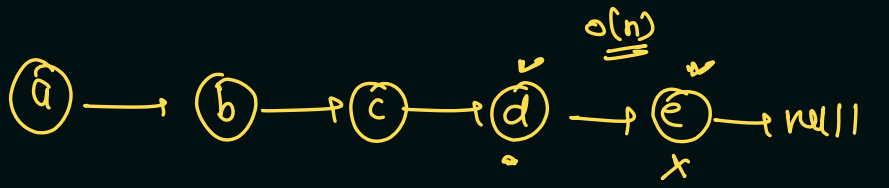
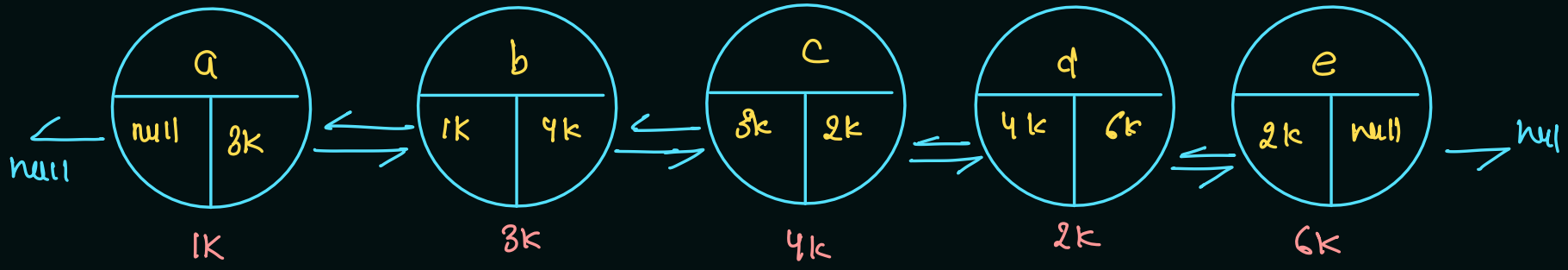
Doubly Linked list creation:



1. add First
2. add Last
3. add At
4. get First
5. get Last
6. get At
7. Remove First
8. remove Last
9. remove At
10. Size
11. display

Node → 1. val
2. next
3. prev

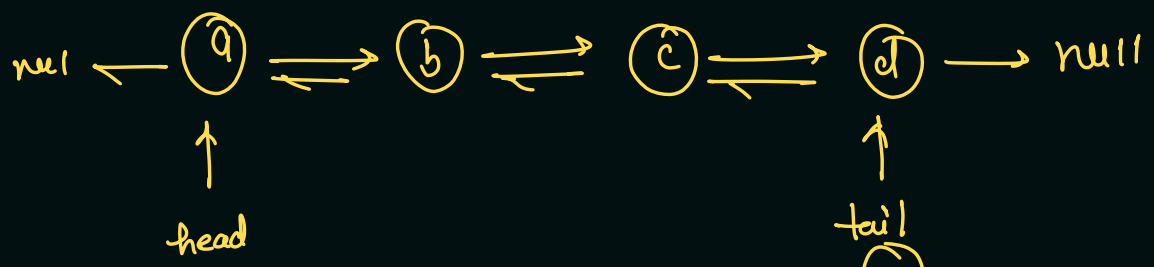


Val

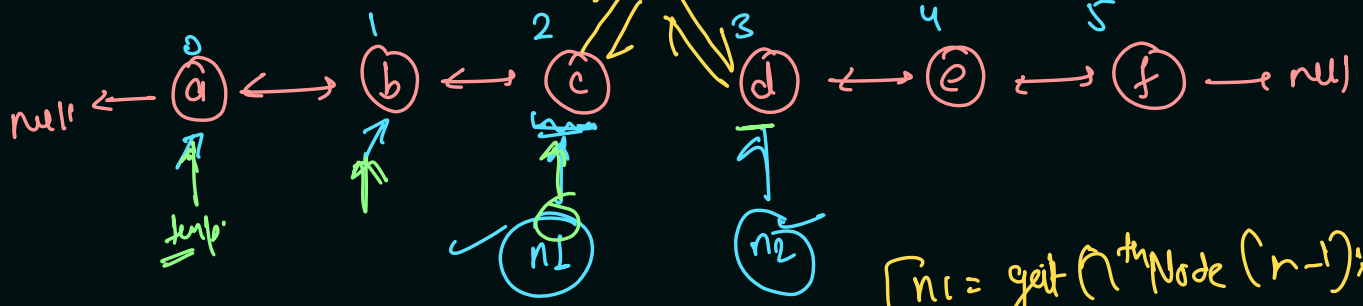
prev ← → next

display

null ← a ↔ b ↔ c ↔ d ↔ e → null



size = 4



indx=2

Add At

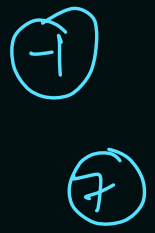
n2 = n1.next

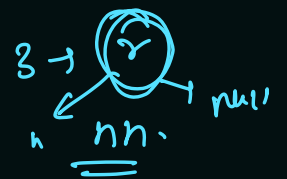
n1 = (n-1)th node ,

n2 = nth node

```
n1 = getNthNode(n-1);
n2 = n1.next
```

```
n1.next = nn
nn.prev = n1
nn.next = n2
n2.prev = nn.
this.size++;
```

