Indexing 5

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

Department of Computer Science

Overview

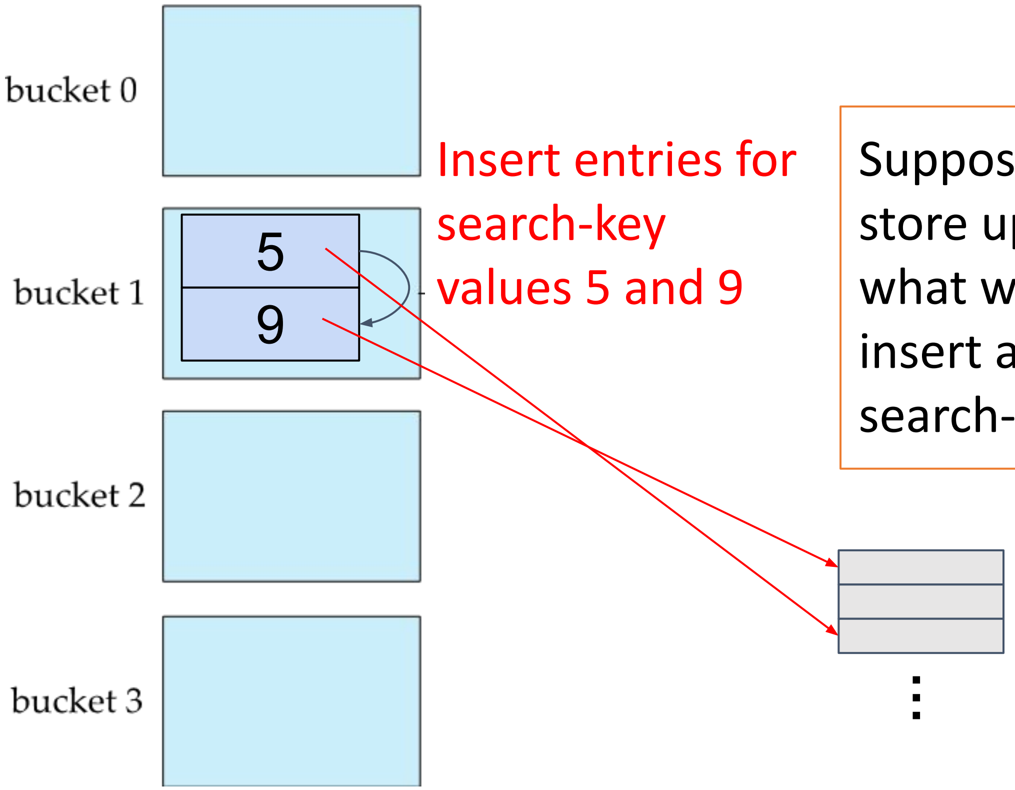
- Hash Indices
- Assignments

Static Hashing

- A hash index is the index that allows us to quickly find the bucket of an entry from its search-key value using a **hash function**
  - A **bucket** is a unit of storage containing one or more entries (a bucket is typically a disk block).
    - If the number of buckets is fixed when the index is created, such a hash indexing is called **static hashing**.
    - Otherwise, it is called **dynamic hashing**
  - Hash function *h* is a function from the set of all search-key values *K* to the set of all bucket addresses *B.*
  - Entries with different search-key values may be mapped to the same bucket; thus the entire bucket has to be searched sequentially to locate an entry.

Handling of Bucket Overflows

- Example static hashing where the bucket contains the actual record
  - Assume we use hash function h, such that h(x) = x % 4



