Chatting with Web Sockets

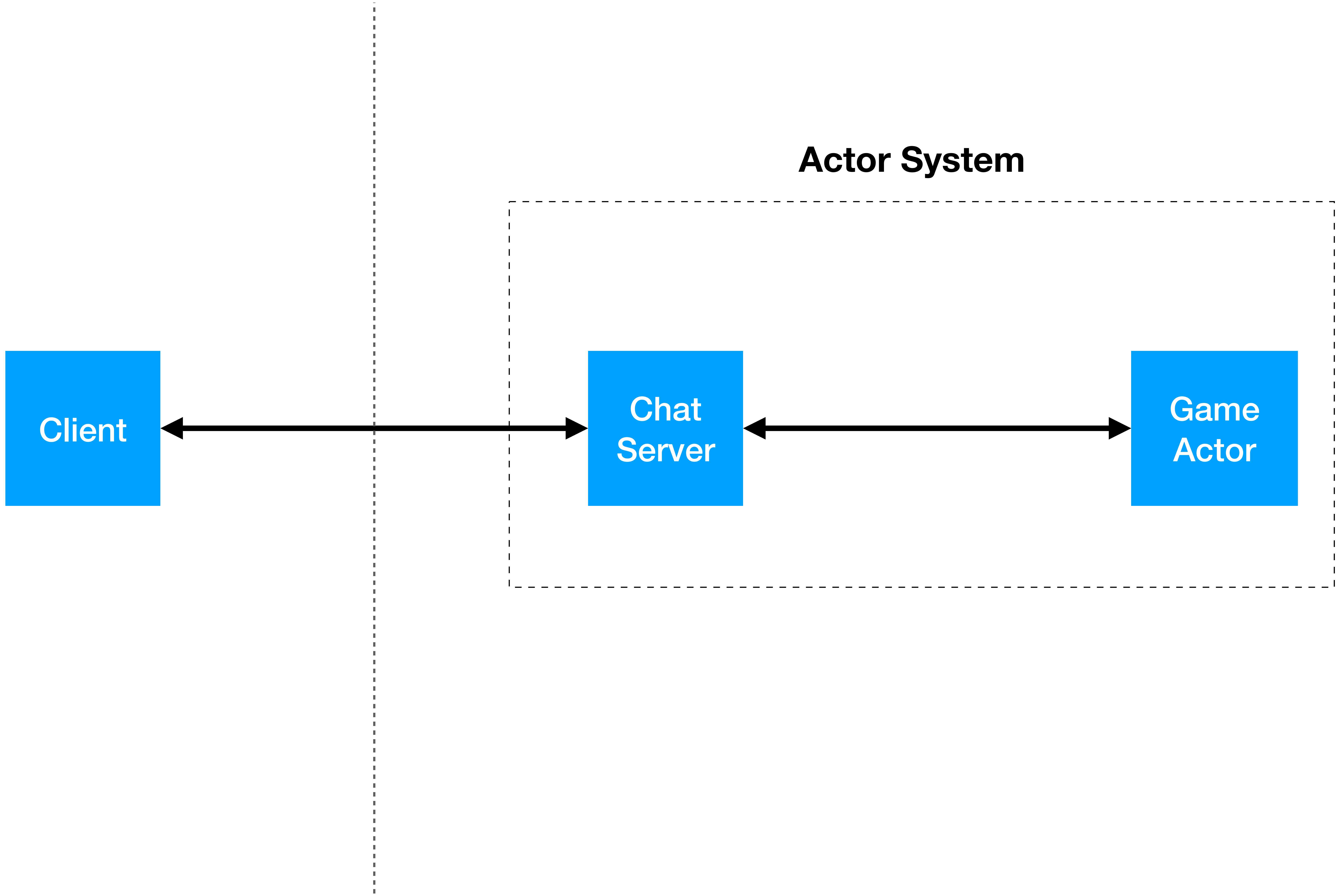
Lecture Question

Task: Write a Web Socket Server for Direct Messages (DMs)

In a package named server, write a class named DMServer that:

