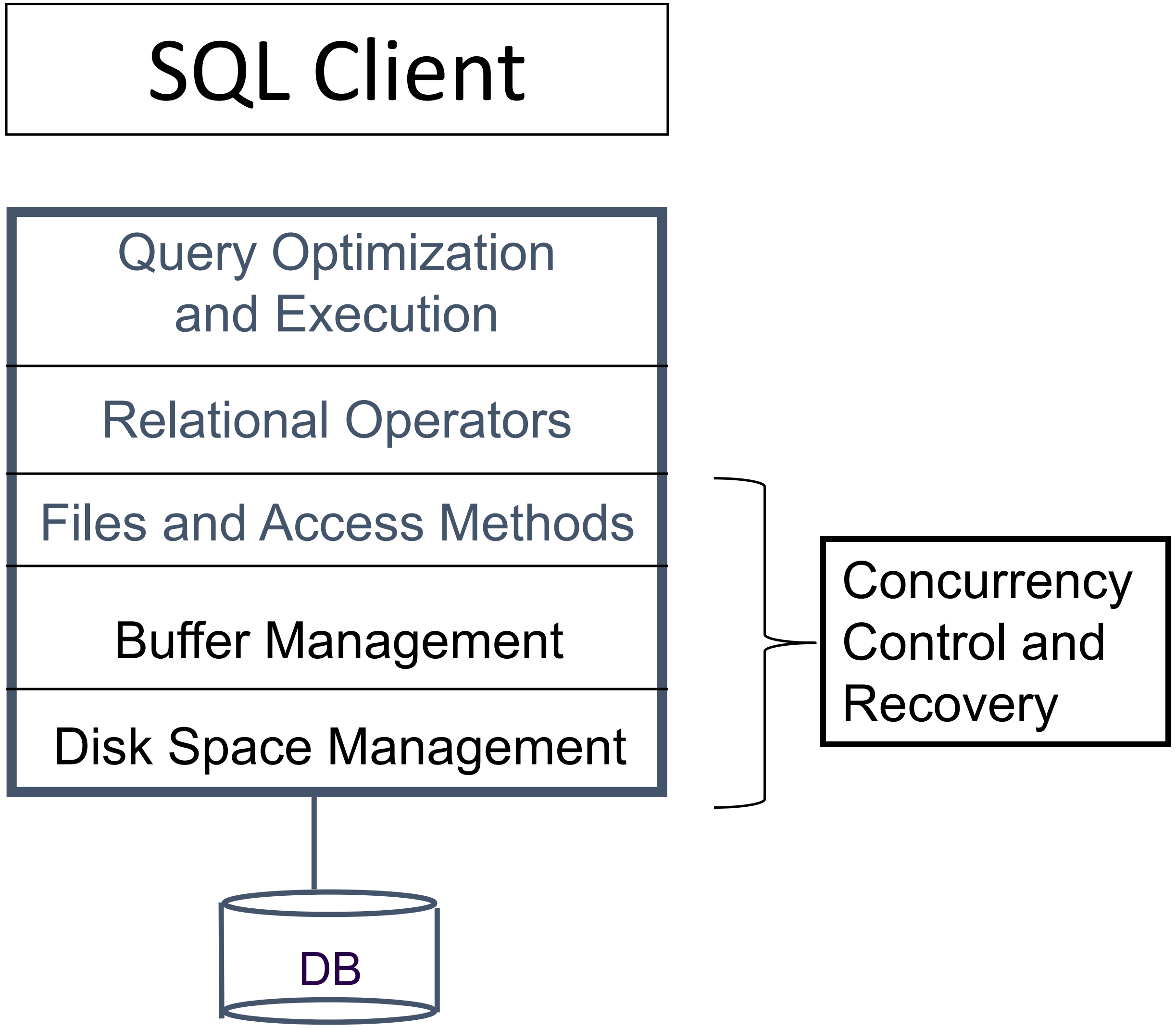
Block diagram of a DBMS



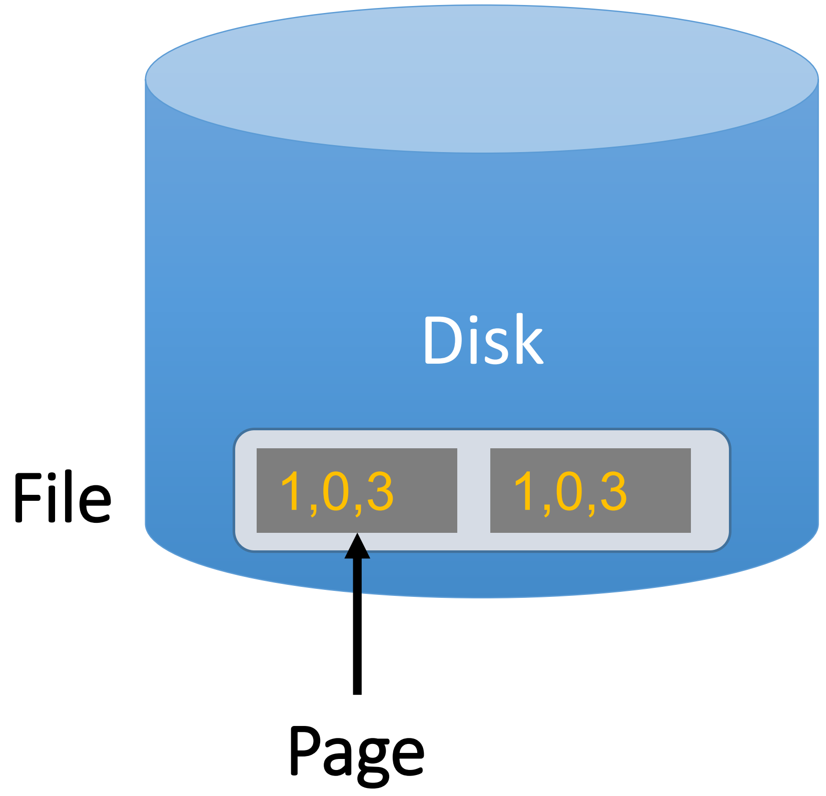
1. The Disk and Files

Overview: Files of Pages of Records

- Tables stored as a *logical files* consisting of *pages* each containing a collection of *records*
- Pages are managed
  - in memory by the buffer manager: higher levels of database only operate in memory
  - on disk by the disk space manager: reads and writes pages to physical disk/files

A Simplified Filesystem Model

- For us, a **page** is a ***fixed-sized array*** of memory
  - Think: One or more disk blocks
  - Interface:
    - write to an entry (called a **slot**) or set to “None”
  - DBMS also needs to handle variable length fields
    - Page layout is important for good hardware utilization as well (see next next lecture)
- And a **file** is a *variable-length list* of pages
  - Interface: create / open / close; next_page(); etc.

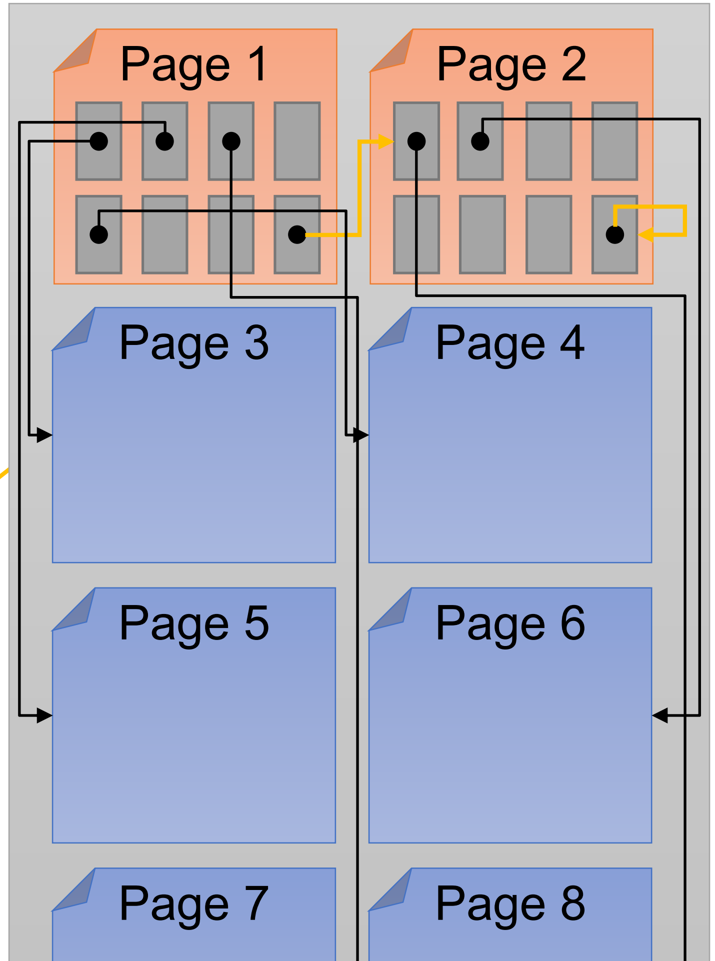


Overview

Table

Name	Addr	Sex	Age	Zip
Bob	Harmon	M	32	94703
Alice	Mabel	F	33	94703
Jose	Chavez	M	31	94110
Jane	Chavez	F	30	94110

