EE108B
Lecture 17
I/O Buses and Interfacing to CPU

Christos Kozyrakis
Stanford University
http://eeclass.stanford.edu/ee108b

Announcements

- Remaining deliverables
  - PA2.2. today
  - HW4 on 3/13
  - Lab4 on 3/19
- In class Quiz 2 on Thu 2/15 (11am – 12.30pm)
  - Closed-books, 1 page of notes, green page, calculator
  - All lectures included
  - SCPD students come to class for exam
- Advice
  - Catch up with lectures and textbook
  - Take advantage of office hours and discussion sessions
- Last review session on Friday

I/O Review

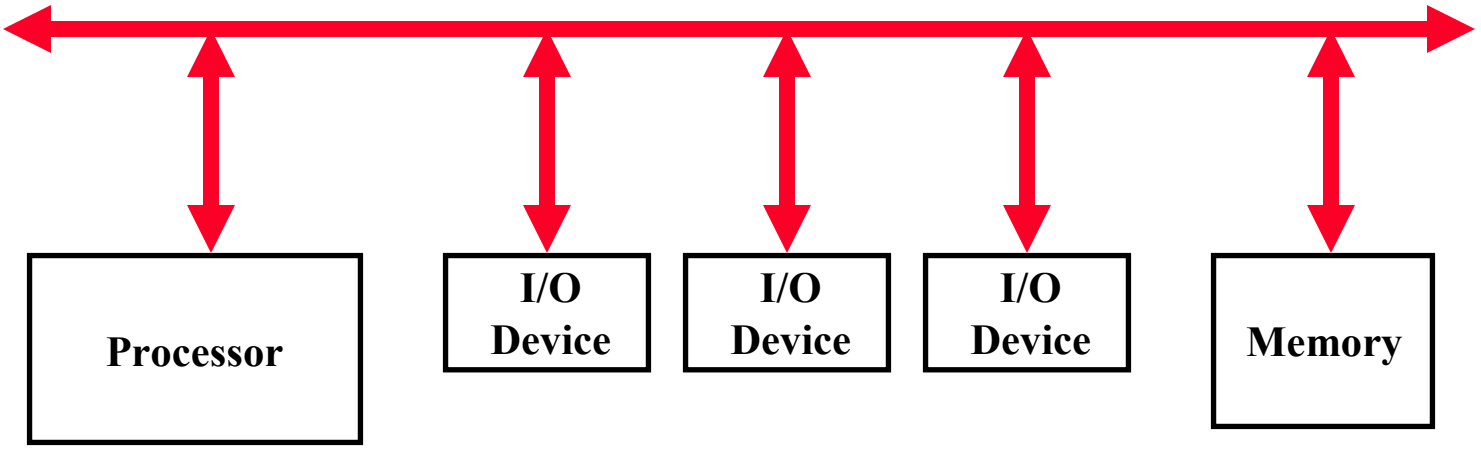
- I/O Performance is dependent upon many factors
  - Device performance, which is usually technology driven
  - CPU and memory system performance
  - Software efficiency (OS)
  - Workload being measured
- Consider the example of disk performance
  - Three major types of mechanical delays
    - Seek time – Time to move head over correct track
    - Rotational delay – Time for sector to rotate under head
    - Transfer time – Time to read data from disk
  - Transaction processing
    - Lots of concurrent requests
    - Instruction behavior – deep pipelines don't work well
    - Memory access – random access to large data
    - Disk access – small files without locality

Today's Lecture

- Buses
- Interfacing I/O with processor and memory
- Read Sections 8.4–8.9

Buses

- A bus is a shared communication link that connects multiple devices
- Single set of wires connects multiple “subsystems” as opposed to a point to point link which only connects two components together
- Wires connect in parallel, so 32 bit bus has 32 wires of data



Advantages/Disadvantages

- Advantages
  - Broadcast capability of shared communication link
  - Versatility
    - New device can be added easily
    - Peripherals can be moved between computer systems that use the same bus standard
  - Low Cost
    - A single set of wires is shared multiple ways
- Disadvantages
  - Communication bottleneck
    - Bandwidth of bus can limit the maximum I/O throughput
  - Limited maximum bus speed
    - Length of the bus
    - Number of devices on the bus
    - Need to support a range of devices with varying latencies and transfer rates

Bus Organization

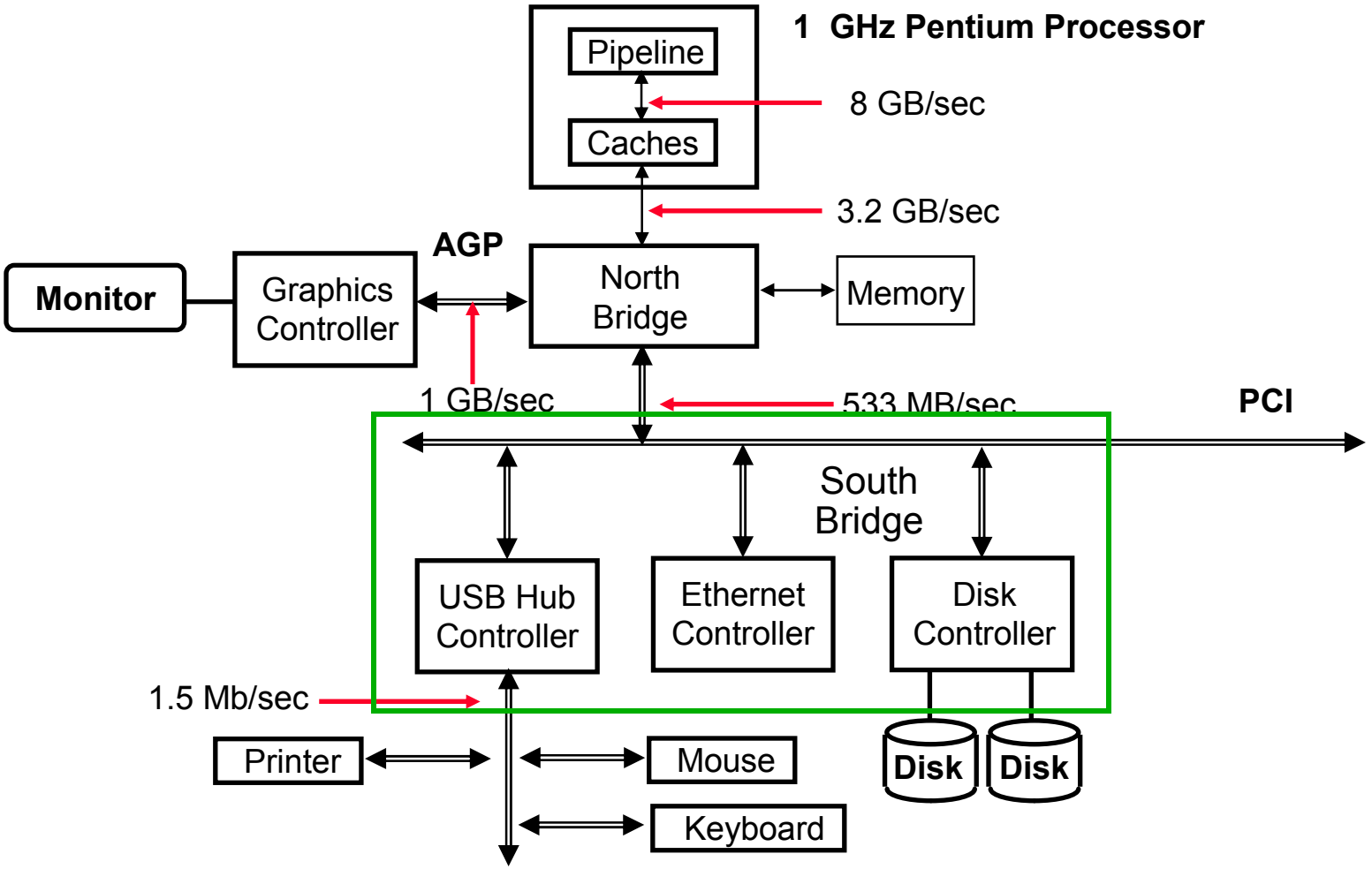
- Bus Components
  - Control Lines
    - Signal begin and end of transactions
    - Indicate the type of information on the data line
  - Data Lines
    - Carry information between source and destination
    - Can include data, addresses, or complex commands

