Recitation for indexing

Part1 Btree

Storage Types

- Cache
  - Fastest/most costly; volatile;
- Main Memory
  - Fast access; too small for entire db; volatile
- Disk
  - Long-term storage of data; random access; non-volatile
- Flash Memory
  - No seeks (cheap reads);

How Disk Works

- ➔ Surface divided into tracks -> sectors (smallest unit of data that can be R/W)
    - ◆ Disk arm swings to position head on right track
    - ◆ Platter spins continually as data is R/W from sector
- ➔ Measuring Disk Speed
    - ◆ Access Time:
        - Seek Time: time to find right track
        - Latency Time: time for sector to appear under head
    - ◆ Data Transfer Rate:
        - The rate at which data can be retrieved
- ➔ Sequential I/O < Random I/O

Problem Statement

- ➔ Many queries reference only a fraction of records
- ➔ It is inefficient for the system to read every single tuple
- ➔ A better design is for the system to locate reference directly

Index

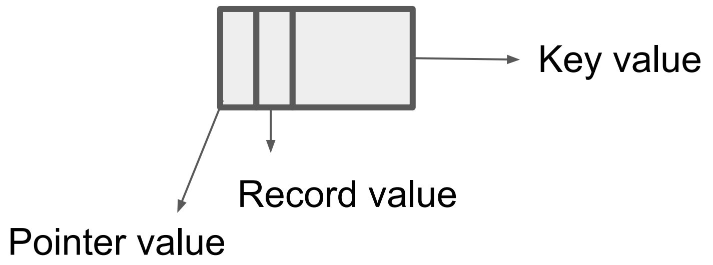
- Indexes
  - Auxiliary data structures over relations that can improve the search time
  - Think about index in a textbook
  - We search for an index, find corresponding pages, and then read information to find what we are looking for
  - The index is much smaller than the book and has words sorted in alphabetical order

B-trees

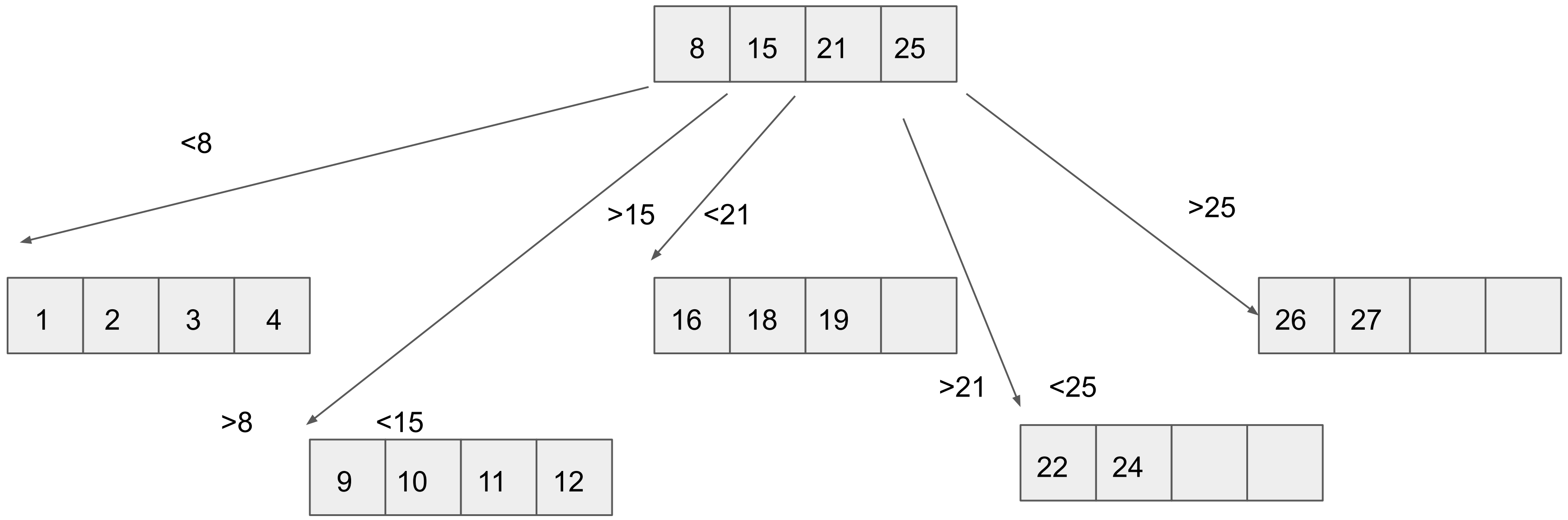
- Most successful family of index schemes (B-trees, B+-trees,...)
- Can be used for primary/secondary, clustering/non-clustering index
- Balanced “n-way” search trees
  - Helpful to think of it as a tree structure with n-pointers in each node

B-tree nodes

- Key values are ordered
- MAXIMUM: n pointers
- MINIMUM: [n/2] pointers
  - Exception: root’s minimum = 2
- Internal nodes must have n-1 key values where n is the number of pointers