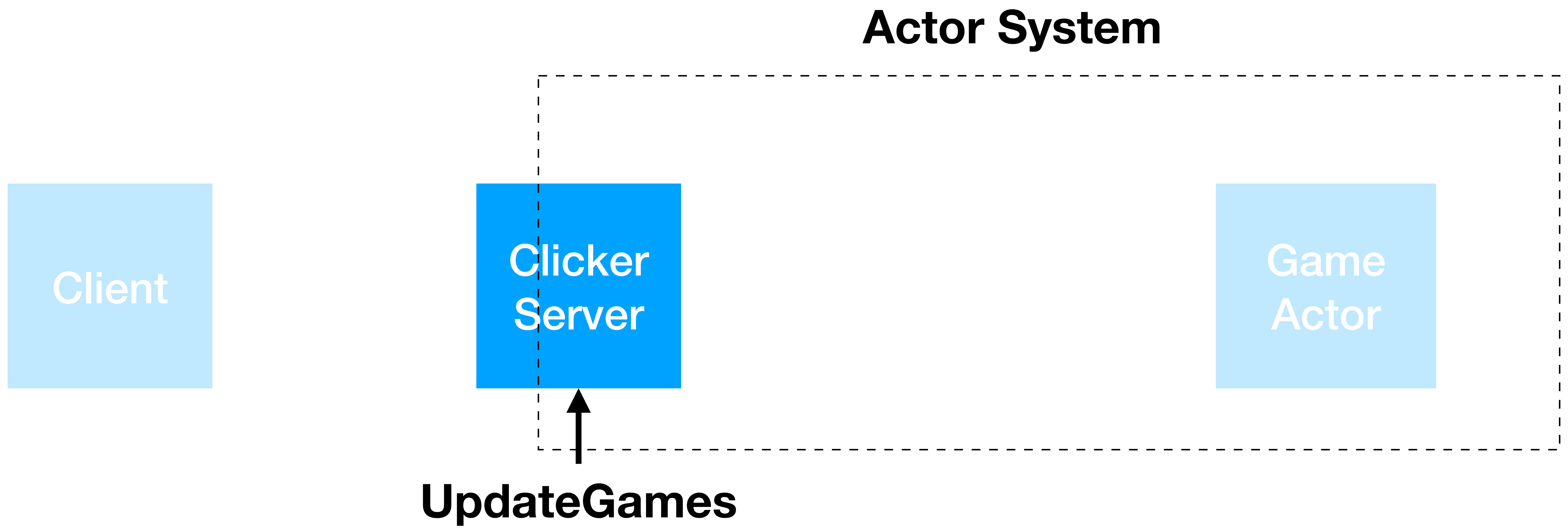
- When created, sets up a web socket server listening for connections on localhost:8080
- Listens for messages of type "register" containing a username as a String (Use data structures to remember which socket belongs to which username)
- Listens for messages of type "direct_message" containing a JSON string in the format {"to":"username", "message":"text"}. When such a message is received:
  - Send a message of type "dm" to the "to" username containing a JSON string in the format {"from":"username", "message":"text"}
- Example: If 2 different users connect to the server and send:
  - emit("register", "Aesop") and emit("register", "Rob")
  - User "Aesop" sends emit("direct_message", '{"to": "Rob", "message": "Happy to be on the food chain at all"}')
- User "Rob" will receive a message from the server of type "dm" containing the string '{"from": "Aesop", "message": "Happy to be on the food chain at all"}'

