Project Architecture

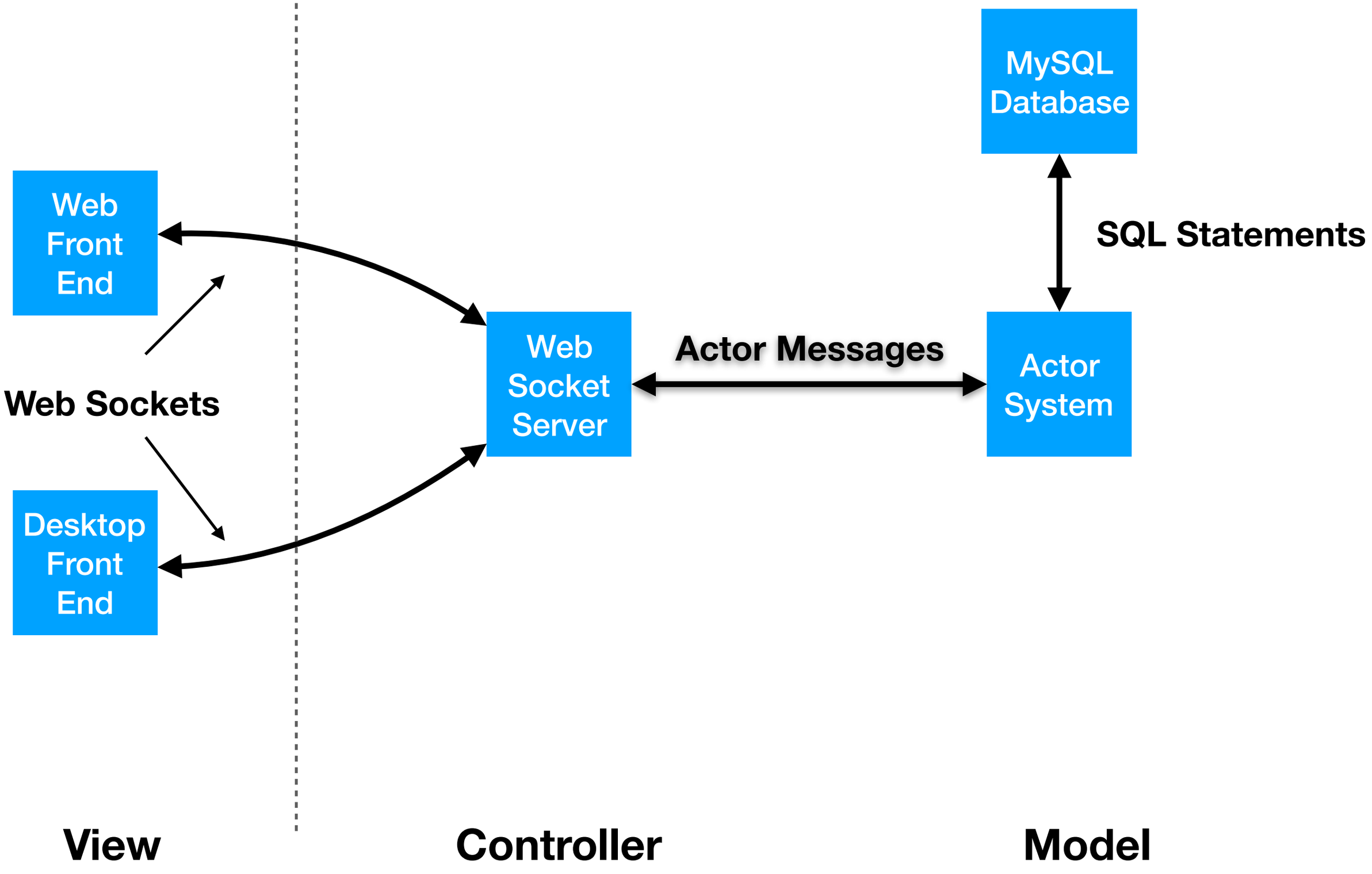
Lecture Question

Google form during lecture

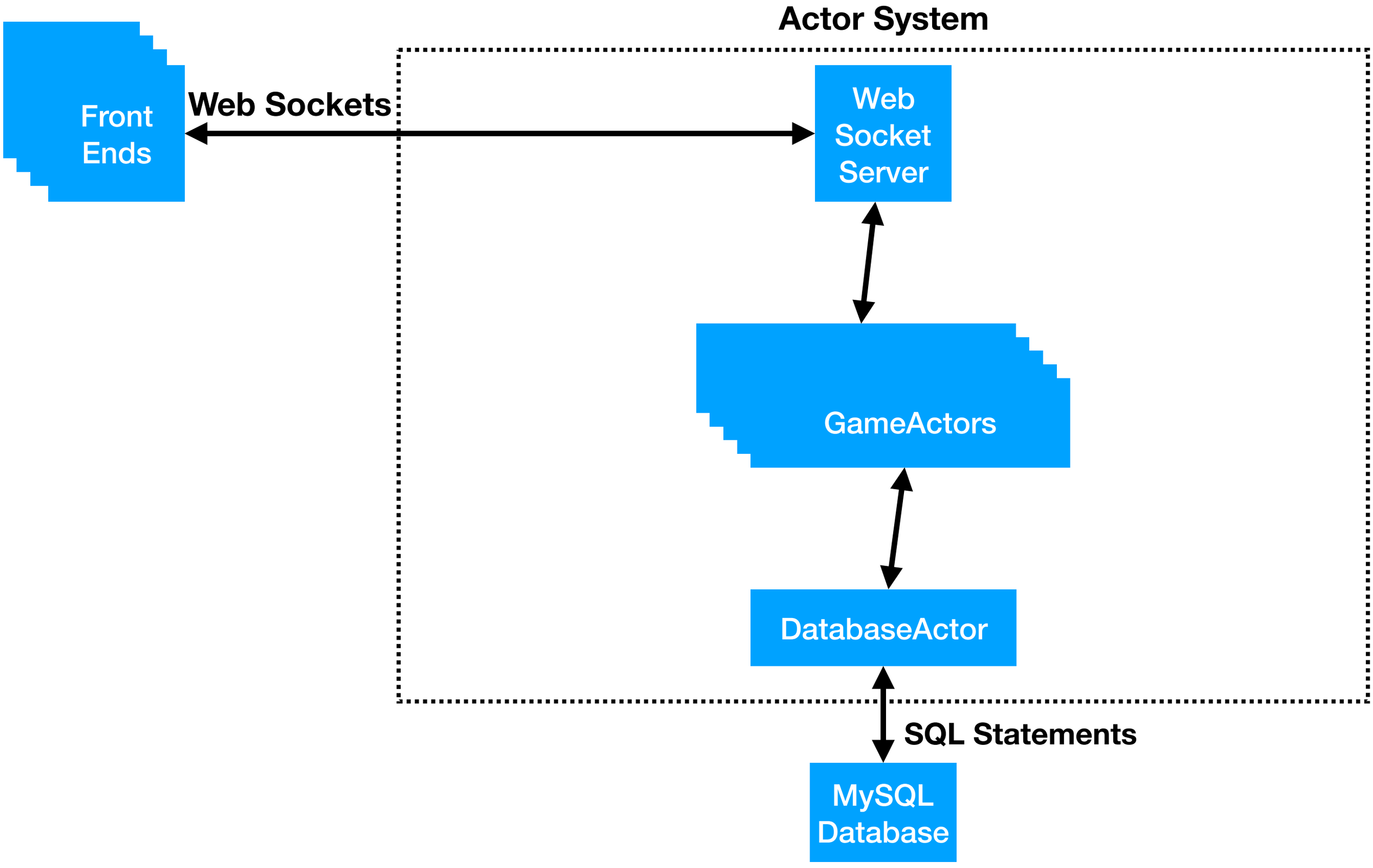
Disclaimer

- This is only one approach to build this app
- This is by no means the only way to design this software