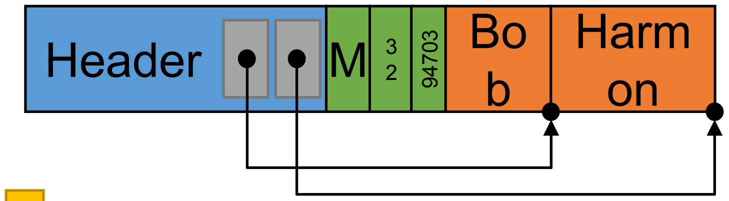
File



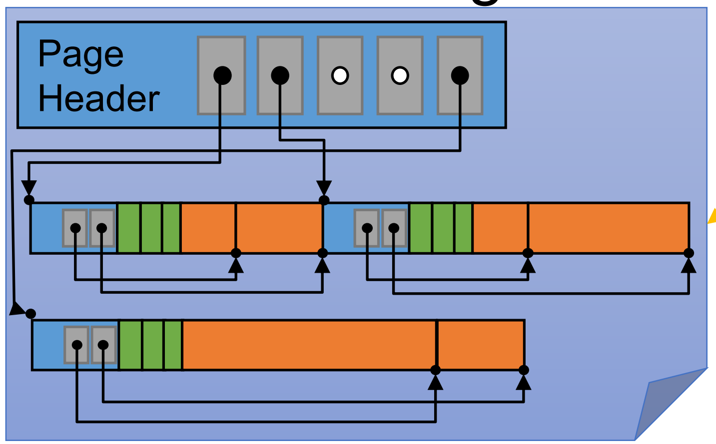
Record

Bob	Harmon	M	32	94703
Varchar	Varchar	Char	Int	Int

Byte Rep. Record

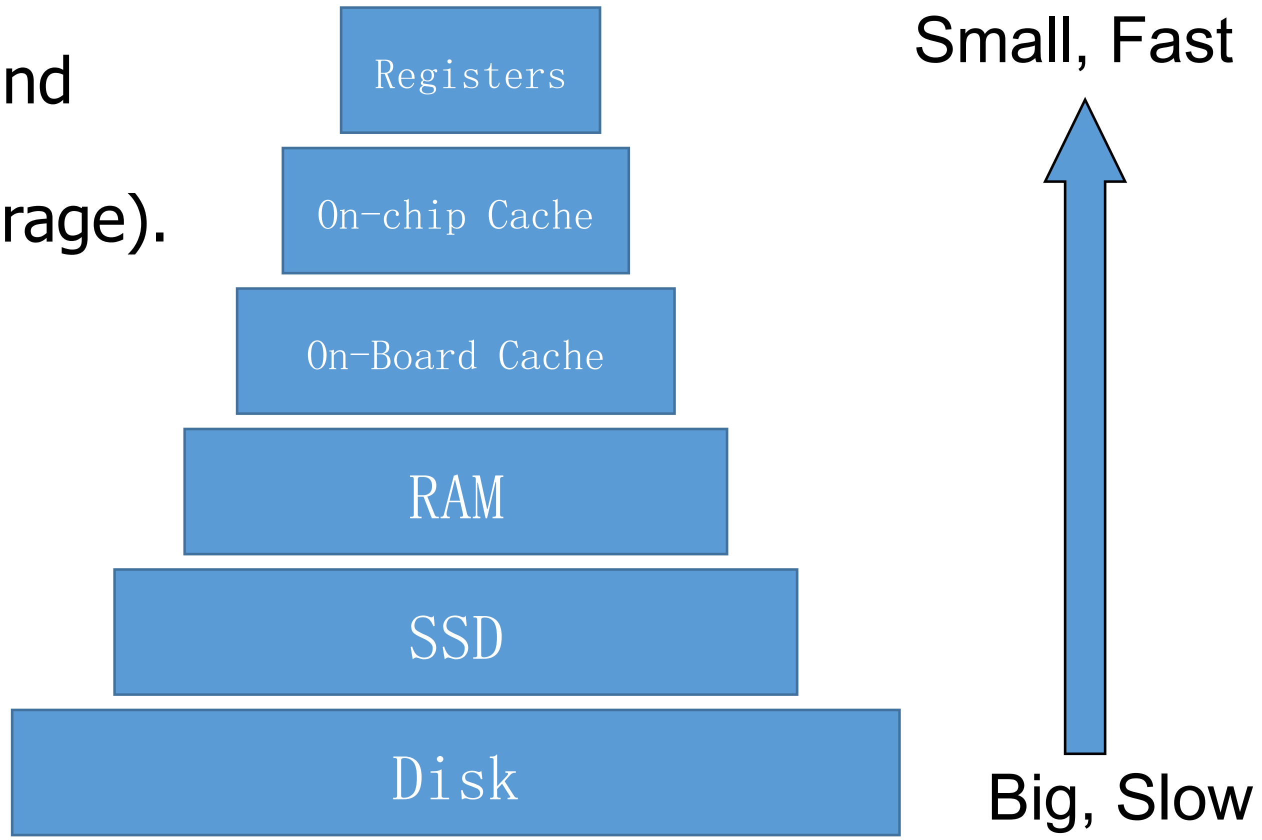


Slotted Page



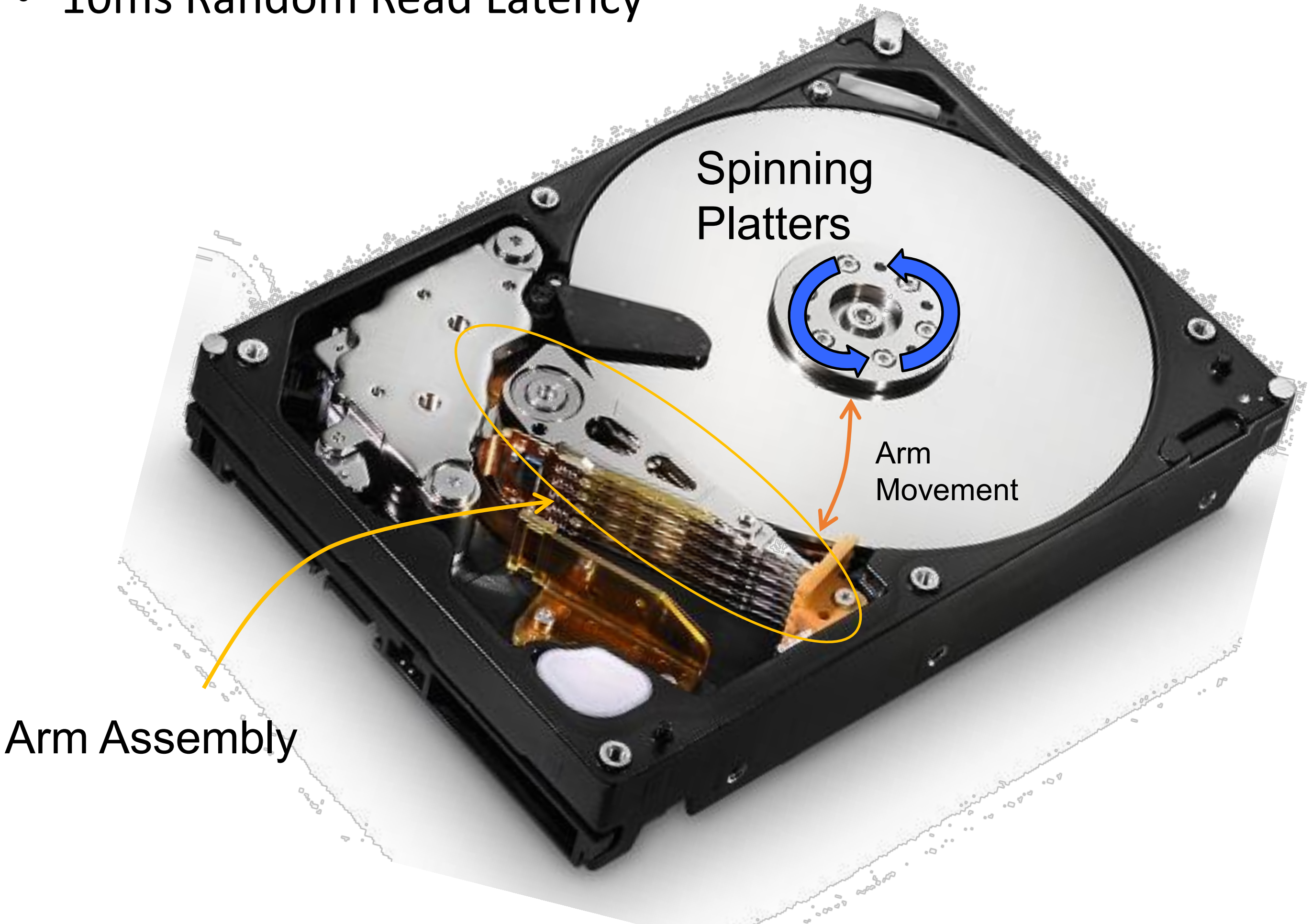
The Storage Hierarchy

- Main memory (RAM) for currently used data.
- Disk for main database and backups/logs (secondary & tertiary storage).
- The role of Flash (SSD) varies by deployment
  - Sometimes the DB
  - Sometimes a cache



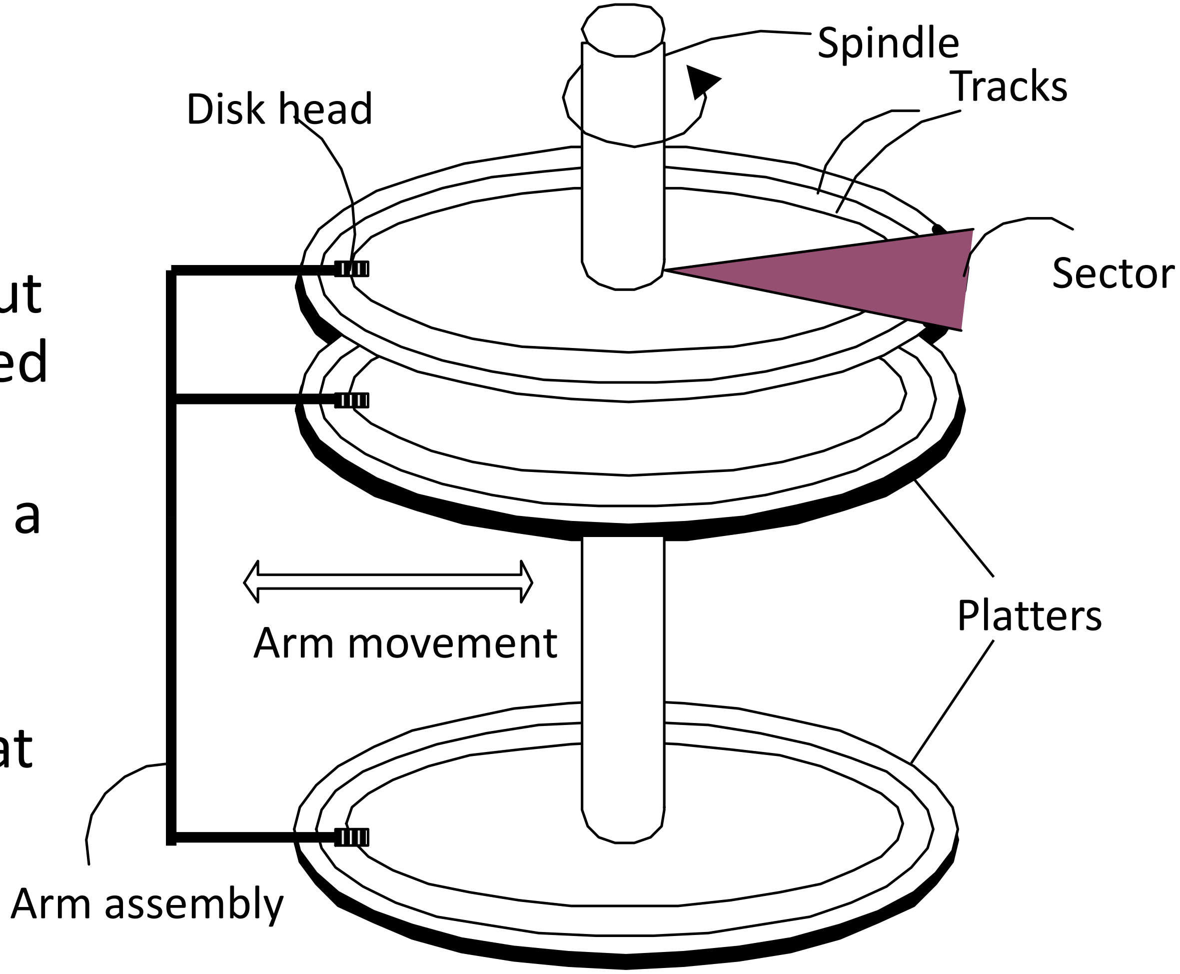
Disks

- DBMS stores information on Disks and SSDs.
  - Disks are a mechanical anachronism (slow!)
    - 10ms Random Read Latency



Components of a Disk

- Platters spin (say 15000 rpm)
- Arm assembly moved in or out to position a head on a desired track.
  - Tracks under heads make a cylinder (imaginary)
- Only one head reads/writes at any one time
- Block/page size is a multiple of (fixed) sector size

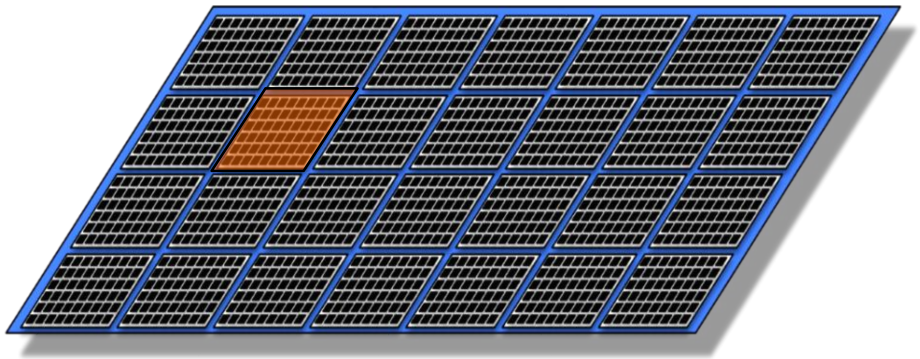


Arranging Pages on Disk

- *"Next"* page concept:
  - pages on same track, followed by
  - pages on same cylinder, followed by
  - pages on adjacent cylinder
- Arrange file pages sequentially on disk
  - minimize seek and rotational delay.
- For a sequential scan, *pre-fetch*
  - several pages at a time!
- Read large consecutive blocks

Notes on Flash (SSD)

- Issues in current generation (NAND)
  - 4-8K reads, 1-2MB writes
  - Only 2k-3k erasures before failure, so move writes around (“wear leveling”)
  - *Write amplification*: big units, need to reorg for garbage collection & wear
- So... read is fast and *predictable*
  - Single read access time: 0.03 ms
  - 4KB random reads: ~500MB/sec
  - Sequential reads: ~525MB/sec
  - 64K: 0.48msec
- But.. write is not! Slower for random
  - Single write access time: 0.03ms
  - 4KB random writes: ~120MB/sec
  - Sequential writes: ~480MB/sec



Disk Space Management

Lowest layer of DBMS, manages space on disk

- Mapping pages to locations on disk
- Loading pages from disk to memory
- Saving pages back to disk & ensuring writes

Higher levels call upon this layer to:

- read/write a pages
- allocate/de-allocate logical pages

Request for a *sequence* of pages best satisfied by pages stored sequentially on disk

- Physical details hidden from higher levels of system
- Higher levels may assume **Next Page** is fast!

Disk Space Management Implementation

Proposal 1: Talk to the device directly

- Could be very fast if you knew the device well
- What happens when devices change?

Proposal 2: Run over filesystem (FS)

- Allocate single large “contiguous” file and assume sequential / nearby byte access are fast
- Most FS optimize for sequential access and temporal locality (buffer cache on hot items)
  - Sometimes disable FS buffering
- May span multiple files on multiple disks / machines

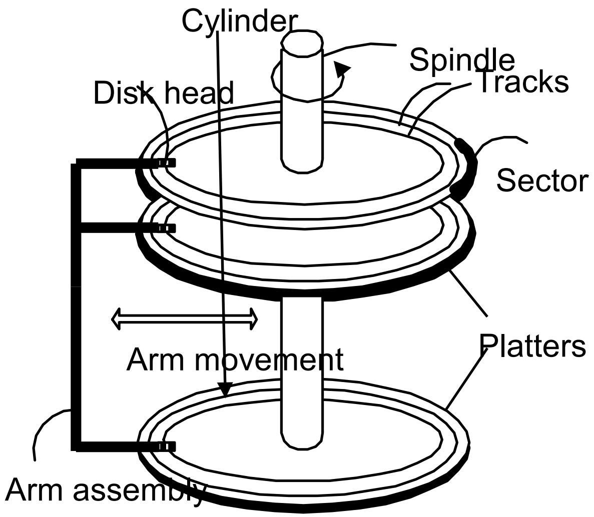
Typically sits on top of local file system

Get Page 4

Get Page 5

2. The Buffer

High-level: Disk vs. Main Memory



<u>Disk</u>:

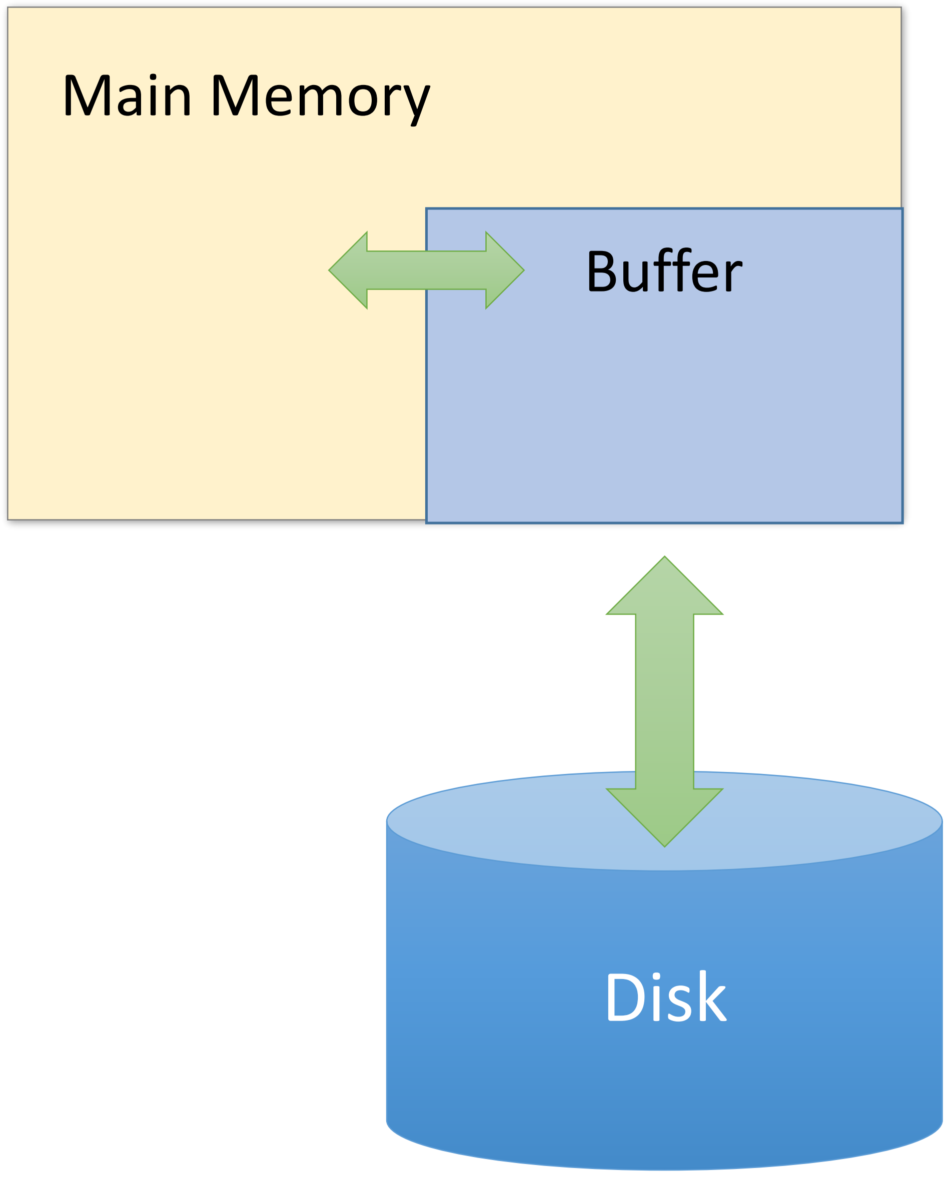
- ***Slow:*** Sequential *block* access
  - Read a blocks (not byte) at a time, so sequential access is cheaper than random
  - **Disk read / writes are expensive!**
- ***Durable:*** We will assume that once on disk, data is safe!
- ***Cheap***

<u>Random Access Memory (RAM) or Main Memory:</u>

- ***Fast:*** Random access, byte addressable
  - ~10x faster for <u>sequential access</u>
  - ~100,000x faster for <u>random access!</u>
- ***Volatile:*** Data can be lost if e.g. crash occurs, power goes out, etc!
- ***Expensive:*** For $100, get 16GB of RAM vs. 2TB of disk!

