I/O Devices

Characteristics of I/O Devices

- Block Devices
  - Information are stored and accessed in fixed-size blocks
  - Addressable, can have sequential or random accesses
  - E.g., disks
- Character Devices
  - Can only be accessed character by character
  - Sequential accesses only
  - E.g., terminals, printers, mouse

Devices

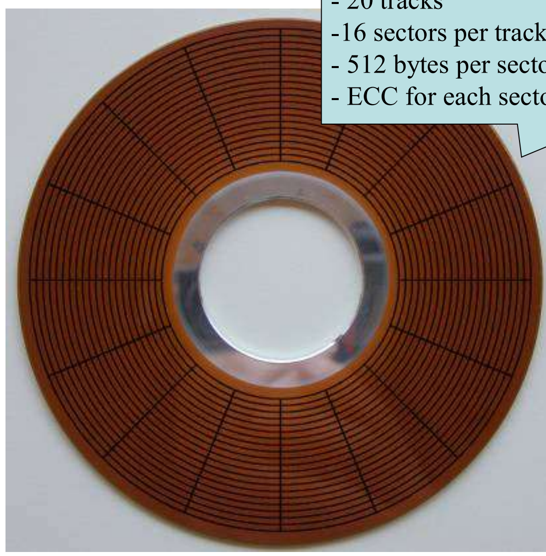
- Disk Device
  - Block device, block I/O
- Keyboard and displays
  - Character device, stream I/O
- Clock device
  - Character device

Disk Device

- **Disk Hardware**
  - Tracks and sectors
    - A disk has many tracks
    - Each track contains many sections
      - Some has fixed number of sectors per track
      - Some has variant number of sectors per track (Zoned)
    - Disk is accessed by sectors (corresponding to blocks)
  - Platters
    - Some disk system has multiple disk platters
  - Cylinders
    - The same track on multiple disk platters form a cylinder

Sample Disk Device

A platter of a 5.25" hard disk
- 20 tracks
-16 sectors per track
- 512 bytes per sector
- ECC for each sector

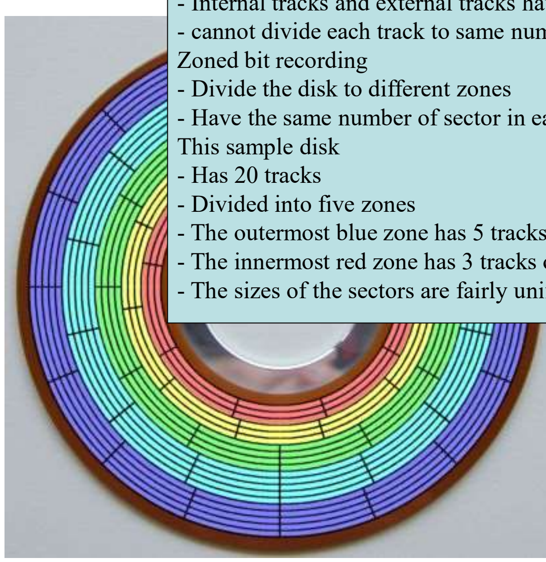


Sample Disk Device

Problem in large sized disks
- Internal tracks and external tracks have very different sizes
- cannot divide each track to same number of sectors

Zoned bit recording
- Divide the disk to different zones
- Have the same number of sector in each zone

This sample disk
- Has 20 tracks
- Divided into five zones
- The outermost blue zone has 5 tracks, each has 16 sectors
- The innermost red zone has 3 tracks of 9 sectors
- The sizes of the sectors are fairly uniform



