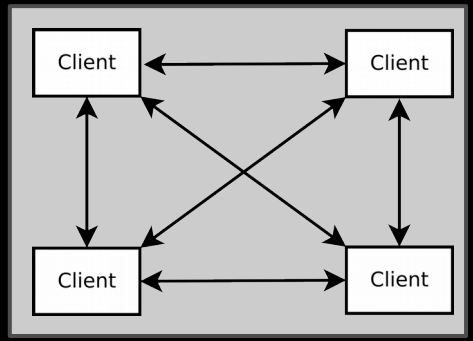
ARM Game

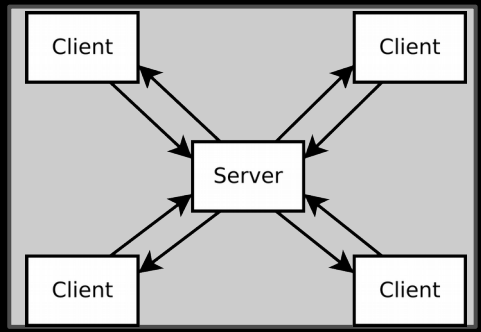
Asynchronous Real-time Multiplayer Game

by
Ravjot Singh and Glen Berseth

Introduction

1. Multiplayer games systems
    a. Peer-to-Peer (Lockstep)
        i. All players must start with same initial state
        ii. Lots of broadcasting
        iii. Latency
        iv. Cheating
    b. Client-Server
        i. Clients can join
        ii. Less messaging
        iii. Not fault tolerant
        iv. No Cheating!





Motivation/Design Goals

1. Want fault-tolerant solution
    a. Where players can join/leave whenever
    b. Real-time
    c. Tolerant to cheating
    d. Scaleable

2. Would make an awesome MMO game

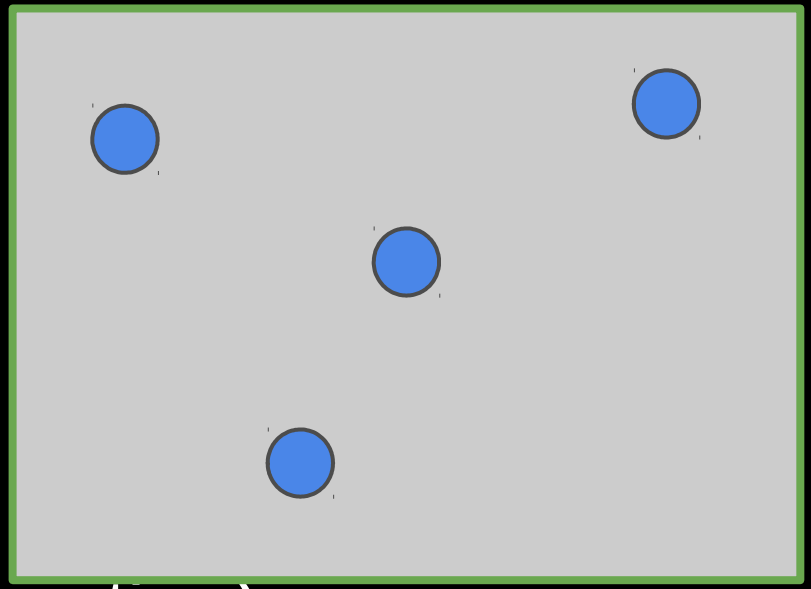
Problem Space

1. How to make a fault tolerant client-server?
    a. Real-time
        i. Each client simulates its own game
        ii. Sends updates to server
    b. Authoritative Server
        i. Synchronizes state between clients
    c. Distribute the server
        i. Consistency Issues
    d. Cheating

Model and Assumptions

1. Security model
    a. Clients can’t collude
    b. Clients can’t hack servers
    c. Can send misinformation
2. Network
    a. No partitioning
    b. Unlimited Bandwidth
    c. Asynchronous communication
3. Fate Sharing

ARM Game



Game Semantics:

- join()
- updateLocation(player,location)
- fire(player,location,direction)

Software each player is running?

