Use RMI to implement (BSS) Birman-Schiper-Stephenson protocol for causal ordering

Birman Schiper Stephenson Protocol

This algorithm is used to maintain the causal ordering of the messages i.e. the message which is sent first should be received first. If send (M1)–>send(M2) then for all processes which receive the messages M1 and M2 should receive M1 before M2.

Features :

•Broadcast based messaging.
•Size of the messages are small.
•More no. of messages are sent.
•Limited state information.

Key Points :

•Each process increases its vector clock by 1 upon sending of messages.
•Message is delivered to a process if the process has received all the messages preceding to it.
•Else buffer the message.
•Update the vector clock for the process.
•**Reference :**

- Process: Pi
- Event: eij , where i:process is number & j: jth event in ith process.
- Tm:vector time stamp for message m.
- Ci vector clock associated with process Pi; jth element is Ci[j] and contains Pi‘s latest value for the current time in process Pj

•Protocol : Pi sending a message to Pj –

- Pi increments Ci[i] and sets the timestamp tm = Ci[i] for message m.

•Pj receiving a message from Pi –

- When Pj, j != i, receives m with timestamp tm, it delays the message's delivery until both these conditions are met:

•Cj[i] = tm[i] - 1; and

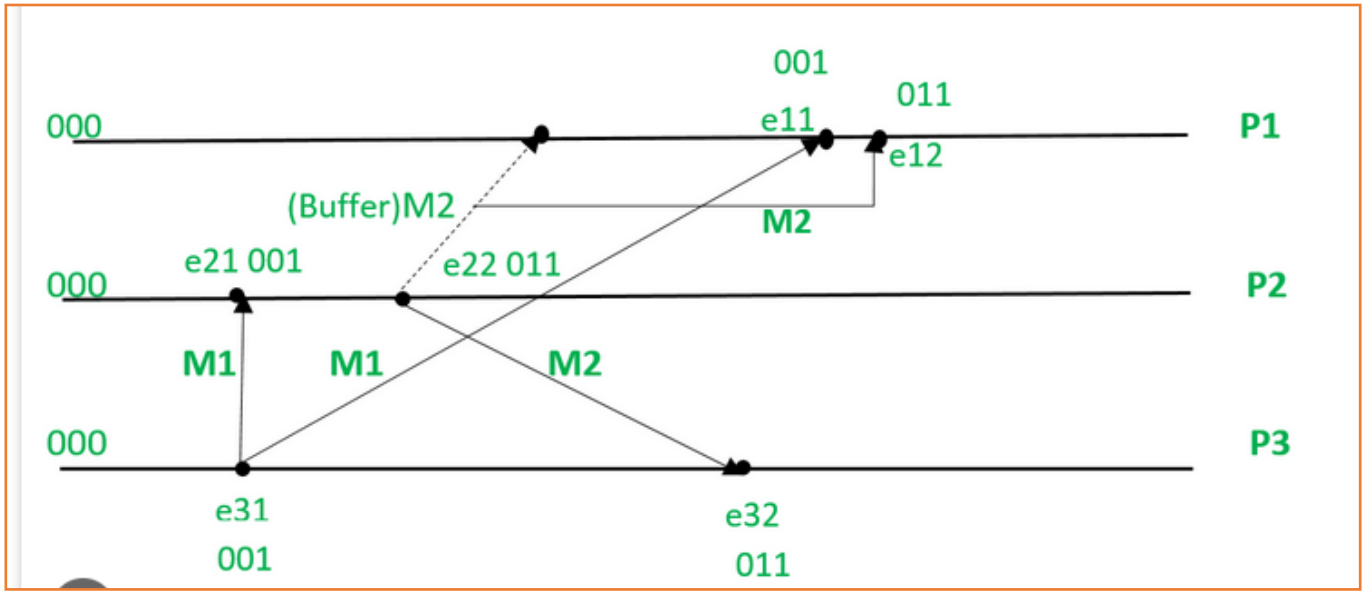
•for all k <= n and k != i, Cj[k] >= tm[k].

- When the message is delivered to Pj, update Pj's vector clock.
- Check buffer to see if any can be delivered.

Problem Statement:-

To Implement **Birman-Schiper-Stephenson (BSS) protocol for causal ordering** using RMI (Remote method invocation).

