Input/Output

Introduction

- Motivation
- Performance metrics
- Processor interface issues
- Buses

Motivation

- CPU Performance: 60% per year
- I/O system performance limited by *mechanical* delays (e.g., disk I/O)

  < 10% per year (IO per sec or MB per sec)
- Amdahl's Law: system speed-up limited by the slowest part!

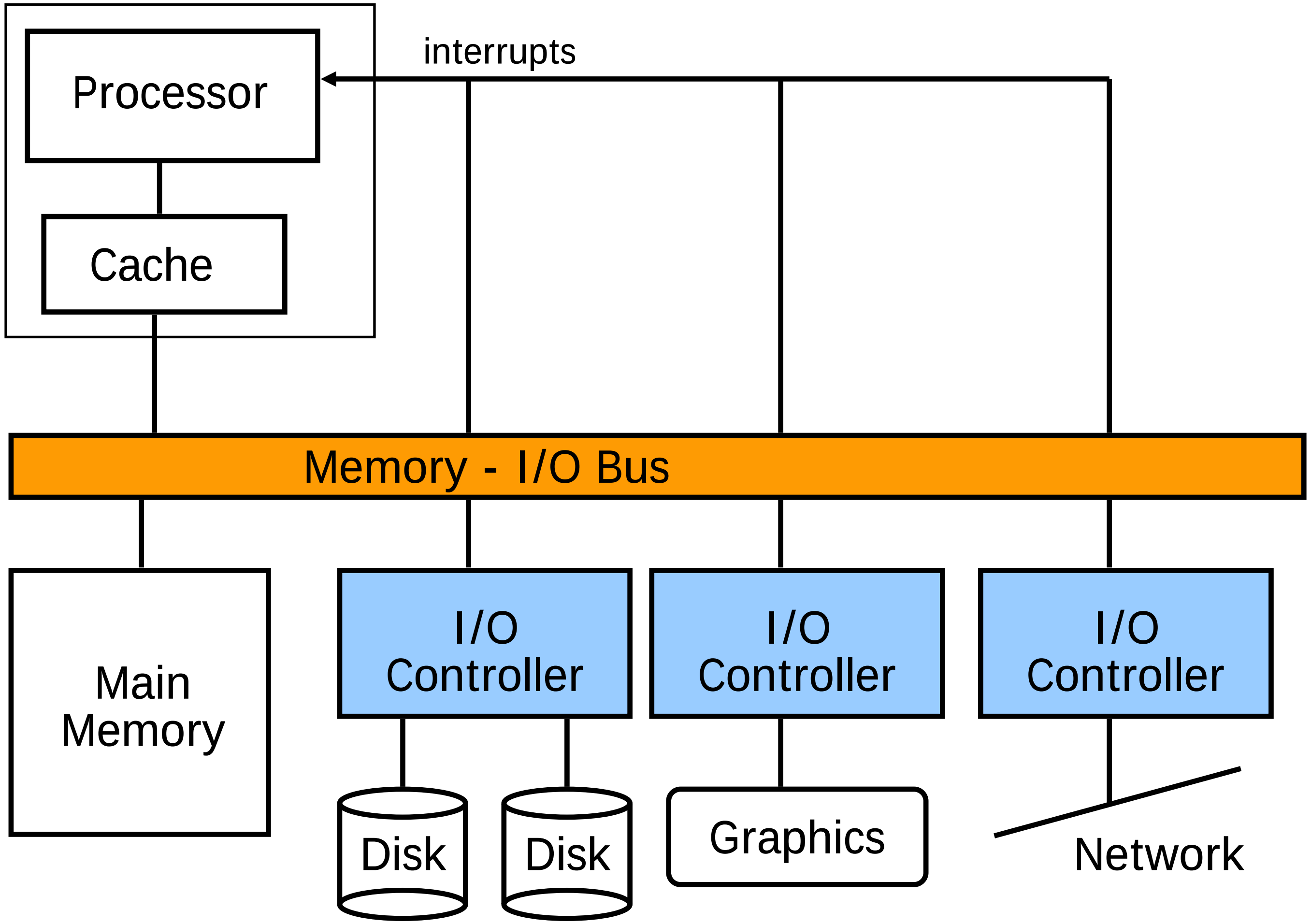
  10% IO & 10x CPU ➡ 5x Performance (lose 50%)

  10% IO & 100x CPU ➡ 10x Performance (lose 90%)
- I/O bottleneck:

  Diminishing fraction of time in CPU

  Diminishing value of faster CPUs

I/O Systems



interrupts
Processor
Cache
Memory - I/O Bus
Main Memory
I/O Controller
I/O Controller
I/O Controller
Disk
Disk
Graphics
Network

I/O performance metrics

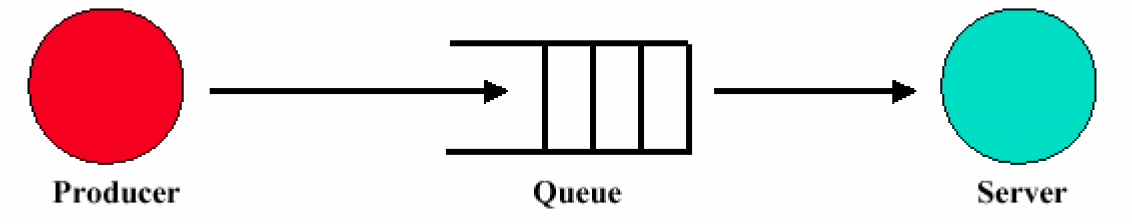
I/O performance

- I/O system performance depends on:
  - The CPU
  - The memory system:
    - Caches
    - Main Memory
  - The underlying interconnection (buses)
  - The I/O controller
  - The I/O device
  - The speed of the I/O software (Operating System)
  - The efficiency of the software using the I/O devices

  Limited by weakest link in the chain
- Two common performance metrics:
  - Throughput: I/O bandwidth
  - Response time: Latency

Throughput vs. response time

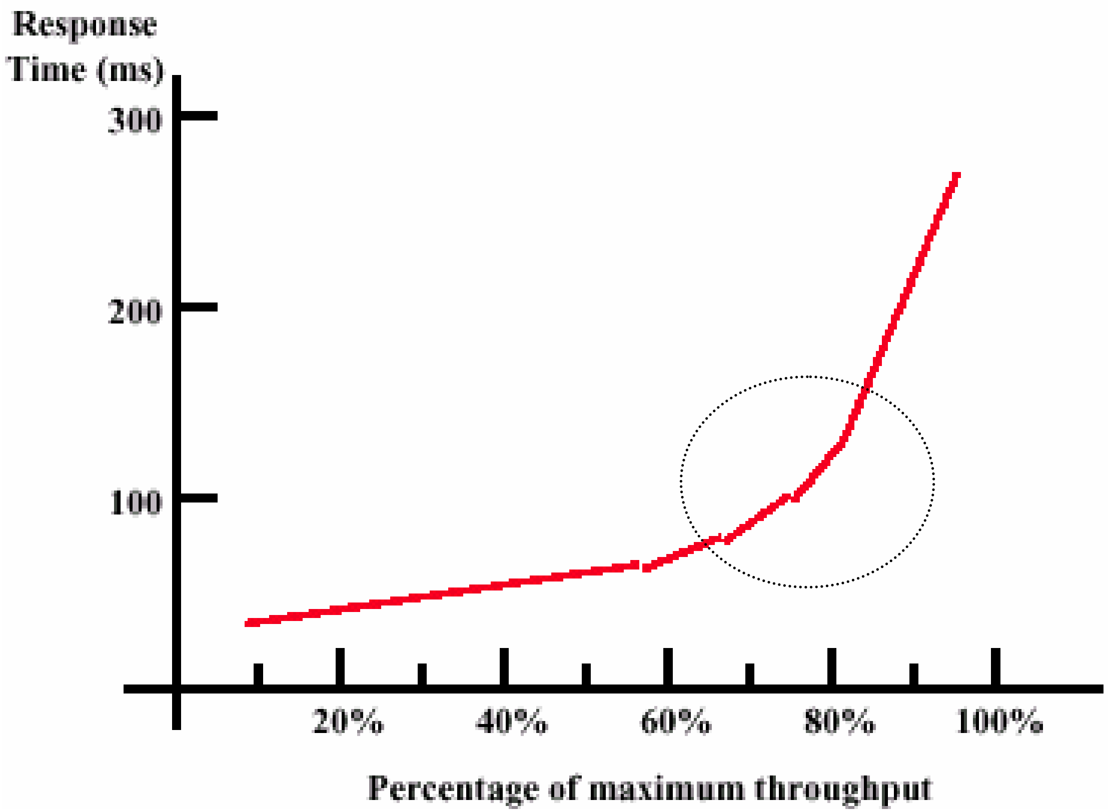
- View device as a queue:



- **Throughput**:
  - The number of tasks completed by the server in unit time
  - In order to get the highest possible throughput:
    - The server should never be idle
    - The queue should never be empty
- **Response time**:
  - Begins when a task is placed in the queue
  - Ends when it is completed by the server
  - In order to minimize the response time:
    - The queue should be empty
    - The server will be idle

Throughput vs. response time (2)

- Response time tend to quickly increase for higher throughput values



I/O metrics

- CPU time different from system time
- In general:

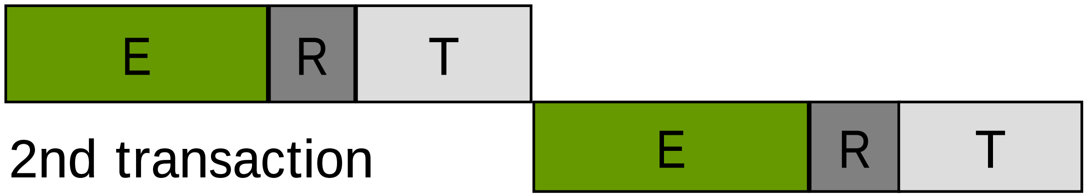
  $$Time_{workload} = Time_{CPU} + Time_{I/O} - T_{overlap}$$

