Top-Level View of a Computer

Lecture Content

- **Instruction Cycle**
- **Interrupts**
- **I/O Functionality**
- **Bus Systems**
- **QPI**
- **PCIe**

Top-Level View of a Computer

Learning Objectives

- Understand the basic elements of an instruction cycle and the role of interrupts
- Understand the way different components are connected in a computer
- Understand the different types of bus systems
- Understand the need for multiple buses and the arrangement of them in a hierarchy

What it Really Means

- We have learned so far how to program a computer and the basic principles of computer arithmetic and boolean algebra

- **Now we want to see how everything works together:**
  - What components are there?
  - How are they connected?
  - How does a computer execute an instruction?

- **Then we start digging deeper into the topic**
  - Memory hierarchies
  - CPU organization
  - CPU details

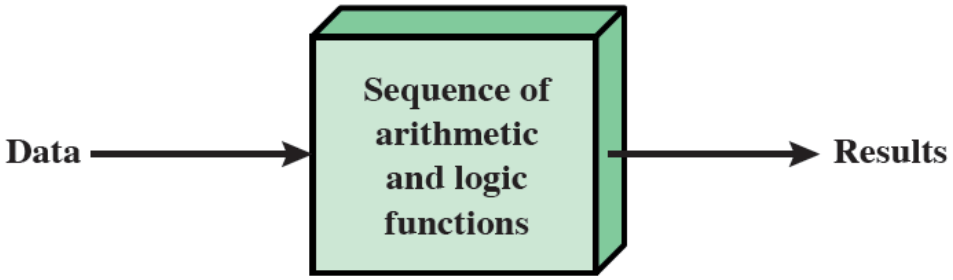
Computer Components

- Contemporary computer designs are based on concepts developed by John von Neumann

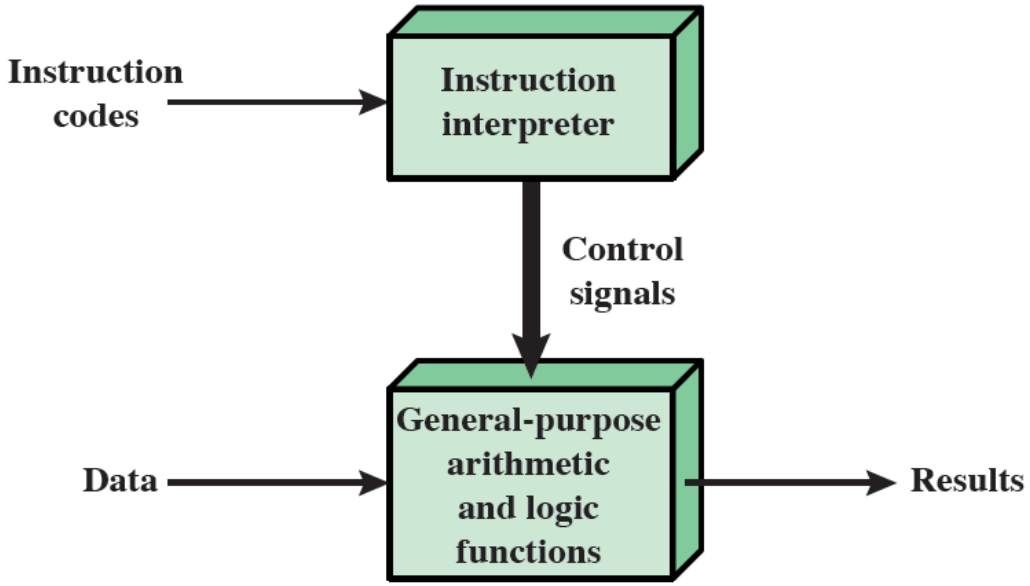
- **Based on three key concepts**
  - Data and instructions are stored in a single read-write memory
  - The contents of this memory are addressable by location, without regard to the type of data contained there
  - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next

- **Hardwired program**
  - The result of the process of connecting the various components in the desired configuration

General-Purpose vs. Hardwired



(a) Programming in hardware



(b) Programming in software

The Top-Level View

- **Software**
  - A sequence of codes or instructions
  - Is interpreted by hardware
  - Much easier than rewiring the hardware ;-)

- **Major components**
  - CPU
    - Instruction interpreter
    - Module of general-purpose arithmetic and logic functions
    - I/O Components
  - Input module
    - Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
  - Output module
    - Means of reporting results

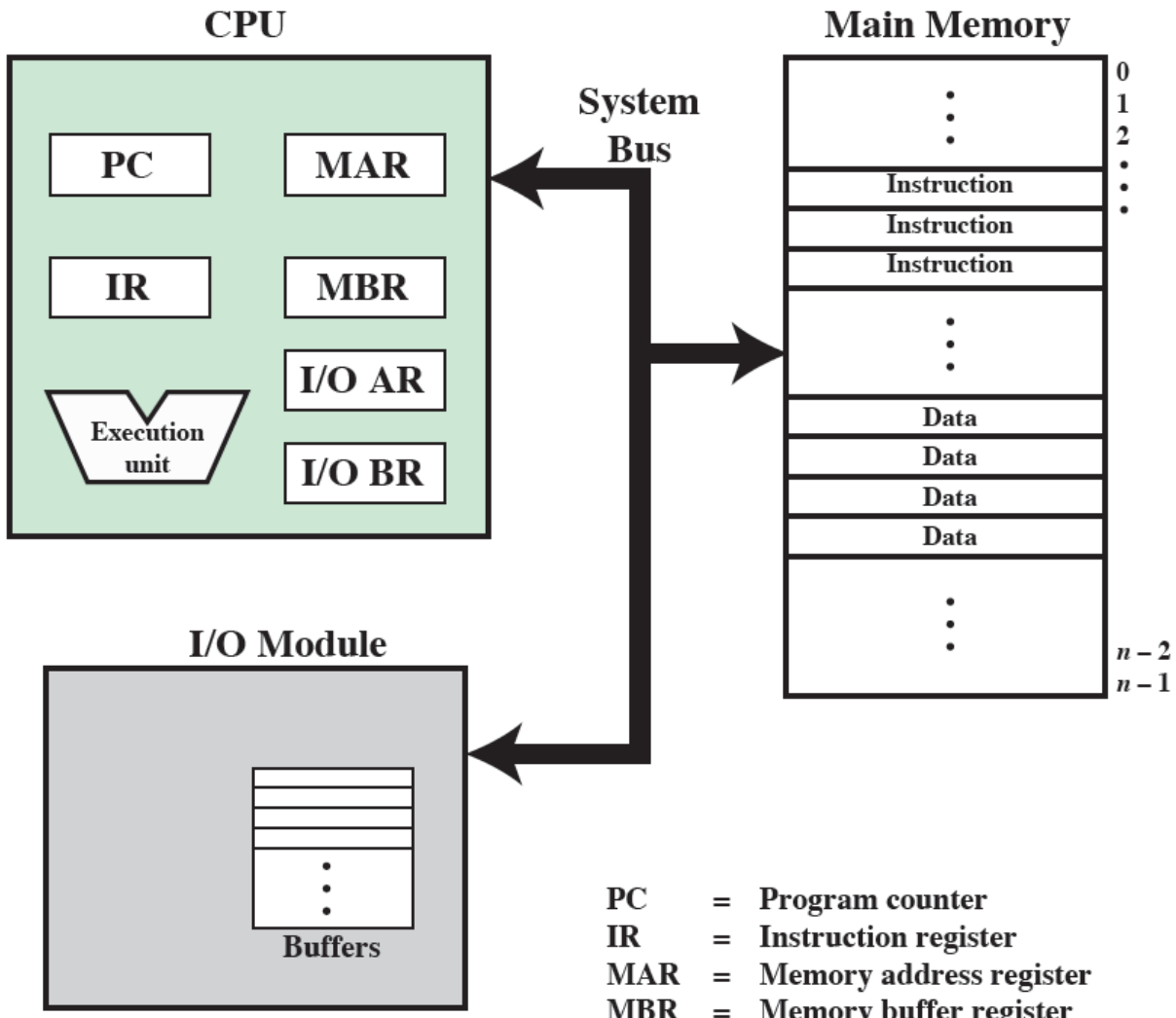
Lets Build a Minimalistic Computer

- First lets have a look at the CPU

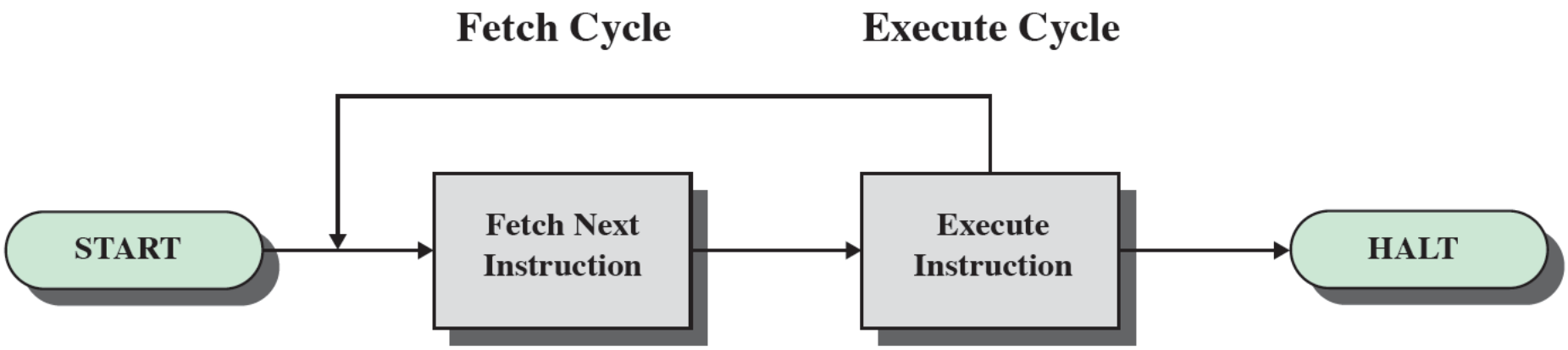
- We definitely need
  - A **Program Counter** (**PC**) register pointing to the next instruction
  - A **Instruction Register** (**IR**) holding the current instruction
  - **Memory Address** and **Buffer** register (**MAR** and **MBR**) for fetching information from the memory
  - The same is required for I/O Operations (**I/O AR** and **I/O BR**)

- In the next step we will look how these components work together

Our Minimalistic Computer



How does Our Little Machine Work?



