Database System
(SW5)

8. Physical Design

Tiantian Liu
Department of Computer Science
Aalborg University
Fall 2025

Motivation

- Physical design directly impacts the performance of database operations.
- Faster databases lead to better user experiences, lower operational costs, and the ability to handle larger datasets effectively.

Learning Goals

- Understand how tables are stored in files
- Understand basic indexing techniques
- Understand the effects of storage and indexes on the performance of basic SQL queries

Agenda

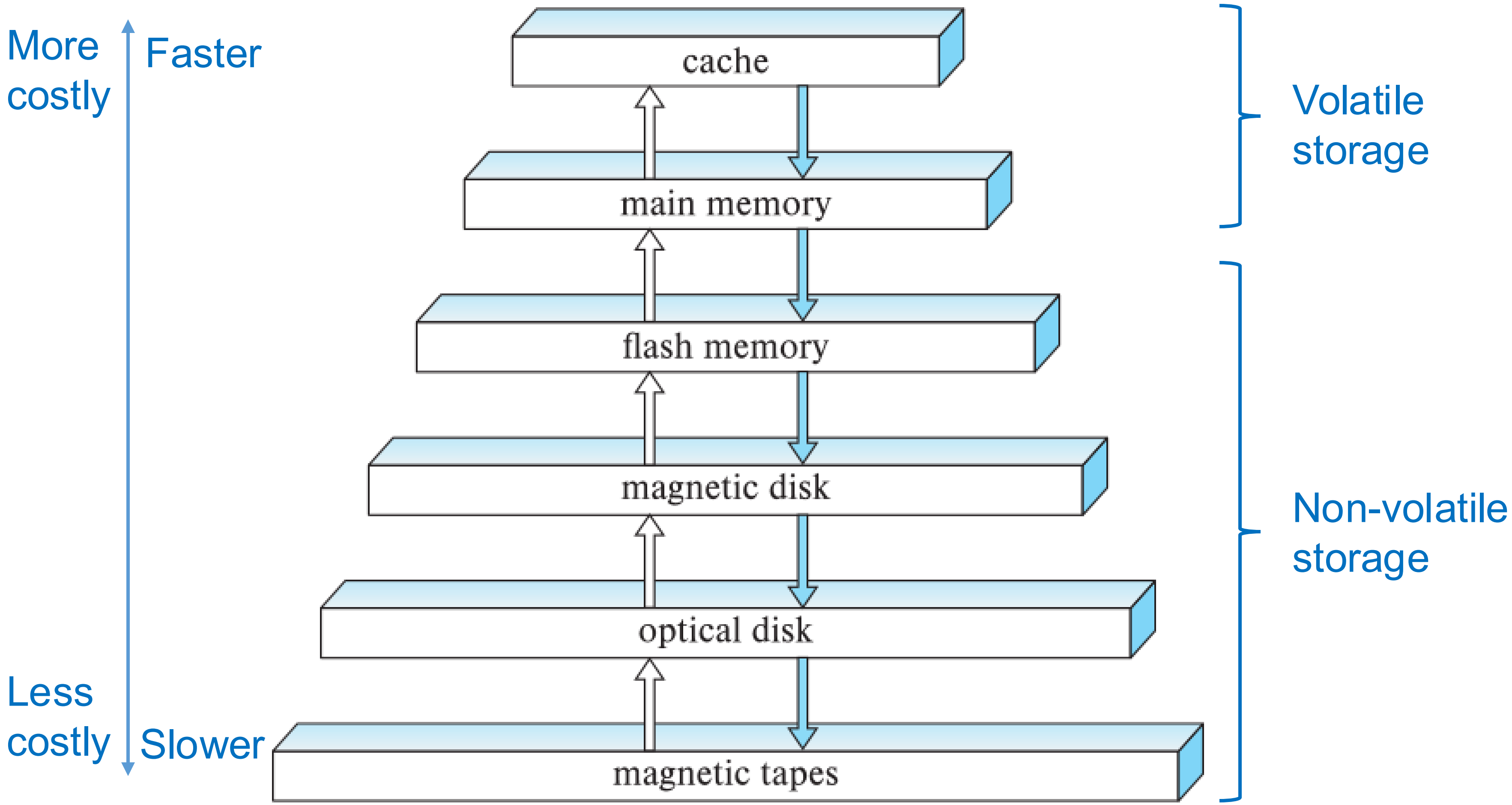
AALBORG UNIVERSITY

- Physical Storage Media
- File Organization
- Index
  - B+-Tree
  - Ordered Index
  - Hashing

Classification of Physical Storage Media

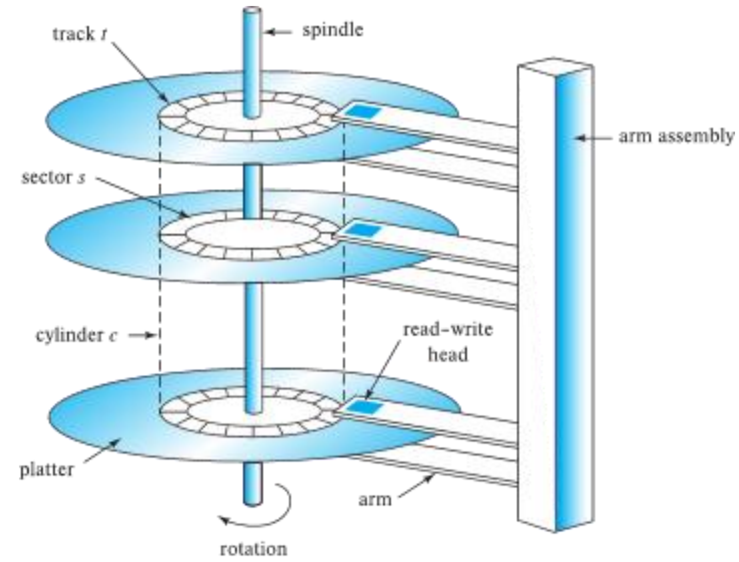
- Can differentiate storage into:
  - **Volatile storage**: Loses contents when power is switched off
  - **Non-volatile storage**: Contents persist even when power is switched off.
- Factors affecting choice of storage media include
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability

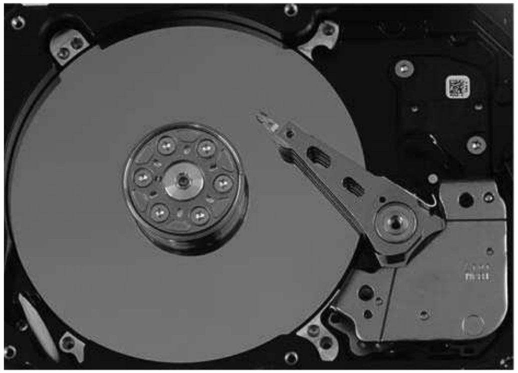
Storage Hierarchy



Magnetic Hard Disk Mechanism

- Read-write head
- Surface of platter divided into circular **tracks**
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors**.
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes





Optimization of Disk-Block Access

AALBORG UNIVERSITY

- A **block (page)** is a contiguous sequence of **sectors** from a single track
  - Basic unit of data transfer between disk and memory
- Functional requirements
  - Processing records sequentially
  - Efficient key-value search
  - Insertion/deletion of records
- Performance objectives
  - Little wasted space
  - Fast response time
  - High number of transactions

Agenda

- Physical Storage Media
- File Organization
- Index
  - B+-Tree
  - Ordered Index
  - Hashing

File Organization

- The database is stored as a collection of files.
  - Each file is a sequence of records.
  - A record is a sequence of fields.
- Fixed-size record
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations

  This case is easiest to implement; will consider variable length records later
- We assume that records are smaller than a disk block

Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte $n * i$ where *n* is the size of each record.
  - Record access is simple, but records may cross blocks
    - Modification: do not allow records to cross block boundaries

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record *i*: alternatives:
  - **move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1**
  - move record *n* to *I*
  - do not move records, but link all free records on a free list
- **Record 3 deleted**

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record *i*: alternatives:
  - move records *i + 1, . . ., n to i, . . . , n – 1*
  - **move record *n* to *I***
  - do not move records, but link all free records on a free list
- **Record 3 deleted and replaced by record 11**

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Fixed-Length Records

- Deletion of record *i*: alternatives:
  - move records *i + 1, . . ., n* to *i, . . . , n – 1*
  - move record *n* to *I*
  - **do not move records, but link all free records on a free list**

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

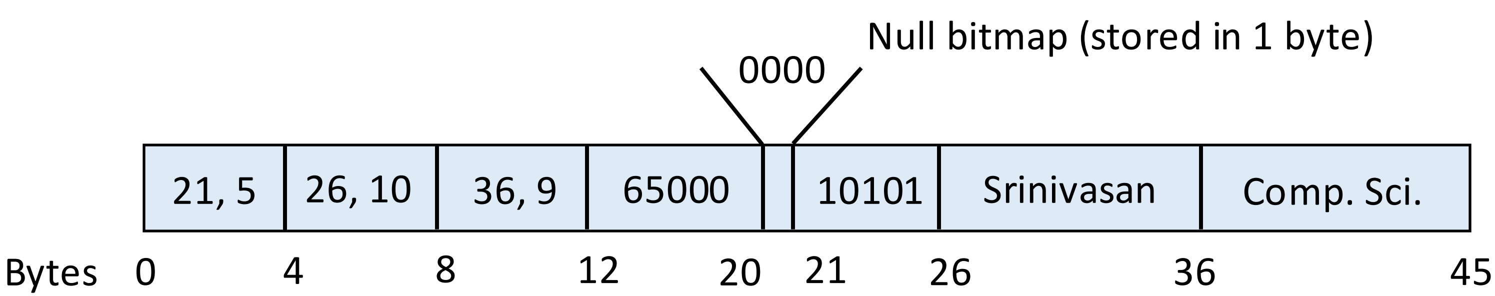
Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (varchar)
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap

Variable-Length Records

- Example
- (*10101, Srinivasan, Comp. Sci., 65000*)

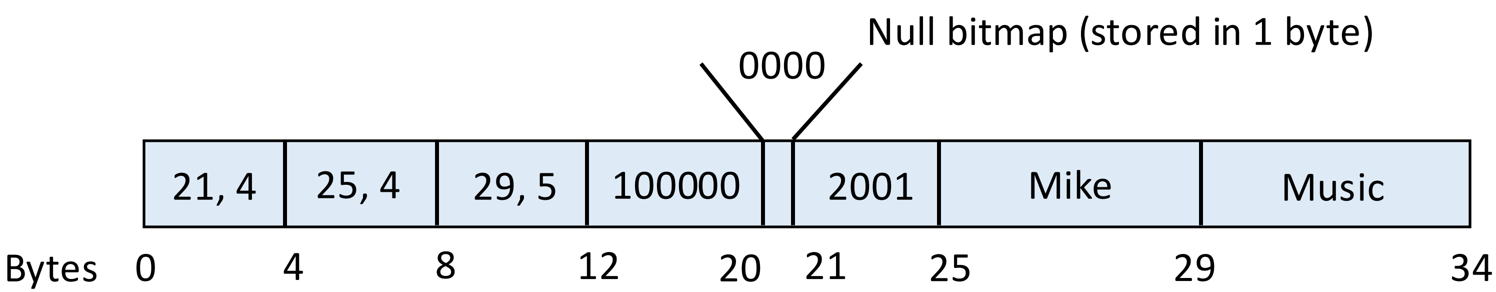
create table *instructor* (
  ID **varchar**(10),
  name **varchar**(20),
  dept_name **varchar**(20),
  salary **int**(8));