  - $T_{overlap}$ = time in which CPU and I/O are overlapped
  - Example:
    - Workload takes 50ns to complete, 30ns CPU, 30ns I/O
    - $T_{overlap}$ = 10ns
- When scaling CPU time (higher speed), $T_{overlap}$ may or may not scale
  - Best case: $T_{overlap}$ scales completely
  - Worst case: $T_{overlap}$ does not scale
  - Average case: $T_{overlap}$ scales partly

Response Time vs. Productivity

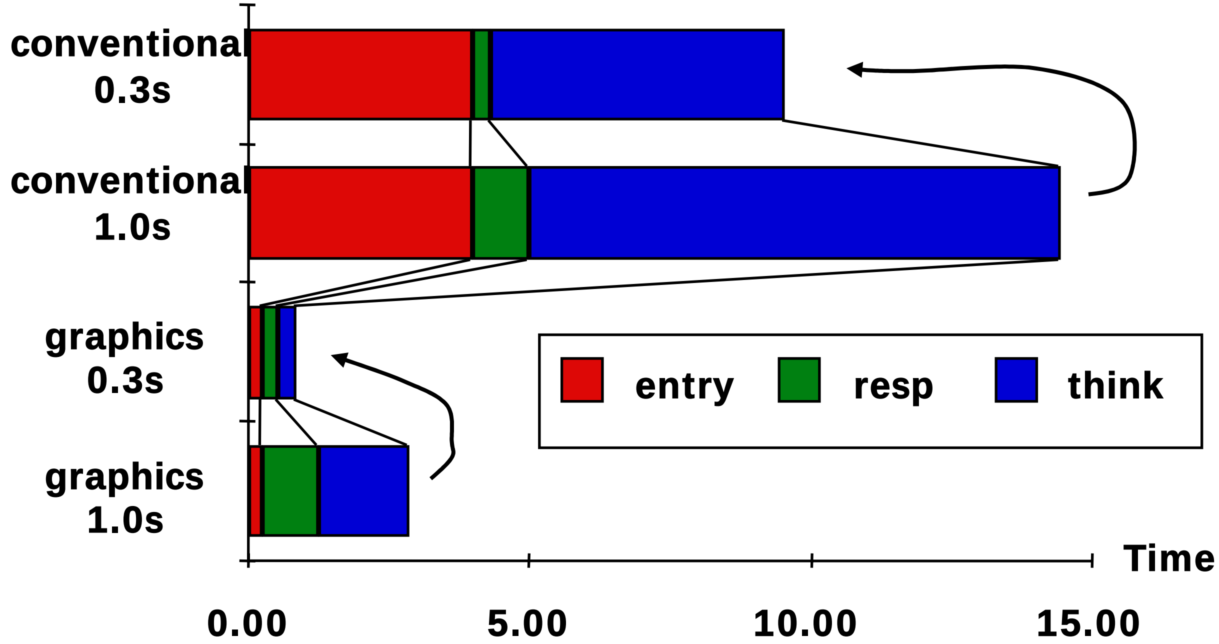
- Interactive environments:
  - Each interaction or transaction has 3 parts:
    - Entry Time: time for user to enter command
    - System Response Time: time between user entry & system replies
    - Think Time: Time from response until user begins next command
  - Example:

1st transaction



2nd transaction

Response Time & Productivity



- What happens to transaction time by shrinking *response time* from 1.0 s to 0.3 s?
  - Keyboard: 4.0 s entry, 9.4 s think time
  - Graphics: 0.25 s entry, 1.6 s think time
  - 0.7 s off response saves 4.9 s (34%) and 2.0 s (70%) total time per transaction ➡ greater productivity

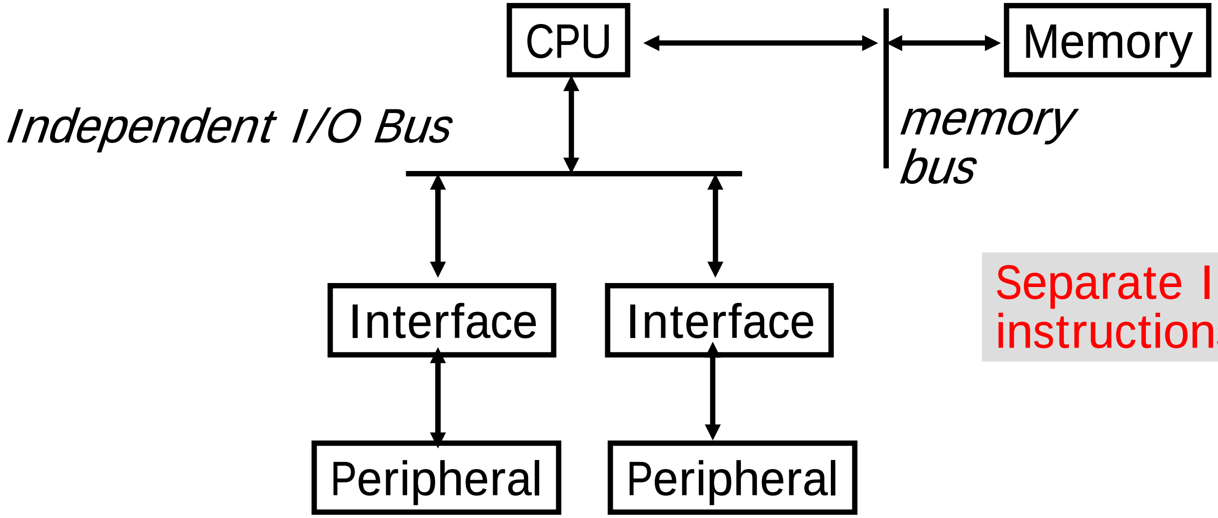
Processor interface issues

Processor Interface Issues

- I/O addressing
  - Isolated I/O
  - Memory mapped I/O
- I/O Control Structures
  - Polling
  - Interrupts
  - DMA
  - I/O Controllers or I/O Processors

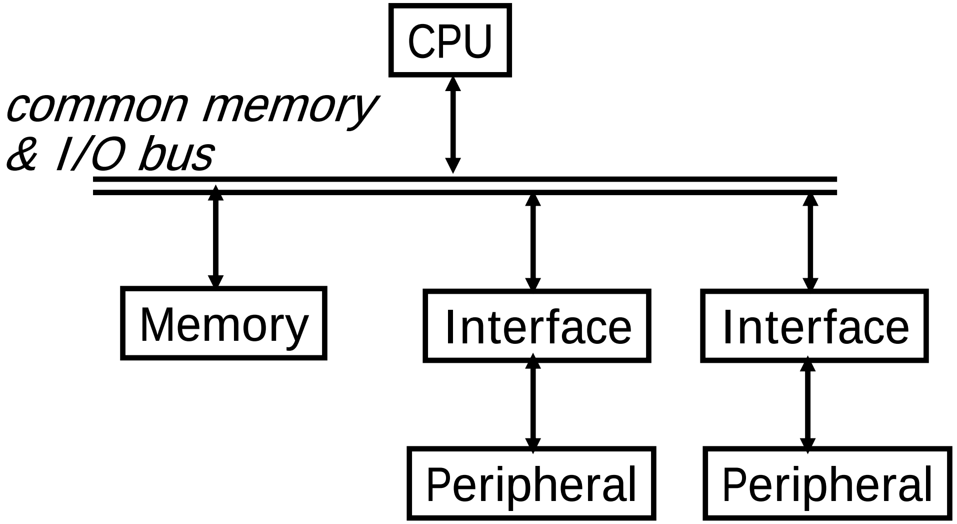
I/O Interface

ISOLATED I/O



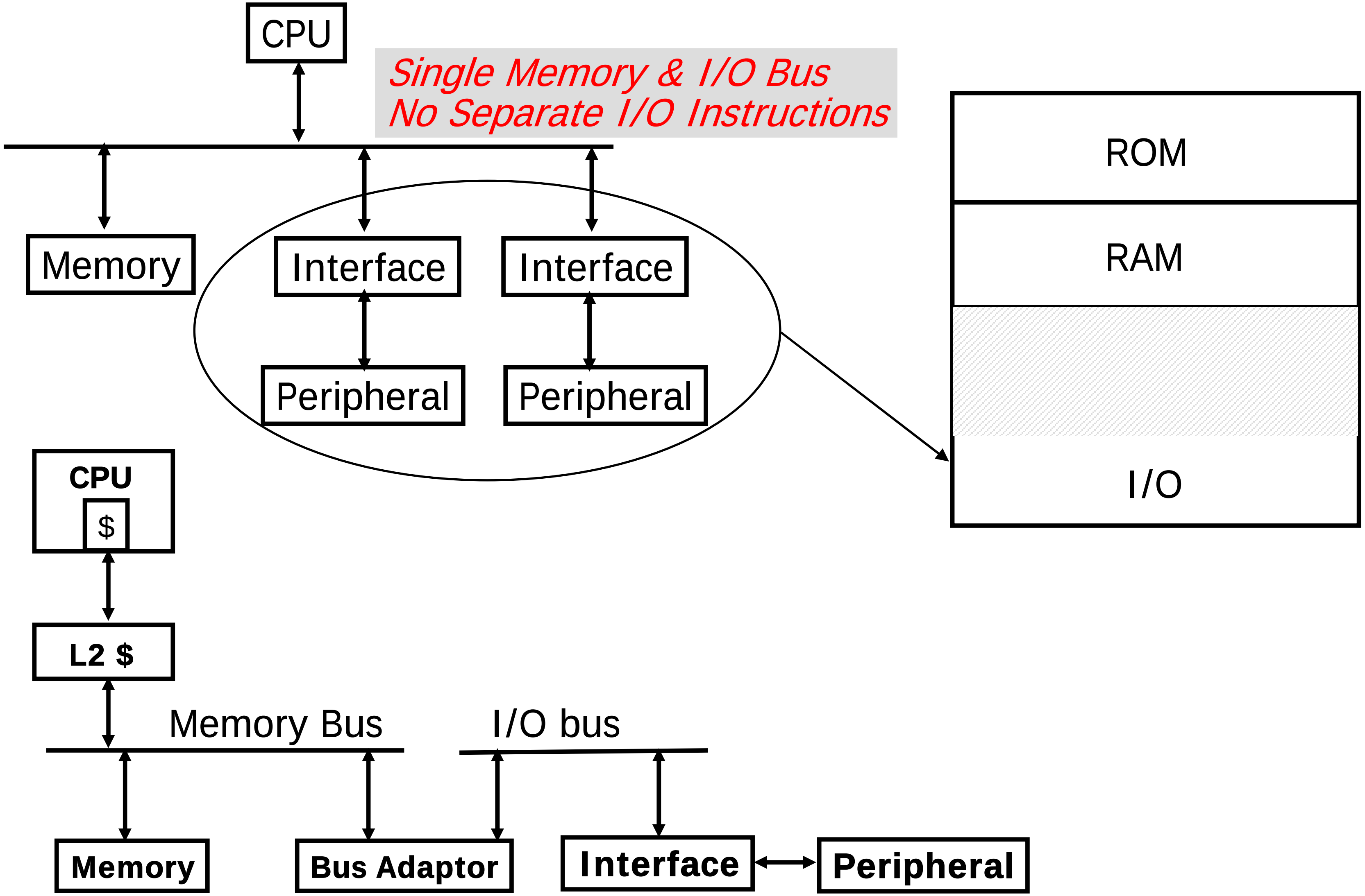
Separate I/O instructions (in,out)