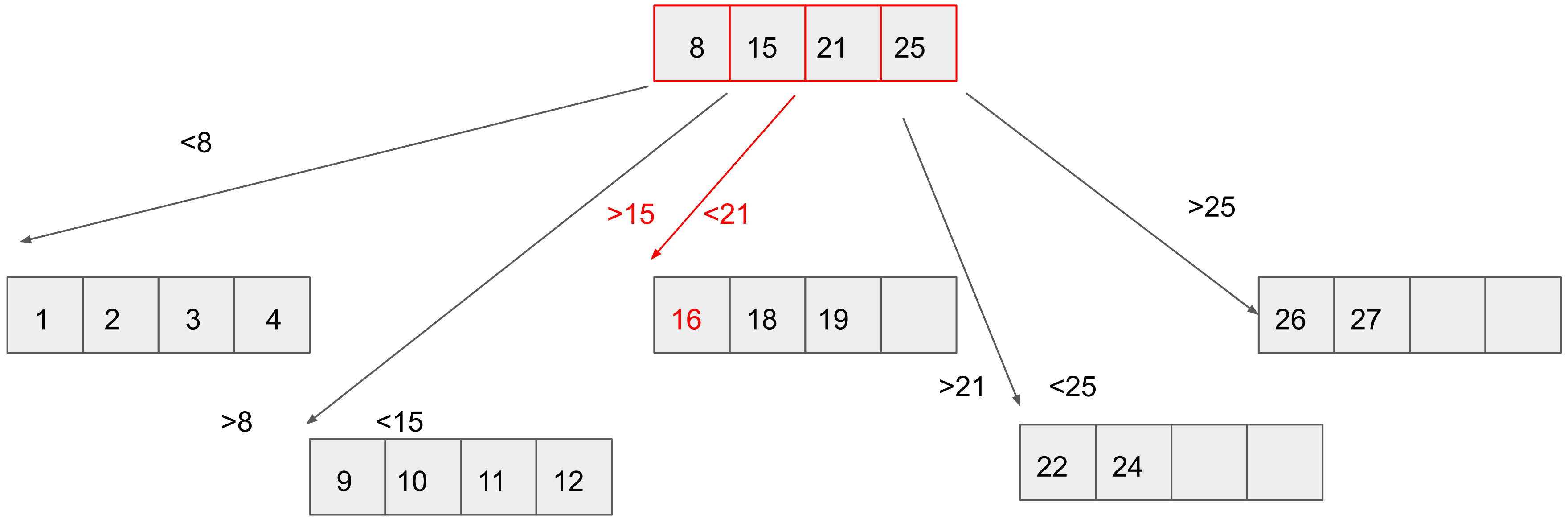
B-tree of order 5



- ➔ Key values appear once
- ➔ Record pointers accompany keys

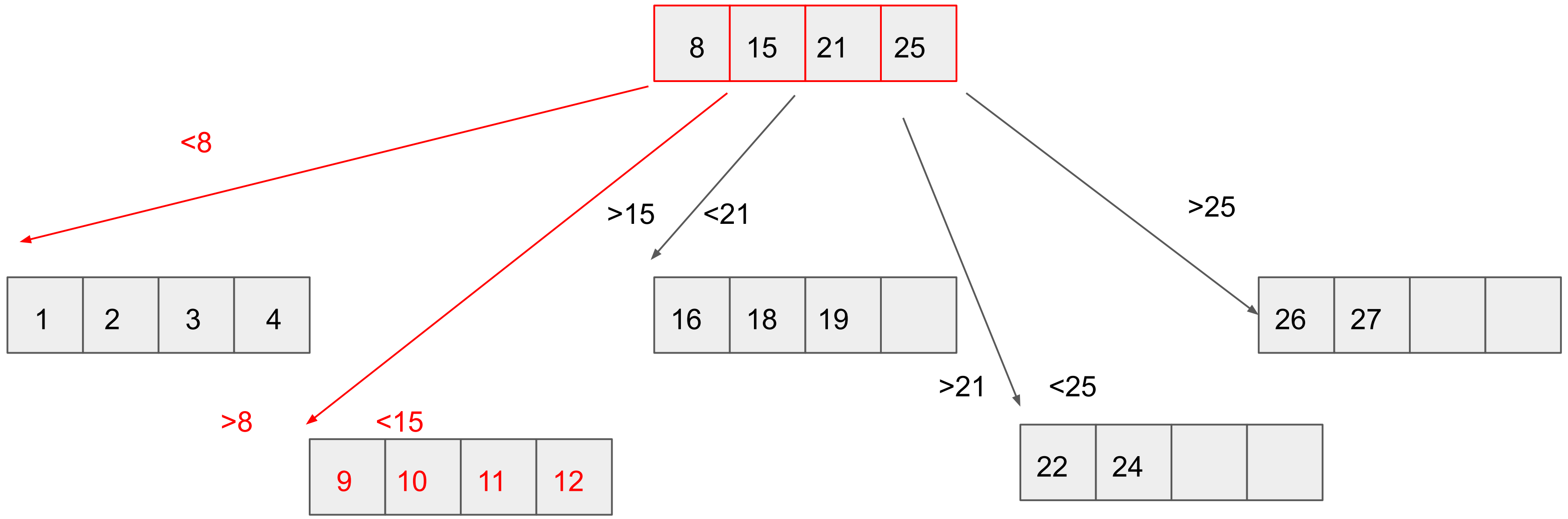
Queries

- ➔ Let’s run through an exact match query
  - ◆ Let’s find the value 16?



Queries

- ➔ Let's run through a range query
  - ◆ Let's find the values between 6 and 13?

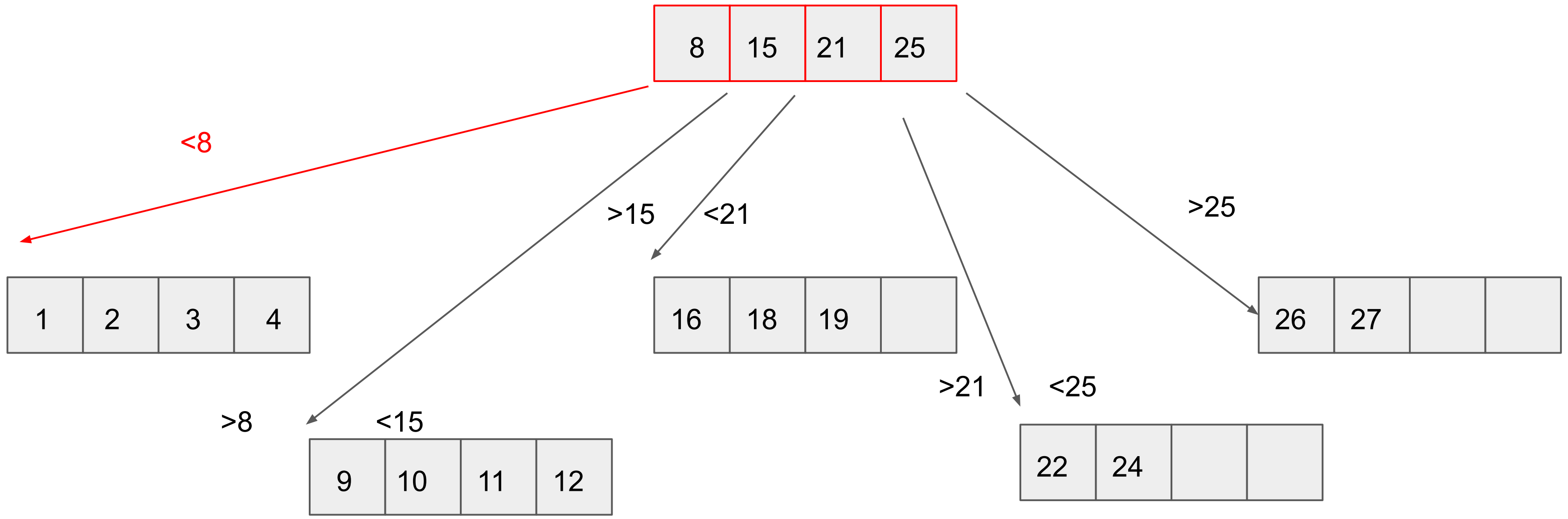


How To Maintain B-trees?

