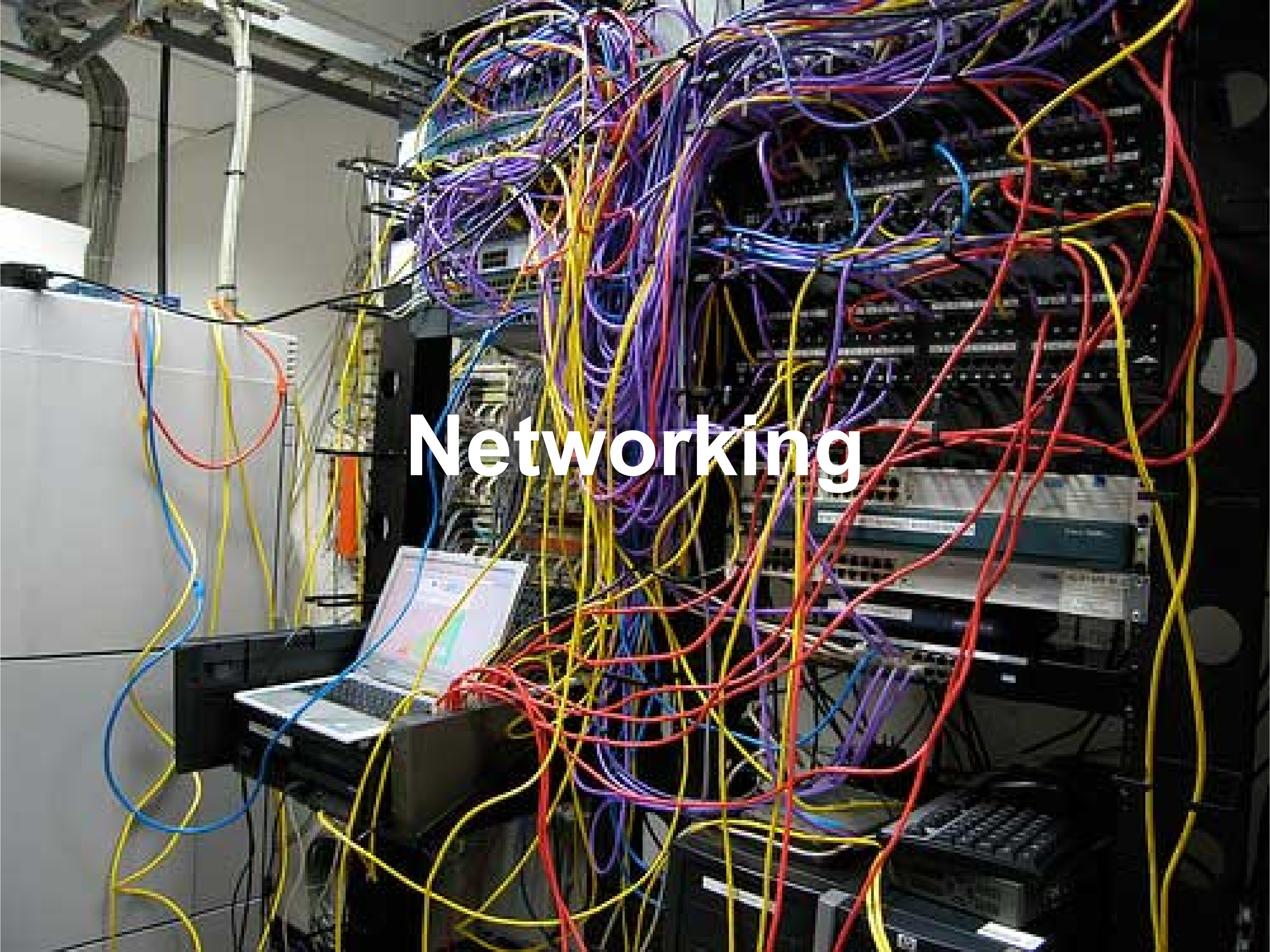
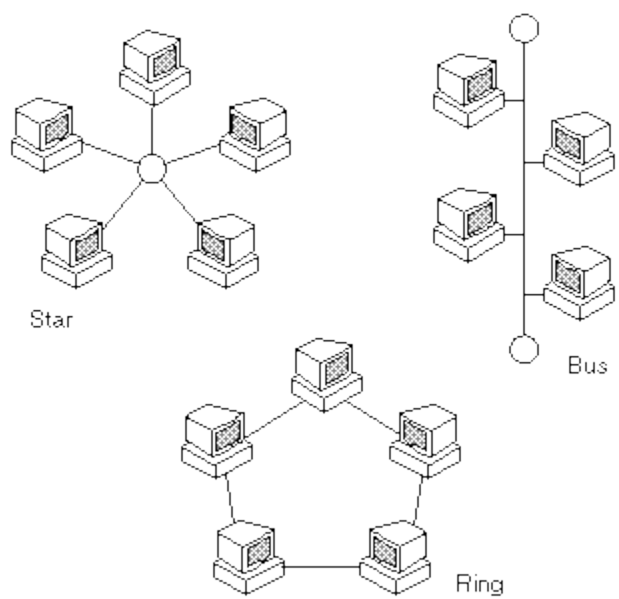
# Networking



