RDBMS and SQL

Madhavan Mukund

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

Lecture 9, 30 October 2025

Storing data

- RAM vs Hard disk vs SSD
- Blocks and latency

10^9 reads/writes per sec

10^6

10^3 reads/writes per sec

Data size is large

Store on disk "Secondary storage"

Primary - RAM

1.4×10^9 Aadhar Cards

100 bytes / card

140×10^9 bytes

RAM $\simeq$ 2 minutes

10^6 factor more than

Storing data

- RAM vs Hard disk vs SSD
- Blocks and latency

Read a chunk at a time

4kB

Search for the block to read

Read time

Seek time

140×10^9 bytes

35×10^6 blocks

$= 10^3$ reads/sec

35,000 second

600 minute

22 hours

Storing data

- RAM vs Hard disk vs SSD
- Blocks and latency

Measuring performance

Read a block $\approx 10^3$ secs

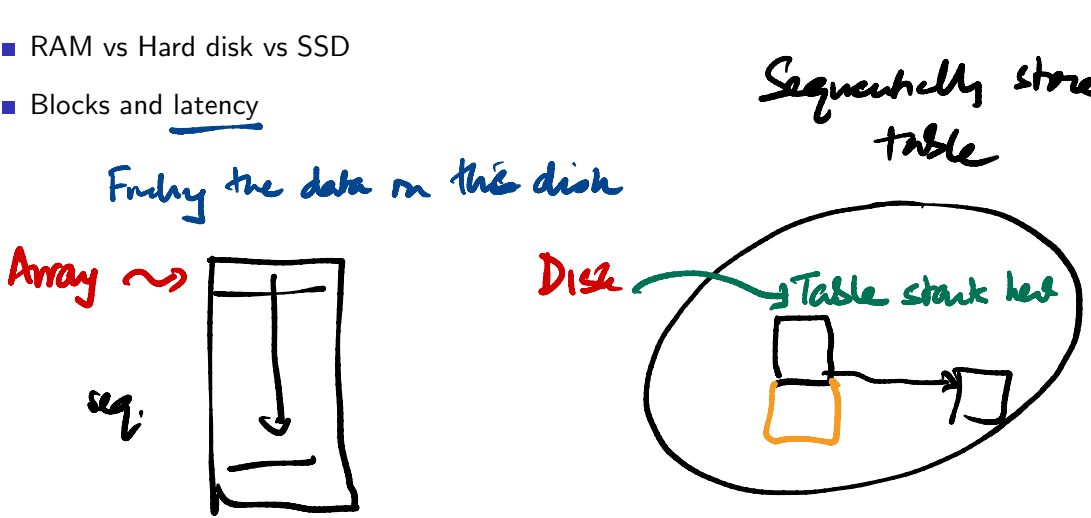
Search for an item 4KB fraction of a second

In-memory computation is "free"

Only disk accesses counted

Storing data

- RAM vs Hard disk vs SSD
- Blocks and latency



Fixed length records

- Blocks and block boundaries

Table size # rows N

Block size B

Row size R

$\frac{B}{R}$ rows per block

$N / (B/R) \Rightarrow$ # blocks

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

Metadata

Deleting a record

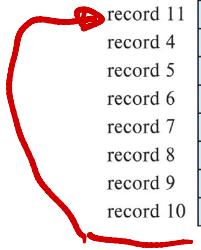
- Remove Einstein
- Compress

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

In memory - "free"

Deleting a record

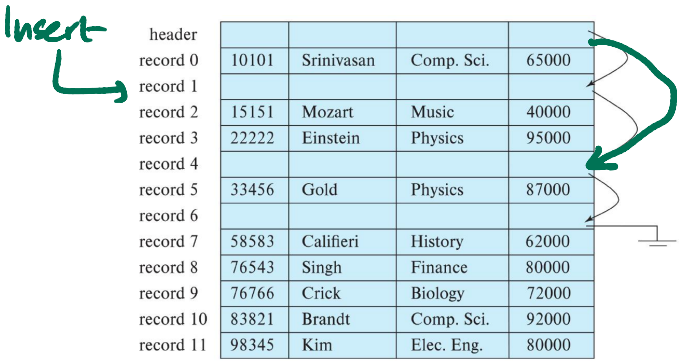
- Remove Einstein
- Compress
- Move last record



