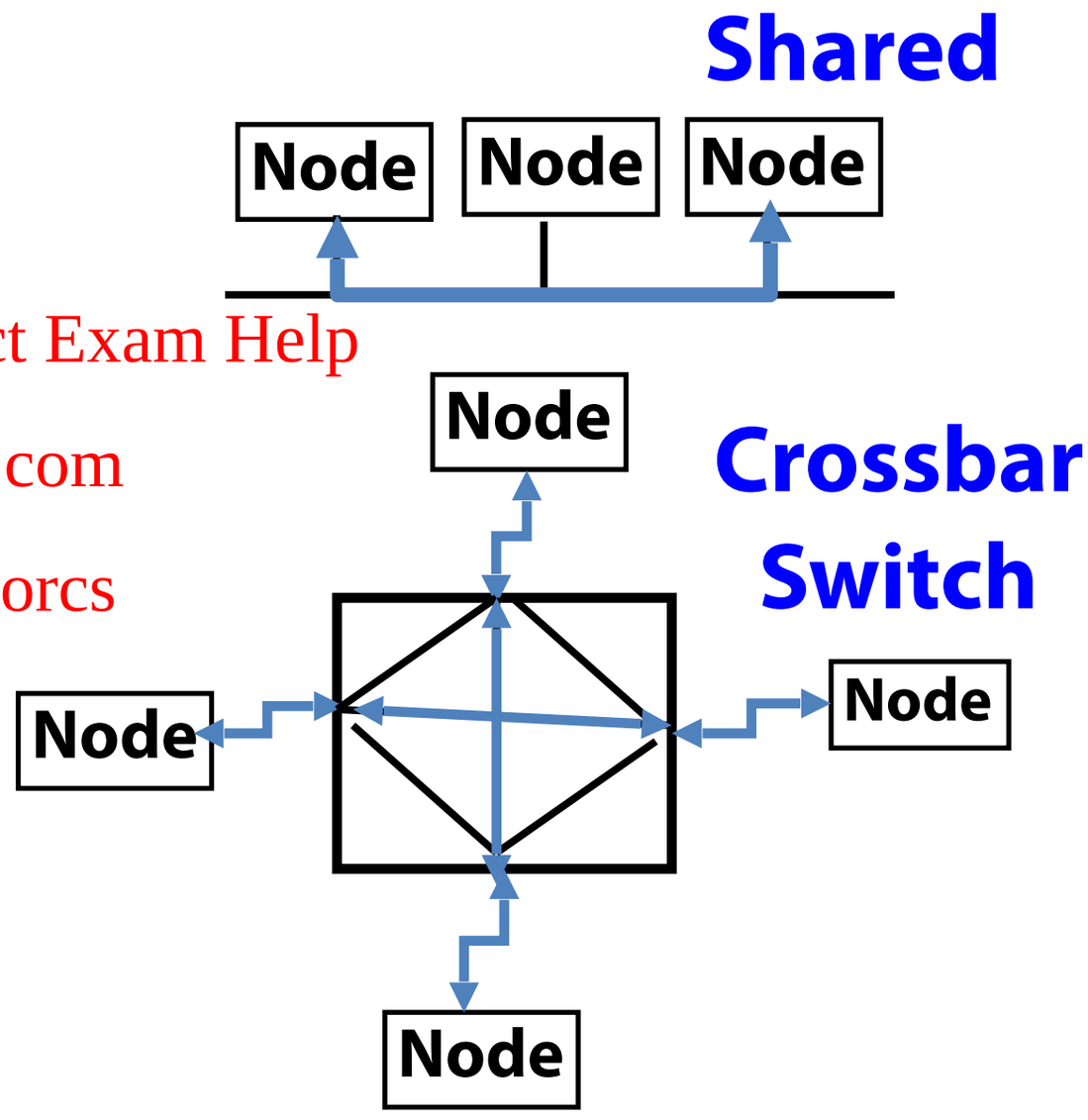
Tim Berners-Lee

World's First web server in 1990



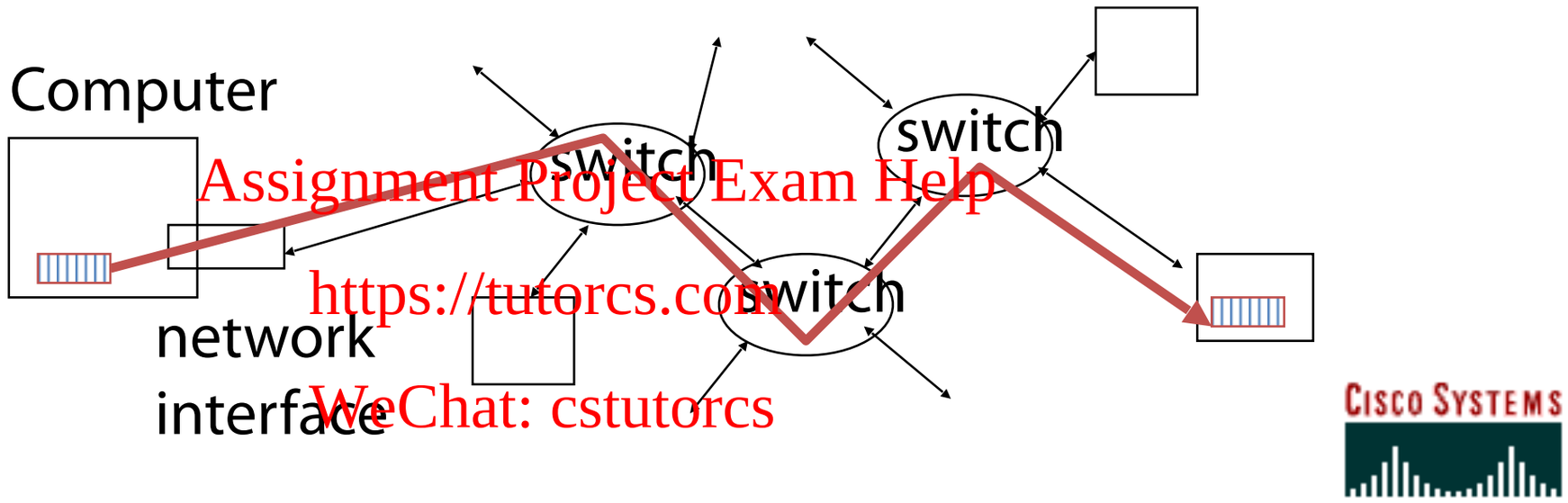
Shared vs. Switch-Based Networks

- Shared vs. Switched:
  - Shared: 1 at a time (CSMA/CD)
  - Switched: pairs ("point-to-point" connections) communicate at same time
- Aggregate bandwidth (BW) in switched network is many times that of shared:
  - Point-to-point faster since no arbitration, simpler interface



What Makes Networks Work?

- Links connecting switches and/or routers to each other and to computers or devices



- Ability to name the components and to route packets of information – messages – from a source to a destination
- Layering, redundancy, protocols, and encapsulation as means of abstraction (big idea)

Software Protocol to Send and Receive

- SW Send steps
  - 1: Application copies data to OS buffer
  - 2: OS calculates checksum, starts timer
  - 3: OS sends data to network interface HW and says start
- SW Receive steps
  - 3: OS copies data from network interface HW to OS buffer
  - 2: OS calculates checksum, if OK, send ACK; if not, delete message (sender resends when timer expires)
  - 1: If OK, OS copies data to user address space, & signals application to continue

Protocols for Networks of Networks?

- What does it take to send packets across the globe?
- Bits on wire or air
- Packets on wire or air
- Pigeons
- Delivery packets within a single physical network
- Deliver packets across multiple networks
- Ensure the destination received the data
- Create data at the sender and make use of the data at the receiver

Protocol for Networks of Networks?