The Buffer

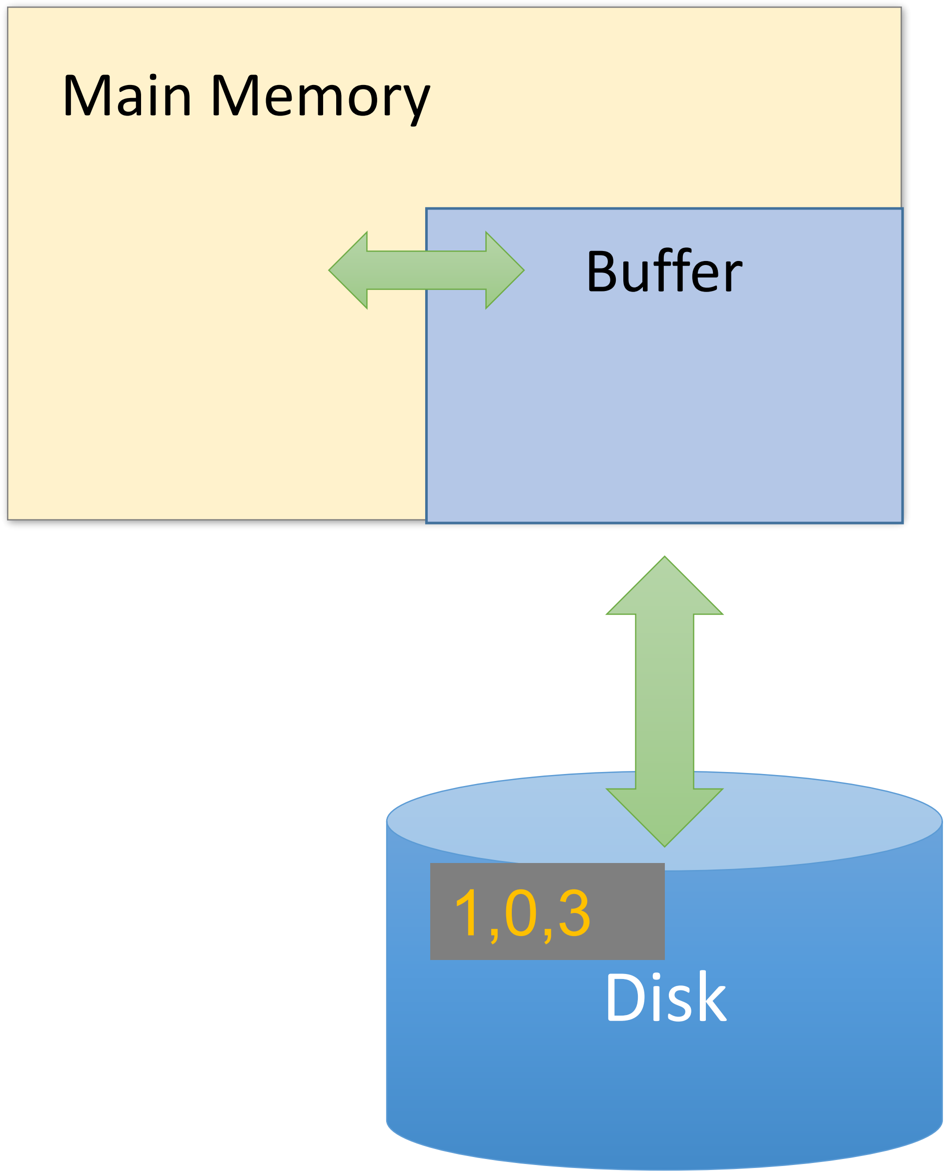
- A **buffer** is a region of physical memory used to store *temporary data*
  - *In this lecture:* a region in main memory used to store **intermediate data between disk and processes**
- *Key idea:* Reading / writing to disk is slow- need to cache data!



The (Simplified) Buffer

- In this class: We'll consider a buffer located in **main memory** that operates over **pages** and **files**:

  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*

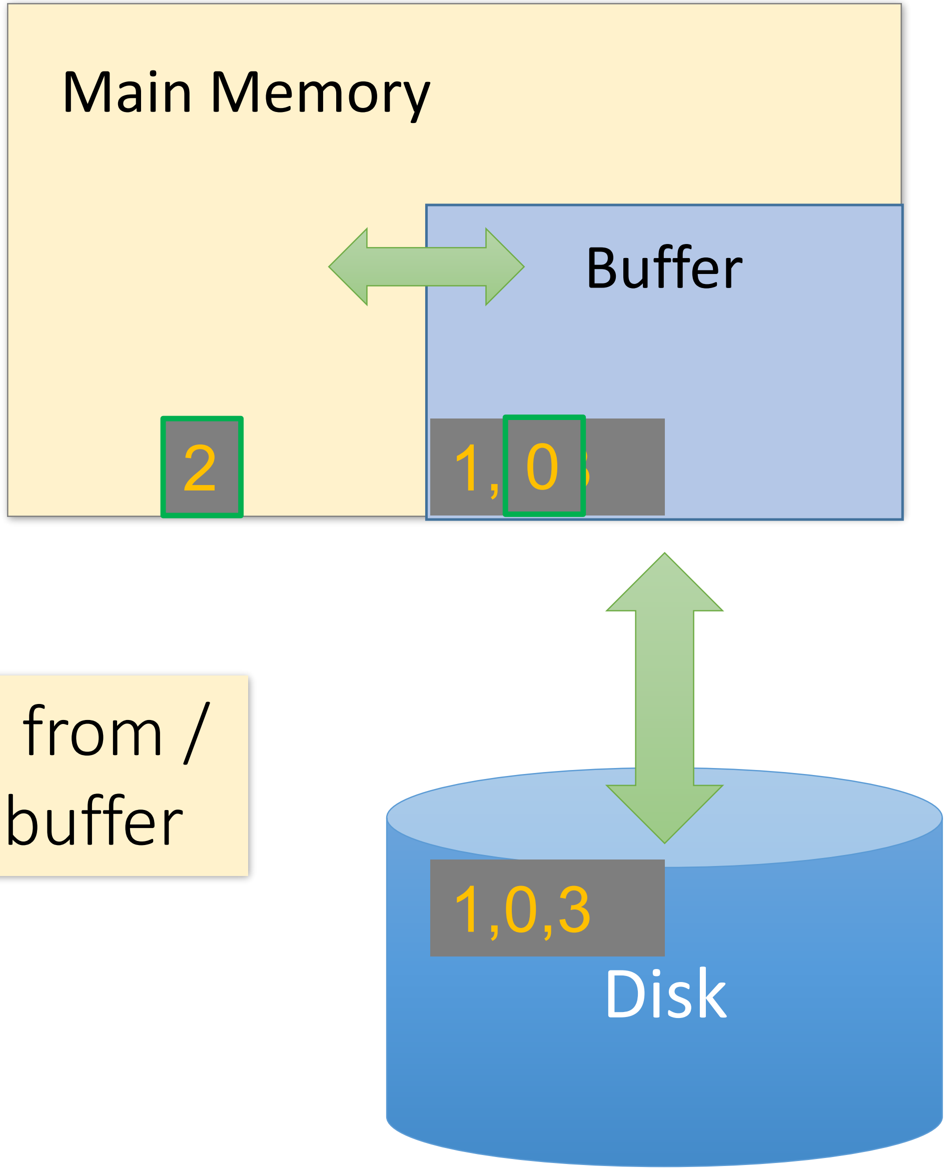


The (Simplified) Buffer

- In this class: We'll consider a buffer located in **main memory** that operates over **pages** and **files**:

    - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*

Processes can then read from / write to the page in the buffer



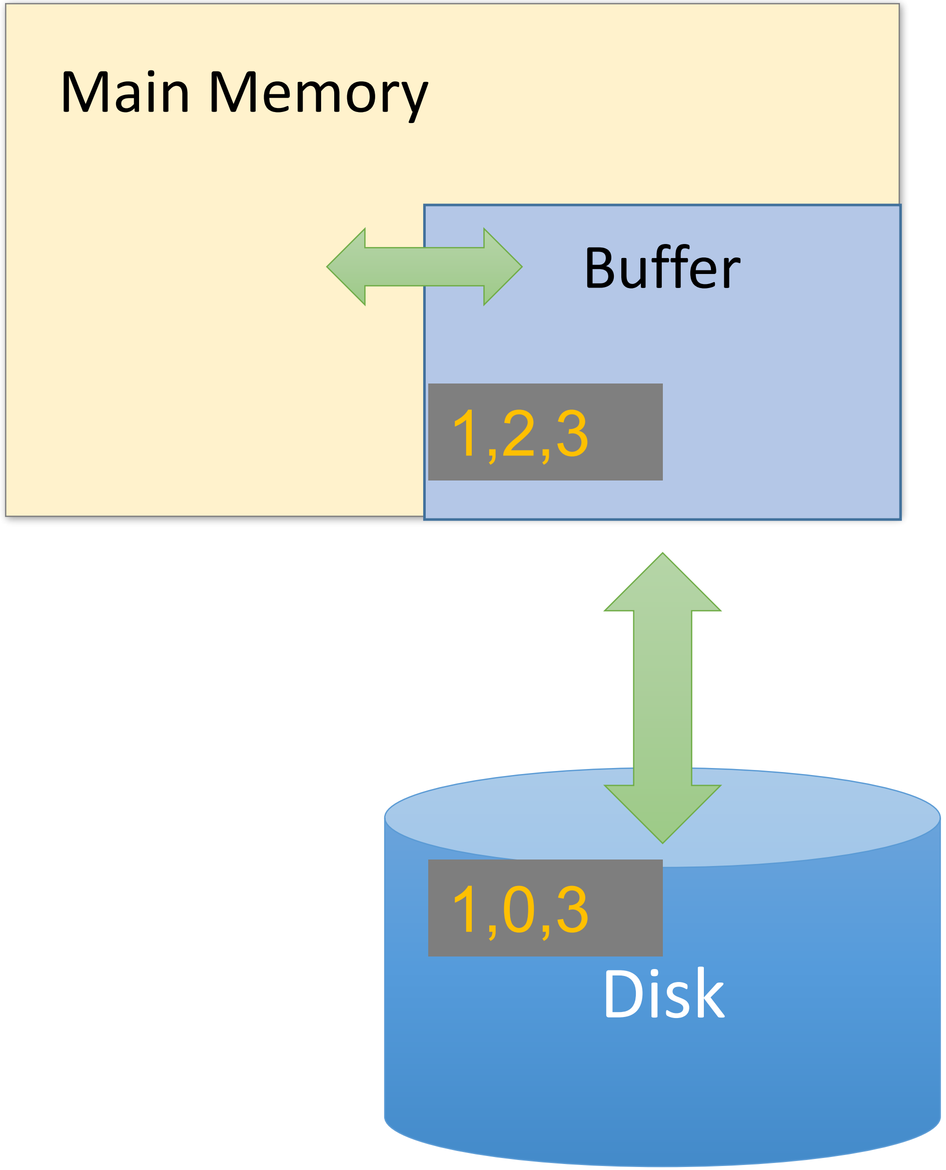
The (Simplified) Buffer

- In this class: We'll consider a buffer located in **main memory** that operates over **pages** and **files**:

  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*

  - **<u>Flush(page):</u>** Evict page from buffer & write to disk

  - **<u>Release(page):</u>** Evict page from buffer *without* writing to disk



When a Page is Requested ...

- Buffer pool information “table” contains:
  <frame#, pageid, pin_count, dirty>

1. If requested page is not in pool:
   a. Choose a frame for *replacement.*
      Only “un-pinned” pages are candidates!
   b. If frame “dirty”, write current page to disk
   c. Read requested page into frame

2. *Pin* the page and return its address.

If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!

After Requestor Finishes

- Requestor of page must:
  1. indicate whether page was modified via *dirty* bit.
  2. *unpin* it (soon preferably!) why?

