Database Indexing

Recap

- Our goal is to efficiently find the records specified in the 'where' clause of a query
- Binary search is a fast search algorithm
- Applying binary search to a table stored on disk has some problems:
  1. we can only sort the records of a table according to one key
  2. even on that one key, how to do the binary search on disk blocks?

Idea: create separate index table

Think of an index at the end of a book:

- it is stored separately from the rest of the book
- the entries are ordered for fast search
- it takes up space
- entries refer to one or more page numbers

Index for a table

index for 'name' field

search key	records
Brandt	
Califieri	
Crick	
Einstein	
…	

instructor

ID	name	dept_name	salary
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

- search key is an attribute used to look up records
- index is ordered, so fast binary search can be used

Using the index

SQL: `select salary from instructor where name=“Crick”;`

```
CREATE INDEX I1 ON INSTRUCTOR(NAME);
```

Perform binary search on index

Get records from file using record ptrs in index

index for ‘name’ field

search key	records
Brandt	
Califieri	
Crick	
Einstein	
…	

instructor

ID	name	dept_name	salary
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

More on indexes

- multiple indexes can be used for one table
- multiple record pointers per index row are allowed
- indexes themselves are tables

dept_name index

search key	records
Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
…	

instructor

ID	name	dept_name	salary
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

Index maintenance

Question: What to do if record inserted?

Question: What to do if instructor record deleted?

index for 'name' field

search key	records
Brandt	
Califieri	
Crick	
Einstein	
…	

instructor

ID	name	dept_name	salary
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

Index pros/cons

Pros:

- allows for binary search on multiple fields in a table

Cons:

- space to store indexes
- time to update indexes as table changes

Example: index on a primary key

In this example, index has same number of rows as instructor table

ID is the "search key"

Primary Index - Clustered

index

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

instructor

ID	name	dept_name	salary
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

Omitting some values from the index

index

10101	
22222	
45565	
76766	

Sparse Index

instructor

ID	name	dept_name	salary
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

Index on a different field

index

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

instructor

ID	name	dept_name	salary
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
33456	Gold	Physics	87000

Note that the index has fewer rows than the table, but all search key values are in the index.

When are multiple pointers needed?

index

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

instructor

ID	name	dept_name	salary
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

Secondary Index (non-clustered)

Recap

We’ve considered various methods for searching data on disk

- Linear search
- Binary search
- Binary search with ordered indexes

Ordered index pros and cons:

+ fast

- complexity

- space overhead

- time overhead when records added/deleted

Reminder: dense index

index

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

instructor

ID	name	dept_name	salary
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

In this example, index has same number of rows as instructor table

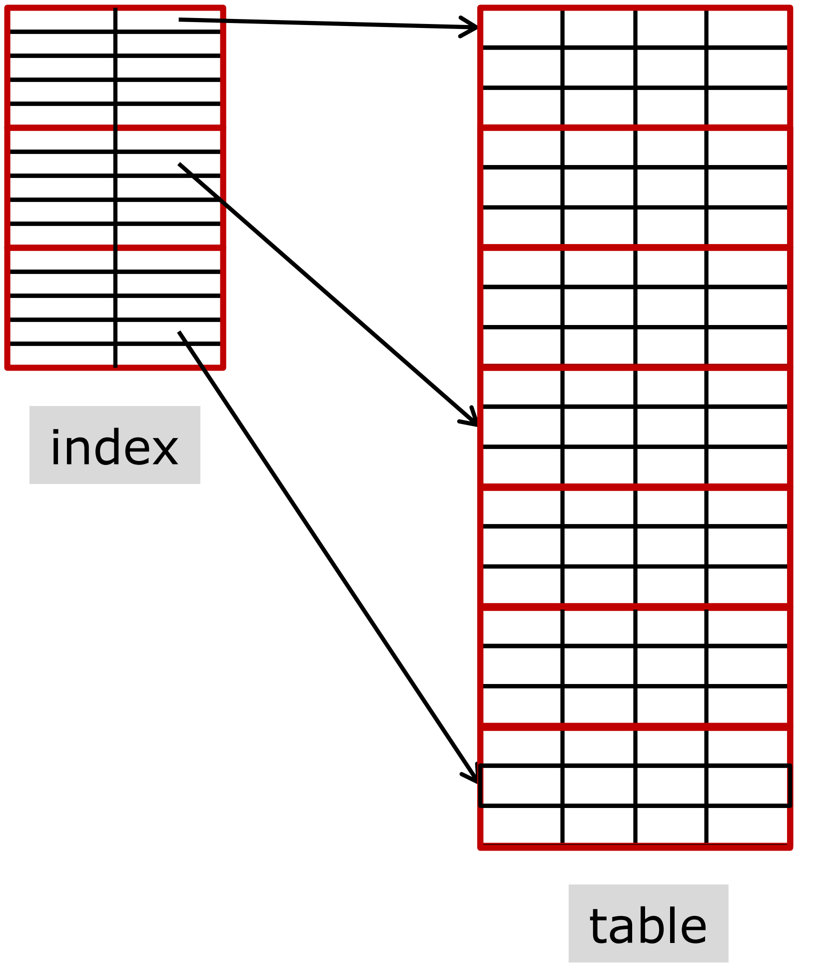
Search with dense index

Search process:

- given search key value
- perform binary search on index
- when index entry found, go to record in a block of table

If **index is small**, it can be kept in memory.

This search is fast.



What happens when table is large?

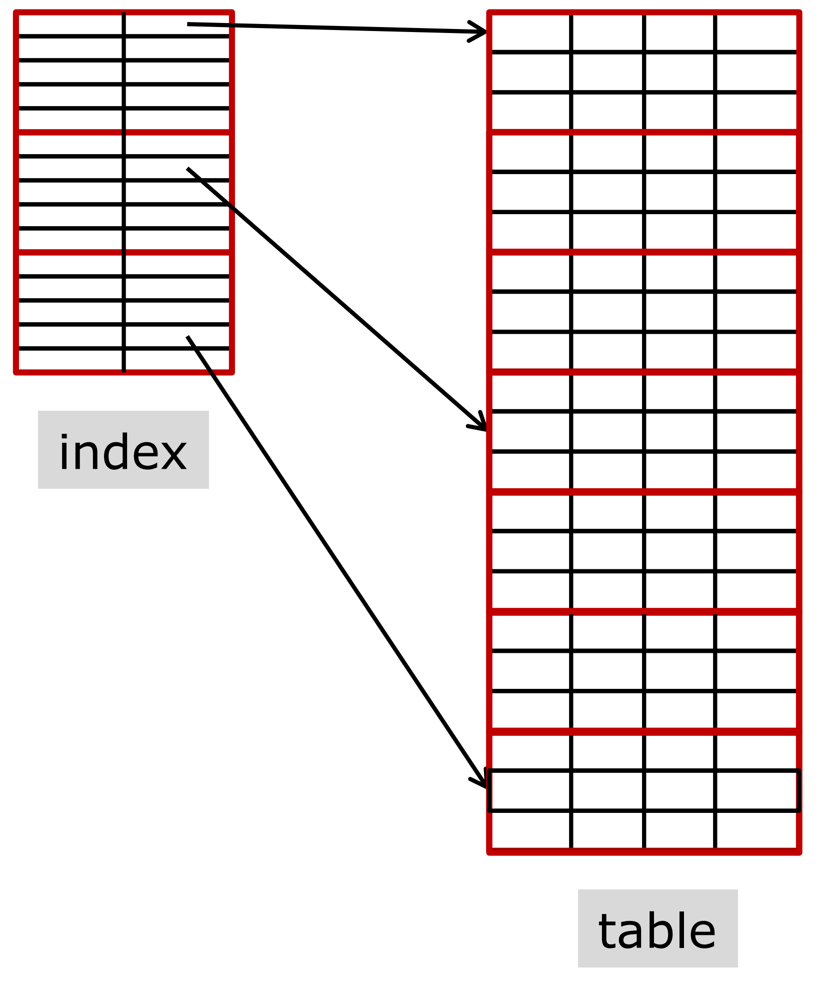
Suppose table has 1 million rows.