- Lots to do and at multiple levels!
- Use abstraction to cope with complexity of communication
- Networks are like ~~ogres~~ onions
  - Hierarchy of layers:
    - Application (chat client, game, etc.)
    - Transport (TCP, UDP)
    - Network (IP)
    - Data Link Layer (ethernet)
    - Physical Link (copper, wireless, etc.)

Protocol Family Concept

- *Protocol*: packet structure and control commands to manage communication
- *Protocol families* (suites): a set of cooperating protocols that implement the network stack
- Key to protocol families is that communication occurs logically at the same level of the protocol, called peer-to-peer…

…but is implemented via services at the next lower level

- Encapsulation: carry higher level information within lower level "envelope"

Inspiration . . .

Dear Bill,

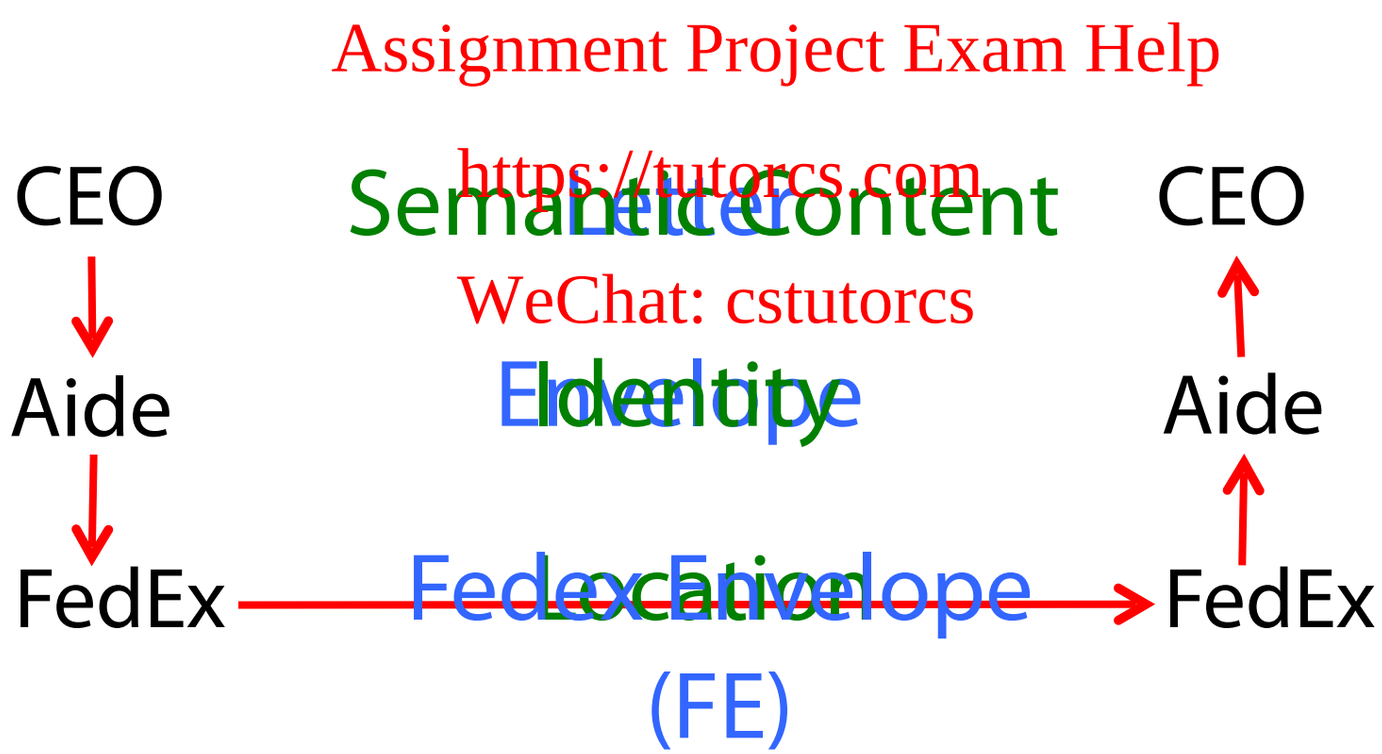
Your days are numbered.

