# LRU Cache - Interview Notes

## Why use HashMap + Doubly Linked List?

----------------------------------------

To implement an LRU (Least Recently Used) Cache, the goal is to achieve:

- O(1) time for get(key)
- O(1) time for put(key, value)

This is done using:

- HashMap<Integer, Node>: for O(1) access by key
- Doubly Linked List: to maintain usage order (MRU at front, LRU at end)

## Why NOT use other data structures?

--------------------------------------

### Singly Linked List:

- Cannot delete a node in O(1) without previous pointer
- Removing LRU takes O(n)

### Stack:

- Only supports LIFO (Last In First Out)
- Cannot access or remove least recently used item efficiently

### Circular Linked List:

- Complicated pointer logic, especially when list has 1 node
- No performance benefit over standard DLL
- More prone to bugs

Queue or Deque (alone):

- Cannot remove an arbitrary node in O(1)
- Needs to be combined with HashMap anyway

Why Dummy Head and Tail in DLL?

---------------------------------

- Avoid null checks at head/tail
- Consistent O(1) insert/delete at both ends

Time Complexity:

-------------------

- get(): O(1) => hashmap lookup + remove + insert at head
- put(): O(1) => hashmap operations + insert/remove in DLL

Interview-Ready Answer:

---------------------------

"To implement LRU Cache in O(1) time for both get and put, we use a combination of HashMap and Doubly Linked List. The HashMap provides fast key lookup, and the doubly linked list allows fast insertion and removal of nodes from any position - especially from the front (most recently used) and back (least recently used). Alternatives like singly linked lists or stacks don't support O(1) deletion from the middle or tail, and circular lists add unnecessary complexity. That's why this combo is the most optimal and widely used in production-grade systems."