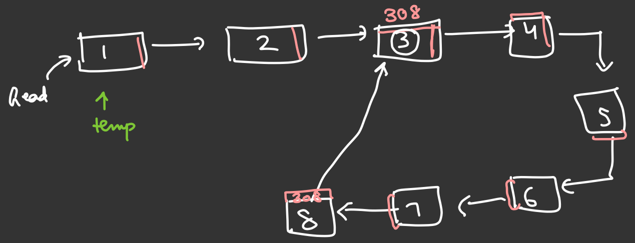
# "LINKED LIST - 3"

- cycle detection
- doubly linked list
- circular linked list
- LRU cache

## (1) Cycle Detection → Detect if the linked list contains a cycle



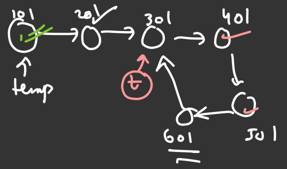
Iterate ⇒

```
while (temp != null)
        temp. = temp.next
```

] ∞ loop

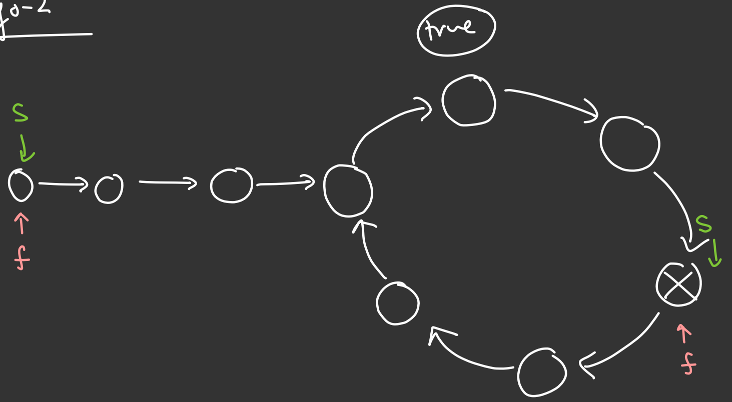
```
Algo-1      bool   detectCycle ( Node head) {
                Hashset <Node >      h ;

         Node  temp  =  head,

                while (temp != null) {
                      if (hs. contains (temp)) {
                              return true,
                      }
                      hs insert (temp);

                      temp = temp.next,

                }

                return  false,

         }
```

{ O(N) Space
  O(N) time.

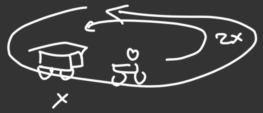


101 ✓
201 ✓
301 ✓
401 ✓
501 ✓
601 ✓

# Algo-2

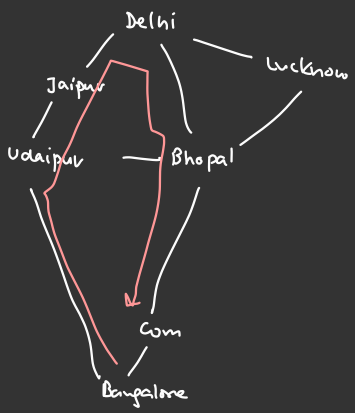


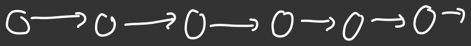
if they meet → loop is there



```
while( —— ){
    fast = fast.next next
    slow = slow next
}
```

