Buffer Management

Concepts

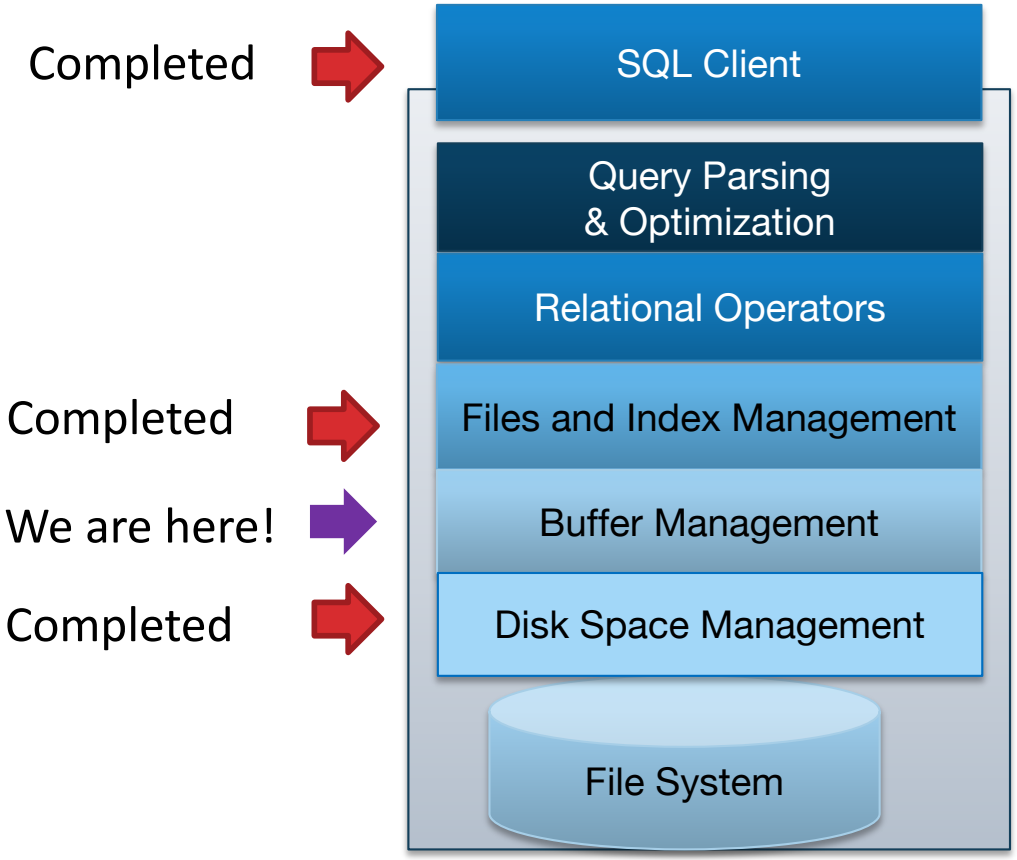
Replacement Policies

Alvin Cheung

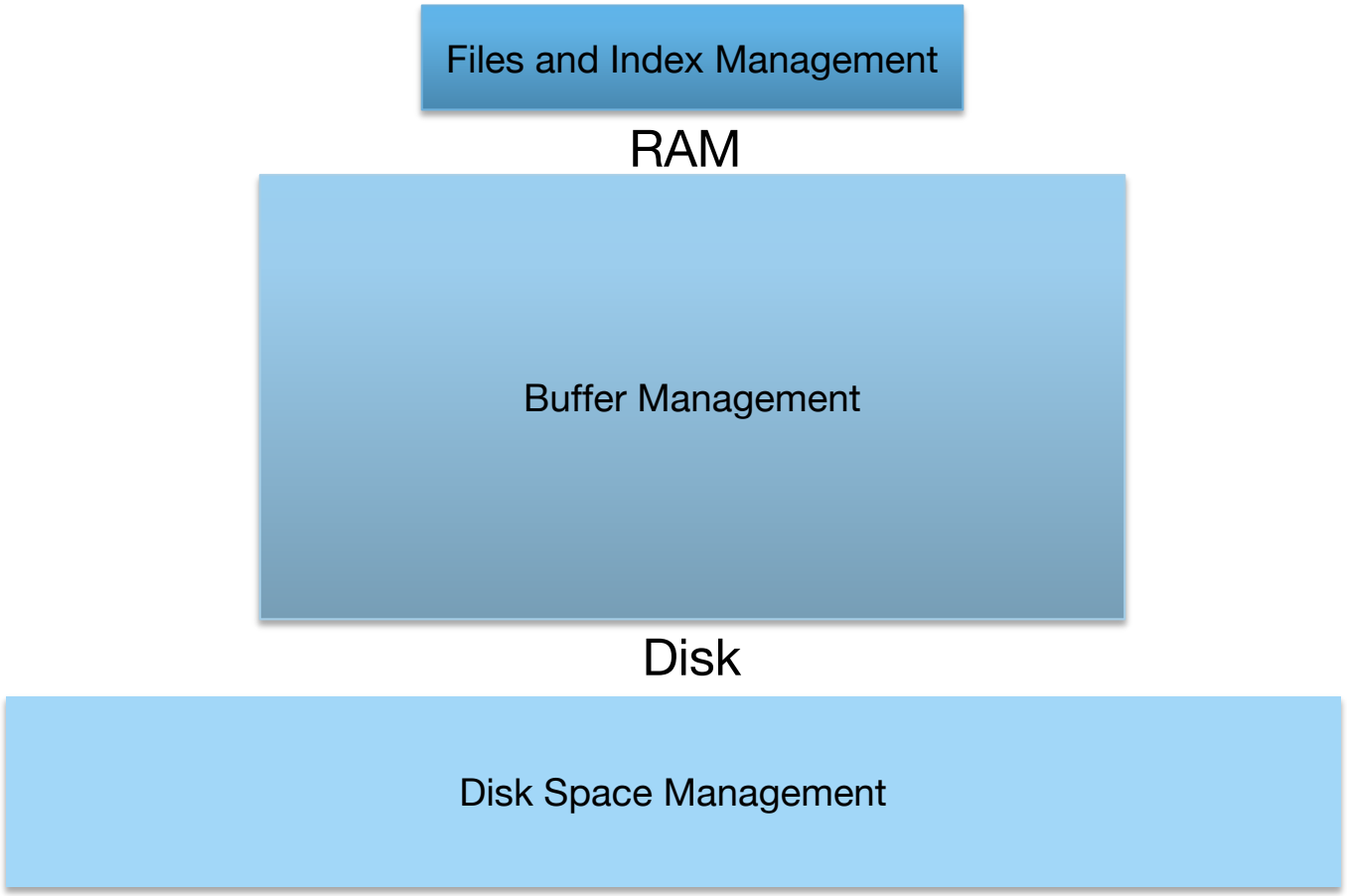
Aditya Parameswaran

Reading: R & G Chapter 9.4

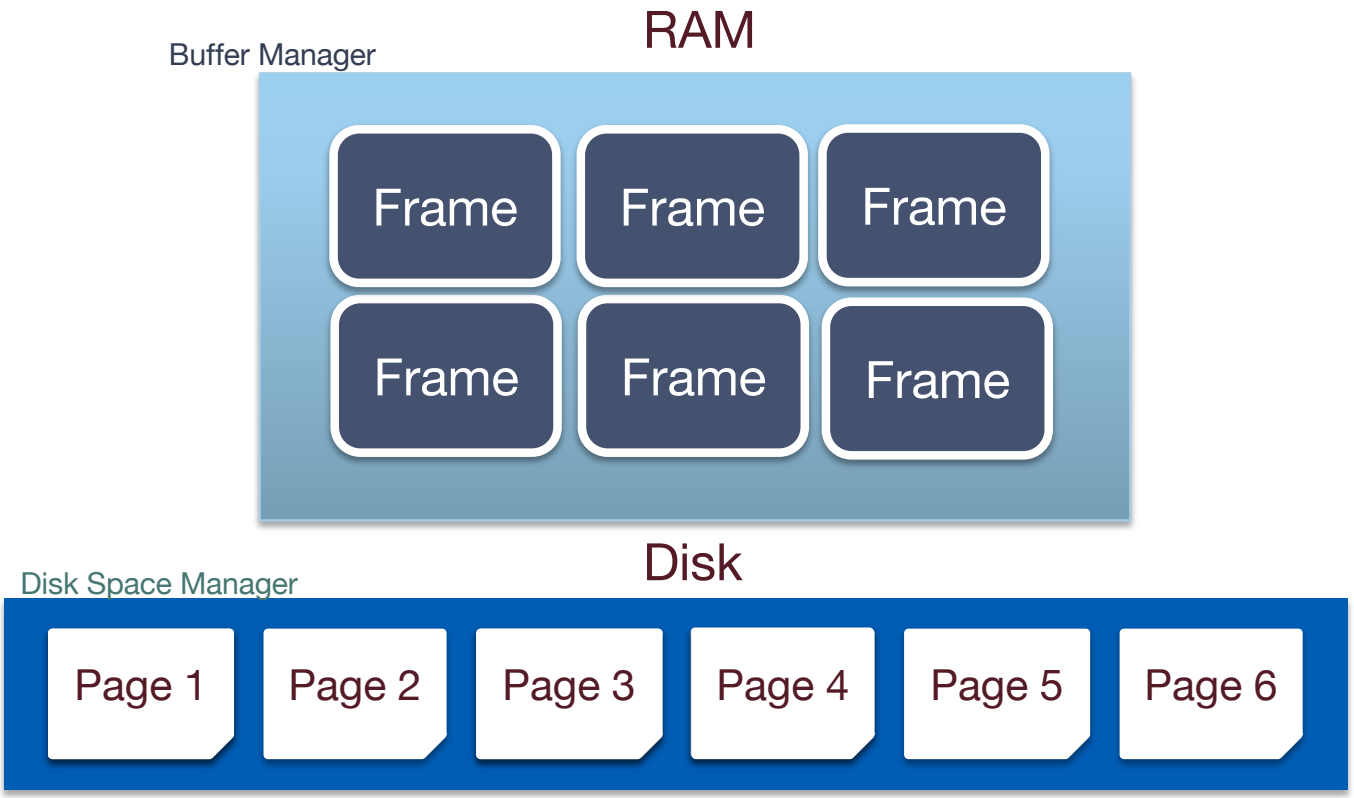
Architecture of a DBMS: What we've learned



