I/O System

- This module begins an overview of I/O
  - How computers handle I/O transactions
  - The I/O system infrastructure
    - Device controllers
    - I/O bus systems
- Functionality as well as performance are important
    - Access to networks and the internet
    - The use of a rich variety of devices
    - Digital cameras
    - Music players
    - Video display devices
    - Printers
    - Storage devices

I/O System

- I/O affects the overall system performance
  - I/O devices are even slower than central memory
  - Overlapping I/O with computation can hide the slowness
- Amdahl's Law applies to I/O as well
  - Limiting the amount of I/O boosts performance
- I/O is defined as a subsystem of components
  - These components move coded data
  - The exchange is between external devices and a host system

I/O System Performance

- I/O performance is measured in two ways:
  - I/O throughput or bandwidth
  - I/O transactions per unit time
- Bandwidth (bytes per second)
  - Depends on clock rate and width of data pathways
  - Important when large amounts of data need to be exchanged
- Transactions per second (TPS)
  - Important when numerous small data exchanges are made

I/O System Performance

- Overall system performance depends on the interaction of all of its components
- Improving the most heavily used components is most effective
- This idea is quantified by Amdahl's Law:

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

S is the overall speedup;
f is the fraction of work performed by a faster component; and
k is the improvement of the faster component

Example: if a component is made 100% faster, then k=2.

Example

Facts:

Processes spend 70% of their time running on the CPU and 30% of their time waiting for disk service

Options:

make the CPU 50% faster for $10,000

make the disk drives 150% faster for $7,000

Question:

Which option would be better based on benefit and cost?

I/O System Performance

- The processor option offers a 30% speedup:

$$f = 0.70, \quad k = 1.5 \qquad S = \frac{1}{(1 - 0.7) + 0.7/1.5} = 1.30$$

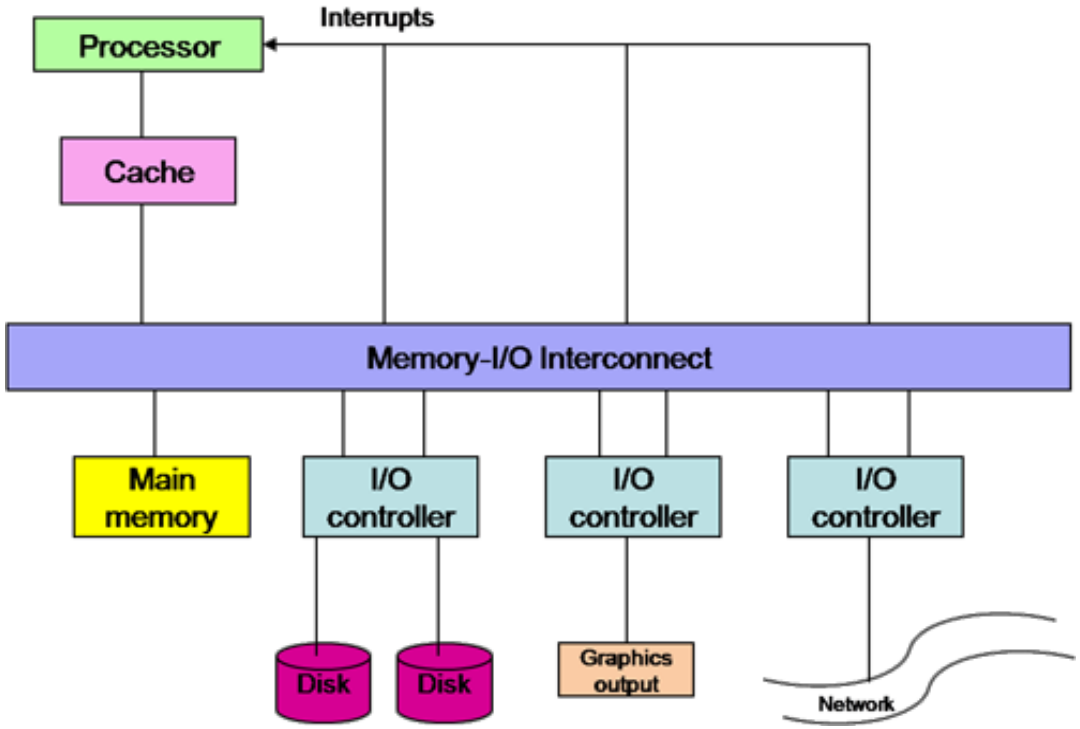
- And the disk drive option gives a 22% speedup:

$$f = 0.30, \quad k = 2.5 \qquad S = \frac{1}{(1 - 0.3) + 0.3/2.5} = 1.22$$

- Each 1% of improvement for the processor costs $333 (10000/30), and for the disk a 1% improvement costs $318 (7000/22).

I/O System

I/O controllers manage communication between I/O devices and the CPU or main memory.



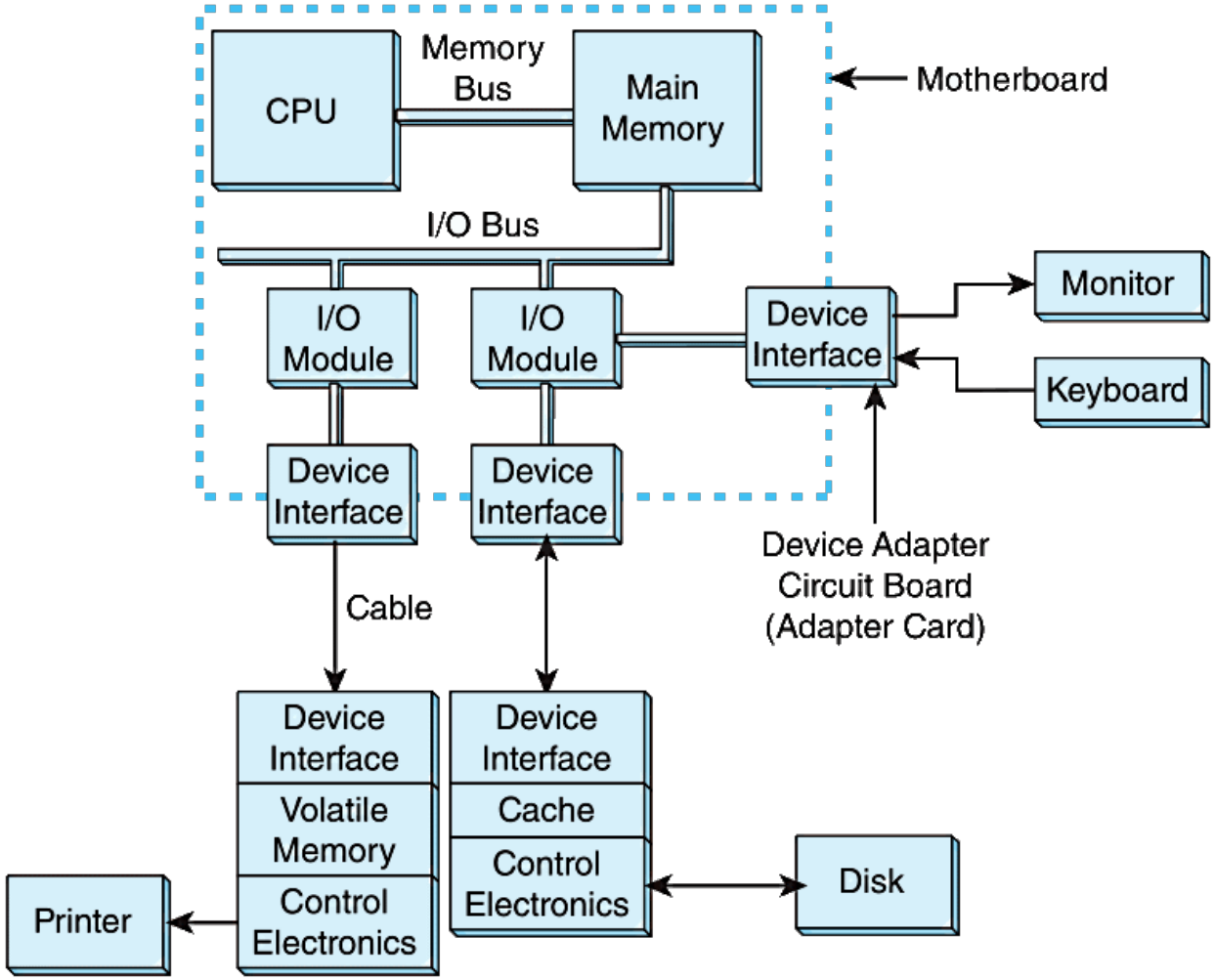
I/O transactions adhere to well-defined rules (*protocol*)

I/O System

I/O subsystems include:

- Blocks of main memory that are devoted to I/O functions.
- Buses that move data into and out of the system.
- Control modules in the host and in peripheral devices
- Interfaces to external devices such as keyboards and disks.
- Cabling or communications links between the host system and its peripherals.

I/O Architecture



Various types of buses allow for communication among components.

I/O Access Methods

- Devices can be accessed in one of two ways:

1 *Memory mapped*

  - One or more registers are assigned to each I/O device
  - The registers correspond to specified memory addresses
  - Part of the address space is thus reserved for I/O
  - Memory access instructions can also perform I/O
  - RISC systems tend to use this approach

2 *Port mapped or Isolated I/O*

  - Each device is assigned one or more port numbers
  - Special I/O instructions transmit data via the ports
  - Access must be identified as memory or I/O accesses
  - CISC systems tend to use this approach

I/O Access Examples

MIPS example:

```
    lui    $t0,0xFFFF    # base address for keyboard
    lw     $t1,4($t0)    # read the next input character
```

Pentium example:

```
KB_DATA    EQU   60H   # port number for keyboard

    in     AL,KB_DATA     # read character from port into AL register
```

Conclusion

Next we will examine more details of low level I/O

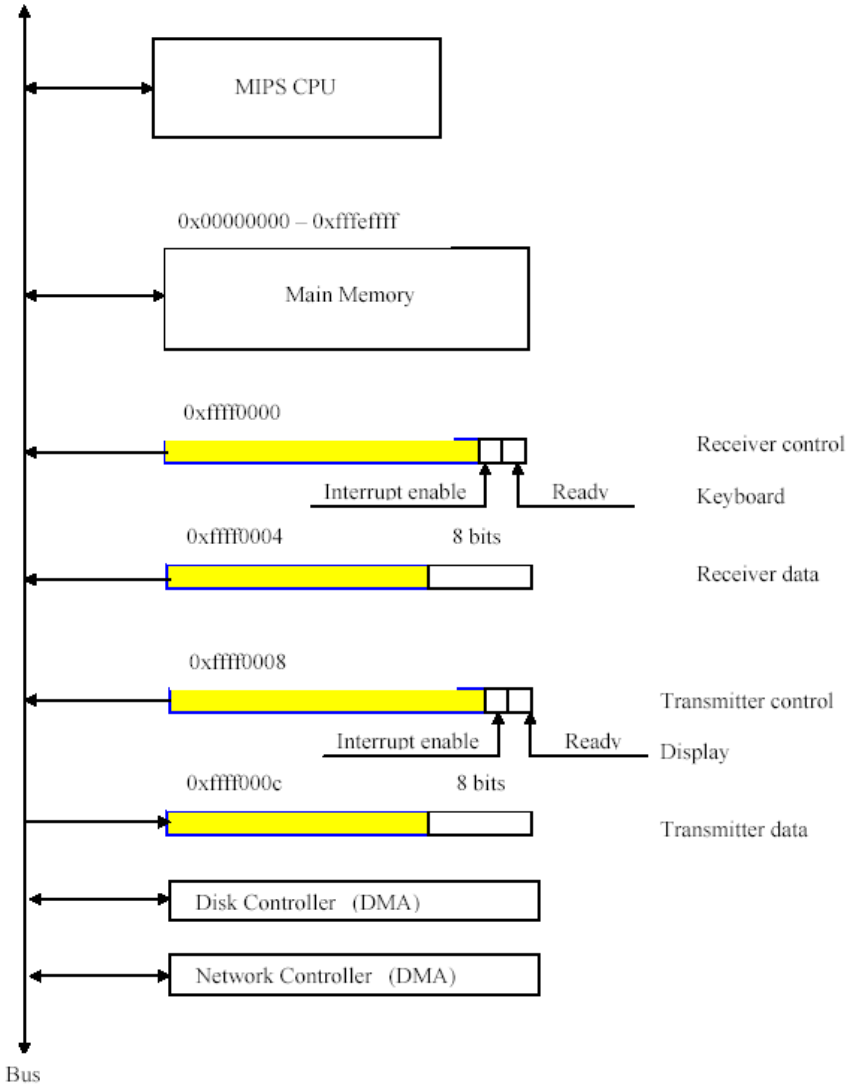
- Programmed I/O
- Interrupt Driven I/O
- DMA

I/O Polling

- Direct program controlled I/O uses polling
  - Device status registers are repeatedly checked
  - CPU is kept busy by the polling operations
- Each device is assigned one or more registers
  - Status register
  - Control register
  - Data register
- Simple devices have fewer registers
  - Keyboard
  - Console display
- Complex devices have more registers

I/O Device Registers

Example memory-mapped system.



I/O Polling Example

Detecting key presses using polling:

```
# device register base address = 0xFFFF0000
          lui    $t3,0xFFFF    # KB status address
poll_CR:  lw     $t1,0($t3)    # read status register
          andi   $t1,$t1,1     # does LSB=1?
          beqz   $t1,poll_CR   # keep checking if not
          lw     $t0,4($t3)    # else read character into $t0
```

Time between keystrokes can be considerable

The status bit is automatically cleared when the data register is read

I/O Polling Example

Output to the console display using polling:

```
          lui   $t3,0xFFFF    # device reg base address
          li    $t0,'>'       # ASCII code for output char
Poll_XR:  lw    $t1,8($t3)    # Console Cntrl reg offset=8
          andi  $t1,$t1,1     # does LSB=1?
          beqz  $t1,poll_XR   # keep checking if not
          sw    $t0,12($t3)   # display the character
```

Status register is polled to see if the console is ready
Ready bit goes to 0 when data register is written
Ready bit goes back to 1 once the character has been displayed

Polling keeps the CPU busy
Using interrupts instead allows the CPU to do other useful work

Interrupt Driven I/O

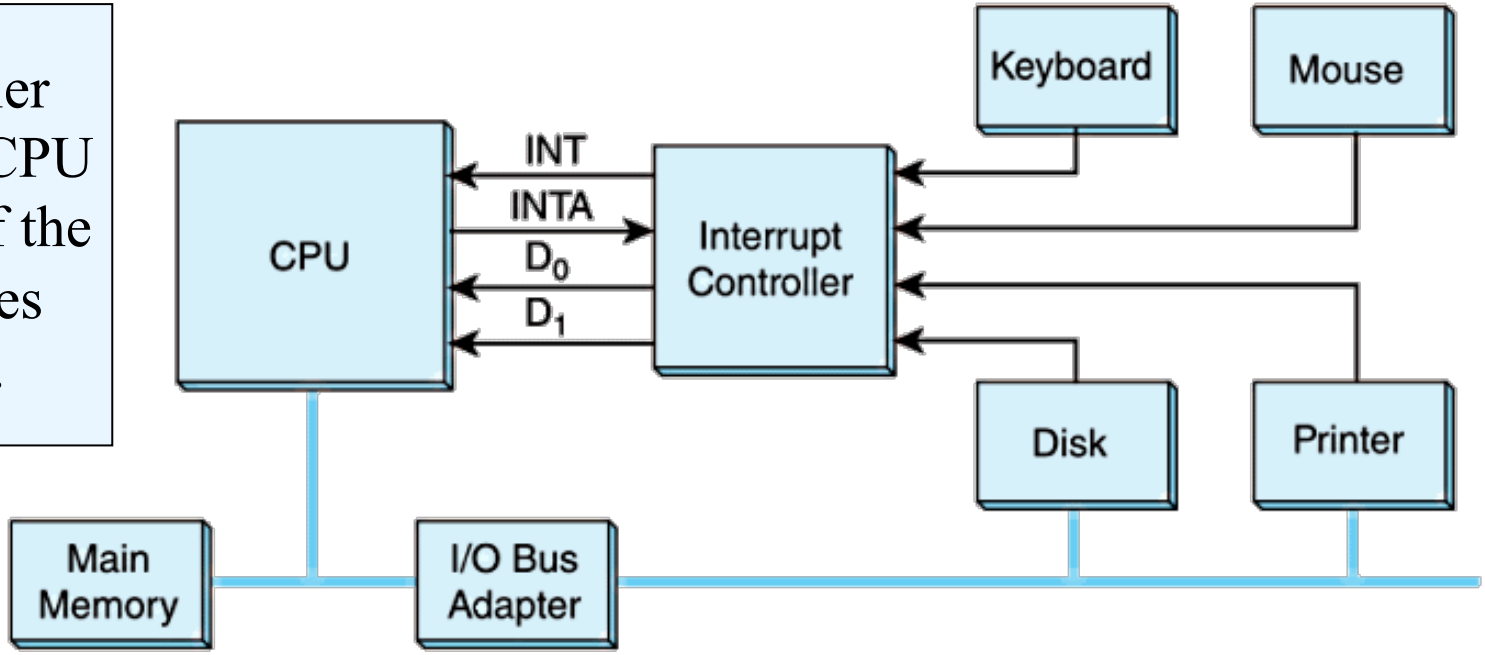
- CPU responds only when I/O request is made
  - Each device is assigned an ID
  - Each may have a different priority level
  - Interrupt signals are sent to notify the CPU of I/O completion
- Some systems employ vectored interrupts
  - Each corresponds to a different interrupt handler address
  - The Interrupt service routine (ISR) is called via the supplied address
- The MIPS uses a single vector address
  - Control goes to this vector address for all exceptions
  - MIPS interrupts are a particular type of exception

Interrupt Driven I/O

This is an idealized I/O subsystem that uses interrupts.

