# Managing Dynamic Shared State

448430
Spring 2009
3/30/2009
Kyoung Shin Park
Multimedia Engineering
Dankook University

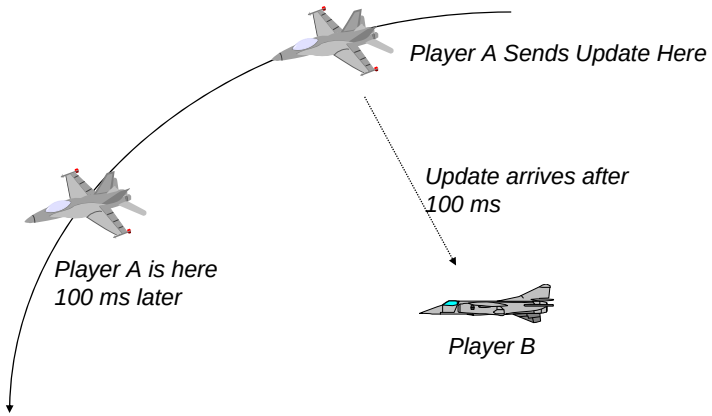
## Overview

- Dynamic Shared State
- Consistency-Throughput Tradeoff
- Centralized/Shared Repository
- Frequent State Regeneration (Blind Broadcast)
- Dead Reckoning

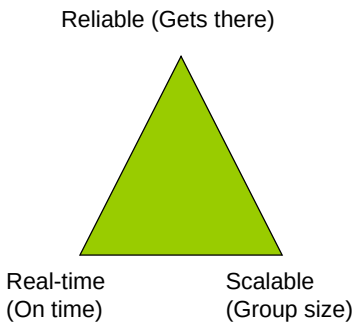
## What is Dynamic Shared State?

- The dynamic information that multiple hosts must maintain about the NVE
- Accurate dynamic shared state is fundamental to creating realistic virtual environments. It is what makes a NVE "multi-user".
- Management is one of the most difficult challenges facing the net-VE designer. The trade off is between resources and realism.

## Network Latency Problem



## Consistency-Throughput Tradeoff




- "It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state."
- We can have either a dynamic world or a consistent world, but not both.

## Another Example

- Player A moves and generates a location update.
- To ensure consistency, player A must await acknowledgments.
- If network lag is 100 ms, acknowledgment comes no earlier than 200 ms.
- Therefore, player A can only update his state 5 times per second.

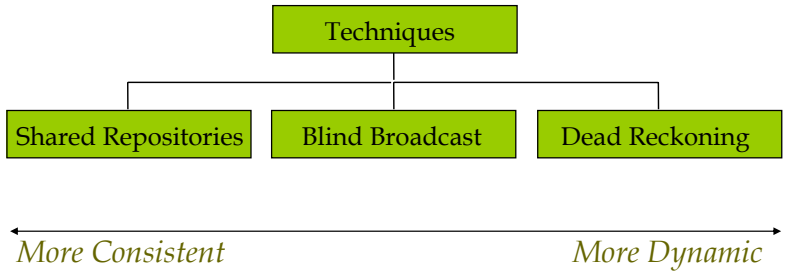
## Design Implications

- For a highly dynamic shared state, hosts must transmit more frequent data updates.
- To guarantee consistent views of the shared state, hosts must employ reliable data delivery.
- Available network bandwidth must be split between these two constraints.

## Tradeoff Spectrum

| *System Characteristic* | *Absolute Consistency* | *High Update Rate* |
|---|---|---|
| *View consistency* | *Identical at all host* | *Determined by data received at each host* |
| *Dynamic data support* | *Low: Limited by consistency protocol* | *High: Limited only by available bandwidth* |
| *Network infrastructure requirements* | *Low latency, high reliability, limited variability* | *Heterogeneous network possible* |
| *Number of participants supported* | *Low* | *Potentially high* |

