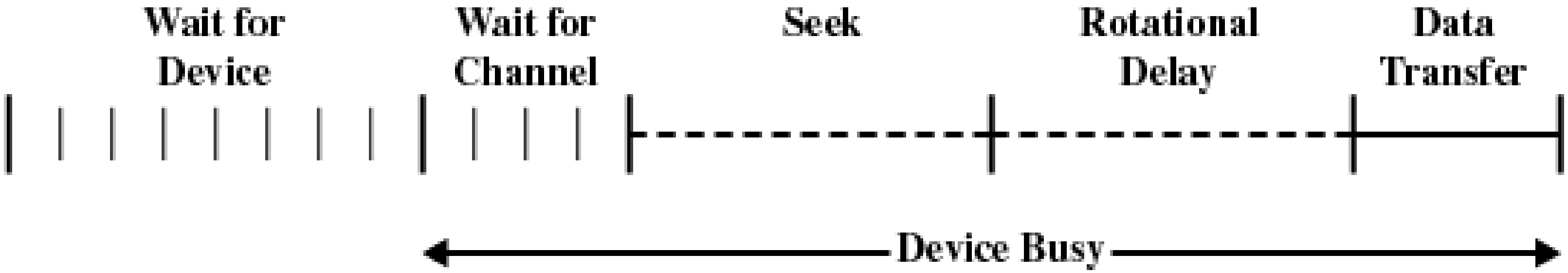
disk scheduling

not on
the book

(regular) Disk Performance Parameters

- To read or write, the disk head must be positioned at the desired track and at the beginning of the desired sector



- this holds only for regular disks
- solid-state/flash drives and “virtual drives” really have different characteristics!

not on
the book

Disk Performance Parameters

- Seek time
  - Time it takes to position the head at the desired track
  - 1 to 20 ms. (average 8 ms)
- Rotational delay or rotational latency
  - Time it takes for the beginning of the sector to reach the head
  - for 7200 rpm -> 8.3 ms for a full rotation and 4.2 for the average

not on
the book

Disk Performance Parameters

- access time
  - sum of seek time and rotational delay
  - the time it takes to get in position to read or write
- data transfer occurs as the sector moves under the head
  - transfer occurs in the disk buffer...
  - ...then in the controller buffer...
  - ...than in main memory
  - **data transfer is much faster than access**
    - several hundreds of sectors per track in current HD

not on
the book

disk scheduling: formal statement

- input:
  - a set of requests (tracks to seek)
  - current disk head location.
  - other algorithm state (e.g. current head direction)

- output: the next request to serve

not on
the book

goals

- max throughput
- fairness
  - disk scheduling with priorities is rare
  - avoid starvation
  - avoid very long waits
- starvation vs. unfairness
  - starvation: a request never served
  - unfairness: certain requests wait longer
  - warning: often confused!

FIFO

- process request sequentially
- fair, no starvation
- if there are many processes it performs like random scheduling

