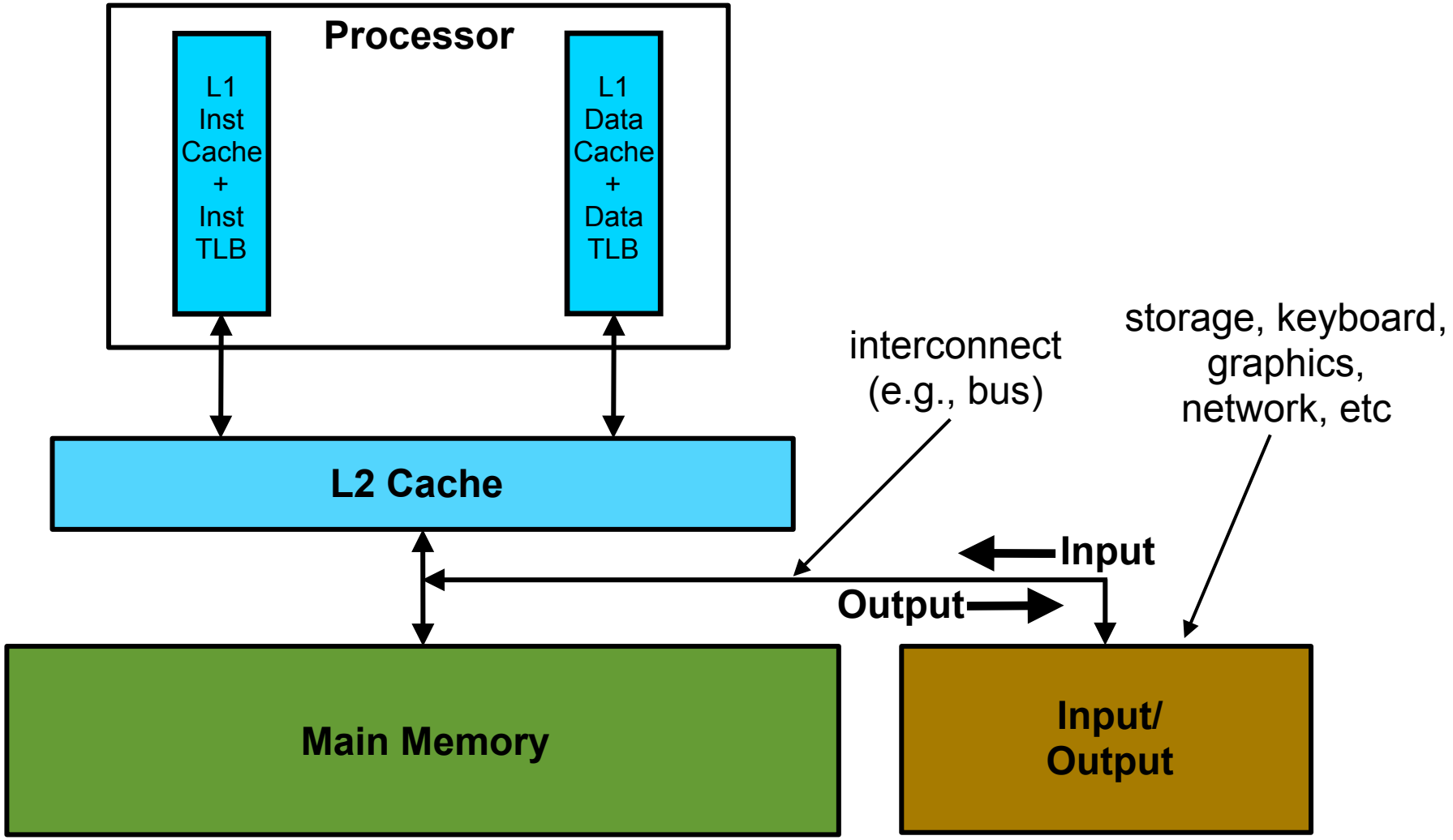
# ECE 2300
# Digital Logic & Computer Organization

