Lecture 4
Local Perception Filter
Bucket Synchronization
Server and clients must have a common notion of “time”
Two choices:

Wall Clock
Game Clock
Wall Clock: clients and server have to synchronize their physical clocks using NTP or SNTP.
now is 10:18:32

now is 10:18:33.5

RTT = 1s
OWD = 0.5s

now is 10:18:29.5

now is 10:18:30

Time now should be 10:18:30.5
Players try to sync the game states with the server at the same wallclock time, and predict ahead with an amount of time equal to one-way delay.
or: synchronize states to game clock, which runs behind the server by an amount of time equals to the one-way delay.
what A sees

what B sees

A  S  B

3  4  2 → game clock
Example: using game clock
A decides to move. Send event to S.
S receives event, moves A, and tells A and B that A is moving at $t = 3$
A moves itself at $t = 3$. 

![Diagram showing the movement of A at t = 3.](image-url)
B moves A at t = 3.
Example: using wallclock
A decides to move. Send event to S.
S receives event, moves A, and tells A and B that A is moving at $t = 1$
A moves itself to where it should be at $t = 2$. 
B finds out A moves at $t = 1$ and moves A to where A should be at $t = 3$
Tight synchronization allows interaction but can lead to visual disruptions.
Asynchronization allows smooth movement but hinder interaction.
Local Perception Filter
Hybrid Model: 
*Render* objects within real-time interaction range in real time, other objects in delayed time.
Two Kinds of Entities

Active:
players (unpredictable)

Passive:
bull, bullet (predictable)
happening now

A  ●  B

happened time $t$ ago
Question:
What if a player A throws a ball at player B?
A happening now

B happened time t ago
Question:
What if a player B throws a ball at player A?
happening now

A

happened time \( t \) ago

B
Distance of ball (thrown by A) from A versus time.
What A sees..

distance (from A)

time

<table>
<thead>
<tr>
<th>distance (from A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>time t ago</td>
</tr>
<tr>
<td>now</td>
</tr>
</tbody>
</table>
Distance of ball (thrown by B) from A versus time.

Distance (from A)
What A sees..

distance (from A)

time

time t ago

now

time
happened $t$ ago
happened $t - 1$ ago
happened $t - 2$ ago
Question:
What if a player A throws a ball at player B?
What A sees..

distance

time
What A sees..

distance

time
Question:
What if a player B throws a ball at player A?
Distance of ball (thrown by B) from A versus time.
Event Ordering for P2P Architecture
Point-to-Point Architecture

Role of clients
Notify clients
Resolve conflicts
Maintain states
Simulate games
**AD-HOC MODE**

Compete against players on nearby PSP® systems. Make sure your Wi-Fi switch is on and you can communicate with local PSP® systems without an Internet connection. You can play together in a house, a backyard, an airport, a lobby - anywhere! Depending on the game, up to 16 PSP® systems within range of each other can be connected using Ad Hoc mode.

For local offline multiplayer gaming (Ad Hoc mode) you need:

- multiple players on PSP® systems within range of each other
- each with a PSP® set up with Ad Hoc on "automatic" or the same channel
- each with a PSP® game that supports multiplayer Ad Hoc mode

**INFRASTRUCTURE MODE**

Play online with people across the globe, all from the comfort of a Wi-Fi hotspot. Just check the back of the game packaging to see if it supports infrastructure mode, connect to a hotspot, insert the game UMD™, follow the on-screen instructions and start playing online!

For online multiplayer gaming (Infrastructure Mode) you need:

- a Wi-Fi Internet hotspot or wireless home network
- a PSP connected to that hotspot or network
- a PSP® game that supports multiplayer Infrastructure mode

**Infrastructure Multiplayer Games**
Age of Empire Series

Received-Order Delivery
Server executes the events as they are received.
Bucket Synchronization
The game is divided into rounds (e.g. 25 rounds per second).
Players are expected to send an event in each round (e.g. 25 updates per second).
Players are expected to send an event in each round (e.g. 25 updates per second).
Every round has a “bucket” that collects the events.
Events generated in the same round go into the same bucket of a future round. We know which round an event is generated in, based on time-stamp.
In each round, the events in the bucket are processed (in order of time-stamp).
Two parameters needed: **round length** and **lag**. Lag depends on maximum network latency among the players; round length depends on rendering speed.

![Diagram showing round length and lag between players A, B, and C.](attachment:diagram.png)
Players periodically ping among themselves and exchange latency information. They also measure the average time taken to render a frame and exchange that information to estimate lag and round length.
If events from another player is lost (or late), we can predict its update (e.g. using dead reckoning) when possible.
Alternative is to ensure every event is received before executing the bucket. What are the drawbacks?
Stop-and-Wait Protocol
aka Synchronized Simulations
Every player sees exactly the same states (but maybe at different time)
Getting all clients to simulate the exact same thing is tricky: must use same number of random calls with same seeds, same precision of floating point calculation etc.
Clients may periodically exchange hash of game states to detect if a player has gone out-of-sync.