ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

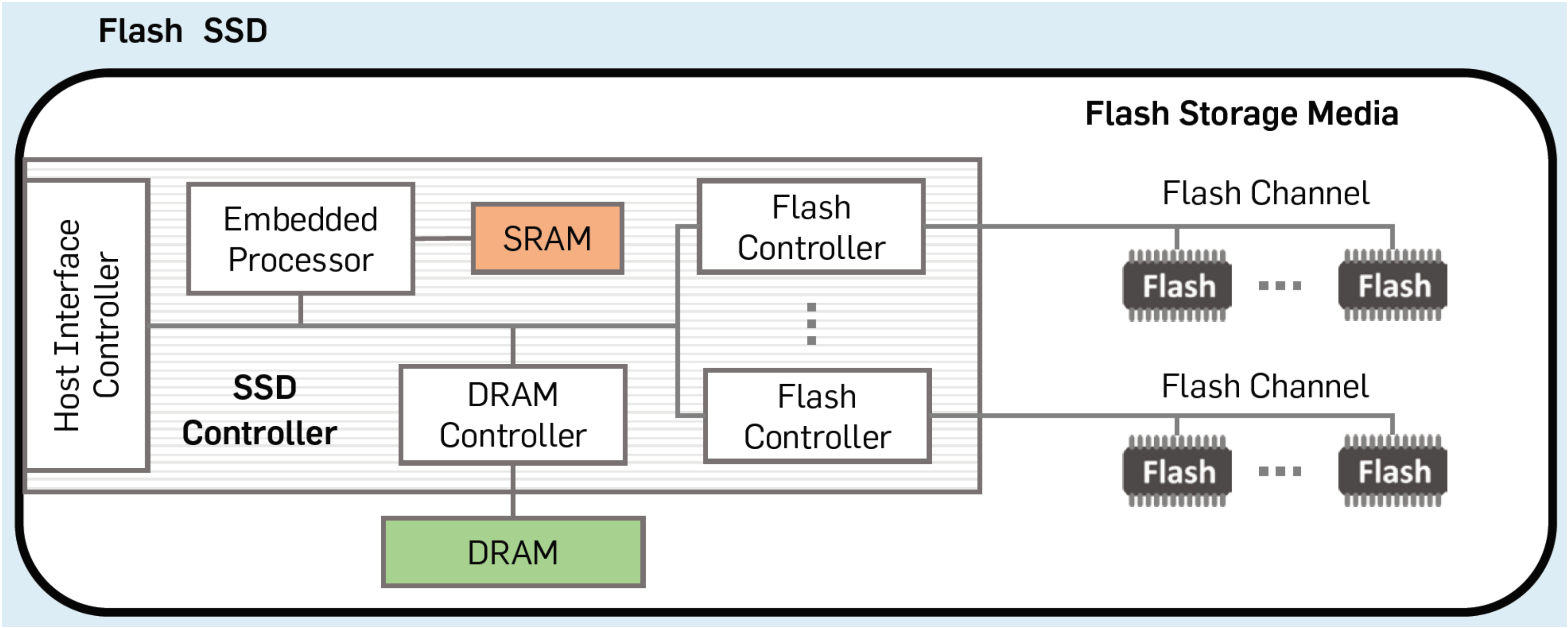
HY590.45
Modern Topics in Scalable Storage Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~hy590-45

Refresher

- Storage devices: SSDs, Disk drives
- File system data structures and disk layout
- File system consistency

