WEEK 3

ADDING AN INDEX TO SPEED UP THE FINDING OF RECORDS.

STUDENT OBJECTIVES

- Upon completion of this video, you should be able to:
  - Define the following types of indices: Primary, Clustered, Secondary, Dense and Sparse
  - Determine at which times you would use each of the types of indices above
  - Differentiate between a single level index and a multi level index
  - Given a number of records, record size and block size, figure out the average number of searches needed to find a record and the worst case scenario for searching for a given record that has a single level index associated with it.

WHAT ELSE COULD WE DO TO SPEED UP SEARCHES?

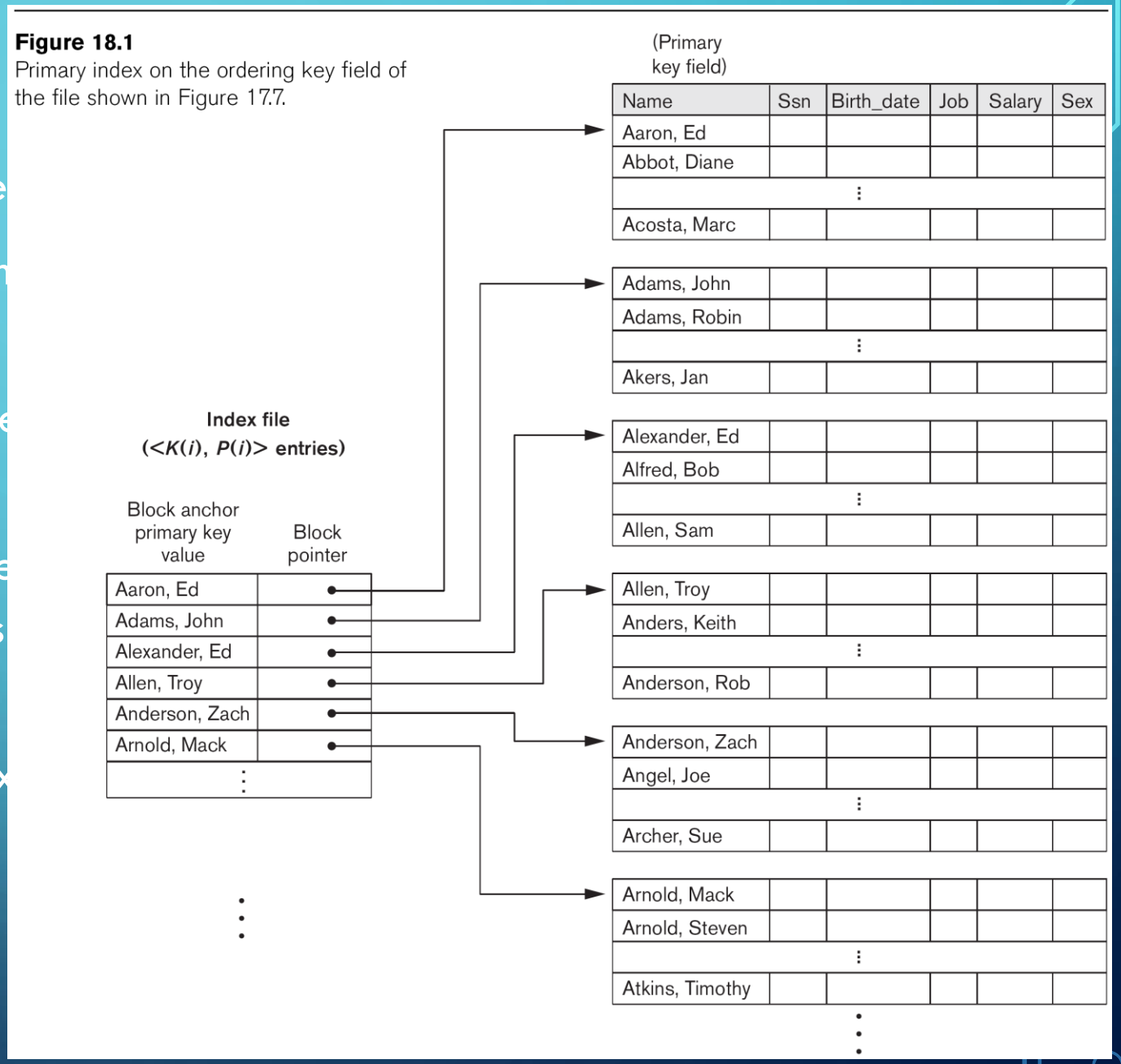
- **QUESTION: if you are looking for something in a large PAPER text book, what do you do?**
- ANSWER: use the index!