## Graphs



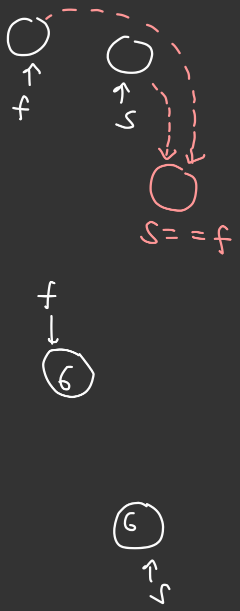


false

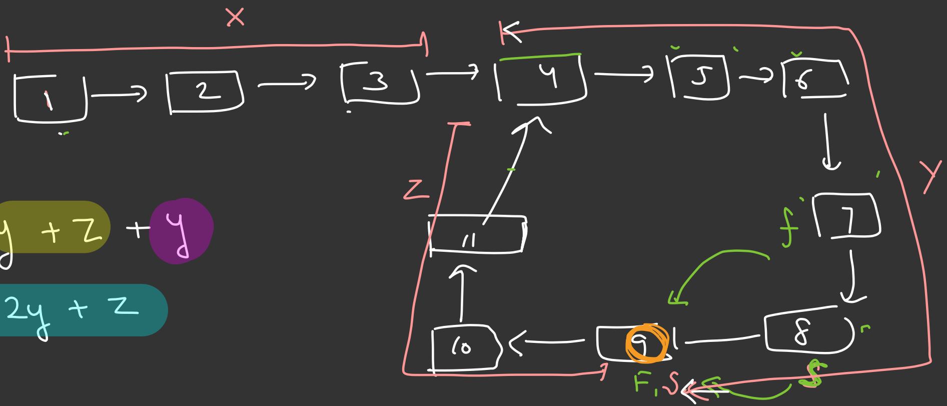
Space → O(1)
Time → O(N)

```
boolean    detectCycle( Node head){
    Node show = head
    Node fast  = head

    while( fast!=null && fast.next !=null ){
        => fast = fast.next.next
        => slow = slow next
        => if( slow == fast) { return T }
    }

    return F ;
}
```

MP

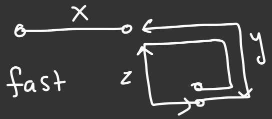


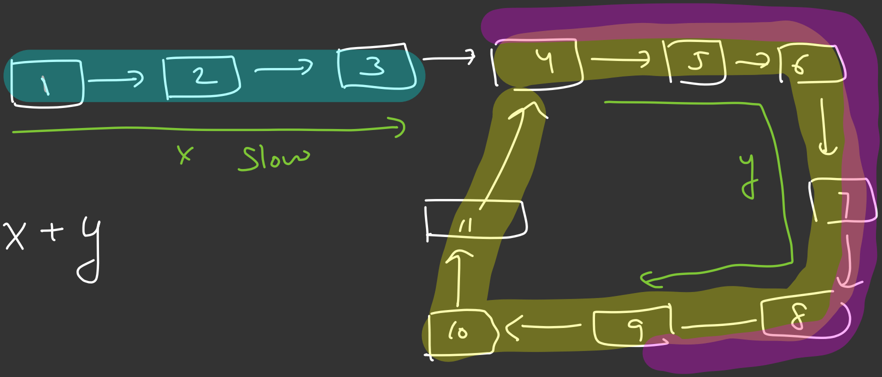
## Q-2 Break the cycle / find the node leading to cycle



