CSCI3170 Introduction to Database Systems

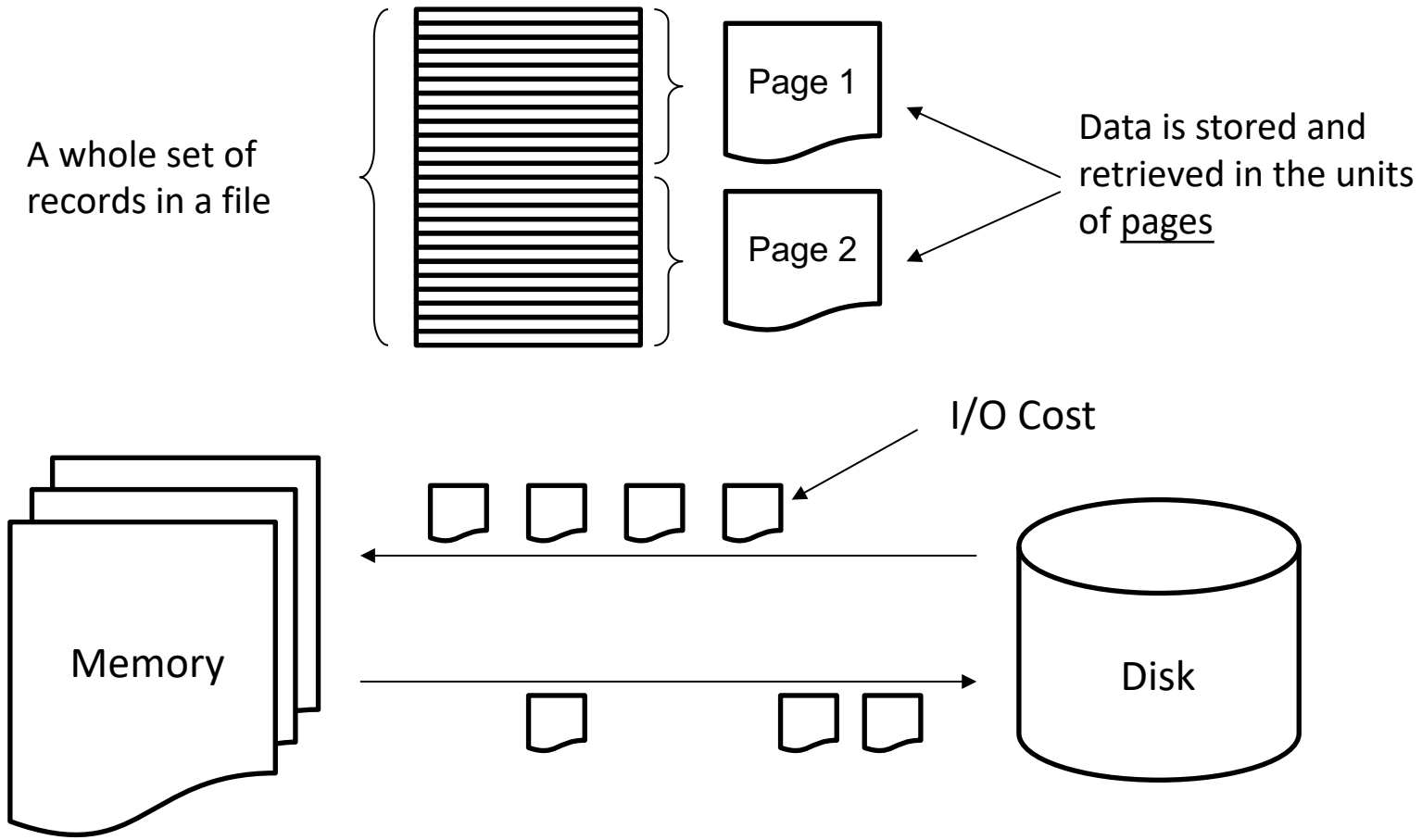
Tutorial 9 – Storage and Indexes (2)

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)