- Initial state for all the processes are 000.
- M1 is broadcasted from P3 to P1 and P2. e31 updates the vector clock to (001) and sends P1 and P2.
- P2 accepts the M1 with timestamp (001) because when it compares it with its initial timestamp i.e. (000) it finds that M1 is the 1st message it is receiving.
- Now we consider that before M1 could reach P1, P2 sends M2 to P1 and P3 with time stamp (011).
- P1 could not accept M2 because upon comparing the timestamp of M2 with its initial timestamp a discrepancy is found because P1 has no message with timestamp (001) received earlier, so M2 is stored in buffer.
- Now M1 is received by P1 and accepted.
- M2 is removed from buffer and accepted by P1.
- M2 is accepted by P3 as there is no discrepancy in the time stamp.



Development method:

1. Requirement – Use RMI to implement **Birman-Schiper-Stephenson (BSS) protocol for causal ordering** using RMI
   Specification:

2. Design – High level design, detailed level design.
   There are two files, one for process act as Server and ready to accept request from client. The second for TokenManager act as server to support token for Server
Server file :- ILab2Process and ILab2BSSManager
Client file :- Lab2ProcessClient

3.Coding/Build/Implement
   Used Python library PyPro5 to communicate between client and Server. The Build is in Window platform and console-based applications.

4. Testing
   Used Unit testing approach such as input data and expected output for each method.

5. Go Live
   There are installable instructions to run the client and server in same machine or different machine.

Agile Development method:

1. Requirement – Implement Use case stories/Use case diagram
   Actor: End user
   functionalities:- End user request from server using the client interface on Window console application.

2. Design :- Implement Sequence diagram, Class diagram, component diagram
   There are two interface one for server and second for client. Similarly, there are two classed one for server and second for client.

3. Coding/Build/Implement – Implement class diagram into the pseudo code. Used Python library and code to implement the BSS algorithm

4. Testing – Implement test cases such as unit test cases, integration test cases, golive test, user test.

5. Go Live – Implement component diagram and use case diagram.

Components:
1. Process Interface: This interface defines methods for processes to interact with the BSS protocol.
Methods
can include:
• send(message): Sends a message to other processes.
• deliver(message): Delivers a causally ordered message to the application layer.
• getToken(): Attempts to acquire the BSS token (optional, can be implicit in send)

Response:-

Used Python based code to implement.
File name: iLab2Process.py

Components:

2 Process Implementation: This class implements the Process interface. It maintains:

- A local queue for holding received messages.
- A vector clock to track message causality.
- A reference to a BSSManager object (explained below).
- It implements methods from the interface:

o send(message): Increments the local vector clock, adds the message with the current clock, and calls BSSManager.send(message).

o deliver(message): Processes the message based on application logic.

o getToken (optional): Implements logic to acquire the token from BSSManager (e.g., using RMI calls).

Response:-

Used Python based code to implement.

File name: iLab2Process.py

Components:

3. BSSManager Interface: This interface manages the BSS protocol logic. Methods can include:

- send(message): Receives a message from a process and performs BSS protocol actions.
- releaseToken(): Releases the token (optional, can be implicit in send).

Response:-

Used Python based code to implement.

File name: iLab2BSSManager.py

Components:

4. BSSManager Implementation: This class implements the BSSManager interface. It maintains:

- The current token holder (process ID).
- A queue for holding messages waiting to be causally ordered.
- It implements methods from the interface:

o send(message): Updates the local vector clock based on the message clock.

 If the process is the token holder, it dequeues messages from the queue in causal order (based on vector clocks) and delivers them to their respective processes using RMI calls to deliver.