- The processing required for a single instruction is called an **instruction cycle**.
- In this simple form, a instruction cycle consists of:
  - Fetch Cycle
  - Execute Cycle

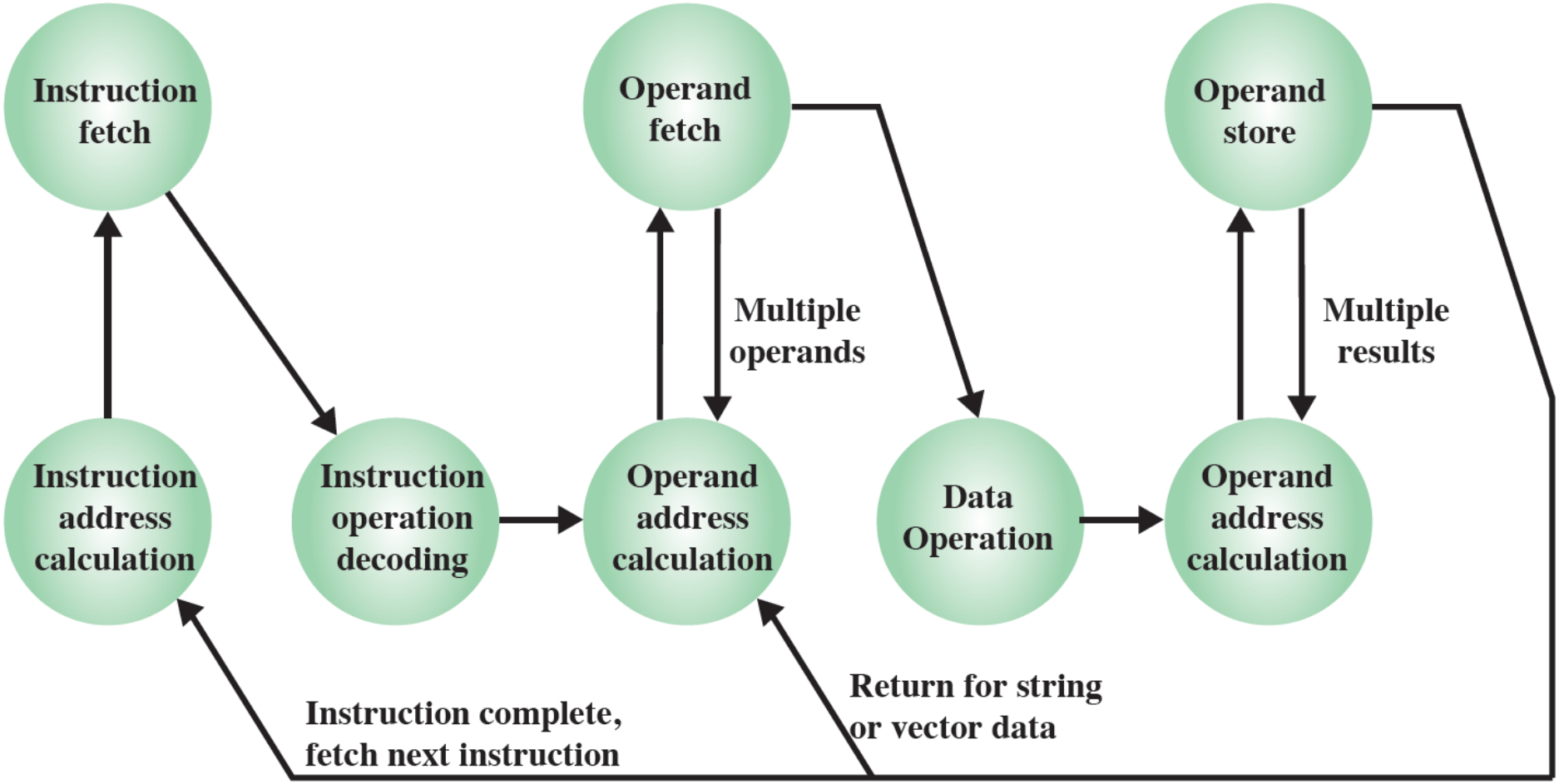
The Fetch Cycle

- At the beginning of each instruction cycle the processor fetches an instruction from memory
- The program counter (PC) holds the address of the instruction to be fetched next
- The processor increments the PC after each instruction fetch
- The fetched instruction is loaded into the instruction register (IR)
- The processor interprets the instruction and performs the required action

Are We Happy With That?

- Is this structure sufficient?
- Some thoughts:
  - Often, we are having several operands
  - What about indirect addressing
  - What about control flow changes
- These considerations require changes in our Instruction Cycle

Advanced Instruction Cycle



Top-Level View of a Computer

Lecture Content

- Instruction Cycle
- Interrupts
- I/O Functionality
- Bus Systems
- QPI
- PCIe

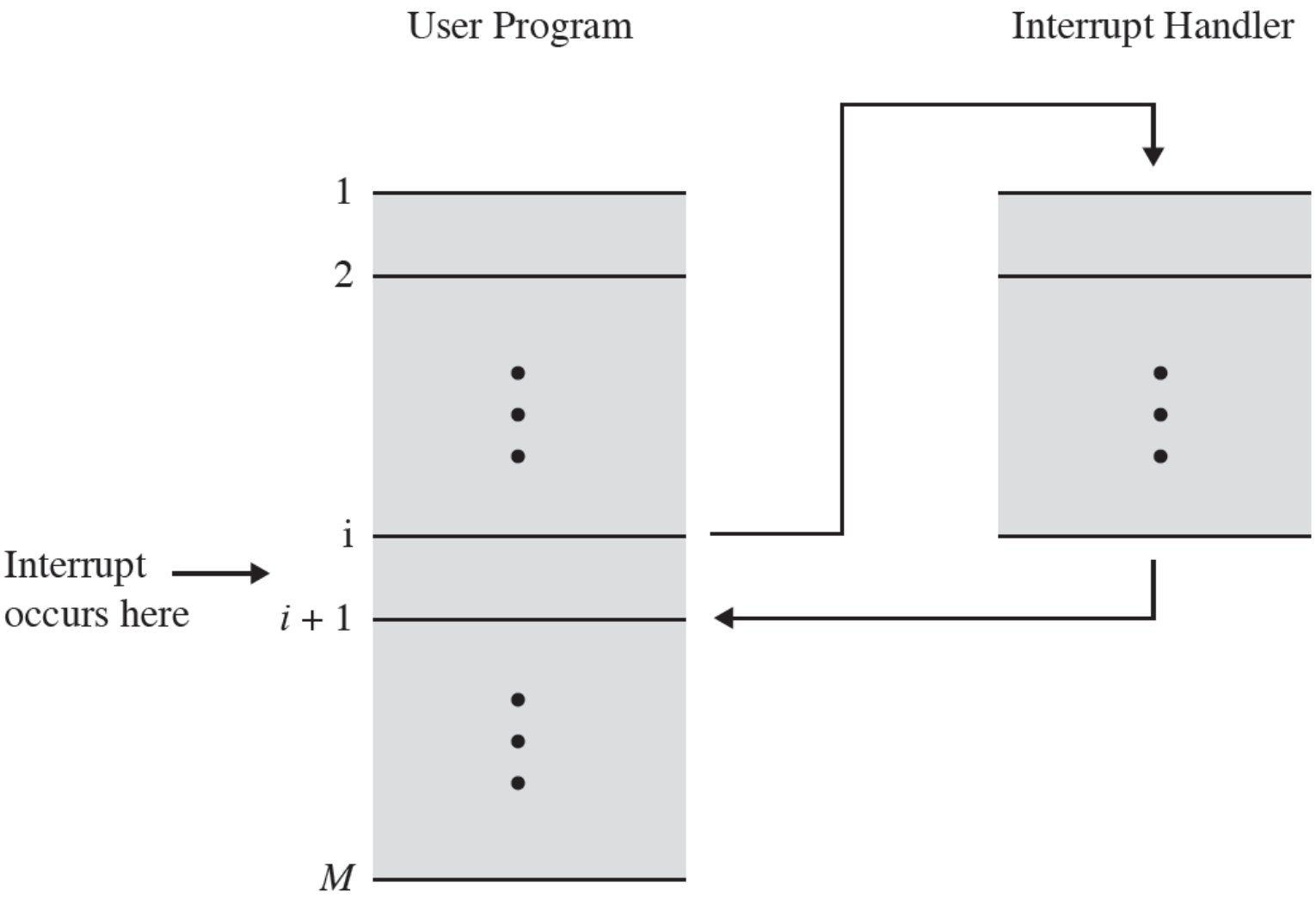
Further Mechanisms: Interrupts

- Virtually all computers provide a mechanism by which other modules may interrupt the normal processing
- We will hear more about interrupts later, for now we just need to know about the concepts
- We distinguish between various types of interrupts

Types of Interrupts

- **Program**
  - Result of an instruction execution
    - arithmetic overflow, division by zero
    - attempt to execute an illegal machine instruction
    - reference outside a user's allowed memory space
- **Timer**
  - Generated by a timer within the processor. Used by the OS to perform certain functions on a regular basis, e.g., scheduling
- **I/O**
  - Generated by an I/O controller
    - signal normal completion of an operation
    - variety of error conditions
- **Hardware failure**
  - Generated by a failure such as power failure or memory parity error.

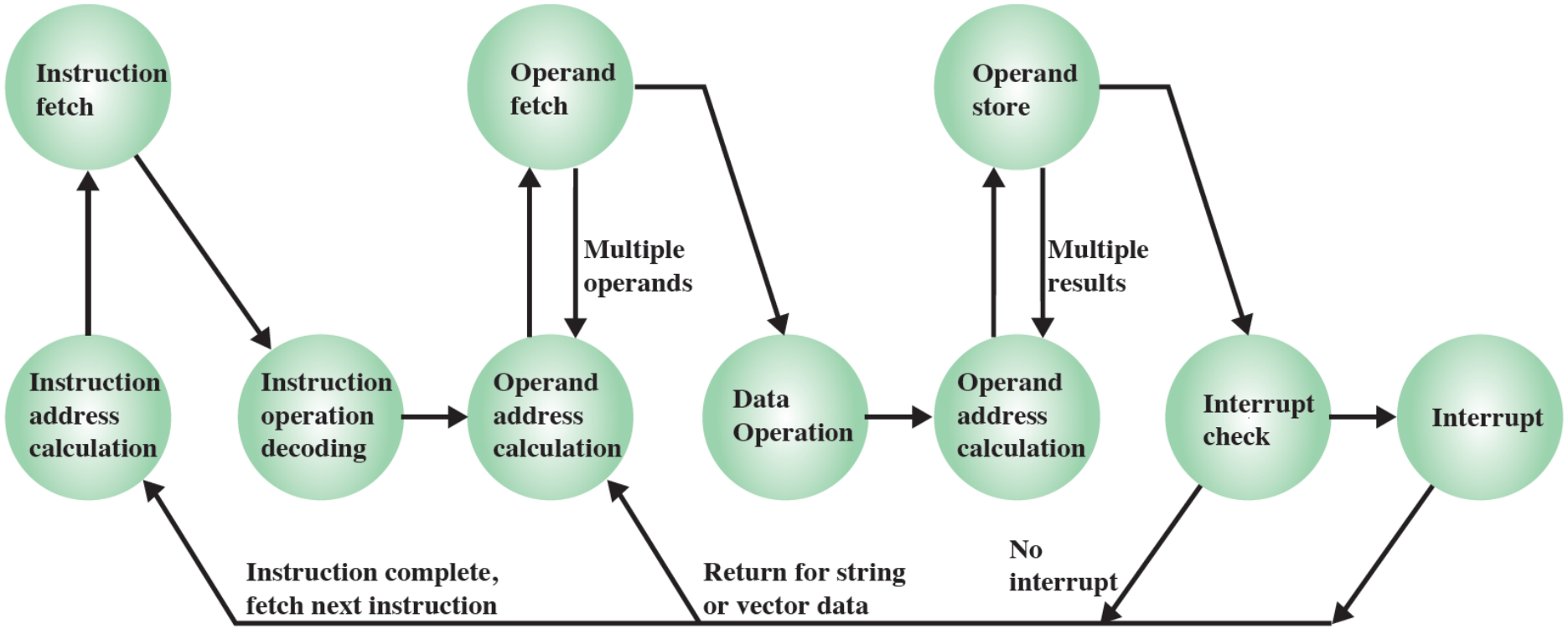
How Does an Interrupt Work?