# What is a network?

- Not a series of tubes!
  - A graph of connected nodes (servers, routers, ...)
  - Information sent in small packets (~1000 bytes)
  - Packets are unreliable and may vanish or arrive twice



# Network Basics

- IP Address
  - Number that identifies a node on a network
  - 32 bits for IPv4, 128 bits for IPv6
- UDP: User Datagram Protocol
  - Lower-level, basically packets with port numbers
  - Send data in chunks
  - Same characteristics as packets: unreliable and non-ordered
- TCP: Transmission Control Protocol
  - Higher-level, abstracts away packets
  - Send data in a continuous stream
  - Data is reliable and ordered (resends dropped packets and ignores duplicates)

# Networking in Games

- Goals
  - Consistent state
  - Minimize latency
  - Prevent cheating
- Challenges
  - Long latencies due to distance and congestion
  - Variable latency due to unreliable network
  - Game state may be larger than available bandwidth
  - Need to interpolate game state between updates
  - Antagonistic users

# A First Attempt

- Peer-to-peer
  - Each client simulates itself, sends its state to everyone else
- Pros
  - Simple
  - Minimizes latency (no overhead of a server)
- Cons
  - Game state not consistent (each client has a different game state)
  - No authoritative game state (who won?)
  - Cheating is easy (dodging)
- Still used in racing games