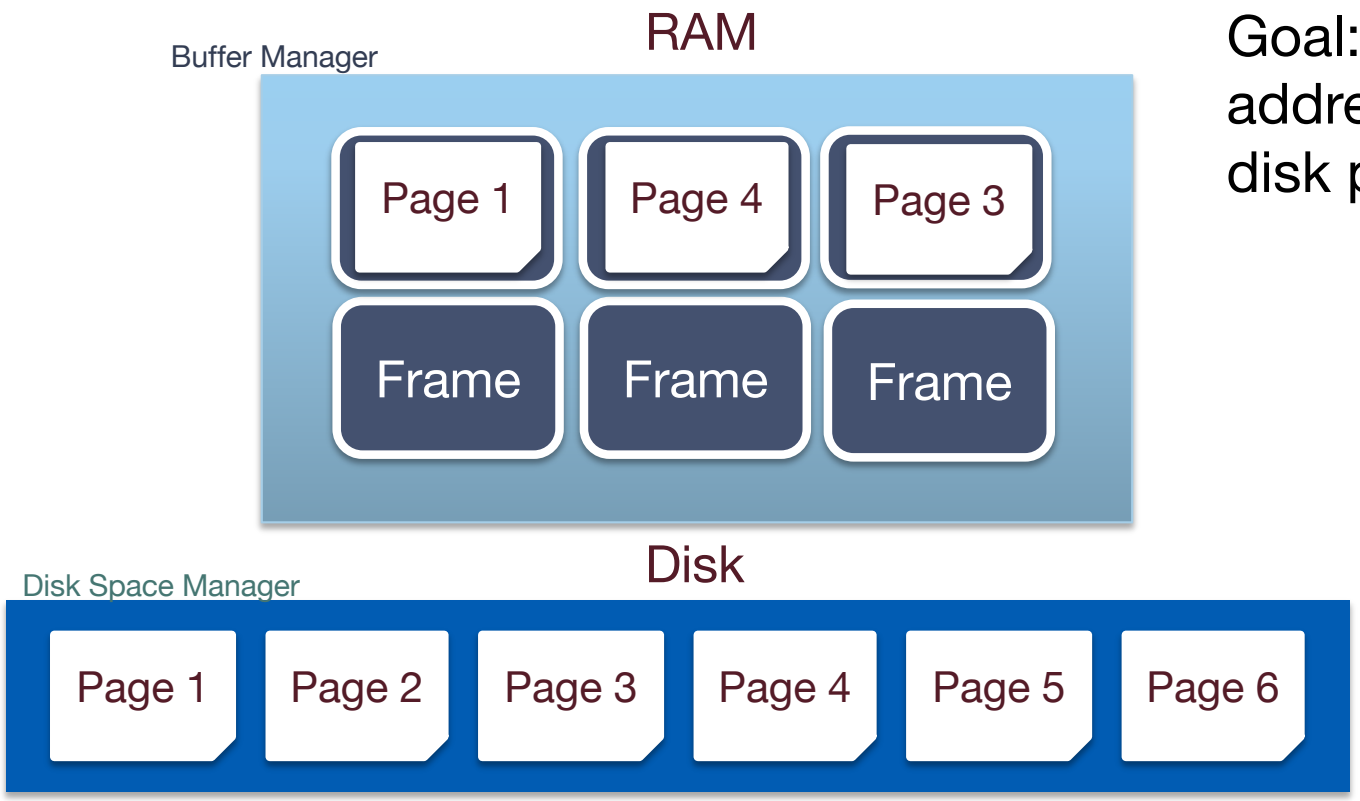
Buffer Management Levels of Abstraction



Buffer Management

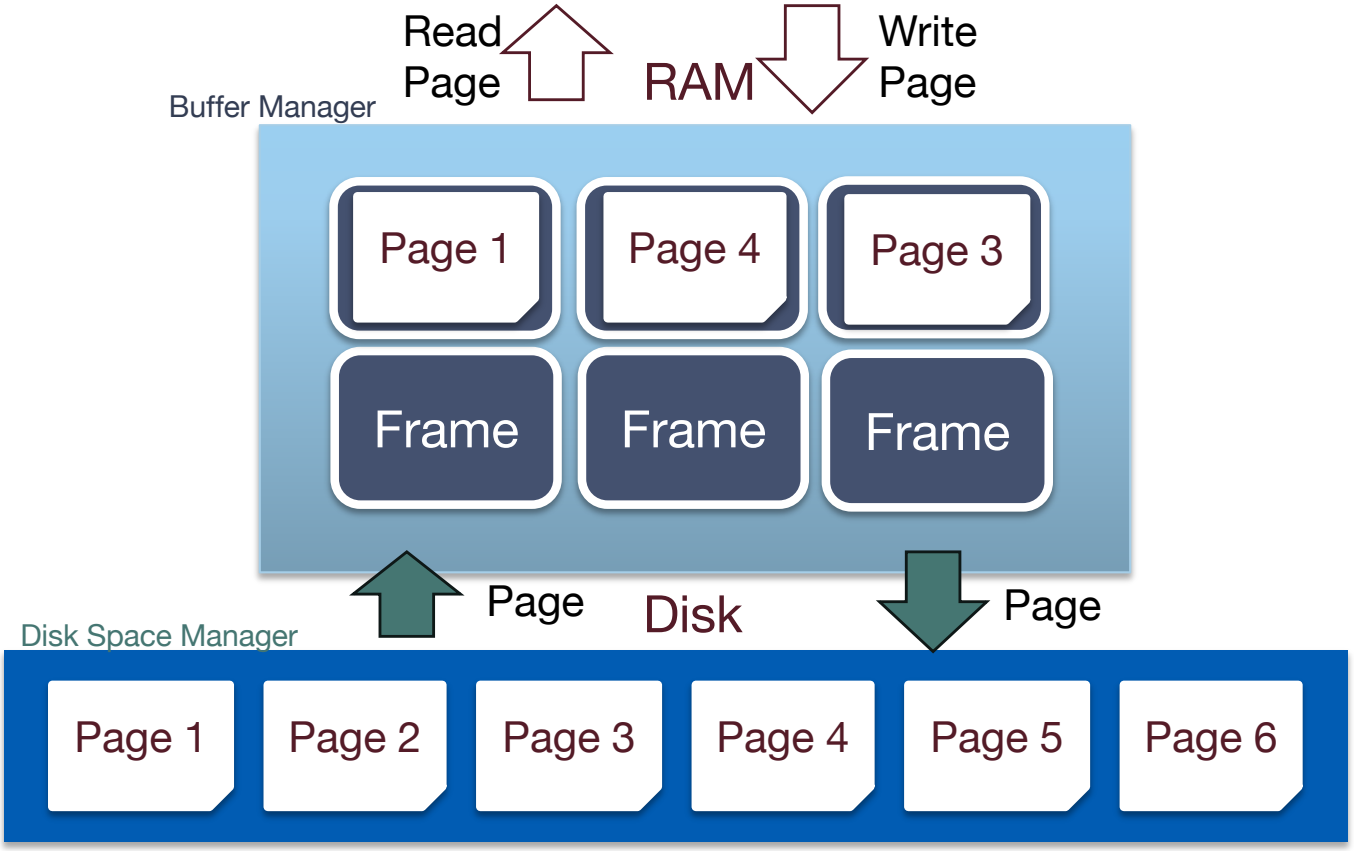


Buffer Management

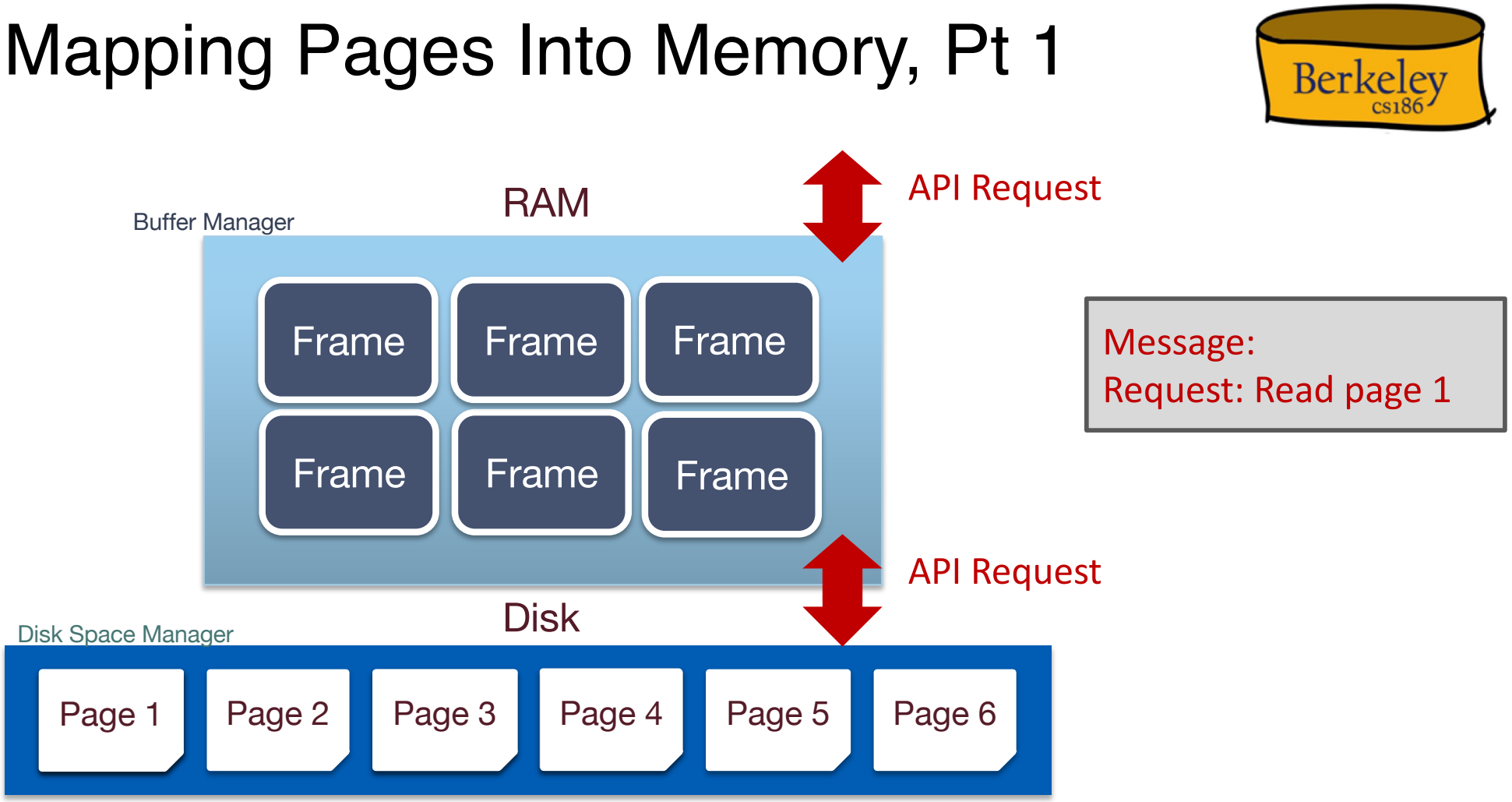


Goal: Create the illusion of addressing and modifying disk pages **in memory.**

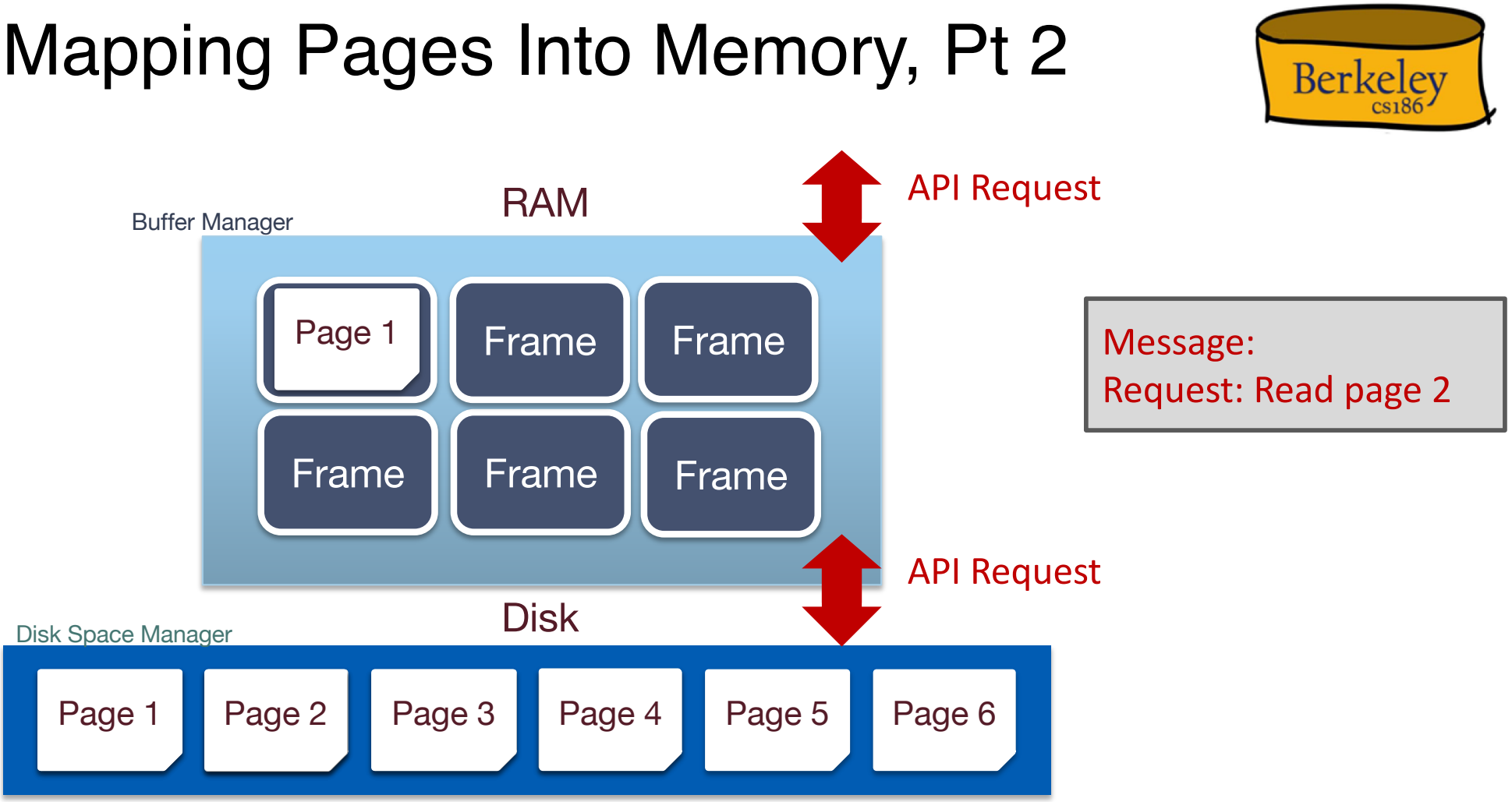
APIs


Read
Page
RAM
Write
Page
Buffer Manager
Page 1
Page 4
Page 3
Frame
Frame
Frame
Page
Disk
Page
Disk Space Manager
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6

