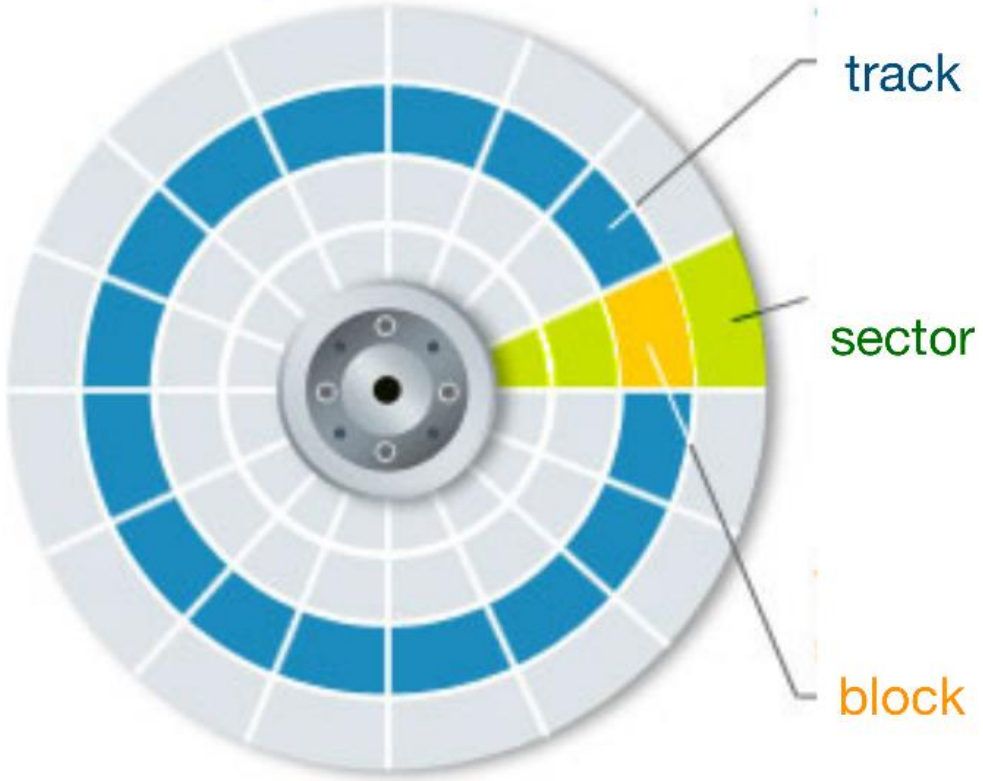
Algoritmen & Datastructuren 3

WPO - H14 - Extern Geheugen: Basisconcepten
Disk & Block ADT

Secondary storage



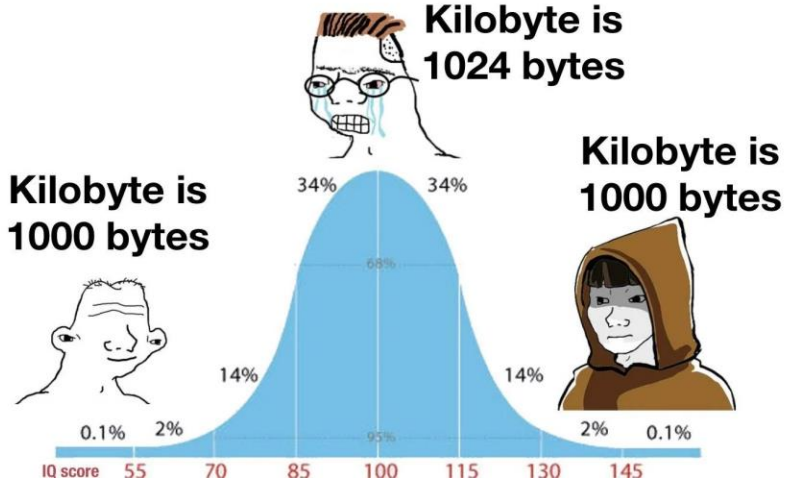
- ✓ Larger, but much **slower**
  - ✓ Speed measured in milliseconds
- ✓ Disk consists of blocks
- ✓ Read/write per block (= unit)
- ✓ Non-volatile

Exercise 1

What is the number of blocks needed on a 4 GiB* disk when the block size is 512 bytes?

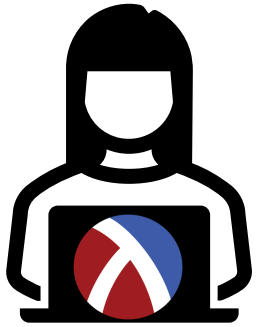
Old IEC-units

1024 bytes = 1 kilobyte $1024 = 2^{10}$
1024 kilobyte = 1 megabyte (1M)
1024 megabyte = 1 gigabyte (1G)
1024 gigabyte = 1 terabyte (1T)



*see https://physics.nist.gov/cuu/Units/binary.html

Reading and writing blocks



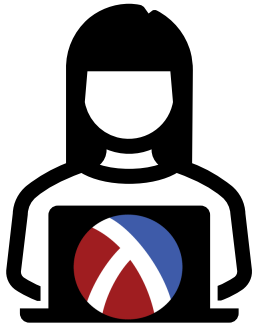
ADT disk

```
read-block:
    disk number -> block
```

ADT block

```
write-block!:
    block -> /
```

Reading and writing blocks

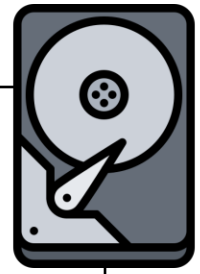


ADT disk

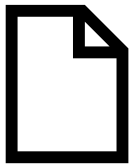
ADT block

```
read-block:
    disk number -> block

write-block!:
    block -> /
```

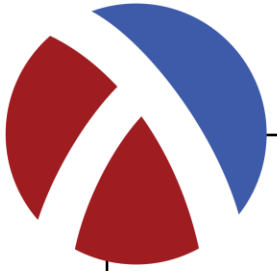


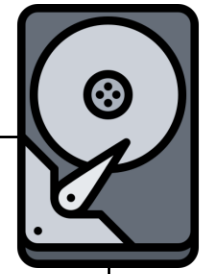
```
(define dsk
    (disk:new
     "/home/XXX/toydisk.disk"))
=> <#disk>
```



Creates new file
/home/XXX/toydisk.disk

Reading and writing blocks





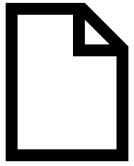
```
(define dsk
    (disk:mount "/home/XXX/toydisk.disk"))
=> <#disk>

(define blk (disk:read-block dsk 7))
=> <#block>

(disk:encode-byte! blk 56 212)

(disk:write-block! blk)

(disk:unmount dsk)
```



/home/XXX/toydisk.disk

Read and transfer a block from the disk

Modifications only occur on in-memory copy of the block!

Write the block back to the disk

Exercise 2

Prevent useless block writes to a disk when no changes have been made to the block

Implementations steps:

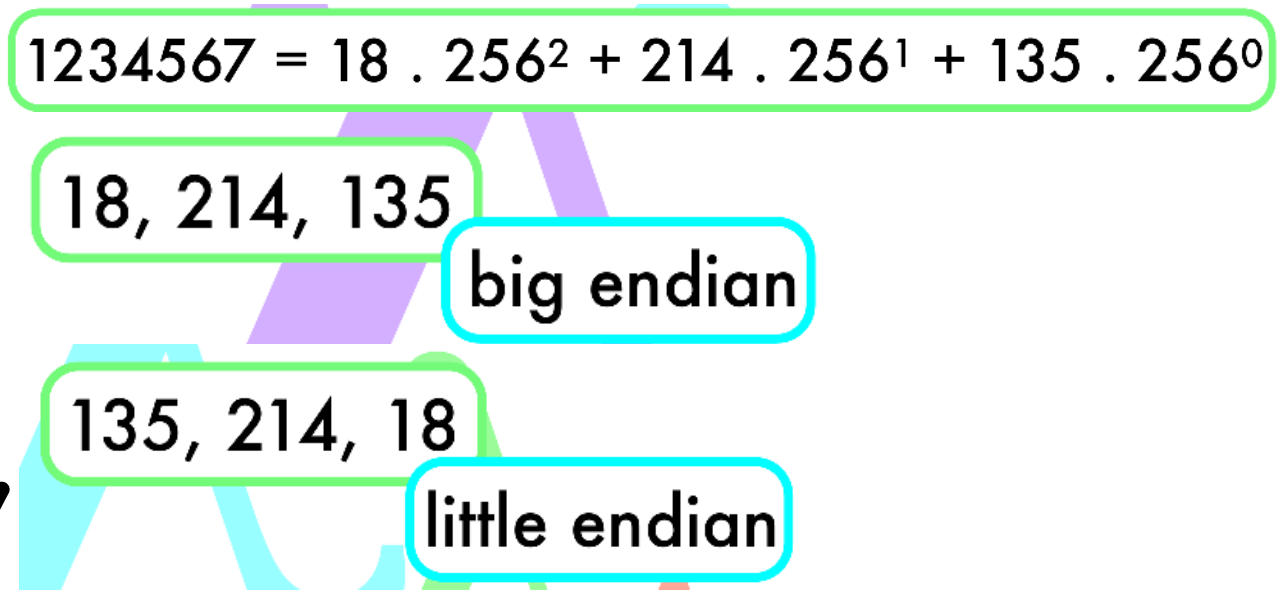
- ✓ add a **dirty flag** to the `block ADT`
- ✓ set the flag whenever block contents change (i.e., in all `disk:encode-XXX!` operations)
- ✓ `write-block!` only performs an actual disk write when block contents were modified (i.e., when the flag is set)
- ✓ make sure to clear the flag after writing back to the disk

Exercise 3

Reimplement **`encode-fixed-natural!`** and **`decode-fixed-natural`** without relying on functions provided by R6RS

- Choose either big or little endian
- What if n is too small to fit?

```
> (define number 1234567)
> (modulo number 256)
135
> (set! number (quotient number 256)
> (modulo number 256)
214
> (set! number (quotient number 256))
> number
18
```



Encoding integers in bytes

Option #1: sign-and-magnitude

- First bit is sign (0 = positive, 1 = negative)
- Remaining bits store absolute value
- e.g., using 1 byte: `-123` = 11111011_b

Option #2: two's complement

- Positive numbers: store unsigned value (starting with 0)
- Negative numbers: add 256^n and stored unsigned value (for n bytes)
- e.g., using 1 byte: `-123` = 10000101_b

Exercise 4A

What are the drawbacks of the **sign-and-magnitude** representation of integers?

Decimal	Sign-and-Magnitude	Two's complement
-8	-	1000
-7	1111	1001
-6	1110	1010
-5	1101	1011
-4	1100	1100
-3	1011	1101
-2	1010	1110
-1	1001	1111
0	1000 or 0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

Exercise 4B

Implement **`encode-fixed-integer!`** and **`decode-fixed-integer`**

- Choose one of both integer representations (choose wisely)
- Choose to use either the little- or big-endian byte-order

Tip for next WPO

Study `file-system.rkt`