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

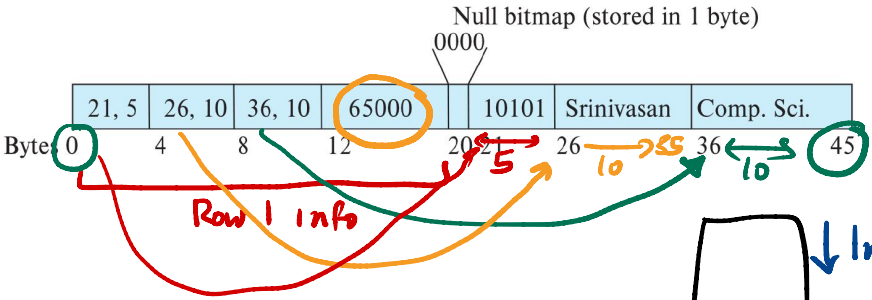
Deleting a record

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



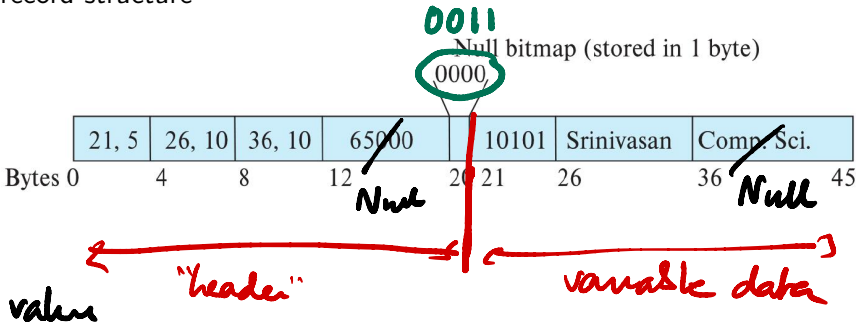
Variable length records

- Single record structure



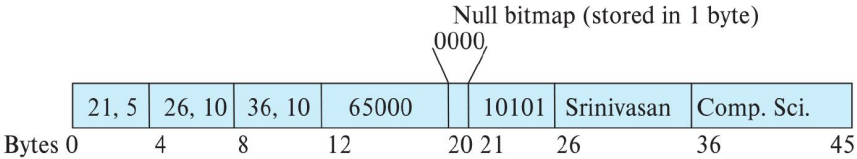
Variable length records

- Single record structure

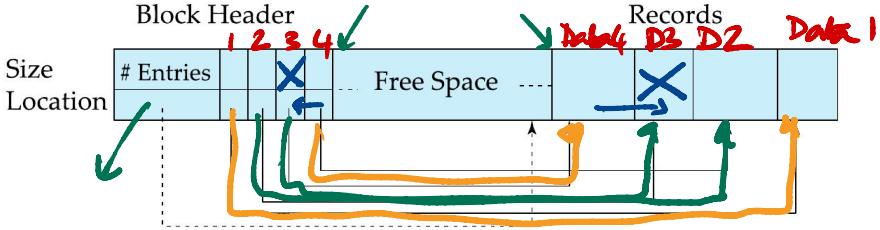


Variable length records

- Single record structure



- Slotted page block structure



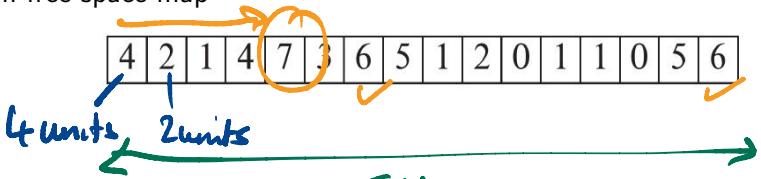
Storing tables — heap file organization

