NDBI007: PRACTICAL CLASS 2

STATIC INDEXES AND BITMAPS

Based on NDBI007 practical class materials created by Petr Škoda; Tutor: Pavel Koupil; October 27th 2021

IMPORTANT TERMS

- B - page size in bytes
- R - object size in bytes
- n - number of objects
- b - blocking factor, i.e., the number of blocks that fit into a single page
  - Can be computed as $b = \lfloor \frac{B}{R} \rfloor$
- h - height of a tree, that is stored using the blocking factor b
  - Can be computed as $h = \lceil \log_b n \rceil$

INDEX SEQUENTIAL FILE

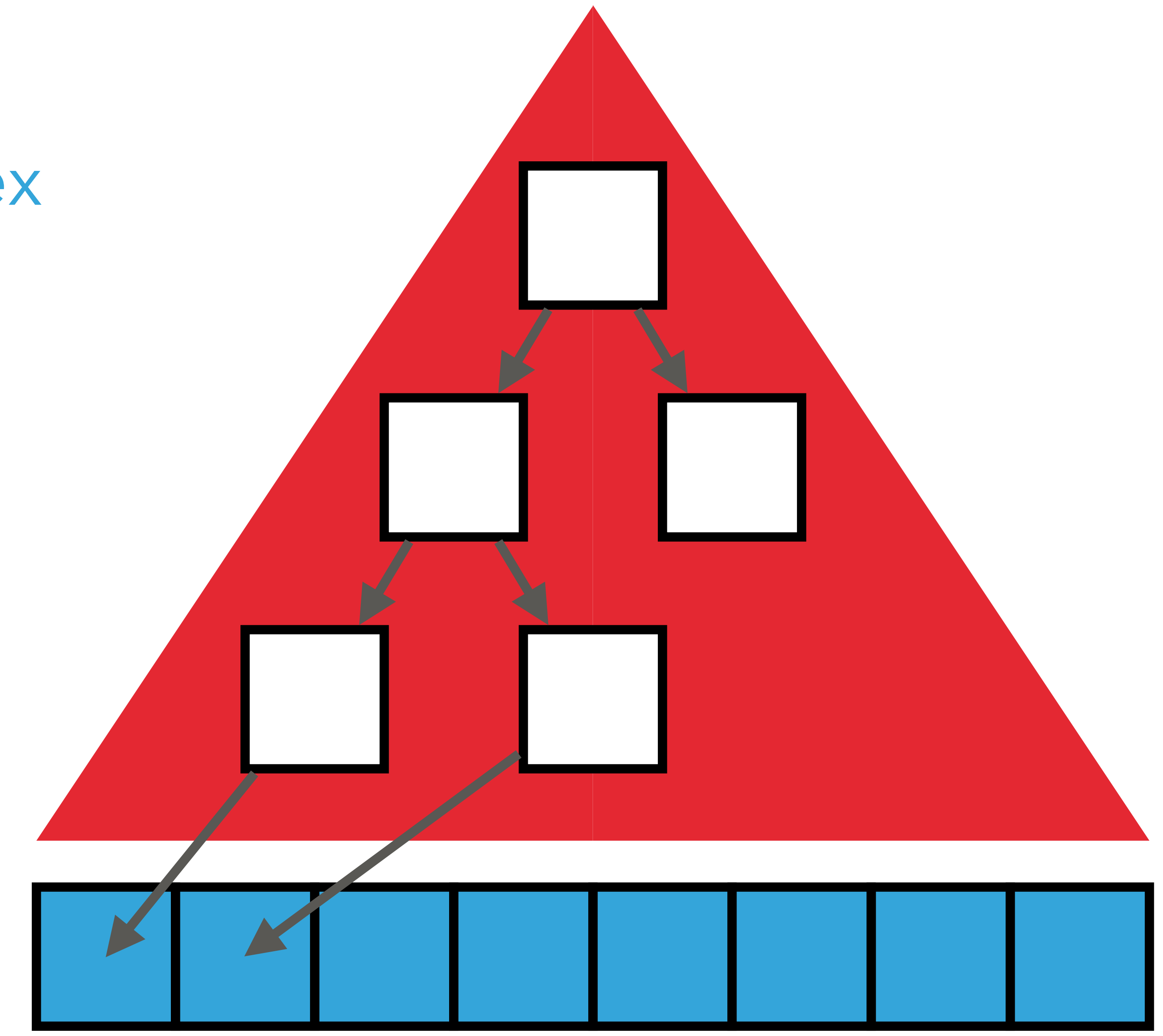
- Consists of at least two files
  - Primary file contains all the data, that are sorted according to a primary key
  - Index file contains the index of the primary key, built over the primary file
- Static index is a hierarchical structure of index pages that contains records of type [value of the primary key; pointer to a page]
- There exist following types of static indexes
  - Primary key, non-primary (secondary) key
  - Direct index
  - Indirect index

