

I/O Management Intro

Chapter 5



Learning Outcomes

- A high-level understanding of the properties of a variety of I/O devices.
- An understanding of methods of interacting with I/O devices.
- An appreciation of the trend towards offloading more I/O handling to devices themselves.



I/O Devices

- There exists a large variety of I/O devices:
 - Many of them with different properties
 - They seem to require different interfaces to manipulate and manage them
 - We don't want a new interface for every device
 - Diverse, but similar interfaces leads to code duplication
- Challenge:
 - Uniform and efficient approach to I/O



Categories of I/O Devices (by usage)

- Human interface
 - Used to communicate with the user
 - Printers, Video Display, Keyboard, Mouse
- Machine interface
 - Used to communicate with electronic equipment
 - Disk and tape drives, Sensors, Controllers, Actuators
- Communication
 - Used to communicate with remote devices
 - Ethernet, Modems, Wireless



I/O Device Handling

- Data rate
 - May be differences of several orders of magnitude between the data transfer rates
 - Example: Assume 1000 cycles/byte I/O
 - Keyboard needs 10 KHz processor to keep up
 - Gigabit Ethernet needs 100 GHz processor.....



Sample Data Rates

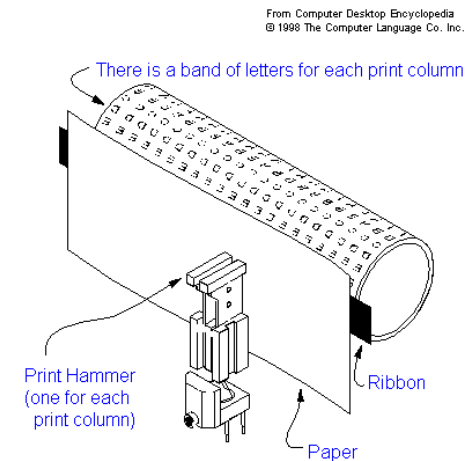
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

USB 3.0 625 MB/s (5 Gb/s)
 Thunderbolt 2.5GB/sec (20 Gb/s)
 PCIe v3.0 x16 16GB/s



I/O Device Handling Considerations

- Complexity of control
- Unit of transfer
 - Data may be transferred as a stream of bytes for a terminal or in larger blocks for a disk
- Data representation
 - Encoding schemes
- Error conditions
 - Devices respond to errors differently
 - `lp0: printer on fire!`
 - Expected error rate also differs

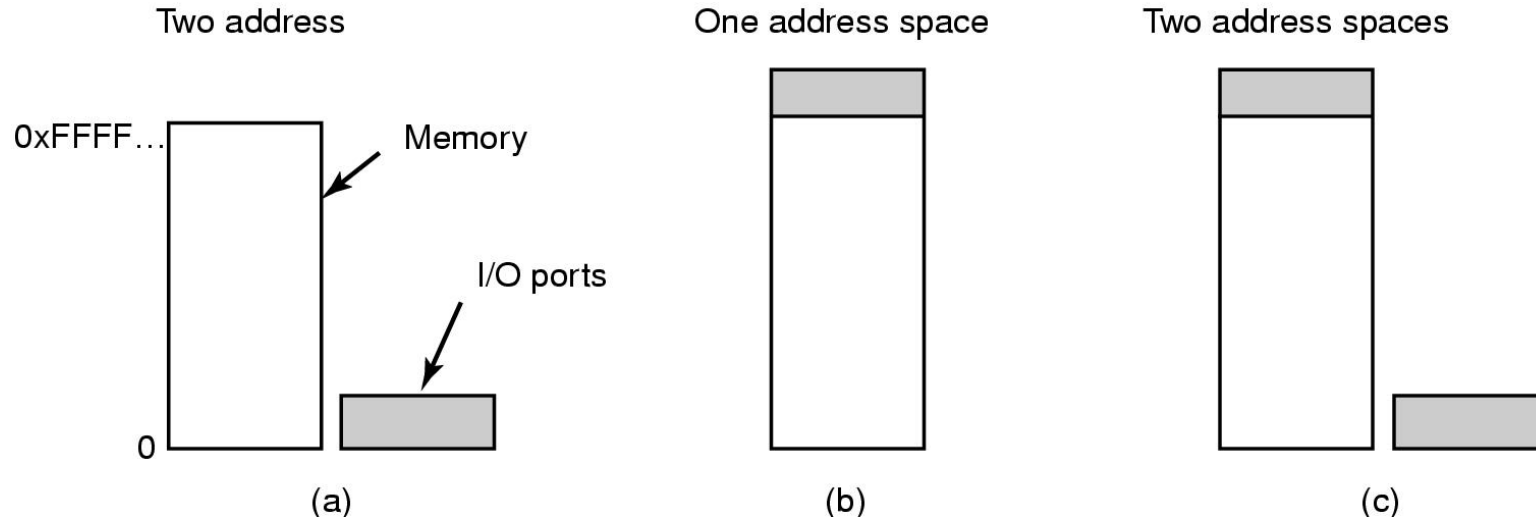


I/O Device Handling Considerations

- Layering
 - Need to be both general and specific, e.g.
 - Devices that are the same, but aren't the same
 - Hard-disk, USB disk, RAM disk
 - Interaction of layers
 - Swap partition and data on same disk
 - Two mice
 - Priority
 - Keyboard, disk, network



