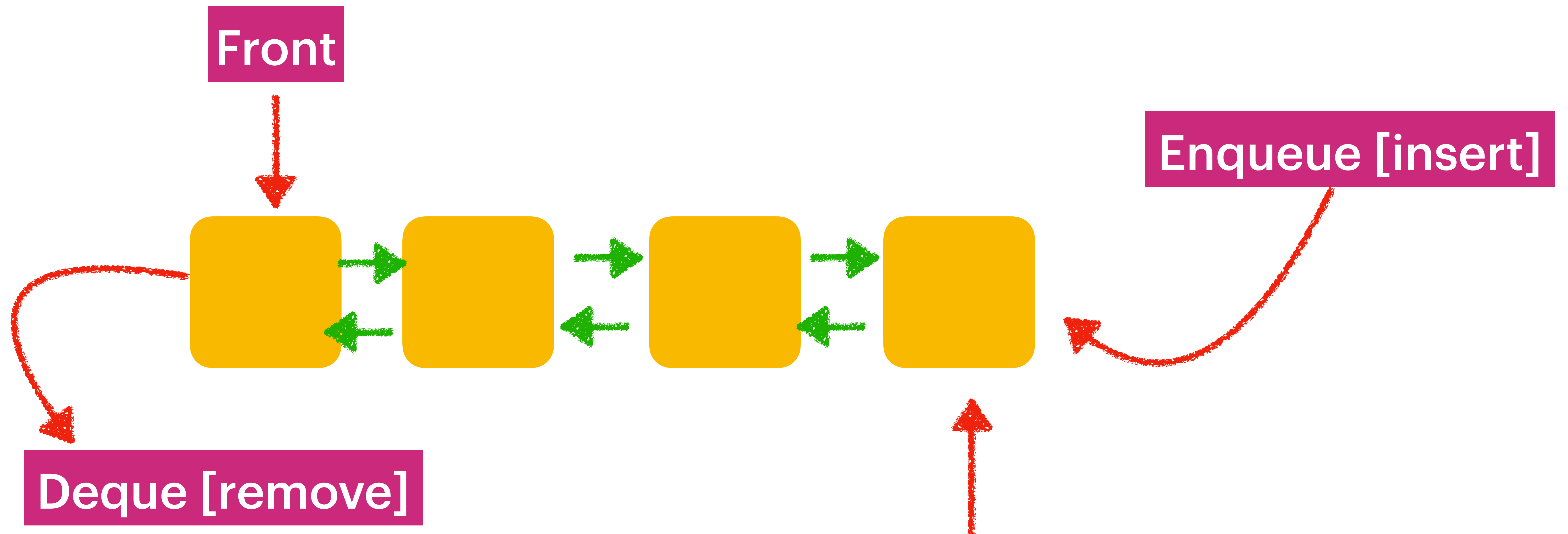
Design Queue



Queue :

- First in First Out
- Allow duplicates

Implementation class of Queue in Java is LinkedList

$$\begin{tikzcd}\text{java.util.Queue} \\ \text{java.util.LinkedList} \arrow[u]\end{tikzcd}$$

Design Queue with LinkedList

public boolean enQueue(int element) : O(1)

public boolean deQueue(int element) : O(1)

public in size() : O(1)

public boolean search(int element): O(n)

public int front(); O(1)

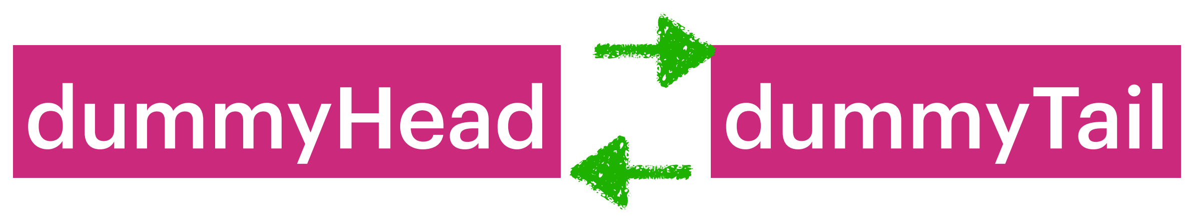
public int rear(): O(1)