## Managing Shared States



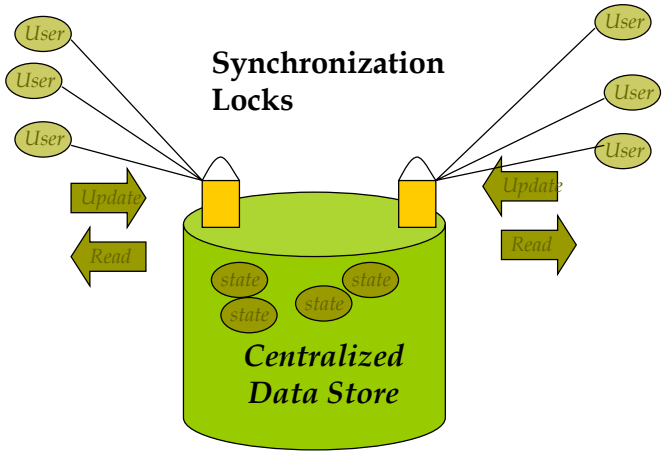

## Centralized / Shared Repositories

- Maintain shared state data in a centralized location.
- Protect shared states via a lock manager to ensure ordered writes.
- Three Techniques
  - Shared File Directory
  - Repository in Server Memory
  - Distributed Repository (Virtual Repository)

## Shared File Directory

- Absolute Consistency!
- Only one host can write data to the same file at a time. Must have locks.
- Slow!
- Does not support many users.

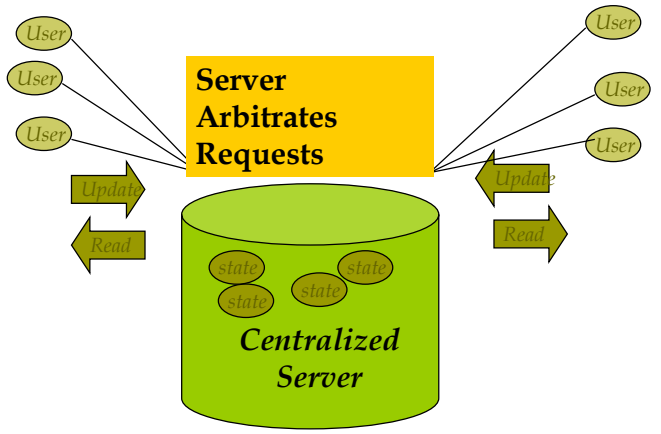
## Shared File Directory



## Server Memory

- Faster than Shared File Directory because each host uses does not have to open and close each file remotely.
- Don't have to have locks. Server arbitrates.
- Server crash is catastrophic.
- Maintaining constant connection may strain server resources.

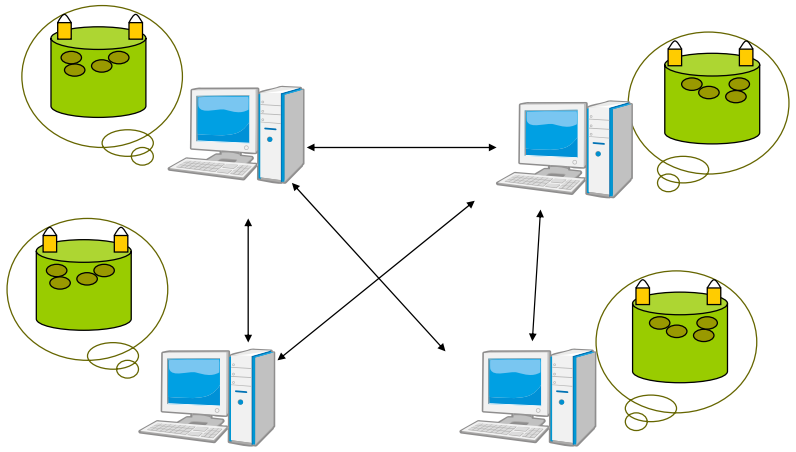
## Server Memory




## Virtual Repository

- Tries to reduce bottleneck at server.
- Hosts communicate directly to each other following a protocol of information sharing. (can even tailor who you talk to).
- Better fault tolerance (Depending on protocol creating the "virtual files" )
- Eventual Consistency.

## Virtual Repository



## Advantages of Centralized/ Shared Repositories

- Provides an easy programming model.
- Guarantees information consistency.
- No sense of data ownership is imposed; any host can update any piece of the shared state.

## Disadvantages of Shared Repositories

- Data access and update operations require an unpredictable amount of time to complete.
- Requires considerable communications overhead due to reliable data delivery.
- Vulnerable to single point failure.
- Push systems may send info where it is not needed.
- Limited number of users (else you overload the server or the network)
- One slow user can drag everyone down.

