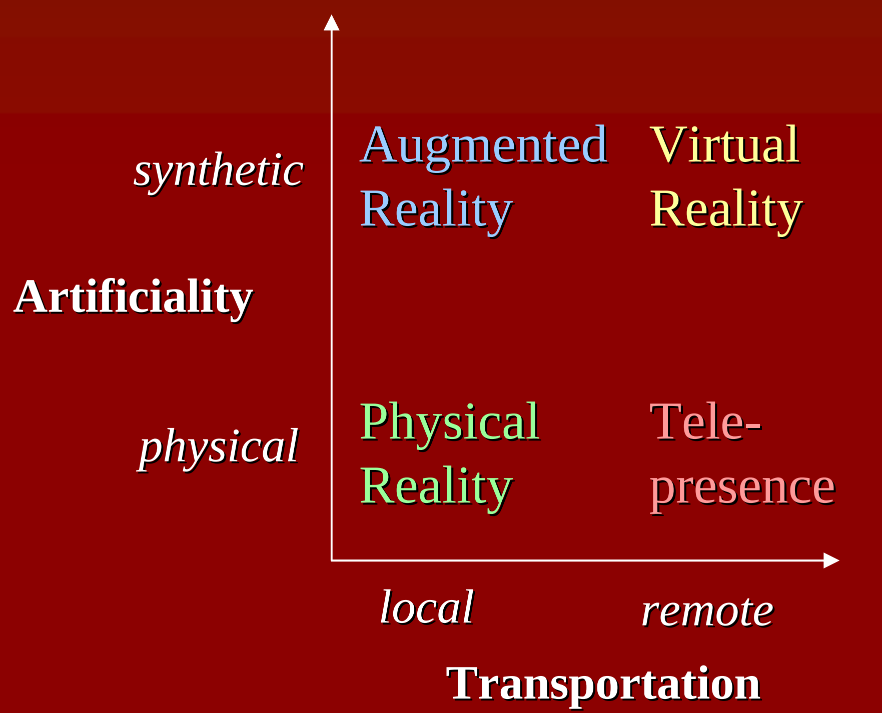
Algorithms and Networking for Computer Games

Chapter 8: Communication Layers

Communication layers

1. Physical platform
2. Logical platform
3. Networked application

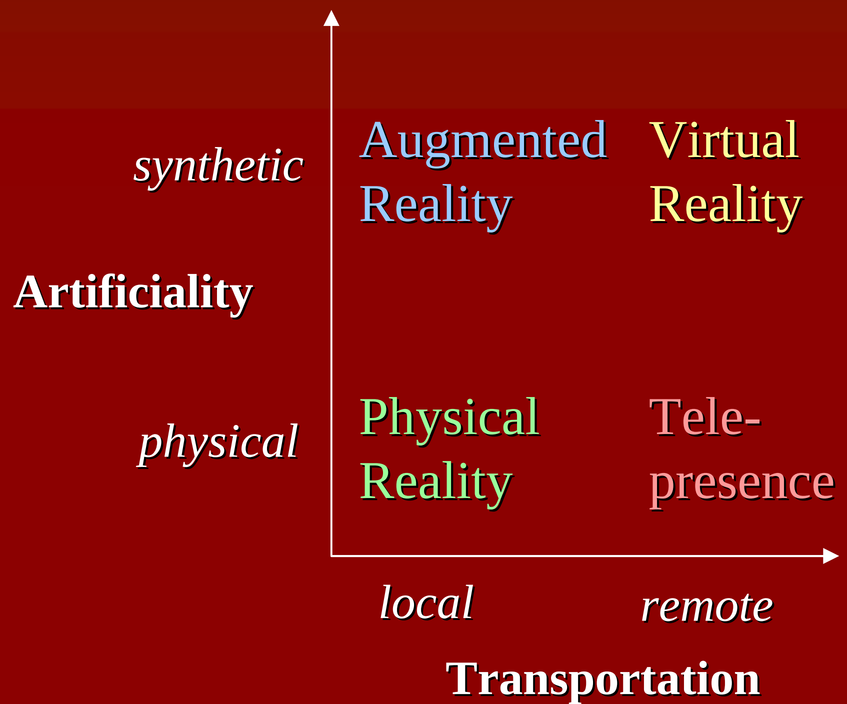
Classification of shared-space technologies 1(2)



Benford et al., 1998

- Physical reality
  - resides in the local, physical world
  - here and now
- Telepresence
  - a real world location remote from the participant's physical location
  - a remote-controlled robot

Classification of shared-space technologies 2(2)



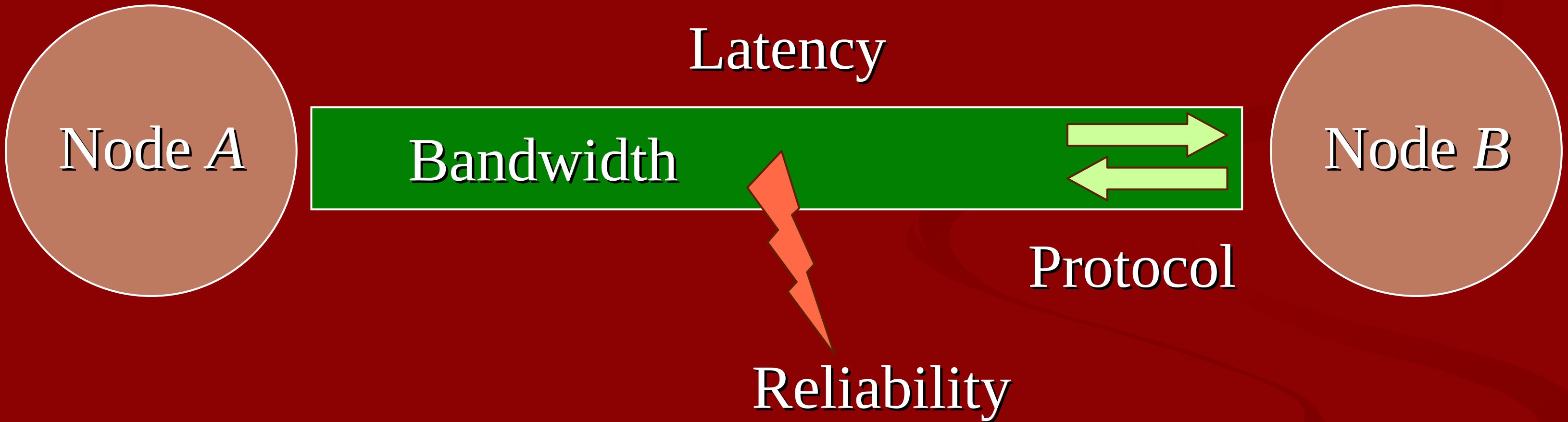
Benford et al., 1998

- Augmented reality
  - synthetic objects are overlaid on the local environment
  - a head-up display (HUD)
- Virtual reality
  - the participants are immersed in a remote, synthetic world
  - multiplayer computer game

Physical platform

- resource limitations
  - bandwidth
  - latency
  - processing power for handling the network traffic
- transmission techniques and protocols
  - unicasting, multicasting, broadcasting
  - Internet Protocol, TCP/IP, UDP/IP

Network communication



Data transfer 1(3)

- Network latency
  - network delay
  - the amount of time required to transfer a bit of data from one point to another
  - one of the biggest challenges:
    - impacts directly the realism of the game experience
    - we cannot much to reduce it
  - origins
    - speed-of-light delay
    - endpoint computers, network hardware, operating systems
    - the network itself, routers

Data transfer 2(3)

- Network bandwidth
  - the rate at which the network can deliver data to the destination host (bits per second, bps)
- Network reliability
  - a measure of how much data is lost by the network during the journey from source to destination host
  - types of data loss:
    - dropping: the data does not arrive
    - corruption: the content has been changed

Data transfer 3(3)

- Network protocol
  - a set of rules that two applications use to communicate with each other
  - packet formats
    - understanding what the other endpoint is saying
  - packet semantics
    - what the recipient can assume when it receives a packet
  - error behaviour
    - what to do if (when) something goes wrong

Internet Protocol (IP)

