

DW Indexing

Need For Indexing: Speed

Consider searching your hard disk using the Windows SEARCH command.

- Search goes into directory hierarchies.
- Takes several seconds, and there are only a few thousand files.

Assume a fast processor and (even more importantly) a fast hard disk.

- Assume file size to be 5 KB.
- Assume hard disk scan rate of a million files per second.
- Resulting in scan rate of 5 GB per second.

Need For Indexing: Speed

Largest search engine indexes more than 8 billion pages

- At above scan rate 1,600 seconds required to scan ALL pages.
- This is just for one user!
- No one is going to wait for 26 minutes, not even 26 seconds.
- If the pages are 400 billion

Hence, a sequential scan is simply not feasible.

Need For Indexing: Query Complexity

- How many customers do I have in Karachi?
- How many customers in Karachi made calls during April?
- How many customers in Karachi made calls to Multan during April?
- How many customers in Karachi made calls to Multan during April using a particular calling package?

Need For Indexing: I/O Bottleneck

- Throwing hardware just speeds up the CPU intensive tasks.
- The problem is of I/O, which does not scale up easily.
- Putting the entire table in RAM is very expensive.
- **Therefore, index!**

Indexing Concept: Library Analogy

- With the library analogy, the time complexity to find a book? The average time taken
- Using a card catalog organized in many different ways i.e. author, topic, title etc and is sorted.
- A little bit of extra time to first check the catalog, but it “gives” a pointer to the shelf and the row where book is located.
- The catalog has no data about the book, just an efficient way of searching.

Indexing Concept

- Purely physical concept, nothing to do with logical model.
- Invisible to the end user (programmer), optimizer chooses it, effects only the speed, not the answer.
- We do not consider the best and worst case
 - We talk about average time

Indexing Goal

Look at as few blocks as possible to find the matching record(s)

Conventional indexing Techniques

- Dense vs. Sparse
- Multi-level (or B-Tree) vs. Hash based indexing
- Primary Index vs. Secondary Indexes

