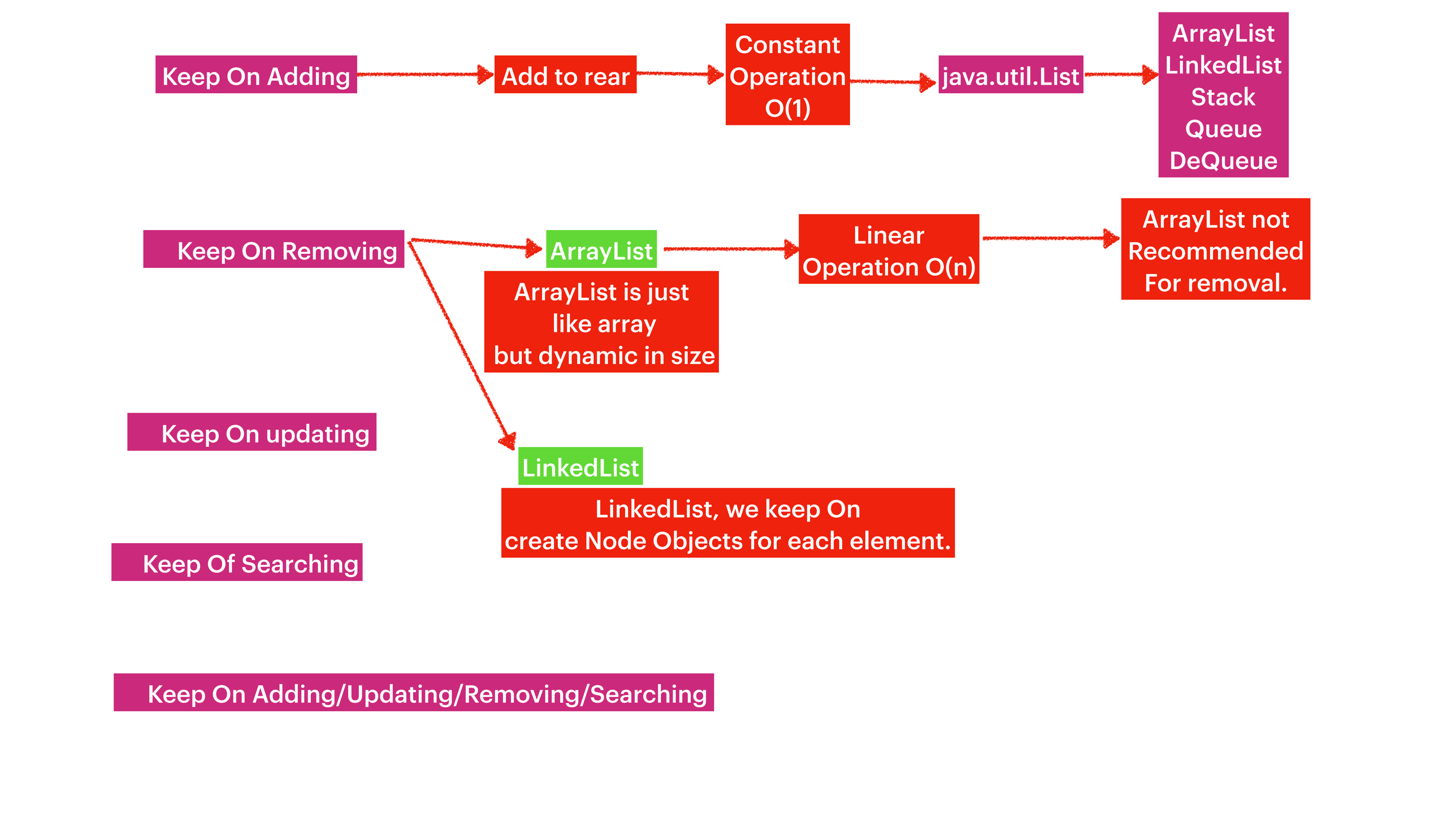

Keep On Adding
Add to rear
Constant Operation O(1)
java.util.List
ArrayList
LinkedList
Stack
Queue
DeQueue
Keep On Removing
ArrayList
Linear Operation O(n)
ArrayList not Recommended For removal.
ArrayList is just like array but dynamic in size
Keep On updating
LinkedList
LinkedList, we keep On create Node Objects for each element.
Keep Of Searching
Keep On Adding/Updating/Removing/Searching

On ArrayList

ArrayList : internal using array :

ArrayList
size 10

5 elements to ArrayList : 5 to []
when size Of ArrayList > n/2 , creates new array with double size then copy element in new array, removes the old one.

size 20 : [....5elements , 15 empty]

add 5 elements :

20 : [....10elements , 10 empty]

when you try to add 11 elements

11 > 20/2

40 :[....11elements , 29 empty]

Remove element from Array

int arr[] = {10 [0] , 11[1] , 12[2], 13[3] };