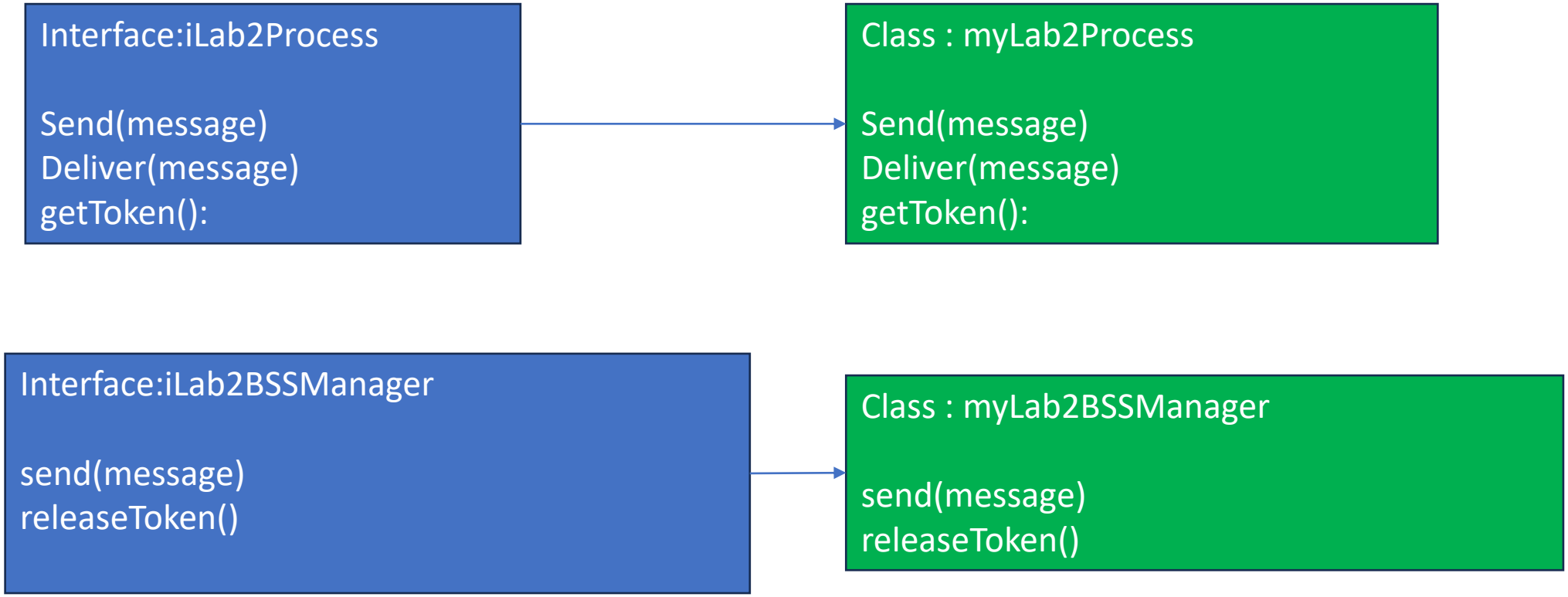
 If the process is not the token holder, it enqueues the message in the waiting queue.

o releaseToken (optional): Updates the token holder based on a pre-defined order (e.g., round-robin)

Response:-

Used Python based code to implement.

File name: iLab2BSSManager.py


Interface:iLab2Process
Send(message)
Deliver(message)
getToken():
Class : myLab2Process
Send(message)
Deliver(message)
getToken():
Interface:iLab2BSSManager
send(message)
releaseToken()
Class : myLab2BSSManager
send(message)
releaseToken()

Implementation Steps:

1. Develop the interfaces and classes mentioned above.
   Implemented using Python.

2. Implement RMI functionalities:

- Use UnicastRemoteObject to create remote objects for Process and TokenManager.
- Use the Naming class to register the TokenManager object on the RMI registry.

Response:

Used Python library and framework Pyro5 to create UnicastRemoteobject for process and token manager.

Used the Python library and framework Pyro5 to register the Process and TokenManger object.

3. Process Logic:

- Each process creates its own Process object and references the shared iLab2BSSManager object.

Implemented in iLab2Process.py.

- Processes use send to send messages. This triggers updates to the local vector clock and sends the message with the clock to the BSSManager.

Implemented in iLab2Process.py

Implementation Steps:

- The BSSManager maintains a waiting queue for messages and a current token holder.

Implemented in Ilab2nBSSManager.py method

When a process receives the token (implicitly or explicitly), it dequeues messages from the waiting queue based on their vector clocks (ensuring causal ordering).

Implemented in Ilab2BSSManager.py method

The BSSManager uses RMI to deliver causally ordered messages to the deliver method of the corresponding Process objects

.

Implemented in Ilab2Process.py method

Benefits:

- Distributed causal ordering: RMI facilitates communication and message ordering across processes.

Implemented in Ilab2BSSManager.py and ILab2Process.py

Limitations:

- Performance overhead: RMI introduces overhead compared to direct communication methods.

Response:-

Used Python based code to trace performance. It is client – server-based implementations on windows platform. Performance is good.

- Complexity: The protocol involves managing the token and maintaining message queues.

Response:-

Used Python library and framework to manage the token and message queues.

