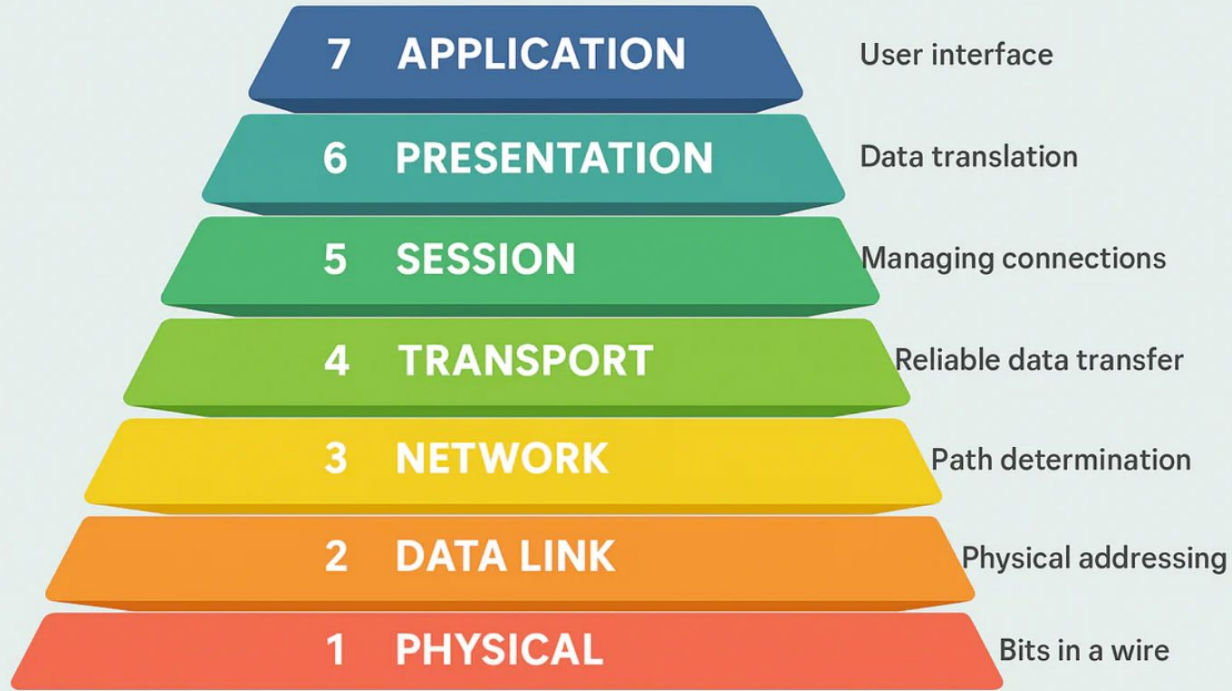


# Networking

**Introduction to Game Development in Unity**  
**Spring 2026, 98-127, Lecture 13**

Instructors: Jingxuan Chen, Dario Quintero, Shangyi Zhu, Jeffrey Wang

# THE OSI MODEL



The OSI Model: computer systems use to communicate over a network

# Protocols (allow devices to communicate)

## TCP (Transmission Control Protocol)

- Guarantees that all transmitted data will reach its destination in the correct order
  - automatically retry (send data) until all data is successfully transmitted
- lags( delays in data transmission)
- Handshake 3 time(send to server, receive back, and then send again)
- Youtube video buffering