Player 1

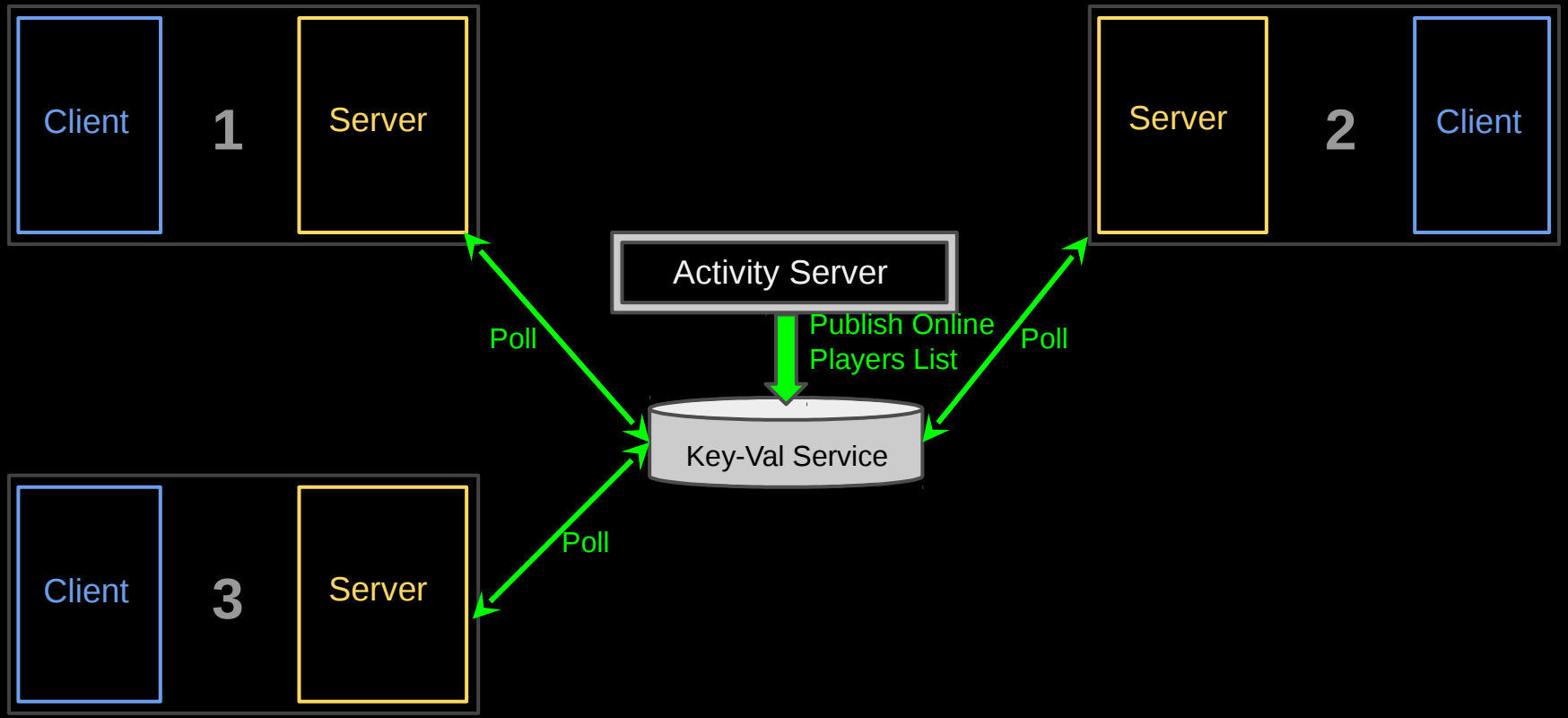
Client

- Game State is guided by Player/AI

Server

- Game State is guided in coordination with other servers.
- Responsible for:
  - validating locationUpdate()
  - validating own termination due to Fire

