Database Technology

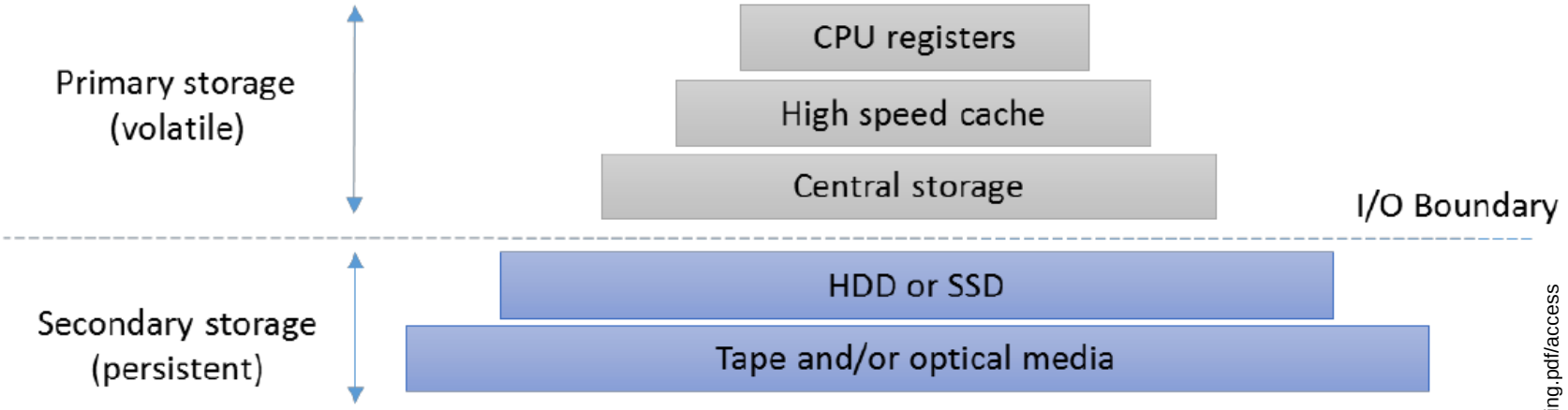
Topic 8: Data Structures for Databases

Olaf Hartig
olaf.hartig@liu.se

LINKÖPING UNIVERSITY

Storage Hierarchy

Quiz



Which of the following statements *is correct*?

1) Secondary storage devices are usually faster than primary storage devices.
2) Data in a primary storage device may be lost when switching of the power.
3) The CPU may operate directly on data that is in a secondary storage device.
4) A piece of data (e.g., a record) may not be held both in a primary storage device and in a secondary storage device at the same time.

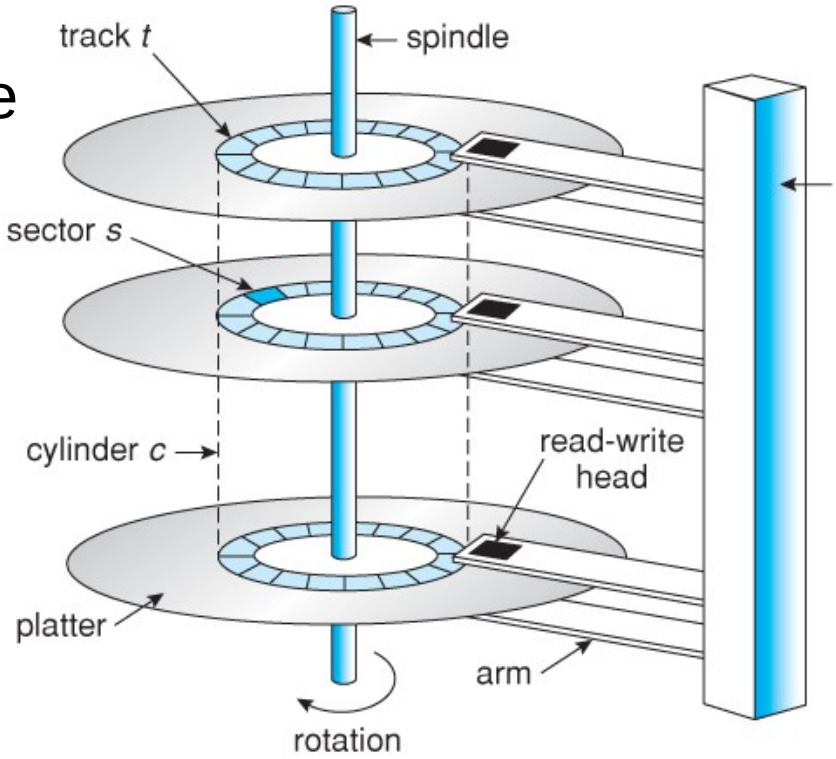
Image source: https://www.pdbmbook.com/vault/Chapter%20Physical%20File%20Organization%20and%20Indexing.pdf/access

Storage Hardware

Quiz

Which of the following statements on an HDD is *not* correct?

1) The platters containing magnetic particles are secured on a spindle that rotates at a constant speed.
2) An HDD needs three-dimensional movements in order to access all of its data.
3) All data on the same cylinder can be read without moving the actuator.
4) The tracks represent concentric circles of magnetic particles; each track consists of individual sectors.



Record Organization

(Organizing Fields / Data Items in Records)

Quiz

T1

A1	A2	A3
alice	3	100
bob	5	23

T2

A1	A2	A3
alice	NULL	41
bob	NULL	NULL

Assume we have two tables, T1 and T2, such that the rows in T1 do cannot contain NULL values whereas rows in T2 may contain several NULL values.

Each table should be stored in a separate physical file.

Which *record organization technique* should we choose for these files if we want to minimize storage space efficiently?

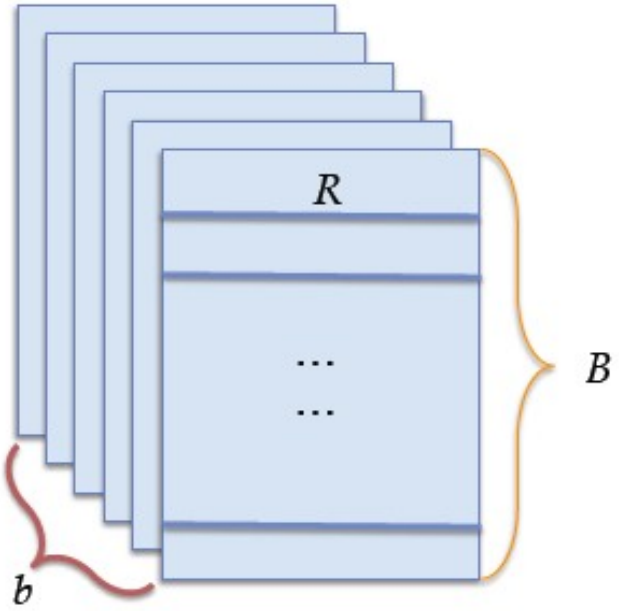
1) *Embedded identification* for the file of T1 and *relative location* for the file of T2
2) *Embedded identification* for the file of T2 and *relative location* for the file of T1
3) *Embedded identification* for both files.
4) *Relative location* for both files.

Record Allocation

(Allocating Record to File Blocks)

Quiz

- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,

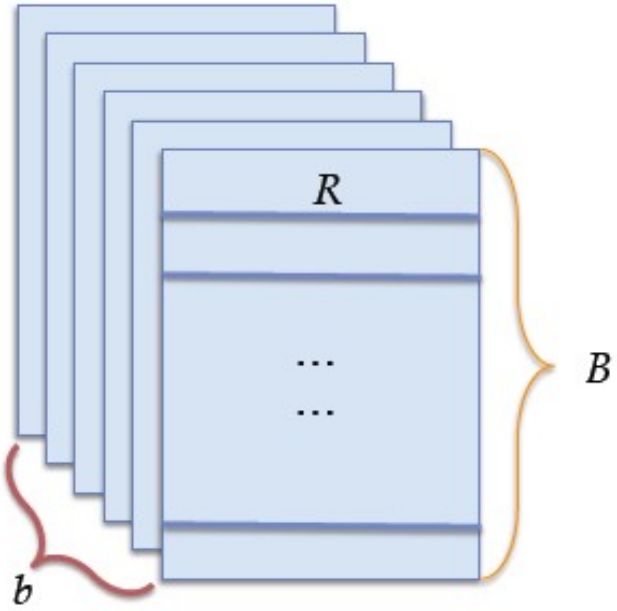


- How many blocks are needed to store the file?

1) b = 10 2) b = 20 3) b = 100 4) b = 200

Quiz

- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,



- How many blocks are needed to store the file?

1) $b = 10$ 2) $b = 20$ 3) $b = 100$ 4) $b = 200$

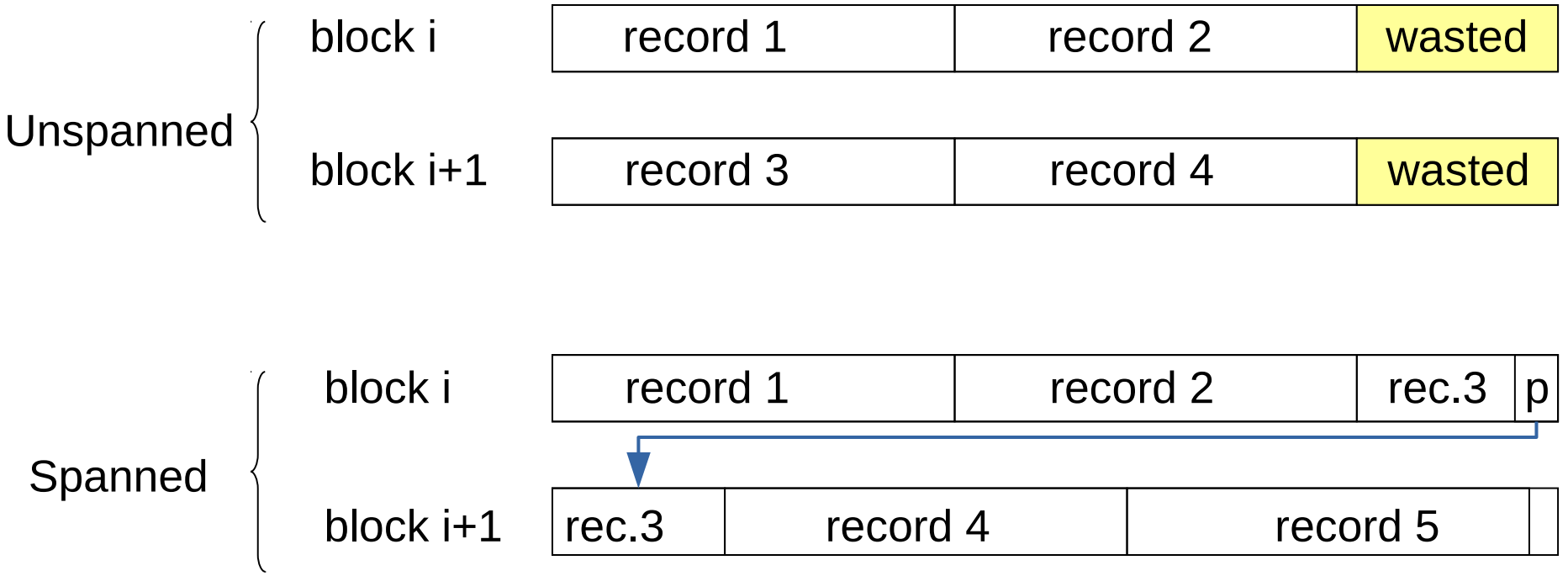
$$bfr = \left\lfloor \frac{B}{R} \right\rfloor \qquad b = \left\lceil \frac{r}{bfr} \right\rceil$$

blocking factor

- Space wasted per block = $B - bfr * R$

Spanned Records

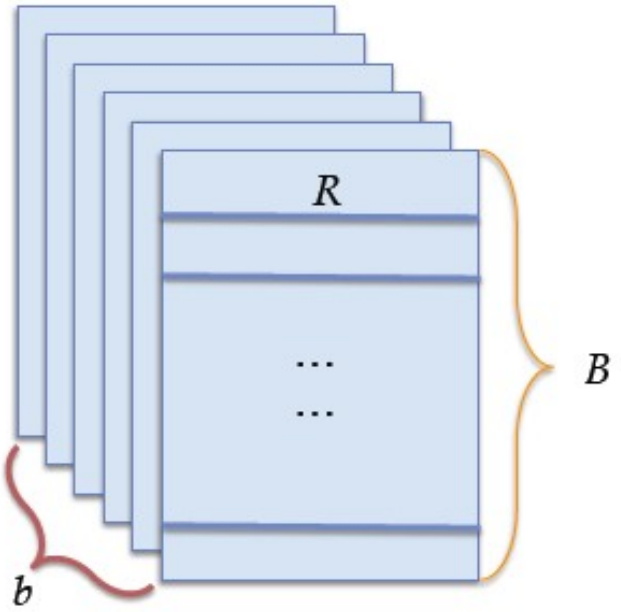
… avoid wasting space



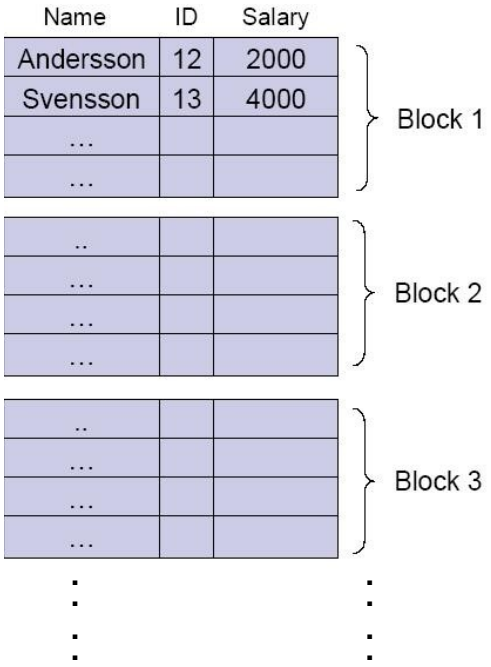
File Organization

(Organizing Records in Files)

Exercise: Heap File

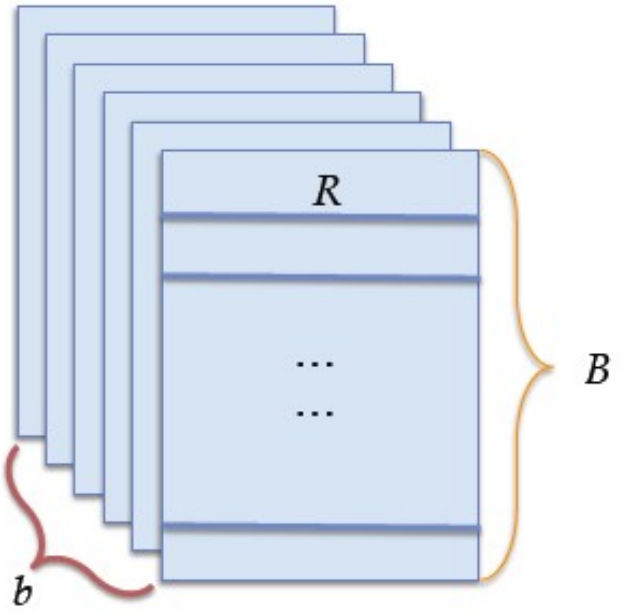


- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a heap file
  - i.e., new records are always appended to the end of the file
- How many blocks do we need to read?

