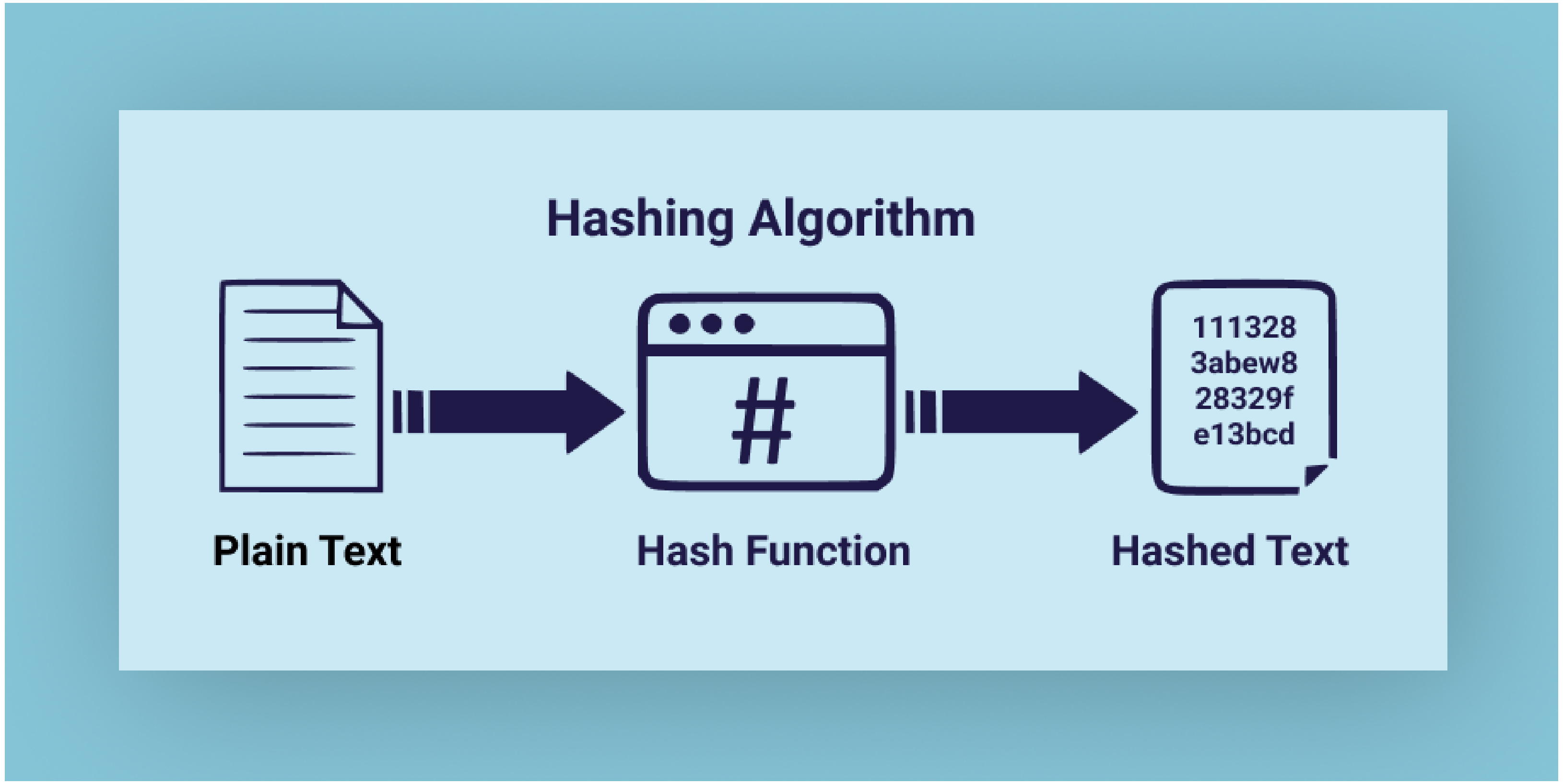
Data Structures, In Detail

CS 374

Hashing/Dictionaries

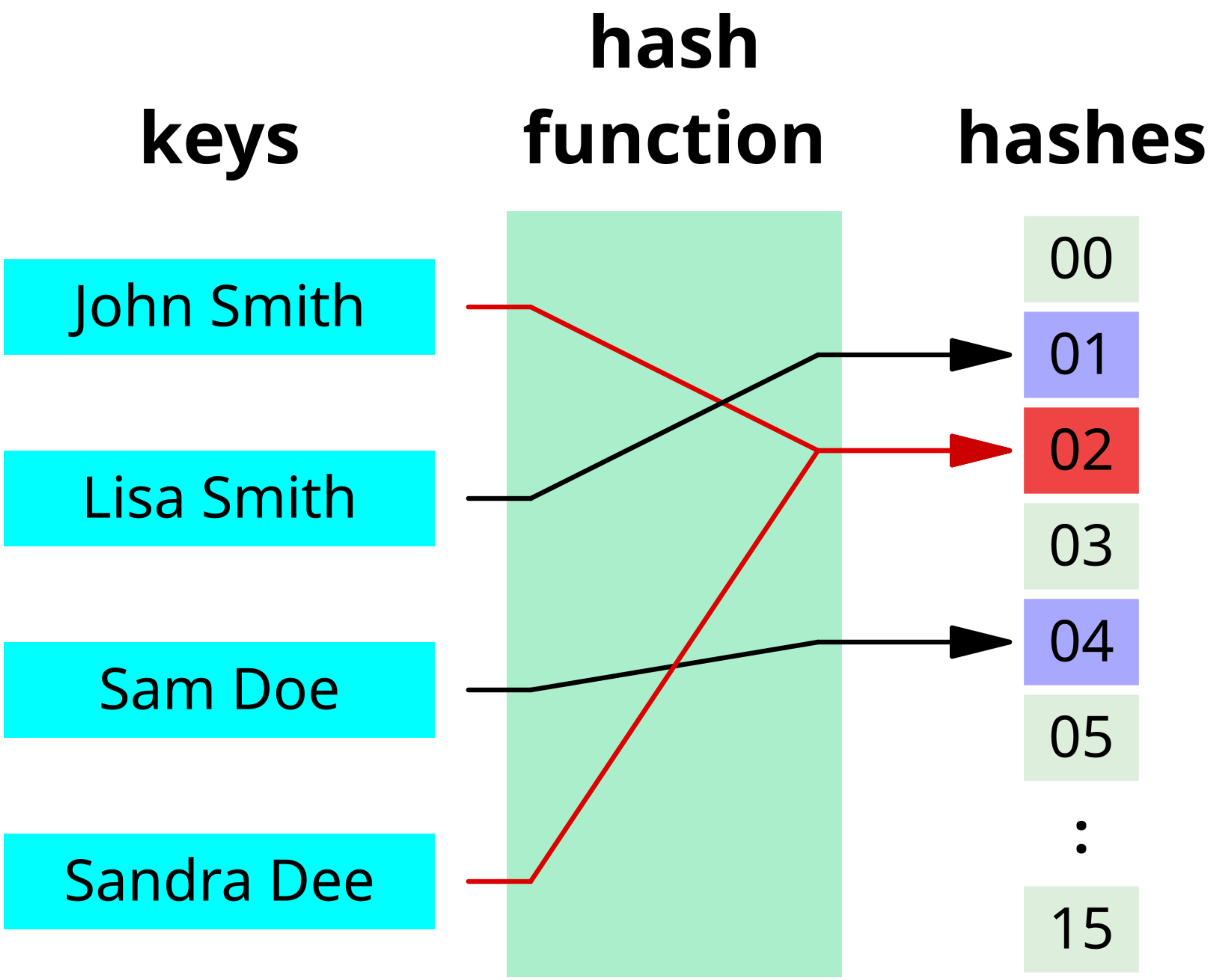

Hashing Algorithm
111328
3abew8
28329f
e13bcd
Plain Text
Hash Function
Hashed Text

Hashing

- A *hash* is a function that maps keys to an integer.
- Used in dictionaries, etc. to help you find it
- Also used for cryptography