- Low-level protocols used by hosts and routers
- Guides the packets from source to destination host
- Hides the transmission path
  - phone lines, LANs, WANs, wireless radios, satellite links, carrier pigeons…
- Applications rarely use the IP directly but the protocols that are written on top of IP
  - Transmission Control Protocol (TCP/IP)
  - User Datagram Protocol (UDP/IP)

TCP versus UDP

Transmission Control Protocol (TCP/IP)

- Point-to-point connection
- Reliable transmission using acknowledgement and retransmission
- Stream-based data semantics
  - data checksums
- Big overhead
- Hard to 'skip ahead'

User Datagram Protocol (UDP/IP)

- Lightweight data transmission
- Differs from TCP
  - connectionless transmission
  - 'best-efforts' delivery
  - packet-based data semantics
- Packets are easy to process
- Transmission and receiving immediate
- No connection information for each host in the operating system
- Packet loss can be handled

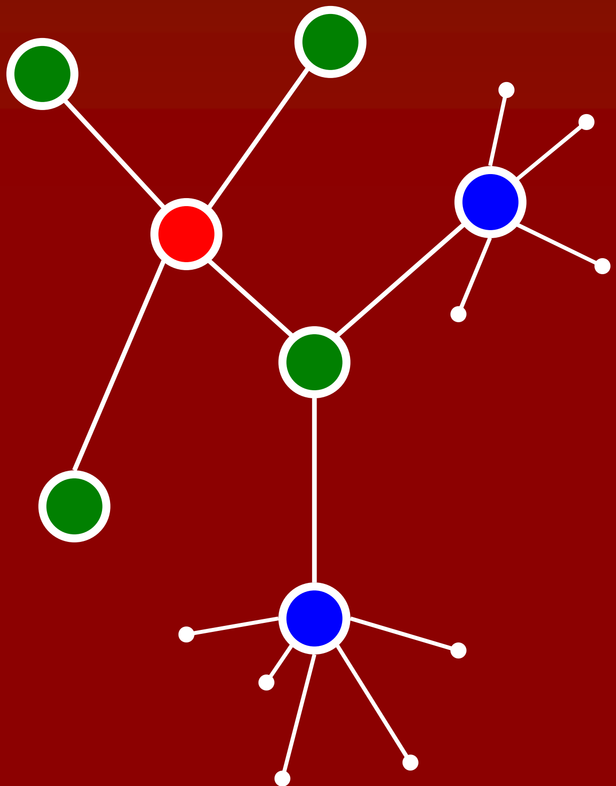
Transmission techniques

- Unicasting
  - single receiver
- Multicasting
  - one or more receivers that have joined a multicast group
- Broadcasting
  - all nodes in the network are receivers

IP Broadcasting

- Using a single UDP/IP socket, the same packet can be sent to multiple destinations by repeating the send call
  - 'unicasting'
  - great bandwidth is required
  - each host has to maintain a list of other hosts
- IP broadcasting allows a single transmission to be delivered to all hosts on the network
  - a special bit mask of receiving hosts is used as a address
- With UDP/IP, the data is only delivered to the applications that are receiving on a designated port
- Broadcast is expensive
  - each host has to receive and process every broadcast packet
- Only recommended (and only guaranteed) on the local LAN
- Not suitable for Internet-based applications

IP multicasting 1(3)



- Packets are only delivered to subscribers
- Subscribers must explicitly request packets from the local distributors
- No duplicate packets are sent down the same distribution path
- Original 'publisher' does not need to know all subscribers
- Receiver-controlled distribution

IP multicasting 2 (3)

- 'Distributors' are multicast-capable routers
- They construct a multicast distribution tree
- Each multicast distribution tree is represented by a pseudo-IP address (multicast IP address, class D address)
  - 224.0.0.0–239.255.255.255
  - some addresses are reserved
  - local applications should use 239.0.0.0–239.255.255.255
- Address collisions possible
  - Internet Assigned Number Authority (IANA)
- Application can specify the IP time-to-live (TTL) value
  - how far multicast packets should travel
  - 0: to the local host
  - 1: on the local LAN
  - 2–31: to the local site (network)
  - 32–63: to the local region
  - 64–127: to the local continent
  - 128–254: deliver globally

IP multicasting 3(3)

- Provides desirable network efficiency
- Allows partitioning of different types of data by using multiple multicast addresses
- The players can announce their presence by using application's well-known multicast address
- Older routers do not support multicasting
- Multicast-aware routers communicate directly by 'tunneling' data past the non-multicast routers (Multicast Backbone, Mbone)
  - Participant's local router has to be multicast-capable

Selecting a protocol 1(4)

- Multiple protocols can be used in a single system
- Not which protocol should I use in my game but which protocol should I use to transmit *this piece of information*?
- Using TCP/IP
  - reliable data transmission between two hosts
  - packets are delivered in order, error handling
  - relatively easy to use
  - point-to-point limits its use in large-scale multiplayer games
  - bandwidth overhead

Selecting a protocol 2(4)

- Using UDP/IP
  - lightweight
  - offers no reliability nor guarantees the order of packets
  - packets can be sent to multiple hosts
  - deliver time-sensitive information among a large number of hosts
  - more complex services have to be implemented in the application
    - serial numbers, timestamps
  - recovery of lost packets
    - positive acknowledgement scheme
    - negative acknowledgement scheme
      - more effective when the destination knows the sources and their frequency
  - transmit a quench packet if packets are received too often

Selecting a protocol 3(4)

- Using IP broadcasting
  - design considerations similar to (unicast) UDP/IP
  - limited to LAN
  - not for games with a large number of participants
  - to distinguish different applications using the same port number (or multicast address):
    - Avoid the problem entirely: assign the necessary number
    - Detect conflict and renegotiate: notify the participants and direct them to migrate a new port number
    - Use protocol and instance magic numbers: each packet includes a magic number at a well-known position
    - Use encryption

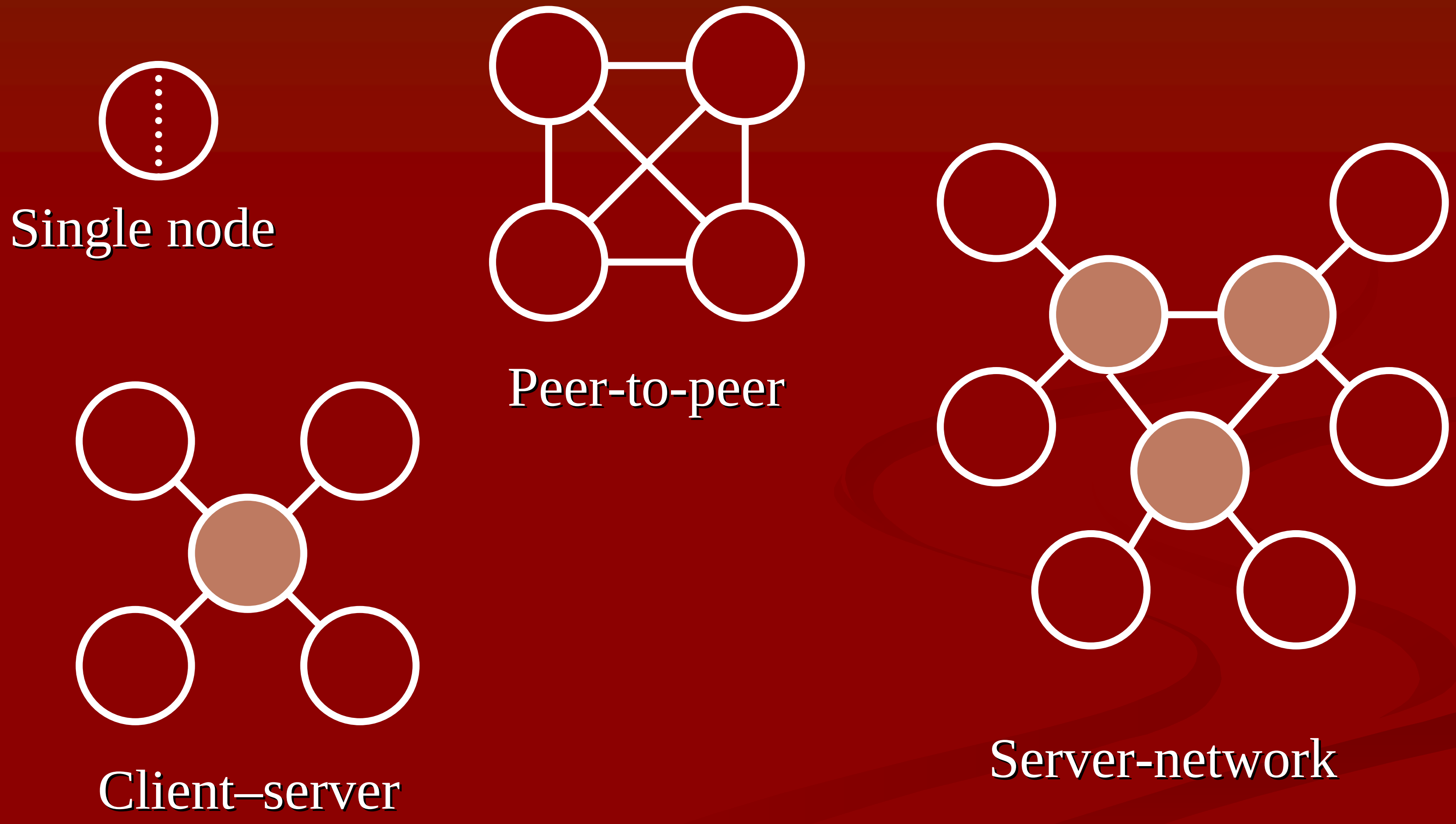
Selecting a protocol 4(4)