MEMORY MAPPED I/O

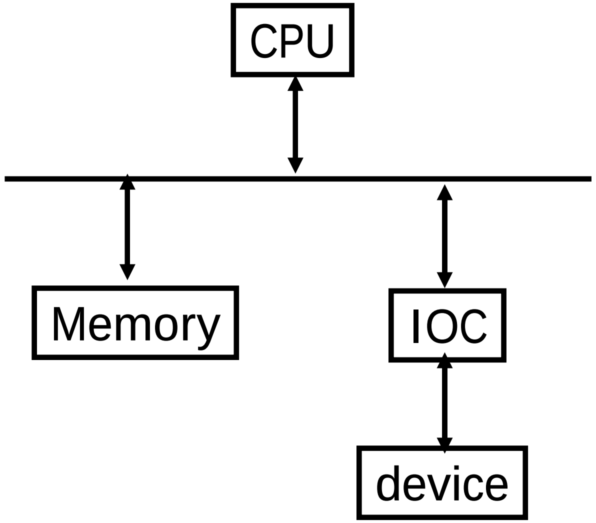


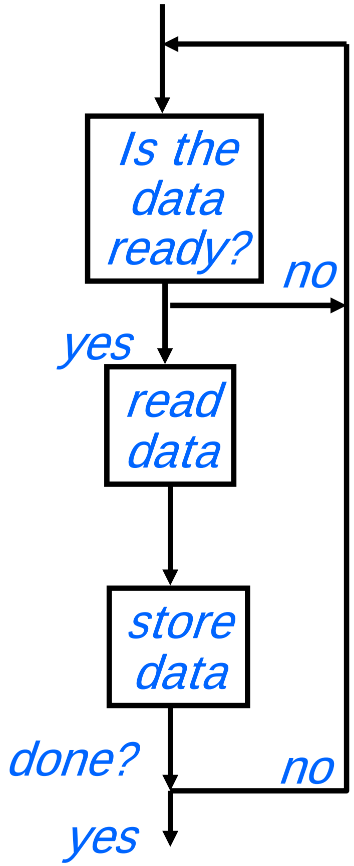
Lines distinguish between I/O and memory transfers

Memory Mapped I/O



Programmed I/O (Polling)

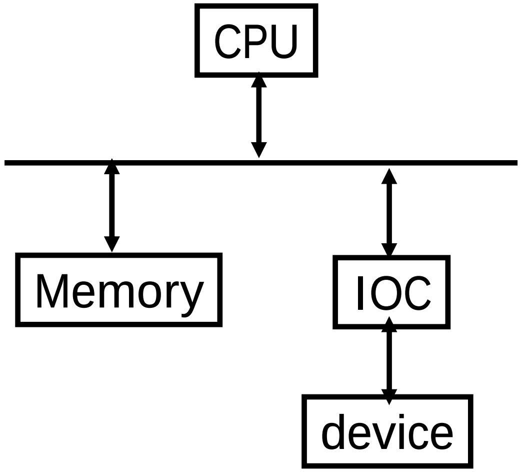


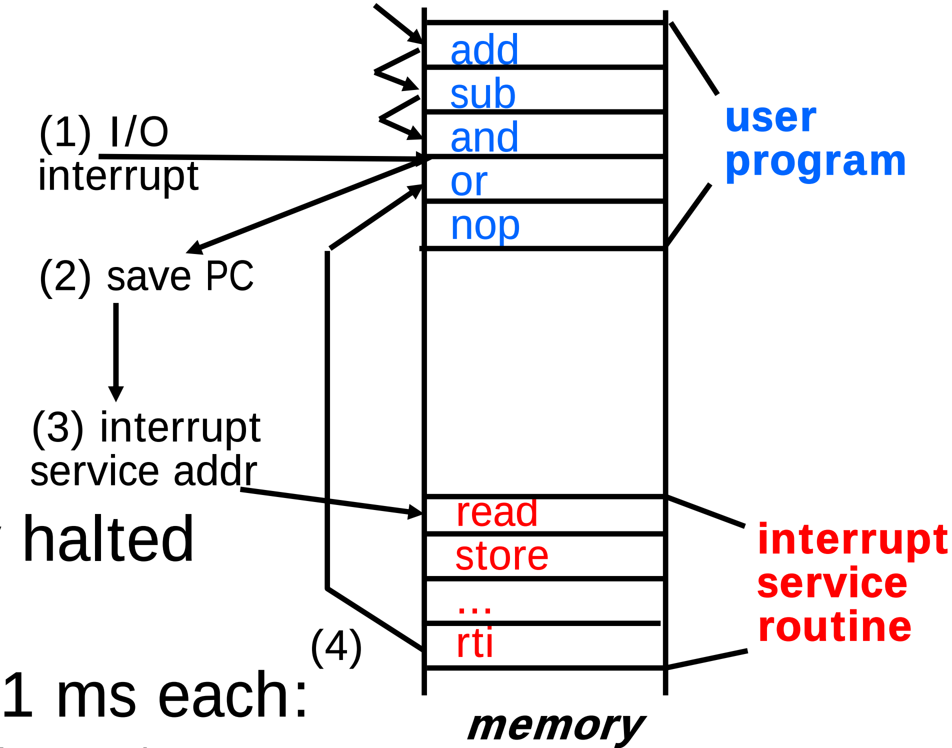


busy wait loop not an efficient way to use the CPU unless the device is very fast!

but checks for I/O completion can be dispersed among computationally intensive code

Interrupt Driven Data Transfer





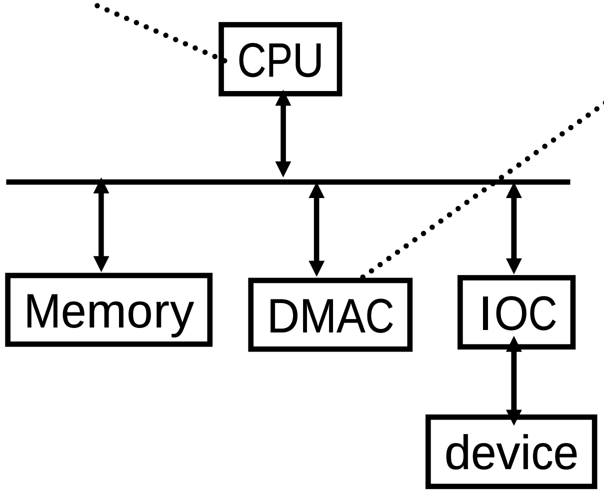
- User program progress only halted during actual transfer
- Example: 1000 transfers at 1 ms each:
  - 1000 interrupts @ 2 µs per interrupt
  - 1000 interrupt service @ 98 µs each = 0.1 CPU seconds
  - Device transfer rate = 10 MB/s => 0.1 x 10^{-6} s/B => 0.1 µs/B
    ➡ 1000 bytes = 100 µsec
  - 1000 transfers x 100 µsecs = 100 ms = 0.1 CPU seconds

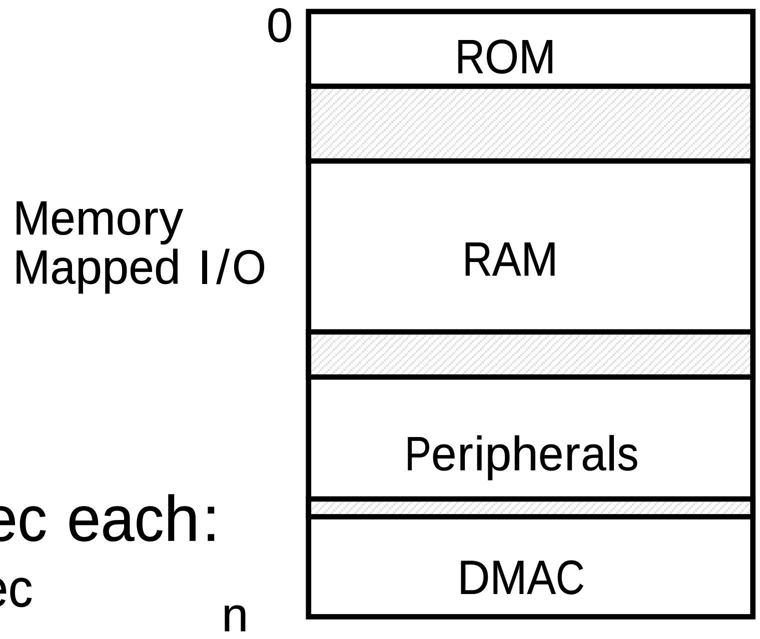
50% overhead

Direct Memory Access

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".

