Input-Output Subsystems

Mark McDermott

Agenda

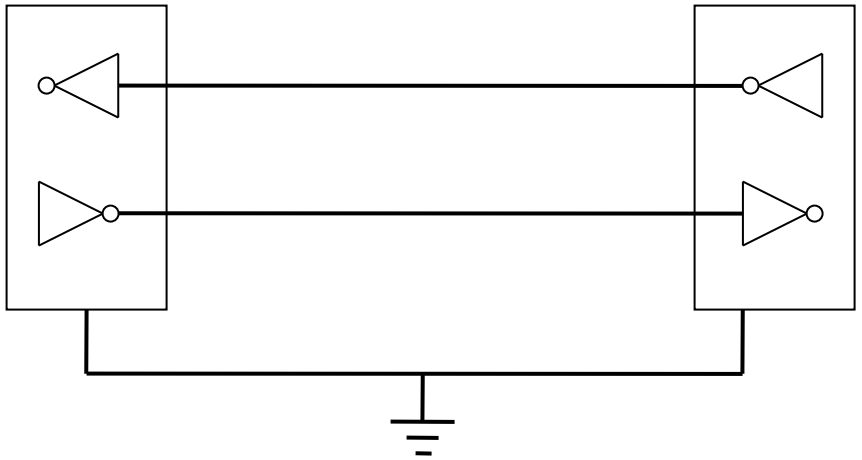
- I/O Signaling
- Parallel Bus architectures
- I/O Performance Measurements
- Examples of I/O Bus Standards
  - Serial bus
    - USB
    - CAN
    - RS-232
  - Parallel bus
    - AMBA
    - PCI

I/O Signaling

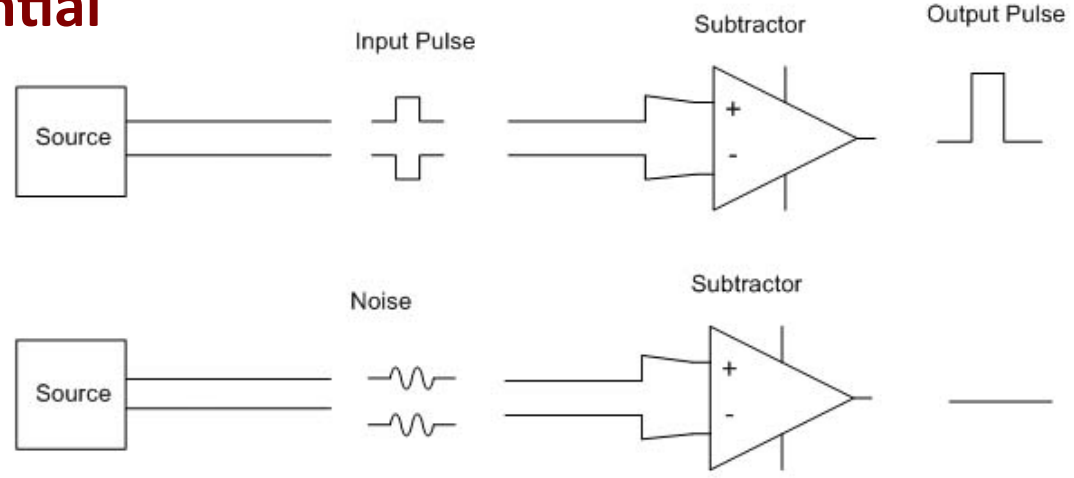
- There are basically two forms of signaling used for input/output applications
  - Single Ended
  - Differential
- In single-ended signaling one wire carries a varying voltage that represents the signal, while the other wire is connected to a reference voltage, usually ground.
  - Single ended signaling is less expensive to implement than differential, but its main limitations are that it lacks the ability to reject noise caused by differences in ground voltage level between transmitting and receiving circuits.
- Differential signaling uses two complementary signals sent on two separate wires.
  - Able to reject common-mode noise
  - More expensive to implement from both a wire perspective as well as the transmit & receive logic.

Single Ended vs. Differential Signaling

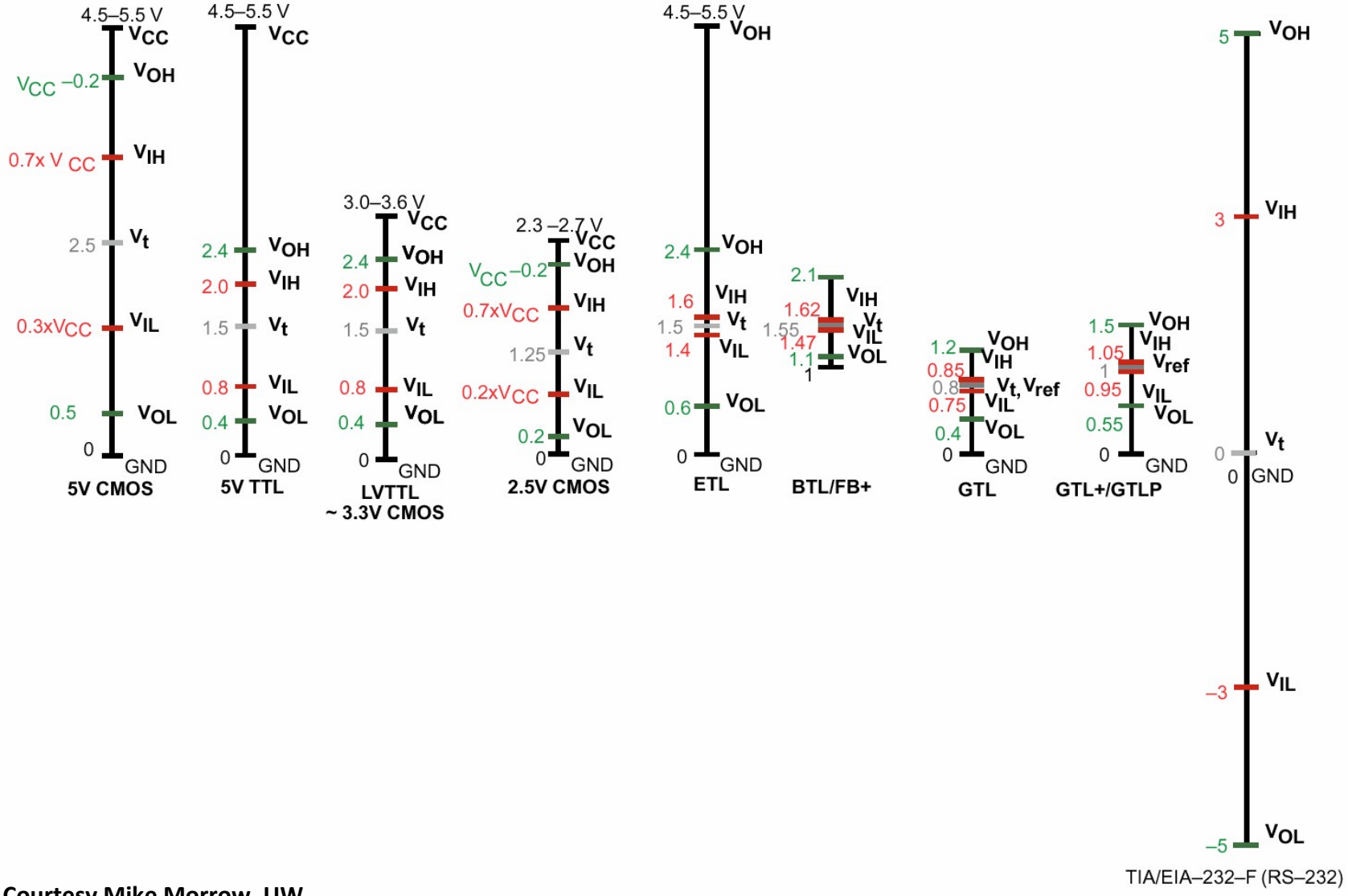
- **Single Ended**



- **Differential**

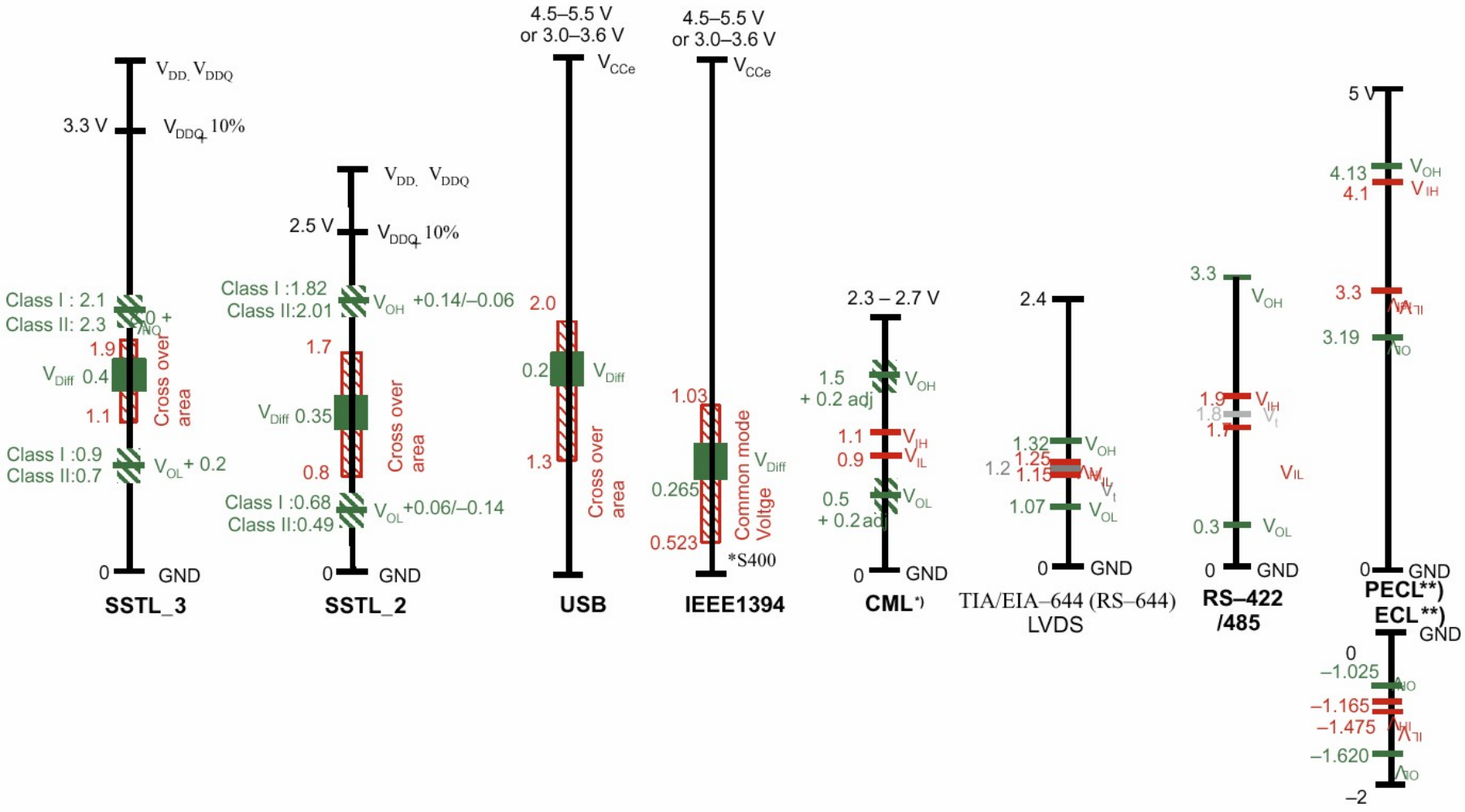


Single-ended Bus Signaling Standards



Courtesy Mike Morrow, UW

Differential Bus Signaling Standards



*) The DC offset (transmit Common Mode Voltage Range) for V /V_{OH} and V_{IL}/V_{IH} levels is adjustable in the full supply range

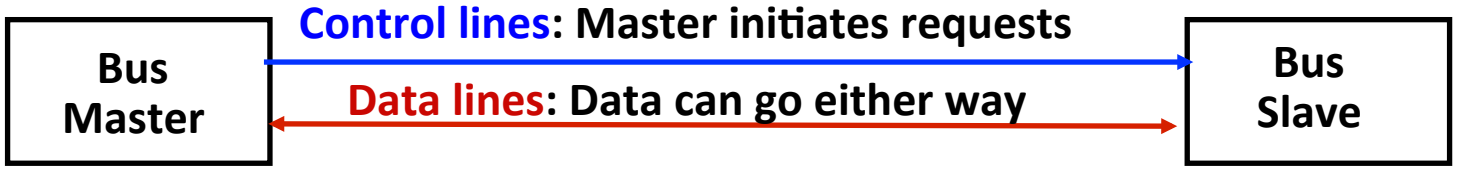
**) levels vary with family and among vendors

Courtesy Mike Morrow, UW

Bus Architecture

Bus Characteristics

- Control lines (side band signals)
  - Signal requests and acknowledgments
  - Indicate what type of information is on the data lines



- Data lines
  - Data, addresses, and complex commands
- Bus transaction consists of
  - Master issuing the command (and address) – request
  - Slave receiving (or sending) the data – action
  - Defined by what the transaction does to memory
    - Input – inputs data from the I/O device to the memory
    - Output – outputs data from the memory to the I/O device

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., ATA, PCIexpress)**
  - The backplane is an interconnection structure within the chassis
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus

Synchronous and Asynchronous Buses

- Synchronous bus (e.g., processor-memory buses)
  - Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    - Every device communicating on the bus must use same clock rate
    - To avoid clock skew, they cannot be long if they are fast

- Asynchronous bus (e.g., I/O buses)
  - It is not clocked, so requires a handshaking protocol and additional control lines
  - Advantages:
    - Can accommodate a wide range of devices and device speeds
    - Can be lengthened without worrying about clock skew or synchronization problems
  - Disadvantage: slow(er)
  - Two Transfer Methods
    - Strobe pulse
      - A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur
    - Handshaking
      - A control signal is accompanied with each data being transmitted to indicate the presence of data The receiving unit responds with another control signal to acknowledge receipt of the data

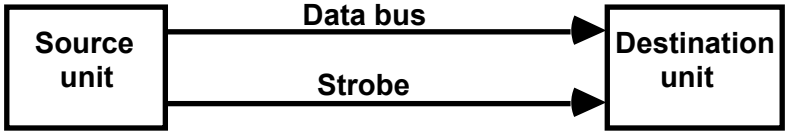
Hennessey & Patterson

Asynchronous Transfer Method: Strobe Pulse

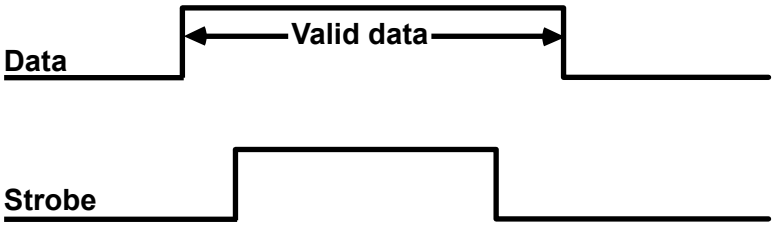
- Employs a single control line to time each transfer
- The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

Block Diagram

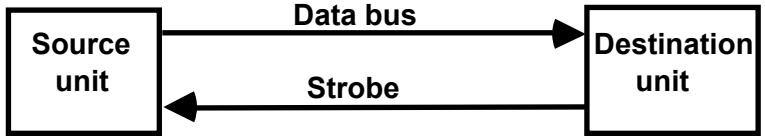


Timing Diagram

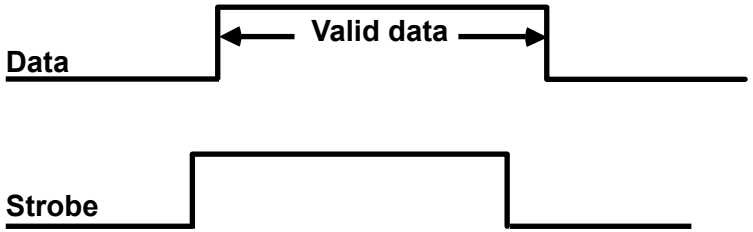


Destination-Initiated Strobe for Data Transfer

Block Diagram



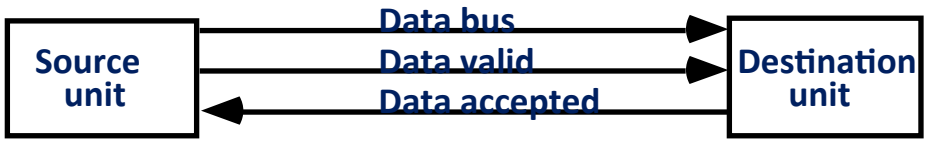
Timing Diagram



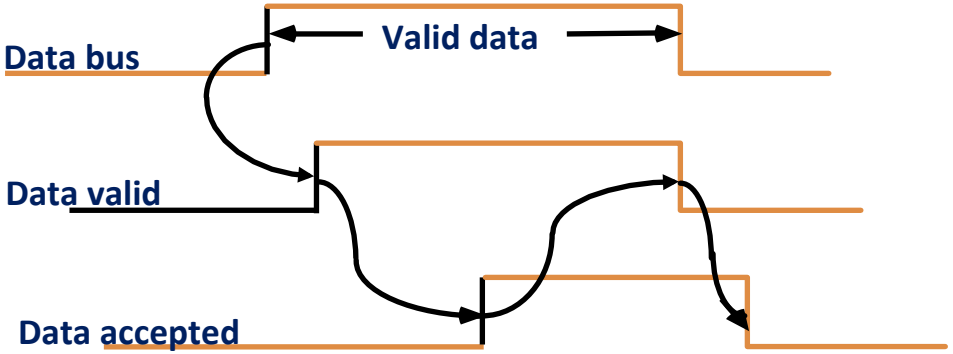
Hennessey & Patterson

Source-Initiated Transfer using Handshake Method

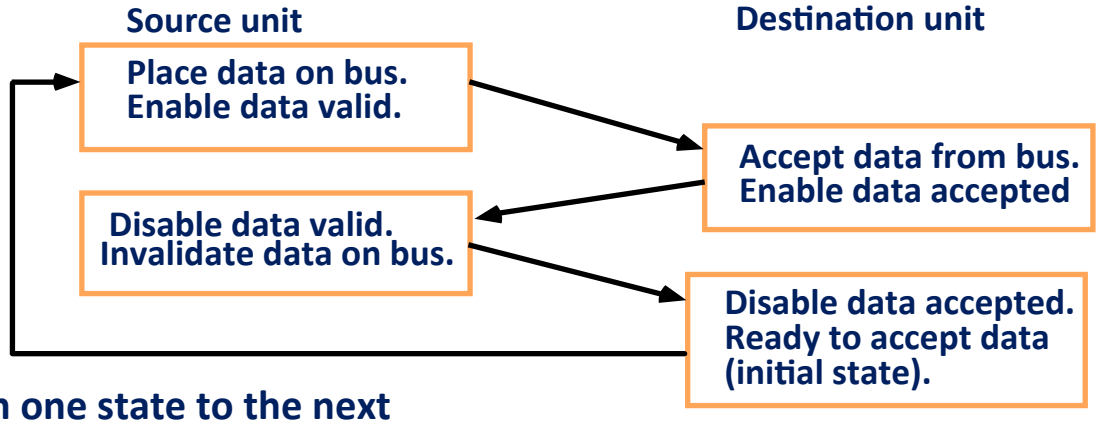
Block Diagram



Timing Diagram



Sequence of Events

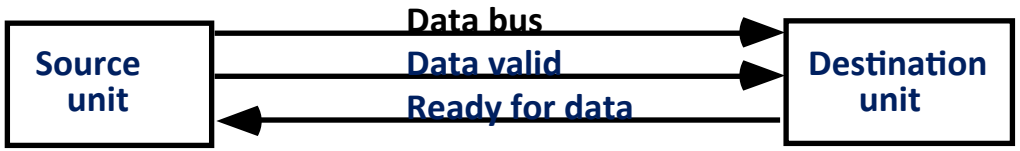


* Allows arbitrary delays from one state to the next
* Permits each unit to respond at its own data transfer rate
* The rate of transfer is determined by the slower unit

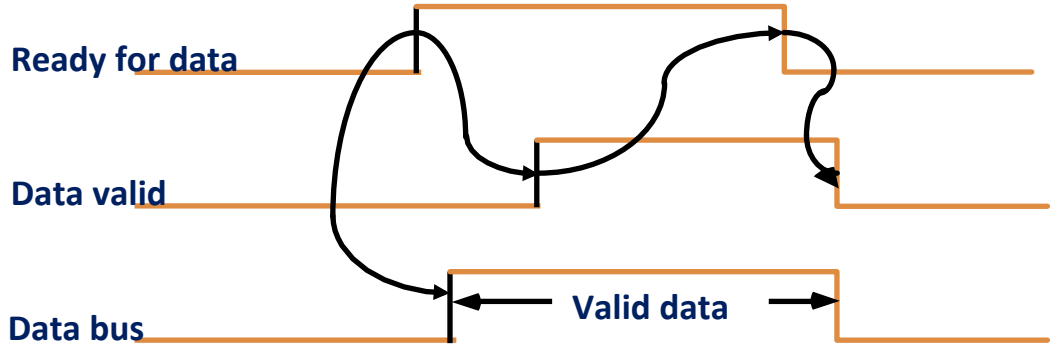
Hennessey & Patterson

Destination-Initiated Transfer using Handshake Method

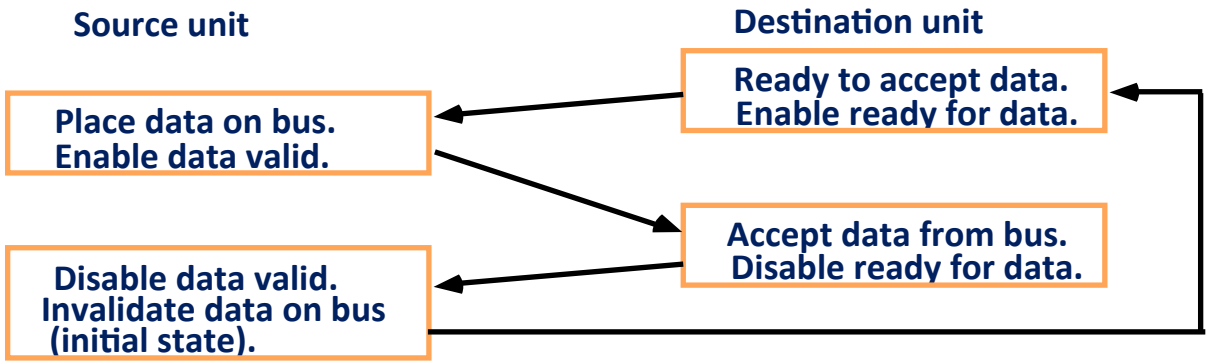
Block Diagram



Timing Diagram



Sequence of Events

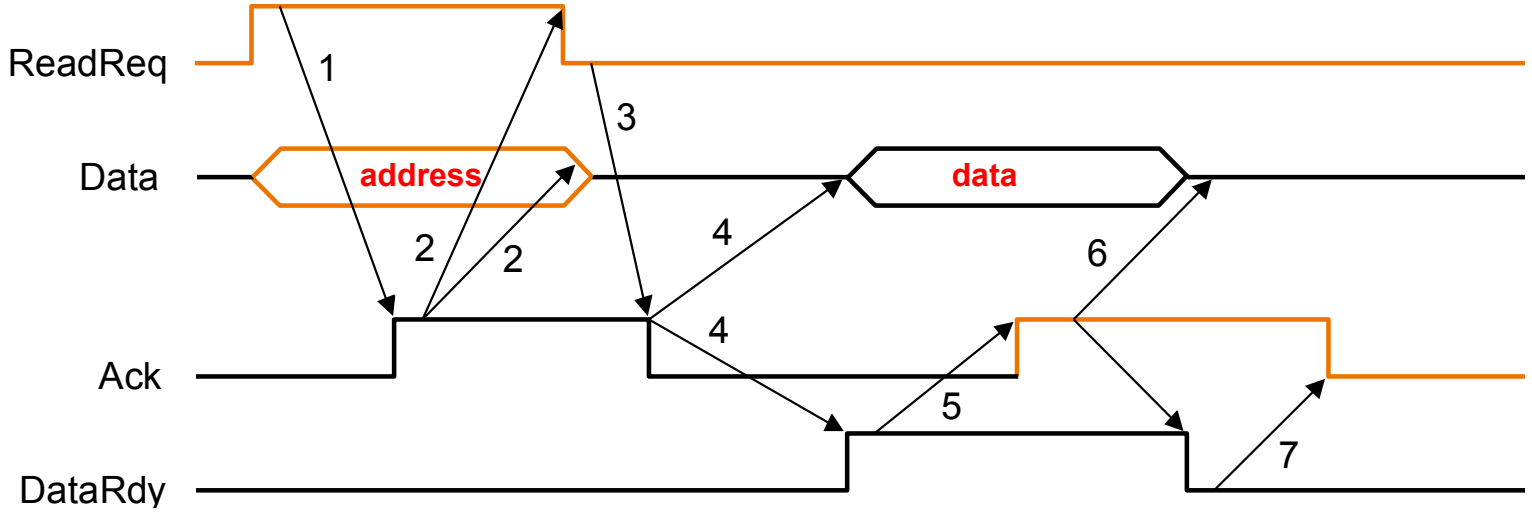


* Handshaking provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units
* If one unit is faulty, data transfer will not be completed
  -> Can be detected by means of a *timeout* mechanism

