COL216
Computer Architecture

Input/Output – 1

14th March 2022

Diversity of Input/Output devices

Communicate with

- Human
- Computer
- Physical systems
- Storage

Variations in terms of

- direction
- speed
- data type . . .

Communication with humans

- Key board, mouse, joy stick, tablet, touch, gesture

  very slow, 10^1-10^2 bytes per sec
- Documents: Scanners, printers, plotters

  8-16 ppm, 300-1200 dpi, 10^2-10^3 KB/sec
- Sound (voice/music): mikes, speakers

  10^1-10^3 KB/sec
- Graphic displays

  mega pixels, refresh rate, 10^1-10^3 MB/sec

Communication with machines

- Wired network controllers

  10^1-10^3 MB/sec

- Wireless network controllers
  - ethernet, blue tooth, zigbee etc

  10^0-10^1 MB/sec

- Modems

  10^1-10^3 KB/sec

Communication with physical systems

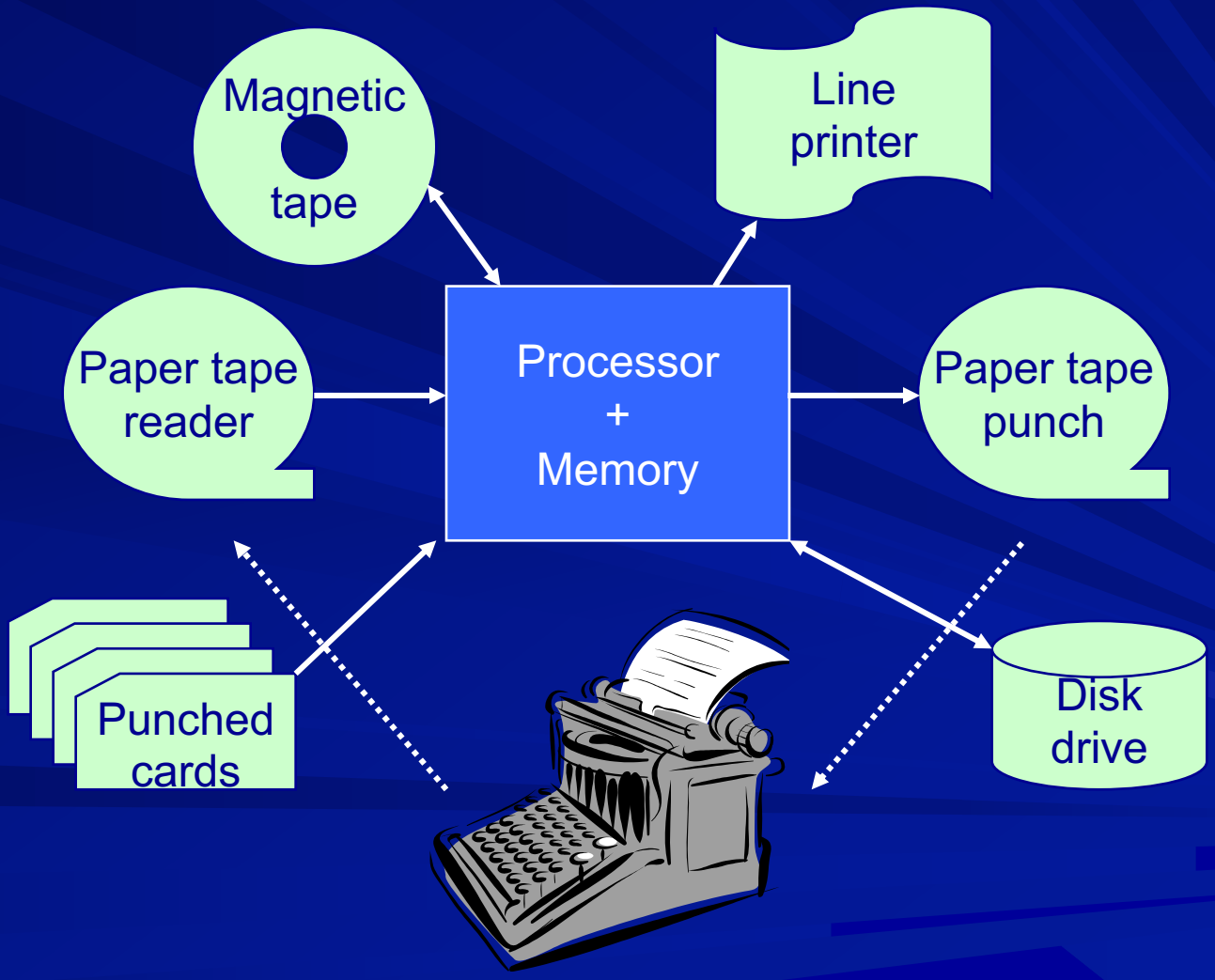
- Sensors
  - pressure, temperature, humidity, flow, level, position, speed, proximity, gas
  - audio, visual
- Actuators
  - switches, valves, solenoid, motors

Signal forms: on/off, analog, pulse train etc

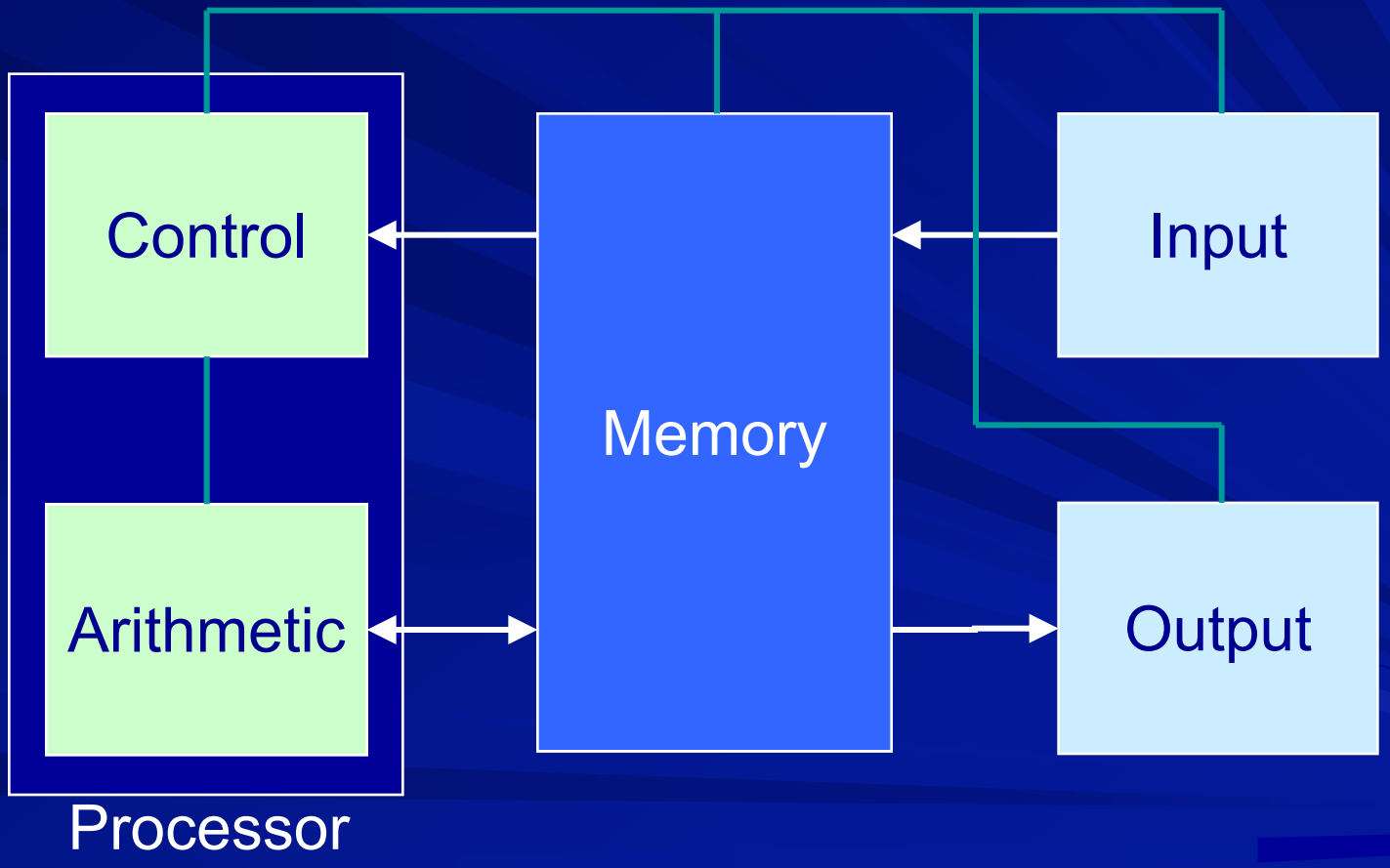
Storage devices

- Magnetic
  - disks, tapes
- Optical
  - CDROM, DVD, Blue Ray
- Semi-conductor
  - Nor Flash, Nand Flash

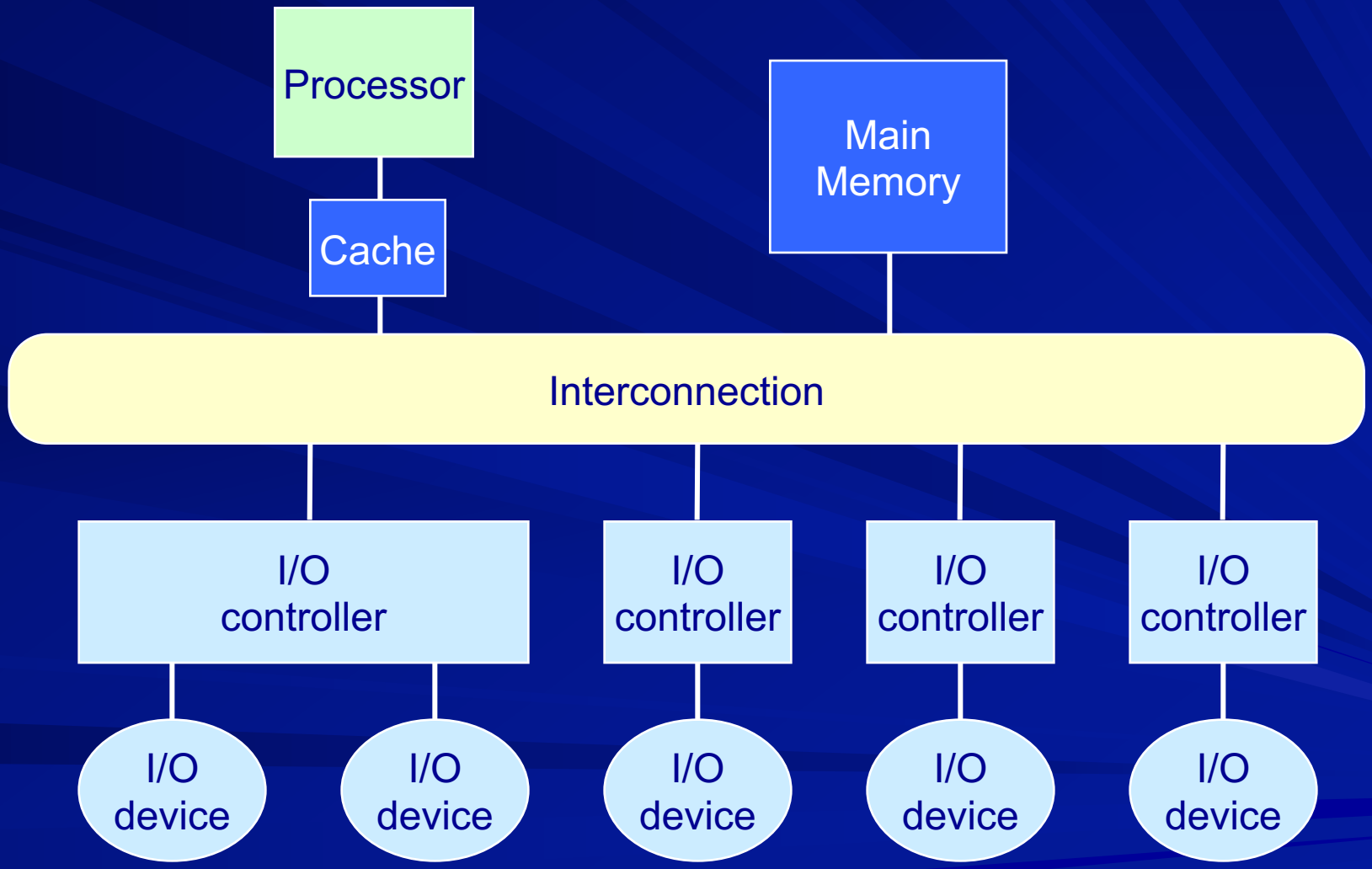
Peripherals circa 1970 : ICL 1909



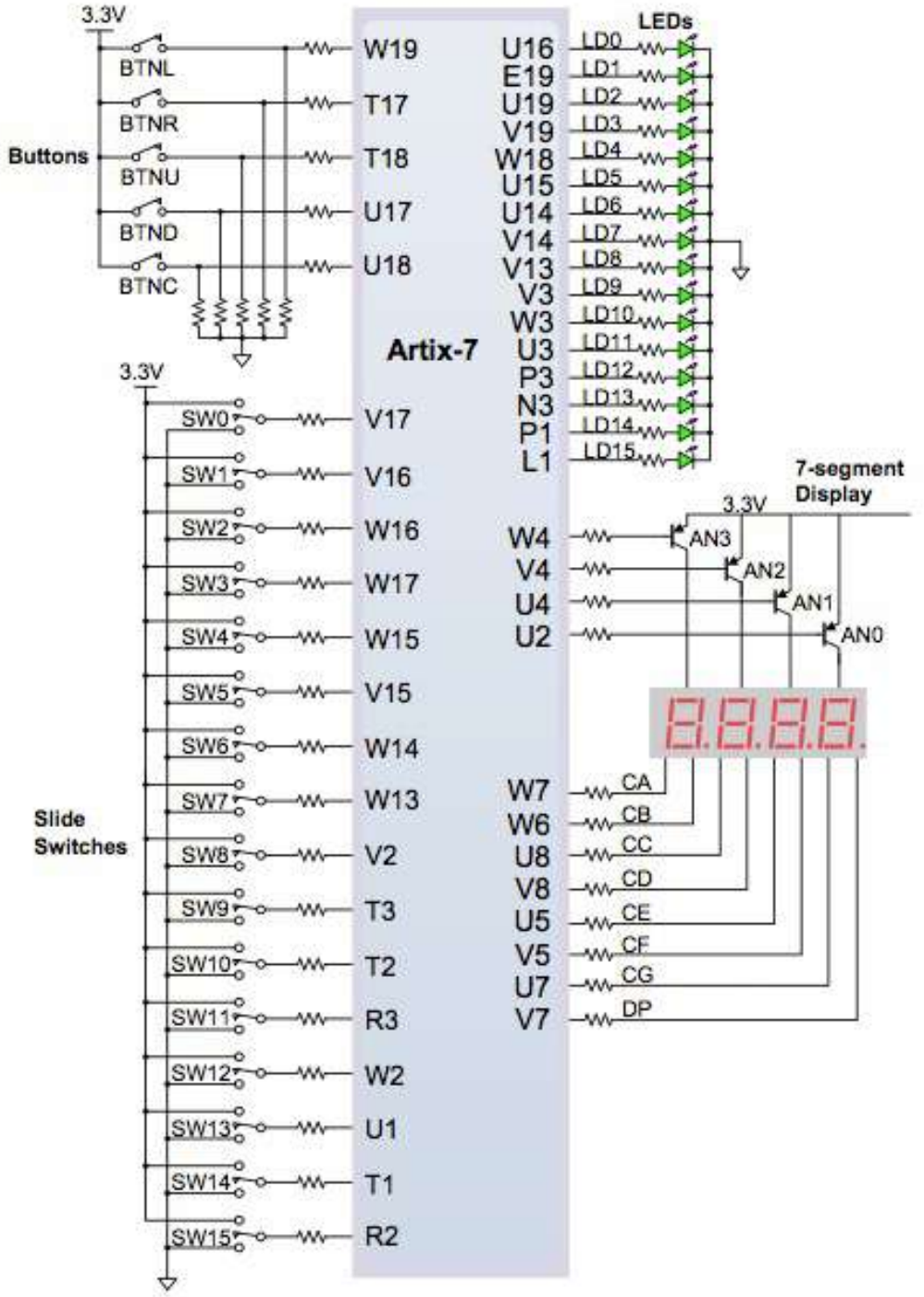
Computer System: Simplified Block Diagram



More realistic block diagram

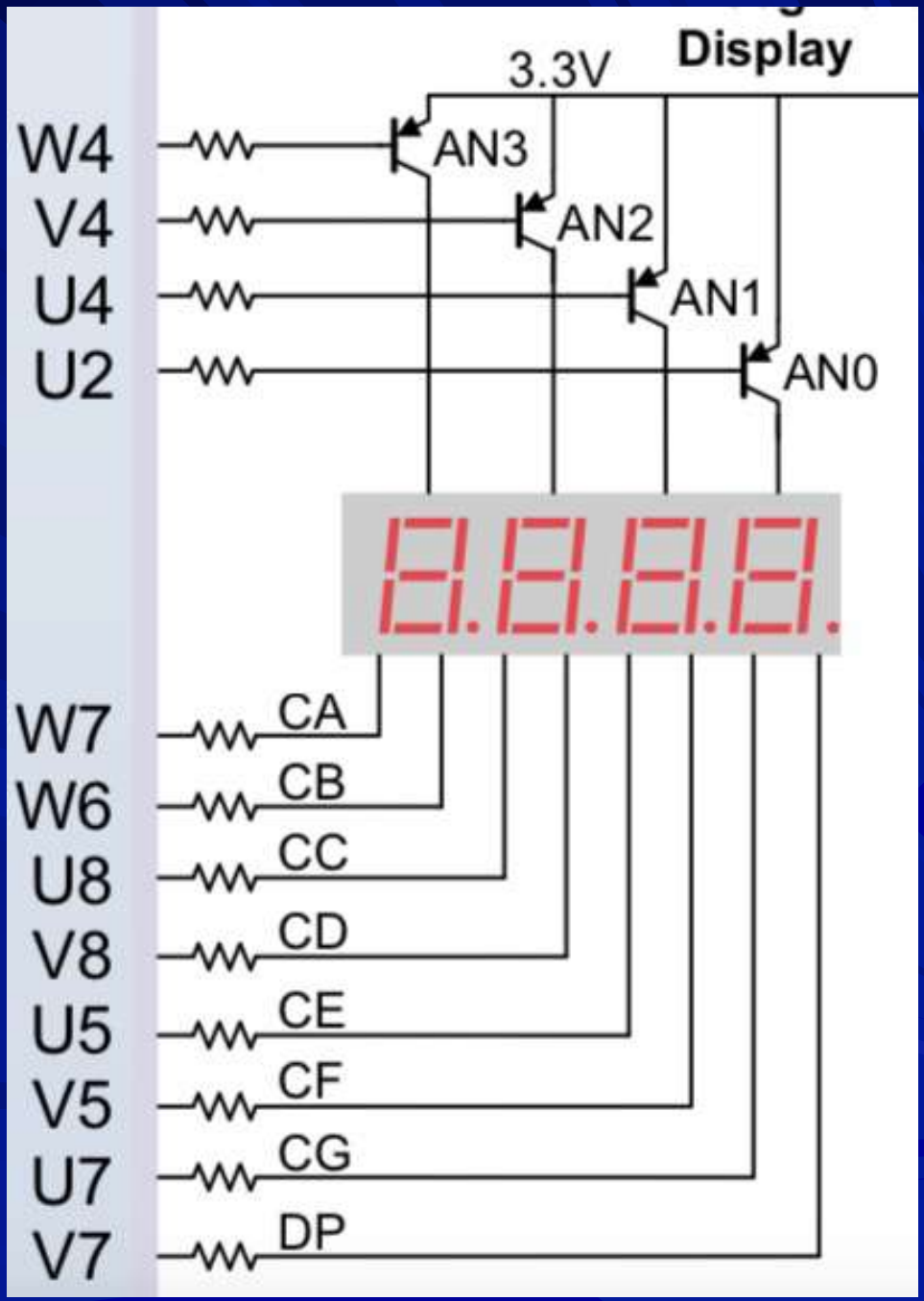


Switches and displays on BASYS3 board

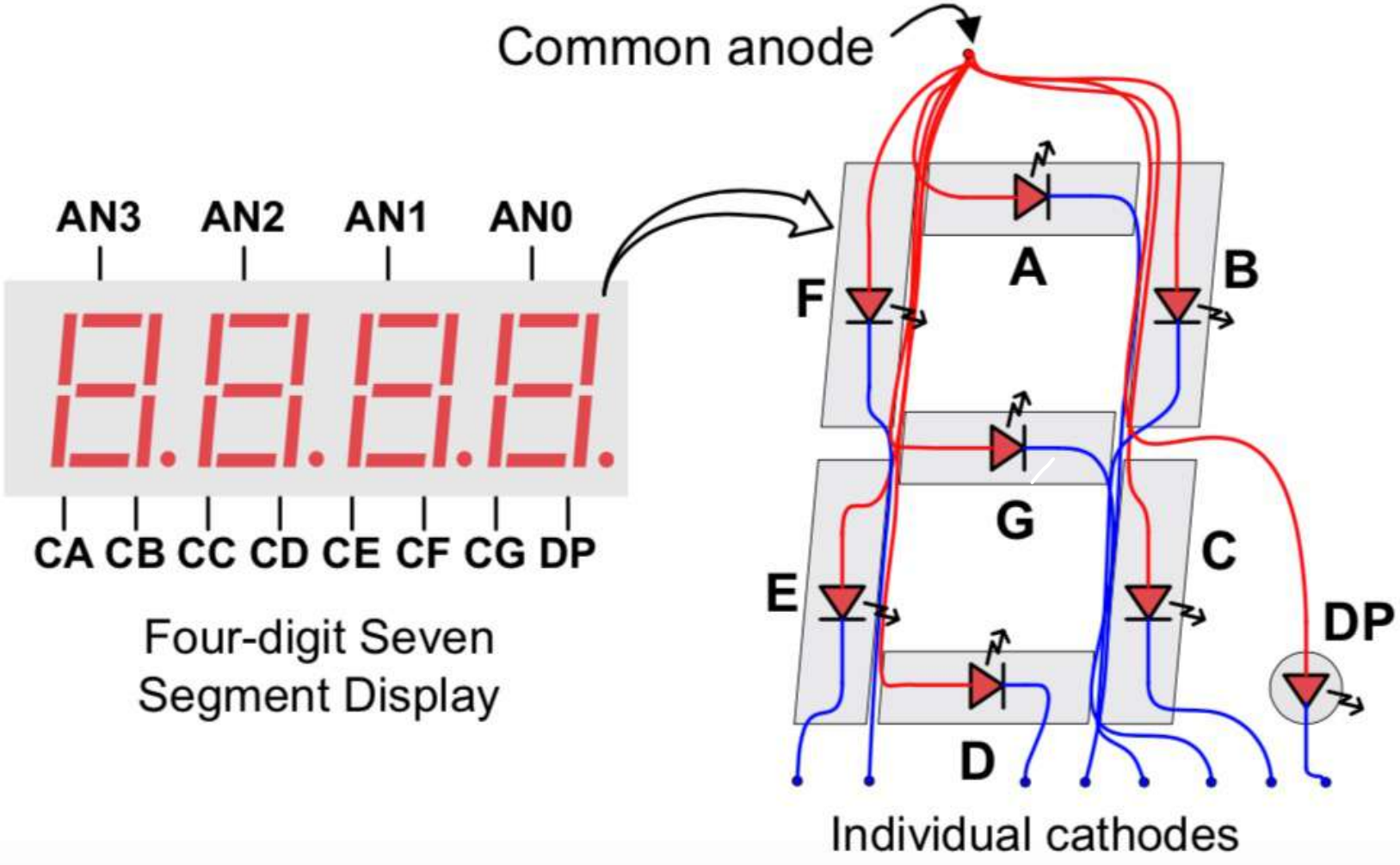


16 Button Key-pad

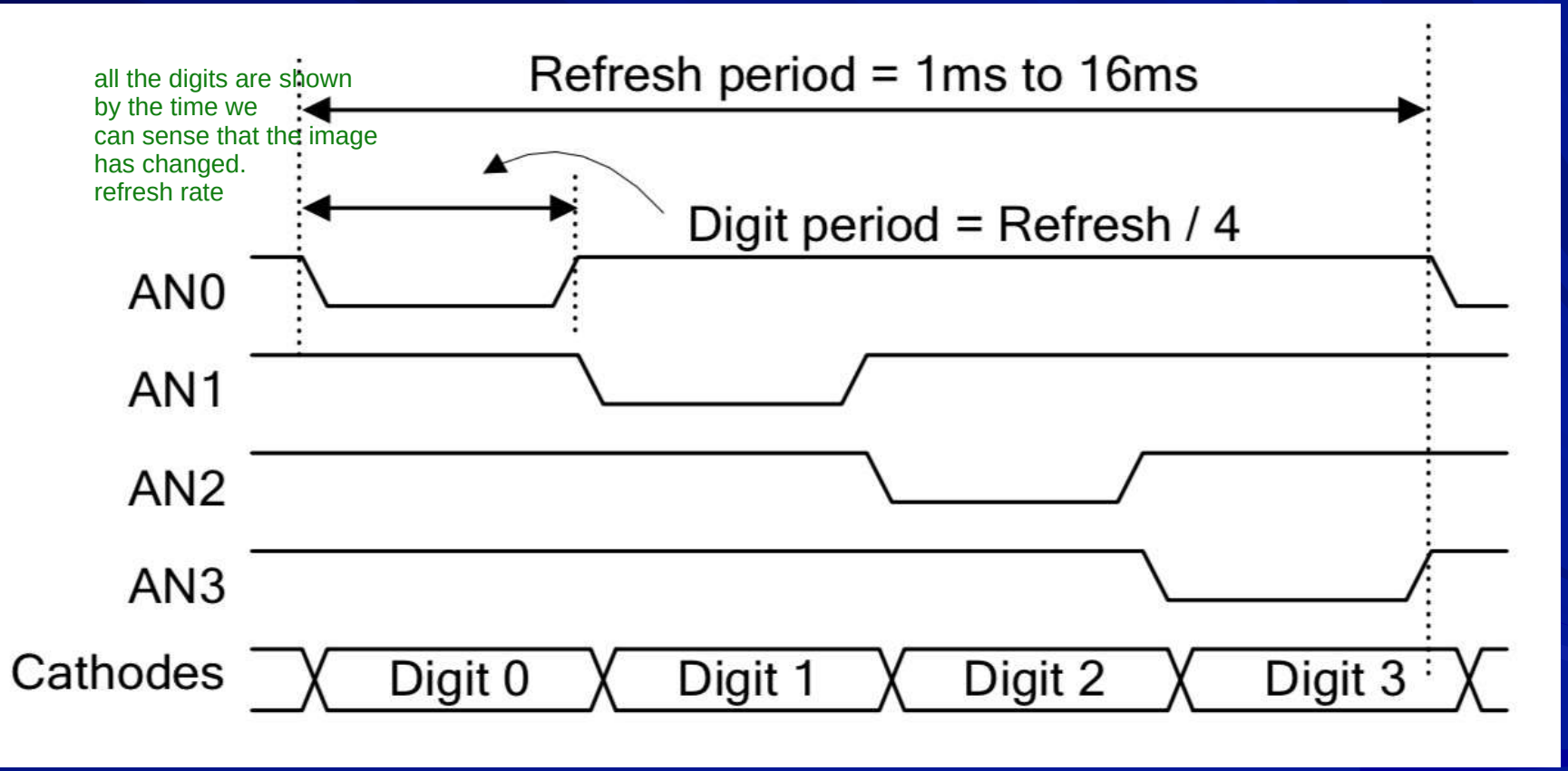
4-digit Display on BASYS 3 Board



7-segment Display

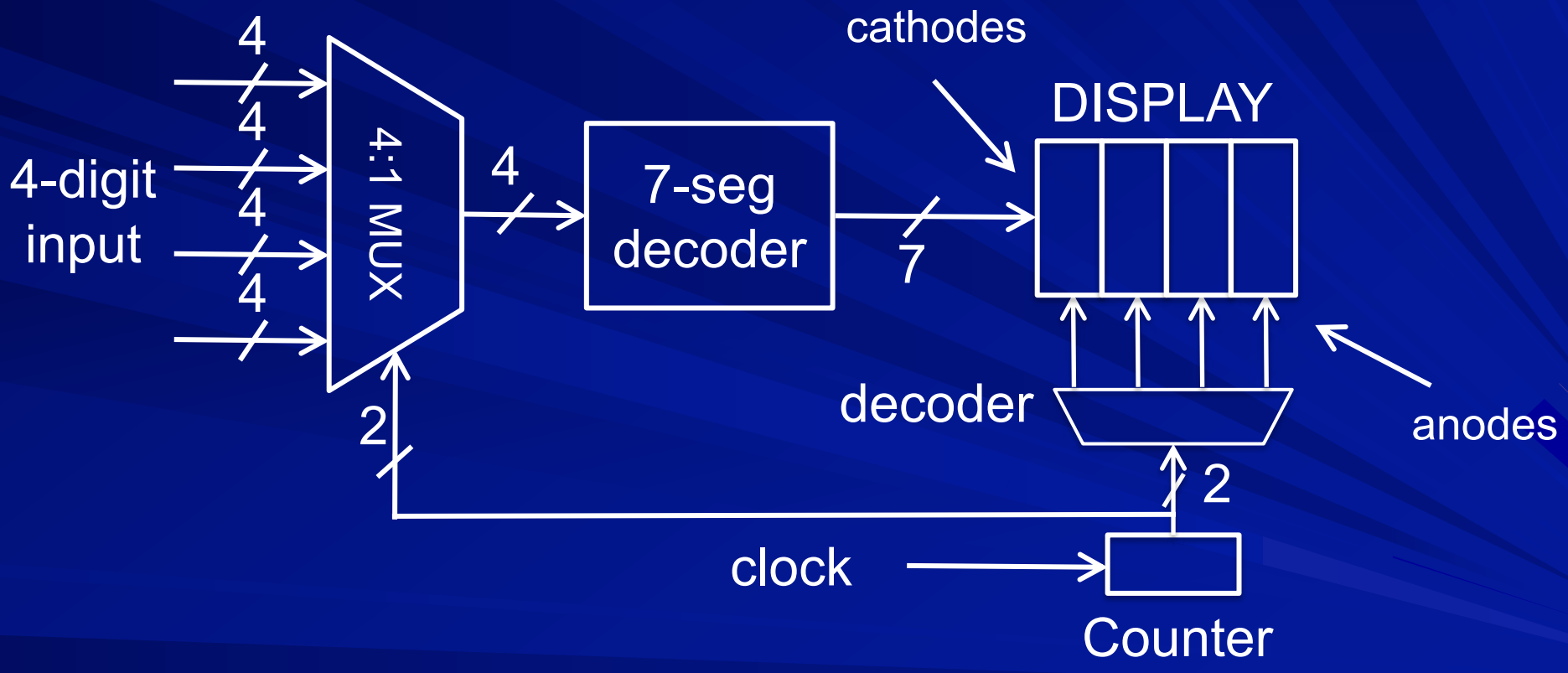


Refresh timings



Processor simply gives the digits. display controller displays the digit

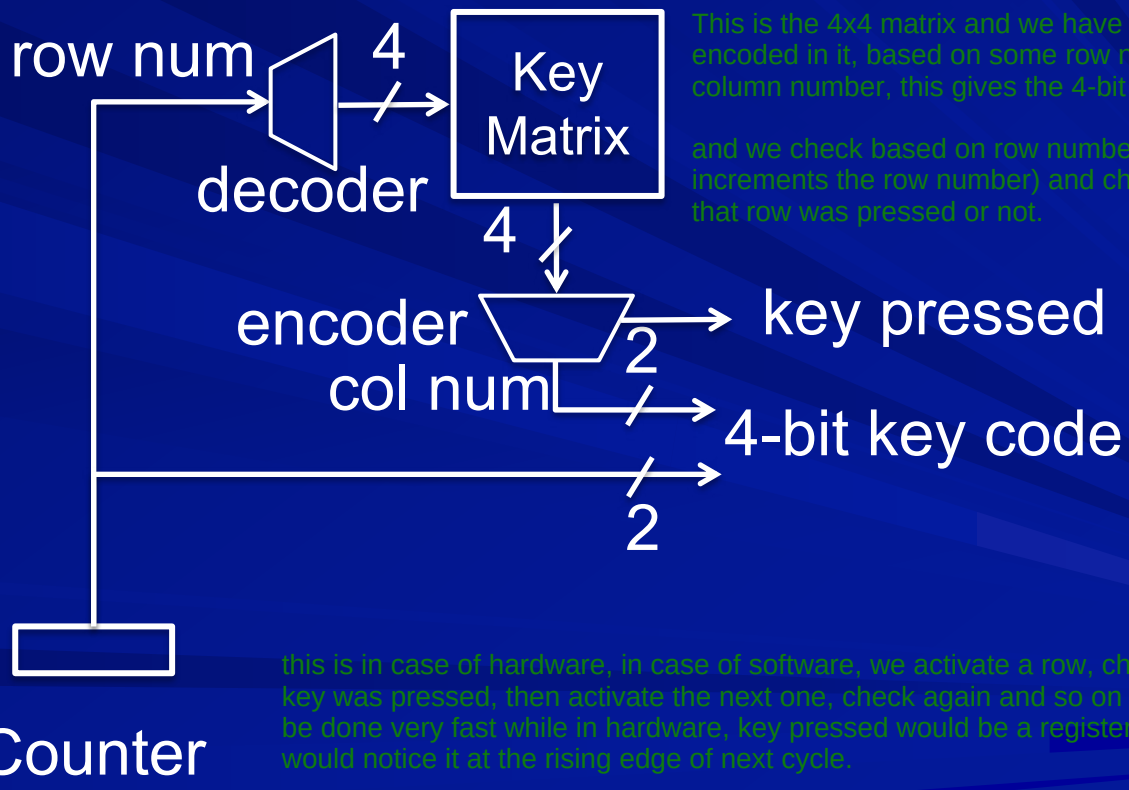
7-segment Display Controller



digits are being displayed in sequence, at a rate faster than our persistence of vision so we are able to see all the lit up pixels simultaneously when actually they are happening in sequence

4 x 4 Keyboard Encoder

We have many keys on the keyboard, so we cant have so many wires moving from the keyboard to the screen, hence we have a decoder

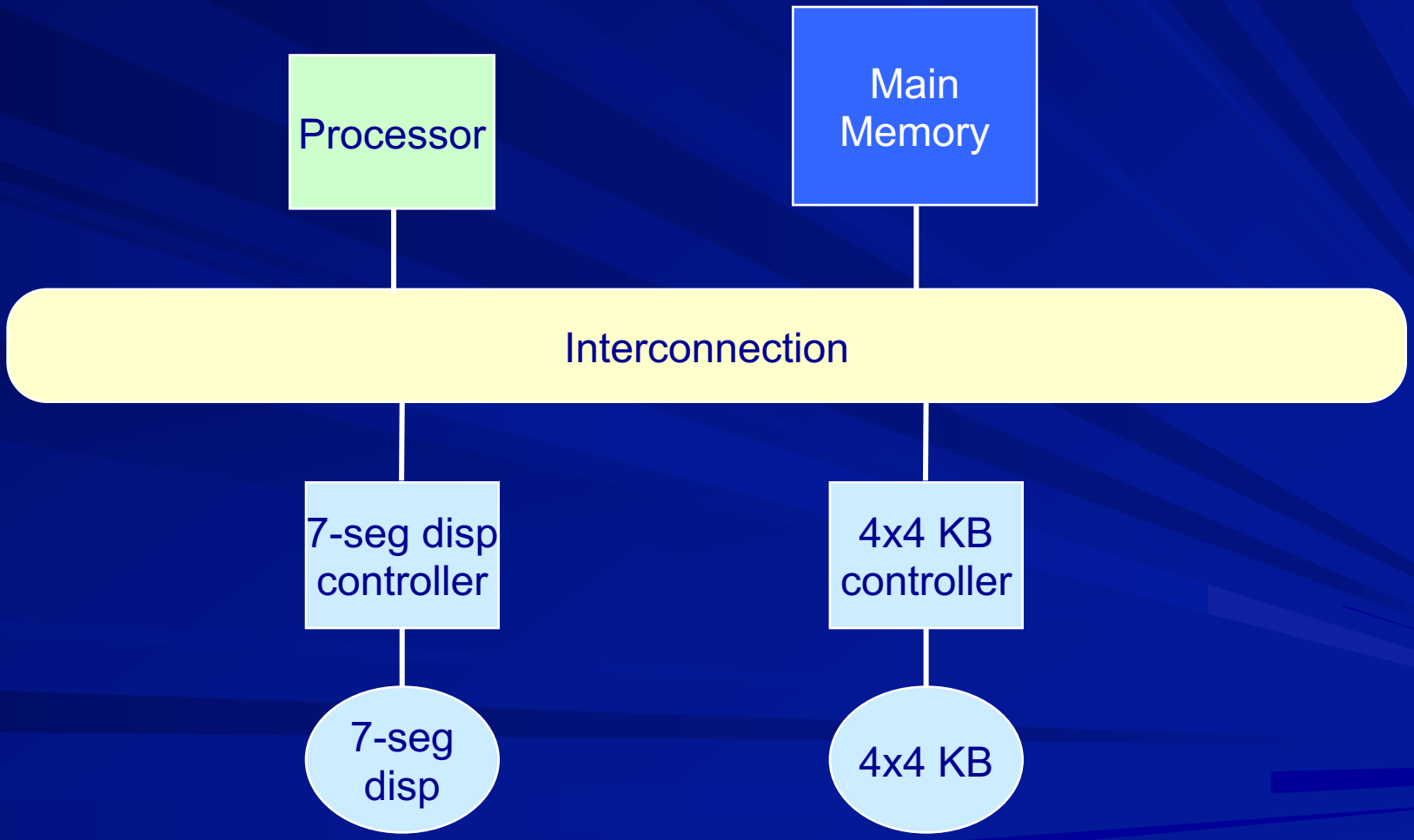


This is the 4x4 matrix and we have each key encoded in it, based on some row number and column number, this gives the 4-bit key code.

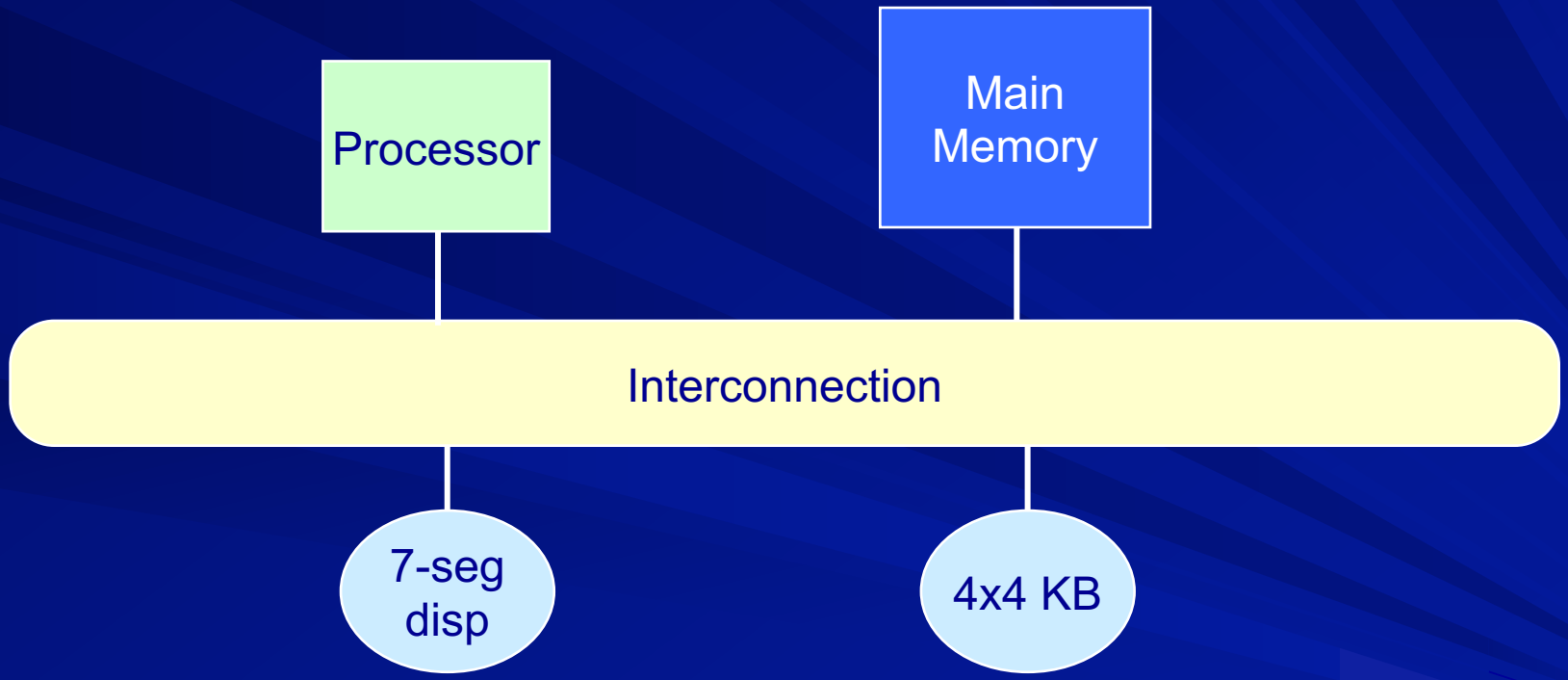
and we check based on row number (counter increments the row number) and check if any key in that row was pressed or not.

this is in case of hardware, in case of software, we activate a row, check if any column key was pressed, then activate the next one, check again and so on and this needs to be done very fast while in hardware, key pressed would be a register so processor would notice it at the rising edge of next cycle.