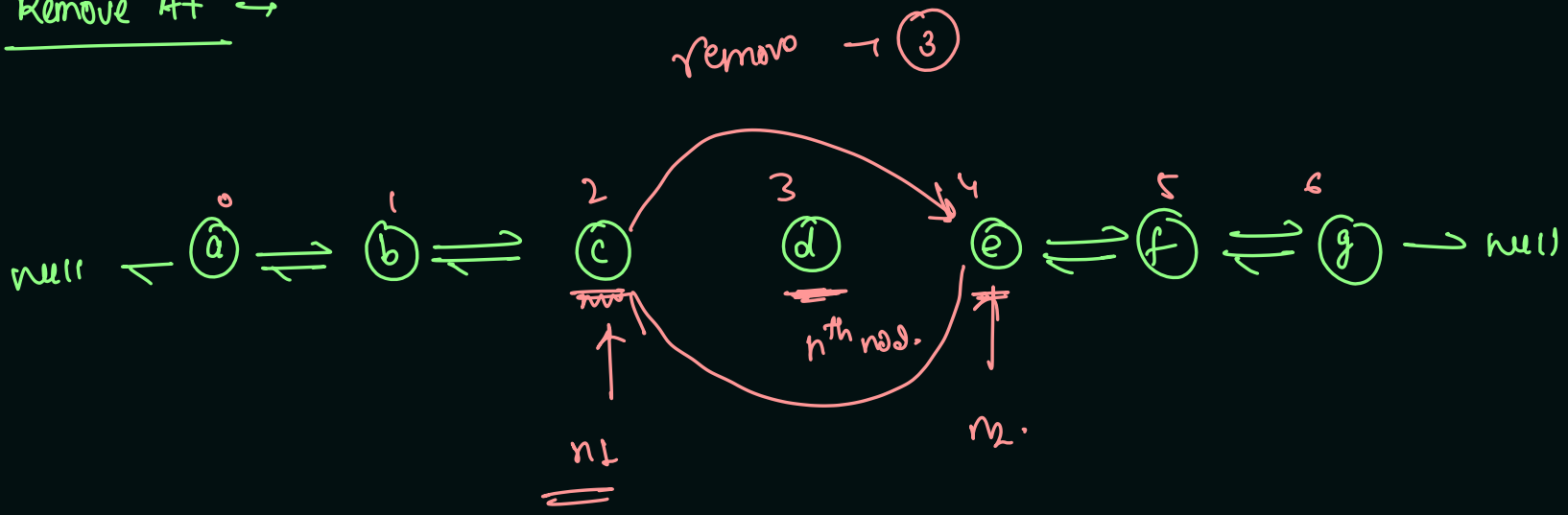





```
private ListNode getNthNode(int indx) {
    ListNode temp = this.head;
    for(int i = 0; i < indx; i++) {
        temp = temp.next;
    }
    return temp;
}
```