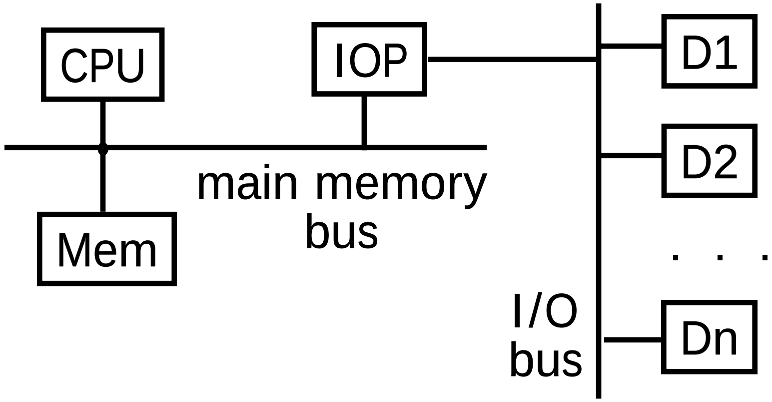
DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.

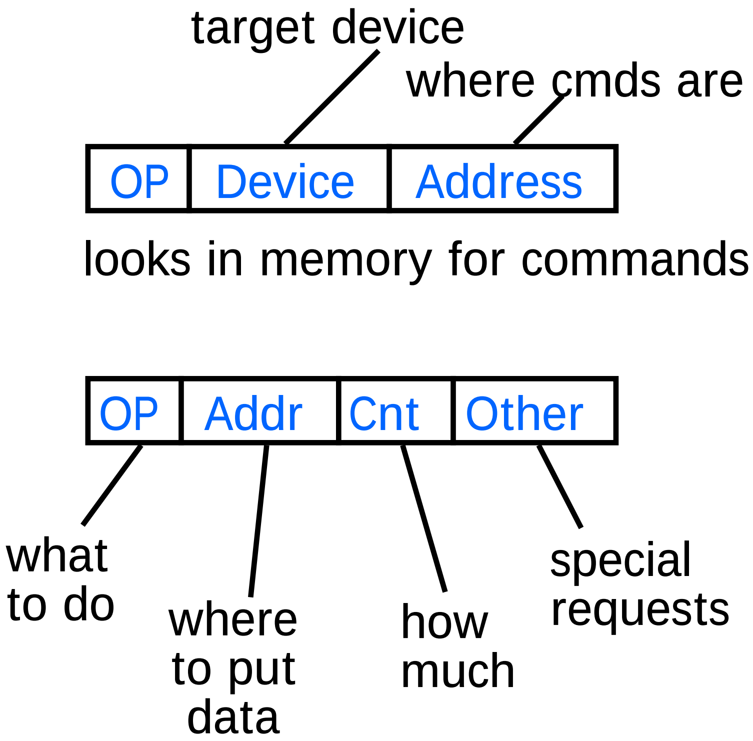




- Time to do 1000 xfers at 1 msec each:
  - 1 DMA set-up sequence @ 50 µsec
  - 1 interrupt @ 2 µsec
  - 1 interrupt service sequence @ 48 µsec
  - .0001 second of CPU time

Input/Output Processors





- CPU issues instruction to IOP
- Device to/from memory transfers are controlled by the IOP directly
  (IOP steals memory cycles)
- Interrupt CPU when done

Relationship to Processor Architecture

- I/O instructions have largely disappeared
- Interrupts:
  - Stack replaced by *shadow registers*
    - Handler saves registers and re-enables higher priority interrupts
    - Interrupt types reduced in number; handler must query interrupt controller
- Caches cause problems for I/O
  - Flushing is expensive, I/O pollutes cache
  - Solution is borrowed from shared memory multiprocessors "snooping"

Bus & interconnects

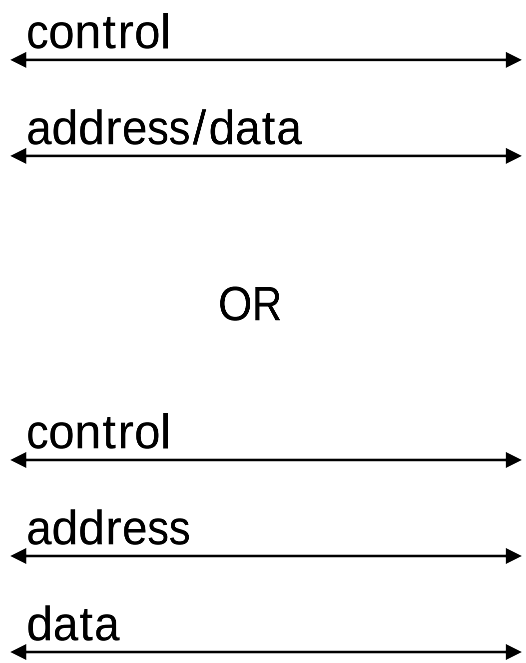
Bus & interconnects

- Bus classification
- Bus transactions
- Bus timing

What is a bus?

- A bus is a shared medium that connects the processor, memory, and I/O devices

- Consists of control and data/address wires
  - **control**: requests, acks, type of data (address or data)
  - **data**: data and addresses
  - **address** (optional): address

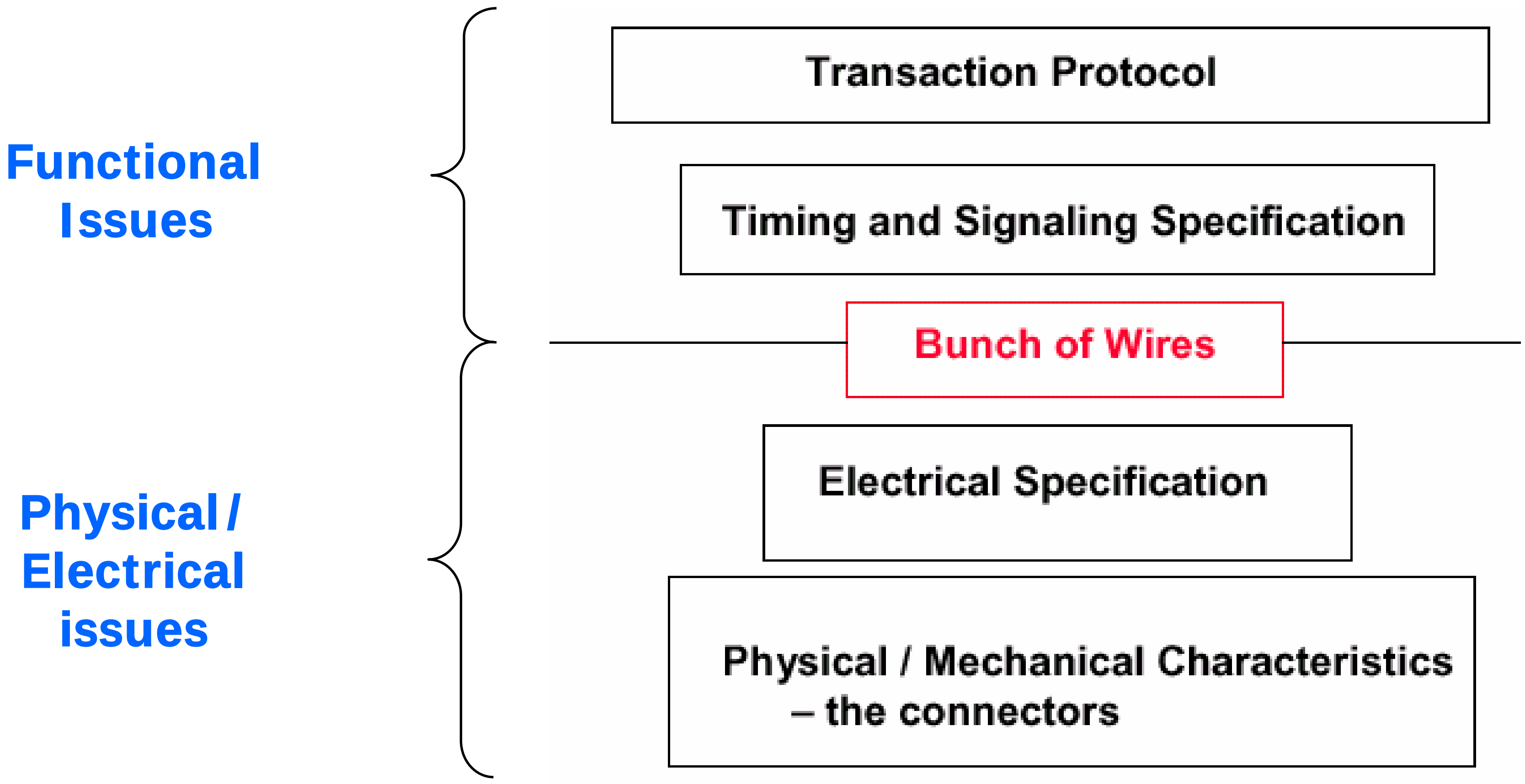


Bus-Based Interconnect

- Advantages:
  - Low cost
    - A single set of wires is shared by multiple ways
  - Versatility
    - Easy to add new devices & peripherals
    - May even be ported between computers using common bus
- Disadvantage
  - Bus is a **communication bottleneck**
    - Bandwidth may limit maximum I/O throughput
  - Bus speed is limited by **physical factors:**
    - The bus length
    - The number of devices (bus **load**)

    Prevent arbitrary bus speedup

What defines a bus?