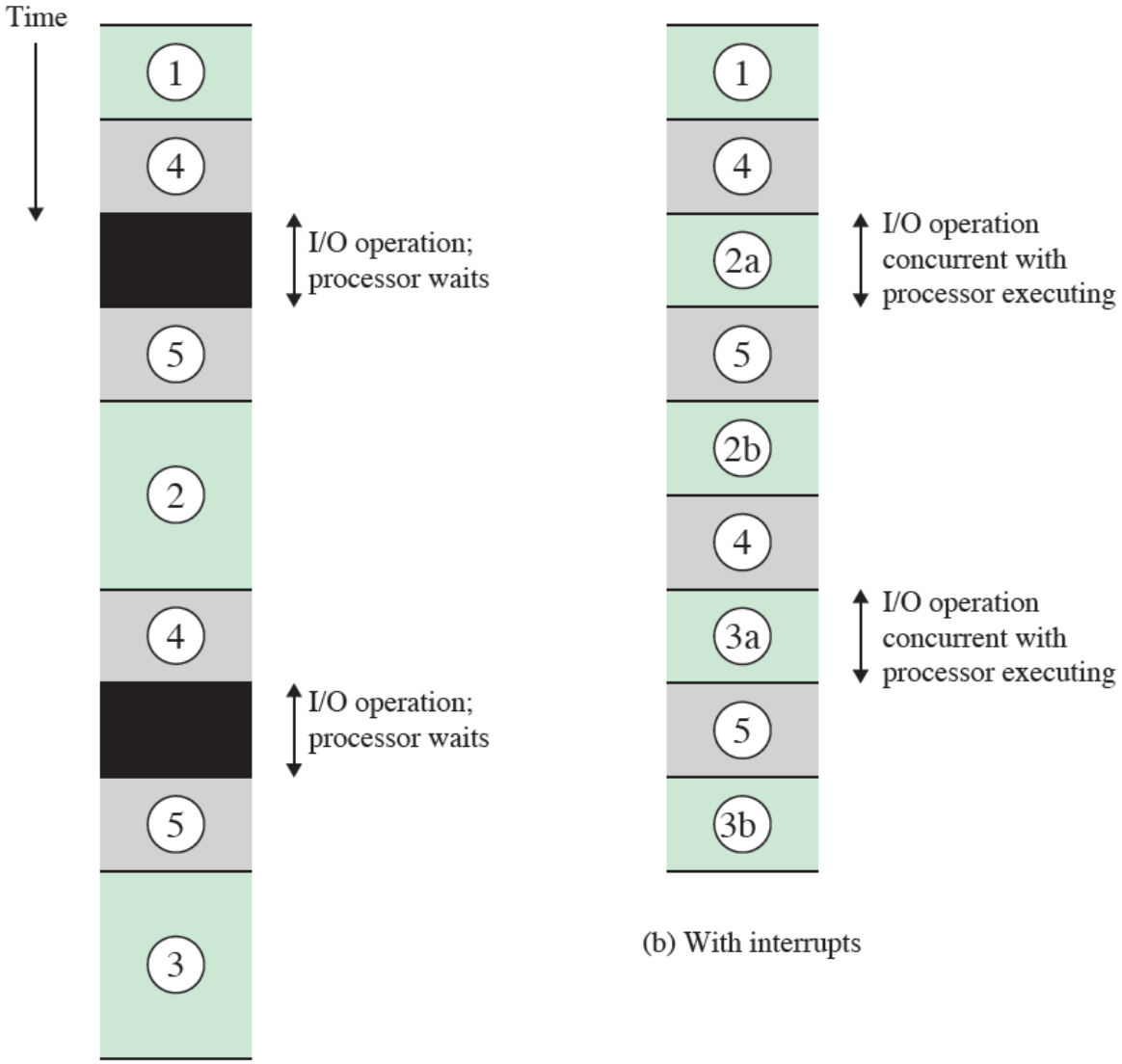
Why Interrupts?

- Interrupts are provided primarily as a way to improve processing efficiency
- For example, most external devices are much slower than the processor
- Suppose that the processor always has to wait until it can finish the I/O operation and go to the next instruction
- Clearly, this is a very wasteful use of the processor.

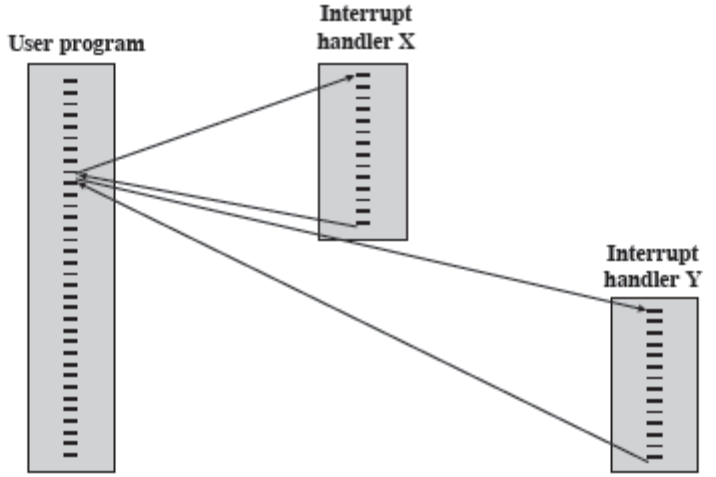
Instruction Cycle State Diagram With Interrupts



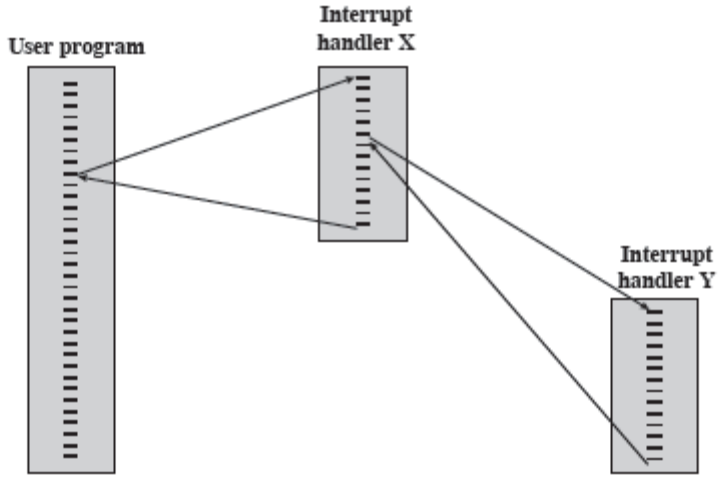
Time Savings With Interrupts



Two Ways of Handling Multiple Interrupts



(a) Sequential interrupt processing



(b) Nested interrupt processing

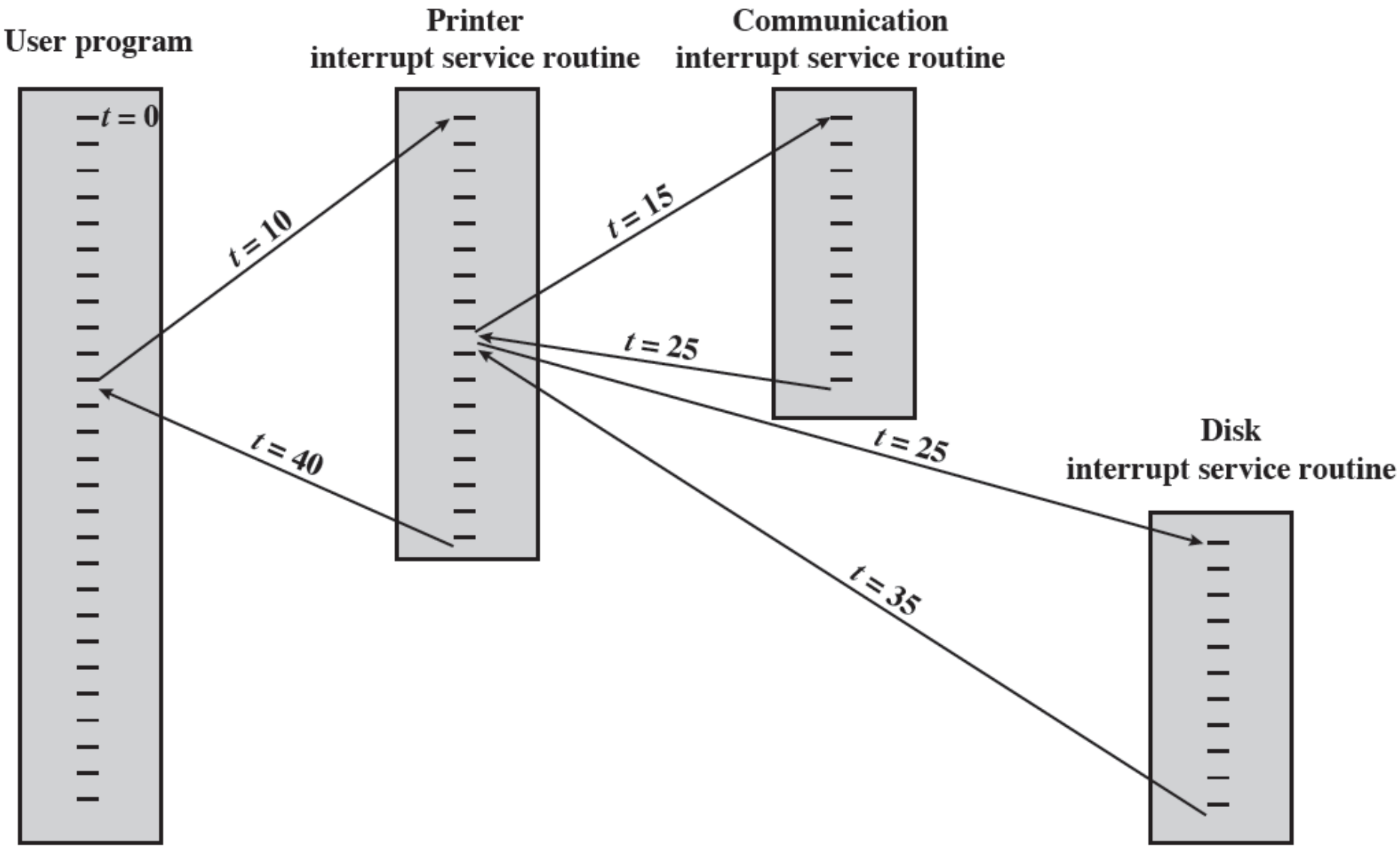
Handling Multiple Interrupts: Sequential

- Interrupts have to wait until the first Interrupt was finished; thus, during one interrupts, others are **disabled**
- Doesn`t consider time criticality of certain interrupts
- With a printer we don’t care
- With communication lines indicating new data, we do
  - Buffers may overflow, no more data can be received
- Generally, we want the system to be responsive

Prioritize Interrupts

- Normally, Interrupts are prioritized
- Interrupts with a higher priority may interrupt ongoing interrupts of lower priority
- Higher maintenance for the machine
- Small Example:
  - We have printer with priority 0 (t=10)
  - A hard disk with priority 4 (t=20)
  - A network with priority 5 (t=15)

Handling Multiple Interrupts



Top-Level View of a Computer

Lecture Content

- Instruction Cycle
- Interrupts
- I/O Functionality
- Bus Systems
- QPI
- PCIe

I/O Functionality

- I/O module can exchange data directly with the processor
- Processor can read data from or write data to an I/O module
  - This happens using special I/O instructions rather than memory referencing instructions

- In some cases it is desirable to allow I/O exchanges to occur directly with memory
  - The processor grants to an I/O module the authority to manipulate part of the memory
  - The I/O module issues read or write commands to memory relieving the processor of responsibility for the exchange
  - This operation is known as direct memory access (DMA)

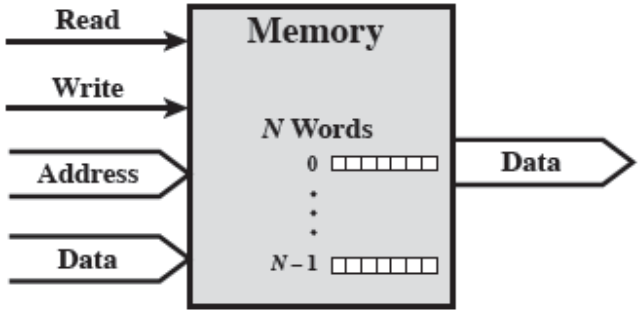
What Needs to be Interchanged?

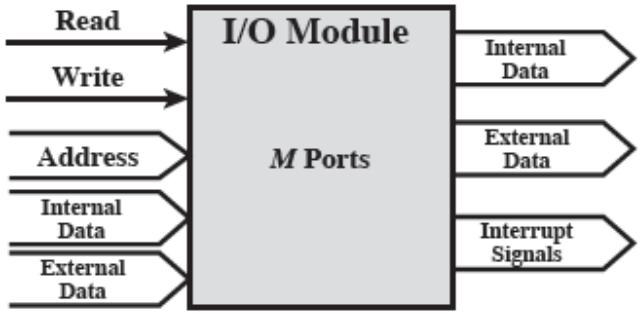
- **Memory to processor**:
  - The processor reads an instruction or a unit of data from memory.
- **Processor to memory**:
  - The processor writes a unit of data to memory.