--Steve

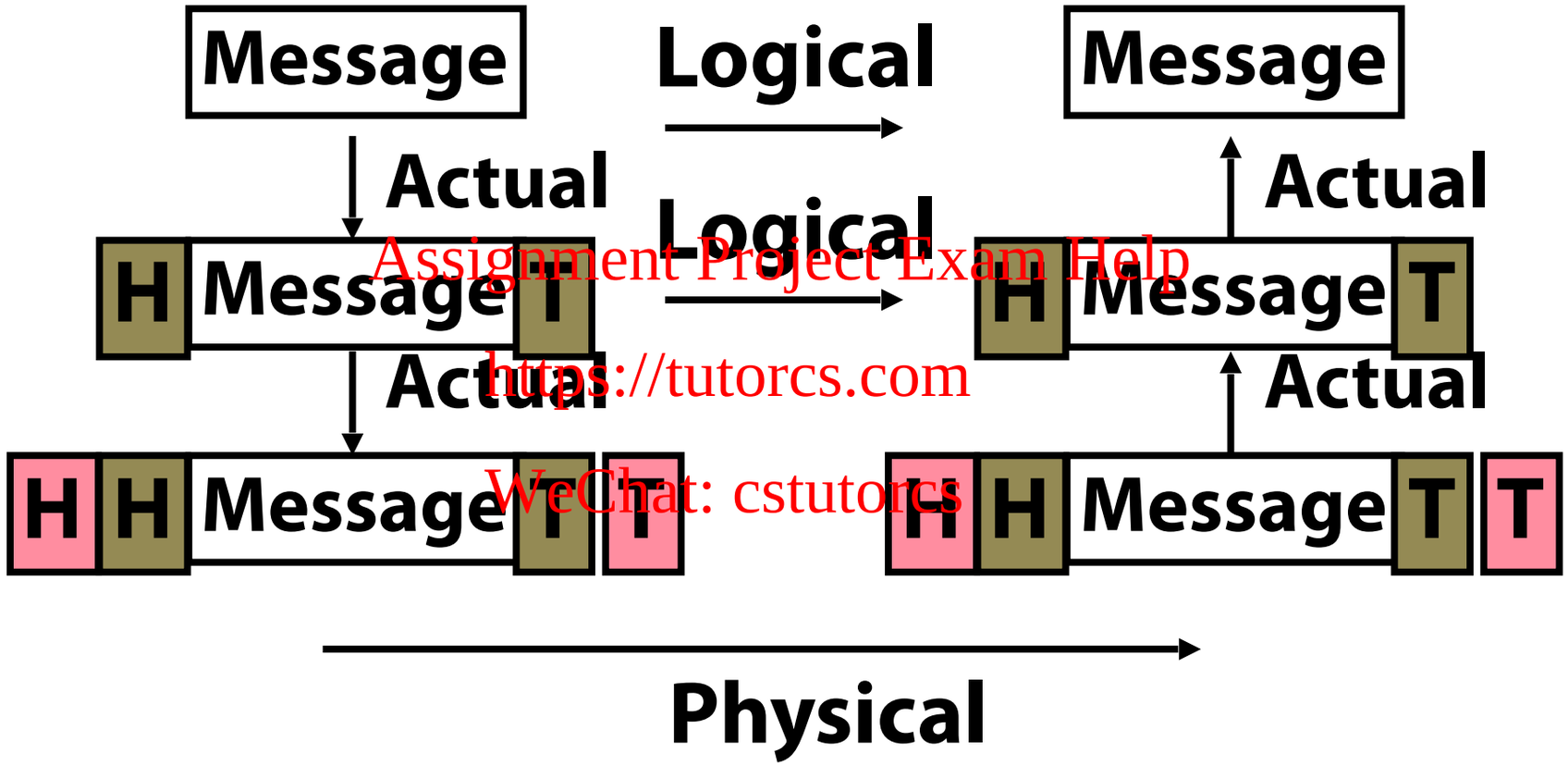
- CEO A writes letter to CEO B
  - Folds letter and hands it to assistant
- Assistant:
  - Puts letter in envelope with CEO B's full name
  - Takes to FedEx
- FedEx Office
  - Puts letter in larger envelope
  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck
- FedEx delivers to other company