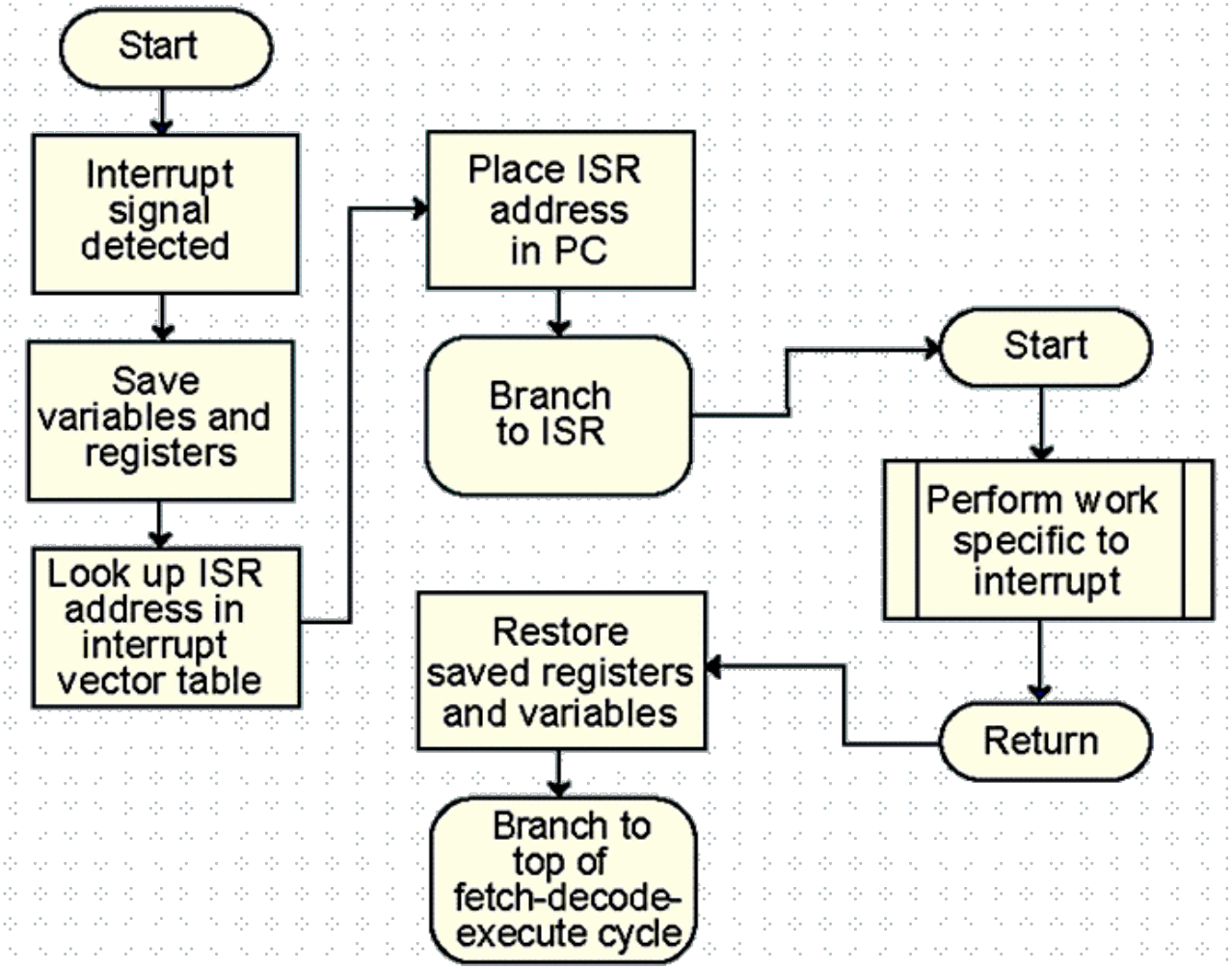
Each device connects its interrupt line to the interrupt controller.

The controller signals the CPU when any of the interrupt lines are asserted.

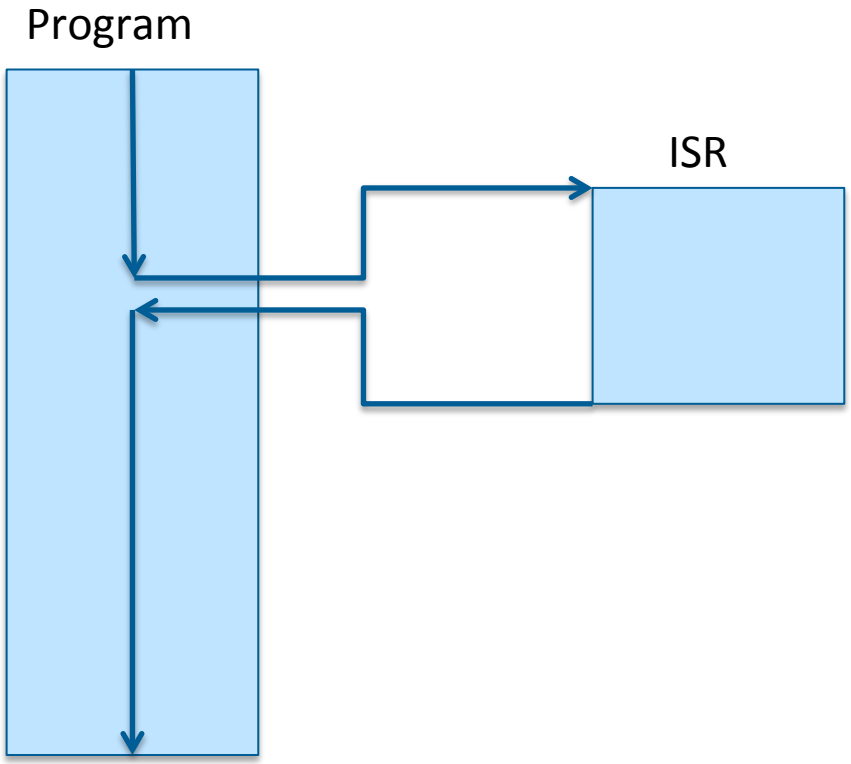


The system's state is saved before the interrupt service routine is executed and is restored afterward (e.g. all registers)

Interrupt Driven I/O



Interrupt Driven I/O

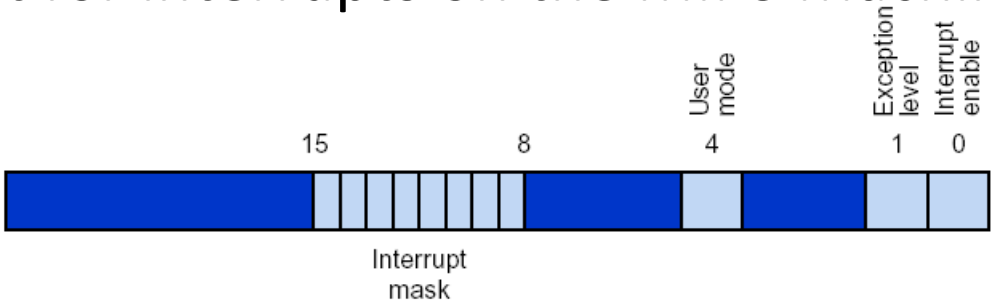


Interrupts are like unsolicited procedure calls
The program may be unaware that an interrupt occurred
Interrupts are delayed until the current instruction completes

Interrupt Driven I/O

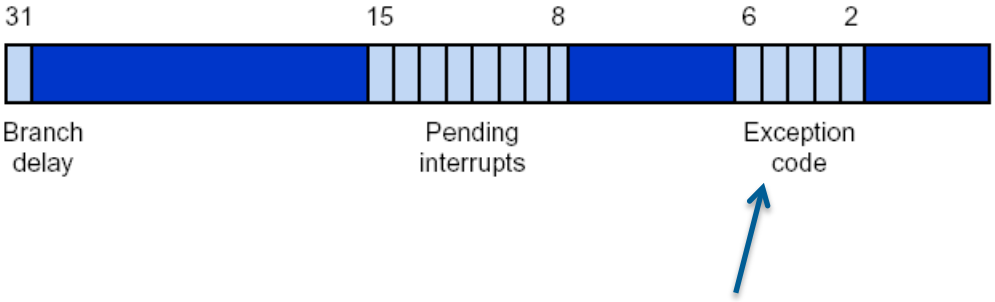
Support for interrupts on the MIPS Machine

CP0 register $12



Status Register Format

CP0 register $13

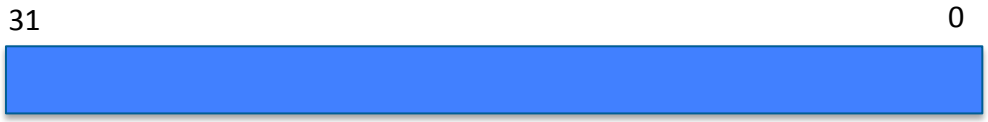


Cause Register Format

Exception code = 0 for interrupts

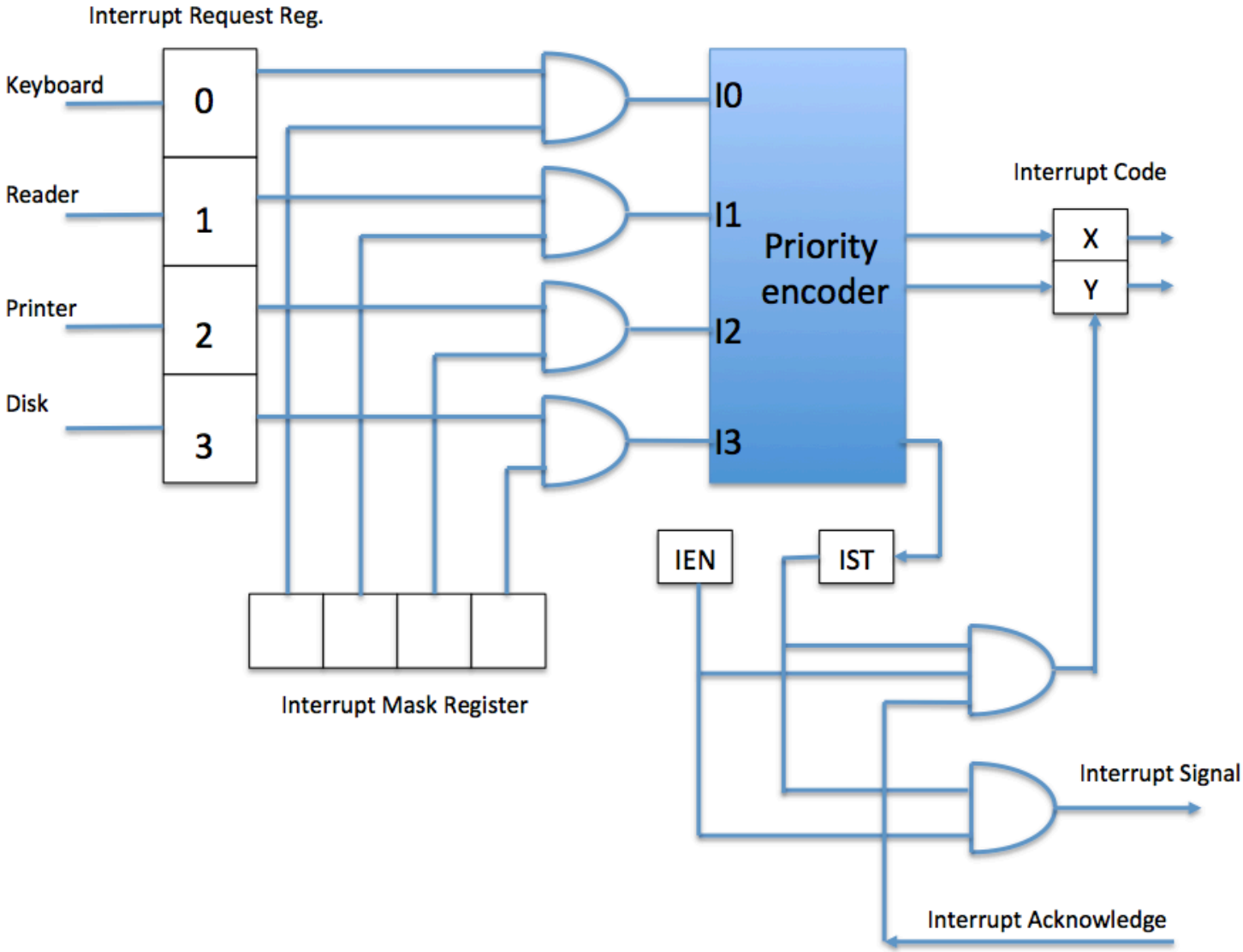
Non-zero value indicates other types of exceptions

CP0 register $14



EPC exception program counter

Interrupt Controller



DMA Devices

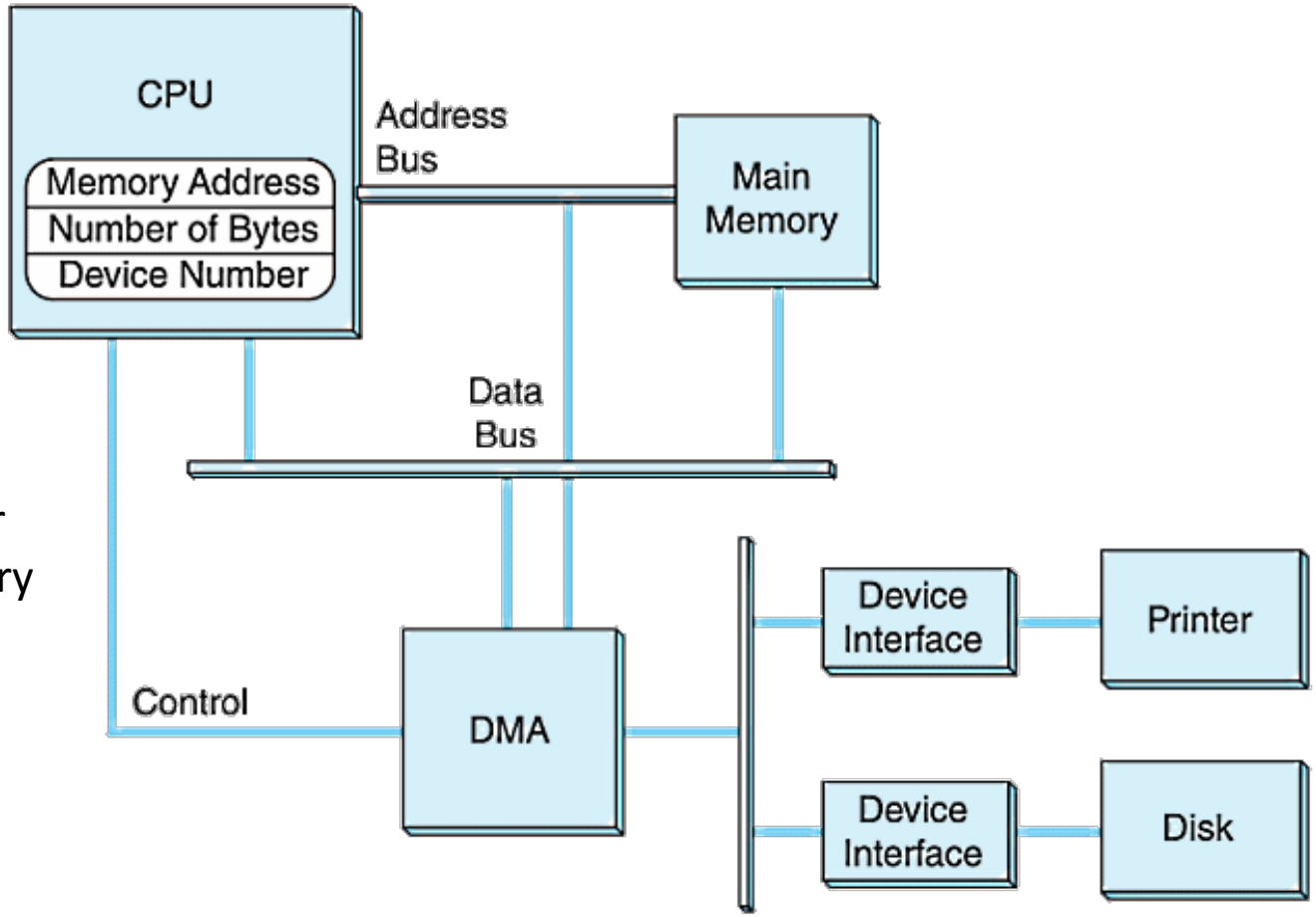
- Some devices can directly access memory
  - Bypasses the CPU and goes directly to memory
- The CPU is only involved at the beginning and at the end
  - DMA Device ID or address
  - Memory address (source or destination of data)
  - Size of block transfer
  - Direction (input or output)
- A single interrupt occurs at the end of the transaction
  - DMA controller decrements count and adjusts pointer to memory
  - Consumes less CPU time than with interrupt after each byte or word

DMA Devices

This is a DMA configuration.

Notice that the DMA and the CPU share the bus.

The DMA runs at a higher priority and steals memory cycles from the CPU.



The DMA does not go through the cache, so the cache contents could be made stale.