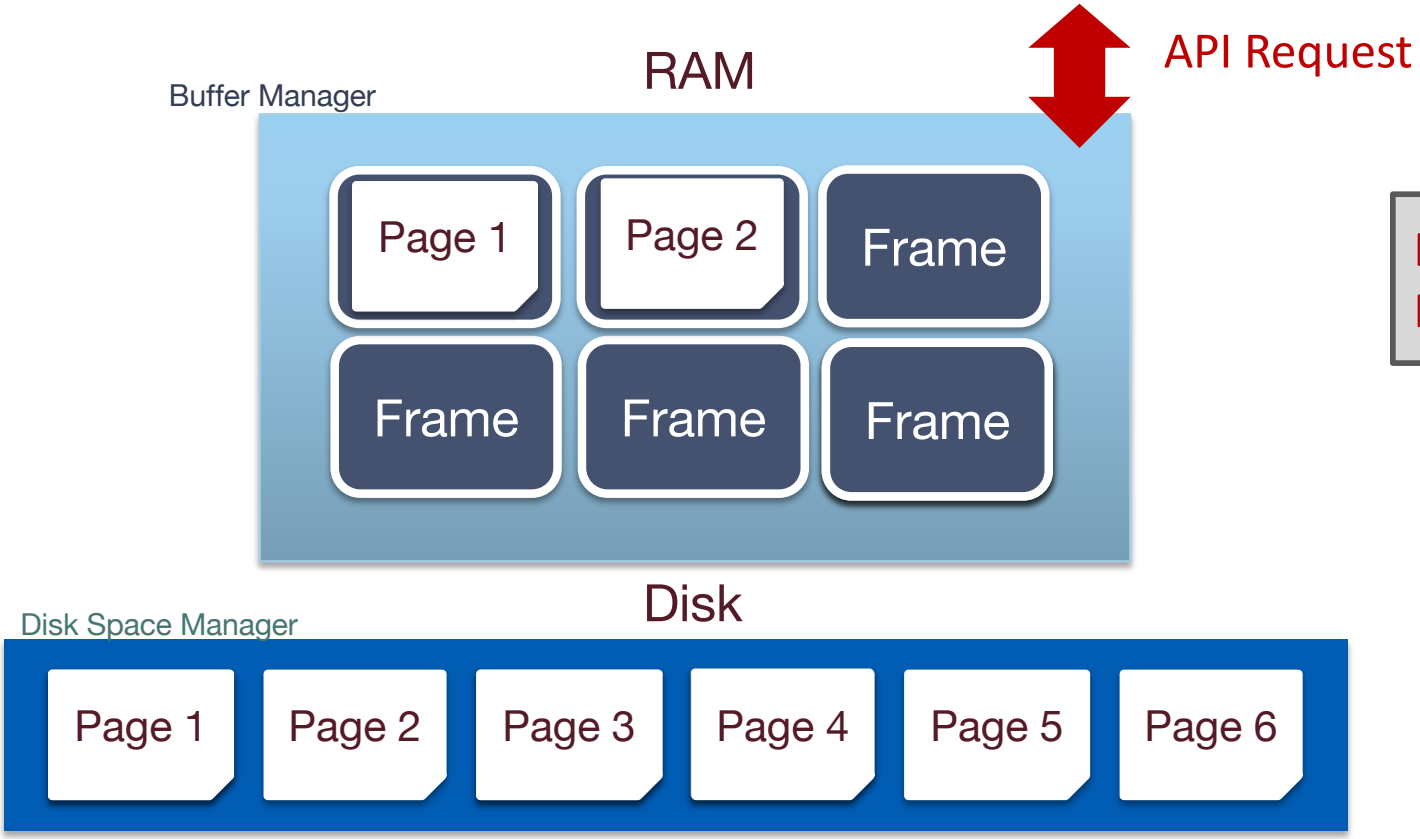
Mapping Pages Into Memory, Pt 1



Mapping Pages Into Memory, Pt 2



Mapping Pages Into Memory, Pt 3



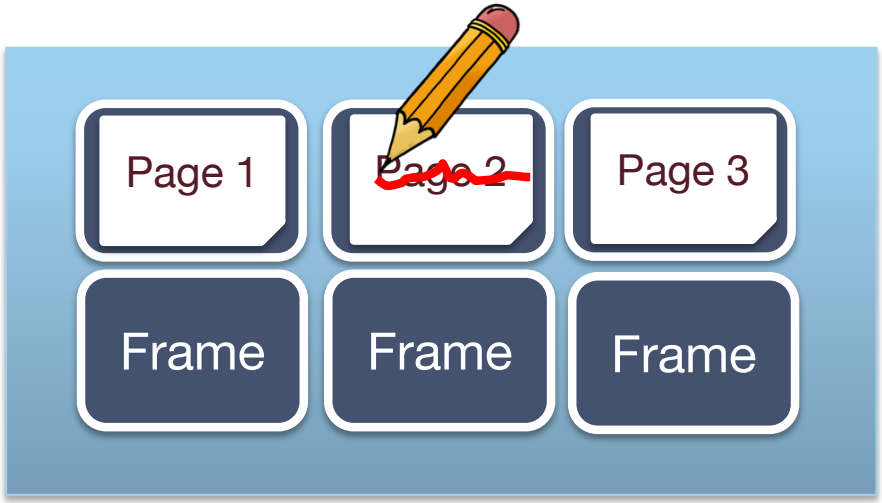
Message:
Request: Read page 2

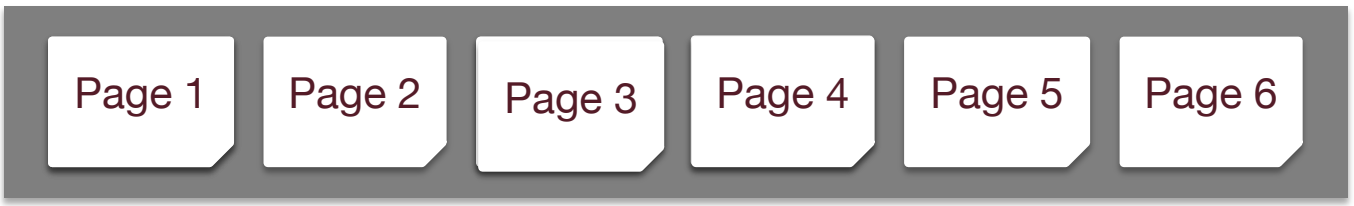
Questions We Need to Answer

1. Handling dirty pages

2. Page Replacement

Q1: Dirty Pages?



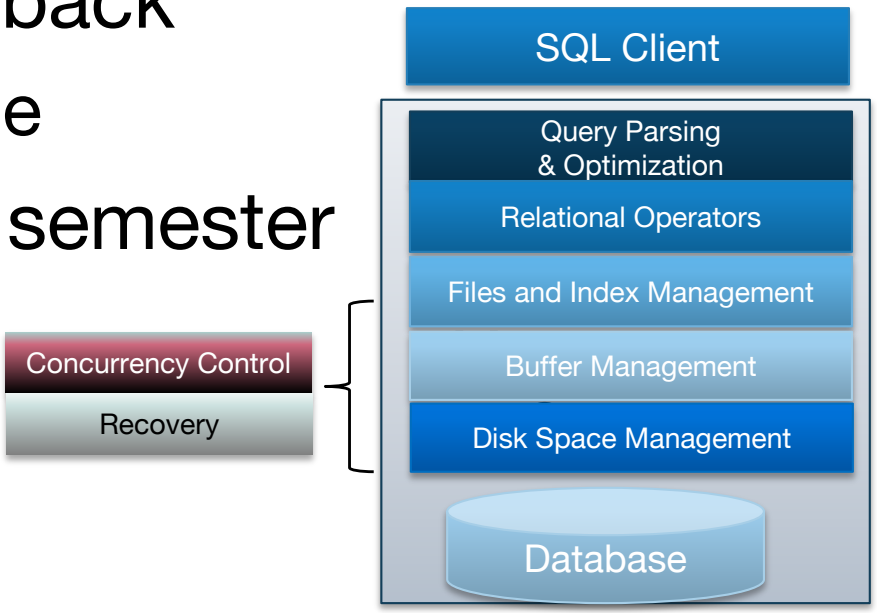


Handling Dirty Pages

- Handling dirty pages
  - How will the buffer manager find out?
    - Dirty bit on page
  - What to do with a dirty page?
    - Write back via disk manager

Advanced Questions

- Concurrent operations on a page
  - Solved by Concurrency Control module
- System Crash before write-back
  - Solved by Recovery module
- Will cover these later in the semester



Buffer Manager

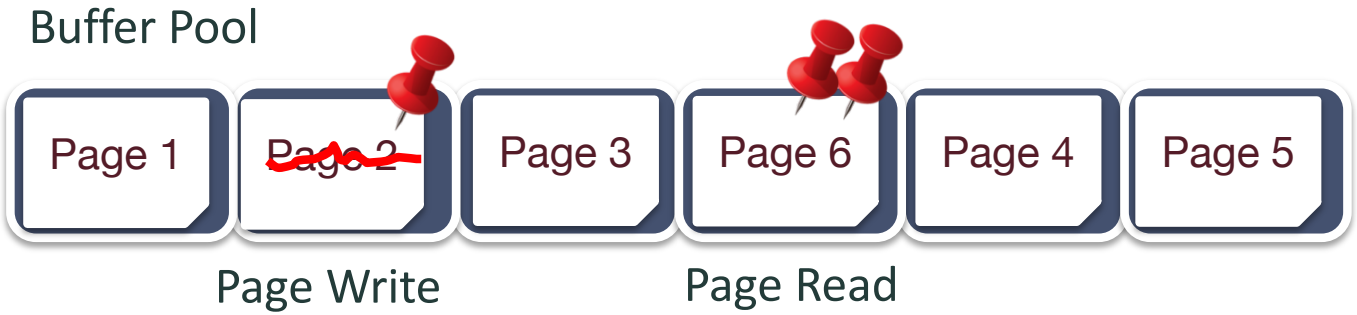
Buffer pool: Large range of memory, malloc'ed at DBMS server boot time (MBs-GBs)

Buffer Manager metadata: Smallish array in memory, malloc'ed at DBMS server boot time

FrameId	PageId	Dirty?	Pin Count
1	1	N	0
2	2	Y	1
3	3	N	0
4	6	N	2
5	4	N	0
6	5	N	0

Keep an in-memory index (hash table) on PageId

Buffer Manager



Buffer Manager metadata

FrameId	PageId	Dirty?	Pin Count
1	1	N	0
2	2	Y	1
3	3	N	0
4	6	N	2
5	4	N	0
6	5	N	0

- Page pin count keeps track of which pages are in use

When a Page is Requested …

1. If requested page is not in pool:
   a. Choose an **un-pinned** (pin count = 0) frame for replacement.
   b. If frame “dirty”, write current page to disk, mark “clean”
   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!

- What happens when pool is full?
  - Need to evict an existing page
  - We need a **page replacement policy**

Page Replacement Policies

- Two policies we will discuss:
  - Least-recently-used (LRU), Clock
  - Most-recently-used (MRU)
- Policy can have big impact on #I/Os
  - Depends on the **access pattern.**

LRU Replacement Policy

- Least Recently Used (LRU)
  - Pinned Frame: not available to replace
  - Track time each frame last unpinned (end of use)
  - Replace the frame which was least recently used

FrameId	PageId	Dirty?	Pin Count	Last Used
1	1	N	0	43
2	2	Y	1	21
3	3	N	0	22
4	6	N	2	11
5	4	N	0	24
6	5	N	0	15

Recall 61c!

Cache Replacement Policies

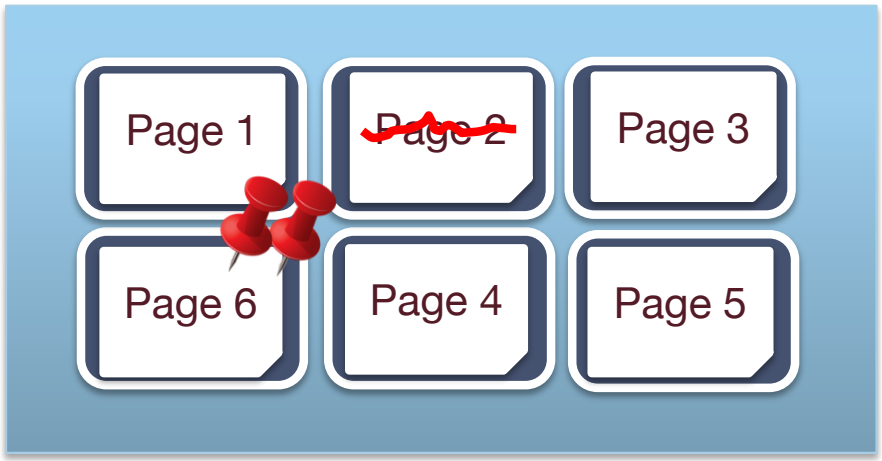
- Random Replacement
  - Hardware randomly selects a cache evict
- Least-Recently Used
  - Hardware keeps track of access history
  - Replace the entry that has not been used for the longest time
  - For 2-way set-associative cache, need one bit for LRU replacement

LRU Replacement Policy, Pt 2

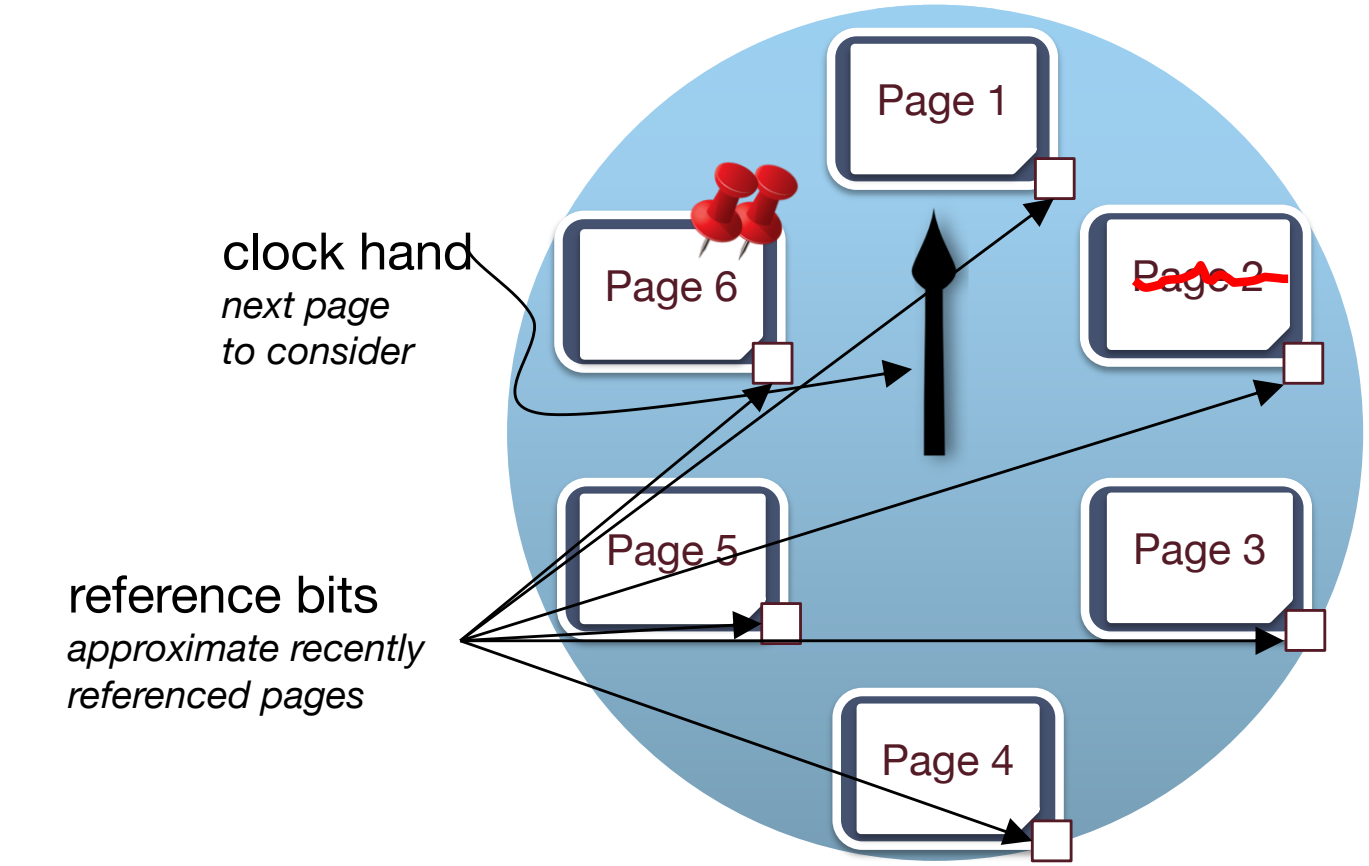
- Very common policy: intuitive and simple
  - Good for repeated accesses to popular pages (temporal locality)
  - Can be costly. Why?
    - Need to "find min" on the last used attribute (priority heap data structure)
- Approximate LRU: **CLOCK** policy

FrameId	PageId	Dirty?	Pin Count	Last Used
1	1	N	0	43
2	2	Y	1	21
3	3	N	0	22
4	6	N	2	11
5	4	N	0	24
6	5	N	0	15

