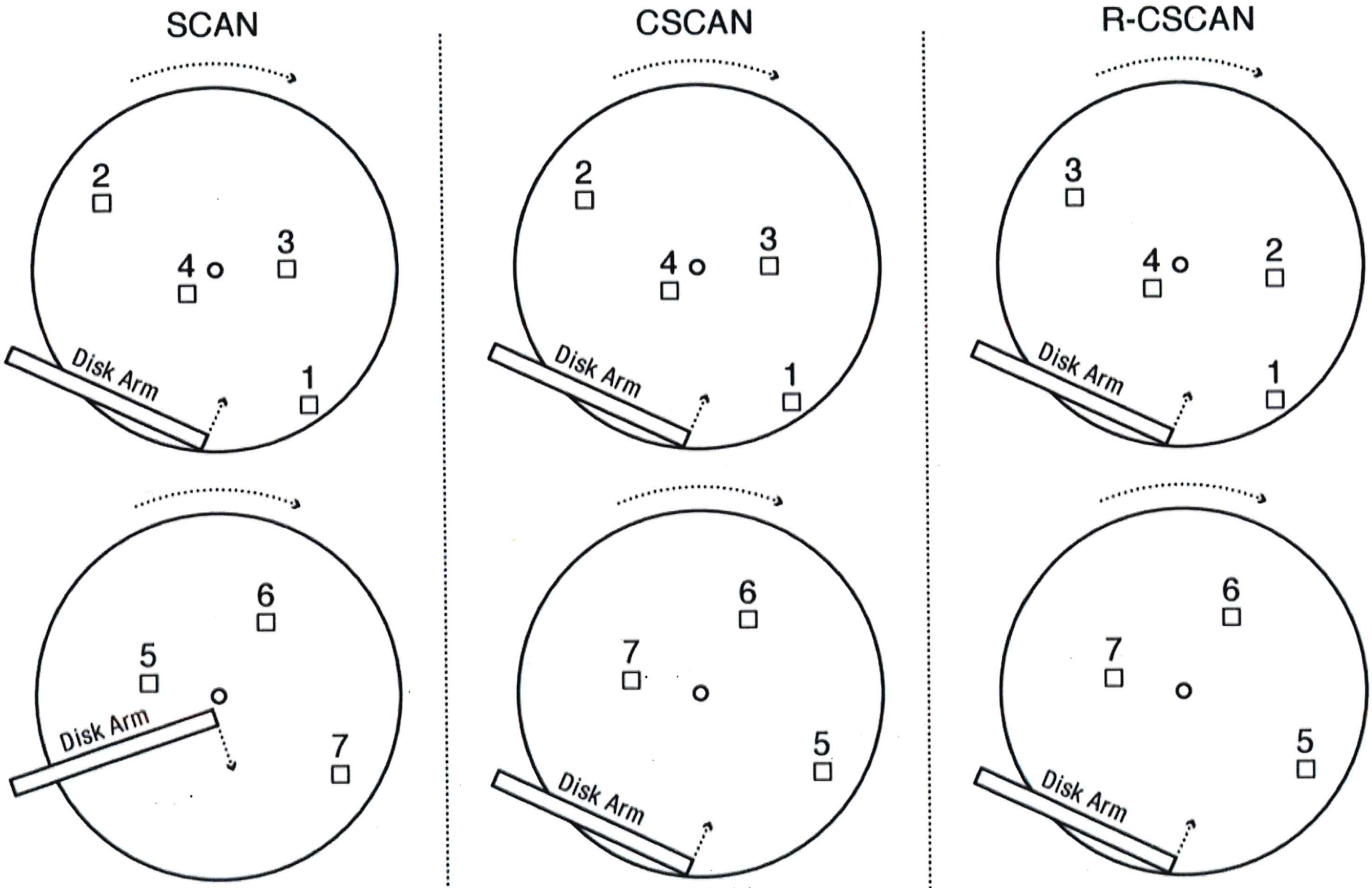
Disk scheduling

SCAN: - like an elevator
- first sweep in one direction, then other

CSCAN: - always sweeps in same direction

R-CSCAN: - allow small backward seeks < rotational delay



18.9

Example

avg seek = 8ms
avg rotation = 4ms
xfer = 250MB/s

How long to complete 100 random 4Kb reads?

Can do this in one sweep!

All seeks will be minimal ≈ 1ms
rotation = 4ms avg
xfer: 4Kb / 250 MB/s = 16 µs

total = 100 · (1ms + 4ms + 0.016ms) = 0.5 s

more than twice as fast as FIFO

How long to read whole disk?