Sample Disk Device

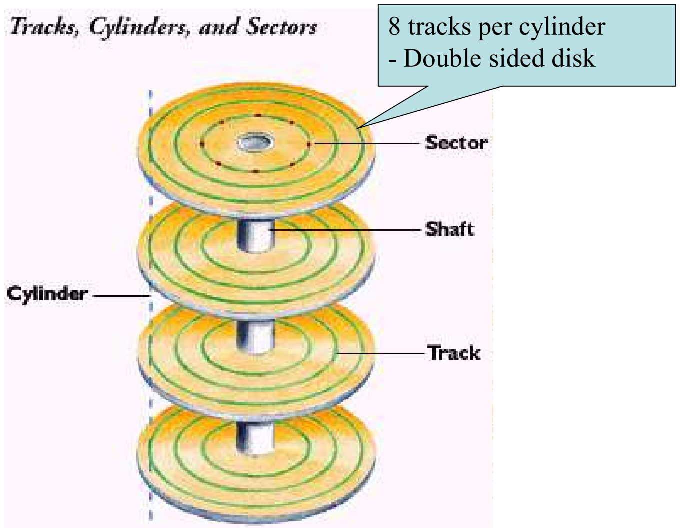
Zone	Tracks in Zone	Sectors Per Track	Data Transfer Rate (Mbits/s)
0	624	792	372.0
1	1,424	780	366.4
2	1,680	760	357.0
3	1,616	740	347.6
4	2,752	720	338.2
5	2,880	680	319.4
6	1,904	660	310.0
7	2,384	630	295.9
8	3,328	600	281.8
9	4,432	540	253.6
10	4,528	480	225.5
11	2,192	440	206.7
12	1,600	420	197.3
13	1,168	400	187.9
14	18,15	370	173.8

Sample disk format data
- 20 GB disk platter
- Zones, tracks, sectors

Disk Device

- How to read/write disk
  - Disk head (disk arm) moves to the correct track
  - Disk rotates to the correct sector
    - Disk rotates constantly
- Fixed head versus movable head
  - Most disks have movable head
  - Fixed head has one head per track
- Single/double sided
- Single/multiple platter(s)

Sample Disk Device



Disk Device

- **Disk Addressing**
  - Only address the sectors, no need for offset within a sector
  - Old style: physical address
    - Addressed by: (platter #, track #, sector #)
    - BIOS limitation: only address up to 65536 tracks
  - New addressing scheme: logical block addressing
    - Number the sectors contiguously
    - If using 32 bits to address the sector number
      - 4G sectors = 2TB disk space (if 512B per sector)
    - New 64 bit addressing
    - The actual disk standard: 22-bit, 28-bit, 48-bit

Disk Device

- **On each sector**
  - Sync: some fixed bytes to indicate the starting of a sector
    - Otherwise, how to recognize the sectors on a disk
  - ID: sector address (e.g., volume #, sector #, etc.)
  - Data: always 512B (up to now)
  - ECC code
  - Gap: to separate important fields

Disk Access Time

- Seek Time: disk head moves to proper track
  - $T_s = nm + s$
    - n: number of tracks to be traversed
    - m: traverse time per track
    - s: start up time
  - Generally, the average seek time is given
- Rotational Delay - rotate to proper sector
  - T_r: determined by the revolutionary speed
  - Consider a 3600 rpm disk
  - One revolution takes 16.7msec
  - Average rotational delay = 1/2 revolution = 8.3msec

Disk Access Time

- Transfer Time - transfer data
  - $T_t = r * (b / N)$
    - b: number of bytes to be transferred
    - N: number of bytes per track
    - r: revolution time
- Total access time
  - $= T_s + T_r + T_t$

Disk Access Time (Example)

- Disk spec
  - 512 bytes per sector
  - 256 sectors per track
  - 3600 rpm
  - Traverse time per track = 0.1 msec
  - Startup time = 0.3msec

Rotational delay:
3600 rpm → 60 rps
→ 1/60 sec per revolution
→ 16.7 msec per revolution
Rotational delay
= 1/2 revolution time
= 8.3msec

Disk Access Time (Example)

- Read a file
  - File size: 1MB
  - Consecutively allocated
  - Starting track = 105
  - Current track = 5

Seek time (first track)
= (105–5)*0.1ms + 0.3ms
= 10.3 msec

Transfer time:
Transfer a track in 16.7 msec
256 sectors/track, 0.5KB/sector
→ Transfer 128KB per 16.7msec
→ Per track seek time = 0.1msec
1MB file requires 8 tracks
Transfer time = 16.7*8 + 0.1*7
= 134.3 msec

Total time required
= 10.3 + 8.3 + 134.3 msec
= 152.9 msec

Disk Access Time (Example)