Hennessey & Patterson

Asynchronous Transfer Example

Output (read) data from memory to an I/O device



I/O device signals a request by raising ReadReq and putting the addr on the data lines

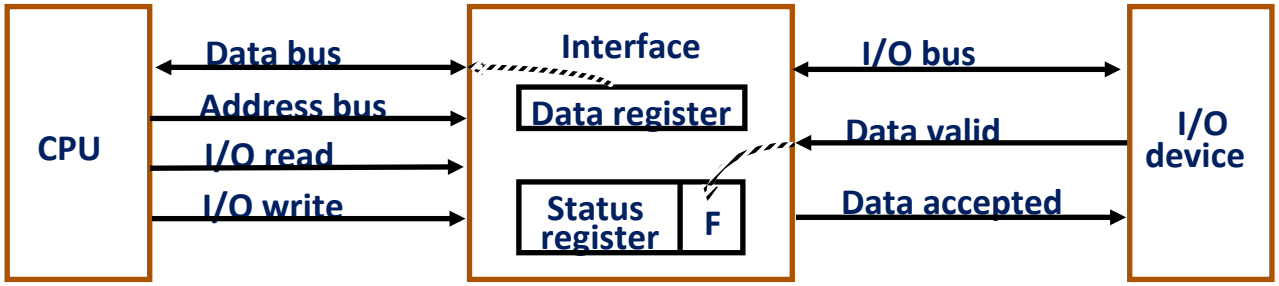
1. Memory sees ReadReq, reads addr from data lines, and raises Ack
2. I/O device sees Ack and releases the ReadReq and data lines
3. Memory sees ReadReq go low and drops Ack
4. When memory has data ready, it places it on data lines and raises DataRdy
5. I/O device sees DataRdy, reads the data from data lines, and raises Ack
6. Memory sees Ack, releases the data lines, and drops DataRdy
7. I/O device sees DataRdy go low and drops Ack

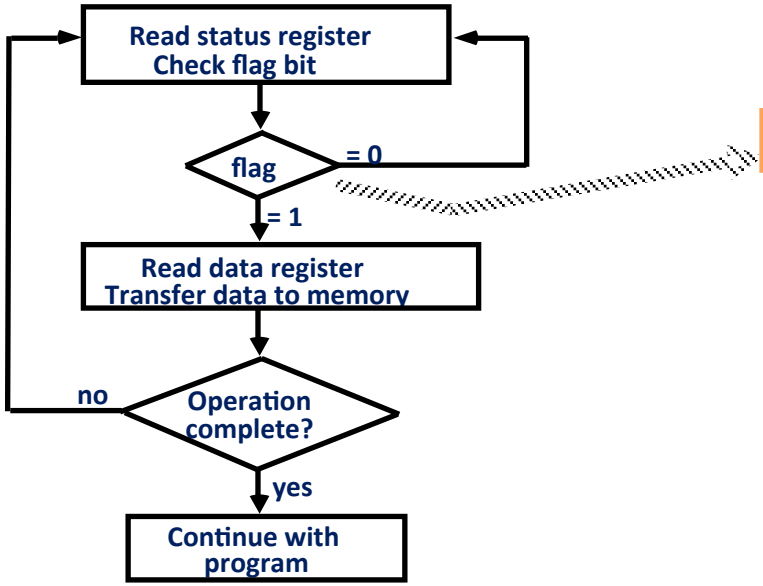
MODES OF TRANSFER

- 3 different Data Transfer Modes between the CPU or Memory and peripherals;
  - Program-Controlled I/O
  - Interrupt-Initiated I/O
  - Direct Memory Access (DMA)

Programmed Control I/O

- Processor performs all I/O functions in software.



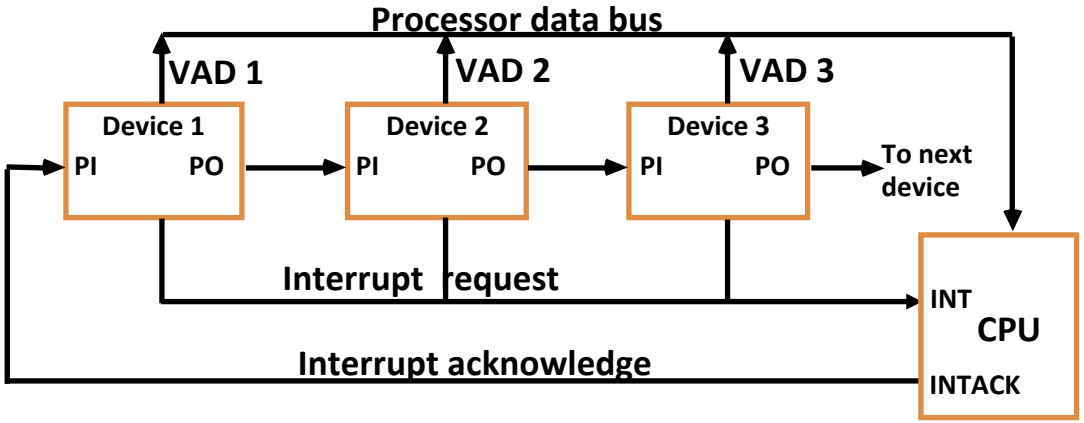


Polling or Status Checking

- Continuous CPU involvement
- CPU slowed down to I/O speed
- Simple
- Least hardware

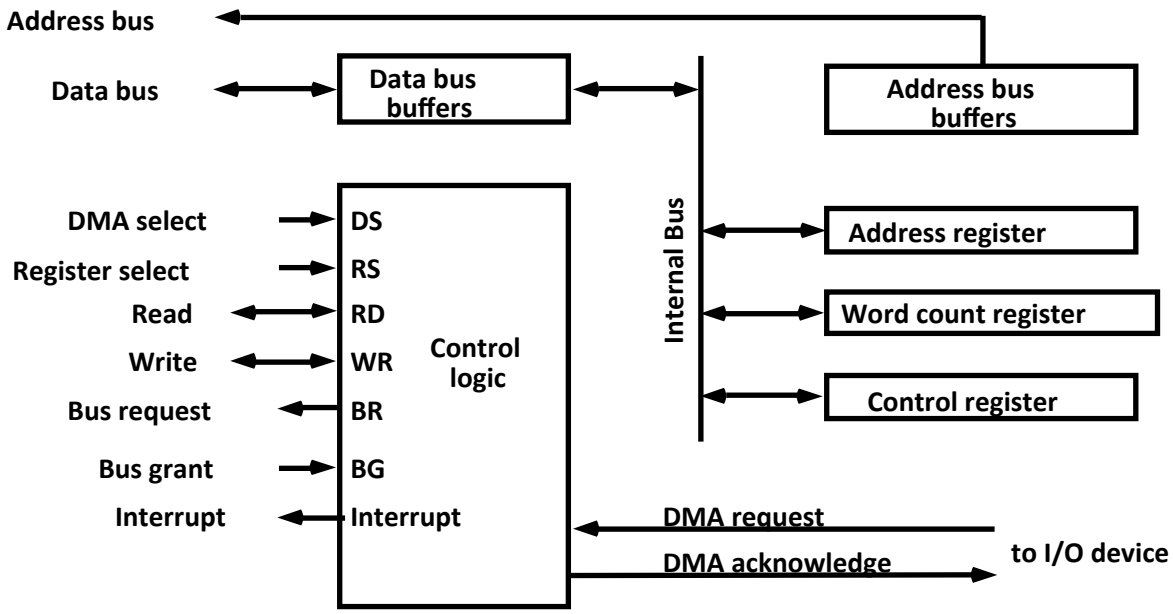
Interrupt Initiated I/O

- Since polling takes valuable CPU time open communication only when some data has to be passed -> Interrupt.
  - I/O interface, instead of the CPU, monitors the I/O device
  - When the interface determines that the I/O device is ready for data transfer, it generates an Interrupt Request to the CPU
  - Upon detecting an interrupt, CPU stops the task it is doing, branches to the interrupt service routine to process the data transfer, and then returns to the task it was performing



DMA Initiated I/O

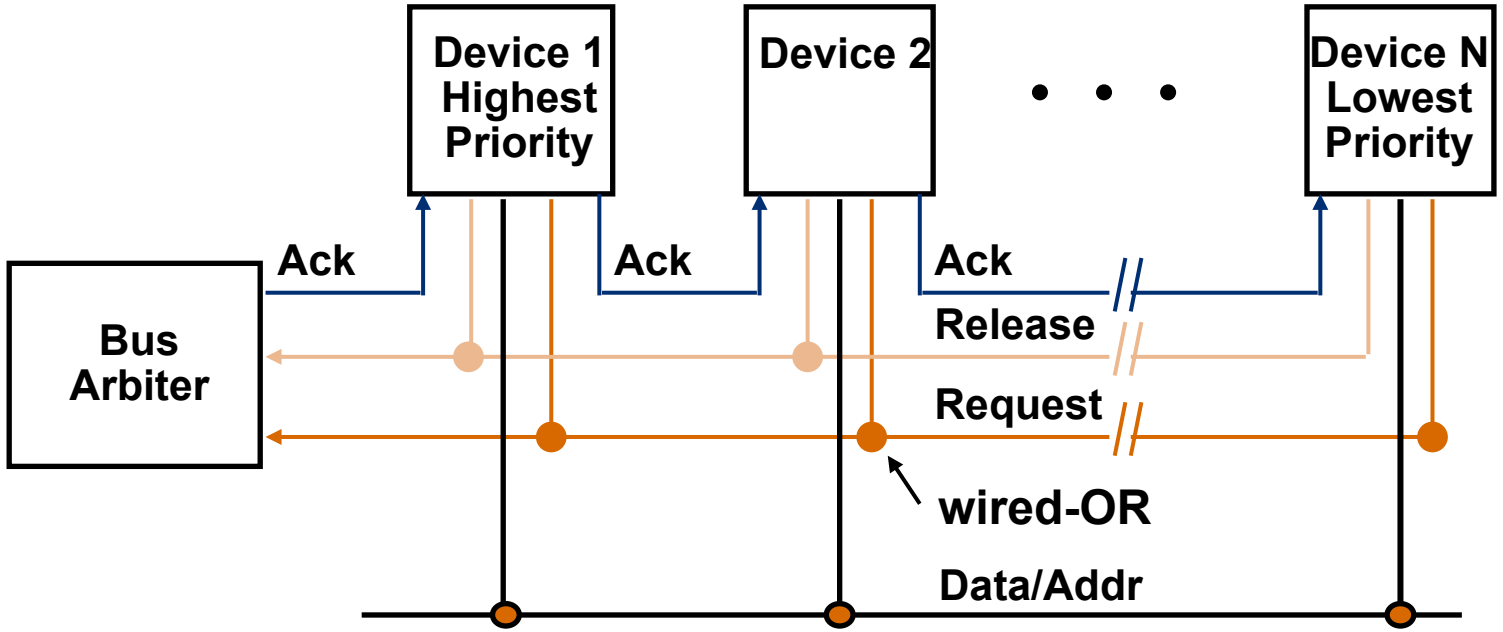
- DMA unit controls the I/O transfer of data directly to and from the memory and the I/O device
  - CPU initializes the DMA controller by sending a memory address and the number of words to be transferred
  - Actual transfer of data is done directly between the device and memory through DMA controller freeing CPU for other tasks



The Need for Bus Arbitration

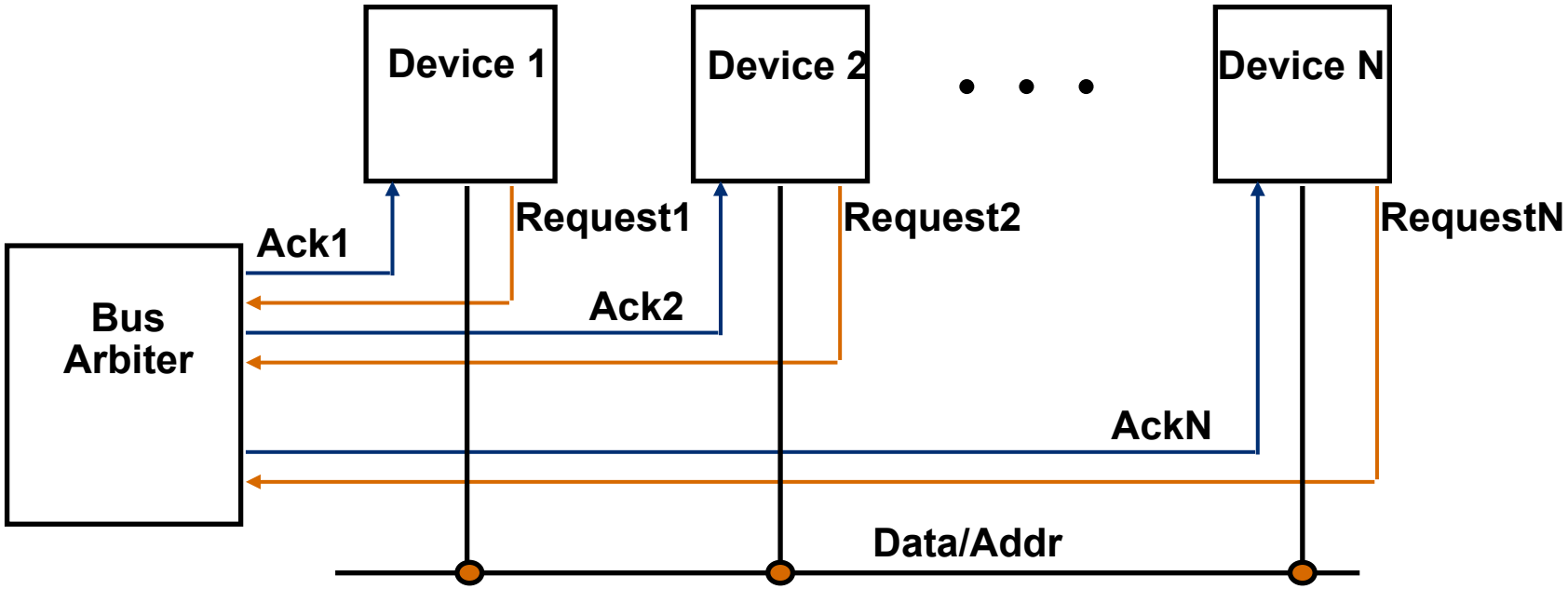
- Multiple devices may need to use the bus at the same time so must have a way to arbitrate multiple requests
- Bus arbitration schemes usually try to balance:
  - Bus priority – the highest priority device should be serviced first
  - Fairness – even the lowest priority device should never be completely locked out from the bus
- Bus arbitration schemes can be divided into four classes
  - Daisy chain arbitration
  - Centralized, parallel arbitration
  - Distributed arbitration by self-selection – each device wanting the bus places a code indicating its identity on the bus
  - Distributed arbitration by collision detection – device uses the bus when its not busy and if a collision happens (because some other device also decides to use the bus) then the device tries again later (Ethernet)

Daisy Chain Bus Arbitration



- Advantage: simple
- Disadvantages:
  - Cannot assure fairness – a low-priority device may be locked out indefinitely
  - Slower – the daisy chain grant signal limits the bus speed

Centralized Parallel Arbitration



- Advantages: flexible, can assure fairness
- Disadvantages: more complicated arbiter hardware
- Used in essentially all processor-memory buses and in high-speed I/O buses

Bus Bandwidth Determinates

- **The bandwidth of a bus is determined by**
  - **Whether it is synchronous or asynchronous and the timing characteristics of the protocol used**
  - **The data bus width**
  - **Whether the bus supports block transfers or only word at a time transfers**

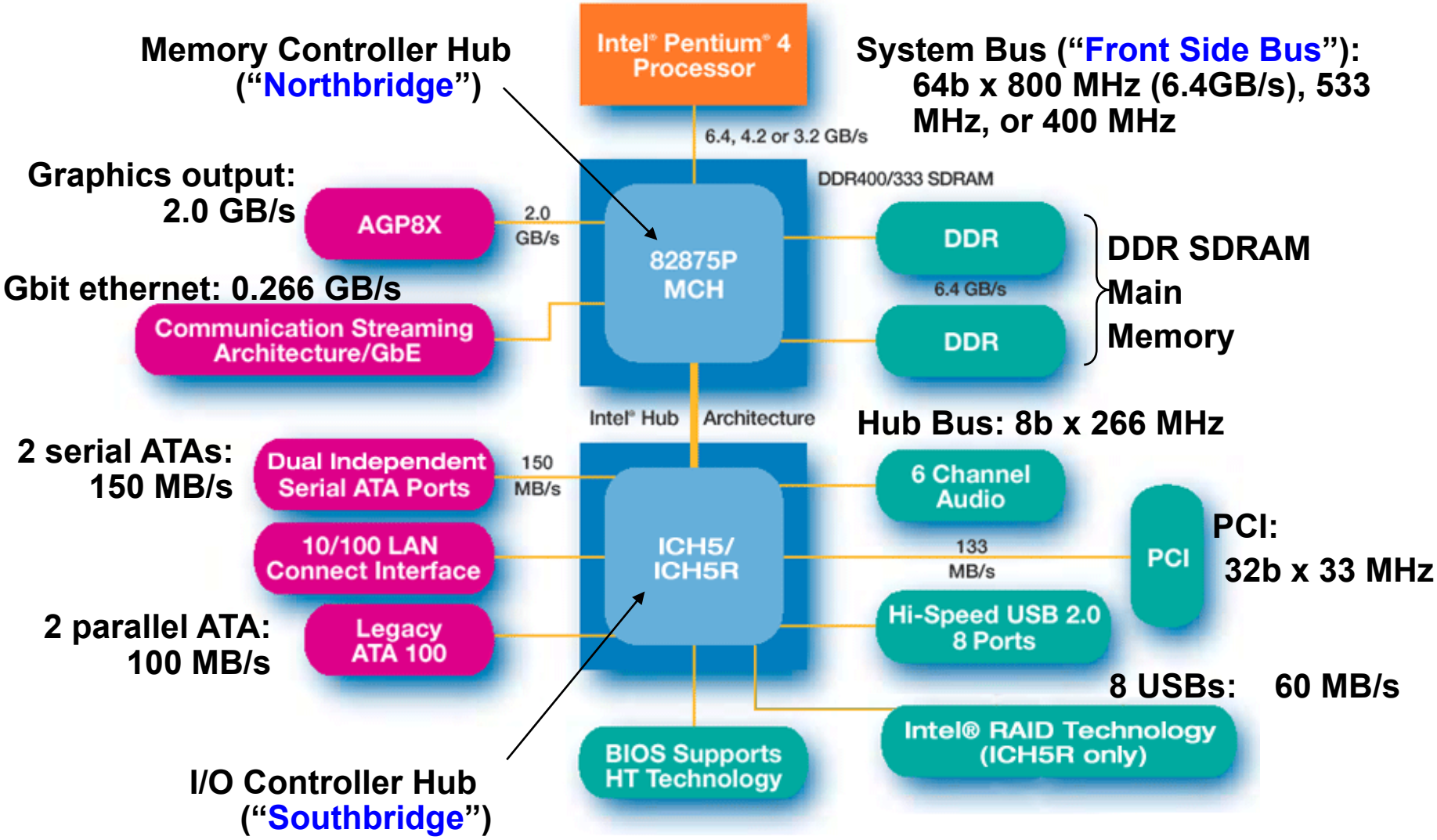
	Firewire	USB 2.0
Type	I/O	I/O
Data lines	4	2
Clocking	Asynchronous	Asynchronous
Max # devices	63	127
Max length	4.5 meters	5 meters
Peak bandwidth	50 MB/s (Firewire 400) 100 MB/s (Firewire 800)	0.2 MB/s (low speed) 1.5 MB/s (full speed) 60 MB/s (high speed)

Buses in Transition

- **Companies are transitioning from synchronous, parallel, *wide* buses to asynchronous *narrow* buses**
  - ***Reflections*** **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	PCIexpress	ATA	Serial ATA
Total # wires	120	36	80	7
# data wires	32 – 64 (2-way)	2 x 4 (1-way)	16 (2-way)	2 x 2 (1-way)
Clock (MHz)	33 – 133	635	50	150
Peak BW (MB/s)	128 – 1064	300	100	375 (3 Gbps)

Example: The Pentium 4's Buses



Hennessey & Patterson

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	100.0000-1000.0000
Magnetic disk	storage	machine	240.0000-2560.0000

