Database Management Systems

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Sai University

Lecture 15, 18 October 2023

Indexing

- Why build an index?

Details of student with ID = "ABCDEF"

Quickly access relevant rows in table

Indexing

- Why build an index?
- Search key
  - As opposed to superkey, candidate key, ...
  - May need multiple search keys for a table

Indexing

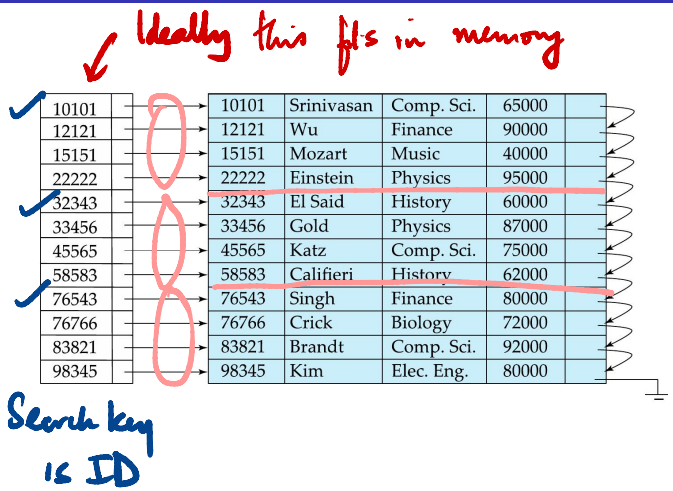
- Why build an index?
- Search key
  - As opposed to superkey, candidate key, ...
  - May need multiple search keys for a table
- Types of queries — point vs range
  - `ID = "10102"`
  - `salary > 75000`

Indexing

- Why build an index?
- Search key
  - As opposed to superkey, candidate key, ...
  - May need multiple search keys for a table
- Types of queries — point vs range
  - `ID = "10102"`
  - `salary > 75000`
- Maintaining an index
  - Inserts, deletes
  - Space

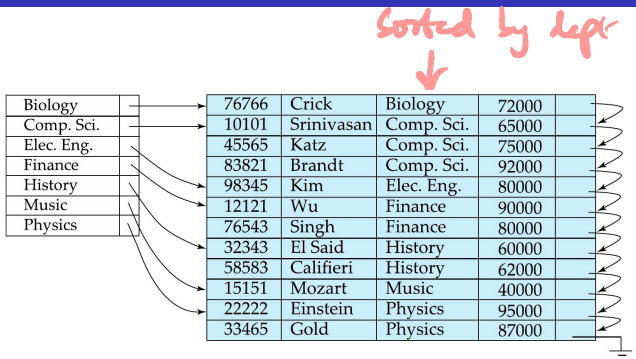
Clustering index

- File is ordered with respect to index values
- Index sequential file
- Dense index — every value is present in the index



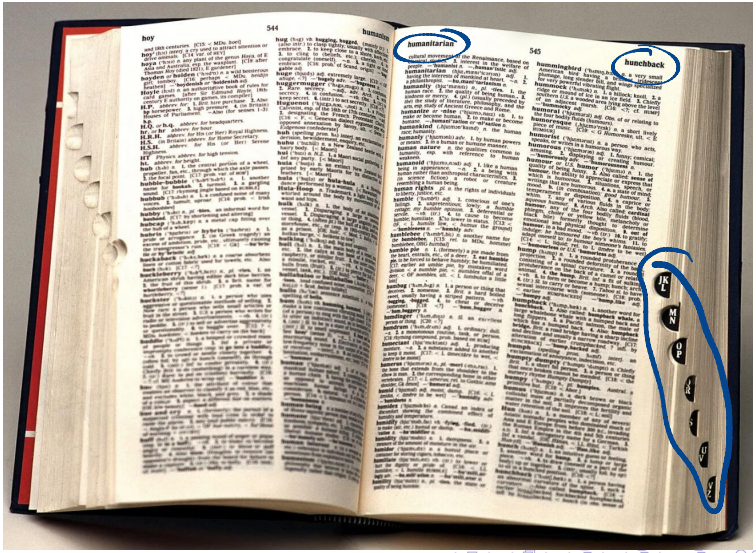
Clustering index

- File is ordered with respect to index values
- Index sequential file
- Dense index — every value is present in the index
  - Index value may match multiple records



