Database Management Systems

Madhavan Mukund

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

Lecture 8, 10 November 2023

Storing data

- RAM vs Hard disk vs SSD 10^{-6}
- Blocks and latency

?

10^{-3}

ms

Python $\approx 10^7$ ops/sec

C++ $\simeq 10^8$ . . . 10^9

memory (RAM) ops

RAM data transfer is in nanoseconds (10^{-9})

Disk ⇄ RAM

CPU

↓

"Blocks" 4KB

2 delays

Find block — "Seek"

Transfer block

Storing data

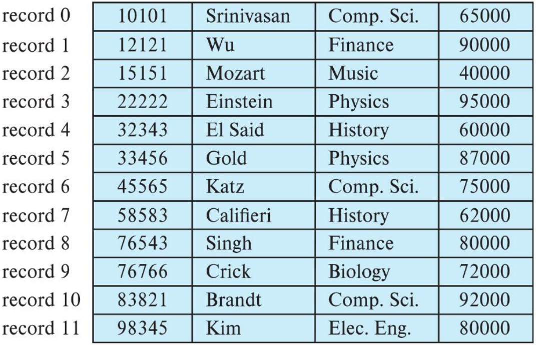
- RAM vs Hard disk vs SSD
- Blocks and latency

"Accounting" – Seeks & block transfers

RAM operations are "free"

