Multiplayer Game in C++

Improving latency handling

Networks and Online Games

Context

- Client server architecture
  - Server as a central node
  - Clients connected to server (star topology)
- Authoritative server
  - Server decides everything
  - Clients wait for server notifications to update world state
- UDP sockets
  - No reliable (packet loss, jitter) but fast

Problems that arise

- Laggy input response
- Abrupt movement of entities
- Non-expected reaction to inputs

Solutions to those problems

- Laggy input response: **Client side prediction**
- Abrupt movement of entities: **Entity interpolation**
- Non-expected reaction to inputs: **Lag compensation**

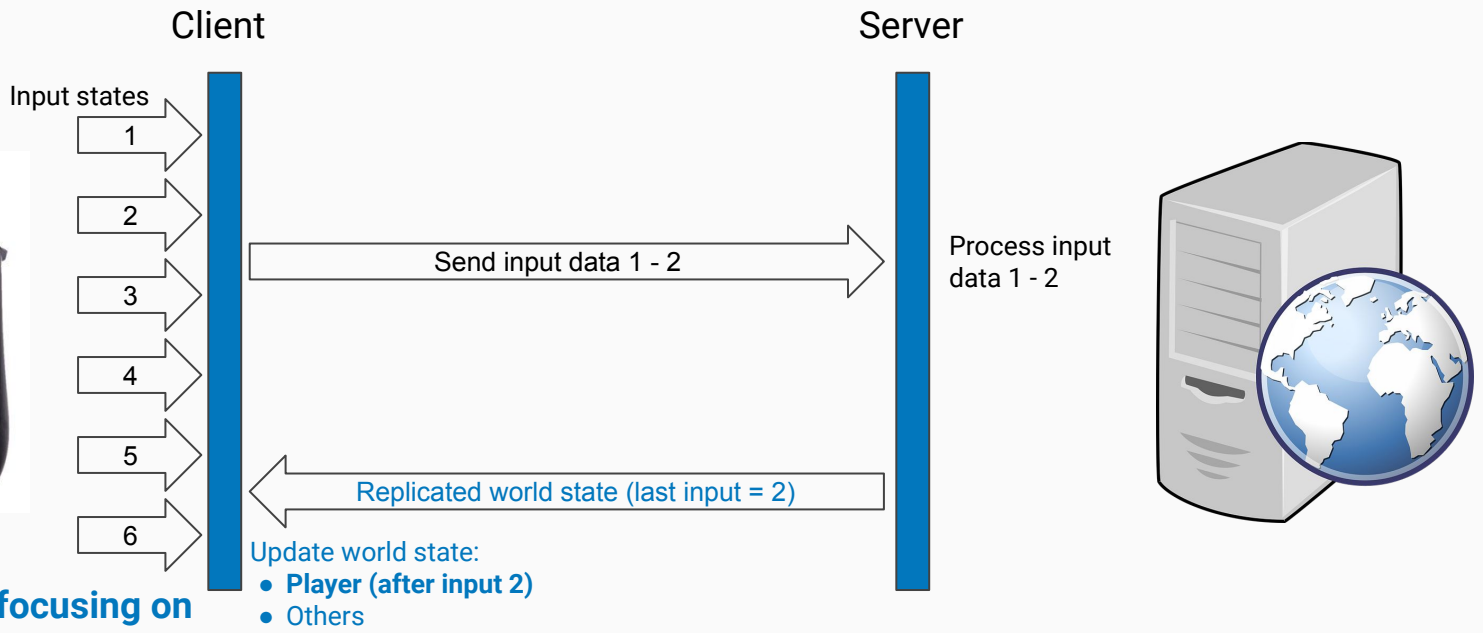
Client side prediction

Client side prediction

Client-side prediction is a network programming technique used in video games intended to conceal negative effects of high latency connections. The technique **attempts to make the player's input feel more instantaneous** while governing the player's actions on a remote server.