- Page in pool may be requested many times,
  - a *pin count* is used.
  - To pin a page: pin_count++
  - A page is a candidate for replacement iff
    pin count == 0 ("unpinned")

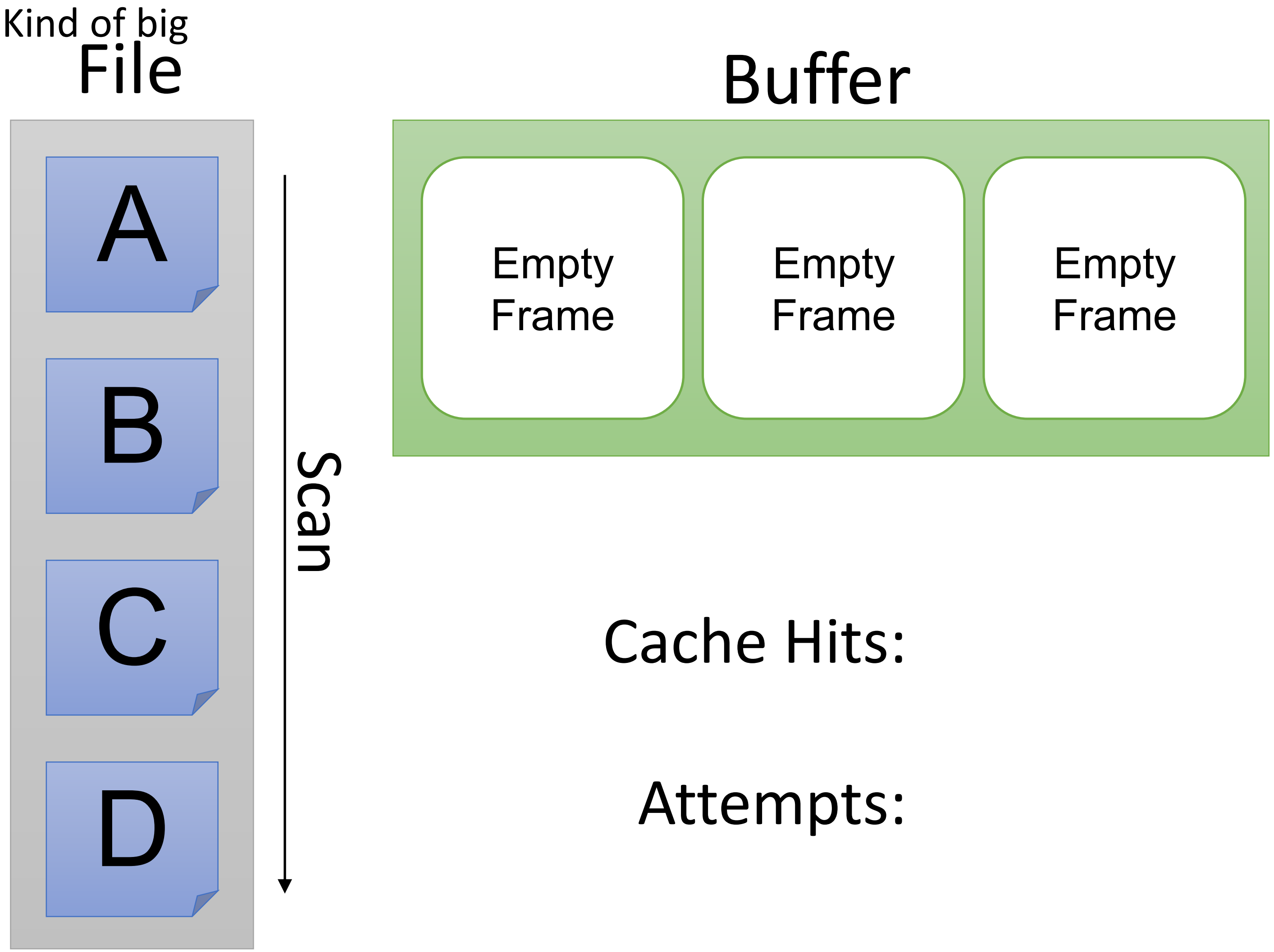
- CC & recovery may do additional I/Os upon replacement.
  - *Write-Ahead Log* protocol; more later!

Page Replacement Policy

- Page is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), Clock
  - Most-recently-used (MRU)

- Policy can have big impact on #I/O’s;
  - Depends on the *access pattern.*

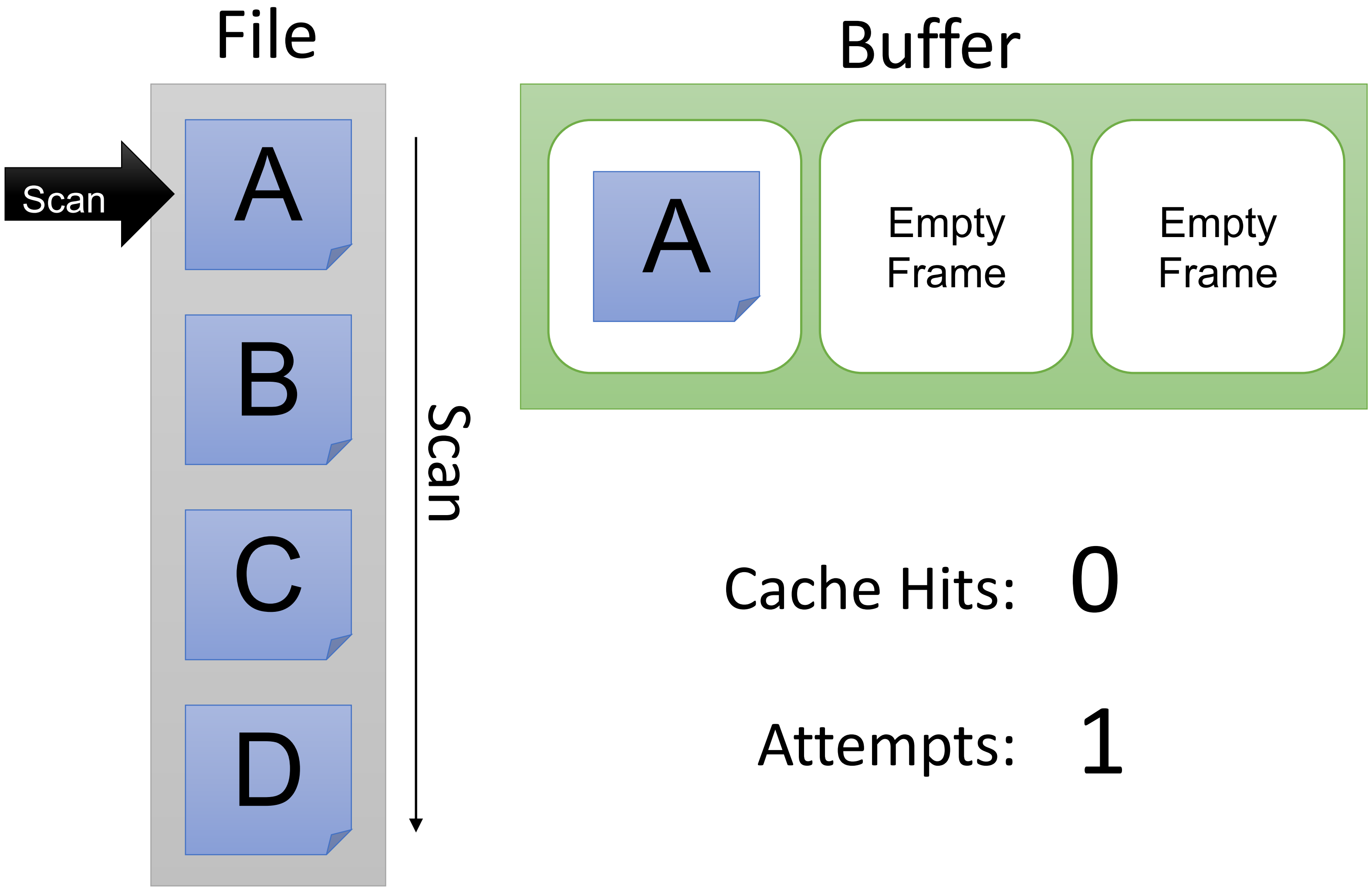
Repeated Scan of Big File (LRU)



Cache Hits:

Attempts:

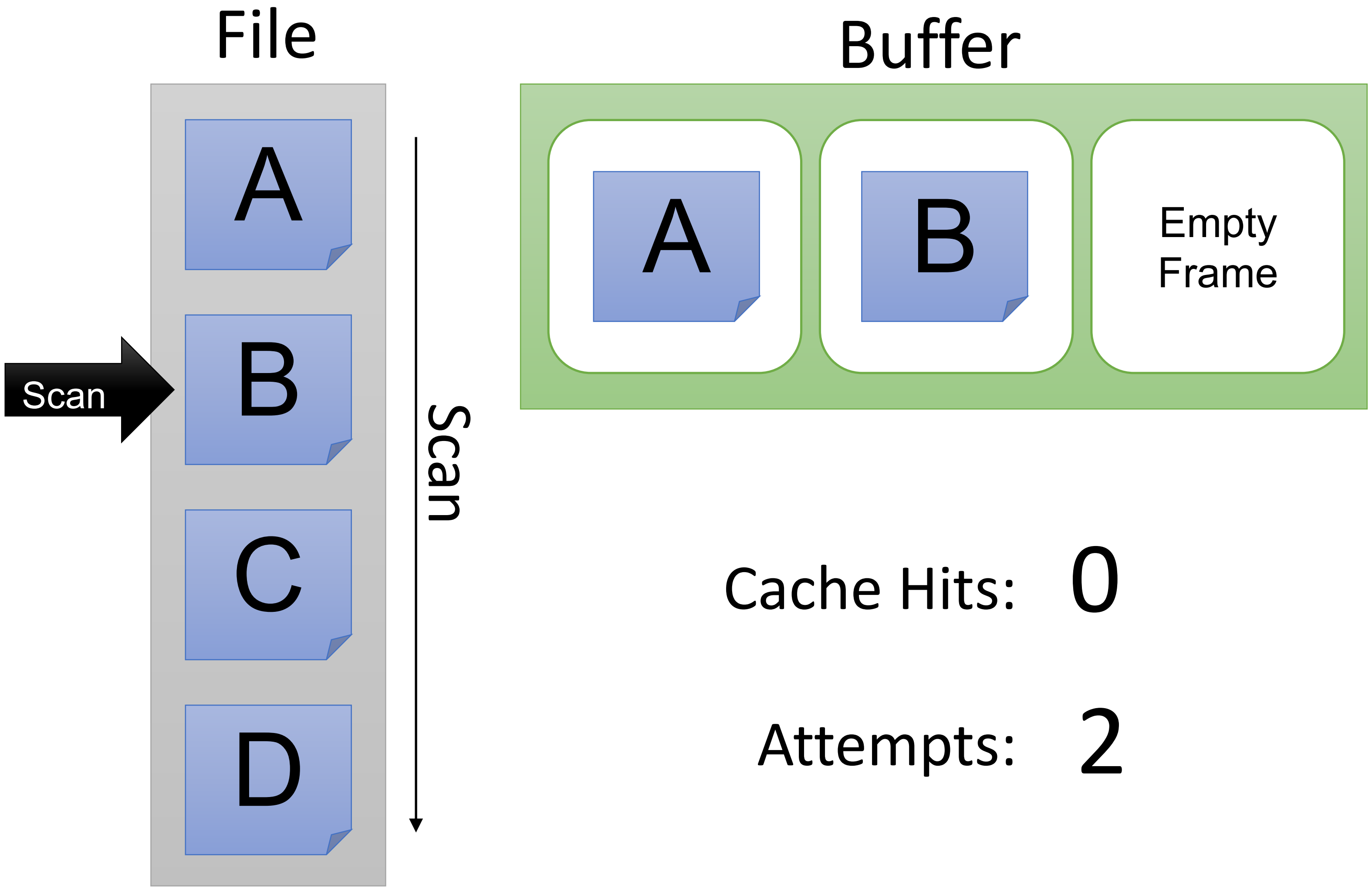
Repeated Scan of Big File (LRU)



Cache Hits: 0

Attempts: 1

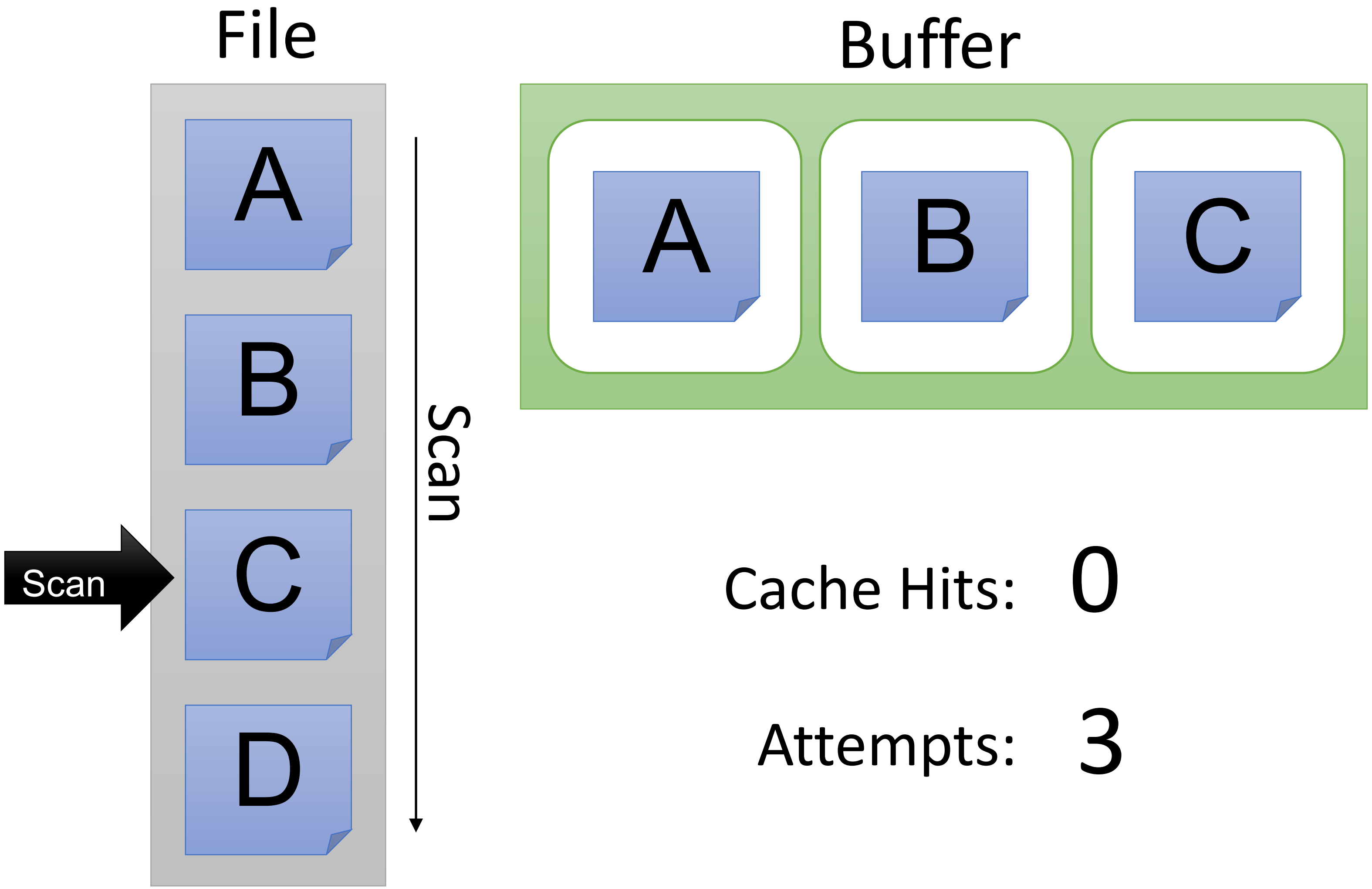
Repeated Scan of Big File (LRU)