I/O Performance Measurements

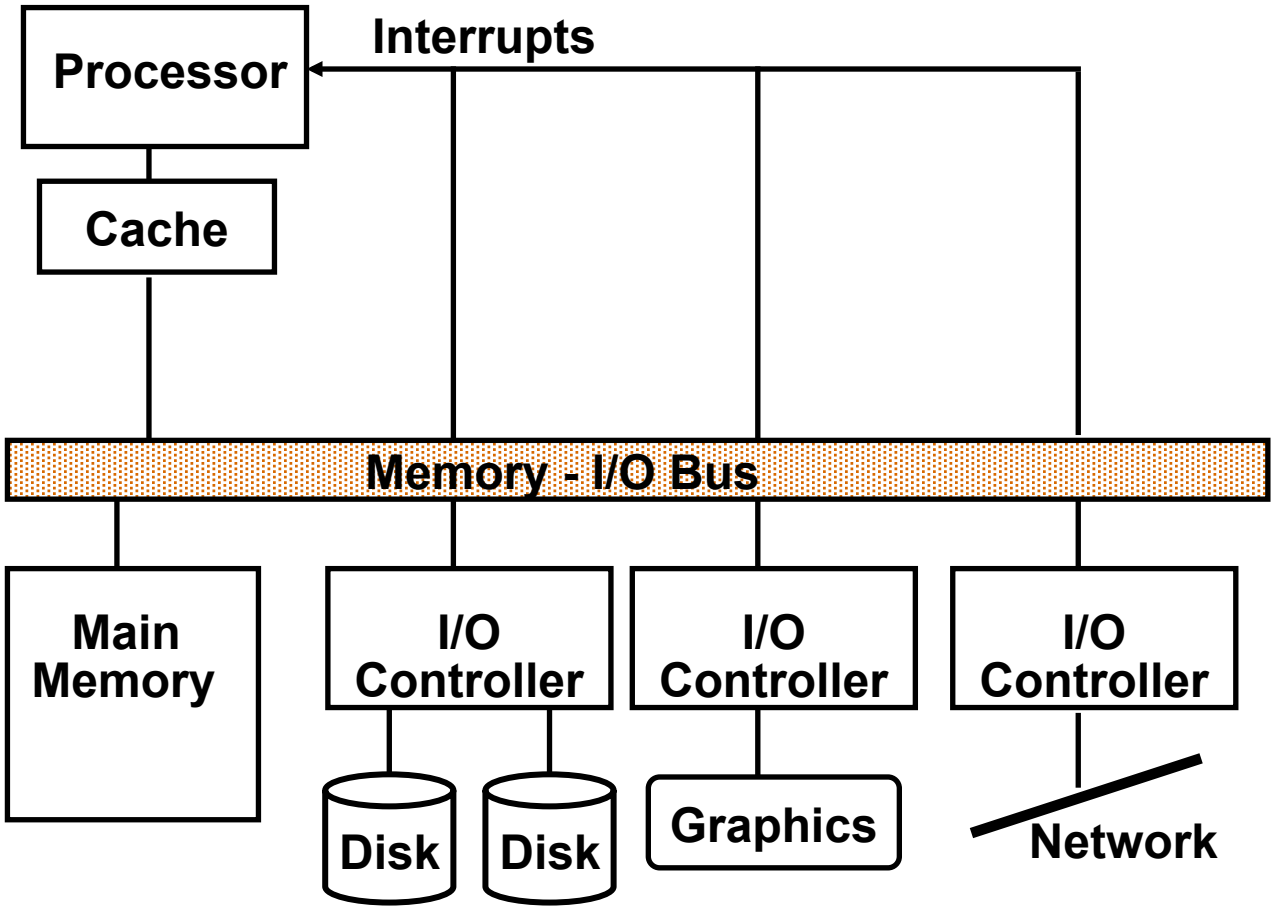
I/O Performance Measures

- 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
  - How much data can we move through the system in a certain time?
  - 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

A Typical I/O System



I/O System Performance

- Designing an I/O system to meet a set of bandwidth and/or latency constraints means
- Finding the weakest link in the I/O system – the component that constrains the design
  - The processor and memory system ?
  - The underlying interconnection (e.g., bus) ?
  - The I/O controllers ?
  - The I/O devices themselves ?
- (Re)configuring the weakest link to meet the bandwidth and/or latency requirements
- Determining requirements for the rest of the components and (re)configuring them to support this latency and/or bandwidth

I/O System Performance Example

A disk workload consisting of 64KB reads and writes where the user program executes 200,000 instructions per disk I/O operation and

1. a processor that sustains 3 billion inst/s and averages 100,000 OS instructions to handle a disk I/O operation

   The maximum disk I/O rate (# I/O's/s) of the processor is

$$\frac{\text{Inst execution rate}}{\text{Inst per I/O}} = \frac{3 \times 10^9}{(200 + 100) \times 10^3} = 10{,}000 \text{ I/O's/s}$$

2. a memory-I/O bus that sustains a transfer rate of 1000 MB/s

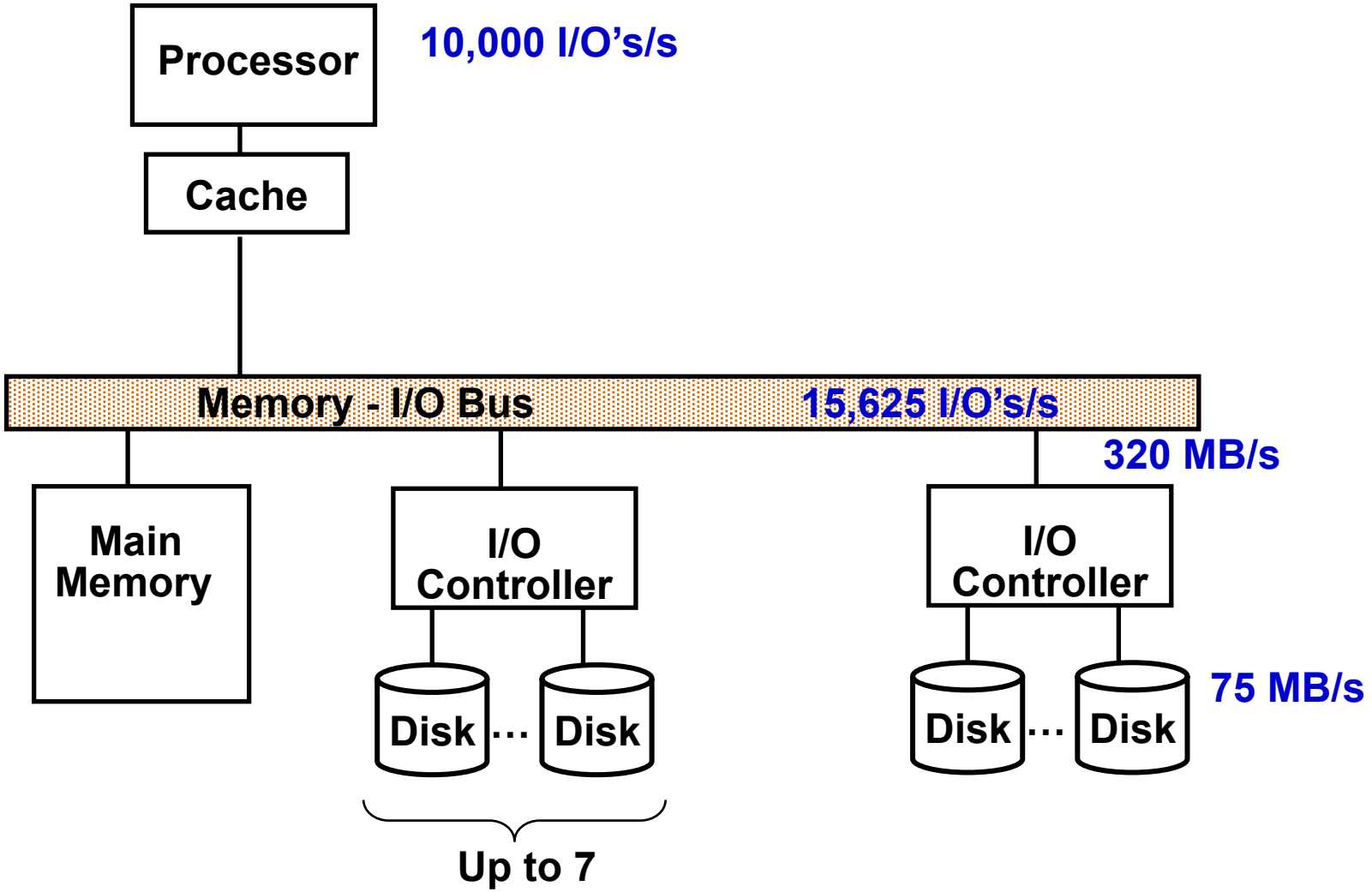
   Each disk I/O reads/writes 64 KB so the maximum I/O rate of the bus is

$$\frac{\text{Bus bandwidth}}{\text{Bytes per I/O}} = \frac{1000 \times 10^6}{64 \times 10^3} = 15{,}625 \text{ I/O's/s}$$

3. SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller
4. disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms

Hennessey & Patterson

Disk I/O System Example



Hennessey & Patterson

I/O System Performance Example (cont)

So the processor is the bottleneck, not the bus

disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms

Disk I/O read/write time = seek + rotational time + transfer time =
6ms + 64KB/(75MB/s) = 6.9ms

Thus each disk can complete 1000ms/6.9ms or 146 I/O's per second. To saturate the processor requires 10,000 I/O's per second or
10,000/146 = 69 disks

To calculate the number of SCSI disk controllers, we need to know the average transfer rate per disk to ensure we can put the maximum of 7 disks per SCSI controller and that a disk controller won't saturate the memory-I/O bus during a DMA transfer

Disk transfer rate = (transfer size)/(transfer time) = 64KB/6.9ms = 9.56 MB/s

Thus 7 disks won't saturate either the SCSI controller (with a maximum transfer rate of 320 MB/s) or the memory-I/O bus (1000 MB/s). This means we will need 69/7 or 10 SCSI controllers.

Hennessey & Patterson

I/O System Interconnect Issues

- 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

Communication of I/O Devices and Processor

- How the processor directs the I/O devices
  - Special I/O instructions
    - Must specify both the device and the command
  - 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
    - Load/stores to the I/O address space can only be done by the OS
- 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 I/O – the I/O device issues an interrupts to the processor to indicate that it needs attention

Interrupt-Driven I/O

- An I/O interrupt is asynchronous WRT instruction execution
  - Is not associated with any instruction so doesn't prevent any instruction from completing
    - You can pick your own convenient point to handle the interrupt
- With I/O interrupts
  - Need a way to identify the device generating the interrupt
  - Can have different urgencies (so may need to be prioritized)
- Advantages of using interrupts
  - Relieves the processor from having to continuously poll for an I/O event; user program progress is only suspended during the actual transfer of I/O data to/from user memory space
- Disadvantage – special hardware is needed to
  - Cause an interrupt (I/O device) and detect an interrupt and save the necessary information to resume normal processing after servicing the interrupt (processor)

Hennessey & Patterson

Direct Memory Access (DMA)

- For high-bandwidth devices (like disks) interrupt-driven I/O would consume a lot of processor cycles
- DMA – the I/O controller has the ability to transfer data directly to/from the memory without involving the processor
  - The processor initiates the DMA transfer by supplying the I/O device address, the operation to be performed, the memory address destination/ source, the number of bytes to transfer
  - The I/O DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
  - When the DMA transfer is complete, the I/O controller interrupts the processor to let it know that the transfer is complete
- There may be multiple DMA devices in one system
  - Processor and I/O controllers contend for bus cycles and for memory

The DMA Stale Data Problem

- In systems with caches, there can be two copies of a data item, one in the cache and one in the main memory
  - For a DMA read (from disk to memory) – the processor will be using stale data if that location is also in the cache
  - For a DMA write (from memory to disk) and a write-back cache – the I/O device will receive stale data if the data is in the cache and has not yet been written back to the memory
- The coherency problem is solved by
  - Routing all I/O activity through the cache – expensive and a large negative performance impact
  - Having the OS selectively invalidate the cache for an I/O read or force write-backs for an I/O write (flushing)
  - Providing hardware to selectively invalidate or flush the cache – need a hardware snooper

I/O and the Operating System

- The operating system acts as the interface between the I/O hardware and the program requesting I/O
  - To protect the shared I/O resources, the user program is not allowed to communicate directly with the I/O device

- Thus OS must be able to give commands to I/O devices, handle interrupts generated by I/O devices, provide equitable access to the shared I/O resources, and schedule I/O requests to enhance system throughput
  - I/O interrupts result in a transfer of processor control to the supervisor (OS) process

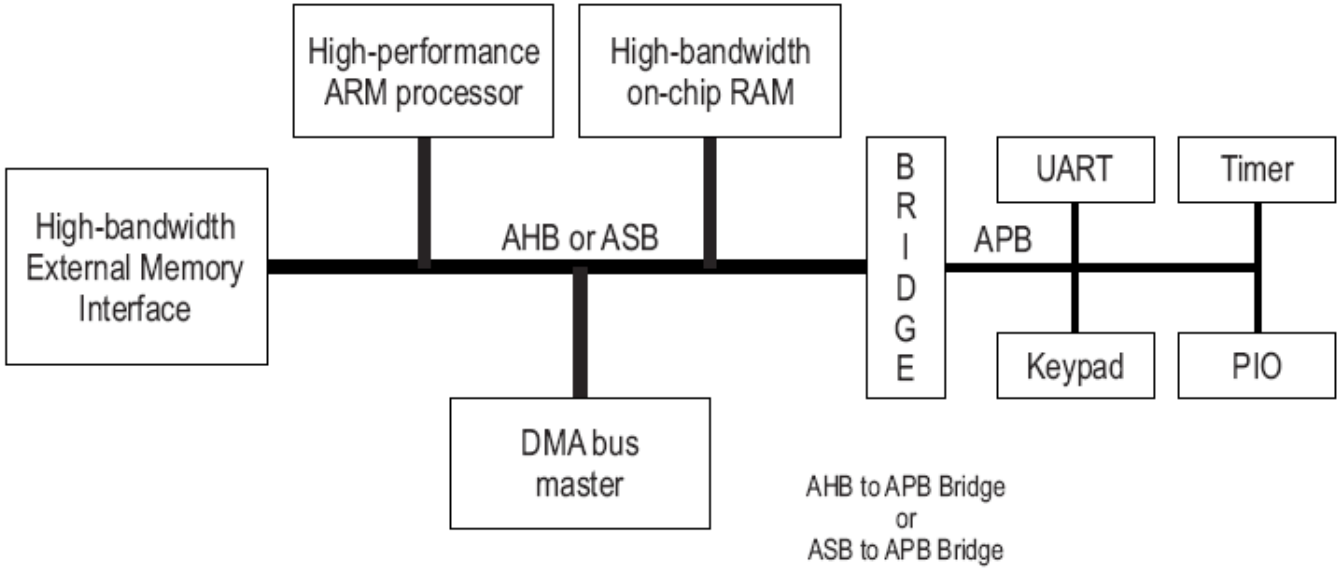
Parallel bus protocols

- AMBA
- PCI-X

AMBA Introduction

- Advanced Microcontroller Bus Architecture (AMBA), created by ARM as an interface for their microprocessors.
- Easy to obtain documentation (free download) and can be used without royalties.
- Very common in commercial SoC's
  (e.g. Qualcomm Multimedia Cellphone SoC)
- AMBA 2.0 released in 1999, includes APB and AHB
- AMBA 3.0 released in 2003, includes AXI

AMBA 2.0 System-Level View



AMBA AHB

* High performance
* Pipelined operation
* Multiple bus masters
* Burst transfers
* Split transactions

AMBA ASB

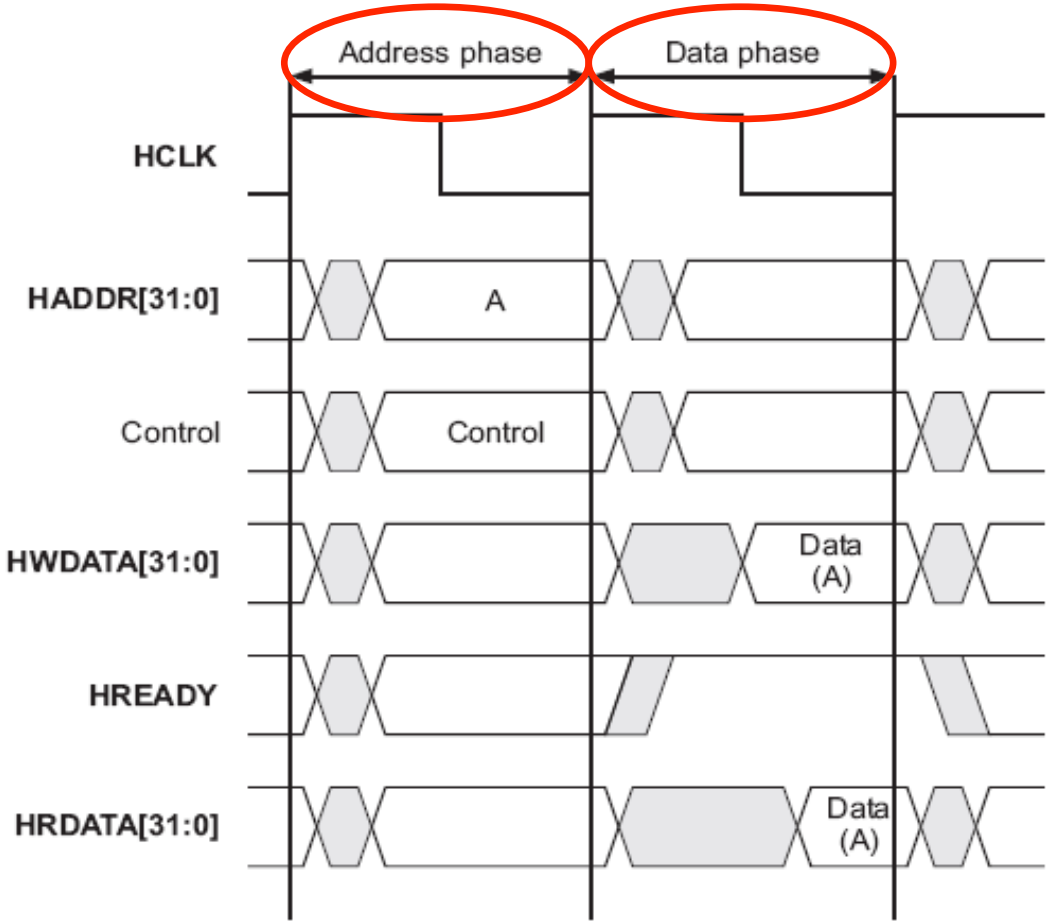
* High performance
* Pipelined operation
* Multiple bus masters

AMBA APB

* Low power
* Latched address and control
* Simple interface
* Suitable for many peripherals

