GMU Spring 2023 – CS 211 – Project 3

Due Date: Sunday, April 2nd, 11:59 pm

Changelog

- Typos fixed:
  - QueueSystem instead of QueueingSystem.
  - Default values of some attributes.
  - Parameter types for Client and VIPClient.
  - Some other typos fixed / Additional clarification added.
  - clientsInQueue attribute in class Queue was missing.
- Additional clarification regarding the increaseTime()
- Assume that the array of Request items Client class will always be sorted in desc order of priority.
- clock=1, start of the simulation, after the first call of increaseTime().

Description

Let's assume we're developing a queueing policy evaluation tool for a company facing high demand. You will implement a program that can simulate and evaluate the performance of a given queueing policy.

There is no template provided for this exercise, so make sure that you read the instructions carefully to determine how your code should be structured. You will be making ten classes in their respective .java files. Make sure to include all the fields and methods asked for, paying particular attention to the access modifiers, return types, capitalization, and parameter order and type.

Overview

1. Create the Java classes Prioritizable, Client, VIPClient, Request, InformationRequest, ReturningItems, BuyingProducts, QueueSystem, Queue, VIPQueue. You will also need to write the Enums Gender and Status.
2. Include all the fields and methods described below.
3. Test your code for correctness.
4. Prepare the assignment for submission and submit the java files through Gradescope.

Rules

1. You may not import any extra functionality besides the default. For example, System and Math are imported by default and thus may be used, whereas something like ArrayList must be explicitly imported so it is disallowed.
2. The main method will not be tested; you may use it any way you want.

3. Comment your code, especially the parts where it is not obvious what you're doing.

Instructions

1. Implement the java classes described below in distinct, appropriately named, .java files.
2. You must do your own testing. Feel free to edit the main method in the driver class to test your code.
3. Submit the files on Gradescope. Make sure that you submit the correct files.
4. For all the private fields define **getters and setters** following this signature:
    a. The getter for a field declared as private String someField will be: **public String getSomeField()**
    b. The setter for a field declared as private String someField will be: **public void setSomeField(String someField)**
5. Feel free to add additional methods or attributes to the classes. **The additional fields must be private.**

The Prioritizable interface:

This interface will be implemented by classes that represent objects to which we can assign a priority value. In this assignment, the Request and VIPClient classes will implement this interface.

Any classes implementing Prioritizable will have the implementation of these methods:

- public void **setPriority**(int a);
- public int **getPriority**();

The Client class:

This class represents a client who arrives at a specific time in the system.

If no server is available, the client joins the waitingLine before joining the first available queue. If the waitingLine is full, the client leaves without being served. See more detailed information about this flow on page 10.

You need to implement constructors and methods to instantiate the fields of a Client.

➔ **Fields**:
- **id**: an **int** that uniquely identifies the client.
- **firstName**: a **String** that represents the first name of the client.

- **lastName**: a **String** that represents the last name of the client.
- **yearOfBirth**: an **int** that represents the year of birth of the client.
- **gender**: an **Enum** type**, Gender,** that represents the gender of the client: ***MALE*** or ***FEMALE***.
- **requests:** an **array of Request**s that the client wants to assign to a server.
- **arrivalTime:** an **int** indicating the arrival time of the client in the system.
- **waitingTime:** an **int** indicating for how long the client has been in the waiting line before joining a server queue / or before leaving without being served.
- **timeInQueue**: an **int** indicating how long the client is in a server queue.
- **serviceTime**: an **int** indicating how long the server takes to process the jobs.
- **departureTime**: an **int** indicating at what time the client leaved the system (after being served or left without being served)
- **patience**: an **int** indicating how long the client can stay before leaving the system if not being served.

All fields are **private**. Feel free to add your own attributes, they must be private.

➔ **Getters** and **Setters.**

➔ **Constructors**: Create three constructors for this class. See the signature of the constructors below:

public **Client** (String **firstName**, String **lastName**, int **yearOfBirth**, String **gender**, int **arrivalTime**, int **patience**, Request[] **requests**)

This constructor will initialize all the fields of the object Client. For the fields not provided, they will be initialized as follow:

- **id**: autoincrement from the last id used. The first client will have id=1, the second will have id=2, etc.
- **waitingTime**: 0
- **timeInQueue: 0**
- **serviceTime**: 0
- **departureTime**: 0 (default value)

NB- **arrivalTime**: If a negative value or zero is provided, use the time indicated by the clock of the QueueSystem.

public **Client** (String **firstName**, String **lastName**, int **yearOfBirth**, String **gender**, int **patience**, Request[] **requests**)

This constructor will initialize all the fields of the object Client. For the fields that are not present in the list of parameters, they will be initialized as follow:

- **id**: autoincrement from the last id used. The first client will have id=1, the second will have id=2, etc.
- **arrivalTime**: The time indicated by the clock of the QueueSystem.
- **waitingTime**: 0
- **timeInQueue: 0**
- **serviceTime**: 0
- **departureTime**: 0

public **Client** (String **firstName**, String **lastName**, int **yearOfBirth**, String **gender,** int **patience**)

This constructor will initialize all the fields of the object Client. For the fields that are not present in the list of parameters, they will be initialized as follow:

- **id**: autoincrement from the last id used. The first client will have id=1, the second will have id=2, etc.
- **arrivalTime**: an int indicating the arrival time of the client. If not provided, use the time indicated by the clock of the QueueSystem.
- **waitingTime**: 0
- **timeInQueue**: 0
- **serviceTime**: 0
- **departureTime**: 0

Business rules:

Regardless of the method used, the object state needs to be consistent and have valid values at all times. The validation and business rules are the following:

- arrivalTime cannot be negative number and must be greater or equal to 1.
- departureTime is either 0 (default value) or needs to be greater than or equal to arrivalTime + waitingTime + timeInQueue. We would observe departureTime = arrivalTime + waitingTime + timeInQueue only if the client left without being served.
- The patience value of a client might increase based on the availability of a tv and/or coffee maker in the QueueSystem;
- The client will exit the system if his time in the waitingLine and a server queue exceeds his patience value.
- waitingTime cannot be a negative number.
- timeInQueue cannot be a negative number.

➔ **Other methods**: You need to define the following methods:

- public int **serviceTime**(): returns the time a server takes to process the requests of the client.

- public int **estimateServiceLevel**(): estimates the level of service: a number from 0 to 10. If the client is still in the system, return -1. Otherwise, deduce 1 point from the maximum value for each of these situations:

    - Waiting time > 4
    - Waiting time > 6
    - Waiting time > 8
    - Waiting time > 10
    - Waiting time > 12
    - Time In Queue > 4
    - Time In Queue > 6
    - Time In Queue > 8
    - Time in Queue > 10
    - Time in Queue > 12

    For instance: If the client had to wait 7s in the waitingLine and stayed 10s in the Queue before being served, the service level is 5.

- public String **toString**(): This method can be called to return the object in a human-friendly manner. It returns the following information:

```
Client: <lastName>, <firstName>
** Arrival Time    : 0
** Waiting Time    : 5
** Time In Queue   : 2
** Service Time    : 2
** Departure Time  : 9
** Served By Server: B
** Service Level   : 9
```

The VIPClient class:

This class represents a Client who is a VIPClient. This class implements Prioritizable. If there is no VIPQueue, this client behaves like any other client. Otherwise, the VIPClient may decide to join the VIP queue if necessary (while the other clients cannot except in some exceptionnal situation described later in this document).