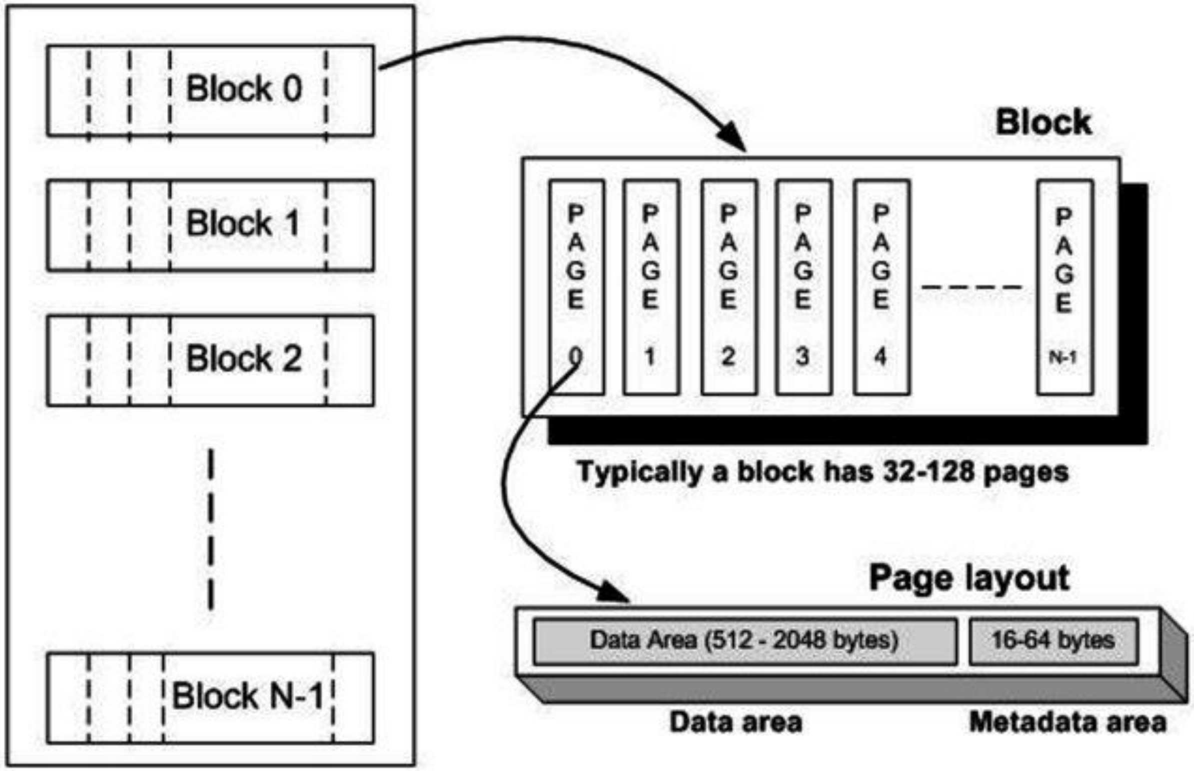
Flash-based SSD



SSD	Micron MTFDDAK480TDN 480GB @200$
HDD (SAS)	Toshiba MG04SCA20ENY 2TB @190$ (4.5x cheaper/GB vs. SSD)
HDD (SATA)	Toshiba MG04ACA200E 2TB @125$ (6.7x cheaper/GB vs. SSD)

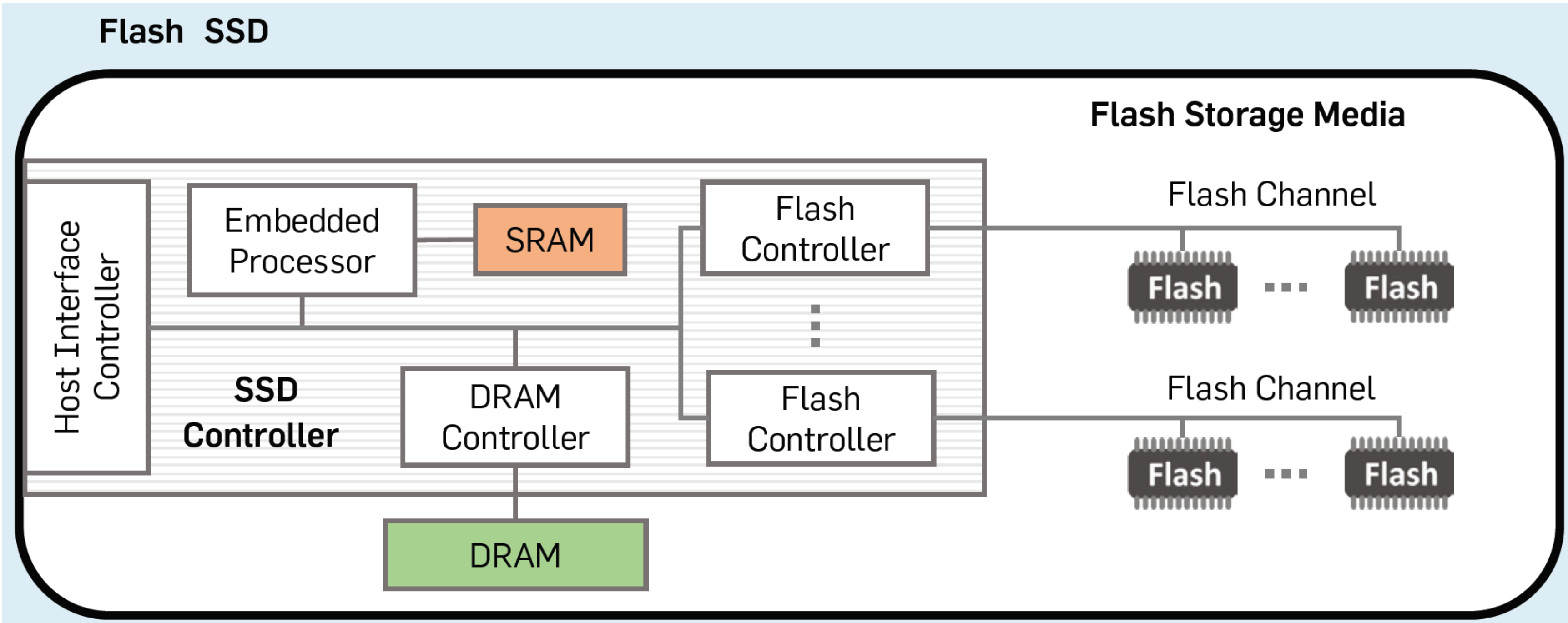
Diagram © Do *et al.*, "Programmable Solid-State Storage in Future Cloud Datacenters", CACM 62(6):56, June 2019

Internal architecture of NAND flash



- Reads/writes are performed by page
- Block ("erase unit") must be erased before being programmed (written)
- Logical I/O units should be mapped to physical NAND pages