## Fall 2016

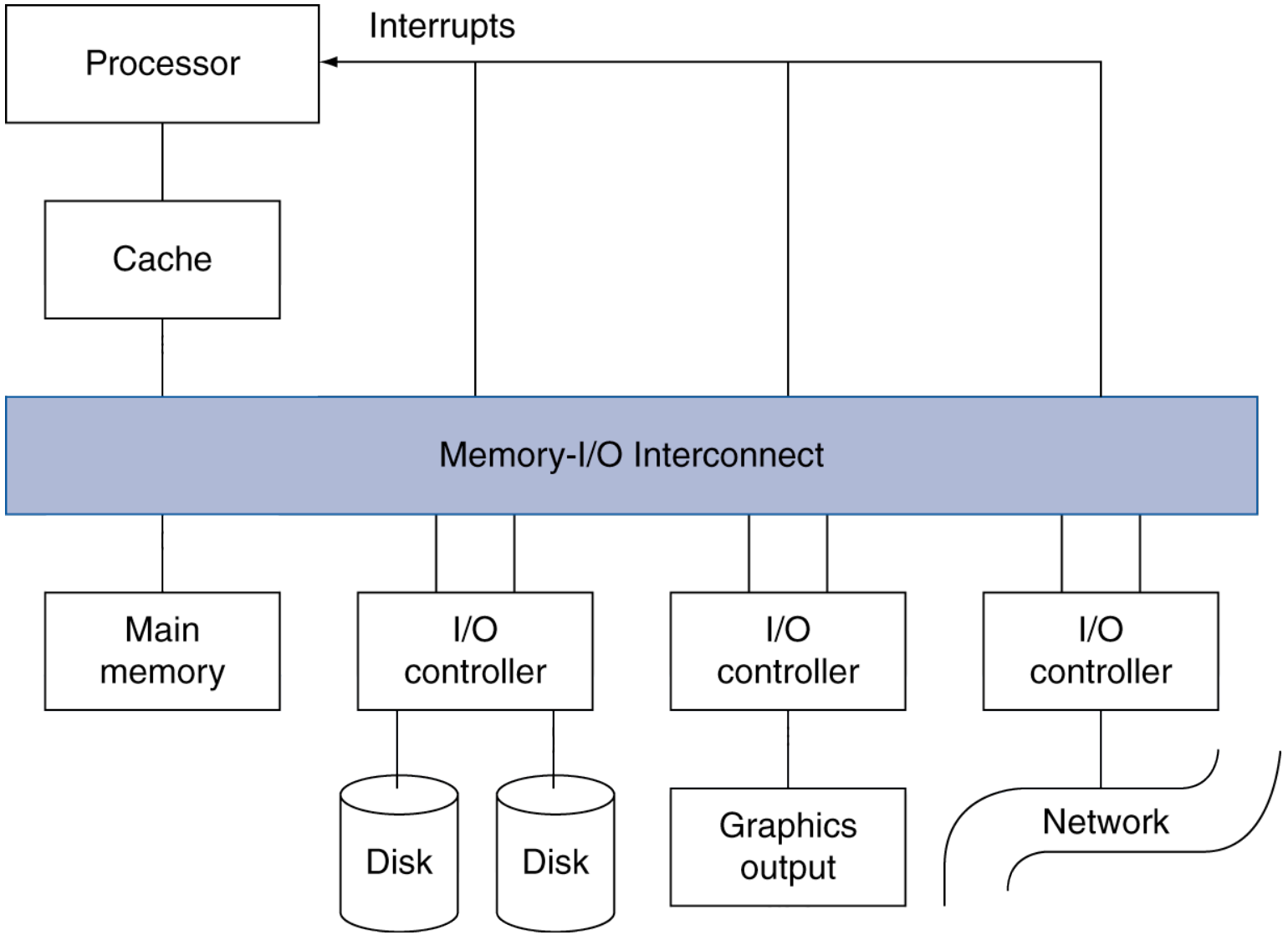
## Input/Output


Cornell University

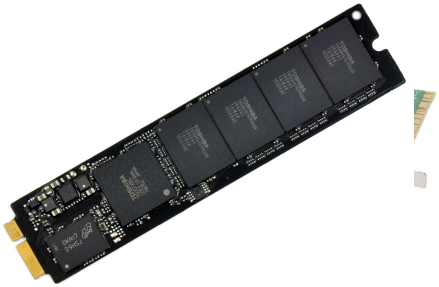
# Computer with Input/Output



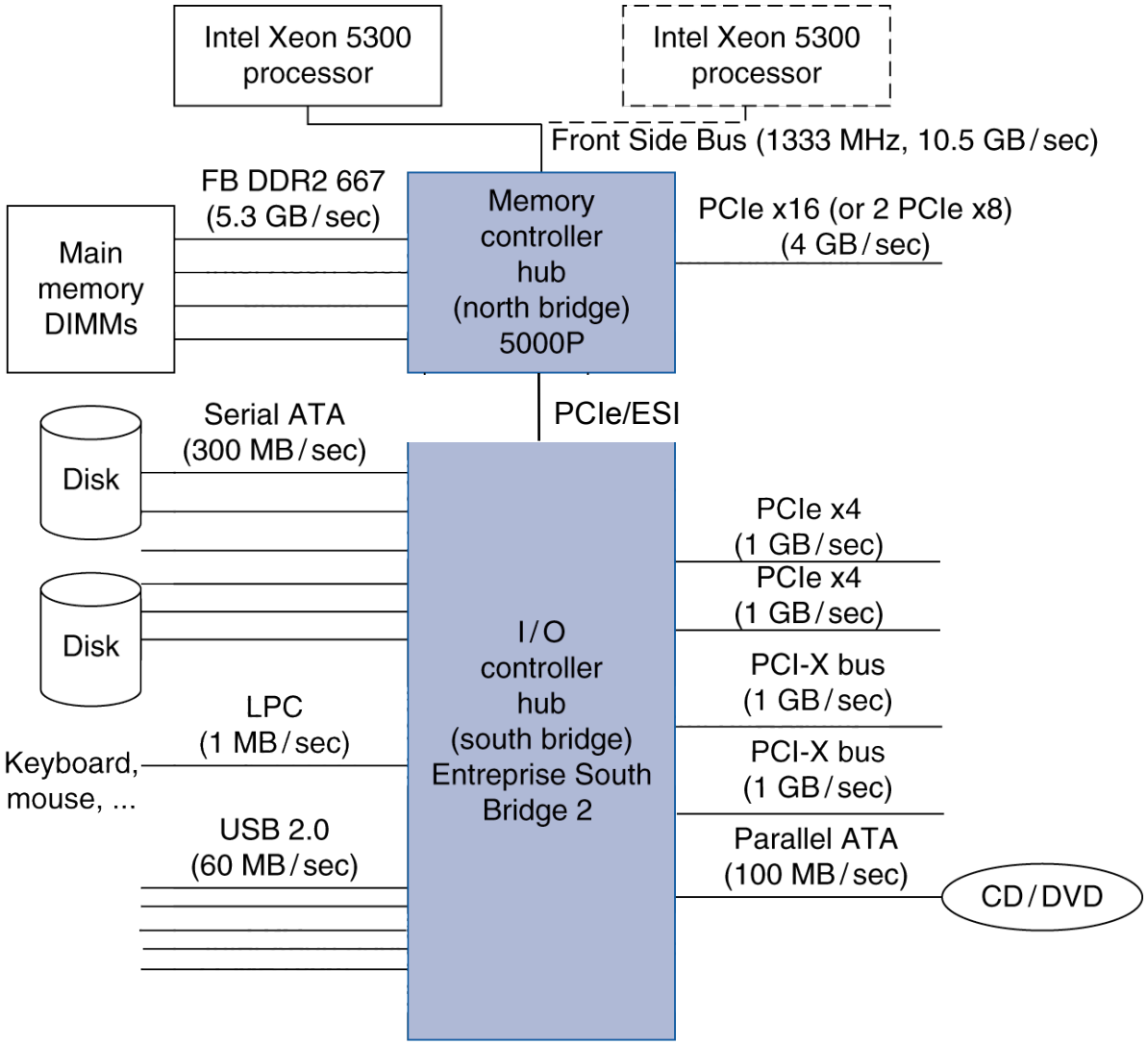
# Computer with Input/Output



# Some Input/Output Devices

# Example Server System with I/O

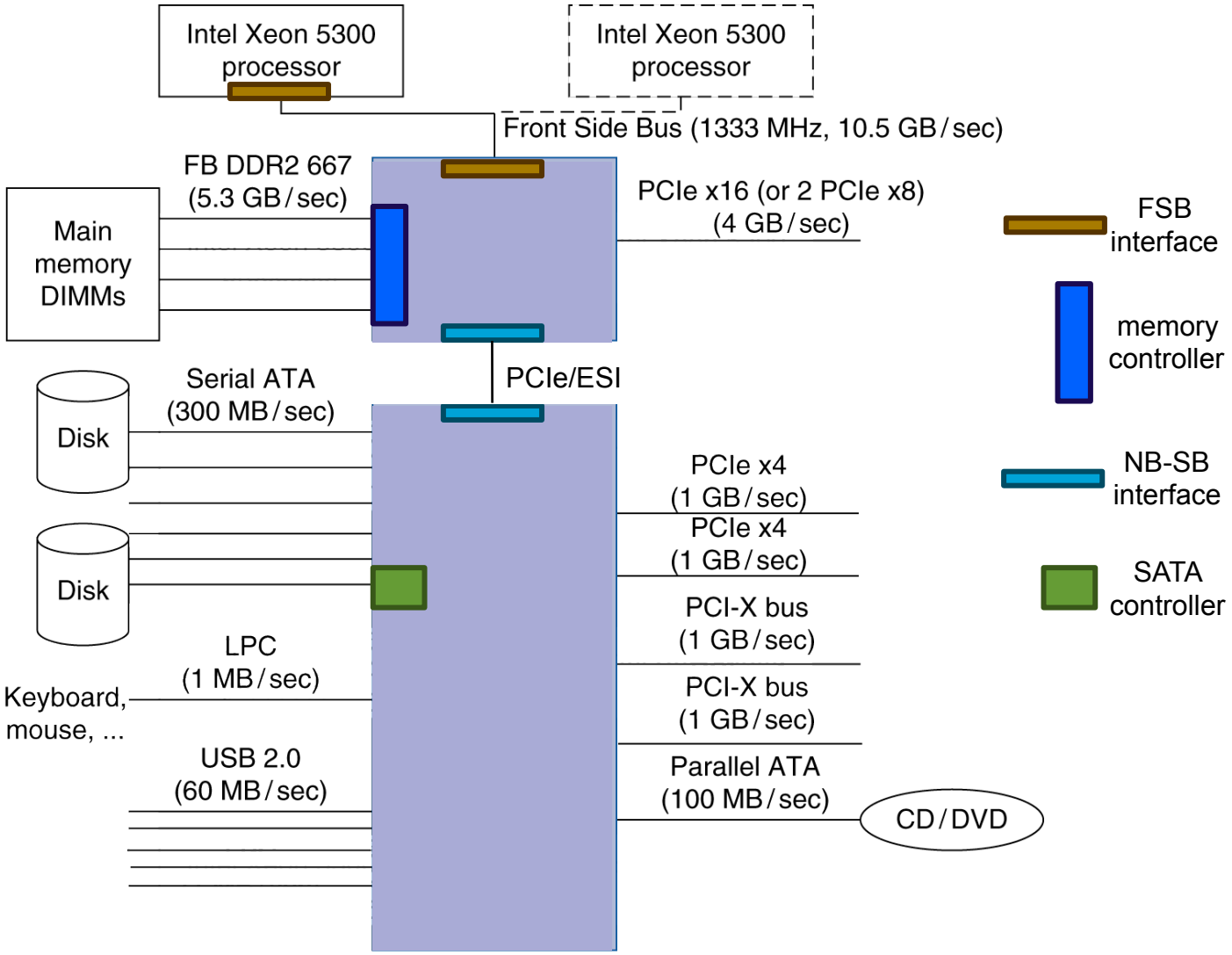


# Interface Standards

| | Firewire | USB 2.0 | PCI Express (PCIe) | Serial ATA | Serial Attached SCSI |
|---|---|---|---|---|---|
| Intended use | External | External | Internal | Internal | External |
| Devices per channel | 63 | 127 | 1 | 1 | 4 |
| Data width | 4 | 2 | 2/lane | 4 | 4 |
| Peak bandwidth | 50MB/s or 100MB/s | 0.2MB/s, 1.5MB/s, or 60MB/s | 250MB/s/lane 1×, 2×, 4×, 8×, 16×, 32× | 300MB/s | 300MB/s |
| Hot pluggable | Yes | Yes | Depends | Yes | Yes |
| Max length | 4.5m | 5m | 0.5m | 1m | 8m |
| Standard | IEEE 1394 | USB Implementers Forum | PCI-SIG | SATA-IO | INCITS TC T10 |

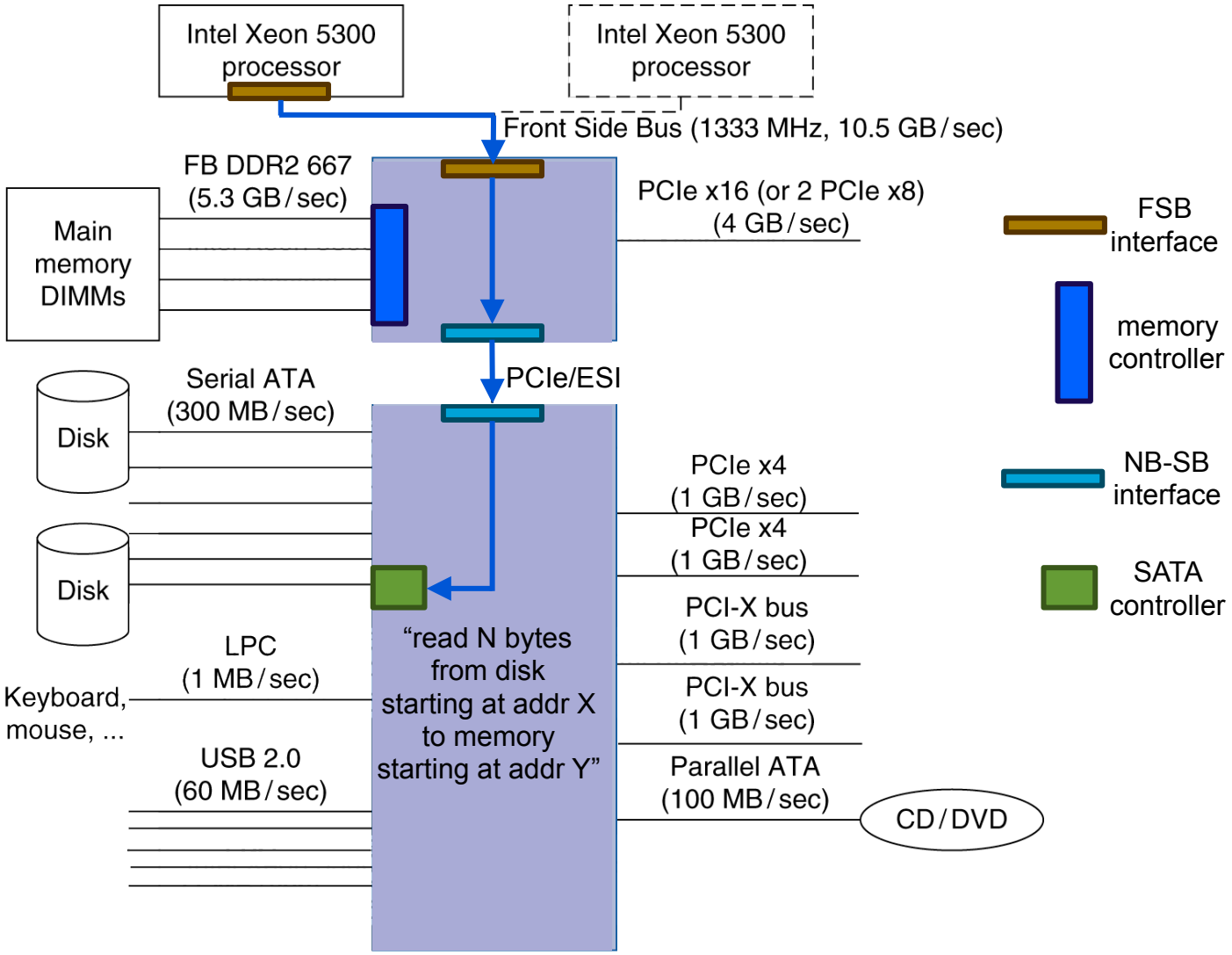
FB DDR2 = Fully-Buffered Double Data Rate Version 2 (DRAM interface standard)