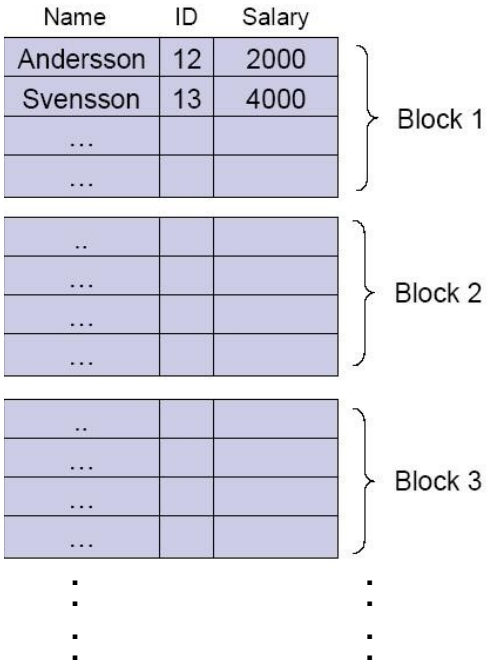


	search field = ID value = 43 (unique)	search field = Name value = Smith (non-unique)
worst case		
best case		
average case		

Exercise: Heap File



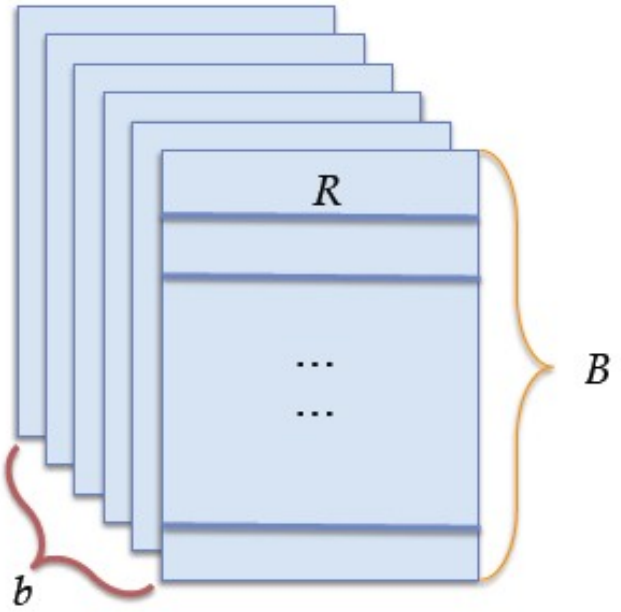
- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a heap file
  - i.e., new records are always appended to the end of the file
- How many blocks do we need to read?



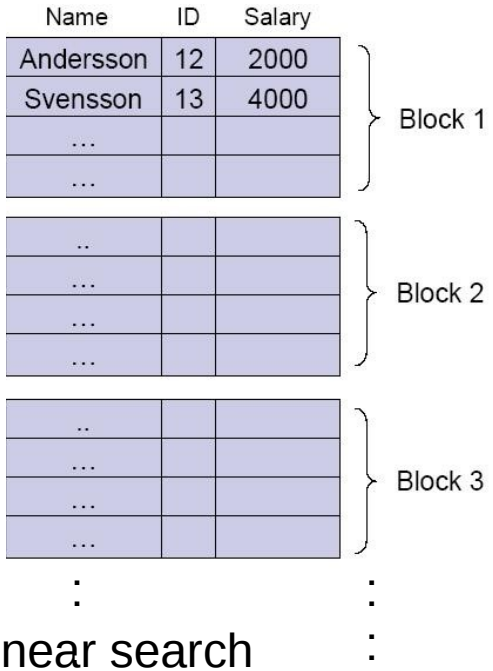
	search field = ID value = 43 (unique)	search field = Name value = Smith (non-unique)
worst case	200	
best case	1	
average case	100	

$\left\lceil \frac{b}{2} \right\rceil$

Exercise: Heap File



- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a heap file
  - i.e., new records are always appended to the end of the file
- How many blocks do we need to read?



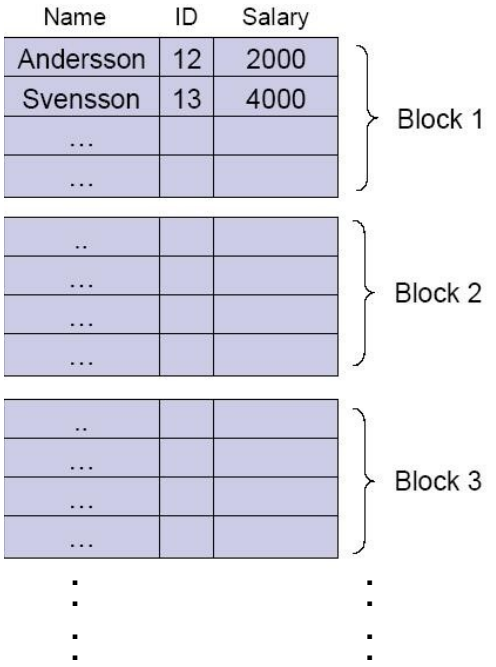
	search field = ID value = 43 (unique)	search field = Name value = Smith (non-unique)
worst case	200	200
best case	1	200
average case	100	200

linear search until last block

$\left\lceil \frac{b}{2} \right\rceil$

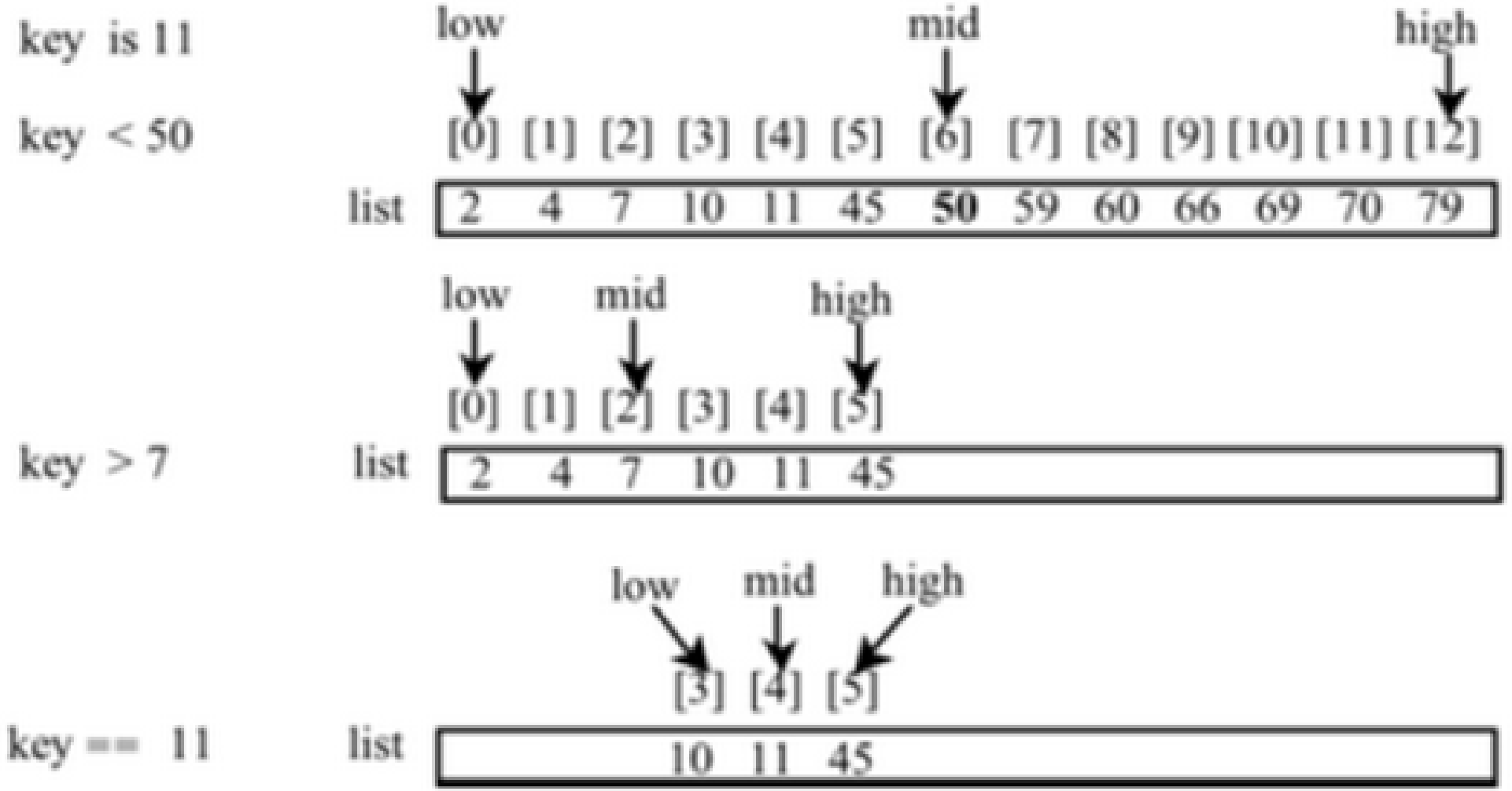
Exercise: Sorted File (a.k.a. Sequential File)

- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a sorted file by using the ID field as the *sorting field*
  - i.e., records inserted based on their ID value
- How many blocks do we need to read?



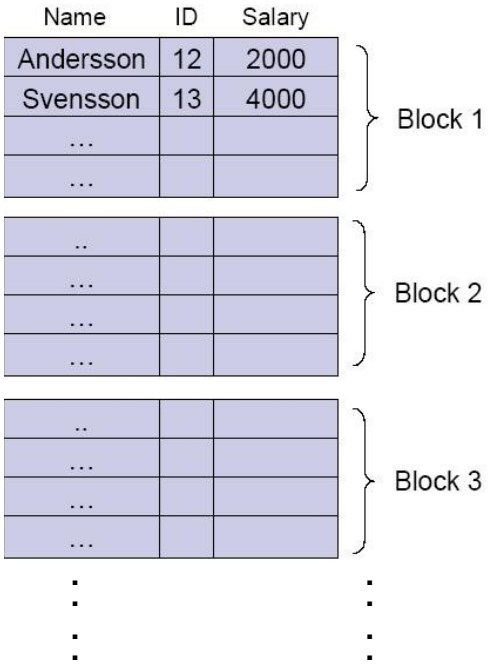
	search field = ID value = 43 (unique)	search field = Name value = Smith (non-unique)
worst case		
best case		
average case		

Binary Search



Exercise: Sorted File (a.k.a. Sequential File)

- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a sorted file by using the ID field as the *sorting field*
  - i.e., records inserted based on their ID value
- How many blocks do we need to read?



	search field = ID value = 43 (unique)	search field = Name value = Smith (non-unique)
worst case	8	200
best case	1	200
average case	8	200

$\lceil \log_2 b \rceil$

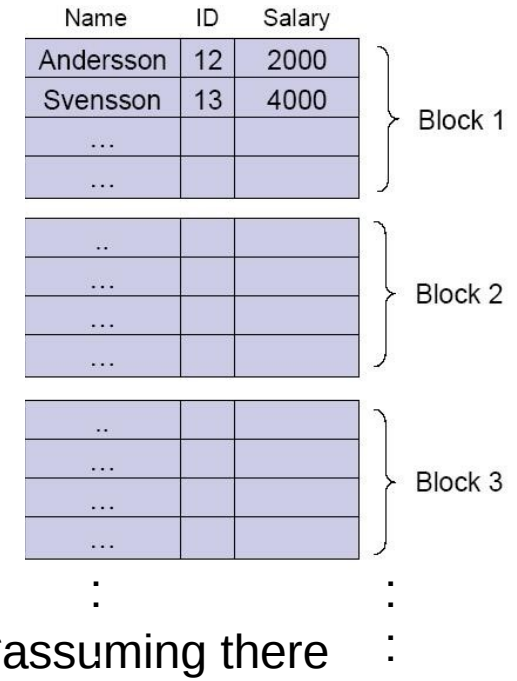
$\log_2(128) = 7$

$\log_2(256) = 8$

Exercise: Hash File (a.k.a. Random File Orga.)

- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a hash file by using the ID field as the *hash field* and 120 buckets with 2 blocks per bucket
- How many blocks do we need to read?*

	search field = ID value = 43 (unique)	search field = Name value = Smith (non-unique)
worst case		
best case		
average case		

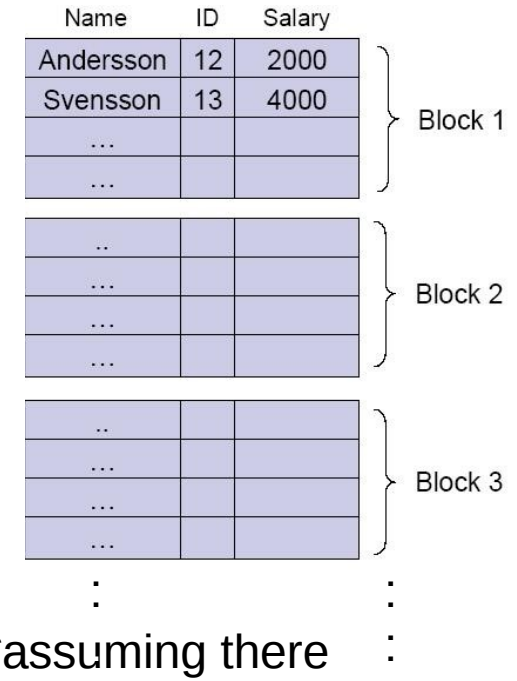


*assuming there are no collisions

Exercise: Hash File (a.k.a. Random File Orga.)

- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a hash file by using the ID field as the *hash field* and 120 buckets with 2 blocks per bucket
- How many blocks do we need to read?*

	search field = ID value = 43 (unique)	search field = Name value = Smith (non-unique)
worst case	2	≥ 200
best case	1	≥ 200
average case	1.5	≥ 200

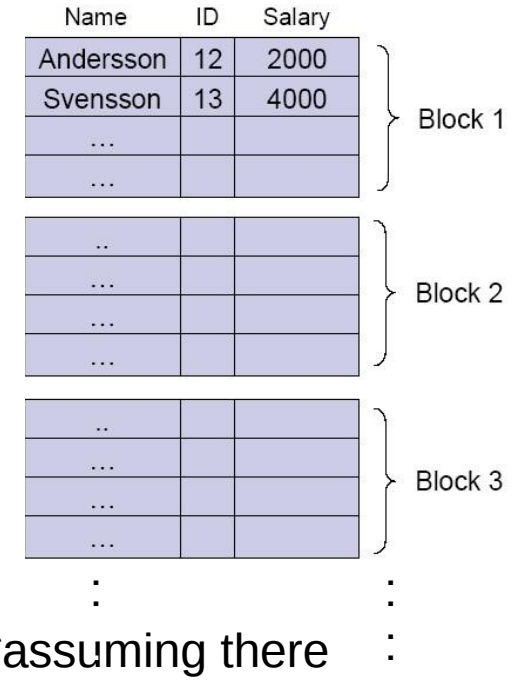


*assuming there are no collisions

Exercise: Hash File (a.k.a. Random File Orga.)

- Assume a file with
  - r = 2,000 records,
  - R = 100 bytes per record, and
  - B = 1,000 bytes per block,
- Hence, b = 200 blocks needed to store the file
- Assume we organize the file as a hash file by using the ID field as the *hash field* and 120 buckets with 2 blocks per bucket
- What if we want to retrieve all records with an ID value and smaller than 10?*

	search field = ID value = 43 (unique)
worst case	9 · 2 = 18
best case	1
average case	depends



*assuming there are no collisions

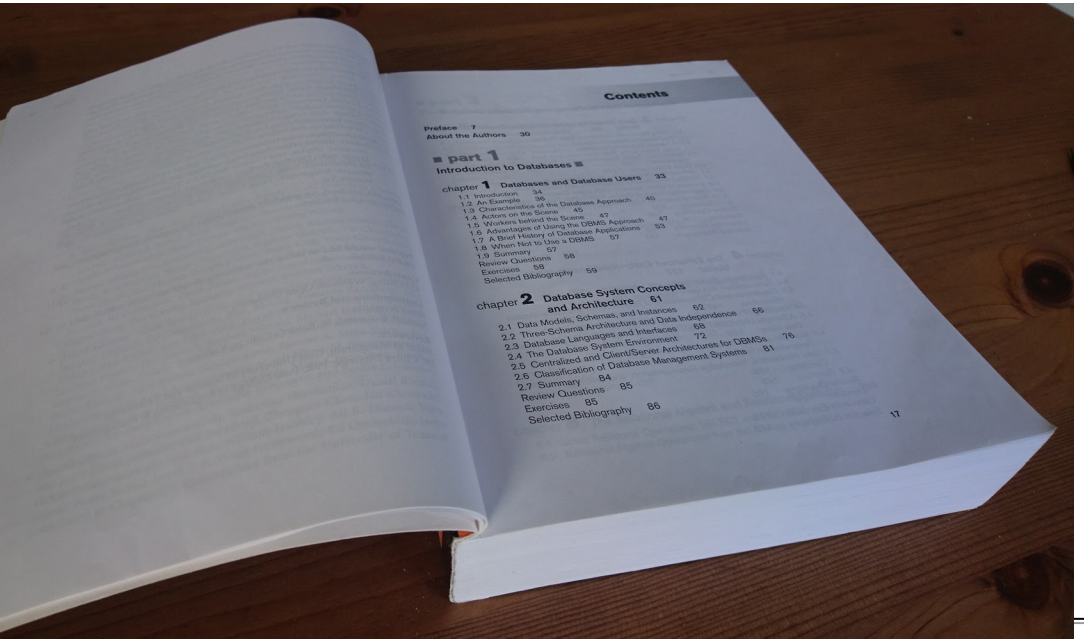
…and IDs cannot be smaller than 1

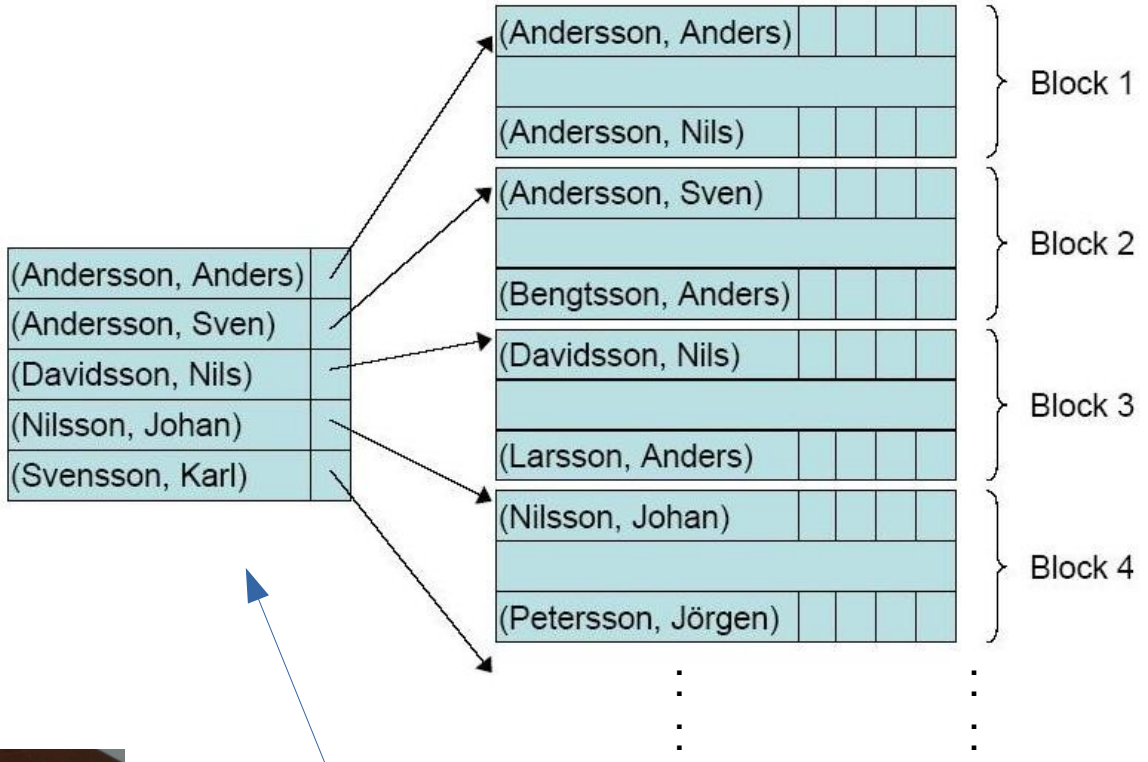
Index Sequential File Organization

(Creating an index on the
sorting field of a sorted file)

Primary Index

Why is it faster to find a random record via a binary search in the index rather than in the (sorted) data file?



