COMP20003

ALGORITHMS AND DATA STRUCTURES

Workshop

- Distribution Counting
- Hashing
- 2-3-4 trees

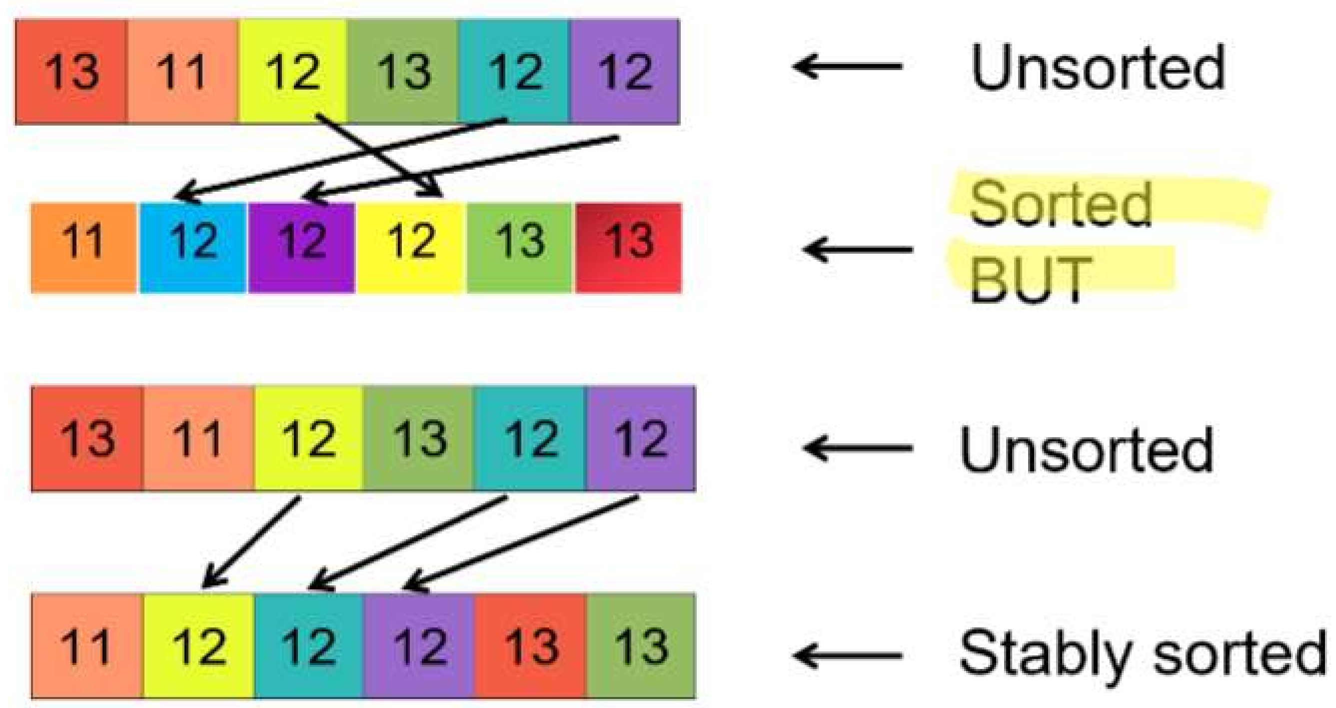
Notes

- Important to know all implementations covered in lectures! Includes BST and AVL (be familiar with code fo AVLs too)
- Main ones to be familiar with – 2-3-4 and KD (2d) trees, expect detailed questions
- Other trees like splay / red-black – high level, compare between this and other trees etc.

K-D 2D trees

- BSTs, where we consider an alternative dimension at a time

Stable Sort


13 11 12 13 12 12
Unsorted
11 12 12 12 13 13
Sorted
BUT
13 11 12 13 12 12
Unsorted
11 12 12 12 13 13
Stably sorted

Distribution Counting

- Sorting with complexity ***O(n)***
- **BUT**
  - Comes at a cost
  - Need to have key values in arrays to be within a certain **range**
- **HOW**
  - Count records for each key, and redistribute elements
- **RESULT**
  - Stable sorted array

Distribution Counting

- [2, 2, 2, 0, 0, 1, 4, 3, 4, 2, 5, 1]
- Array – count records for each key [index is key, value is counts]
- Array2 – cumulative counts for each key [does NOT include the current bin]
- Array3 – aux array, redistributed elements
- Cumulative counts are the INDEX at which we put the items in the new array
- Traverse original list to build aux array
- Put in item, cuml_count - = 1 for that record [the record is the indexes of the cuml. Array]

Distribution Counting

- Analysis
  - O(n + range)
    - getting counts (n) + cuml array (range) + traverse original array (n)
  - O(n + range) - space
    - Original array (n) + 2 range arrays (2xrange) + Aux array (n)

- Faster than comparison-based sorting! Omega(nlogn)
- But space costly

Let's do a search on an array

Unsorted

[1, 12, 4, 7, 0, 8, 19]

If sorted:

[0, 1, 4, 7, 8, 12, 19]

Can we do better?

Hashing

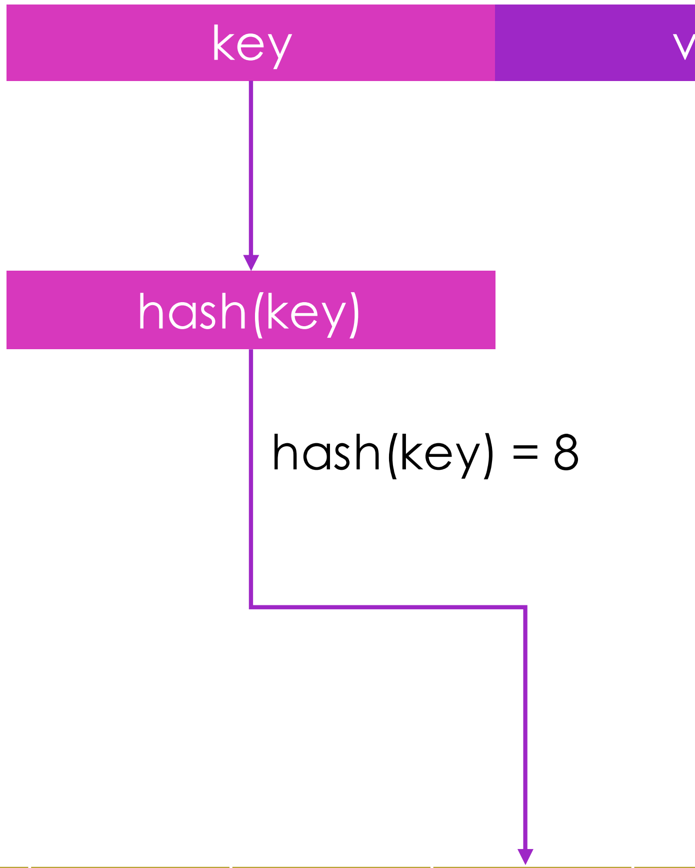
- Best known implementation of dictionary
- How Python dicts are implemented
- If need lots of searches on DS, use this
- $O(1)$ average lookup
- Catastrophic worst case

Hashing

- Best known implementation of dictionary
- How Python dicts are implemented
- If need lots of searches on DS, use this
- $O(1)$ average lookup
- Catastrophic worst case

Hash Tables

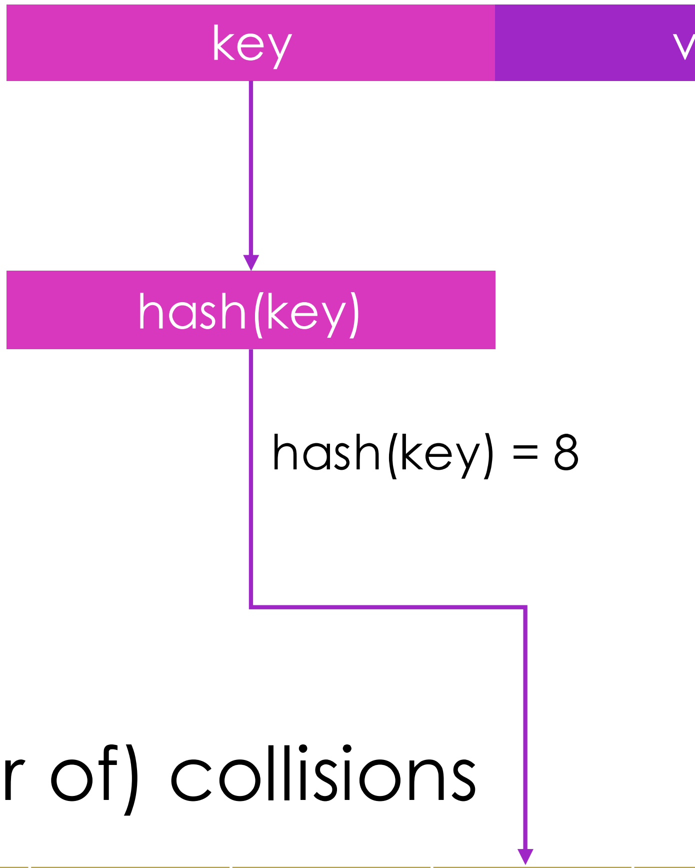
- Dictionary
- Insert
  - O(?) Average
- Search
  - O(?) Average



Index	0	1	2	3	4	5	6	7	8	9
key	0			3	4				key	
value	Data1			Data2	Data3				value	

Hash Tables

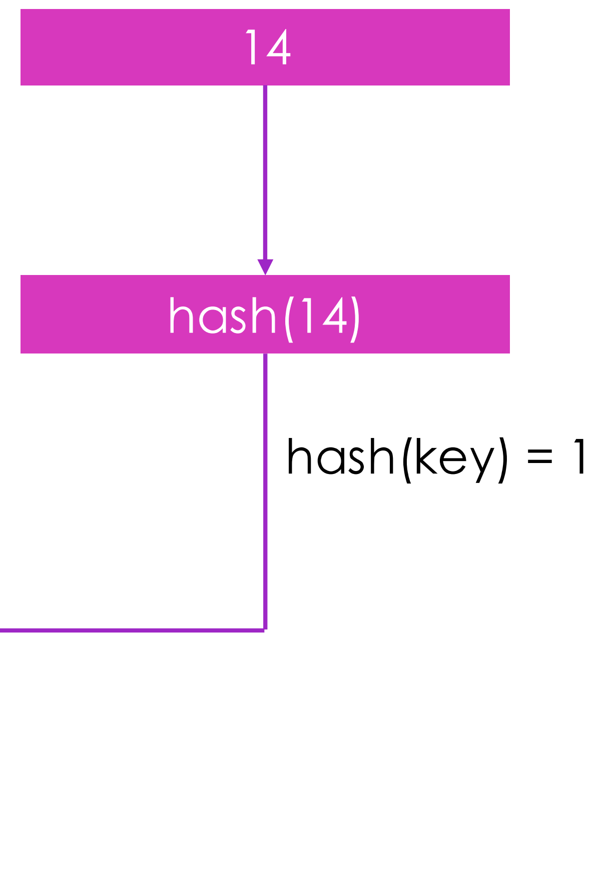
- Dictionary
- Insert
  - O(1) Average
- Search
  - O(1) Average
- O(1) when no (or a constant number of) collisions



Index	0	1	2	3	4	5	6	7	8	9
key	0			3	4				key	
value	Data1			Data2	Data3				value	

Hash Tables

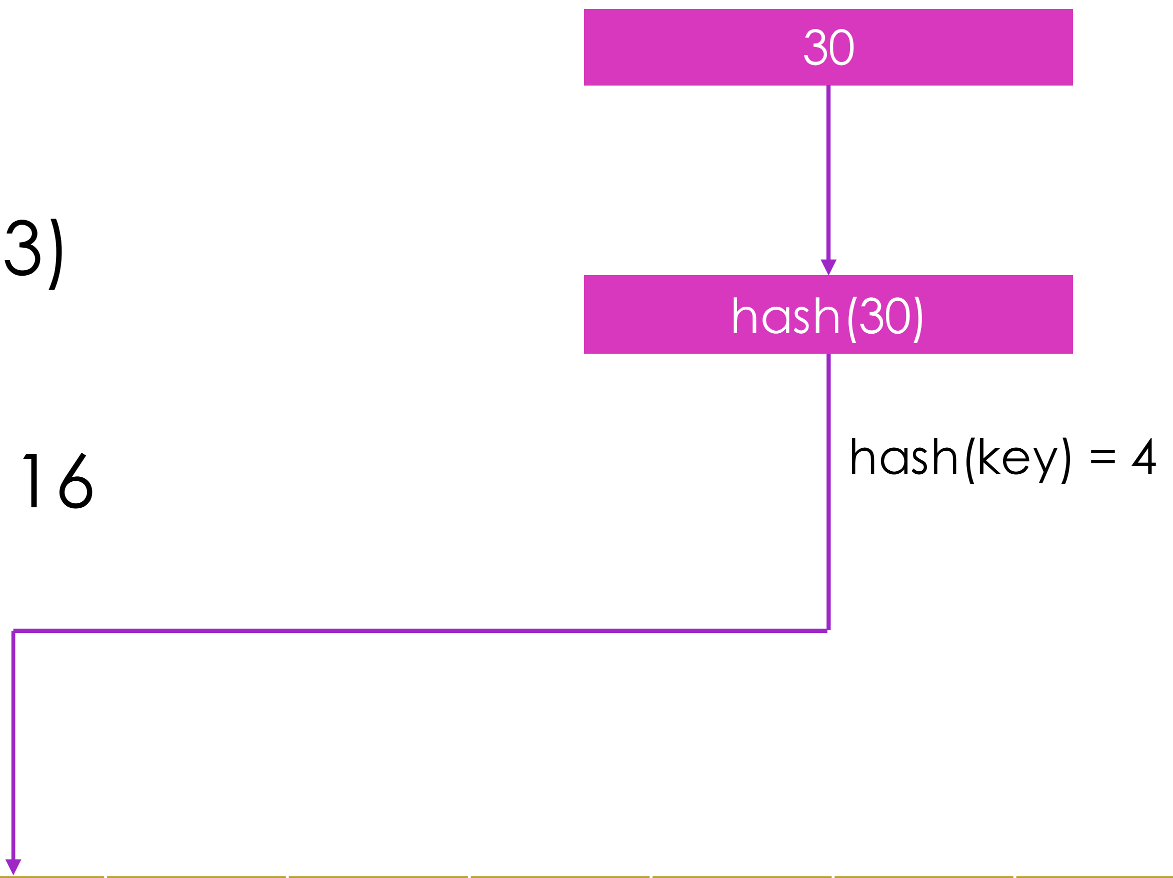
- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14										

Hash Tables

- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16

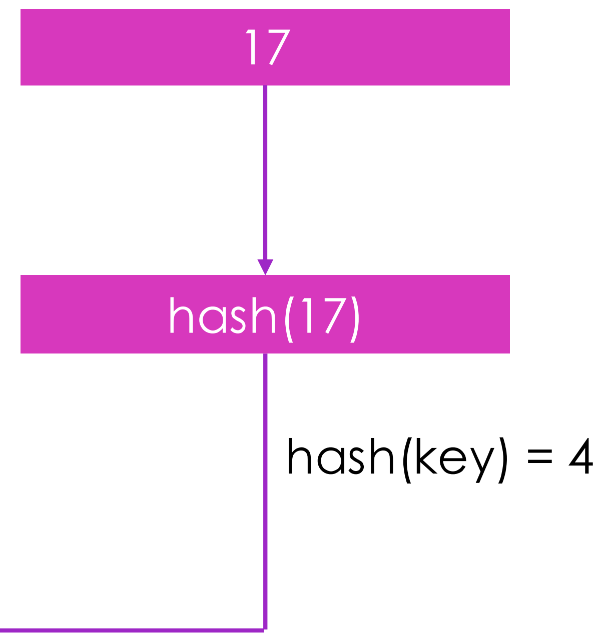


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14			30							

Hash Tables

- hash(key) = (key % 13)

