# GAMES NETWORKING & SECURITY

## WEEK 4

# LATENCY MITIGATION & COMPENSATION

Jack Ingram & Stephen Selwood

# PART 1:

# LATENCY COMPENSATION TECHNIQUES: PREDICTION

# LEARNING OUTCOMES

To understand the need for latency compensation techniques in networked games.

To investigate how prediction can be used by clients and servers to reduce latency.

To be able to describe player and opponent predictive algorithms.

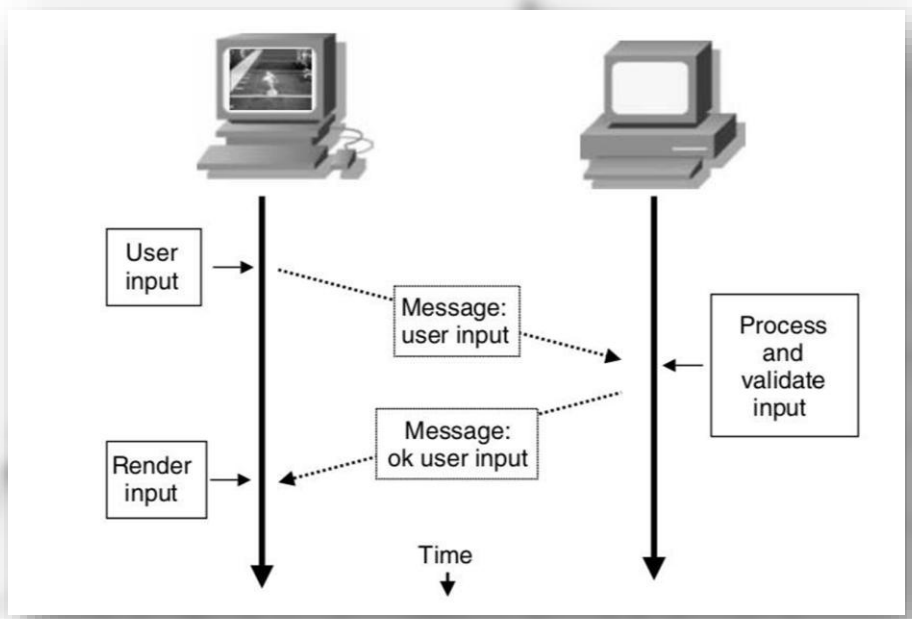
# Latency Compensation Techniques

## Client-Server Architecture | Game States

Most games today run on client-server architecture with a single, authoritative server that continually creates a valid game state based on all client updates.

- Clients collect inputs from their user and sends it to the server.
- Server computes the game state, sending revised game state back to all clients.
- Client renders new game state to the player.

This process repeats for the duration of the game.

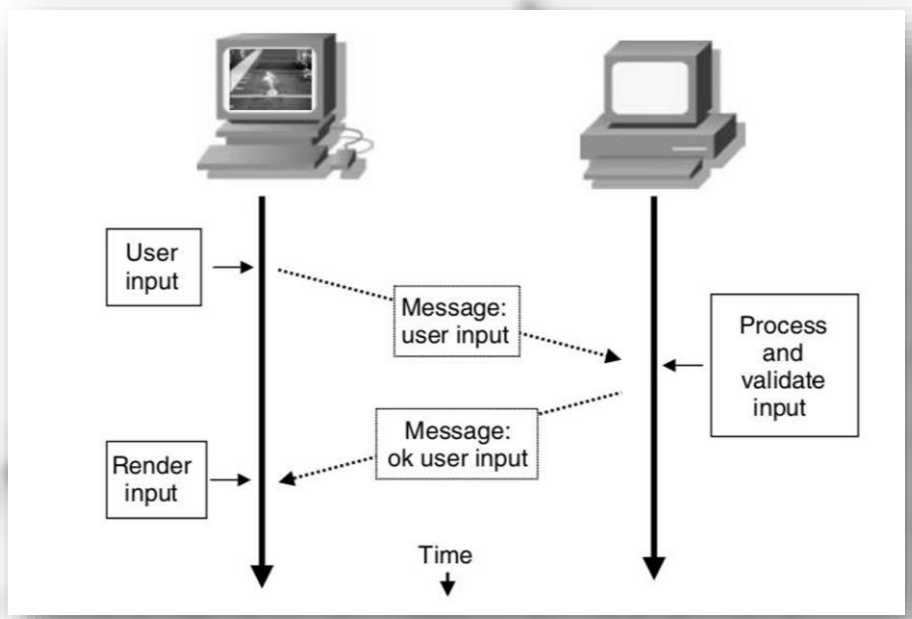


# Latency Compensation Techniques

## Client-Server Architecture | Game States

This is the basic networking algorithm from the client's perspective.

1. Sample the user input.
2. Pack up data and send to the server.
3. Receive updates from the server and unpack.
4. Determine visible objects and game states.
5. Render scene.
6. Repeat. 🔁



# Latency Compensation Techniques

## Client-Server Architecture | Game States

Let's take a look at an example of what can happen when there is latency in these server updates.

When playing a fast-paced shooter like *Call of Duty*, even a small delay in communication between the player and the server can cause noticeable gameplay issues.

Common Latency Symptoms:

1. Delayed Movement
   You press forward to sprint, but your character reacts a fraction of a second later. Feels like your controls are "heavy" or unresponsive.
2. Rubber-Banding
   Your character suddenly snaps back to a previous position. Happens when the server determines your actual location differs from what your game predicted. Especially common when sprinting around corners or dodging fire.


0
0
30
90

# Techniques | Player Prediction

# Latency Compensation Techniques

## Techniques | Player Prediction

Client-side prediction lets the player's game 'guess' what the server will say, instead of waiting for it.

The game reacts instantly to your input, then later checks with the server and tries to fix anything that was incorrectly predicted.

Key Benefits:

1. Immediate response to player input
2. Hides network latency
3. Makes online games feel like offline games

# Latency Compensation Techniques

## Techniques | Player Prediction

We predict two things:

1. Our player's actions
   What happens when we move or shoot.
2. Other players' actions
   Guessing where enemies/teammates are until the server confirms.

# Latency Compensation Techniques

## Techniques | Player Prediction

Why We Need Clarification:

Our client = the game on our computer

Other clients = players we see online

The server = the referee with the final say

The server sends verified updates to fix our predicted game state.

# Latency Compensation Techniques

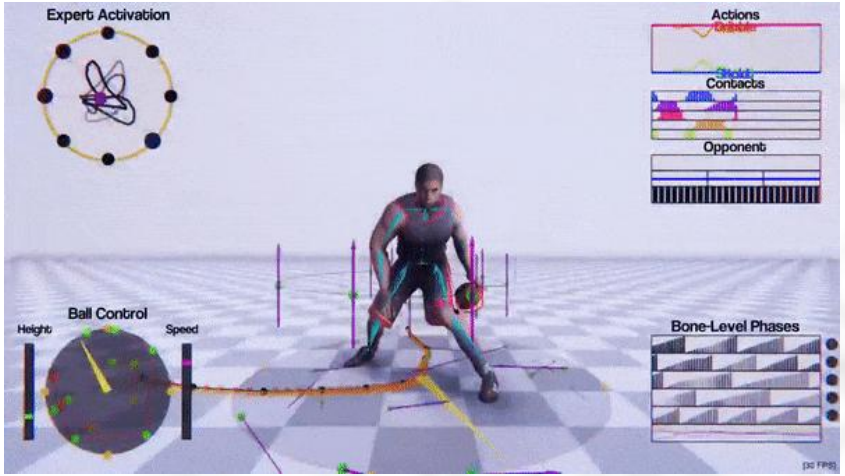
## Techniques | Player Prediction

How Player Prediction Works:

Our character moves immediately when we press a key.

Shortly after, the server confirms and we adjust if needed.

Feels instant → avoids lag




An example of Deep Learning trying to predict a players movement direction based on inputs and character physics.

# Latency Compensation Techniques

## Techniques | Player Prediction

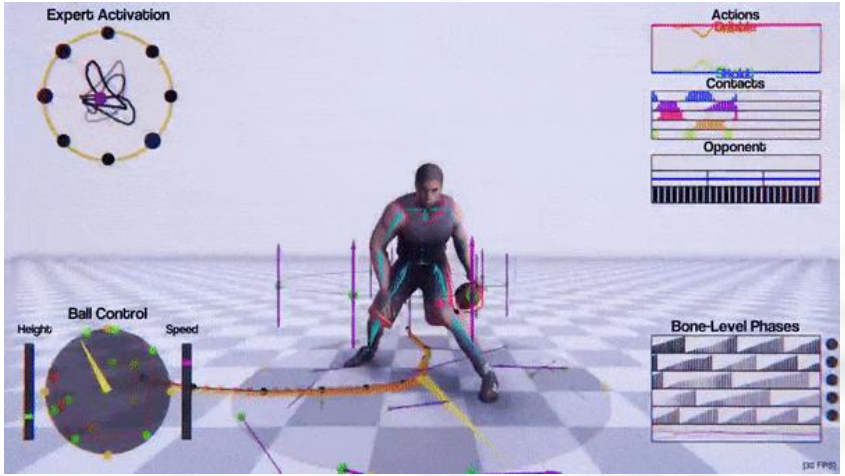
Prediction Errors:

Prediction isn't perfect.
Sometimes it guesses wrong.

Causes:

- Severe latency
- Sudden, unexpected player actions
- Differences in physics calculations

Solution: Correct gently, not suddenly.