# "Dumb Terminal" Model

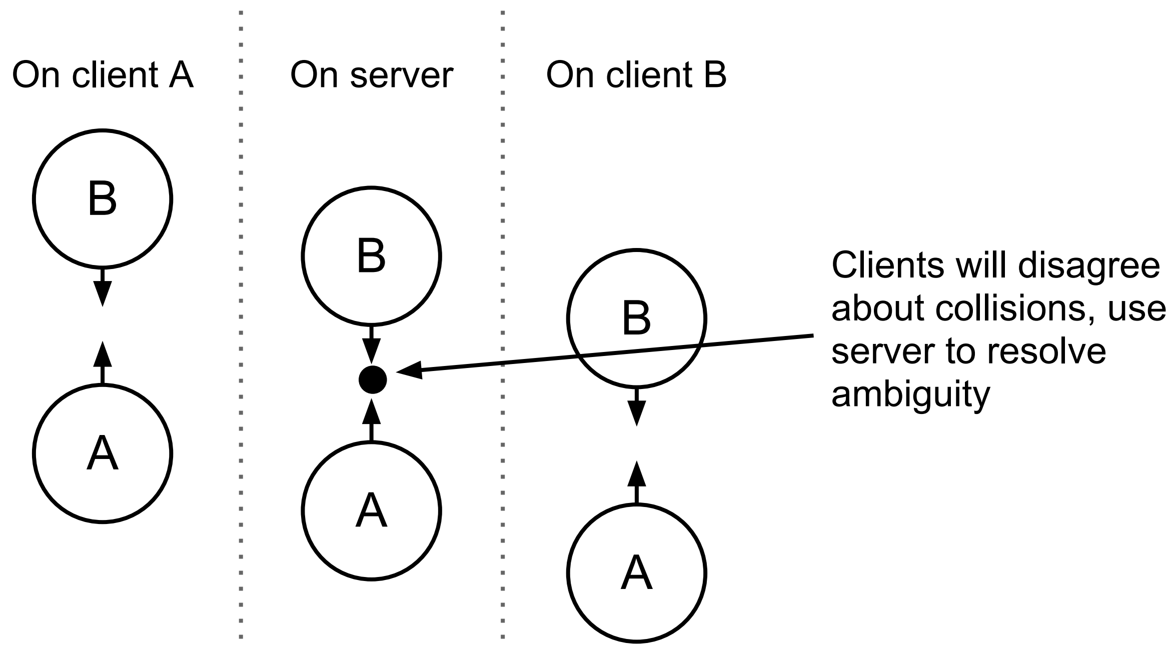
- Client/server
  - Client sends keyboard and mouse events to server
  - Server simulates next screen for client
  - Server sends information for screen frames to client
- Pros
  - Simple
  - Consistent authoritative game state
  - Eliminates some forms of cheating
- Cons
  - Clients no longer move instantly, have to wait for round-trip time (RTT)
- Used in original Quake and by OnLive

# Instant Local Movement

- Client/server
  - Move local player instantly (don't wait for confirmation from server)
  - Also runs game simulation, server still authoritative
- Pros
  - Faster user feedback
  - Still avoids cheating, server still authoritative
- Cons
  - Complicated
  - Clients no longer see one consistent world state (remote players are always behind local player)

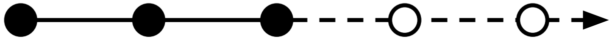
# Instant Local Movement

- Server still needs to be authoritative
  - Don't create entities without server confirmation
  - Server should be able to set position of local player



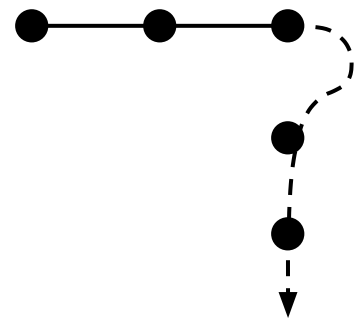
# Client-Side Prediction

- Prediction
  - Extrapolating the current properties of an entity based on historical authoritative data and local guesses about the future
- Only get updates periodically (~10Hz)
  - Need to interpolate/extrapolate positions between updates
  - Usually want smooth interpolation, can include velocity in smoothing too