INDEX SEQUENTIAL FILE (STRUCTURE STUDENT)

- Consider a database with 5,000,000 records (i.e., student objects)
  - Page size is 4 kB ($4 \bullet 2^{10}$ B)
  - Pointer size is 4 B
- Simplified structure for representation of a student
  - Every property has artificially different size
  - Size of the structure is 229 B, but we round it up to 256 B as we can add additional attributes and it also align the page size

- Student structure:
  - ID 5 B (primary key)
  - first_name 20 B
  - second_name 25 B
  - age 1 B
  - birthday 4 B
  - address 40 B
  - phone_number 9 B
  - note 125 B

PRIMARY KEY INDEX



- In sequential file, we have the primary file sorted based on the primary key*
  - It is exploited in the primary index structure as it enables to omit one level of the index
- The primary key record consists of two values (total size of a record is 9 B)
  - Value of the primary key 5 B
  - Pointer to a page 4 B
- Total size of a record is 9B
  - The size is fixed for all the records
  - Only on the last level of the index the pointers point not to another index page, but to a page of the primary file

* In case of non-sequential file, it is the same as direct index

EXERCISE 1: PRIMARY KEY INDEX

- Build primary key index for a sequential file that contains 5,000,000 student records (of size 256 B)
  - Determine index height and compute the size of every index level
- You will have to compute blocking factor for the primary file in order to determine number of blocks N
  - Remember that the first (bottom) level points directly into the primary file N
- You will have to compute blocking factor for the primary index
  - Suppose page size equal to 4 kB and record size 9 B
- The number of pages on the next level can be computes as $n_{PAGES,L=i} = \lceil \frac{n_{PAGES,L=i-1}}{b} \rceil$