Buffer Manager State

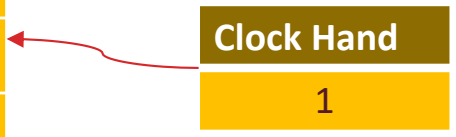


Clock Policy State: Illustrated



Clock Policy State

FrameId	PageId	Dirty?	Pin Count	Ref Bit
1	1	N	1	1
2	2	N	1	1
3	3	N	0	1
4	4	N	0	0
5	5	N	0	0
6	6	N	0	1

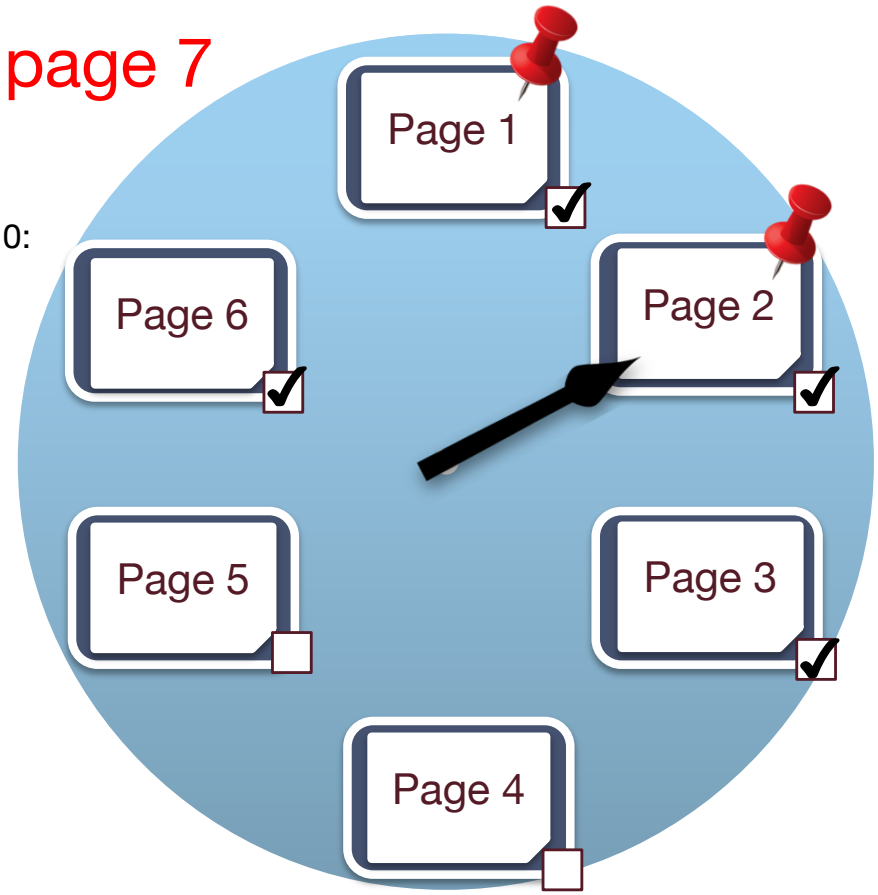


Clock Policy State: Illustrated Part 1

Request: Read page 7

Current frame has pin-count > 0:

Skip



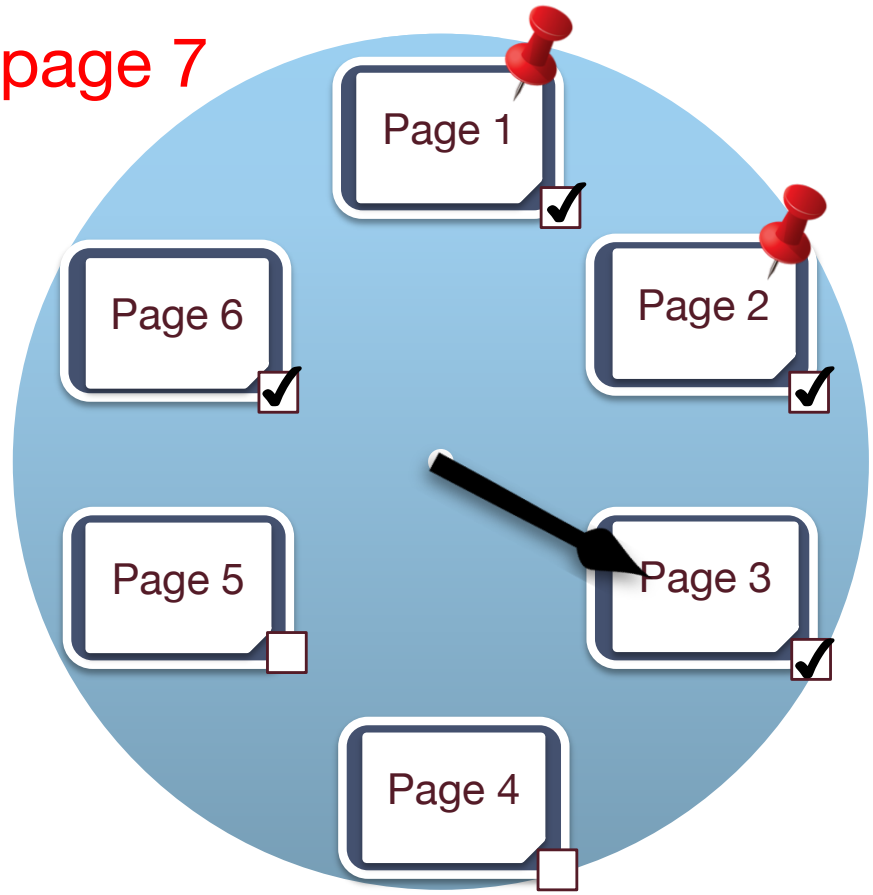
Clock Policy State: Illustrated, Part 2

Request: Read page 7

Current frame not pinned,
Ref bit set:

Clear ref bit

Skip

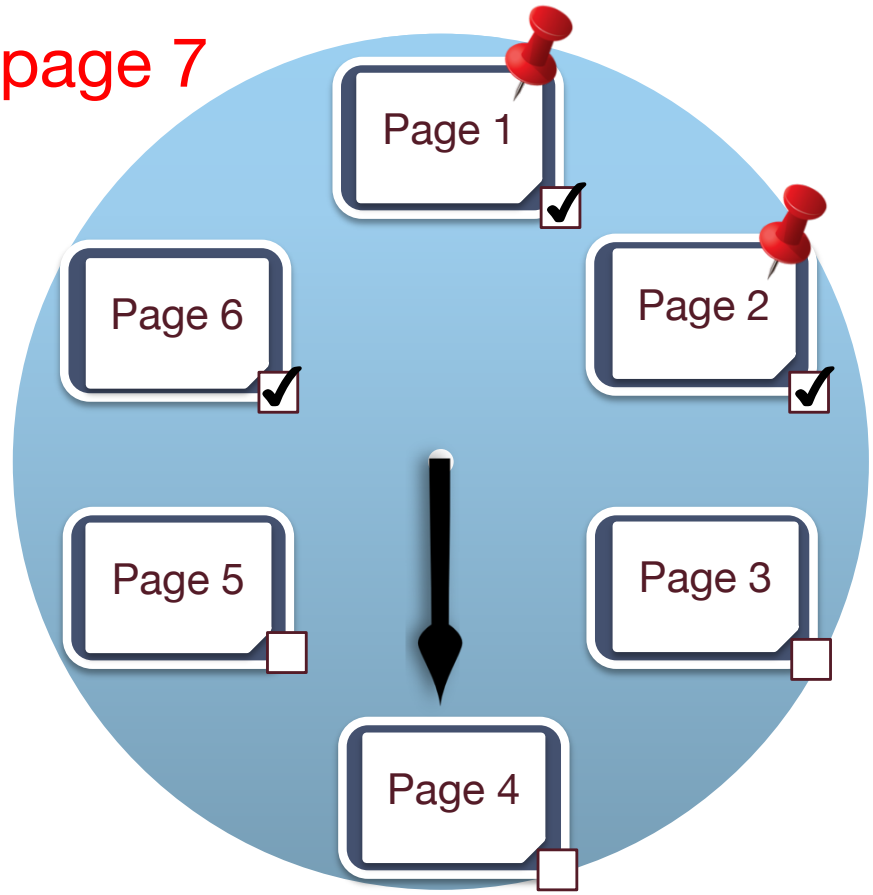


Clock Policy State: Illustrated, Pt 3

Request: Read page 7

Current frame not pinned
Ref bit unset:

Replace

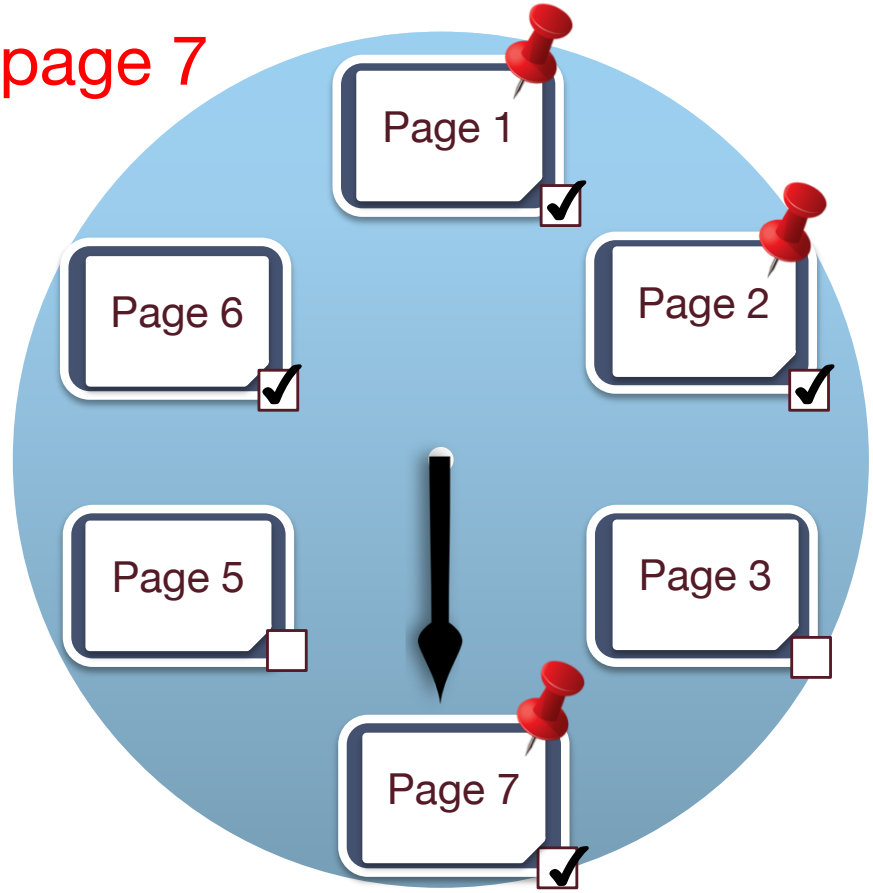


Clock Policy State: Illustrated, Pt 4

Request: Read page 7

Current frame not pinned
Ref bit unset:

- **Replace**
- **Set pinned**
- **Set ref bit**
- **Advance clock**

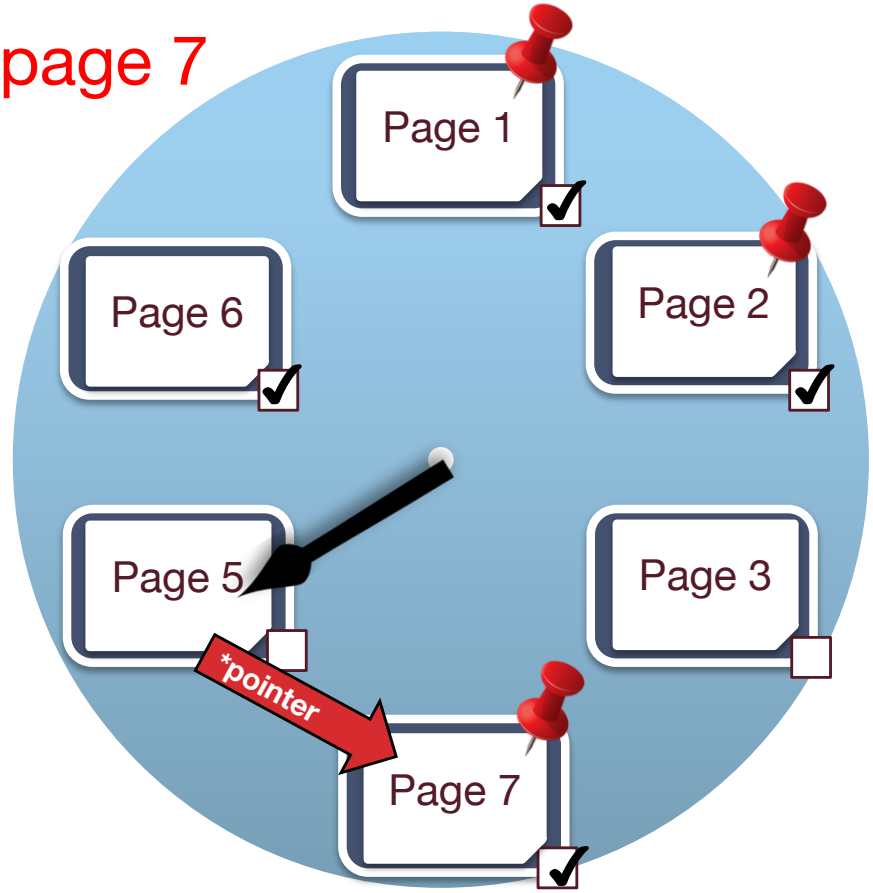


Clock Policy State: Illustrated, Pt 5

Request: Read page 7

Current frame not pinned
Ref bit unset:

- **Replace**
- **Set pinned**
- **Set ref bit**
- **Advance clock**
- **Return pointer**

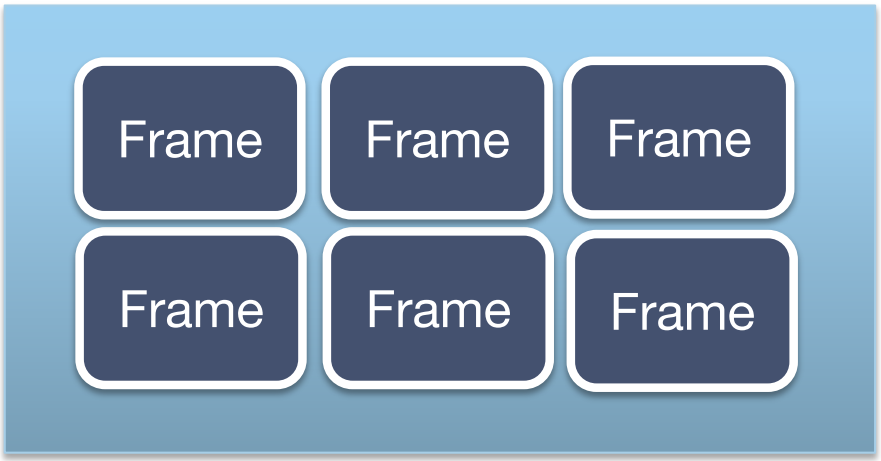


Is LRU/Clock Always Best?