Types of Buses

- Processor-Memory Bus (or *front-side* bus or *system* bus)
  - Short, high-speed bus
  - Connects memory and processor directly
  - Designed to match the memory system and achieve the maximum memory-to-processor bandwidth (cache transfers)
  - Designed specifically for a given processor/memory system

- I/O Bus (or peripheral bus)
  - Usually long and slow
  - Connect devices to the processor-memory bus
  - Must match a wide range of I/O device performance characteristics
  - Industry standard

Speeds and Feeds of a PC System

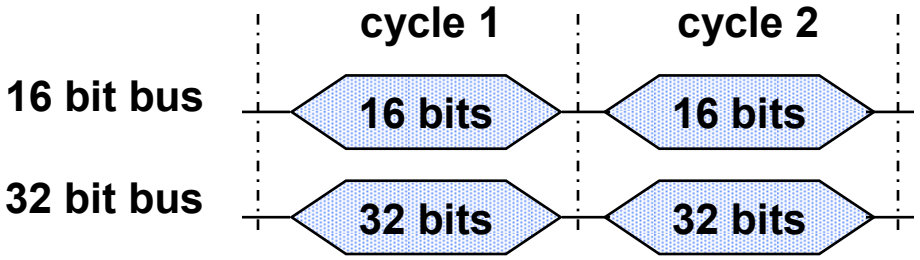


Synchronous versus Asynchronous

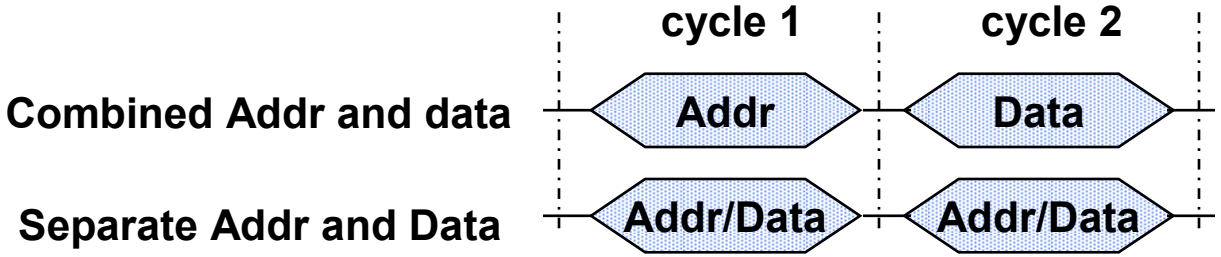
- Synchronous Bus
  - Includes a clock in control lines
  - Fixed protocol for communication relative to the clock
  - Advantages
    - Involves very little logic and can therefore run very fast
  - Disadvantages
    - Every decision on the bus must run at the same clock rate
    - To avoid clock skew, bus cannot be long if it is fast
  - Example: Processor-Memory Bus
- Asynchronous Bus
  - No clock
  - Can easily accommodate a wide range of devices
  - No clock skew problems, so bus can be quite long
  - Requires handshaking protocol

Increasing Bus Bandwidth

- Several factors account for bus bandwidth
  - Wider bus width
    - Increasing data bus width => more data per bus cycle
    - Cost: More bus lines

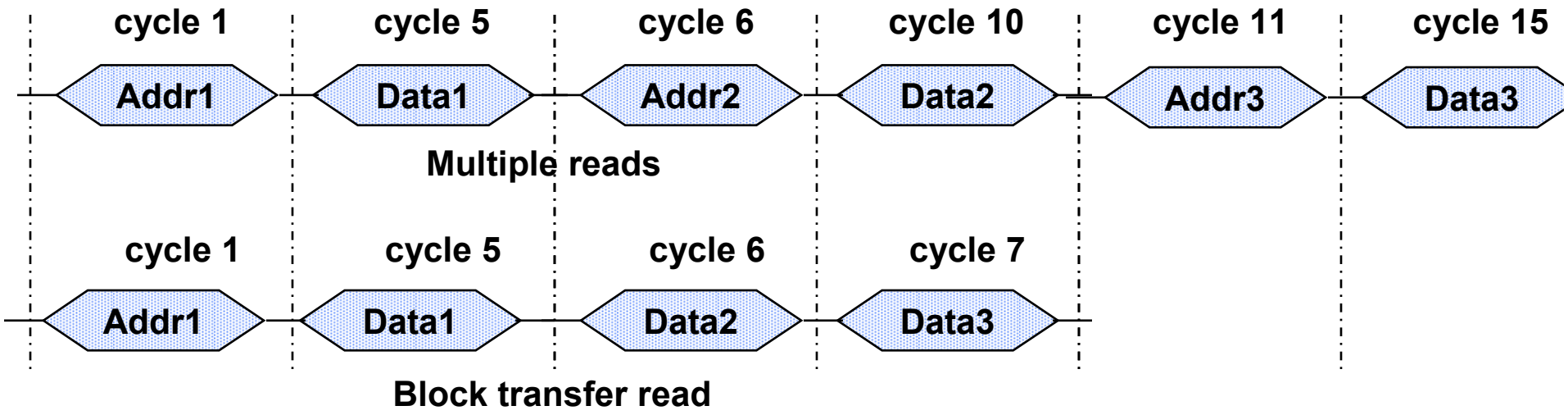


  - Separate address and data lines
    - Address and data can be transmitted in one bus cycle if separate address and data lines are available
    - Costs: More bus lines



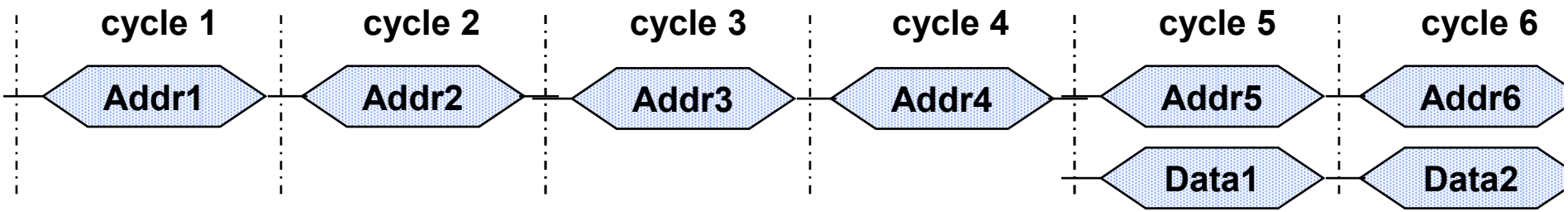
Increasing Bus Bandwidth

- Several factors account for bus bandwidth
  - Block transfers
    - Transfer multiple words in back-to-back bus cycles
    - Only one address needs to be sent at the start
    - Bus is not released until the last word is transferred
    - Costs: Increased complexity and increased response time for pending requests



Increasing Bus Bandwidth

– Split transaction "pipelining the bus"
  - Free the bus during time between request and data transfer
  - Costs: Increased complexity and higher potential latency



Split transaction bus with separate Address and Data wires

Accessing the Bus

- Up to this point we have not addressed one of the most important questions in bus design: How is the bus reserved by a device that wishes to use it?
- Master-slave arrangement
  - Only the bus master can control access to the bus
  - The bus master initiates and controls all bus requests
  - A slave responds to read and write requests
- A simple system
  - Processor is the only bus master
  - All bus requests must be controlled by the processor
  - Major drawback is the processor must therefore be involved in every transaction!

