SUPSI

Storage devices

Operating Systems

Amos Brocco, Lecturer & Researcher

14 settembre 2018

Objectives

- Understand how data is stored on disk
- Understand the differences between disk drives and SSD

Browsing
- Get a rapid overview.

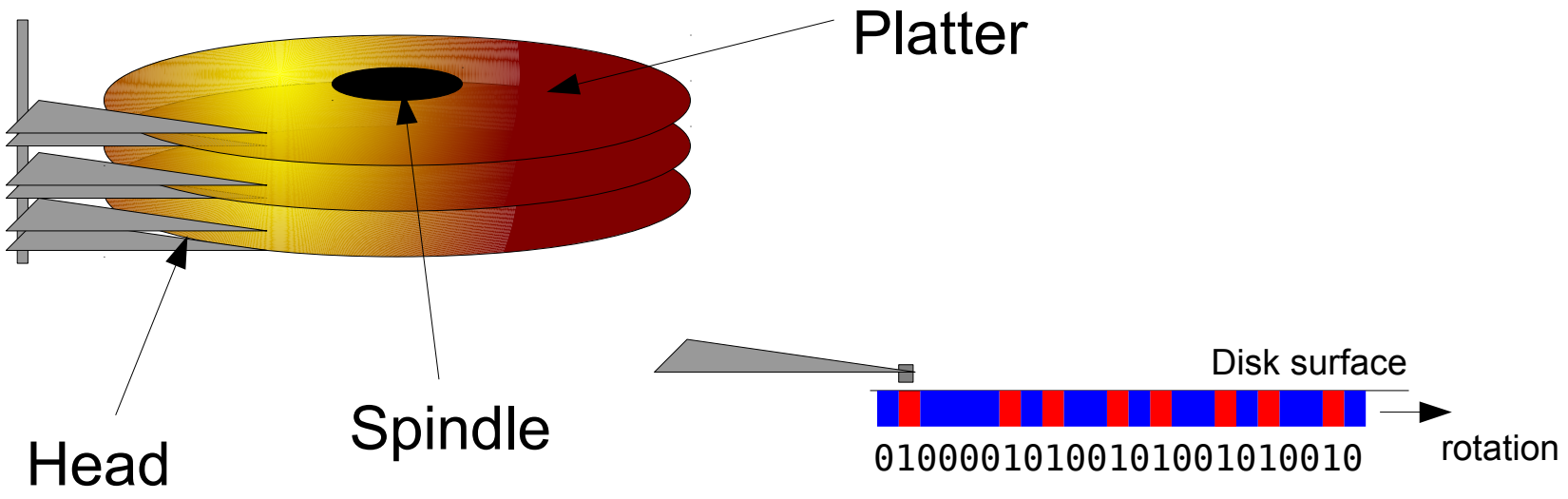
Reading
- Read it and try to understand the concepts.

Studying
- Read in depth, understand the concepts as well as the principles behind the concepts.

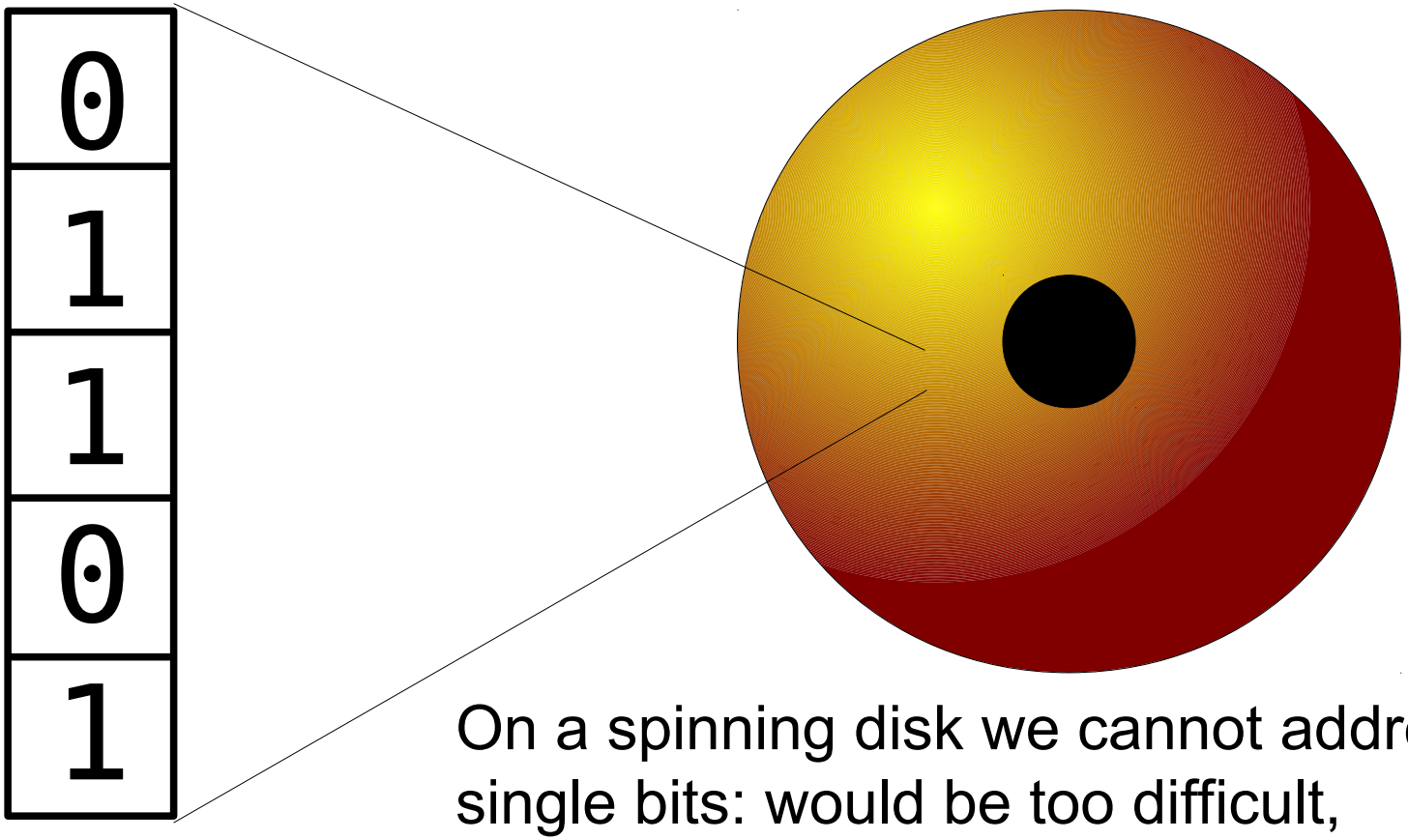
You are also encouraged to try out (compile and run) code examples!

The hard disk drive

- A disk drive consists of a set of **platters** which are bound around a **spindle**
  - the surface of the platter is coated with a magnetic layer which can permanently store bits
  - the disk spins at a constant rate (for example, 7200 RPM)

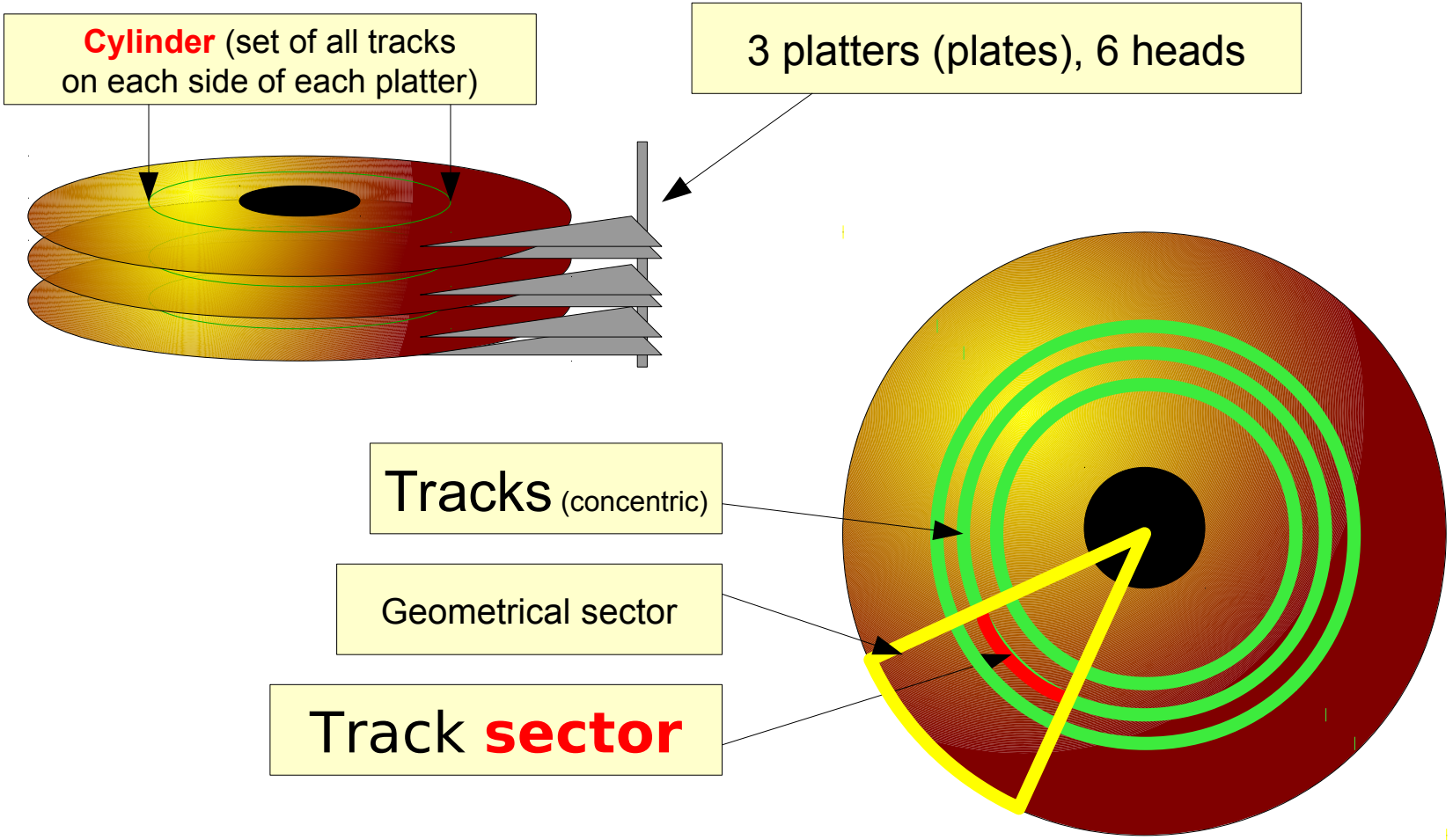


Addressing on the disk



On a spinning disk we cannot address single bits: would be too difficult, inefficient (who reads/write single bits anyway?) and would require a very large address