Accessing I/O Controllers



a) **Separate I/O and memory space**

- I/O controller registers appear as I/O ports
- Accessed with special I/O instructions

b) **Memory-mapped I/O**

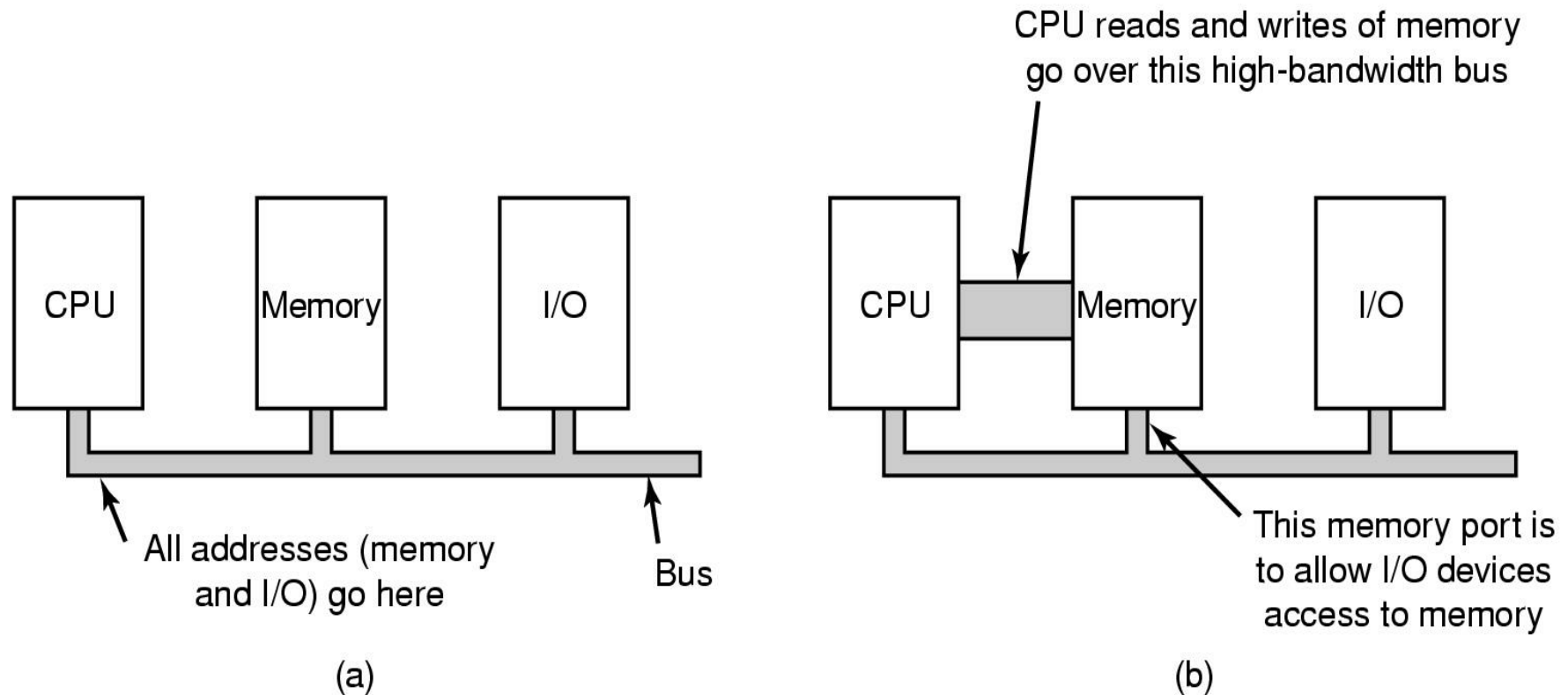
- Controller registers appear as memory
- Use normal load/store instructions to access