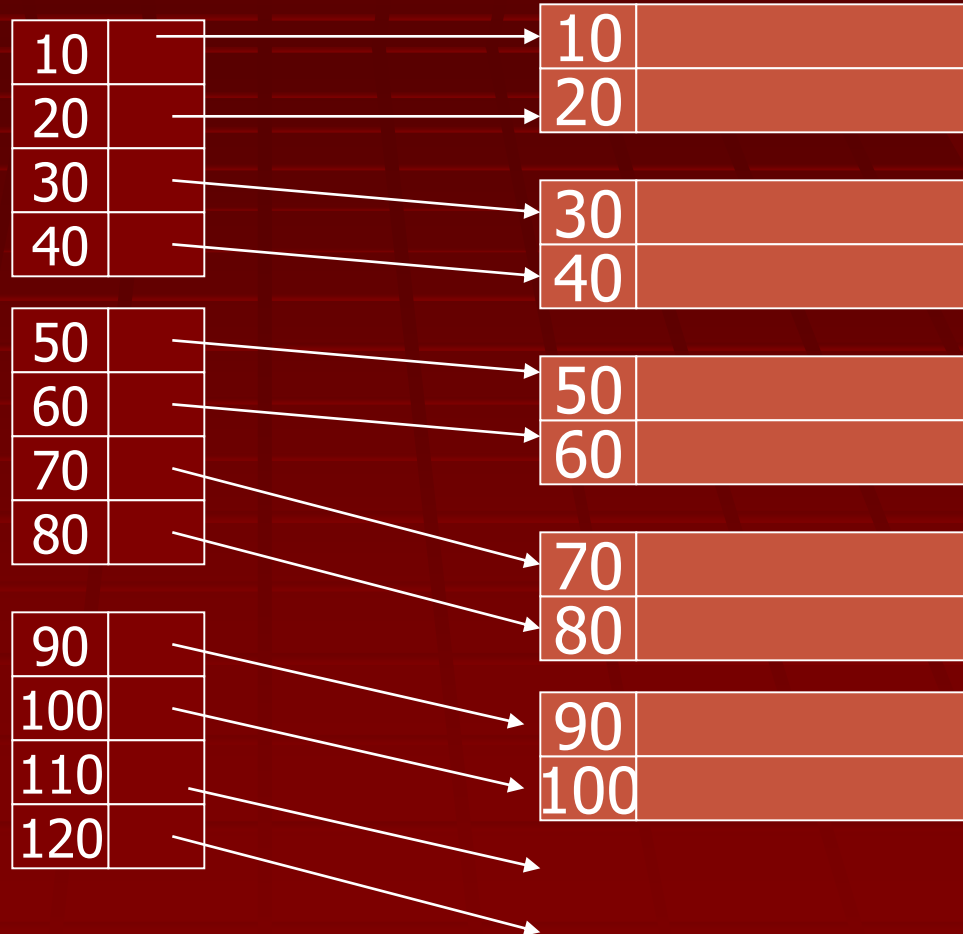
Dense vs. Sparse

Dense Index: Concept

Dense Index

File

Every key in the data file is represented in the index file



Dense Index: Adv & Dis Adv

- Advantage:
 - A dense index, if fits in the memory, is very efficient in locating a record given a key
- Disadvantage:
 - A dense index, if too big and doesn't fit into the memory, will be expensive when used to find a record given its key

Sparse Index: Concept

Sparse Index

Tuple

Normally keeps only one key per data block

Some keys in the data file will not have an entry in the index file

10	—
30	—
50	—
70	—

90	—
110	—
130	—
150	—

170	—
190	—
210	—
230	—

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

Sparse Index: Adv & Dis Adv

- Advantage:
 - A sparse index uses less space at the expense of somewhat more time to find a record given its key
 - Support multi-level indexing structure
- Disadvantage:
 - Locating a record given a key has different performance for different key values