Source: AMBA Specification, Rev. 2.0

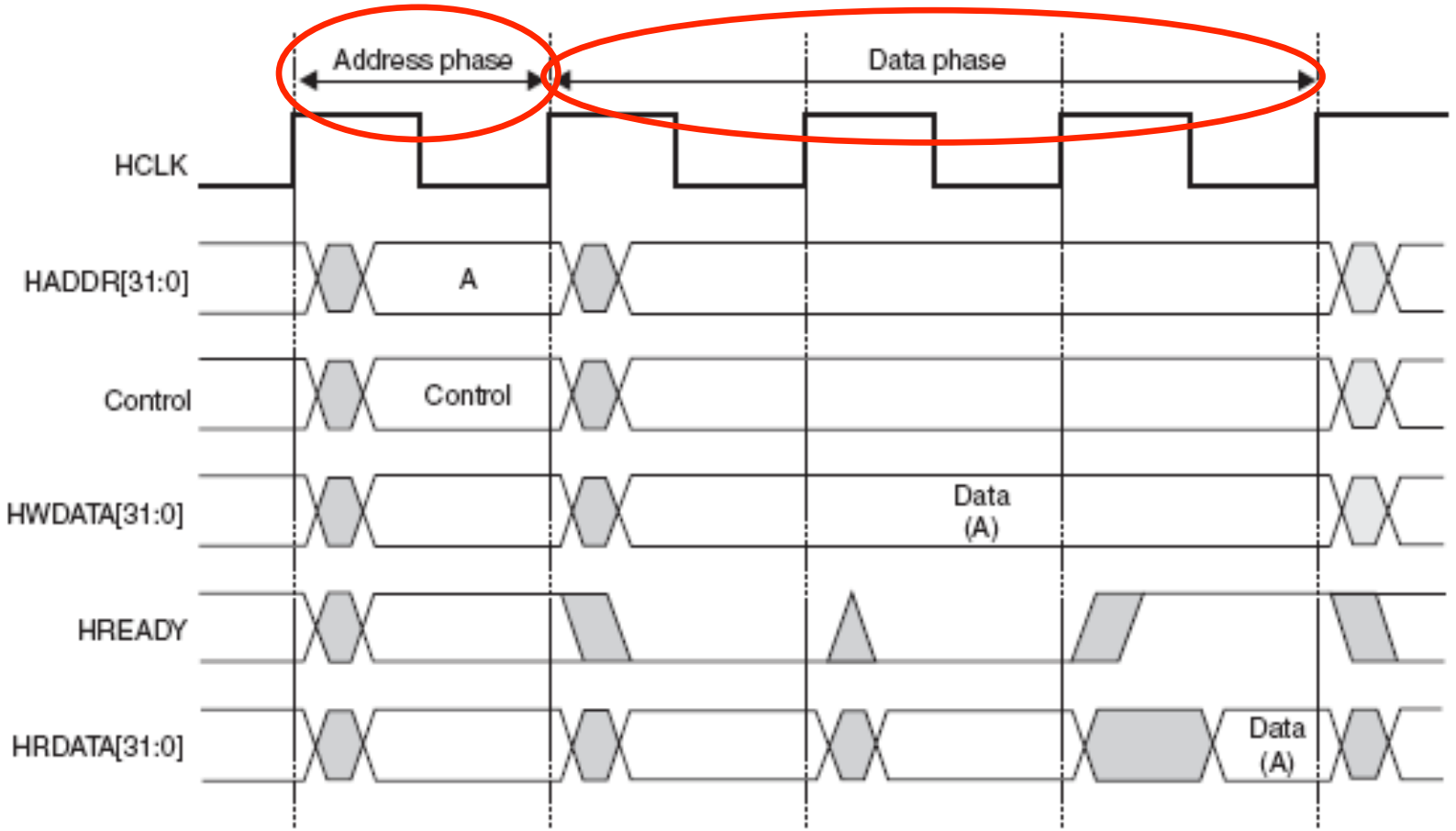
AHB Basic Transfer



- Split ownership of Address and Data bus

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

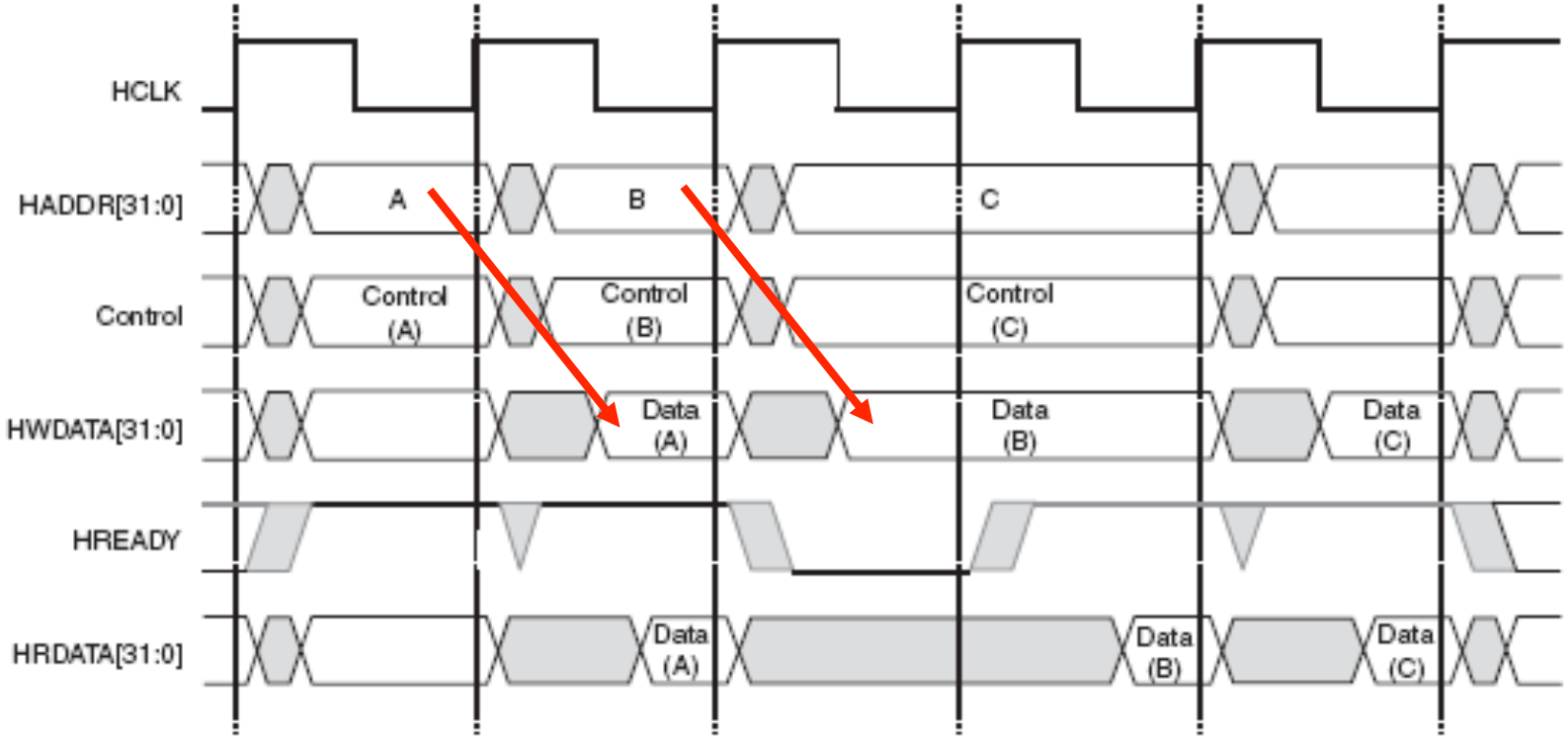
AHB Basic Transfer



- Data transfer with slave wait states

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

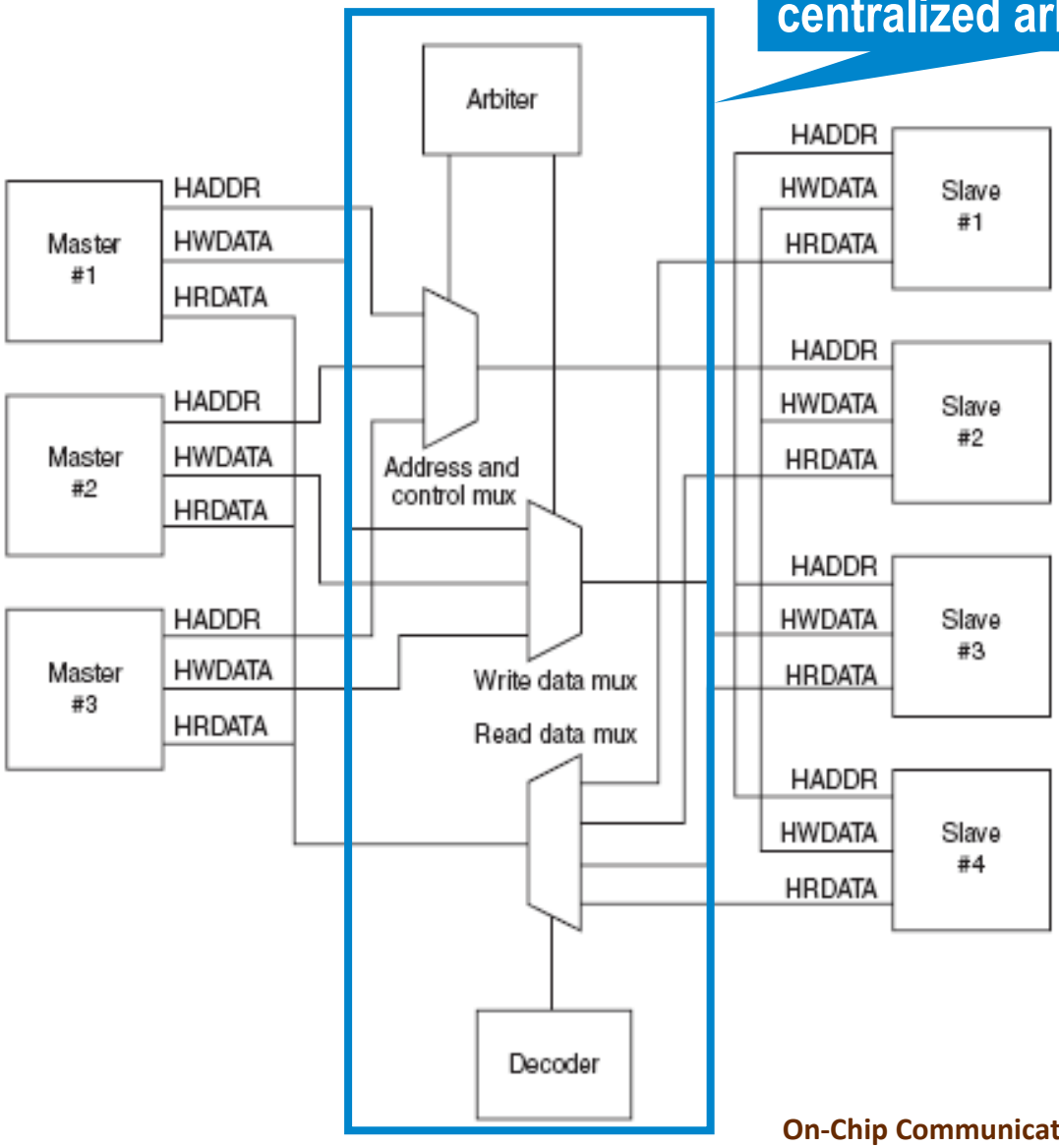
AHB Pipelining



- Transaction pipelining increases bus bandwidth

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

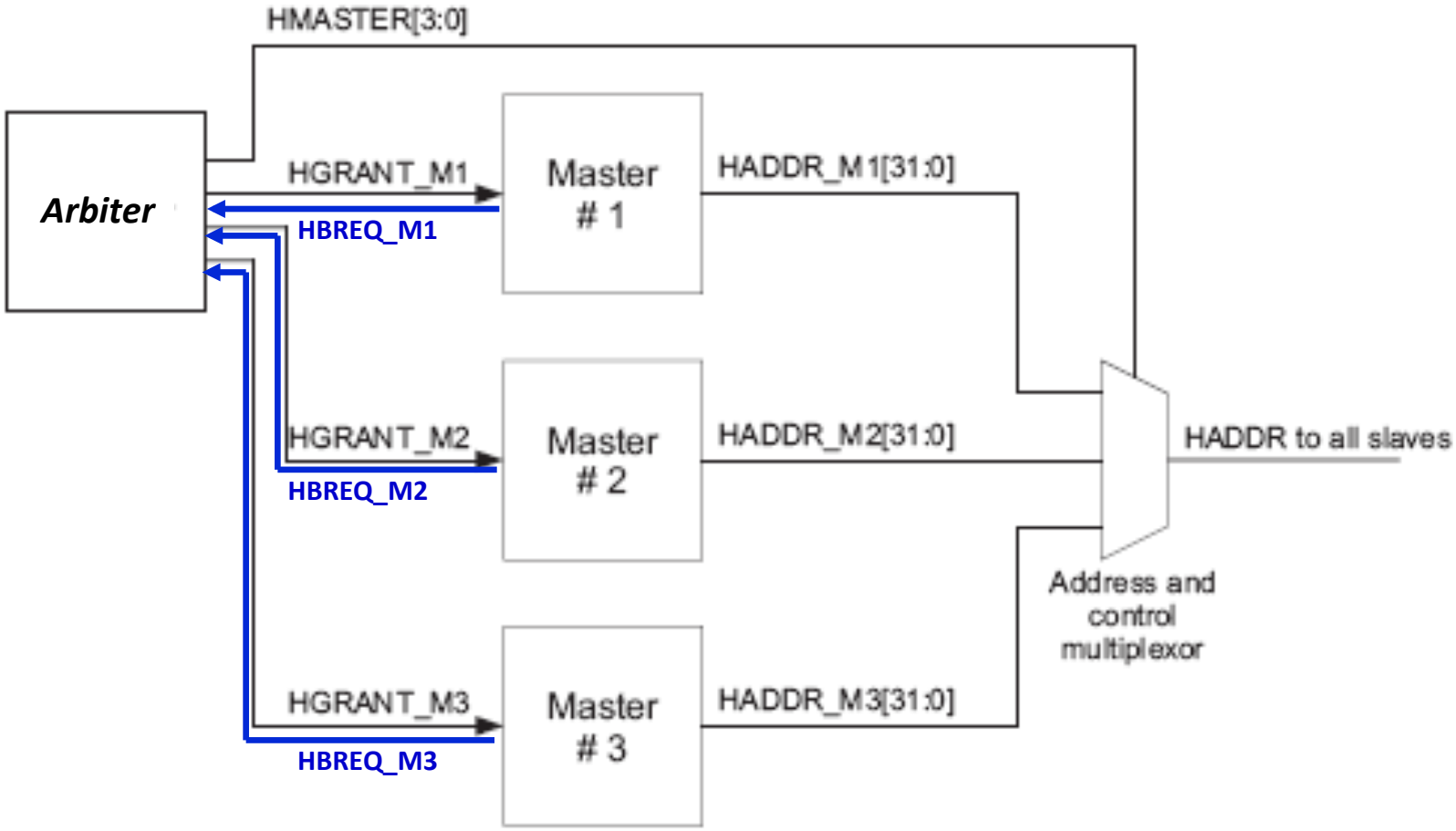
AHB Architecture



- 1 unidirectional address bus (HADDR)
- 2 unidirectional data buses (HWDATA, HRDATA)
- At any time only 1 active data bus

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

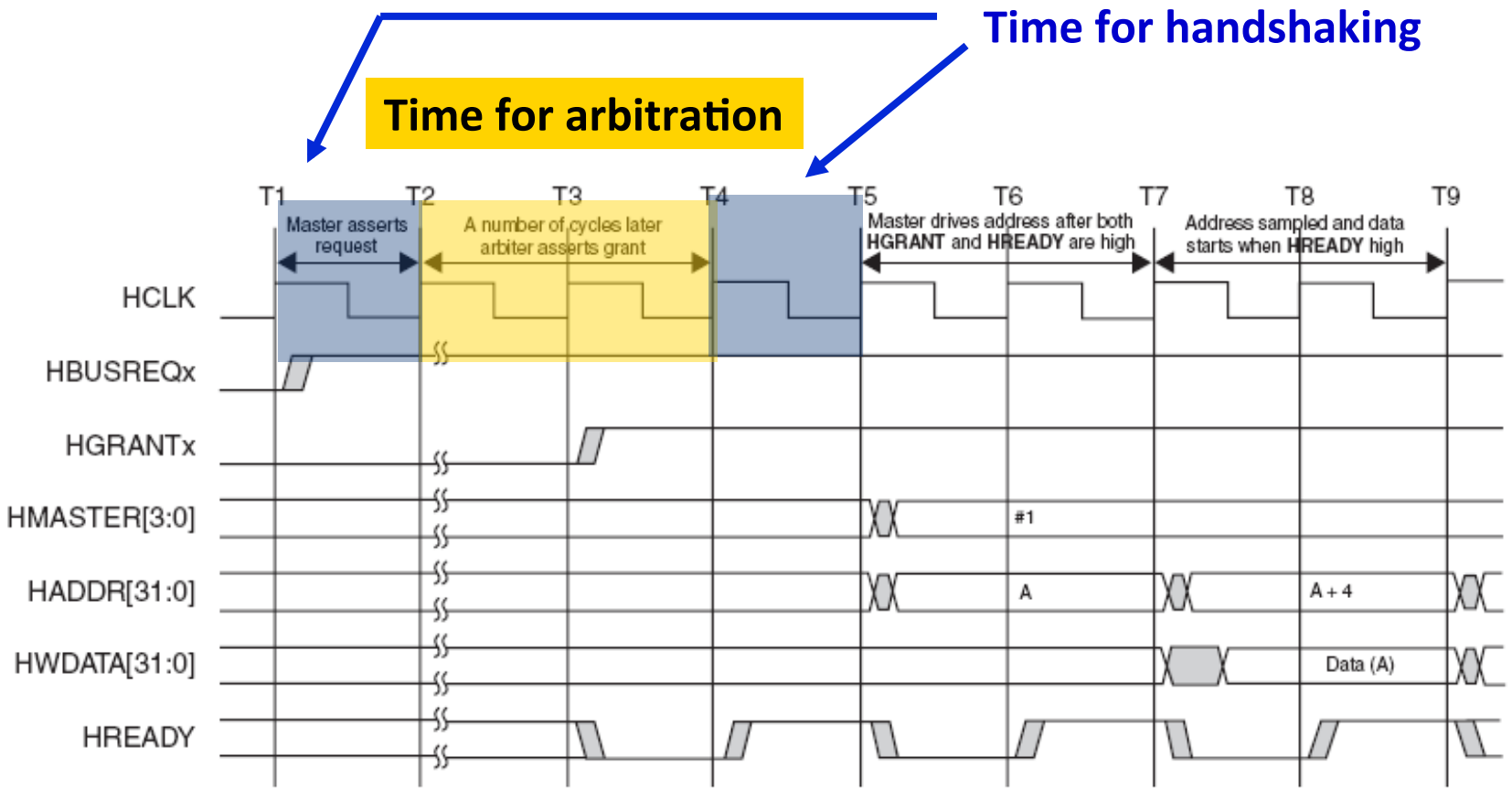
AHB Arbitration



- Arbitration protocol is specified, but not the arbitration policy

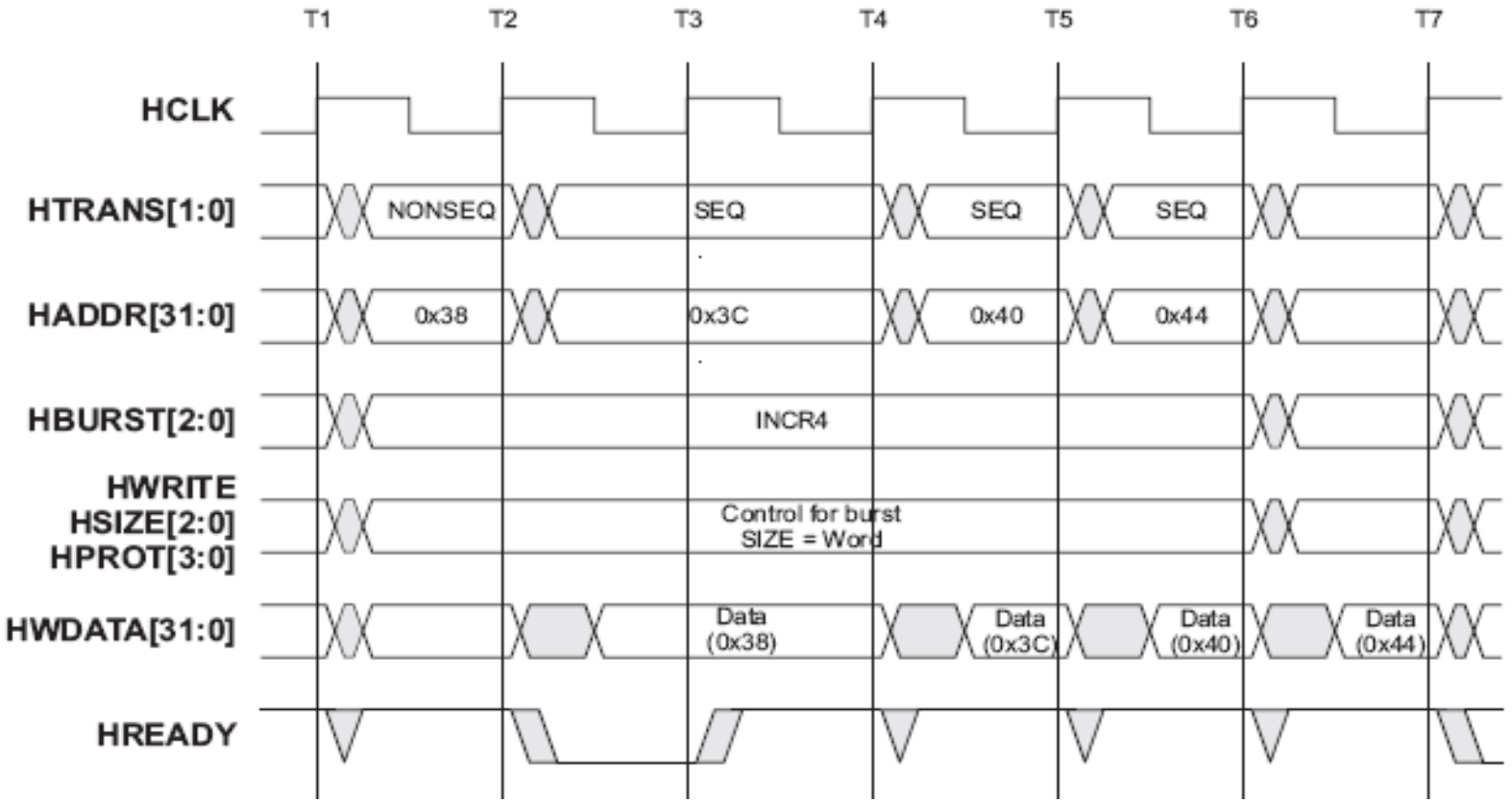
On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

Cost of Arbitration in AHB



On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

AHB Pipelined Burst Transfers



- **Bursts cut down on arbitration, handshaking time, improving performance**

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

AHB Burst Types

Fixed length bursts

HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

- Incremental bursts access sequential locations
  - e.g. 0x64, 0x68, 0x6C, 0x70 for INCR4, transferring 4 byte data
- Wrapping bursts "wrap around" address if starting address is not aligned to total no. of bytes in transfer
  - e.g. 0x64, 0x68, 0x6C, 0x60 for WRAP4, transferring 4 byte data

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

AHB Control Signals

- Transfer direction
  - HWRITE – write transfer when high, read transfer when low
- Transfer size
  - HSIZE[2:0] indicates the size of the transfer

HSIZE[2]	HSIZE[1]	HSIZE[0]	Size	Description
0	0	0	8 bits	Byte
0	0	1	16 bits	Halfword
0	1	0	32 bits	Word
0	1	1	64 bits	-
1	0	0	128 bits	4-word line
1	0	1	256 bits	8-word line
1	1	0	512 bits	-
1	1	1	1024 bits	-

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

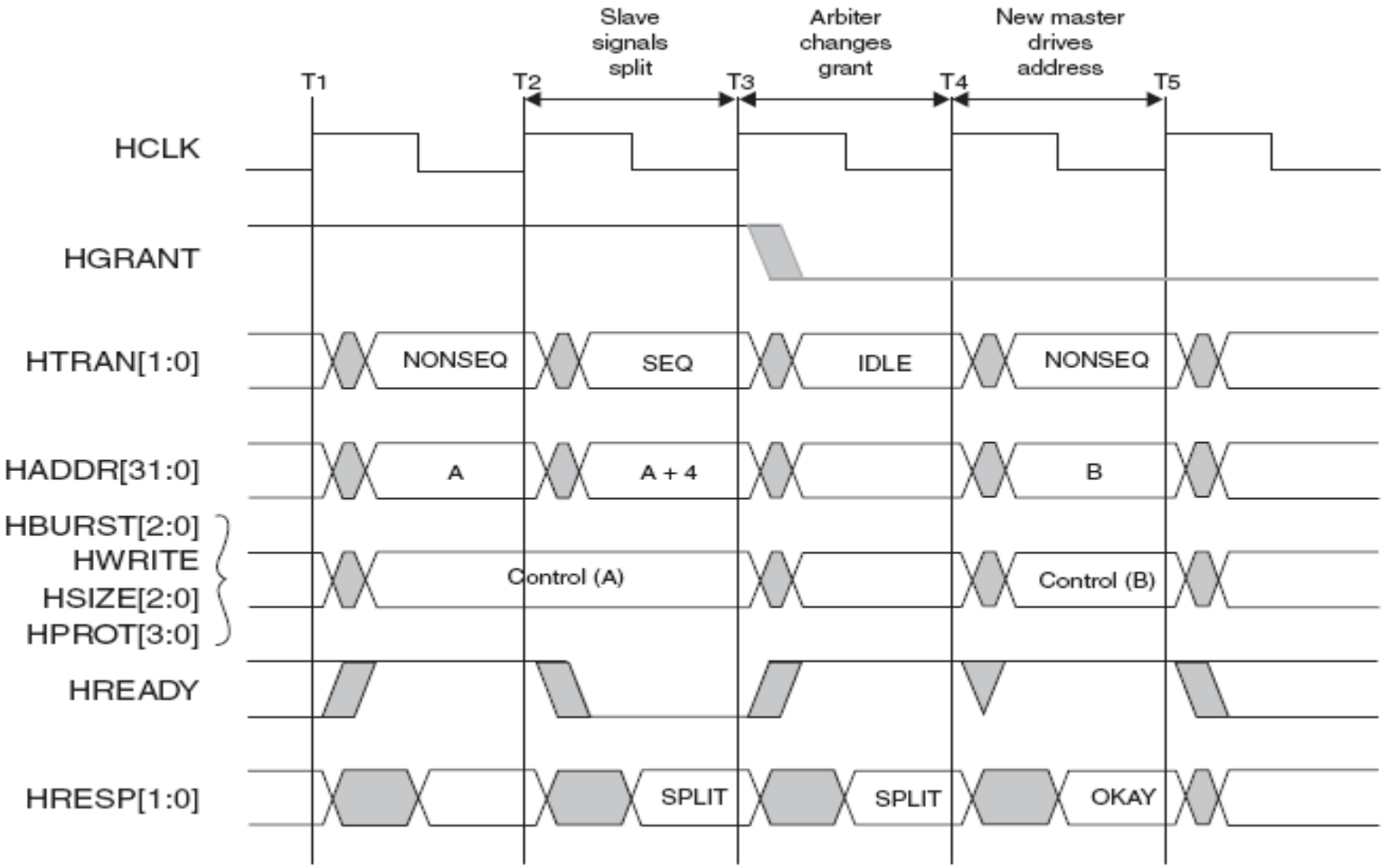
AHB Control Signals

- **Protection control**
  - **HPROT[3:0], provide additional information about a bus access**

