# COMP163

Database Management Systems

## Static Indexes
## Sections 18.1-18.2

# Access Structures

- Access structures are persistent data structures added to get faster results for specific classes of queries
- static indexes
  - must be rebuild to accommodate changes in the data
  - simple structures
- dynamic indexes
  - reconfigure as the data changes
  - more complex structure
  - variations on balanced search trees

# Single-level Primary Indexes

- If a file is ordered by the primary key, we can construct an ordered index consisting of keys and pointers to blocks
  - the index key is the key of the first record in a block

- Lookup algorithm:
  - binary search on index, then read data block
  - cost with index: $\log_2(r_i) + 1$ $r_i$ = # index blocks
  - cost without index: $\log_2(r)$ r = # data blocks

# Index Blocking Factors

- Access structures also need to be blocked
  - records are now index nodes

- *Blocking factor* determines how many records can fit in a block
  - $bfr_i = floor(B/R_i)$ where B = block size and $R_i$ = node size

- Blocks required to store index:
  - $b_i = ceil(r_i / bfr_i)$ where $r_i$ = number of nodes

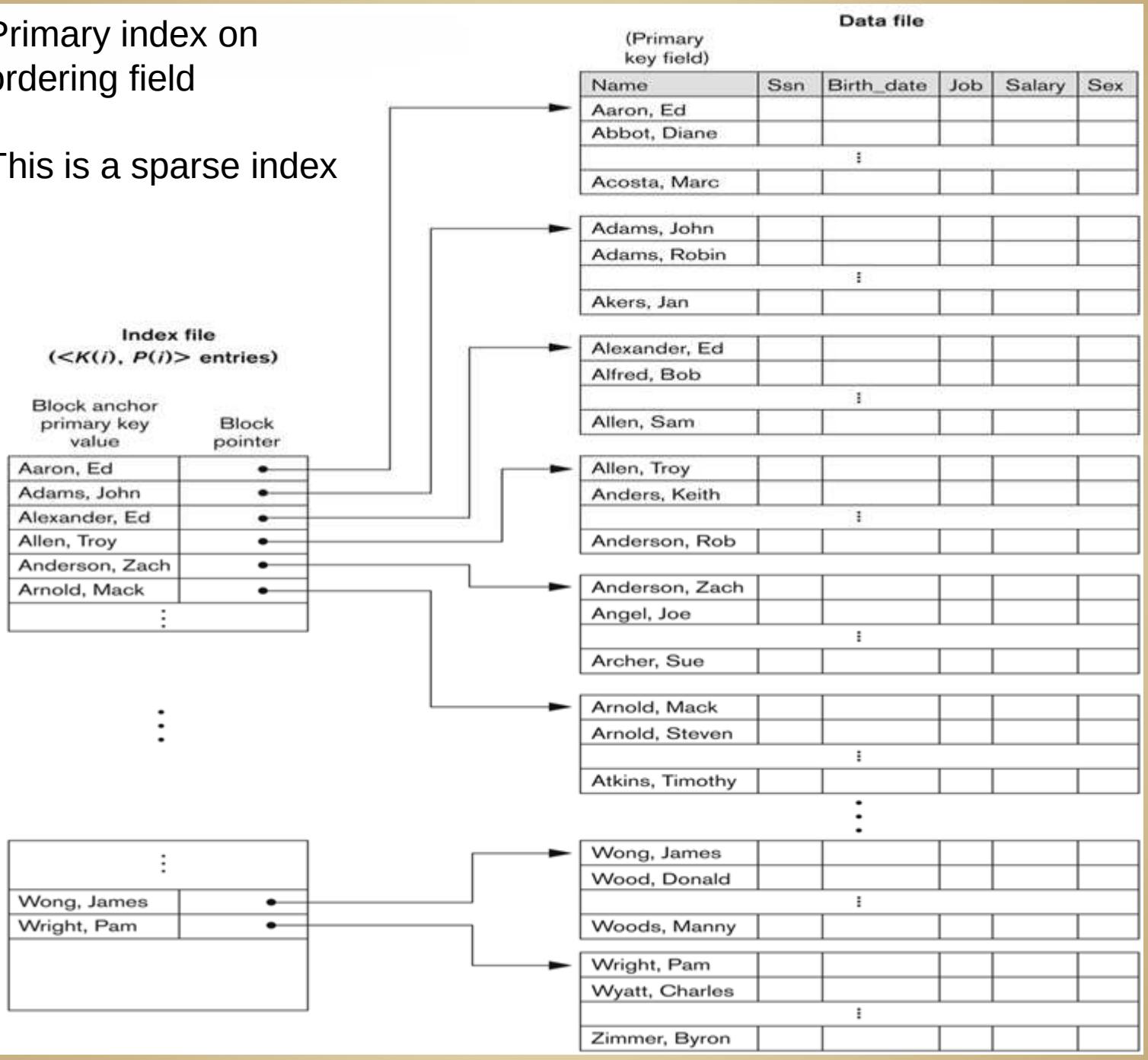
# Index Classifications

- **dense index:** index entry for every field value (and hence every record) in the data file.
- **sparse (nondense) index:** index entries for only some of the field values