Fixed length records

- Blocks and block boundaries

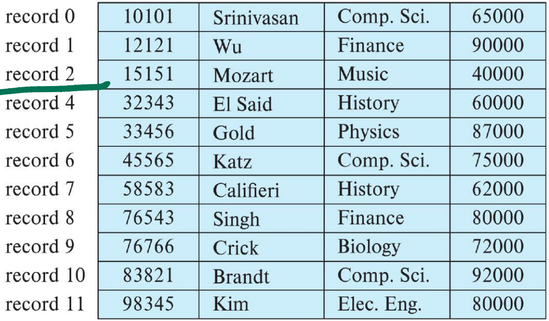


record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Deleting a record

- Compress

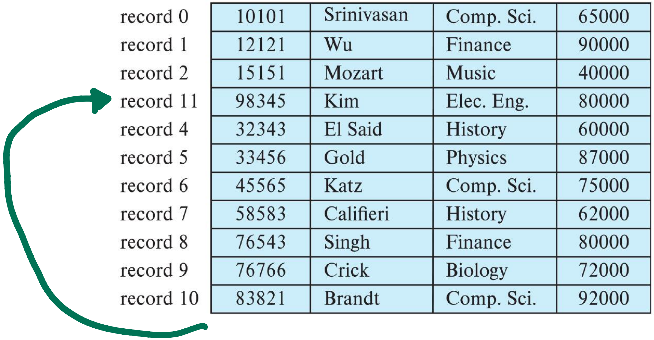
Delete Einstein



record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Deleting a record

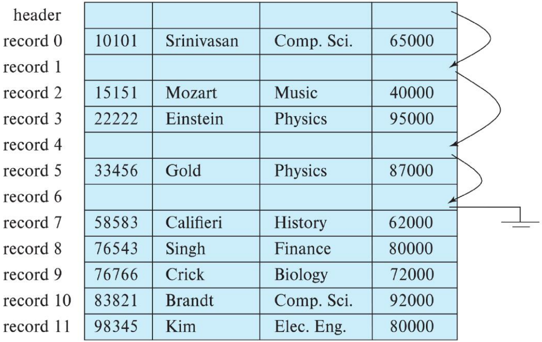
- Compress
- Move last record



record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Deleting a record

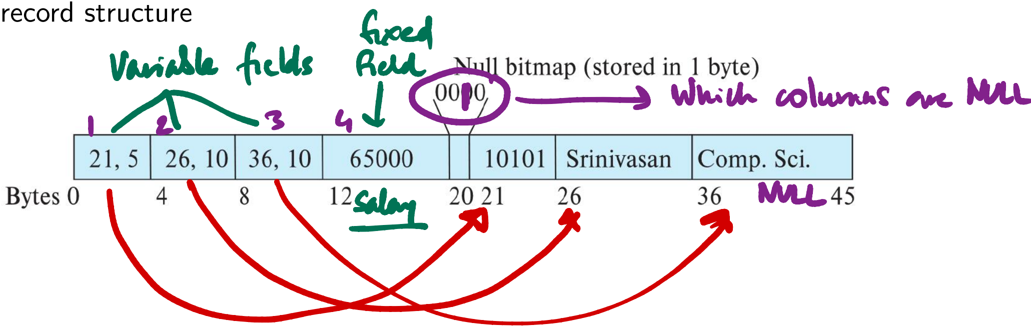
- Compress
- Move last record
- Maintain free list of empty slots



header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Variable length records

- Single record structure



Null bitmap (stored in 1 byte)

