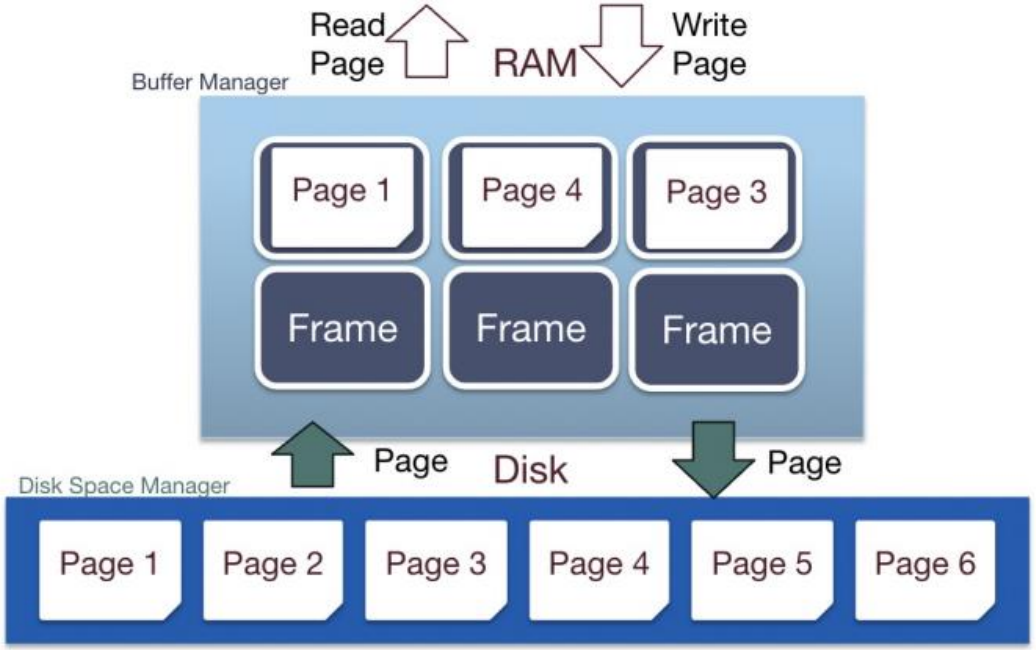
Discussion 4

Buffer Manager & B+ Tree Refinements

& Relational Algebra

Buffer Manager

- manage pages in memory
- process page requests from the file and index manager



Buffer Pool

- **Frame ID** : uniquely associated with a memory address
- **Page ID** : determine which page a frame currently contains
- **Dirty Bit** : verify whether or not a page has been modified
- **Pin Count** : track the number of requestors currently using a page

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

A buffer frame can hold the same amount of data as a page can

Handling Page Requests

- Page hit - requested page already exists within memory:

  the page’s pin count is incremented

  the page’s memory address is returned

- Page miss - requested page is not in pool:
  - If there is still space, the next empty frame is found and the page is read into that frame, then pin the page and return its address.
  - Else, Page replacement policy

Page replacement policy

- Choose an un-pinned (pin_count = 0) frame for replacement
- If frame “dirty”, write current page to disk, mark “clean”
- Read requested page into frame
- Pin the page and return its address

LRU Replacement Policy

- the Least Recently Used unpinned page which has pin count = 0
- a Last Used column is added to the metadata table and measures the latest time at which a page's pin count is decremented -- costly!

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

Clock Policy

- Iterate through frames within the table, skipping pinned pages and wrapping around to frame 0 upon reaching the end, until the first unpinned frame with ref bit = 0 is found.
- if the current frame's ref bit = 1, set the ref bit to 0 and move the clock hand to the next frame.
- if ref bit = 0, evict the existing page (and write it to disk if the dirty bit is set; then set the dirty bit to 0), read in the new page, set the frame's ref bit to 1, and move the clock hand to the next frame.

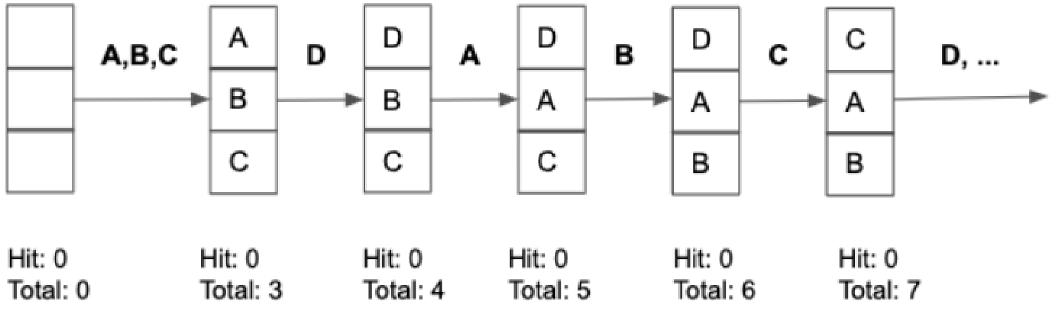
FrameId	PageId	Dirty?	Pin Count	Ref Bit
1	1	N	1	1
2	2	N	1	1
3	3	N	0	1
4	4	N	0	0
5	5	N	0	0
6	6	N	0	1