User will have proper protocol and recommended practice on windows OS to install pypro5 library and python to run the modules of server and clients. So, Message overhead handles properly with Python code.

- Scalability: Performance might be impacted with a large number of processes.

Window OS with python handles large number of process and performance is good.

Additional Notes:

- This is a high-level overview. The actual implementation will involve exception handling,

Response:-

Used try, except and finally block to handle exception. Implemented using Python try .. block
Used user based natural language English with proper function name to provide proper information.
e.g.

```
try:
            # if message is empty then BSSManager.send method will return Token Value
            self.BSSManagerMessage = self.BSSManager.send ("")

        except Exception as error:
            print("getToken(): Process getToken failed.")
            print (error)
        finally:
            print("getToken(): Process getToken completed successfully.")
        return self.BSSManagerMessage
```

Additional Notes:
serialization of objects for RMI communication, and thread synchronization for concurrent access to queues and sequence numbers.

Serialization, also known as **marshaling**, is the process of translating a piece of data into an interim representation that's suitable for transmission through a network or persistent storage on a medium like an optical disk. Because the serialized form isn't useful on its own, you'll eventually want to restore the original data. The inverse operation, which can occur on a remote machine, is called **deserialization** or **unmarshaling**.

Python ships with the following modules in the standard library, which provide binary data serialization formats for different purposes:

pickle: Python object serialization
marshal: Internal object serialization
shelve: Python object persistence
dbm: An interface to Unix databases

Additional Notes:
Thread synchronization for concurrent access to queues and sequence numbers.

Used Multithreading based python library to thread synchronization. Thread.join method used to synchronization of thread.

As we know we are using window OS based threads to communicate between client and servers. Server will create threads when request will come from client and destroy thread when client request finished.

The server thread synchronizes with windows OS thread for access local memory and IO operations with main processors.

We have used Threading.lock method when process access Critical section and release lock when exit from the critical section.

Additional Notes:
Consider using existing RMI libraries and frameworks for easier implementation.
Response:--
Implemented with Python on window OS. As we have used python library and python tools such as pypro5 library and python compiler, loader, assembler and runner to execute the server in one process and client to execute in multiple process. So we have used Python based RMI library and Python based frameworks to implement alogrihtm.

 The BSS protocol can be further optimized for specific use cases.
Response:-
The BSS protocol can be optimized in the use case such as,

-- Real-Time Collaborative Applications (Ensuring the correct sequence of user actions and enhanced user experience with accurate and synchronized updates.)

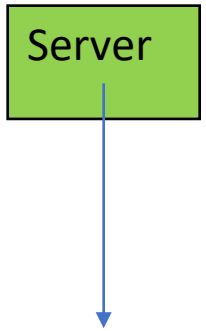
-- Distributed Databases (Maintaining transaction order for consistency across nodes and improved data integrity and reliability.)

-- Multiplayer Online Games (Synchronizing player actions across different servers and smooth and fair gameplay experience.)

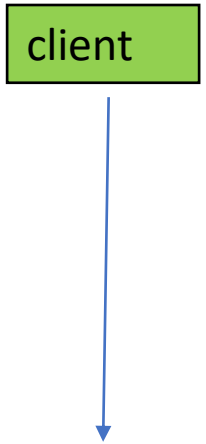
-- Decentralized Finance Platforms (Managing time-sensitive transactions and smart contract execution order and increased trust and efficiency in financial operations.)

To implement BSS using RMI with Python's Pyro library, you can follow these steps.

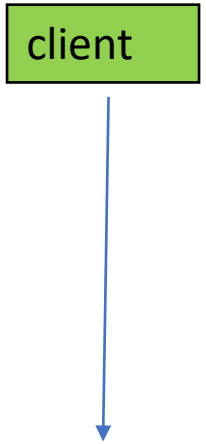
1. **Install Pyro (**pip install Pyro5)
2. **Remote Process (Server) (**file named iLab2process.py and iLab2BSSManager.py)
3. **Client Process (**file named iLab2ProcessClient.py)
5. **Run the Server** (python iLab2process.py)
6. **Run the Client** (python iLab2ProcessClient.py)