① Dist Fast $= x + y + z + y$

$= x + 2y + z$





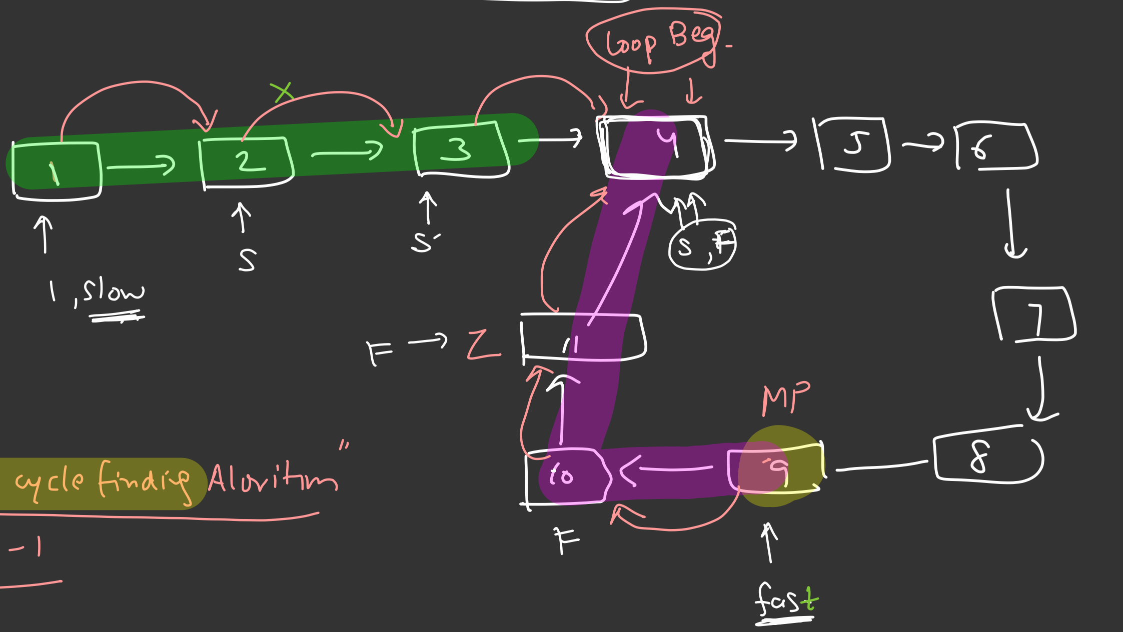
② Dist Slow $= x + y$

$$D_{Fast} = 2 * D_{slow}$$

⇒ $x + 2y + z = 2(x+y)$

⇒ $x + \cancel{2y} + z = 2x + \cancel{2y}$

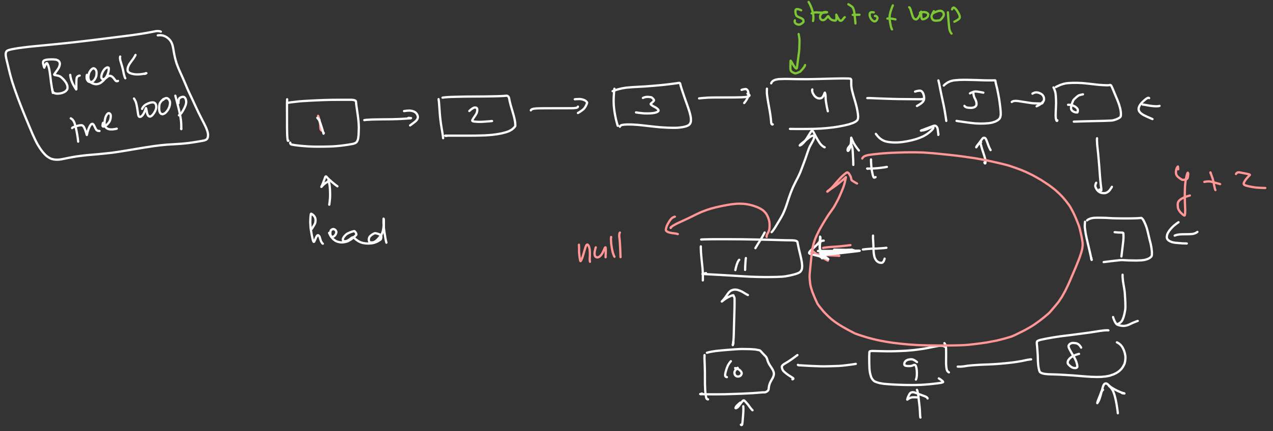
⇒ $\boxed{z = x}$ Result ⇒ significance



"Floyd's cycle finding Alorithm"

