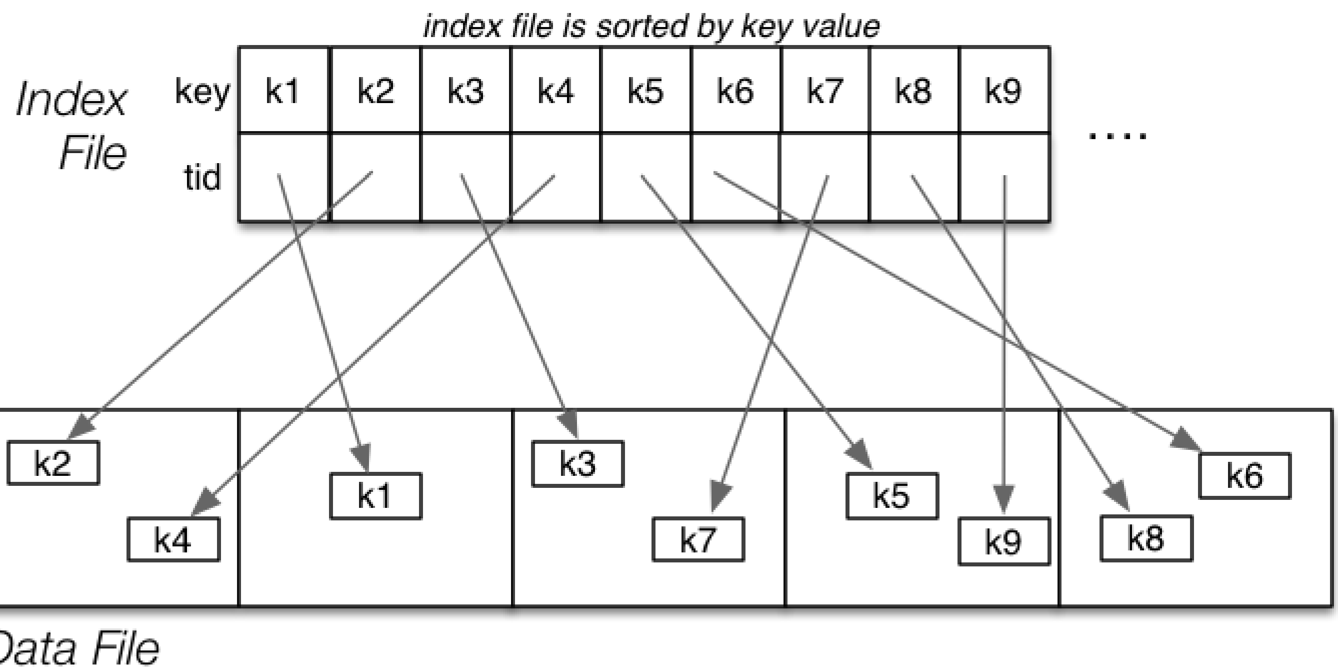
>>

Indexing

/\ >>

❖ Indexing

An index is a file of (keyVal,tupleID) pairs, e.g.



❖ Indexes

A 1-d index is based on the value of a single attribute *A*.

Some possible properties of *A*:

- may be used to sort data file (or may be sorted on some other field)
- values may be unique (or there may be multiple instances)

Taxonomy of index types, based on properties of index attribute:

primary	index on unique field, may be sorted on *A*
clustering	index on non-unique field, file sorted on *A*
secondary	file *not* sorted on *A*

A given table may have indexes on several attributes.