You need to implement constructors and methods to instantiate the fields of a VIPClient.

➔ **Fields**:

- **memberSince**: an **int** indicating the year since this client is a VIPClient;

- **priority**: an **int** indicating the level of priority of this client.

➔ **Getters and Setters.**

➔ **Constructors**: Create a constructor for this class. See the signature of the constructors below:

public **VIPClient** (String **firstName**, String **lastName**, int **birthYear**, String **gender**, int **arrivalTime**, int patience, Request[] **requests,** int **memberSince,** int **priority**)

➔ **Other methods**: You need to define the following methods:

- public String **toString**(): This method can be called to return the object in a human-friendly manner. It returns the following information:
  Client: <lastName>, <firstName>

```
** Arrival Time     : 0
** Waiting Time     : 5
** Time In Queue    : 2
** Service Time     : 2
** Departure Time   : 9
** Served By Server: B
** Service Level    : 9
** VIP since        : 2002
** priority         : 3
```

The Request abstract class:

This is an abstract class representing the attributes and the behaviors that any specific requests should implement. It must implement the interface Prioritizable.

➔ **Fields**:
- **description**: a **String** describing the request;
- **priority**: an **int** indicating the priority of a request;
- **difficulty**: an **int** indicating the difficulty of the request;
- **factor**: an **int** indicating the factor by which the difficulty is related to the processing time of the request.
- **startTime**: an **int** indicating the processing start time of the request.
- **endTime**: an **int** indicating the processing end time of the request.
- **completionLevel**: an **int** indicating the completion level of the request (0-100).

- **status**: an Enum **Status** indicating if the state of the request: ***NEW***, ***IN_PROGRESS***, ***PROCESSED***

All fields are **private**.

➔ **Getters and Setters.**

➔ **Abstract methods:**
public int calculateProcessingTime ()

NB
The time required to process a request depends on the difficulty level of this request and on the type of request. Compute this time using the following formula:

- *Information request:*
  Processing time = Request difficulty x factor x numberOfQuestions
- *Returning item:*
  Processing time = Request difficulty x factor x numberOfItems
- *Buying product:*
  Processing time = Request difficulty x factor x numberOfItems

The InformationRequest class:

This class extends Request and represents a request for information.

➔ **Fields**:
- **questions**: an array of String representing the list of questions.

➔ **Getters and Setters.**

➔ **Constructors**: Create a constructor for this class. See the signature of the constructors below:

- public **InformationRequest** (String description, int priority, int difficulty, String[] questions)
  Use those values for the attributes not mentioned in the parameter list:
  - factor: 1
  - status: NEW
  - all other fields has the default value;

The ReturningItems class:

This class extends Request and represents a request about returning some items.

- ➔ **Fields**:
  - ◆ **itemsToReturn:** an array of String representing the list of items to return.

- ➔ **Getters and Setters.**

- ➔ **Constructors**: Create a constructor for this class. See the signature of the constructors below:

  - public **ReturningItems** (String description, int priority, int difficulty, String[] itemsToReturn)
    Use those values for the attributes not mentioned in the parameter list:
    - factor: 3
    - status: NEW
    - all other fields have the default value;

The BuyingProducts class:

This class extends Request and represents a request about purchasing some items.

- ➔ **Fields**:
  - ◆ **itemsToBuy:** an array of String representing the list of items to buy.

- ➔ **Getters and Setters.**

- ➔ **Constructors**: Create a constructor for this class. See the signature of the constructors below:

  - public **BuyingProducts** (String description, int priority, int difficulty, String[] itemsToBuy)
    Use those values for the attributes not mentioned in the parameter list:
    - factor: 2
    - status: NEW
    - all other fields have the default value;

The QueueSystem class:

This class will represent the System. Here is the detailed specification:

➔ **Fields**:

- **clock :** an **int** indicating the time of the system.
- **totalWaitingTime:** an **int** storing the total WaitingTime of the Clients.
- **clientsWorld:** an **array of clients** ready to enter the system.
- **totalClientsInSystem:** an int, stores the number of clients in the system.
- **waitingLineSize:** an **int** indicating the size of the waiting line.
- **waitingLine:** an **array of Clients** indicating the clients in the waiting line.
- **tvInWaitingArea:** a **boolean** indicating whether or not a TV is available in the waiting area.
- **coffeeInWaitingArea:** a **boolean** indicating whether or not a coffee maker is available in the waiting area.
- **queues:** an **array of Queues** of the system.

There will be exactly one copy of these fields in memory.

➔ **Getters and Setters.**

➔ **Constructors**: Create one constructor for this class as follows.

- public **QueueSystem** (int **waitingLineSize**, boolean **tvInWaitingArea**, boolean **coffeeInWaitingArea**):
  - clock is set to 0: the clock value before the start of the simularion starts.
  - Initialize the array of Clients: waitingLine based on the waitingLineSize.
  - Set tvInWaitingArea and coffeeInWaitingArea to the values provided as arguments.

N.B- If there is a TV in the waiting area, the patience of each client is increased by 20. Also, if there is a coffee machine available in the waiting area, the patience of each adult client is increased by 15.

➔ **Other methods**: You need to define the following methods:

public void **increaseTime**(): advances the clock by 1 unit of time. The function will also make all changes necessary to reflect the progress in all the queues: i.e set objects attributes to its correct values.

- Increase the clock;
- Check if server have processed the requests.
  - Make sur all clients being served have their request being processed: update the request attributes to their correct value based on the time on the clock;

        - Update each request status, if necessary. Make sure the request attributes are updated.
        - Check if the client being served has no more requests and remove this client from the system: the client is not being served anymore and the departure time is the time when all requests are processed. This client will be added to the clientsHistory of the Queue.
    - Once all the servers from the queues have been checked, move a client from the queue to the server (FIFO). If a queue is empty, one client from the waiting line must go to an available server. The selection criteria is described below.
    - From the waitingLine, a client enters server queue if there is a spot available;
    - From the waitingLine to a serverQueue:
        - Apply the FIFO policy. If many clients enter the system at the exact same time, priority is given to the older client. Then, priority to the client with tasks requiring less processing time. Then, alphabetic order of last name. Then, alphabetic order or first name. Then the smaller id.
        - For the VIP Client, priority is given to the client with a higher priority value. Then the one who has been a member for a longer period. Then we applied FIFO. Then, apply the same criteria described above.
    - The client will join the queue with the most available spots. In case of equality, the client will join the closest one (the queue having the smaller index in the array of queues of the system. i.e queues[3] is closer to the clients than queues[4]).
    - Check if some clients in the world should enter the system: use the arrivalTime and the clock value for this purpose. In case more than one client should enter the system at a given time (ref. clock), consider the ascending order of their id.
    - From the world, a client entering the system will go to waitingLine if he cannot go to a server queue. This client will join a server queue directly if one is available.
    - Remove Client who needs to leave the system without being served (compare the waitingTime + queueTime with the patience value. The initial patience value might have been increased due to the presence of a TV or coffee maker).