# Reading Data from Disk

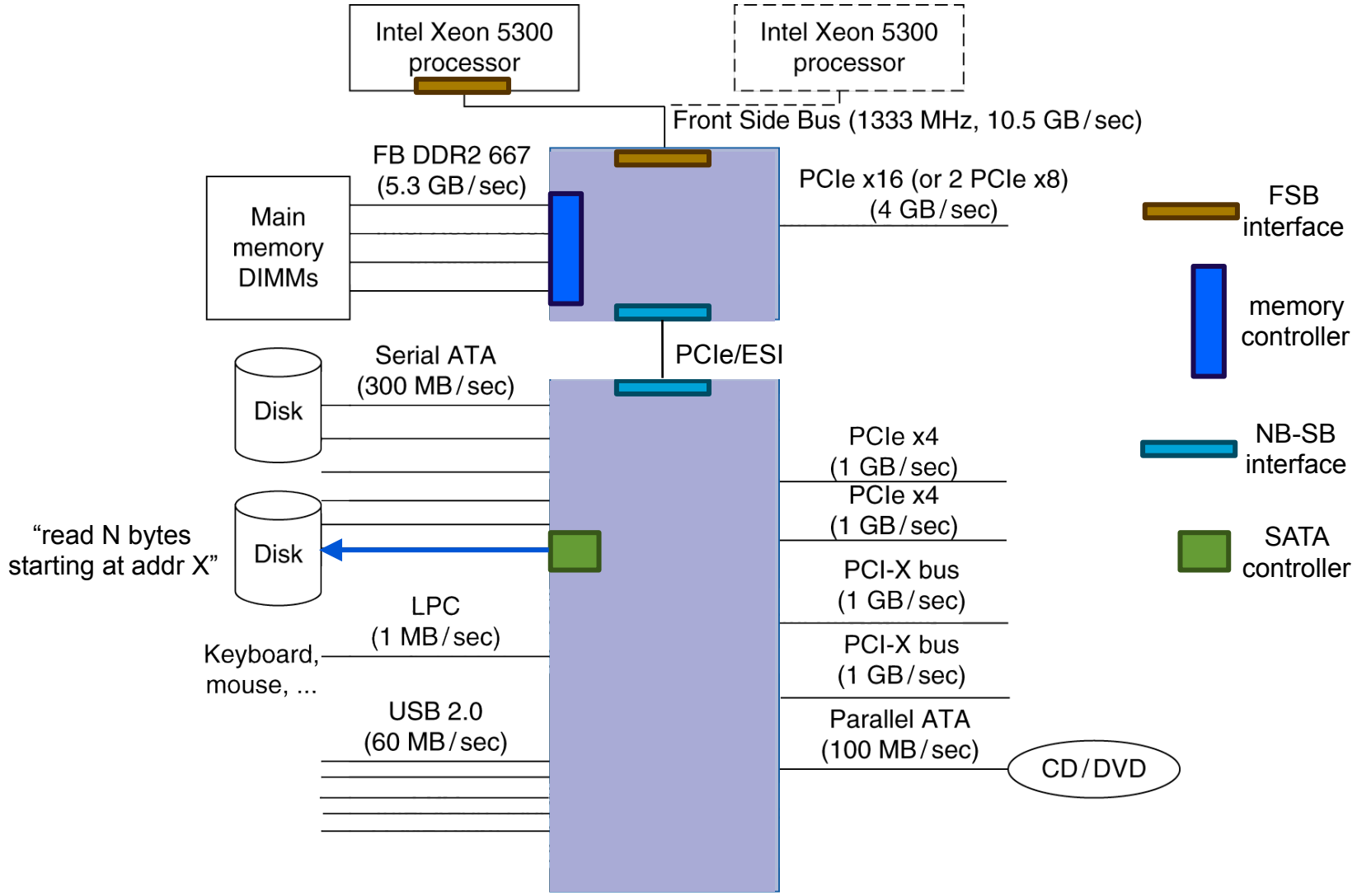



Caveat: The examples don't reflect the exact operation of this system

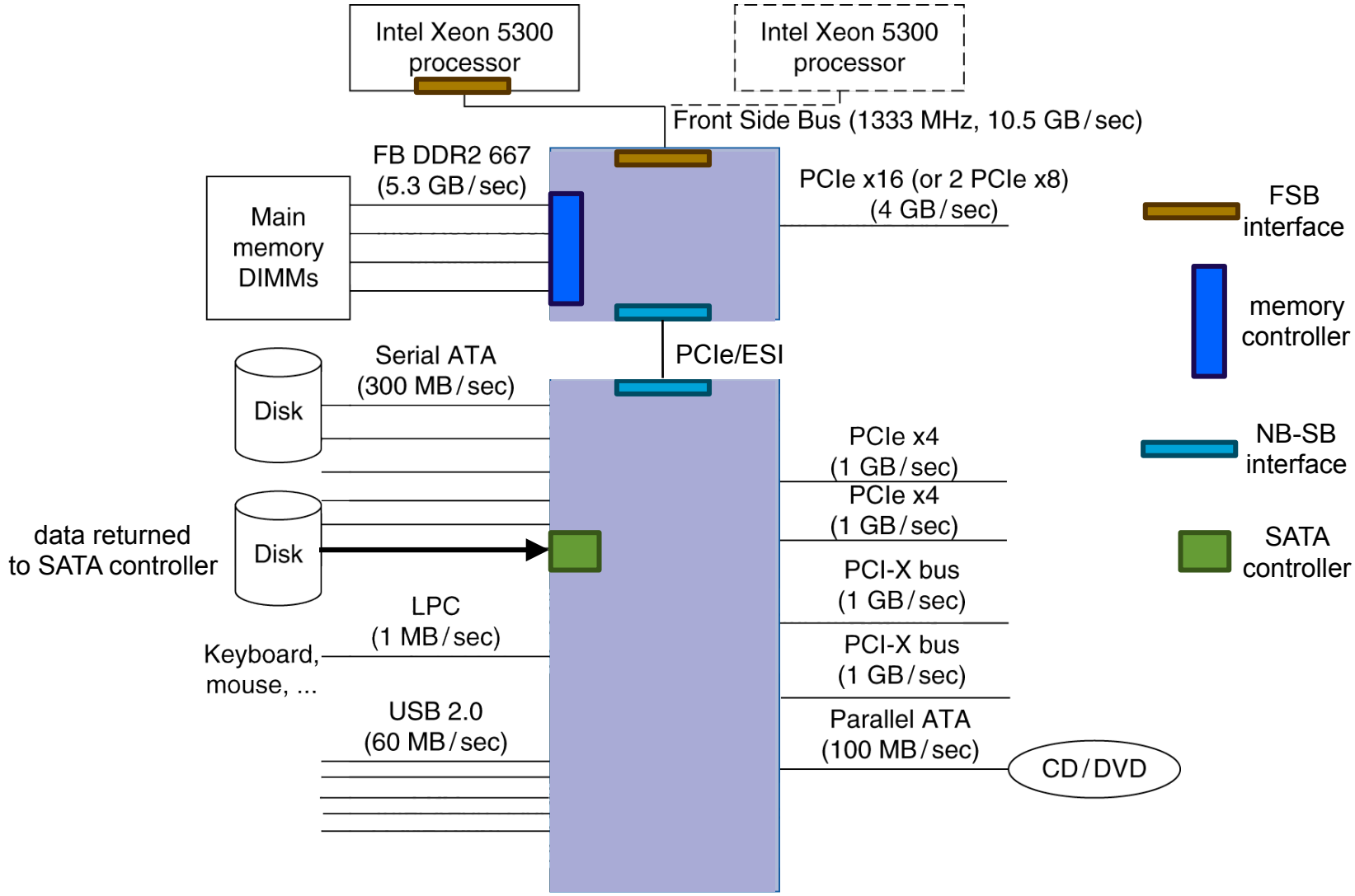
# Reading Data from Disk



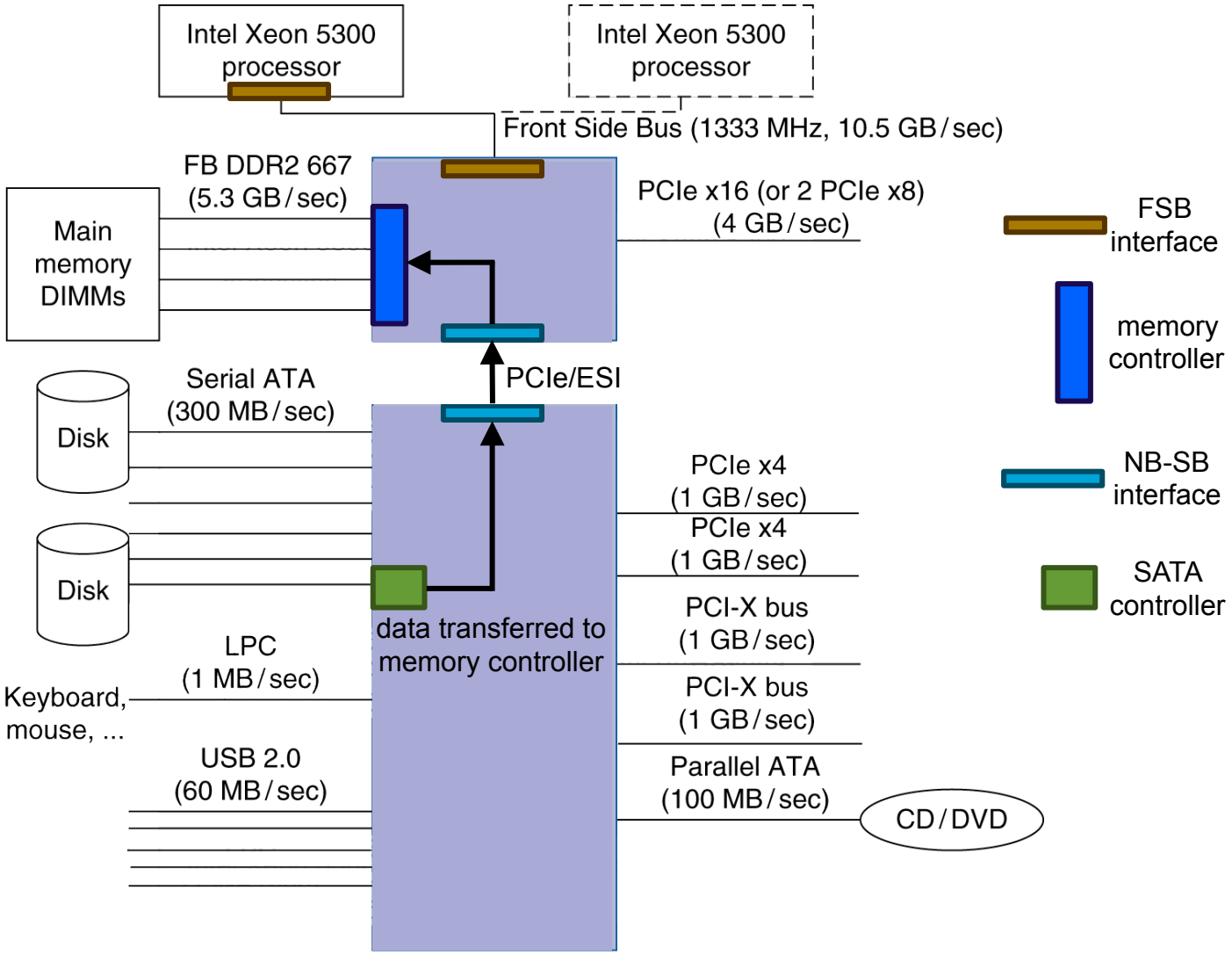
# Reading Data from Disk



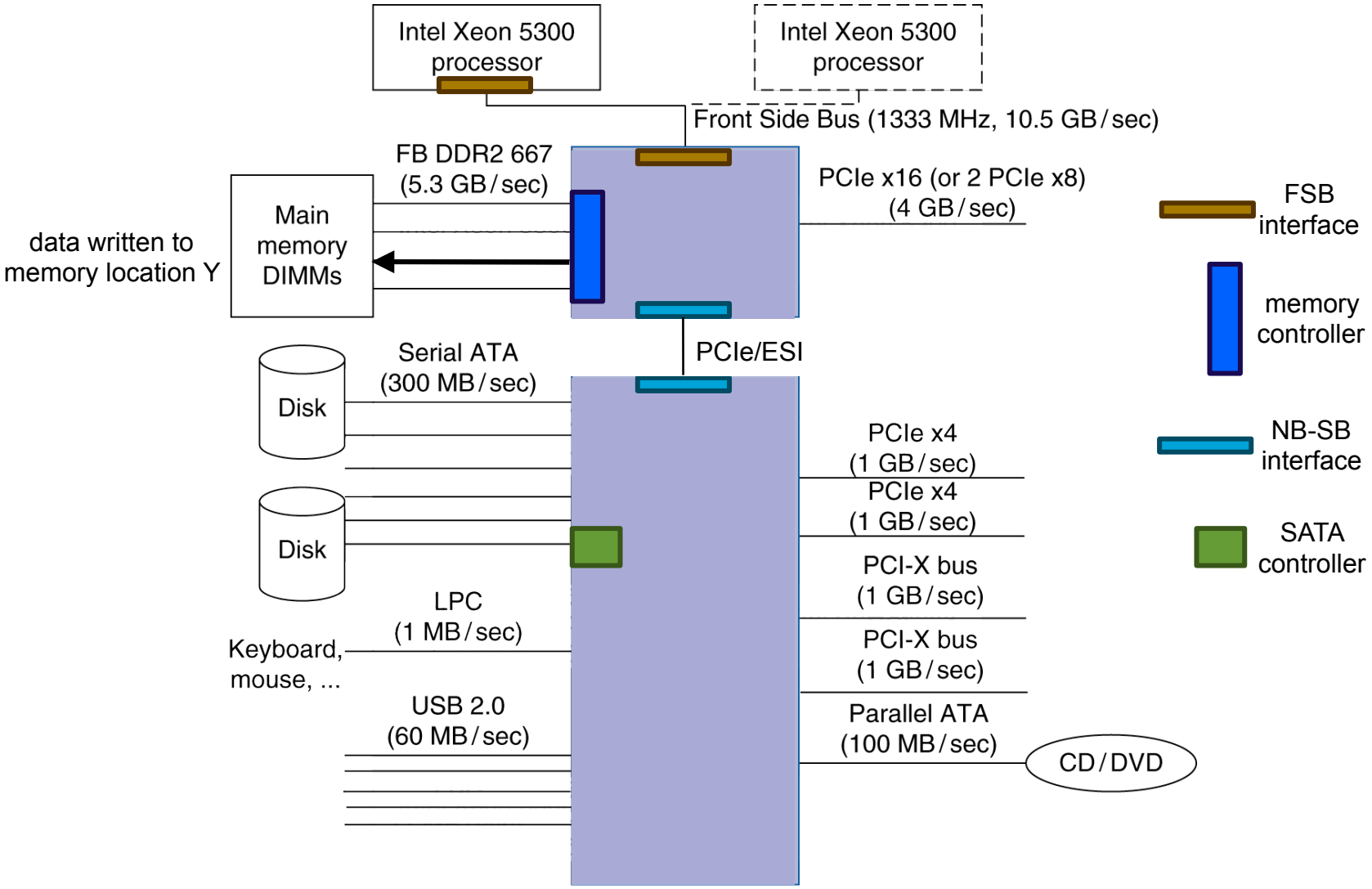
# Reading Data from Disk



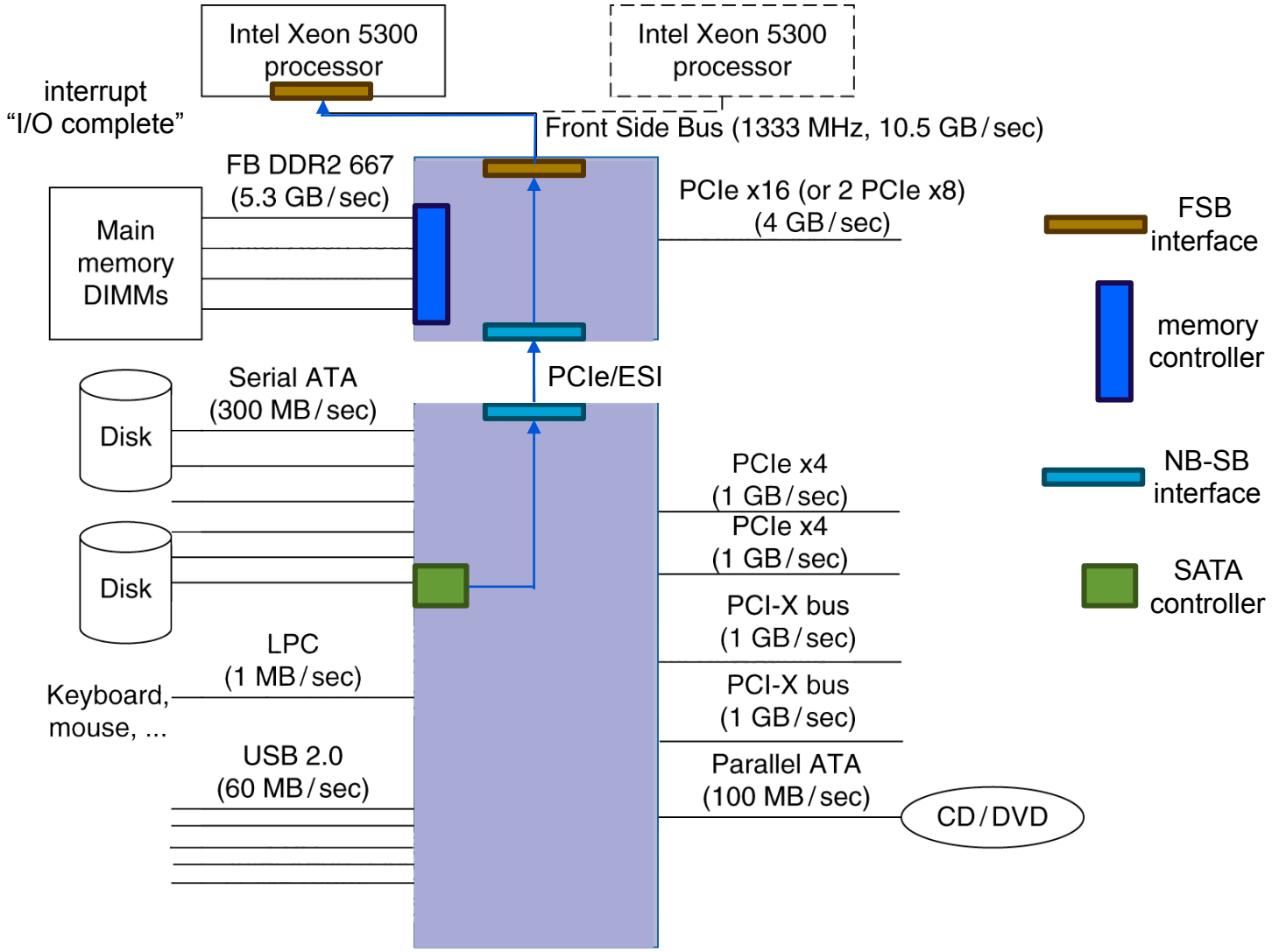
# Reading Data from Disk



# Reading Data from Disk



# Reading Data from Disk



# Sending Commands to I/O Devices

- I/O controllers have special registers for communication with the processor

- Command register
  - Tells the device to do something
  - "read N bytes from disk..."
  - Written by the processor/OS using Store instructions

- Status register
  - Indicates the status of the device (ready, busy, error)