- Works well for repeated accesses to popular pages
  - Temporal locality
- LRU can be costly → Clock policy is cheap
  - Clock and ref bits approximate least recently used page
  - If you like, try to find cases where they differ.
- When might they perform poorly?

Repeated Scan (LRU)

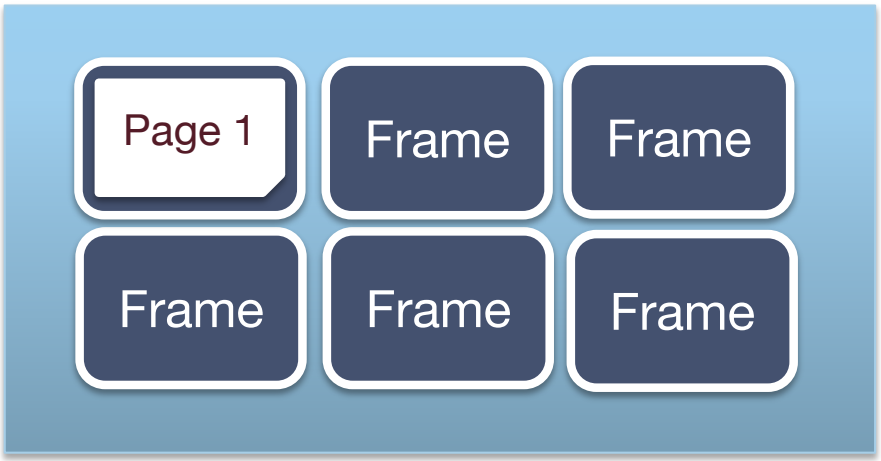
- Cache Hits: 0
- Attempts: 0



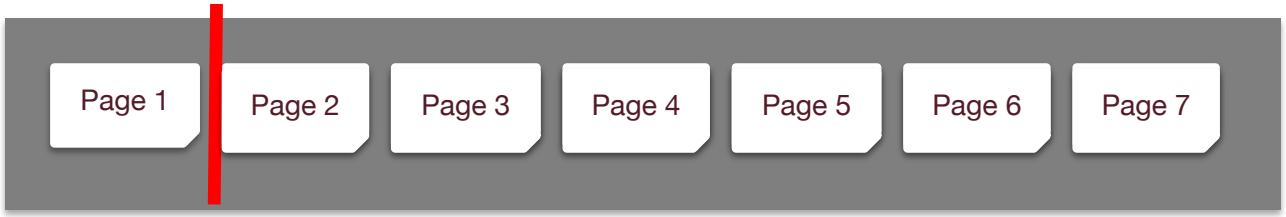
Disk Space Manager

Repeated Scan (LRU): Read Page 1

- Cache Hits: 0
- Attempts: 1

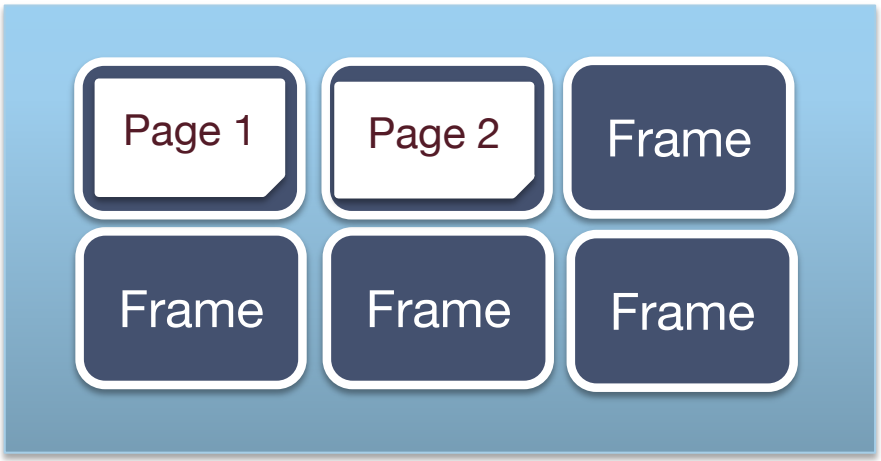


Disk Space Manager

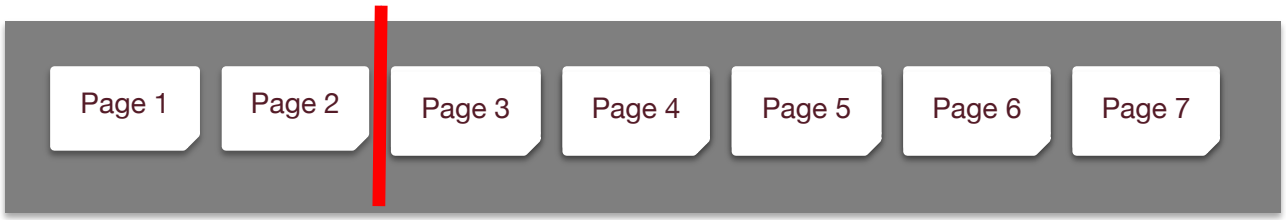


Repeated Scan (LRU): Read Page 2

- Cache Hits: 0
- Attempts: 2

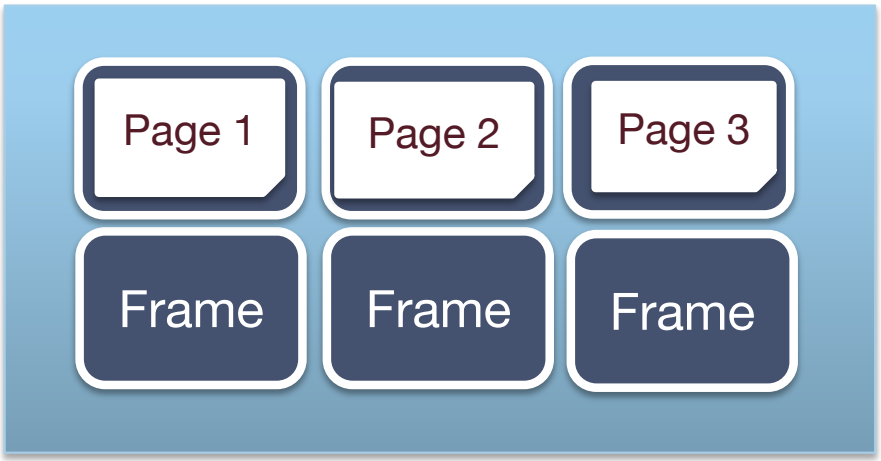


Disk Space Manager



Repeated Scan (LRU): Read Page 3

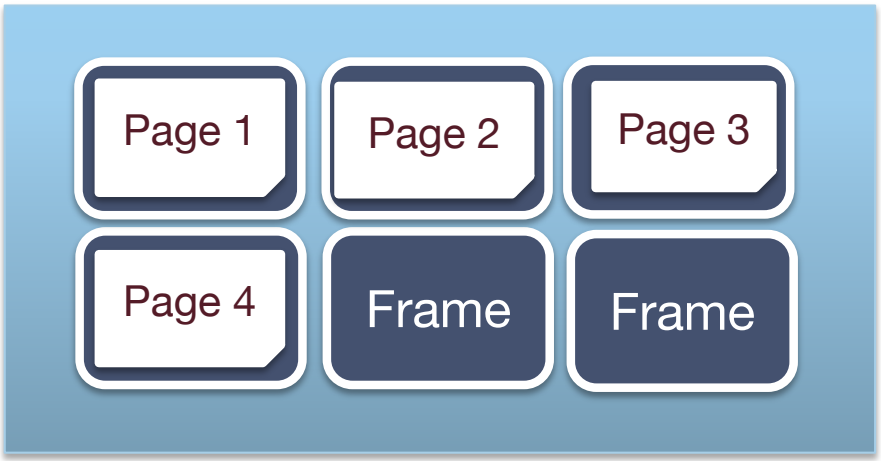
- Cache Hits: 0
- Attempts: 3



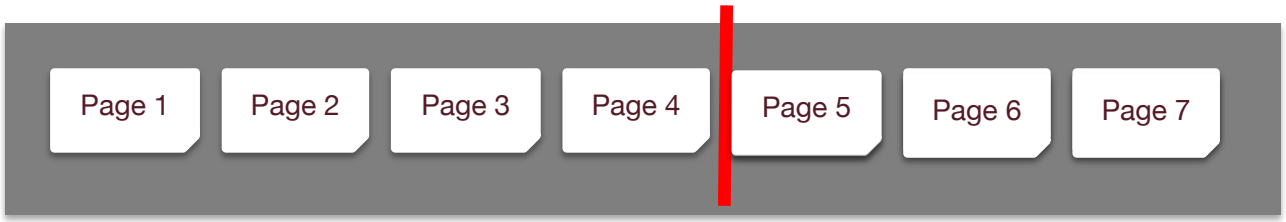
Disk Space Manager

Repeated Scan (LRU): Read Page 4

- Cache Hits: 0
- Attempts: 4

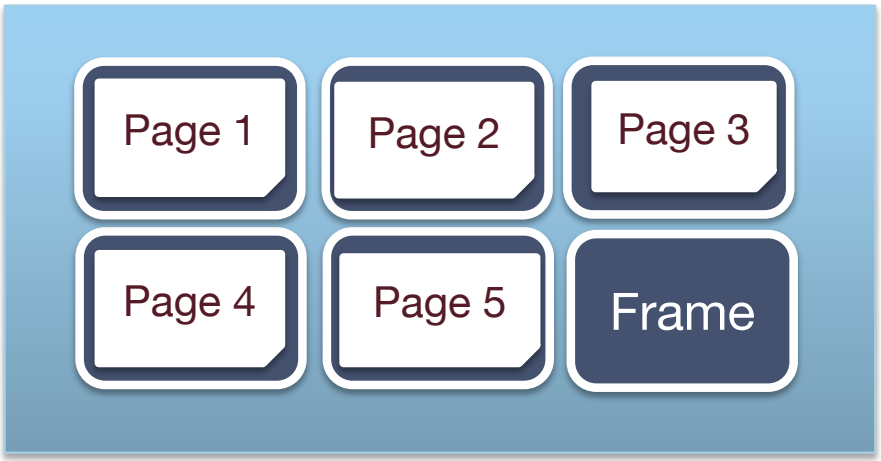


Disk Space Manager

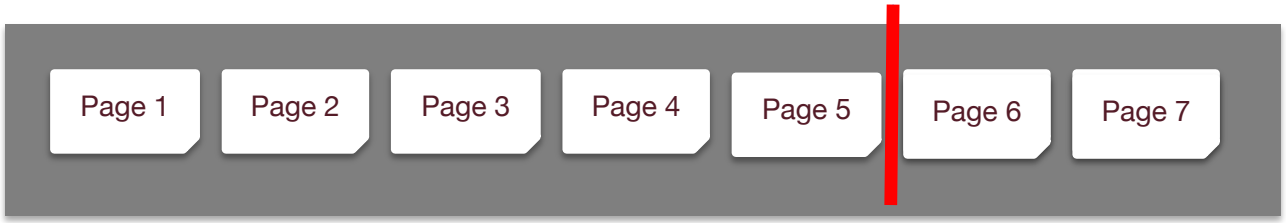


Repeated Scan (LRU): Read Page 5

- Cache Hits: 0
- Attempts: 5

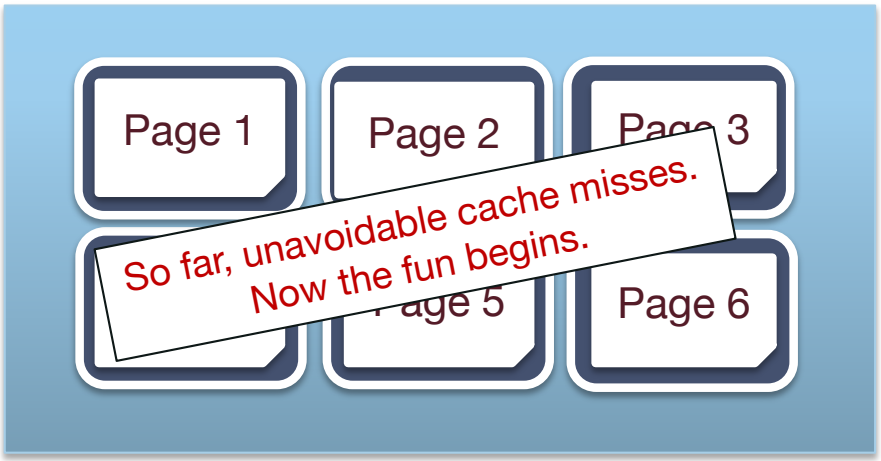


Disk Space Manager

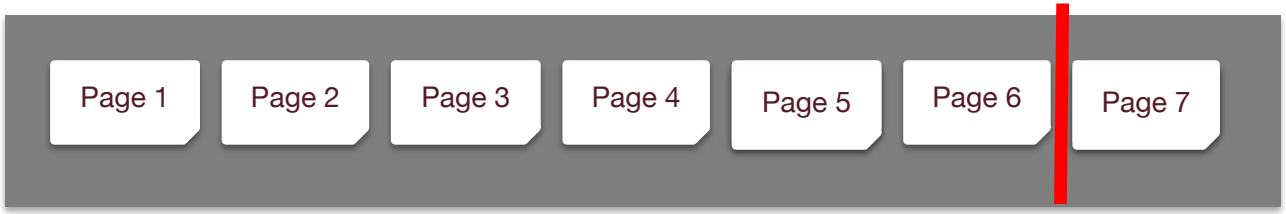


Repeated Scan (LRU): Read Page 6

- Cache Hits: 0
- Attempts: 6

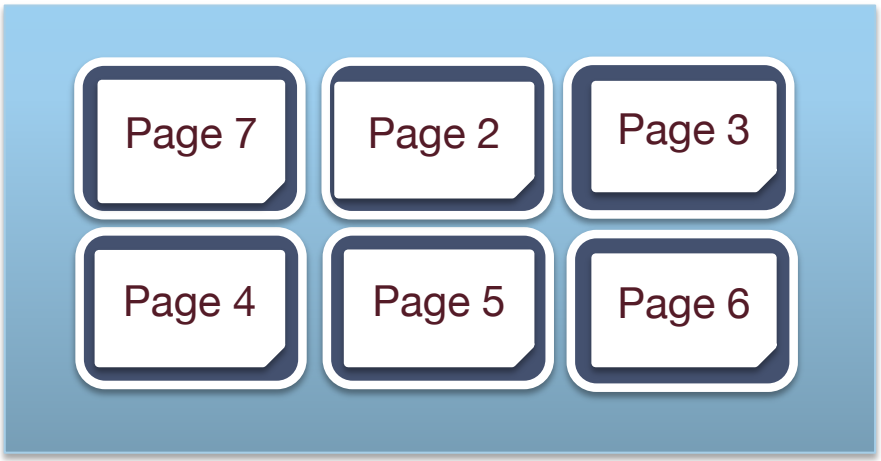