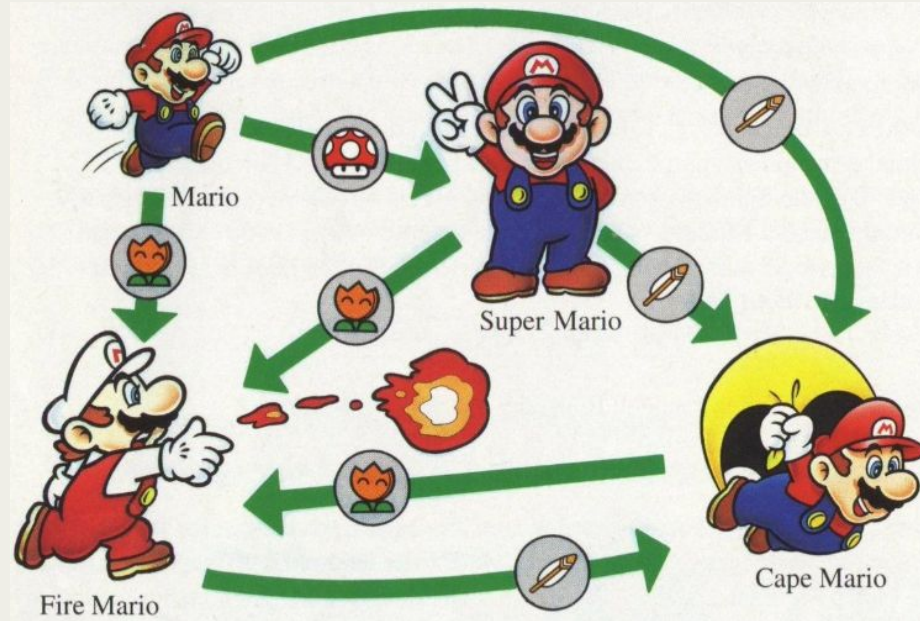
## UDP (User Datagram Protocol)

- Faster
- Does not guarantee deliver or packet order
- Streaming pixel glitching

## RUDP(Reliable User Datagram Protocol)

- Combine UDP speed with delivery order and prevention of data loss from TCP
  - Get confirmation that data packet has been recieved with cached previous data
- **Unity Transport Package**

# Data sent over the Network



## States

Current ground true of the game

- position
- velocity
- health
- score
- who is alive

## Event

Action that has happened/attempted

- player fired
- player was tagged
- round started
- item picked up

## Inputs

Physical controller inputs

- move left
- jump
- press space attack

# 2 Main Types of Synchronization

## Frame Synchronization

- Send **inputs** to the server
- Server sends all player inputs to each client
- Client side computes all of the logic
- Suitable for fighting games(usually much less players), MOBA, cinematic heavy games

## State Synchronization

- Send **states** to the server
- **Server Authority:** The server runs the definitive physics simulation and game logic.
- Send the data back to each player
- Suitable for FPS, MMORPG, Puzzle games/board games