# Client-Side Prediction

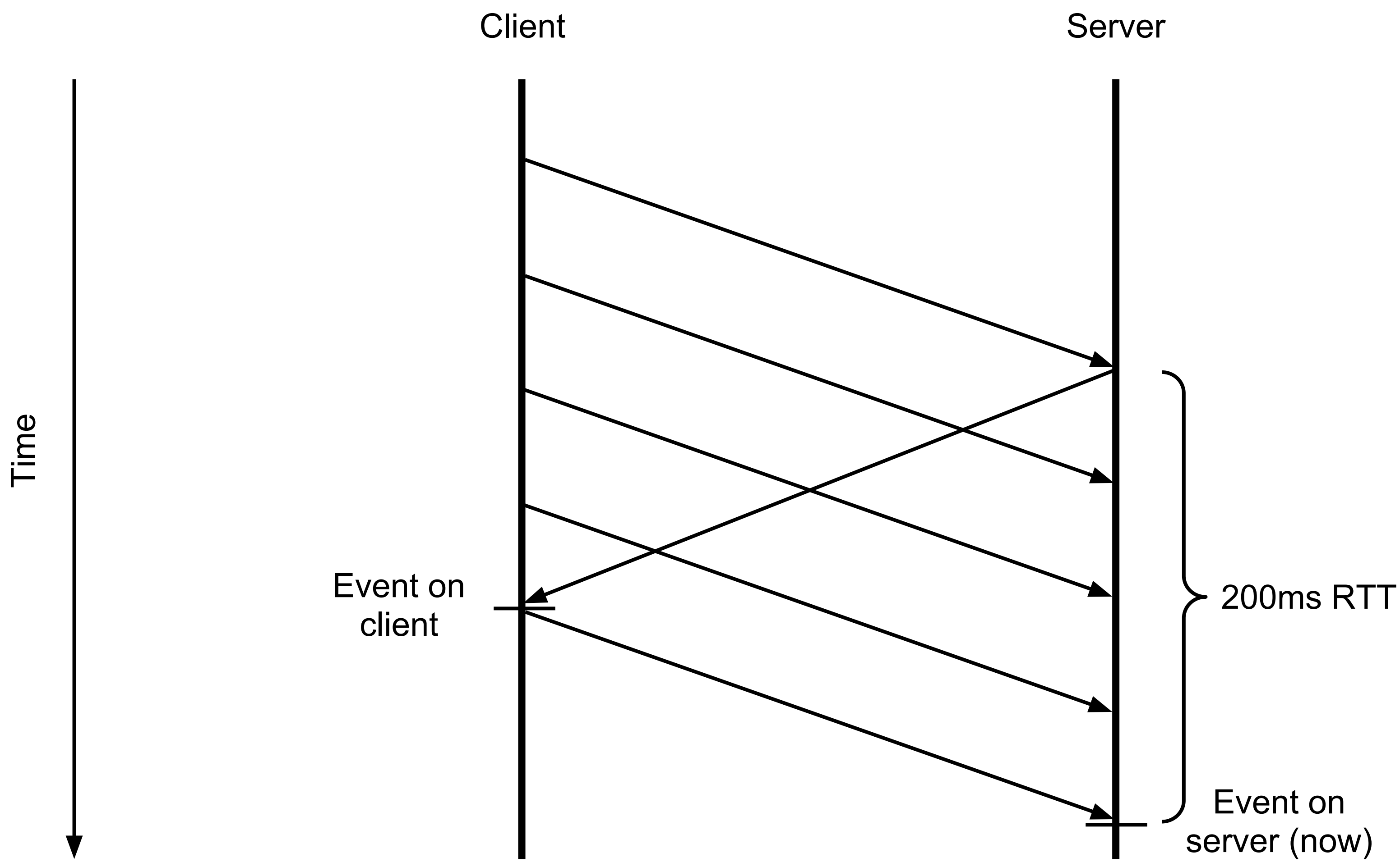
- Prediction
  - Extrapolating the current properties of an entity based on historical authoritative data and local guesses about the future
- Only get updates periodically (~10Hz)
  - Need to interpolate/extrapolate positions between updates
  - Usually want smooth interpolation, can include velocity in smoothing too



# Lag Compensation

- Problems with Instant Local Movement
  - Hard to use, especially for FPS games
  - Have to lead shots
- Goals
  - Want players to not notice any lag when shooting
  - Could just have clients determine hits, but cheating
- Lag compensation
  - Clients each run their own game simulation
  - Server runs one main simulation but temporarily calculates client world state (compensates for lag) for important decisions
- Complicated