Cache Hits: 0

Attempts: 2

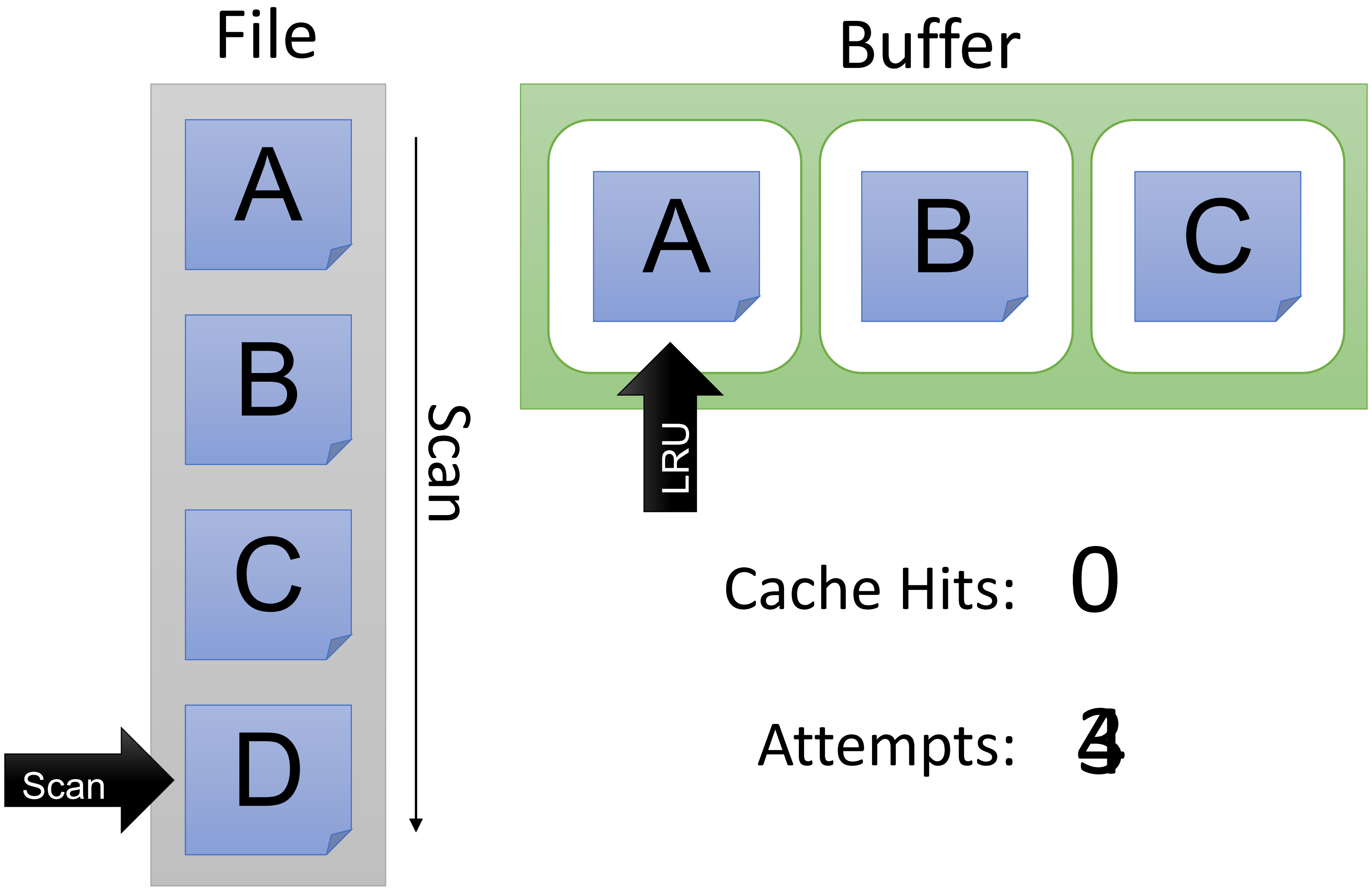
Repeated Scan of Big File (LRU)



Cache Hits: 0

Attempts: 3

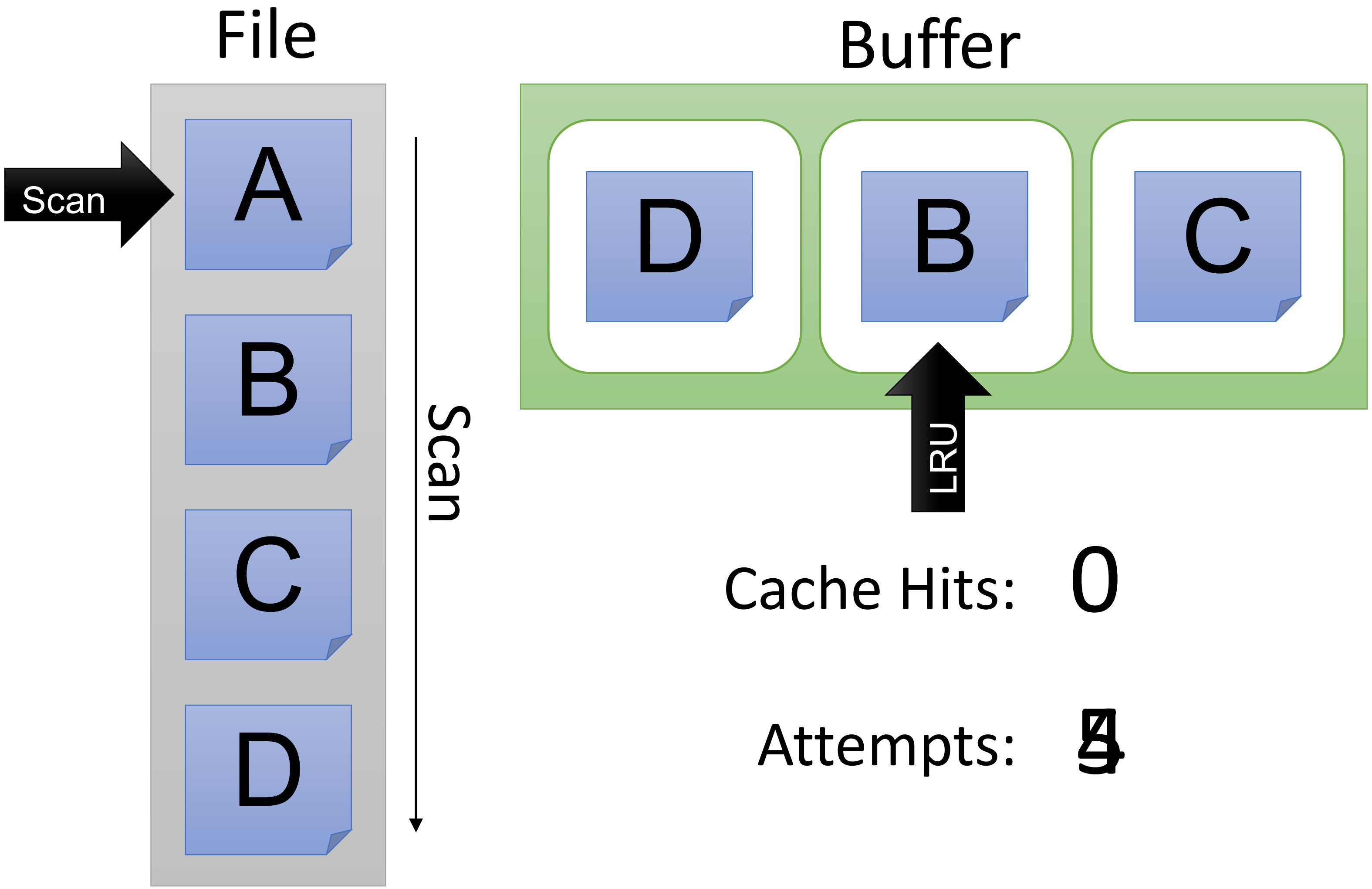
Repeated Scan of Big File (LRU)



Cache Hits: 0

Attempts: 4

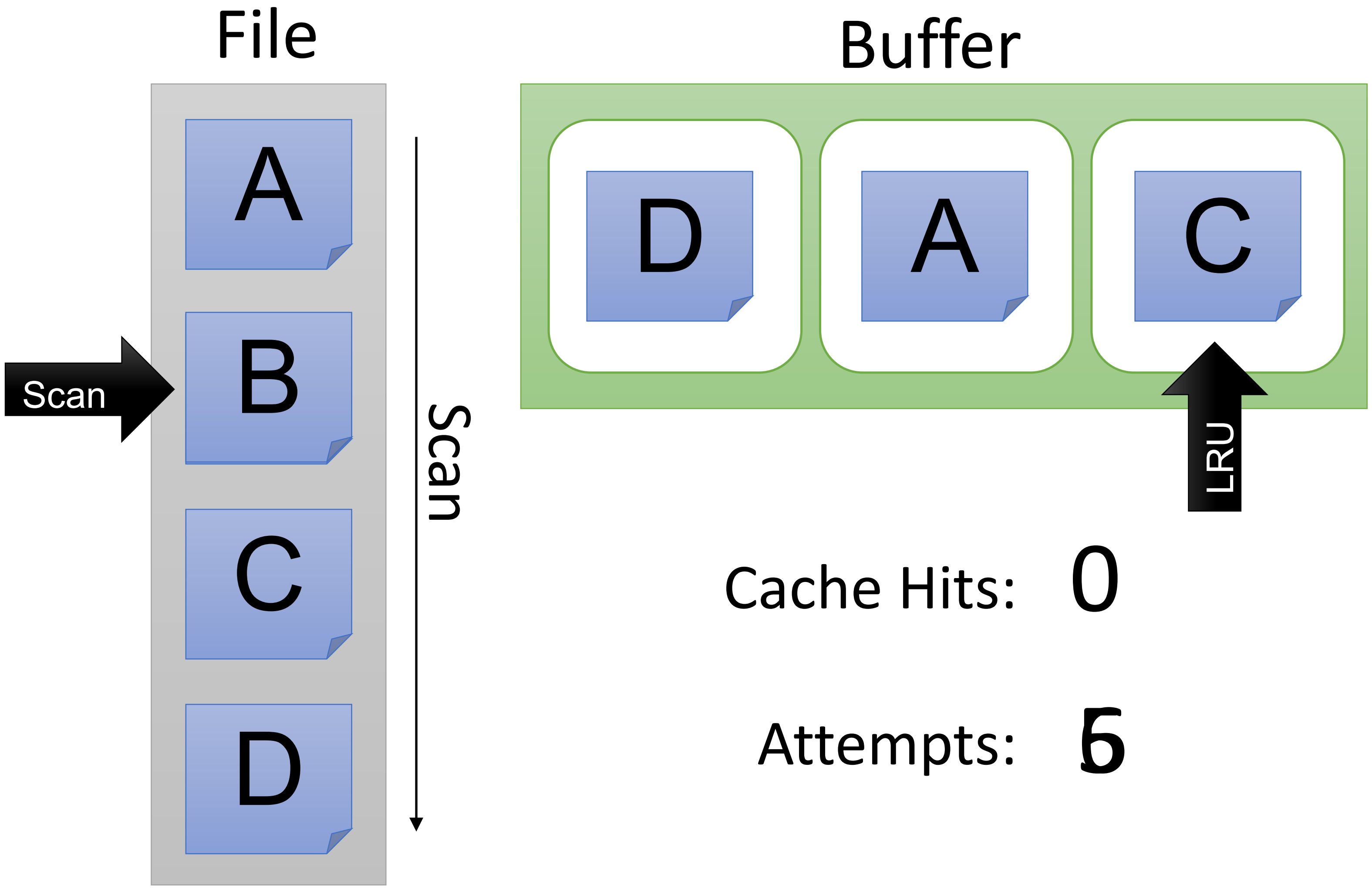
Repeated Scan of Big File (LRU)



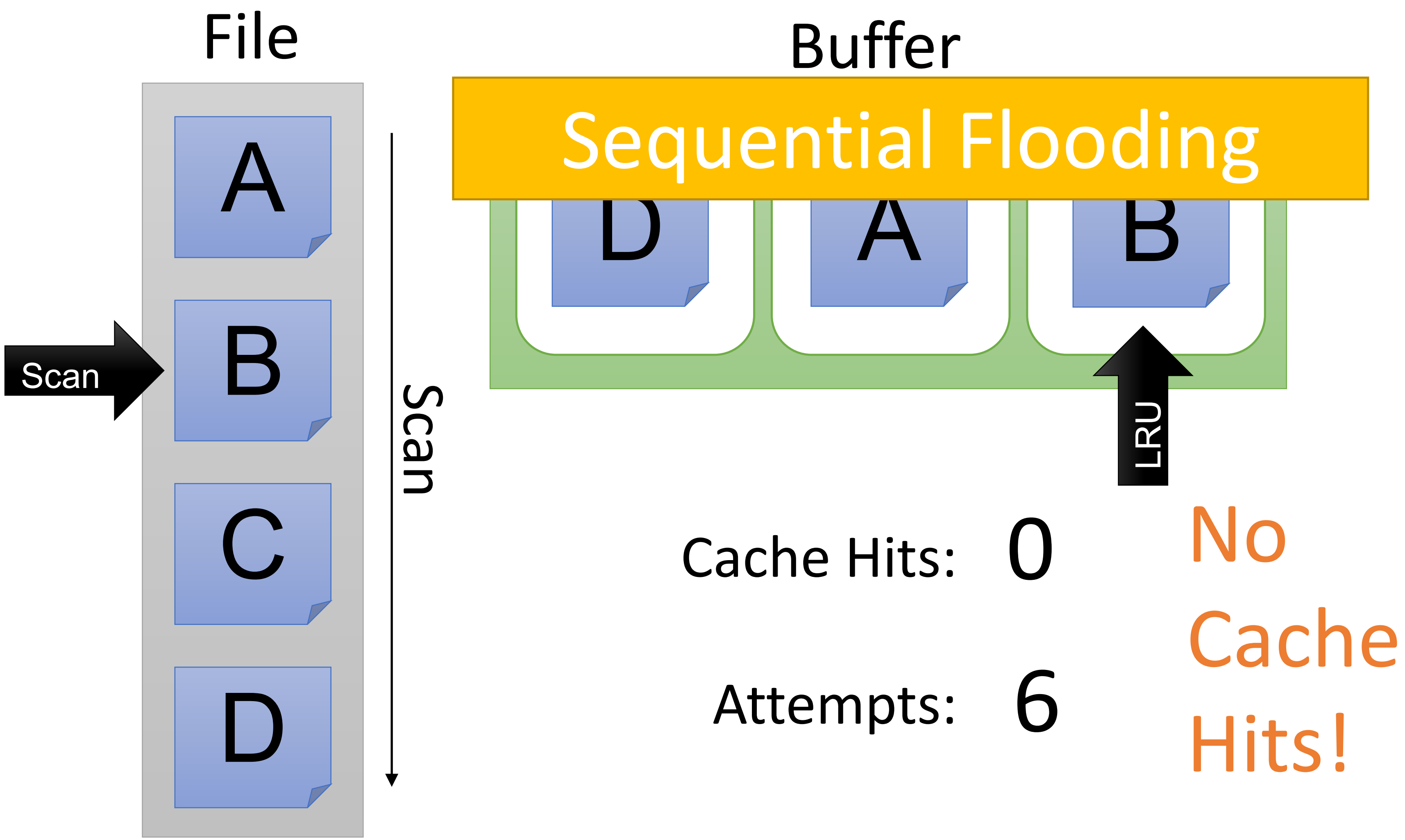
Cache Hits: 0

Attempts: 4

Repeated Scan of Big File (LRU)