If 100 index entries per disk block, then index is 10,000 blocks.

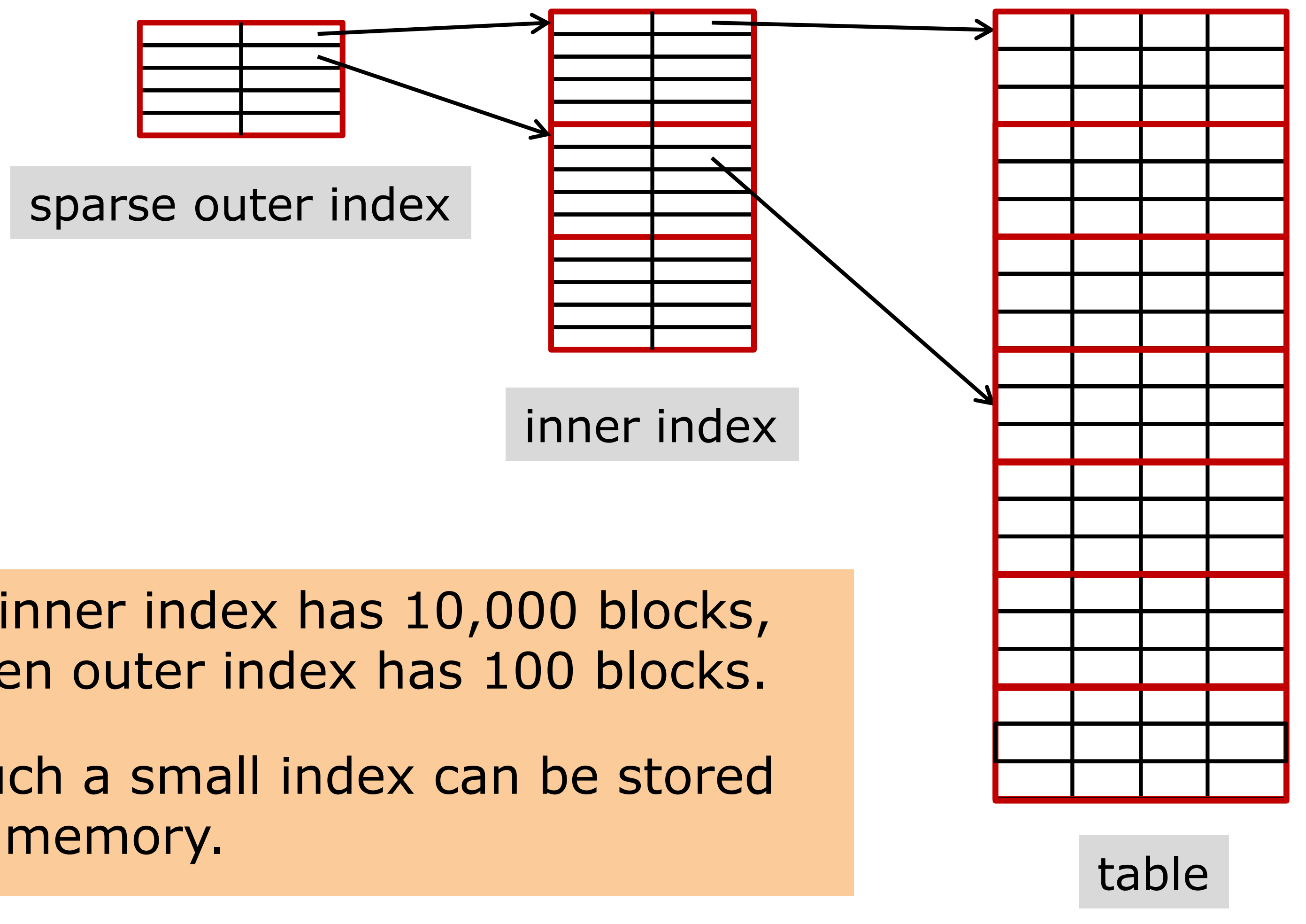
Index is probably too big for memory.