# Lag Compensation



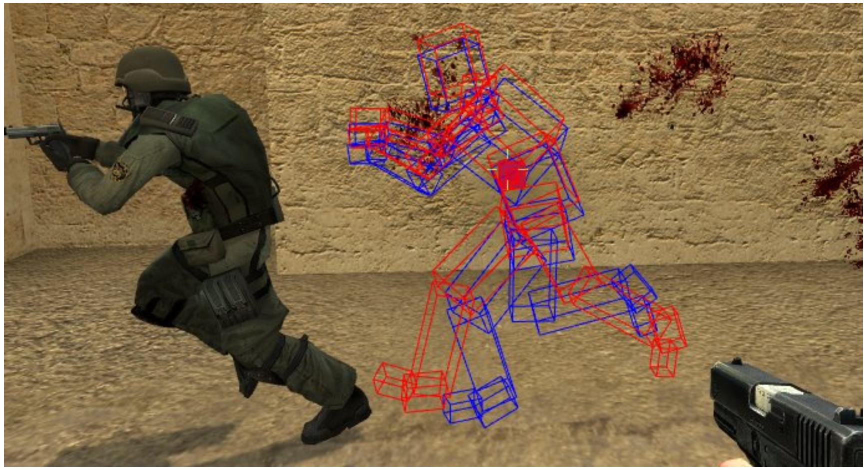
# Lag Compensation

The server will, on receiving an action:

1. Compute player's latency (say 200ms RTT)
2. Roll back world state by 200ms
3. Extrapolate that player forward by 200ms using updates received in the last 200ms (since local players are always ahead of remote players)
4. Execute the action
5. Return objects to current position, recomputing current state for affected objects

# Lag Compensation Example

- Image below is when server registers a hit
  - From the Source engine (Valve)
- Server rolls back target 200ms (RTT)
  - Position indicated by blue hitboxes
- 100ms ago on client, target at red position
  - Difference due to time measurement errors



# Lag: Gameplay Implications

- Without lag compensation
  - With trying to shoot / target an opponent, need to lead based on latency
  - Unrealistic for the shooter
  - Realistic for the shootee
- With lag compensation
  - Aim directly at target
  - Possible for player to be shot around corner
    - Player A (laggy) shoots at player B, player B runs behind a wall, then server receives A's command, rolls back B's position, registers a hit
  - Realistic for the shooter
  - Unrealistic for the shootee

# Hiding the Lag

- Alternatives to lag compensation
- Example 1: Throwing a bomb
  - Must wait for server to spawn bomb entity
  - Can still start bomb throwing animation
- Example 2: Directing a unit to move
  - Must wait until all clients have that command
  - Can still play the "Right away commander!" sound