Clicker Architecture



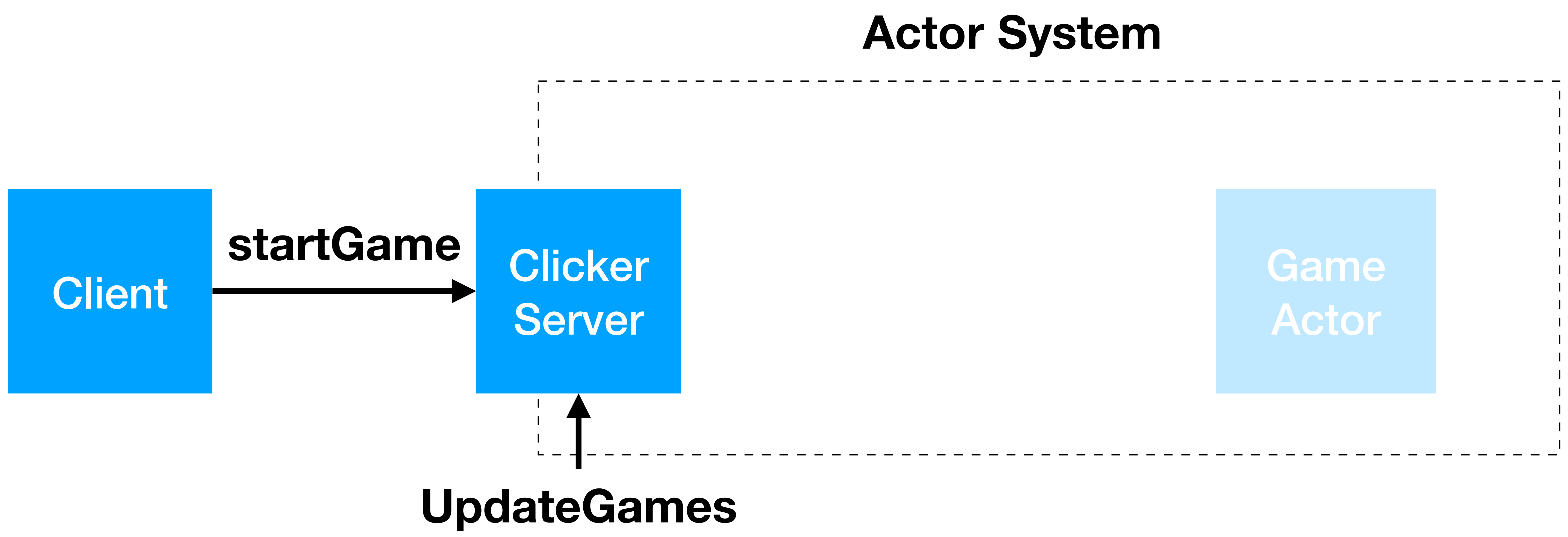
Clicker App

- When the app starts
  - An actor system is created
  - A ClickerServer actor is added to the system
  - UpdateGames message is sent to the server at regular intervals



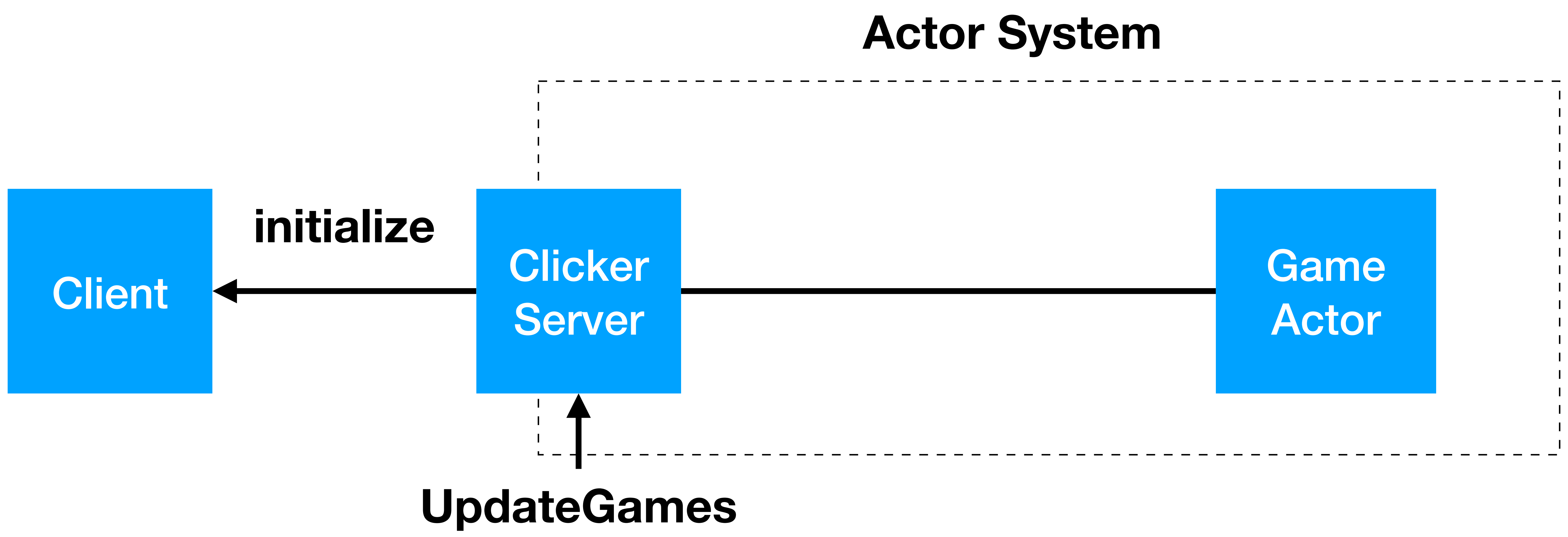
Clicker App

- When a client connects and chooses a username
  - This username is sent to the server in a WebSocket message of type startGame