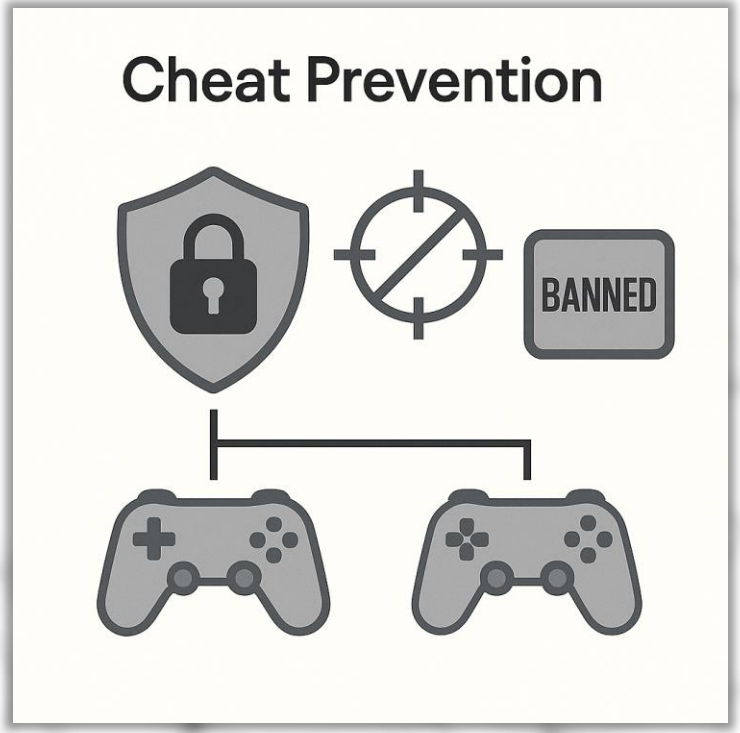

An example of Deep Learning trying to predict a players movement direction based on inputs and character physics.

# Latency Compensation Techniques

## Techniques | Player Prediction

Why it matters in security:

- Helps prevents client-faked moves (anti-cheat)
- Server authoritative state avoids hacking
- Client only suggests movement, server validates it



# Latency Compensation Techniques

## Techniques | Player Prediction

A reminder of the basic networking sequence of events from our client's perspective:

1. Sample the user input.
2. Pack up data and send to the server.
3. *Receive updates from the server and unpack.*
4. Determine player positions and game state.
5. Render scene.
6. Repeat.

This is the sequence of events from our client's perspective when using *prediction* algorithms.

1. Sample the user input.
2. Pack up data and send to the server.
3. Determine player positions and game state.
4. Render scene.
5. *Receive updates from the server and unpack.*
6. *Fix up any discrepancies.*
7. Repeat.

# Latency Compensation Techniques

## Techniques | Player Prediction

This should, in theory, improve the responsiveness of a networked game.

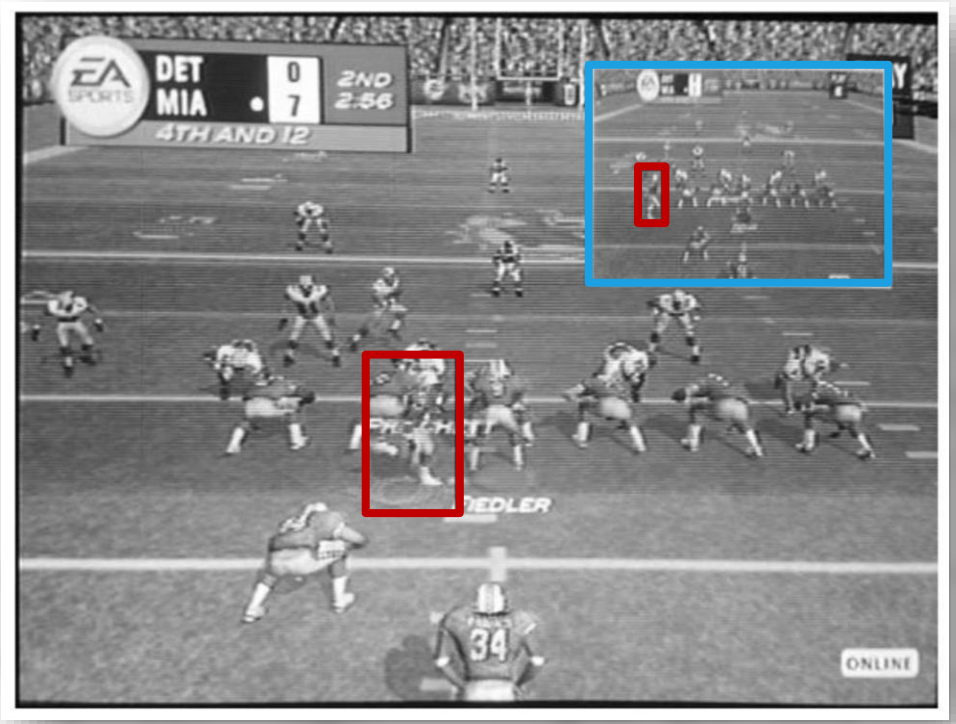
We can use a scenario from Madden NFL as an example to consider the effects of latency in this architecture.

# Latency Compensation Techniques

## Techniques | Player Prediction

In this case our client is the larger window, and the server is the smaller inset window outlined in blue.

The red box is drawn around the same player in both windows.

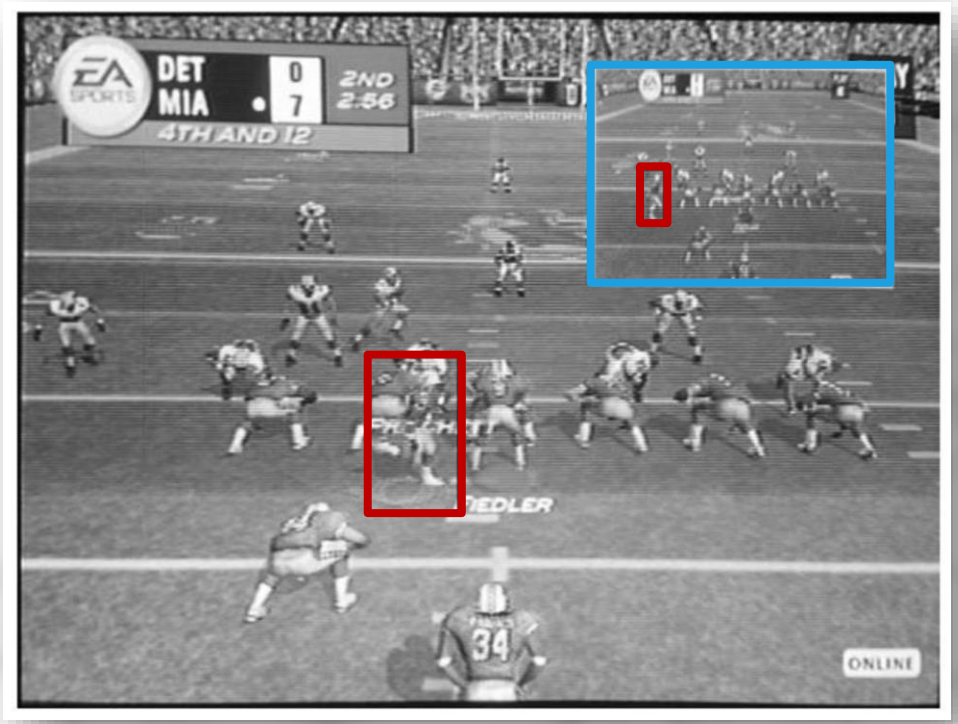


# Latency Compensation Techniques

## Techniques | Player Prediction

Our player has told their character to run right, which happens immediately on the client, but the message has not yet reached the server.

Therefore, there is a discrepancy between the position of this character on our client and the server.

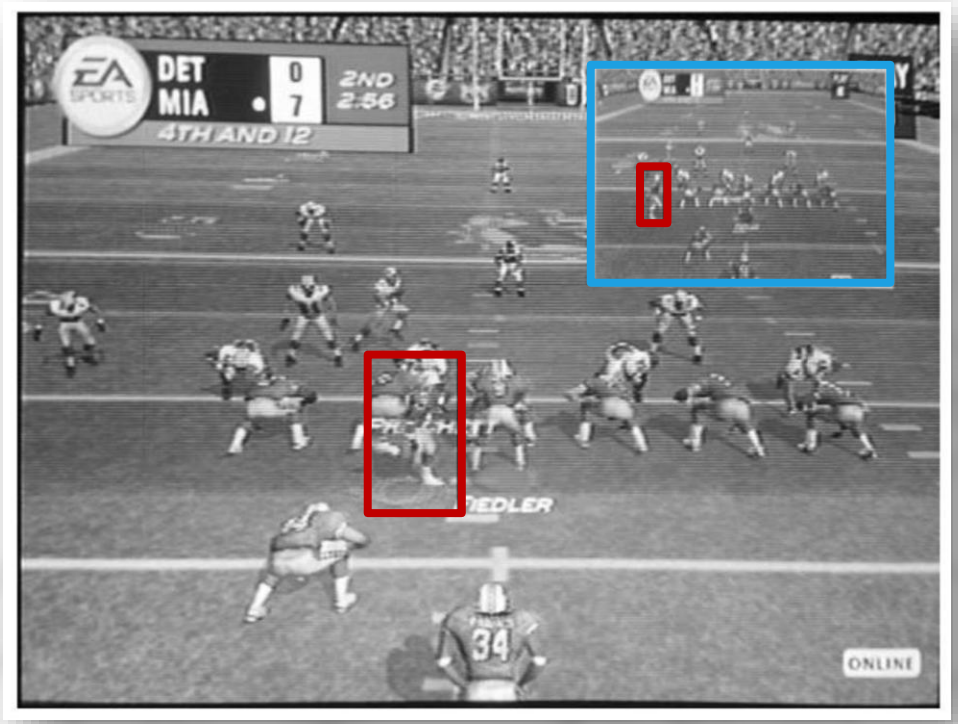


# Latency Compensation Techniques

## Techniques | Player Prediction

Some time later, when the server receives the input command, it checks if the character was allowed to move to the right.

If so, an updated game state with the validated move is sent to our client where it is confirmed and the game continues without issue.