# Multiplayer Deconstruction

## Service

account identity  
invites  
lobbies  
matchmaking/session discovery

Unity Authentication + Unity  
Lobby

## Connection - how data packets travel(udp, tcp)

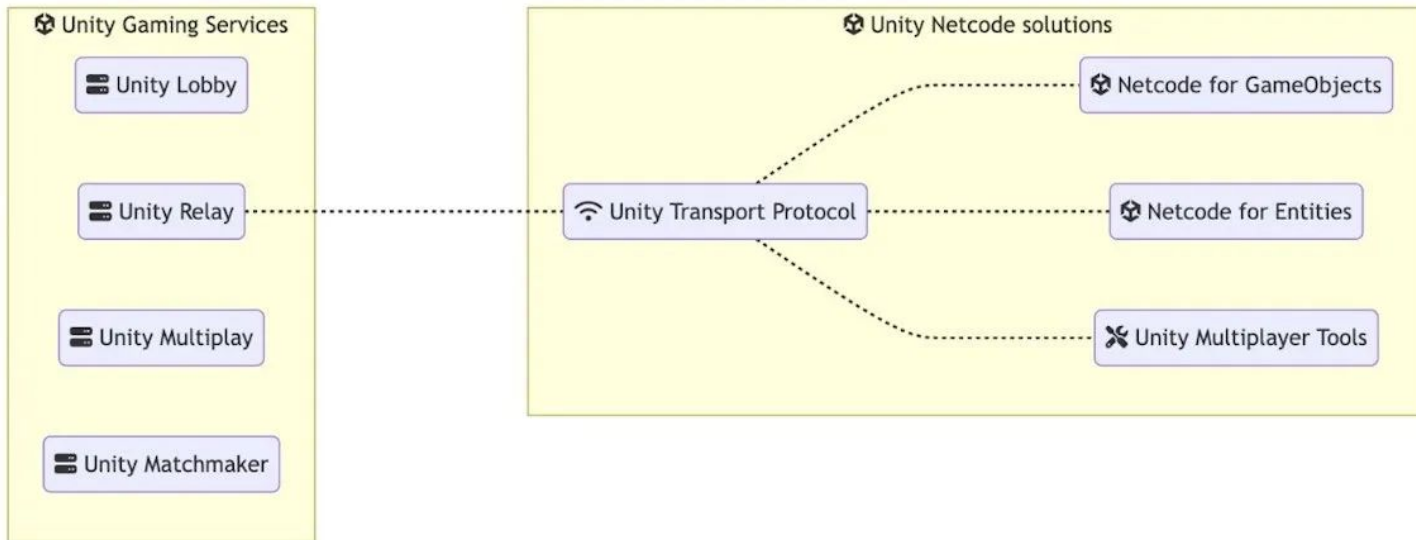
Unity relay + unity transport

## Netcode

synchronized objects  
state replication  
RPCs  
ownership

Unity Netcode for Game  
objects

# Netcode for GameObjects

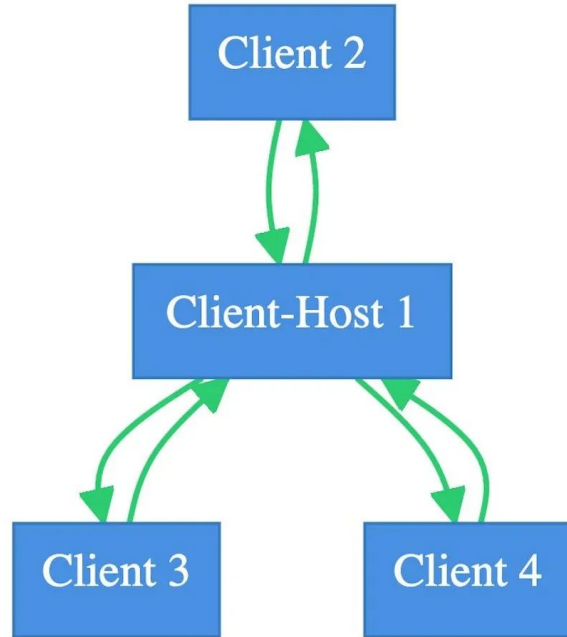


# Authority

Authority defines actions on the game world, and synchronizes the simulation to all other connected clients

# Host-Client Server A.K.A Listen server

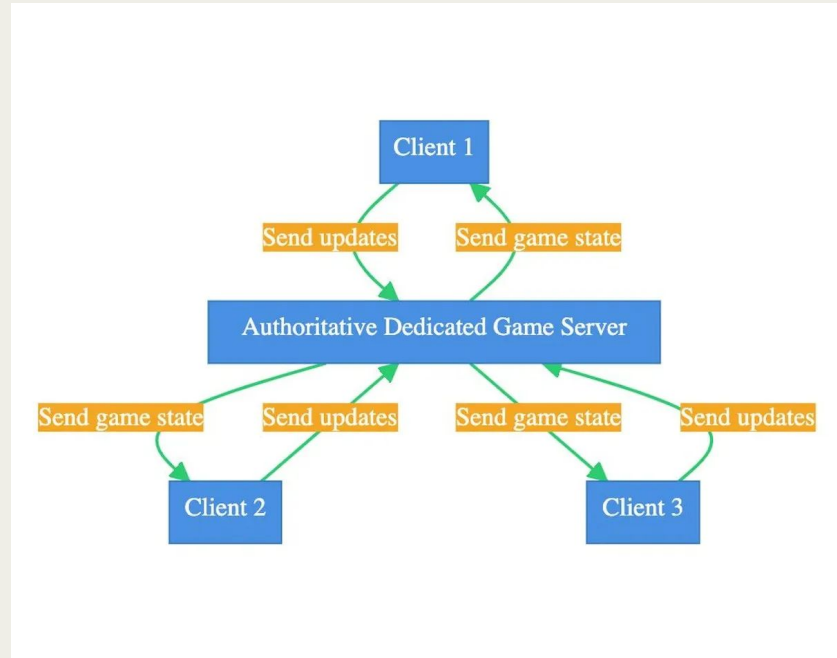
One player is the host that runs the server that writes, updates, and solve all the network states. All clients must send data to the one server