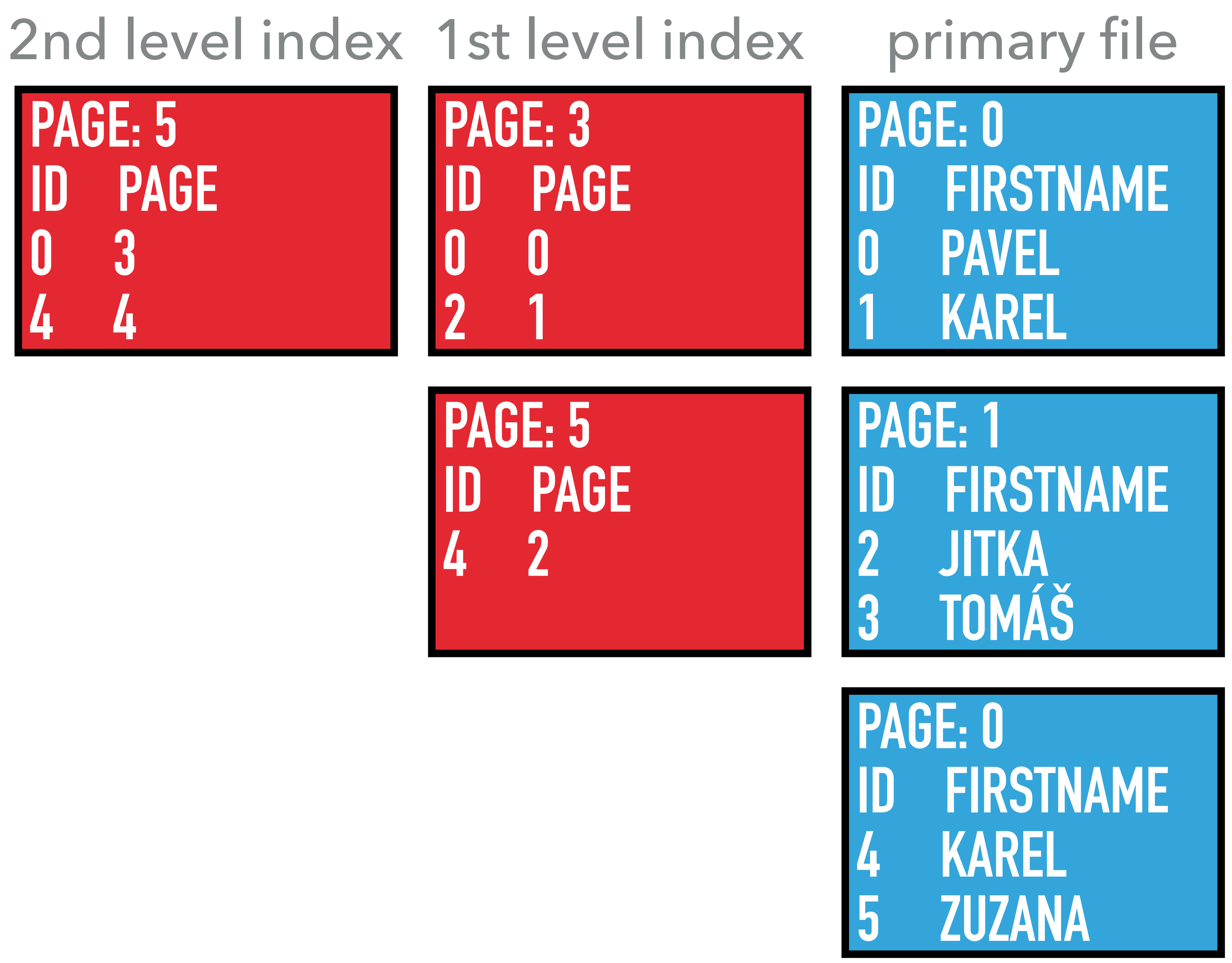
PRIMARY KEY INDEX: ACCESS TO HARD DRIVE

- If the index is stored in external memory, it requires $n + 1$ hard drive accesses to get a record based on a primary key
- The first two index levels are small so we keep them in the main memory to save external memory accesses
  - Therefore, we need only $n - 1$hard drive accesses to retrieve a record
- In real applications, the whole primary index is commonly kept in the primary memory (RAM)
  - The primary key is typically small (4-8 B).
  - The retrieval of a record based on the primary key requires only 1 access to the external memory
  - The presence of primary index in main memory is also utilized by the indirect indexes

PRIMARY KEY DIRECT INDEX

ID	firstName	secondName
0	Pavel	Straka
1	Karel	Zeman
2	Jitka	Nováková
3	Tomáš	Zelený
4	Karel	Svoboda
5	Zuzana	Novotná

- Primary file cannot be sorted by keys of multiple indexes
- The sample depicts the primary key index for the database for ID*
- To see how this structure works we can query for Tomas
  - The query is ID = 3
  - We start at page 5 (index root)
  - Then we go to page 3 (we follow the highest lowest ID value)
  - From page 3 the to page 1 (the same principle as before)
  - We find Tomas on the page 1