i=0 1 2

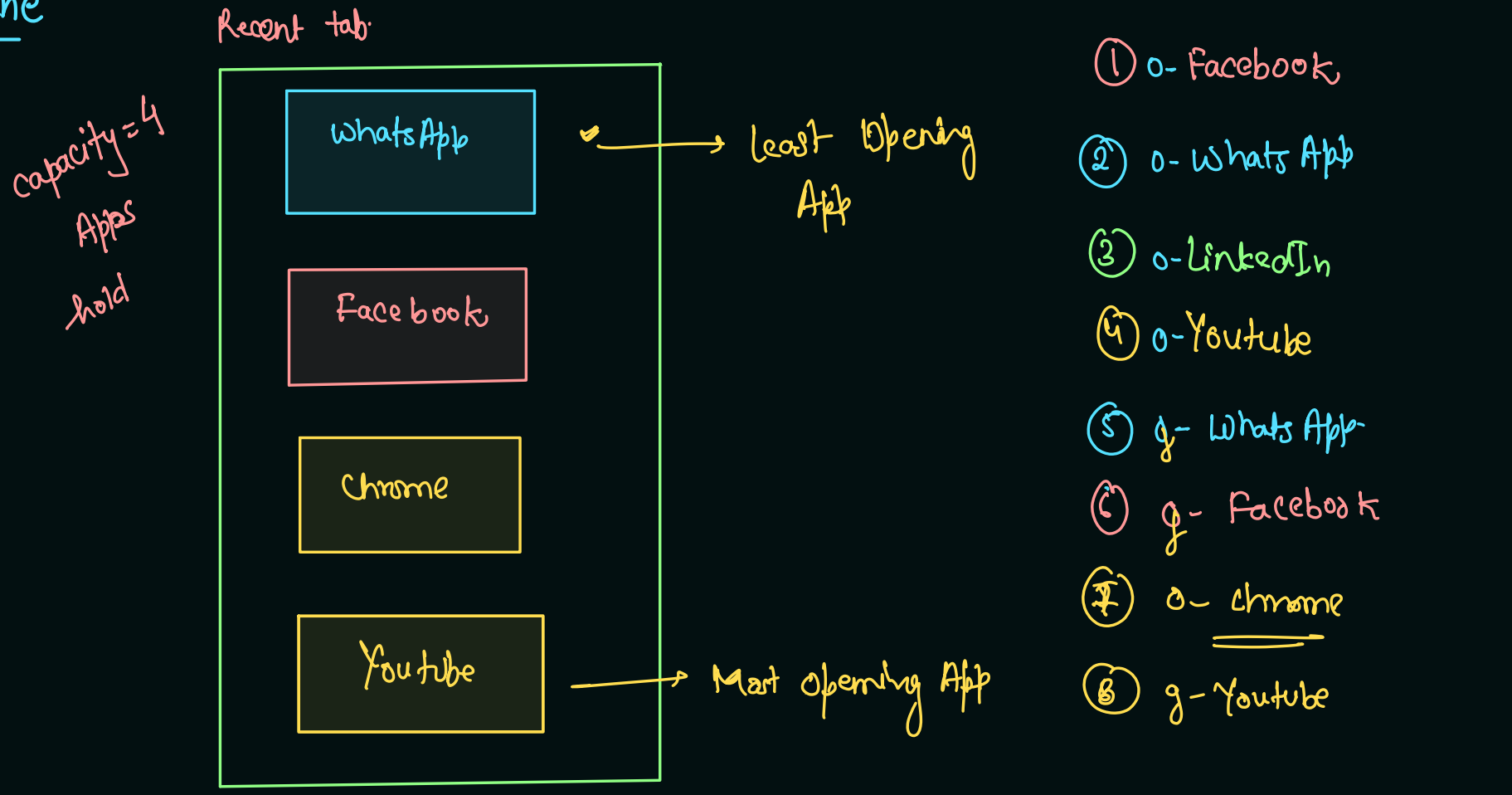
Remove At →




```
n1 = getNthNode(3-1);   // pos-1
n  = n1.next;
n2 = n.next
```

```
n1.next = n2;
n2.prev = n1;
n.next = n.prev = null
```

LRU Cache



① o- Facebook
② o- WhatsApp
③ o- LinkedIn
④ o- Youtube
⑤ g- WhatsApp
⑥ g- Facebook
⑦ o- Chrome
⑧ g- Youtube

Management of Recent tabs is an example of LRU cache.

LRU → Least Recently used.

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

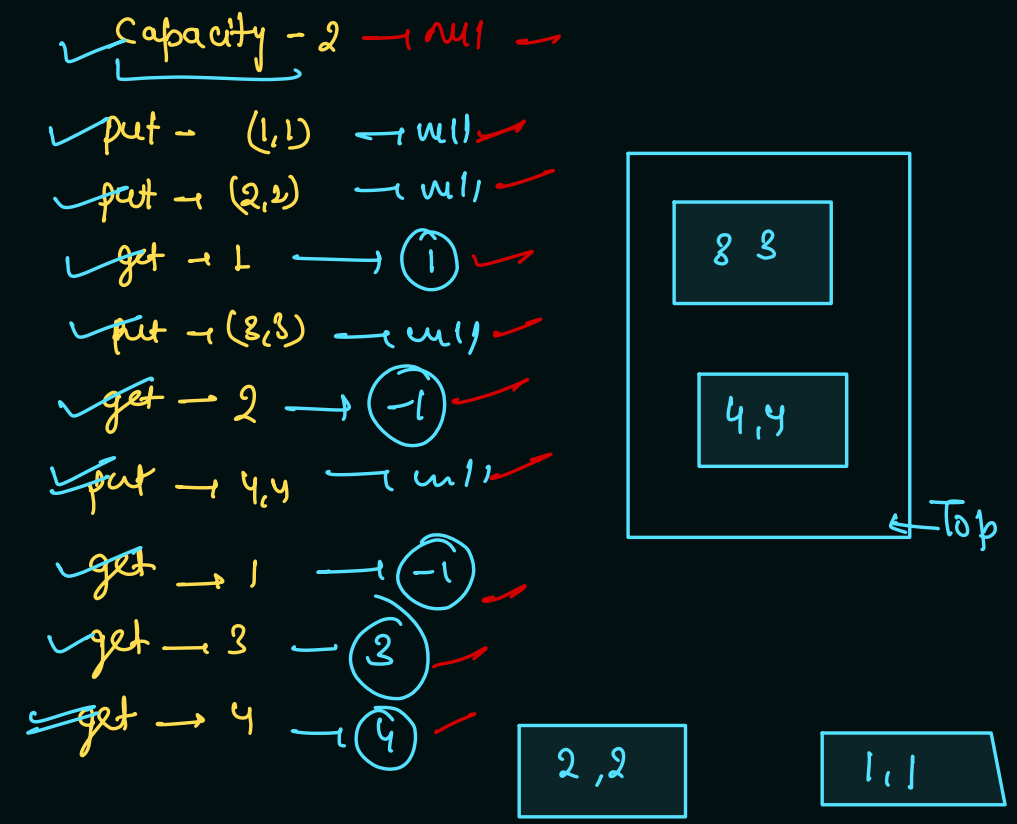
Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Example 1:

```
Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]
```