- **primary index:** defined for a **ordering key field** in the data records.
- **secondary index:** non-key field or unordered file

Primary index on ordering field

This is a sparse index



# 1-level Primary Index Example

| | |
|---|---|
| # records | r = 200,000 records |
| record size | R = 200 bytes |
| block size | B = 1024 bytes |
| key size | V = 9 bytes |
| block pointer size | P = 6 bytes |

bfr = floor(B/R) = 5 records/block
b = ceil(r/bfr) = 40,000 blocks

$R_i$ = V + P = 15 bytes
$r_i$ = b = 40,000 index entries (sparse index)
$bfr_i$ = floor(B/$R_i$) = 68 index entries / block
$b_i$ = ceil($r_i$/$bfr_i$) = 589

binary search cost
with index:
$\log_2(b_i) + 1 = 11$
without index:
$\log_2(b) = 16$

# 1-level Primary Index Example

| | |
|---|---|
| # records | r = 2,000,000 records |
| record size | R = 30 bytes |
| block size | B = 512 bytes |
| key size | V = 9 bytes |
| block pointer size | P = 6 bytes |

bfr = floor(B/R) = 17 records/block
b = ceil(r/bfr) = 117,648 blocks

$R_i$ = V + P = 15 bytes
$r_i$ = b = 117,648 index entries (sparse index)
$bfr_i$ = floor(B/$R_i$) = 68 index entries / block
$b_i$ = ceil($r_i$/$bfr_i$) = 1731

binary search cost
with index:
$\log_2(b_i) + 1 = 12$
without index:
$\log_2(b) = 17$

# Clustering Index

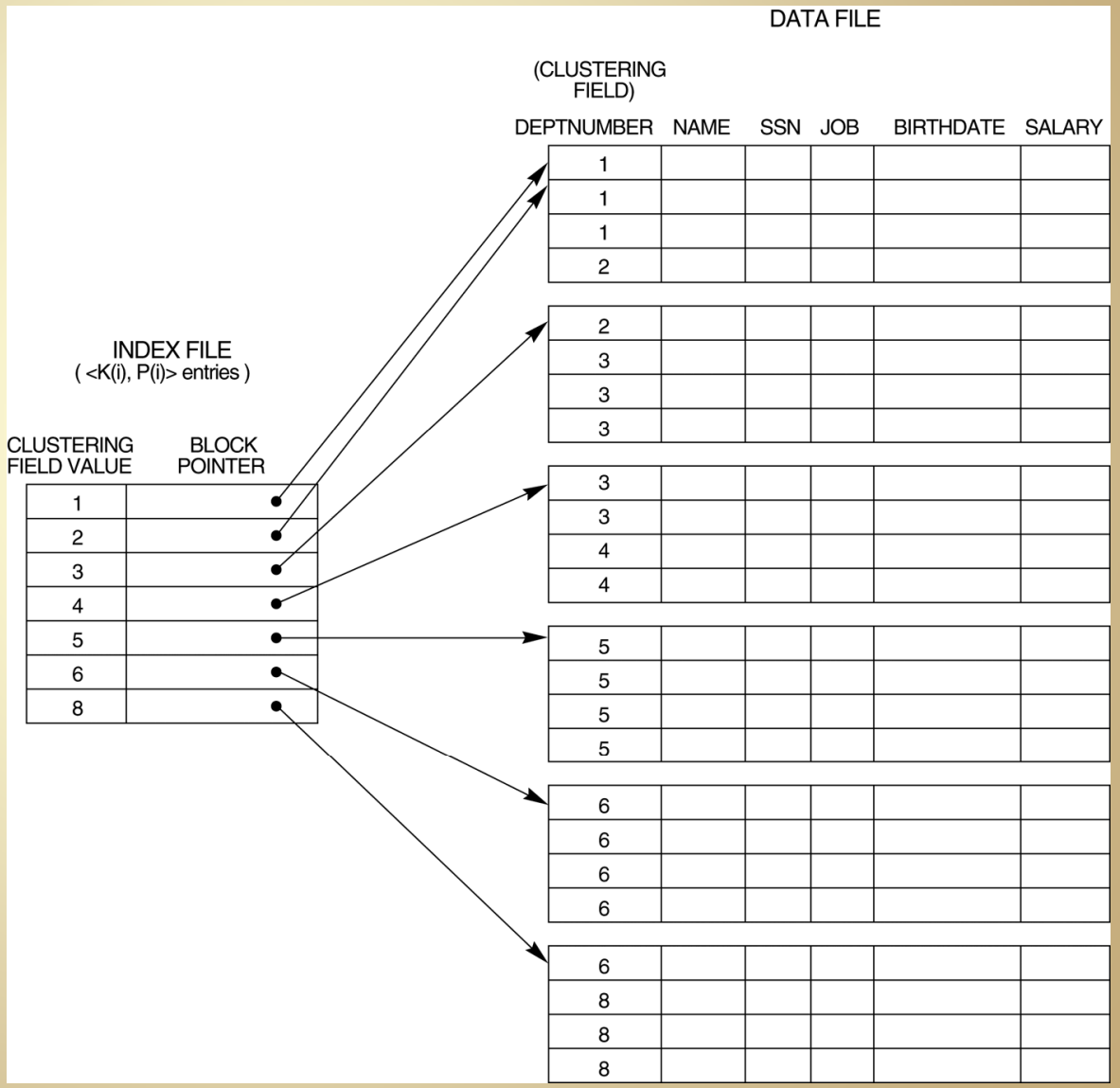
- Defined on an **ordered** data file
- The data file is ordered on a *non-key field* – there may be multiple records with the same index value.
- one index entry *for each distinct value* of the field
  - the index entry points to the first data block that contains records with that field value.
- This is a *sparse* index

# A Clustering Index Example

Data file is ordered on non-key field DEPTNUMBER

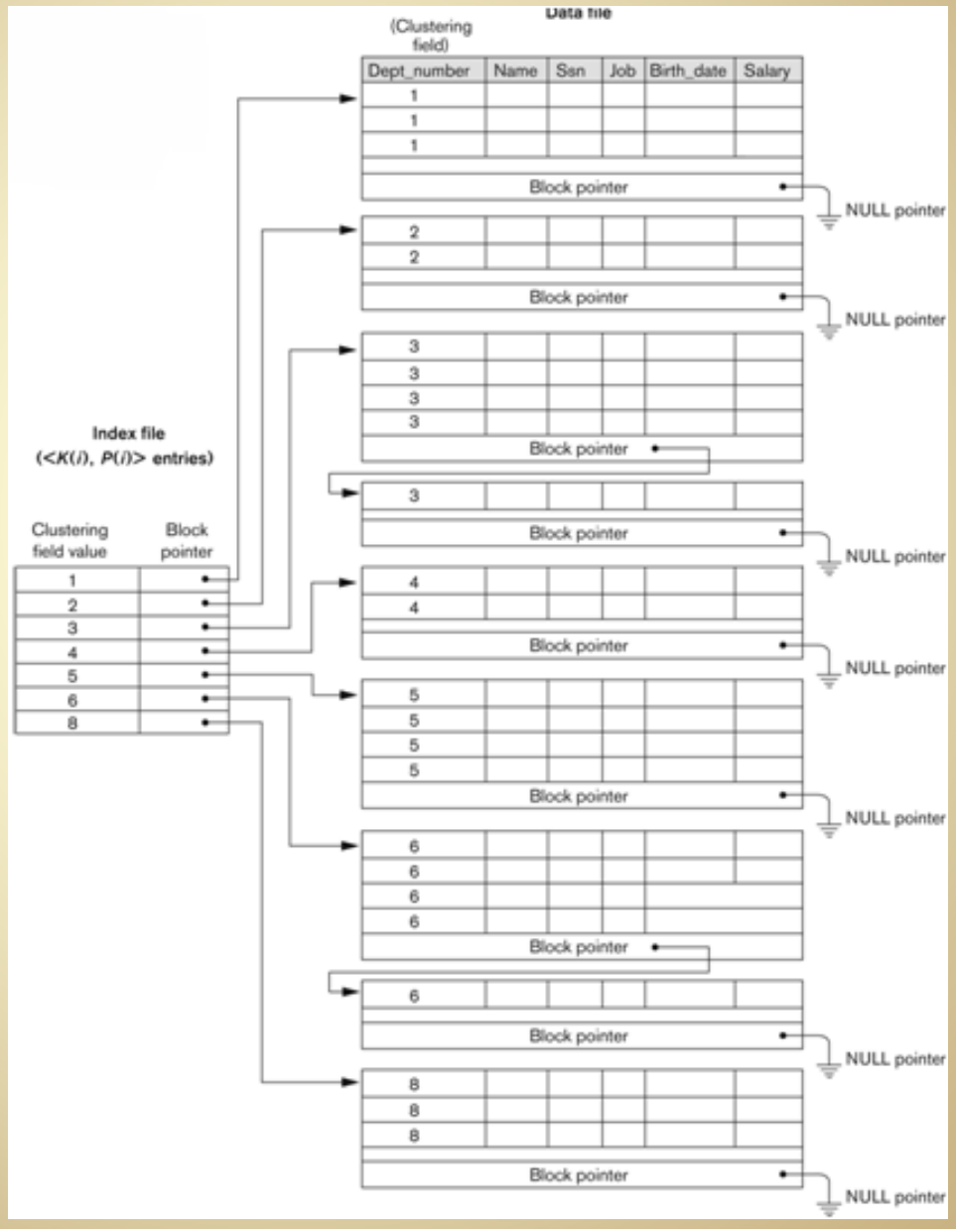
Index holds pointer to first block that contains each index value

This is an example of a clustering index



# Another Clustering Index Example

Clustering index with a separate block cluster for each value of clustering field

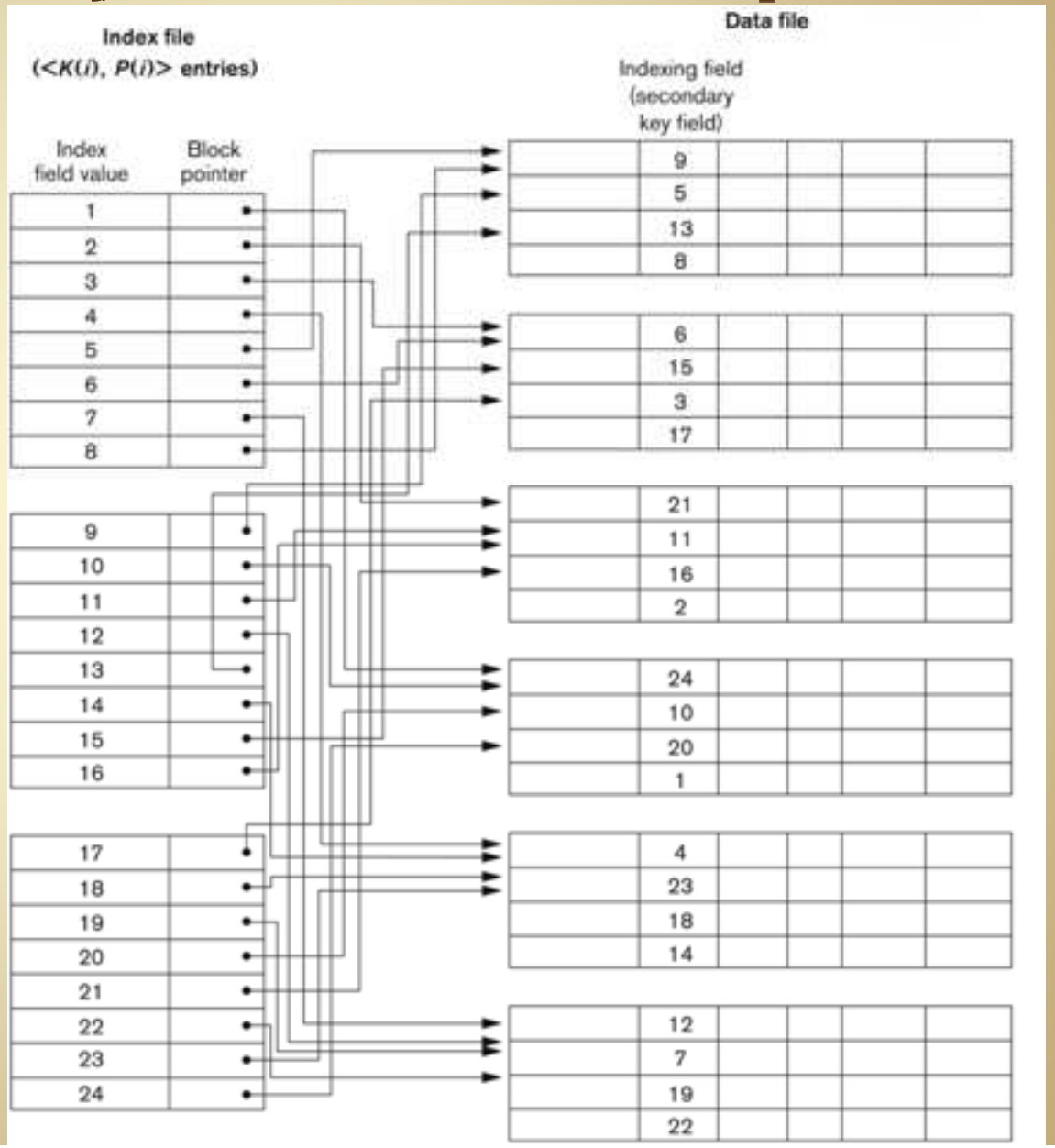


# Secondary Indexes

- A secondary index provides a secondary means of accessing a file for which some primary access already exists
  - may be on candidate key (unique value in every record)
  - may be on non-key with duplicate values.
- The index is an ordered file
- There can be *many* secondary indexes (and hence, indexing fields) for the same file.
- Includes one entry *for each record* in the data file
  - a *dense index*

# Secondary Index Example

dense secondary index on a nonordering key field

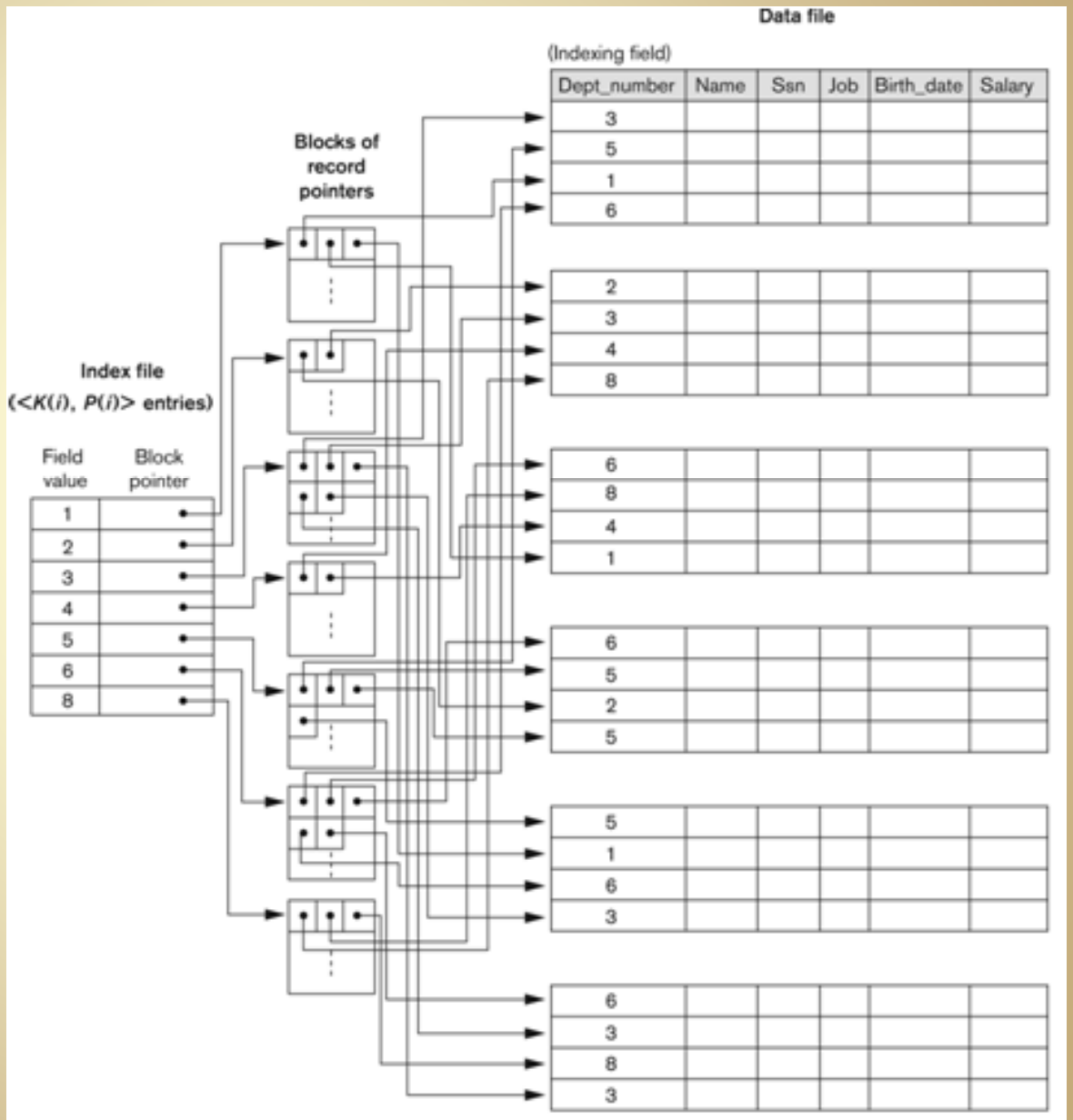


# Secondary Index Example

secondary index on a nonkey field

one level of indirection allows fixed length index records with unique field values

note that this example uses record pointers into data file



# Properties of Index Types

| Type of Index | Number of (First-level) Index Entries | Dense or Nondense | Block Anchoring on the Data File |
|---|---|---|---|
| Primary | Number of blocks in data file | Nondense | Yes |
| Clustering | Number of distinct index field values | Nondense | Yes/no[a] |
| Secondary (key) | Number of records in data file | Dense | No |
| Secondary (nonkey) | Number of records[b] or Number of distinct index field values[c] | Dense or Nondense | No |

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

# Multi-level Indexes

# Multi-level Primary Indexes

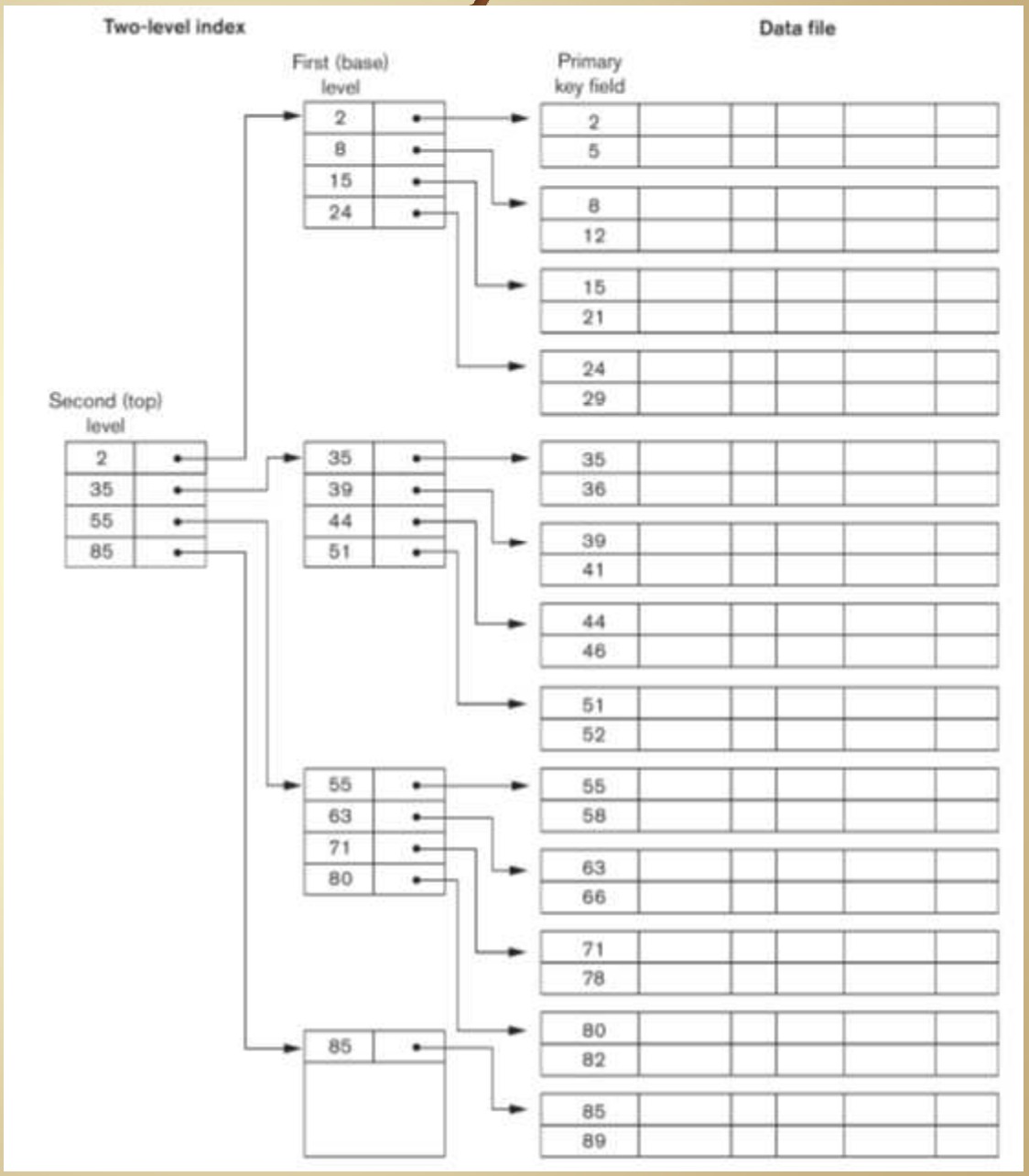
- multi-level index:
  a tree built by indexing the indexes

