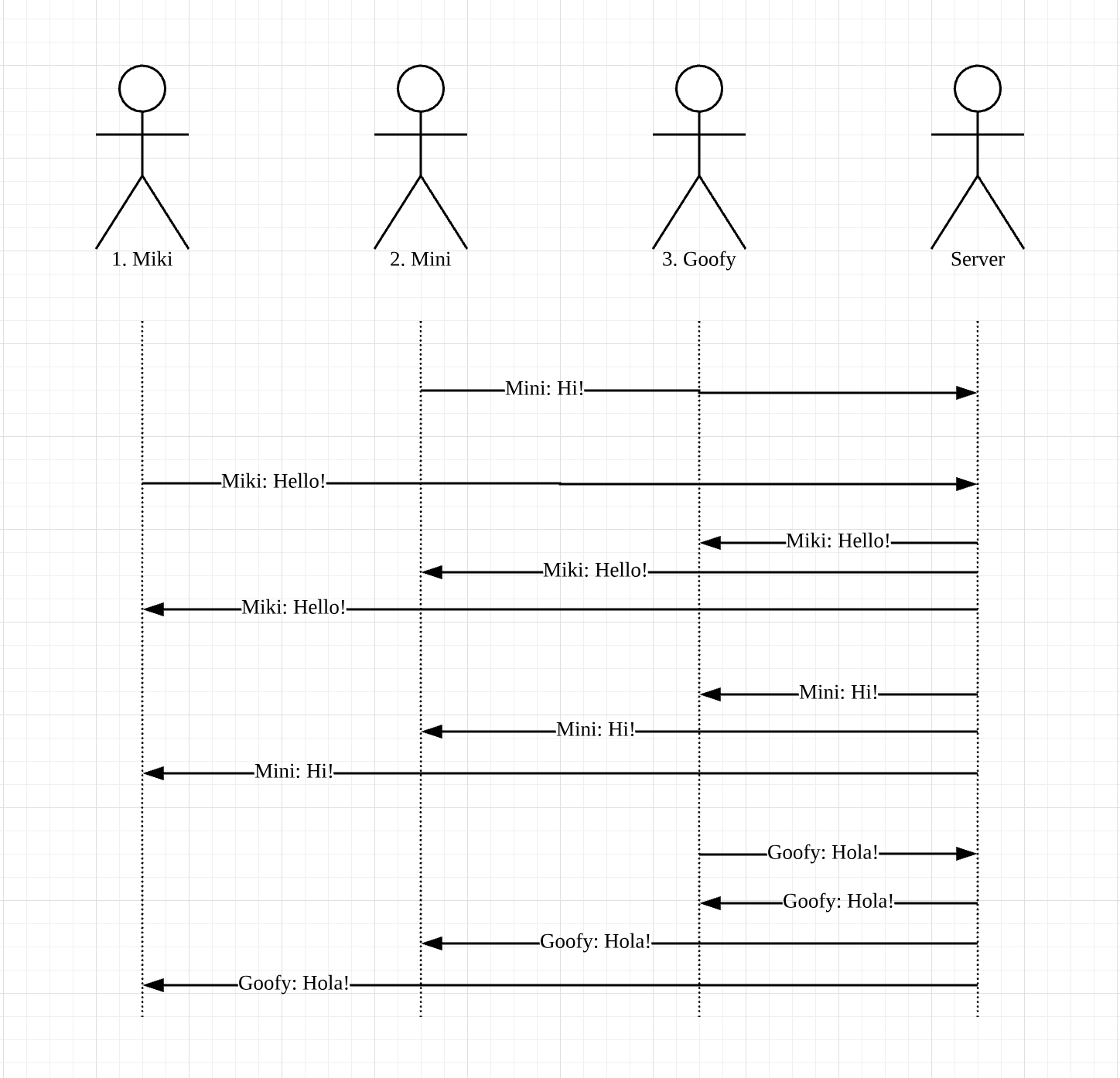
# Documentation

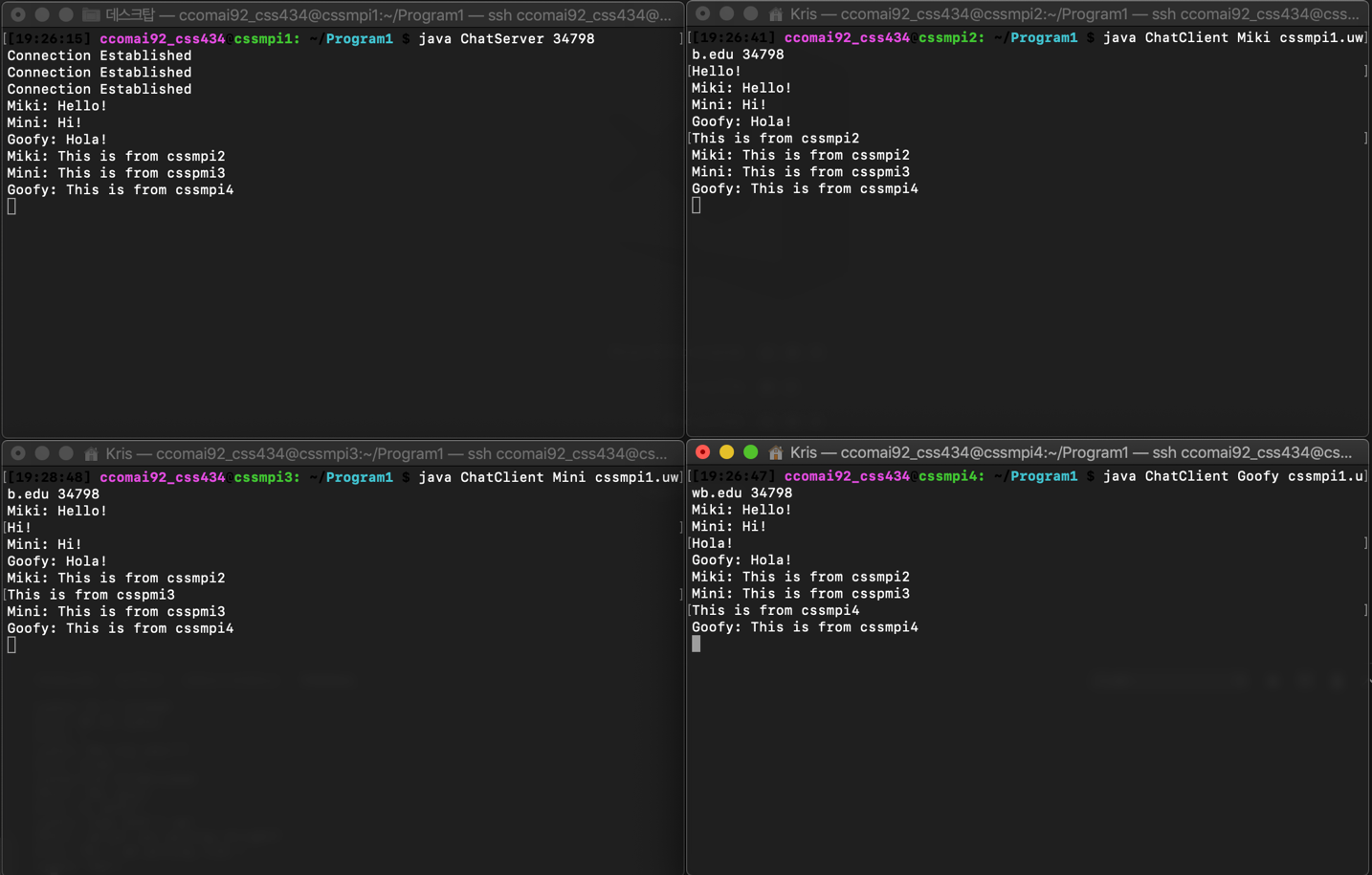
## Program: Two Chat System

**Part 1. Chat in Consistent Order**

* Server-Client Program (ChatServer.java & ChatClient.java)
  + Two different programs for server and client.
  + Clients are connected to the server.
  + Server handles the ordering.
* Algorithm
  + 1. The server-side program utilizes consistent ordering algorithm in combination of the FIFO. Since TCP connection takes care of FIFO ordering, the server program deals with the total ordering.
    2. First, server continuously checks and accepts connections from the clients with their name. If there is no connection request from new clients, server checks available message buffer of every connection.