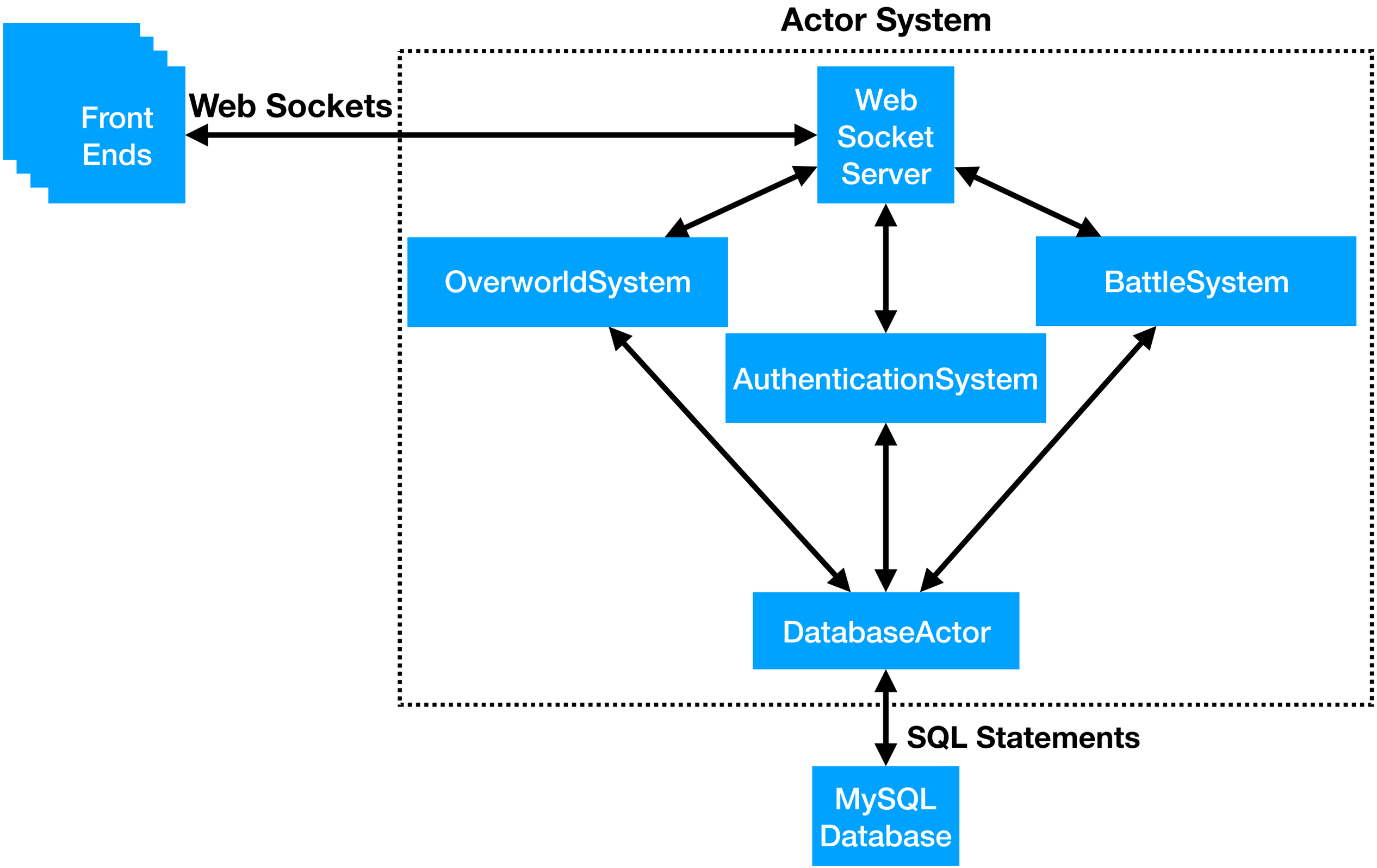
MMO Architecture



Clicker Architecture (Not an MMO)



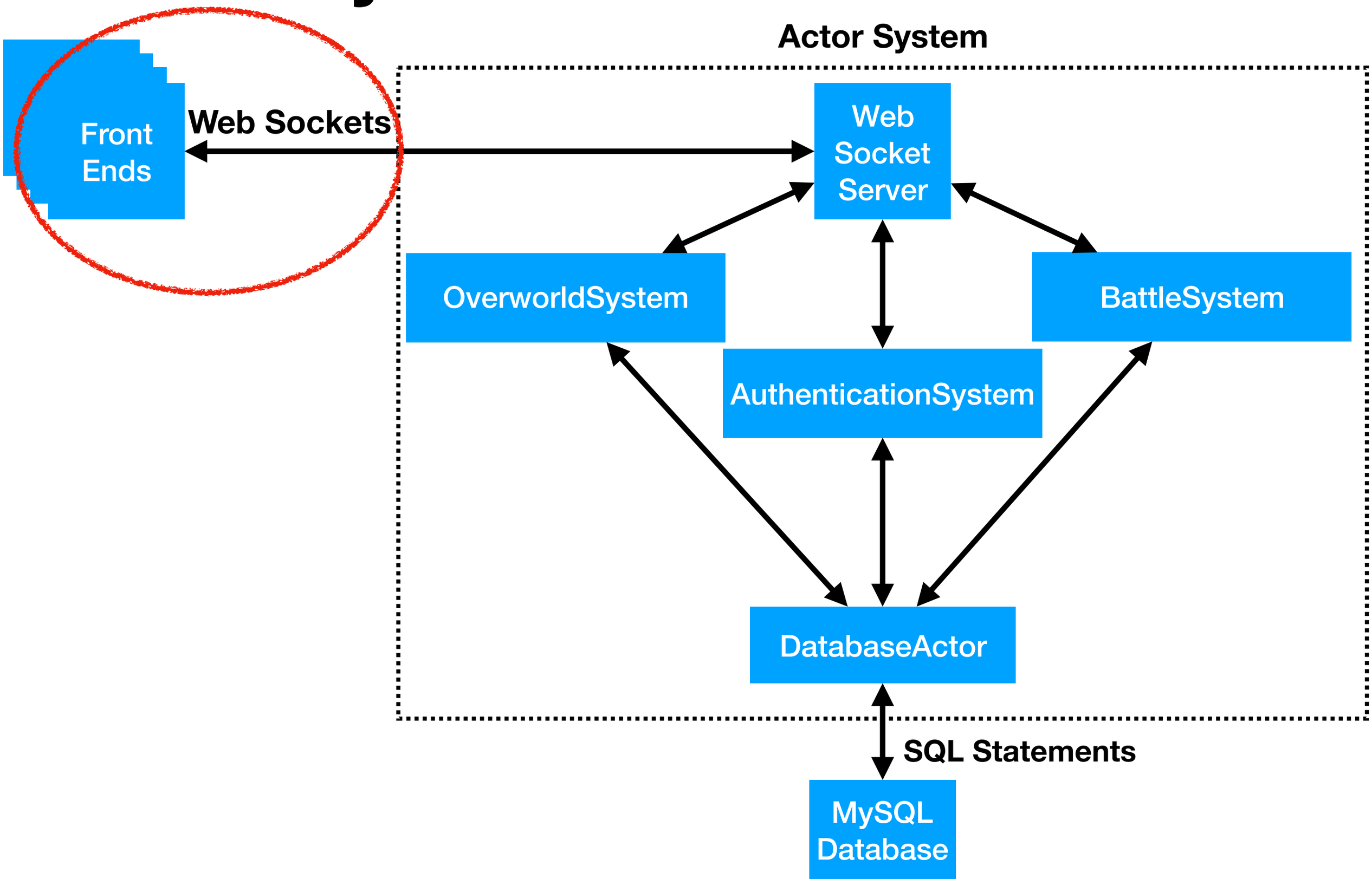
Project Architecture



Project Architecture

- For Demo 3 you will choose between:
  - Web Socket Server
  - Overworld System
  - Battle System
  - Authentication System