INDEX

- More versatile than hashing (can use on a range of values or part of a key)
- Records are stored in one file and the index(es) is stored in another file, stored in the disk.
- Load up the index first (hopefully the index can fit all in one block and can load the entire index into Main Memory).
- We then search the index file for the field (attribute) we are looking for and it tells us where to find the block that holds the complete item (record) associated with that field
- Types of Single-level Ordered Indexes
  - Primary
  - Clustered
  - Secondary

INDEX

- More versatile than hashing (can use
- Records are stored in one file and th disk.
- Load up the index first (hopefully the entire index into Main Memory).
- We then search the index file for the us where to find the block that holds field
- Types of Single-level Ordered Index
  - Primary
  - Clustered
  - Secondary



WHAT DOES THE INDEX LOOK LIKE?

- Usually of the form:

  <field value , pointer to record>

  e.g. 250012345, pointer to block 45

- Sample Index File:

250012345, block 45
250012347, block 23
250012350, block 45

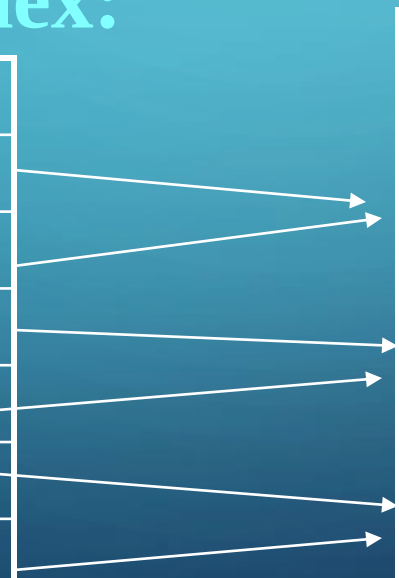
250999998, block 71

MORE TERMINOLOGY

- **Dense** index has an index entry for every search key value
- **Sparse** (non-dense) index has index entry for only some of the search values

Dense Index:

Index File	
SA9	1
SG5	1
SG14	2
SG37	2
SL21	3
SL41	3



Data File	Page (Block)
All of SA9 Record	1
All of SG5 Record	
All of SG14 Record	2
All of SG37 Record	
All of SL21 Record	3
All of SL41 Record	

Sparse Index:

Index File	
SA9	1
SG14	2
SL21	3

Data File	Page (Block)
All of SA9 Record	1
All of SG5 Record	
All of SG14 Record	2
All of SG37 Record	
All of SL21 Record	3
All of SL41 Record	