DMA Modes

Ways CPU and DMA device can share memory bus:

- Transparent mode
  - Bus is used for DMA only when not needed by CPU
  - This is the ideal theoretical case
  - Can't really predict when CPU will not need the bus
- Cycle stealing mode
  - CPU frees bus long enough for DMA of one data unit
  - Cycles are given up by CPU to allow DMA transfers
- Burst mode
  - CPU relinquishes bus long enough for a block transfer
  - Feasible if CPU is actively getting cache hits

I/O Processors (IOP)

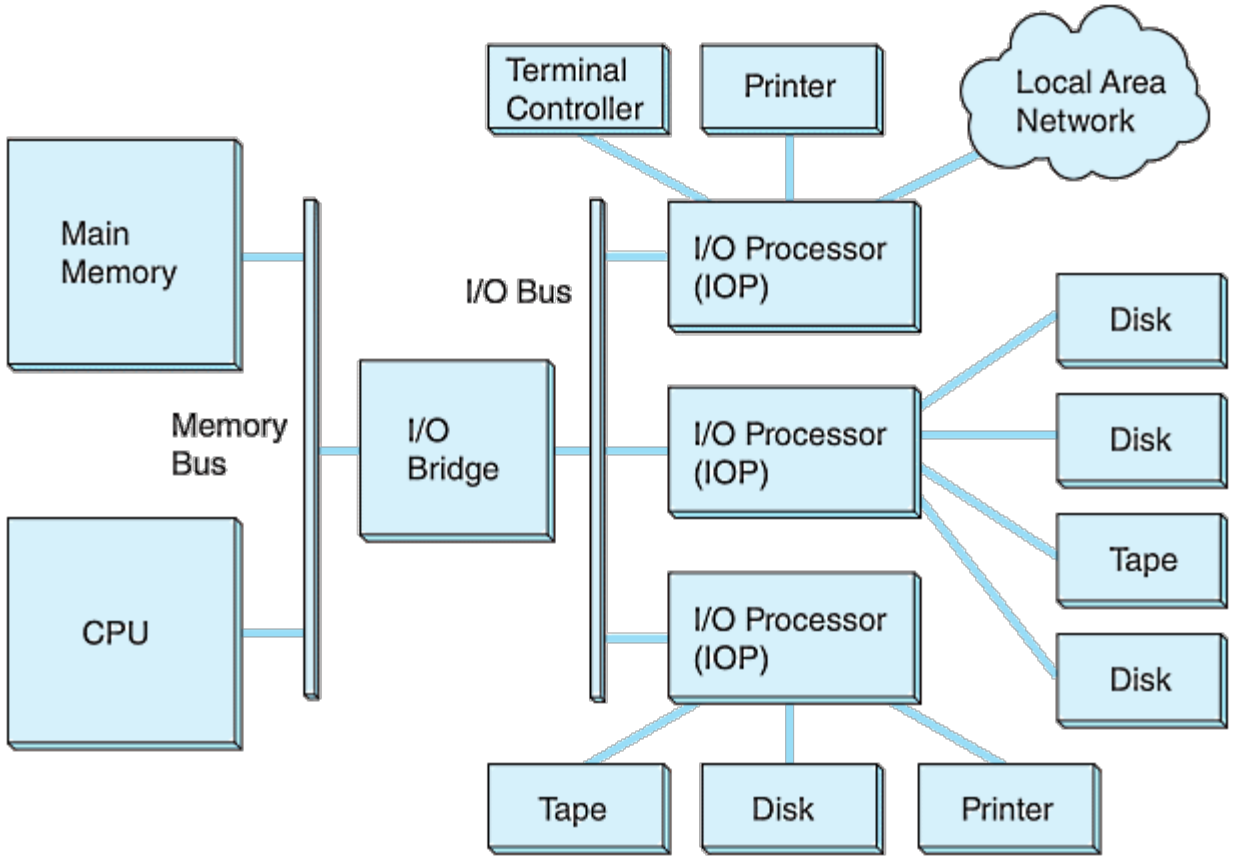
- IOPs are also known as I/O channels
  - They have their own instruction set tailored for I/O
  - They can carryout a series of multi-step transactions
  - They run in parallel with and independent of the CPU
  - They transfer entire files or groups of files
- “channel command” IBM’s term for I/O instruction
- Intel 8086 had a companion 8089 I/O processor
- Relieves the CPU of most I/O duties
  - Less burden on CPU than interrupts or DMA

I/O Processors (IOP)

- IOPs have more intelligence than DMA controllers
  - They negotiate protocols
  - issue device commands
  - translate storage coding to memory coding
  - transfer entire files or groups of files independently of the host CPU
- The host CPU creates the I/O program
  - These are I/O specific instructions
  - CPU tells IOP where to find the I/O program

I/O Processors (IOP)

- This is a channel I/O configuration.



Issues with I/O and Cache

- Memory mapped device registers are accessed like any other data items.
  - Copies of the registers in the cache may be stale
  - CPU accesses go through the cache
  - I/O device accesses do not go through the cache
- DMA controllers use physical addresses
  - Network packet buffers could be a problem if cached
  - DMA transfer goes directly to memory
  - If the buffer maps to cache, the CPU may be unaware

Issues with I/O and Cache

- Some CPUs have a cache flush instruction
  - This invalidates one or more the cache lines
  - References to the items then go directly to memory
- An alternative is to use uncached areas
  - Specified address ranges used by the CPU do not go through cache
  - I/O device registers and I/O buffers could be assigned to these uncached areas

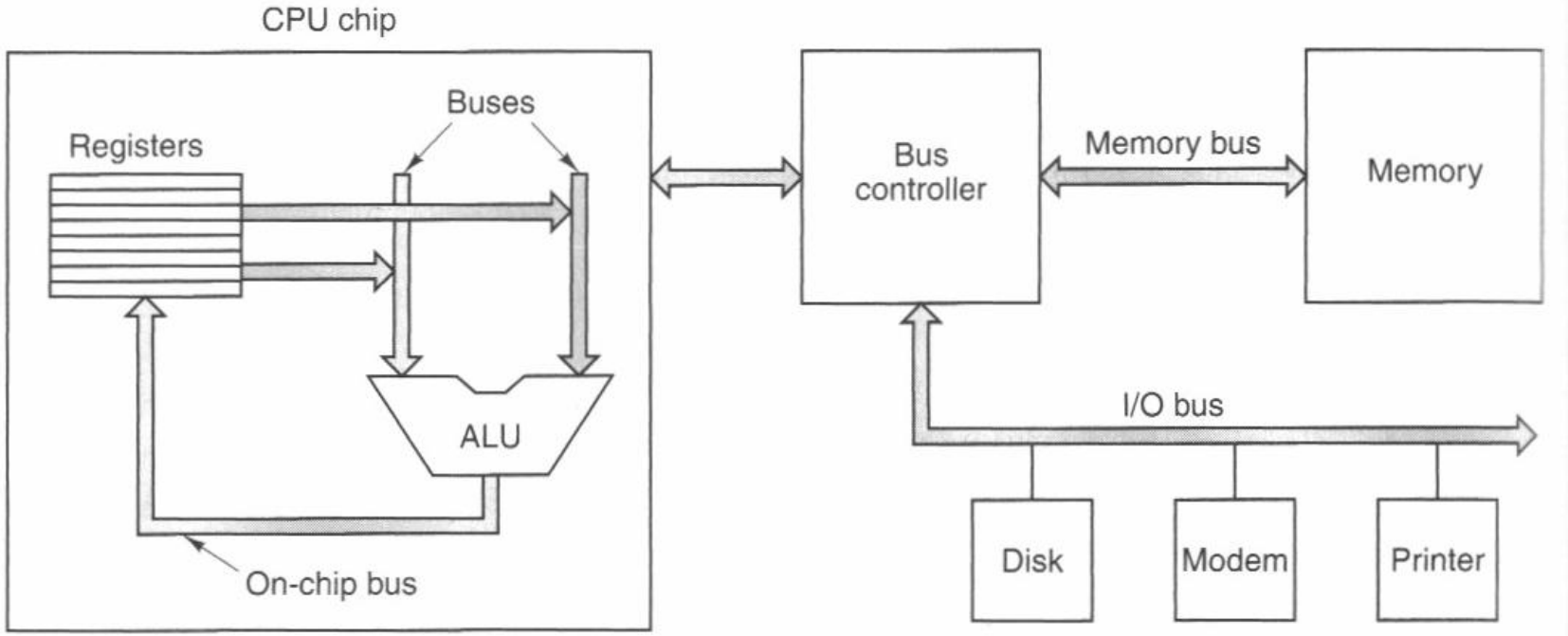
Conclusion

- Next we will examine bus systems in more detail
  - Buses are required to access memory and I/O devices
- We will also examine disk arrays (RAID systems)

Bus Systems

- Components exchange information via buses
  - A bus is a set of shared lines or electrical traces
  - Chips have pins that connect to these shared lines
- Computers use a hierarchy of buses
    - Some are internal to the CPU
    - External buses are used for I/O
- Bus protocols are rules for using the bus
  - specify timing techniques
  - Arbitration rules to resolve conflicting requests
  - Synchronous versus asynchronous communication

Bus Systems

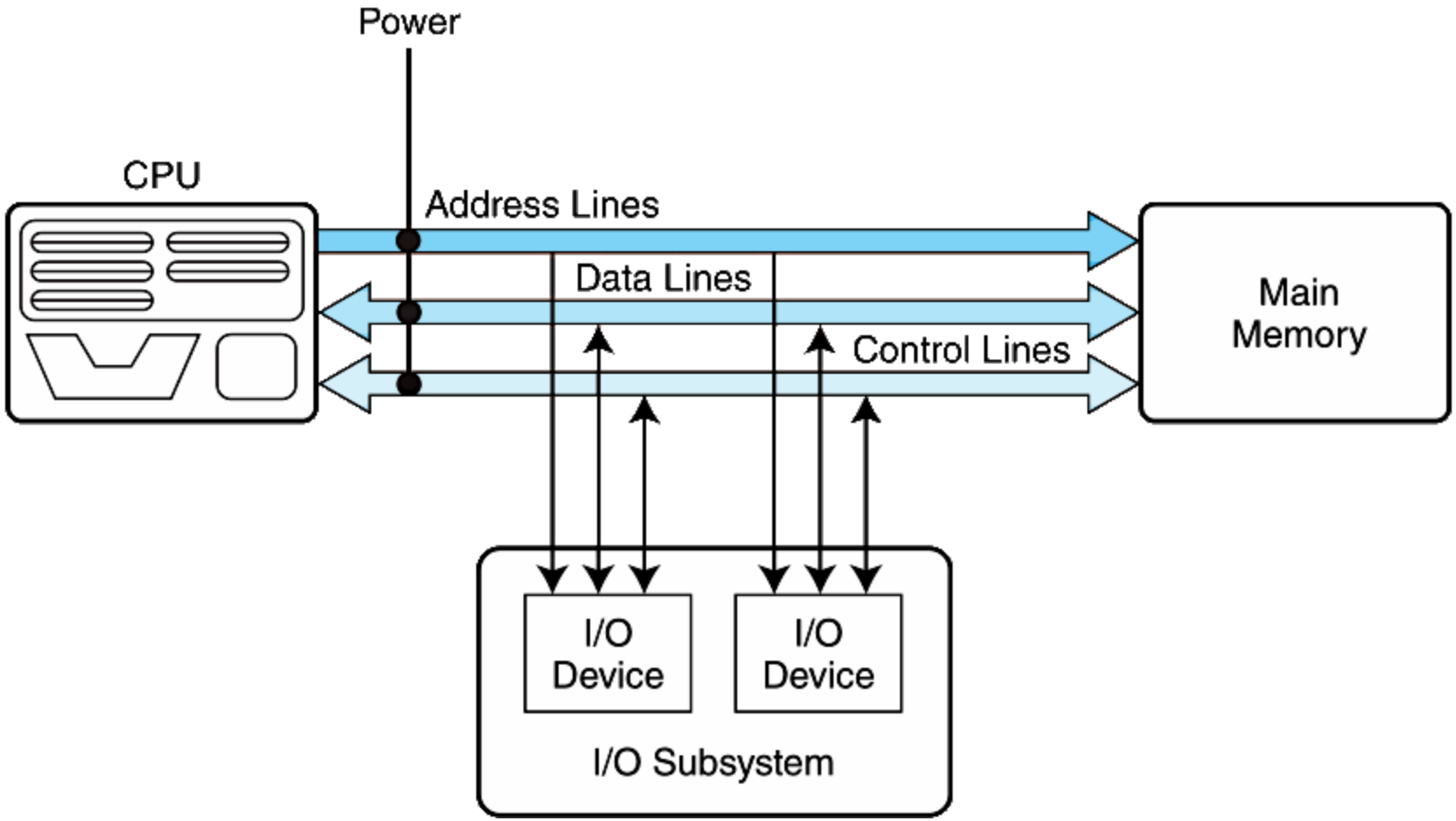


- CPU-to-memory buses are parallel synchronous buses
  - Operate at higher rates than I/O buses
  - Clock signals synchronize the exchanges
- I/O buses can be parallel or serial
  - Operate at rates that match slow I/O device speeds
  - Communicate asynchronously using handshake signals

Bus Systems

- Buses consist of:
  - data lines
  - control lines
  - address lines
- Data lines convey bits from one device to another
- Signals on control lines determine:
  - the direction of data flow
  - when each device can access the bus
- Address lines determine the location of the source or destination of the data

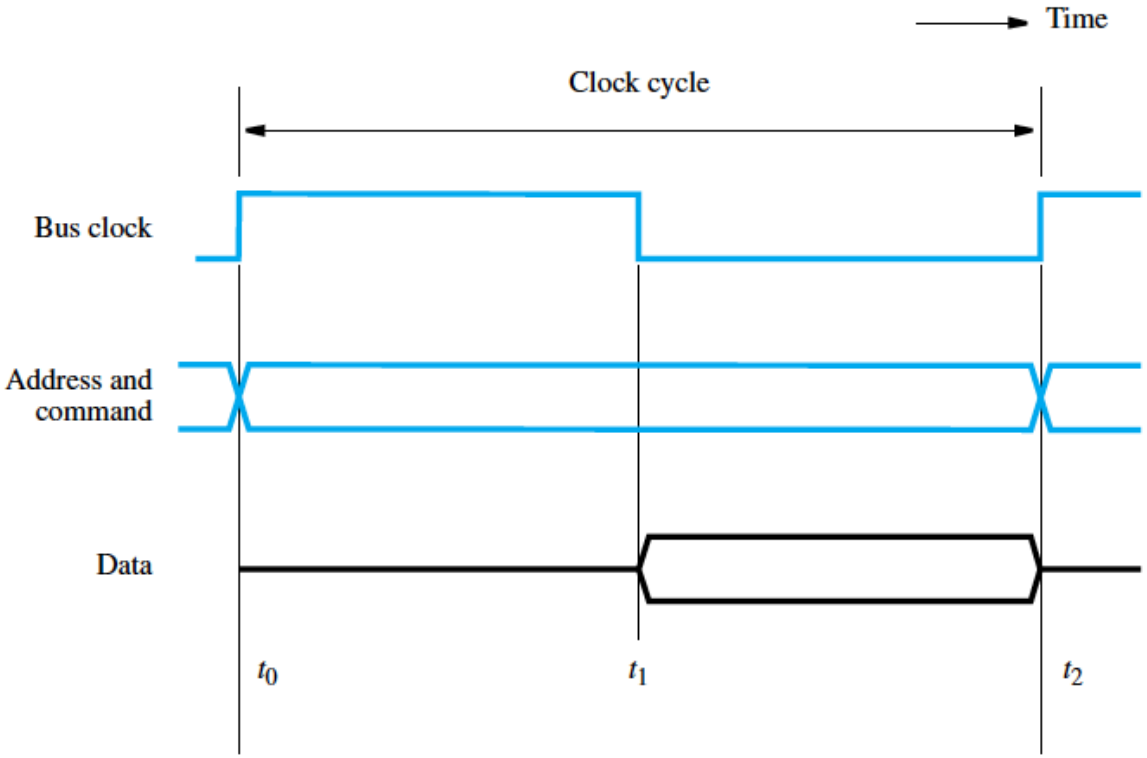
Bus Systems


Power
CPU
Address Lines
Data Lines
Control Lines
Main Memory
I/O Device
I/O Device
I/O Subsystem

Bus Systems

- Buses may be serial or parallel
  - Bits are sent simultaneously along each parallel bus line
  - Bits are sent one after the other along a serial bus line
- Bus bandwidth = amount of data transferred per unit time
  - Depends on width (number of data lines)
  - Depends on cycle time and number of cycles used
- A common clock signal is used by *synchronous* buses
  - Transactions adhere to a bus schedule
    - The schedule must accommodate the slowest device
    - Slowest device dictates the cycle time
  - Clock skew can be a problem
    - Different devices may not see the clock signal at the exact same time

Synchronous Bus Read



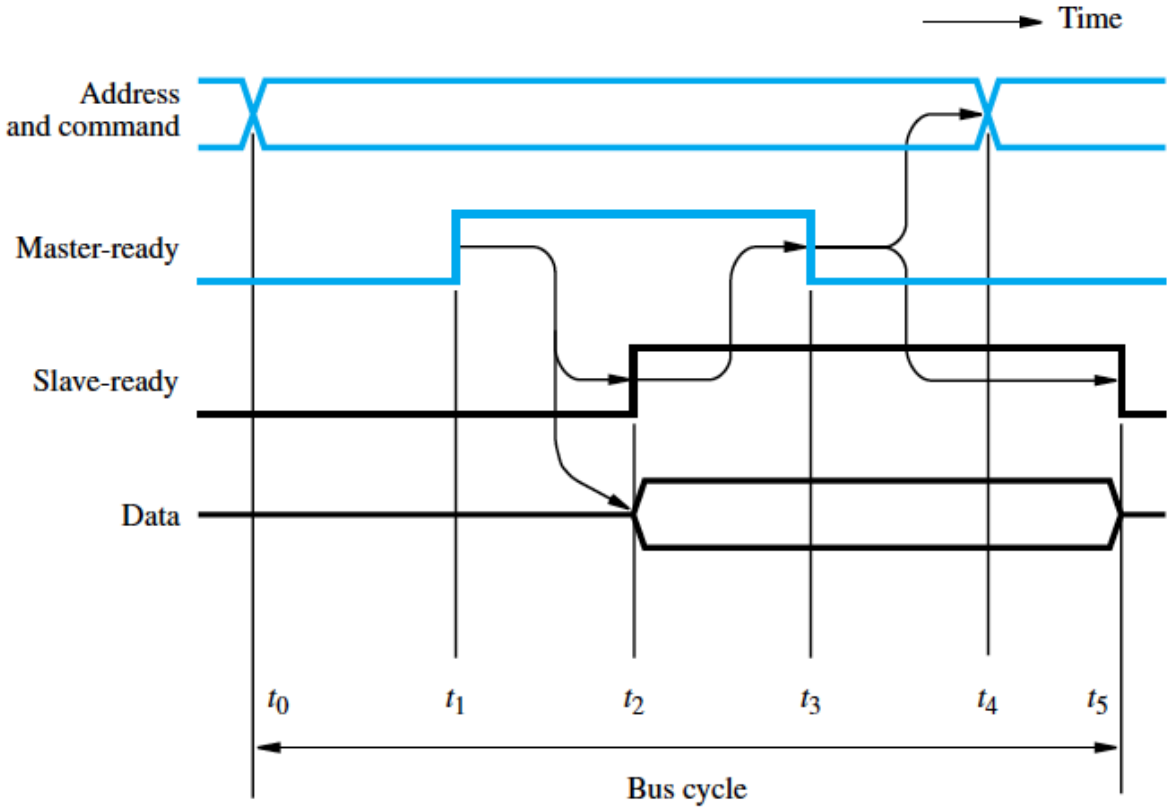
Data requested from the specified address is available at t_1

