Engineering Databases

Lecture 8 –Indexing, and Complexity

January 18, 2023

M. Saeed Mafipour & Mansour Mehranfar

Content of Lecture 7

- Normalization improves relation database design in a formal way
- Normalizations avoid update, insert and delete anomalies
- Using the ER-Diagram mostly generates normalized tables
- NF1: all attributes must be atomic
- NF2: all non-key attributes are fully functional dependent on all key attributes
- NF3: No non-key attribute is transitively dependent on a non-key attribute
- BCNF: removes any ambiguous keys

Indexing

- Simple analogy
  - Browsing a book
  - Find the data

- General problem
  - `SELECT * FROM Products WHERE ProductID = 123456;`
  - Each tuple has to be retrieved from the disk *
  - For each tuple the ProductID attribute has to be tested against 123456
  - Large Database = Millions of tuples → Query takes too long

Complexity

- How efficient is an algorithm or piece of code?
  - Efficiency covers lots of resources:
    - CPU (time) usage
    - Memory usage
    - Disk access

- An algorithm's complexity
  - How do the resource requirements of an algorithm scale?
  - What happens as the size of the problem gets larger?

Complexity

- We are not interested in the exact number of operations
- We are interested in the relation of the number of operations to the problem size

- Typically, we are interested in the worst case, the maximum number of operations
- Big-O notation

  For a problem of size N:

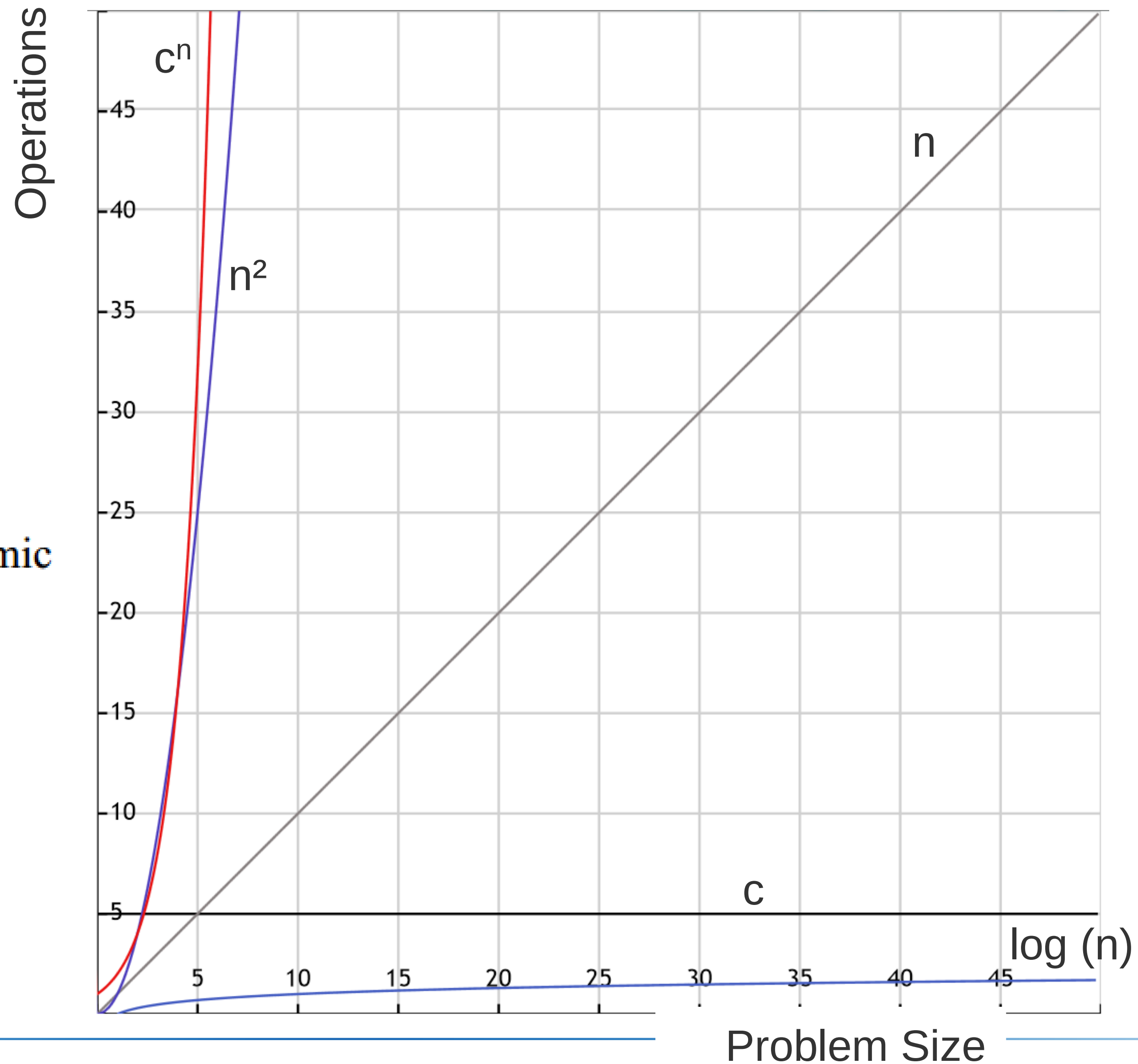
  - a constant-time algorithm is "order 1":
  - a linear-time algorithm is "order N":
  - a quadratic-time algorithm is "order N squared":

notation	**name**
$O(1)$	constant
$O(\log(n))$	logarithmic
$O((\log(n))^c)$	polylogarithmic
$O(n)$	linear
$O(n^2)$	quadratic
$O(n^c)$	polynomial
$O(c^n)$	exponential

Complexity

Big-O notation

notation	name
$O(1)$	constant
$O(\log(n))$	logarithmic
$O((\log(n))^c)$	polylogarithmic
$O(n)$	linear
$O(n^2)$	quadratic
$O(n^c)$	polynomial
$O(c^n)$	exponential



Indexing

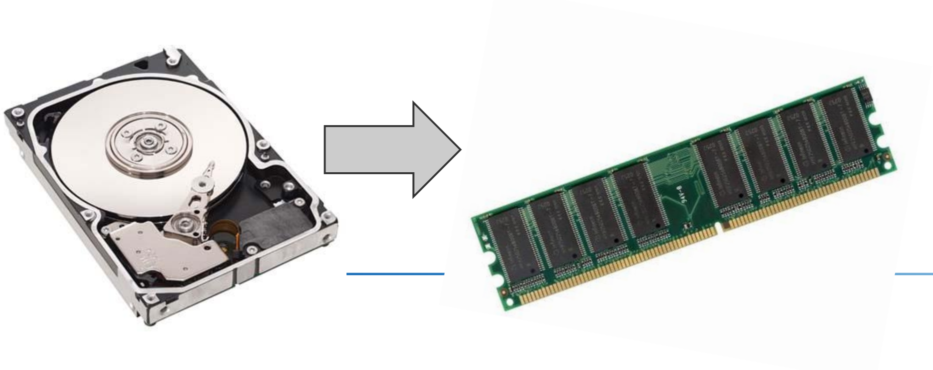
How to improve the query time (the algorithms complexity in Big-O)

Apply an index key

- One attribute, for which the search should be optimized
- e.g. Name in Instructors
- Mostly, but not necessarily the primary key of the relation

Think about the page

- Standard disk space unit
- Minimum amount of data that is read from disk to main memory
- Varies from system to system