HPROT[3] cacheable	HPROT[2] bufferable	HPROT[1] privileged	HPROT[0] data/opcode	Description
-	-	-	0	Opcode fetch
-	-	-	1	Data access
-	-	0	-	User access
-	-	1	-	Privileged access
-	0	-	-	Not bufferable
-	1	-	-	Bufferable
0	-	-	-	Not cacheable
1	-	-	-	Cacheable

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

AHB Split Transfers

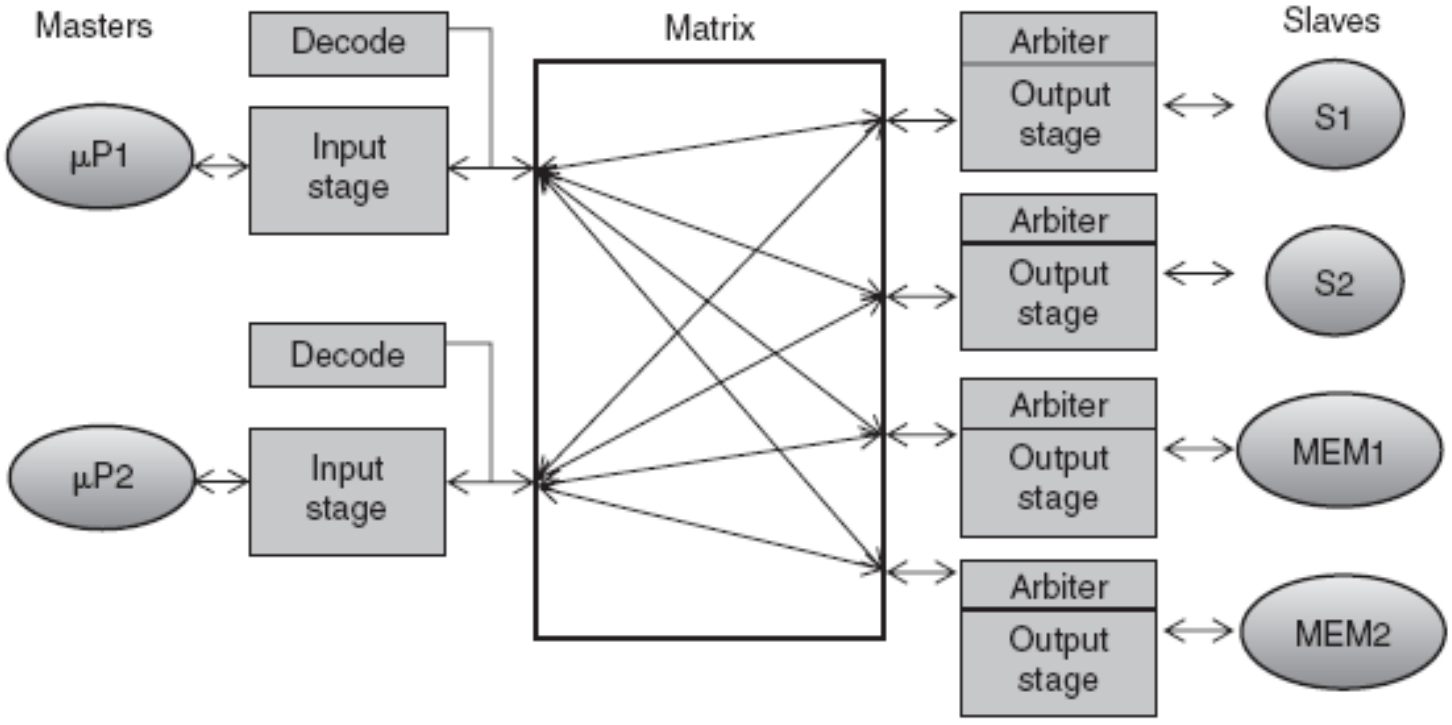


- Improves bus utilization
- May cause deadlocks if not carefully implemented

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

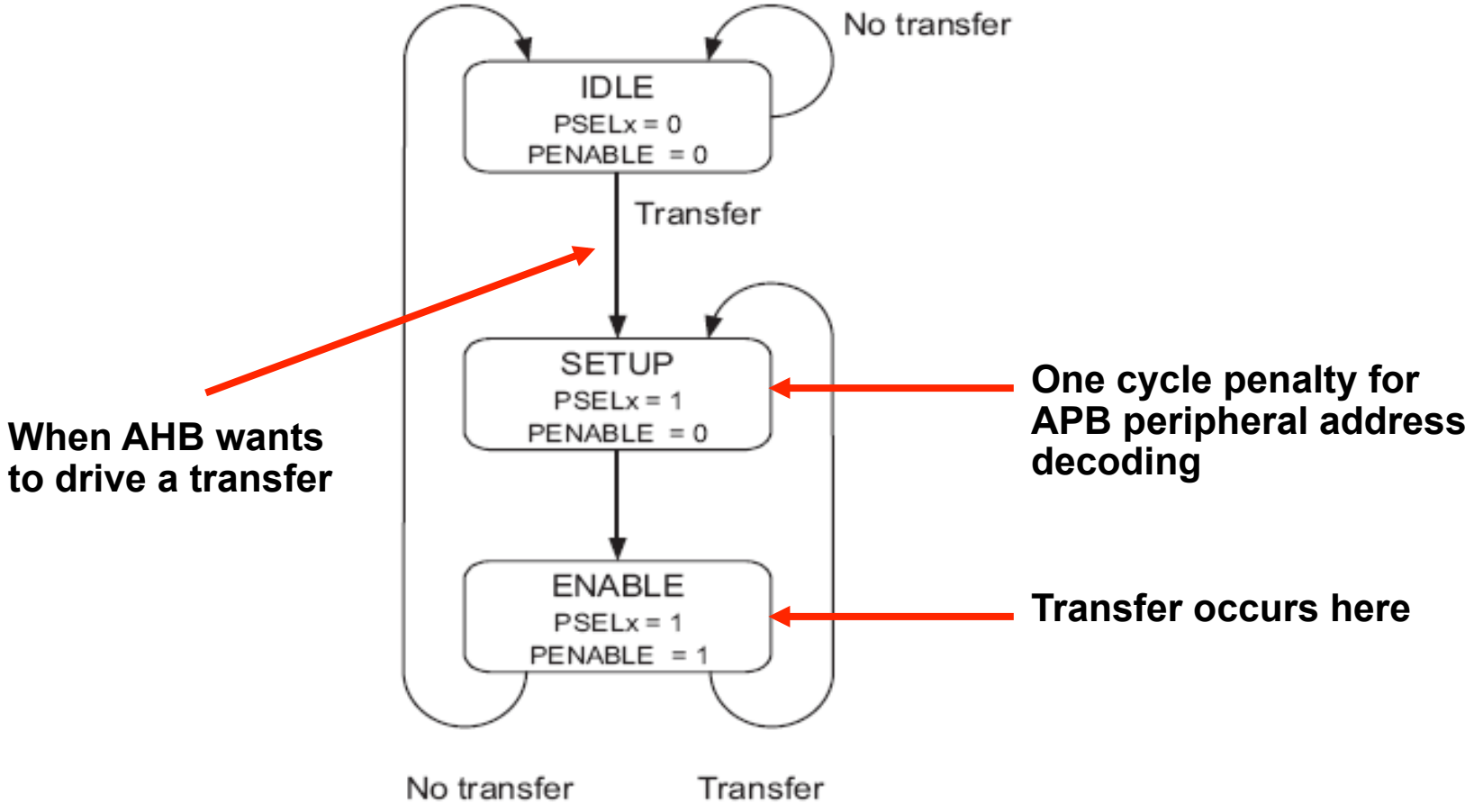
AHB Bus Matrix Topology

- In addition to shared bus and hierarchical bus, AHB can be implemented as a bus matrix



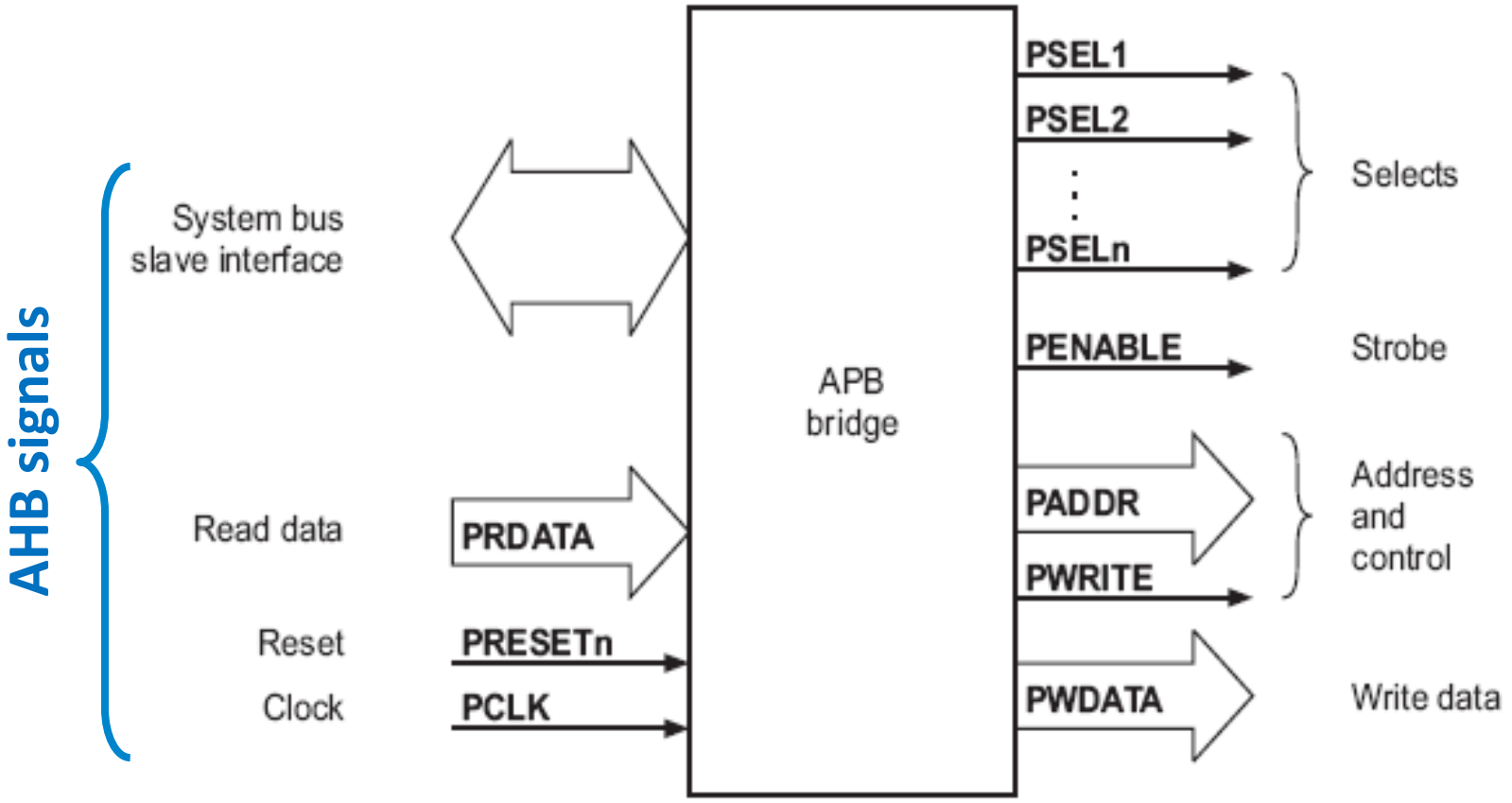
On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

APB State Diagram



- no (multi-cycle) bursts, pipelined transfers

AHB-APB Bridge



High performance **Low power (and performance)**

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

AMBA 3.0

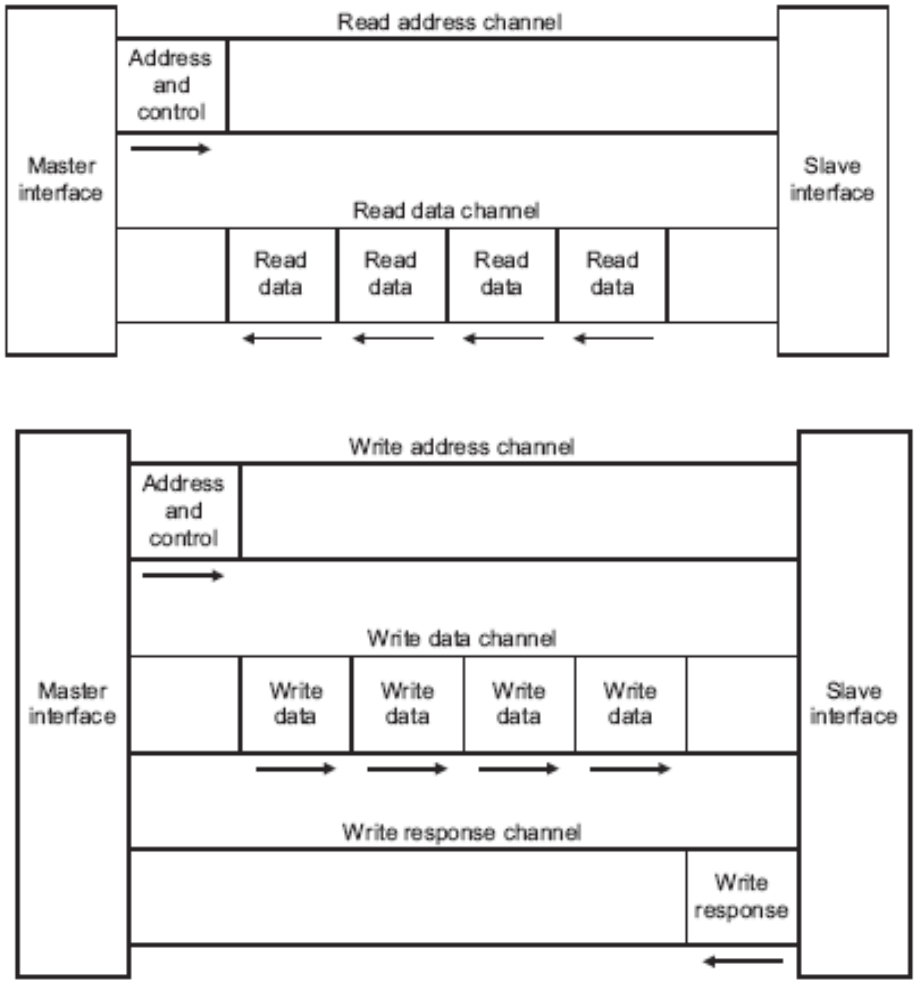
- **AXI high performance protocol**
  - Support for separate read address, write address, read data, write data, write response channels
  - Up to 16 masters allowed
  - Requires ~77 control signals
  - Out of order (OO) transaction completion
  - Fixed mode burst support
    - Useful for I/O peripherals
  - Advanced system cache support
    - Specify if transaction is cacheable/bufferable
    - Specify attributes such as write-back/write-through
  - Enhanced protection support
    - Secure/non-secure transaction specification
  - Exclusive access (for semaphore operations)
  - Register slice support for high frequency operation

Source: AMBA AXI Protocol Specification

On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

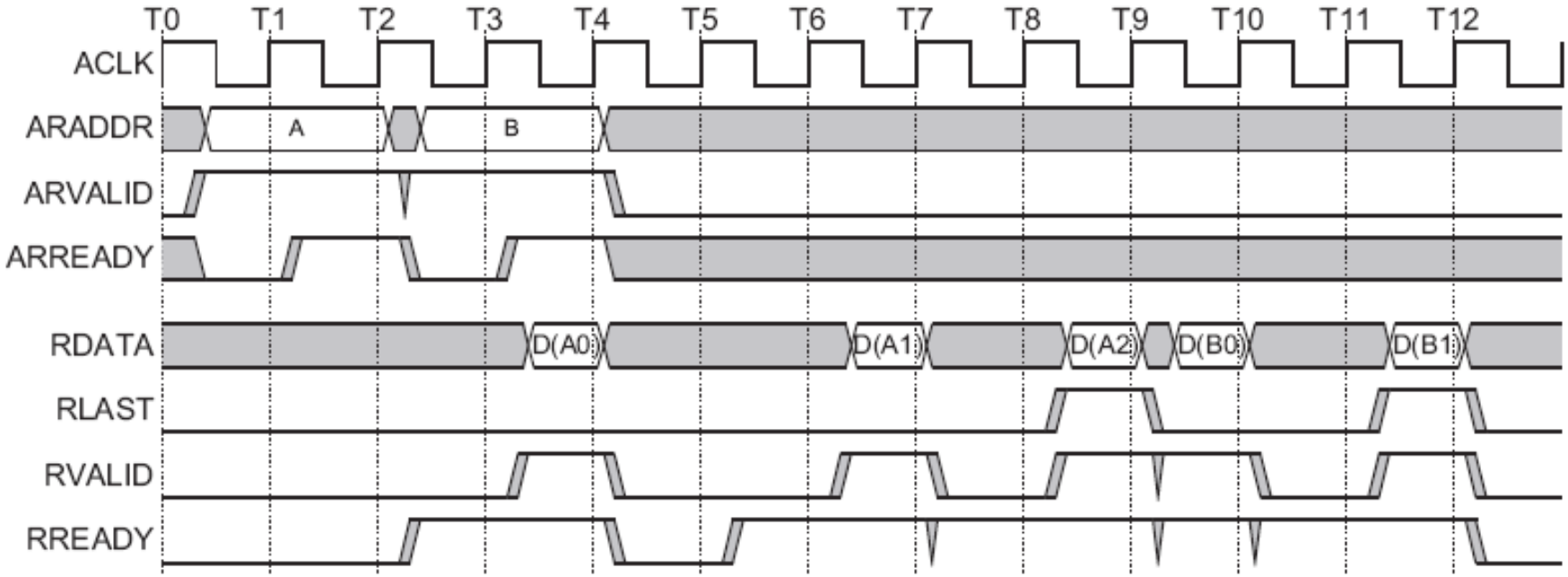
Multi-Channel Support

- Address, Data, and Response split between channels, rather than phases
- Allows simultaneous reads and writes



Source: AMBA AXI Protocol Specification

AXI Read Transactions

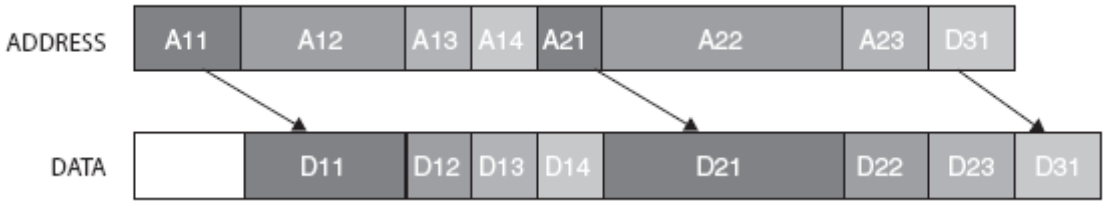


- Up to 16 transactions can be queued at once

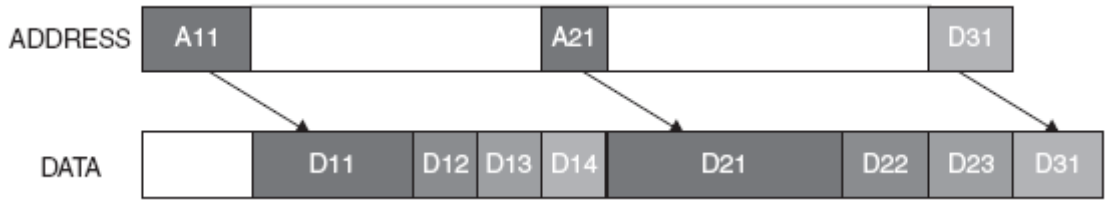
Source: AMBA AXI Protocol Specification

AHB vs. AXI Burst

- **AHB Burst**
  - Address and Data are locked together (single pipeline stage)
  - HREADY controls intervals of address and data

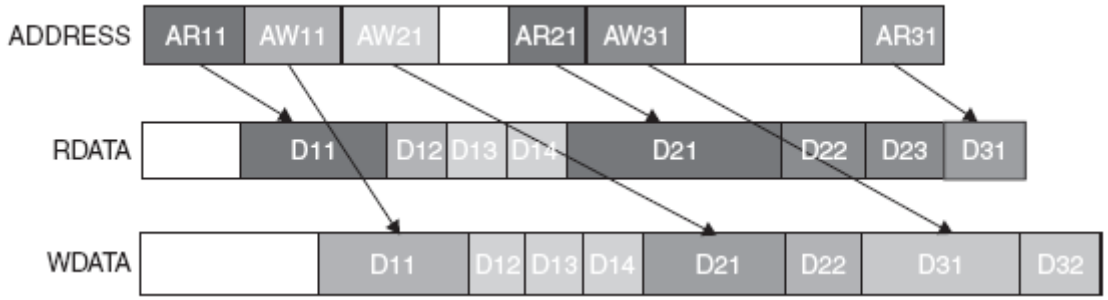


- **AXI Burst**
  - One Address for entire burst



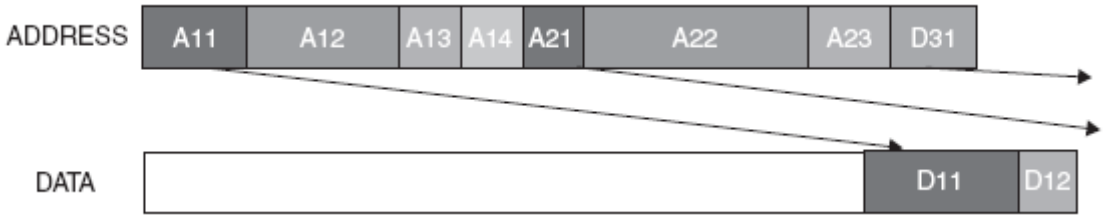
AHB vs. AXI Burst

- **AXI Burst**
  - Simultaneous read, write transactions
  - Better bus utilization

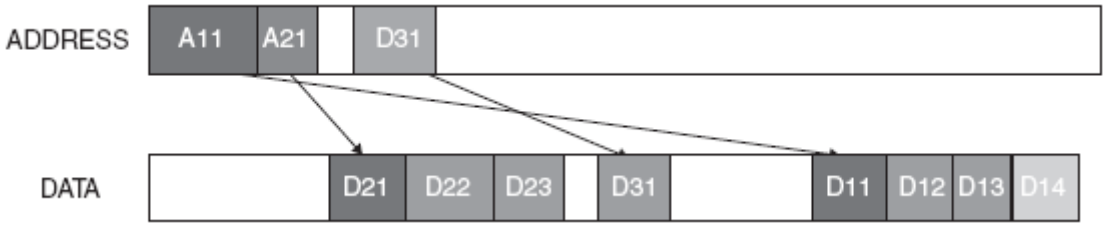


AXI Out of Order Completion

- **With AHB**
  - If one slave is very slow, all data is held up
  - SPLIT transactions provide very limited improvement