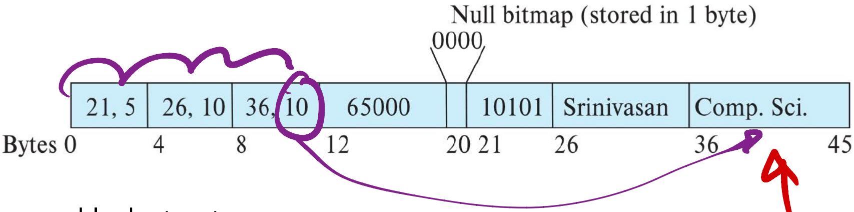
0000

21, 5	26, 10	36, 10	65000		10101	Srinivasan	Comp. Sci.

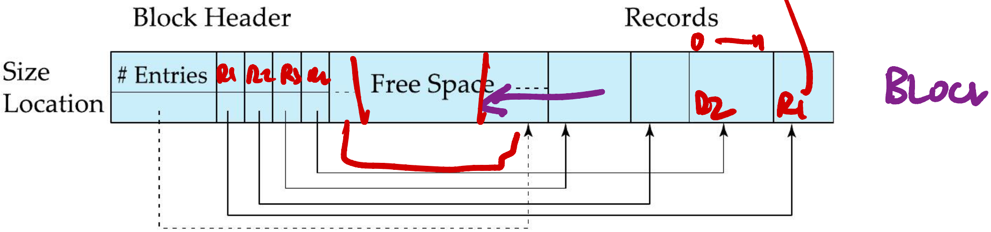
Bytes 0 4 8 12 20 21 26 36 45

Variable length records

- Single record structure



- Slotted page block structure

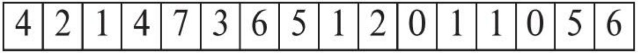


Storing tables — heap file organization

- Use first available free slot

Storing tables — heap file organization

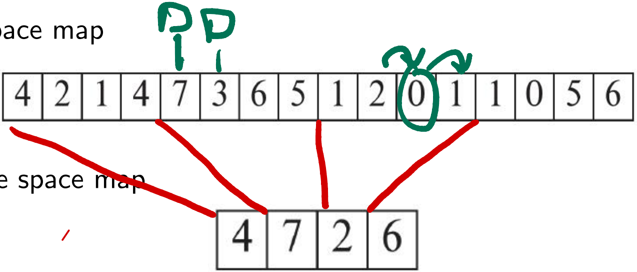
- Use first available free slot
- Maintain free space map



4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

Storing tables — heap file organization

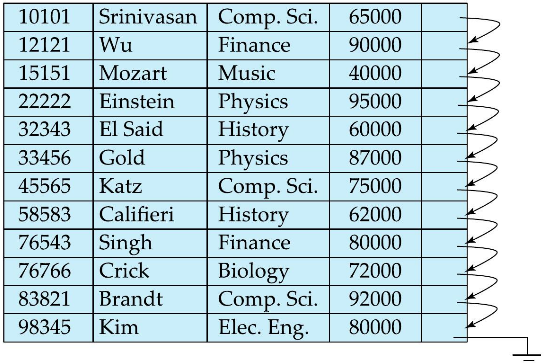
- Use first available free slot
- Maintain free space map
- Second level free space map



4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

4	7	2	6

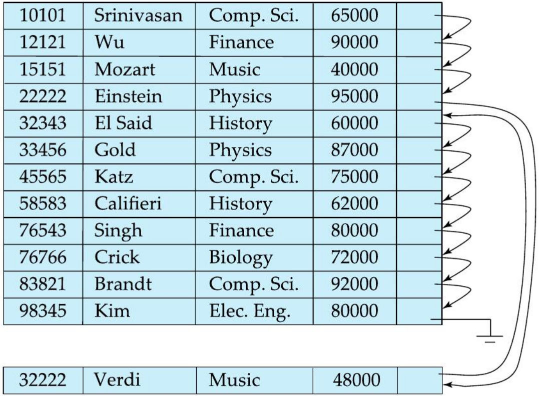
Storing tables — sequential file organization



10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Storing tables — sequential file organization

- Overflow block



10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Indexing

- Why build an index?

Indexing

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

Indexing

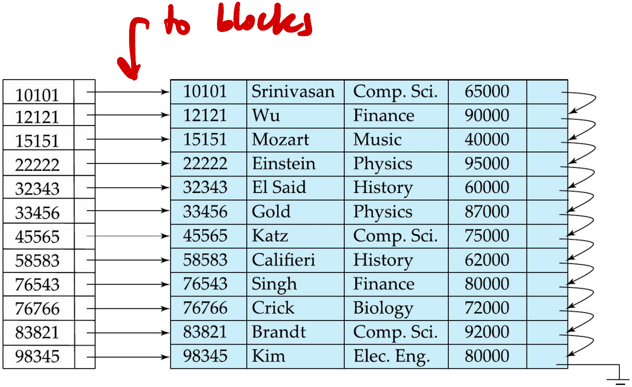
- Why build an index?
- Search key
  - As opposed to superkey, candidate key, . . .
  - May need multiple search keys for a table
- Types of queries — point vs range
  - `ID = "10102"`
  - `salary > 75000`