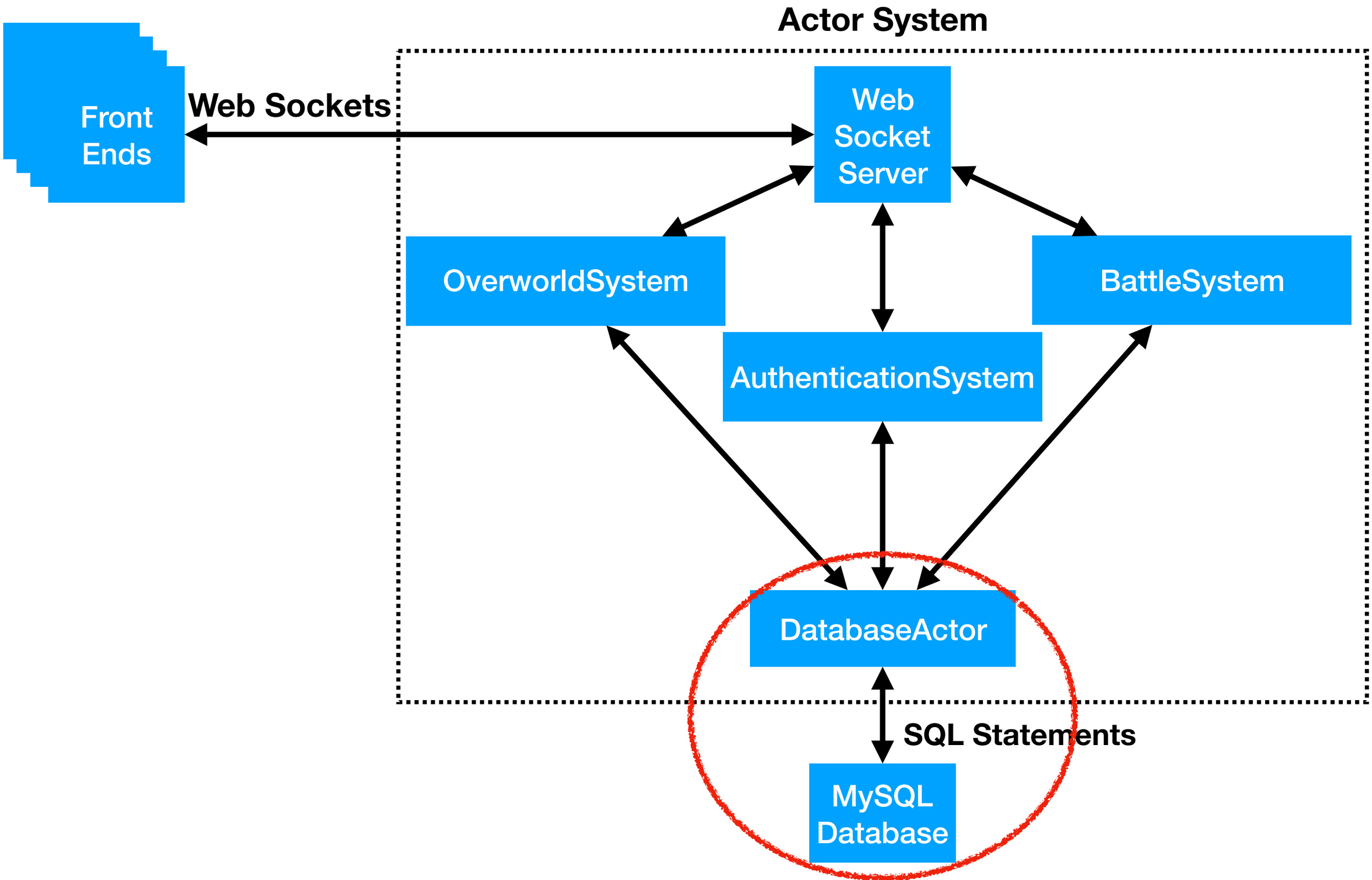
Project Architecture



Front Ends

- Built for project demo 2
- Will add socket connections
- Overworld
  - Send movement direction based on keys pressed (or mouse clicks)
  - Receive and render the map and locations of all parties
- Battle
  - Send actions taken
  - Receive render state of all characters in battle
  - Receive and animate actions taken
  - Receive notification when the user can take an action