Variable-Length Records

- Exercise
- (*2001, Mike, Music, 100000*)

```
create table instructor (
    ID          varchar(10),
    name        varchar(20),
    dept_name   varchar(20),
    salary      int(8));
```



Organization of Records in Files

- **Heap** – record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Multitable clustering file organization** – records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O
- **B+-tree file organization**
  - Ordered storage even with inserts/deletes
- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

AALBORG UNIVERSITY

Sequential File Organization

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - If there is free space insert there
  - If no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Multitable Clustering File Organization

- Store several relations in one file using a **multitable clustering** file organization

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

department

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

instructor

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

multitable clustering of *department* and *instructor*

Agenda

- Physical Storage Media
- File Organization
- Index
    - B+-Tree
    - Ordered Index
    - Hashing

Basic Concepts

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- Search Key - attribute to set of attributes used to look up records in a file.
- An index file consists of records (called index entries) of the form

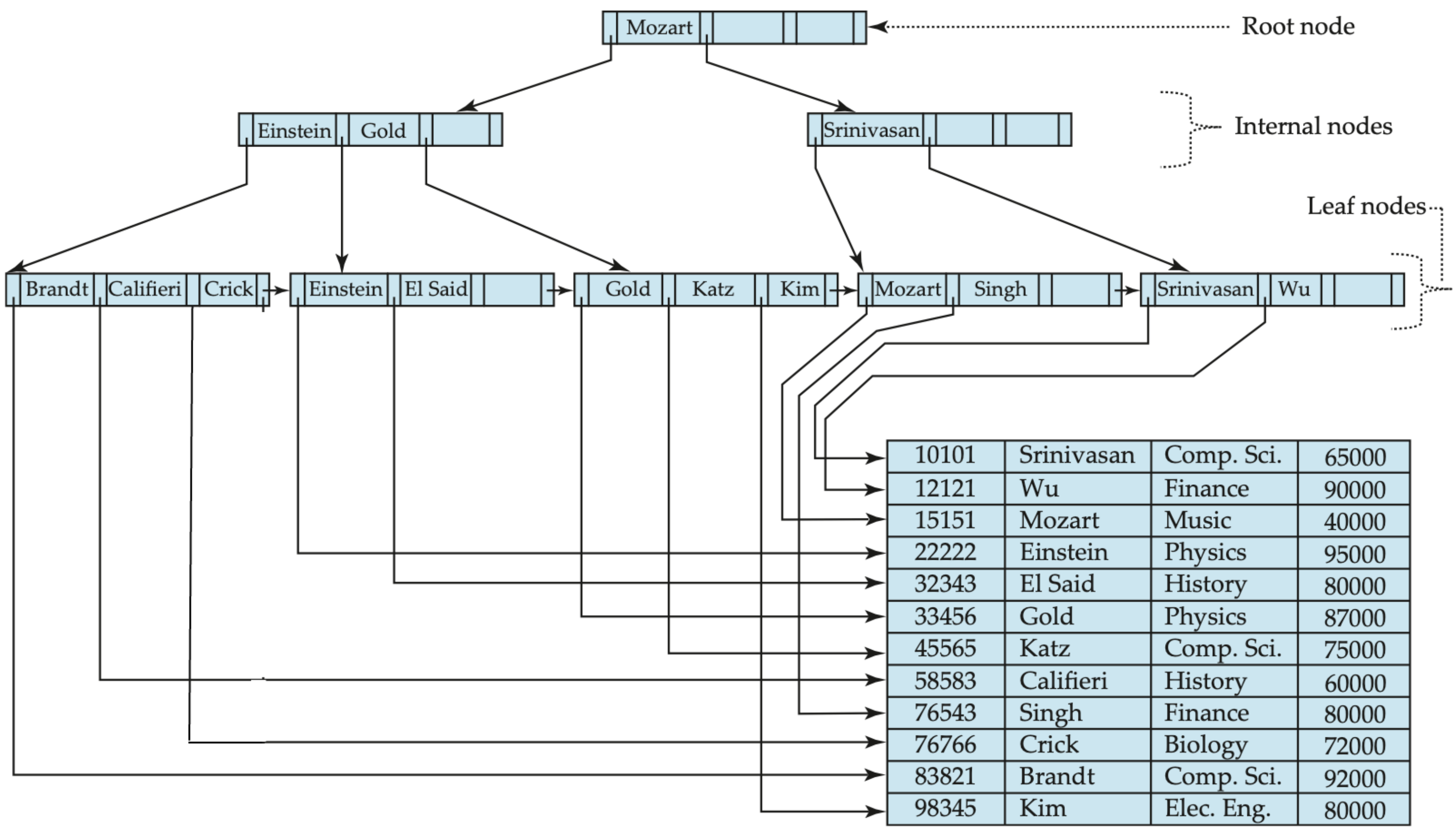
search-key	pointer

Agenda

AALBORG UNIVERSITY

- Physical Storage Media
- File Organization
- Index
  - B+-Tree
  - Ordered Index
  - Hashing

Example of B+-Tree



B+-Tree Node Structure

- Typical node

P_1	K_1	P_2	K_2	…	K_{n-2}	P_{n-1}	K_{n-1}	P_n

- K_i are the search-key values
- P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered

$$K_1 < K_2 < K_3 < \ldots < K_{n-1}$$

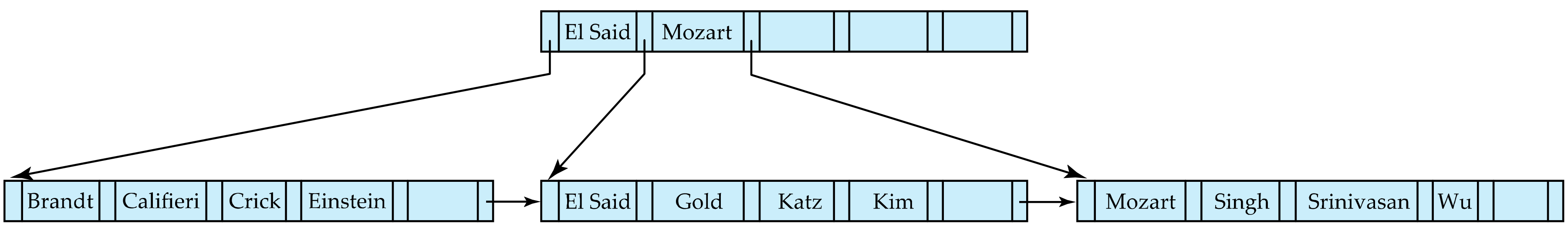
(Initially assume no duplicate keys)

B+-Tree Index Files

- A B+-tree with *n* degree (fanout) is a rooted tree satisfying the following properties:
  - All paths from root to leaf are of the same length
  - Each internal node has between $\lceil n/2 \rceil$ and n children (pointers)
  - A leaf node has between $\lceil (n-1)/2 \rceil$ and n–1 values (keys)
  - Special case:
    - If the root is not a leaf, it has at least 2 children.

Example of B+-tree

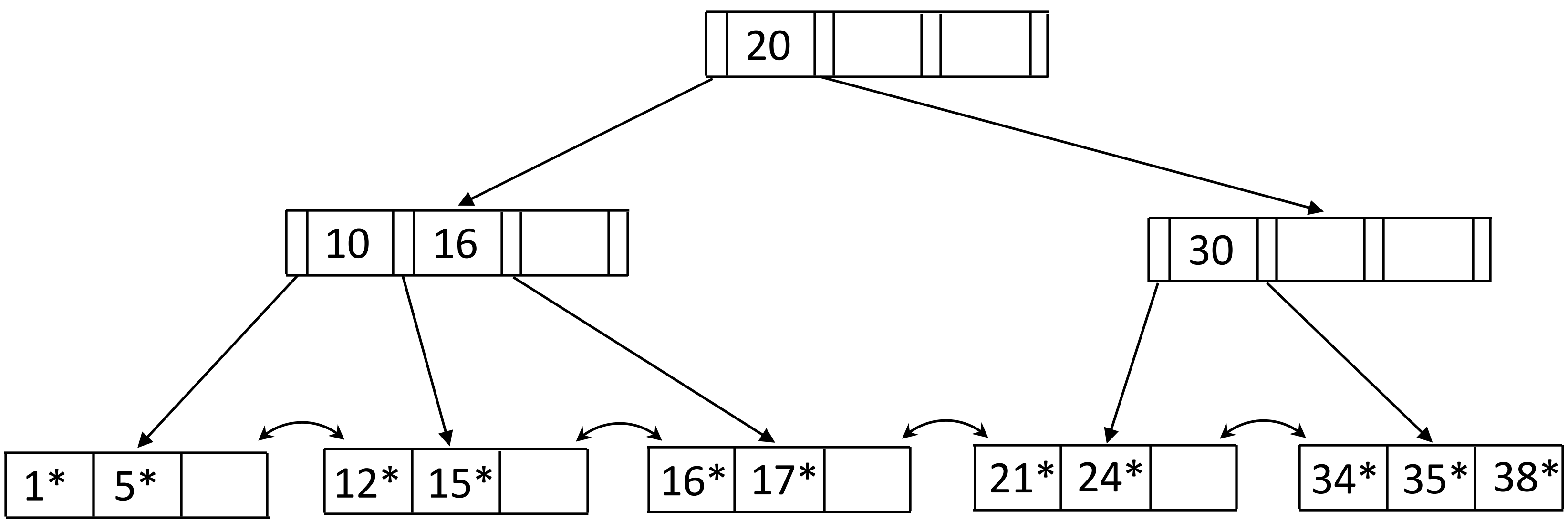
- B+-tree for *instructor* file (n = 6)



- Leaf nodes must have between 3 and 5 values (keys) ($\lceil (n-1)/2 \rceil$ and n–1, with n = 6).
- Non-leaf nodes other than root must have between 3 and 6 children (pointers) ($\lceil n/2 \rceil$ and n with n = 6).
- Root must have at least 2 children (pointers).