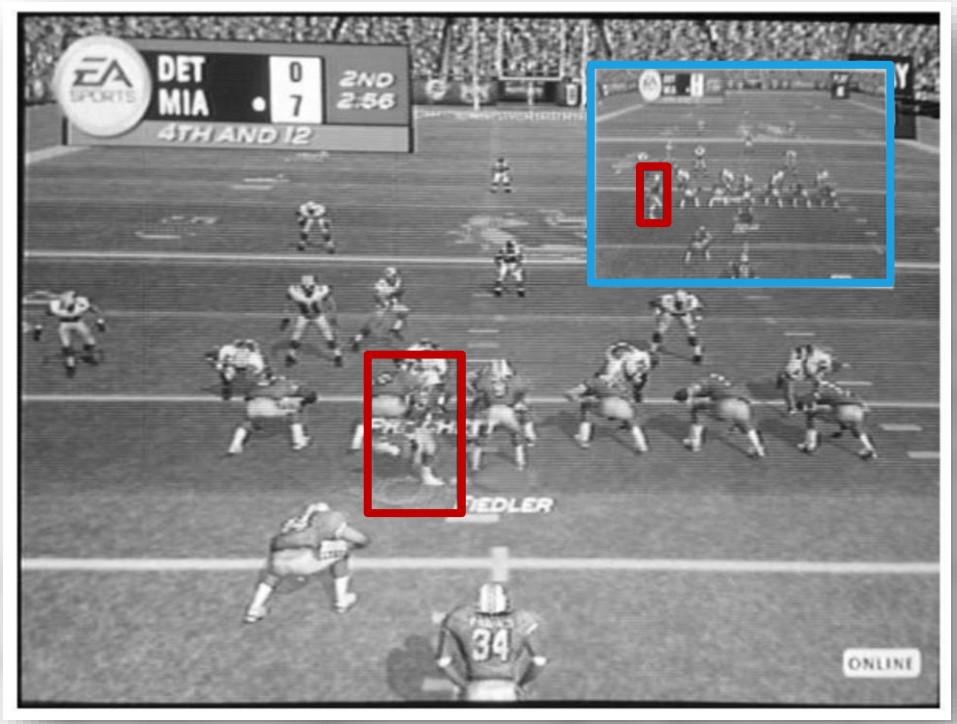


# Latency Compensation Techniques

## Techniques | Player Prediction

If, however, another player's character had moved to the same spot first, and blocks our character's path, the server would respond to our client that the movement was not valid. It would then superimpose the correct game state on our client.

Upon receiving the authoritative server update, our client would have to fix the discrepancy between the server's master state and our client's predicated state - ultimately rendering the correct game state to our client.



# Latency Compensation Techniques

## Techniques | Player Prediction

The benefit of client-side prediction to responsiveness is huge - an online game can feel as responsive as a non-networked game.

However, fixing up the discrepancies between the actual server-controlled game state and the client side predicted game state can sometimes be equally destructive.

# Latency Compensation Techniques

## Techniques | Player Prediction

This can involve having the display of an avatar abrupt changed by rendering the correct game state over the incorrect predicted scene.

This process is known as *'warping'* or *'rubber-banding'* as the world moves from the incorrect state to the correct one.

This is what we saw in the video earlier

# Latency Compensation Techniques

## Techniques | Player Prediction

Fundamentally, while client-side prediction allows the game to be more responsive - it trades off consistency between the game state at the client and server.


The more prediction is used, the less consistent the experience is for the players.

# Techniques | Opponent Prediction

# Latency Compensation Techniques

## Techniques | Opponent Prediction

With Opponent Prediction, the location of a unit that is controlled by another player is estimated by our client.

The estimation starts with the last known position of the other player's unit and computes its current predicted position based on the speed and direction it was travelling.

The predicted position is then used on our client until the server sends the updated location, speed and direction of the other player's unit.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

Our client will then fix the other player's unit's state (location, speed, health, etc) in our current game state when the other player's predicted movements fall outside a predetermined threshold.

The update sent by the server would have the correct position as well as the new location, speed and direction of the other player's unit, that our client can use to initiate a new prediction.

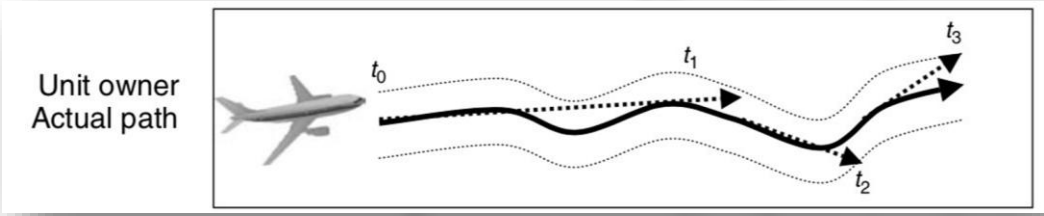
# Latency Compensation Techniques

## Techniques | Opponent Prediction

This example shows the difference between the actual path of another player's unit and the predicted path computed by our client.

The picture shows the direction and velocity of the other player's unit

The solid line in the middle shows the actual path of the unit on the other player's client.

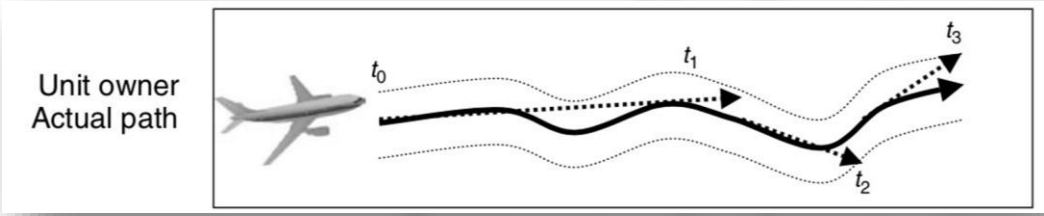


# Latency Compensation Techniques

## Techniques | Opponent Prediction

The two thinner, dotted lines that run parallel to the middle line represent the threshold for the opponent prediction.

The thicker dashed lines with arrows represent the direction and velocity (unit state) of the other players unit at the time the update was sent to the server ($t_0$, $t_1$, $t_2$, etc)

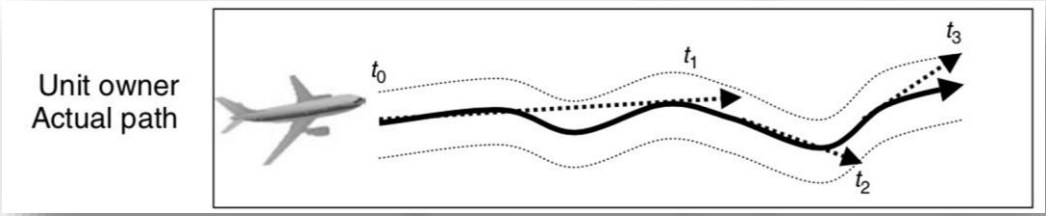


# Latency Compensation Techniques

## Techniques | Opponent Prediction

If the other player's unit predicted direction and velocity (unit state) goes outside the assigned update threshold, a message is sent to the server asking for an updated unit state (direction and velocity) for the other player.

After the initial position and direction is sent at $t_0$, updates are sent at $t_1$, $t_2$ and $t_3$ when the predicted direction and velocity cross the update threshold.

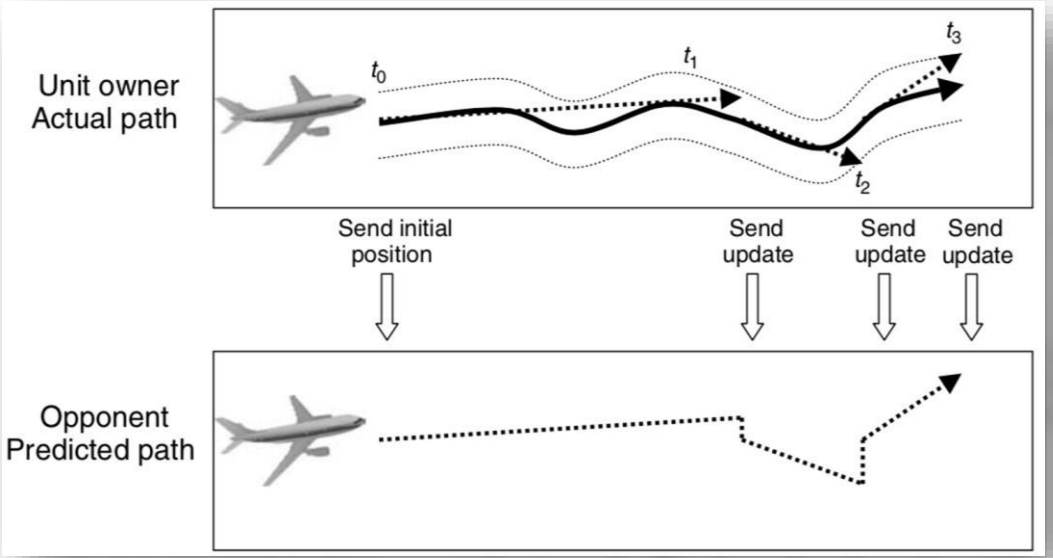


# Latency Compensation Techniques

## Techniques | Opponent Prediction

The bottom picture depicts the movement of the other player's unit from our players point-of-view based on updates from the server.

Our client will use the last updated direction and velocity of the other player's unit to predict the other players movement in our game state until a new update is received from the server.



# Latency Compensation Techniques

## Techniques | Opponent Prediction

Smaller values for the update threshold can provide more fidelity in opponent predictions at the cost of requiring more frequent updates.

This brings with it a higher bandwidth and processing overhead.

The optimal values for the threshold depend on the game, the network and processing capacities of the clients and, to some extent, player preferences.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

There are two commonly used unit estimation algorithms, both derived from basic physics:

- Predict the location of a unit on the basis of its last known position and its velocity at that time.
- Predict the location of a unit on the basis of the last known position, velocity and acceleration.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

More sophisticated algorithms can use roll, pitch, yaw or any other properties.

Complete predictions can be done on different parts of a unit independently.

This could involve the rotation of a tank turret being predicted independently of the direction and rotation of the tank itself.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

Unique prediction algorithms can even be created for specific games.

For example, a real-time strategy game may define what it means for a unit to ‘return to base’ and as long as the unit continues to return to base, all clients can accurately predict the position over time without any updates, as this is a predefined move.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

There are drawbacks to this technique.

It requires that each client runs an algorithm to extrapolate the location of each unit for each frame rendered.

If units behave unpredictably, such as is often the case in a frantic first-person shooter with loose physics constraints, the benefits of trying to predict unit locations diminish.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

The benefits do result in a reduction in latency, but it can also greatly reduce bitrates.

For large-scale simulations with relatively static units, or units that move in a predictable fashion, the use of opponent prediction can eliminate the interchange of many state update messages.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

For computer-controlled units that use random variables, as long as the clients have the same prediction algorithms and same initial random seed, their predictions can remain accurate.