Binary search on index now requires about 14 (log2 10,000) disk accesses.

search time: ~ 140 ms



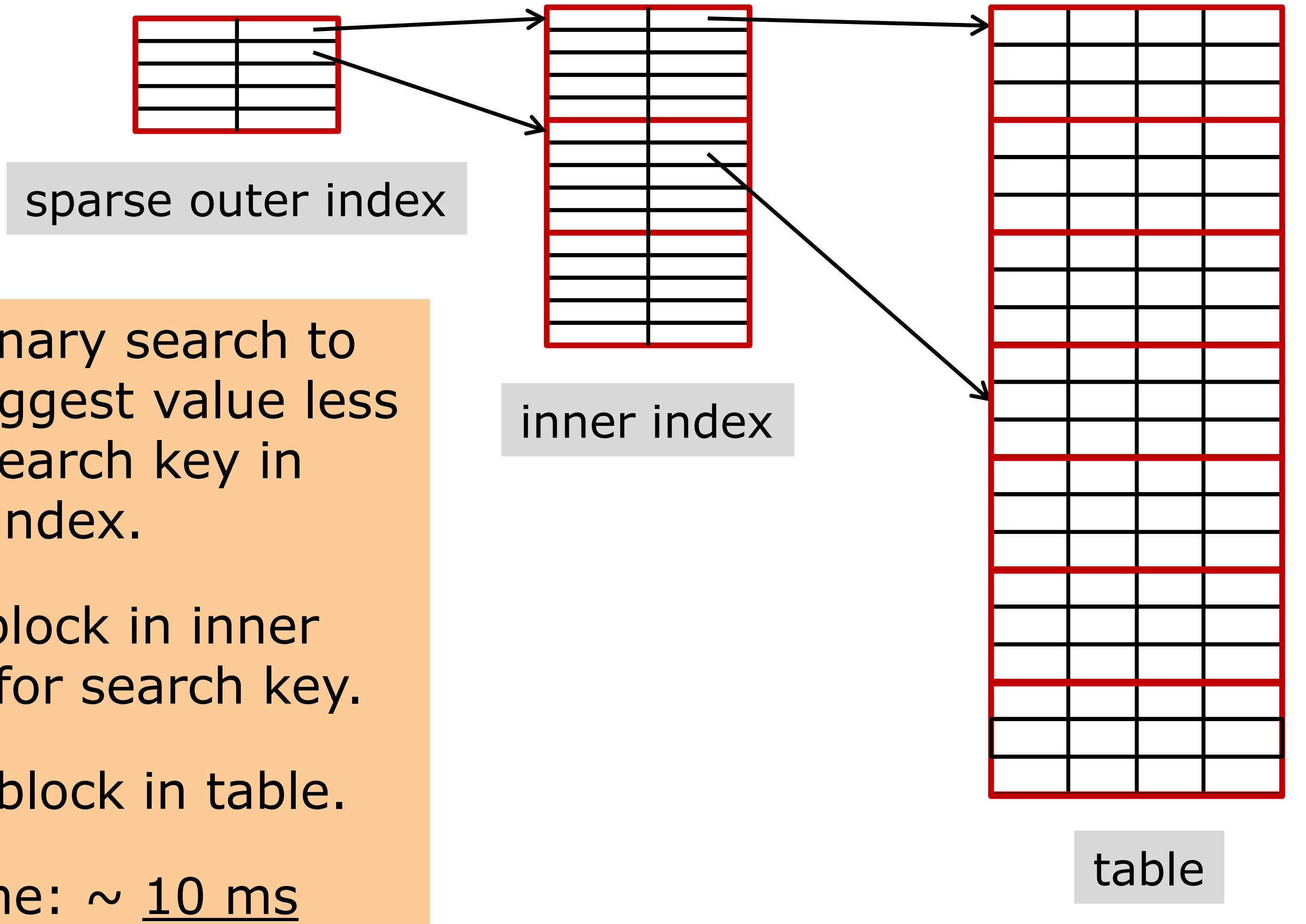
Idea: add an “outer” index



If inner index has 10,000 blocks, then outer index has 100 blocks.

Such a small index can be stored in memory.