## Frequent State Regeneration/Blind Broadcasts

- Owner of each state transmits the current value asynchronously and unreliably at regular intervals.
- Clients cache the most recent update for each piece of the shared state.
- Hopefully, frequent state update compensate for lost packets.
- No assumptions made on what information the other hosts have.
- Broadcast is sent "blind" to everyone.
- Usually the entire entity state is sent.
- No acknowledgements
- No assurances of delivery
- No ordering of updates.

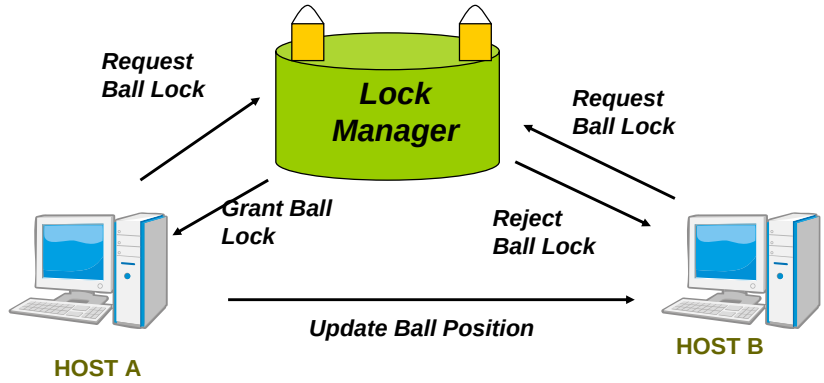
## Why use?

- Can't afford overhead of centralized repository
- May not have demanding consistency requirements

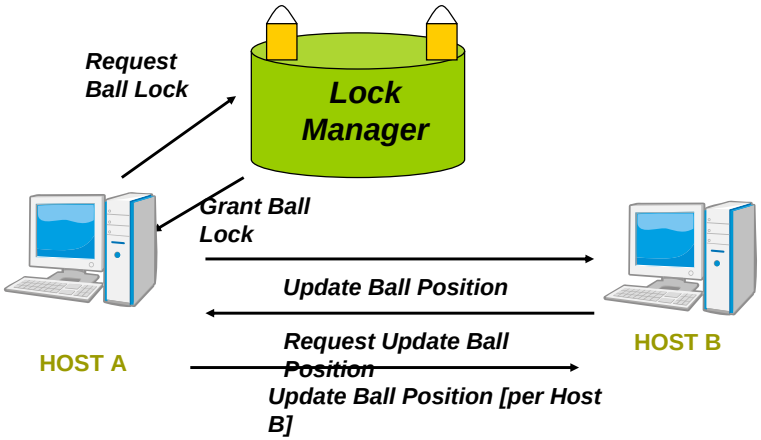
## Explicit Object Ownership

- With blind broadcasts, multiple hosts must not attempt to update an object at the same time.
- Each host takes explicit ownership of one piece of the shared state (usually the user's avatar).
- To update an un-owned piece of the shared state, either proxy updates or ownership transfer is employed.
- Unlike "locks" in Shared Filed servers, multiple updates are allowed until ownership is transferred
- Commonly used in online gaming (DOOM, Diablo), tele-surgery, and in trying to simulate video conferencing.

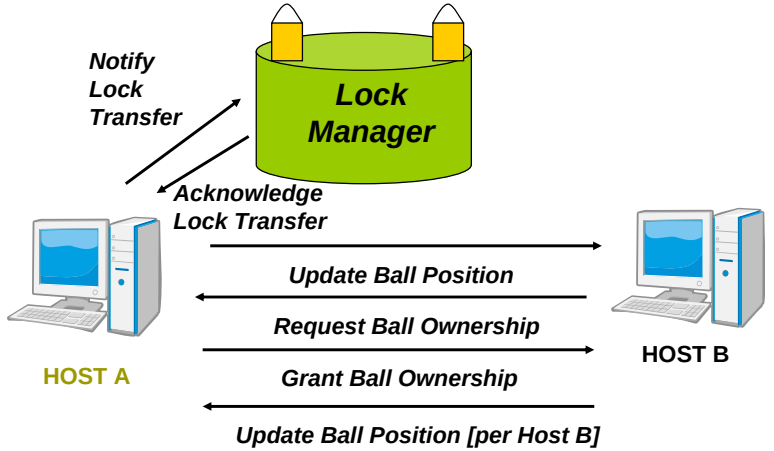
## Explicit Object Ownership Gaining Ownership




## Explicit Object Ownership Proxy Update




## Explicit Object Ownership Transferring Ownership



## Reducing Broadcast Scope

- Each host sends updates to all participants in the NVE.
- Reception of extraneous updates consumes bandwidth and CPU cycles.
- Need to filter updates, perhaps at a central server (e.g. RING system) which forwards updates only to those who can "see" each other.
- VEOS - epidemic approach- each host send info to specific neighbors.

## Advantages of Blind Broadcasts

- Simple to implement; requires no servers, consistency protocols or a lock manager (except for filter or shared items)
- Can support a larger number of users at a higher frame rate and faster response time.

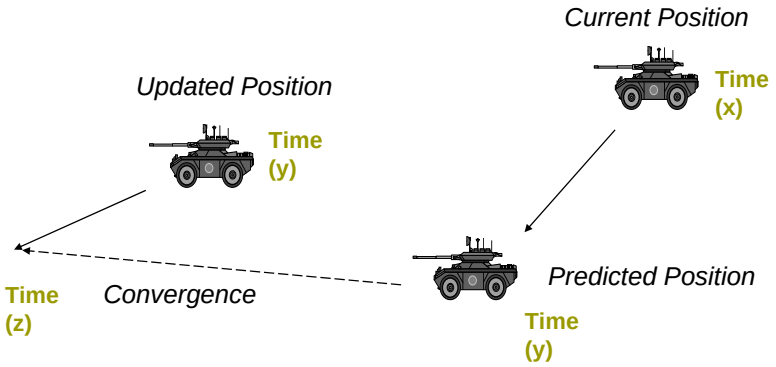
## Disadvantages of Blind Broadcasts

- Requires large amount of bandwidth.
- Network latency impedes timely reception of updates and leads to incorrect decisions by remote hosts.
- Network jitter impedes steady reception of updates leading to 'jerky' visual behavior.
- Assumes everyone broadcasting as the same rate which may be noticeable to users if it is not the case (this may be very noticeable between local users and distant destinations).

## Dead Reckoning Protocols

- Transmit state updates less frequently by using past updates to estimate the true shared state.
- Prediction
  - How the object's current state is computed based on previously received packets.
  - Each host estimates entity locations based on past data.
- Convergence
  - How the object's estimated state is corrected when another update is received.
- No need for central server.
- Sacrifices accuracy of shared state for more participants.

## Dead Reckoning Illustration




## Prediction Using Derivative Polynomials

- Zero Order (simplest)
  - $x(t + \Delta t) = x(t)$ (really state regeneration-assumes the object doesn't move)
- First Order (velocity)
  - $x(t + \Delta t) = x(t) + v_x \Delta t$
- Second Order (acceleration)
  - $x(t + \Delta t) = x(t) + v_x \Delta t + a_x (\Delta t)^2$
- Higher Order Approximations

## Hybrid Polynomial Prediction

- Need not be absolute
- Instead of using a fixed prediction scheme, dynamically choose between first or second order based on object's history.
- Use first order when acceleration is negligible or acceleration information is unreliable (changes frequently).

## Hybrid Polynomial Prediction

- Position History-Based Dead Reckoning Protocol (PHBRR) - only included position. Required the host machine to calculate velocity and acceleration based on past positions.
- Actually more accurate as "snapshot" velocities and accelerations can be misleading.

## Limitations of Derivative Polynomials

- Why not use more terms?
  - greater bandwidth required
  - greater computational complexity
  - less accurate prediction since higher order terms are harder to estimate and errors have disparate impact
- Do not take into account capabilities or limitations of objects.

## Object Specialized Prediction

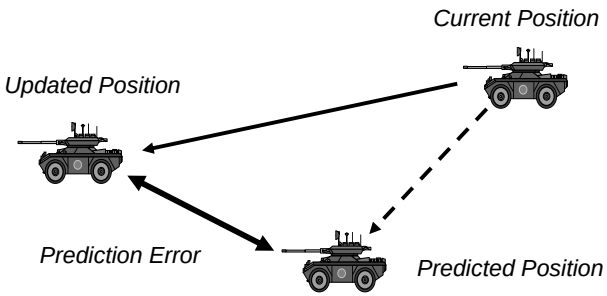
- Object behavior may simplify prediction scheme.
- A plane's orientation angle is determined solely by its forward velocity and acceleration.
- Land based objects need only two dimensions specified.