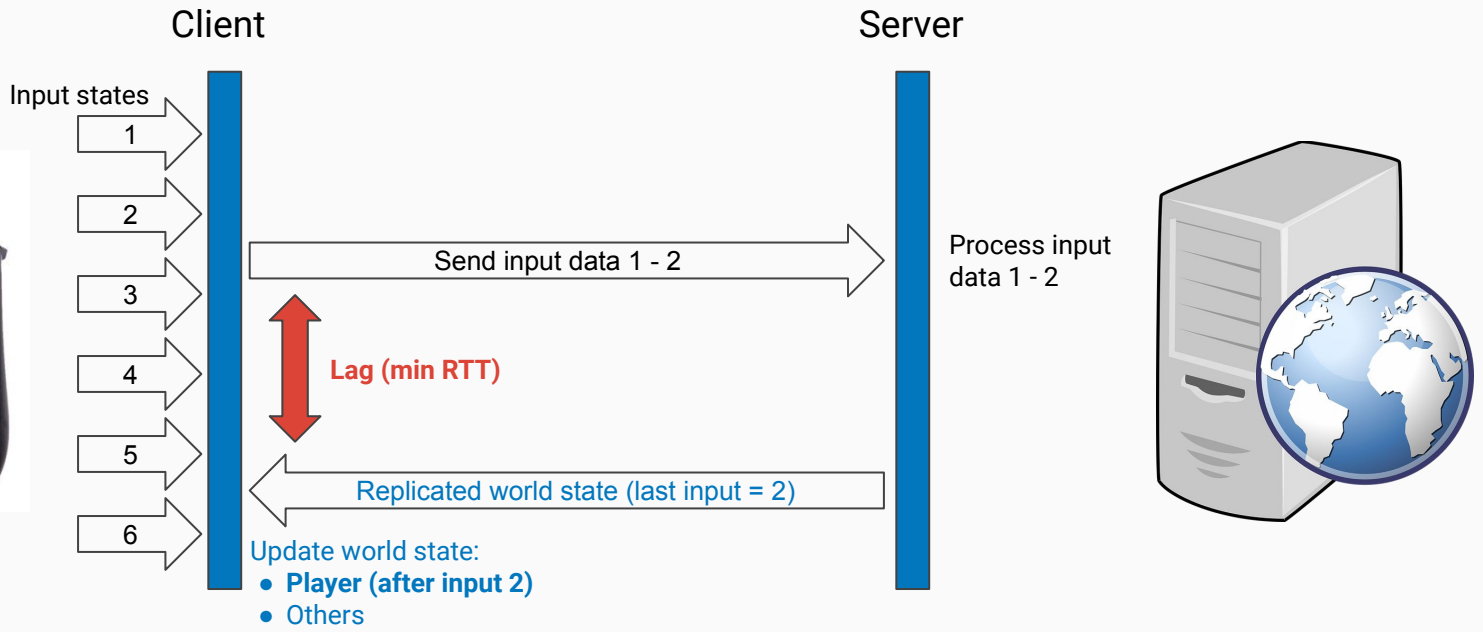
From Wikipedia, the free encyclopedia

The problem

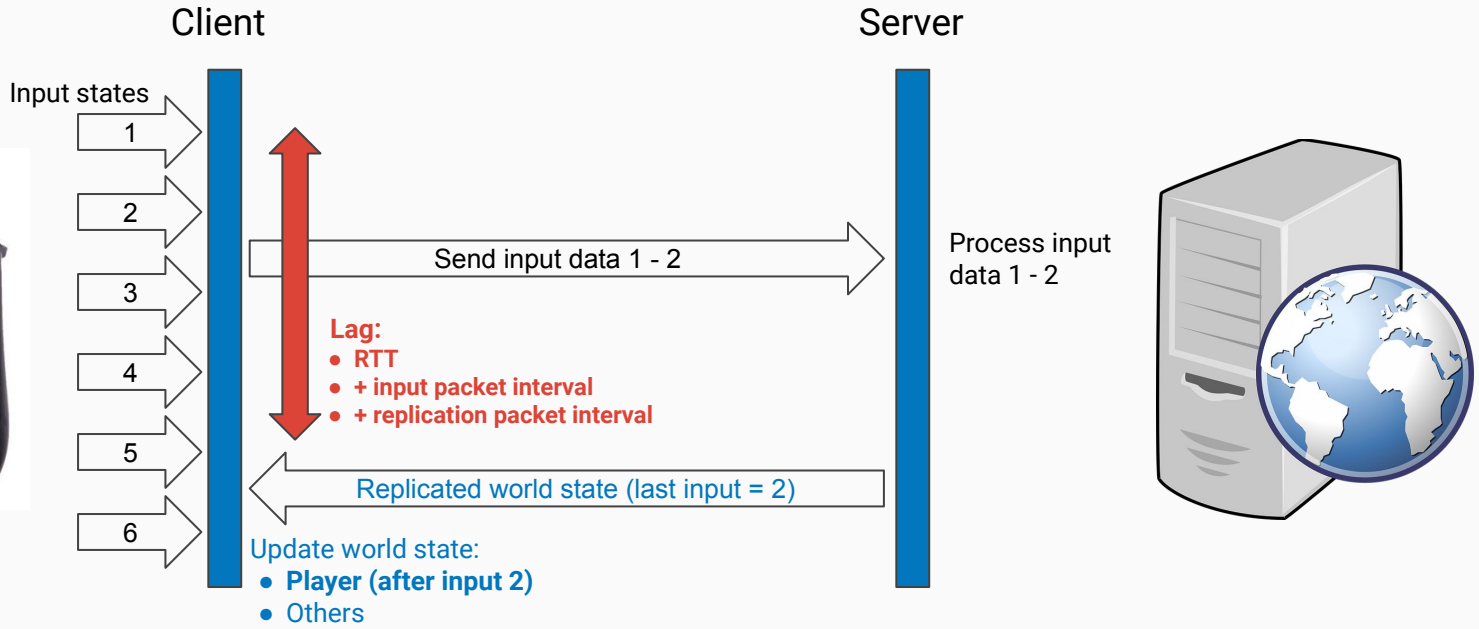


We are focusing on local player updates

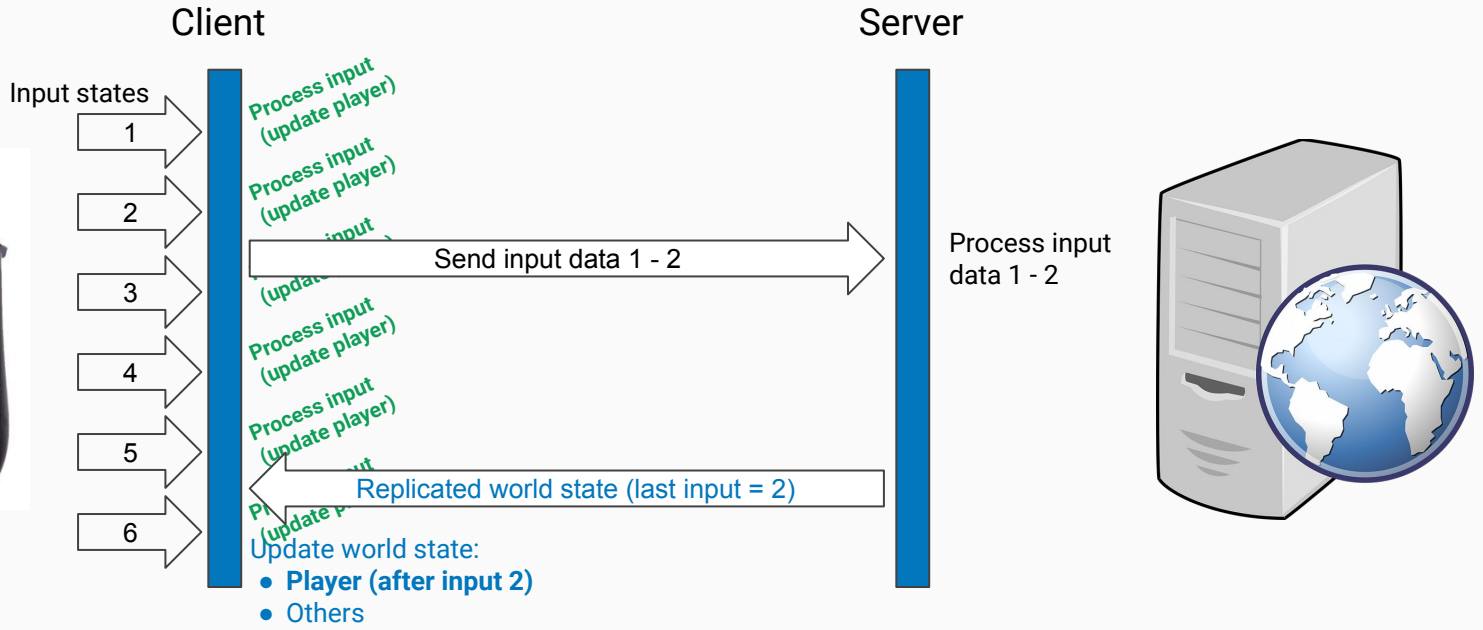
The problem


Client
Server
Input states
1
2
3
4
5
6
Send input data 1 - 2
Lag (min RTT)
Replicated world state (last input = 2)
Process input data 1 - 2
Update world state:
• Player (after input 2)
• Others

