Chapter 3

QUEUES

Objective

- To introduce:

  Queues concept

  Queues operations

  Application using queues

CONTENT

3.1 Introduction

3.2 Queues using Array-Based Implementation

3.3 Queues Application: Queue Simulation

3.1 Introduction

- Queue is linear list in which data can only be inserted at one end, called the **rear**, and deleted from the other end, called the **front**.
- First-In-First-Out (FIFO) concept.

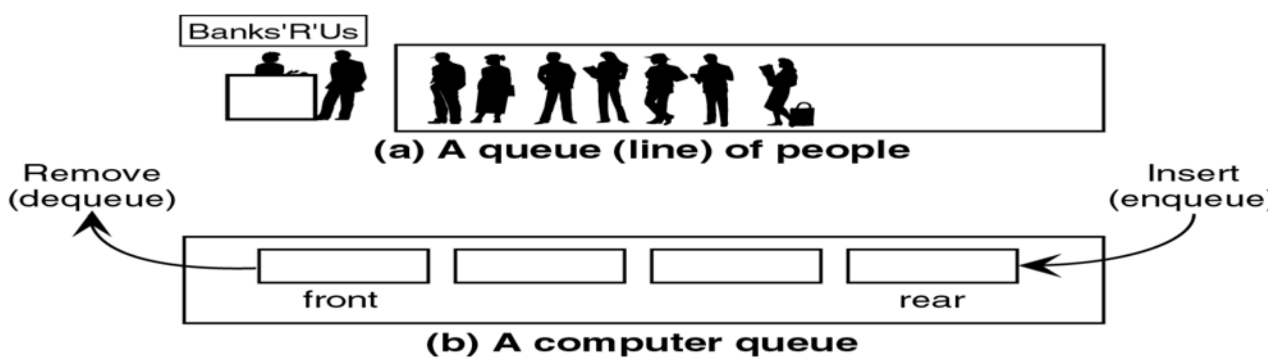


Figure 1: The queue concept

3.1 Introduction

- Two simple queue applications:

  ✓**Queue Simulation** – modeling activity used to generate statistics about the performance of queues.

  ✓**Categorizing Data** – rearrange data without destroying their basic sequence (multiple queues).

3.1 Introduction

- Basic operations:

1) Enqueue– insert a given element at the back of the queue.

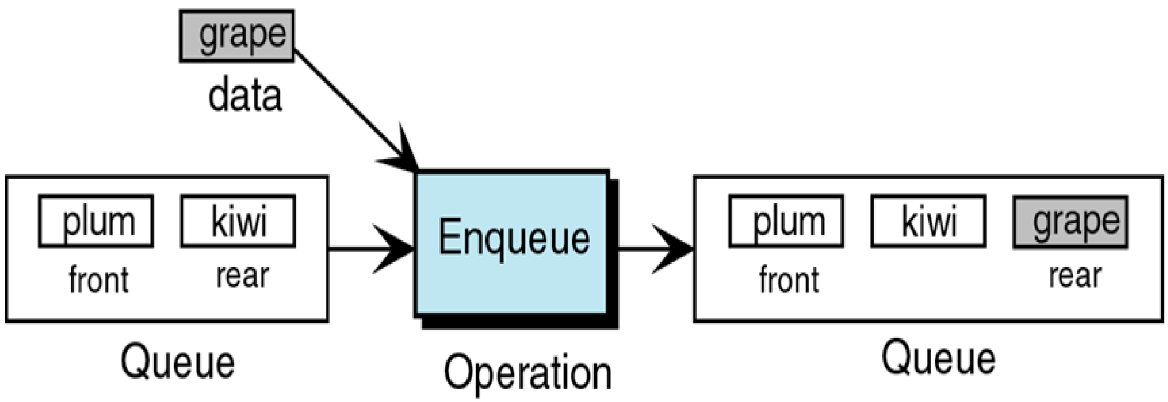


Figure 2: Add

3.1 Introduction

2) Dequeue – if the queue is not empty, delete and return the element that is at the front of the queue.

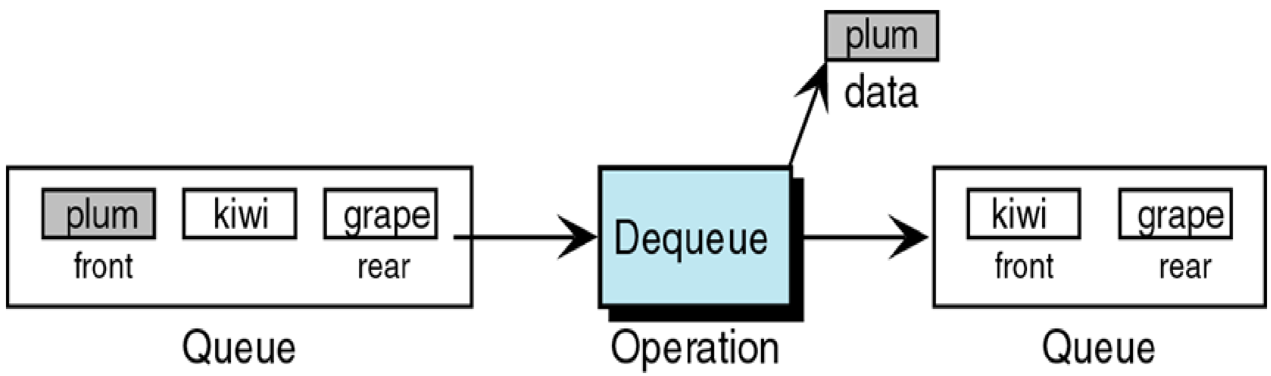


Figure 3: Remove

3.1 Introduction

3) Queue_front – if the queue is not empty, return the element that is at the front of the queue.