  - Read by the processor/OS using Load instructions

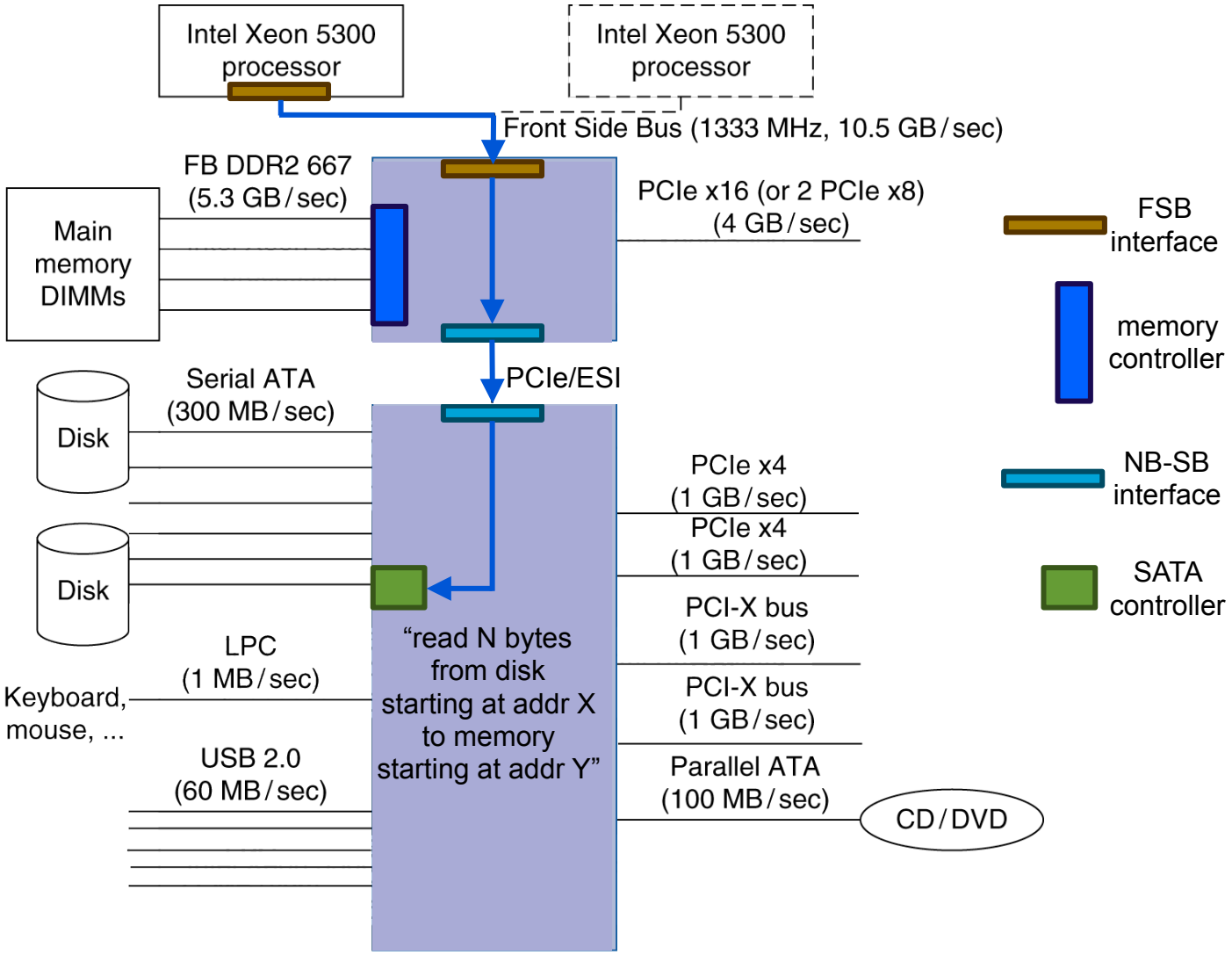
# Sending Commands to I/O Devices

- How do we get a command to the right device?

- Memory-mapped I/O
  - Portion of the physical memory space is assigned to I/O device registers
  - Only the OS can use these addresses
  - Each I/O device register has a unique memory address

- I/O instructions
  - Separate Load/Store instructions to access I/O registers
  - Only the OS can use these instructions (U/S = 1)
  - Separate addresses for I/O devices

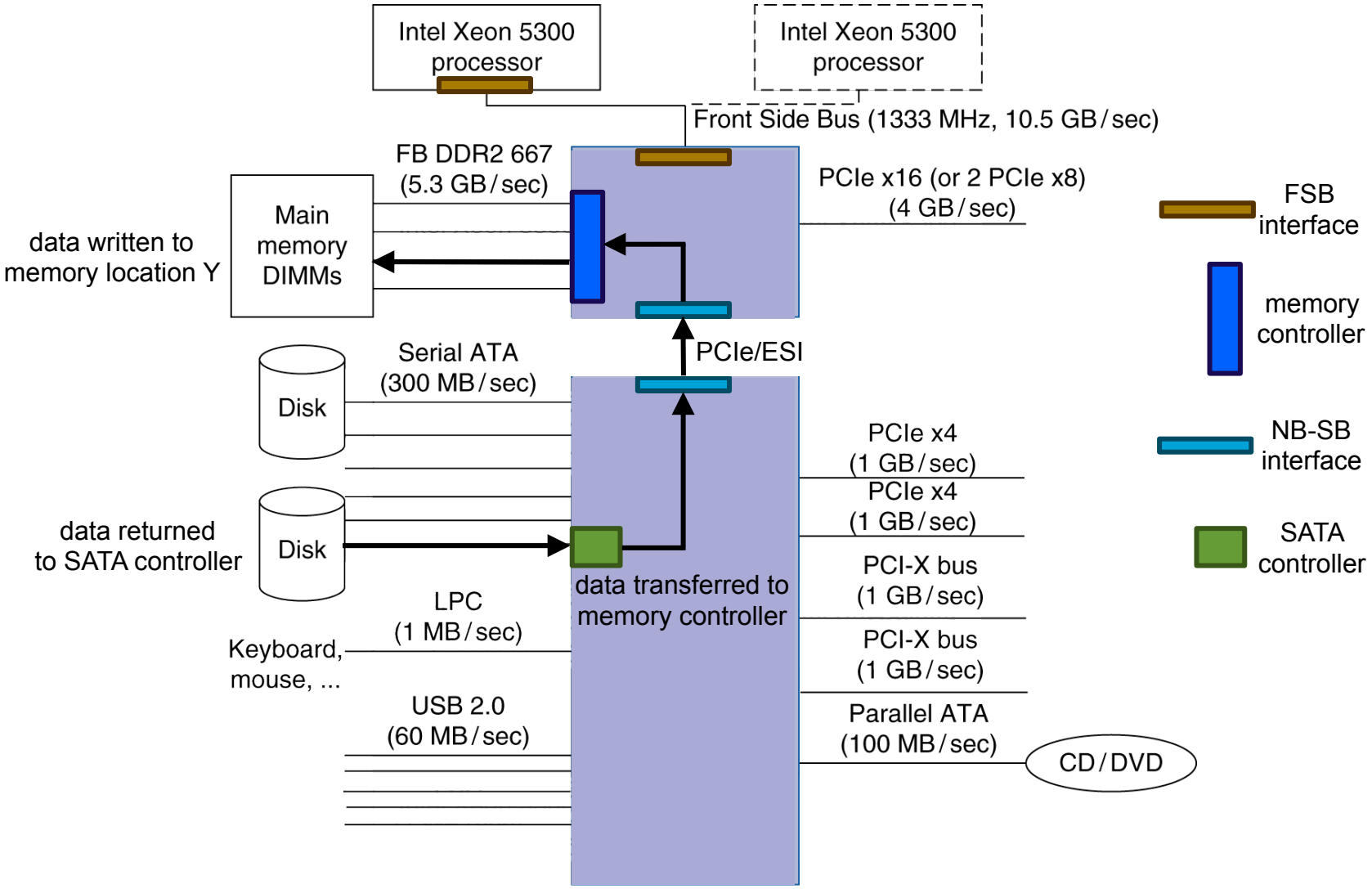
# I/O Command to Disk Controller



# Data Transfer Between I/O and Memory

- *Direct Memory Access (DMA)*
- I/O device transfers data directly to main memory
- Processor/OS sets up the transfer through I/O commands
  - And then can do something else, like run another program