    3. If the program finds message available in the buffer, it reads the message with corresponding client name. Then, the program broadcasts received message to every connected clients.
       - If any error occurs during the reading and writing operation, the error caused connection will be marked so that it can be removed later.
    4. Finally, the server program checks error marked connections and remove it from the client connection list.
    5. Go to step 2
* Consistent Ordering Example Illustration (# in front of name is order in the connection list)



* Execution Snapshot



* Discussion
  + Algorithm Efficiency

The efficiency of the algorithm is the order of n^2 since it checks and every communication path, and again, send message to every communication path.

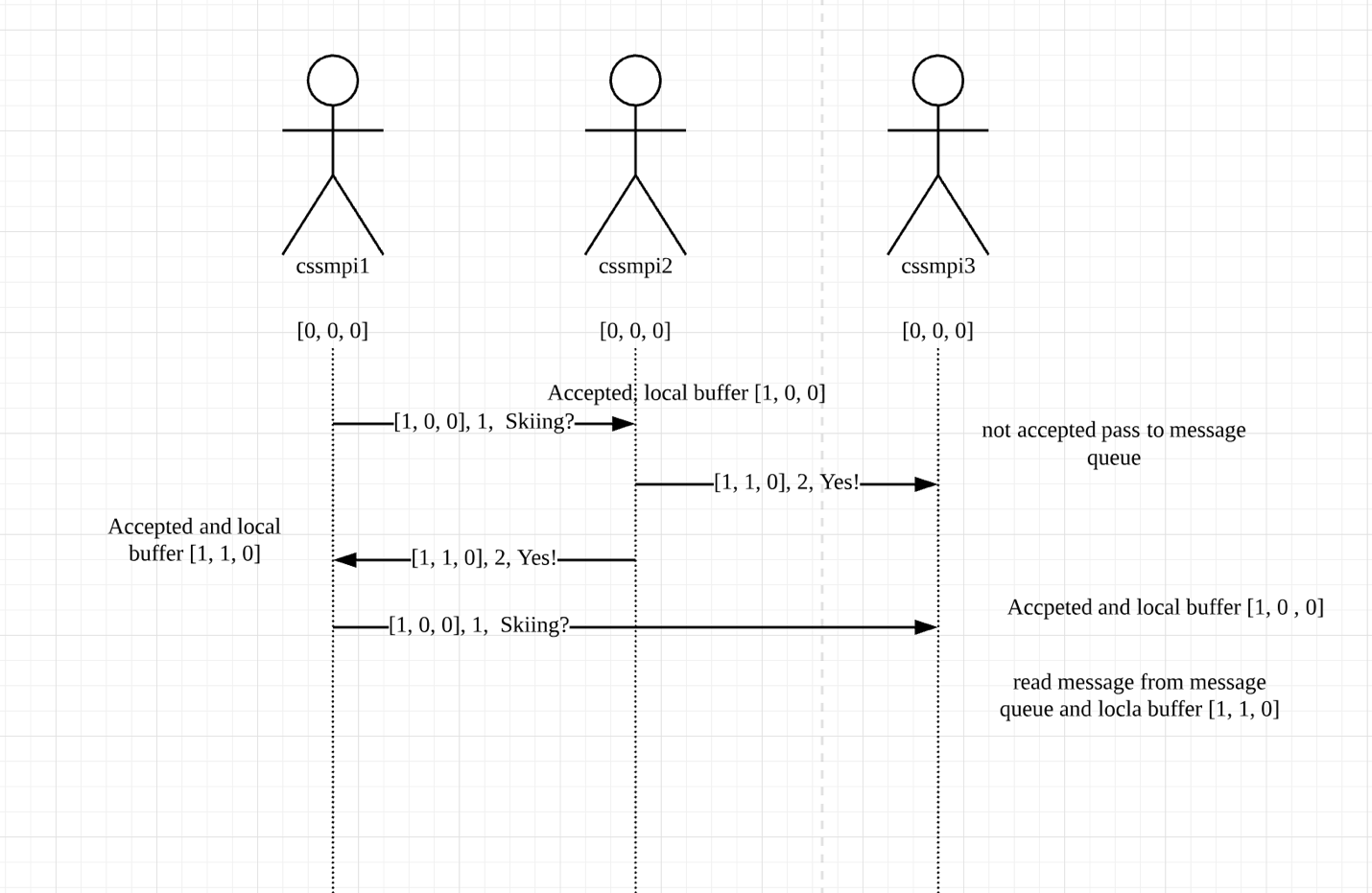
The consistent ordering algorithm takes advantage of TCP implementation. By definition of TCP protocol and in accordance with FIFO message ordering, consistent ordering can be efficient to make sure message to be ordered and delivered. However, this can have serious vulnerability in terms of reliability of the system. Due to the nature of the centralized system, server system has all overheads such as connections and tasks. Failure of server can cause serious problem in the system.