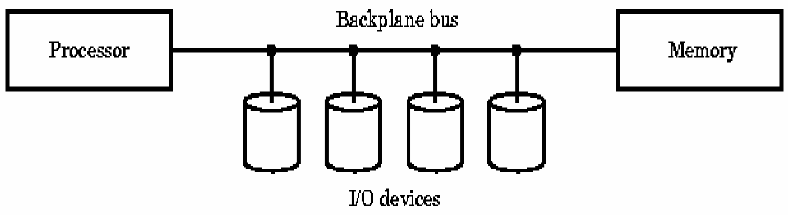


Types of buses

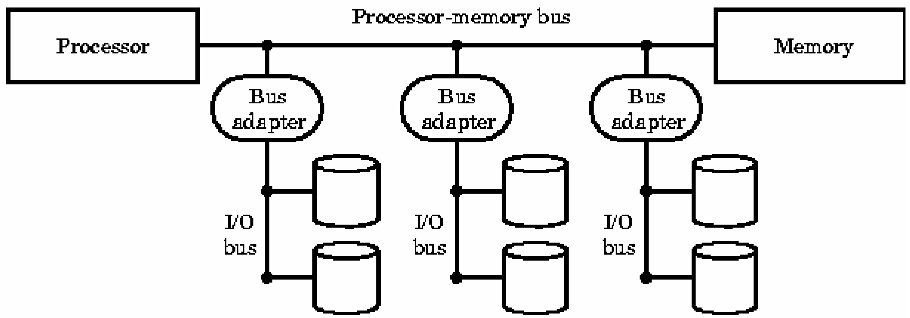
- CPU–memory buses:
  - High speed
  - Custom & proprietary
  - Matched to the memory system to maximize memory–CPU bandwidth
- I/O buses:
  - Long
  - Industry standard
  - Many types of devices connected
    - Wide range in the data bandwidth
- Backplane buses:
  - Backplane: an interconnection structure in the chassis
  - Allow memory, processor, I/O to coexist
  - Maybe proprietary or standard

Bus configurations

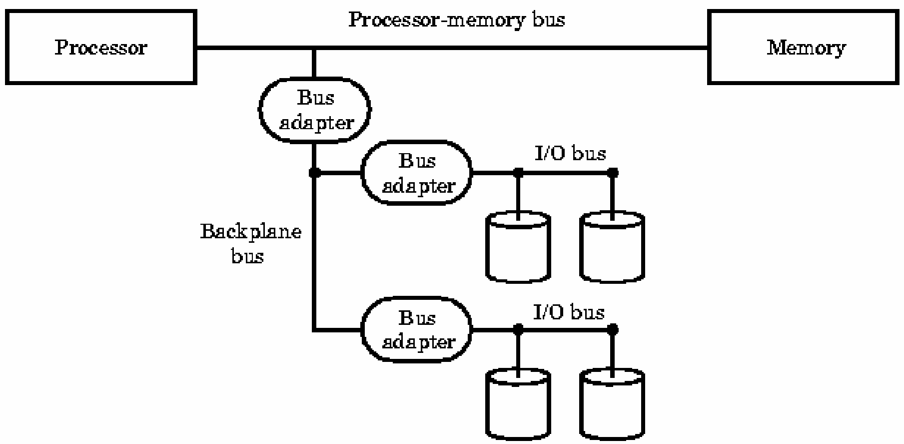
- Single bus
  - Cheap, but critical bottleneck
  - Obsolete



- Separate bus for memory and I/O traffic
  - Via bus adapters



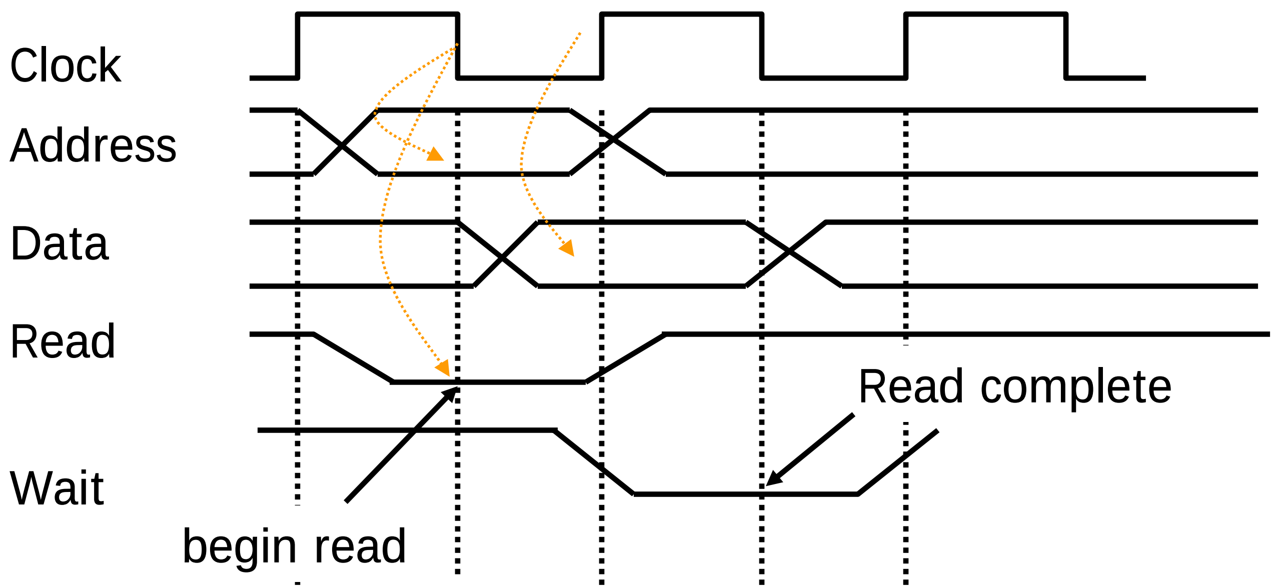
- All three types
  - Backplane bus connected via adapter to memory bus
  - Backplane bus connects via adapters other I/O buses



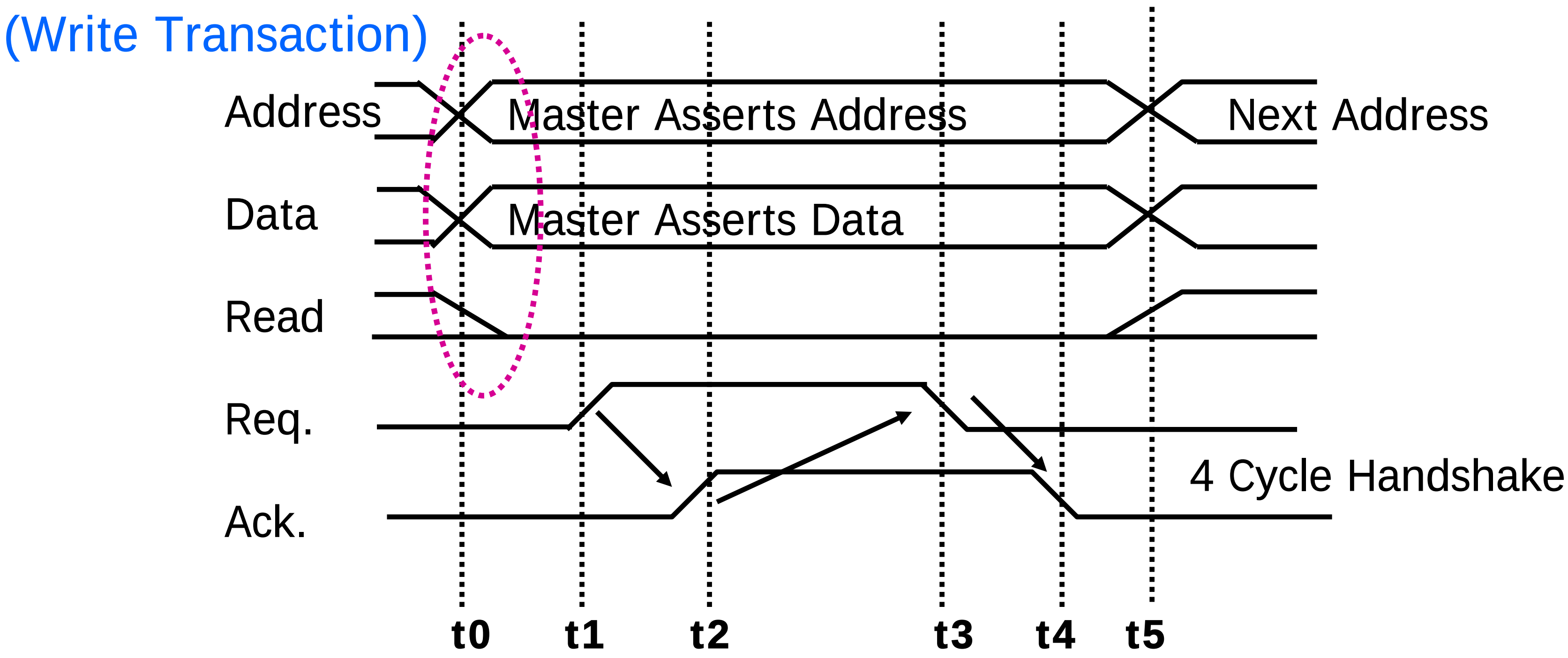
Bus clocking

- **Synchronous Bus**:
  - Includes a clock in the control lines
    - Defines bus cycle (= N clock cycles)
  - A fixed protocol for communication that is relative to the clock
  - Advantage:
    - Involves very little logic and can run very fast
  - Disadvantages:
    - Every device on the bus *must run at the same clock rate*
      - *Older devices may not work*
    - To avoid clock skew, they cannot be long if they are fast
- **Asynchronous Bus**:
  - Not clocked
  - Can accommodate a wide range of devices
  - Can be lengthened without worrying about clock skew
  - Requires a **handshaking protocol**