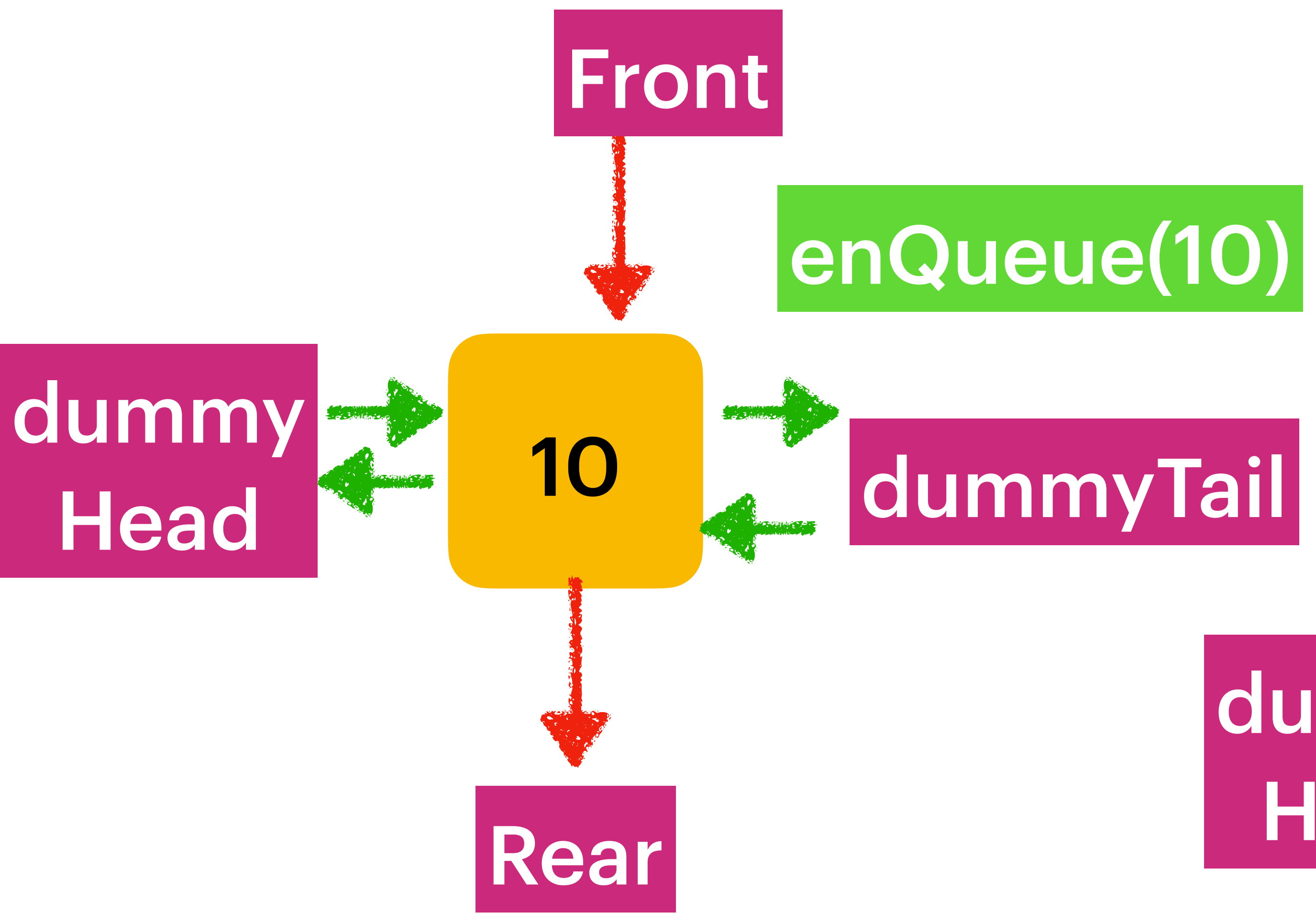
java.util.Queue

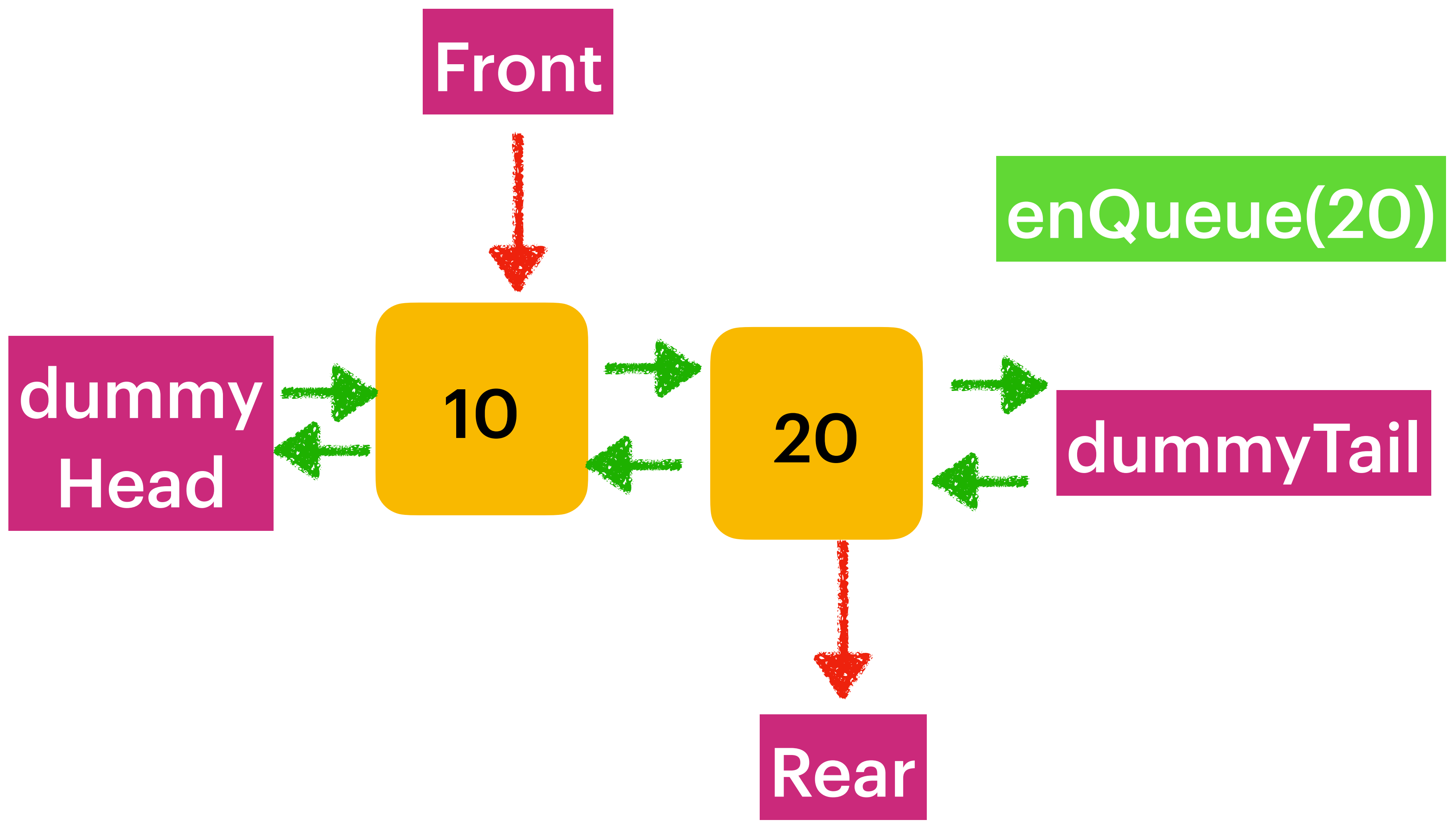
Add element to Rear
public abstract boolean add(E);
public abstract boolean offer(E);

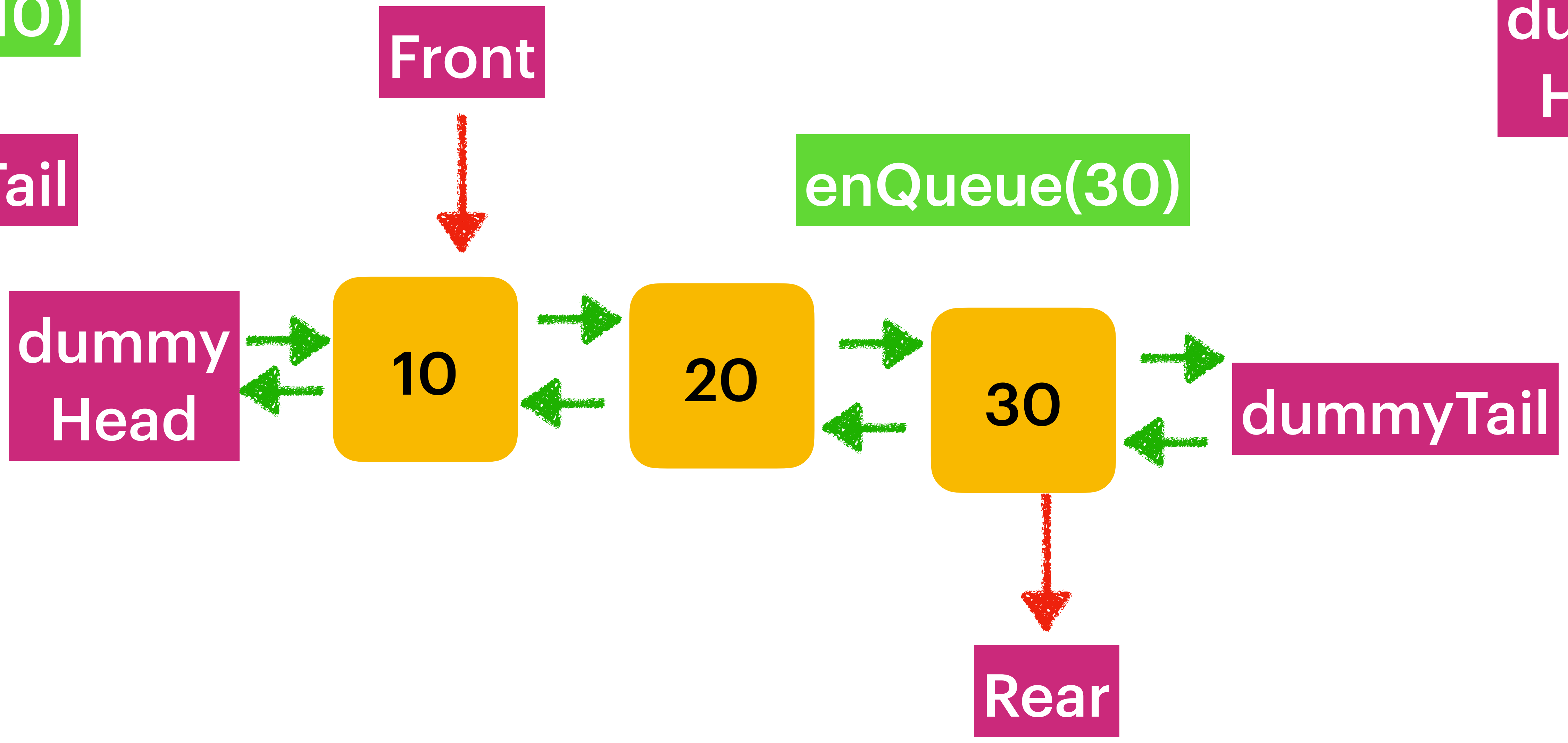
Removes the Front Element
public abstract E remove();
public abstract E poll();