- 14, 30, 17, 55, 31, 29, 16



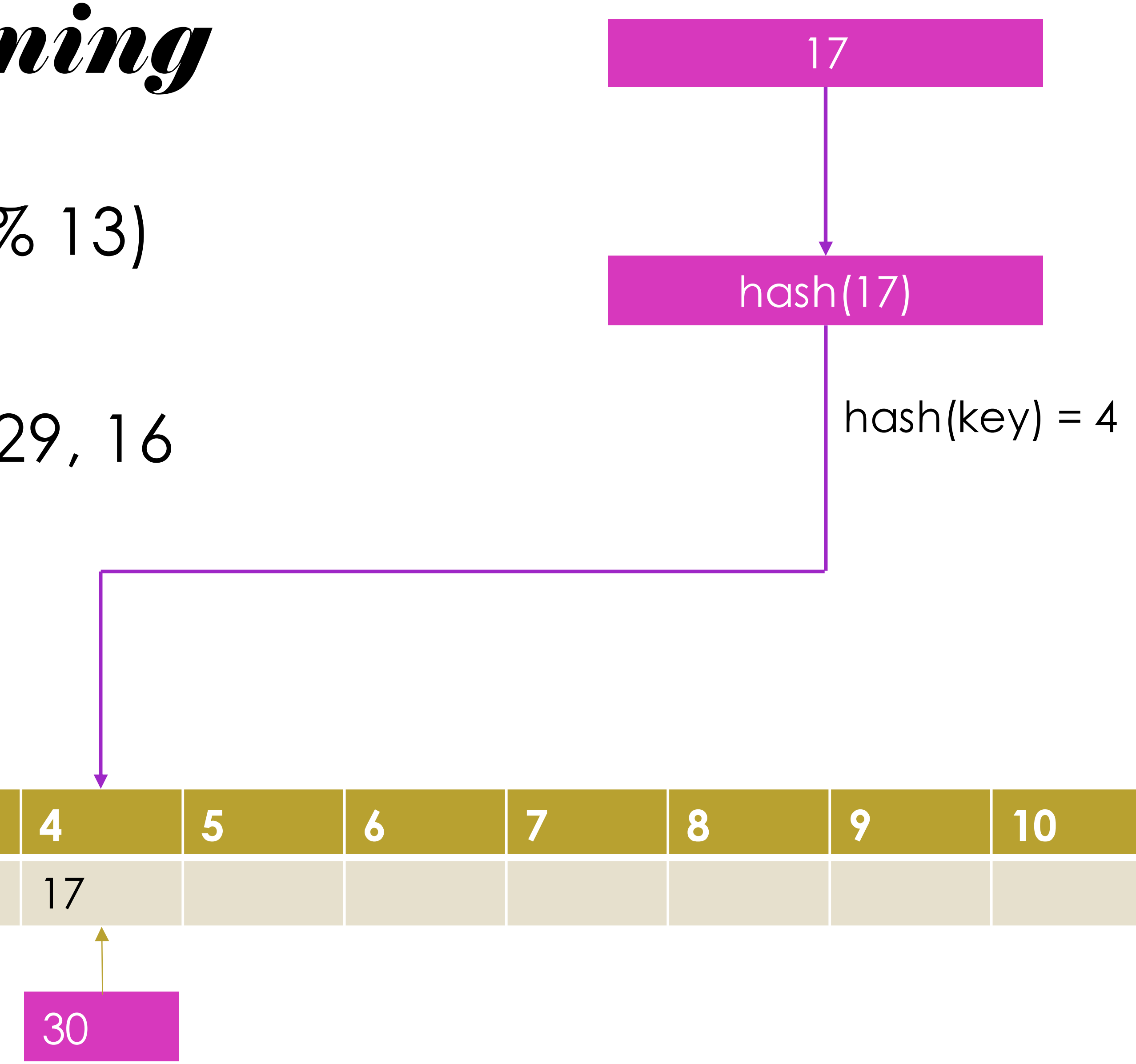
Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14			30							

Hash Table Collisions

- Collision Resolution
  - Linear Chaining
  - Linear Probing
  - Double Hashing

Hash Tables
Linear Chaining

- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16

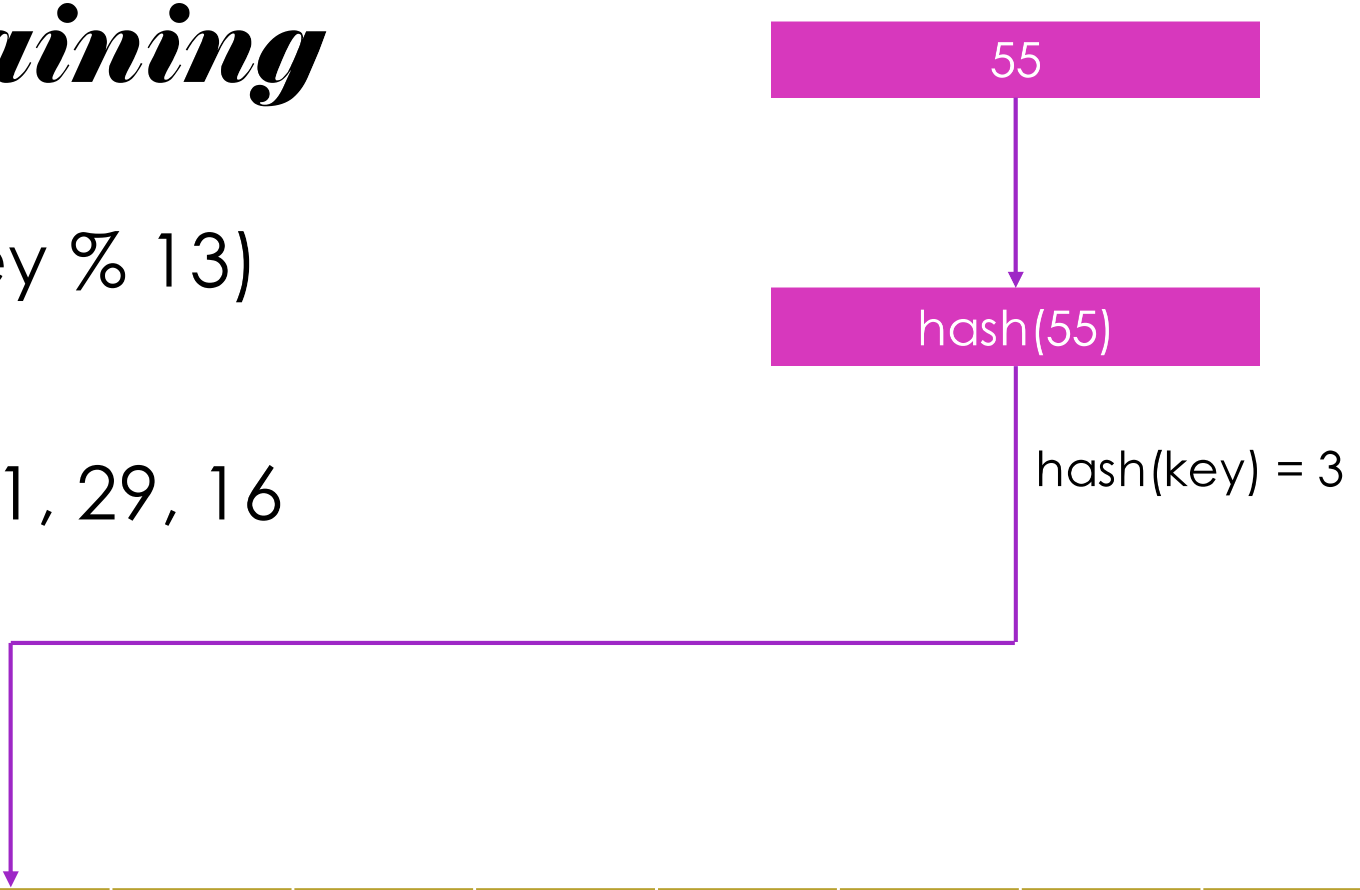


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14			17							

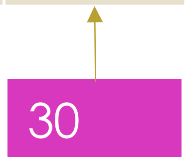
Hash Tables
Linear Chaining

- hash(key) = (key % 13)

- 14, 30, 17, 55, 31, 29, 16



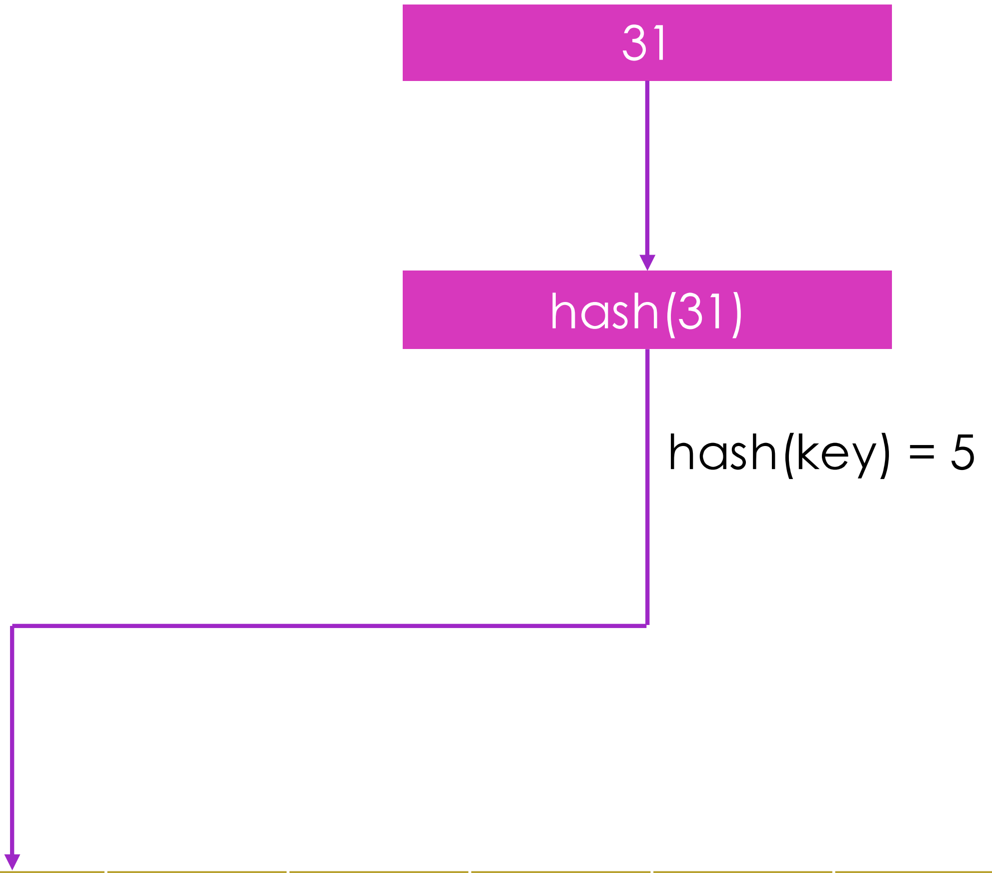
Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	17							



Hash Tables
Linear Chaining

- hash(key) = (key % 13)

- 14, 30, 17, 55, 31, 29, 16

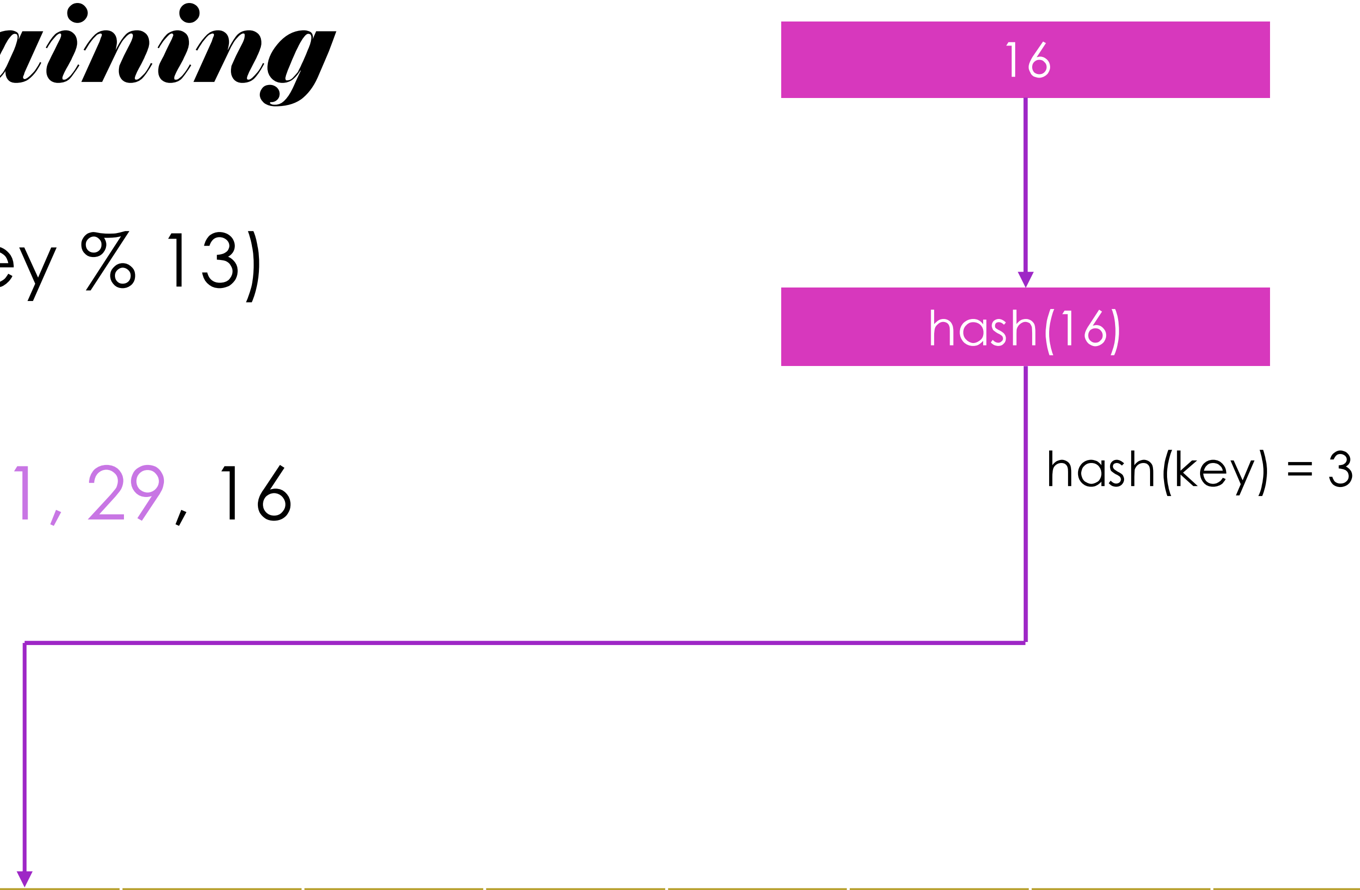


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	17	31						

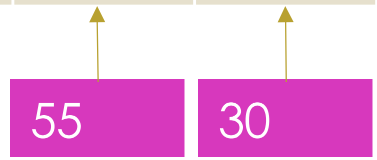
Hash Tables
Linear Chaining

- hash(key) = (key % 13)

- 14, 30, 17, 55, 31, 29, 16



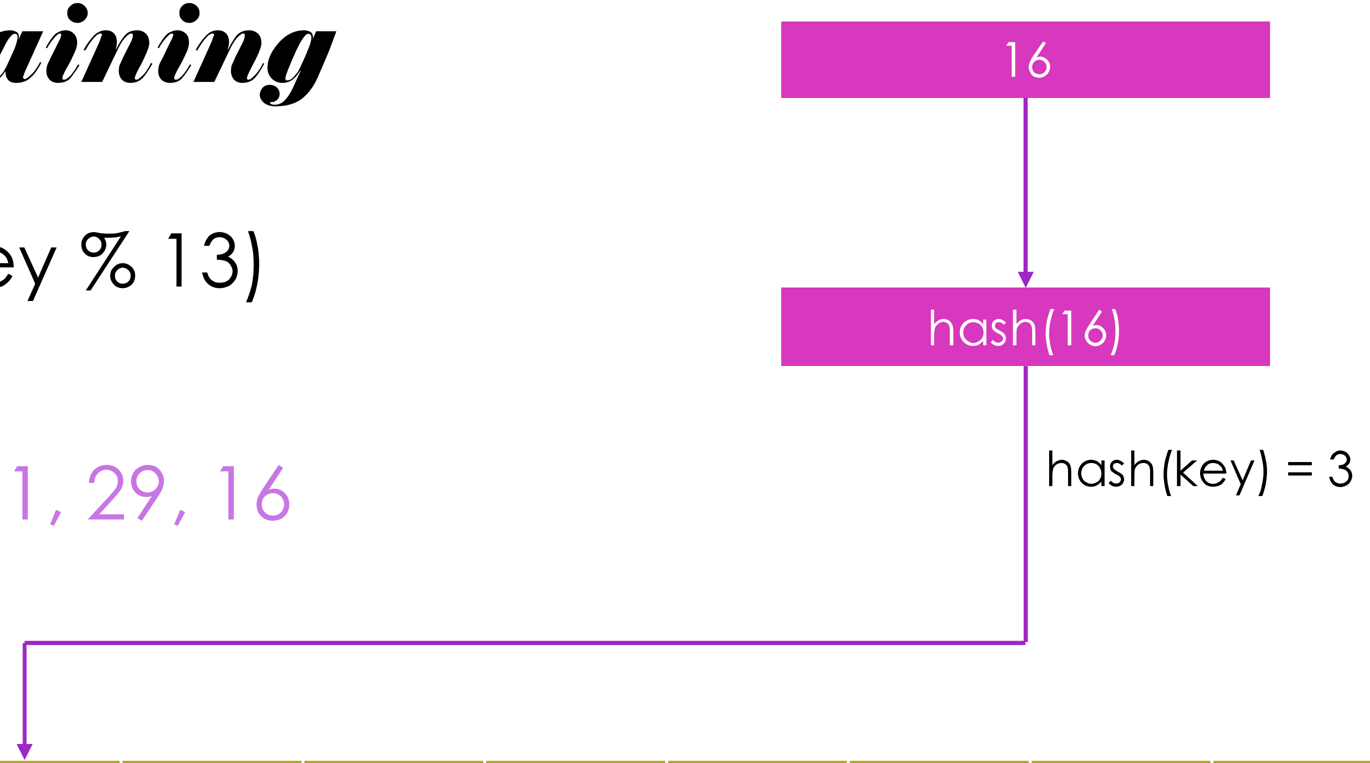
Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		29	17	31						



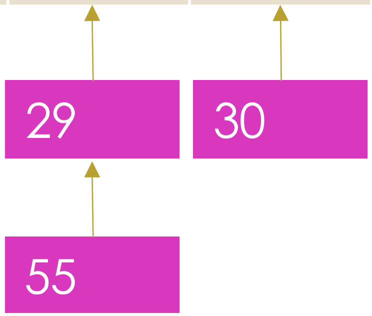
Hash Tables
Linear Chaining

- hash(key) = (key % 13)

- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		16	17	31						



Hash Tables
Linear Probing

- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16

17

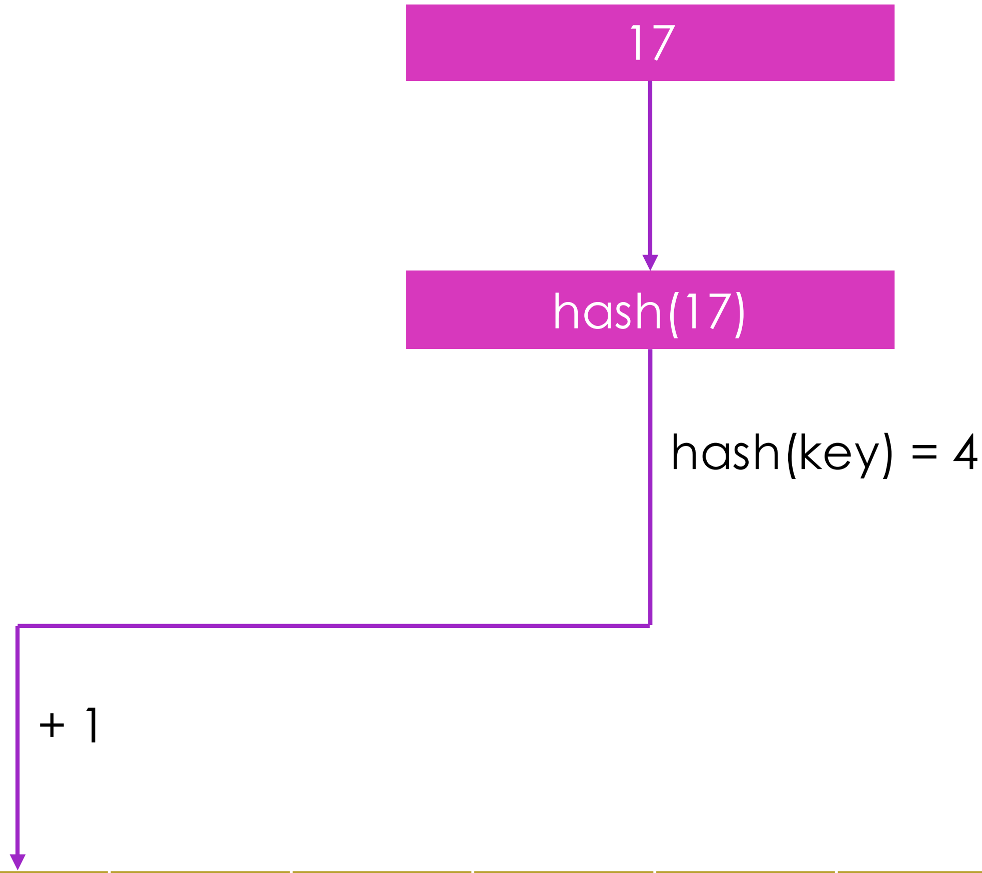
hash(17)

hash(key) = 4

Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14			30							

Hash Tables
Linear Probing

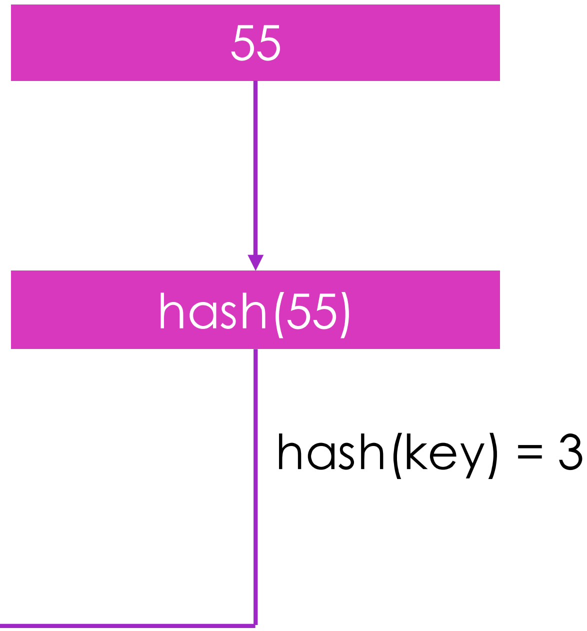
- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14			30	17						

Hash Tables
Linear Probing

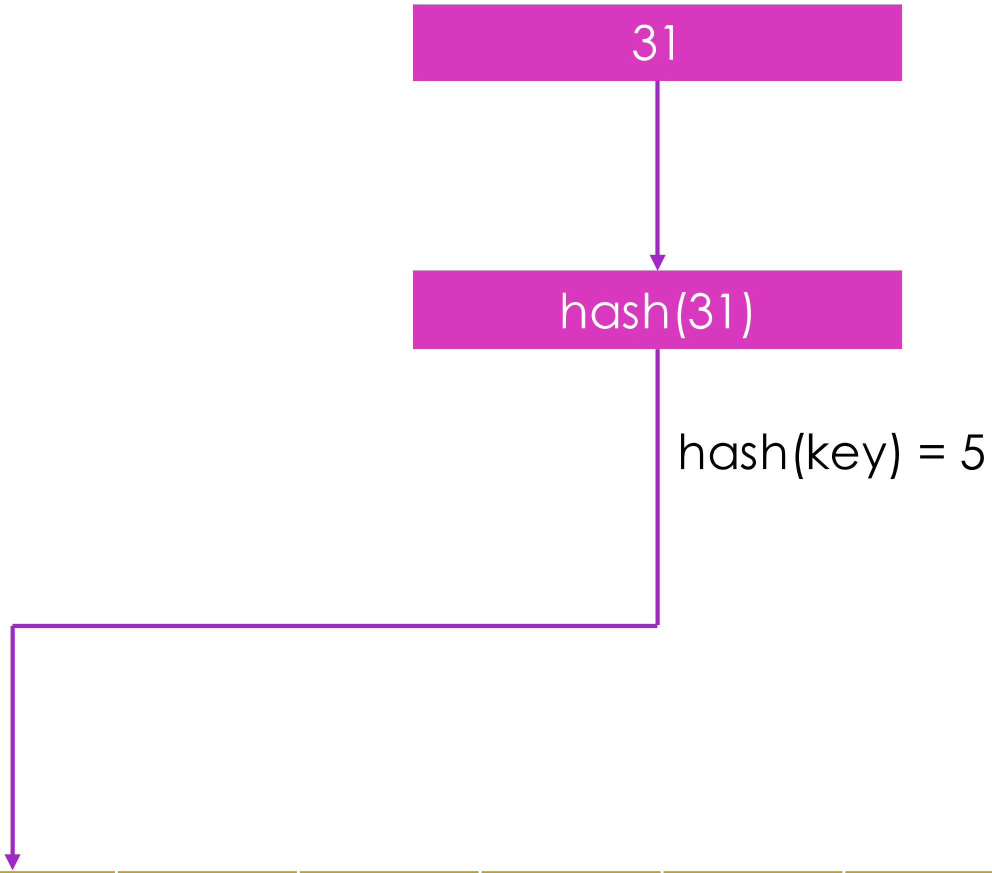
- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17						

Hash Tables
Linear Probing

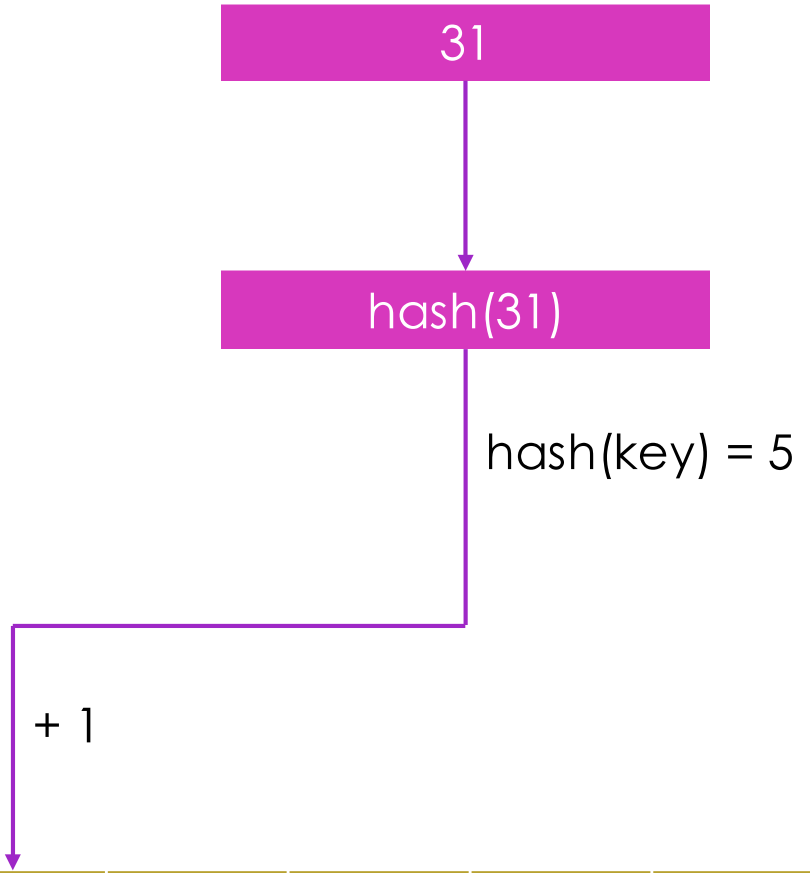
- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17						

Hash Tables
Linear Probing

- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16

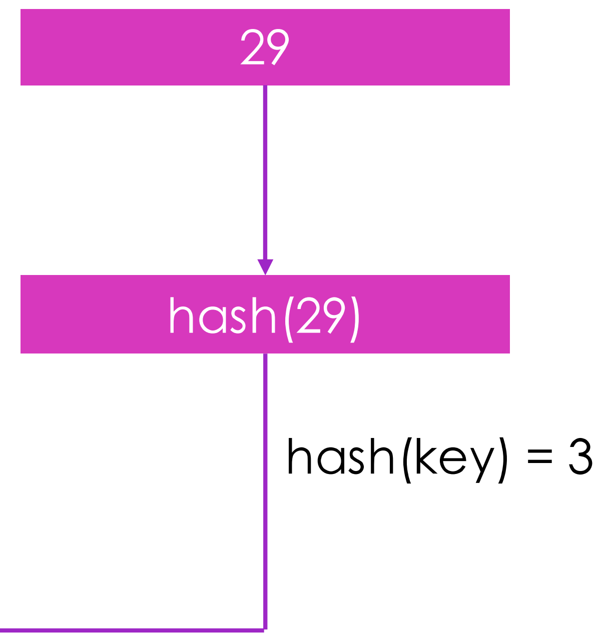


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31					

Hash Tables
Linear Probing

- hash(key) = (key % 13)

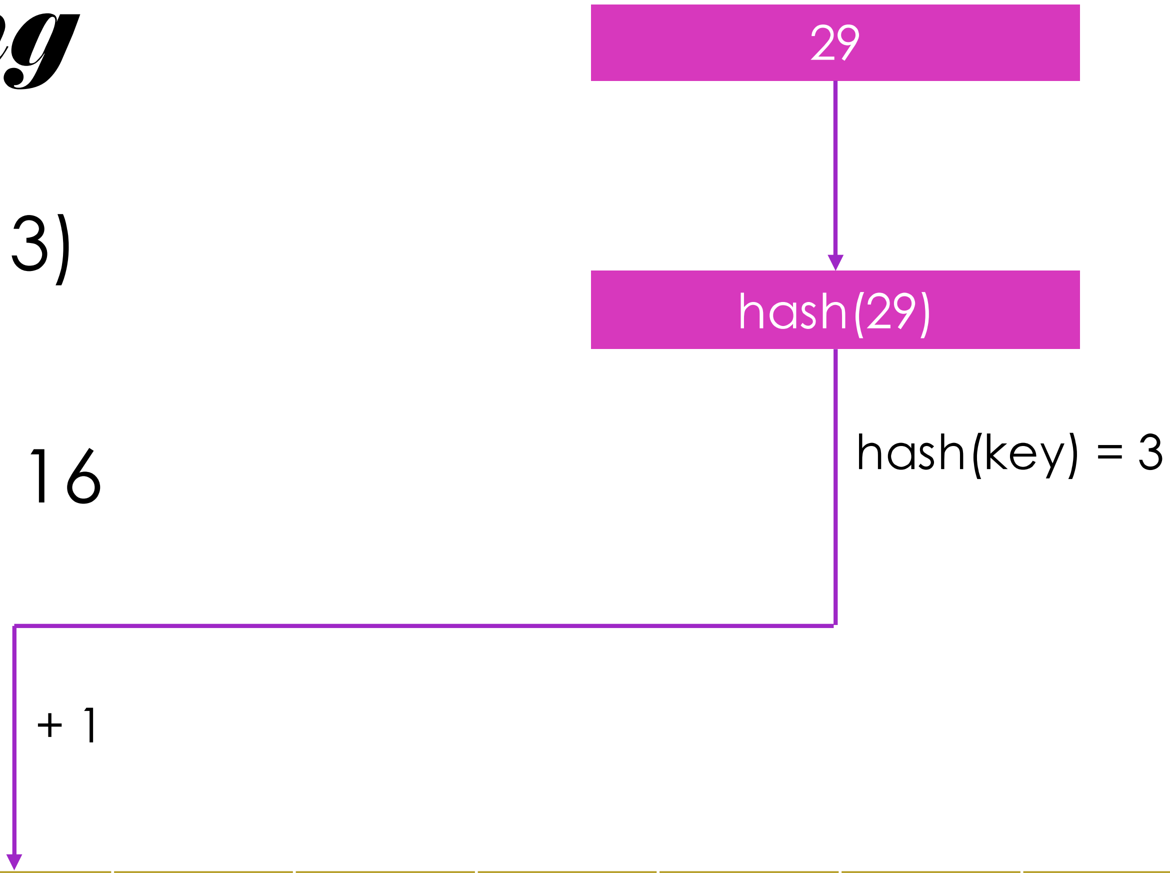
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31					

Hash Tables
Linear Probing

- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16

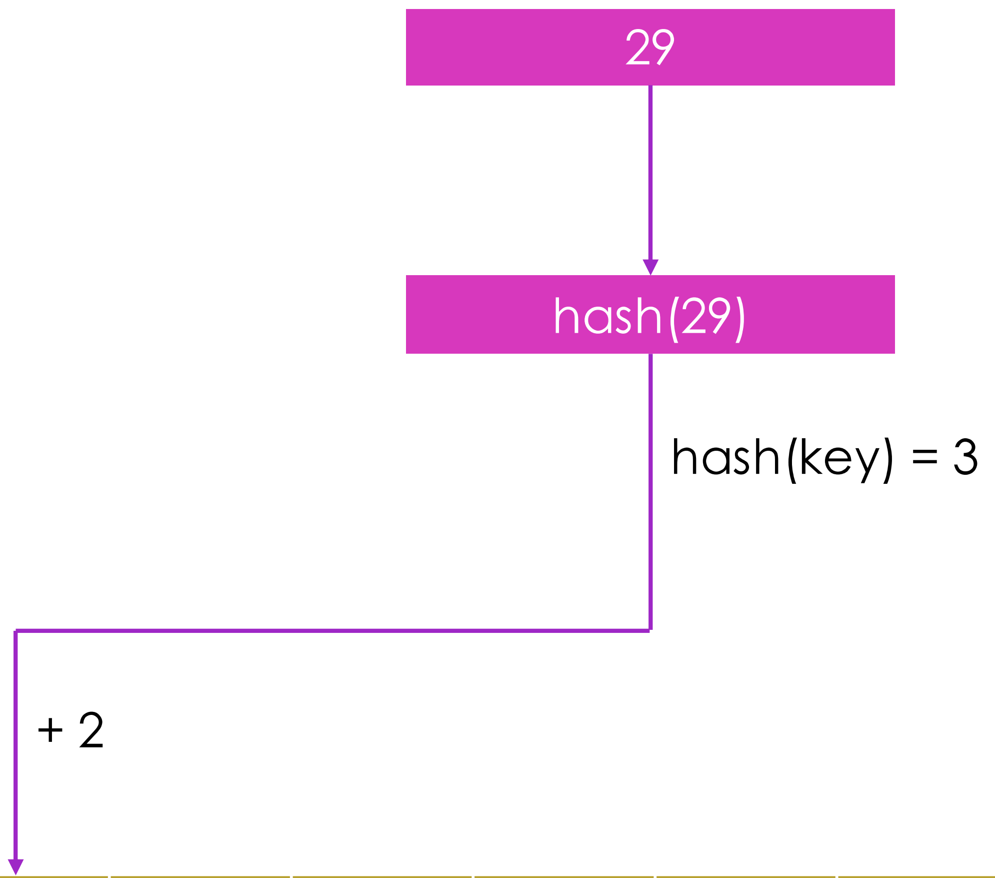


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31					

Hash Tables
Linear Probing

- hash(key) = (key % 13)

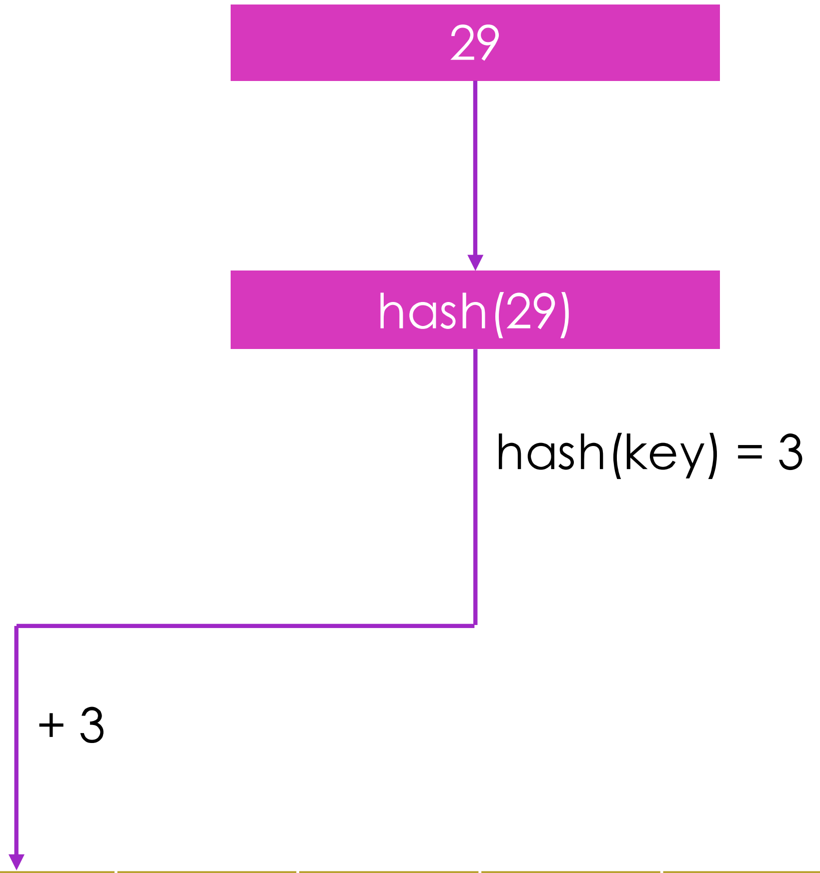
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31					