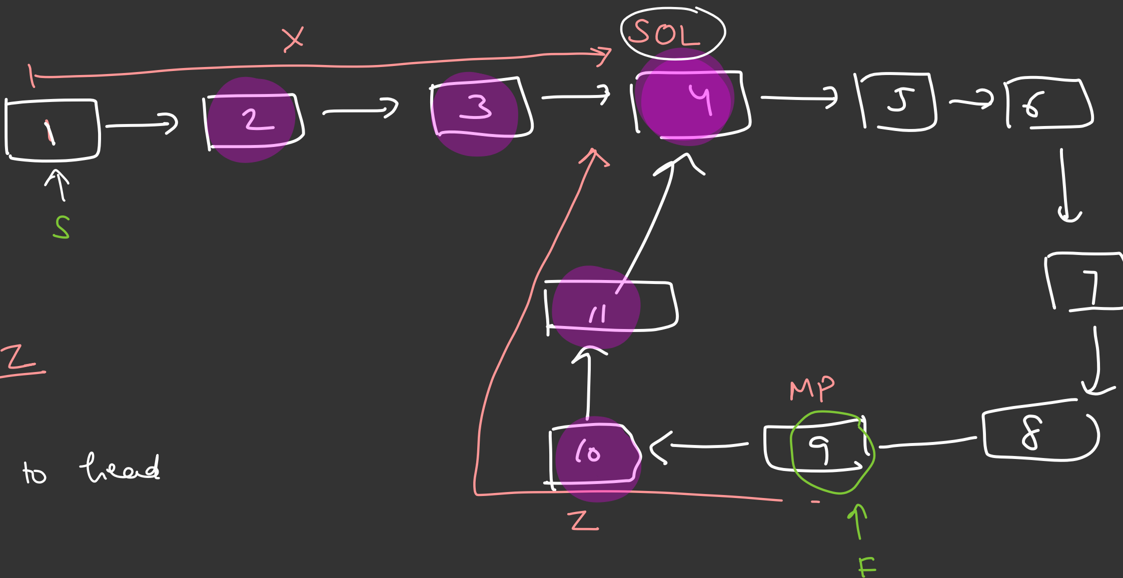
Algo - 1

```
Node   get StartOF Loop(       ) {
    fast =  ]
            ]← = MP from cycle detection code
    slow =  ]

    slow = head

    while ( Slow != fast) {

        slow = Slow.next
        fast = fast.next

    }

    return Slow      // Beginning of Loop
}
```



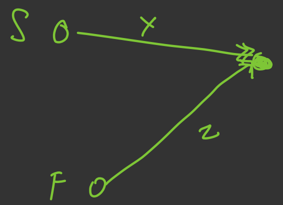
```
Node temp = StartOFloop();

while( temp.next != StartOFloop {

        temp = temp next,

}

temp.next =null
```

= O(N)

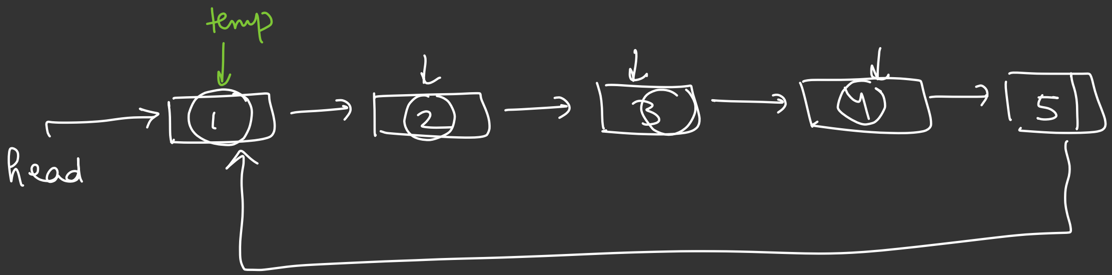


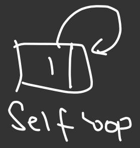
Proved x=Z

Reset S to head



# "Circular linked List"





/

tail.next = head, [Construction]

