CSCI3170 Introduction to Database Systems

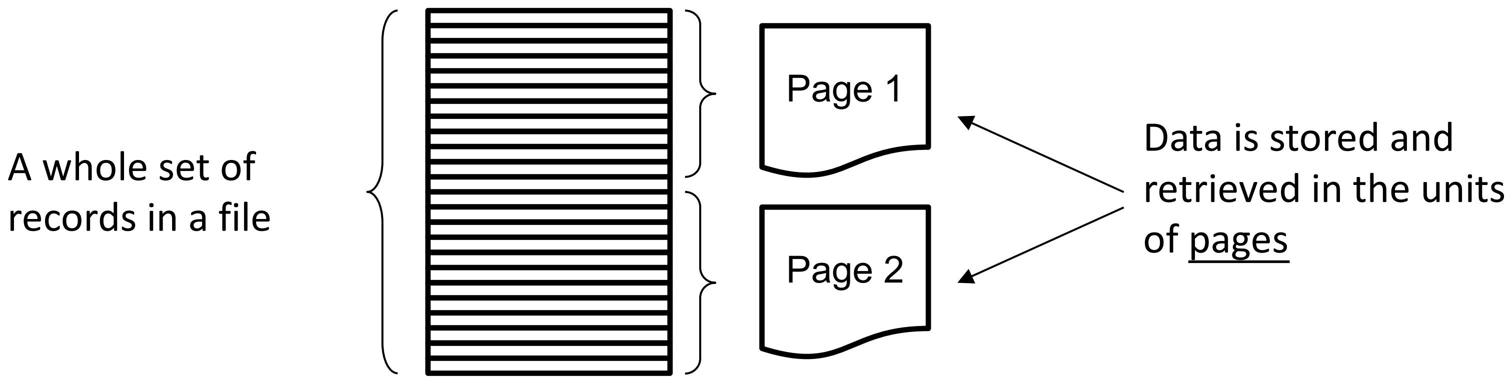
Tutorial 8 – Storage and Indexes (1)

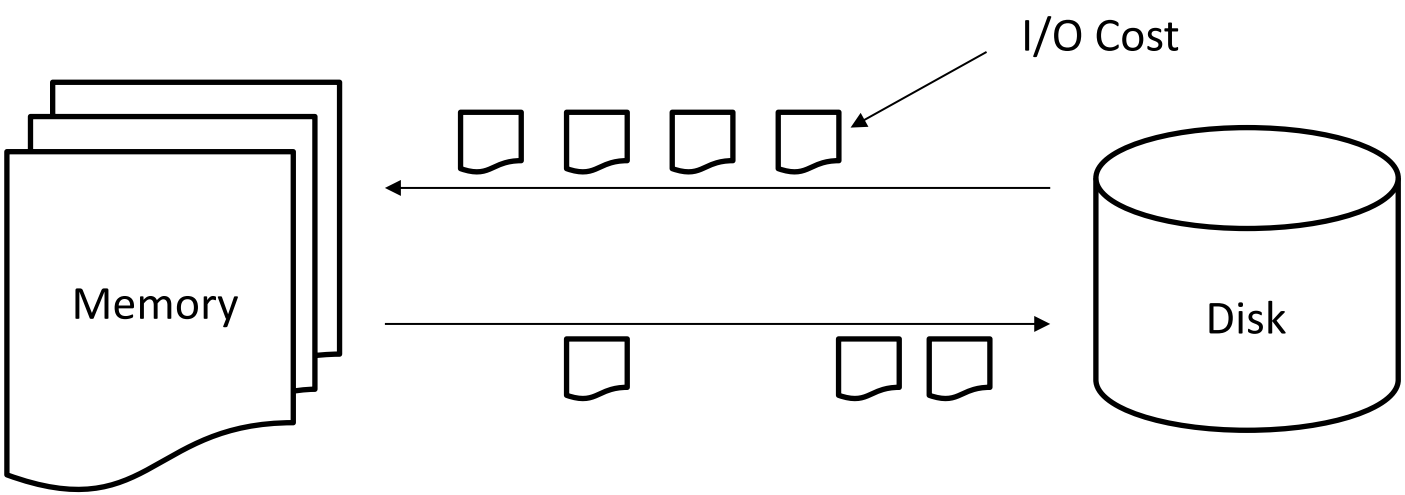
Outline

- Overview of Storage and Indexes
- Tree-structured Indexing
  - B+ Tree
- Hash-based Indexes
  - Static Hashing
  - Dynamic Hashing

Storage

- Files and pages





Record and Index

- Record
  - Record ID = <Page ID>+<Offset>
    - E.g. Record ID: <3>+<10> = The 10th record in the 3rd page
- Index
  - Given a search key K, index can be used to speed up the selection of a set of particular pages.
  - A index file contains a collection of data entries.
  - The data entry of search key K is denoted as K*.
    - K* = <K, Record ID (rid)>

Index Classification (1)

- Primary and Secondary
  - Primary
    - Search key contains primary key.
  - Secondary
    - Otherwise

Index Classification (2)

- Dense and Sparse
  - Dense
    - K* appear for ALL search key
  - Sparse
    - Otherwise

Index Classification (3)

- Clustered and Unclustered
  - Clustered
    - Data entries and records are sorted by K.
    - Order of records are equal/close to the order of data entries in index.
    - Support range search.
  - Unclustered
    - Otherwise

Index Classification: an Example

- Employee = { <u>Name</u> , Office }

contains primary key

in the same order

Name
Bryan
Eric
Fanny
Raymond
Steve
Teddy

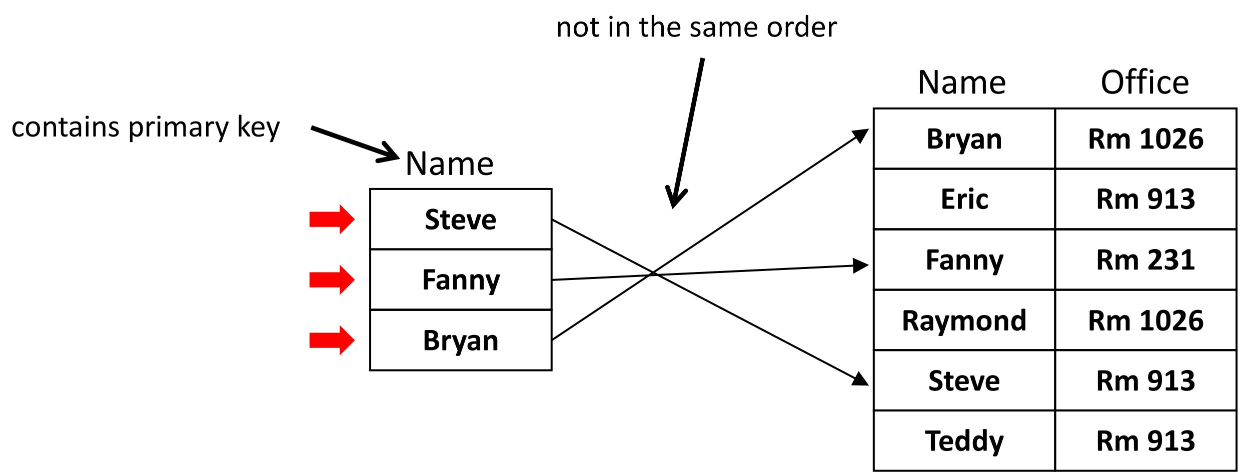
Name	Office
Bryan	Rm 1026
Eric	Rm 913
Fanny	Rm 231
Raymond	Rm 1026
Steve	Rm 913
Teddy	Rm 913

Primary	~~Secondary~~
Clustered	~~Unclusterd~~
Dense	~~Sparse~~

Name
Bryan ✓
Eric ✓
Fanny ✓
Raymond ✓
Steve ✓
Teddy ✓

Index Classification: an Example

- Employee = { Name , Office }



Primary	~~Secondary~~
~~Clustered~~	Unclusterd
~~Dense~~	Sparse

Name
Bryan
Eric
Fanny
Raymod
Steve
Teddy

Index Classification: an Example

- Employee = { <u>Name</u> , Office }

not in the same order

does not contain primary key

Office
Rm 1026
Rm 231

Name	Office
Fianny	Rm 231
Steve	Rm 913
Eric	Rm 913
Teddy	Rm 913
Raymond	Rm 1026
Bryan	Rm 1026

~~Primary~~	Secondary
~~Clustered~~	Unclusterd
~~Dense~~	Sparse

Name	
Rm 231	✓
Rm 913	✗
Rm1026	✓

Index Classification: an Example

- Employee = { Name , Office }

in the same order

does not contain primary key

Office
Rm 231
Rm 913
Rm 1026

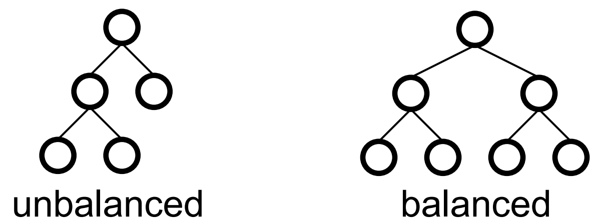
Name	Office
Fianny	**Rm 231**
Steve	**Rm 913**
Eric	**Rm 913**
Teddy	**Rm 913**
Raymond	**Rm 1026**
Bryan	**Rm 1026**

~~Primary~~	Secondary
Clustered	~~Unclusterd~~
Dense	~~Sparse~~

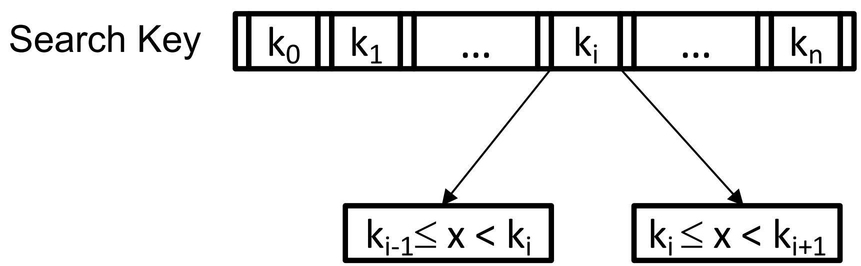
Name
Rm 231
Rm 913
Rm1026

Tree-structured Indexing

- B+ Tree Index Files
  - Balanced Tree – The length of every path from the root to a leaf is the same, maintaining efficient searching

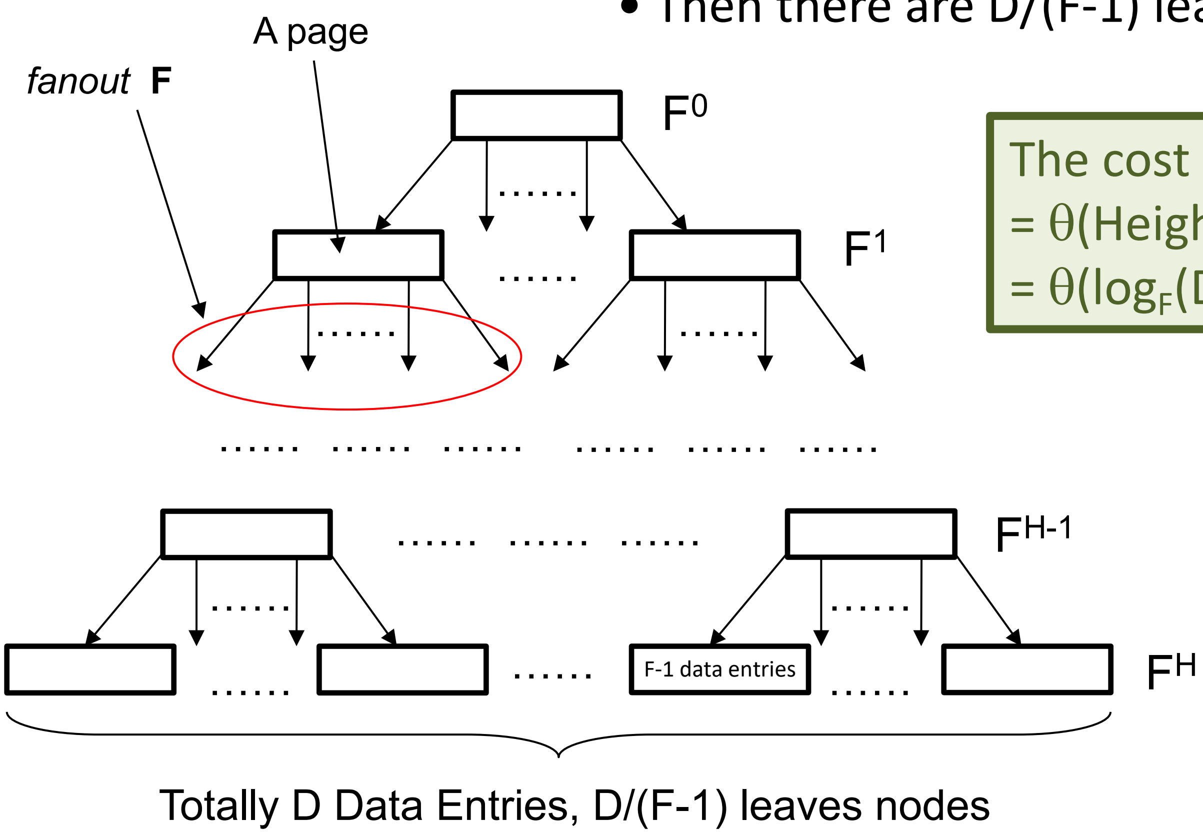


  - Support range search and equality selection



Search Cost of B+ Tree

- Let average number of pointers (*fanout*) is **F**
- Each leaf node can at most store F-1 data entries
- Suppose there are totally D data entries
- Then there are D/(F-1) leaf nodes



The cost to reach a leave node:
$= \theta(\text{Height of tree})$
$= \theta(\log_F(D/(F-1)))$

Insertion of B+ Tree

Original Tree

1*	4*	7*	

Insert 13

1*	4*	7*	13*

(No overflow)

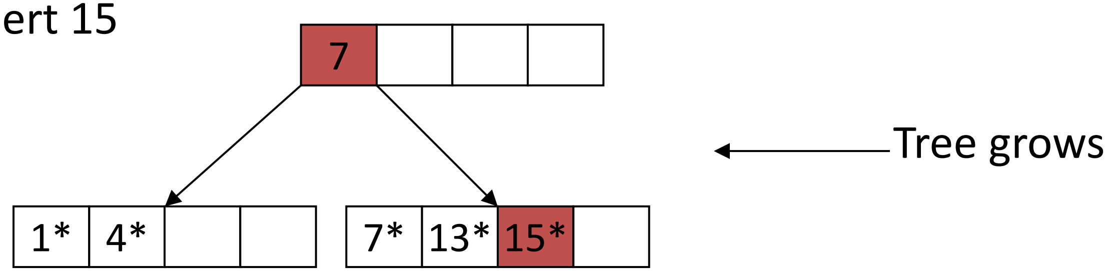
Insert 15

1*	4*	7*	13*	15*

(Overflow)

Leaf node overflow: split leaf node, **copy** middle **key** to the parent.

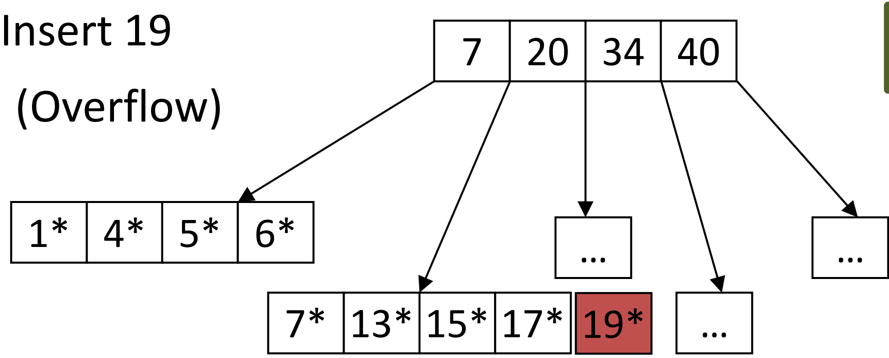
After Insert 15



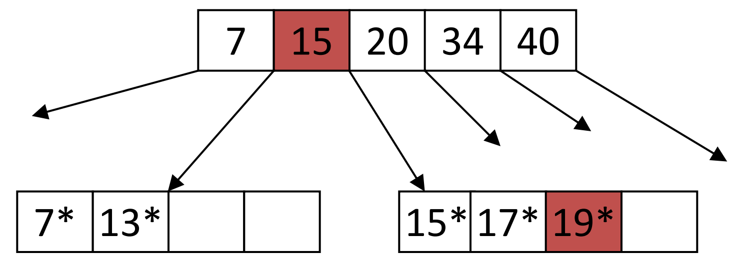
Insertion of B+ Tree

Insert 19

(Overflow)