Hash Tables
Linear Probing

- hash(key) = (key % 13)
- 14, 30, 17, 55, 31, 29, 16

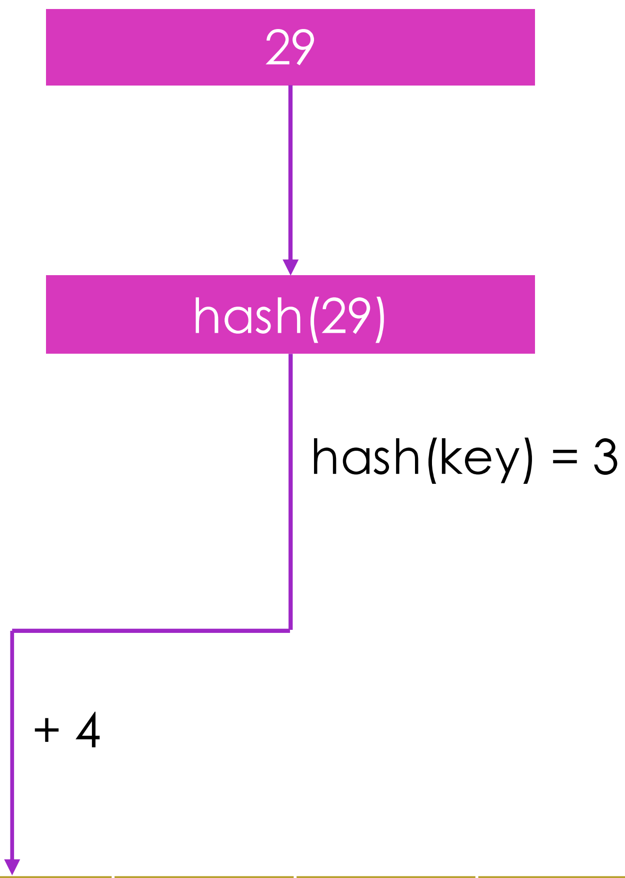


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31					

Hash Tables
Linear Probing

- hash(key) = (key % 13)

- 14, 30, 17, 55, 31, 29, 16

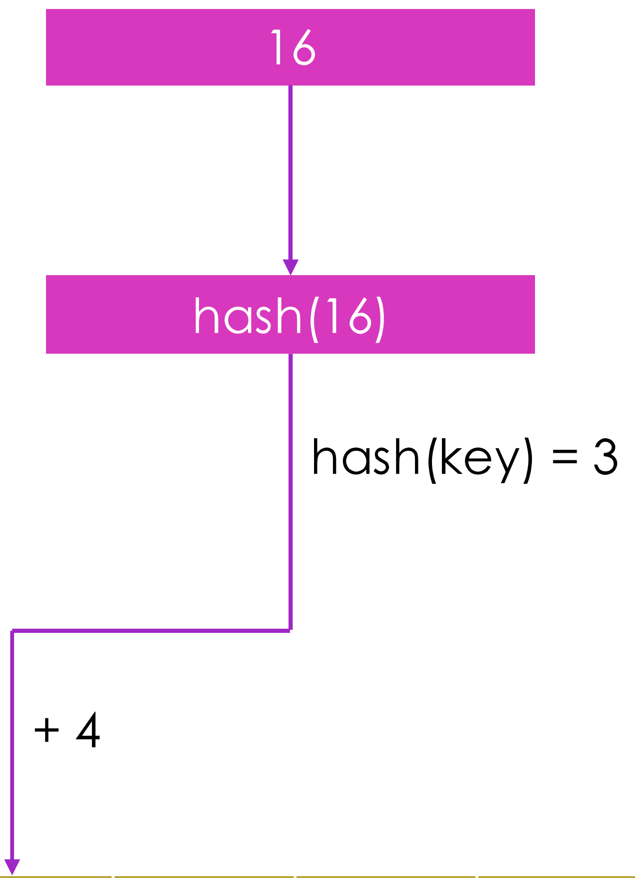


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31	29				

Hash Tables
Linear Probing

- hash(key) = (key % 13)

- 14, 30, 17, 55, 31, 29, 16

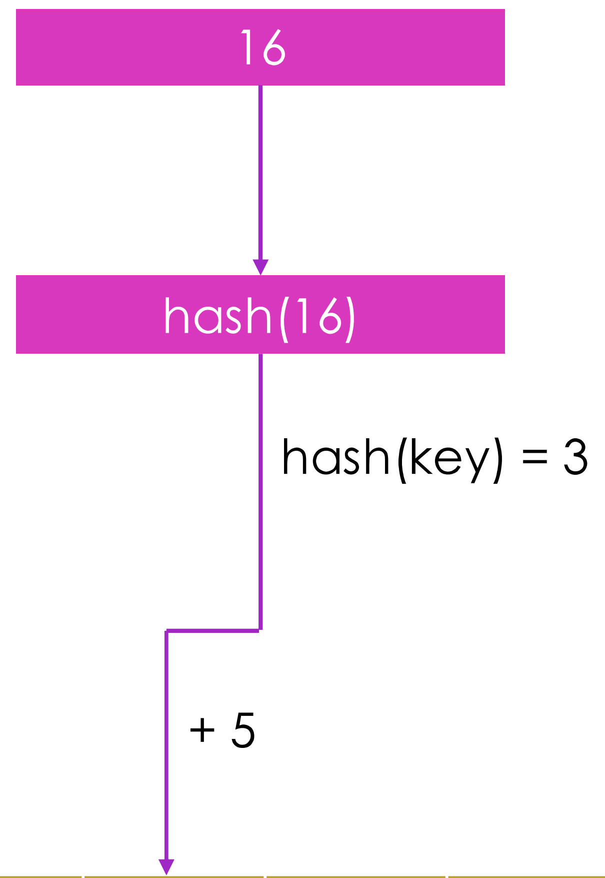


Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31	29				

Hash Tables
Linear Probing

- hash(key) = (key % 13)

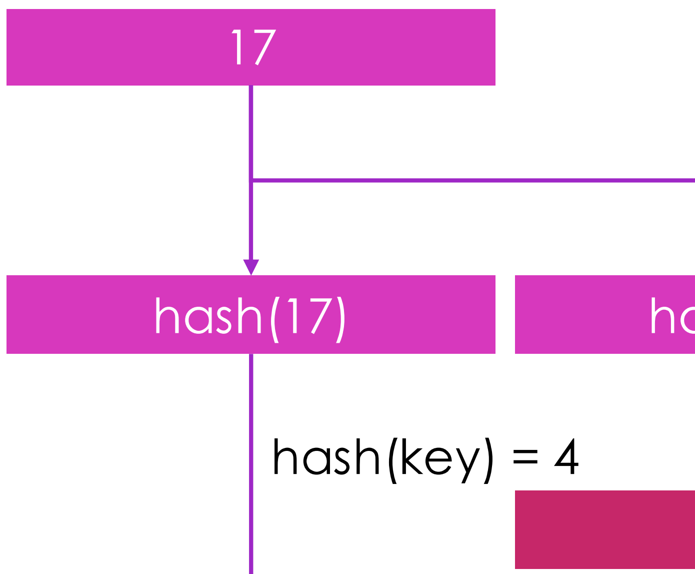
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	17	31	29	16			

Hash Tables
Double Hashing

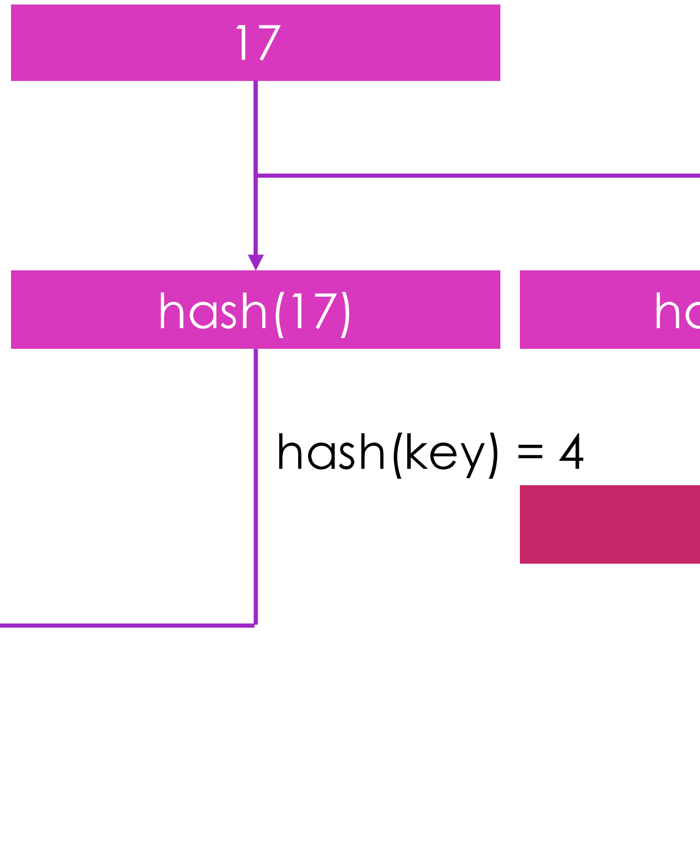
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14			30							

Hash Tables
Double Hashing

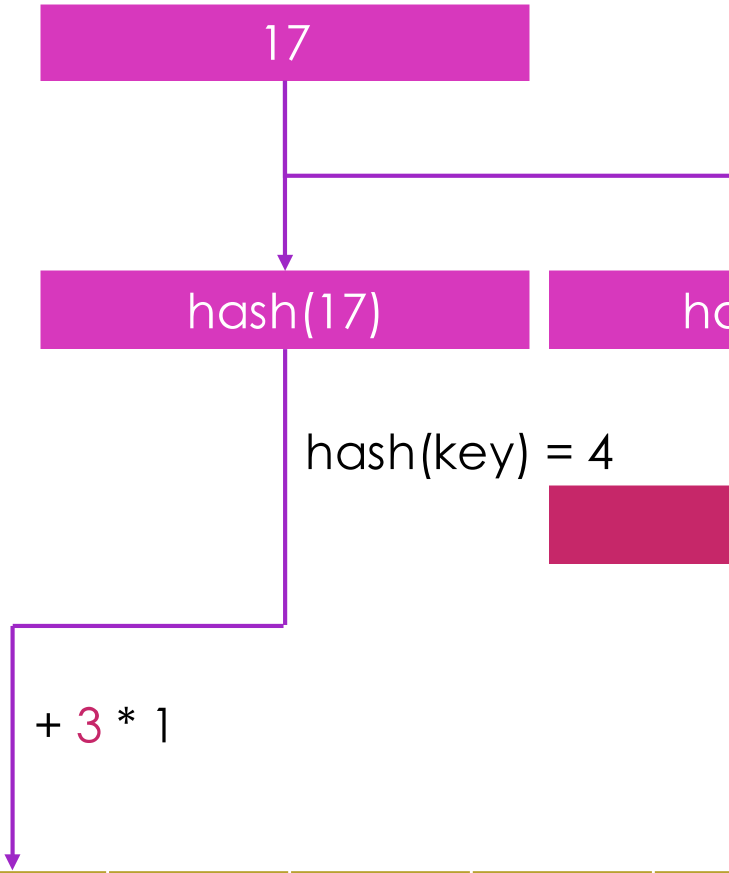
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16
- Which index will we try next?



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		?	30			?				

Hash Tables
Double Hashing

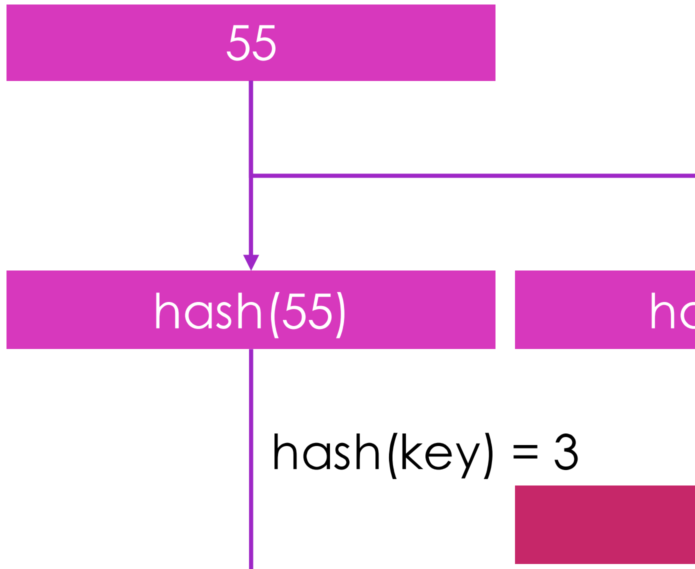
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14			30			17				

Hash Tables
Double Hashing

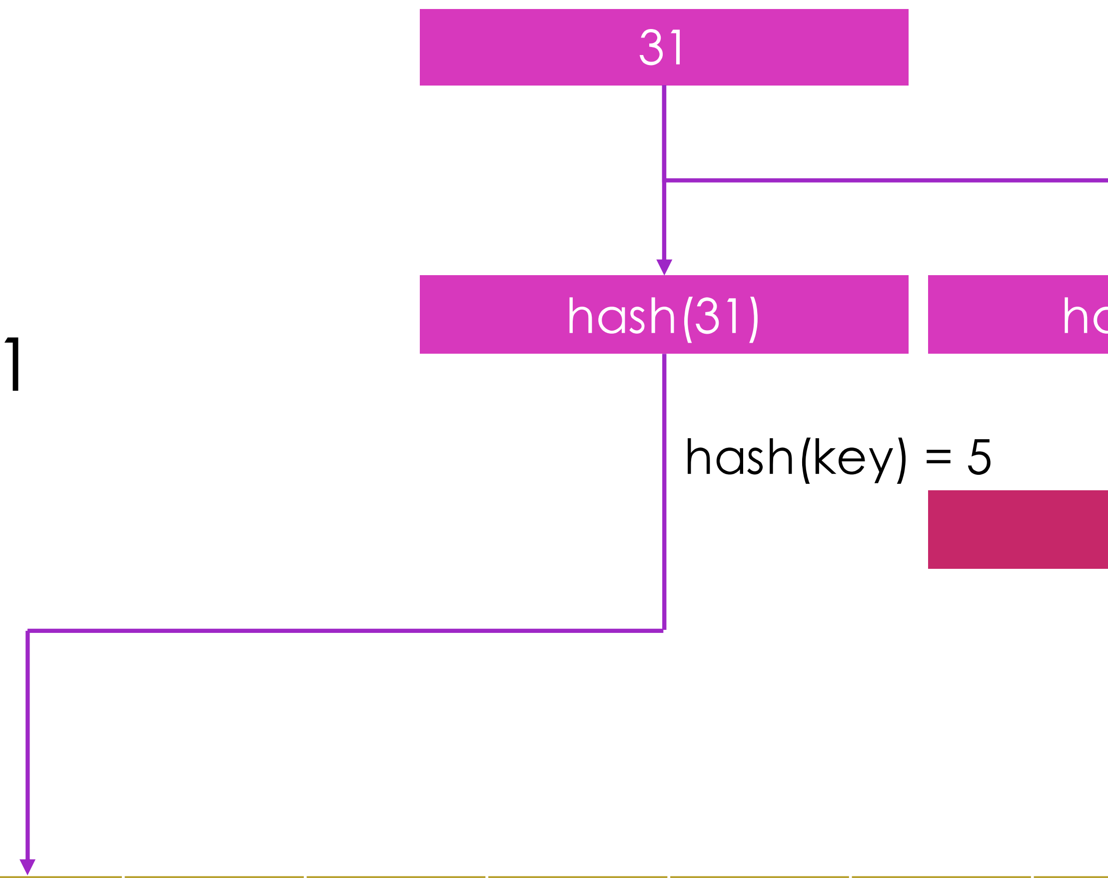
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30			17				

Hash Tables
Double Hashing

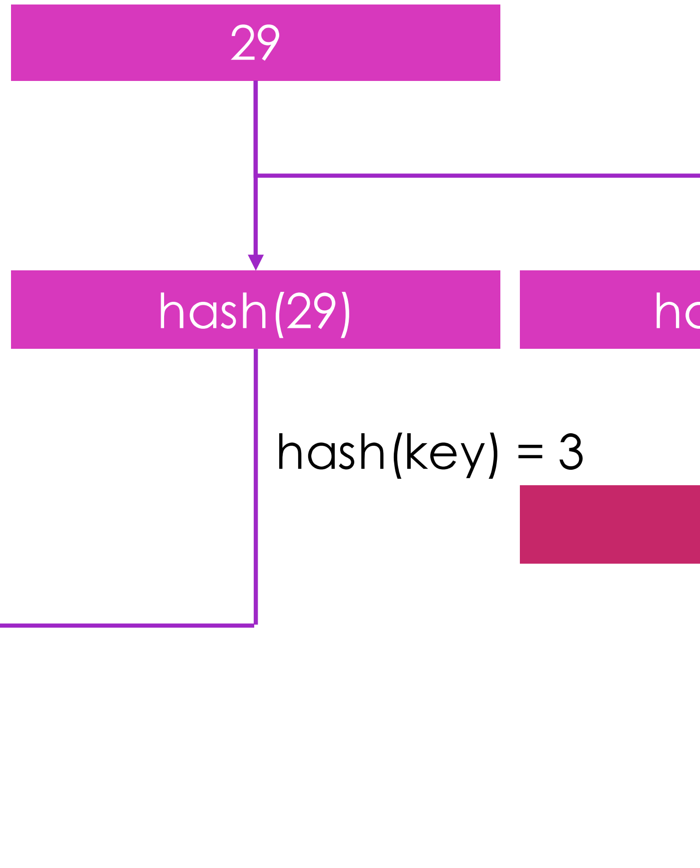
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17				

Hash Tables
Double Hashing

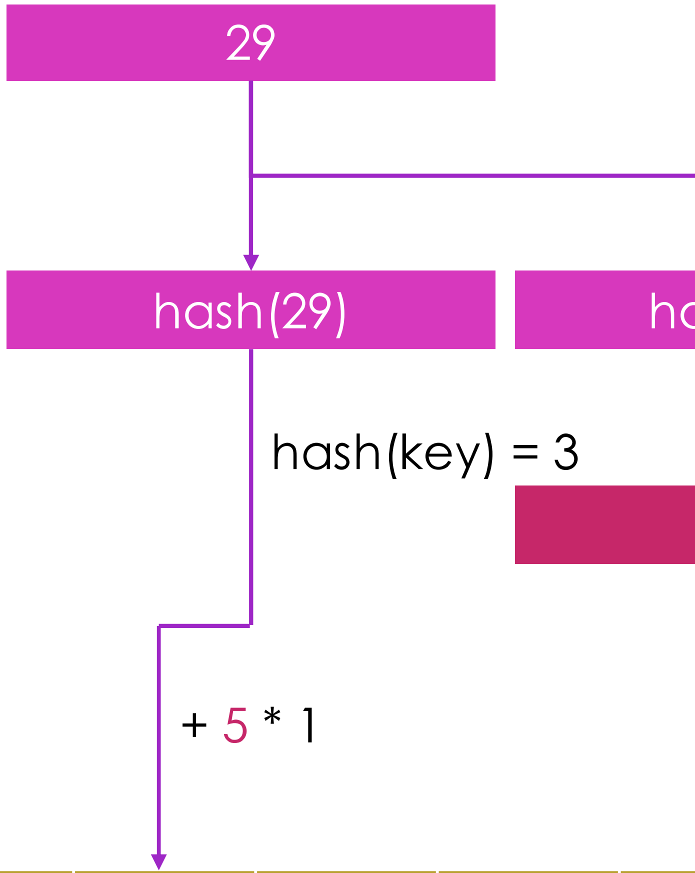
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17				

Hash Tables
Double Hashing

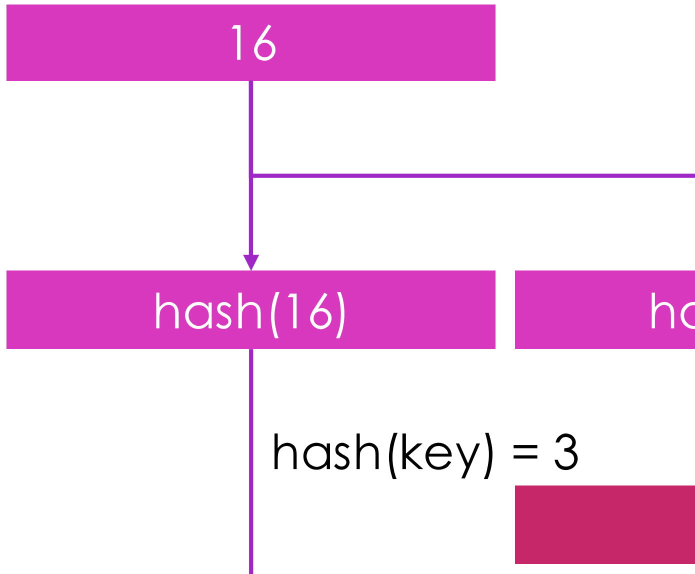
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17	29			

Hash Tables
Double Hashing

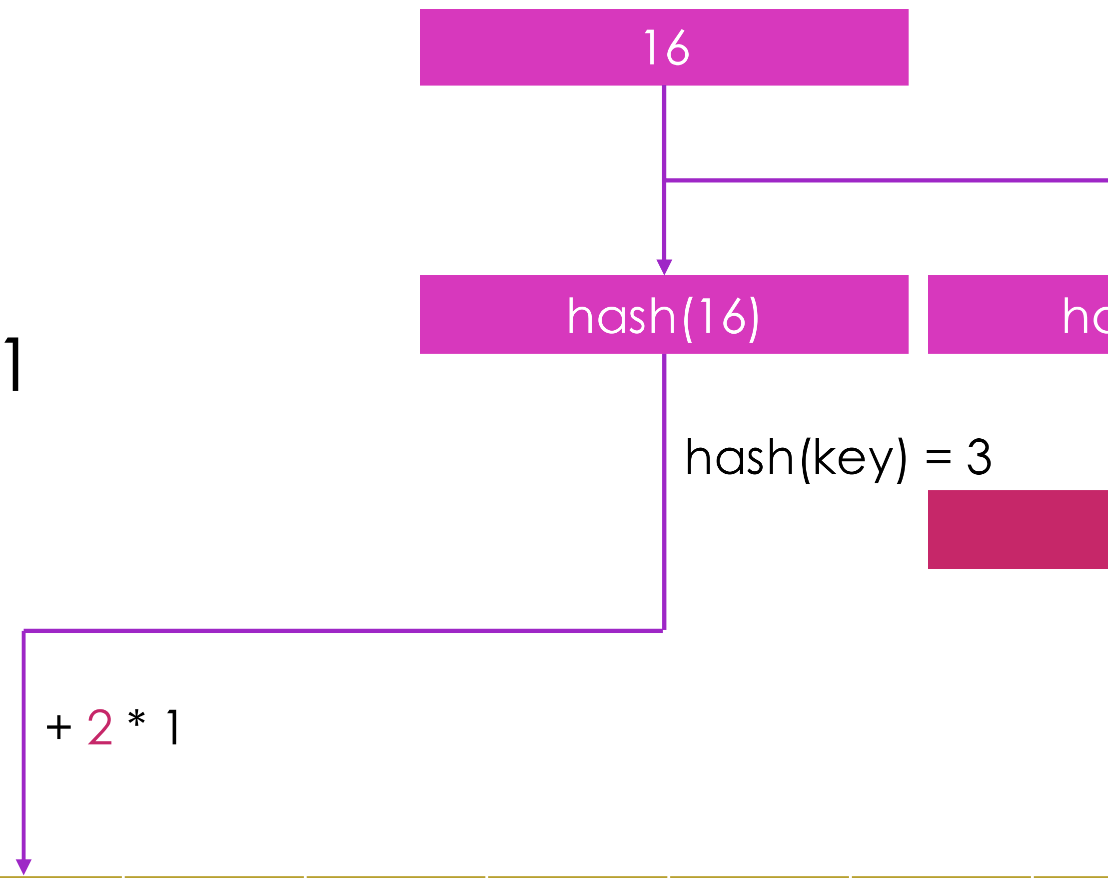
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17	29			

Hash Tables
Double Hashing

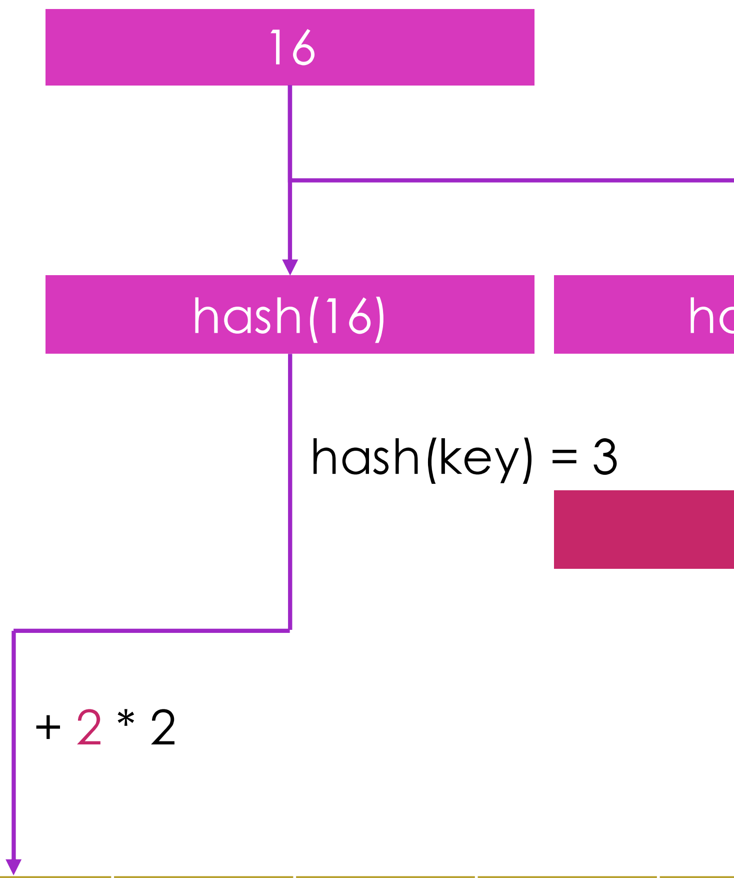
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17	29			

Hash Tables
Double Hashing

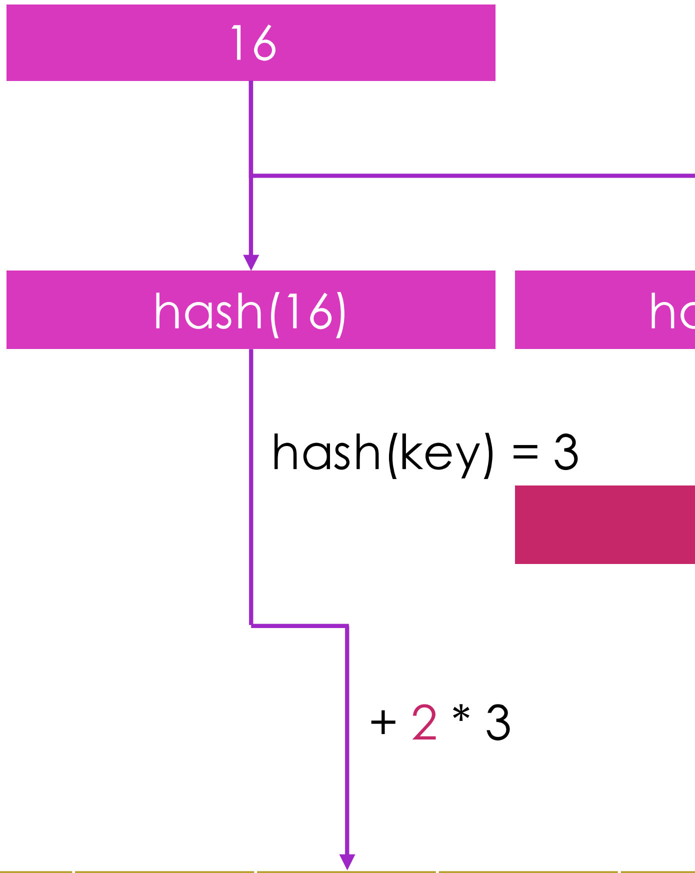
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17	29			

Hash Tables
Double Hashing

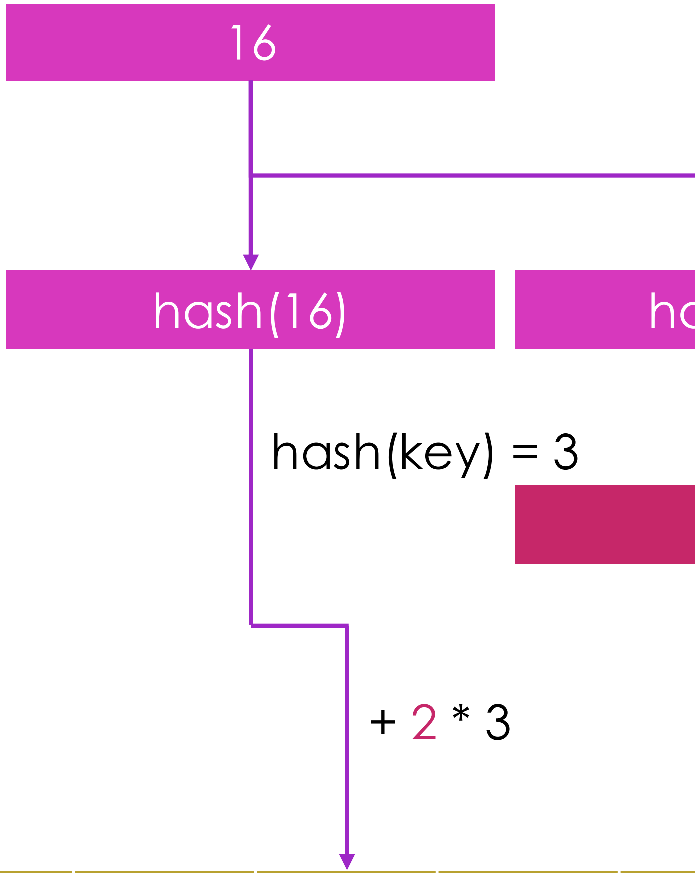
- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16



Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17	29	16		

Hash Tables
Double Hashing

- hash(key) = (key % 13)
- hash2(key) = (key % 5) + 1
- 14, 30, 17, 55, 31, 29, 16
- Co-prime modulus
- +1 so we always advance



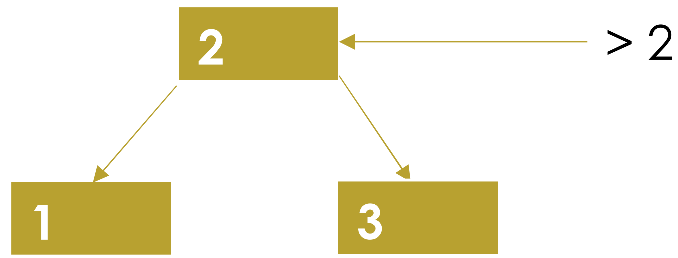
Index	0	1	2	3	4	5	6	7	8	9	10	11
Key		14		55	30	31		17	29	16		

2-3-4 Trees

- Binary → 2 pointers for each node
- 2-3-4 Trees → 2, 3, or 4 pointers for each node
- Always insert at bottom
- Break up nodes on **the way down**

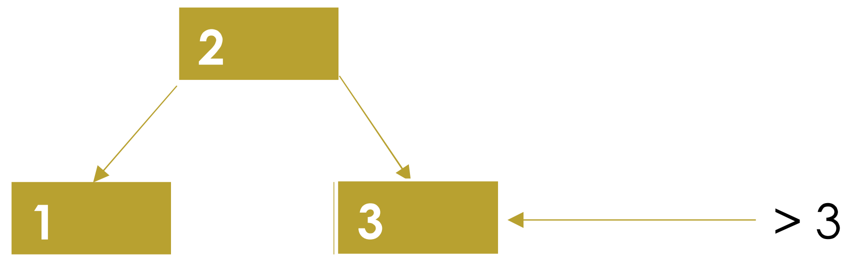
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 4



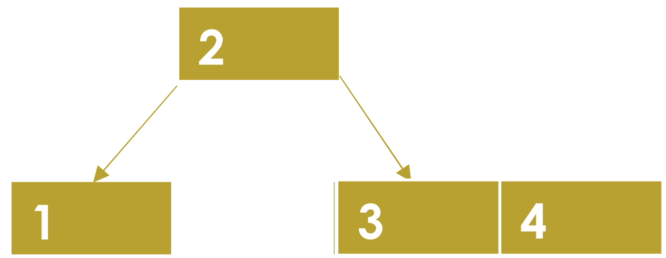
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 4



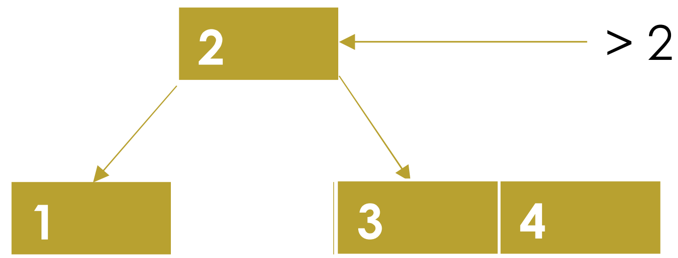
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 4



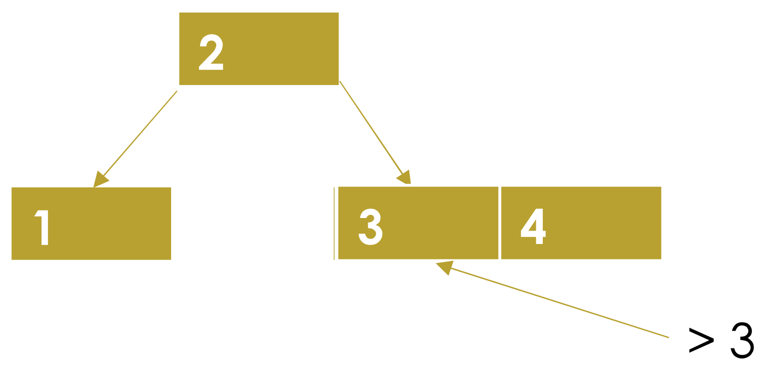
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 5



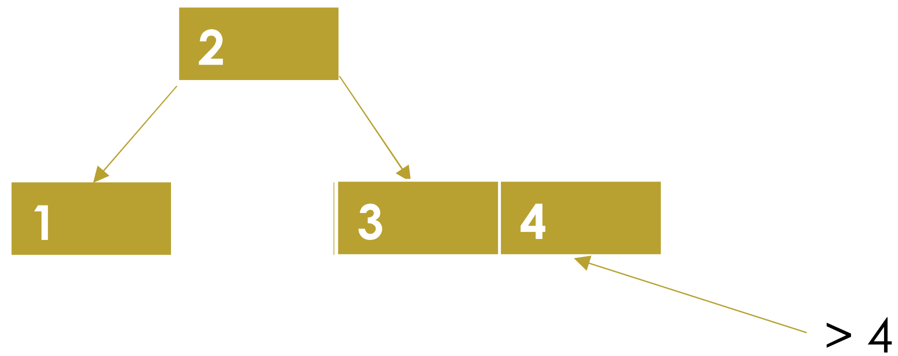
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 5



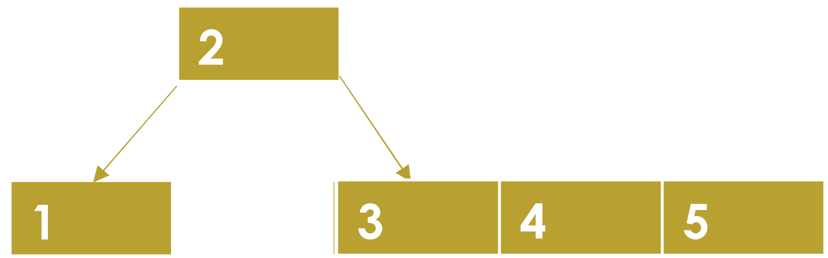
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 5



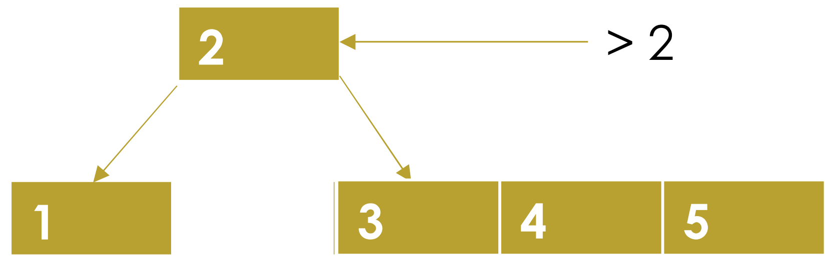
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 5



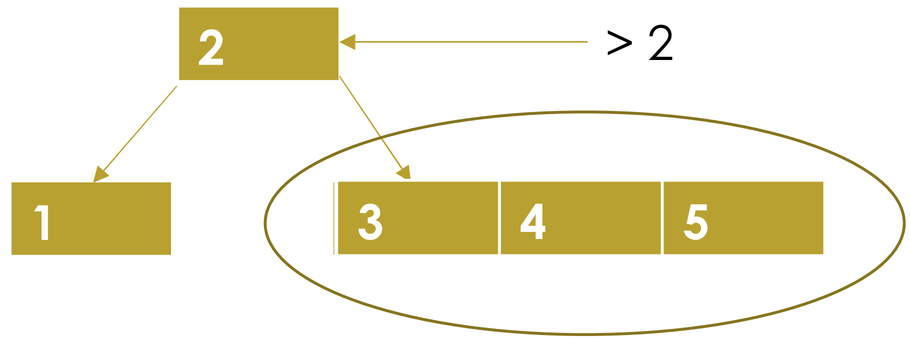
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 6



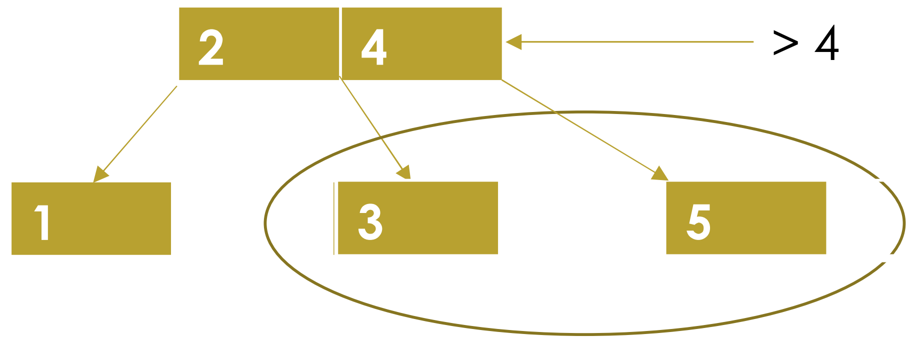
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 6



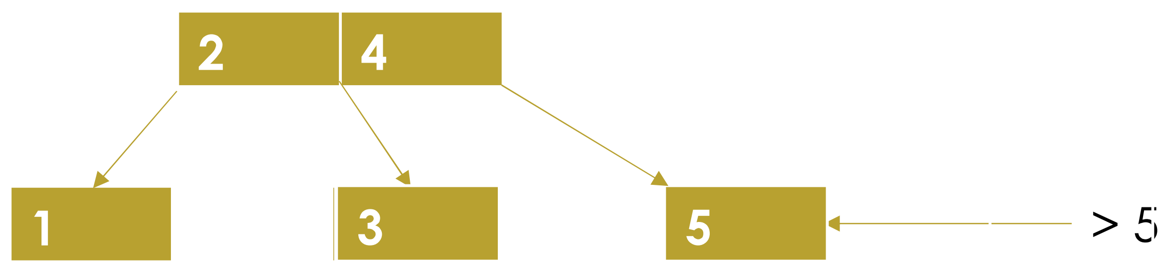
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 6



2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 6

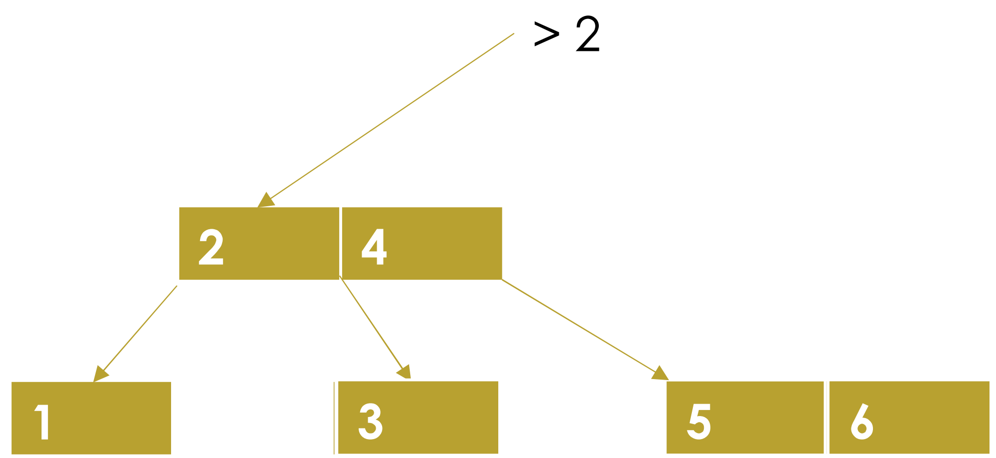


2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 6

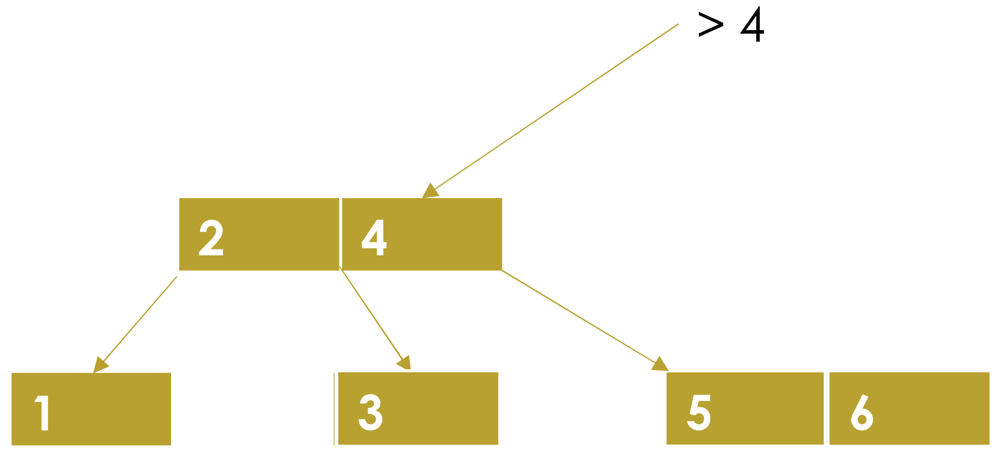
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_1



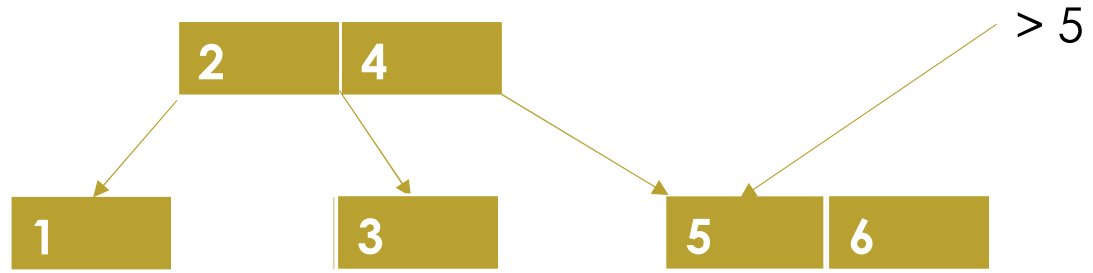
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_1



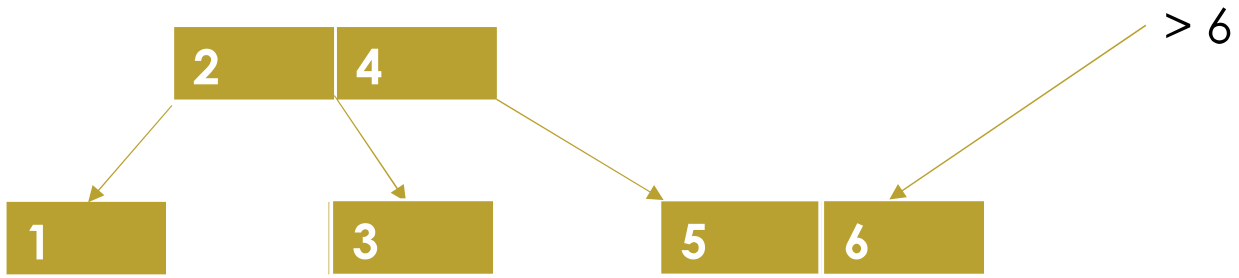
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_1



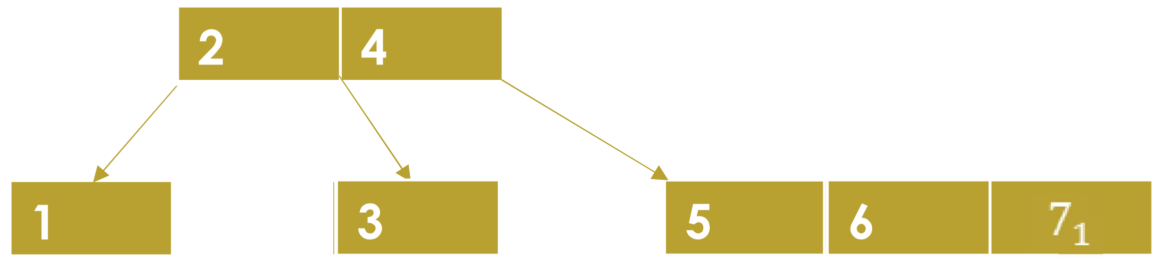
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_1



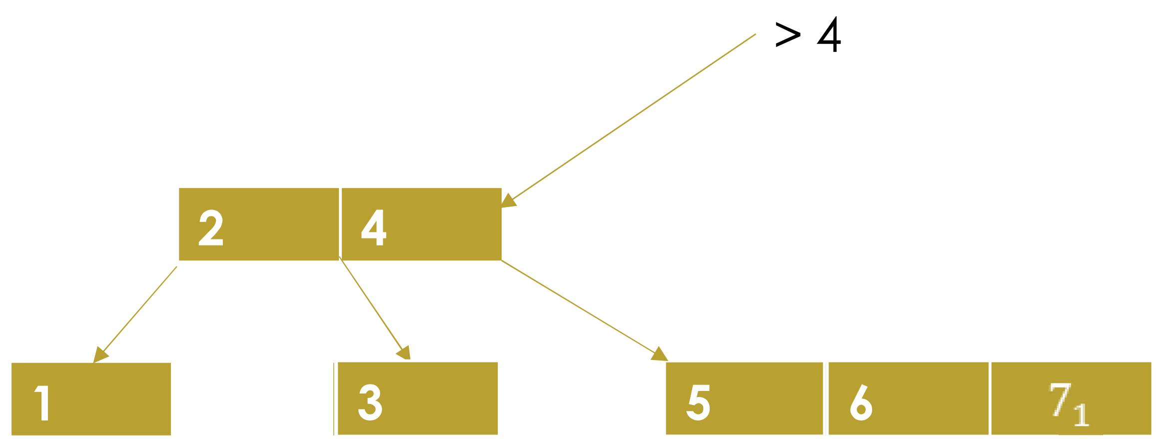
2-3-4 Tree Example

- $1,2,3,4,5,6,7_1,7_2,7_3,8$
- Insert 7_1



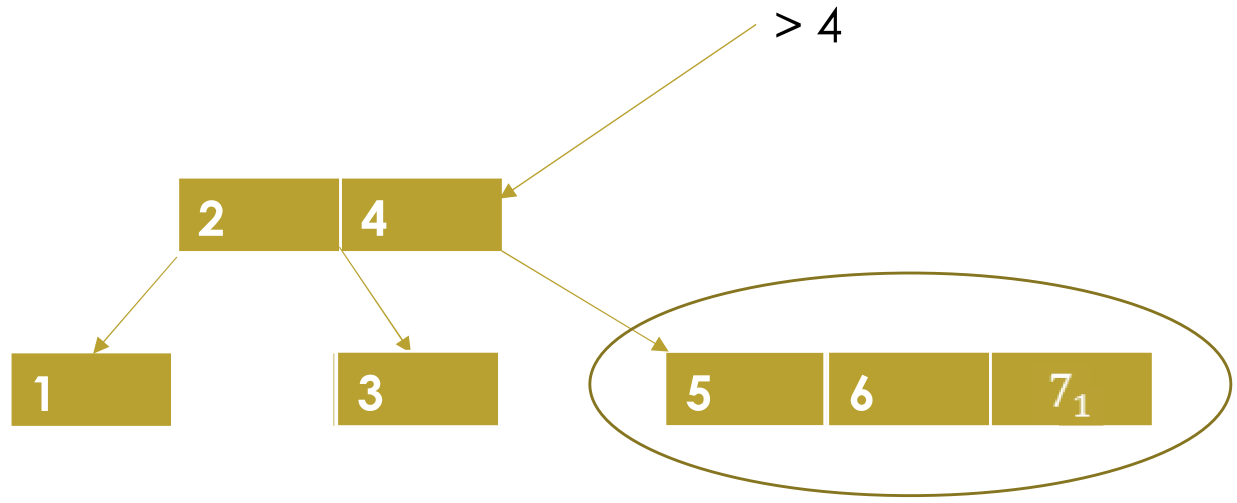
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_2



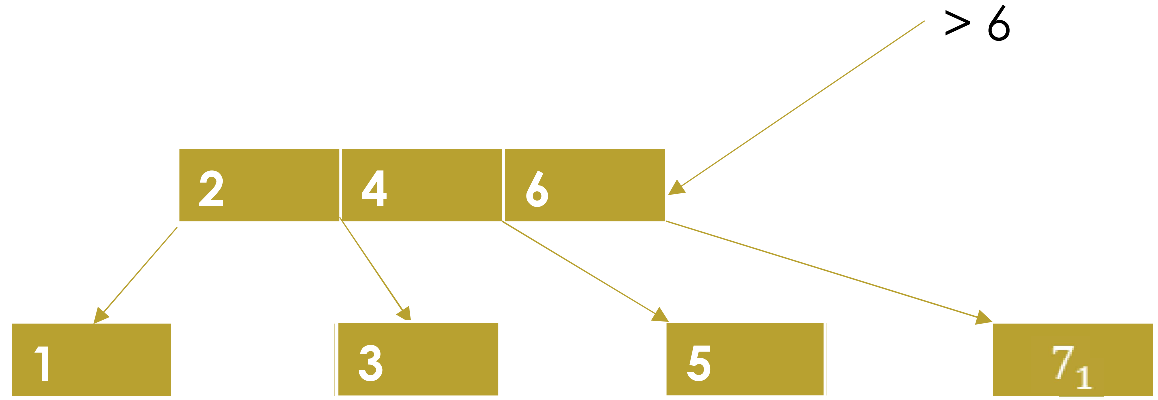
2-3-4 Tree Example

- $1,2,3,4,5,6,7_1,7_2,7_3,8$
- Insert 7_2



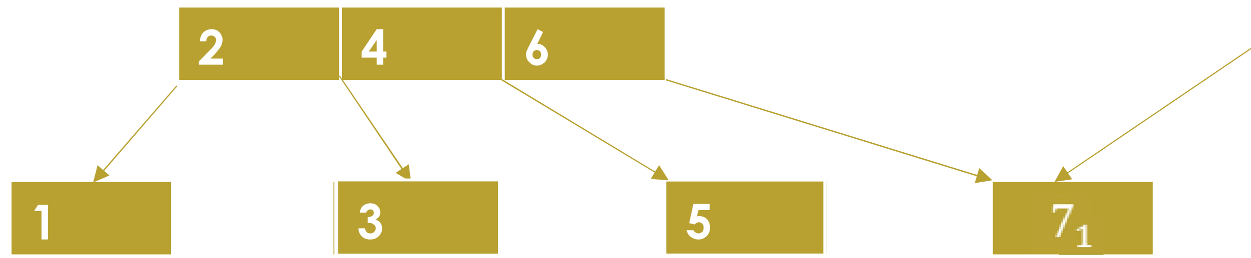
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_2



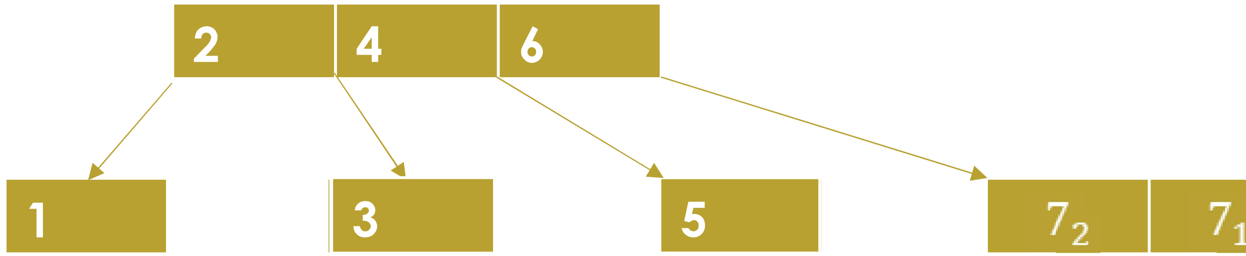
2-3-4 Tree Example

- $1,2,3,4,5,6,7_1,7_2,7_3,8$
- Insert 7_2



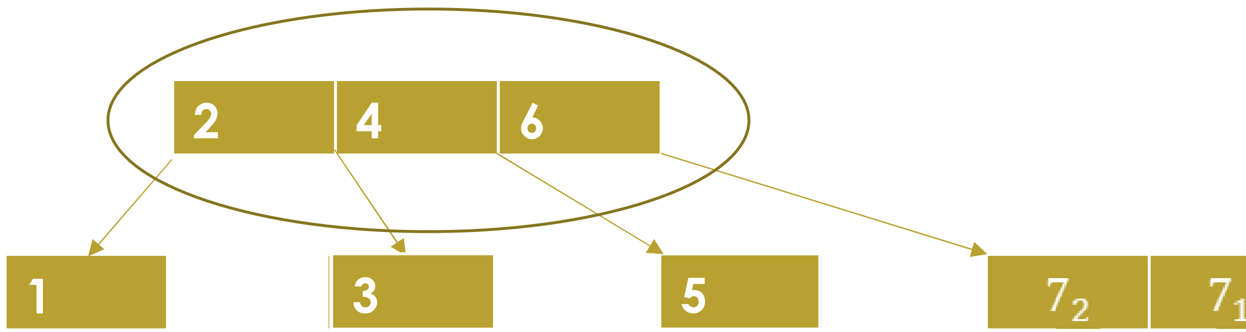
2-3-4 Tree Example

- $1,2,3,4,5,6,7_1,7_2,7_3,8$
- Insert 7_2



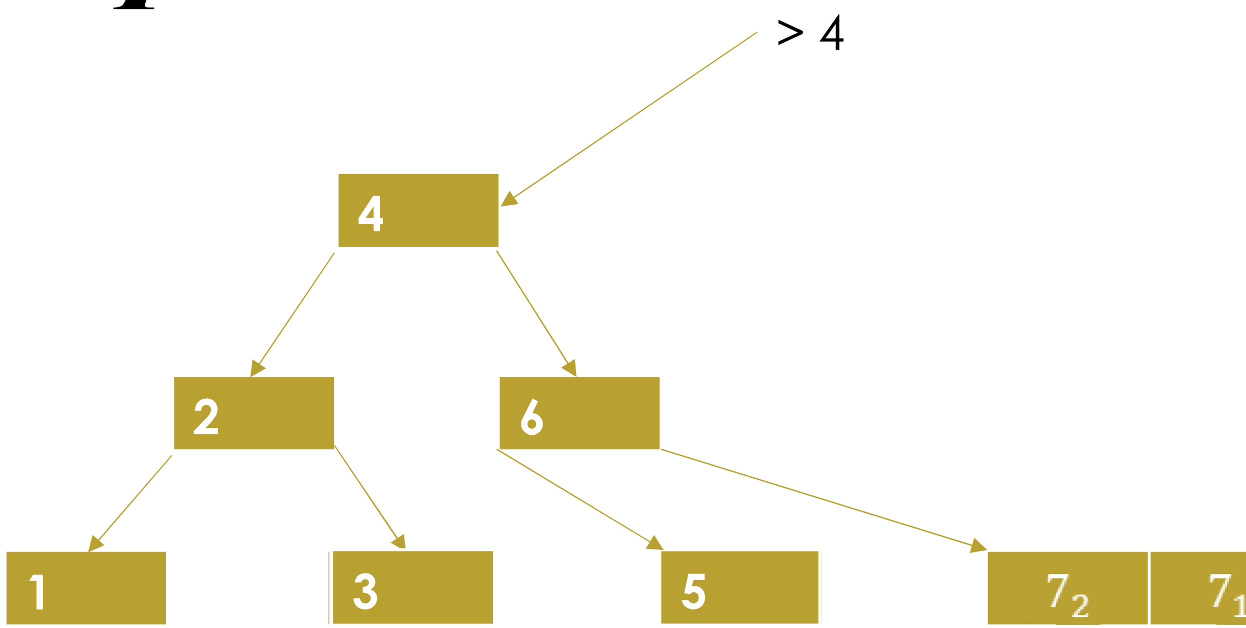
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_3



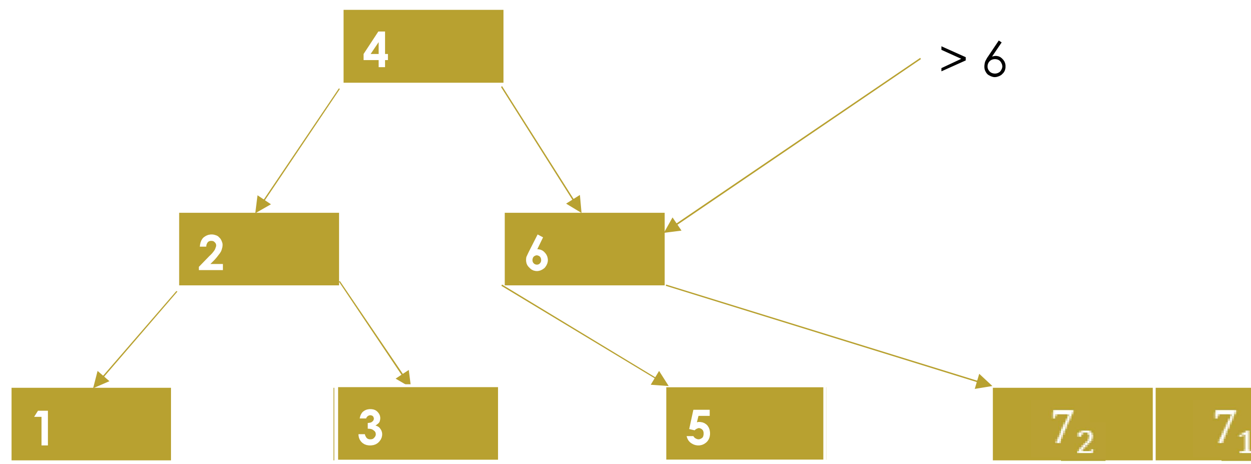
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_3



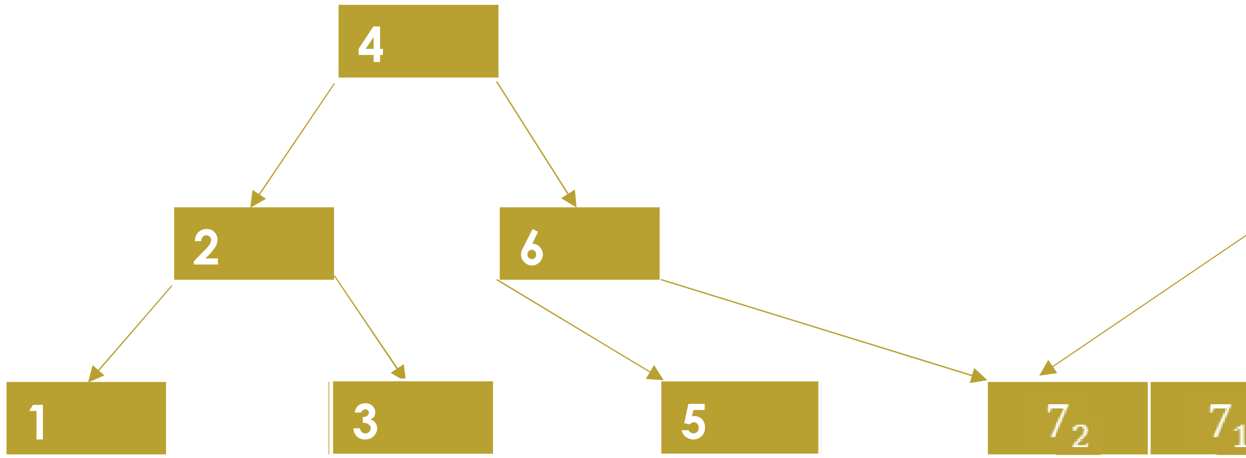
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_3



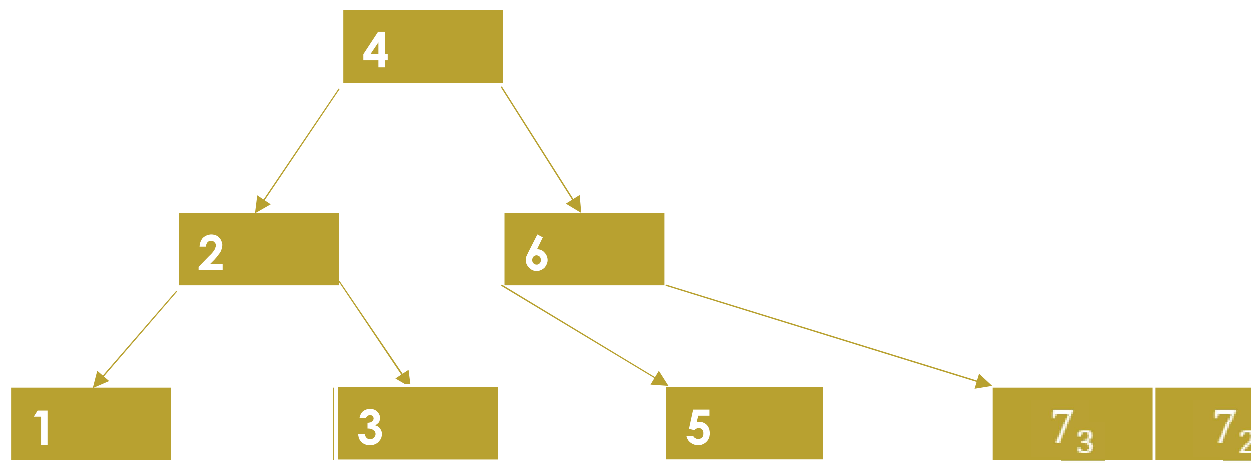
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_3



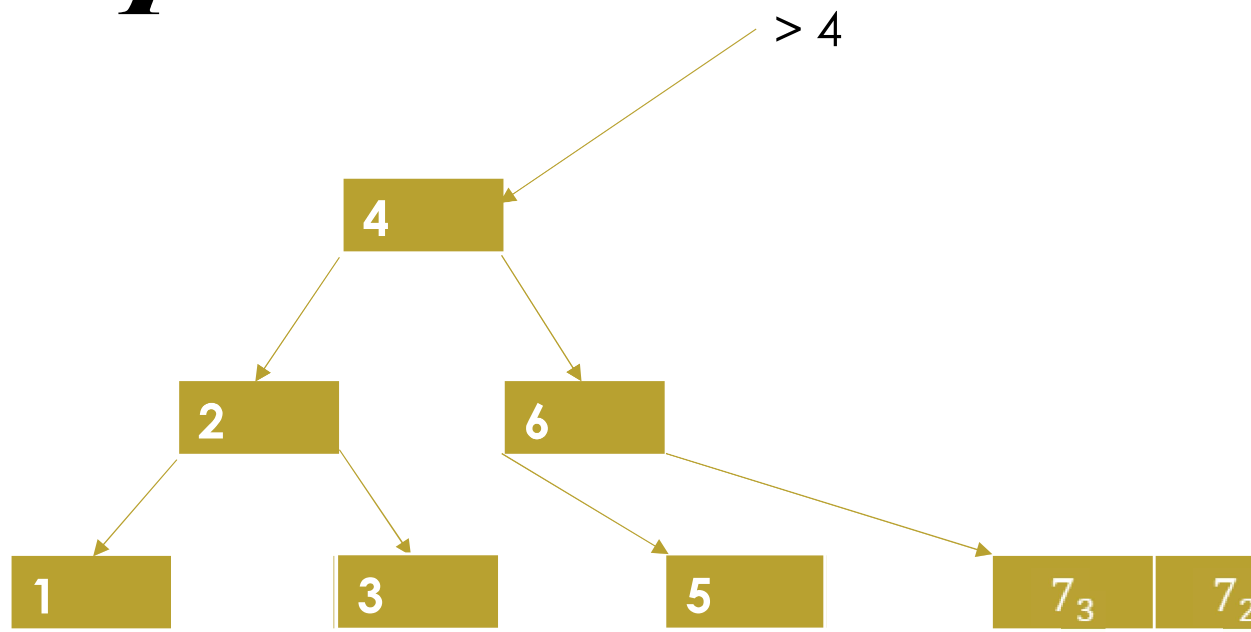
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 7_3



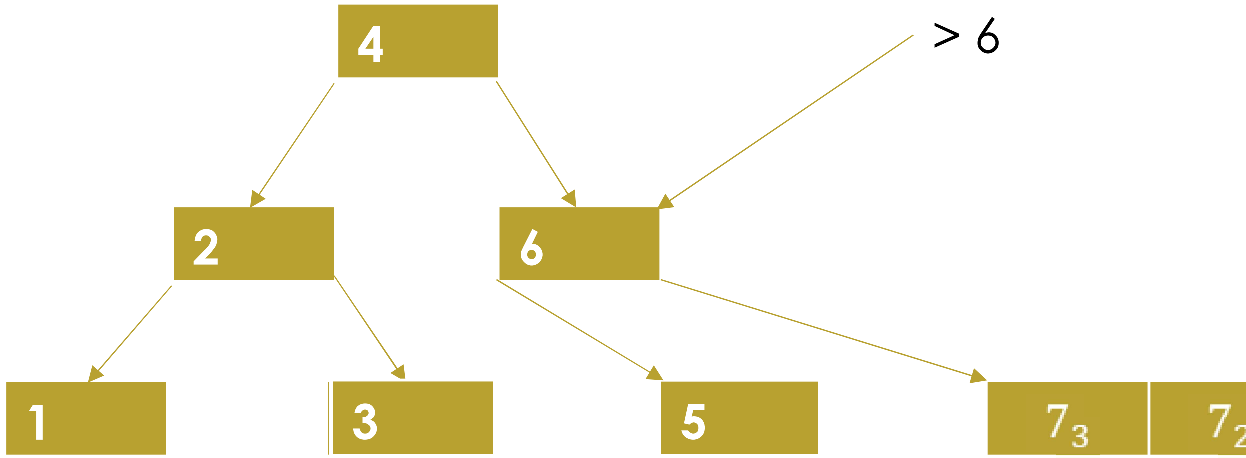
2-3-4 Tree Example

- $1,2,3,4,5,6,7_1,7_2,7_3,8$
- Insert 8



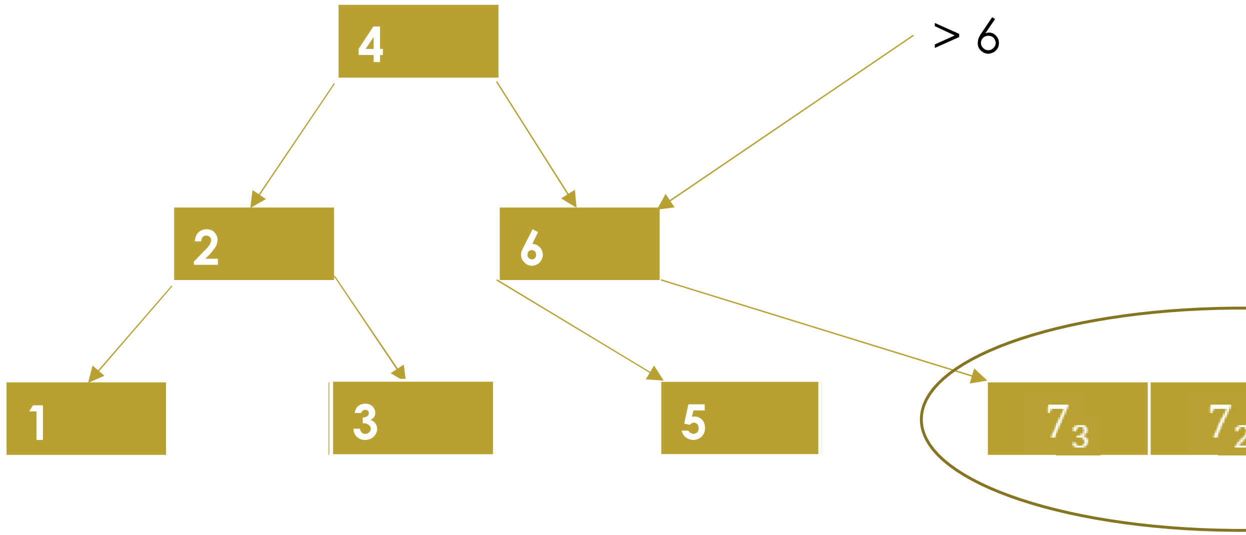
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 8