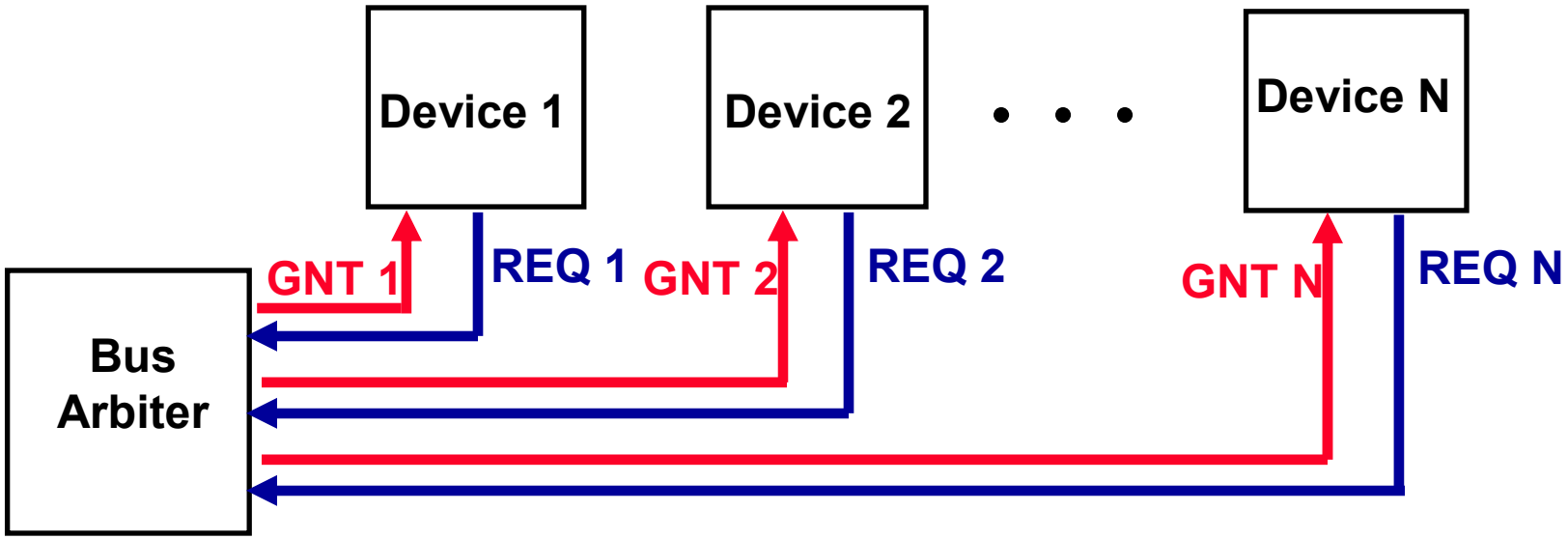
Multiple Masters

- With multiple masters, arbitration must be used so that only one device is granted access to the bus at a given time

- Arbitration
  - The bus master wanting to use the bus asserts a bus request
  - The bus master cannot use the bus until the request is granted
  - The bus master must signal the arbiter when finished using the bus

- Bus arbitration goals
  - Bus priority – Highest priority device should be serviced first
  - Fairness – Lowest priority devices should not starve

Centralized Parallel Arbitration

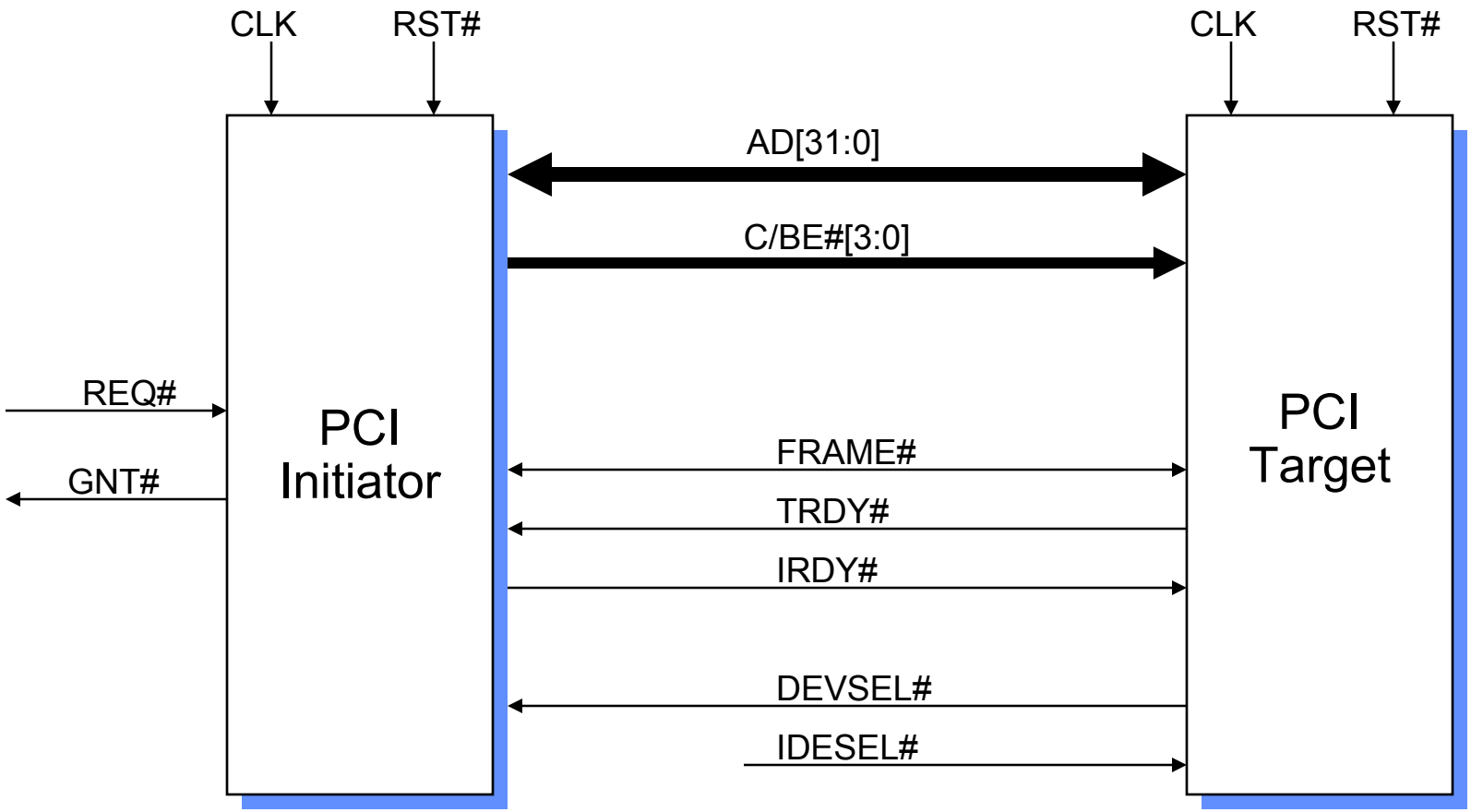


- Advantages
  - Centralized control where all devices submit request
  - Any fair priority scheme can be implemented (FCFS, round-robin)
- Disadvantages
  - Potential bottleneck at central arbiter