For unreliable network transmissions, such as UDP, the use of opponent prediction can help smooth over lost update messages.

The trade-off is using more CPU cycles at each client to reduce the latency and, in some cases, the bitrate.

# Latency Compensation Techniques

## Techniques | Opponent Prediction

One approach to managing any inconsistent behaviour in predicted game state information when an update arrives is to perform a ‘rendezvous’ with the goal of achieving the new state in a short amount of time.

Instead of immediately applying the new state, custom sets of rules determine how the game world is to be updated.

# REVIEW

Latency can affect the performance of a networked game.

Latency is inherent in the system but can be managed through compensation techniques.

Client and server prediction can improve the experience by allowing the game state to continue without an authoritative command from the server.

Once the server has captured the complete game state from all clients an authoritative game state can be sent back to the clients.

Each client will then apply the the authoritative game state.

# QUESTIONS?

# BREAK

Take 10 minutes to stretch your legs.

# PART 2:

# LATENCY COMPENSATION TECHNIQUES | TIME MANIPULATION & OTHER TECHNIQUES

# LEARNING OUTCOMES

To understand the need for latency compensation techniques in network games.

To investigate how time manipulation and other techniques can be used by clients and servers to reduce latency.

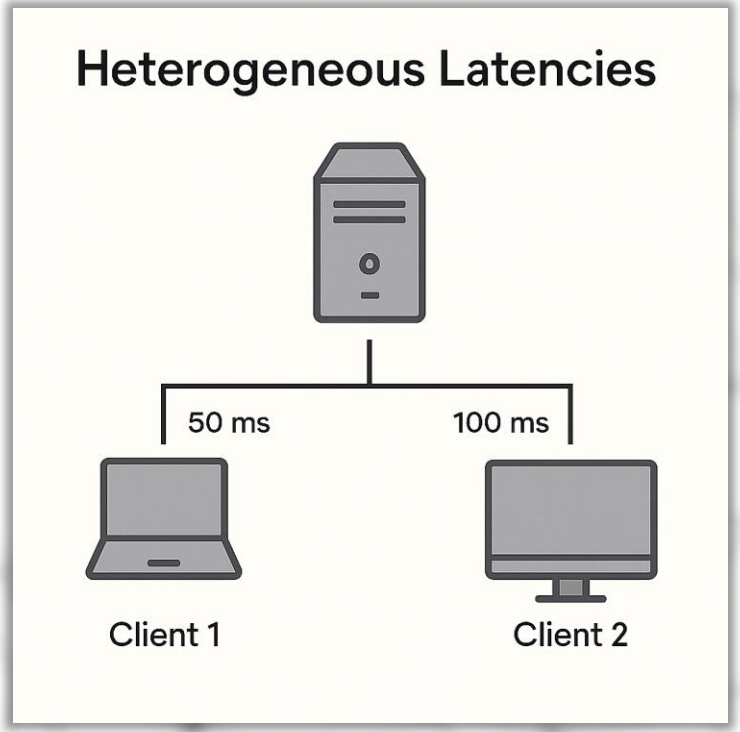
# Latency Compensation Techniques

## Heterogeneous Latency | Game States

There is potential for unfairness with opponent prediction in the case where players have heterogeneous latencies.

If an update on the actual position is sent to several players, the players with higher network latency will get the update later.

This results in a larger difference between the predicted position and the actual position, and therefore a less accurate game world than closer players.

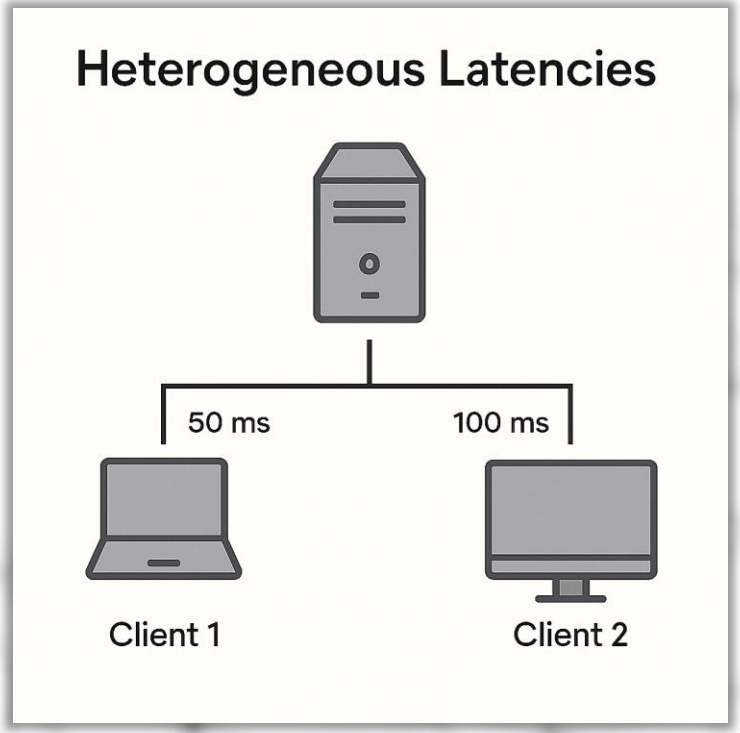


# Latency Compensation Techniques

## Heterogeneous Latency | Game States

This imbalance can be removed completely by the use of time delay, whereby the update to closer players are delayed by exactly enough time to make their effective latency the same as the farthest player.

This results in another trade-off. Minimising the differences in the accuracy of opponent prediction among players, while maximising the amount of prediction error all players receive.

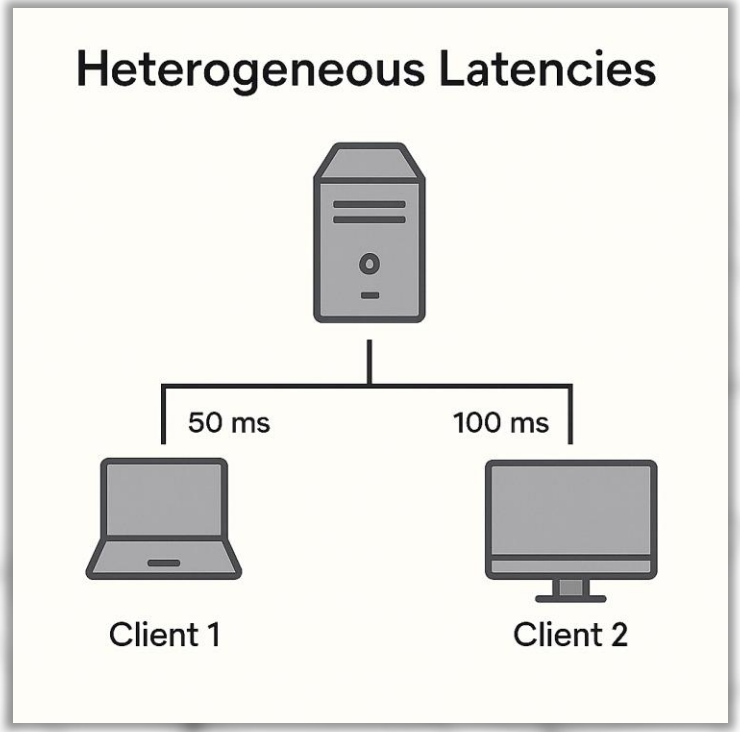


# Latency Compensation Techniques

## Heterogeneous Latency | Game States

Another way to reduce unfairness is to send more frequent updates to the players that are geographically further away from the game server, thus reducing the error in their predicted versus actual prediction.

If there is a budget on the total number of updates, the update rate to the closer players can be reduced to make the update rate to the further players higher.



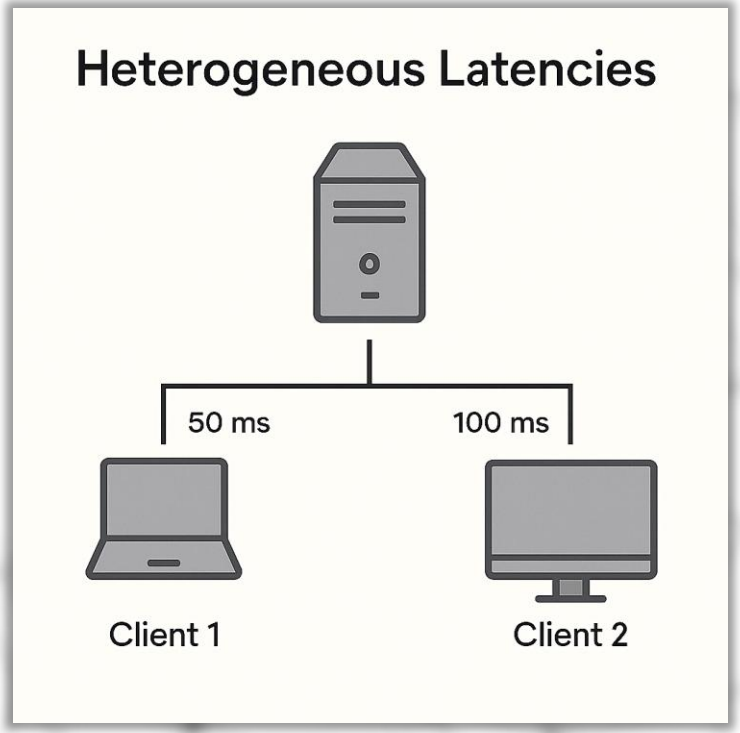
# Latency Compensation Techniques

## Heterogeneous Latency | Game States

Even without prediction, game states rendered at clients will differ from each other.

This is because it takes some time for a client to receive the world state from the server.

When one client is further away (has a higher round trip time) from the server than another client, there may be unfairness in game play.