- Using IP multicasting
  - provides a quite efficient way to transmit information among a large number of hosts
  - information delivery is restricted
    - time-to-live
    - group subscriptions
  - preferred method for large-scale multiplayer games
  - how to separate the information flows among different multicast groups
    - a single group/address for all information
    - several multicast groups to segment the information

Logical platform

- communication architecture
  - peer-to-peer
  - client-server
  - server-network
- data and control architecture
  - centralized
  - replicated
  - distributed

Communication architecture



Communication architecture (cont'd)

- Logical connections
  - how the messages flow
- Physical connections
  - the wires between the computers
  - the limiting factor in communication architecture design

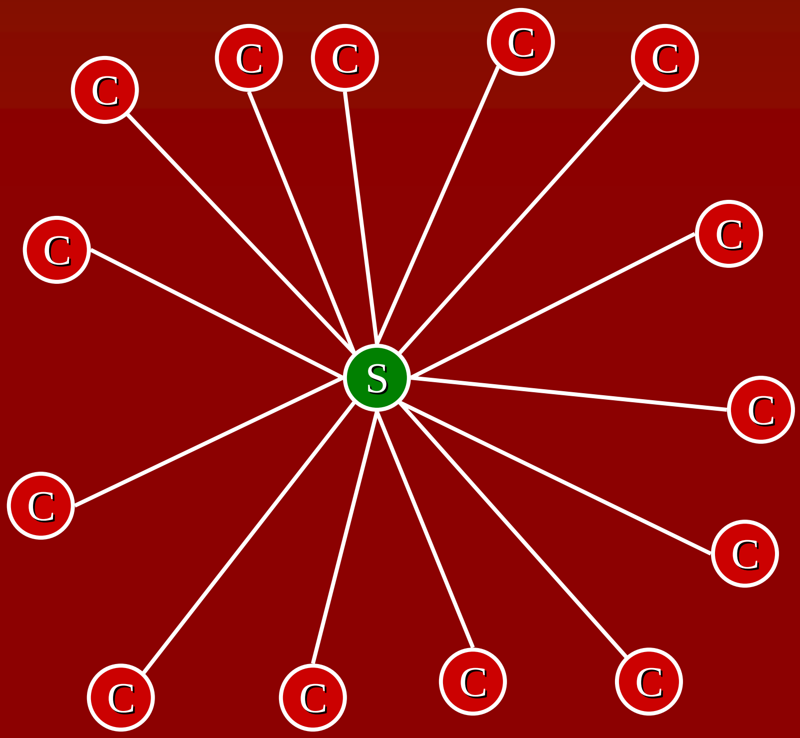
Multiplayer client–server systems: logical architecture

- Client-server system
  - each player sends packets to other players via a server
- Server slows down the message delivery
- Benefits of having a server
  - no need to send all packets to all players
  - compress multiple packets to a single packet
  - smooth out the packet flow
  - reliable communication without the overhead of a fully connected game
  - administration

Multiplayer client–server systems: physical architecture (on a LAN)

- All messages in the same wire
- Server has to provide some added-value function
  - collecting data
  - compressing and redistributing information
  - additional computation

Traditional client–server

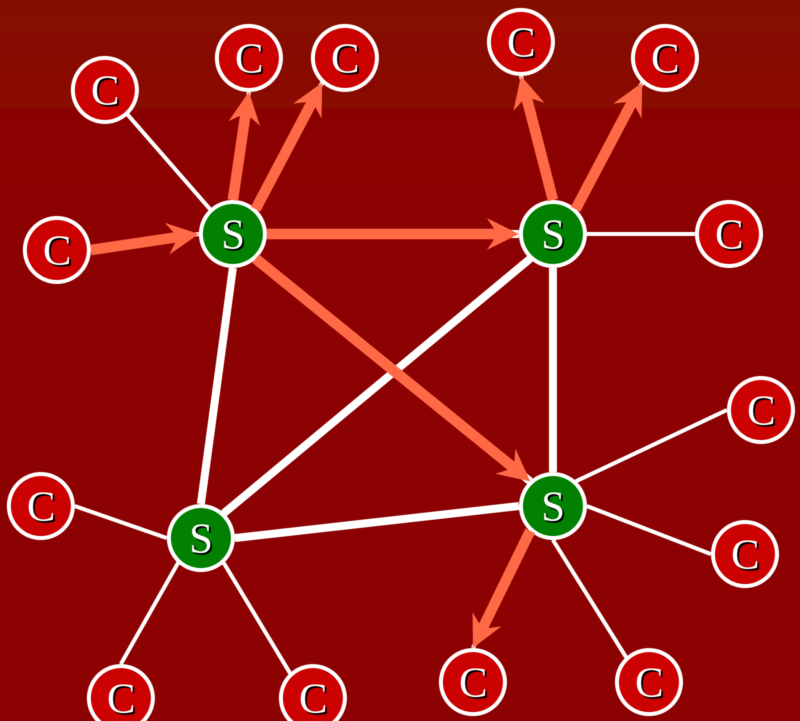


- Server may act as
  - broadcast reflector
  - filtering reflector
  - packet aggregation server
- Scalability problems
  - all traffic goes through the server
- Server-network architecture

Multiplayer server-network architecture

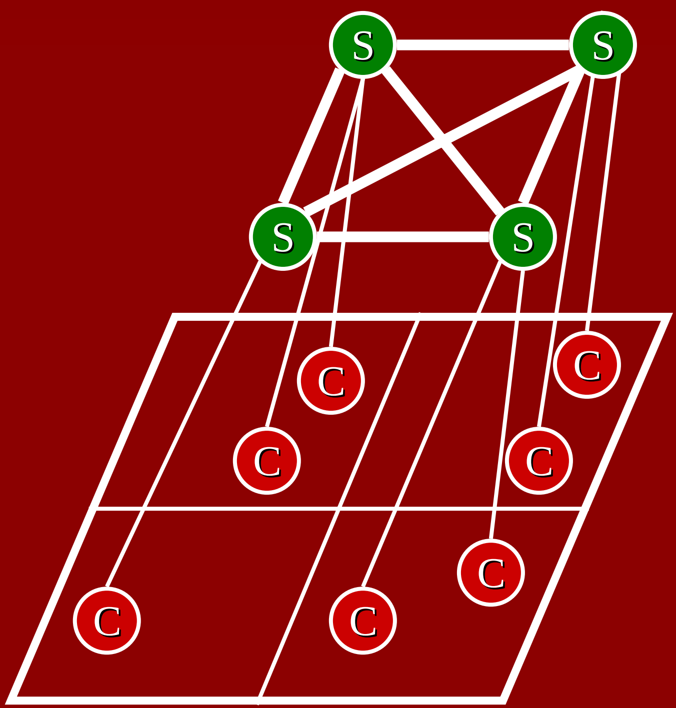
- Players can locate in the same place in the game world, but reside on different servers
  - real world ≠ game world
- Server-to-server connections transmit the world state information
  - WAN, LAN
- Each server serves a number of client players
  - LAN, modem, cable modem
- Scalability