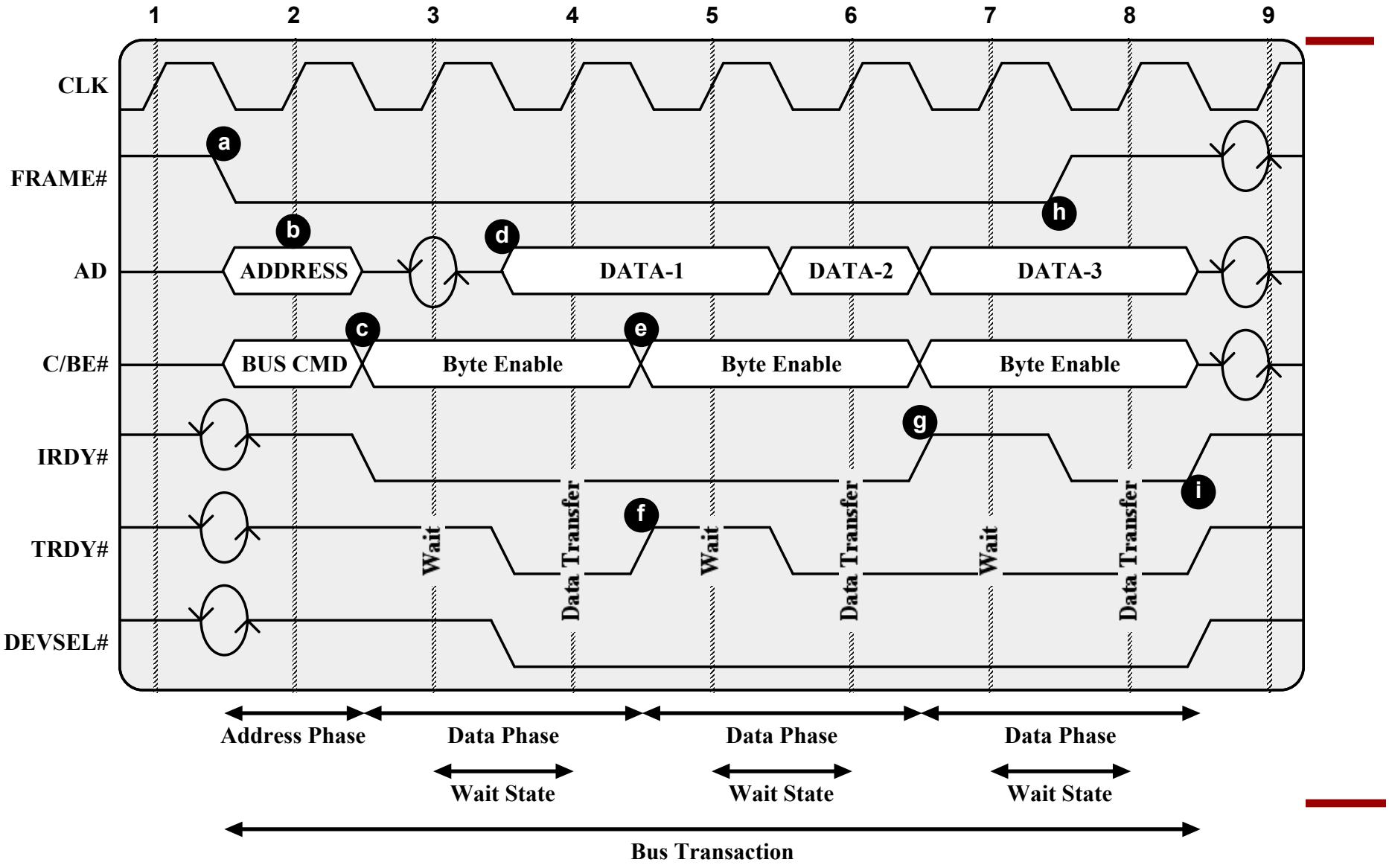
Case Study: PCI

- Peripheral Component Interconnect (PCI) peripheral backplane bus standard
- Clock Rate: 33 MHz (or 66 MHz in PCI Version 2.1) [CLK]
- Central arbitration [REQ#, GNT#]
  - Overlapped with previous transaction
- Multiplexed Address/Data
  - 32 lines (with extension to 64) [AD]
- General Protocol
  - Transaction type (bus command is memory read, memory write, memory read line, etc) [C/BE#]
  - Address handshake and duration [FRAME#, TRDY#]
  - Data width (byte enable) [C/BE#]
  - Variable length data block handshake between Initiatory Ready and Target Ready [IRDY#, TRDY#]
- Maximum bandwidth is 132 MB/s (533 MB/s at 64 bit/ 66 MHz)

32 bit PCI Signals



PCI Read


1
2
3
4
5
6
7
8
9
CLK
FRAME#
AD
C/BE#
IRDY#
TRDY#
DEVSEL#
a
b
c
d
e
f
g
h
i
ADDRESS
DATA-1
DATA-2
DATA-3
BUS CMD
Byte Enable
Byte Enable
Byte Enable
Wait
Data Transfer
Wait
Data Transfer
Wait
Data Transfer
Address Phase
Data Phase
Data Phase
Data Phase
Wait State
Wait State
Wait State
Bus Transaction

PCI Read Steps 1

a) Once a bus master has gained control of the bus, it initiates the transaction by asserting FRAME. This line remains asserted until the last data phase. The initiator also puts the start address on the address bus, and the read command on the C/BE lines.

b) The target device recognizes its address on the AD lines.

c) The initiator ceases driving the AD bus. A turnaround cycle (marked with two circular arrows) is required before another device may drive any multiple-source bus. Meanwhile, the initiator changes the C/BE lines to designate which AD lines are to be used for data transfer (from 1-4 bytes wide). The initiator also asserts IRDY to indicate that it is ready for the first data item.

d) The selected target asserts DEVSEL to indicate that it has recognized its address and will respond. It places the requested data on the AD lines and asserts TRDY to indicate that valid data is present on the bus.

PCI Read Steps 2

e) The initiator reads the data at the beginning of clock 4 and changes the byte enable lines as needed in preparation for the next read.

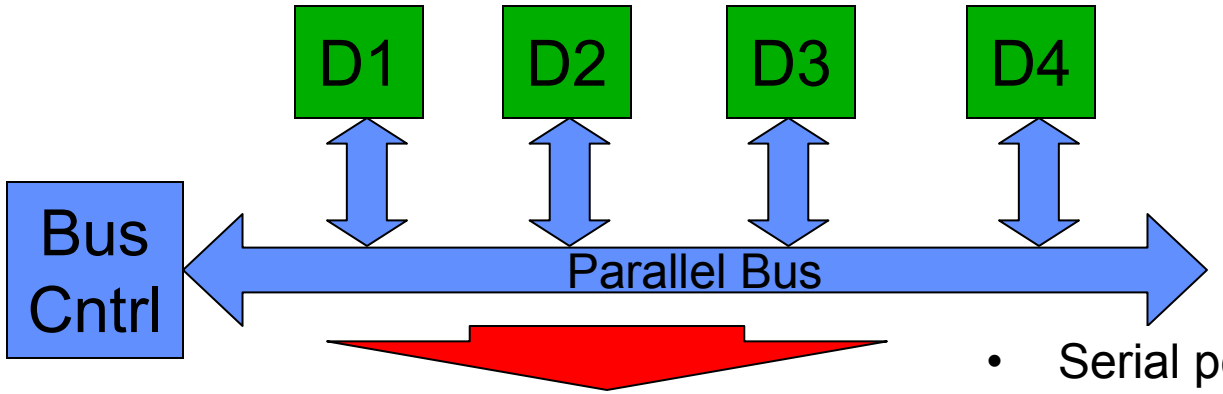
f) In this example, the target needs some time to prepare the second block of data for transmission. Therefore, it deasserts TRDY to signal the initiator that there will not be new data during the coming cycle. Accordingly, the initiator does not read the data lines at the beginning of cycle 5 and does not change the byte enable on that cycle. The block of data is read at the beginning of cycle 6.

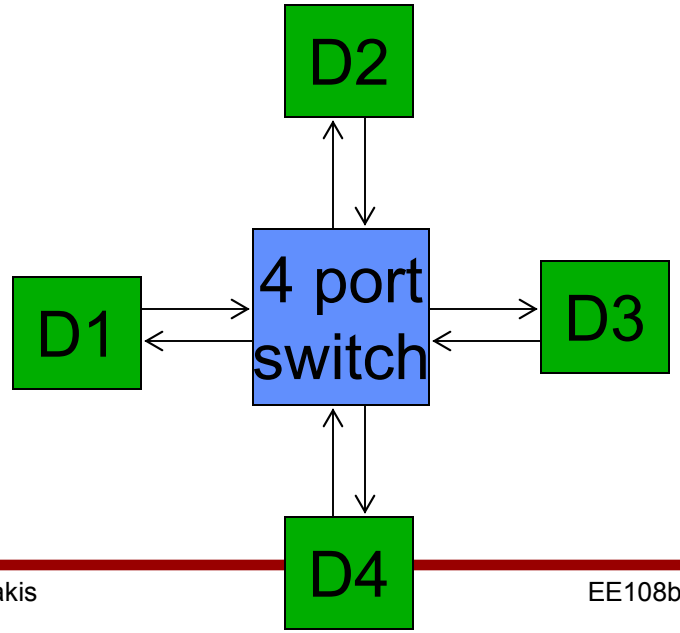
g) During clock 6, the target places the third data item on the bus. However, in this example the initiator is not yet ready to read the data item (i.e. temporarily buffers are full). It therefore deasserts IRDY. This will cause the target to hold the data for an extra cycle.

h) The initiator deasserts FRAME to signal the target that the third data transfer is the last, and asserts IRDY to signal that it is ready.

i) Return to the idle state. The initiator deasserts IRDY, and the target deasserts TRDY & DEVSEL.