Project Architecture



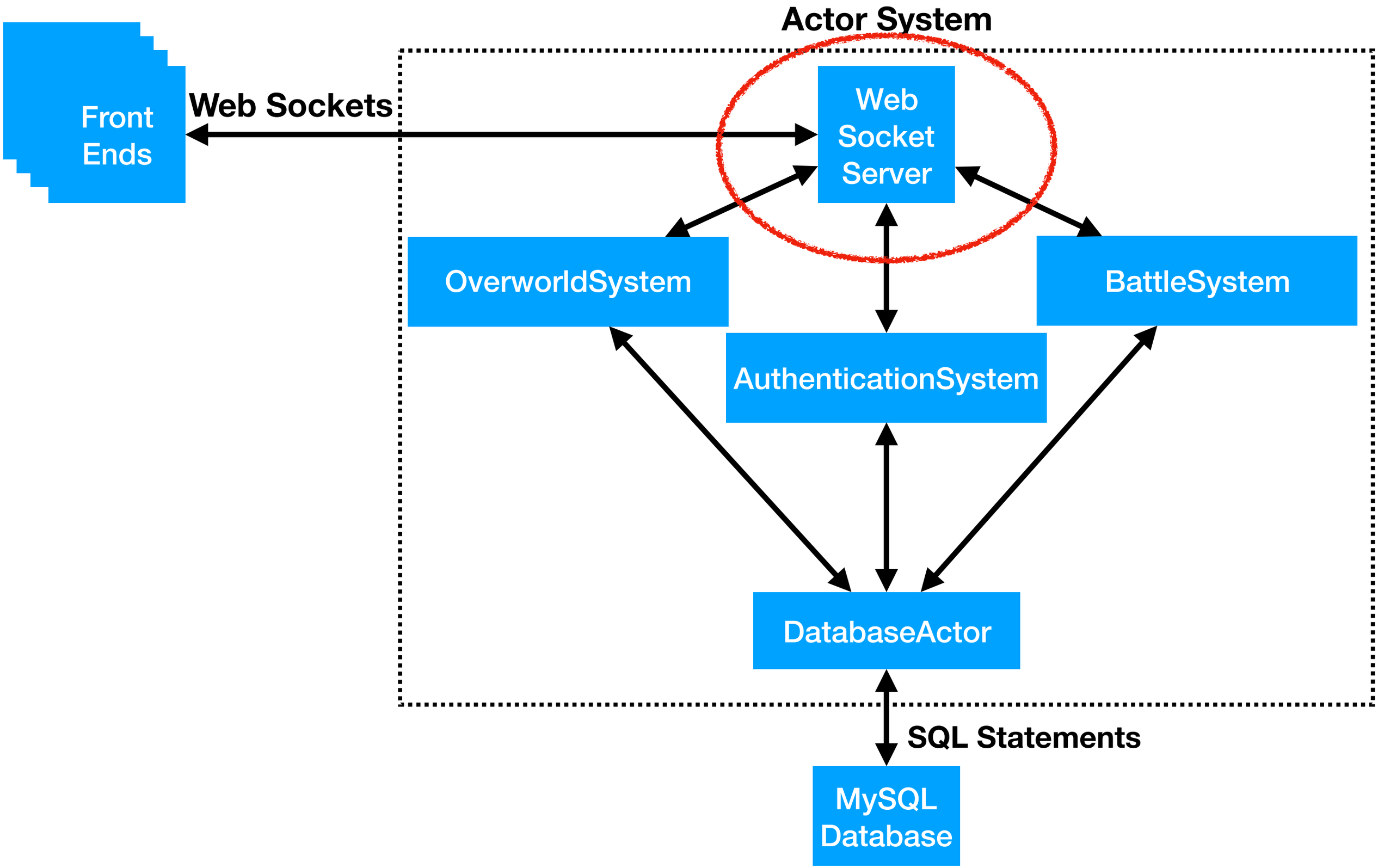
Database

- You've used this in Clicker
- Create an actor that will connect to the database
- MySQL is not part of the actor system
  - It's a separate program entirely
- Allows the server to restart without losing data

Database

- Authentication
  - Store usernames and hashed passwords
- Battle System
  - Store levels/stats/XP
- Overworld
  - Store map and party locations

Project Architecture



Web Socket Server

- The server acts as a controller
  - No game logic is processed here
  - (All the rest of demo 3 is part of the model)
- Acts as a translator between web sockets and actors
- Connects and communicates between the three systems of the app