If data cannot be provided in time, “wait states” are inserted

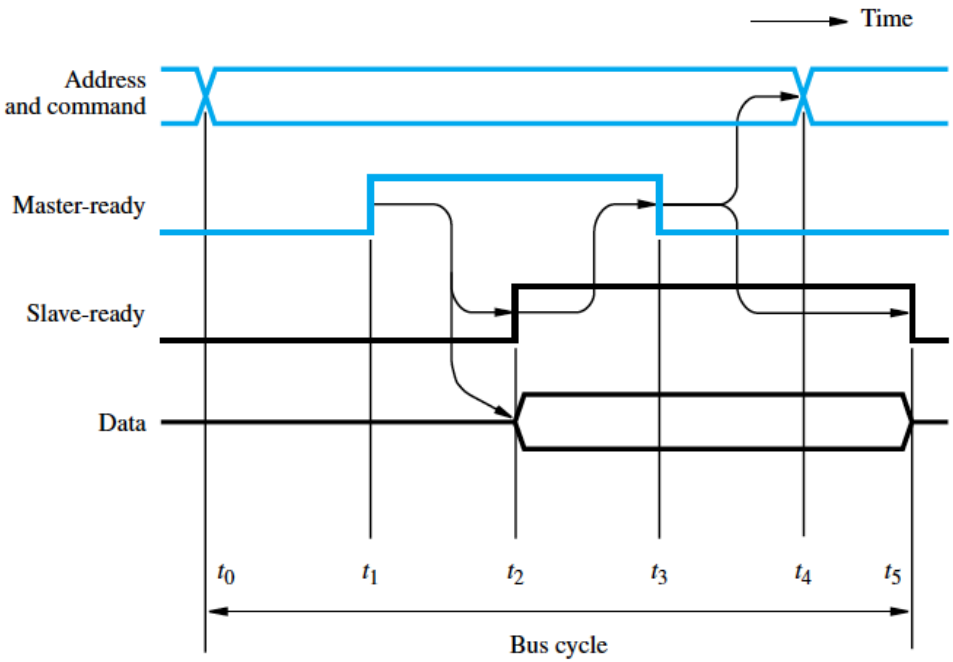
Wait states are additional complete clock cycles

Bus Systems

- *Asynchronous* buses exchange handshake signals
  - Each step in the transaction requires a signal or response
  - transaction speed varies with the devices involved (self-paced)



Asynchronous Bus Read



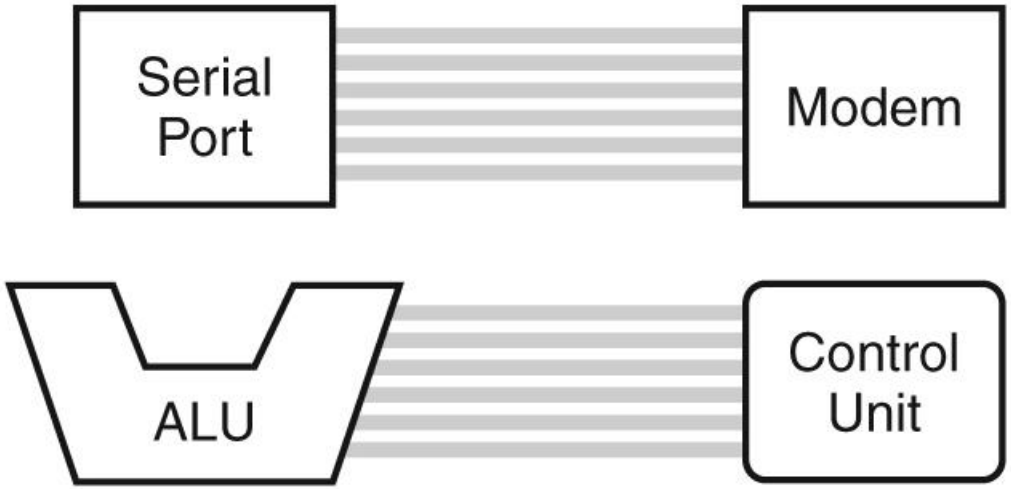
Master-ready is asserted to signal valid address and command
Slave-ready is asserted when requested data is available
master-ready is de-asserted in response to receiving the data
Slave-ready is de-asserted to allow next transaction to begin

A positive action is required to proceed to the next step

Bus Systems

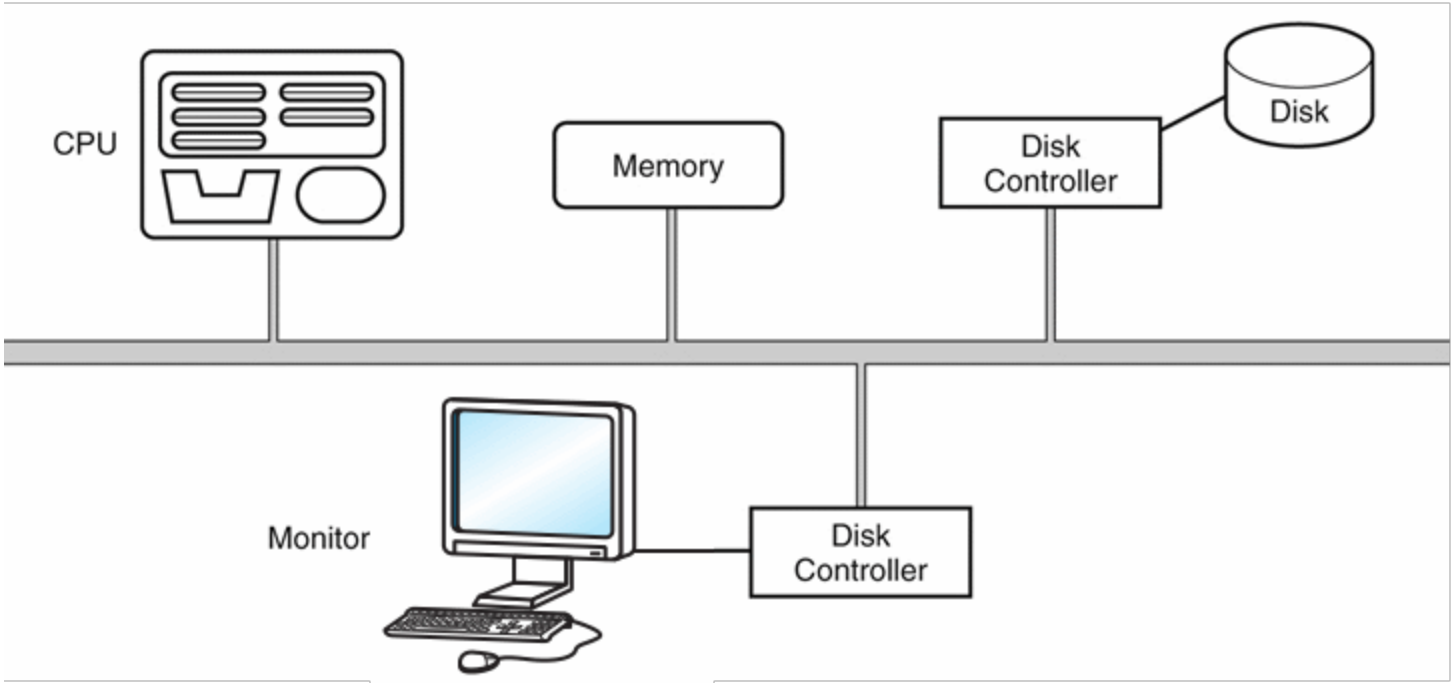
- Two types of buses are commonly found in computer systems: *point-to-point*, and *multipoint* buses.

These are point-to-point buses:



Bus Systems

- A multipoint bus is a shared resource
- Protocols can be built into the hardware
  - They control access to the bus

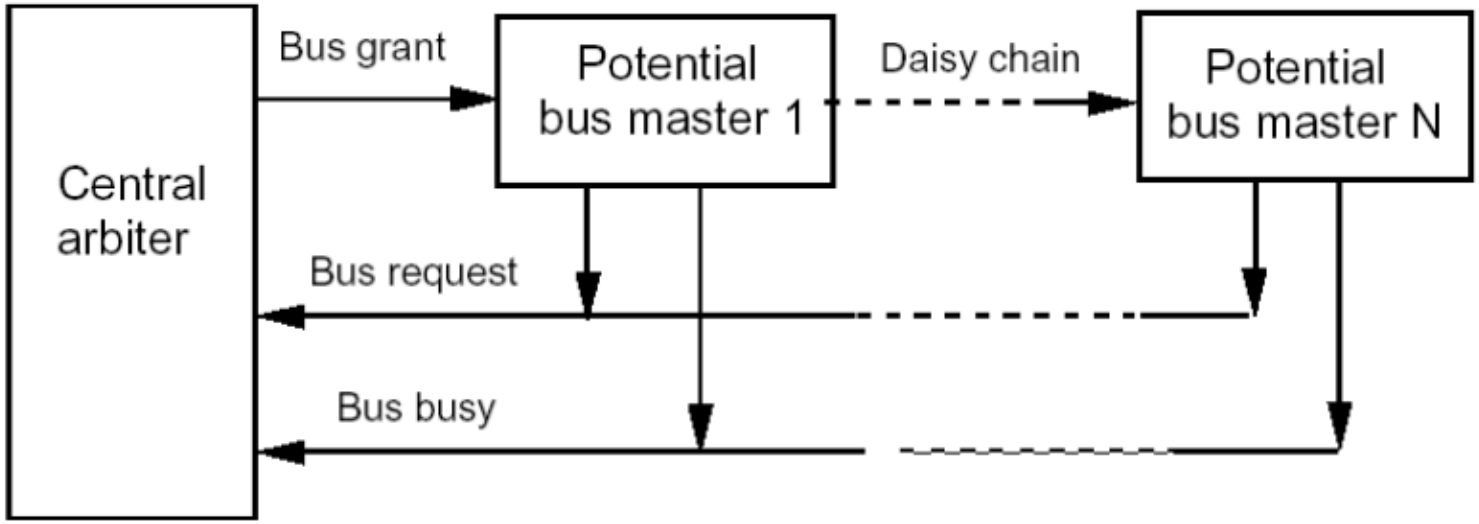


Bus Systems

- Bus transactions:
  - All the actions needed to exchange information
  - Occurs in a “bus cycle”
    - May correspond to multiple clock cycles
- Only one device at a time controls the bus
  - Called an “initiator” or “master”
  - Initiator or master communicates with “target” or “slave”
- Arbitration resolves competing requests for control

Bus Arbitration

Daisy chain: Permission passed along the chain of devices



Priority is based on proximity to arbiter

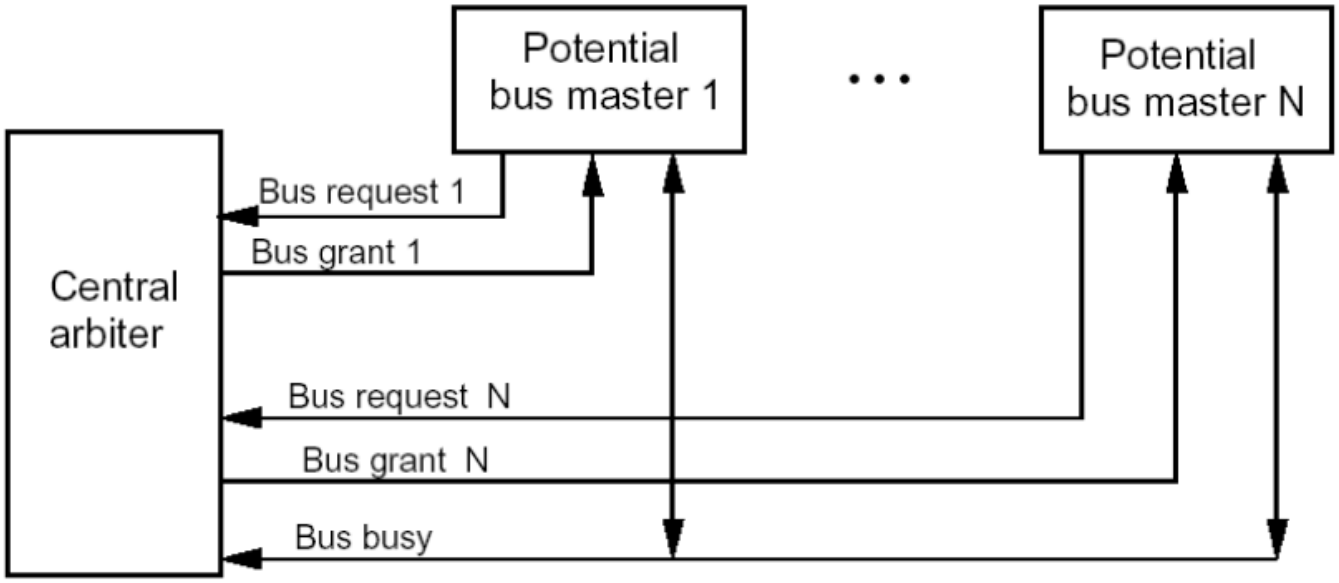
Malfunctioning device can lockout others that follow

Requires relatively few bus lines

Bus Arbitration

Centralized parallel:

Each device is directly connected to the arbiter



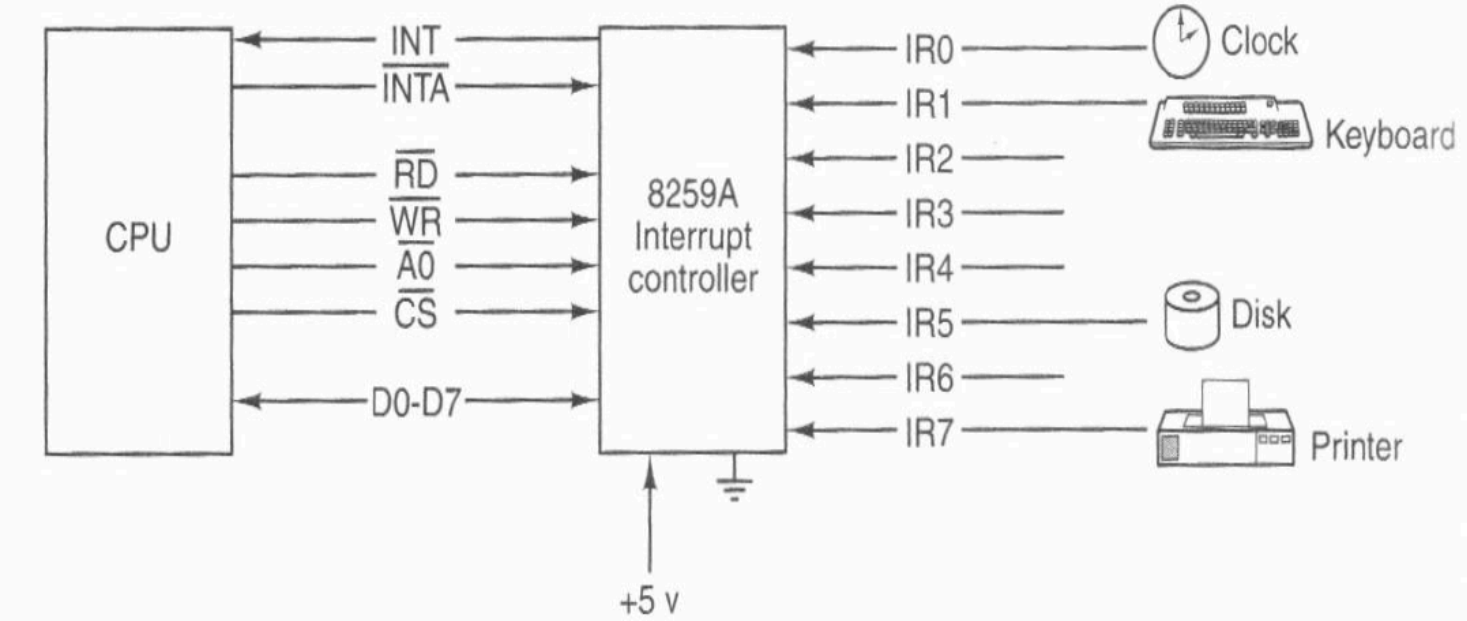
Arbiter assigns priority based on separate grant lines

Malfunctioning devices can be ignored

Requires more bus lines than daisy chained alternative

Bus Arbitration

Centralized parallel example:



Up to 8 I/O controller chips can be connected

Using the 8 IRQ (interrupt request) lines of the 8259A controller

Used on early PC systems (Intel)

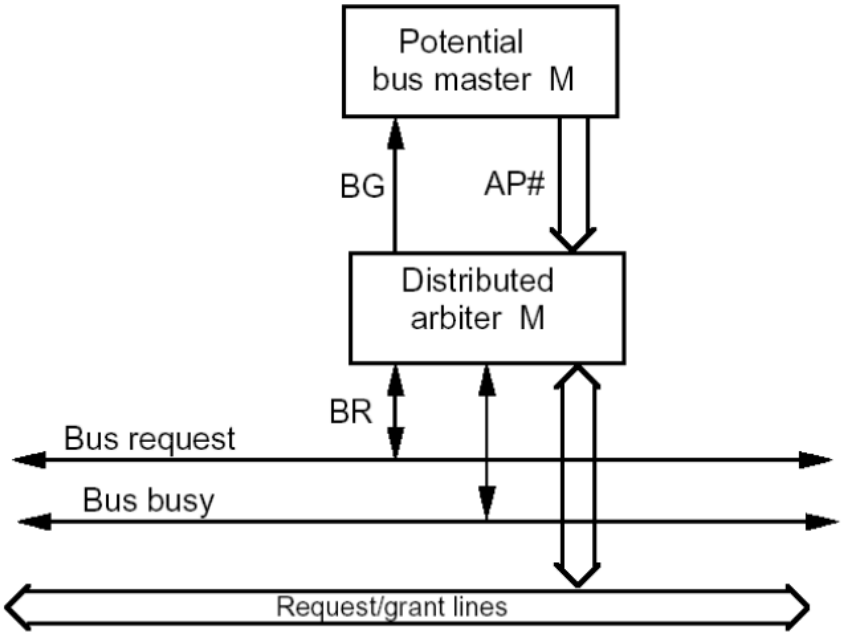
Bus Arbitration

- Distributed arbitration using self-detection:
  - Devices decide among themselves who gets the bus

- Arbitration priority numbers (AP#) determine order
  - Requester AP# is compared with that of other current requesters
  - Requests with lower AP# are removed
  - Remaining request is granted

- Distributed arbitration using collision-detection:
  - Any device can try to use the bus
  - Waits and tries again if a collision occurs

Bus Arbitration



- Starvation can occur with priority based arbitration
  - Very frequent high priority requests can lockout others

- Fairness can be imposed by centralized or distributed arbiter
  - Limit number of consecutive grants to high priority devices

Summary

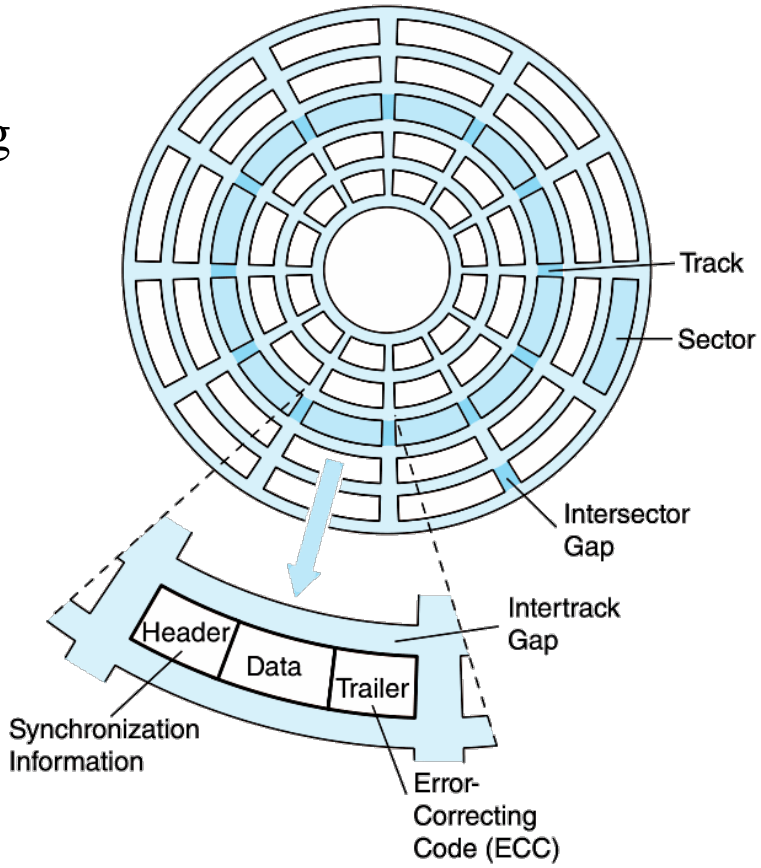
Option	Higher performance	Lower cost
Type	Separate address, data and control lines operate in parallel	Multiplexed lines carry address and data at different times
Transfer size	Block transfers have less overhead	each data unit incurs overhead for individual transfers
Bus masters	Multiple requires arbitration but increases flexibility	Single master requires no arbitration
Transaction type	Split transactions allow interleaving of transfers	Continuous connection may tie up lines while waiting
Width	More lines allow more data per transfer	Fewer lines reduce cost but limit amount of data in a transfer
Clocking	Synchronous is faster but may be limited by the slowest device	Asynchronous requires handshake signals (more overhead) but is not limited by the slowest device

DISK Systems

- Processor speeds far outpace the speed of disks
  - Instructions execute in nanoseconds
  - Disks still require milliseconds to access
- Parallel access can hide the slowness of disks
  - Separate blocks are read from different disks at the same time
- The seek and access delays are not reduced
  - The beginning of the data must still be determined
  - Multiple pieces of data are accessed in parallel
  - The multiple pieces are buffered and reassembled
  - The complete file can then be transferred into memory

DISK Systems

Disk tracks are numbered from the outside edge, starting with zero.



A sector is the smallest unit of transfer.

DISK Systems

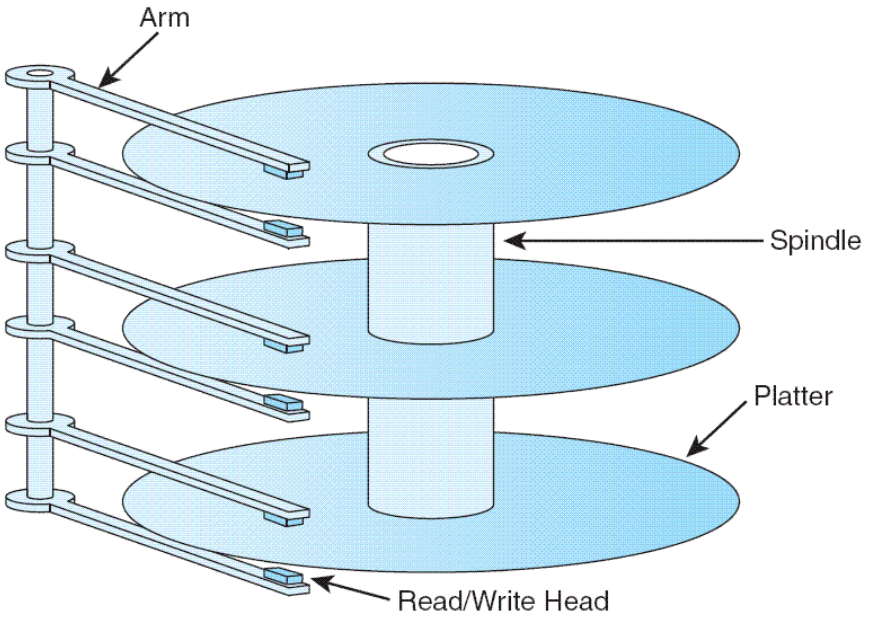
Hard disk platters are mounted on spindles.

Read/write heads are mounted on a comb that swings radially to read the disk.

The rotating disks form a logical cylinder beneath the read/write heads.

Data blocks are addressed by their cylinder, surface, and sector

A disk can perform one read or one write at a time.



DISK Systems

Seek time is the time that it takes for a disk arm to move into position over the desired cylinder.

Rotational delay is the time that it takes for the desired sector to move into position beneath the read/write head.

Seek time + rotational delay = *access time*.

The data transfer time is also a factor that must be considered.

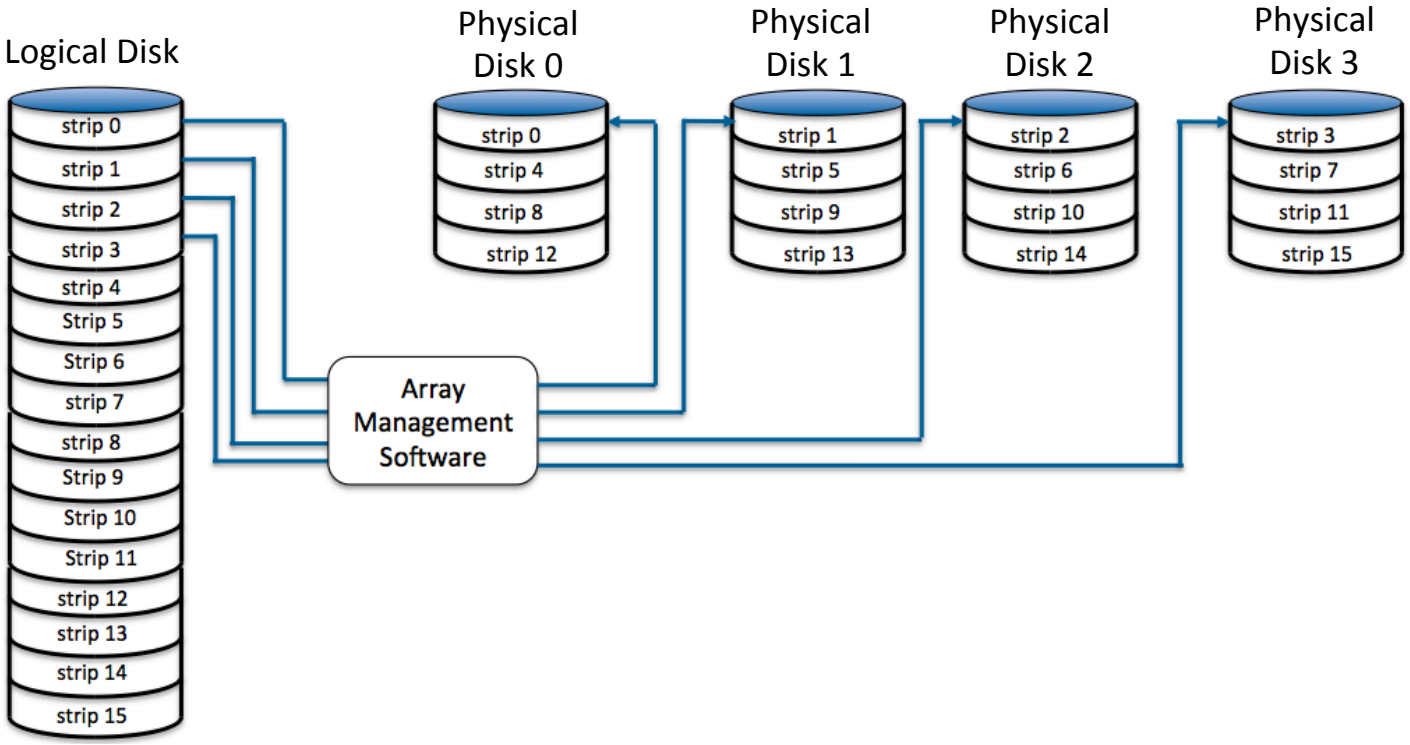
RAID Systems

- RAID is an acronym:
  - Redundant array of inexpensive disks
  - Most disks are now inexpensive
  - So “inexpensive” was replaced by “independent”
- Different level numbers do not imply ranking
  - They only distinguish one category from another
  - Originally, levels 0 through 5 were identified
- Not all levels duplicate data
- Not all levels provide redundancy

RAID Systems

- Data Striping can improve performance
  - Files are split into smaller pieces (blocks)
  - The pieces are stored on different disks
  - For reads, all disks access their portions of the data
- Multiple categories or levels have been defined
  - Files are split into separate parts
  - The separate parts are on different disks
  - Multiple disks can be accessed at the same time
- Using multiple disks can yield greater reliability
  - Duplicate copies of data can be stored
  - Error correcting information can be included
  - Contents of failing disks can be reconstructed

RAID 0

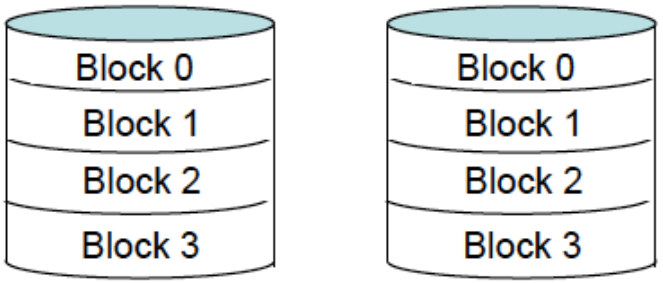


RAID 0 provides no redundancy (low reliability)

Speeds access to large amounts of logically contiguous data

Supports multiple transactions that map to separate strips or blocks

RAID1



Redundancy provided by duplicating each data disk

- Each data disk has a mirror image or shadow disk

If a disk fails, its mirror image is used

- copied to replacement disk

Disk spindles are not synchronized

Reads obtain data from first available copy

Writes must update both mirror images

Main disadvantage is the cost of fully duplicating the data

Other RAID Levels

Levels 2 through 5 do not fully duplicate data

- Redundancy is provided by:
  - Error correcting information or
  - Parity

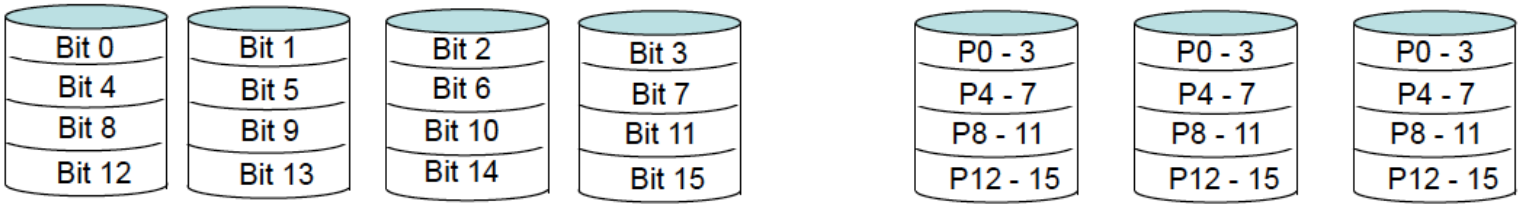
Levels 2 and 3 employ parallel access

- All disks in the array are accessed at the same time
- Heads on each disk move in unison
- Heads on each disk are at the same relative position

Levels 4 and 5 use independent access

- They differ in how the parity is distributed
- Spindles are not synchronized

RAID2



Each access involves all disks

- Parity disks must be written along with data

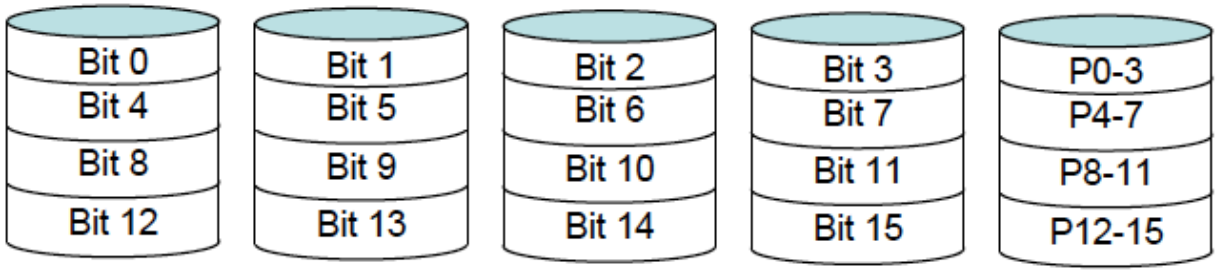
Employs Hamming code for error correction

- Uses almost as much overhead as level 1
- # of parity disks is function of data disks
- For example: 3 error disks for 4 data disks

Nibble, byte or bit level striping

Not used commercially

RAID3

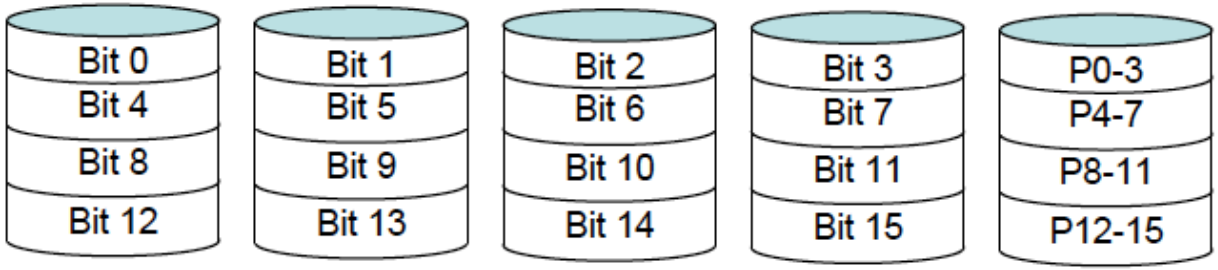


Uses a single parity drive

- Parity based on simple XOR function
- Parity disks must be written along with data
- Parity drive creates a bottleneck
- Parity only used to reconstruct data
- Failures often involve just one disk

All strips within the same “row” constitute a stripe

RAID3 Example



Suppose drive containing bit3 fails:

P0-3 = bit0 ^ bit1 ^ bit2 ^ bit3 (^ denotes XOR operator)

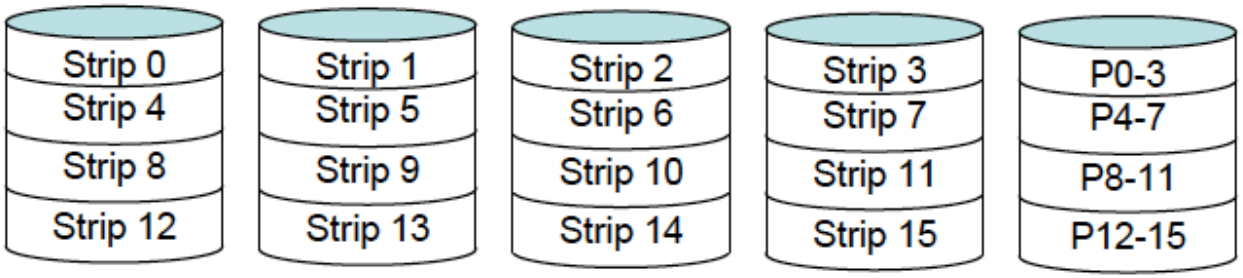
bit3 ^ P0-3 ^ P0-3 = bit0 ^ bit1 ^ bit2 ^ bit3 ^ bit3 ^ P0-3

bit3 = bit0 ^ bit1 ^ bit2 ^ P0-3

So bit3 can be reconstructed from the parity and remaining data disks (entire contents of failed disk can be reconstructed this way)

Writes or reads map to one stripe at a time

RAID4

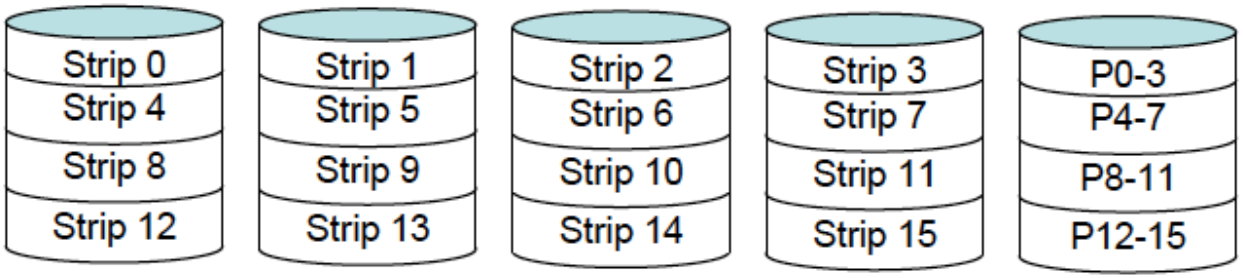


RAID4 uses block level striping (unlike RAID3 strips)
Access heads move independently (unlike RAID 3)

Example: strip0 contains bits 0 – 1023
strip1 contains bits 1024 – 2047, etc.