<< ^ >>

❖ Indexes (cont)

Indexes themselves may be structured in several ways:

dense	every tuple is referenced by an entry in the index file
sparse	only some tuples are referenced by index file entries
single-level	tuples are accessed directly from the index file
multi-level	may need to access several index pages to reach tuple

Index file has total i pages (where typically $i \ll b$)

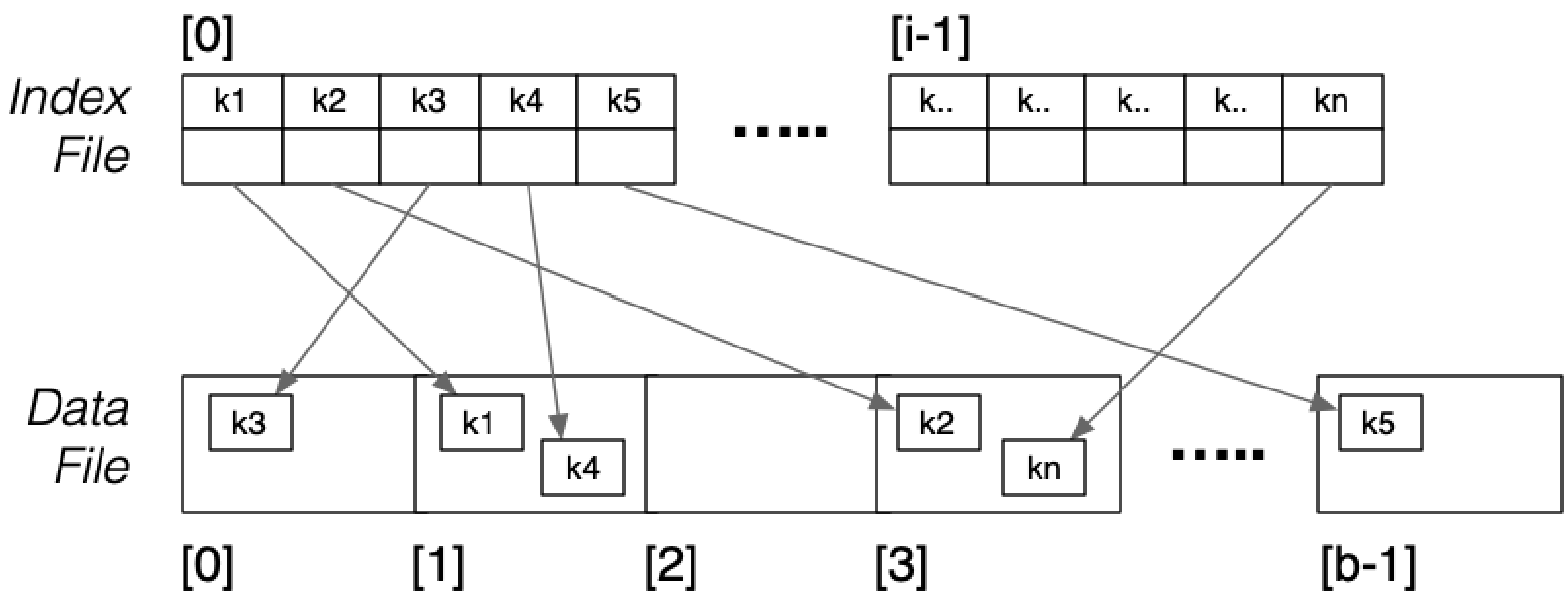
Index file has page capacity c_i (where typically $c_i \gg c$)