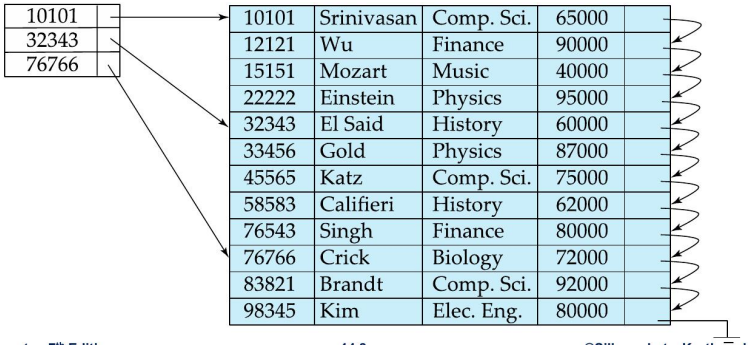
Indexing — sparse indices

- Maintain indices for a subset of values
  - Page headers in a dictionary



Indexing — sparse indices

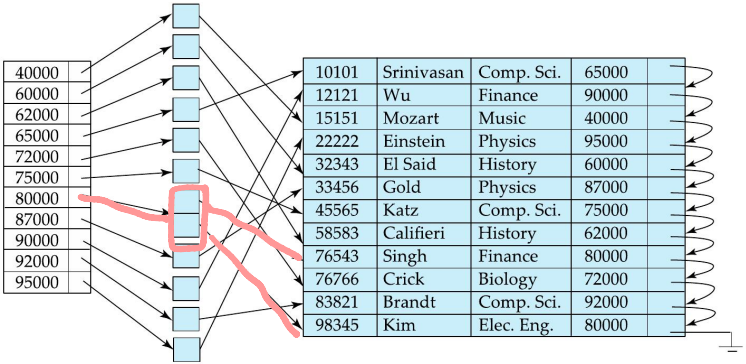
- Maintain indices for a subset of values
  - Page headers in a dictionary
- Align to block boundaries
  - Records are still sequential with respect to index
  - Sparse index identifies first record in each block



Indexing — secondary index

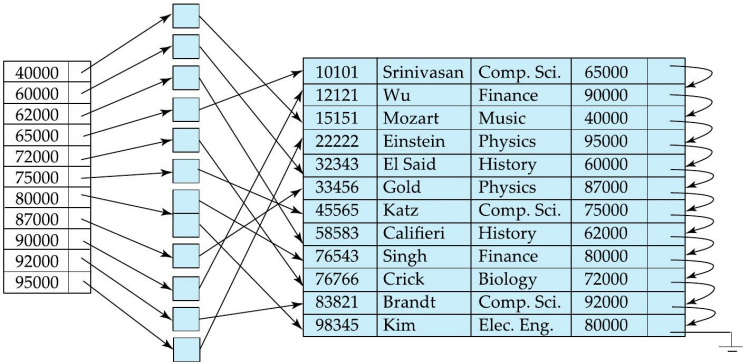
- Index for an attribute that does not match sequence in which table is stored

Must be dense



Indexing — secondary index

- Index for an attribute that does not match sequence in which table is stored
- Key points to block that contains pointers to matching records
  - Can have multiple records for same search key

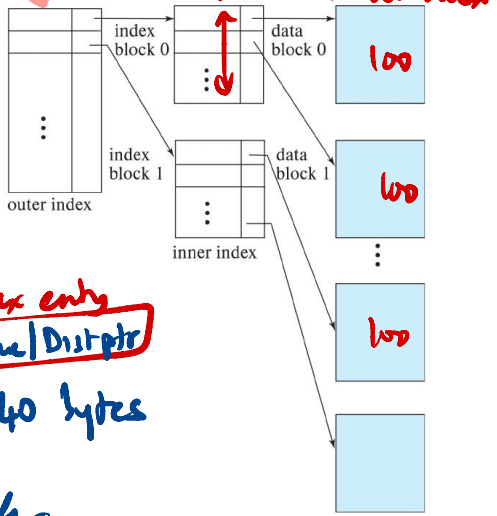


Storage

- Typically, index will not fit in RAM

Storage

- Typically, index will not fit in RAM
- Store index as a sequential file
  - Build a sparse index for the index file
  - Multi-level, till sparse index fits in one block



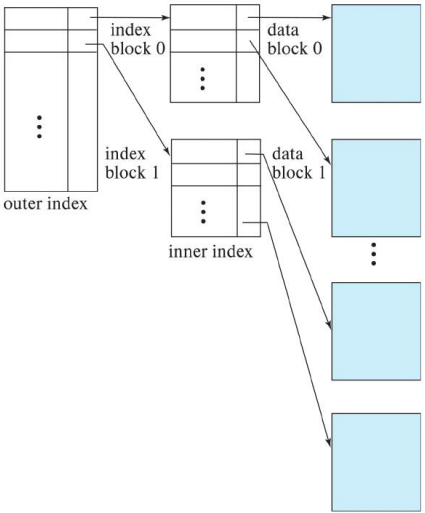
Index entry
Key value | Disk ptr
≈ 40 bytes

4096 byte block
100 values per block ←

10^6 records → 10^4 blocks for index

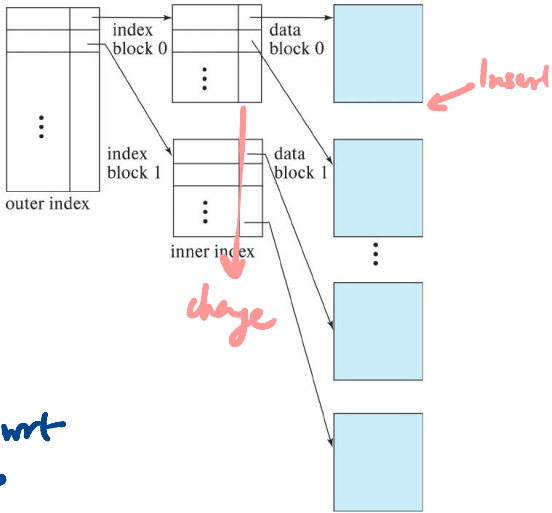
Storage

- Typically, index will not fit in RAM
- Store index as a sequential file
  - Build a sparse index for the index file
  - Multi-level, till sparse index fits in one block
- Binary search to find required key



Storage

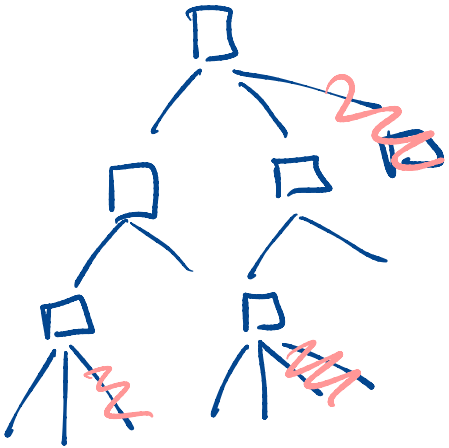
- Typically, index will not fit in RAM
- Store index as a sequential file
  - Build a sparse index for the index file
  - Multi-level, till sparse index fits in one block
- Binary search to find required key
- Idea leads to a more efficient structure



Robust wrt updates

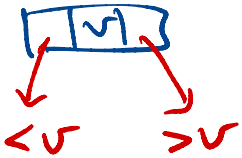
Search trees

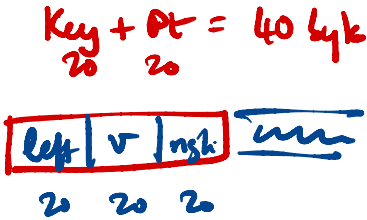
- Binary search trees
  - Binary search on dynamic data
  - Balanced tree has logarithmic height

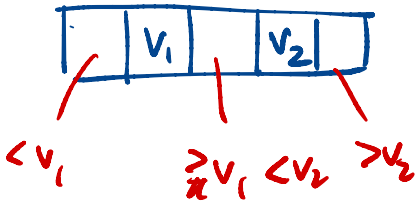


Search trees

- Binary search trees
  - Binary search on dynamic data
  - Balanced tree has logarithmic height
- Block-based access
  - Binary tree node has one search key value, two pointers
  - Block can hold much more

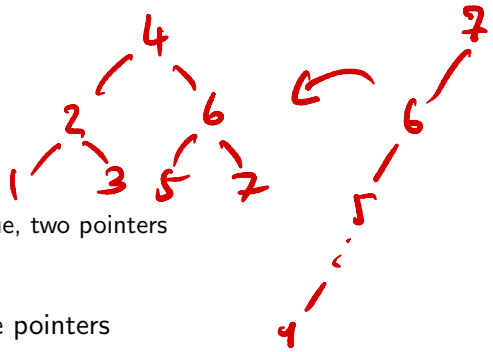






Search trees

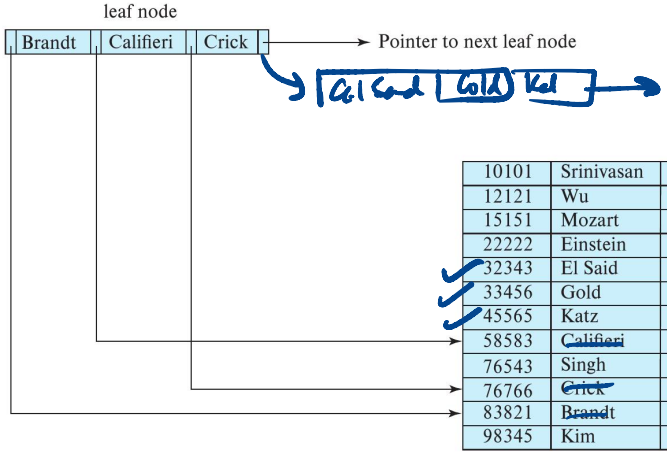
- Binary search trees
  - Binary search on dynamic data
  - Balanced tree has logarithmic height
- Block-based access
  - Binary tree node has one search key value, two pointers
  - Block can hold much more
- Generalize to multiple key values, multiple pointers



P_1	K_1	P_2	...	P_{n-1}	K_{n-1}	P_n

B+ trees

- Leaf nodes form a dense index — linked list of leaves, each one block

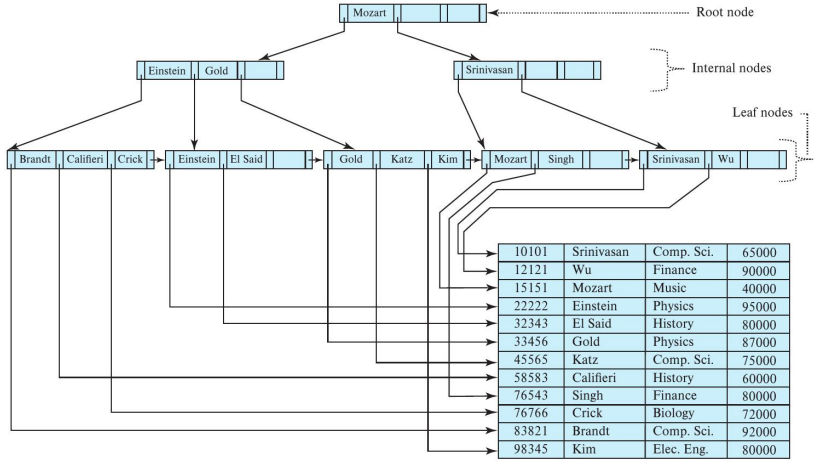


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor file

B+ trees

- Leaf nodes form a dense index — linked list of leaves
- Non-Leaf nodes form a sparse index



B+ trees

- Leaf nodes form a dense index — linked list of leaves
- Non-leaf nodes form a sparse index
- Constraints — assume n keys and pointers can fit in a block
  - Each leaf has at least $\lceil (n-1)/2 \rceil$ key values
  - Each non-leaf has at least $\lceil n/2 \rceil$ pointers
  - Height of the tree is proportional to $\log_{n/2}(N)$

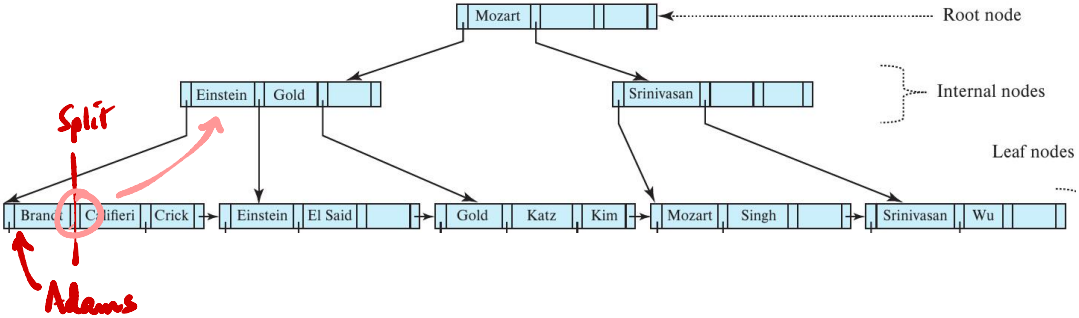
N - table size

In our example $n = 4$

No constraint on root

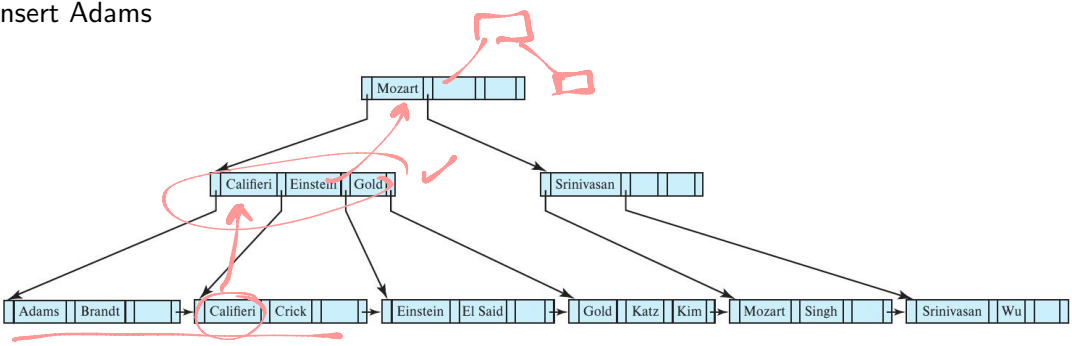
B+ trees — insertion

- Insert Adams



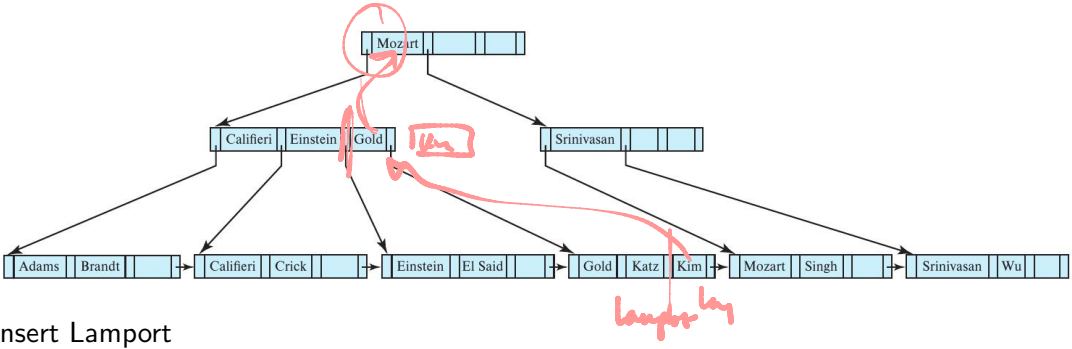
B+ trees — insertion

- Insert Adams



B+ trees — insertion

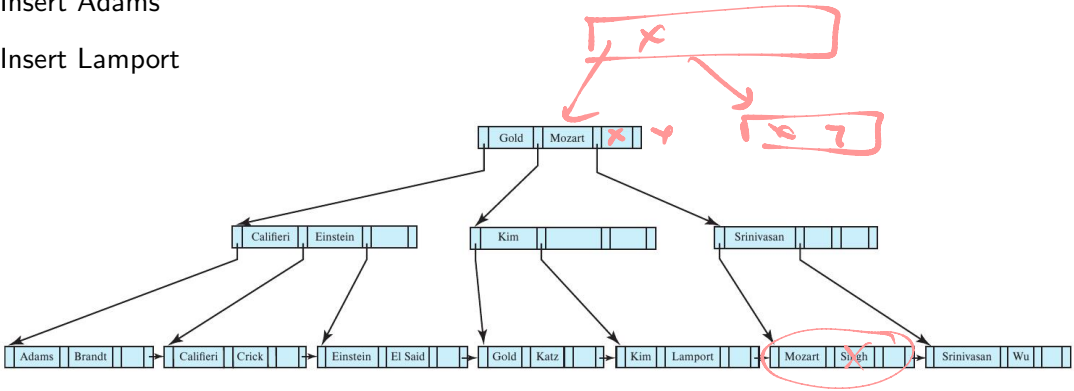
- Insert Adams



- Insert Lamport

B+ trees — insertion

- Insert Adams
- Insert Lamport

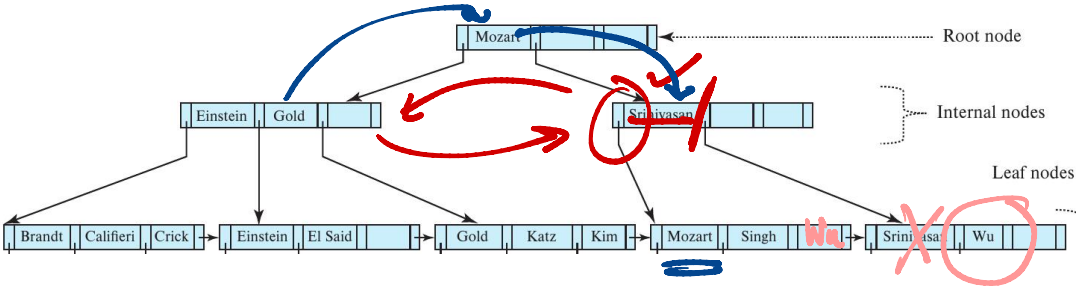


B+ trees — insertion

- Insert Adams
- Insert Lamport
- Recursively insert from leaf level upwards
  - Split nodes when needed and adjust search keys and pointers

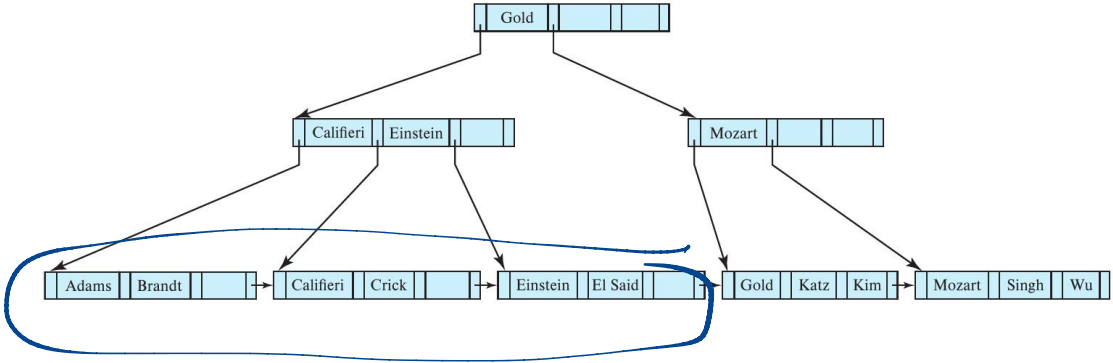
B+ trees — deletion

- Delete Srinivasn



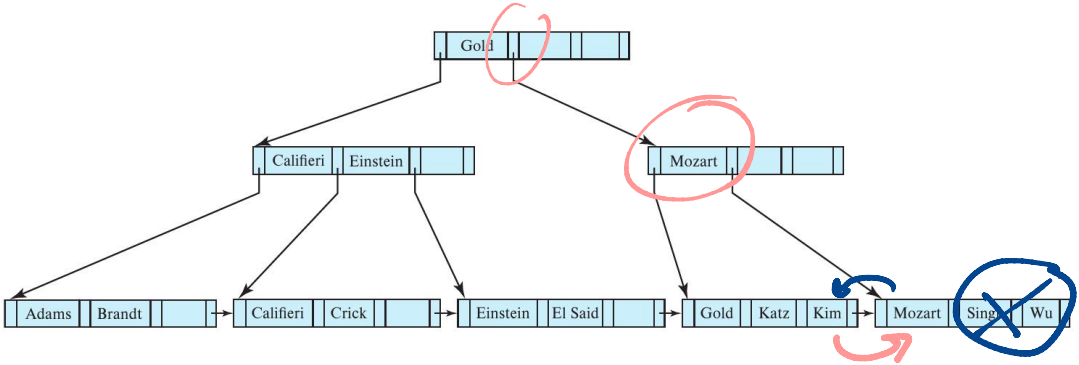
B+ trees — deletion

- Delete Srinivasn



B+ trees — deletion

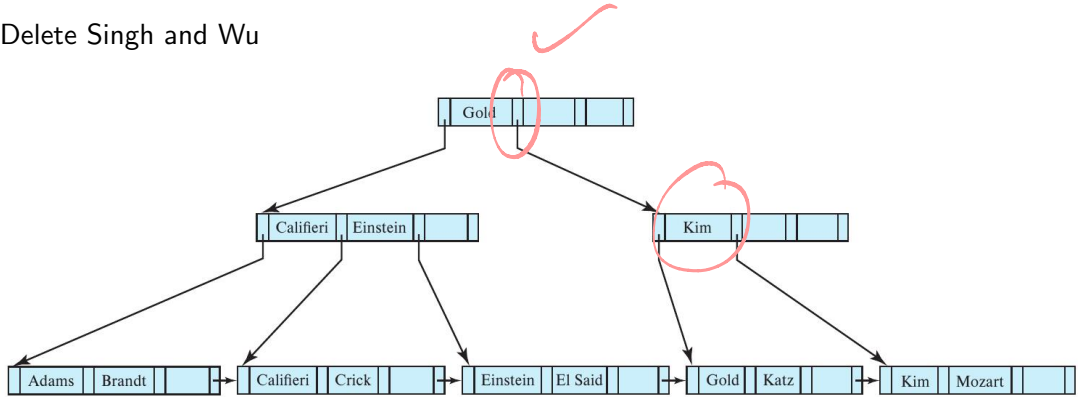
- Delete Srinivasn



- Delete Singh and Wu

B+ trees — deletion

- Delete Srinivasn
- Delete Singh and Wu

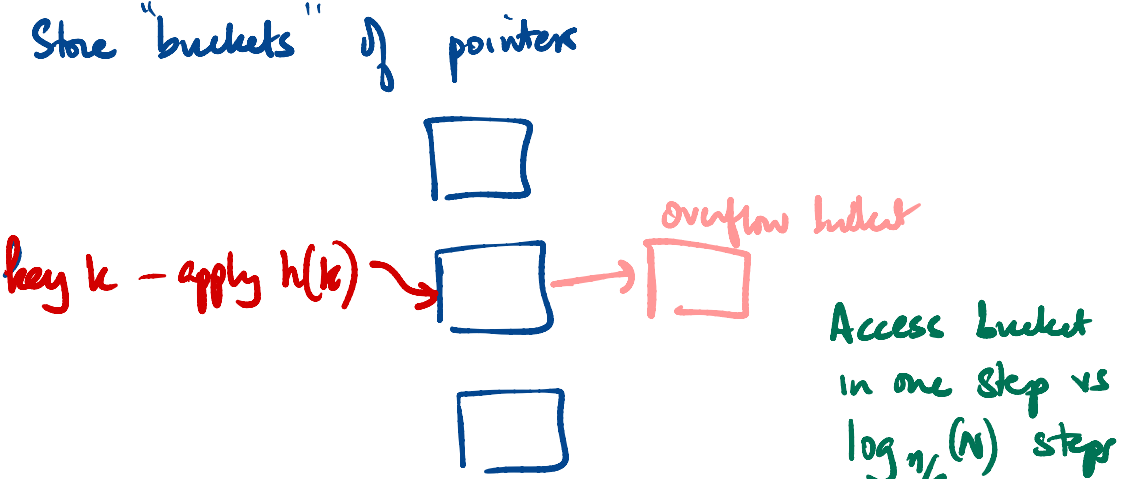


B+ trees — deletion

- Delete Srinivasn
- Delete Singh and Wu
- Recursively delete from leaf level upwards
  - Merge or redistribute with neighbour

B+ tree can answer both point queries & range queries

Hashing


Store "buckets" of pointers
overflow bucket
key k – apply h(k)
Access bucket in one step vs $\log_{n/2}(N)$ steps
Good for point queries, useless for range queries

Indices on multiple keys

```
select ID
from instructor
where dept_name = 'Finance' and salary = 80000;
```

Separate indices — search on both — intersect

build multi attribute index on (dept_name, salary)

Indices on multiple keys

select *ID*
from *instructor*
where *dept_name* = 'Finance' **and** *salary* = 80000;

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Creating an index in SQL

- Not part of SQL standard

Creating an index in SQL

- Not part of SQL standard
- `create index I on T(A,B,C)`

Creating an index in SQL

- Not part of SQL standard
- `create index I on T(A,B,C)`
- `drop index I`

Creating an index in SQL

- Not part of SQL standard
- `create index I on T(A,B,C)`
- `drop index I`
- DBMS may create and maintain index on its own for efficient processing