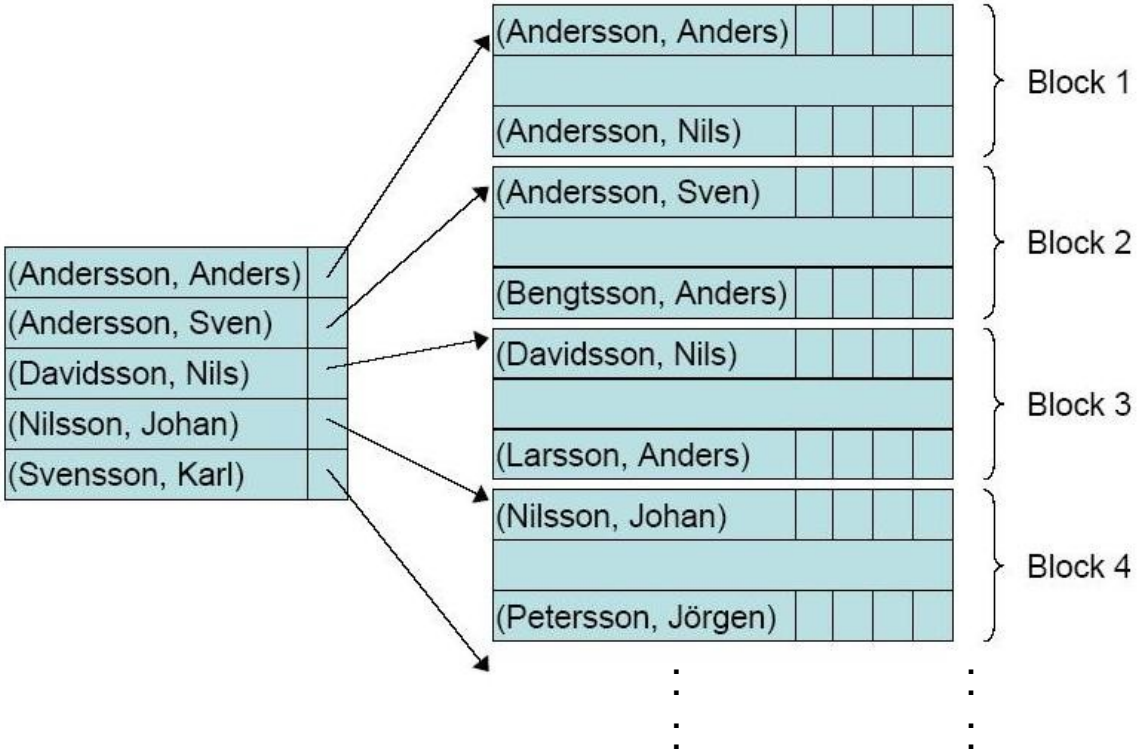


sorted file with all the records

- sorted by name, and
- name is a key

index file (sorted) with one record per block in the data file

Primary Index

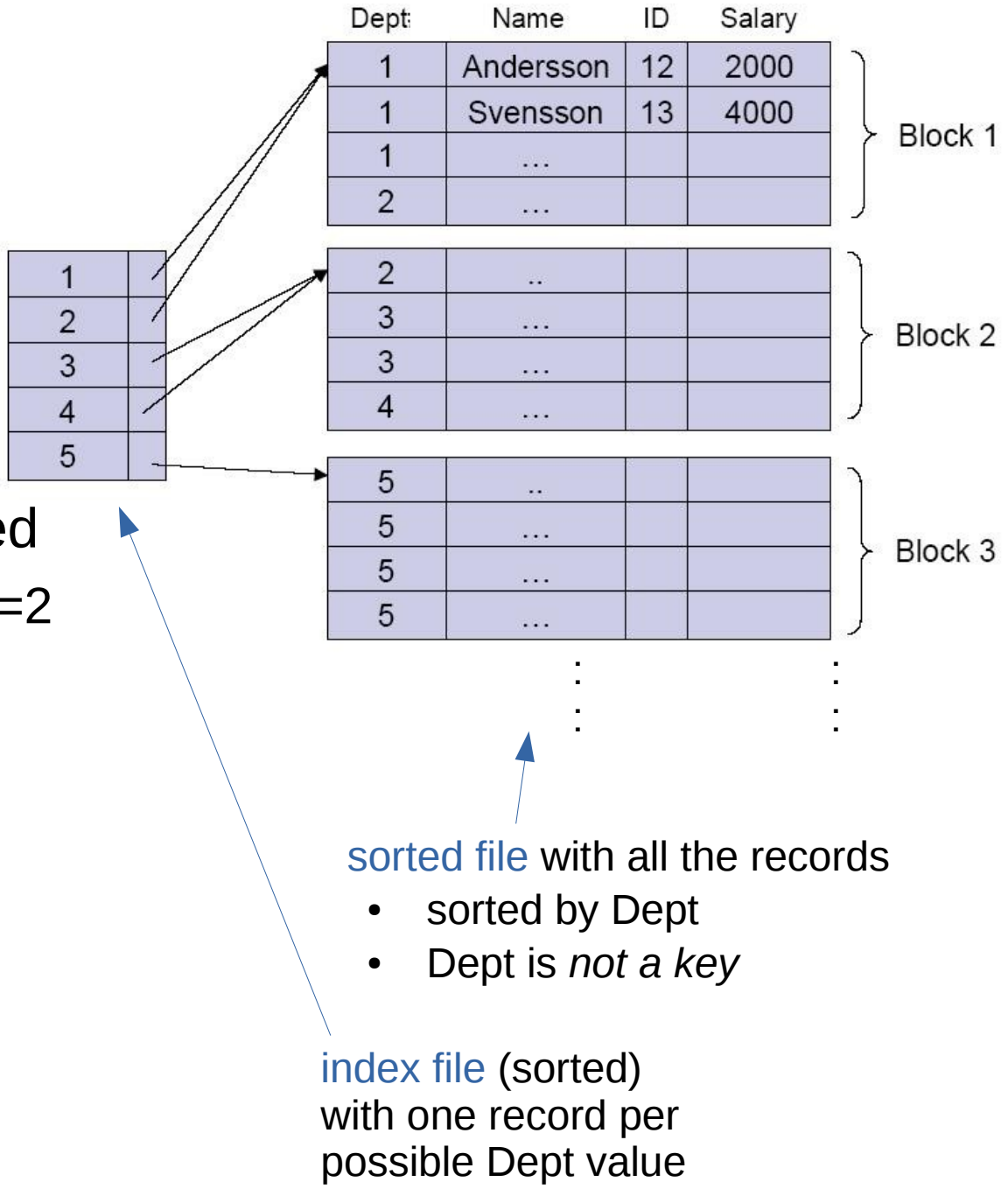


Why is it faster to find a random record via a binary search in the index rather than in the (sorted) data file?

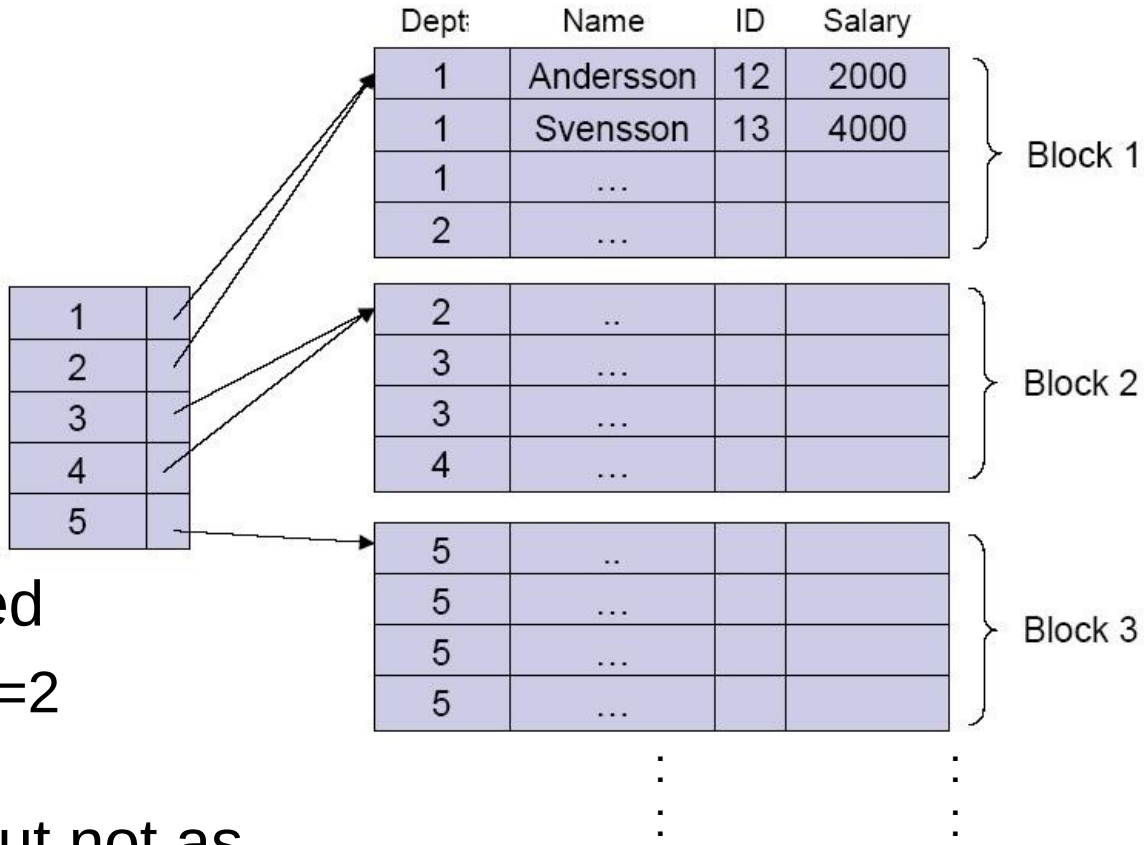
- Index file has significantly fewer blocks because:
  - number of index records << number of data records
  - Index records smaller than data records (i.e., blocking factor for the index file higher than for the data file)

Clustering Index

- Attention: after binary search in the index file, multiple data file blocks may need to be accessed
  - see, for instance, Dept=2

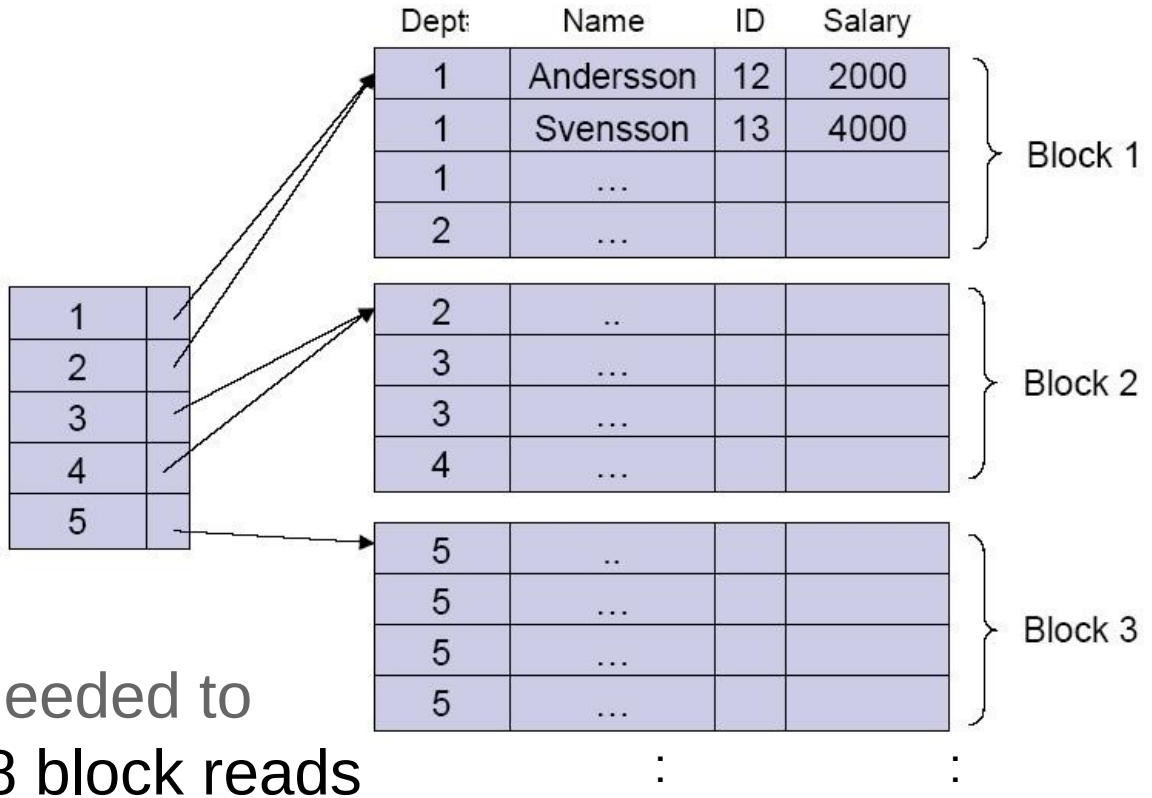


Clustering Index



- Attention: after binary search in the index file, multiple data file blocks may need to be accessed
  - see, for instance, Dept=2
- Index file also smaller, but not as much as for a primary index
  - number of index records ≤ number of data records
  - at least, index records smaller than data records (like in a primary index)

Exercise



- Assume sorted file with
  r = 2,000 records,
  R = 100 bytes per record,
  B = 1,000 bytes per block
- Hence, b = 200 blocks needed to store the file and, thus, 8 block reads for a binary search on the file
- Assume
  r' = 300 different *Dept* values
  R' = 10 bytes per index record
  B = 1,000 bytes per index block
- How many block reads for a binary search on the index?

$\lceil \log_2 b \rceil$

$$bfr = \left\lfloor \frac{B}{R} \right\rfloor \qquad b = \left\lceil \frac{r}{bfr} \right\rceil$$

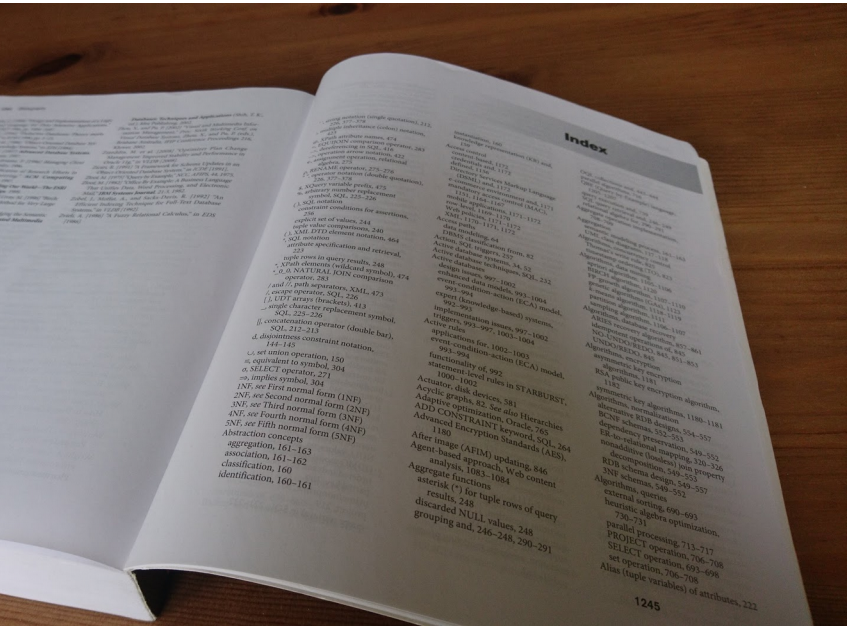
$\log_2(2) = 1,\ \log_2(4) = 2,$
$\log_2(8) = 3,\ \log_2(16) = 4$
$\log_2(32) = 5,\quad \log_2(64) = 6,$
$\log_2(128) = 7,\quad \log_2(256) = 8$

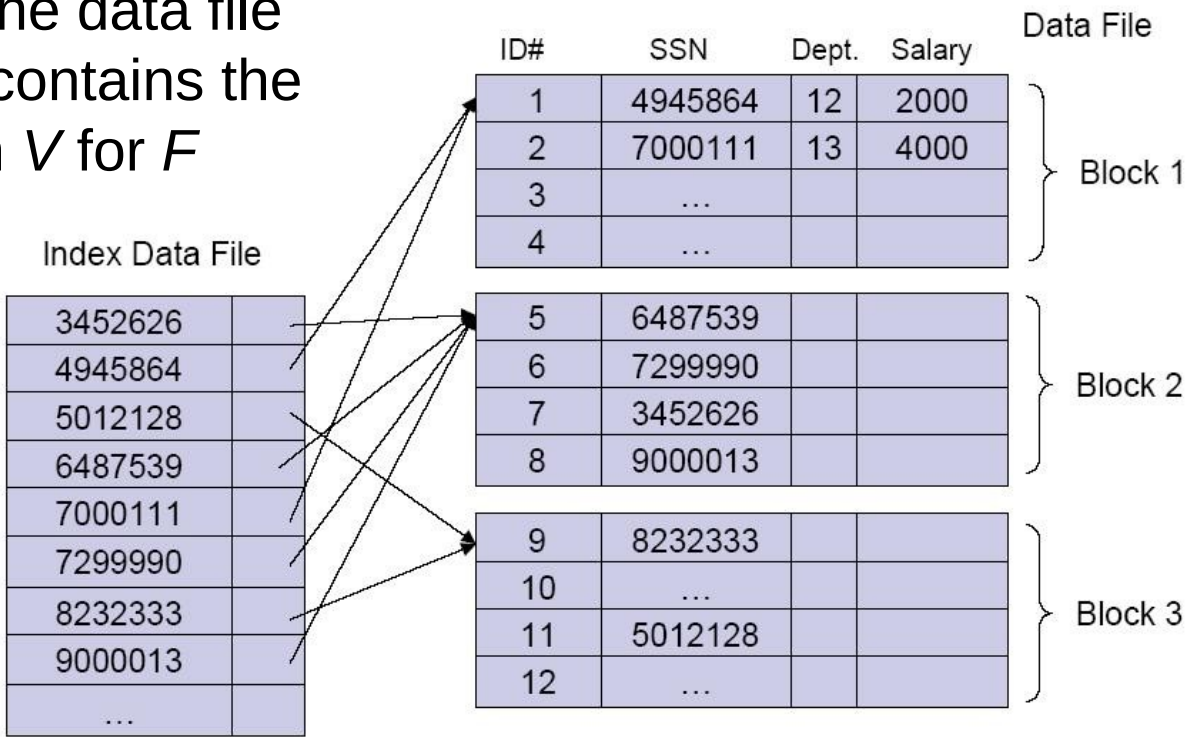
Secondary Indexes

(Creating an index on a field other than the sorting field)