# Latency Compensation Techniques

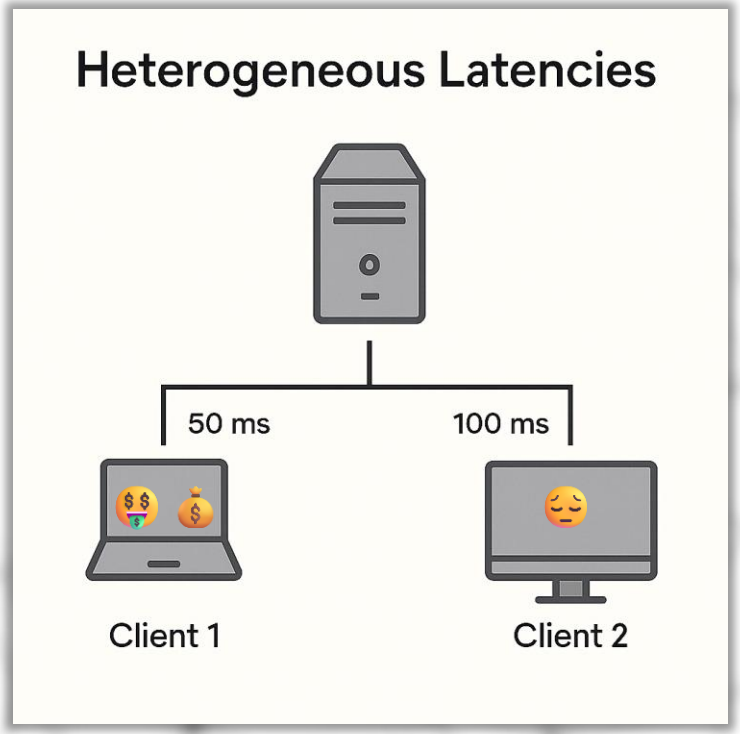
## Time Manipulation | Game States

Example: two players finally defeat a monster they had been battling.

The server generates some loot as a reward for the battle and drops it on the ground. A message with the location of the treasure is sent to the clients of each player.

Suppose the first client is geographically closer to the server. They would see the treasure immediately and pick it up.

The loot may have already been collected by the time the second player receives the message.



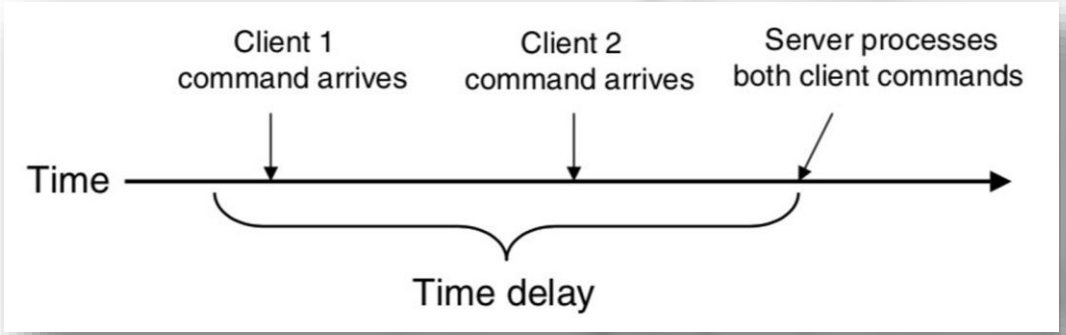
# Latency Compensation Techniques

## Time Manipulation | Time Delay

As mentioned a few slides ago, another common technique for dealing with differences among clients is to delay processing and sending of user commands to equalise latency.

Instead of processing client commands right away, the server delays them for some time, allowing a client that is further away to respond to the game state.

In essence, this allows both clients to have the same effective latency in providing updates to the game world.

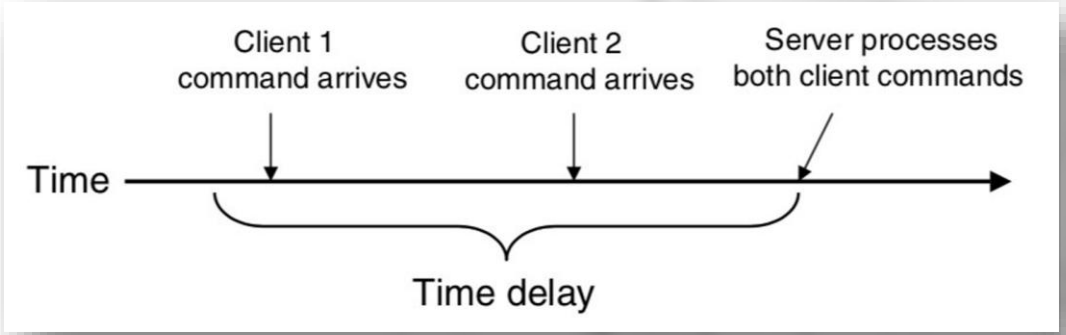


# Latency Compensation Techniques

## Time Manipulation | Time Delay

Alternatively, the clients themselves can use buffering to equalise the latency, with the closer client delaying processing of the server state updates.

Buffering can provide fairness between clients with disparate latencies, but at the cost of making the gameplay less responsive.

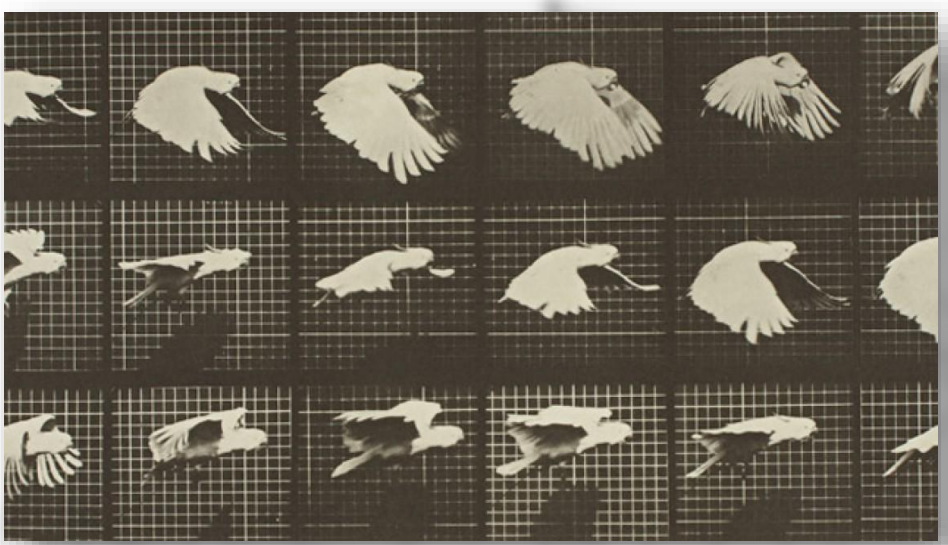


# Latency Compensation Techniques

## Time Manipulation | Time Delay

In choosing a time delay buffer, it should be noted that users often prefer a consistent, even large, delay rather than a variable delay that jumps about wildly.

Therefore, the size of any time delay chosen should be adjusted infrequently, even if the latency measurements from the server change more frequently.

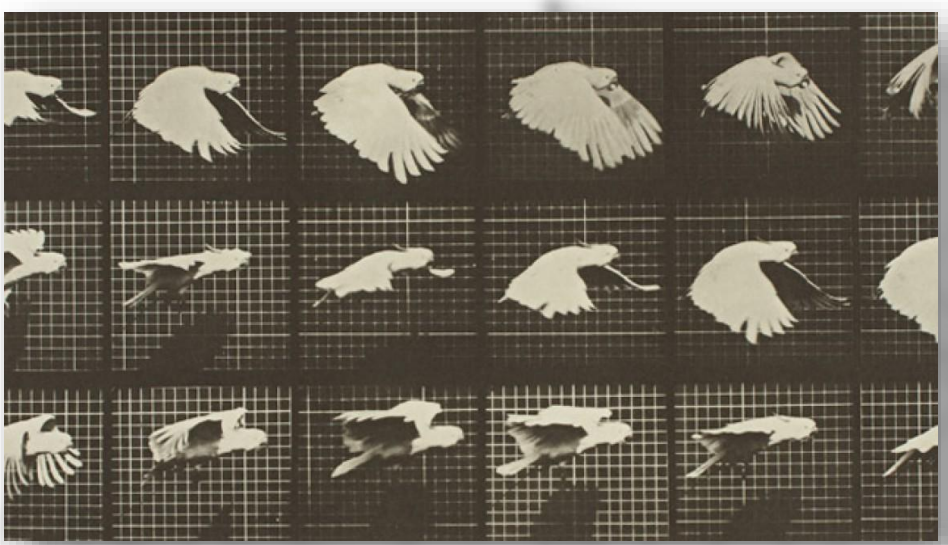


# Latency Compensation Techniques

## Time Manipulation | Time Delay

If games states on a server and local client differ, and the client needs to be updated to the ‘true’ game state, it can use interpolation to ‘blend’ the current client game state to that of the server over time.

This reduces the impact on the players experience as the client transitions to the new validated game state from the server smoothly and without noticeable jumps.



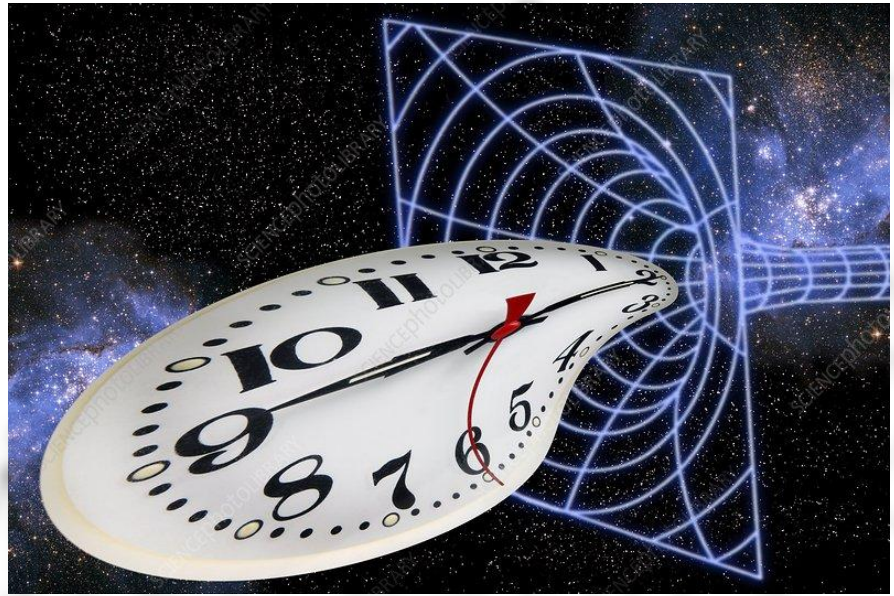
# Latency Compensation Techniques

## Time Manipulation | Time Warp

A widely-used time manipulation mechanism is to have the server rollback (or time warp) the events on a game client to the last validated update to the server.