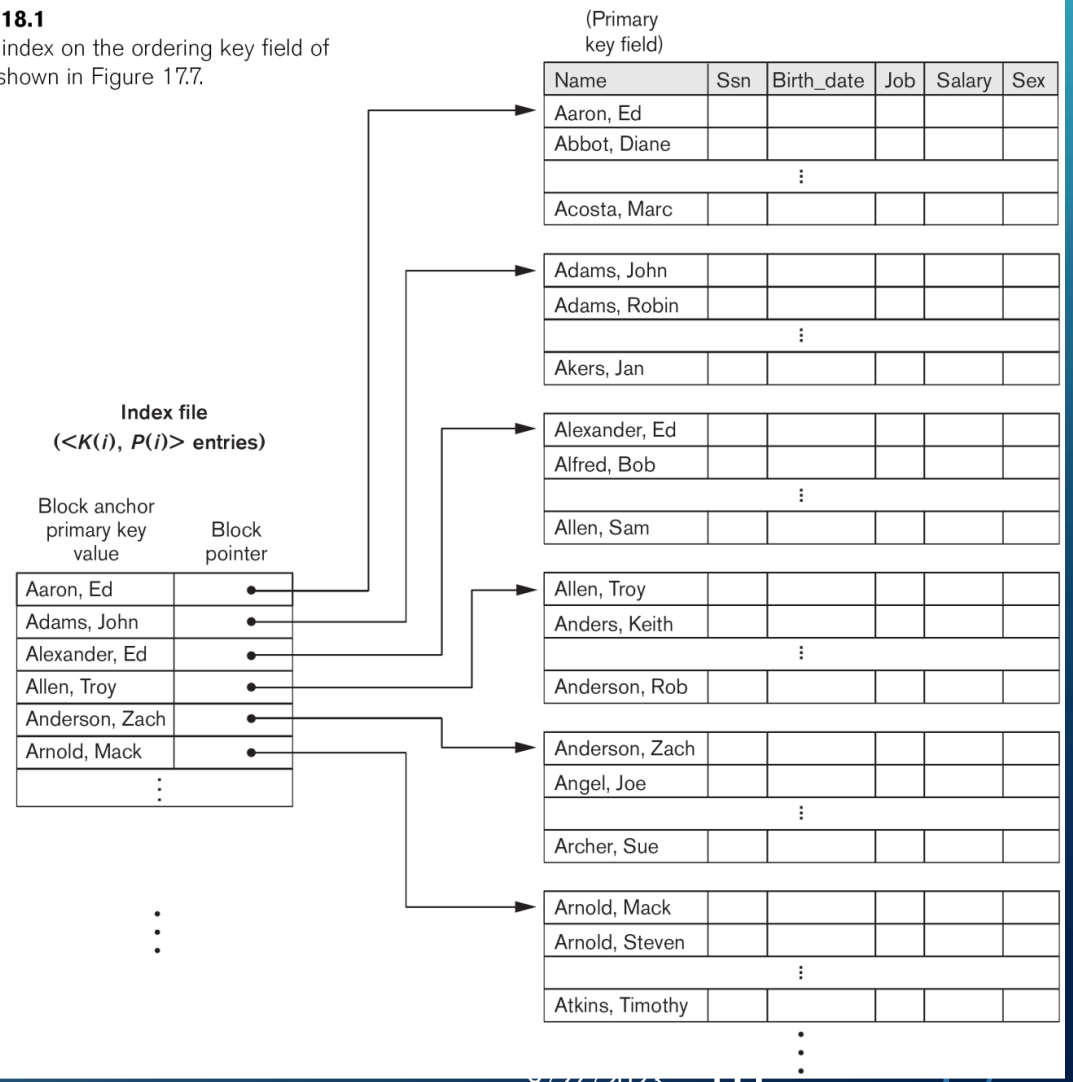
PRIMARY INDEX

- Always defined on an ordered data file
- Ordered on the ordering key field (but might not be the primary key of the table)
- Includes one index entry for EACH BLOCK in the data file (not for each record)

PRIMARY INDEX

- Always defined on an ordered data file
- Ordered on the ordering key field (but mi
  table)
- Includes one index entry for EACH BLOCK
  record)

Figure 18.1
Primary index on the ordering key field of the file shown in Figure 17.7.



CLUSTERING INDEX

- Defined on an ordered data file
- Ordered on a **non key** field, thus every ordering field might not be distinct
- Includes one index entry for each distinct value in the ordering field
- Insertions and deletions are straightforward if each cluster starts a new block

CLUSTERING INDEX

- Defined on an ordered data file
- Ordered on a **non key** field, thus every field might not be distinct
- Includes one index entry for each distinct va the ordering field
- Insertions and deletions are straightforwar cluster starts a new block