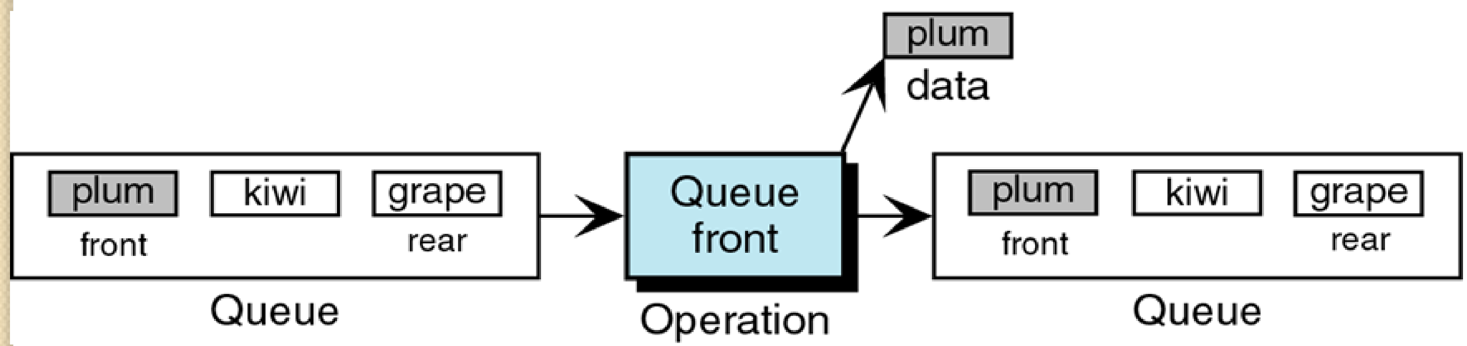


Figure 4: First

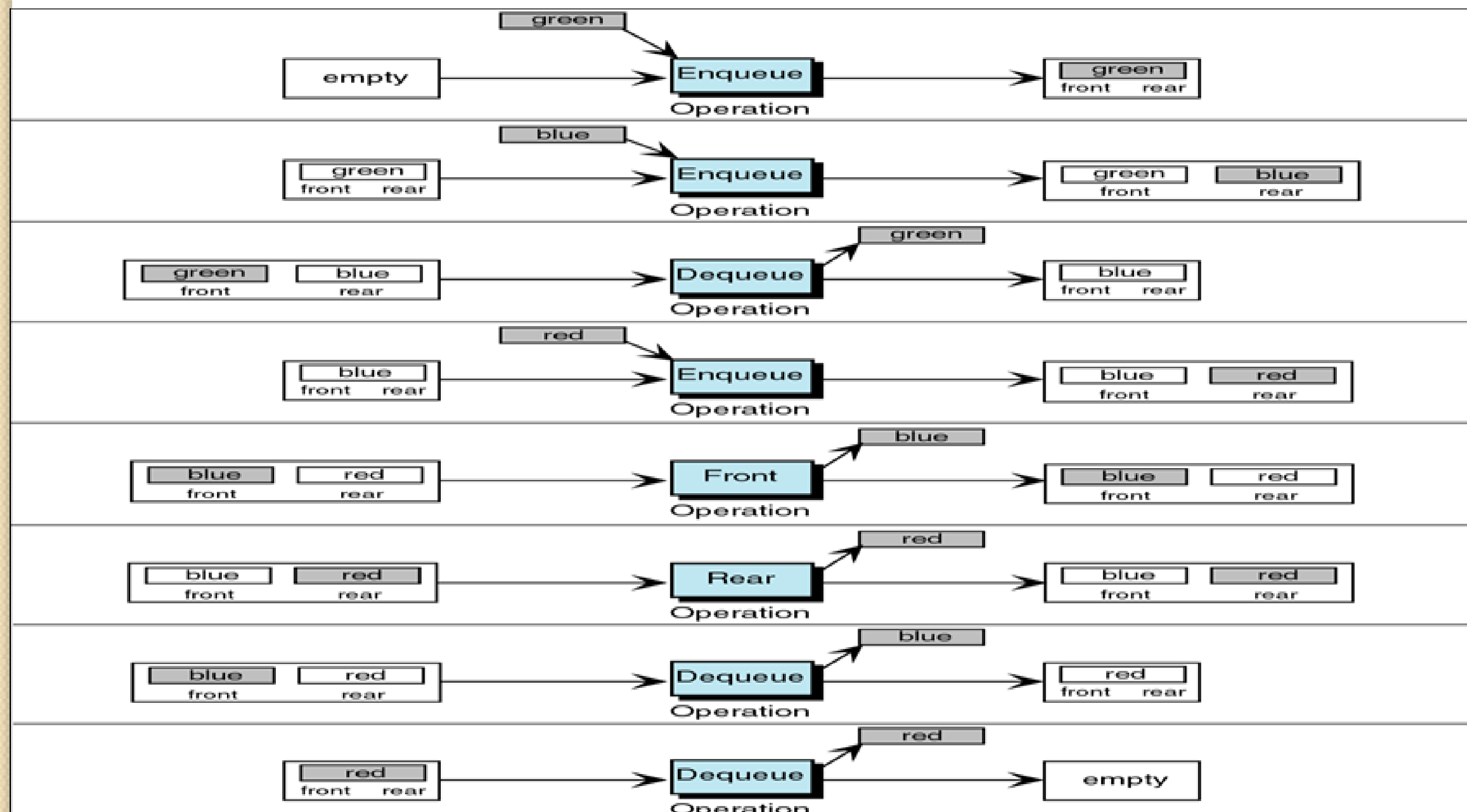
4) Size – return the number of elements in the queue.

- Example:

Given queue Q with list of operations as below. Illustrate the queue operations step by step:

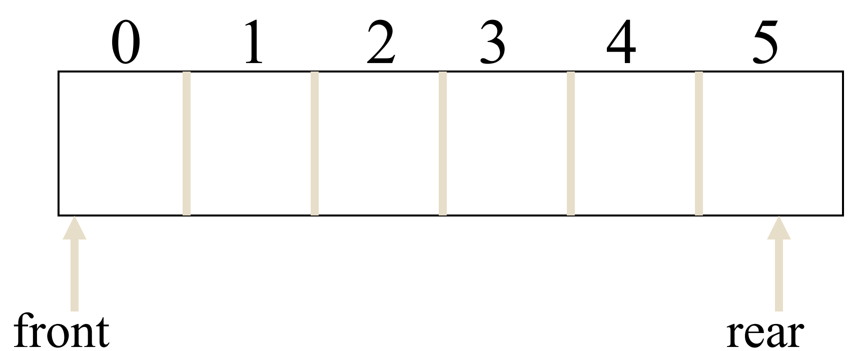
Q.enqueue(green), Q. enqueue(blue), x=Q.dequeue( ), Q.enqueue (red),

frontItem = Q.Front( ), lastItem = Q.Rear(), x=Q.dequeue( ), y=Q.dequeue().