* + Performance and functional improvement

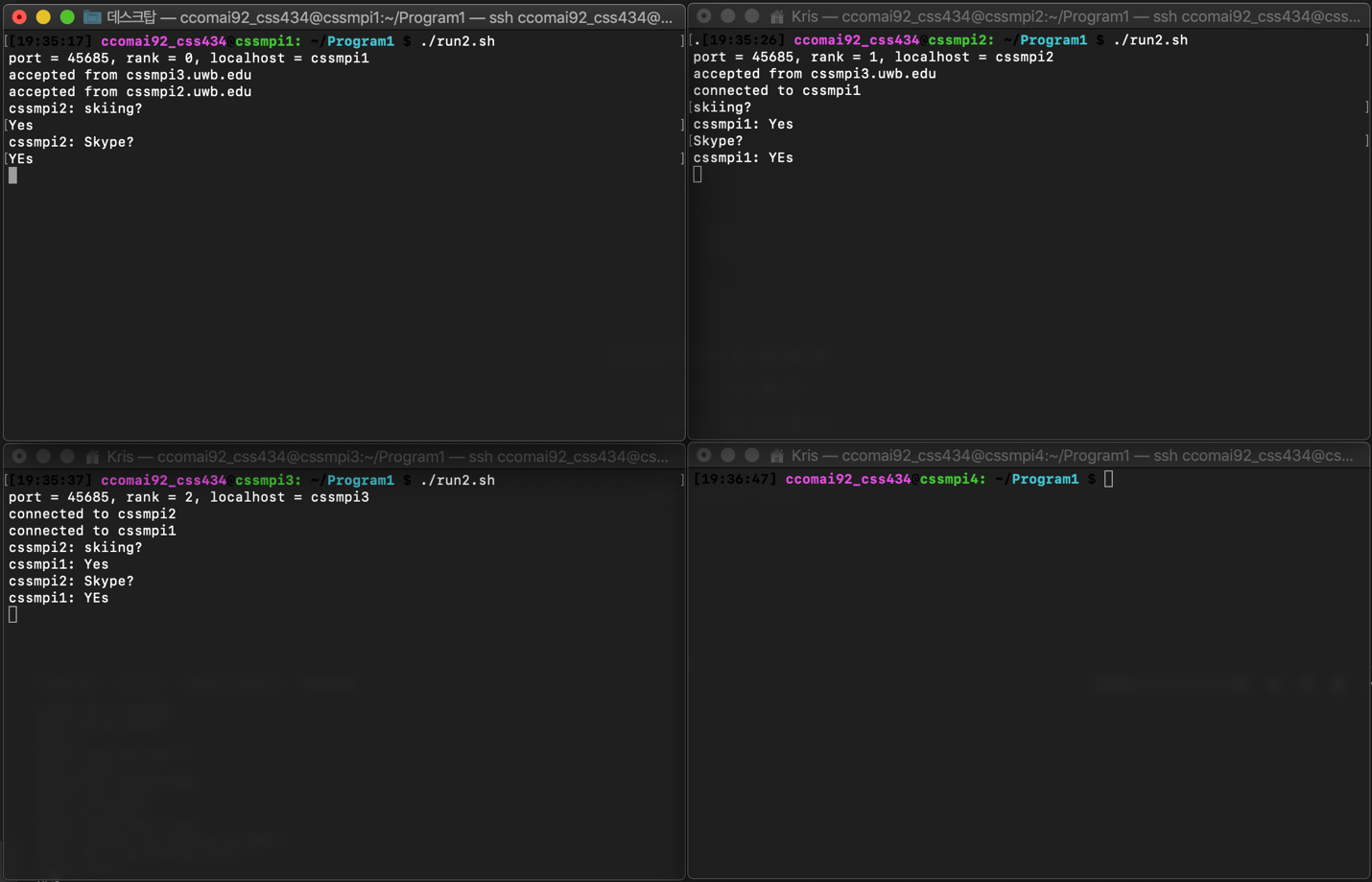
The server program is not implemented in multi-threading programming. Therefore, one thread is checking everything while operating tasks to order and deliver messages. In such small, system, this would not be a big problem. However, the program requires improvements to be scalable. Although the separation of accepting connection and processing message delivery methods would increase the performance in general, it will the efficiency of the algorithm would be still N^2.