Block diagram



Block diagram



Low level control by software

Instructions for I/O

Use SWI instruction/system call

Require instruction for I/O inside handler

- Special input/output instructions
  - specify device
  - specify register/memory location

=> Dedicated I/O

- Use LDR, STR instructions
  - use devices as memory locations

=> Memory mapped I/O

I/O address space

- Memory mapped I/O
  - part of memory address space reserved for I/O
  - usual load/store instructions are used for I/O
  - memory ignores these addresses
- Dedicated I/O
  - separate address space for I/O - much smaller than the memory space
  - control signal from processor indicates which space is being addressed
  - separate instructions are required

Addressing I/O devices

Each I/O device (controller) viewed as a set of registers or ports by CPU

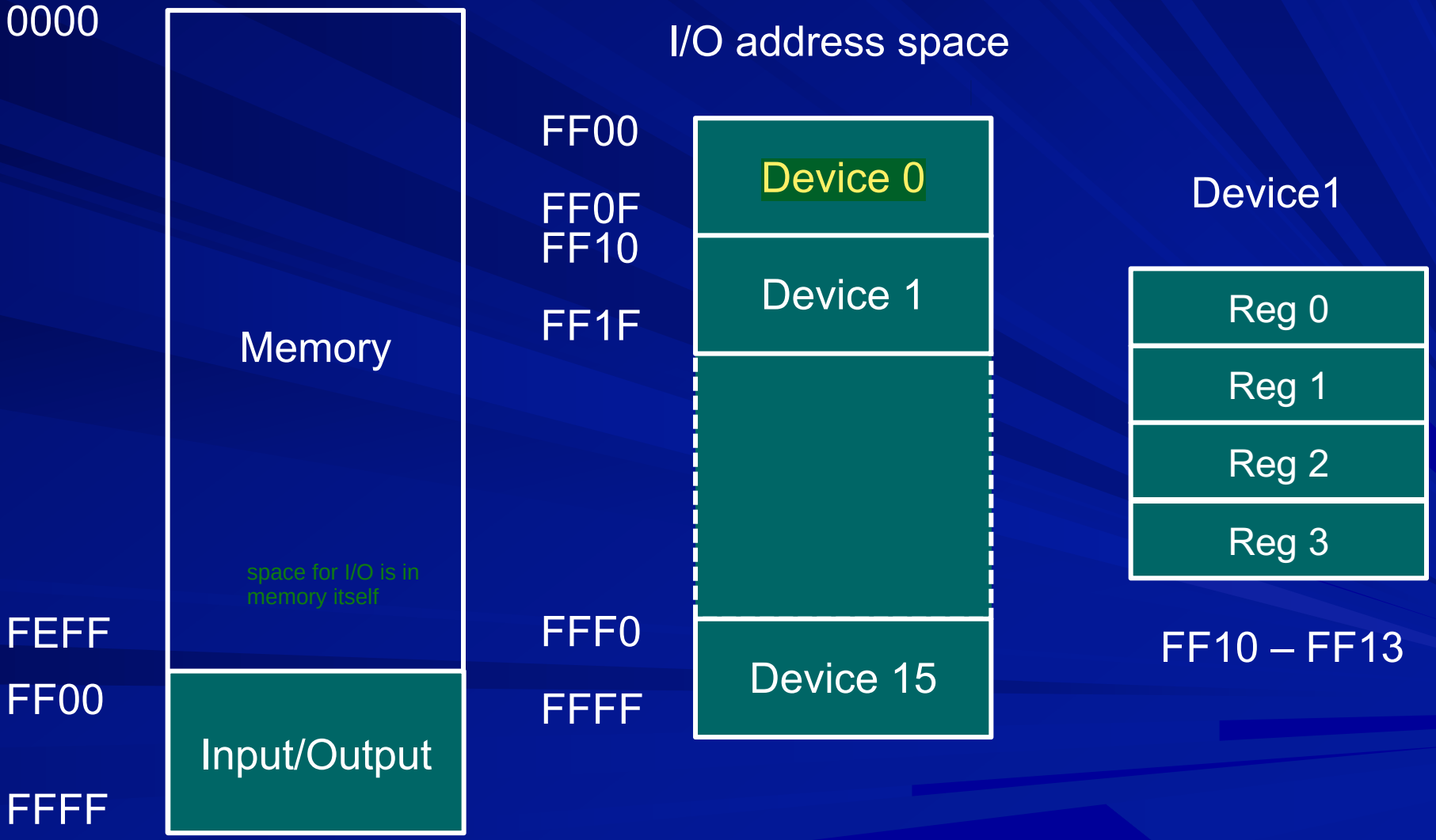
CPU can use these registers to

- write commands and parameters
- read status
- read/write data

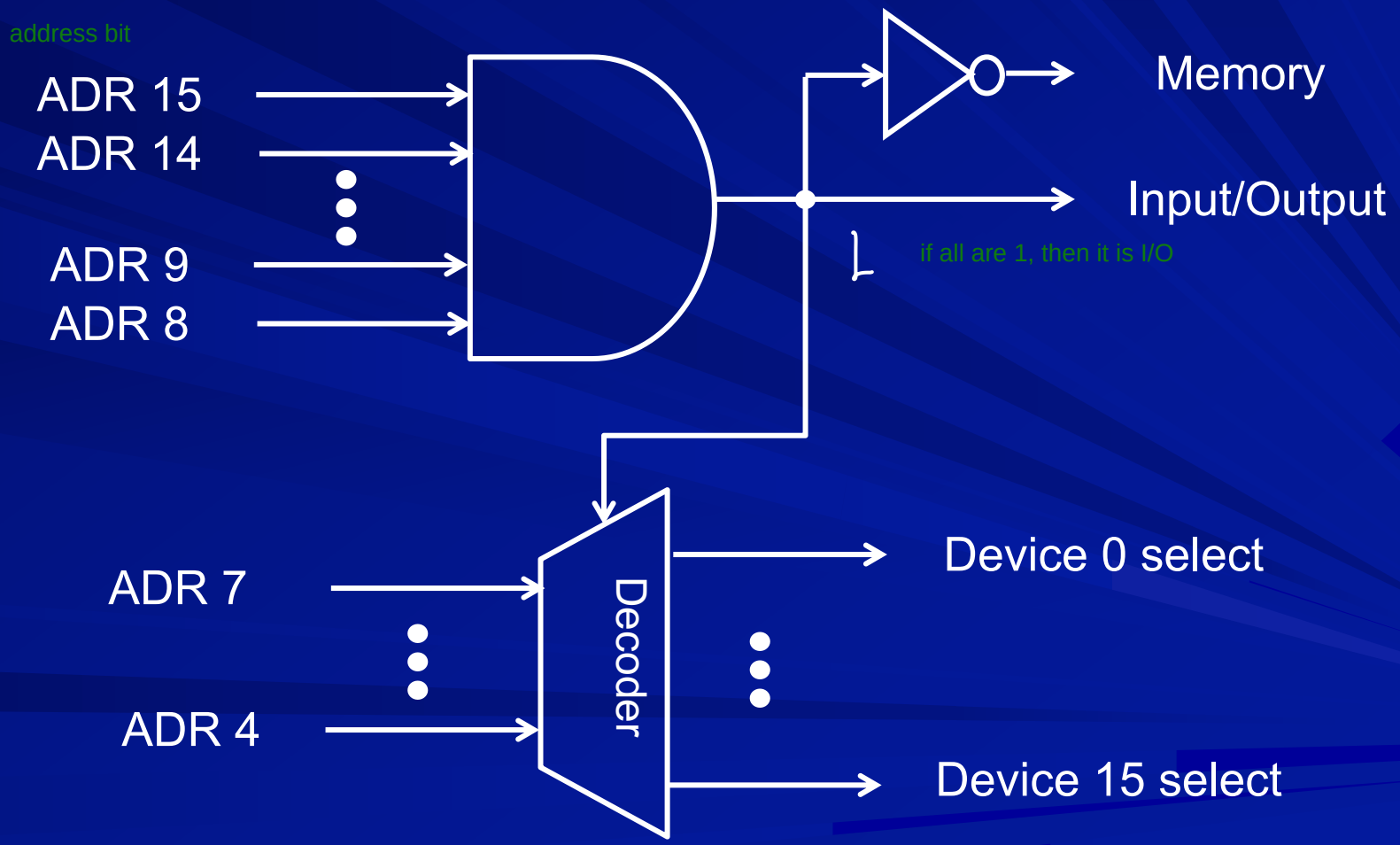
These registers are addressable

- Memory mapped I/O
- Dedicated I/O

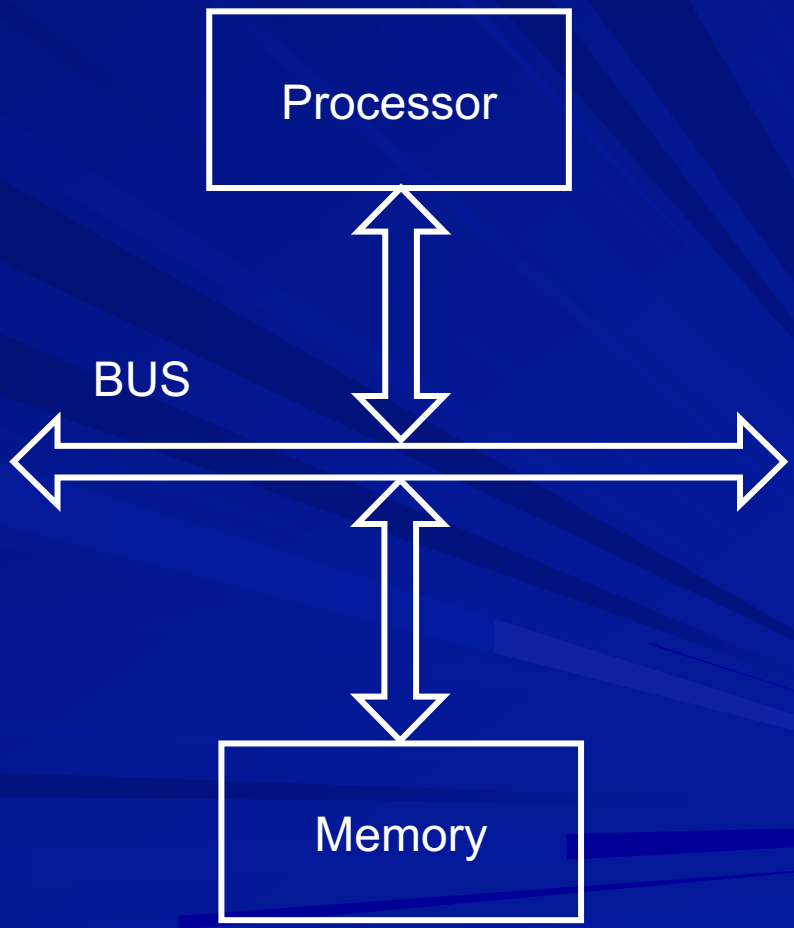
Address Map Example



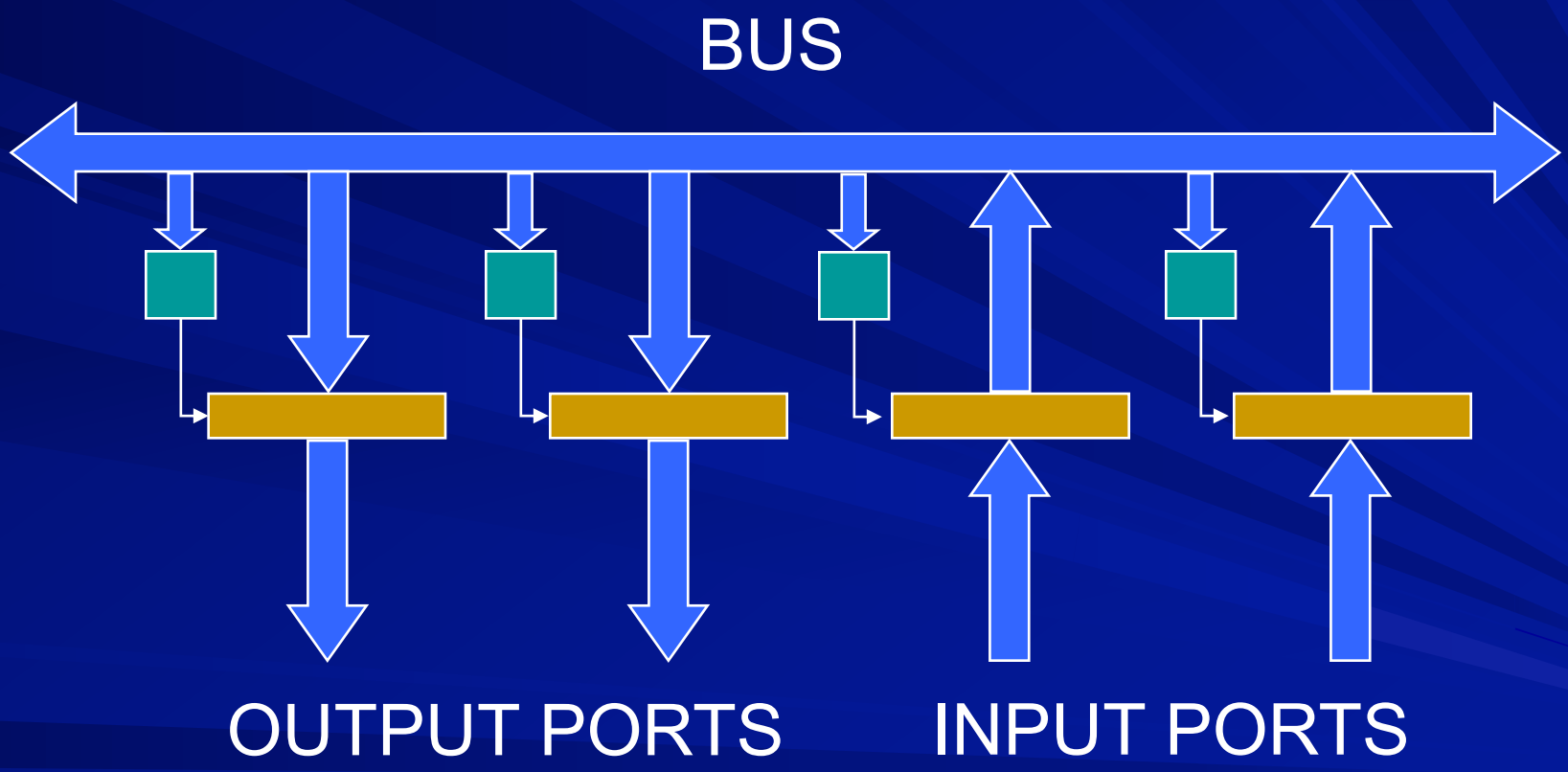
Decoding Address



Interconnection using a bus



INPUT / OUTPUT PORTS



controlled by hardware

Ports for Disp, KB controllers

7 segment display controller

Display

out	Digit 0
out	Digit 1
out	Digit 2
out	Digit 3

input

Key Board

key has been pressed or not

in	status
in	key code

controlled by software

Ports for raw Disp, KB

raw means low level control

Display

out — anode pattern — 4 bit anode

out — cathode pattern — 7 bit cathode

since we do this for 1 digit, so we need to refresh every 10 ms

Key Board

out — row pattern

in — column pattern

then we won't deal with code, we shall be dealing with row pattern and column pattern directly.

refresh required

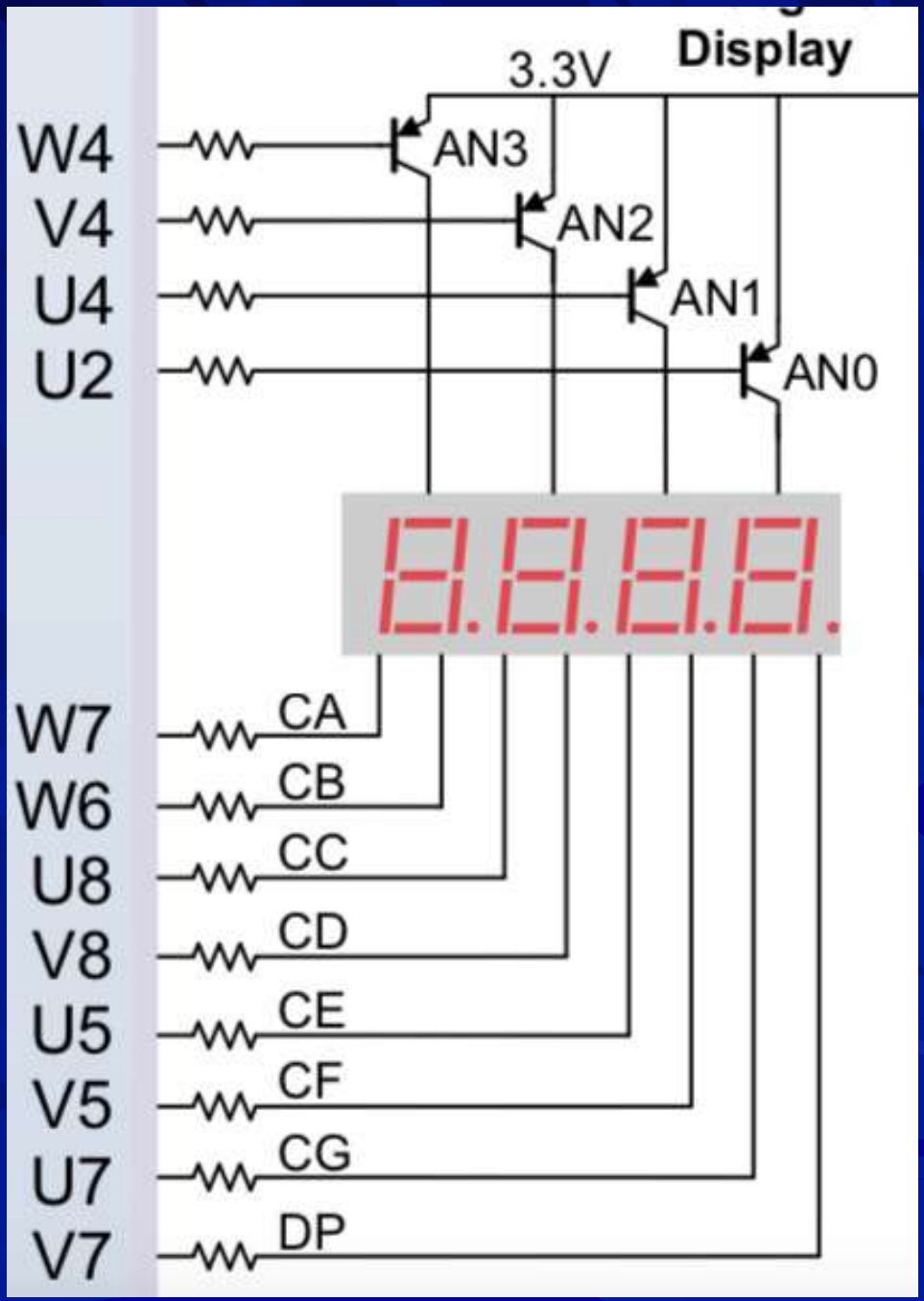
repeated scan required

COL216
Computer Architecture

Input/Output – 2

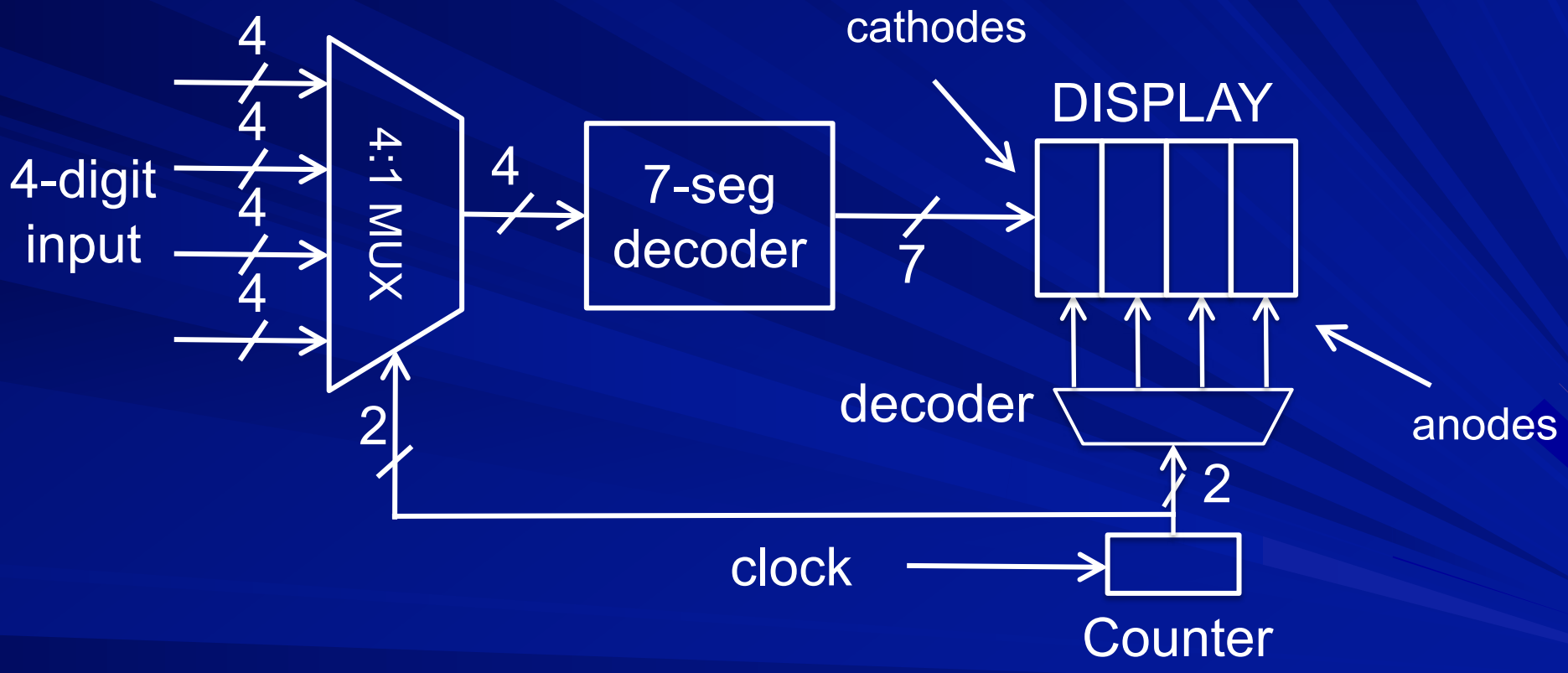
17th March 2022

4-digit Display on BASYS 3 Board

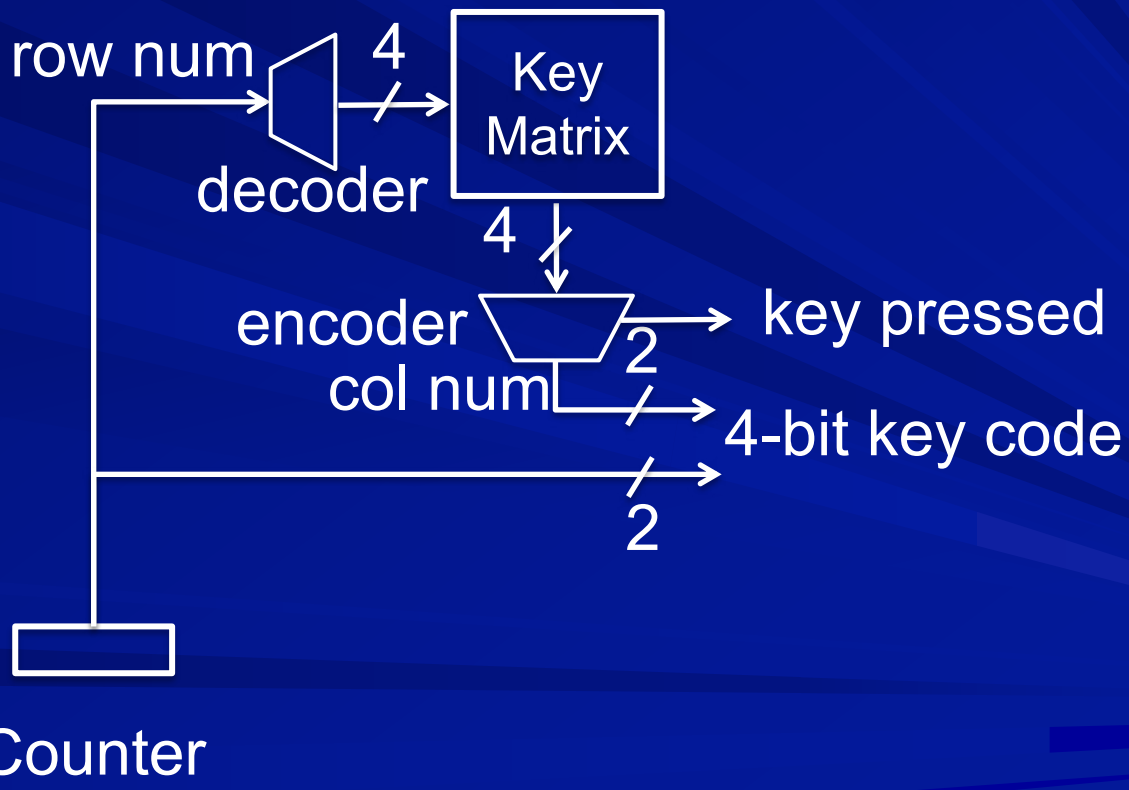


16 Button Key-pad

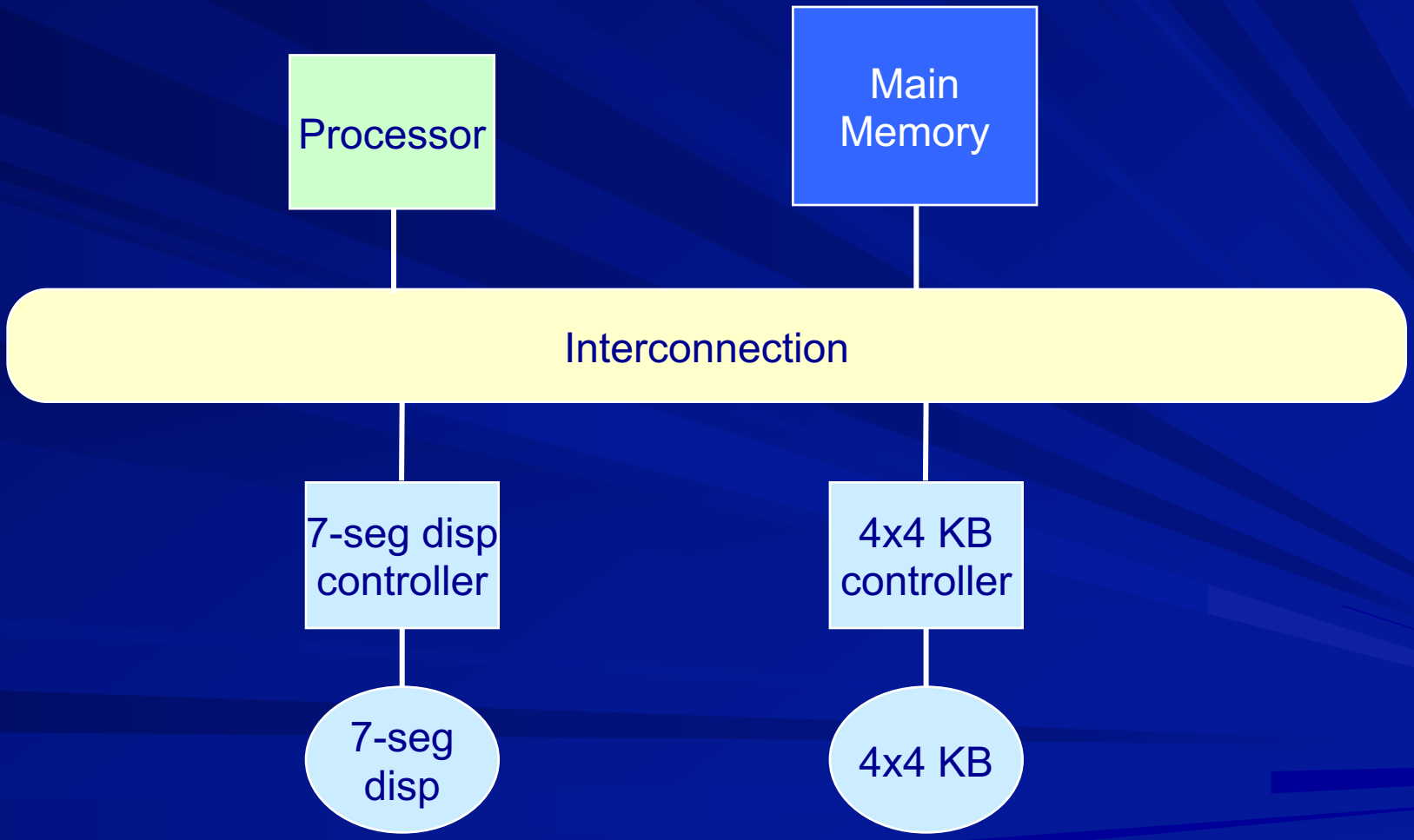
7-segment Display Controller



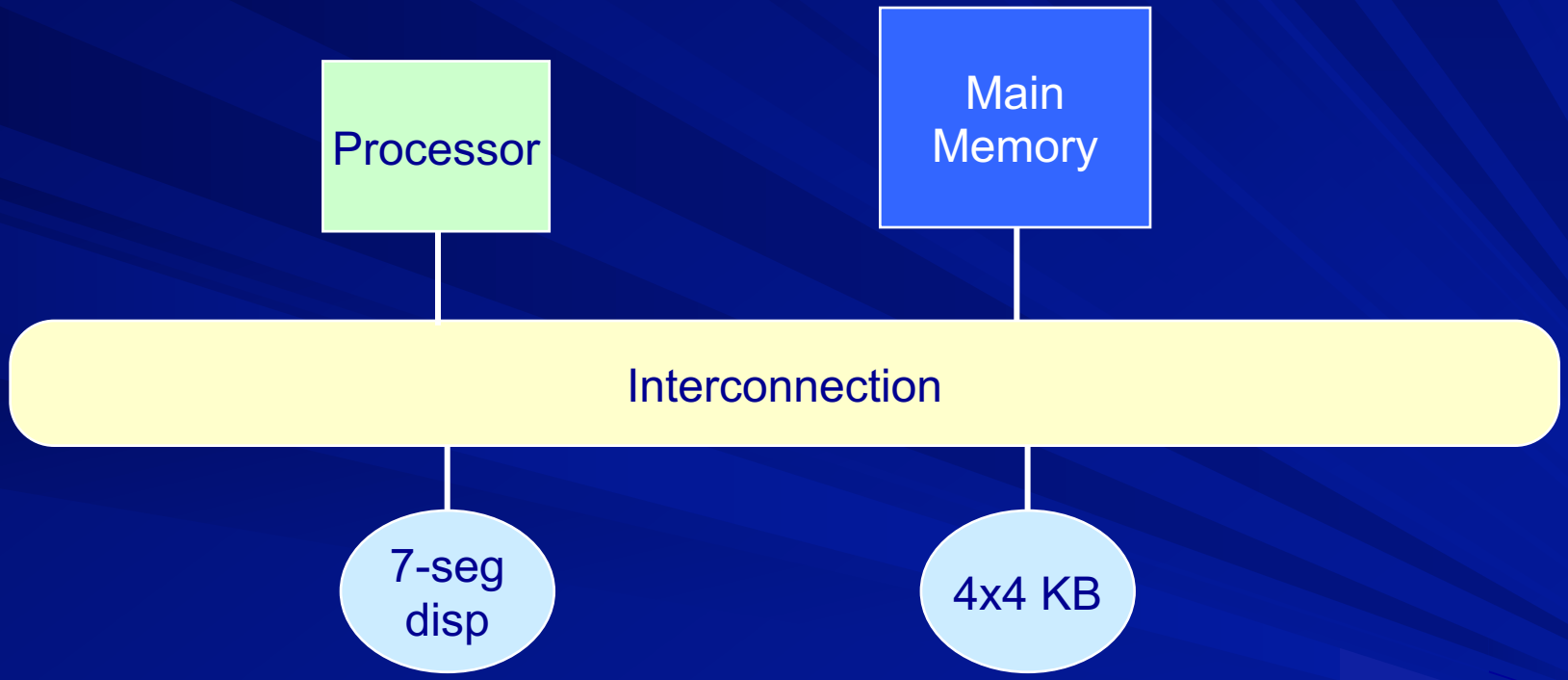
4 x 4 Keyboard Encoder



Block diagram

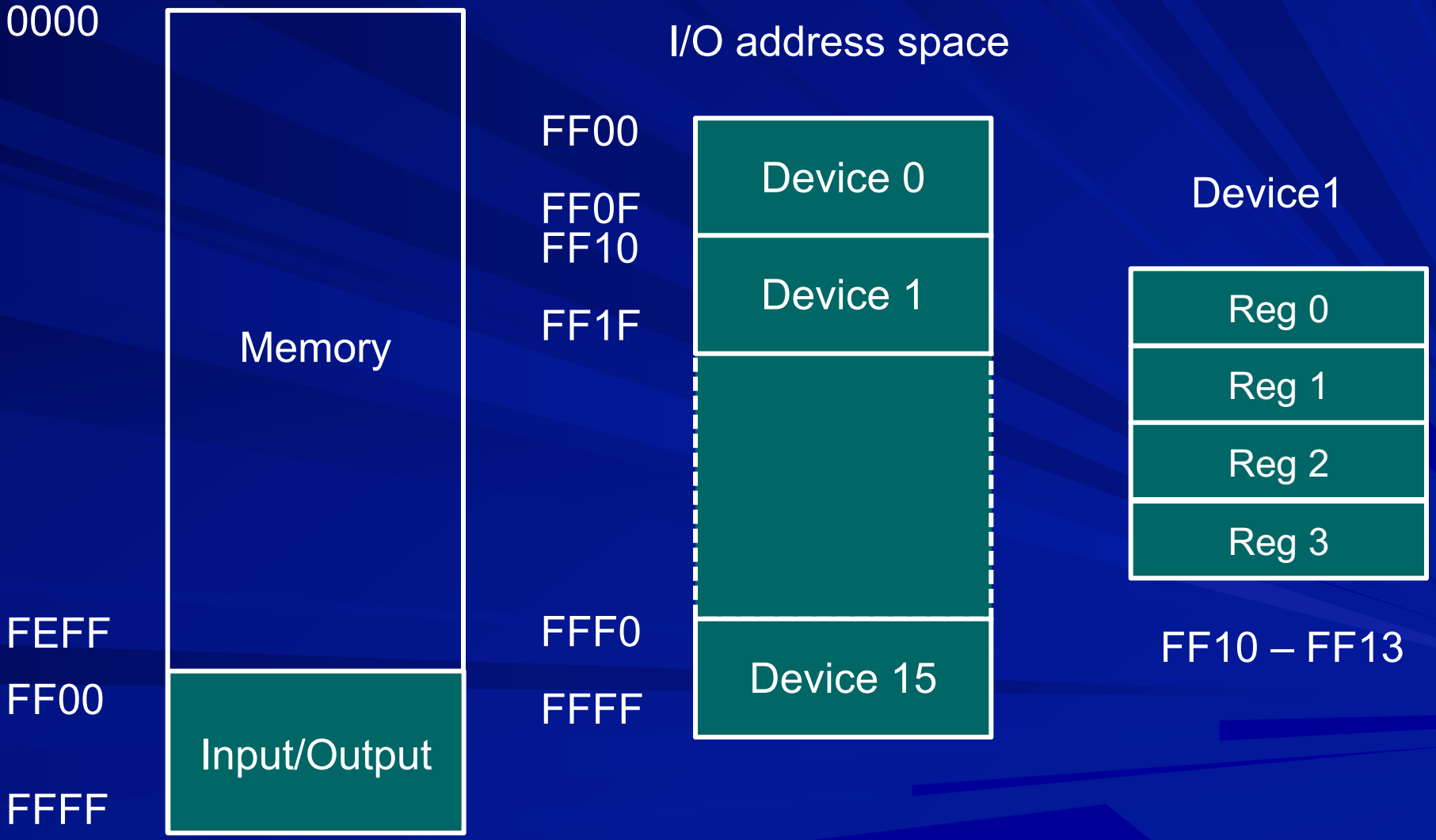


Block diagram

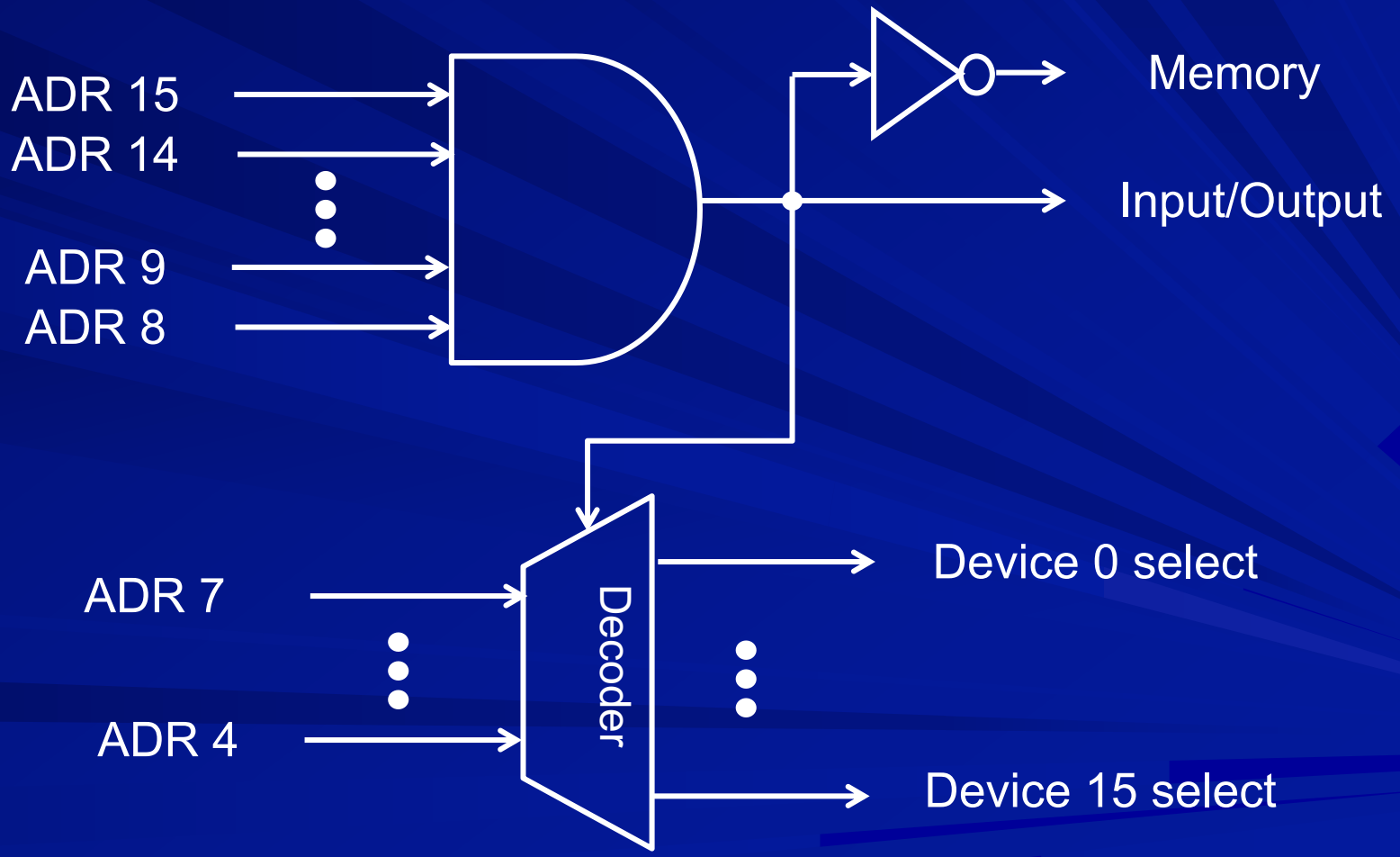


Low level control by software

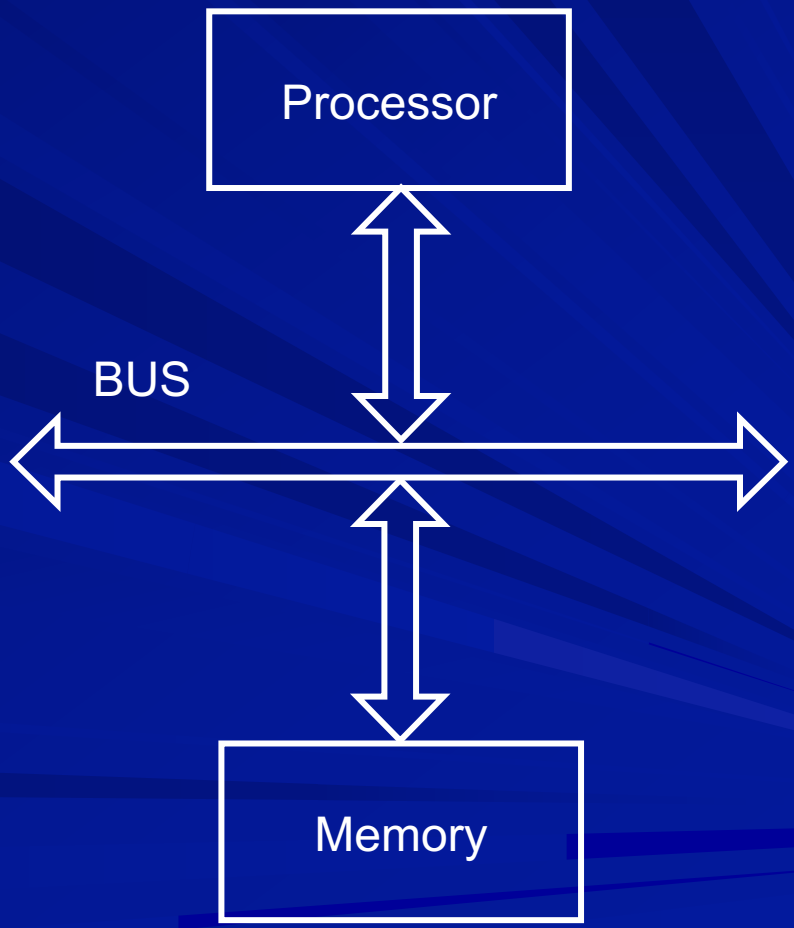
Address Map Example



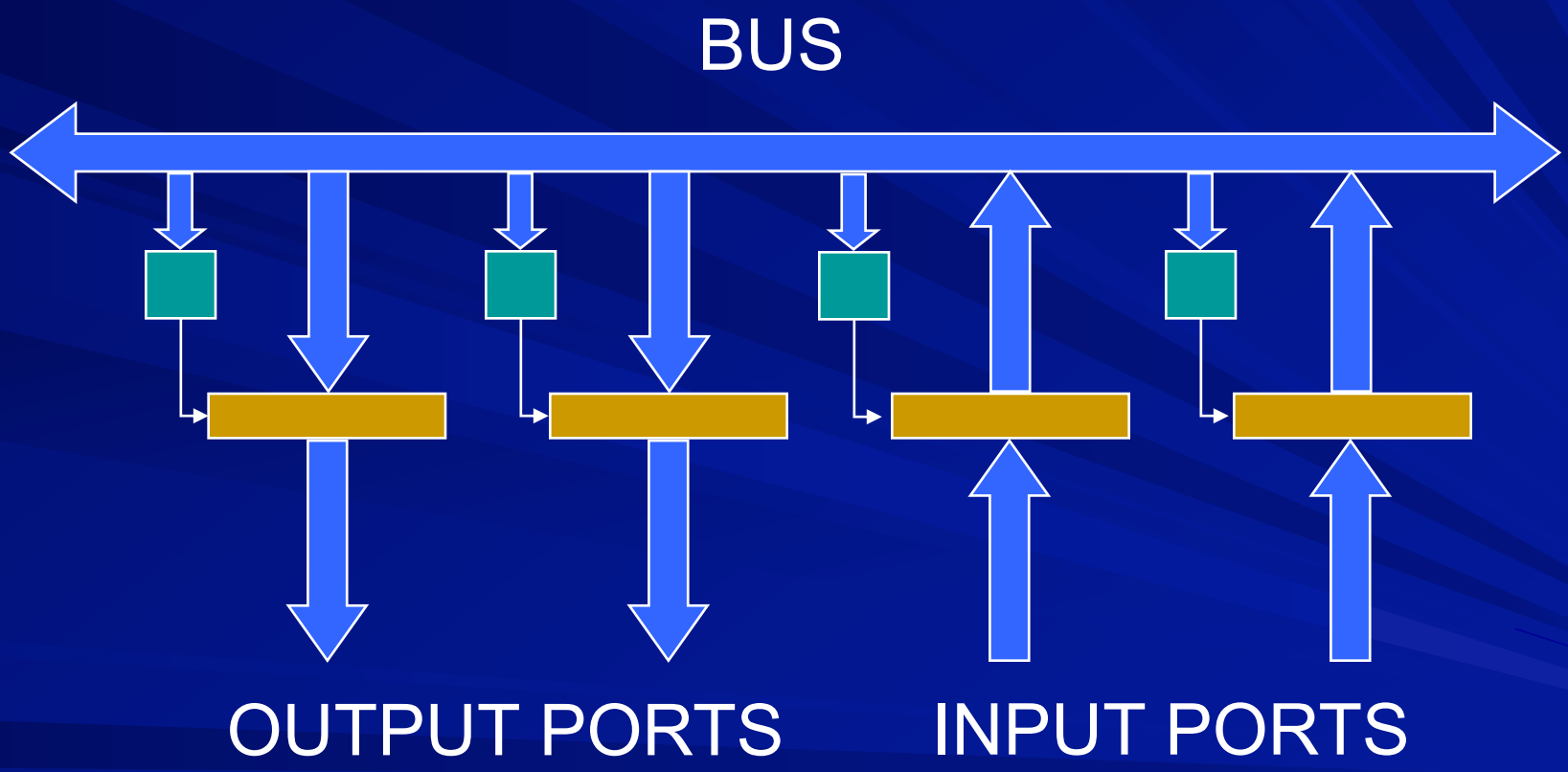
Decoding Address



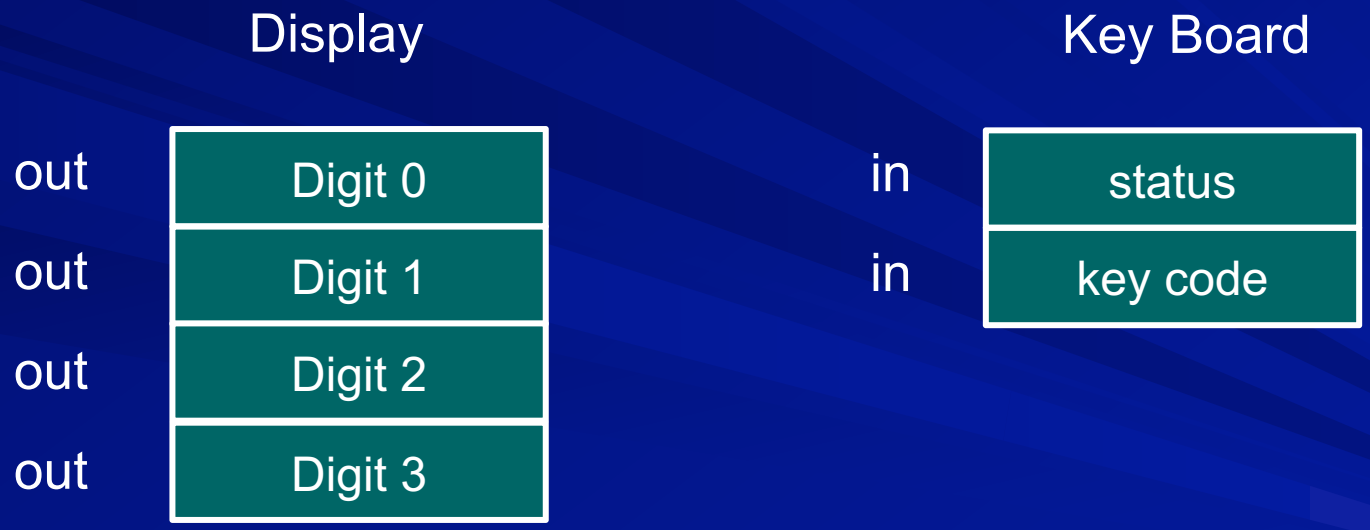
Interconnection using a bus



INPUT / OUTPUT PORTS



Ports for Disp, KB controllers



Ports for raw Disp, KB

refresh required

repeated scan required

Taking care of device timings

Devices may have latency of hundreds or thousands of cycles.