## Object Specialized Prediction

- Desired level of detail - often do not need to be precise with some aspects. Do we have to accurately model the flicker of the flames of a burning vehicle or is it enough to say it is on fire.
- The same with smoke. Some VEs need to accurately model smoke, other do not.

## Convergence Algorithms

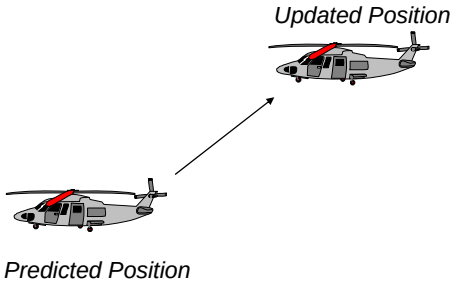
- Tells us what to do to correct an inexact prediction:




- Trade-off between computational complexity and perceived smoothness of displayed entities

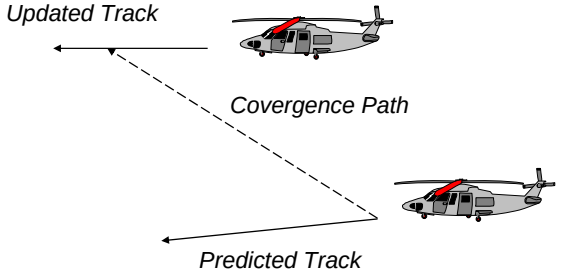
## Convergence Algorithms

- Zero order or snap convergence
- *Advantage: Simple*
- *Disadvantage: Poorly models real world & "Jumping" entities may distract users.*



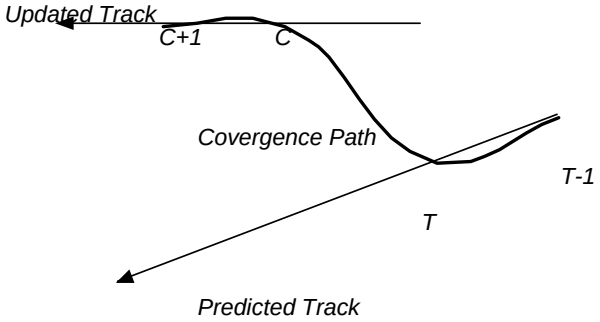

## Convergence Algorithms

- Linear Convergence
- *Advantage: Avoids jumping*
- *Disadvantage: Does not prevent "sudden" or unrealistic changes in speed or direction.*




## Convergence Algorithms

- Cubic Spline
- *Advantage: Smoothest looking convergence*
- *Disadvantage: Computationally expensive*




## Convergence Algorithms

- Choice of convergence algorithm may vary within a VE depending on the entity type and characteristics.
- Hybrids may be used (PHBRR)

## Non-Regular Updates

- Slow update rate if prediction at remote host is within an error tolerance.
- The source host models the prediction algorithm used by the remote hosts.
- Only transmit an update when an error threshold is reached or after a timeout period (heartbeat).
- Entities must have a "heartbeat" otherwise cannot distinguish between live entities and ones that have left the system.

## Advantages of Dead Reckoning

- Reduces bandwidth requirements because updates are sent less frequently.
- Potentially larger number of players.
- Each host does independent calculations

## Disadvantages of Dead Reckoning

- Not all hosts share the identical state about each entity.
- Protocols are more complex to implement to develop, maintain and evaluate.
- Must customize for object behavior to achieve best results.
- Must have convergence to cover prediction errors.
- Collision detection difficult to implement.
- Poor convergence methods lead to jerky movements and distract from immersion.

## Conclusions

- Shared state maintenance is governed by the Consistency - Throughput Tradeoff.
- Three broad types of maintenance:
  - Centralized/Shared repository
  - Frequent State Regeneration(Blind Broadcast)
  - Dead Reckoning
- The correct choice relies on balancing many issues including bandwidth, latency, data consistency, reproducibility, and computational complexity.