Dense index: $i = ceil(r/c_i)$ Sparse index: $i = ceil(b/c_i)$

<< ^ >>

❖ Dense Primary Index

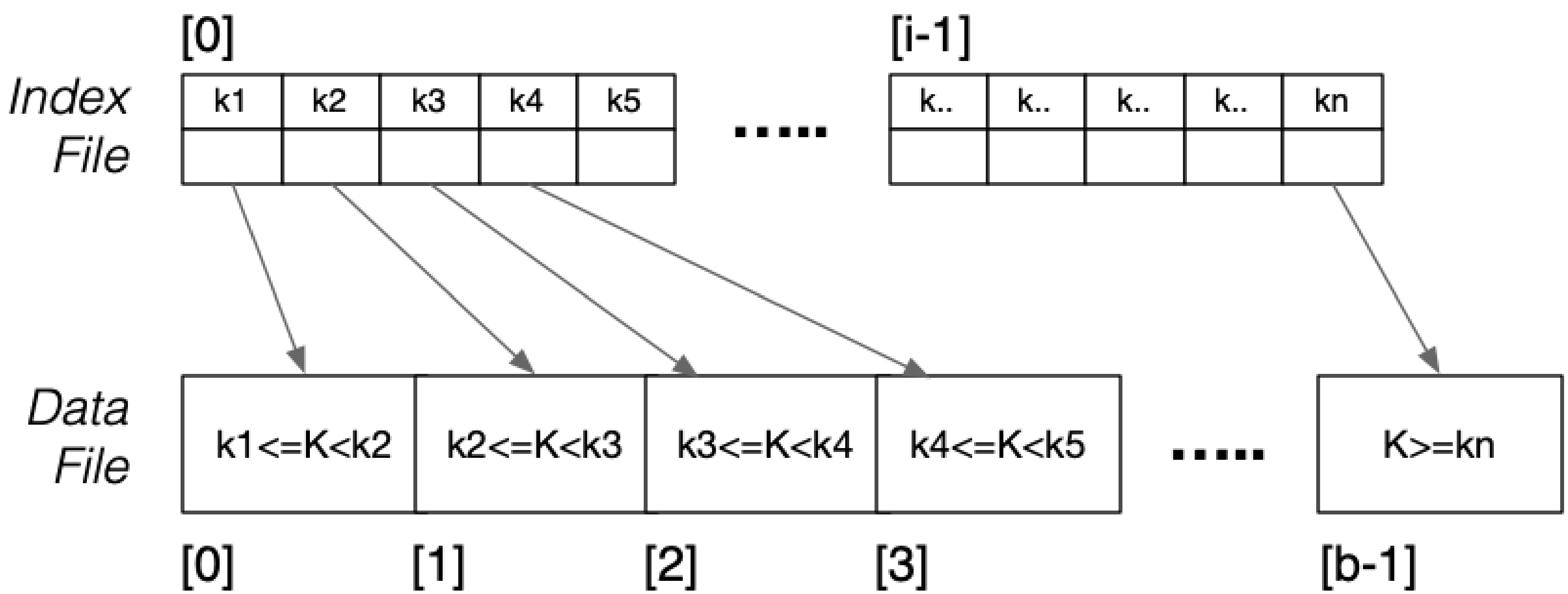
Data file unsorted; one index entry for each tuple



<< ∧ >>

❖ Sparse Primary Index

Data file sorted; one index entry for each page



<< ^ >>

❖ Selection with Primary Index

For *one* queries:

```
ix = binary search index for entry with key K
if nothing found { return NotFound }
b = getPage(pageOf(ix.tid))
t = getTuple(b,offsetOf(ix.tid))
    -- may require reading overflow pages
return t
```

Worst case: read $log_2 i$ index pages + read $1+Ov$ data pages.

Thus, $Cost_{one,prim}$ = $log_2 i + 1 + Ov$

Assume: index pages are same size as data pages ⇒ same reading cost

❖ Selection with Primary Index (cont)

For *range* queries on primary key:

- use index search to find lower bound
- read index sequentially until reach upper bound
- accumulate set of buckets to be examined
- examine each bucket in turn to check for matches

For *pmr* queries involving primary key:

- search as if performing *one* query.

For queries not involving primary key, index gives no help.