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



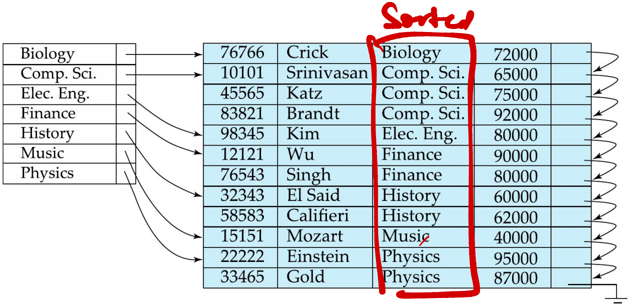
to blocks

10101
12121
15151
22222
32343
33456
45565
58583
76543
76766
83821
98345

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

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



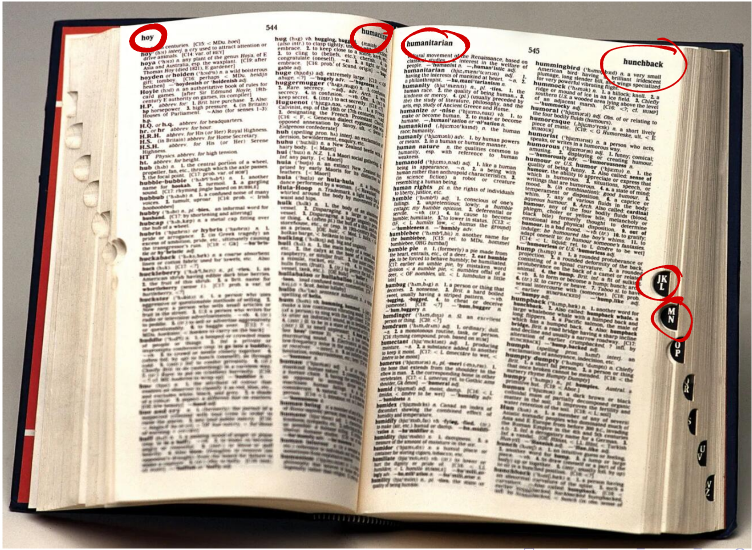
Sorted

Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

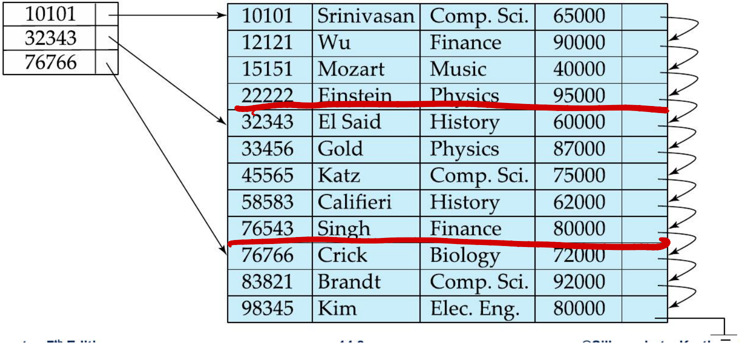
Indexing — sparse indices

- 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

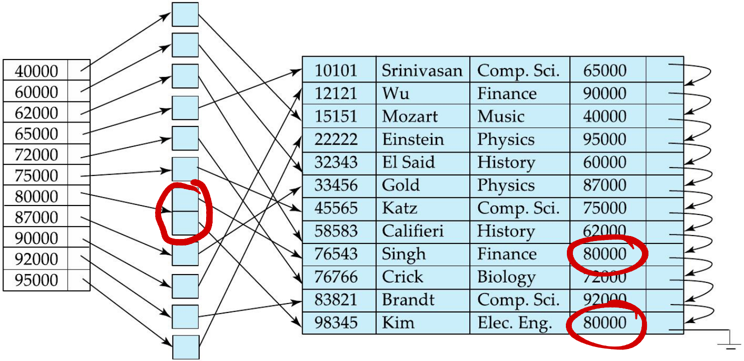


10101
32343
76766

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Indexing — secondary index

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

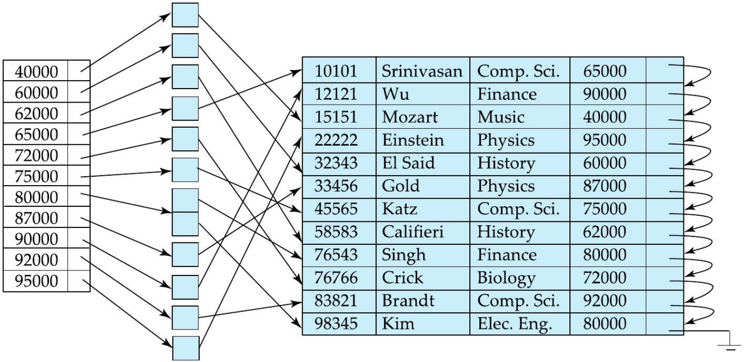


40000
60000
62000
65000
72000
75000
80000
87000
90000
92000
95000

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

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



40000
60000
62000
65000
72000
75000
80000
87000
90000
92000
95000

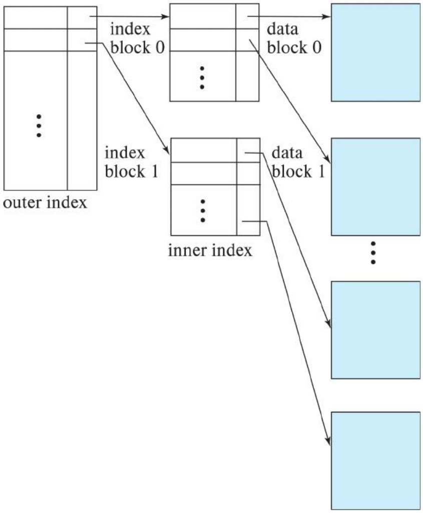
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Storage