# DMA Transfer from Disk Controller



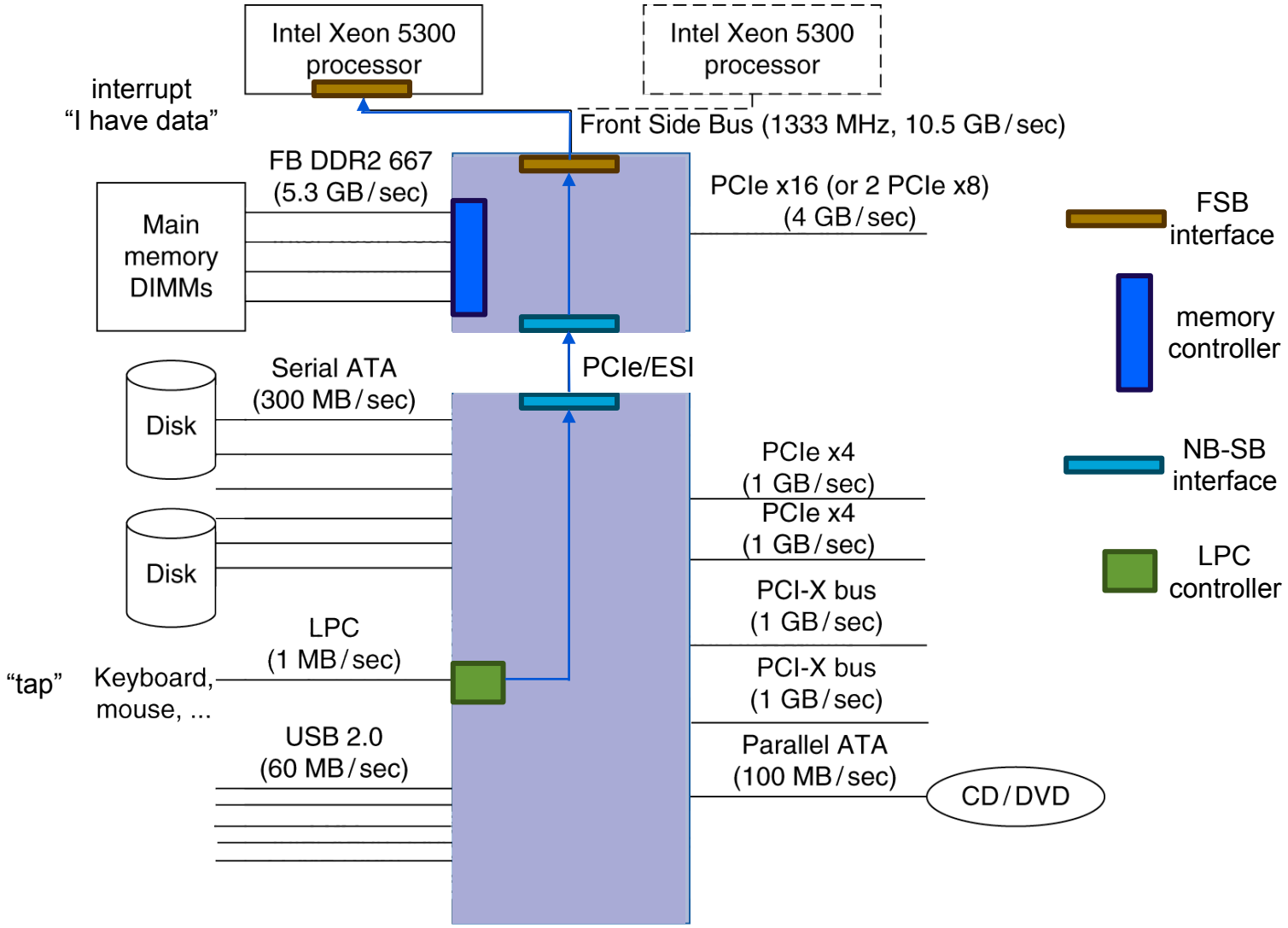
# Data Transfer Between I/O and Memory

- *Programmed I/O (PIO)*: Processor completely handles transfer of data from device to memory

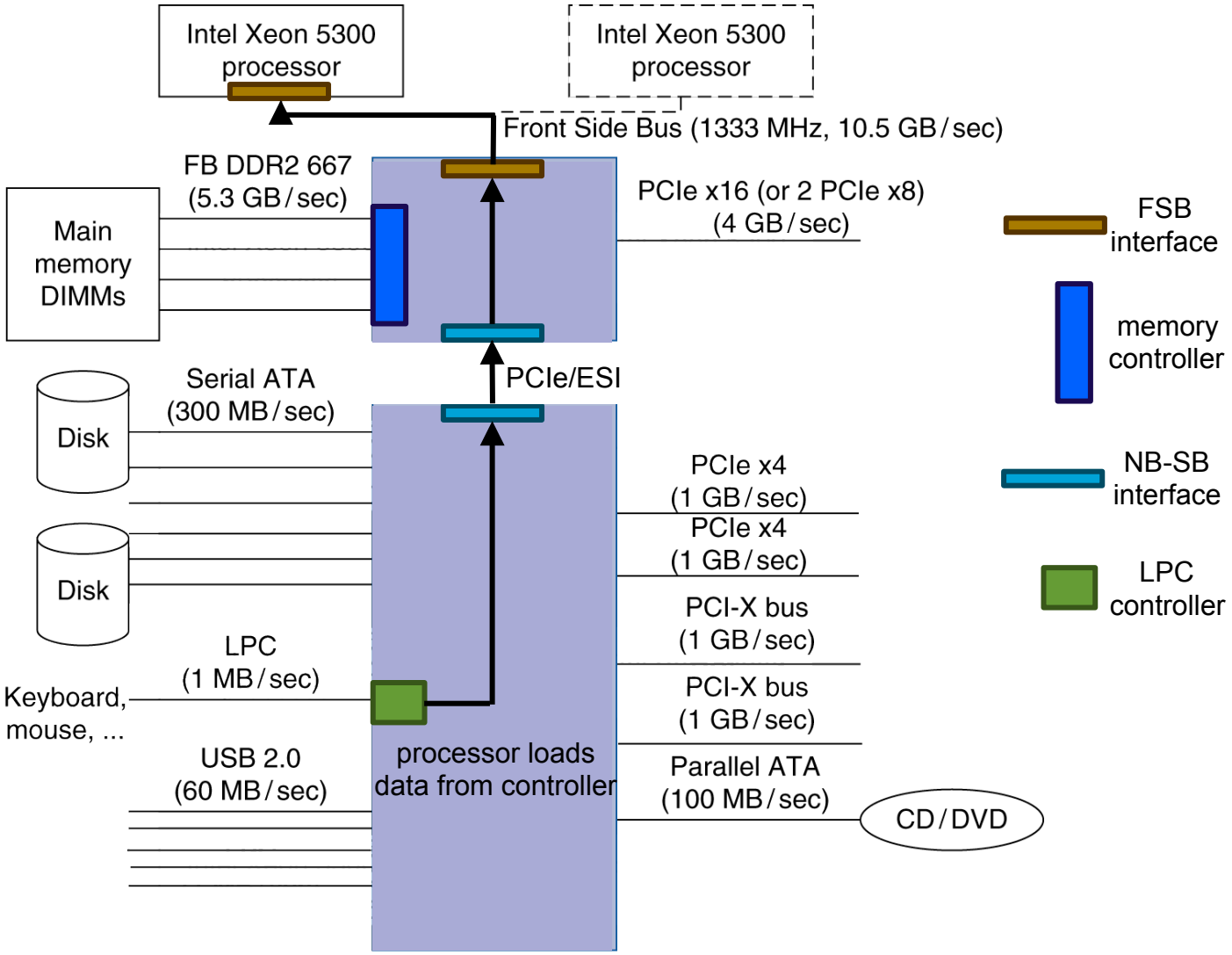
- Input
  - Sets up an I/O read through I/O commands
  - Waits until data becomes ready in the device
  - Reads data from I/O device Data Output register, then writes it into memory

- Output
  - Reads data from memory, then writes it to I/O device Data Input register
  - Sets up the I/O write through I/O commands