❖ Selection with Primary Index (cont)

Method for range queries (when data file is not sorted)

```
// e.g. select * from R where a between lo and hi
pages = {}   results = {}
ixPage = findIndexPage(R.ixf,lo)
while (ixTup = getNextIndexTuple(R.ixf)) {
   if (ixTup.key > hi) break;
   pages = pages ∪ pageOf(ixTup.tid)
}
foreach pid in pages {
   // scan data page plus ovflow chain
   while (buf = getPage(R.datf,pid)) {
      foreach tuple T in buf {
         if (lo<=T.a && T.a<=hi)
            results = results ∪ T
}  }  }
```

<< ∧ >>

❖ Insertion with Primary Index

Overview:

```
tid = insert tuple into page P at position p
find location for new entry in index file
insert new index entry (k,tid) into index file
```

Problem: order of index entries must be maintained

- need to avoid overflow pages in index (but see later)
- so, reorganise index file by moving entries up

Reorganisation requires, on average, read/write half of index file:

$$Cost_{insert,prim} = (\log_2 i)_r + i/2.(1_r+1_w) + (1+Ov)_r + (1+\delta)_w$$

<< ∧ >>

❖ Deletion with Primary Index

Overview:

```
find tuple using index
mark tuple as deleted
delete index entry for tuple
```

If we delete index entries by marking ...

- $Cost_{delete,prim} = (\log_2 i)_r + (1 + Ov)_r + 1_w + 1_w$

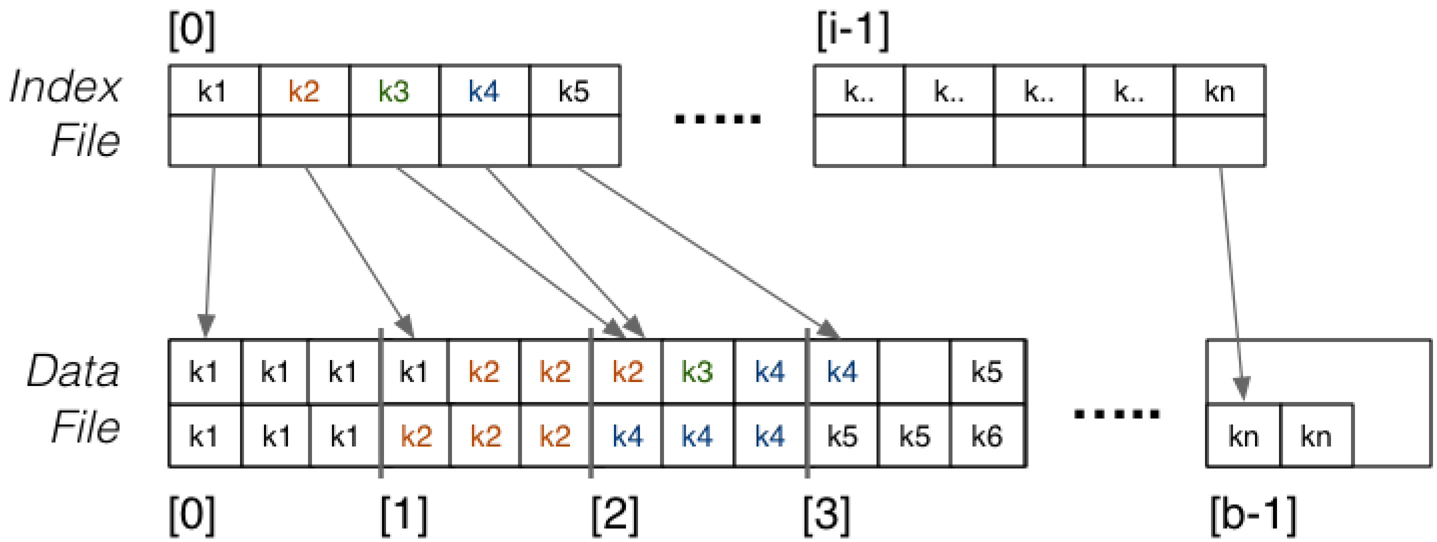
If we delete index entry by index file reorganisation ...