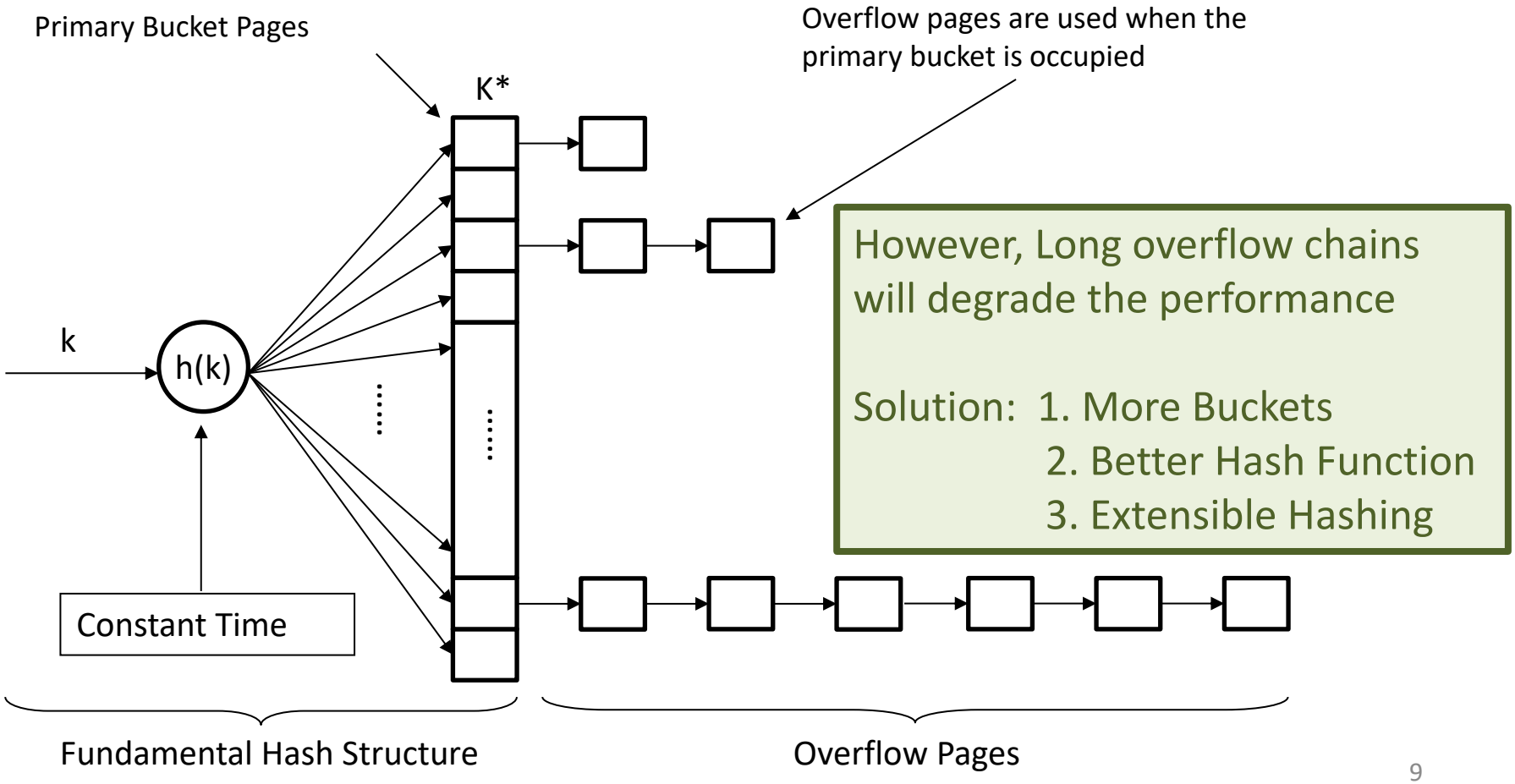
- Clustered and Unclustered
  - Clustered
    - Data entries and records are sorted by K.
    - Order of records are equal/close to the order of data entries in index.
    - Support range search.
  - Unclustered
    - Otherwise

Hashing

- Properties
  - Data entries are kept in bucket.
  - Hashing function h(k) = address of the bucket.
  - h(k) should distribute K* uniformly.
  - Best for equality selection.
  - Not support range search.
  - Constant time retrieval.

Static Hashing

No. of primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.



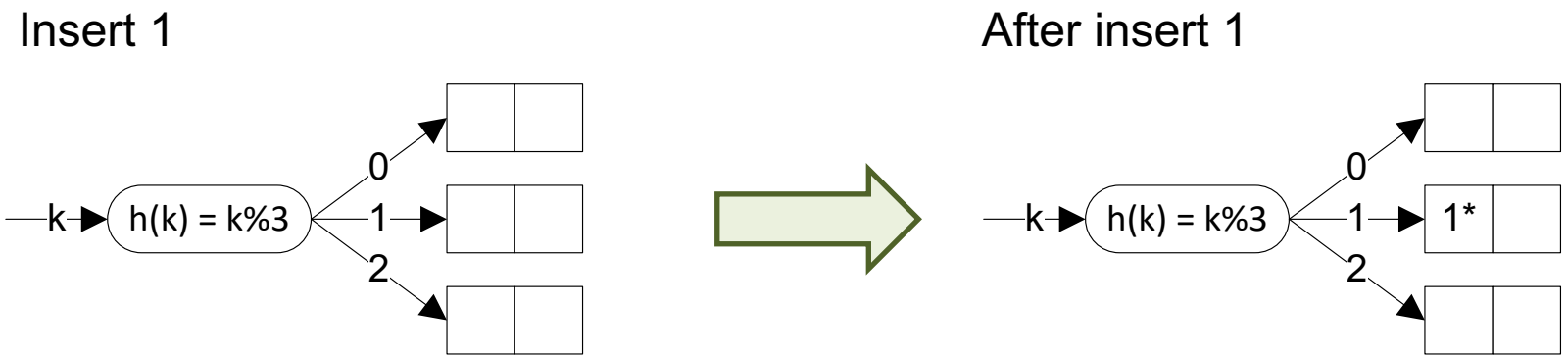
However, Long overflow chains will degrade the performance

Solution: 1. More Buckets
2. Better Hash Function
3. Extensible Hashing

Static Hashing: an example

- Suppose the hash function is **h(x) = x mod 3** and a bucket can hold at most 2 data entries. Below show a static hash structure after inserting 1, 4, 5, 7, 8, 2.

