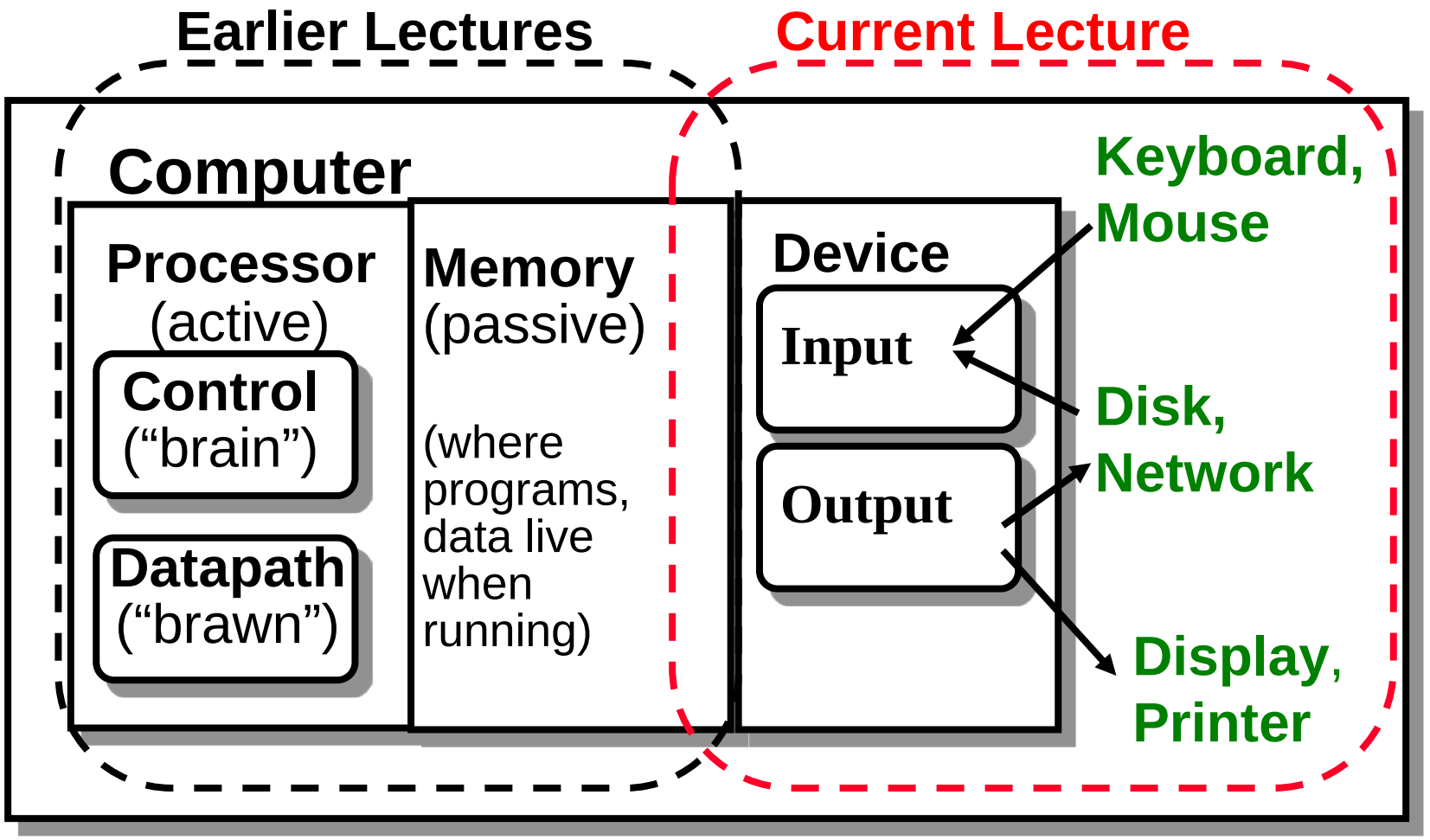
C335
Computer Structures

Input/Output
– Dealing with Exceptions and Interrupts

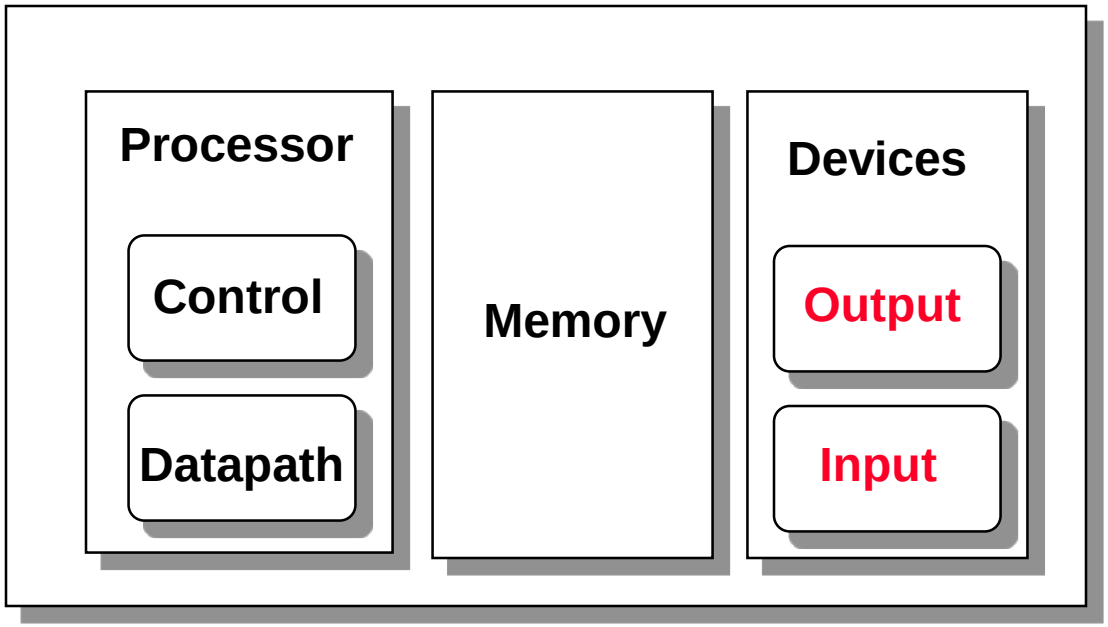
Dr. Liqiang Zhang

Department of Computer and Information Sciences

Recall : 5 components of any Computer



I/O Systems



- Important metrics for an I/O system
  - Dependability
  - Expandability and diversity
  - Performance
  - Cost, size, weight