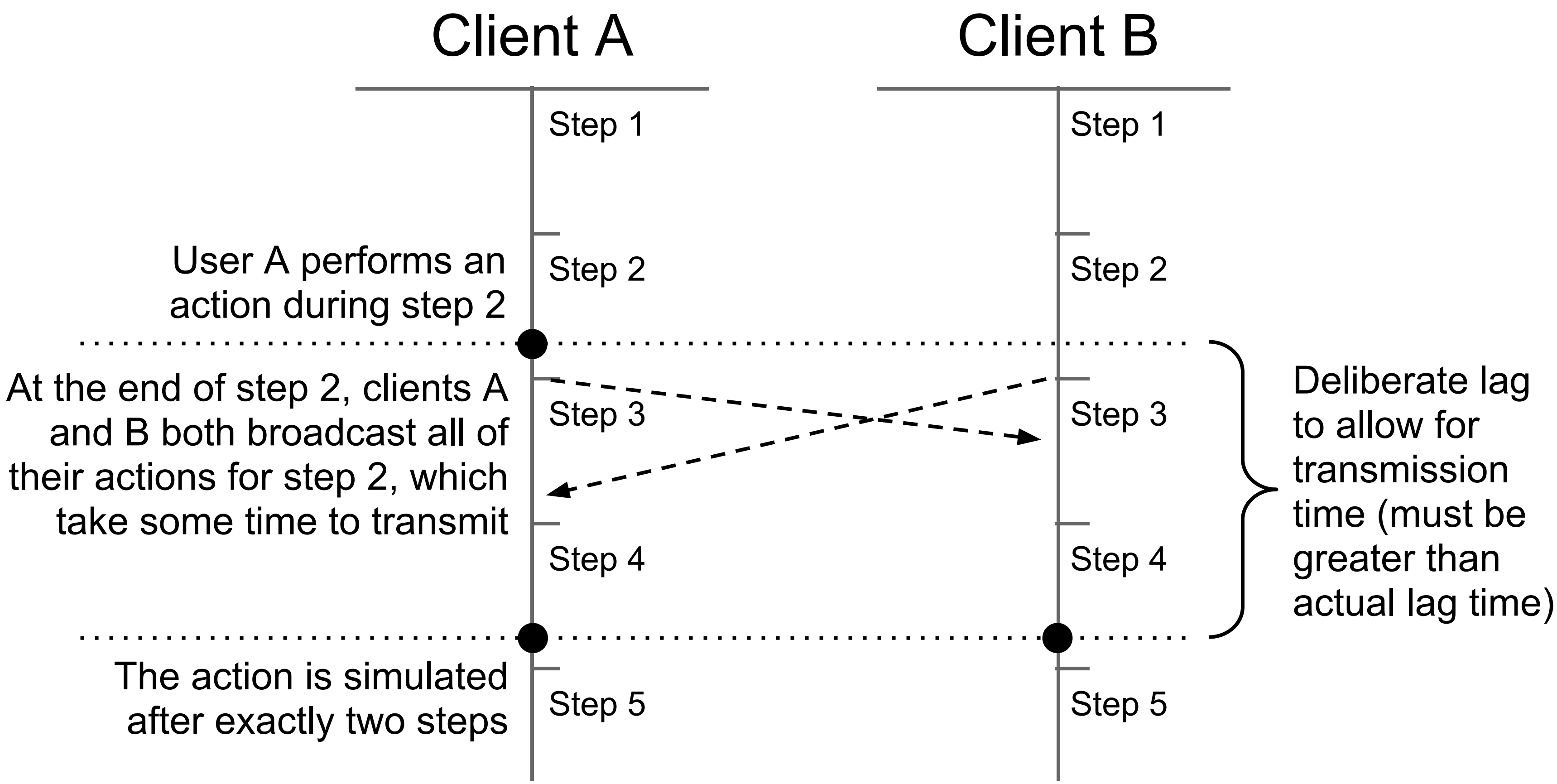
# State Replication

- Can't send entire game state per update
  - Not enough bandwidth
  - Most state not relevant anyway
- Only send state relevant to player
  - Objects near player
  - Objects in player's field of view
- Advanced: per-object priorities
  - Always too much data, write each packet until full
  - Priorities based on importance and size in packet
  - Player > AI > projectile > pickup > vehicle
  - Paper: The TRIBES Engine Networking Model

# Lockstep Networking

- Client/server or peer-to-peer
- Each peer simulates exact same game
  - Only transmit user input
  - Game must be deterministic down to the last bit!
  - All clients are authoritative (all have same state)
- Game time divided up into fixed-sized steps
  - Peers send out actions for the next step
  - Step cannot be simulated until actions received from all peers for that step
  - All actions delayed by at least two steps
    - Allows for some latency

# Lockstep Networking



# Lockstep Networking

- Pros
  - Consistent game state
  - Minimal network traffic
  - Great for replays, just store the inputs and re-run the simulation
- Cons
  - Supporting joining the game late is difficult
  - Everyone as lagged as the highest latency player
  - Simulations must be deterministic!
- Used heavily in industry
  - Ideal for turn-based and RTS games
  - Only option for games with huge state sizes

# Simulation Determinism

- Programs are surprisingly non-deterministic
  - All random streams must be seeded the same
  - Must use the same number of calls to random()
  - Must update all objects in the same global order
  - Cannot sort or hash by pointers (e.g. set<Obj *>)
  - Must ensure floating-point rounding mode is consistent (especially after library calls)
  - Won't work across architectures (x86 vs ARM), different optimization levels, or different compilers (or even different versions of the same compiler!)
- Floating point alternatives
  - Integer/fixed-point math (might need 64 bits)
  - Use Java, which has more floating-point guarantees

# Simulation Determinism

- Desync bugs are catastrophic
  - Desync: internal state of two simulations diverge, even by one bit
  - Won't notice it at first, later entire game will be different (butterfly effect)
- Desync bugs are hard to debug
  - Send hash of game state across the network
  - When hashes differ, immediately stop and log large amounts of information
- Lockstep networking is difficult to get right
  - But still widely used