- Typically, index will not fit in RAM

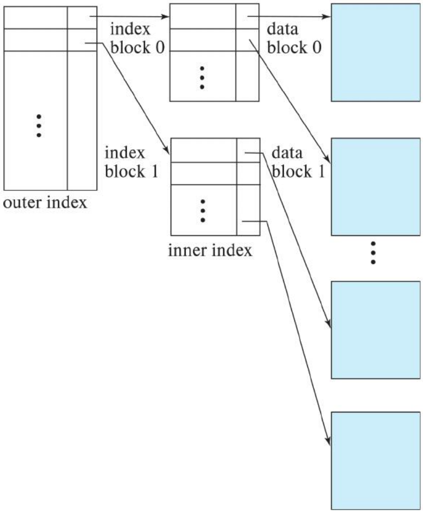
Storage

- 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



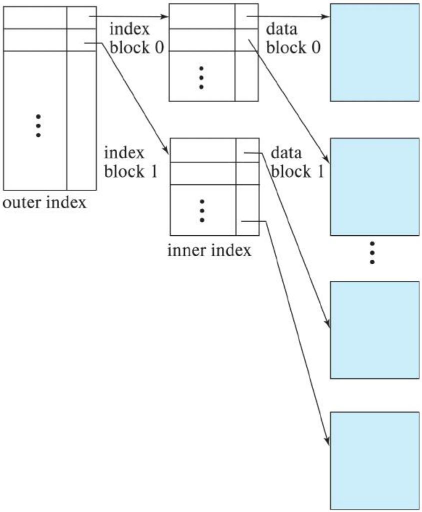
Storage

- 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



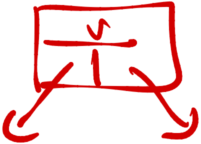
Storage

- 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



Search trees

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



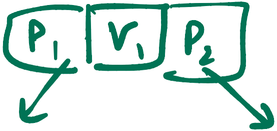
Node size = 1 block

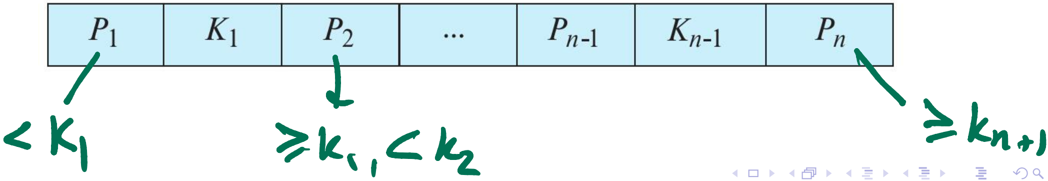
Search trees

- 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





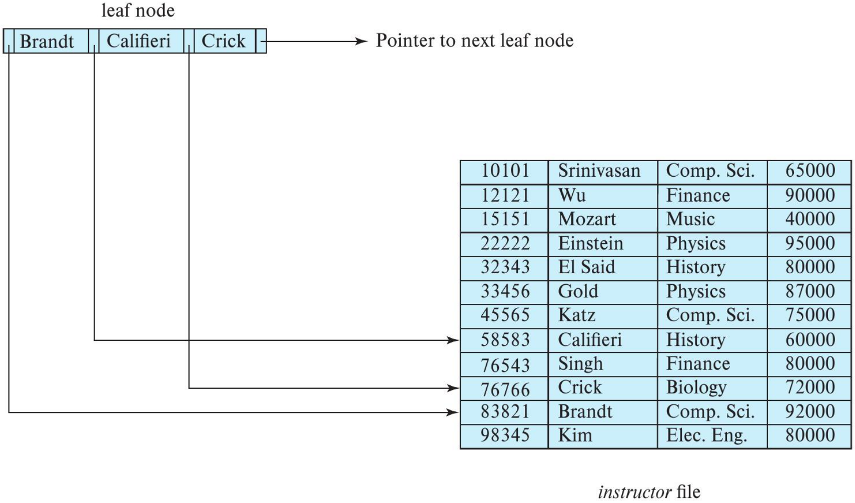
P_1	K_1	P_2	...	P_{n-1}	K_{n-1}	P_n

B+ trees

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

4 ptrs
3 keys

n = 4

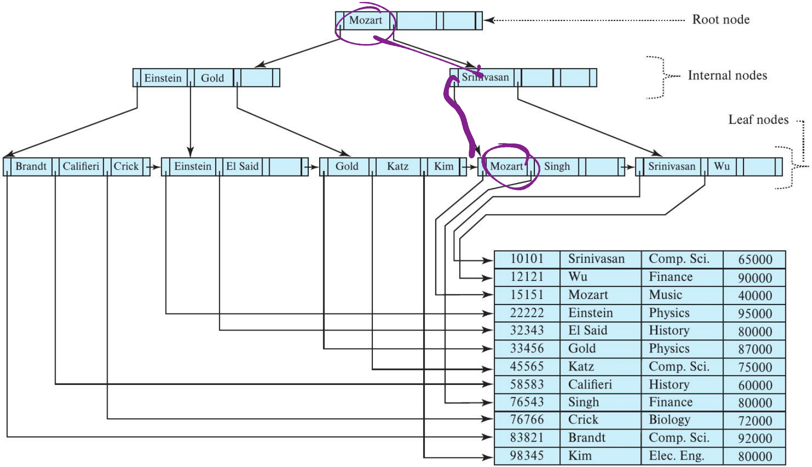


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000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



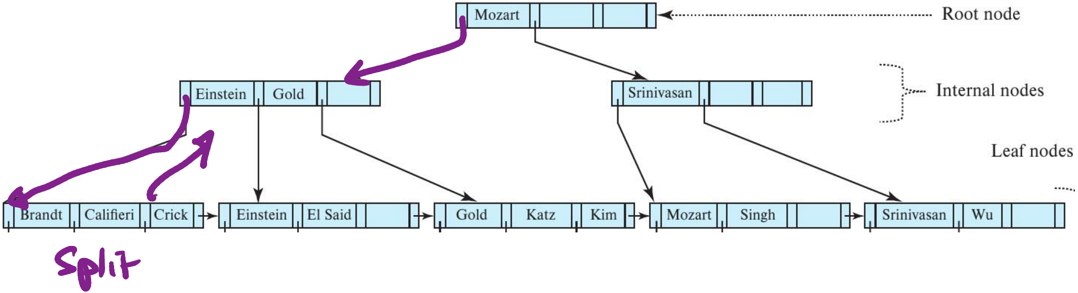
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000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