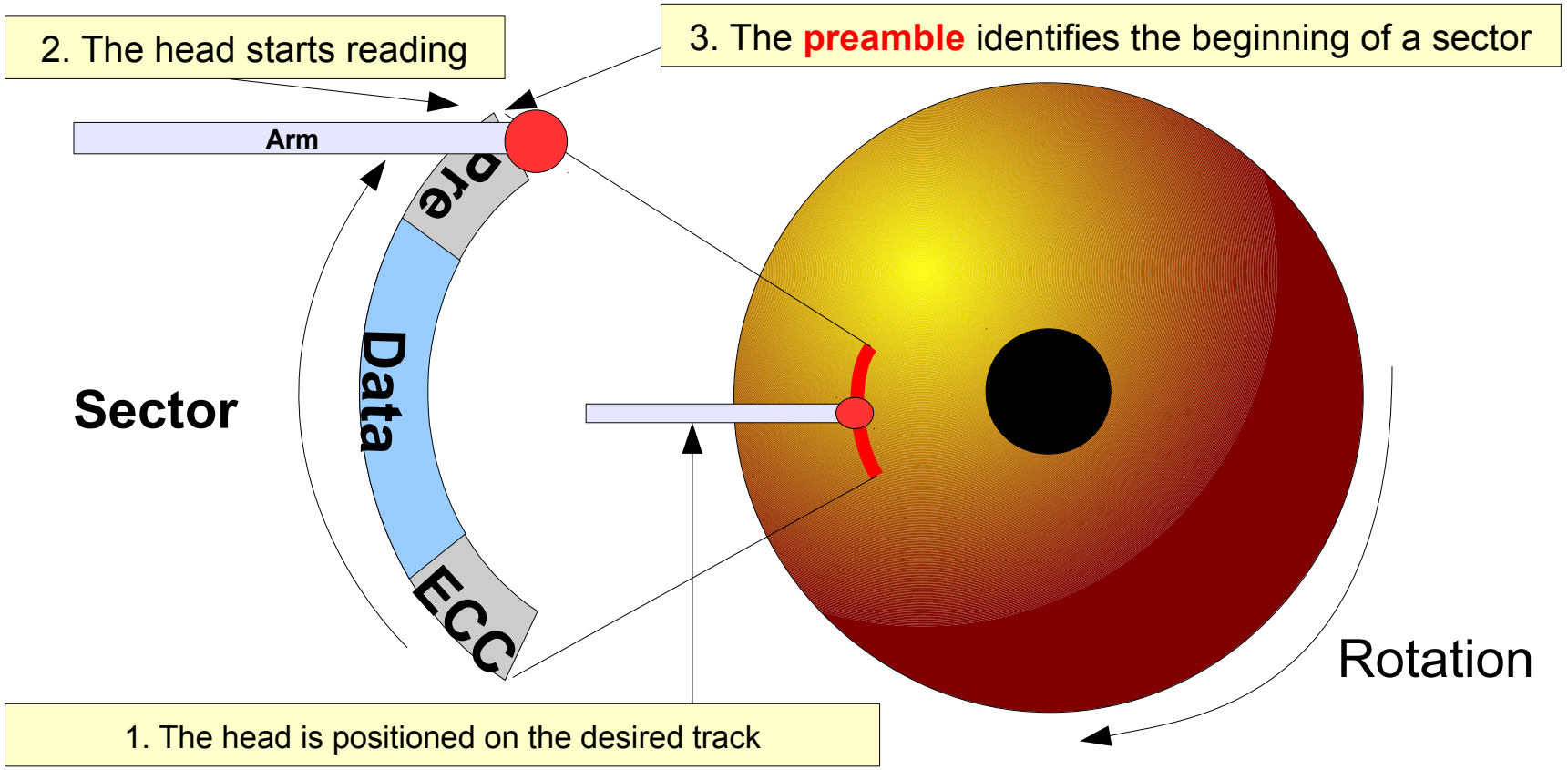
Cylinders, tracks and sectors



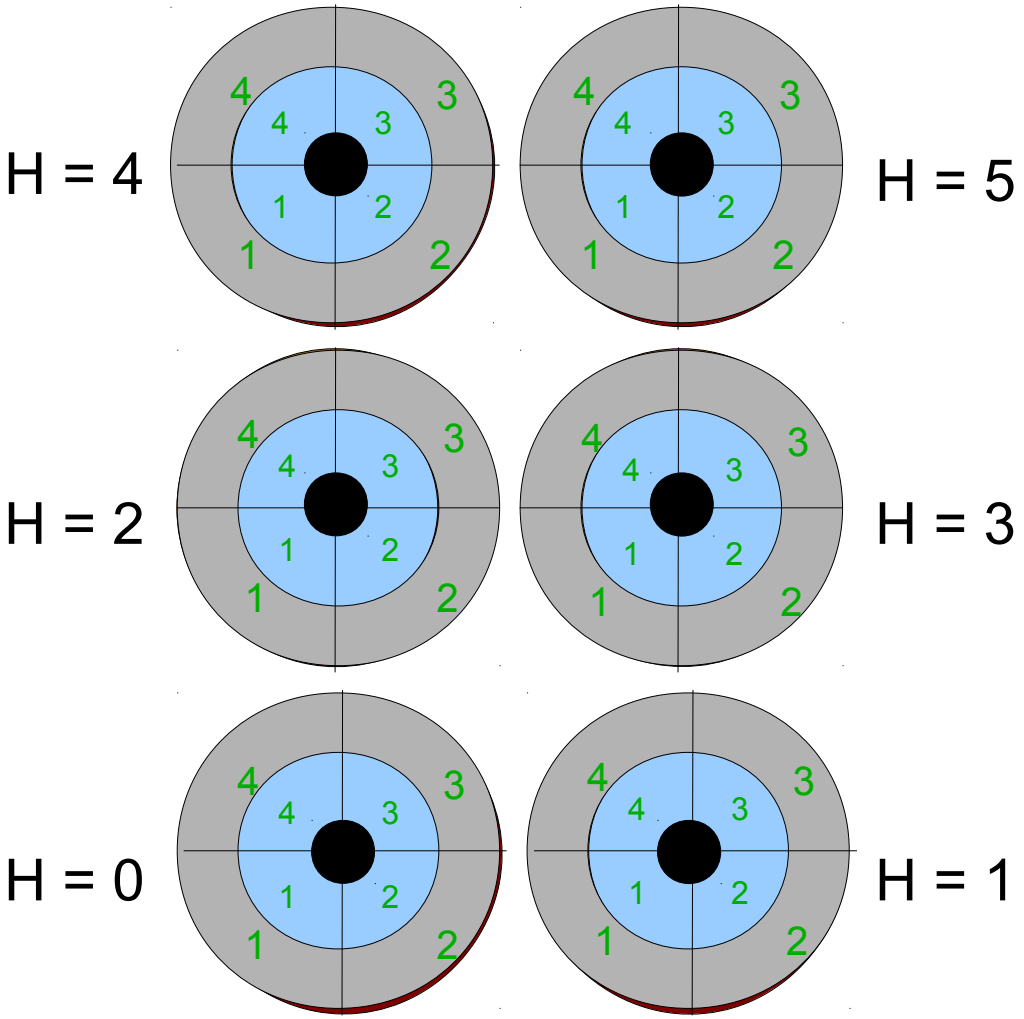
Reading and writing sectors

Sector: smallest storage unit which can be addressed and written atomically (typically 512B or 4096B)

Cluster: set of contiguous sectors



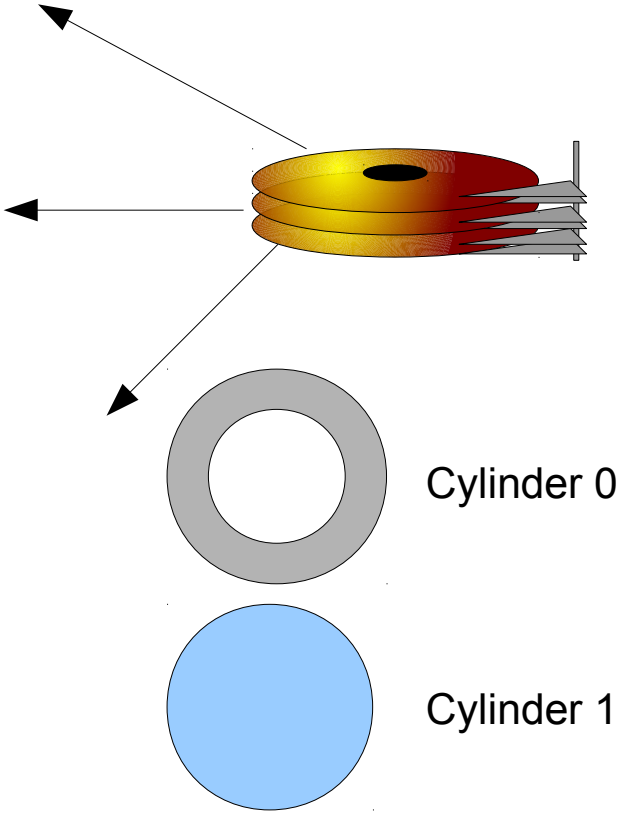
Addressing sectors: CHS (Cylinder-head-sector)



Note: sector number starts at 1

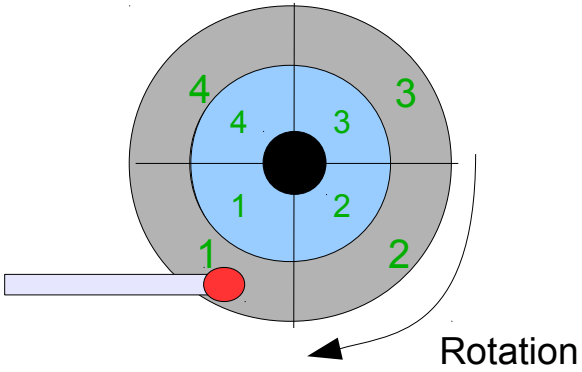
Disk geometry:

6 heads
2 cylinders
4 sectors per track

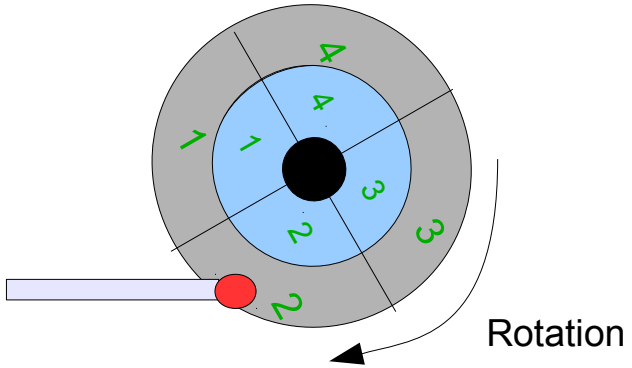


Latency on a single-track: rotational delay

- Reading / Writing different sectors on the same track of a spinning disk incurs in a **rotational delay**:
  - in the worst case we must wait for a complete rotation



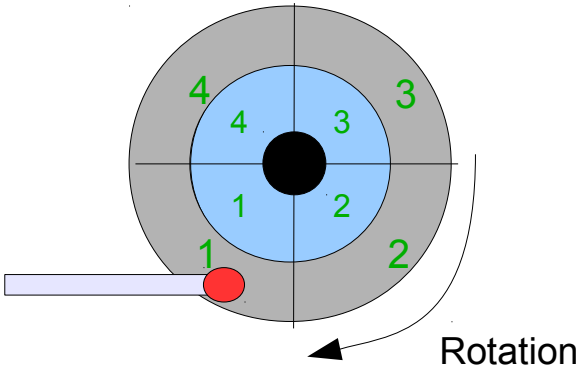
Request to read sector (0, 1, 1)



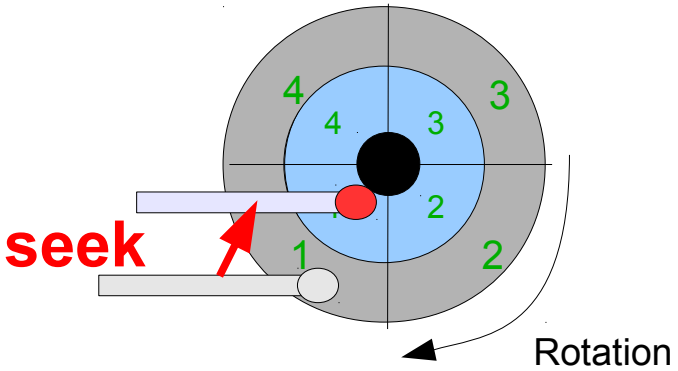
Request to read sector (0, 1, 4)

Latency when moving between tracks: seek time

- Reading / Writing on different tracks requires a **seek** (which takes some... **seek time**)
- The **settling time** (time before we can successfully access the new track) is influenced by:
  - the **acceleration time** (the head starts moving)
  - the **coasting time** (the head moves)
  - the **deceleration time** (the head slows down over the target track)



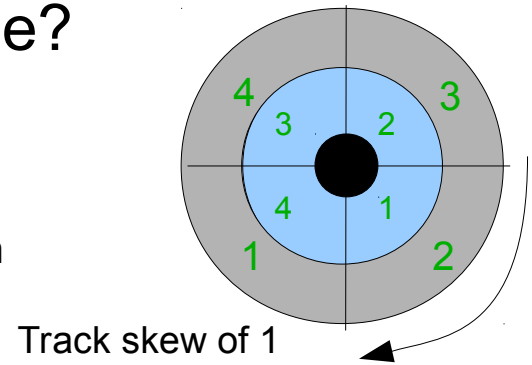
Request to read sector (0, 1, 1)



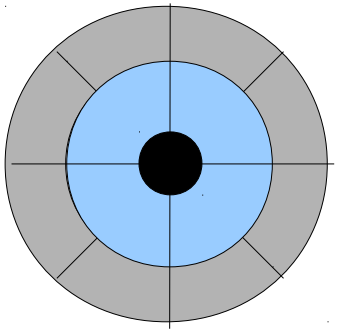
Request to read sector (1, 1, 1)

Optimizing I/O Time

- When reading/writing on a disk the total I/O time is:
  - **rotational delay** + **seek time** + **transfer time**
- How can we do more in (possibly) less time?
  - implement **track skew** (interleaving)
    - sectors on different tracks are “not aligned” (on purpose)
    - while reading sectors sequentially, reduce the delay when jumping from one track to the other
  - do **block** transfers, use **DMA**
    - the OS must know the optimal block size or how to program the DMA
  - implement **caches** / buffers (HW or SW)
    - read more data than requested
    - store data temporarily during writes
  - use multiple sector densities (**multi-zones**)
    - outer cylinders have “more room”



Track skew of 1

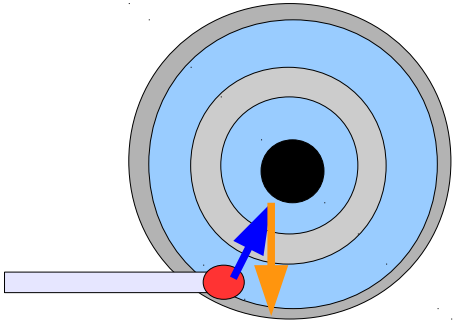


Operating system optimizations

- The operating system can also improve I/O latency
  - by deciding the order of I/O operations using a **disk scheduler** (I/O scheduler)
    - First-come, First-served
    - Shortest Seek Time First (SSTF)
      - the queue of I/O requests is ordered in order to serve requests for the nearest track first
      - *Issues: can lead to starvation*
    - Elevator (SCAN, F-SCAN, C-SCAN)
    - LOOK / C-LOOK
    - Shortest Positioning Time First (SPTF)
      - takes into account rotational delay

Elevator

- The **elevator** algorithm (**SCAN**) processes requests while sweeping the disk
- Move head from one end of the disk to the other servicing requests along the way



sweep (outer-to-inner)

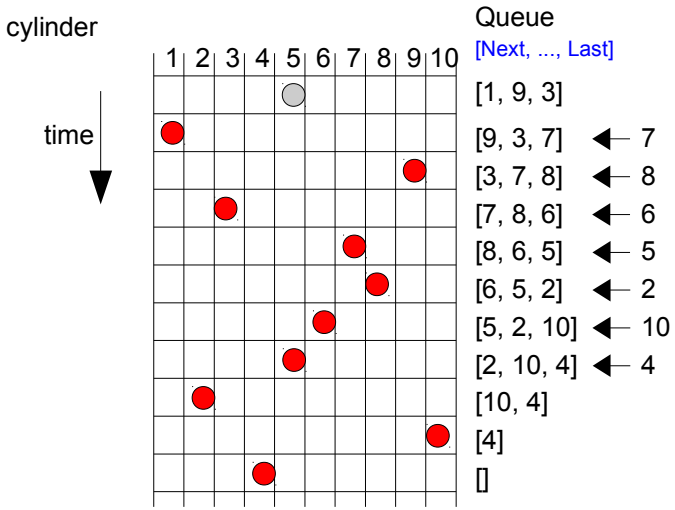
sweep (inner-to-outer)

- **F-SCAN**: delay incoming requests while sweeping across the disk
- **C-SCAN** (Circular SCAN): only travel in one direction (jump back when reaching an edge)
- **LOOK**: reverse direction if the are no requests ahead in the current direction

Comparison (1)

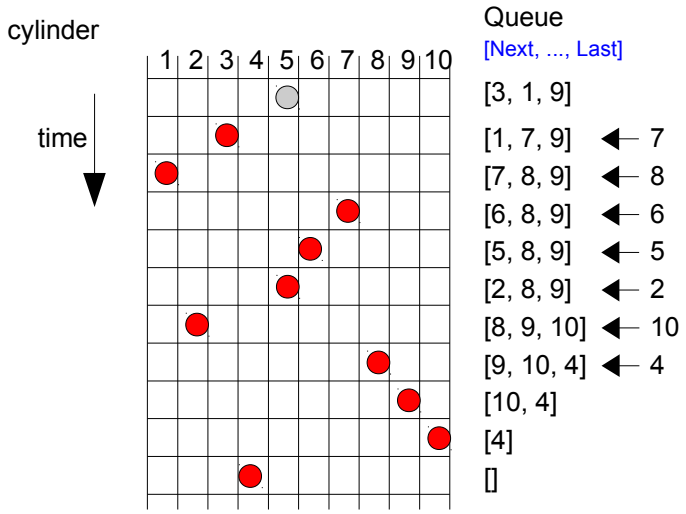
- We consider 10 cylinders, the head is initially at cylinder 5
- Request order: 1, 9, 3, 7, 8, 6, 5, 2, 10, 4
  - we assume that the queue initially contains [1, 9, 3] and has a maximum depth of 3 requests, other requests arrive as soon as an existing one has been served

FCFS



total seek time: 43ms
(based on an average 1ms per cylinder)

SSTF



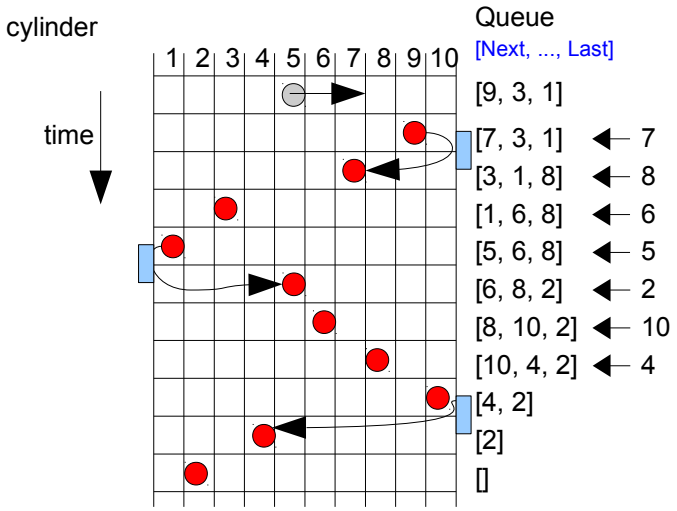
total seek time: 29ms
(based on an average 1ms per cylinder)

...but requests for cylinders 8, 9 have to wait for a long time!