c) **Hybrid**

- x86 has both ports and memory mapped I/O



Bus Architectures

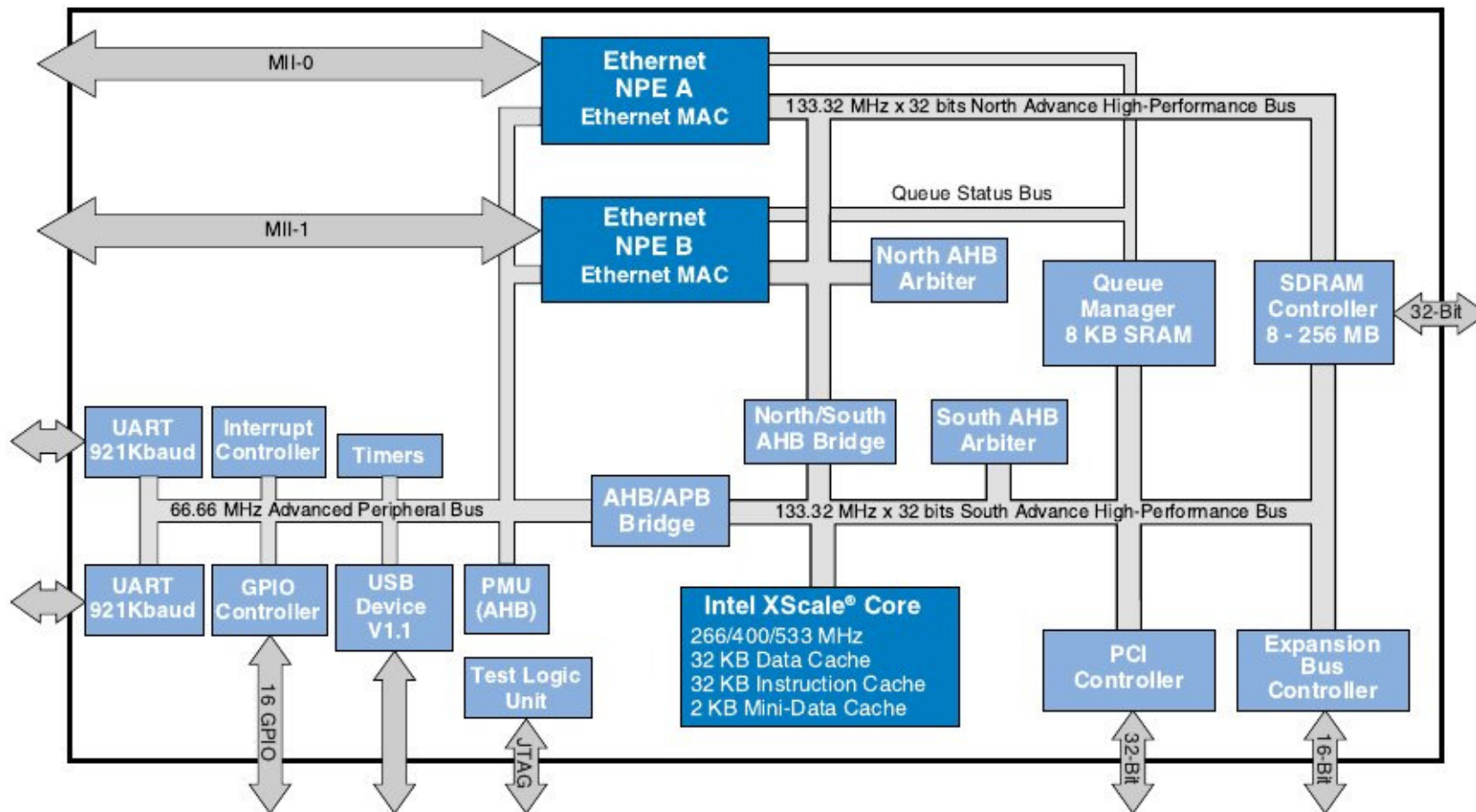


(a) A single-bus architecture

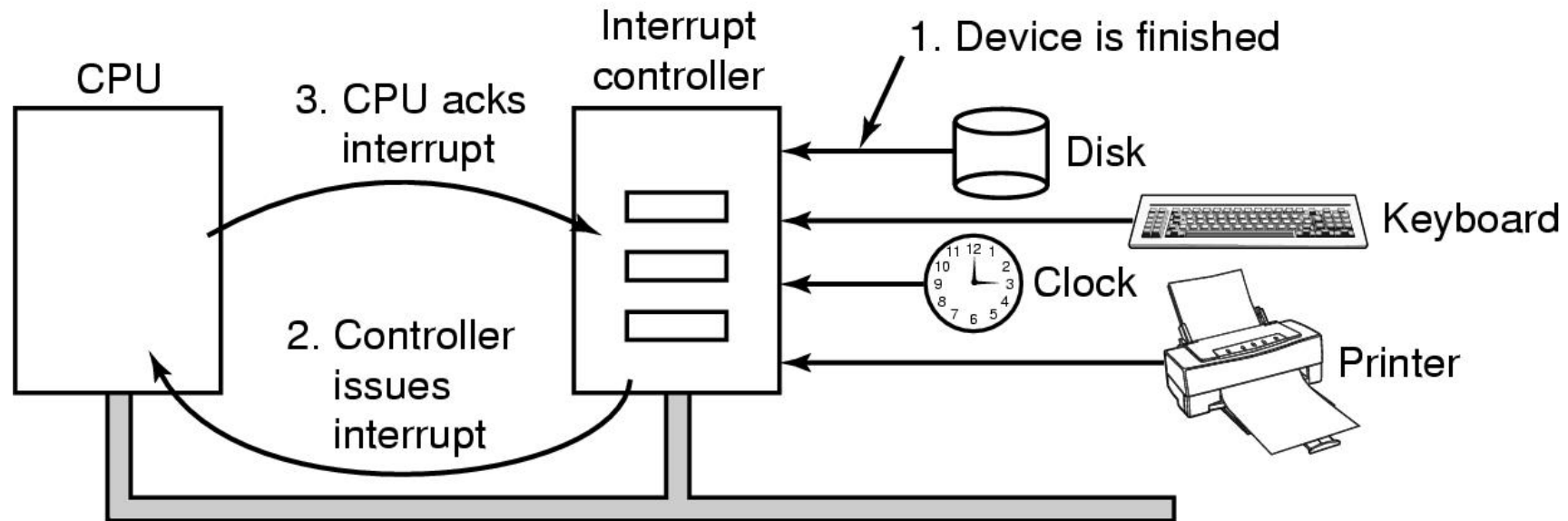
(b) A dual-bus memory architecture



Intel IXP420



Interrupts



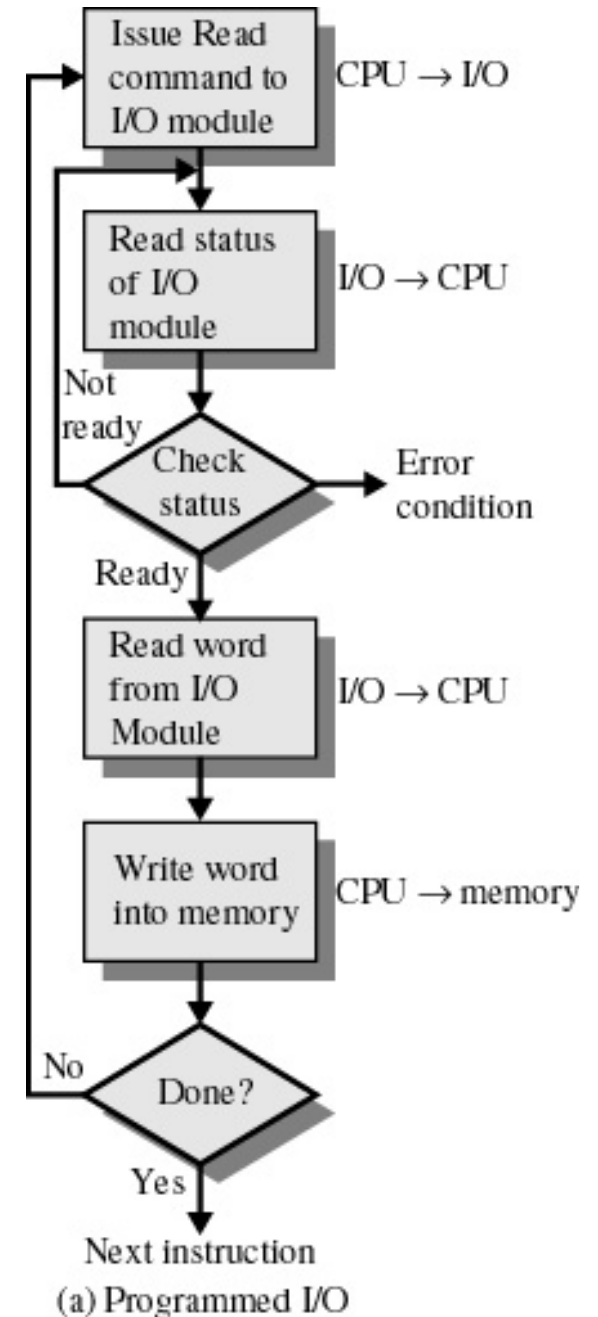
- Devices connected to an *Interrupt Controller* via lines on an I/O bus (e.g. PCI)
- Interrupt Controller signals interrupt to CPU and is eventually acknowledged.
- Exact details are architecture specific.