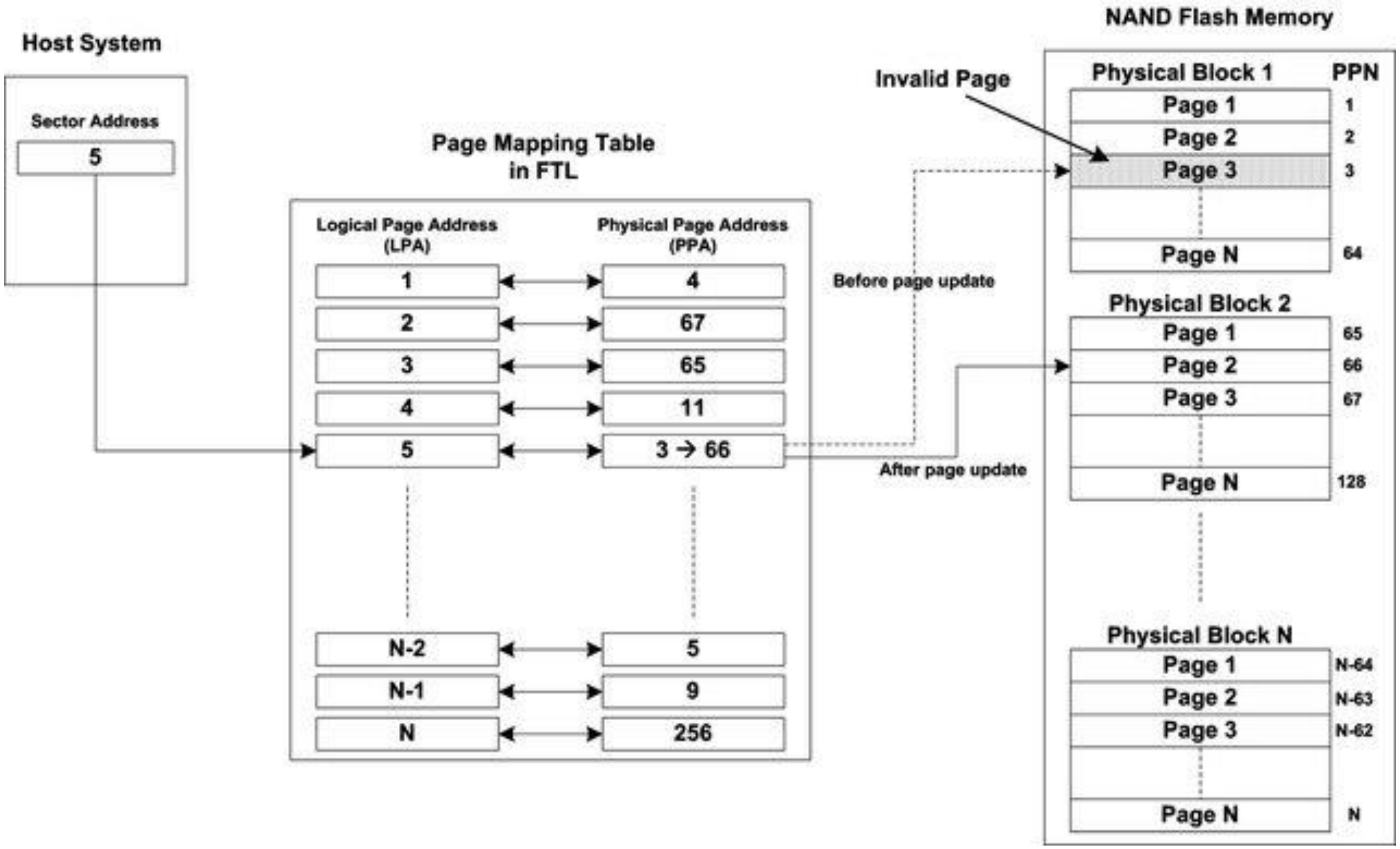
Flash-based SSD



- Typical flash memory performance
  - Sequentially reading at up to 400-500MB/sec
  - Sequentially writing at 100-200MB/sec due to complexity of programming
- Latency is a major benefit
  - 50-100 μsec for reads/writes
  - Random I/O (4KB) at ~2,000-40,000 IOPS (varies with op-type, device)
  - Erasure time: several milliseconds
- With HDDs, random reads and writes typically in the order of 10msec

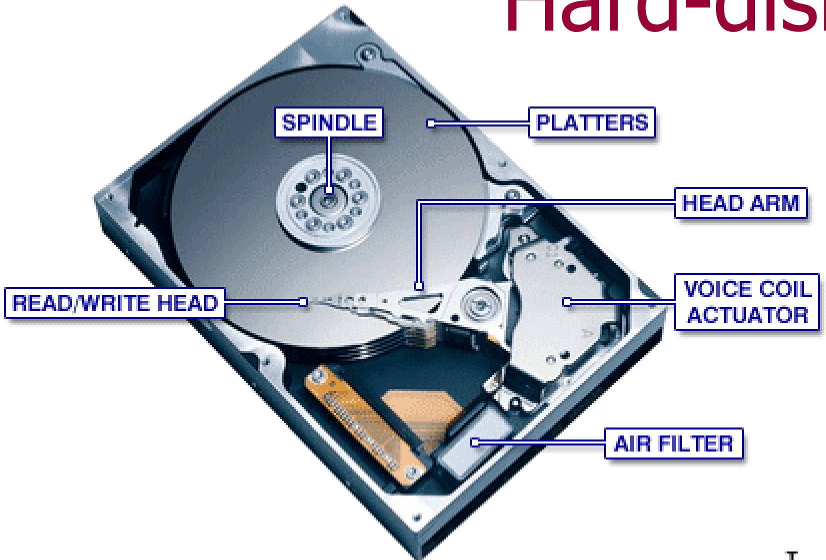
Diagram © Do *et al.*, "Programmable Solid-State Storage in Future Cloud Datacenters", CACM 62(6):56, June 2019