- Track skew
  - If the starting sector of each track is at the same location
    ⇒ Incurs a big rotational delay for each track switch
  - In the previous example:
    - In sequential read, move to next track requires 0.1msec
    - Rotational continues ⇒ missed the first sector on the new track
      ⇒ need to wait for the rotation, almost 16.7ms
  - Solution: skew the alignment
    - Start the first sector of a consecutive track a few sectors off
    - In the example, it is roughly 1.5 sectors off the previous track
  - Without track skew, the previous example:
    - Add 7 additional revolution time
      - On each track, need to wait for almost the entire revolution

Disk Access Time (Example)

- Disk spec
  - 512 bytes per sector
  - 256 sectors per track
  - 7200 rpm
  - Average seek time = 10 msec
- Read a file
  - File size: 1MB
  - Dynamically allocated
  - File block size = 4KB

Rotational delay:
7200 rpm → 120 rps
→ 1/120 sec per revolution
→ 8.3 msec per revolution
Rotational delay
= 4.2 msec

Transfer rate:
Transfer a file block
(8.3 / 256 msec) * 8
= 0.26 msec per block

Total time required
= (10 + 4.2 + 0.26) * 256
= 3702 msec = 3.7 sec

Disk Access Time Samples

Series	Seagate U6	Maxtor DiamondMax VL40	Western Digital WDx00AB
Formatted Capacity (GB)	80/60/40	40/30/20/10	80/60/40/30
Internal Transfer Rate (max) (Mbits/sec)	436	374	424
Average Seek Time, Read (ms)	8.9	9.5	9.5
Track-to-Track Seek, Read (ms)	1.2	1.0	2.0
Average Latency (ms)	5.55	5.55	5.0
Buffer Size	2 MB	2 MBs	2 MB
Spindle Speed (RPM)	5400	5400	5400

Disk Access Time Samples

Series	Seagate U6	Maxtor DiamondMax VL40	Western Digital WDx00AB
Formatted Capacity (GB)	80/60/40	40/30/20/10	80/60/40/30
Internal Transfer Rate (max) (Mbits/sec)	436	374	424
Average Seek Time, Read (ms)	8.9	9.5	9.5
Track-t… Read (r…			
Averag…			
Buffer …			
Spindle…			

Note: disk head movement is not exactly
traverse time per track * # tracks to travese
The disk head moving from beginning to end is called a full stroke.
A full stroke time is roughly 20-30ms.
So when the disk head has to move a large number of tracks, it will move very fast and then slow down in order to stop at the correct track, and this can be much faster than per track movement.

OS Issues for Disk Devices

- Disk Allocation Strategies
  - Allocation for files
  - Try to put data blocks of a file close together
    - To avoid too long a seek time
  - Discussed with the file systems
- Disk-arm Scheduling Algorithms
  - Try to read closest track next
    - To reduce seek time

Disk-arm Scheduling Algroithms

- First-Come-First-Served
  - simple, but less efficient
- Scan
  - Serve the request closest to current track toward the current arm direction
- C-Scan
  - Always consider forward direction of arm movement
- FScan
  - Problem in other Scan algorithms: Arm-stickiness (starvation problem)
  - Use two queues, when one queue is served, all new requests go to the other queue

Disk-arm Scheduling Algroithms

- Consider the scenario
  - Request sequence: 5, 102, 95, 102, 210, 80, 45, 200, 105
  - Current arm location: 90 with forward direction
- First-Come-First-Served
  - 90 – 5 – 102 – 95 – 102 – 210 – 80 – 45 – 200 – 105
    - Total tracks traversed: 85+97+7+7+108+130+35+155+95 = 719
- Scan
  - 90 – 95 – 102 – 102 – 105 – 200 – 210 – 80 – 45 – 5
  - Total tracks traversed: 90 --- 210 --- 5 = 325
- C-Scan
  - 90 – 95 – 102 – 102 – 105 – 200 – 210 – 5 – 45 – 80
  - Total tracks traversed: 90 --- 210 --- 5 --- 80 = 400

Clock Device

- Clock device
  - Hold register
    - Every clock has a different speed (quartz oscillation)
    - Hold register holds the count = # oscillations per unit time
  - Counter
    - At the beginning of each time unit, reset to hold register
    - Decremented the counter
    - When counter = 0, generate an interrupt signal
    - Each interrupt period is called a clock tick
  - Example
    - Time unit is 10μs, Hold register value is 100, counter value is 38
    - After how long will a clock tick happen? ⇒ 3.8μs
    - Of course, the data here is far from reality