Sparse Index: Multi level

Sparse 2nd level

File

10	
90	
170	
250	

330	
410	
490	
570	

10	
30	
50	
70	

90	
110	
130	
150	

170	
190	
210	
230	

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

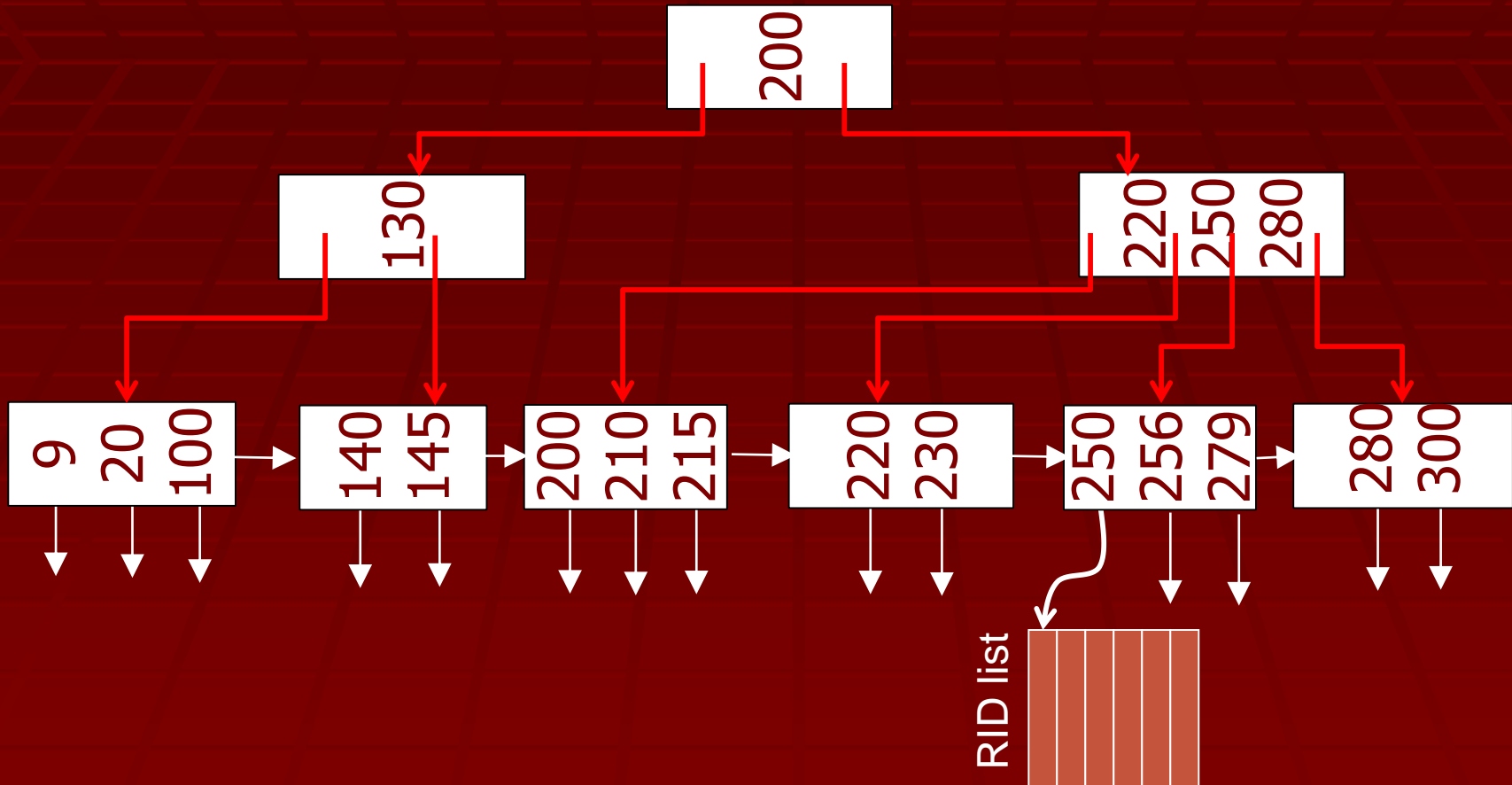
B-tree vs. Hash

B-tree Indexing: Concept

- Can be seen as a general form of multi-level indexes.
- Generalize usual (binary) search trees (BST).
- Allow efficient and fast exploration at the expense of using slightly more space.
- Popular variant: B⁺-tree
- Support more efficiently queries like:
 - `SELECT * FROM R WHERE 0 ≤ b and b < 42`

B-tree Indexing: Example

Looking for Empno 250



Each node stored in one disk block

B-tree Indexing: Limitations

- If a table is **large** and there are **fewer unique values**.
- Outcome varies with inter-character spaces.
- A noun spelled differently will result in different results.
- **Insertion can be very expensive.**

B-tree Indexing: Limitations Example

Given that MOHAMMED is the most common first name in Pakistan, a 5-million row Customers table would produce many screens of matching rows for MOHAMMED AHMAD, yet would skip potential matching values such as the following:

VALUE MISSED	REASON MISSED
Mohammed Ahmad	Case sensitive
MOHAMMED AHMED	AHMED versus AHMAD
MOHAMMED AHMAD	Extra space between names
MOHAMMED AHMAD DR	DR after AHMAD
MOHAMMAD AHMAD	Alternative spelling of MOHAMMAD