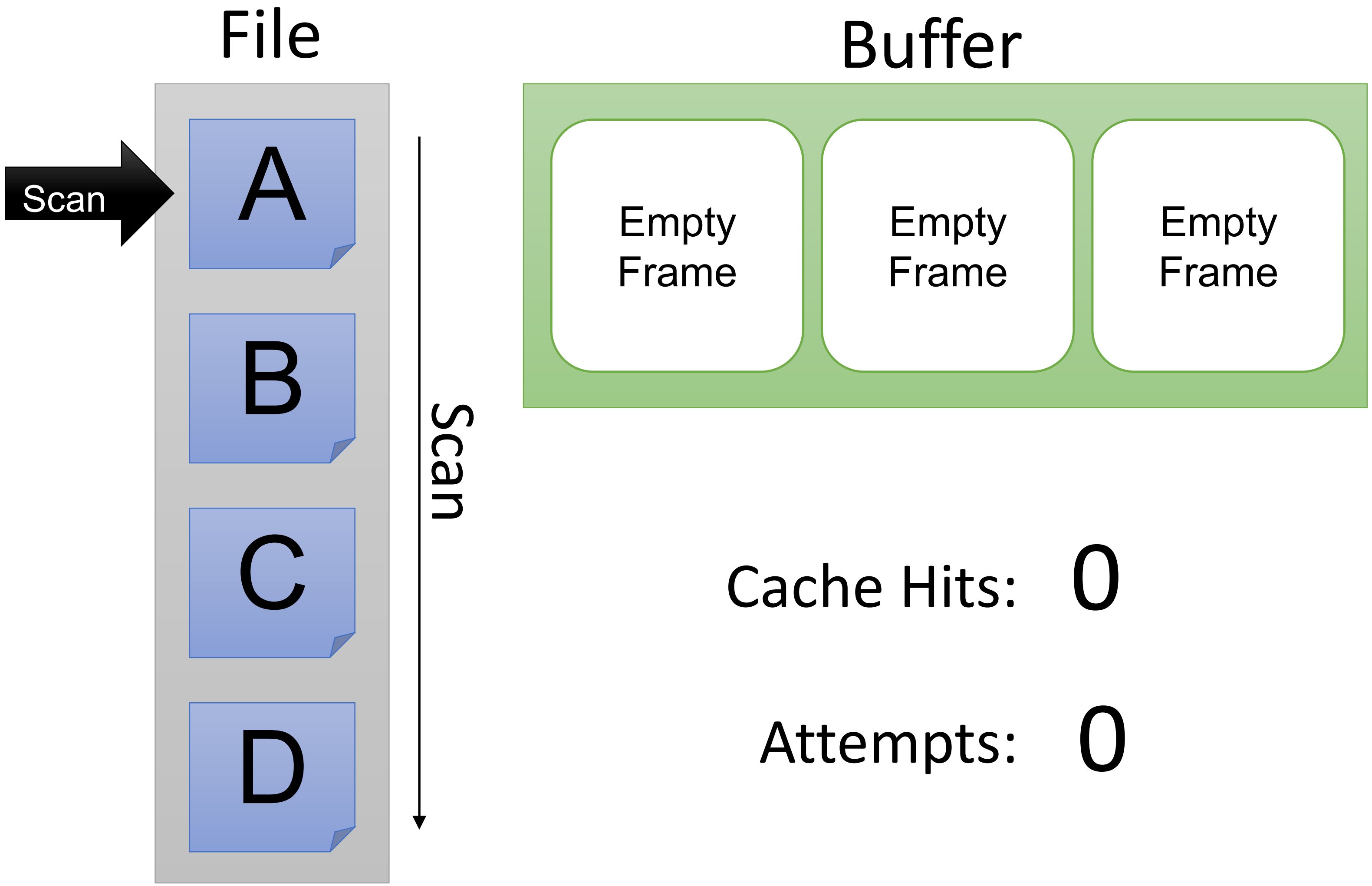


Repeated Scan of Big File (LRU)



Repeated Scan of Big File **(MRU)**

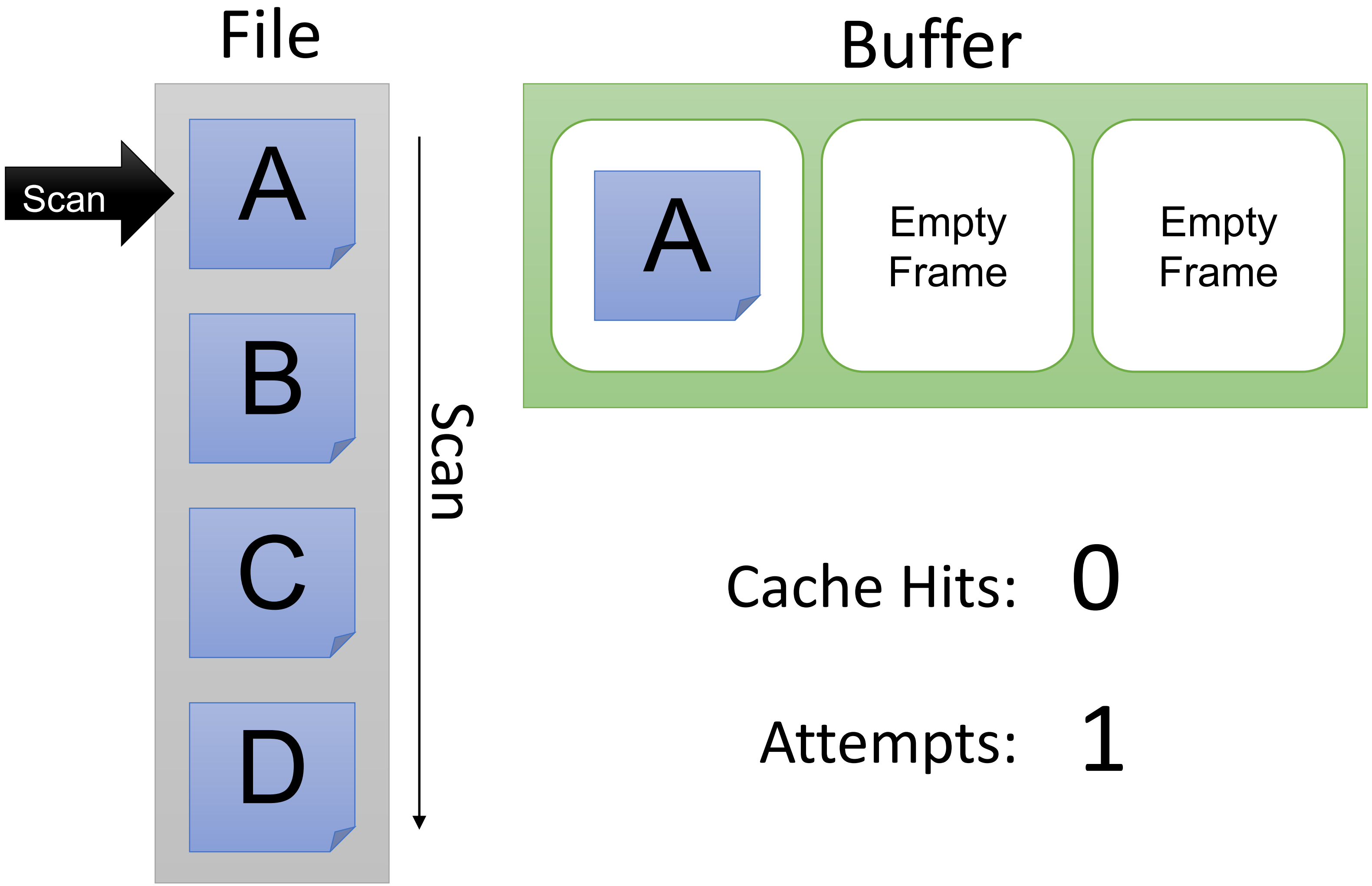
Most Recently Used



Cache Hits: 0

Attempts: 0

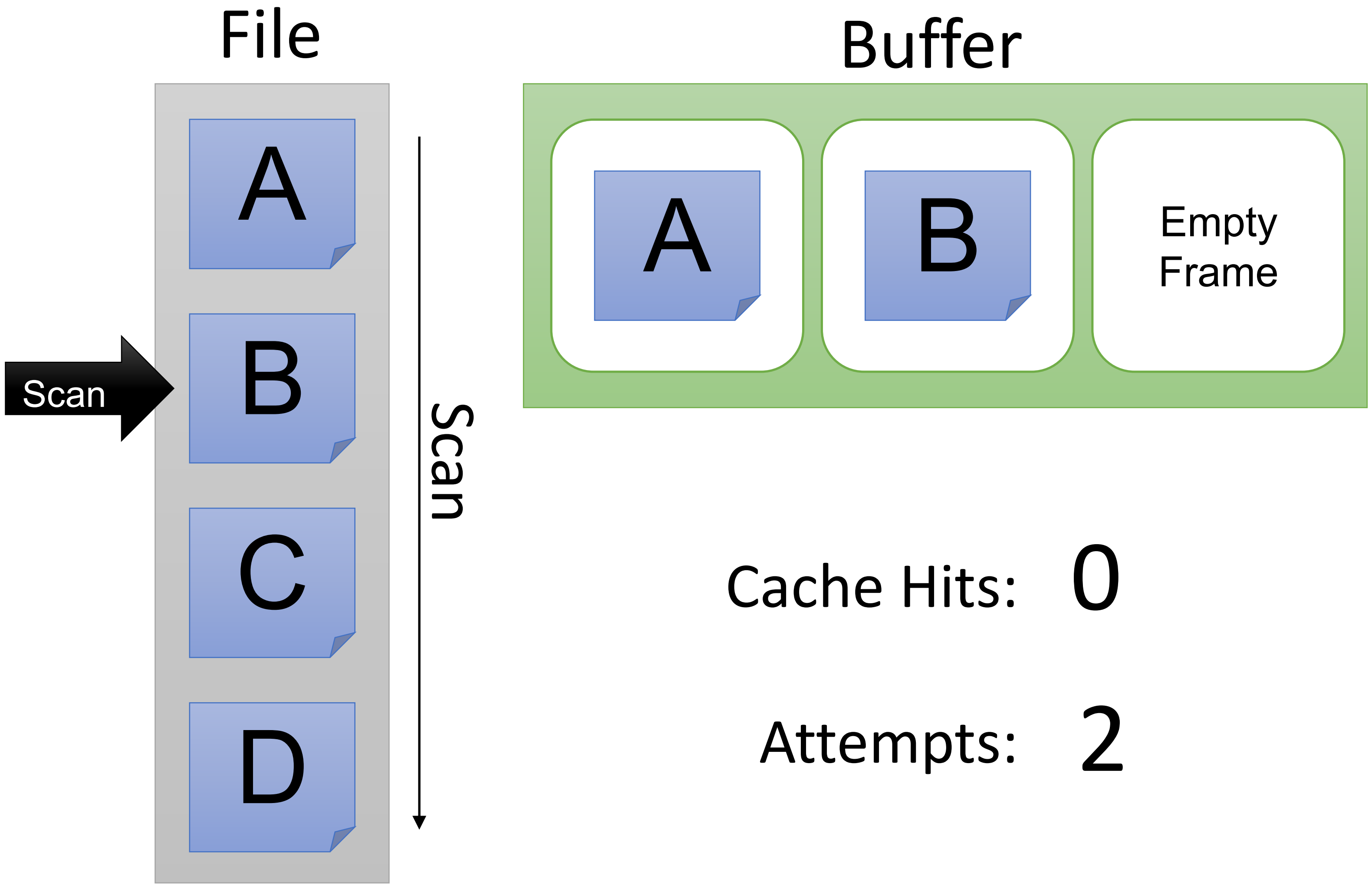
Repeated Scan of Big File (MRU)



Cache Hits: 0

Attempts: 1

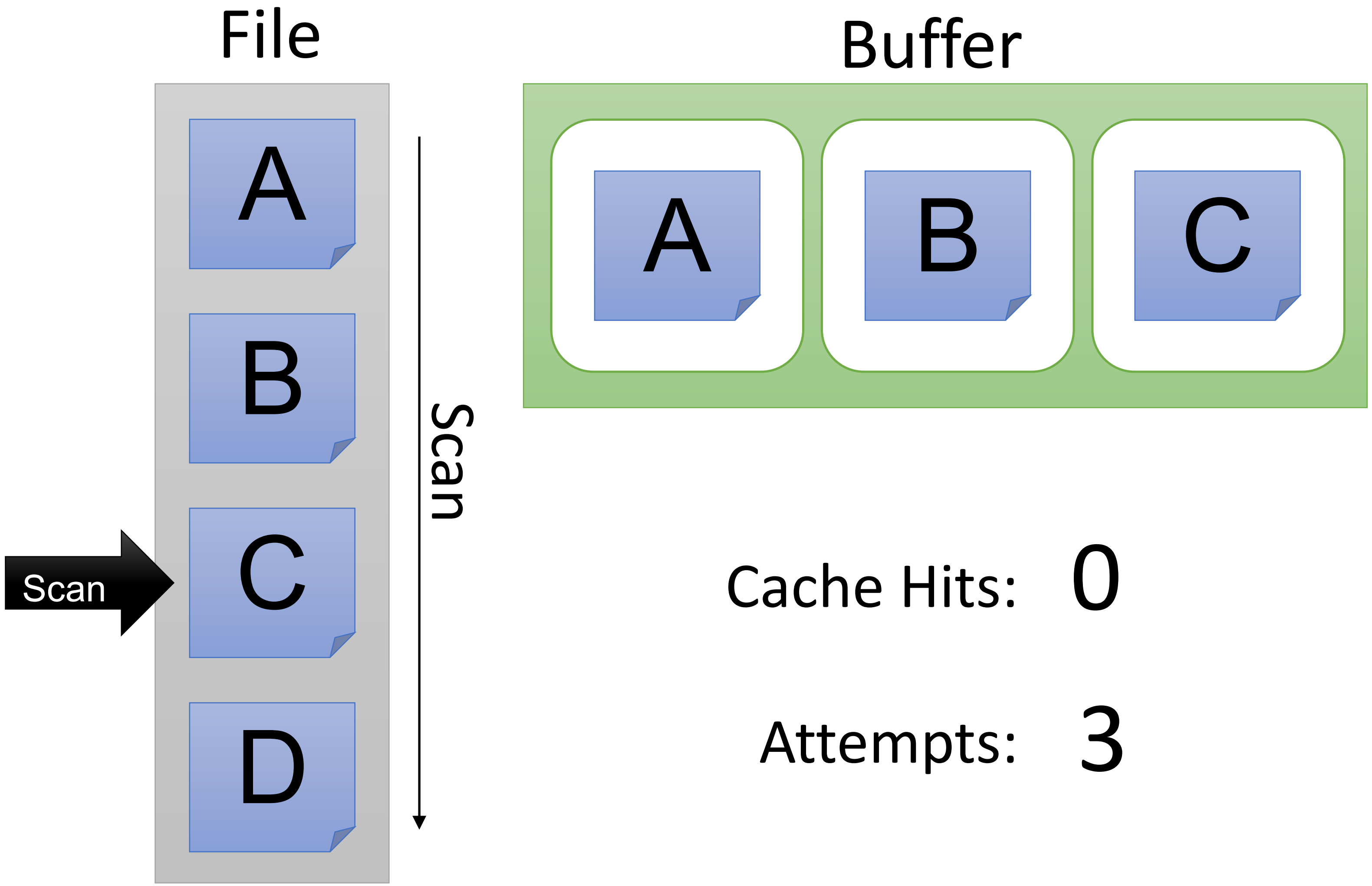
Repeated Scan of Big File (MRU)