As we know, the client sends updates based on the current game state.

Because of the lag between the client sending the update and the server receiving the update, the authorised state of the game on the server may have changed.



# Latency Compensation Techniques

## Time Manipulation | Time Warp

Let's look at an example:

In an FPS match, *Player1* shoots and hits *Player2.*

*Player2's* position at this time is based on the last server update.

By the time the client update from *Player1* is received by the server, *Player2* has moved.

# Latency Compensation Techniques

## Time Manipulation | Time Warp

To resolve this issue the server can *roll back* the events it had processed since both players provided the update.

Based on the server's gameplay protocols the server may determine that *Player2's* move was overridden and award the kill to *Player1*.

Likewise, the server may determine that *Player2's* move *was* valid and *not* award the kill to *Player1*.

# Latency Compensation Techniques

## Time Manipulation | Time Warp

Which one is best comes down to what the server, or rather you the developer, consider to take priority and designate a ‘successful’ action:

Do we consider an action successful and valid the moment a client takes a user input?

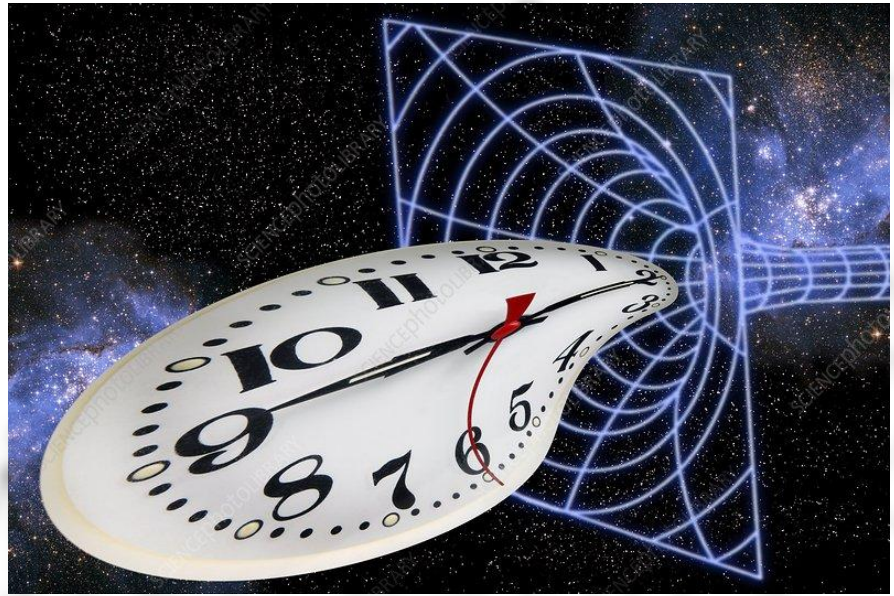
Or do we consider an action successful and valid the moment a client updates the server as to the action that was undertaken?

# Latency Compensation Techniques

## Time Manipulation | Time Warp

The time warp algorithm for the server is as follows:

1. Receive update from client.
2. Extract user input information.
3. Elapsed time = current time – latency to client.
4. Rollback all events in reverse order to current time – elapsed time.
5. Execute user input.
6. Repeat all events in order, updating any clients affected.
7. Repeat. 🔁

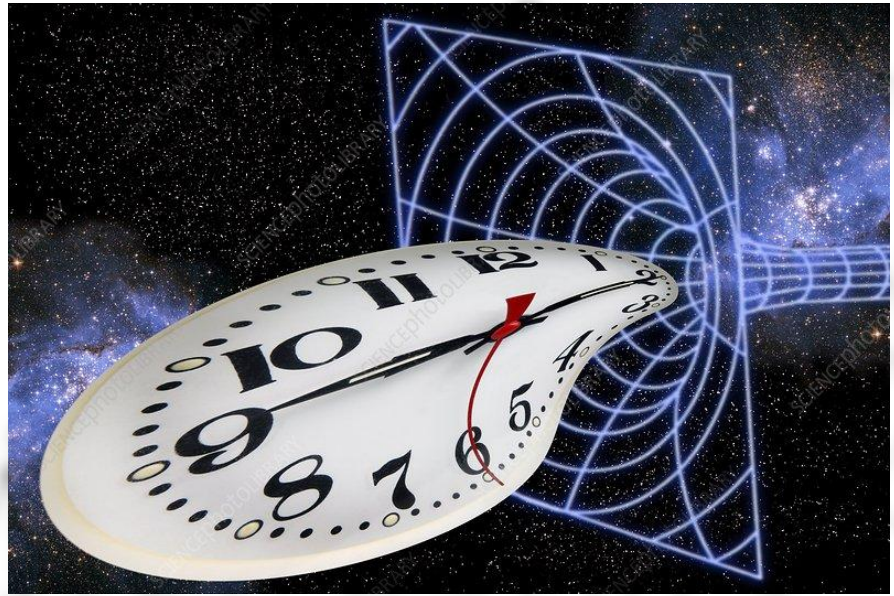


# Latency Compensation Techniques

## Time Manipulation | Time Warp

For time warp to be effective, it requires an accurate measurement of the latency in order that the game time  can be rolled back the proper amount.

Fortunately, the frequent message exchanges between server and client provide many opportunities for client-server latencies to be refined as the game is played.



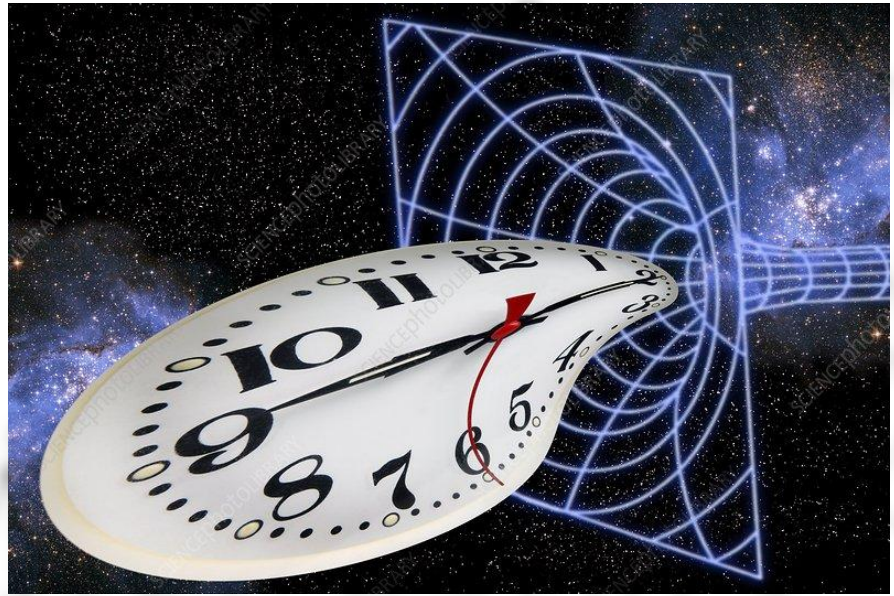
# Latency Compensation Techniques

## Time Manipulation | Time Warp

By using time warp, clients that have a high latency can still have their commands executed in the correct game-time order without impacting other players.

A player in a first-person shooter can aim directly at an opponent, not having to worry about the opponent moving before the server gets the shot update message.

Without time warp, a player would have to aim in front of the target in order to hit the opponent when they did move.



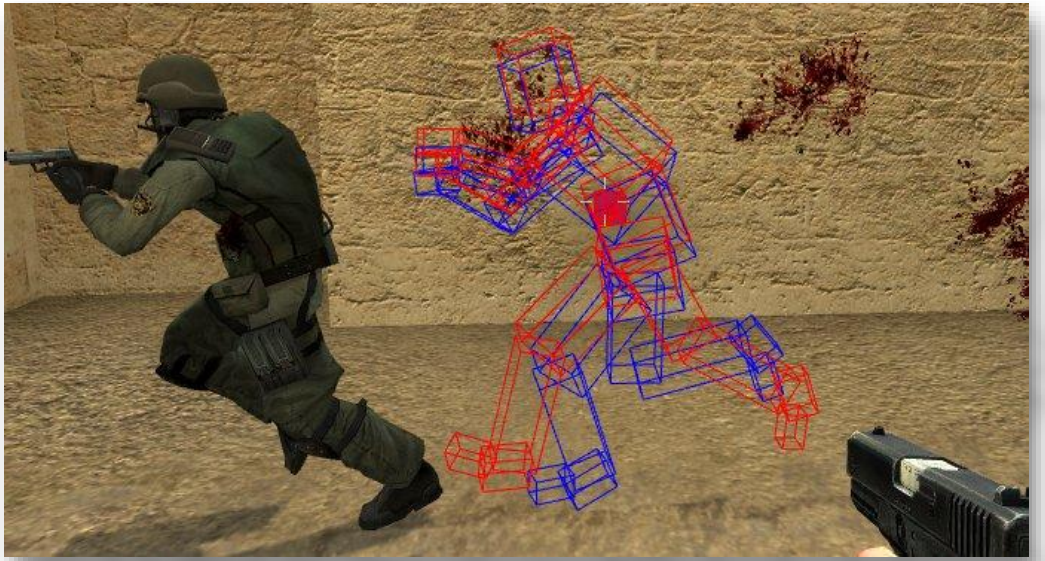
# Latency Compensation Techniques

## Time Manipulation | Time Warp Example

Counter-Strike & the Source engine makes use of time warp. This image (right) shows time-warp in action.

In this example the round-trip latency is 200ms, meaning the client player's commands are executed 100ms before the screenshot.

The red boxes show the target position on the client where it was 100ms ago.