- tree arity (fan-out) is the index blocking factor
- tree height is $\log_{fo}(b)$

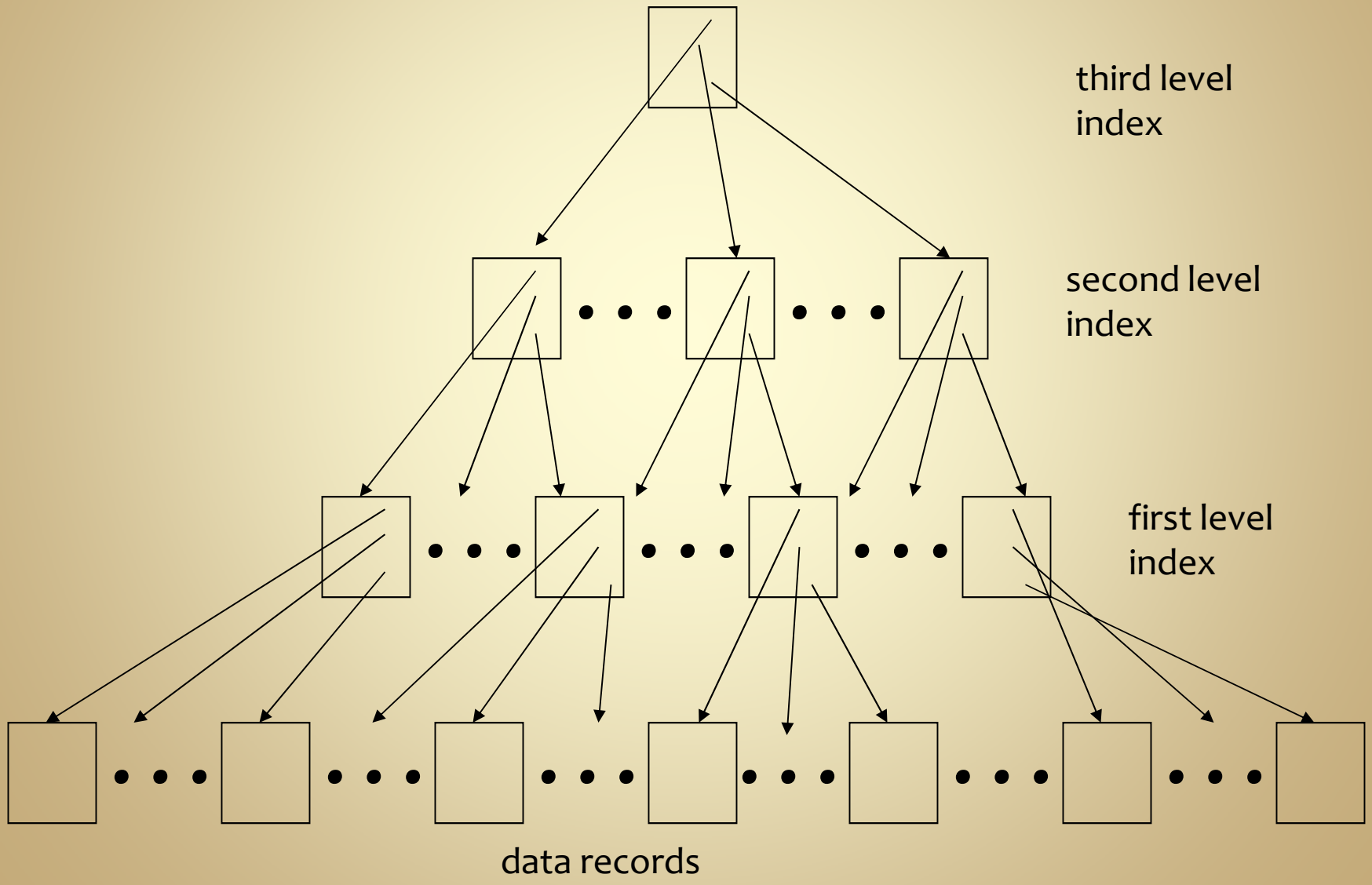
- Example: b = 20,000 data blocks, $bfr_i = fo = 60$
  - first level index: $b_1 = \lceil 20000/60 \rceil = 334$ blocks
  - second level index: $b_2 = \lceil 334/60 \rceil = 6$ blocks
  - third level index: $b_3 = \lceil 6/60 \rceil = 1$ block
  - lookup cost:
    one index block read per level + one data block read
    = 4 (random) block reads

# Two Level Primary Index

similar to Indexed Sequential Access Method (ISAM)



