MONASH University
Information Technology

FIT3159
Computer Architecture

I/O, Interrupts, DMA, Mass Storage / Hierarchies

Dr Carlo Kopp, SMIEEE, AFAIAA, PEng
Carlo.Kopp@monash.edu
Clayton School of Information Technology,
Monash University
Australia

Why I/O, Interrupts, DMA, Mass Storage?

- ***I/O, Interrupts, DMA, Mass Storage impacts performance and functionality of all modern computers.***
- **Foundation knowledge: Understand key performance impacts in I/O design, interrupt handling mechanisms, and storage -> important in understanding operating systems.**
- **Foundation knowledge: Mass storage is critical to system performance and a major area in contemporary machine design.**
- **Practical skills: Ability to configure a machine for good I/O performance by making good decisions on interfaces.**
- **Practical skills: Ability to recognise and differentiate peripheral performance by understanding specifications.**
- **Practical skills: Understand key concepts and problems in operating systems and device driver design.**

Asynchronous I/O

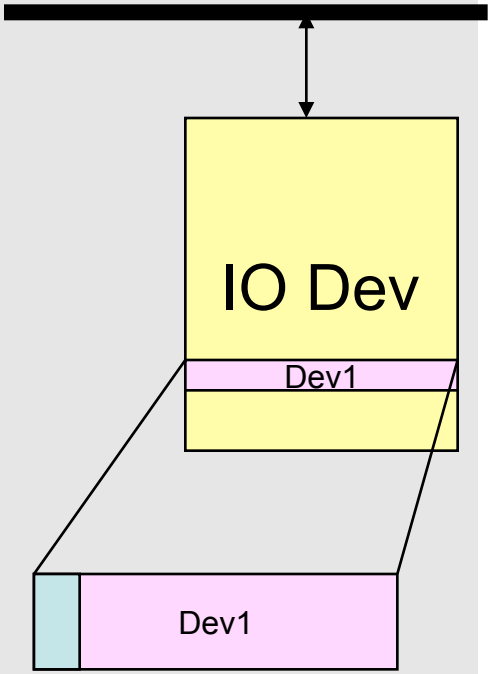
- Computers are seldom operated in the manner where a program is started and the CPU is allowed to run it until it is completed.
- The CPU must also service I/O devices such as disks, tapes, keyboards, mice, graphics adapters, modems, CD drives etc.
- Such devices are designed to read, or read and write data, either to/from storage, a communications channel, or display device.

Asynchronous I/O

- An I/O device will take some finite amount of time to perform a command issued to it by the CPU
- It must be capable of telling the CPU that it has finished its activity.
- Communications adapters and user input devices may receive input data destined for the CPU at any time. The CPU cannot know *a priori*, exactly when this data will arrive.
- Question: ***how can the CPU service such devices without having them wait unnecessarily?***

Polled I/O

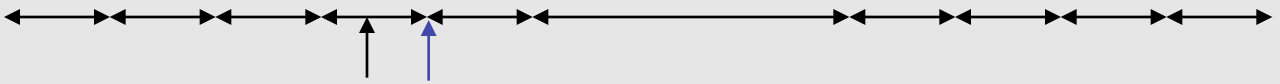
- The most simple strategy for servicing I/O devices is termed “polling”.
- In a polled system the CPU executes a program which cycles through an endless loop, checking each I/O device once per loop to see if it needs to be serviced, and then executing whatever main program it is to run.
- If a device needs to be serviced, the CPU executes the service routine and then resumes its polling loop.




```
wait:
    LDA   DEV1
    JGTZ  wait
/*Dev Ready */
```

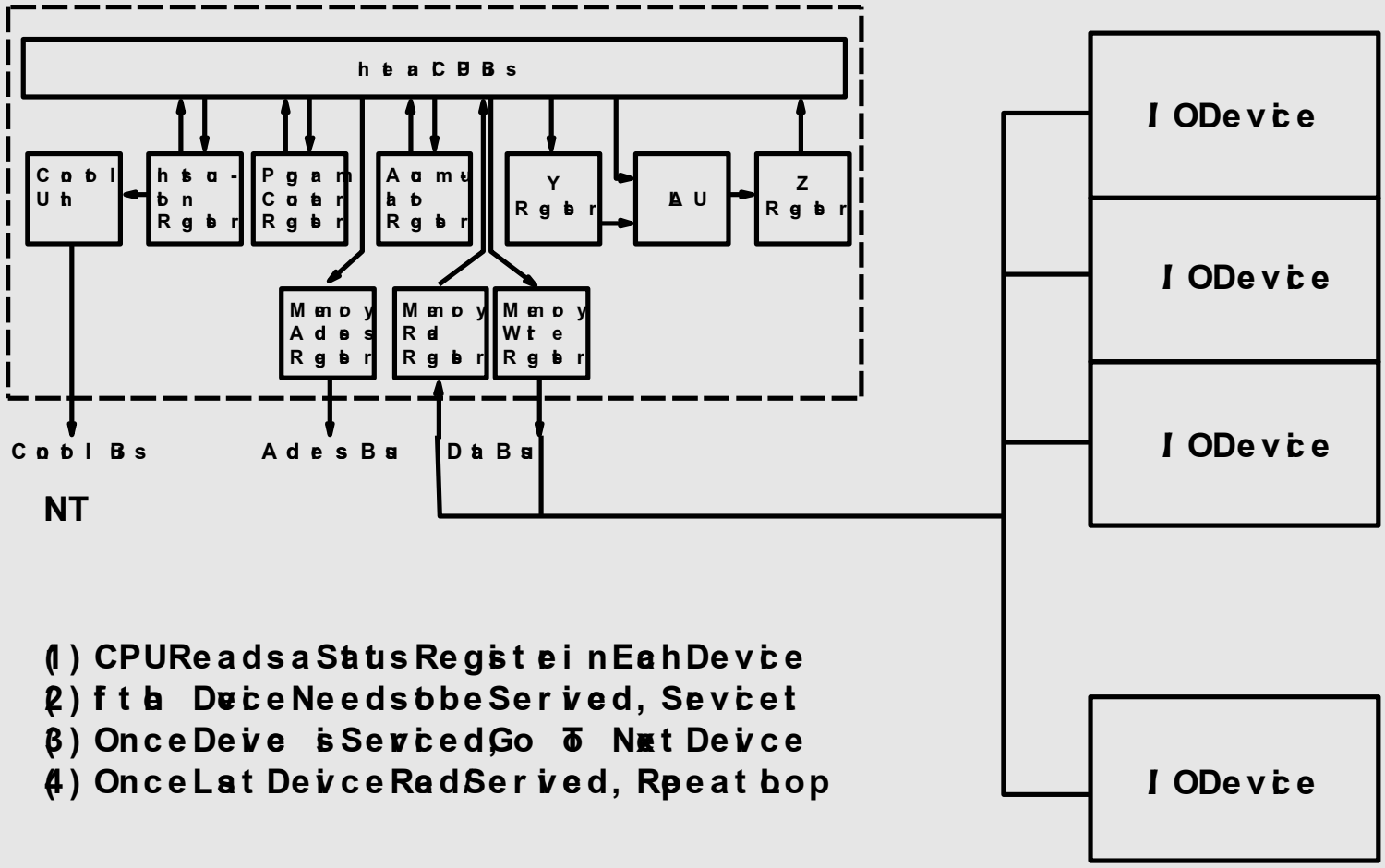
Polled I/O

- **While polling is often appealing due to its conceptual simplicity, it has two fatal flaws:**
  - **Response time:** if a device becomes ready to be serviced just after it has been polled, it must wait for the whole duration of the polling loop until it can be serviced.



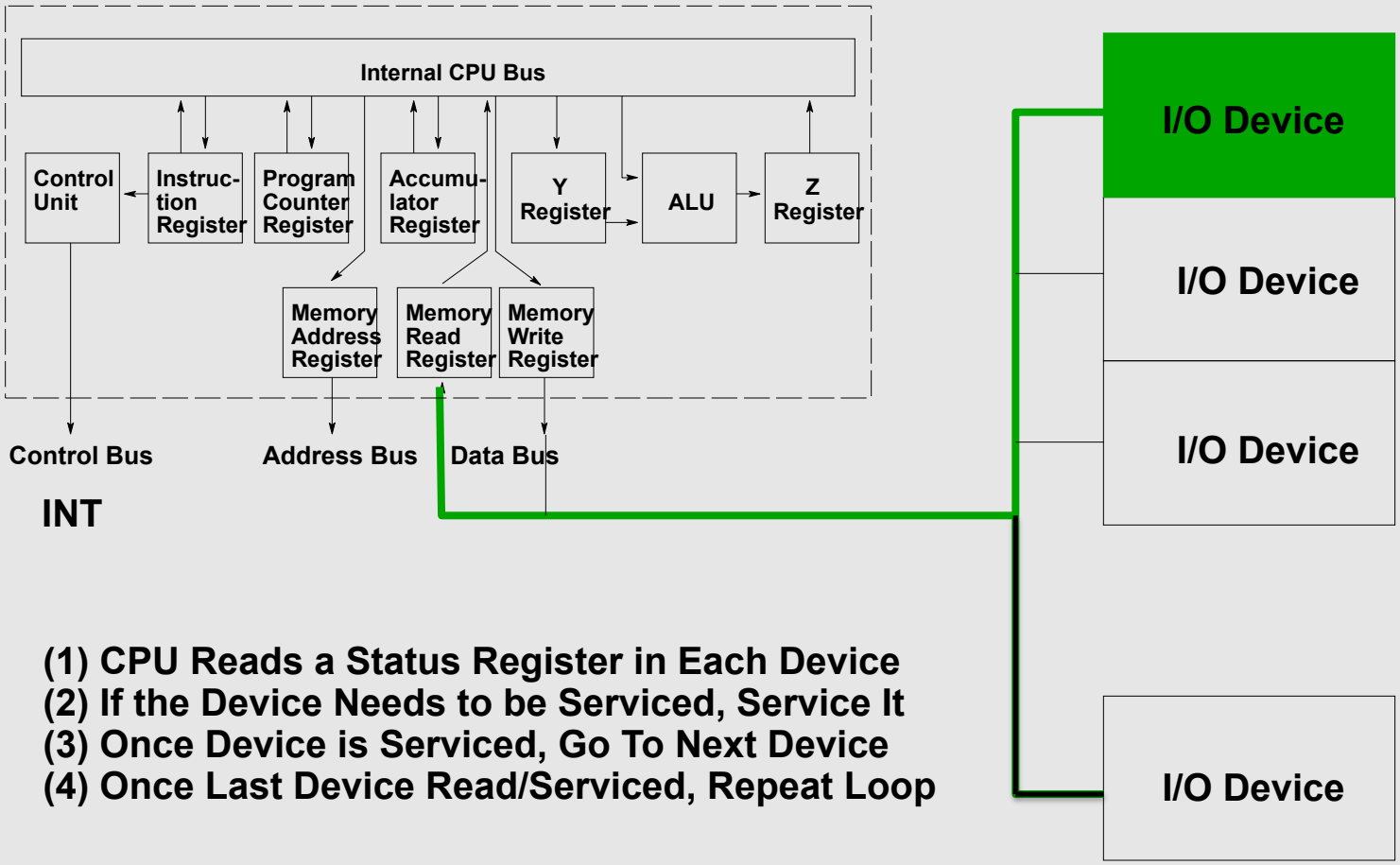
  - **Unpredictability of Response Time:** since it is impossible to know *a priori* how many devices will need to be serviced, the time to poll a device will vary with the number of devices ready to be serviced.

Handling Multiple Devices



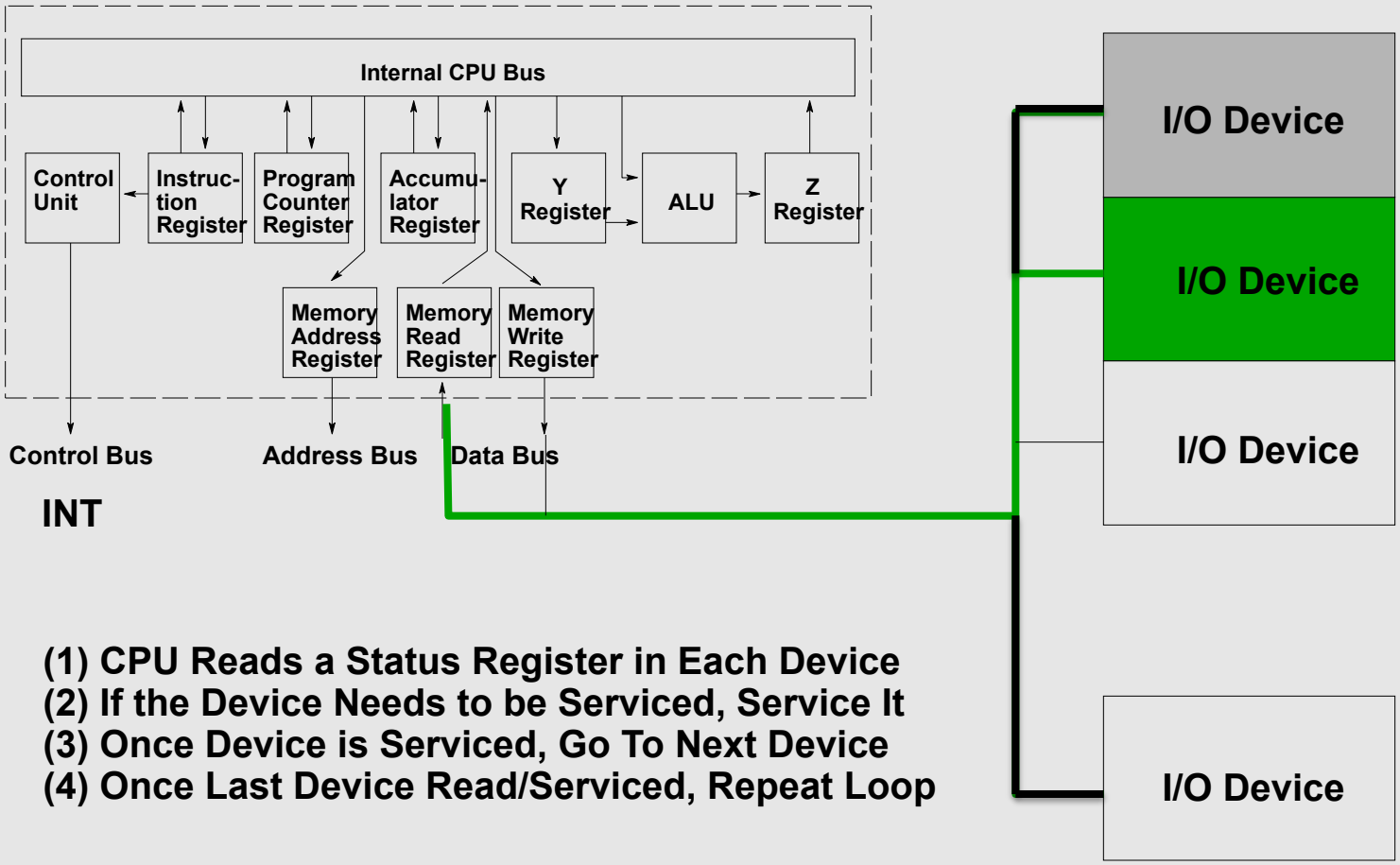
(1) CPU Reads a Status Register in Each Device
(2) If the Device Needs to be Serviced, Service it
(3) Once Device is Serviced, Go To Next Device
(4) Once Last Device Read/Serviced, Repeat Loop

Handling Multiple Devices



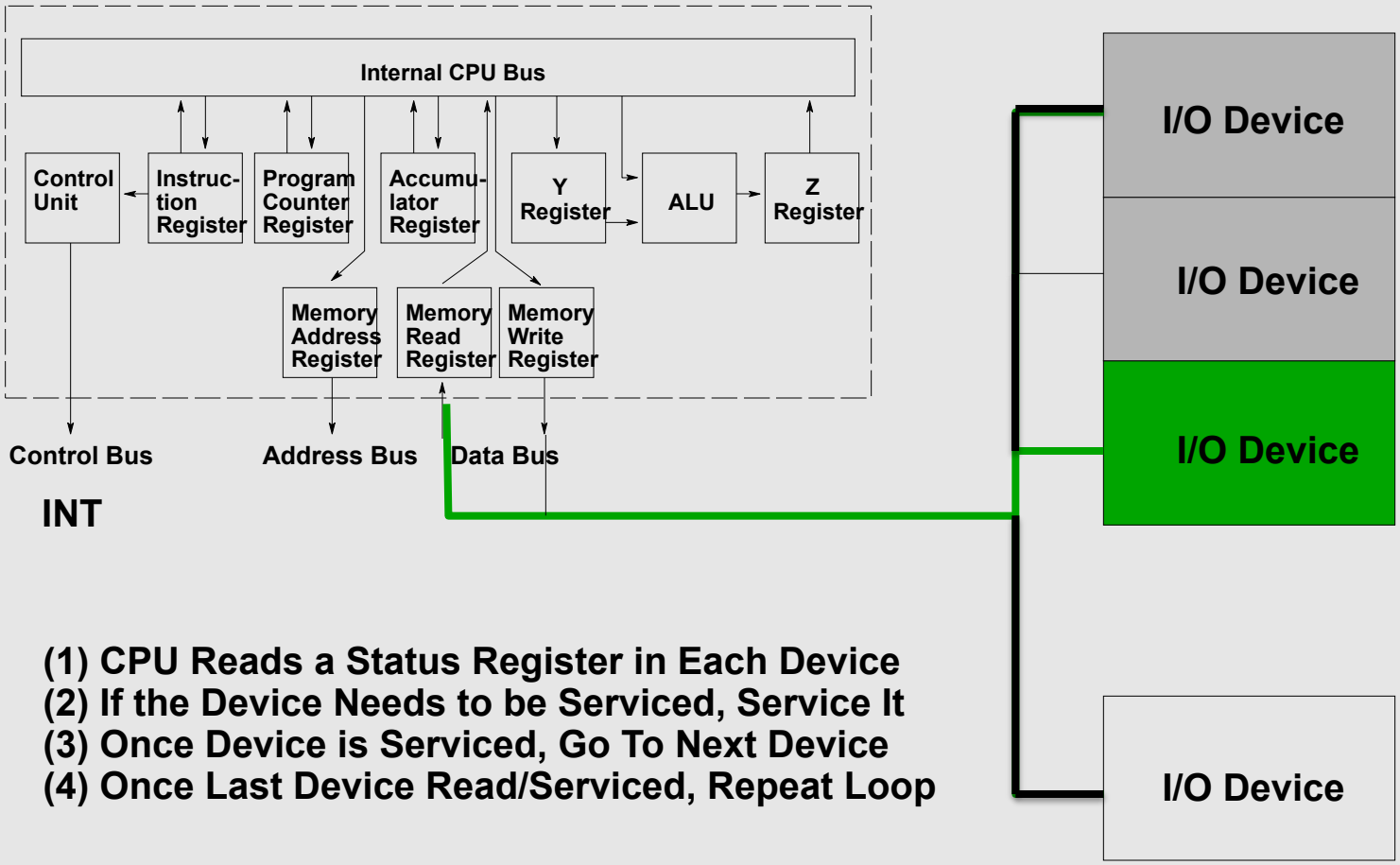
(1) CPU Reads a Status Register in Each Device
(2) If the Device Needs to be Serviced, Service It
(3) Once Device is Serviced, Go To Next Device
(4) Once Last Device Read/Serviced, Repeat Loop

Handling Multiple Devices



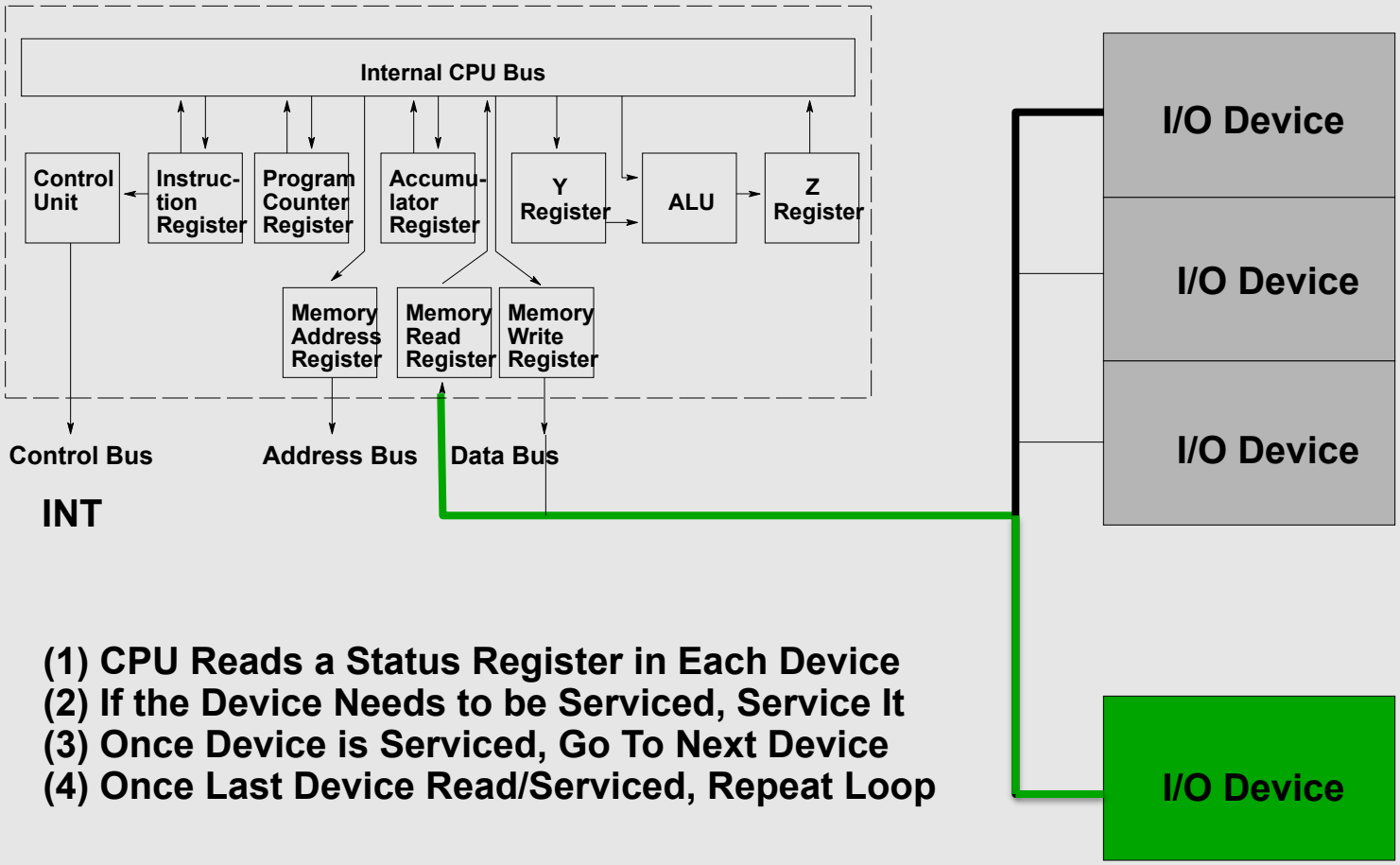
(1) CPU Reads a Status Register in Each Device
(2) If the Device Needs to be Serviced, Service It
(3) Once Device is Serviced, Go To Next Device
(4) Once Last Device Read/Serviced, Repeat Loop

Handling Multiple Devices



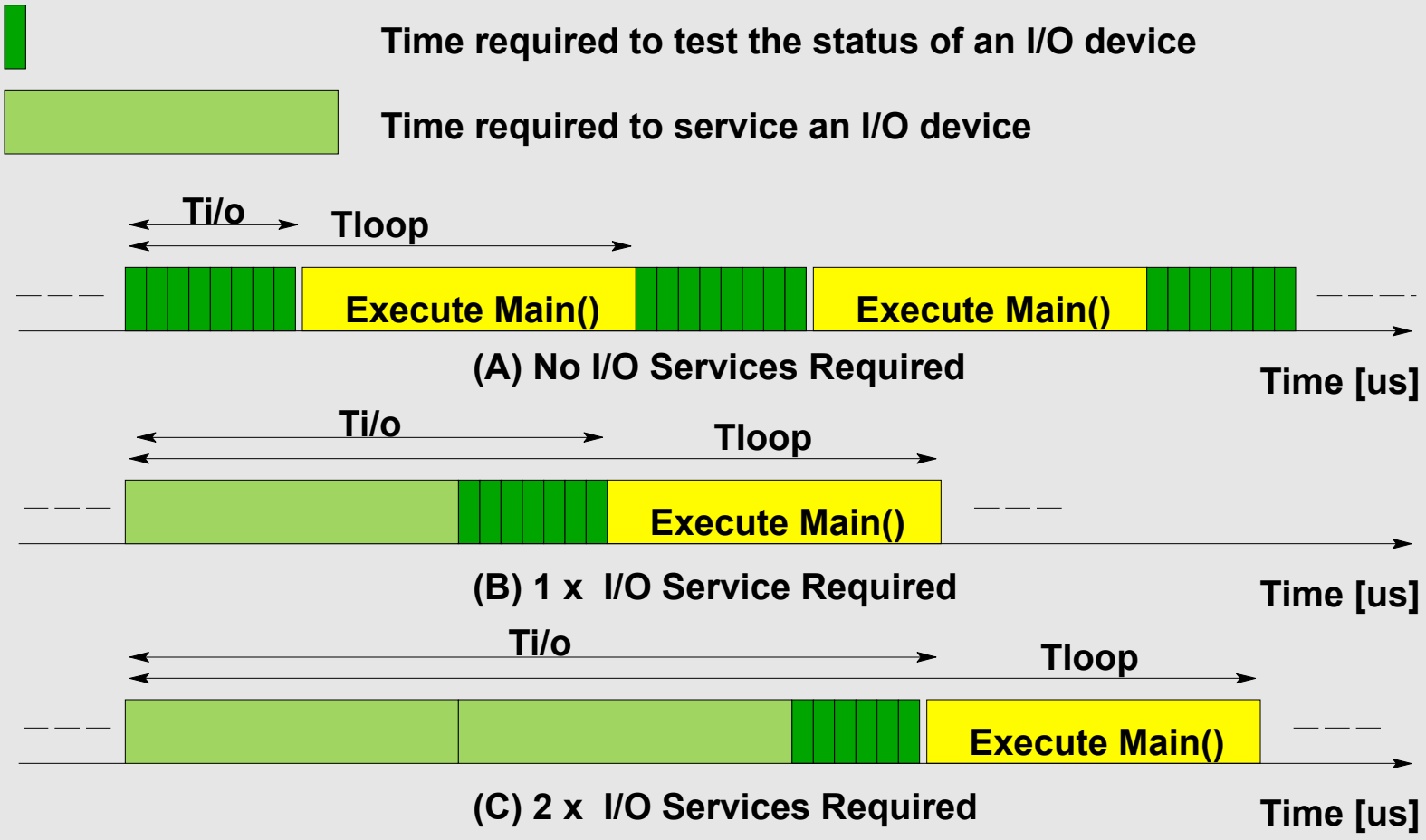
(1) CPU Reads a Status Register in Each Device
(2) If the Device Needs to be Serviced, Service It
(3) Once Device is Serviced, Go To Next Device
(4) Once Last Device Read/Serviced, Repeat Loop

Handling Multiple Devices



(1) CPU Reads a Status Register in Each Device
(2) If the Device Needs to be Serviced, Service It
(3) Once Device is Serviced, Go To Next Device
(4) Once Last Device Read/Serviced, Repeat Loop

Service Time in Polled I/O

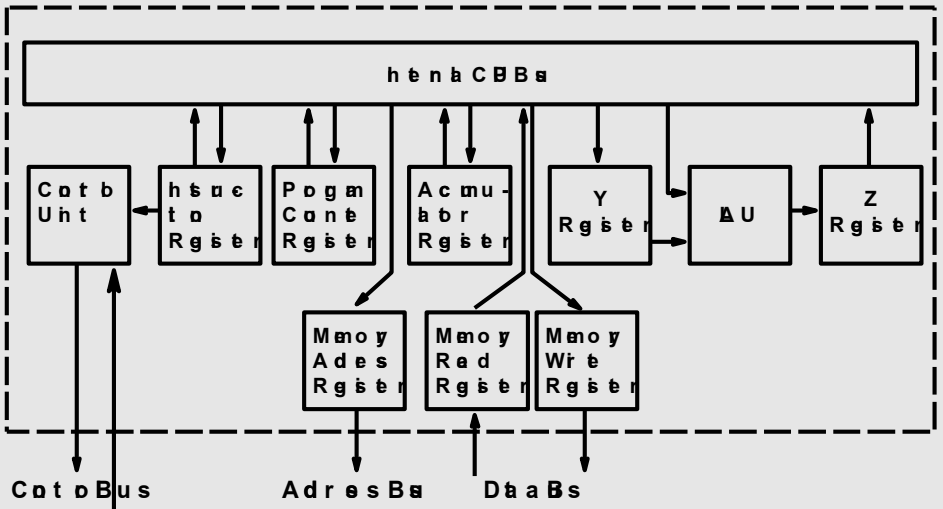


As the number of I/O devices requiring service increases, the rate of polling slows down.

Interrupts

- The preferred alternative to polling is to “interrupt” the CPU.
- Interrupt driven I/O used in most practical applications.
- Interrupting the CPU amounts to providing it with a logical signal which says: *“immediately stop what you are working on and service me”*.
- Upon receiving the interrupt signal, the CPU must cease executing the current program, and jump to the start of the “interrupt service routine” (ISR) to service the interrupting device.