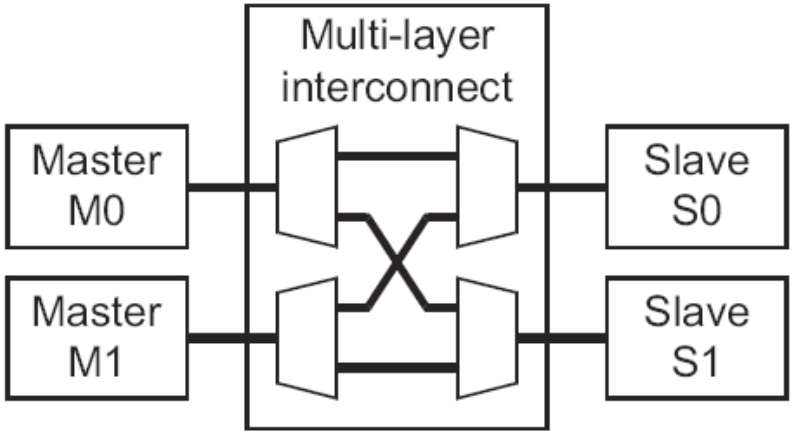
- **With AXI Burst**
  - Multiple outstanding addresses, out of order (OO) completion allowed
  - Fast slaves may return data ahead of slow slaves



On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

Multi-Layer Connectivity

- PL300 Interconnect is implemented as a crossbar:
- Multiple masters can talk to multiple slaves simultaneously



Source: PL300 Technical Reference Manual

Comparison of AMBA Bus Types

	APB	AHB	AXI / PL300
Processors	all	ARM7,9,10	ARM11/Cortex
Control Signals	4	27	77
No. of Masters	1	1-15	1-16
No. of Slaves	1-15	1-15	1-16
Interconnect Type	Central MUX?	Central MUX	Crossbar w/ 5 channels
Phases	Setup, Enable	Bus request, Address, Data	Address, Data, Response
Xact. Depth	1	2	16
Burst Lengths	1	1-32	1-16
Simultaneous Read & Write	no	no	yes

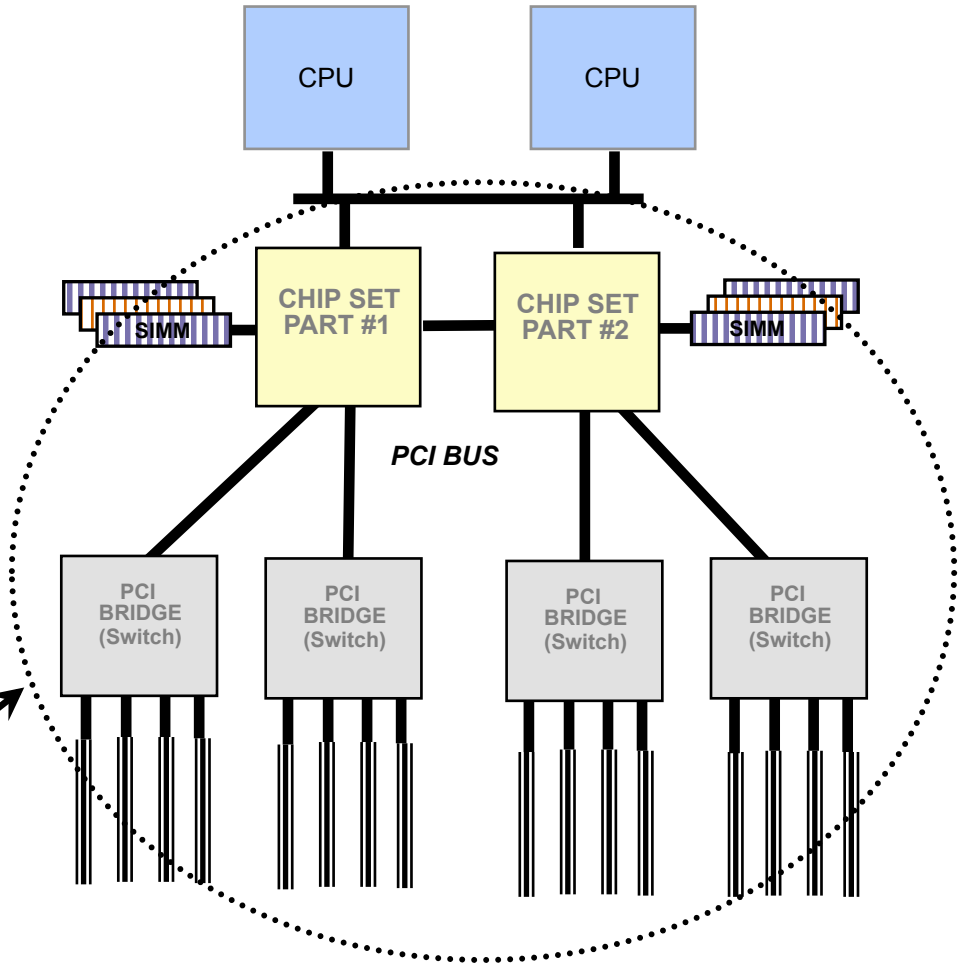
On-Chip Communication Architectures: Sudeep Pasricha & Nikil Dutt

Parallel bus protocols

- AMBA
- PCI-X

The PCI Hierarchy

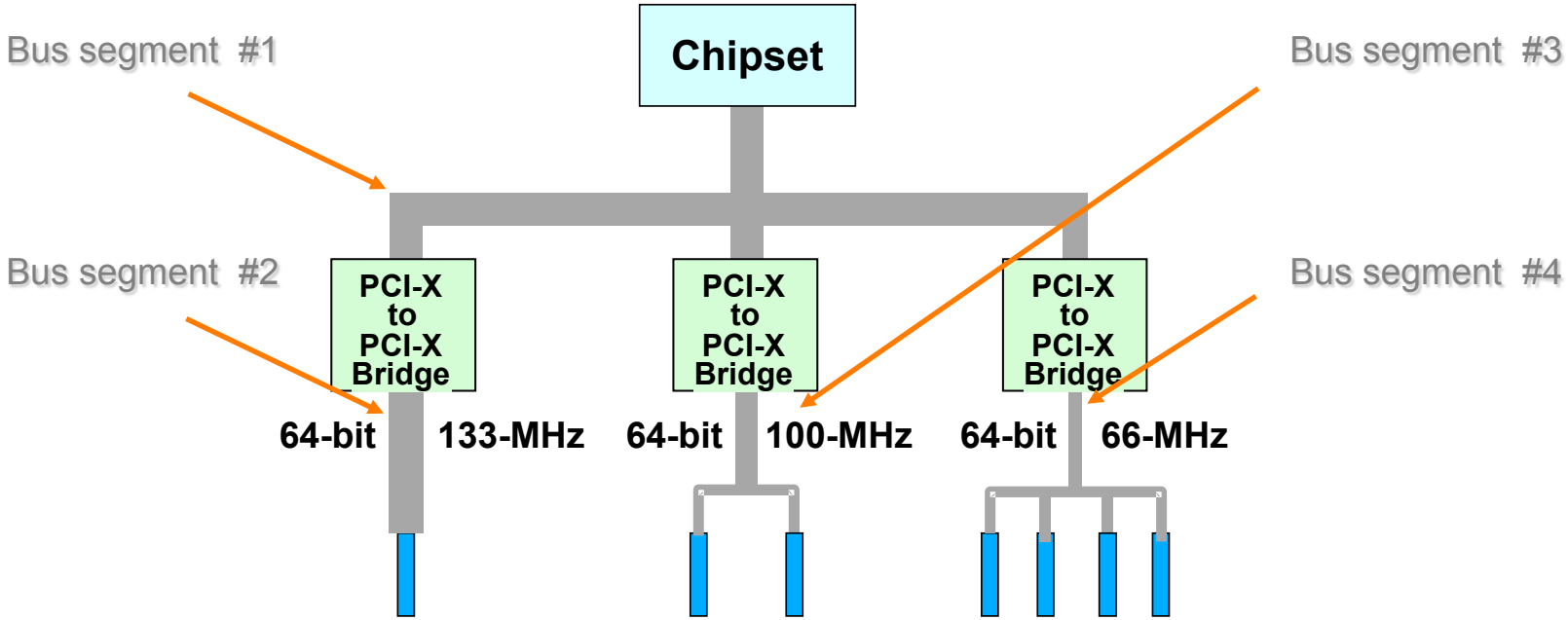
- The PCI hierarchy can scale to a maximum 256 bus segments
- Each bus segment can have a maximum of 32 PCI devices.
- Each PCI device can have a maximum of 8 unique functions (SCSI,NIC,etc) within the device



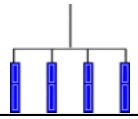
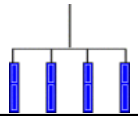
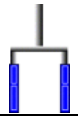
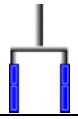
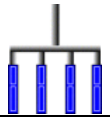
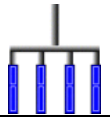
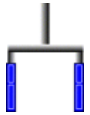
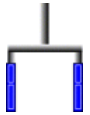
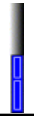
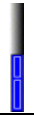
The PCI Hierarchy

PCI-X Scalability

PCI-X bridges allow PCI-X to link as many as 256 separate bus segments.



PCI Modes and Speeds

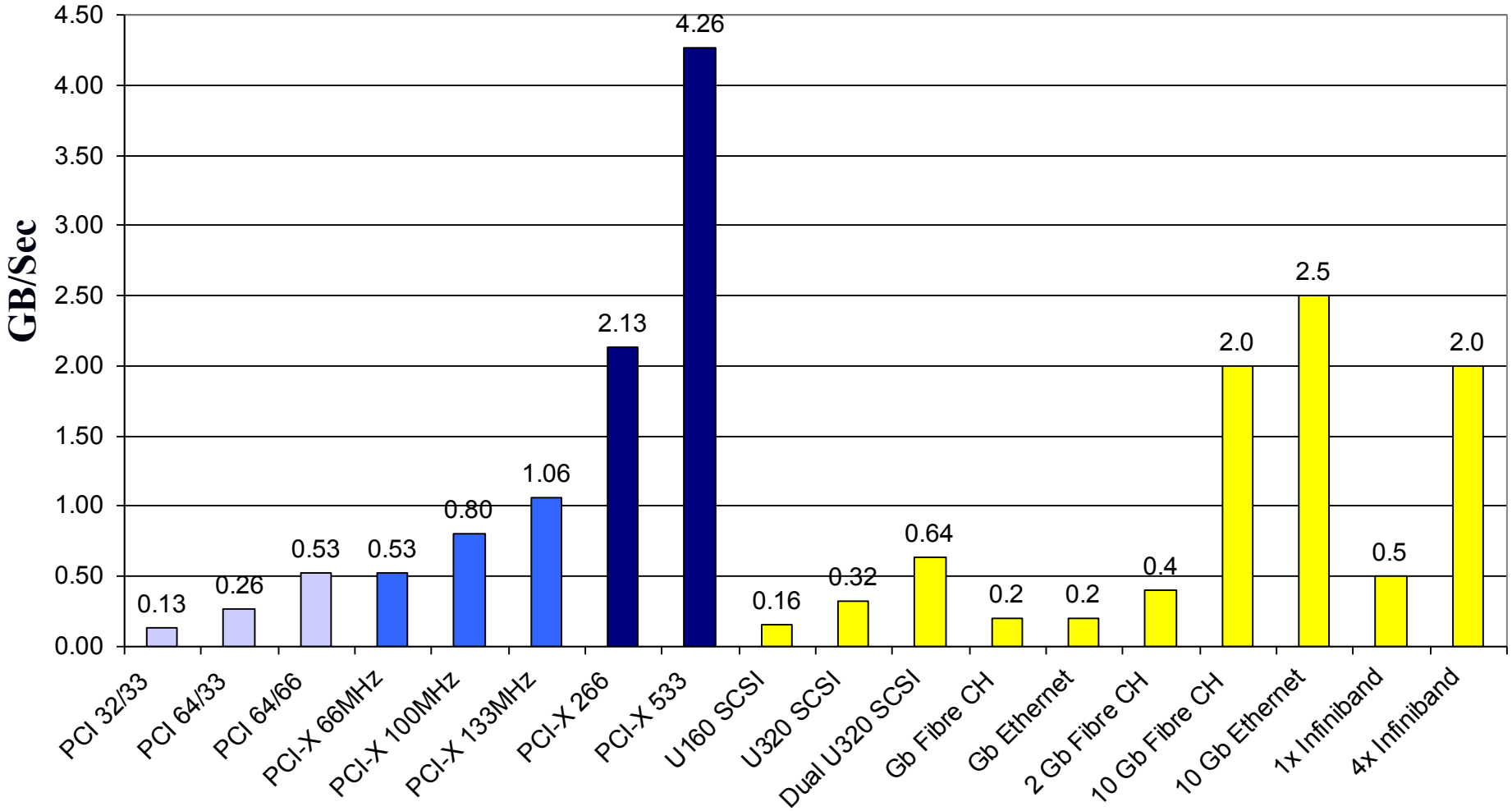
		64-Bit		32-Bit		16-Bit
Mode	**$V_{I/O}$**	**Slots**	**MB**	**Slots**	**MB**	
PCI 33	5V/3.3V		266		133	N/A
PCI 66*	3.3V		533		266	N/A
PCI-X 66	3.3V		533		266	N/A
PCI-X 133 (operating at 100 MHz)	3.3V		800		400	N/A
PCI-X 133	3.3V		1066		533	N/A
PCI-X 266	1.5V		2133		1066	533
PCI-X 533	1.5V		4266		2133	1066

All speed grades of PCI and PCI-X can be used to support point-to-point applications. The slower speed grades can be used to support multi-drop buses as well. Usage is dependent upon the needs of the application.

PCI-X Protocol Features

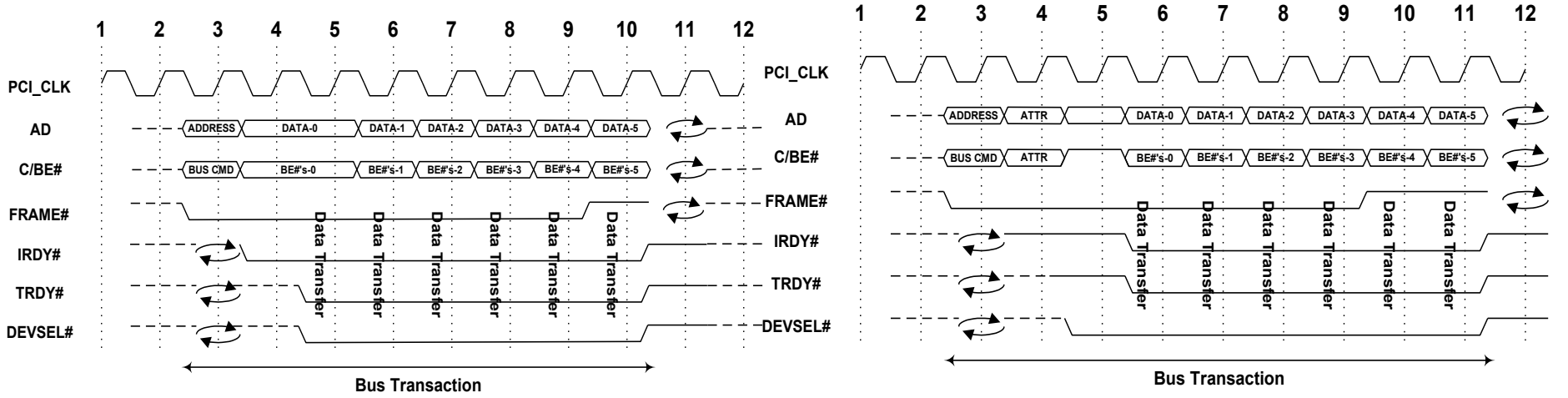
- Split Transactions like Intel's Pentium & Itanium FSB and Compaq's Alpha processors EV6 interconnect
- Byte Count on every transaction like USB
- Transaction ID's like IEEE 1394
- Enhanced recovery mechanism improves system availability
- Fixed I/O cache size similar to processor busses like Intel's Pentium III & Itanium, and Compaq's Alpha processors EV6 interconnect
- Transaction tagging to simplify multi-threaded operations
- Standard 64-bit addressing eases scalability
- Power management aware devices, as required for Window 2000
- Registered protocol like USB, IEEE 1394, Intel FSB, Compaq's Alpha EV6 interconnect

PCI-X 2.0 Bandwidth Span vs. Target Applications

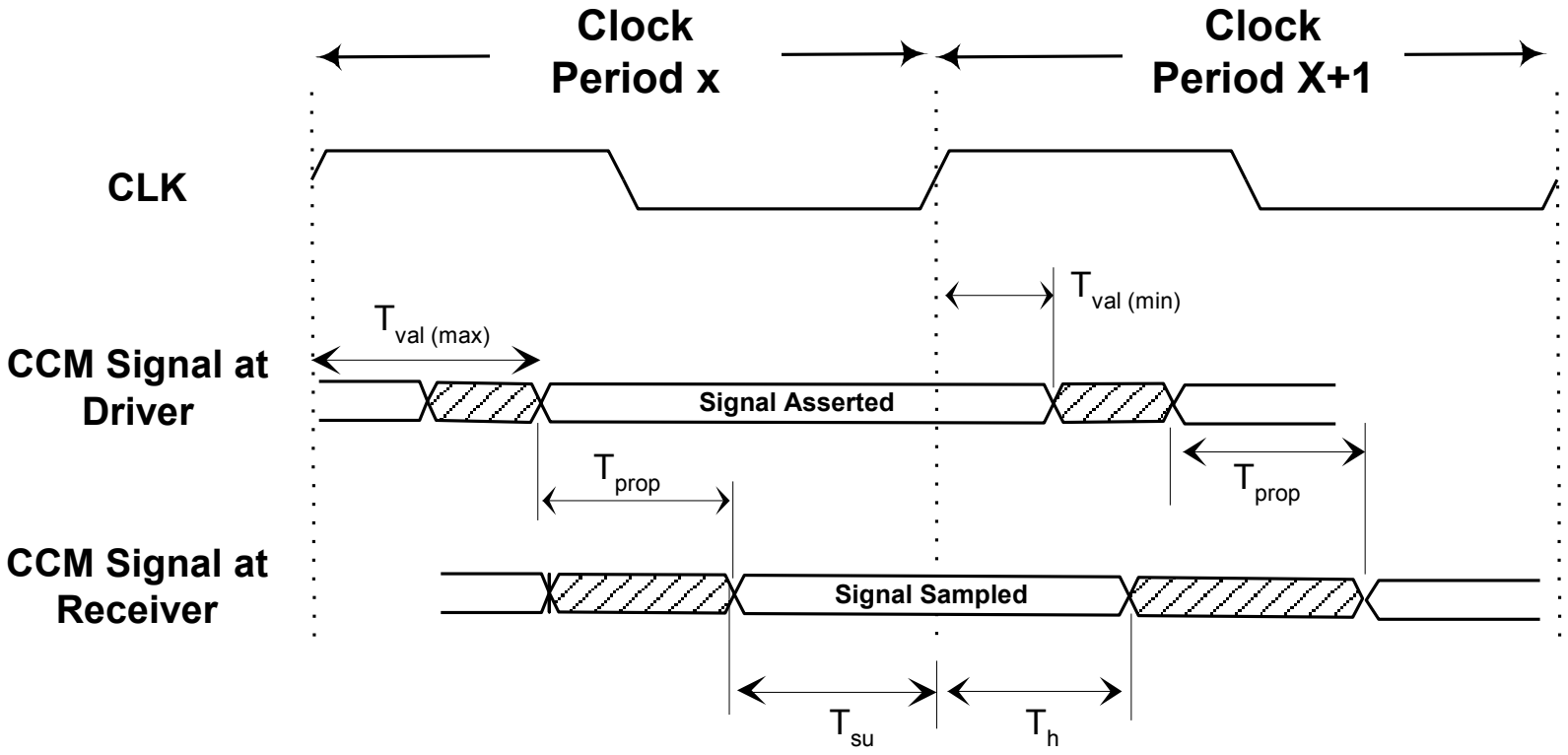


PCI 2.2 vs. PCI-X Mode 1

- Write transaction with the same device select timing and same wait states, using 6 data phases
- Conventional PCI bus requires 9 clocks, PCI-X bus requires 10 clocks



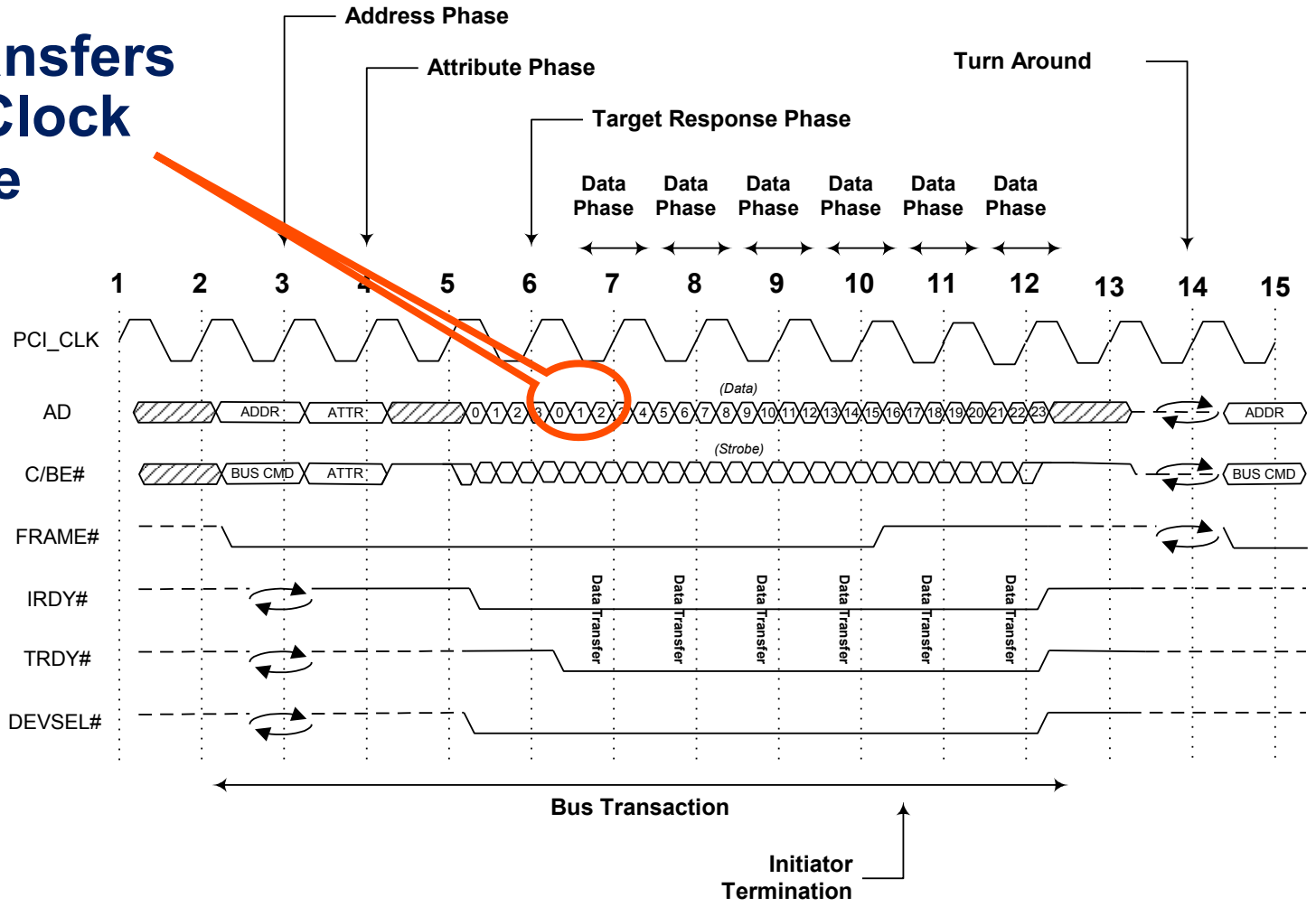
Common Clock Transfer Window



Note: Shared/common/synchronous clock protocols are specified relative to a single clock. As a result, the signal flight time, which remains unchanged regardless of the signaling levels, naturally limits the maximum frequency for such protocol.