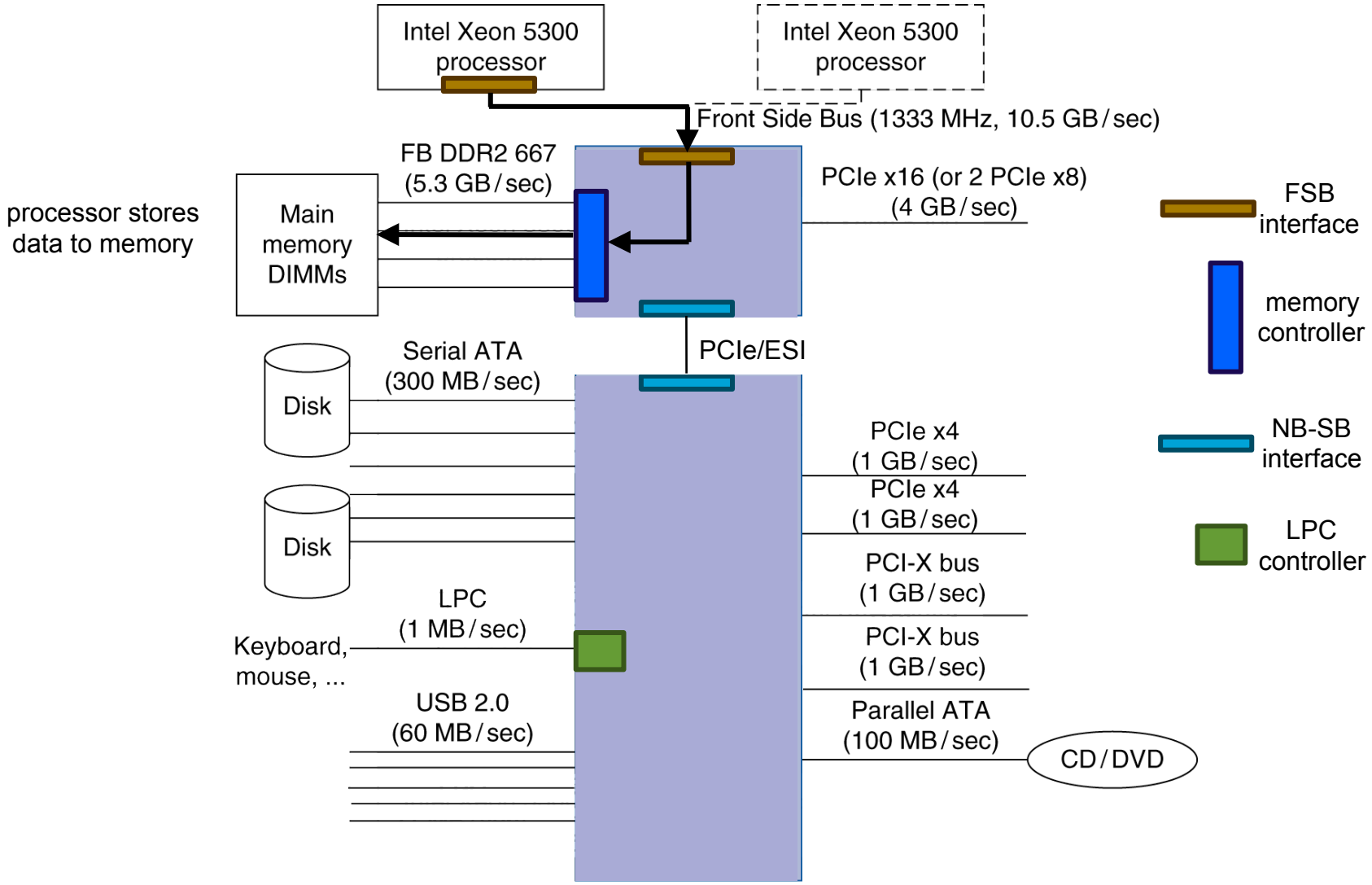
# PIO Transfer from Keyboard



# PIO Transfer from Keyboard



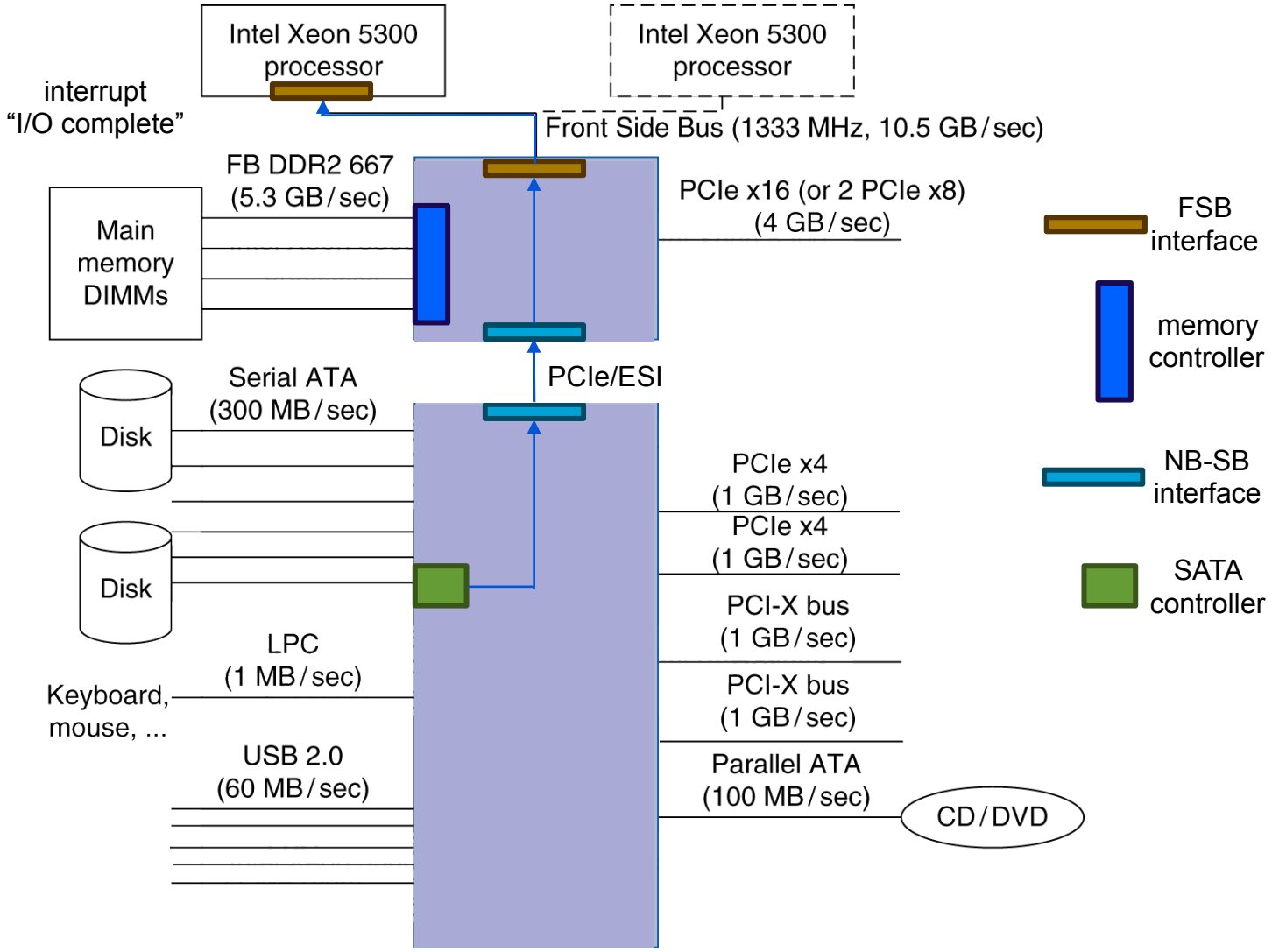
# PIO Transfer from Keyboard



# Informing the Processor

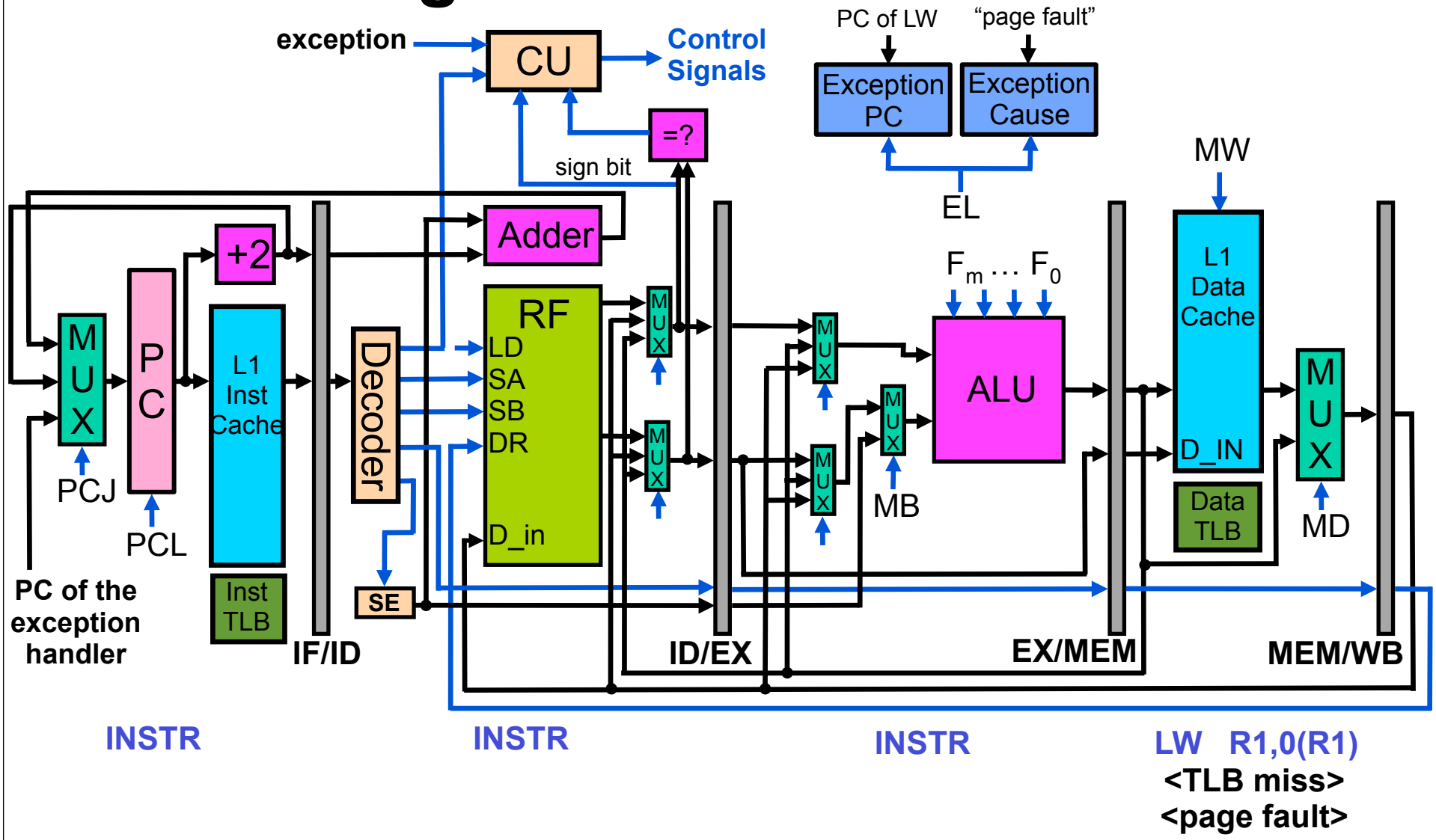
- Need to inform the processor when an I/O operation is completed
- Polling
  - I/O device Status Register indicates when an operation is done
  - Processor periodically reads the Status Register
- Interrupt-driven I/O
  - I/O device sends an interrupt to the processor when the operation is done

# Informing Using an Interrupt

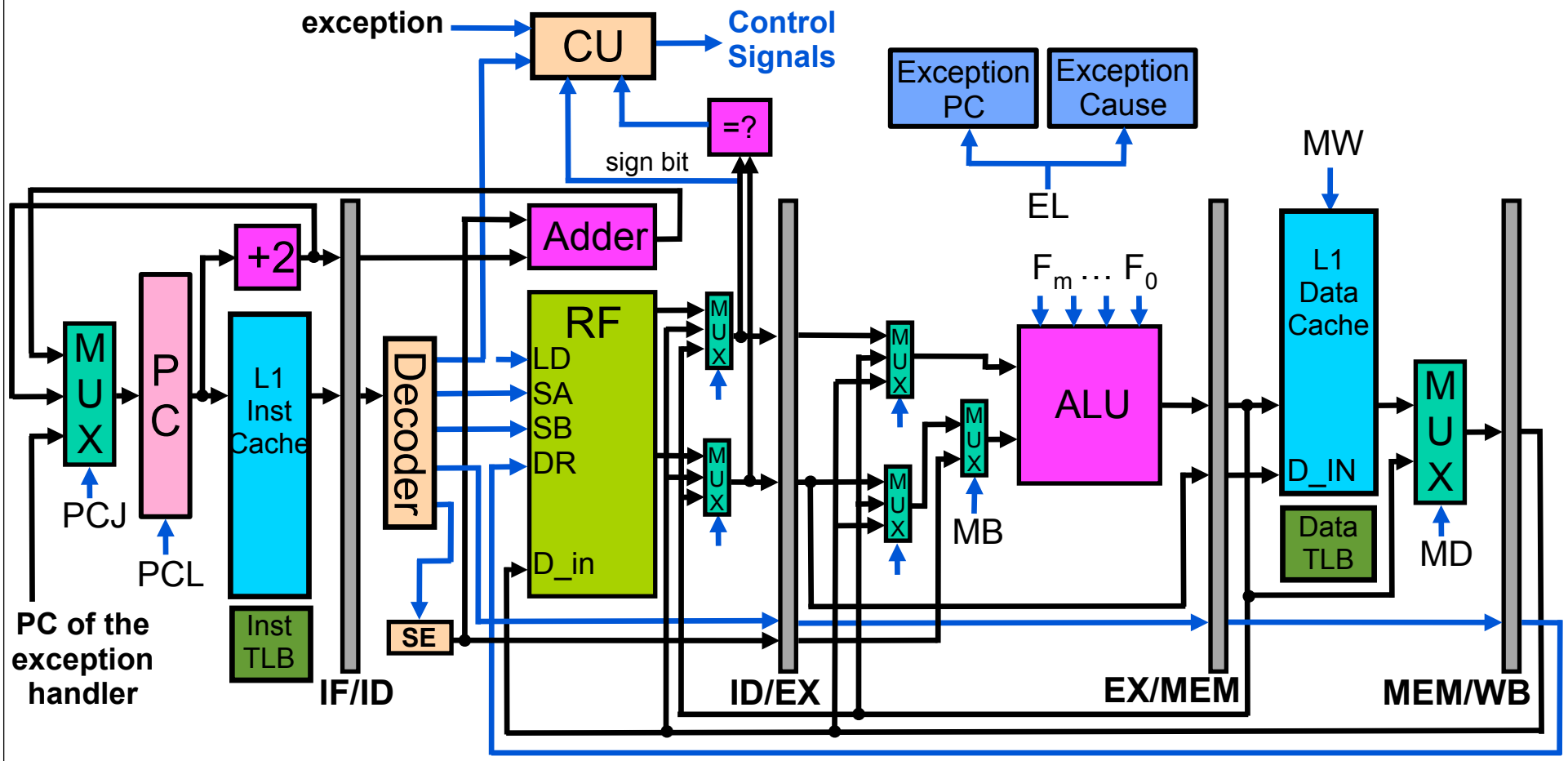


# Let's Pull Some Pieces Together

# Data Page Fault Occurs in Task A



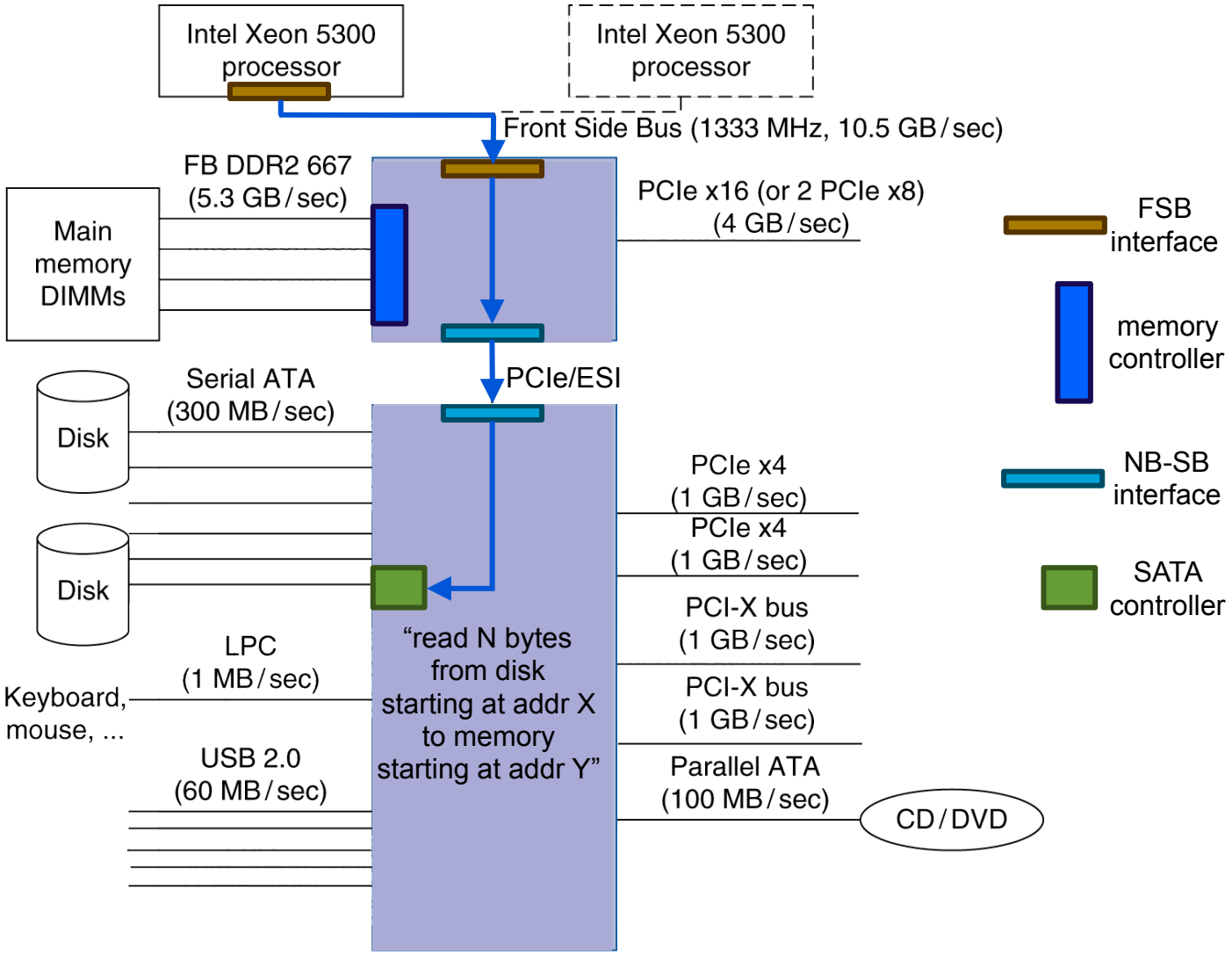
# Exception Handler Gets Loaded



# Exception Handler Takes Action

- Saves task A state (PC, PTR, and registers)
- Reads the Cause register and determines that a page fault occurred
- Calls the appropriate part of the OS

