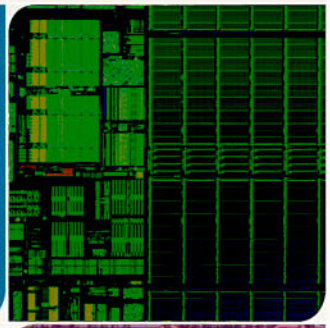
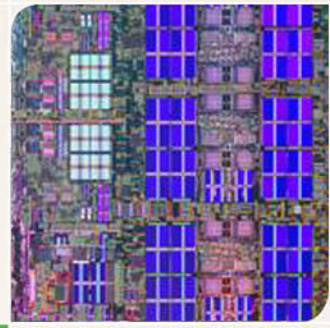
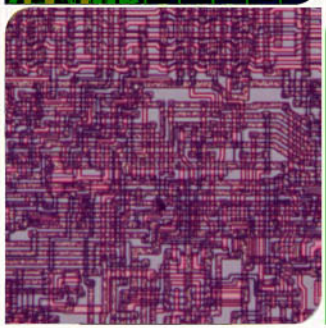
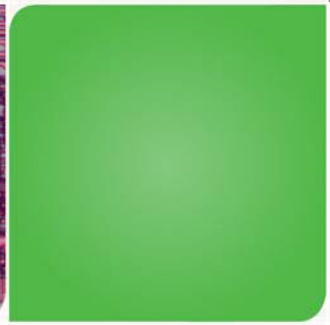
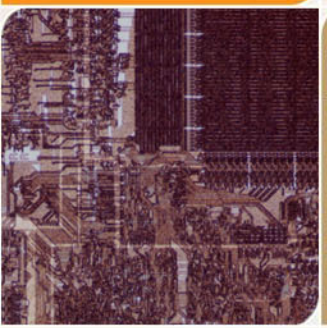



CS4290/CS6290

Fall 2011

Prof. Hyesoon Kim

Georgia Tech College of Computing

Thanks to Prof. Loh & Prof. Prvulovic



Storage Systems

- I/O performance (bandwidth, latency)
  - Bandwidth improving, but not as fast as CPU
  - Latency improving very slowly
  - Consequently, by Amdahl's Law:
    fraction of time spent on I/O increasing
- Other factors just as important
  - Reliability, Availability, Dependability
- Storage devices very diverse
  - Magnetic disks, tapes, CDs, DVDs, flash
  - Different advantages/disadvantages and uses

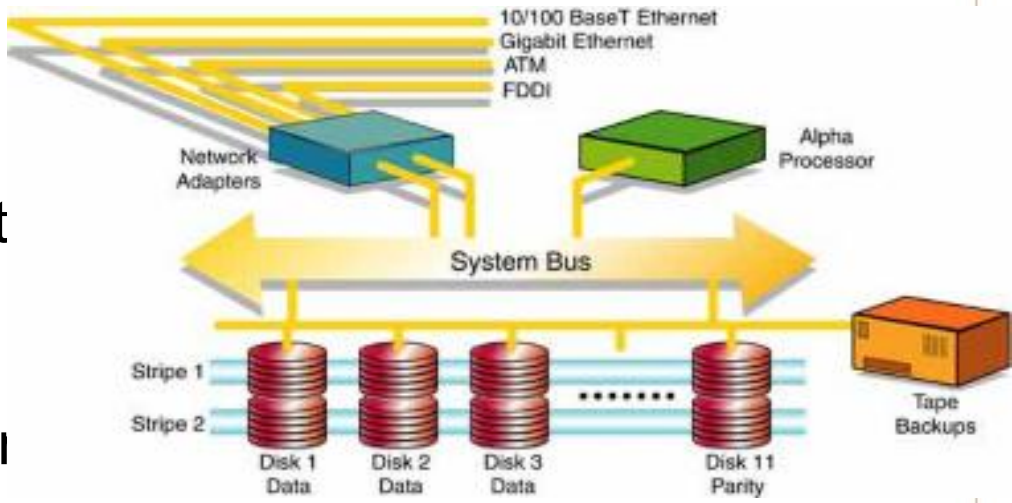


Buses for I/O

- Traditionally, two kinds of busses
  - CPU-Memory bus (fast, short)
  - I/O bus (can be slower and longer)
- Now: mezzanine buses (PCI)
  - Pretty fast and relatively short
  - Can connect fast devices directly
  - Can connect to longer, slower I/O busses
- Data transfers over a bus: transactions