PCI-X 533 Burst Write Timings

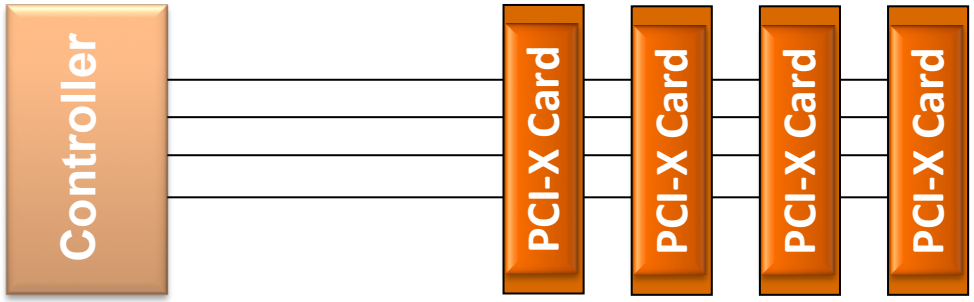
4 Transfers per Clock Cycle



PCI-X 533 achieves high data rates by transferring 4 QWORDS of data per clock cycle.

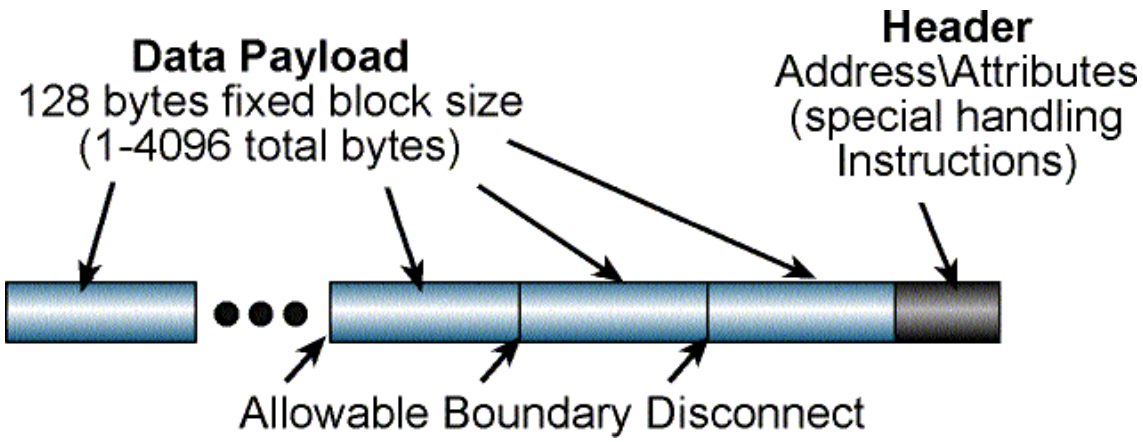
PCI-X Bus Configurations

- PCI-X bus technology supports up to 4 loads with just one controller.



- Multi-drop capabilities of PCI-X, provide maximum connectivity per controller pin (32 bit bus, 66MHz)
  - One load – 128 pins per connection.
  - Two loads – 64 pins per connection.
  - Four loads – 32 pins per connection.

PCI-X Packet Format



Payload –

- 1 byte to 4K bytes in length
- Payload is partitioned into logical packets, naturally aligned to 128-bytes (lower 7-bit of address all 0's)
- Transaction can be disconnected on any naturally aligned packet boundary

Header –

- Part of every Transaction
- Includes the address and the attribute field
- Address field carries routing information
- Attribute field carries special handling instructions

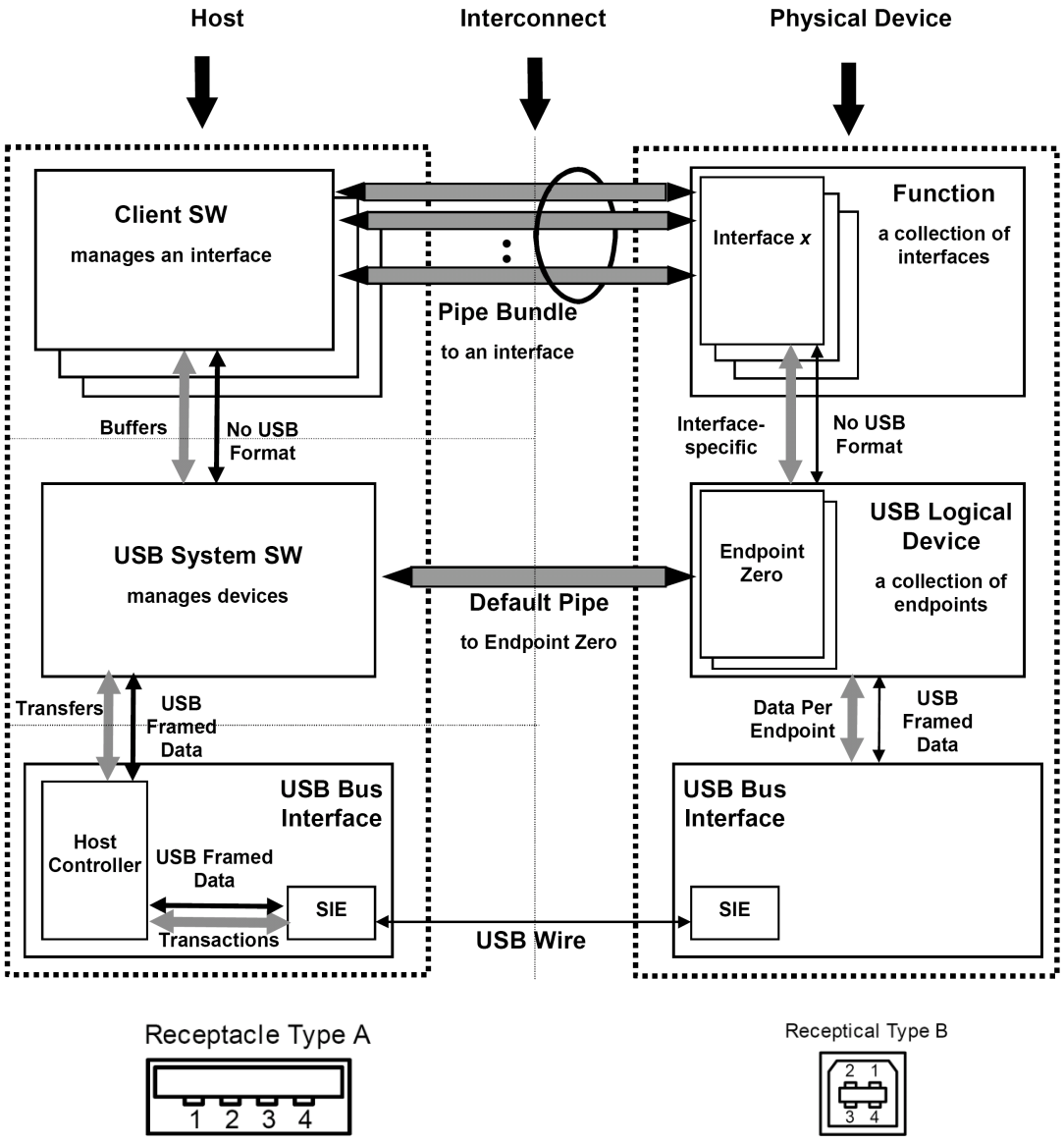
Serial bus protocols

- Universal Serial Bus (USB)
- CAN Bus
- RS232 Bus

USB Device Basics

- **Universal Serial Bus (USB) is a connection between a host computer and a number of peripheral devices**
  - Created to replace a wide range of slow buses (parallel, serial, and keyboard connections)
  - Can support up to 480MB/s in theory
  - A tree built out of several point-to-point links
    - Links are four-wire cables (ground, power, and two signal wires)
  - A USB device can never start sending data without first being asked by the host controller
  - Single-master implementation
    - Host polls various devices
    - A device can request a fixed bandwidth (for audio and video I/O)

System Block Diagram



www.USB.org

USB Device Basics

- USB protocol defines a set of standards that any device can follow
  - If a device falls into one of the predefined classes and follows that standard, no need to write a special device driver
  - Predefined classes: storage devices, keyboards, mice, joysticks, network devices, and modems
  - No defined standard for video devices and USB-to-serial devices
    - A driver is needed for every device
- Linux supports two types of USB drivers
  - Drivers on a host system
    - Control the USB devices that are plugged into it
  - Drivers on a device (USB gadget drivers)
    - Control how that single device looks to the host computer as a USB device

Endpoints

- The most basic form of USB communication is through an endpoint
  - Carries data in one direction
    - From the host to device (OUT endpoint)
    - From the device to the host (IN endpoint)
- Four basic types of endpoints:
  - Control
  - Bulk
  - Interrupt
  - Isochronous
- Control and bulk endpoints are used for asynchronous data transfers
- Interrupt and isochronous endpoints are periodic with reserved bandwidth

Four endpoint types

- CONTROL
  - Used for configuring the device, retrieving information and status about the device, or sending commands to the device
  - Every device has a control endpoint called endpoint 0
    - Used by USB core to configure the device at insertion time
    - Transfers are guaranteed with reserved bandwidth
- INTERRUPT
  - Transfer small amounts of data at a fixed rate
  - For USB keyboards and mice
  - Also used to control the device
  - Not for large transfers
  - Guaranteed reserved bandwidth
- BULK
  - Transfer large amounts of data
  - No data loss
  - Not time guaranteed
  - A BULK packet might be split up across multiple transfers
  - Used for printers, storage, and network devices
- ISOCHRONOUS
  - Transfer large amount of data
  - No guarantees (potential data loss)
  - For real-time data collections, audio and video devices

Interfaces

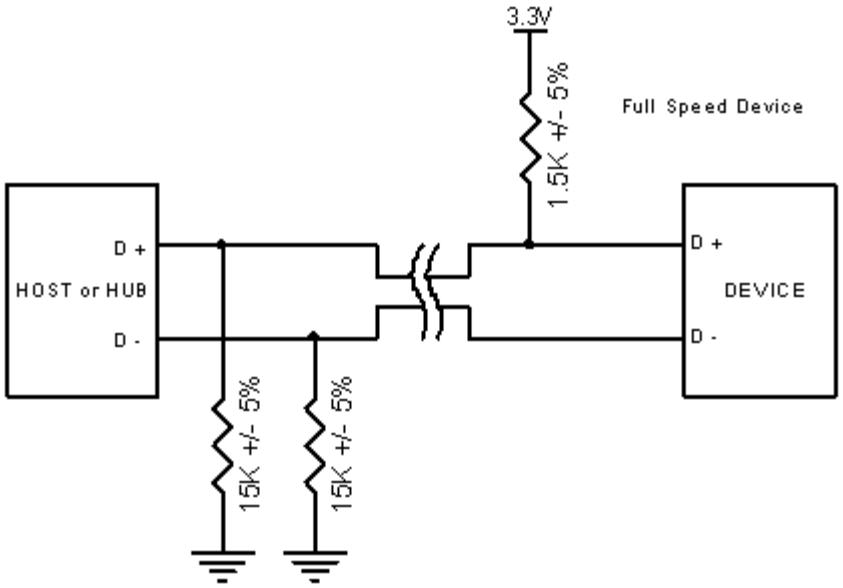
- USB endpoints are bundled into interfaces
  - A interface handles only one type of logical connection (e.g., a mouse)
  - Some devices have multiple interfaces
    - E.g., a speaker
      - One interface for buttons and one for audio stream
- USB interface may have alternate settings
  - E.g., different settings to reserve different amounts of bandwidth for the device

USB Signaling Levels

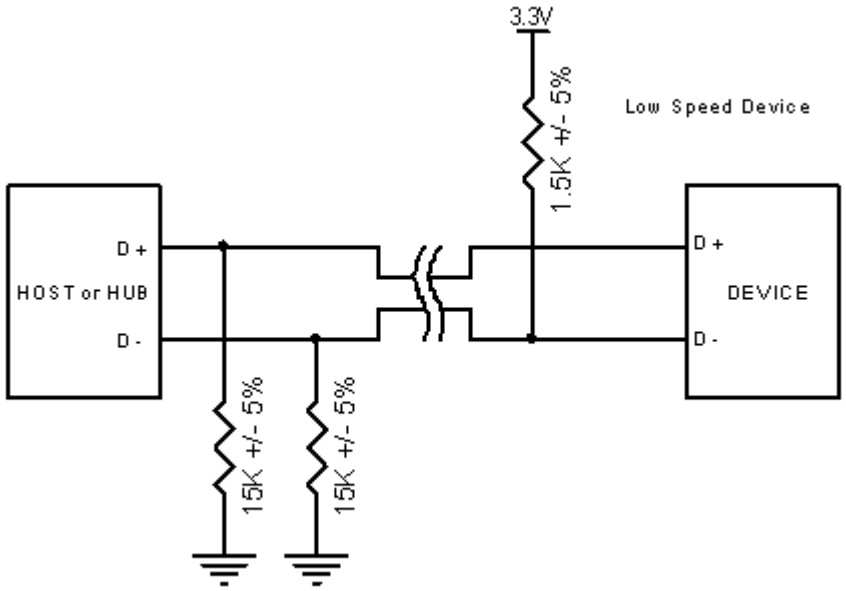
Pin Number	Cable Color	Function
1	Red	V_{BUS} (5 volts)
2	White	D-
3	Green	D+
4	Black	Ground

Bus State	Signaling Levels		
	At originating source connector (at end of bit time)	At final target connector	
		Required	Acceptable
Differential "1"	D+ > VOH (min) and D- < VOL (max)	(D+) - (D-) ≥ 200mV and D+ ≥ VIH (min)	(D+) - (D-) ≥ 200mV
Differential "0"	D- > VOH (min) and D+ < VOL (max)	(D-) - (D+) > 200mV and D- ≥ VIH (min)	(D-) - (D+) > 200mV
Single-ended 0 (SE0)	D+ and D- < VOL (max)	D+ and D- < VIL (max)	D+ and D- < VIH (min)
Data J state: Low-speed Full-speed	 Differential "0" Differential "1"	 Differential "0" Differential "1"	
Data K state: Low-speed Full-speed	 Differential "1" Differential "0"	 Differential "1" Differential "0"	
Idle state: Low-speed Full-speed	N.A.	 D- > VIHZ (min) and D+ < VIL (max) D+ > VIHZ (min) and D- < VIL (max)	 D- > VIHZ (min) and D+ < VIH (min) D+ > VIHZ (min) and D- < VIH (min)
Resume state	Data K state	Data K state	
Start-of-Packet (SOP)	Data lines switch from Idle to K state		
End-of-Packet (EOP)[4]	SE0 for approximately 2 bit times[1] followed by a J for 1 bit time[3]	SE0 for ≥ 1 bit time[2] followed by a J state for 1 bit time	SE0 for ≥ 1 bit time[2] followed by a J state
Disconnect (at downstream port)	N.A.	SE0 for ≥2.5μs	
Connect (at downstream port)	N.A.	Idle for ≥2ms	Idle for ≥2.5 μs
Reset	D+ and D- < VOL (max) for ≥10ms	D+ and D- < VIL (max) for ≥10ms	D+ and D- < VIL (max) for ≥2.5 μs

High Speed & Low Speed Detection

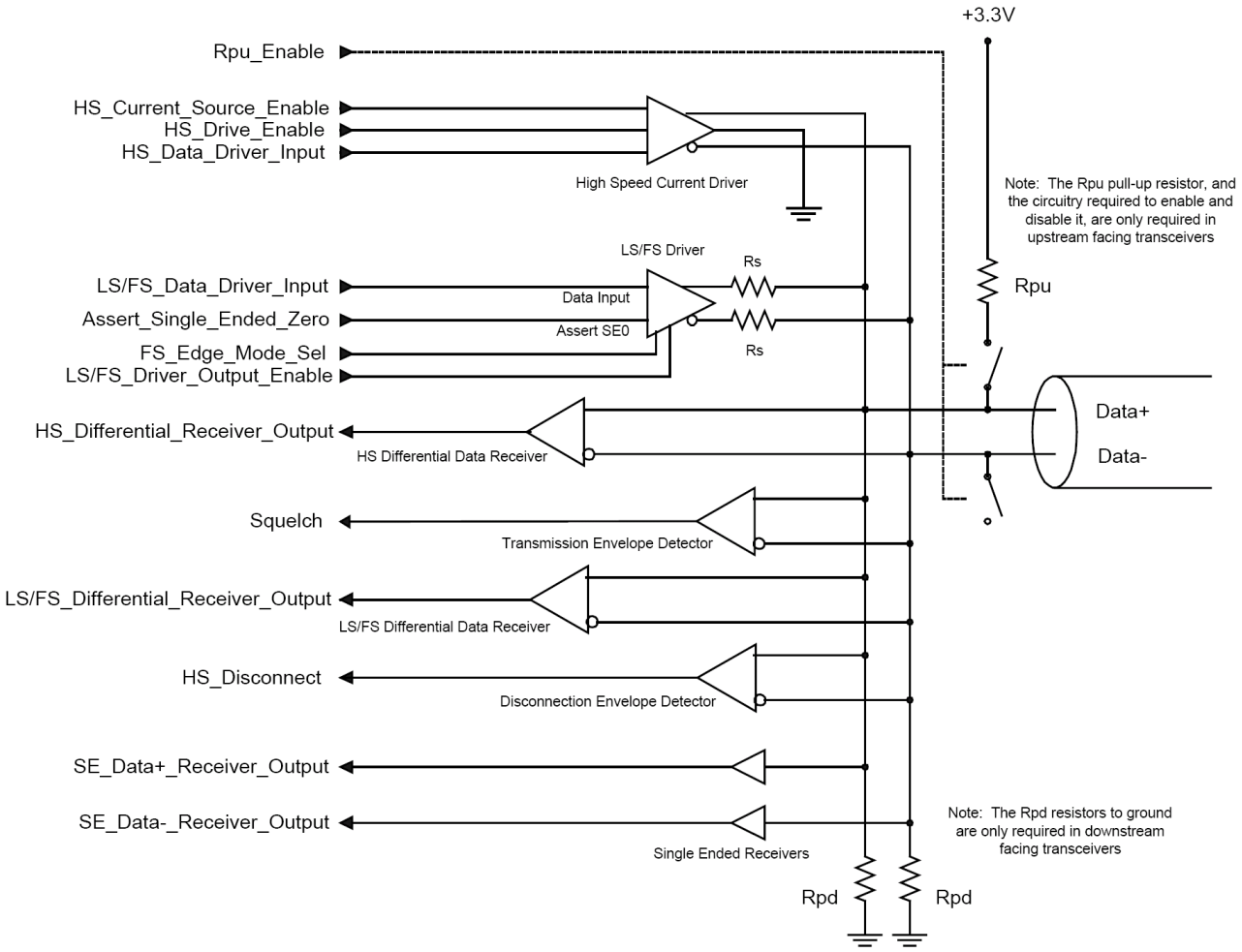


Full Speed Device with pull up resistor connected to D+



Low Speed Device with pull up resistor connected to D-

USB 2.0 Transceiver Circuit



Serial bus protocols

- Universal Serial Bus (USB)
- CAN Bus
- RS232

Controller Area Network (CAN) Bus

- Created in mid-1980s for automotive applications by Robert Bosch.
- Design goal was to make automobiles more reliable, safer, and more fuel efficient.
- Why buses in vehicles?
  - Number of electronic equipments is increasing heavily in vehicles (expensive wiring)
  - Modern vehicles control system needs a lot of information from sensors
  - Easier fault diagnostic (tester can be connected to a single plug)
  - Reduces amount of wire needed.

Vehicle Bus Classification

Network Classification	Speed	Application
Class A	< 10 kb/s Low Speed	Convenience Features: trunk release, mirror adjustment
Class B	10 – 125 kb/s Medium Speed	General information Transfer: instruments, power windows
Class C	125 – 1000 kb/s High Speed	Real time control: power train, vehicle dynamics
Class D	> 1000 kb/s	Multimedia applications: Internet, Digital TV, hard real time critical functions

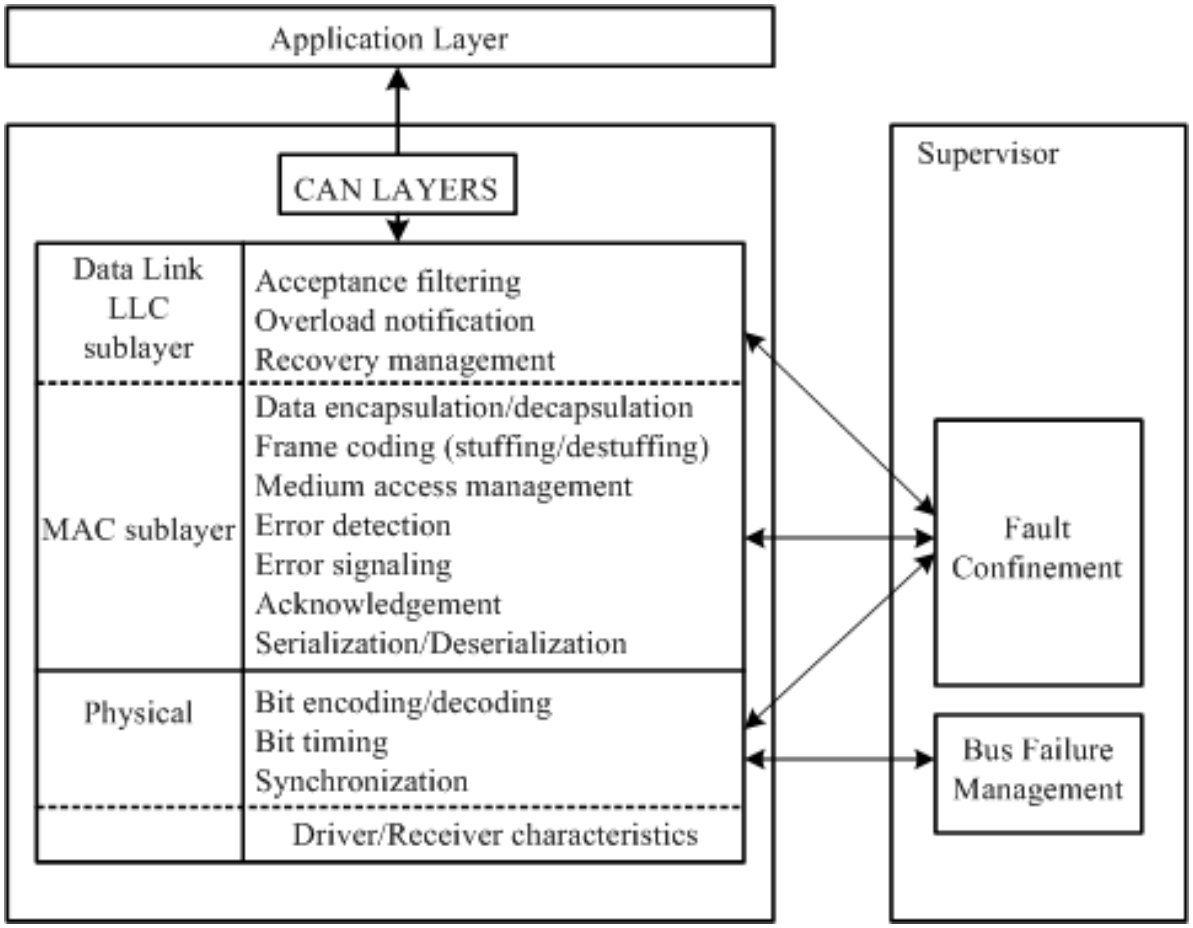
- **CAN (Controller Area Network)**
  - most European vehicle
  - both Class B & Class C
- **LIN (Local Interconnect Network)**
  - all vehicles in future
  - Class A
- **Flexray**
  - New protocol being proposed to replace CAN (Class B, C, D)
- **VAN (Vehicle Area Network)**
  - French vehicles (Renault, …)
  - both Class B & Class C
- **ABUS**
  - VW
- **D2B**
  - Mercedes
  - Class D

Layered Approach in CAN

- Only the logical link and physical layers are described.
- Data link layer is divided into two sub-layers: logical link control (LLC) and medium access control (MAC).
  - LLC sub-layer deals with message acceptance filtering, overload notification, and error recovery management.
  - MAC sub-layer presents incoming messages to the LLC sub-layer and accepts messages to be transmitted forward by the LLC sub layer.
  - MAC sub-layer is responsible for message framing, arbitration, acknowledgement, error detection, and signaling.
  - MAC sub-layer is supervised by the fault confinement mechanism.
- The physical layer defines how signals are actually transmitted, dealing with the description of bit timing, bit encoding, and synchronization.
- CAN bus driver/receiver characteristics and the wiring and connectors are not specified in the CAN protocol.