Hash based indexing

Hash Based Indexing: Concept

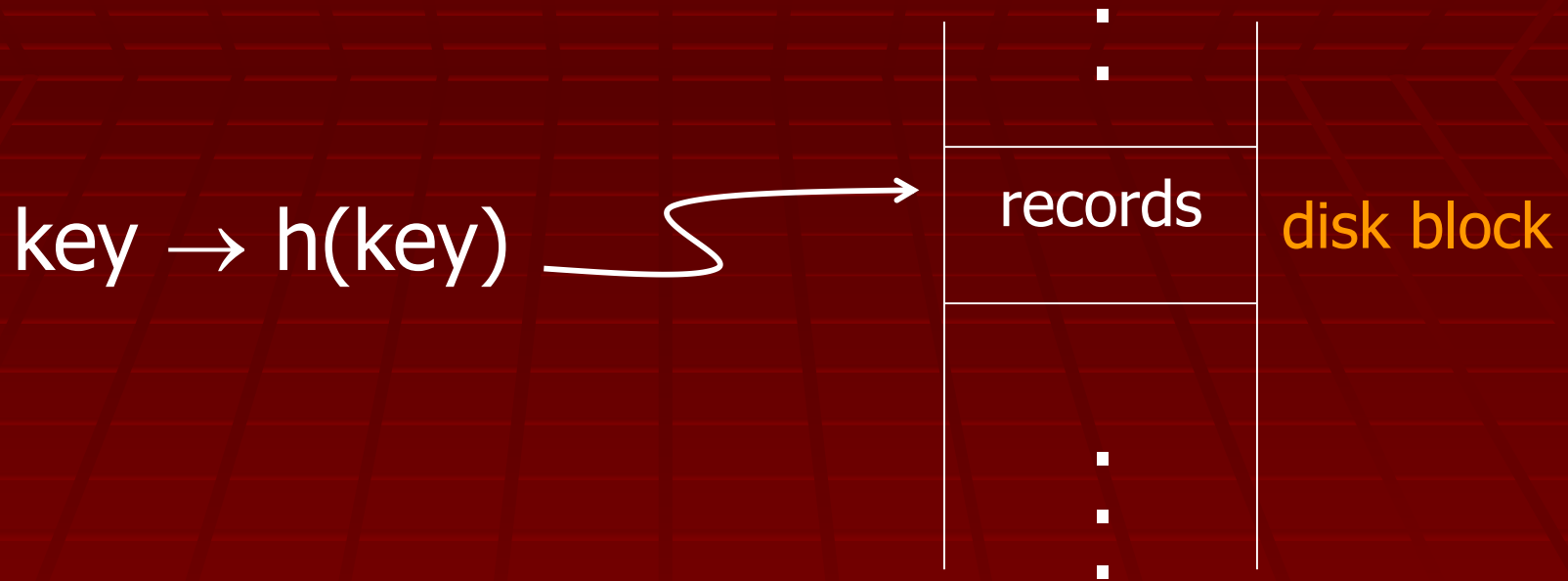
In contrast to B-tree indexing, hash based indexes do not (typically) keep index values in sorted order.

- Index entry is found by hashing on index value requiring exact match.

SELECT * FROM Customers WHERE AccttNo= 110240

- Index entries kept in hash organized tables rather than B-tree structures.
- Index entry contains ROWID values for each row corresponding to the index value.