- B-tree rules must be obeyed on every insert + delete
- Rules:
  - Insert in leaf, if room exists
  - On overflow (no more room)
    - Split: create a new internal node
    - Redistribute keys
      - So that it preserves B-tree properties
      - Push middle key up (recursively)

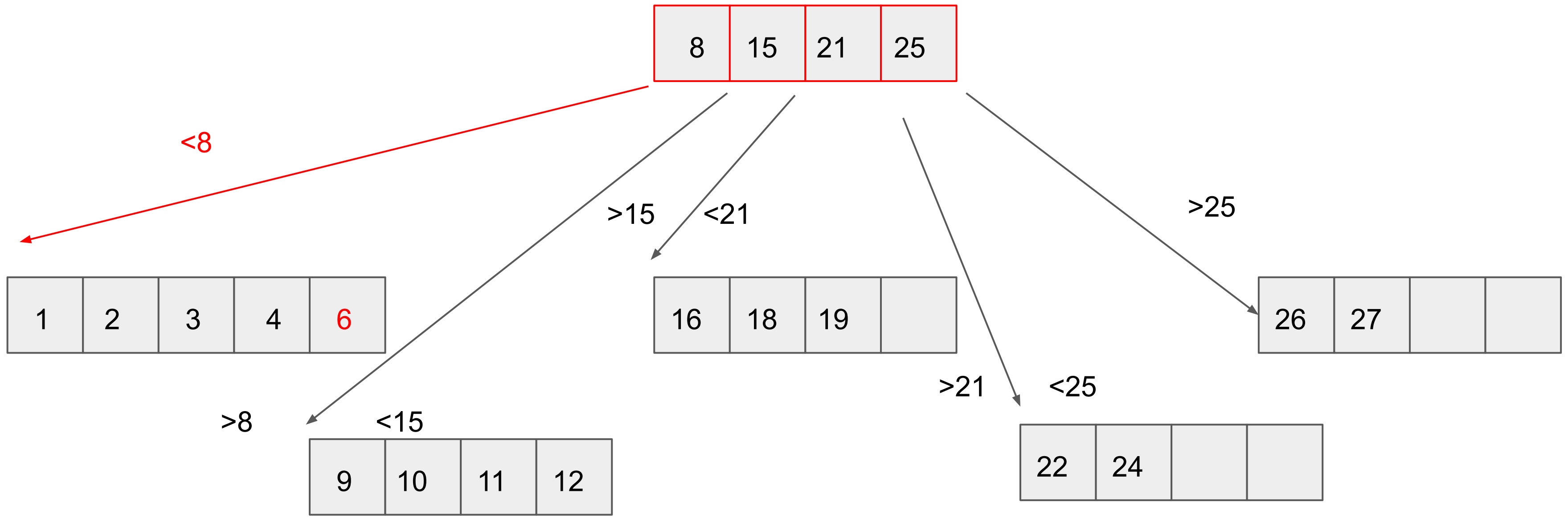
Queries

- Let’s run through an overflow insertion
  - Let’s insert the value 6:



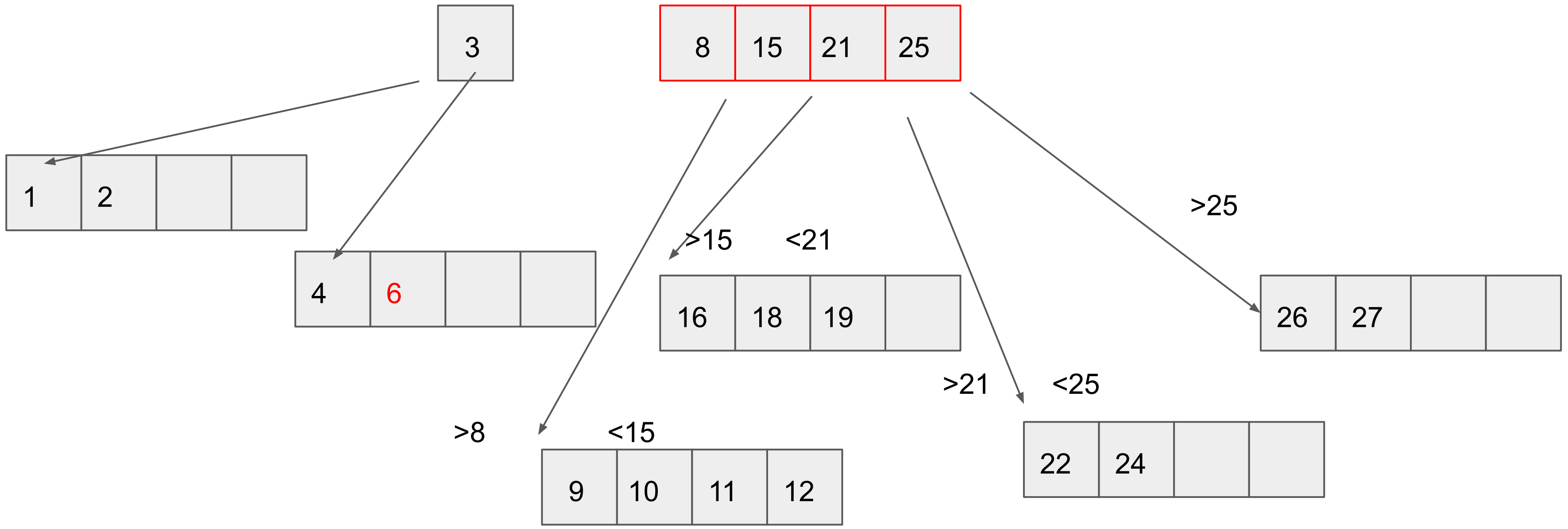
Queries

- Let’s run through an overflow insertion
  - Let’s insert the value 6:



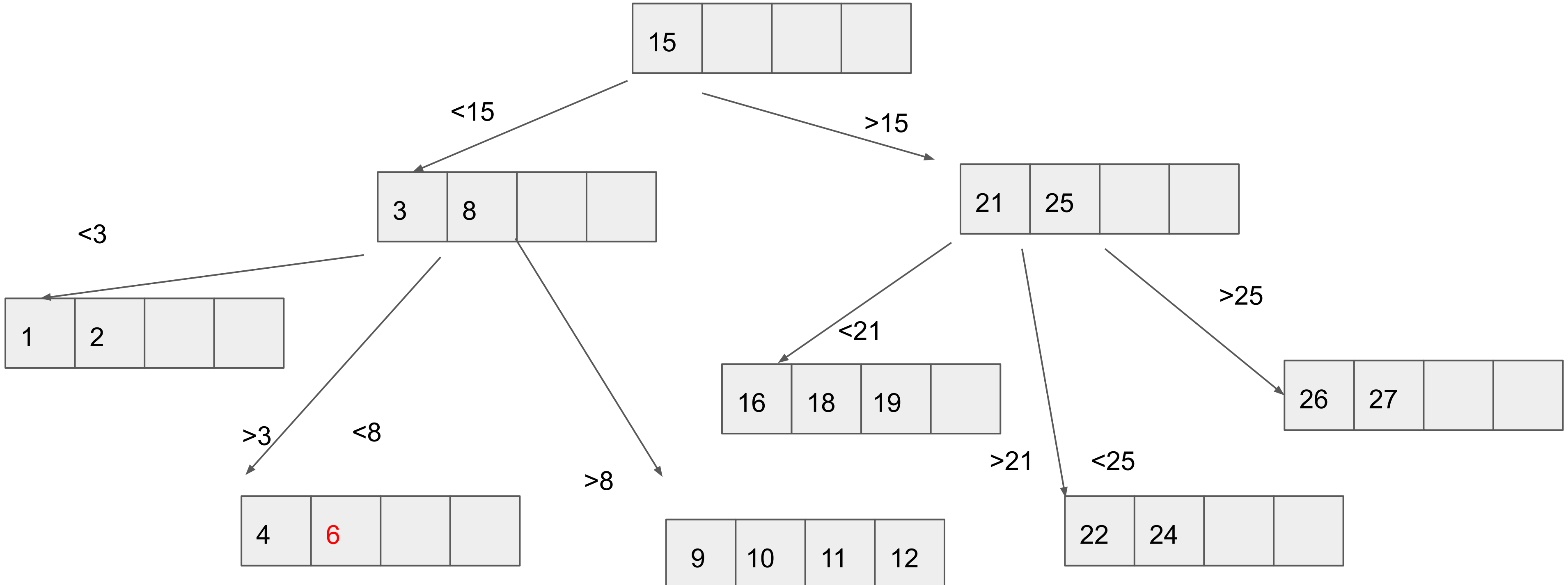
Queries

- Let’s run through an overflow insertion
  - Let’s insert the value 6:



Queries

- Let’s run through an overflow insertion
  - Let’s insert the value 6:

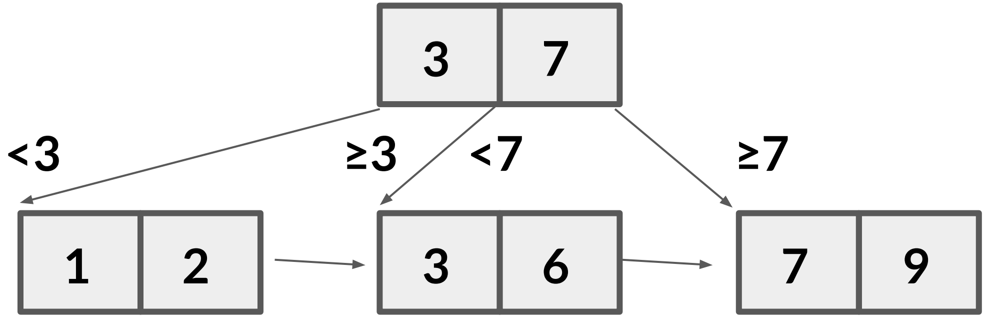


B+-trees

- ➔ Allow sequential operations
  - ◆ String all leaf nodes together
  - ◆ Every key appears at leaf level (some keys show up more than once)
- ➔ This mean non-leaf nodes do not contain data for the key, only the pointer value and key value