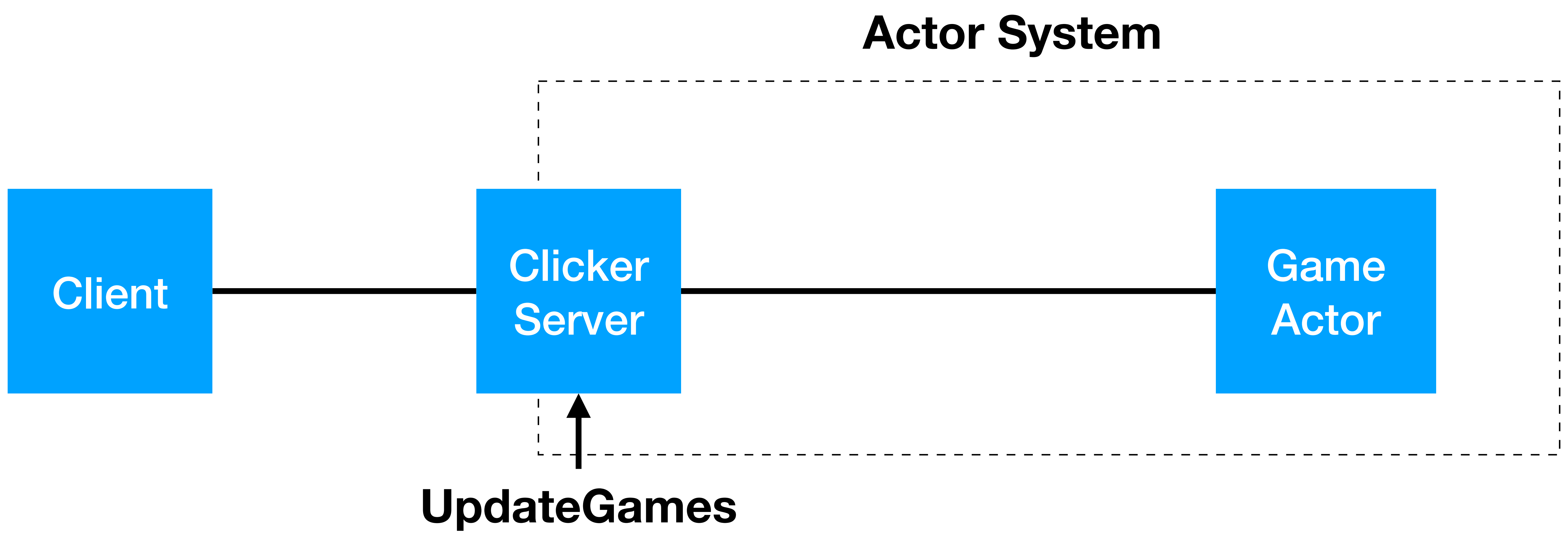
Clicker App

- In response to receiving the gameStart message, the server:
  - Sends the client the game configuration in a message of type initialize
  - Creates a GameActor in the actor system
  - Updates data structure(s) to remember that this game actor is associated with this web socket



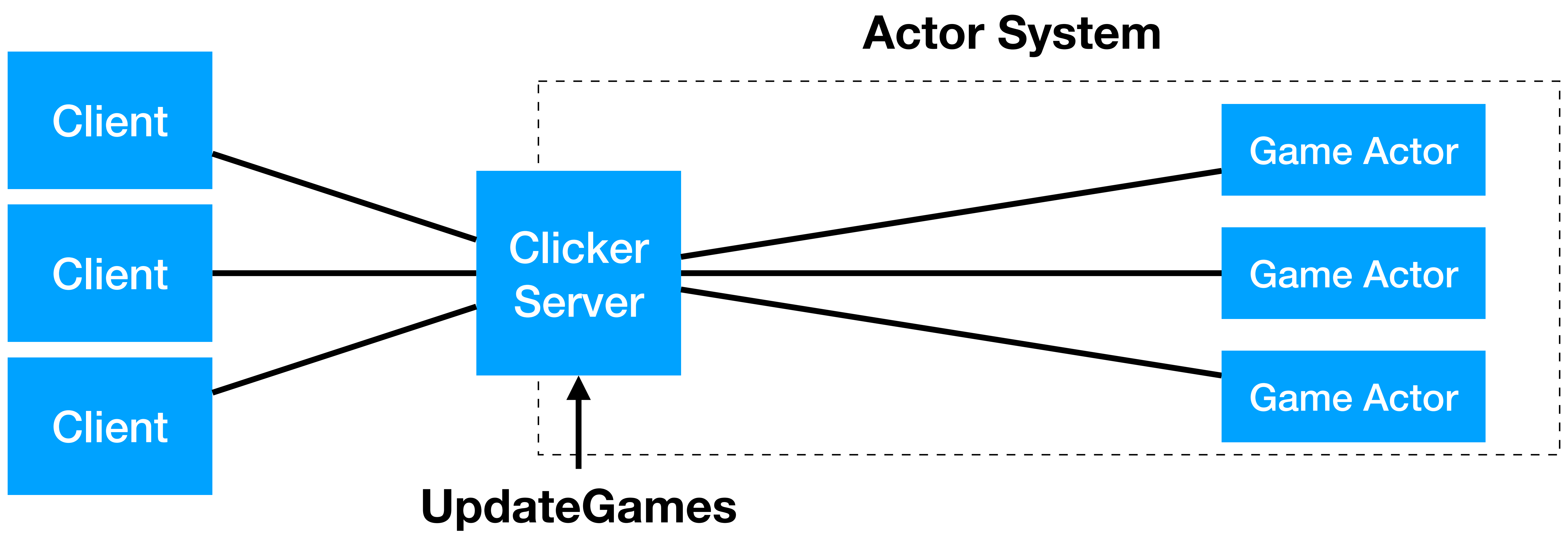
Clicker App

- To create a new Actor
  - Use the context variable of any actor
  - Use this context the same as the actor system
  - Ex. clickerServer.context.actorOf...