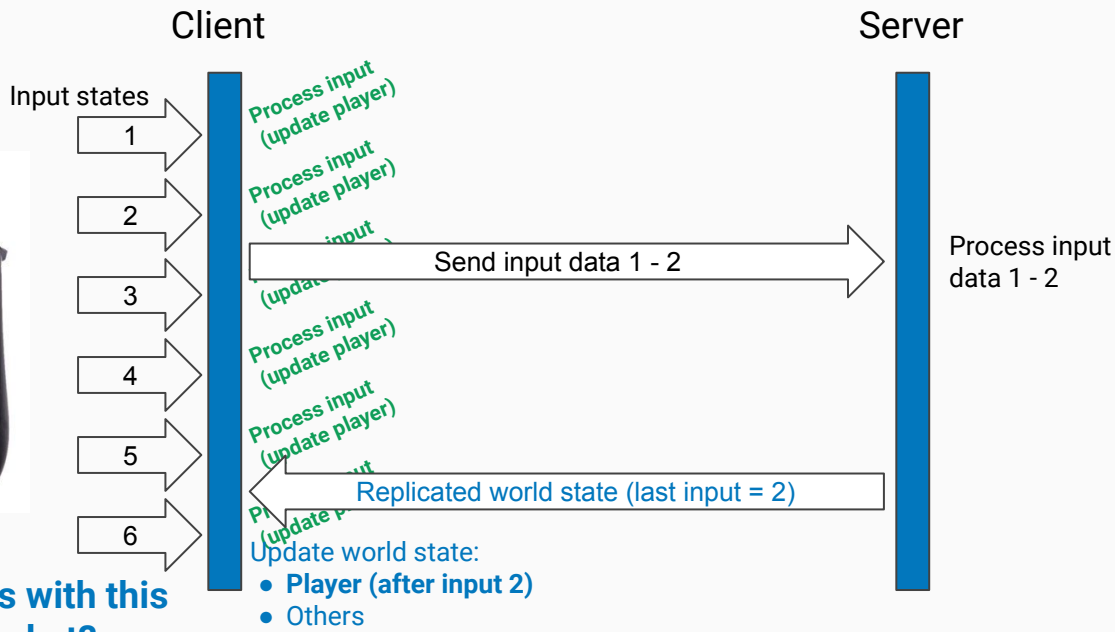
The problem


Client
Server
Input states
1
2
3
4
5
6
Send input data 1 - 2
Process input data 1 - 2
Lag:
RTT
+ input packet interval
+ replication packet interval
Replicated world state (last input = 2)
Update world state:
Player (after input 2)
Others

The solution: Client side prediction



The solution: Client side prediction



What happens with this replication packet?