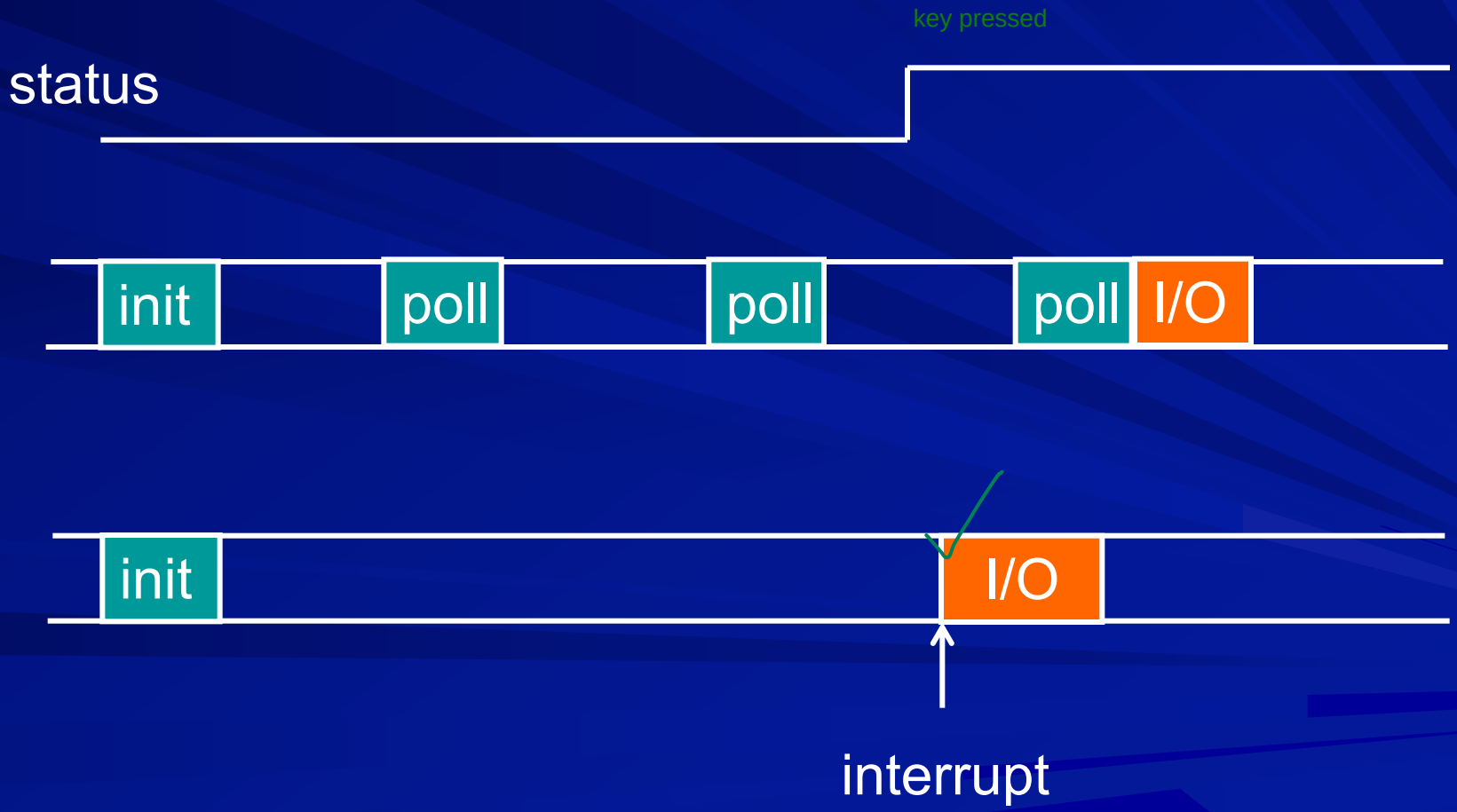
users are much slower than processor clock

Can the instruction execution be stretched for so long?

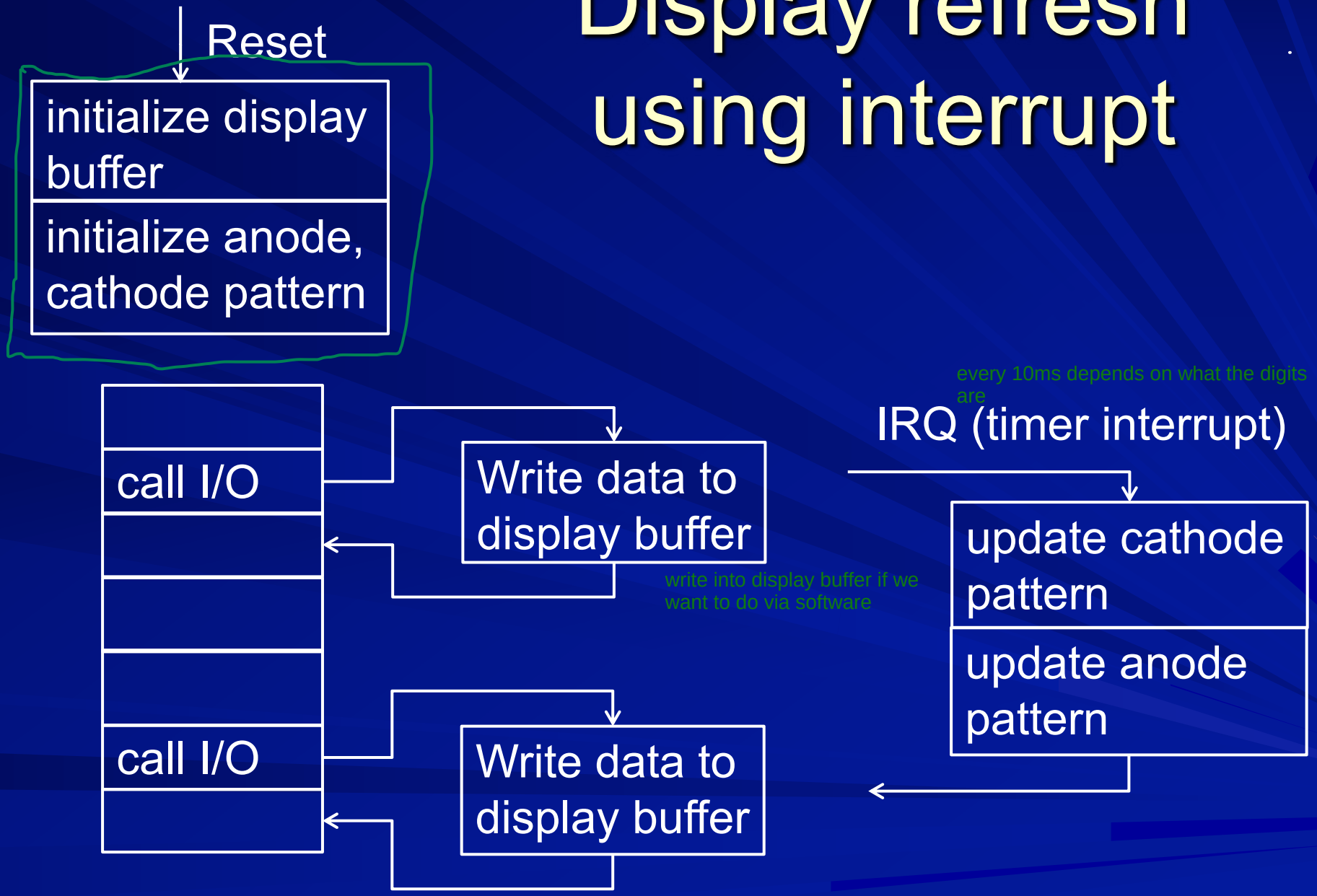
How CPU checks the device status

- Polling
  - periodically read the status register and figure out whether the device is ready or not
  - should be frequent enough so that no events are missed out
- Interrupt
  - keep busy with some other task and allow the device to intimate its readiness by sending a signal

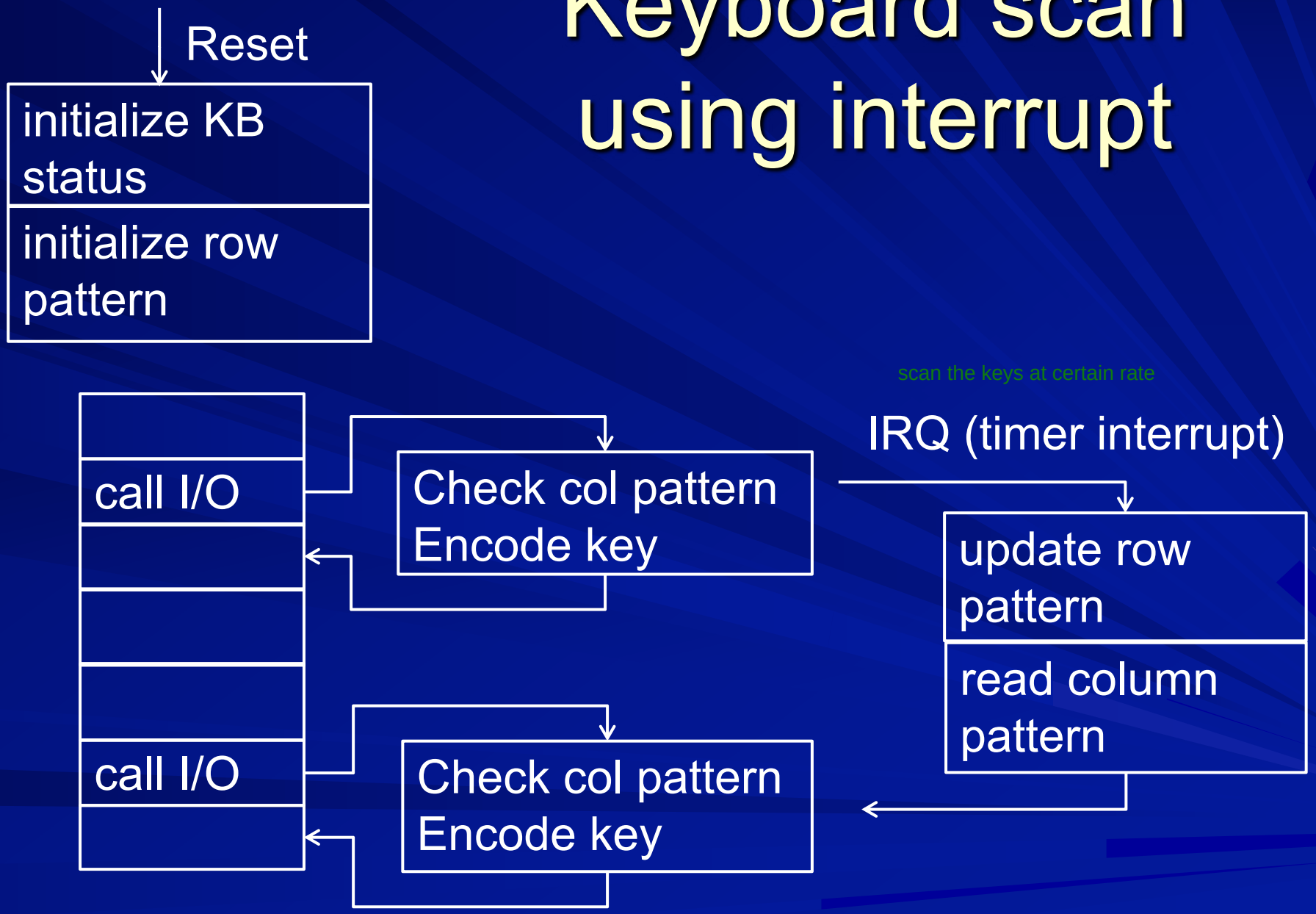
Polling vs interrupt



Display refresh using interrupt



Keyboard scan using interrupt



I/O mechanism

Direct memory access

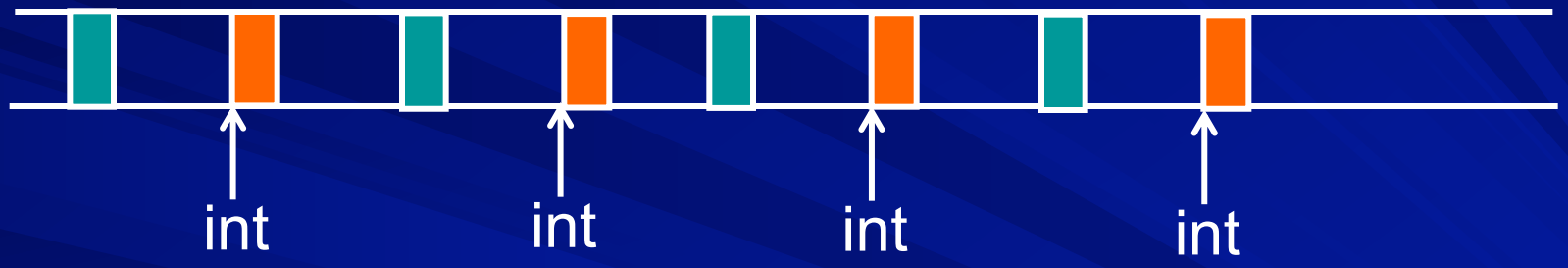
- Allow direct data transfer between the device and the memory
- The process is initiated by the processor
- When the entire job is done, the device informs the processor by an interrupt
- A specialized controller called DMA controller is used

transfer big chunk of data

processor has no role here, it is an independent controller to communicate between device and memory

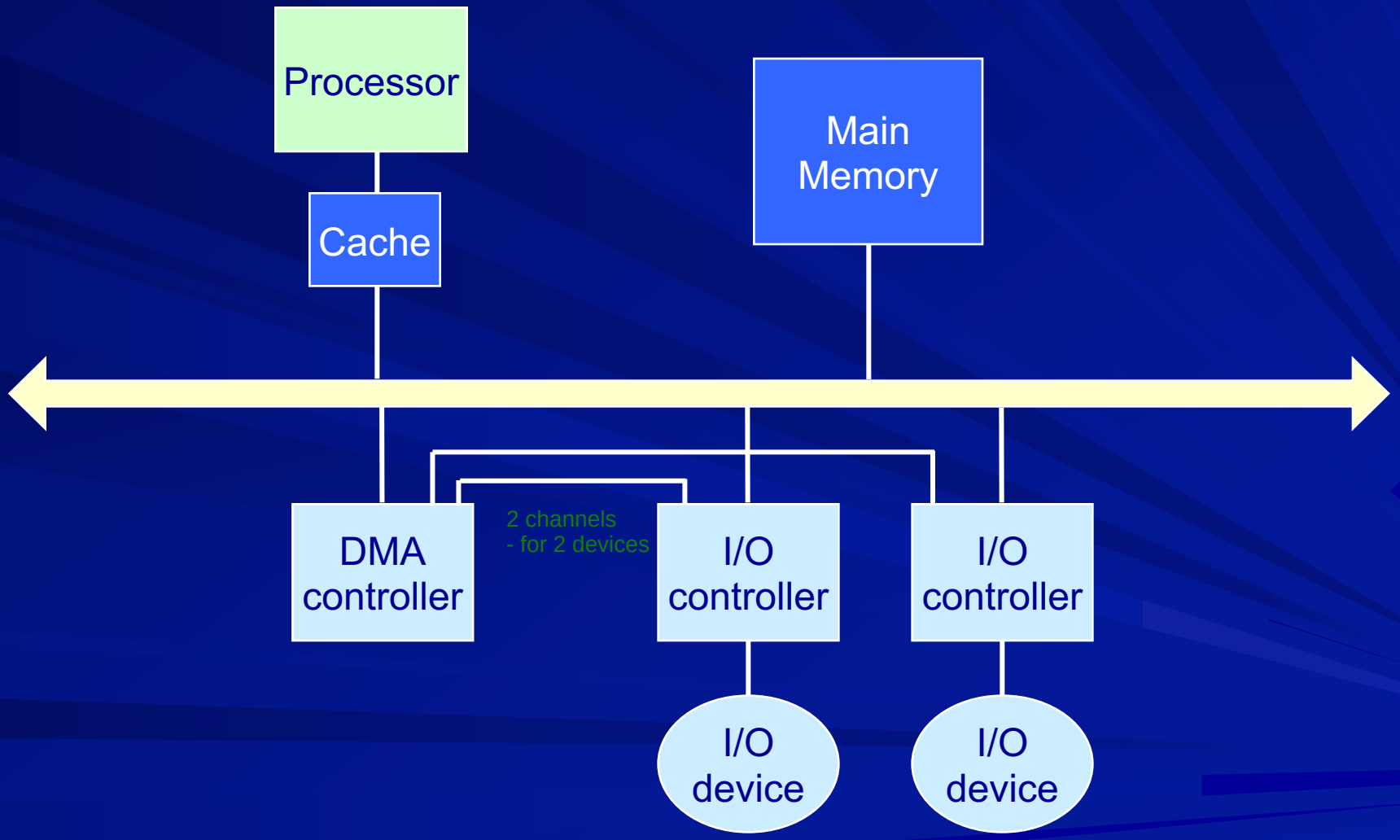
Benefit of DMA

Without DMA



With DMA

Direct memory access



COL216
Computer Architecture

Input/Output – 3

21st March 2022

Interrupts/Exceptions

What are exceptions

- Unusual events/conditions
  - Source: internal or external
  - Synchronous or asynchronous
  - Intentional or unintentional
- Require change in flow of control
  - Mechanism to alter the control flow
  - Event identification and response to it
  - Halt or resume original control flow

Hardware and Software Interrupts

- Software interrupts
  - caused by specific instructions to get some system service
  - alternative terms : system calls, traps, exceptions
- Hardware interrupts are caused by events/conditions detected by hardware
  - intentional : e.g., I/O event
  - unintentional : e.g., hardware fault, power outage, arithmetic overflow

Terminology

Exceptions vs. Interrupts

- Use both the terms interchangeably
- MIPS, ARM: exception - internal as well as external, interrupt - external
- Intel 80x86: interrupt - internal as well as external
- PowerPC: exception - unusual event, interrupt - change in flow of control

Examples of exceptions

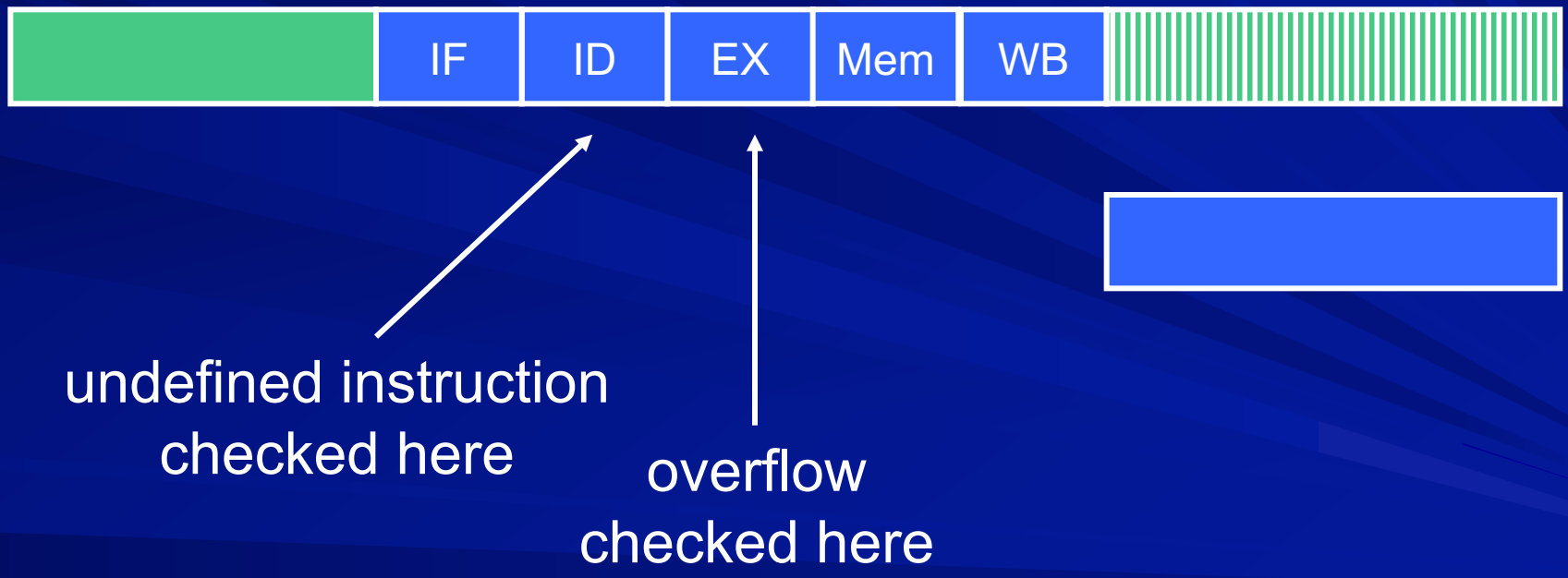
♦ synchronous ♦ asynchronous

	Intentional	Unintentional
Internal	Invoke OS function, Trace/debug	Access to privileged inst, Overflow/underflow, Undefined instruction, Hardware malfunction
External	I/O device request	Mem access exception, Alignment error, Timeouts, Power down, Hardware malfunction

Interrupt identification and response

- S/w interrupt identified during instruction decoding
- H/w interrupt identified by checking specific signals
- Response mechanism in both cases (h/w and s/w interrupts) is same
  - execution of some code : interrupt handler or interrupt service routine (ISR)
  - after serving interrupt, terminate or resume

Checking for exceptions



hardware interrupts can be checked any time

why can't we have a normal subroutine? Like when we face an exception, we transfer the control to some other flow so the part of the program where we transfer our function call to is actually an ISR

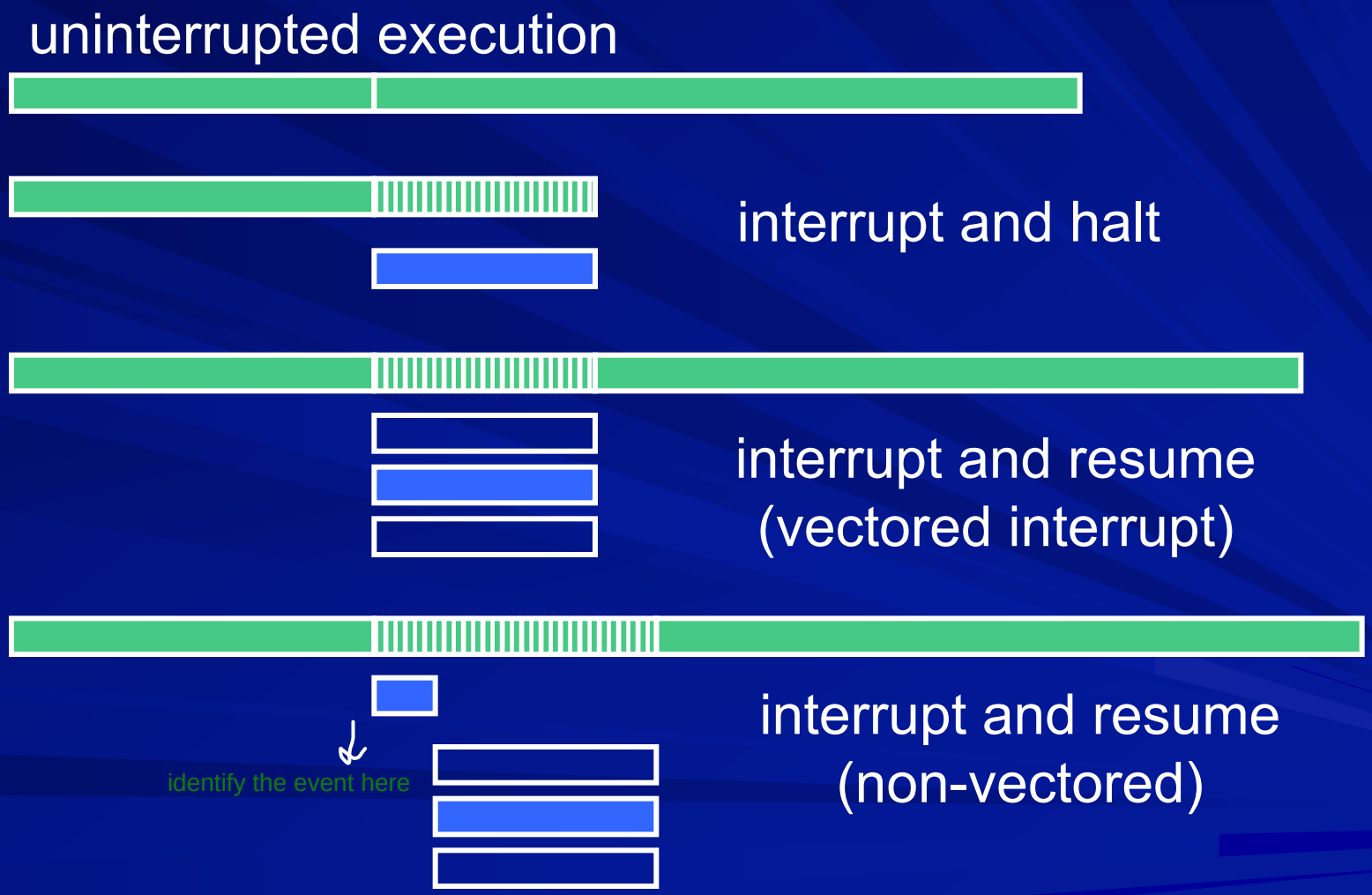
ISR vs normal subroutine

- Processors have two or more modes
  - normal mode / user mode
  - privileged mode / kernel mode / supervisor mode
- Application program executes in user mode

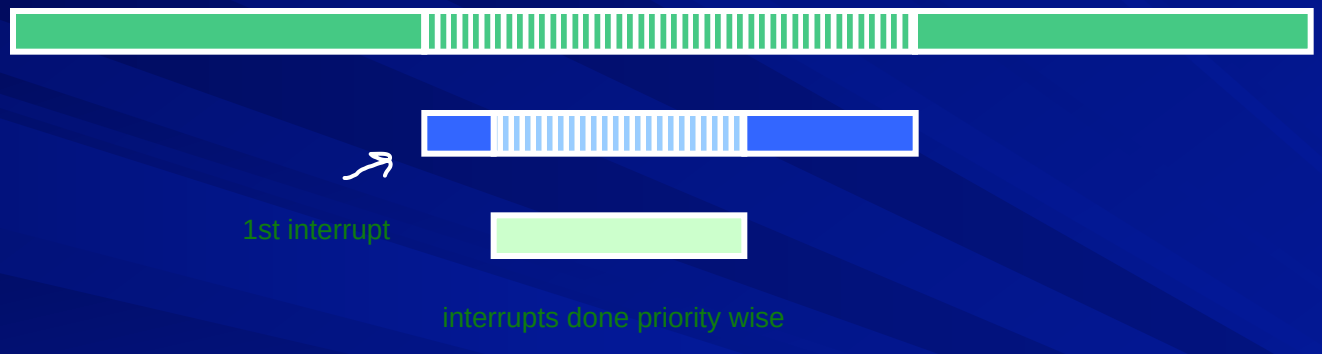
i/o is done in kernel mode
virtual memory is done in supervisor mode, OS sets up page table and it cannot be modified.

- To do certain privileged tasks, execution of some code in OS kernel is required
- ISR executes in privileged mode, provides controlled access to kernel functions

How exceptions are handled



Nested interrupts



Exceptions in ARM

give a message to the user and terminate the program

Exception type	Mode	Address of the ISR
Reset (pc = 0 and execution starts from address 0 again)	Supervisor	0x00000000
Undef instr	Undefined	0x00000004
S/W interrupt	Supervisor	0x00000008
Prefetch Abort	Abort	0x0000000C
Data Abort	Abort	0x00000010
Interrupt	IRQ	0x00000018
Fast interrupt	FIQ	0x0000001C

place a branch instruction here, that takes me to ISR, these are not directly accessible to users

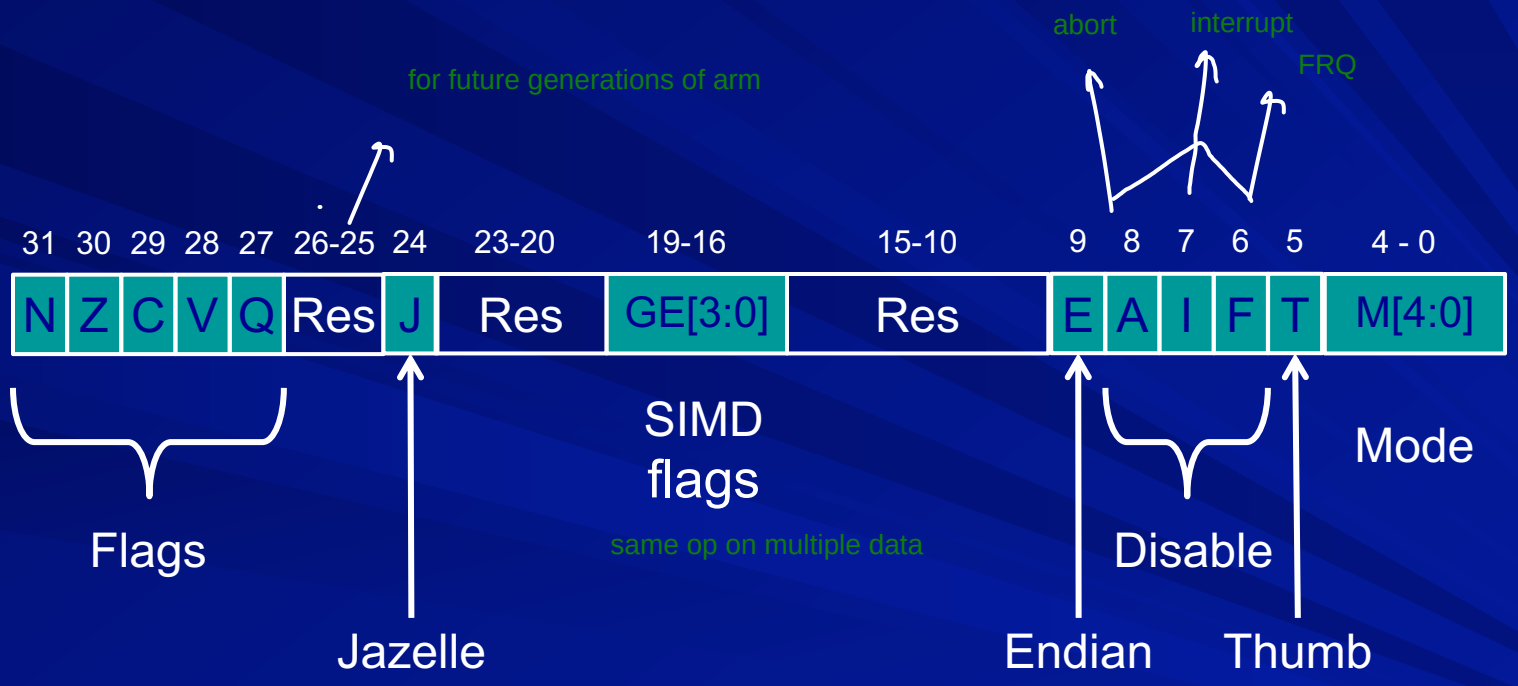
external

depending on type, the hardware transfers control to the address specified.

SWI software interrupt

- Control transferred to address 0x00000008
- Mode changes to 'supervisor' mode
- CPSR (Current Processor Status Register) is saved in $SPSR_{svc}$ (Saved Processor Status Register)
- PC is saved in $R14_{svc}$
- Comment field may specify a particular service/function

Processor status register



ARM register set

Mode	Registers						
			SP	LR	PC	17th register	
User:	R0-R7	R8-R12	R13	R14	R15	CPSR	
System:							
Supervisor:			R13	$R14_{svc}$	-> in supervisor mode		SPSR
Abort:			R13	R14			SPSR
Undefined:			R13	R14			SPSR
Interrupt:			R13	R14			SPSR
Fast int:		R8-R12	R13	R14			SPSR

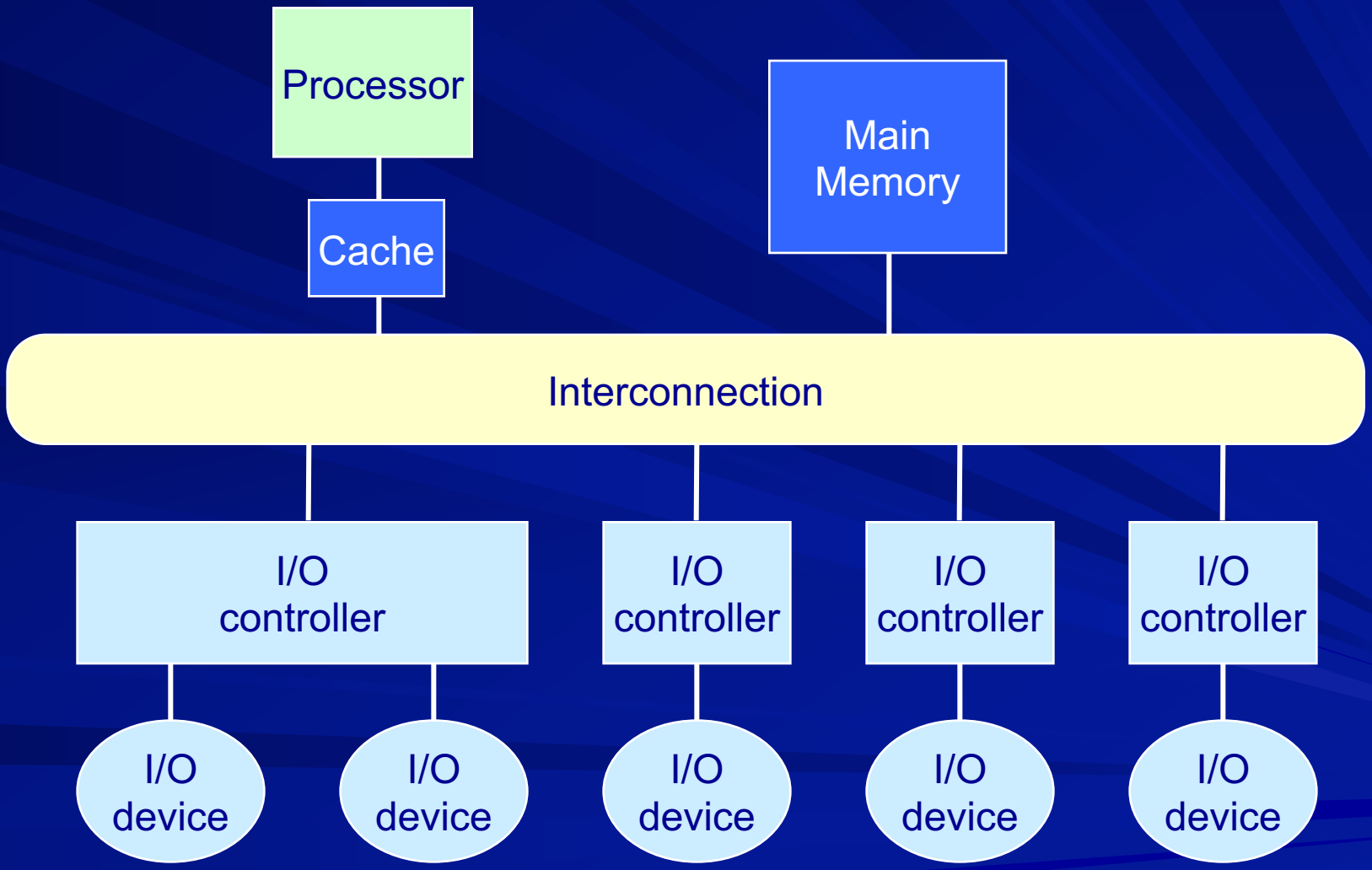
Total 37

Exception handling in pipelined design

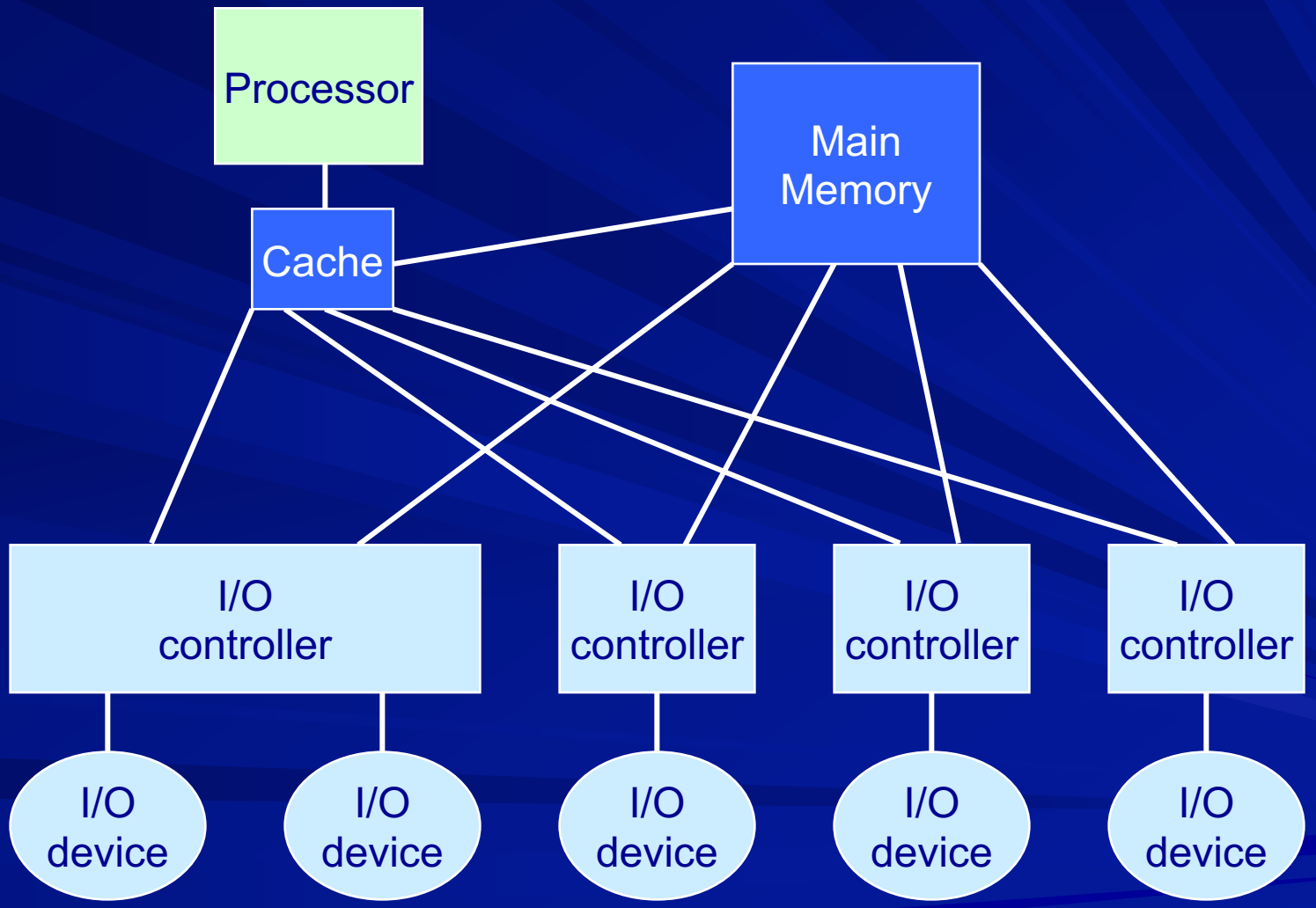
- Difficulty in status saving
  - multiple instructions under execution
- Possibility of
  - multiple exceptions in same cycle
  - change in order of exceptions
- Handling approaches
  - precise
  - imprecise