Input and Output Devices

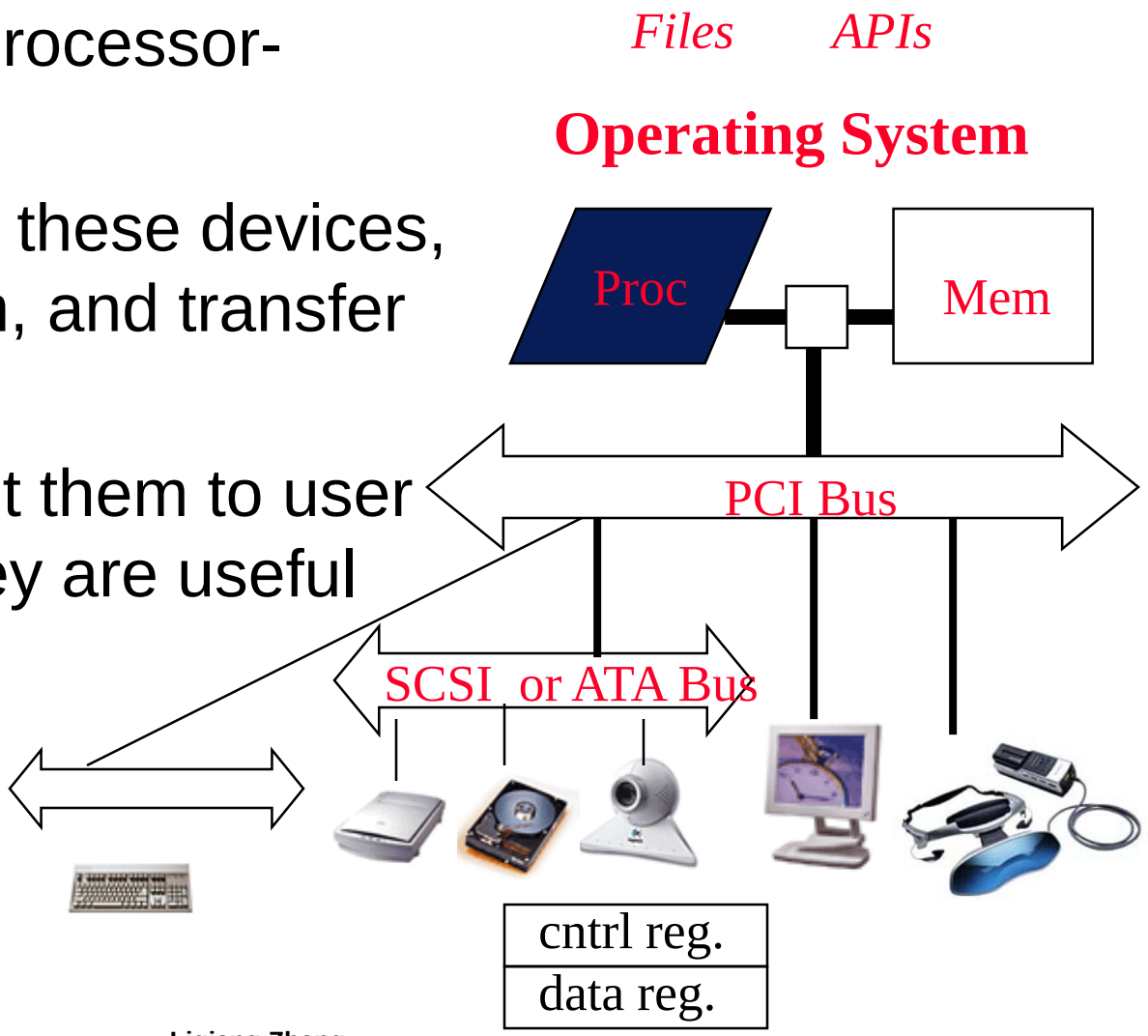
- I/O devices are incredibly diverse with respect to
  - Behavior – input, output or storage
  - Partner – human or machine
  - Data rate – the peak rate at which data can be transferred between the I/O device and the main memory or processor

Device	Behavior	Partner	Data rate (Mb/s)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Laser printer	output	human	3.2000
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-10000.0000
Magnetic disk	storage	machine	240.0000-2560.0000

8 orders of magnitude range

What do we need to make I/O work?

- A way to connect many types of devices to the Processor-Memory
- A way to control these devices, respond to them, and transfer data
- A way to present them to user programs so they are useful

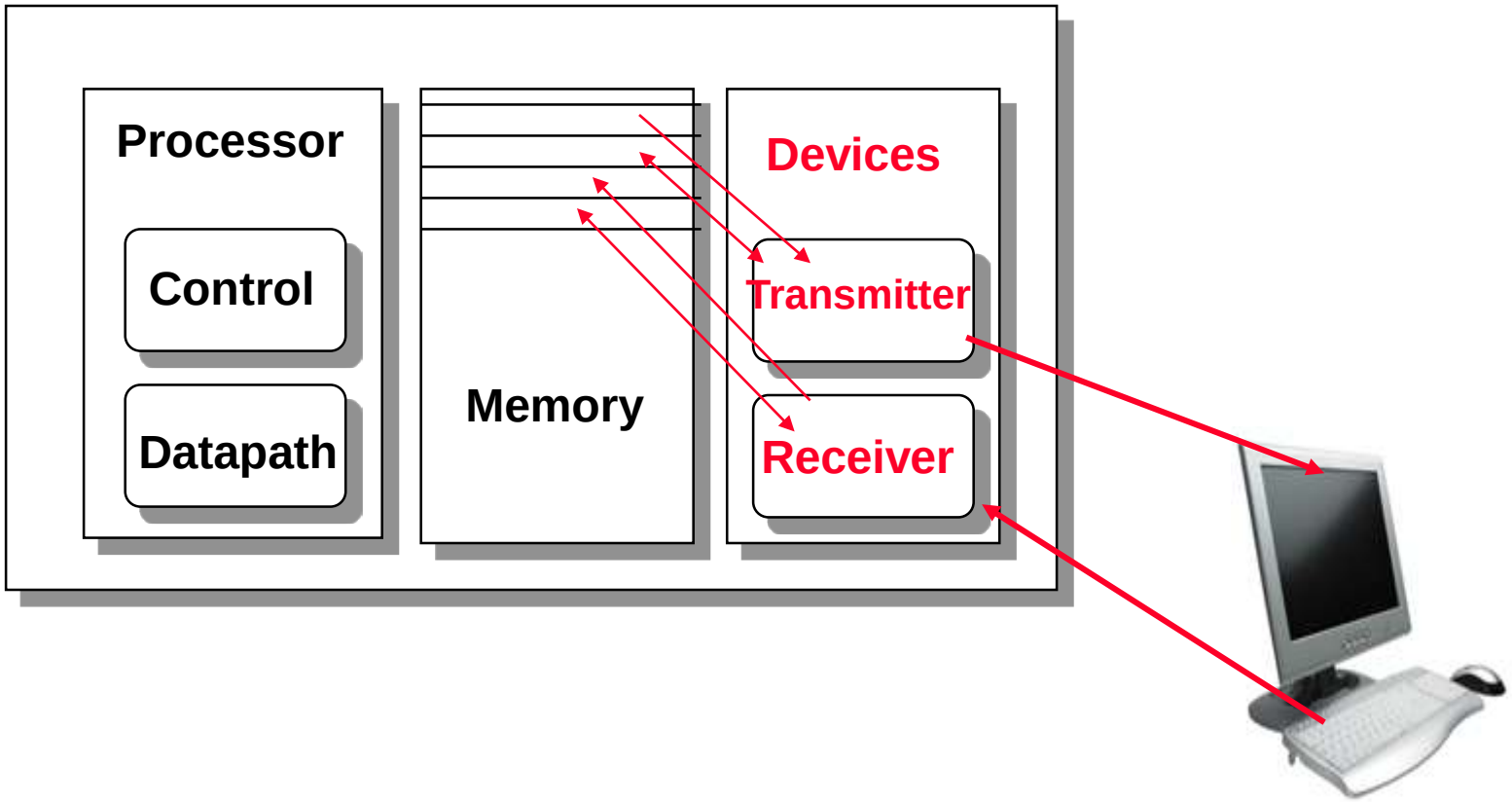


Communication of I/O Devices and Processor

- How the processor directs the I/O devices
  - Special I/O instructions (Standard I/O)
    - Must specify both the device and the command
    - Additional pin (*M/IO*) on bus indicates whether a memory or peripheral access
  - Memory-mapped I/O
    - Portions of the high-order memory address space are assigned to each I/O device
    - Read and writes to those memory addresses are interpreted as commands to the I/O devices
- How the I/O device communicates with the processor
  - Polling – the processor periodically checks the status of an I/O device to determine its need for service
    - Processor is totally in control – but does all the work
    - Can waste a lot of processor time due to speed differences
  - Interrupt-driven – the I/O device issues an interrupts to the processor to indicate that it needs attention

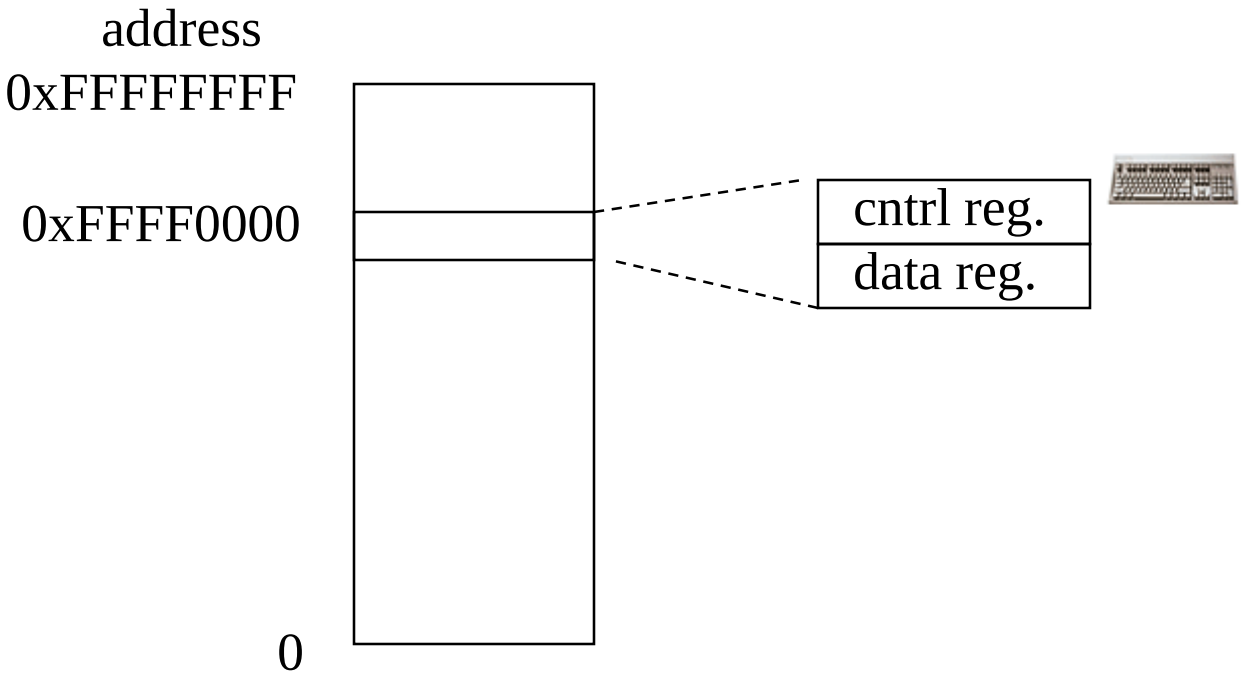
I/O in SPIM

- SPIM supports one memory-mapped I/O device – a terminal with two independent units
  - Receiver reads characters from the keyboard
  - Transmitter writes characters to the display

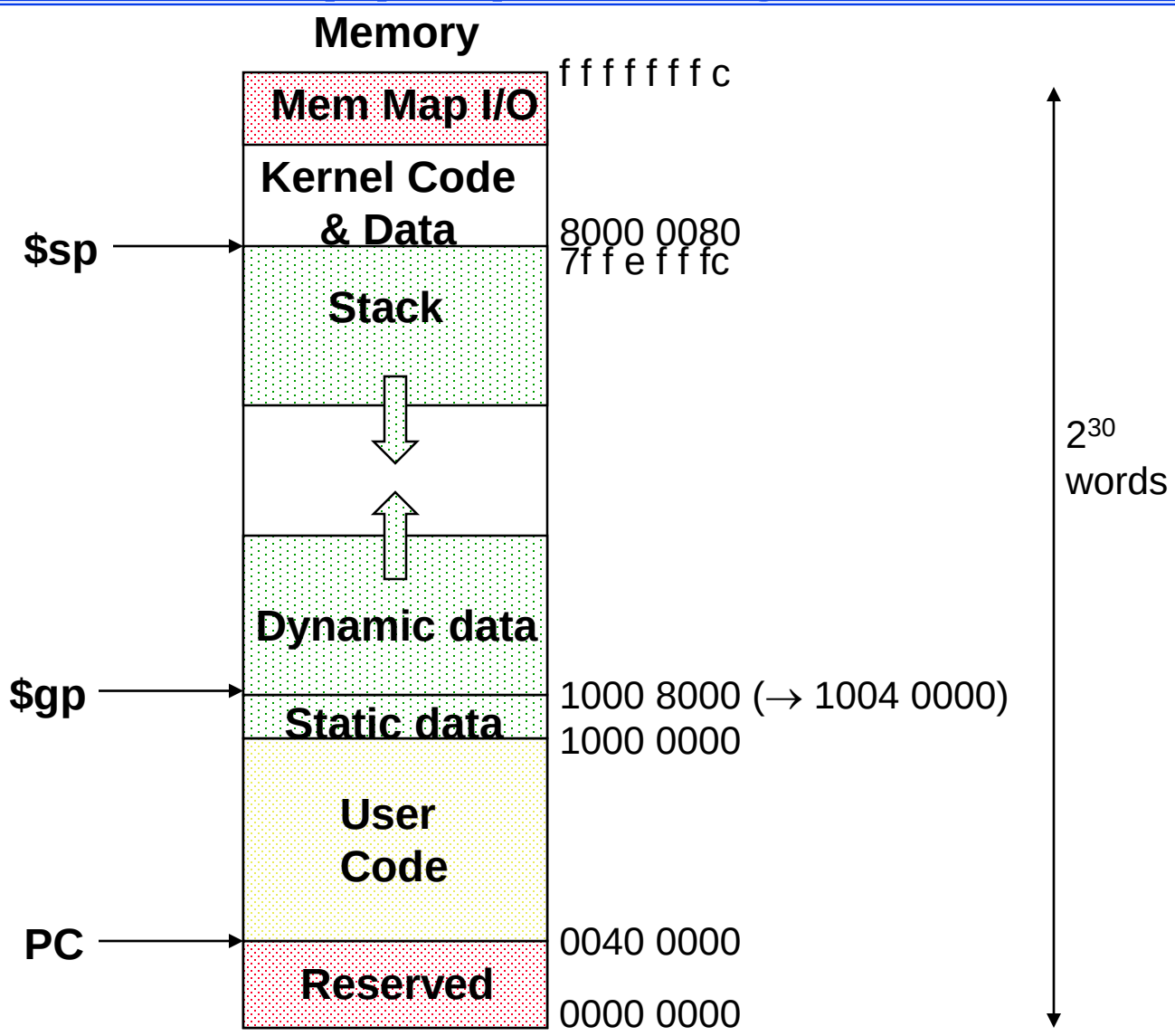


Memory Mapped I/O

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

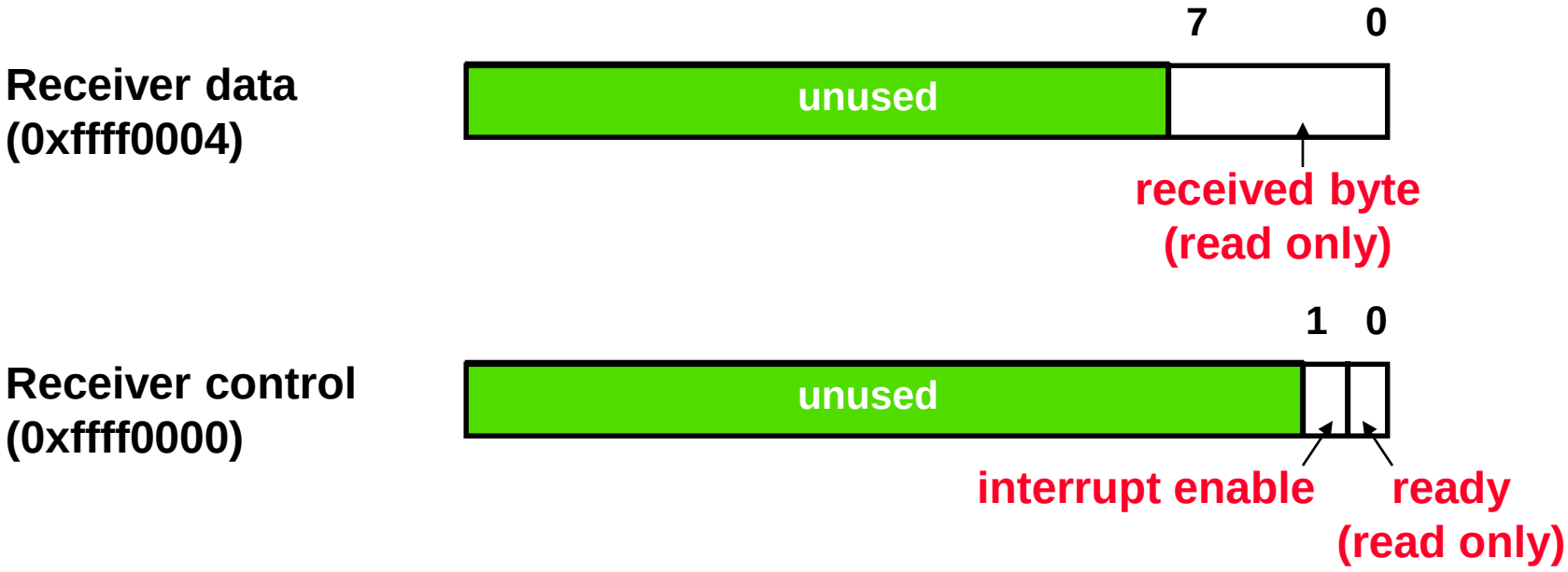


Review: MIPS (spim) Memory Allocation



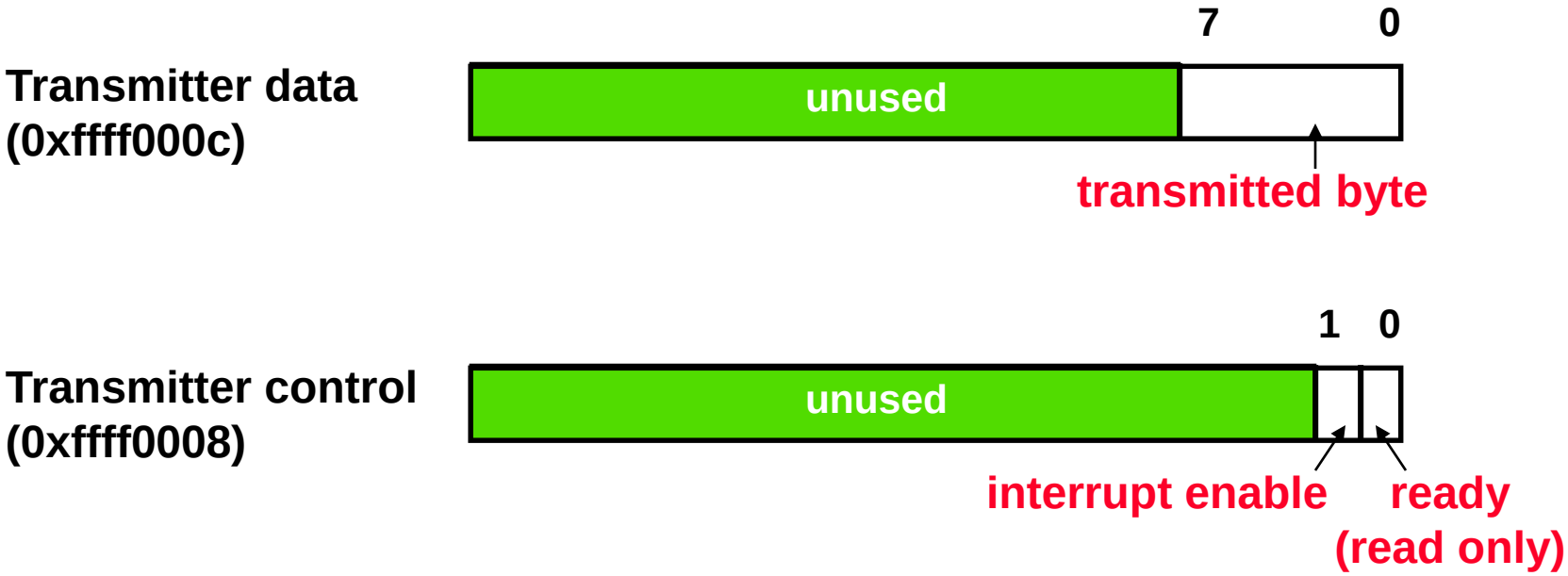
Terminal Receiver (Input) Control with SPIM

- Terminal input is controlled via two memory-mapped device registers (i.e., each register appears as a special memory location)



Terminal Output Control with SPIM

- Terminal output is controlled via two memory-mapped device registers (i.e., each register appears as a special memory location)



Processor-I/O Speed Mismatch

- 1GHz microprocessor can execute 1 billion load or store instructions per second, or 32,000Mb/s data rate
  - I/O devices data rates range from 0.0001 Mb/s to 10,000 Mb/s
- Input: device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- Output: device may not be ready to accept data as fast as processor stores it
- What to do?

Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - 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 $\Rightarrow$ 1) to say its OK
- Processor then loads from (input) or writes to (output) data register
  - Load from or Store into Data Register resets Ready bit (1 $\Rightarrow$ 0) of Control Register

SPIM I/O

- Control register rightmost bit (0): Ready
  - Receiver: Ready==1 means character in Data Register not yet been read;
    1 $\Rightarrow$ 0 when data is read from Data Reg
  - Transmitter: Ready==1 means transmitter is ready to accept a new character;
    0 $\Rightarrow$ Transmitter still busy writing last char
- Data register rightmost byte has data
  - Receiver: last char from keyboard; rest = 0
  - Transmitter: when write rightmost byte, writes char to display

I/O Example

- Input: Read from keyboard into $v0

```
                lui   $t0, 0xffff #ffff0000
Waitloop:       lw    $t1, 0($t0) #control
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop
                lw    $v0, 4($t0) #data
```

- Output: Write to display from $a0

```
                lui   $t0, 0xffff #ffff0000
Waitloop:       lw    $t1, 8($t0) #control
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop
                sw    $a0, 12($t0) #data
```

- "Ready" bit is from processor's point of view!
- Processor waiting for I/O called "Polling"

Cost of Polling?

- Assume for a processor with a 1GHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling
  - Mouse: polled 30 times/sec so as not to miss user movement
  - Floppy disk: transfers data in 2-Byte units and has a data rate of 50 KB/second.
    No data transfer can be missed.
  - Hard disk: transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed.

% Processor time to poll

Mouse Polling [clocks/sec]

= 30 [polls/s] * 400 [clocks/poll] = 12K [clocks/s]

- % Processor for polling:

  $12*10^3$ [clocks/s] / $1*10^9$ [clocks/s] = 0.0012%

  - Polling mouse little impact on processor

Frequency of Polling Floppy

= 50 [KB/s] / 2 [B/poll] = 25K [polls/s]

- Floppy Polling, Clocks/sec

  = 25K [polls/s] * 400 [clocks/poll] = 10M [clocks/s]

- % Processor for polling:

  $10*10^6$ [clocks/s] / $1*10^9$ [clocks/s] = 1%

  - OK if not too many I/O devices

% Processor time to poll hard disk

Frequency of Polling Disk

= 16 [MB/s] / 16 [B/poll] = 1M [polls/s]

- Disk Polling, Clocks/sec
  = 1M [polls/s] * 400 [clocks/poll]
  = 400M [clocks/s]
- % Processor for polling:

  $400*10^6$ [clocks/s] / $1*10^9$ [clocks/s] = 40%

  $\Rightarrow$ Unacceptable

What is the alternative to polling?

- Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
- Would like an unplanned procedure call that would be invoked only when I/O device is ready
- Solution: use exception mechanism to help I/O. Interrupt program when I/O ready, return when done with data transfer

I/O Interrupt

- An I/O interrupt is like overflow exceptions except:
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  - I/O interrupt does not prevent any instruction from completion

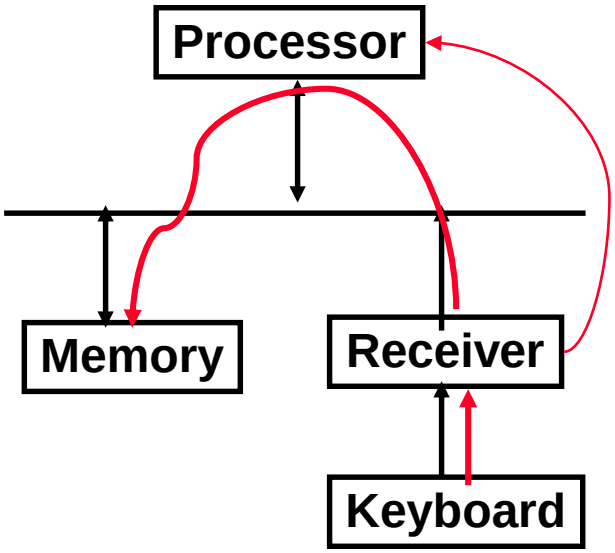
I/O Interrupts

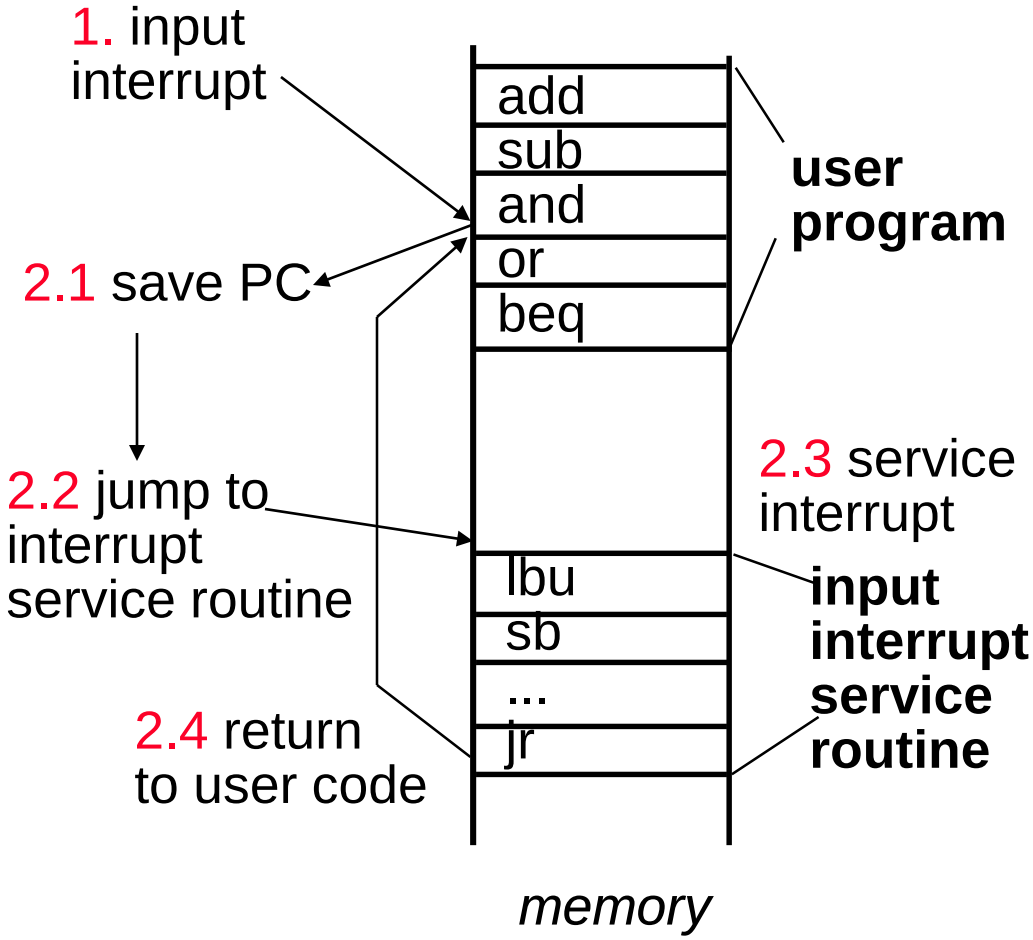
- An I/O interrupt is used to signal an I/O request for service
  - Can have different urgencies (so may need to be prioritized)
  - Need to somehow convey the identity of the device generating the interrupt
- Advantage
  - User program progress is only halted during actual transfer of I/O data to/from user memory space
- Disadvantage - special hardware is needed to
  - Cause an interrupt (I/O device)
  - Detect an interrupt and save the proper information to resume after servicing the interrupt (processor)

Benefit of Interrupt-Driven I/O

- Find the % of processor consumed if the hard disk is only active 5% of the time. Assuming 500 clock cycle overhead for each transfer, including interrupt:
  - Disk Interrupts/s = 16 [MB/s] / 16 [B/interrupt]
    = 1M [interrupts/s]
  - Disk Interrupts [clocks/s]
    = 1M [interrupts/s] * 500 [clocks/interrupt]
    = 500,000,000 [clocks/s]
  - % Processor for during transfer:
    $500*10^6$ [clocks/s] / $1*10^9$ [clocks/s] = 50%
- Disk active 5% $\Rightarrow$ 5% * 50% $\Rightarrow$ 2.5% busy

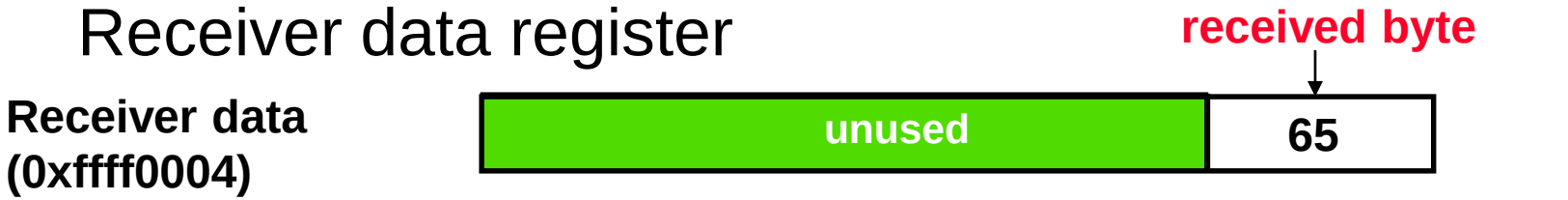
Interrupt Driven Input





Interrupt Driven Input in SPIM

1. the Receiver indicates with an interrupt that it has input a new character from the keyboard into the Receiver data register

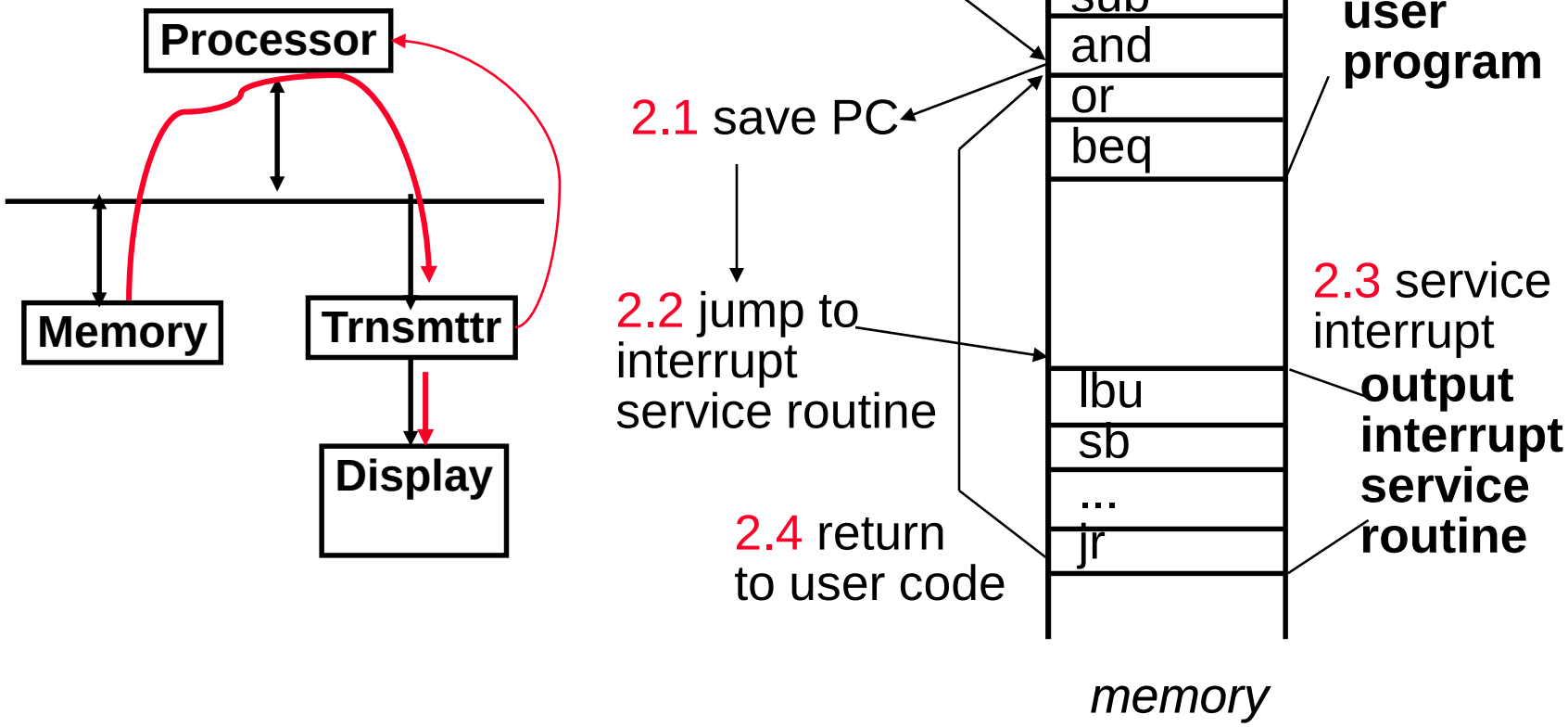


   - writing to the Receiver data register sets the Receiver control register ready bit to 1

2. the user process responds to the interrupt by transferring control to an interrupt service routine that copies the input character into the user memory space

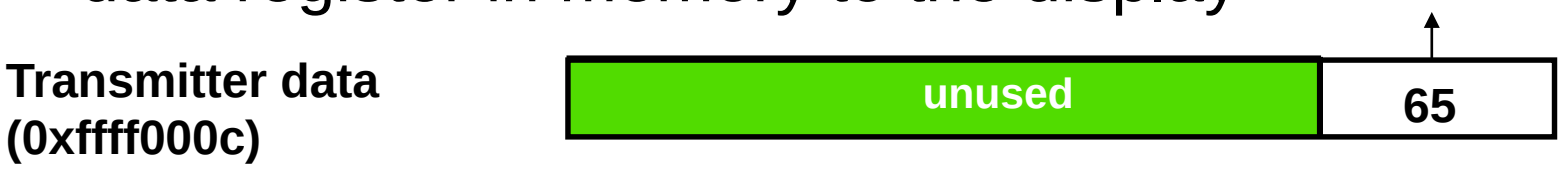
   - reading the Receiver data register resets the Receiver control register ready bit to 0

Interrupt Driven Output



Interrupt Driven Output in SPIM

1. the transmitter indicates with an interrupt that it has successfully output the character in the Transmitter data register in memory to the display

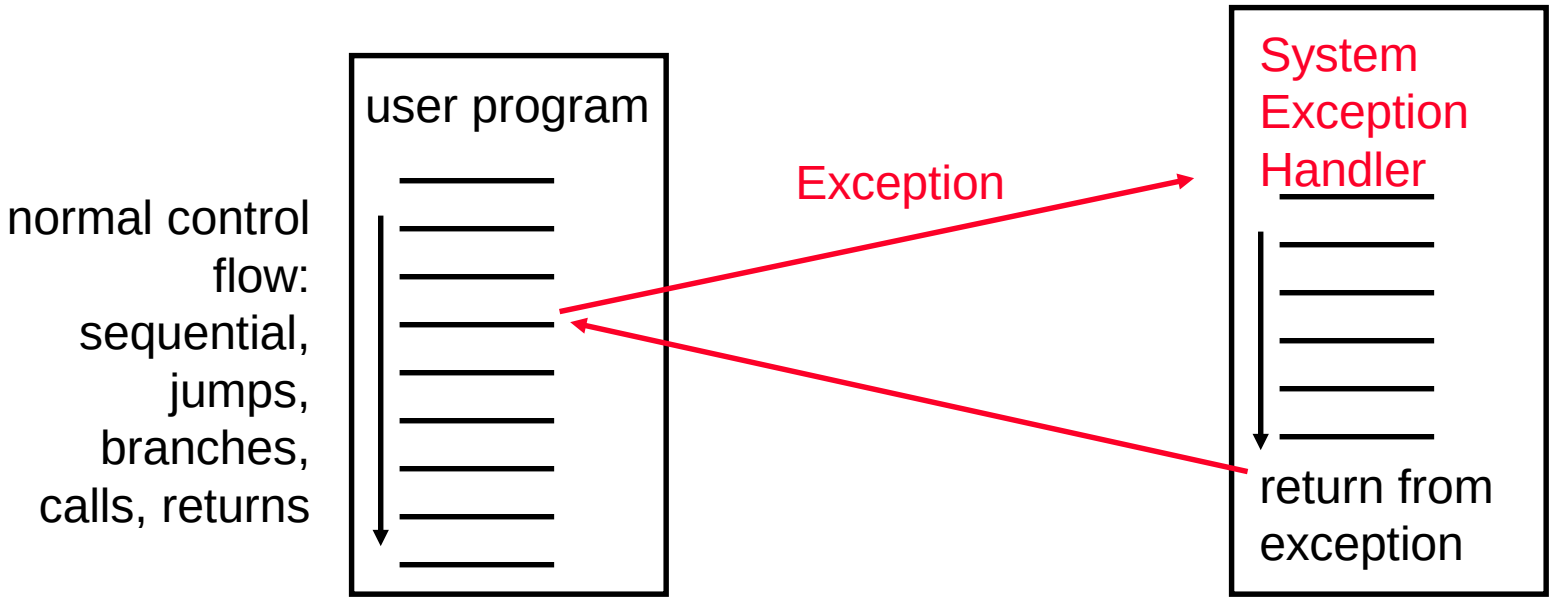


   - reading from the Transmitter data register sets the Transmitter control register ready bit to 1

2. the user process responds to the interrupt by transferring control to an interrupt service routine that writes the next character to output from the user memory space into the Transmitter data register

   - writing to the Transmitter data register resets the Transmitter control register ready bit to 0

Exceptions in General



- Exception = unprogrammed control transfer
  - system takes action to handle the exception
    - must record the address of the offending or next to execute instruction and save (and restore) user state
  - returns control to user after handling the exception

Two Types of Exceptions

- Interrupts
  - caused by external events (i.e., request from I/O device)
  - **asynchronous** to program execution
  - may be handled between instructions
  - simply suspend and resume user program

- Traps (Exception)
  - caused by internal events
    - exceptional conditions (e.g., arithmetic overflow, undefined instr.)
    - errors (e.g., hardware malfunction, memory parity error)
    - faults (e.g., non-resident page – page fault)
  - **synchronous** to program execution
  - condition must be remedied by the trap handler
  - instruction may be retried and program continued or program may be aborted

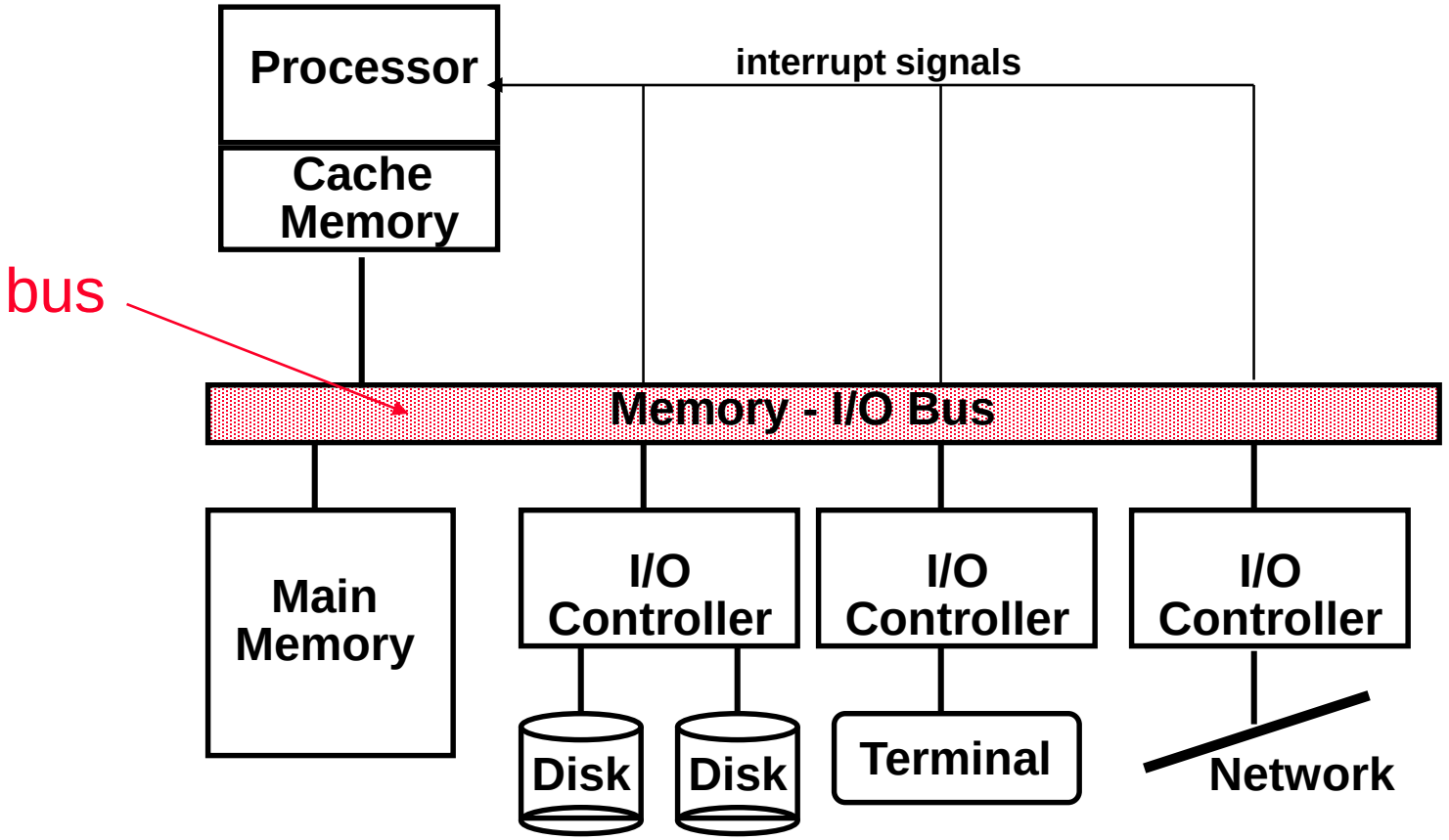
Additions to MIPS ISA for I/O Interrupts

- Control signals to write EPC, Cause, and Status
- Hardware to record the type of interrupt in Cause
- Modify the Datapath and Control, so that
  - the address of interrupt handler ($8000\ 0080_{hex}$) can be loaded into the PC, so must increase the size of PC multiplex
  - and save the address of the current/next instr in EPC

Example I/O Interrupt Service Routine

```
            .ktext 0x80000080
            sw     $t0, save0          #save working reg's
            . . .
            mfc0   $k0, $13            #move Cause into $k0
            mfc0   $k1, $14            #move EPC into $k1
            sgt    $t0, $k0, 0x00      #is it an interrupt?
            bgtz   $t0, excpt
intrpt:     li     $t0, 0xffff0000
            lw     $t1, 0($t0)         #test Receiver control
            beq    $t1, $0, tstout     #  for input ready
inpt:       lb     $v0, 4($t0)         #input char. into $v0
tstout:     lw     $t1, 8($t0)         #test Trans. control
            beq    $t1, $0, excpt      #  for output ready
            sb     $v1, c($t0)         #output char. from $v1
            j      done                #
excpt:      . . .                      #code to handle excpts
            addiu  $k1, $k1, 4         #do not re-execute
                                       #  faulting instr
done:       lw     $t0, save0          #restore working reg's
            . . .
            eret                       #return to user code (EPC)
            .kdata
save0:      .word  0
            . . .
```

I/O System Interconnect Issues



- Usually have more than one I/O device in the system connected to the processor via a bus
  - each I/O device is controlled by an I/O Controller

Buses

- A bus is a shared communication link (a single set of wires used to connect multiple subsystems) that needs to support a range of devices with widely varying latencies and data transfer rates
  - Advantages
    - Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
    - Low cost – a single set of wires is shared in multiple ways
  - Disadvantages
    - Creates a communication bottleneck – bus bandwidth limits the maximum I/O throughput
- The maximum bus speed is largely limited by
  - The length of the bus
  - The number of devices on the bus

I/O Performance Measures

- I/O bandwidth (throughput) – amount of information that can be input (output) and communicated across an interconnect (e.g., a bus) to the processor/memory (I/O device) per unit time
  1. How much data can we move through the system in a certain time?
  2. How many I/O operations can we do per unit time?

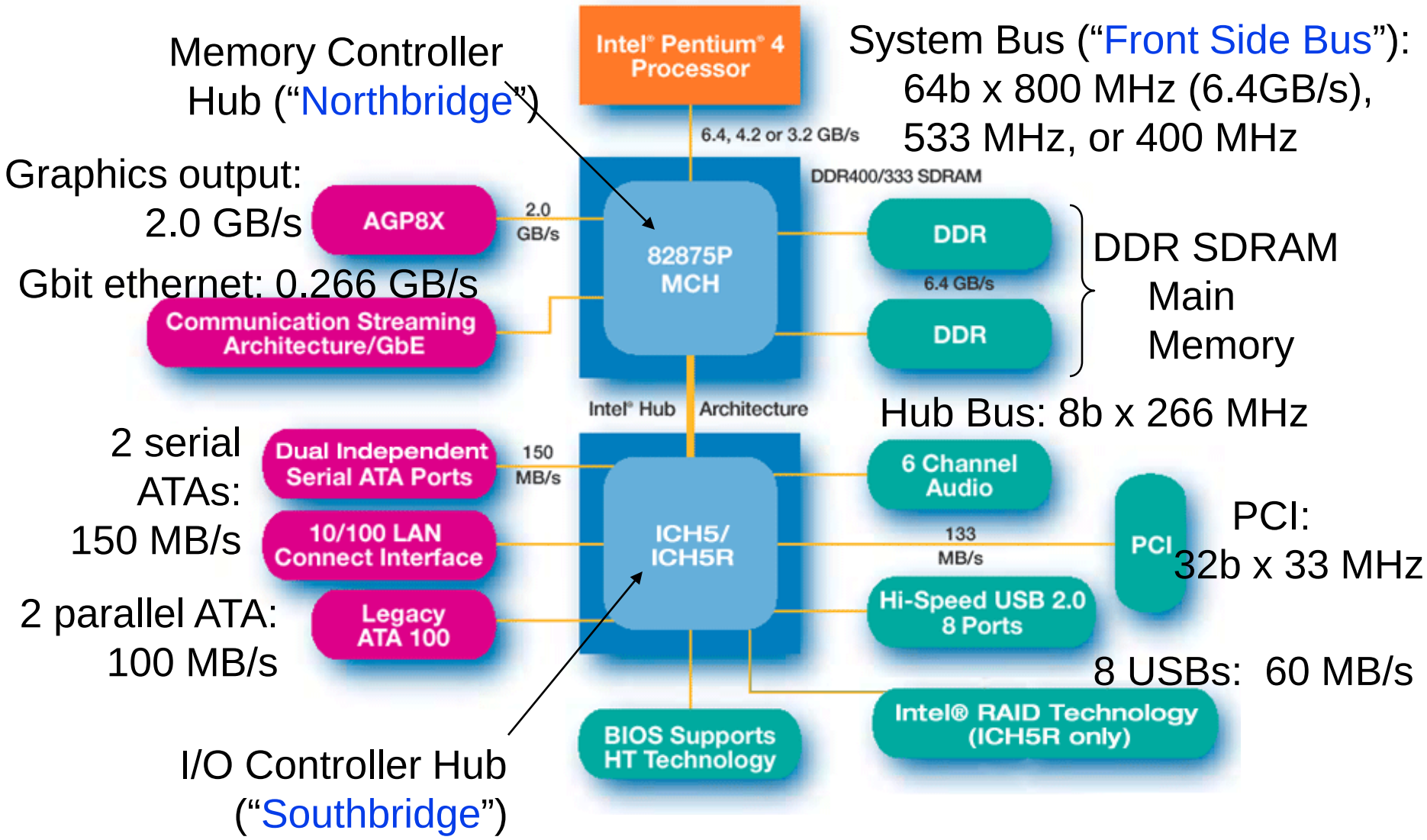
- I/O response time (latency) – the total elapsed time to accomplish an input or output operation
  - An especially important performance metric in real-time systems