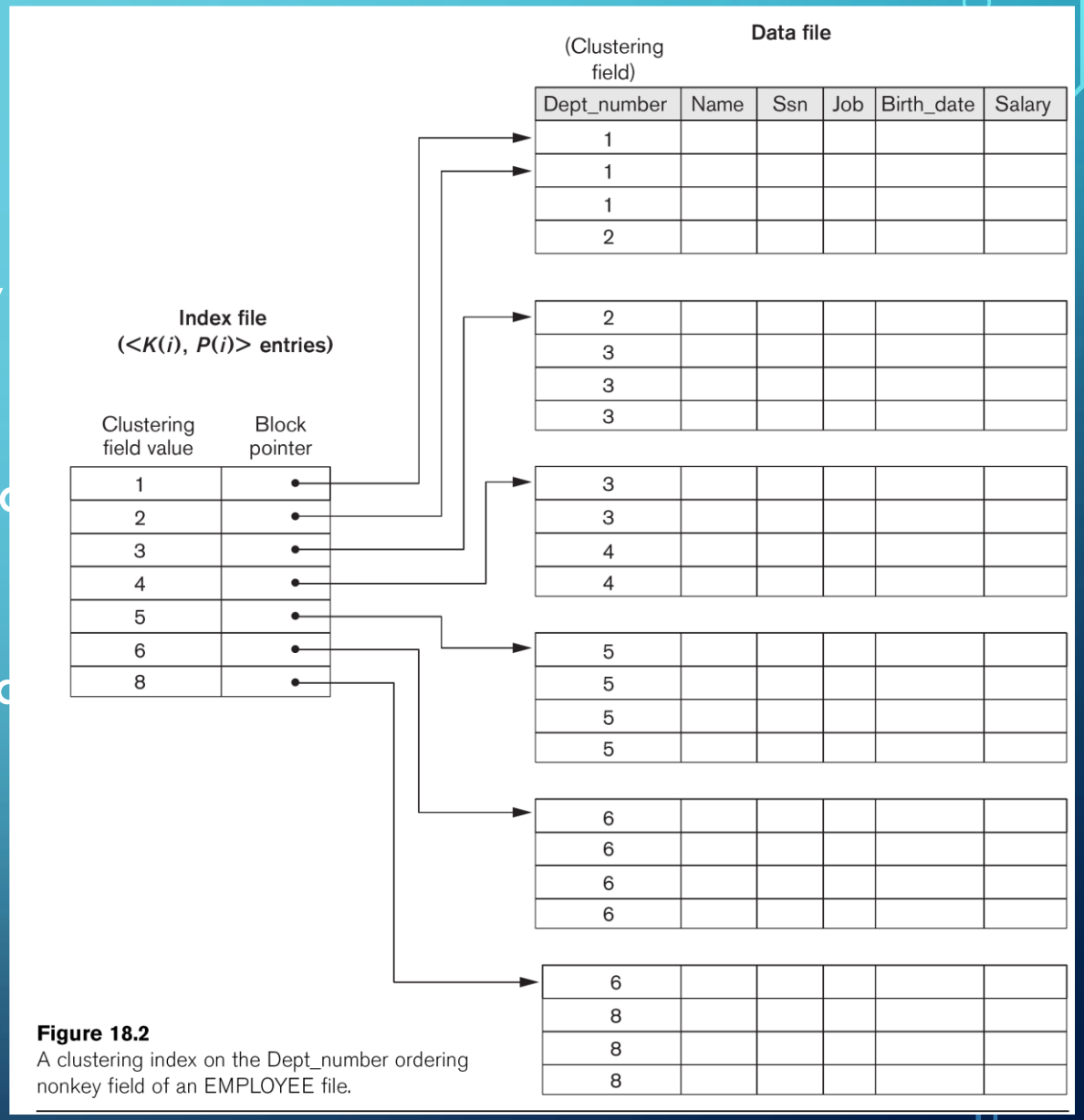
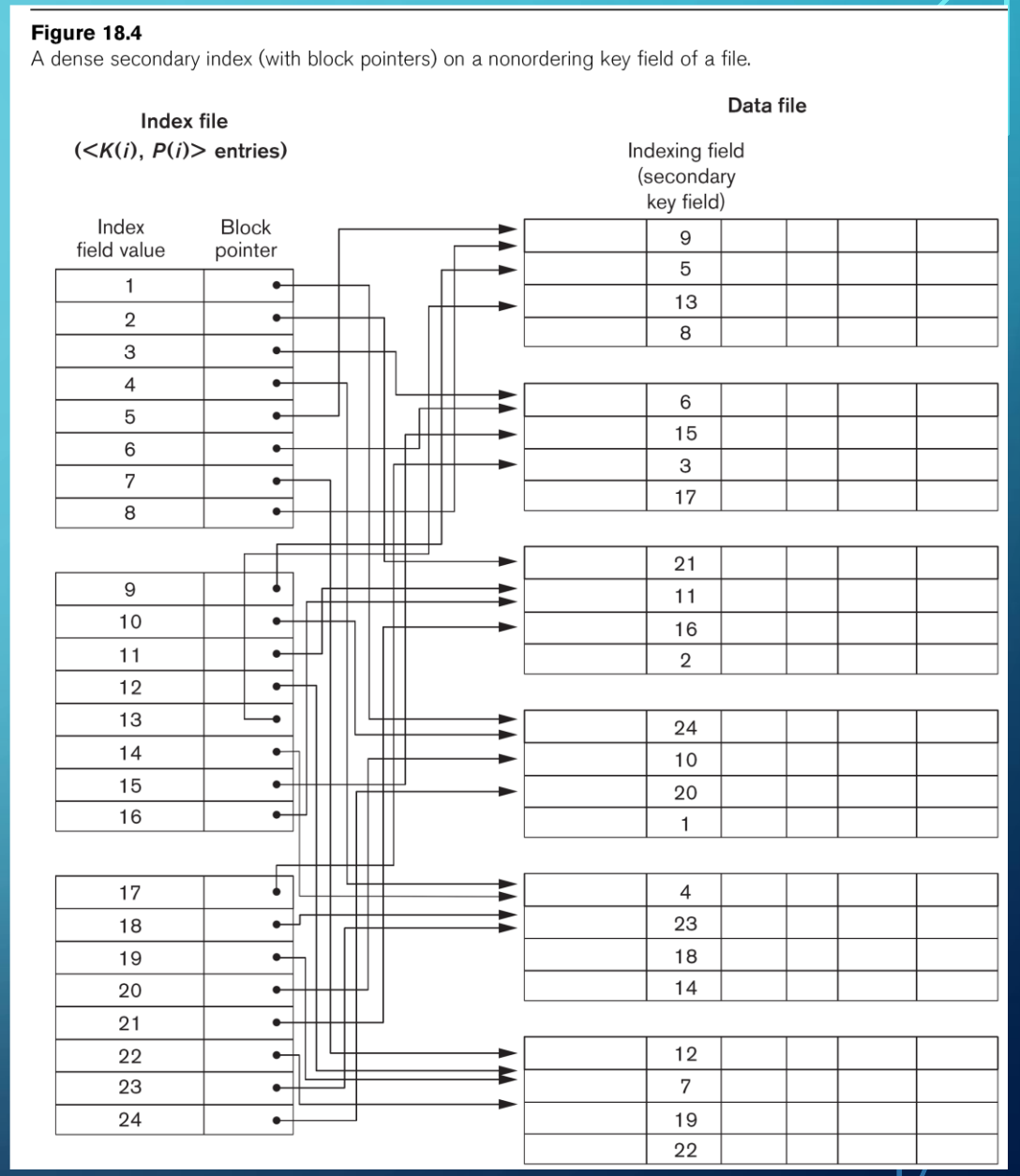