Clock Hand
1

Sequential Scanning Performance

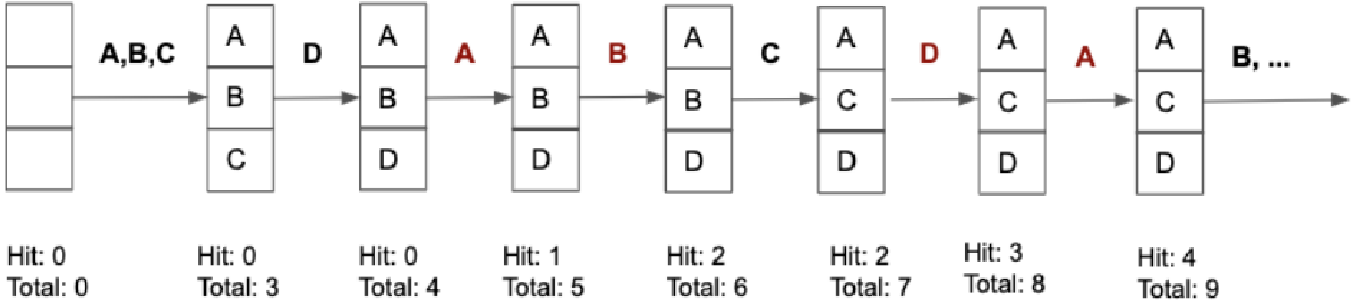
A B C D A B C D A B C D A B C D

LRU



MRU

Most Recently Used



Search Key and Ordering

In an ordered index (e.g. B+-tree) the keys are ordered lexicographically by the search key columns:

Composite Keys: more than one column

- **Lexicographic order**
- Search a range

Age > 31 & Salary = 400

SSN	Last Name	First Name	Age	Salary
123	Adams	Elmo	31	$300
443	Grouch	Oscar	32	$400
244	Oz	Bert	55	$140
134	Sanders	Ernie	55	$400
176	Grump	Donald	79	$300

Three basic alternatives for data entries in any index

- Alternative 1: By Value
  - Record contents are stored in the **index file** --No need to follow pointers
- Alternative 2: By Reference
  - By **Reference**, <**k**, rid of matching data record>
- Alternative 3: By List of references
  - By **Reference**, <**k**, rid of matching data record>
  - For very large rid lists, single data entry **spans multiple blocks**

Alternative 2
Index data entries

Key	Record Id
Gonzalez	[3, 1]
Gonzalez	[3, 2]
Gonzalez	[3, 3]
Hong	[3, 4]

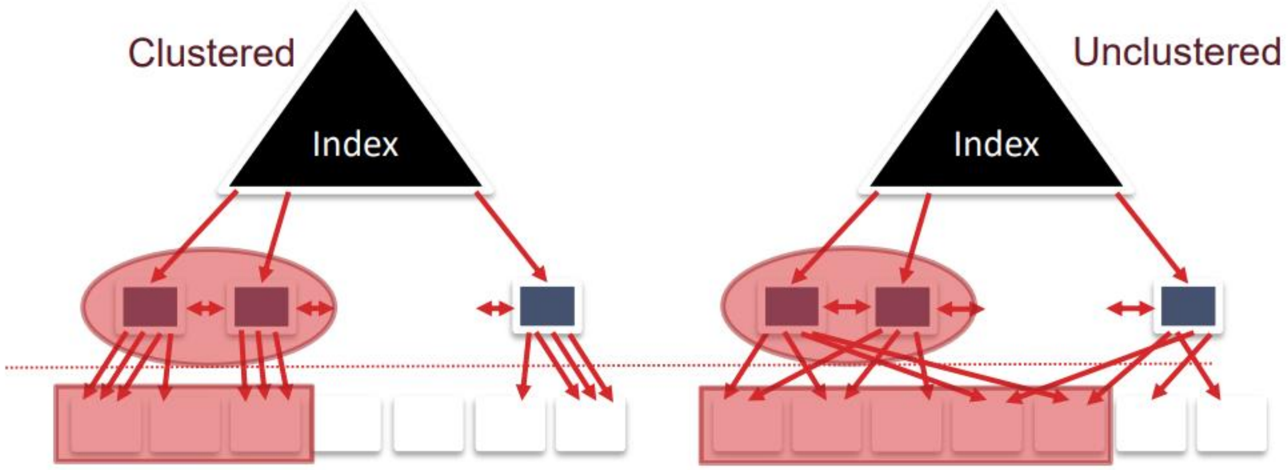
SSN	Last Name	First Name	Salary
123	Gonzalez	Amanda	$400
443	Gonzalez	Joey	$300
244	Gonzalez	Jose	$140
134	Hong	Sue	$400