32 TB / 250 MB/s = $32 \cdot 4000 = 128{,}000$ s = 35.5 hrs

Chance ≥ 1 bit is corrupt? bit error rate = 10^{-15}

32 TB = 256 Tbit = 2.56×10^{14} bit → ≈ 23% chance of ≥ 1 error

Lec 19

File Systems

File System Abstraction

- persistent, named data
- hierarchical (directories/folders that nest)
- access control

- File: named collection of data
  - linear sequence of bytes
- crash and storage error tolerant
- good performance on typical block devices

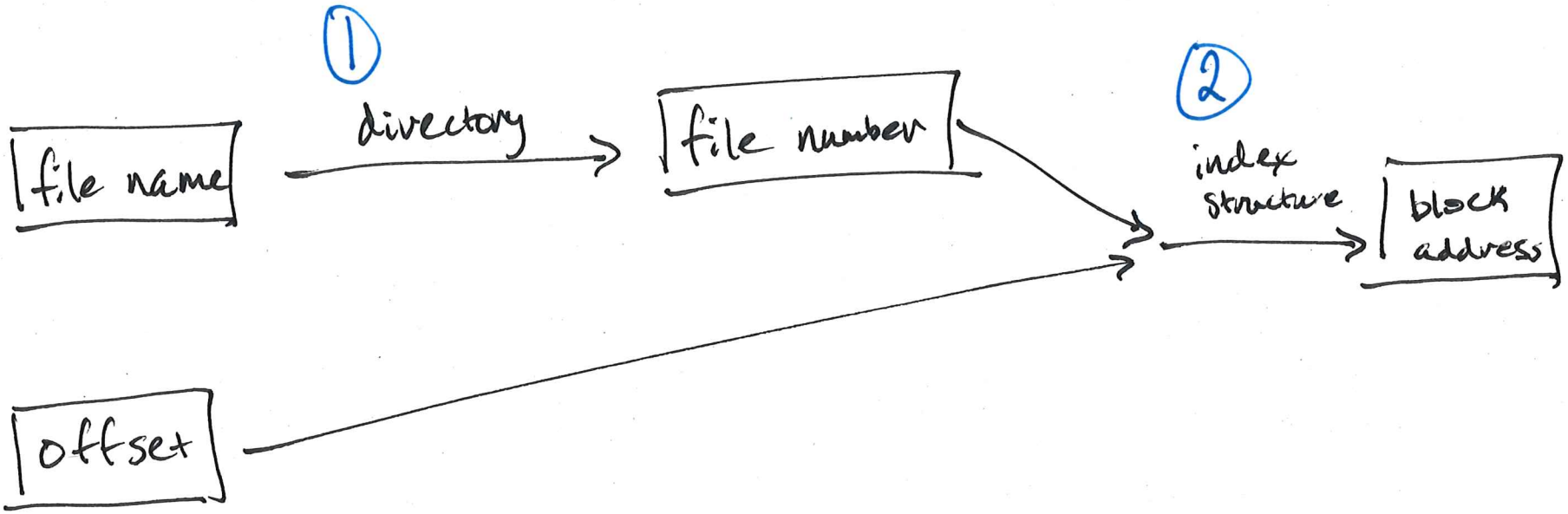
More Abstractions

- Directory
  - group of named files + subdirectories
  - maps name to metadata location
- Path: string that names a file/directory
  eg /cse/web/courses/cse451/25sp/index.html
- Links: hard link: two filenames point to same file
  soft link: one filename points to another filename
- Mount: map a path in one FS to root of another FS

Unix File System API

- create, link, unlink, createdir, rmdir
- open, close, read, write, seek
- fsync
  - Write does not actually send to disk – cached
  - fsync writes back the cache to disk
  - Warning: most programmers use fsync incorrectly

Implementing Names



read("/foo", offset = 400)

index structure maps "virtual" offset to "physical" offset

File System Workload

file size: are most files large or small?
which accounts for more total storage?

file access: are most accesses to large or small files?
which accounts for more total I/O?

are most accesses sequential or random?

are file sizes known at creation time?

File system design constraints

- for small files:
  - small blocks (low overhead)
  - locality across files
- for large files:
  - large blocks (low overhead)
  - locality within file
  - efficient random access
- unknown at creation time:
  - file size, file lifetime, access pattern

File System Design Options

	FAT	FFS	NTFS
index structure	linked list	fixed shape asymmetric tree	dynamic tree
granularity	block	block	extent
free space management	FAT array	bitmap (fixed location)	bitmap (file)
locality	defragmentation	block groups + reserve space	- extents - best fit - defragmentation