Connection to CPU

- Devices connect to ports or **system bus**:
  - Allows devices to communicate w/CPU
  - Typically shared by mult devices
- Two ways of communicati with CPU:
  - **Ports** (programmed I/O)
  - **Direct-Memory Access** (DMA)





Ports (a.k.a. Programmed I/O)

- **Device port** – 4 registers
  - **Status** indicates device busy, data ready, error
  - **Control** indicates command to perform
  - **Data-in** read by host to get input from device
  - **Data-out** written by CPU to device
- **Controller** receives commands from bus, translates into actions, reads/writes data onto bus



Polling

- CPU:
  - Busy-wait until status = idle
  - Set command register, data-out (output)
  - Set status = command-ready
    - Controller: set status = busy
- Controller:
  - Reads command register, performs command
  - Places data in data-in (input)
  - Change state to idle or error



Interrupts

- Avoids busy waiting
- Device interrupts CPU when I/O operation completes
- On interrupt:
  - Determine which device caused interrupt
  - If last command to device was input operation, retrieve data from register
  - Initiate next operation for device



DMA

- Ports ("programmed I/O")
  - Fine for small amounts of data, low-speed
  - Too expensive for large data transfers!

- Solution: **Direct Memory Access (DMA)**
  - Allows devices to transfer data w/o subjecting the CPU to a heavy overhead
  - CPU initiate the transaction (send command)
  - DMA interrupts CPU when entire transfer complete
  - DMA can lead to a cache coherence problem



Bus Design Decisions

- Split transactions
  - Traditionally, bus stays occupied between request and response on a read
  - Now, get bus, send request, free bus (when response ready, get bus, send response, free bus)
- Bus mastering
  - Which devices can initiate transfers on the bus
  - CPU can always be the master
  - But we can also allow other devices to be masters
  - With multiple masters, need arbitration



CPU-Device Interface

- Devices typically accessible to CPU through control and data registers
- These registers can be either
  - Memory mapped
    - Some physical memory addresses actually map to I/O device registers
    - Read/write through LS/ST
    - Most RISC processors support only this kind of I/O mapping
  - Be in a separate I/O address space
    - Read/write through special IN/OUT instrs
    - Used in x86, but even in x86 PCs some I/O is memory mapped



Dependability

- Quality of delivered service that justifies us relying on the system to provide that service
  - Delivered service is the *actual behavior*
  - Each module has an ideal *specified behavior*
- Faults, Errors, Failures
  - Failure: actual deviates from specified behavior
  - Error: defect that results in failure
  - Fault: cause of error



Failure Example

- A programming mistake is a fault
  - An add function that works fine, except when we try 5+3, in which case it returns 7 instead of 8
  - It is a *latent error* until activated
- An activated fault becomes *effective error*
  - We call our add and it returns 7 for 5+3
- Failure when error results in deviation in behavior
  - E.g. we schedule a meeting for the 7^{th} instead of 8^{th}
  - An effective error need not result in a failure (if we never use the result of this add, no failure)