Comparison (2)

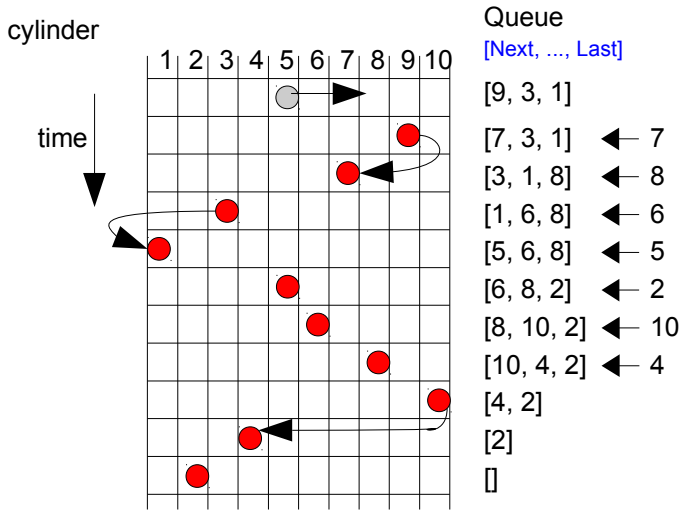
- We consider 10 cylinders, the head is initially at cylinder 5
- Request order: 1, 9, 3, 7, 8, 6, 5, 2, 10, 4
  - we assume that the queue initially contains [1, 9, 3] and has a maximum depth of 3 requests, other requests arrive as soon as an existing one has been served

SCAN



total seek time: 31ms
(based on an average 1ms per cylinder)

LOOK



total seek time: 29ms
(based on an average 1ms per cylinder)

What about solid state drives?

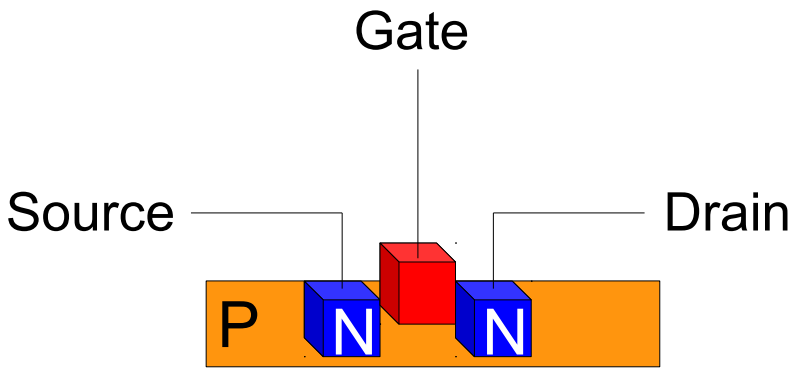
Solid state drives (EEPROM, flash) consists in an array of **memory cells**

- they do not have any mechanical part...
- ... should the operating system care?

- **Yes**!
  - previous I/O scheduling algorithms are useless (since there is no seek time / rotational delay)
  - ... other problems arise...

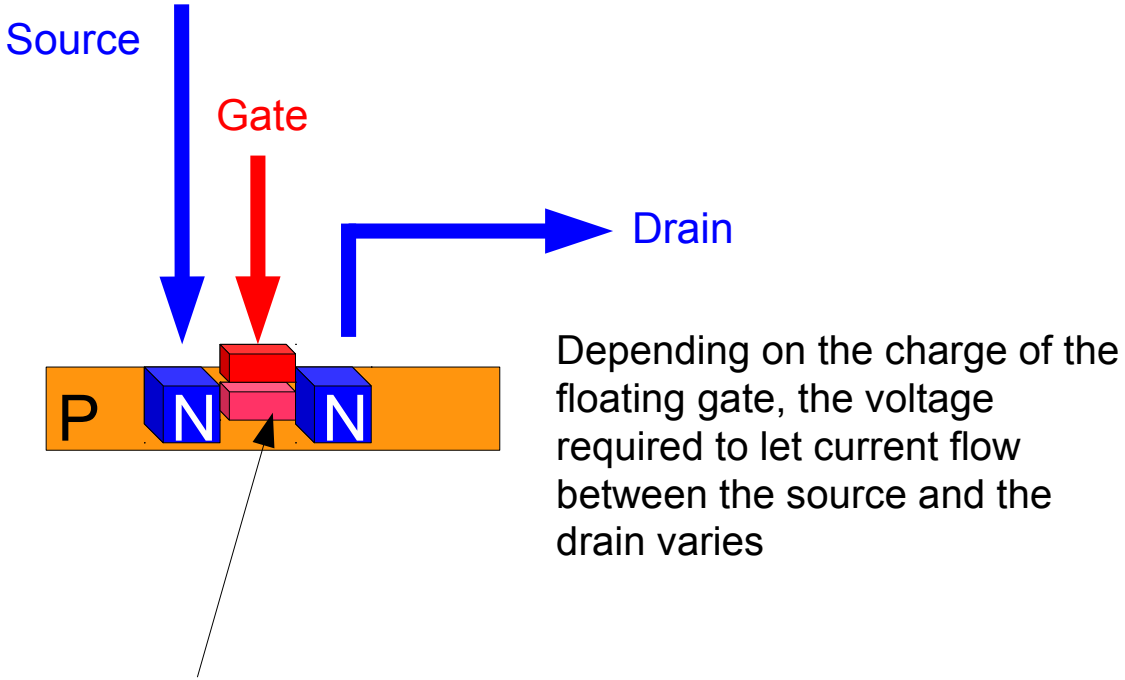
At the core: the Field-Effect Transistor (FET)

- "Digital switch"
  - 3 pins: **source** , **drain** , **gate**
  - The voltage that is applied to the input gate controls the current flowing between the source and the drain:
    - the output current is proportional to the input voltage
    - current starts to flow then a voltage greater than a **threshold** is applied to the gate



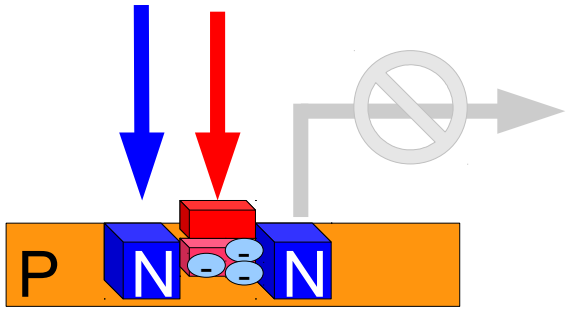
Floating gate transistor

- At the core of a solid-state drives (SSD) we typically have **memory cells** based on **floating gate transistors** which use electrons to store data

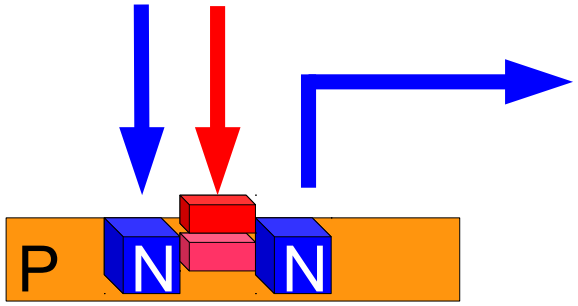


Storing bits in the floating gate

- When the floating gate is negatively charged the threshold voltage required to have current flowing between the source and the drain increases (the **cell is programmed**)

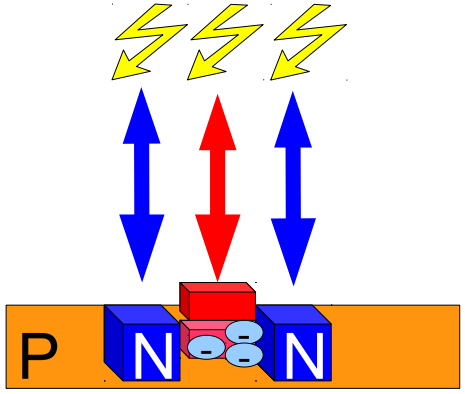


- When the floating gate is not charged (neutral) the transistor operates as normal (the **cell is erased**)



Programming and erasing a cell

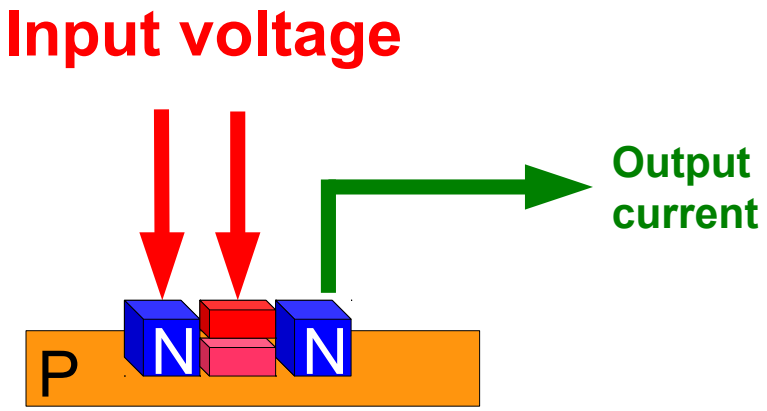
- Programming (*putting an electron charge in the floating gate*) and erasing (*removing an electron charge from the floating gate*) a cell is achieved by applying high voltages to the source, drain, and gate pins.
  - these operations stress the cell and cause some **wear**