Insert entries for search-key values 5 and 9

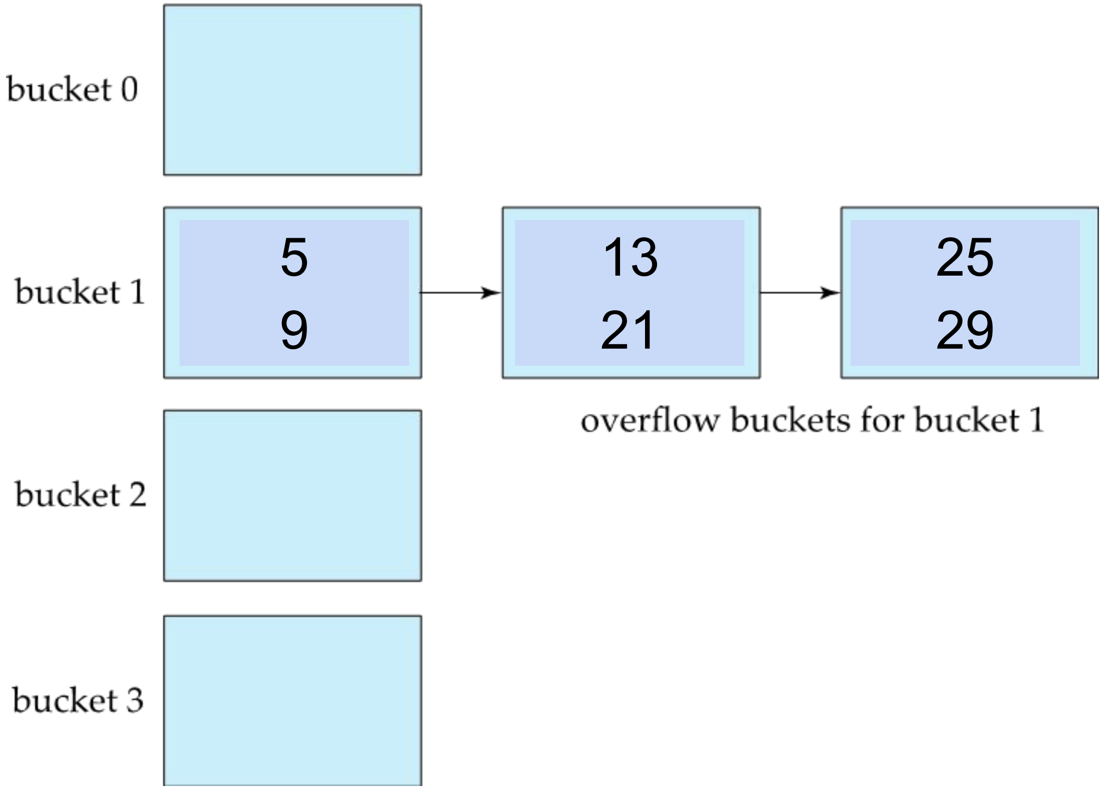
Suppose the bucket can store up to 2 records, what will happen if we insert an entry for search-key value 13?

Handling of Bucket Overflows (Cont.)

- 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***.