I/O Interaction



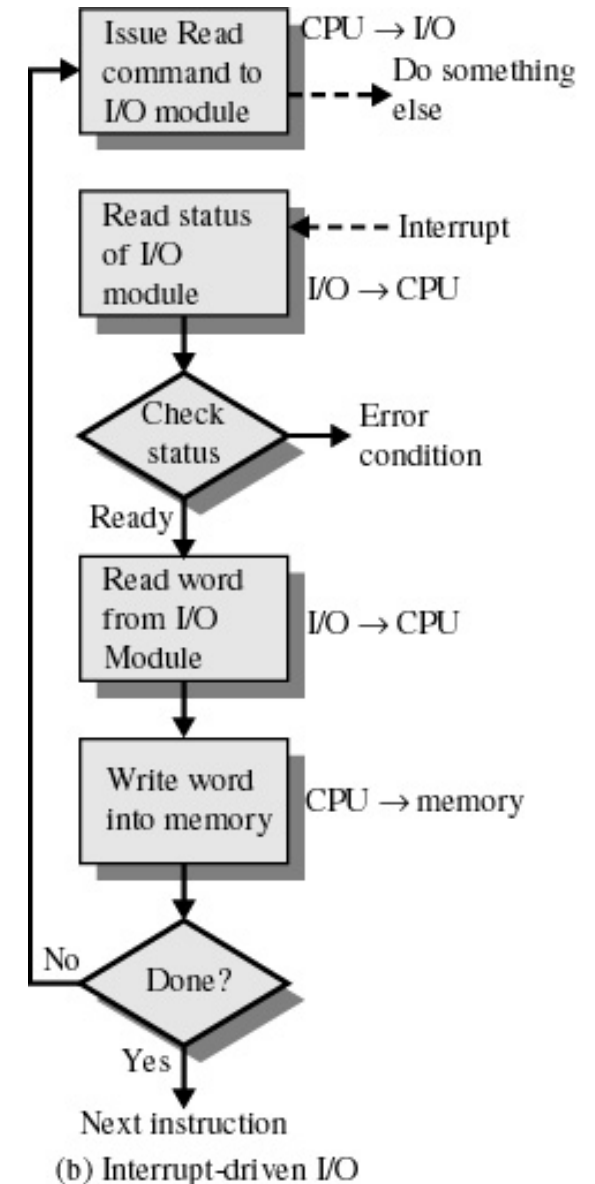
Programmed I/O

- Also called *polling*, or *busy waiting*
- I/O module (controller) performs the action, not the processor
- Sets appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete
 - Wastes CPU cycles



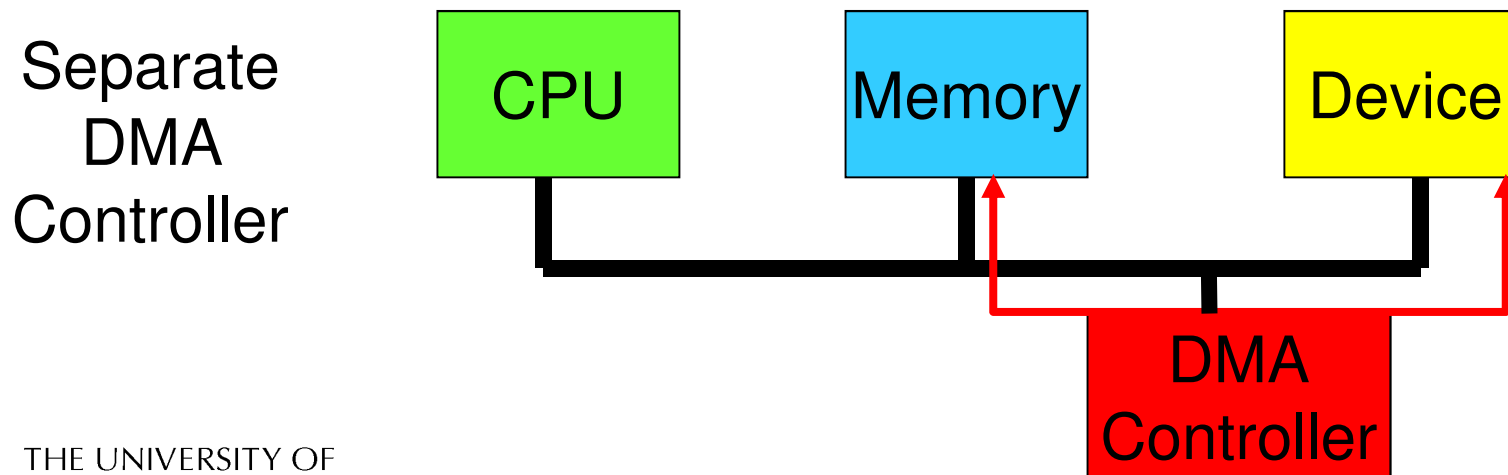
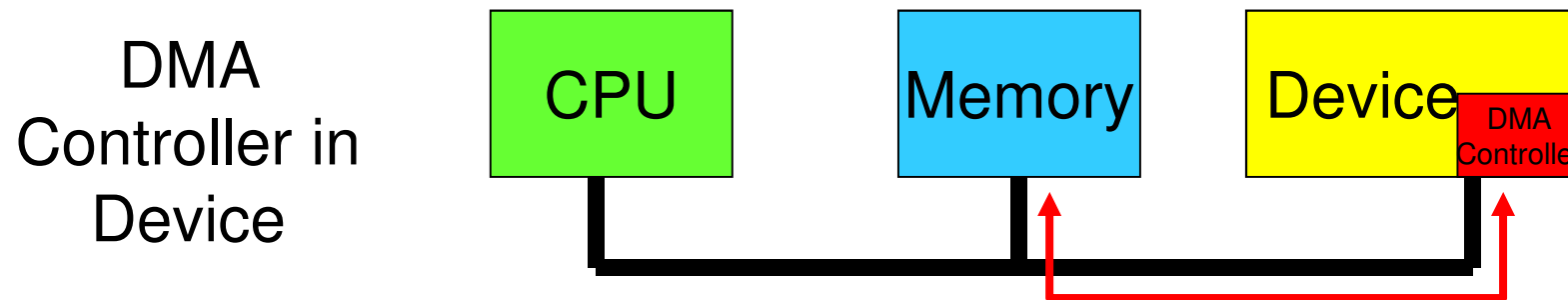
Interrupt-Driven I/O

- Processor is interrupted when I/O module (controller) ready to exchange data
- Processor is free to do other work
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor



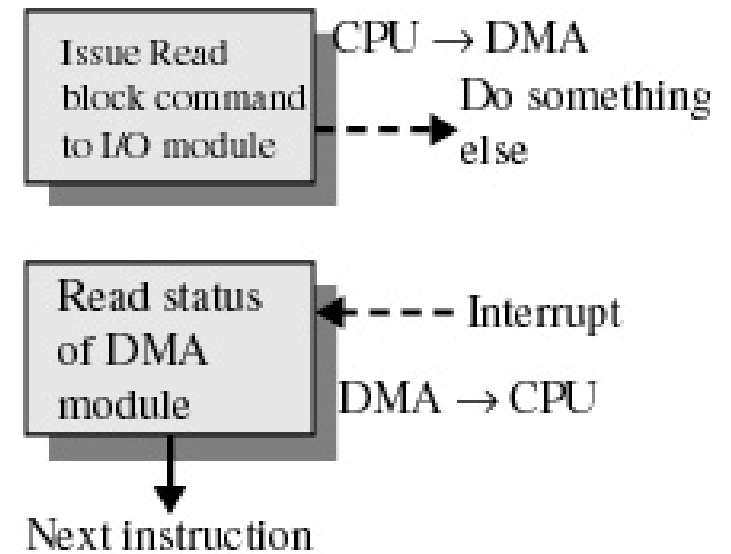
Direct Memory Access

- Transfers data directly between Memory and Device
- CPU not needed for copying



Direct Memory Access

- Transfers a block of data directly to or from memory
- An interrupt is sent when the task is complete
- The processor is only involved at the beginning and end of the transfer

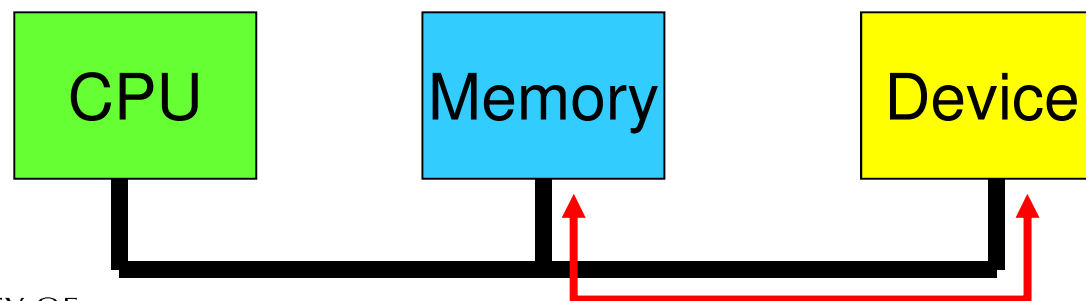


(c) Direct memory access

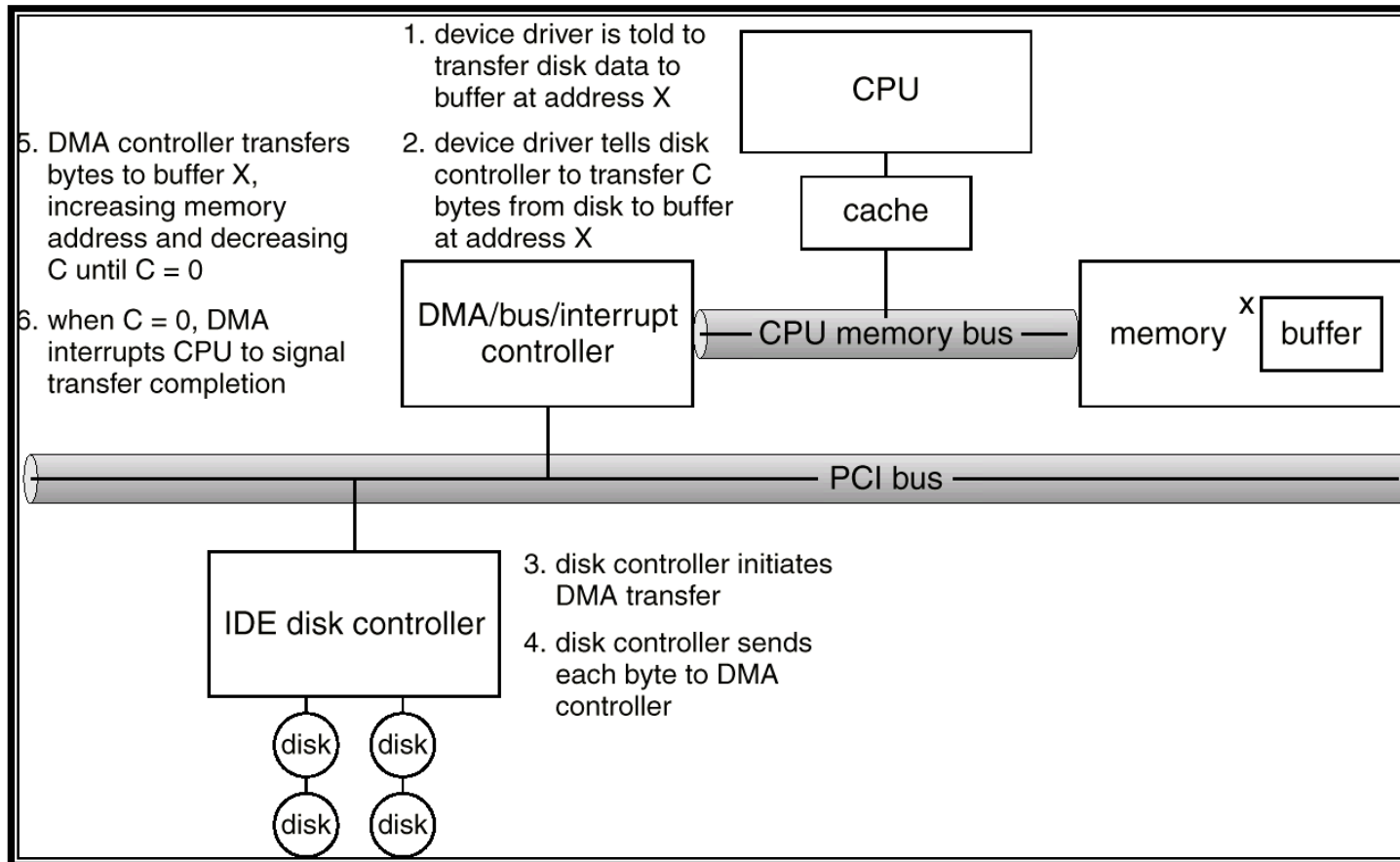


DMA Considerations

- ✓ Reduces number of interrupts
 - Less (expensive) context switches or kernel entry-exits
- ✗ Requires contiguous regions (buffers)
 - Copying
 - Some hardware supports “Scatter-gather”
- Synchronous/Asynchronous
- Shared bus must be arbitrated (hardware)
 - CPU cache reduces (but not eliminates) CPU need for bus



The Process to Perform DMA Transfer

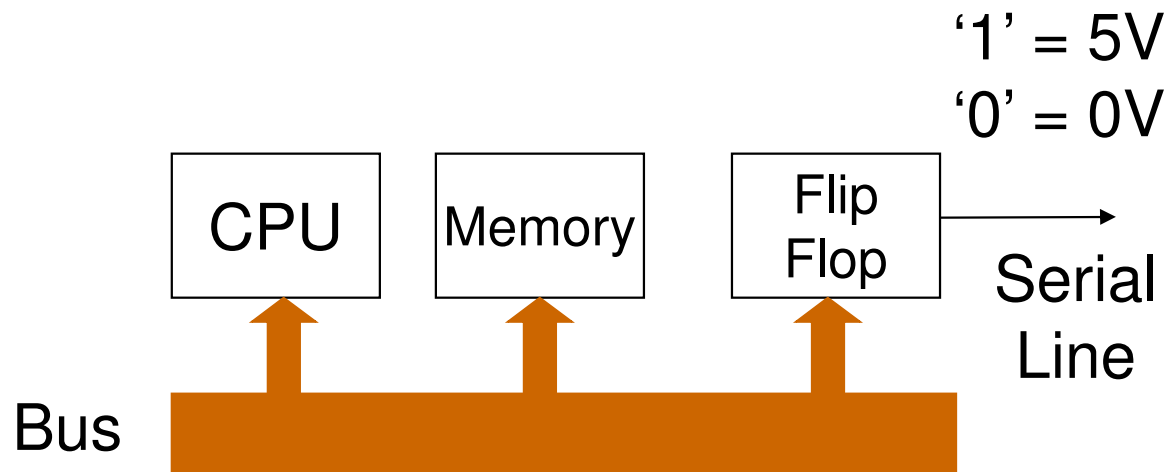


Device Evolution - Complexity and Performance



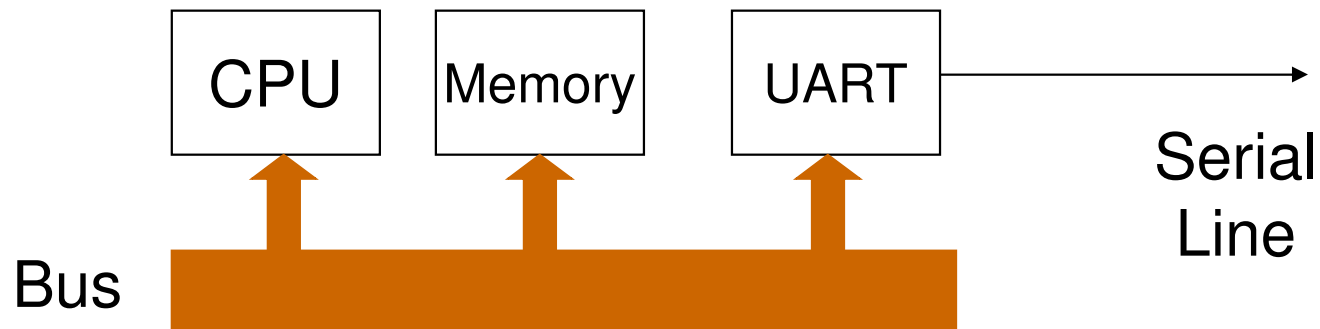
Evolution of the I/O Function

- Processor directly controls a peripheral device
 - Example: CPU controls a flip-flop to implement a serial line



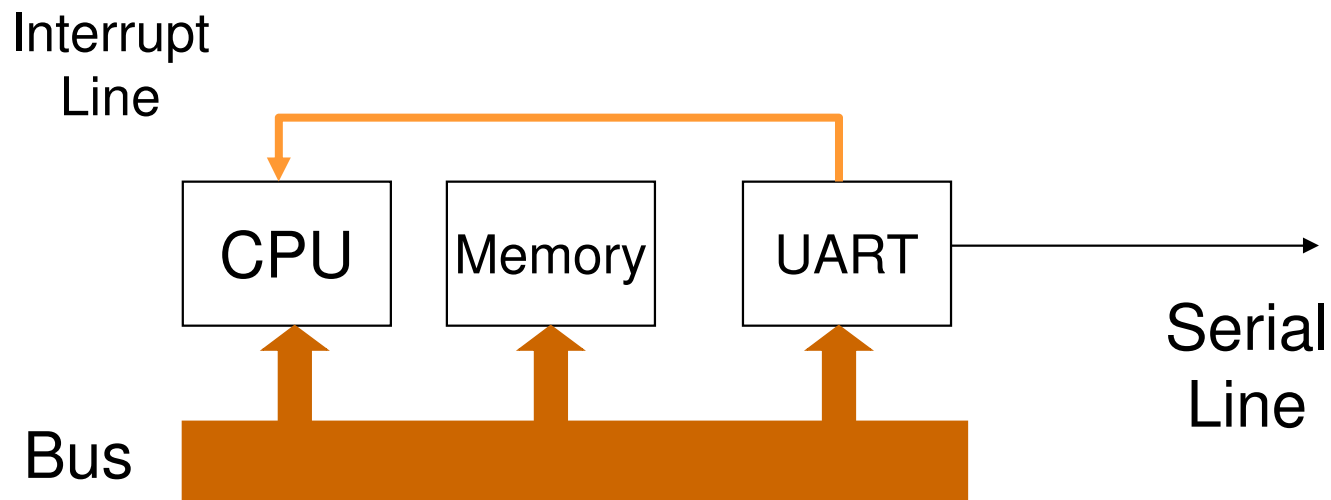
Evolution of the I/O Function

- Controller or I/O module is added
 - Processor uses programmed I/O without interrupts
 - Processor does not need to handle details of external devices
 - Example: A Universal Asynchronous Receiver Transmitter
 - CPU simply reads and writes bytes to I/O controller
 - I/O controller responsible for managing the signaling



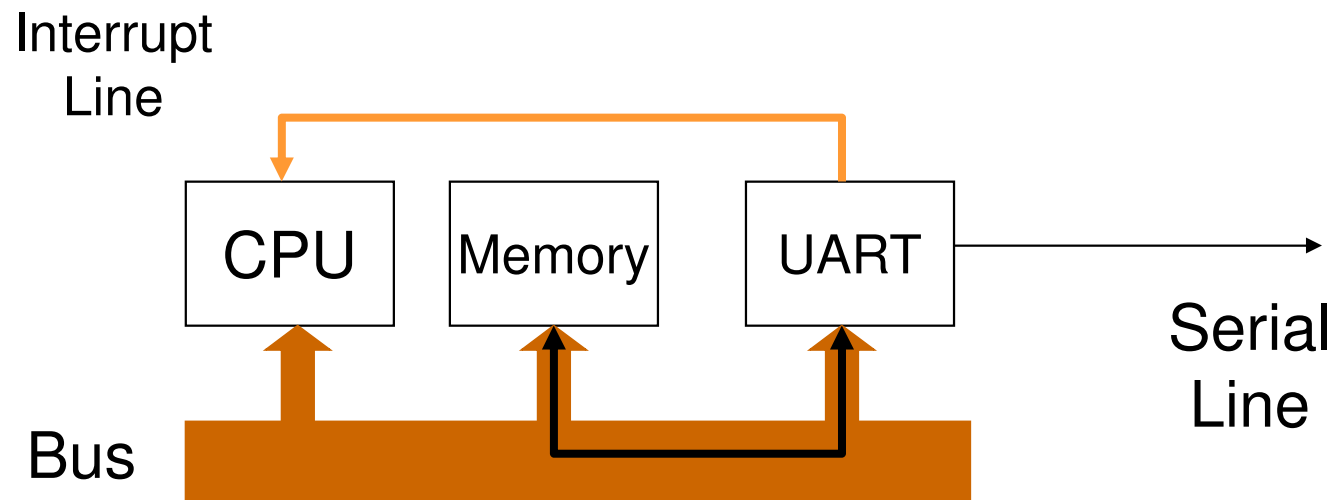
Evolution of the I/O Function

- Controller or I/O module with interrupts
 - Processor does not spend time waiting for an I/O operation to be performed



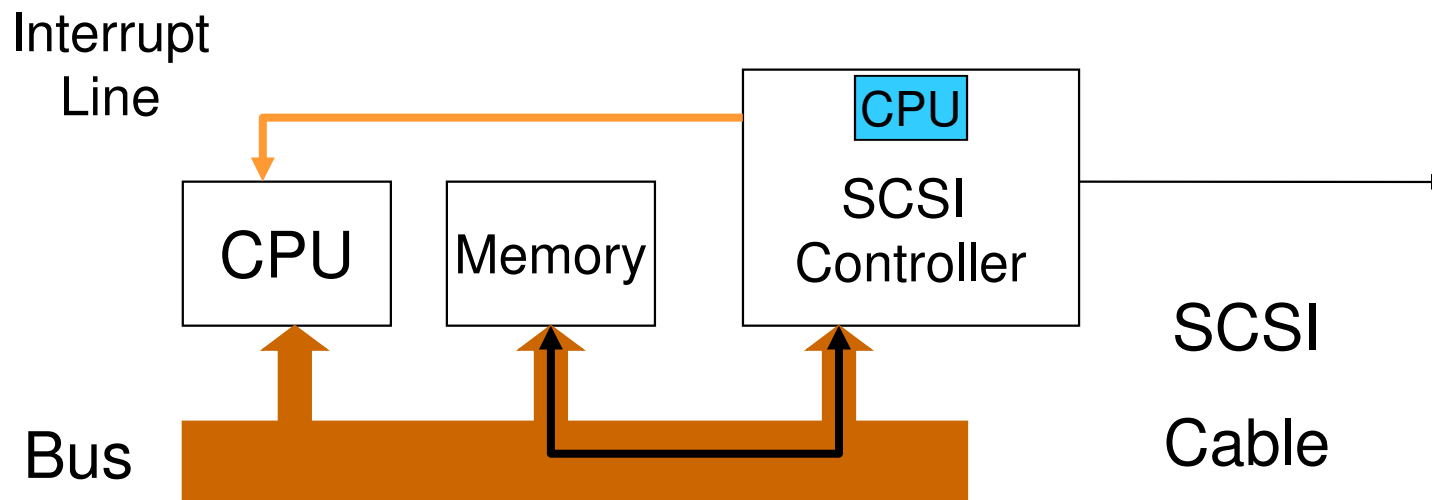
Evolution of the I/O Function

- Direct Memory Access
 - Blocks of data are moved into memory without involving the processor
 - Processor involved at beginning and end only