Interconnecting Subsystems

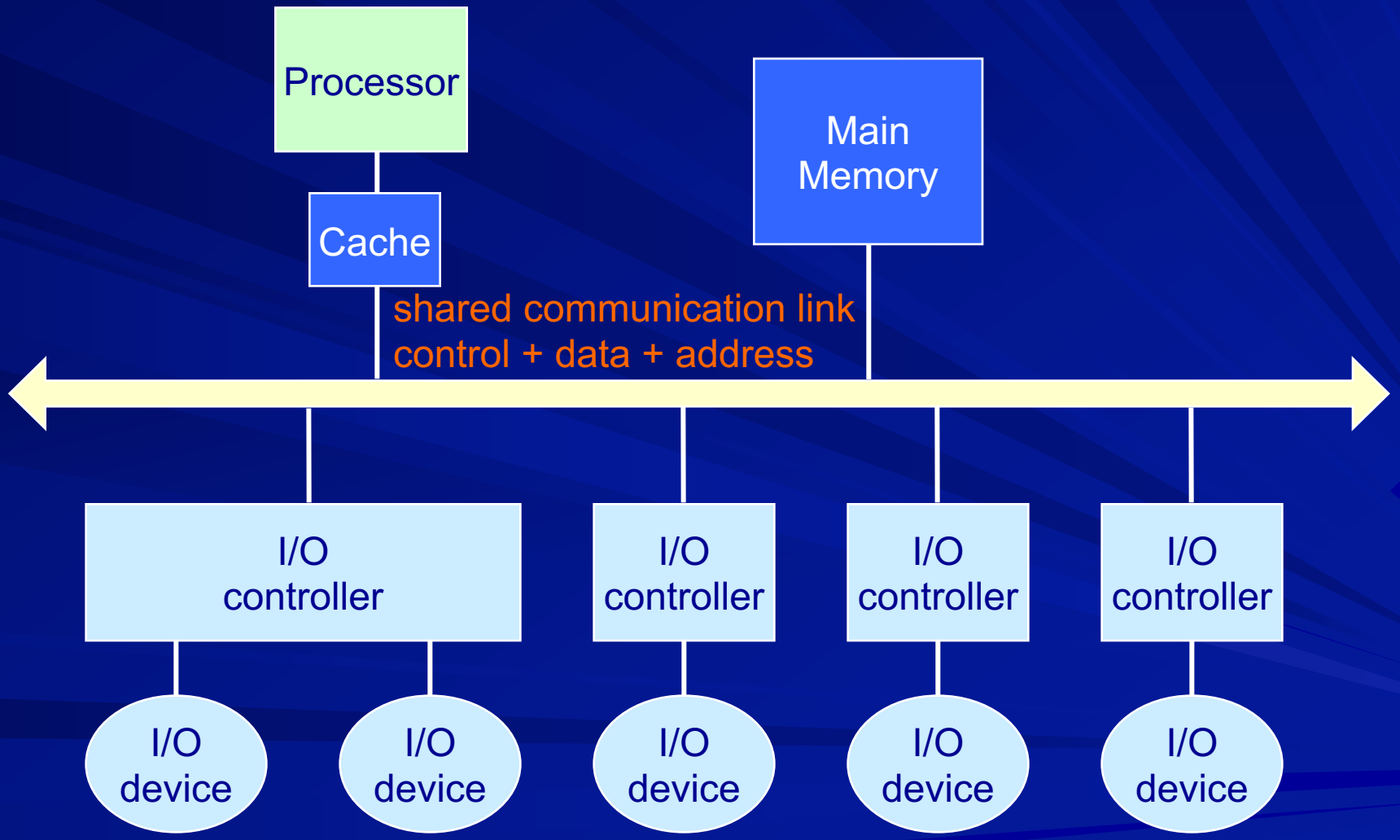
Interconnecting Subsystems



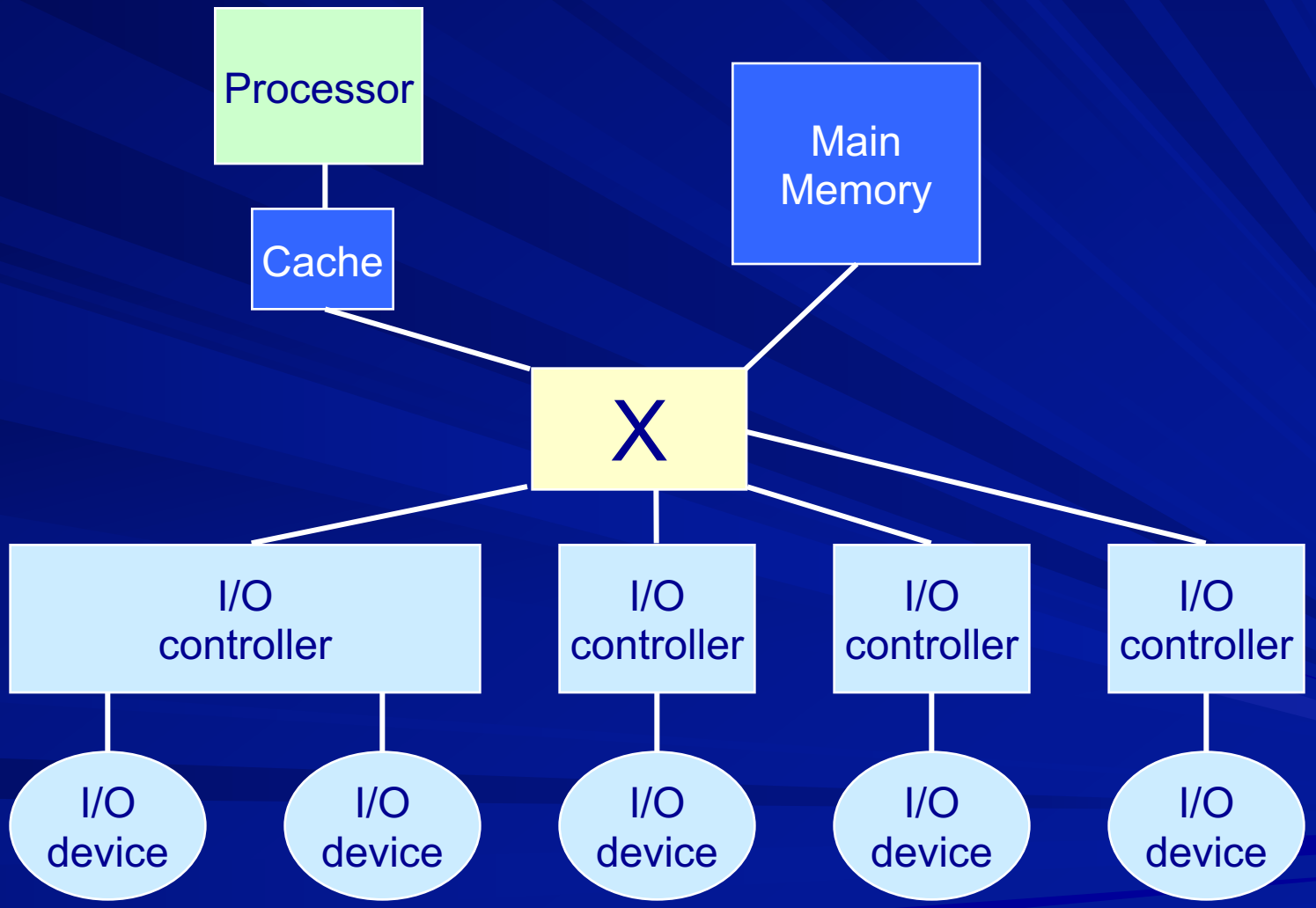
Point to point connections



Single bus connecting all



Cross bar switch



Comparison

	P to P	Bus	X-bar
Cost	high	low	high
Throughput	high	low	high
Ports	multi	single	single
Expansion	difficult	easy	difficult

only one at a particular time

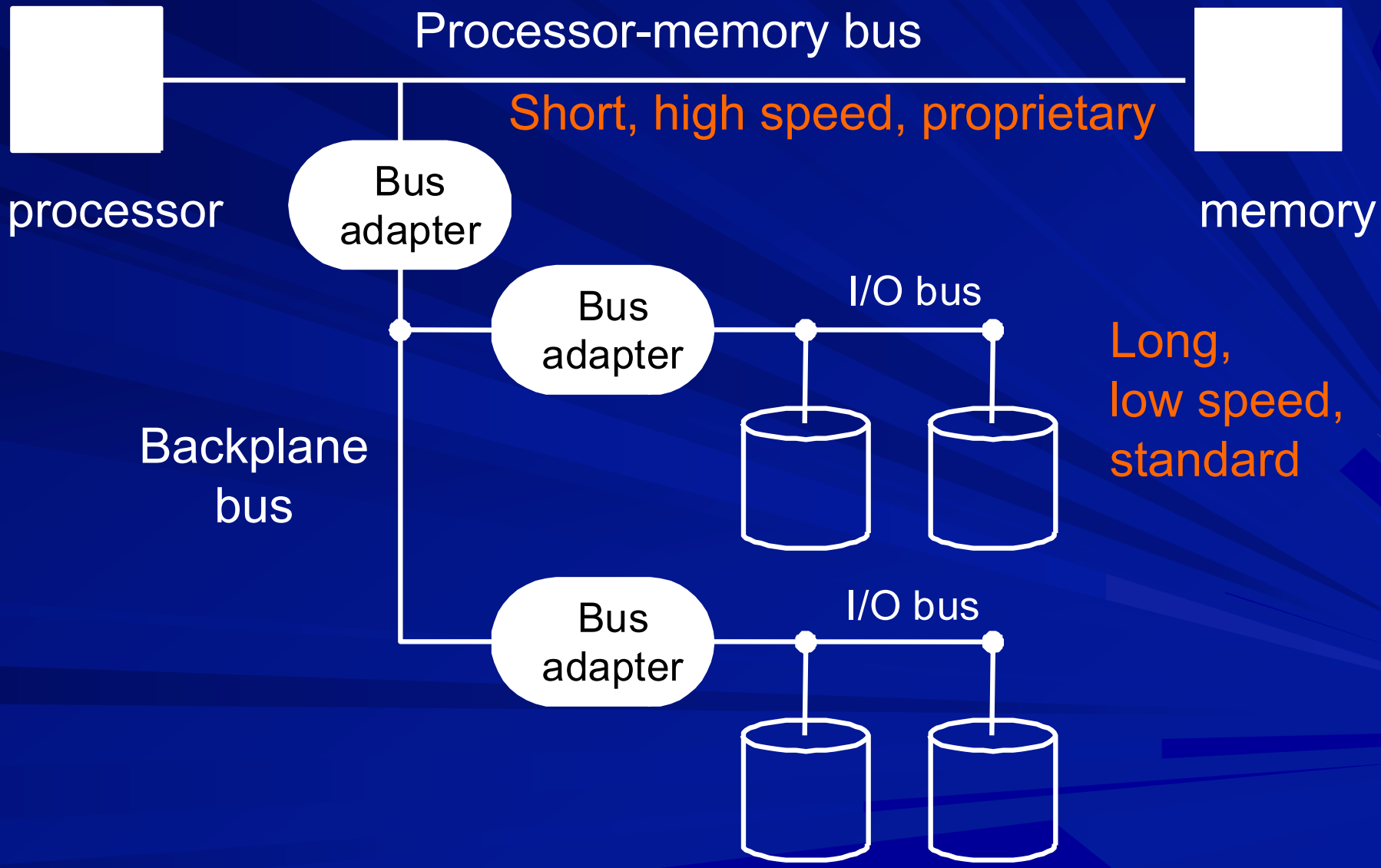
Multi-buses: More performance, more cost

Multi-switches: Lower cost, same performance

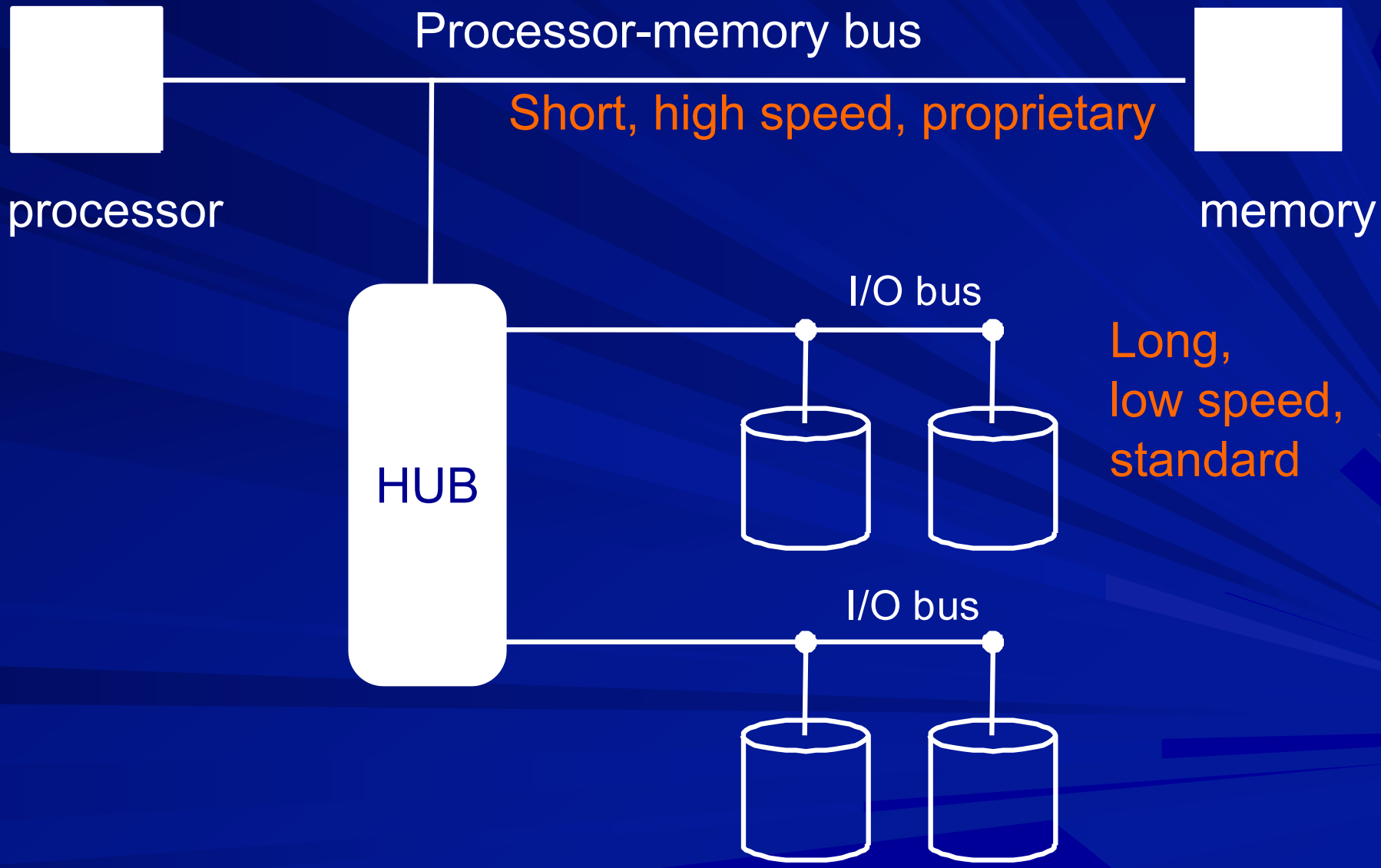
cross-bar

Backplane bus

System with multiple buses



System with multiple buses



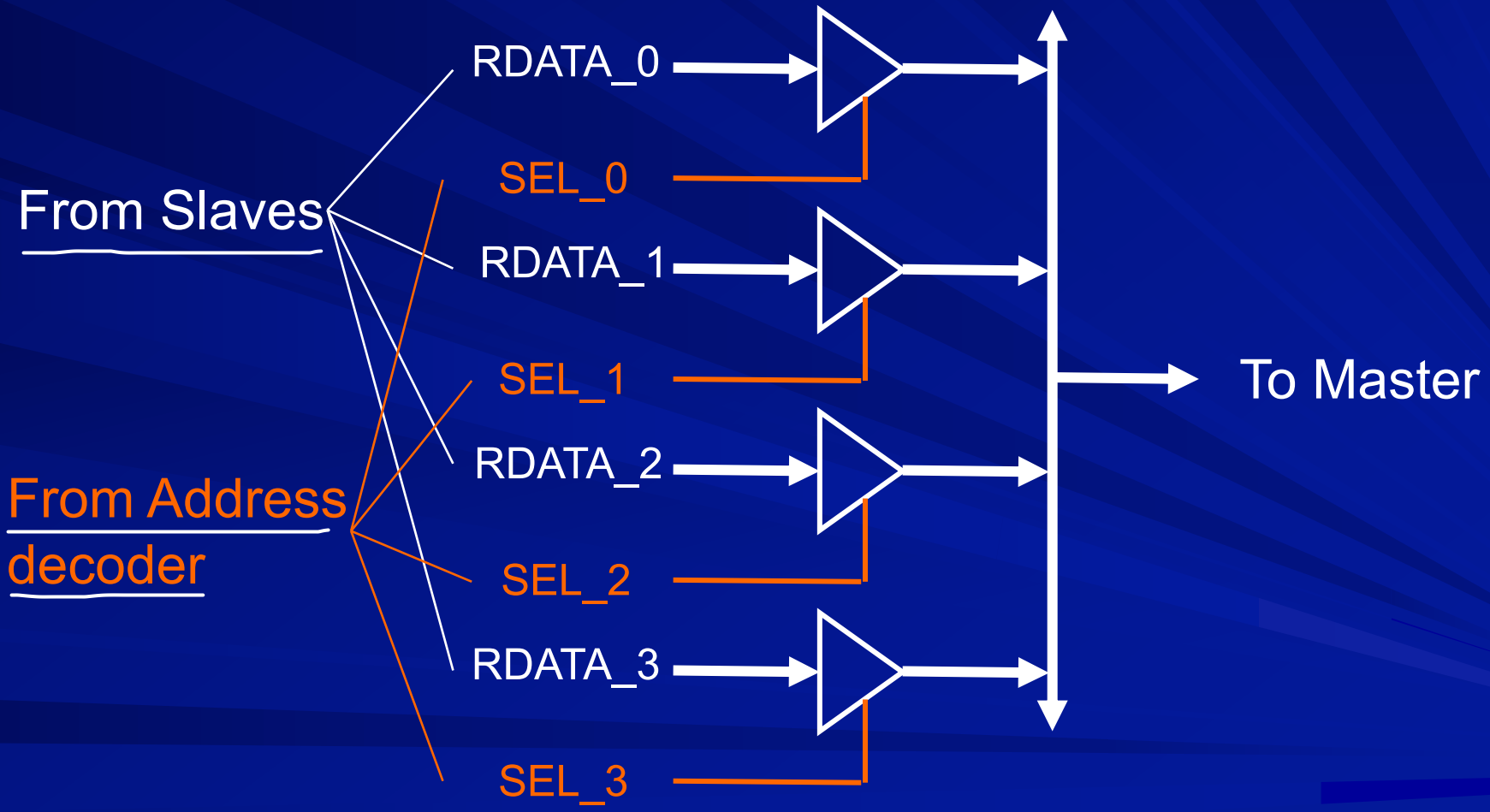
Bus Standards

- Physical / mechanical
  - Pins, connectors, cables
- Electrical
  - Voltage / current levels, impedances
- Logical
  - Definition of signals
  - Timings and protocols

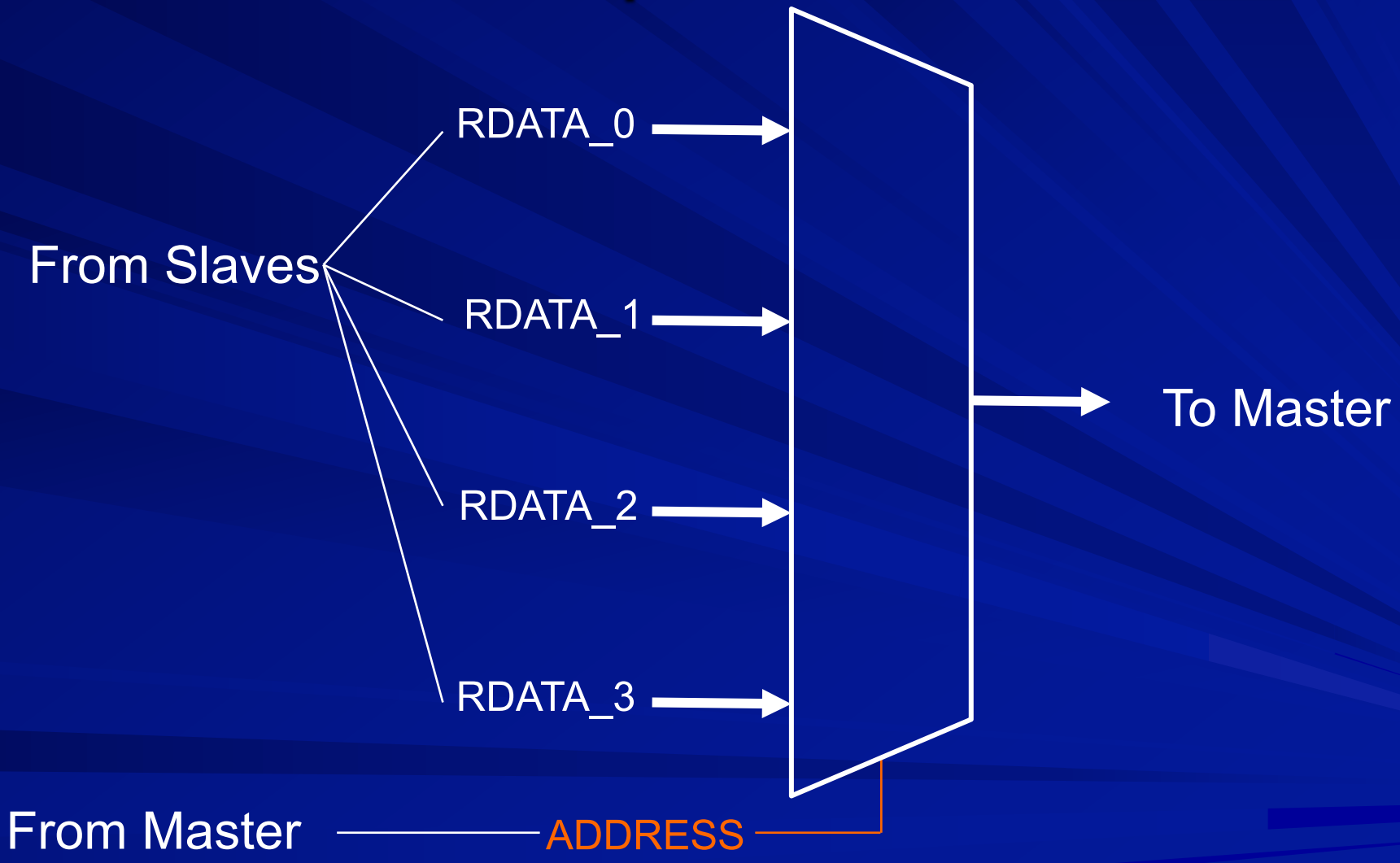
Subsystems on the buses

- Master
  - Initiates transfer
- Slave
  - Responds to master
- Bridge / bus adapter
  - Connects two buses
- Hub
  - Connects many buses
- Arbiter
  - Resolves master requests
- Decoder
  - Selects slaves

Tri-state Bus



Multiplexed Bus



AMBA : Advanced Microcontroller Bus Architecture

- ARM open standard for on-chip buses
- For System-on-Chip (SoC)
- Variants
  - AHB : Advanced High speed Bus
  - APB : Advanced Peripheral Bus
  - AXI : Advanced eXtensible Interface
  - AHB-Lite : subset of AHB

AHB-Lite

- Single master, multiple slaves
- Parallel
- Multiplexed (not tri-state)
- Separate data, address, control
- Synchronous
- Pipelined
- Burst transfer supported
- Split transaction not supported

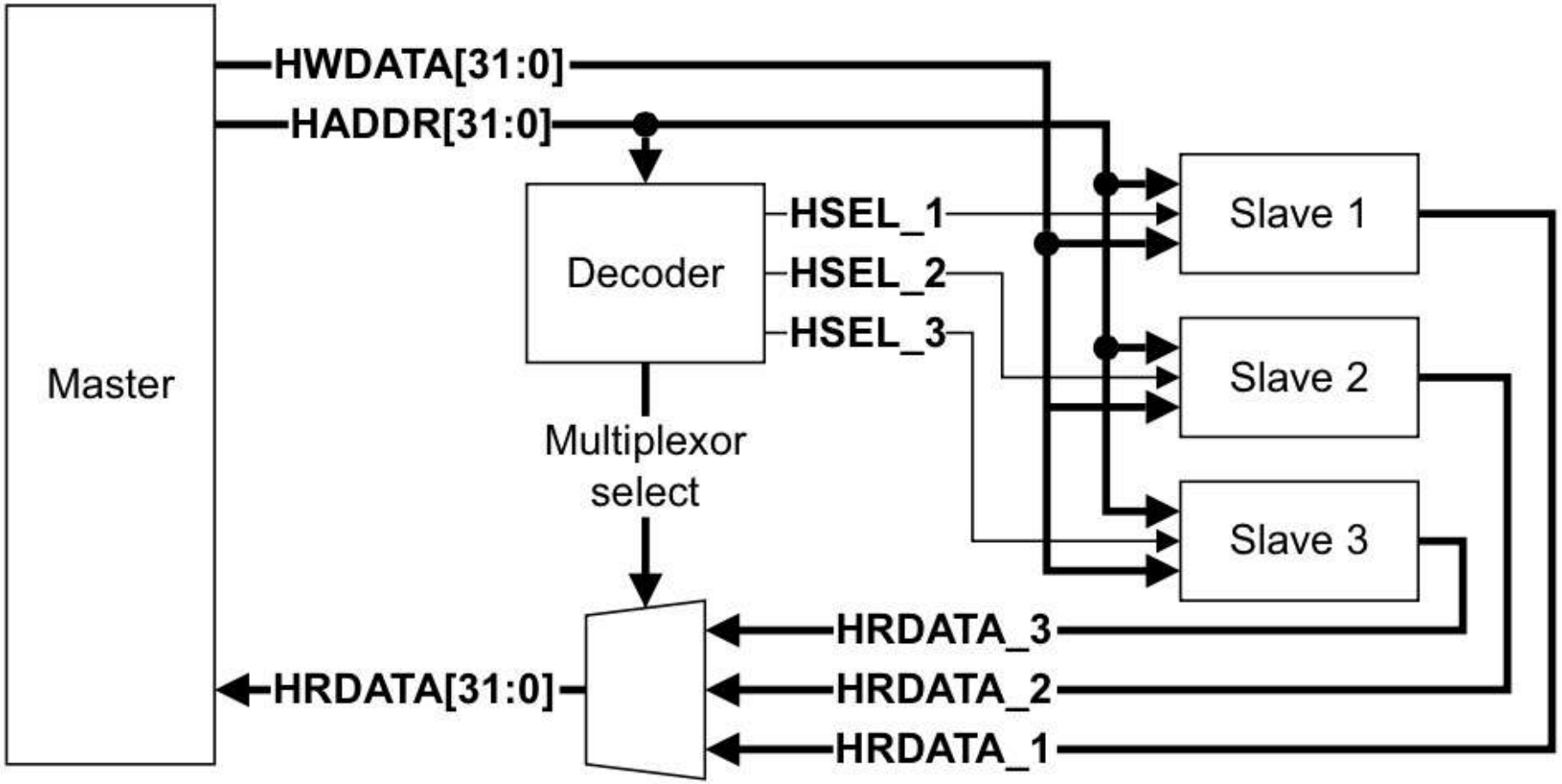
usb is serial bus- one bit at a time

all transitions happen synchronised with the clock

each transaction involves burst of data, large data

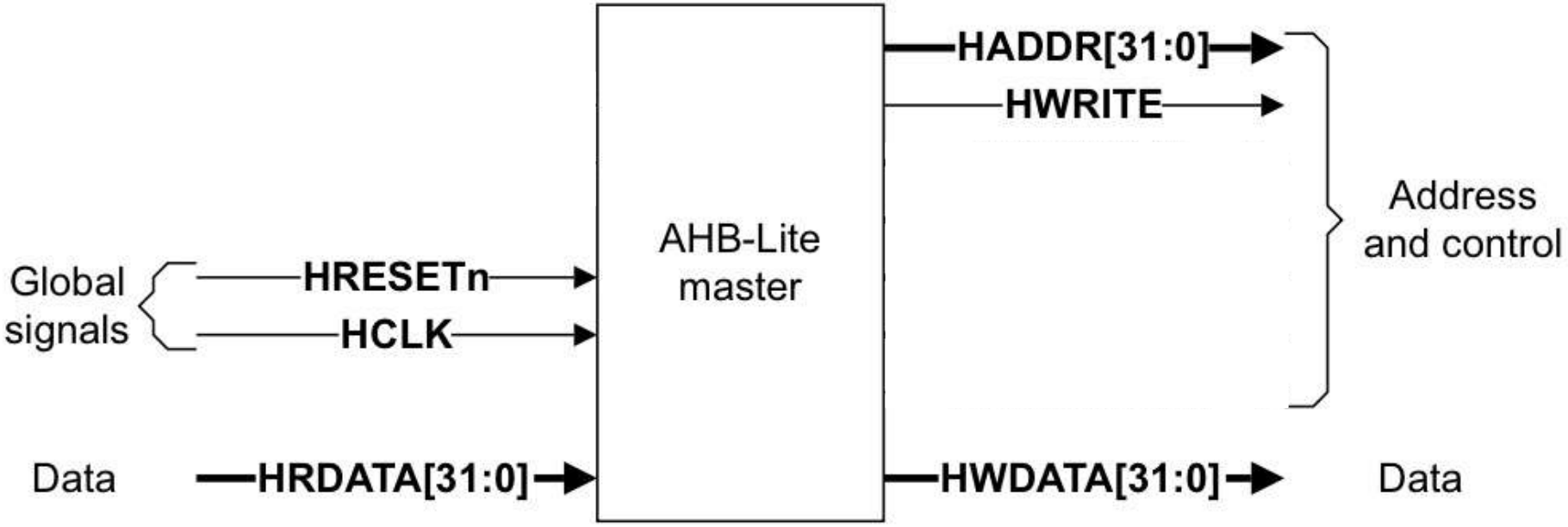
transaction between initial and final transaction, in between we could not start another transaction

AHB-Lite Block Diagram



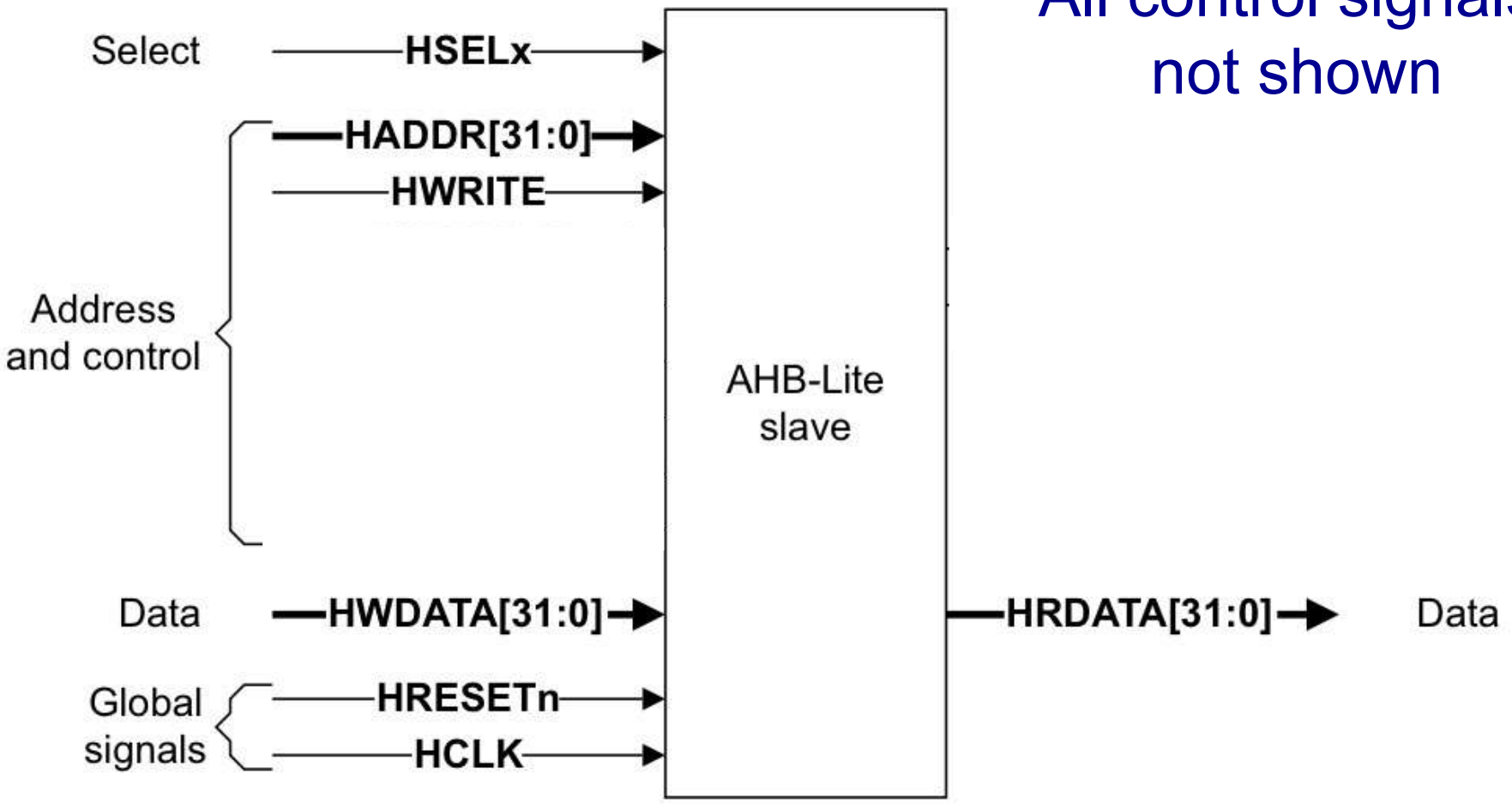
DMA controller is a master

Master Interface



All control signals not shown

Slave Interface



All control signals not shown

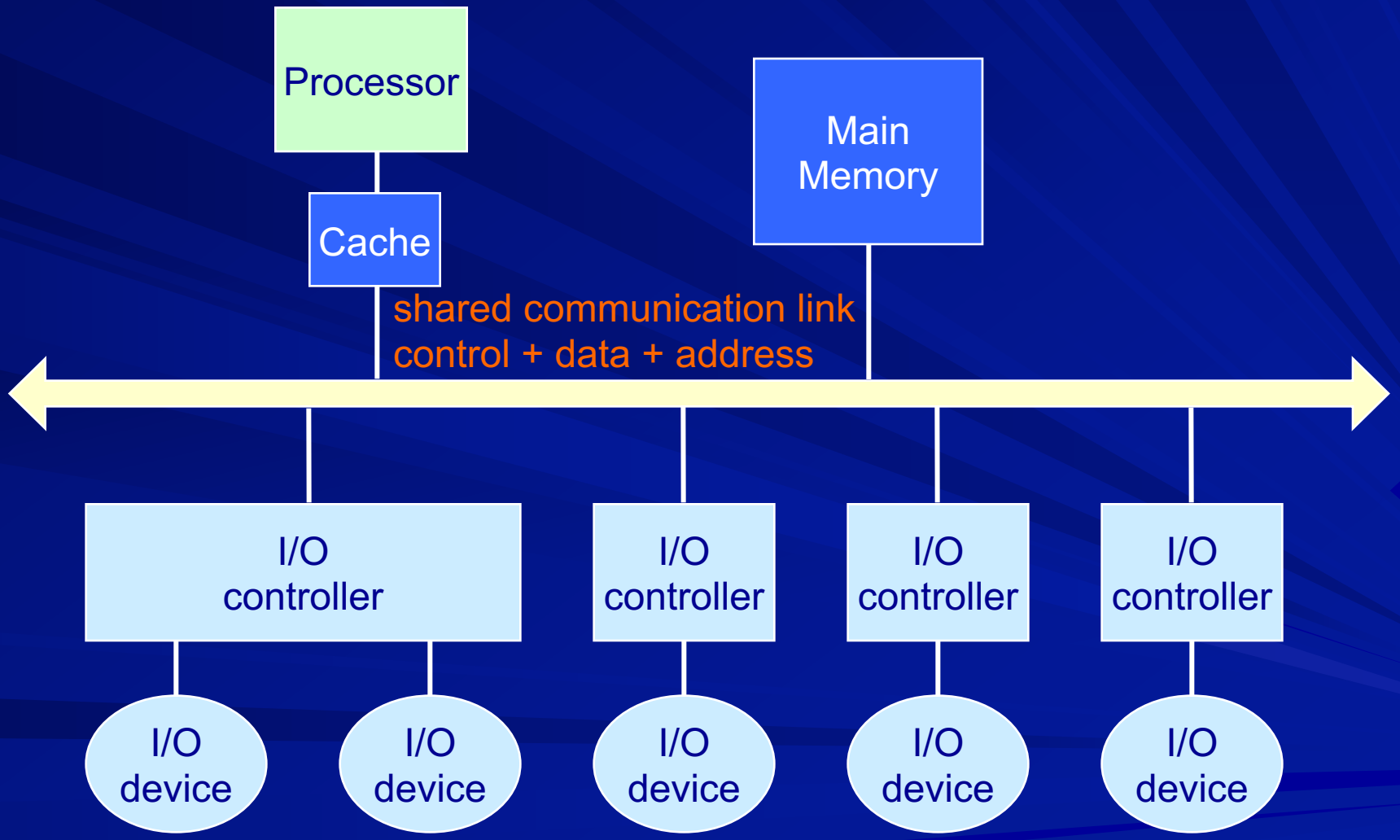
THANKS

COL216
Computer Architecture

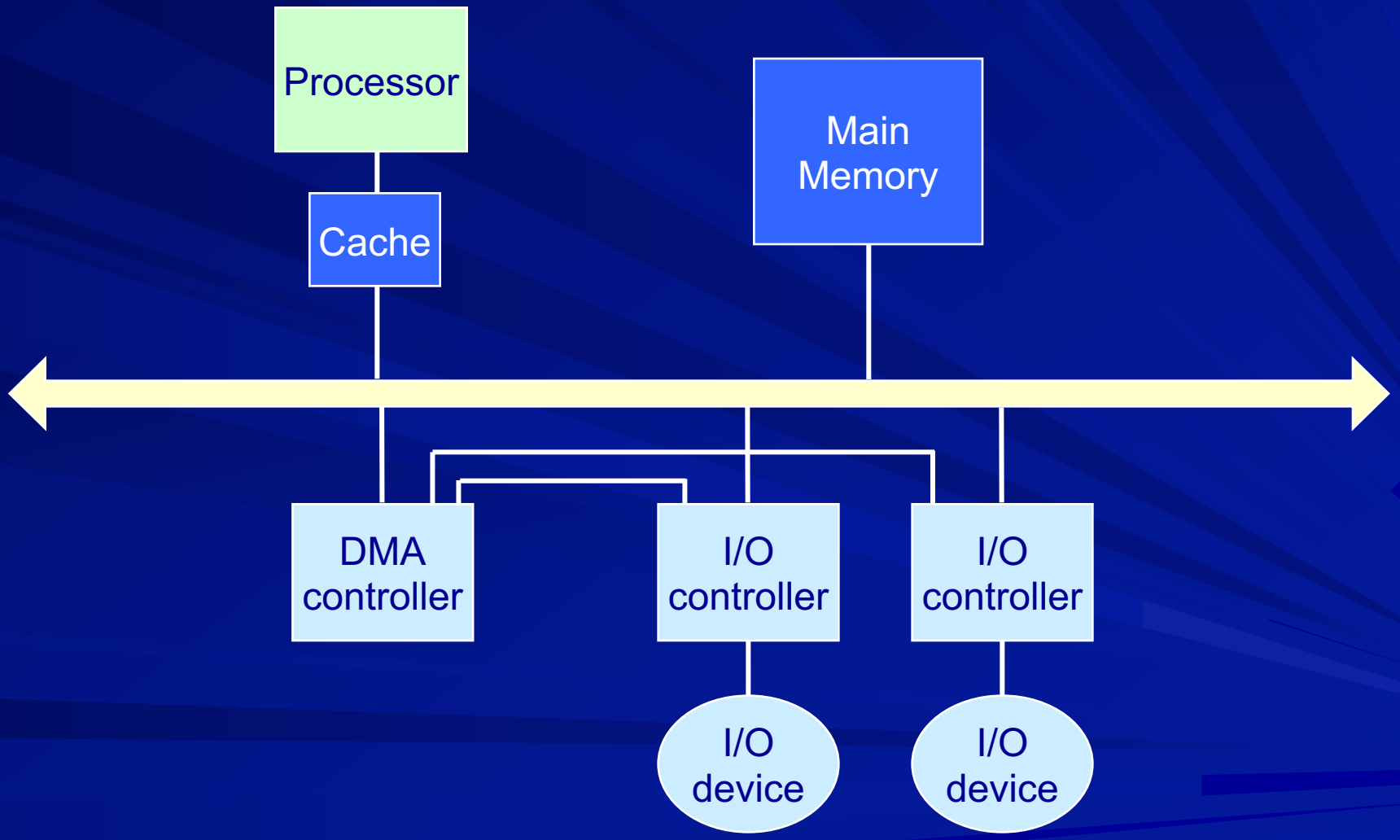
Input/Output – 4

24th March 2022

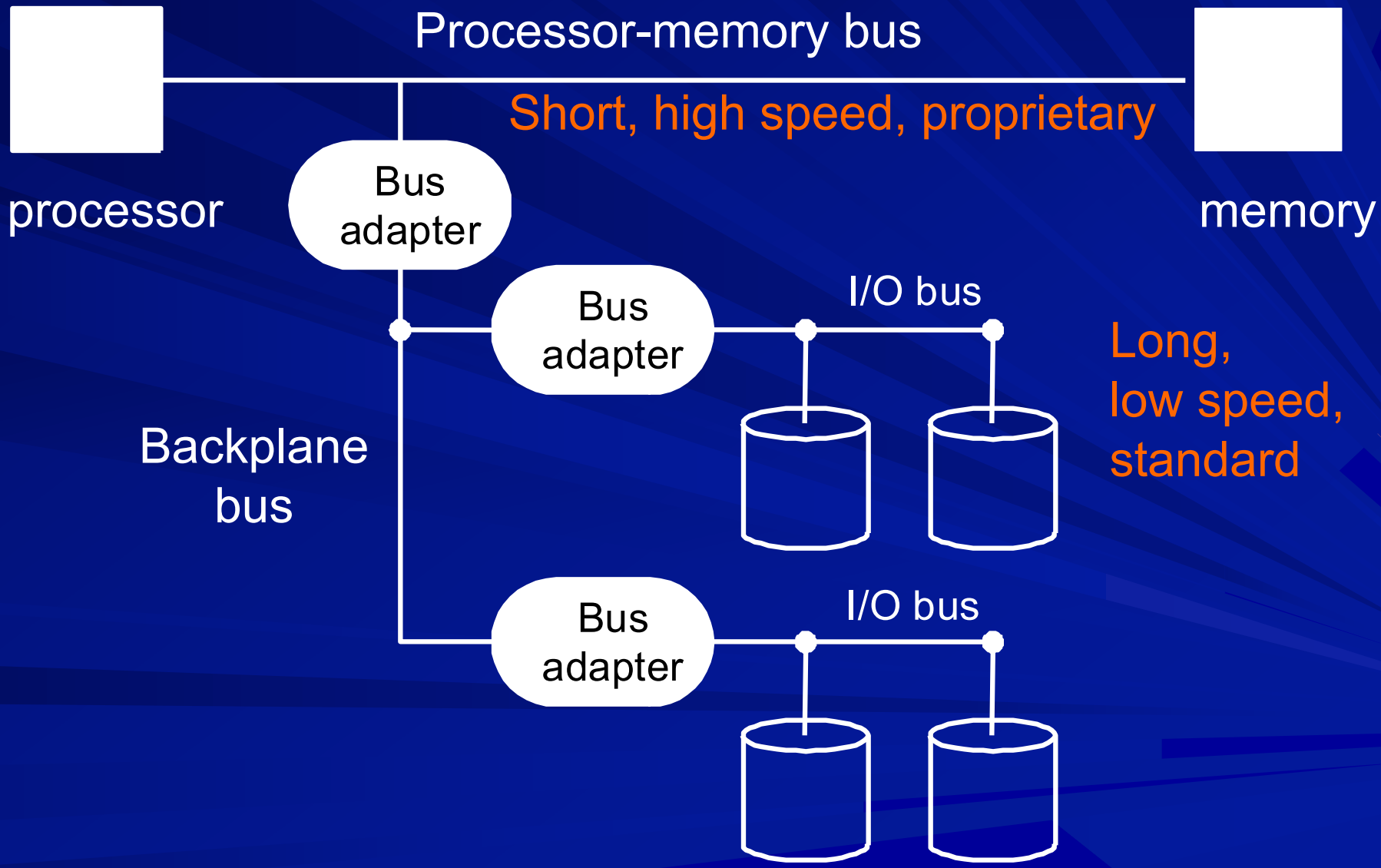
Single bus connecting all



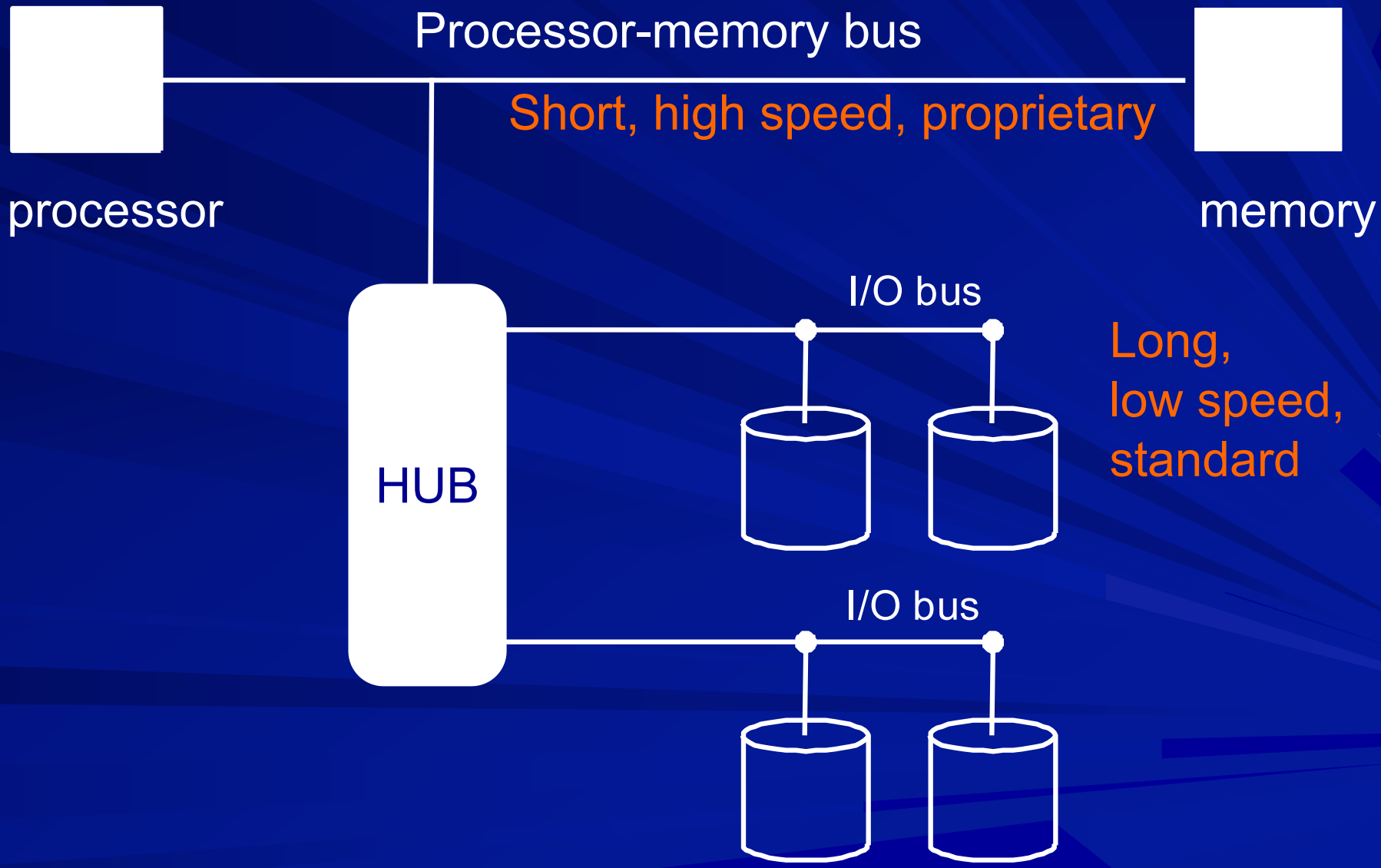
Direct memory access



System with multiple buses



System with multiple buses



Bus Standards

- Physical / mechanical
  - Pins, connectors, cables
- Electrical
  - Voltage / current levels, impedances
- Logical
  - Definition of signals
  - Timings and protocols