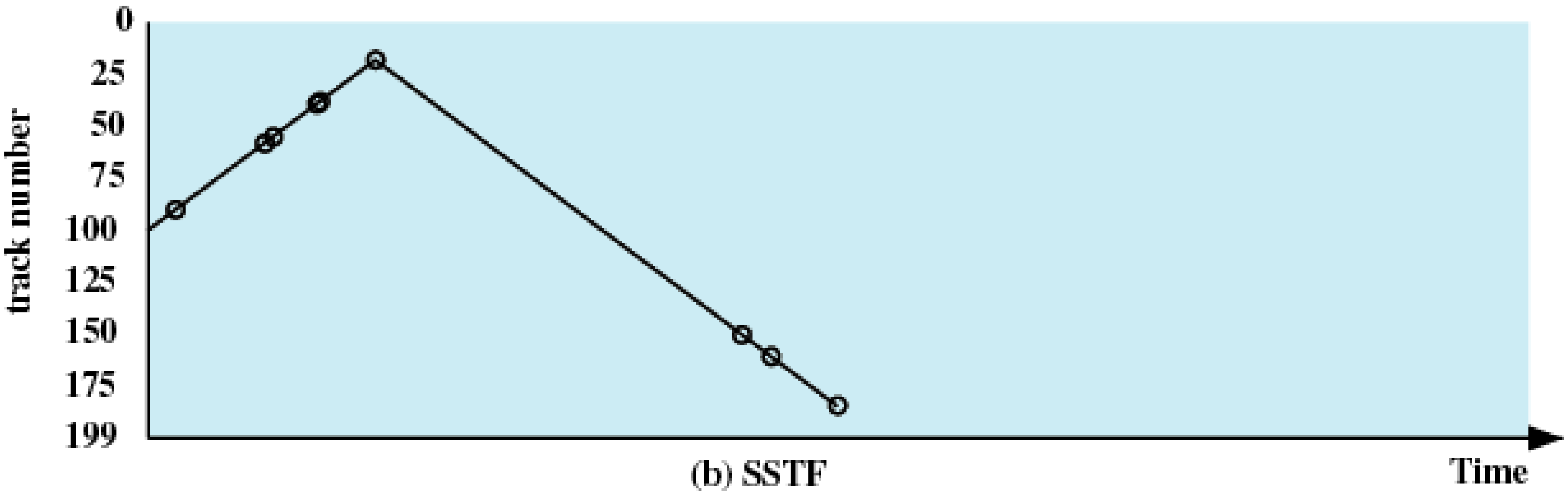
LIFO

- Last-in, first-out
- Good for transaction processing systems
  - The device is given to the most recent user so there should be little arm movement
- possibility of starvation since a job may never regain the head of the line
- only of theoretical interest

not on
the book

shortest service time first

- Select the disk I/O request that requires the least movement of the disk arm from its current position
- optimal throughput!
- highly unfair!
- starvation when consecutive requests are for near tracks

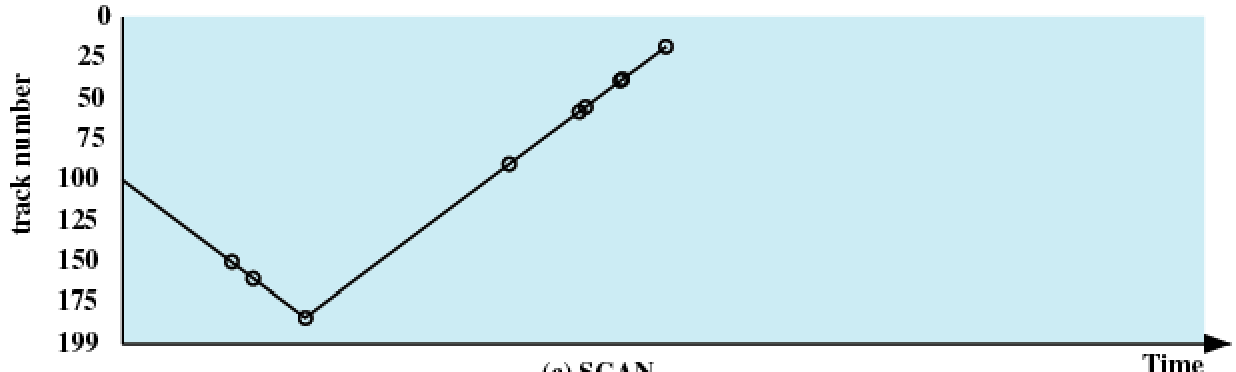


(b) SSTF

not on
the book

elevator

- famous, also called SCAN or LOOK
- arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction
- at the end direction is reversed
- good throughput
- unfair: maximum wait time for tracks on the edge is twice that for the tracks in the middle (same average)
- starvation only for continous read on the same track
- used in practice