First solution problems

Replication of world state

- Server is authoritative, it has the correct state of the world
- Updates all objects, including client controlled ones (e.g. the spaceship)
  - **Local simulations are lost on receiving replication packets**

Possible workaround

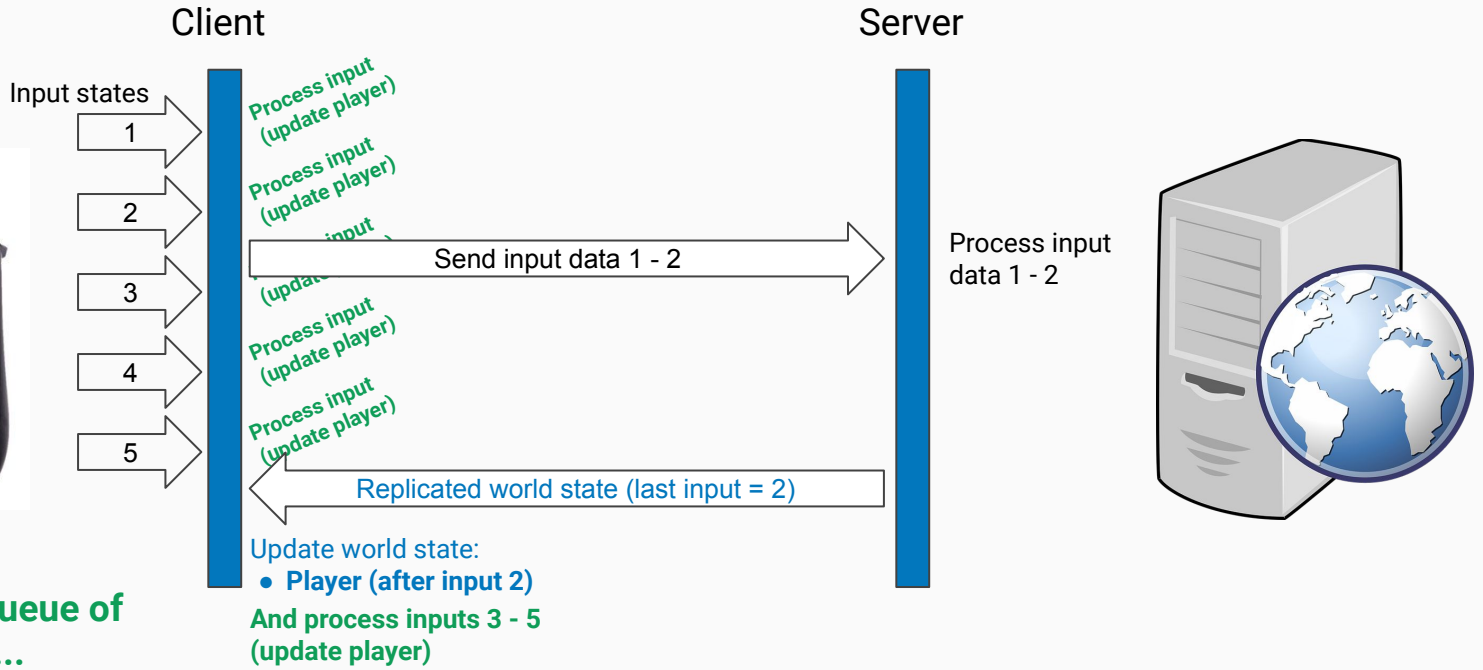
- Discard updates for client controlled game objects
- **What is the issue then?**

Final solution: Server reconciliation

Another solution to the desynchronization issue, commonly used in conjunction with client-side prediction, is called server reconciliation[2]. The client includes a **sequence number in every input sent to the server**, and keeps a local copy. When the server sends an authoritative update to a client, it includes the sequence number of the last processed input for that client. The client accepts the new state, and **reapplies the inputs not yet processed by the server**, completely eliminating visible desynchronization issues in most cases.

From Wikipedia, the free encyclopedia

Client side prediction + server reconciliation



We need a queue of input states...

Client side prediction + server reconciliation

In ModuleNetworkingClient.h

```
// Input ////////////

static const int MAX_INPUT_DATA_SIMULTANEOUS_PACKETS = 64;

// Queue of input data
InputPacketData inputData[MAX_INPUT_DATA_SIMULTANEOUS_PACKETS];
uint32 inputDataFront = 0;
uint32 inputDataBack = 0;
```