Base Architecture Working

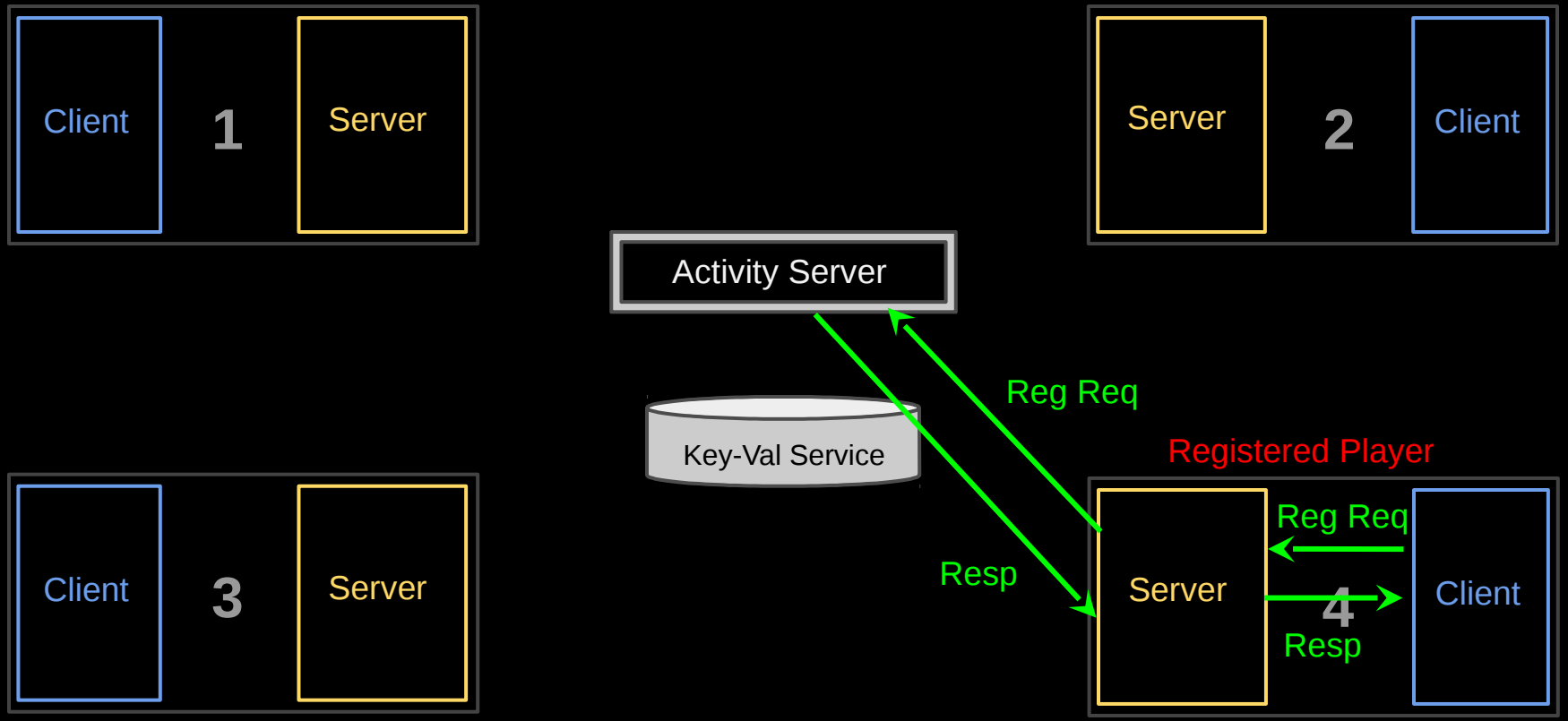


Joining

Joining is done in two steps:

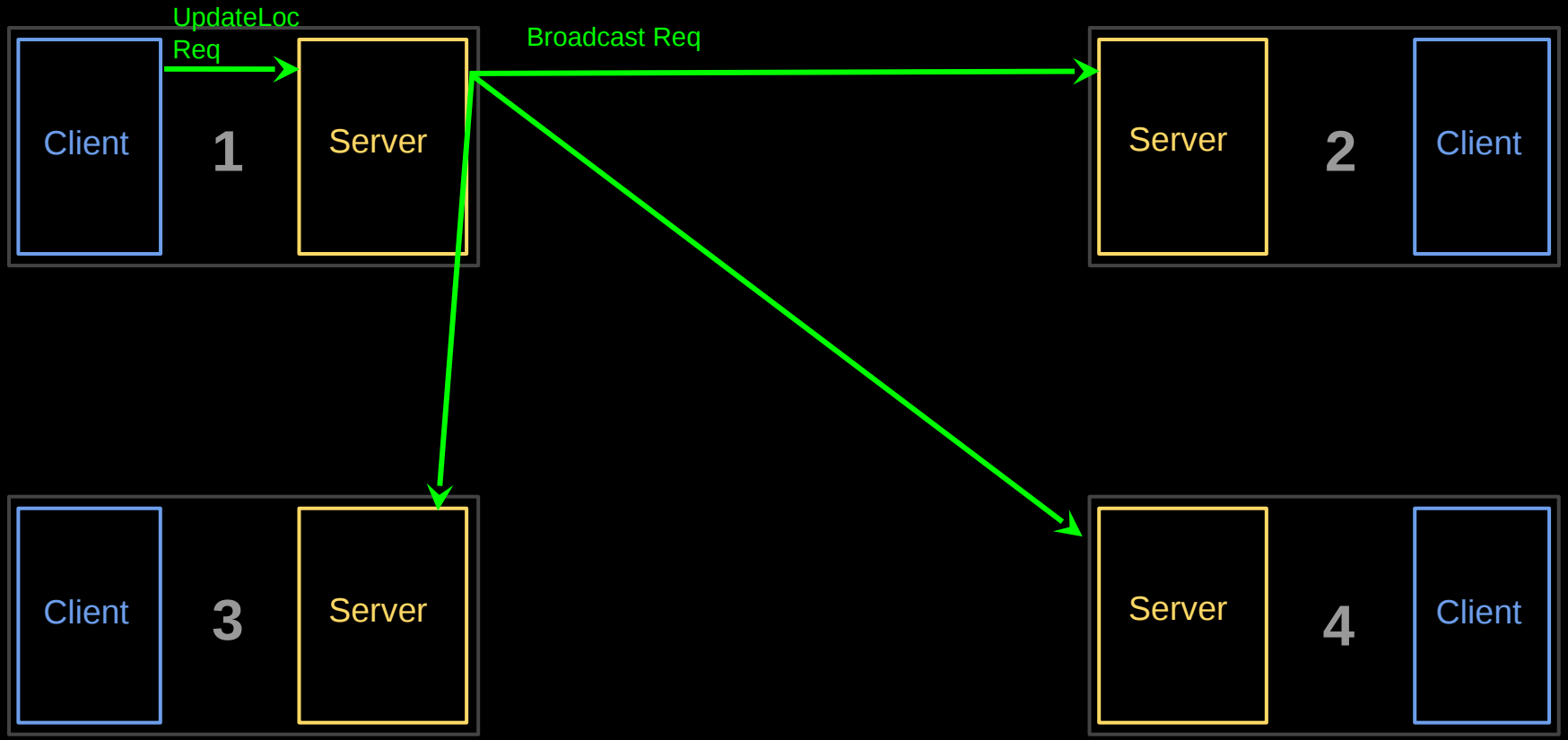
1. Registration
2. Game State Construction

Registration

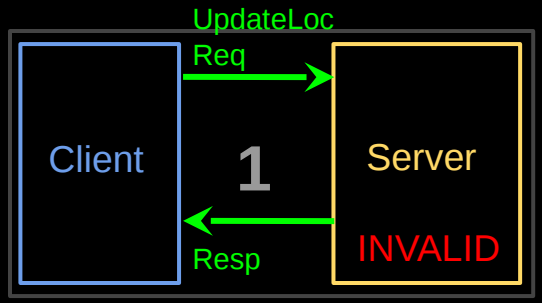


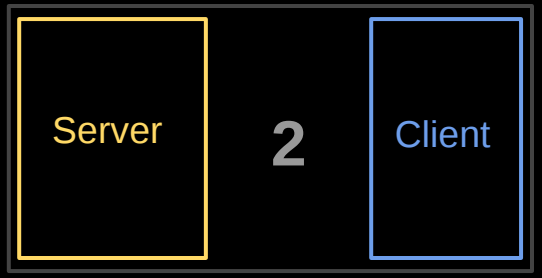
Game State Construction

Move (x,y)