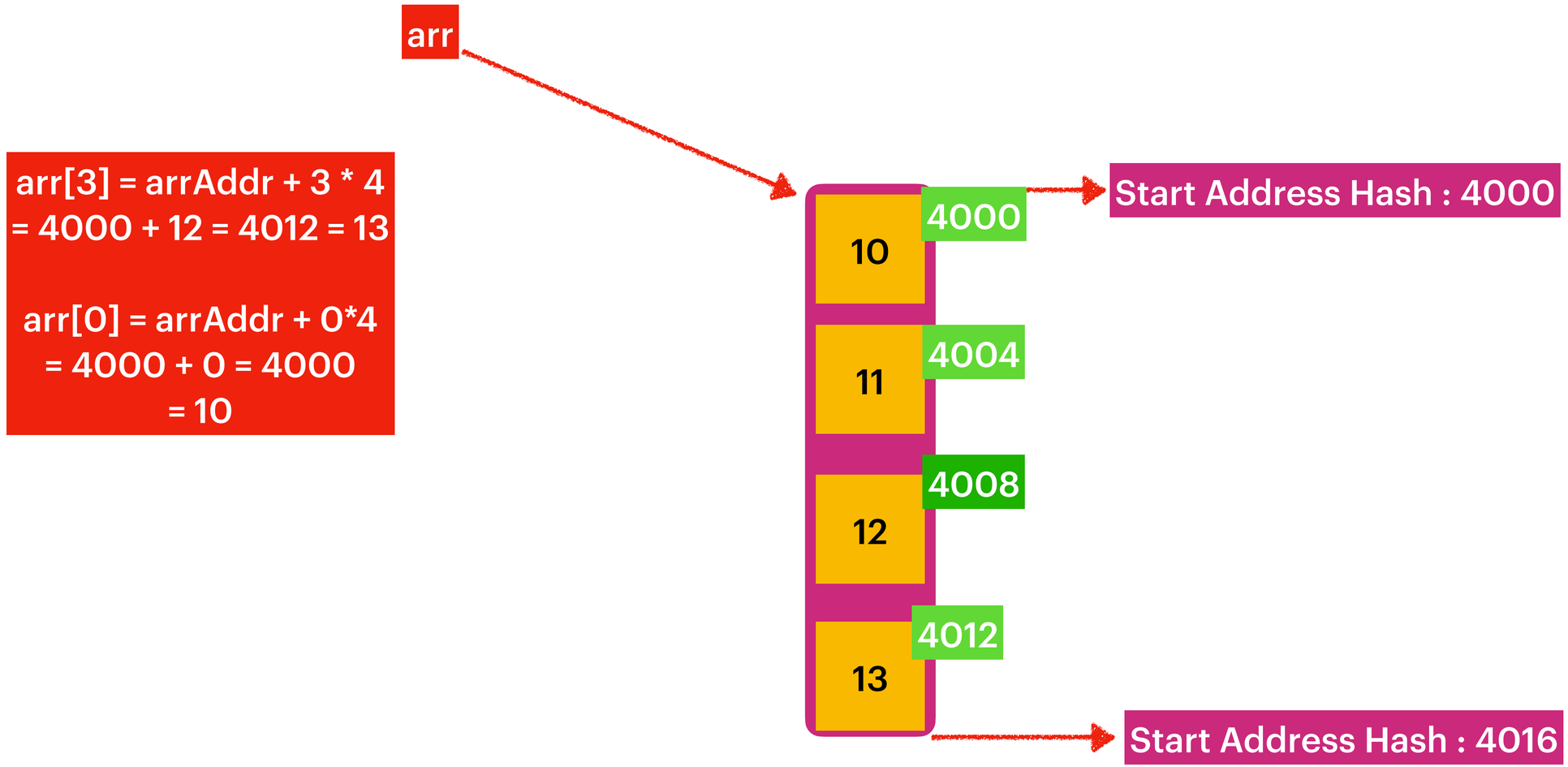
size : 4
index start 0 to n-1 : 0 to 3

Access element through index in constant time

arr[40] = ? returns value of inex 40 in a constant time.

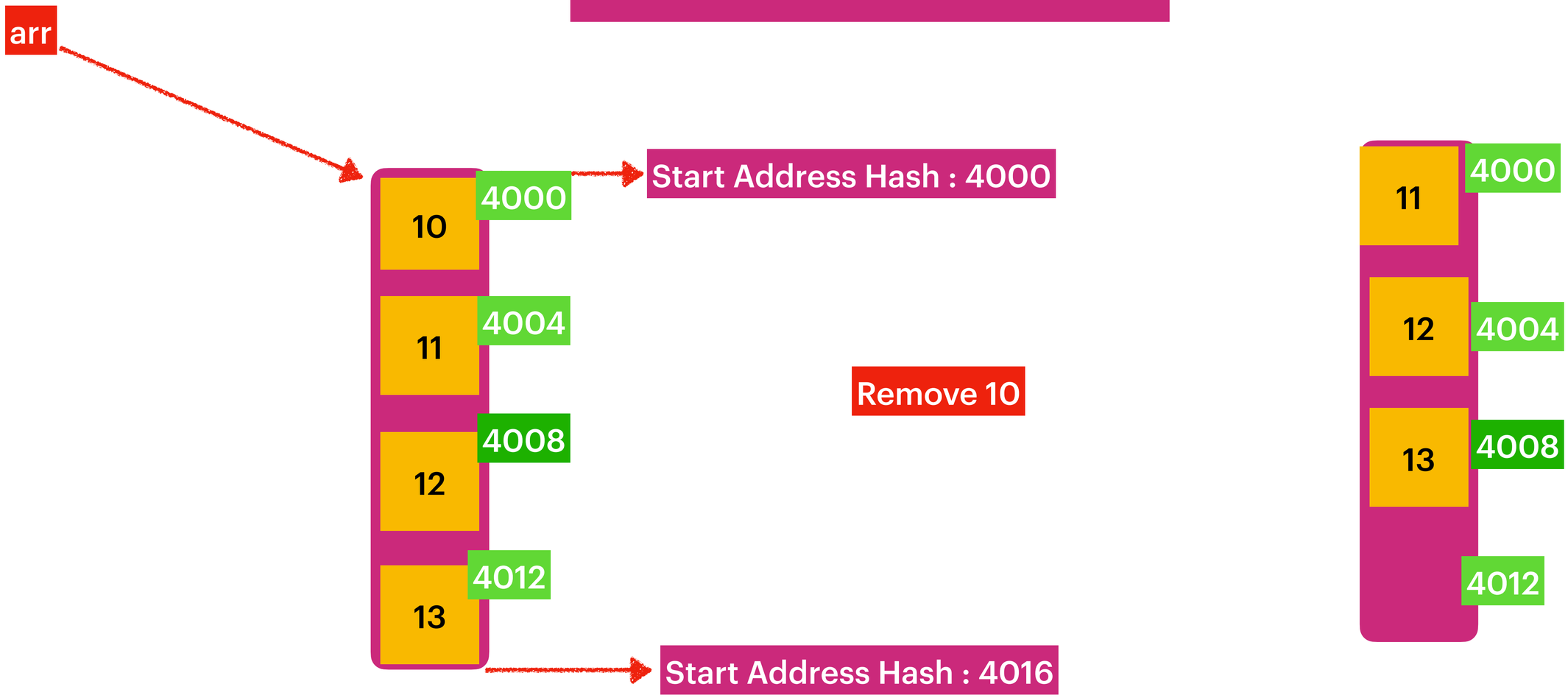
Array ::

int arr[] = {10 [0] , 11[1] , 12[2], 13[3] };