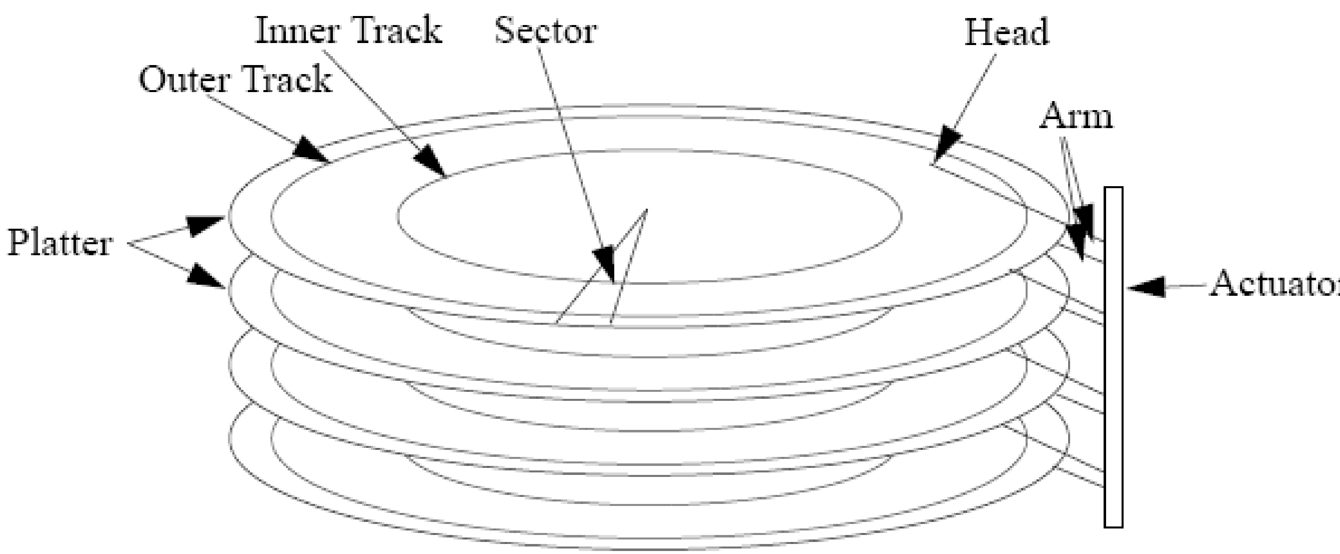
Page mapping scheme in Flash Translation Layer (FTL)



- Example: Consider consecutive writes to LPAs 5, 3, 5, 2
- FTL tasks: Mapping, garbage collection, wear leveling

Hard-disk drives (HDDs)





Characteristics

- Seek time
  - Move across cylinder boundary/ies
- Head switch time
  - Move across track boundary
- Write settle time
  - Position head onto right sector for a write
- Skew
  - Sector arrangement to avoid unnecessary rotational latency

Sequential LBN space vs. physical cylinder

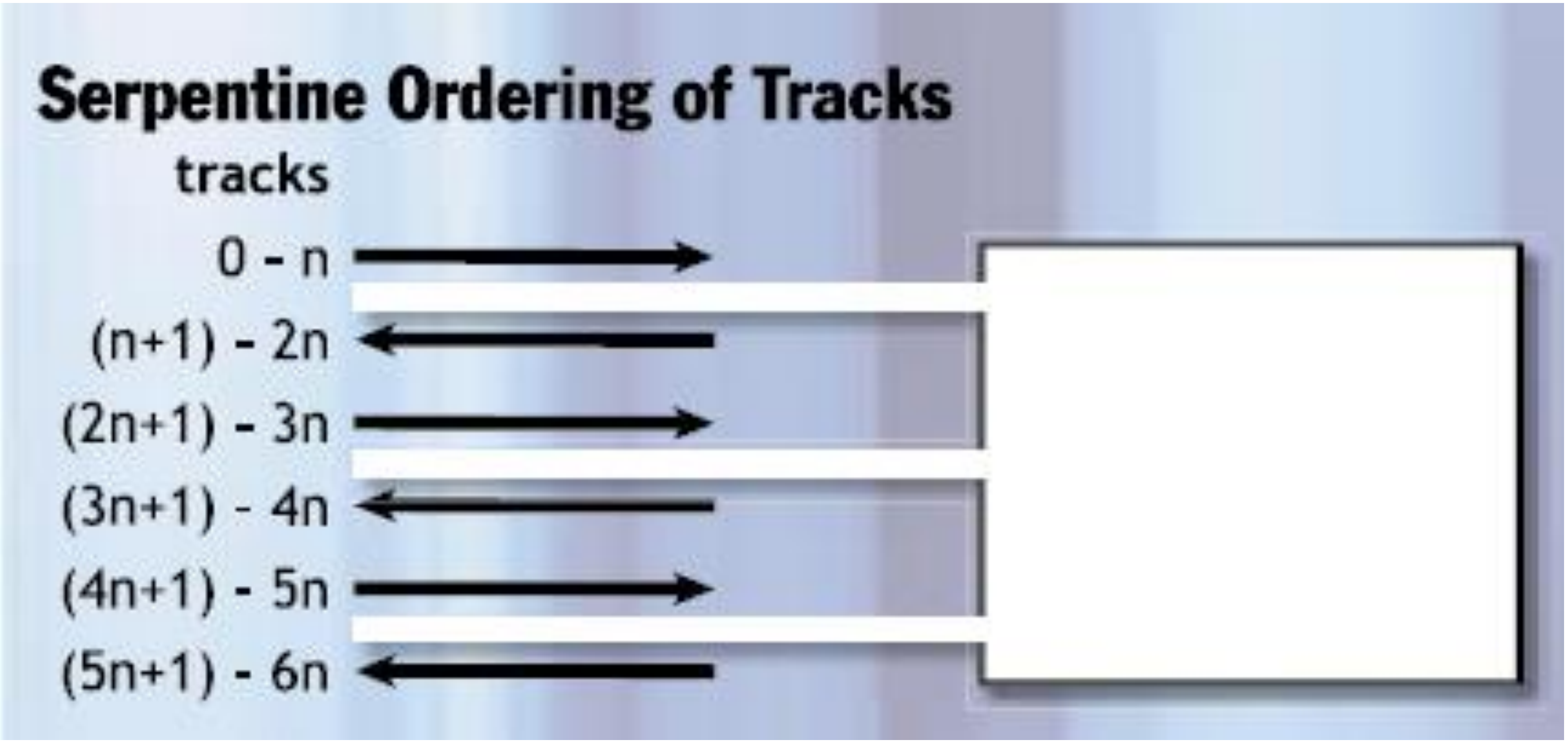


Image © D. Anderson "You don't know jack about disks", ACM Queue, June 2003

Addressing in modern disks

- LBN address space simplifies things for the OS
  - Highest bandwidth achieved for sequential access
- Caveat: OS must present I/Os to disk at right time
  - Sequential I/Os should have small inter-arrival times at the disk to avoid rotational latency
- Disk throughput typically measured in
  - IOPS for random I/Os
  - MB/s for sequential I/Os