Clock Device

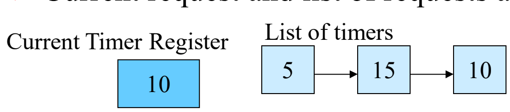
- Clock register
  - At each clock tick, increment the clock register
  - The second level register keeps the time since 1/1/1970, in seconds
  - The lower register keeps the finer readings, e.g., microseconds
  - E.g., On Feb 11 of 1971 at time 3:30:10am, the second level register value is: (365+31+10)*86400+3*3600+30*60+10
- Clock device
  - Keep track of time + Provide alarm service
  - Every clock tick → Check whether any alarm is up
  - User or OS can "setTimer"
- All computer components sync on clock ticks
  - Data/control-signal send/receive in the clock cycle
  - CPU clock cycle is generally at a finer grain

Clock Device

- Alarm service by OS
- OS maintain a list of timer requests
  - Requests from OS and users
    - E.g., refresh monitor every 1/30 seconds
    - E.g., CPU time quantum expiration
    - E.g., sleep (5)
  - List is in sorted order
  - Each entry:
    - Time is kept in terms of # clock ticks from previous entry
  - First entry is copied to a register
  - The register decremented at every clock tick
  - When the register = 0, an timer interrupt is generated

Timer Requests Example

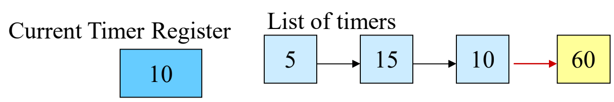
- Example:
  - Each clock tick = 1 μsec
  - Current time: 15:59:59 and 999850 μsec
  - Current request and list of requests as follows



  - Requests
    - Next time quantum: 100 μsec after, issued at current time
    - User request: usleep (30), set at 3 time units after
    - User request: alarm at 16:00 exact, set at the same time as above
      - Convert: 147 μsec after

Timer Requests Example

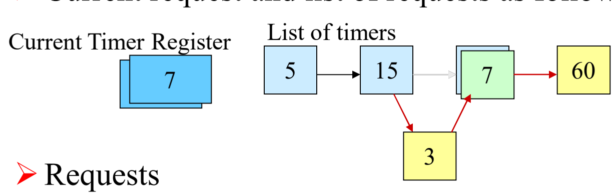
- Example:
  - Each clock tick = 1 µsec
  - Current time: 15:59:59 and 999850 µsec
  - Current request and list of requests as follows



  - Requests
    - Next time quantum: 100 µsec after, issued at current time
    - User request: usleep (30), set at 3 time units after
    - User request: alarm at 16:00 exact, set at the same time as above
      - Convert: 147 µsec after

Timer Requests Example

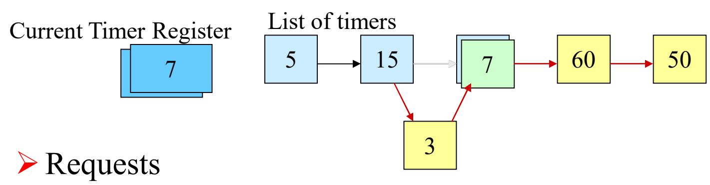
- Example:
  - Each clock tick = 1 µsec
  - Current time: 15:59:59 and 999850 µsec
  - Current request and list of requests as follows



  - Requests
    - Next time quantum: 100 µsec after, issued at current time
    - User request: usleep (30), set at 3 time units after
    - User request: alarm at 16:00 exact, set at the same time as above
      - Convert: 147 µsec after

Timer Requests Example

- Example:
  - Each clock tick = 1 μsec
  - Current time: 15:59:59 and 999850 μsec
  - Current request and list of requests as follows



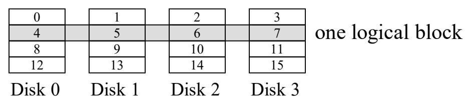
  - Requests
    - Next time quantum: 100 μsec after, issued at current time
    - User request: usleep (30), set at 3 time units after
    - User request: alarm at 16:00 exact, set at the same time as above
      - Convert: 147 μsec after