Synchronous Bus Protocols

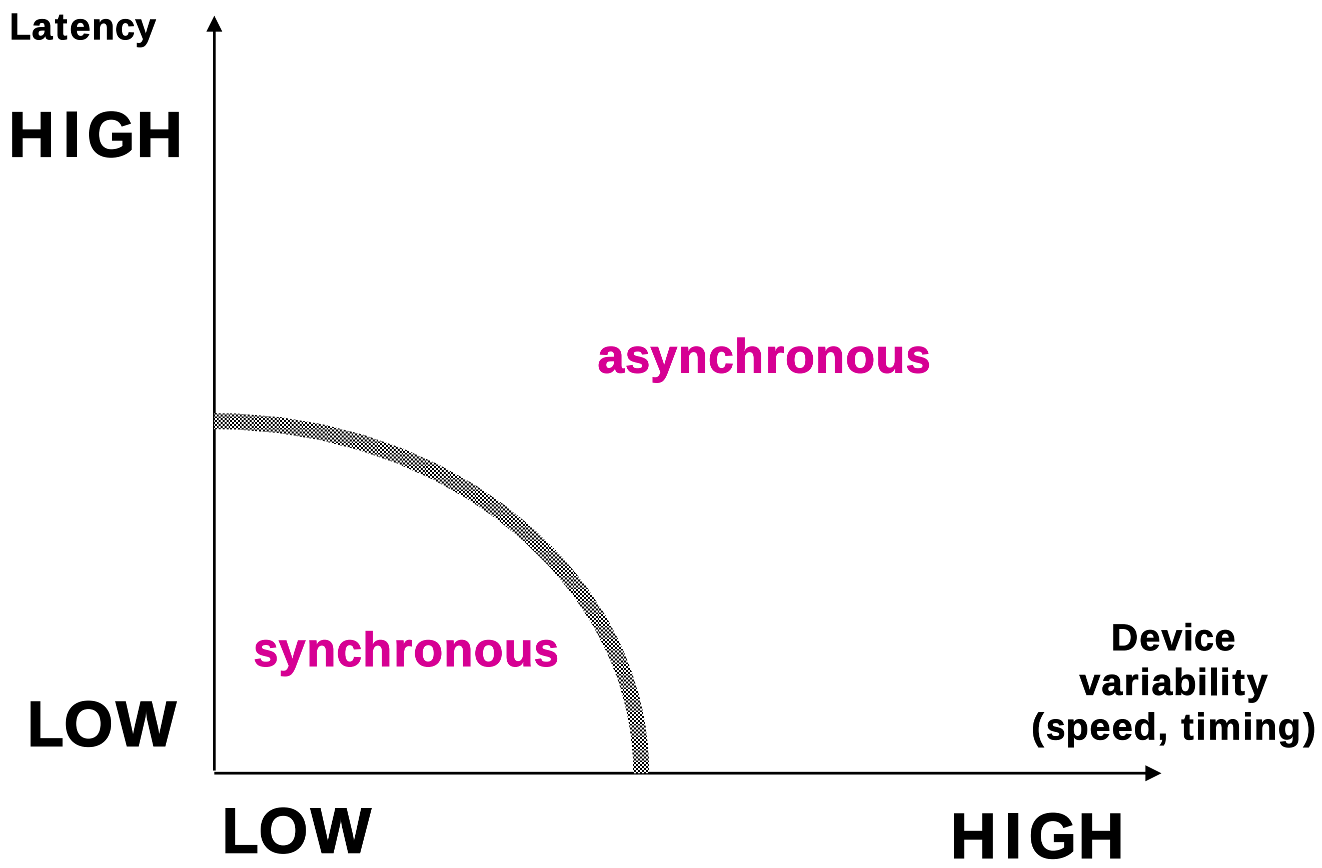


Asynchronous Handshake



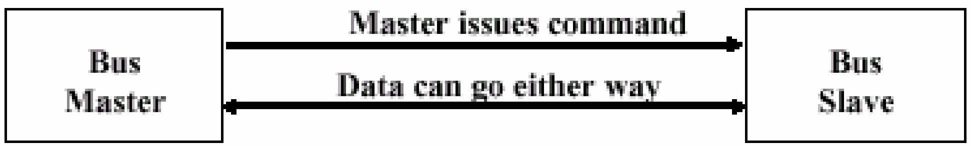
- t0: Master has obtained control and asserts address, direction, data. Waits a specified amount of time for slaves to decode target
- t1: Master asserts request line
- t2: Slave asserts ack, indicating data received
- t3: Master releases req
- t4: Slave releases ack

Synchronous vs. asynchronous



Bus transactions

- A bus transaction includes two parts:
  - Issuing the command (and address) – **request**
  - Transferring the data – **action**
- Master is the one who starts the bus transaction
  - Issues the command (and the address)
- Slave is the one who responds to the address:
  - Sends data to the master if the master ask for data
  - Receives data from the master if the master wants to send data



Arbitration

- Problem:
  - **How is the bus reserved by a devices that wishes to use it?**
- Chaos is avoided by a master-slave arrangement:
  - Only the bus master can control access to the bus:
  - A slave responds to read and write requests
- The simplest system:
  - Processor is the only bus master
  - All bus requests must be controlled by the processor
  - Drawback: the processor is involved in every transaction

Arbitration (2)

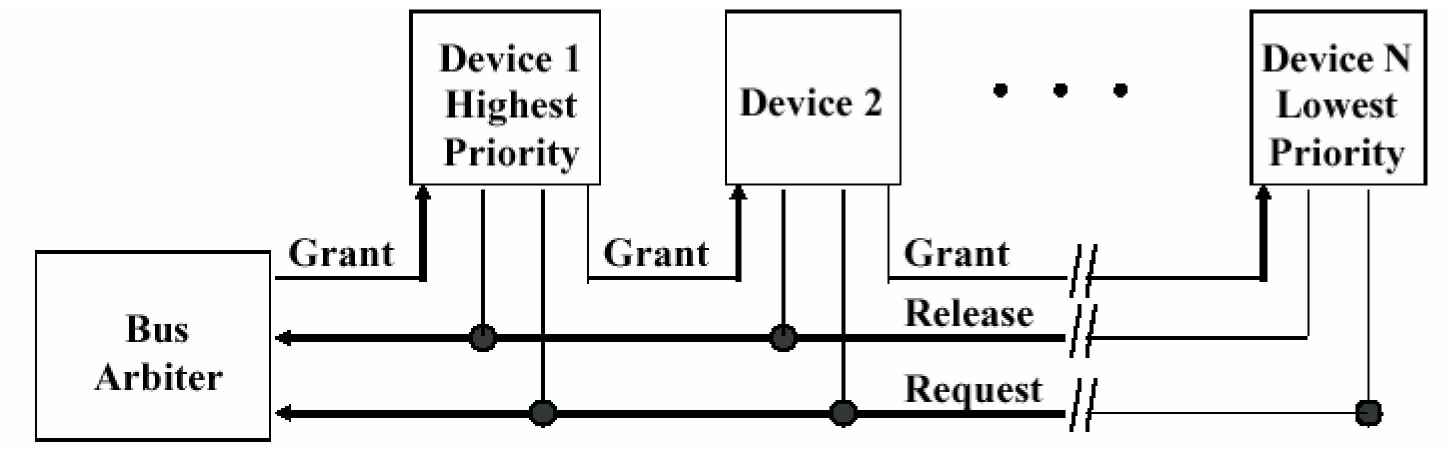
- Consequence: need multiple bus masters
  - Managing multiple masters requires **arbitration**!
- Arbitration goals:
  - Functionality
    - Prevent bus conflicts (two bussed simultaneous drivers)
  - Performance
    - Need to make decisions quickly
  - Priority
    - Some masters maybe more desperate than others
    - Example: DRAM refresh
  - Fairness
    - Every equal priority master should get equal service
    - No starvation: Every requestor should eventually get bus

Arbitration (3)

- Arbitration schemes:
  - Daisy chain arbitration:
    - single device with all request lines.
  - Centralized
  - Distributed arbitration by self-selection
    - each device wanting the bus places a code indicating its identity on the bus
    - Requires some sort of state duplication
  - Distributed arbitration by collision detection
    - e.g., as for Ethernet

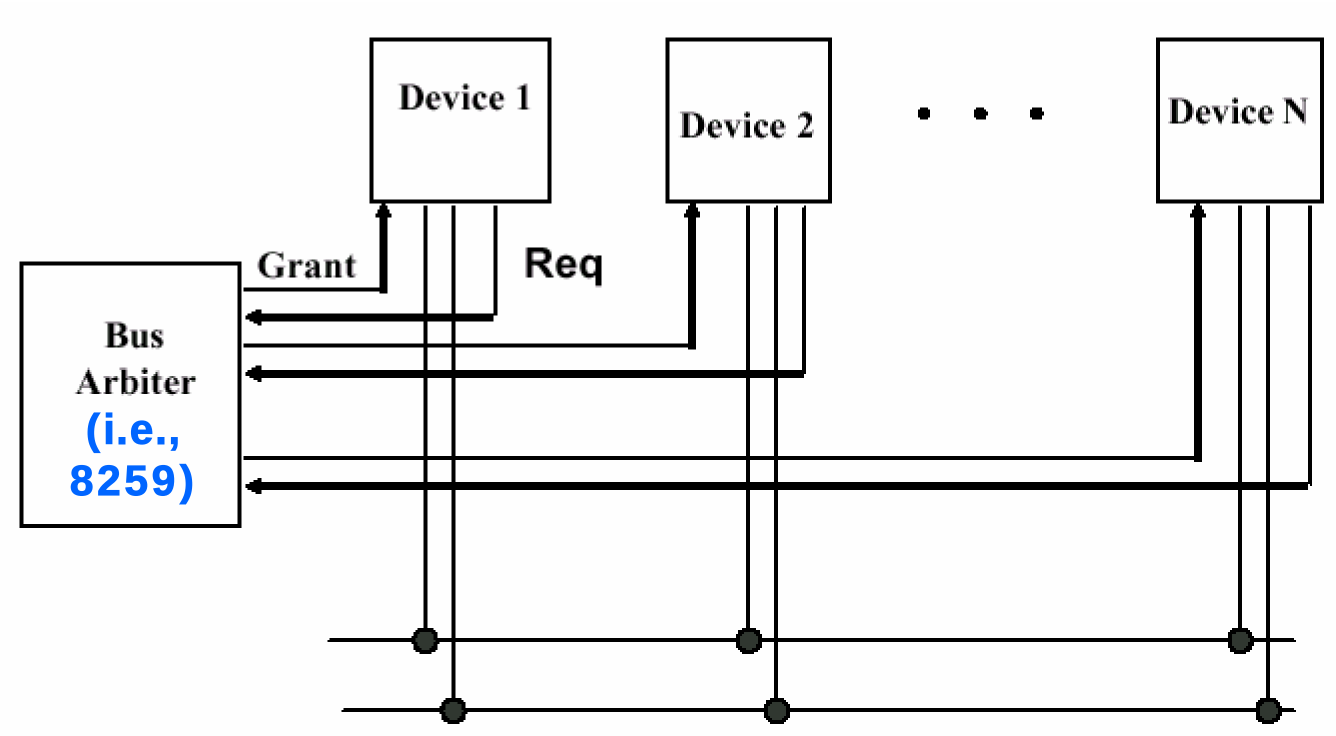
Daisy-chained arbitration

- Advantage: simple
- Disadvantages: *Cannot assure fairness*
  - A low-priority device may be locked out indefinitely
  - The use of the daisy chain grant signal also limits the bus speed