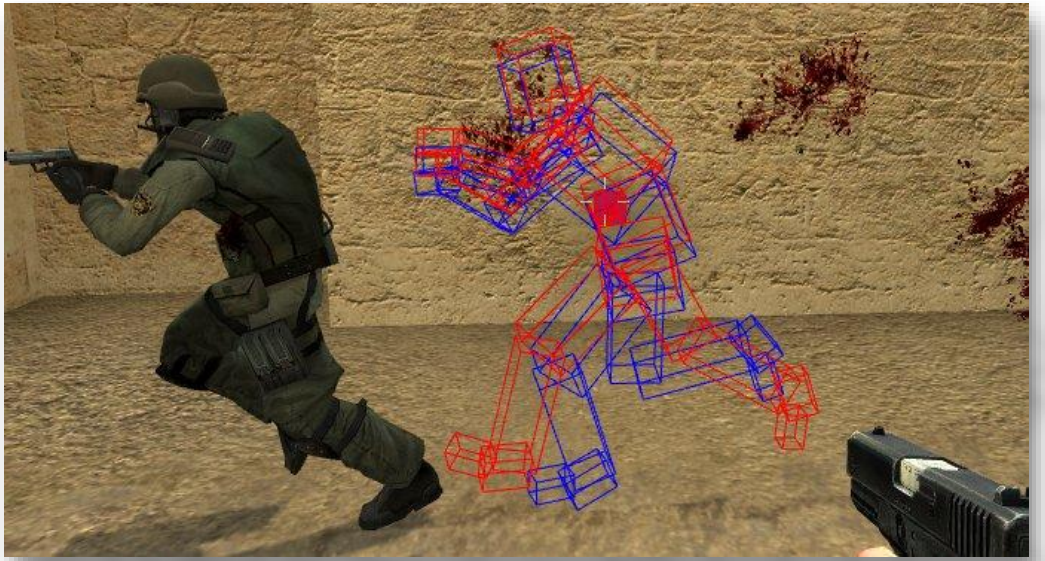


# Latency Compensation Techniques

## Time Manipulation | Time Warp Example

Since then, the target moved to the left while the client player's command was travelling over the network to the server.

When the client command arrives at the server, the server rolls back to put the target in the position it was in at the time of the shot, indicated by blue boxes.

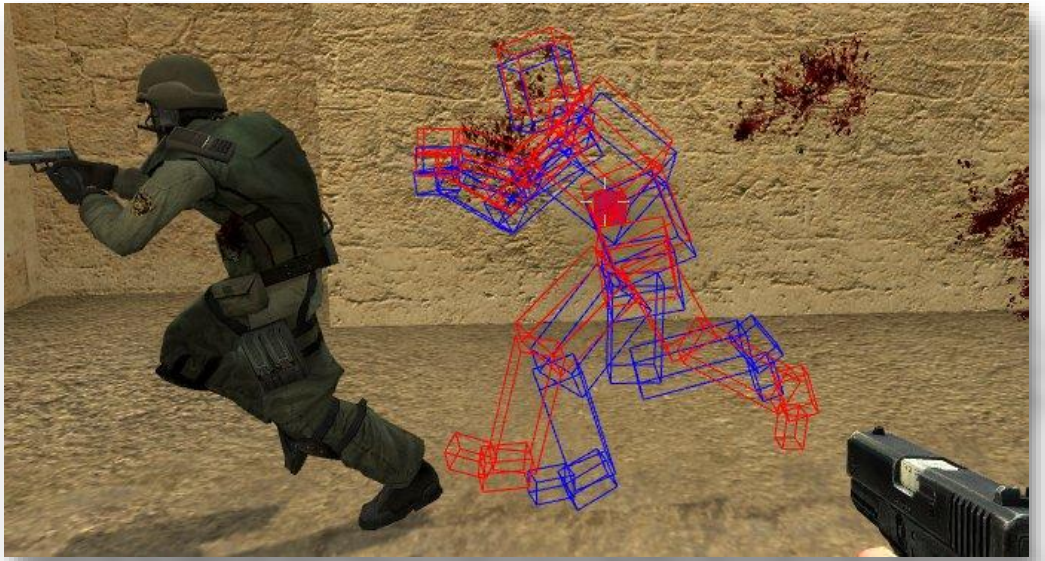


# Latency Compensation Techniques

## Time Manipulation | Time Warp Example

The server determines there was a hit.

Note that the red and blue boxes do not match precisely because of small differences in the time measurements.



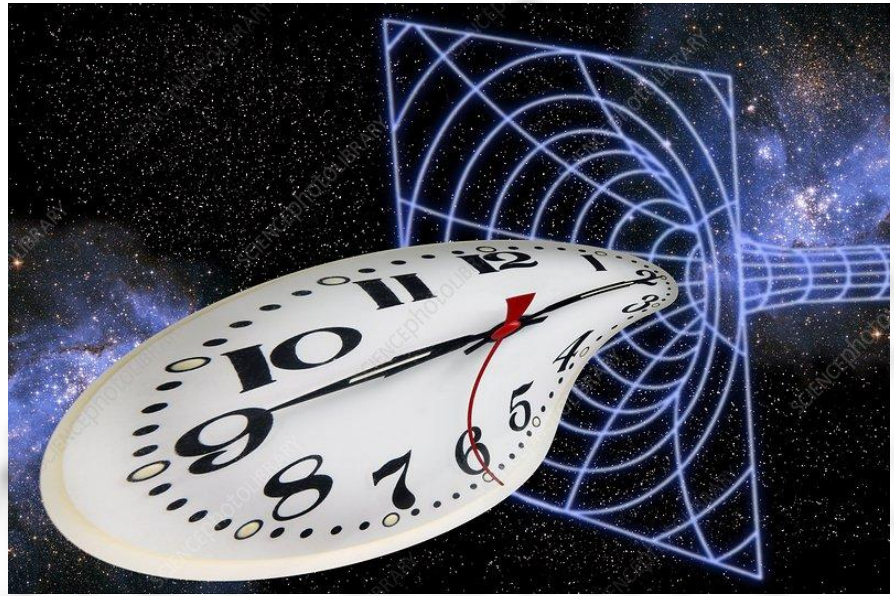
# Latency Compensation Techniques

## Time Manipulation | Time Warp

Time warp can also cause inconsistencies between different player's views.

Suppose a player places the crosshairs of a gun on an opponent and fires. The server, using time warp, will ultimately determine this is an 'on-target' shot and award the point.

In the meantime, because of client-server latency, the opponent may have moved, perhaps even around a corner and out of sight.

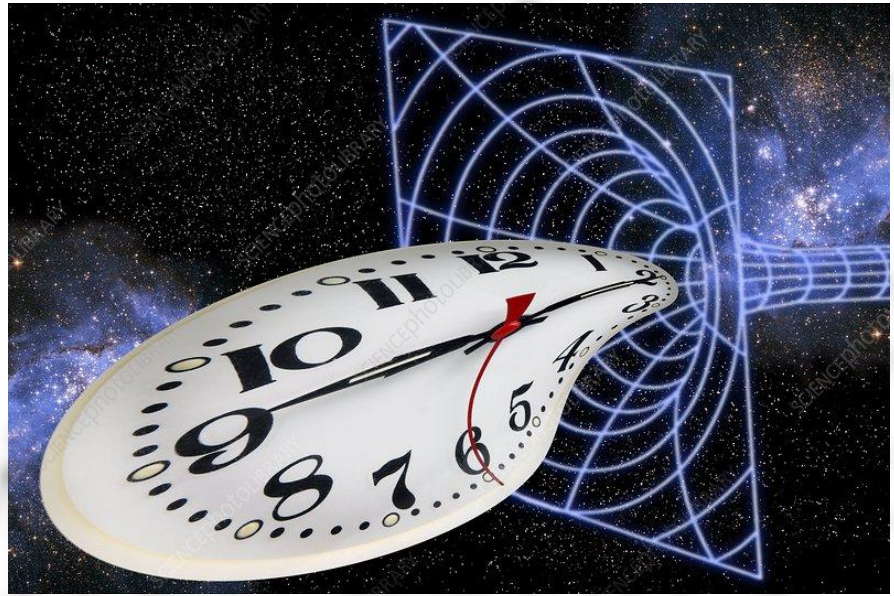


# Latency Compensation Techniques

## Time Manipulation | Time Warp

When the server warps time back to when the shot was fired and determines the opponent was shot, it will seem to the opponent that the bullets actually followed the opponent around the corner.

Fortunately, this disconcerting effect is minimised if the opponent cannot see the attacker or if the opponent was still in the open and not hiding.

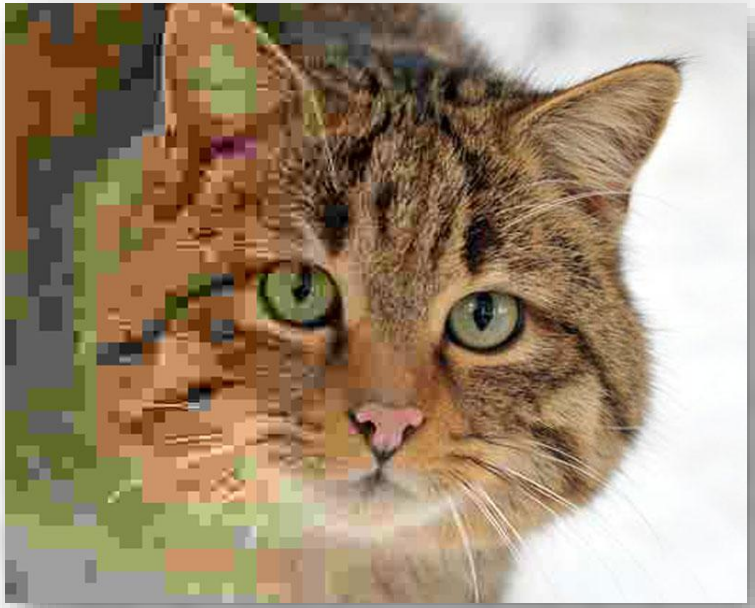


# Latency Compensation Techniques

## Packet Management | Compression

Reducing the size of the data packages sent between client and server can also reduce latency.

A small packet has a shorter transmission time than a large packet because of serialisation delay (we looked at this in previous lectures).