Trends for Buses
Logical Bus and Physical Switch





- Serial point-to-point advantages
  - Faster links
  - Fewer chip package pins
  - Higher performance
  - Switch keeps arbitration on chip

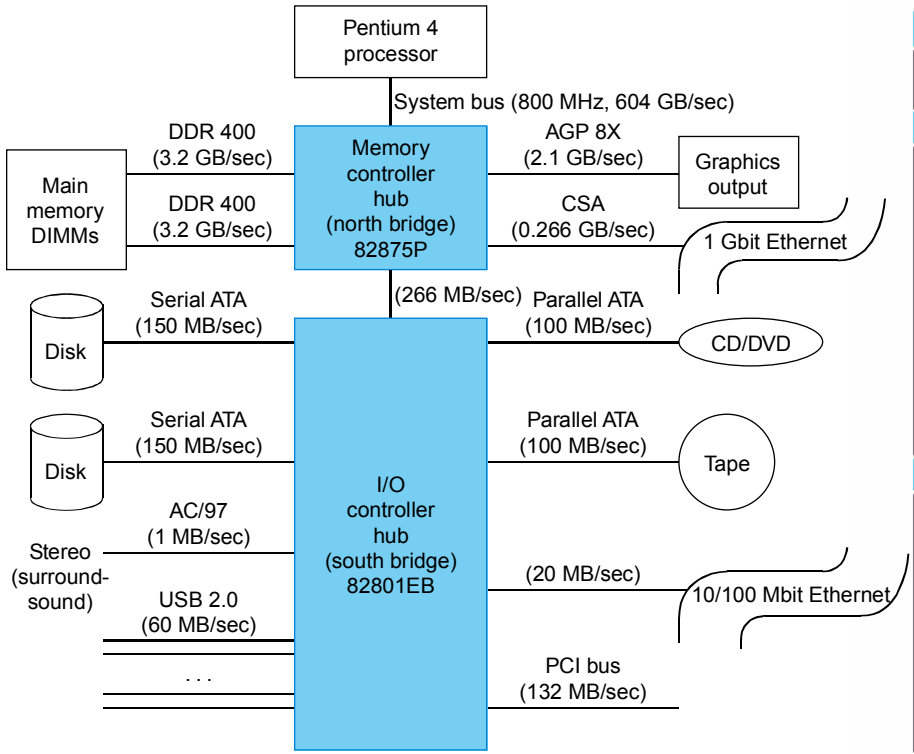
- Many bus standards are moving to serial, point to point
- 3GIO, PCI-Express(PCI)
- Serial ATA (IDE hard disk)
- AMD Hypertransport versus Intel Front Side Bus (FSB)

PCI vs. PCI Express

- Same bus protocol
  - Same driver software
- PCI
  - 32–64 shared wires
  - Frequency: 33MHz – 133 MHz
  - Bandwidth: 132 MB/s – 1 GB/s
- PCI Express
  - 4 wires per direction
  - Frequency: 625 MHz
  - Bandwidth: 300 MB/s per direction
- PCI Express Advantage
  - 5-10 x pin bandwidth
  - Multiple links for more bandwidth

Modern Pentium 4 I/O

- I/O Options



	875P chip set	845GL chip set
Target segment	Performance PC	Value PC
System bus (64 bit)	800/533 MHz	400 MHz
Memory controller hub ("north bridge")		
Package size, pins	42.5 × 42.5 mm, 1005	37.5 × 37.5 mm, 760
Memory speed	DDR 400/333/266 SDRAM	DDR 266/200, PC133 SDRAM
Memory buses, widths	2 × 72	1 × 64
Number of DIMMs, DRAM Mbit support	4, 128/256/512 Mbits	2, 128/256/512 MBits
Maximum memory capacity	4 GB	2 GB
Memory error correction available?	yes	no
AGP graphics bus, speed	yes, 8X or 4X	no
Graphics controller	external	Internal (Extreme Graphics)
CSA Gigabit Ethernet interface	yes	no
South bridge interface speed (8 bit)	266 MHz	266 MHz
I/O controller hub ("south bridge")		
Package size, pins	31 × 31 mm, 460	31 × 31 mm, 421
PCI bus: width, speed, masters	32-bit, 33 MHz, 6 masters	32-bit, 33 MHz, 6 masters
Ethernet MAC controller, interface	100/10 Mbit	100/10 Mbit
USB 2.0 ports, controllers	8, 4	6, 3
ATA 100 ports	2	2
Serial ATA 150 controller, ports	yes, 2	no
RAID 0 controller	yes	no
AC-97 audio controller, interface	yes	yes
I/O management	SMbus 2.0, GPIO	SMbus 2.0, GPIO

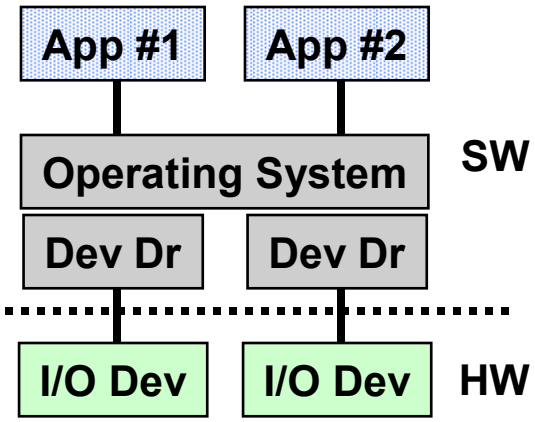
FIGURE 8.12 Two Pentium 4 I/O chip sets from Intel. The 845GL north bridge uses many fewer pins than the 875 by having just one memory bus and by omitting the AGP bus and the Gigabit Ethernet interface. Note that the serial nature of USB and Serial ATA means that two more USB ports and two more Serial ATA ports need just 39 more pins in the south bridge of the 875 versus the 845GL chip sets.

Review: Bus Summary

- Bus design issues
  - Bus width
  - Synchronization
  - Arbitration
  - Bus transactions
    - Read/write protocols
    - Block addressing
    - Split transactions
- Three basic buses
  - Processor-memory: Front-side bus
  - Backplane bus: PCI
  - I/O bus: USB
- Bus design trends
  - Point-to-point serial connections with switches

Operating System Tasks

- The OS is a resource manager and acts as the interface between I/O hardware and programs that request I/O



- Several important characteristics of the I/O system:
  - I/O system is *shared* by multiple programs
  - I/O systems often use *interrupts* to notify CPU
    - Interrupts = externally generated “exceptions”
    - Typically “Input available” or “Output complete” messages
    - OS handles interrupts by transferring control to kernel mode
  - Low-level control of an I/O device is complex
    - Managing a set of concurrent events
    - Requirements for correct device control are very detailed
    - I/O device drivers are the most common area for bugs in an OS!