Parallel centralized arbitration

- Most used
  - Essentially standard for all processor-memory busses and in high-speed I/O busses



Bus performance optimization

- Improving **bandwidth**:
  - Separate vs. multiplexed address and bus lines
    - Address and data can be transmitted in one bus cycle if separate
  - Data bus width:
    - Transfers of multiple words require fewer bus cycles
  - Block transfers:
    - Allow the bus to transfer multiple words in back-to-back bus cycles
    - Only one address needs to be sent at the beginning

Bus performance optimization (2)

- Improving **transaction rate**:
  - Overlapped arbitration
    - Perform arbitration for next transaction during current transaction
  - *Bus parking*
    - Master can hold onto bus and performs multiple transactions as long as no other master makes request
  - Overlapped address / data phases
    - Requires one of the above techniques
  - Split-phase (or packet switched) bus
    - Completely separate address and data phases
    - Arbitrate separately each one

Split transaction bus (1)

- With multiple masters, a bus can offer higher bandwidth by going to packets, as opposed to holding the bus for a full transaction.
- This technique is called split transactions.
- The read transaction is now split into
  - A read request transaction that contains the address
  - A memory reply transaction that contains the data.

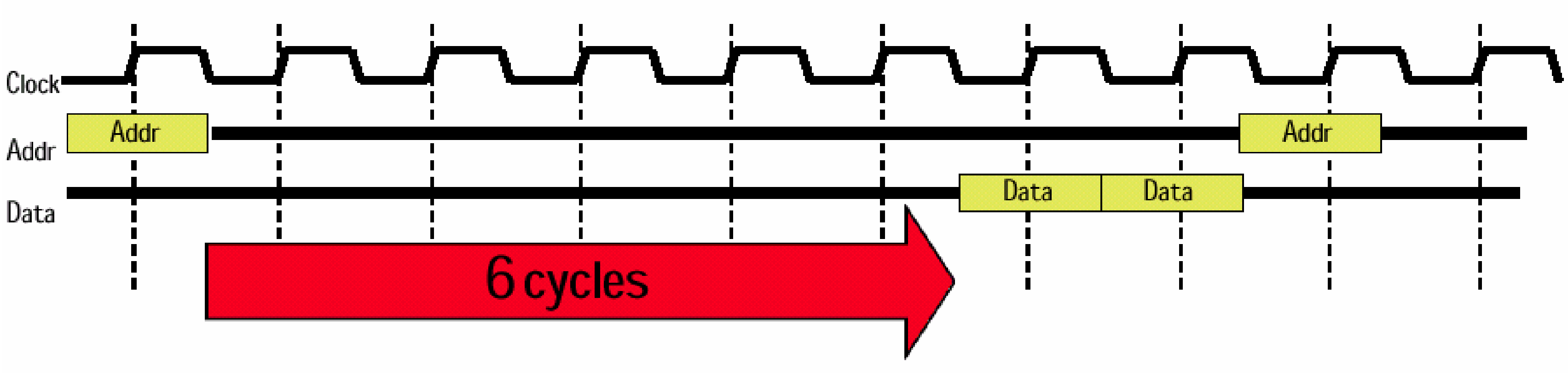
Split transaction bus (2)

- On a split transaction bus, each transaction must be **tagged** so that the processor and memory can tell what it is.
- Split transactions make the bus available for other masters while the memory reads the words from the requested address.
- The CPU must arbitrate for the bus in order to send the data and the memory must arbitrate in order to return the data.
  - Higher bandwidth
  - Higher latency

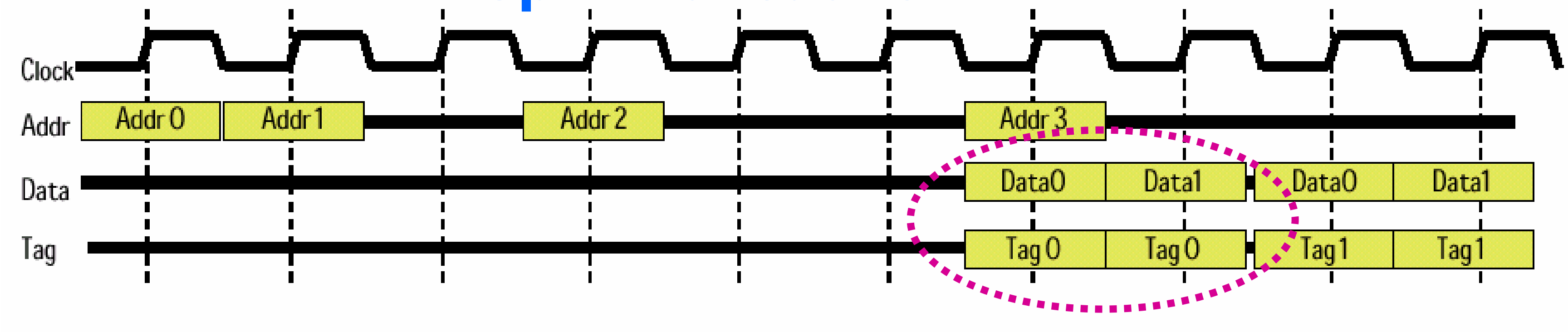
  than non-split buses

Split transactions

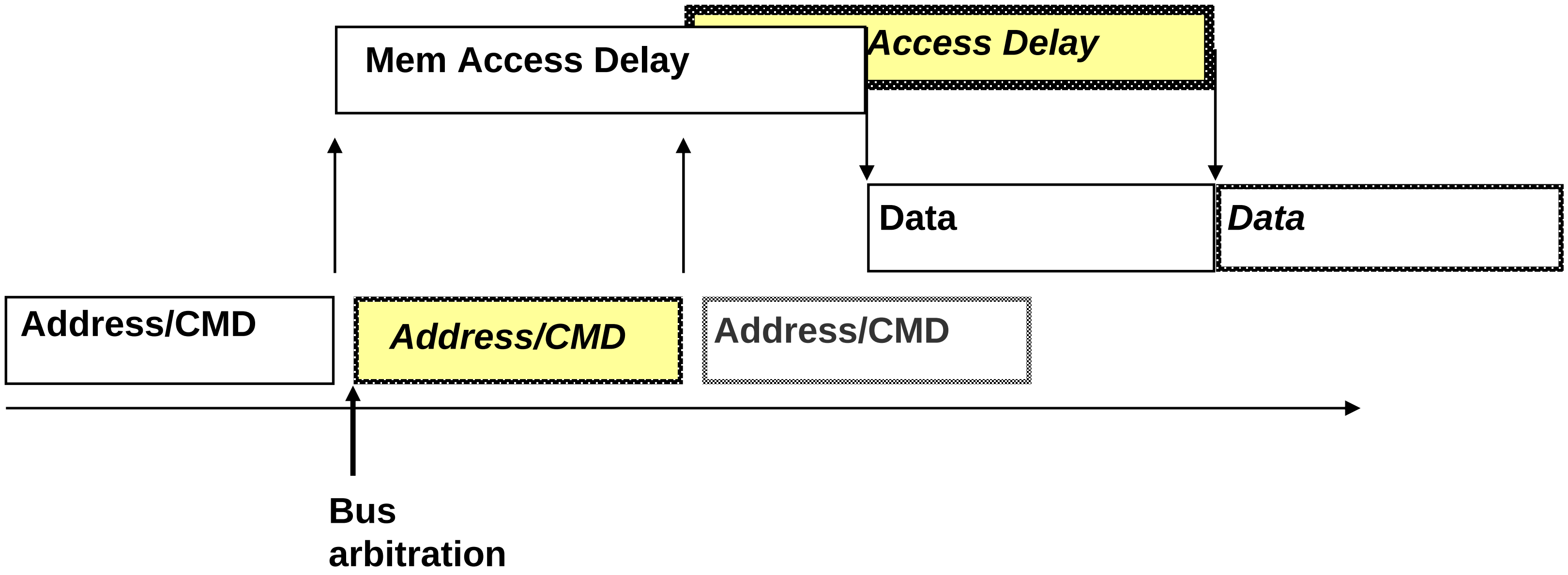
- Used to solve issue of long wait times
- Example:



Split transaction



Split transactions



Bus Options

Option	*High performance*	*Low cost*
Bus width	Separate address & data lines	Multiplex address & data lines
Data width	Wider is faster (e.g., 32 bits)	Narrower is cheaper (e.g., 8 bits)
Transfer size	Multiple words has less bus overhead	Single-word transfer is simpler
Bus masters	Multiple (requires arbitration)	Single master (no arbitration)
Split transaction?	**Yes:** separate Request and Reply packets gets higher bandwidth (needs multiple masters)	**No:** continuous connection is cheaper and has lower latency
Clocking	Synchronous	Asynchronous

1993 MP Memory Bus Survey

Bus	Summit	Challenge	XDBus
Originator	HP	SGI	Sun
Clock Rate (MHz)	60	48	66
Split transaction?	Yes	Yes	Yes?
Address lines	48	40	??
Data lines	128	256	144 (parity)
Data Sizes (bits)	512	1024	512
Clocks/transfer	4	5	4?
Peak (MB/s)	960	1200	1056
Master	Multi	Multi	Multi
Arbitration	Central	Central	Central
Addressing	Physical	Physical	Physical
Slots	16	9	10
Busses/system	1	1	2
Length	13 inches	12? inches	17 inches

I/O buses

- Designed to support wide variety of devices
  - Full set not know at design time
  - Allow data rate match between arbitrary speed devices
- Typically asynchronous
  - Modern I/O buses (especially for fast I/O) synchronous as well