In ModuleNetworkingCommons.h

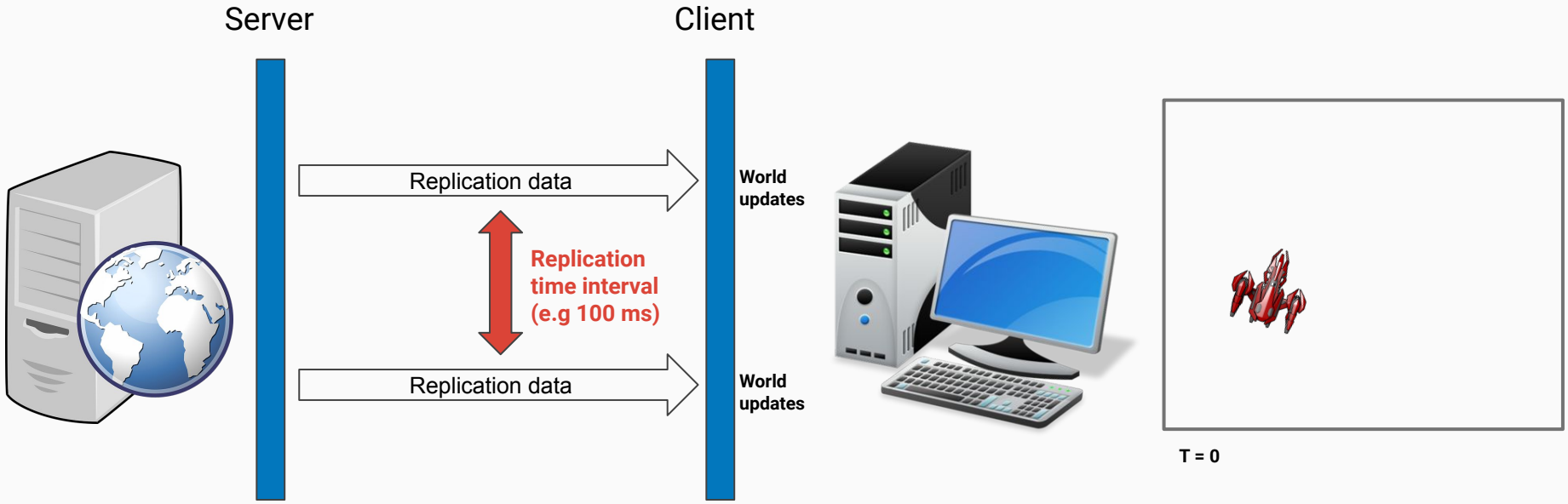
```
InputController inputControllerFromInputPacketData(const InputPacketData &inputPacketData, const InputController &previousGamepad);
```

Entity interpolation

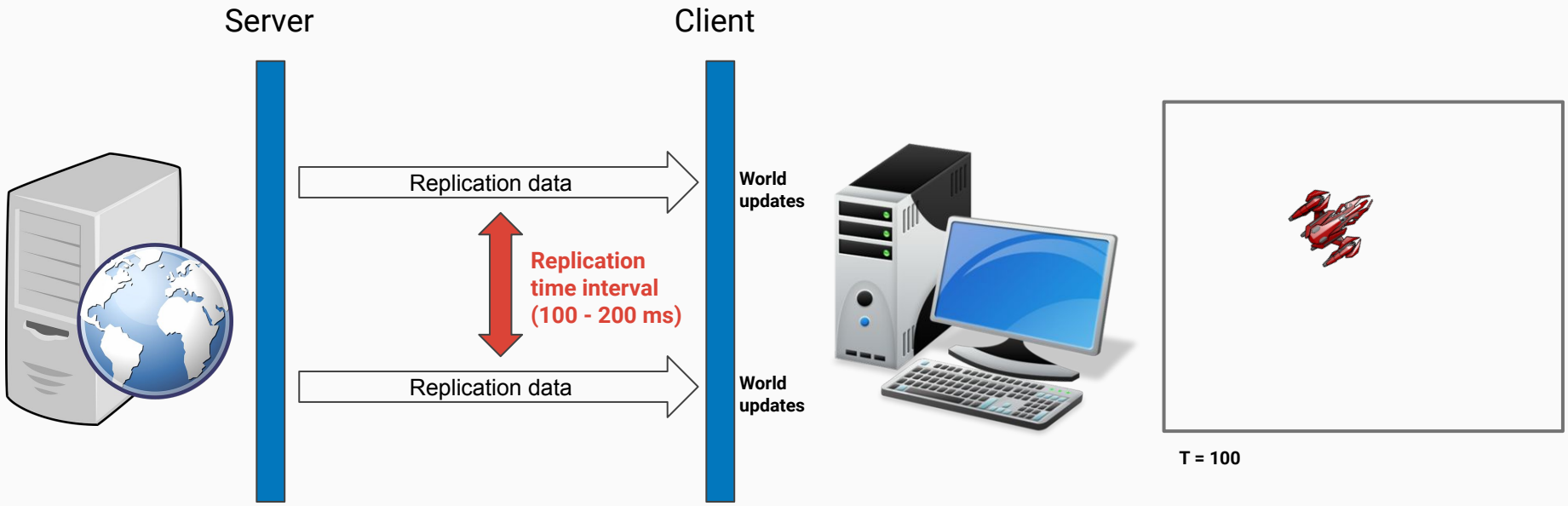
Entity interpolation

Client-side prediction is a network programming technique used in video games intended to conceal negative effects of non-continuous updates. The technique attempts to **make the player feel that networked objects controlled by other players are being continuously update,** even when receiving updates at a (relatively) low frequency.