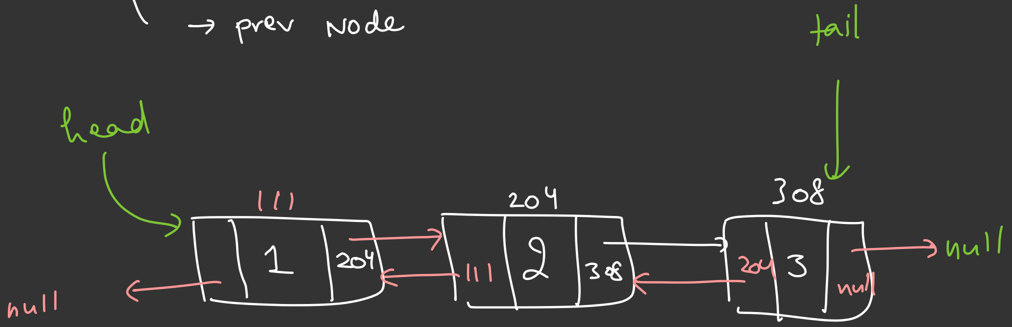
```
printCL ( Node  head){
        temp = head .
        while ( temp.next != head)(
                print (temp.data)
                temp = temp.next
        }
        print (temp.data);
```

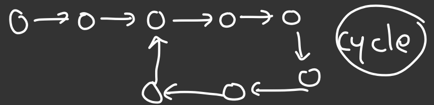
1, 2, 3, 4, $\textcircled{5}$

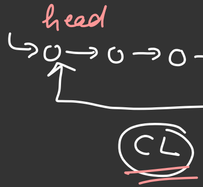
3

# "Doubly linked list"

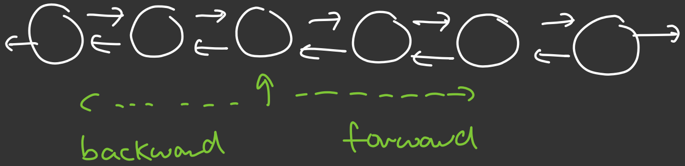
{ → next Node
→ prev Node







```
class Node {
    int data;
    Node next;
    Node prev;
}
```