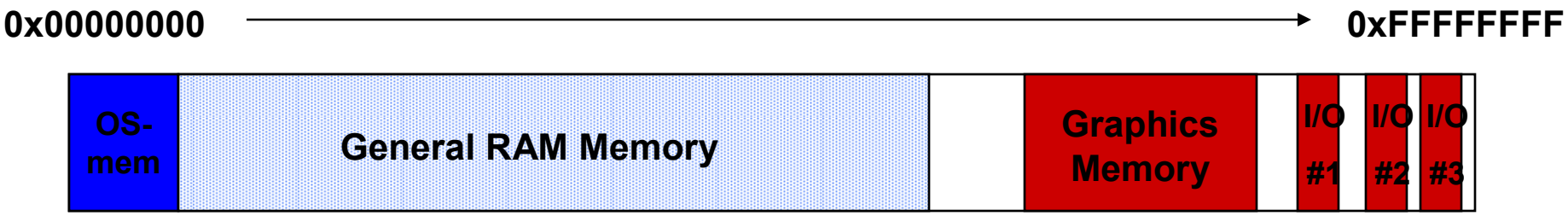
OS Communication

- The operating system should prevent user programs from communicating with I/O device directly
  - Must protect I/O resources to keep sharing fair
  - Protection of shared I/O resources cannot be provided if user programs could perform I/O directly

- Three types of communication are required:
  - OS must be able to give commands to I/O devices
  - I/O device must be able to notify OS when I/O device has completed an operation or has encountered an error
  - Data must be transferred between memory and an I/O device

I/O Commands:
A method for addressing a device

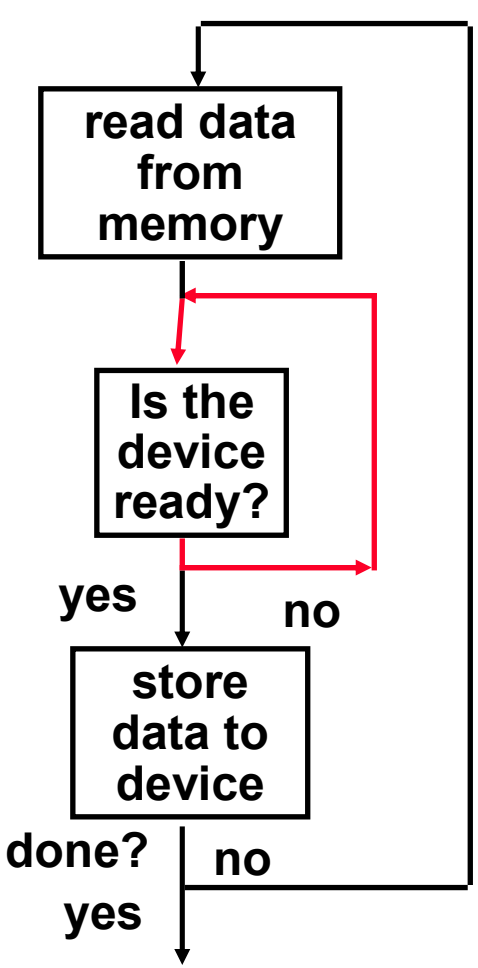
- Memory-mapped I/O:
  - Portions of the address space are assigned to each I/O device
  - I/O addresses correspond to device registers
  - User programs prevented form issuing I/O operations directly since I/O address space is *protected* by the address translation mechanism



Communicating with the CPU

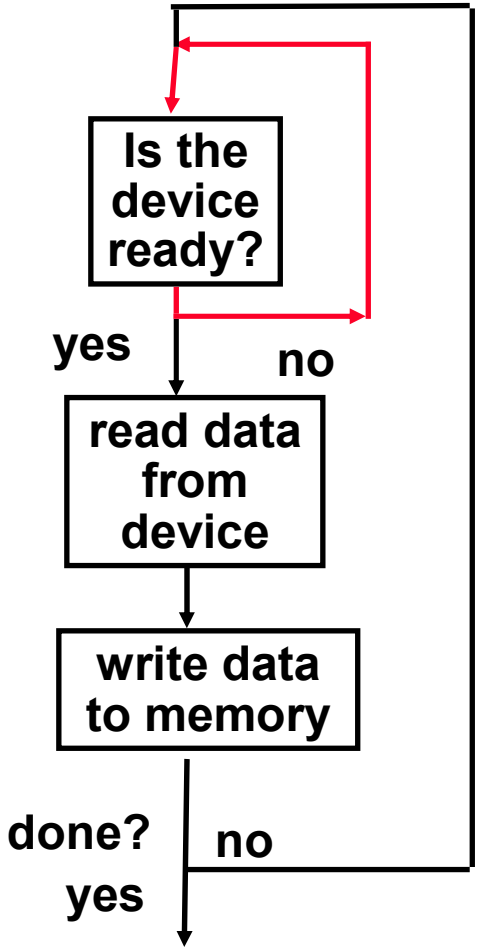
- Method #1: Polling
  - I/O device places information in a status register
  - The OS periodically checks the status register
  - Whether polling is used is often dependent upon whether the device can initiate I/O independently
    - For instance, a mouse works well since it has a fixed I/O rate and initiates its own data (whenever it is moved)
    - For others, such as disk access, I/O only occurs under the control of the OS, so we poll only when the OS knows it is active
  - Advantages
    - Simple to implement
    - Processor is in control and does the work
  - Disadvantage
    - Polling overhead and data transfer consume CPU time

Polling and Programmed I/O



Busy wait loop not an efficient way to use the CPU unless the device is very fast!

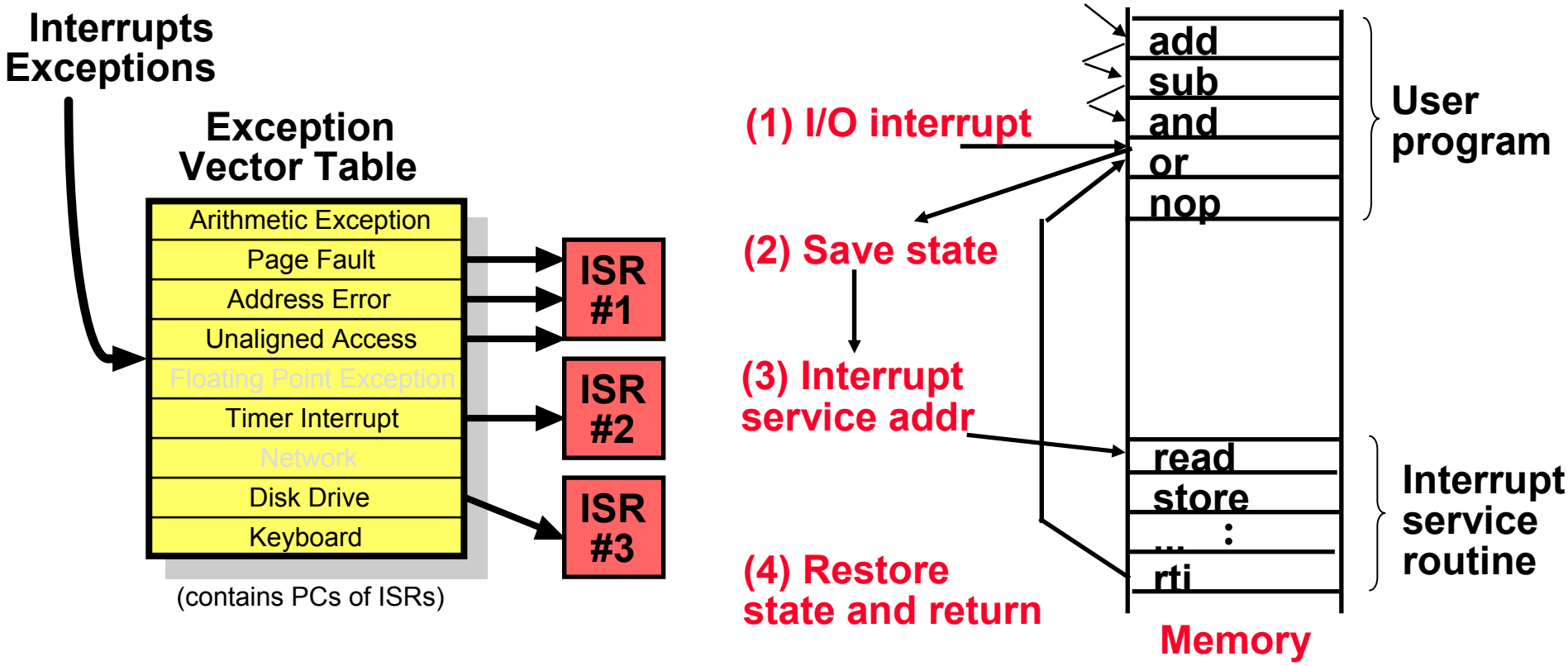
Processor may be inefficient way to transfer data