Partitioning clients across multiple servers



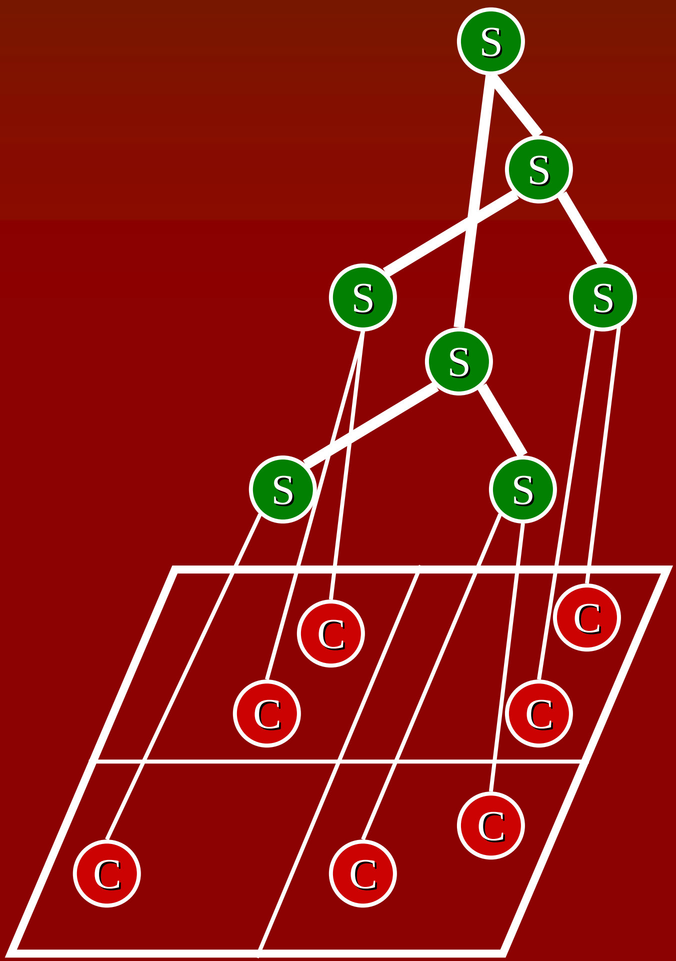
- The servers exchange control messages among themselves
  - inform the interests of their clients
- Reduces the workload on each server
- Incurs a greater latency
- The total processing and bandwidth requirements are greater

Partitioning the game world across multiple servers



- Each server manages clients located within a certain region
- Client communicates with different serves as it moves
- Possibility to aggregate messages
- Eliminates a lot of network traffic
- Requires advanced configuration
- Is a region visible from another region?

Server hierarchies



- Servers themselves act as clients
- Packet from an upstream server:
  - deliver to the interested downstream clients
- Packet from a downstream client:
  - deliver to the interested downstream clients
  - if other regions are interested in the packet then deliver it to the upstream server

Peer-to-peer architectures

- In the *ideal* large-scale networked game design, avoid having servers at all
  - eventually we cannot scale out
  - a finite number of players
- Design goal
  - peer-to-peer communication
  - scalable within resources
- Peer-to-peer: communication goes directly from the sending player to the receiving player (or a set of them)

Peer-to-peer with multicast

- For a scalable multiplayer game on a LAN, use multicast
- To utilize multicast, assign packets to proper multicast groups
- Area-of-interest management
  - assign outgoing packets to the right groups
  - receive incoming packets to the appropriate multicast groups
  - keep track of available groups
  - even out stream information

Peer-server systems

- Peer-to-peer: minimizes latency, consumes bandwidth
- Client–server: effective aggregation and filtering, increases latency
- Hybrid peer-server:
  - over short-haul, high-bandwidth links: peer-to-peer
  - over long-haul, low-bandwidth links: client-server
- Each entity has own multicast group
- Well-connected hosts subscribe directly to a multicast group (peer-to-peer)
- Poorly-connected hosts subscribe to a *forwarding server*
- Forwarding server subscribes to the entities' multicast groups
  - aggregation, filtering

Data and control architectures

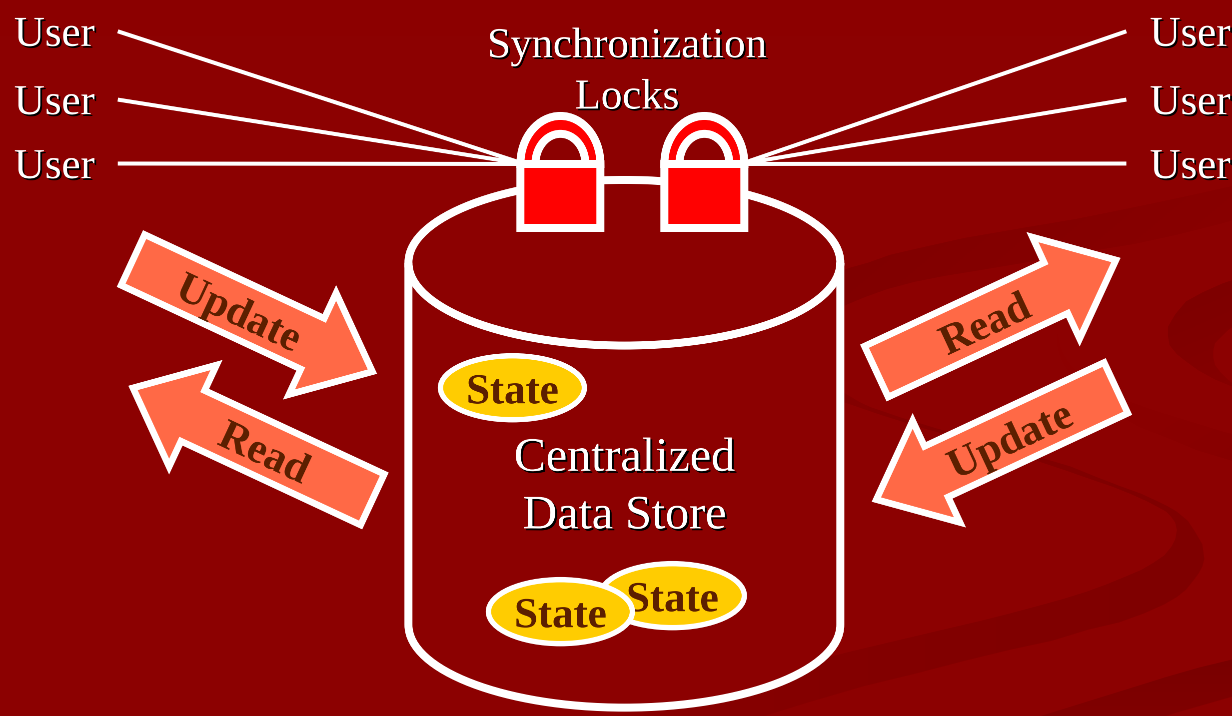
- Where does the data reside and how it can be updated?
  - Centralized
    - one node holds a full copy of the data
  - Replicated
    - all nodes hold a full copy of the data
  - Distributed
    - one node holds a partial copy of the data
    - all nodes combined hold a full copy of the data
- Consistency vs. responsiveness