First bit of P0-3 = XOR of bits 0, 1024, 2048, etc.
(bitwise XOR of data strips)

RAID4



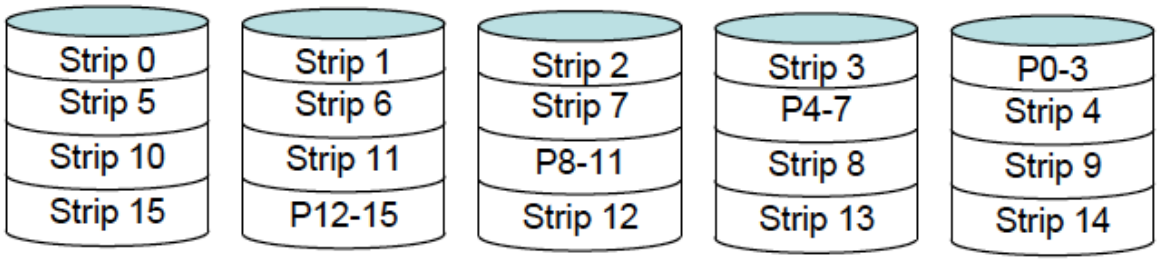
Parity disk must be written if any of the data strips are written

With parity on only one disk, the parity disk is a bottleneck

Tolerates at most one failing disk at a time

If a second disk fails before a replacement occurs, the system is inoperable

RAID5



Uses block-interleaved distributed parity

Different stripes can be updated in parallel (no bottleneck)

- Parity and data blocks are on separate drives

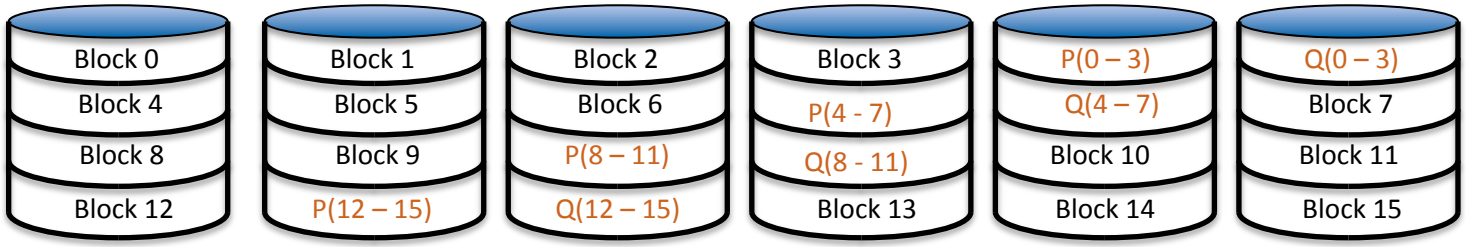
Most popular system

- good reliability
- good performance

RAID5 can only tolerate one failing disk at a time

Failed disk must be replaced before a second disk fails

RAID6



Uses dual-redundancy

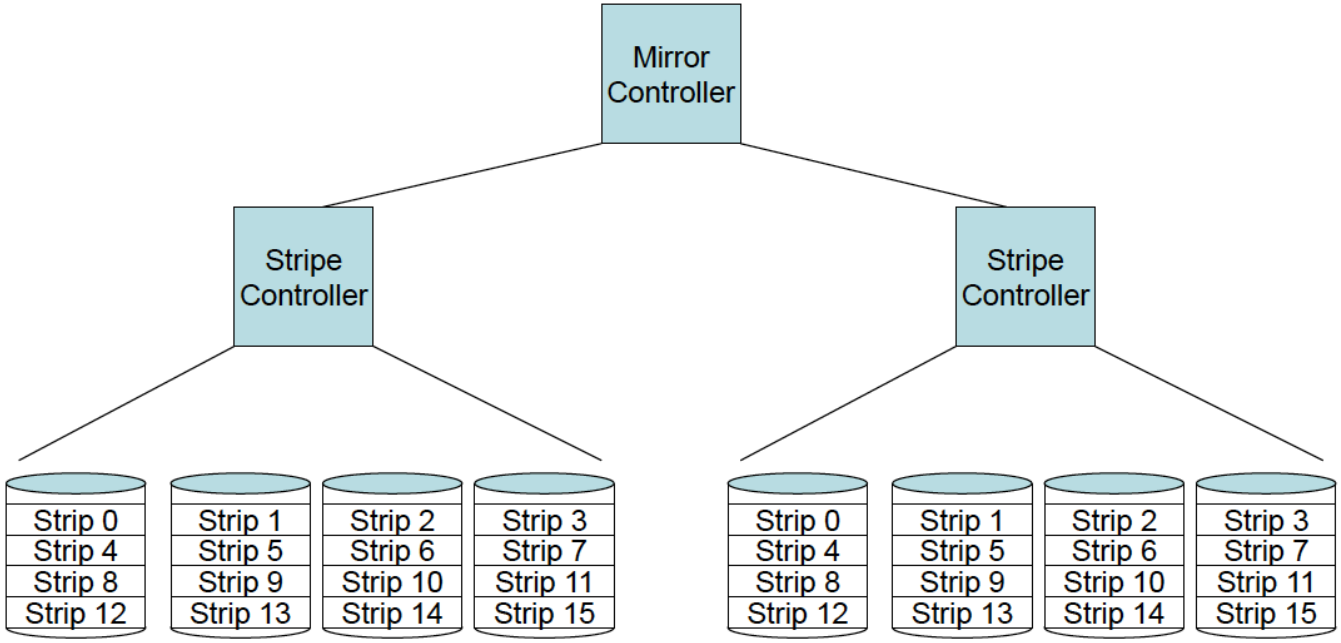
- P is the same XOR function used in RAID5 and RAID4
- Q is based on a different error check scheme
  - Such as Reed-Solomon code
- missing data can be reconstructed from either parity

Tolerates up to two failing disks

Access heads on separate disks operate independently

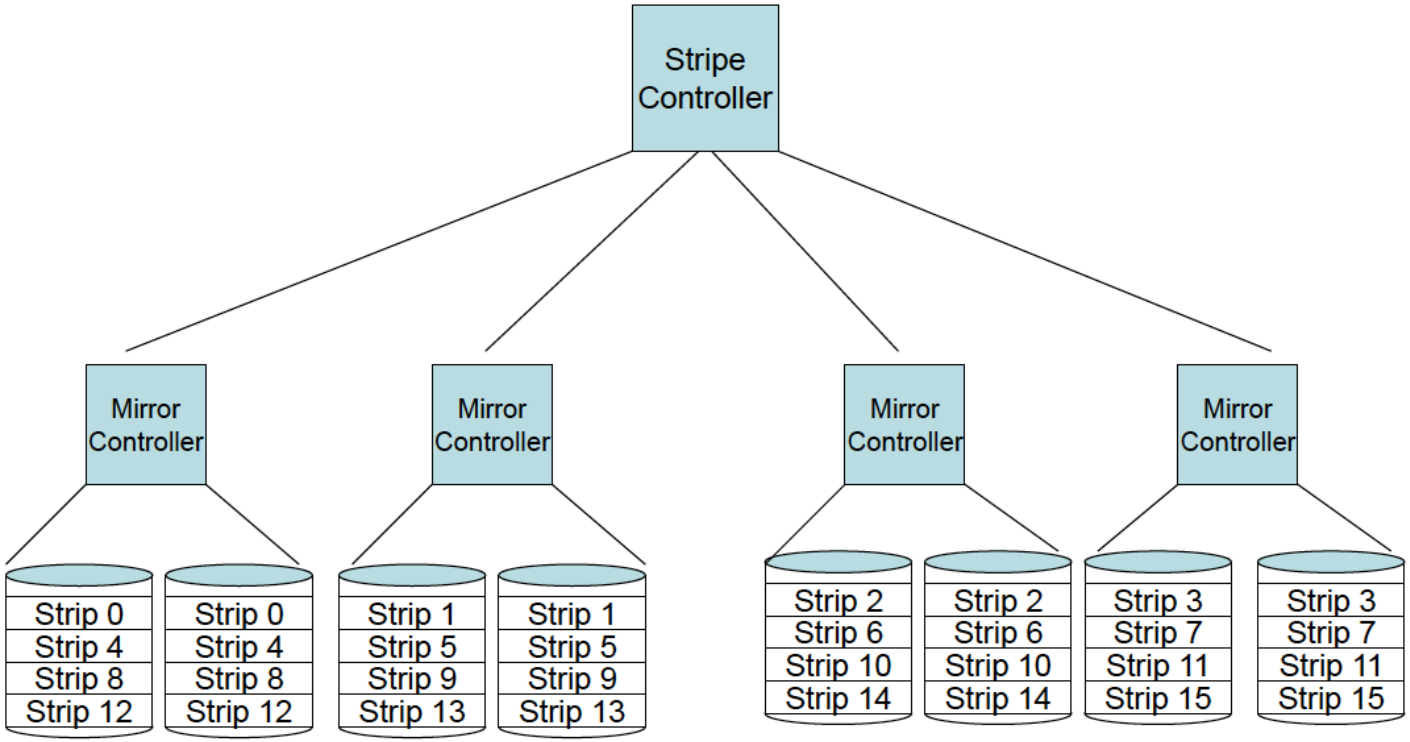
I/O requests to separate disks can be handled in parallel

RAID01



More recent systems combine levels: e.g. Mirrored RAID0 systems (01)
Each stripe controller makes its 4 disks appear as a single disk to mirror controller
mirror controller makes the mirrored pair appear as a single disk to the I/O system

RAID10

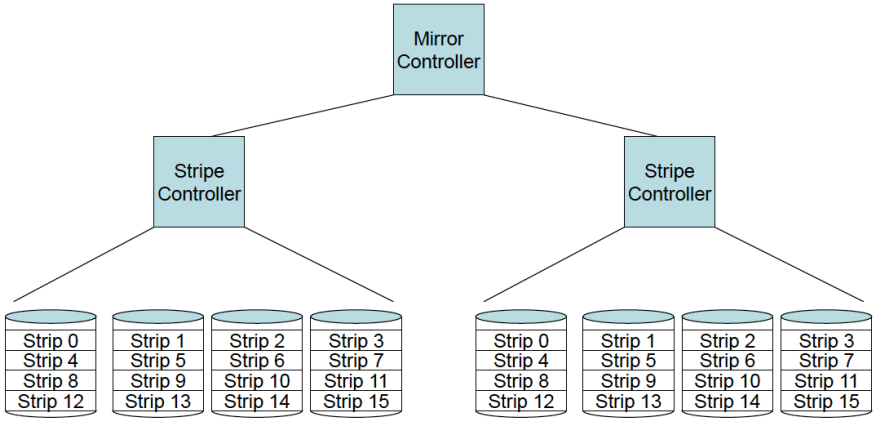


Stripped Mirrored RAID0 systems (01)

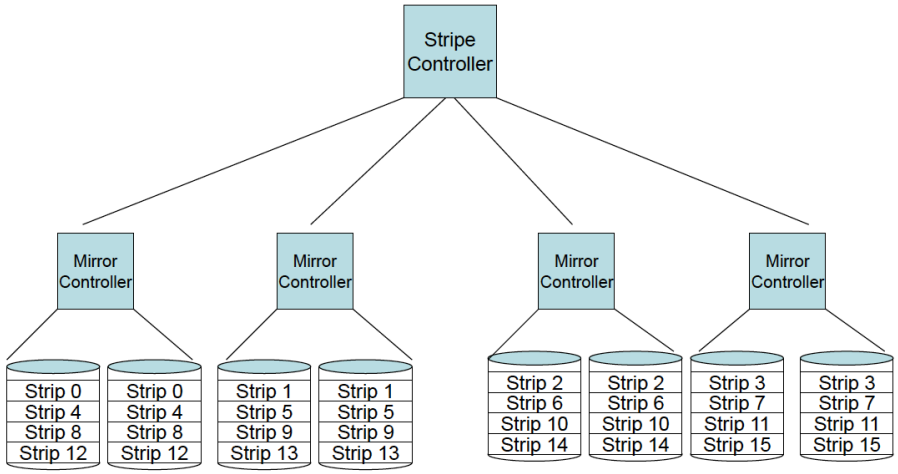
Each mirror controller makes its 2 disks appear as a single disk to the stripe controller

Stripe controller makes the 4 striped disks appear as a single disk to the I/O system

RAID10 versus RAID01



RAID 01

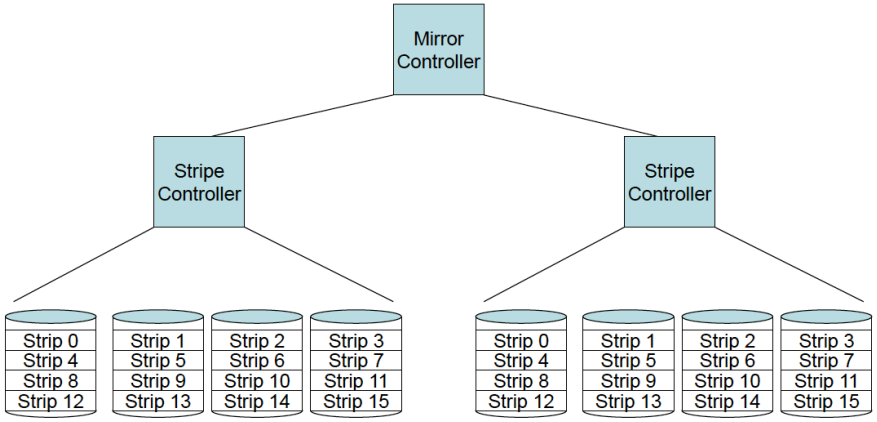


RAID 10

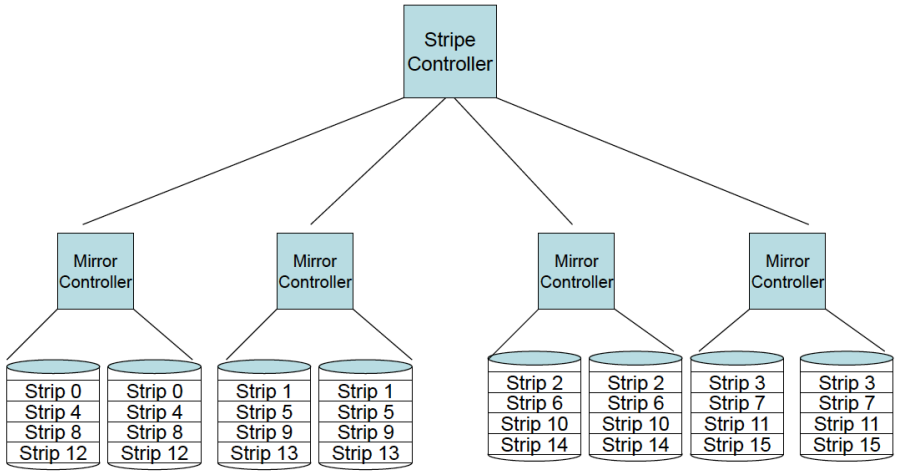
Expense:

- 3 controllers are needed for the level 01 system
- 5 controllers are needed for the level 10 system

RAID10 versus RAID01



RAID 01

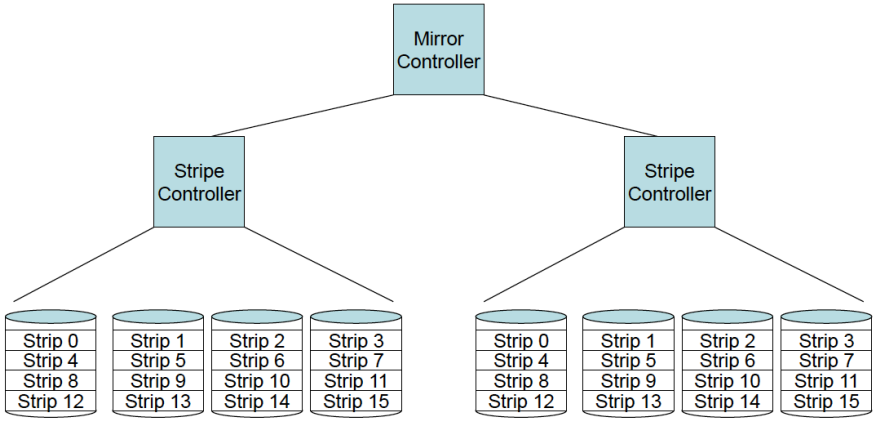


RAID 10

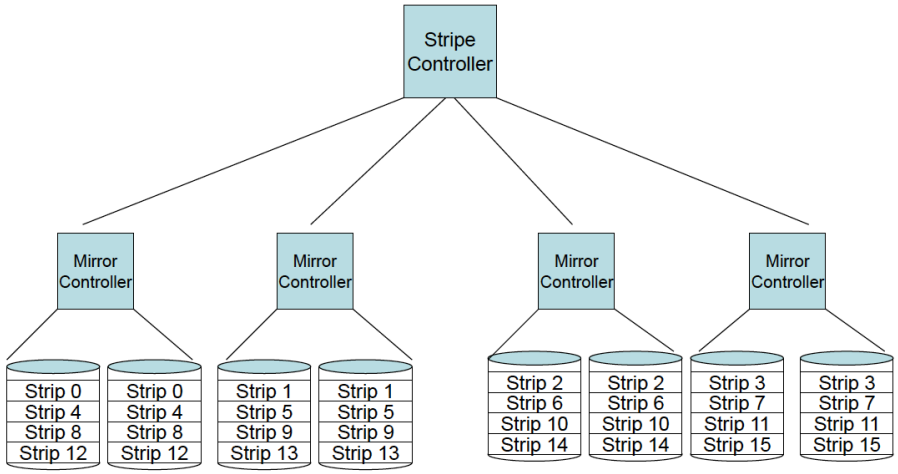
Expense:

- 3 controllers are needed for the level 01 system
- 5 controllers are needed for the level 10 system

RAID10 versus RAID01



RAID 01



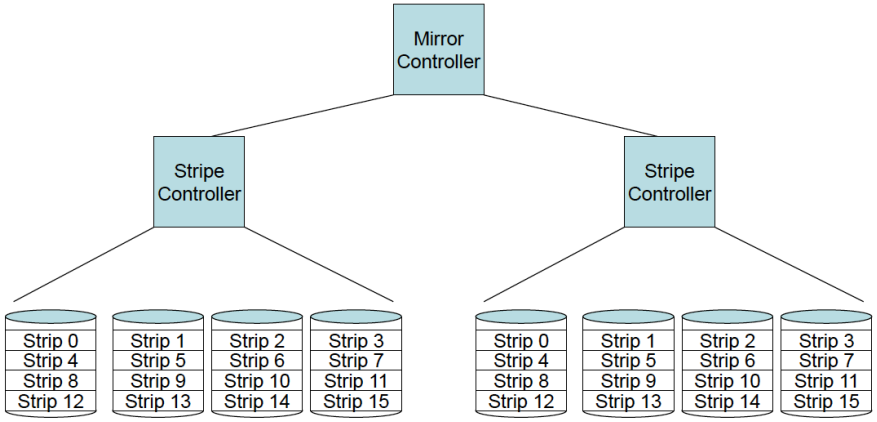
RAID 10

Reliability:

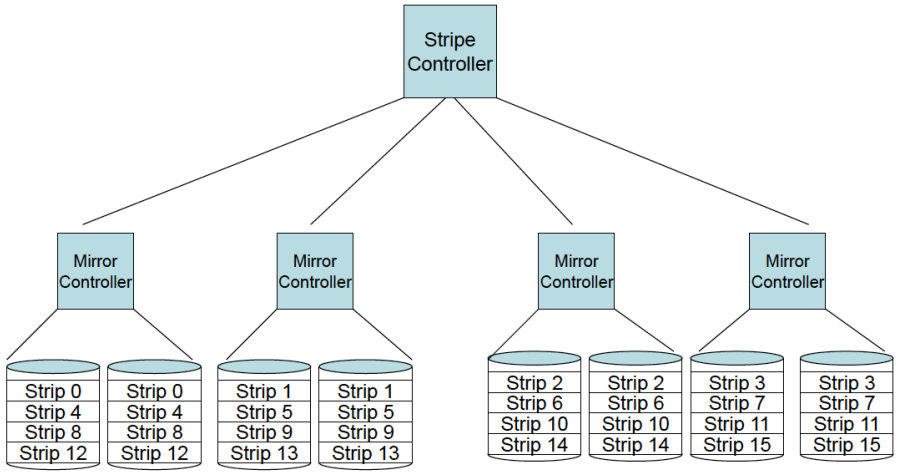
RAID01 is inoperable if a disk fails in each striped group even if the two disks are not mirror images