I/O Notification (cont)

- Method #2: I/O Interrupt
  - Whenever an I/O device needs attention from the processor, it *interrupts* the processor
  - Interrupt must tell OS about the event and which device
    - Using “cause” register(s): Kernel “asks” what interrupted
    - Using *vectored* interrupts: A different exception handler for each
    - Example: Intel 80x86 has 256 vectored interrupts
  - I/O interrupts are asynchronous events, and happen anytime
    - Processor waits until current instruction is completed
  - Interrupts may have different priorities
    - Ex.: Network = high priority, keyboard = low priority
  - Advantage: Execution is only halted during actual transfer
  - Disadvantage: Software overhead of interrupt processing

I/O Interrupts



Data Transfer

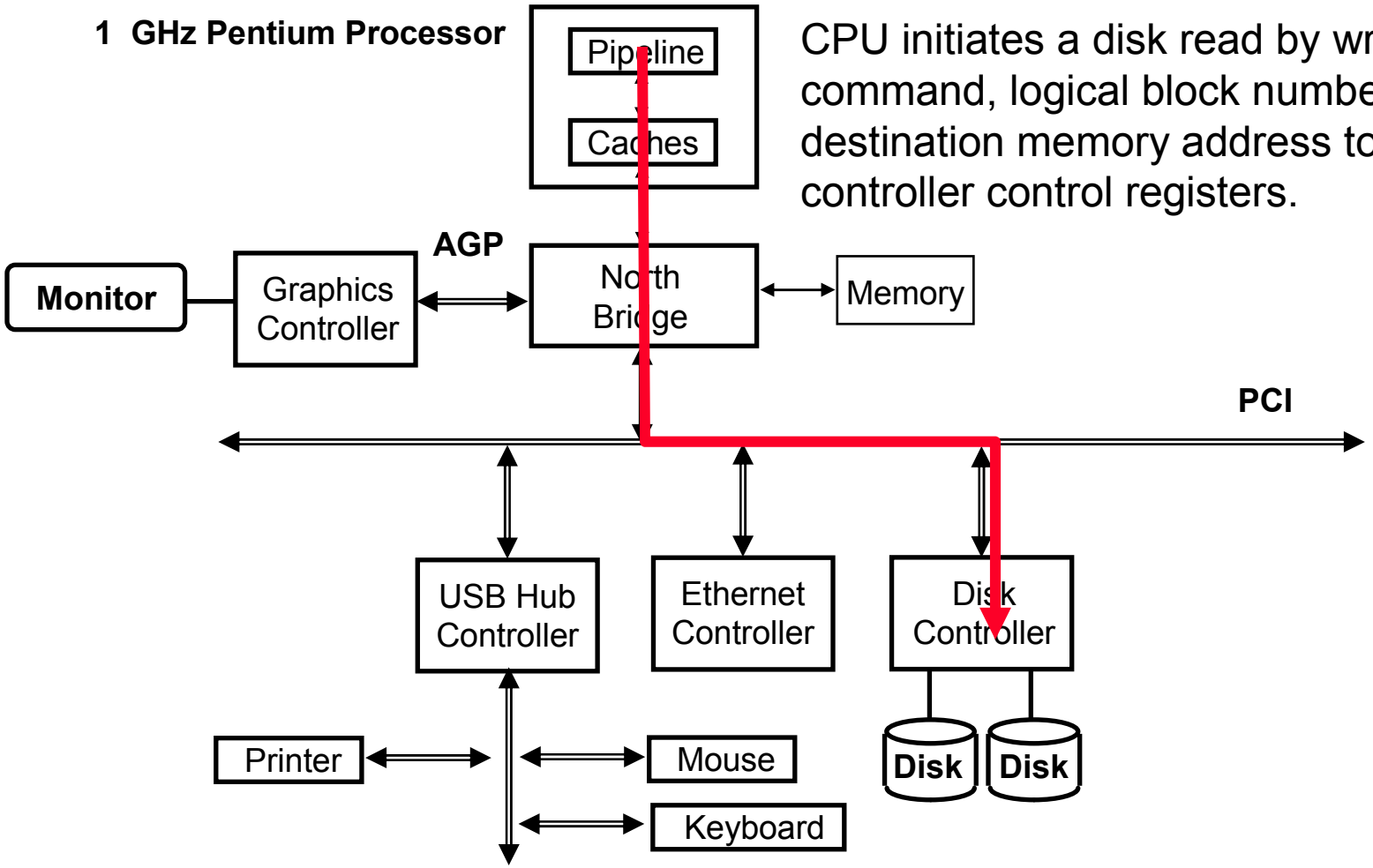
- The third component to I/O communication is the transfer of data from the I/O device to memory (or vice versa)
- Simple approach: "Programmed" I/O
  - Software on the processor moves *all* data between memory addresses and I/O addresses
  - Simple and flexible, but wastes CPU time
  - Also, lots of excess data movement in modern systems
    - Ex.: Mem --> NB --> CPU --> NB --> graphics
    - When we want: Mem --> NB --> graphics
- So need a solution to allow data transfer to happen *without* the processor's involvement

Delegating I/O: DMA

- Direct Memory Access (DMA)
  - Transfer *blocks* of data to or from memory without CPU intervention
  - Communication coordinated by the *DMA controller*
    - DMA controllers are integrated in memory or I/O controller chips
  - DMA controller acts as a bus master, in bus-based systems

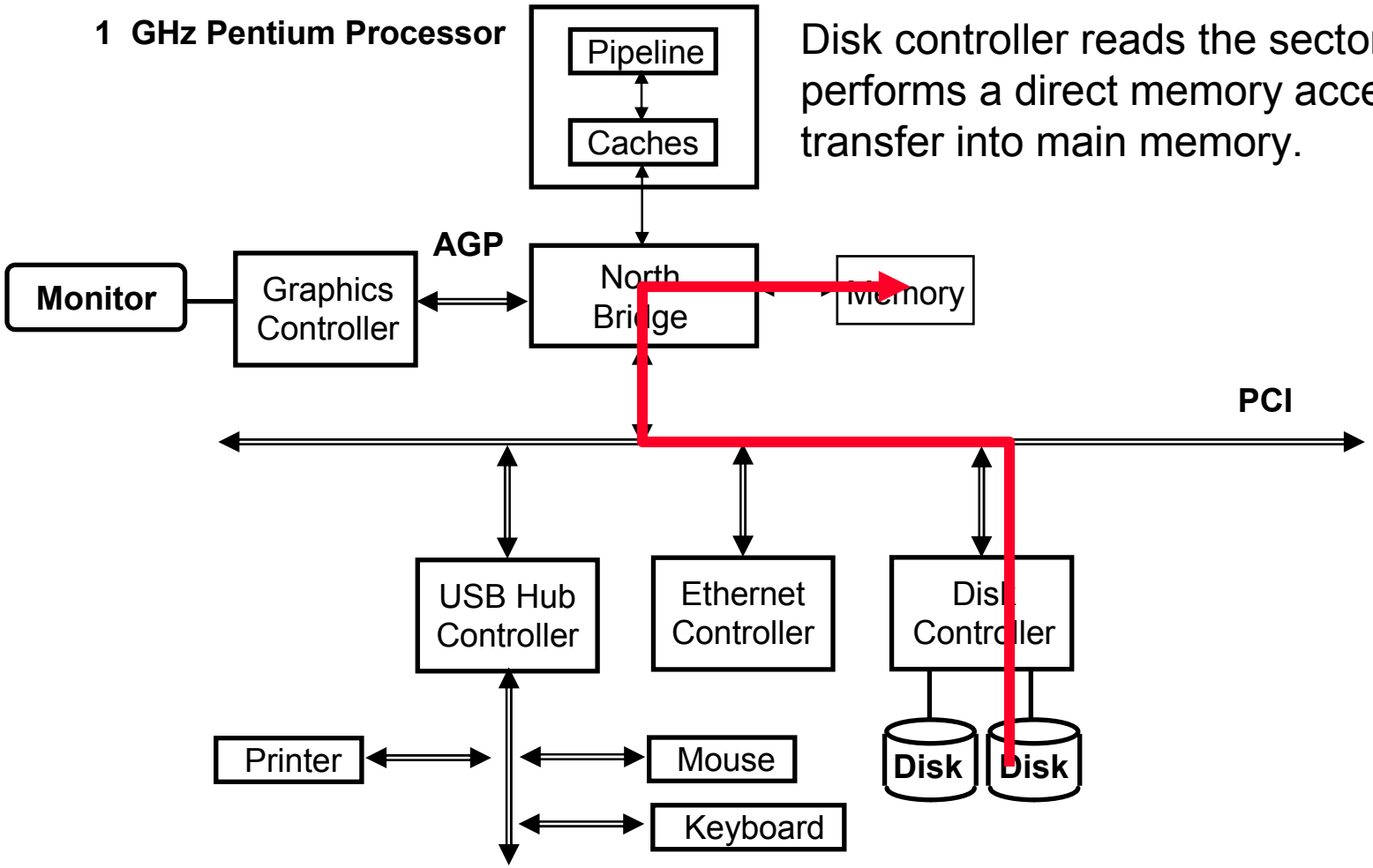
- DMA Steps
  - Processor sets up DMA by supplying:
    - Identity of the device and the operation (read/write)
    - The memory address for source/destination
    - The number of bytes to transfer
  - DMA controller starts the operation by arbitrating for the bus and then starting the transfer when the data is ready
  - Notify the processor when the DMA transfer is complete or on error
    - Usually using an interrupt