**Part 2. Chat in Causal Order**

* No Server-Client programs, (only Chat.java)
* Algorithm (Assumption, TCP is reliable connection)
  + 1. Once cssmpi1, 2, and 3 are connected (given), the program on each computing node initializes timestamp vector as [0, 0, 0], and a waiting message queue.
    2. The program checks chat user input. If user type any message, the program increment stamp vector’s value of current rank by one and create message packet with current computing node’s rank, time stamp, and the actual message. Then, current computing node send message packet to every other node.
    3. Now current computing node checks whether message packets are available to read. If message arrives, current computing node checks whether the time stamp in the message packet is acceptable.
       - Comparing time stamp
         * If sender’s value in packet stamp, is less or equal to the sender’s value in local stamp, it is not acceptable. Otherwise, acceptable.
         * If other computing node’s (other than the sender) value in packet’s stamp is bigger than local stamp’s corresponding value, it is not acceptable. Otherwise, acceptable.
       - If acceptable, current program accepts the message and update local time stamp by incrementing value of sender’s by one. Then, checks if message in the waiting message queue can be accepted now.
       - If not acceptable, the program push message packet in the back of the waiting list.
    4. Go back to 2
* Causal Ordering Example Illustration



* Execution Snapshot



* Discussion
  + Algorithm Efficiency

Although the algorithm is efficient at keeping orders among the messages, the efficiency of the algorithm is N^2. However, message requires additional information such as sender and time stamp. It requires overheads in every message packet that are delivered. Moreover, when one message is delivered late, the size of message queue could consume significant amount of memory resources if the system is large.

* + Performance and functional improvement

The Causal ordering algorithm is decentralized message ordering and there is no such focal point as server. Moreover, knowing the number of computers joining the chat system would be hard and requires another discussion. To improve functionality, it would be nice to have that decentralized computing nodes can join in the middle of the chat session. Finally, for the performance improvement, if any can join in during the chat session, separating the thread of connection and the thread of chat operation will improve the performance in general. However, the big O notation of efficiency would not change.