Requirements for data and control architectures

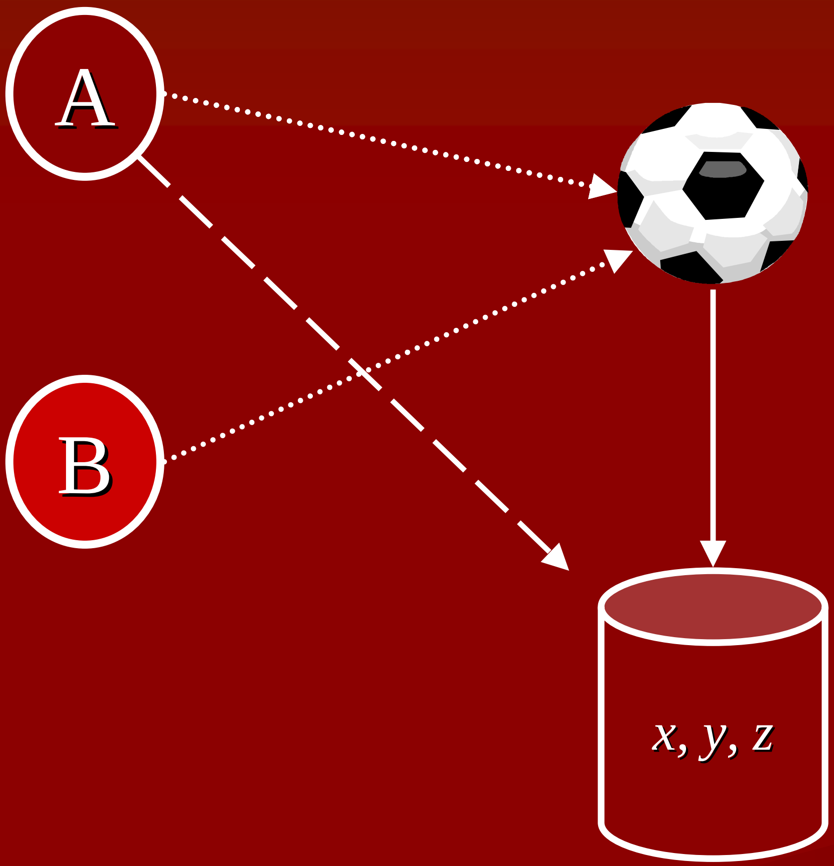
- Consistency: nodes should have the same view on the data
  - centralized: simple—one node binds them all!
  - replicated: hard—how to make sure that every replica gets updated?
  - distributed: quite simple—only one copy of the piece of data exists (but where?)
- Responsiveness: nodes should have a quick access to the data
  - centralized: hard—all updates must go through the centre node
  - replicated: simple—just do it!
  - distributed: quite simple—just do it (if data is in the local node) or send an update message (but to whom?)

Centralized architecture

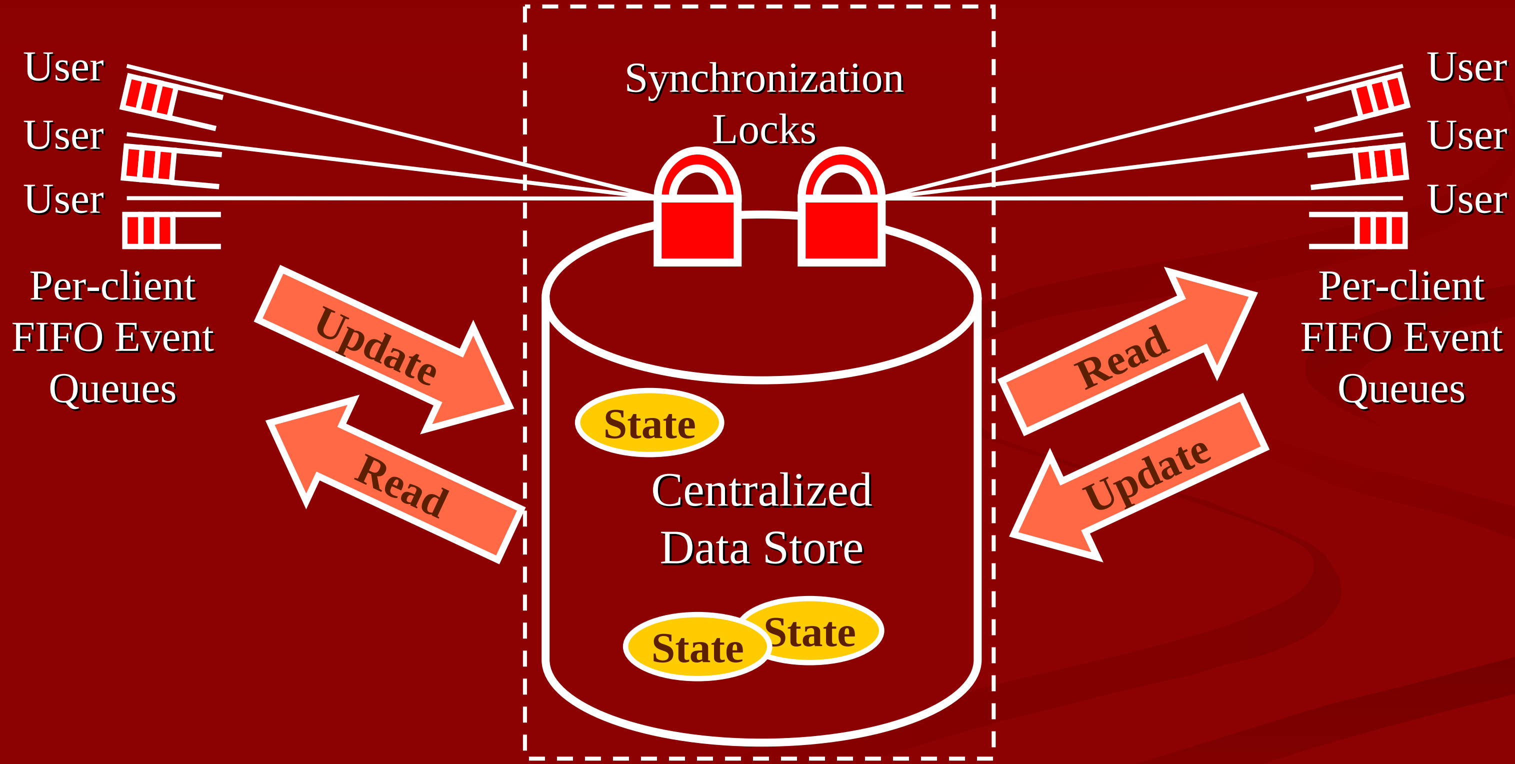
- Ensure that all nodes have identical information



Problem: Who's got the ball now?



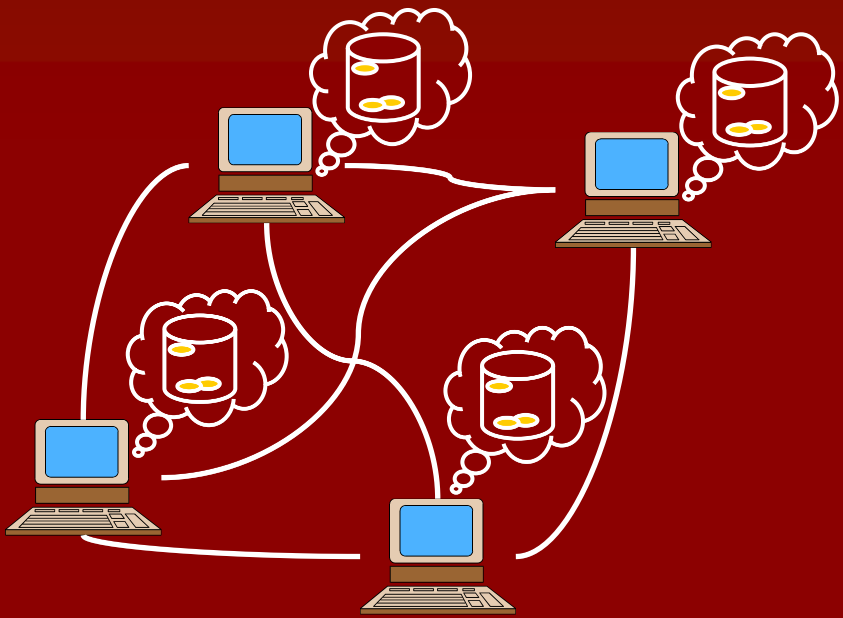
‘Eventual’ consistency



Pull and push

- The clients 'pull' information when they need it
  - make a request whenever data access is needed
  - problem: unnecessary delays, if the state data has not changed
- The server can 'push' the information to the clients whenever the state is updated
  - clients can maintain a local cache
  - problem: excessive traffic, if the clients are interested only a small subset of the overall data