Interrupts



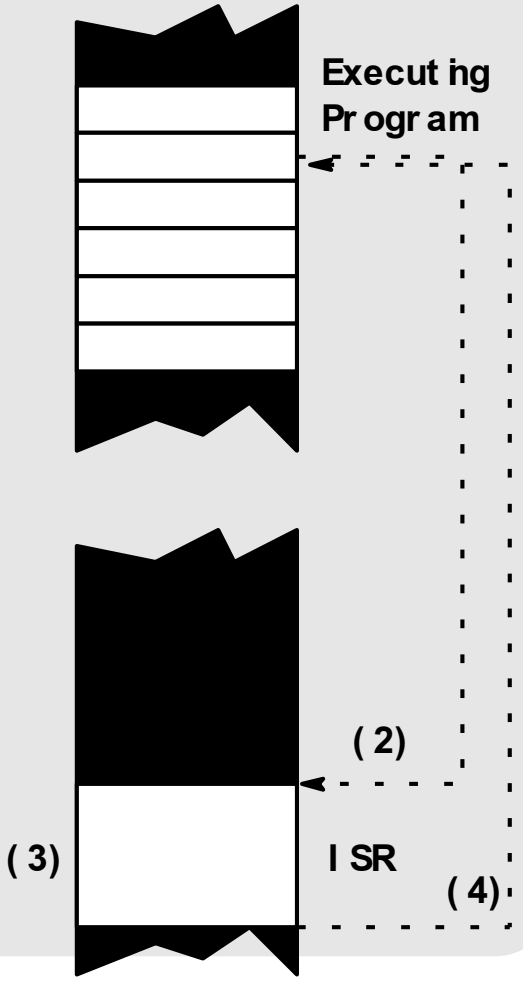
(1)

(1) I/O Device Asserts INT
(2) CPU Jumps to ISR
(3) ISR Services I/O Device
(4) Once ISR Completed, CPU Resumes

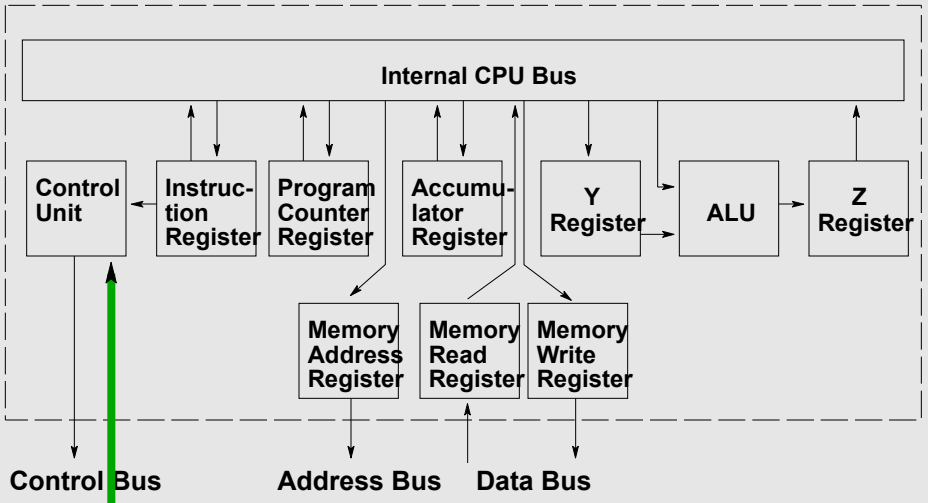
"1" == Service Me"
"0" == Do Nothing"

INTERRUPT

Interrupting I/O Device



Interrupts



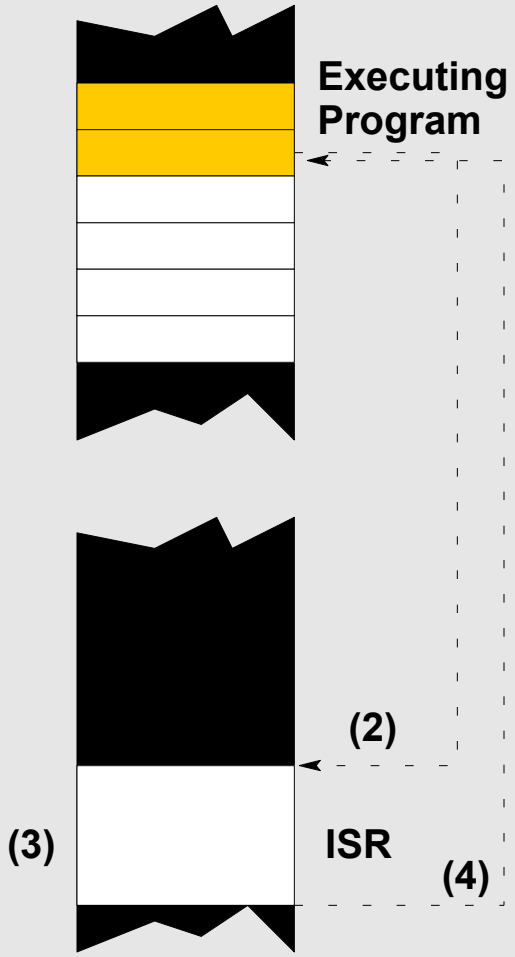
(1)

(1) I/O Device Asserts INT
(2) CPU Jumps to (ISR)
(3) ISR Services I/O Device
(4) Once ISR Completed, CPU Resumes

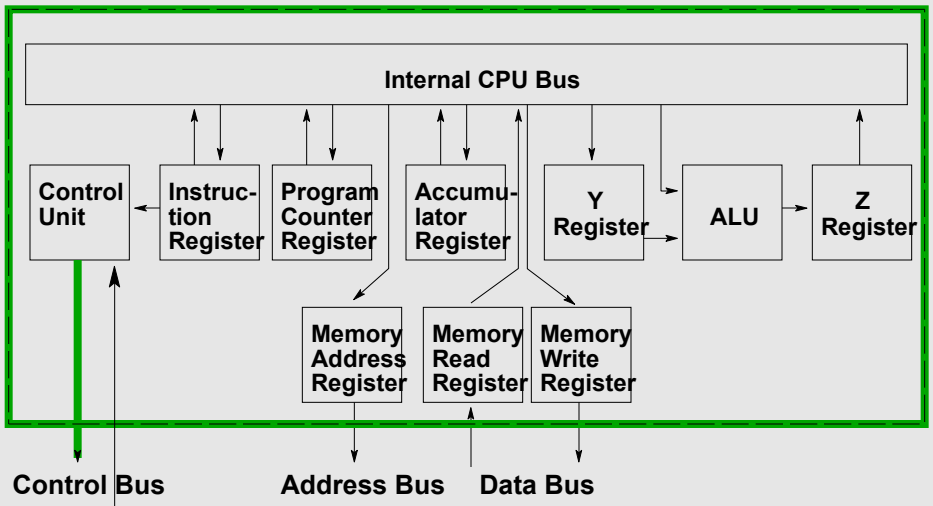
"1" == "Service Me"
"0" == "Do Nothing"

INTERRUPT

Interrupting I/O Device



Interrupts

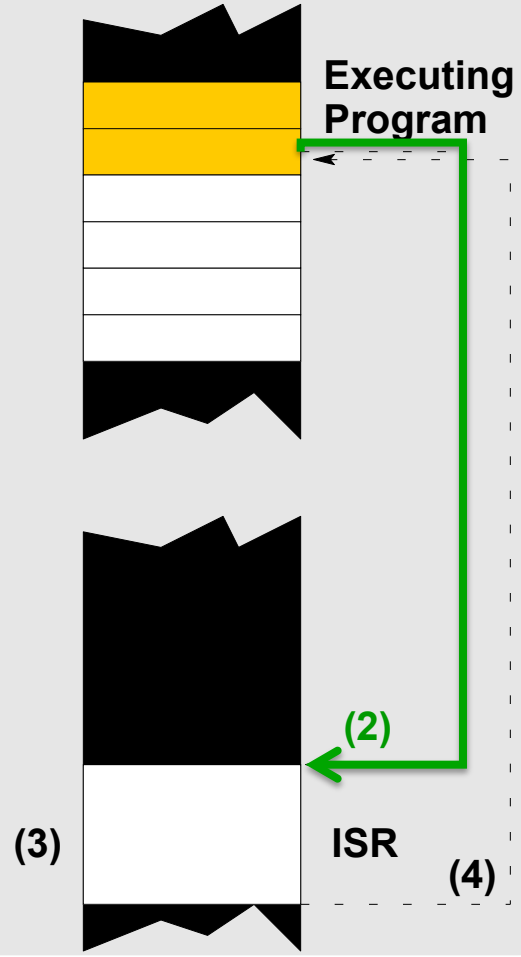


(1) (1) I/O Device Asserts INT
(2) CPU Jumps to (ISR)
(3) ISR Services I/O Device
(4) Once ISR Completed, CPU Resumes

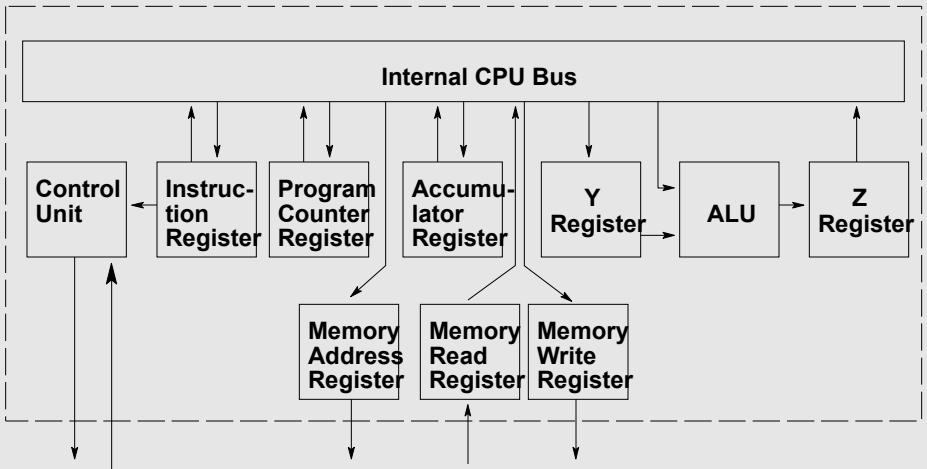
"1" == "Service Me"
"0" == "Do Nothing"

INTERRUPT

Interrupting I/O Device



Interrupts



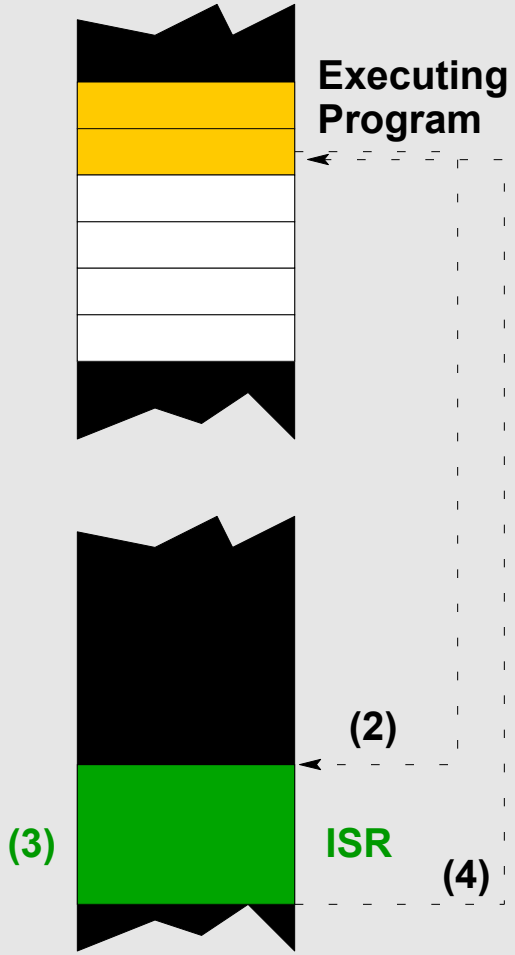
(1)

(1) I/O Device Asserts INT
(2) CPU Jumps to (ISR)
(3) ISR Services I/O Device
(4) Once ISR Completed, CPU Resumes

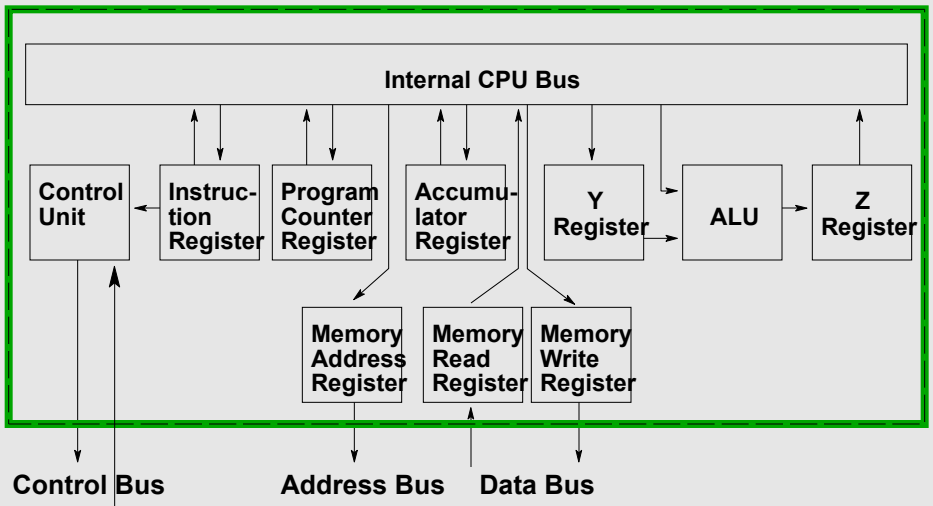
"1" == "Service Me"
"0" == "Do Nothing"

INTERRUPT

Interrupting I/O Device



Interrupts

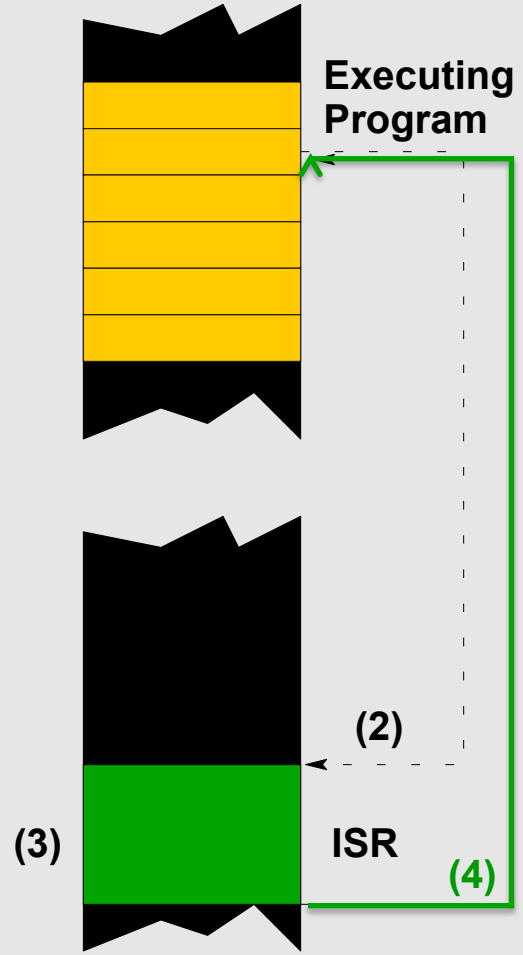


(1) (1) I/O Device Asserts INT
(2) CPU Jumps to (ISR)
(3) ISR Services I/O Device
(4) Once ISR Completed, CPU Resumes

"1" == "Service Me"
"0" == "Do Nothing"

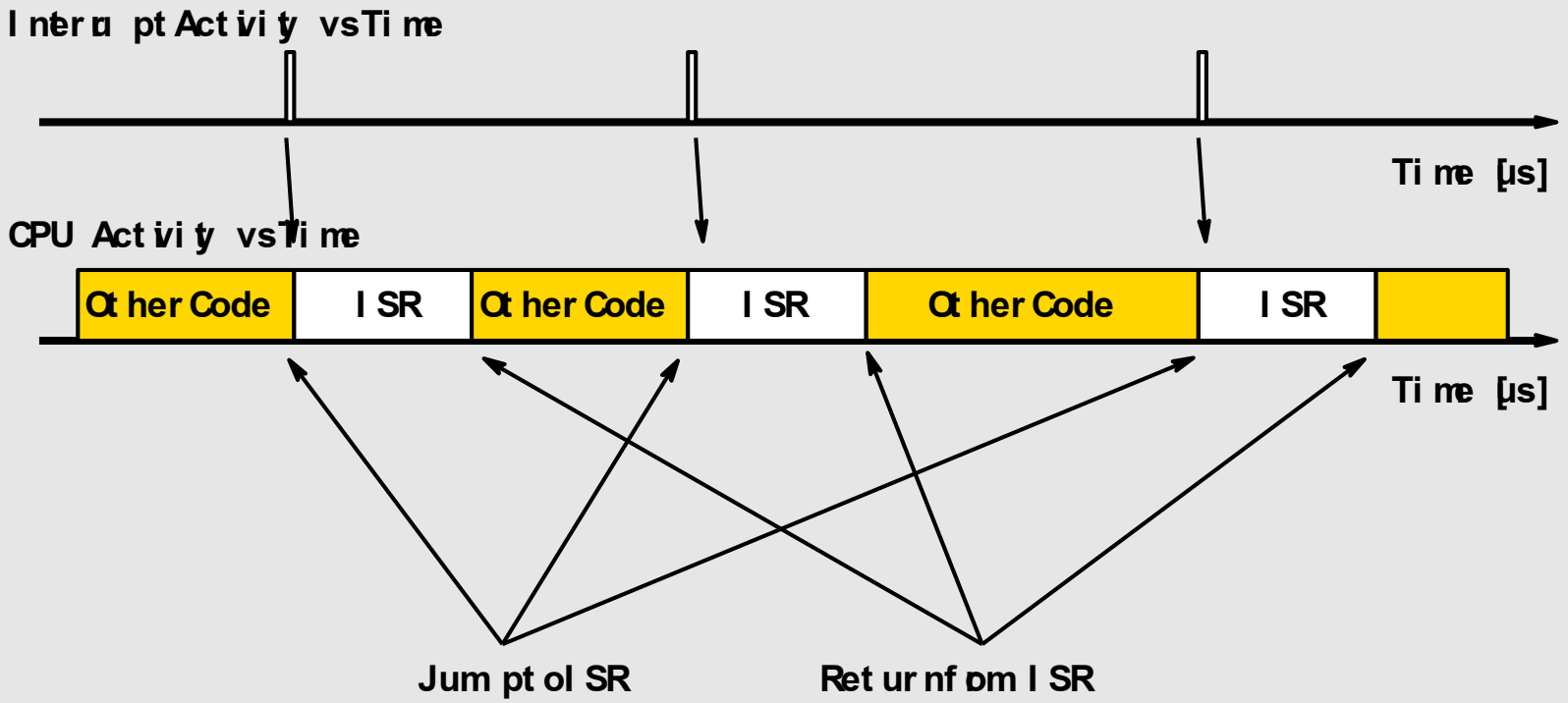
INTERRUPT

Interrupting I/O Device



Interrupt Timing

When the INT is asserted, the CPU executes the ISR to service the device, once the ISR is completed, it resumes executing the previous program.



How Interrupts are Detected

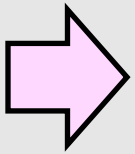
- In a microcoded CPU like the simple example used earlier, at the beginning or the end of each instruction a microword is used which tests the master interrupt line.
- If the interrupt line is active, the next microprogram to be executed is one which saves the state of the CPU, and then loads the PC with the address of the ISR.
- In a vectored system, this microcode tells the device to assert its vector.
- In a hardwired CPU this function will be performed by dedicated logic.

Servicing Interrupts - Polling

- The central problem in handling interrupts is that of very quickly identifying which is the interrupting device.
- The simplest technique is to poll the devices which are wired to the interrupt signal.
- The first ready device which is polled is then serviced.
- Polled interrupt servicing is relatively slow, since many devices may need to be checked before the ready device is found.

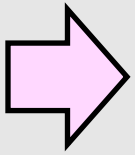
Logic for handling interrupts

Normal operation



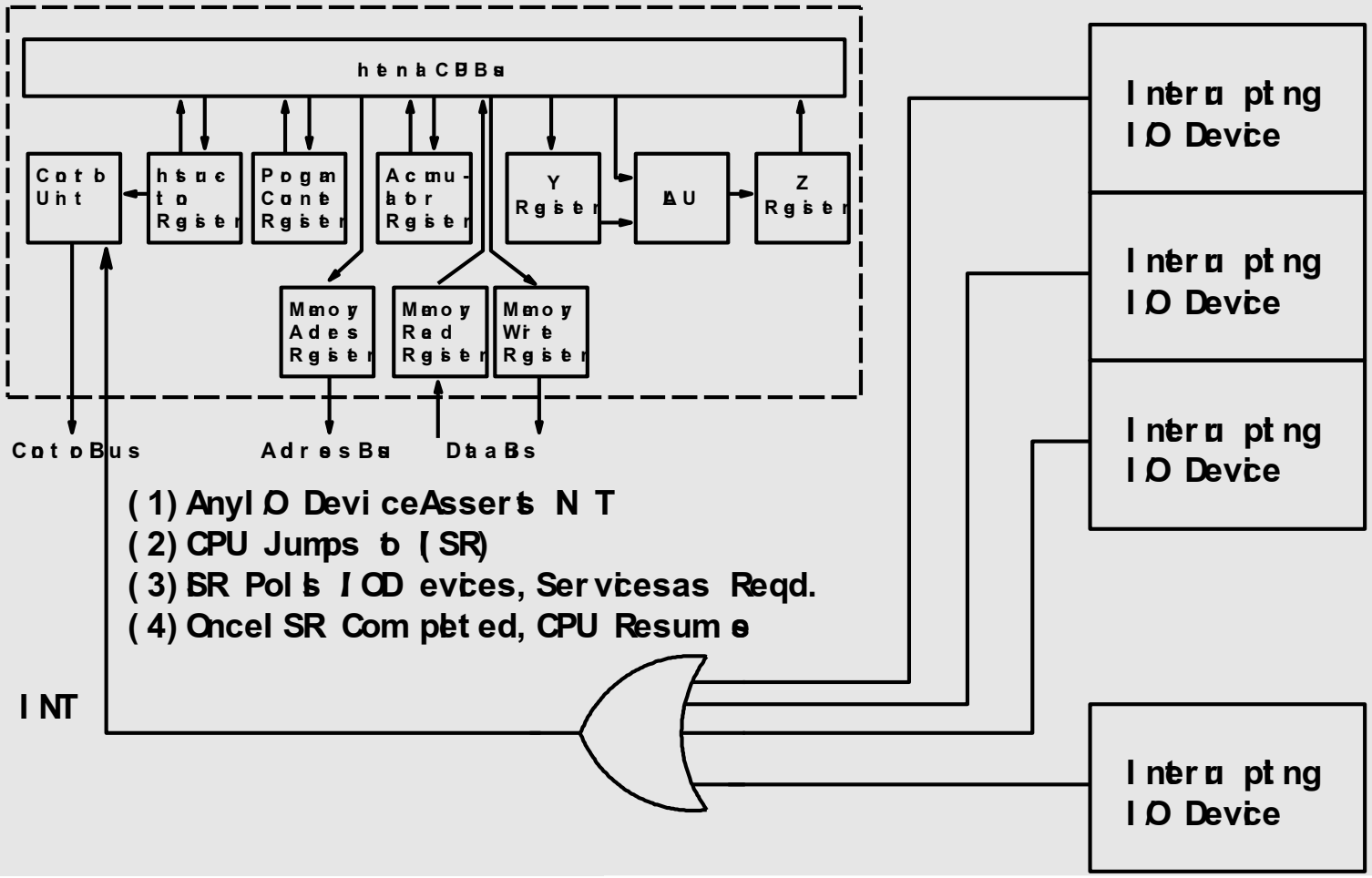
```
while running do begin
        fetchinstruction();
        decodeinstruction();
        executeinstruction();
        end
```

Interrupt operation

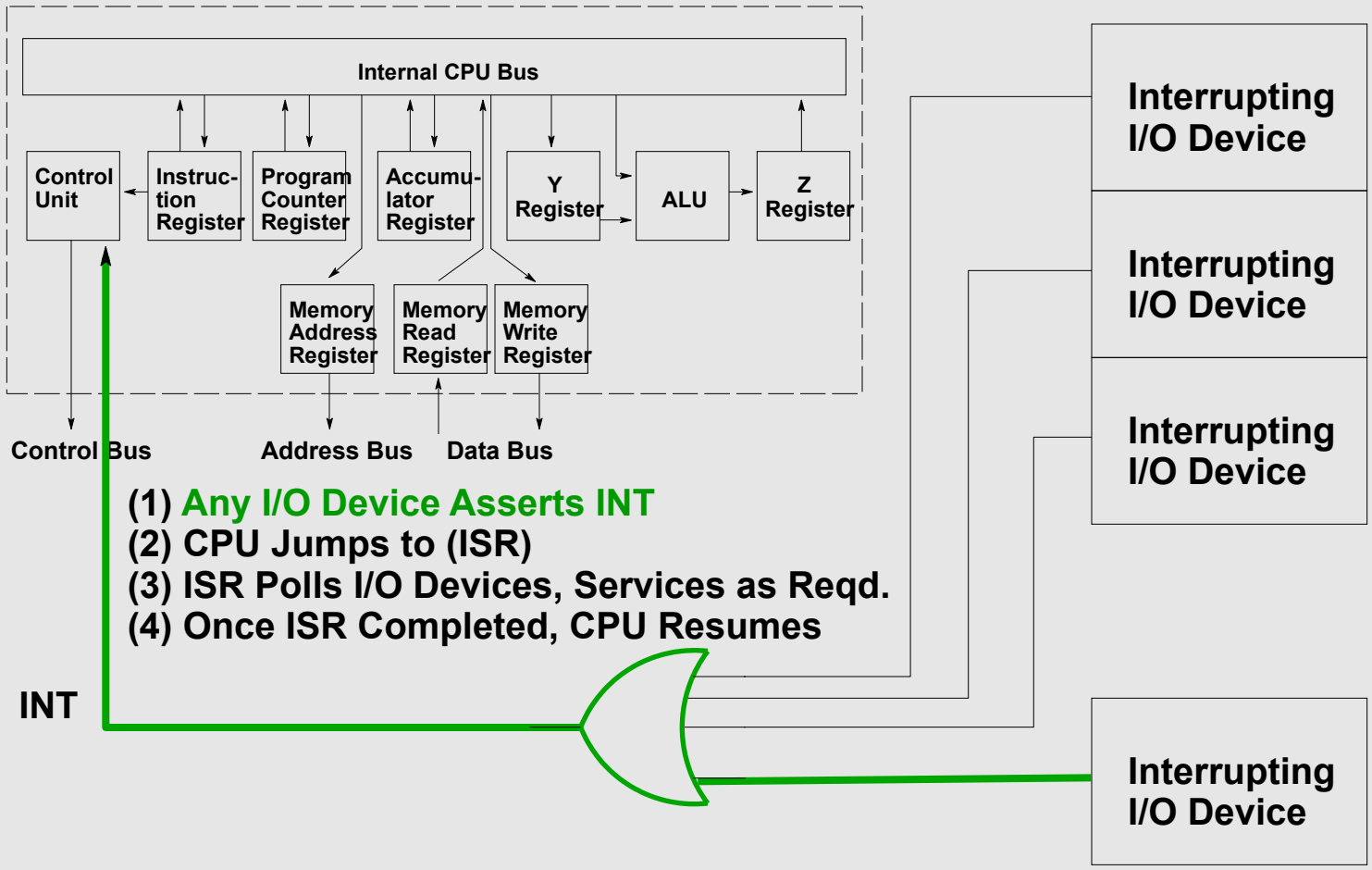


```
while running do begin
        fetchinstruction();
        decodeinstruction();
        executeinstruction();
        if interrupts_enabled and interrupt_line_set then
                begin
                :
                end;
end
```