Exercise 1

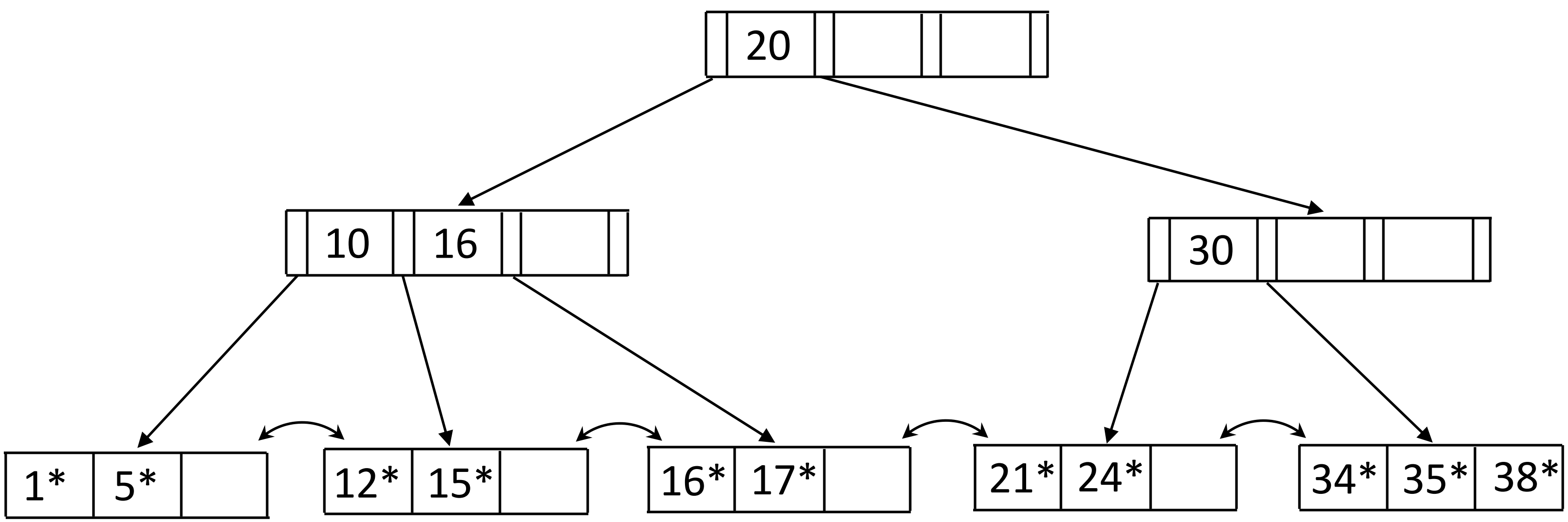
- For a B+-Tree with 4 degree
- How many children should an internal node have? (range)
- How many values should a leaf node have? (range)



B+ Tree with degree n = 4

Exercise 1 -- Solution

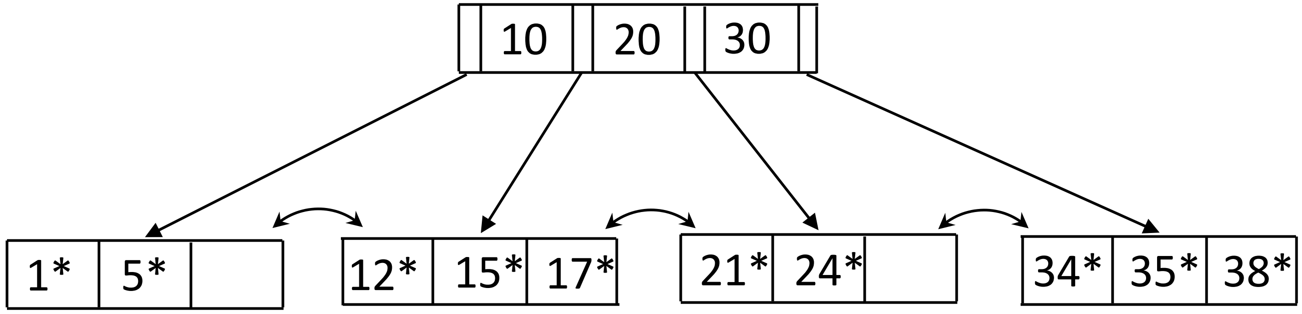
- For a B+-Tree with 4 degree
- How many children should an internal node have? [2, 4]
- How many values should a leaf node have? [2, 3]



B+ Tree with degree n = 4

Example of B+-Tree

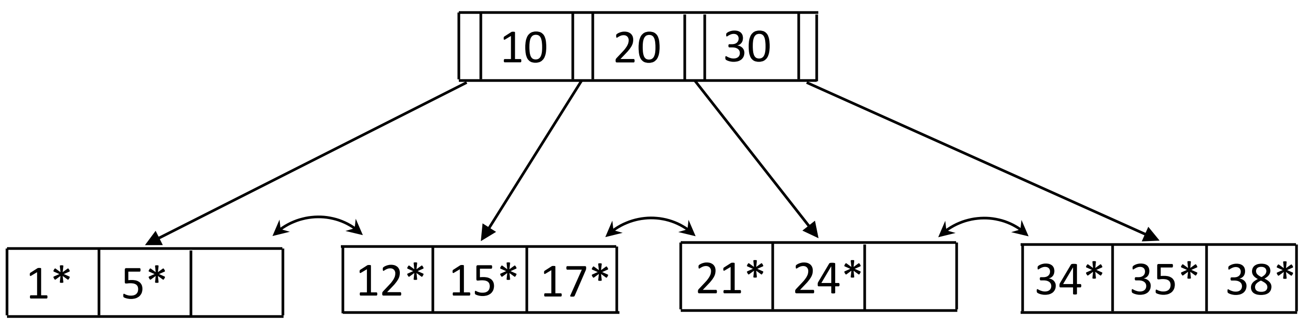
- The left pointer of a key K in a non-leaf node leads towards keys less than K, while the right pointer leads towards keys greater than or equal to K.
- A leaf node underflows when the number of keys goes below 2.
- An internal node underflows when the number of pointers goes below 2.
- Assume the entries in leaf nodes contain the actual data.



B+ Tree with degree n = 4

Search

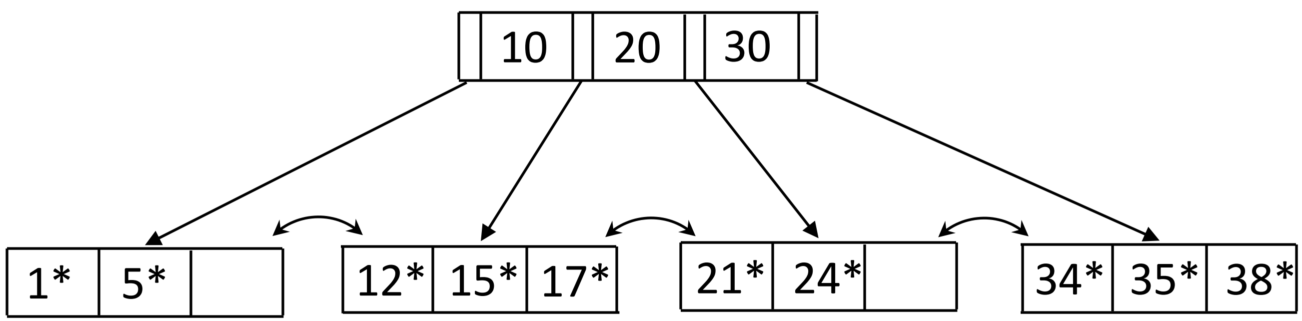
- Search for 5*, 15*
- Search for all data entries >= 20*



B+ Tree with degree n = 4

B+-Tree--Insert

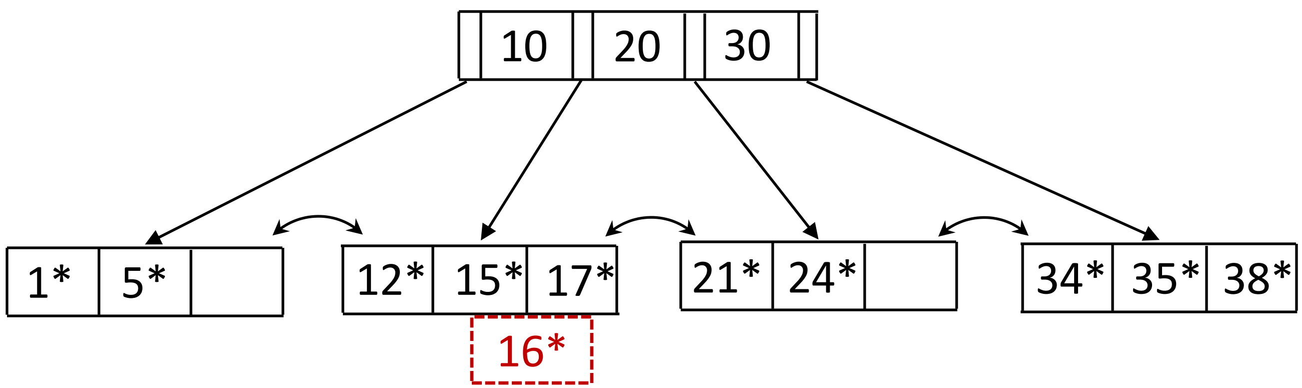
- Go to the correct leaf
- Do recursively:
  - If non-full then
    - insert the entry
  - else
    - split and copy/push up the middle key to the parent node



B+ Tree with degree n = 4

B+-Tree--Insert 16*

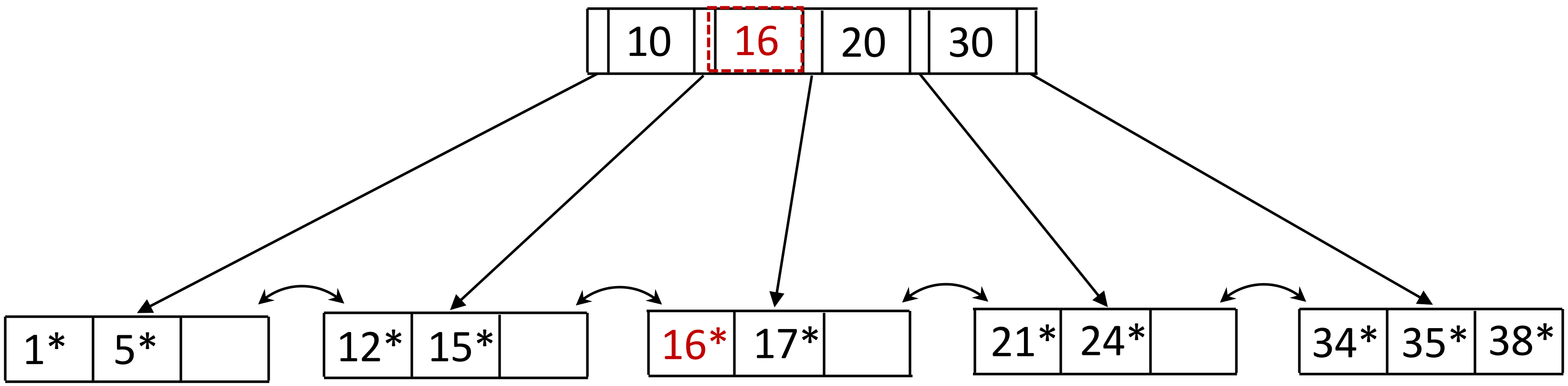
- Go to the correct leaf



B+ Tree with degree n = 4

B+-Tree--Insert 16*

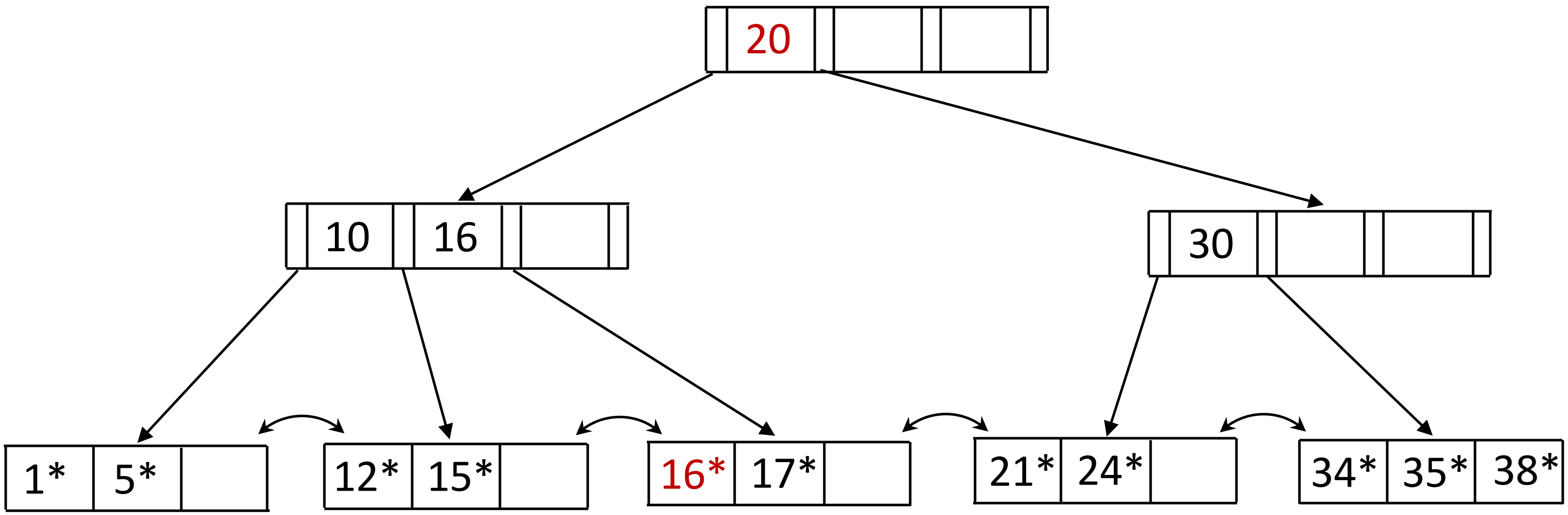
- Split the leaf node and copy up the middle key