Replicated architecture

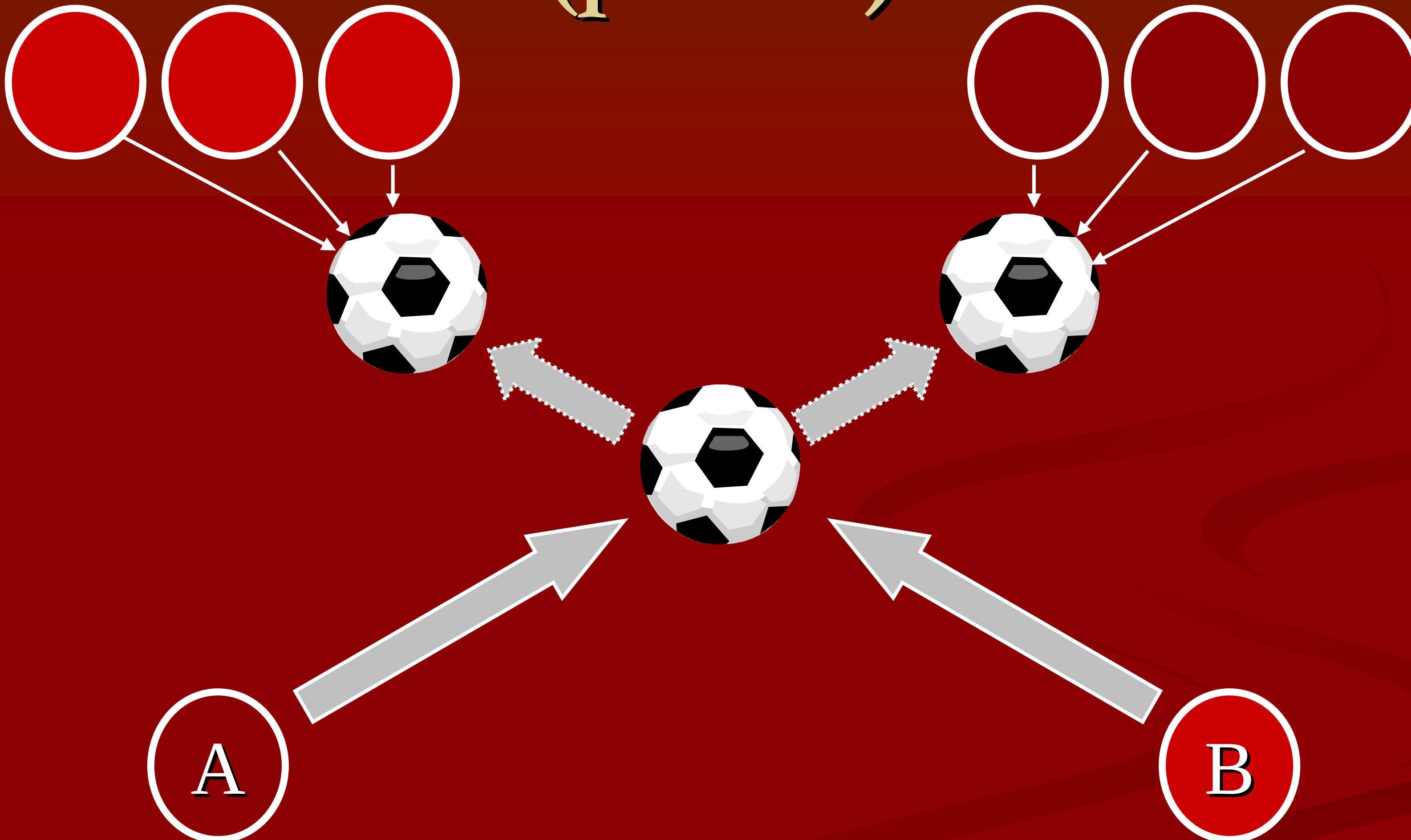


- Nodes exchange messages directly
  - ensure that all nodes receive updates
  - determine a common global ordering for updates
- No central host
- Every node has an identical view
- All state information is accessed from local node

Distributed architecture

- State information is distributed among the participating players
  - who gets what?
  - what to do when a new player joins the game?
  - what to do when an existing player leaves the game?
- ⇒ Entity ownership

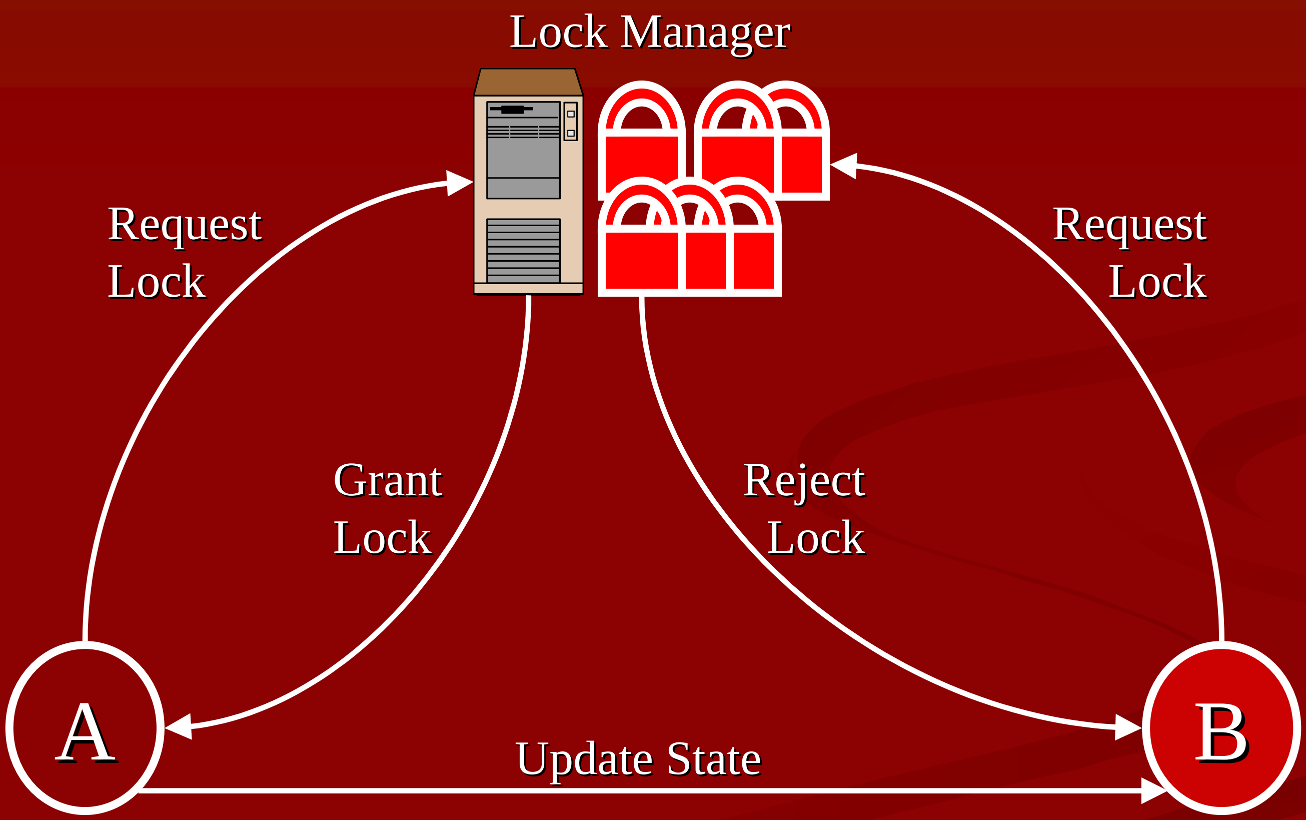
Problem: Who's got the ball now? (part II)



Entity ownership

- Ensure that a shared state can only be updated by one node at a time
  - exactly one node has the ownership of the state
  - the owner periodically broadcasts the value of the state
- Typically player's own representation (avatar) is owned by that player
- Locks on other entities are managed by a lock manager server
  - clients query to obtain ownership and contact to release it
  - the server ensures that each entity has only one owner
  - the server owns the entity if no one else does
  - failure recovery