# OS Sets Up Disk Transfer

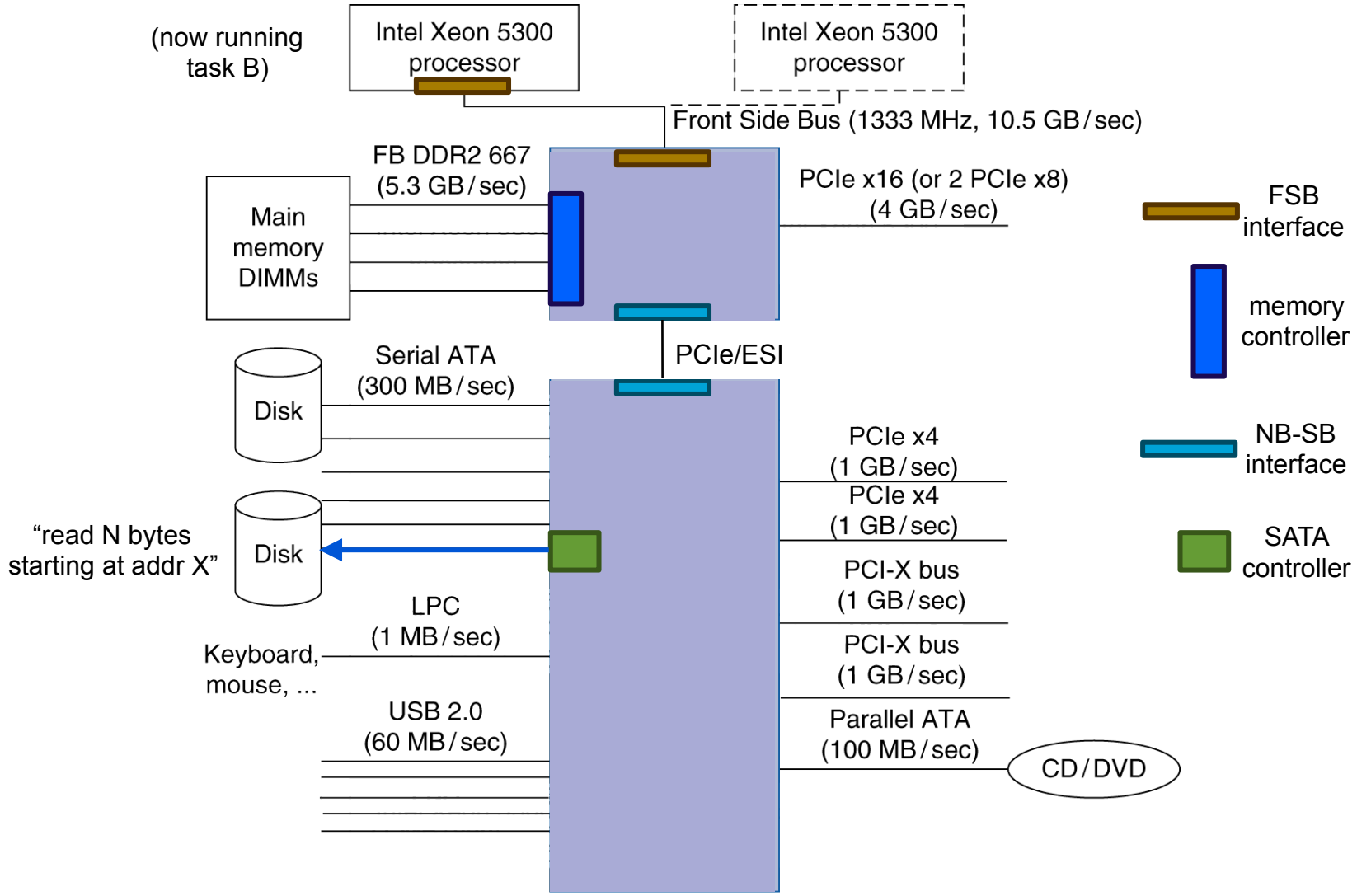


# OS Schedules Another Task

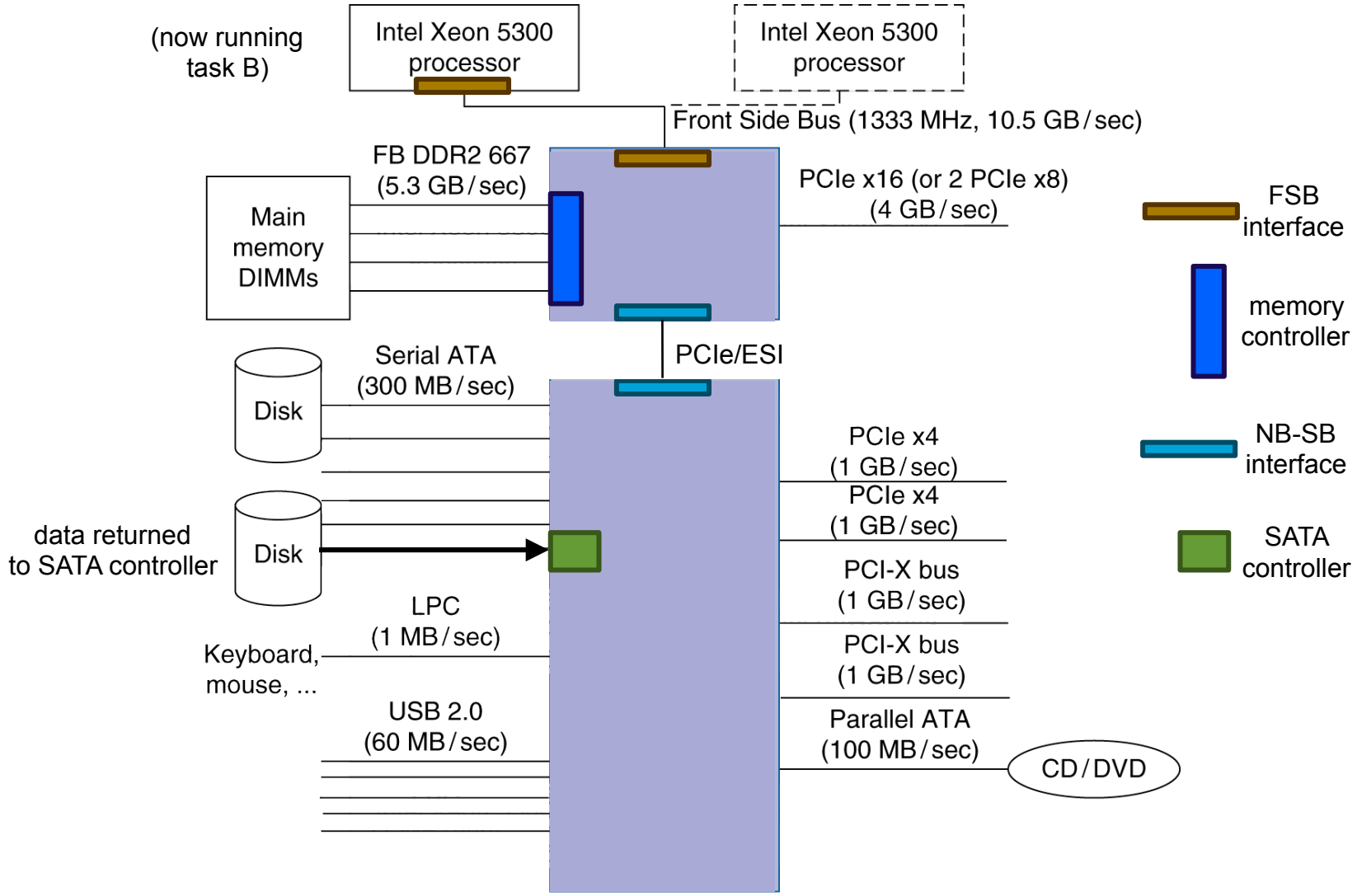
- While waiting for the disk read to complete, the OS scheduler runs a different task (task B)

- It loads the processor with the state (PC, PTR, and registers) of task B, and sets U/S = 0