B+ Tree with degree n = 4

B+-Tree--Insert 16*

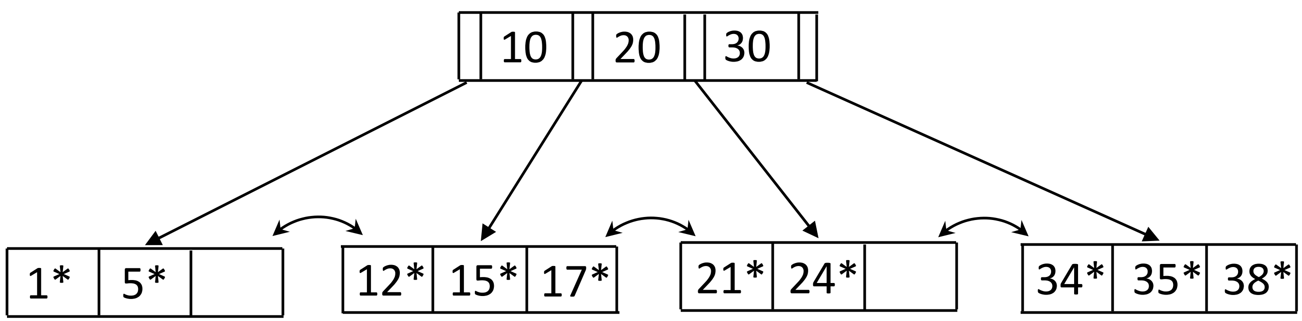
- Split the non-leaf node and push up the middle key



B+ Tree with degree n = 4

B+-Tree--Delete

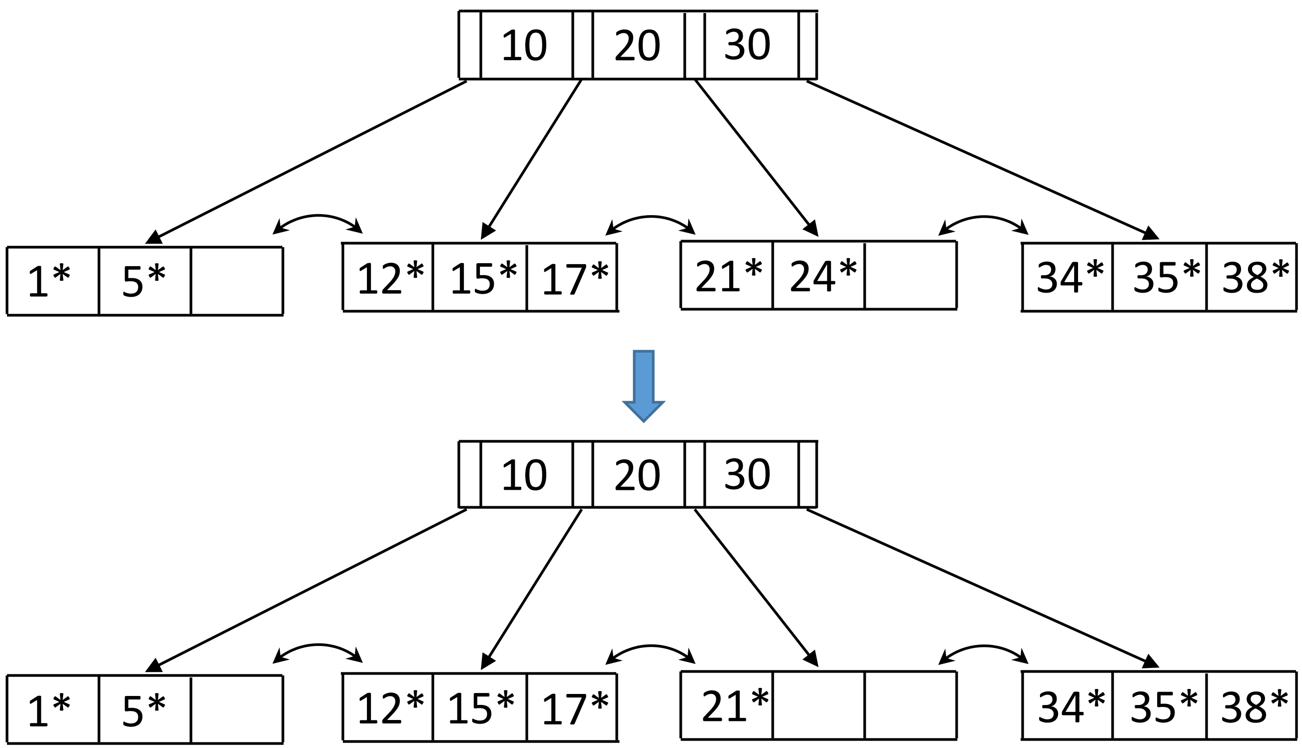
- Go to the correct leaf and delete the entry
- If it underflows, then
  - redistribute with the sibling;
  - if the sibling doesn't have enough entries then
    - merge with the sibling;



B+ Tree with degree n = 4

B+-Tree--Delete 24*

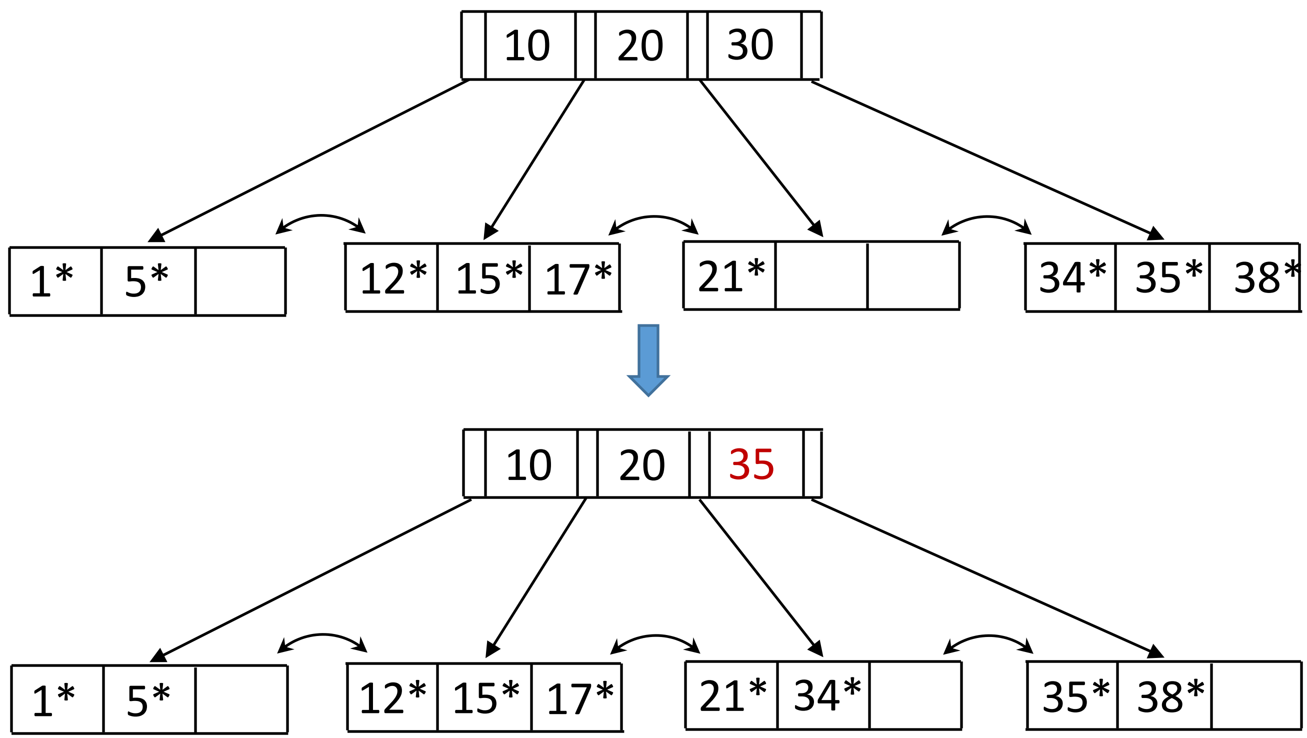
- Go to the correct leaf and delete the entry