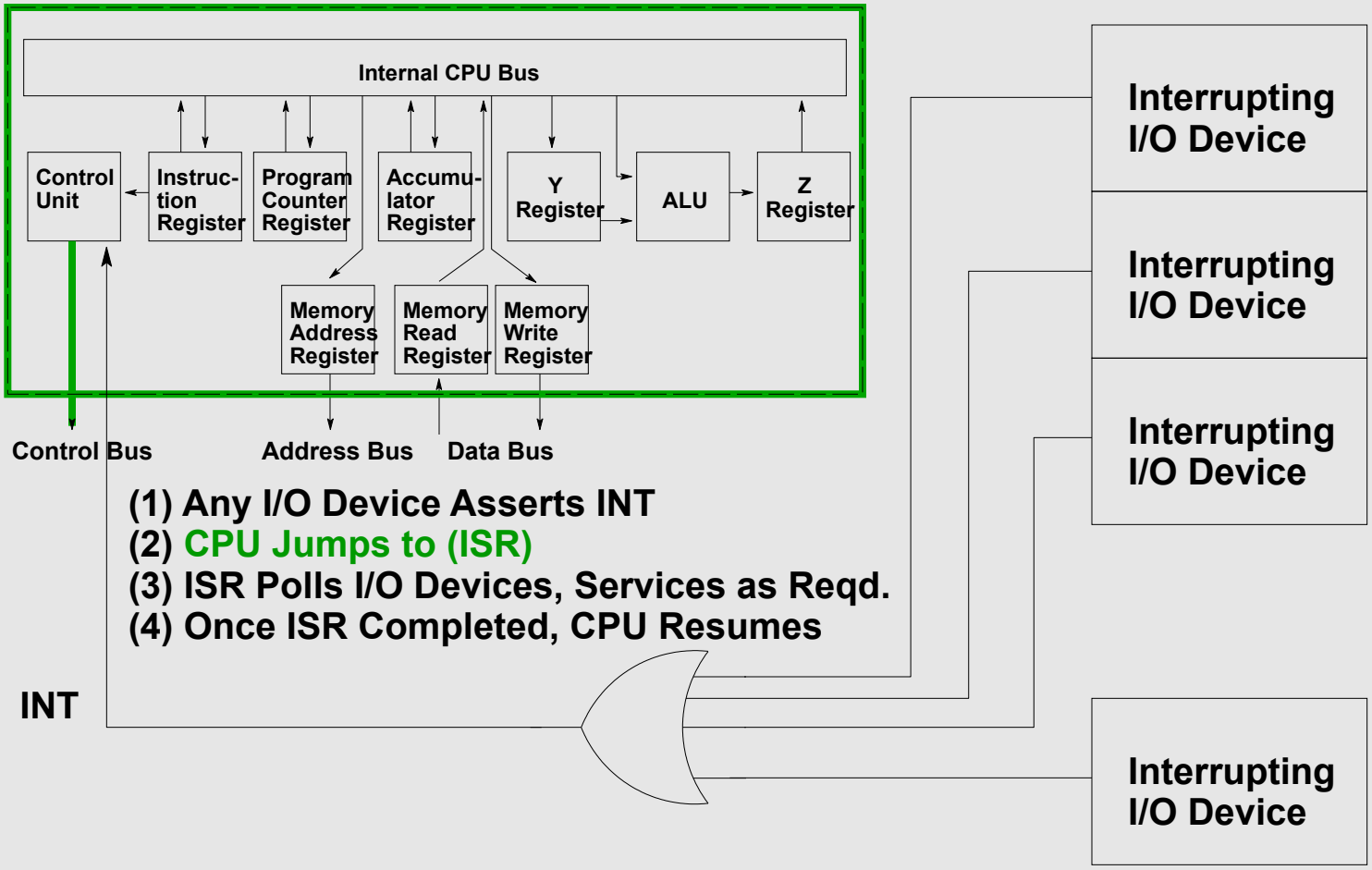
Servicing Interrupts - Polling



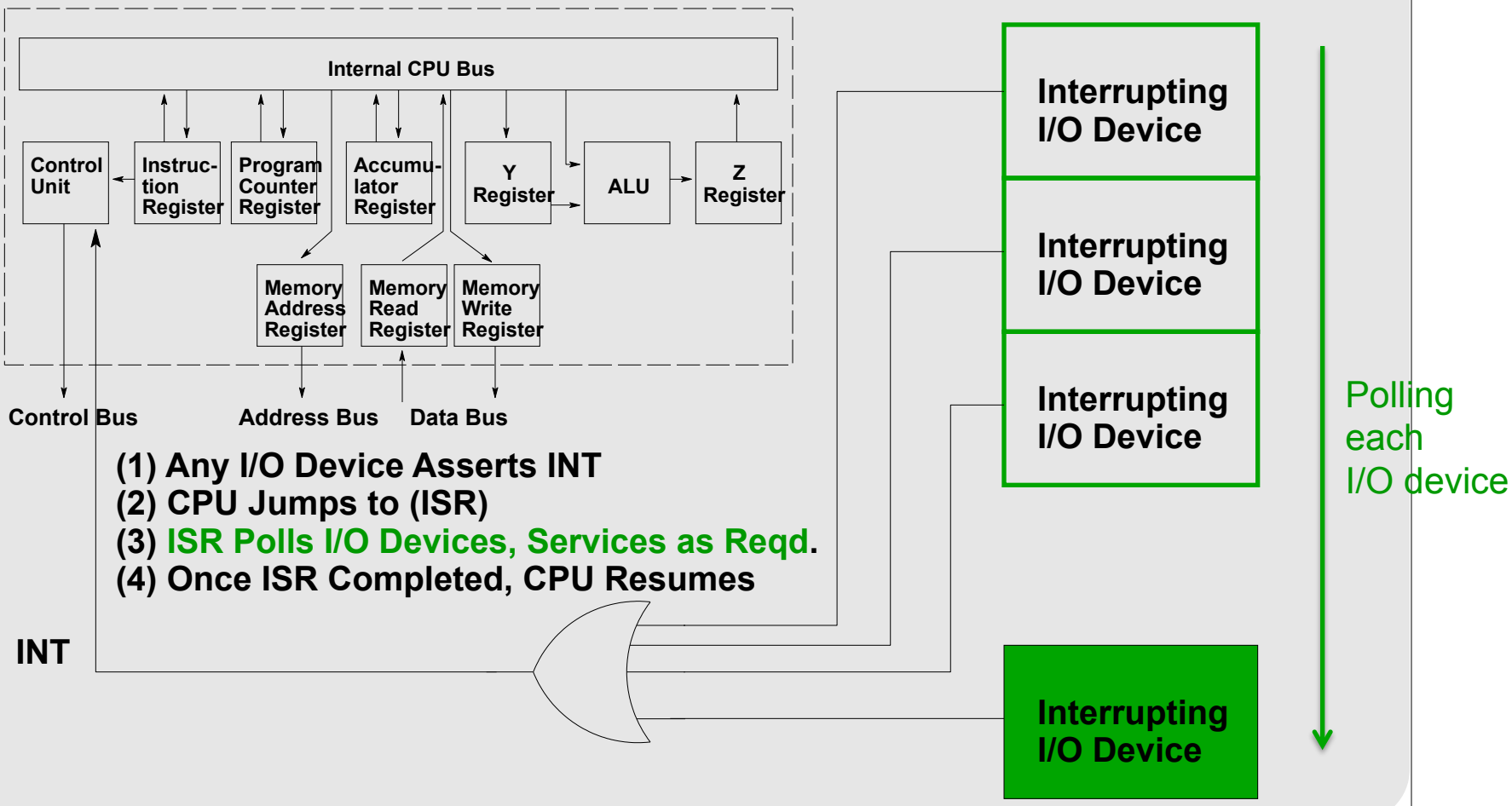
Servicing Interrupts - Polling



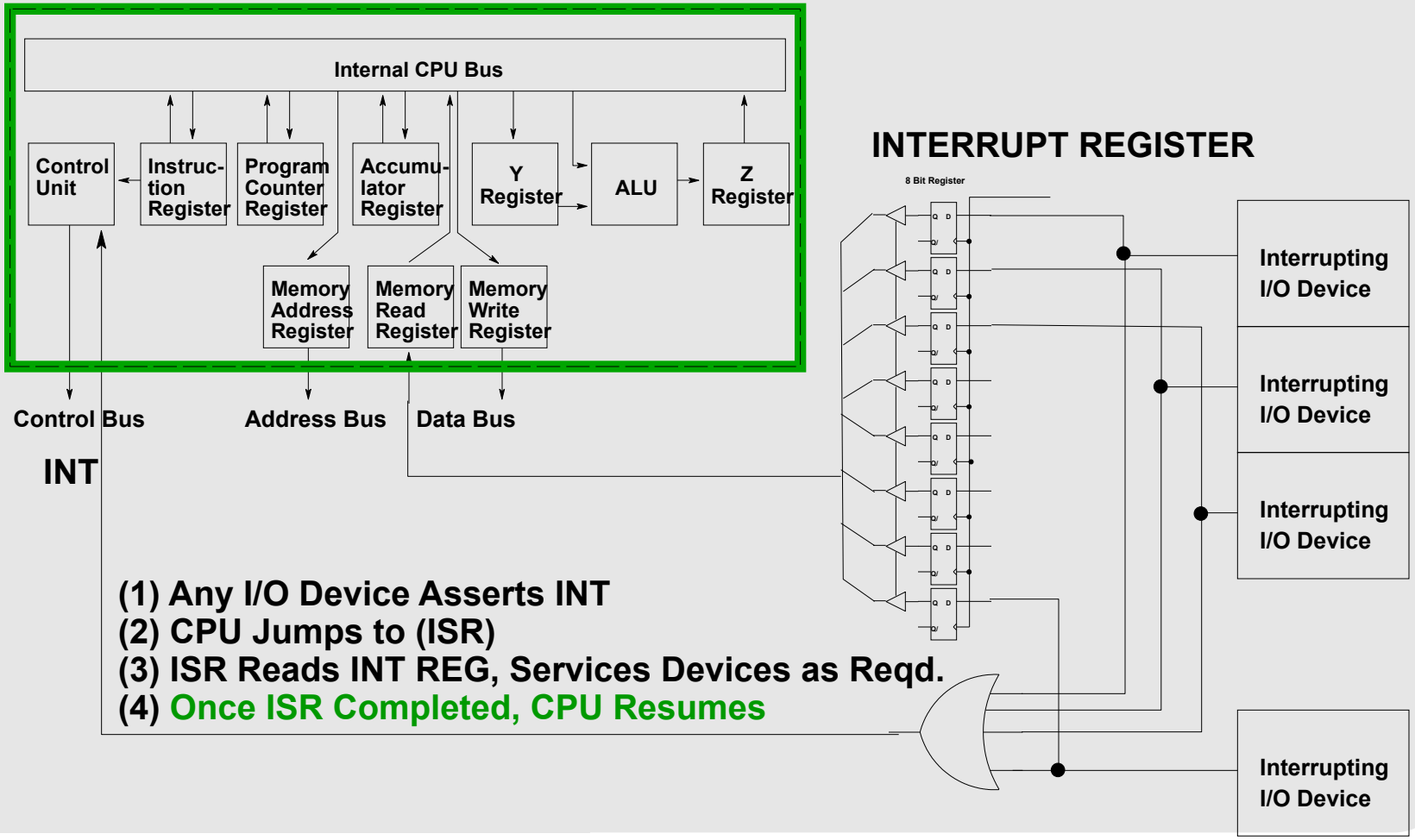
Servicing Interrupts - Polling



Servicing Interrupts - Polling



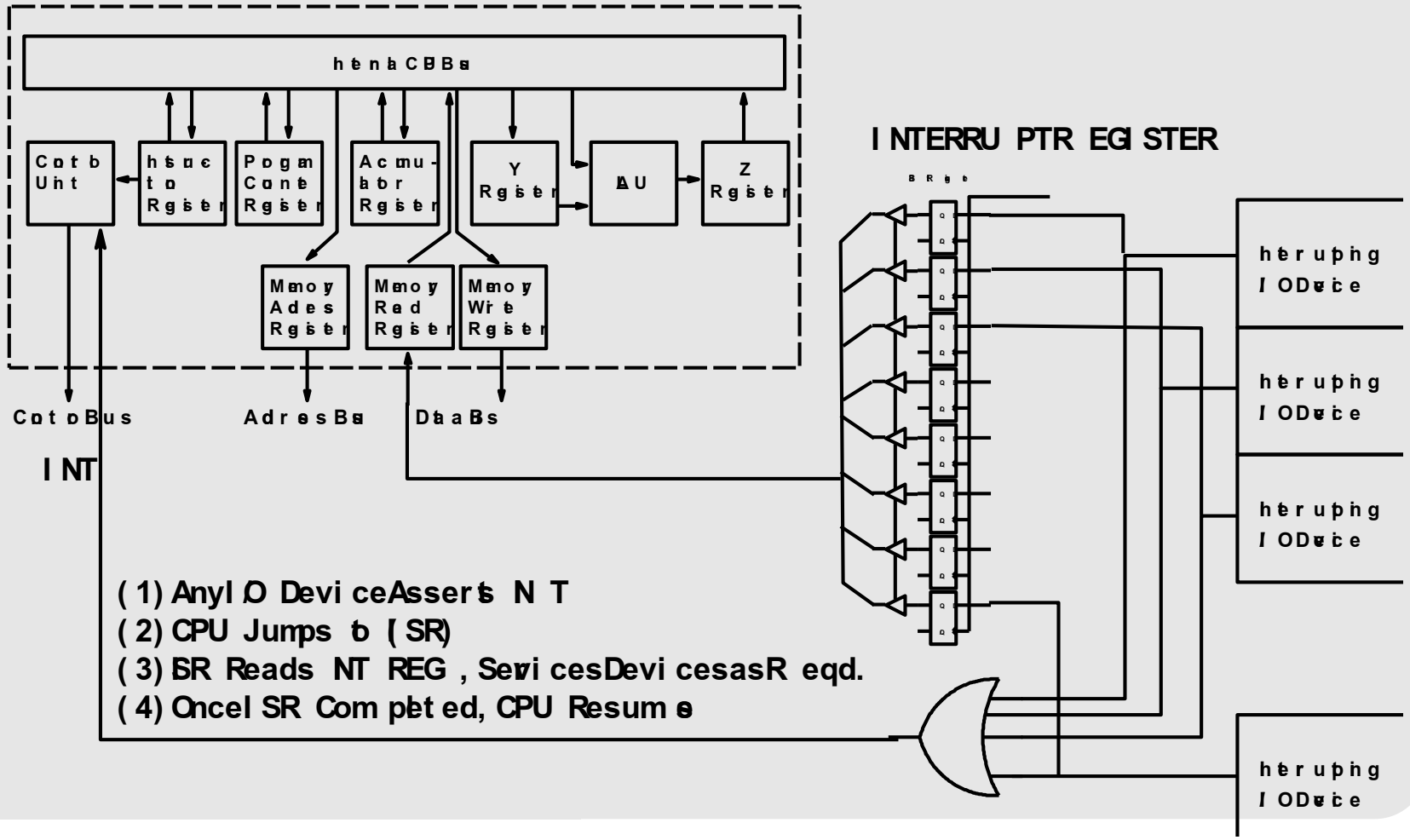
Servicing Interrupts via Register



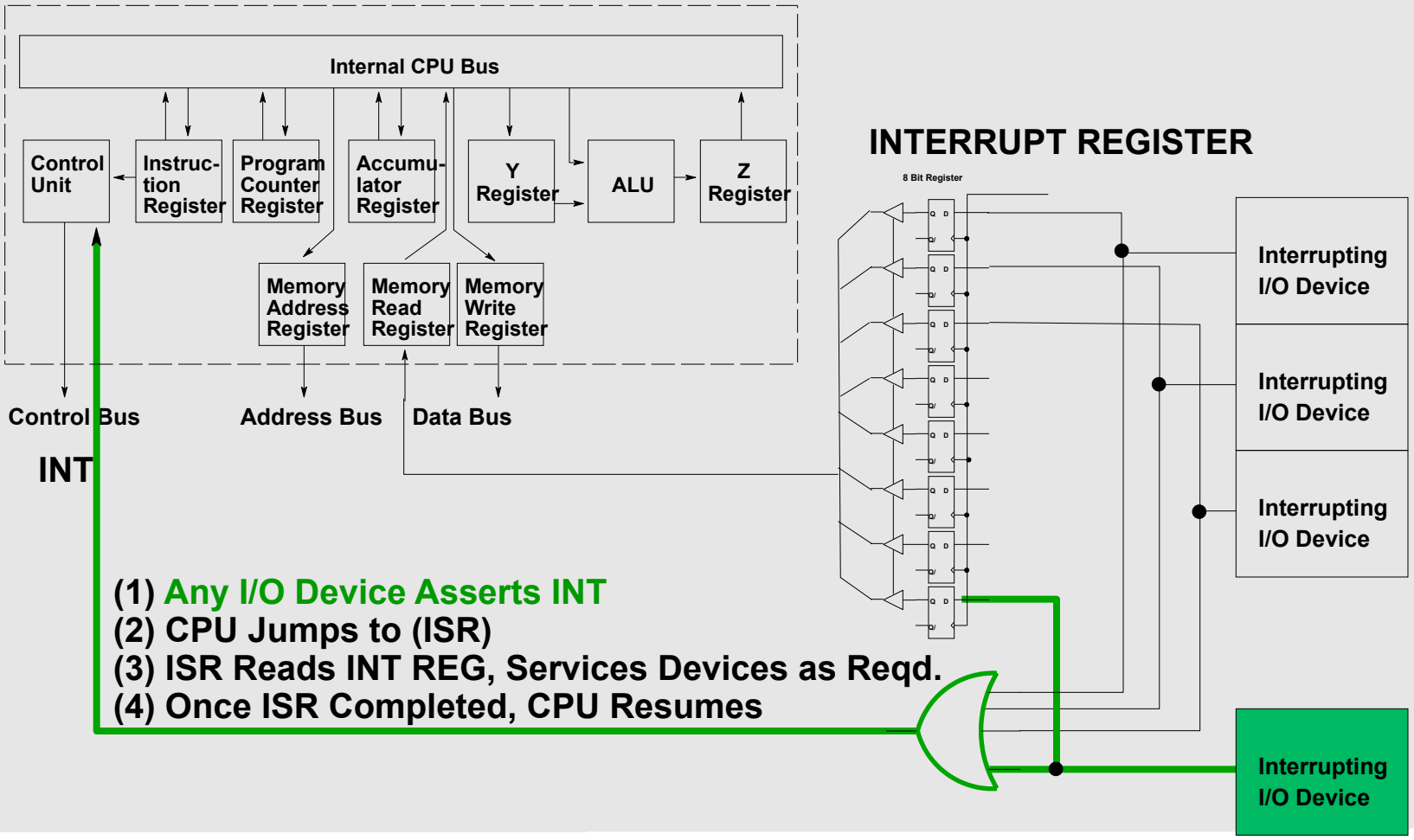
Servicing Interrupts via Register

- An improvement upon the basic interrupt polling method can be achieved by including within the system an “interrupt register”, in which each bit is wired to the interrupt line from a particular device.
- These lines are all ORed into the single “master interrupt” line into the CPU.
- When the CPU is interrupted, it reads the register and tests which of the bits is set, to select the interrupting device.

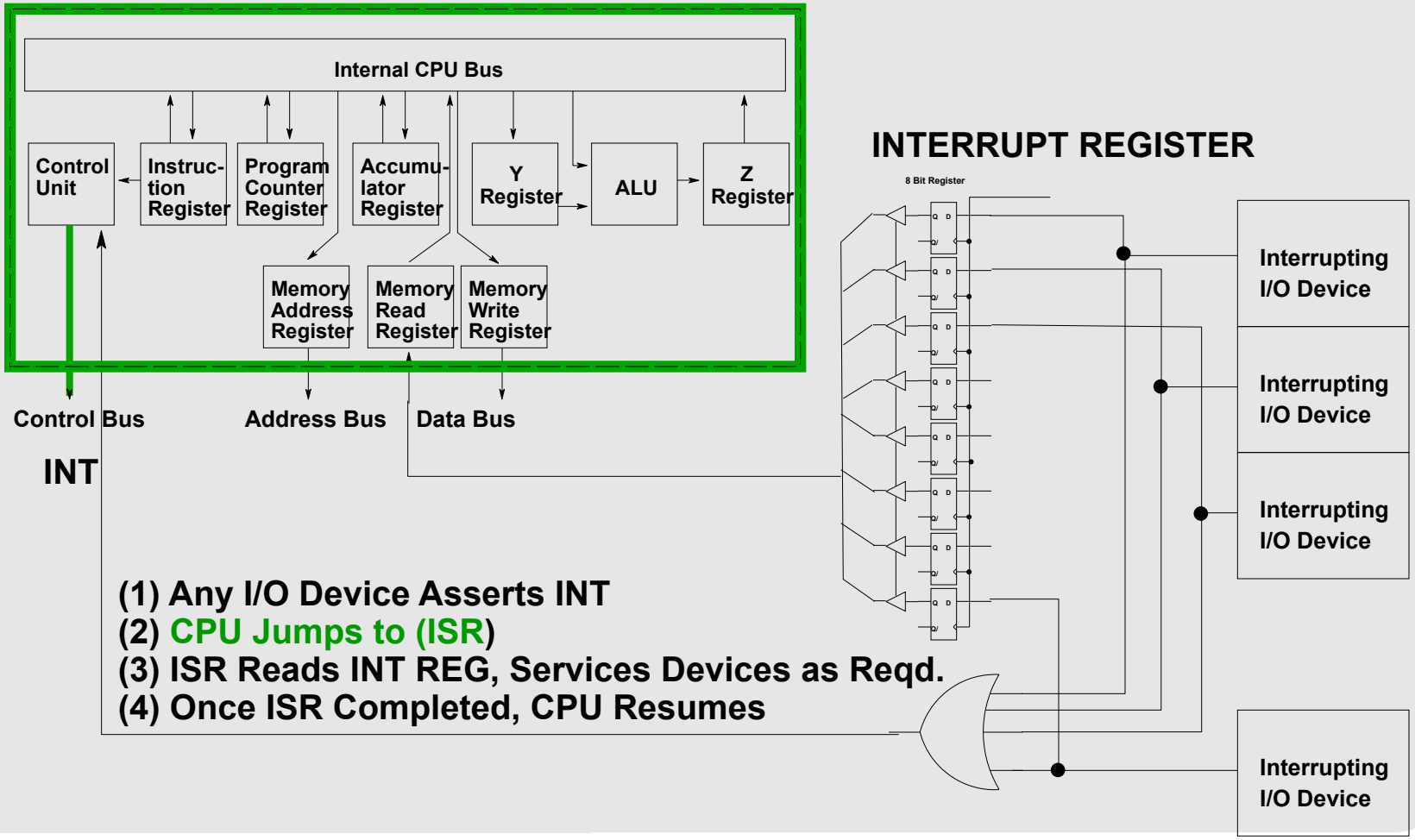
Servicing Interrupts via Register



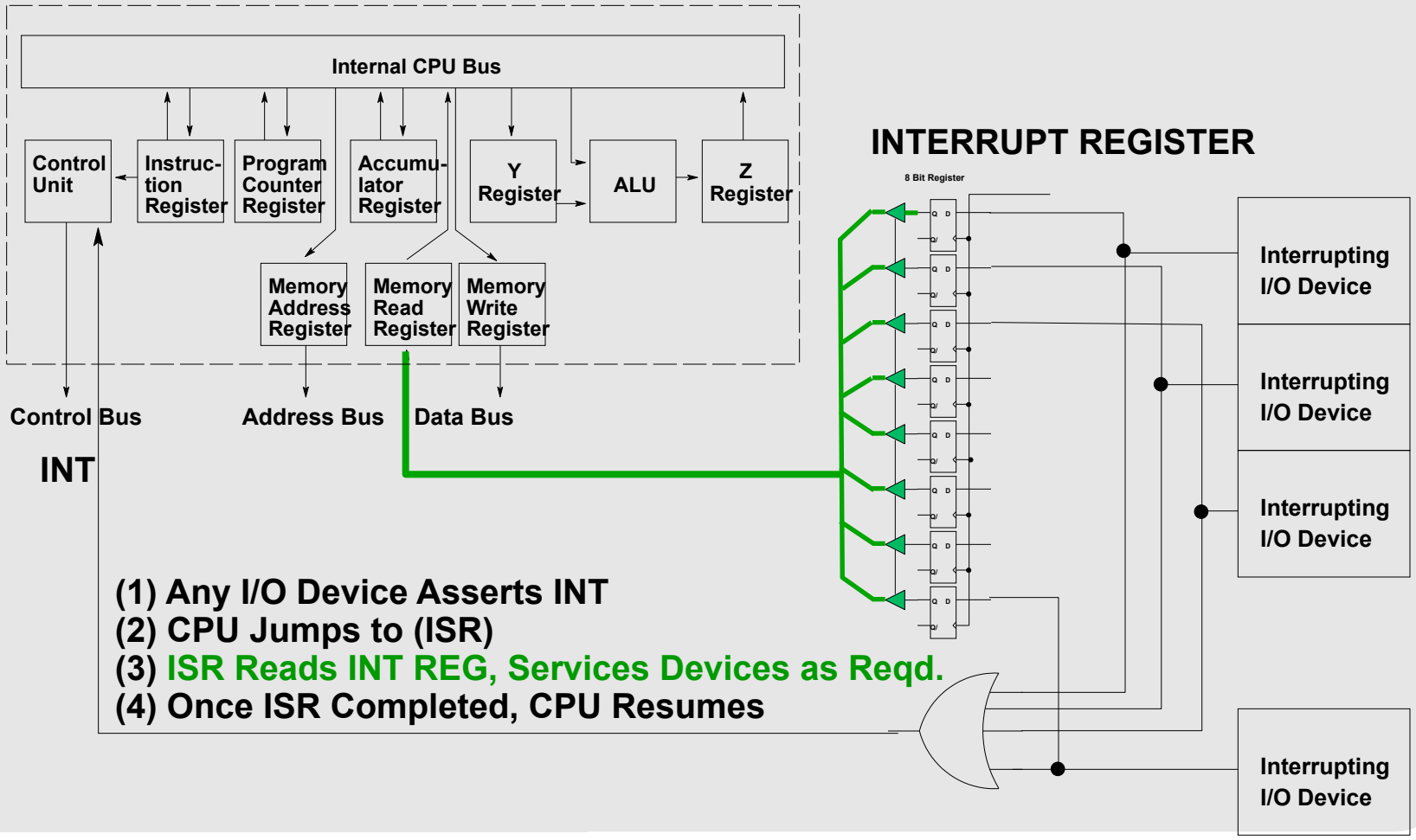
Servicing Interrupts via Register



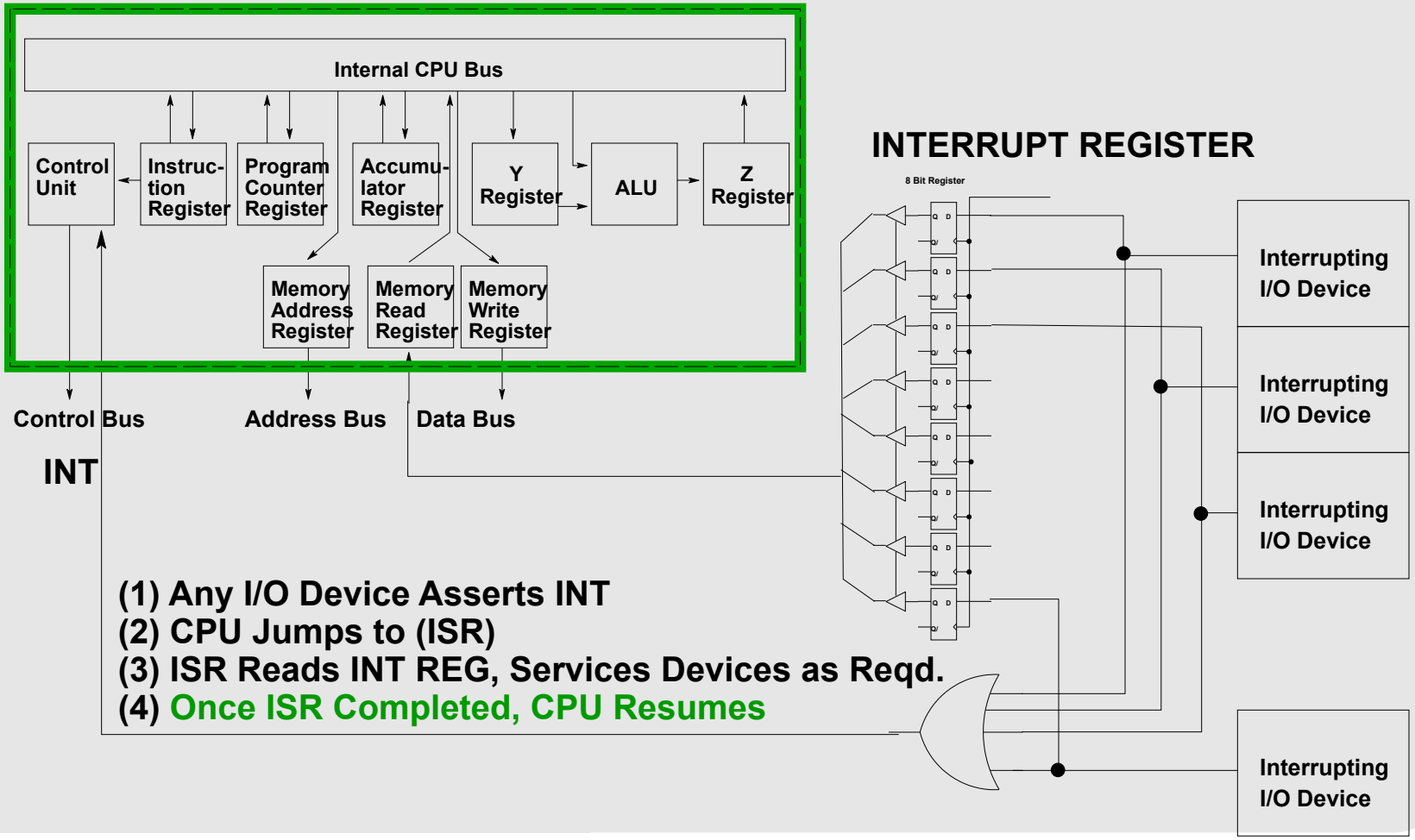
Servicing Interrupts via Register



Servicing Interrupts via Register

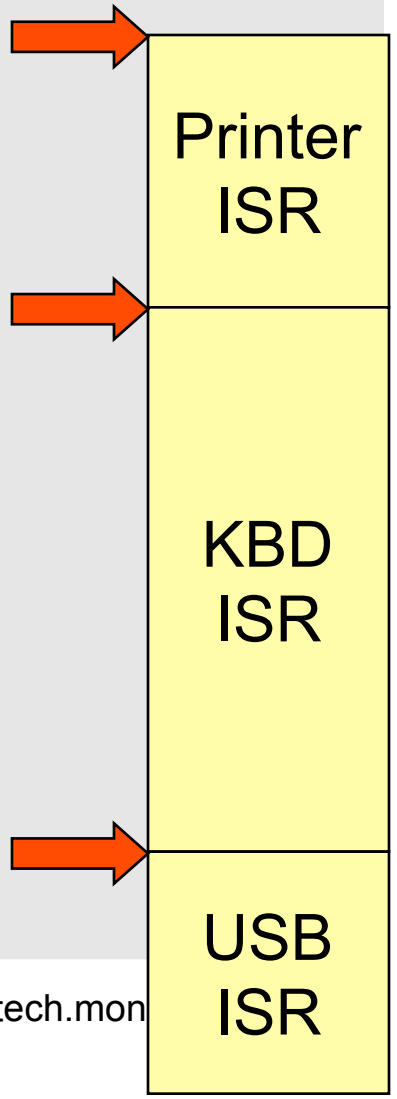


Servicing Interrupts via Register



Vectored Interrupts

- The most flexible technique for identifying an interrupting device is termed “interrupt vectoring”.
- In this arrangement, the interrupting device provides the CPU with the starting address of the ISR, so it can jump to the code required to service the interrupting device.
- Once the ISR has been executed, the CPU returns to executing the program which was interrupted.



Vectored Interrupt Alternatives

- Interrupt vectoring can be implemented in a number of ways.
- The hardware to resolve priorities between interrupting devices can be embedded within the CPU or reside within an external device.
- Priority resolution is necessary where multiple devices may interrupt at the same time, since only one ISR can be executed at any time.
- The interrupt vector which is produced by the hardware can point directly at the starting address of the ISR, or it may point to a "vector table" in memory.

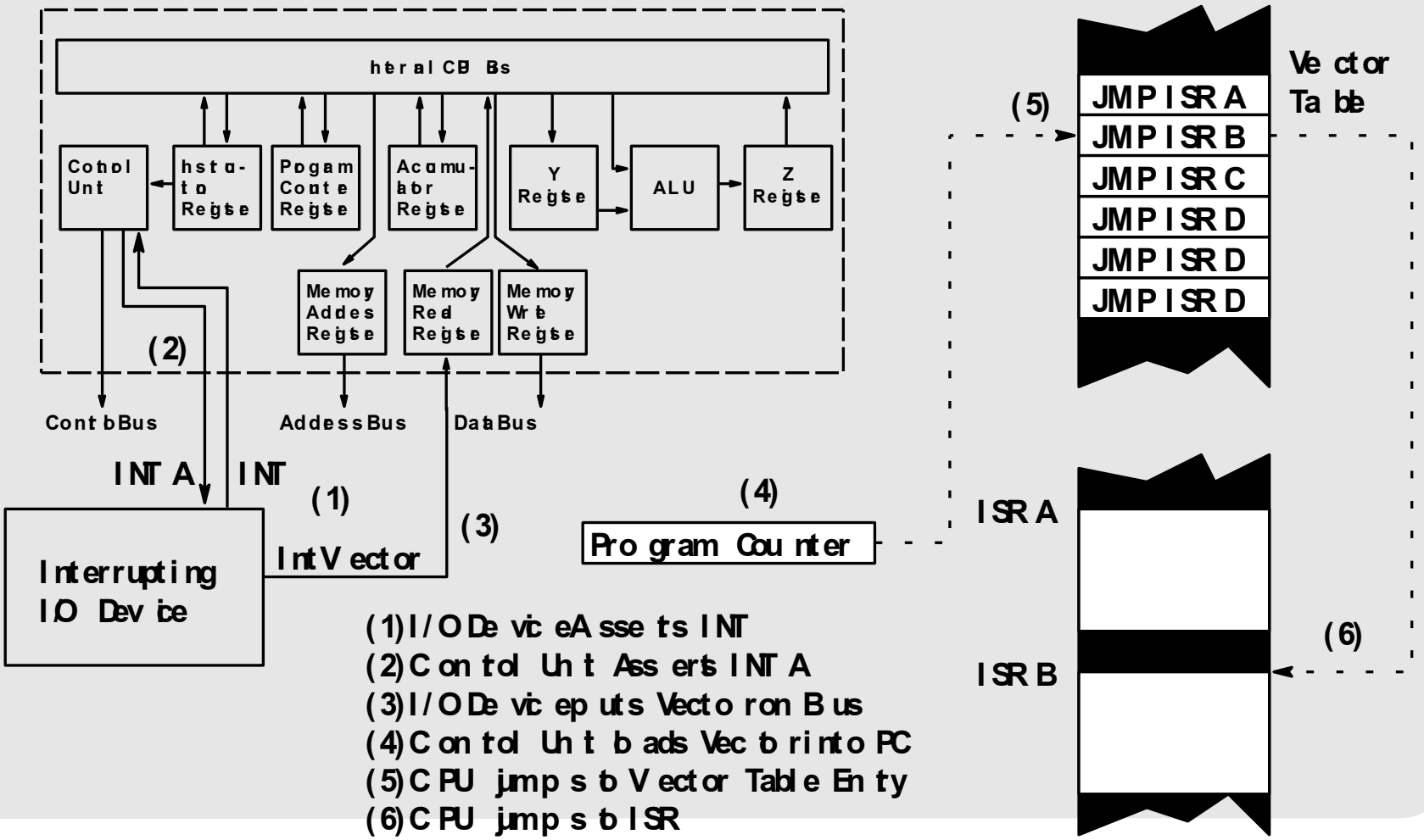
Vector Tables

- Where a vector table is used, each entry in the table is typically a jump instruction to the start of the ISR.
- The first instruction loaded into the IR is therefore a jump to the ISR proper, which then starts executing.

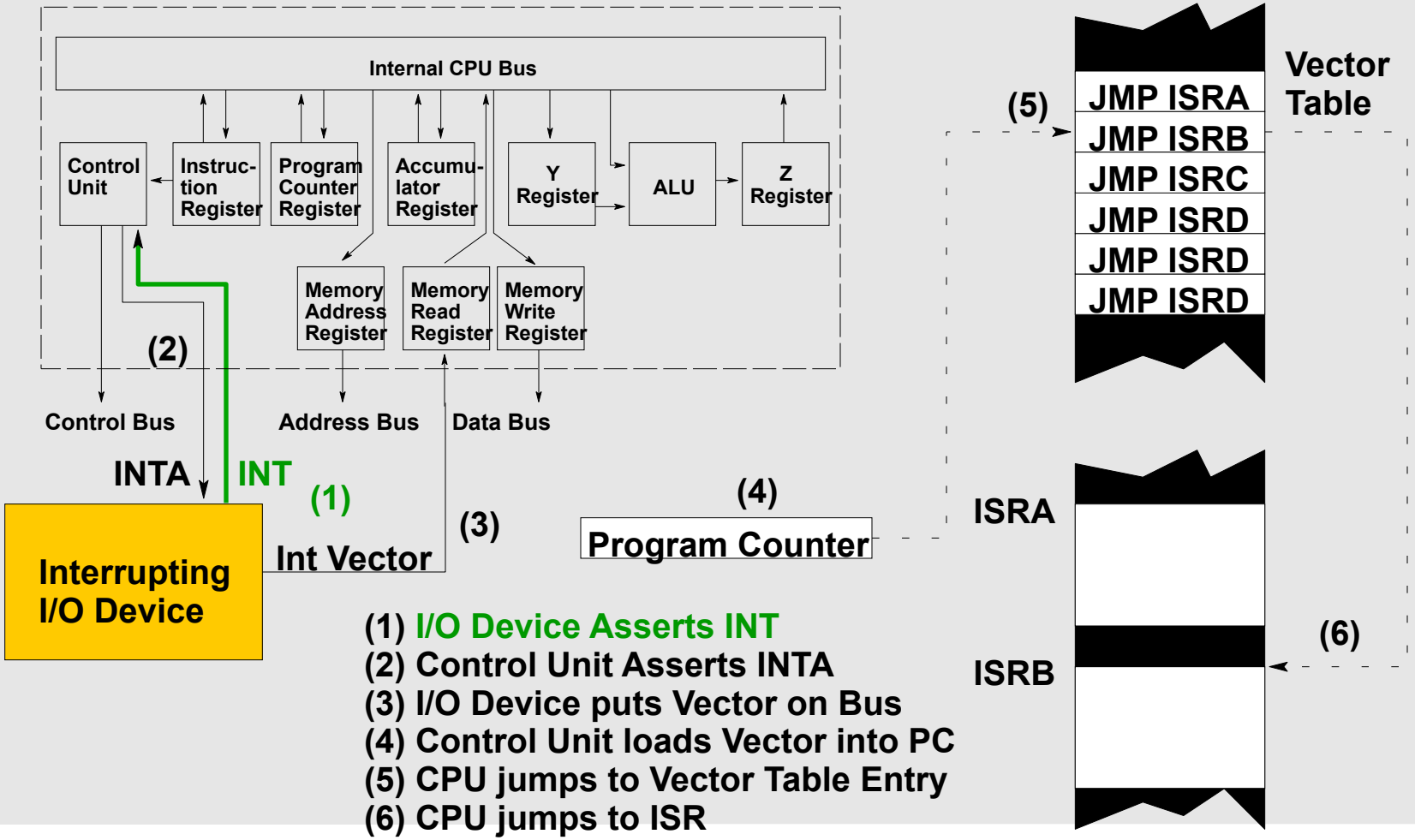
Vector Tables

- The advantages of using a vector table are that each entry occupies the same space in memory, and that multiple vectors can point to the same ISR. The latter can save memory, where many I/O devices of one type are used.
- The disadvantage of using a vector table is that the ISR must execute an additional jump instruction when it starts, and this can cost performance in a slower CPU.