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 $\log_{n/2}(n)$

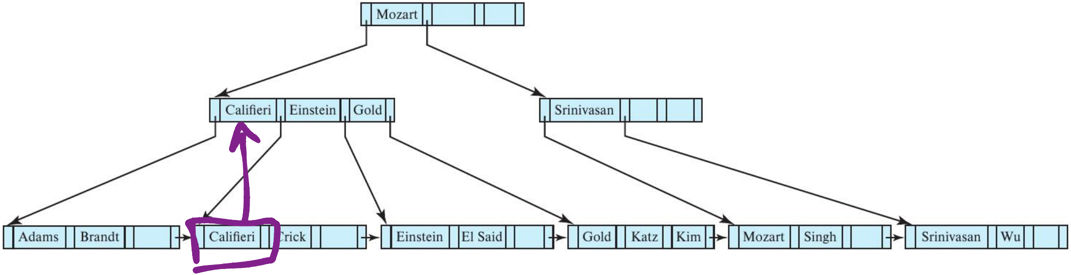
B+ trees — insertion

- Insert Adams



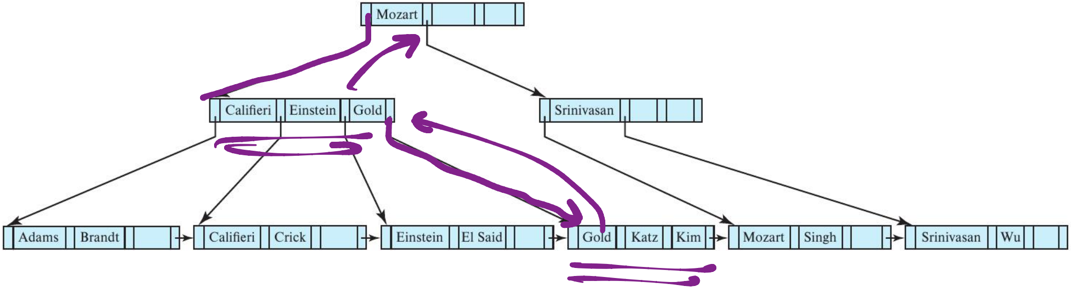
B+ trees — insertion

- Insert Adams



B+ trees — insertion

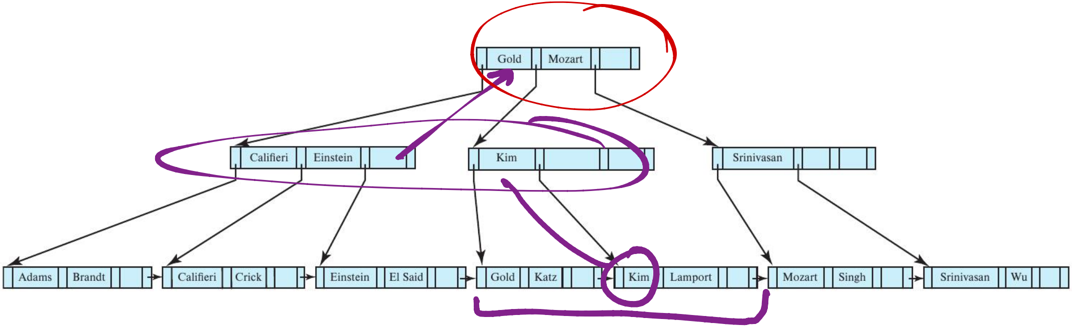
- Insert Adams



- Insert Lamport

B+ trees — insertion

- Insert Adams
- Insert Lamport