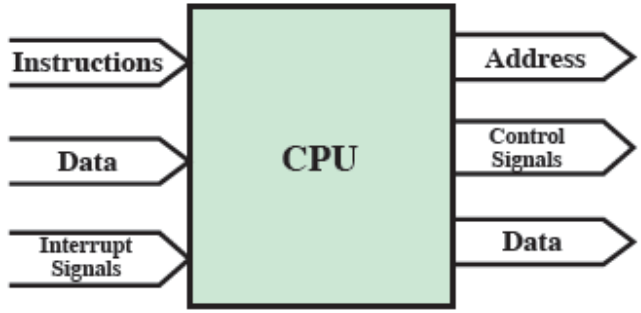
- **I/O to processor**:
  - The processor reads data from an I/O device via an I/O module.
- **Processor to I/O**:
  - The processor sends data to the I/O device.

- **I/O to or from memory**:
  - For these two cases, an I/O module is allowed to exchange data directly with memory (DMA)

The Computer as a Set of Components







Top-Level View of a Computer

Lecture Content

- Instruction Cycle
- Interrupts
- I/O Functionality
- Bus Systems
- QPI
- PCIe

Connecting these Modules: BUS

- A communication pathway connecting two or more devices
  - Key characteristic is that it is a shared transmission medium
- Signals from on device are visible to all devices attached to the bus
  - Two Senders: Garbage
- Typically consists of multiple communication lines
- Computers has several busses for various levels of the computer system hierarchy
- **System bus**
  - A bus that connects major computer components (processor, memory, I/O)

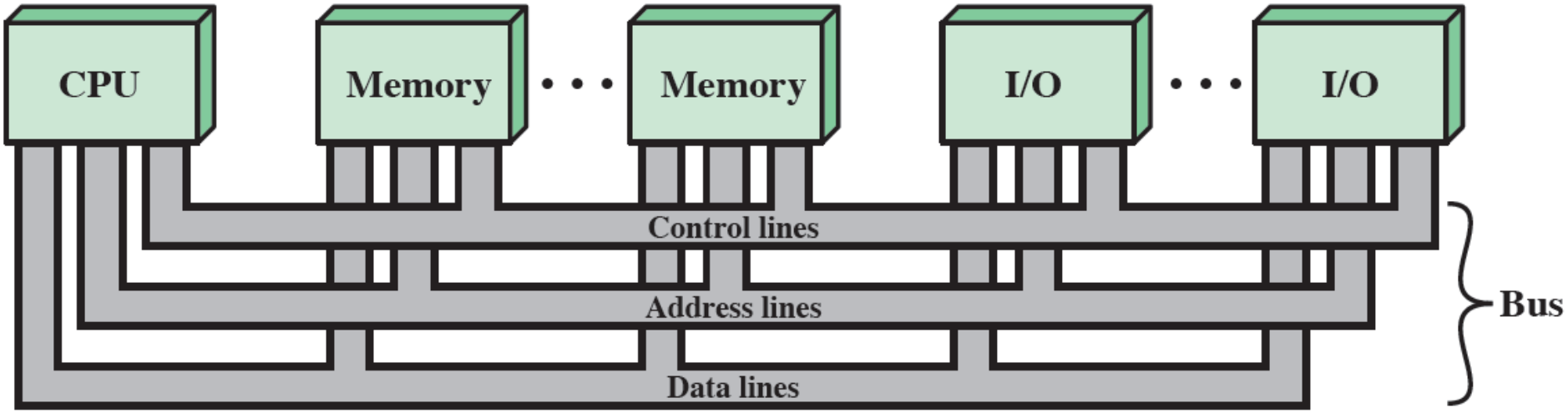
Data Bus

- Data lines that provide a path for moving data among system modules
- May consist of 32, 64, 128, or more separate lines
- The number of lines is referred to as the width of the data bus
- This basically limits the amount of data which can be transferred at a time
  - The width of the data bus is a key factor in determining overall system performance
  - But there are more points to performance than the bus width

Control Bus

- Because the data and address lines are shared by all components there must be a means of controlling their use
- Control signals transmit both command and timing information among system modules
- Timing signals indicate the validity of data and address information
- Command signals specify operations to be performed
- Typical control lines include:
  - Memory or I/O read/write
  - Transfer ACK
  - Bus request/grant
  - Interrupt request/ACK
  - Clock
  - Reset

A Typical Bus Layout



Central Elements of a Bus Design

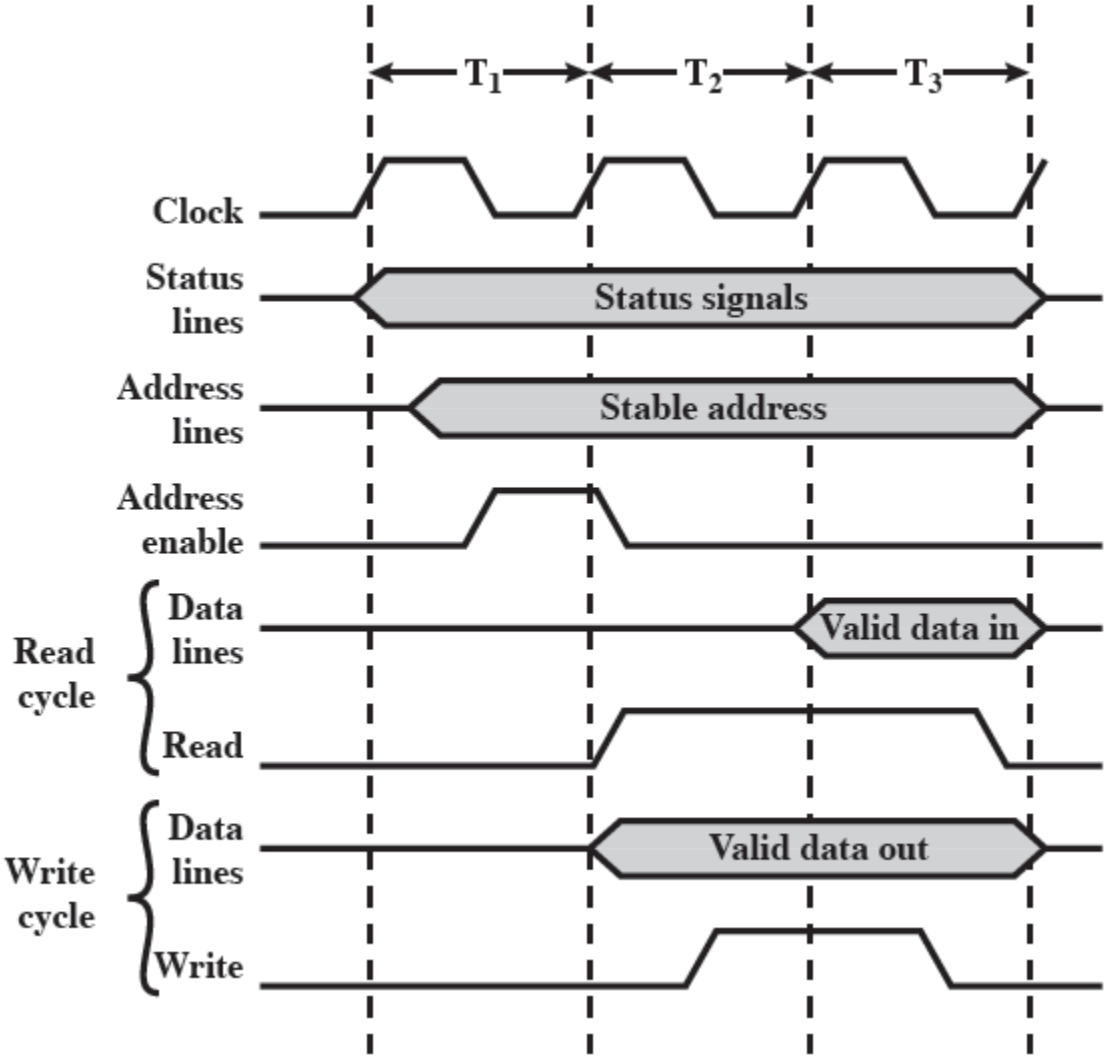
- **Bus Line Types**
  - **Dedicated:**
    - Such a bus line is always assigned to one function or component
    - E.g.: Permanent control line or address lines
  - **Multiplexed:**
    - Fewer lines
    - More complex, less parallelism

- **Method of Arbitration**
  - **Centralized Arbitration:**
    - A single hardware device (arbiter) is responsible for allocating time on the bus
  - **Distributed Arbitration:**
    - The use of the bus is negotiated between all modules

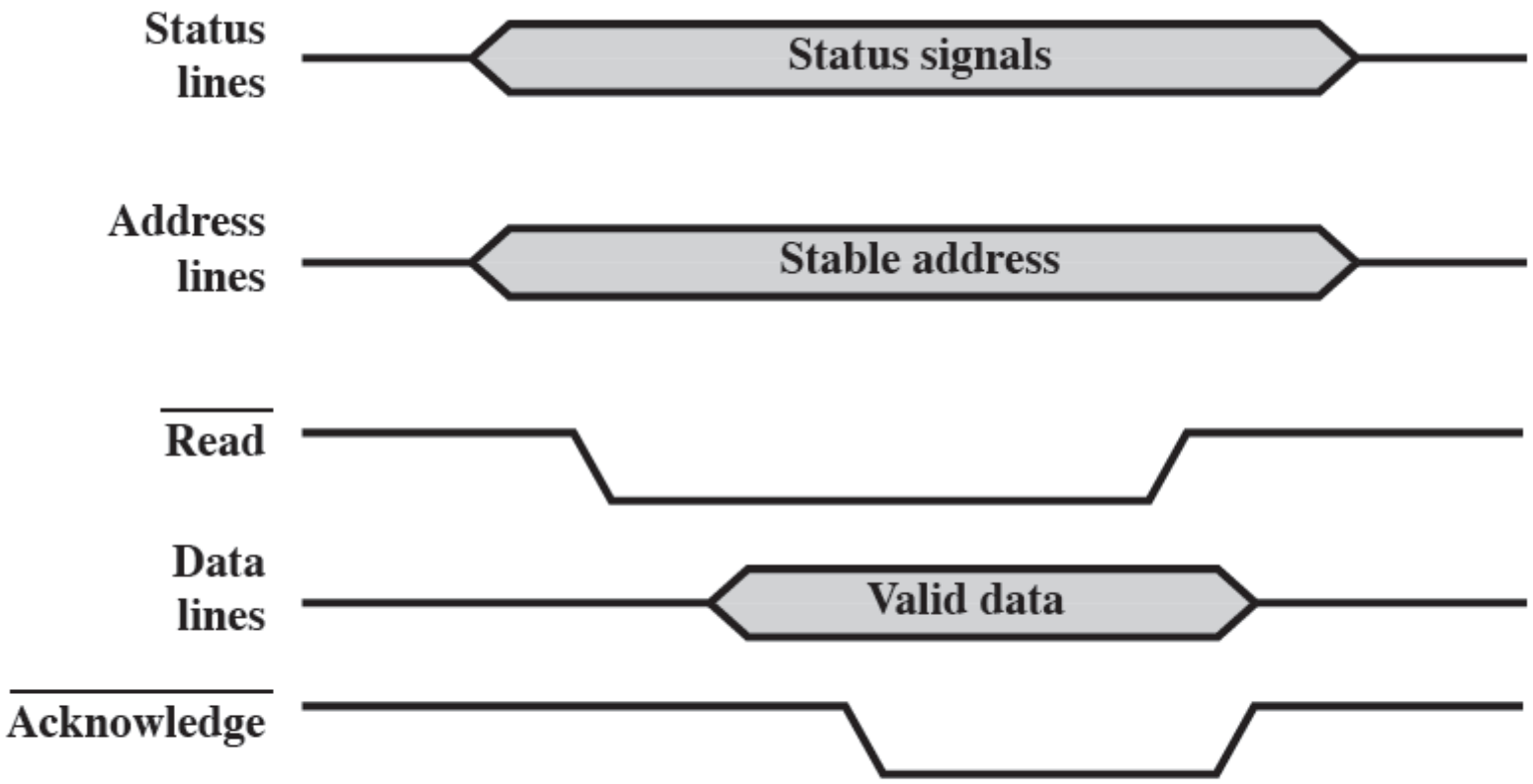
Central Elements of a Bus Design

- **Bus width**
  - Address
  - Data
- **Data Transfer Types**
  - Read
  - Write
  - Read-modify-write
  - Read-after-write
  - Block

Synchronous Bus

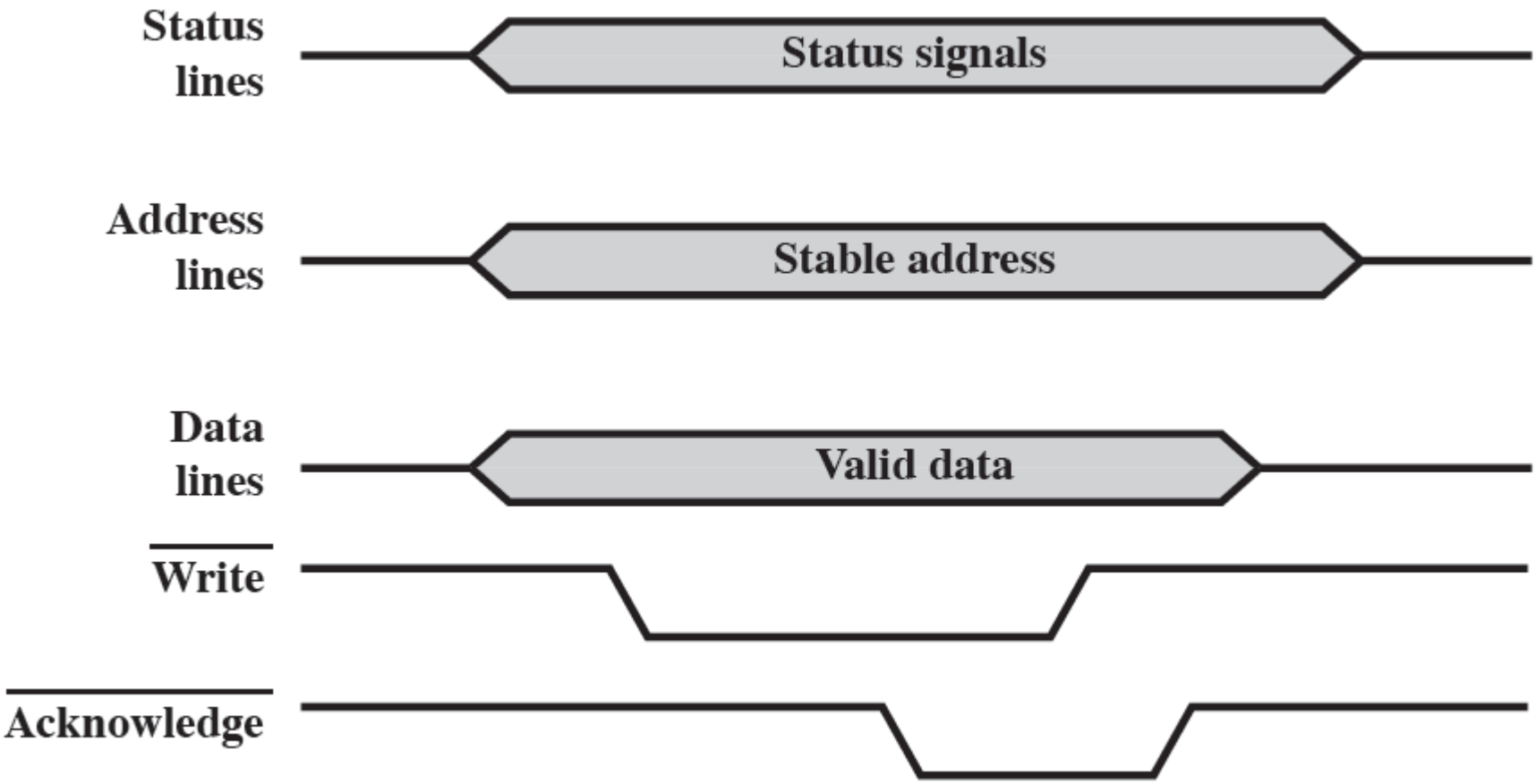


Asynchronous Read



(a) System bus read cycle

Asynchronous Write



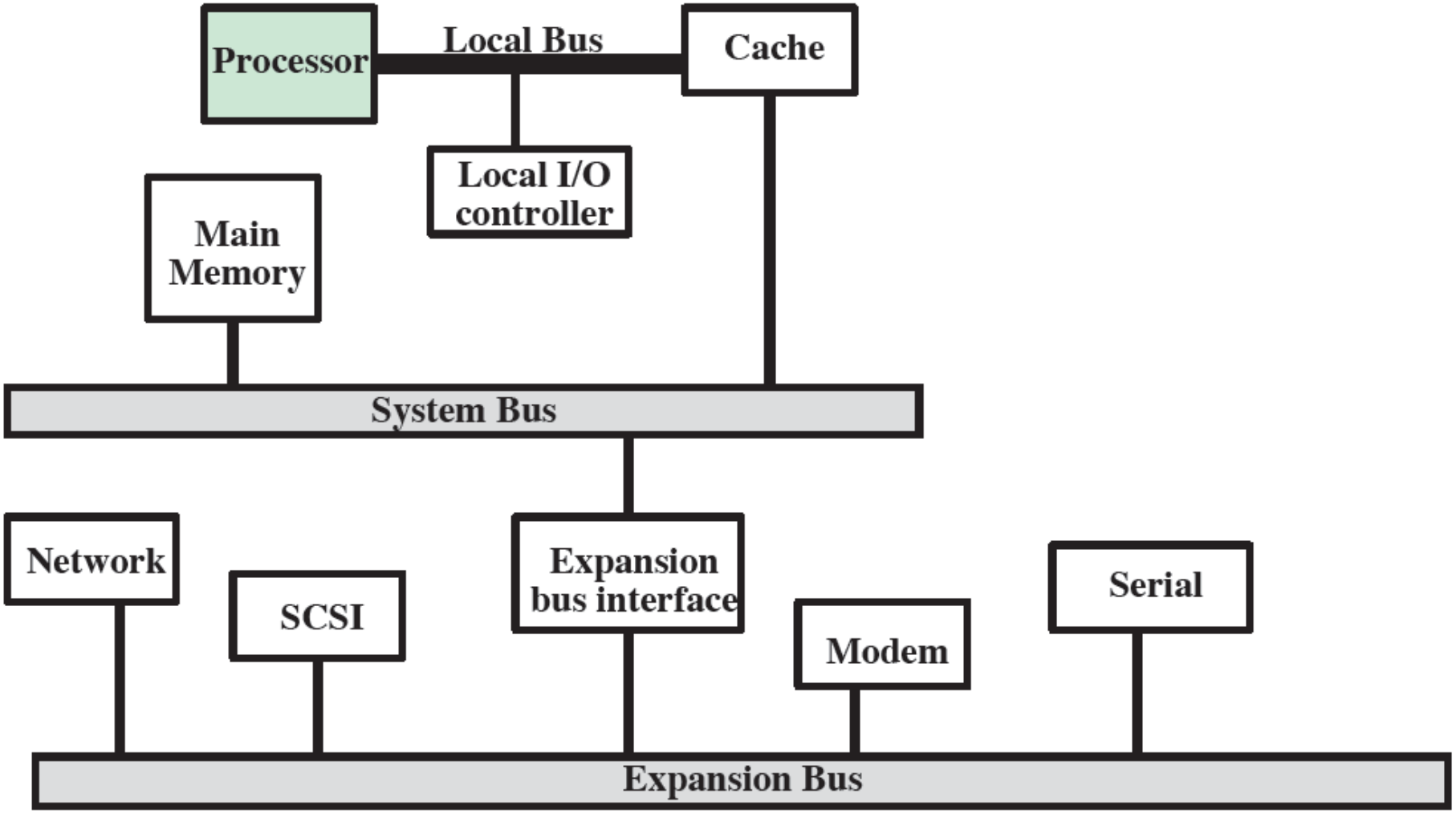
Factors Limiting the Speed of a Bus

- If a great number of devices are connected to the bus, performance will suffer:
  - the more devices, the longer the bus (physically) -> greater latency
  - higher coordination overhead
  - The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.

- Counter measures:
  - Increase the data width
  - Increase clock speed of bus

- These measures won't ultimately solve that problem

A Classical Hierarchical Bus Layout



Using a Bus Hierarchy

- In our Example:
  - Local Bus: Connecting processor to cache
  - Cache Memory Controller connects the cache to a system bus
  - System Bus: Connection to main memory and I/O

- The cache reduces the memory access which enabled the move of the memory to the System Bus
- This in turn enables the direct communication of I/O devices over the system bus without interfering with the processor

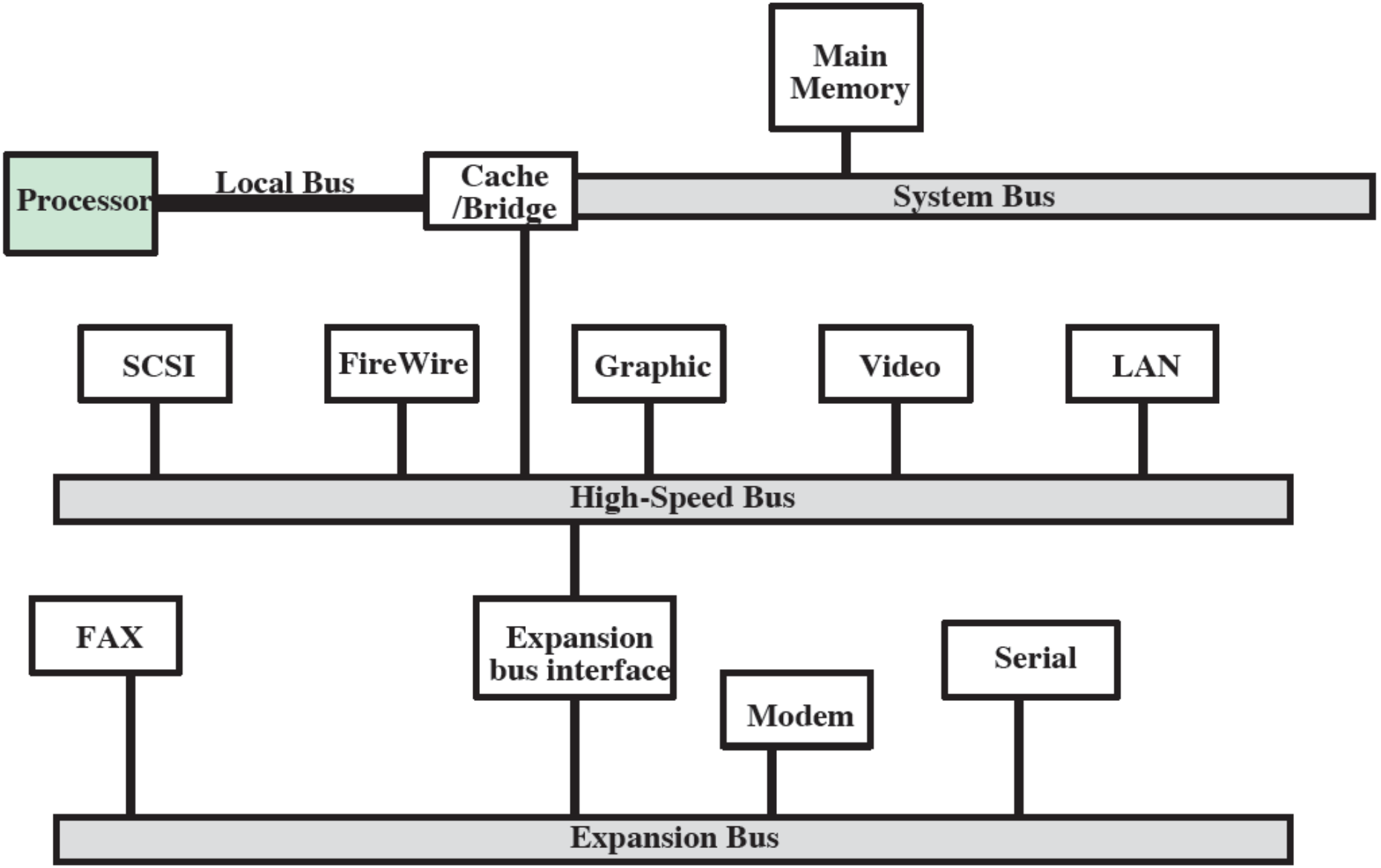
- Normally, I/O devices are not directly connected to system bus but to a secondary bus, the Expansion bus interface

Using a Bus Hierarchy

- This set-up is also reaching its boundaries

- Due to the ever increasing demand in I/O performance, a newer approach is taken:
  - So called mezzanine architecture
  - A high speed bus tightly integrated into the system
  - The cache is integrated into a bridge or buffering device which connects the processor's bus and the high-speed bus is required

Nowadays Design



Advantages of the Mezzanine Architecture

- High-speed bus brings high demand devices into closer integration with the processor
- At the same time it is independent of the processor
- Thus, differences in processor and high-speed bus speeds and signal line definitions are tolerated
- Changes in processor architecture do not affect the high-speed bus, and vice versa

Top-Level View of a Computer

Lecture Content

- Instruction Cycle
- Interrupts
- I/O Functionality
- Bus Systems
- QPI
- PCIe

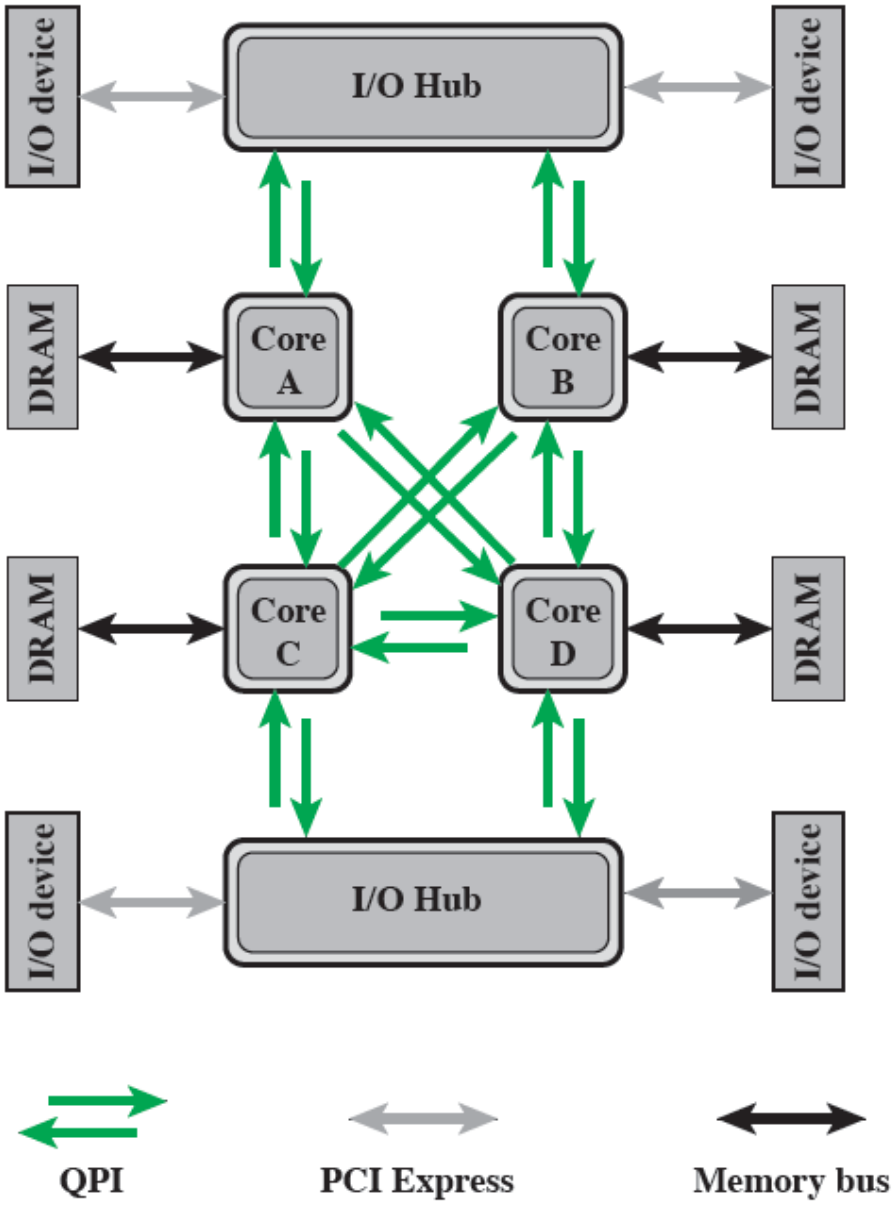
Point to Point Interconnect

- Wide synchronous buses came to a limit with the ever increasing clock speed
- At higher and higher data rates it becomes increasingly difficult to perform the synchronization and arbitration functions in a timely fashion
- Especially a conventional shared bus on the same chip magnified the difficulties
- Solution: Point to Point Interconnect. Has lower latency, higher data rate, and better scalability

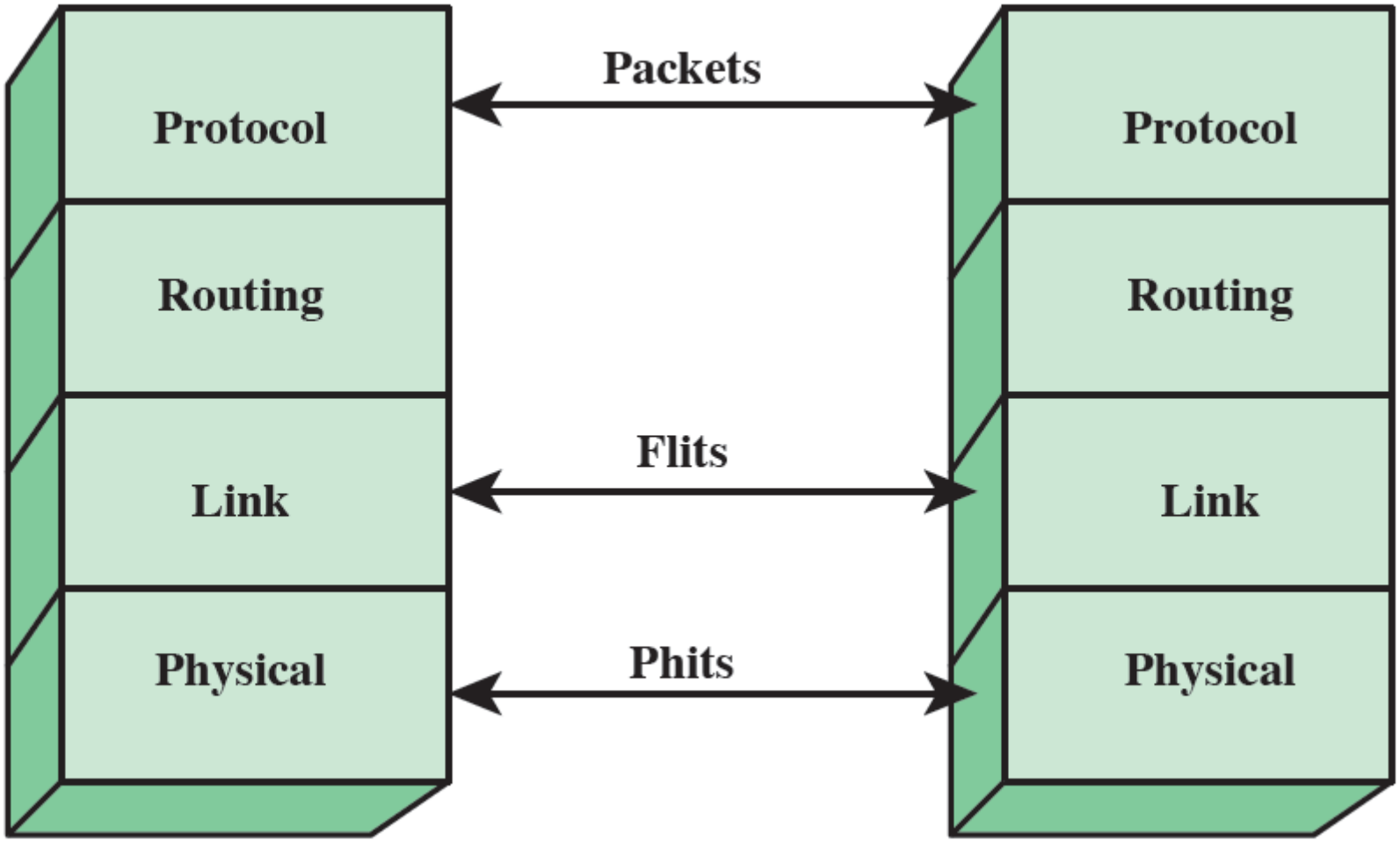
Important Example: Quick Path from Intel

- Introduced in 2008, replaced the Front Side Bus (FSB)
- Direct pairwise connections to other components
  - No need for arbitration
- Layered protocol architecture
  - In contrast to simple use of control signals found in shared bus
  - 20 Bit physical (**phits**), 80 Bit logical (**flits**)
- Packetized data transfer
  - Data are sent as a sequence of packets each of which includes control headers and error control codes

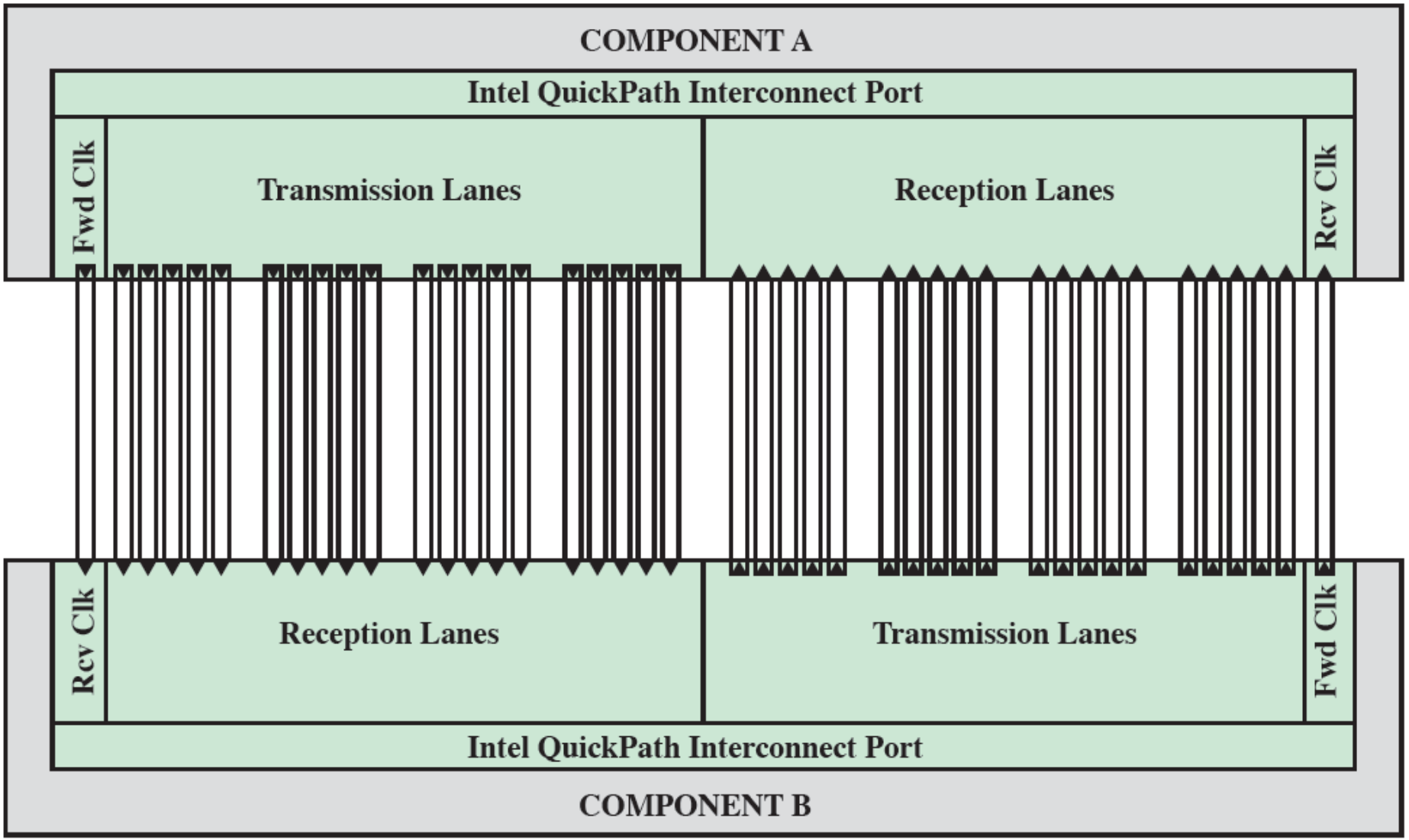
Example with 4 Cores



The QPI Layers



QPI Physical Connections



QPI Facts

- 84 individual links; each data path consists of a pair of wires, called a lane.
- Operates at 6.4 GT/s. At 20 bits per transfer, that adds up to 16 GB/s in one direction, the total capacity is 32 GB/s.

- The form of transmission on each lane is known as low-voltage differential signaling, or balanced transmission.
  - The signal is transmitted by small voltage differences
  - The receiver consists of a resistor
  - The polarity of the current going through the resistor defines 0 or 1
  - Very low electro magnetic noise

The QPI Link Layer

- Performs two key functions: **flow control** and **error control**
  - Operate on the level of the flit (flow control unit)
  - Each flit consists of a 72-bit message payload and an 8-bit error control code called a *cyclic redundancy check* (CRC)

- Flow control function
  - In order to not overwhelm the receiver
  - Credit based system

- Error control function
  - Detects and recovers from bit errors, and so isolates higher layers from experiencing bit errors
  - Requires buffering of the sender (in case something needs to be resent)

QPI Routing Layer

- Used to determine the course that a packet will traverse across the available system interconnects
- Defined by firmware and describe the possible paths that a packet can follow
- The routing table options may be quite complex, depending on how
  - devices are populated
  - system resources are partitioned
  - reliability events result in mapping around a failing resource

Protocol Layer

- Packet is defined as the unit of transfer
- One key function performed at this level is a cache coherency protocol which deals with making sure that main memory values held in multiple caches are consistent
- A typical data packet payload is a block of data being sent to or from a cache
- Contains 8 Bit for routing information
  - That means one 80 Bit flit contains 64 bit payload, 8 bit routing and 8 bit CRC

Top-Level View of a Computer

Lecture Content

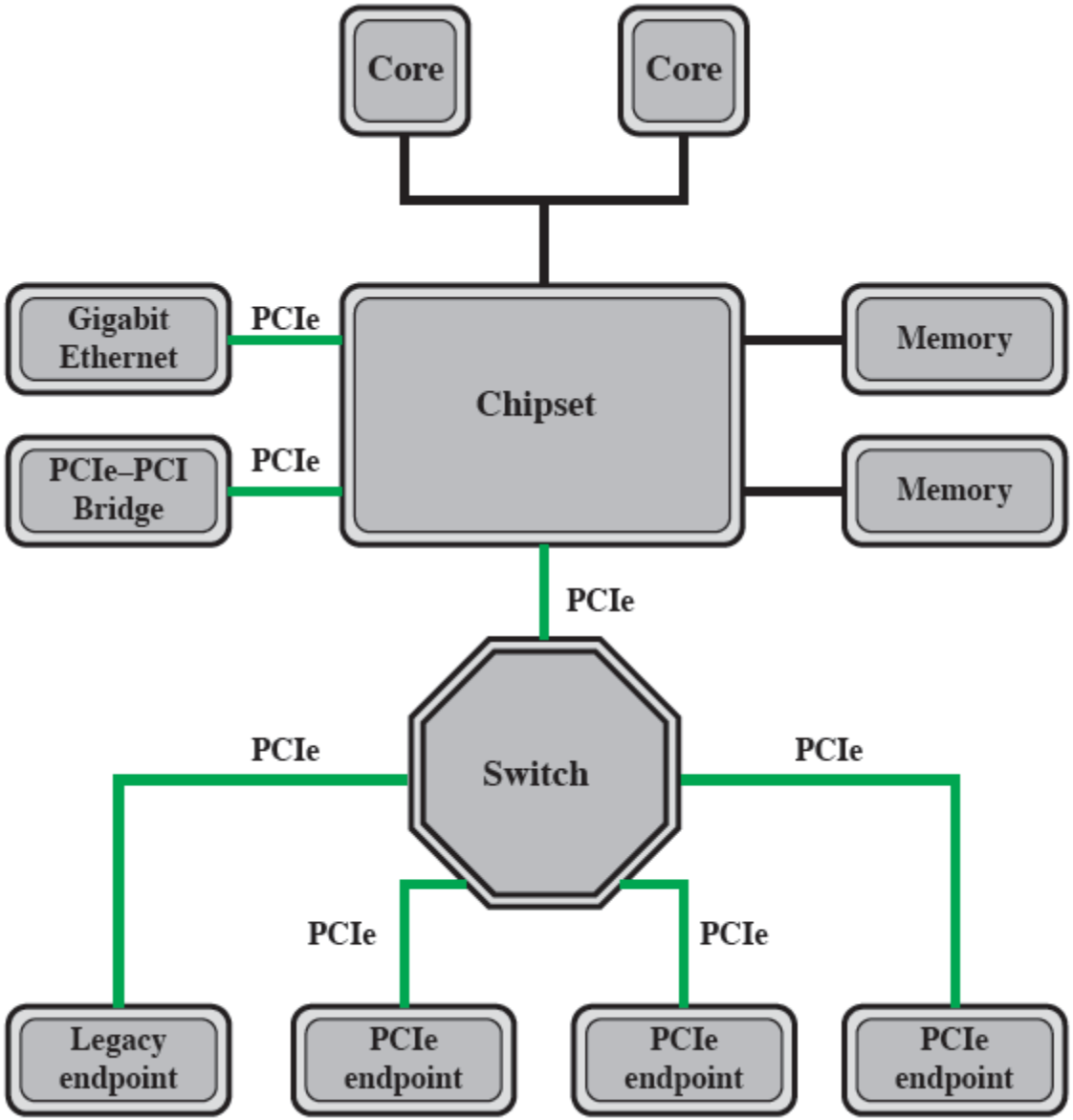
- Instruction Cycle
- Interrupts
- I/O Functionality
- Bus Systems
- QPI
- PCIe

Peripheral Component Interconnect (PCI)

- A popular high bandwidth, processor independent bus
- Delivers better system performance for high speed I/O subsystems
- PCI Special Interest Group (SIG)
  - First developed by Intel
  - Widely adopted

- PCI Express (PCIe)
  - Point-to-point interconnect scheme intended to replace bus-based schemes such as PCI
  - Key requirement is high capacity to support the needs of higher data rate I/O devices, such as Gigabit Ethernet
  - Another requirement deals with the need to support time dependent data streams

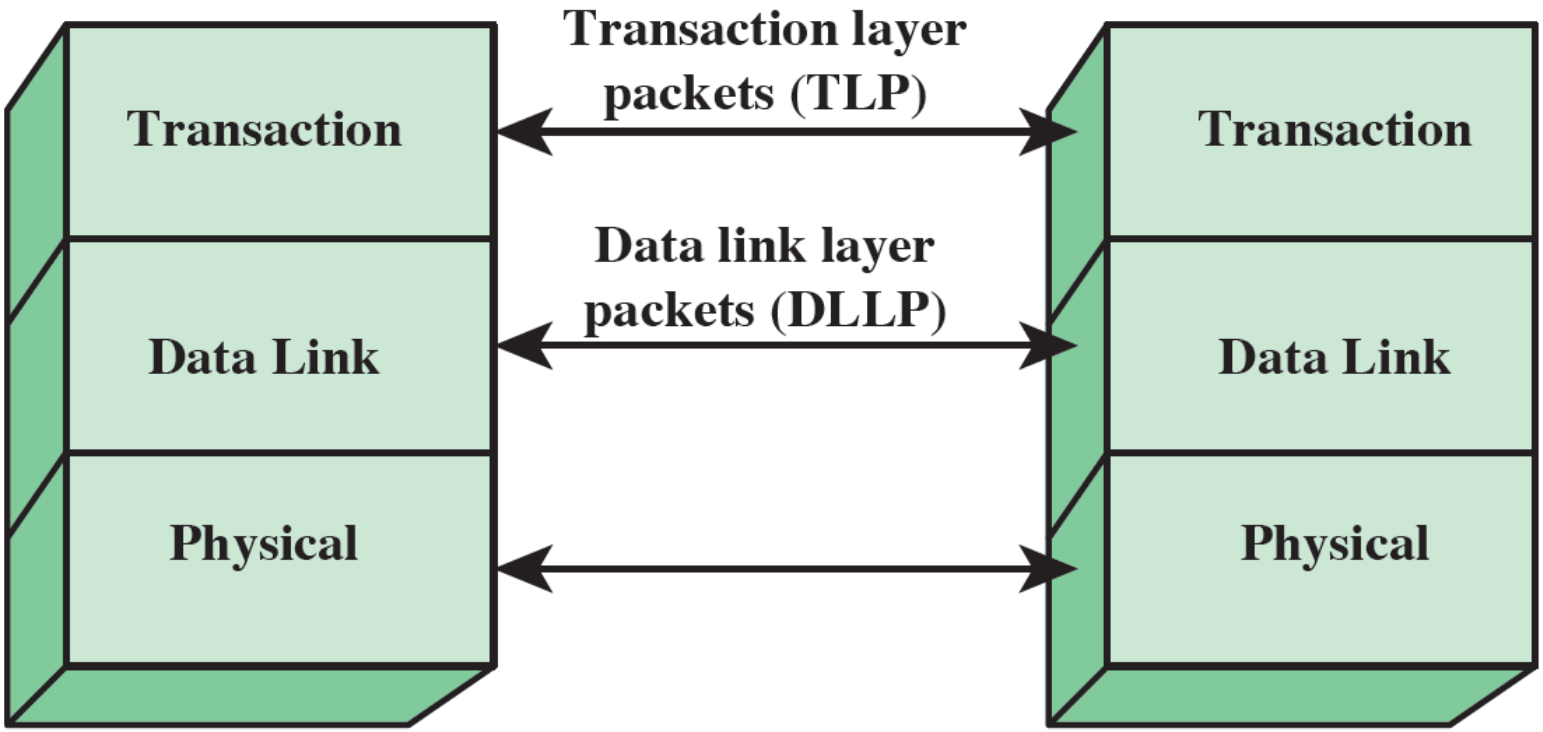
PCIe Logical Architecture



PCIe Configuration

- **Root Complex** device
  - also referred to as a chipset or a host bridge
  - connects the processor and memory subsystem to the PCI Express switch
  - Acts as buffering device to deal with the difference in the datarate
- The chipset will typically support multiple PCIe ports
  - some of which attached directly to a PCIe device
  - some are attached to a switch that manages multiple PCIe streams.
- Legacy endpoints
  - Allows for existing designs to be ported to PCIe
- PCIe/PCI bridge