Alternative 3
Index data entries

Key	Record Id
Gonzalez	[3, {1, 2, 3}]
Hong	[3,4]

Clustered vs. Unclustered Index

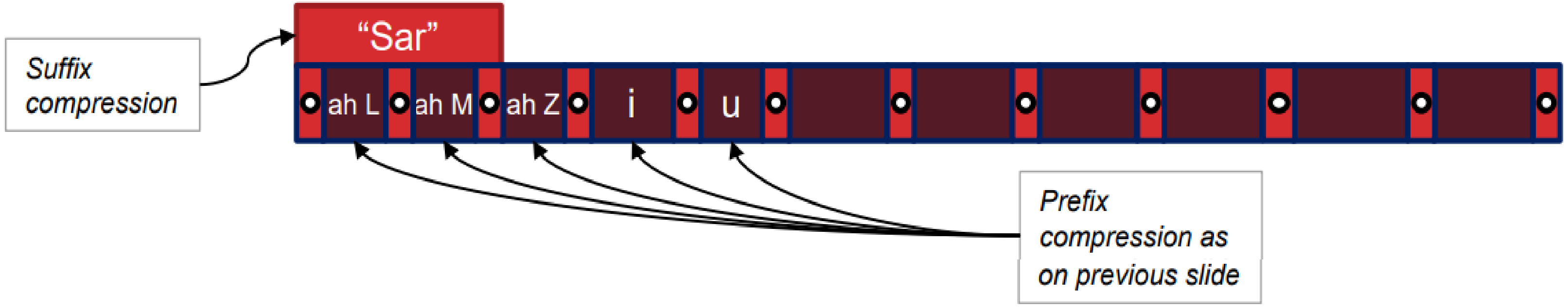
- Heap file records are kept mostly ordered according to **search keys** in index
- To build a clustered index, first sort the heap file
  - Leave some free space on each block for future inserts
- Blocks at end of file may be needed for inserts
  - Order of data records is “close to”, but not identical to, the sort order



B+ Tree Refinement: Variable-Length Keys

- Prefix Key Compression
  - determine minimum splitting prefix and copy up

- Suffix Key Compression
  - Move common prefix to header, leave only (compressed) suffix next to pointer



B+-TREE COSTS

	Heap File	Sorted File	Clustered Index
Scan all records	O(B)	O(B)	O(B)
Equality Search	O(B)	$O(\log_2 B)$	$O(\log_F B)$
Range Search	O(B)	$O(\log_2 B)$	$O(\log_F B)$
Insert	O(1)	O(B)	$O(\log_F B)$
Delete	O(B)	O(B)	$O(\log_F B)$

- B: The number of data blocks
- R: Number of records per block
- D: Average time to read/write disk block
- F: Average internal node fanout
- E: Average # data entries per leaf

Relational Algebra Operators

Unary Operators: on single relation

- Projection (π): Retains only desired columns (vertical)
- Selection (σ): Selects a subset of rows (horizontal)
- Renaming (ρ): Rename attributes and relations.

Binary Operators: on pairs of relations

- Union ($\cup$): Tuples in r1 or in r2.
- Set-difference (-): Tuples in r1, but not in r2.
- Cross-product ($\times$): Allows us to combine two relations

Compound Operators: common “macros” for the above

- Intersection ($\cap$): Tuples in r1 and in r2.
- Joins ($\bowtie_\theta$, $\bowtie$): Combine relations that satisfy predicates

Compound Operator: Join

- Joins are compound operators (like intersection):
  - Generally, R $\bowtie_\theta$ S $= \sigma_\theta$ (R$\times$S)
- Hierarchy of common kinds:
  - **Theta Join** ($\bowtie_\theta$): join on logical expression θ
    - **Equi-Join**: theta join with theta being a conjunction of equalities
      - **Natural Join** ($\bowtie$): equi-join on **all matching column names**