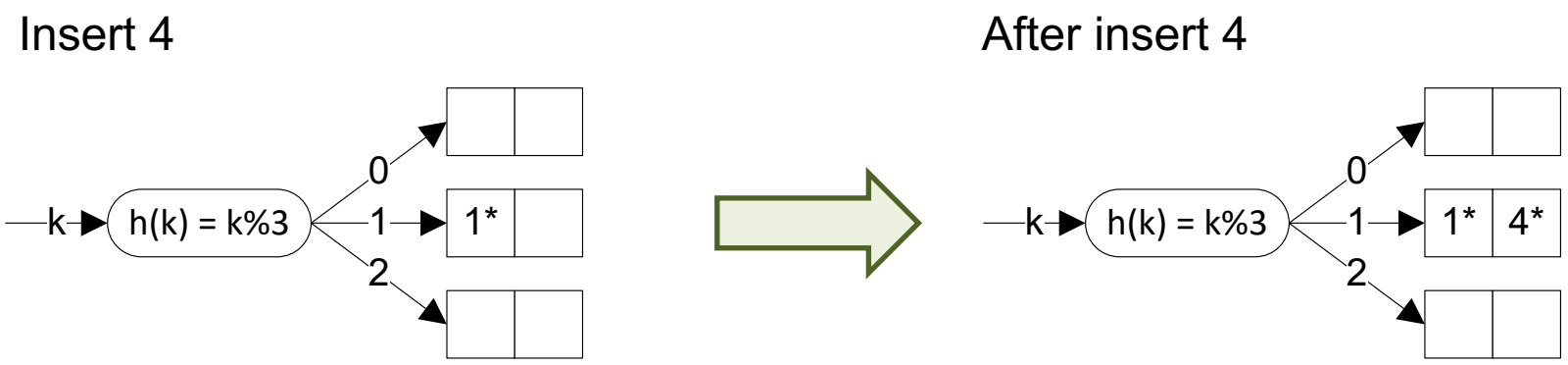
h(1) = 1



Static Hashing: an example

- Suppose the hash function is **h(x) = x mod 3** and a bucket can hold at most 2 data entries. Below show a static hash structure after inserting 1, 4, 5, 7, 8, 2.

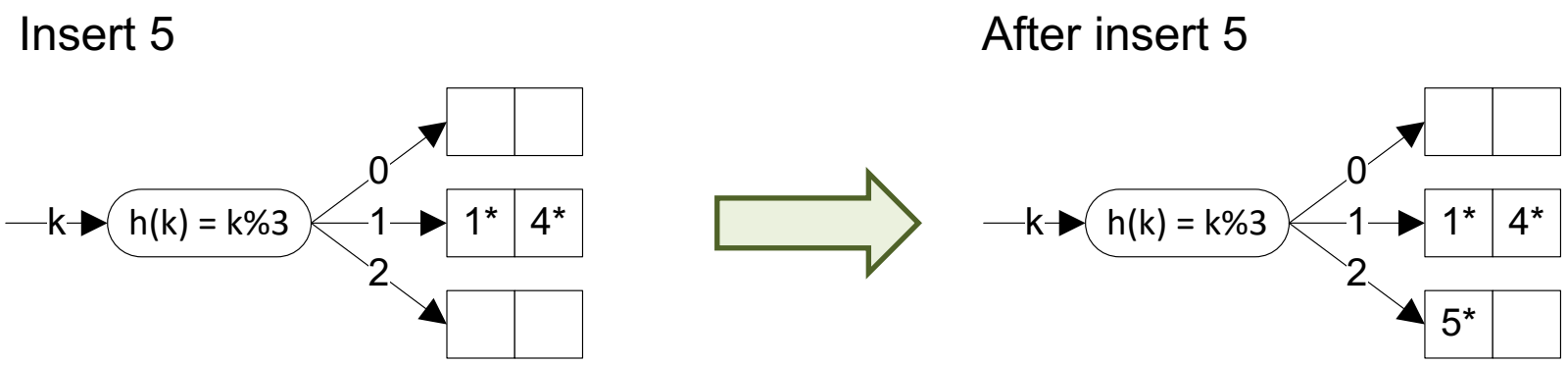
h(4) = 1



Static Hashing: an example

- Suppose the hash function is **h(x) = x mod 3** and a bucket can hold at most 2 data entries. Below show a static hash structure after inserting 1, 4, 5, 7, 8, 2.