- Have a function, then mod by m to stick it into spot m in the structure

- Collisions are an issue – what if two things have same hash?


hash function
keys
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
15

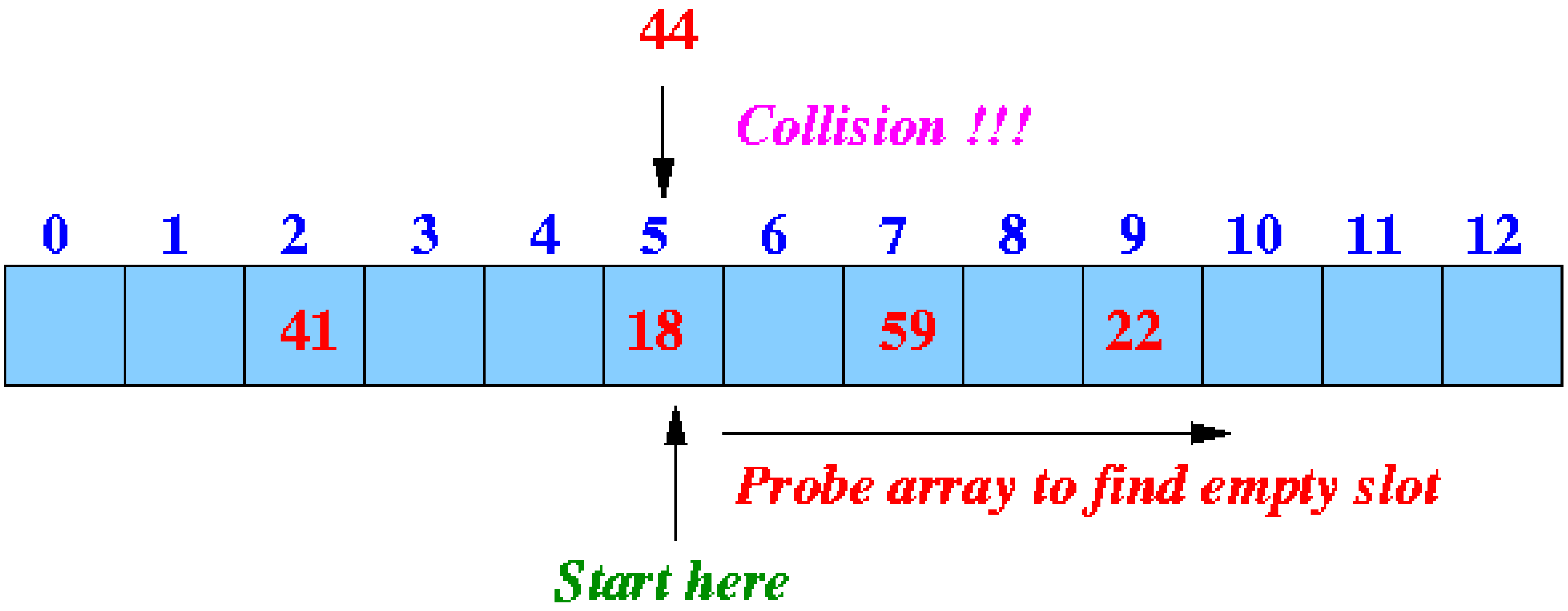
Example

- 3. Given a hash function of $f(x) = x + 31$ mod 11, determine *if* there is a collision for items inserted with the keys 60, 9, 45, 3, 70, 5, 90, 93, 39, 77, 70 in that order.
- At which item does the first collision occur?

0	1	2	3	4	5	6	7	8	9
34	5	55	21		2		3	8	13

Figure 3.11: Collision resolution by open addressing and sequential probing, after inserting the first eight Fibonacci numbers in increasing order with $H(x) = (2x + 1) \mod 10$. The red elements have been bumped to the first open slot after the desired location.

Linear Probing



Example

- 3. Given a hash function of $f(x) = x + 31$ mod 11, determine *if* there is a collision for items inserted with the keys 60, 9, 45, 3, 70, 5, 90, 93, 39, 77, 70 in that order.
- What is the result of the array when resolving collisions using linear/sequential probing? Use arrows/different colors to demonstrate when probing occurs.

Chaining

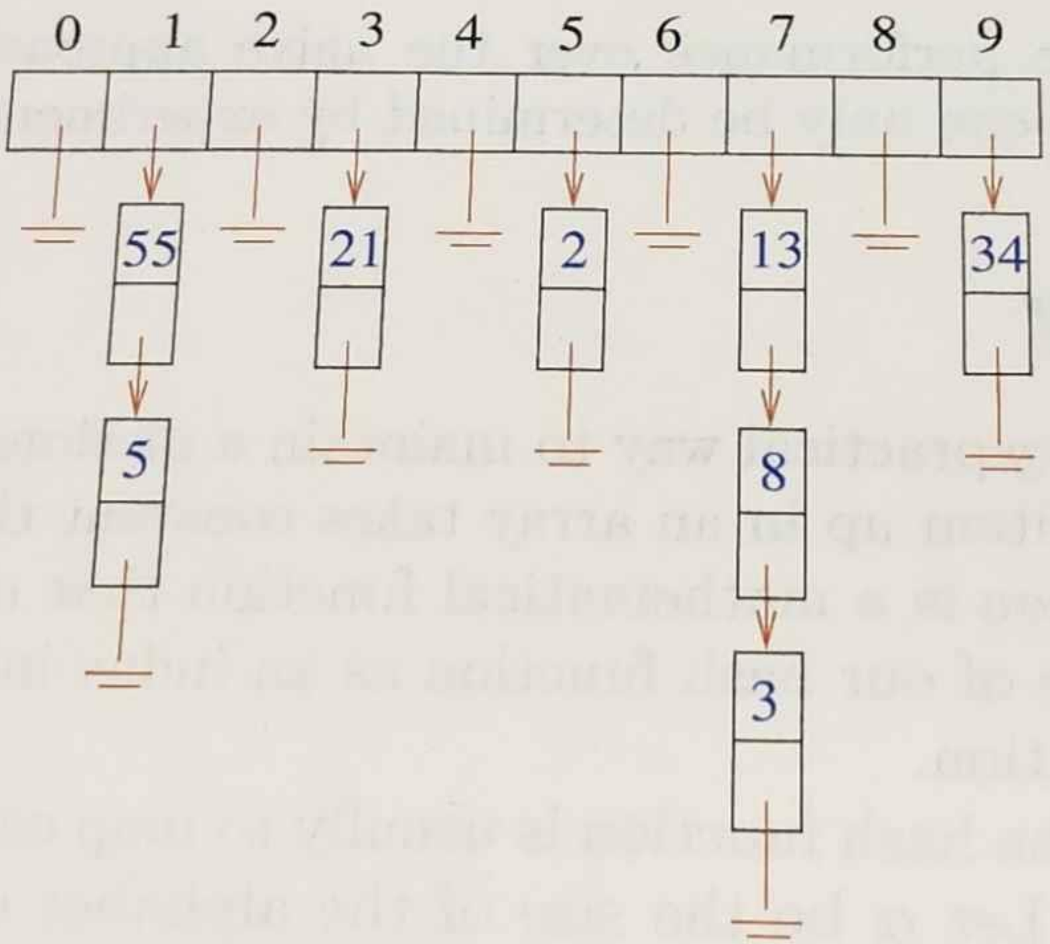


Figure 3.10: Collision resolution by chaining, after hashing the first eight Fibonacci numbers in increasing order, with hash function $H(x) = (2x + 1) \bmod 10$. Insertions occur at the head of each list in this figure.

Example

- 3. Given a hash function of $f(x) = x + 31$ mod 11, determine *if* there is a collision for items inserted with the keys 60, 9, 45, 3, 70, 5, 90, 93, 39, 77, 70 in that order.
- What is the result of the array when resolving collisions using chaining? Draw a picture.

$f(x) = x + 31$ mod 11: 60, 9, 45, 3, 70, 5, 90, 93, 39, 77, 70

$f(x) = x + 31 \bmod 11$: 60, 9, 45, 3, 70, 5, 90, 93, 39, 77, 70

More Collision Resolution

- Quadratic hashing
  - If value x matches to y,
  - Instead of moving y+1, move $y+1^2$.
  - Then, if $y+1^2$ is full, try $y+2^2$
  - Chances are, you'll find an empty spot faster.

- Double hashing
  - Have a separate hash function that you run when there are collisions

- And more!

How many things are in there?

- Hash table has fixed size
  - Like if implemented with arrays
- So, need to keep track of "how full" it is
- Load factor: $\alpha = \frac{n}{m}$
  - n = size of data (num. entries)
  - m = number of buckets (capacity)

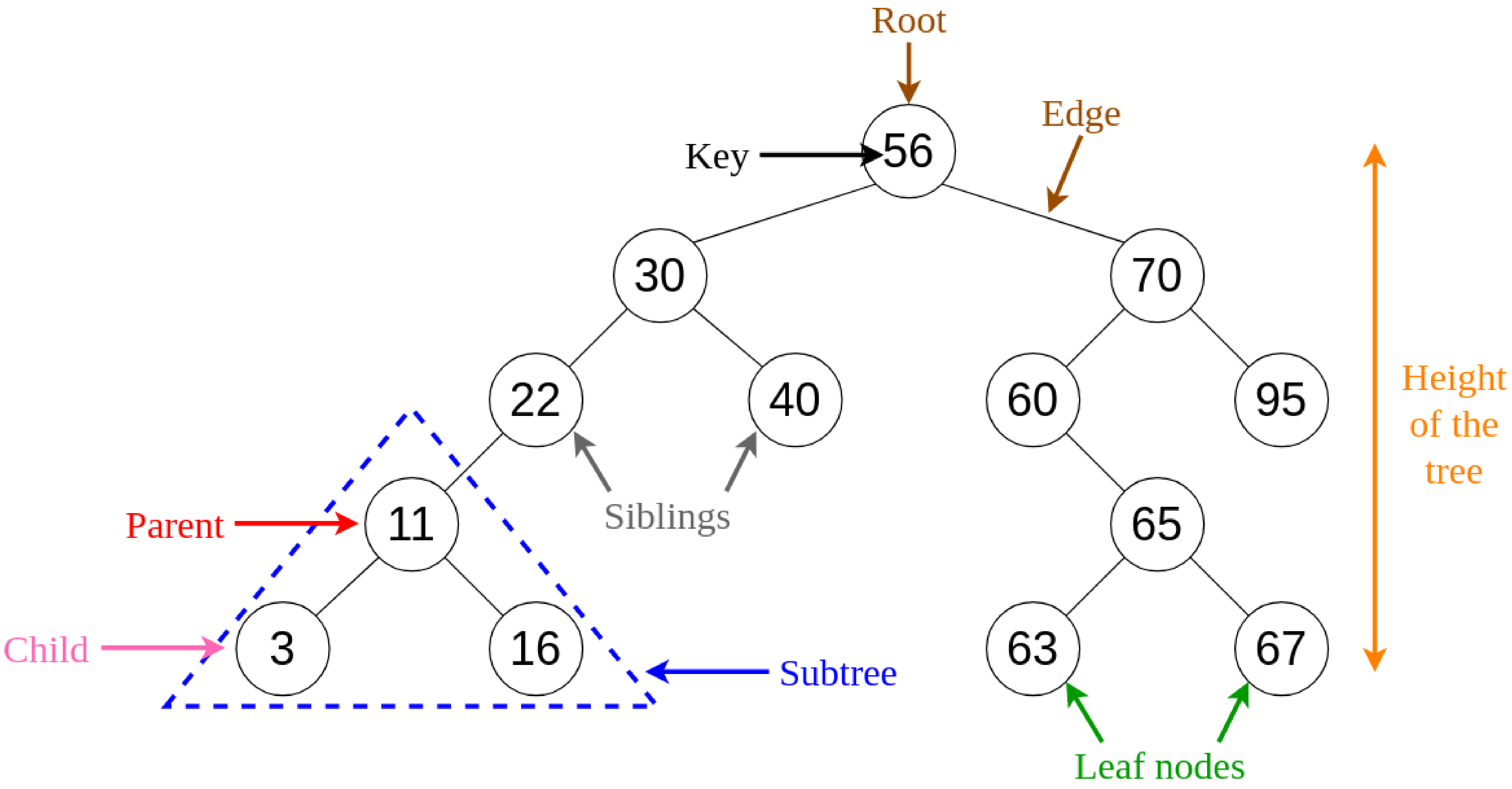
Load factor

- You decide how full you want it to get
  - Set some α_{max} threshold
- Resize, *then rehash* when it gets too big (α_{max}), or when gets too small ($\alpha_{max}/4$??)
- Also may depend on method of collision resolution

Hash Table Operations

	Hash table (expected)	Hash table (worst case)
Search(L, k)	$O(n/m)$	$O(n)$
Insert(L, x)	$O(1)$	$O(1)$
Delete(L, x)	$O(1)$	$O(1)$
Successor(L, x)	$O(n+m)$	$O(n+m)$
Predecessor(L, x)	$O(n+m)$	$O(n+m)$
Minimum(L)	$O(n+m)$	$O(n+m)$
Maximum(L)	$O(n+m)$	$O(n+m)$

Binary Search Trees


Root
Edge
Key
56
30
70
22
40
60
95
11
65
3
16
63
67
Parent
Child
Siblings
Subtree
Leaf nodes
Height of the tree

Insert the following items into a BST:
60, 9, 45, 3, 70, 5, 90, 93, 39, 77, 70

Tree A

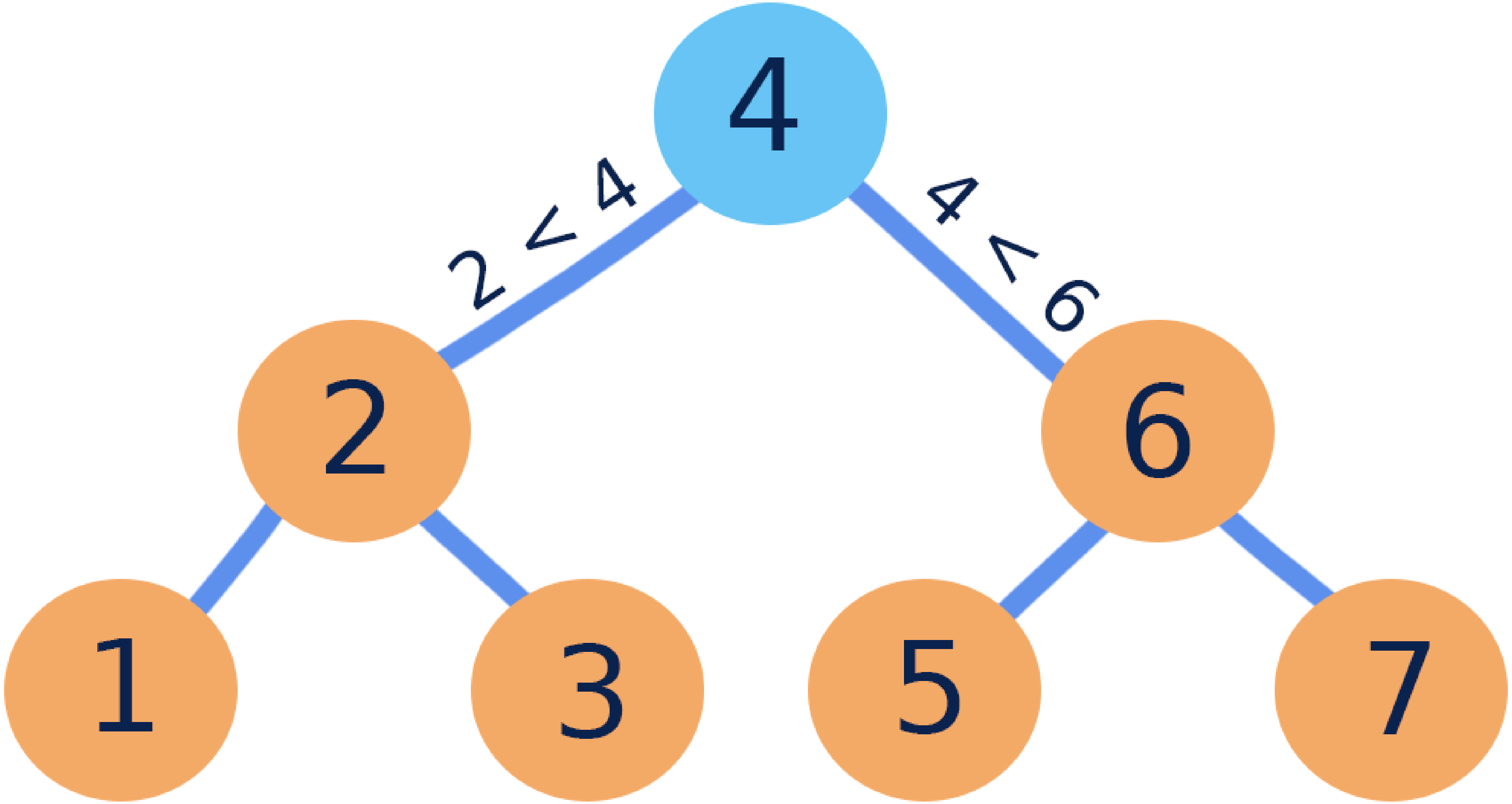
Tree B

Insert the following items into a BST:

58 47 84 39 55 14 98 53 79 42 87 3 90 1

BST Terms

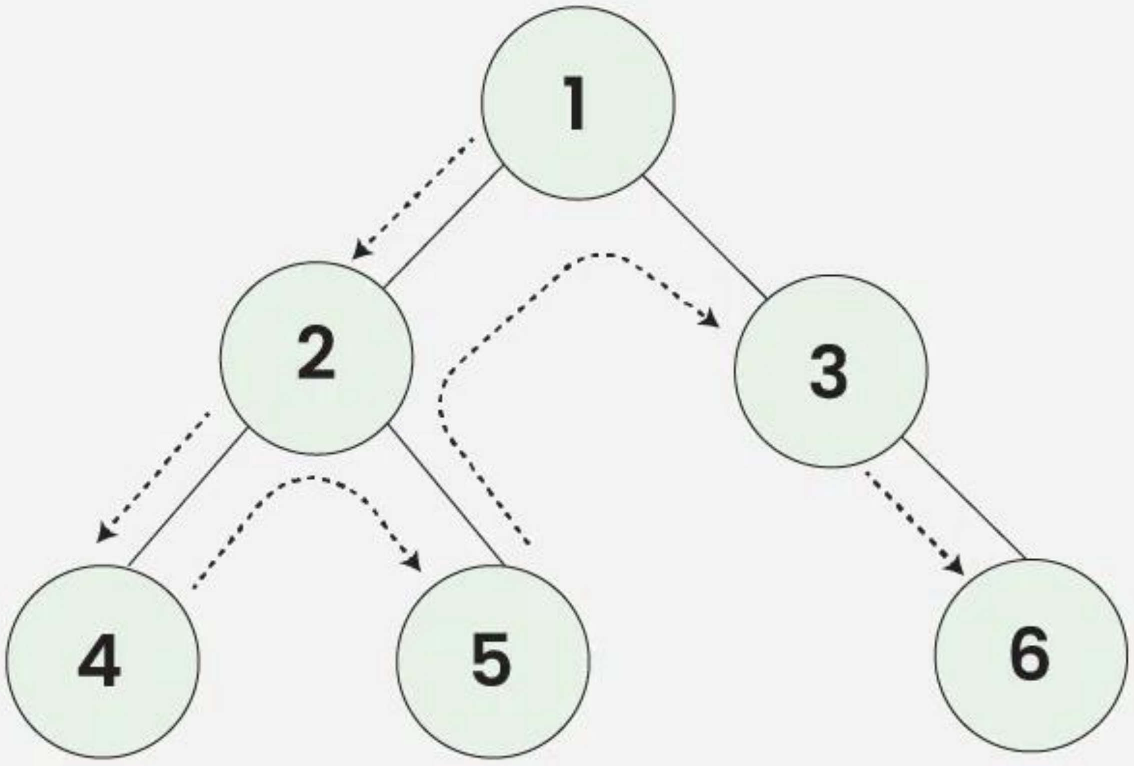
- Complete
  - All levels full
- Balanced
  - One side not more than other



In Order Traversal: 1 2 3 4 5 6 7

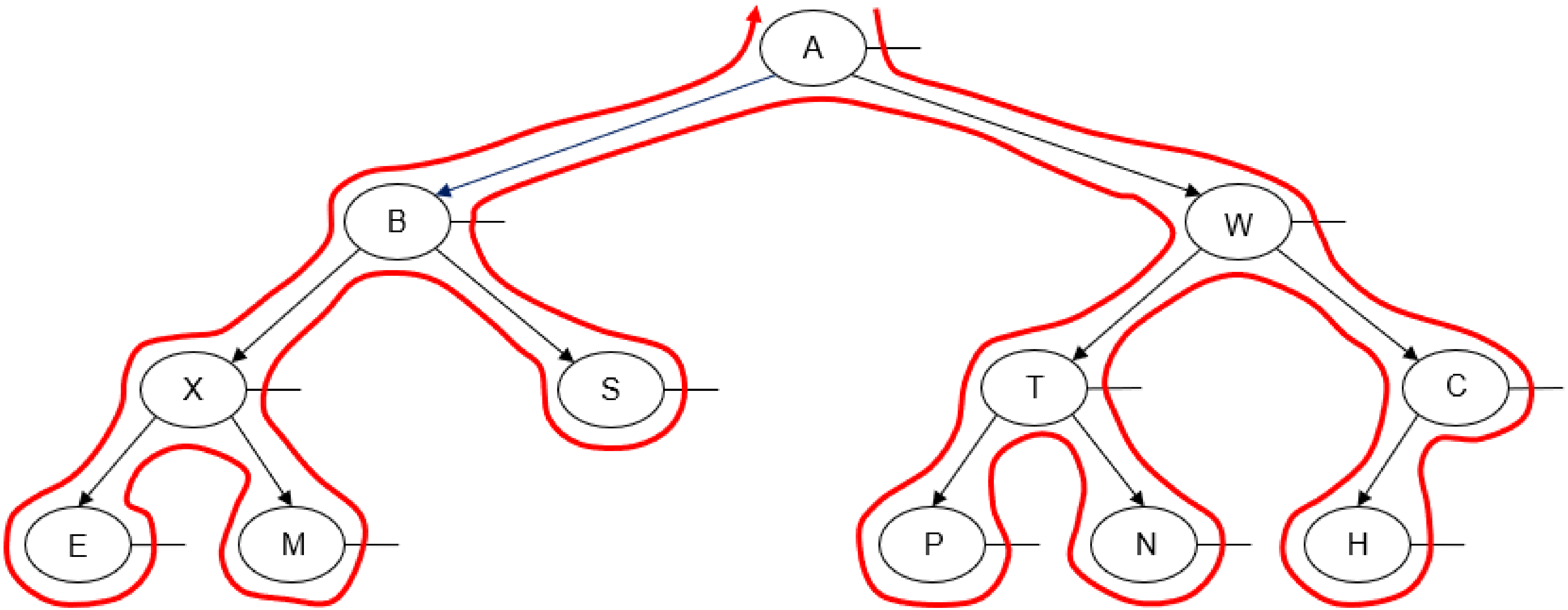
InOrder Traversal of tree A

Preorder Traversal of Binary Tree



Preorder Traversal: 1 → 2 → 4 → 5 → 3 → 6

PreOrder Traversal of tree A



- Post-order traversal: E M X S B P N T H C W A
- (this isn't a BST, just a BT)

PostOrder Traversal of tree A

BST Operations

- Search – O(h), where h= height of tree
- Find min/max – O(h)
- Traversal – O(n)
- Insertion – O(h)
- Deletion – O(h)

These slides are work-in-progress; more will be added