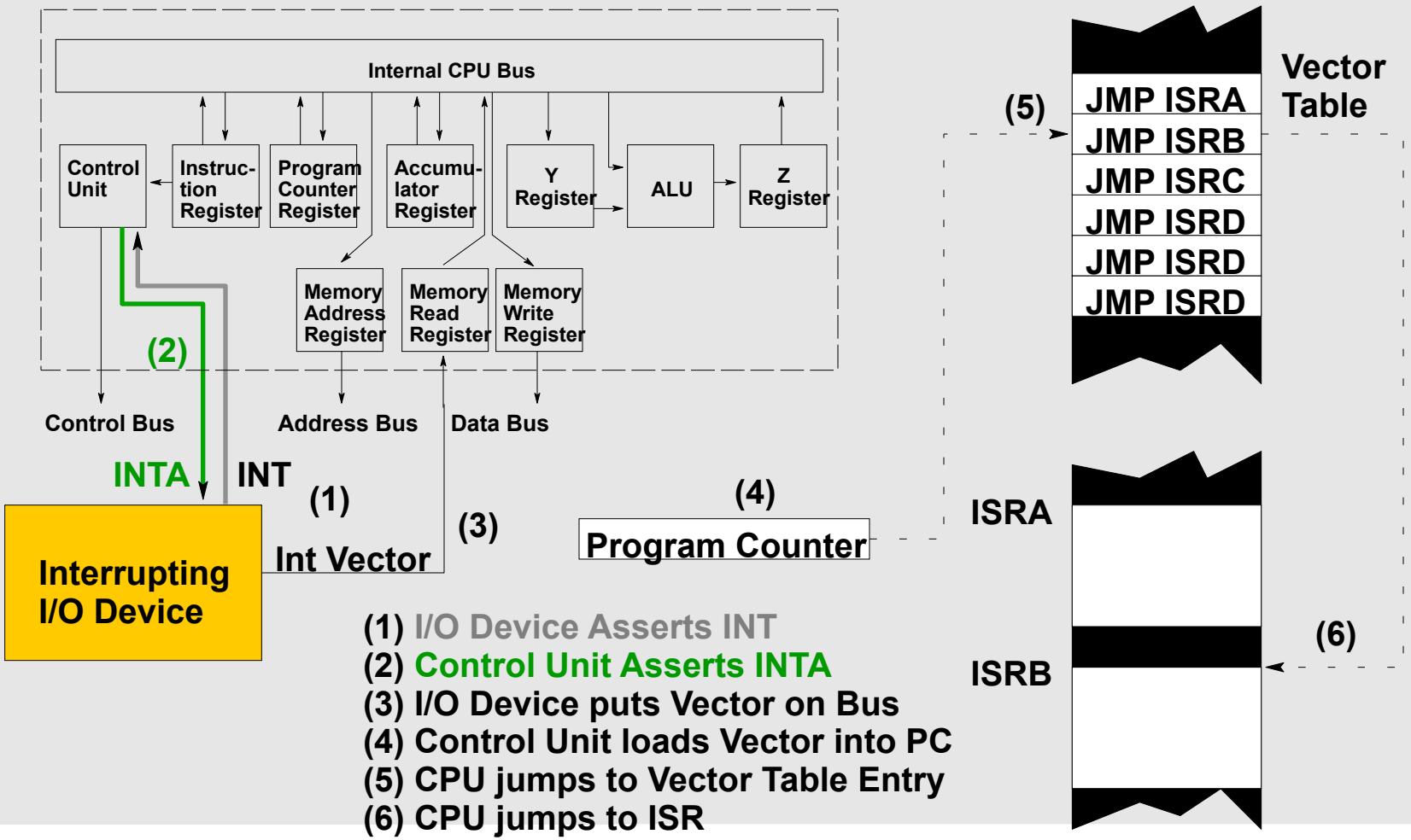
Vector Tables



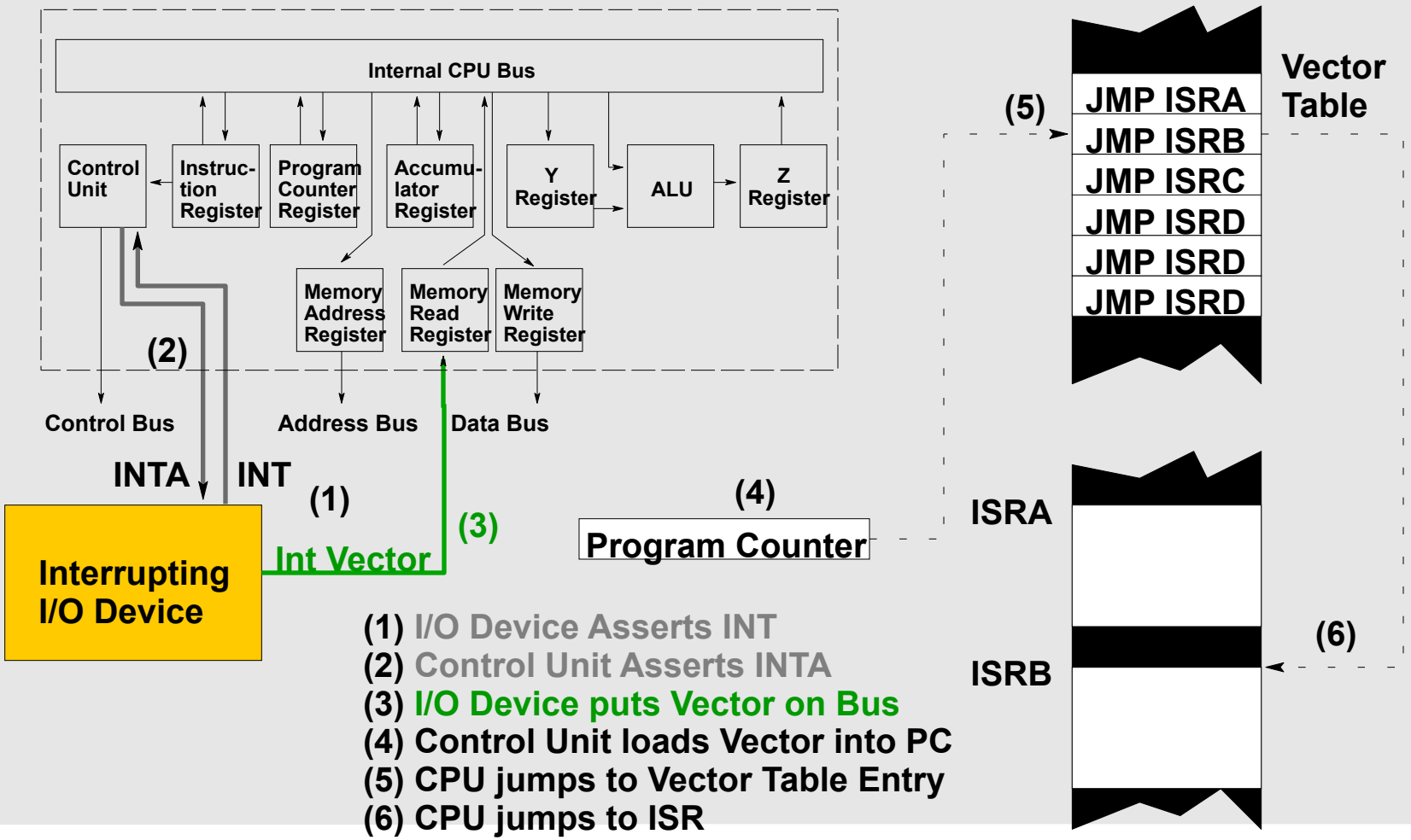
Vector Tables



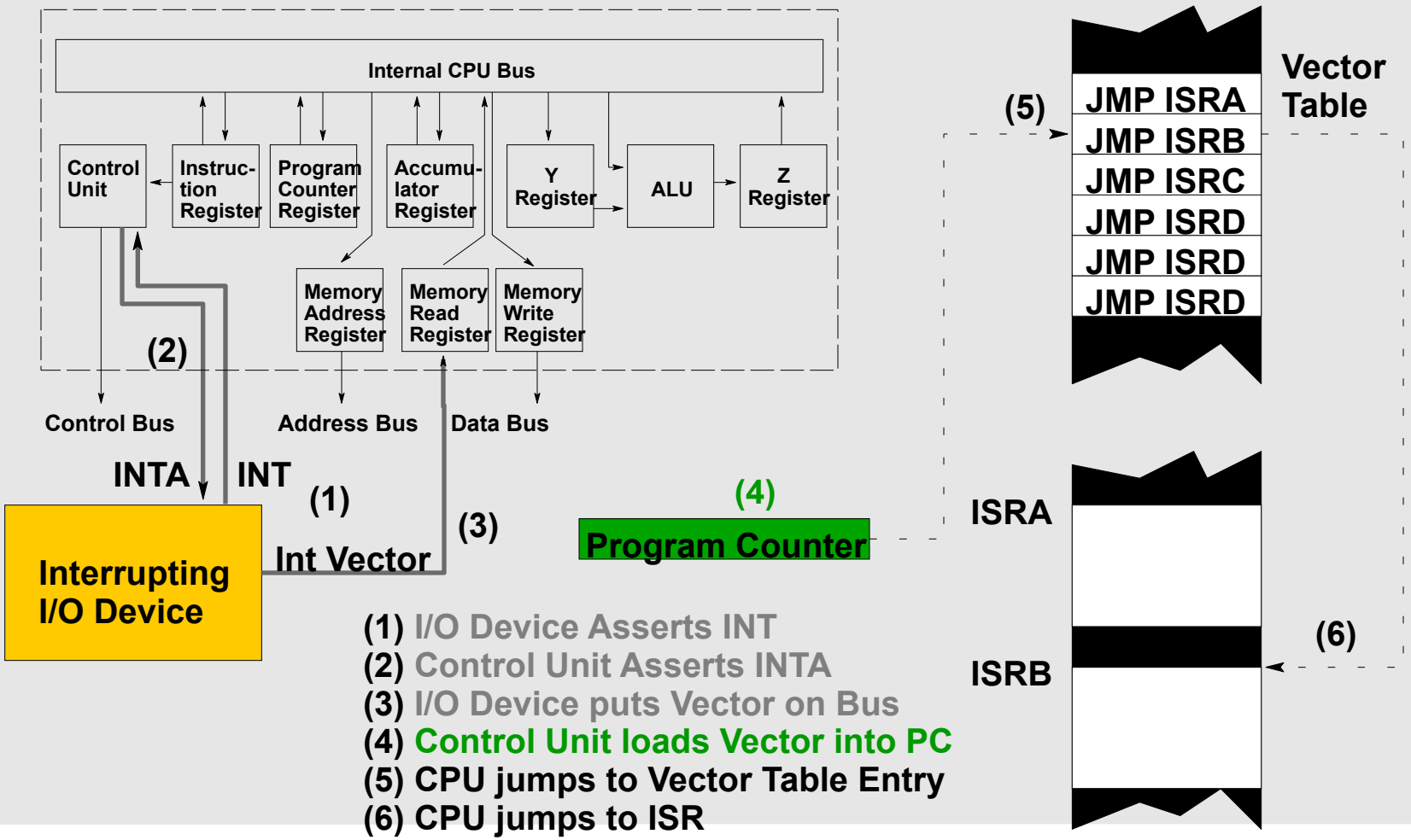
Vector Tables



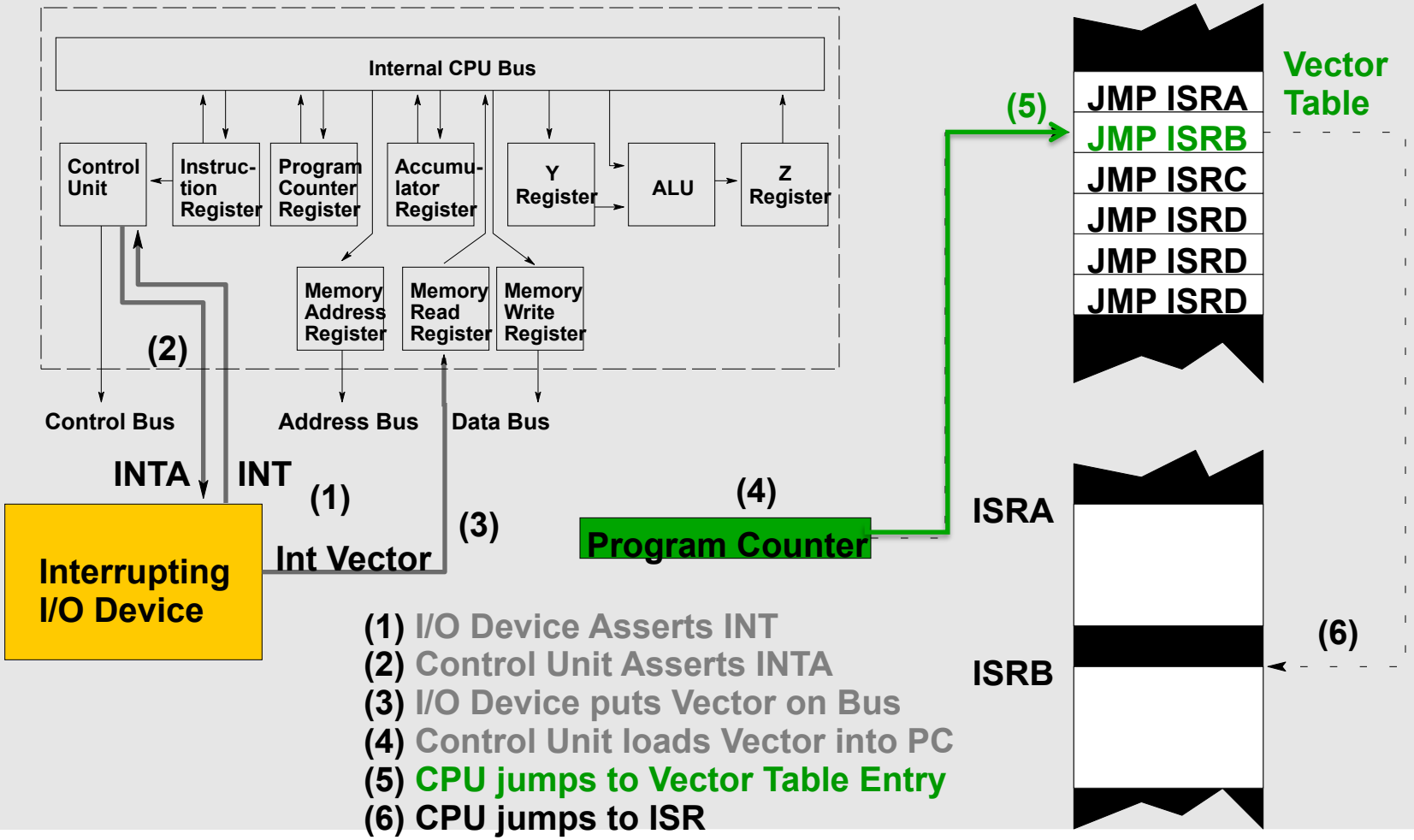
Vector Tables



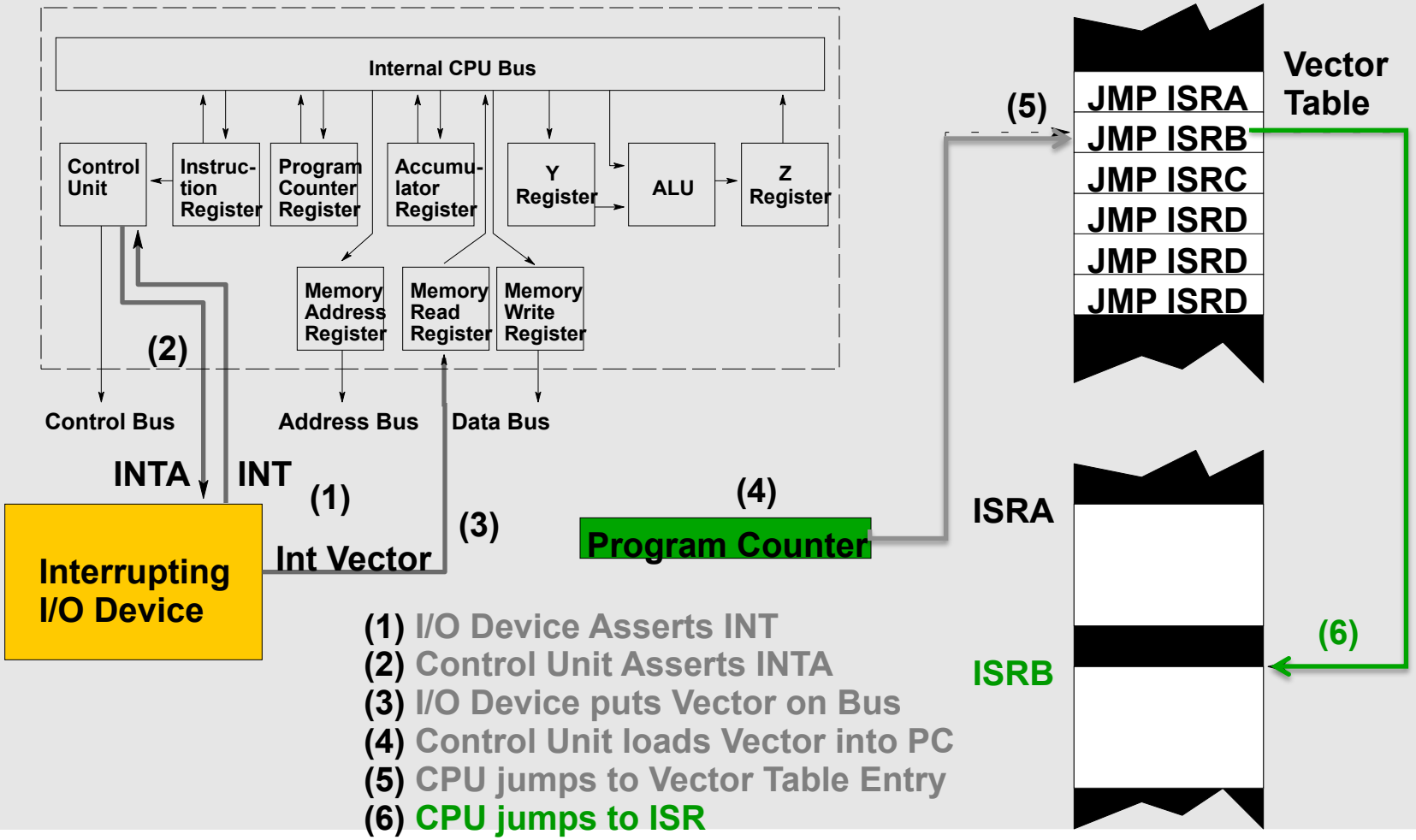
Vector Tables



Vector Tables



Vector Tables



Priority Resolution - Pre-emptive

- When several devices interrupt at once it is necessary to select one for servicing before the others can be attended to.
- Two schemes are commonly used:

1. In “priority pre-emptive” servicing, interrupts are assigned each a fixed priority. The interrupt with the higher priority is always serviced before those with lower priority.
2. This scheme is frequently used for hardware devices and “real-time” systems.

Priority Resolution - Round Robin

- In “round robin” scheduling, the first interrupt to arrive is serviced first.
- Sometimes this is implemented by dynamically altering the priorities in a “priority pre-emptive” scheme.
- Once serviced, the priority of that interrupt is shuffled to become the lowest in the group.
- This scheme is frequently used where the relative priorities of devices are not critical.

Interrupts while executing an ISR

- **What happens if the CPU is executing an ISR and another, higher priority interrupt arrives ?**
  - The simplest strategy is to disable all interrupts at the beginning of an ISR, and ignore the outside world until the ISR is finished. Therefore other interrupts must wait their turn, even if their priority is higher.
  - For some systems this is not useful. The interrupts are then disabled only for the first part of the ISR, when processor state is being saved, after which they are enabled again so the ISR can be interrupted like any other program.

Interrupting ISRs

- **Most systems provide for “interrupt masking”, whereby the software can set bits in an "interrupt mask register" to disable specific interrupts.**
- ***A programmer can use masking to allow higher priority interrupts to interrupt the ISRs of lower priority interrupts, while the ISR ignores interrupts of lower or equal priority.***

Software Interrupts

- **Many CPU architectures allow the software to also generate interrupts, by writing bits in a register, or executing special instructions.**
- **In many architectures, arithmetic errors, virtual memory conditions or special conditions such as a power failure can trigger an interrupt.**
- ***What is common to all such interrupts is that they result from software activity rather than external or internal I/O hardware.***

Direct Memory Access (DMA)

- DMA remains widely used to reduce the load on a CPU.
- When a CPU transfers a block of data between two areas in memory, it typically does so by copying data word by word between the source locations and destination locations, incrementing a counter variable placed in a register.
- The idea behind DMA is to put some dedicated hardware into the system, which performs such copy operations without CPU intervention.
- The CPU will initialise the “DMA Controller” with the source address, destination address and word count for the transfer.
- After the CPU starts the DMA controller, it will then perform the copy, and interrupt the CPU once the copy is completed.

Exceptions, Machine Checks, Traps

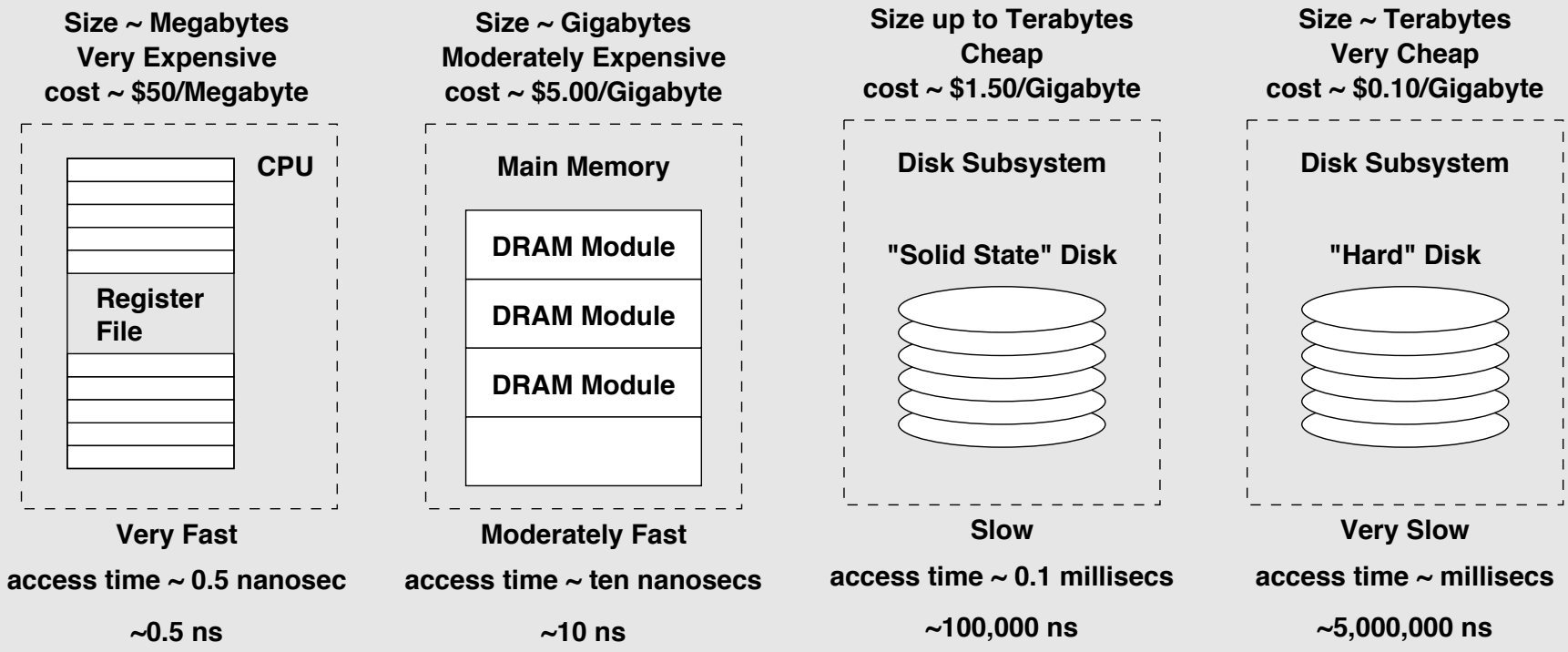
- **Software generated interrupts are usually termed “exceptions”, “machine checks”, “traps” or “software interrupts”.**
- **They are typically serviced in in a similar manner to hardware generated interrupts, and are incorporated into the CPU interrupt priority resolution scheme.**
- **Examples:**
  - Unix “bus error” or
  - Microsoft “BSOD”.

Mass Storage
and
Storage Hierarchies

Storage Hierarchies

- **Storage hardware varies in performance, and price per Megabyte – strong Moore's Law impact:**
  - The fastest storage devices are registers within the CPU, which are typically also the most expensive. Access times can be as short as 0.5 nanosecond in a CPU with a 2 GHz clock.
  - Main memory is significantly slower than a register file, and typically has access times of the order of tens of nanoseconds. The cost per Gigabyte is modest, ~$10 – $100/Gigabyte.
  - Solid State Disk (SSD) based on FLASH RAM devices is slow with access times of ~0.1 milliseconds, with sizes up to Terabytes, and is more expensive than hard disks.
  - Disk storage is very slow with access times of milliseconds, in comparison with registers and memory, but is also very cheap,~$0.10/Gigabyte.

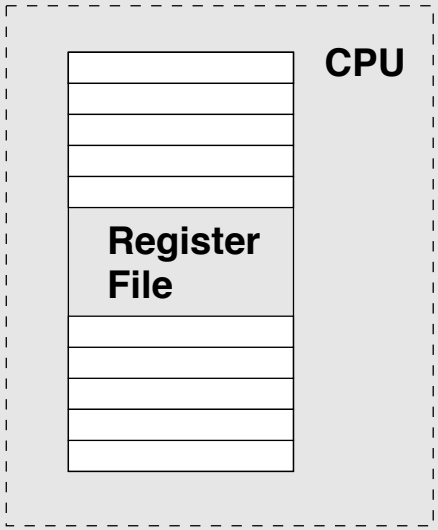
Storage Hierarchies



Note the trade between capacity and cost per capacity;
This tradeoff evolves due to Moore's and Kryder's Laws

