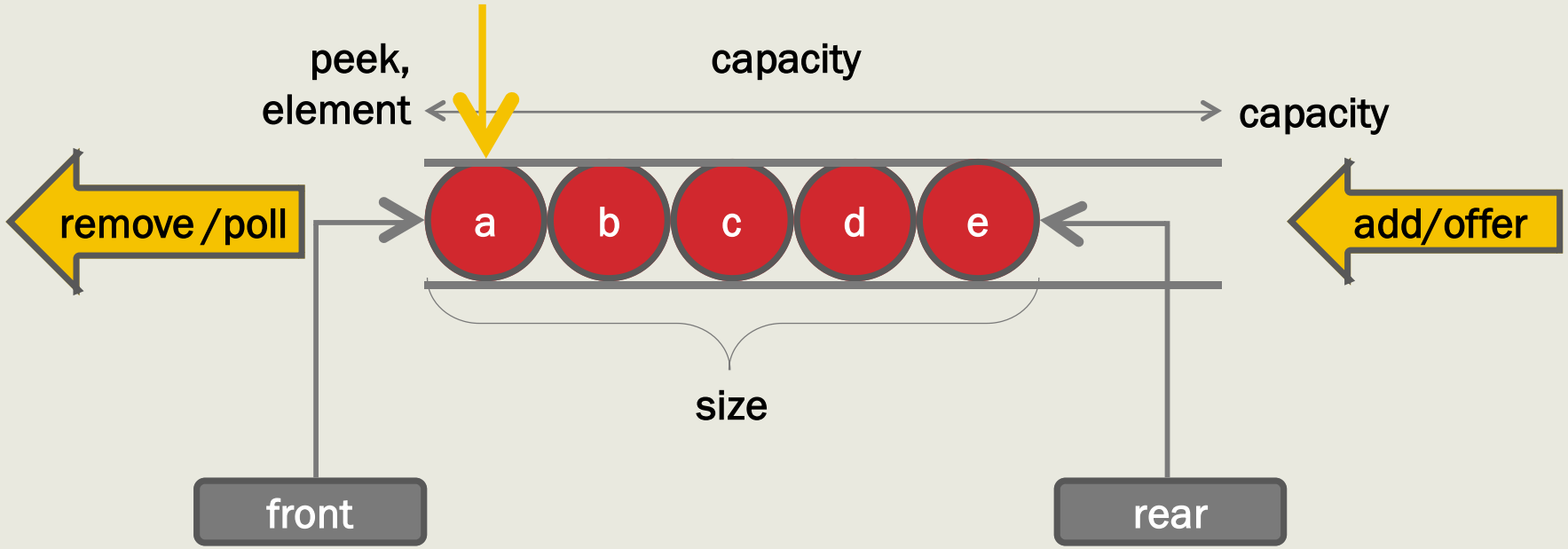
# QUEUE /ОПАШКА/

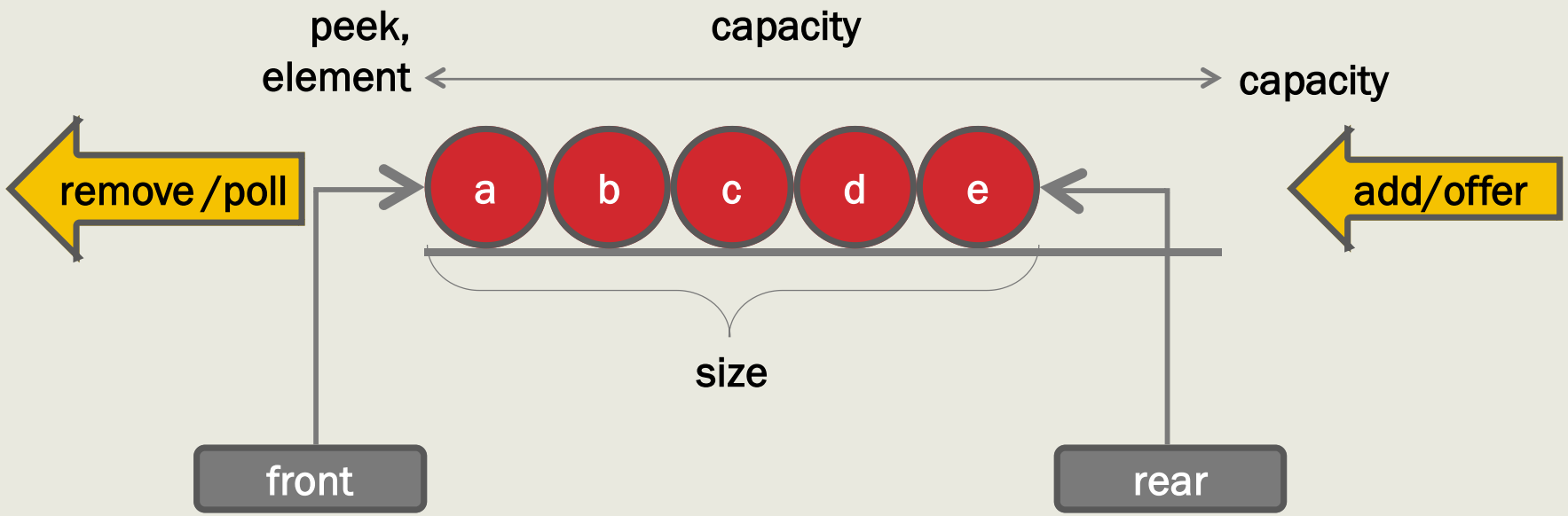
FIFO /First-In-First-Out/

# Queue /опашка/

- Колекцията от опашки се **използва за задържане на елементите, които предстои да бъдат обработени**, и осигурява различни операции като вмъкване, премахване и т.н.
- Това е подреден списък на обекти с използването му ограничено за вмъкване на елементи в края на списъка и изтриване на елементи от началото от списъка.
- Той следва принципа FIFO или принципа First-In-First-Out.
- Java Queue поддържа всички методи на  Collection interface.
- LinkedList, ArrayBlockingQueue и PriorityQueue най-често се използват за имплементиране на опашка.




- add() - Този метод се използва за добавяне на елементи в опашката на опашката.
- peek() - Този метод се използва за преглед на първия елемент на опашката, без да я премахвате. Връща Null, ако опашката е празна.
- element()- Този метод е подобен на peek(). Той връща грешка *NoSuchElementException*, когато опашката е празна.




- **remove() -** Този метод премахва и връща главата на опашката. Той хвърля NoSuchElementException, когато опашката е празна.
- **poll()-** Този метод премахва и връща главата на опашката. Връща null, ако опашката е празна.
- **size() -** Този метод връщане броя на елементи в опашката
- Тъй като е подтип на Collections class, тя наследява всички методи като *size(), isEmpty(), contains() etc.*

# Пример, който илюстрира тези методи

```java