Figure 18.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

SECONDARY INDEX

- Provides a secondary access for a file which already has primary access
- Might be on a candidate key or any field
- Can have many secondary indexes
- Includes one entry for each record, hence it is DENSE



SECONDARY INDEX

- Provides a secondary access for a file which already has primary access
- Might be on a candidate key or any fi
- Can have many secondary indexes
- Includes one entry for each record, her is DENSE

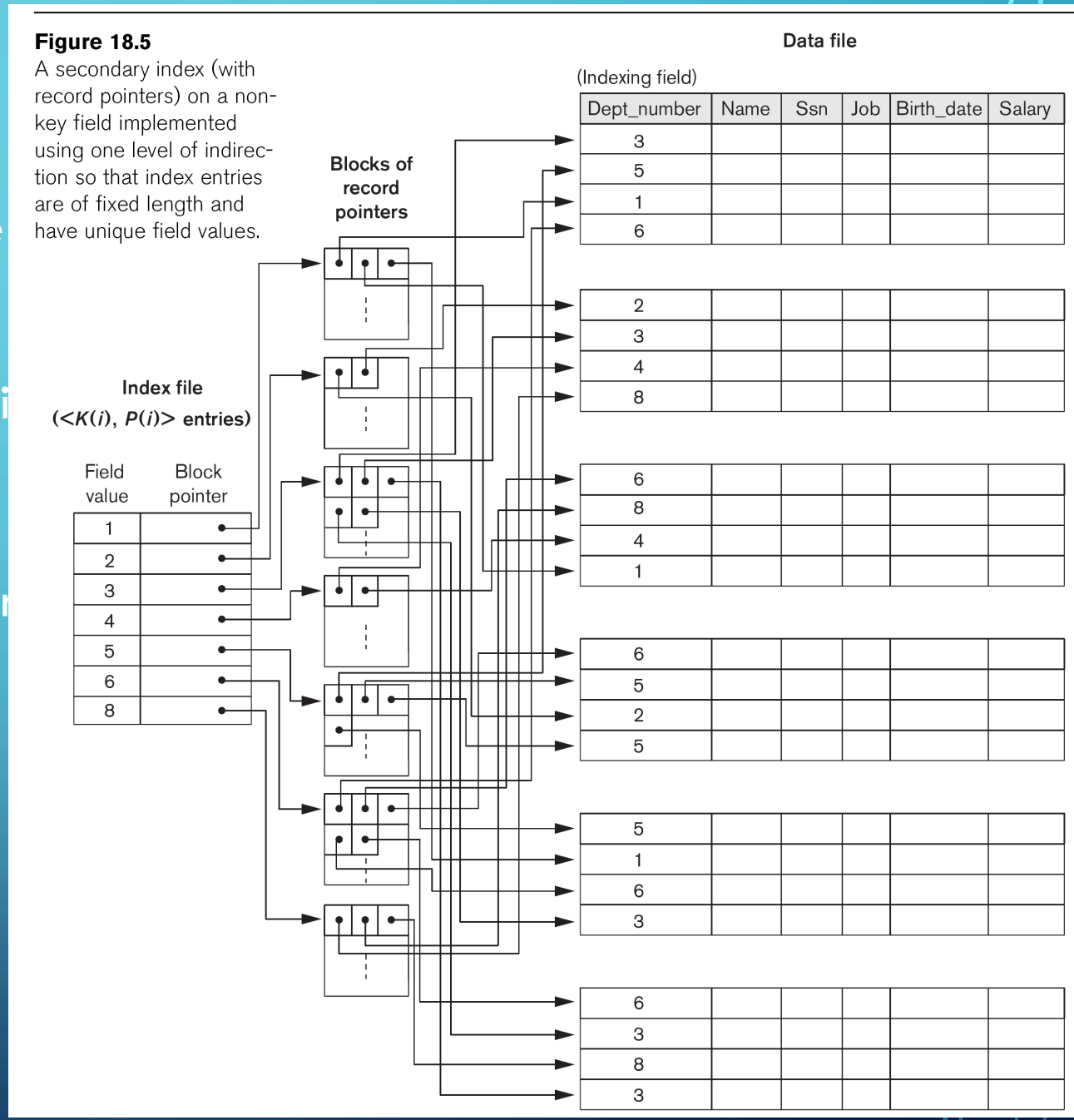


Figure 18.5
A secondary index (with record pointers) on a non-key field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