Disks are better in scheduling I/Os

Operation	Starting LBA	Length
Read	724	8
Read	100	16
Read	9987	1
Read	26	128

The host might reorder this to:

Operation	Starting LBA	Length
Read	26	128
Read	100	16
Read	724	8
Read	9987	1

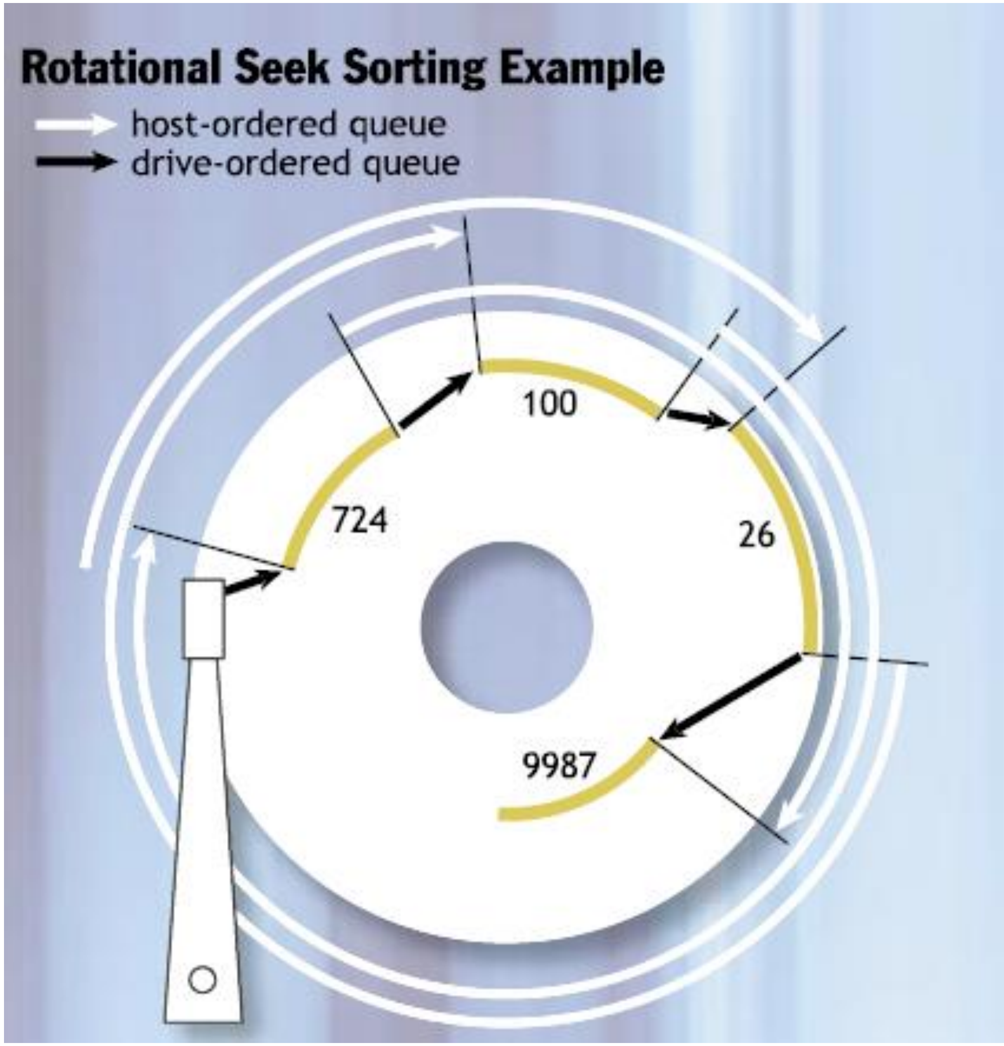


Image © D. Anderson "You don't know jack about disks", ACM Queue, June 2003

Evolution towards fewer platters

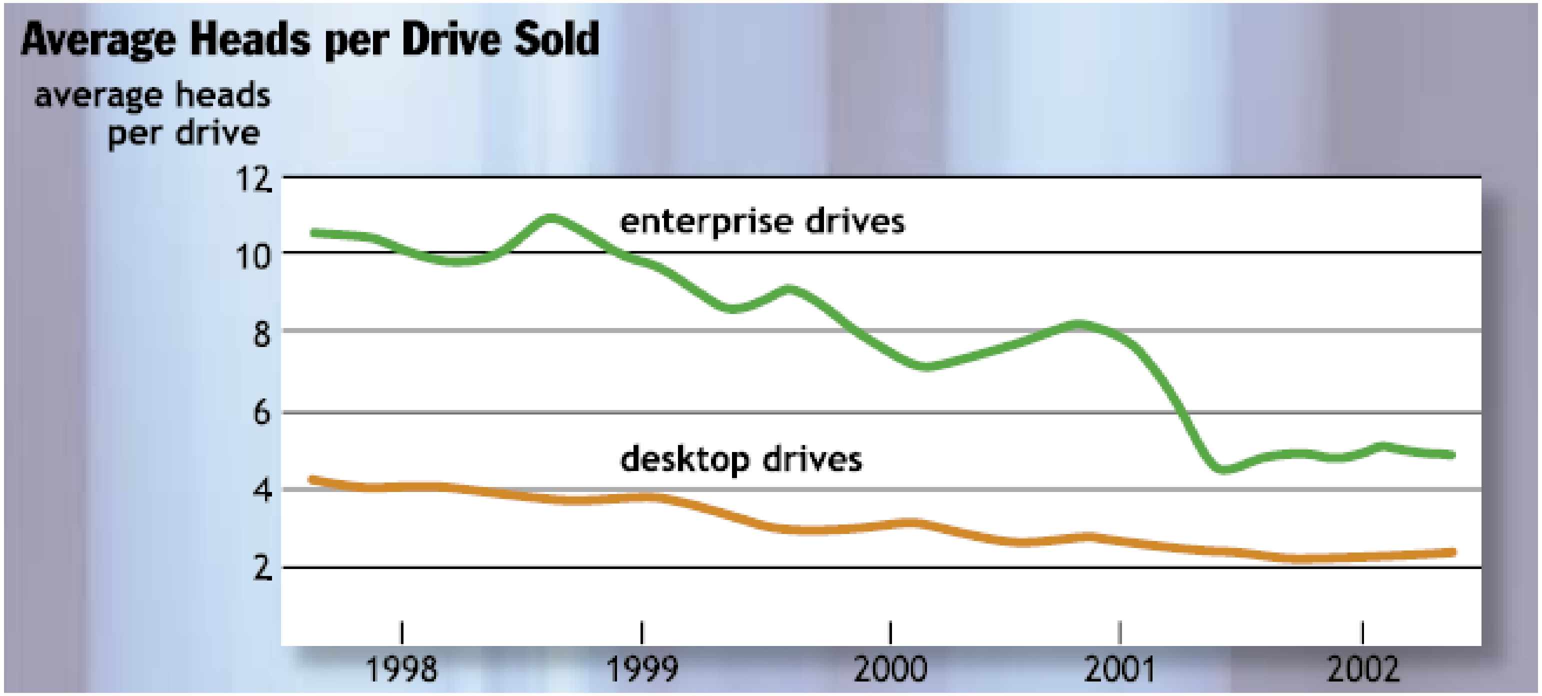


Image © D. Anderson “You don’t know jack about disks”, ACM Queue, June 2003

Drive technology comparison

- Serial ATA (SATA)
  - High capacity, best $/GB
  - 7200RPM, slow seeks
  - Multiple (4-5) wide platters
- Serially-attached SCSI (SAS)
  - Best performance
  - 15000RPM, fast seeks
  - Single narrow platter

Refresher

- Disk drive
- File system data structures and disk layout
- File system consistency

File system data structures

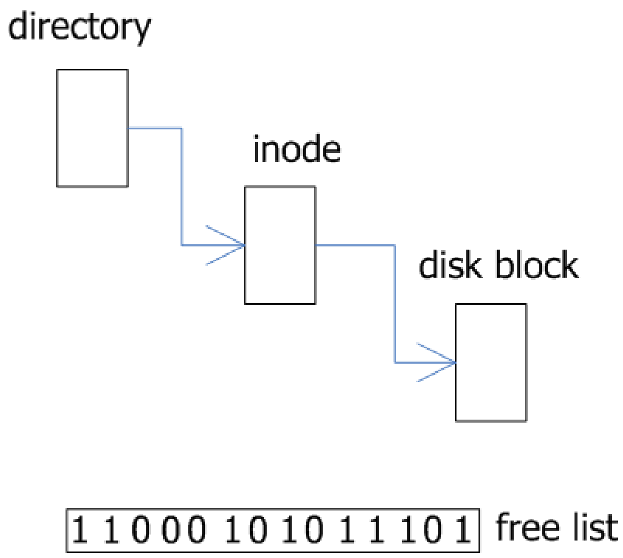
- Metadata
  - Directories
  - inodes
  - Indirect blocks
  - Free lists
  - Superblock

- Metadata operations involve several steps
  - E.g., rm <file>, truncate file to zero length

- Updates should preserve metadata consistency

Examples

- System call : rm <file>
  1. remove directory entry
  2. free inode
  3. free disk blocks used by file



- System call : truncate <file>
  1. modify inode
  2. free disk blocks used by file

- Consistency requirement (weak): No stale pointers left in stable store after a crash

Achieving consistency by ordering writes

- Older FSes ensure order via synchronous writes
  - Significant impact on performance (critical path)
  - Recovery requires scavenging, which can take hours (or days!) for large file systems
- Recovery through UNIX fsck
  - Check inodes, build bitmap of used data blocks
  - Record inode numbers, block addrs of all directories
  - Validate structure of directory tree
  - Validate directory contents to account for all files
  - Handle errors
    - Orphan directories and files in lost+found
    - Check bitmaps and summary counts