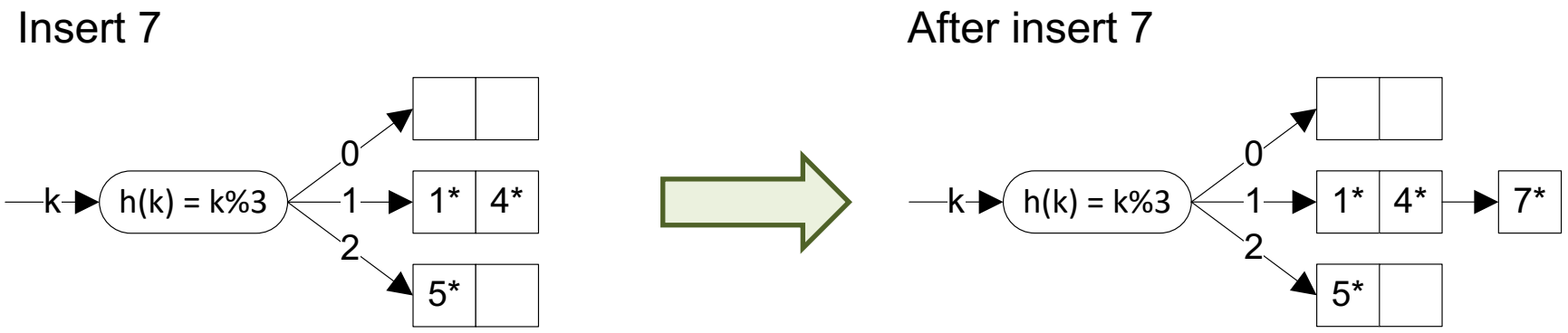
h(5) = 2



Static Hashing: an example

- Suppose the hash function is **h(x) = x mod 3** and a bucket can hold at most 2 data entries. Below show a static hash structure after inserting 1, 4, 5, 7, 8, 2.

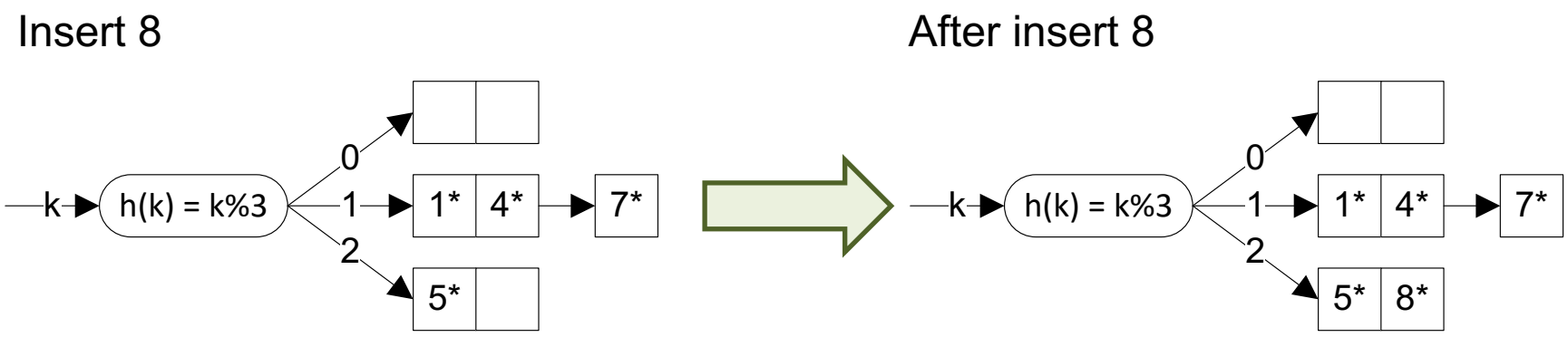
h(7) = 1



Static Hashing: an example

- Suppose the hash function is **h(x) = x mod 3** and a bucket can hold at most 2 data entries. Below show a static hash structure after inserting 1, 4, 5, 7, 8, 2.