1.
Run Lab2process , BSSManager and
Copy the server URI

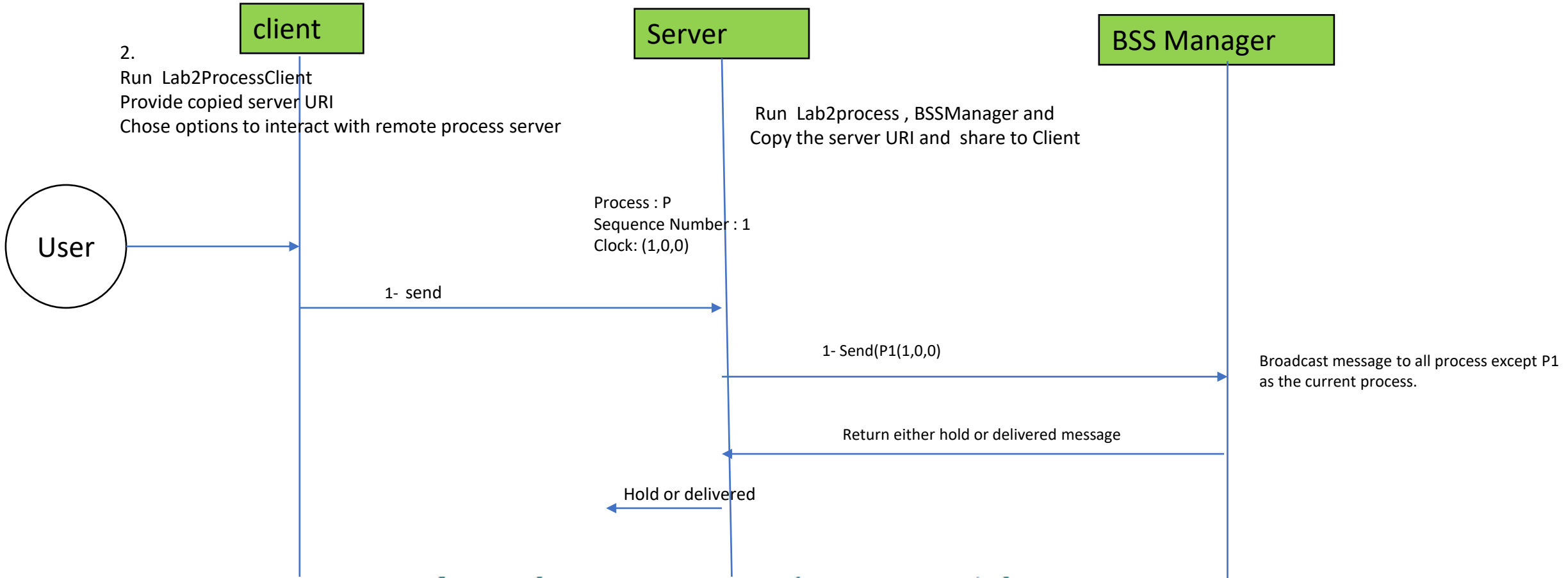


2.
Run Lab2ProcessClient
Provide copied server URI
Chose options to interact with remote process server



3.