Terminal Device

- Include keyboard and display
  - Use RS-232 protocol transmits 1-bit at a time (serial port)
  - Transmission rate: 1.2-19.2Kbps
    - New graphical monitor uses a different protocol to efficiently refresh screen
- CPU gets interrupted for every character typed
  - Echo whatever being typed
  - Overhead for CPU
    - New intelligent terminal system directly route the typed character from keyboard to monitor
    - CPU only gets interrupted after the <return>

Parallel Disks

- Concept
  - N disks being accessed in parallel
    - A logical block consists of N physical blocks



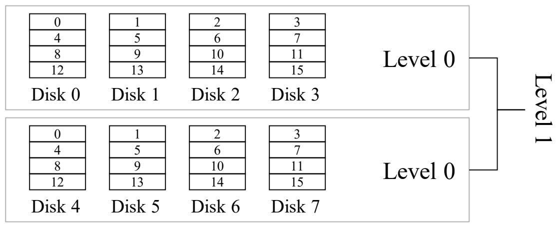
    - N folds of bandwidth
      - Transfer rate of one disk: 10Mb/sec $\Rightarrow$ Transfer rate for 32 parallel disks: 320Mb/sec
  - Issus
    - Performance: How to store the data to maximize the performance
    - Failure: The probability that one of the N disks will fail increases

RAID

- RAID
  - Redundant Array of Inexpensive Disks (RAID)
    - Redundant: To incorporate fault tolerance
    - Various configurations to achieve different tradeoffs
  - RAID level 0: N parallel disks without redundancy
    - No fault tolerance
      - Claimed: one disk fails $\Rightarrow$ system fails, data lost
    - Block level striping
  - RAID level 1: Mirroring
    - No stripping (new definition, relative to RAID 01 and 10)
    - Can read mirrored disks in parallel, “may” double the performance (depend on the layout and what are to be retrieved)
    - Have to write to all disks

RAID

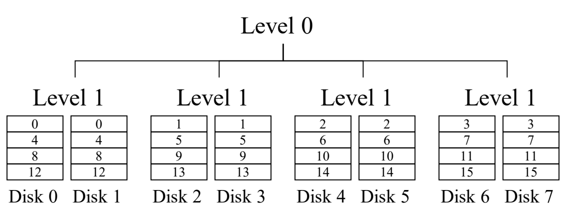
- RAID level 01 (0+1)
  - Mirroring, fully replicate the disks (= original definition for level 1)
  - Can be configured as nested controllers: two level 0 controllers, each controls one stripped disk array, one level 1 controller on top



  - Fault tolerant, but less robust
    - If one disk of each level 0 RAID fails ⇒ system failure
  - Reduced the space to 1/2 + has to write to mirroring disks

RAID

- RAID level 10 (1+0)
  - Level 1 at the low level and level 0 at the high level



  - More robust than 0+1
  - System would work as long as one of the two mirrors is working

RAID

- RAID Level 1 in general

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

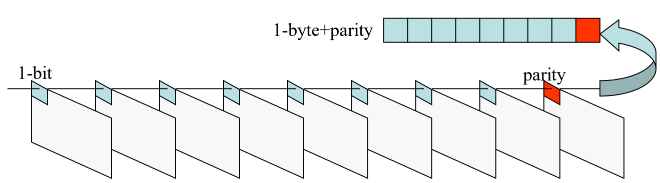
  - Reading: All disks can read in parallel
  - But does not do well in sequential reads
    - Example read
      - block 0 on disk 0; block 1 on disk 2; block 2 on disk 1; 3 on disk 3
      - block 4 on disk 0; block 5 on disk 2; block 6 on disk 1; 7 on disk 3
    - ⇒ Each disk reads alternate blocks, the disk hardware is anyway rotating through those blocks ⇒ No speed up at all
    - How about this layout:
      - But current design is simple mirror

Disk 0	Disk 1	Disk 2	Disk 3
0	2	1	3
4	6	5	7
…	…	…	…
2	0	3	1
6	4	7	5

  - No problem with random reading, speed up by 4X with 4 disks
  - Writing: Synchronization problem
    - A write is only done when both writes are done
    - Failure of one disk causes inconsistency of the two disks