Secondary Indexes on Key Field

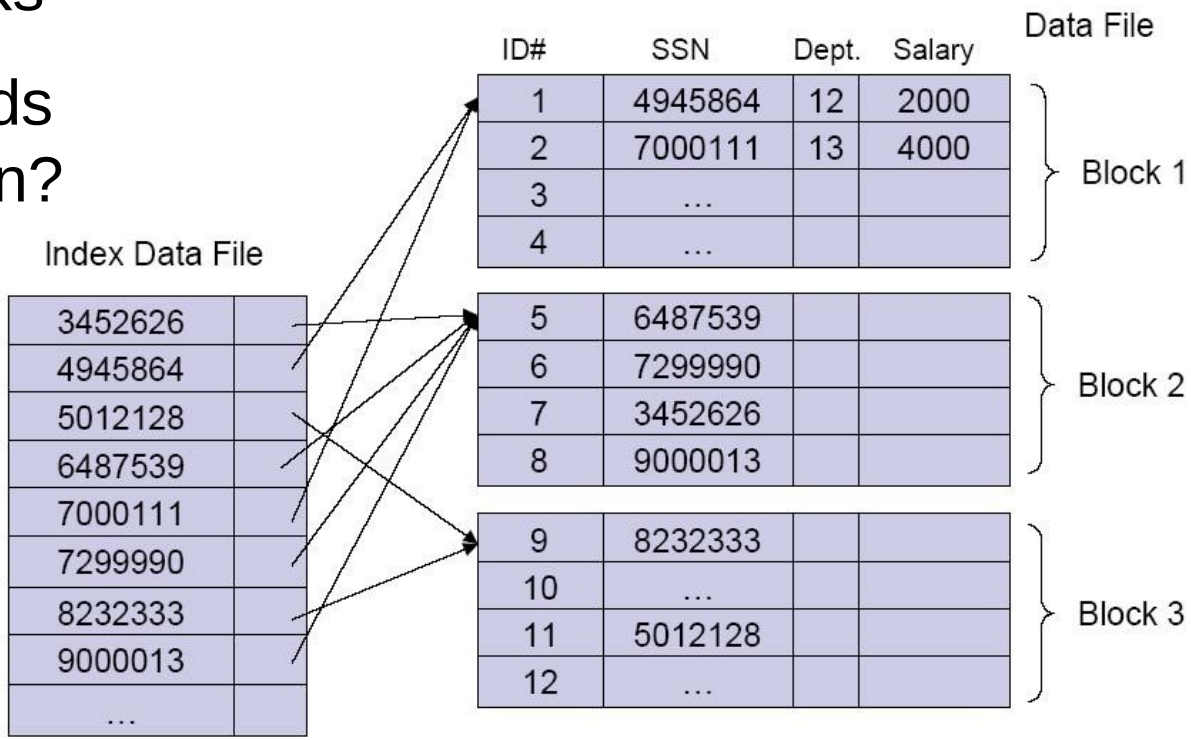
- Index on a *non*-ordering *key* field *F*
  - Data file may be sorted or not
- Secondary index: additional *sorted* file whose records contain two fields:
  - *V* - one of the values of *F*
  - *P* - pointer to the data file block that contains the record with *V* for *F*





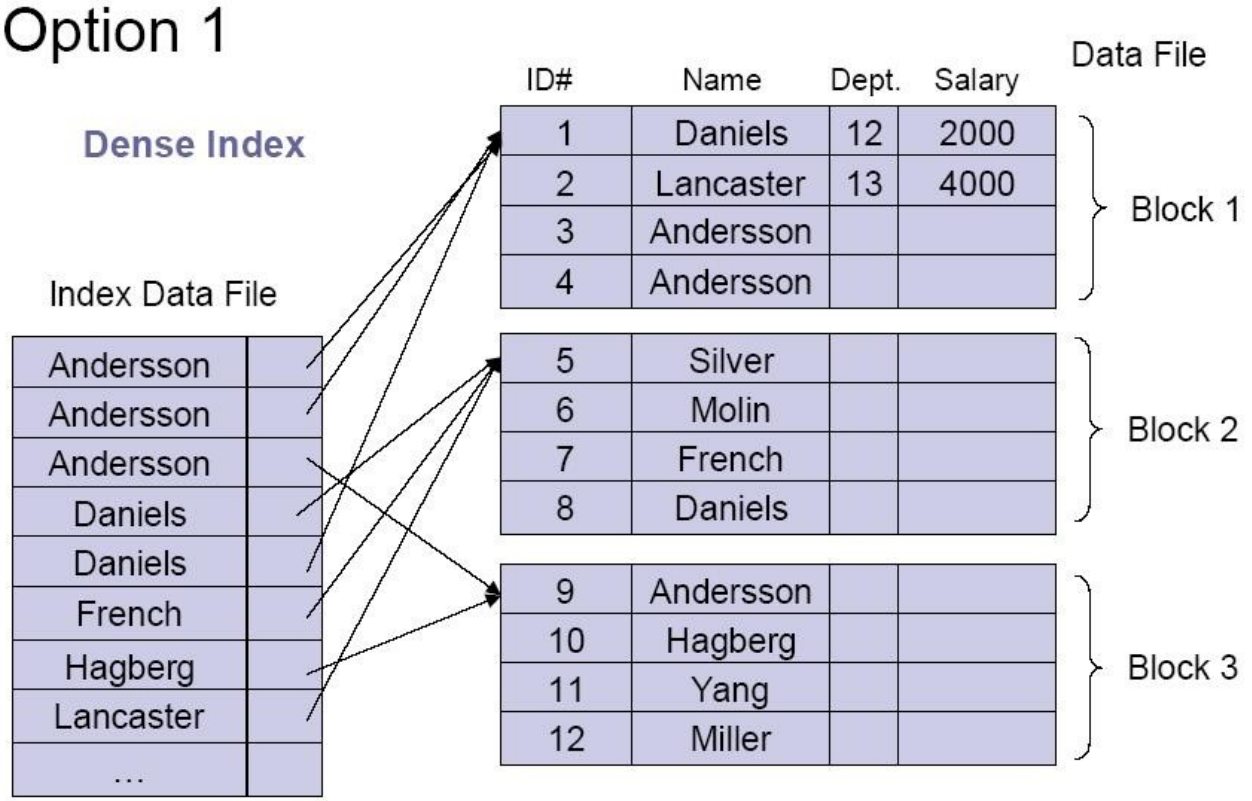
Quiz

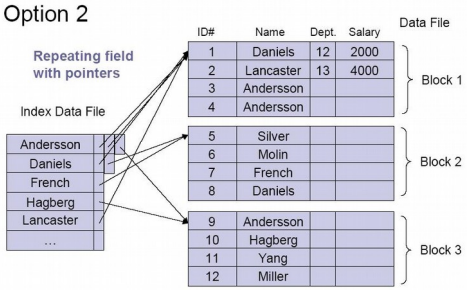
- Assume we create such a secondary index (on a *non*-ordering *key* field) over a data file that has
  - 2,000 records,
  - a blocking factor of 10,
  - and, thus, 200 blocks
- How many index records would this index contain?
  1) 20,000
  2) 2,000
  3) 1,000
  4) 200