B+ Tree with degree n = 4

B+-Tree--Delete 24*

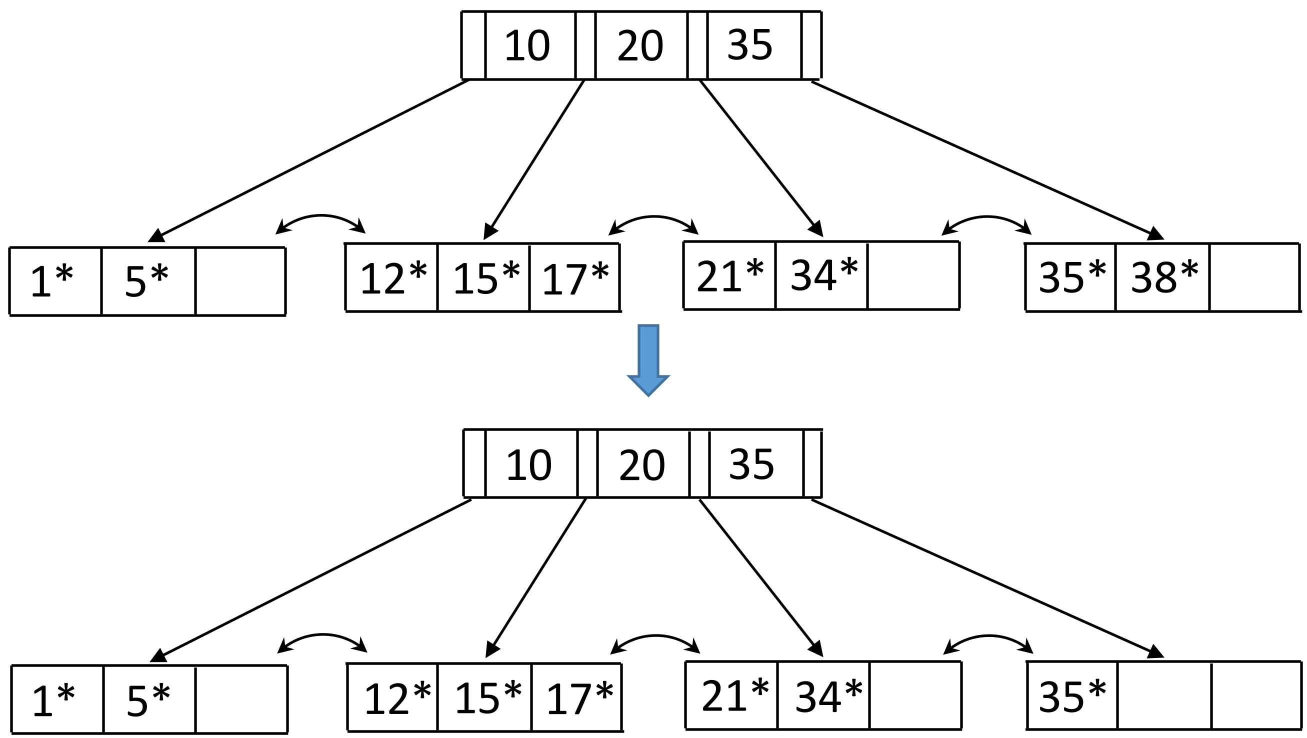
- Redistribute with the sibling



B+ Tree with degree n = 4

B+-Tree--Delete 38*

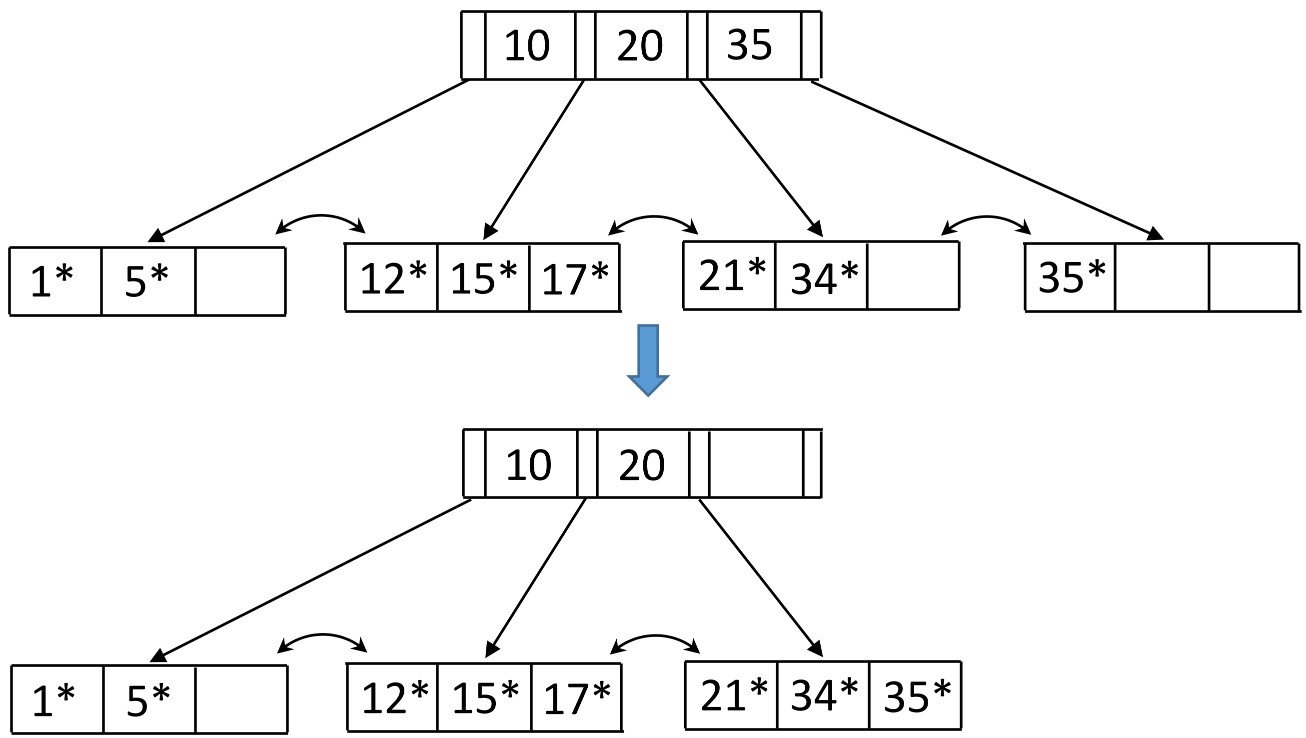
- Go to the correct leaf and delete the entry



B+ Tree with degree n = 4

B+-Tree--Delete 38*

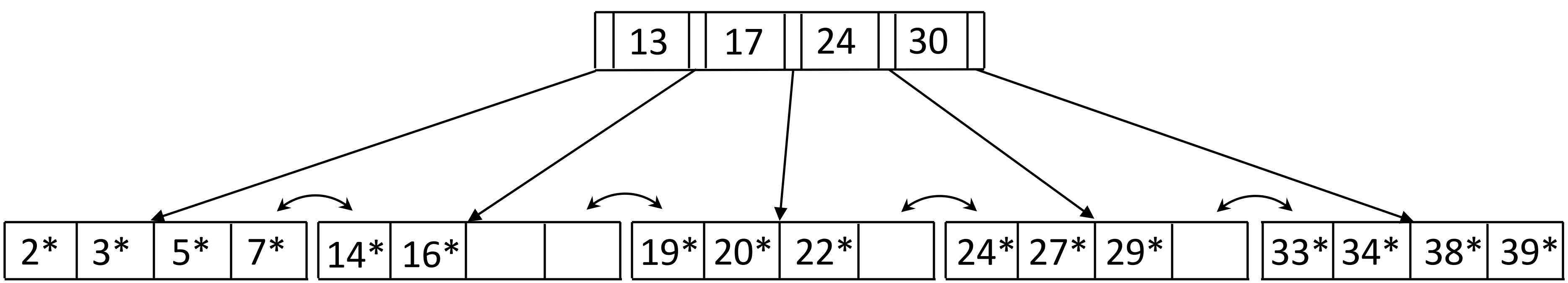
- Merge with the sibling



B+ Tree with degree n = 4

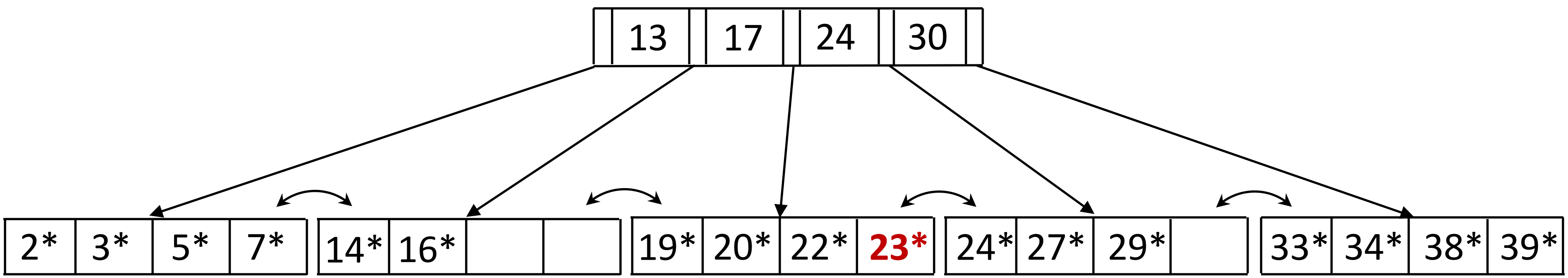
Exercise 2--Insert 23*

- A leaf node underflows when the number of keys goes below 2.
- An internal node underflows when the number of pointers goes below 3.
- Assume the entries in leaf nodes contain the actual data



B+ Tree with degree n = 5

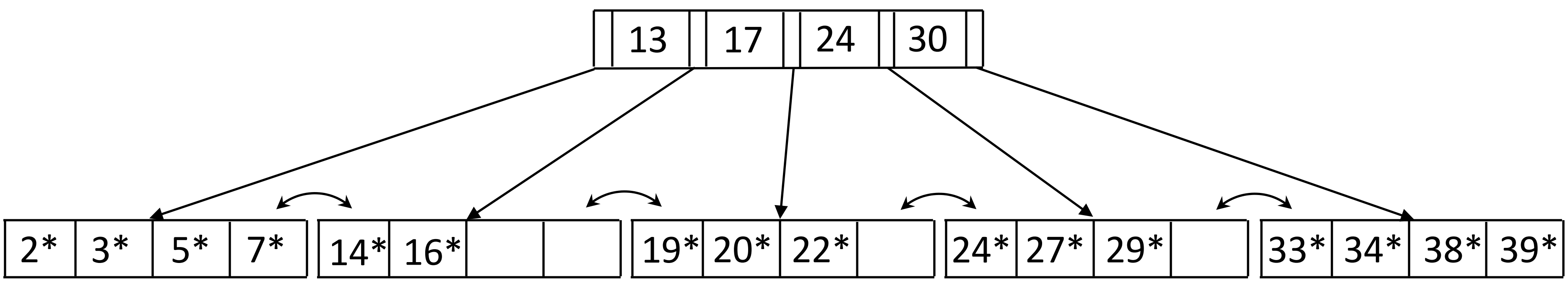
Solution 2--Insert 23*



B+ Tree with degree n = 5

Exercise 3--Insert 8*

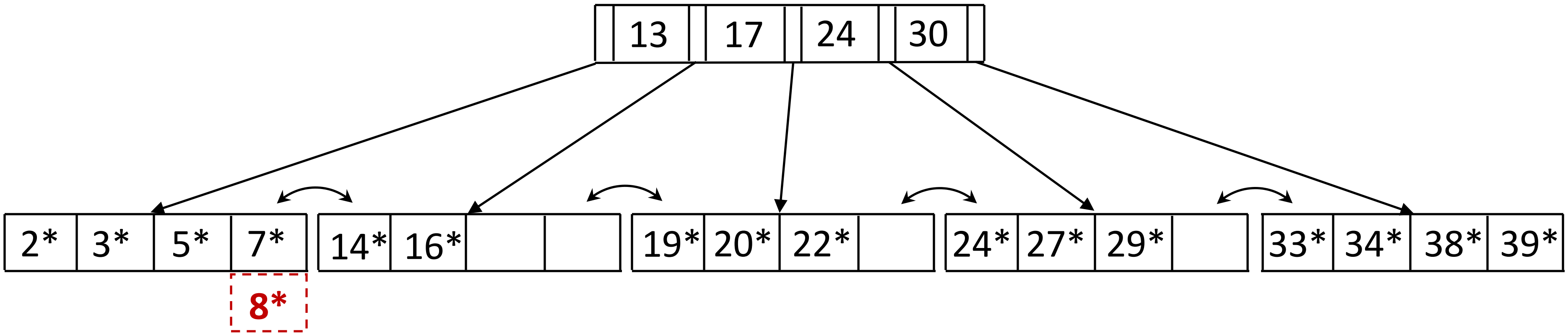
- A leaf node underflows when the number of keys goes below 2.
- An internal node underflows when the number of pointers goes below 3.
- Assume the entries in leaf nodes contain the actual data



B+ Tree with degree n = 5

Solution 3--Insert 8*

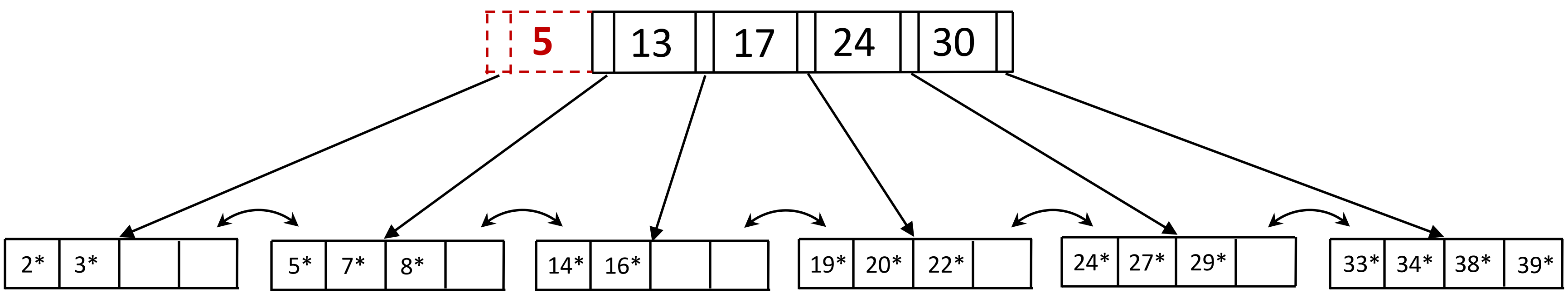
- Go to the correct leaf



B+ Tree with degree n = 5

Solution 3--Insert 8*

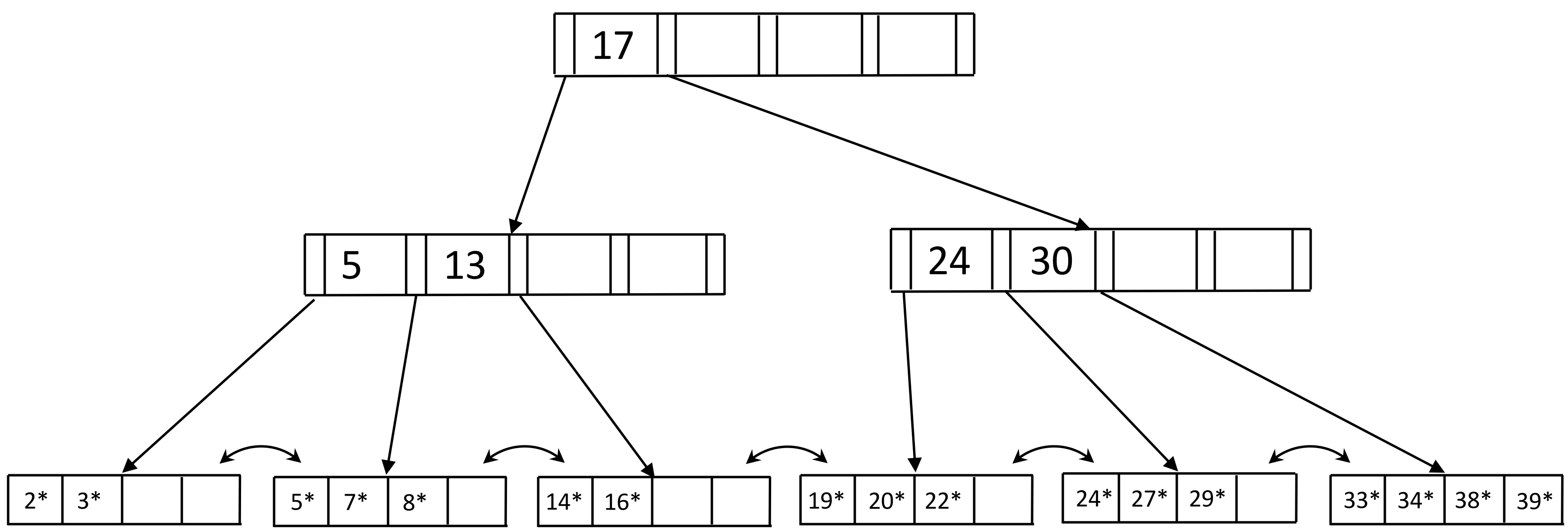
- Split and copy up the middle key to the parent node



B+ Tree with degree n = 5

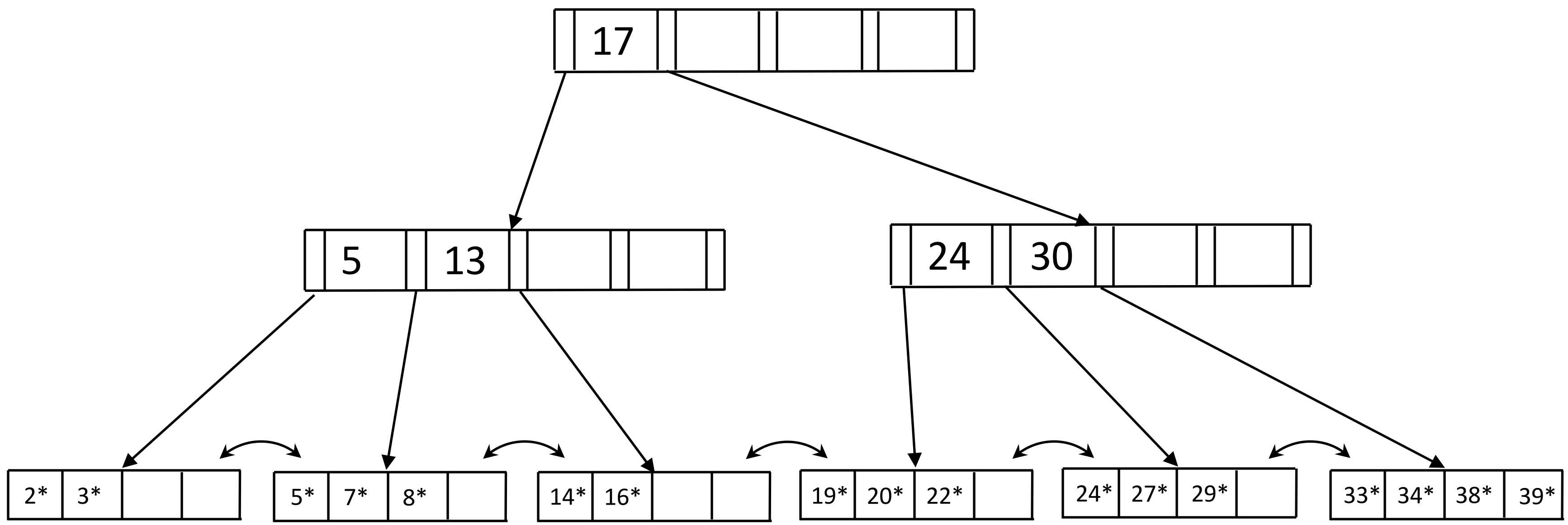
Solution 3--Insert 8*

- Split and push up the middle key to the parent node



B+ Tree with degree n = 5

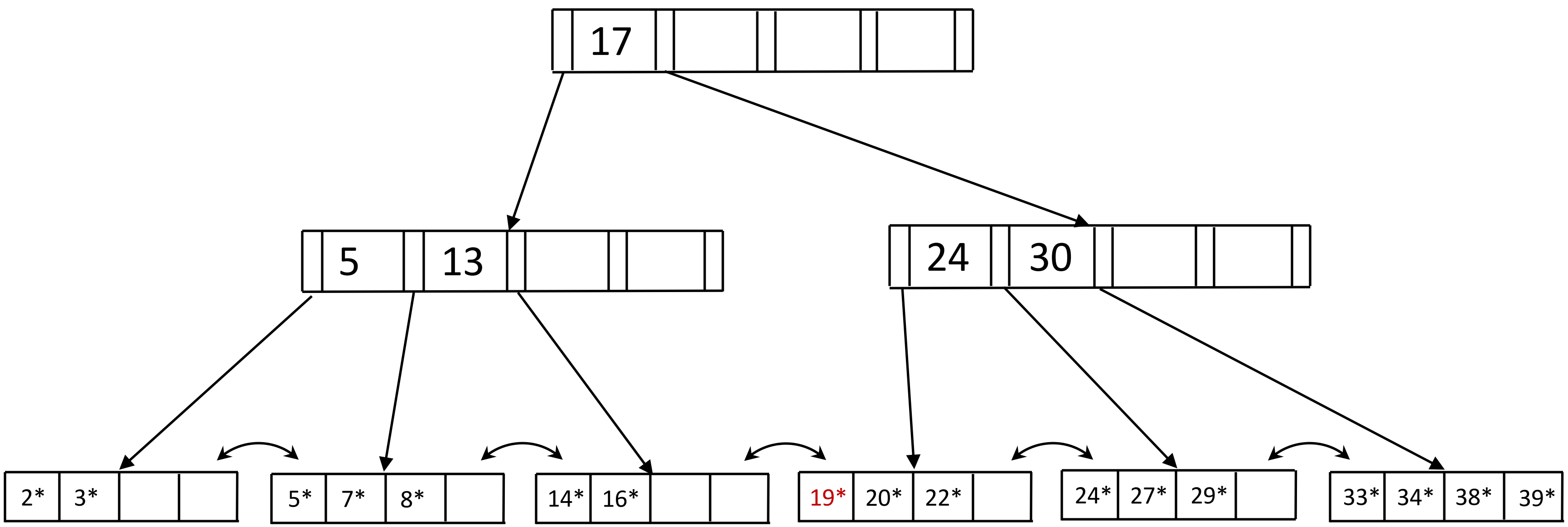
Exercise 4--Delete 19*



B+ Tree with degree n = 5

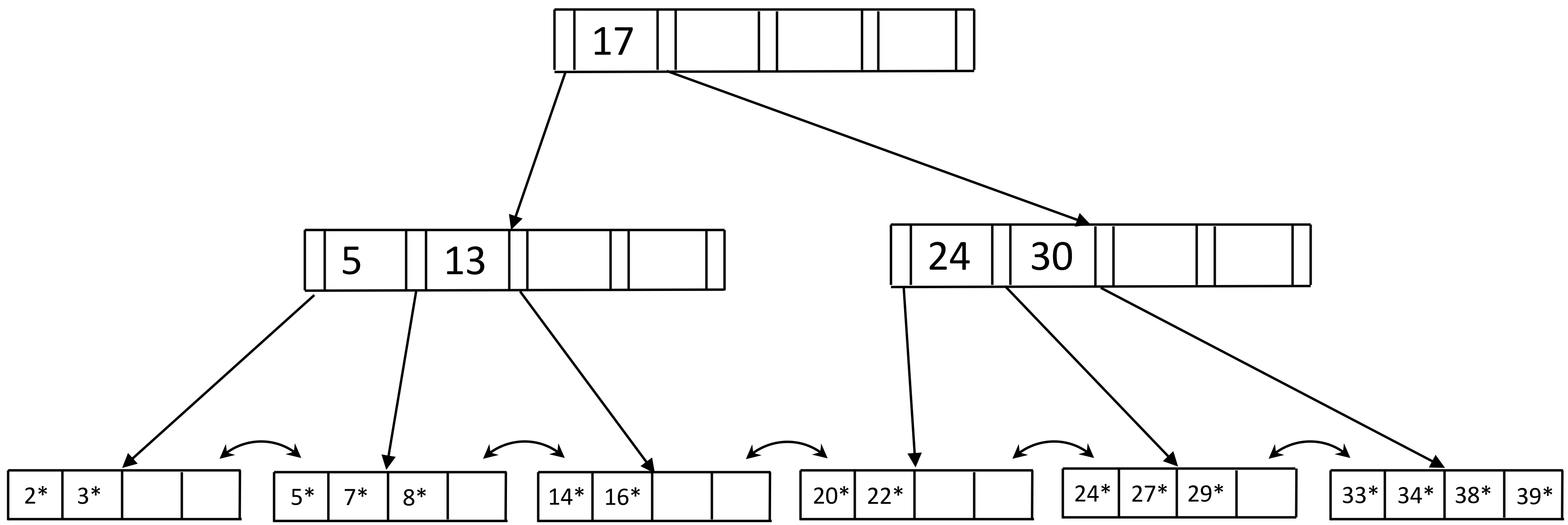
Solution 4--Delete 19*

- Go to the correct leaf and delete the entry



B+ Tree with degree n = 5

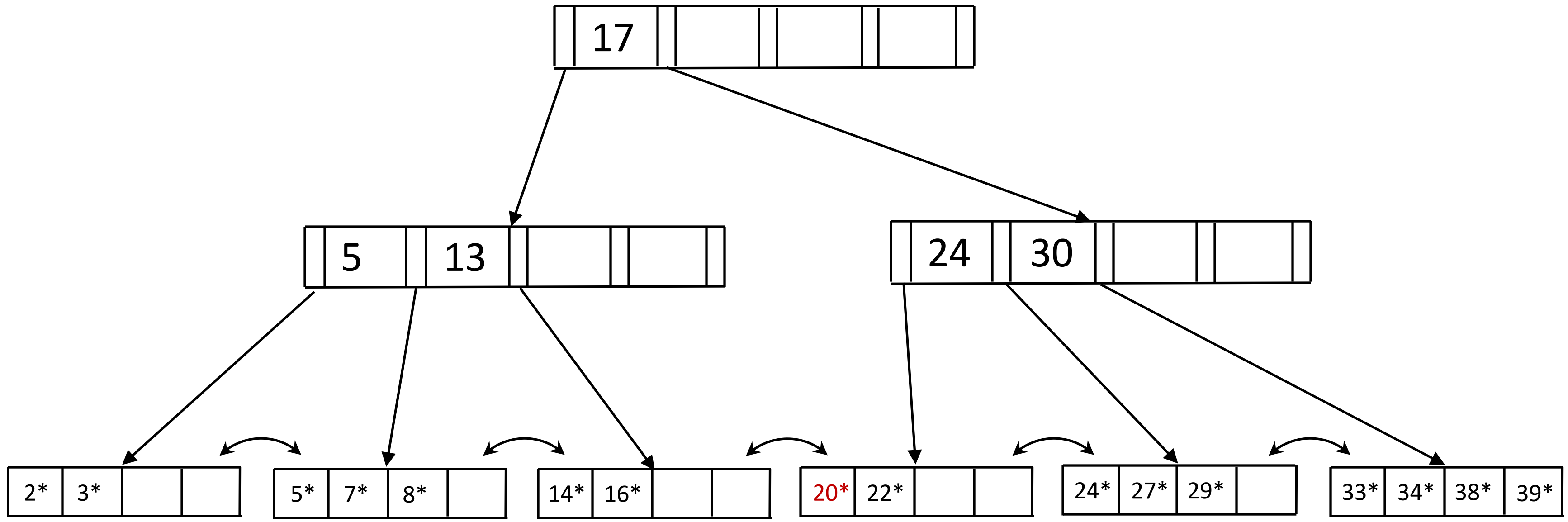
Exercise 5--Delete 20*



B+ Tree with degree n = 5

Solution 5--Delete 20*

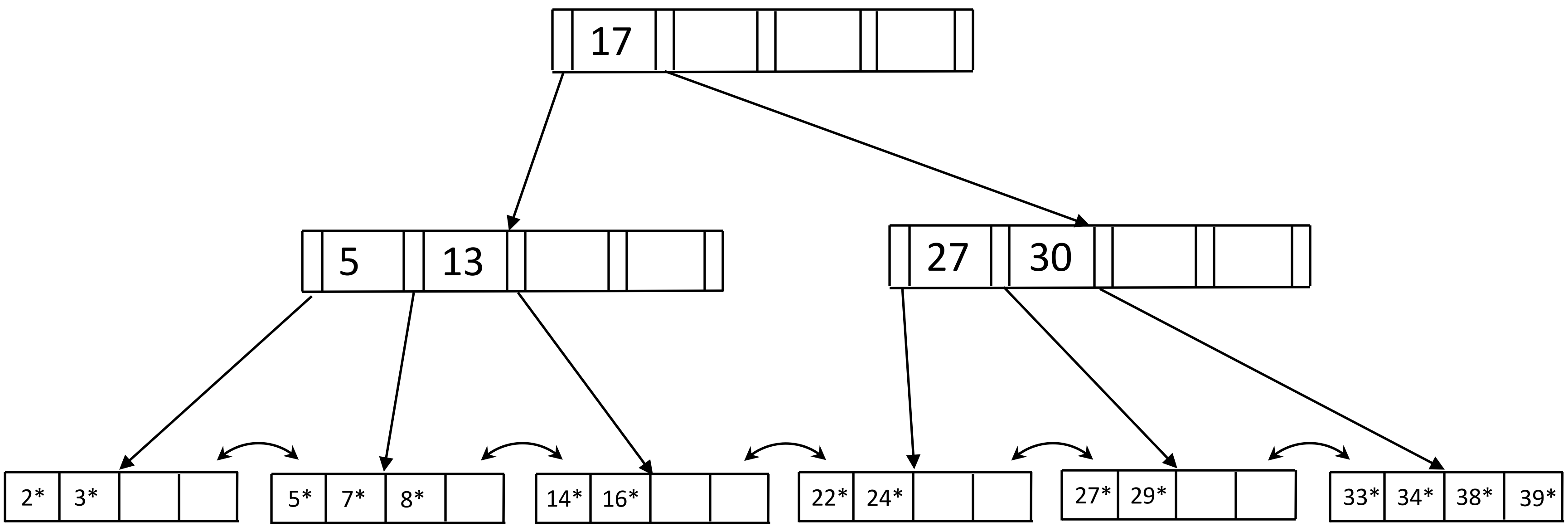
- Go to the correct leaf and delete the entry



B+ Tree with degree n = 5

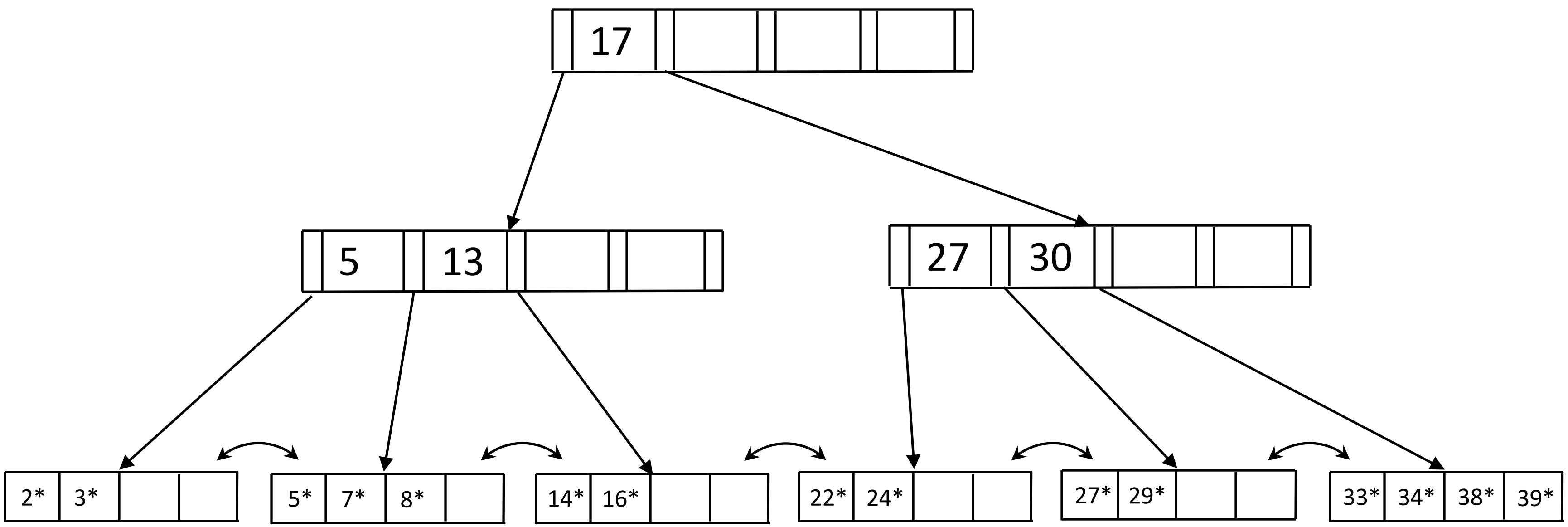
Solution 5--Delete 20*

- Redistribute with the sibling
- Copy up of middle key “27”



B+ Tree with degree n = 5

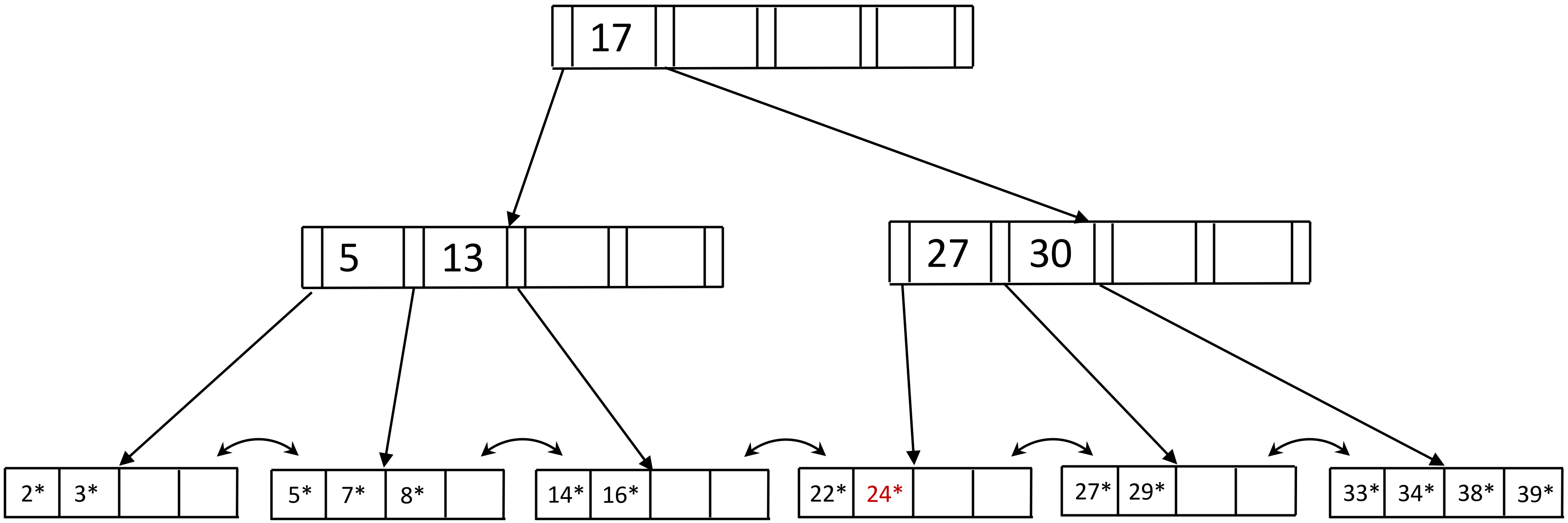
Exercise 6--Delete 24*



B+ Tree with degree n = 5

Solution 6--Delete 24*

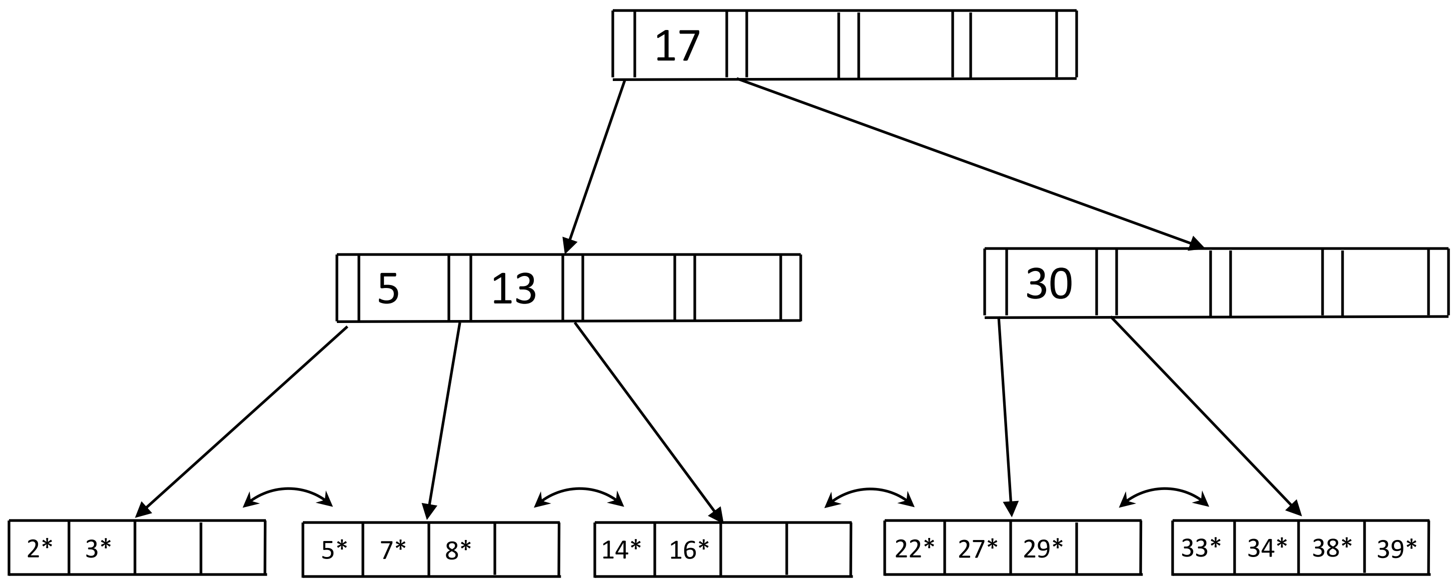
- Go to the correct leaf and delete the entry



B+ Tree with degree n = 5

Solution 6--Delete 24*

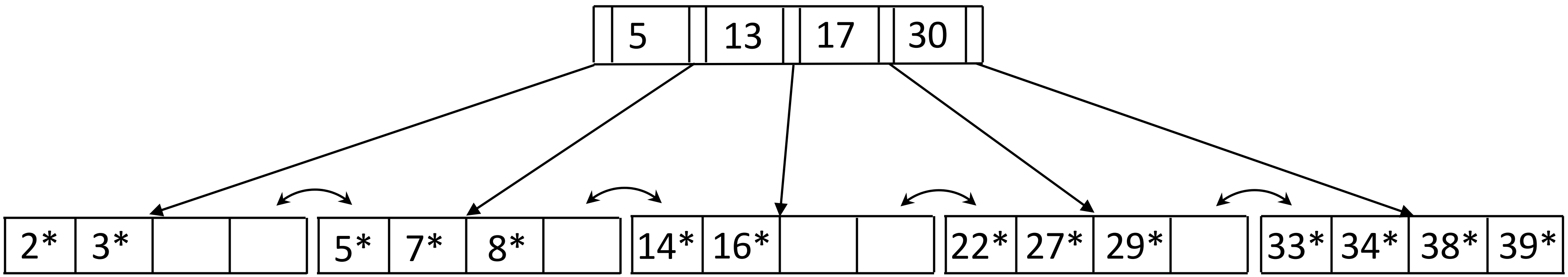
- Merge with the sibling



B+ Tree with degree n = 5

Solution 6-- Delete 24*

- Merge with the sibling



B+ Tree with degree n = 5

Summary

- Physical Storage Media
- File Organization
- Index
  - B+-Tree
  - Ordered Index (Self study)
  - Hashing (Self study)

Next Lecture

- Query Execution and Optimization
  - Understand how selection statements are executed
  - Understand the basic join algorithms
  - Understand the basics of heuristic (logical) query optimization
  - Understand the basics of physical query optimization
- Transaction and Concurrency Control
  - Understanding the transaction concept
  - Understanding serializability
  - Understand and use lock-based concurrency control
  - Understand and use two-phase locking

Agenda

- Physical Storage Media
- File Organization
- Index
  - B+-Tree
  - Ordered Index
  - Hashing

Ordered Index

AALBORG UNIVERSITY

- In an **ordered index**, index entries are sorted on the search key value.
  - **Primary index**: in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
    - Also called clustering index
    - The search key of a primary index is usually but not necessarily the primary key.
  - **Secondary index**: an index whose search key specifies an order different from the sequential order of the file.
    - Also called nonclustering index.

Primary Dense Index Files

- **Dense index** — Index record appears for every search-key value in the file.
  - E.g. index on *ID* attribute of *instructor* relation

10101	
12121	
15151	
22222	
32343	
33456	
45565	
58583	
76543	
76766	
83821	
98345	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Primary Dense Index Files (Cont.)

- Dense index on *dept_name*, with *instructor* file sorted on *dept_name*

Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

76766	Crick	Biology	72000	
10101	Srinivasan	Comp. Sci.	65000	
45565	Katz	Comp. Sci.	75000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
12121	Wu	Finance	90000	
76543	Singh	Finance	80000	
32343	El Said	History	60000	
58583	Califieri	History	62000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
33465	Gold	Physics	87000	

Primary Sparse Index Files

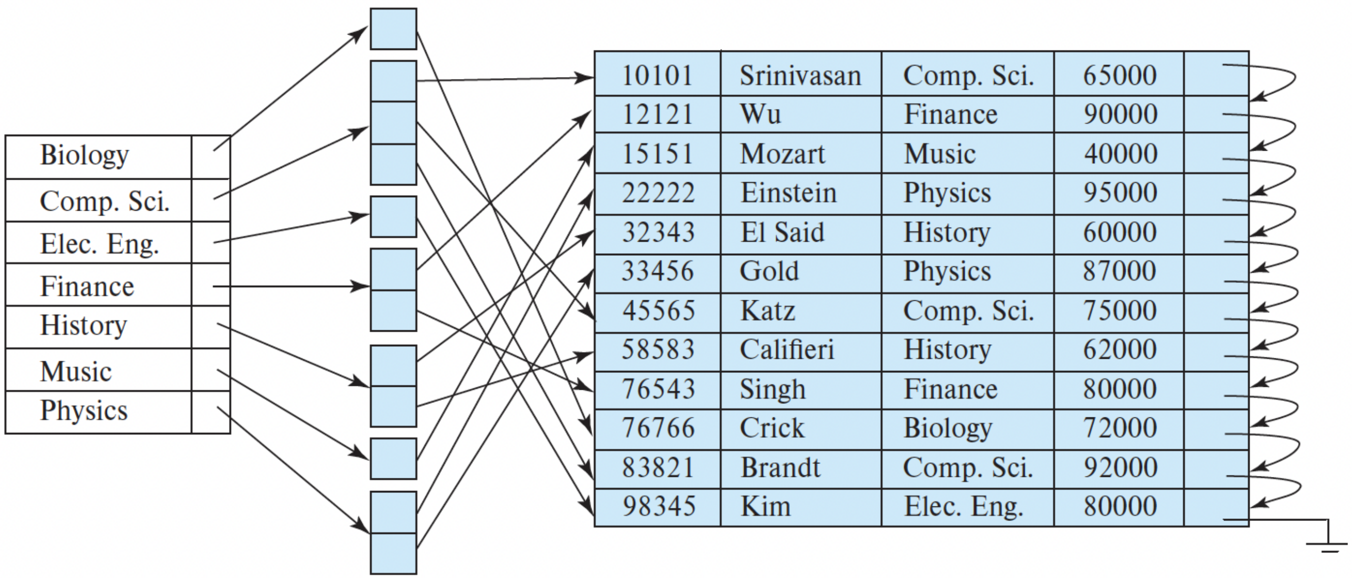
- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points

10101	
32343	
76766	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Secondary Index Example

- Secondary index on salary field of instructor



- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value.
- Secondary indices have to be dense

Index Summary

- Primary vs. secondary (clustering vs. non-clustering)
  - Is the file ordered on the search key of the index?
- Dense vs. sparse: A separate index entry for each record (unique search key value)?
  - Yes → dense
  - No → sparse
- Tradeoff
  - Dense index: faster location of records
  - Sparse index: smaller index

Agenda

- Physical Storage Media
- File Organization
- Index
  - B+-Tree
  - Ordered Index
  - Hashing

Static Hashing

- A bucket is a unit of storage containing one or more entries (a bucket is typically a disk block).
  - We obtain the bucket of an entry from its search-key value using a hash function
- Hash function *h* is a function from the set of all search-key values *K* to the set of all bucket addresses *B*.
  - Hash function is used to locate entries for access, insertion as well as deletion.
- Entries with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate an entry.
- In a hash index, buckets store entries with pointers to records

Example of Hash File Organization

AALBORG UNIVERSITY

- Hash file organization of *instructor* file, using *dept_name* as key
  - There are 10 buckets
  - The hash function returns the sum of the binary representations of the characters modulo 10
  - E.g. $h(Music) = 1$ $h(History) = 2$
    $h(Physics) = 3$ $h(Elec.\ Eng.) = 3$

Example of Hash File Organization

- Hash file organization of instructor file, using *dept_name* as key.

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Handling of Bucket Overflows

AALBORG UNIVERSITY

- Bucket overflow can occur because of
  - Insufficient buckets
  - Skew in distribution of records. This can occur due to two reasons:
    - multiple records have same search-key value
    - chosen hash function produces non-uniform distribution of key values
- Although the probability of bucket overflow can be reduced, it cannot be eliminated; it is handled by using overflow buckets.

Handling of Bucket Overflows

- Overflow chaining – the overflow buckets of a given bucket are chained together in a linked list.
- Above scheme is called closed addressing (also called closed hashing or open hashing depending on the book you use)