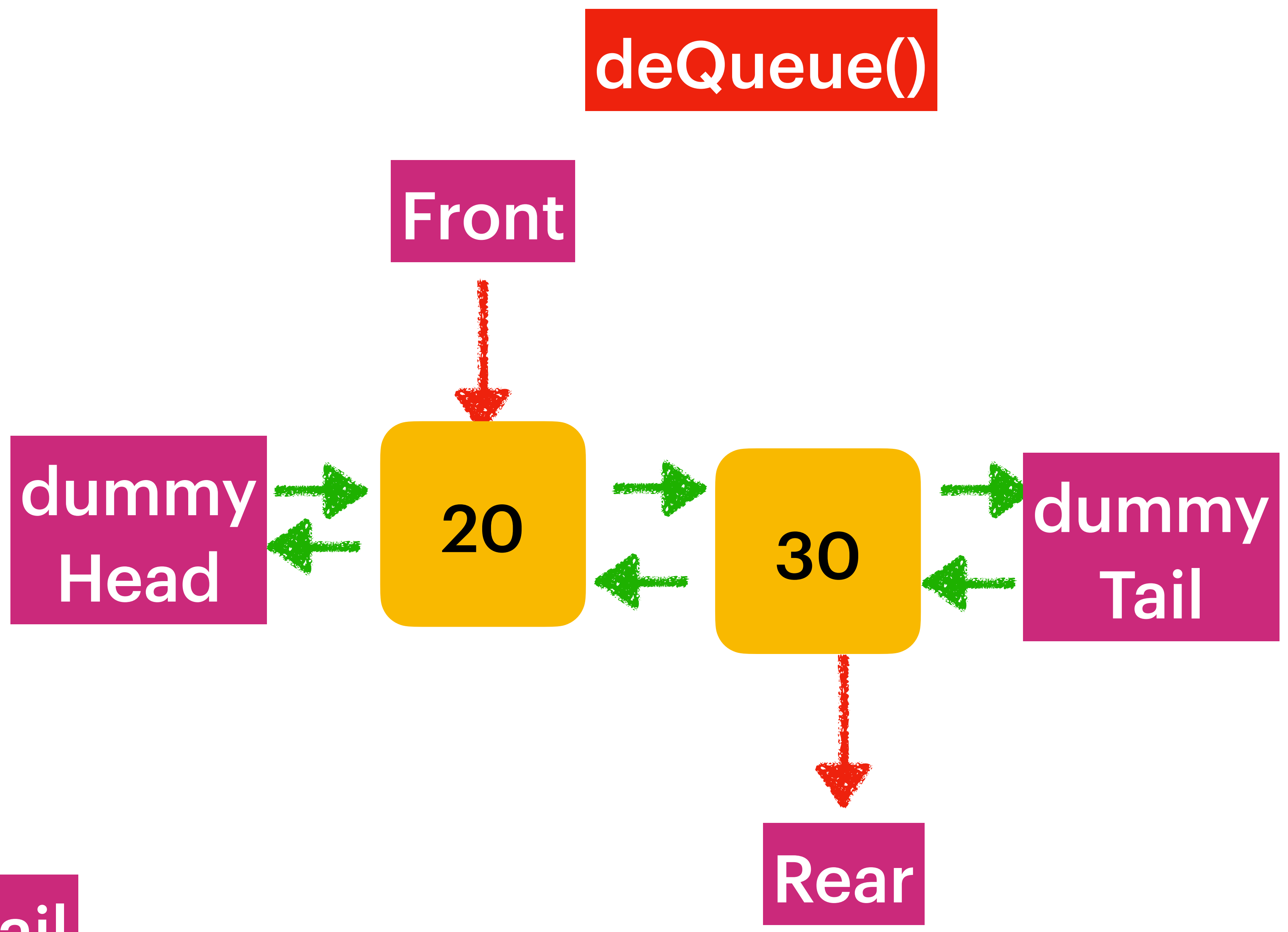
Returns the Front
public abstract E element();
public abstract E peek();