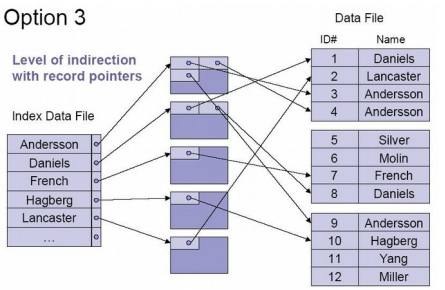


Secondary Indexes on Non-Key

- Index on a *non*-ordering *non*-key field

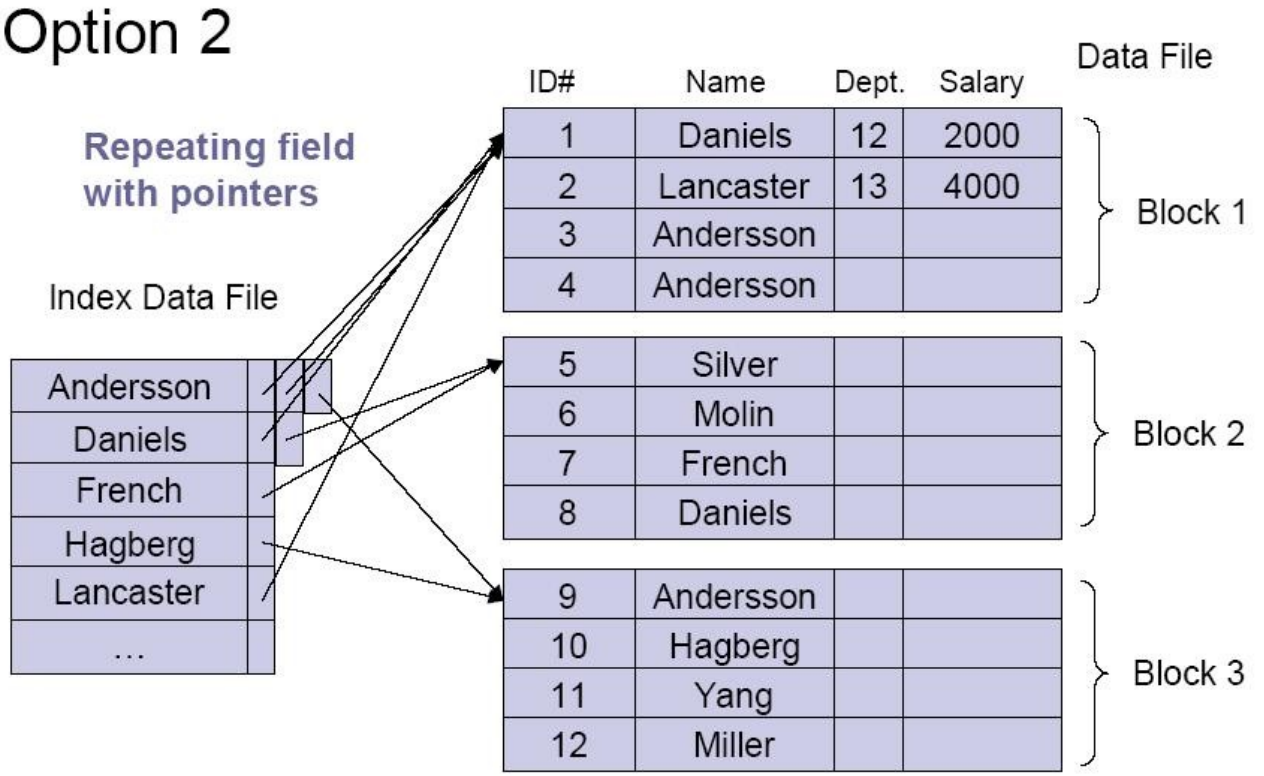


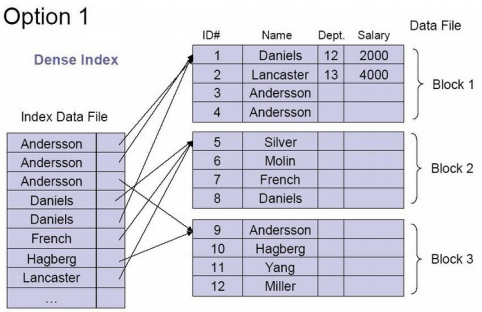


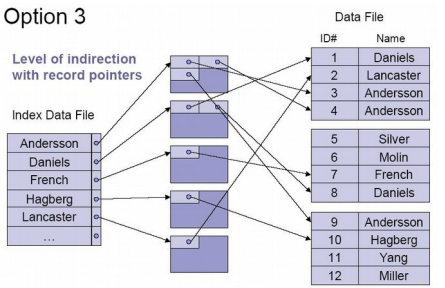


Secondary Indexes on Non-Key

- Index on a *non*-ordering *non*-key field

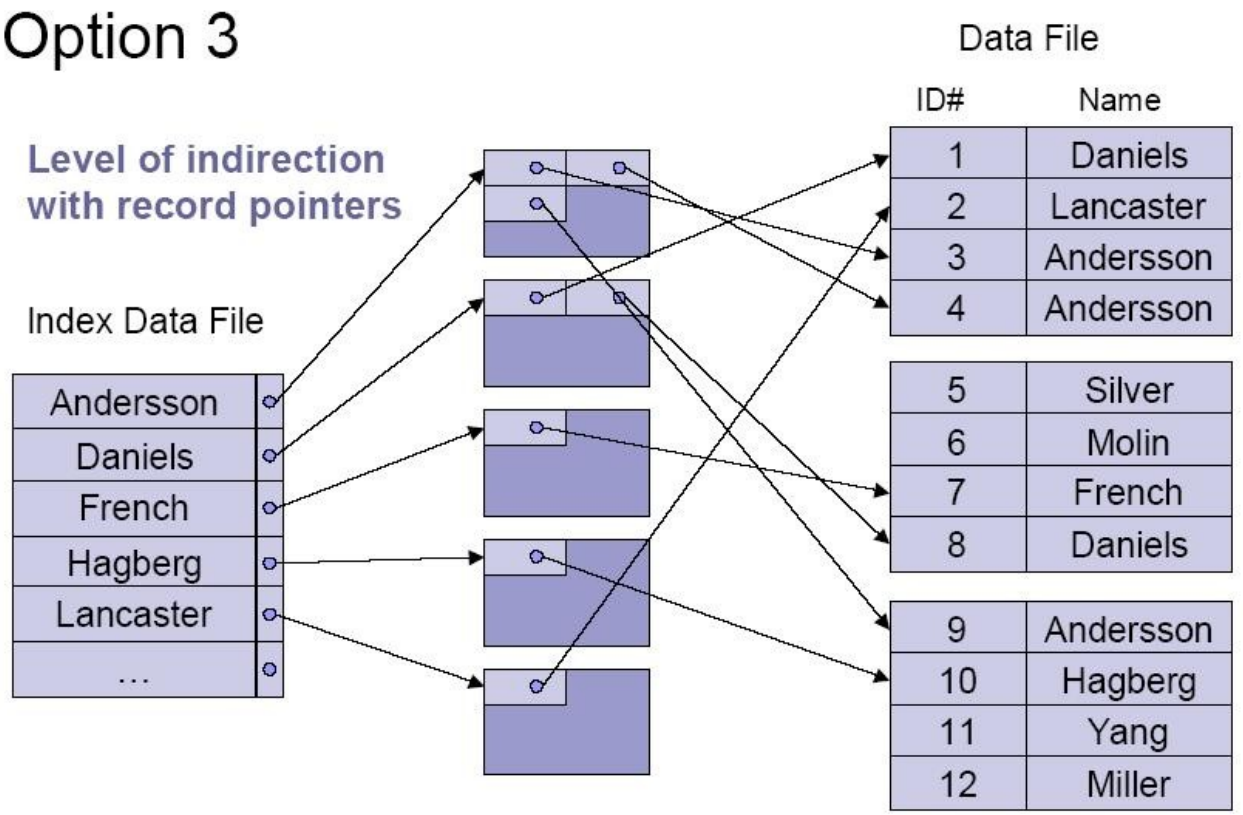


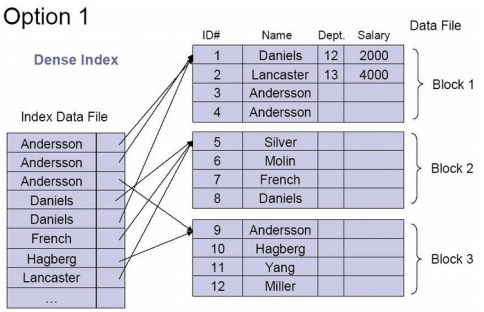


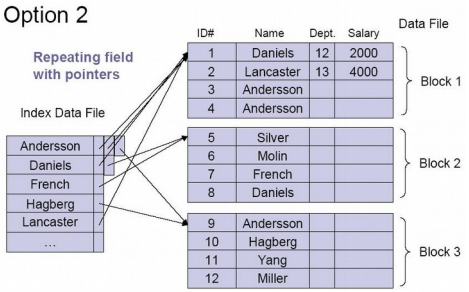


Secondary Indexes on Non-Key

- Index on a *non*-ordering *non*-key field
- also called *inverted file*







Summary of Single-Level Indexes

	Index field used for sorting the data records	**Index field *not* used for sorting the data records**
Index field is a key	Primary index	Secondary index (key)
Index field is not a key	Clustering index	Secondary index (non-key)

Quiz

	Index field used for sorting the data records	**Index field *not* used for sorting the data records**
Index field is a key	Primary index	Secondary index (key)
Index field is not a key	Clustering index	Secondary index (non-key)

Which of these four types of indexes has the *smallest number of index records*?

1) Primary index
2) Clustering index
3) Secondary index on a key field
4) Secondary index on a non-key field

Summary of Single-Level Indexes (cont'd)

	Index field used for sorting the data records	**Index field *not* used for sorting the data records**
Index field is a key	Primary index	Secondary index (key)
Index field is not a key	Clustering index	Secondary index (non-key)

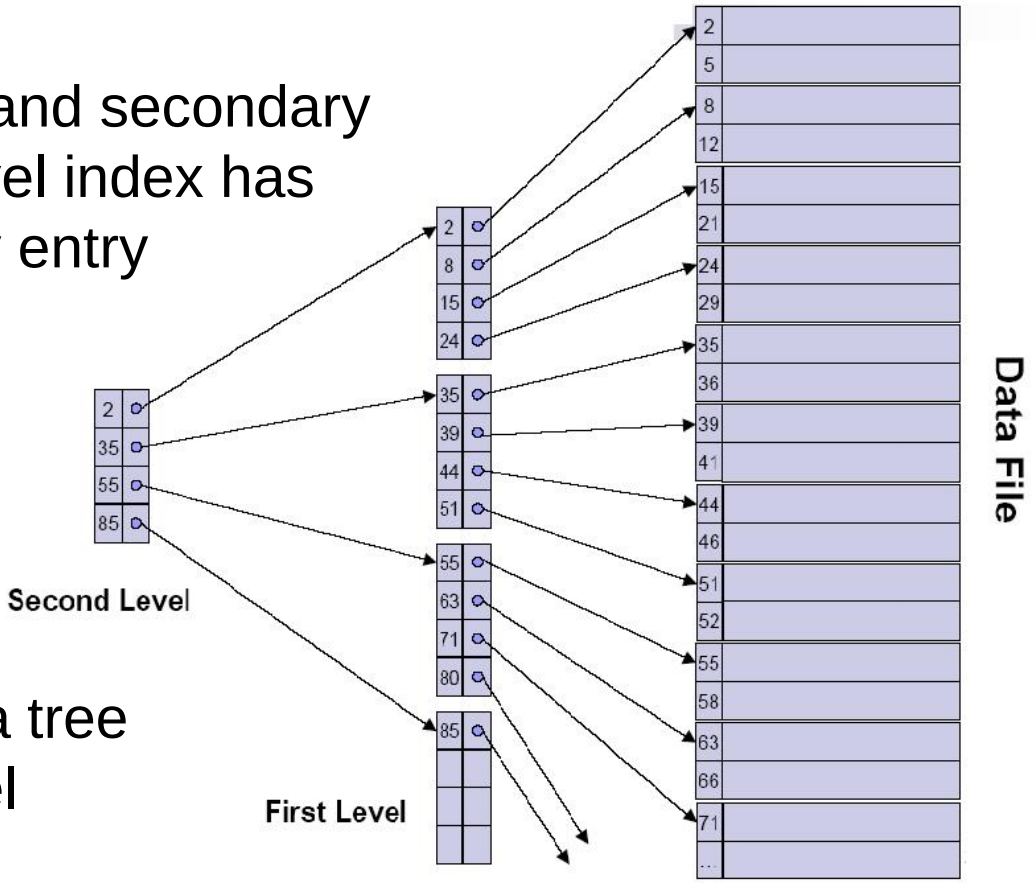
Type of index	**Number of index entries**
Primary	Number of blocks in data file
Clustering	Number of distinct index field values
Secondary (key)	Number of records in data file
Secondary (non-key)	Number of records or number of distinct index field values

Multilevel Indexes

(Stacking indexes on top of one another)

Multilevel Indexes

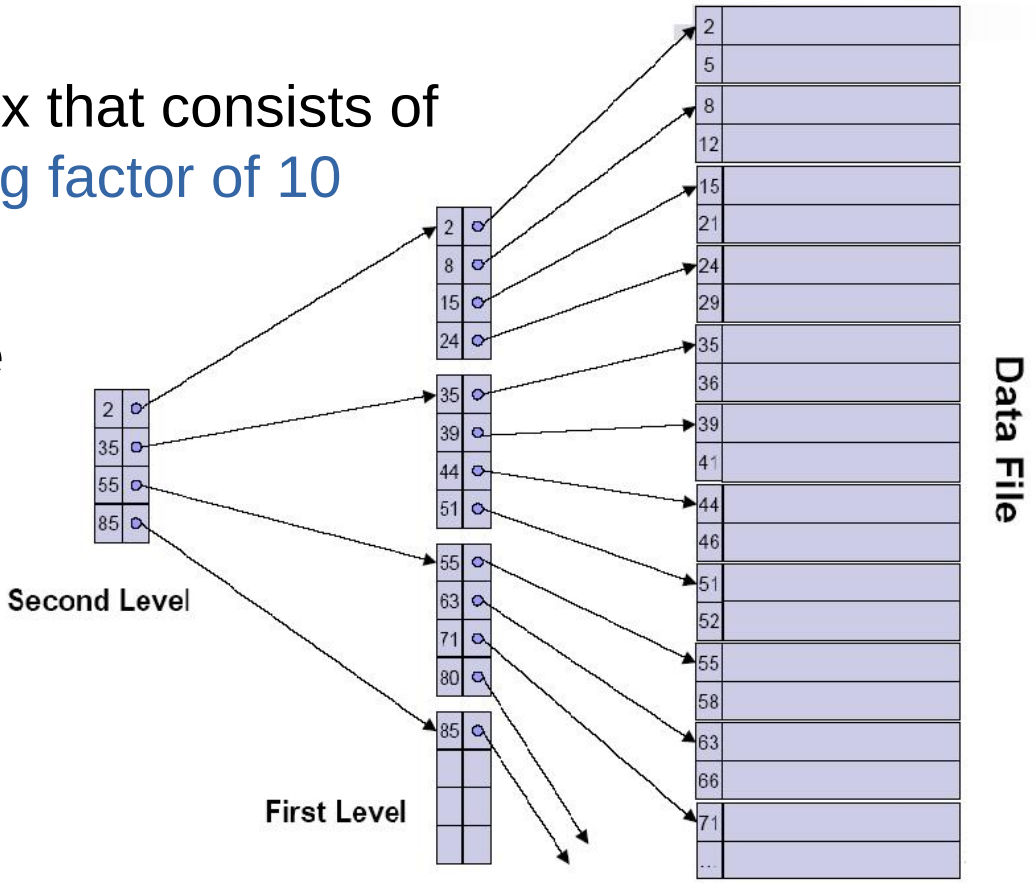
- Works for primary, clustering, and secondary indexes as long as the first-level index has a distinct index value for every entry
- How many levels?
  - until the highest level fits into a single block
- Such a full multilevel index is a tree
  - single block of highest level is the root node in this tree



Quiz

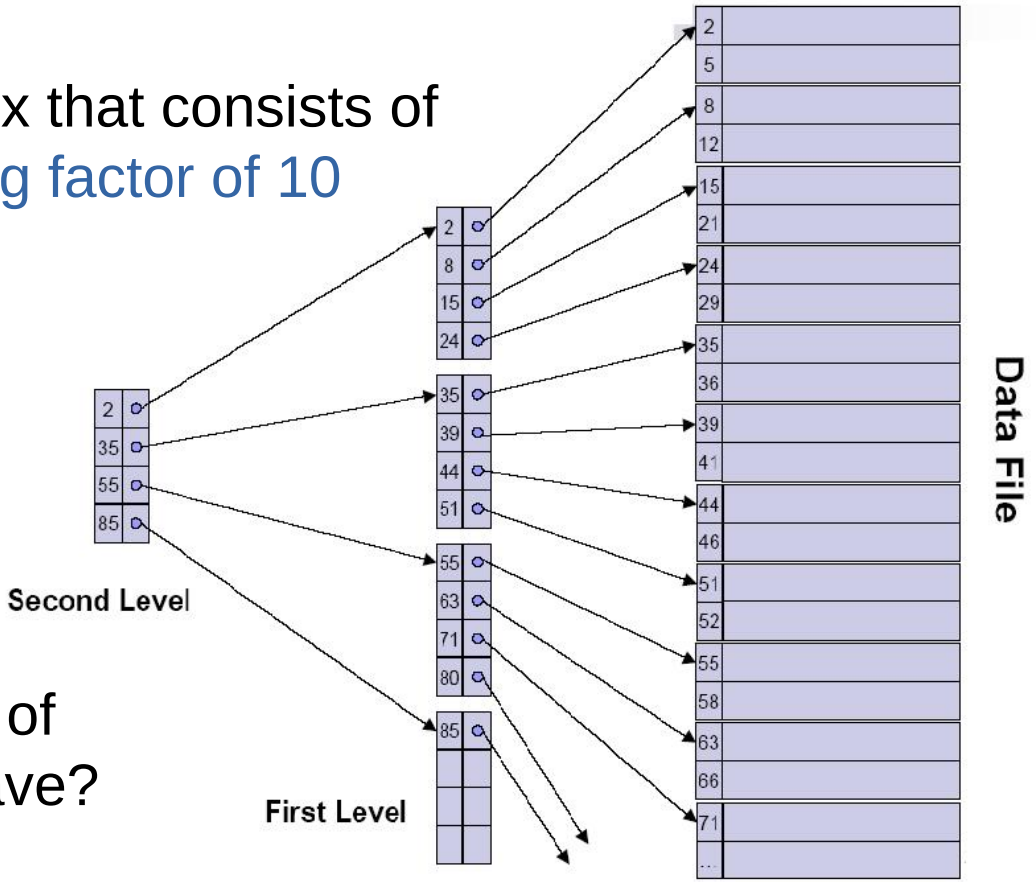
- Assume such a multilevel index that consists of 3 levels and that has a blocking factor of 10 for all index levels
- How many block accesses are needed to retrieve a random record if the index has been created on the search key?