The Path of the Letter

- "Peers" on each side understand the same things
- No one else needs to
- Lowest level has most packaging



Protocol Family Concept



Each lower level of stack "encapsulates" information from layer above by adding header and trailer

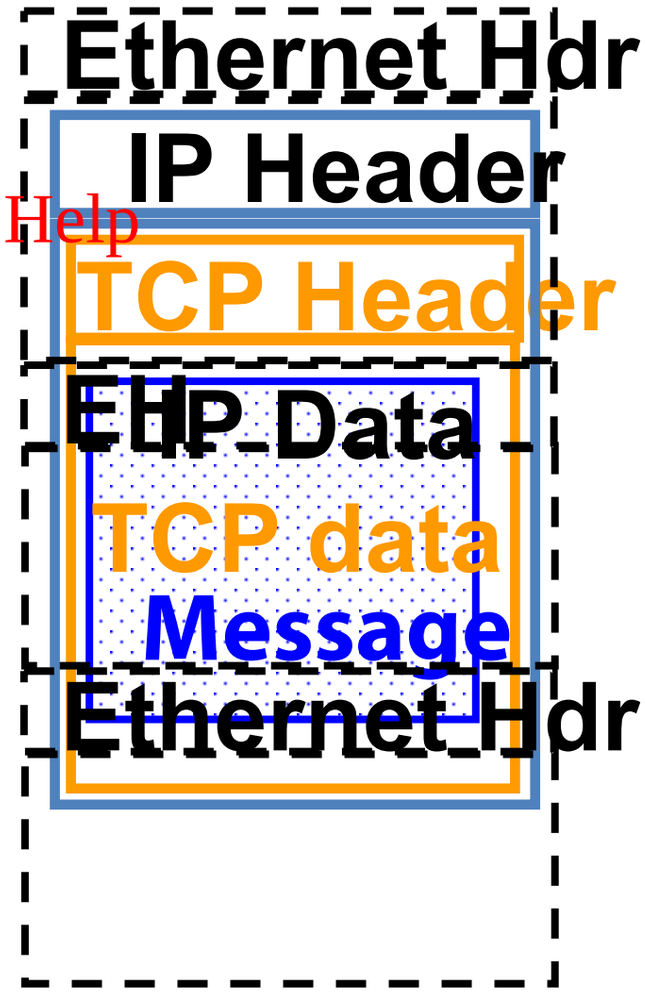
Most Popular Protocol for Network of Networks

- Transmission Control Protocol/Internet Protocol (TCP/IP)
- This protocol family is the basis of the Internet,
  a WAN (wide area network) protocol
  - IP makes best effort to deliver
  - Packets can be lost, corrupted
  - TCP guarantees delivery
  - TCP/IP so popular it is used even when communicating locally: even across homogeneous LAN (local area network)

TCP/IP Packet, Ethernet Packet, Protocols

- Application sends message

- TCP breaks into 64KiB segments, adds 20B header
- IP adds 20B header, sends to network
- If Ethernet, broken into 1500B packets with headers, trailers



"And, in Conclusion…"

- I/O gives computers their 5 senses
- I/O speed range is 100-million to one
- DMA to avoid wasting CPU time on data transfers
- Disks for persistent storage, being replaced by flash and emerging "storage class memory"
- Networks: computer-to-computer I/O
  - Protocol suites allow networking of heterogeneous components. Great Idea: Layers and Abstraction
  - Emerging class: Rack-scale/Storage-class Memory accessible over RDMA or other network interconnect