RAID

- RAID level 2
  - Block level striping
  - Uses Hamming code for error correction
  - Some disks are dedicated to store the Hamming code
- RAID level 3
  - bit-interleaved with a parity disk ⇒ tolerate any single disk failure
  - Read/Write access all disk units ⇒ high bandwidth but not good if most data objects are small
  - Frequently used in multimedia applications



RAID

- ➢ RAID level 4
  - block-interleaved, use parity disk
    - Multiple small data objects can be accessed in parallel ⇒ Better performance if system has a significant number of small objects
  - Problem
    - Read: do not access the parity disk
    - Write:
      - o Parity disk may be accessed at a higher rate ⇒ hinder performance
      - o Need extra disk accesses to compute parity
      - o E.g., write to 0, 5, 14, 11 ⇒ Which disks need to be accessed?
      - o E.g., write to 0, 1, 2 ⇒ Which disks need to be accessed?
    - Both read and write operations create unbalanced load

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P4
8	9	10	11	P8
12	13	14	15	P12

Level 4

RAID

- ➢ RAID level 5
  - block-interleaved, use parity disk (but distributed them over)
  - Read/Write accesses will be distributed more evenly on disks
    - level 5 is the most commonly used scheme

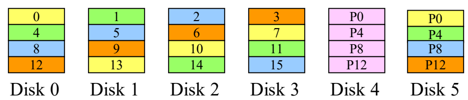
Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	P4	7
8	9	P8	10	11
12	P12	13	14	15
P16	16	17	18	19

Level 5

RAID

❖ RAID 6

- Block level striping
- 2 parity disks, horizontal parity + diagonal parity
- How to compute the diagonal parity?
  - Consider K data disks and K data size (w = K)



0	0	0	0	0	0
1	0	0	0	1	0
0	0	1	1	0	0
1	1	0	0	0	1

- But recovery may be a problem!

RAID

❖ Recovery problem in simple 2-parity solution

- Consider two disk failures

0	?	0	?	0	0
0	?	0	?	0	0
0	?	0	?	0	0
0	?	0	?	0	0

- Multiple solutions

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

0	1	0	1	0	0
0	1	0	1	0	0
0	1	0	1	0	0
0	1	0	1	0	0

RAID-DP

- RAID double parity
  - Used in RAID-6
  - Consider K disk and K data bits from each disk
    - K+1 is a prime

0	1	1	0	0	1
1	1	0	0	0	0
0	1	1	1	1	1
0	0	0	1	1	1

  - Horizontal parity
  - Diagonal parity
    - The horizontal parity is also used to compute the diagonal parity
    - Gray diagonal has no diagonal parity

RAID-DP

	1	2	3	4	5	
1	0	1	1	0	0	1
2	1	1	0	0	0	0
3	0	1	1	1	1	1
4	0	0	0	1	1	1
0	0	0	0	0	0	0

1 disk failure or
1 data disk + D-parity disk
⇒ no problem

- Any two disk failures, not the diagonal parity
  - Horizontal parity disk is the same as data disk
    - They are horizontal parities for each other
  - If disk 1 and another disk failed (e.g., 3)
    - Always can find a non-gray diagonal with only 1 lost bit
      - The other diagonal is gray, which also lost only one bit
    - Blue: (2,1) → (2,3) → (4,1) → (4,3) → (1,1) → (1,3) → (3,1) → (3,3) → done

RAID-DP

	1	2	3	4	5	
1	0	1	1	0	0	1
2	1	1	0	0	0	0
3	0	1	1	1	1	1
4	0	0	0	1	1	1
0	0	0	0	0	0	0

- Any two disk failures, not the diagonal parity
  - If failed does not include disk 1 (e.g., 2 and 5)
    - Always can find 2 non-gray diagonals with only 1 lost bit
    - Orange: (3,2) → (3,5) → (1,2) → (1,5) → stuck
    - Green: (2,5) → (2,2) → (4,5) → (4,2) → stuck, but done

Readings

- Sections 11.1-11.2, 11.5
- Section 1.7
- RAID
    - E.K. Lee and R.H. Katz, “The performance of parity placement in disk arrays, ” IEEE Trans. Computers, vol. 42, no. 6, pp. 651-664, June 1993.
    - Row-diagonal parity for double disk failure correction, USENIX FAST, 2004.