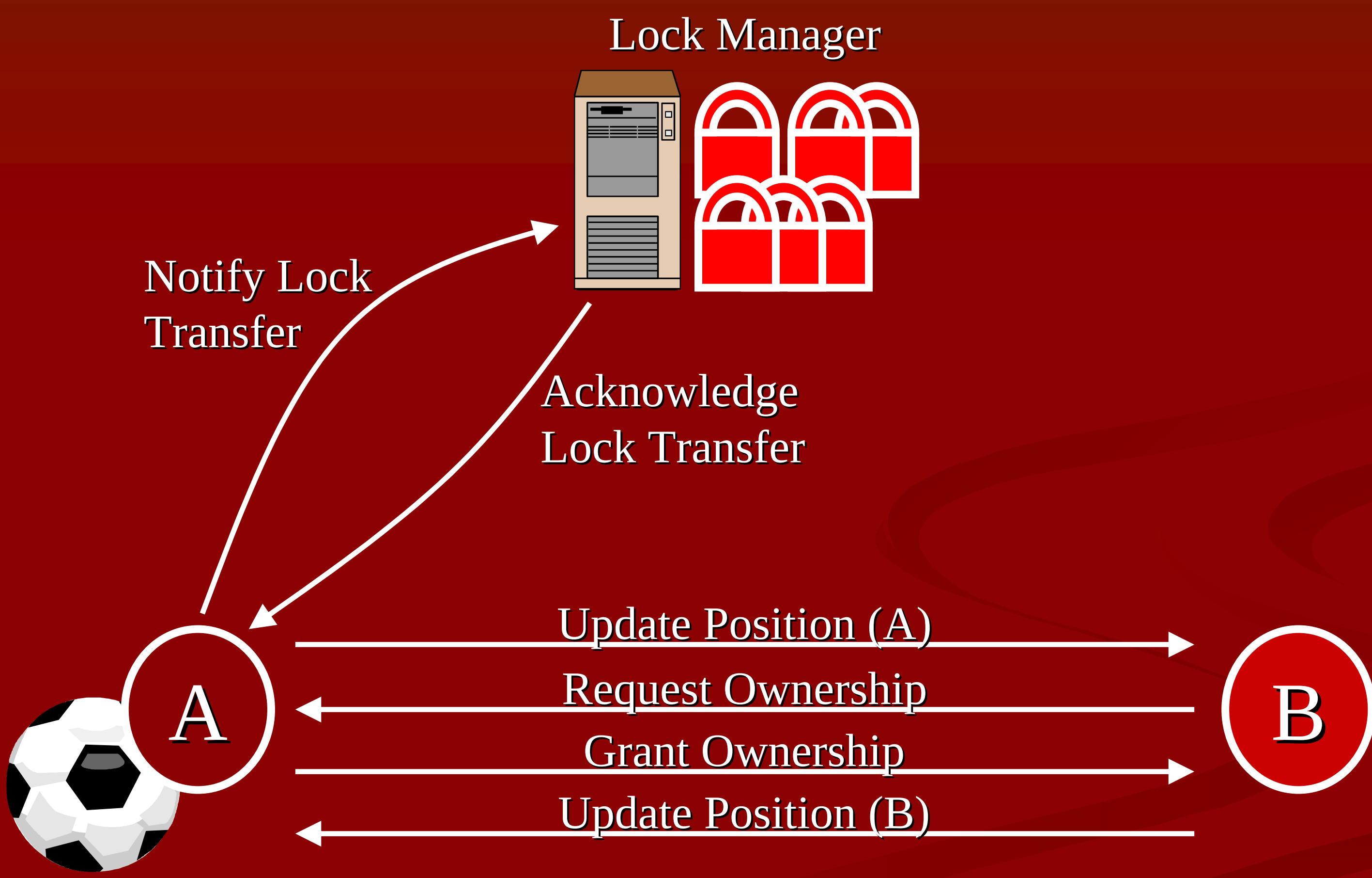
Lock manager: Example



Proxy update

- Non-owner sends an update request to the owner of the state
- The owner decides whether it accepts the update
- The owner serves as a proxy
- Generates an extra message on each non-owner update
- Suitable when non-owner updates are rare or many nodes want to update the state

Ownership transfer



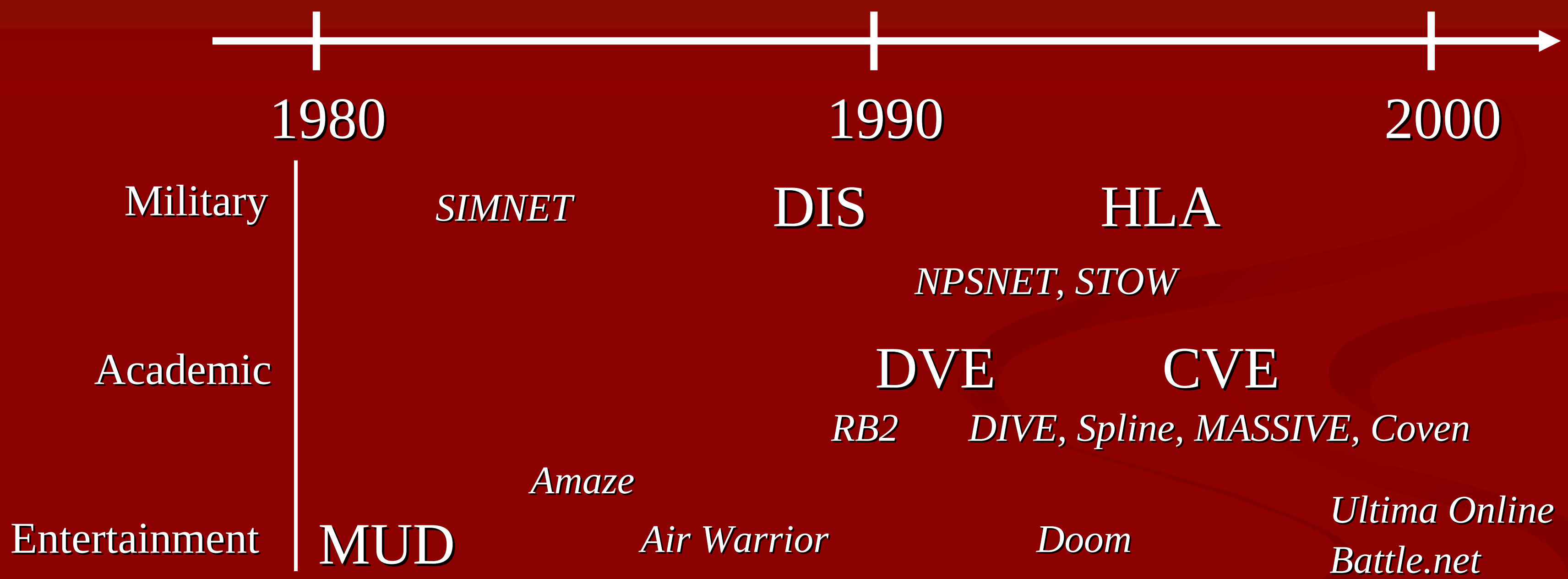
Ownership transfer (cont'd)

- The lock manager has the lock information at all times
- If the node fails, the lock manager defines the current lock ownership state
- Lock ownership transfer incurs extra message overhead
- Suitable when a single node is going to make a series of updates and there is little contention among nodes wishing to make updates

Networked application

- Department of Defense (DoD)
  - SIMNET
  - Distributed Interactive Simulation (DIS)
  - High-Level Architecture (HLA)
- Academic NVEs
  - PARADISE
  - DIVE
  - BrickNet
  - other academic projects
- Networked games and demos
  - SGI *Flight, Dogfight* and *Falcon A.T.*
  - *Doom*
  - other multiplayer games

History and evolution



U.S. Department of Defense (DoD)

- The largest developer of networked virtual environments (NVEs) for use as simulation systems
  - one of the first to develop NVEs with its SIMNET system
  - the first to do work on large-scale NVEs
- SIMNET (simulator networking)
  - begun 1983, delivered 1990
  - a distributed military virtual environment developed for DARPA (Defense Advanced Research Projects Agency)
  - develop a 'low-cost' NVE for training small units (tanks, helicopters,…) to fight as a team