Indexing

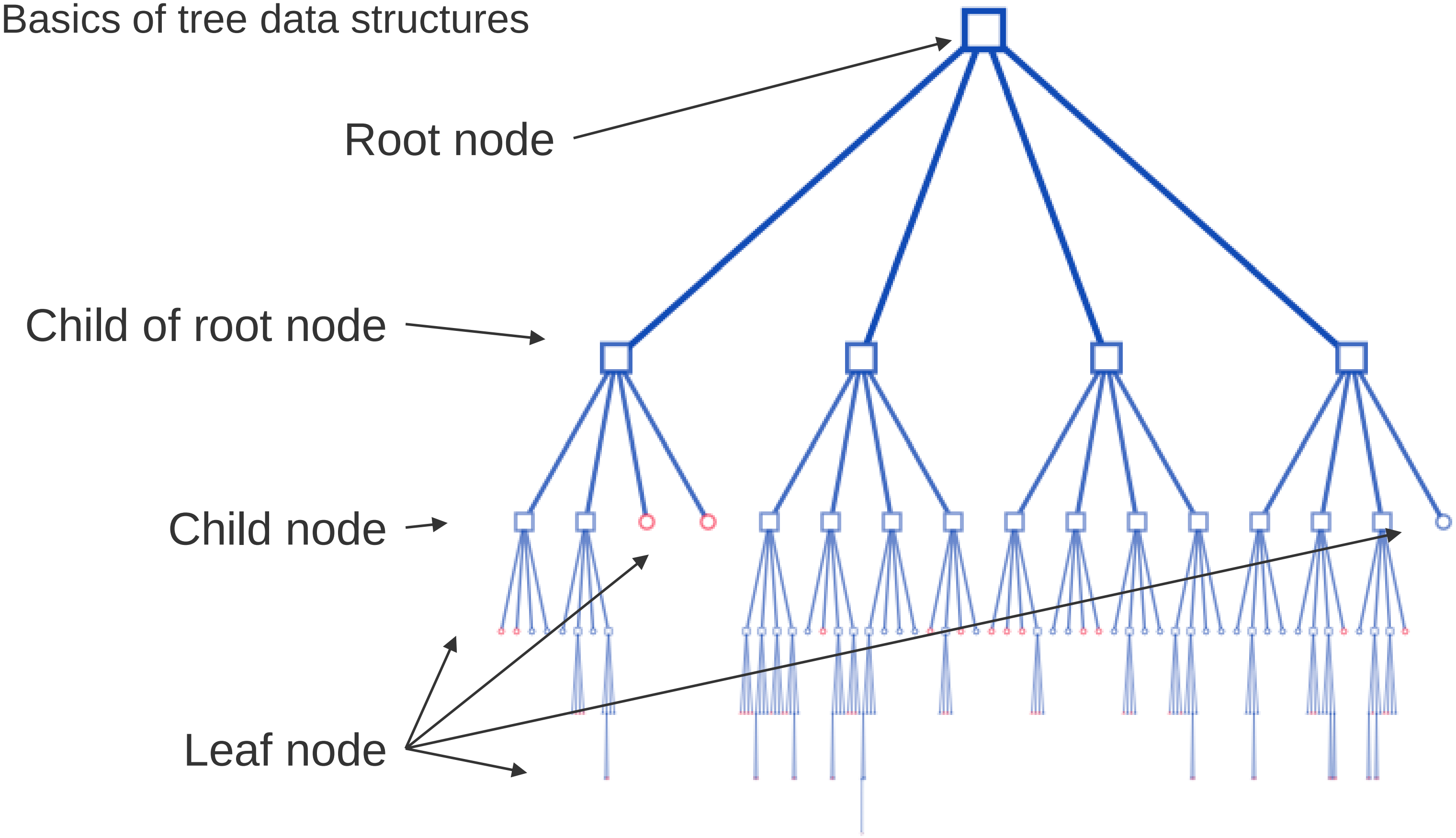
Solution to handle time complexity for database queries

- Allow to limit the comparisons to a small set of tuples
- Different types of index structures:
  - ISAM, B-Tree, B+Tree, hashing indices
- There can be multiple indices for a single relation
  - primary / secondary index

- But indices are not for free
  - Higher effort for insertion and update operations
  - Additional disk space required

Trees

- Basics of tree data structures



Indexing ISAM

- Method: Index Sequence Access Method (ISAM)

Custom
1000
2000
3000
4000
5000

CustomerID	Name
1000	Augustiner
1027	Auer
1290	Paulaner
1313	Spaten
2000	Eder
2132	Grimm
2323	Stacheter

CustomerID	Name
1000	Augustiner
1027	Auer
1290	Paulaner
1313	Spaten

CustomerID	Name
2000	Eder
2132	Grimm
2323	Stacheter

For finding a Name here, the worst case number of compare operations is: N

The worst case number of compare operations: M + K << N

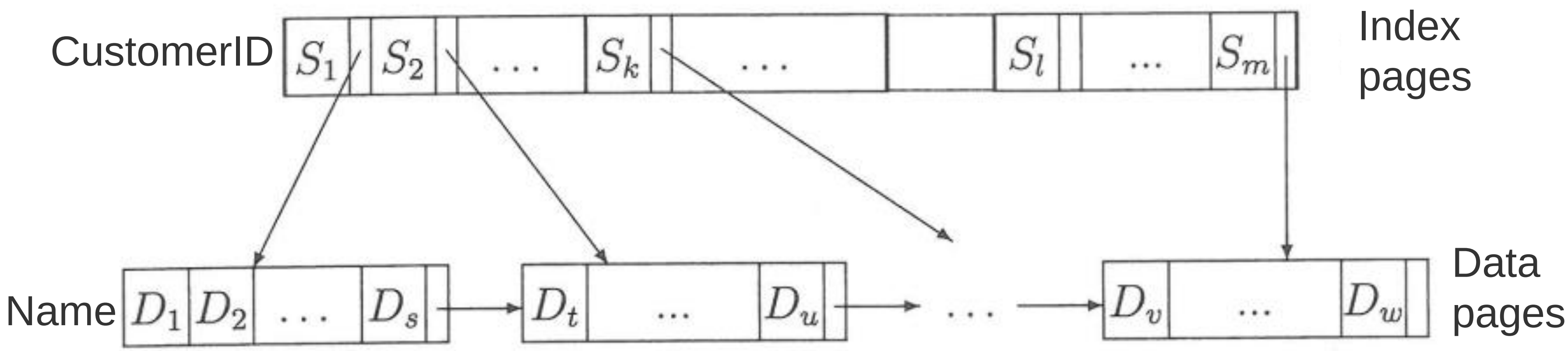
Indexing ISAM

- Very similar to dictionaries
  - First, choose a range
  - Then, search line-wise

CustomerID
1000
2000
3000

CustomerID	Name
1000	Augustiner
1313	Spaten

CustomerID	Name
2000	Eder
2323	Stacheter

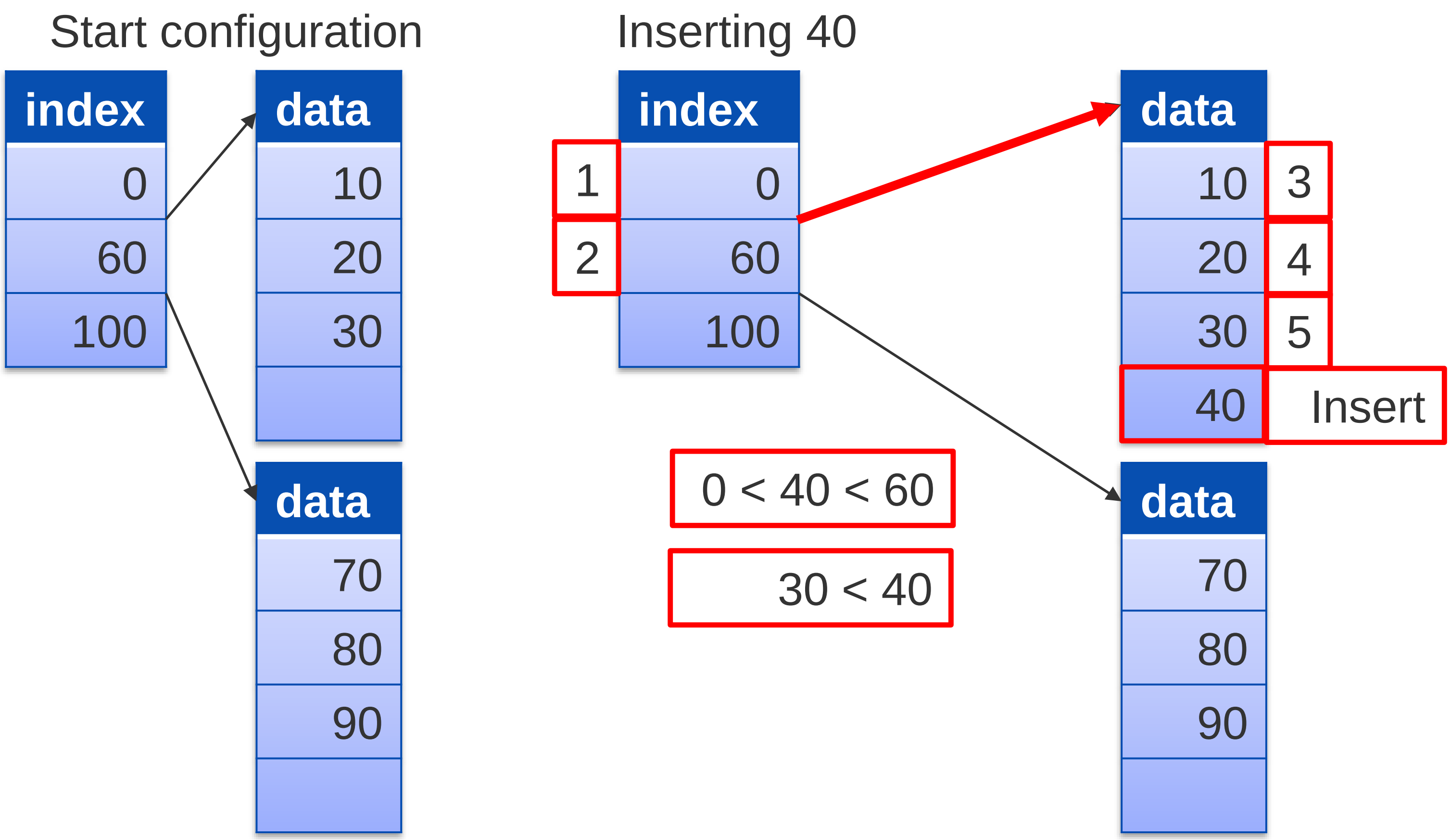


- Store keys S_i and corresponding pointers in the index pages
- Pointer between S_i and S_{i+1} points to a page with key is > S_i and <= S_{i+1}