Standard buses/ports/interfaces

- Standardization required for subsystems from multiple sources to work together
- Technological developments lead to continuous improvement of specs, standardization implies freezing the specs
- Standards need to be revised and improved periodically
- Standardization is done by consortium of several organizations or professional bodies

AMBA : Advanced Microcontroller Bus Architecture

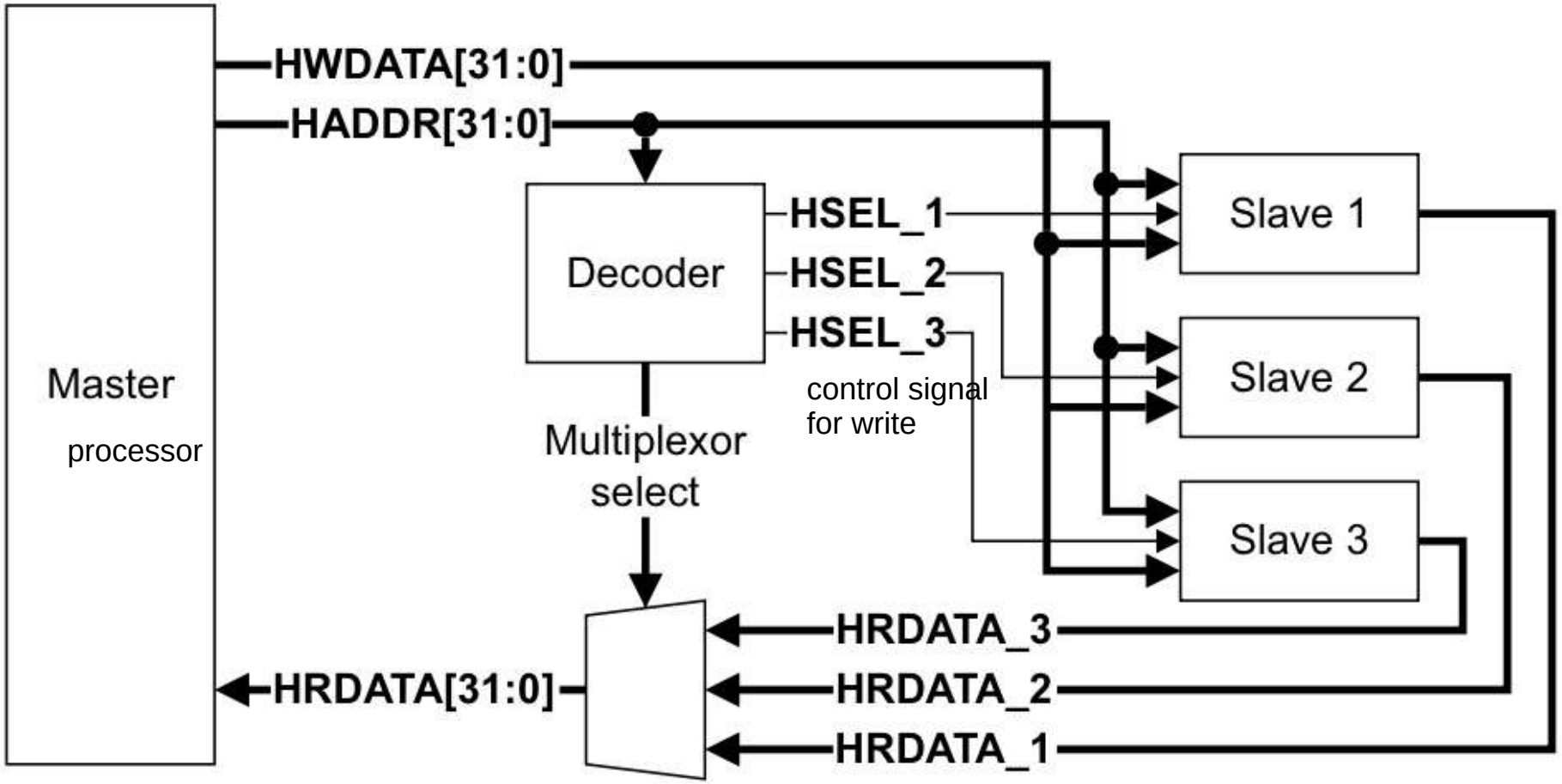
for ARM

- ARM open standard for on-chip buses
- For System-on-Chip (SoC)
- Variants
  - AHB : Advanced High speed Bus
  - APB : Advanced Peripheral Bus
  - AXI : Advanced eXtensible Interface
  - AHB-Lite : subset of AHB

AHB-Lite

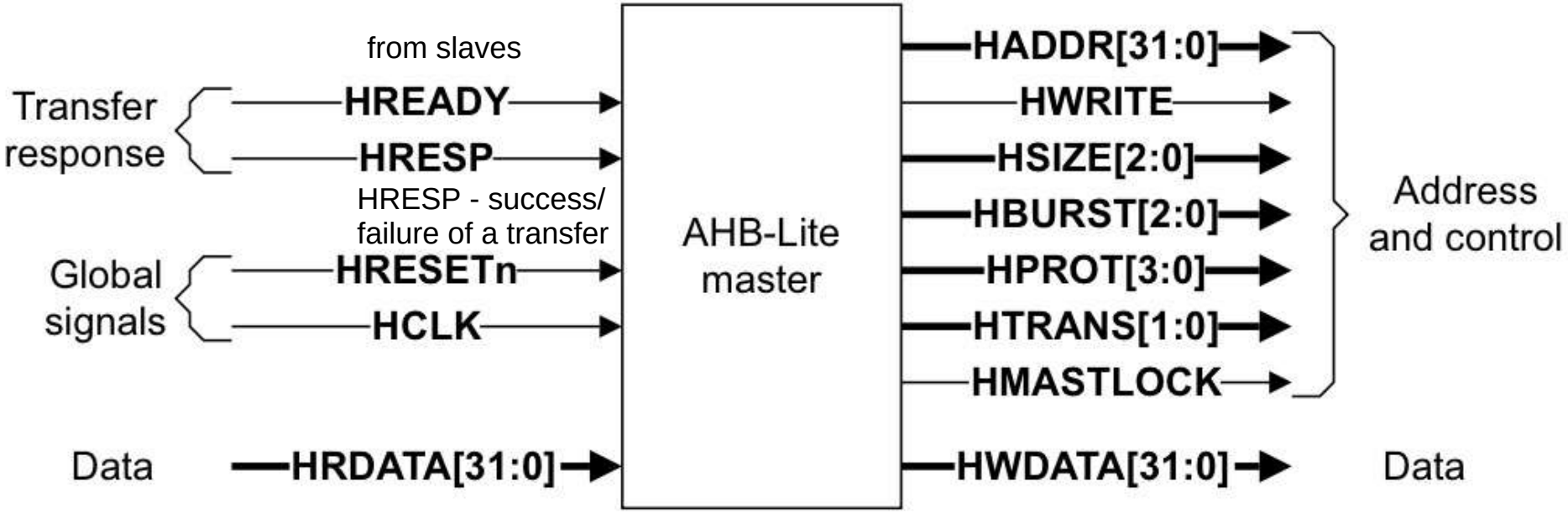
- Single master, multiple slaves
- Parallel
- Multiplexed (not tri-state)
- Separate data, address, control
- Synchronous
- Pipelined
- Burst transfer supported
- Split transaction not supported

AHB-Lite Block Diagram



Master Interface

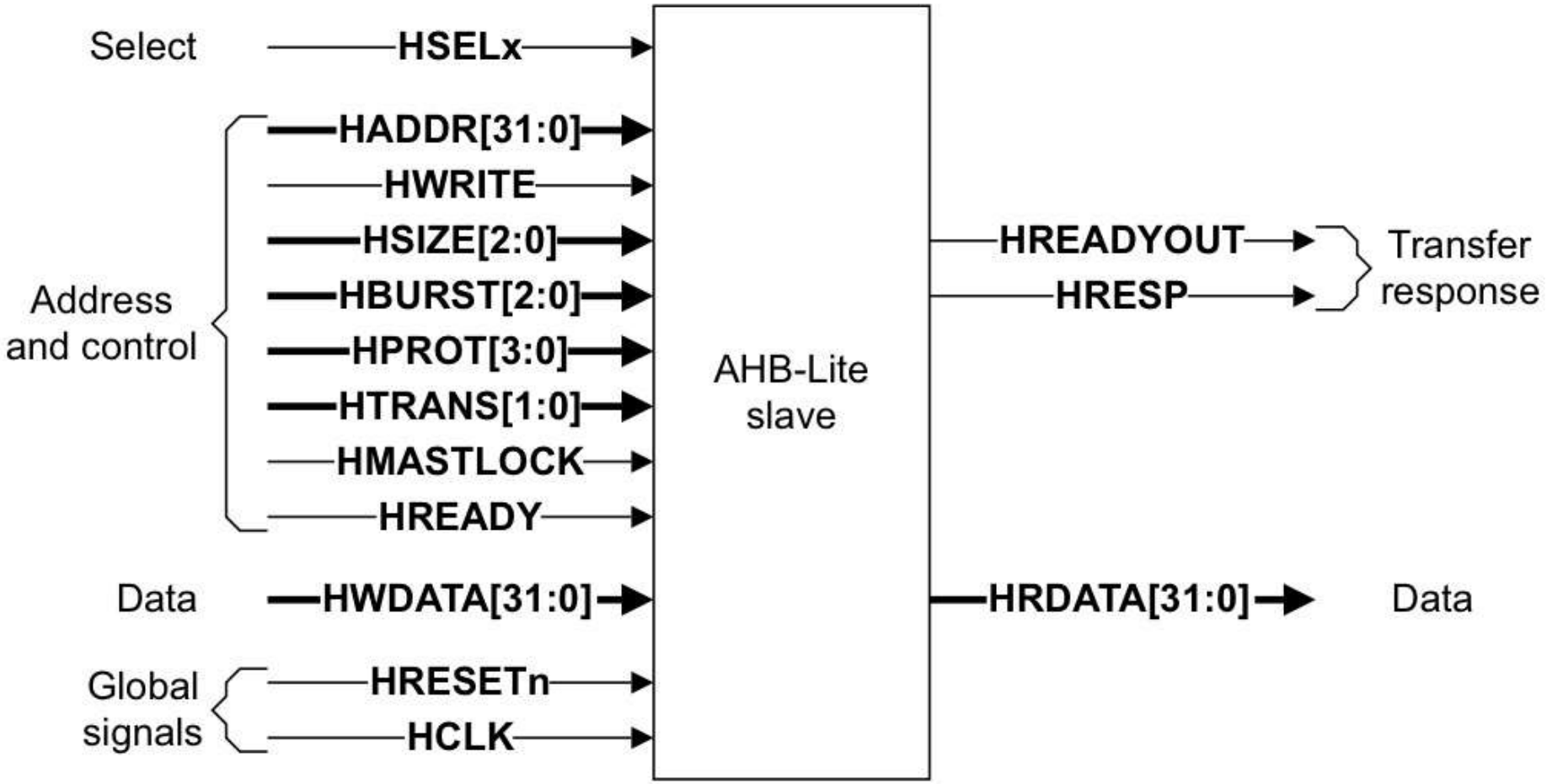
HREADY - slave asks the master
for extension of data phase by asking master to introduce wait cycles



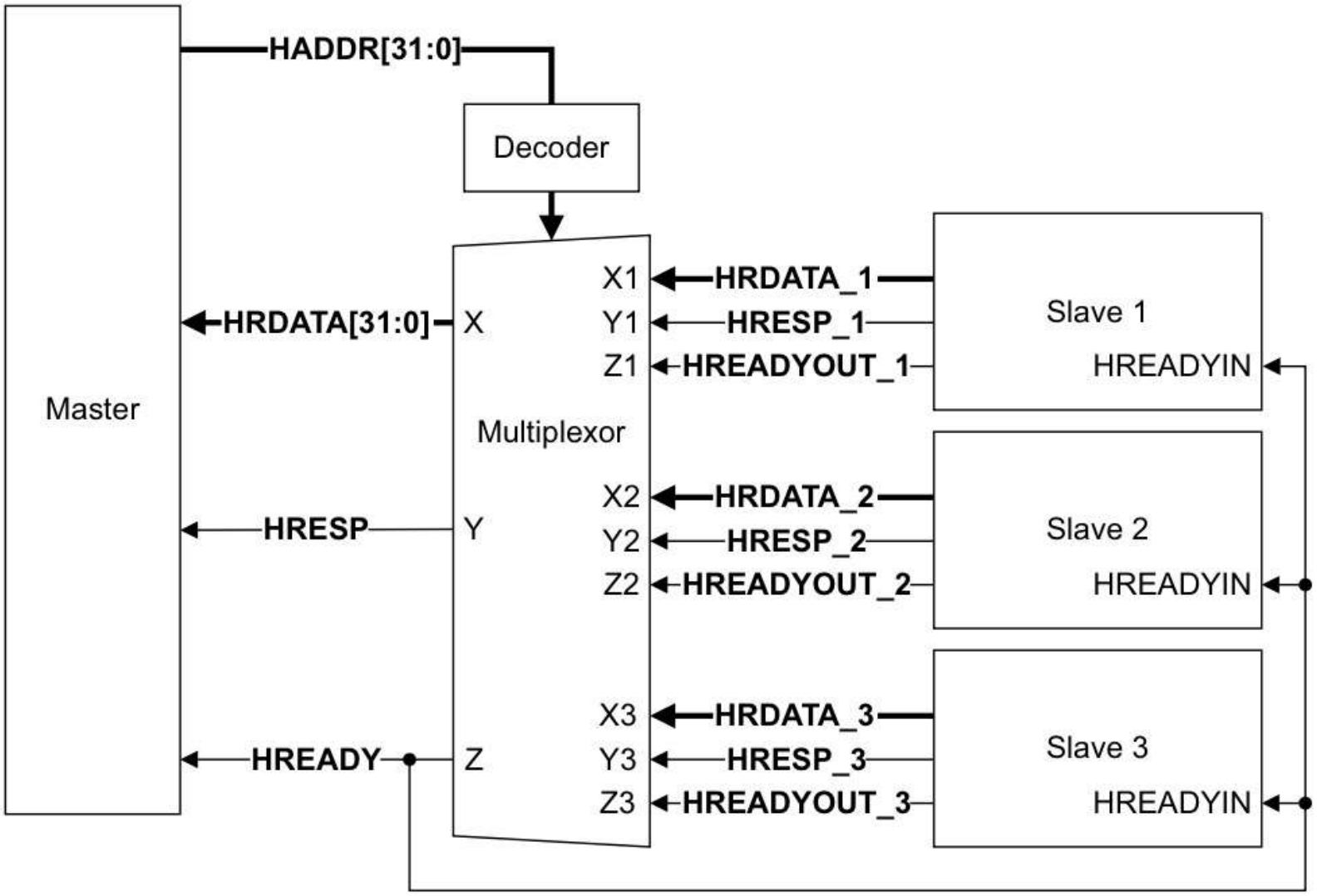
write data bus transfers data to be written from master to slave while read data bus moves from slave to master

Slave Interface

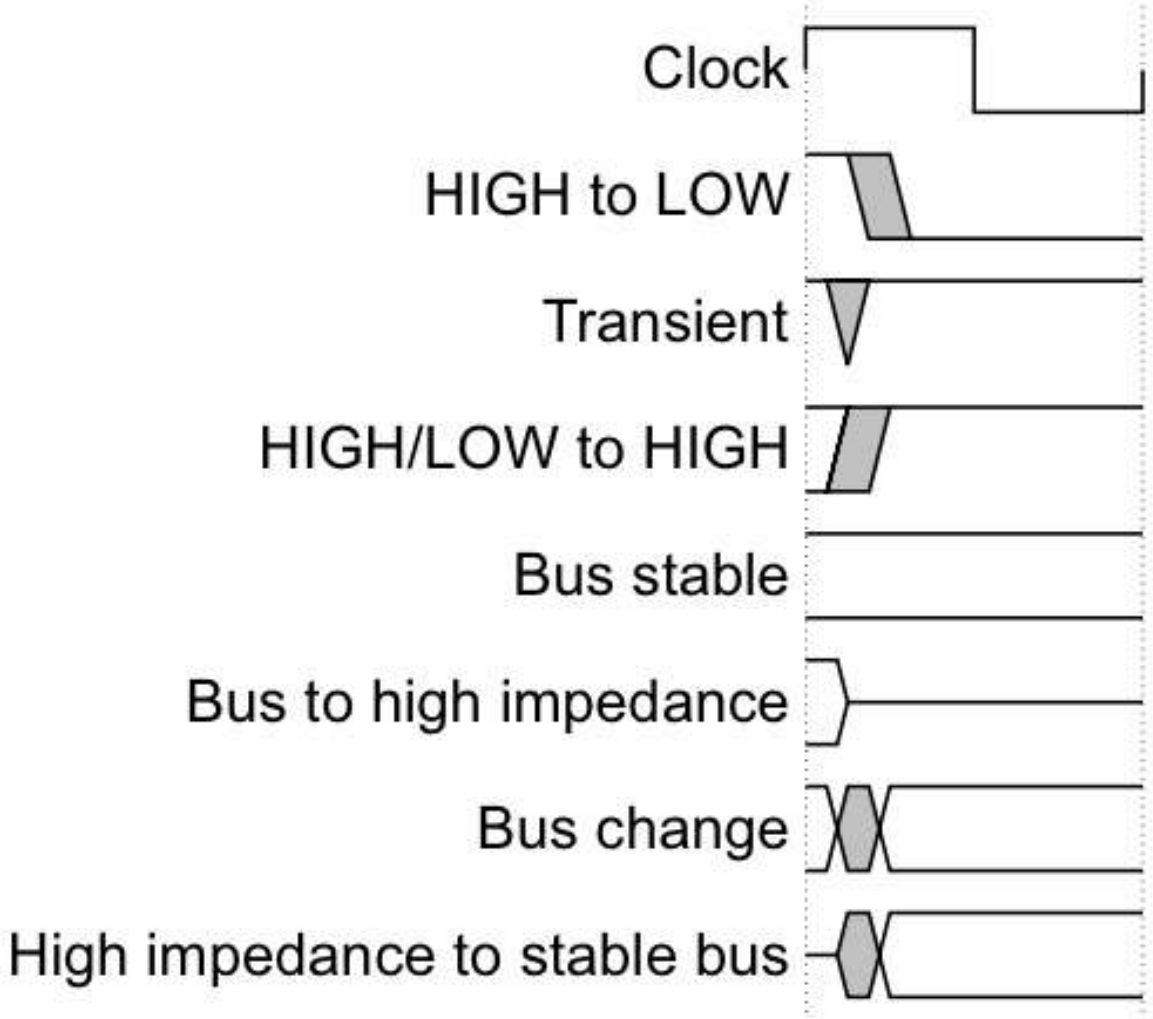
HREADY signal indicates to master and all slaves that the previous transfer is complete



Multiplexor Interconnection

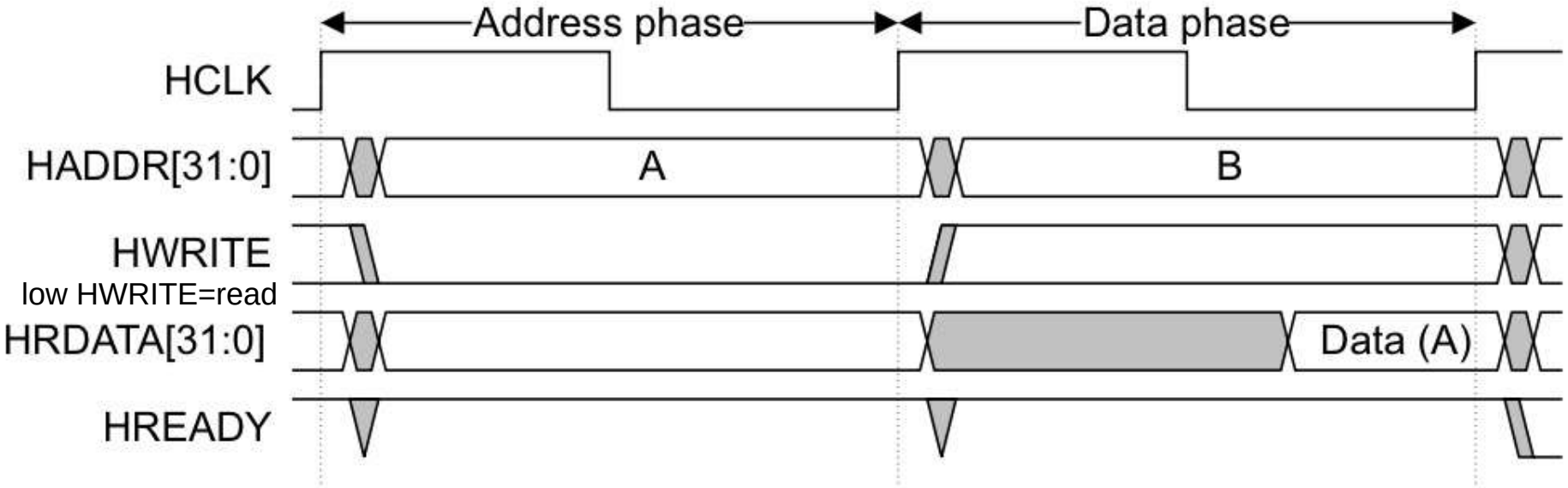


Timing Diagram Conventions



Read Transfer

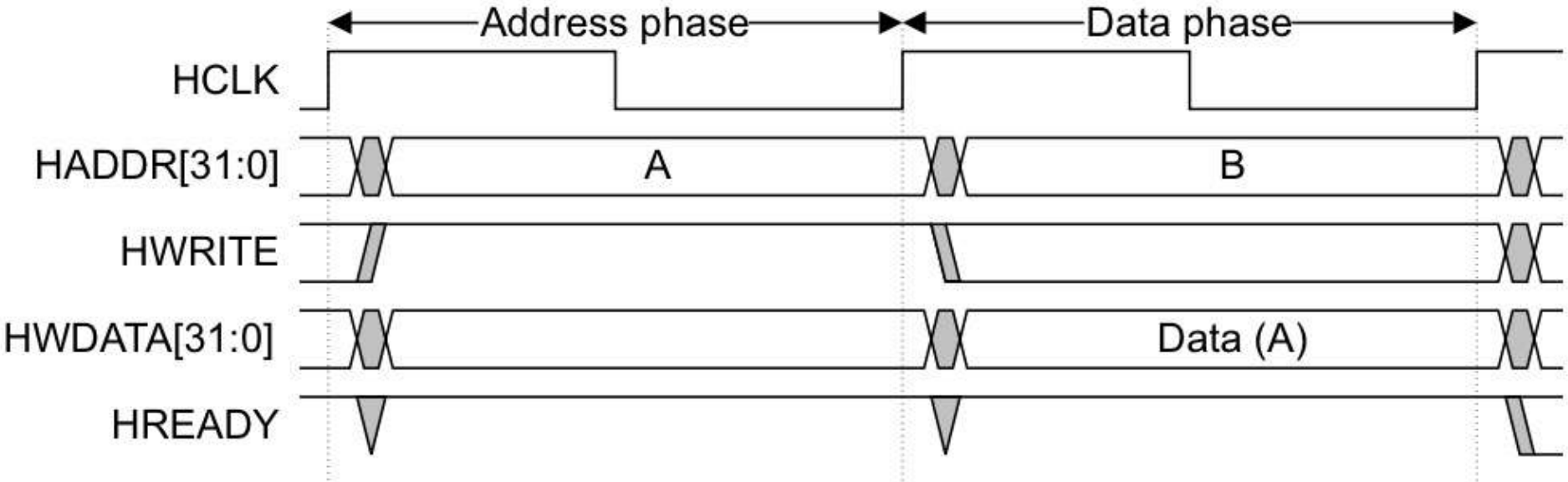
Advantage of this is pipelining, we can have overlap



Here, HREADY =1 indicates that the master has sent the address and next HREADY = 1 indicates that the slave is ready with the data. If not, it introduces wait cycles.

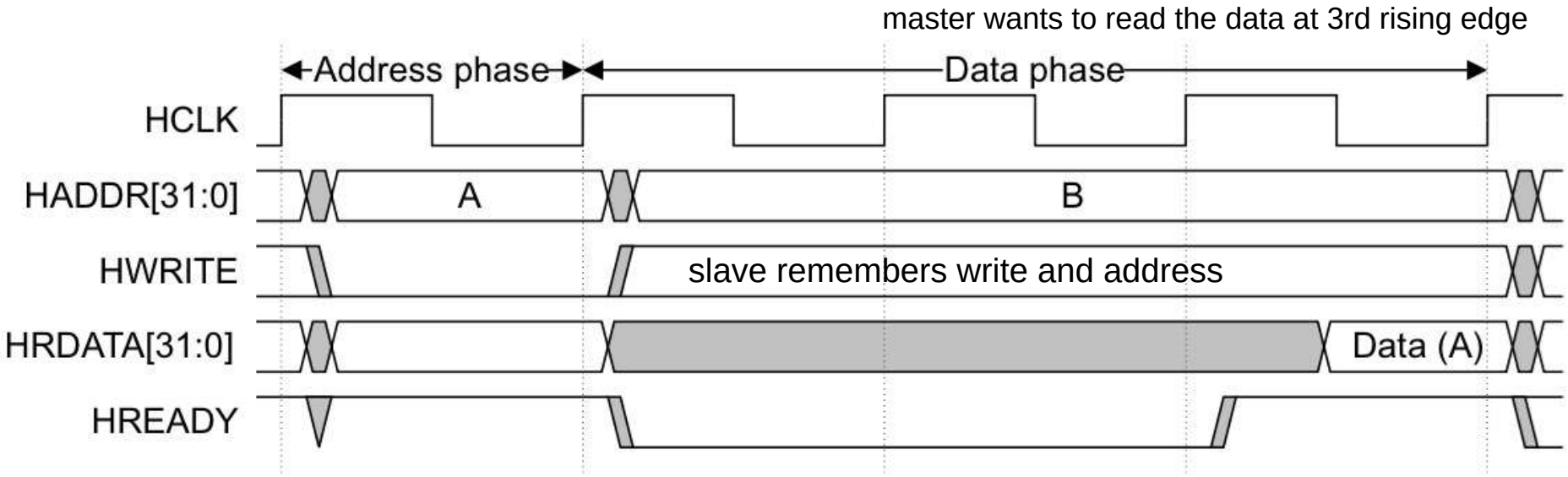
Write Transfer

note: in read transfer, data is stable a long time after the start, while in write transfer, we see data is stable just after the start.



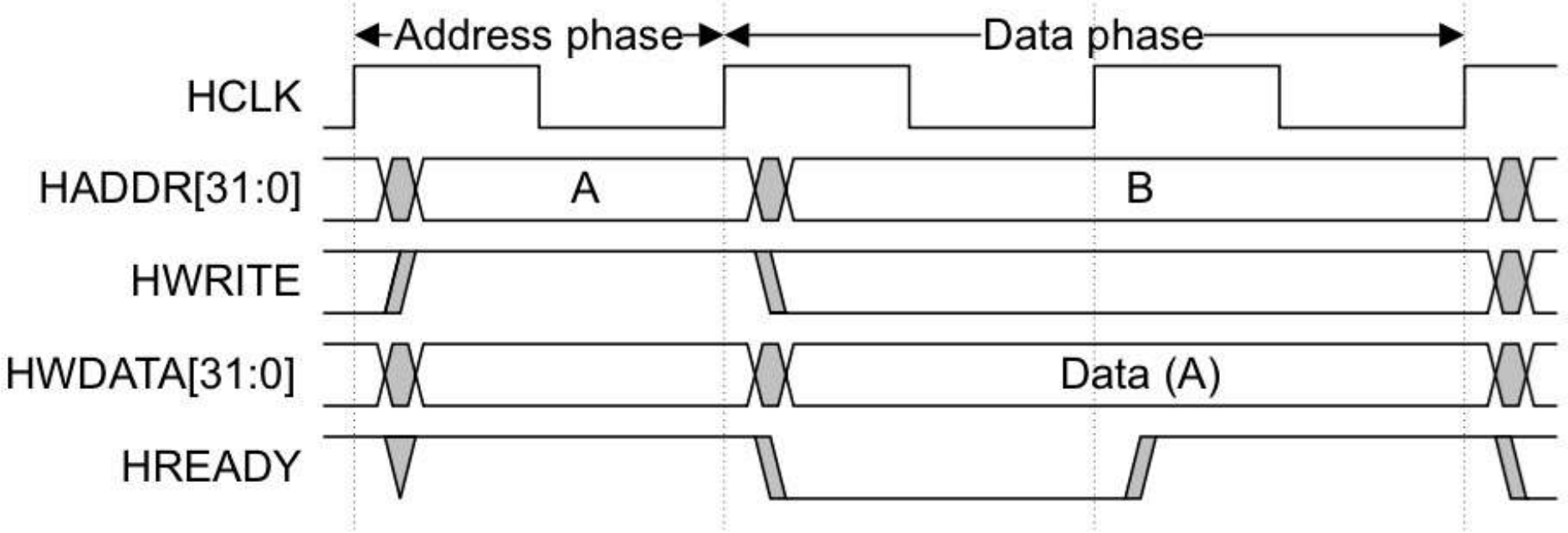
Data gets written at the rising edge of the 3rd clock cycle, after slave has seen the data to be written by keeping HREADY as 1 during the data phase itself. HWRITE is 1 to indicate the data needs to be written

Read Transfer with Wait States

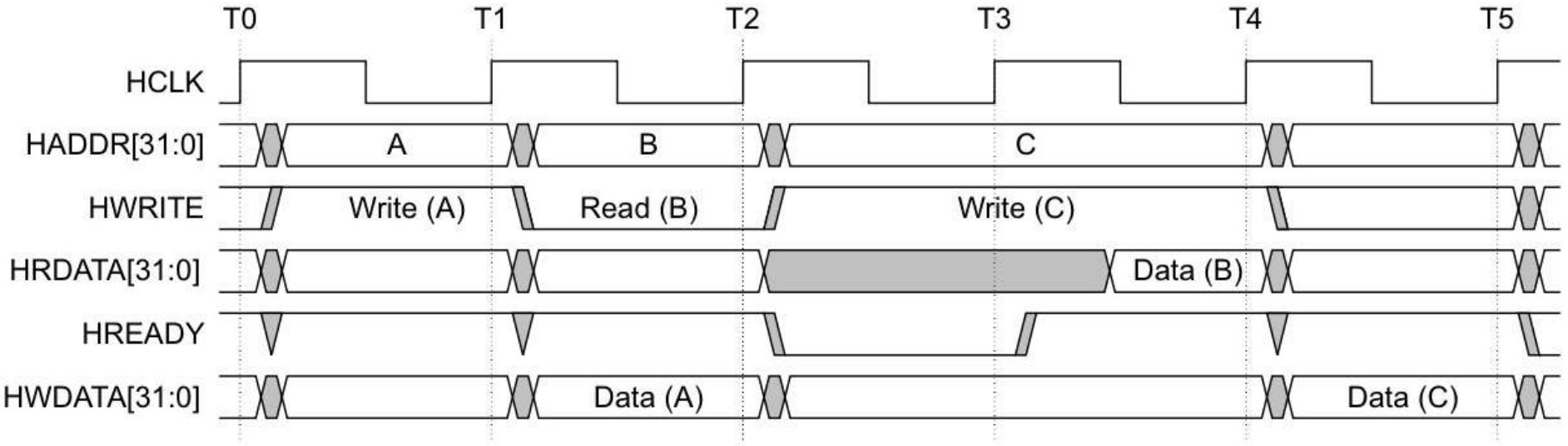


Control comes to slave when we are talking about data while control remains with master when we are talking about address

Write Transfer with Wait States



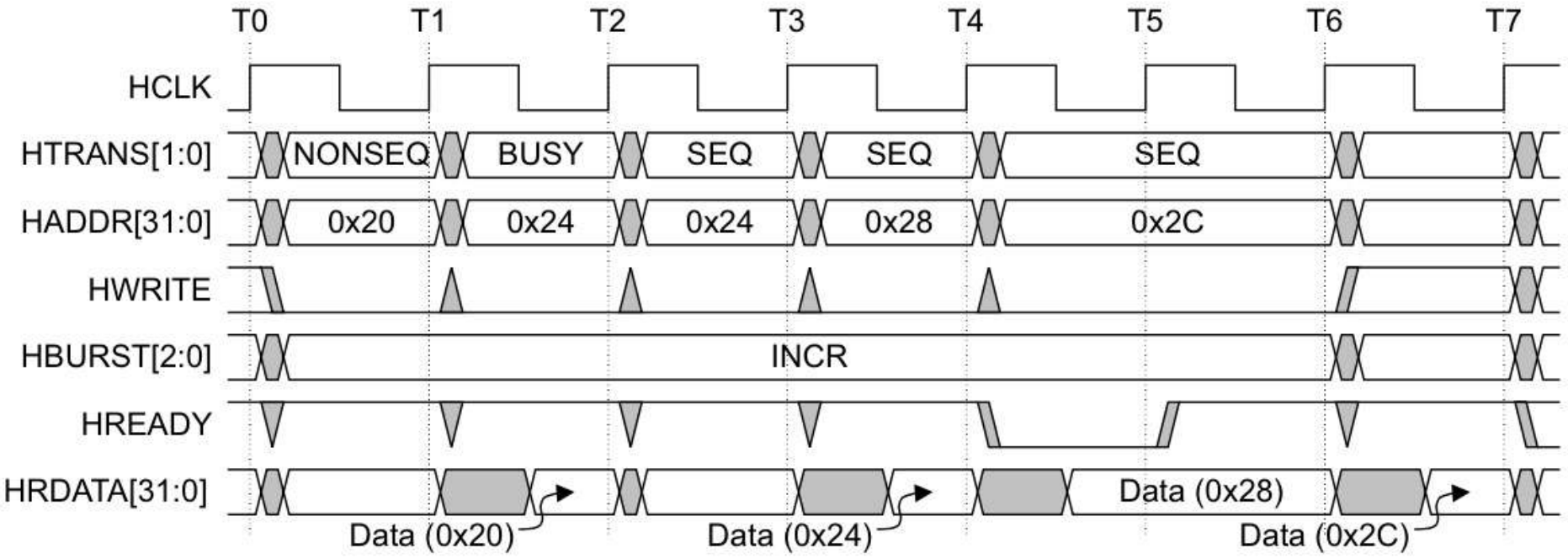
Multiple Transfers



Transfer Types

HTRANS[1:0]	Type	
00	IDLE	Master does not want transfer
01	BUSY	data on data bus or address bus can be ignored Master inserts wait cycle
10	NONSEQ	Single transfer or first transfer of a burst
11	SEQ	Remaining transfers of a burst

Transfer Type Examples



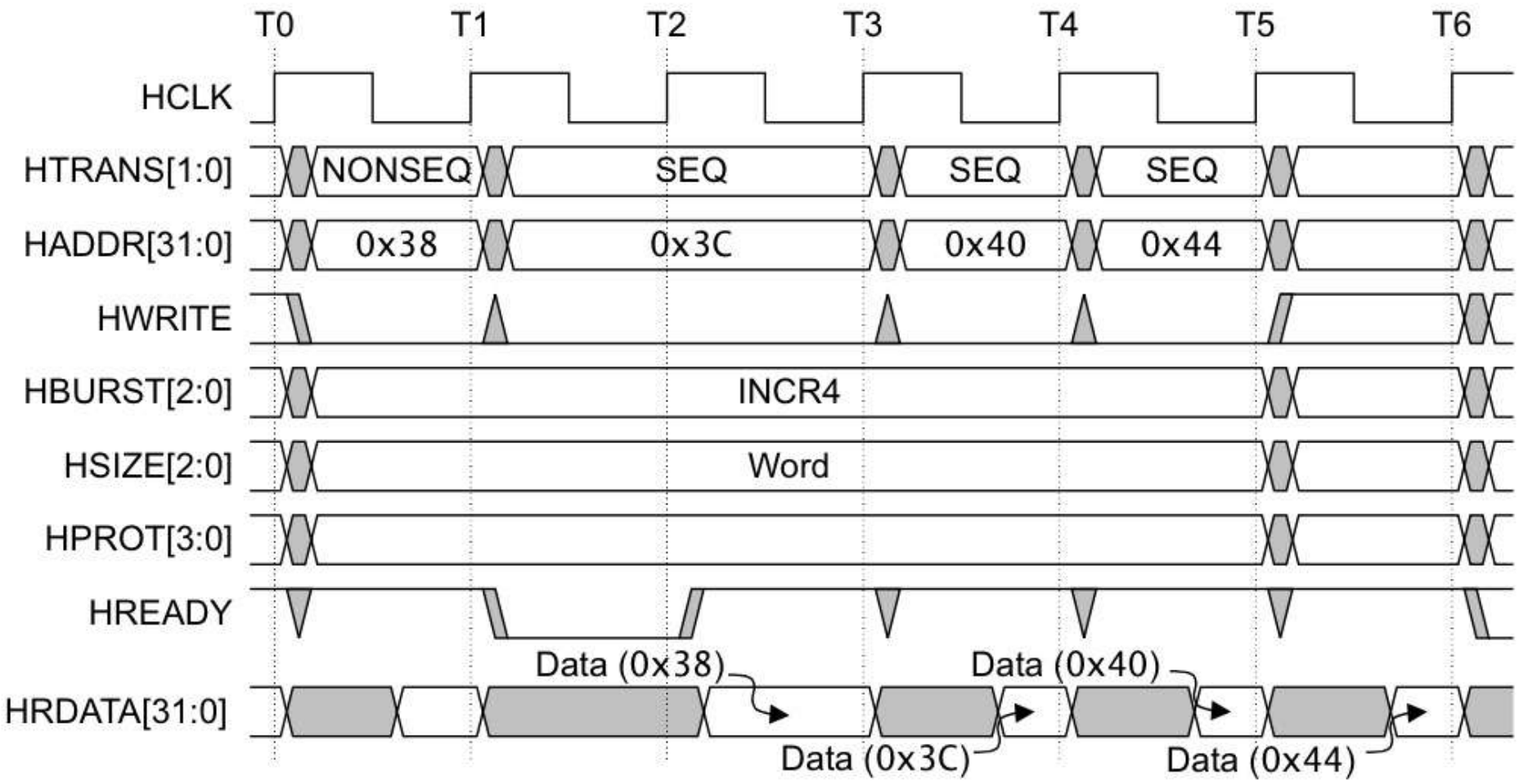
Transfer Size

HSIZE[2:0]	Size (bits)	Size (bytes)
000	8	1
001	16	2
010	32	4
011	64	8
100	128	16
101	256	32
110	512	64
111	1024	128