Run Lab2ProcessClient
Provide copied server URI
Chose options to interact with remote process server

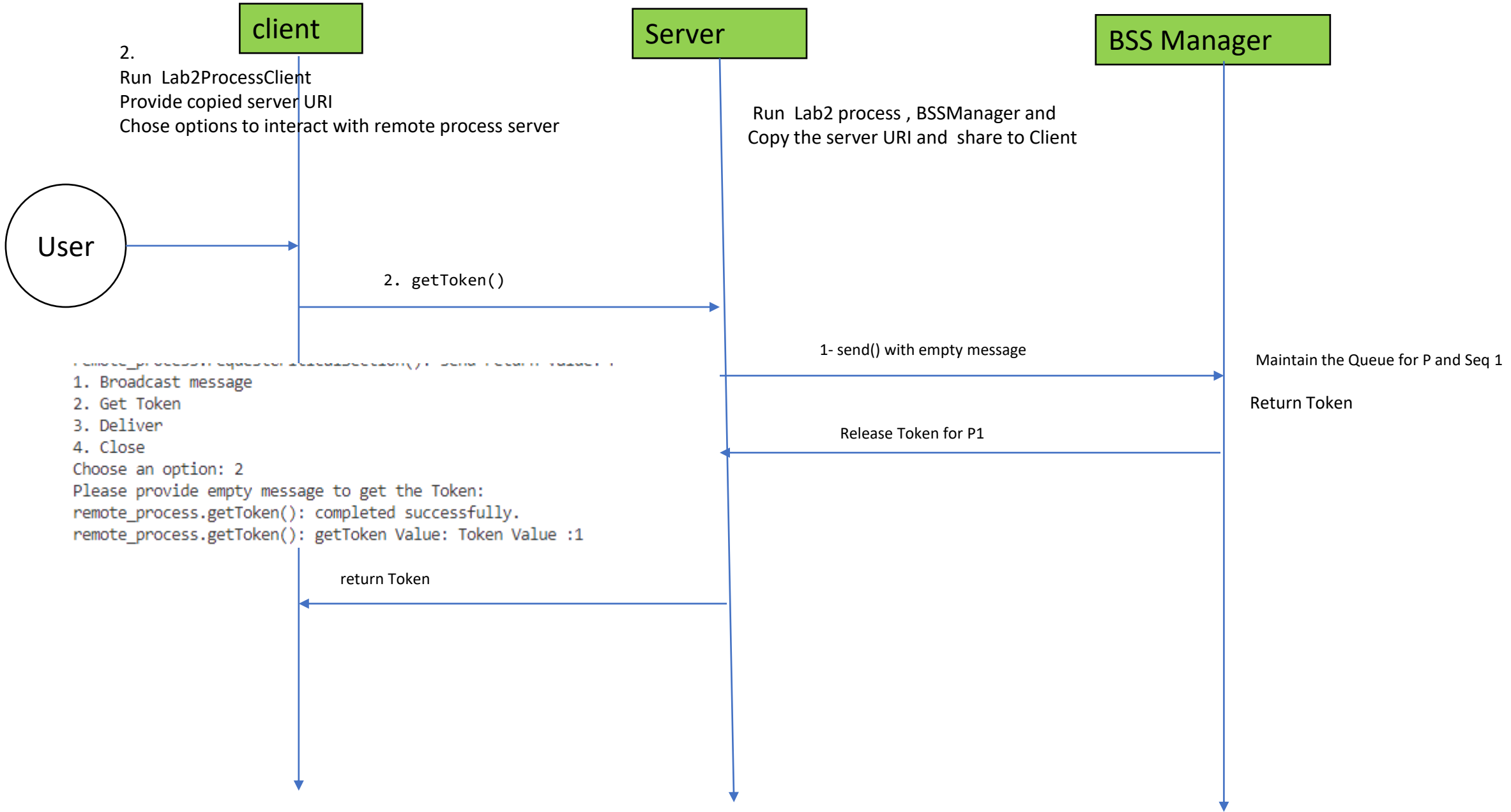
To implement BSS using RMI with Python's Pyro library, you can follow these steps.