- Use first available free slot

Storing tables — heap file organization

- Use first available free slot
- Maintain free space map

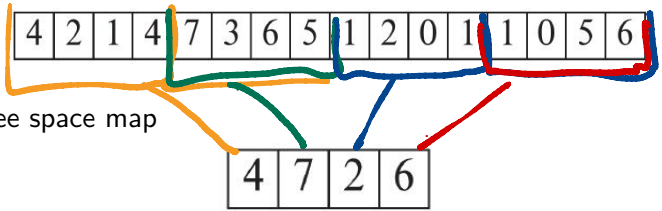
Fixed length / Variable – free space



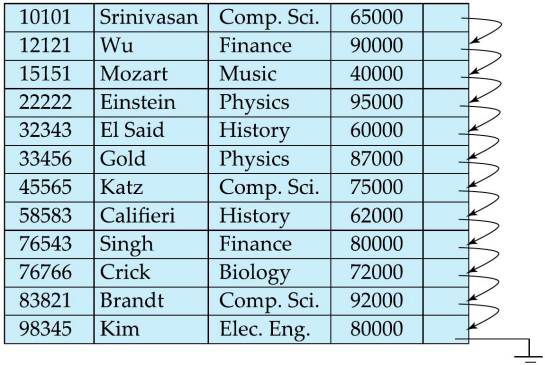
Insert a row of 6 units size

Storing tables — heap file organization

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



Storing tables — sequential file organization

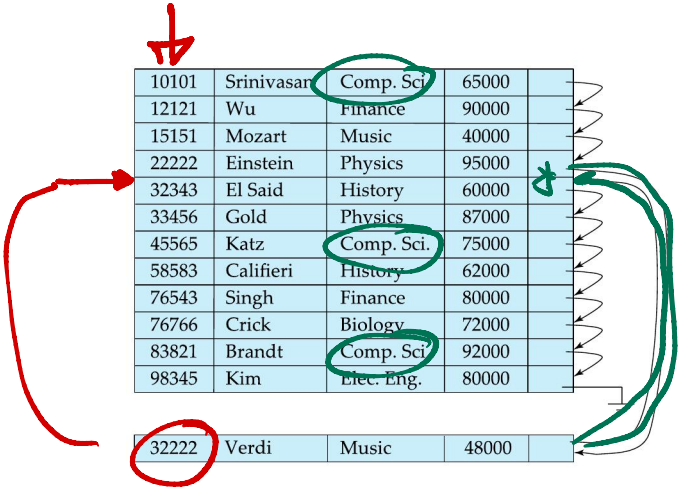


10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp.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