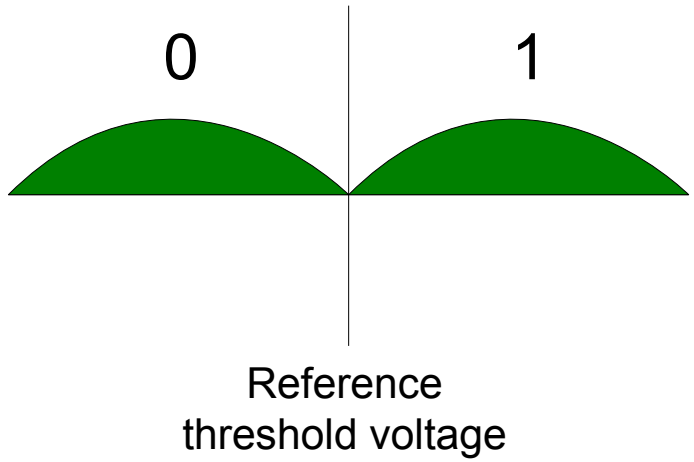


How many bits can we store in a cell? SLC vs MLC vs TLC

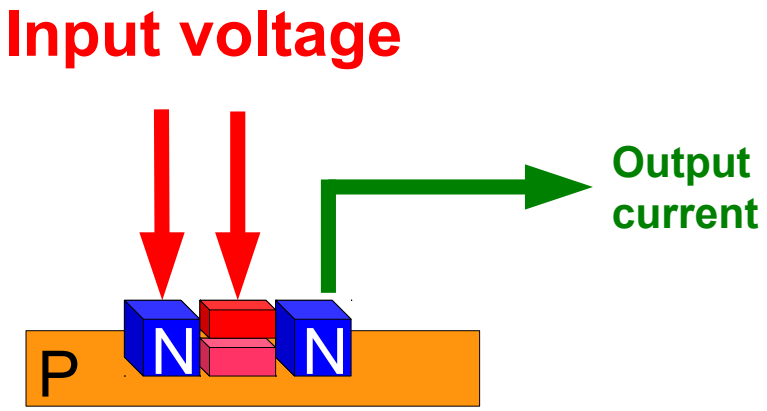
- **SLC (Single Level Cell)**: each cell can only store 1 bit
  - 2 charge states = 1 possible voltage threshold
  - Reliable, but expensive
- **MLC (Multi Level Cell)**: each cell can store 2 bits
  - 4 charge states = 3 possible voltage thresholds
  - Higher densities, lower cost-per-bit
- **TLC (Triple Level Cell)**: each cell can store 3 bits
  - 8 charge states = 7 possible voltage thresholds
  - Higher densities, lower cost-per-bit

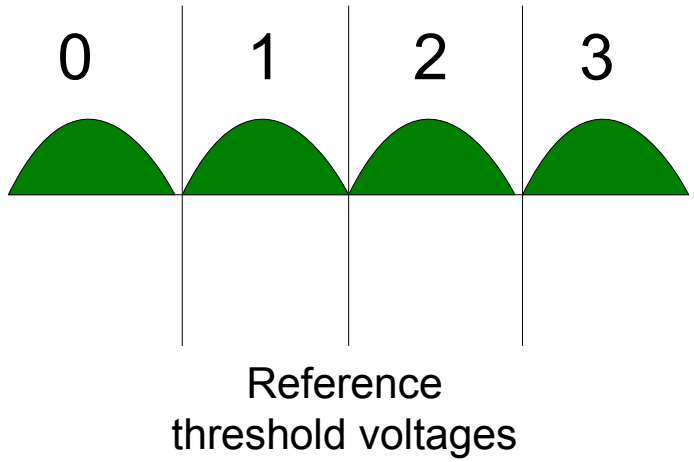
SLC (Single Level Cell)



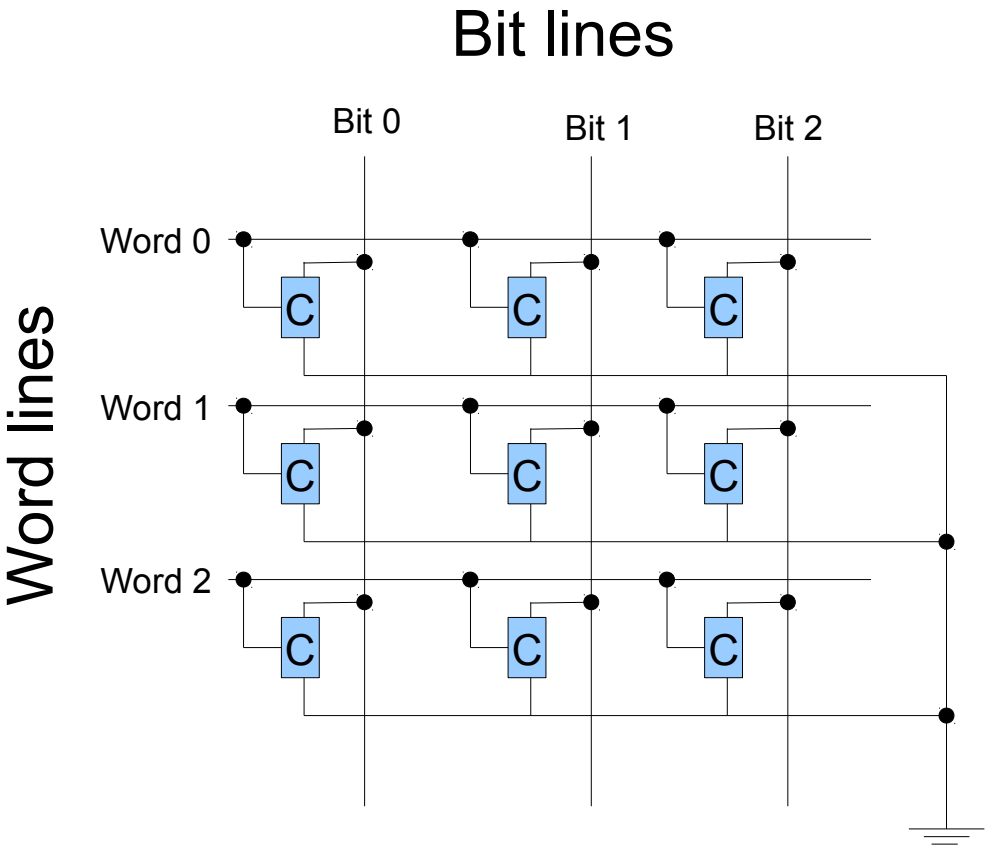


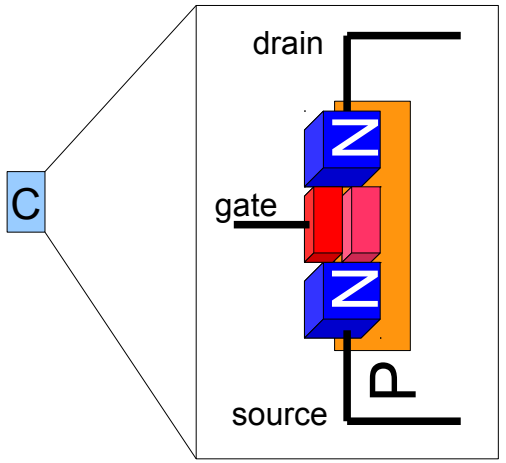
MLC (Multi Level Cell)





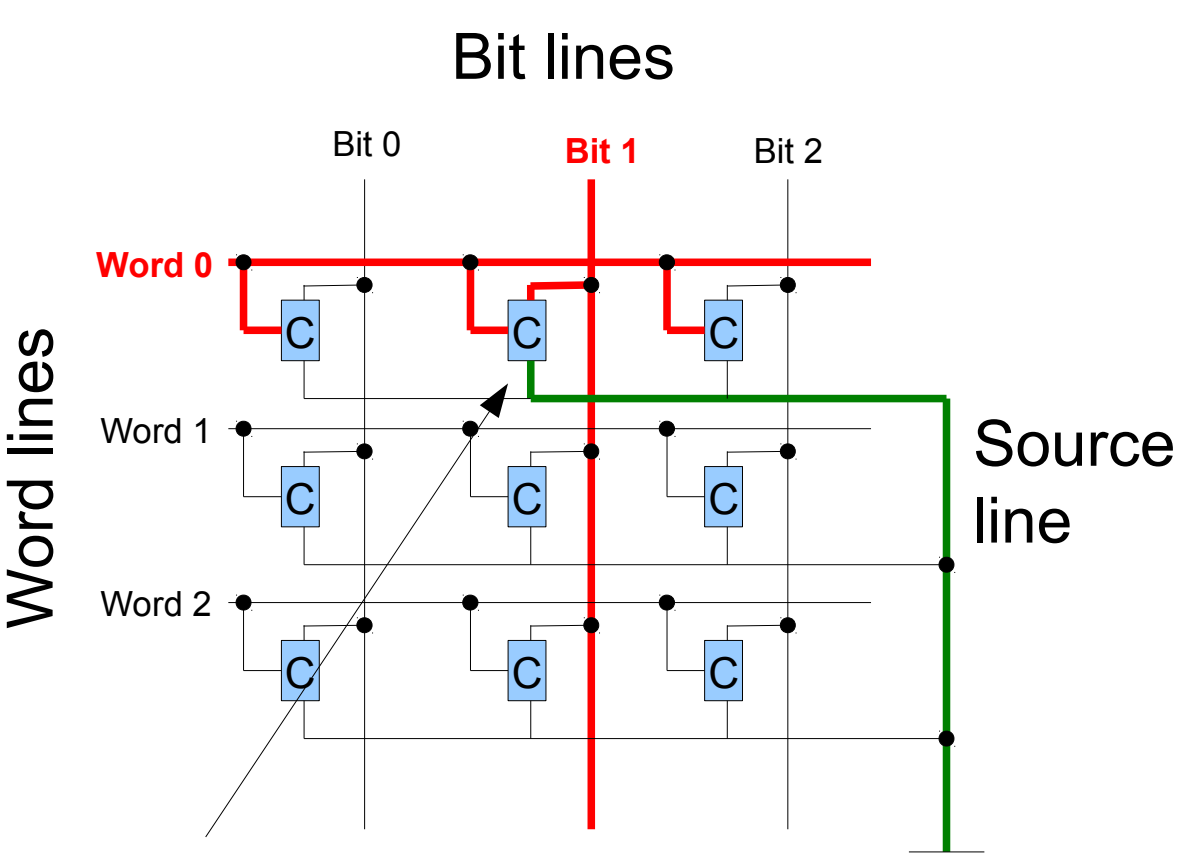
Cell organization: NOR

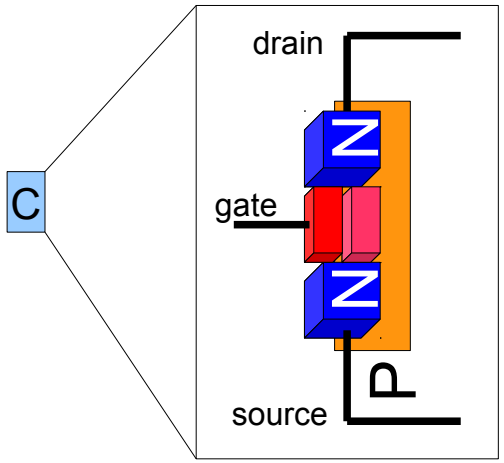




NOR memory provides **true random access** since each cell can be addressed independently, but takes more place on the chip (many address lines)

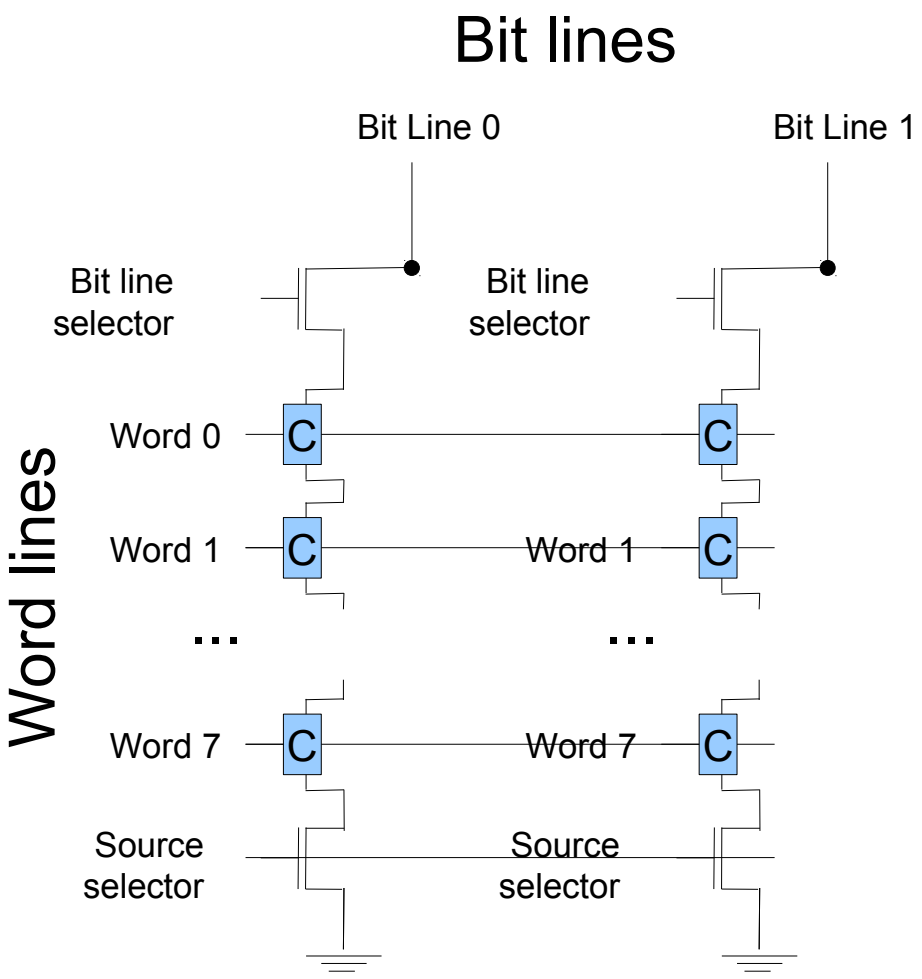
Cell organization: NOR (Read operation)

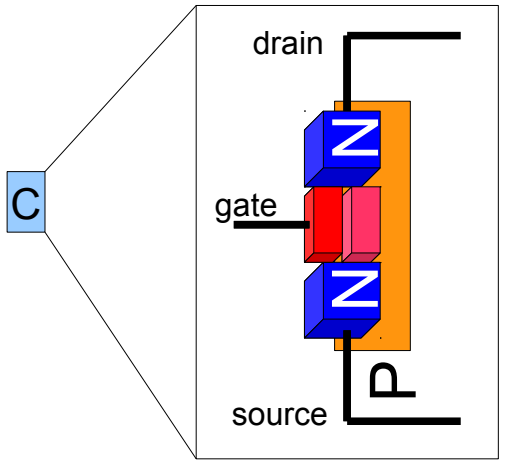




The word and bit lines corresponding to the cell to be read are put high. If the cell is erased (floating gate not charged), the bit line is pulled down to ground → **we read 0**, otherwise the bit line remains high → **we read 1**

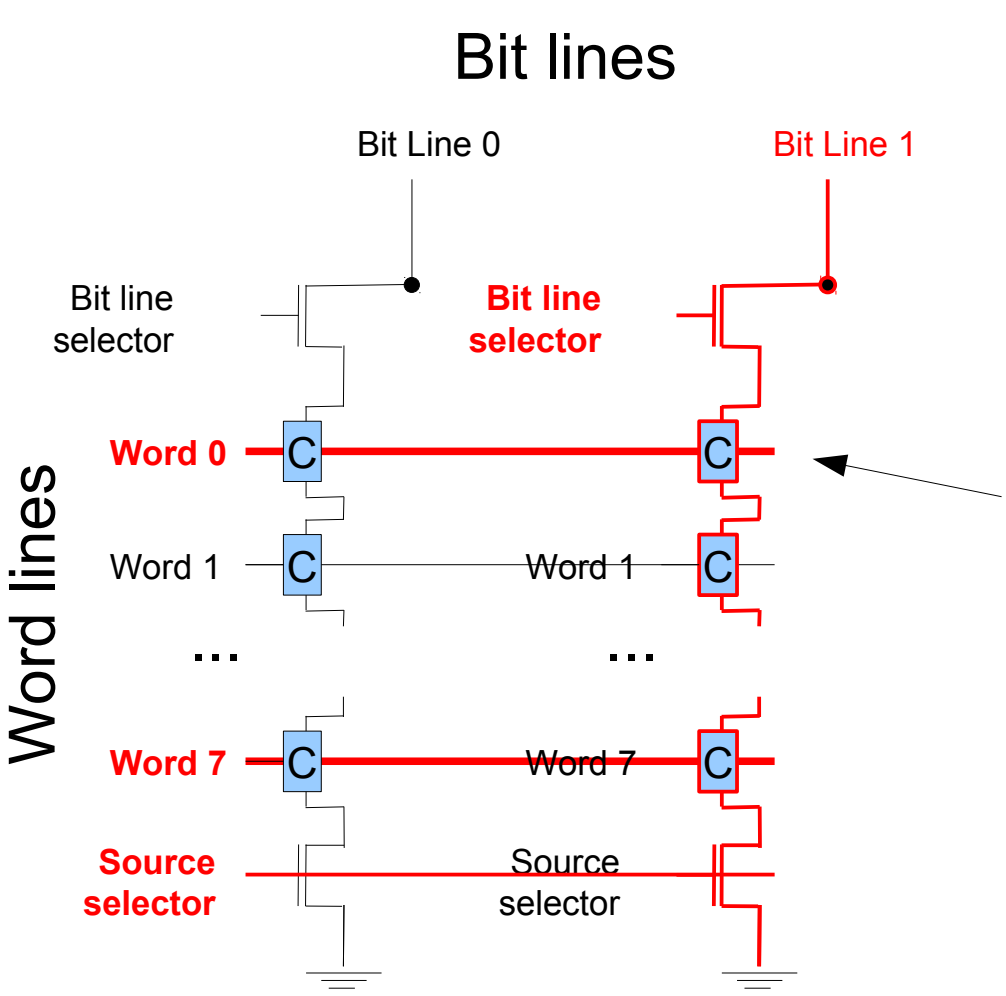
Cell organization: NAND

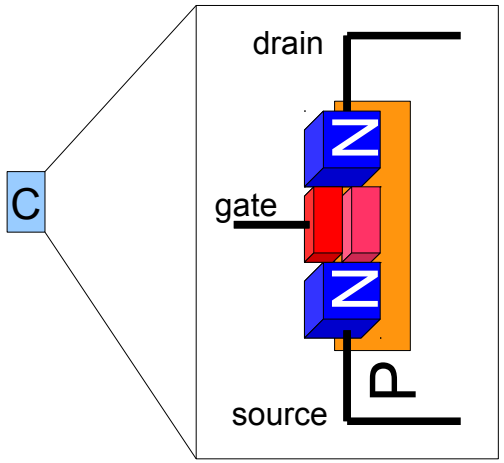




With NAND we trade true random access for **lower complexity** (and manufacturing costs). Moreover, to read/write one cell we must enable many of them.

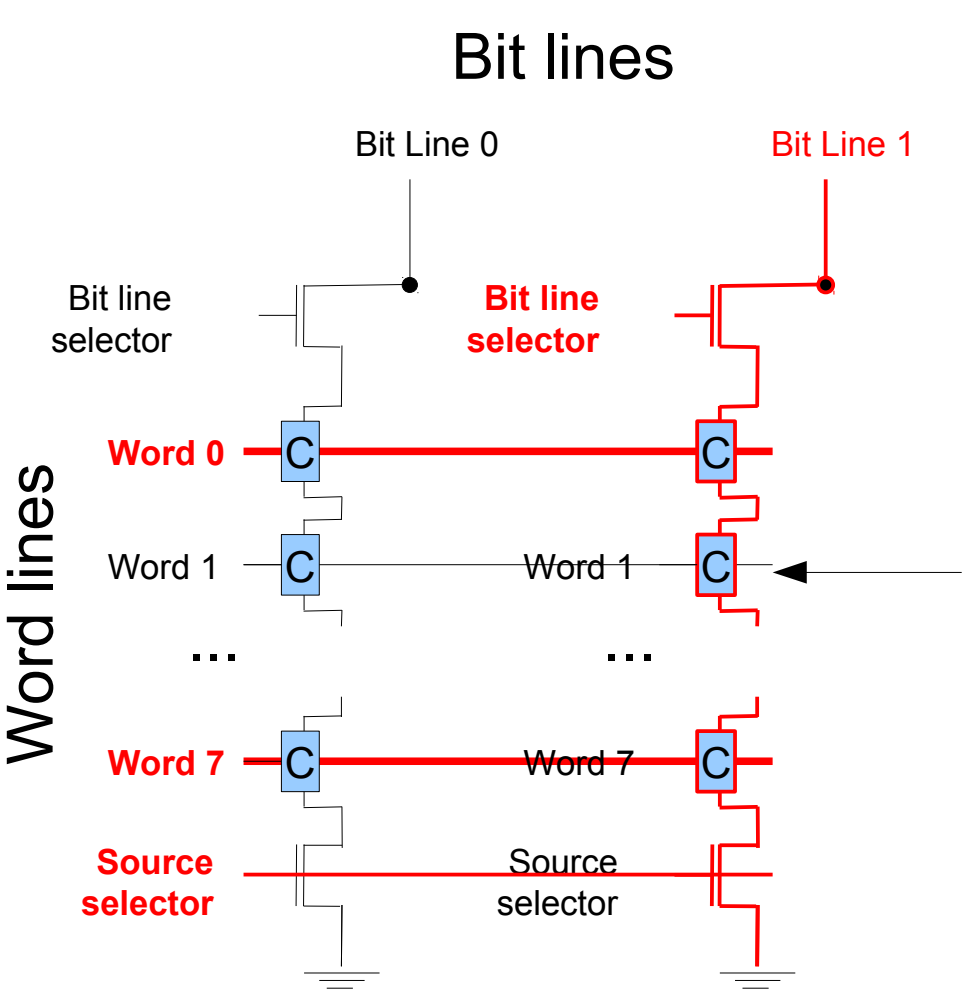
Cell organization: NAND (Read operation)

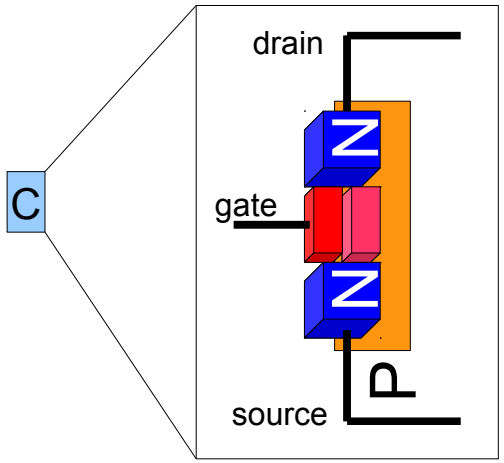




1. The bit line corresponding to the cell to be read is put high. We apply a high voltage (>> threshold) to to all word lines but the one to be read (floating gate transistors with high gate voltage let current flow between source and drain regardless of the charge)

Cell organization: NAND (Read operation)





2. On the word line of the cell that we want to read we put a threshold voltage: if the cell is erased it will pull down the bit line to ground (→ **we read 0**) otherwise it will stay high (→ **we read 1**)

Addressing

- Memory cells are organized in **pages**, several pages create a **block**
  - For NOR memory, reading and programming (writing) are random-access, **while erasing is block-wise**
  - For NAND memory, reading and programming are page-wise, **while erasing is block-wise**

Addressing: CHS does not work very well with SSDs

Disk-device dependent addressing:

CHS addressing (Cylinder-head-sector)

What about SSD?

Works on disk based devices...

Depends on the geometry of the disk...

Large drives?

Abstraction!

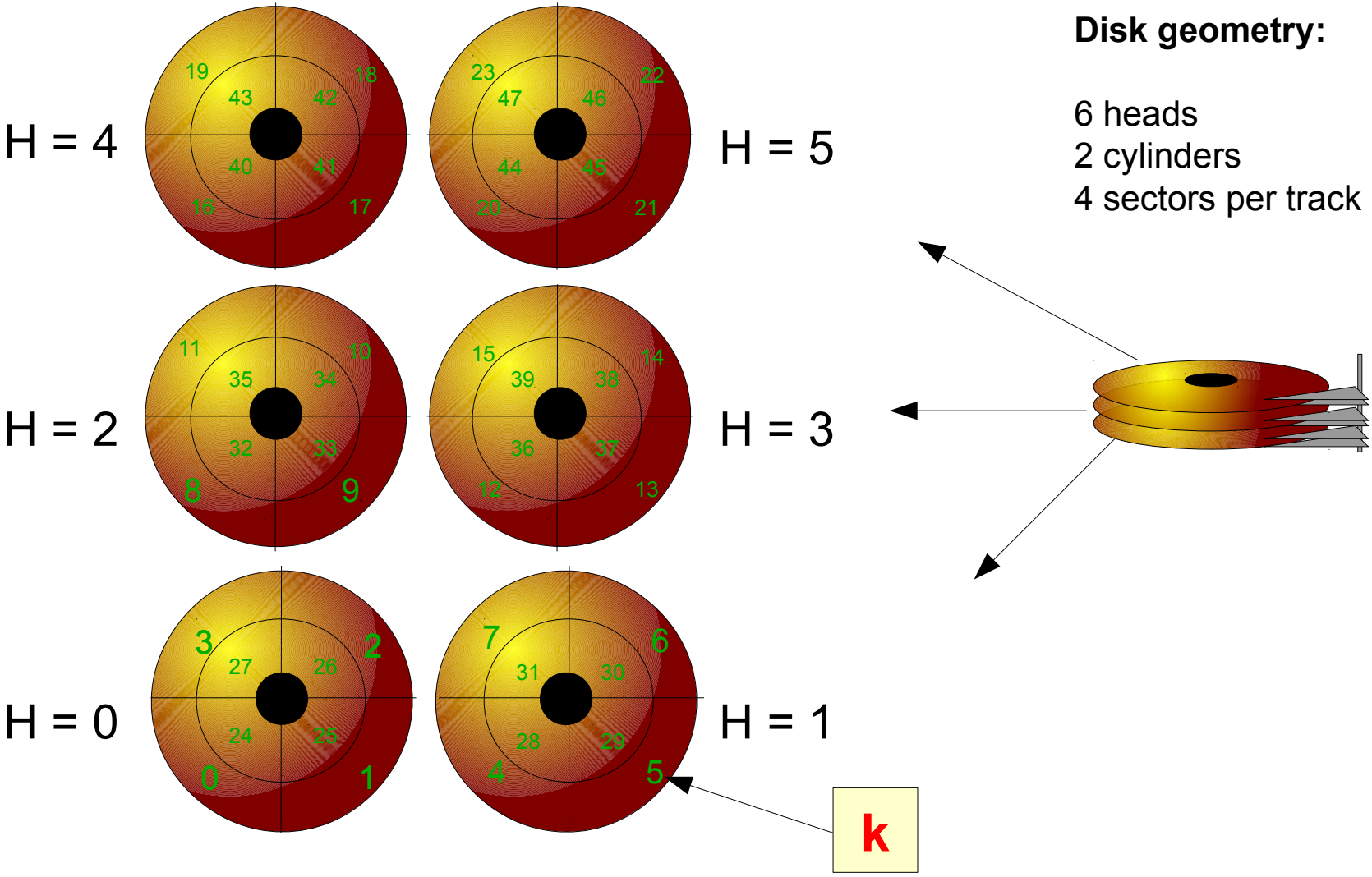
Consider all storage devices as equal, as capable of reading or writing a blocks of data → **block devices**

Use device-independent addressing:

LBA (Logical Block Address):

sectors/blocks are numbered in a sequential order

LBA (Logical Block Address): hard-disk example



CHS → LBA

$$k_{LBA} = \mathbf{C} * Sectors_{Cylinder} + \mathbf{H} * Sectors_{Track} + \mathbf{S} - 1$$

Sectors for each "full" cylinder

Sectors for each "full" track in the last cylinder

Remaining sectors

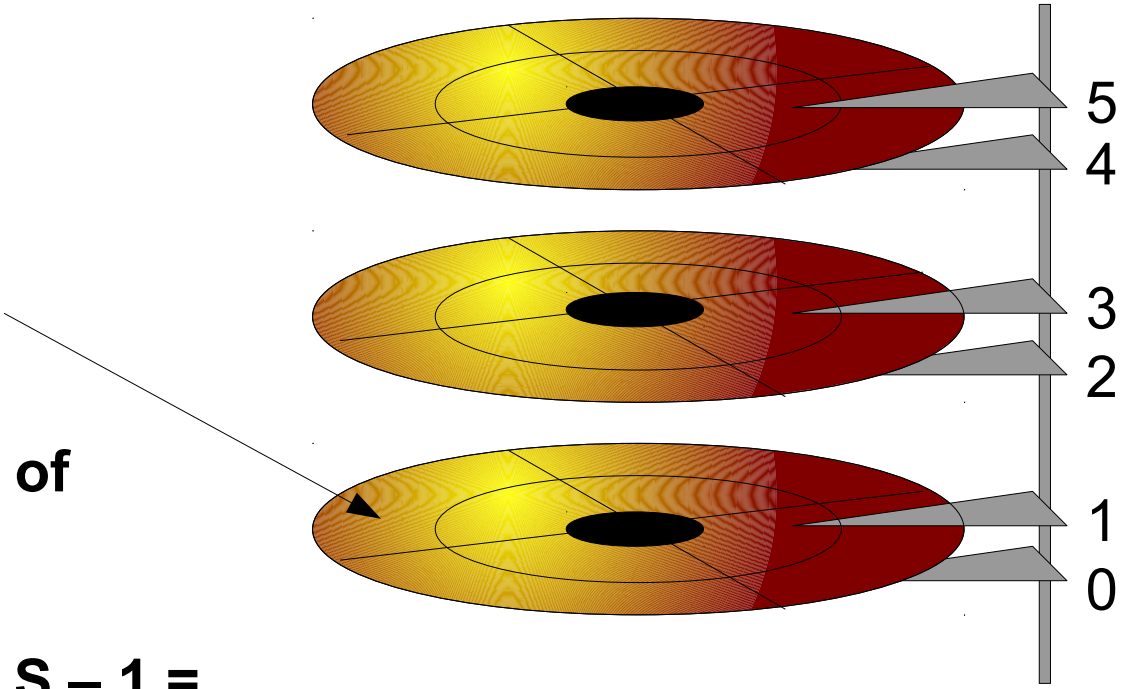
$$C * N_{heads} * Sectors_{Track}$$

$$k_{LBA} = ((\mathbf{C} * N_{heads} + \mathbf{H}) * Sectors_{Track}) + \mathbf{S} - 1$$

Example

Disk geometry:

6 heads
2 cylinders
4 sectors per track



Compute LBA address of CHS (0,1,3)