QUEUE USING ARRAY-BASED IMPLEMENTATION

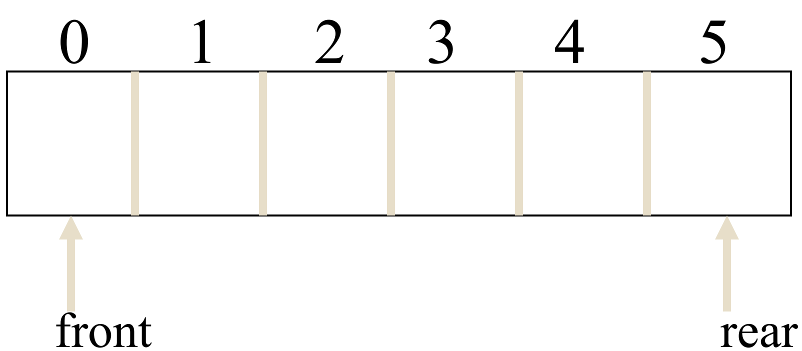
- Array Implementation
  - Different compared to Stack because of two variables: front and rear.
  - Let Q store characters, defined as arrays of 6 elements numbered 0 to 5 as follows with front and rear index.



QUEUE USING ARRAY-BASED IMPLEMENTATION

1. Start operation: construct a queue

```
Queue a;
front = 0, rear = 5;
```

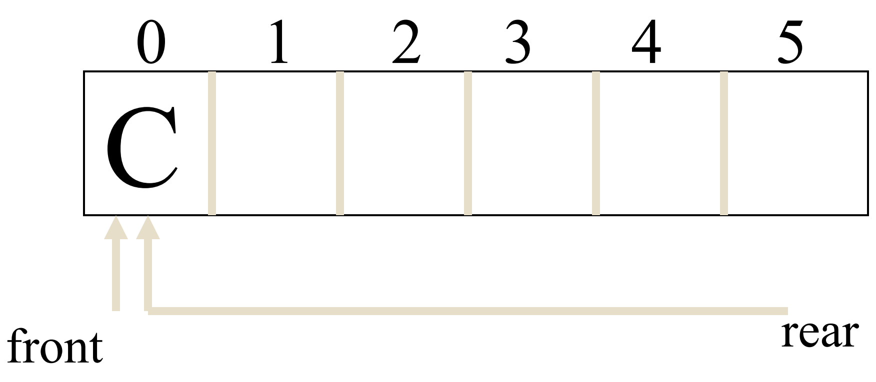


QUEUE USING ARRAY-BASED IMPLEMENTATION

2. Input character 'C', 'S', 'C', '3', '2', '0', '2':

```
a.enqueue('C');

rear = (rear + 1) % a.length;
a[rear] = newInput;
```

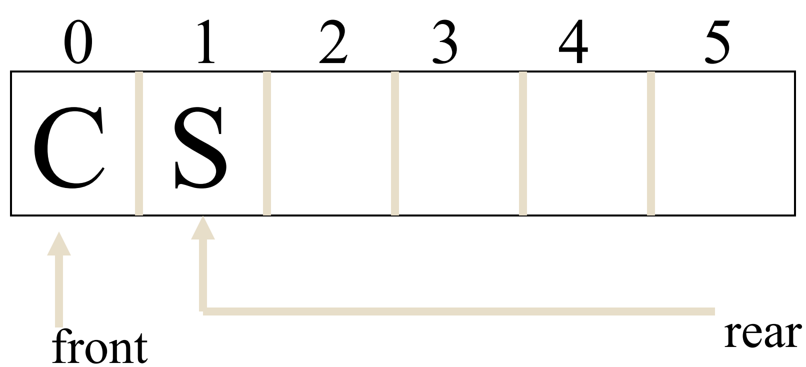


QUEUE USING ARRAY-BASED IMPLEMENTATION

3. Next input:

```
a.enqueue('S');
```

```
rear = (rear + 1) % a.length;
a[rear] = newInput;
```

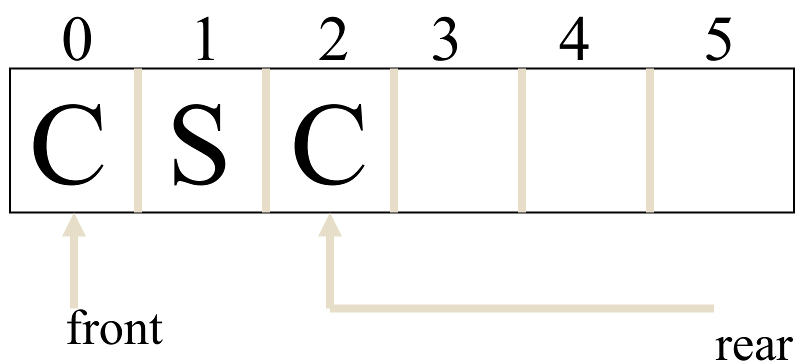


QUEUE USING ARRAY-BASED IMPLEMENTATION

4. Next input:

a.enqueue('C');

```
rear = (rear + 1) % a.length;
a[rear] = newInput;
```

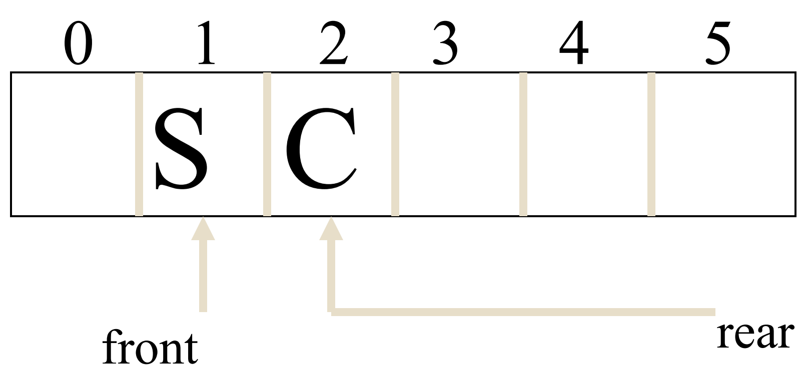


QUEUE USING ARRAY-BASED IMPLEMENTATION

5. Remove operation: remove the front item

```
a.dequeue()

ch = a[front];
a[front] = null;
front = (front + 1) % a.length;
```