Reading a Disk Sector (1)



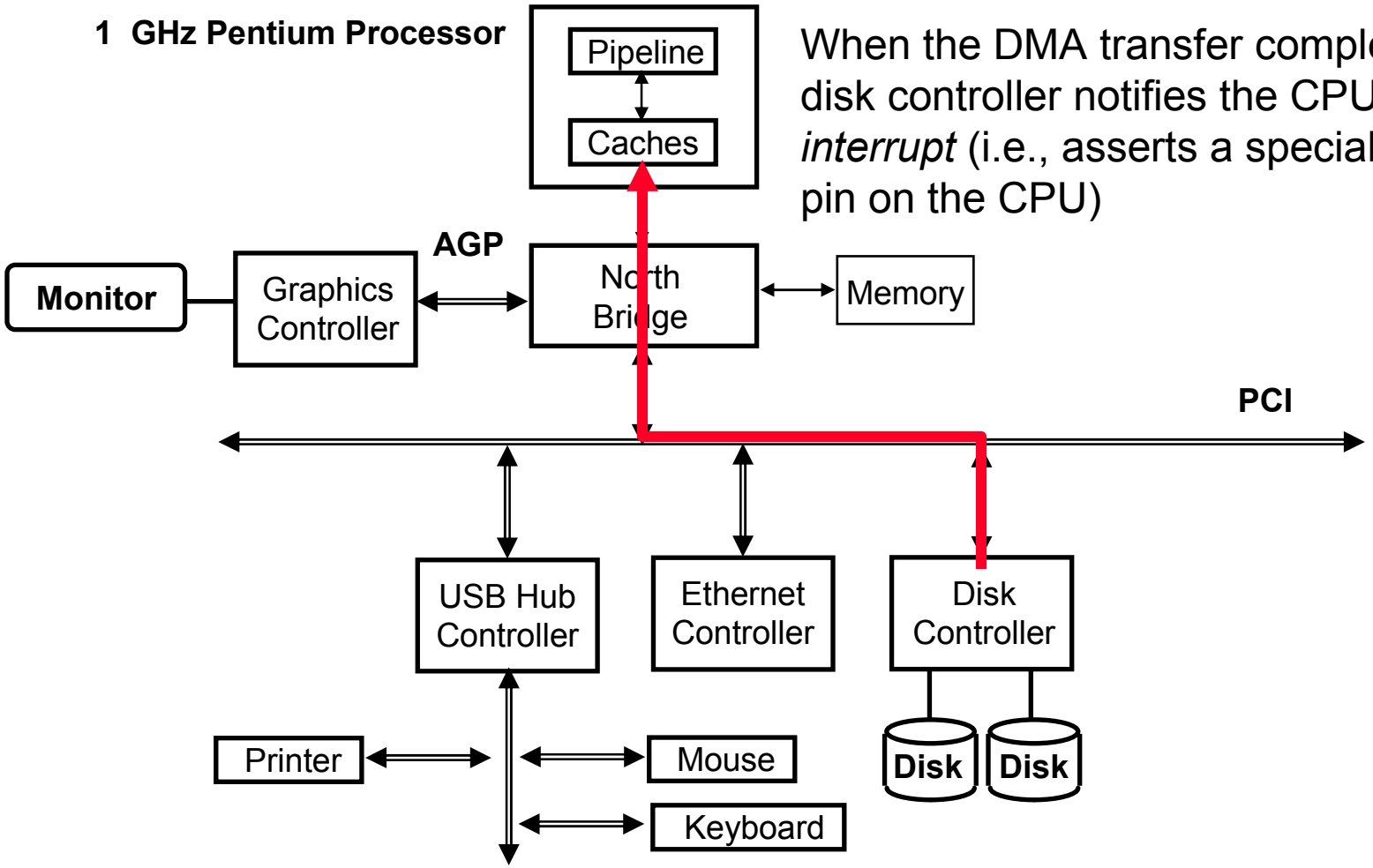
CPU initiates a disk read by writing a command, logical block number, and a destination memory address to disk controller control registers.

Reading a Disk Sector (2)



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Reading a Disk Sector (3)



When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)

DMA Problems: Virtual Vs. Physical Addresses

- If DMA uses physical addresses
  - Memory access across physical page boundaries may not correspond to contiguous virtual pages (or even the same application!)
- Solution 1: $\leq$ 1 page per DMA transfer
- Solution 1+: chain a series of 1-page requests provided by the OS
  - Single interrupt at the end of the last DMA request in the chain
- Solution 2: DMA engine uses virtual addresses
  - Multi-page DMA requests are now easy
  - A TLB is necessary for the DMA engine

- For DMA with physical addresses: pages must be pinned in DRAM
  - OS should not page to disks pages involved with pending I/O

DMA Problems: Cache Coherence

- A copy of the data involved in a DMA transfer may reside in processor cache
  - If memory is updated: Must update or invalidate "old" cached copy
  - If memory is read: Must read latest value, which may be in the cache
    - Only a problem with write-back caches
- This is called the "cache coherence" problem
  - Same problem in multiprocessor systems

- Solution 1: OS flushes the cache before I/O reads or forces writebacks before I/O writes
  - Flush/write-back may involve selective addresses or whole cache
  - Can be done in software or with hardware (ISA) support
- Solution 2: Route memory accesses for I/O through the cache
  - Search the cache for copies and invalidate or write-back as needed
  - This hardware solution may impact performance negatively
    - While searching cache for I/O requests, it is not available to processor
  - Multi-level, inclusive caches make this easier
    - Processor searches L1 cache mostly (until it misses)
    - I/O requests search L2 cache mostly (until it finds a copy of interest)

I/O Summary

- I/O performance has to take into account many variables
  - Response time and throughput
  - CPU, memory, bus, I/O device
  - Workload e.g. transaction processing
- I/O devices also span a wide spectrum
  - Disks, graphics, and networks
- Buses
  - bandwidth
  - synchronization
  - transactions
- OS and I/O
  - Communication: Polling and Interrupts
  - Handling I/O outside CPU: DMA