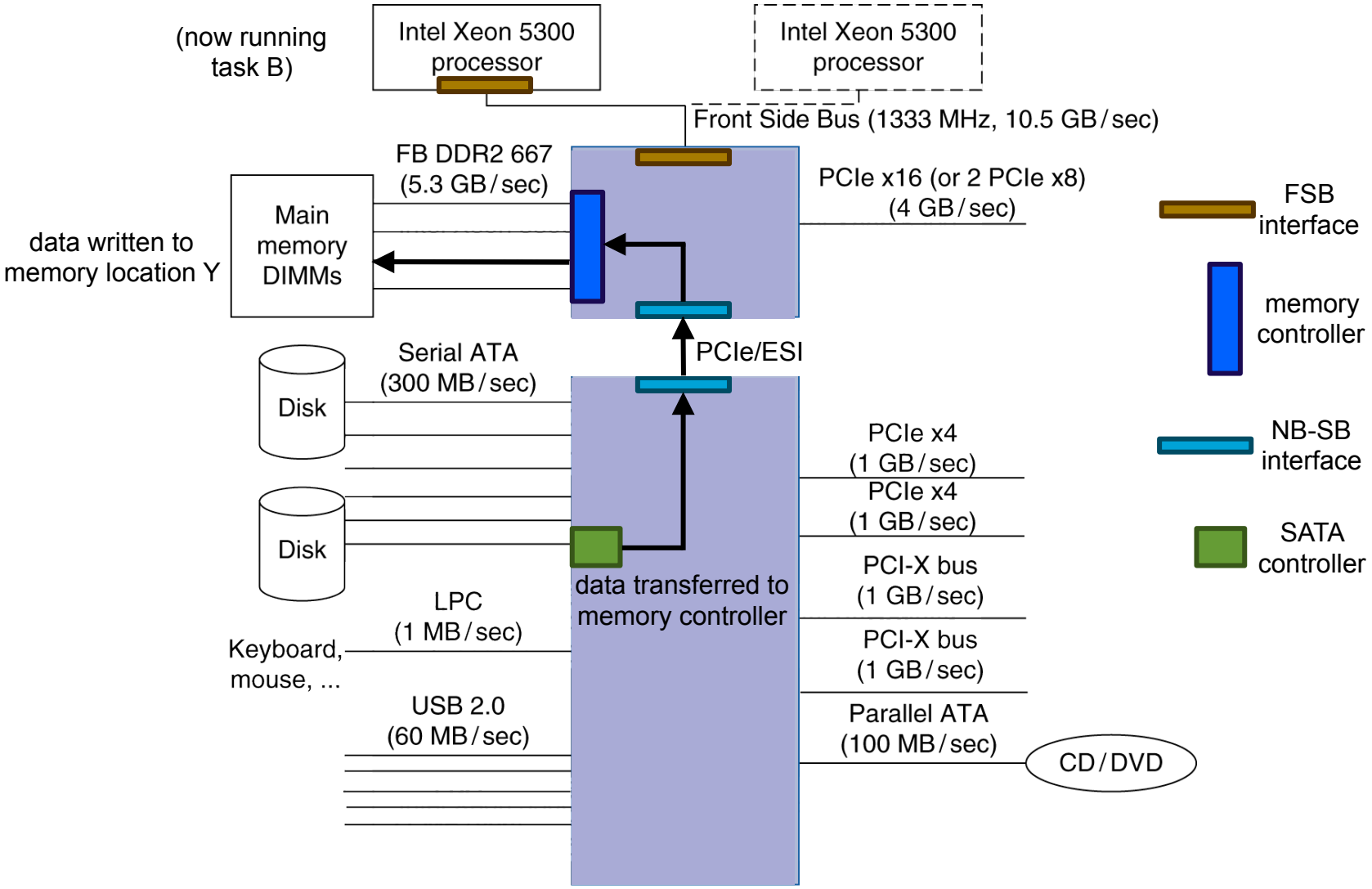
# Data is Read from Disk



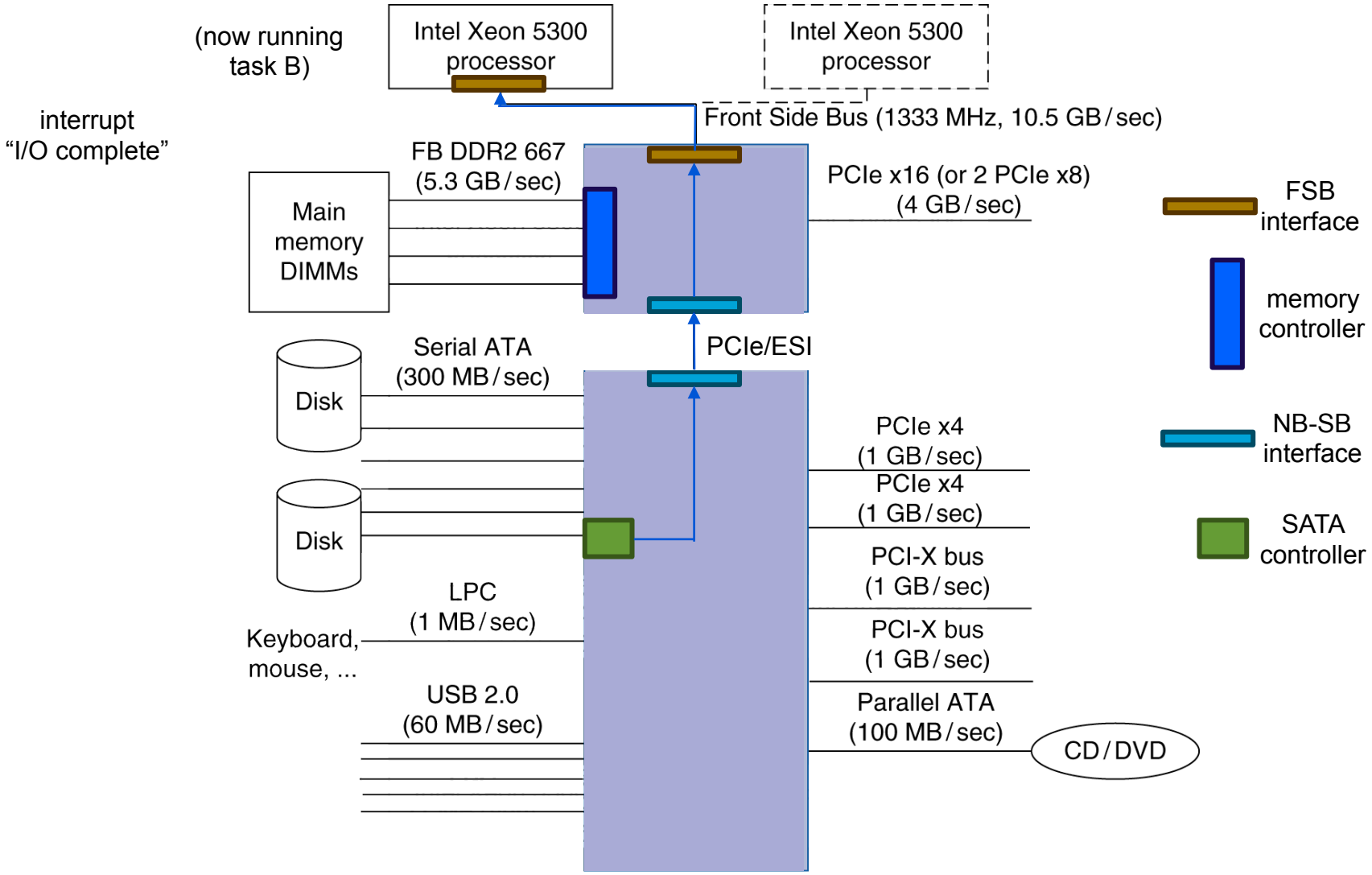
# Data is Read from Disk



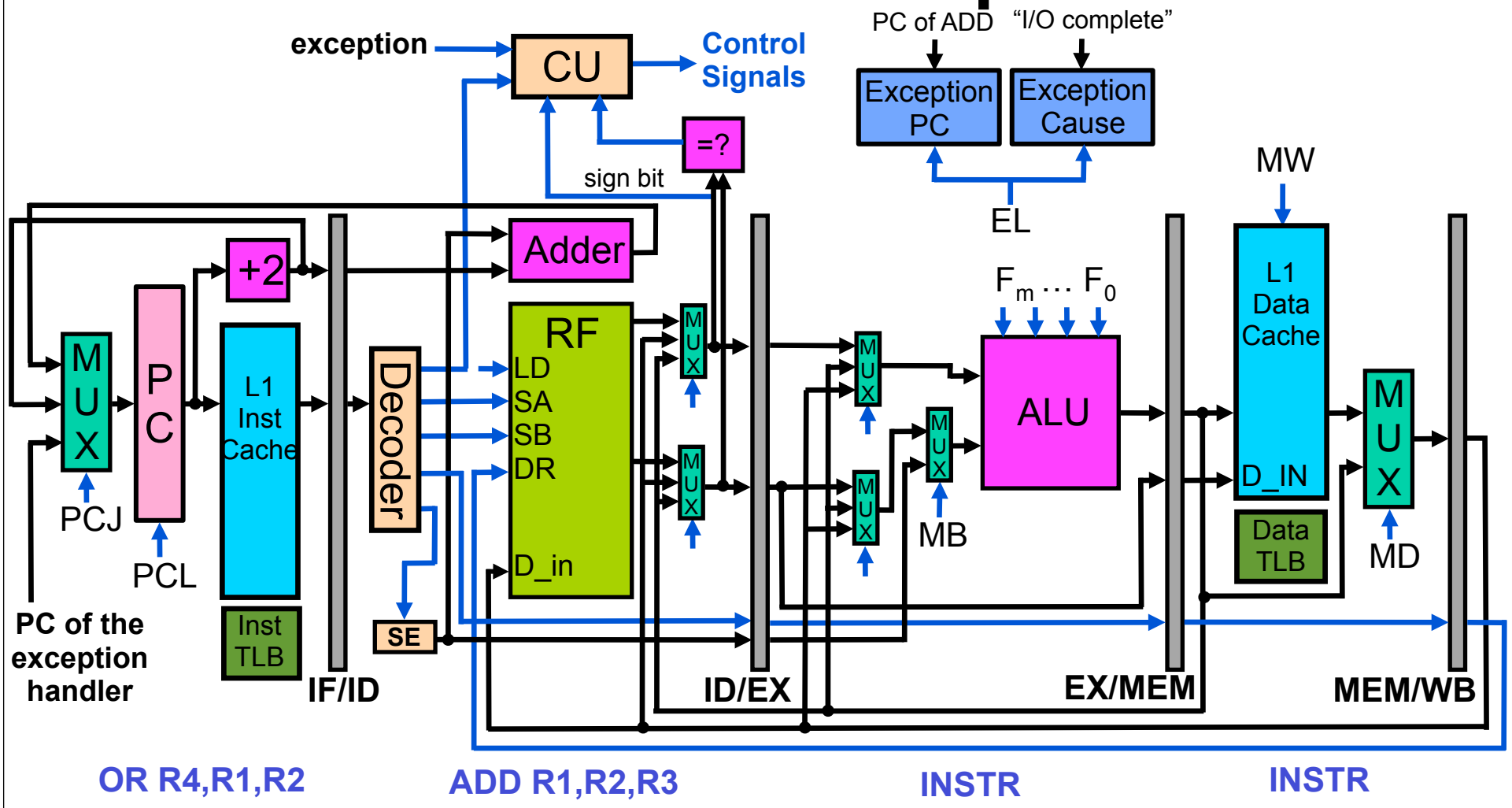
# DMA Transfer to Memory



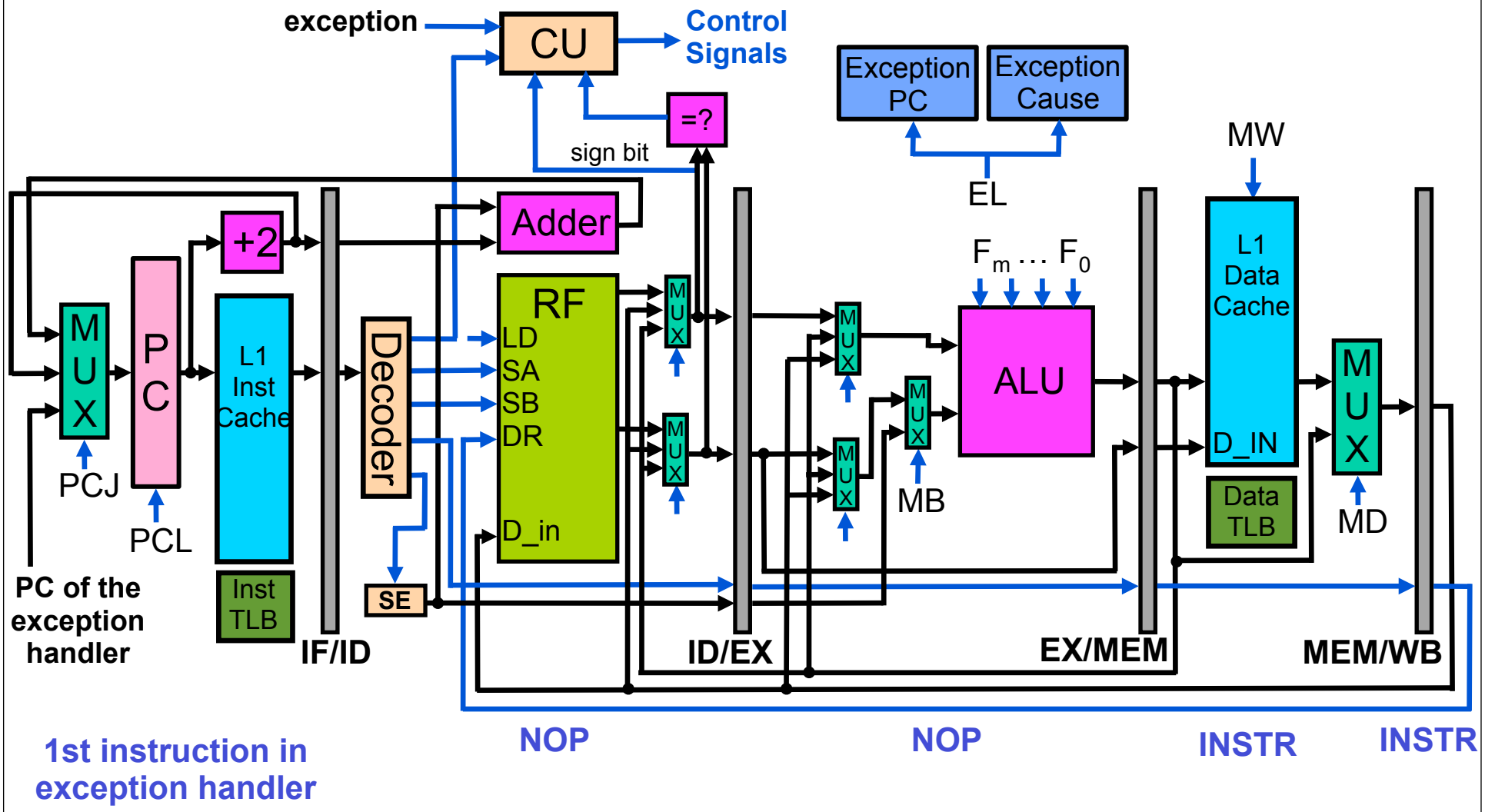
# Device Interrupts Processor



# Task B is Interrupted



# Exception Handler Gets Loaded



# Task A Can Now Run Again

- Page fault was handled, so OS marks task A as "runnable"
- If OS scheduler chooses to run task A, it loads its state (PC of the LW, PTR, and registers)
- Key point: Processor was free to do other work during the long I/O transfer time
  - DMA: Processor did not have to directly handle data transfers from device to memory
  - Interrupt-driven I/O: Processor did not have to poll the device to see when the I/O operation completed

# Next Time

## Advanced Topics