# Debugging Multiplayer Games

- Problem: Breakpoints only stop one client
  - Other clients keep running!
  - Connection will time out and break session
- Synchronous debugging
  - Clients keep time since last update
  - If above a threshold (0.5 seconds), halt and wait for response before continuing
- Journaling (for really tricky bugs)
  - Record user inputs, frame times, packets received
  - Can play back entire session without network

# Middleware: RakNet

- Industry standard
  - Open source, cross platform C++
  - Free for indie games (under $50K)
  - Large feature set (lobby, RPC, patching, ...)
  - Wide usage: Unity, Maxis, Demiurge
- Optimized for games
  - Implements TCP-like reliable packet delivery over UDP for lower latency
  - Fine-grained control over how packets are delivered
  - Supports compression and state replication

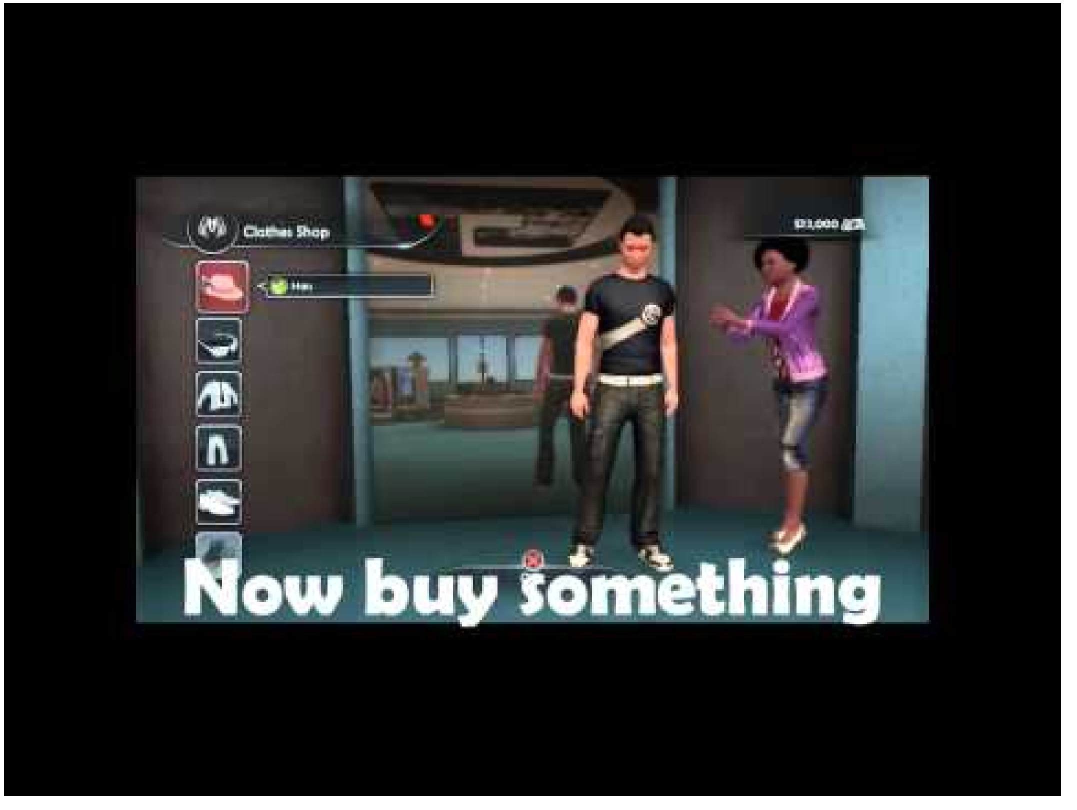
# Final Advice

- Networking is hard!
  - Make sure you design for it from the beginning
  - Correct networking necessitates architectural decisions that can't easily be added afterwards
- Always ask:
  - Where is the lag?
  - Who is authoritative for each object?
- Don't do lockstep unless you enjoy IEEE 754 compliance