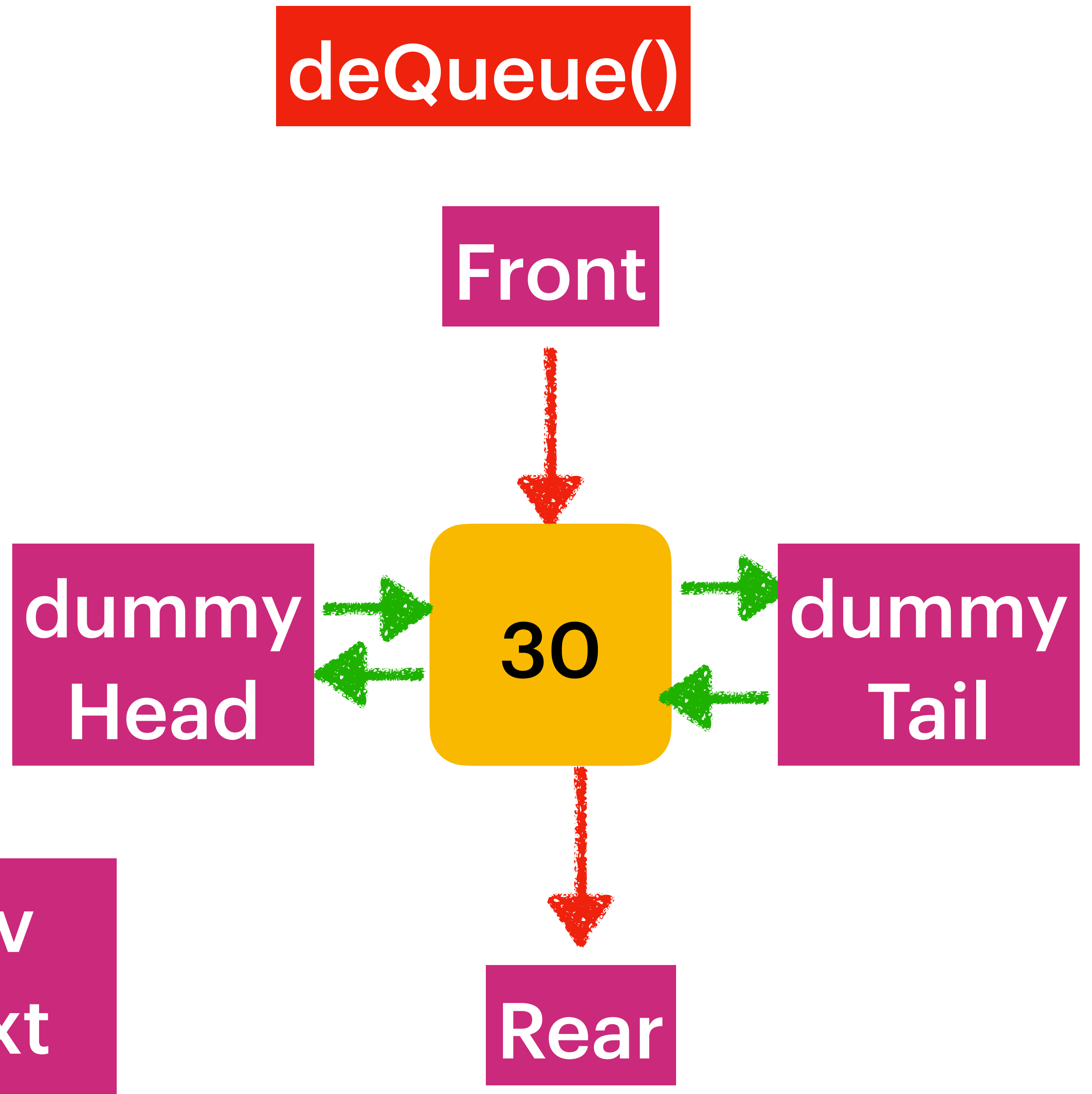












enQueue(int) —> Add tp dummyTail.prev
deQueue() —> remover dummyHead.next
Base check dummyHead.next == dummyTail
then queue is Empty.

346. Moving Average from Data Stream

Easy 1333 124 Add to List Share

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

Implement the `MovingAverage` class:

- `MovingAverage(int size)` Initializes the object with the size of the window `size`.
- `double next(int val)` Returns the moving average of the last `size` values of the stream.