  1) 30
  2) 10
  3) 4
  4) 3



Quiz

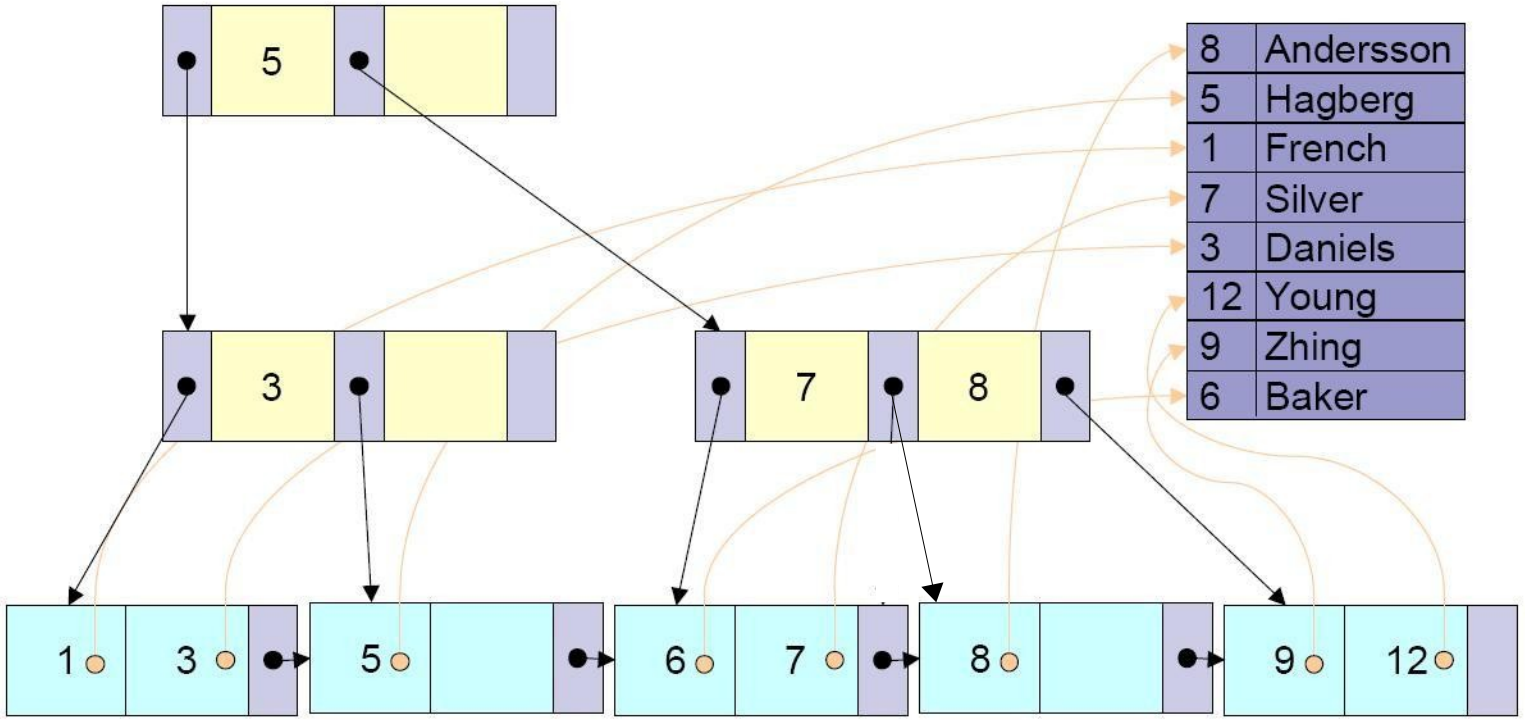
- Assume such a multilevel index that consists of 3 levels and that has a blocking factor of 10 for all index levels
- Assume the first level is a primary index
  - i.e., the data file is sorted on a key field and the index has been created on this key field
- What is the maximum number of blocks that the data file can have?
  1) 3,000
  2) 1,000
  3) 300
  4) 100



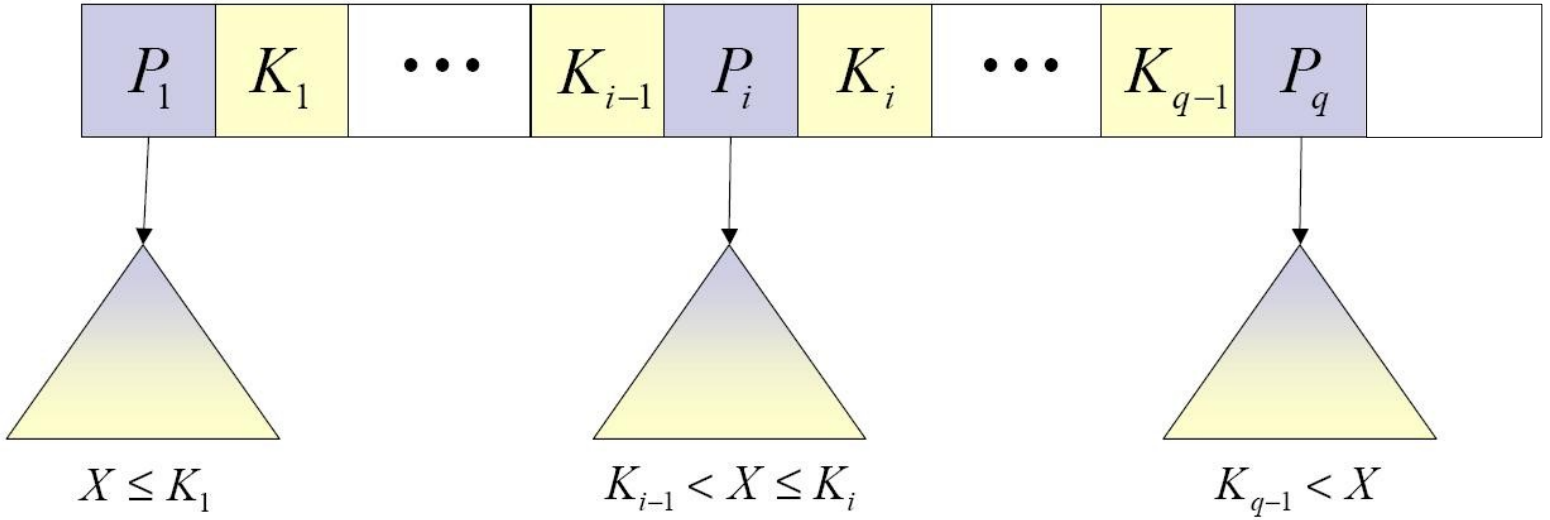
B^+-Trees

Dynamic Multilevel Indexes

Example B+-Tree

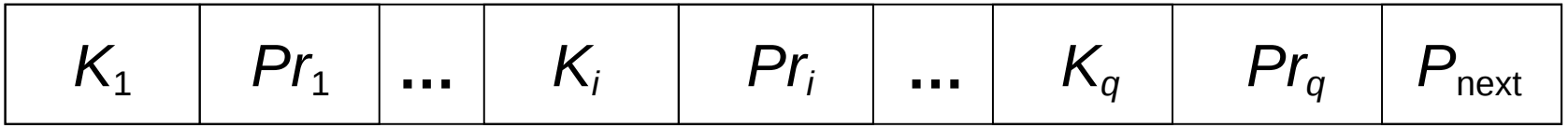


Internal Nodes of a B+-Tree



- $q \leq p$ (where p is the order of the B+-tree)
- Every K_i is an index value, every P_i is a tree pointer
- Within each node: $K_1 < K_2 < \ldots < K_{q-1}$
- For every value X in the P_i subtree: $K_{i-1} < X \leq K_i$
- Each internal node (except the root) must be at least half full
  - i.e., there must be at least $\left\lceil \frac{p}{2} \right\rceil$ tree pointers

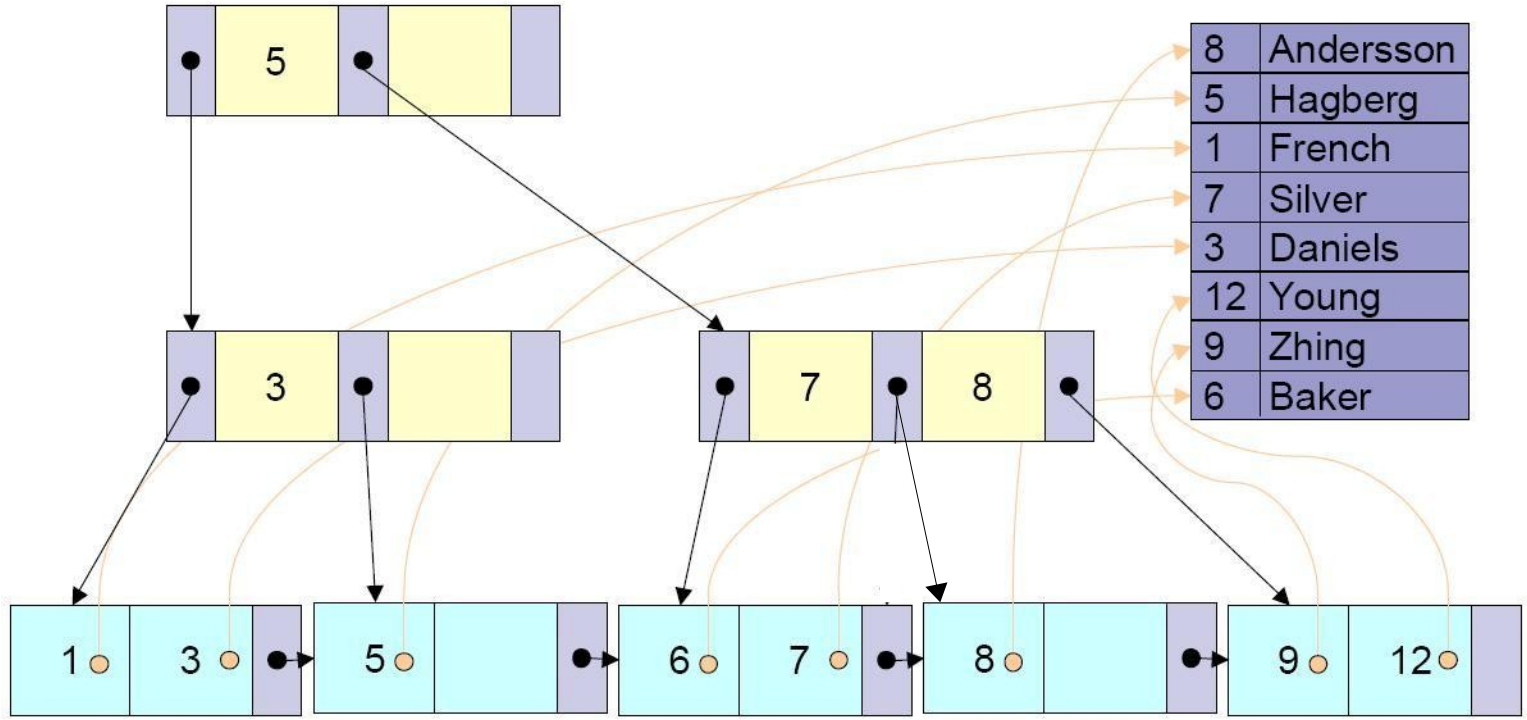
Leaf Nodes of a B$^+$-Tree



- $q \leq p$ (where p is the order for leaf nodes of the B$^+$-tree)
- Every K_i is an index value
- Every Pr_i is a data pointer to the data file block that contains the record with index value K_i
- P_{next} is a pointer to the next leaf node
- Within each node: $K_1 < K_2 < \ldots < K_q$
- Every leaf node must be at least half full
  - i.e., at least $\left\lceil \frac{p}{2} \right\rceil$ index values in each leaf node