- $Cost_{delete,prim} = (\log_2 i)_r + (1 + Ov)_r + i/2.(1_r+1_w) + 1_w$

<< ^ >>

❖ Clustering Index

Data file sorted; one index entry for each key value



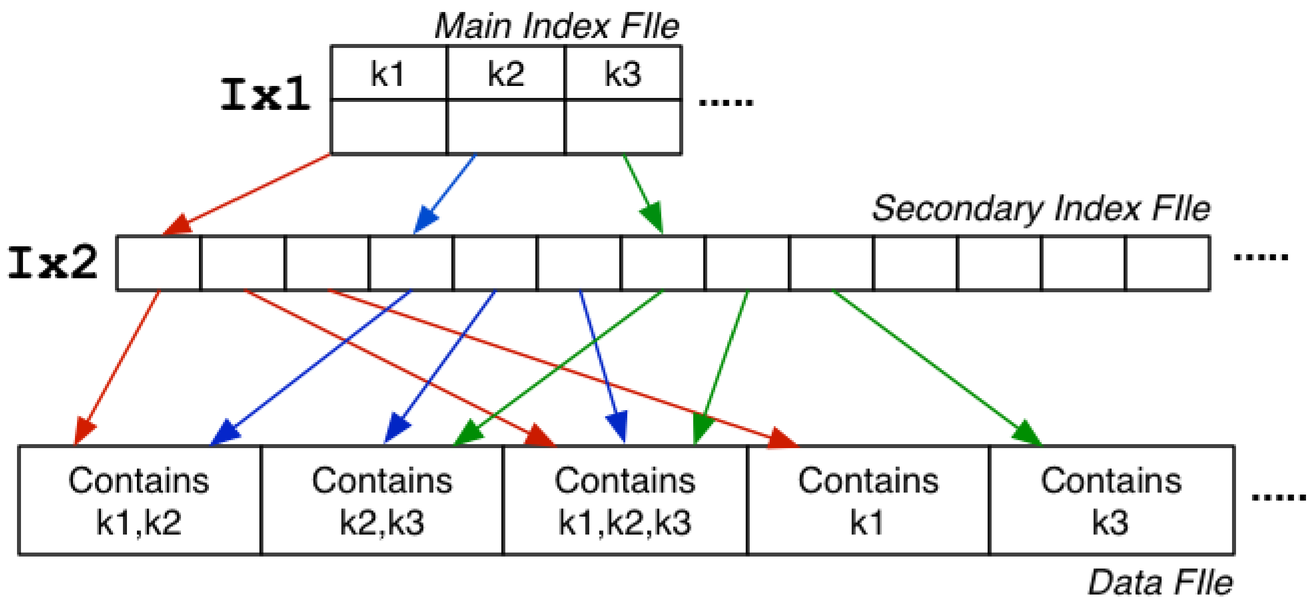
Cost penalty: maintaining both index and data file as sorted

(Note: can't mark index entry for value X until all X tuples are deleted)

<< ^ >>

❖ Secondary Index

Data file not sorted; want one index entry for each key value



$$Cost_{pmr} = (log_2 i_{ix1} + a_{ix2} + b_q.(1 + Ov))$$

❖ Multi-level Indexes

Secondary Index used two index files to speed up search

- by keeping the initial index search relatively quick
- **`Ix1`** small (depends on number of unique key values)
- **`Ix2`** larger (depends on amount of repetition of keys)
- typically, $b_{Ix1} \ll b_{Ix2} \ll b$