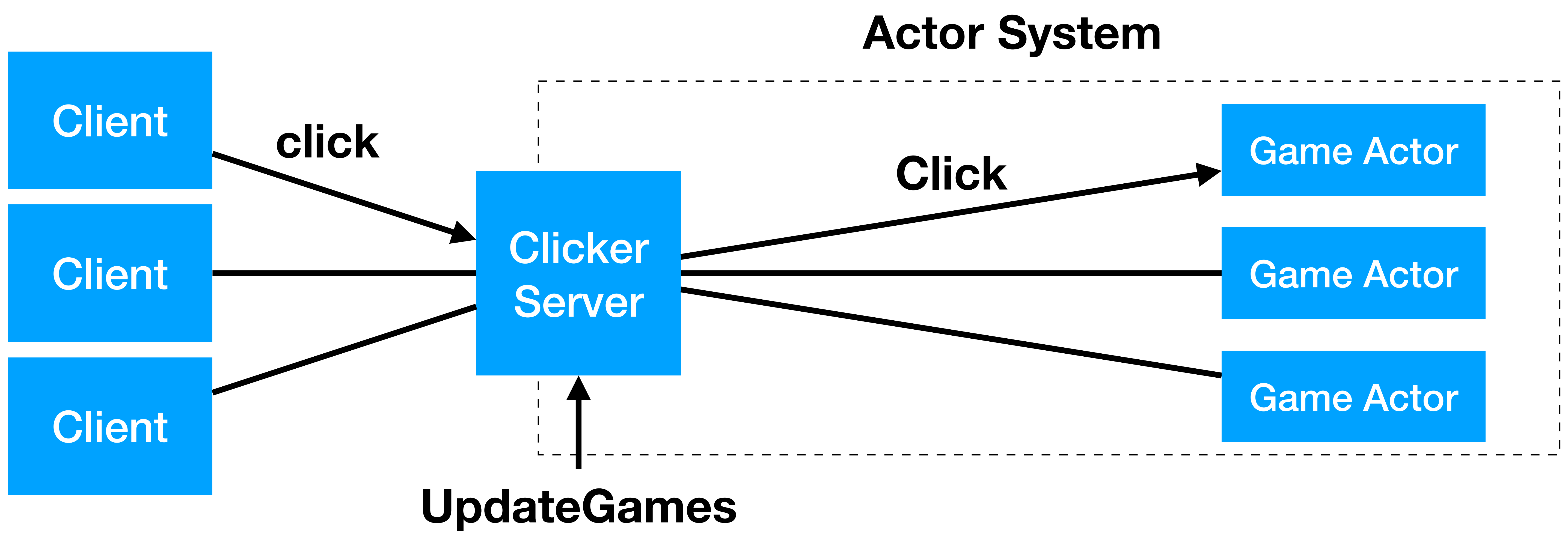
Clicker App

- A new game actor is created for each connected client
- Important to update all data structures to associate clients with their actors