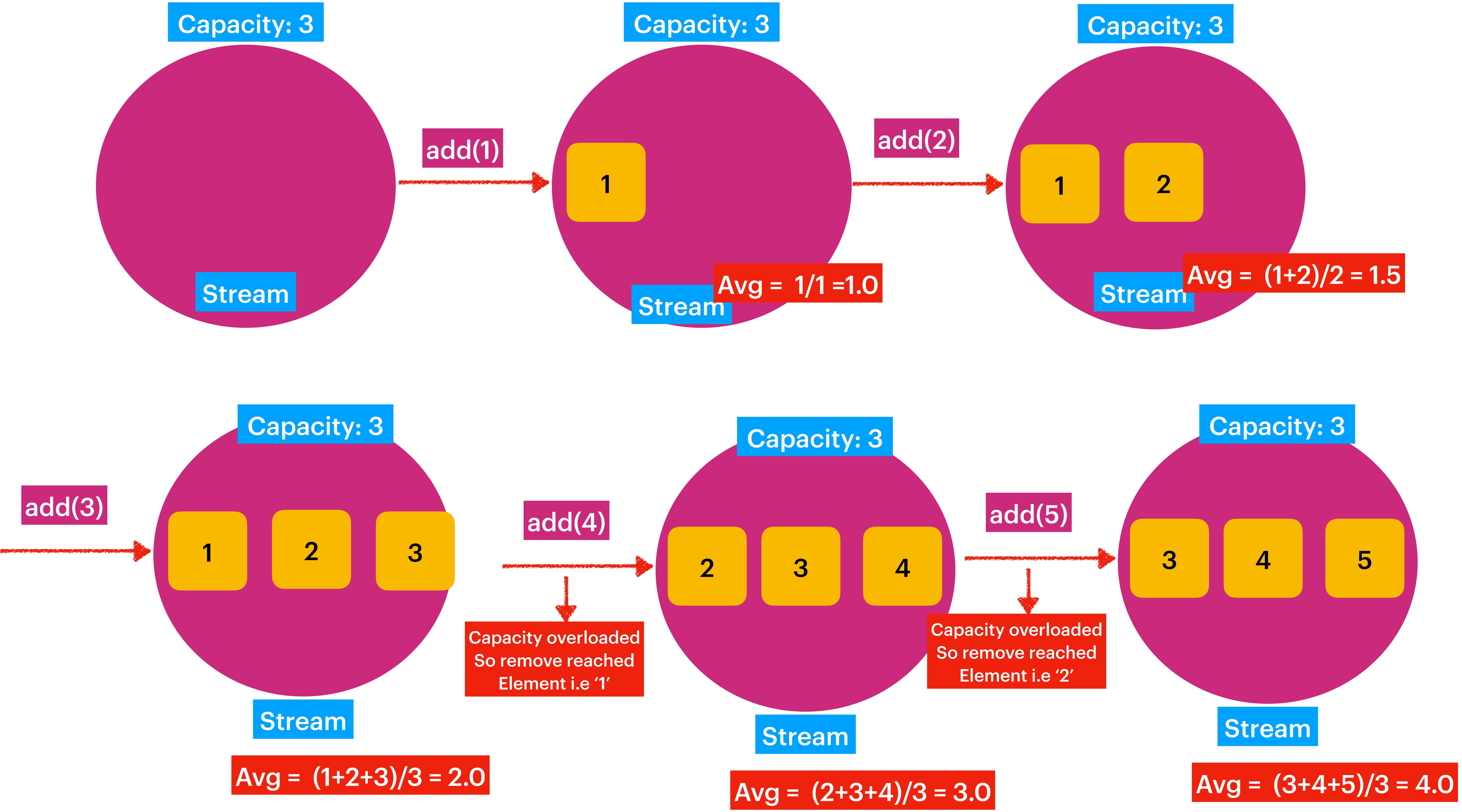
Example 1:

```
Input
["MovingAverage", "next", "next", "next", "next"]
[[3], [1], [10], [3], [5]]
Output
[null, 1.0, 5.5, 4.66667, 6.0]

Explanation
MovingAverage movingAverage = new MovingAverage(3);
movingAverage.next(1); // return 1.0 = 1 / 1
movingAverage.next(10); // return 5.5 = (1 + 10) / 2
movingAverage.next(3); // return 4.66667 = (1 + 10 + 3) / 3
movingAverage.next(5); // return 6.0 = (10 + 3 + 5) / 3
```

Constraints:

- `1 <= size <= 1000`
- $-10^5 <=$ `val` $<= 10^5$
- At most 10^4 calls will be made to `next`.


Capacity: 3
Stream
add(1)
Capacity: 3
1
Avg = 1/1 =1.0
Stream
add(2)
Capacity: 3
1
2
Avg = (1+2)/2 = 1.5
Stream
add(3)
Capacity: 3
1
2
3
Stream
Avg = (1+2+3)/3 = 2.0
add(4)
Capacity overloaded So remove reached Element i.e '1'
Capacity: 3
2
3
4
Stream
Avg = (2+3+4)/3 = 3.0
add(5)
Capacity overloaded So remove reached Element i.e '2'
Capacity: 3
3
4
5
Stream
Avg = (3+4+5)/3 = 4.0