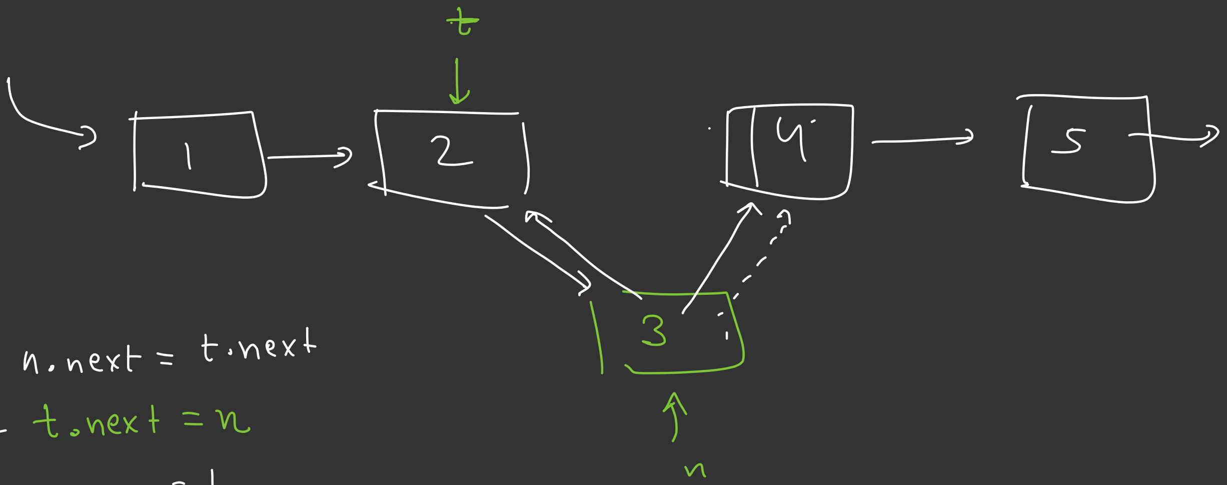


Disadv

↳ more memory

Adv

↳ Traversal in both direction



n.next = t.next
t.next = n

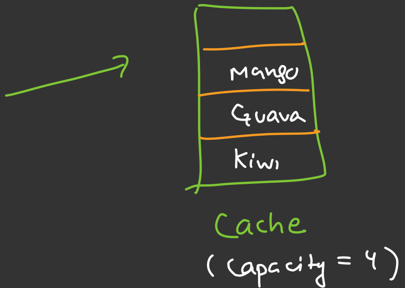
n.prev = t
n.next.prev = n

# LRU Cache

Theory – Slides ✓

10.40

## Discuss.



Apple –

insert ( key , value )

② Cache is full

"Banana"

① Delete last Node O(1)

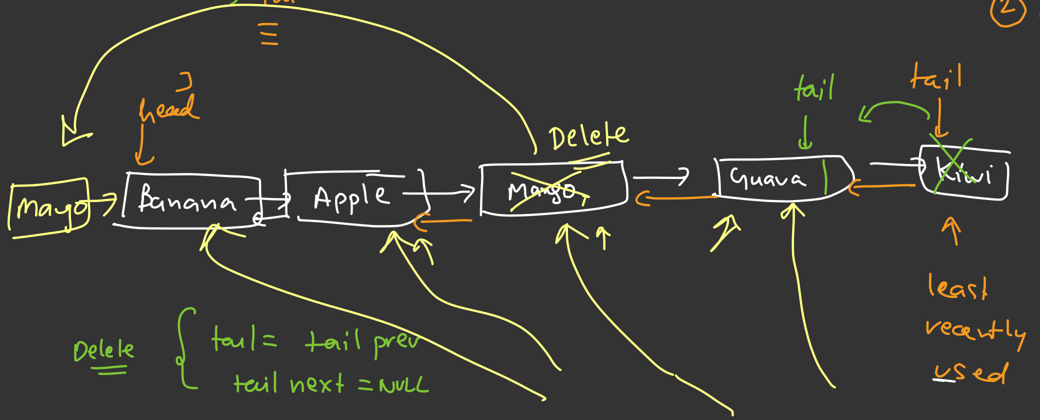
② Insert new Node

① Not full

insert at beginning

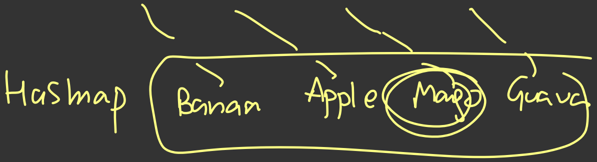
"Apple"

Doubly Linked List {
→ head
→ tail
≡



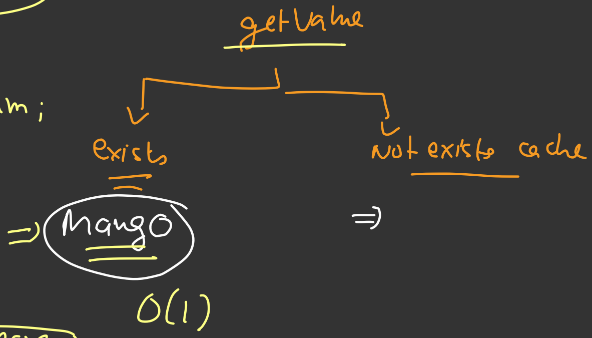
Delete { tail = tail prev
tail next = NULL