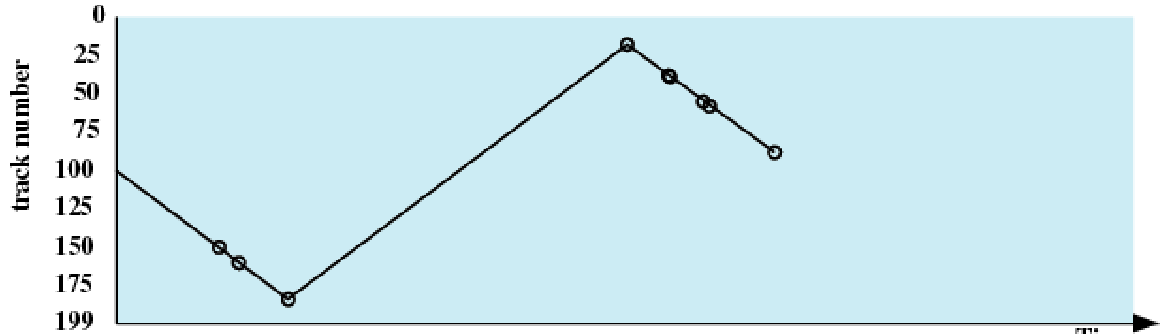


(c) SCAN

not on the book

one-way (or cyclic) elevator

- also called C-SCAN
- like elevator but restricts scanning to one direction only
- when the last track is reached, the arm is returned to the opposite end and the scan begins again
  - performance penalty with respect to elevator
- good throughput
- fair
- starvation only for continous read on the same track
- used in practice



(d) C-SCAN

not on
the book

request merging

- requests for adjacent blocks are treated as one
- avoid access time penalty
  - use the access of the first request
- may increase performance
  - e.g. irrelevant for SSTF, important for FIFO
- used in real systems

not on
the book

the "write-starving-reads" problem

- writes can be issued sequentially
  - processes usually do not depend on write request been actually fulfilled
- read are not usually issued sequentially
  - processes wait for the data before requesting the next read operation
- not really "starvation", actually is "unfariness"

not on
the book

linux

- noop
  - FIFO + request merging
- deadline
  - one-way elevator
  - reads cannot wait more 500ms, and writes 5s, to avoid the write-starving-reads problem
    - but it seeks back an forth to meet deadlines!

not on
the book

linux

- anticipatory
  - request merging
  - deadline approach
  - after a read, waits 6ms for another read
    - avoid the write-starving-reads problem
    - no seek back and forth
  - waiting is not always performed
    - heuristics to estimate the behavior of the running processes are implemented
  - good for streaming, bad for dbms

linux

not on the book

- complete fair queueing (cfq)
  - request merging
  - one-way elevator
  - fair with a round robin approach
    - schedule requests of each process for a few milliseconds
    - if no more requests for the scheduled process, wait a bit for further requests in the time slice
    - support process I/O priorities (command *ionice*)
  - good for mutliuser environments
  - latest versions performs as good as “anticipatory” for streaming
  - usually set as default

not on the book

linux: disk scheduling switching

- scheduling algorithm can be switched
  - at run time
  - per device

```
pisolo:~# cat /sys/block/hda/queue/scheduler
noop anticipatory deadline [cfq]
pisolo:~# echo anticipatory >
  /sys/block/hda/queue/scheduler
pisolo:~# cat /sys/block/hda/queue/scheduler
noop [anticipatory] deadline cfq
```

not on the book

solid state (flash/pen) drives

- no seek time!
  - elevator does not bring any benefit
- random writes performs poorly on SSD
- write-starving-reads is still a problem
  - but this is an application behavior problem
- open issues
  - auto select scheduler?
  - which scheduler!
    - noop is recommended
- google for speed-up tricks for SSD or on

not on the book

virtual drives (virtual machines)

- it should be treated as an ideal drive
- optimization should be performed by the virtualization layer
- different underlying technologies are possible
  - plain file on conventional disk
  - plain file on SSD
  - Storage Area Network
  - etc.

not on
the book

RAID

not on
the book

RAID

- Redundant Array of Independent Disks
- Set of physical disk drives viewed by the operating system as a single logical drive
- improves...
  - ...performance
  - ...fault tollerance when one or more hard drives fail
    - availability: the service may still be available when a fault occour
    - data security: no data loss

not on
the book

degradation, rebuilding, and spare disks

- when a disk fails the array enters “**degraded**” (or **critical**) state
  - performance and redundancy is not as the full working array
- when the disk is substituted it must be updated with the data to fully work in the array: **rebuilding**
  - lasts hours, performance may be even worse
- substitution can be automatic in systems that have unused disks available (**hot spare** disk)

not on
the book

techniques

- **mirroring**
  - data are stored duplicated on (at least) two disks
  - duplexing: the two disks are controlled by a distinct controller
- **parity** or humming error correction code
  - redundant bits are stored with the data
- **striping**
  - “consecutive” data are distributed across the physical drives of an array
  - bit, byte, block level granularity

not on
the book

several kinds of RAIDs

warning: this terminology is not followed by everybody

- **RAID0 used a lot, inexpensive**
- **RAID1 used a lot, inexpensive**
- RAID2 (not used any more)
- RAID3
- RAID4 similar to RAID3
- **RAID5 used a lot**
- RAID6 rare, expensive
- RAID7 proprietary
- nested (or multiple) RAID
  - 01/**10**, 03/30, 05/50, 15/51, ecc.