EXAMPLE

Index file which is 834 blocks long

Actual data on the disk which is 25000 blocks long

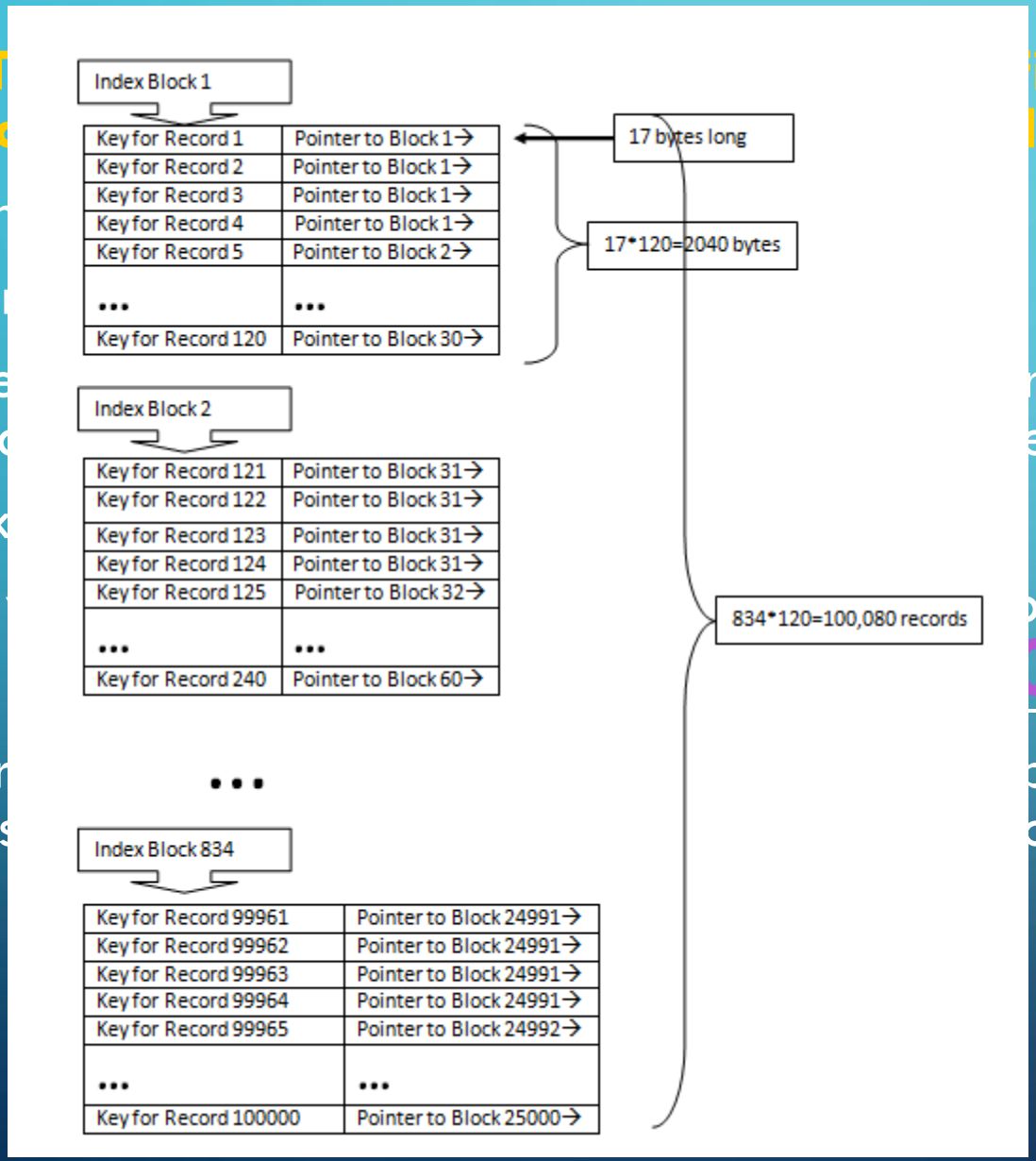
QUESTION: Find the worst case search time to find a record if you use a binary search on an dense (one entry for each record) index the following scenario:

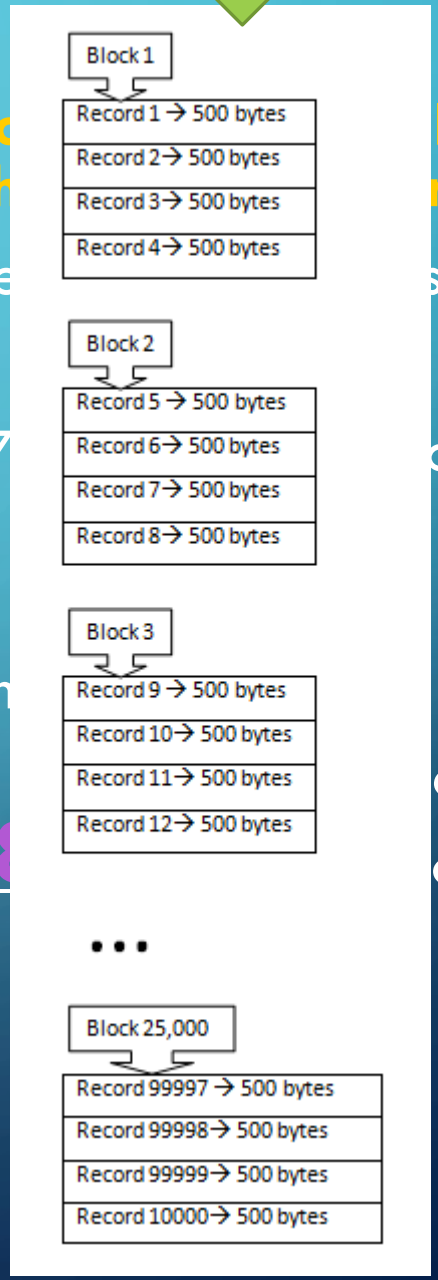
- We have100,000 records stored on a disk with block size B = 2048 bytes.
- Records are fixed size of R = 500 bytes.
- Ordering key field is K = 10 bytes, a block pointer P = 7 bytes, thus size of the primary index record is 17 bytes per record
- Blocking Factor for the index file = 2048/17
- Thus we can hold 120 key fields and block pointers on 1 block
- # of blocks needed for the index is 100,000/120 = 834 blocks
- Binary Search would need approximately log2b = $\log_2$ 834 = 10 block accesses + 1 to get to the data = 11 block accesses

EXAMPLE

Index file which is 834 blocks long

Actual data on the disk which is 25000 blocks long





QUEST **ind a reco** **binary**
searc **l) index th** **nario:**

- We h block size s.
- Reco
- Orde nter P = 7 of the prim ecord
- Block
- Thus pointers on
- # of cks
- Binar $b = \log_2$ ock acce cesses

QUESTION: Find the w … to find a record if y…
an sparse (one entry for each …OCK not each record) index the following s…nario:

Index file which is 209 blocks long

Actual data on the disk which is 25000 blocks long

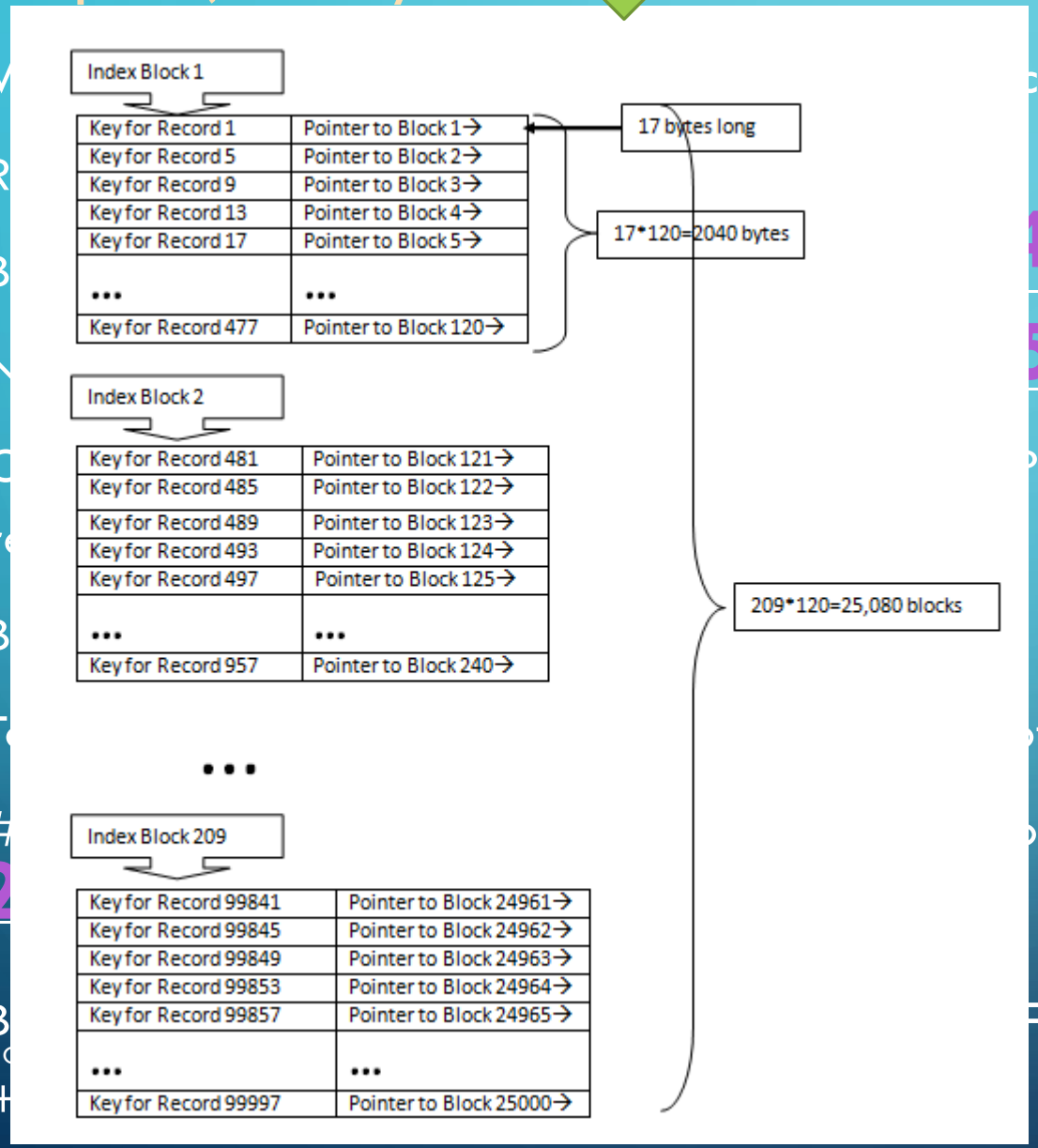
- We have 100,000 records stored on a disk with block size B = 2048 bytes.
- Records are fixed size of R = 500 bytes.
- Blocking Factor for the records = 2048 / 500 = 4
- Number of blocks needed to hold the records = 25000
- Ordering key field is K = 10 bytes, a block pointer P = 7 bytes, thus size of the primary record is 17 bytes per record
- Blocking Factor for index = 2048 /17 = 120
- Total number of entries in the index = total number of blocks= 25000
- # of blocks needed for the index is = total number of blocks / Blocking Factor for Index = 25000/120= 209 blocks
- Binary Search would need approximately $\log_2 b = \log_2 209 = 8$ block accesses + 1 to get to the data = 9 block accesses

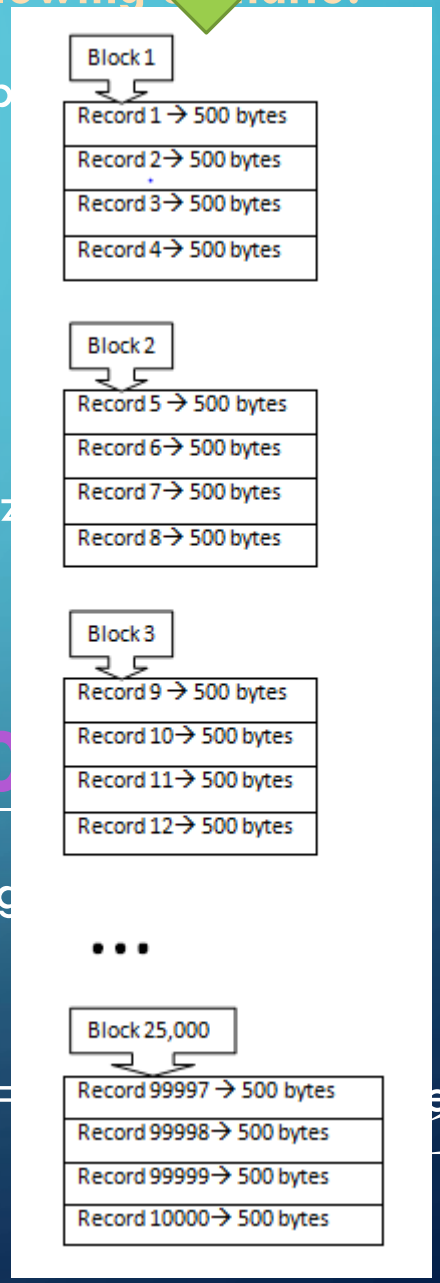
QUESTION: Find the v … to find a record if y…

…an sparse (one entry for each …LOCK not each record) index the following s…nario:

Index file which is 209 blocks long

Actual data on the disk which is 25000 blocks long





- V…
- R…
- B…
- N…
- C… r…
- B…
- T…
- #… 2…
- B… +…

…ck size B = 2048 b…

…4

…5000

…P = 7 bytes, thus siz…

…f blocks= 250…

…f blocks / Blocking…

…= $\log_2$ 209 =…

…es

…18