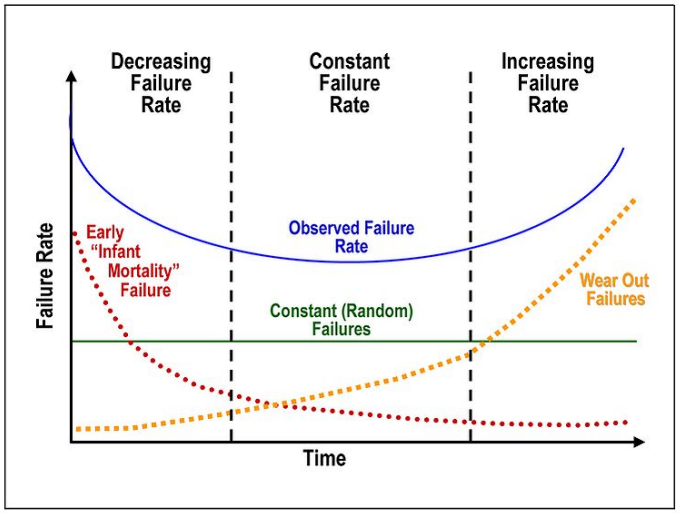


Reliability and Availability

- System can be in one of two states
  - Service Accomplishment
  - Service Interruption
- Reliability
  - Measure of continuous service accomplishment
  - Typically, Mean Time To Failure (MTTF)
- Availability
  - Service accomplishment as a fraction of overall time
  - Also looks at Mean Time To Repair (MTTR)
    - MTTR is the average duration of service interruption
  - Availability=MTTF/(MTTF+MTTR)

Faults Classified by Cause

- Hardware Faults
  - Hardware devices fail to perform as designed
- Design Faults
  - Faults in software and some faults in HW
  - E.g. the Pentium FDIV bug was a design fault
- Operation Faults
  - Operator and user mistakes
- Environmental Faults
  - Fire, power failure, sabotage, etc.



Faults Classified by Duration

- Transient Faults
  - Last for a limited time and are not recurring
  - An alpha particle can flip a bit in memory but usually does not damage the memory HW
- Intermittent Faults
  - Last for a limited time but are recurring
  - E.g. overclocked system works fine for a while, but then crashes… then we reboot it and it does it again
- Permanent Faults
  - Do not get corrected when time passes
  - E.g. the processor has a large round hole in it because we wanted to see what's inside…



Improving Reliability

- Fault Avoidance
  - Prevent occurrence of faults by construction
- Fault Tolerance
  - Prevent faults from becoming failures
  - Typically done through redundancy
- Error Removal
  - Removing latent errors by verification
- Error Forecasting
  - Estimate presence, creation, and consequences of errors



Disk Fault Tolerance with RAID

- Redundant Array of Independent Disks
  - Several smaller disks play a role of one big disk
- Can improve performance
  - Data spread among multiple disks
  - Accesses to different disks go in parallel
- Can improve reliability
  - Data can be kept with some redundancy



RAID 0

- Striping used to improve performance
  - Data stored on disks in array so that consecutive "stripes" of data are stored on different disks
  - Makes disks share the load, improving
    - Throughput: all disks can work in parallel
    - Latency: less queuing delay – a queue for each disk
- No Redundancy
  - Reliability actually lower than with single disk (if *any* disk in array fails, we have a problem)

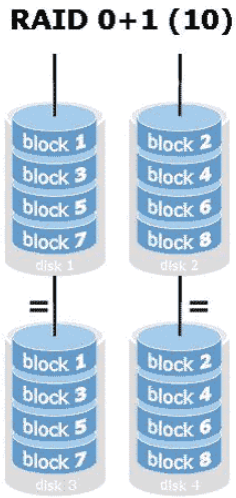


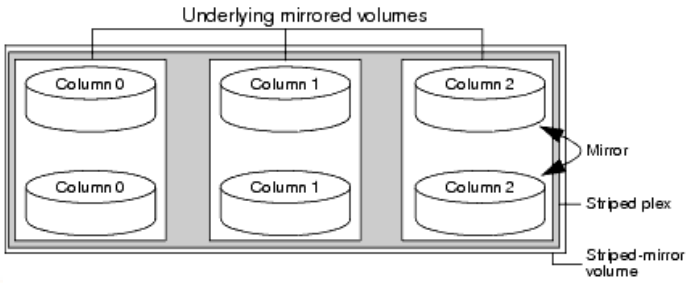
RAID 0 Reliability

- P - probability that a drive will die during one hour
  - For a single drive, MTTF = 1/P hours
- For RAID 0 with two drives:
  - Probability that both drives survive an hour: $(1-P)^2$
  - Probability of failure: $1-(1-P)^2$
  - MTTF is $1/(1-(1-P)^2)$
- Say P=0.01, MTTF is
  - 100 hours for single drive
  - 50.25 hours for 2-drive RAID-0
  - 25.38 hours for 4-drive RAID-0