PCIe also has a Protocol



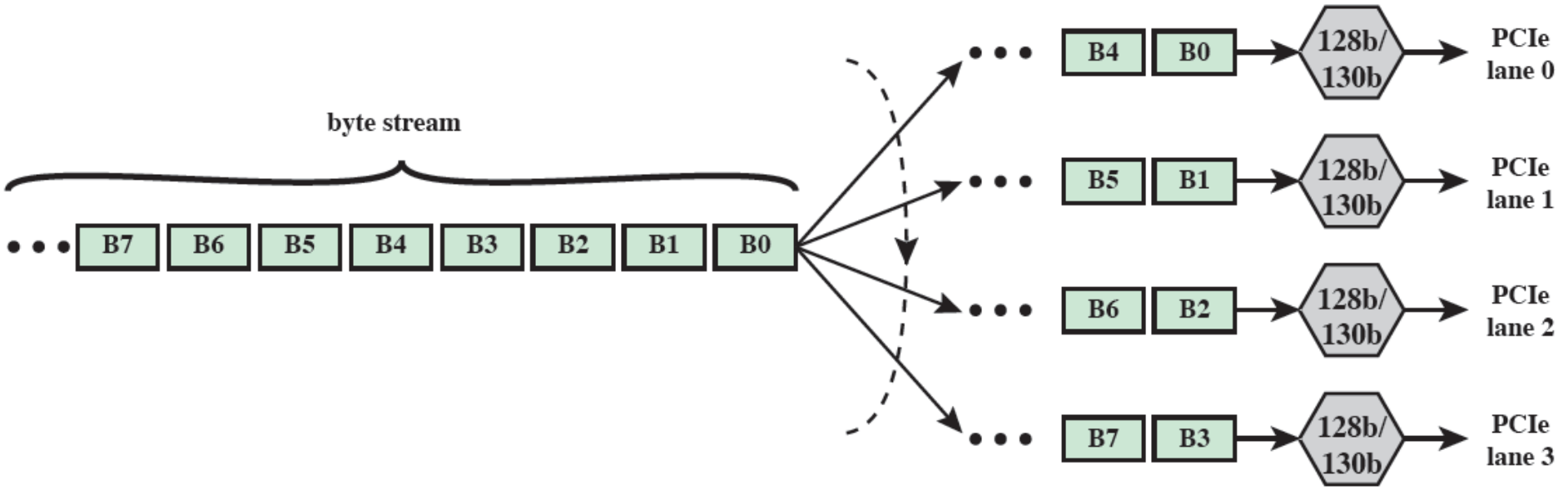
PCIe Protocol Layers

- **Physical Layer**
  - The actual wires carrying the signals
  - Circuitry and logic required in the transmission and receipt of data
- **Data Link Layer**
  - Responsible for reliable transmission and flow control
  - Data packets generated and consumed by the DLL are called Data Link Layer Packets (DLLPs).
- **Transaction Layer**
  - Generates and consumes data packets
  - Manages the flow control of those
  - Data packets generated and consumed by the TL are called Transaction Layer Packets (TLPs).

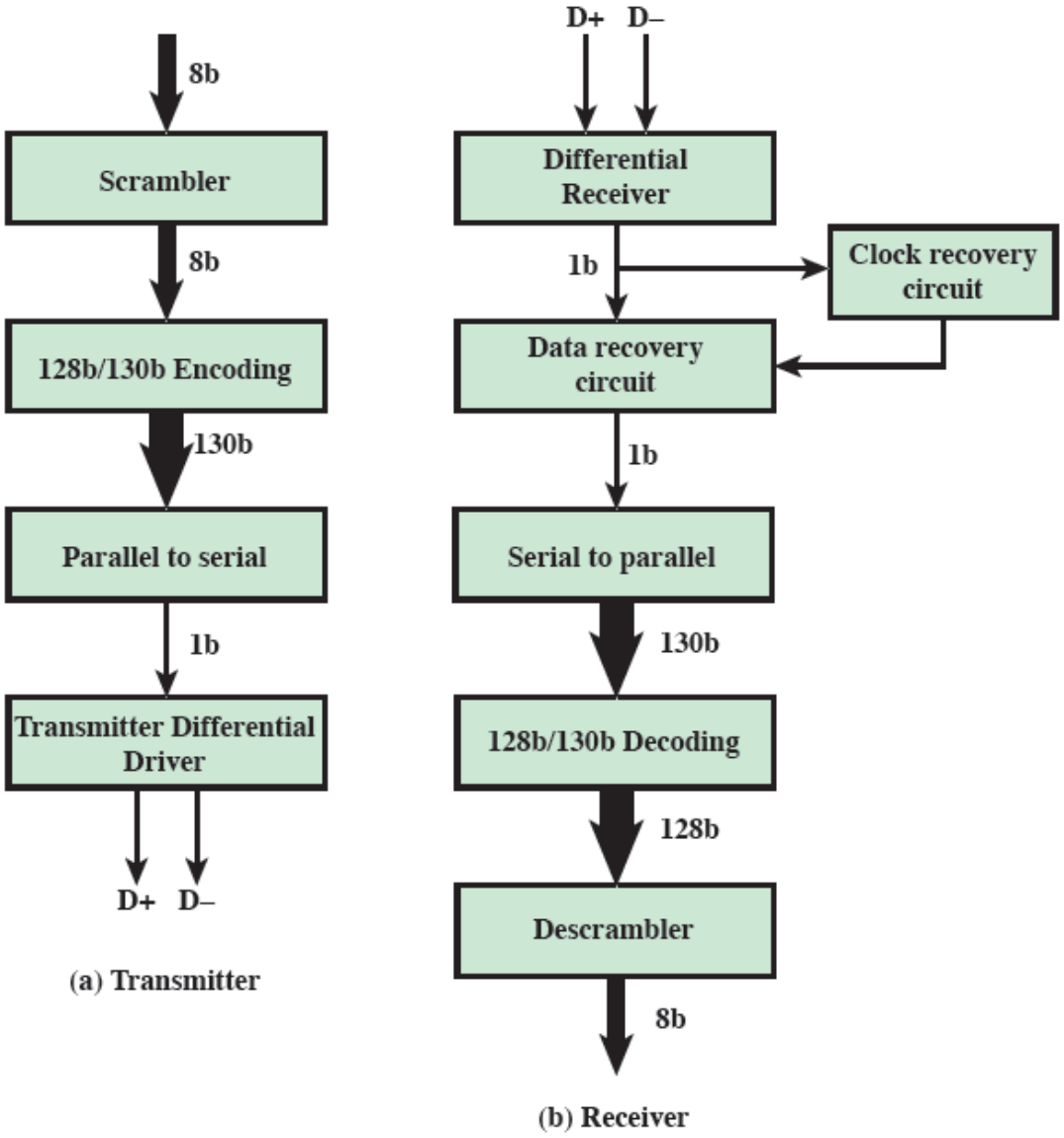
Details on the Physical Layer

- 1, 4, 6, 16, or 32 bidirectional lanes per port, again differential signaling
- As with QPI, PCIe uses a multilane distribution technique.
- At each physical lane, data are buffered and processed
  - Blocks of 16 bytes (128 bits) at a time
  - Each block is encoded into a unique 130-bit codeword for transmission
  - 128b/130b encoding.
- 128b/130b
  - PCIe has no clock. The two additional bytes are used to synchronize

PCIe Multilane Distribution



PCIe Transmit and Receive Block Diagram



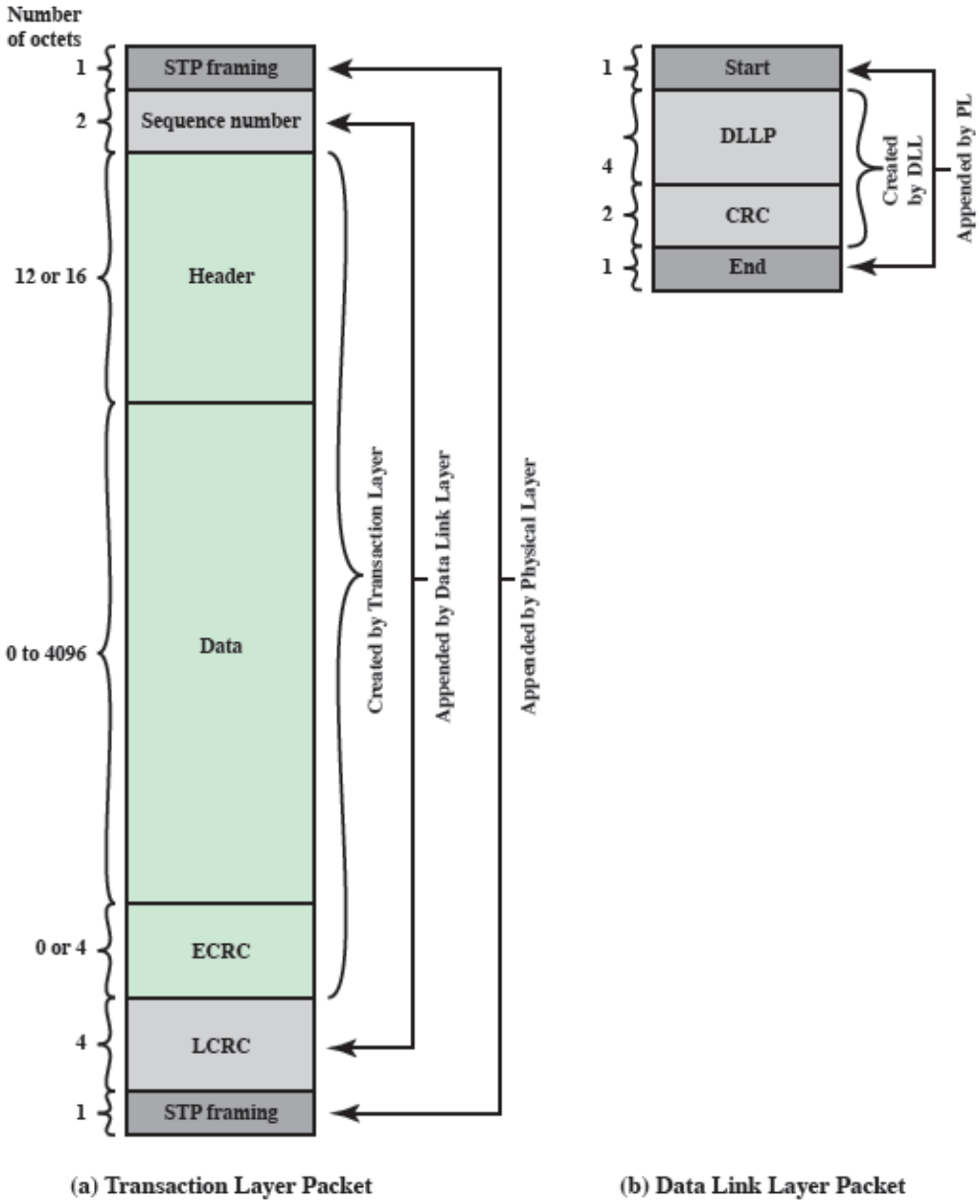
PCIe Transaction Layer

- Receives read and write requests from the software above the TL and creates request packets for transmission to a destination via the link layer
- Most transactions use a split transaction technique
  - A request packet is sent out by a source PCIe device which then waits for a response called a completion packet
- TL messages and some write transactions are posted transactions (meaning that no response is expected)
- TL packet format supports 32-bit memory addressing and extended 64-bit memory addressing

PCIe Transaction Layer: Address Spaces

- **Memory**
  - The memory space includes system main memory. It also includes PCIe I/O devices. Certain ranges of memory addresses map into I/O devices.
- **I/O**
  - This address space is used for legacy PCI devices, with reserved memory address ranges used to address legacy I/O devices.
- **Configuration**
  - This address space enables the TL to read/write configuration registers associated with I/O devices.
- **Message**
  - This address space is for control signals related to interrupts, error handling, and power management.

PCIe Protocol Data Unit Format



(a) Transaction Layer Packet

(b) Data Link Layer Packet

TLP Memory Request Format