Retrieval of Records in a B$^+$-Tree

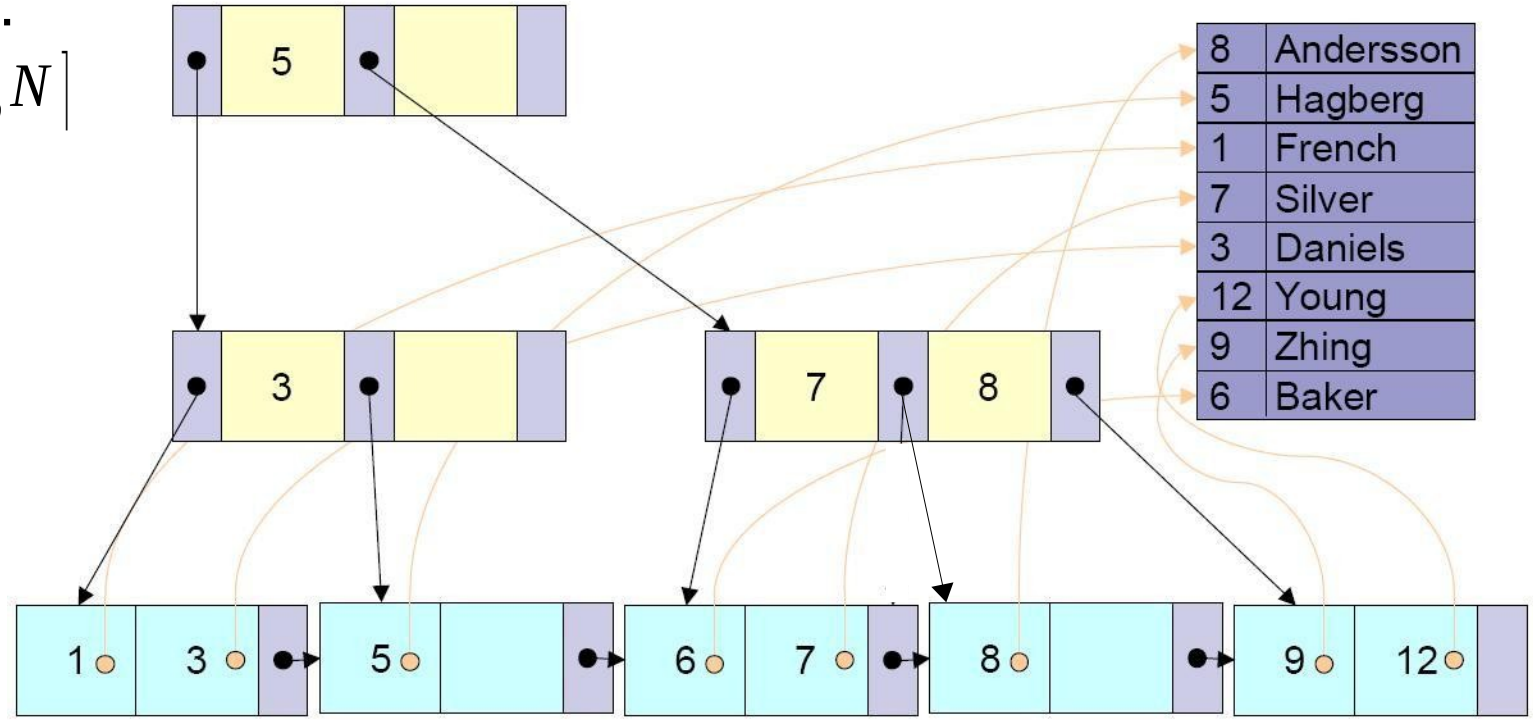
- Very fast retrieval of a random record
- Number of block accesses: depth of tree + 1



Depth of a B$^+$-Tree

- Given that internal nodes must have at least $\left\lceil \frac{p}{2} \right\rceil$ children,
- For a depth of d, the number N of leaf nodes is at least $\left\lceil \frac{p}{2} \right\rceil^d$
- Hence, in the worst case, d is at most $\left\lceil \log_{\left\lceil \frac{p}{2} \right\rceil} N \right\rceil$
- Best case:

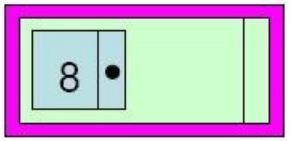
$$\left\lceil \log_p N \right\rceil$$



B+-Tree Insertion

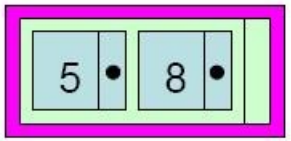
Insert: 8

B+-Tree Insertion



Insert: 5

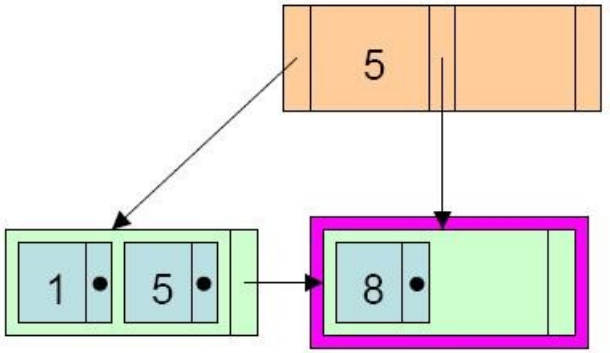
B+-Tree Insertion



Overflow – create a new level

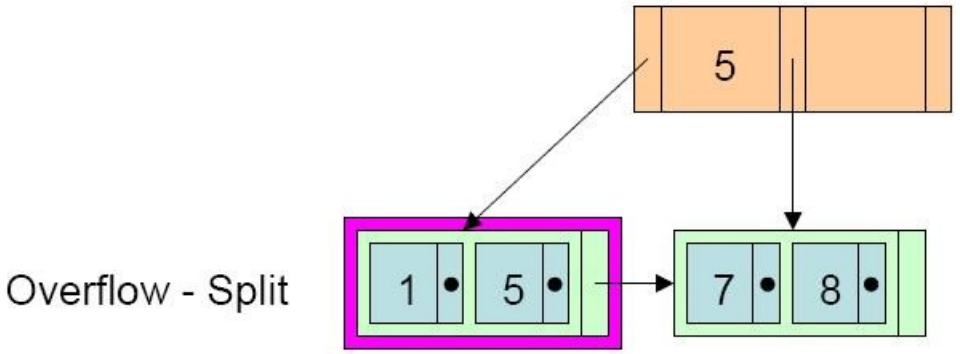
Insert: 1

B$^+$-Tree Insertion



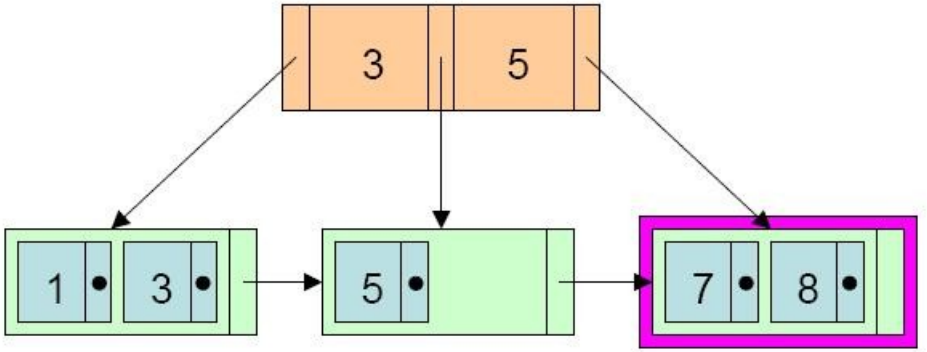
Insert: 7

B+-Tree Insertion



Insert: 3

B$^+$-Tree Insertion

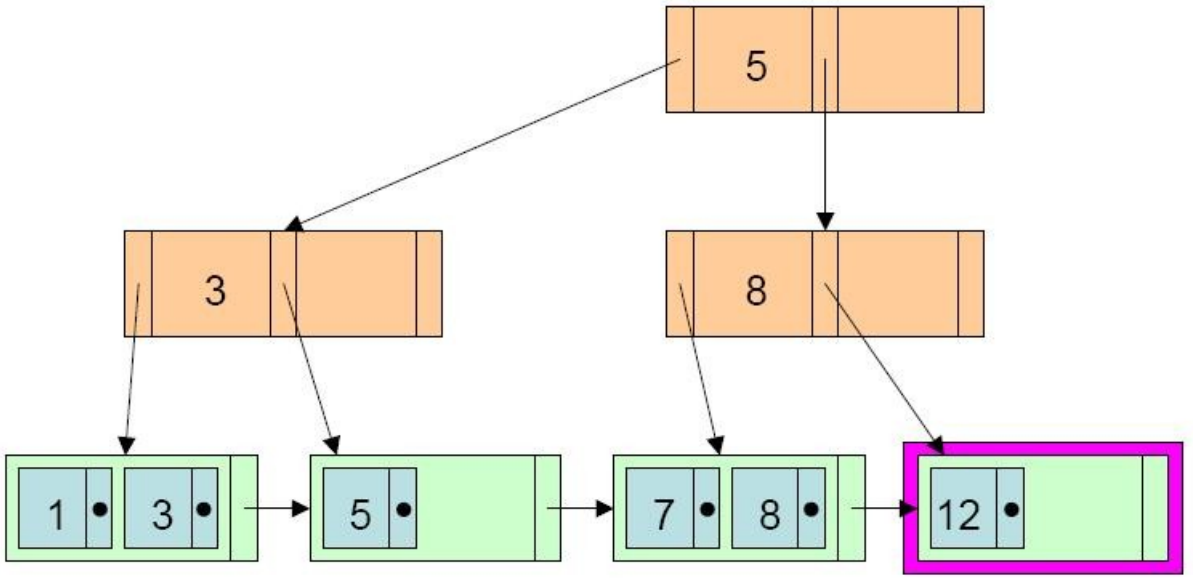


Overflow - Split

Propagates a new level

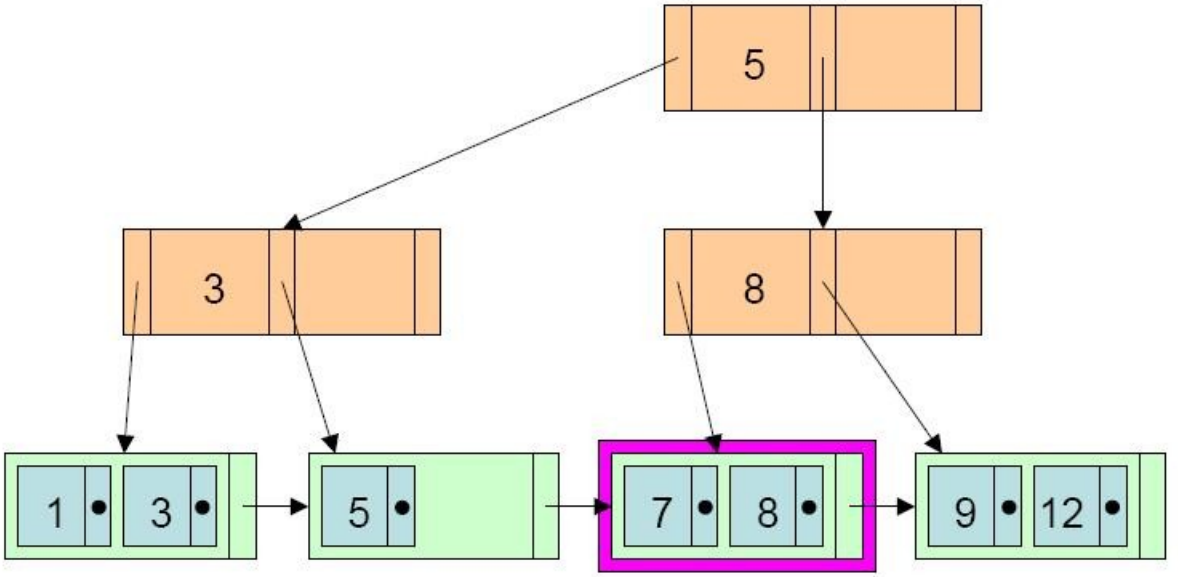
Insert: 12

B+-Tree Insertion



Insert: 9

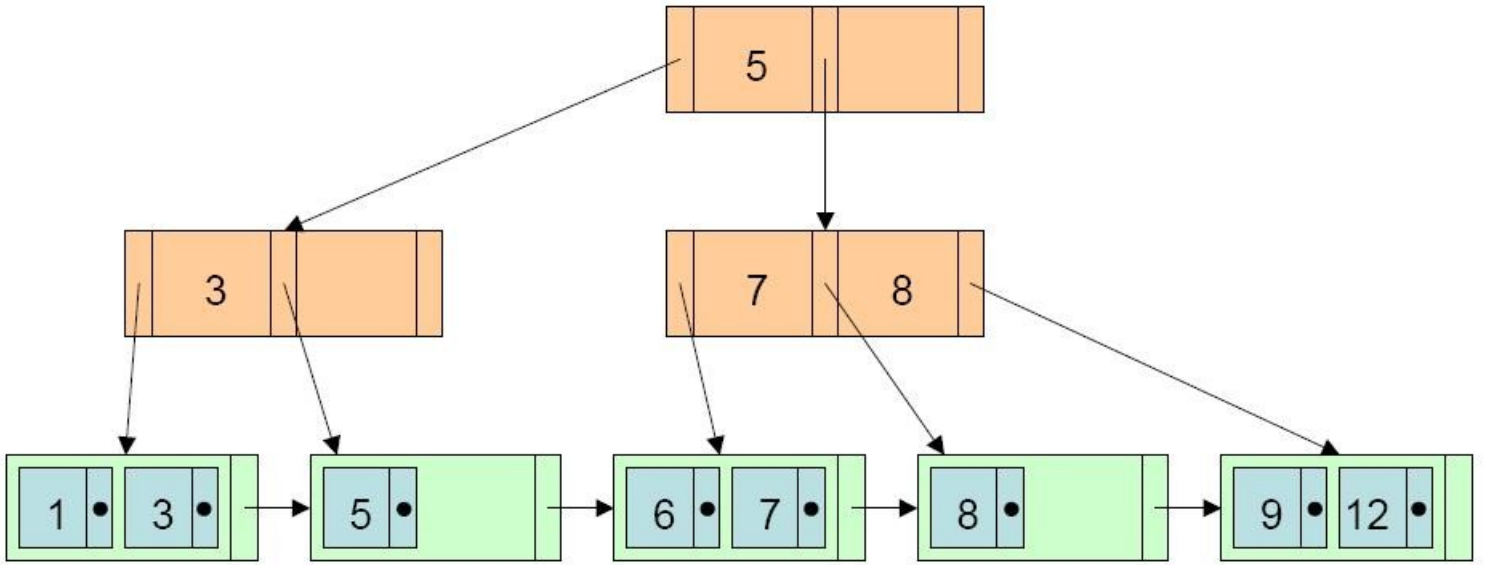
B+-Tree Insertion



Overflow – Split, propagates

Insert: 6

B+-Tree Insertion



Resulting B+-tree

www.liu.se

LINKÖPING UNIVERSITY