Disk Space Manager

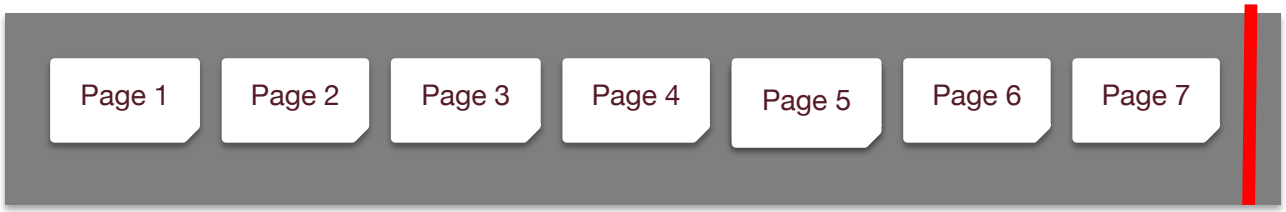


Repeated Scan (LRU): Read Page 7

- Cache Hits: 0
- Attempts: 7

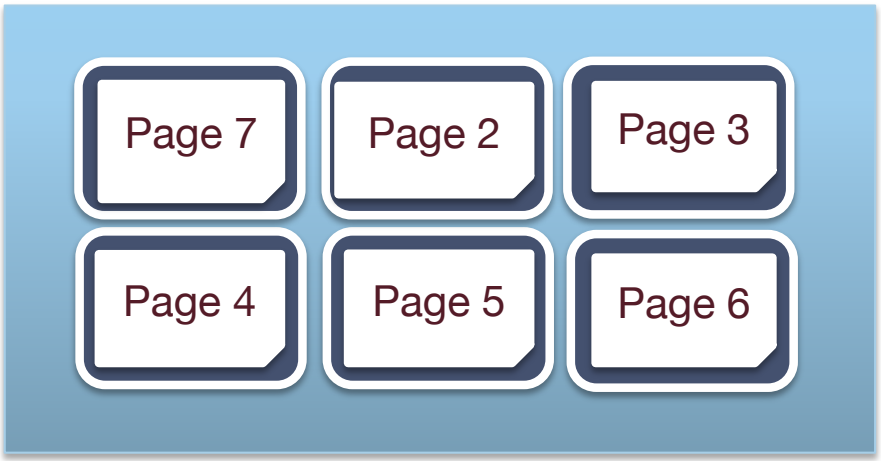


Disk Space Manager



Repeated Scan (LRU): Reset to beginning

- Cache Hits: 0
- Attempts: 7



Disk Space Manager

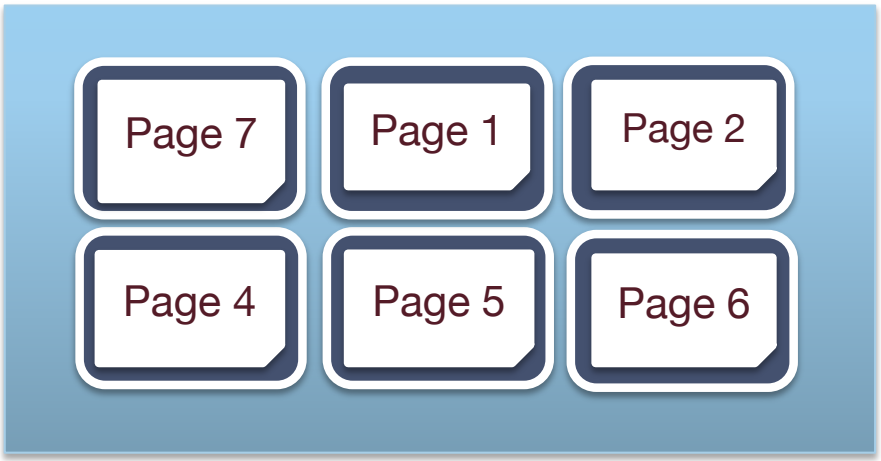
Repeated Scan (LRU): Read Page 1 (again)

- Cache Hits: 0
- Attempts: 8

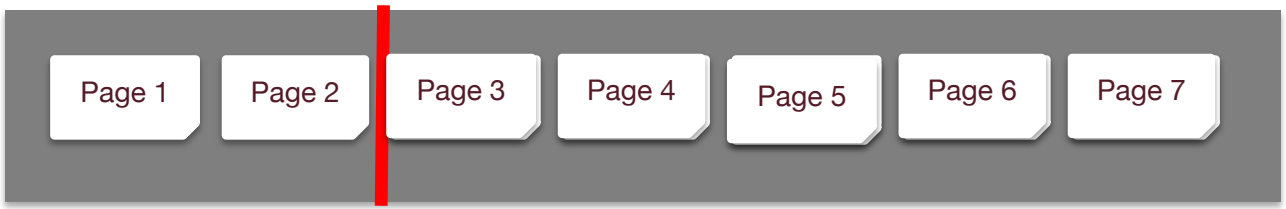
Disk Space Manager

Repeated Scan (LRU): Read Page 2 (again)

- Cache Hits: 0
- Attempts: 9

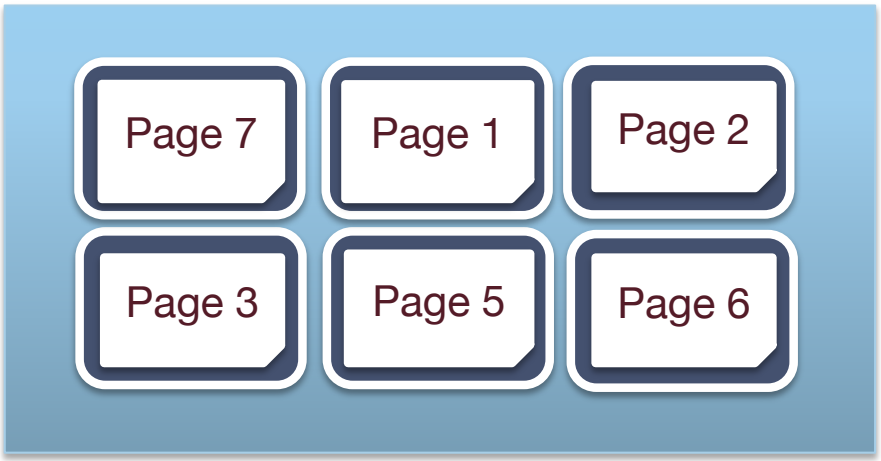


Disk Space Manager



Repeated Scan (LRU): Read Page 3 (again)

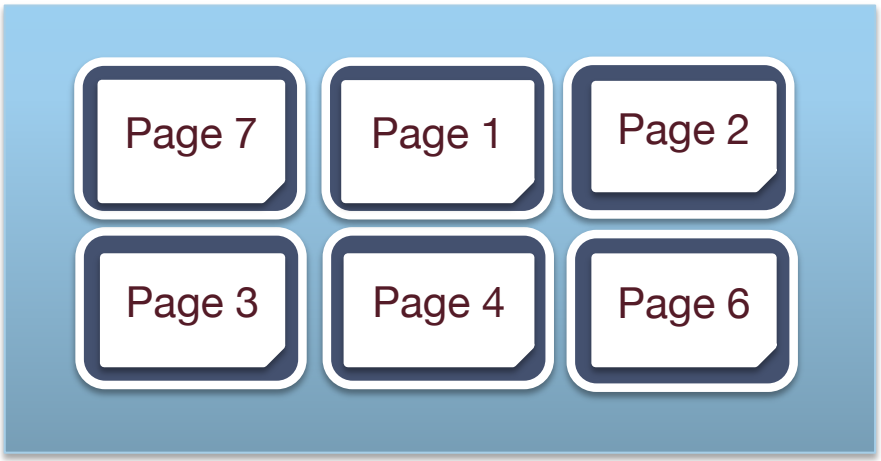
- Cache Hits: 0
- Attempts: 10



Disk Space Manager

Repeated Scan (LRU): Page 4 (again)

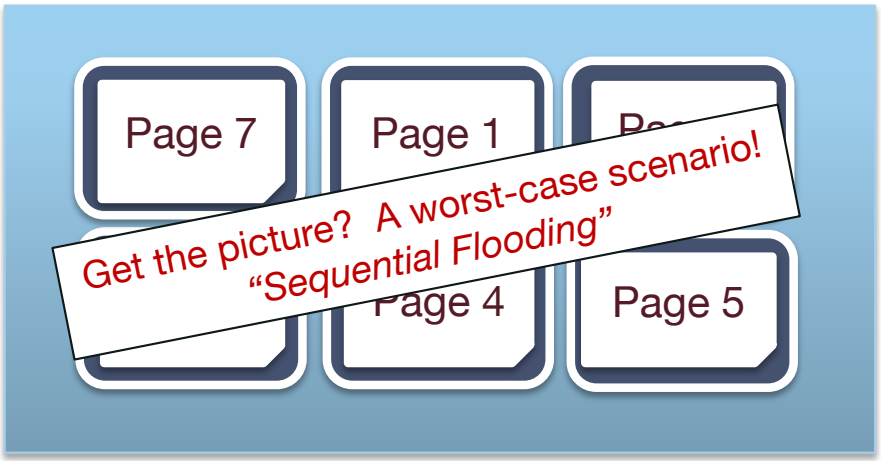
- Cache Hits: 0
- Attempts: 11



Disk Space Manager

Repeated Scan (LRU): Read Page 5, cont

- Cache Hits: 0
- Attempts: 12



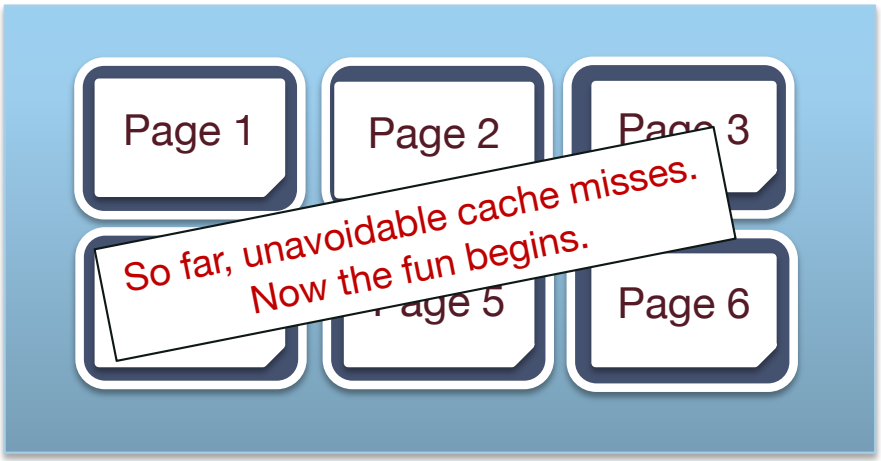
Disk Space Manager

Sequential Scan + LRU

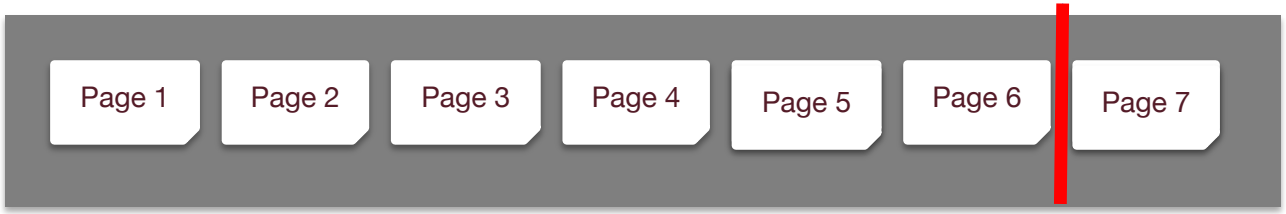
- Leads to sequential flooding
  - 0% hit rate in cache!
- Repeated sequential scan very common in database workloads
  - We will see it in nested-loops join
- How can we do better?
  - **Most Recently Used** (MRU) policy

Repeated Scan (MRU)