# Dedicated Server

Every client keeps track and update their own state, meaning different client could have different state, and synchornization by sending changes to every other client

Server has the authority and final say on what is the ground truth



# Peer-to-Peer

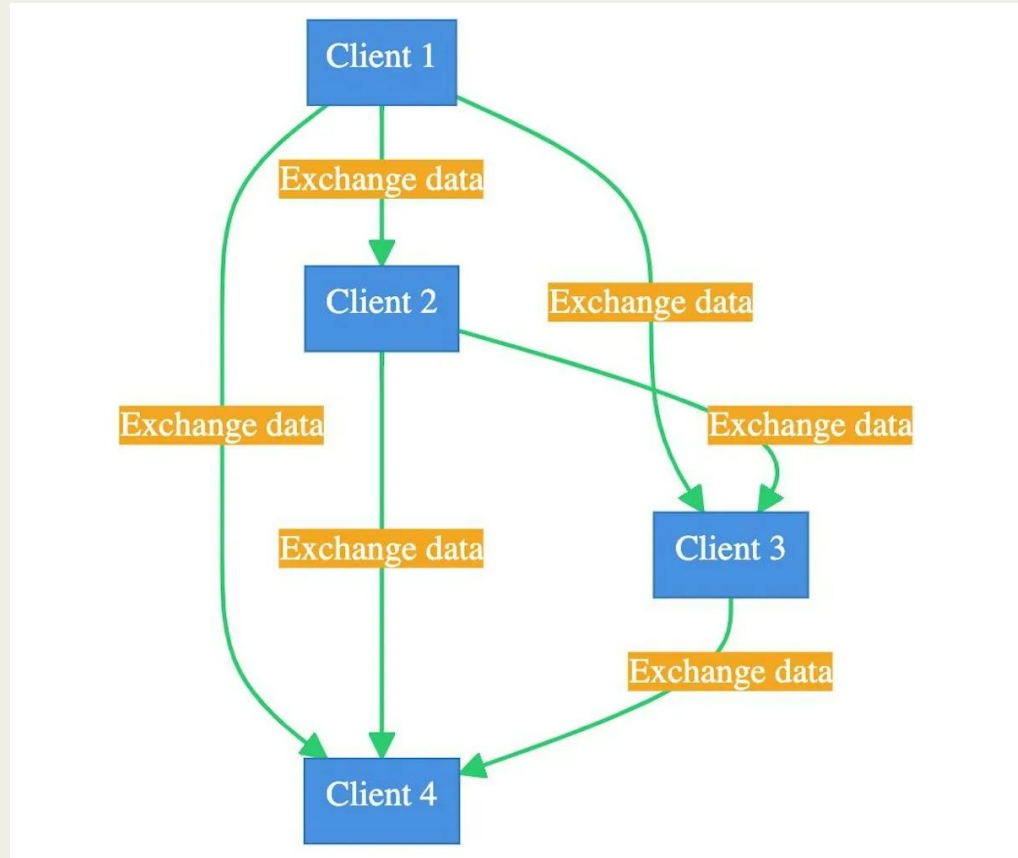
Each client(player) communicates directly with others.

Pros:

- potentially cheaper hosting
- simpler for very small casual games
- can work for turn-based or low-action games

Cons:

- cheating/security is harder
- connectivity issues
- Growing complexity as player count grows



# Hosting Multiplayer

1. Direct IP host-client(port forwarding)
2. (Service hosting)Steam host-client
3. Dedicated server



General area of  
Indie developers  
and game  
developers

Double AA and  
Triple AAA and  
larger games

### **Service**

account identity  
invites  
lobbies  
matchmaking/session discovery

**Connection - how data packets travel(udp, tcp)**

### **Netcode**

synchronized objects  
state replication  
RPCs  
ownership

## **1. Who helps players find and connect to each other?**

- Unity Lobby/Relay
- Steam lobbies/networking
- direct IP
- custom backend

## **2. Who runs the game simulation?**

- one player as host
- a dedicated server
- all players in a peer model

# Multiplayer Deconstruction

## Service

account identity  
invites  
lobbies  
matchmaking/session discovery

## Connection - how data packets travel(udp, tcp)

## Gameplay

synchronized objects  
state replication  
RPCs  
ownership

**Steam SDK API (any other service handler API)** will handle the identity, lobby and match matching, and network relay

Unity Netcode for Game objects

# Different types of Multiplayer games

# Turn-based Games(discrete decision-based multiplayer)

Peer-to-peer or Host-client

- No need for real time calculation(little bandwidth)
- No continuous state syncing
- Action happen in discrete sequences(turns)

**Sending inputs and events** instead of continuously updating state

# Turn-based Games(discrete decision-based multiplayer)

## Inputs / commands

- “Play card 7”
- “Move unit A to tile (3,5)”
- “End turn”
- “Attack enemy unit B”

## Events

- “Turn changed to Player 2”
- “Card resolved”
- “Unit died”
- “Round started”

## State

Recompute from commands and sync snapshots after each turn

- current board
- health totals
- whose turn
- hand size
- resource count

## 2.continuous real-time multiplayer(most common)

Listen server / host-client

- Requiring continuous updates fo states, meaning need one authority to sync the rest of the client to decide teh ground truth
- Real-time interaction between clients and the game that are viewed by multiple clients (collisions and hit checks matter) difference from turn base in
- players are acting at the same time
- the world is changing continuously
- multiple clients are viewing and affecting a shared live state
- you now need state synchronization over time
- 

Examples:

- tag
- co-op action
- minecraft
- party games

# Mid-speed action game like tag

## Inputs / commands

- “Play card 7”
- “Move unit A to tile (3,5)”
- “End turn”
- “Attack enemy unit B”

## Events

- “Turn changed to Player 2”
- “Card resolved”
- “Unit died”
- “Round started”

## State

Recompute from commands and sync snapshots after each turn

- current board
- health totals
- whose turn
- hand size
- resource count

# 3. latency-critical competitive real-time multiplayer

Similar criteria as the continuous real time game with a key difference:

**Latency (lag) and timing of inputs affect gameplay - think frame data**

you need prediction, reconciliation, lag compensation, or rollback as naive state sync is not enough of sending to a server is too slow

Examples:

- FPS
- fighting games
  - Smash games
  - Street fighter

# Fighting game - speed is most important

Peer-to-peer rollback, or rollback over a relay

- inputs are the most important data
- deterministic simulation is possible( take order of inputs and simulate the outcome
- rollback
  - correct a simulation when predictions turn out wrong.
  - It keeps historical states of the game for past frames and checks whether an input received for a past simulation frame matches the input prediction made at that frame. If it does, the simulation is correct up to that point; if it doesn't match, the simulation state is restored to the last known valid state, and all frames up to the current one are re-simulated using the fresh inputs.
- both players exchange **inputs every frame**
- each machine simulates locally
- if delayed remote input arrives, rollback and resimulate
- Fighting games usually synchronize player inputs and re-simulate deterministically, instead of constantly syncing full state like a shooter

# FPS game

listen server for casual and dedicated server for competitive

- hit validation is important
- host advantage is much more noticeable

Because FPS usually relies on:

- client-side prediction
- server reconciliation of states from clients
- lag compensation

# FPS

In FPS games, clients often send inputs, the server computes authoritative results, and clients smooth over latency with prediction and reconciliation.

## Inputs

- move forward
- strafe left
- jump
- fire weapon
- aim direction

## Events

- shot fired
- player hit
- player killed
- reload complete
- round start/end
- 

## State

- authoritative player positions
- velocities
- health
- projectile or weapon state
- world object states

# Netcode GameObjects

1. **NetworkObject:** This component, added to a GameObject, declares a prefab or in-scene placed object as a networked object that can have states synchronized between clients and/or a server. A NetworkObject component allows you to identify each uniquely spawned instance
2. **NetworkBehaviour:** allows synchronize state and write netcode scripts
3. **NetworkManager:** overall network session configuration and session management component. A NetworkManager component is necessary to start or join a network session.

# Network manager

- starting as host, server, or client
- knowing which prefabs are allowed to be network spawned
- connecting and disconnecting
- player prefab configuration
- transport configuration
- tick rate
- connection callbacks
- scene synchronization if enabled

Handles `OnClientConnectedCallback` and `OnClientDisconnectCallback`

# Network Synchronization

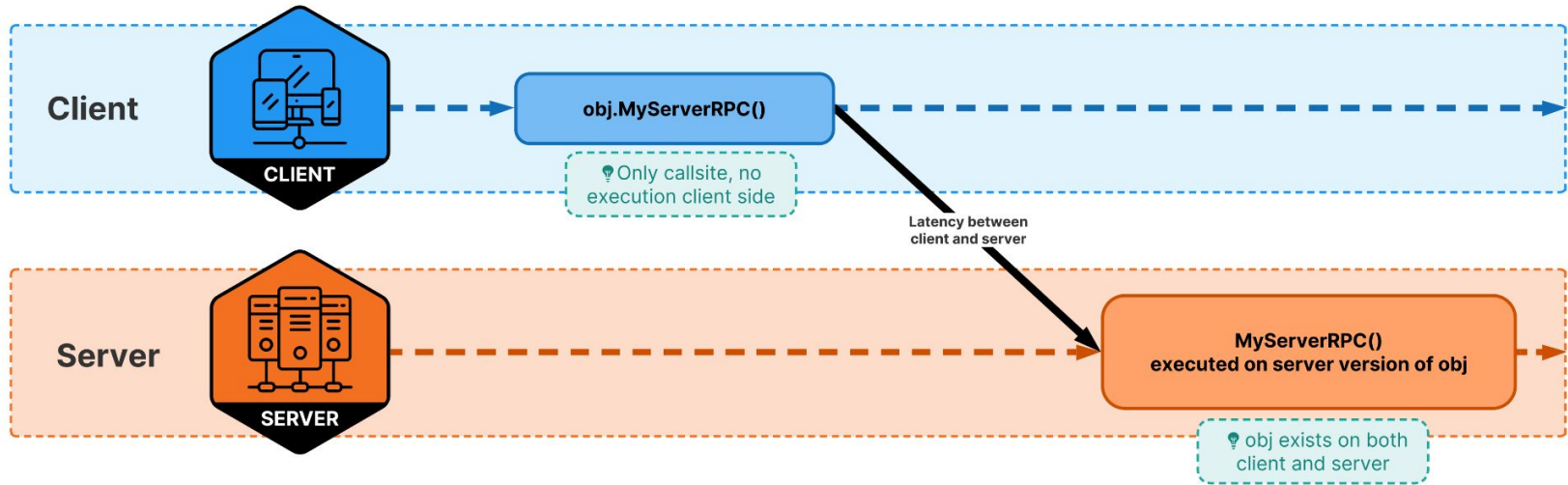
synchronizes where an object is, so other players can see it moving.