Indexing ISAM

Data page size is 4

- ISAM Example



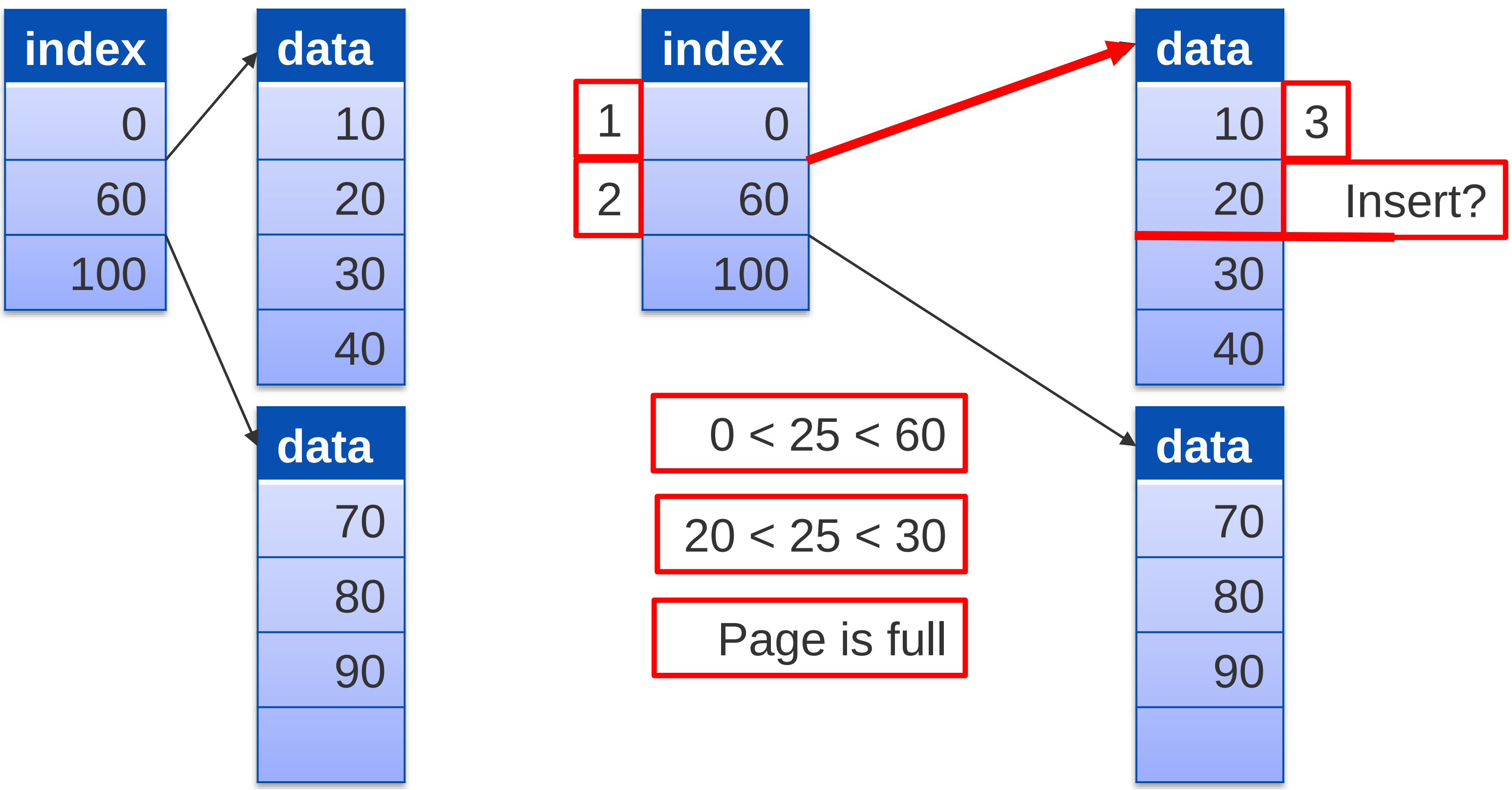
Indexing ISAM

Data page size is 4

- ISAM Example

Start configuration

Inserting 25



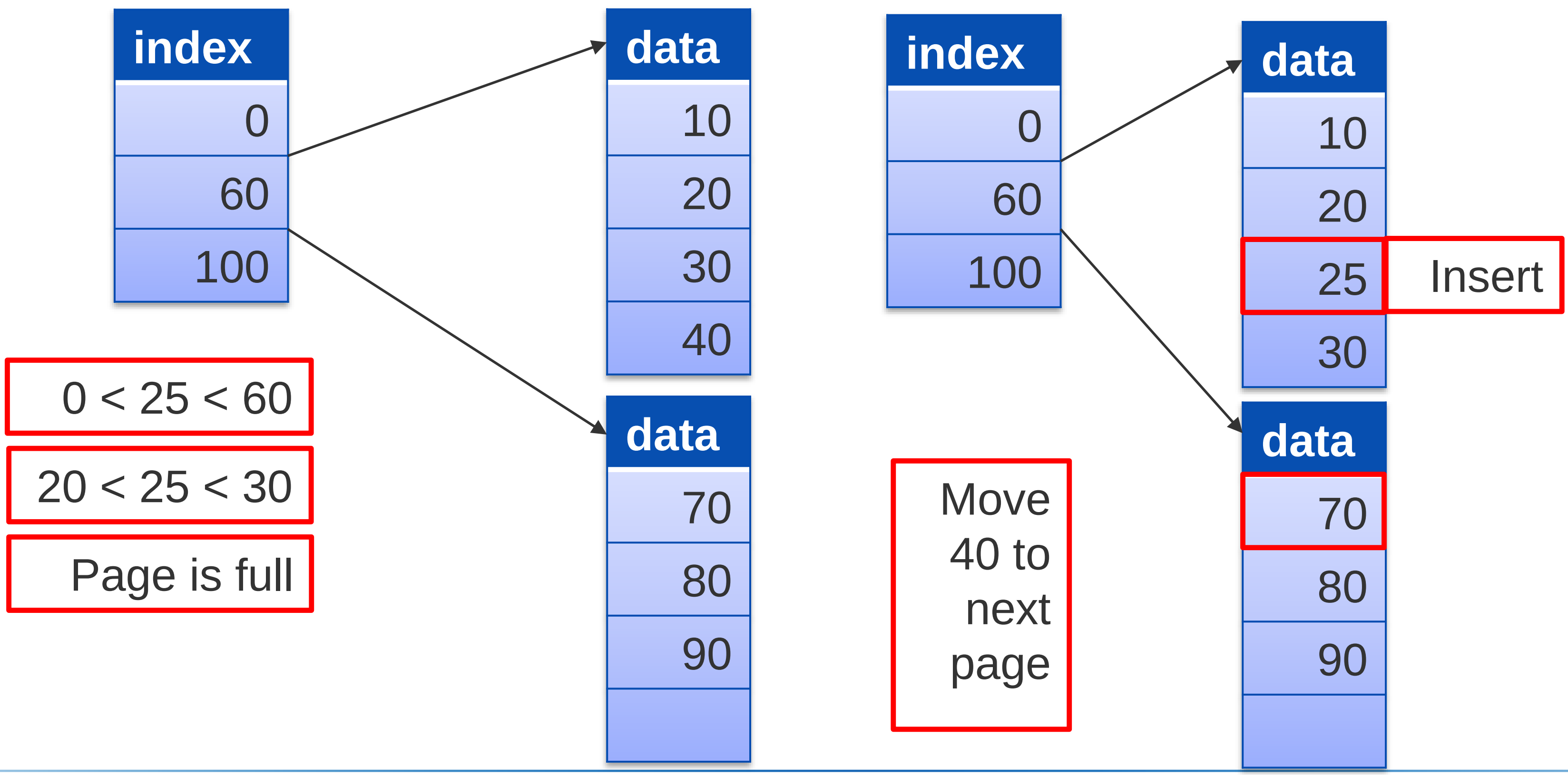
Indexing ISAM

Data page size is 4

- ISAM Example

Inserting 25

Rearrange index and data. The ranges and page size have to be valid!