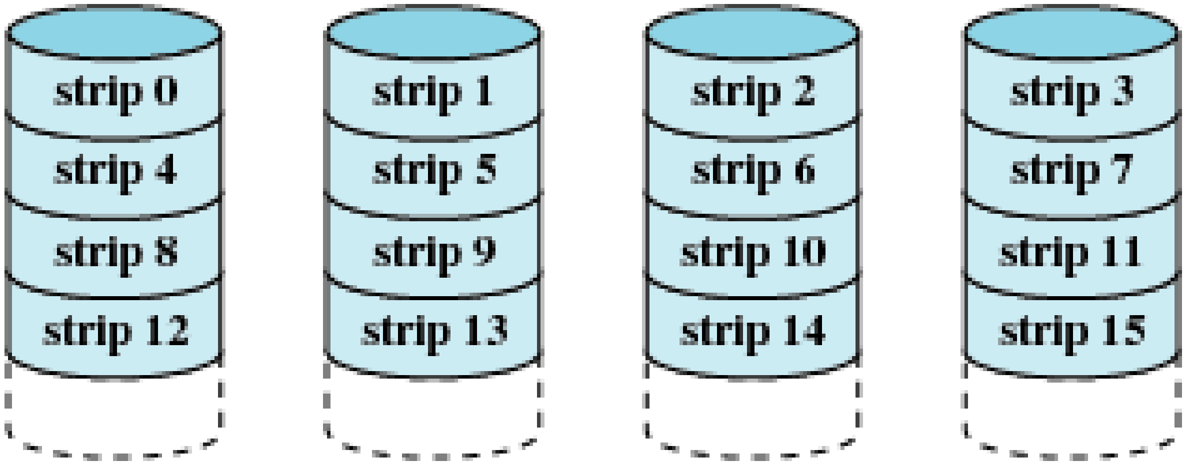
not on
the book

type of requests

- sequential I/O
  - big files (streaming, bulk)
  - blocks are stored contiguously
- random I/O
  - high frequency of requests for a very small amount of data (OLTP)
  - blocks are scattered through the disk
- read, write

not on
the book

RAID 0 (non-redundant)



- just striping
- I/O: always very good
  - on average speedup xN (almost)
- failure of one disk makes all the array to fail

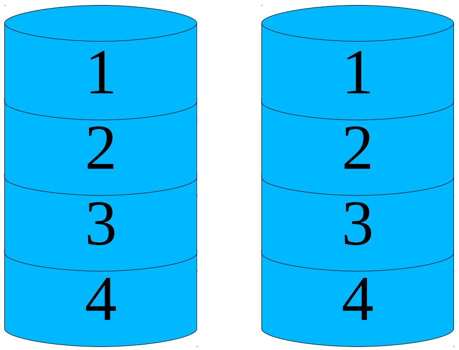
not on
the book

RAID0: mean time between failures

- MTBF= 1/λ
  - where λ is the fault frequency
- if a system A fail when one of its components D1,..., DN fails $\lambda_A=\lambda_{D1}+\lambda_{D2}+...+\lambda_{DN}$
  - frequencies can be summed up if faults are independent
- if disks are identical and in RAID0
- $MTBF_{Array}=MTBF_{Disk}/N$