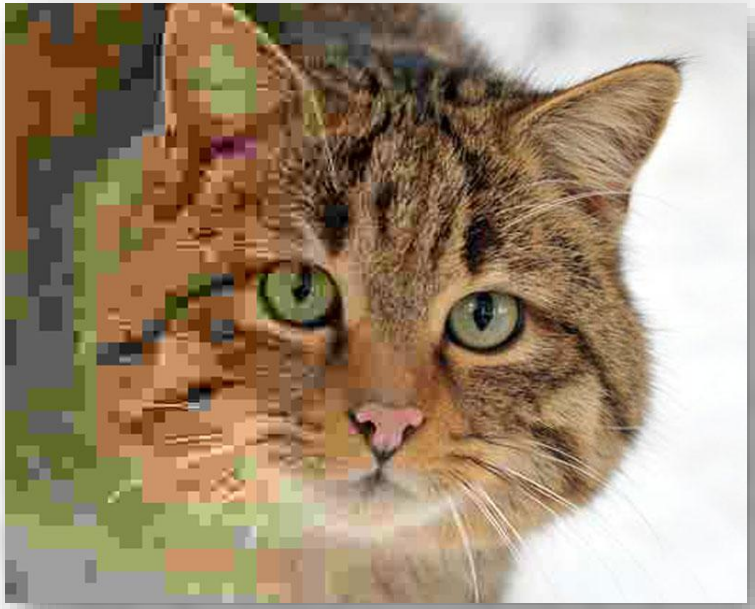


# Latency Compensation Techniques

## Packet Management | Compression

There are several ways that packet sizes can be reduced:

- Lossless compression
- Delta compression
- Interest management
- Peer-to-peer
- Update Aggregation



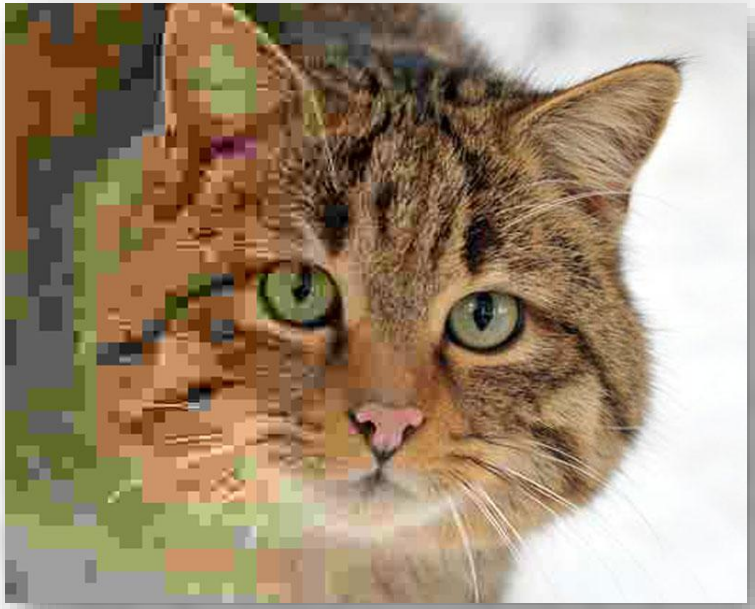
# Latency Compensation Techniques

## Packet Management | Lossless Compression

Data can be manually compressed using well-known algorithms. Most are based on the *LZW algorithm.*

Unlike compression for audio and video, compression for game data must be lossless, meaning all the bits that are compressed must be restored when uncompressed.

Lossless data compression finds repeated patterns in the bits and compresses the repeats to use fewer bits.



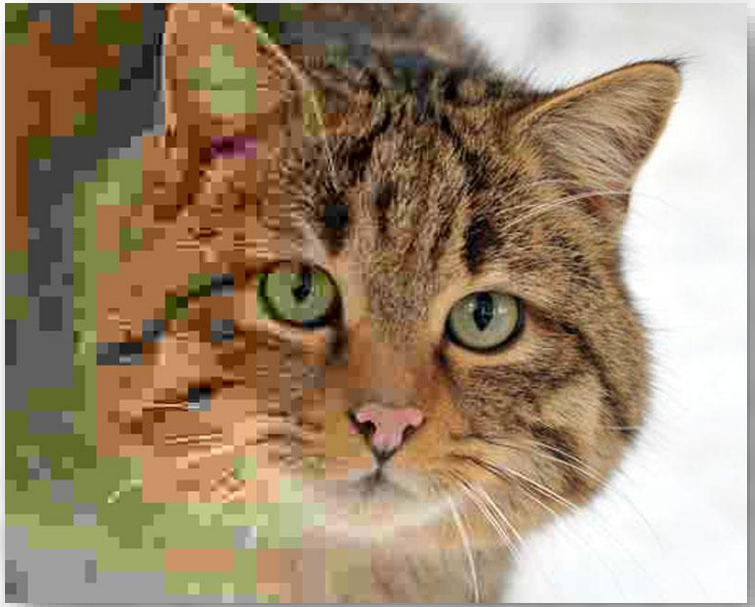
# Latency Compensation Techniques

## Packet Management | Interest Management

Instead of sending all data to each client, only a subset of data that is of interest to the client can be sent instead.

The area of interest for a client is called the Aura and is where the interaction between the client and the other game units occurs.

Auras need not be symmetric, where the focus of one object needs to intersect the nimbus of another object.

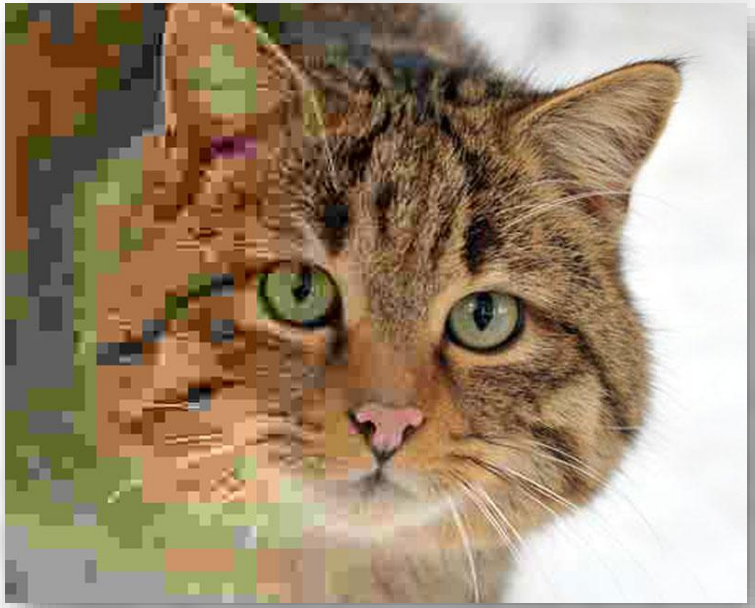


# Latency Compensation Techniques

## Packet Management | Peer-2-Peer

Using a peer-to-peer network where clients send data directly to each other rather than to the server can reduce bitrates to the server.

P2P architectures are used for some common game aspects, such as voice chat, but can be extended to include other non-essential game aspects.



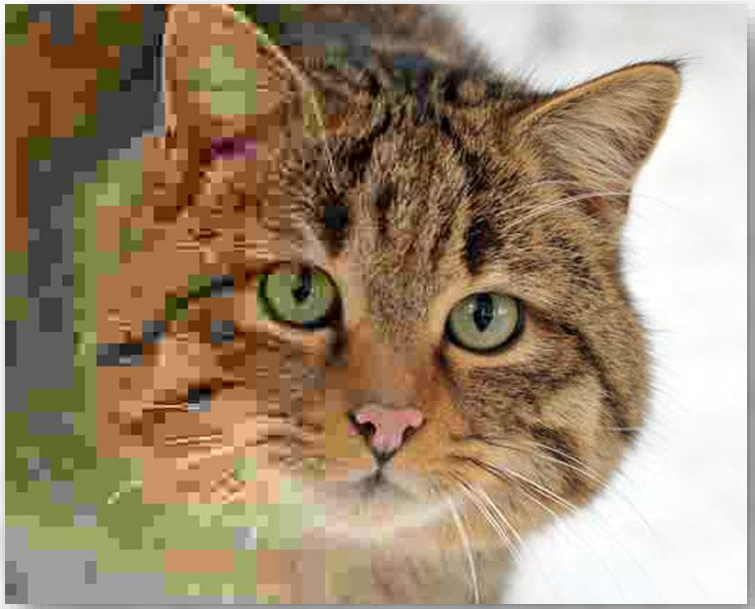
# Latency Compensation Techniques

## Packet Management | Update Aggregation

Sending updates after some periodic delay can avoid some of the network overhead associated with each message.

If *Player1* moves at time $^{t}0$ and *Player2* moves at time $^{t}1$; rather than send two messages to *Player3*, the server may choose to send one, larger message at time $^{t}1$, containing the moves for both players.

This avoids the network overhead for packing and sending an additional message.



# Latency Compensation Techniques

## Game Management | Visual Tricks

A start-up animation can be used to hide latency from the client to the server.

For example, if a boat gets ready to move, the game may require it to visually raise sails before it starts to actually move.

Such animation delays can allow a message to go to the server and back before the unit actually moves. This way there is no discrepancy to fix.

# Latency Compensation Techniques

## Game Management | Visual Tricks

Similarly, local confirmation can be used immediately even if the remote effect is not confirmed by the server.

If a player pulls a trigger on a gun, the game client can play a shooting sound effect and show a puff of smoke, even if the impact of the shot is not determined for some time.

This makes the player feel immersed, and hides the calculations going on under the surface with clever trickery...

# REVIEW

Latency can affect the performance of a networked game.

Latency is inherent in the system but can be managed through compensation techniques.

Along with predictive techniques, time management, compression, data packet management and in-game visual cues can improved latency, or perceived latency from the client point of view.

# QUESTIONS?

# Task | Questions to you

Group Task: Get into groups of 4 or 5 and discuss these questions.
Make note of your answers on paper or your laptop/phone.

1. Describe in your own words, how **Player Prediction** algorithms usually work.
2. Describe in your own words, how **Opponent Prediction** algorithms usually work.
3. What is/are the drawback(s) of the **Player Prediction** model?
4. What is/are the drawback(s) of the **Opponent Prediction** model?
5. *GodMode question:* Can you think of a different prediction or state resolution method that could potentially yield better performance on the client nodes? You can describe your answer with words or pseudocode.
6. How does compression help to minimise latency?
7. Give two ways that packet sizes can be reduced.
8. For a game like Fortnite, justify what you think is the **best method** of latency compensation.
9. What is a drawback of buffering server updates?
10. In the time delay buffer method, what is better: Consistent delays or frequently adjusted delays?

# LECTURE FINISHED