at Head

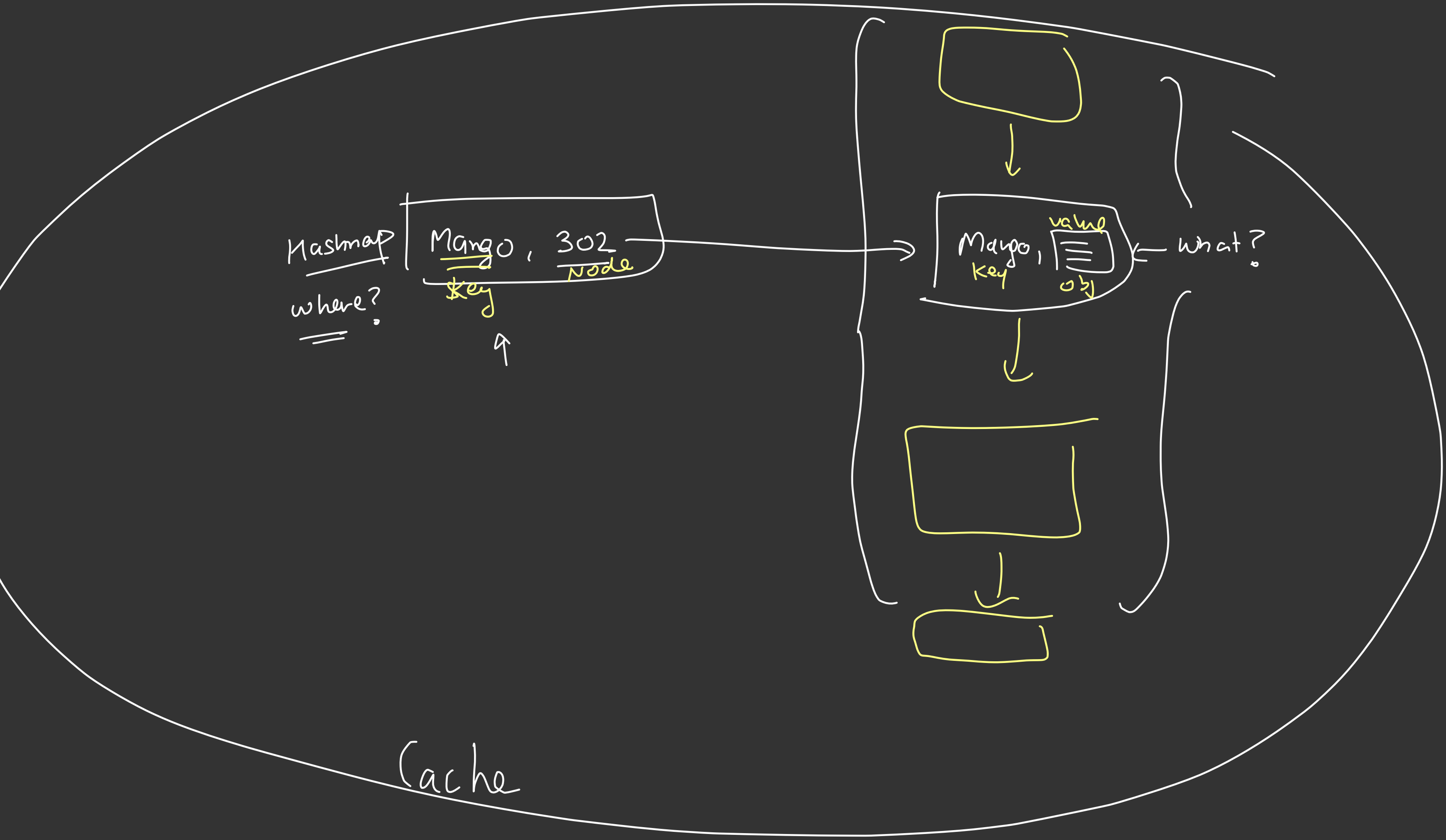


```
class Node {
    ≡

}

class Linked List {
    Node head;
    Node tail

    insertAt tail() {
        tail → ≡
    }
}
```

Hashmap<Strig, Node > hm;



query → $\boxed{\text{Mango}}$


Hashmap
Mango , 302
key
Node
where?
Mango,
key
value
obj
what?
Cache

LRU :)