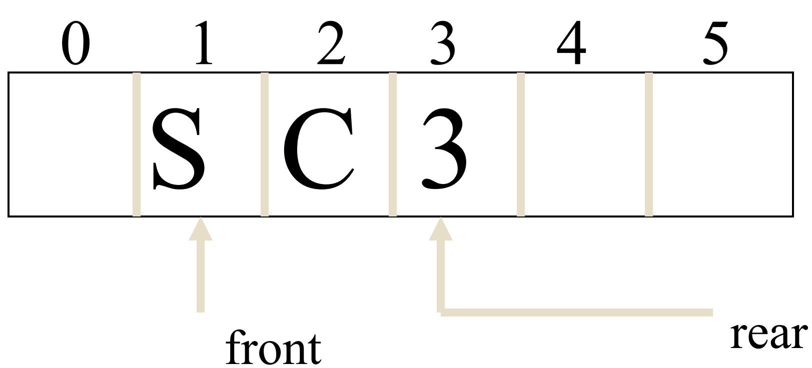


QUEUE USING ARRAY-BASED IMPLEMENTATION

6. Next input:

```
a.enqueue('3');

rear = (rear + 1) % a.length;
a[rear] = newInput;
```

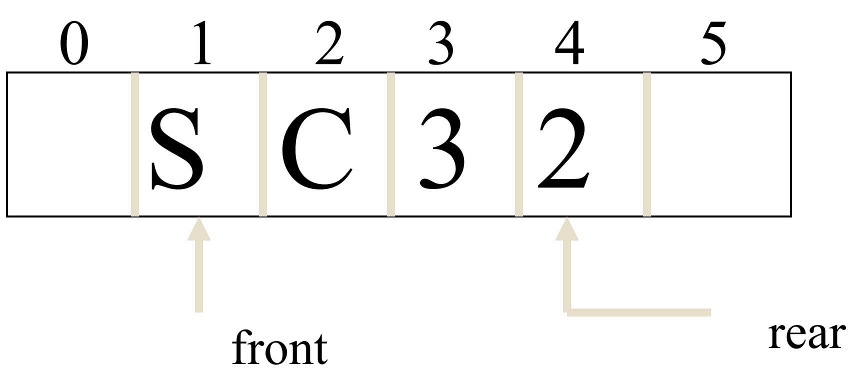


QUEUE USING ARRAY-BASED IMPLEMENTATION

7. Next input:

```
a.enqueue('2');

rear = (rear + 1) % a.length;
a[rear] = newInput;
```

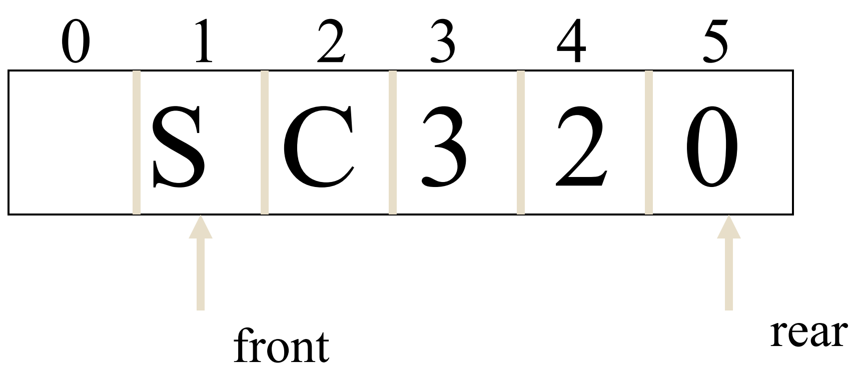


QUEUE USING ARRAY-BASED IMPLEMENTATION

8. Next input:

```
a.enqueue('0');

rear = (rear + 1) % a.length;
a[rear] = newInput;
```

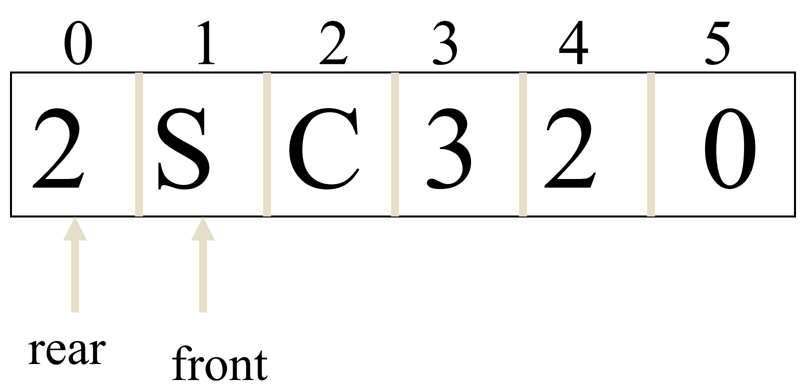


QUEUE USING ARRAY-BASED IMPLEMENTATION

9. Next input:

```
a.enqueue('2');

rear = (rear + 1) % a.length;
a[rear] = newInput;
```

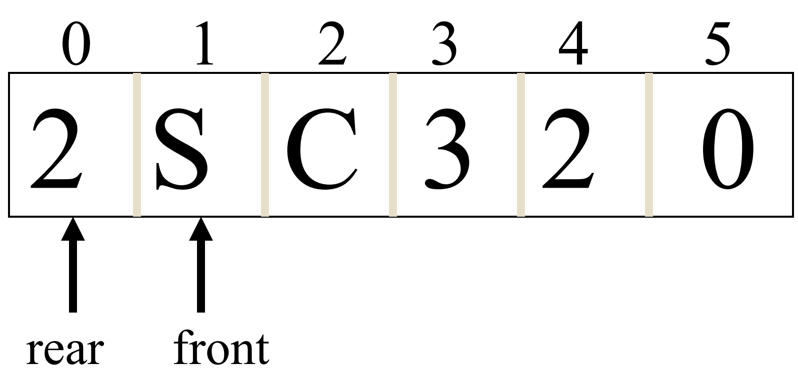


QUEUE USING ARRAY-BASED IMPLEMENTATION

10. The queue is already full. Before you can insert anymore character, check the queue first:

```
public boolean isFull() {
    return (size == a.length);
}
```

Cannot just only test whether `front == ((rear + 1) % a.length` since it is true both when queue is full and when queue is empty.