Cache Hits: 0

Attempts: 2

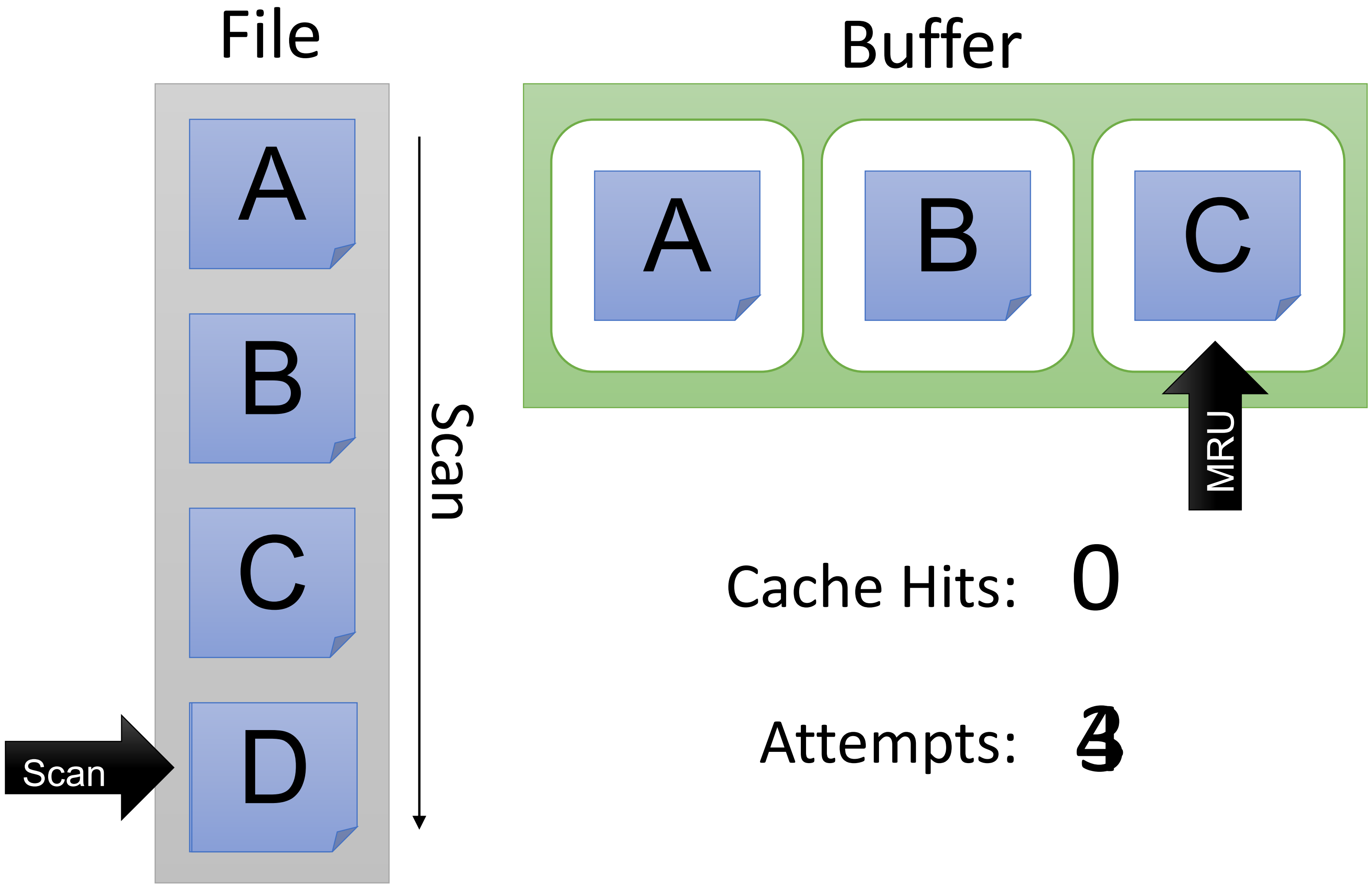
Repeated Scan of Big File (MRU)



Cache Hits: 0

Attempts: 3

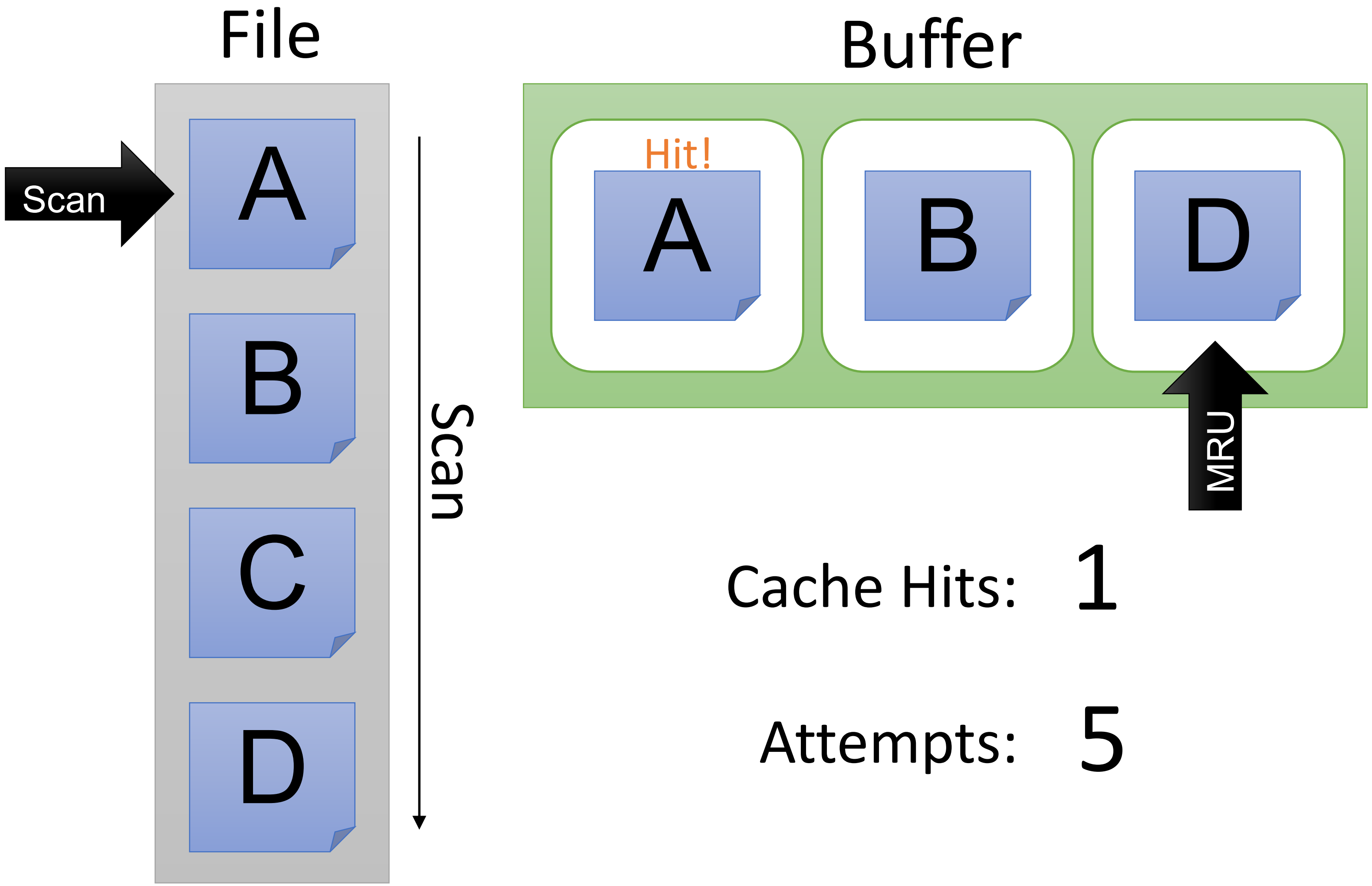
Repeated Scan of Big File (MRU)



Cache Hits: 0

Attempts: 4

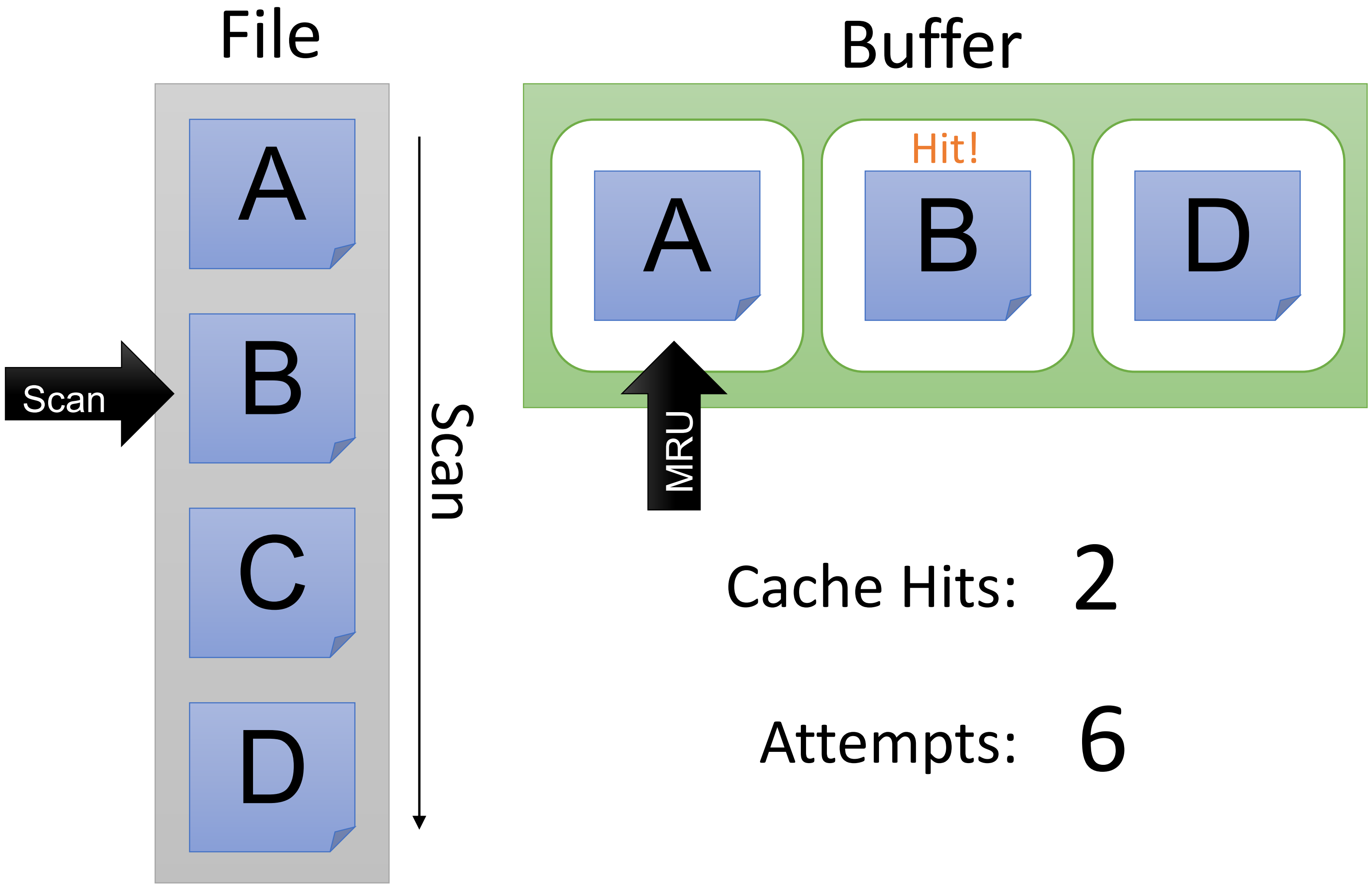
Repeated Scan of Big File (MRU)