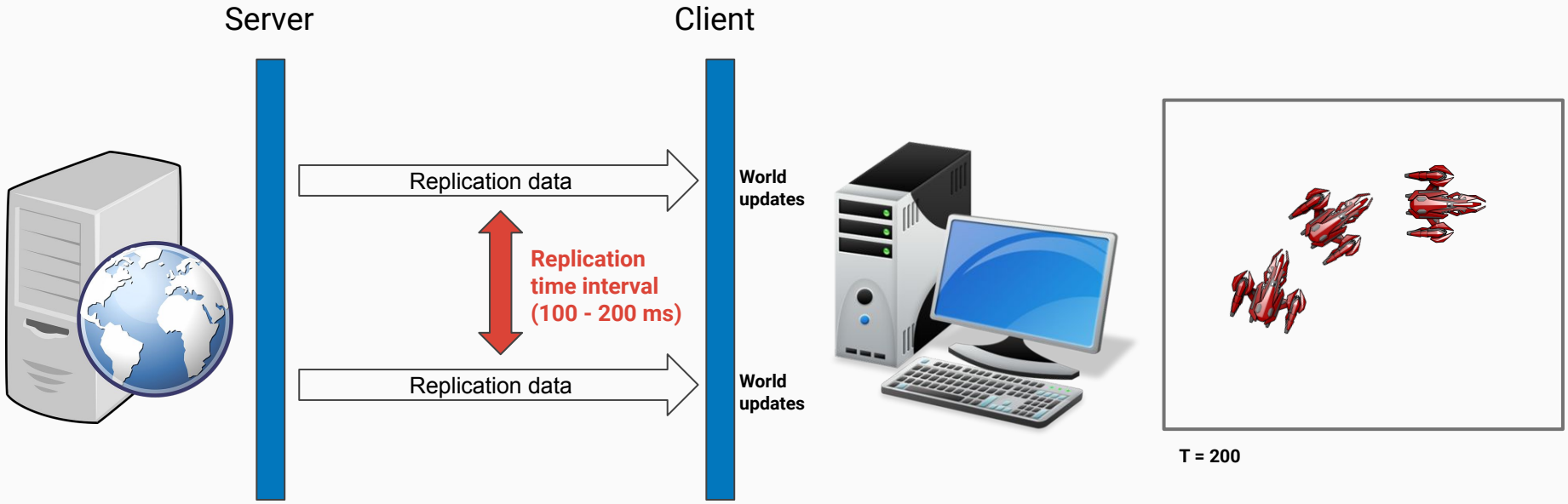
The problem


Server
Client
Replication data
Replication time interval (e.g 100 ms)
Replication data
World updates
World updates
T = 0

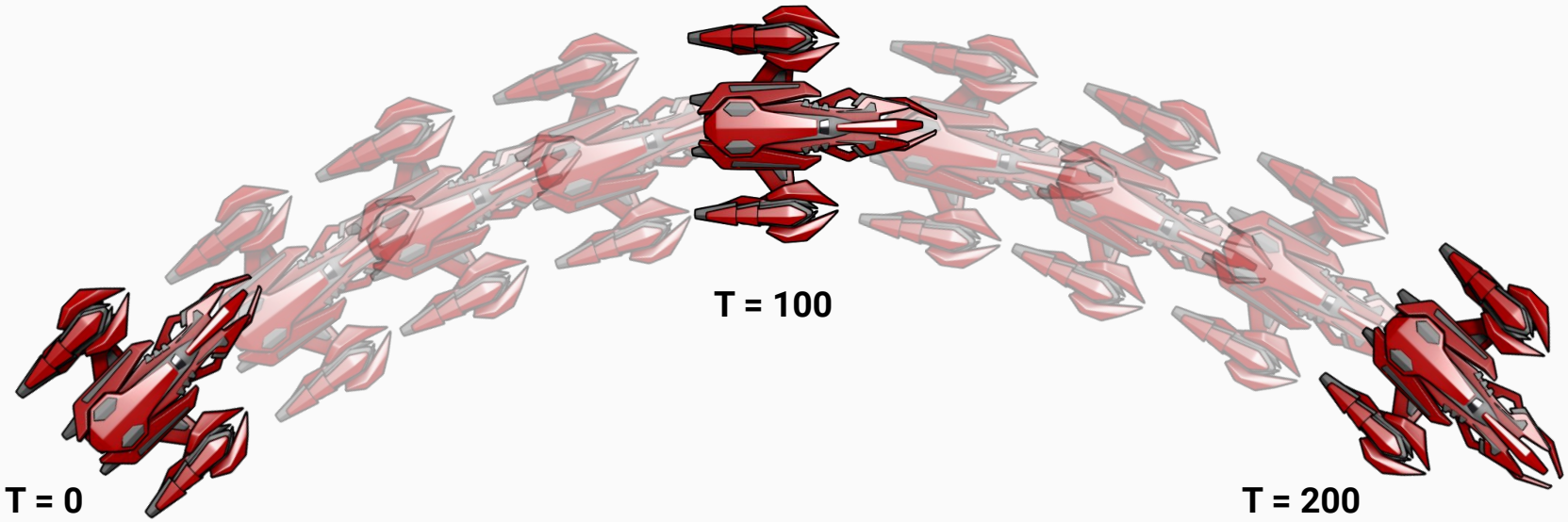
The problem


Server
Client
Replication data
Replication time interval
(100 - 200 ms)
Replication data
World updates
World updates
T = 100