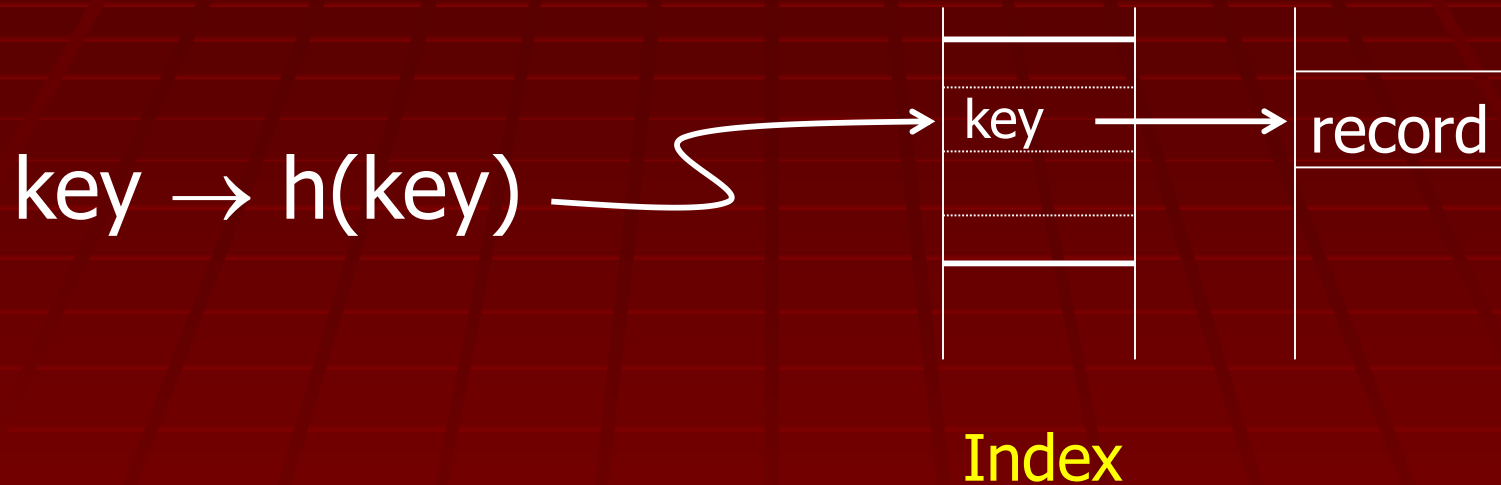
Hashing as Primary Index



Note on terminology:

The word "indexing" is often used synonymously with "B-tree indexing".

Hashing as Secondary Index



Can always be transformed to a secondary index using indirection, as above.

Indexing the Index

B-tree vs. Hash Indexes

- Indexing (using B-trees) good for range searches, e.g.:

SELECT * FROM R WHERE A > 5

- Hashing good for match based searches, e.g.:

SELECT * FROM R WHERE A = 5

Primary Key vs. Primary Index

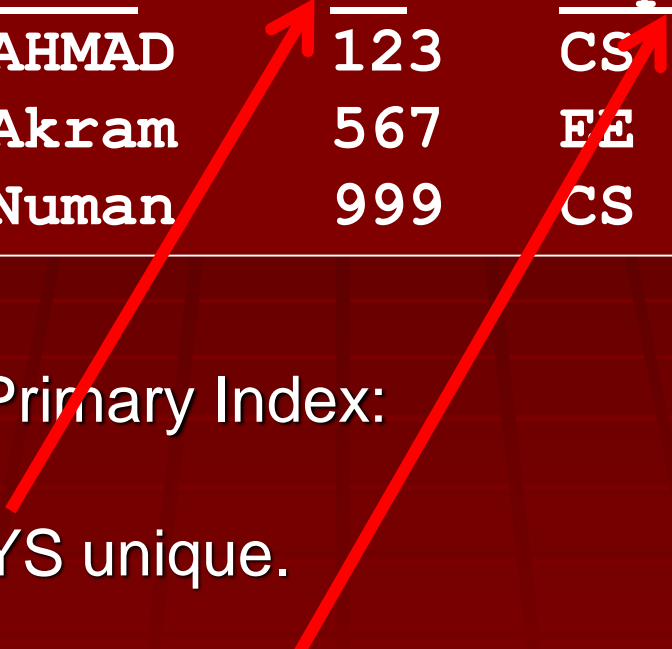
Relation Students

<u>Name</u>	<u>ID</u>	<u>dept</u>
AHMAD	123	CS
Akram	567	EE
Numan	999	CS

Primary Key vs. Primary Index

Relation Students

<u>Name</u>	<u>ID</u>	<u>dept</u>
AHMAD	123	CS
Akram	567	EE
Numan	999	CS



- Primary Key & Primary Index:
 - PK is ALWAYS unique.
 - PI can be unique, but does not have to be.
- In DSS environment, very few queries are PK based.

Primary Indexing: Criterion

- Primary index selection criteria:
 - Common join and retrieval key.
 - Can be unique UPI or non-unique NUPI.
 - Limits on NUPI.
 - Only one primary index per table (for hash-based file system).

Primary Indexing Criteria: Example

Call Table

call_id	decimal (15,0) NOT NULL
caller_no	decimal (10,0) NOT NULL
call_duration	decimal (15,2) NOT NULL
call_dt	date NOT NULL
called_no	decimal (15,0) NOT NULL

What should be the primary index of the call table for a large telecom company?

No simple answer!!

Primary Indexing

- Almost all joins and retrievals will occur through the caller_no foreign key.
 - Use caller_no as a NUPI.
- In case of non uniform distribution on caller_no or
- if phone number have very large number of outgoing calls (e.g., an institutional number could easily have several thousand calls).
 - Use call_id as UPI for good data distribution.