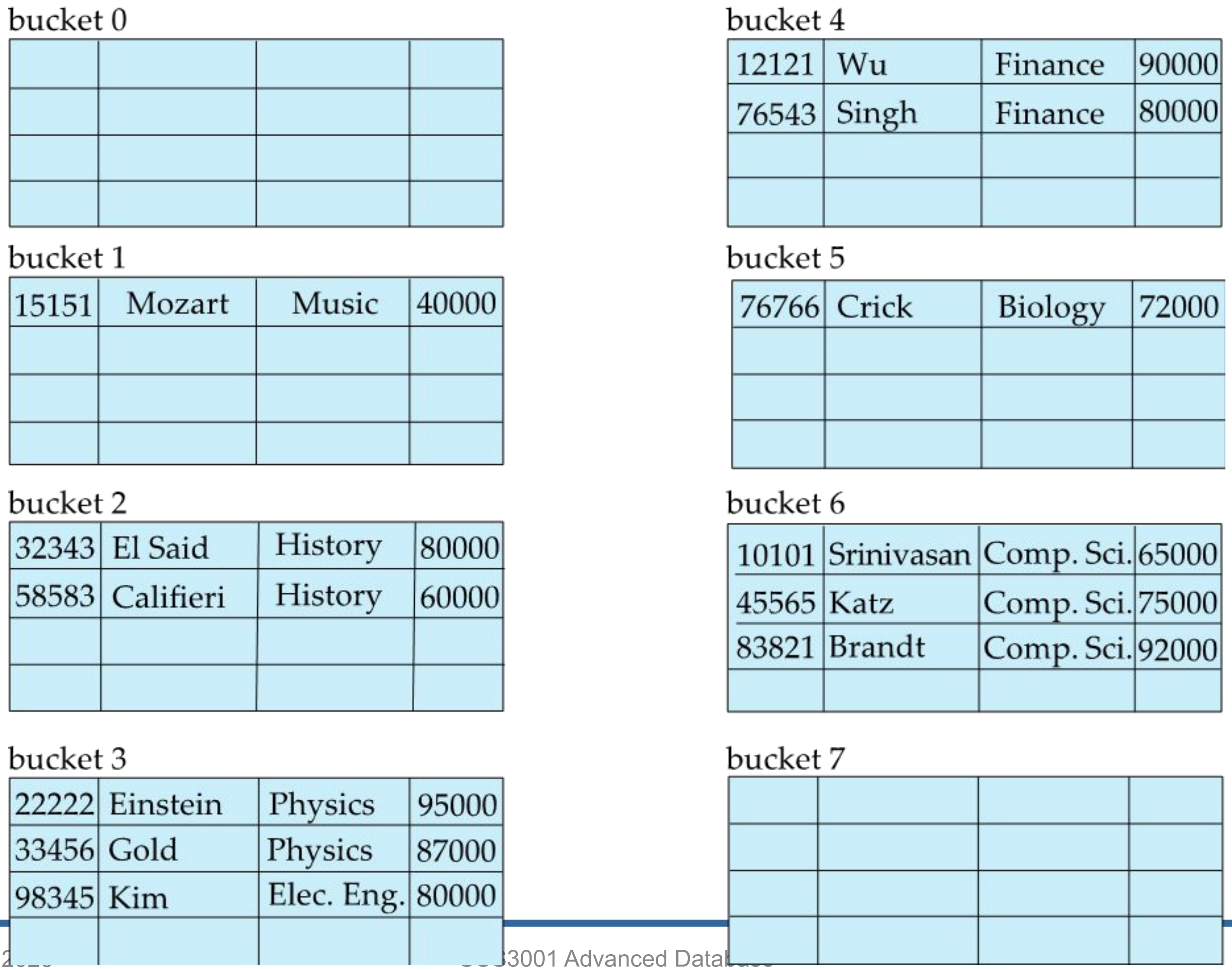
Handling of Bucket Overflow (Cont.)

- **Overflow chaining** – the overflow buckets of a given bucket are chained together in a linked list.
- Above scheme using overflow chaining is called **closed addressing** (or, less commonly, **closed hashing)**
  - ○ An alternative, called **open addressing** which does not use overflow buckets, is not suitable for database applications.



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

Deficiencies of Static Hashing

- In static hashing, function *h* maps search-key values to a fixed set of *B* of bucket addresses. Databases grow or shrink with time.
  - If initial number of buckets is too small, and file grows, performance will degrade due to too much overflows.
  - If space is allocated for anticipated growth, a significant amount of space will be wasted initially (and buckets will be underfull).
  - If database shrinks, again space will be wasted.
- One solution: periodic re-organization of the file with a new hash function
  - Expensive, disrupts normal operations
- Better solution: Dynamic hashing (refer to 24.5). Allow the number of buckets to be modified dynamically
  - Linear Hashing
  - Extendable Hashing

Comparison of Ordered Indexing and Hashing

- Expected type of queries:
  - Hashing is generally better at retrieving records having a specified value of the key.
  - If range queries are common, ordered indices are to be preferred
- In practice:
  - PostgreSQL supports hash indices, but discourages use due to poor performance
  - Oracle supports static hash organization, but not hash indices
  - SQLServer supports only B^+-trees

Overview

- Hash Indices
- Assignments

Assignments

- Reading: 14.5

The End