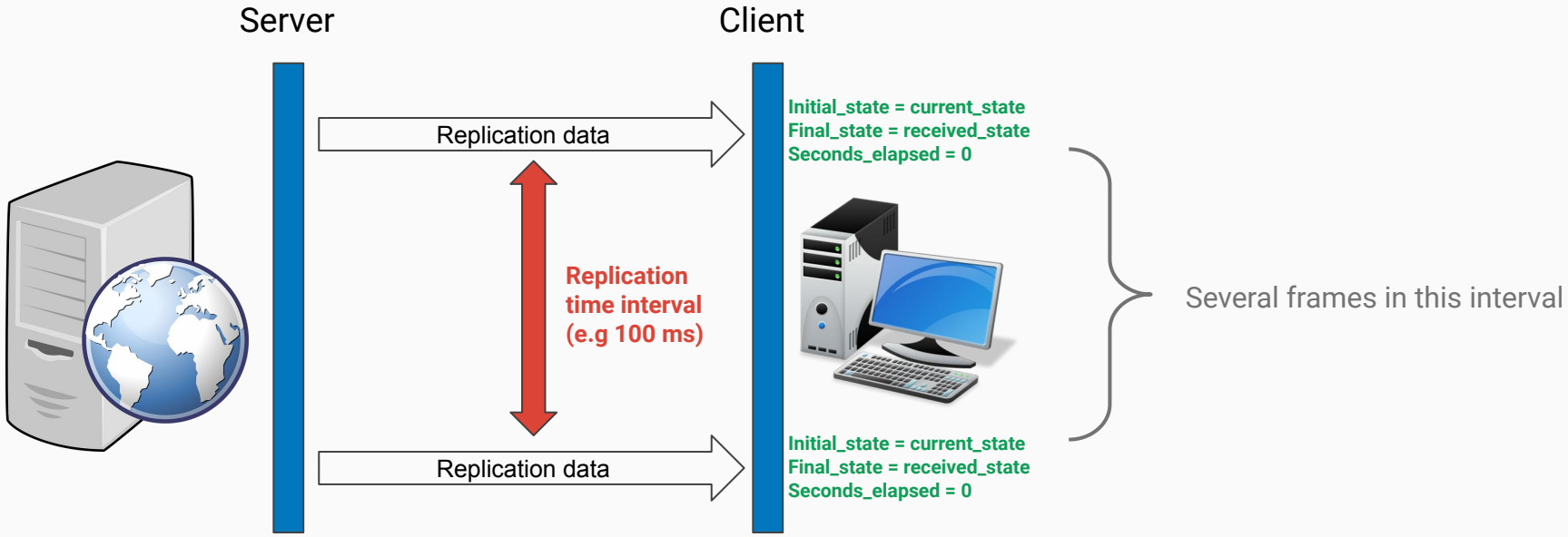
The problem


Server
Client
Replication data
Replication time interval
(100 - 200 ms)
Replication data
World updates
World updates
T = 200

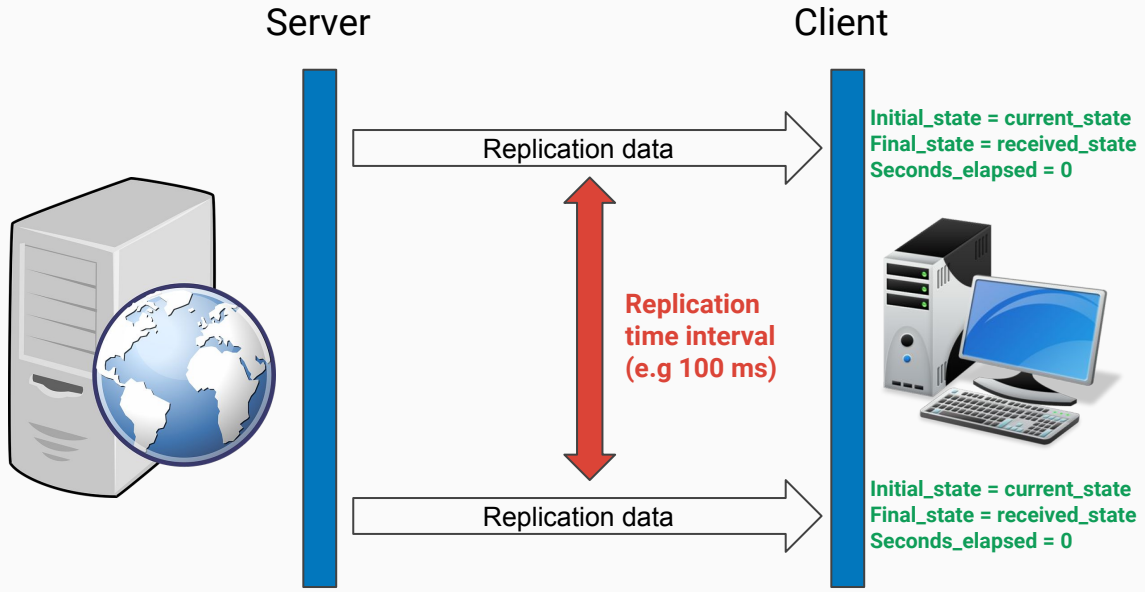
Entity interpolation



Entity interpolation



Entity interpolation



The current state is interpolated every frame between receptions of replication data.