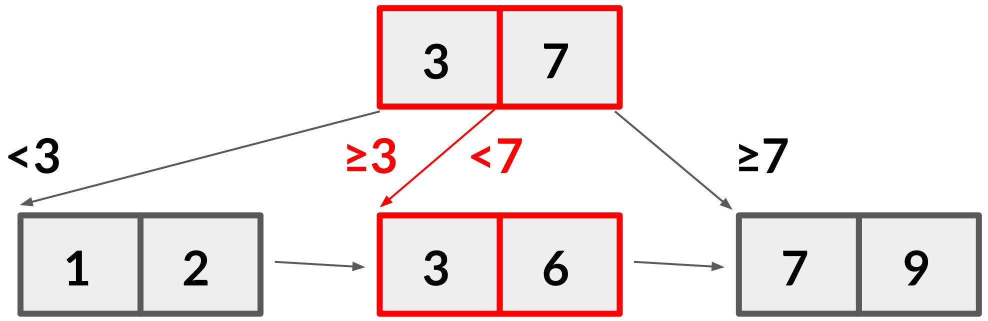
B+-tree Insertion

➔ Insert 5 into B-tree of order 3



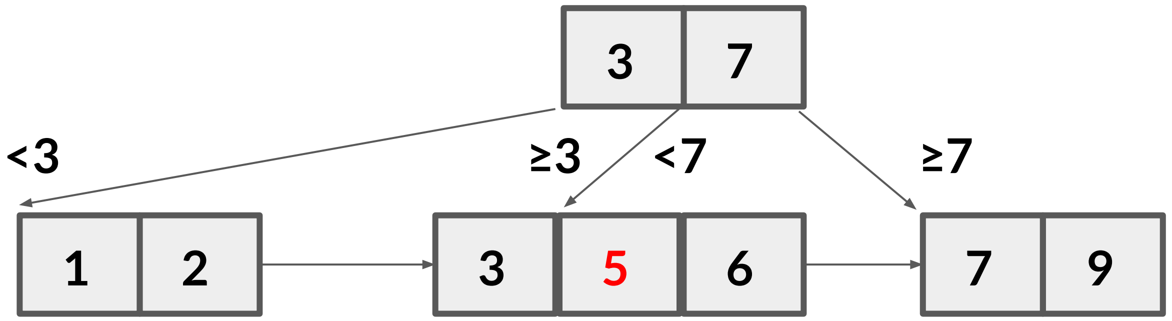
B+-tree Insertion

➔ Insert 5



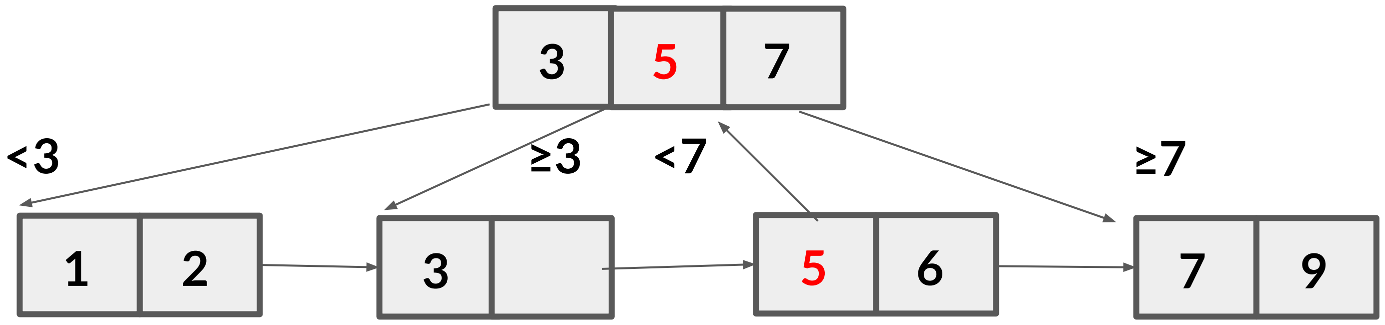
B+-tree Insertion

➔ Insert 5



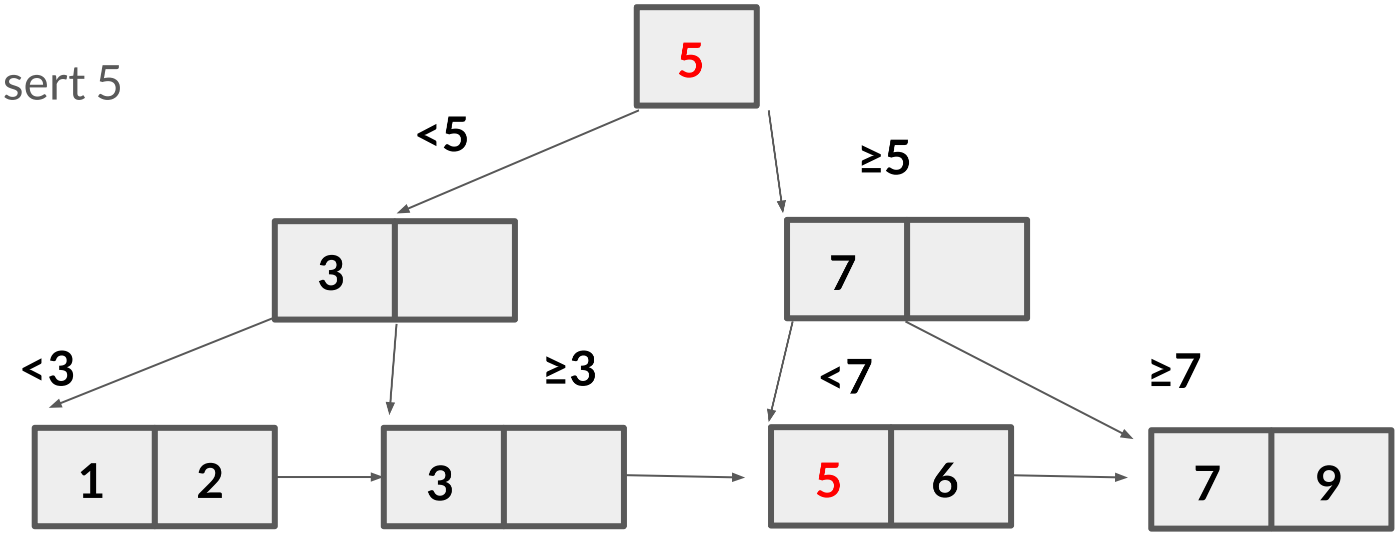
B+-tree Insertion

- ➔ Insert 5



B+-tree Insertion

- Insert 5



Recitation for indexing

Part-2 hashing

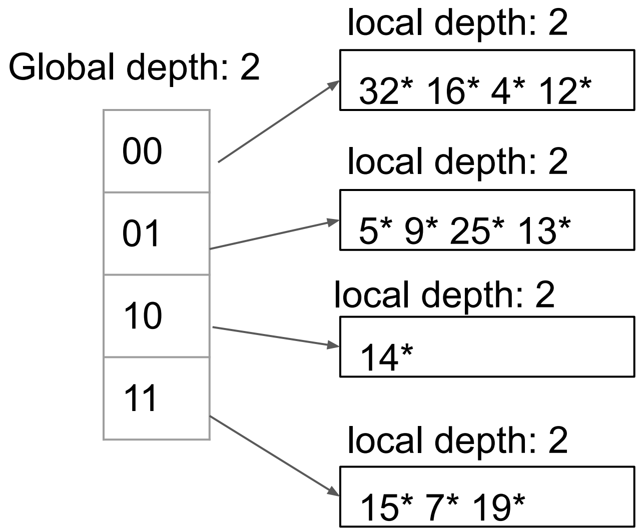
When do we need hashing?