RAID 1

- Disk mirroring
  - Disks paired up, keep identical data
  - A write must update copies on both disks
  - A read can read any of the two copies
- Improved performance and reliability
  - Can do more reads per unit time
  - If one disk fails, its mirror still has the data
- If we have more than 2 disks (e.g. 8 disks)
  - "Striped mirrors" (RAID 1+0)
    - Pair disks for mirroring, striping across the 4 pairs
  - "Mirrored stripes" (RAID 0+1)
    - Do striping using 4 disks, then mirror that using the other 4





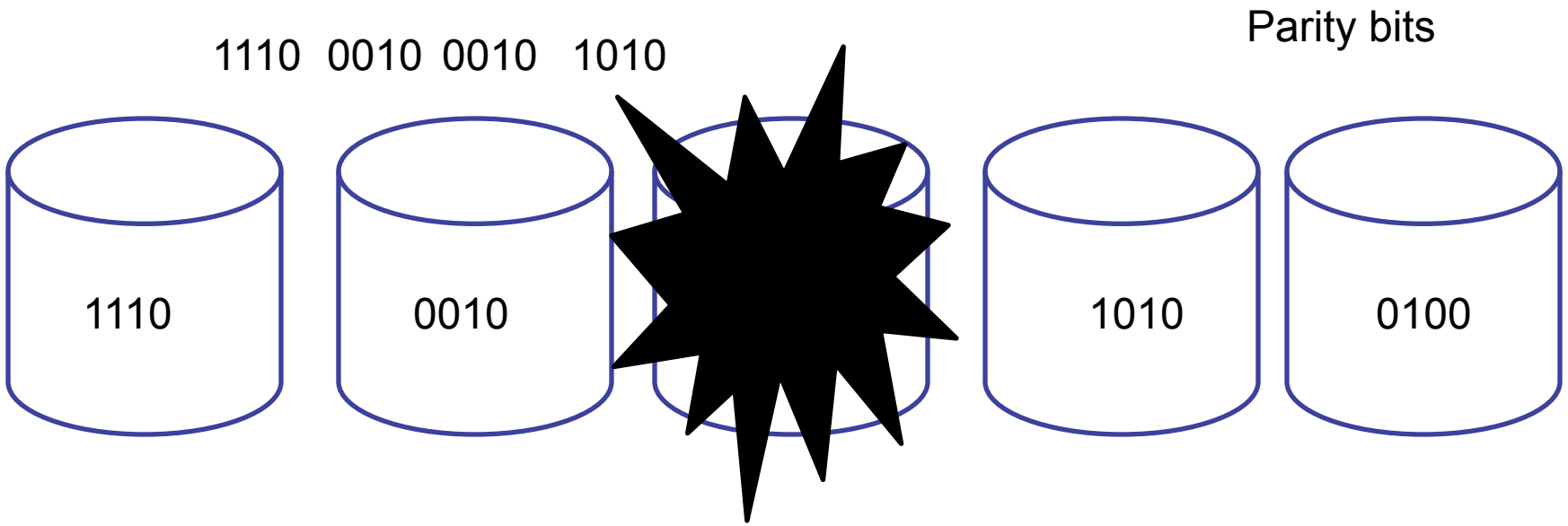


RAID 1 Reliability

- P - probability that a drive will die during one hour
  - For a single drive, MTTF = 1/P hours
- For RAID 1 with two drives:
  - Probability that both drives fail during one hour: P^2
  - MTTF is $1/P^2$
- Say P=0.01, MTTF is
  - 100 hours for single drive
  - 10,000 hours for 2-drive RAID-1
  - 100,000,000 hours for 4-drive RAID-1

Using Parity Codes

- Given N bits {b1, b2, … bN}, the parity bit will be the bit {0,1}.