- Cache Hits: 0
- Attempts: 6

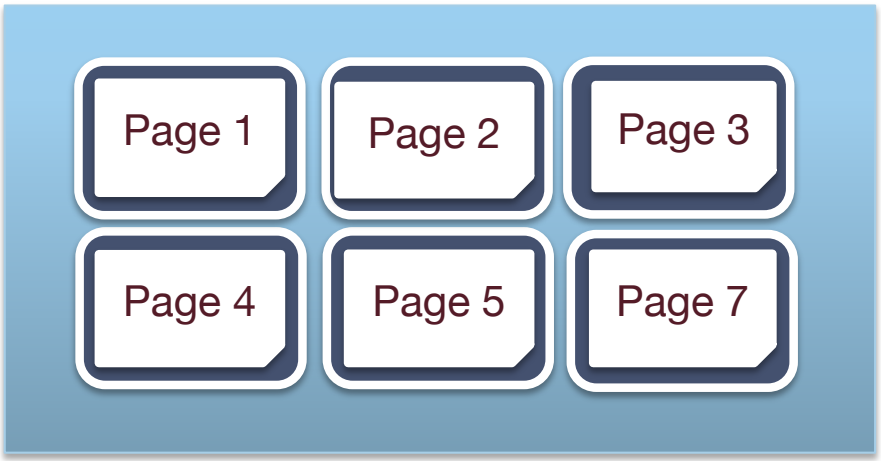


Disk Space Manager



Repeated Scan (MRU): Read Page 7

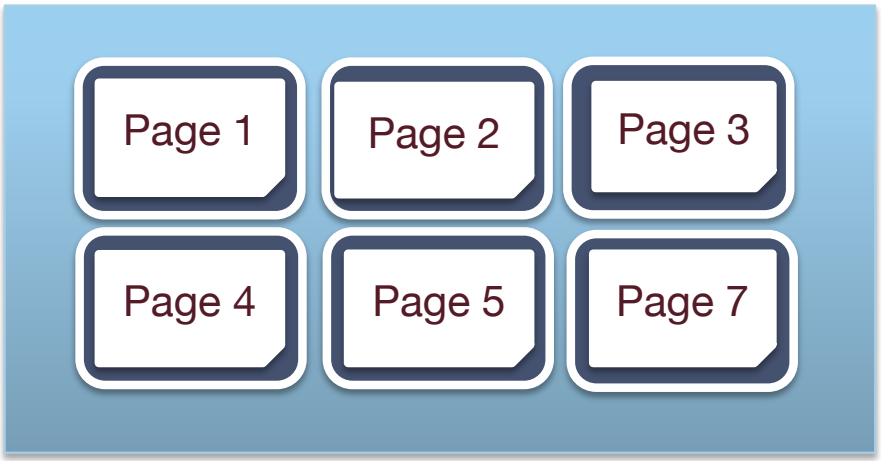
- Cache Hits: 0
- Attempts: 7



Disk Space Manager

Repeated Scan (MRU): Reset

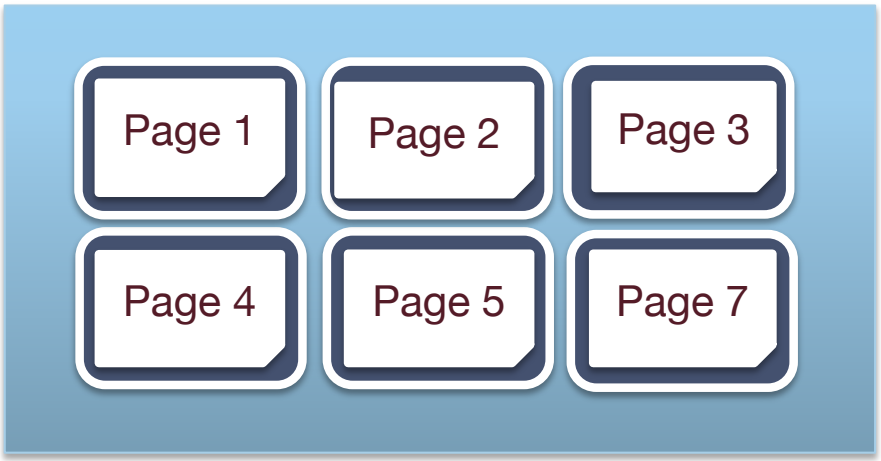
- Cache Hits: 0
- Attempts: 7



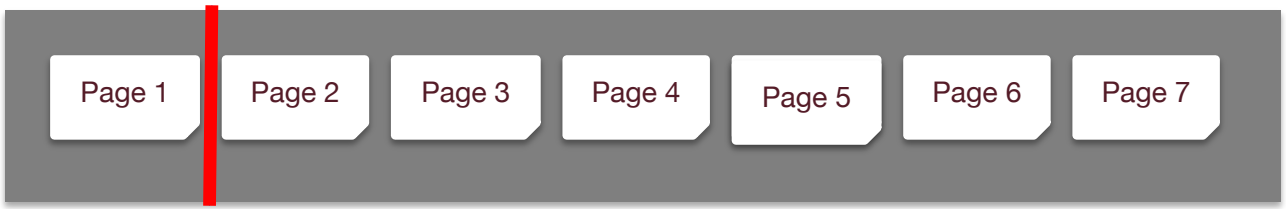
Disk Space Manager

Repeated Scan (MRU): Read Page 1 (again)

- Cache Hits: 1
- Attempts: 8

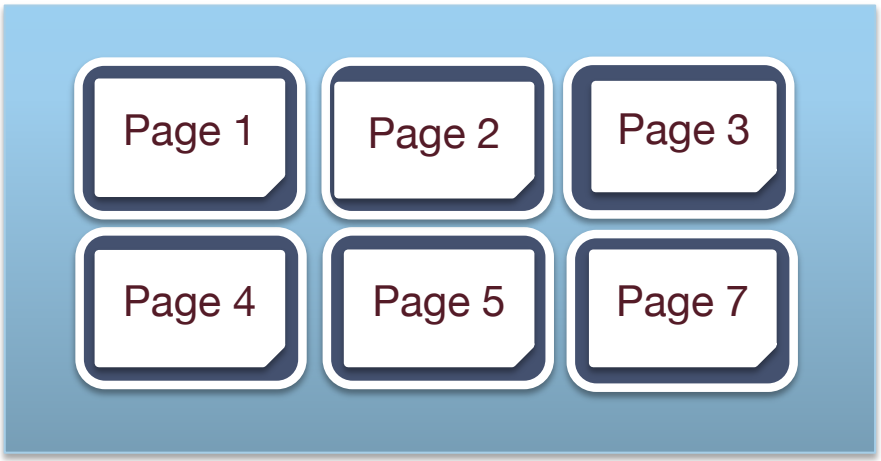


Disk Space Manager

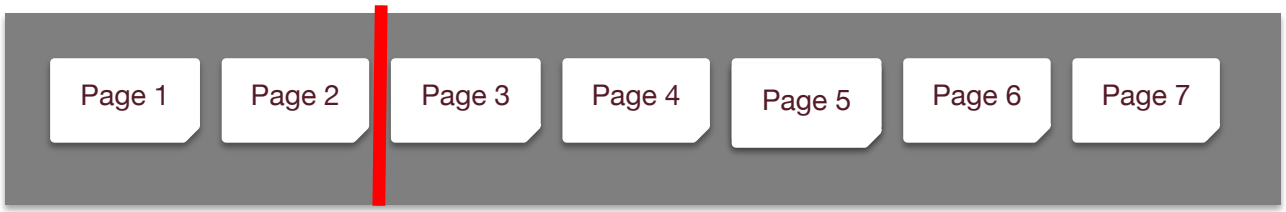


Repeated Scan (MRU): Read Page 2 (again)

- Cache Hits: 2
- Attempts: 9

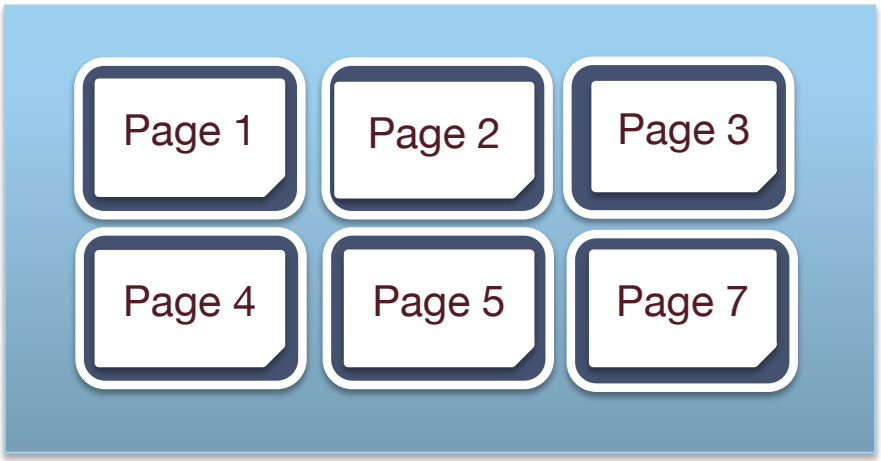


Disk Space Manager



Repeated Scan (MRU): Read Page 3 (again)

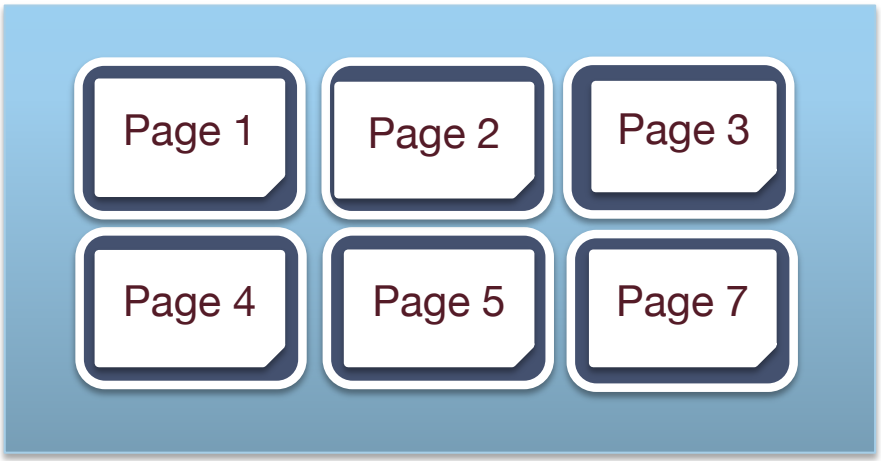
- Cache Hits: 3
- Attempts: 10



Disk Space Manager

Repeated Scan (MRU): Read Page 4 (again)

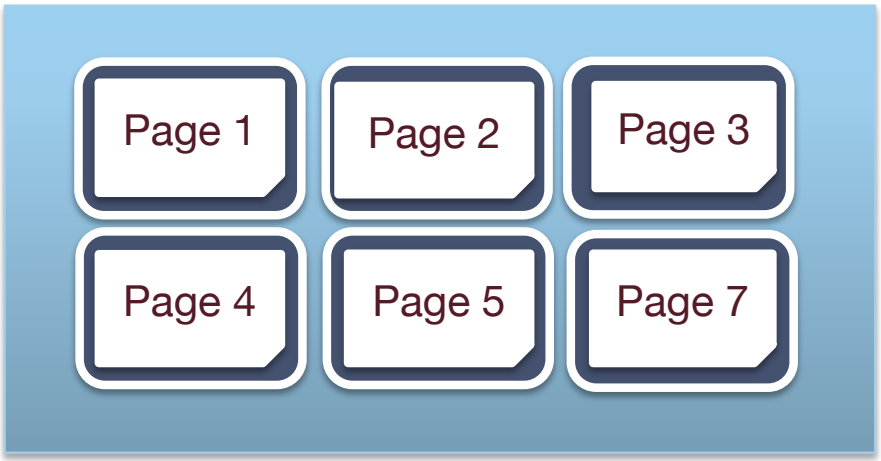
- Cache Hits: 4
- Attempts: 11



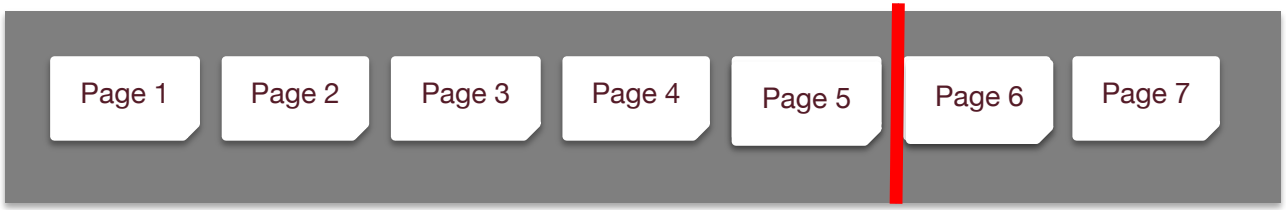
Disk Space Manager

Repeated Scan (MRU): Read Page 5 (again)

- Cache Hits: 5
- Attempts: 12

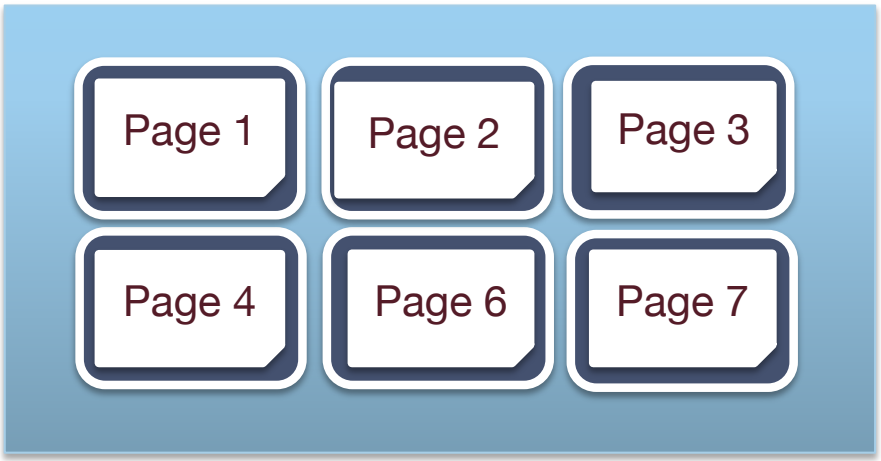


Disk Space Manager

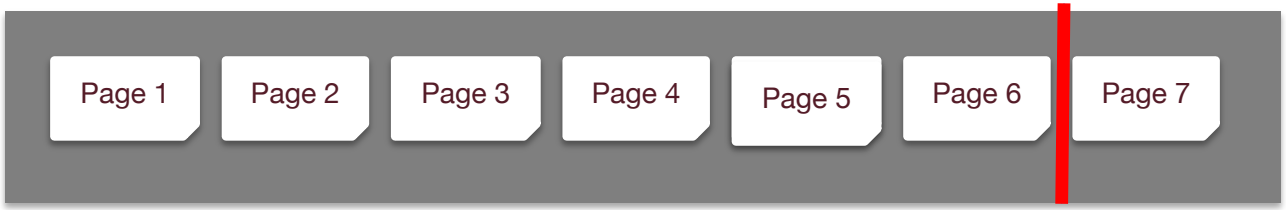


Repeated Scan (MRU): Read Page 6 (again)