1993 I/O Bus Survey (P&H)

Bus	EISA	TurboChannel	MicroChannel	PCI
Originator	Intel	DEC	IBM	Intel
Clock Rate (MHz)	8.33	12.5-25	async	33
Addressing	Virtual	Physical	Physical	Physical
Data Sizes (bits)	16,32	8,16,24,32	8,16,24,32,64	8,16,24,32,64
Master	Single	Single	Multi	Multi
Arbitration	Central	Central	Central	Central
32 bit read(MB/s)	33	25	20	33
Peak (MB/s)	?	84	75	111 (222)
Max Power (W)	?	26	13	25

1990 Bus survey (P&H)

Bus	VME	FutureBus	Multibus II	IPI	SCSI
Signals	128	96	96	16	8
Addr/Data mux	no	yes	yes	n/a	n/a
Data width	16-32	32	32	16	8
Masters	multi	multi	multi	single	multi
Clocking	Async	Async	Sync	Async	either
MB/s (0ns,word)	25	37	20	25	1.5 (a) 5 (s)
Max devices	21	20	21	8	7
Max meters	0.5	0.5	0.5	50	25

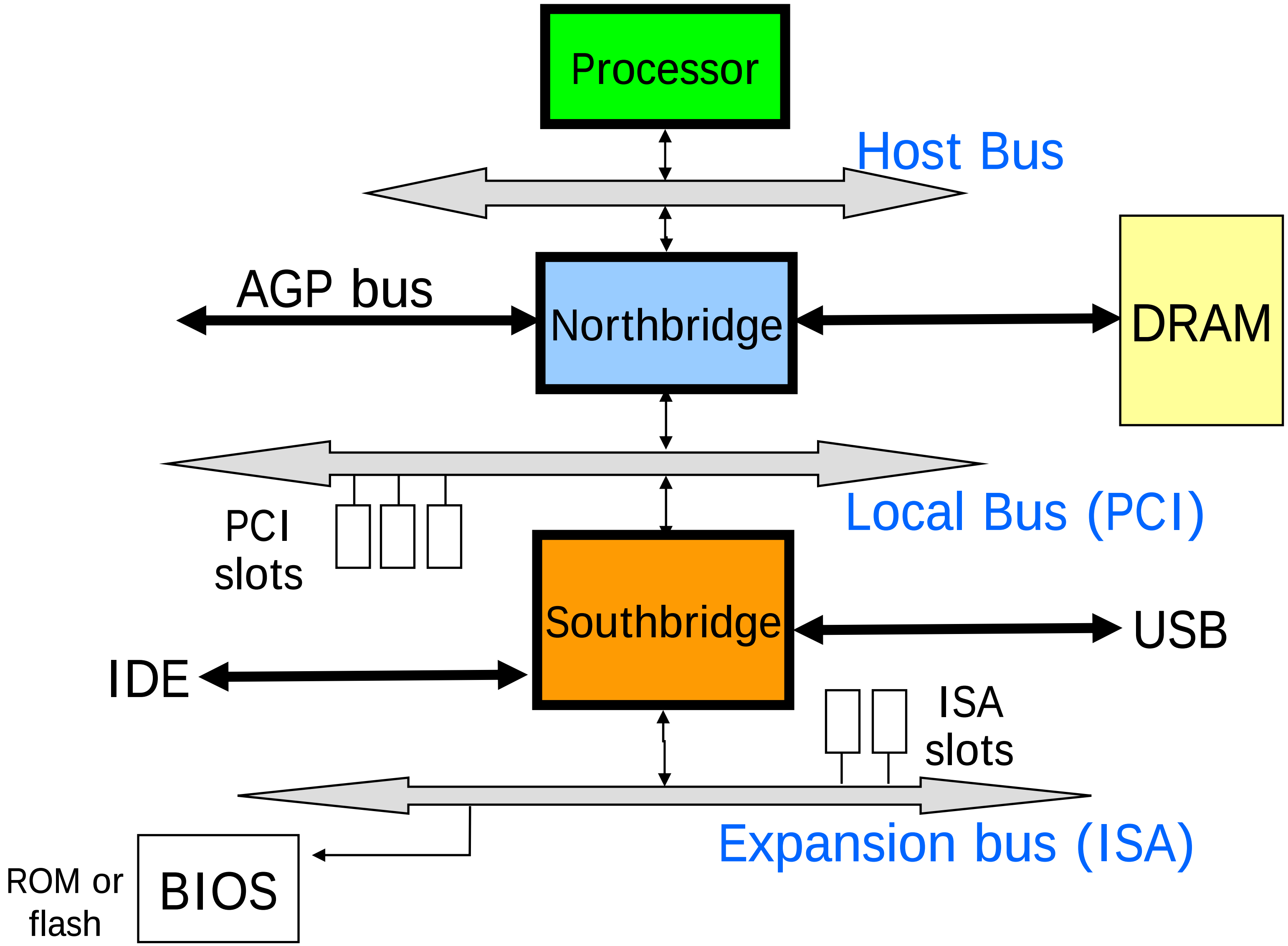
Modern bus architectures

- Modern architectures employ a hierarchical bus structure
  - Host bus
    - Processor/memory bus
    - No standard
  - Local bus
    - Fast peripherals
    - **PCI is the standard**
  - Expansion bus
    - Slow peripherals, i.e., true I/O bus
    - Corresponds to older "system bus"
    - **ISA is the standard**
- No clear notion of "backplane" bus

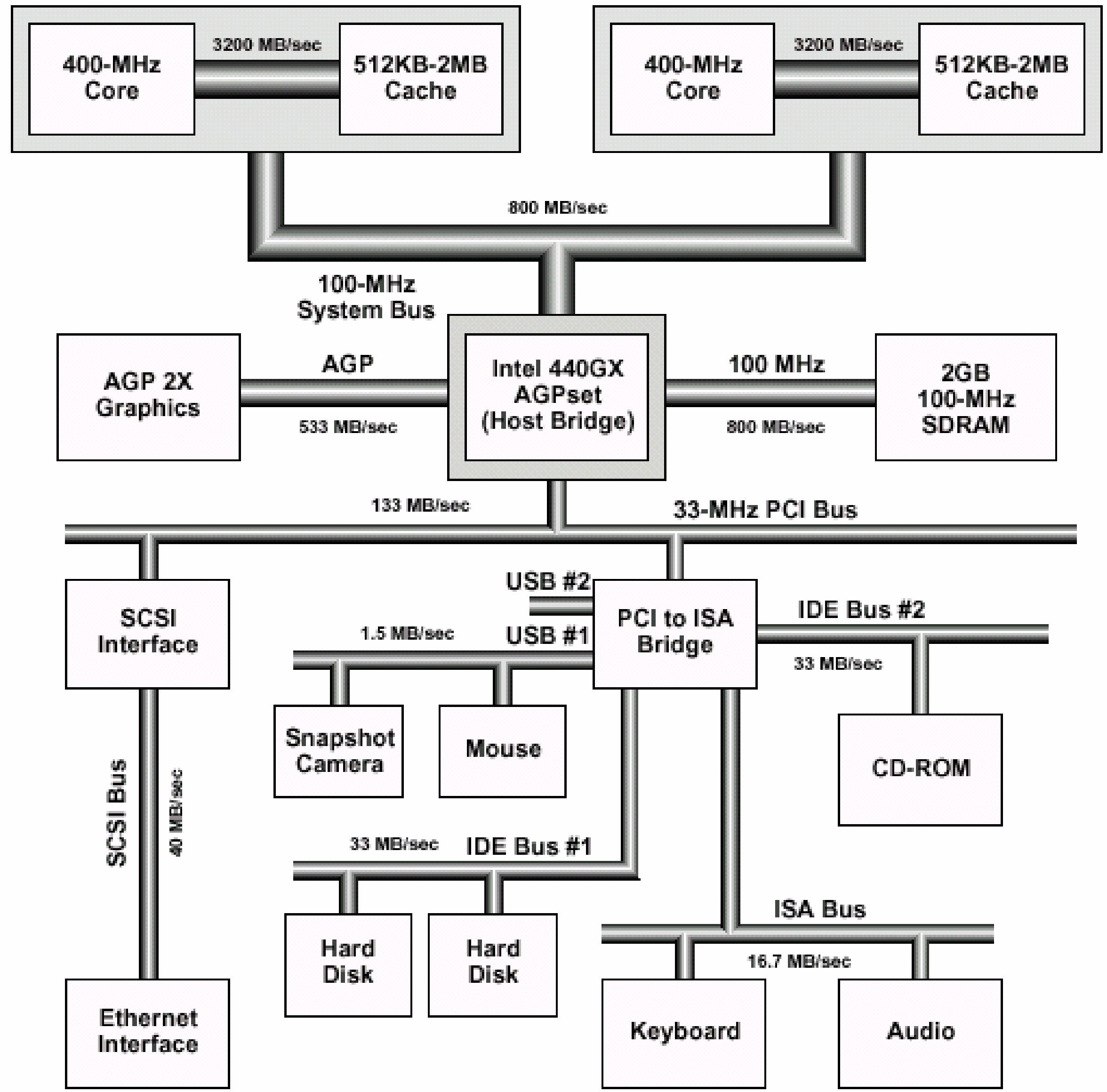
Modern bus architectures (2)

- Non-standard bus hosts force to use “adapters” for the lower bus levels
- Each CPU has a corresponding **chipset** (i.e., set of chips) that defines the interaction with the local and I/O buses
  - NOTE: Chipset™ is a registered mark...
  - Intel calls it PCIset (PCI is the backbone)
- Chipset (historically) consists of:
  - **Northbridge**: connects CPU to memory and fast I/O
  - **Southbridge**: connects mid bus to I/O devices

Northbridge and southbridge



Dual-Pentium II (Xeon) bus



Pentium chipsets

- With PentiumIII, Intel has moved towards a slightly different architecture
  - Memory controller hub (MCH, replaces NB)
  - I/O controller hub (ICH, replaces SB)
  - Firmware controller hub (FCH)
- Conceptually similar, but:
  - PCI is not central anymore
  - Connection MCH-ICH through a dedicated, 8-bit bus @266MHz (2x PCI)

Example chipset (PentiumIV)