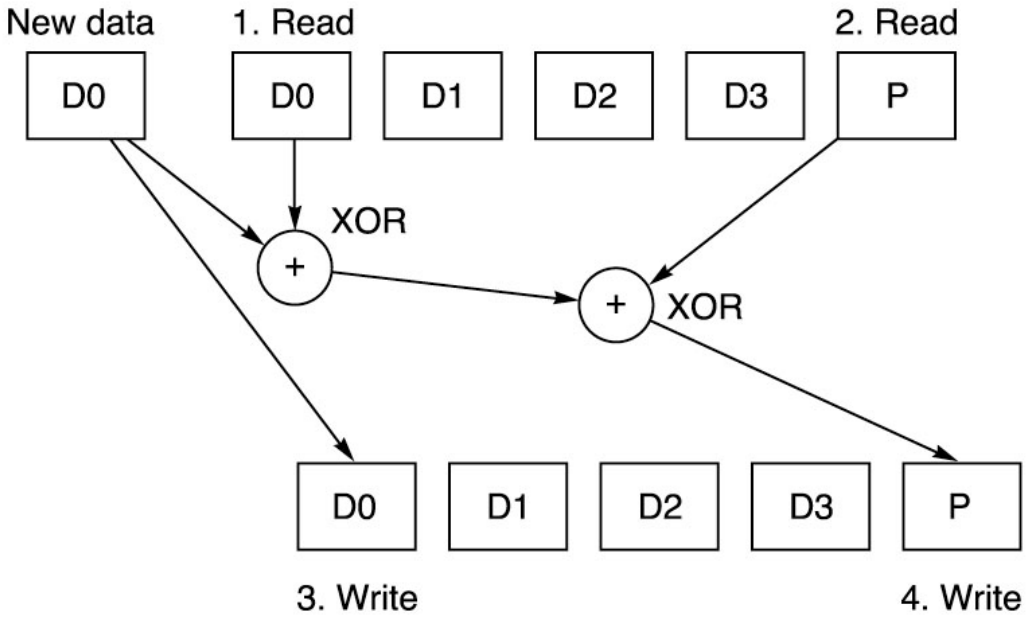




RAID 4

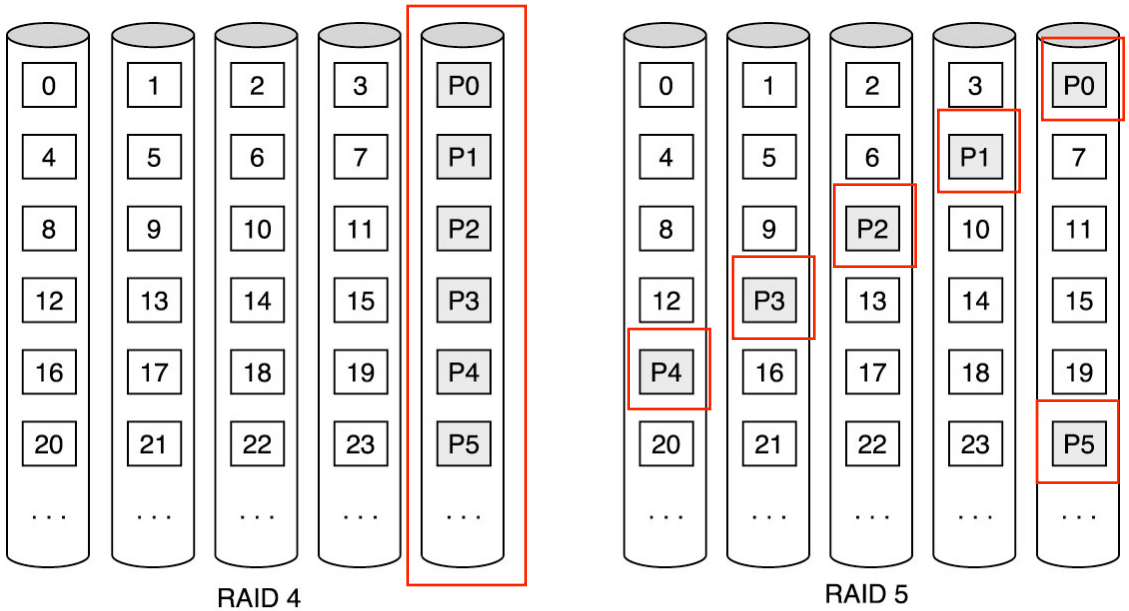
- Block-interleaved parity
  - One disk is a parity disk, keeps parity blocks
  - Parity block at position X is the parity for all blocks whose position is X on any of the data disks
  - A read accesses only the data disk where the data is
  - A write must update the data block and its parity block
  - Can recover from an error on any one disk
    - Use parity and other data disks to restore lost data
  - Note that with N disks we have N-1 data disks and only one parity disk, but can still recover when one disk fails
  - But write performance worse than with one disk (all writes must read and then write the parity disk)