**NetworkTransform** - handles the transporting of movement between network objects

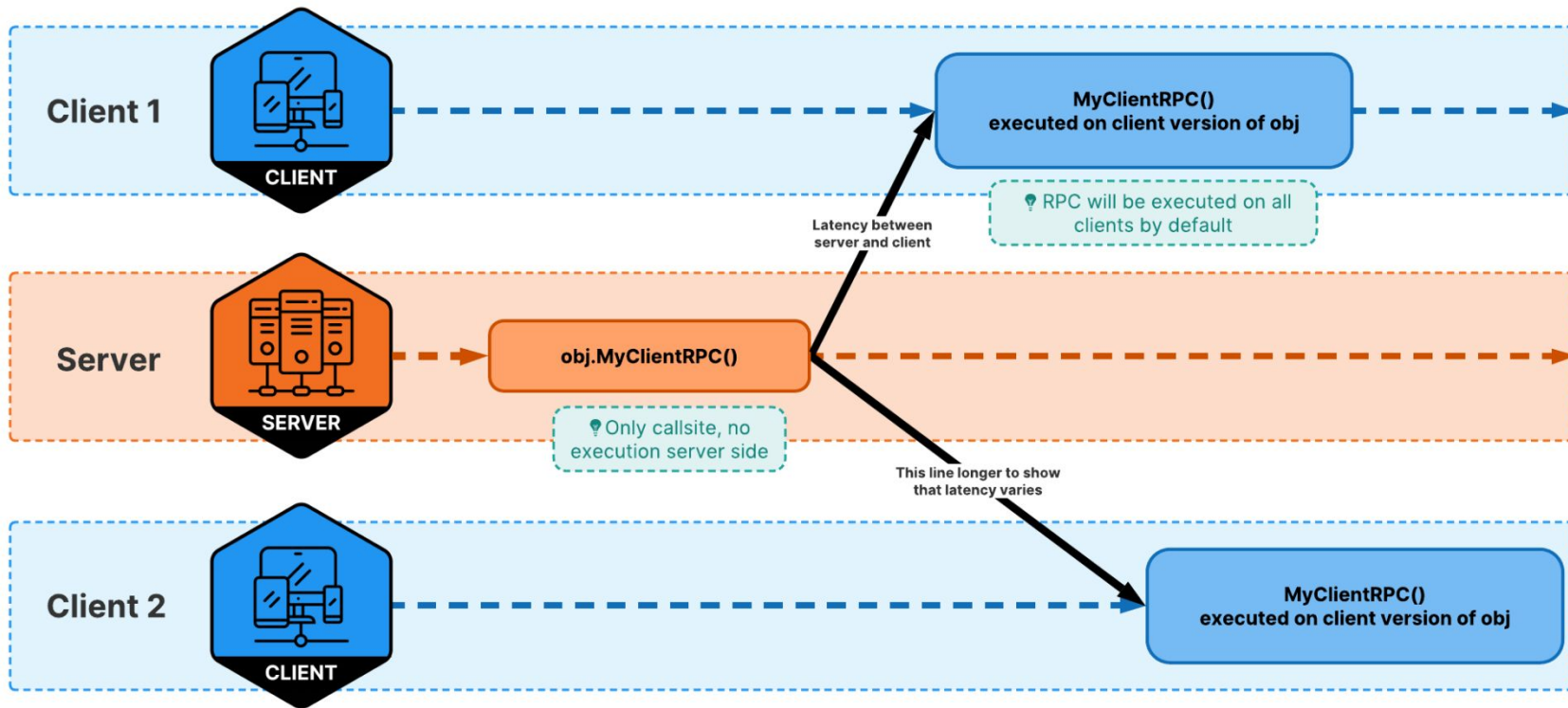
**NetworkRigidBody** - handle rigid body interactions surrounding network objects

# RPCS (Remote Protocol Call)

## Server RPCs



# Client RPCs



# Network variables

Equivalent of a save file (**ongoing game state**)

- alive/dead
- score
- match started
- who is “it”
- ready status in lobby

NetworkVariables are specifically for persistent synchronized state and automatically catch late joiners up to the current value.

**OnValueChanged** callback

## Event

Something that happens at a moment.

- pressed attack
- opened a chest
- played explosion effect
- requested to use a doom

## RPC

## Persistent state

Something that should stay true until changed.

- the door is open
- health is 60
- player is dead
- score is 3
- round timer is at 20 seconds
- player is ready in the lobby

## NetworkVariable

## Continuous Movement

Something changing over time that other players need to see continuously.

- player movement
- rotation
- moving platform

## NetworkTransform

# Resources

<https://unity.com/blog/games/how-to-choose-the-right-netcode-for-your-game>  
[Unity Realtime Multiplayer, Part 1: Networking Basics | by MY.GAMES | MY.GAMES | Medium](#)  
[Netcode for GameObjects | Netcode for GameObjects | 2.11.0](#)  
[About Multiplayer Tools | Multiplayer Tools | 2.2.8](#)  
[Multiplayer • Multiplayer • Unity Docs](#)