Primary Indexing

For a hash-based file system, primary index is free!

- No storage cost.
- No index build required.

OLTP databases use a page-based file system and therefore do not deliver this performance advantage.

Special Index Structures

- Inverted index
- Bit map index
- Cluster index

Inverted index: Concept

Inverted Index: Example-1

D1: M. Aslam BS Computer Science Lahore Campus

D2: Sana Aslam of Lahore MS Computer Engineering with GPA 3.4 Karachi Campus

Inverted index for the documents D1 and D2 is as follows:

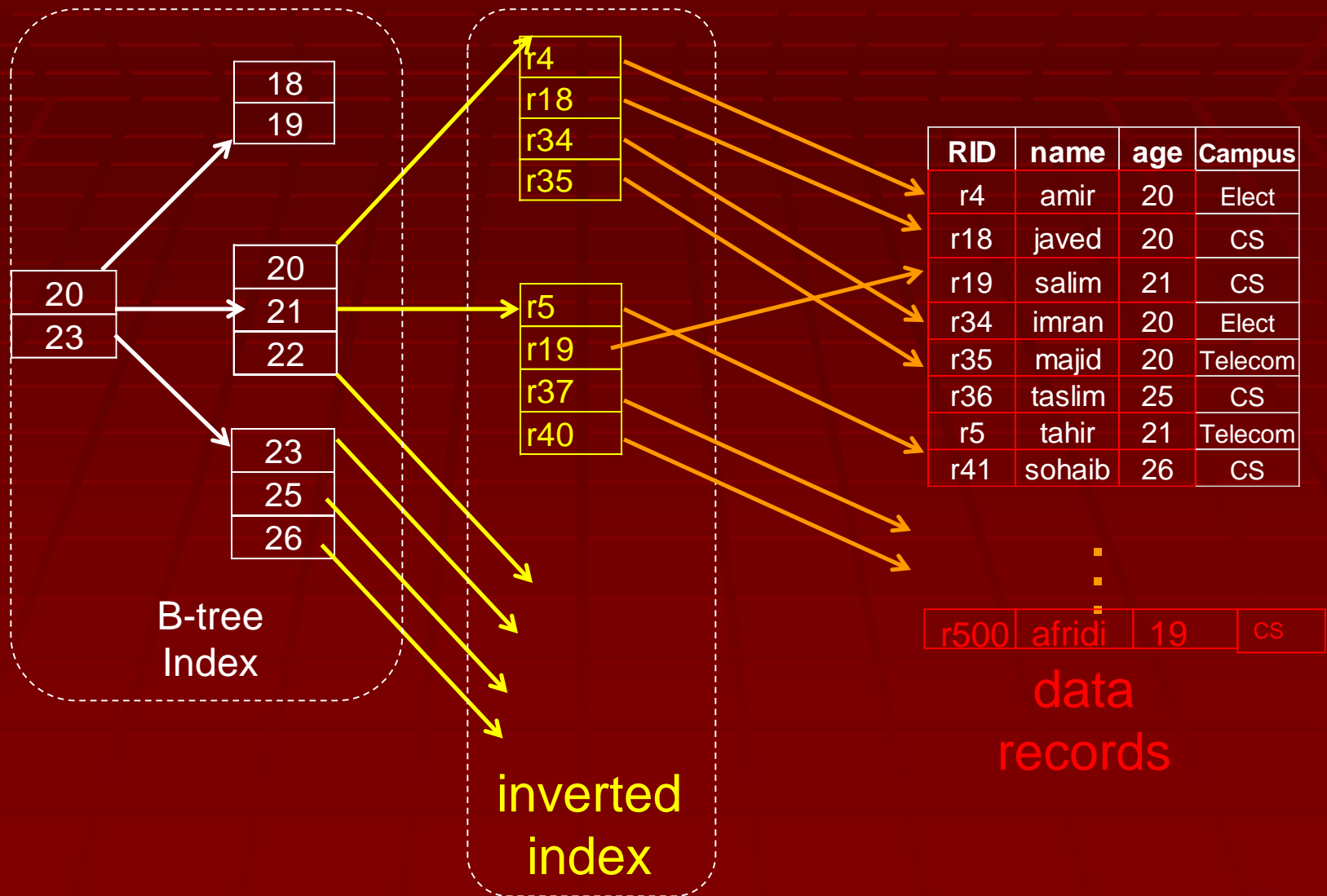
3.4	→ [D2]	Karachi	→ [D2]
Asalm	→ [D1, D2]	Lahore	→ [D1, D2]
BS	→ [D1]	M.	→ [D1]
Campus	→ [D1, D2]	MS	→ [D2]
Computer	→ [D1, D2]	of	→ [D2]
Engineering	→ [D2]	Sana	→ [D2]
GPA	→ [D2]	Science	→ [D1]
		with	→ [D2]