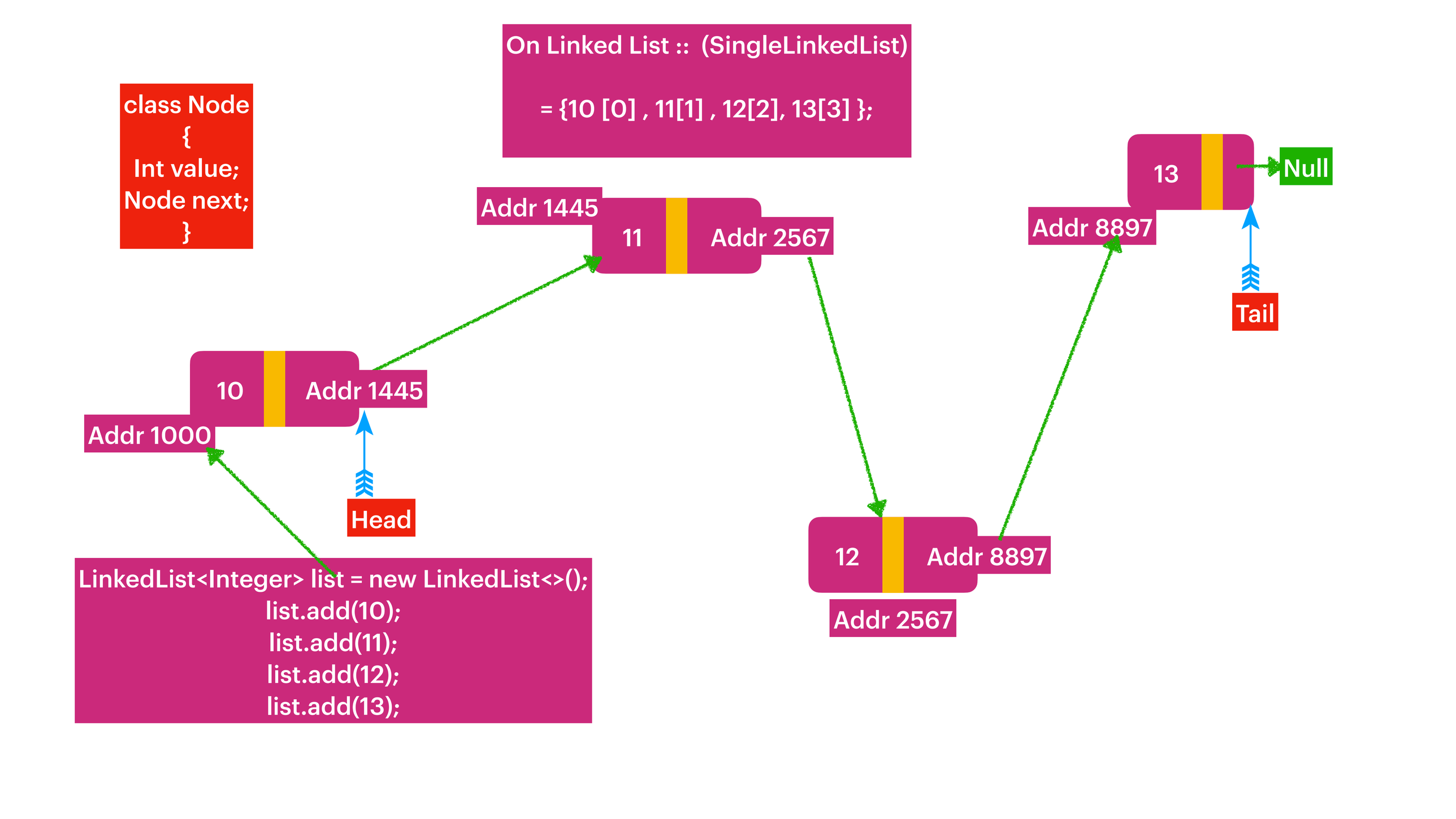


Array ::

int arr[] = {10 [0] , 11[1] , 12[2], 13[3] };



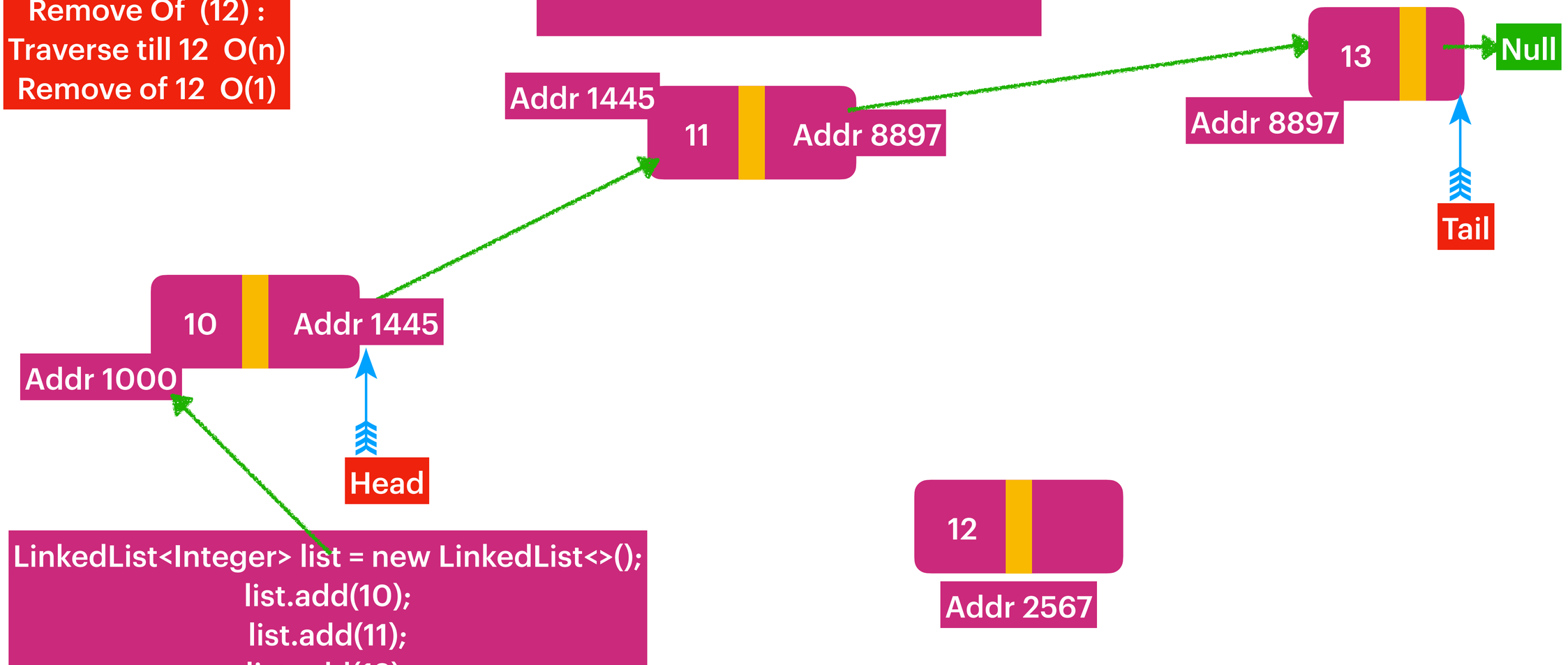
On ArrayList :
Insertion : O(1)
Deletion : O(n)
Search : O(1) on index based
Seach : O(n) on value based


On Linked List :: (SingleLinkedList)
= {10 [0] , 11[1] , 12[2], 13[3] };
class Node
{
Int value;
Node next;
}
Addr 1445
11
Addr 2567
13
Null
Addr 8897
Tail
10
Addr 1445
Addr 1000
Head
12
Addr 8897
Addr 2567
LinkedList<Integer> list = new LinkedList<>();
list.add(10);
list.add(11);
list.add(12);
list.add(13);

On Linked List :: (SingleLinkedList)

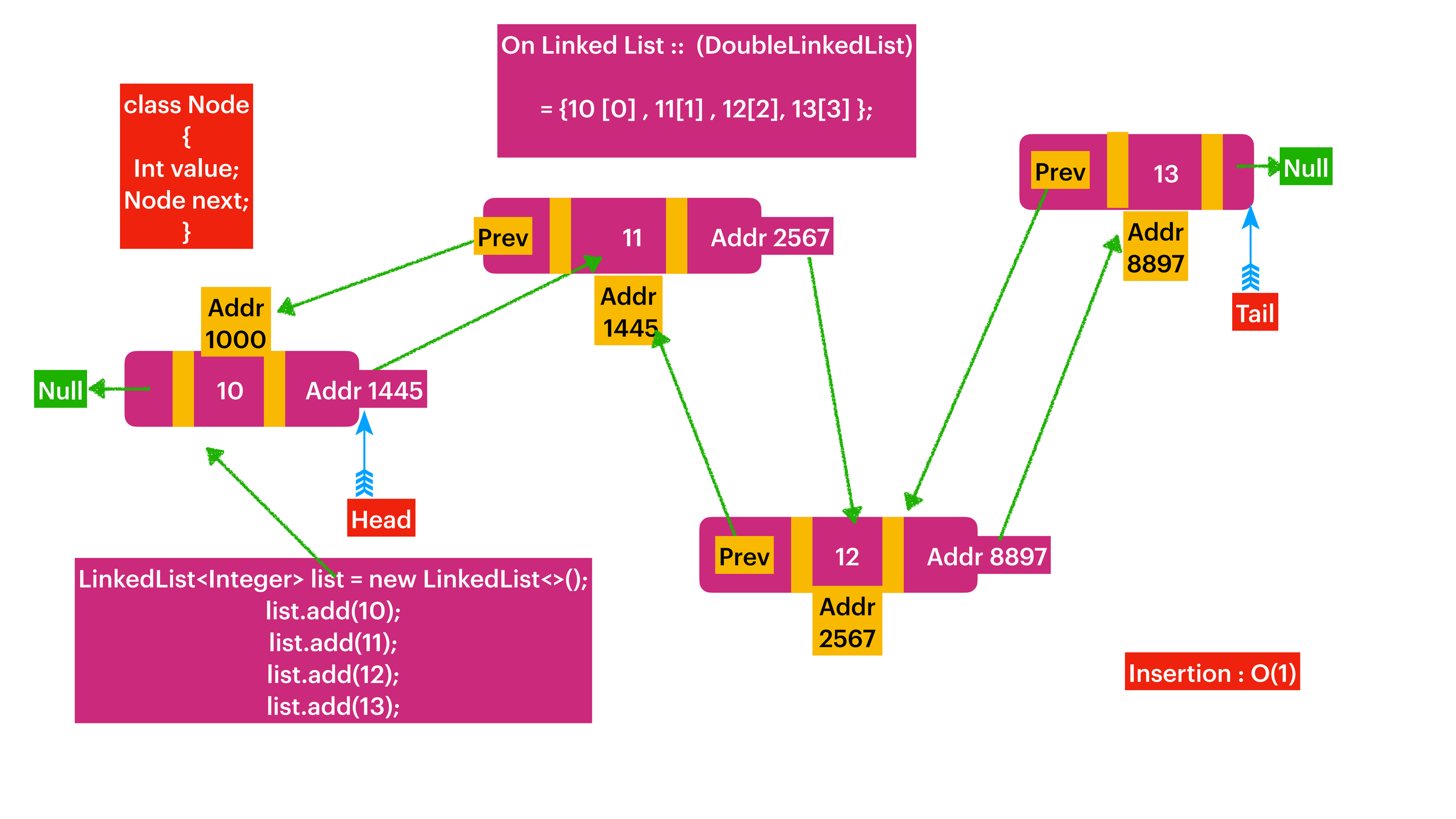
= {10 [0] , 11[1] , 12[2], 13[3] };

Remove Of (12) :
Traverse till 12 O(n)
Remove of 12 O(1)

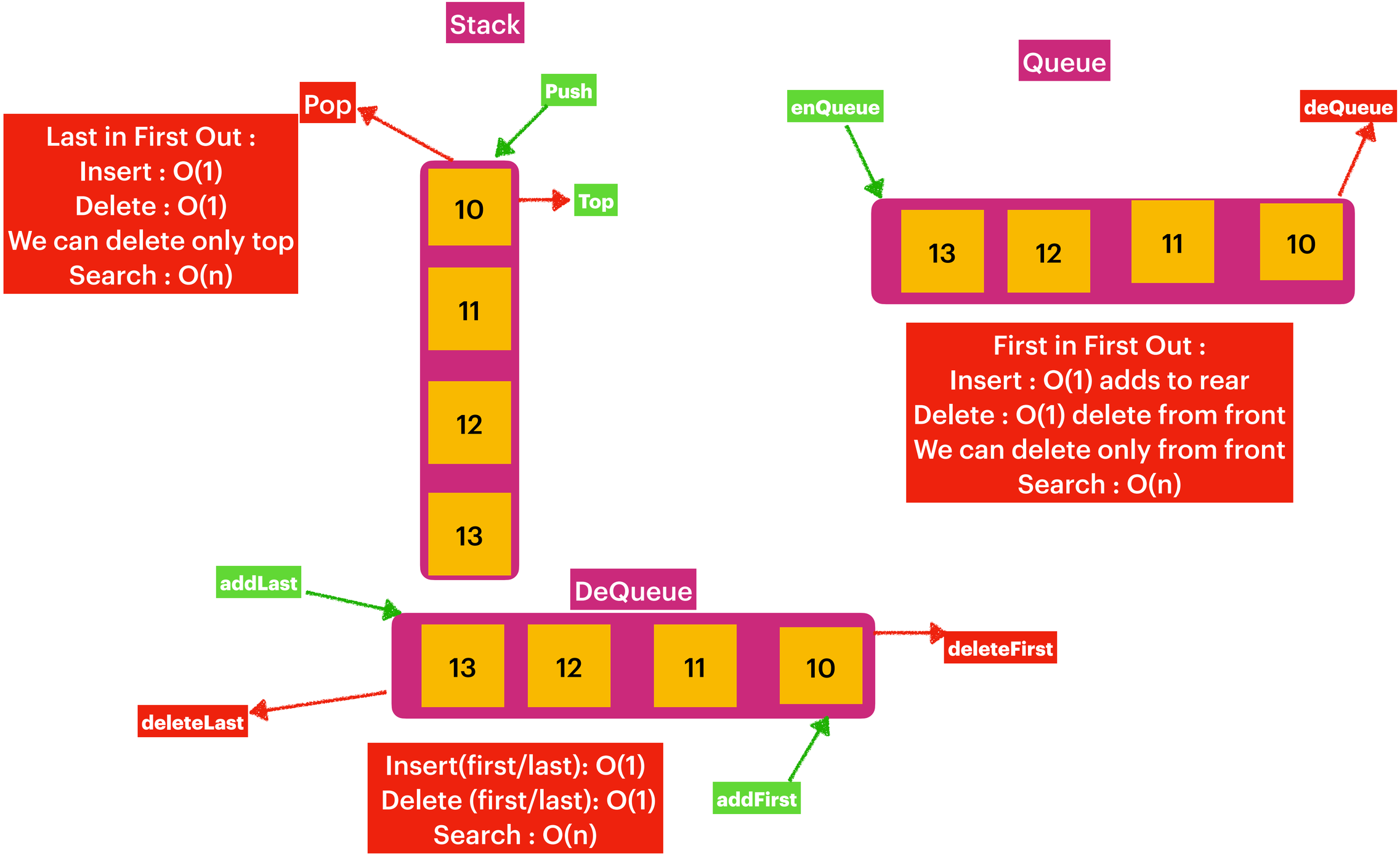



```
LinkedList<Integer> list = new LinkedList<>();
list.add(10);
list.add(11);
list.add(12);
list.add(13);
```

Deletion : O(n) Traverse + O(1) remove = O(n)


On Linked List :: (DoubleLinkedList)
= {10 [0] , 11[1] , 12[2], 13[3] };
class Node
{
Int value;
Node next;
}
Prev
11
Addr 2567
Addr
1445
Addr
1000
Null
10
Addr 1445
Head
Prev
13
Null
Addr
8897
Tail
Prev
12
Addr 8897
Addr
2567
LinkedList<Integer> list = new LinkedList<>();
list.add(10);
list.add(11);
list.add(12);
list.add(13);
Insertion : O(1)

On LinkedList :
Insertion : O(1)
Deletion : O(n) but O(1) shifts
Search : O(n) on index based
Seach : O(n) on value based


Stack
Pop
Push
Top
10
11
12
13
Last in First Out :
Insert : O(1)
Delete : O(1)
We can delete only top
Search : O(n)
Queue
enQueue
deQueue
13
12
11
10
First in First Out :
Insert : O(1) adds to rear
Delete : O(1) delete from front
We can delete only from front
Search : O(n)
addLast
DeQueue
13
12
11
10
deleteFirst
deleteLast
addFirst
Insert(first/last): O(1)
Delete (first/last): O(1)
Search : O(n)

Hashing

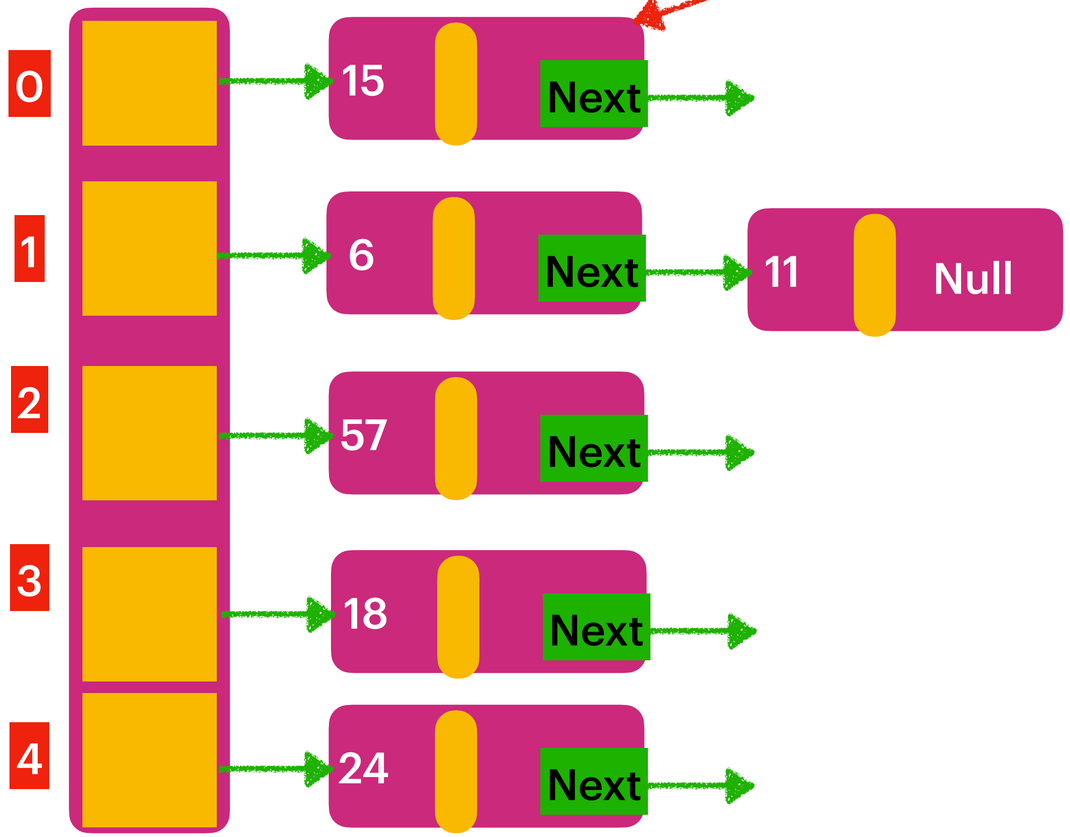
6, 11, 15,1 8, 57, 15, 24

15%5 = 0

15%5 = 0 As key:15 is already exist so no insert.

6

Actual HashCode : 1156
hashCode()
method returns
its own value : 6



search(11) :=> Get HashCode of(11) = 11 => 11%5 = 1

delete(11) :=> Get HashCode of(11) = 11 => 11%5 = 1

Insert : O(1) / log(n)
Update : O(1) / log(n)
Search : O(1) / log(n)
Delete : O(1) / log(n)

6%5 = 1

11%5 = 1

18%5 = 3

57%5 = 2

24%5 = 4

Hashing

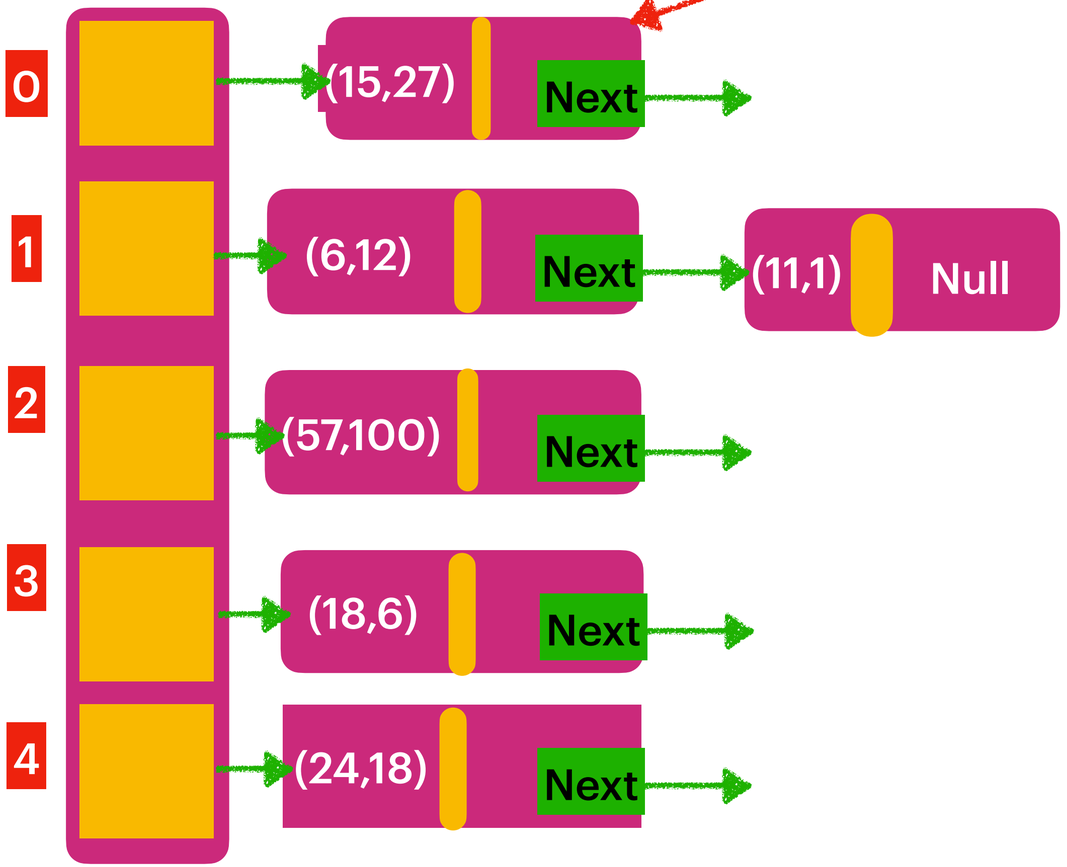
(6,12), (11,1), (15,22), (18,6), (57,100), (15,27) (24,18)

(15,22) = 15%5 = 0

(15,27) -> 15%5 = 0 As key:15 is already exist so no insert but Replaces the Value.

6

Actual HashCode : 1156
hashCode()
method returns
its own value : 6



search(11) :=> Get HashCode of(11) = 11 => 11%5 = 1

delete(11) :=> Get HashCode of(11) = 11 => 11%5 = 1

6%5 = 1

11%5 = 1

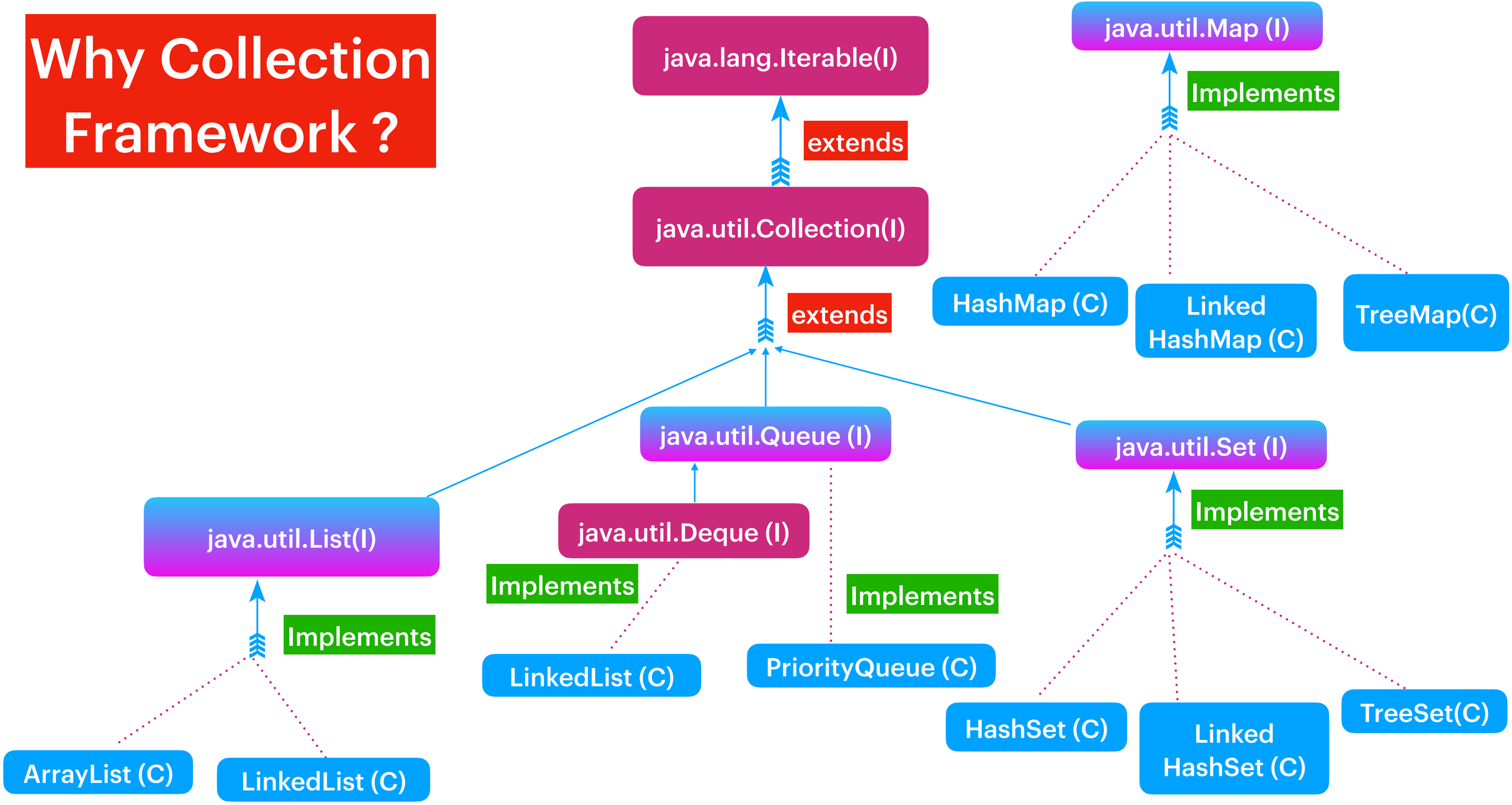
18%5 = 3

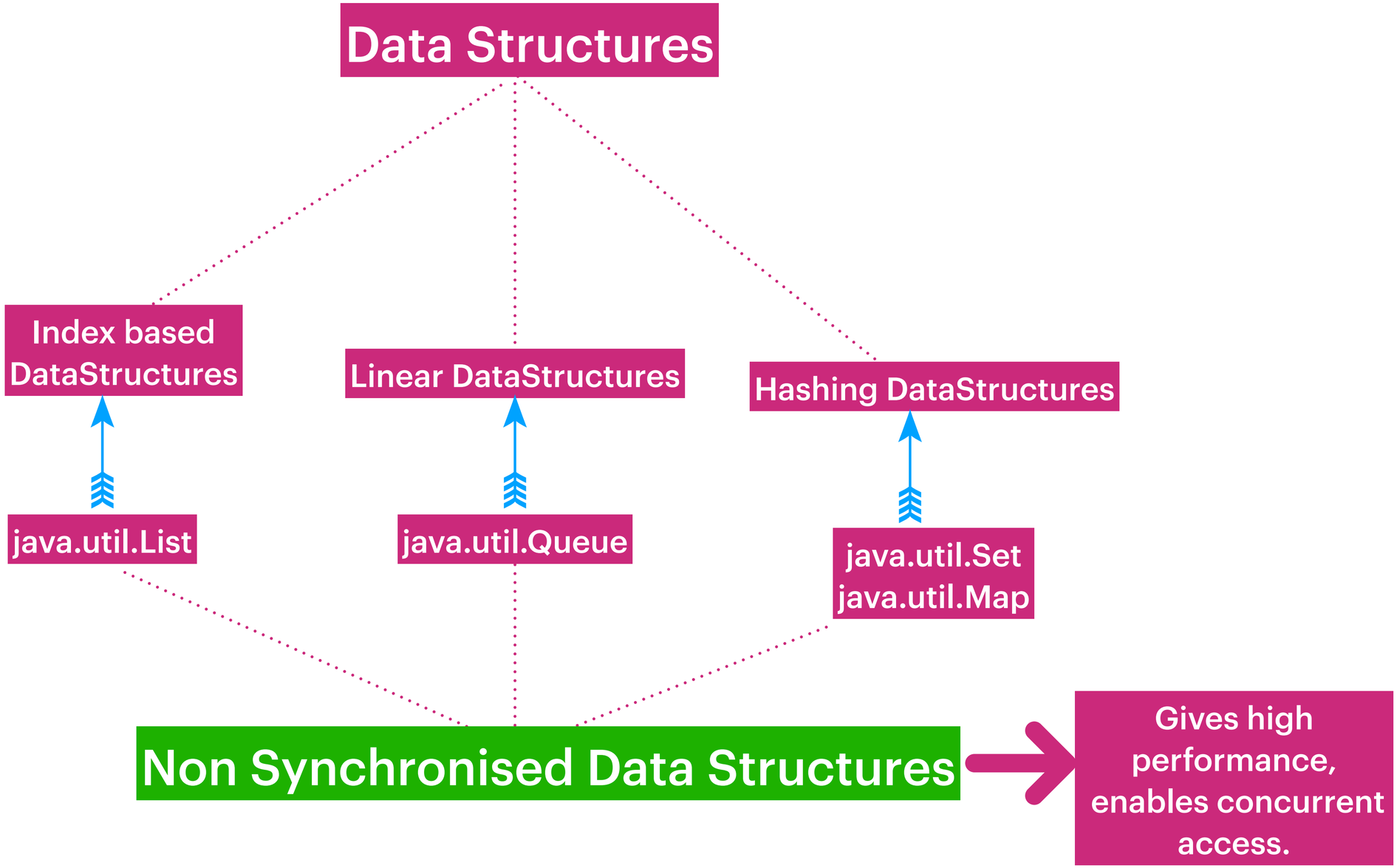
57%5 = 2

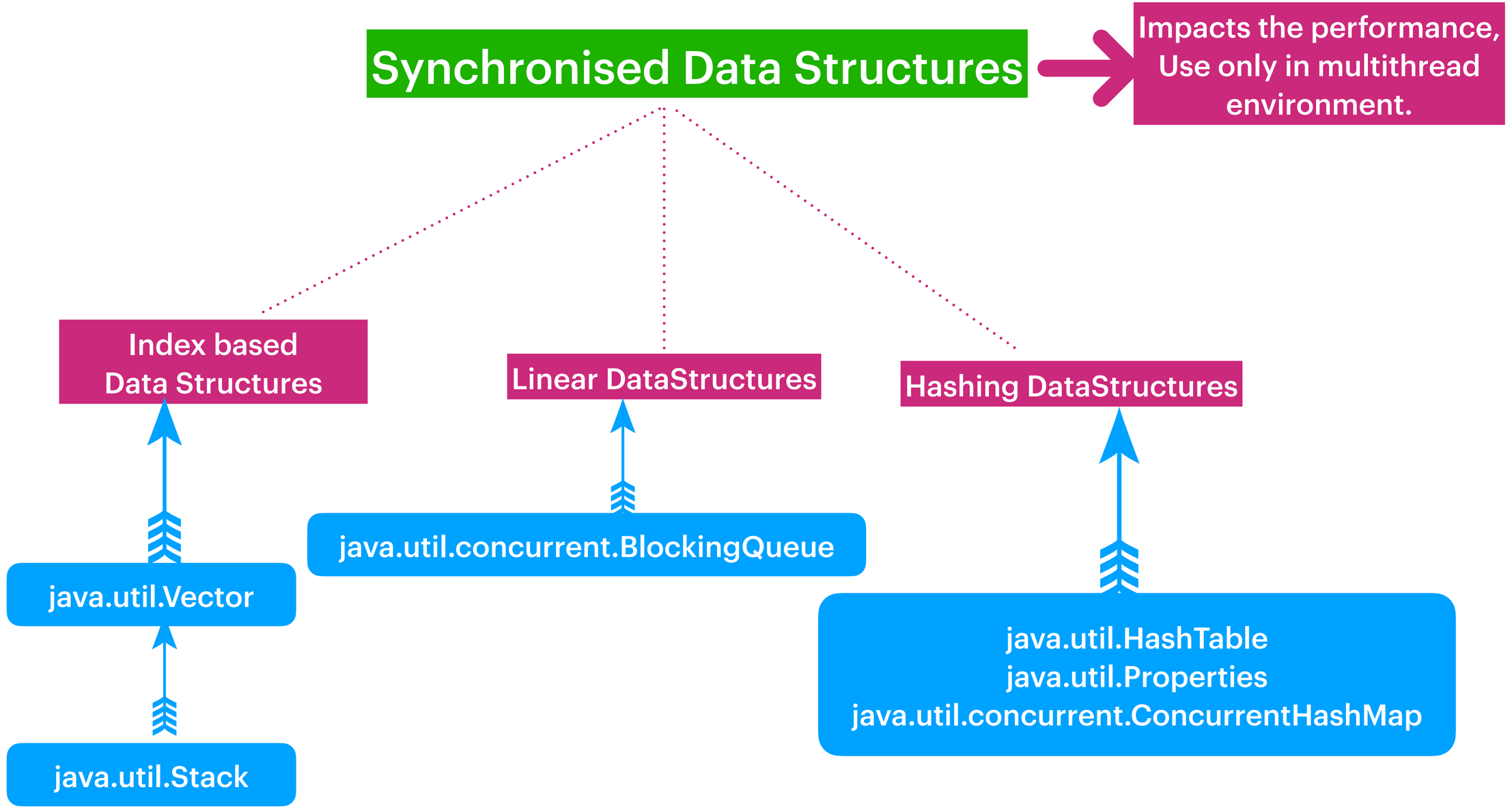
24%5 = 4

Insert : O(1) / log(n)
Update : O(1) / log(n)
Search : O(1) / log(n)
Delete : O(1) / log(n)

Why Collection Framework ?




Data Structures
Index based DataStructures
Linear DataStructures
Hashing DataStructures
java.util.List
java.util.Queue
java.util.Set
java.util.Map
Non Synchronised Data Structures
Gives high performance, enables concurrent access.


Synchronised Data Structures
Impacts the performance, Use only in multithread environment.
Index based Data Structures
Linear DataStructures
Hashing DataStructures
java.util.Vector
java.util.Stack
java.util.concurrent.BlockingQueue
java.util.HashTable
java.util.Properties
java.util.concurrent.ConcurrentHashMap