Search process with two indexes



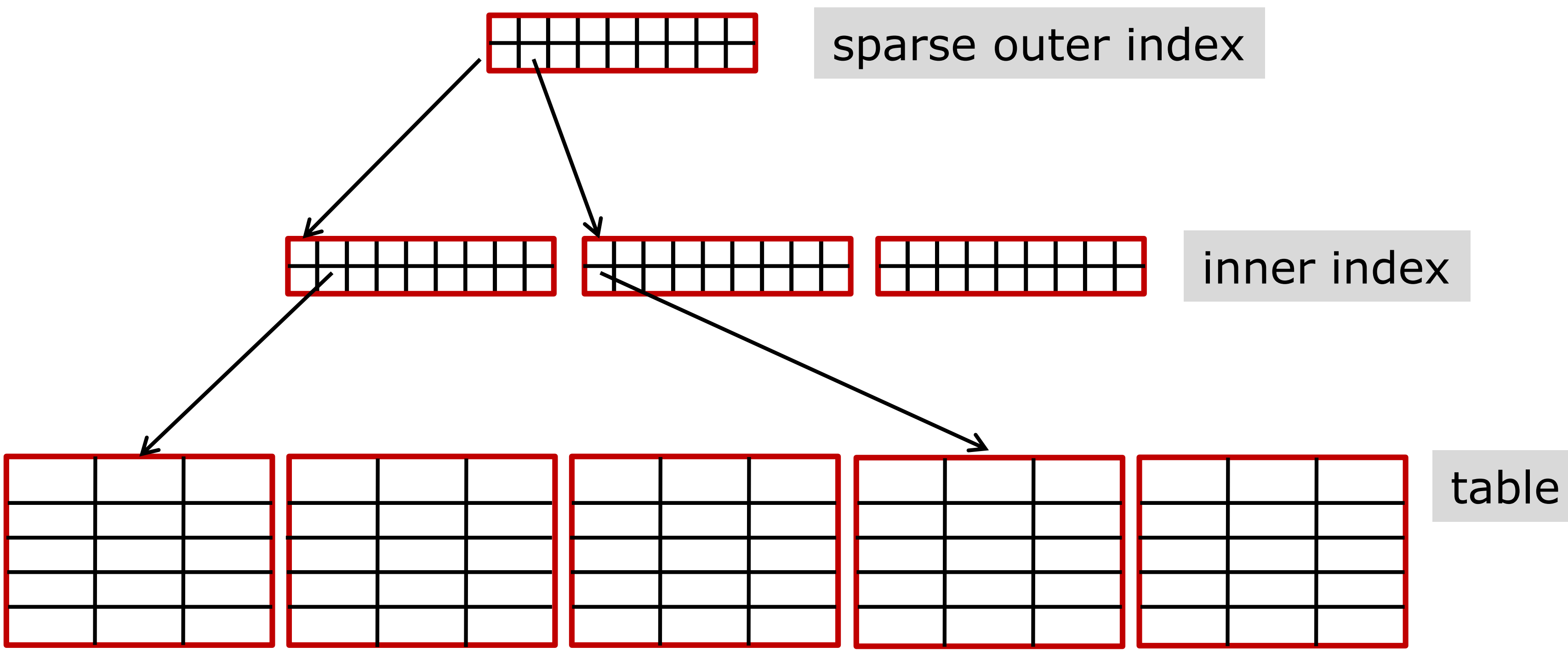
1. Use binary search to find biggest value less than search key in outer index.

2. Scan block in inner index for search key.

3. Go to block in table.

search time: ~ 10 ms

Draw it sideways



Now it's clearly a search tree

The nodes are the size of disk blocks.

If the table gets bigger, more tree levels can be used.