not on
the book

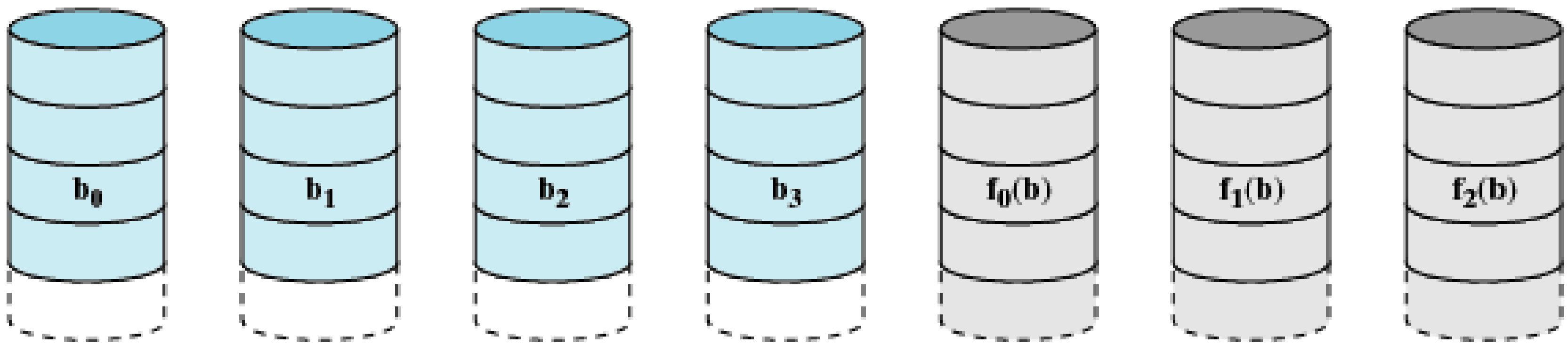
RAID 1



- just mirroring (or duplexing)
  - rarely more than one mirror
- one disk may fail but the other still works
- I/O:
  - writes must be done on both disks
  - reads can be done in parallel on the two disk
- limited in size by the size of a single disk

not on
the book

RAID 2



(c) RAID 2 (redundancy through Hamming code)

- bit-level striping, disks should be syncronized
- error correction by humming code, useful for high bit error rate
  - but now all HD have error correction code built in!
- requires expensive proprietary hw, never used!

not on
the book

RAID 3

- byte-level striping with dedicated parity disk
  - a disk block contains more stripes
    - usually less then 1024 bytes in length
- read: very good, see RAID0
- write: poor, parity disk is a bottleneck, computation load on the CPU, hw implementation is preferred
- tolerant to 1 disk failure (rebuilding by XOR)

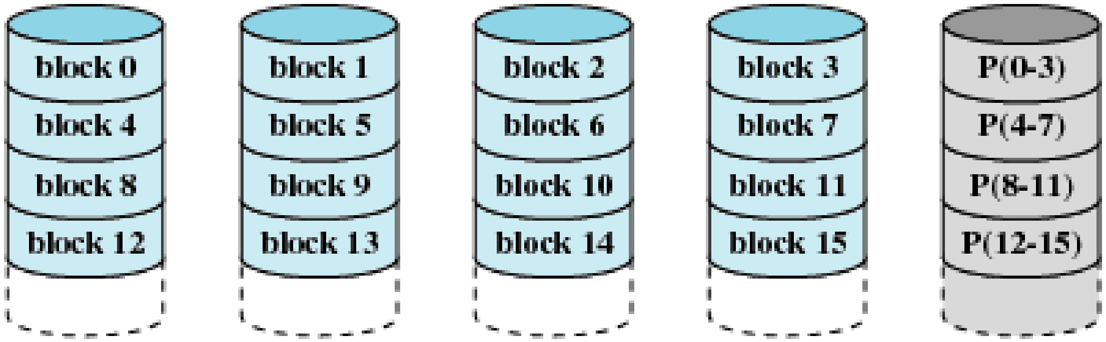
not on
the book

RAID3 write: computing parity

- in the same row: stripes A, B, C and parity stripe P = A xor B xor C
- A is written as A'
- new parity P' can be computed in two ways
  - P' = A' xor B xor C
    - read B and C? a cache may be very much useful
  - P' = A' xor A xor P
    - A may be read and in cache, P should be read.

not on
the book

RAID 4

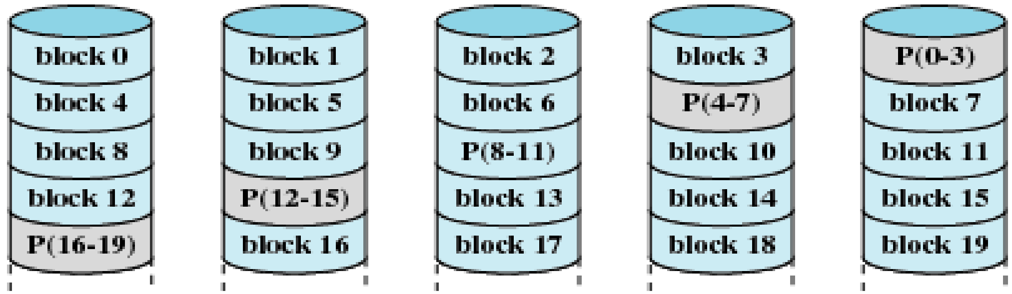


(e) RAID 4 (block-level parity)