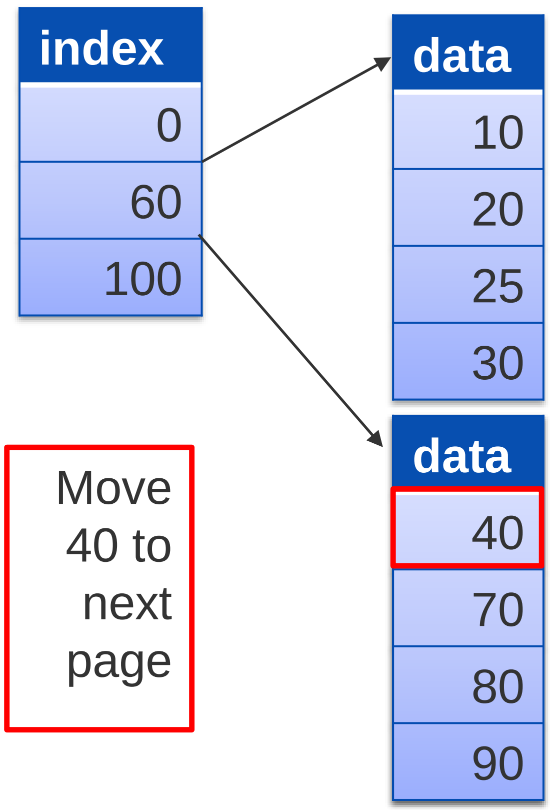
Indexing ISAM

Data page size is 4

Rearrange index and data. The ranges and page size have to be valid!

- ISAM Example

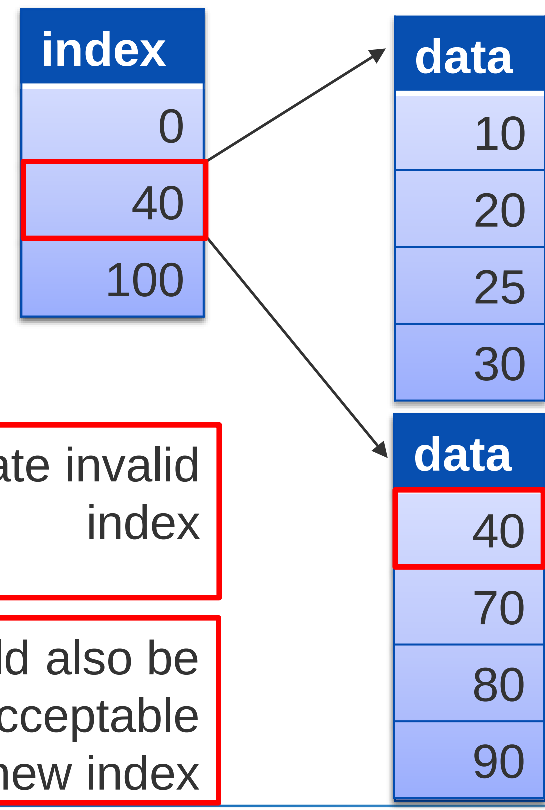
  Inserting 25



Move 40 to next page

Update invalid index

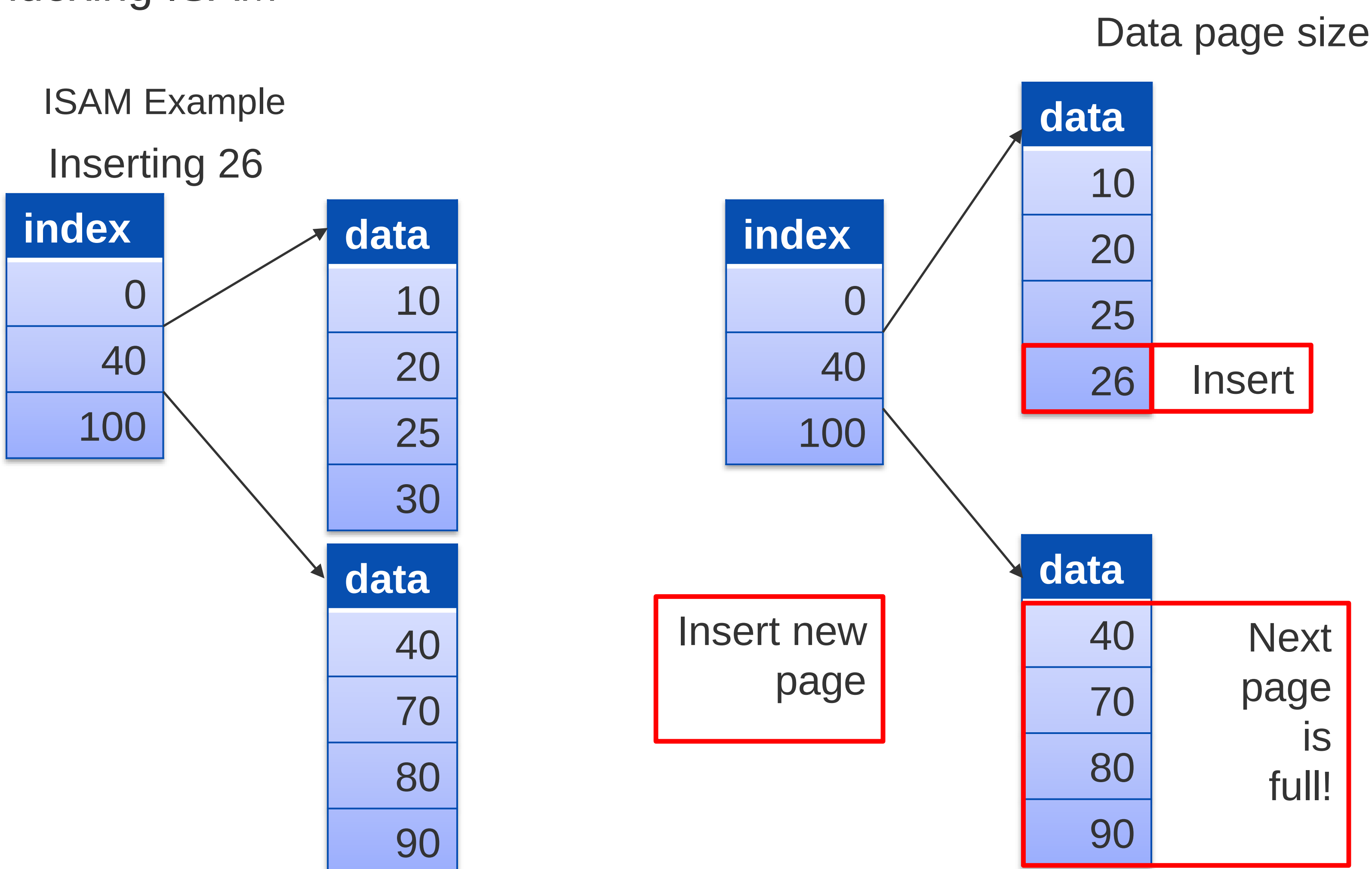
35 would also be an acceptable new index