Layered Approach in CAN (cont)

- System designer can choose from several different media to transmit the CAN signals.



General Characteristics of CAN

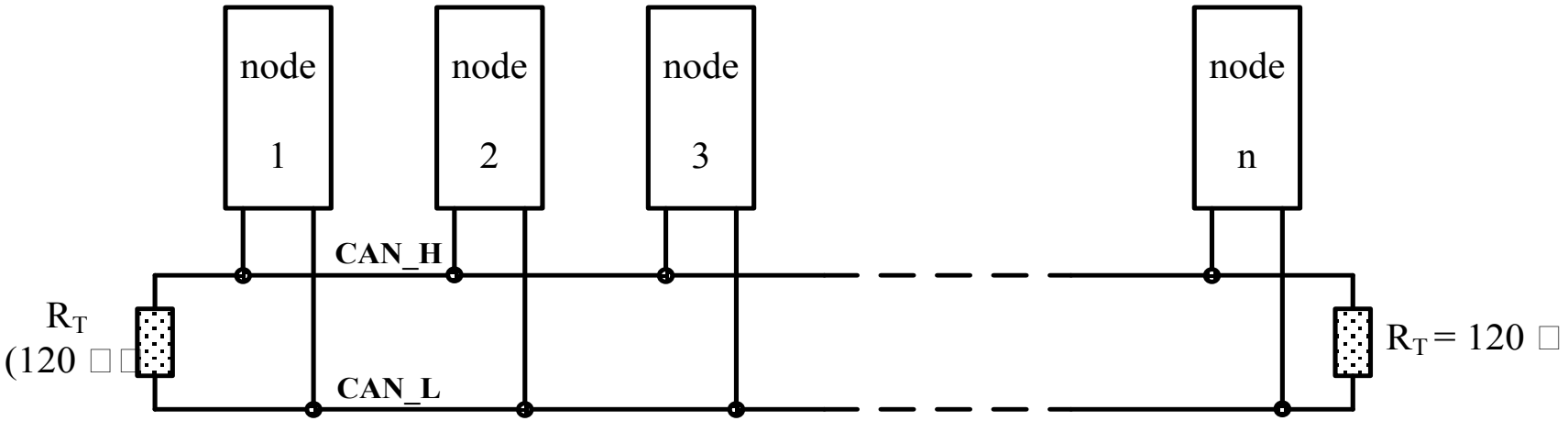
- **Carrier Sense Multiple Access with Collision Detection (CSMA/CD)**
  - Every node on the network must monitor the bus (carrier sense) for a period of no activity before trying to send a message on the bus.
  - Once the bus is idle, every node has equal opportunity to transmit a message.
  - If two nodes happen to transmit simultaneously, a nondestructive arbitration method is used to decide which node wins.
- **Message-Based Communication**
  - Each message contains an identifier.
    - Identifiers allow messages to arbitrate and also allow each node to decide whether to work on the incoming message.
    - The lower the value of the identifier, the higher the priority of the identifier.
  - Each node uses one or more filters to compare the incoming messages to decide whether to take actions on the message.
  - CAN protocol allows a node to request data transmission from other nodes.
  - There is no need to reconfigure the system when a new node joins the system.

General Characteristics of CAN (cont)

- **Error Detection and Fault Confinement**
  - The CAN protocol requires each node to monitor the CAN bus to find out if the bus value and the transmitted bit value are identical.
  - The CRC checksum is used to perform error checking for each message.
  - The CAN protocol requires the physical layer to use bit stuffing to avoid long sequence of identical bit value.
  - Defective nodes are switched off from the CAN bus.

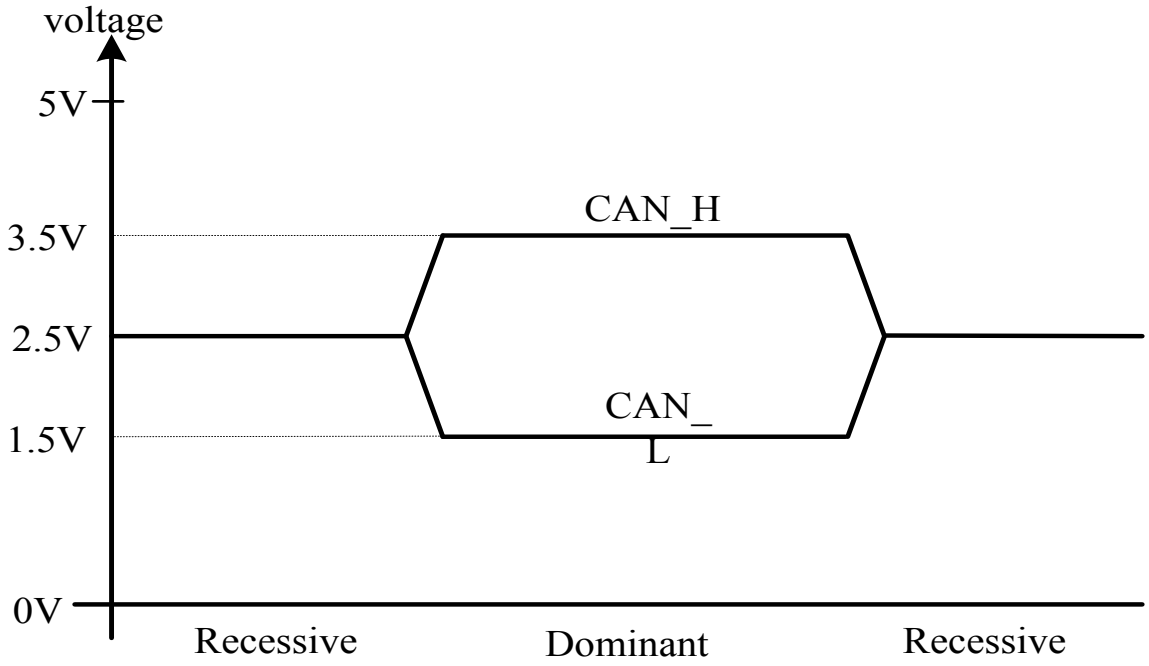
CAN Bus System Topology

- CAN is designed for data communication over a short distance.
- CAN protocol does not specify what medium to use for data communication.
- Using a shielded or unshielded cable is recommended for a short distance communication.
- A typical CAN bus setup is shown below



A typical CAN bus setup

CAN Bus Signaling Levels



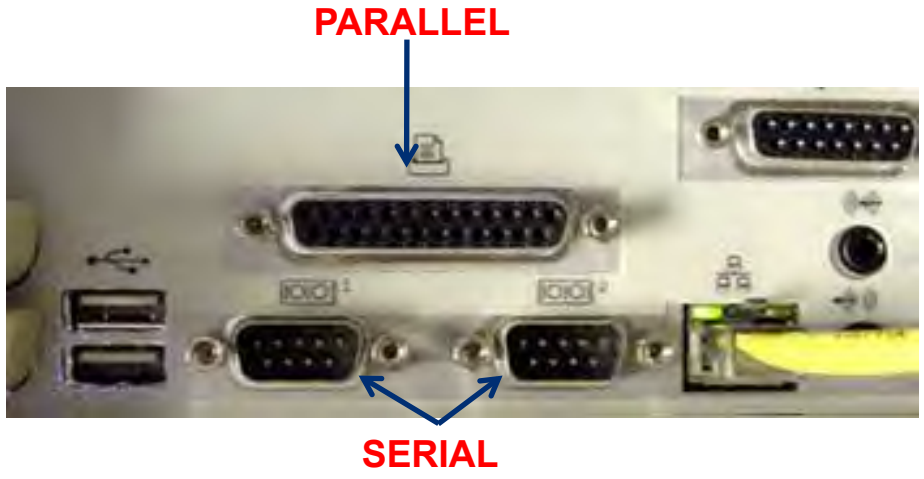
Nominal CAN bus levels

Serial bus protocols

- Universal Serial Bus (USB)
- CAN Bus
- RS232

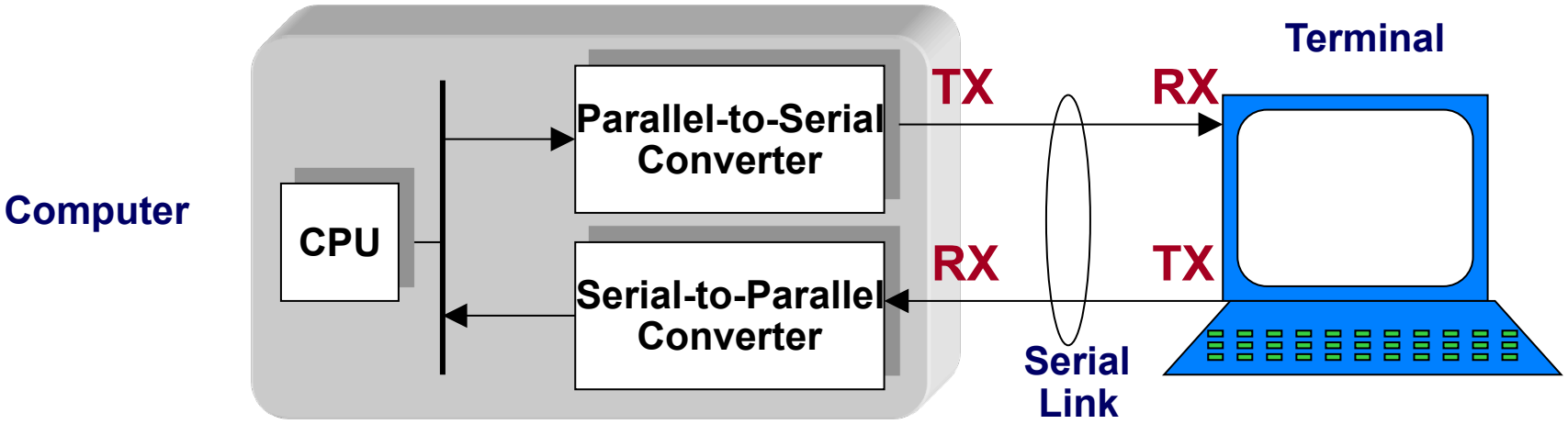
Serial vs. Parallel

- **Serial ports**
  - Universal Asynchronous Receiver/Transmitter (UART): controller
  - Takes the computer bus' parallel data and serializes it
  - Transfer rate of 115 Kbps
  - Example usage: Modems
- **Parallel ports**
  - Sends the 8 bits in parallel
  - 50-100 KBps (standard), upto 2 MBps (enhanced)
  - Example usage: Printers, Zip drives



Data Communication

- **Communications between computer and monitor over serial line**
  - Data is converted from parallel (bytes) to serial (bits) in the bus interface
  - Bits are sent over wire (TX) to terminal (or back from terminal to computer)
  - Receiving end (RX) translates bit stream back into parallel data (bytes)



Types of Serial Communication

- **Two types of configurations**
  - point-to-point: two end stations communicate as peers
  - multi-drop: one device is designated as master (primary), the other devices are slaves (secondaries)
- **Physical link can consist of either two or four wires**
  - Two wire link provides signal and ground wires
  - Four wire link provides two sets of signal and ground wires
- **Full-duplex link**
  - One signal wire is for transmitting, the other for receiving
  - Closed loop: individual characters are echoed back to transmitter
  - Open loop: data is assumed to reach its destination
- **Half-duplex link**
  - One signal wire is for both transmitting and receiving

Data Modulation

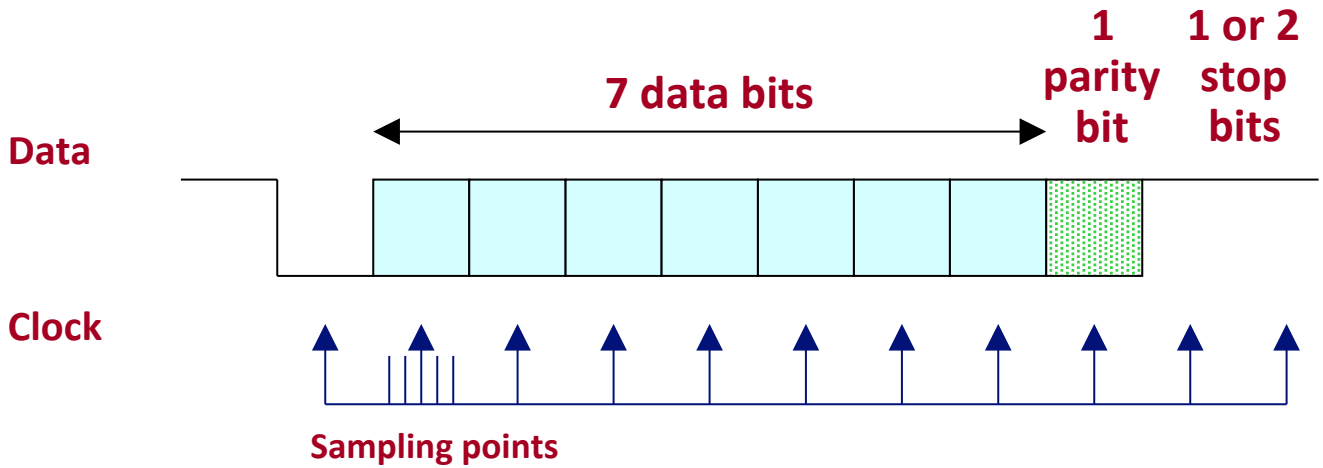
- When sending data over serial lines, logic signals are converted into a form the physical media (wires) can support
- RS232C uses bipolar pulses
  - Any signal greater than +3 volts is considered a mark
  - Any signal less than -3 volts is considered a space
- 20 mA current loop
  - Uses four wires, trasmit+, transmit-, receive+ and receive-
  - Voltage is applied at one end and the current travels to the far end, passes through a load resistor and then returns to the transmitter
  - HIGHs and LOWs are determined by the presence or absence of 20 mA current
- Other schemes modulate the amplitude or frequency of a frequency carrier signal

Serial Communications Protocols

- A Communications protocol is a convention for data transmission that includes such functions as timing, formatting and data representation
- Two categories of protocols:
  - Asynchronous protocols
  - successive data appear in the data stream at arbitrary times, with no specific clock control governing the relative delays in data
  - Synchronous protocols
  - each successive datum in a stream of data is governed by a master data clock and appears at a specific interval of time
    - Often, protocols deliver serial data in 8-bit characters -- asynchronous protocols treat each character as an individual message, and the characters appear in the data stream at arbitrary relative times. However, within each character, the bits are transmitted at a fixed predetermined clock rate

Asynchronous Protocols

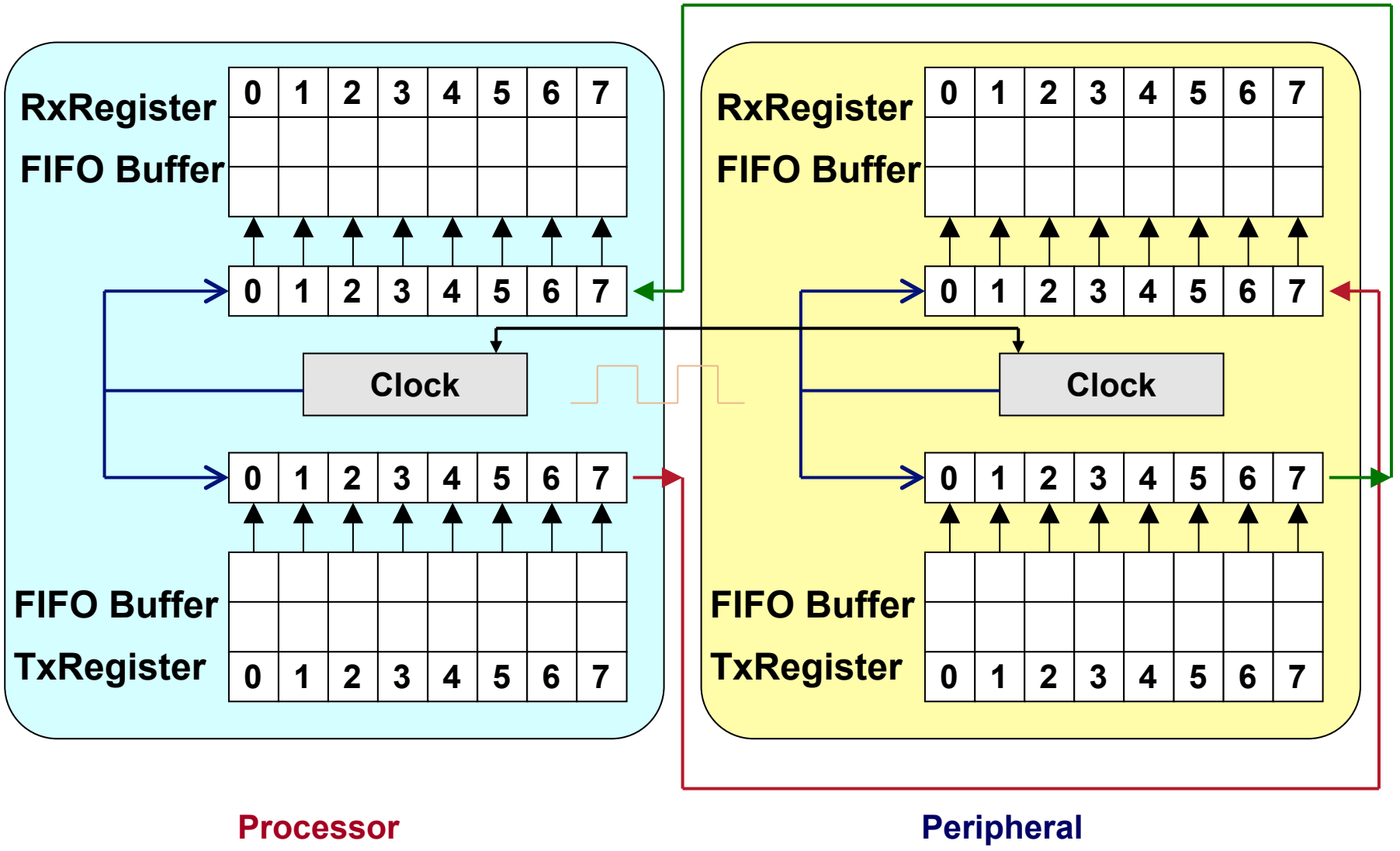
- Many different types of electrical connection standards
  - RS-232-C
  - 20 mA current loop
  - RS-422, RS-423, and RS-499
- Timing is the same
  - idle line is assumed to be in high (1) state
  - each character begins with a zero (0) bit, followed by 8 data bits and then 1, 1-1/2, or 2 closing 1 bits.
  - Bits are usually encoded using ASCII (American Standard Code for Information Interchange)



Asynchronous Protocols (cont'd)

- Start (stop) bits identify the beginning (end) of each character -- also permits receiver to resynchronize local clock to each new character
  - Remember: characters can begin at arbitrary times
- Bit Sampling
  - Receiver's clock is not identical to the transmitter's clock
  - Best if the sample is near the middle of a bit
  - Not always possible because of clock differences
  - Receiver clock (in relation to transmitter) cannot gain or lose more than 1/2 bit per 10 to 11 clock periods (time to transmit one character)
    - Clocks must be accurate to within 5%
  - Most receivers use a fast clock to determine the "middle" of a bit
    - Typical is 16X clock which can take 16 samples per 1 bit
      - Begins by detecting start bit which clears clock counter
      - Clock counter increments to 16 for each tick of the receiver clock
      - When the counter reaches 8, it has reached the middle of the bit and takes the sample

Serial Port


RxRegister
FIFO Buffer
0 1 2 3 4 5 6 7
Clock
FIFO Buffer
TxRegister
Processor
Peripheral

Transmitting Bits

Transmitter

n	0	1	2	3	4	5	6	7
n+1		0	1	2	3	4	5	6
n+2			0	1	2	3	4	5
n+3				0	1	2	3	4
n+4					0	1	2	3
n+5						0	1	2
n+6							0	1
n+7								0
n+8								

Interrupt when Transmitter (Tx) is empty

Receiver

n								
n+1	7							
n+2	6	7						
n+3	5	6	7					
n+4	4	5	6	7				
n+5	3	4	5	6	7			
n+6	2	3	4	5	6	7		
n+7	1	2	3	4	5	6	7	
n+8	0	1	2	3	4	5	6	7

Interrupt when Receiver (Rx) is full

Coping with Errors - Parity

- A single bit determined by the number of 1's in a word
- The simplest form of error detection
- Parity bit is appended to each group-of-bits (byte, word)
- Odd parity
  - Parity bit is chosen to force the number of 1's to be odd
  - Example: 0 0 0 0 1 0 1 0
    0 0 0 0 1 0 1 0 1 ← parity bit
- Even parity
  - Parity bit is chosen to force the number of 1's to be even
- When receiving data, the receiver will check every group of bits
  - Parity checker determines if the parity of a group of bits is correct
  - If wrong, an error (exception) will be raised to let the processor and software know an error has occurred in parity bit

Interfacing Serial Data to Microprocessor

- Processor has parallel buses for data -- need to convert serial data to parallel (and vice versa)
- Standard way is with UART
- UART - Universal asynchronous receiver and transmitter
  - USART - Universal synchronous and asynchronous receiver and transmitter