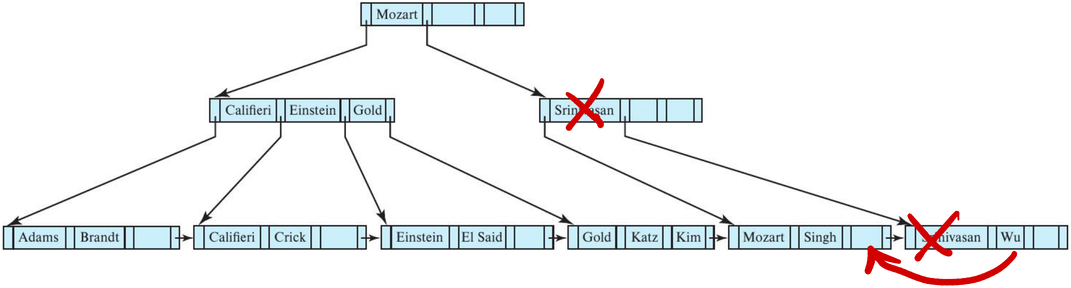


B+ trees — insertion

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

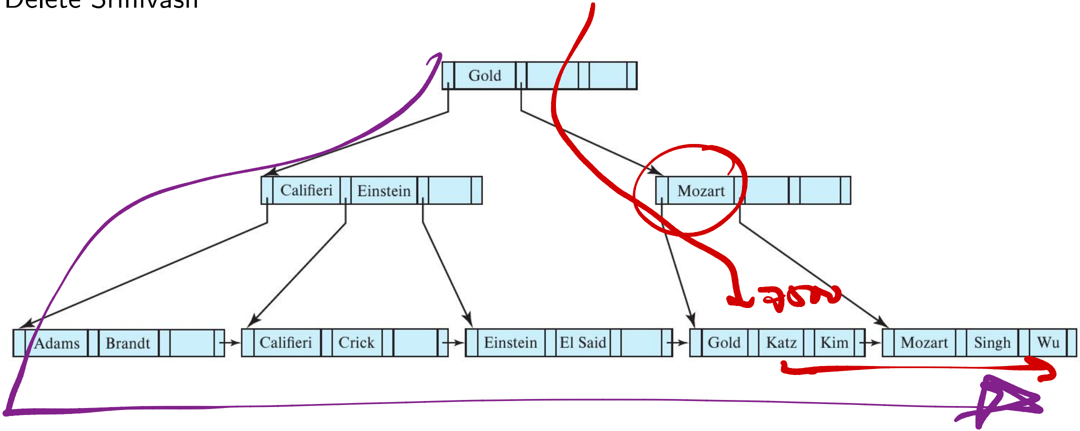
B+ trees — deletion

- Delete Srinivasn



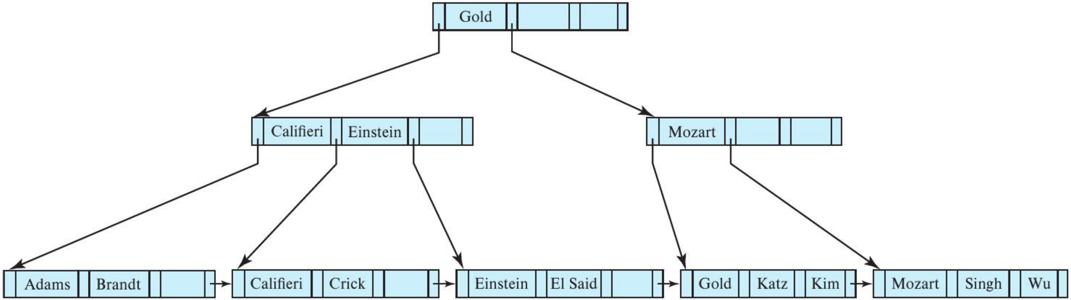
B+ trees — deletion

- Delete Srinivasn



B+ trees — deletion

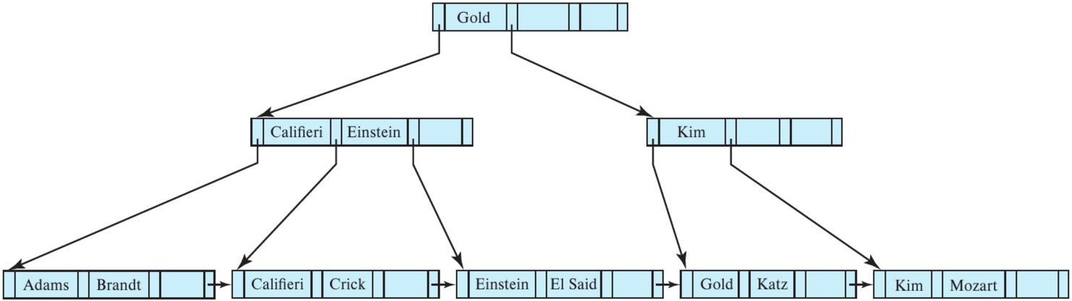
- Delete Srinivasn



- 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