- block-level striping with dedicated parity disk
  - a stripe spans more disk blocks
- see RAID3

not on
the book

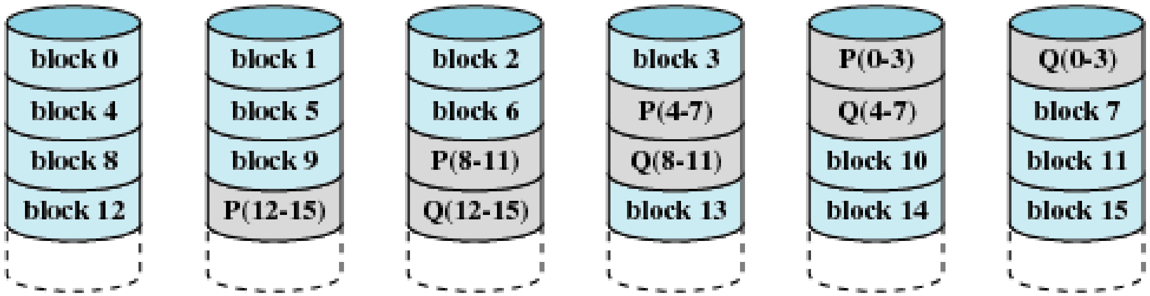
RAID 5



- block-level striping with distributed parity
- read: slightly better than RAID0 (1 disk more)
- write:
  - better than RAID4 (no parity disk bottleneck)
  - computation requires knowledge of the entire row or old parity (read? cache?)
  - cpu intensive, hw implementation is preferred
- tolerant to 1 disk failure (rebuilding by XOR)

not on
the book

RAID 6



(g) RAID 6 (dual redundancy)

- block-level striping with *dual* distributed parity
- two disks may fail without data loss
- read: slightly better than RAID5 (1 disk more)
- write: slightly worse than RAID5 (2 parities)
- costly

not on
the book

nested RAID arrays

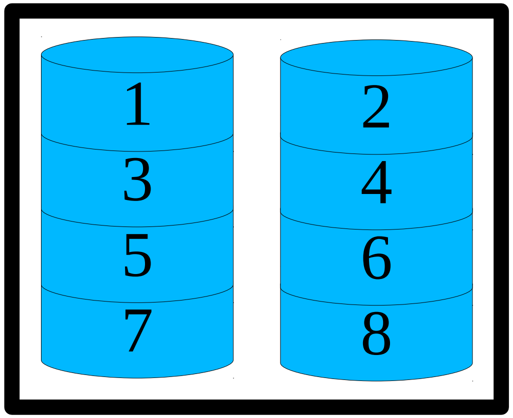
- the idea: treat several raid array as a disk and build a RAID array out of them
- notation: RAID XY
  - means that you first have several RAID-X array and you build a RAID-Y array out of them
  - warning: this notation is not followed by everybody

not on
the book

example: RAID 10

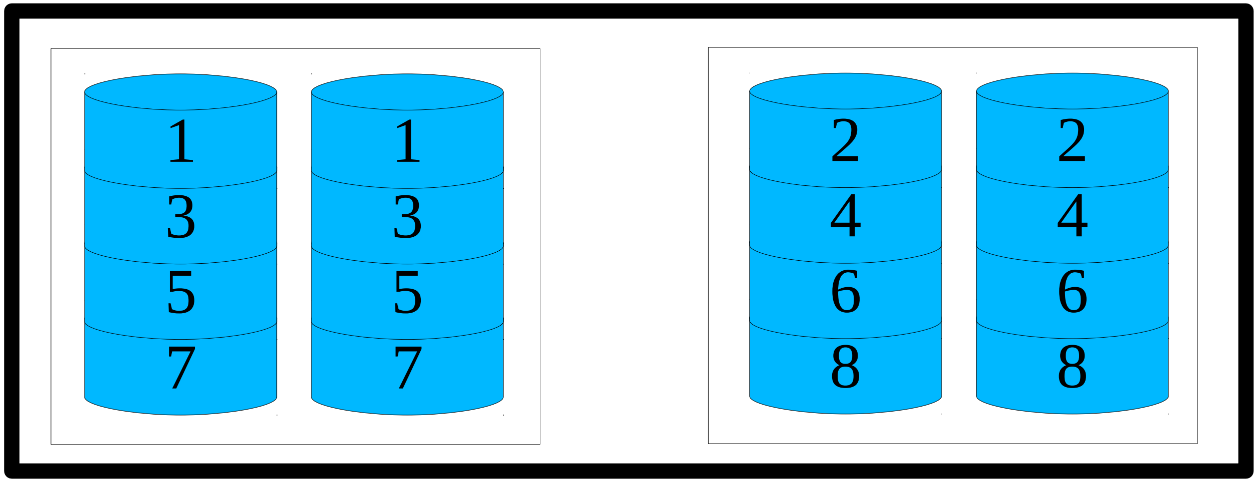
- it is like composition of mapping functions