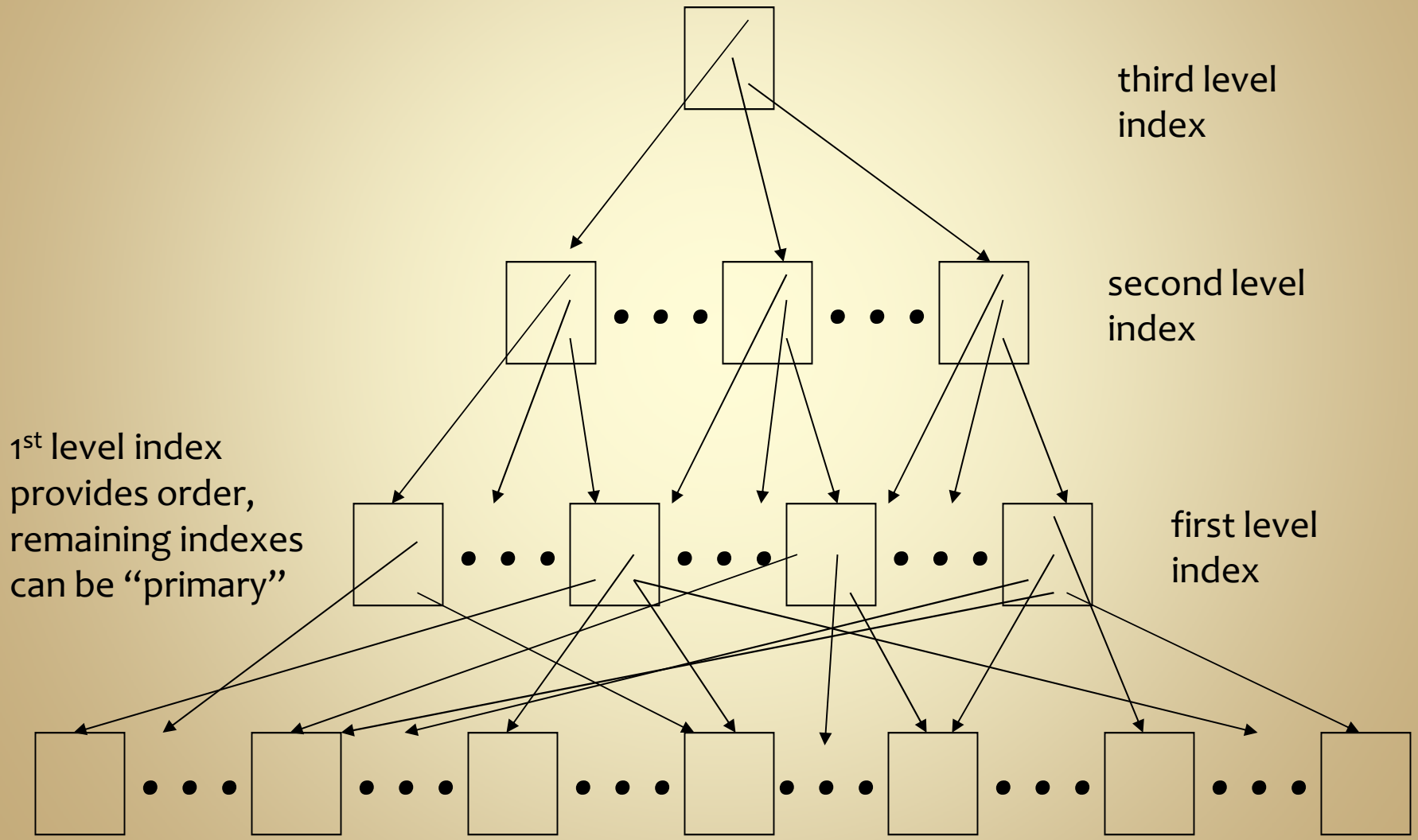
# Multi-level Primary Index



# Multilevel Secondary Indexes

- Since the first level index is ordered,
  it can be indexed in the same manner
  as primary key indexes

# Multi-level Secondary Index



# Static vs. Dynamic Indexes

- Indexes seen so far are static
  - built from a fixed data file
- Updates to data file require rebuilding the index
  - could be very expensive
- Solution: Dynamic Indexes
  - B-trees and $B^+$-trees can be adjusted as the data file changes

# powers of 2

| | |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 |
| 11 | 2048 |
| 12 | 4096 |
| 13 | 8192 |
| 14 | 16384 |
| 15 | 32768 |
| 16 | 65536 |
| 17 | 131072 |
| 18 | 262144 |
| 19 | 524288 |
| 20 | 1048576 |
| 21 | 2097152 |