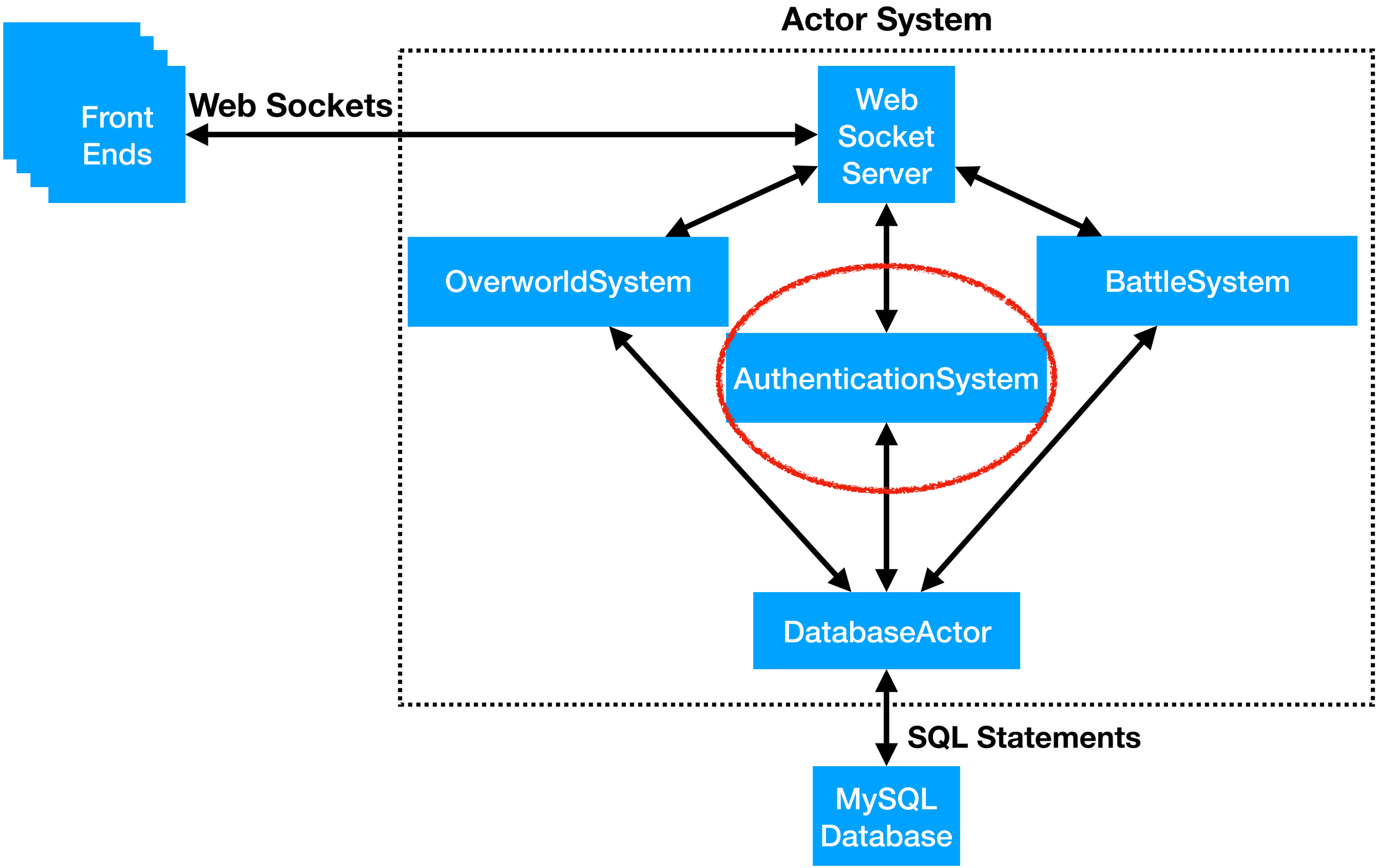
Web Socket Server

- Listens to the web socket server for:
  - Connections/Disconnections - Coordinate with authentication system to login/verify the username and add their party to the overworld and battle systems
  - Overworld movements - Lookup the party by socket and notify the overwork system
  - Battle actions - Lookup party by socket and notify the battle system

Web Socket Server

- As part of the Actor System:
  - Initializes the system and controls the update loop (periodically send update messages to all systems)
  - Receives updated states and events from overworld and battle systems
    - Forwards states to appropriate web sockets
    - Sends events to concerned parties (Ex. When the overworld detects a battle starts, notify the battle system)
- Receives verification from authentication system that user is logged in

Project Architecture



AuthenticationSystem

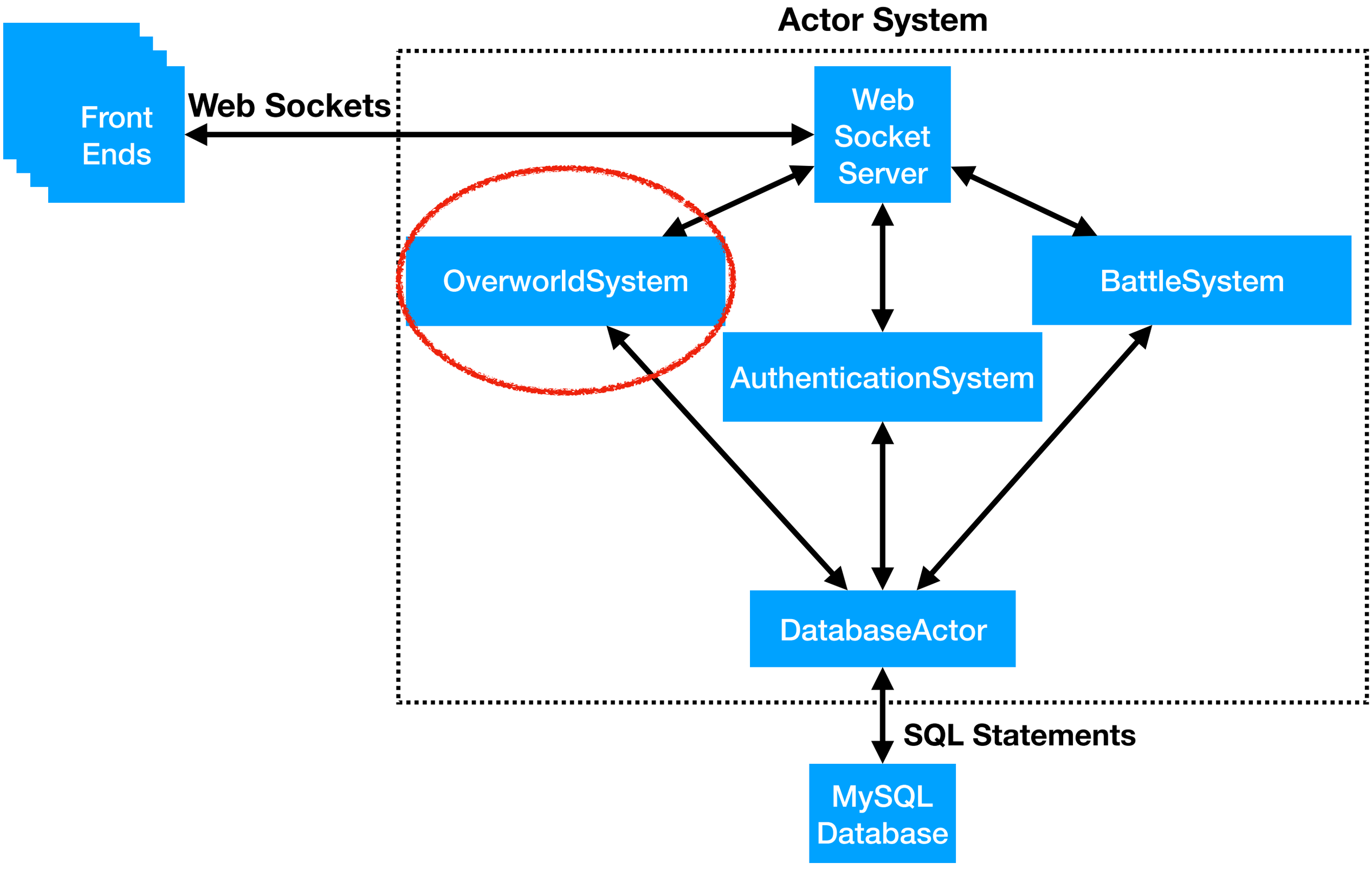
- Notified when a user connects via a socket connection
- Listens for register messages
    - Create an account and store it in the database (Always hash+salt the password)
    - Creates/receives character types for each party member
- Listens for login messages
    - Verifies the username/password
    - Sends a verified message back to the server if the hashes match
    - Loads information associated with the account

AuthenticationSystem

- Only store salted hashes of passwords
  - Need to find a library for hashing and salting

- Insecure handling of user passwords is **not** acceptable

Project Architecture



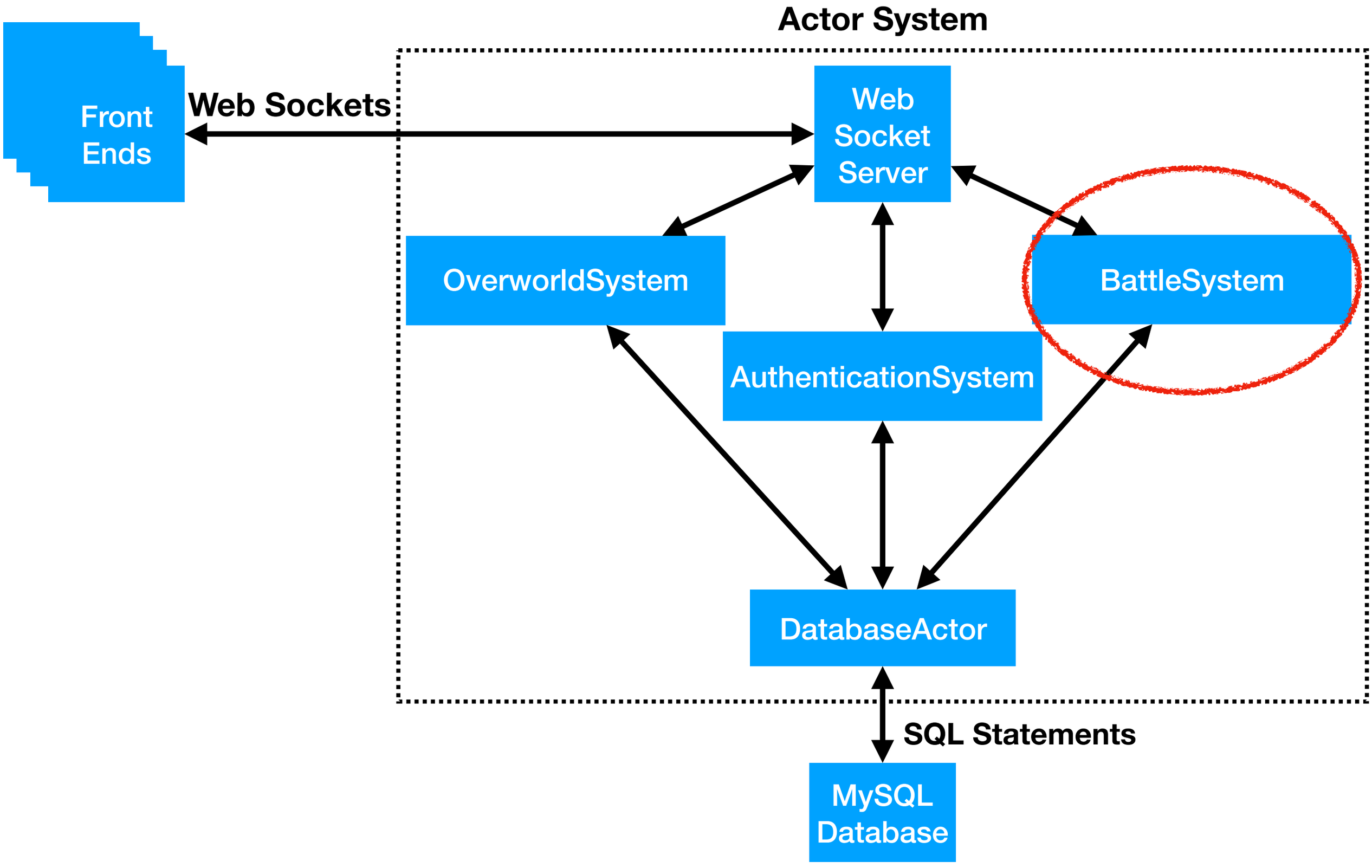
OverworldSystem

- Controls the overworld mechanics
- Creates, stores, and shares the world map
- Receives movements information for each party from the server
- Updates and shares the location of each party on each update
    - Detects collisions and notifies the server that a battle is starting
    - Doesn't allow parties to move through impassible tiles of the edge of the map
- Can use PhysicsEngine for the movement and collision detection