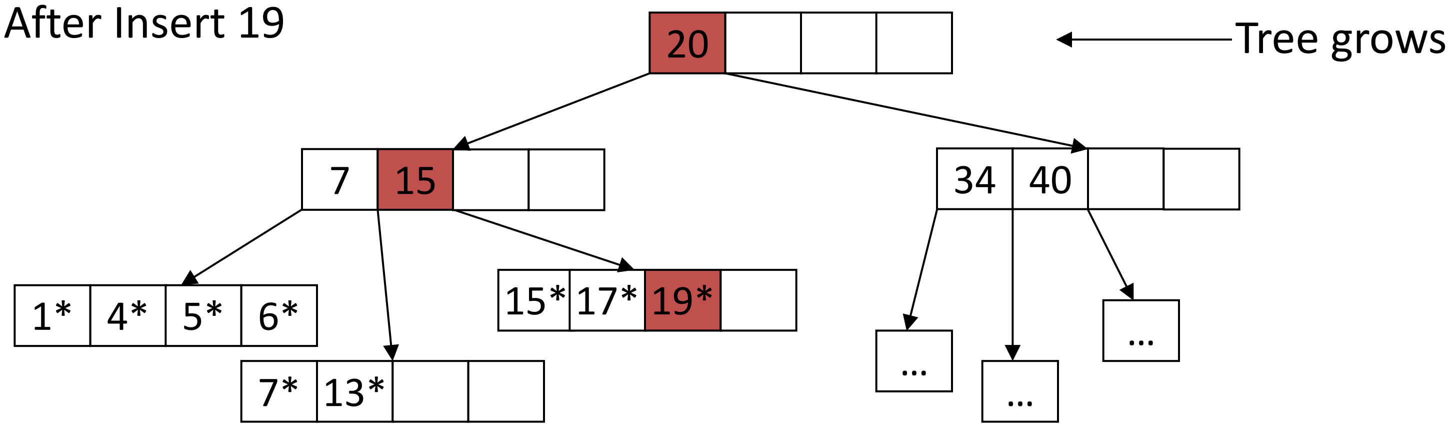


1. Leaf node overflow: split leaf node.



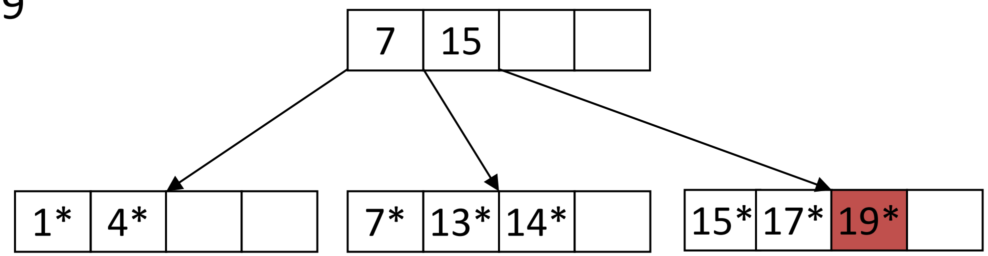
2. Non-leaf node overflow: split non-leaf node, **push** middle key up.