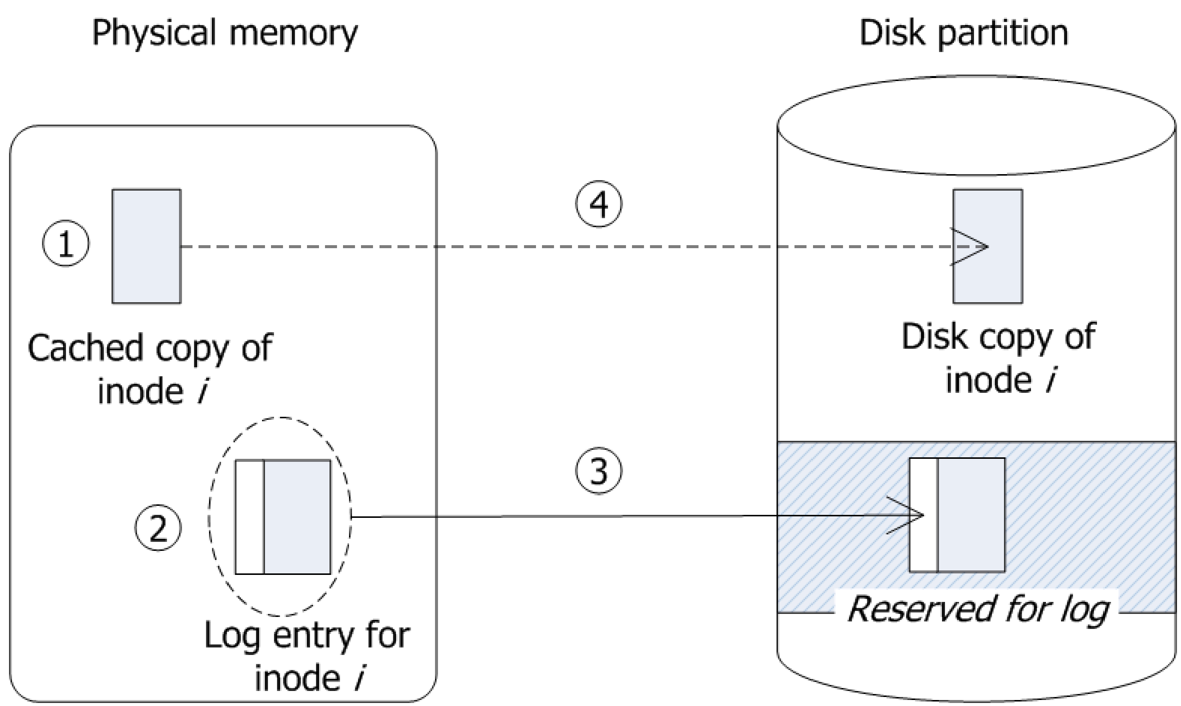
File system consistency

- How to achieve efficiently
  - Logging (or journaling) on disk or NVRAM
- Logging is general method for achieving atomic updates – the *"all or nothing"* property
  - Write "intentions" to an append-only log before issuing the actual (in-place) I/Os
  - Atomicity based on atomic primitive (sector write) supported by (or constructed over) stable store (disk)
- General transactions ensure ACID properties
  - Atomicity, consistency, isolation, durability

Logging (or Journaling)

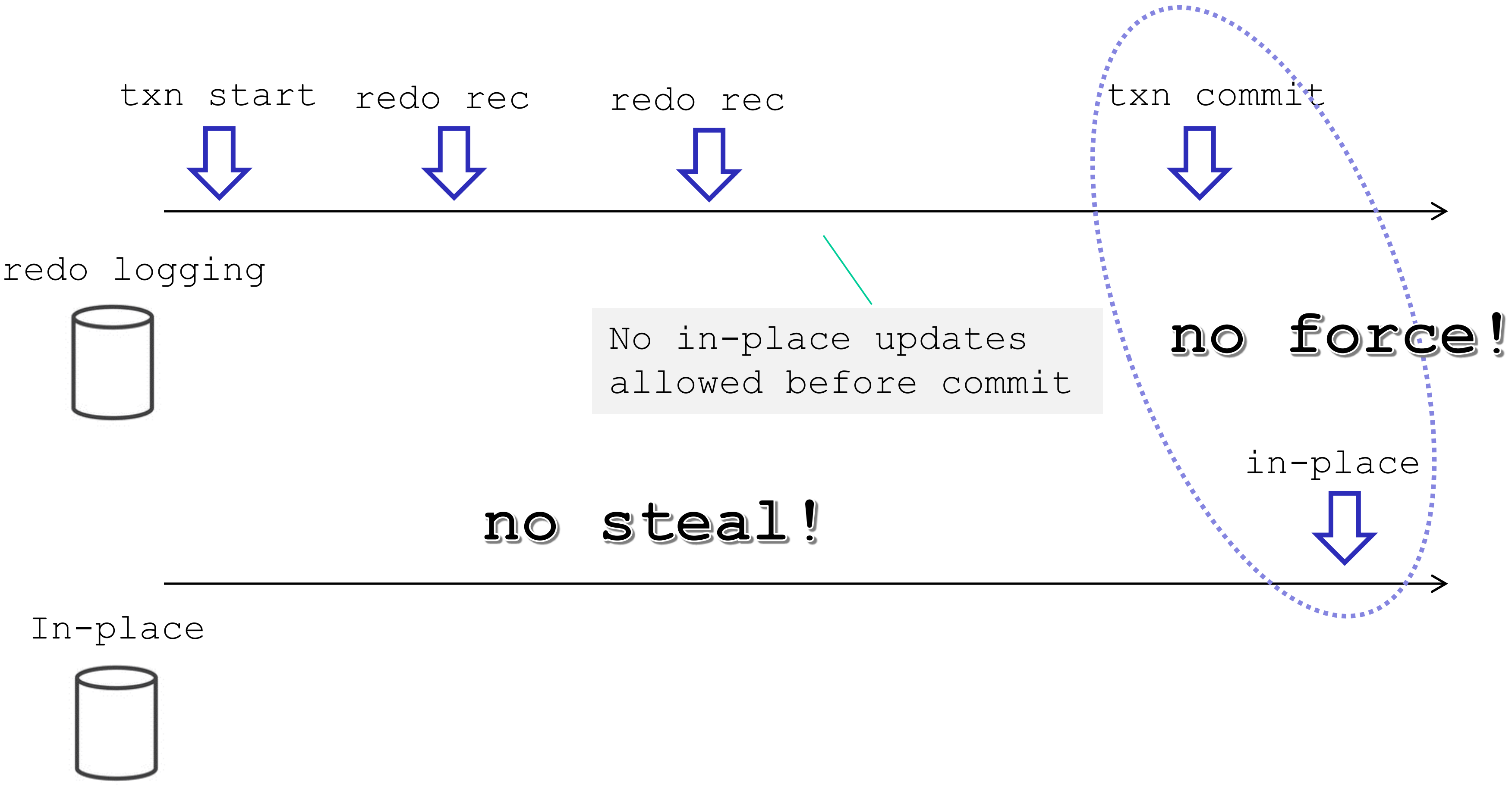
- Redo vs. undo records
  - Log new value, old value (or both)
- Rapid recovery
  - Only the tail of the log needs to be examined, replayed
- Good performance
  - Sequential writes (appends), saves I/Os
  - However, need to consider interference with other activity
- Other issues
  - Group commit
  - Garbage collection
  - Indexing for data retrieval