- Many applications require both high throughput and short response times

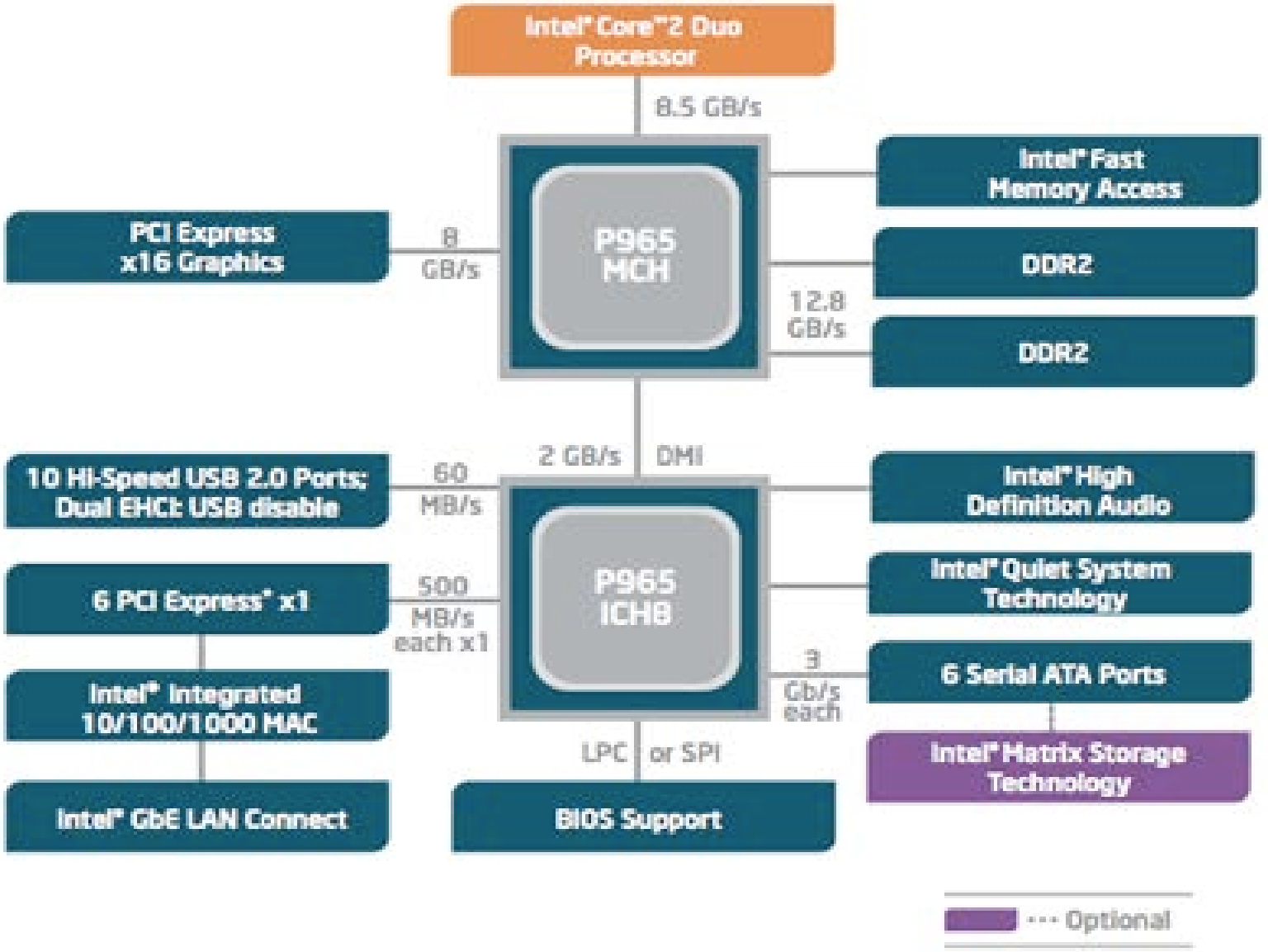
Types of Buses

- Processor-memory bus (proprietary)
  - Short and high speed
  - Matched to the memory system to maximize the memory-processor bandwidth
  - Optimized for cache block transfers
- I/O bus (industry standard, e.g., SCSI, USB, Firewire)
  - Usually is lengthy and slower
  - Needs to accommodate a wide range of I/O devices
  - Connects to the processor-memory bus or backplane bus
- Backplane bus (industry standard, e.g., ISA, PCI, PCI Express)
  - The backplane is an interconnection structure within the chassis
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus

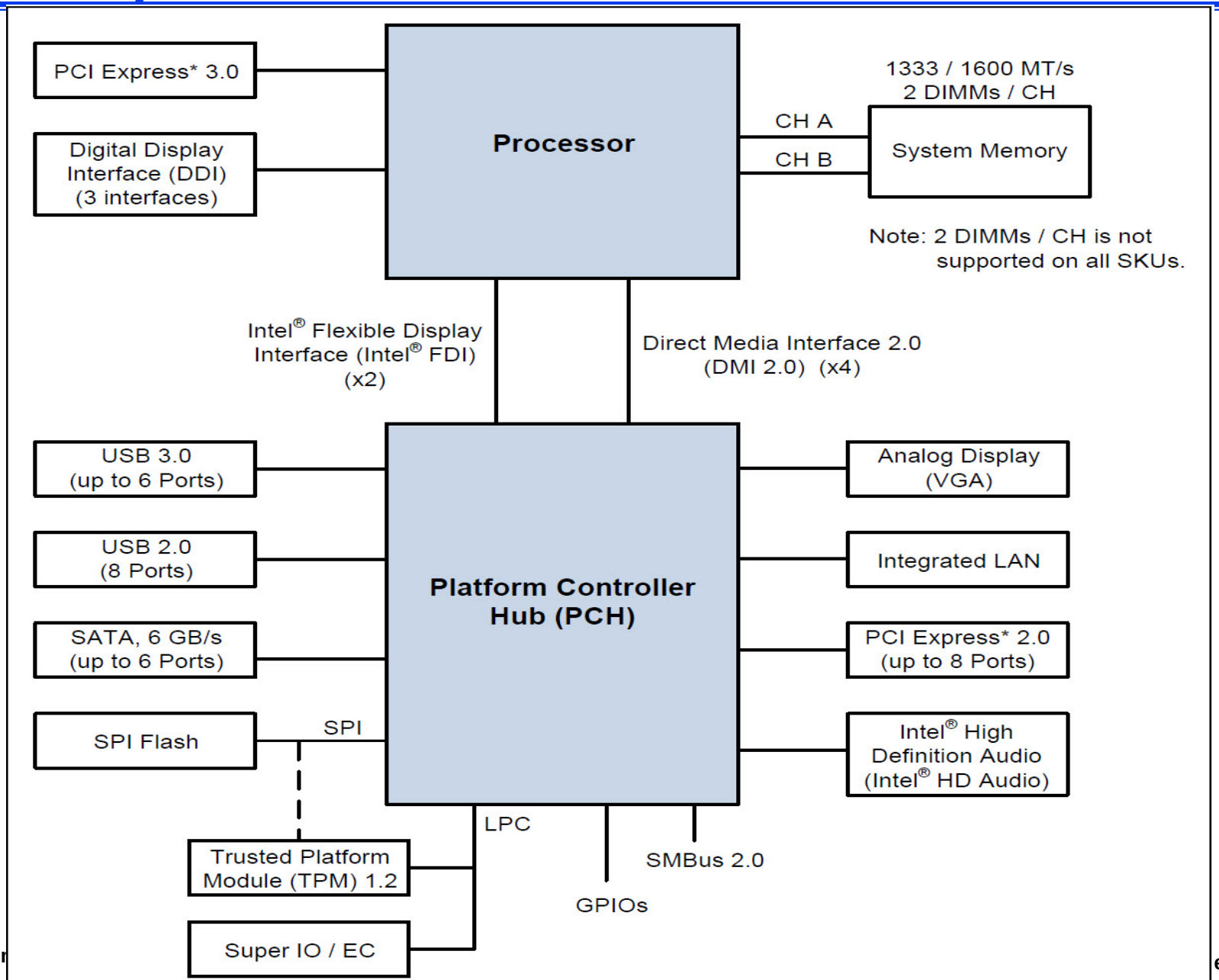
Example: The Pentium 4's Buses



Example: The Pentium Core 2 Duo Buses



Example: The Intel 4th Gen Core i7



Bus Bandwidth Determinates

- The bandwidth of a bus is determined by
  - Whether its is synchronous or asynchronous and the timing characteristics of the protocol used
  - The bus width (i.e., number of data lines)
  - Whether the bus supports block transfers or only word at a time transfers

	Firewire	USB 2.0
Type	I/O	I/O
Data lines	4	2
Architecture	Peer-to-peer	Master-slave
Max # devices	63	127
Max length	4.5 meters	5 meters
Peak bandwidth	400 Mbps , 800 Mbps 1600 Mbps, 3200 Mbps	1.5 Mb/s (low), 12 Mb/s (full), 480 Mb/s (high), 5Gbps (super)

Buses in Transition

- Companies are transitioning from synchronous, parallel, *wide* buses to asynchronous *narrow* buses
  - Reflection on wires and clock skew makes it difficult to use 16 to 64 parallel wires running at a high clock rate (e.g., ~400 MHz) so companies are transitioning to buses with a few one-way wires running at a very high “clock” rate (~2 GHz)

	PCI	PCI Express x1	ATA	Serial ATA
Total # wires	120	36	80	7
Clock (MHz)	33 – 133	16000	50	16000
Peak BW (MB/s)	128 – 1064	1969 (v4.0)	100	1969 (rev. 3.2)

Buses in Transition