OverworldSystem

- Overworld is only concerned with the movement of each party
- The overworld does not, and should not, know anything about the stats of each party related to battles
- Overworld can be sent the level of each party to make the party locations JSON easier to make
    - Recall the overworld GUI needs the level of each party

Project Architecture



BattleSystem

- Controls all the battle mechanics
- The only system with an instance of the Party class (From demo 1) for each party
    - Party contains mutable variables so it must not be shared across actors (unless you want really strange bugs)
- Creates a Party object for each player when they log on
    - Stores this object even when they are not in battle

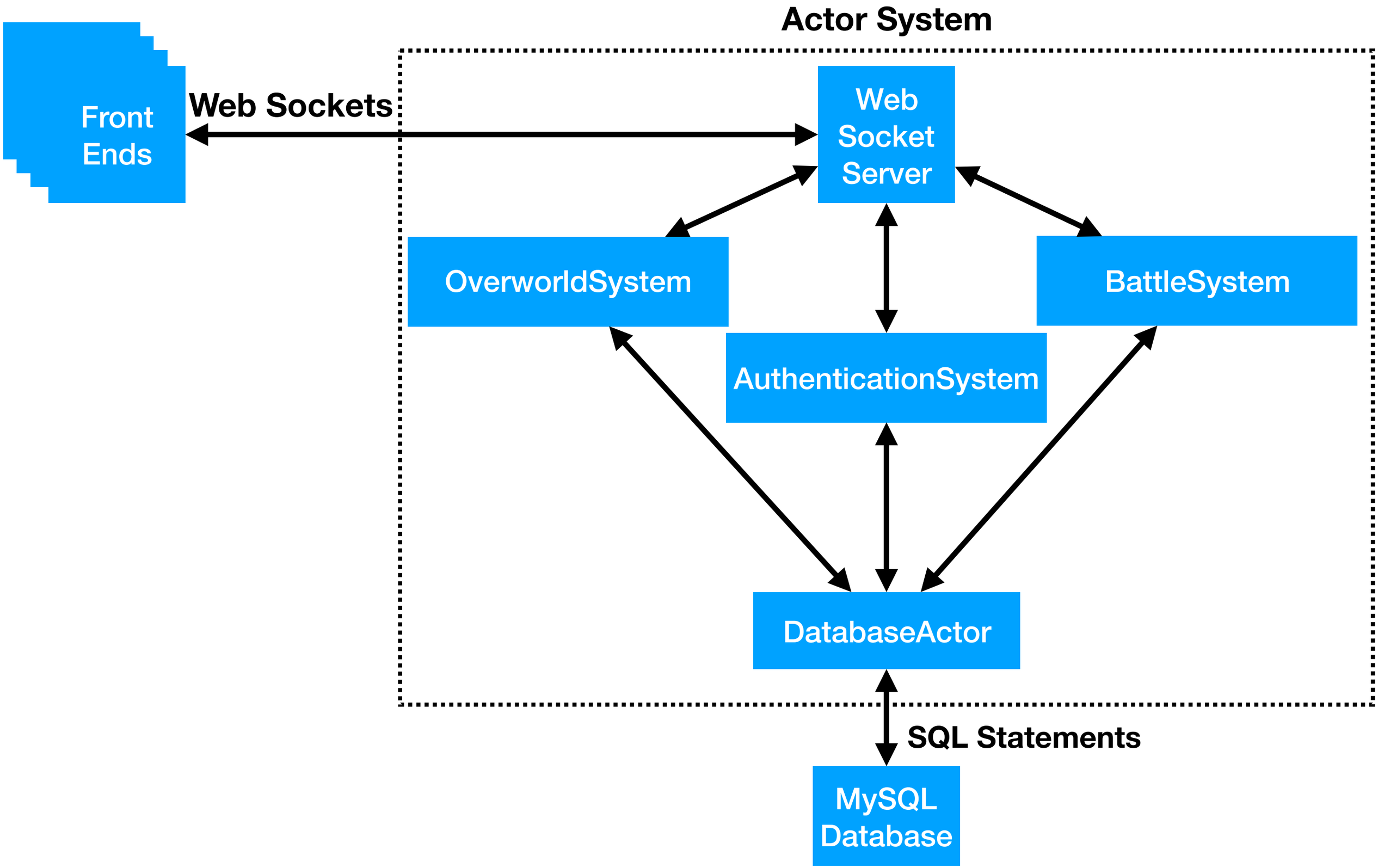
BattleSystem

- Receives a battle start message and initializes a battle
- Receives action taken messages each time an action is chosen
    - Call the attack methods from demo1/ LA4 to process the action
    - If the action ends the battle, call the experience methods and notify the server that the battle has ended

BattleSystem

- Maintains an Active Time Battle (ATB) System for each battle
  - A turn-based system would introduce a lot of waiting in an MMO
- Each character in the battle has a battle timer to decide when it is their turn
- When enough real time has passed, notify the server that a turn is ready and the player will be notified with their options
- When one player is choosing their options, they can be attacked by the other party

Project Architecture



Lecture Question

Google form right now