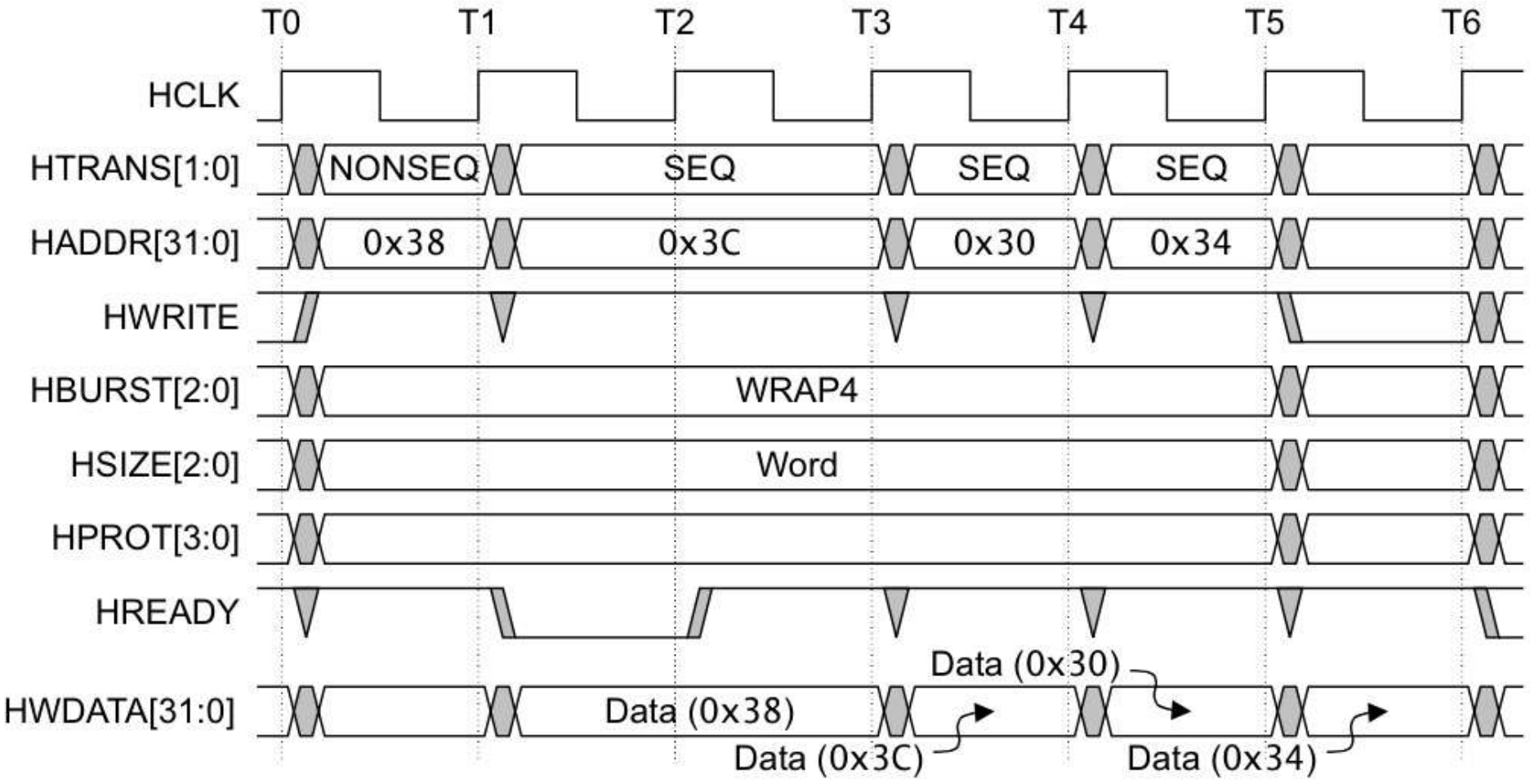
Burst type and size

HBURST[2:0]	Type and size
000	SINGLE
001	INCR
010	WRAP4
011	INCR4
100	WRAP8
101	INCR8
110	WRAP16
111	INCR16

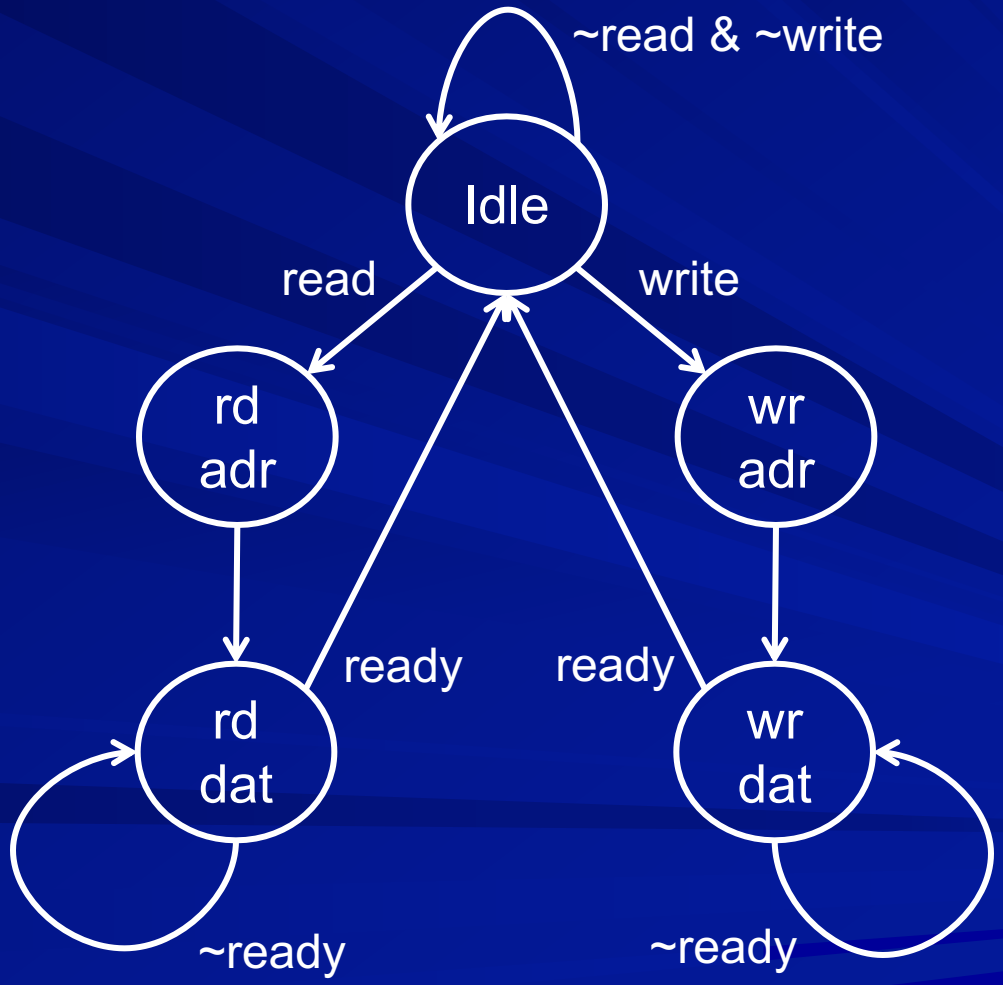
Four-Beat Incrementing Burst



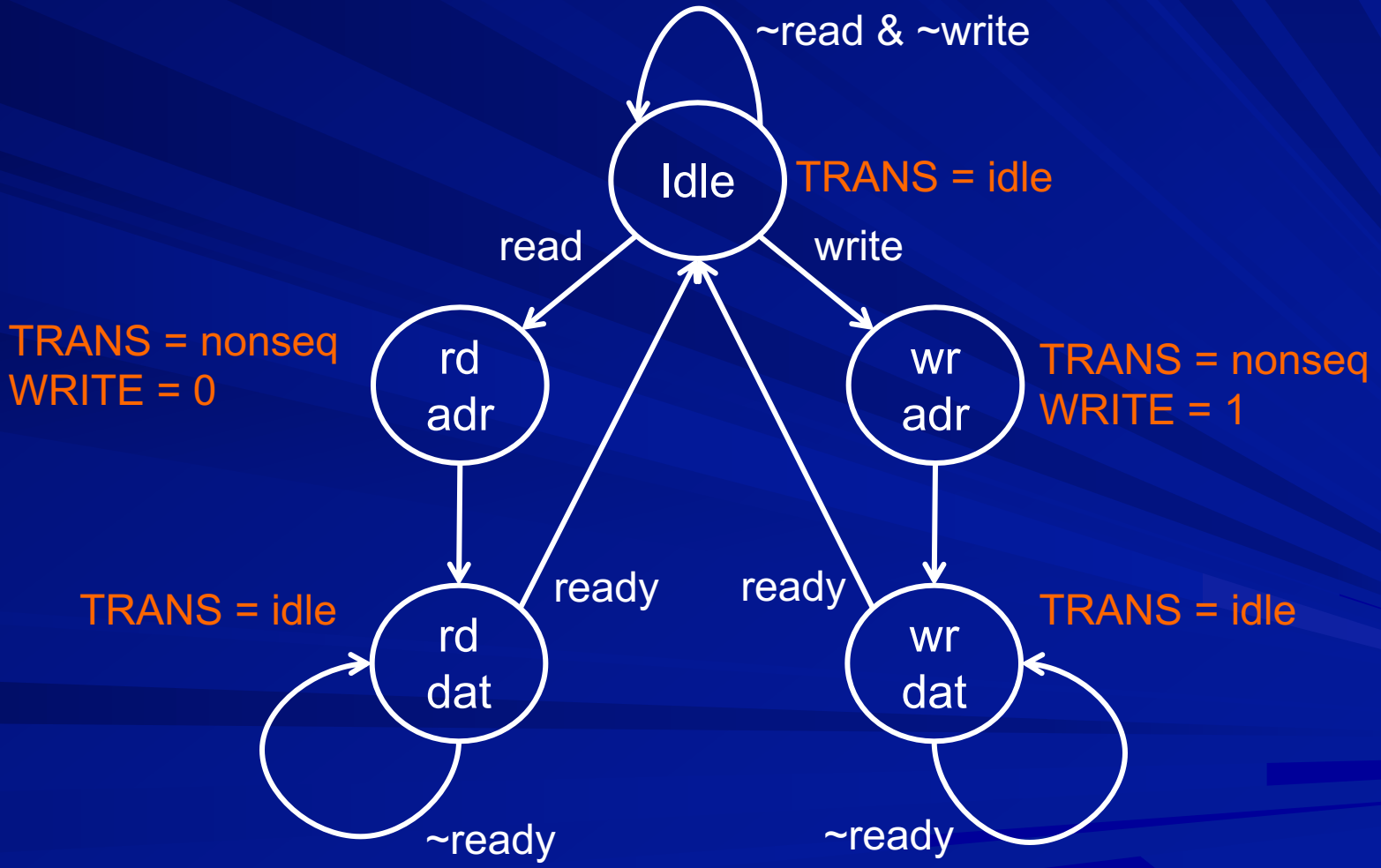
Four-Beat Wrapping Burst



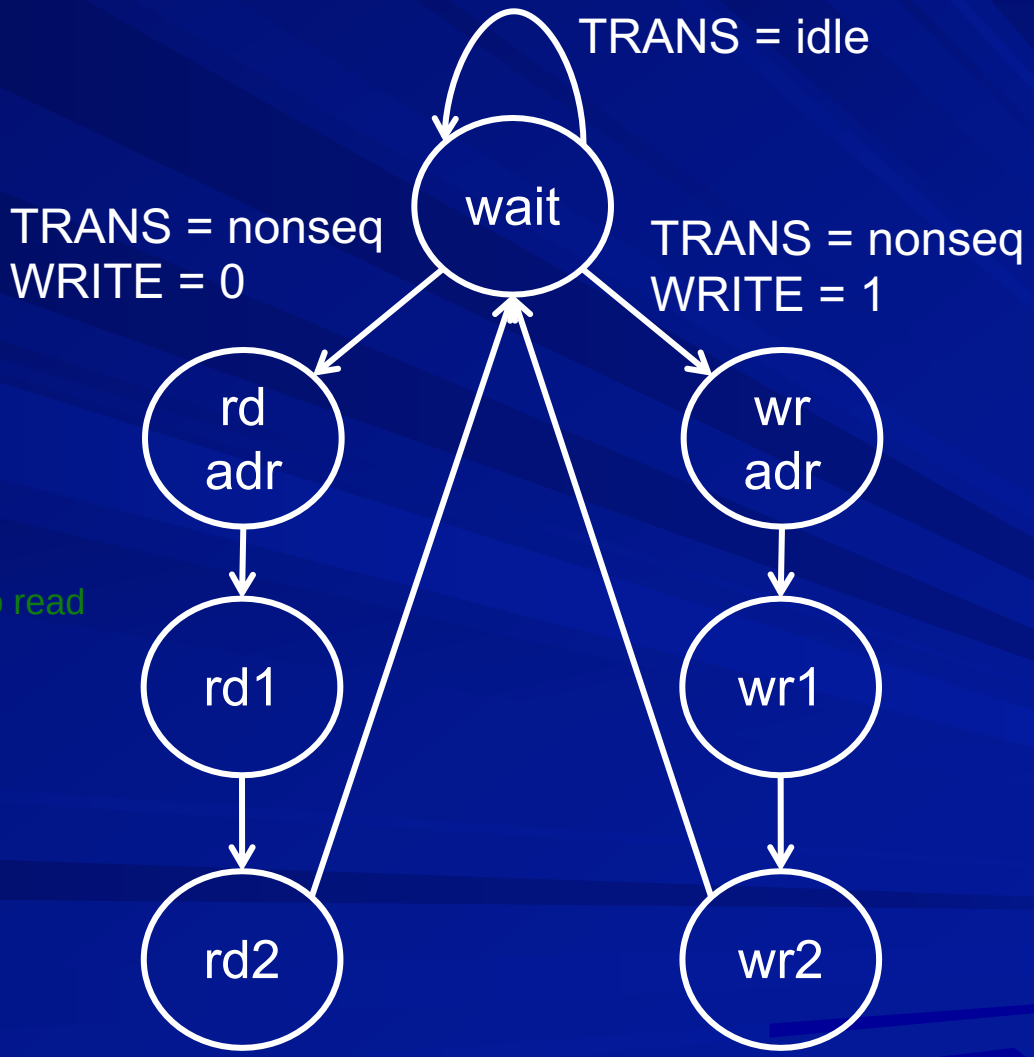
Master interface



Master interface



Slave interface



here it takes 2 cycles to read

Obtaining access to a bus

I/O device are peripherals could act as master via DMA controller

Masters and slaves on a bus

masters (processors/peripherals) initiate transactions

slaves (peripherals/memories) respond to the masters

How does a master get control of a bus?

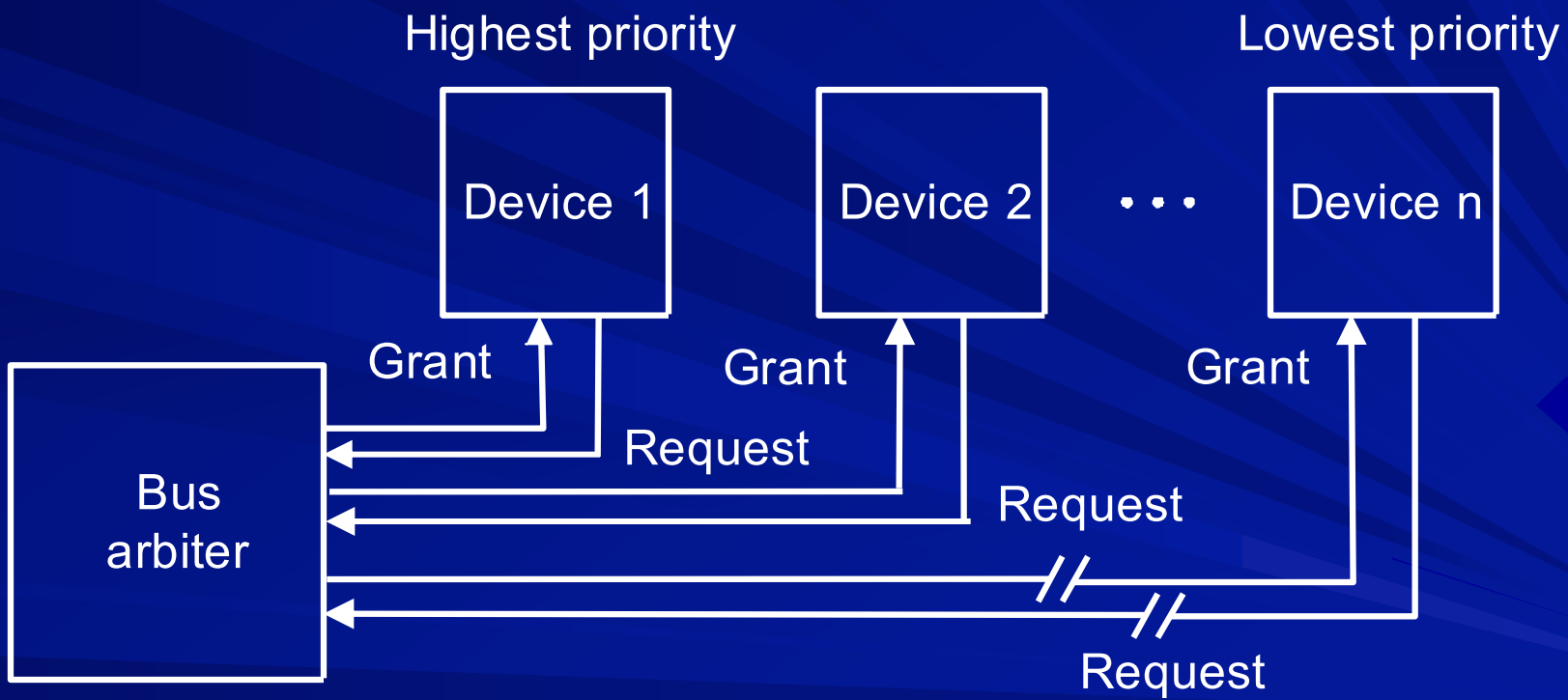
Priorities may be assigned. Fairness is essential.

- Daisy chaining
- Centralized parallel arbitration
- Distributed arbitration by self selection
- Distributed arbitration by collision detection

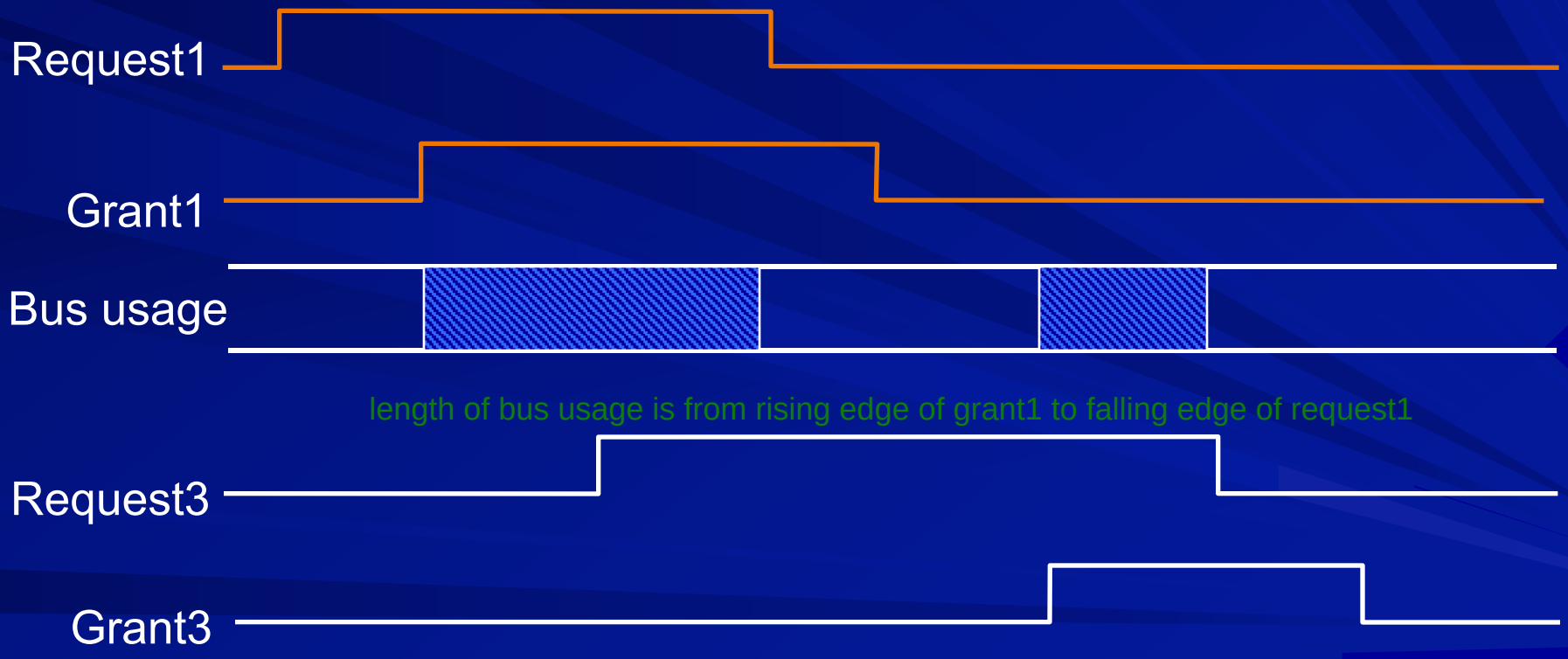
multiple masters - these devices are multiple masters

Centralized parallel arbitration

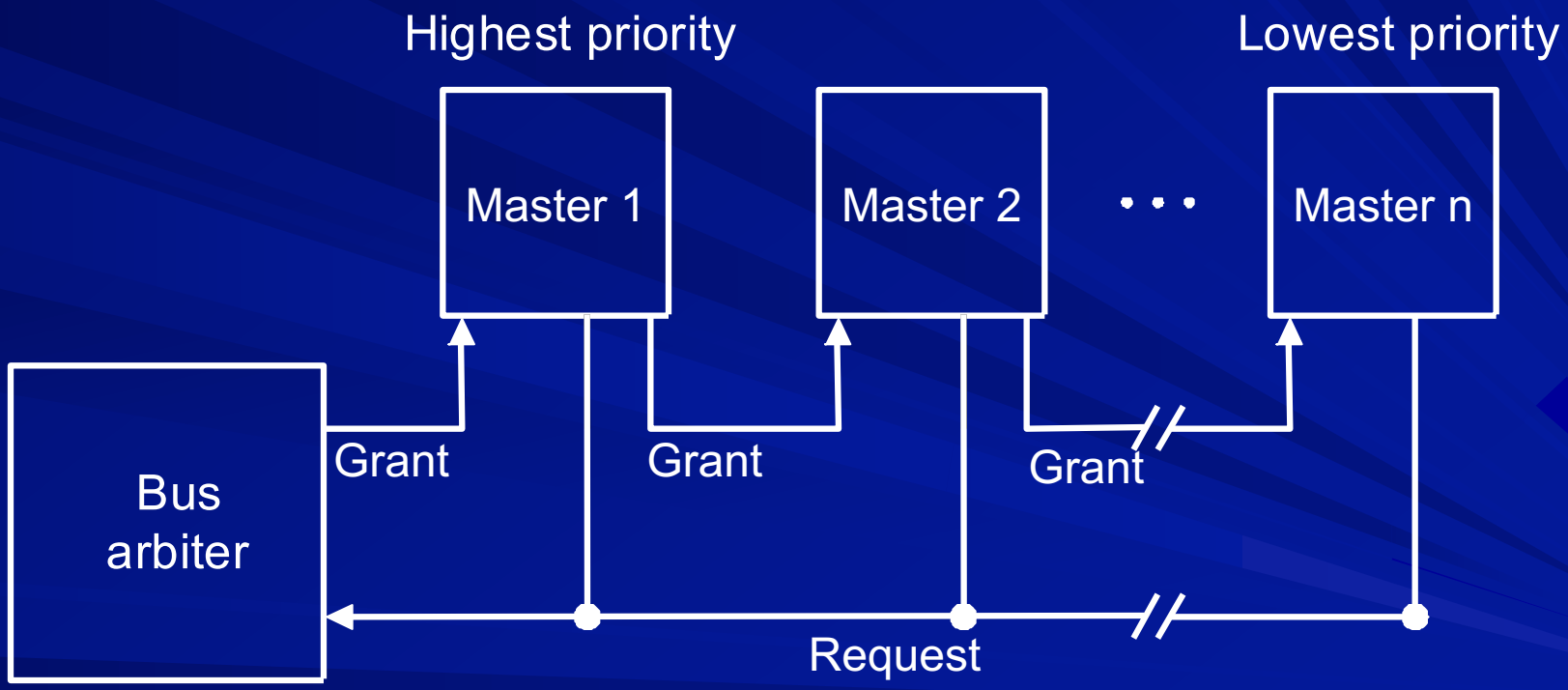
Example: PCI bus (a backplane bus)



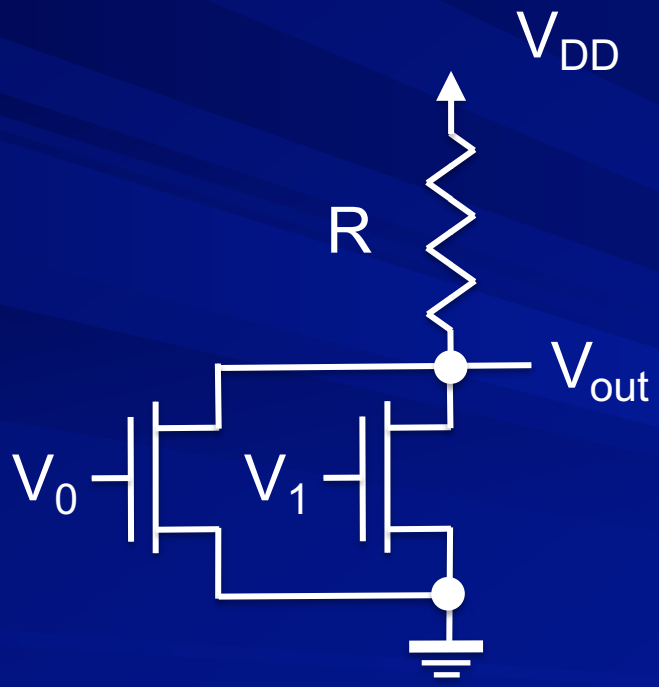
Request and Grant signals



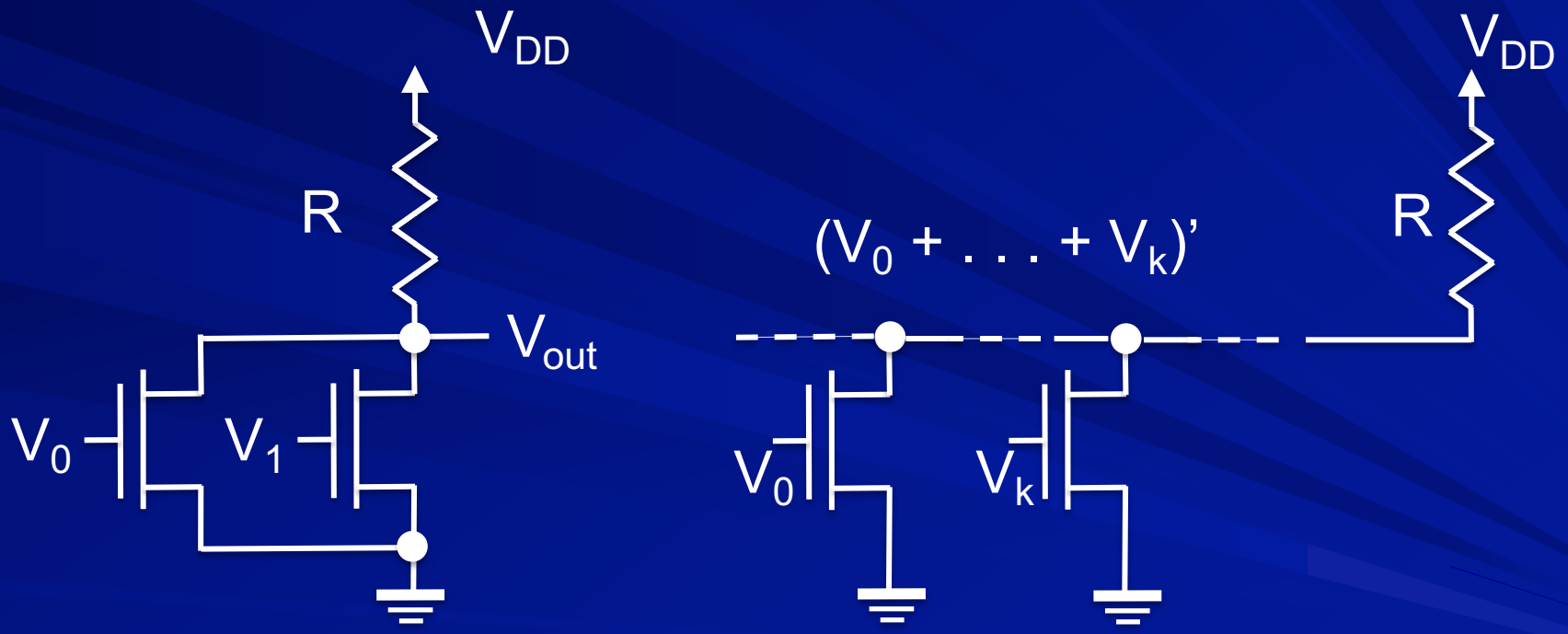
Combining/chaining signals



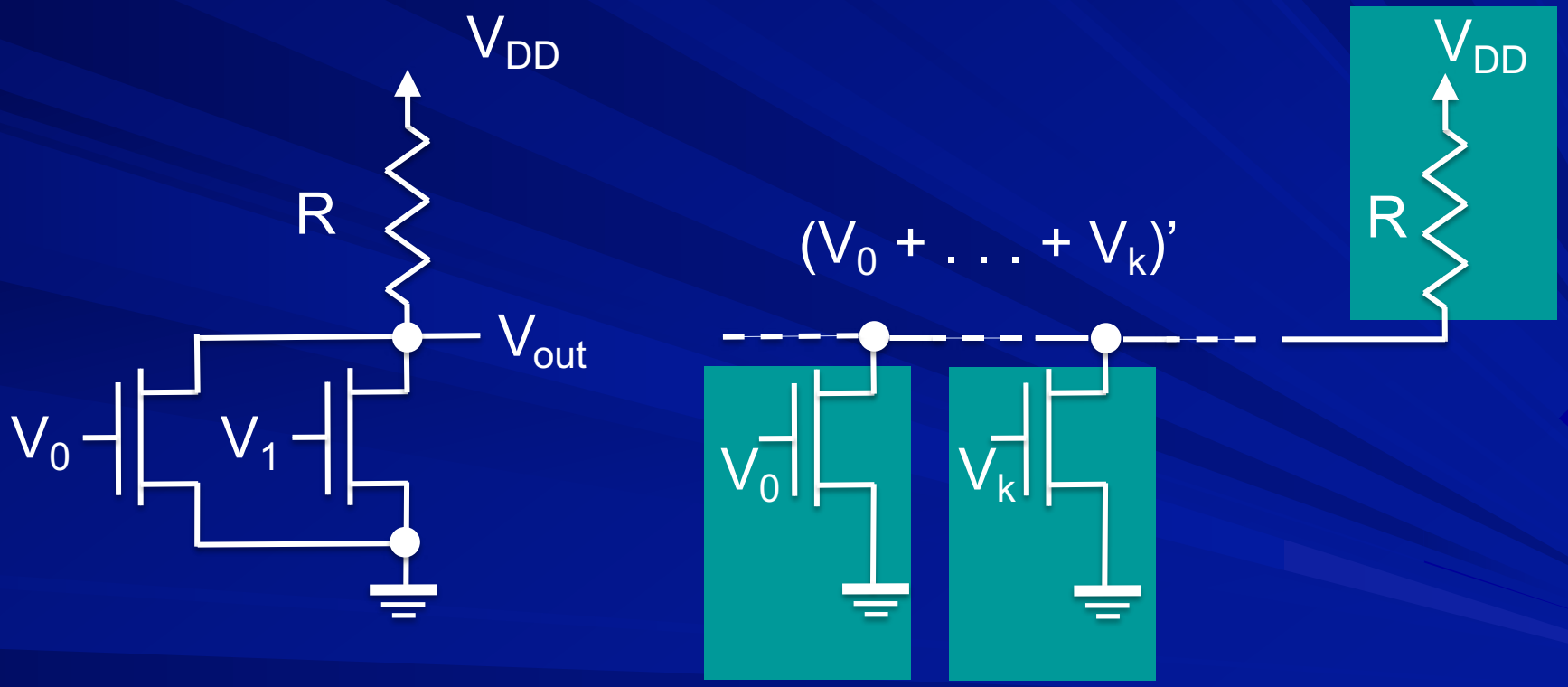
Wired OR



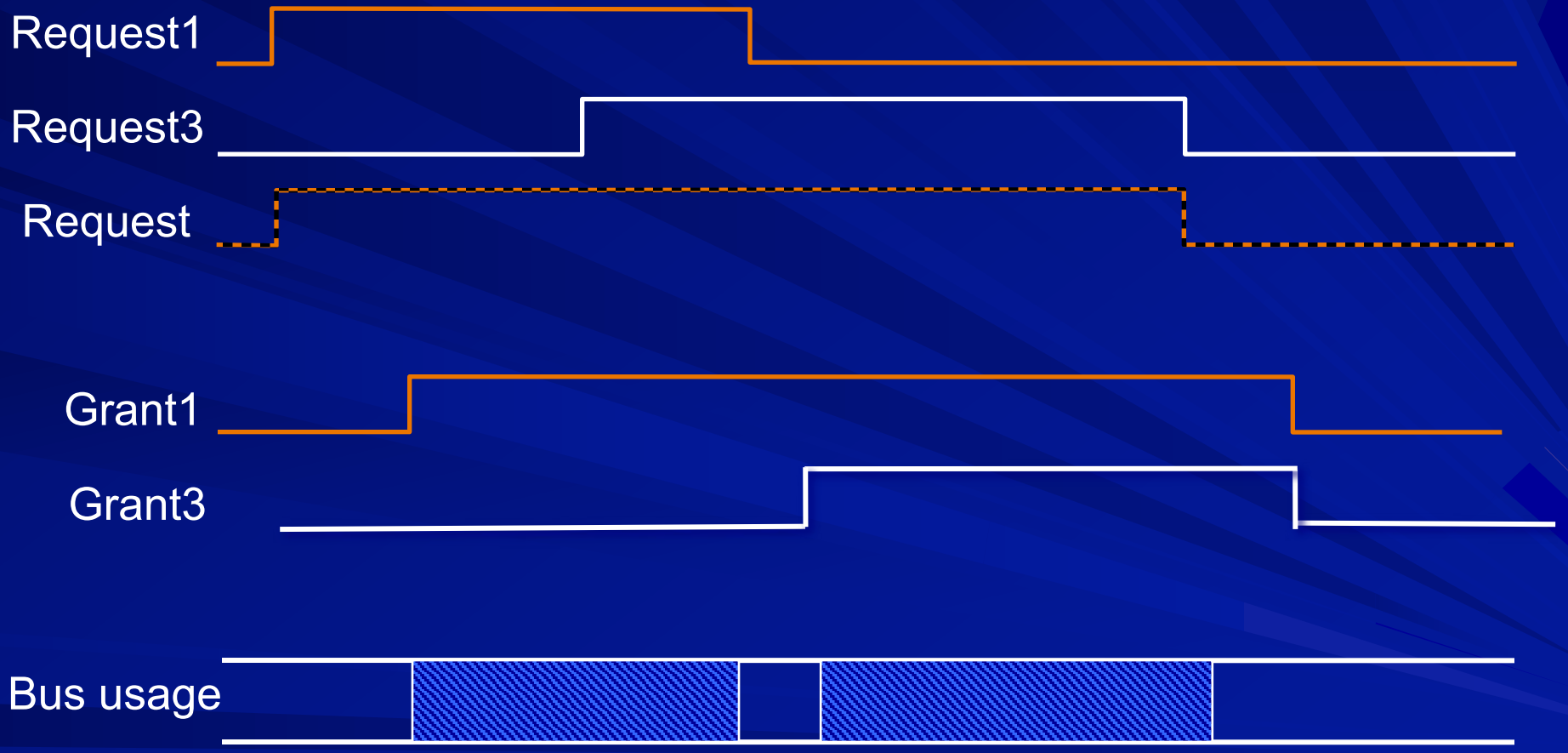
Wired OR



Wired OR

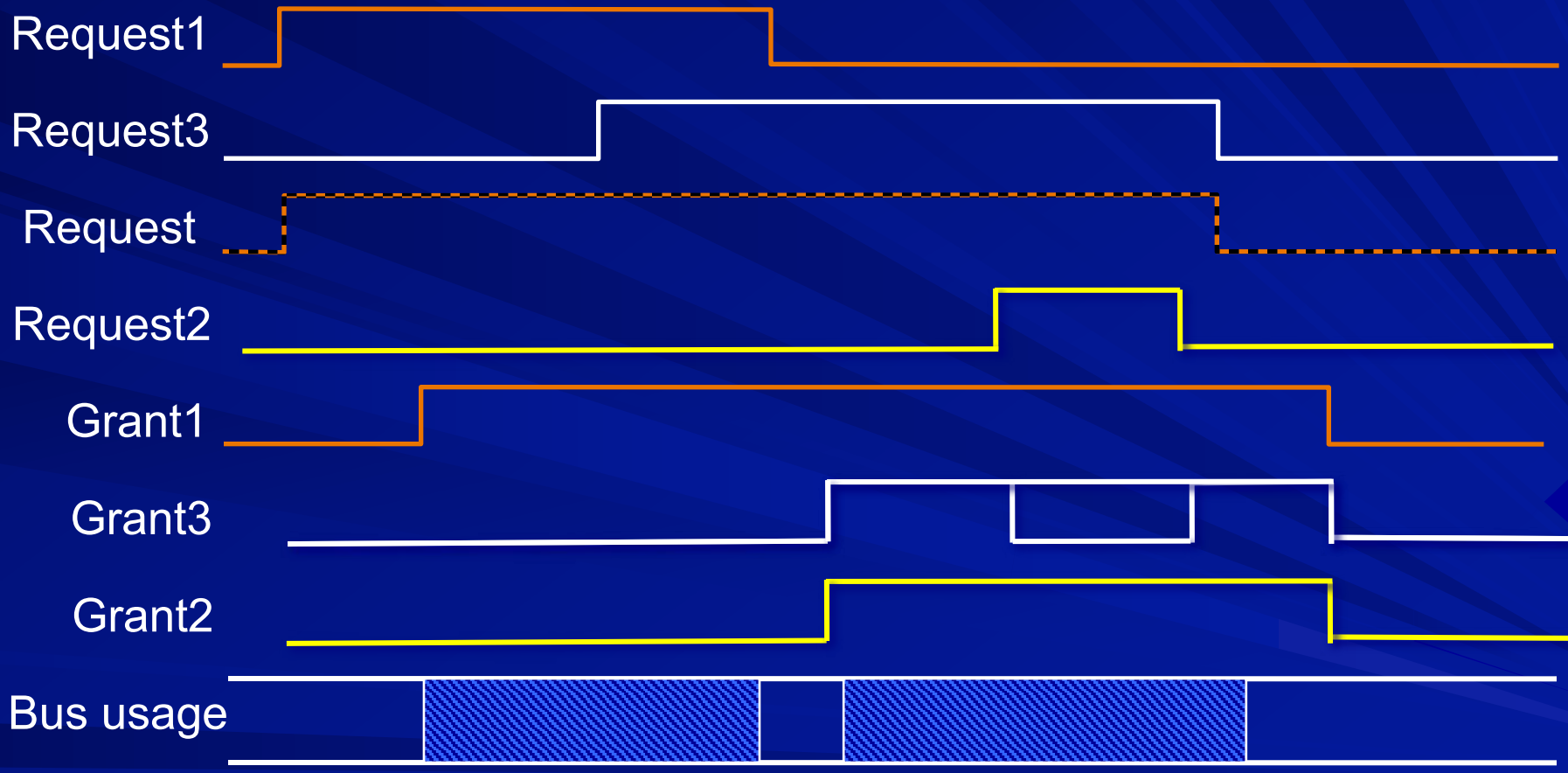


Request and Grant signals



Problem: High priority device can usurp the grant signal

Request and Grant signals



Problem: High priority device can usurp the grant signal
Solution: Daisy chain – request, grant, release

As soon as we see request high, we do grant high and immediately make request low, then use bus and do release high and immediately grant goes down and release goes down after that.

Daisy chain

Arbiter

Grant ↑

Grant ↓

Device

Request ↑

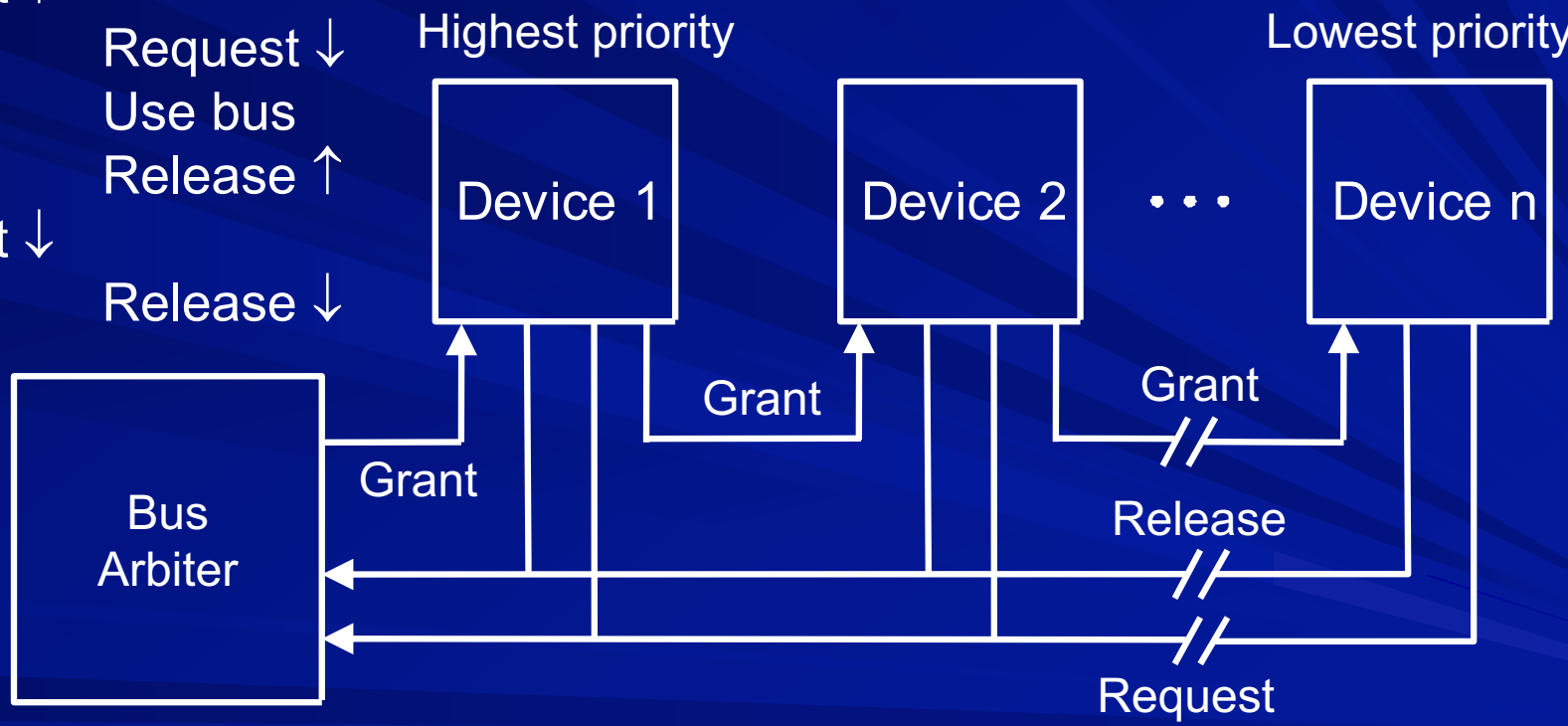
Request ↓

Use bus

Release ↑

Release ↓

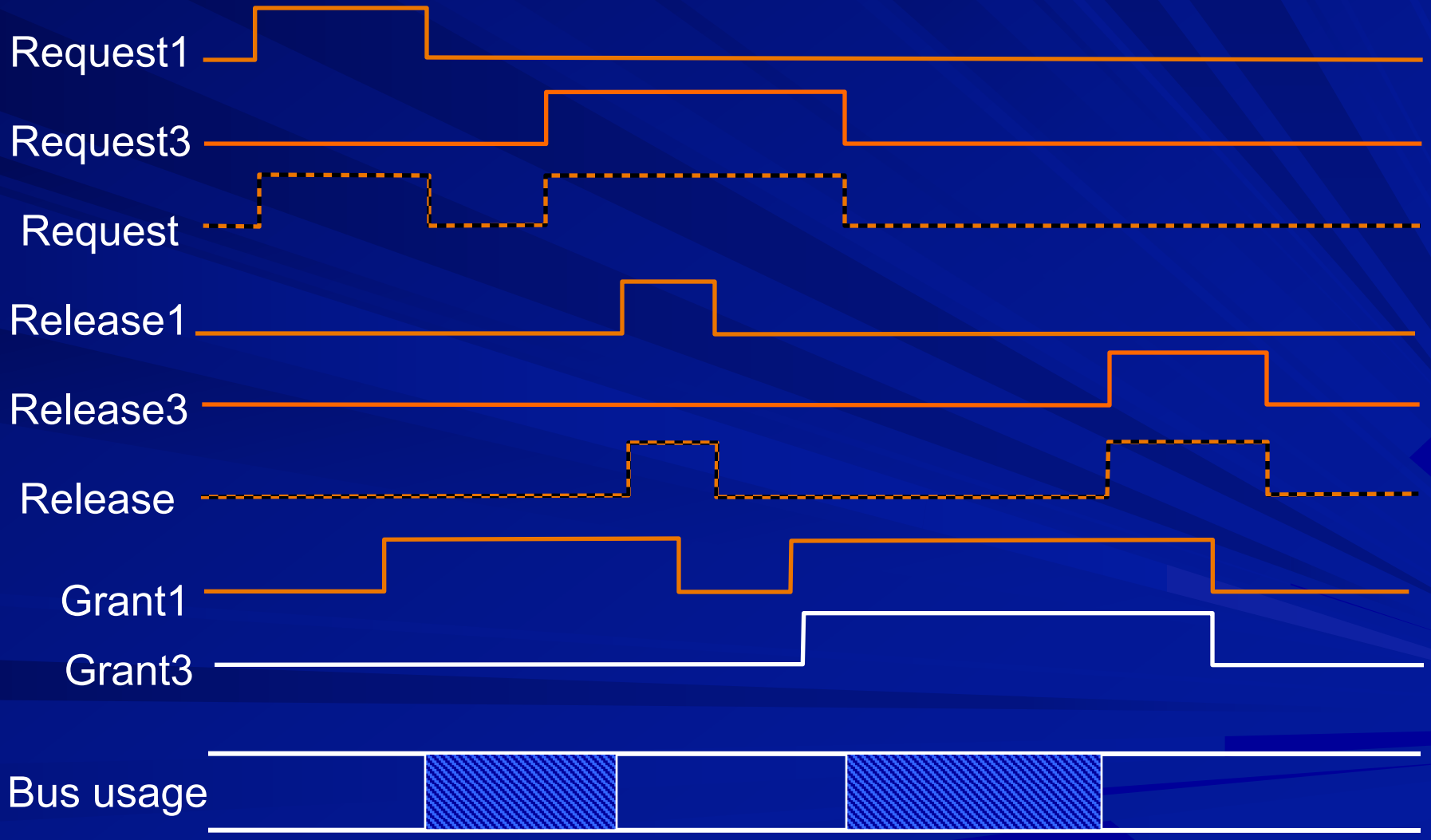
Example: VME bus



Note: Rising edge on Grant signal is significant, not level

grant sees request3 is 1 so it immediately goes up and comes down when release3 is 1. bus usage is between grant start and release rising edge

Request, release and grant signals



so if master2 or master1 comes again with a request before master3 had a chance to use it then we will have starving of master3. rising edge of grant can only determine if the master gets to use the bus or not. if master2 comes after master3 has started using the bus, then it has to wait till it sees the grant's rising edge

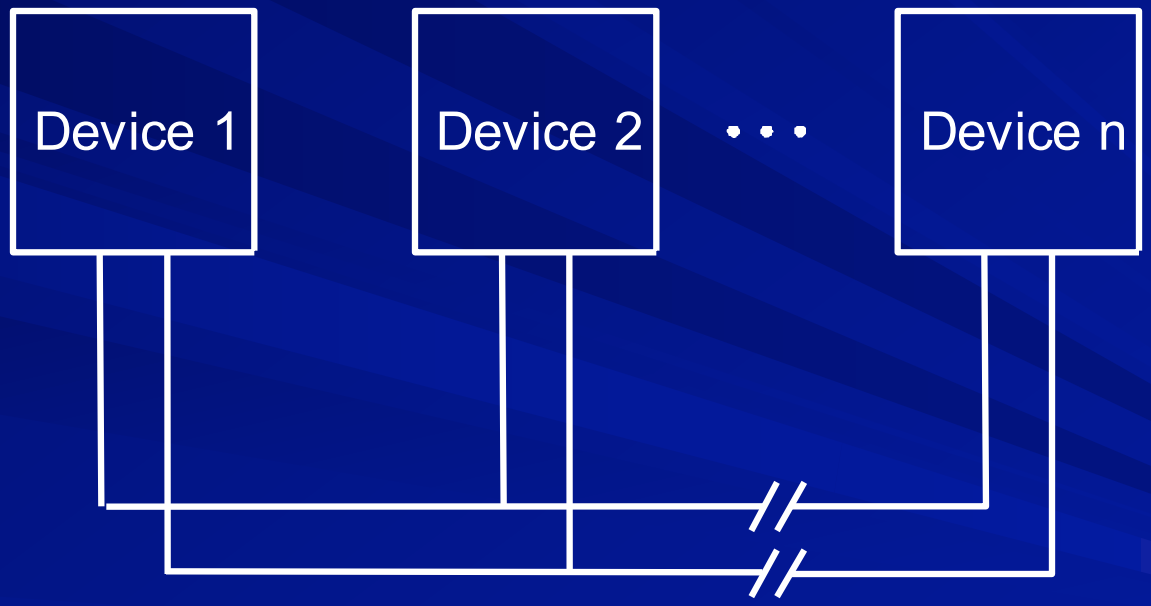
Daisy chain characteristics

- Simple and inexpensive
- Speed is limited due to chaining of grant signal
- Starvation of low priority devices may not be prevented

Rule to improve fairness: A device that has just used the bus cannot reacquire it until it sees the request line go low

Distributed arbitration

Example: NuBus (a backplane bus) in Apple Macintosh



Request lines and identity codes

Distributed arbitration in NuBus

- $req = req^0 + req^1 + .. + req^{n-1}$ (wired OR)
- $ID^i_3 .. ID^i_0$ = ID of i^{th} requester
- $arb^i_3 .. arb^i_0$ = arb output of i^{th} requester
- $arb_{3..0} = arb^0_{3..0} + .. + arb^{n-1}_{3..0}$ (wired OR)
- $arb^i_3 = ID^i_3 . req^i$
- $arb^i_2 = ID^i_2 . req^i . (ID^i_3 + arb_3')$
- $arb^i_1 = ID^i_1 . req^i . (ID^i_3 + arb_3') . (ID^i_2 + arb_2')$
- $arb^i_0 = ID^i_0 . req^i . (ID^i_3 + arb_3') . (ID^i_2 + arb_2') . (ID^i_1 + arb_1')$

if all are 0, then u can continue, else if u have 1 then u can continue, this logic, basically determines the highest priority device.

we have previous ones since once we have dropped off the race, we should not continue even if we have 1 later in lsb

NuBus arbitration example-1

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1				
2	0	1	1	0				
3	1	1	0	0				
4	0	0	1	1				

NuBus arbitration example-1

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0	0	1
2	0	1	1	0	0	1	1	0
3	1	1	0	0	1	1	0	0
4	0	0	1	1	0	0	1	1
			OR		1	1	1	1

NuBus arbitration example-1

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1			
2	0	1	1	0	0			
3	1	1	0	0	1			
4	0	0	1	1	0			
					1			

NuBus arbitration example-1

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0		
2	0	1	1	0	0	x		
3	1	1	0	0	1	1		
4	0	0	1	1	0	x		
					1	1		

NuBus arbitration example-1

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0	x	
2	0	1	1	0	0	x	x	
3	1	1	0	0	1	1	0	
4	0	0	1	1	0	x	x	
					1	1	0	

NuBus arbitration example-1

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0	x	x
2	0	1	1	0	0	x	x	x
3	1	1	0	0	1	1	0	0
4	0	0	1	1	0	x	x	x
					1	1	0	0

NuBus arbitration example-2

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1				
2	0	1	1	0				
3	1	0	1	0				
4	0	0	1	1				

NuBus arbitration example-2

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0	0	1
2	0	1	1	0	0	1	1	0
3	1	0	1	0	1	0	1	0
4	0	0	1	1	0	0	1	1
			OR		1	1	1	1

NuBus arbitration example-2

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1			
2	0	1	1	0	0			
3	1	0	1	0	1			
4	0	0	1	1	0			
					1			

NuBus arbitration example-2

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0		
2	0	1	1	0	0	x		
3	1	0	1	0	1	0		
4	0	0	1	1	0	x		
					1	0		

NuBus arbitration example-2

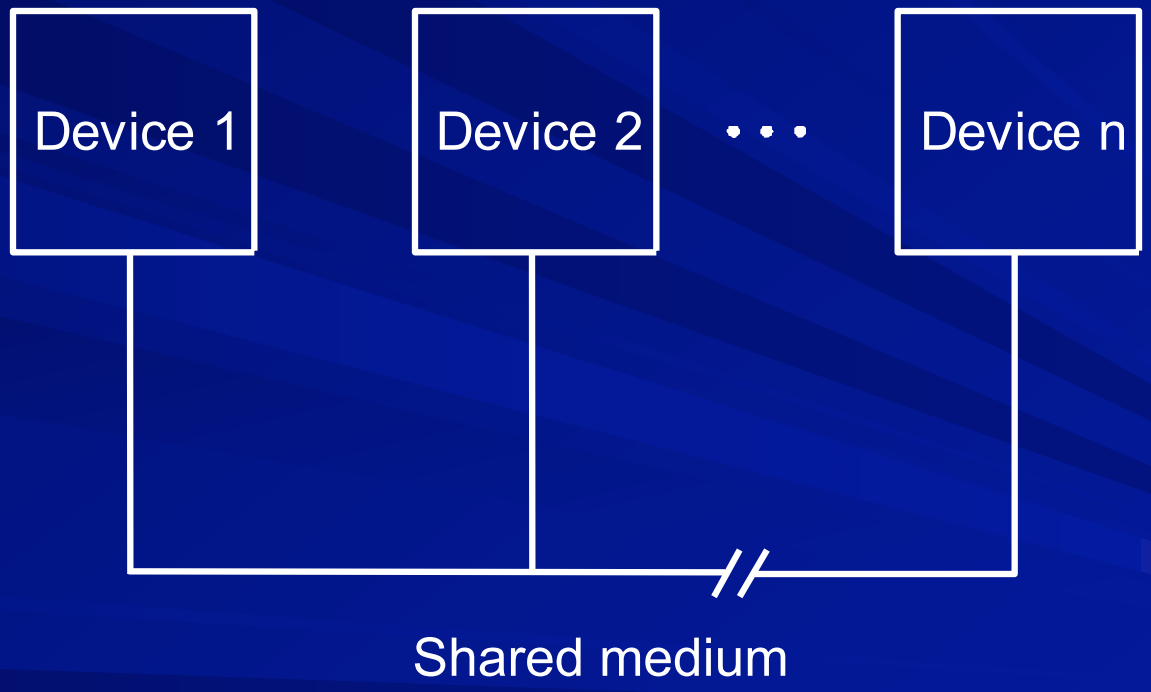
k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0	0	
2	0	1	1	0	0	x	x	
3	1	0	1	0	1	0	1	
4	0	0	1	1	0	x	x	
					1	0	1	

NuBus arbitration example-2

k	ID^k_3	ID^k_2	ID^k_1	ID^k_0	arb^k_3	arb^k_2	arb^k_1	arb^k_0
1	1	0	0	1	1	0	0	x
2	0	1	1	0	0	x	x	x
3	1	0	1	0	1	0	1	0
4	0	0	1	1	0	x	x	x
					1	0	1	0

Arbitration by collision detection

Example: Ethernet

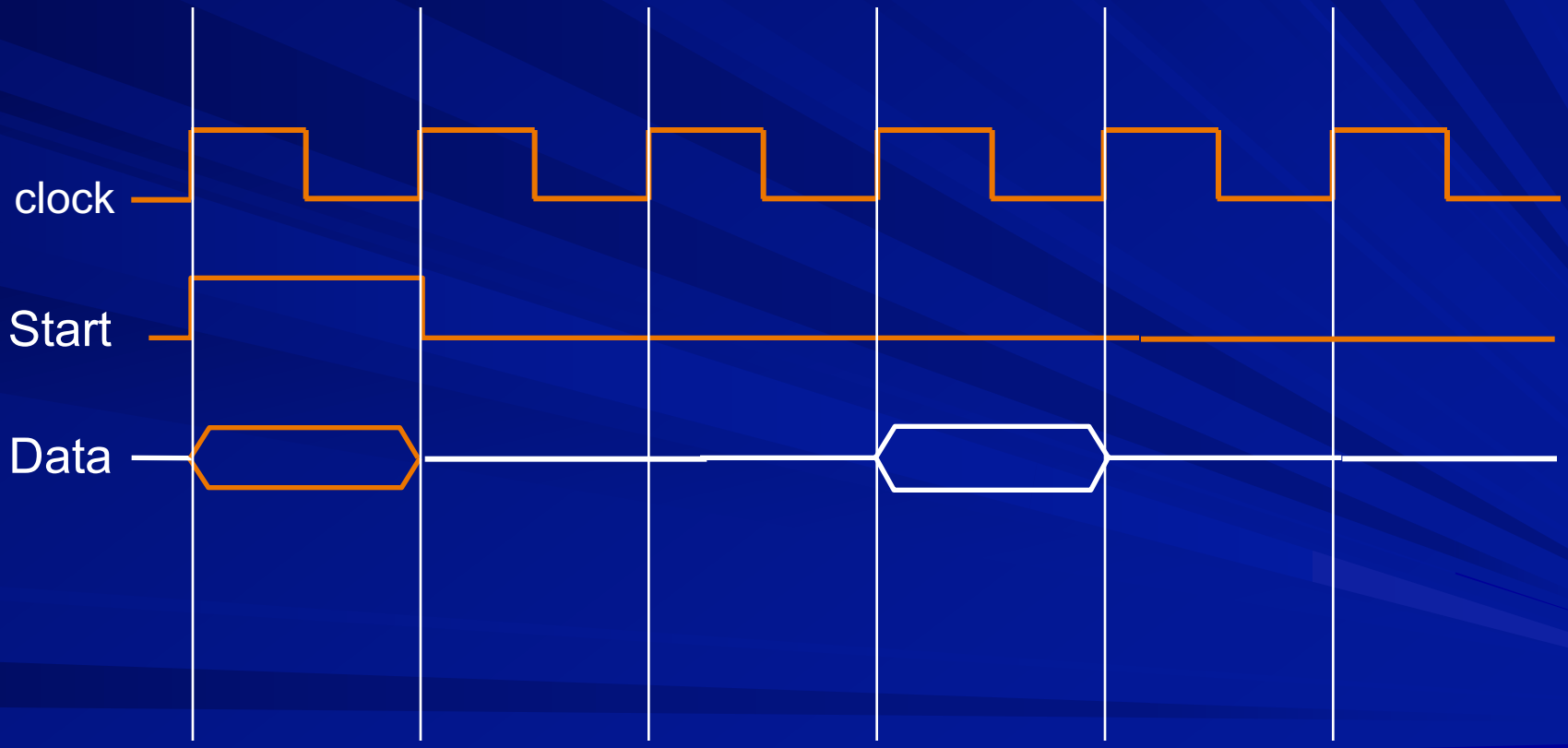


if collision then try again after random interval

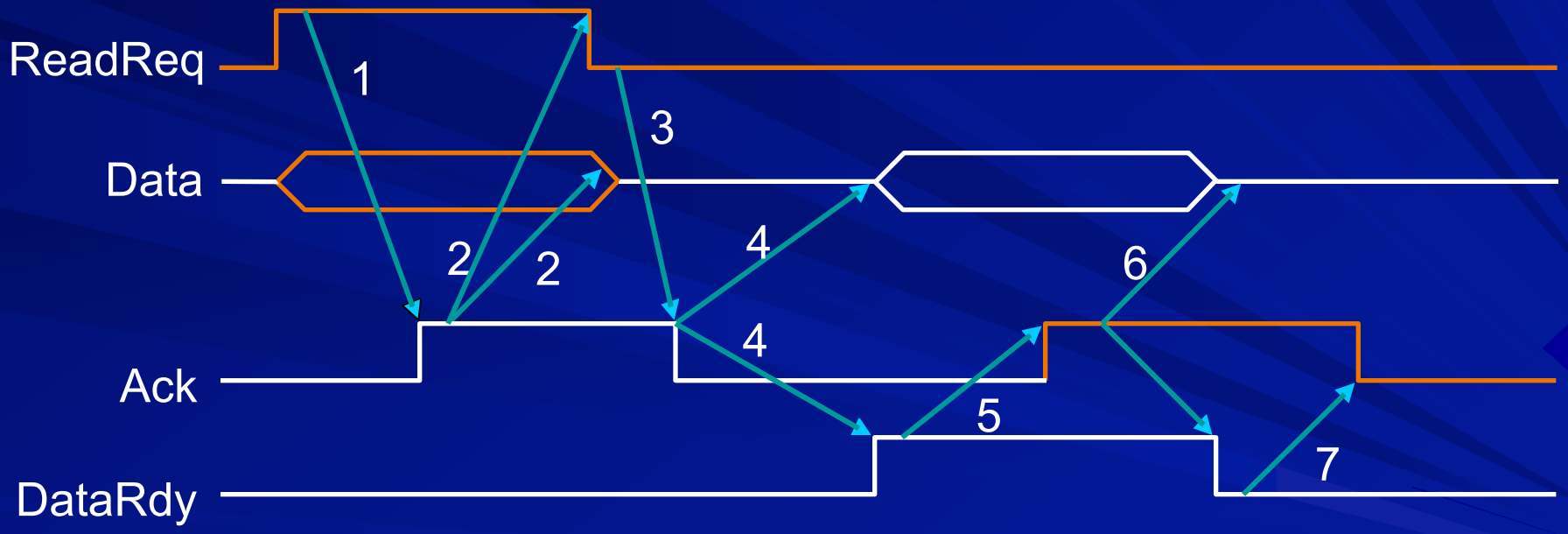
Increasing the bus bandwidth

- increase bus width
- separate data and address lines
- pipelining
- multiple word blocks
- synchronous
- split transaction

Synchronous transfer



Asynchronous handshaking



THANKS

COL216
Computer Architecture

Input/Output – 5

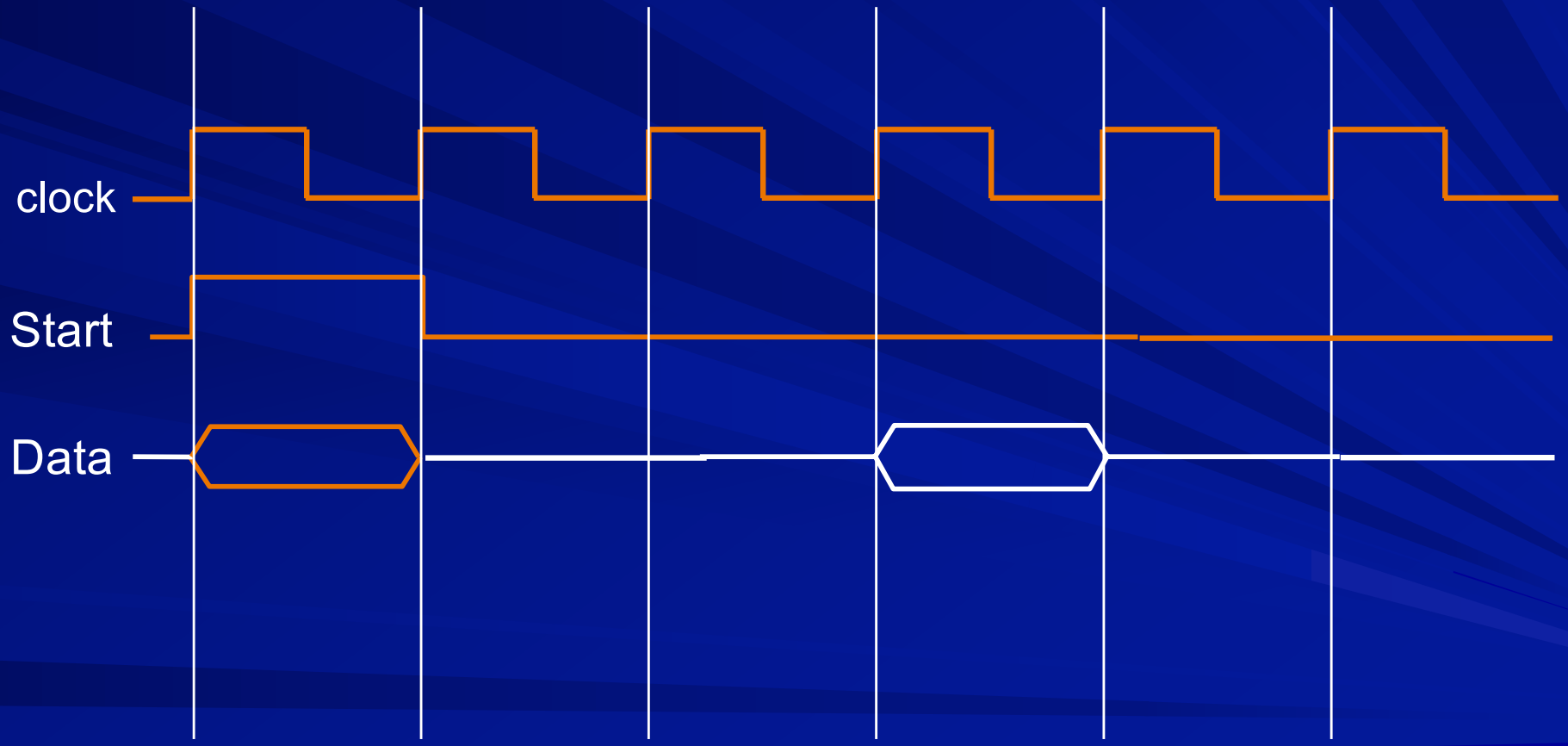
Parallel and Serial Buses

28th March 2022

Increasing the bus bandwidth

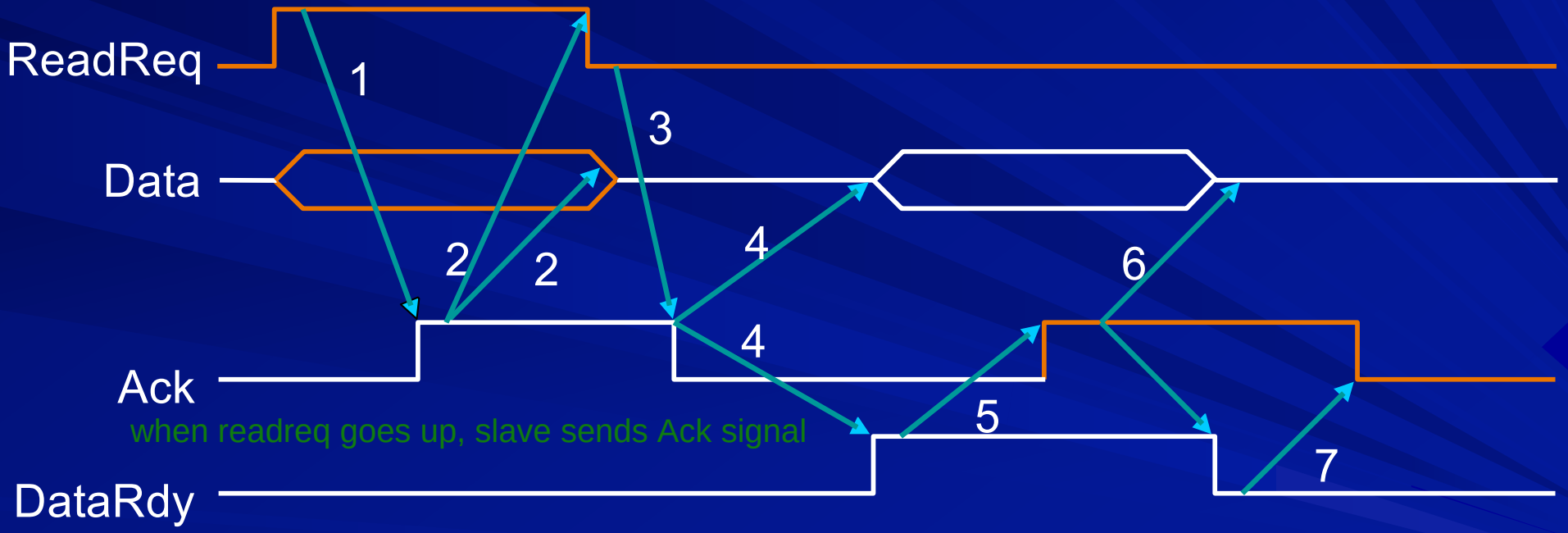
- increase bus width
- separate data and address lines
- pipelining
- multiple word blocks
- synchronous
- split transaction

Synchronous transfer



Asynchronous handshaking

level triggered

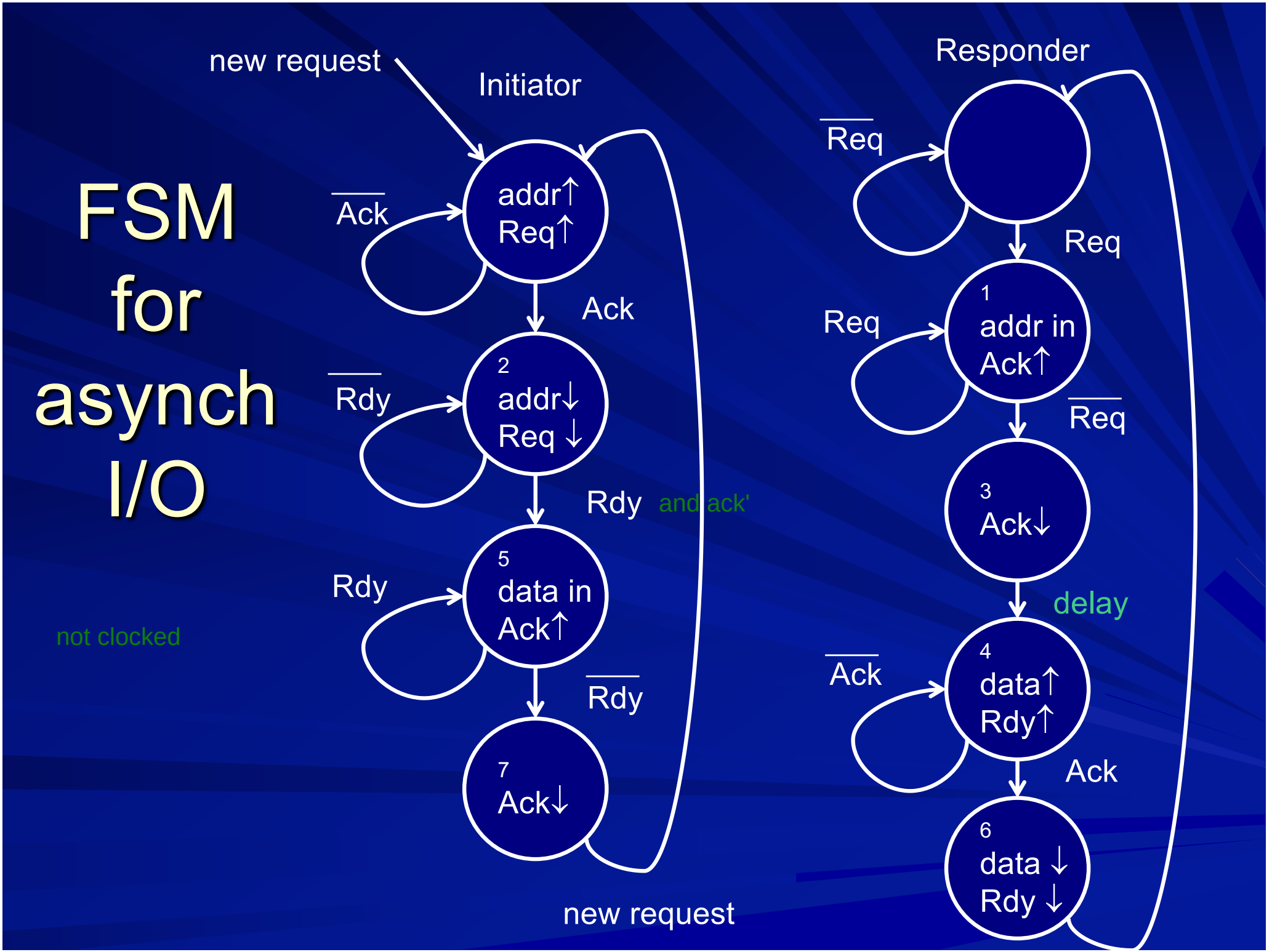


when readreq goes up, slave sends Ack signal

after readreq goes down, ack goes down

data ready signal sent by user when data is ready

FSM for asynch I/O



not clocked

Split transactions

in between, we could release the bus and then a different transaction could begin

Parallel vs Serial

- Parallel
  - multiple wires carry bits in parallel : 8-bit, 16-bit, 32-bit, 64-bit . . .
  - address, read data, write data may use same wires or separate wires
- Serial
  - only one bit at a time
  - read data, write data may use same wires or separate wires, no separate address wires

Series/parallel bus examples

Parallel

- ISA (Industry Standard Architecture)
- EISA (Extended ISA)
- VLB (VESA (Video Electronics Standards Association) Local Bus)
- PCI (Peripheral Component Interconnect)
- AGP (Accelerated Graphics Port)

Serial

- USB (Universal Serial Bus)
- Fire wire
- Fibre channel
- PCIe (Peripheral Component Interconnect express)
- SATA (Serial Advanced Technology Attachment)

Why serial buses?

- Serial buses are less expensive
- At high speed, keeping all the bits synchronized in parallel buses is very difficult
- Longer the signals travel, more skew is likely among the bits of parallel buses

Move from parallel to serial

PCI Bus

- PCI 1.0: 33MHz, 32-bit, peak BW 133MB/s
- PCI 2.2: 66 MHz, 64-bit, BW 533 MB/s
- PCI-X: 133MHz, 64-bit, BW 1066 MB/s
- PCI-X 2.0: 266MHz, BW 2133 MB/s

PCI-Express (serial bus), 1-16 lanes

- PCIe 1.0: 250 MB/s per lane
- PCIe 2.0: 500 MB/s per lane
- PCIe 3.0: 985 MB/s per lane

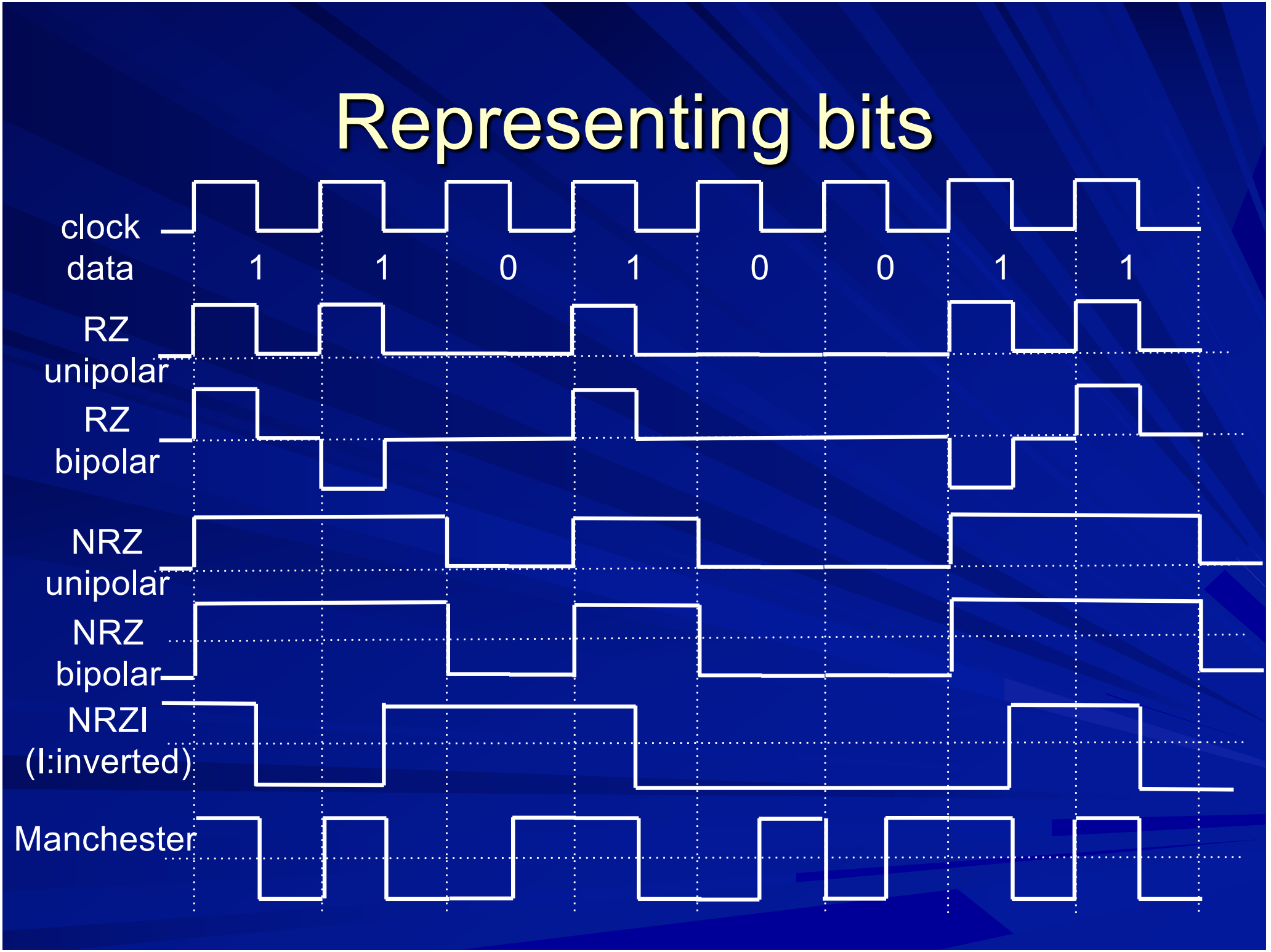
Move from parallel to serial

Parallel ATA
33 MB/s =>
66 MB/s =>
100 MB/s =>
133 MB/s
Serial ATA
150 MB/s =>
300 MB/s =>
600 MB/s =>
1969 MB/s

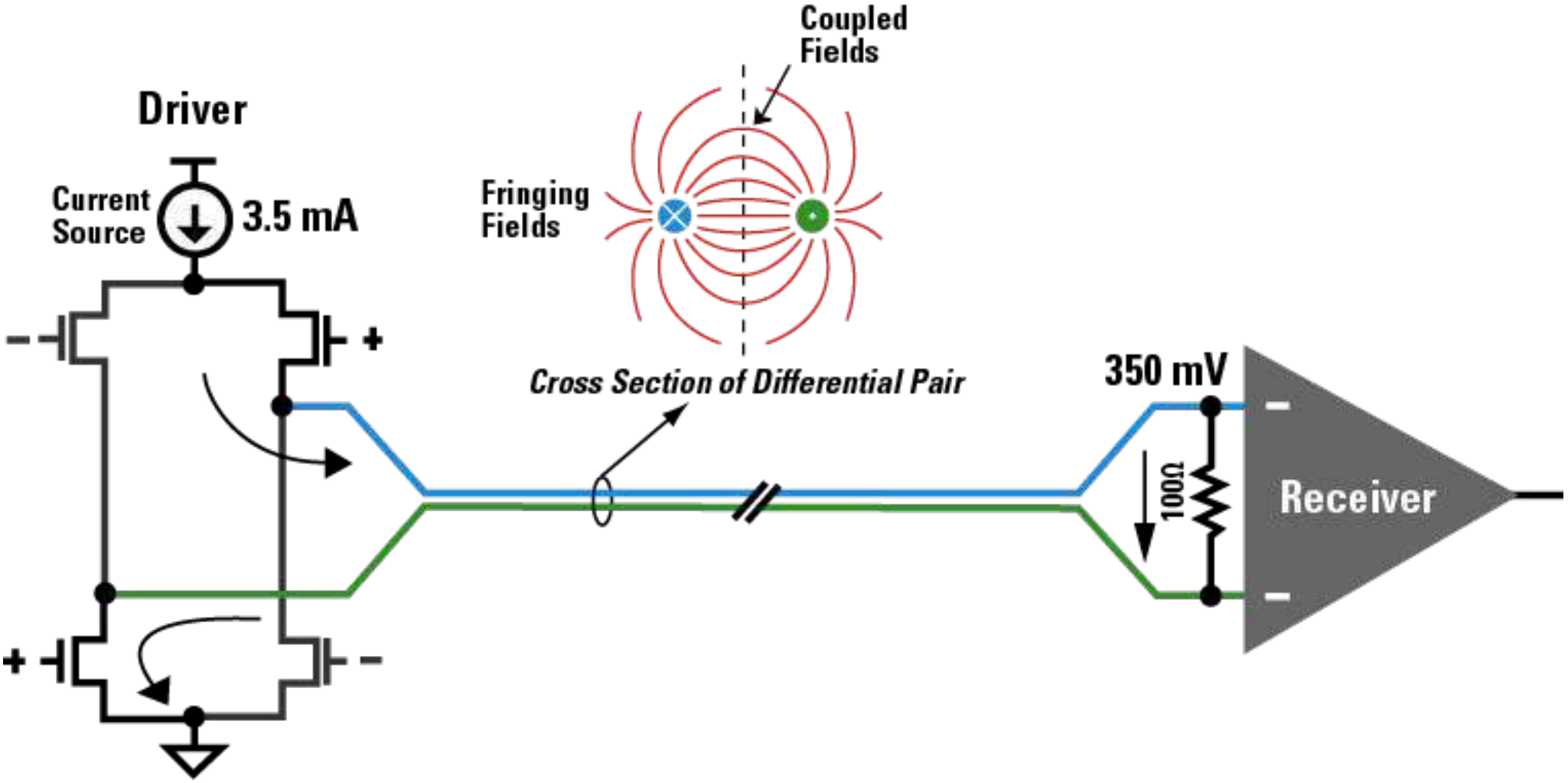
Representing bits

- Unipolar / bipolar
- Active high / active low
- Return to zero (RZ) / Non return to zero (NRZ)
- Encode using levels or transitions
- One wire / two wire (differential)

Representing bits



Low voltage differential signalling (LVDS)

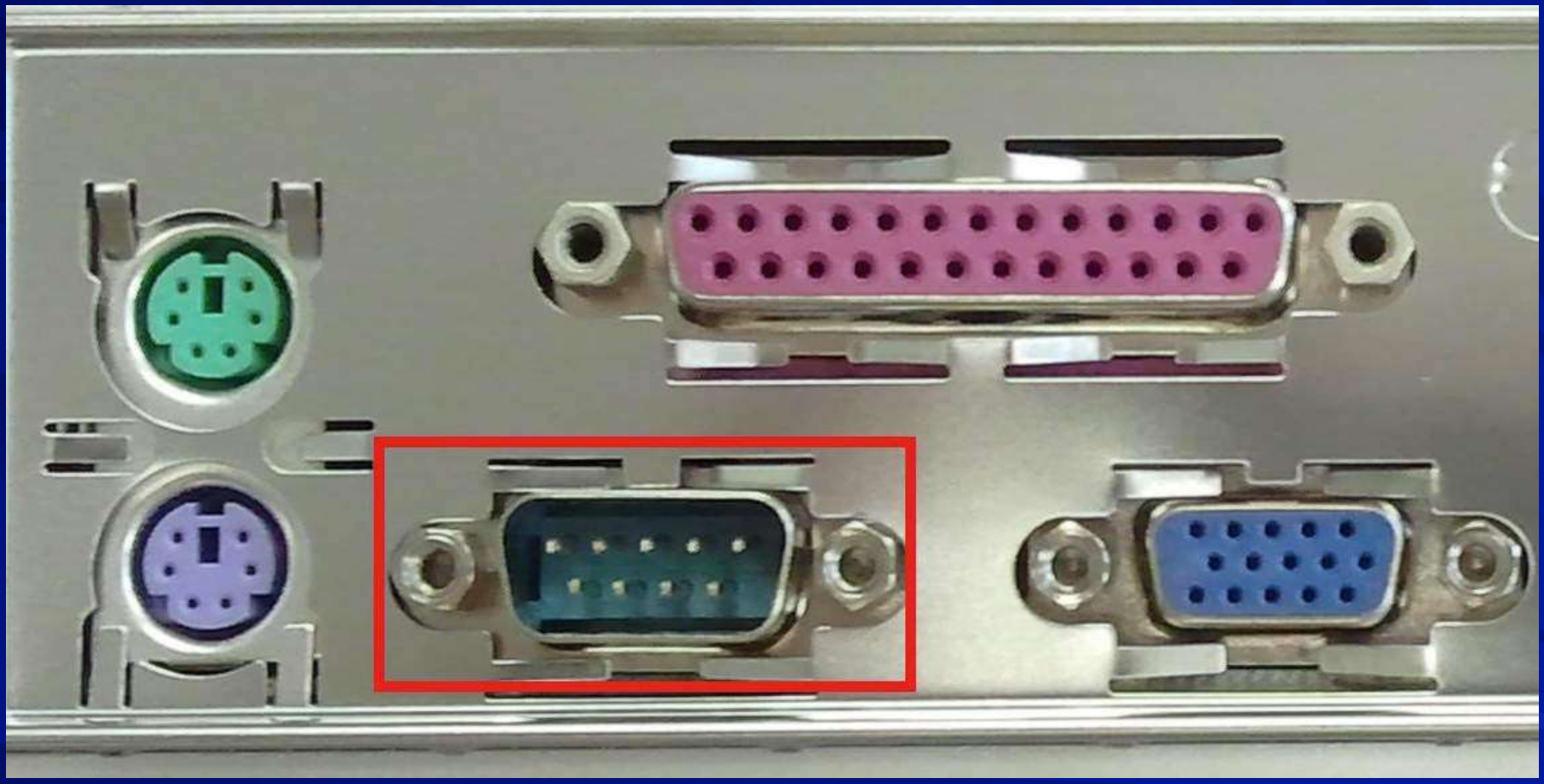


from wikipedia

Timing in serial transfer

- Synchronous
  - shared clock
  - same frequency and phase
- Mesochronous
  - transfer clock signal, separately or with data
  - same frequency but not phase
- Plesiochronous
  - locally generated clock
  - nearly same frequency, changing phase

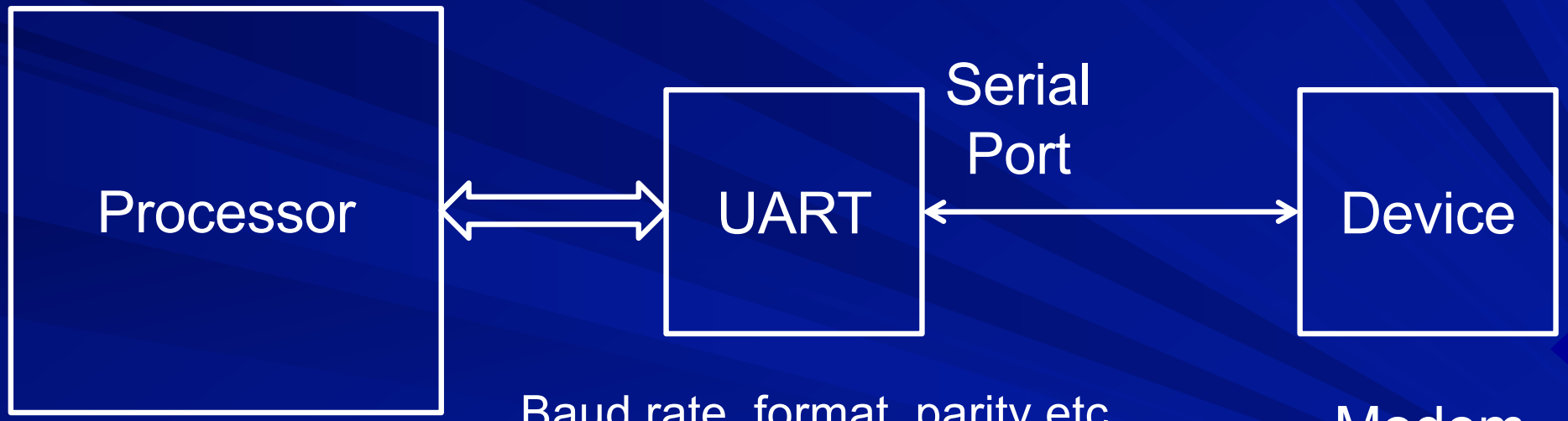
Serial port / COM port



RS232 has been a common standard
75 to 115.2 K bits/s

RS232 Interface Standard

UART
(Universal Asynchronous Receiver Transmitter)



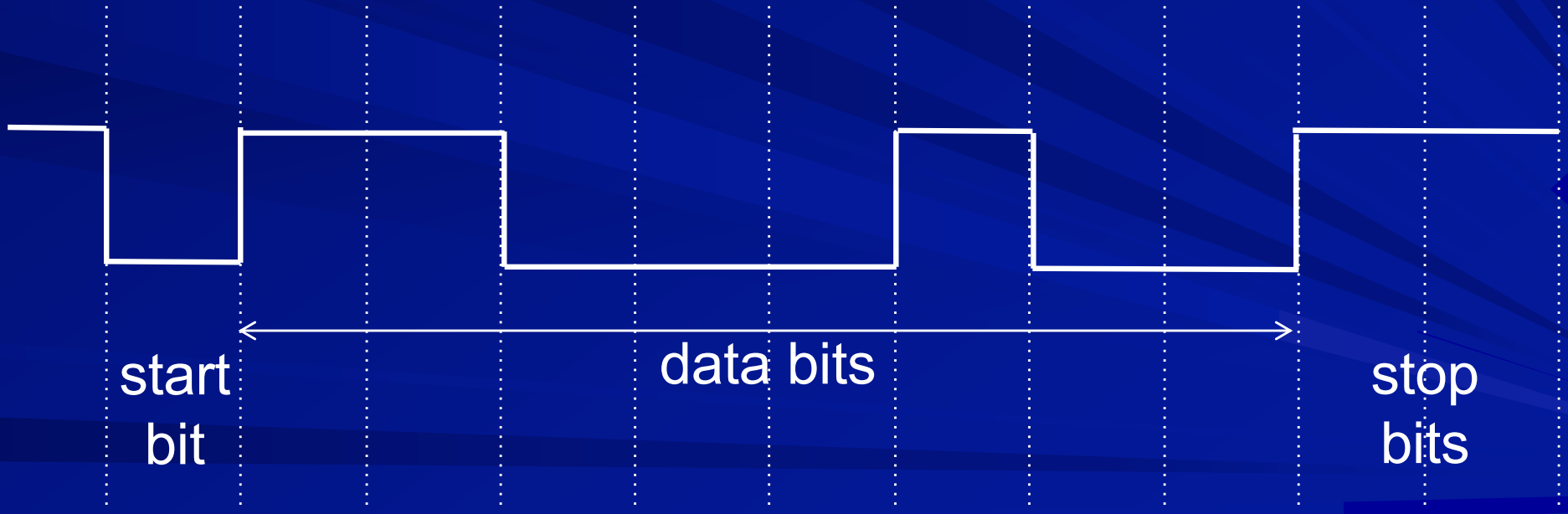
Baud rate, format, parity etc can be programmed

Modem, Printer, Terminal etc

UART

Serial port: data waveform

- Plesiochronous
  - Synchronous at bit level
  - Asynchronous at frame level

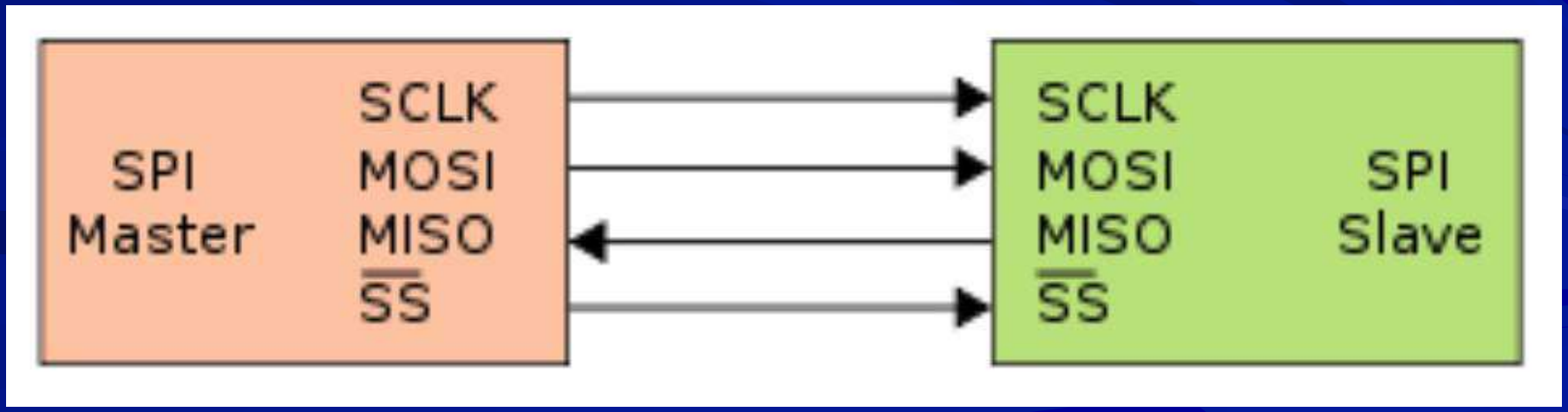


Data Framing

Data bits : 5 to 8
Parity : even / odd / none
Start bits : 1
Stop bits : 1 or 2
Data rate : 300 / 600 / 1200 / 2400 / 4800 / 9600 . . .
bits/sec (bauds)

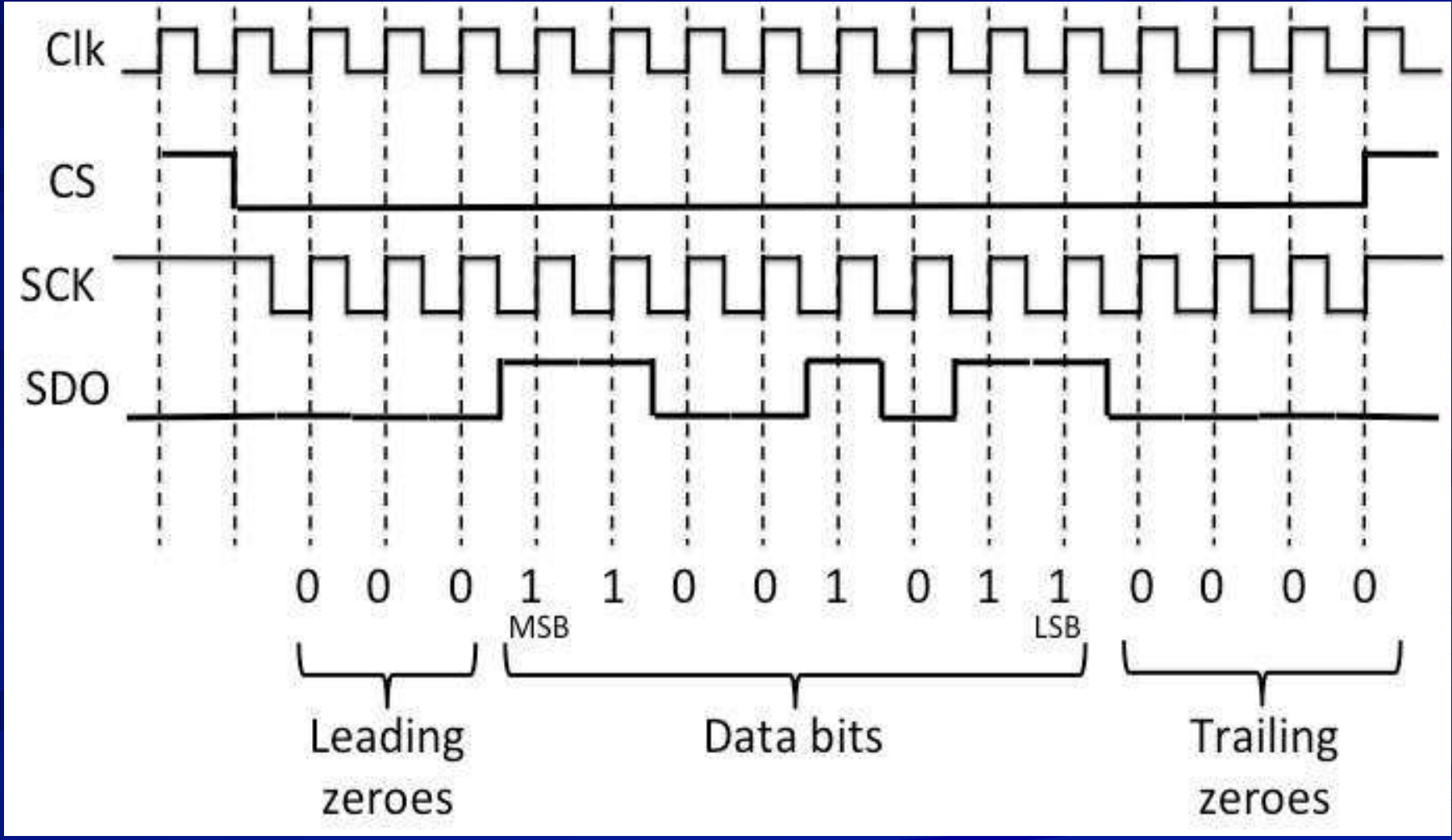
SPI : A simple serial interface

- SPI (Serial Peripheral Interface) is a synchronous serial protocol.
- Used for connecting some Pmods with BASYS-3.



Ref: Wikipedia

SPI Signal Waveforms



COL216
Computer Architecture

Input/Output – 5

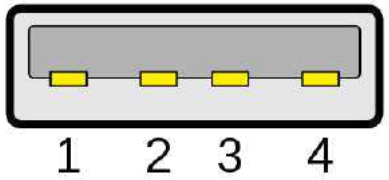
USB

28th March 2022

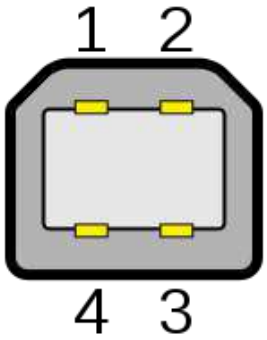
USB : Universal Serial Bus

- Introduced in mid-90's
- Used for a huge variety of devices
- Replaced earlier serial and parallel ports
- Much higher speed than RS232 etc
- Packet oriented, not character oriented
- Clock is recovered from data
- Differential signalling (D+ and D-)
- Upto 127 devices through hubs, each with up to 16 IN and 16 OUT end points

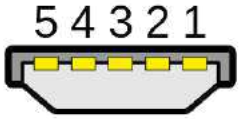
USB connectors



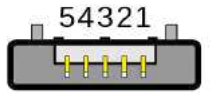
Type A



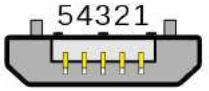
Type B



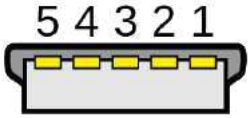
Mini-A



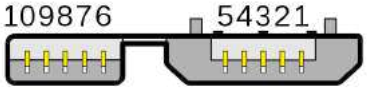
Micro-A



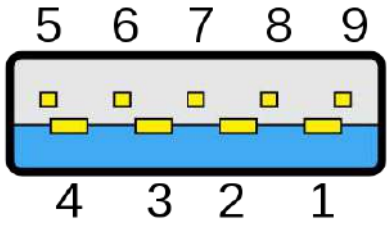
Micro-B



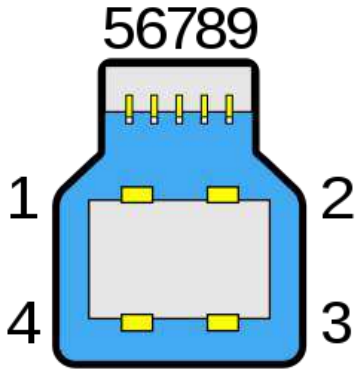
Mini-B



Micro-B SuperSpeed



Type A SuperSpeed



Type B SuperSpeed

Type-C

source : wikipedia

USB speed, levels

- USB 1.0 Low Speed 1.5 Mb/s
- USB 1.0 Full Speed 12 Mb/s
- USB 2.0 High Speed 480 Mb/s
- USB 3.0 Super Speed 5 Gb/s
- USB 3.1 Super Speed+ 10 Gb/s
- USB 3.2 Super Speed+ 20 Gb/s
- USB 4 forthcoming

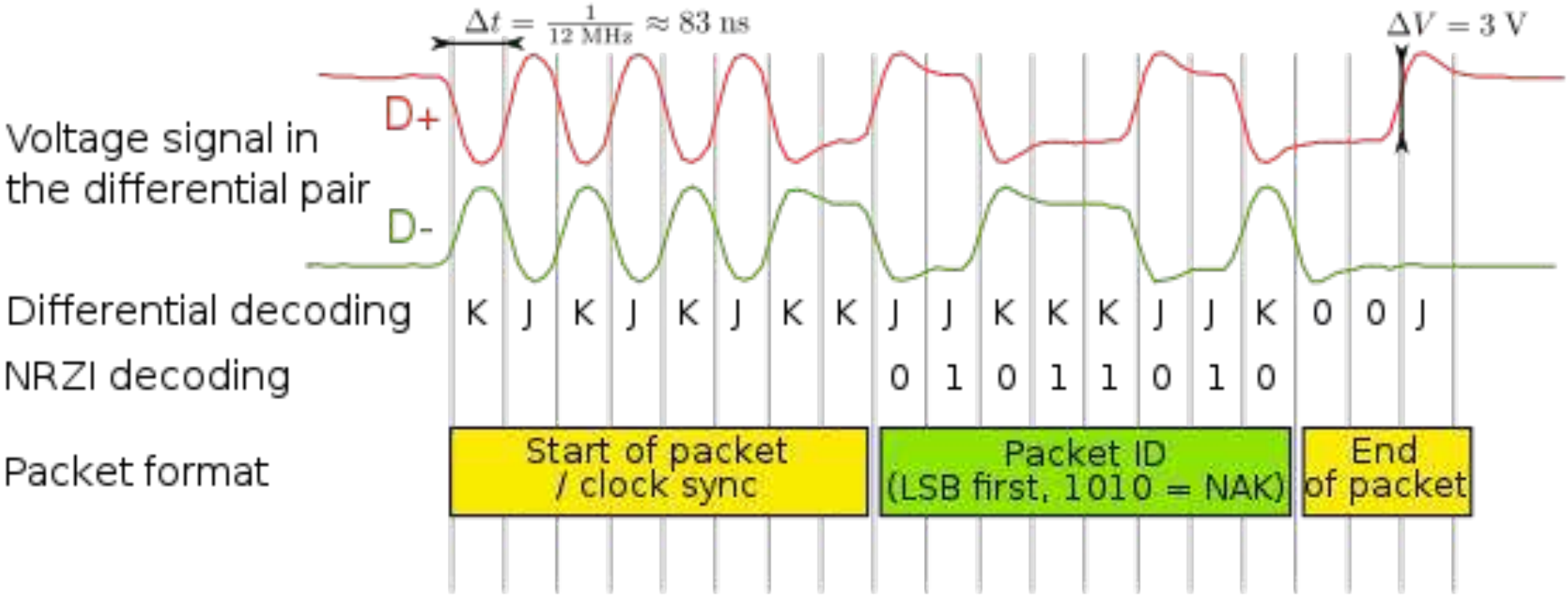
LS, FS : 0.0 to 0.3 V 2.8 to 3.6 V

HS, SS : -10 to 10 mV 360 to 440mV

What D+,D- patterns can indicate

- Device found detached
- Device wakes up host
- Host and device idling
- Start of packet
- End of Packet
- Reset device to a known state
- Power down device
- Host wakes up device
- Host wants device to stay awake
- Device wants to wake up

USB NAK packet



source : wikipedia

Packet IDs

- Token packets
  - SPLIT: Split transaction
  - PING
  - OUT, IN: Address for data transfer
  - SOF: start of frame marker
  - SETUP: Device management
- Handshake packets
  - ACK: Packet accepted
  - NAK: Packet not accepted, retransmit
  - NYET: Data not ready yet
  - STALL: Transfer impossible, do recovery
- DATA0, DATA1 etc: data packet

COL216
Computer Architecture

Input/Output – 5

Secondary Storage

28th March 2022

Hard Disk Drive Example

Capacity:	14TB
Size:	3.5”
RPM:	7200
Data rate:	250 MB/s
Disks:	8 (16 heads)
Sector:	512B (logical)
	4096B (physical)
Density:	2426 Kb/in
	436 Ktracks/in
Cache:	256 MB

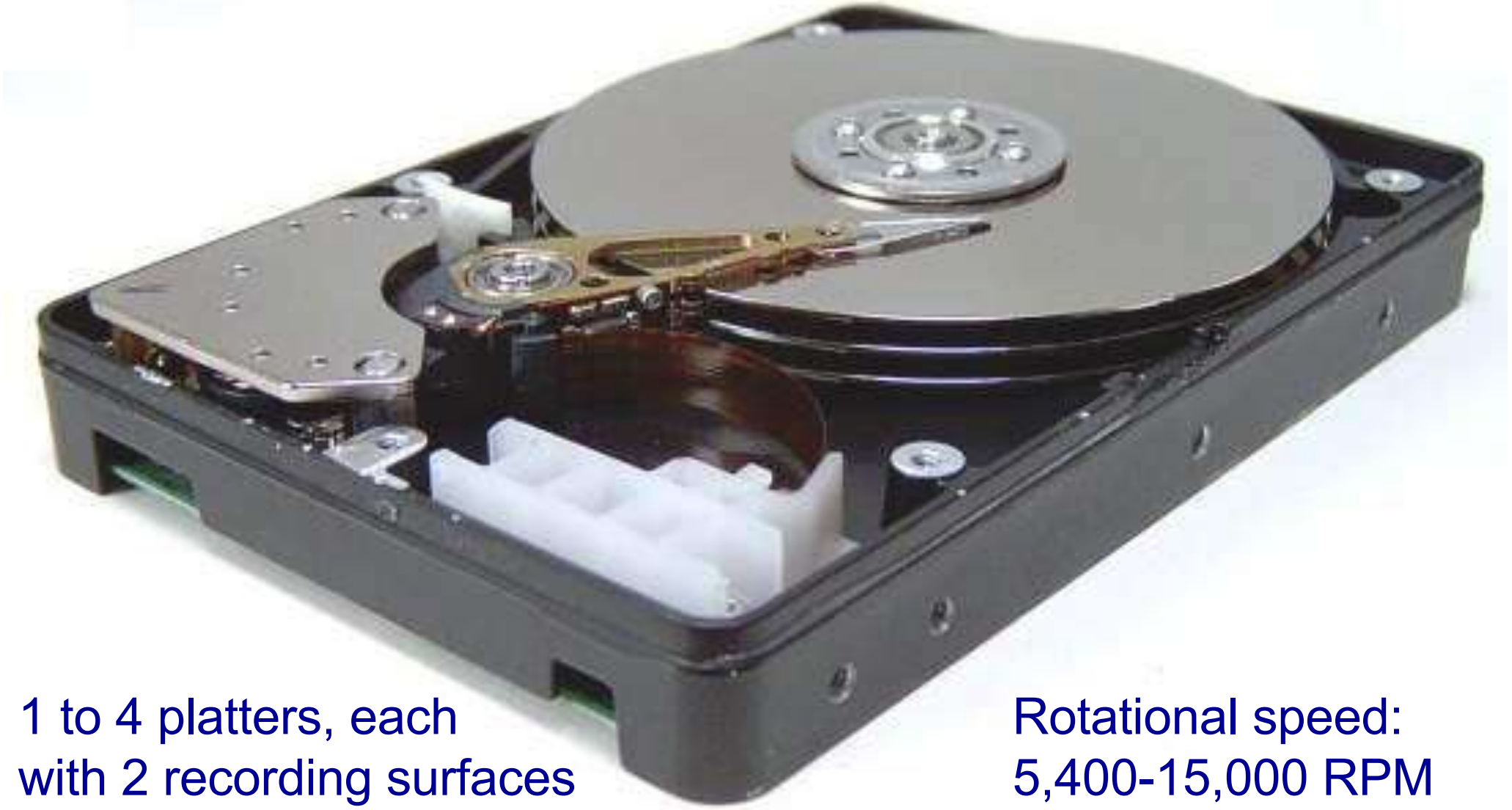
HDD comparison – then and now

Parameter	1957	2018-20	Improvement
Capacity	3.75 MB	18 TB	4.8×10^6
Volume	1.9 m^3	34 cm^3	5.6×10^4
Weight	910 kg	62 g	1.5×10^4
Access Time	600 ms	2.5 – 10 ms	2×10^2
Price	USD 9.2 / KB	USD 24 / TB	3×10^6
Density	2K b / in^2	1.3 TB / in^2	6.5×10^8
MTBF	2000 hrs	285 yrs	1.25×10^3

mean time between failures

Source: Wikipedia

Hard Disk Drive



1 to 4 platters, each
with 2 recording surfaces

Rotational speed:
5,400-15,000 RPM

https://www.pctechguide.com/wp-content/uploads/2011/09/31harddrive.jpg

Hard disk drive close-up

Tracks:
10-50 K
Sector/track
100-500
Bytes/sector
512
Seek time:

- Track to track
  .5-2 ms
- Average
  ~10 ms

Disk Latency

Disk Latency =

Seek time + Rotational Latency + data transfer time + controller overhead

Average rotational latency =

time for ½ rotation = 15000 rpm

(.5 x 60 x 1000 / rpm) msec

Transfer rate = 10^1 – 10^2 MB/sec

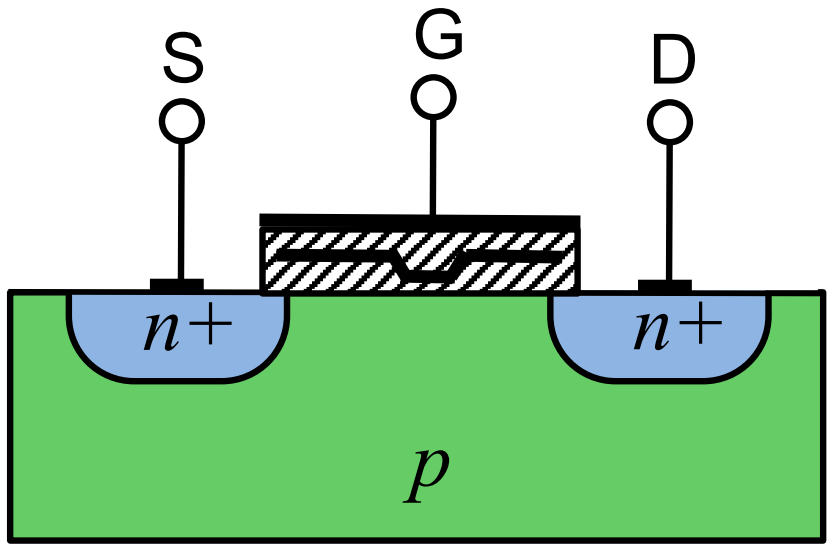
Bulk storage without moving parts

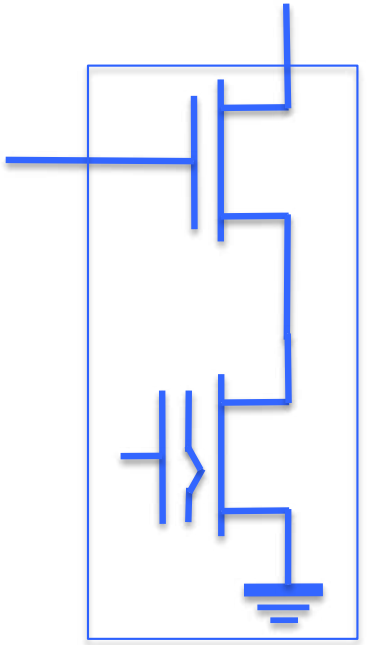
- Flash: semiconductor memory like SRAM, DRAM, but non-volatile like magnetic disks
- Latency is 100–1000 times less than disk
- Smaller, lighter, more power efficient, and more shock resistant than disk
- Has caught up with HDD in capacity, but more expensive
- SSD wears out faster than HDD

Flash Memory

- Semiconductor memory made up of “Floating gate” transistors – these store information in a non-volatile manner
- Reading is fast, but writing is slow as it requires large current
- Writing involves one way state change (e.g. 0 to 1), therefore needs erasing first (make all bits 0 first)
- Block erase, unlike byte erase of EEPROM

Floating gate transistor





Programming: pass high current through channel
Electrons are trapped on the floating gate

Before trapping: like a normal NMOS transistor
After trapping: transistor remains OFF

NOR and NAND flash

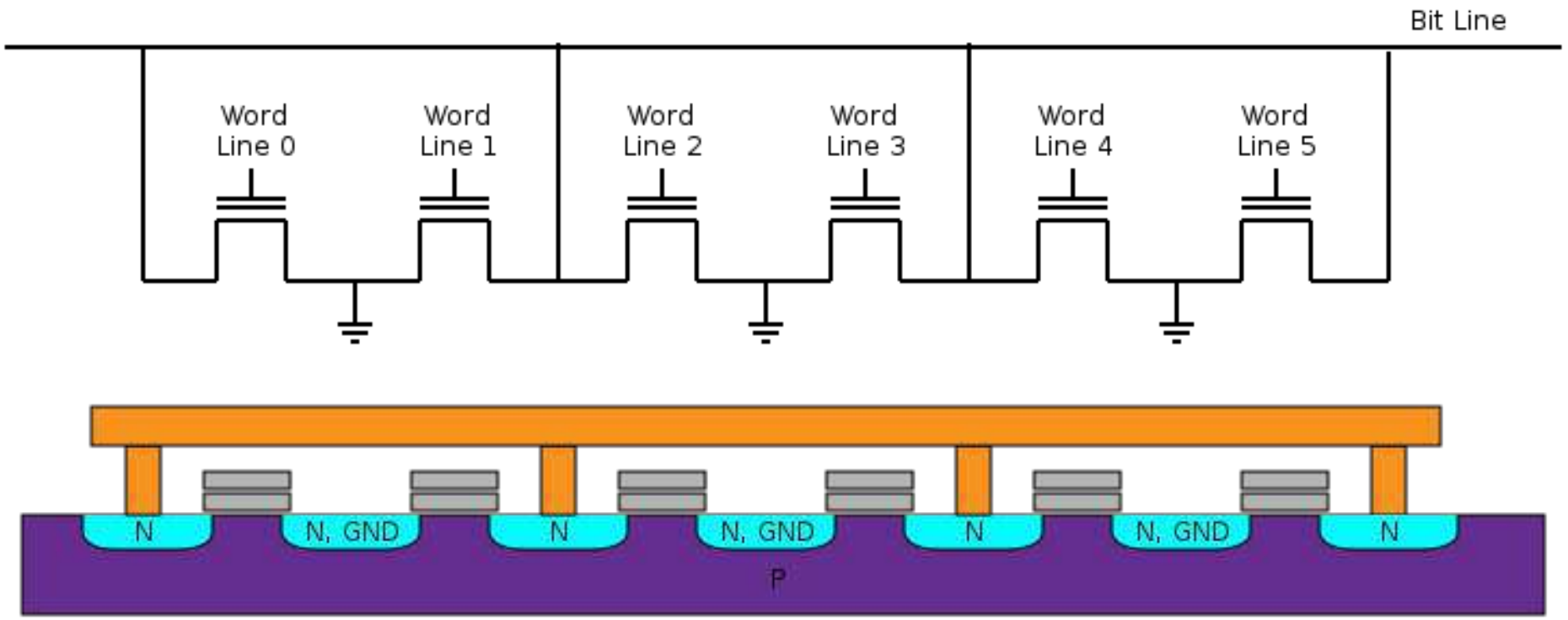
NOR flash

- Structure like NOR gate
- Random access
- Lower density
- More expensive
- Used for BIOS memory

NAND flash

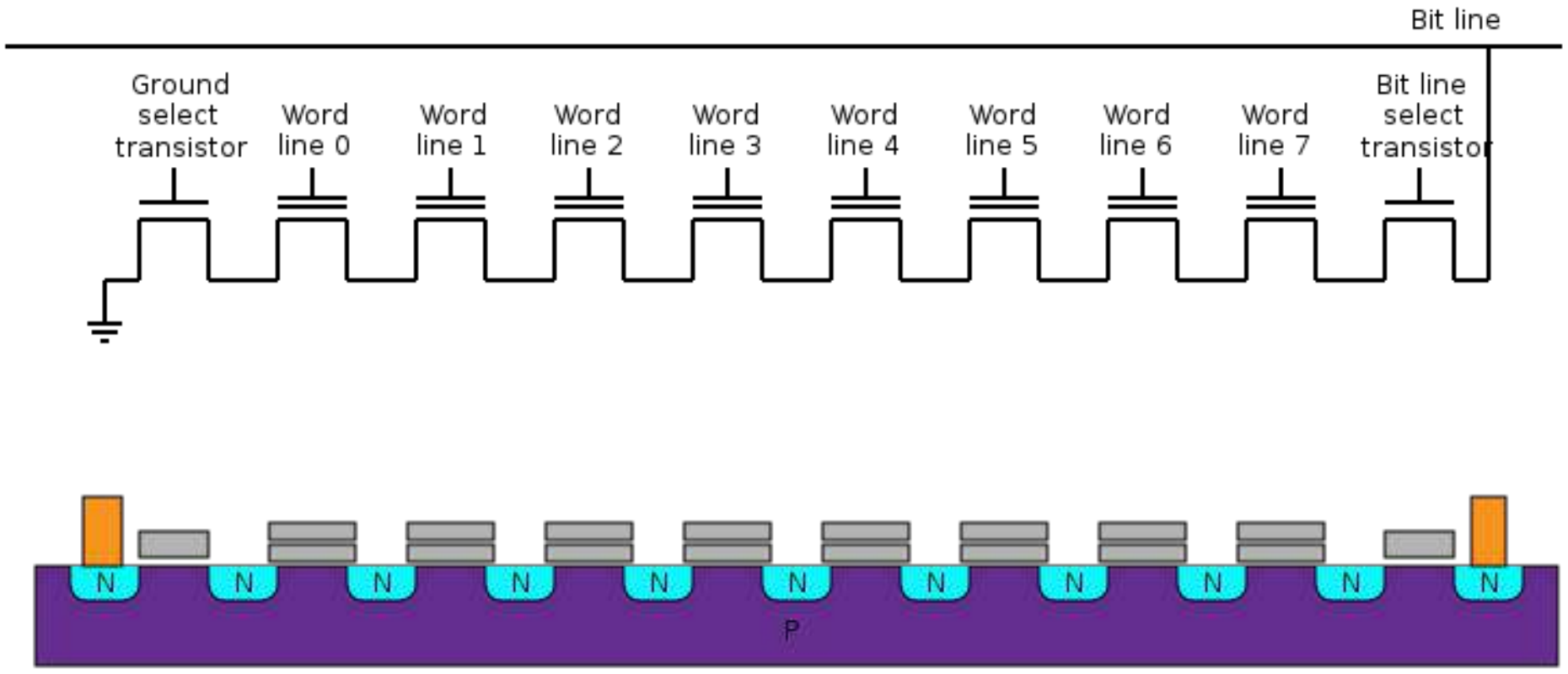
- Structure like NAND gate
- Random plus sequential access
- Higher density
- Less expensive
- Used for pen drive, SD card, SSD
- 32 Mb on BASYS 3

NOR Flash



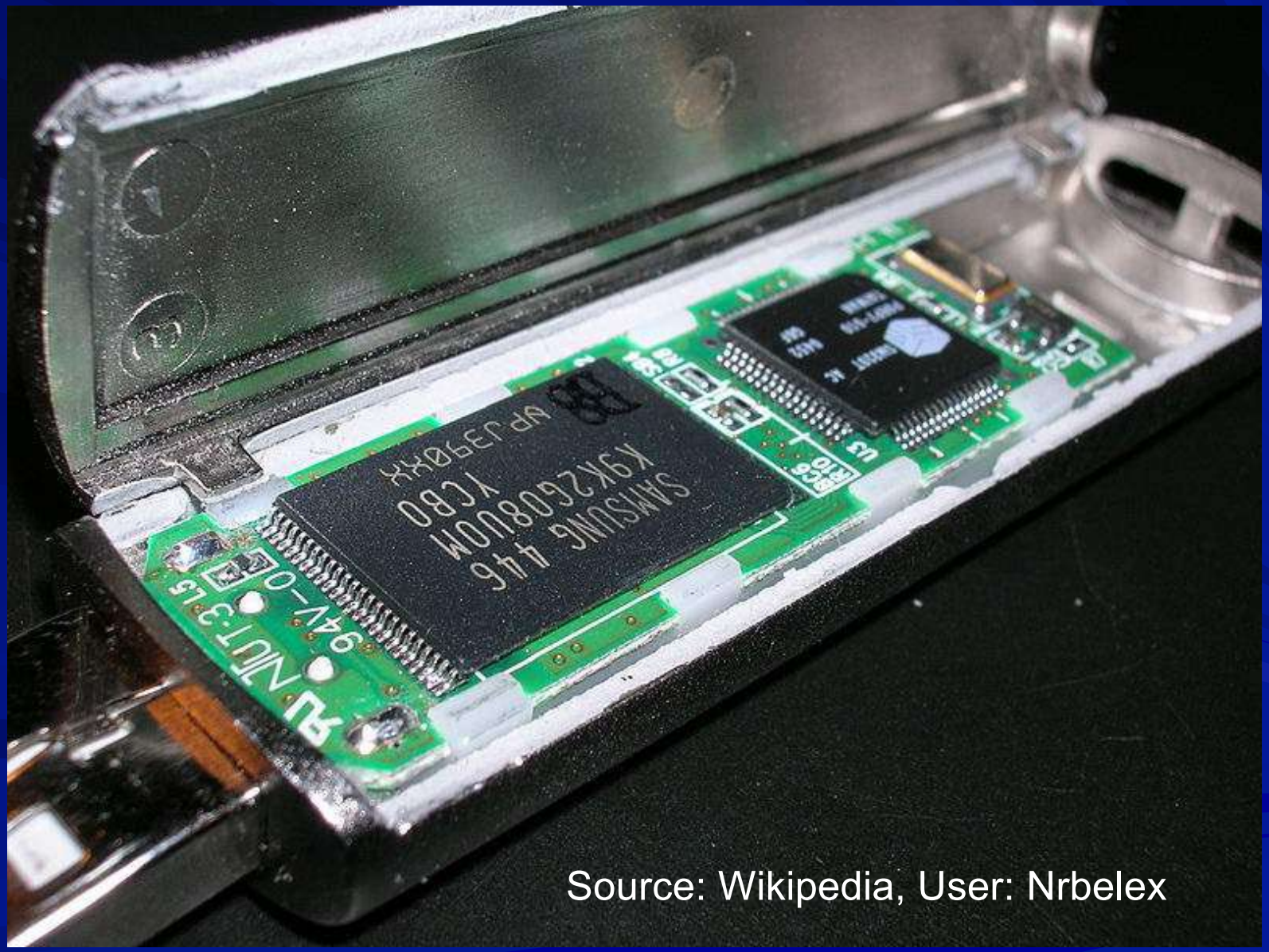
Source: Wikipedia, Author: Cyferz

NAND Flash



Source: Wikipedia, Author: Cyferz

A USB Flash Drive



Source: Wikipedia, User: Nrbelex

COL216
Computer Architecture

Input/Output – 5

I/O Performance

28th March 2022

I/O Performance

- It often gets neglected
- Suppose a benchmark executes in 100 sec
  - CPU 90%
  - I/O 10%
- CPU performance improves by 50% every year for 5 years
  - $90 \rightarrow 60 \rightarrow 40 \rightarrow 27 \rightarrow 18 \rightarrow 12$ (total = 7.5 times)
- I/O performance unchanged
- Overall performance
  - $100 \rightarrow 70 \rightarrow 50 \rightarrow 37 \rightarrow 28 \rightarrow 22$ (total = 4.5 times)
- If program is I/O bound, improvement is even smaller

I/O performance definitions

- Throughput
  - Amount of data transfer in unit time (KB/s, MB/s)
  - Number of I/O operations in unit time
- Response time
  - Relevant in interactive environment, embedded systems
- Both important in some cases

Examples

- Supercomputers
  - data throughput (KB or MB/s), more output than input
- Transaction processing
  - transactions per sec, response time
- File server
  - file operations per sec

Discrepancy in units

- Memory size
  - 1 KB = 1024 B (2^{10})
  - 1 MB = 1024 KB (2^{10}) = 1048576 B (2^{10})
- I/O rate
  - 1 KB/s = 1000 B/s (10^3)
  - 1 MB/s = 1000 KB/s (10^3) = 1000000 B/s (10^6)

I/O system performance example

Given :

- Server configuration
- I/O intensive application characteristics

Required :

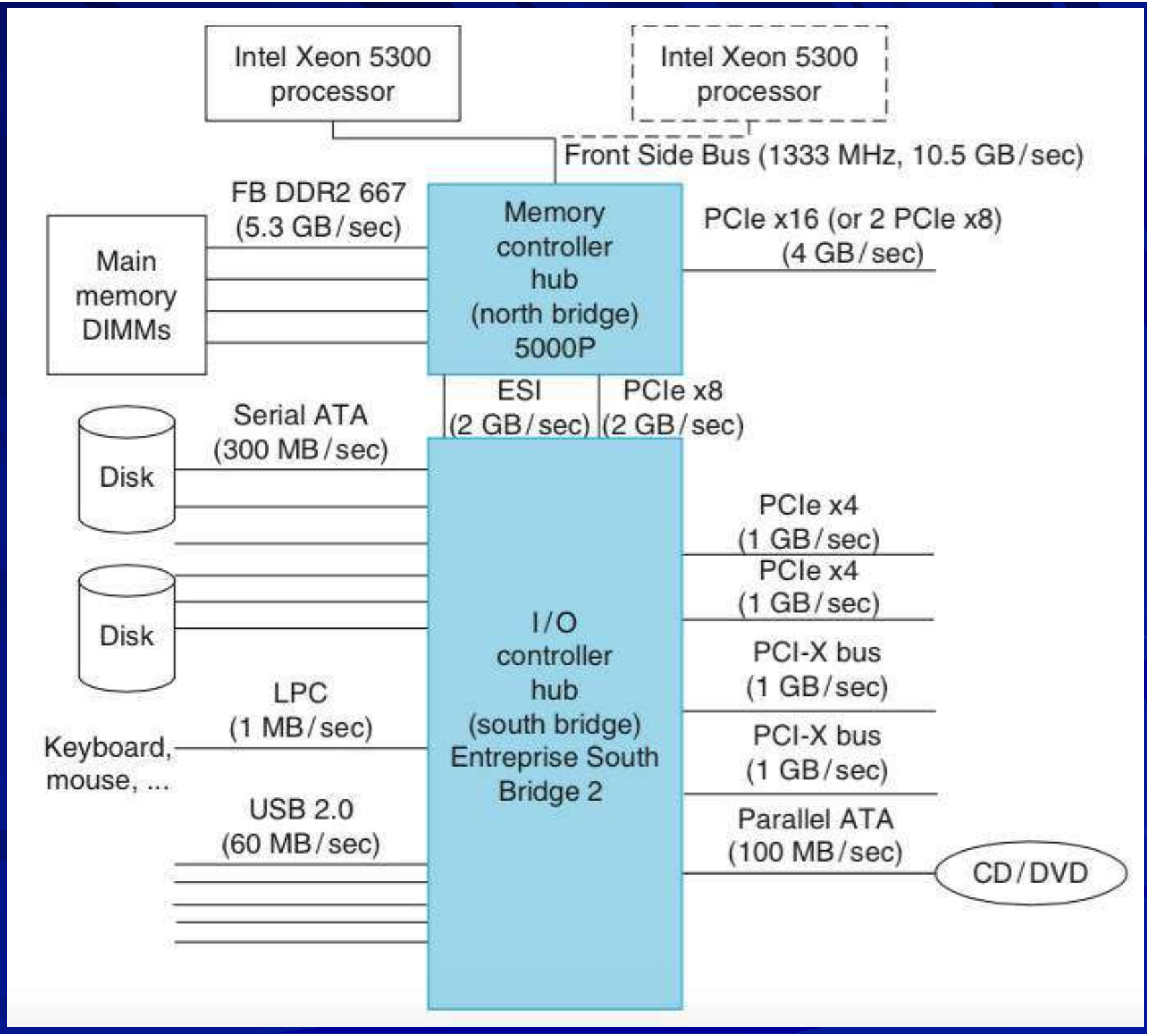
- Maximum sustainable performance
  (in terms of rate of I/O operations)

I/O example: server configuration

Sun Fire x4150 server

- 8 processors, across two sockets (Xeon 5345)
- 64 GB DRAM (DDR2 667), across 16 FBDIMMs
- 8 HDDs, 2.5 inch, 15,000 RPM 73 GB SAS
- 4 Ethernet ports 10/100/1000 Gbps
- 3 PCI Express x8 ports
- 4 external and 1 internal USB 2.0 ports

Server Architecture



I/O example: application characteristics

- User program uses 200,000 instructions per I/O operation
- Operating system averages 100,000 instructions per I/O operation
- Workload consists of 64 KB reads
- Each processor sustains 1 billion instructions per second

I/O example: requirement

- Find the maximum sustainable I/O rate for a fully loaded Sun Fire x4150
  - for random reads
  - sequential reads
- Assume that the reads can always be done on an idle disk if one exists (i.e., ignore disk conflicts)

200 for user program, 100 for OS

IOPS for processors

Maximum I/O rate of 1 processor = Instruction execution rate / Instructions per I/O

$= 1 \times 10^9 / (200 + 100) \times 10^3 = 3{,}333$ IOPS

One socket has four processors

$\Rightarrow$ 4 x 3,3333 = 13,333 IOPS

Two sockets with eight processors

$\Rightarrow$ 2 x 13,333 = 26,666 IOPS.

IOPS for disks

Time per I/O at disk (for random access)

= seek + rotational time + transfer time

= .725 ms + 2 ms + 64 KB / 112 MB/s

= 3.3 ms

Each disk can do 1000/3.3 = 303 IOPS

8 disks can do 8 x 303 = 2,424 IOPS

For sequential access,

in sequential, we don't bother about seek and rotational time, so we have only transfer time

time = 64 KB / 112 MB/s => 1,750 IOPS

i.e, 14,000 IOPS for 8 disks

IOPS for PCI express

Bandwidth of a PCIe lane = 250 MB/s

Bandwidth of 8 PCIe lanes = 8 x 250 MB/s

= 2,000 MB/s throughput in terms of I/O blocks

IOPS for 8 PCIe lanes = 2,000 MB/s / 64 KB

= 31,250

each I/O operation requires 64KB to be transferred

IOPS for DRAM

memory latency is 667 MHz

Bandwidth of 667 MHz FBDIMM

= 8 x 667 = 5,336 MB/s

IOPS for one FBDIMM = (5,336 / 64) x 1000

= 83,375

Max no. of DIMMs = 16

finally we compare all the IOPS to find the bottleneck

front side bus

FSB limit

FSB peak bandwidth = 10.6 GB/s

Sustained bandwidth = 5.3 GB/s

IOPS for each FSB = $5.3 \times 10^6 / 64 = 81{,}540$

For 2 FSBs = > 2 x 81,540 = 163,080 IOPS

IOPS at a glance

- 8 Processors : 26,667
- 8 Disks : 2,424 (random)
  14,000 (sequential)
  disk read is slowest
- PCIe 8x : 31,250
- 1 FBDIMM : 83,375
- 1 FSB : 81,540

THANKS