Indexing ISAM

- ISAM Example

Inserting 26

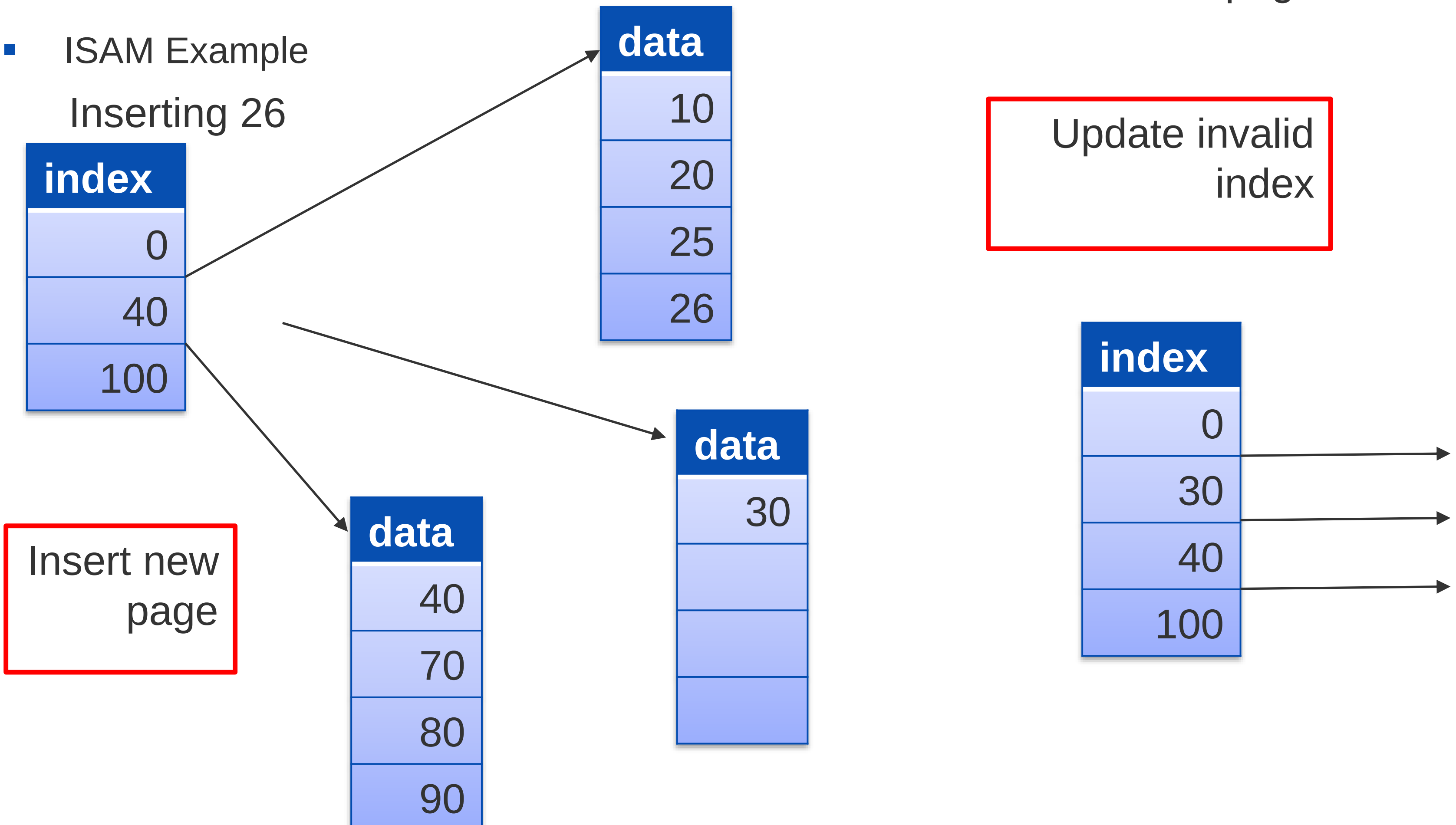


Indexing ISAM

Data page size is 4

- ISAM Example

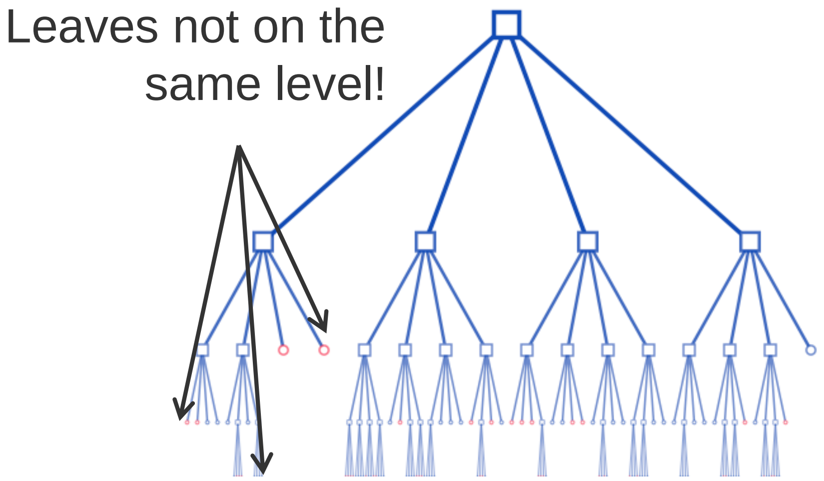
  Inserting 26



Indexing ISAM

- Parameter
  - Size of a data page
- Advantages
  - Relatively simple
  - Great for true sequential access
- Disadvantages
  - Insertion can be very laborious if the page is full
  - Inefficient for lots of overflow pages
  - Does not allow transactions
    Entire tables are locked instead of rows

Indexing B-Tree

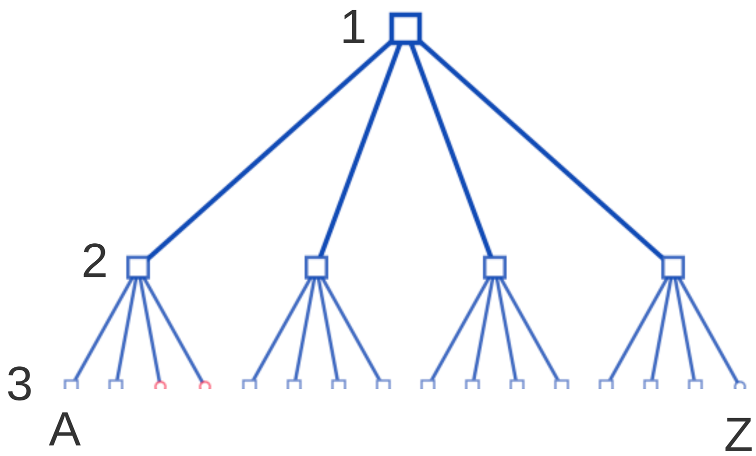


- B-Trees are balanced
  Leaves are on the same depth
  This makes the B-Tree efficient!

- Maximum number of disk accesses
  is limited by height of the tree

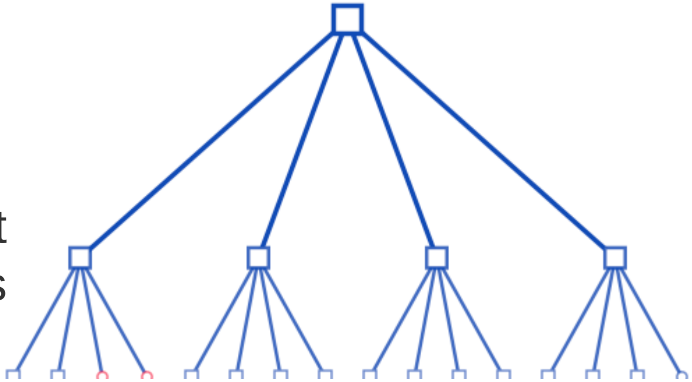
- Same number of operation for each search
  E.g. search for “Alfred” takes as
  long as for “Zeppelin”

- Search complexity is O(log n)



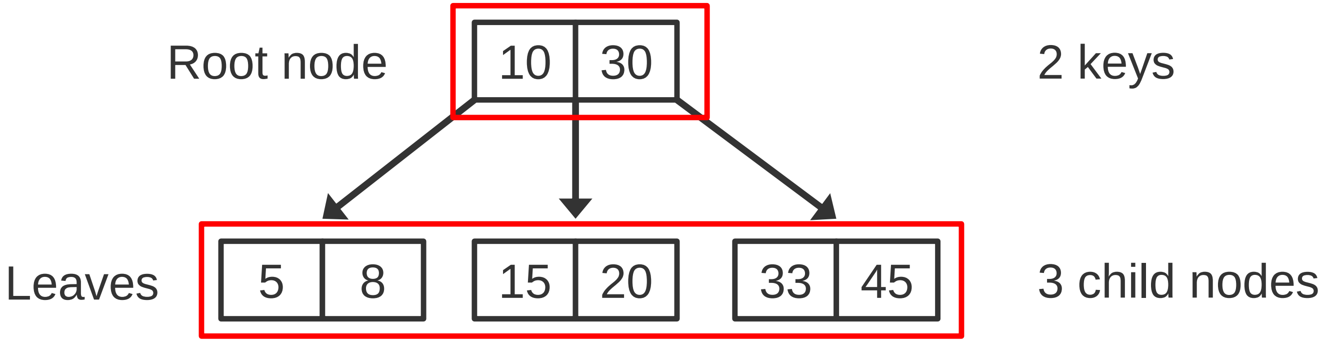
Indexing B-Tree

- All leaves are on the same depth (level).



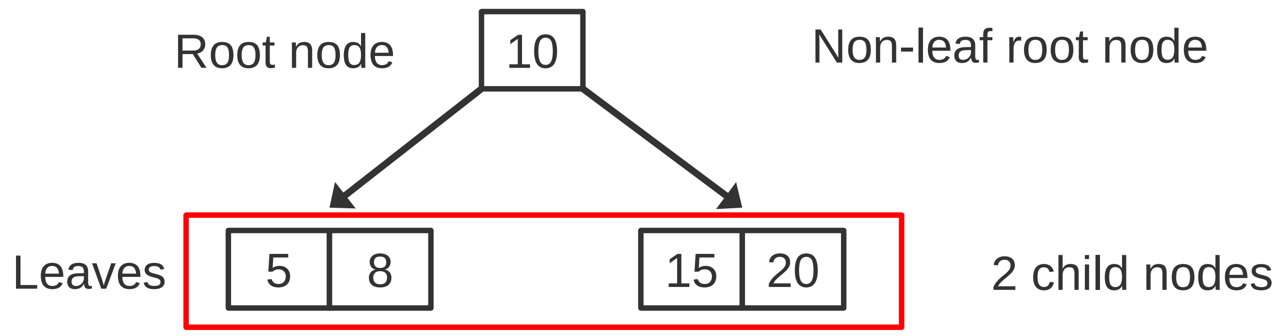
Indexing B-Tree

- All keys in a node are in ascending order.
- A non-leaf node with **n-1** keys must have **n** child nodes.



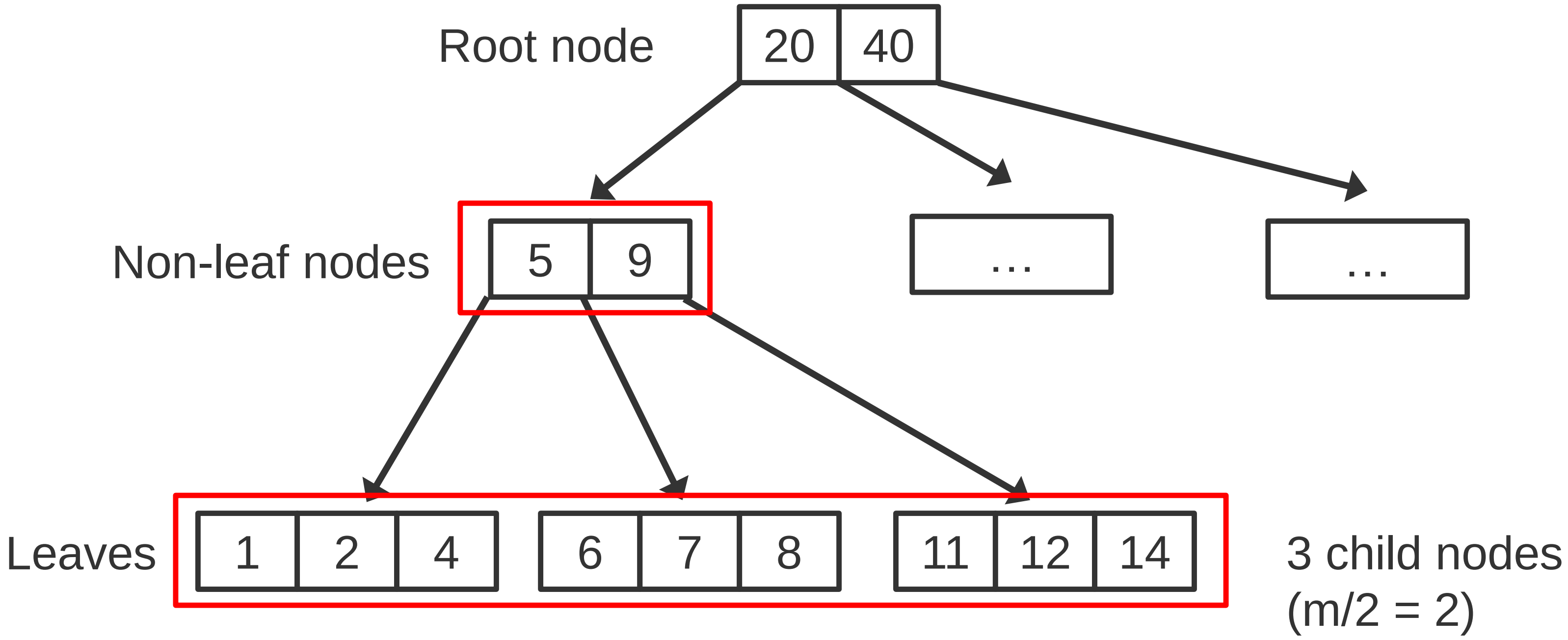
Indexing B-Tree

- If the root node is a non-leaf node, it must have at least **2** child nodes.



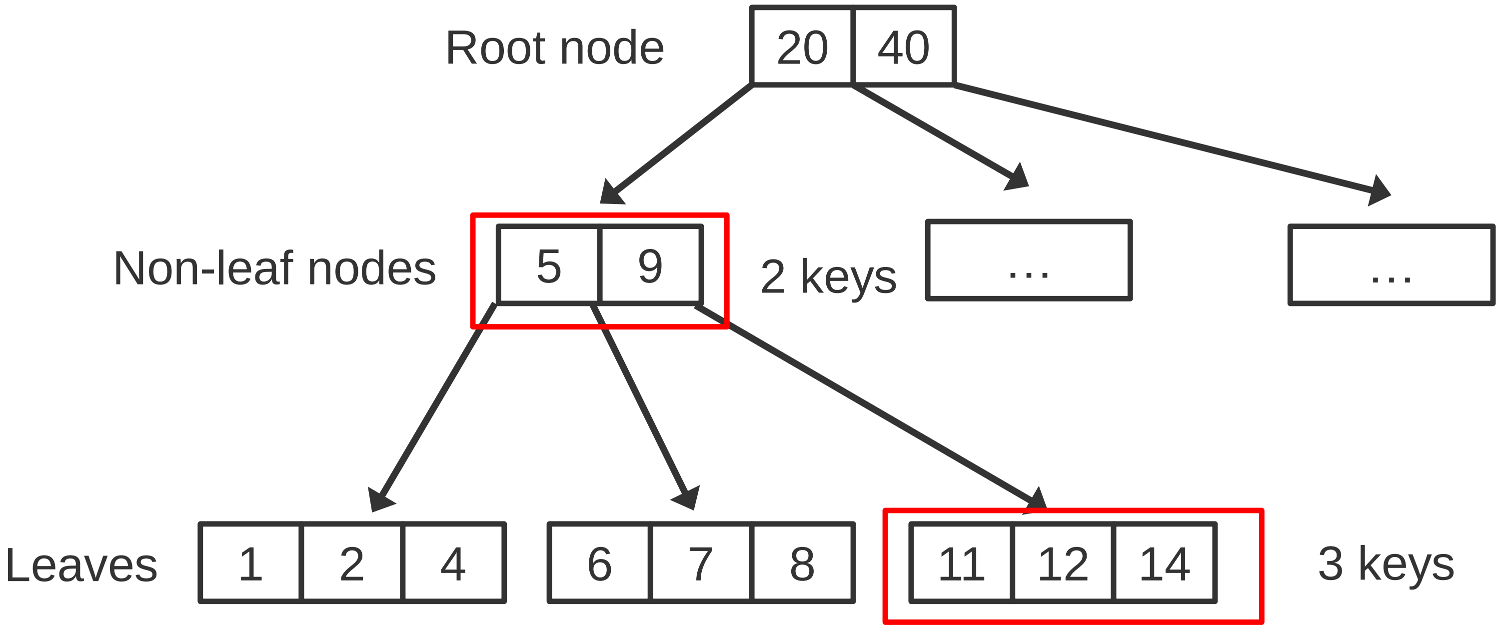
Indexing B-Tree

- Let m = 4
- A non-root non-leaf node must have at least **m/2** child nodes.

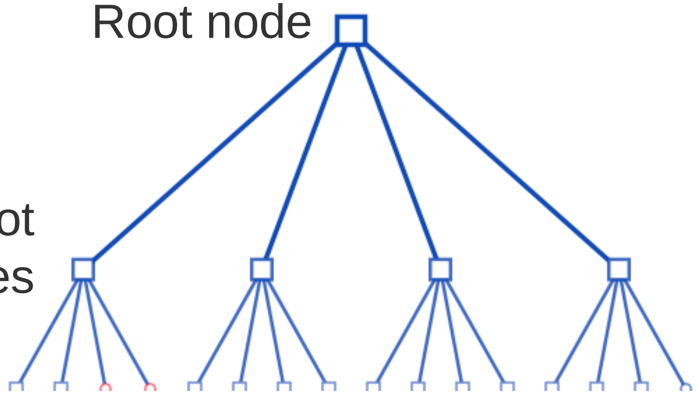


Indexing B-Tree

- A node must have at least **m/2-1** keys (except the root node) and maximal **m-1** keys.
- Let m = 4. Thus, m/2-1 = 1 and m-1 = 3



Indexing B-Tree



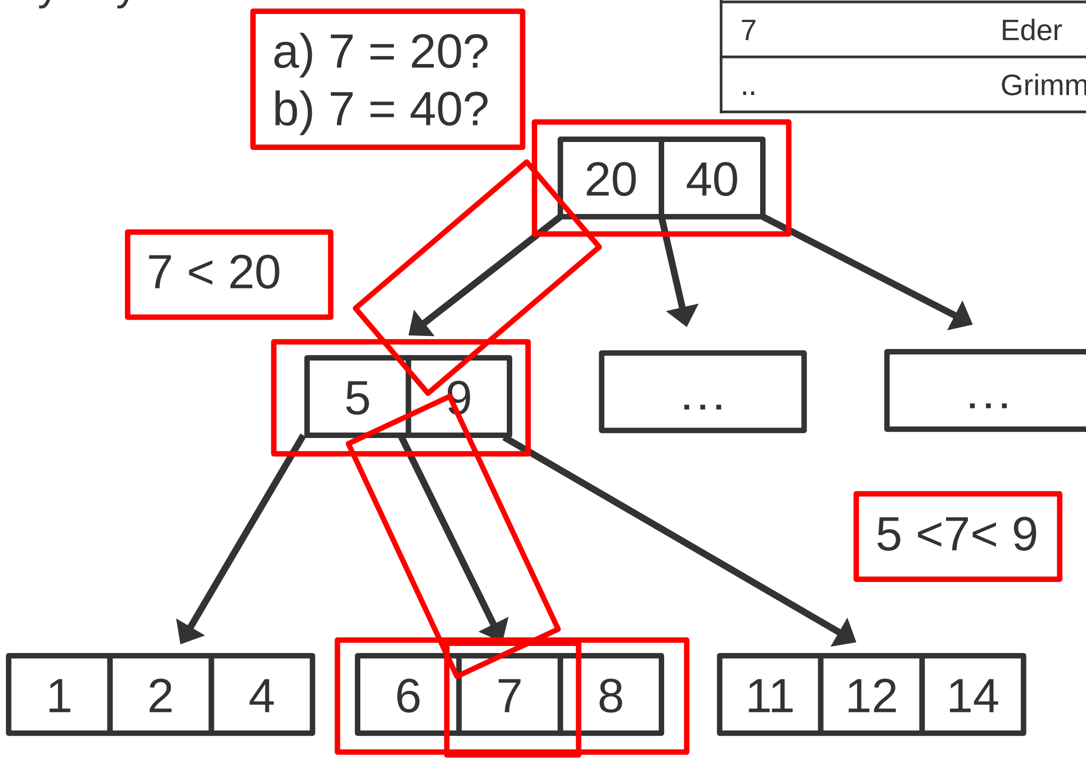
- Parameter:
  Order m ≥ 3
- All leaves are on the same depth (level).
- All keys in a node are in ascending order.
- A non-leaf node with **n-1** keys must have **n** child nodes.
- If the root node is a non-leaf node, it must have at least **2** child nodes.
- A non-root non-leaf node must have at least **m/2** child nodes.
- A node must have at least **m/2-1** keys (except the root node) and maximal **m-1** keys.
- Hint: if e.g. m = 3 and $\frac{3}{2} - 1 = 0.5$, we use the ceiling method. Thus, $\left\lceil \frac{3}{2} - 1 \right\rceil = 1$

CustomerID	Name
20	Augustiner
5	Auer
50	Paulaner
1	Spaten
7	Eder
..	Grimm

Indexing B-Tree Search Algorithm

- We look for the value stored by key 7

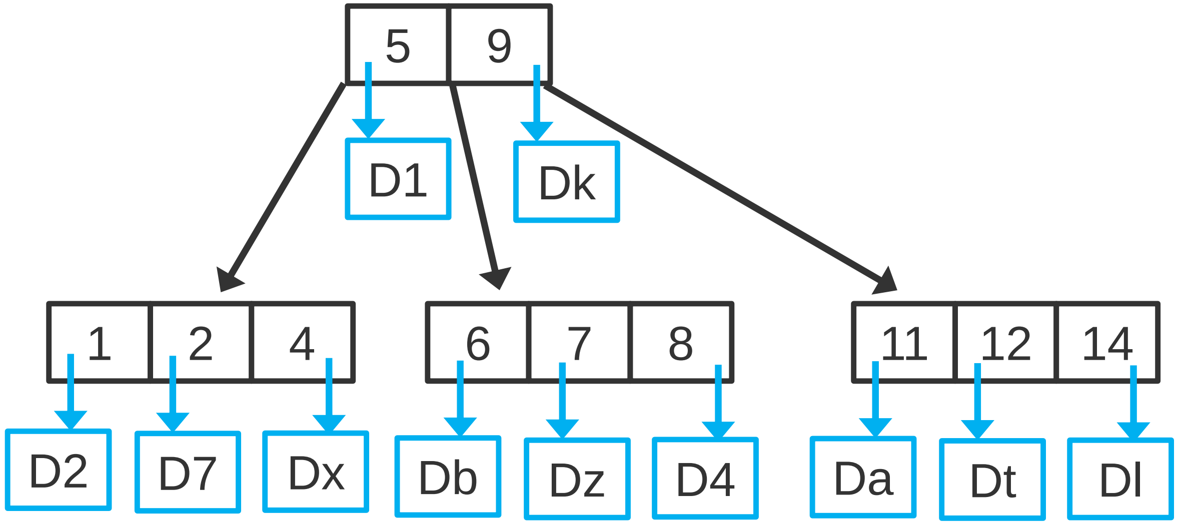
1. Start at the root node
2. Check if the key is in the current node
3. If found, stop.
4. Not found, follow the pointer in range regarding the node's keys.
5. Repeat 2 – 4 until no child node exist or the key is found.



Read data (7,Eder) from the table for key 7.
7 points to a memory position.

Indexing B-Tree Data Pointers

- In B-Trees, each key points to a data record (tuple, row).
- The real structure looks like this:



Indexing B-Tree Insert Algorithm

1. If the tree is empty,
create a root node with the key.

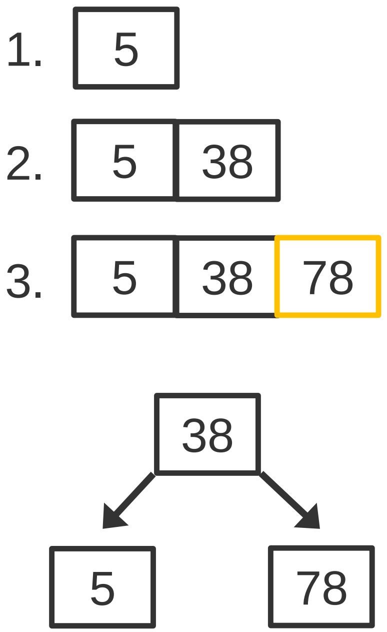
2. If the tree is not empty,
use search to find the node
where to put in the new key value.

3. If the node is not full
(maximal **m-1** keys), add the key.

4. If the node is full, split the node.

Split:
Put the middle key to the parent node.
Move other keys to new or existing nodes.
Repeat this for the parent node if it is full.