After Insert 19



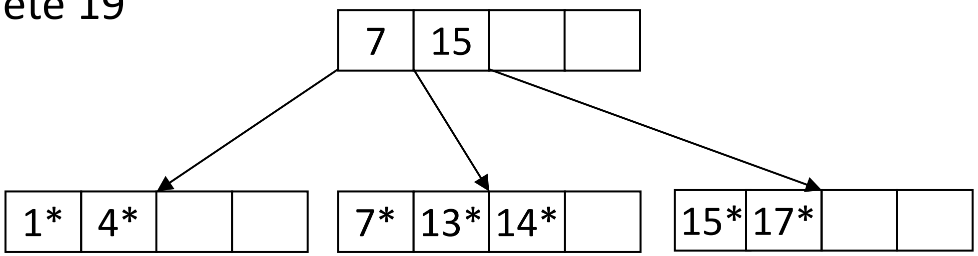
Deletion of B+ Tree

Delete 19



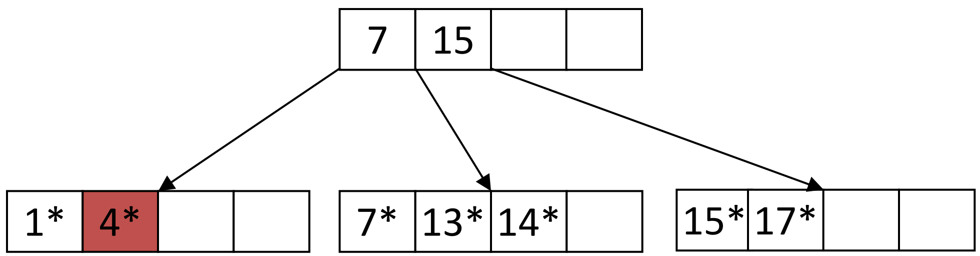
(No underflow)

After Delete 19



Deletion of B+ Tree

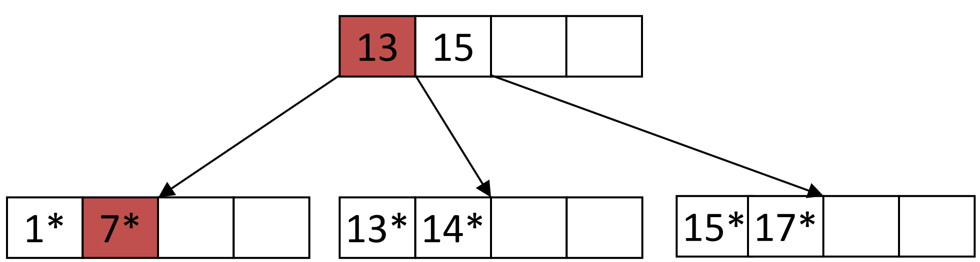
Delete 4



(Underflow)

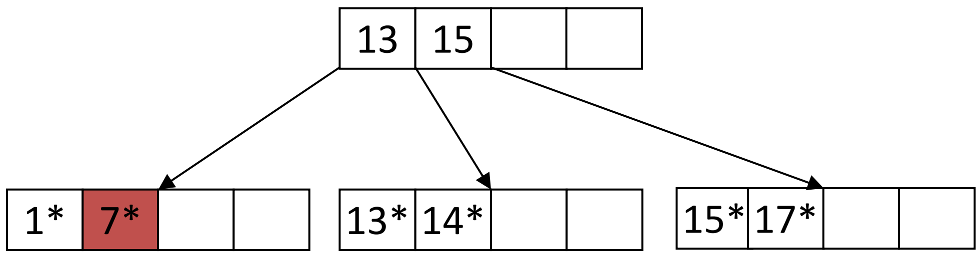
After delete 4, the page will underflow, need to borrow from sibling, update the parent.

After Delete 4



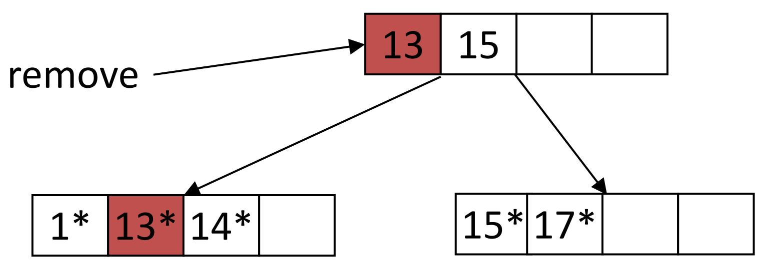
Deletion of B+ Tree

Delete 7



(Underflow)

After delete 7, the page will underflow, can't borrow from sibling, need to merge with sibling and update the parent.



After Delete 7