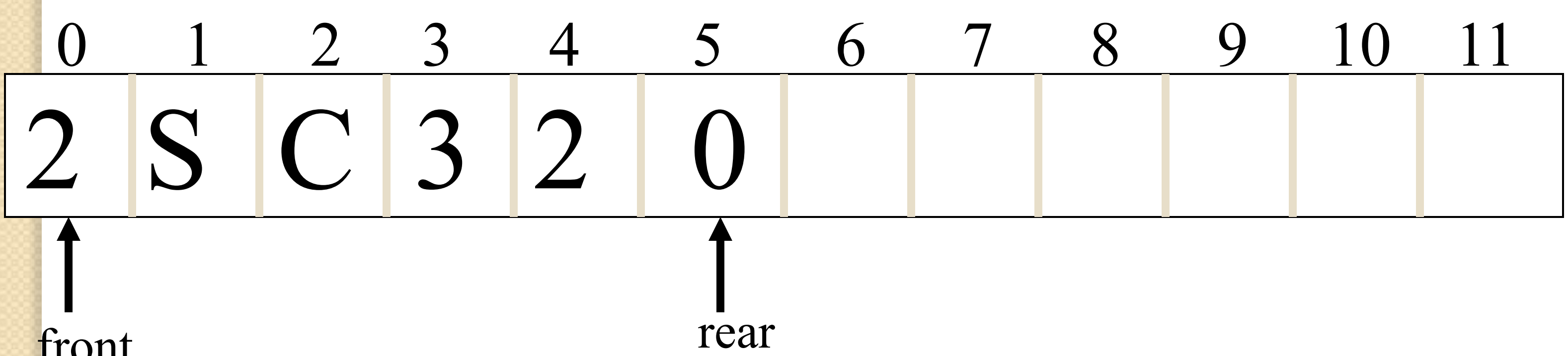


11. You can increase the size:

```
if (isFull()) {
     Object[] oldQ = a;

     // Create new array of double size
     a = new Object[2 * oldQ.length];

     // Copy old array contents into new array
     for (int i = 0; i < oldQ.length; i++) {
        a[i] = oldQ[front];
        front = (front + 1) % oldQ.length;
     }
     front = 0;
     rear = oldQ.length - 1;
}
```



QUEUE USING ARRAY-BASED IMPLEMENTATION

Queue Program: A Queue Interface

```
/* The <code>Queue</code> interface specifies the
 * basic operations
 * of a first-in-first-out (FIFO) containers. */

public interface Queue {

/* Adds the specified element to the back of this
 * queue. */

public void enqueue(Object object);

```

```
/*Returns the element at the front of this queue.
 *
* @throws IllegalStateException if this queue is
* empty */

public Object getFront();

/* Removes and returns the element at the front of
 * this queue.
 *
 * @throws IllegalStateException if this queue is
 * empty */

public Object dequeue();

/* Returns the number of elements in this queue. */

public int size();
}
```

QUEUE USING ARRAY-BASED IMPLEMENTATION

- Queue Program: A Queue Interface

```
/* The <code>Queue</code> interface specifies the
 * basic operations
 * of a first-in-first-out (FIFO) containers. */

public interface Queue {

/* Adds the specified element to the back of this
 * queue. */

public void enqueue(Object object);

```

```
/*Returns the element at the front of this queue.
 *
  * @throws IllegalStateException if this queue is
  * empty */

public Object getFront();

/* Removes and returns the element at the front of
 * this queue.
 *
  * @throws IllegalStateException if this queue is
  * empty */

public Object dequeue();

/* Returns the number of elements in this queue. */

public int size();
}
```

QUEUE USING ARRAY-BASED IMPLEMENTATION

The ArrayQueue class

```
public class ArrayQueue implements Queue {
   private Object[] objQ;
   private int front;
   private int rear;
   private int size;

   public ArrayQueue(int capacity) {
      objQ = new Object[capacity];
      front = 0;
      rear = capacity - 1;
      size = 0;
   }
```

```
public void enqueue(Object object) {
   ensureCapacity();
   rear = (rear + 1) % objQ.length;
   objQ[rear] = object;
   size++;
}

public Object dequeue() {
   Object frontObj;

   if (isEmpty()) throw new
         IllegalStateException("queue is
    empty");

   frontObj = objQ[front];
   front = (front + 1) % objQ.length;
   size--;

   return frontObj;
}
```

```java
public boolean isFull() {
    return size == objQ.length;
}

private void ensureCapacity() {
   if (isFull()) {
      Object[] oldQ = objQ;

      objQ = new Object[2 * oldQ.length];

      for (int i = 0; i < oldQ.length; i++) {
         objQ[i] = oldQ[front];
         front = (front + 1) % oldQ.length;
      }
      front = 0;
      rear = oldQ.length - 1;
   }
}
```

```java
public boolean isEmpty() {
   return size == 0;
}

public Object getFront() {
   Object frontObj;

   if (isEmpty()) throw new
      IllegalStateException("queue is empty");

   frontObj = objQ[front];

   return frontObj;
}

public int size() {
   return size;
}
```

The implementation class:

```
public class TestArrayQueue {
  public static void main (String[] args) {
      ArrayQueue queue = new ArrayQueue(4);

      queue.enqueue("CARROTS");
      queue.enqueue("ORANGES");
      queue.enqueue("RAISINS");
      queue.enqueue("PICKLES");
      System.out.println("queue.size(): " + queue.size() +
              "\tqueue.getFront(): " + queue.getFront());
      System.out.println("queue.dequeue(): " +
                         queue.dequeue());
      System.out.println("queue.dequeue(): " +
                         queue.dequeue());
      System.out.println("queue.dequeue(): " +
                         queue.dequeue());
```

```java
    System.out.println("queue.size(): " + queue.size() +
            "\tqueue.getFront(): " + queue.getFront());
    queue.enqueue("WALNUTS");
    queue.enqueue("OYSTERS");
    queue.enqueue("BANANAS");
    System.out.println("queue.size(): " + queue.size() +
            "\tqueue.getFront(): " + queue.getFront());
    System.out.println("queue.dequeue(): " +
                            queue.dequeue());
    System.out.println("queue.dequeue(): " +
                            queue.dequeue());
    System.out.println("queue.dequeue(): " +
                            queue.dequeue());
    System.out.println("queue.dequeue(): " +
                            queue.dequeue());
    System.out.println("queue.dequeue(): " +
                            queue.dequeue());
  }
}
```

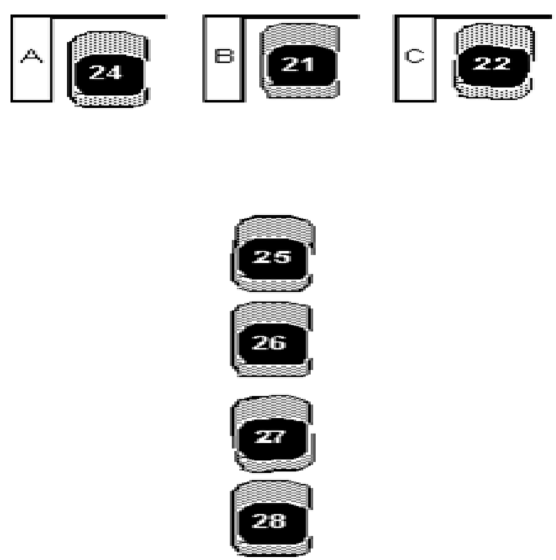
```
queue.size(): 4    queue.getFront(): CARROTS
queue.dequeue(): CARROTS
queue.dequeue(): ORANGES
queue.dequeue(): RAISINS
queue.size(): 1    queue.getFront(): PICKLES
queue.size(): 4    queue.getFront(): PICKLES
queue.dequeue(): PICKLES
queue.dequeue(): WALNUTS
queue.dequeue(): OYSTERS
queue.dequeue(): BANANAS
Exception in thread "main"
  java.lang.IllegalStateException: queue is empty
        at ArrayQueue.dequeue(ArrayQueue.java:##)
        at TestArrayQueue.main(TestArrayQueue.java:##)
```

3.3 QUEUE APPLICATION: Queue Simulations

- Anything involving FIFO
  - Lines of people at checkout counters
  - Vehicles at traffic lights
  - Planes waiting to land and take off at airports
  - Assembly lines at a factory
  - Email processing
    - Some emails can have higher priority than others – use a **priority queue**?
- Usually need to model arrivals into queue using probability distributions.

3.3 QUEUE SIMULATION (John R. Hubbard)

- Consider the real-world example of cars arriving at a station of toll booths. The clients are the cars and the servers are the toll booths (or their operators).
- The client/server system is pictured as below with three toll booths, labeled A, B and C. The cars are numbered. Cars 24, 21 and 22 are being served, while cars 25-28 are waiting in the queue.



3.3 QUEUE APPLICATION: QUEUE SIMULATION

- The objects are:
  - ✓ Clients (cars)
  - ✓ Servers (toll booths)
  - ✓ A queue

- The events are:
  - ✓ A client arrives at the queue
  - ✓ A server begins serving a client
  - ✓ A server finishes serving a client

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Algorithm – Client/Server Simulation

1. Repeat steps 2 and 6 for times t = 0, 1, ...
2. If **t** = time for next arrival, do steps 3-5.
3. Create a new **client.**
4. Add the **client** to the queue.
5. Set **time** for next arrival.
6. Repeat steps 7 and 8 for each server.
7. If t = **time** for the **server** to finish serving, have it stop.
8. If **server** is idle and the queue is not empty, do steps 9-10.
9. Remove **client** from the queue.
10. Tell **server** to start serving **client**.

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Client/Server Simulation

```
for (int t=0; ; t++) {   //step 1
  if (t==nextArrivalTime) {    //step 2
    Client client = clients[i++] = new SimClient(i,t);   //step 3
    queue.add(client);   //step 4
    nextArrivalTime = t + randomArrival.nextInt();   //step 5
  }
  for (int j=0; j<numServers; j++) {   //step 6
    Server server = servers[j];
    if (t==server.getStopTime()) server.stopServing(t); //step 7
    if (server.isIdle() && !queue.isEmpty()) {    //step 8
      Client client = (SimClient)queue.remove();   //step 9
      server.startServing(client,t);   //step 10
    }
  }
}
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Server Interface

```java
public interface Server {
    public int getMeanServiceTime();
    public int getStopTime();
    public boolean isIdle();
    public void startServing(Client client, int t);
    public void stopServing(int t);
}
```

- Client Interface

```java
public interface Client {
    public void setStartTime(int t);
    public void setStopTime(int t);
}
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Server and Client objects

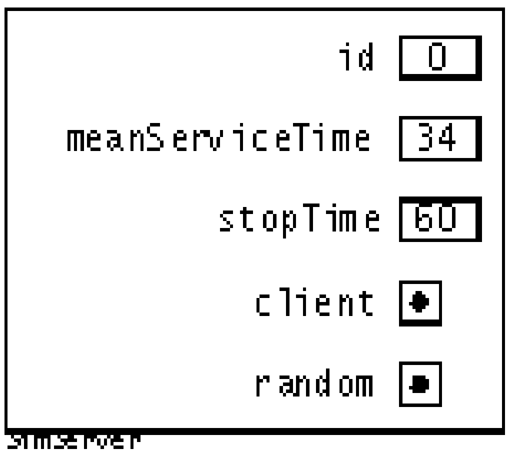


Figure 6.8 A **SimServer** object

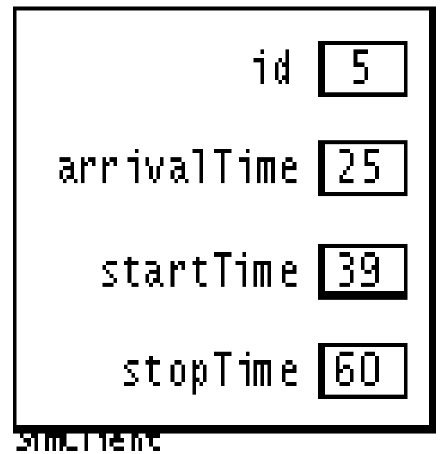


Figure 6.9 A **SimClient** object

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- A Server class

```
public class SimServer implements Server {
  private Client client;
  private int id, meanServiceTime, stopTime=-1;
  private java.util.Random random;

  public SimServer(int id, int meanServiceTime) {
    this.id = id;
    this.meanServiceTime = meanServiceTime;
    this.random = new ExponentialRandom(meanServiceTime);
  }

  public int getMeanServiceTime() {
    return meanServiceTime;
  }

  public int getStopTime() {
    return stopTime;
  }
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

```
public boolean isIdle() {
    return client==null;  }

public void startServing(Client client, int t) {
    this.client = client;
    this.client.setStartTime(t);
    this.stopTime = t + random.nextInt();
    System.out.println(this + " started serving " + client
      + " at time " + t + " and will finish at time " + stopTime);  }

public void stopServing(int t) {
    client.setStopTime(t);
    System.out.println(this+ " stopped serving " + client
      + " at time " + t);
    client = null;
  }

  public String toString() {
    String s="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    return "Server " + s.charAt(id);  }
}
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Exponential Distribution

```
public class ExponentialRandom extends java.util.Random {
  private double mean;

  public ExponentialRandom(double mean) {
    super(System.currentTimeMillis());
    this.mean = mean;
  }

  public double nextDouble() {
    return -mean*Math.log(1.0-super.nextDouble());
  }

  public int nextInt() {
    return (int)Math.ceil(nextDouble());
  }
}
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- A Client Class

```
public class SimClient implements Client {
  int id, arrivalTime=-1, startTime=-1, stopTime=-1;
  public SimClient(int id, int t) {
    this.id = id;
    arrivalTime = t;
    System.out.println(this + " arrived at time " + t);   }

  public int getStartTime() {
    return startTime;   }

  public int getStopTime() {
    return stopTime;   }

  public void setStartTime(int t) {
    startTime = t;   }

  public void setStopTime(int t) {
    stopTime = t;   }

  public String toString() {
    return "Client " + id;   }
}
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- A Simulation class

```
public class Simulation {
  static int numServers;
  static int numClients;
  static int meanServiceTime;
  static int meanInterarrivalTime;
  static Server[] servers;
  static Client[] clients;
  static Queue queue = new LinkedQueue();
  static java.util.Random randomArrival;
  static java.util.Random randomService;

  public static void main(String[] args) {
    init(args);
    //  See Listing 6.3 on page 173 (John R. Hurbbard)
  }
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

```
static void init(String[] args) {
    if (args.length<4) {
      System.out.println("Usage: java Simulation <numServers> "
      + "<numClients> <meanServiceTime> <meanInterarrivalTime>");
      System.out.println(" e.g.: java Simulation 3 100 12 4");
      System.exit(0);
    }
    numServers = Integer.parseInt(args[0]);
    numClients = Integer.parseInt(args[1]);
    meanServiceTime = Integer.parseInt(args[2]);
    meanInterarrivalTime = Integer.parseInt(args[3]);
    servers = new Server[numServers];
    clients = new Client[numClients];
    randomService = new ExponentialRandom(meanServiceTime);
    randomArrival = new ExponentialRandom(meanInterarrivalTime);
    queue = new LinkedQueue();
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

```
for (int j=0; j<numServers; j++)
      servers[j] = new SimServer(j,randomService.nextInt());
    System.out.println("     Number of servers = " + numServers);
    System.out.println("     Number of clients = " + numClients);
    System.out.println("     Mean service time = " + meanServiceTime);
    System.out.println("Mean interarrival time = "
    + meanInterarrivalTime);
    for (int j=0; j<numServers; j++)
      System.out.println("Mean service time for " + servers[j]
        + " = "+ servers[j].getMeanServiceTime());
  }}
```

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Simulation objects

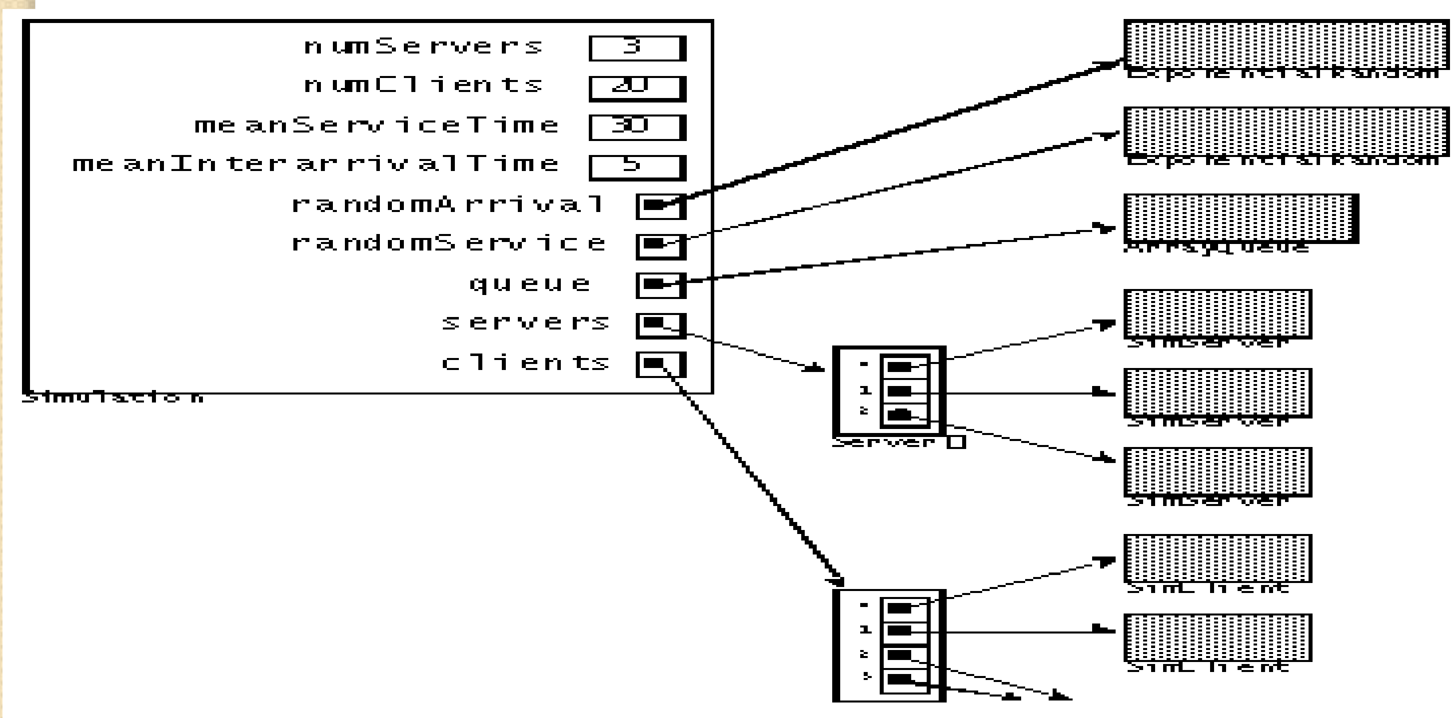
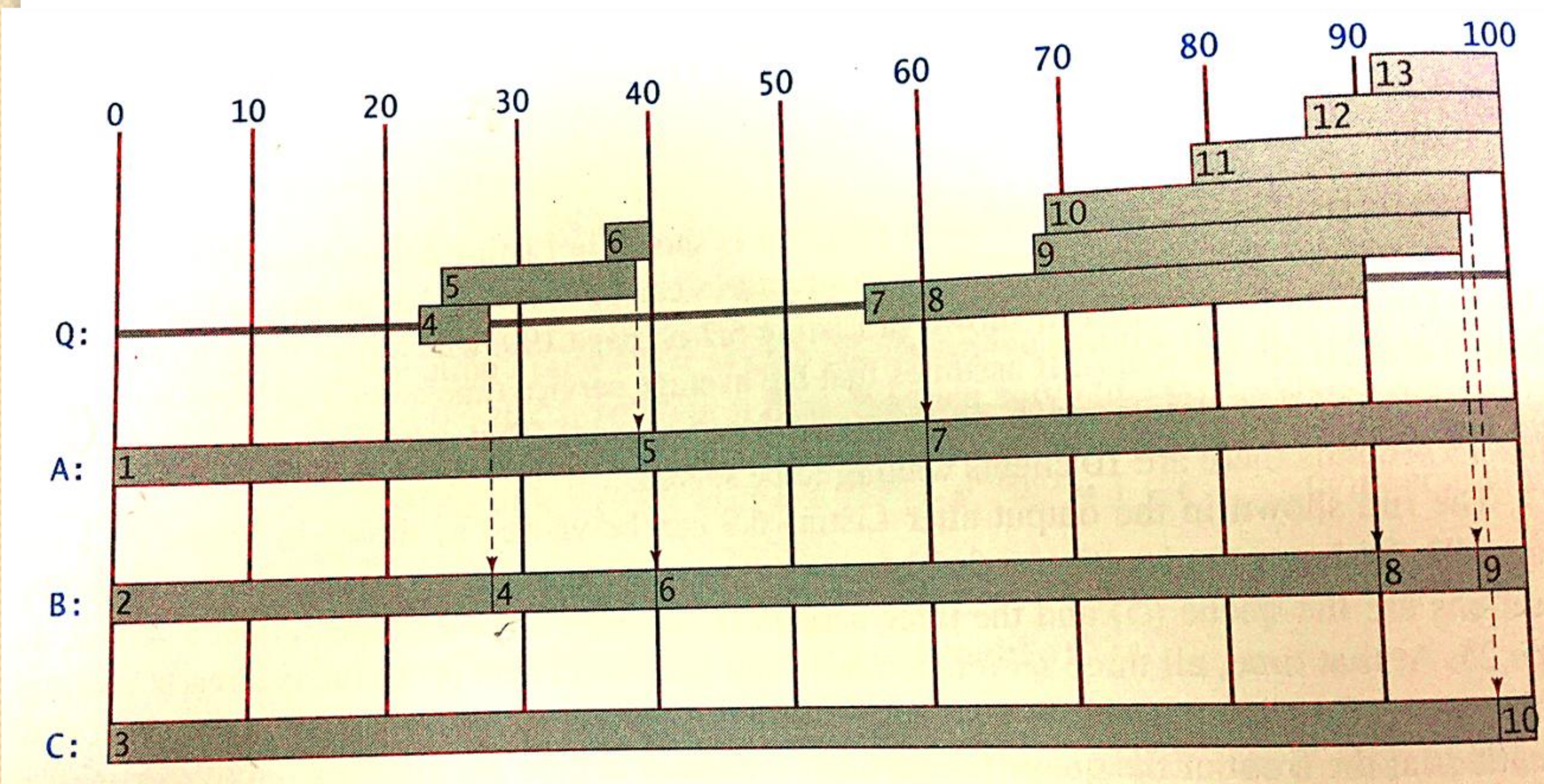


Figure 6.10 Simulation objects

3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Arrivals and Departures



3.3 QUEUE APPLICATION: QUEUE SIMULATION

- Output

Number of servers = 3

Number of clients = 20

Mean service time = 30

Mean interarrival time = 5

Mean service time for Server A = 34

Mean service time for Server B = 19

Mean service time for Server C = 78

Client 1 arrived at time 0

The queue has 1 clients

The queue has 0 clients

3.3 QUEUE APPLICATION: QUEUE SIMULATION

Server A started serving Client 1 at time 0 and will finish at time 39

Client 2 arrived at time 6

The queue has 1 clients

The queue has 0 clients

Server B started serving Client 2 at time 6 and will finish at time 28

Client 3 arrived at time 10

The queue has 1 clients

The queue has 0 clients

Server C started serving Client 3 at time 10 and will finish at time 98

…

…

…