Current_state = interpolate(initial_state, final_state, seconds_elapsed)

Current_state = interpolate(initial_state, final_state, seconds_elapsed)

Current_state = interpolate(initial_state, final_state, seconds_elapsed)

Current_state = interpolate(initial_state, final_state, seconds_elapsed)

Current_state = interpolate(initial_state, final_state, seconds_elapsed)

Entity interpolation

- Additional per GameObject information
  - Embedded into GameObject?
  - Separated interpolation component?
- When receiving replication updates
  - Assign current state to the initial values
  - Assign replicated state to the final values
  - Reset timer
- At each frame, for each GameObject
  - Interpolate between initial and final state
  - Update GameObject current state

```
// For entity interpolation

vec2 initial_position = vec2{ 0.0f, 0.0f };
float initial_angle = 0.0f;

vec2 final_position = vec2{ 0.0f, 0.0f };
float final_angle = 0.0f;

float secondsElapsed = 0.0f;
```

Known issues

Players see themselves in the present

Players see the world in the past

- **Replication delay**
  - As usual
- **Interpolation delay**
  - Not actually the last received state
  - Something in between the two last received states

Not actually so bad even sending packets every 100ms

Lag compensation

The problem

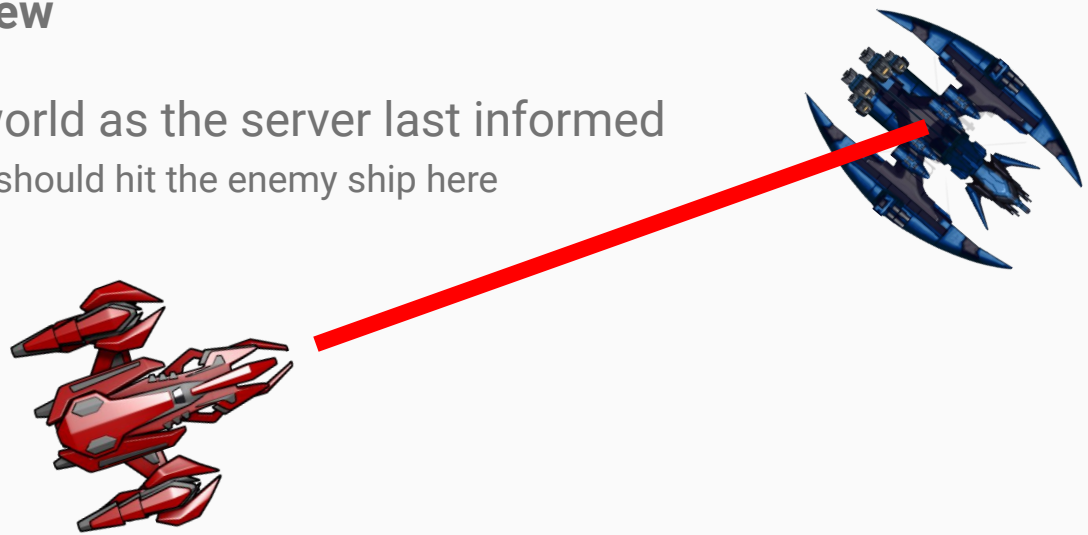
World state not the same for client and server

- Synchronization not instantaneous
  - Latency issues

The problem

Client point of view

- It sees the world as the server last informed
  - The laser should hit the enemy ship here



The problem

Server point of view

- It has the real state of the world
  - Newer than any client (and possibly different)
  - **Laser command received too late**

The problem

Server point of view

- It has the real state of the world
  - Newer than any client (and possibly different)
  - **Laser command received too late**
- Very noticeable in fast-paced games
  - Headshots not possible on characters moving fast

Lag compensation

In the server

- Record all network object states within a time window
  - At least 2 RTT (if sending packets at each frame)
  - Half a second should be enough in most cases
- Simulate world updates using old state
  - E.g: Laser collisions
    - Test against state visible by client

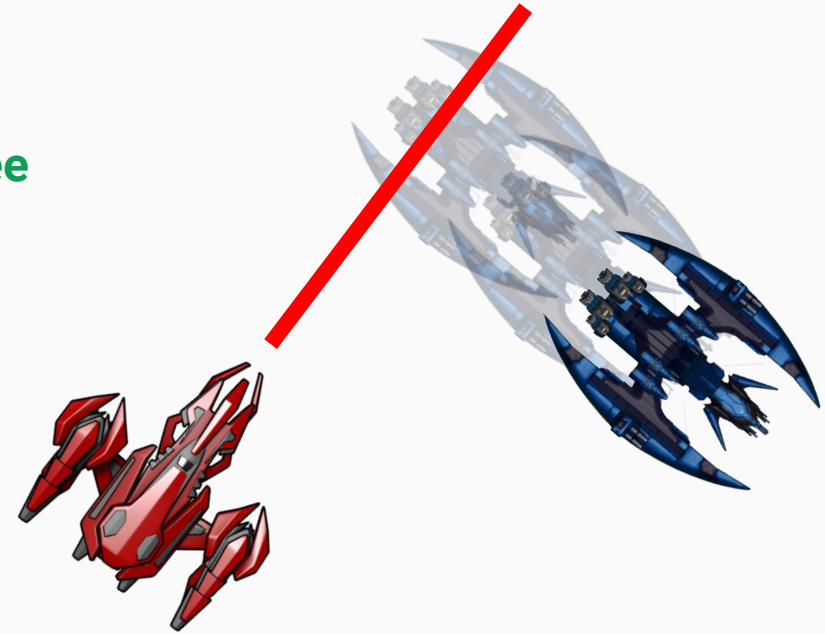
Lag compensation: known issues

Point of view of the shooter

- **Perfect! The shooters hit what they see**

Point of view of the victim

- **Shit, I was out of reach already!!!**



References

References

Visit Gabriel Gambetta's website for an explanation of the previous techniques:

Client-side prediction + server reconciliation

Entity interpolation

Lag compensation