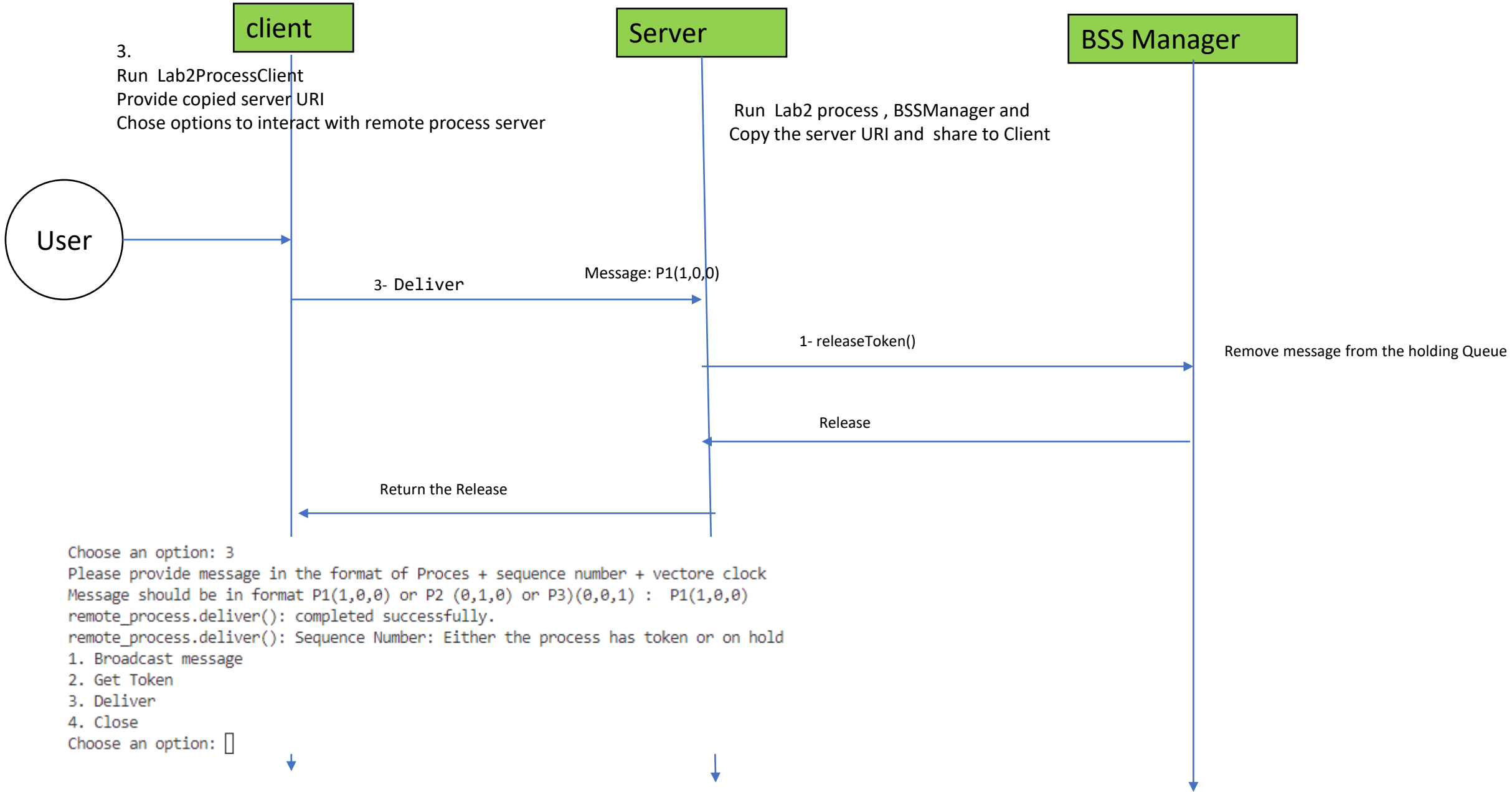

```
Enter the UnicastRemoteObject for BSS Protocol URI: PYRO:obj_f746d9e1c9644e449a9f1224379140cb@localhost:65150
Pyro5.api.Proxy(): ClockArry Pyro5.api.Proxy completed successfully while calling.
1. Broadcast message
2. Get Token
3. Deliver
4. Close
Choose an option:
```

```
Choose an option: 1
Please provide message in the format of Proces + sequence number + vectore clock
Message should be in format P1(1,0,0) or P2 (0,1,0) or P3)(0,0,1) : P3(0,0,1)
remote_process.send(): completed successfully.
remote_process.requestCriticalSection(): send return value: Hold
```