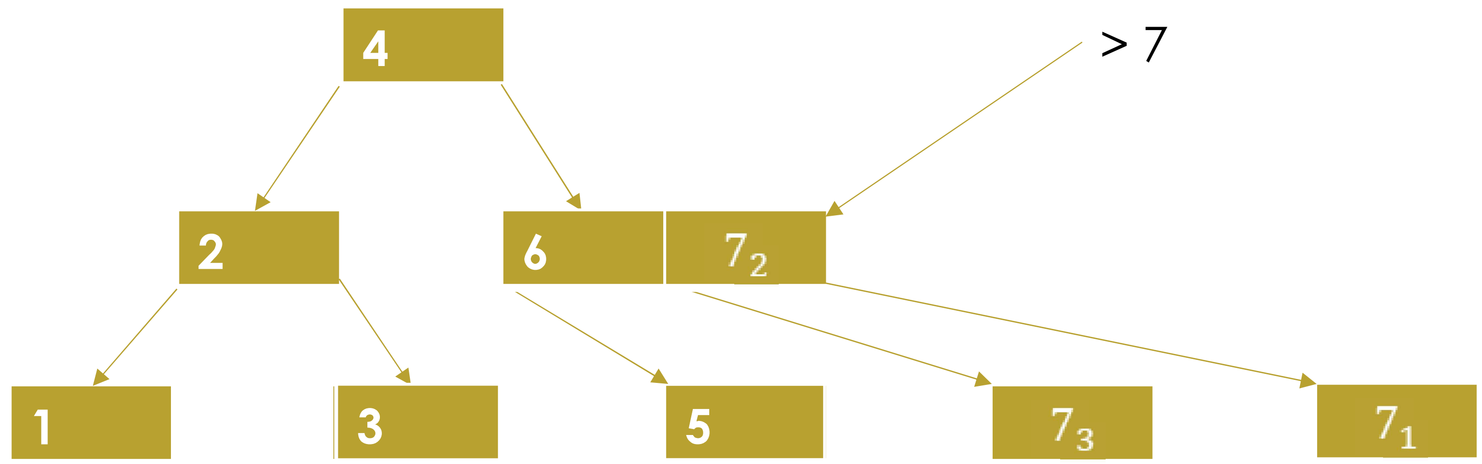
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 8



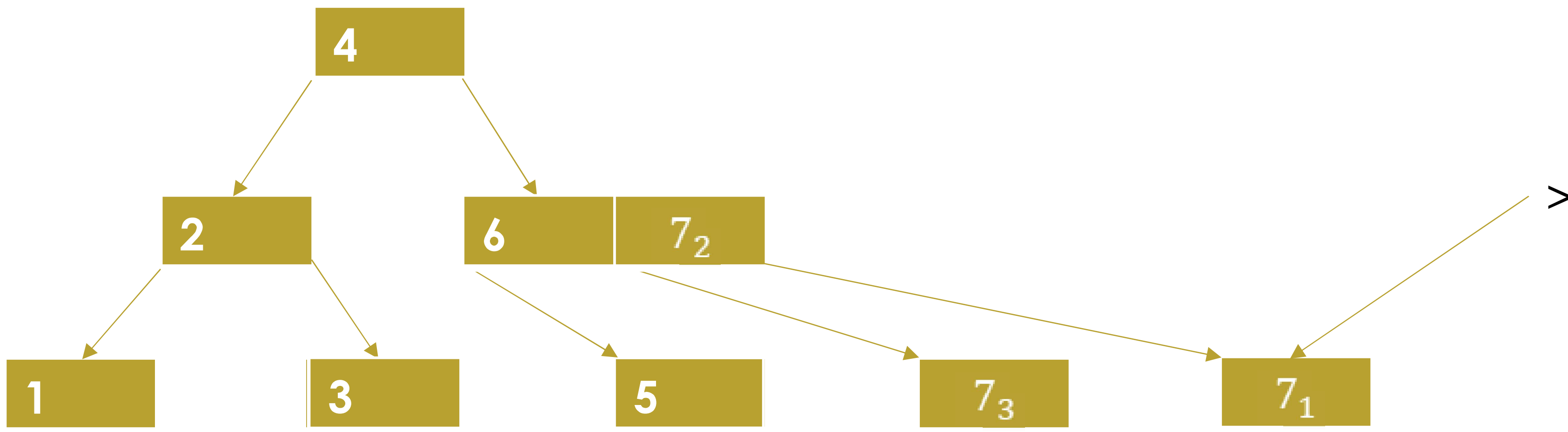
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 8



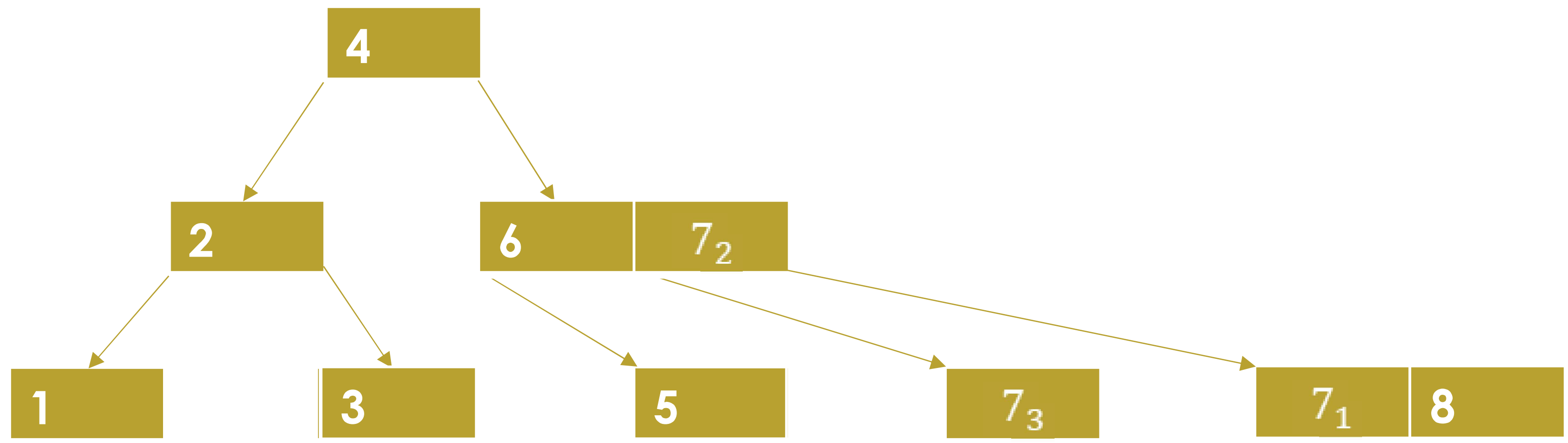
2-3-4 Tree Example

- $1,2,3,4,5,6,7_1,7_2,7_3,8$
- Insert 8



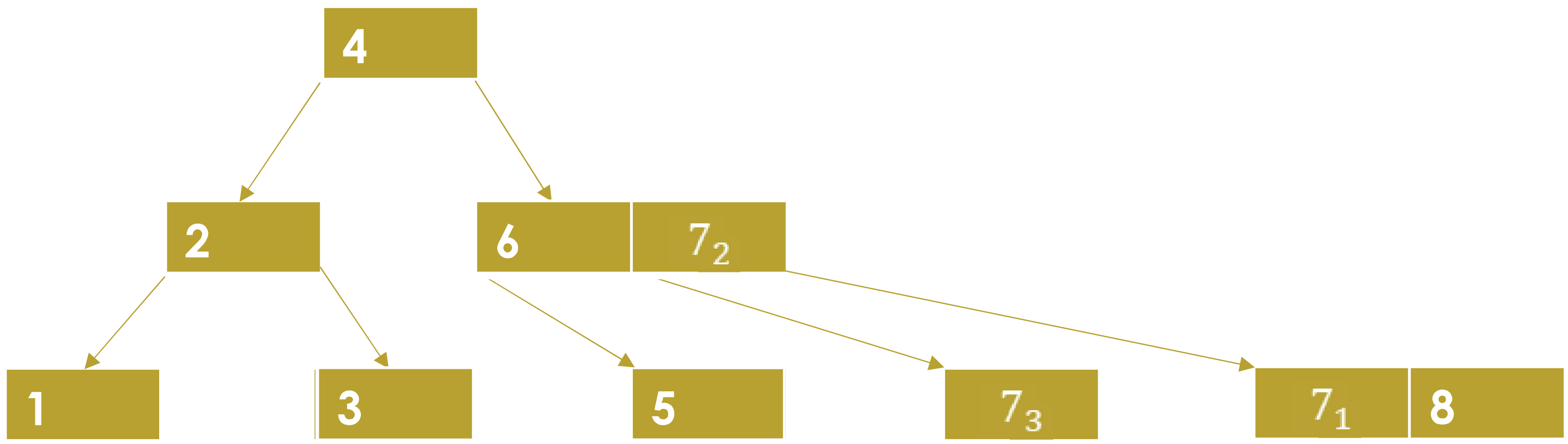
2-3-4 Tree Example

- 1,2,3,4,5,6,7_1,7_2,7_3,8
- Insert 8



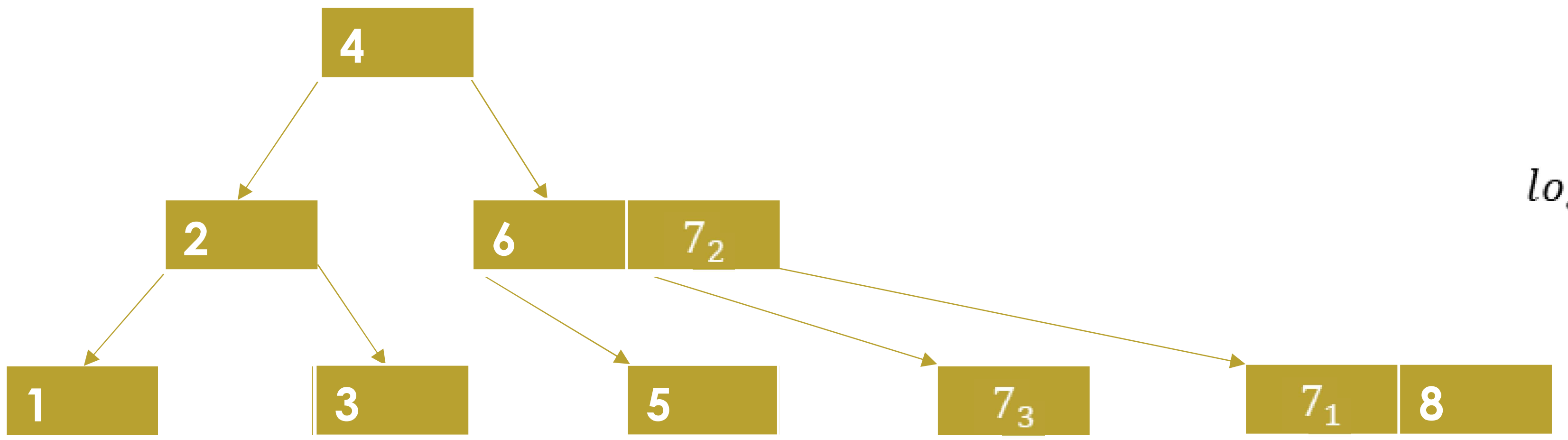
2-3-4 Tree

- Insert
  - O(?)
- Lookup
  - O(?)



2-3-4 Tree

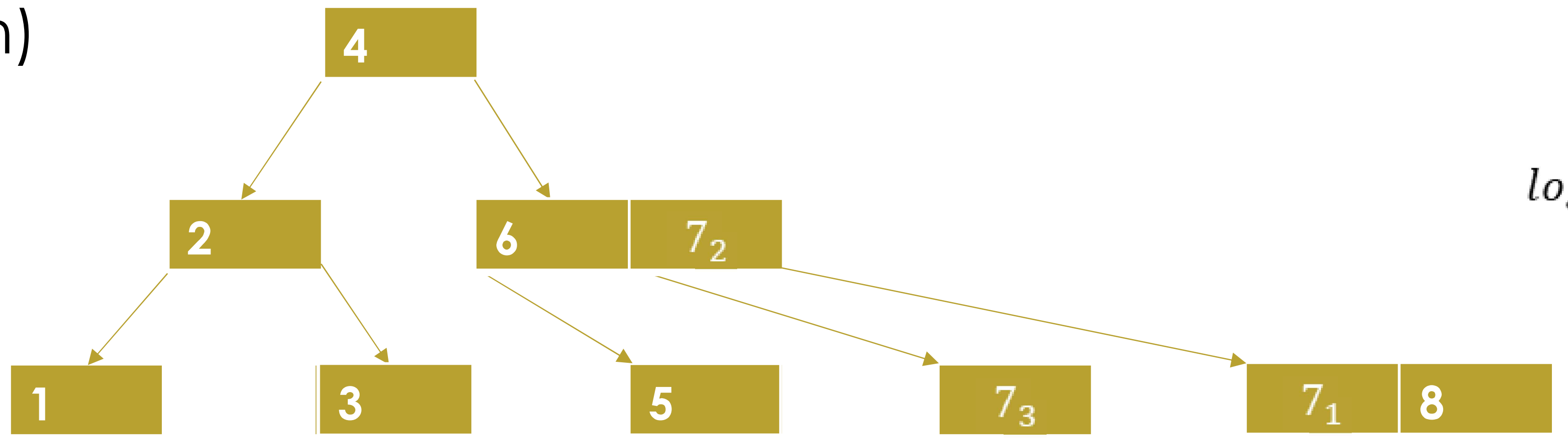
- Insert
  - $\Theta(\log n)$
- Lookup
  - O(?)



log_2

2-3-4 Tree

- Insert
  - $\Theta(\log n)$
- Lookup
  - $O(\log n)$



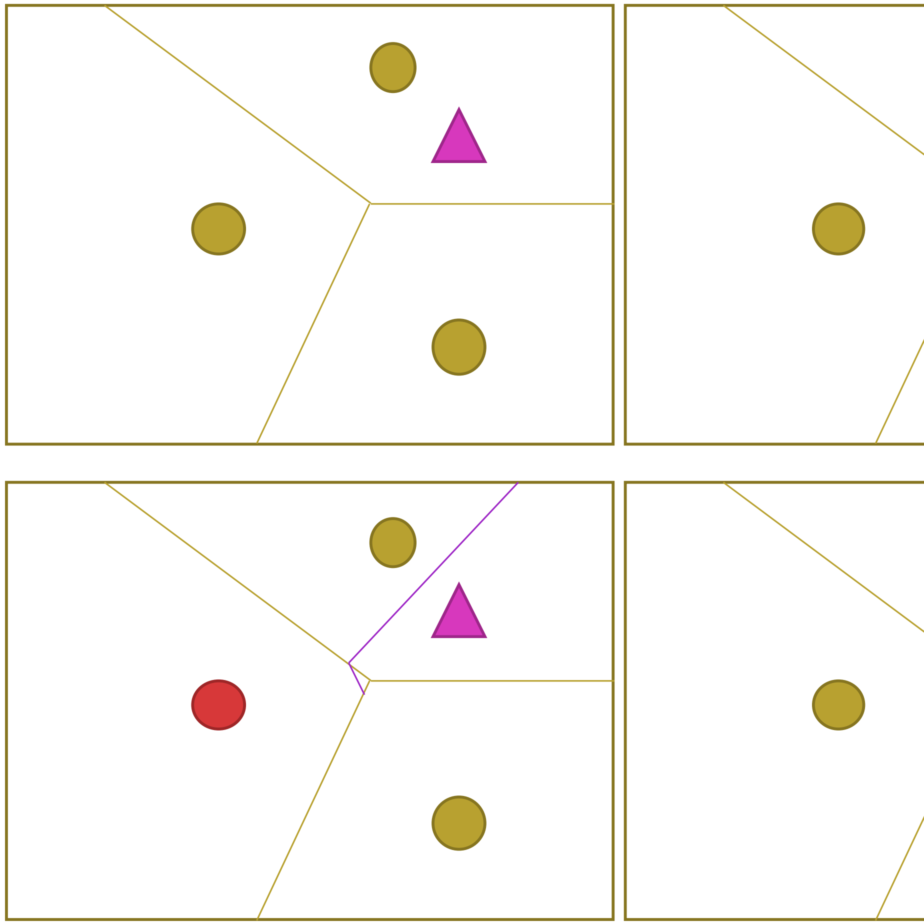
log_2

2-3-4 trees

- demo
- https://www.educative.io/page/5689413791121408/800

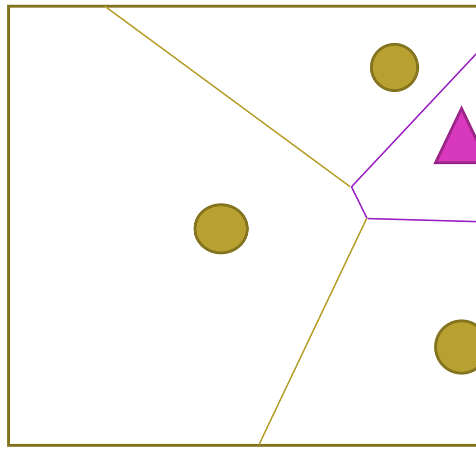
Incremental Voronoi Construction

- Assignment 4 parts
  - Calculate Bisectors
  - Intersection edge + points (code given)
  - Incrementally Construct Voronoi Diagram
  - Sort Faces by Largest Distance Between Vertices in Face



Incremental Voronoi Construction

- Incrementally Construct Voronoi Diagram
  - Find Face (Assignment 1, provided)
  - Add Bisector
  - Find Intersection (Provided)
  - Perform Split (Assignment 1, provided)
  - Set Edges Inside New Face to -1
  - Find furthest vertices (diameter) by looping through all pairs



Pair Programming Exercises

- Work on the exercises in pairs