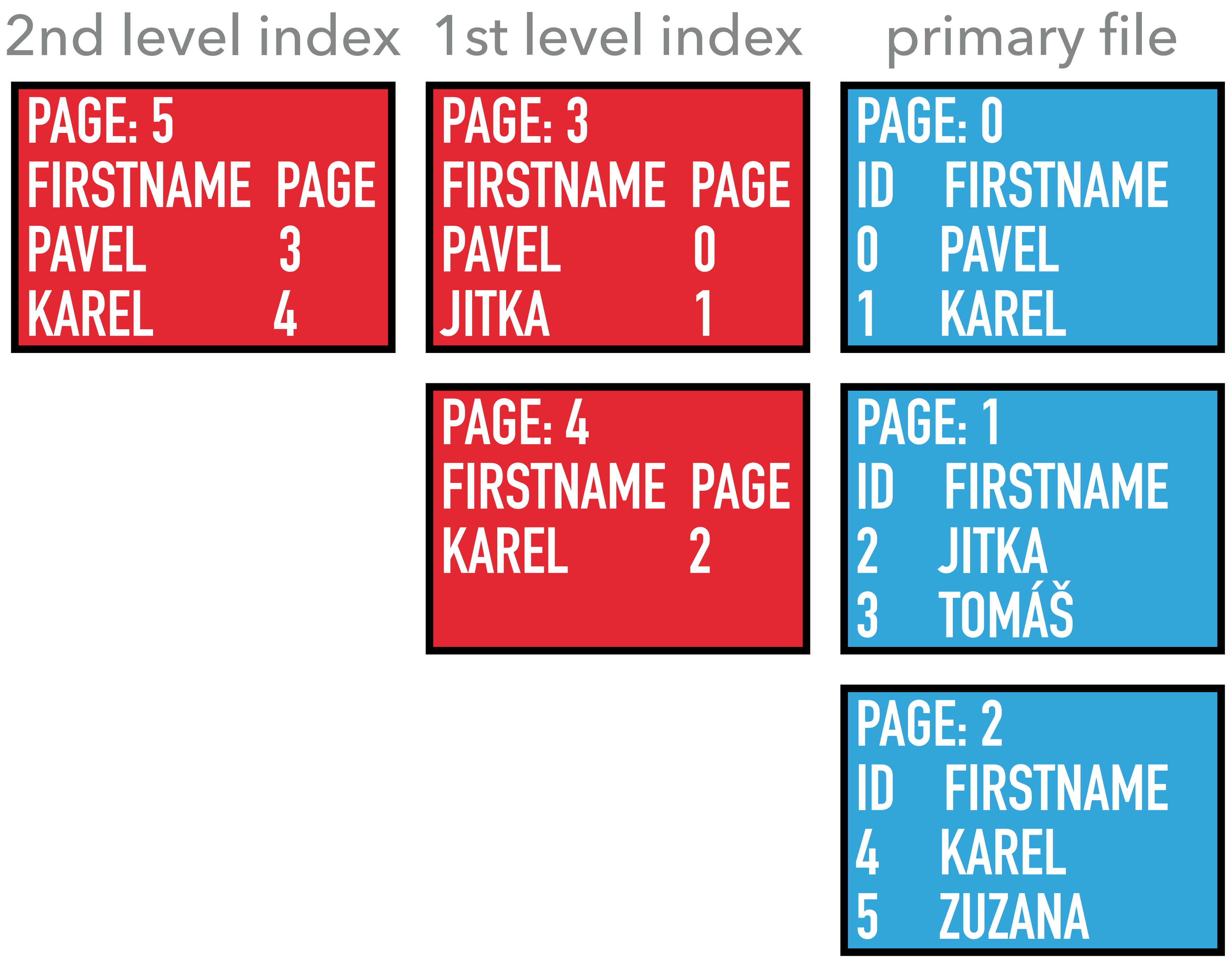


* Note, that we use different page size in pictures just to save space and make picture simpler

NON–PRIMARY KEY DIRECT INDEX

ID	firstName	secondName
0	Pavel	Straka
1	Karel	Zeman
2	Jitka	Nováková
3	Tomáš	Zelený
4	Karel	Svoboda
5	Zuzana	Novotná

- We try to apply the same process to build a direct index for a non-primary key attribute, i.e., firstName
- However, this approach does not work, i.e., **the index is broken**
- It can be easily demonstrated by a simple query for Karel
  - We start at page number 5 (root of the index)
  - Here, we take the largest smaller key, i.e., Pavel, and we go to page 3
  - In page 3, we repeat the same process, this time Jitka is the largest smaller key. Jitka stands for page number 1
  - But in this way we fail to retrieve Karel on page 0



NON-PRIMARY KEY DIRECT INDEX (CORRECT)

ID	firstName	secondName
0	Pavel	Straka
1	Karel	Zeman
2	Jitka	Nováková
3	Tomáš	Zelený
4	Karel	Svoboda
5	Zuzana	Novotná

- Solution: Addition of another level between the index and the primary file
  - I.e., we add zero level index
- Query for Karel once again:
  - We start on page 8 (index root)
  - We continue on page 6 (Jitka) and the on page 3
  - Here, we see the first record for Karel, then we scan the following index pages until we reach a higher key. By this, we get Karel on page 0 as well as Karel on page 2
- The zero level is a copy of given key with pointer to the respective pages. This level is sorted by the key. It's basically a very thin replication of a primary file
- Note: The "zero level index" is also used in case of non-sequential file with index. In case of non-sequential file the primary file is not sorted by any property

2nd level index

PAGE: 8

FIRSTNAME	PAGE
JITKA	6
TOMÁŠ	7

1st level index

PAGE: 6

FIRSTNAME	PAGE
JITKA	3
KAREL	4

PAGE: 7

FIRSTNAME	PAGE
TOMÁŠ	5

0th level index

PAGE: 3

FIRSTNAME	PAGE
JIRKA	1
KAREL	0

PAGE: 4

FIRSTNAME	PAGE
KAREL	2
PAVEL	0

PAGE: 5

FIRSTNAME	PAGE
TOMÁŠ	1
ZUZANA	2

primary file

PAGE: 0

ID	FIRSTNAME
0	PAVEL
1	KAREL

PAGE: 1

ID	FIRSTNAME
2	JITKA
3	TOMÁŠ

PAGE: 2

ID	FIRSTNAME
4	KAREL
5	ZUZANA

EXERCISE 2: DIRECT INDEX

- Build direct index on firstName for a sequential file that contains 5,000,000 student records
  - Suppose that index record size is 20 B + 4 B (size of key + size of the pointer)
  - Determine index height and compute the size of every index level
  - Compare the structure with primary key index structure
    - I.e., number of levels, sizes of levels, total size of index (in MB)