Cache Hits: 1

Attempts: 5

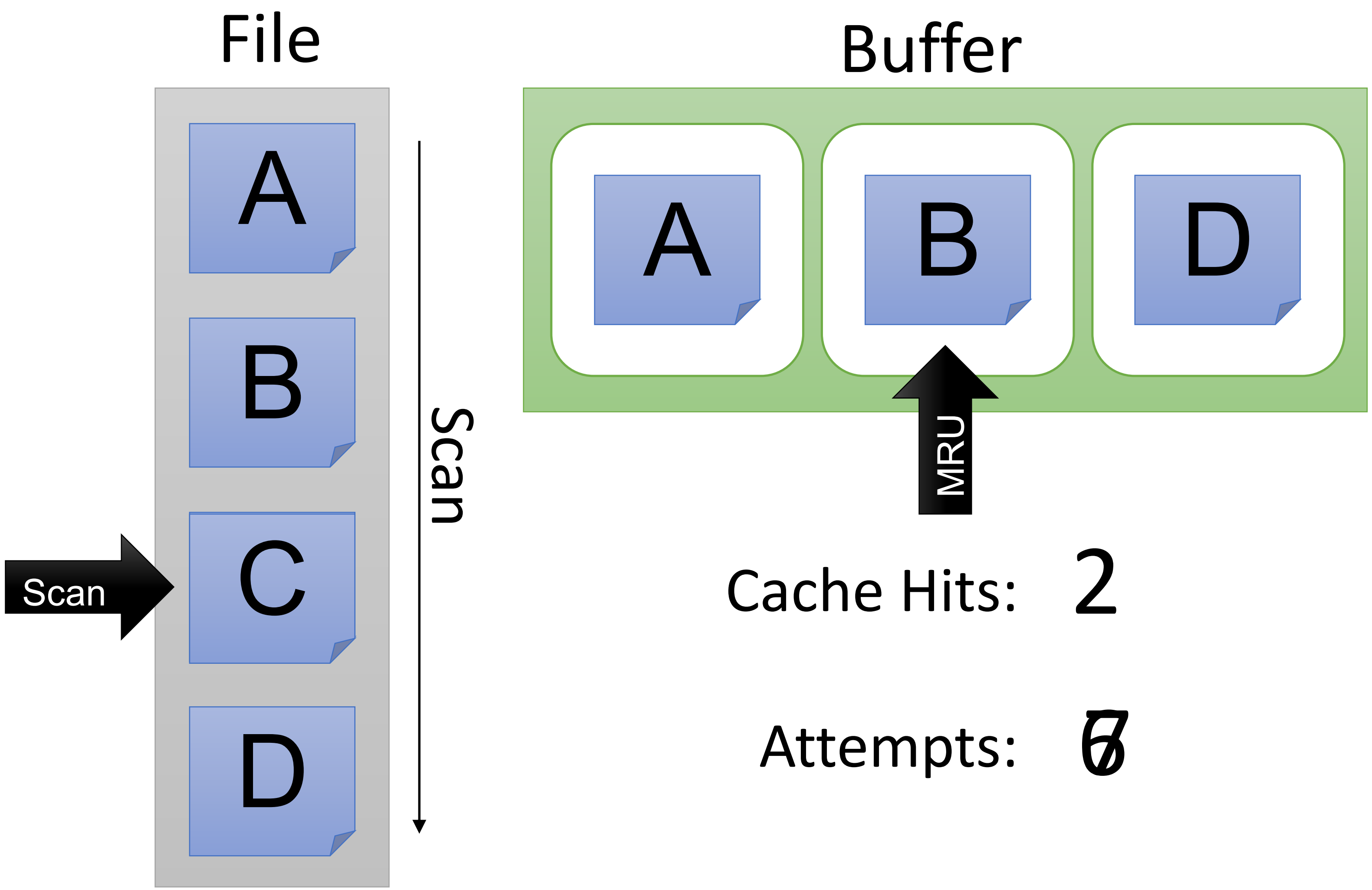
Repeated Scan of Big File (MRU)



Cache Hits: 2

Attempts: 6

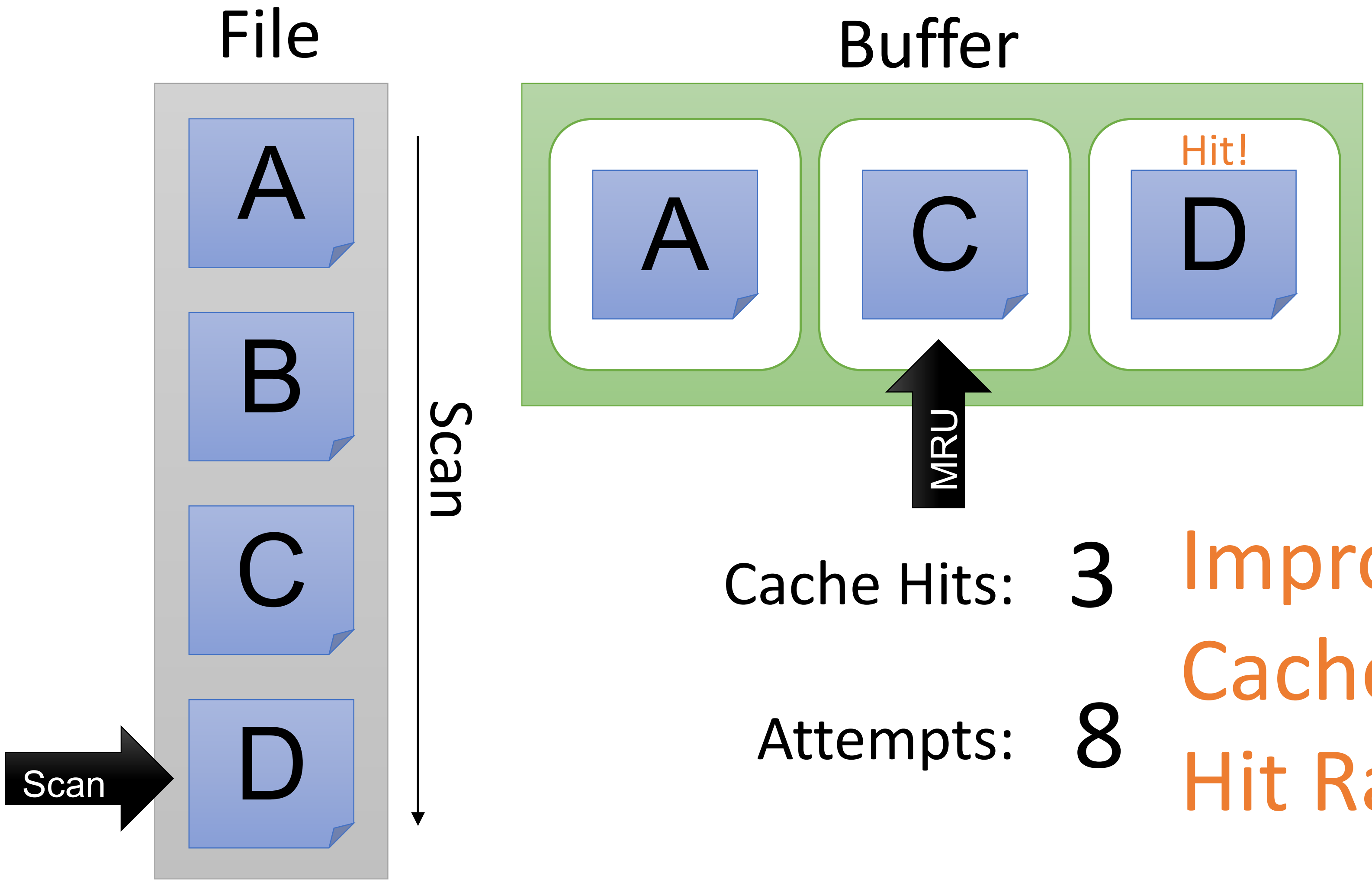
Repeated Scan of Big File (MRU)



Cache Hits: 2

Attempts: 7

Repeated Scan of Big File (MRU)

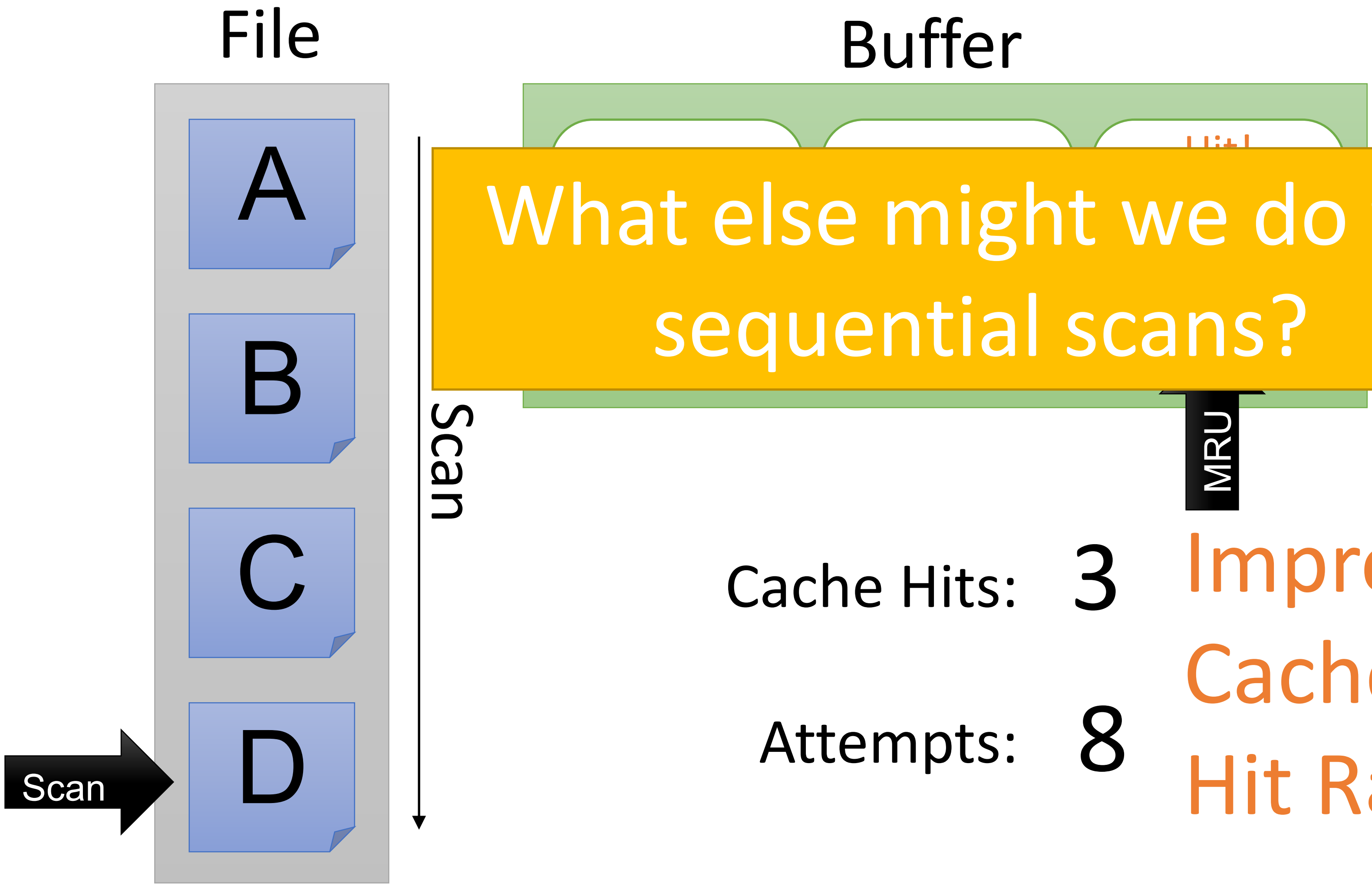


Cache Hits: 3

Attempts: 8

Improved Cache Hit Rate!

Repeated Scan of Big File (MRU)



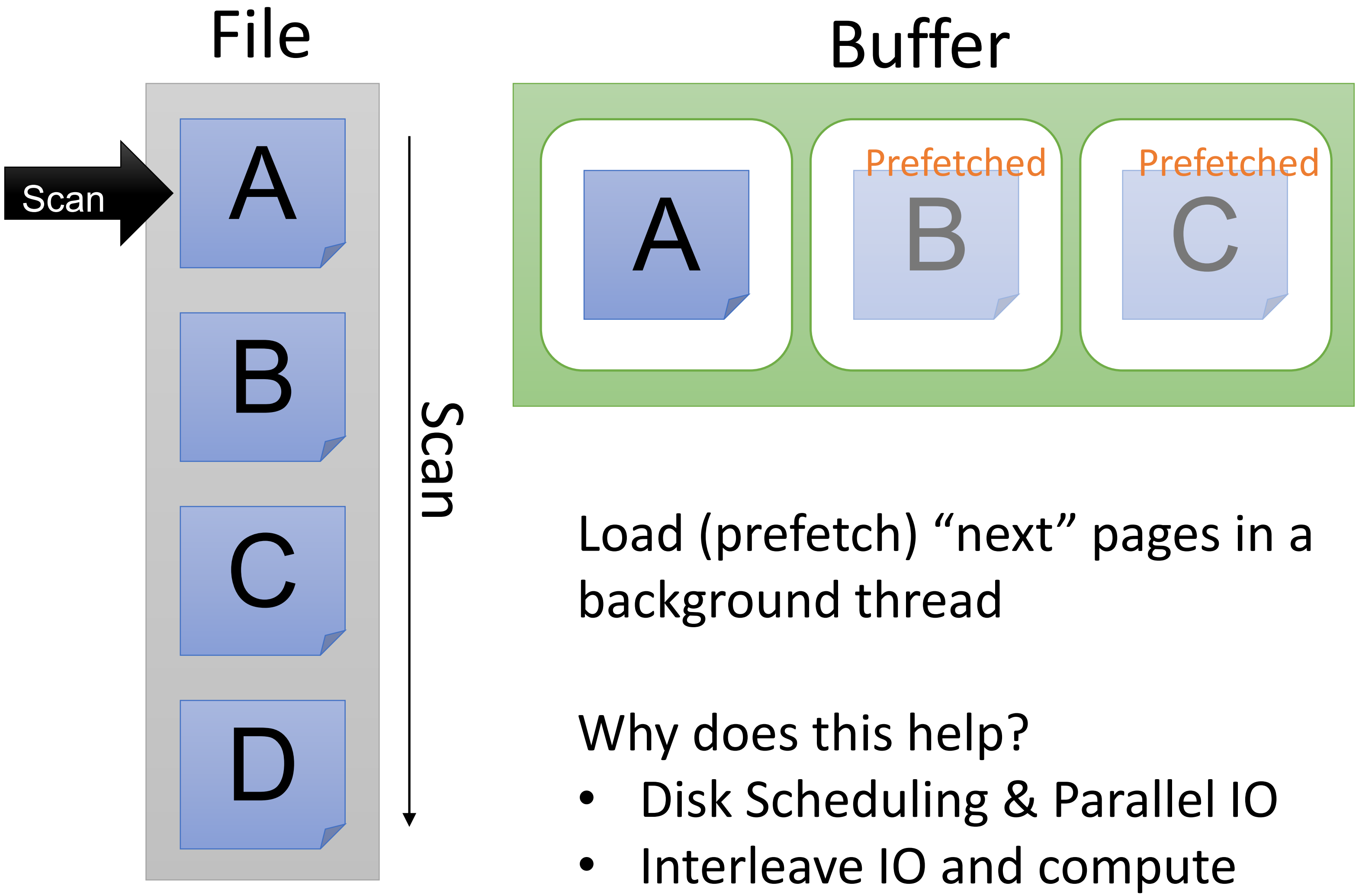
What else might we do for sequential scans?

Cache Hits: 3

Attempts: 8

Improved Cache Hit Rate!

Background Prefetching



Load (prefetch) “next” pages in a background thread

Why does this help?

- Disk Scheduling & Parallel IO
- Interleave IO and compute

The Buffer Manager

- A **buffer manager** handles supporting operations for the buffer:
  - Primarily, handles & executes the “replacement policy”
    - i.e. finds a page in buffer to flush/release if buffer is full and a new page needs to be read in
  - DBMSs typically implement their own buffer management routines