- Cache Hits: 5
- Attempts: 13

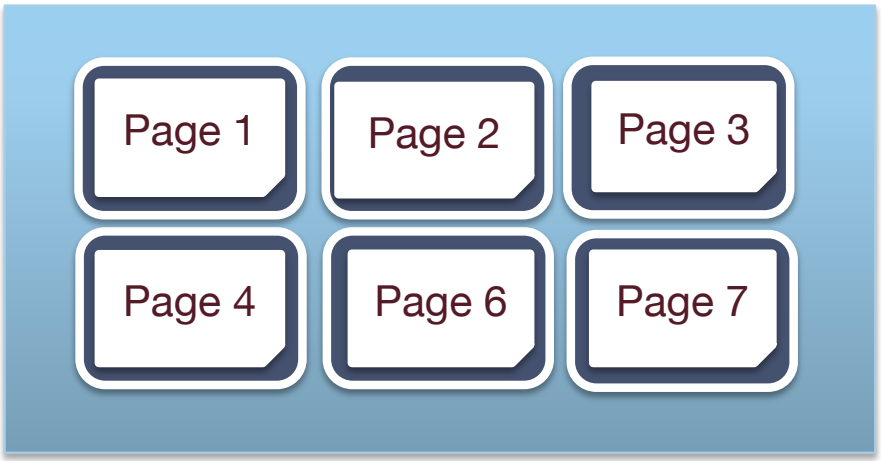


Disk Space Manager

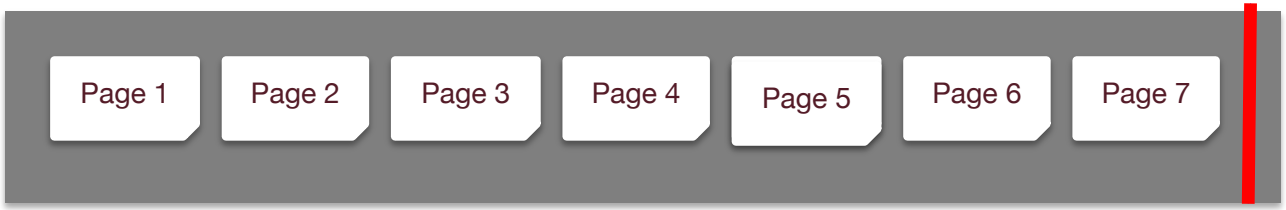


Repeated Scan (MRU): Read Page 7 (again)

- Cache Hits: 6
- Attempts: 14

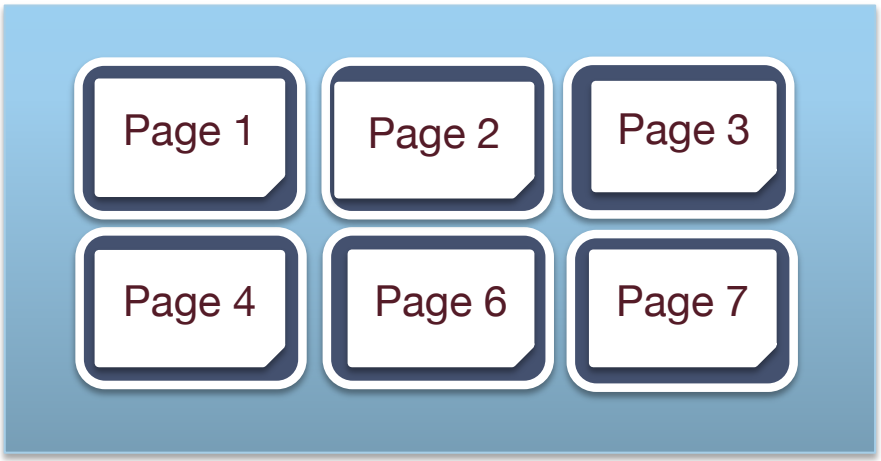


Disk Space Manager

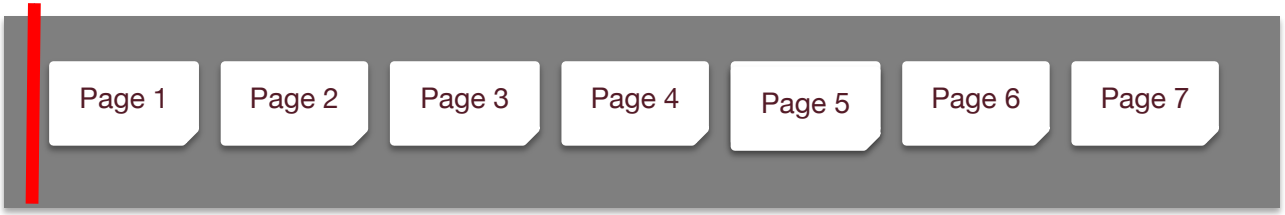


Repeated Scan (MRU): Reset (again)

- Cache Hits: 6
- Attempts: 14

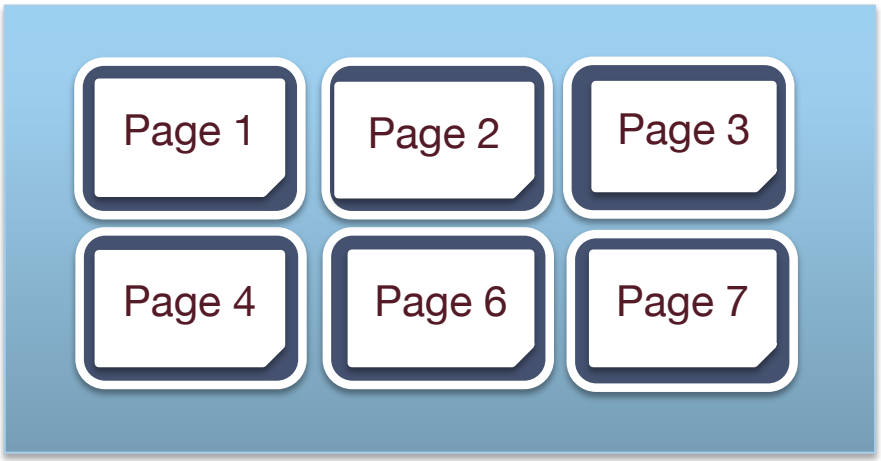


Disk Space Manager



Repeated Scan (MRU): Read Page 1 (again x2)

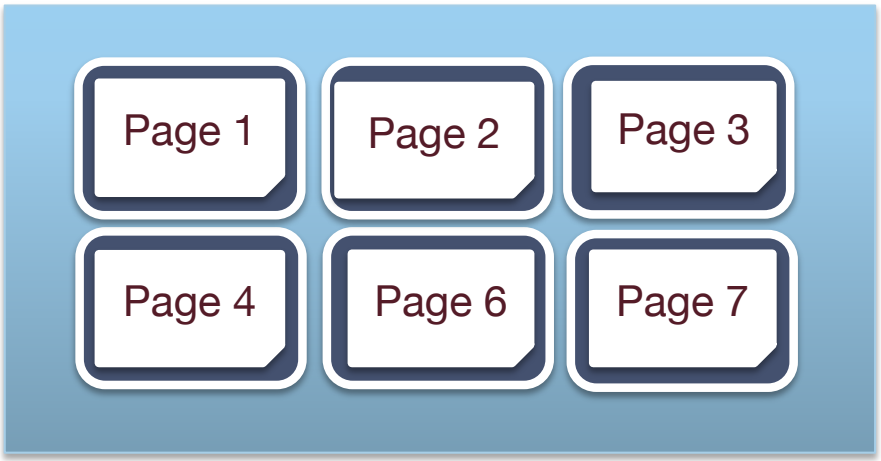
- Cache Hits: 7
- Attempts: 15



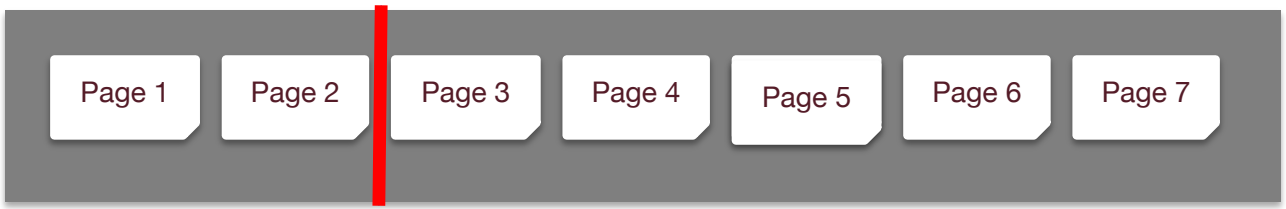
Disk Space Manager

Repeated Scan (MRU): Read Page 2 (again x2)

- Cache Hits: 8
- Attempts: 16

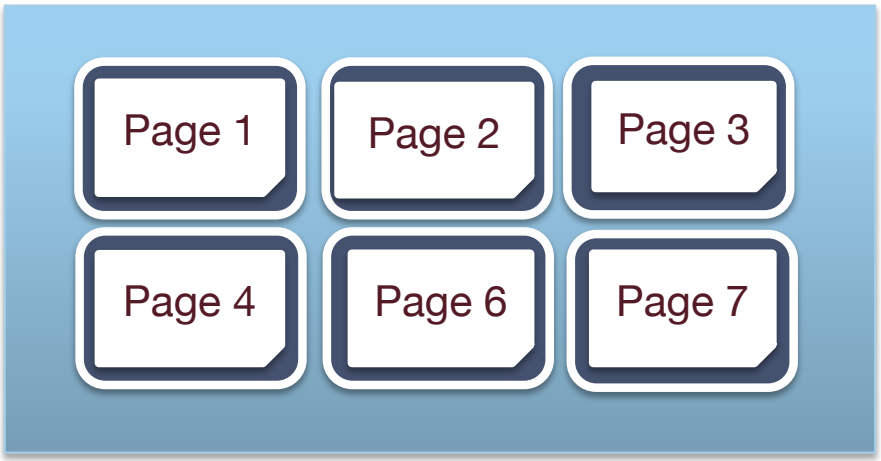


Disk Space Manager



Repeated Scan (MRU): Read Page 3 (again x2)

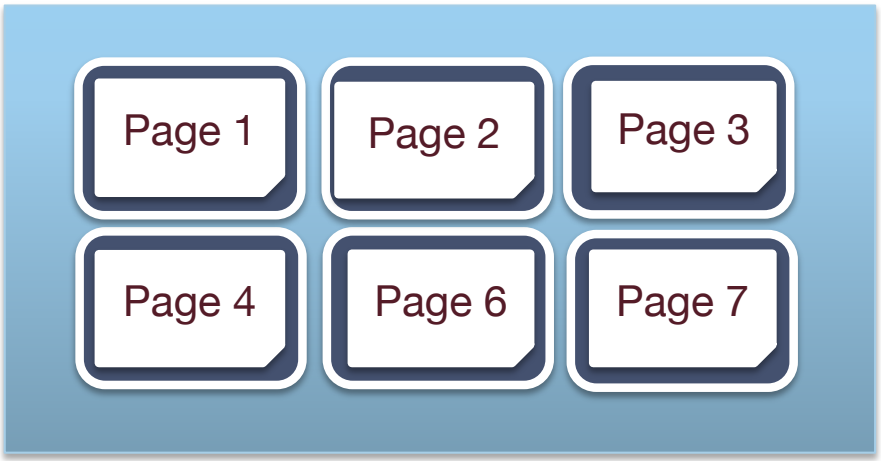
- Cache Hits: 9
- Attempts: 17



Disk Space Manager

Repeated Scan (MRU): Read Page 4 (again x2)

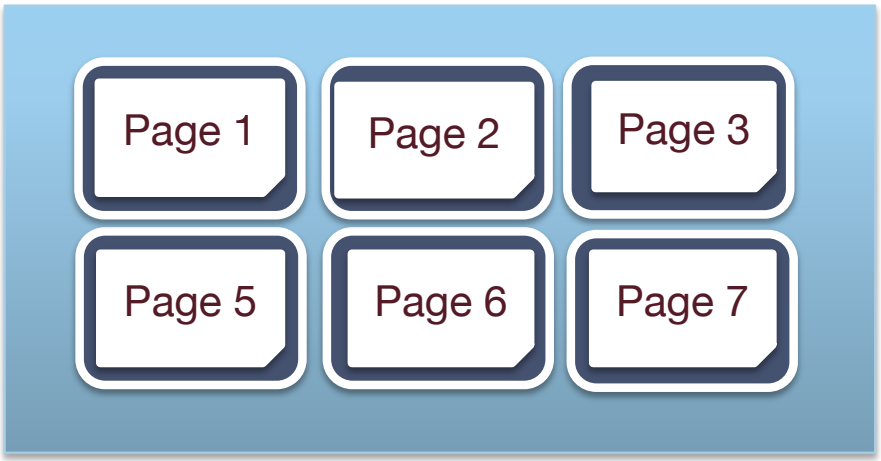
- Cache Hits: 10
- Attempts: 18



Disk Space Manager

Repeated Scan (MRU): Read Page 5 (again x2)

- Cache Hits: 10
- Attempts: 19



Disk Space Manager

General Case: SeqScan + MRU

B buffers

N > B pages in file

First pass (N attempts): 0 hits
Next N attempts: B-1 hits [for pages 1 → B-1]
Next N attempts: B-1 hits [for pages N → B-2]
Next N attempts: B-1 hits [for pages N-1 → B-3]
…

In limit we get about (B-1)/N average hit rate per attempt

Improvement for sequential scan: prefetch

- Prefetch: Ask disk space manager for a run of sequential pages
  - E.g., On request for Page 1, ask for Pages 2-5

- Why does this help?
  - Amortize random I/O overhead for magnetic disks
  - Allow computation while I/O continues in background
    - Disk and CPU are “parallel devices”

Seems like we need a hybrid!

- LRU wins for random access (hot vs. cold)
  - When might we see that behavior?
- MRU wins for repeated sequential
  - When might we see that behavior?

- Lots of fancier policies
  - Random, Not Recently Used, neural network
  - Cost of “running” the policy can be an issue

Summary

- Pin Counts and Dirty Bits:
  - When do they get set/unset?
  - By what layer of the system?
- LRU, MRU and Clock
  - Be able to run each by hand
  - For Clock:
    - What pages are eligible for replacement
    - When is reference bit set/unset
    - What is the point of the reference bit?
- Sequential flooding
  - And how it behaves for LRU (Clock), MRU