    - N.B: A client can join the VIPQueue if no VIPClient is in the system.

public void **increaseTime**(**int** time): advances the clock by *time* units of time. This method will reuse the code from the method defined earlier.

public int Client[] **getClientsBeingServed**(): returns the list of clients being served.

public String **toString**(): returns a String representing the actual state of the system. Each client is represented by the estimated processing time (see example in yellow, clients in a server queue). Same representation for the clients in the waiting line (see example in blue). For the client being served, it is represented by the estimated remaining processing time (see example in green, client being served)

Format and example of output on the console:

```
[Queue:1][09]-----[13][21][12][08][03]
[Queue:2][03]-----[22][11][10][18][13]
[Queue:3][19]-----[21][22][11][08][03]
[Queue:4][12]-----[13][31][10][28][13]
[Queue:5][03]-----[41][24][14][08][05]
[Queue:6][05]-----[13][21][10][38][04]
---
[WaitingLine]-[11][21][10][08][03][11][21][10][08][03]
```

public String **toString**(boolean showID): if showID is true, returns a String representing the actual state of the system, representing each client by their id. Otherwise, the method returns the representation described above.

Format and example of output on the console:

```
[Queue:1][01]-----[07][13][19][25][31]
[Queue:2][02]-----[08][14][20][26][32]
[Queue:3][03]-----[09][15][21][27][33]
[Queue:4][04]-----[10][16][22][28][34]
[Queue:5][05]-----[11][17][23][29][35]
[Queue:6][06]-----[12][18][24][30][36]
---
[WaitingLine]-[37][38][39][40][41][42][43][44][45][46]
```

The Queue class:

This class represents a queue in the System. The queue has a size limit. A client can join a queue if the queue is not full, and the server is busy. If the server is not busy, and the queue is empty, the client goes to the server directly.

- ➔ **Fields**:
  - **serverName:** a **String**, represents the name of the server.
  - **queueSize:** an **int**, represents the size of the queue.

- **clientBeingServed:** a **Client**, references the client being served.
- **requestInProgress:** a **Request**, references the request being processed.
- **processingStartTime:** an **int**, represents the start time of the process.
- **clientsHistory:** an **array of Clients**, stores all the clients served by this server.
- **clientsInQueue:** an **array of Clients**, stores all the clients in the queue. (The client being served is not in the queue.)

➔ **Getters and Setters.**

➔ **Constructors**: Create one constructor for this class as follows.

- public **Queue** (String **serverName**, int **queueSize**)

➔ **Other methods**: You need to define the following methods:

public String toString(): return the String representation of the queue. For instance, the first queue in the array of queues may have this representation. (Client ID 09 is being served / Client ID 13 is in this Queue and will be served next.)

```
[Queue:1][09]-----[13][21][12][08][03]
```

public String toString(boolean showID): if showID is true, returns a String representing the actual state of the queue, representing each client by their id. Otherwise, the method returns the representation described above. (01 time unit remaining to finish processing Client 09 requests / Client ID 13 is in this Queue and will be served next: The estimated time required to process all his/her request is 07)

Format and example of output on the console:

```
[Queue:1][01]-----[07][13][19][25][31]
```

The VIPQueue **class:**

A VIPQueue is a Queue that will be used mostly by VIPClients. In some situations, this queue might accept clients who are not VIPClient.

➔ **Fields**:

- **acceptingAnyClients:** a **boolean** indicating if any client can enter this queue.

➔ **Getters and Setters.**

➔ **Constructors**: Create two constructors for this class as follows.

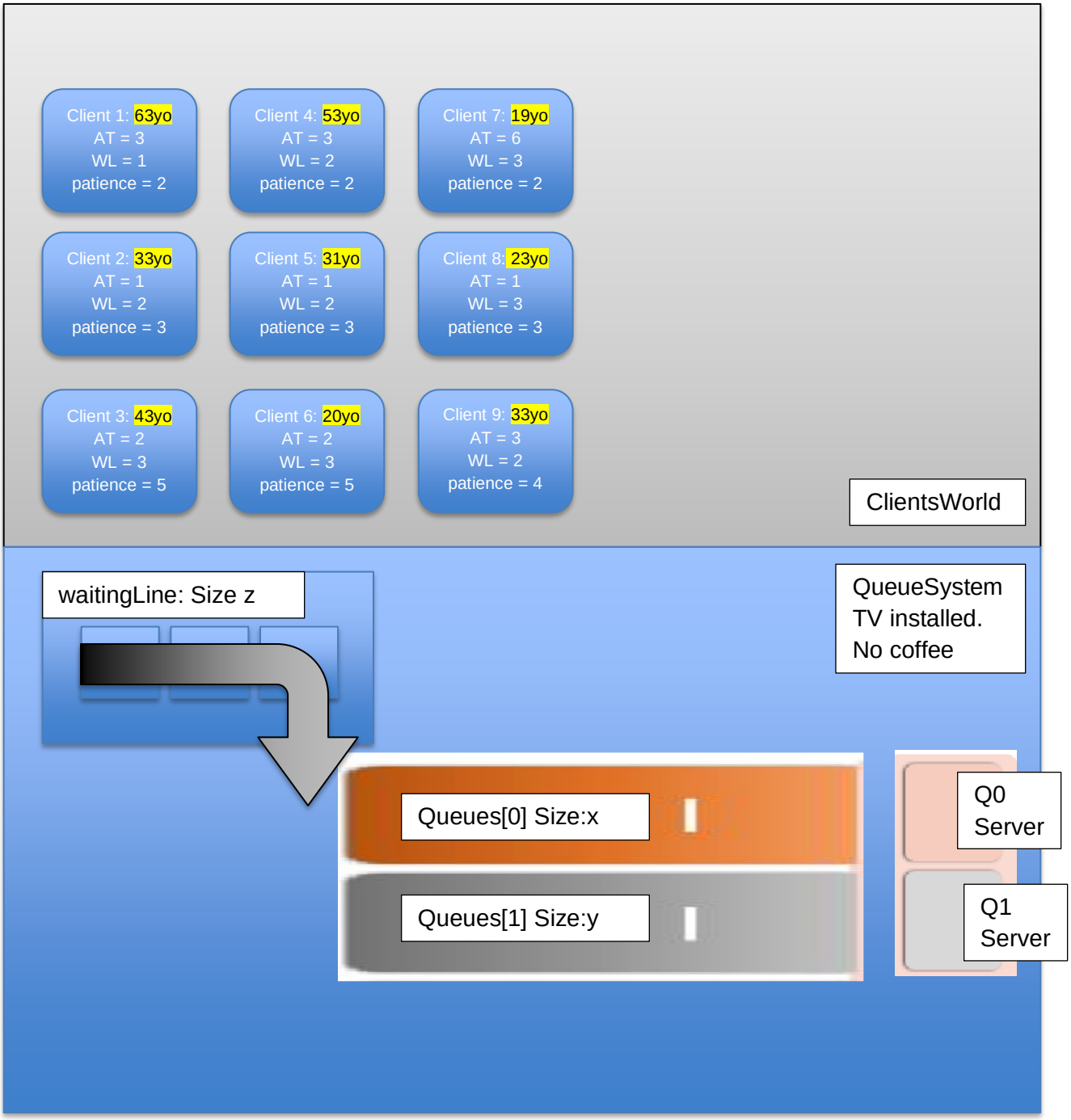
- public **VIPQueue** (String **serverName**, int **queueSize,** boolean **acceptAnyClients**)

- public **VIPQueue** (String **serverName**, int **queueSize**)

State of the world: initial

WL: Represent the time that is required to process the client requests.