Hash-based indexes are best for equality selections. Cannot support range searches.

Two kinds of hashing:

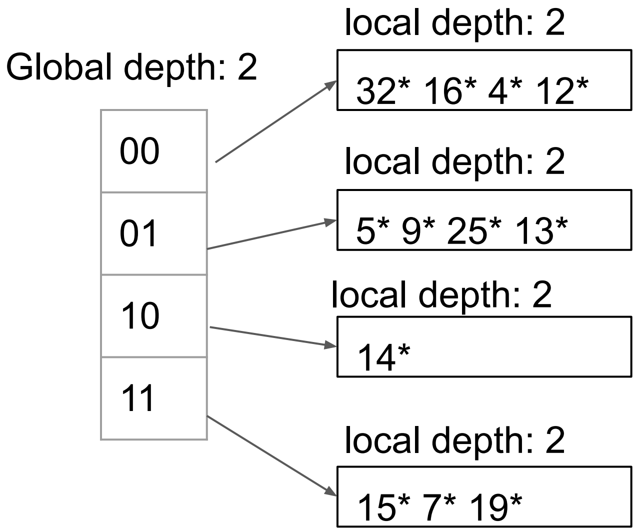
- Static hashing: will degrade performance with long overflow chains
- Dynamic hashing: Extendible hashing and linear hashing

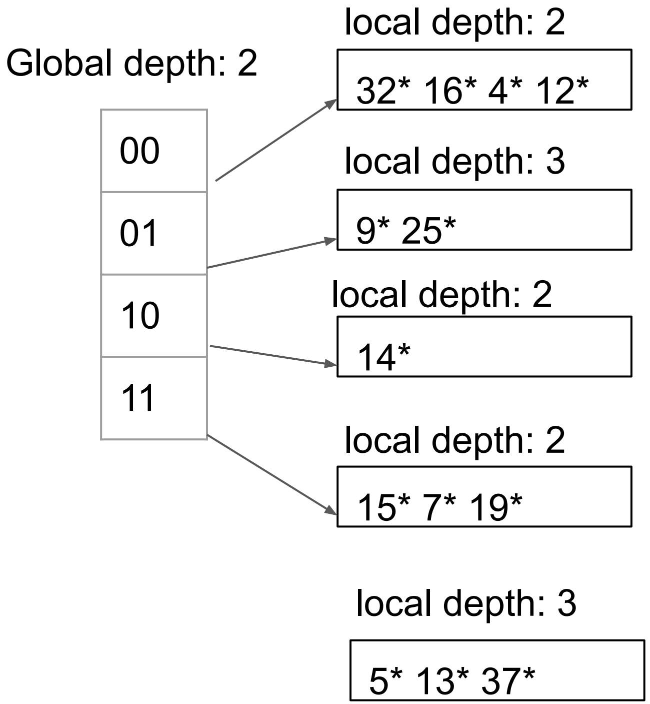
Extendible hashing



Extendible hashing

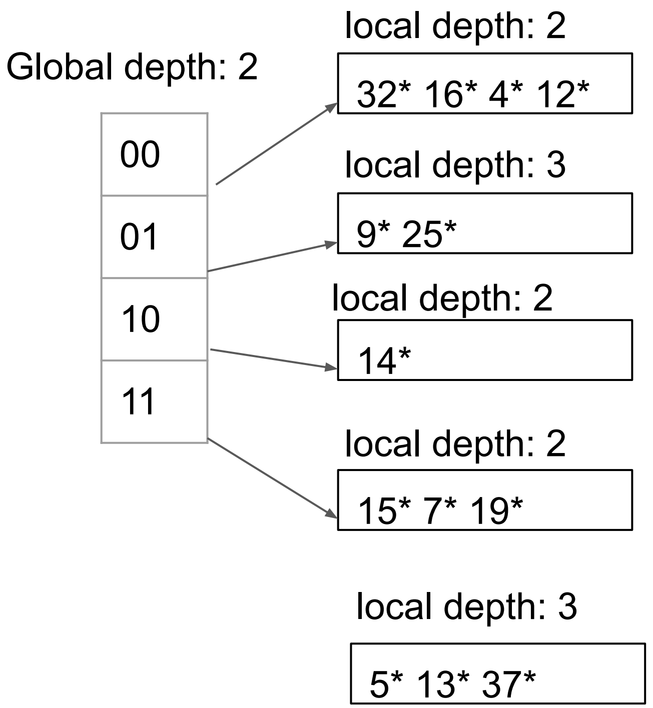
Insert 37

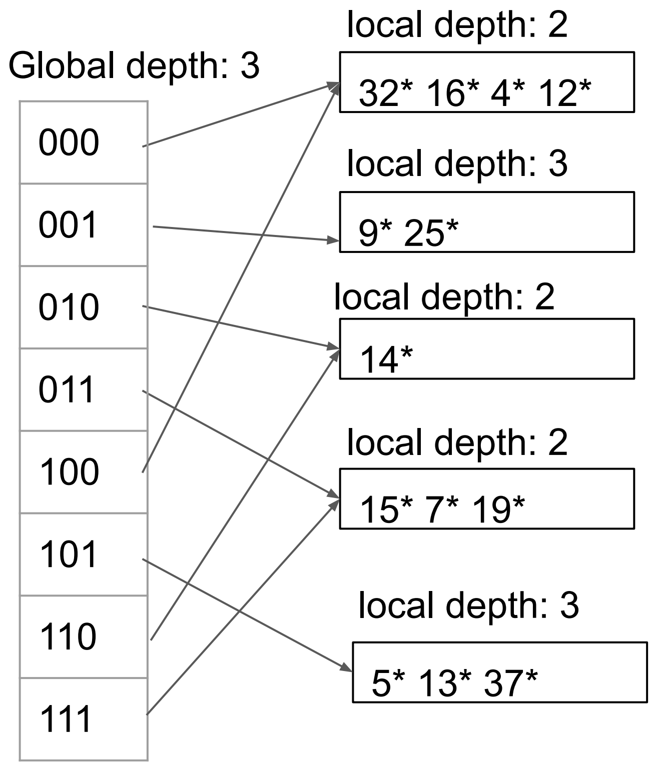




Extendible hashing

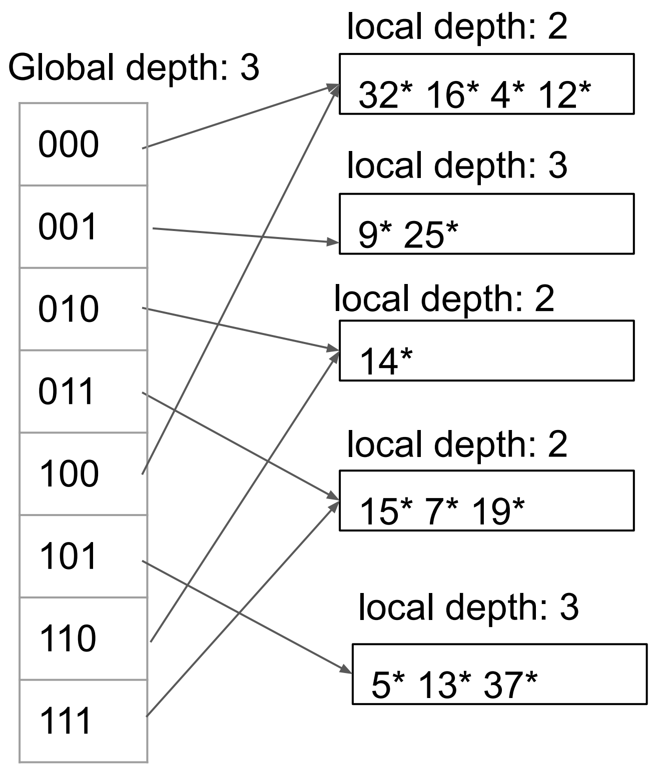
Insert 37





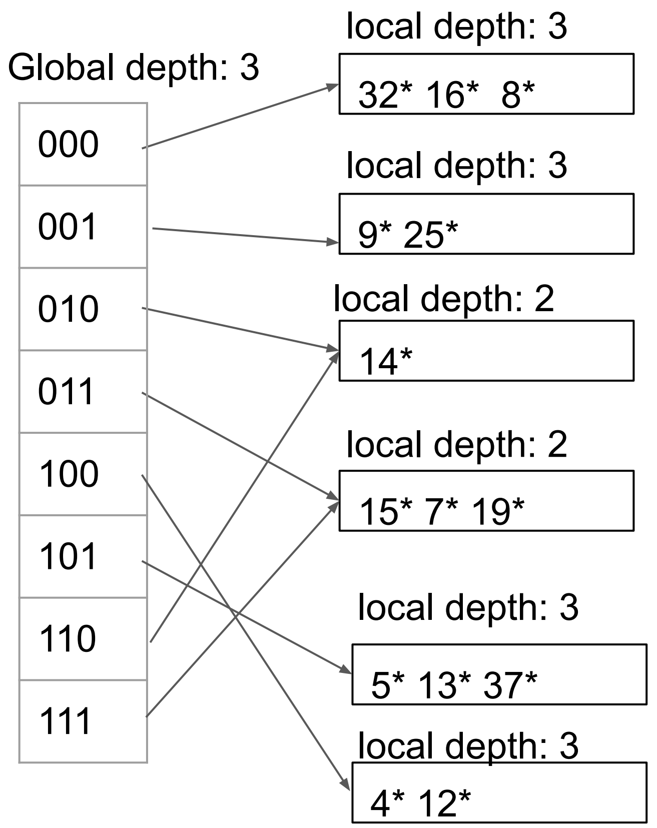
Extendible hashing

What will happen when Inserting 8*?



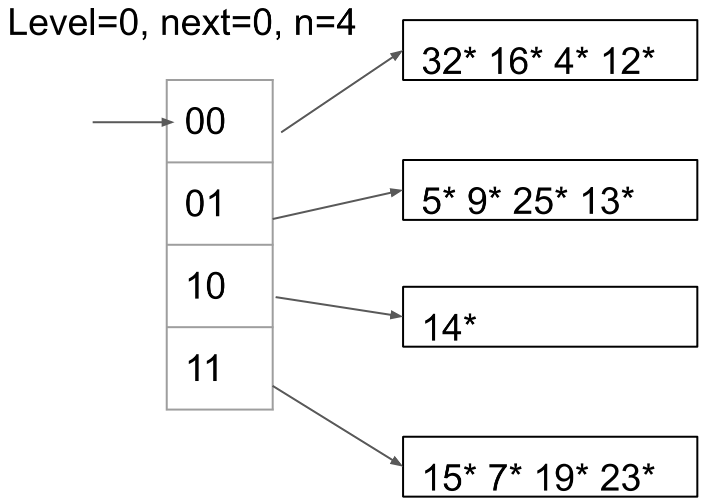
Extendible hashing

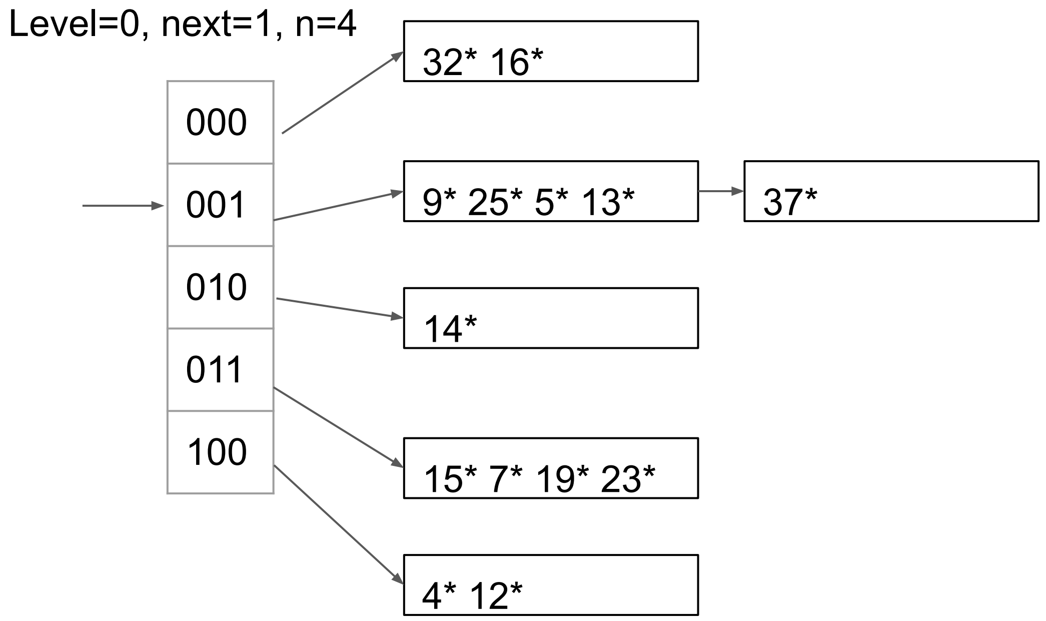
What will happen when Inserting 8*?



Linear Hashing

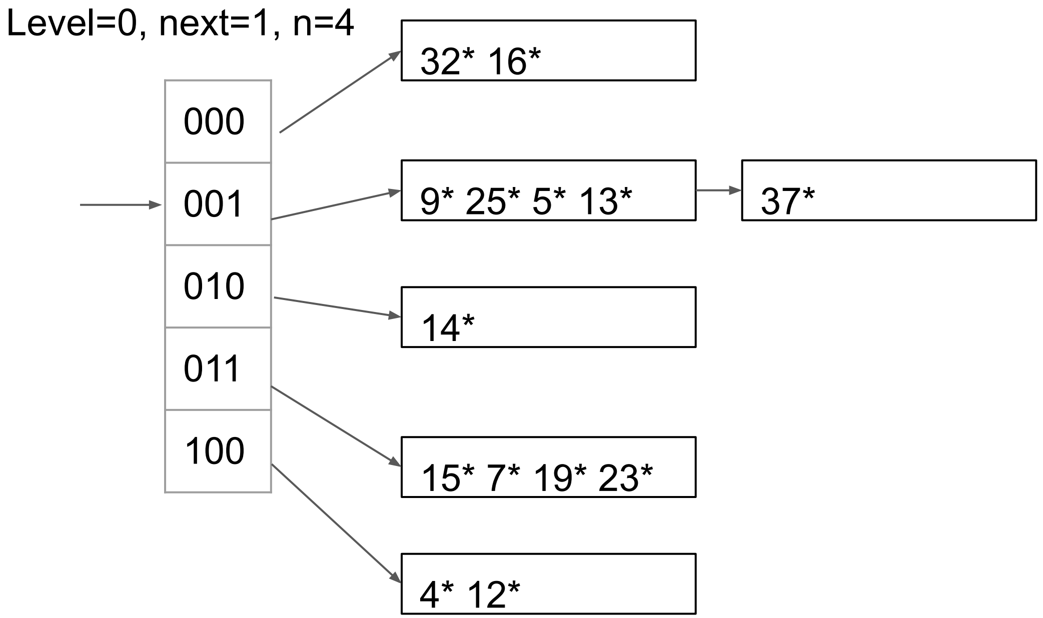
Insert 37





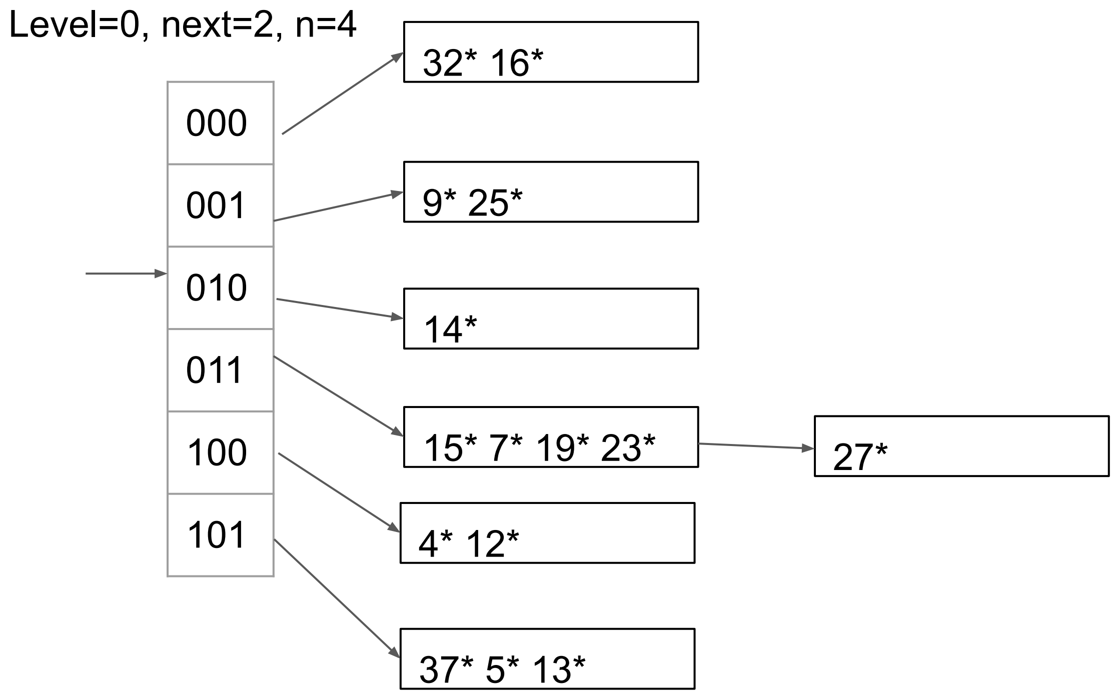
Linear Hashing

What will happen when inserting 27?



Linear Hashing

What will happen when inserting 27?



Level=0, next=2, n=4
000
001
010
011
100
101
32* 16*
9* 25*
14*
15* 7* 19* 23*
27*
4* 12*
37* 5* 13*