Subset table

Student	Name	Age	Campus	Tech
s1	amir	20	Lahore	Elect
s2	javed	20	Islamabad	CS
s3	salim	21	Lahore	CS
s4	imran	20	Peshawar	Elect
s5	majid	20	Karachi	Telecom
s6	taslim	25	Karachi	CS
s7	tahir	21	Peshawar	Telecom
s8	sohaib	26	Peshawar	CS
s9	afridi	19	Lahore	CS

...

Inverted Index: Example-2



Inverted Index: Query

- Query:
 - Get students with age = 20 and tech = “telecom”
- List for age = 20: r4, r18, r34, r35
- List for tech = “telecom”: r5, r35
- Answer is intersection: r35

Bitmap Indexes: Concept

Bitmap Indexes: Example

- The index consists of bitmaps, with a column for each unique value:

Index on City (larger table):

SID	Islamabad	Lahore	Karachi	Peshawar
1	0	1	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	0	1
5	0	0	1	0
6	0	0	1	0
7	0	0	0	1
8	0	0	0	1
9	0	1	0	0

Index on Tech (smaller table):

SID	CS	Elect	Telecom
1	1	0	0
2	0	1	0
3	0	1	0
4	1	0	0
5	0	0	1
6	0	1	0
7	0	0	1
8	1	0	0
9	1	0	0

Bitmap Index: Query

- Query:
 - Get students with age = 20 and campus = "Lahore"
- List for age = 20: 1101100000
- List for campus = "Lahore": 1010000001
- Answer is AND : 1000000000
- Good if domain cardinality is small
- Bit vectors can be compressed
 - Run length encoding

Bitmap Index: Compression

Basic Concept

Case-1

11110000111100000011110000001111 INPUT

14#04#14#06#15#06#15 OUTPUT

Case-2

10 INPUT

11#01#11#01#11#01#11#01#... OUTPUT

Case-3

1111111111111111111100000000000000000000 INPUT

117#017 OUTPUT

Bitmap Index: More Queries

- “Which students from Lahore are enrolled in ‘CS’?”
- “How many students are enrolled in ‘CS’?”

Cluster Index: Concept

Cluster Index: Example

Student	Name	Age	Campus	Tech
s9	afridi	19	Lahore	CS
s1	amir	20	Lahore	Elect
s2	javed	20	Islamabad	CS
s4	imran	20	Peshawar	Elect
s5	majid	20	Karachi	Telecom
s3	salim	21	Lahore	CS
s7	tahir	21	Peshawar	Telecom
s6	taslim	25	Karachi	CS
s8	sohaib	26	Peshawar	CS

Cluster indexing on AGE

One indexing column at a time

Student	Name	Age	Campus	Tech
s9	afridi	19	Lahore	CS
s2	javed	20	Islamabad	CS
s3	salim	21	Lahore	CS
s6	taslim	25	Karachi	CS
s8	sohaib	26	Peshawar	CS
s1	amir	20	Lahore	Elect
s4	imran	20	Peshawar	Elect
s5	majid	20	Karachi	Telecom
s7	tahir	21	Peshawar	Telecom

Cluster indexing on TECH