Evolution of the I/O Function

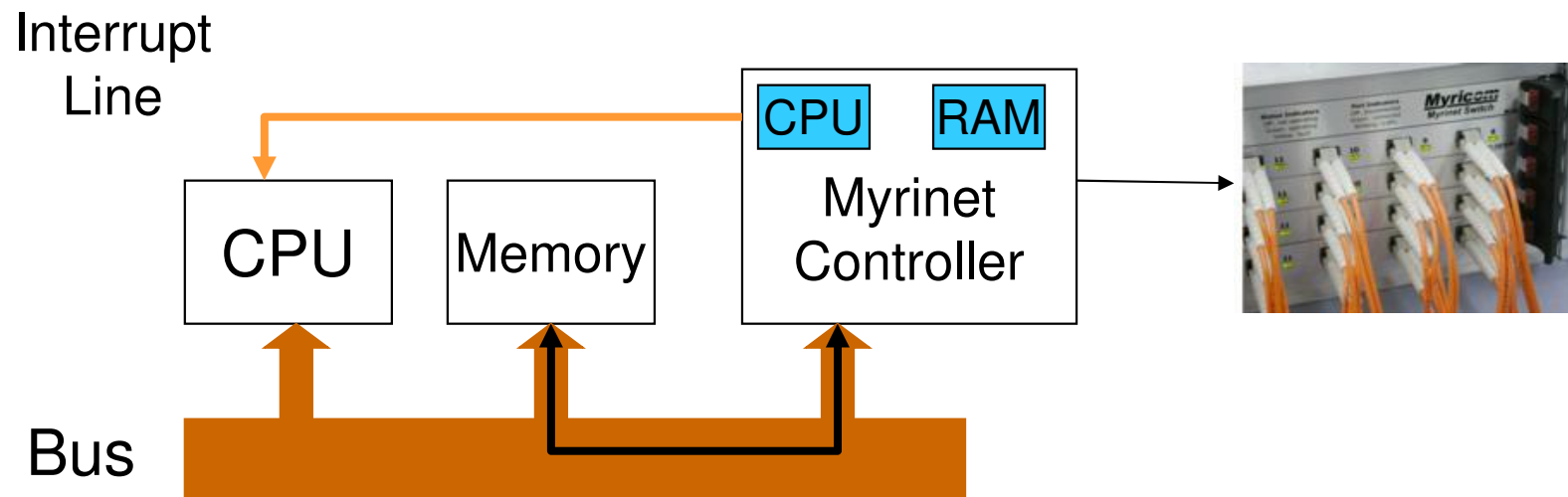
- I/O module has a separate processor
 - Example: SCSI controller
 - Controller CPU executes SCSI program code out of main memory



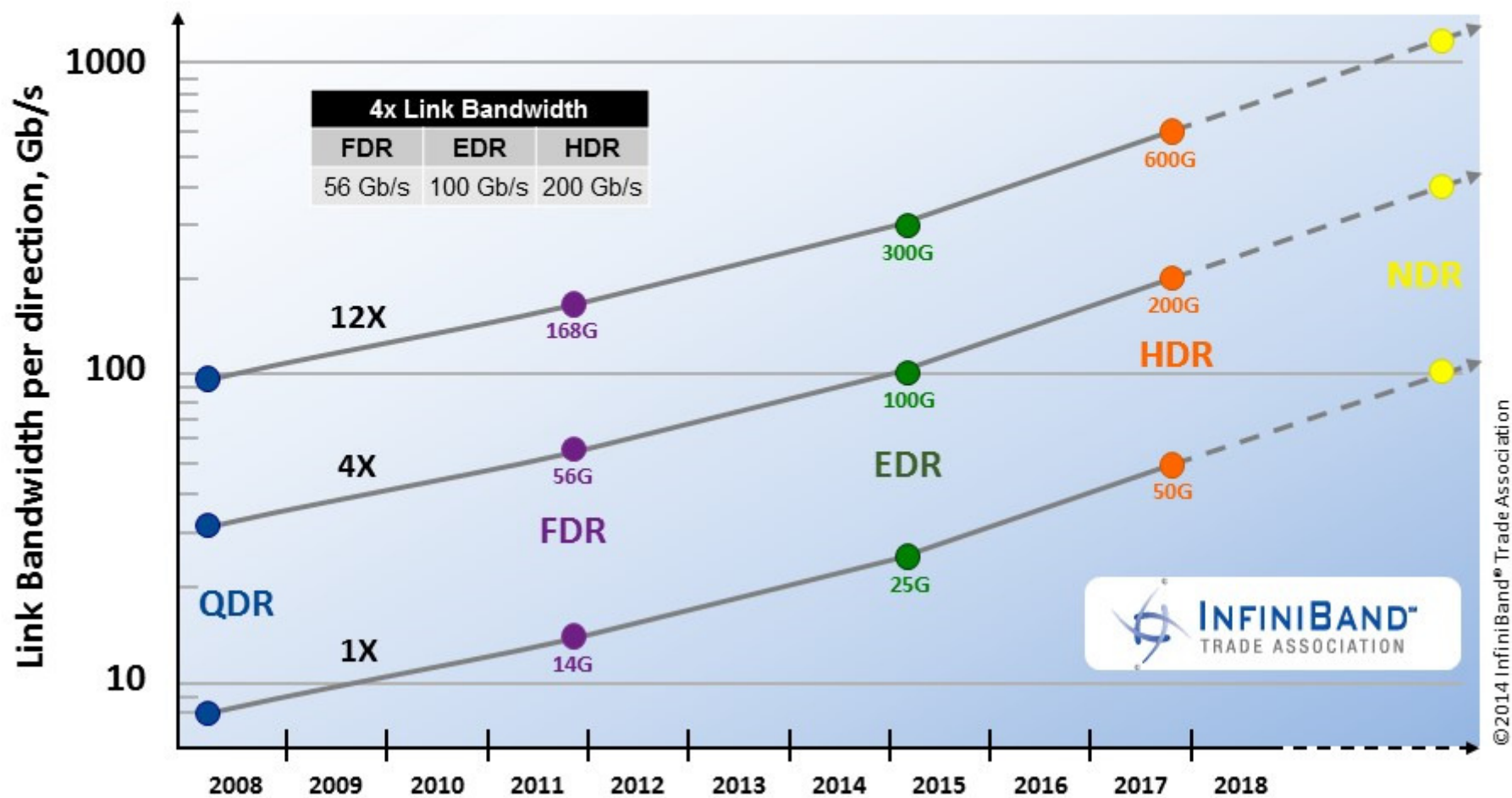
Evolution of the I/O Function

- **I/O processor**

- I/O module has its own local memory, internal bus, etc.
- Its a computer in its own right
- Example: Myrinet 10 gigabit NIC



InfiniBand Roadmap



General Trend

- More specialised hardware
- Offloading more functionality into hardware
 - Reduced load on CPU
- Improved performance