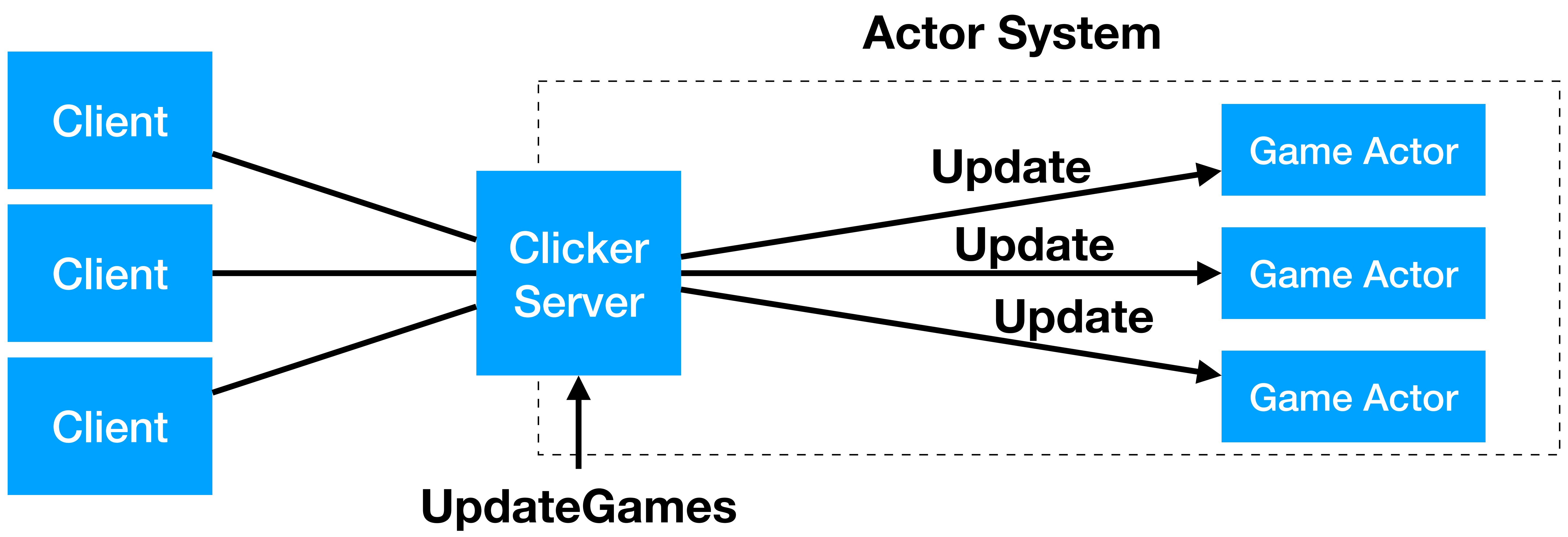
Clicker App

- When the server receives click and buy message from a web socket
  - Forward the action as an actor message to the appropriate actor
  - Game actor will update its state according to the configuration of the game



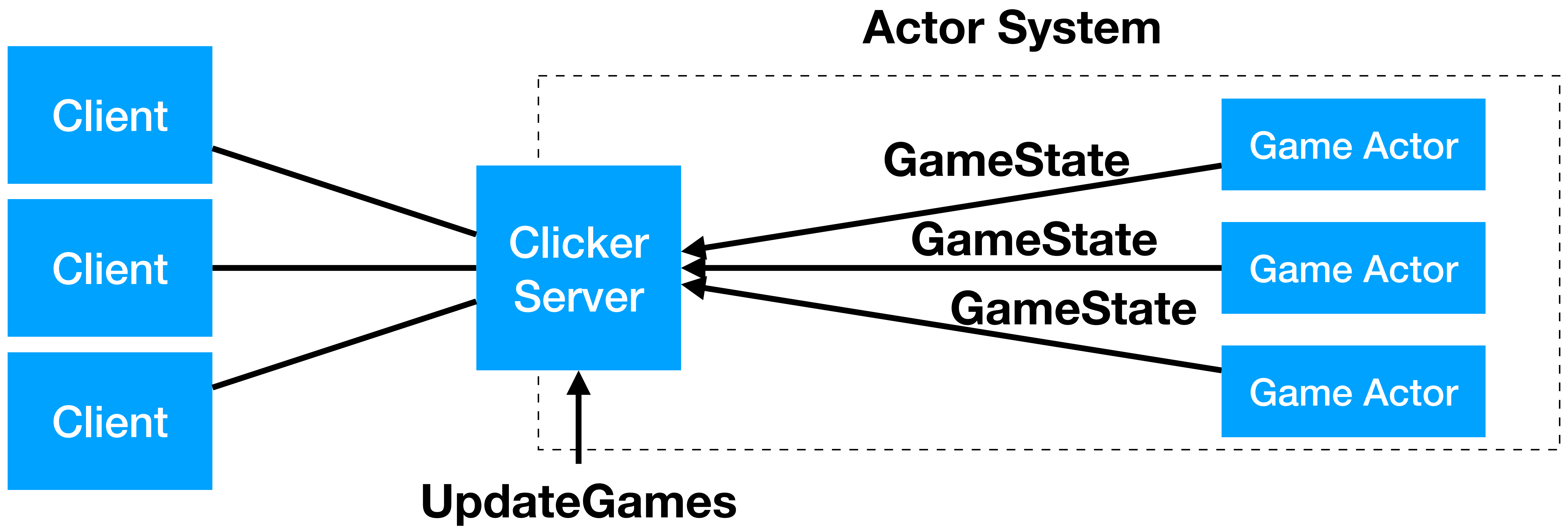
Clicker App - Update

- Each time the clicker server receives the UpdateGames actor message
  - Send an Update message to each game actor