SIMNET

- Technical challenges
  - how to fabricate high-quality, low-cost simulators
  - how to network them together to create a consistent battlefield
- Testbed
  - 11 sites with 50–100 simulators at each site
  - a simulator is the portal to the synthetic environment
  - participants can interact/play with others
  - play was unscripted free play
  - confined to the chain of command

SIMNET (cont'd)

- Basic components
  1. An object-event architecture
  2. A notion of autonomous simulator nodes
  3. An embedded set of predictive modelling algorithms (i.e., 'dead reckoning')

i. Object-event architecture

- Models the world as a collection of *objects*
  - vehicles and weapon systems that can interact
  - a single object is usually managed by a single host
  - ‘selective functional fidelity’
- Models interactions between objects as a collection of *events*
  - messages indicating a change in the world or object state
- The basic terrain and structures are separate from the collection of objects
  - if the structure can be destroyed then it has to be reclassified as an object, whose state is continually transmitted onto the network

ii. Autonomous simulator nodes

- Individual players, vehicles, and weapon systems on the network are *responsible* for transmitting *accurately* their current state
- Autonomous nodes do not interact with the recipients by any other way
- Recipients are responsible for receiving state change information and making appropriate changes to their local model of the world
- Lack of a central server
  - single point failures do not crash the whole simulation
  - players can join and leave at any time (persistency)
- Each node is responsible for one or more objects
  - the node has to send update packets to the network whenever its objects have changed enough to notify the other nodes of the change
  - a 'heartbeat' message, usually every 5 seconds

iii. Predictive modelling algorithms

- An embedded and well-defined set of predictive modelling algorithms called *dead reckoning*
- Average SIMNET packet rates:
  - 1 per second for slow-moving ground vehicles
  - 3 per second for air vehicles
- Other packets
  - fire: a weapon has been launced
  - indirect fire: a ballistic weapon has been launced
  - collision: a vehicle hits an object
  - impact: a weapon hits an object

Distributed Interactive Simulation (DIS)

- Derived from SIMNET
  - object-event architecture
  - autonomous distributed simulation nodes
  - predictive modelling algorithms
- Covers more simulation requirements
  - to allow any type of player, on any type of machine
  - to achieve larger simulations
- First version of the IEEE standard for DIS appeared 1993
- Protocol data unit (PDU)
  - determine when each vehicle (node) should issue a PDU
  - the DIS standard defines 27 different PDUs
  - only 4 of them interact with the environment: entity state, fire, detonation, and collision

Issuing PDUs

- The vehicle's node is responsible of issuing PDUs
  - entity state PDU
    - when position, orientation, velocity changes sufficiently (i.e., others cannot accurately predict the position any more)
    - as a heartbeat if the time threshold (5 seconds) is reached after the last entity state PDU
  - fire PDU
  - detonation PDU
    - a fired projectile explodes
    - node's vehicle has died (death self-determination)
  - collision PDU
    - vehicle has collided with something
    - detection is left up to the individual node

High-Level Architecture (HLA)

- Aims at providing a general architecture and services for distributed data exchange.
- While the DIS protocol is closely linked with the properties of *military* units and vehicles, HLA does not prescribe any specific implementation or technology.
  - could be used also with non-military applications (e.g., computer games)
  - targeted towards new simulation developments
- HLA was issued as IEEE Standard 1516 in 2000.

Academic research projects

- DoD's projects
  - large-scale virtual environments
  - most of the research is unavailable
  - lack-of-availability, lack-of-generality
- Academic community has reinvented, extended, and documented what DoD has done
  - PARADISE
  - DIVE
  - BrickNet
  - and many more…

PARADISE

- Performance Architecture for Advanced Distributed Interactive Simulations Environments (PARADISE)
- Initiated in 1993 at Stanford University
- A design for a network architecture for thousands of users
- Assign a different multicast address to each active object
- Object updates similar to SIMNET and DIS
- A hierarchy of area-of-interest servers
  - monitor the positions of objects
  - which multicast addresses are relevant

DIVE

- Distributed Interactive Virtual Environment (DIVE)
- Swedish Institute of Computer Science
- To solve problems of collaboration and interaction
- Simulate a large shared memory over a network
- Distributed, fully replicated database
- Entire database is dynamic
  - add new objects
  - modify the existing databases
  - reliability and consistency

BrickNet

- National University of Singapore, started in 1991
- Support for graphical, behavioural, and network modelling of virtual worlds
- Allows objects to be shared by multiple virtual worlds
- No replicated database
- The virtual world is partitioned among the various clients

Other academic projects

- MASSIVE
  - different interaction media: graphics, audio and text
  - awareness-based filtering: each entity expresses a focus and nimbus for each medium
- Distributed Worlds Transfer and Communication Protocol (DWTP)
  - each object can specify whether a particular event requires a reliable distribution and what is the event's maximum update frequency
- Real-Time Transport Protocol (RTP/I)
  - ensures that all application instances look as if all operations have been executed in the same order
- Synchronous Collaboration Transport Protocol (SCTP)
  - collaboration on closely coupled, highly synchronized tasks
  - the interaction stream has critical messages (especially the last one) which are sent reliably, while the rest are sent by best effort transport

Networked demos and games

- SGI *Flight*
  - 3D aeroplane simulator demo for Silicon Graphics workstation, 1983–84
    - serial cable between two workstations
    - Ethernet network
    - users could see each other's planes, but no interaction
- SGI *Dogfight*
  - modification of *Flight*, 1985
  - interaction by shooting
  - packets were transmitted at frame rate → clogged the network
  - limited up to ten players
- *Falcon A.T.*
  - commercial game by Spectrum Holobyte, 1988
  - dogfighting between two players using a modem

Networked games: *Doom*

- id Software, 1993
- First-person shooter (FPS) for PCs
- Part of the game was released as shareware in 1993
  - extremely popular
  - created a gamut of variants
- Flooded LANs with packets at frame rate

Networked games: ‘First generation’

- Peer-to-peer architectures
  - each participating computer is an equal to every other
  - inputs and outputs are synchronized
  - each computer executes the same code on the same set of data
- Advantages:
  - determinism ensures that each player has the same virtual environment
  - relatively simple to implement
- Problems:
  - persistency: players cannot join and leave the game at will
  - scalability: network traffic explodes with more players
  - reliability: coping with communication failures
  - security: too easy to cheat

Networked games: 'Second generation'

- Client–server architectures
  - one computer (a server) keeps the game state and makes decisions on updates
  - clients convey players' input and display the appropriate output but do not inlude (much) game logic
- Advantages:
  - generates less network traffic
  - supports more players
  - allows persistent virtual worlds
- Problems:
  - responsiveness: what if the connection to the server is slow or the server gets overburdened?
  - security: server authority abuse, client authority abuse

Networked games: ‘Third generation’

- Client–server architecture with prediction algorithms
  - clients use dead reckoning
- Advantages:
  - reduces the network traffic further
  - copes with higher latencies and packet delivery failures
- Problems:
  - consistency: if there is no unequivocal game state, how to solve conflicts as they arise?
  - security: packet interception, look-ahead cheating

Networked games: 'Fourth generation'

- Generalized client–server architecture
  - the game state is stored in a server
  - clients maintain a subset of the game state locally to reduce communication
- Advantages:
  - traffic between the server and the clients is reduced
  - clients can response more promptly
- Problems:
  - boundaries: what data is kept locally in the client?
  - updating: does the subset of game state change over time?
  - consistency: how to solve conflicts as they occur?

Future trends? Part 1: Massive multiplayer online games

Name	Publisher	Released	Subscribers
Ultima Online	Origin Systems	1997	250,000
EverQuest	Sony Entertainment	1999	430,000
Asheron's Call	Microsoft	1999	N/A
Dark Age of Camelot	Sierra Studios	2001	250,000
Sims Online	Electronic Arts	2002	97,000
Star Wars Galaxies	LucasArts	2003	N/A

source: http://www.mmorpg.com

Future trends? Part 2: Location-based games

- *ARQuake*, School of Computer and Information Science, University of South Australia
- augmented reality version of *Quake*: walk around in the real world and play *Quake* against virtual monsters
- components
  - head mounted display
  - mobile computer
  - head tracker
  - GPS system