B+ trees

The problem with multi-level indexes:

- As records are inserted and deleted, the index structure needs to be reorganized

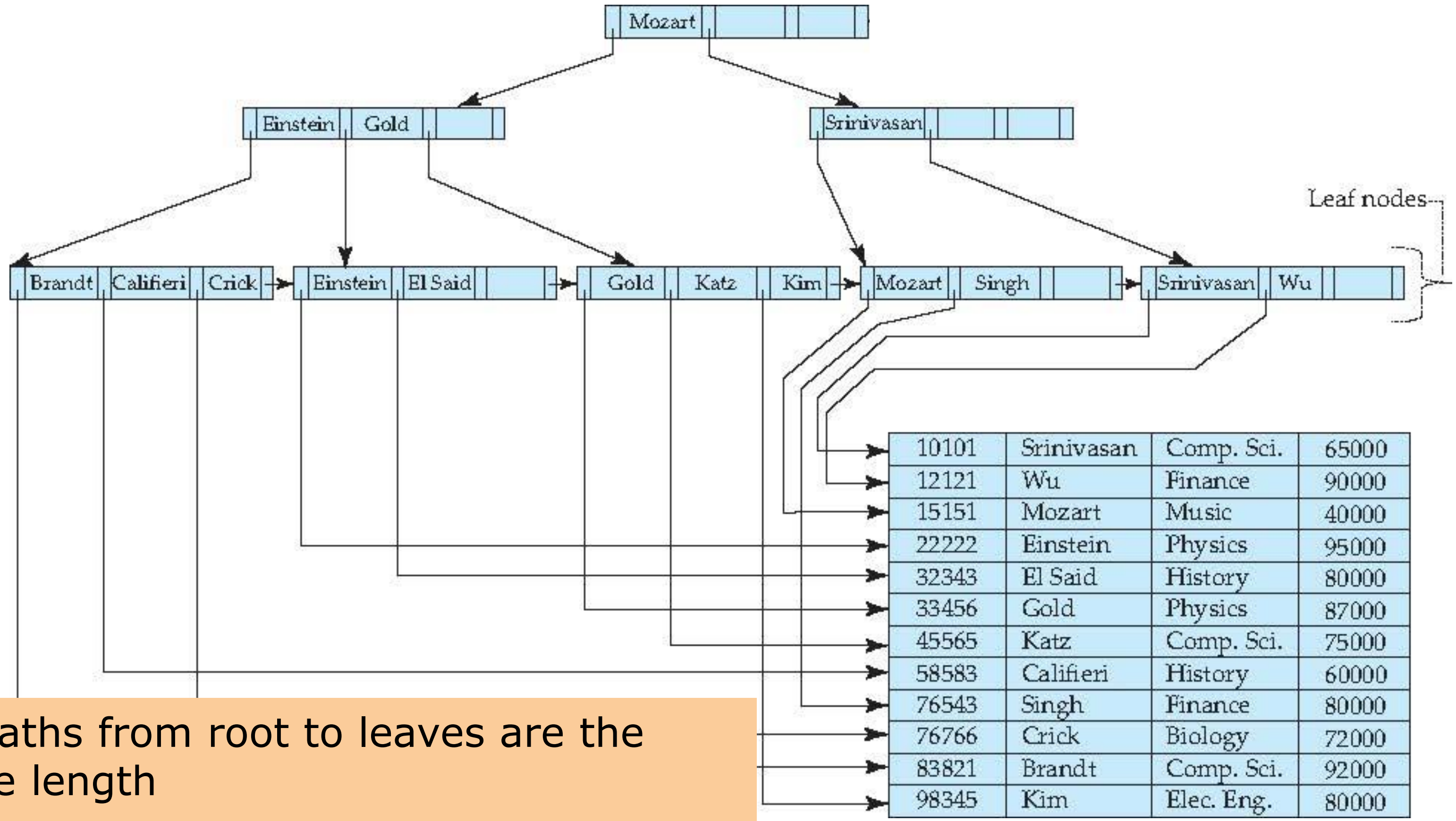
A B+ tree is similar to a multi-level index:

- nodes are the size of disk blocks
- search works from the top to the bottom

but unlike a multi-level index:

- B+ trees are “self-balancing”
- nodes can be partly empty

Features of B+ trees



1. All paths from root to leaves are the same length
2. B+ trees tend to be very "fat" – about 200 children per node

Cost of B+ tree search: roughly the time for 1 block read times the tree depth

(figure from Database System Concepts, Silbershatz, Korth, and Sudarshan)