Could improve further by

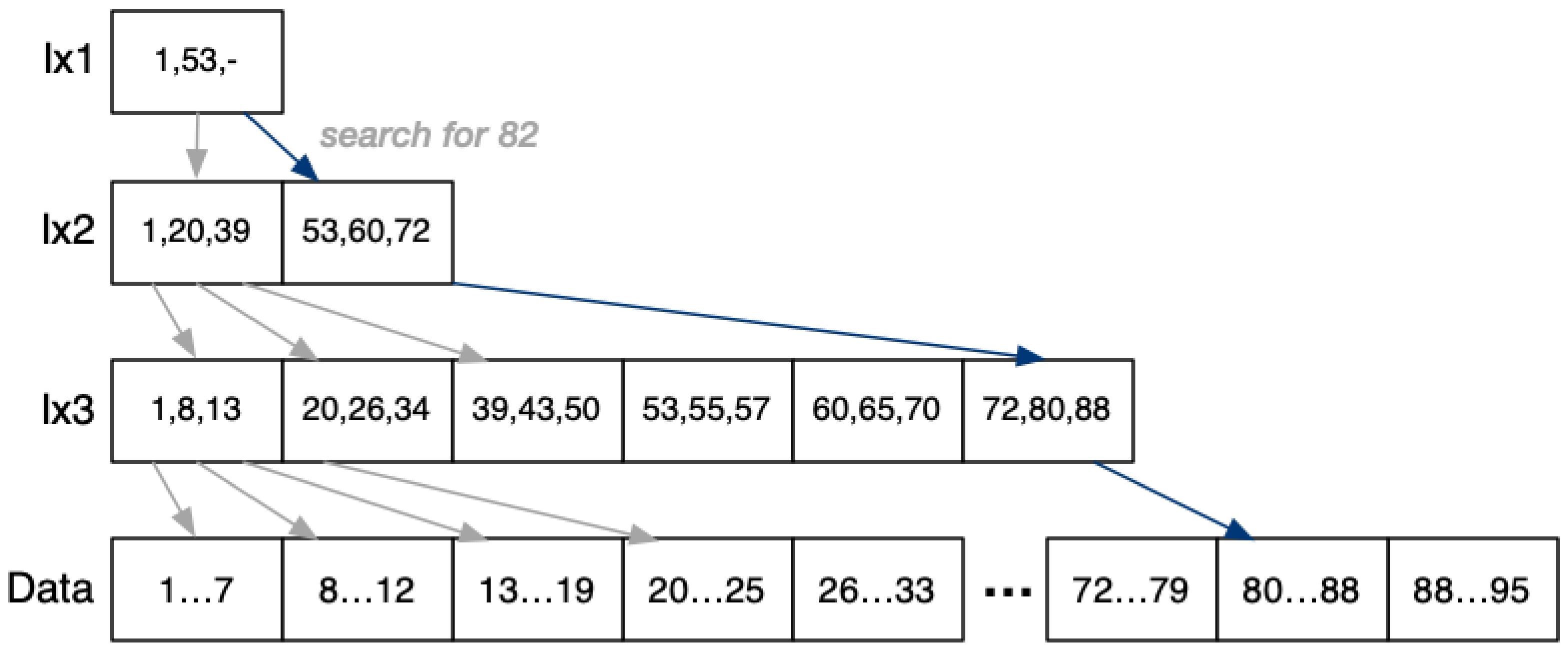
- making **`Ix1`** sparse, since **`Ix2`** is guaranteed to be ordered
- in this case, $b_{Ix1} = ceil(b_{Ix2} / c_i)$
- if **`Ix1`** becomes too large, add **`Ix3`** and make **`Ix2`** sparse
- if data file ordered on key, could make **`Ix3`** sparse

Ultimately, reduce top-level of index hierarchy to one page.

<< ^ >>

❖ Multi-level Indexes (cont)

Example data file with three-levels of index:



Assume: not primary key, $c = 20$, $c_i = 3$

In reality, more likely $c = 100$, $c_i = 1000$

COMP9315 21T1 ♢ Indexing ♢ [14/15]

<< ^

❖ Select with Multi-level Index

For *one* query on indexed key field:

```
xpid = top level index page
for level = 1 to d {
    read index entry xpid
    search index page for J'th entry
        where index[J].key <= K < index[J+1].key
    if (J == -1) { return NotFound }
    xpid = index[J].page
}
pid = xpid  // pid is data page index
search page pid and its overflow pages
```

$Cost_{one,mli} = (d + 1 + Ov)_r$

(Note that $d = ceil(log_{c_i} r)$ and c_i is large because index entries are small)

Produced: 13 Mar 2021