Let m=3, min: 3/2-1=1, max = m-1=2



Tree empty

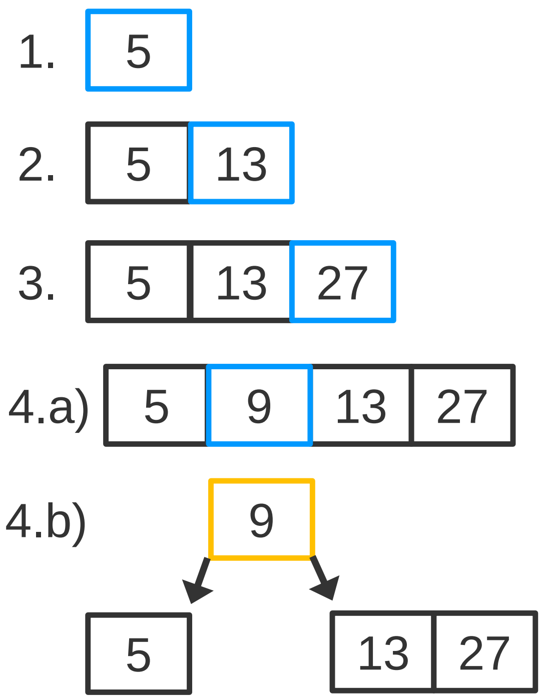
Not empty

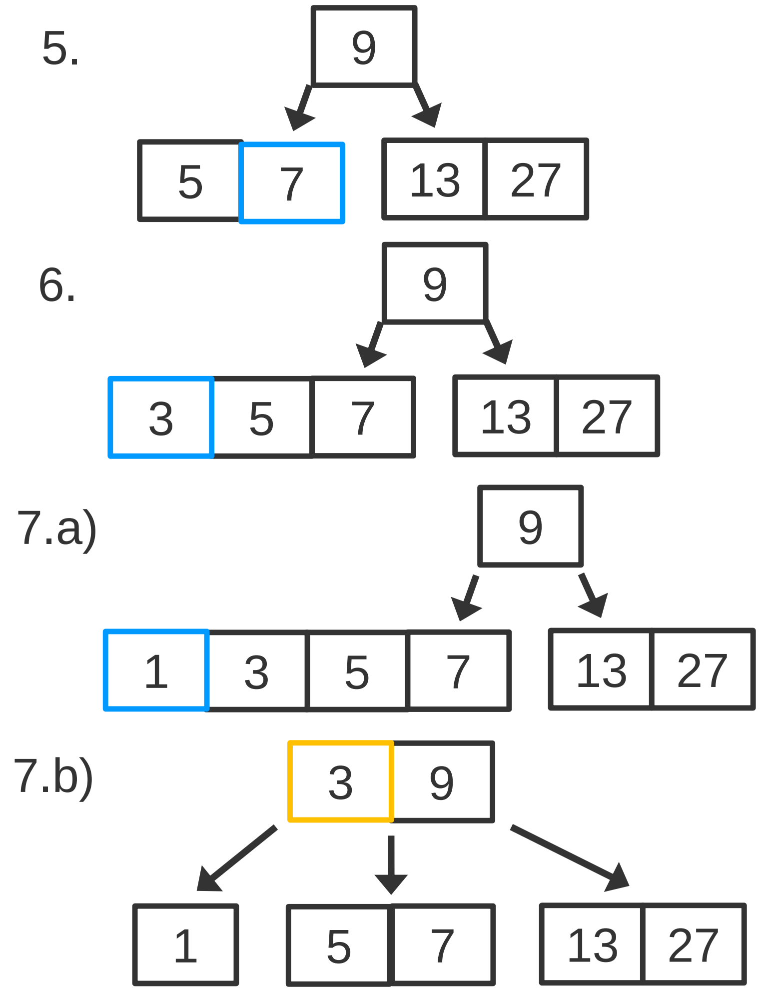
Full, split

Middle up,
left keys and
right keys

Indexing B-Tree Insert Algorithm Exercise

- Let m=4, min: 4/2-1=1, max = m-1=3
- Add 5, 13, 27, 9, 7, 3, 1, 4, 8, 10, 11

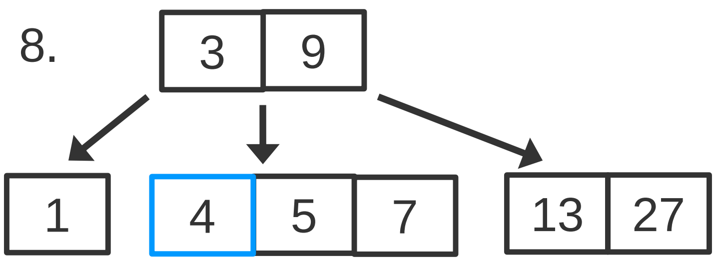




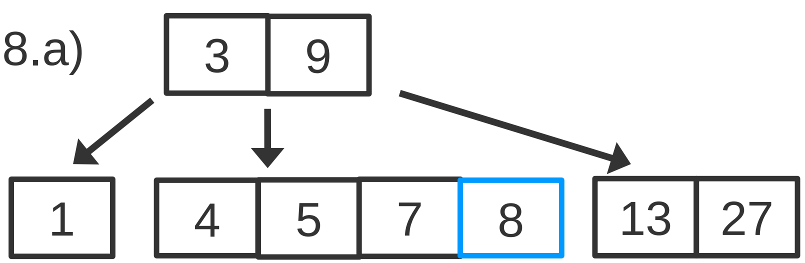
Indexing B-Tree Insert Algorithm Exercise

- Let m=4, min: 4/2-1=1, max = m-1=3
- Add 5, 13, 27, 9, 7, 3, 1, 4, 8, 10, 11

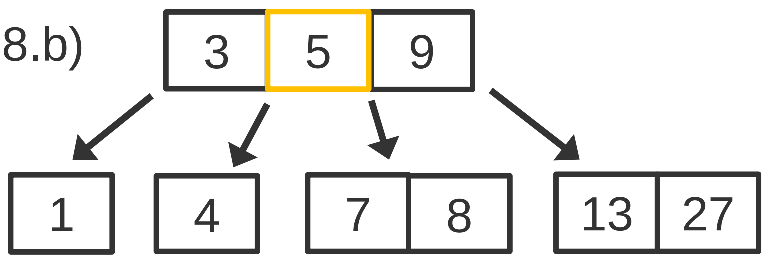
8.



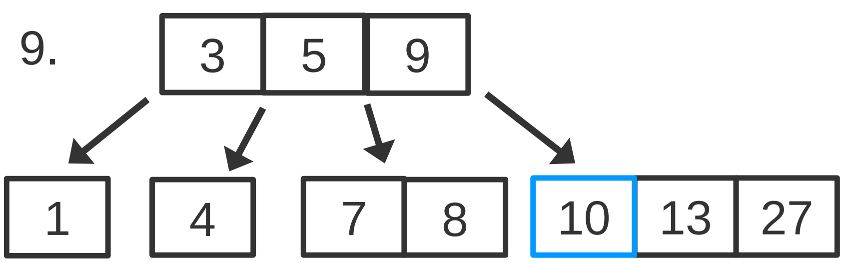
8.a)



8.b)



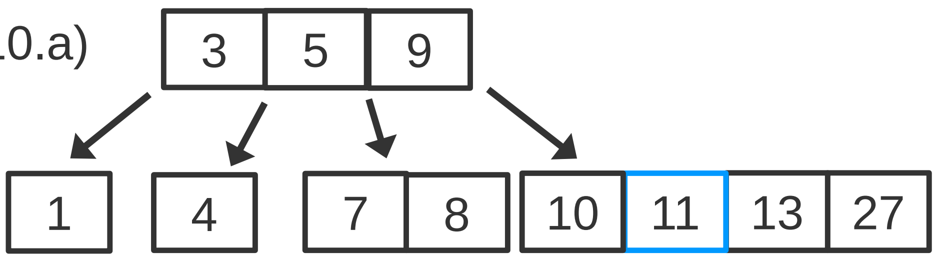
9.



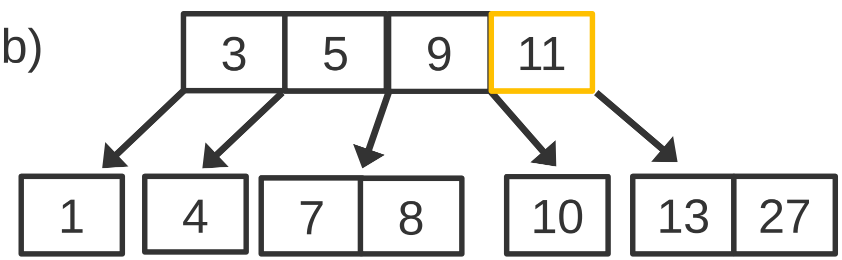
Indexing B-Tree Insert Algorithm Exercise

- Let m=4, min: 4/2-1=1, max = m-1=3
- Add 5, 13, 27, 9, 7, 3, 1, 4, 8, 10, 11

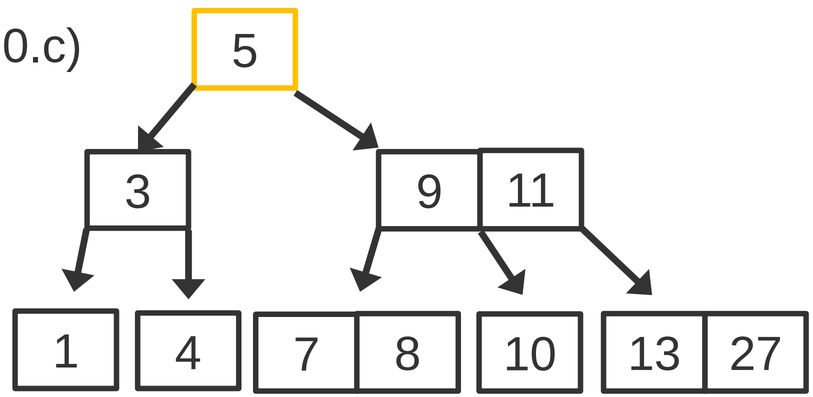
10.a)



10.b)



10.c)



End of Lecture

Thank you for your attention