Metadata logging implementation

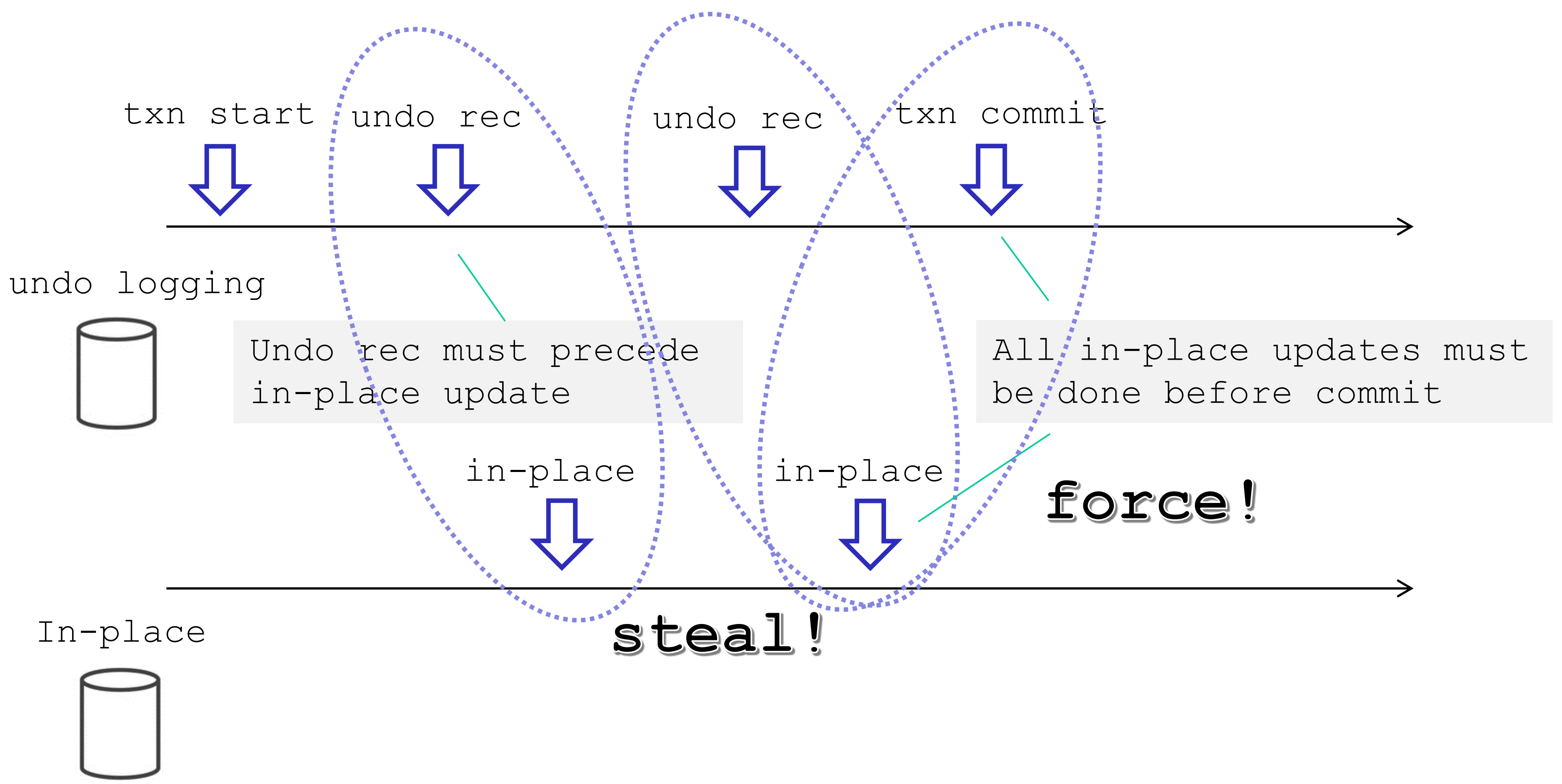


- Each metadata update is written to disk twice
- However, in-place updates can be batched or eliminated
- Batching can be applied to log writes as well (group commit)

Redo logging



Undo logging



Undo vs. Redo logging

- Undo
  - Can *steal* dirty bufs but must *force* in-place I/O before commit
  - Undo records are logged (WAL) then updates written in-place
    - Thus, dirty buffers can be cleaned up (*stolen*) before commitment
  - In-place updates must be *forced* <u>before</u> commit time
    - Without force, it would not be possible to apply changes after commitment
- Redo
  - *Cannot steal* dirty buffers but *need not force* in-place I/O
  - Redo records logged before commitment
  - In-place updates permitted only after commitment (*no-steal)*
  - Need not force in-place updates at commit time (*no force*)
- Redo-undo: Best of both worlds
  - At the expense of some space and complexity

Log consistency between concurrent operations on same set of objects

- Suppose p1 modifies A & concurrently p2 reads A and based on its contents modifies B
- Example: p1 deletes a file from block A of a directory; p2 finds that the directory does not have name A, then creates a directory entry in block B of the directory