LBA = C * 24 + H * 4 + S – 1 =

= 0 * 24 + 1 * 4 + 3 – 1 = 6

LBA → CHS

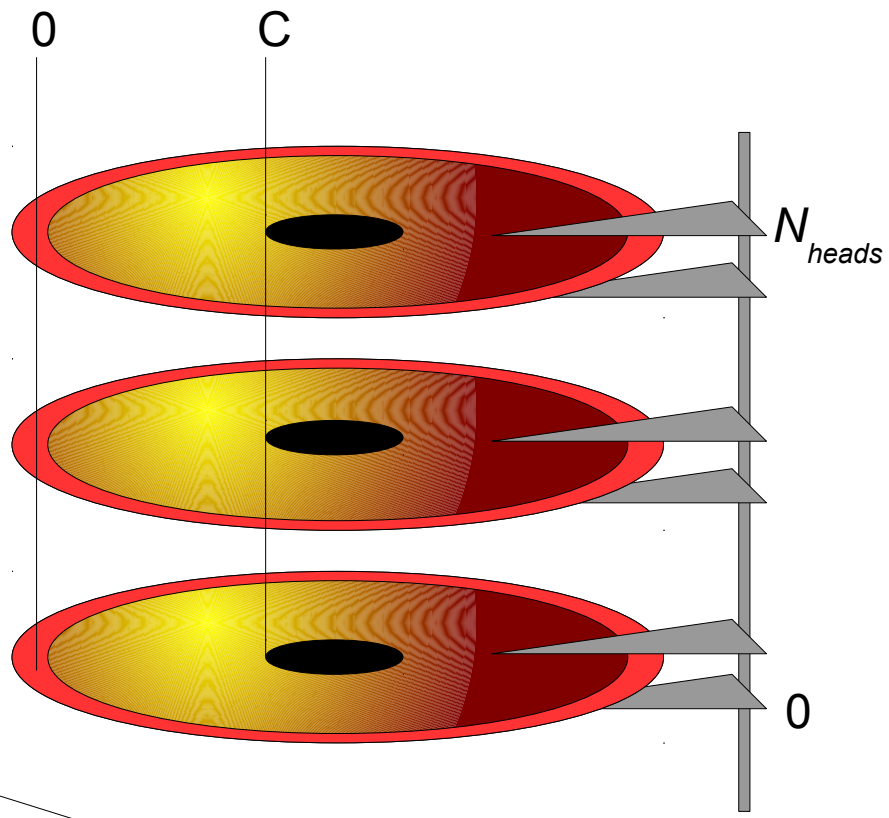
$Sectors_{Cylinder} = (N_{heads} * Sectors_{Track})$

Cylinder = LBA / $Sectors_{Cylinder}$

R = LBA % $Sectors_{Cylinder}$

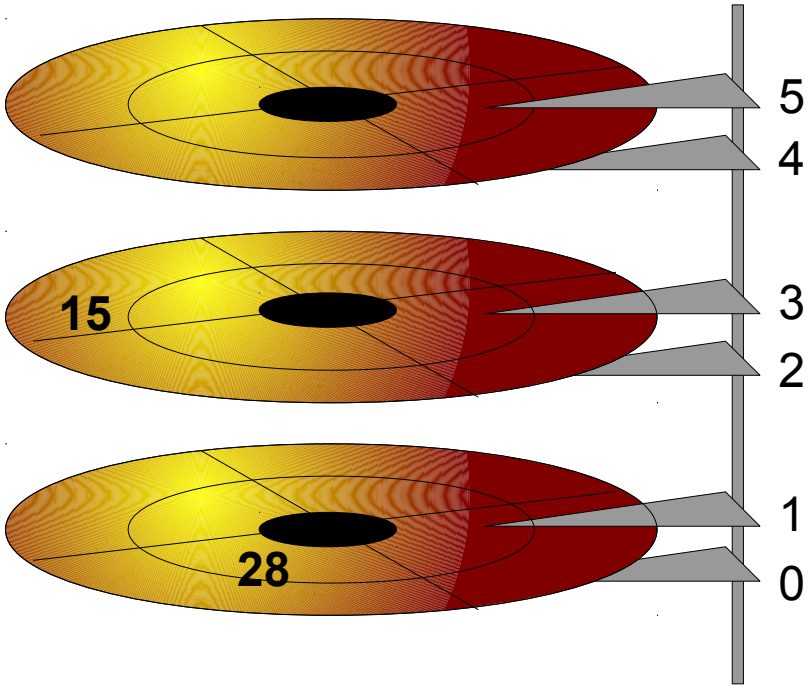
Head = R / $Sectors_{Track}$

Sector = R % $Sectors_{Track}$ +1



+ 1 because with LBA blocks are numbered starting from 0

Example



Disk geometry

6 heads
2 cylinders
4 sectors per track

Compute CHS for LBA k = 15

C = LBA / Sectors$_{Cylinder}$ = 15 / (6 * 4) = 0
R = LBA % Sectors$_{Cylinder}$ = 15 % 24 = 15
H = R / Sectors$_{Track}$ = 15 / 4 = 3
S = R % Sectors$_{Track}$ +1 = 15 % 4 + 1 = 4

Compute CHS for LBA k = 28

C = LBA / Sectors$_{Cylinder}$ = 28 / (6 * 4) = 1
R = LBA % Sectors$_{Cylinder}$ = 28 % 24 = 4
H = R / Sectors$_{Track}$ = 1 / 4 = 1
S = R % Sectors$_{Track}$ +1 = 4 % 4 + 1 = 1

Why should the operating system care about SSDs?

- SSDs suffer from two problems:
  - **erase-before-write**, namely flash memory must be erased before it can be rewritten
  - **write endurance**, each cell can only be erased a limited number of time before failing → **wear out**

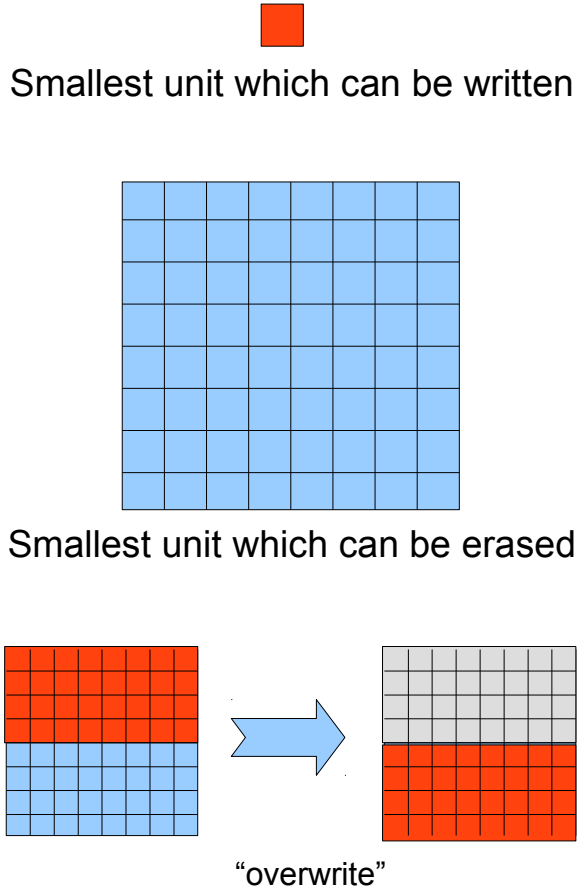
- The operation of an SSD also introduces **write amplification** , namely the actual amount of physical information written is a multiple of the logical amount intended to be written.

Why should the operating system care about SSDs?

- The operating system can implement techniques to reduce wear:
  - **wear leveling** : avoid writing on the same blocks over and over again
  - limit “useless” writes (for example, by keeping log data in-memory or by skipping frequent meta-data updates)
  - support **TRIM**

Garbage collection

- An SSD can only erase cells in blocks, which are larger than a page
  - For example, while the SSD can write 4KiB it can only erase 256KiB at a time
- Since cells cannot be erased/ overwritten, if we overwrite the data the block is filled with the new data and old pages are marked as **stale**



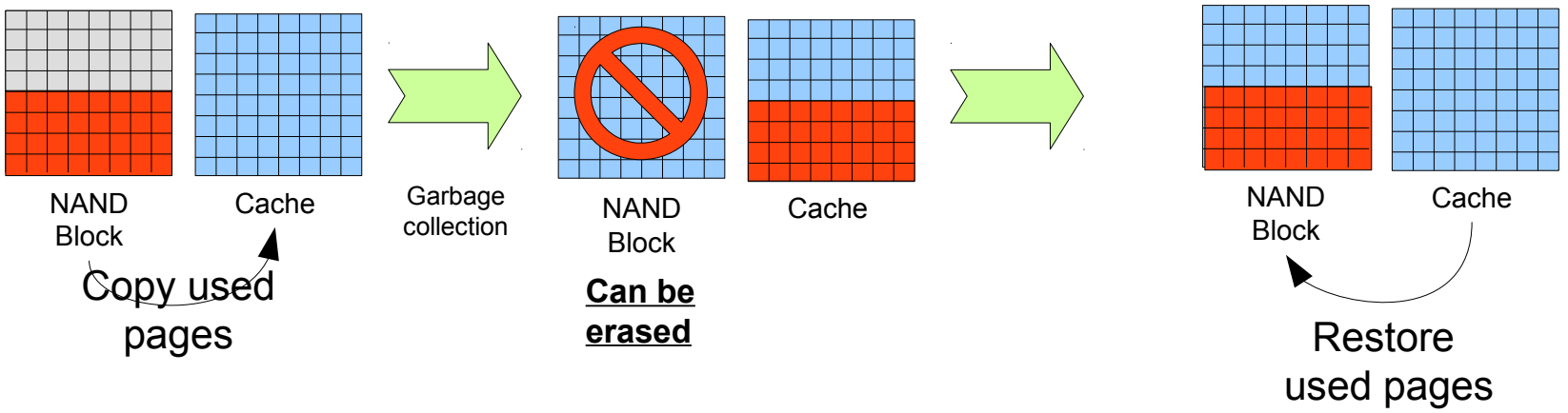
Free page

Used page

Stale page

Garbage collection

- The SSD then performs **garbage collection** to “save” used pages and consequently erase whole blocks



- ... OK, but some pages might actually have been freed by the OS (for example, from a deleted file) but are not marked as stale by the SSD... How can we prevent saving and restoring them?

Filesystem-aware garbage collection

- The operating system knows which block can be freed, and can issue **TRIM** commands to the disk to let it know about these blocks
  - the SSD can then avoid copying stale pages which would otherwise generate additional wear