Standard RAID 0



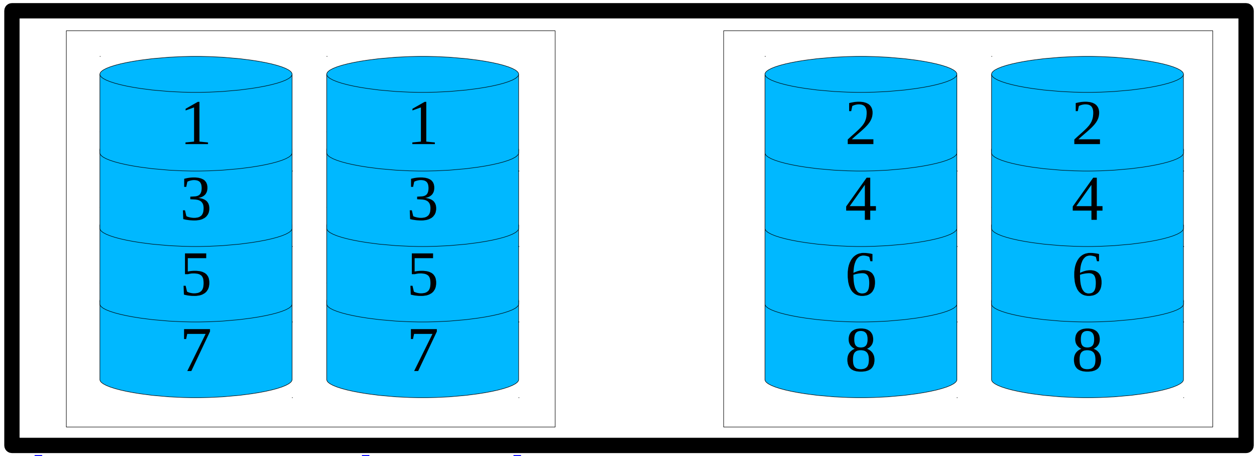
RAID 10: that is...
RAID 0 on top of disks implemented as RAID 1

The two mappings are independently performed.
This process does not depends on the type of RAID considered



not on
the book

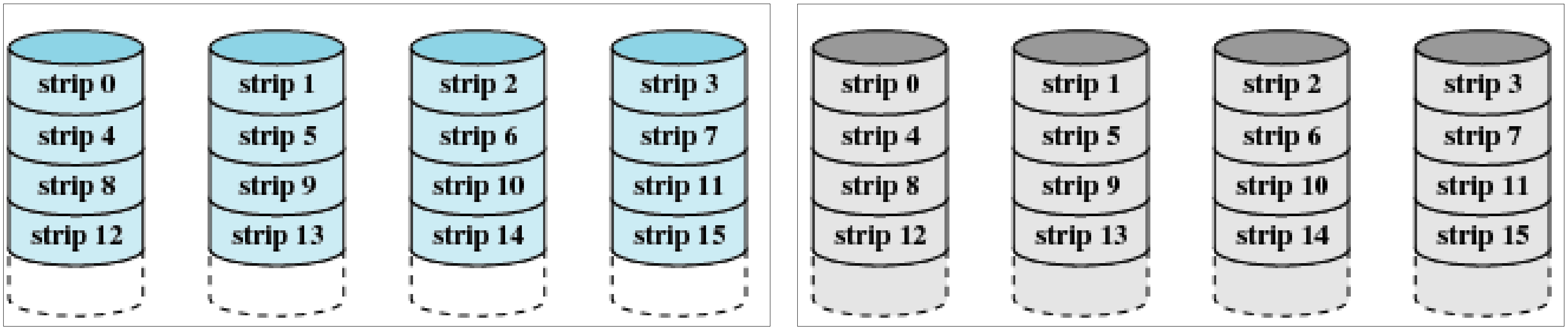
RAID 10



- striping on mirroring
- performances very very good (see RAID0 and 1)
- each RAID1 array can support a disk failure
- good performances also in rebuilding
  - only the affected array is slowed down