# Cheating in Games

# Cheating in Games

Memory scanning (Tsearch)




http://www.youtube.com/watch?v=ojwYkDWUQeY

# Cheating in Games

● Read DirectX calls and send mouse inputs

http://www.youtube.com/watch?v=FBZT2ukipkc

# Cheating in Games

- Exploit game logic loopholes
  - Warcraft III forfeit loophole
- Hired help
  - Gold farming in World of Warcraft
- Bots / reflex enhancers
  - Aimbots in FPS games
- Modifying data
  - Transparent wall textures (wallhacks)
- Scripting viruses
  - Arbitrary server uploads

# Cheating Protection

- Continuous bug patching
- Bot pattern detection
- Anti-cheat techniques
  - Try to detect debugger hooks
  - PunkBuster, VAC
- Community reporting of cheaters
  - Banned by moderators
- Impossible to fully prevent cheating
  - Can only make it harder

# Cheating in Networked Games

- Suppressed update
  - Protocol has wiggle room for latency
  - Wait for that time, then send the optimal move
- P2P cheats
  - Send different info to different peers
  - If using timestamps, set timestamps in the past
- Information exposure
  - Packet sniffing
- Denial of service
- Spoofing and replay attacks

# Networked Cheating Protection

- Encryption with per-message nonce
- Lockstep
  - Commit/reveal
- Purely local replication
  - Don't send irrelevant info
- Consistency among random peers

# Alternative Prevention Measures

- Trusted computing (consoles)
  - Raises effort level significantly
  - Still succumbs to physical attacks
- Real-money auction house
  - If you can't beat 'em, join 'em
  - Diablo III, Blizzard charges for listing
- Achievements
  - Most players won't risk losing everything

# C++ Tip of the Week

- Expression Templates
  - Represent AST of an expression using templates
  - Expression can be stored and evaluated later
- Simple example: Vector operations
  - Basic libraries make a copy
  - Expression templates to create an optimized loop on assignment

```
// The syntax we want
d = a + b + c;

// Goal for generated code: no temporary copies
for (i) { d[i] = a[i] + b[i] + c[i]; }
```

# Expression Templates Example

```
struct Vector {
  std::vector<float> d;
  float operator [] (int i) const { return d[i]; }
  template <typename T> void operator = (const T &v) {
    for (int i = 0; i < d.size(); i++) d[i] = v[i]; }
};

template <typename L, typename R> struct Plus {
  const L &l; const R &r;
  Plus(const L &l, const R &r) : l(l), r(r) {}
  float operator [] (int i) const { return l[i] + r[i]; }
};

template <typename L, typename R>
Plus<L, R> operator + (const L &l, const R &r) { return Plus<L, R>(l, r); }

Vector a, b, c, d;
d = a + b + c; // type is Plus<Plus<Vector, Vector>, Vector>
```

# Case Study: Boost.Lambda

- Added anonymous functions before C++11
  - Lazy evaluation
  - Placeholders are arguments of lambda: _1, _2, _3

```
using namespace boost::lambda;

// Useful with STL algorithm function templates
for_each(a.begin(), a.end(), std::cout << _1 << ' ');

// Can also bind to members using member pointers
sort(b.begin(), b.end(),
  bind(&Foo::x, _1) < bind(&Foo::x, _2));
```

# References


- http://gafferongames.com/networking-for-game-programmers/
- http://udn.epicgames.com/Three/NetworkingOverview.html
- http://www.pingz.com/wordpress/wp-content/uploads/2009/11/tribes_networking_model.pdf
- http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking
- http://www.gamasutra.com/view/feature/3094/1500_archers_on_a_288_network_.php
- http://blog.wetfish.net/starcraft-2-api-for-bots-ai/
- "Cheating in networked computer games – A review," Steven Daniel Webb and Sieteng Soh (http://delivery.acm.org/10.1145/1310000/1306839/p105-webb.pdf)