Storage Hierarchies – One Decade Ago

Size ~ hundreds of bytes
Very Expensive
cost ~ $2000/Megabyte

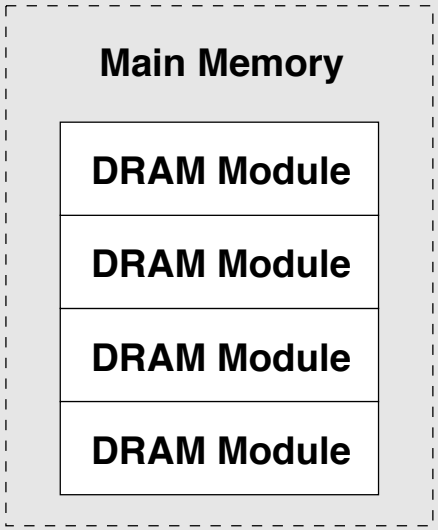


Very Fast

access time ~ nanosecs

~1 ns

Size ~ hundreds of Megabytes
Moderately Expensive
cost ~ $1.50/Megabyte

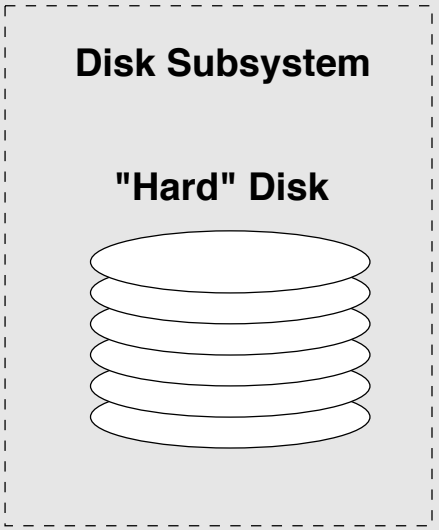


Moderately Fast

access time ~ tens of nanosecs

~50 ns

Size ~ tens of Gigabytes
Very Cheap
cost ~ $0.15/Megabyte



Very Slow

access time ~ millisecs

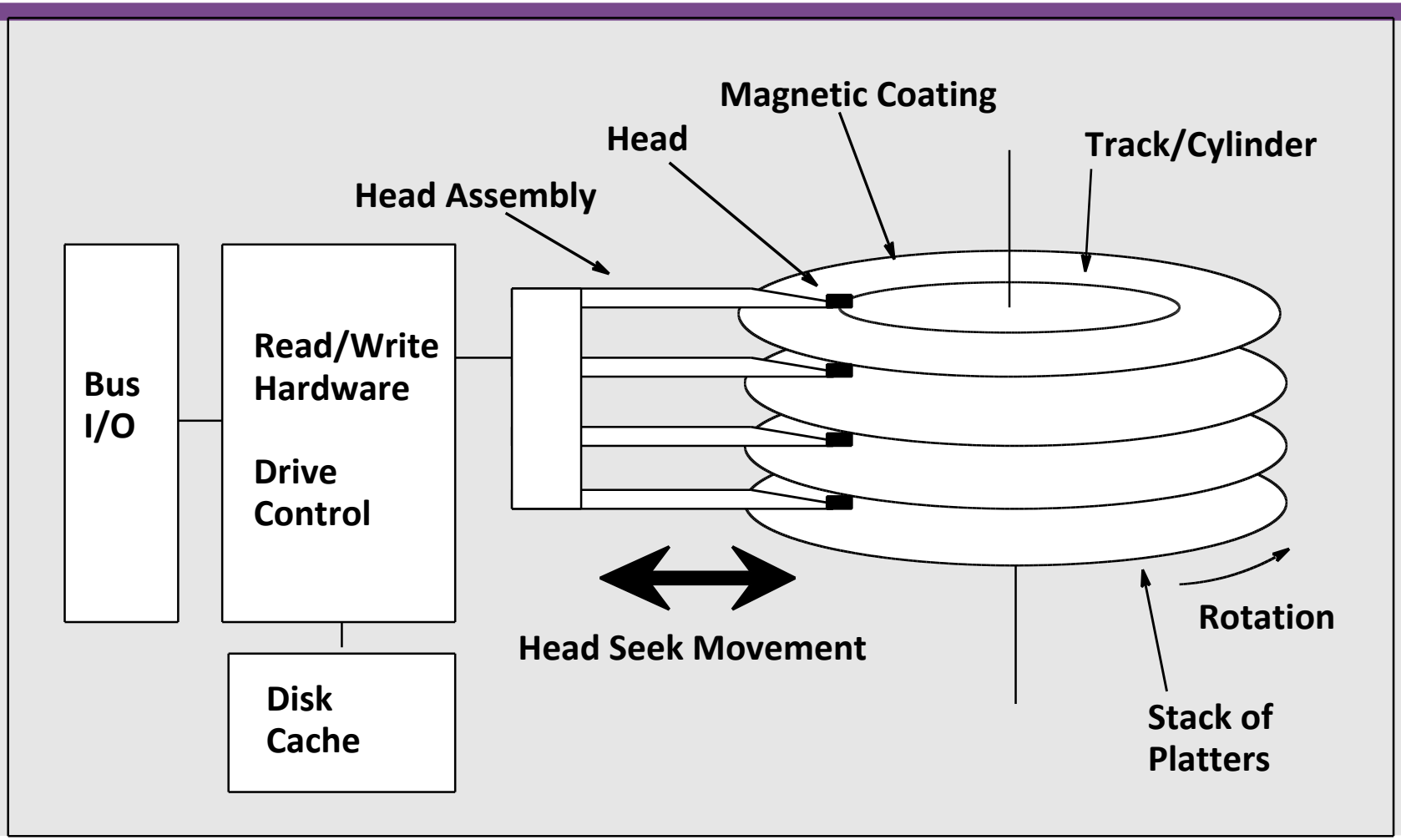
~5,000,000 ns

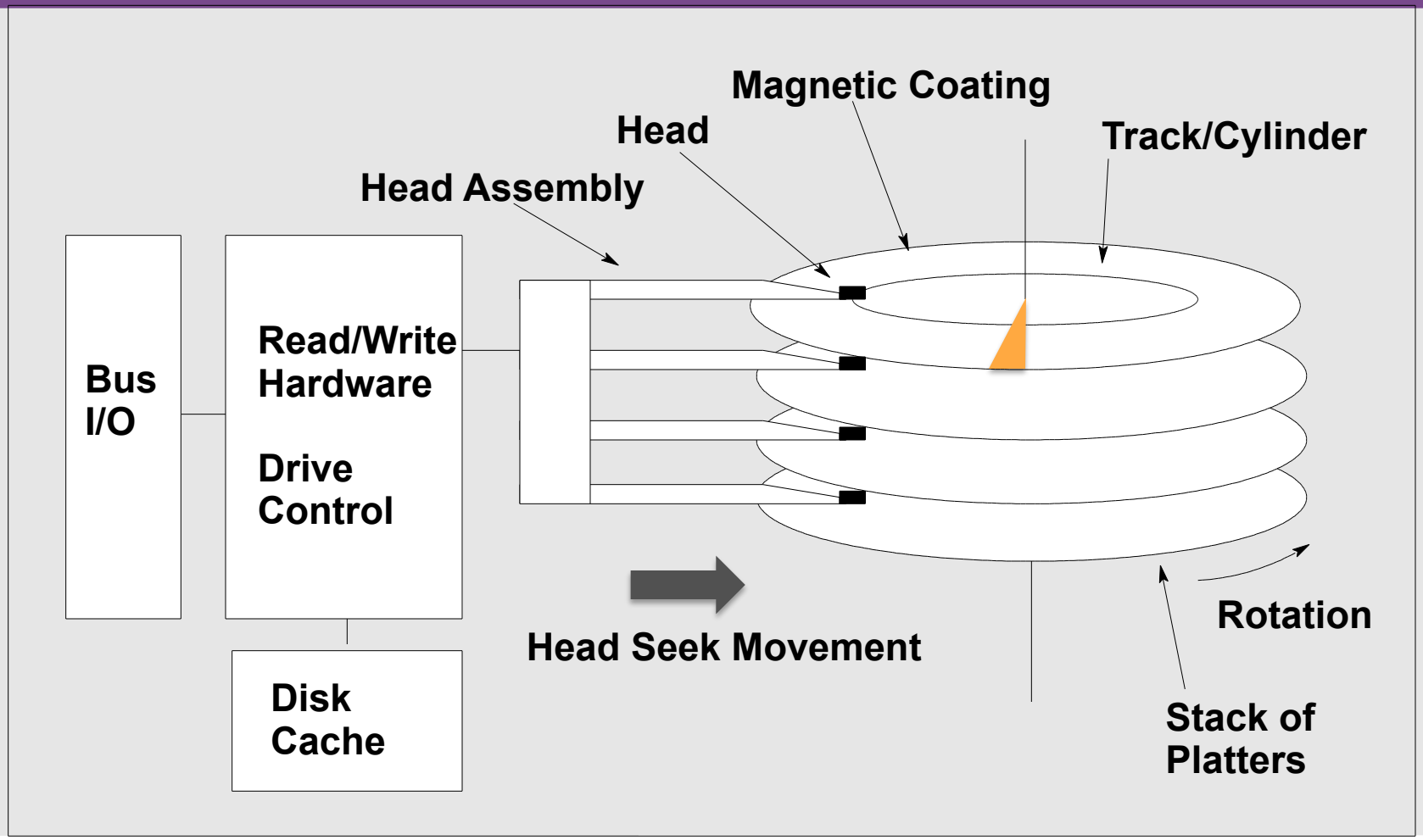
Rotating Disk Operating Principles

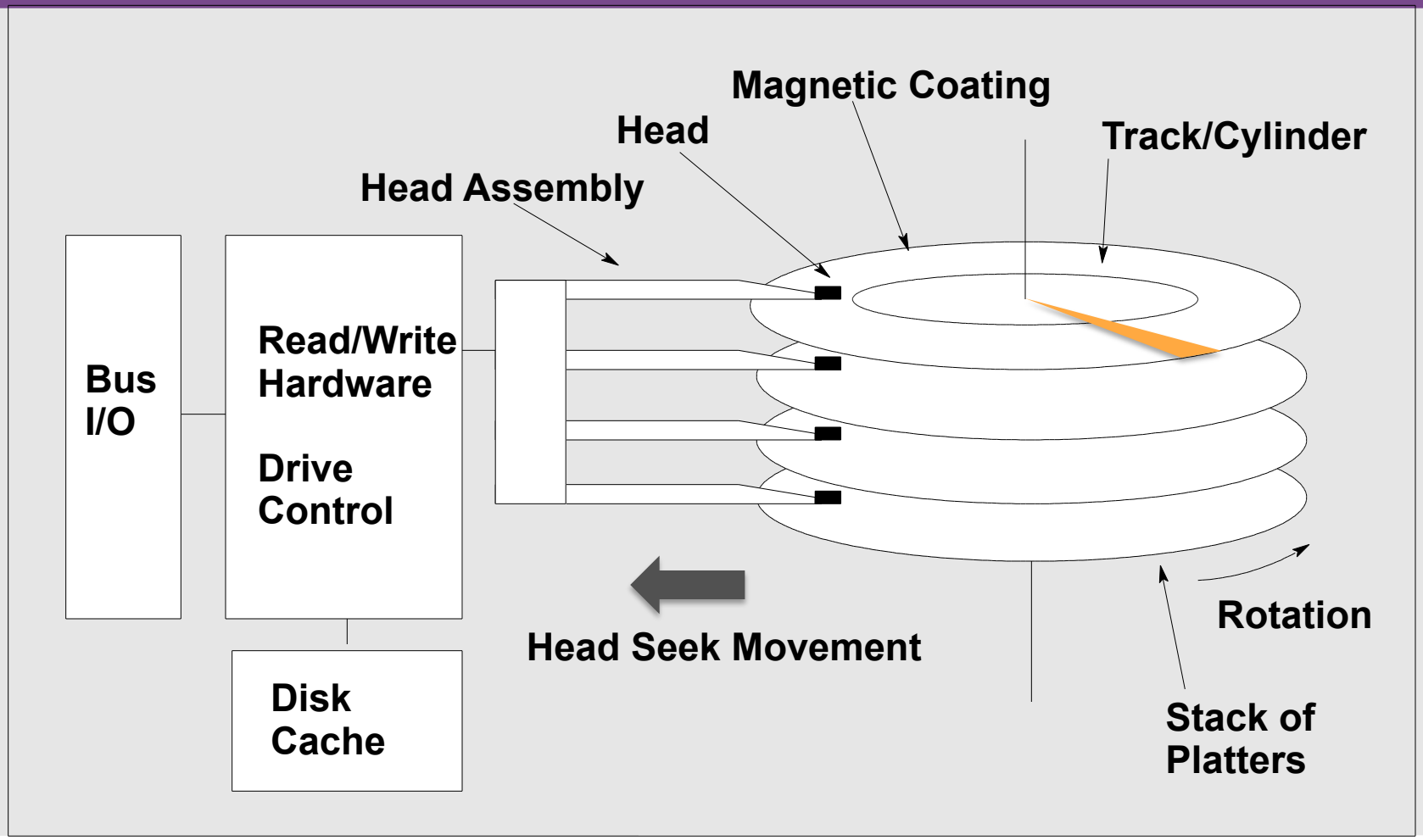
- The “Hard Disk”, “Fixed Disk” or “Winchester Disk” is the most commonly used mass storage device in modern systems;
- Winchester Disks replaced older “Removable Disks” which were popular during the 1960s, 1970s and 1980s;
- Contemporary Winchester Disk Drives can store thousands of Gigabytes / Terabytes in a single 3.5” or 2.5” wide package;
- Density of data storage in hard disks has grown exponentially over time – Kryder’s Law.

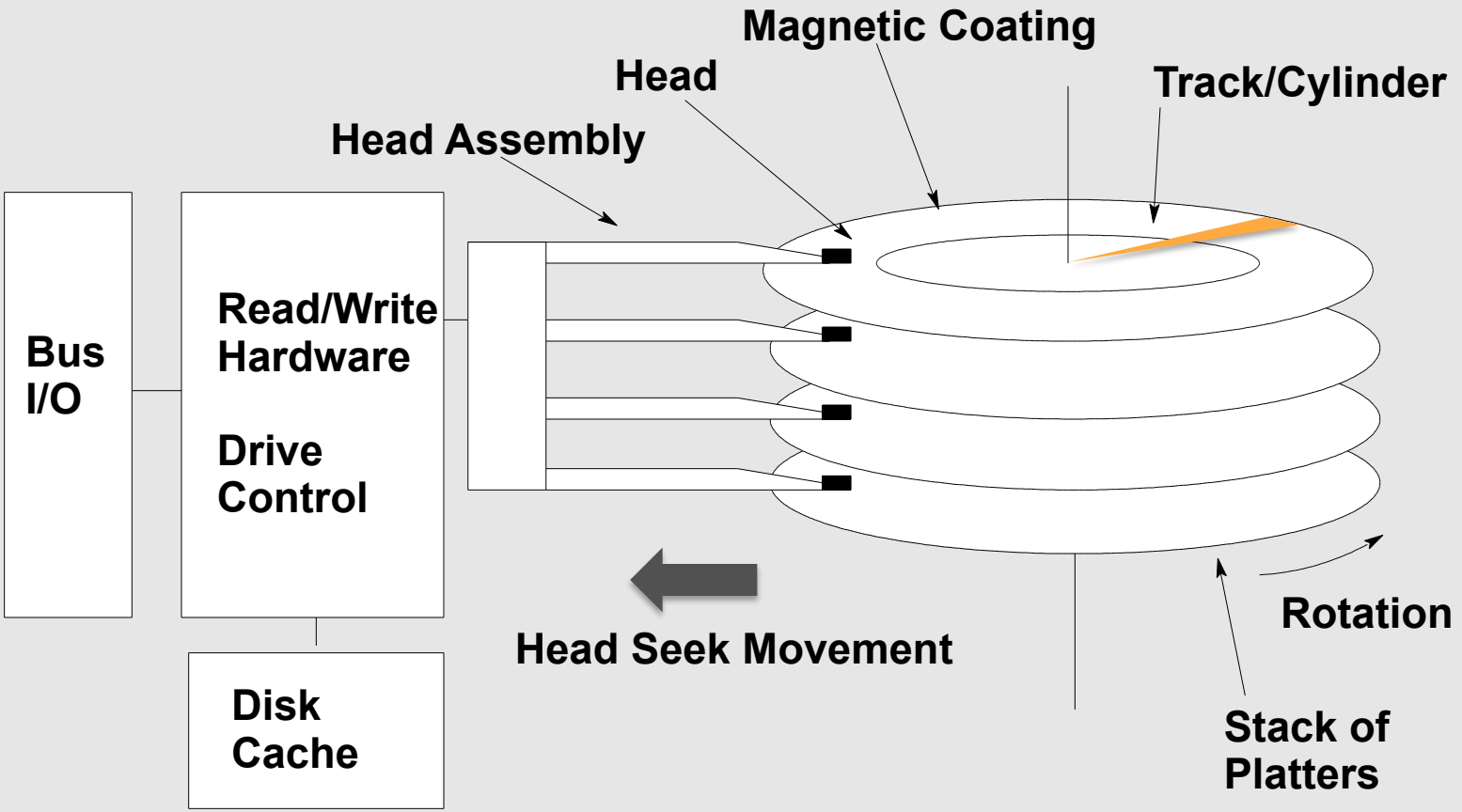
Rotating Disk Operating Principles

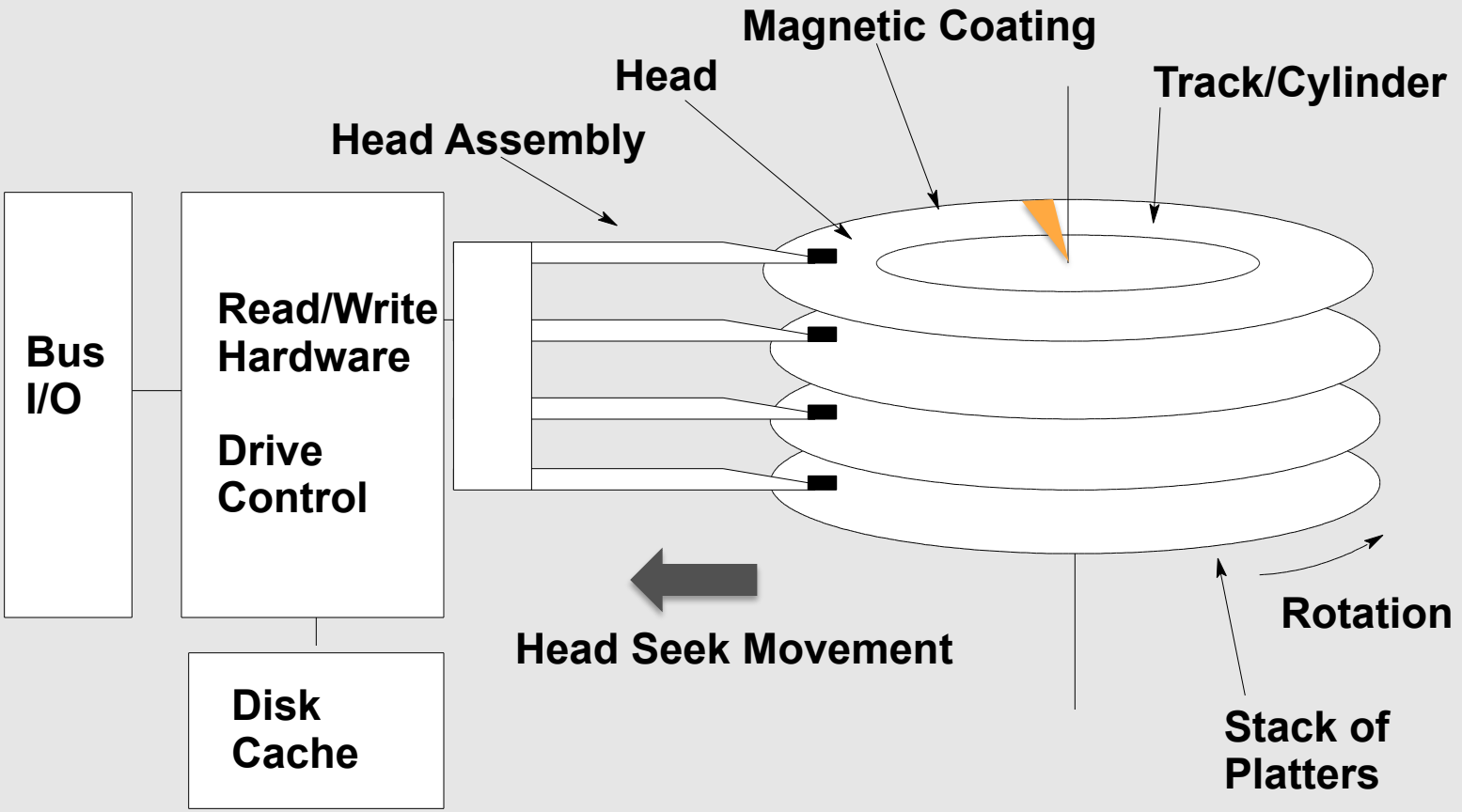
- Disk drives use a magnetic coating on a rotating disk platter to store information;
- Data is written serially in “cylinders”;
- Each cylinder comprises a very large number of consecutive data “blocks”, 512 or 4096 bytes each;
- An electromagnetic transducer, termed a “head” is used to write or read from the disk surface;
- A pivoting mechanical arm or translating mechanical finger is used to position the head over the part of the disk where data is to be read or written;

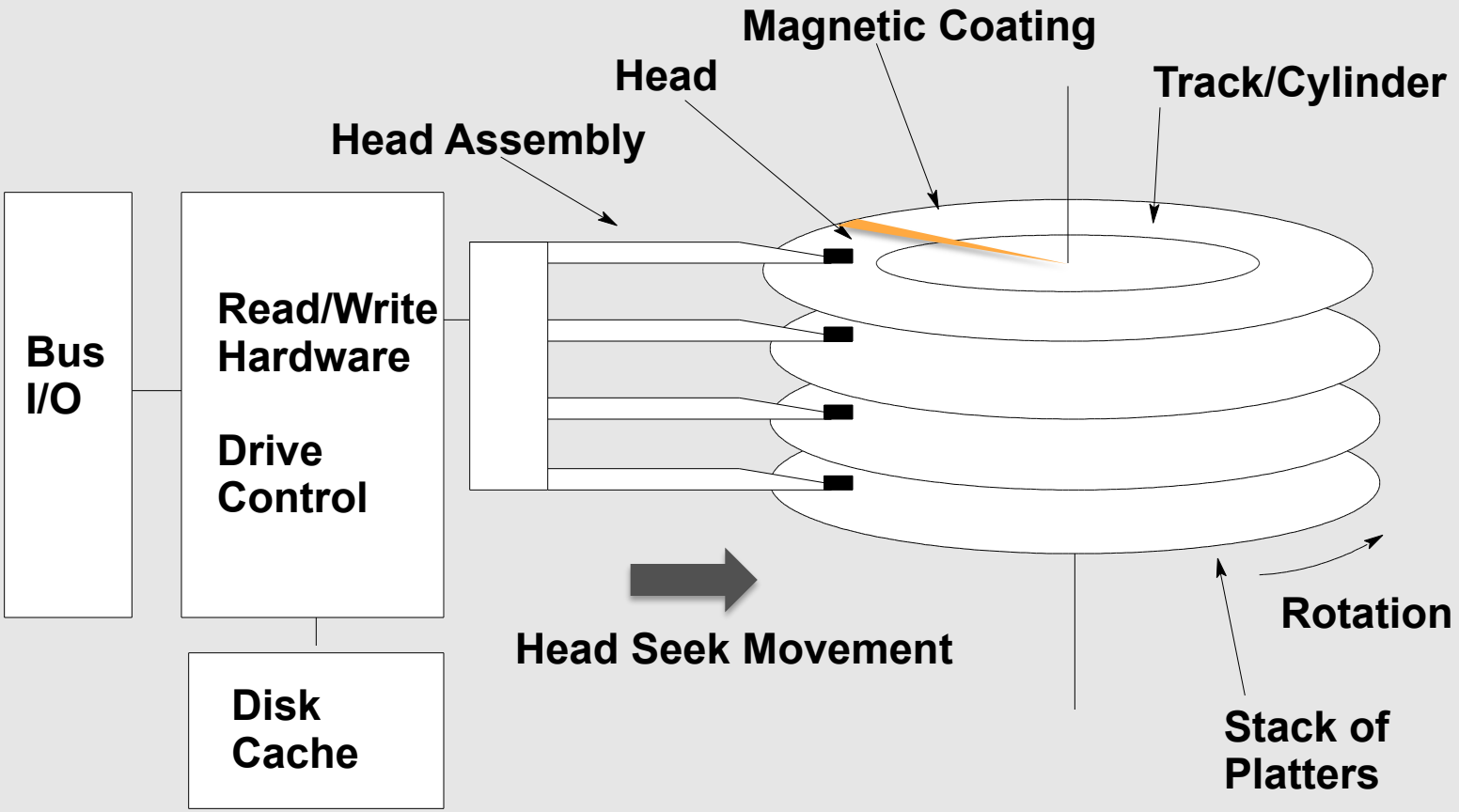

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters

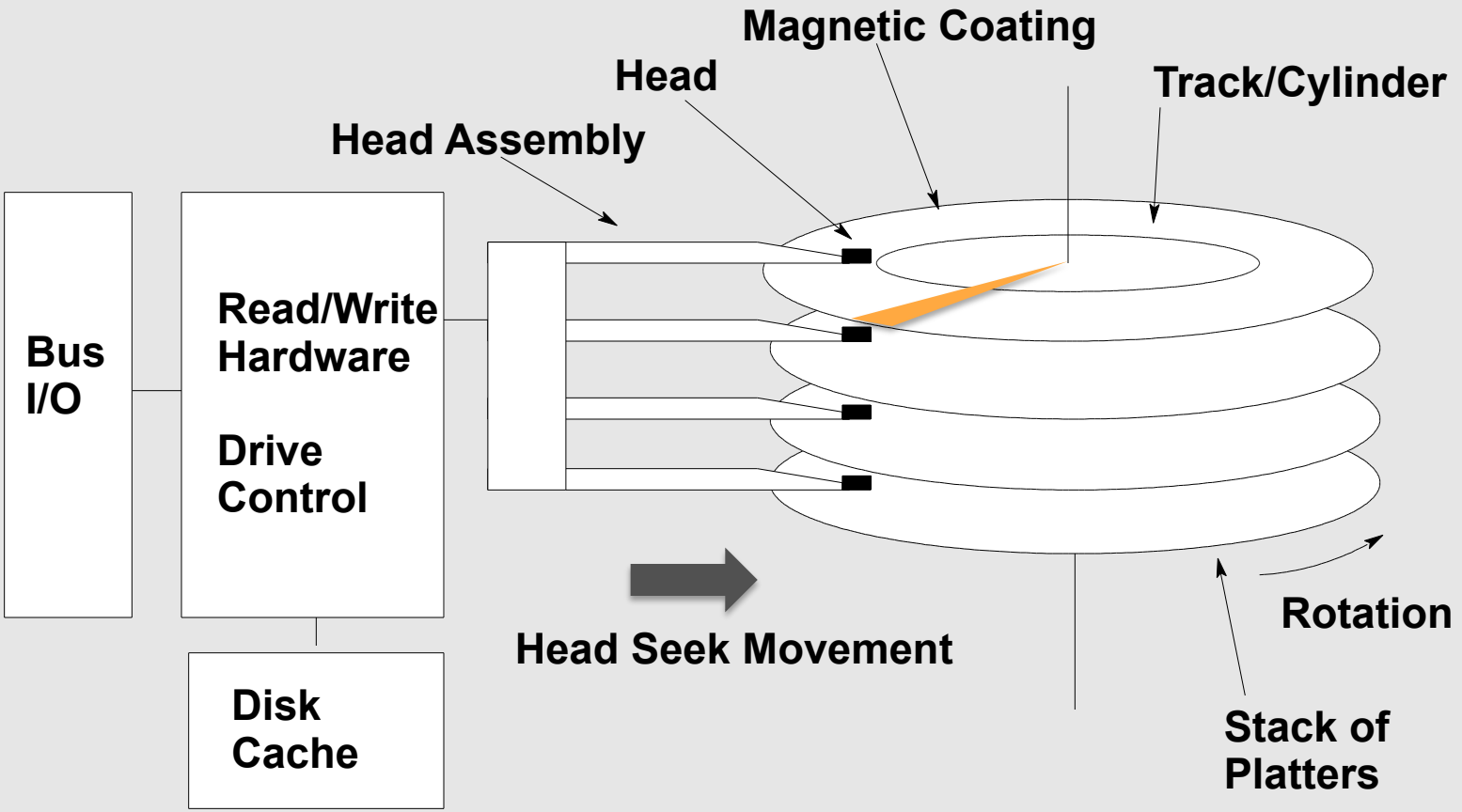

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters

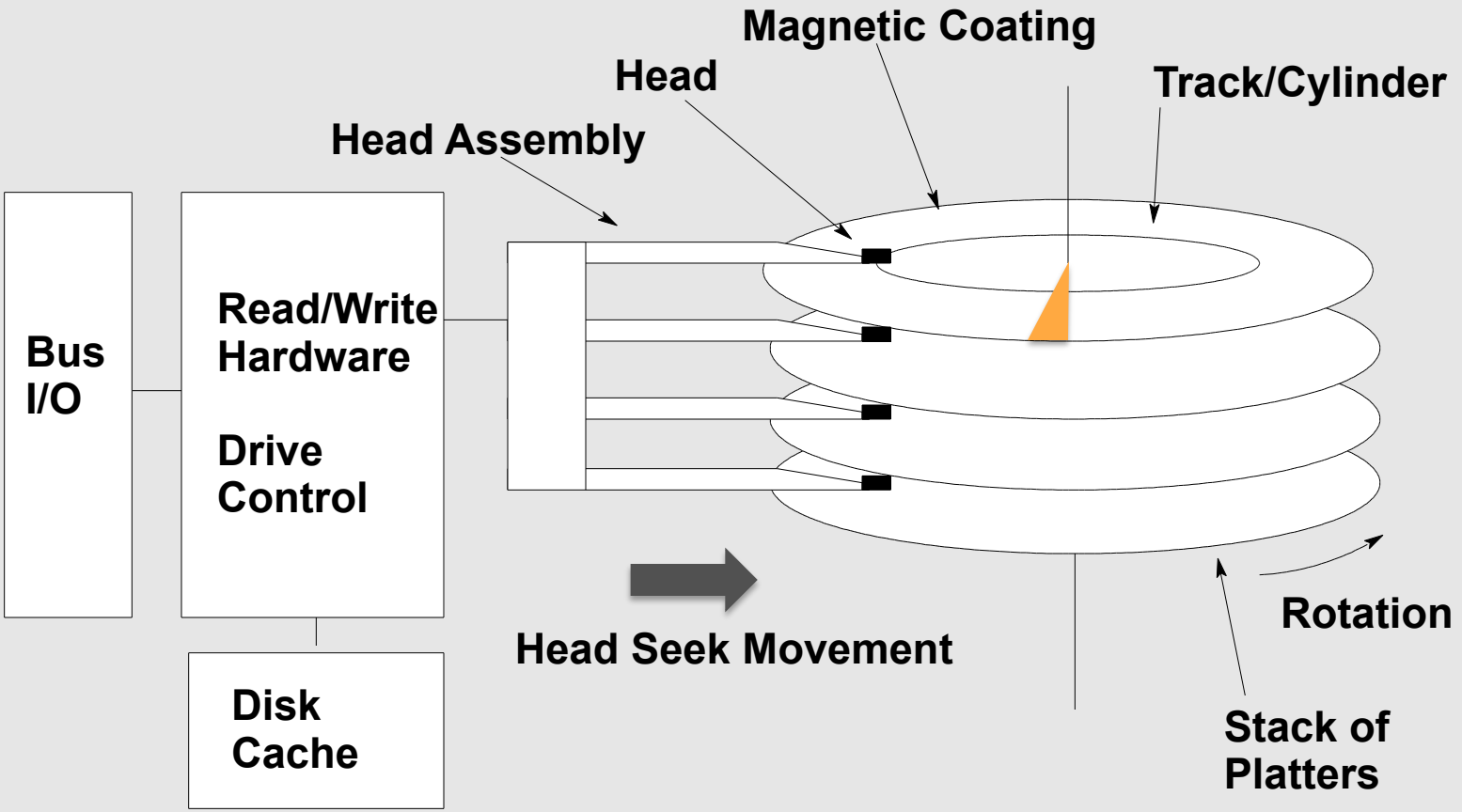

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters

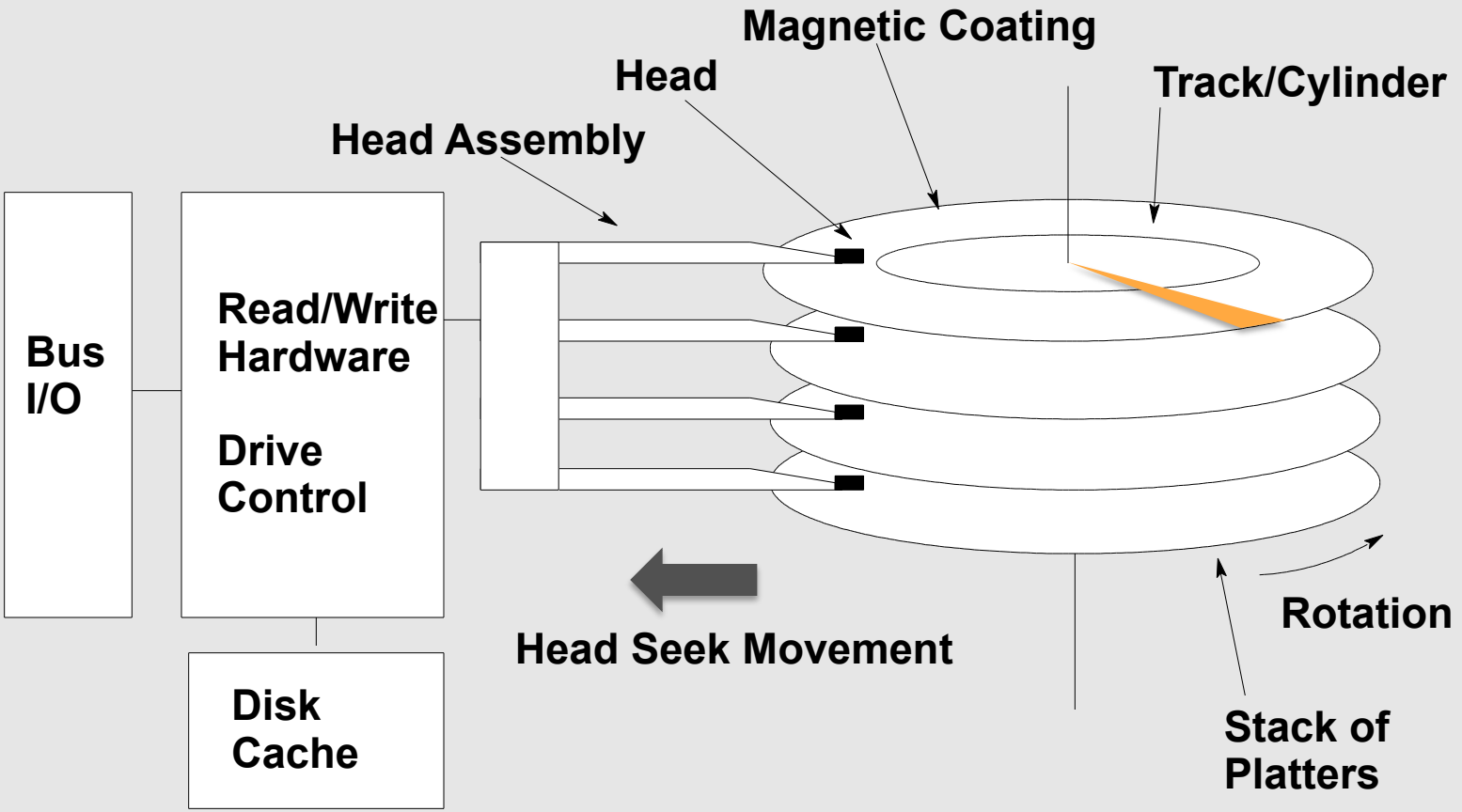

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters

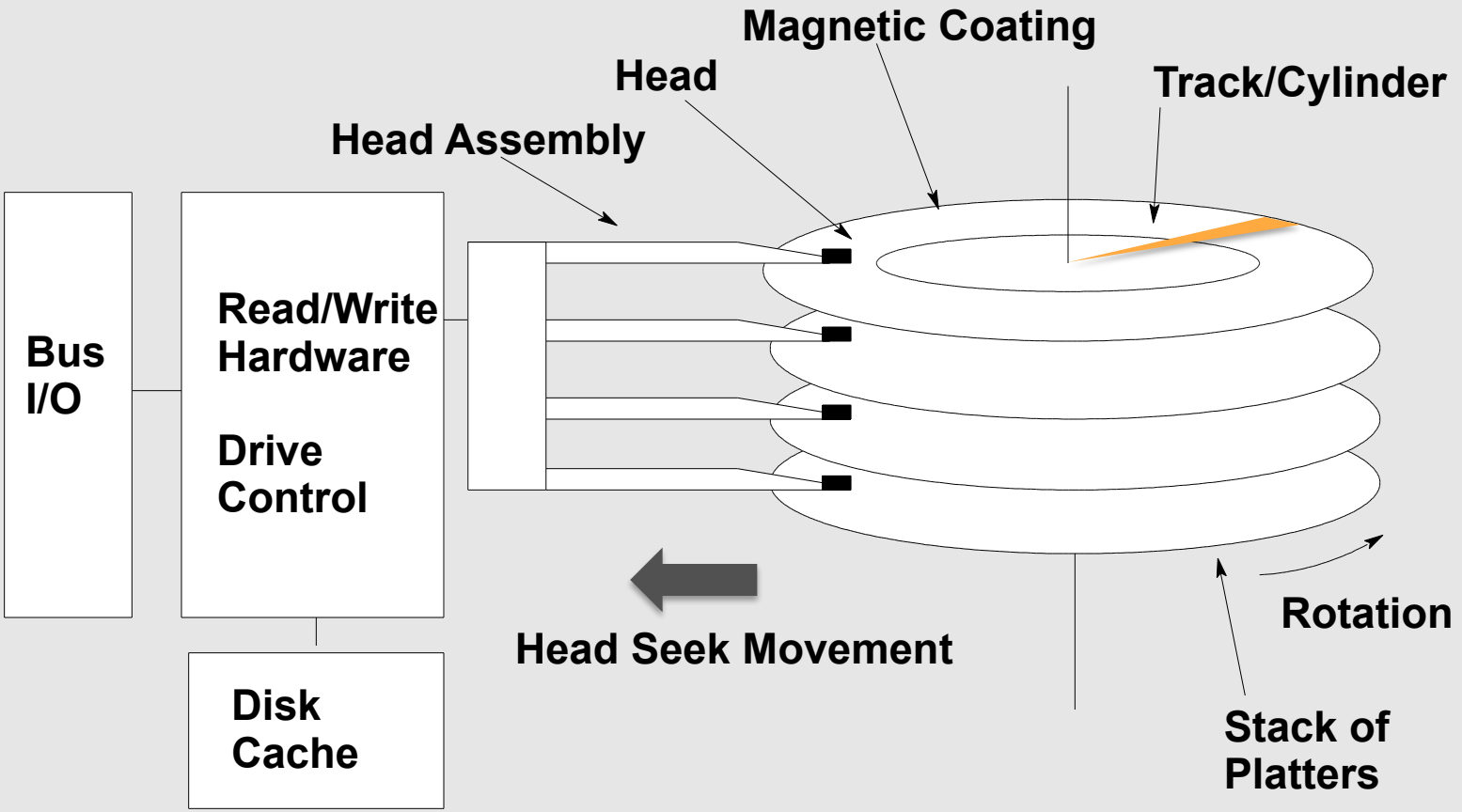

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Head Seek Movement
Rotation
Disk
Cache
Stack of
Platters