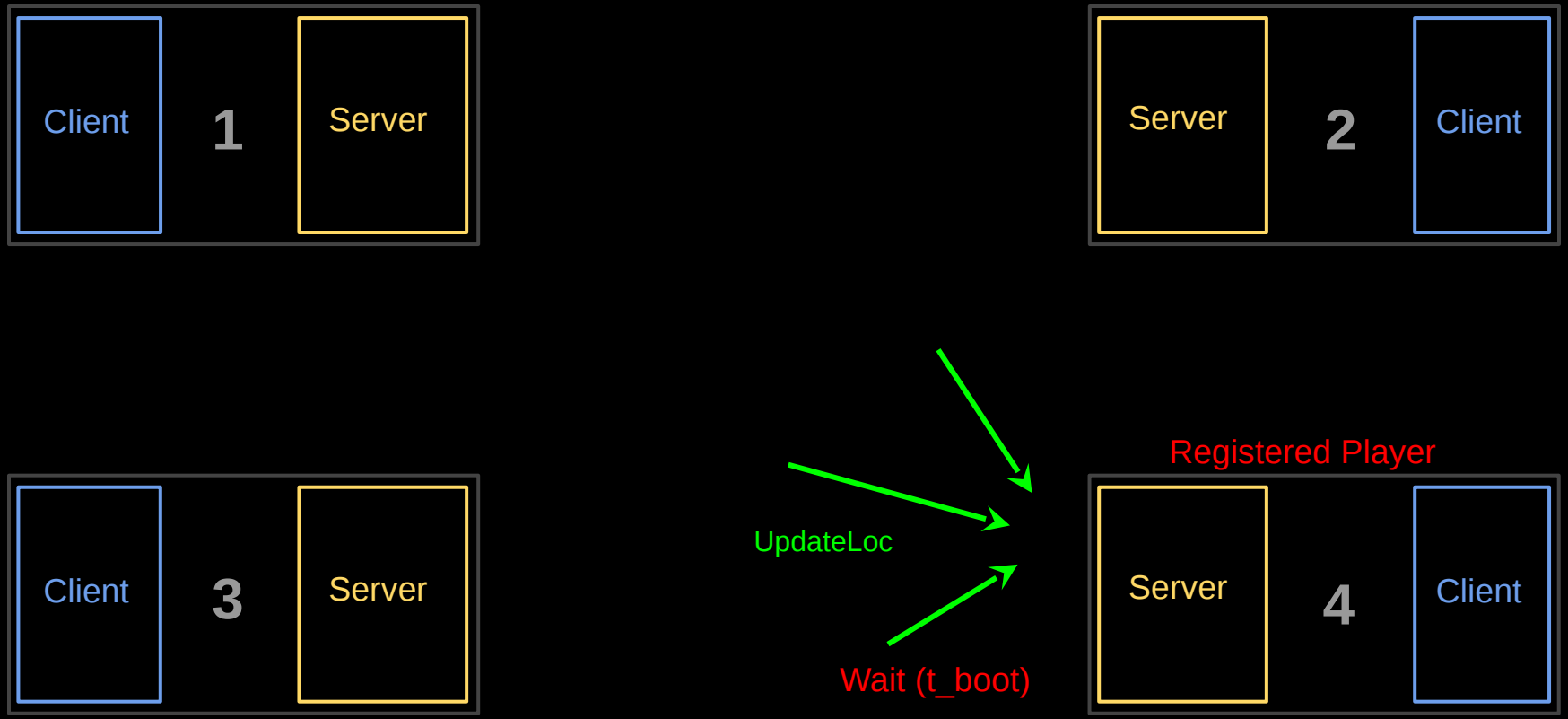

UpdateLoc
Req
Broadcast Req
Client
1
Server
Server
2
Client
Client
3
Server
Server
4
Client

Invalid Move (x,y)