RAID 4 Parity Update



RAID 5

- Distributed block-interleaved parity
  - Like RAID 4, but parity blocks distributed to all disks
  - Read accesses only the data disk where the data is
  - A write must update the data block and its parity block
    - But now all disks share the parity update load





RAID 5 Reliability

- P - probability that a drive will die during one hour
  - For a single drive, MTTF = 1/P hours
- For RAID 5 (and RAID 4) with 4 drives:
  - Probability that none of the four drives fails: $(1-P)^4$
  - Probability that exactly one drive fails: $4*P*(1-P)^3$
  - Probability that 2 or more drives fail: $1- [(1-P)^4+4*P*(1-P)^3]$
- Say P=0.01, MTTF is
  - 100 hours for single drive
  - 10,000 hours for 2-drive RAID-5 (same as RAID-1)
  - 1,689 hours for 4-drive RAID-5



RAID 6

- Two different (P and Q) check blocks
  - Each protection group has
    - N-2 data blocks
    - One parity block
    - Another check block (not the same as parity)
- Can recover when two disks are lost
  - Think of P as the sum and Q as the product of D blocks
  - If two blocks are missing, solve equations to get both back
- More space overhead (only N-2 of N are data)
- More write overhead (must update both P and Q)
  - P and Q still distributed like in RAID 5



RAID Levels

- **RAID-1: Mirroring**
  - Just copy disks = 2x disks, ½ for checking
- **RAID-2: Add error-correcting checks**
  - Interleave disk blocks with ECC codes (**parity**, XOR)
  - 10 disks requires 4 check disks
  - Same performance as level 1
- **RAID-4: block-interleaved Striping data**
  - Spread blocks across disks
  - Improves read performance, but impairs writes
- **RAID-5: block-interleaved distributed parity: Striping data & check info**
  - Removes bottleneck on check disks
- **RAID-6: striped disks with dual parity**
  - Row-diagonal parity
  - Protects against two disk failures



Coding

- Information redundancy
- Parity codes: odd(even) parity code
- Checksums
- ECC (Error correction code)

Single bit Error Correction Double Bit Error Detection

111	110	101	100	**011**	010	001	000	
D63	D62	D61	D60	D59	D58	D57	*CB7*	1000
D56	D55	D54	D53	D52	D51	D50	D49	0111
D48	D47	D46	D45	D44	D43	D42	D41	0110
D40	D39	D38	D37	D36	D35	D34	D33	0101
D32	D31	D30	D29	**D28**	~~D27~~	~~D26~~	~~*CB6*~~	**0100**
D25	D24	D23	D22	D21	D20	D19	D18	0011
D17	D16	D15	D14	D13	D12	D11	*CB5*	0010
D10	D9	D8	D7	D6	D5	D4	*CB4*	0001
D3	D2	D1	*CB3*	D0	*CB2*	*CB1*	No Error	0000



Little's Law (Review)

- Mean number of tasks in system = arrival rate x mean response time
- Server utilization = arrival rate x $\text{time}_{\text{server}}$
- Example
  - A single disk 50 I/O requests per second. The average time for a disk to service an I/O request is 10 ms. What is the utilization of the I/O system?
  - 50/sec * 0.01 sec = 0.5