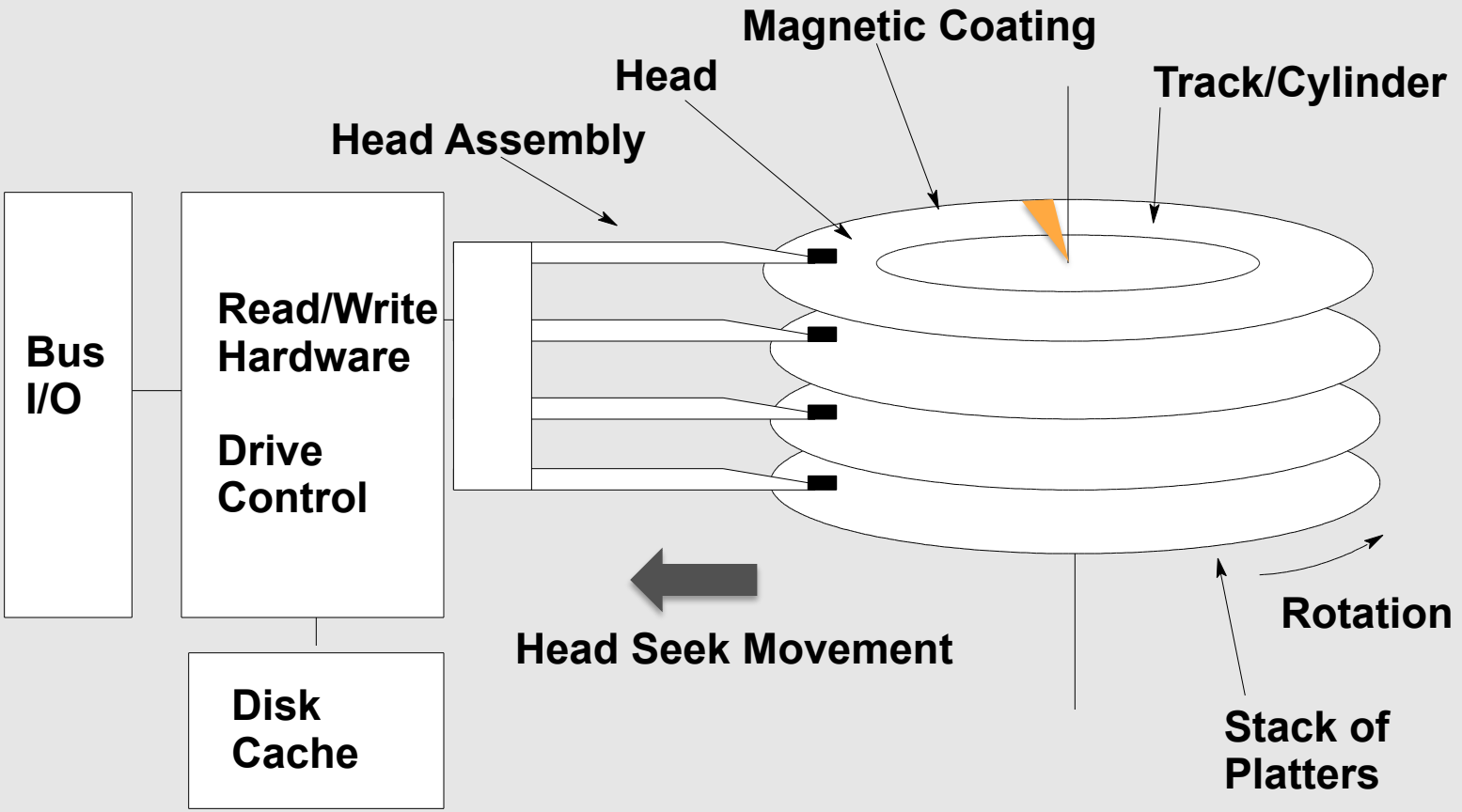

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters

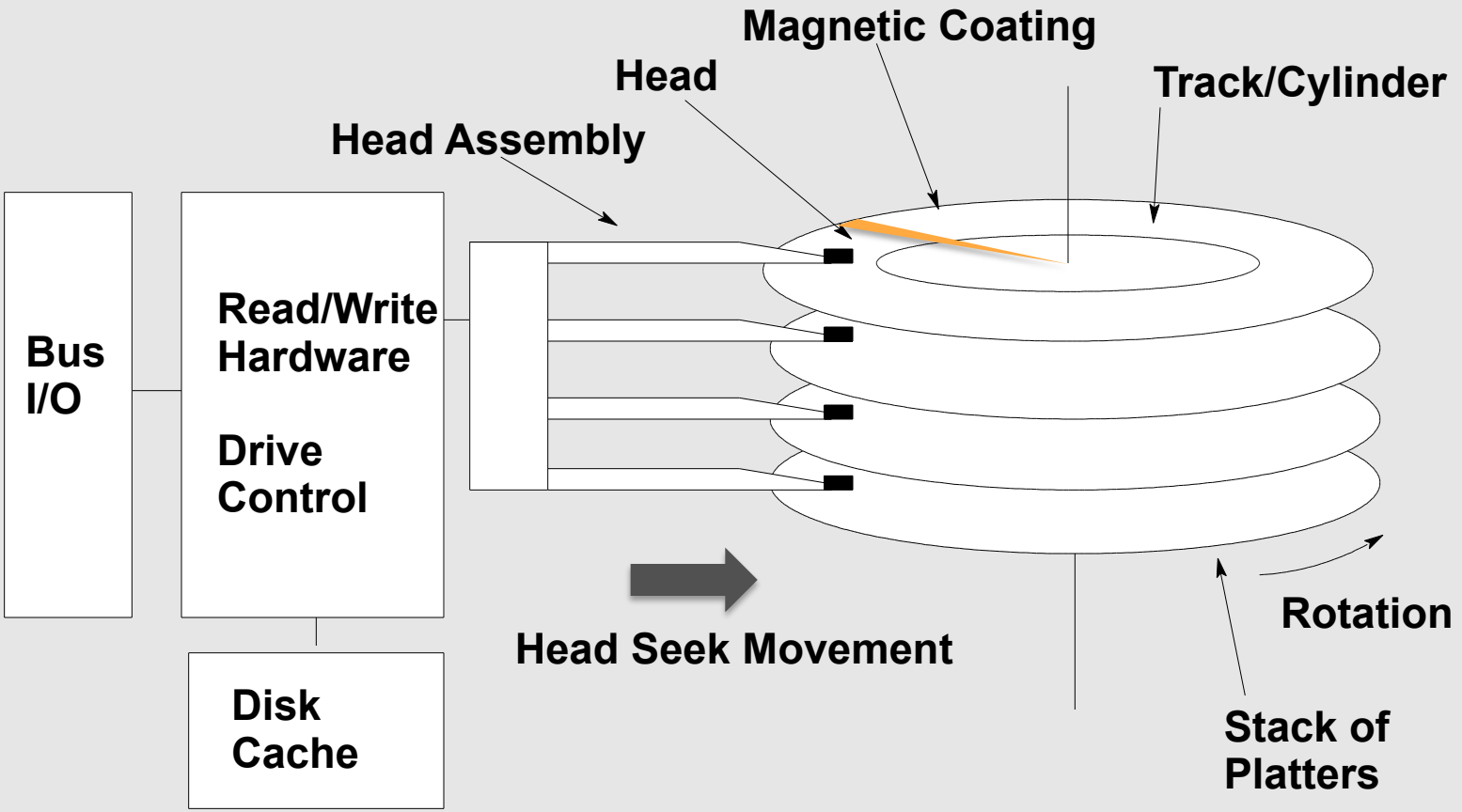

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters

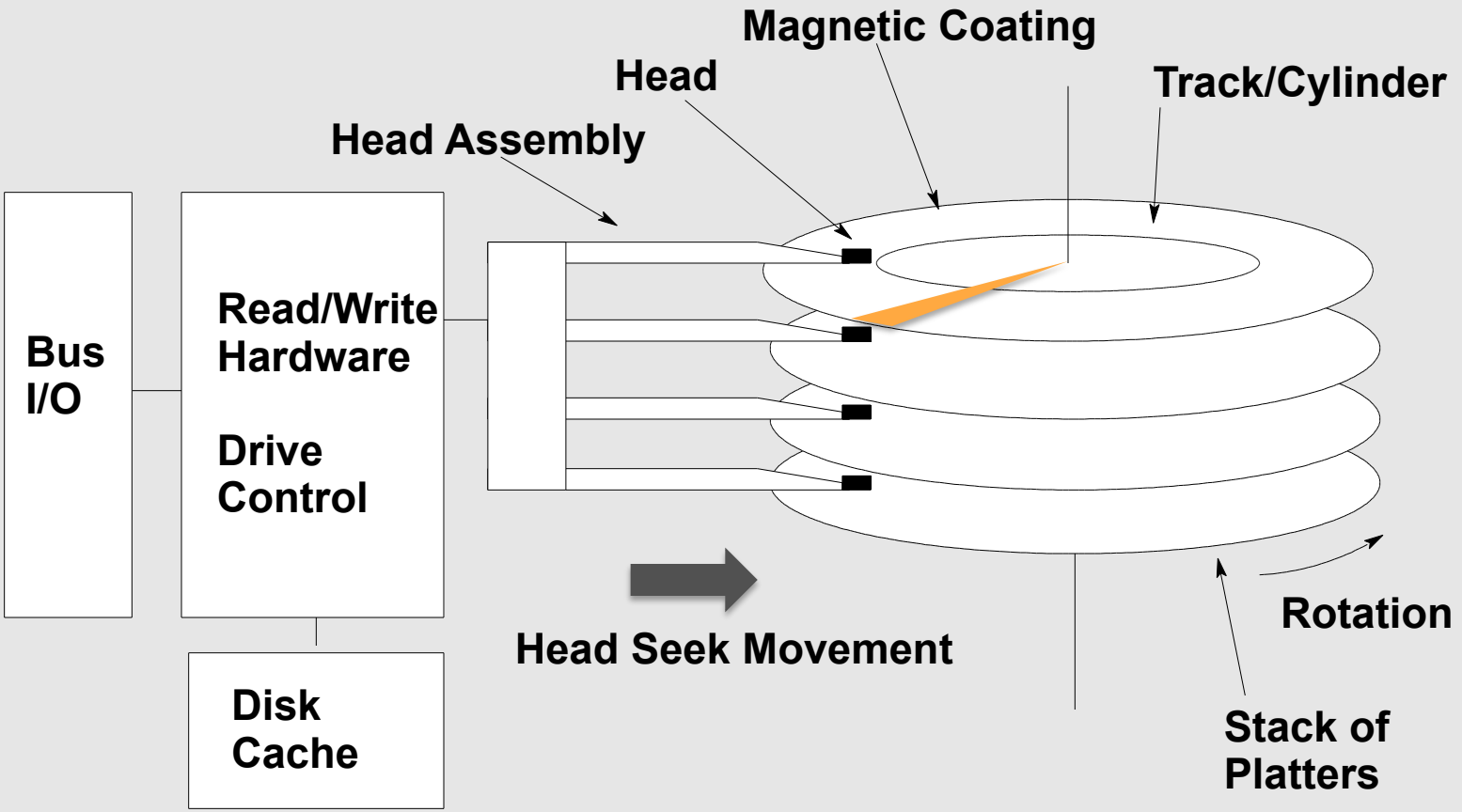

Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters


Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters


Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters


Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters


Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters


Magnetic Coating
Head
Track/Cylinder
Head Assembly
Bus
I/O
Read/Write
Hardware
Drive
Control
Disk
Cache
Head Seek Movement
Rotation
Stack of
Platters

Rotating Disk Performance

- **The stack of disk platters rotates at a constant RPM, in older disks 3,600 or 4,500 RPM, in newer types, 5,400, 7,200, 10,000 or 15,000 RPM.**
- **The time to access any item of data on the disk surfaces therefore depends on two parameters:**
  1. Rotational Latency - the time it takes for platter to rotate into position under the head.
  2. Seek Time - the time it takes to move the head over the cylinder holding the data.

<Access Time> = <Seek Time> + <Rotational Latency> [ms]

<… Time> - denotes mean or arithmetic average time

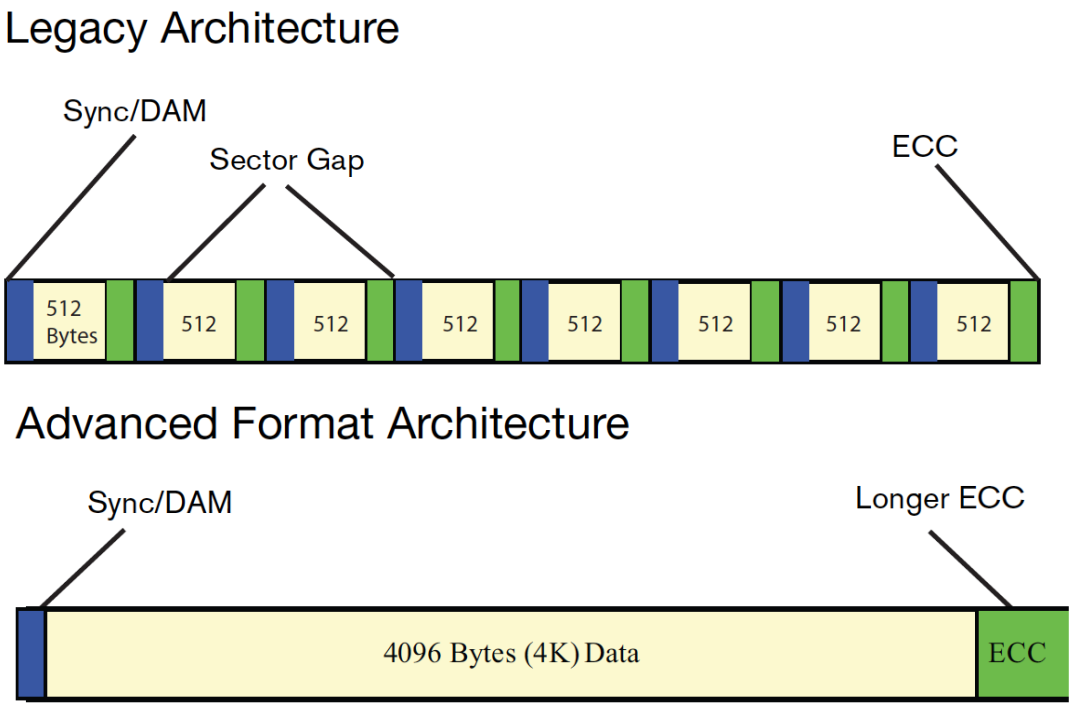
Rotating Disk Access Time Performance

- **The average rotational latency is 50% of the time it takes for a disk to rotate through 360 degrees:**
  - 3600 RPM - 8.33 milliseconds
  - 7200 RPM - 4.17 milliseconds
  - 15,000 RPM – 2.0 milliseconds
- **The average seek time between two adjacent cylinders (tracks) depends on the mechanical design of the head system:**
  - Typically 1.5-5 milliseconds
- **<Access Time> is usually 3.5 -> 13 [ms]**
- ***Extremely slow compared to main memory!***
- **Manufacturer specifications may be biased.**

Improving Disk Access Performance

- **Because mechanical disk hardware is very slow, we need to find ways of improving access performance:**
  - Data on disks is stored in blocks of 512 byte - 16 kbyte, to minimise the overheads of accessing it e.g. 4 kbyte "AF" disks.
- **Caching is a very useful technique for this purpose, and is used in two ways:**
  - A hardware "disk block cache" is built into the disk drive assembly, typically using 8 - 64 Megabyte of SRAM or DRAM.
  - "Hybrid" drives with 4 – 8 Gigabyte of DRAM cache within the drive enclosure (eg Seagate *Momentus XT* series – 2011).
  - The operating system uses portions of the main memory, ie "buffer cache" to cache disk blocks;
  - Multiple GB of on-disk and main memory cache are common now.

Disk Access Performance - Blocks



Advanced Format Technology White Paper, Western Digital, 2006

ECC – Error Correction Code Bit Pattern – cf network packets with headers and ECC

Rotating Disk Transfer Rate Performance

- Disk transfer rate performance is often called "bandwidth" in industry, and is a measure of the number of Megabytes per second transferred between the drive head and disk surface during a read or write operation;
- This parameter is the ultimate performance limit for reading or writing data to a rotating disk;
- It is *mostly* determined by the linear density of the data written on the disk surface, and the RPM of the disk;
- Increasing capacity (TB) ➔ increasing density ➔ increasing transfer rate or "bandwidth";
- *Transfer rate is often confused with access time by laymen – NB common error in popular literature;*
- Other impacts are cache and storage bus "bandwidth".

Mass Storage Busses

- Specialised busses for internal or external mass storage, e.g busses, magnetic tapes, optical drives;
- Designed for block transfers rather than byte transfers;
- Usually based on widely used industry standards, with specific signals, connectors and cables;
- Until a decade ago were mostly parallel busses, more recently serial bus designs – e.g. IDE/EIDE/ATA evolved to SATA (Serial ATA) standards;
- *Internal SATA 3.0 at 6 Gigabits/s ≈ 600 Megabytes/s; external variant is eSATA;*
- *External USB 3.0 at 5 Gigabits/s ≈ 450 Megabytes/s;*
- *External IEEE 1394b "Firewire" at 3.2 Gigabits/s s ≈ 256 Megabytes/s;*
- *SAS or "Serial Attached SCSI" at 6–12 Gigabits/s. 480-960 MB/s*

Mass Storage System Transfer Rate Performance

- *How "fast" is a mass storage subsystem?*
- We need to consider disk access time, disk transfer rates, mass storage bus transfer rates, interface chips in machines, and the operating system being used;
- Operating systems also matter – filesystem design impacts access times and transfer rates;
- Benchmarks performed with real hardware reflect the aggregate impacts of all components in the mass storage subsystem;
- *To get maximum performance, good choices must be made in what disk is used, what bus is used, and what operating system is run on a given item of hardware;*
- Example: an SSD with a SATA transfer rate of 600 Megabytes/s will only deliver 80 Megabytes/s on a Firewire 800 bus – the bus is the "bottleneck" in overall performance.

Benchmarking Transfer Rate Performance

- *Consider a 2012 built 2.5" 10,000 RPM 1TB SATA 3 Disk Drive:*
- Theoretical SATA 3 limit ≈ 480 MB/s; Measured = 400.8 MB/s;
- Measured Best Drive Transfer Rate = 209.1 MB/s;
- Measured Worst Drive Transfer Rate = 114.7 MB/s;
- Theoretical SATA 1 limit ≈ 160 MB/s;
- Measured Best DTR (iMac 5,1/Snow Leopard) = 121.8 MB/s;
- *Consider a 2012 built 3.5" 7,200 RPM 1TB SATA 3 Disk Drive:*
- Measured Best DTR (iMac 12,2/Snow Leopard) = 113.6 MB/s;
- *Consider SATA 2/3 Disk Drives in USB2 Enclosures (≈ 40 MB/s):*
- Measured Best DTR (iMac 5,1/Snow Leopard) = 36.4 MB/s;
- Measured Best DTR (iMac 12,2/Snow Leopard) = 31.7 MB/s;
- Measured Best DTR (MM 4,1/Mountain Lion) = 39.1 MB/s;
- Comparison: SATA 3 SSD Measured DTRs ≈ 200 – 500 MB/s

SATA Bus Variant Comparison / Kryder's Law

Kryder's Law => exponential density growth => exponential transfer rate growth

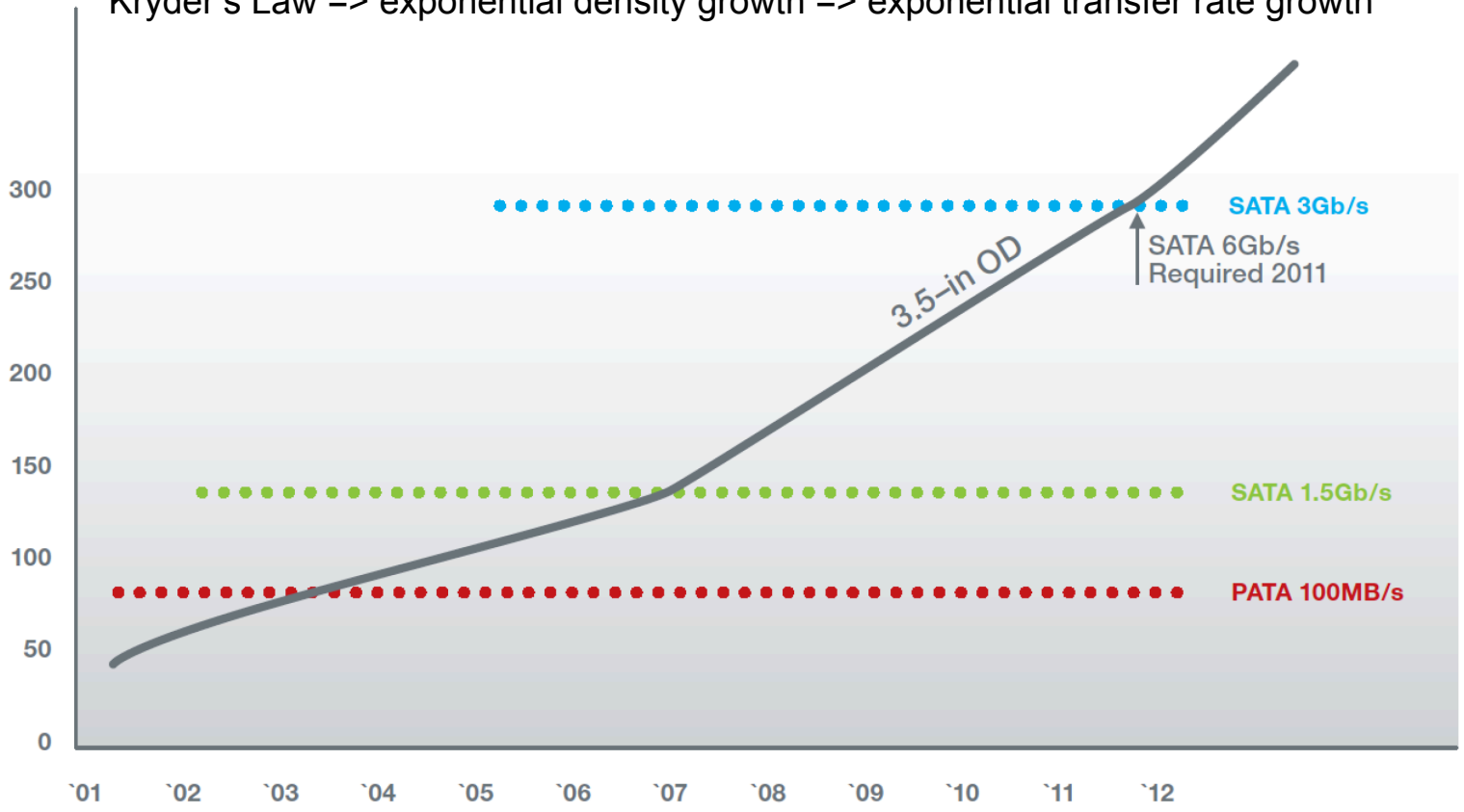
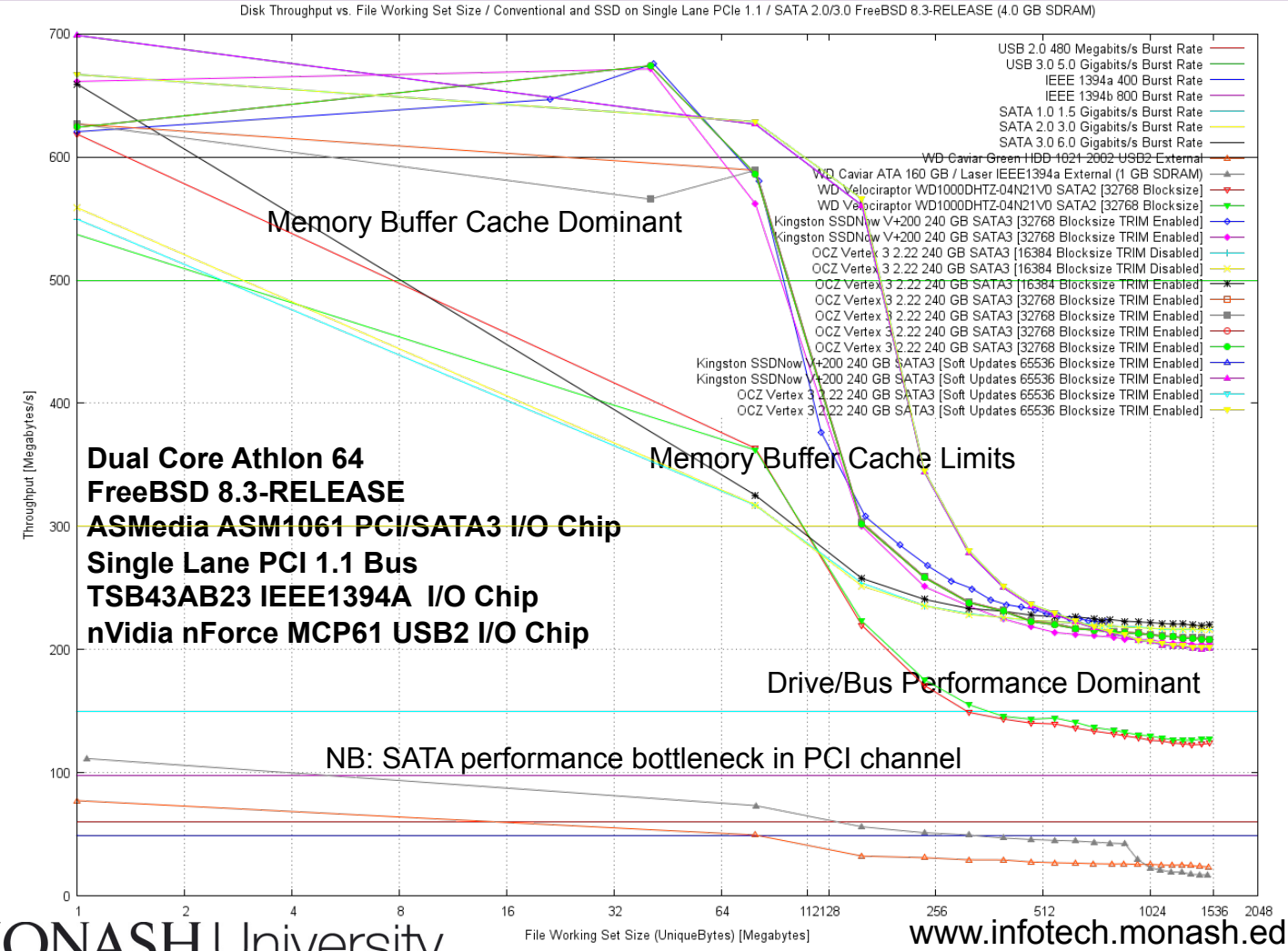


Figure 1. Sustained Data Transfer Rate Estimates

Source: Evolution to the SATA 6Gb/s Storage Interface; Seagate Technology Paper; 2010

Self Scaling I/O Benchmark (Chen 1992/Kopp 2013)

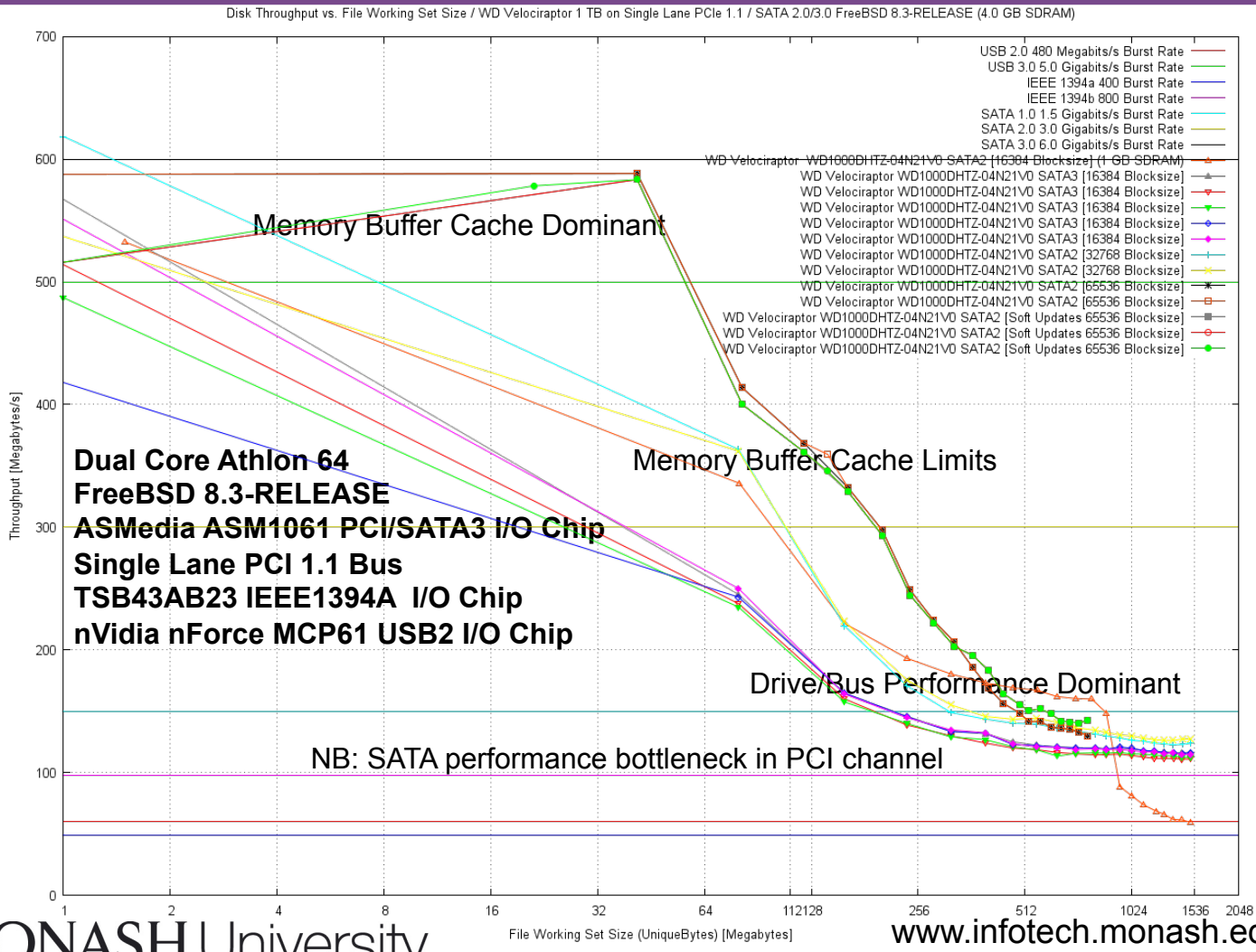


MONASH University
Information Technology

www.infotech.monash.edu

Not Examinable

Self Scaling I/O Benchmark (Chen 1992/Kopp 2013)



MONASH University
Information Technology

www.infotech.monash.edu

Not Examinable

Mass Storage Arrays

- A method widely employed in large servers, and some smaller high performance systems (including some gaming machines) is the use of mass storage arrays of rotating hard disks (or Solid State Disks);
- Storage arrays can be employed to:

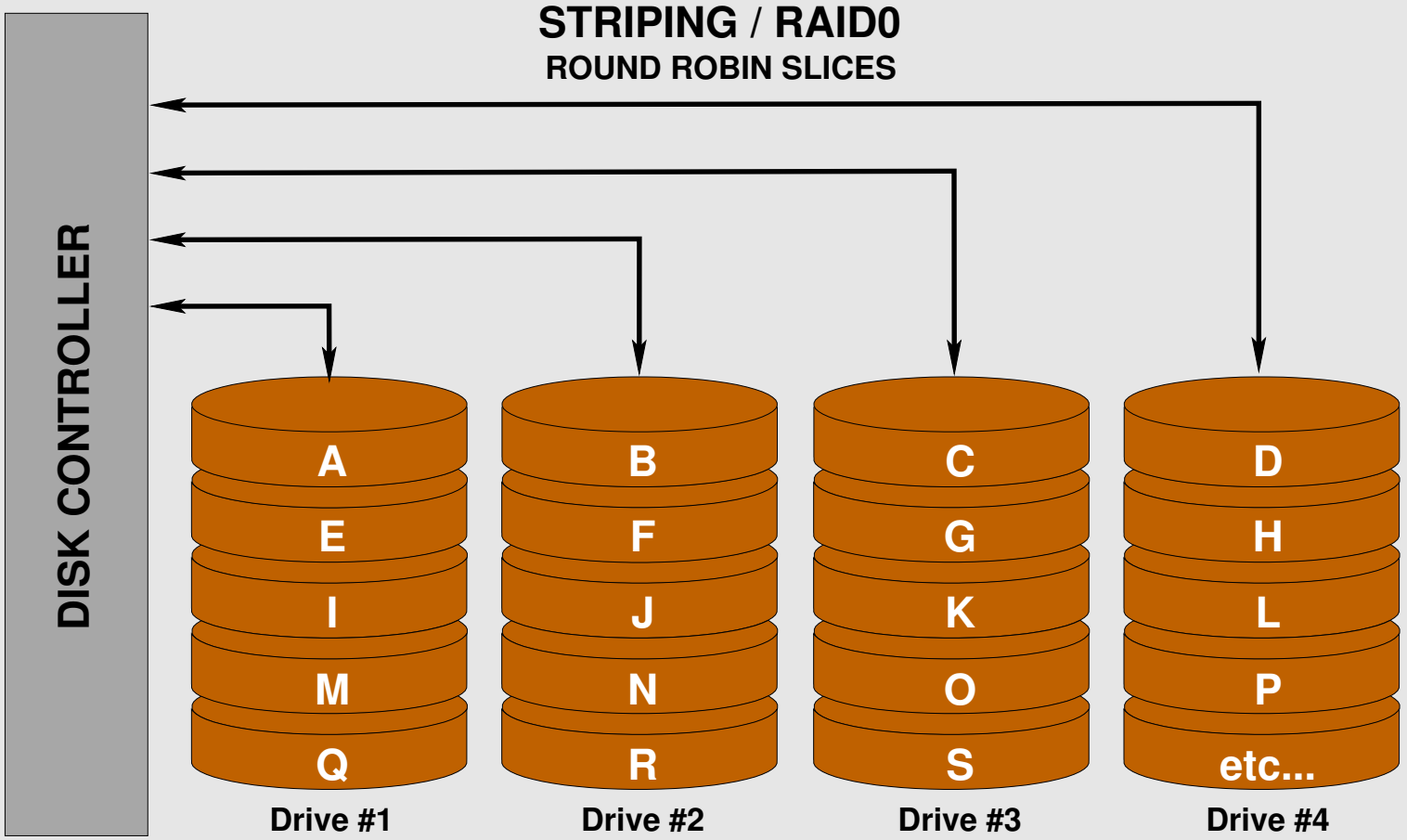
1. Improve mass storage transfer rate performance;
2. Improve mass storage subsystem availability (reliability);
3. Improve both transfer rate performance and availability;

- Early implementations of mass storage array techniques used dedicated disk controller hardware, but more recently software implementations have become very widely used;
- ***Mass storage array techniques can produce large improvements in transfer rates, but cannot improve access time performance of rotating disks!***

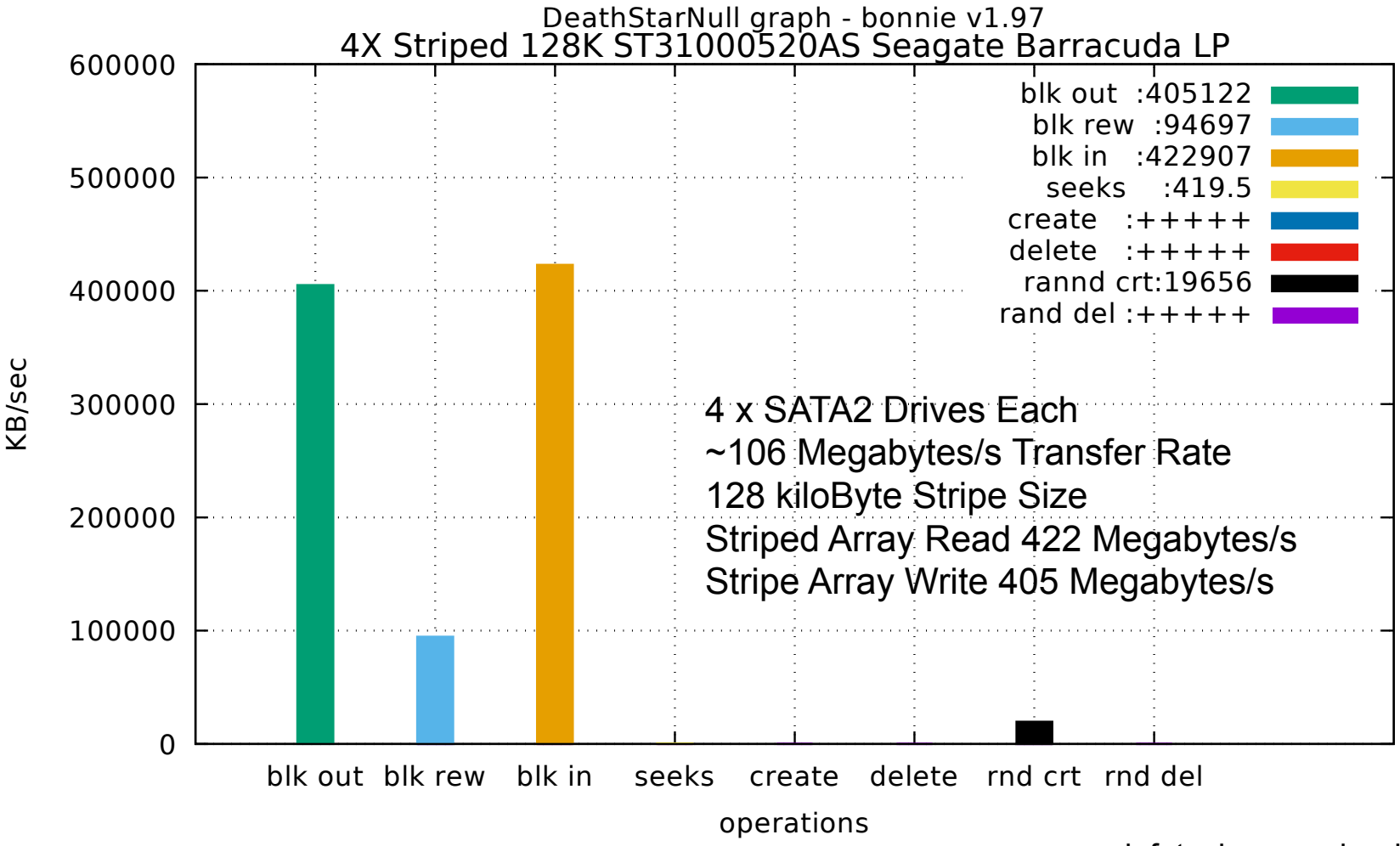
Mass Storage Array Striping (1)

- Striping is oldest disk array technique, first introduced in Cray supercomputing systems to increase transfer rate performance from a large storage array [Johnson, 1984];
- Striping remains widely used, as it is simple to implement, and performs well;
- It is most commonly used on large server systems, and HPC systems, but is sometimes used on high performance single user systems, including gaming machines;
- The central idea behind all striping schemes is to interleave data across multiple disks, in fixed sized "stripe units" or "slices", each usually being one or more disk blocks in size;
- When a file is being read from or written to the striped array, slice sized chunks of data are accessed on multiple disks in parallel, concurrently;

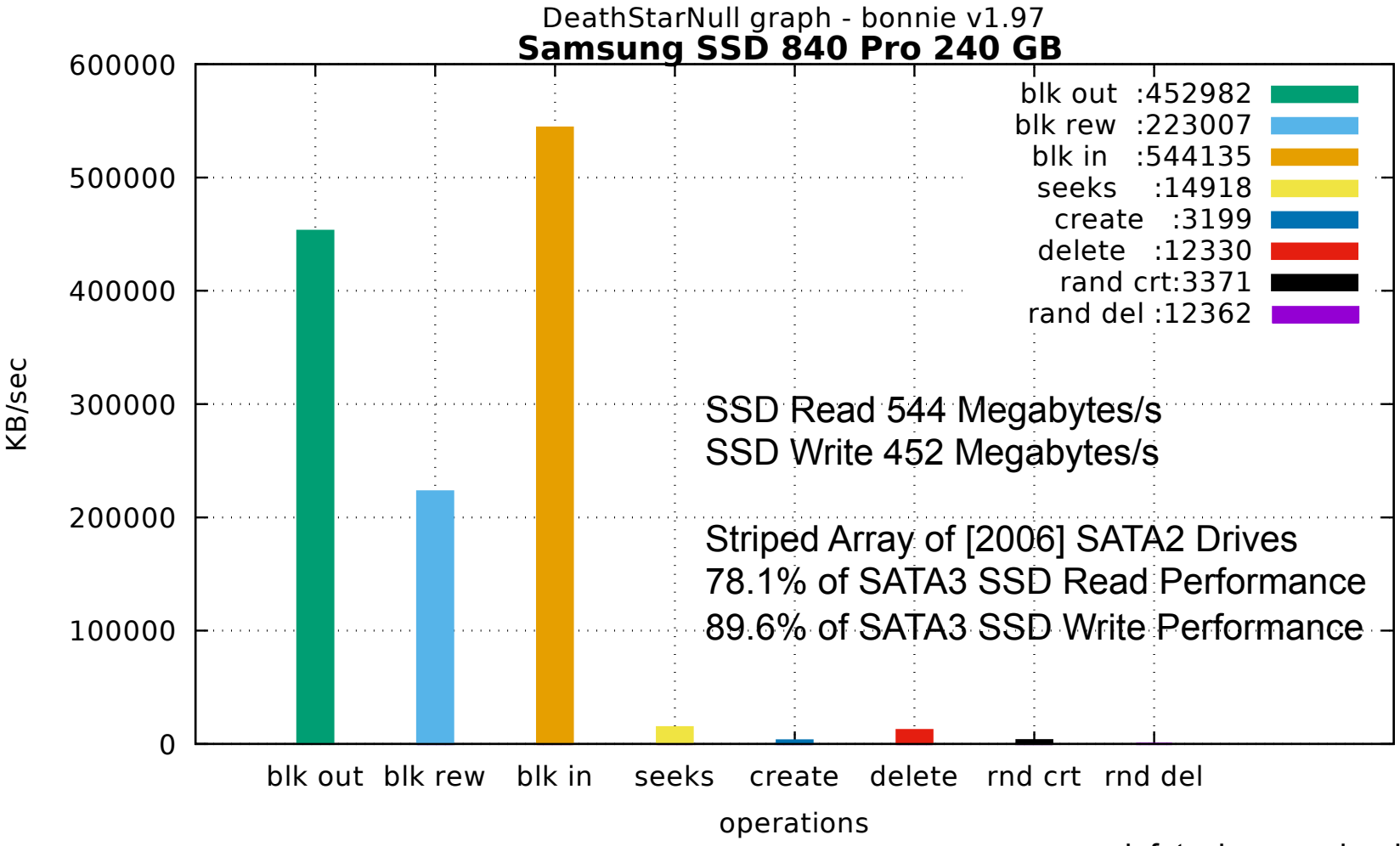
Mass Storage Array Striping (2)



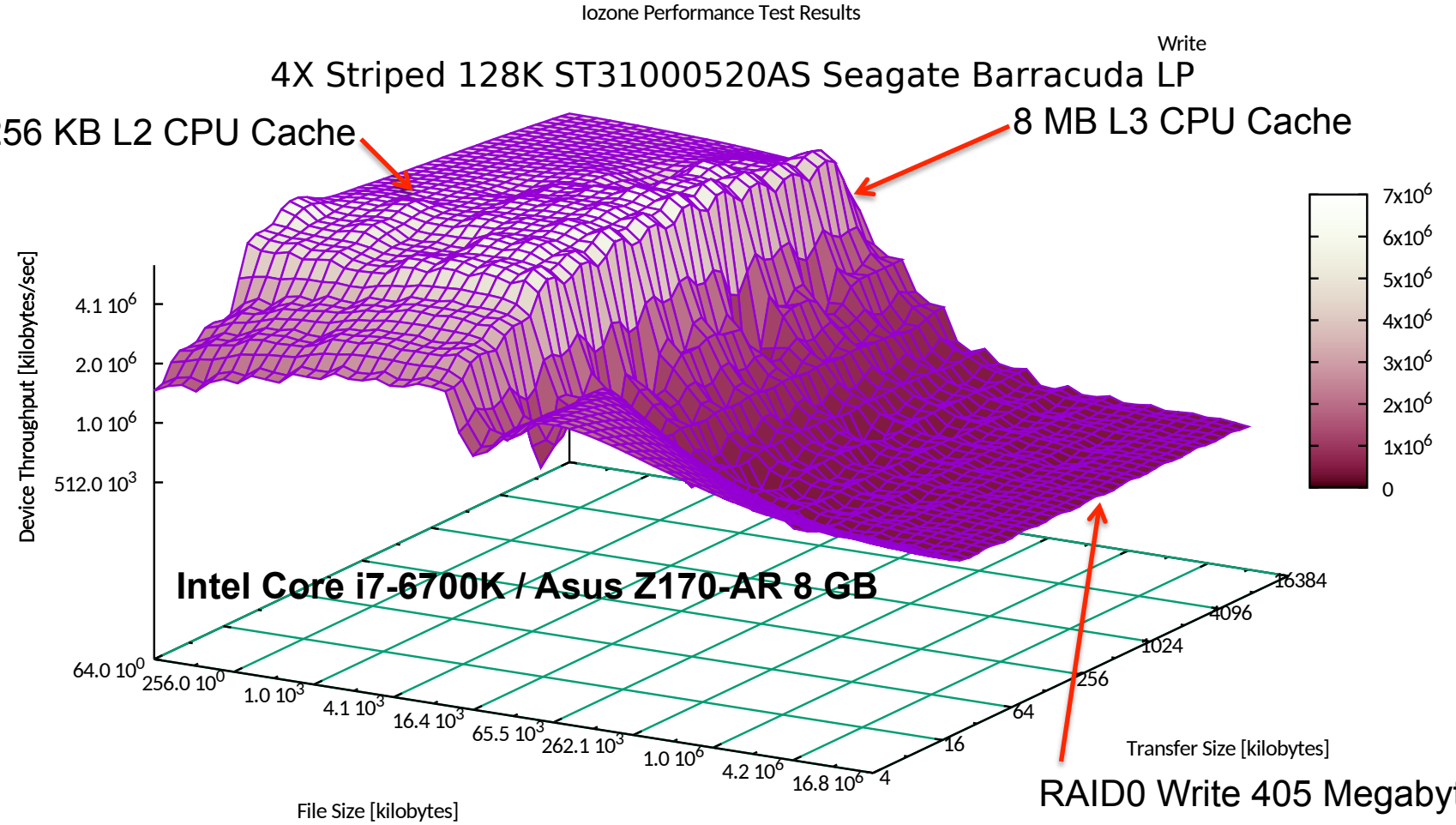
Mass Storage Array Striping Performance



Solid State Disk Performance - Comparison



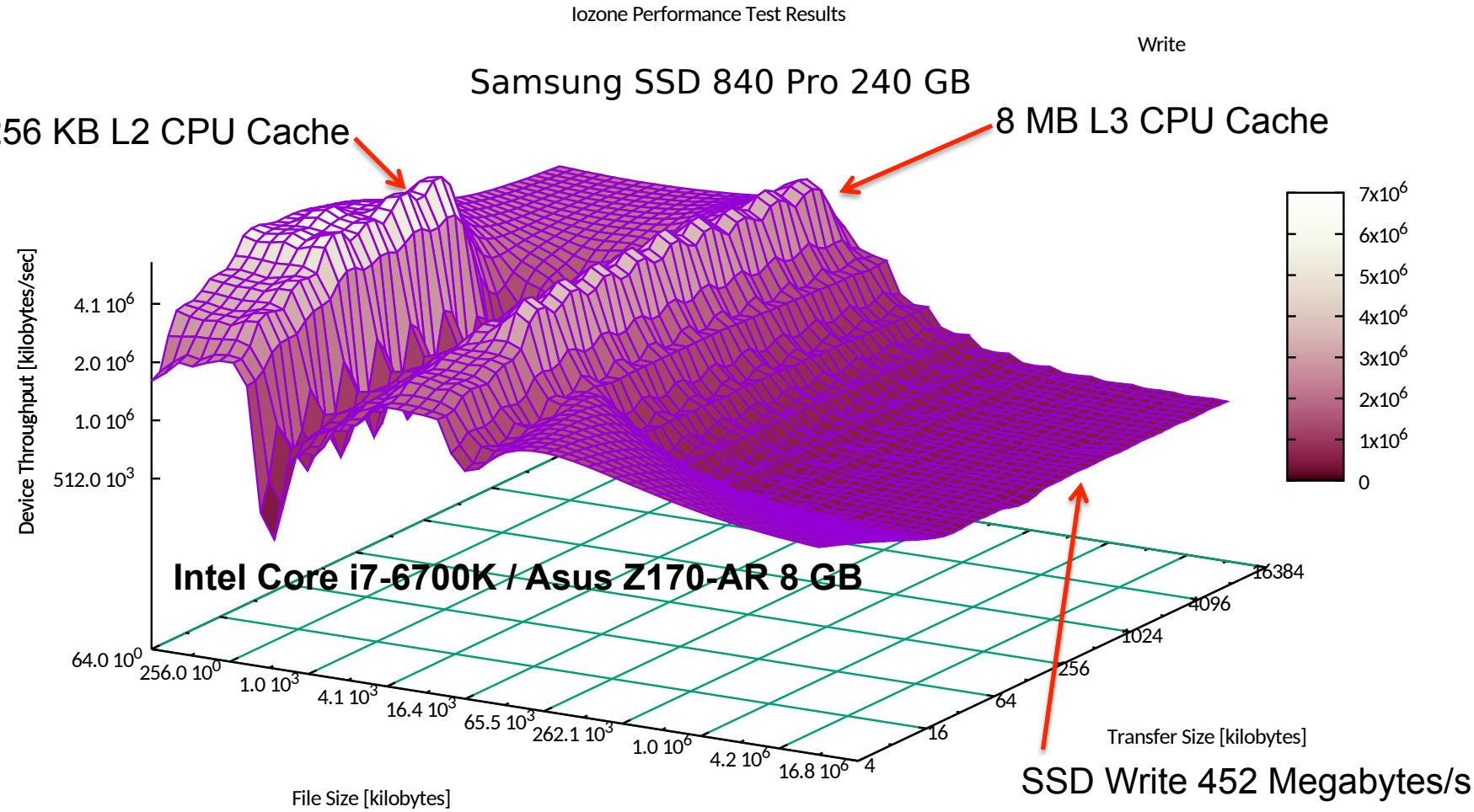
Mass Storage Array Striping Performance



www.infotech.monash.edu

MONASH University
Information Technology

Solid State Disk Performance - Comparison



MONASH University
Information Technology

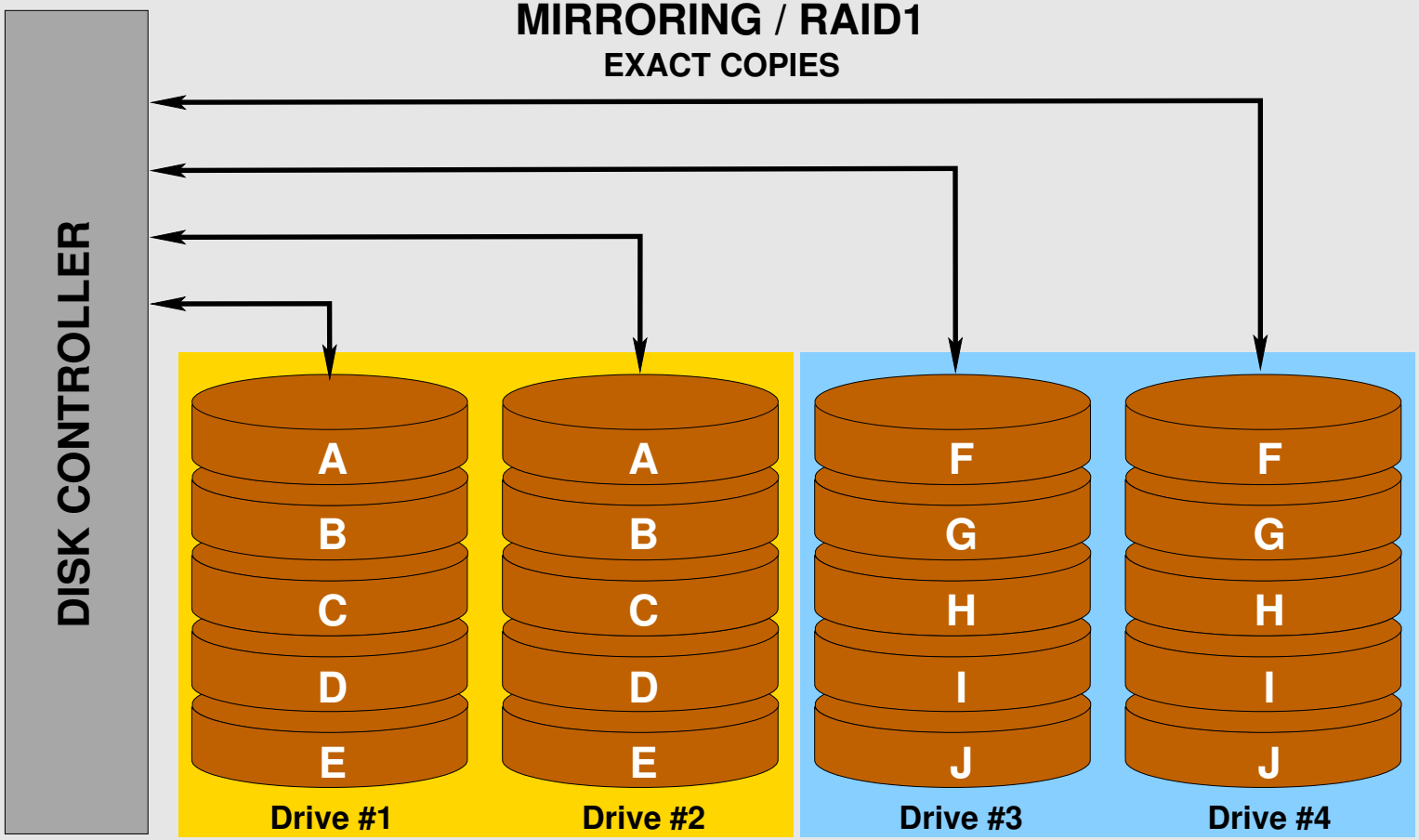
RAID – Redundant Array of Inexpensive Disks

- Devised by Patterson, Gibson and Katz @ UCB in 1988;
- Initially required specialised RAID controller hardware;
- Most desktop PC BIOS software now includes embedded RAID controller software, many RAID controller PCI/PCIe cards now available in the market, usually 4-8 SATA ports;
- Intention of RAID to improve performance and reliability by aggregating arrays of very cheap low performance disks;
- Patterson et al defined a number of "RAID Levels" to describe different ways of distributing the data across the disks to achieve different performance objectives;
- Some of the RAID schemes improve reliability, some performance, and some try to balance performance and reliability; Striping in RAID language is "RAID Level 0";
- RAID is used in NAS (Network-Attached Storage) devices.

RAID Level 1 - Mirroring

- Mirroring, like striping, is an older technique, and is intended to improve the reliability of data storage;
- In a mirrored array, disks are always arranged in identical pairs;
- Each of the two drives holds an exact copy of the data, block by block, held on the other drive;
- Whenever a block of data is written to one drive, the mirroring hardware or software writes the same block of data to the same block address on the other drive;
- If one of the two drives fails, it can be replaced by a blank drive, and the mirroring hardware or software will produce an image of the working drive on the replacement drive;
- The drawback of mirroring is that the cost of disk hardware is always doubled, for no gain in mass storage performance;

RAID Level 1 – Mirroring (2)



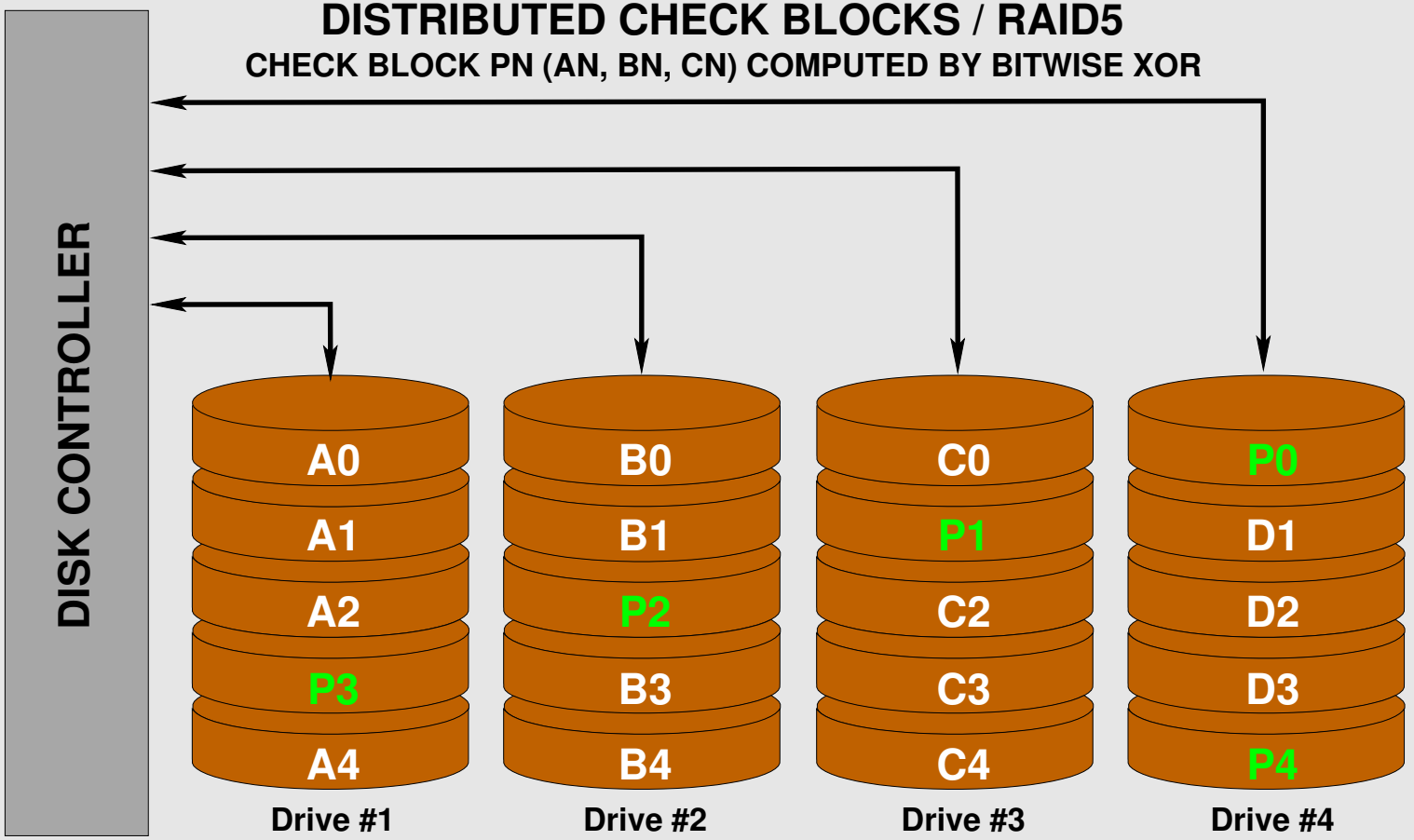
RAID Levels 2, 3 and 4

- In a RAID Level 2 system, data is bit-wise interleaved across an array of drives, where some are used for data storage, and a smaller number to store a Hamming code ECC word;
- RAID Level 2 was used for some HPC applications;
- In a RAID Level 3 system, data is byte-wise interleaved, and a single “check” disk is used, commonly arranged as 4+1;
- RAID Level 3 required synchronised disk spindles, and is now uncommon due to its requirement for expensive specialised disk and controller hardware;
- In a RAID Level 4 system, data is block-wise interleaved, and a single “check” disk is used to store per block ECC data;
- RAID Level 4 suffered from a write performance bottleneck in the “check” disk and was rapidly displaced by Level 5 in industry;

RAID Level 5 – Distributed Check Blocks

- In a RAID Level 5 system, data is block-wise interleaved across an array of drives, with data and "check" blocks spread across all drives in the array;
- RAID Level 5 is the most widely employed scheme at this time, as it can use "generic" drives, is very efficient in its use of disk capacity, and is robust;
- RAID Level 5 can recover from the loss of one drive in the array, but cannot handle the loss of more than one drive;
- RAID Level 5 performance is inferior to RAID Level 0 and 3, but superior to Levels 1, 2 and 4;
- RAID Level 5 capability is commonly embedded in the BIOS of commodity motherboards, firmware of most PCI/PCIe multiport SATA/SAS expansion cards, and in specialised NAS hardware;

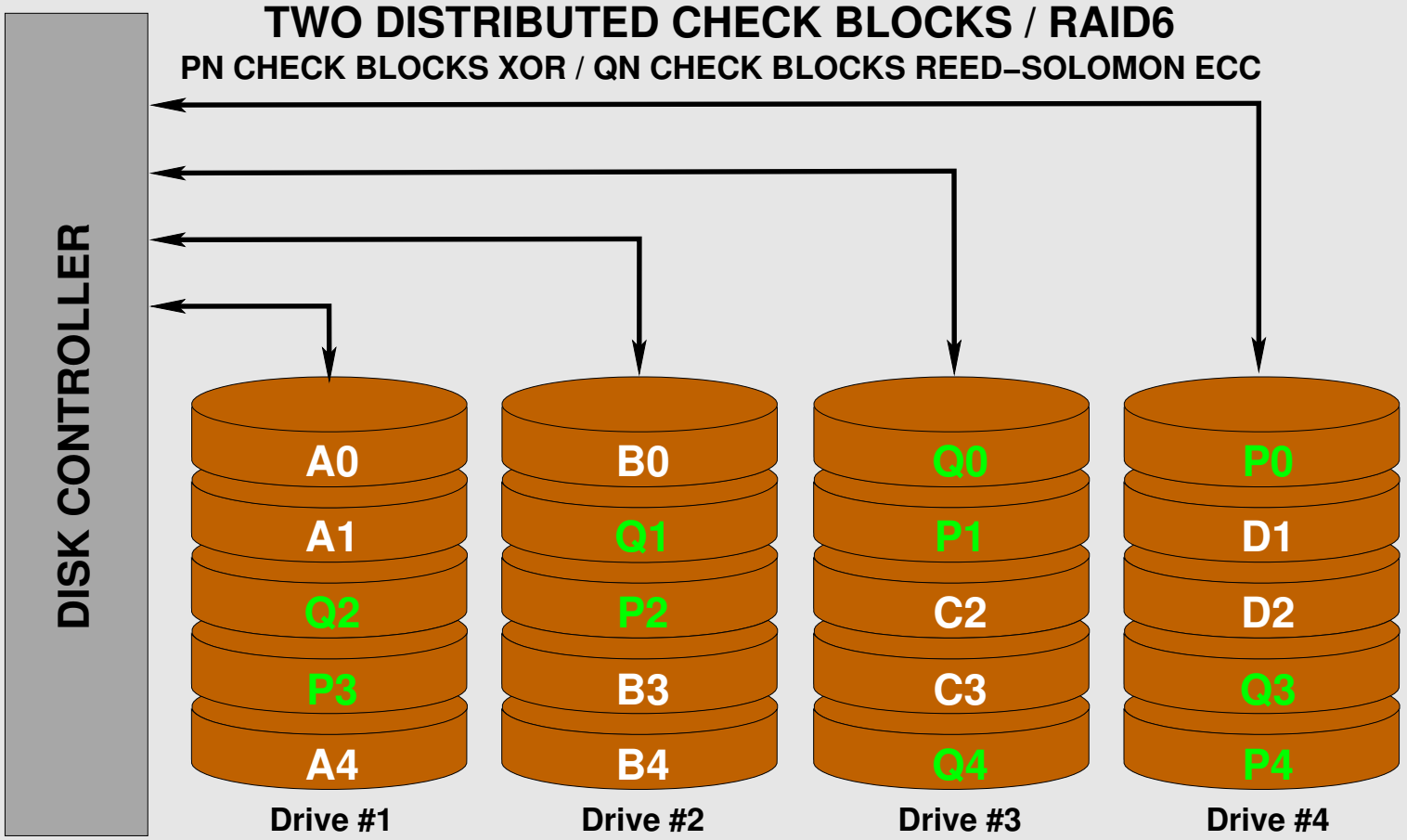
RAID Level 5 – Distributed Check Blocks (2)



RAID Level 6

- In a RAID Level 6 system, data is block-wise interleaved across an array of drives, with data and pairs of “check” blocks spread across all drives in the array;
- One check block is computed by bitwise XOR, the other using a Reed-Solomon Error Correction Code to identify which two drives failed;
- RAID Level 6 is used to provide better reliability than Level 5, but this is at the expense of both performance and efficiency as twice as many “check” blocks are required compared to Level 5, and twice as many write operations for check data;
- RAID Level 6 can recover from the loss of two drives in the array, but cannot handle the loss of more than two drives;
- RAID Level 6 can be frequently found in specialised NAS hardware;