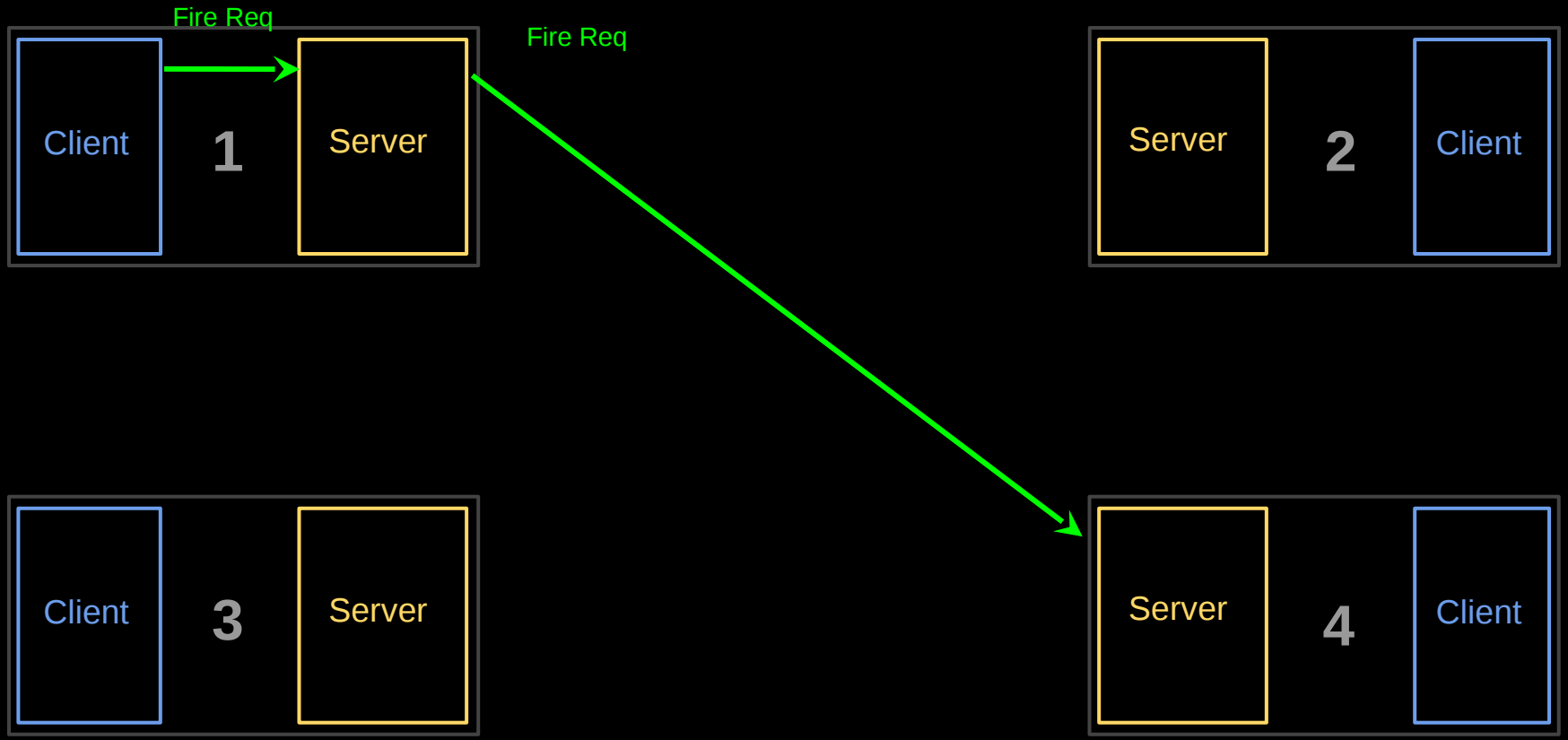




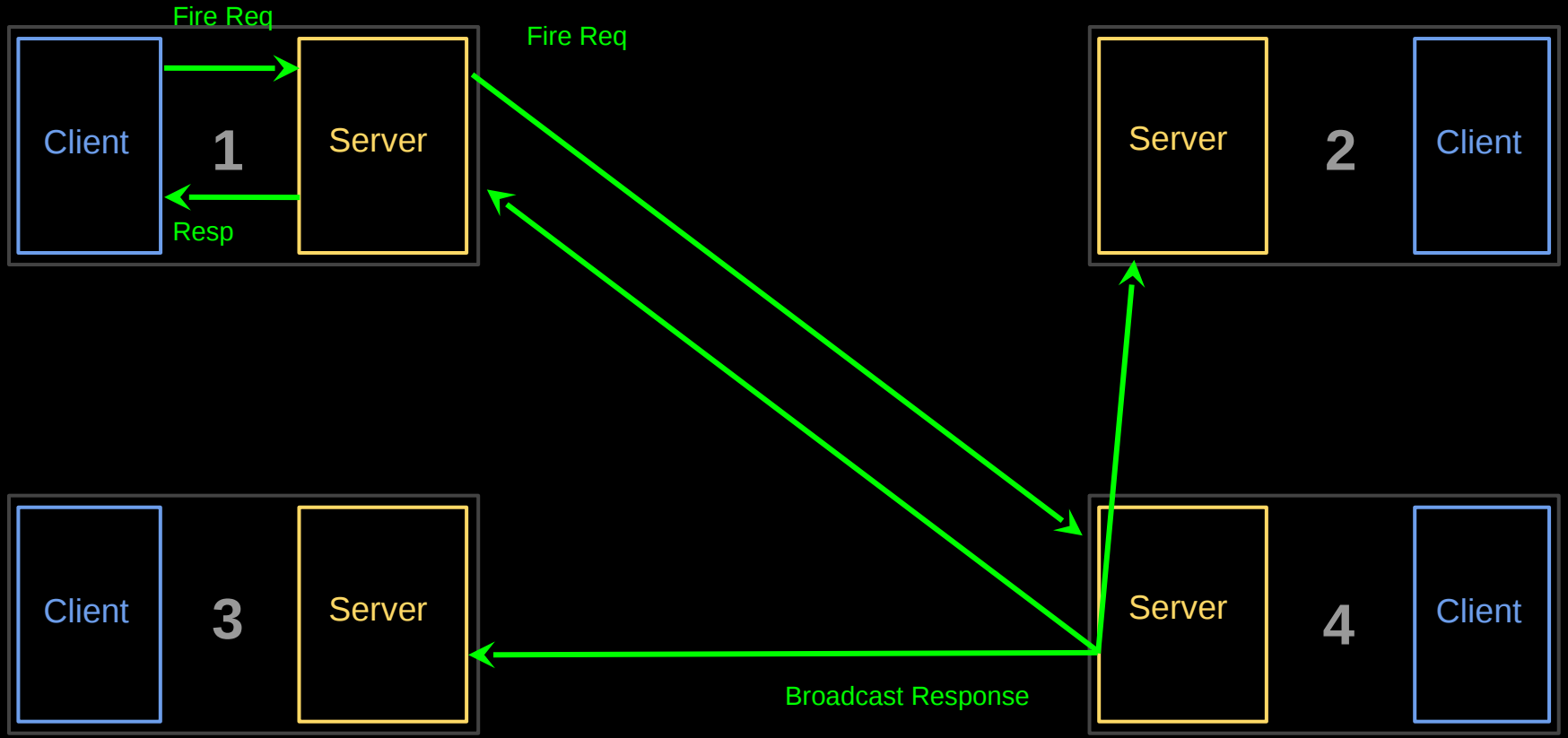
Game State Construction



Fire (ownLoc, dirn, xpctedHit)



Fire (ownLoc, dirn, xpctedHit)



What we Achieved!

Scalable:
- no bottleneck (everybody responsible for themselve)
- optimistic approach - minimal msgs

Minimal Latency
- optimistic approach - minimum msgs

Fault tolerant
- Since each node is simulating its own game, system can tolerate n-1 faults in n node system

Consistency
- sacrificed consistency to gain scalability and reduced latency

Questions ?

Design Goals

Fault tolerant (??)
Minimal Latency
Lose Consistency
End-to-End Argument
Scalable

Model and Assumptions

1. Fate Sharing
    a. If a client goes down, so does the server.
2. Synchronized Clocks
    a. Not necessary
    b. Vector clocks (for each client)

ARM Game

1. Agent locations
2. World bounds
3. maximum velocity
4. Shooting rate
5. Shooting distance
6. Shooting direction
7. Artificial Intelligence