Storing tables — sequential file organization

- Overflow block

Sorted by ID



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`

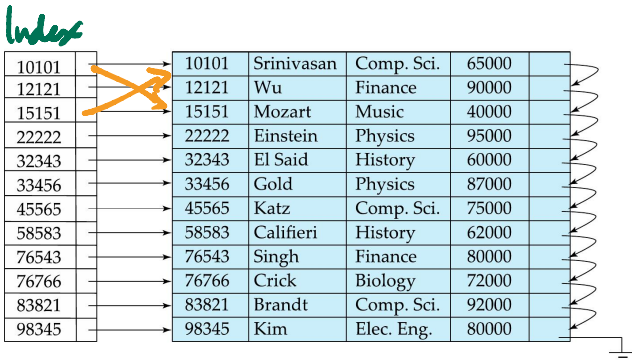
Indexing

- 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

Card Catalogue for library

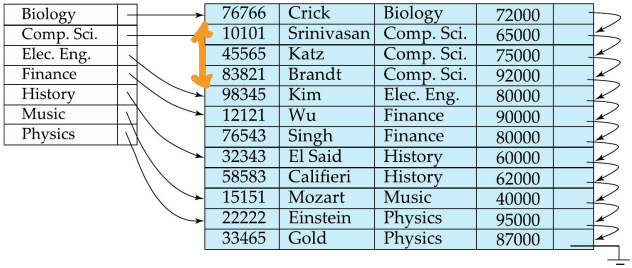
Clustering index

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



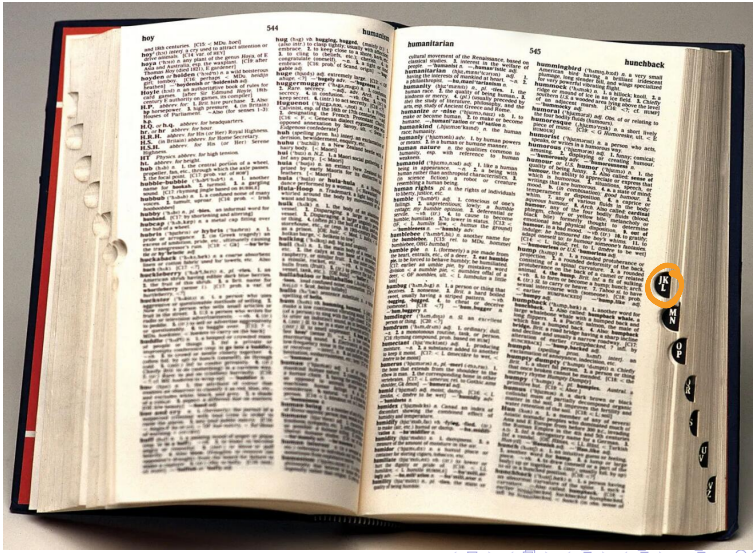
Clustering index

- 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



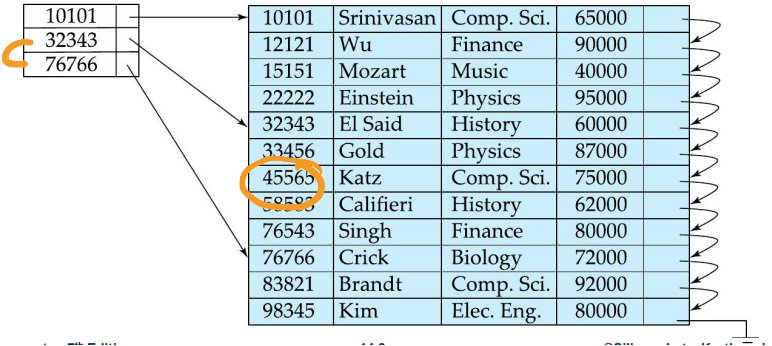
Indexing — sparse indices

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



Indexing — sparse indices

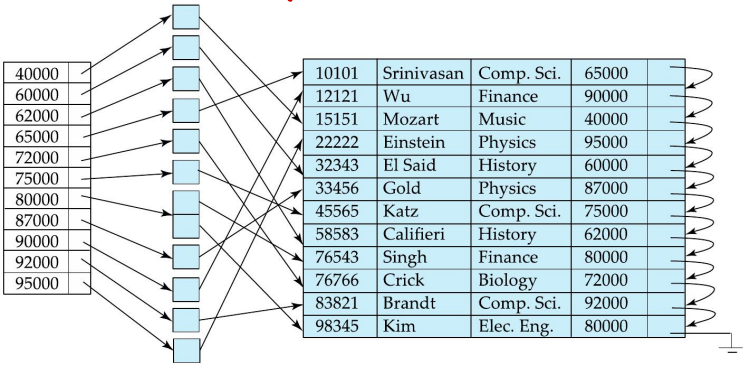
- Maintain indices for a subset of values
  - Page headers in a dictionary
- Align to block boundaries
  - Records are still sequential with respect to index
  - Sparse index identifies first record in each block



Indexing — secondary index

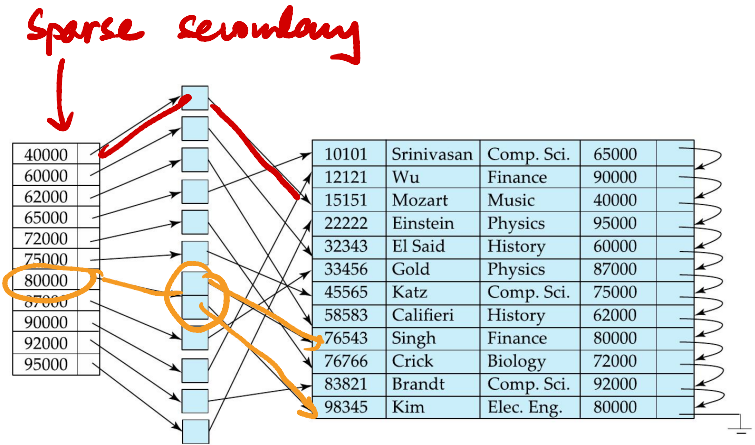
Not sorted wrt index value

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



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

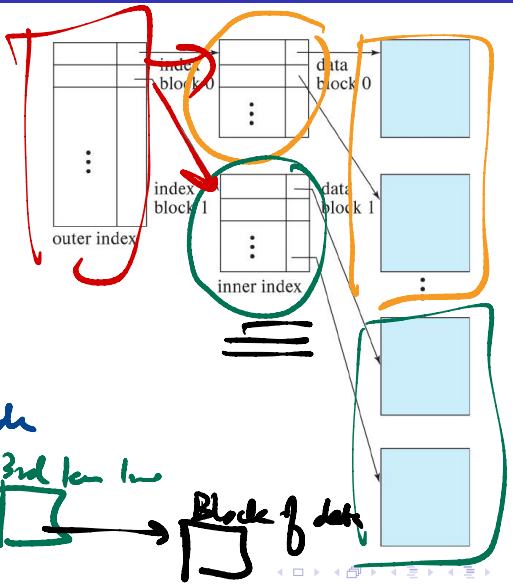


Storage

- Typically, index will not fit in RAM

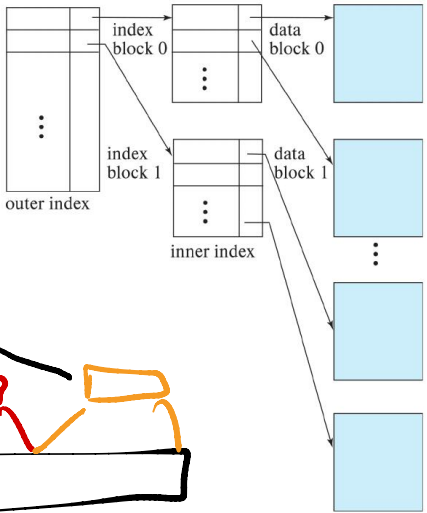
Storage

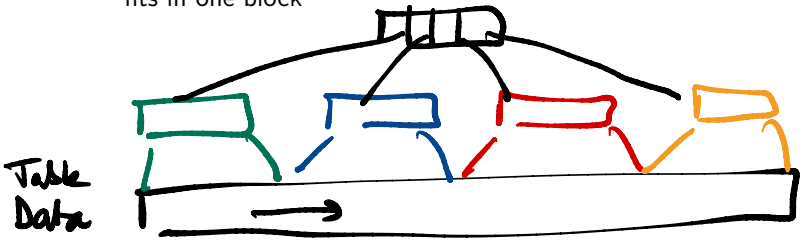
- Typically, index will not fit in RAM
- Store index as a sequential file
  - Build a sparse index for the index file
  - Multi-level, till sparse index fits in one block



Storage

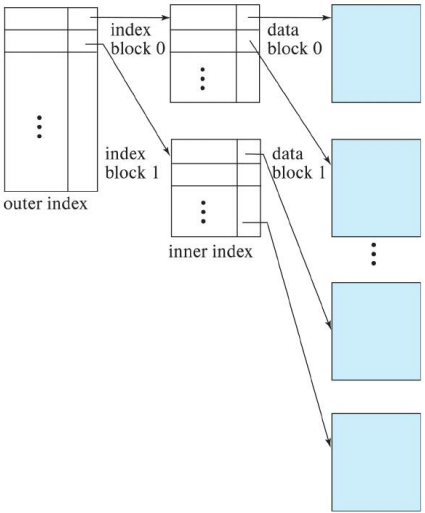
- Typically, index will not fit in RAM
- Store index as a sequential file
  - Build a sparse index for the index file
  - Multi-level, till sparse index fits in one block





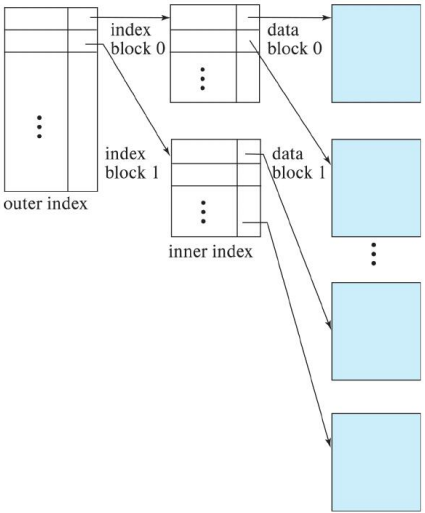
Storage

- 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



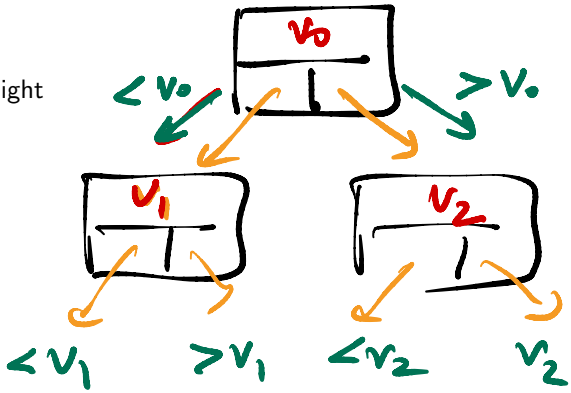
Search trees

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

Search trees

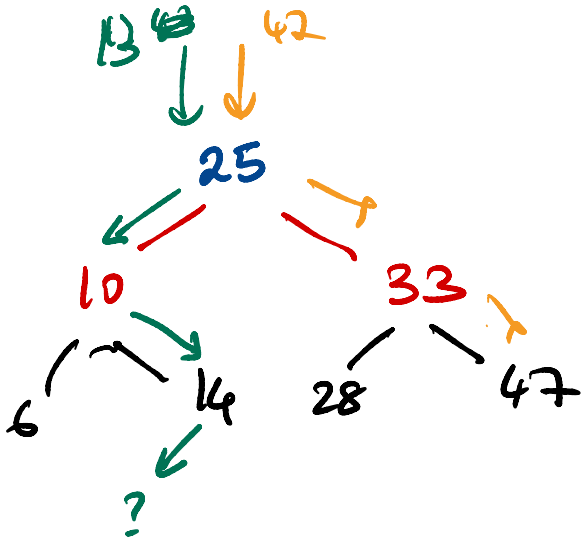
- Binary search trees
  - Binary search on dynamic data
  - Balanced tree has logarithmic height

Binary tree



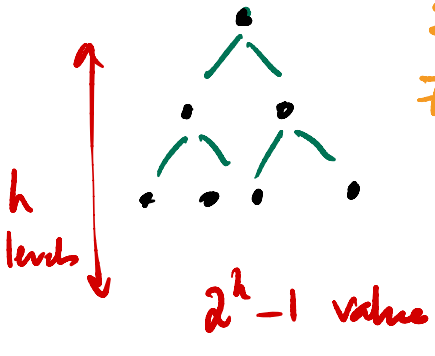
Search trees

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

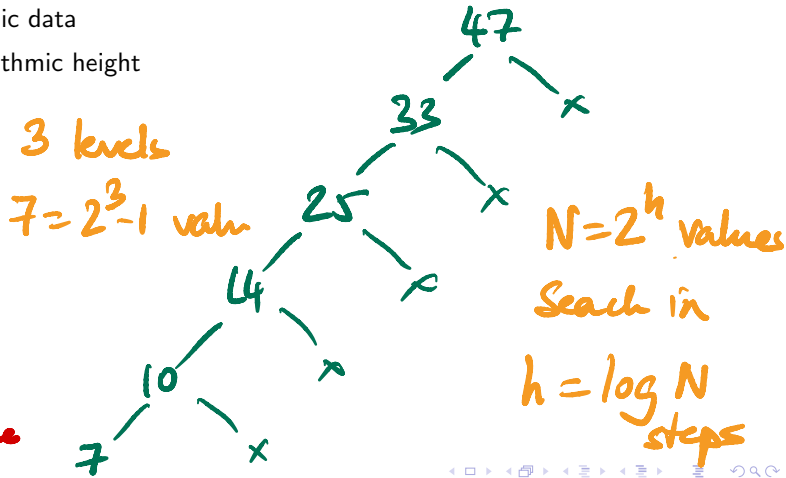


Search trees

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

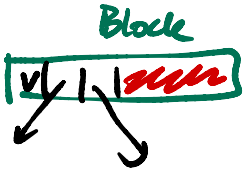


Unbalanced



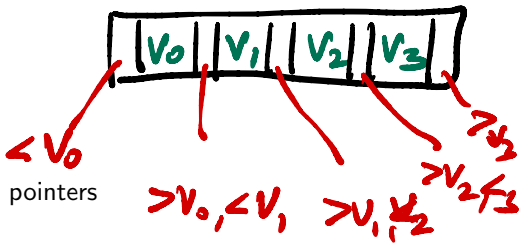
Search trees

- 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



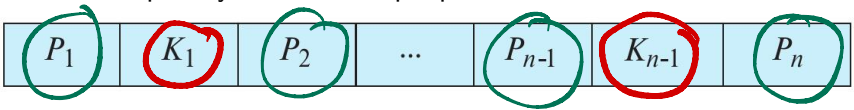
Search trees

- 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



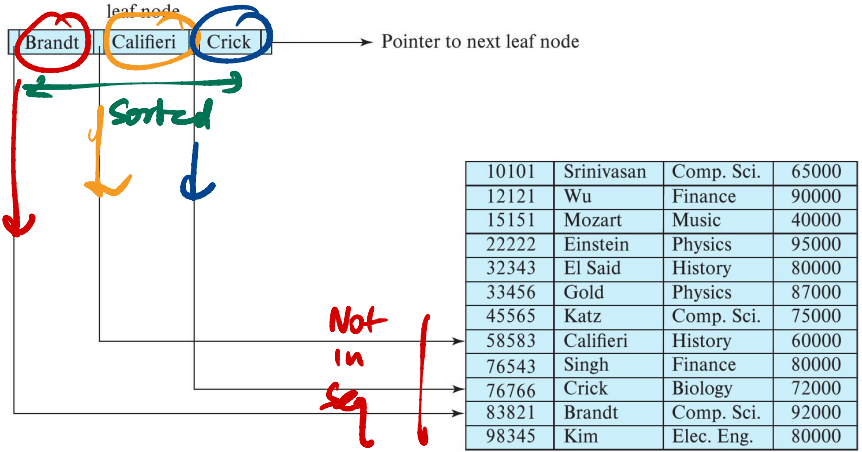
Search trees

- 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



B+ trees

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

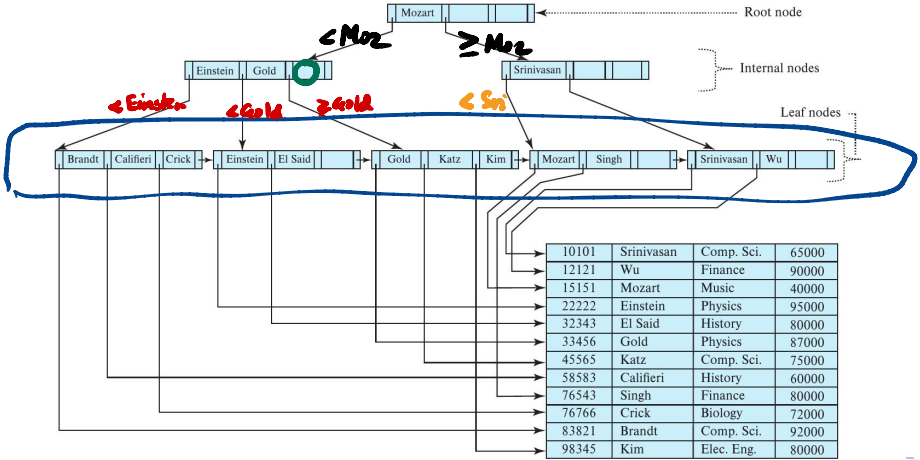


10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000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

k-ary branching
height = $\log_k(N)$