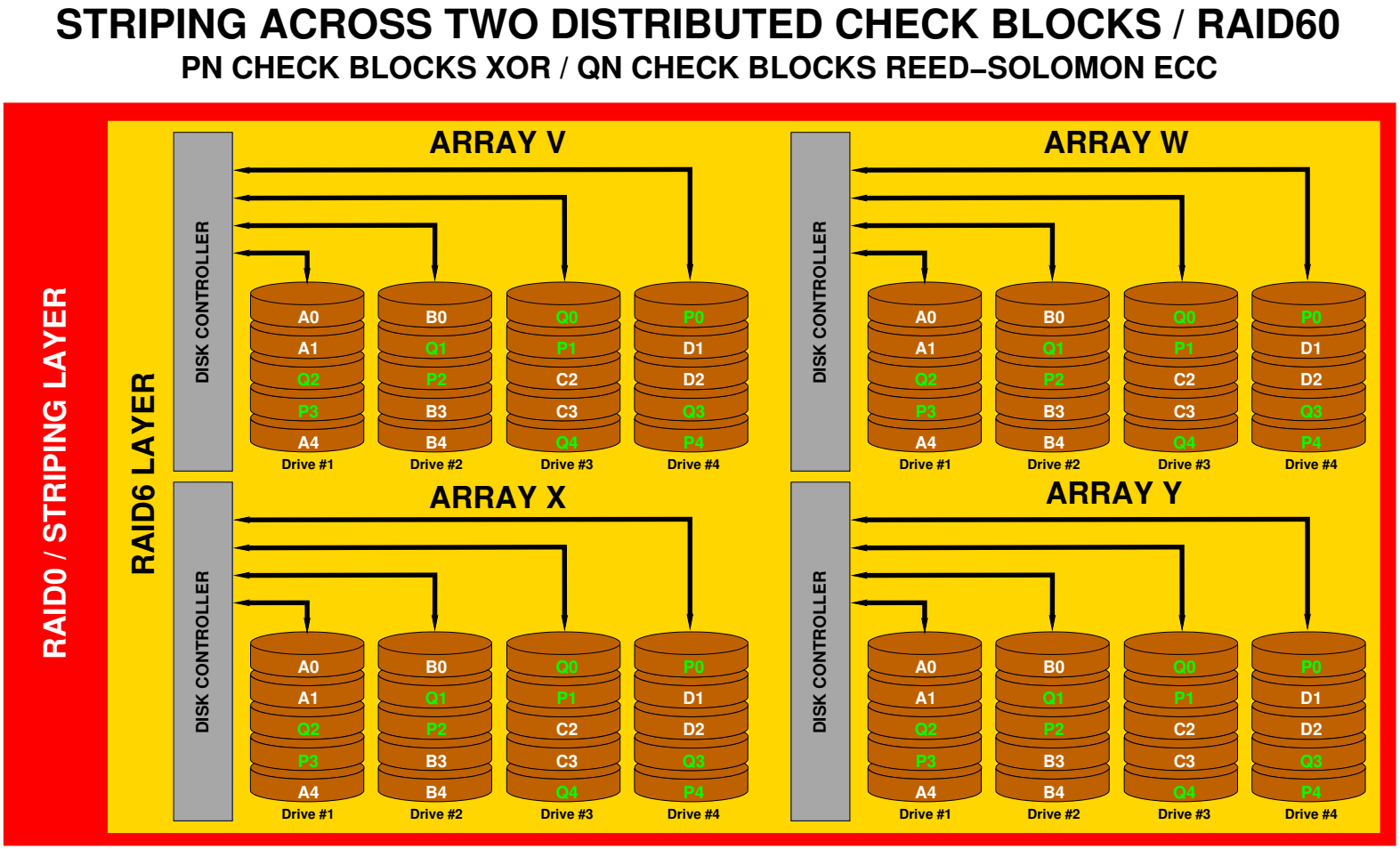
RAID Level 6 – Two Distributed Check Blocks (2)



Nested RAID Levels 10, 01(0_1), 50 and 60

- The intent of “Nested” RAID is to improve performance;
- In a RAID Level 10 (“one – zero”) and 01 (“zero – one”) systems, mirroring (Level 1) is combined with striping (Level 0) to achieve both high performance and reliability;
- A RAID Level 10 system will stripe (RAID0) data across a mirror (RAID1) of two arrays of drives;
- A RAID Level 01 (0_1) system will mirror (RAID1) data across two arrays of striped (RAID0) drives;
- In a RAID Level 50 system, data is striped (Level 0) across multiple RAID Level 5 arrays to achieve high performance and reliability, but at lower cost than Level 10;
- In a RAID Level 60 system, data is striped (Level 0) across multiple RAID Level 6 arrays to achieve high performance and better reliability than Level 50, but at lower cost than Level 10;

RAID Level 60 – Striping Multiple RAID6 Arrays



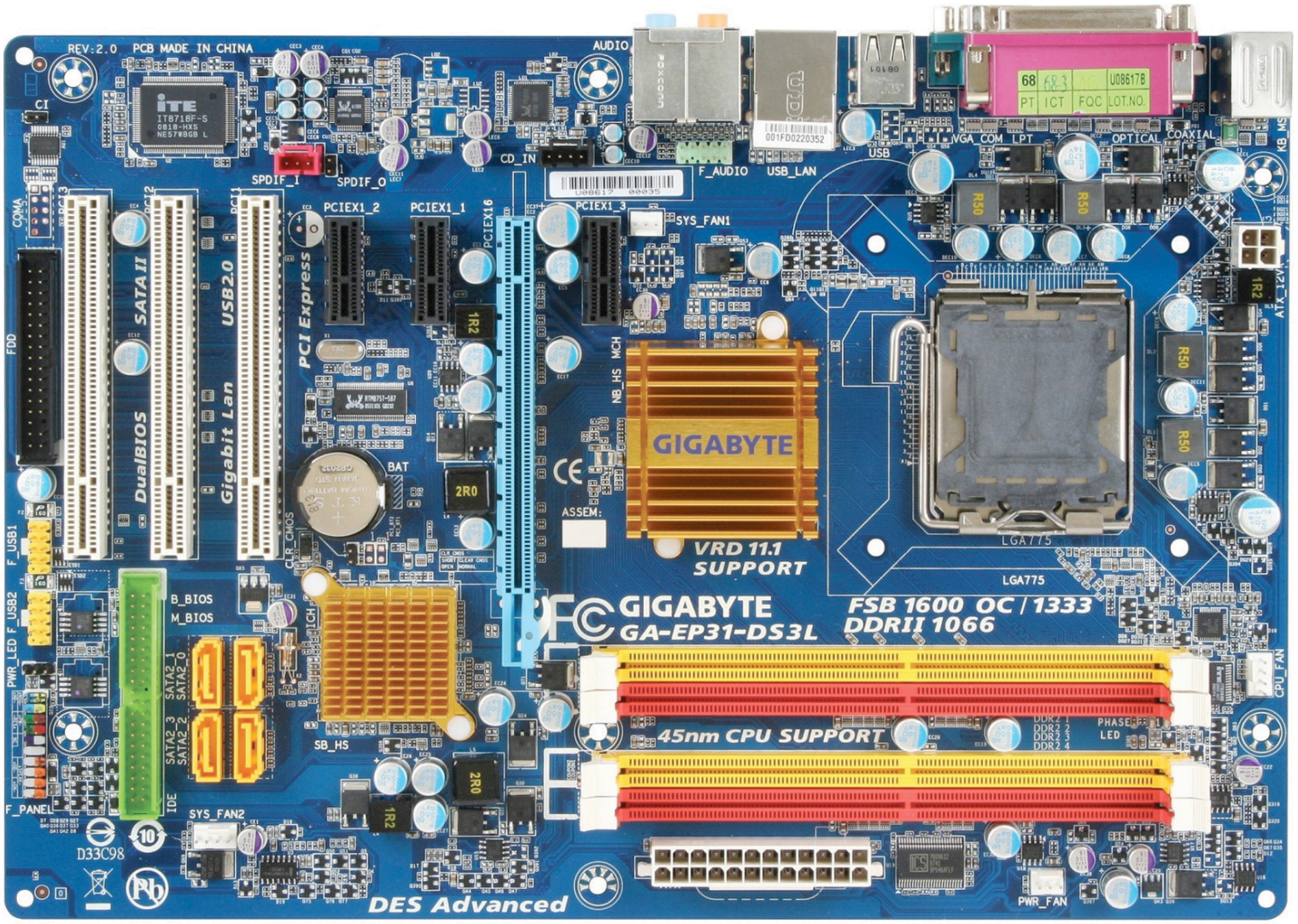
Summary / What's Next

- I/O schemes
- Interrupts and interrupt handling
- DMA
- Storage Hierarchies
- Disk Performance
- RAID Storage Arrays

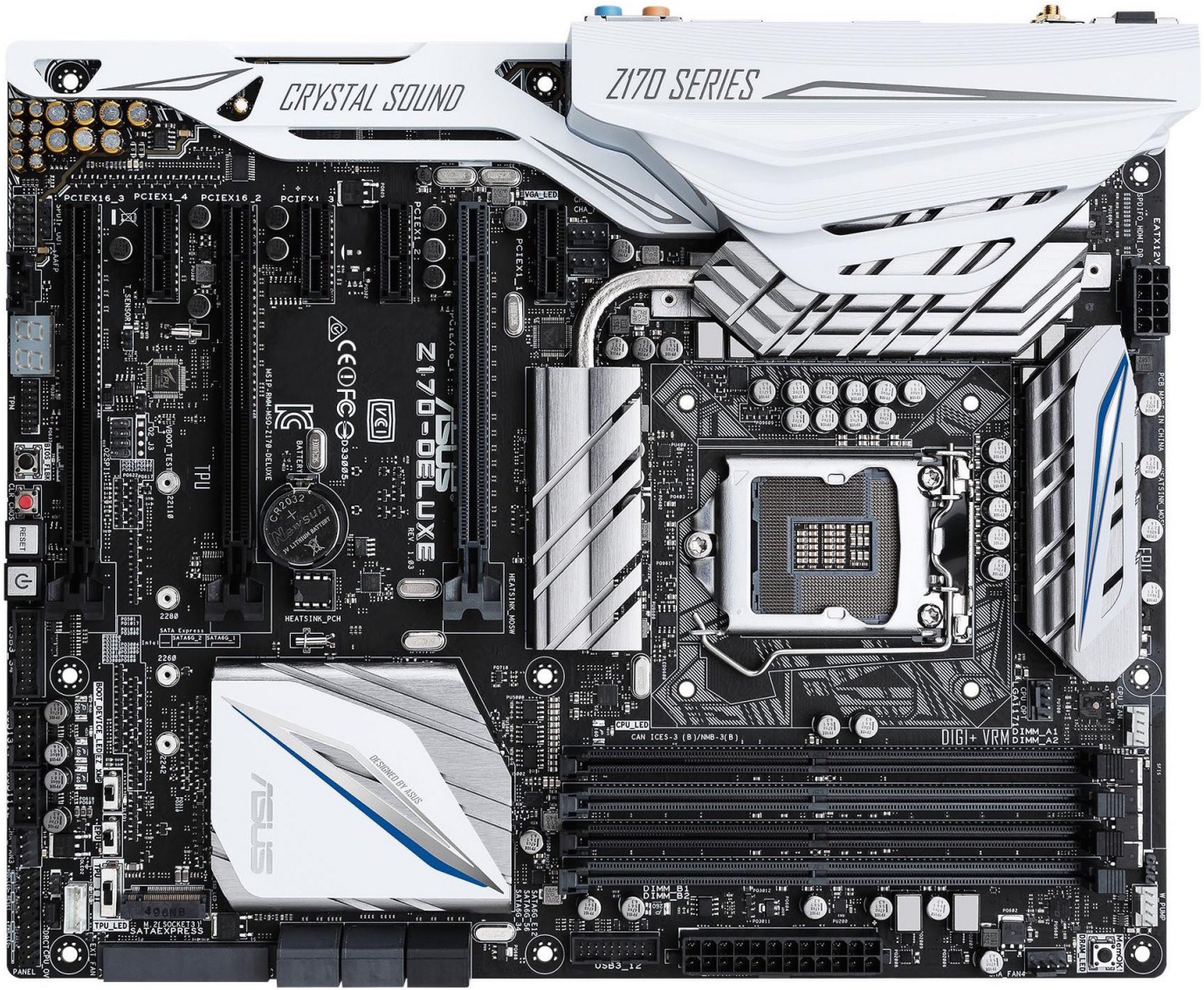
- Next?
- Caches

Examples of motherboard, main memory and disk hardware

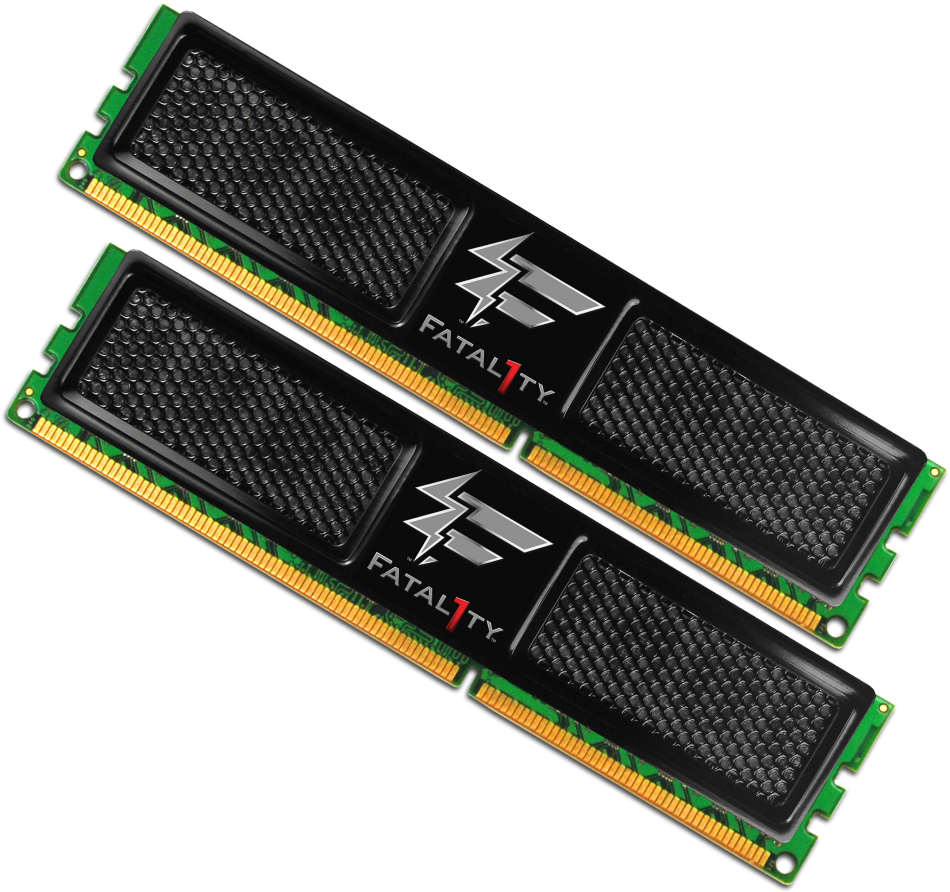
ATX Format Motherboard (Gigabyte GA-EP31)

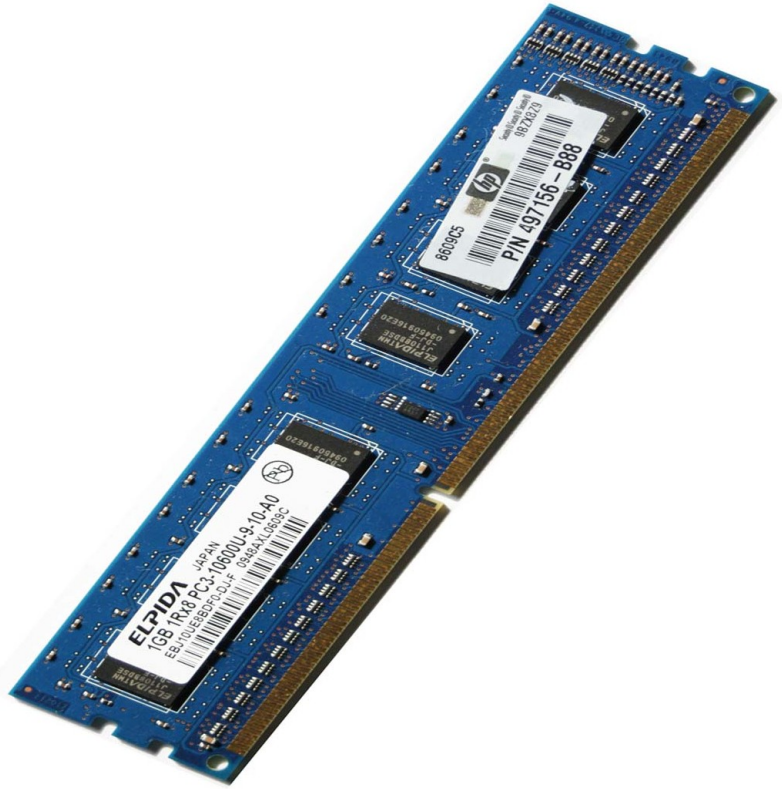


ATX Format Motherboard (Asus Z170 - 2016)



DDR3 SDRAM Main Memory (OCZ; Elpida/HP)





Example Storage Device Hardware

- Some images of contemporary rotating disk and solid state disk hardware;
- All devices have SATA interfaces;
- Images without and without covers to show internal design features;
- All devices in 3.5 inch form factor;
- This is typical post 2010 hardware for desktop applications;

WD Velociraptor 10,000 RPM 3.5 inch Disk

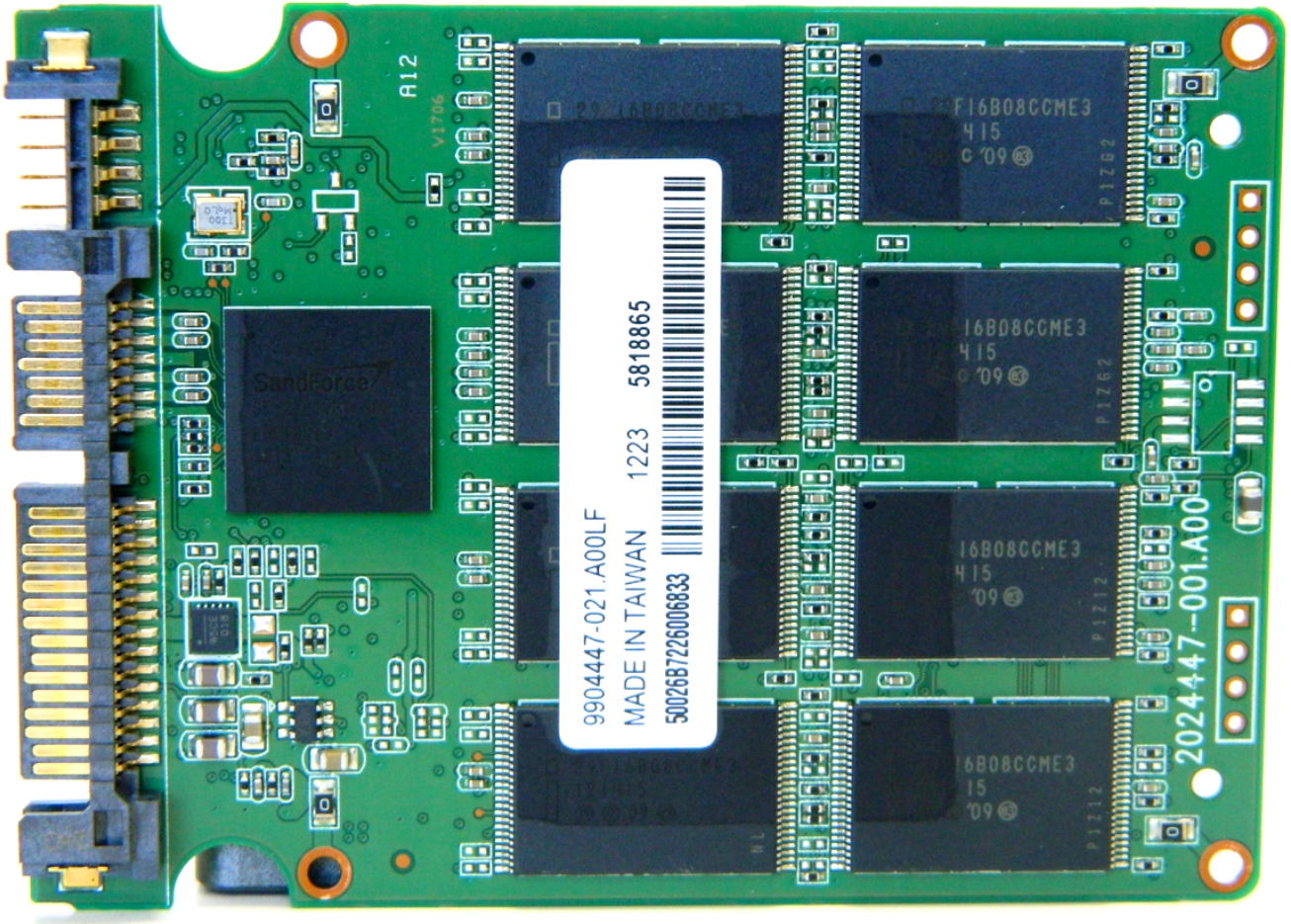
WD Velociraptor 10,000 RPM 3.5 inch Disk

Seagate Cheetah 15,000 RPM 3.5 inch Disk

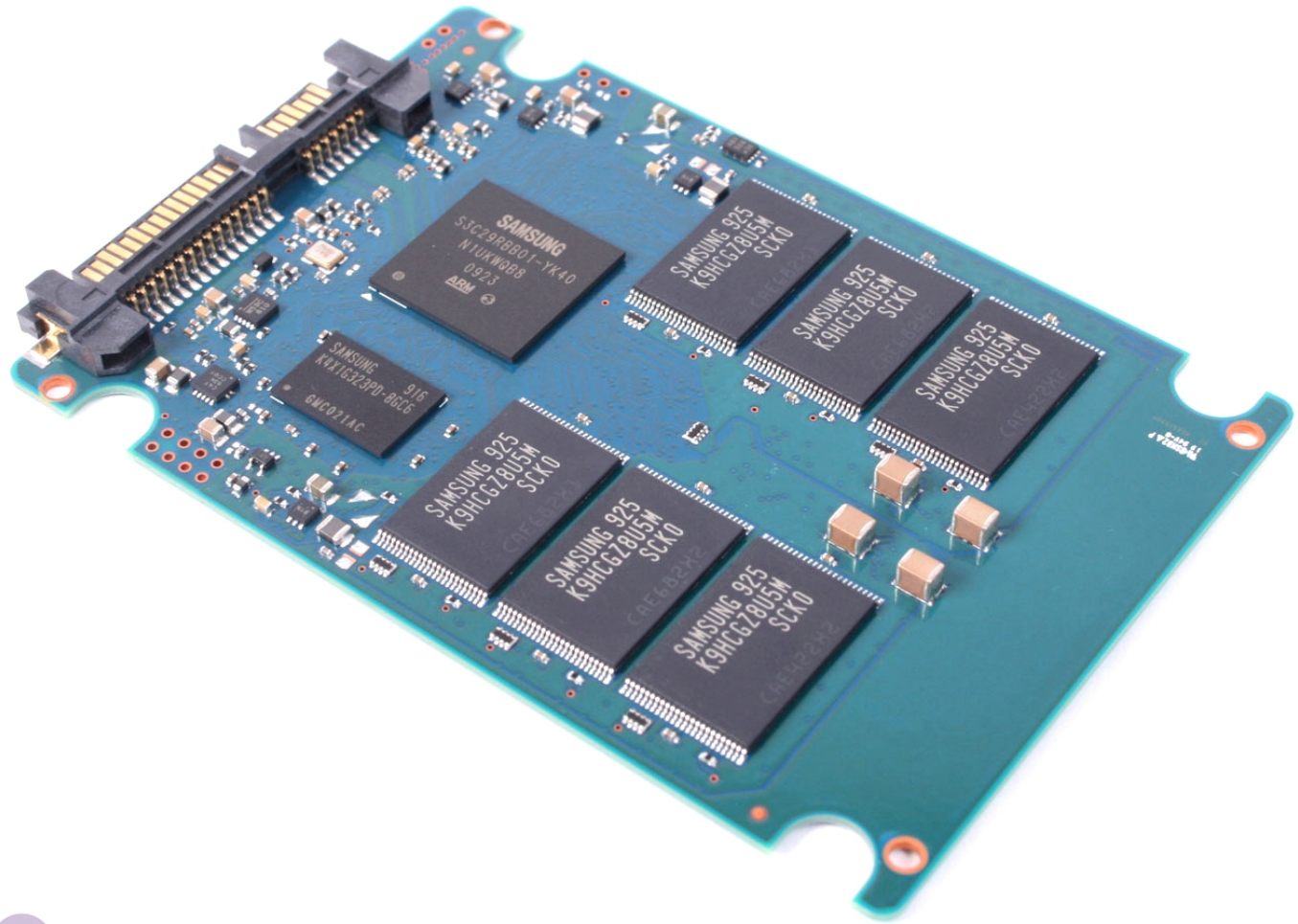
Seagate Cheetah 15,000 RPM 3.5 inch Disk

Kingston SSDNow V+200 SSD – 2.5 inch Format

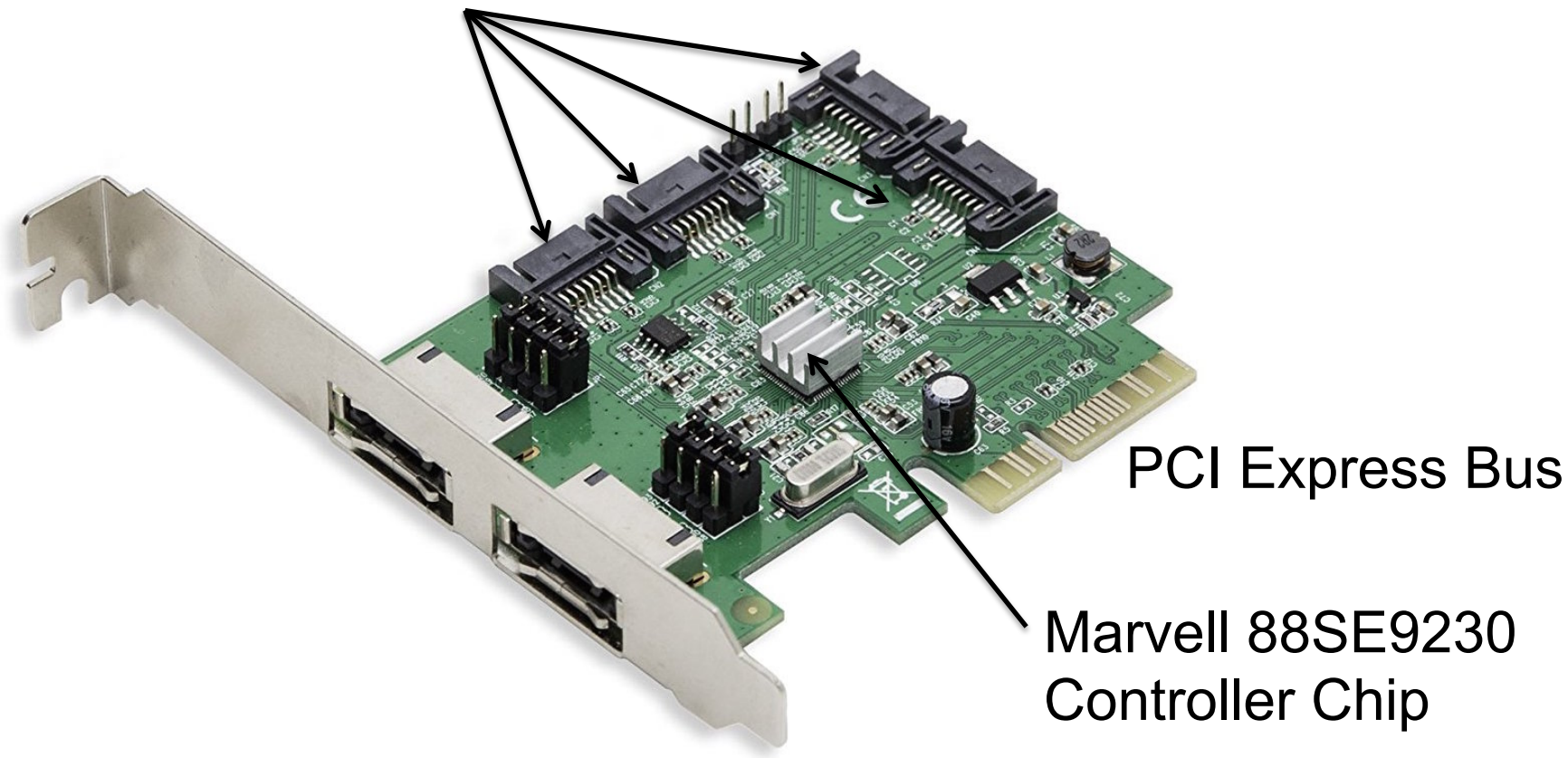
Kingston SSDNow V+200 SSD – 2.5 inch Format



Samsung SSD – 2.5 inch Format

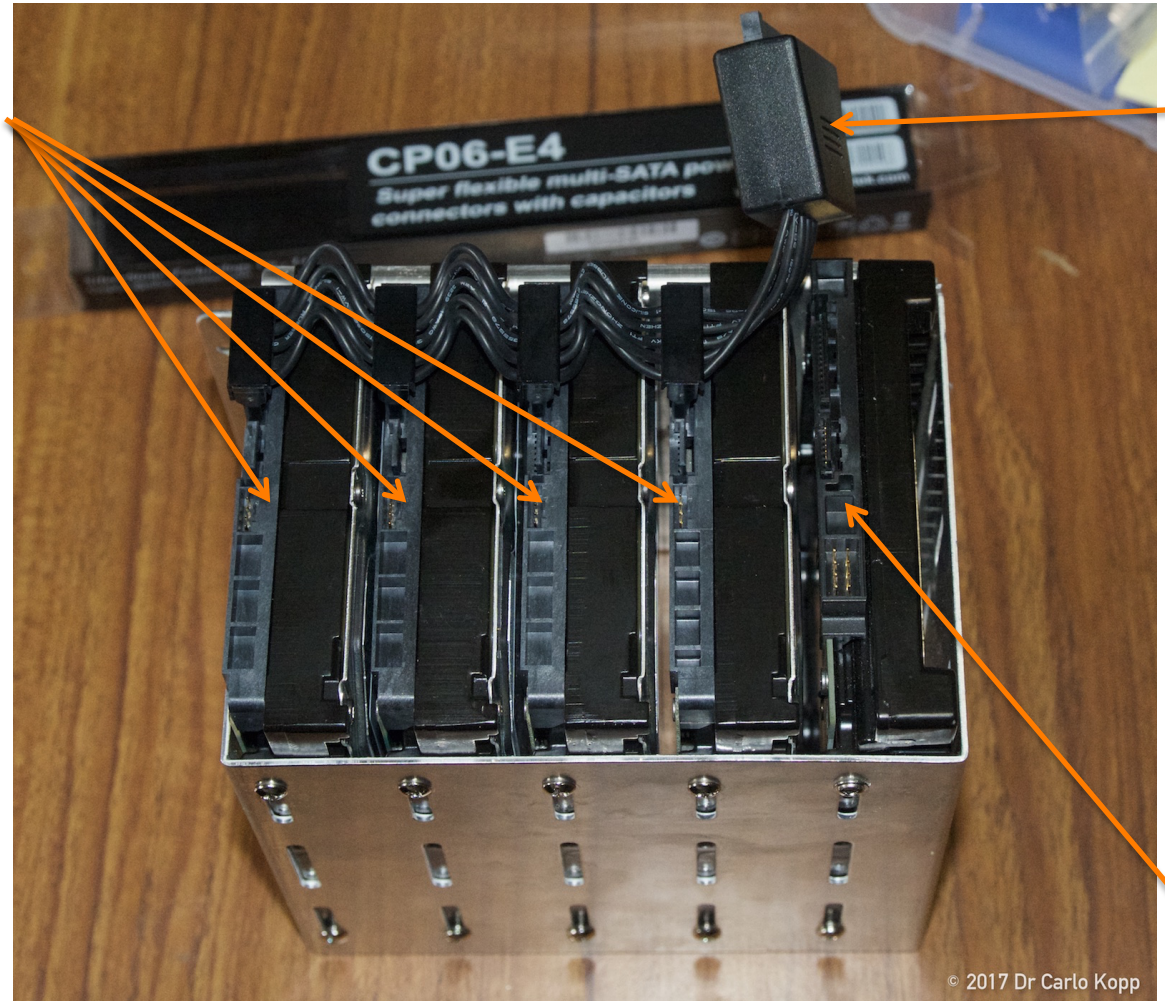


4x SATA3 RAID Controller – Syba SD-PEX40054



Disk Array with 4x SATA2 Drives

4 x SATA2 Disk Array



SATA Power Cable Harness

SATA3 Drive [Not in Array]

Disk Array Cage for 4x 3.5in Drives [Icy Box]