INDIRECT INDEX

- Direct indexing and primary index share one disadvantage
  - In the case of any modification (records shuffling) in the primary file, the first (zero) level must be updated

- The solution is indirect indexing that does not point to the primary file pages
  - It points to the primary keys, i.e., indirect index can be described as a map from some property to a primary key
  - In addition, indirect index does not point to the file directly, therefore it is not affected by modifications of the primary file
  - As the primary index is commonly stored in primary memory (RAM), we just need to read pages for the indirect index and retrieve pages from the primary file
  - The main difference is that the last level points to the primary index

- Although the first level is slightly larger than that of a direct index, the main advantage is that an indirect index does not need to be updated in case of primary file movements

EXERCISE 3: INDIRECT INDEX

- Build indirect index on secondName for a sequential file that contains 5,000,000 student records
  - Note that first level records and other level records differ in its size
    - First level: 25 B + 5 B (second name key size + primary key size)
    - Other level: 25 B + 4 B (second name key size + pointer to another page)
  - Determine index height and compute the size of every index level

SEARCHING IN INDEX FROM MULTIPLE ATTRIBUTES

- Two properties can be concatenated (e.g., firstName and secondName)
  - Enables us to search for both of the attributes at once
  - Attribute ordering in the index is fixed
    - E.g., firstName followed by secondName does not allow us to search for secondName followed by firstName

BITMAPS

- Note: Having 50 percent men and 50 percent women in our database, usage of previous indices is not effective at all
  - We prefer bitmaps with database sequential scan over hierarchical index

- Bitmap consists of multiple columns
- Each column is stored in separate page
  - Pages are stored sequentially, allowing effective reading
- A value of a given column is represented by a single bit
  - E.g., having page size 4 kB, we can store 4,096 * 8 = 32,768 values in a single page
  - Useful for attributes having small domain, e.g., traditional concept of gender (male, female)
- Bitmaps allow effective evaluation of logical operations over columns (true = 1, false = 0)
- Based on the value distribution we may also consider some compression (i.e., RLE compression*)

ID	isMale	isFemale
0	1	0
1	1	0
2	0	1
3	1	0
4	1	0
5	0	1

* https://en.wikipedia.org/wiki/Run-length_encoding

EXAMPLE: BITMAPS FOR BIRTHDAYS

ID	01/01	...	07/03	...	31/12
0	1	...	0	...	0
1	1	...	0	...	0
2	0	...	1	...	0
3	0	...	1	...	0
4	1	...	0	...	0
5	0	...	0	...	1

- Birthday (day, month) can be represented in different way using bitmap
- One column for each day in year
  - Positives:
    - One column is read to get all people having birthday in a certain day
    - We can easily add information about other important day for a price of just another single column
  - Negatives:
    - Bitmap takes a lot of space*, i.e., $366 \bullet 153 \bullet 4\ kB = 218.7\ MB$
      - Compression may decrease the size but read time increases as we need to decompress bitmap

* We consider database having 5,000,000 records, therefore $5{,}000{,}000 \div 32{,}768 = 153$ pages are required to store single column

EXAMPLE: BITMAPS FOR BIRTHDAYS

- Two sets of bitmaps, one for days (31) and other for months (12)
  - We need a single AND operation to read this
  - Positives:
    - Smaller size, i.e., $43 \bullet 153 \bullet 4\ kB = 25.7\ MB$
  - Negative:
    - We have to read two columns to get information about birthdays in a given day

day

ID	1	...	31
0	1	...	0
1	1	...	0
2	0	...	0
3	0	...	0
4	1	...	0
5	0	...	1

month

ID	1	...	31
0	1	...	0
1	1	...	0
2	0	...	0
3	0	...	0
4	1	...	0
5	0	...	1

EXAMPLE: BITMAPS FOR BIRTHDAYS

- Binary representation of a day in a year
  - Number 366 can be saved into 9 bits
  - E.g., 01/01 = 000 000 001, 02/01 = 000 000 010, 01/02 = 000 100 000
  - Positives:
    - Much smaller size, i.e., $9 \bullet 153 \bullet 4\ kB = 5.4\ MB$
  - Negatives:
    - We have to read all columns to find all birthdays in a certain day

ID	9	...	3	2	1
0	1	...	0	1	0
1	1	...	0	1	1
2	0	...	1	0	0
3	0	...	1	0	1
4	1	...	1	1	0
5	0	...	1	1	1