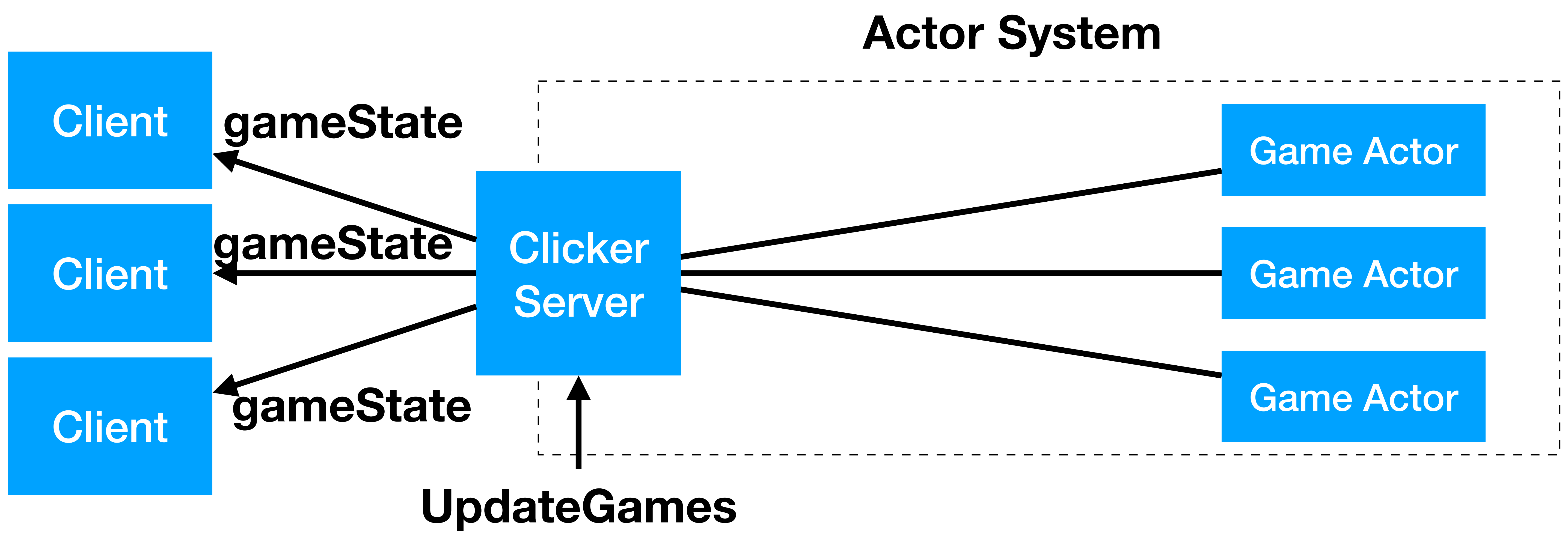
Clicker App - Update

- Each game actor responds with the GameState message (to the sender())
- GameState contains all information of the game in a JSON string