Both mirror images must fail for RAID10 to stop operating

RAID10 versus RAID01



RAID 01

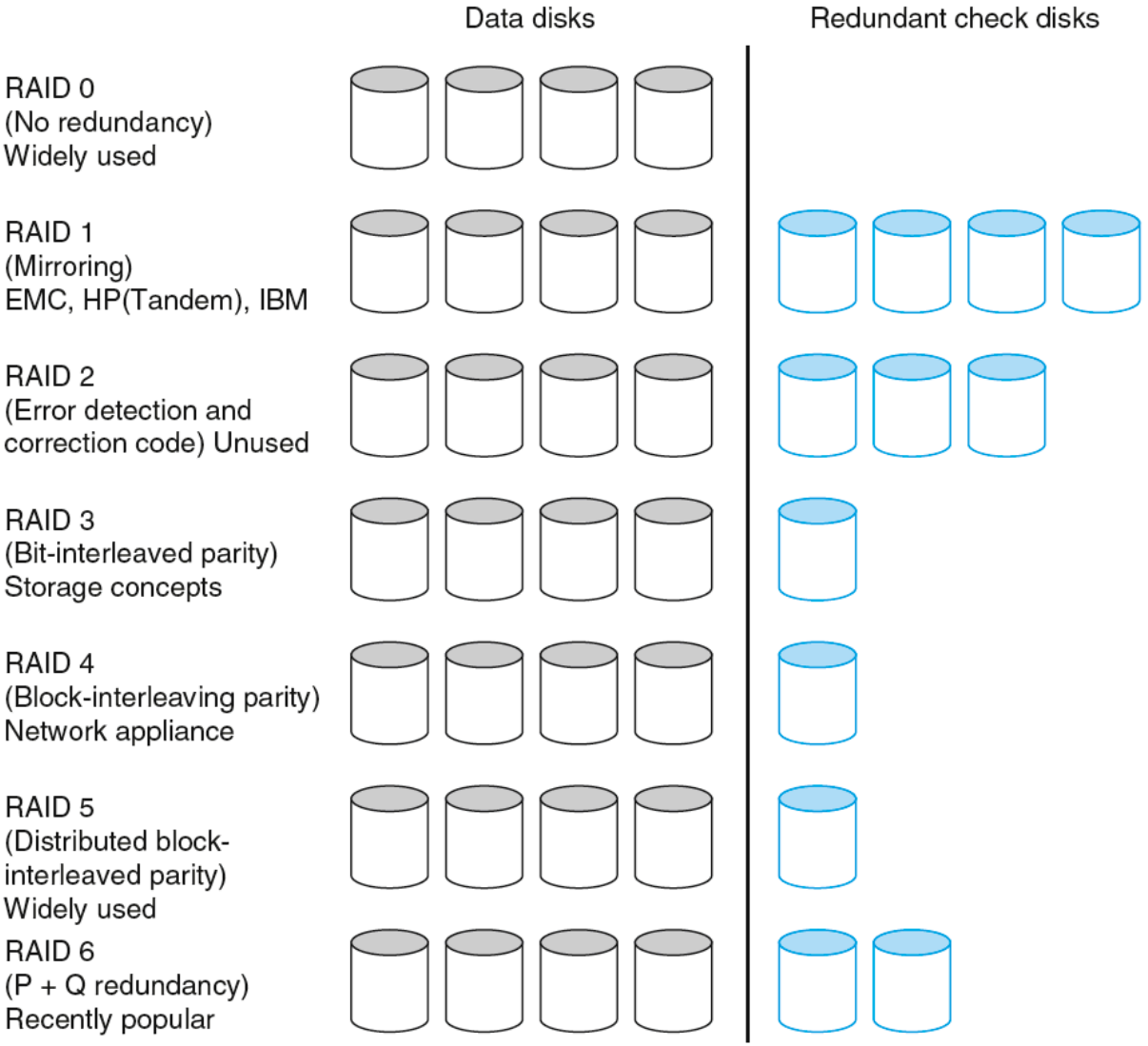


RAID 10

Time required for restoration

- All disks within a group would be re-written for RAID01
- Only mirror image of failed disk would be re-written for RAID10

Summary


Data disks
Redundant check disks
RAID 0
(No redundancy)
Widely used
RAID 1
(Mirroring)
EMC, HP(Tandem), IBM
RAID 2
(Error detection and
correction code) Unused
RAID 3
(Bit-interleaved parity)
Storage concepts
RAID 4
(Block-interleaving parity)
Network appliance
RAID 5
(Distributed block-
interleaved parity)
Widely used
RAID 6
(P + Q redundancy)
Recently popular

Solid State Disk Drives

- Solid State Drives (SSD) employ Flash Memory
  - They mimic the electrical interfaces of hard drives
  - SSD's can plug into hard disk sockets
  - They provide higher performance,
    - lower weight & less power consumption
    - Greater tolerance to shock
  - They are more costly and have a lower storage capacity

Solid State Disk Drives

- SSD's have no moving parts as with HDDs
  - They are truly random access
  - No rotational latency or seek delay
  - Response times are shorter
  - Yield higher input/output operations per second (IOPS)
  - Generate less heat
  - Provide quite operation
  - Cost more per gigabyte of storage than HDDs
  - Fragmentation is not a problem as with hard drives

SSD Features

- Hard disks have effectively equal read, write and erase times
- SSD's use different read and write mechanisms
  - Writes take longer than reads
  - Write performance can be up to 50% lower than read performance
  - Blocks cannot simply be overwritten
    - they must first be erased and then written
- There is a limit to how may write cycles can be performed
- The cells fail after tens of thousands of write cycles
- Repeated write cycles may cause cells to become stuck at 0
- Wear leveling techniques are distribute the writes

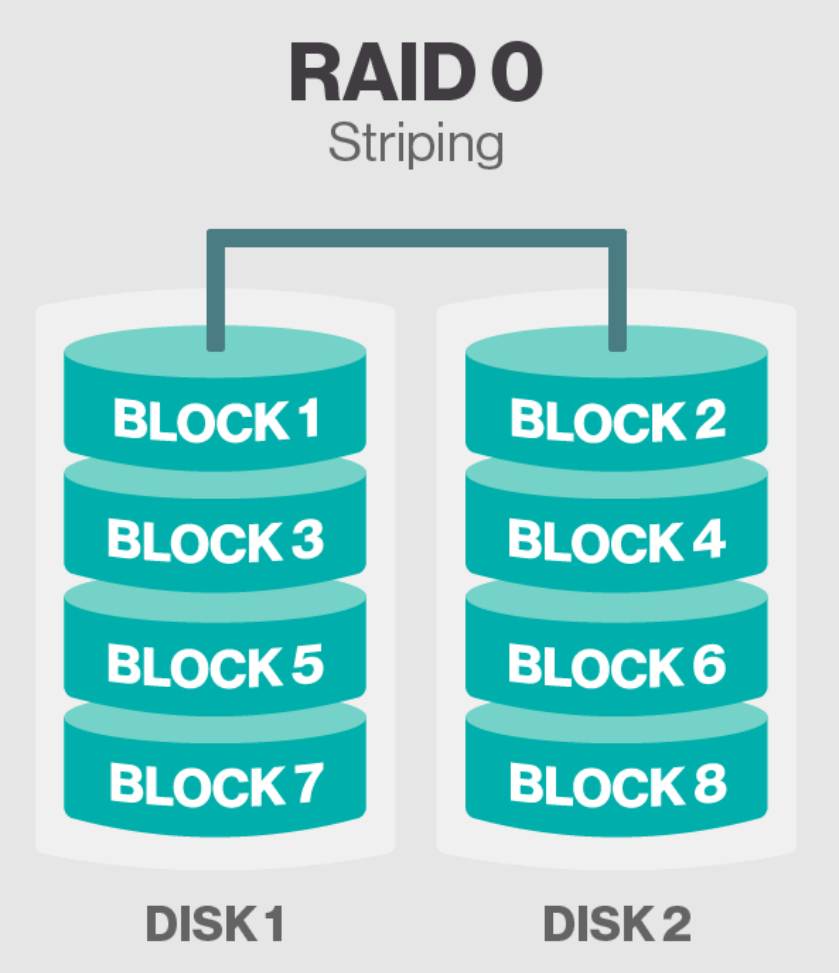
SSD Features

- Wear leveling
- Requested blocks are mapped onto physical block addresses
- Controller monitors how often physical blocks are used
- Adjusts mapping table to ensure all blocks share the load
- This is required when SSD's are used as secondary storage

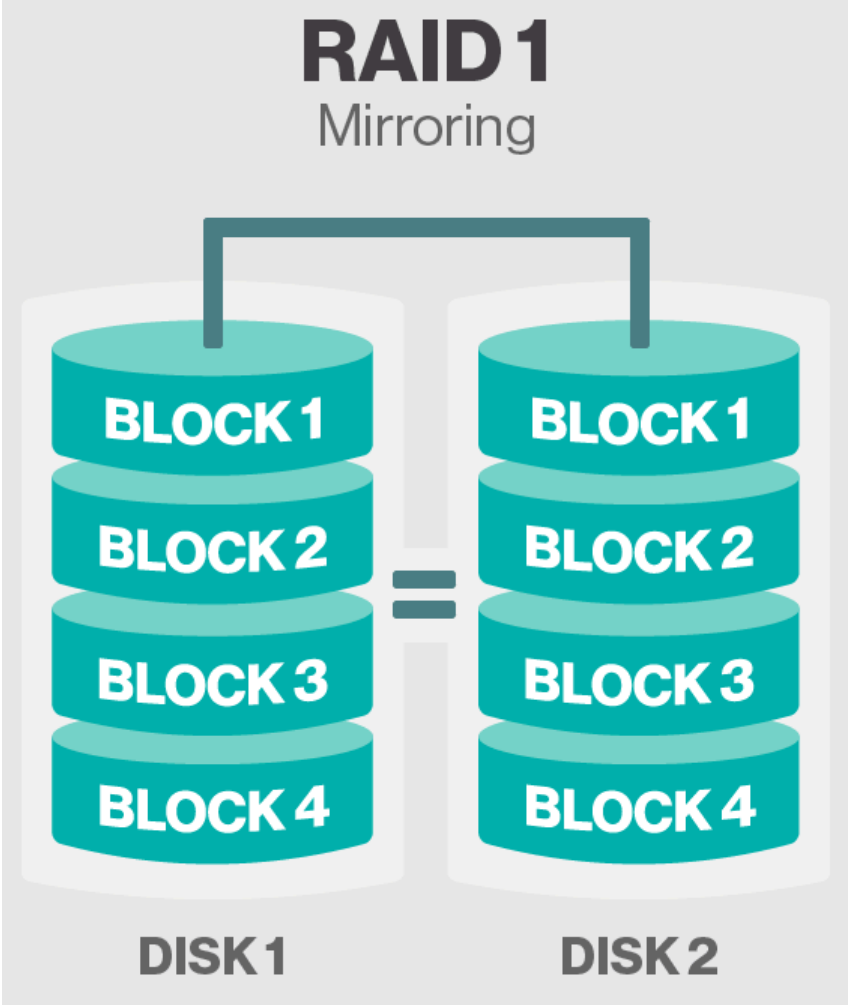
- Wear leveling is less of an issue for other applications
- BIOS
- Digital cameras
- Music or video players

SSD Features

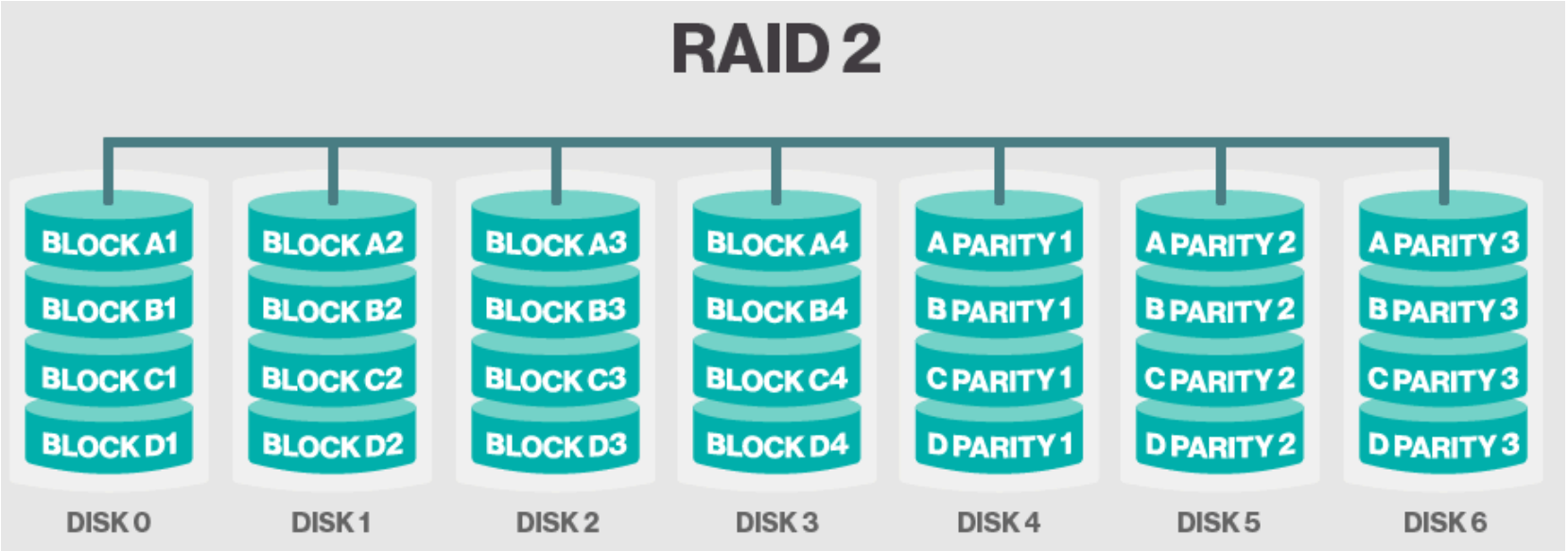
- Management of free space
  - Unused blocks that are erased can be used immediately
  - Unused blocks containing unneeded data must first be erased
- Entire blocks are erased (e.g. 4KB)
- SSD use is expanding with the declining cost of flash memory
- Their greater reliability means less frequent replacement



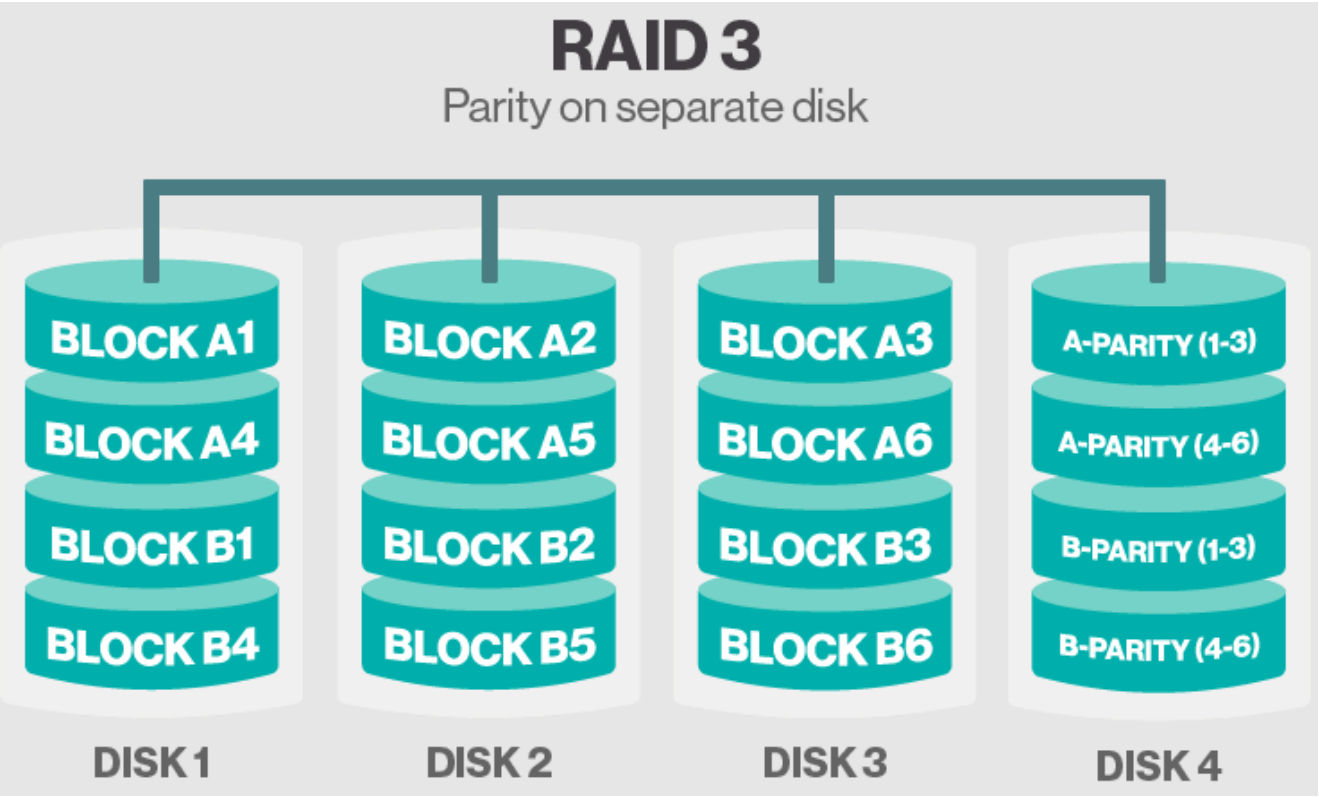
RAID 0: This configuration has striping but no redundancy of data. It offers the best performance but no fault-tolerance.



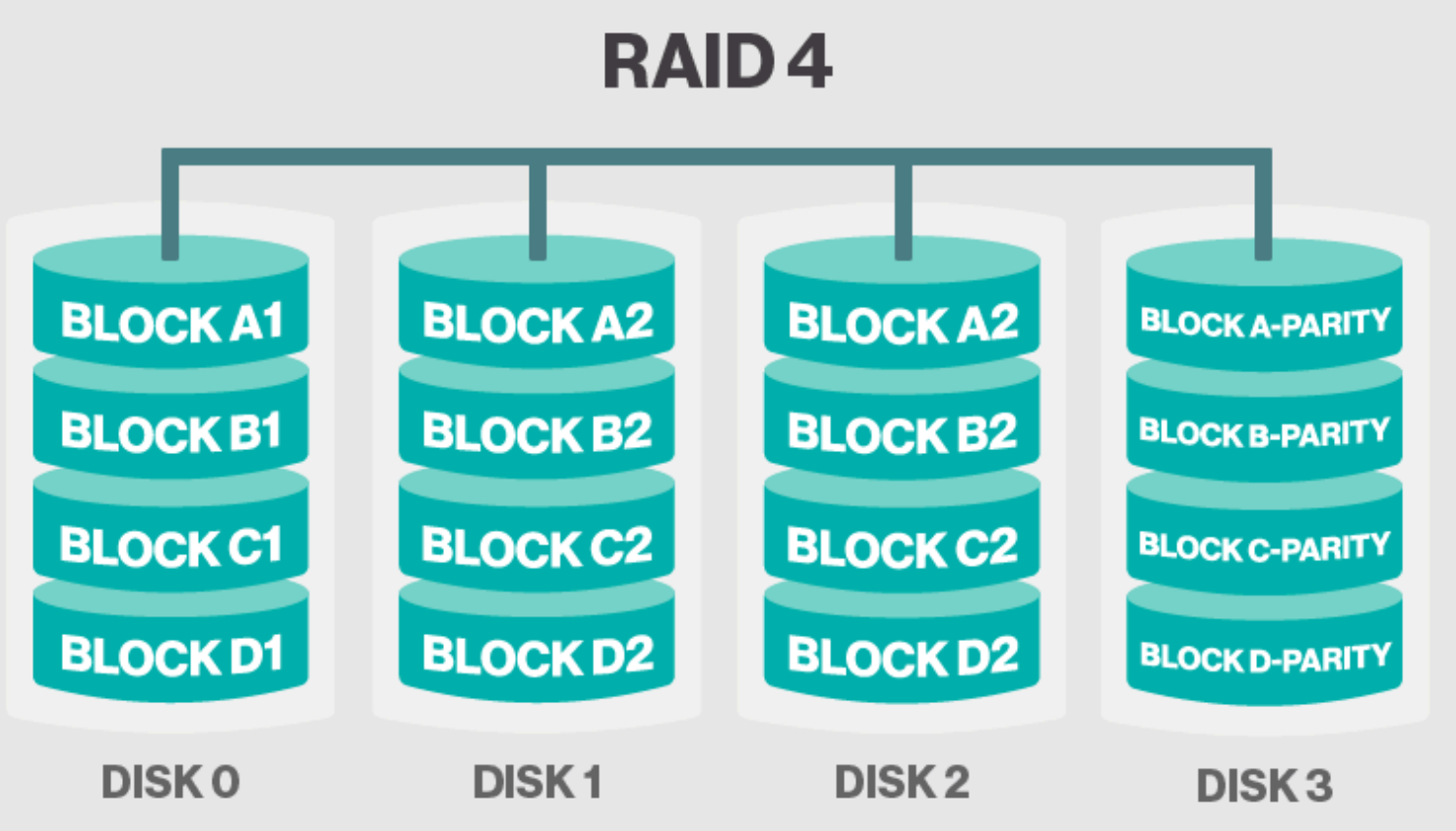
RAID 1: Also known as *disk mirroring*, this configuration consists of at least two drives that duplicate the storage of data. There is no striping. Read performance is improved since either disk can be read at the same time. Write performance is the same as for single disk storage.



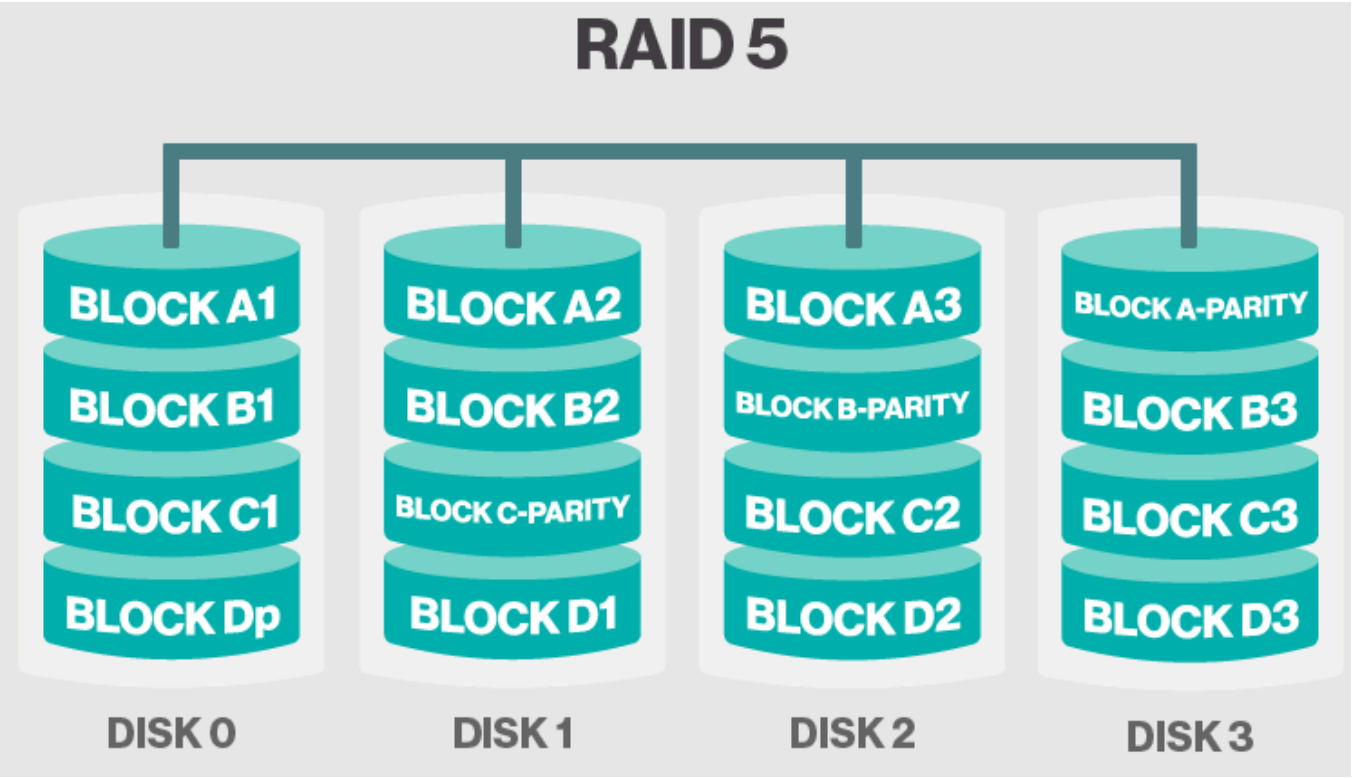
RAID 2: This configuration uses striping across disks with some disks storing error checking and correcting (ECC) information. It has no advantage over RAID 3 and is no longer used.



RAID 3: This technique uses striping and dedicates one drive to storing parity information. The embedded ECC information is used to detect errors. Data recovery is accomplished by calculating the exclusive OR (XOR) of the information recorded on the other drives. Since an I/O operation addresses all drives at the same time, RAID 3 cannot overlap I/O. For this reason, RAID 3 is best for single-user systems with long record applications.

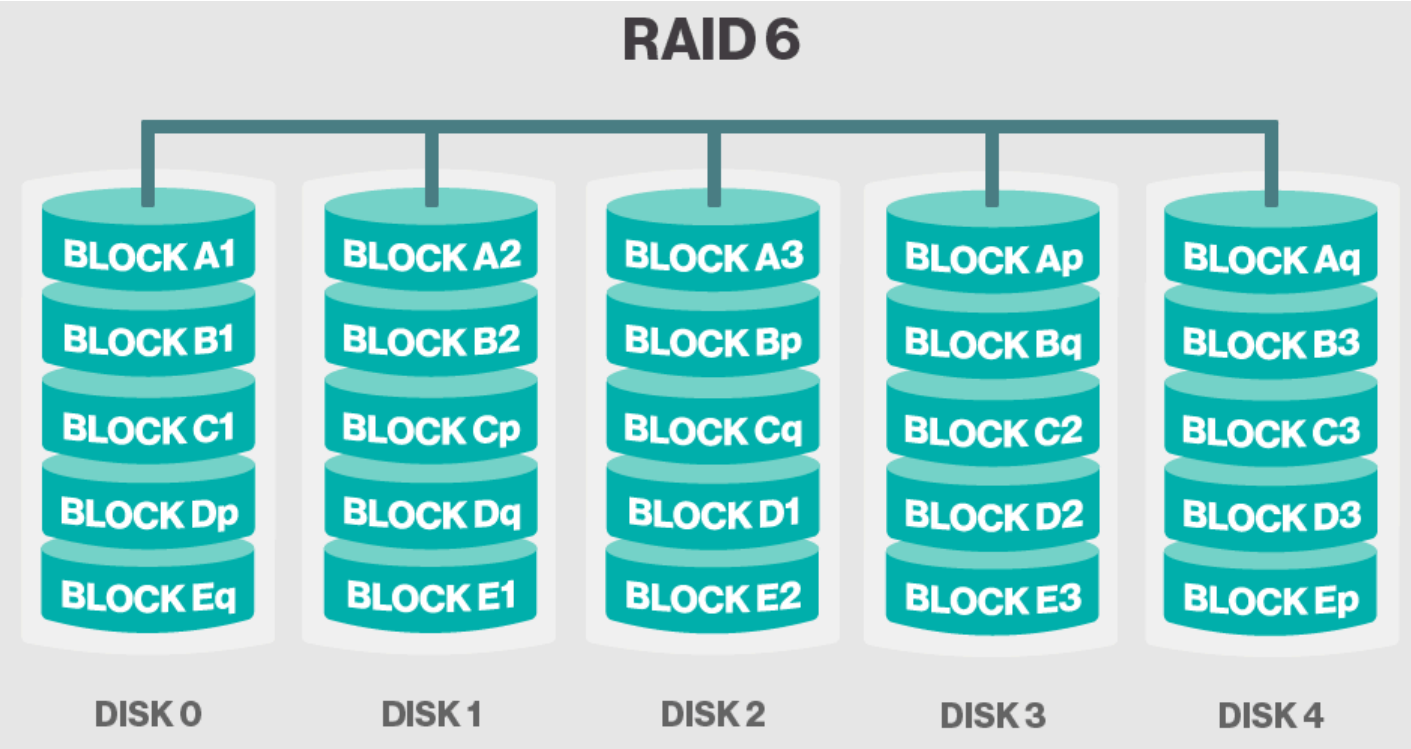


RAID 4: This level uses large stripes, which means you can read records from any single drive. This allows you to use overlapped I/O for read operations. Since all write operations have to update the parity drive, no I/O overlapping is possible. RAID 4 offers no advantage over RAID 5.



RAID 5: This level is based on block-level striping with parity. The parity information is striped across each drive, allowing the array to function even if one drive were to fail. The array's architecture allows read and write operations to span multiple drives. This results in performance that is usually better than that of a single drive, but not as high as that of a RAID 0 array. RAID 5 requires at least three disks, but it is often recommended to use at least five disks for performance reasons.

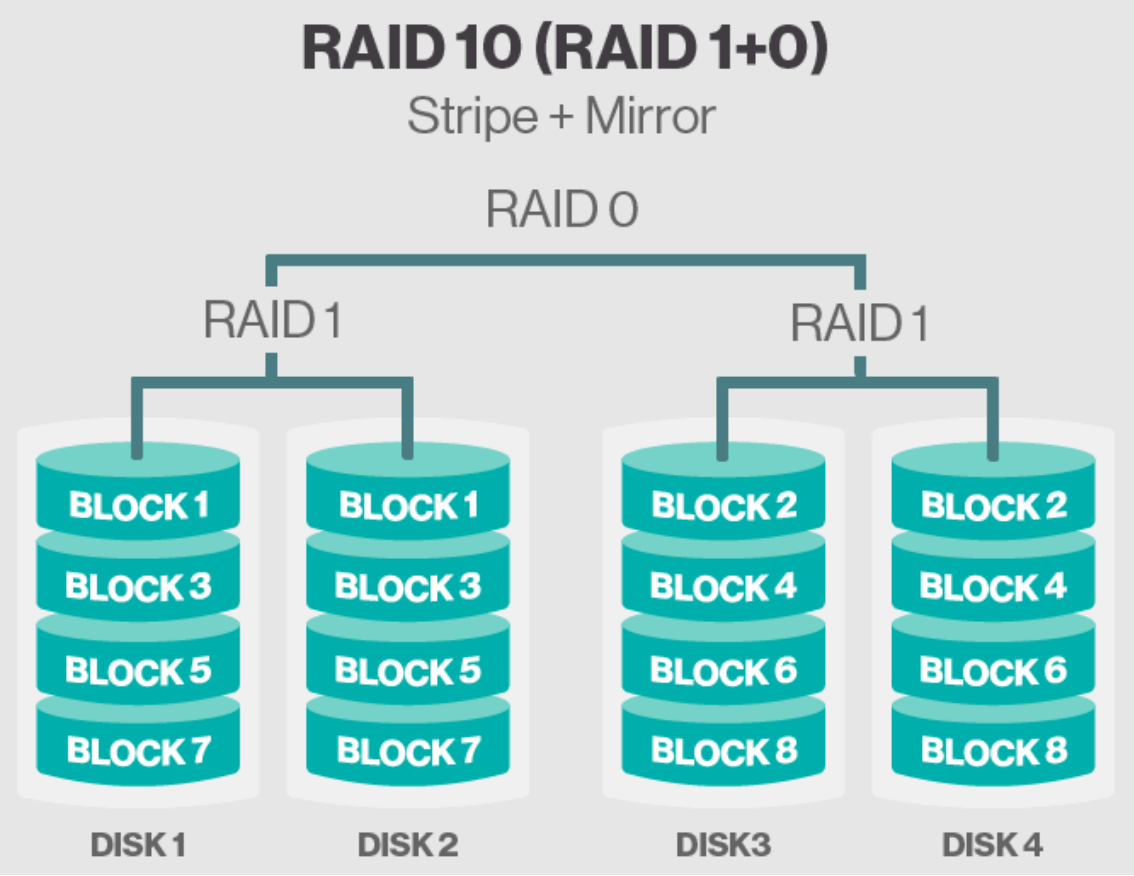
RAID 5 arrays are generally considered to be a poor choice for use on write-intensive systems because of the performance impact associated with writing parity information. When a disk does fail, it can take a long time to rebuild a RAID 5 array. Performance is usually degraded during the rebuild time and the array is vulnerable to an additional disk failure until the rebuild is complete.



RAID 6: This technique is similar to RAID 5 but includes a second parity scheme that is distributed across the drives in the array. The use of additional parity allows the array to continue to function even if two disks fail simultaneously. However, this extra protection comes at a cost. RAID 6 arrays have a higher cost per gigabyte (GB) and often have slower write performance than RAID 5 arrays.

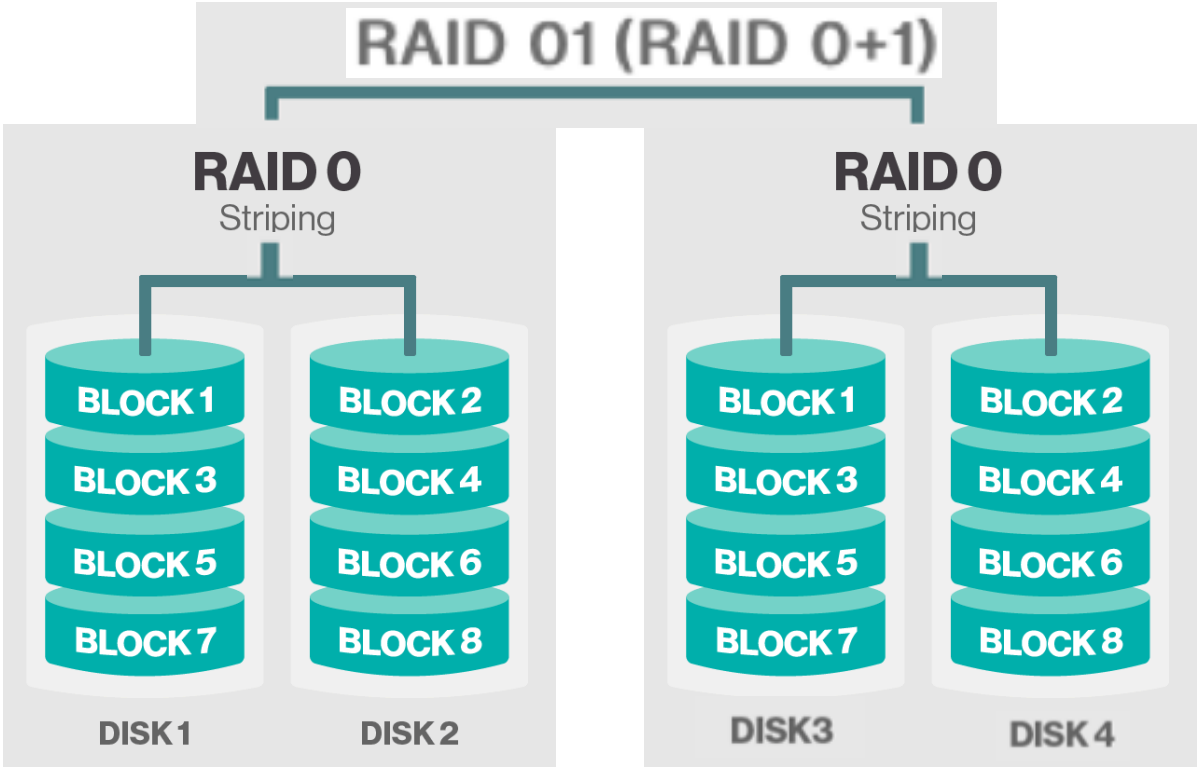
Nested RAID levels

Some RAID levels are referred to as *nested RAID* because they are based on a combination of RAID levels.



RAID 10 **(RAID 1+0):** Combining RAID 1 and RAID 0, this level is often referred to as RAID 10, which offers higher performance than RAID 1 but at a much higher cost. In RAID 1+0, the data is mirrored and the mirrors are striped.

Nested RAID levels



RAID 01 (RAID 0+1): RAID 0+1 is very similar to RAID 1+0, except the data organization method is slightly different. Rather than creating a mirror and then stripping the mirror, RAID 0+1 creates a stripe set and then mirrors the stripe set.