not on
the book

RAID 01



- mirroring on striping
- as RAID 10 but when one disk fails **the whole RAID0 array is considered down** by many controllers
- avoid it

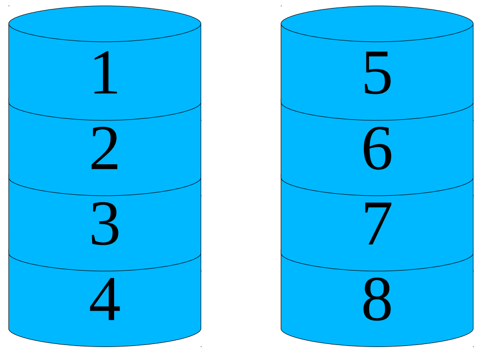
not on
the book

RAID: suggested exercises

- how are organized blocks of a logical disk in
  - RAID 50, RAID 05
  - RAID 51, RAID 15
  - choose the number of disks as you think is more useful
- apply the same rule for a three level RAID
  - RAID 150
  - this configuration is never (or rarely) used in practice

not on the book

just a bunch of disks (JBOD)



- a.k.a spanning, not really a RAID configuration
- same drawbacks as in RAID0
- sw/hw complexity as in RAID0
- usable with odd drives without wasting space
- easier disaster recovery than RAID0

not on
the book

comparison

- from www.pcguide.com

RAID Level	Number of Disks	Capacity	Storage Efficiency	Fault Tolerance	Availability	Random Read Perf	Random Write Perf	Sequential Read Perf	Sequential Write Perf	Cost
0	2,3,4,...	S*N	100%	none	★	★★★★	★★★★	★★★★½	★★★★	$
1	2	S*N/2	50%	★★★★	★★★★	★★★	★★★	★★	★★★	$$
2	many	varies, large	~ 70-80%	★★	★★★★	★★	★	★★★★	★★½	$$$$$
3	3,4,5,...	S*(N-1)	(N-1)/N	★★★	★★★★	★★★	★	★★★★	★★½	$$
4	3,4,5,...	S*(N-1)	(N-1)/N	★★★	★★★★	★★★★	★½	★★★	★★	$$
5	3,4,5,...	S*(N-1)	(N-1)/N	★★★	★★★★	★★★★½	★★	★★★½	★★½	$$
6	4,5,6,...	S*(N-2)	(N-2)/N	★★★★½	★★★★★	★★★★½	★	★★★½	★★	$$$
7	varies	varies	varies	★★★	★★★★	★★★★½	★★★★	★★★★½	★★★★	$$$$$
01/10	4,6,8,...	S*N/2	50%	★★★★	★★★★½	★★★★½	★★★½	★★★★½	★★★½	$$$
03/30	6,8,9,10,...	S*N0*(N3-1)	(N3-1)/N3	★★★½	★★★★	★★★★	★★	★★★★½	★★★	$$$$
05/50	6,8,9,10,...	S*N0*(N5-1)	(N5-1)/N5	★★★½	★★★★	★★★★½	★★★	★★★★	★★★	$$$$
15/51	6,8,10,...	S*((N/2)-1)	((N/2)-1)/N	★★★★★	★★★★★	★★★★	★★★	★★★★	★★★	$$$$$

not on
the book

implementation

- software (in the OS)
  - no need for special drivers
  - may be inefficient for parity computation
  - Linux: (0, 1, 4, 5, and any kind of nesting)
  - Windows: XP (0,jbod), 2000 Server (0,1,5, jbod)
- hardware (in the controller)
  - need special drivers/software
  - efficient when parity should be computed
- hybrid (the bios drives the controller)
  - need special drivers
  - inefficient when parity computation is required