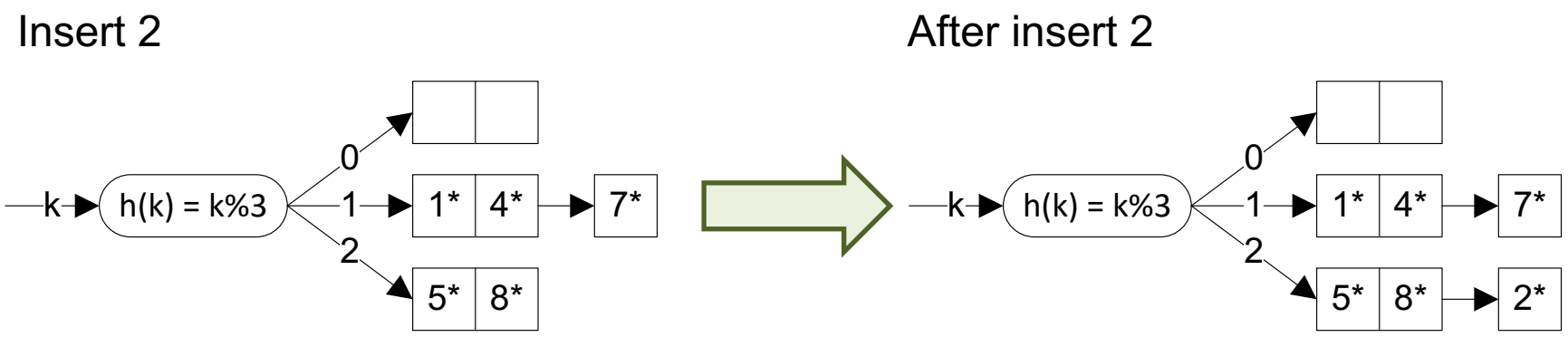
h(8) = 2



Static Hashing: an example

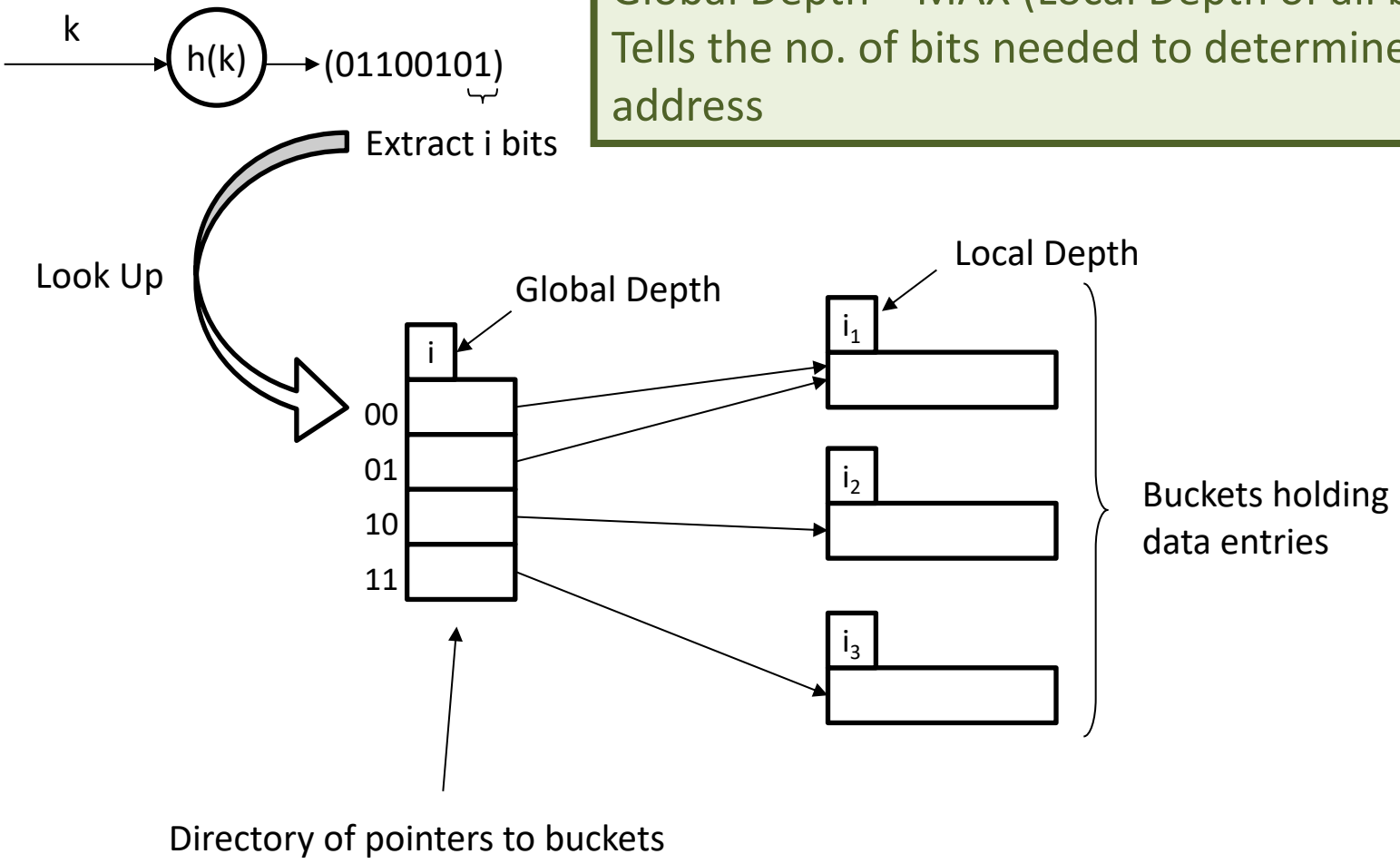
- Suppose the hash function is **h(x) = x mod 3** and a bucket can hold at most 2 data entries. Below show a static hash structure after inserting 1, 4, 5, 7, 8, 2.

h(2) = 2



Extensible Hashing

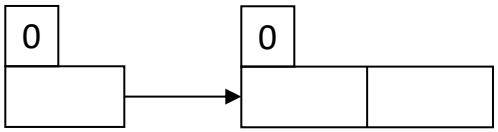
Global Depth = MAX (Local Depth of all buckets)
Tells the no. of bits needed to determine the address



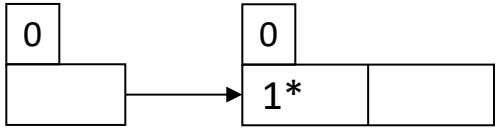
Insertion of Extensible Hashing

- Suppose the hash function is h(x) = x mod 8 and each bucket can hold at most 2 data entries. Below show a extendable hash structure after inserting 1, 4, 5, 7, 8, 2.

Initial:



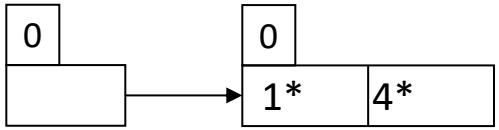
After insert 1:



(ok)

h(1) = 001

After insert 4:

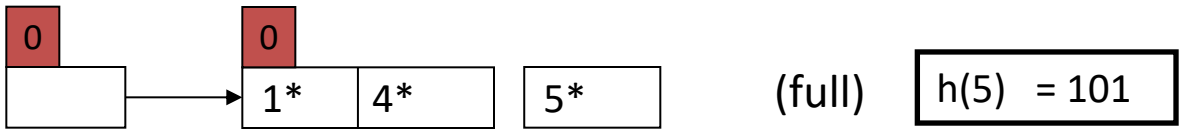


(ok)

h(1) = 001
h(4) = 100

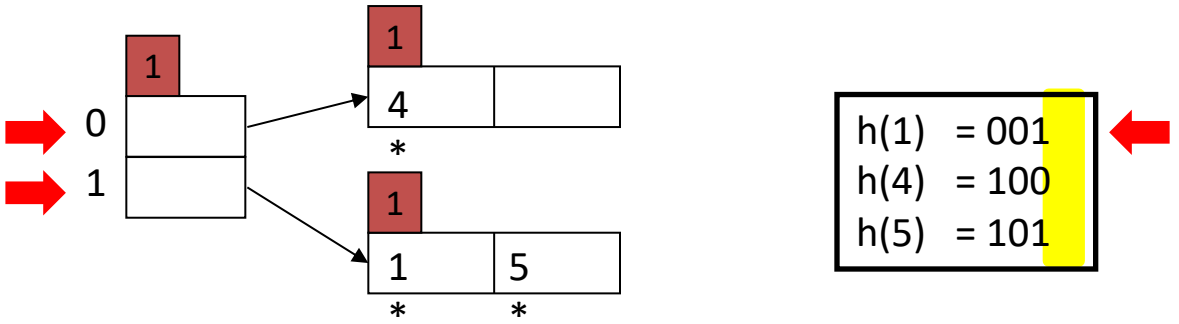
Insertion of Extensible Hashing

Insert 5:



If bucket with local depth = global depth, then double the directory and split the bucket.

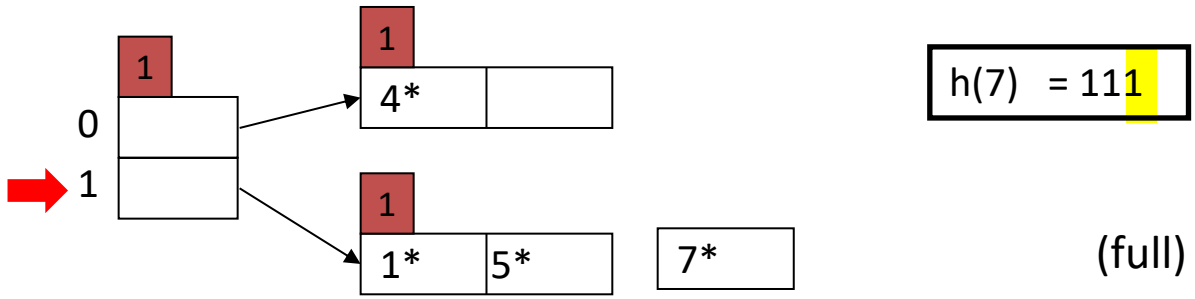
After insert 5:



Increase the global depth and local depth of the new buckets by 1.

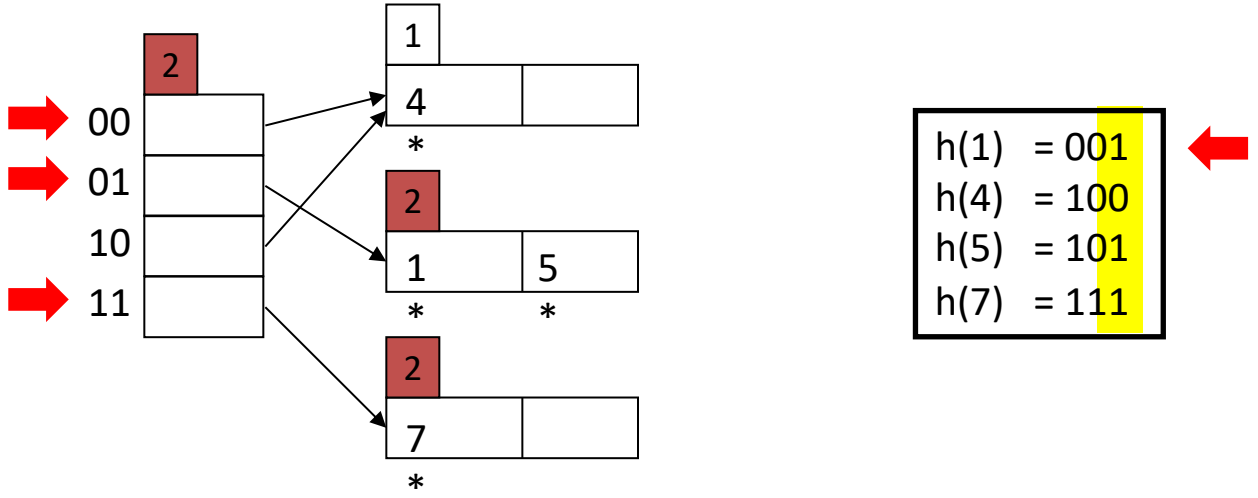
Insertion of Extensible Hashing

Insert 7:



If bucket with local depth = global depth, then double the directory and split the bucket.

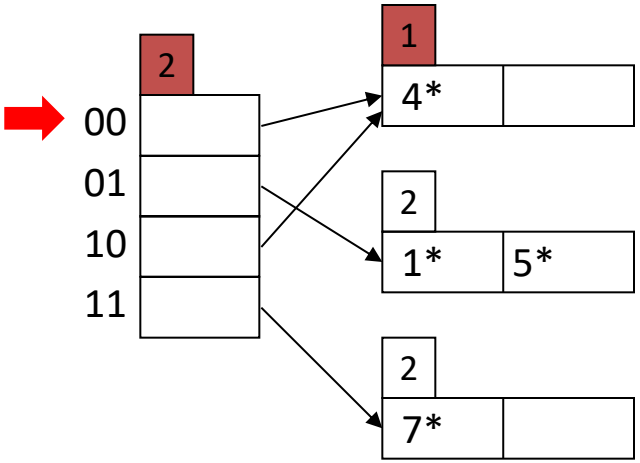
After insert 7:



Increase the global depth and local depth of the new buckets by 1.

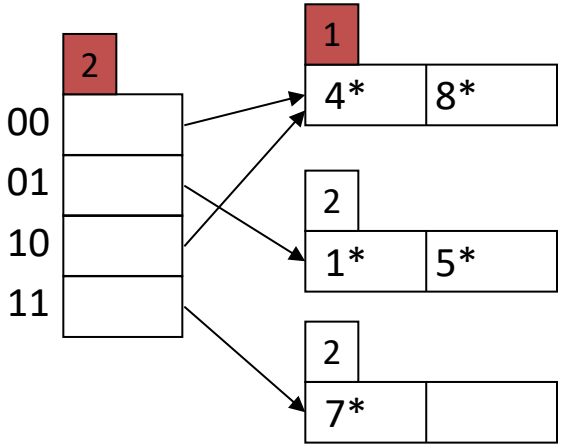
Insertion of Extensible Hashing

Insert 8:



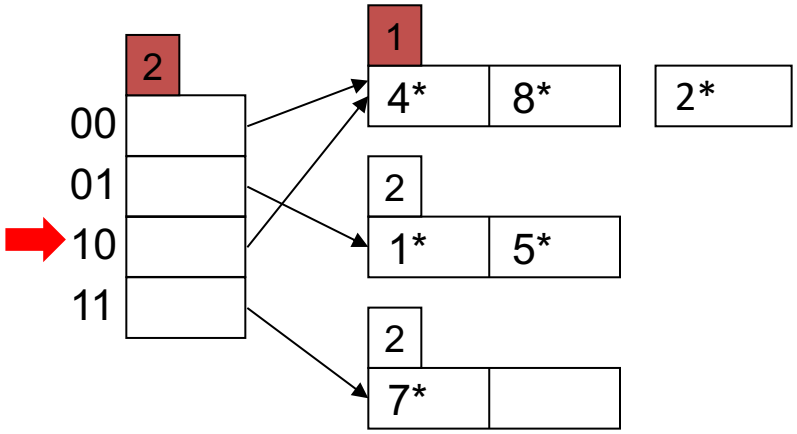
h(8) = 000

After insert 8:



Insertion of Extensible Hashing

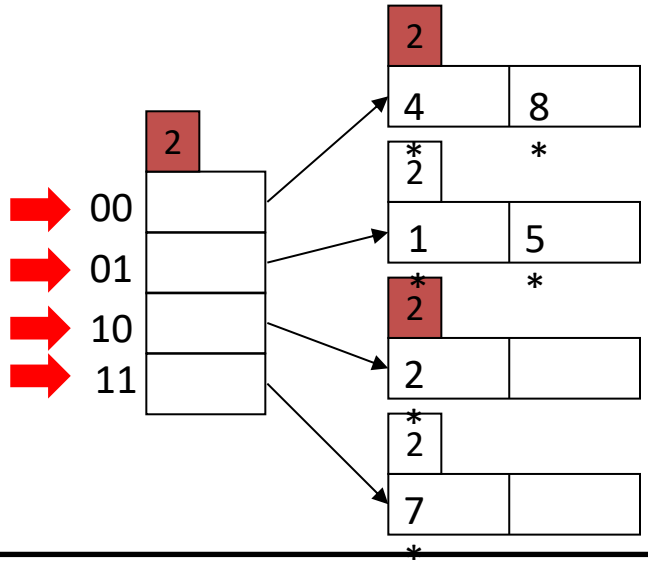
Insert 2:



h(2) = 010

If bucket with local depth < global depth, then split the bucket and update the pointers.

After insert 2:

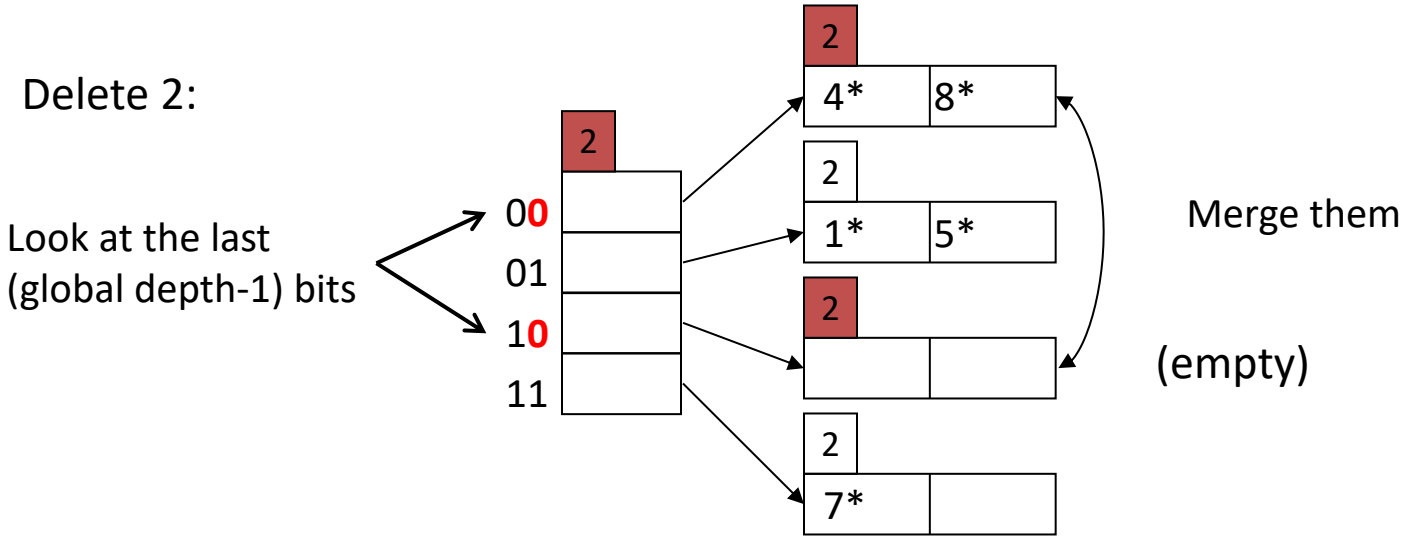


h(1) = 0001
h(2) = 0010
h(4) = 0100
h(5) = 0101
h(7) = 0111
h(8) = 1000

Increase the local depth of the new buckets by 1.

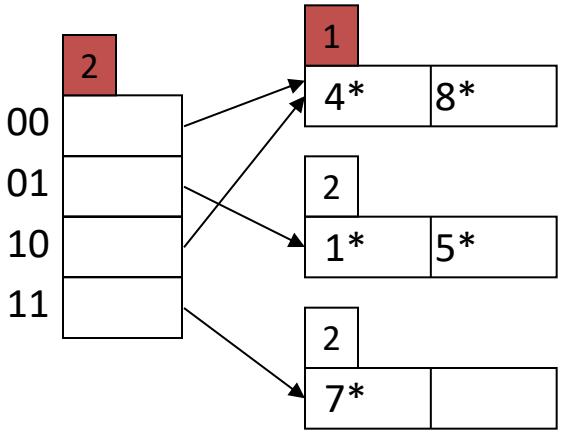
Deletion of Extensible Hashing

Delete 2:



If the removal makes the bucket empty, then remove the bucket and update the pointer.

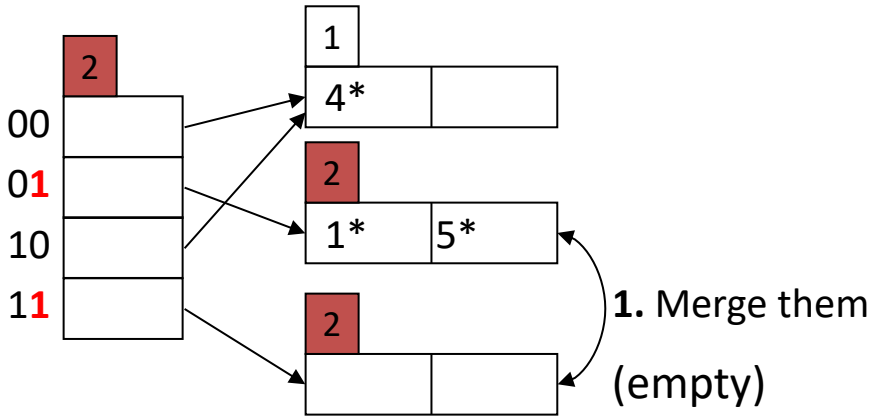
After delete 2:



Decrease the local depth of the new bucket by 1.

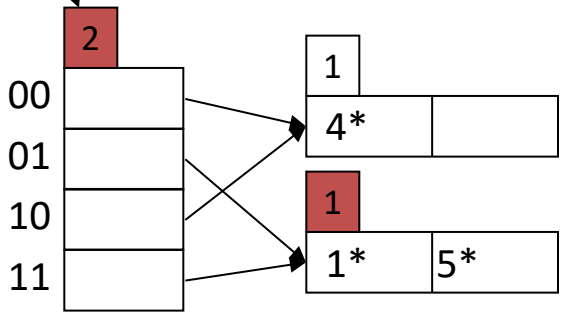
Deletion of Extensible Hashing

Delete 7:



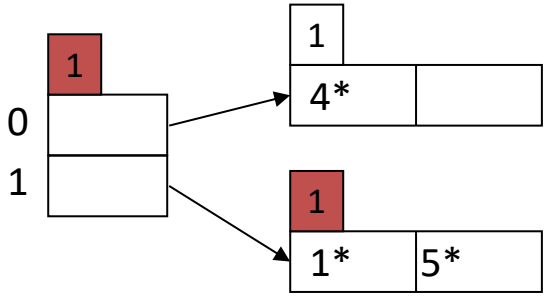
1. Merge them

2. Halve the directory



If the mergence makes max(local depth of all buckets) < global depth, then halve the size of directory.

After delete 7:



Decrease the global depth of the new bucket by 1.