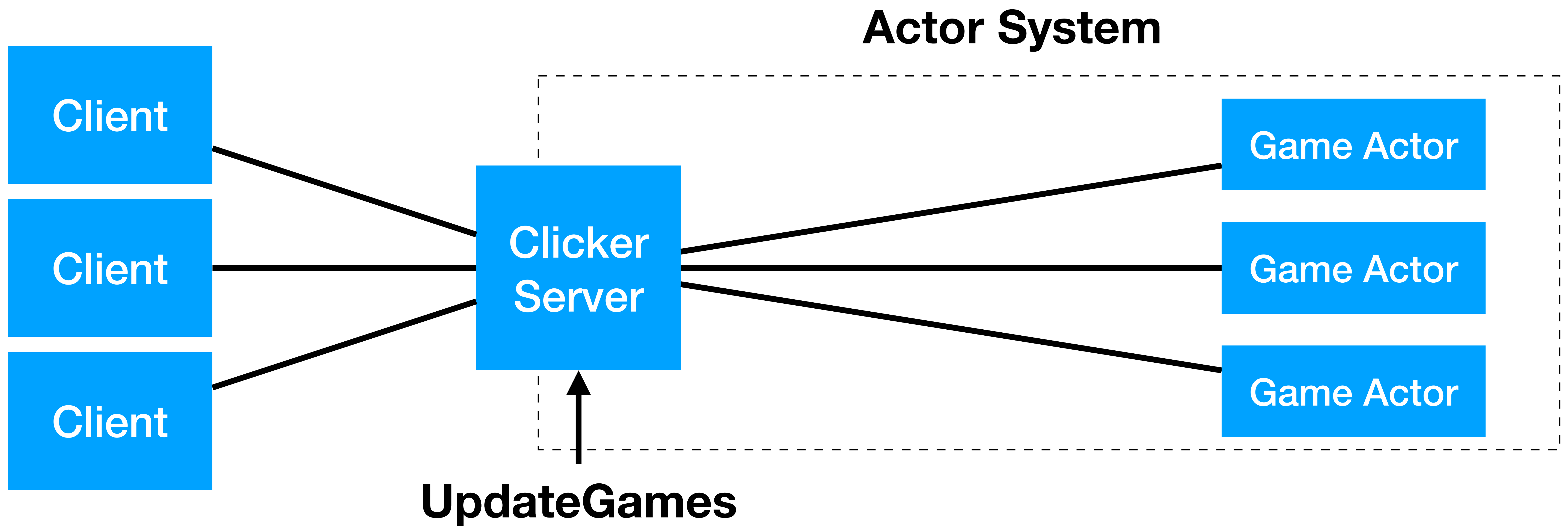
Clicker App - Update

- The clicker server forwards each game state to the appropriate client in a gameState message
- Each client parses the JSON string and updates the GUI for the user to see



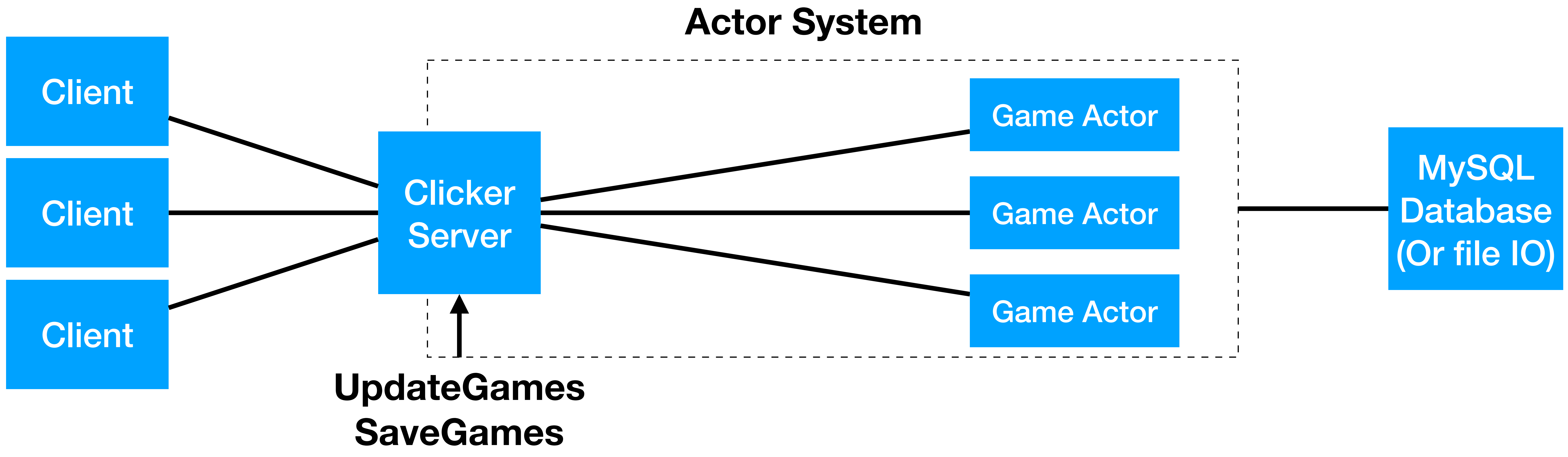
Clicker App - Update

- This update process occurs at regular intervals
  - 10 times/second in the handout code
- Notice that all the game logic occurs on the server
- Client only sends user inputs and renders the game state



Clicker App - Expansion

- Expansion objective - AutoSave
- Send messages to save all games at regular intervals
- Store all game states in a way that will persist
- If a user sends the startGame message with a username that has a saved game, load their game



Lecture Question

Task: Write a Web Socket Server for Direct Messages (DMs)

In a package named server, write a class named DMServer that:

- When created, sets up a web socket server listening for connections on localhost:8080
- Listens for messages of type "register" containing a username as a String (Use data structures to remember which socket belongs to which username)
- Listens for messages of type "direct_message" containing a JSON string in the format {"to":"username", "message":"text"}. When such a message is received:
  - Send a message of type "dm" to the "to" username containing a JSON string in the format {"from":"username", "message":"text"}
- Example: If 2 different users connect to the server and send:
  - emit("register", "Aesop") and emit("register", "Rob")
  - User "Aesop" sends emit("direct_message", '{"to": "Rob", "message": "Happy to be on the food chain at all"}')
- User "Rob" will receive a message from the server of type "dm" containing the string '{"from": "Aesop", "message": "Happy to be on the food chain at all"}'