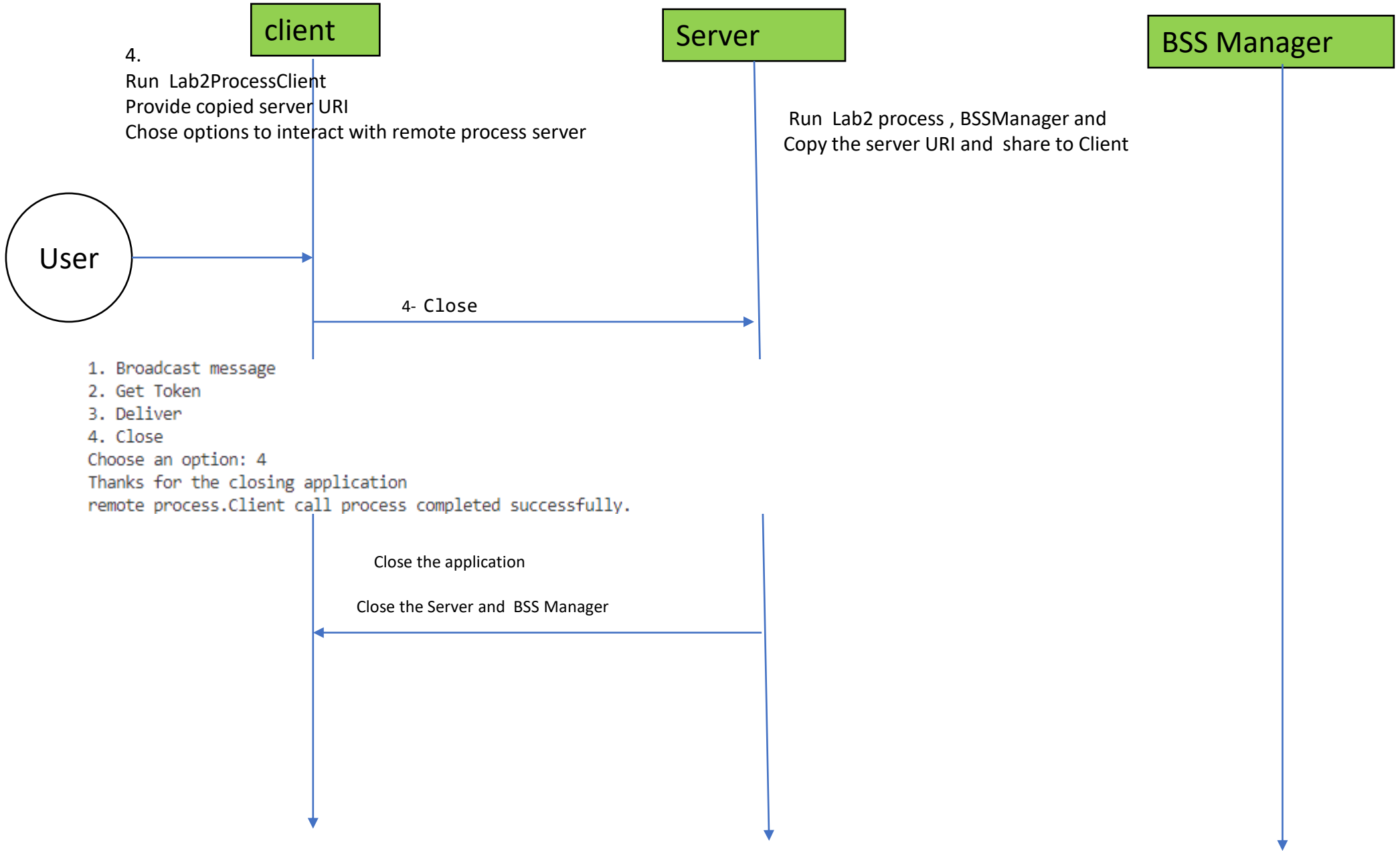
To implement BSS using RMI with Python's Pyro library, you can follow these steps.



To implement BSS using RMI with Python's Pyro library, you can follow these steps.



To implement BSS using RMI with Python's Pyro library, you can follow these steps.



Thank you

Questions

???

```
C:\Jatinkrai\MS Program\SIU-Sajesh\Course\Sem1\DistributedSystem\Lab1-Document\project-lab21\Draftv0\coding>python iLab2ProcessClient.py
Enter the UnicastRemoteObject for BSS Protocol URI: PYRO:obj_08a4e7e9816f49cfb59b49f78af5d0d0@localhost:60512
Pyro5.api.Proxy(): ClockArry Pyro5.api.Proxy completed successfully while calling.
1. Broadcast message :
2. Get Token :
3. Deliver :
4. Close :
Choose an option: 1
Please provide message in the format of Proces + sequence number + vectore clock
Message should be in format P1(1,0,0) or P2 (0,1,0) or P3)(0,0,1) : P1(1,0,0)
remote_process.send(): completed successfully.
remote_process.send(): send return value: Hold
List of process recived :
P:2:M1:(1, 0, 0)
P:3:M1:(1, 0, 0)
1. Broadcast message :
2. Get Token :
3. Deliver :
4. Close :
Choose an option: 3
Please provide message in the format of Proces + sequence number + vectore clock
Message should be in format P1(1,0,0) or P2 (0,1,0) or P3)(0,0,1) :  P2(0,1,0)
remote_process.deliver(): completed successfully.
remote_process.deliver(): Sequence Number:
P:3:M1:(1, 0, 0)
1. Broadcast message :
2. Get Token :
3. Deliver :
4. Close :
Choose an option: 3
Please provide message in the format of Proces + sequence number + vectore clock
Message should be in format P1(1,0,0) or P2 (0,1,0) or P3)(0,0,1) :  P3(0,0,1)
remote_process.deliver(): completed successfully.
remote_process.deliver(): Sequence Number: RELEASE
1. Broadcast message :
2. Get Token :
3. Deliver :
4. Close :
Choose an option: 4
Thanks for the closing application
remote_process.Client call process completed successfully.
```