LRU

App is associated with key

opening of a new App

capacity.

get . put

↳ average time → O(1)

Doubly linked list → Add First, Remove First, Add Last, Remove Last → O(1)

key-value

HashMap

LRU

HashMap → Key.value

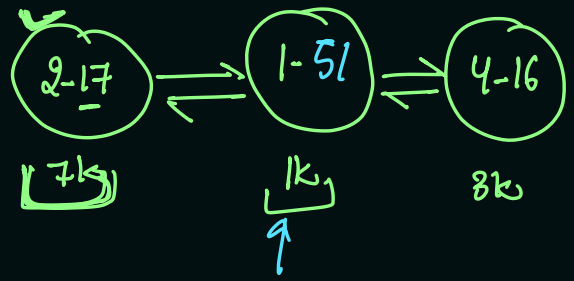
(int - LL.Node)

(Key - Node)

key	Node
1	1K
2	7K
4	3K

Capacity → 3 LRU Managemnt

Doubly Linked List



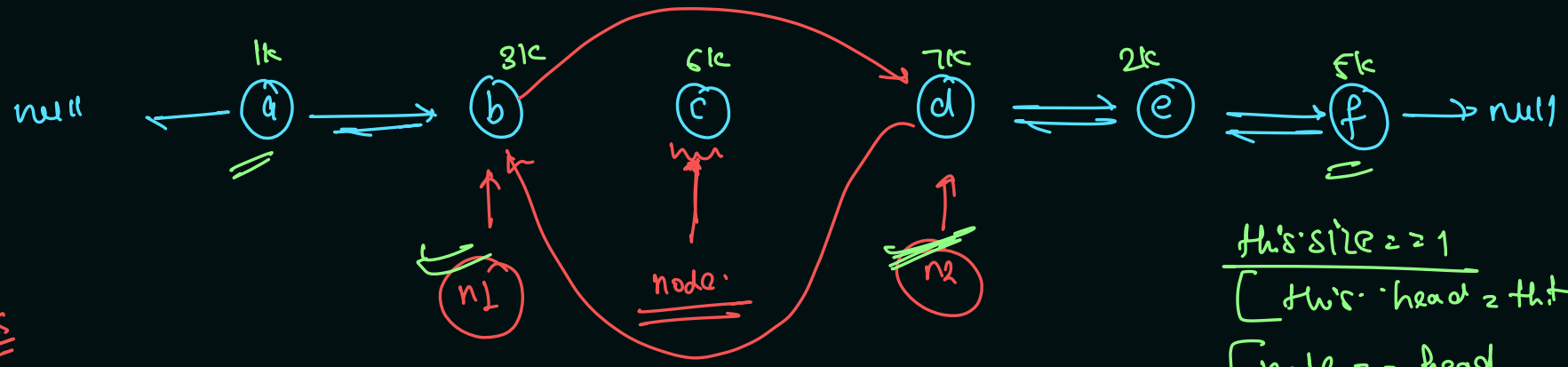

```
node = map.get(key) → 7K
removeNode(node) → O(1)
addLast(node)
```

Node → key, value

LRU cache → 3 (cap-)

	K	Value
put	1	23
put	2	17
put	3	19
get	2 →	17
get	1 →	23
put	1	51
put	4	16 →
get	3 →	-1
put	5	28
put	7	81
get	1	
get	3	
put	9	27
get	2	

new App → store in last of linkedlist → convention



6k

this.size == 1

this.head = this.tail = null

node == head

head = head.next

head.prev = null

node == tail

tail = tail.prev

tail.next = null

else

n1.next = n2

n2.prev = n1

node.next = node.prev = null

this.size --

n1.next = n1.next.next