// Java orogram to demonstrate working of Queue
// interface in Java
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample
{
  public static void main(String[] args)
  {
    Queue<Integer> q = new LinkedList<>();

    // Adds elements {0, 1, 2, 3, 4} to queue
    for (int i=0; i<5; i++)
     q.add(i);

    // Display contents of the queue.
    System.out.println("Elements of queue-"+q);

    // To remove the head of queue.
    int removedele = q.remove();
    System.out.println("removed element-" + removedele);

    System.out.println(q);
```

1

add

remove

# Пример, който илюстрира тези методи

2

```
    // To view the head of queue
    int head = q.peek();
    System.out.println("head of queue-" + head);

    // Rest all methods of collection interface,
    // Like size and contains can be used with this
    // implementation.
    int size = q.size();
    System.out.println("Size of queue-" + size);
  }
}
```

peek

size

**Output:**

```
Elements of queue-[0, 1, 2, 3, 4]
removed element-0
[1, 2, 3, 4]
head of queue-1
Size of queue-4
```

# Проверка дали присъства даден елемент в опашка

```
Queue<String> queue = new LinkedList<>();

queue.add("Mazda");

boolean containsMazda = queue.contains("Mazda");
boolean containsHonda = queue.contains("Honda");
```

След стартиране на този код containsMazda променливата ще има стойността, true докато containsHonda променливата ще има стойността false - тъй като опашката съдържа стринг елемент "Mazda", но не и низ елемент "Honda".

# Iterate All Elements in Queue

```
Queue<String> queue = new LinkedList<>();

queue.add("element 0");
queue.add("element 1");
queue.add("element 2");

//access via Iterator
Iterator<String> iterator = queue.iterator();
while(iterator.hasNext(){
  String element = iterator.next();
}

//access via new for-loop
for(String element : queue) {
    //do something with each element
}
```

Iterating a Queue via its **Iterator**

Iterating a Queue via **for-each loop**

# Използване на опашка

Нека сега разгледаме прост пример. Да си създадем една опашка и добавим в нея няколко елемента. След това ще извлечем всички чакащи елементи и ще ги изведем на конзолата:

```
public static void main(String[] args) {
   Queue<String> queue = new LinkedList<String>();
   queue.offer("Message One");
   queue.offer("Message Two");
   queue.offer("Message Three");
   queue.offer("Message Four");
```

```
   while (queue.size() > 0) {
      String msg = queue.poll();
      System.out.println(msg);
   }
}
```

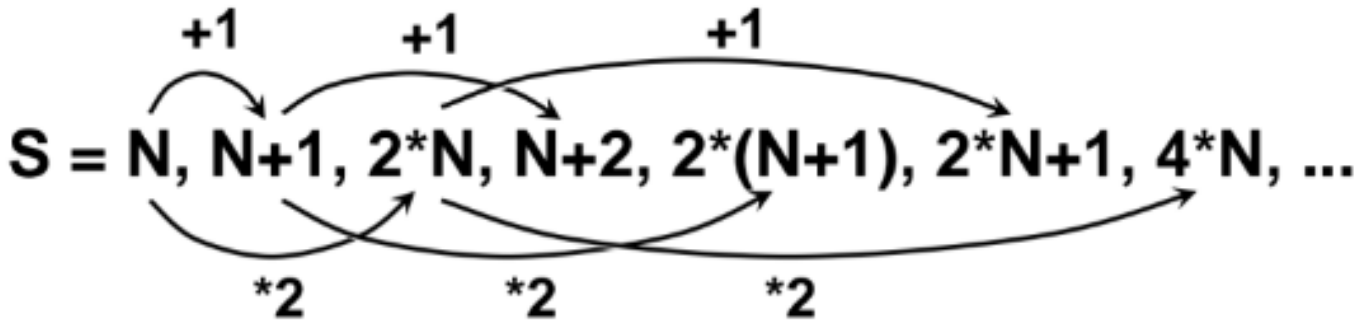
Ето как изглежда изходът е примерната програма:

```
Message One
Message Two
Message Three
Message Four
```

Вижда се, че елементите излизат от опашката в реда, в който са постъпили в нея.

## Редицата N, N+1, 2*N – пример

Нека сега разгледаме задача, в която използването на структурата опашка ще бъде много полезна за реализацията. Да вземем редицата числа, чиито членове се поличават по-следния начин: първият елемент е N; вторият получаваме като съберем N с 1; третият – като умножим първия с 2 и така последователно умножаваме всеки елемент с 2 и го добавяме накрая на редицата, след което го събираме с 1 и отново го поставяме накрая на редицата. Можем да илюстрираме този процес със следната фигура:




Както виждаме, процесът се състои във взимане на елементи от началото на опашка и поставянето на други в края ѝ. Нека сега видим примерна реализация, в която N=3 и търсим номера на член със стойност 16:

# Използване на опашка

```
public static void main(String[] args) {
   int n = 3;
   int p = 16;

   Queue<Integer> queue = new LinkedList<Integer>();
   queue.offer(n);
   int index = 0;
   System.out.print("S =");
   while (queue.size() > 0) {
      index++;
      int current = queue.poll();
      System.out.print(" " + current);
      if (current == p) {
         System.out.println();
         System.out.println("Index = " + index);
         return;
      }
      queue.offer(current + 1);
      queue.offer(2 * current);
   }
}
```

Ето как изглежда изходът е примерната програма:

```
S = 3 4 6 5 8 7 12 6 10 9 16
Index = 11
```