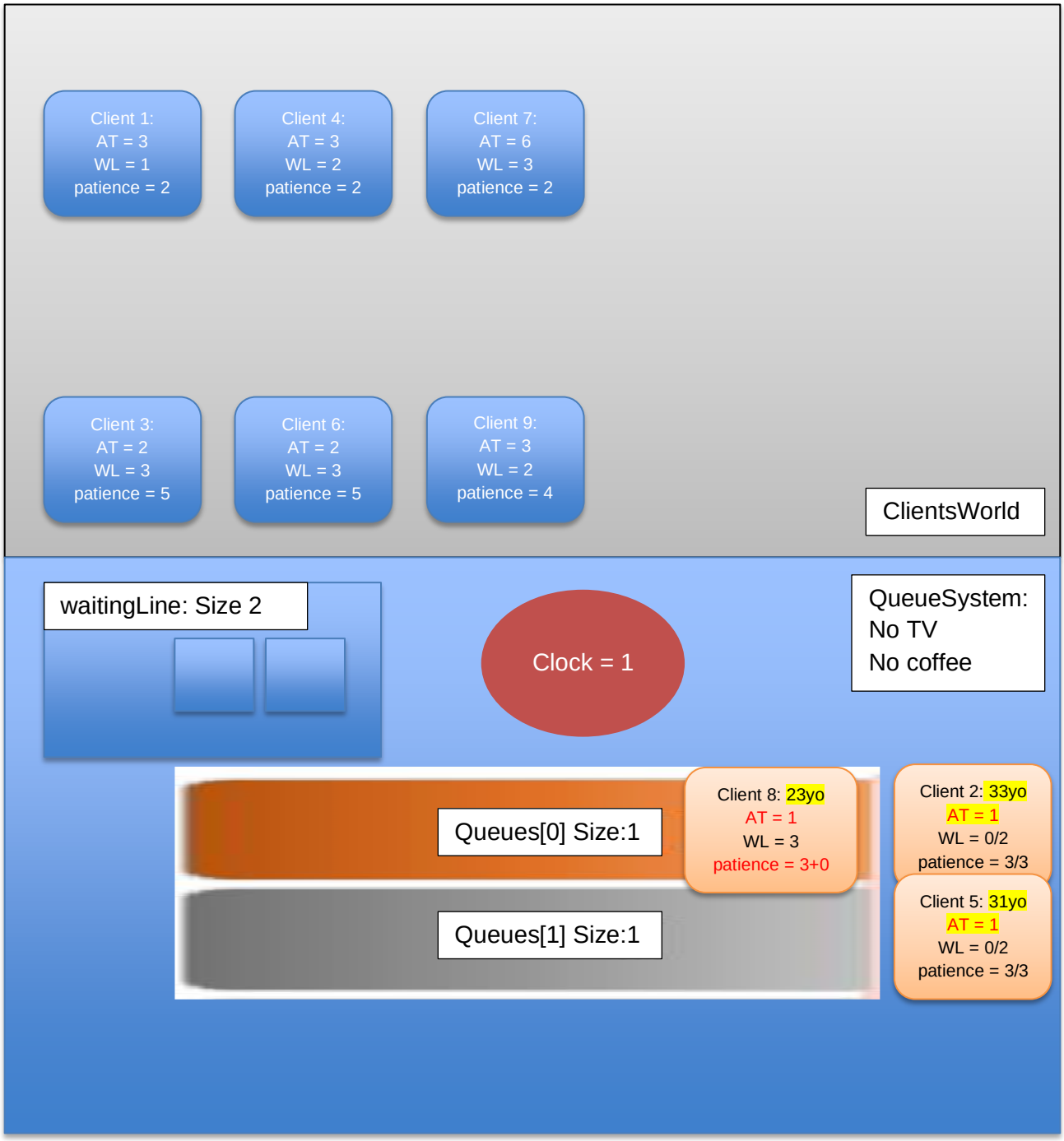
Age Information Added



Clock = 0

After calling: increaseTime()

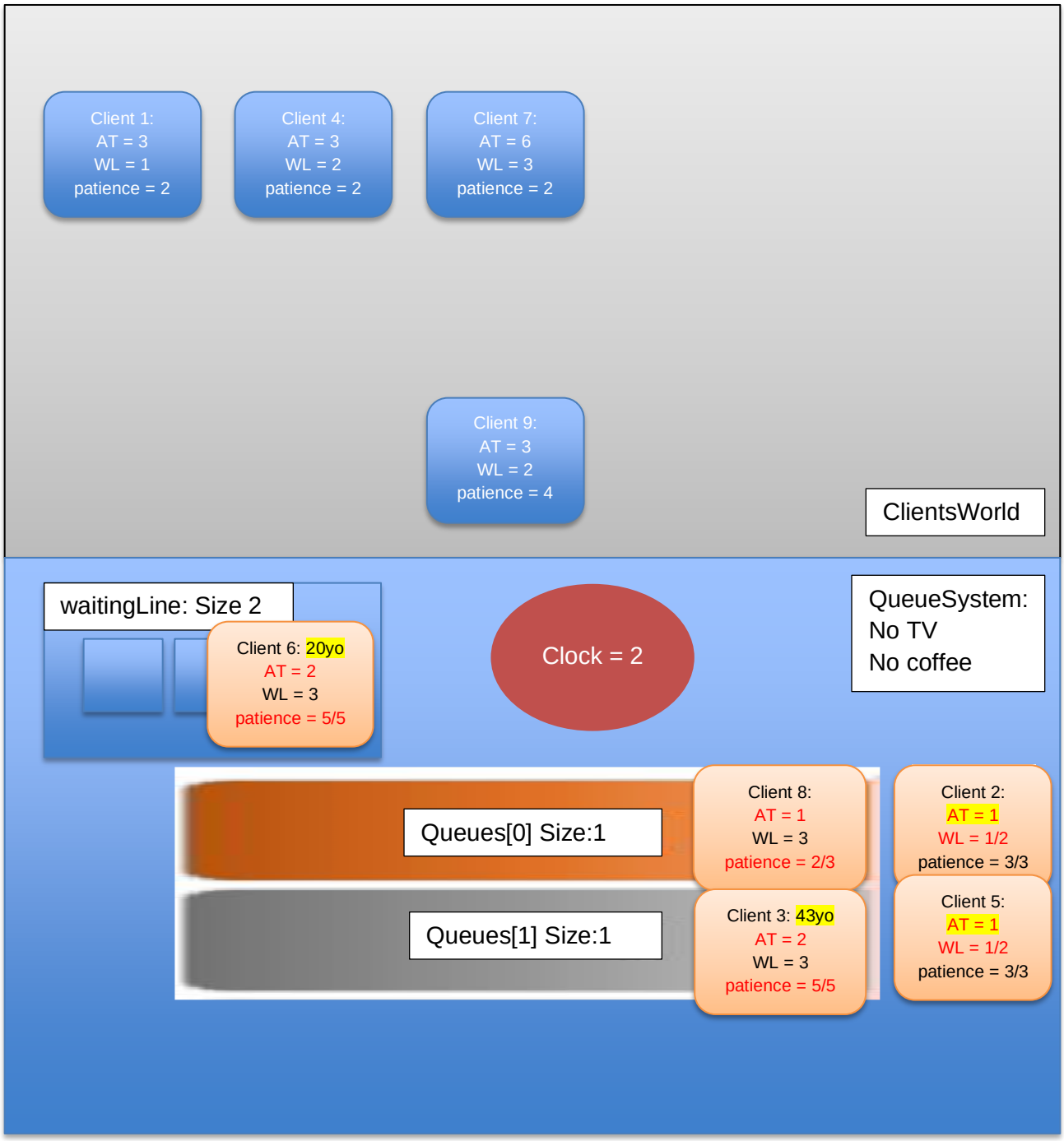
State of the world:



Clock = 1

After calling: increaseTime()

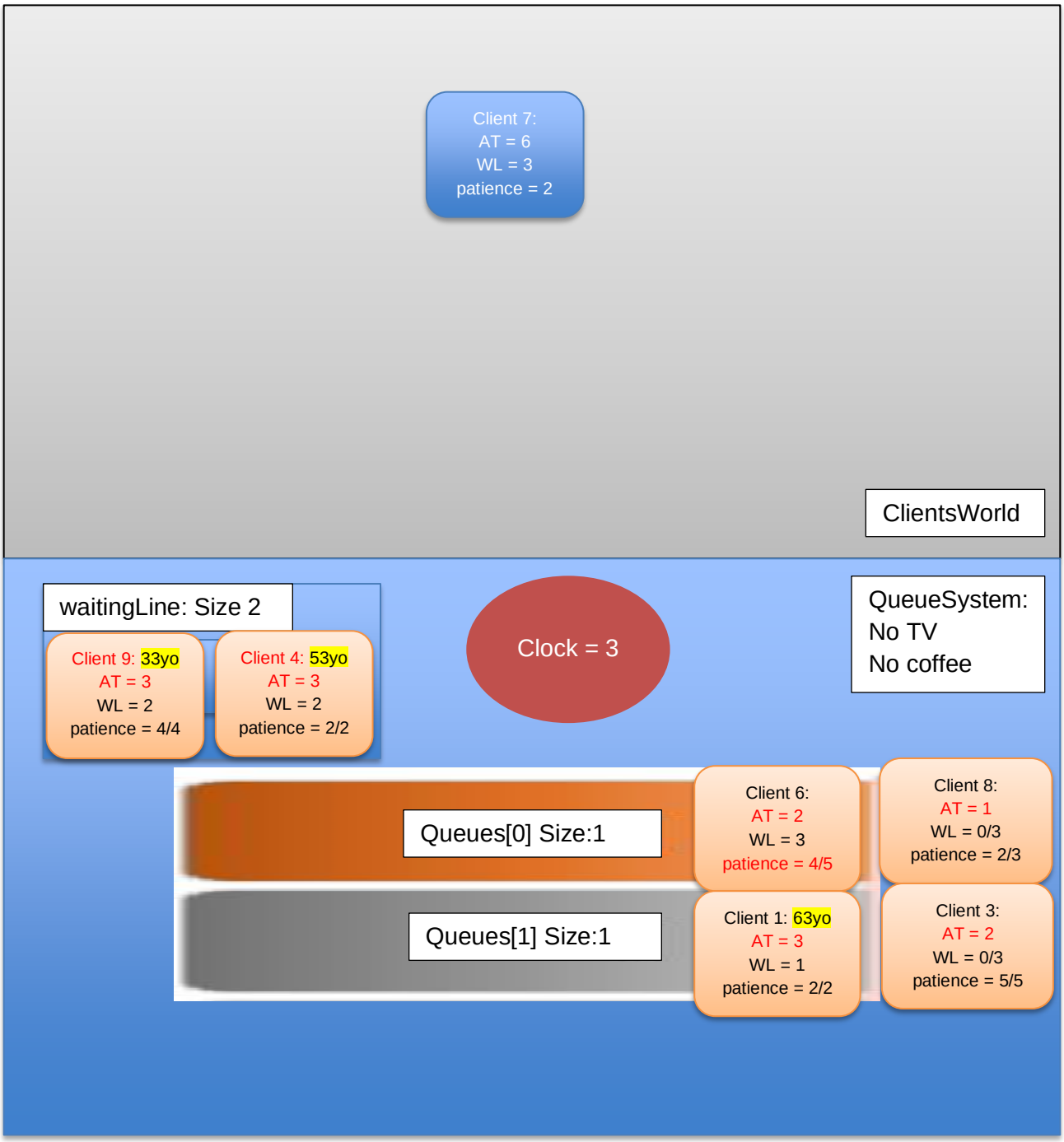
State of the world:



Clock = 2

After calling: increaseTime()

State of the world:



Queues[0] ClientsHistory

Client 2:
AT=1 / DT=3
WL = 2/2
patience = 3/3

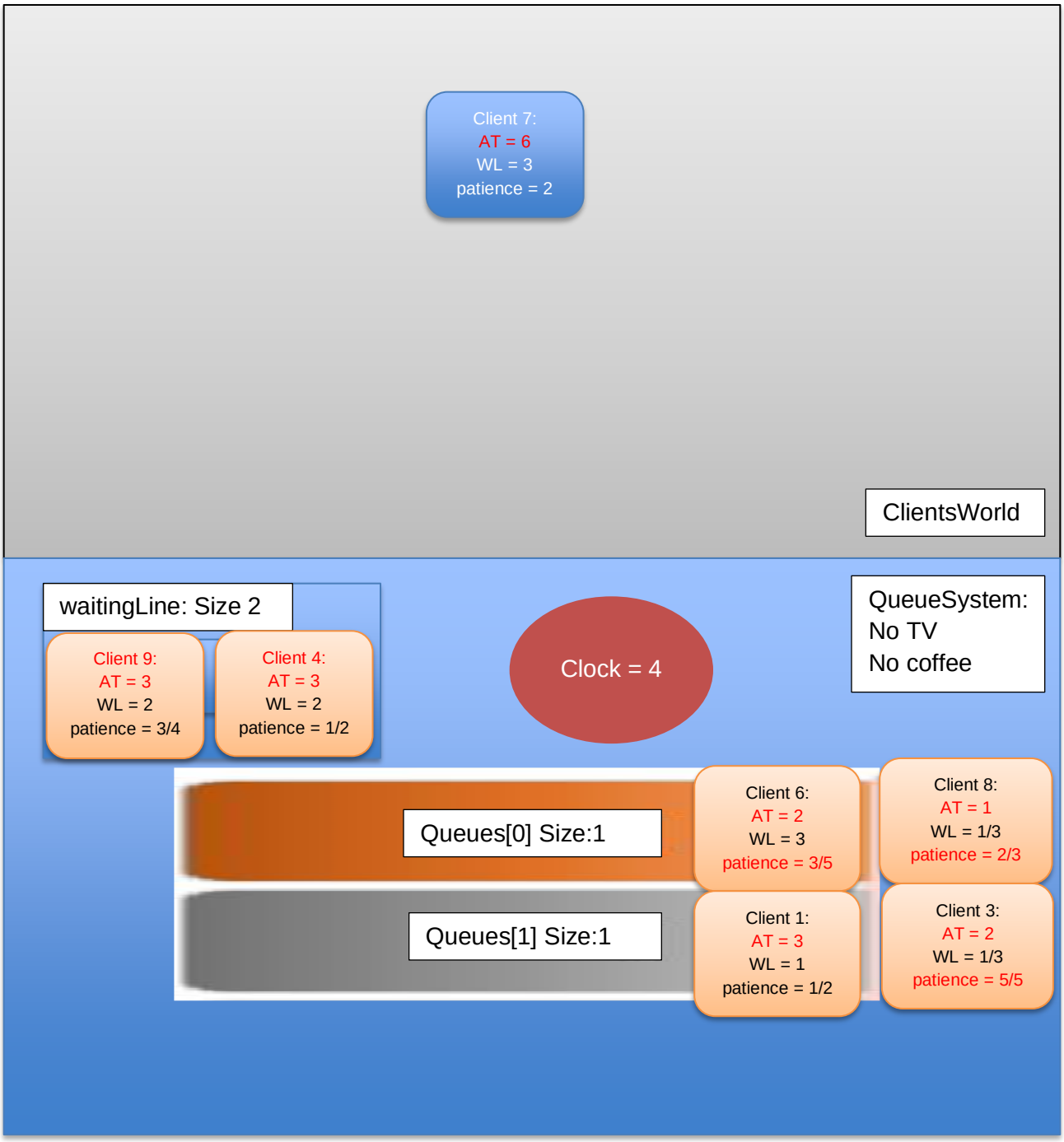
Queues[1] ClientsHistory

Client 5:
AT = 1 / DT=3
WL = 2/2
patience = 3/3

Clock = 3

After calling: increaseTime()

State of the world:



Queues[0] ClientsHistory

Client 2:
AT=1 / DT=3
WL = 2/2
patience = 3/3

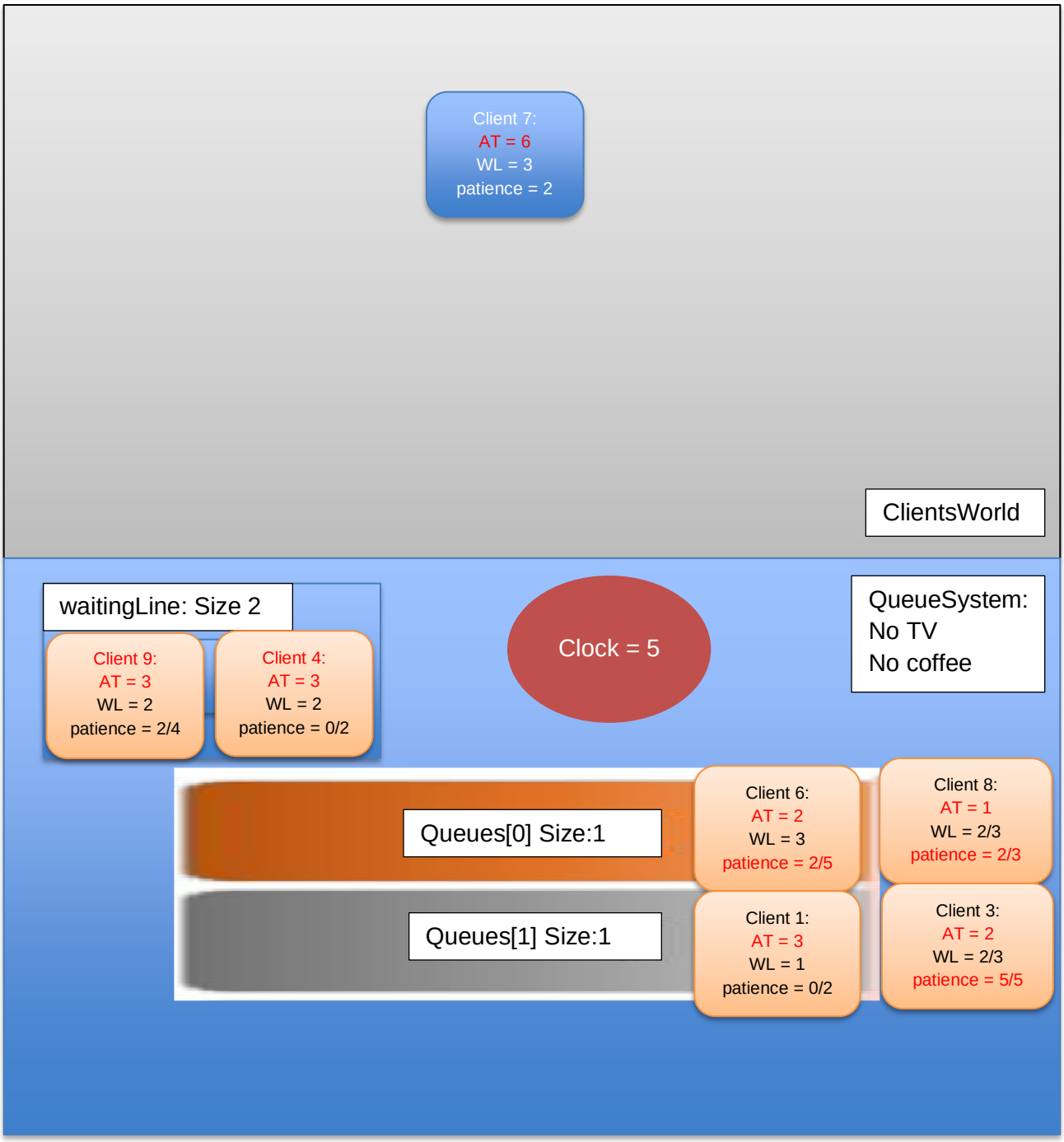
Queues[1] ClientsHistory

Client 5:
AT = 1 / DT=3
WL = 2/2
patience = 3/3

Clock = 4

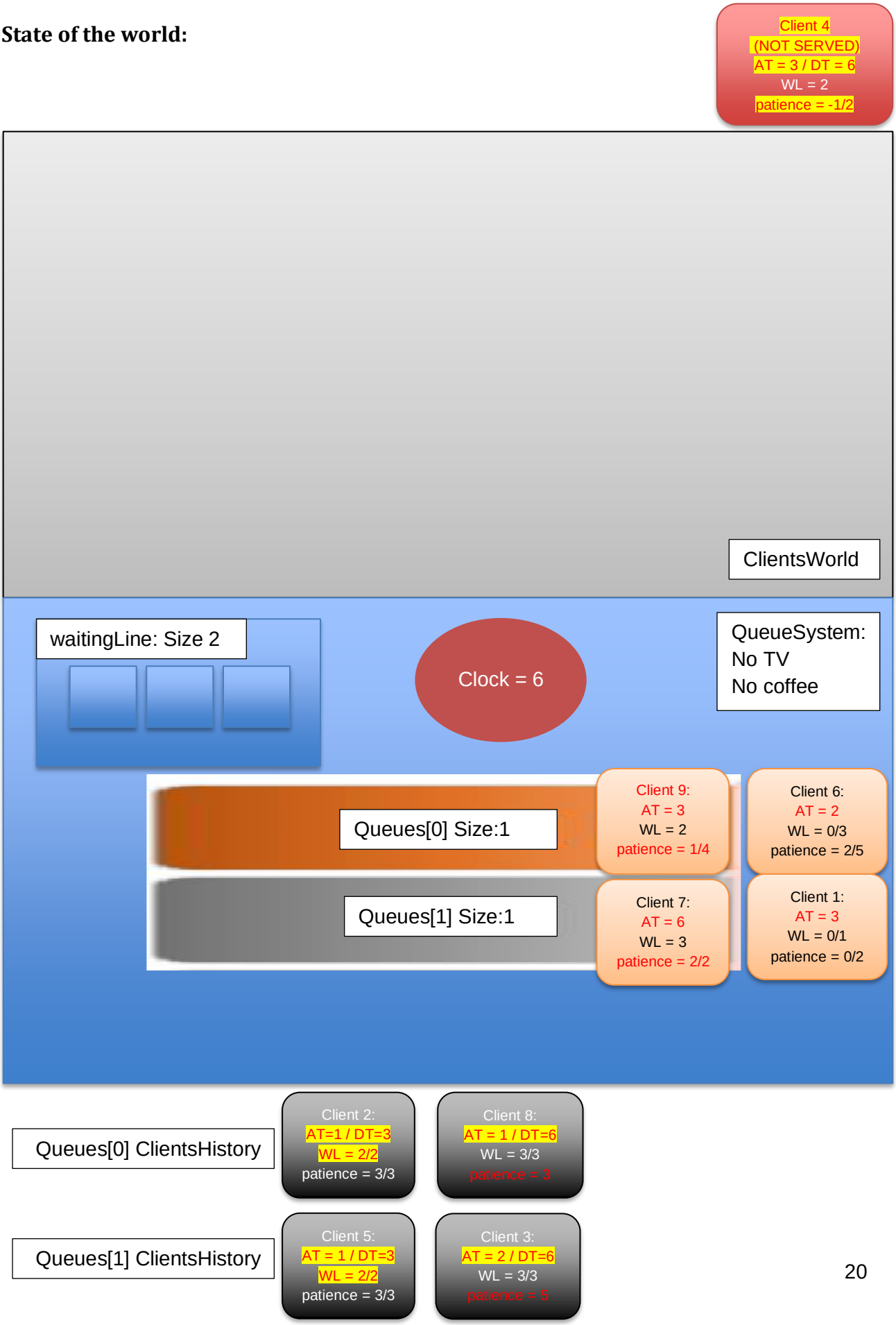
After calling: increaseTime()

State of the world:



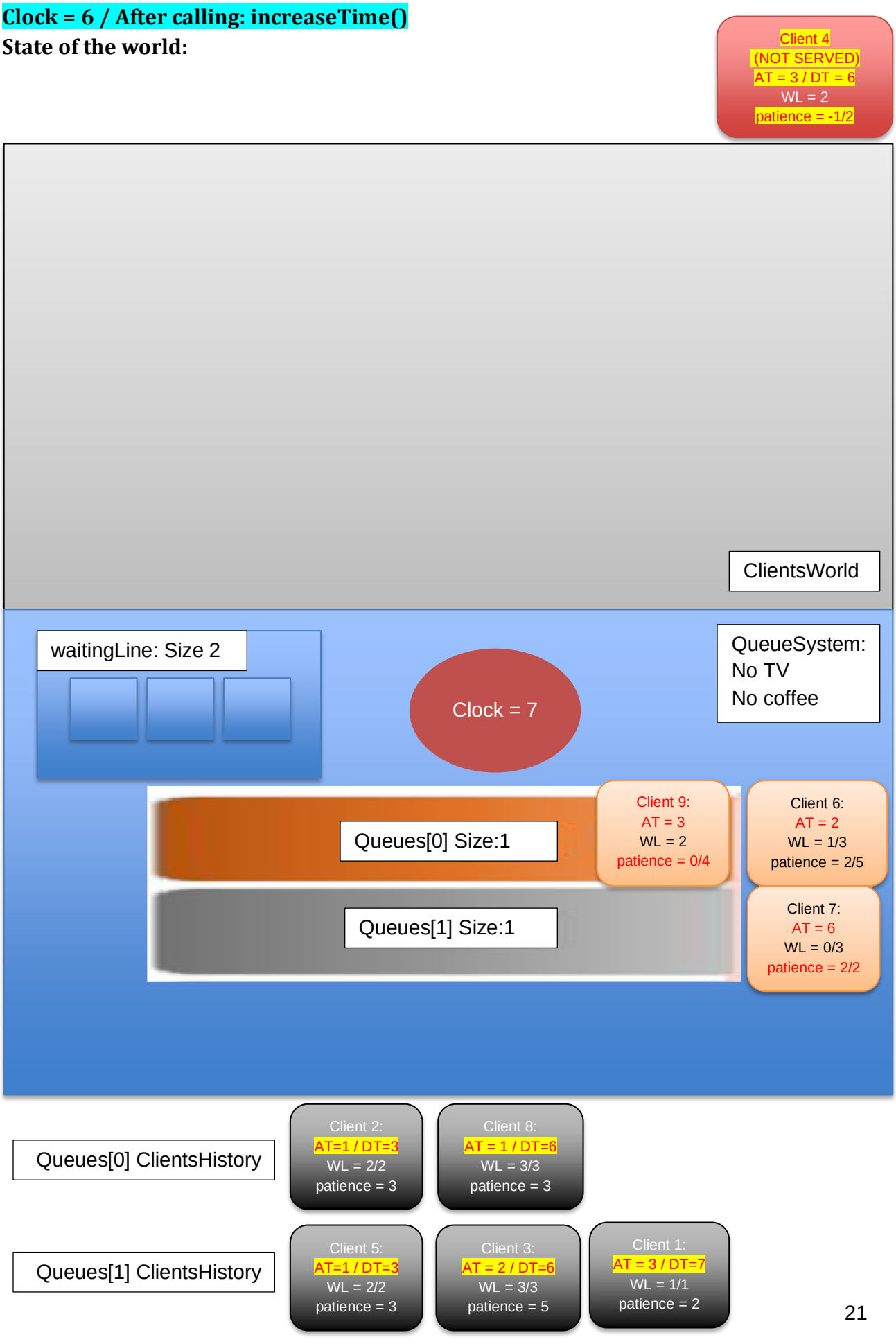
Clock = 5 / After calling: increaseTime()

State of the world:



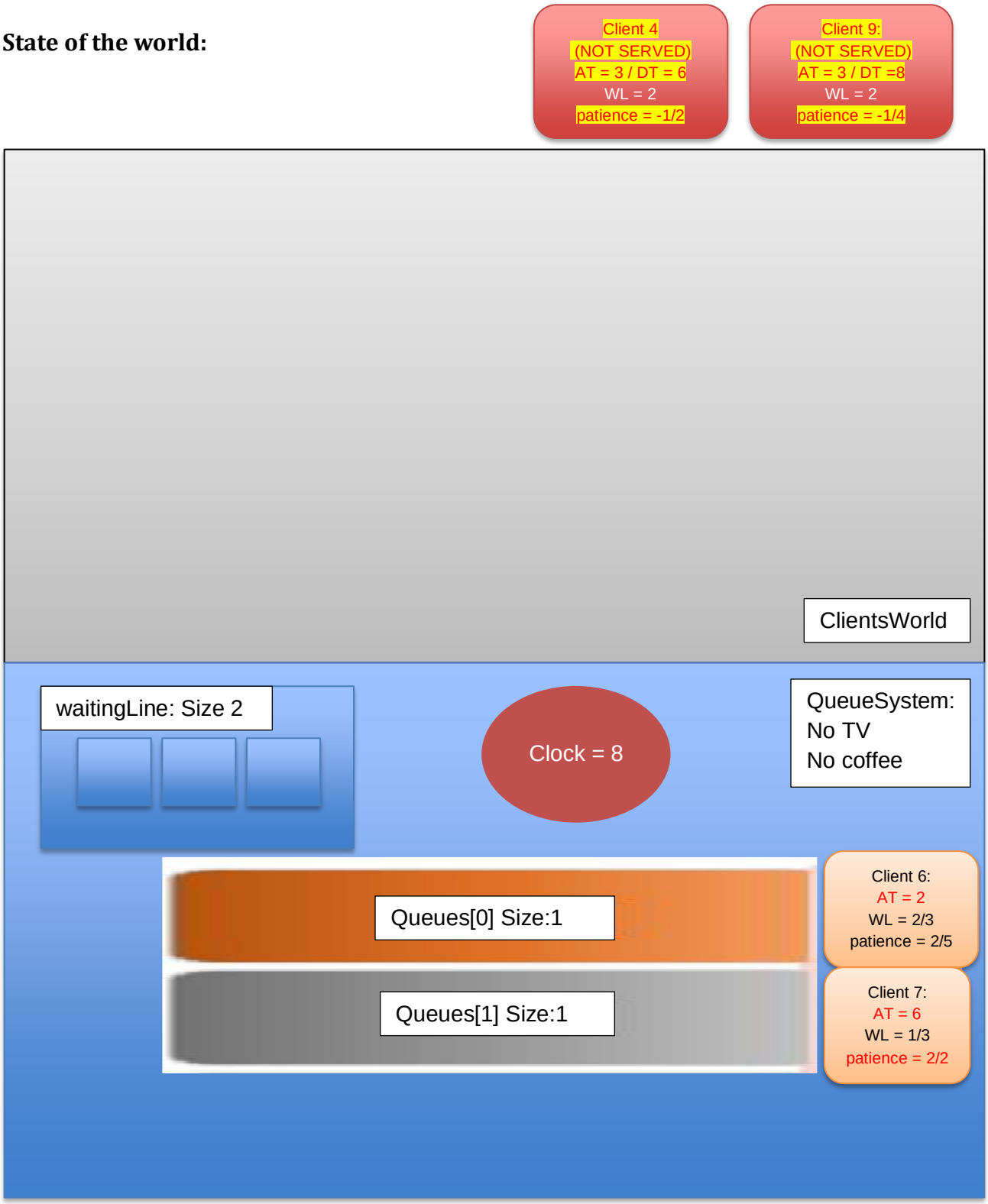
Clock = 6 / After calling: increaseTime()

State of the world:



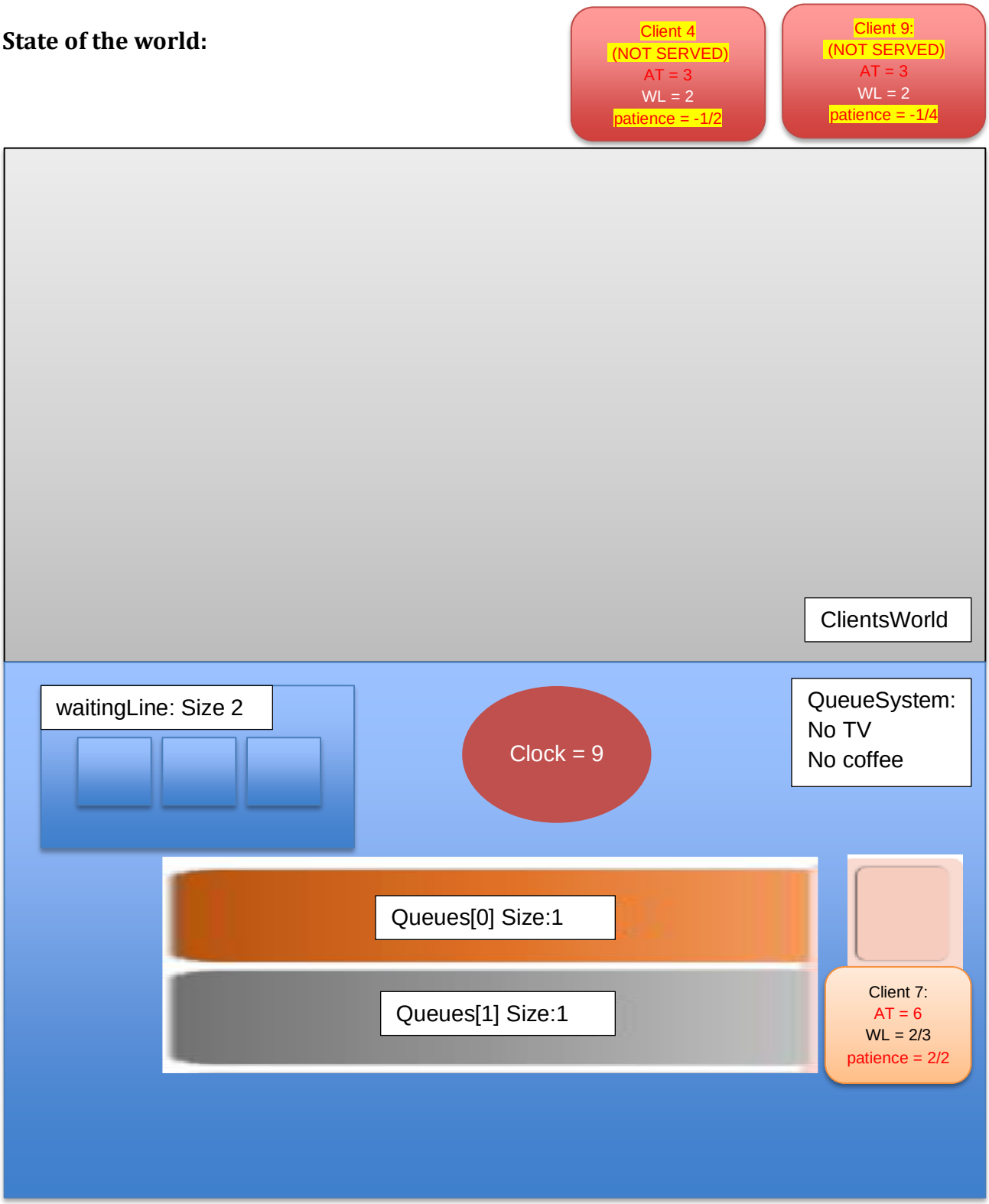
Clock = 7 / After calling: increaseTime()

State of the world:



Clock = 8 / After calling: increaseTime()

State of the world:



Clock = 10 / After calling: increaseTime()

State of the world: