Networked Virtual Environments
Origins & Architectures

Patrice Torguet

Paul Sabatier University / IRIT / VORTEX

Schedule

- Introduction
- Applications
- Architectures for Networked Virtual Environments
  - Distributed architectures
  - Dynamic shared state management
  - Interest management techniques
- Networked games specifics
  - FPS
  - RTS
  - MMORPG

Introduction

- Definitions
  - Virtual Reality (VR) : **techniques** and **tools** used in order to give someone the **illusion** that s/he is really immersed in a **synthetic world**

Introduction

- Definitions
  - Virtual Reality (VR) : **techniques** and **tools** used in order to give someone the **illusion** that s/he is really immersed in a **synthetic world**
  - Networked Virtual Environments (NVEs) : **several** people **share** the same virtual world thanks to a computer network

Introduction

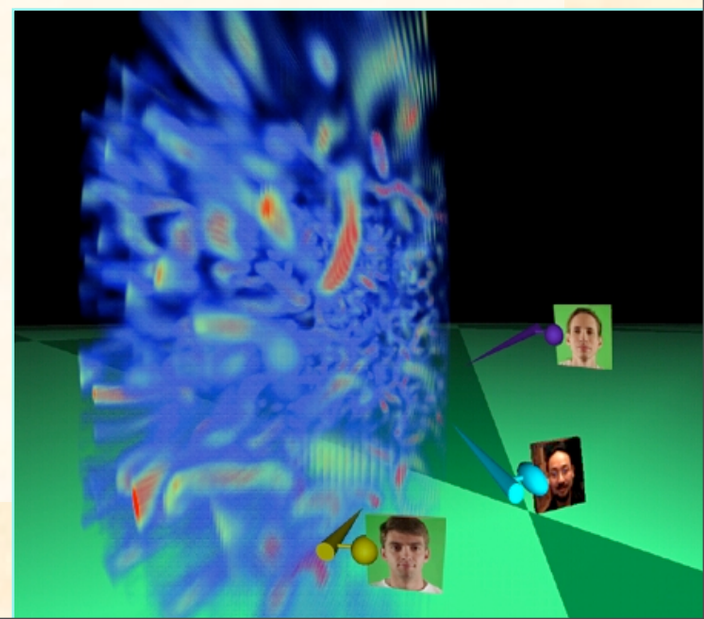
- Synonyms :
  - Networked Virtual Environments (NVEs)
  - Networked Virtual Worlds (NVWs)
  - Distributed Virtual Reality (DVR)
  - Collaborative Virtual Environments (CVEs)
  - Collaborative Virtual Worlds (CVWs)
  - Shared Virtual Worlds (SVWs)
  - Massively Multiplayer Online Games (MMOGs)

Entertainment Applications

- Multiplayer Games
  - First Person Shooter (FPS)
    - ✓Doom, Quake, Unreal Tournament...
  - Real Time Strategy (RTS)
    - ✓Warcraft, Starcraft, Age of Empires...
  - Computer Role Playing Games (CRPG)
    - ✓Neverwinter Nights, Final Fantasy...
  - Sport simulations
    - ✓Pro Evolution Soccer, Need for Speed...
  - MMOGs
    - ✓Ultima Online, Everquest, World of Warcraft...
  - ...

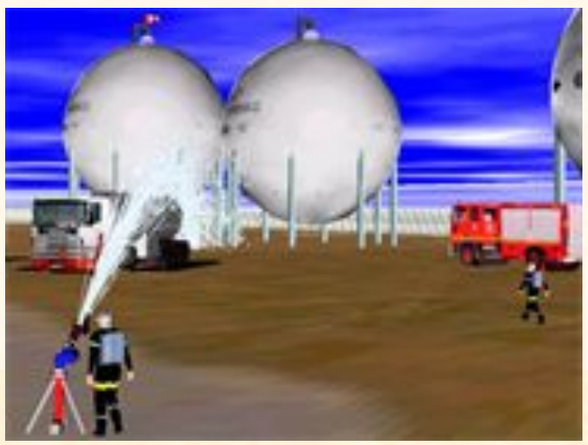
Serious Applications

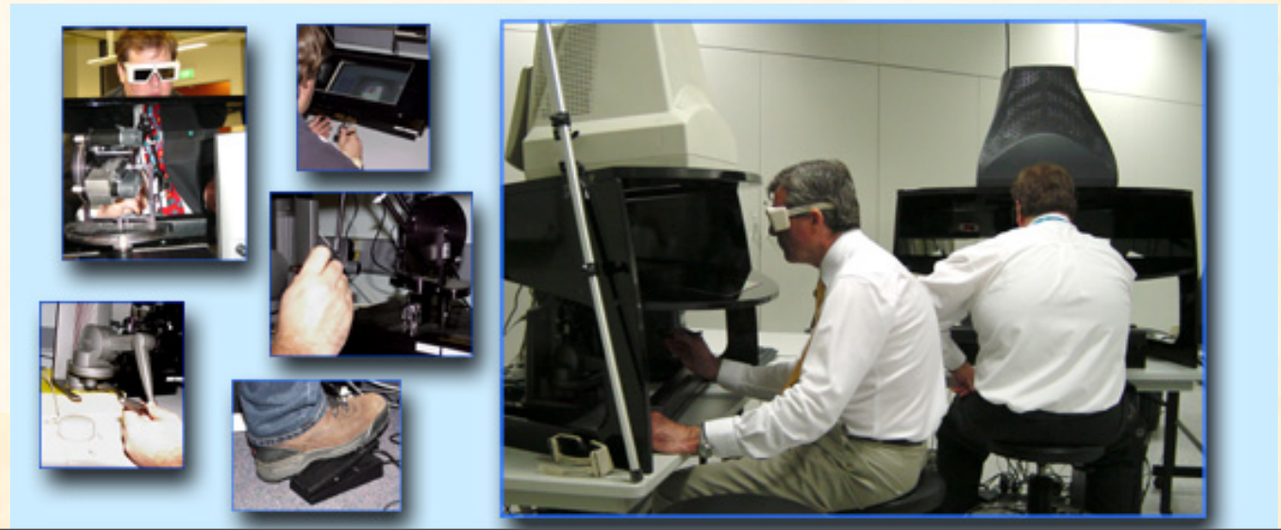
- Digital Mockups (DMUs)
  - Allow several engineers to work on the same digital mockup
  - Examples :
    - ✓CSA (Common Support Aircraft) for the US DoD
    - ✓Several DMUs were used during Airbus A380 design
- Scientific Visualization
  - Collaborative work on scientific data
  - Examples :
    - ✓Cooperative visualization of proteins and other molecules (Buffalo University)
    - ✓Oil Tanks Collaborative Visualisation (Northern Arizona University)
    - ✓Volumetric data visualization (University of Illinois at Chicago)



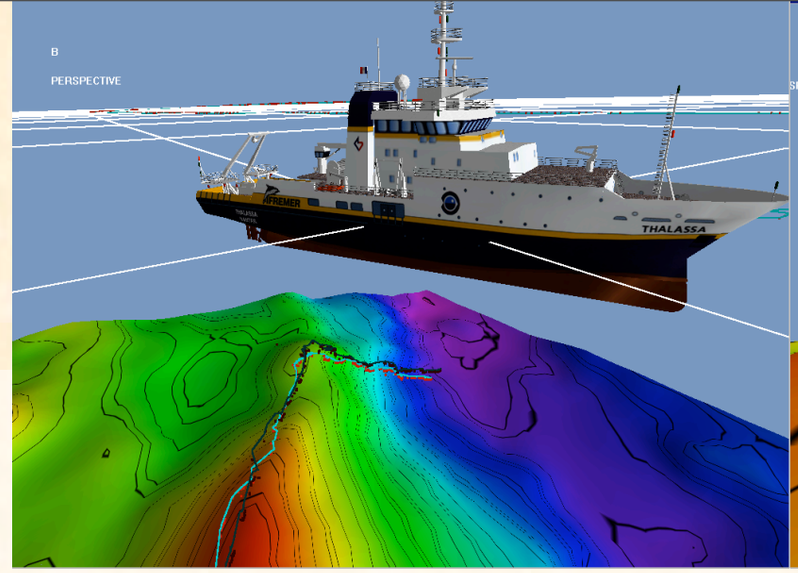
Serious Applications

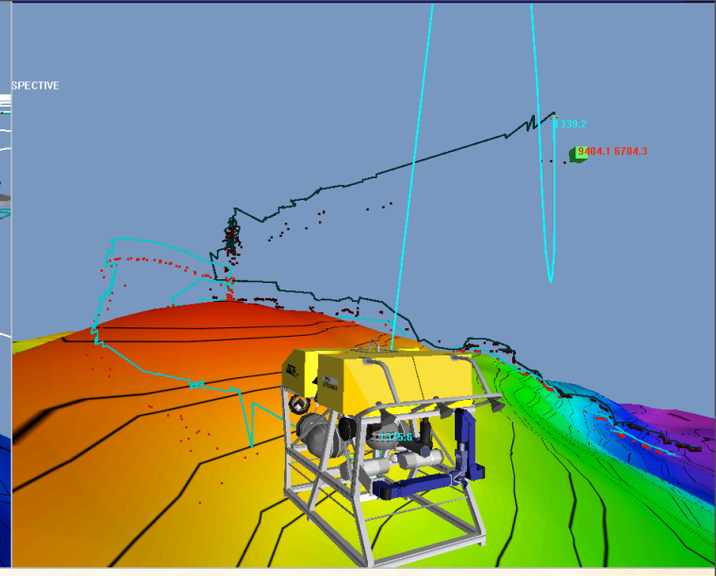
- Military Simulation
  - Military Training
  - Examples :
    - ✓Networked flight simulators (Aéronavale)
    - ✓Close Combat Tactical Trainer (CCTT)
- Civilian Simulation
  - Civilian Training
  - Examples :
    - ✓Firefighter training (ENSTB)
    - ✓Surgeons training (ICT Australia)



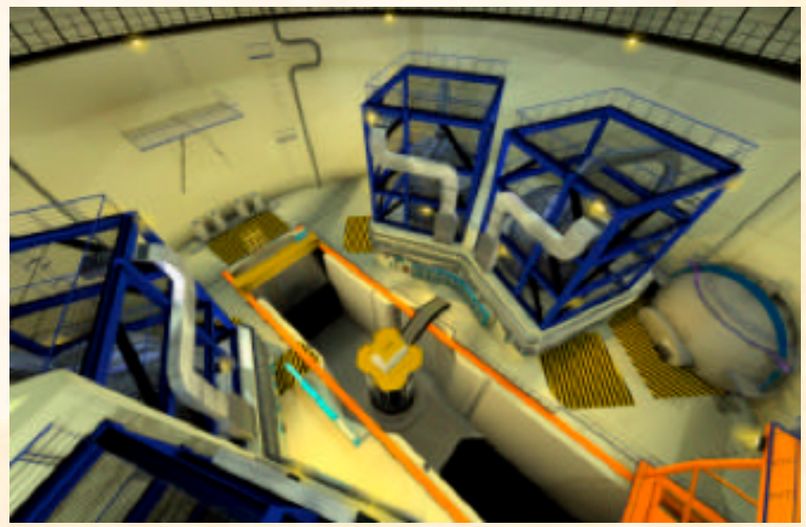


Serious Applications





- Teleoperation
  - Robot remote control in remote and/or dangerous environments
  - Examples :
    - ✓Submarine Remotely Operated Vehicle (ROV) (Ifremer)
    - ✓Remote control of a cleaning robot in a nuclear plant (EDF)
    - ✓Semi-automatic remote control of a robot on Mars (NASA)
- Telepresence
  - Bring the expertise of a human being in a distant place
  - Example :
    - ✓Medical expertise for surgery or emergency situations (British Telecom)



Network Programming (Overview)

- Network problems
  - Latency
    - ✓ Time spent when sending one bit of data from one place to another place (delay, lag, ping …)
    - ✓ A big latency induces a low interactivity level for the VE
    - ✓ Latency is due to:
      - light-speed (about 8.25 ms per time zone)
      - sending and receiving (depends on computer speed)
      - network added delay (commutation)
  - Throughput/Bandwidth
    - ✓ Number of bits that can be transmitted by the network in 1 second
    - ✓ Depends on cables and network equipments
  - Reliability
    - ✓ A network can drop packets (congestion)
    - ✓ Data can be modified (transmission errors)
    - ✓ If we need reliability we need to have acknowledgement messages

Network Programming (Overview)

- Application level protocol
  - Application layer
    - ✓(other protocols exist for transport, routing…)
  - Describe a set of rules that 2 applications must follow in order to communicate correctly
  - 3 Components
    - ✓ Message Format
      - Data contained in a message
      - How to extract these data ?
    - ✓ Semantic of messages
      - When do we need to send a message ?
      - What must we do when we receive a message ?
    - ✓ Error management
      - What must the sender and/or the receiver do when they detect an error ?

Network Programming (Overview)

- API: most used is BSD Sockets on TCP/IP (Internet)
- Inter-process communication similar to file input/output
  - Sending data to a process ~ writing data in a file
  - Wait/receive data from a process ~ reading data from a file
- Only difference: Connection or destination parameters
  - We must choose the destination computer: IP address or name (DNS)
  - We must choose the destination application: Port number
  - We must choose a transport protocol:
    - ✓Reliable (no loss of data and ordered send/receive) but « slower » (TCP)
    - ✓Not reliable but « faster » (UDP)
    - ✓broadcast or multicast UDP
- Other big difference: asynchronous (function calls may block)
  - Receiving is a blocking operation
    - ✓=> You can use Threads (e.g. with Java)
    - ✓=> You can use select (you can wait on several channels for a specified amount of time)
  - Sending may block with TCP (but not often)

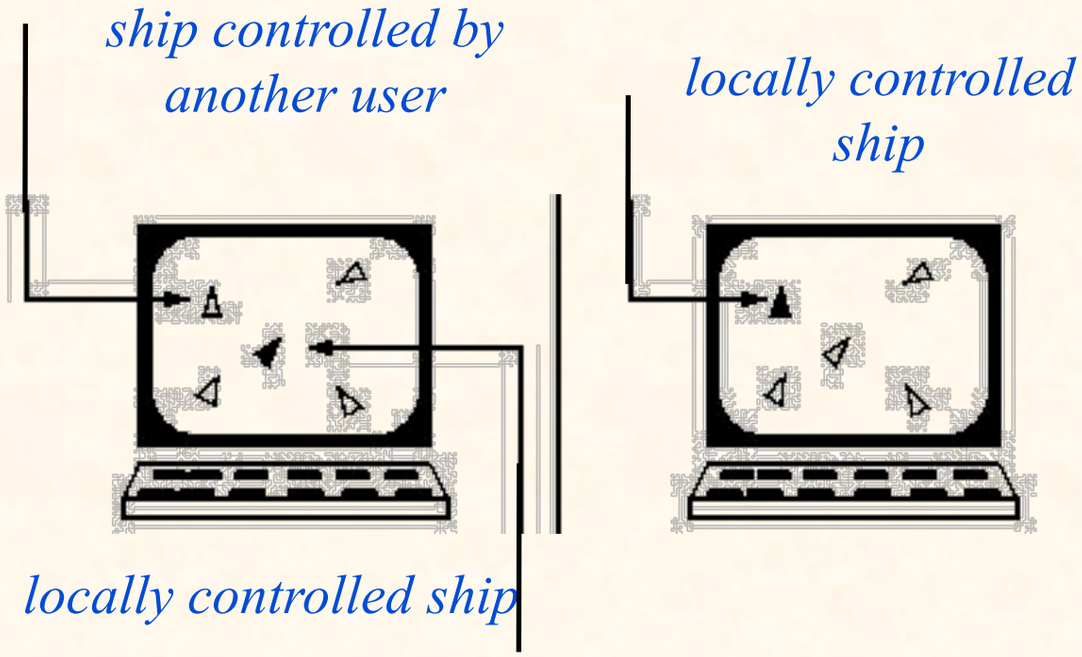
Let's start with a simple example

- 2D example
- Each user controls a spaceship (a triangle !)
- Distant ships are shown on each application

- Written in Java for simplicity

- MUVE (multi user virtual environment)

- Downloadable from my website:



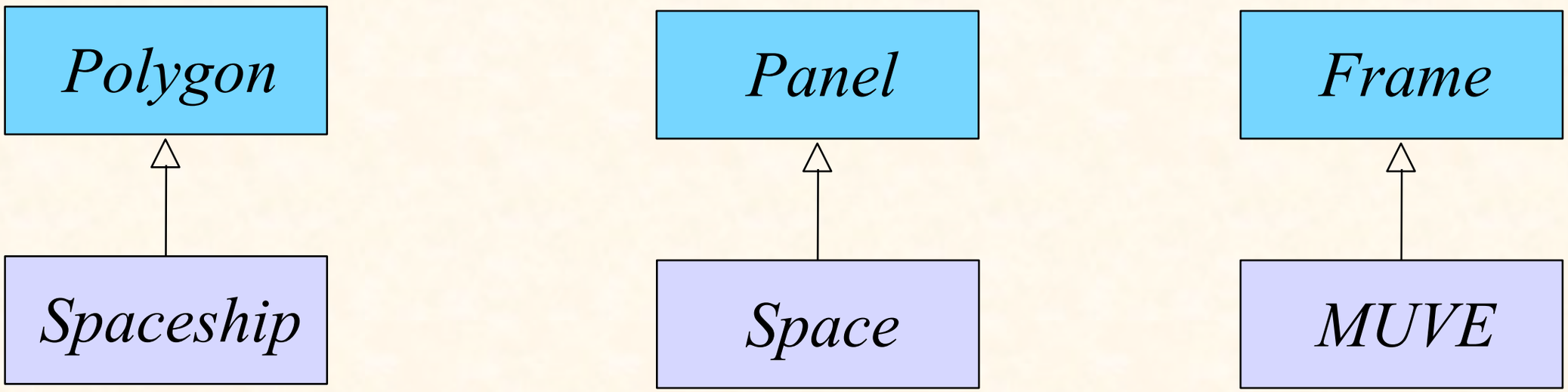
http://torguet.net/cours

Let's start with a simple example

- Coded with BSD Sockets using UDP multicast
  - Message Format
    - ✓ship unique identifier
    - ✓position and orientation of the ship
  - Semantic of messages
    - ✓Each time the local ship moves its application sends a message to every other application
    - ✓When the first message describing a ship is received a local triangle is created and displayed
    - ✓For the following messages we just update the position and orientation of the triangle
  - No error management

Let's start with a simple example

- The classes



Attributes :
id
x, y, velocity, currentAngle
oldX, oldY, oldAngle

Methods :
Spaceship(id)
translate(x,y)
rotate(angle)
update(time)
sendNetworkUpdate()

Attributes :
local_ship, spaceships
lastTime
socket, internetAddress (ia)

Methods :
Space(id)
connectToNetwork()
updateLocalShip()
paintComponent()
run()

Attributes :
space

Methods :
MUVE(id)
run()
initComponents()
myKeyPressed()
exitForm()
main()

Let's start with a simple example

- Spaceship class
  - void sendNetworkUpdate()

```
// if nothing changed we don't send (Semantic of packets)
    if ((oldAngle == currentAngle) && (oldX == x) && (oldY == y))
      return;
// we update old values
   oldAngle = currentAngle; oldX = x; oldY = y;

// we place data to send in a buffer (Packet Format)
    ByteArrayOutputStream stream = new ByteArrayOutputStream(40);
    DataOutputStream      dos    = new DataOutputStream(stream);
    dos.writeInt(id); dos.writeInt(x); dos.writeInt(y); dos.writeDouble(currentAngle);
// we create a datagram
    DatagramPacket dp = new DatagramPacket(stream.toByteArray(),
                        stream.toByteArray().length, Space.ia, 2000);
// we send the datagram
    Space.socket.send(dp);
```

Let's start with a simple example

- Space class
  - void connectToNetwork()

```
// we use multicast
// we create a socket using a chosen port
        socket = new MulticastSocket(2000);

// we join a multicast group
        ia = InetAddress.getByName("224.11.4.2");
        socket.joinGroup(ia);

// we create a thread for network receives
        Thread networkListener = new Thread(this);
        networkListener.start();
```

Let's start with a simple example

- Space class
  - void run()

// **this buffer will receive network messages data**

```
byte buffer[] = new byte[256];
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);


while (true) {
   // we wait for a datagram
      socket.receive(dp);
   // we extract data from the datagram (Packet Format)
      ByteArrayInputStream  stream = new ByteArrayInputStream(buffer);
      DataInputStream       dis    = new DataInputStream(stream);
   // we first extract the id
      int id = dis.readInt(); Integer theId = new Integer(id);
```

Let's start with a simple example

- Space class
  - void run() (continued)

```
// (Semantic of packets)
// we search for the spaceship in the list (a hashtable) of ships
      Spaceship sp = (Spaceship) spaceships.get(theId);

// if the ship isn't found and is not the local ship we create it
      if ((sp == null) && (id != local_ship.id)) {
                  sp = new Spaceship(id);
                  spaceships.put(theId, sp);
      }

 // if we have a ship
      if (sp != null) {
        // we update its position and orientation (Packet Format)
        sp.x = dis.readInt();
        sp.y = dis.readInt();
        sp.currentAngle = dis.readDouble();
        // rotate will draw the ship
        sp.rotate(0);
      }
} // end while
```

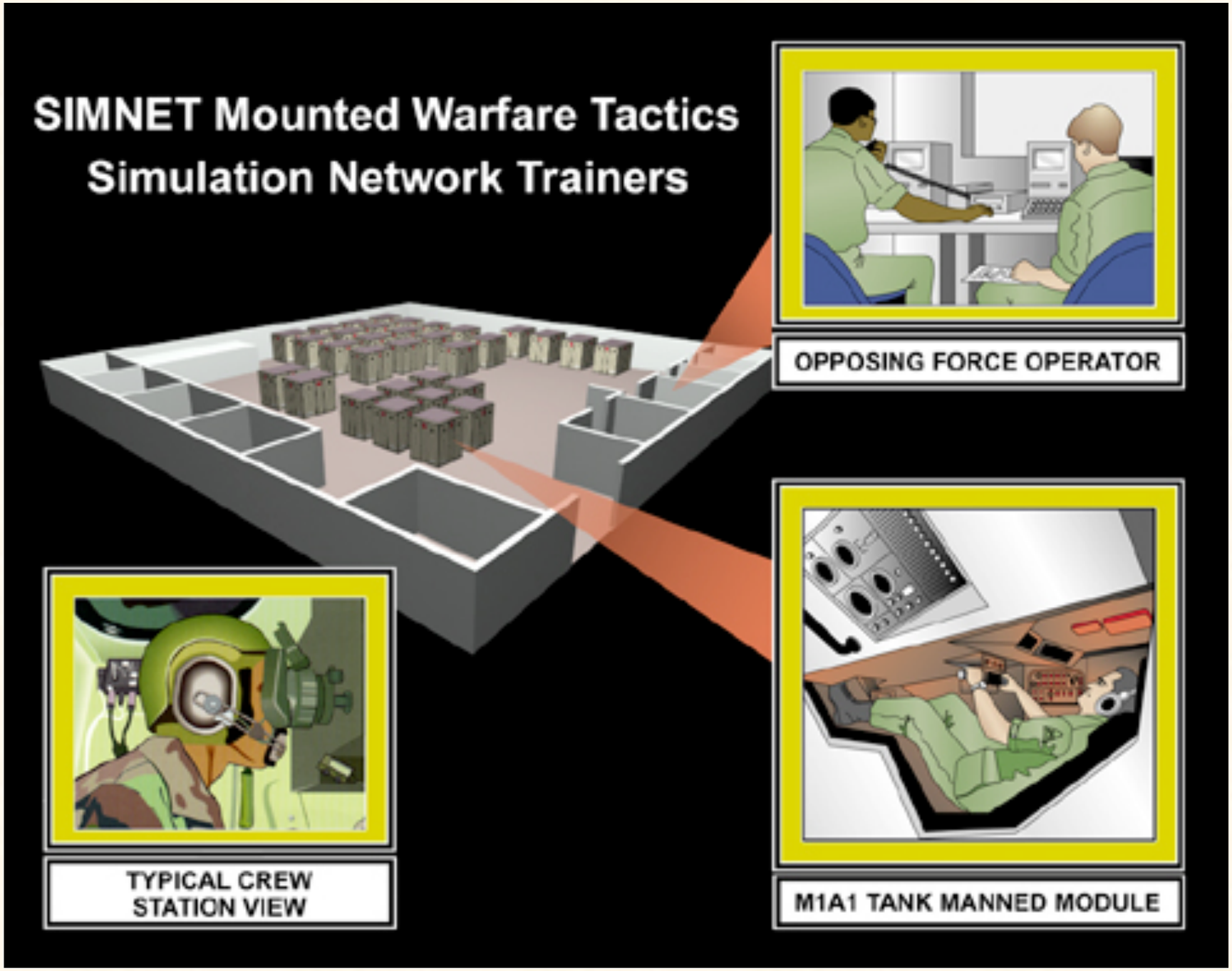
Networked VEs (short) history

- Military applications
  - SIMNET
  - DIS
- Networked games
  - MUD1 and MUDs/MOOs/MUSHs
  - SGI Flight & Dogfight
  - Lucasfilm Habitat
  - Doom
  - Warcraft
  - Ultima Online
  - WOW
  - EVE Online

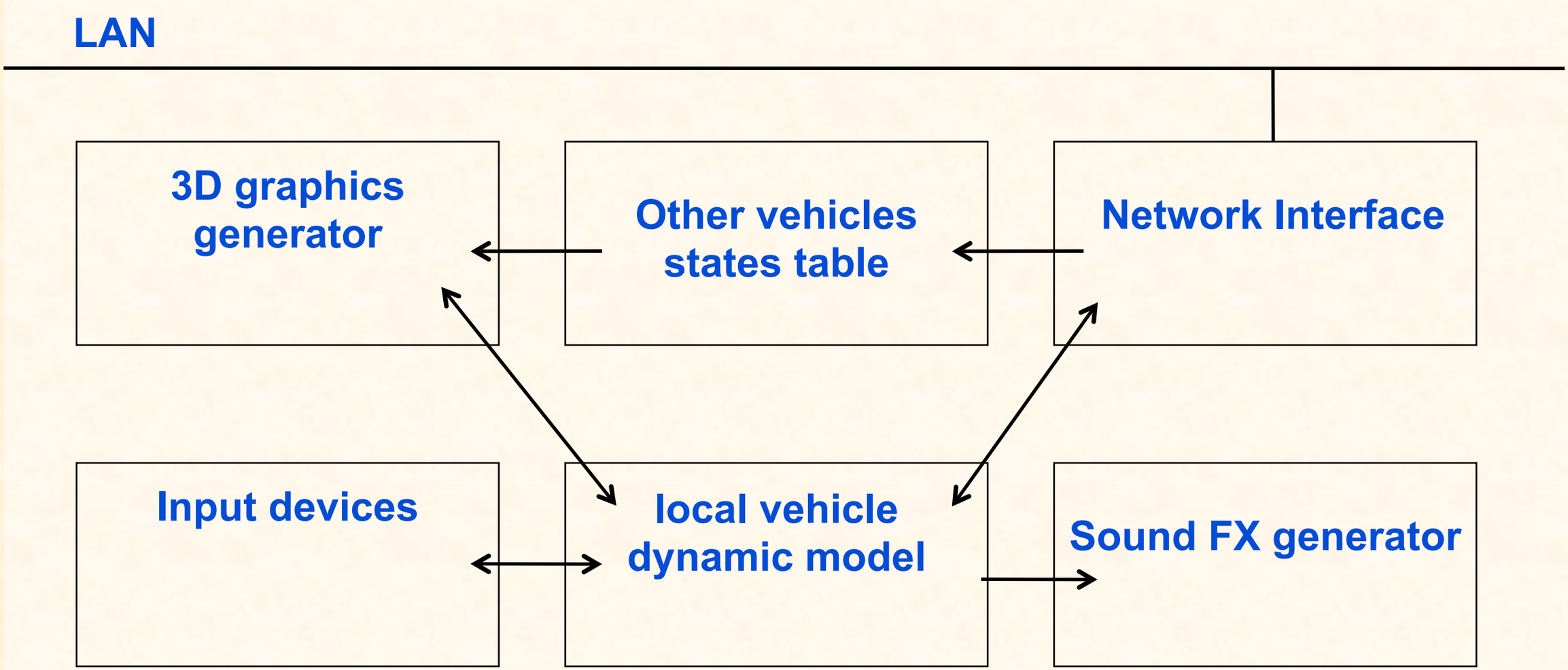
SIMNET (simulator networking)

- SIMNET is a networked VE developed for DARPA by BBN (Bolt, Beranek and Newman), Perceptronics and Delta Graphics.
- Specifications started in 1983 first operational version delivered at end of march 1990.
- Specifications
  - Simulate from 100 to 100 000 entities (hardware simulators).
  - Multi-sites (geographically distributed).
  - Heterogenous Simulations (several different simulators).
  - Low cost compared to the the cost of real exercises.
  - Totally distributed (no central point of failure).
  - Realtime.

SIMNET (simulator networking)



SIMNET Architecture



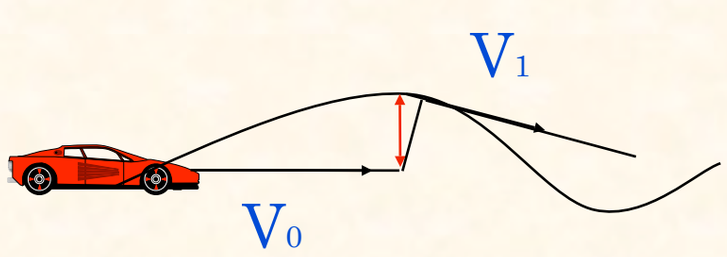
SIMNET (simulator networking)

- The software architecture is based on 3 components:
  - An objects/events architecture,
  - Simulation nodes are autonomous,
  - “dead reckoning” predictive algorithms
- Object/events architecture models the world with a collection of objects whose interactions are modeled by a collection of events
  - Objects are vehicles and weapon systems
  - Events are messages sent through the network indicating changes to the world or to objects

SIMNET (simulator networking)

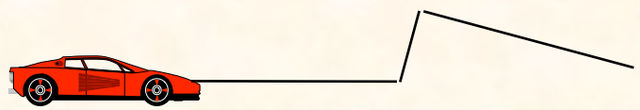
➢« Dead-Reckoning »

Local Site



Real model
+ extrapolated model

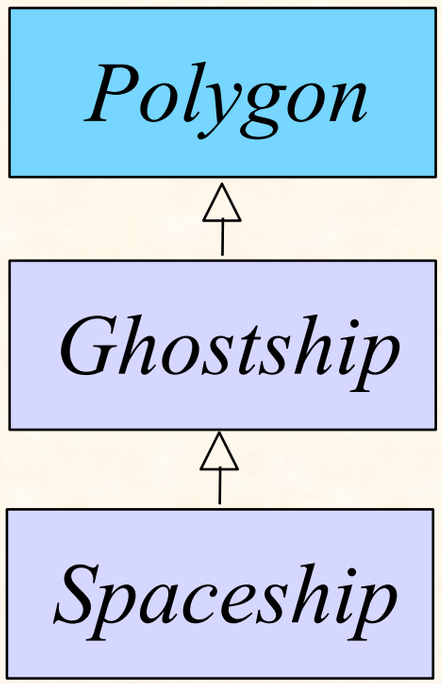
Distant Site



Extrapolated model

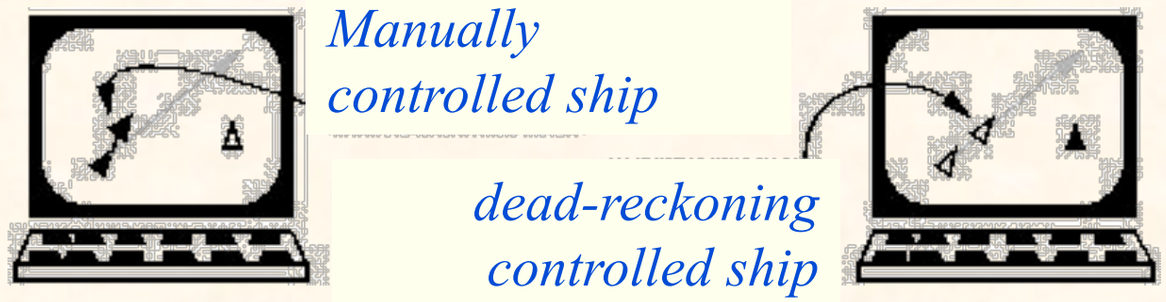
Back to the example

- Let's add dead-reckoning



Attributes :
id, velocity,
x, y, currentAngle

Methods :
Ghostship(id)
translate(x,y)
rotate(angle)
update(time)



Attributes :
ghost, maxDistance2

Methods :
Spaceship(id)
update*(time)*
sendNetworkUpdate*()*

Back to the example

- Spaceship class (revisited)
  - void update(double time)

```
// move the ship
…
// move its ghost
ghost.update(time);
```

  - void sendNetworkUpdate()

```
// we compare the distance between the ship and its ghost to the threshold
    if ((x - ghost.x)*(x - ghost.x) + (y - ghost.y)*(y - ghost.y) < maxDistance2)
      return;
// update the ghost
     ghost.x = x; ghost.y = y; ghost.currentAngle = currentAngle; ghost.velocity = velocity;
// data to send is placed in a buffer
    ByteArrayOutputStream stream = new ByteArrayOutputStream(40);
    DataOutputStream      dos    = new DataOutputStream(stream);
    dos.writeInt(id); dos.writeInt(x); dos.writeInt(y);
    dos.writeDouble(currentAngle); dos.writeInt(velocity);
// a datagram is created
    DatagramPacket dp = new DatagramPacket(stream.toByteArray(),
                            stream.toByteArray().length, Space.ia, 2000);
// and sent through the network
    Space.socket.send(dp);
```

Back to the example

- Space class (revisited)
  - synchronized void paintComponent(Graphics g)

```
// display local ship
    …
// update (with DR algorithms) and then display distant ships
    Enumeration ships = ghostships.elements();
    while(ships.hasMoreElements()) {
      Ghostship sp = (Ghostship)(ships.nextElement());
      sp.update(time - lastTime);
      g.drawPolygon(sp);
    }
```

  - void run()

```
…
 // we search the ship in the list of ghost ships
       Ghostship sp = (Ghostship) ghostships.get(theId);
 // if the ship isn't found and is not local then we create a ghost ship
       …
 // if we have a ship
       if (sp != null) {
         // we update it
         sp.x = dis.readInt(); sp.y = dis.readInt();
         sp.currentAngle = dis.readDouble(); sp.velocity = dis. readDouble();
         ...
```

SIMNET (simulator networking)

- SIMNET and scalability
  - The SIMNET network software architecture proved scalable with an exercise in March of 1990 having some 850 objects at five sites, with most of those objects being semi-automated forces.
  - Objects in that test averaged one packet per second, with each packet being some 156 bytes in size for a peak requirement of 1.06 Mbits/second, just under the T-1 speed of the connecting links.

DIS (Distributed Interactive Simulation)

- DIS is SIMNET successor but it is more generic
- It has the same 3 components :
  - object/event architecture
  - autonomous simulation nodes
  - « dead reckoning » algorithms
- The core of the DIS network software architecture is the protocol data unit (PDU).
- Determining when each vehicle (node) of the simulation should issue a PDU is the key to this architecture.

DIS (Distributed Interactive Simulation)

- The DIS (IEEE 1278) standard defines 27 different PDUs, only four of which (Entity State, Fire, Detonation, and Collision) are used by nodes to interact with the virtual environment.
- In fact, most DIS-compliant simulations only implement those four PDUs, either throwing away the other 23 PDUs without comment or issuing a brief error message indicating a non-supported PDU was received.

DIS (Distributed Interactive Simulation)

- A demonstration at the 1993 Interservice/Industry Training and Education Conference (I/ITSEC) showed that Entity State PDUs comprised 96% of the total DIS traffic
- Remaining 4% distributed mainly amongst Transmitter (50%), Emission (39%), Fire (4%), and Detonation (4%).
- The simulation contained 79 players sending PDUs, though the actual mix of vehicles involved in this exercise is not available.
- Air vehicles issued one ESPDU/second average in that demonstration, while land vehicles averaged 0.17 ESPDUs/ second. Some participants in that demonstration issued packets at frame rate: 20 ESPDUs/second

DIS (Distributed Interactive Simulation)

- In DIS, we get more of a notion that any type of computer plugged into the network that reads/writes DIS PDUs and manages the state of those PDUs properly can fully participate in a DIS environment.
- This fully distributed, heterogeneous network software architecture means that workstation class machines can play against PC class machines.
- There are several instances of fairly large DIS engagements, much larger than the 300 to 500 players for which DIS is designed.
- However, these “DIS” engagements actually modify the DIS network software architecture for their particular circumstances to achieve useful demonstrations.

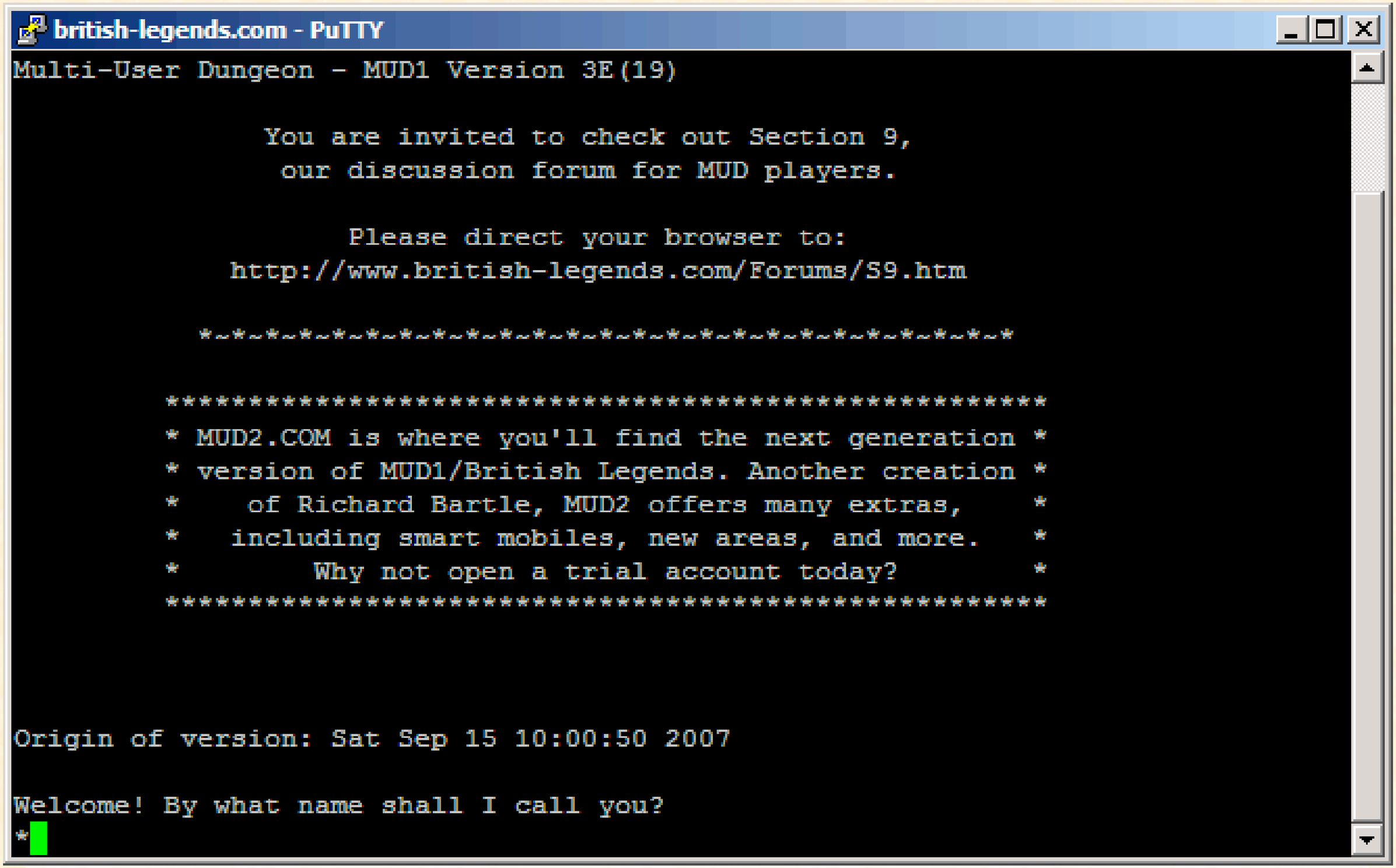
DIS example: CCTT

- The US Army's Close Combat Tactical Trainer (CCTT) is (was?) one of the larger scale networked virtual environments.

MUD1 and MUDs

- In 1978, Roy Trubshaw and Richard Bartle coded, with the MACRO-10 assembler on a DecSystem-10 at university of Essex, a multi-user text game called MUD (Multi User Dungeon)
- It's a text mode game which allows players to move in a virtual world, take virtual objects and use them to kill monsters and other users (they may also talk to each other...)
- Several such multi-user games have been developed since 1978

MUD1 et les MUDs



SGI Flight & Dogfight

- Gary Tarolli of Silicon Graphics, Inc. is probably the person that most in the networked virtual environment community would credit as the originator of their thoughts on networking virtual environments.
- Gary was the original programmer of the Silicon Graphics demo program, Flight, in the summer of 1983. Flight is the program everyone showed you if they had purchased an SGI workstation in the 1984-1992 time period.
- Networking was added into Flight in stages, beginning in 1984.
- The initial networked version of Flight actually used a serial cable between two SGI workstations and ran at something like 7 frames per second on a Motorola 68000 based workstation (about 1 MIPS with maybe 500 polygons per second graphics capability).

SGI Flight & Dogfight

- That demonstration was then upgraded to use XNS multicasting on an Ethernet network in time for SIGGRAPH 1984.
- Flight was distributed in networked form on all SGI workstations sometime after SIGGRAPH 1984 and could be seen in practically every SGI-outfitted lab at that time, either during the day on breaks or after hours.
- Sometime after the release of the networked version of Flight, in early 1985 it is believed, SGI engineers modified the code of Flight to produce the demonstration program Dogfight.
- This modification dramatically upgraded the visibility of net-VEs as players could now interact by shooting at each other.
- They probably created the first networked virtual environment.
- Problem: packets were (are ?) sent at frame rate => saturating the network.

Lucasfilm Habitat

- First commercial graphical virtual world (1986)
- 2D Chatworld (MUDs, MOOs, MUSHs)
- Developed for Lucasfilm Games and Quantum Computer Services (AOL)
- First version was for Commodore 64
- Users payed 8 cent/minute (1 people spent about $1000 in one month ~ 3.5 online days)
- 15 000 users from a potential customer population of 100 000 (BBS users)

DOOM

- On 10 December 1993, Id Software delivered the shareware version of DOOM
- This game is probably the ancestor of most of current networked games
- Like flight and dogfight it used a huge network bandwidth
- An estimated 15 million shareware copies of Doom have been downloaded around the world, passed from player to player by floppy disk or online networks.

Warcraft

- Dune II (1992) is the first Realtime Strategy (RTS) mouse controlled game.

- Warcraft (1994) is the first RTS to include a multi-user mode (through modems, direct serial link or IPX LANs).

Ultima Online and MMORPGs

- Ultima Online is a direct descendant of MUDs or Habitat like games.
- In 1997, it was the first to bring a real massive scale to net-VEs with 100 000 paying players ($9.95/month) in one year (totaling a 12 million dollars revenue per year when adding the cost of retail game boxes).

World of Warcraft

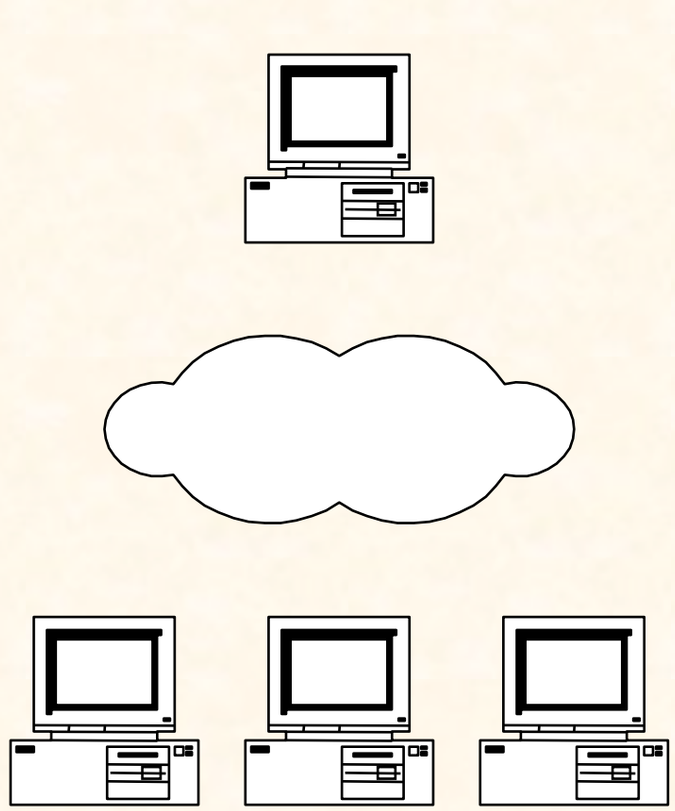
- World of Warcraft, delivered in 2004 (10 years after Warcraft 1), is currently the most played MMO in the world (at least in the western countries) with more than 11.5 millions regular players (12/08).
- However each server (in fact cluster of servers) “only” manages 5000 simultaneous players.

EVE Online

- EVE Online is currently the most “massive” MMO managing more than 50 000 simultaneous players on the same server (cluster) (53 850 in 03/09).
- Interactivity is somewhat limited however.

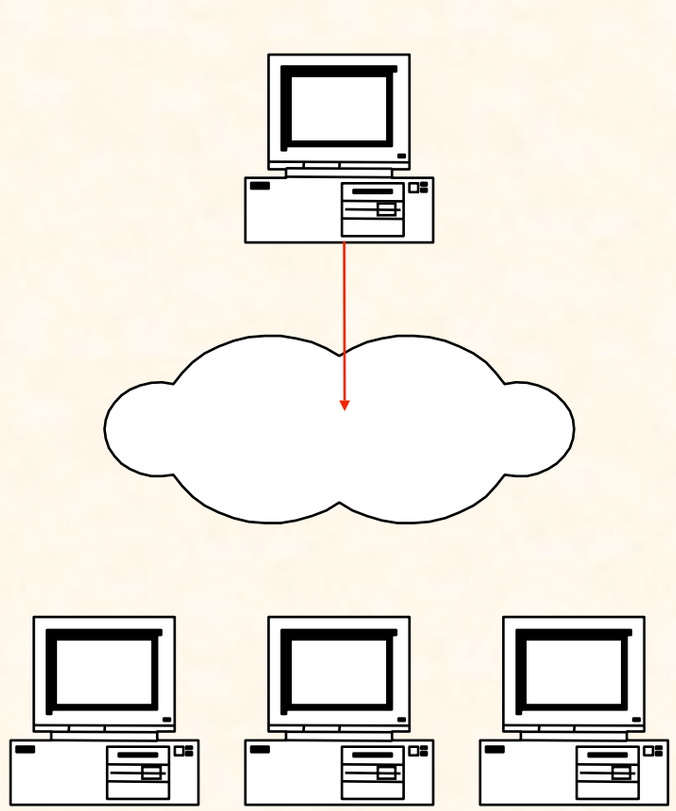
Net VEs architecture

- Communication model



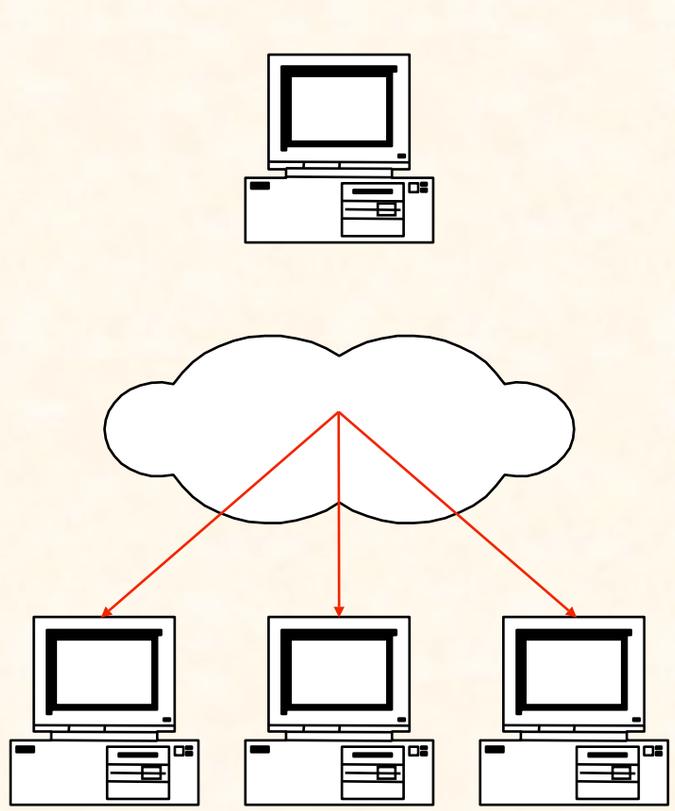
Net VEs architecture

- Communication model
  - Broadcast



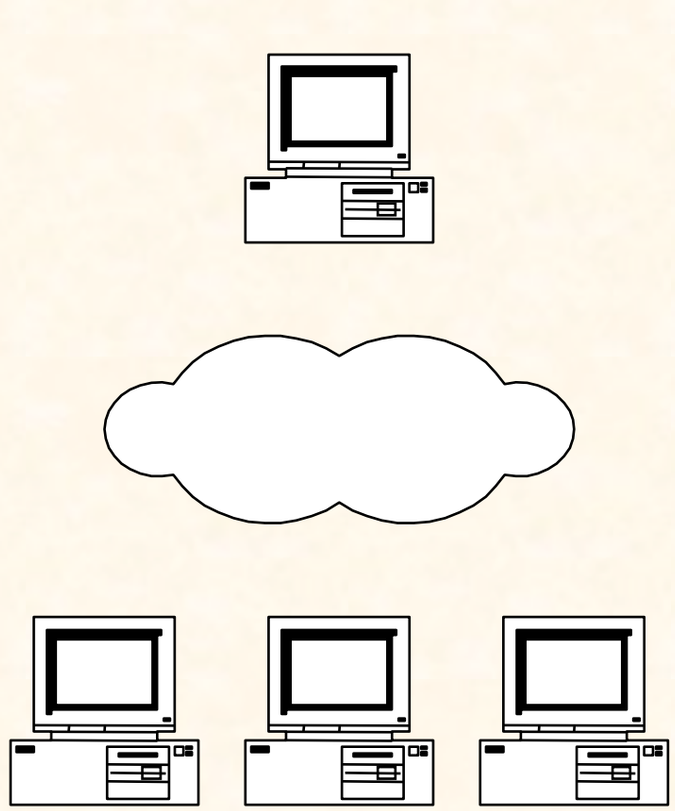
Net VEs architecture

- Communication model
  - Broadcast



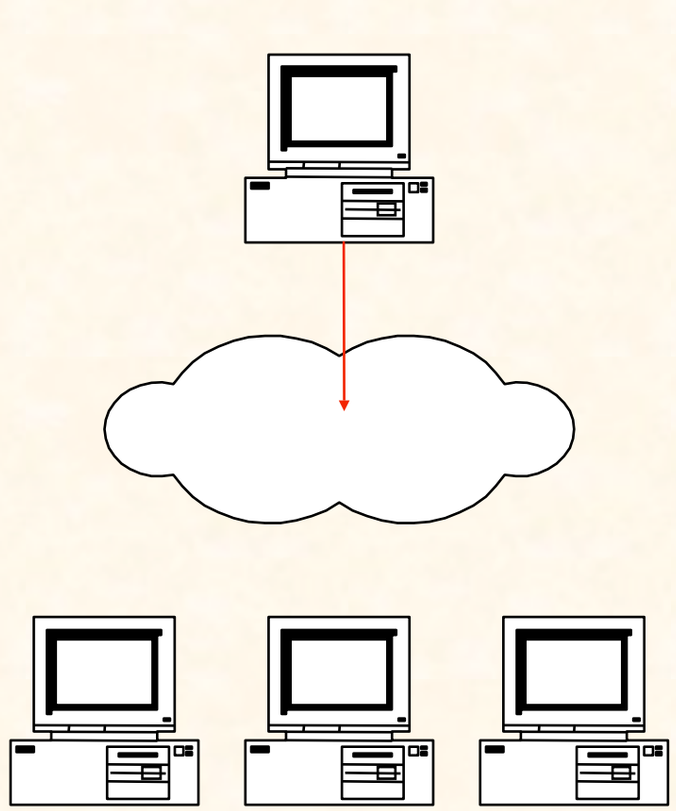
Net VEs architecture

- Communication model
  - Broadcast



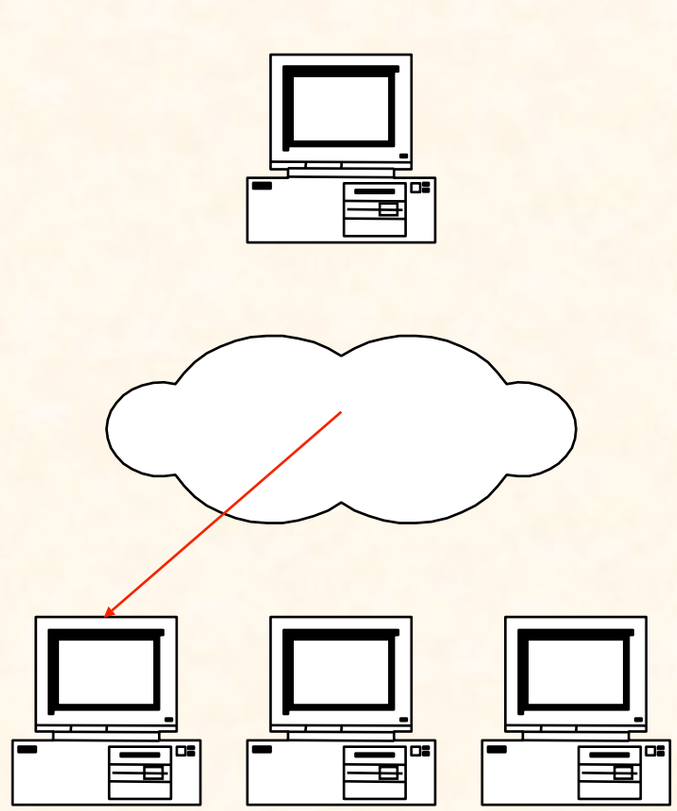
Net VEs architecture

- Communication model
  - Broadcast
  - Point to point



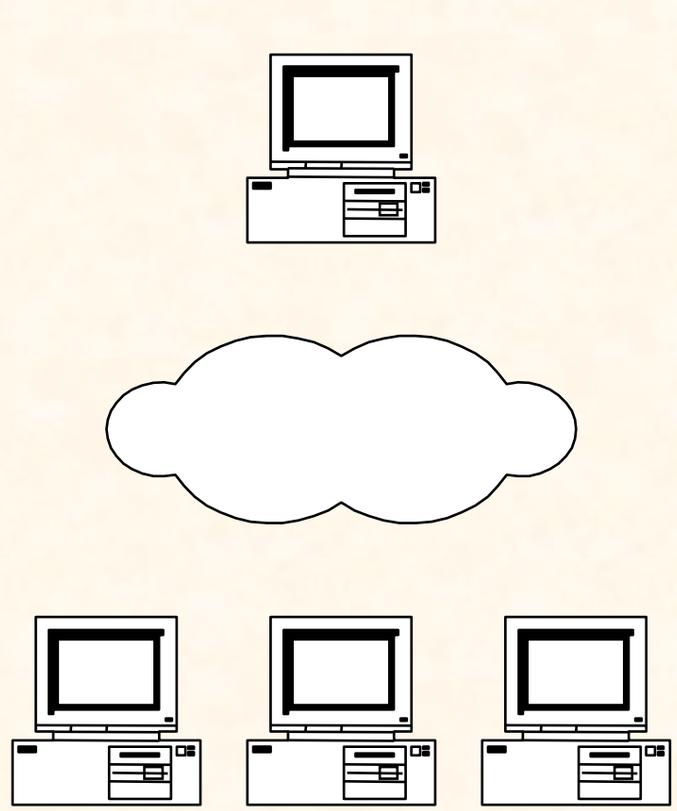
Net VEs architecture

- Communication model
  - Broadcast
  - Point to point



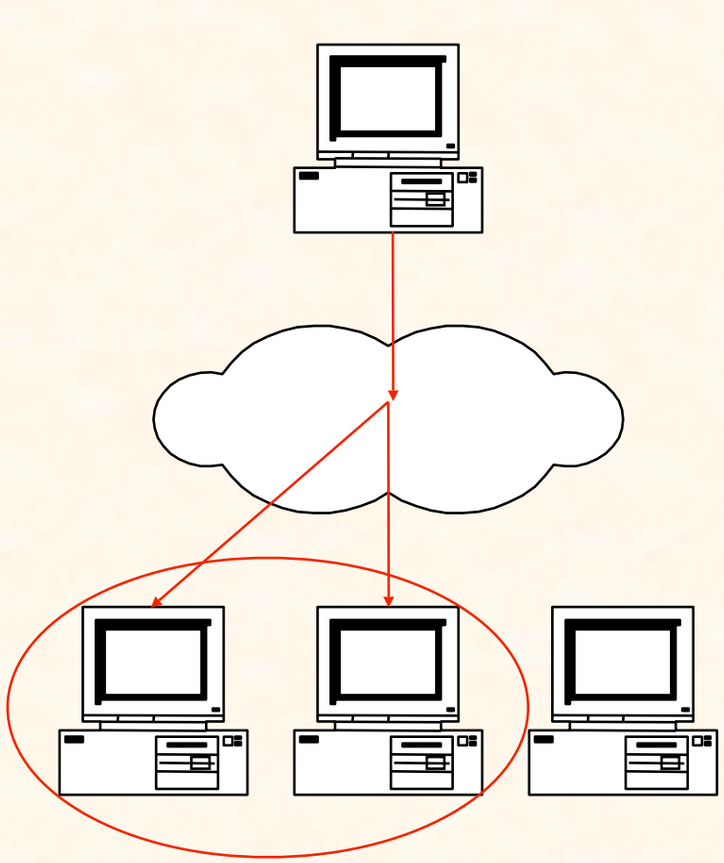
Net VEs architecture

- Communication model
  - Broadcast
  - Point to point



Net VEs architecture

- Communication model
  - Broadcast
  - Point to point
  - Communication groups (multicasting)



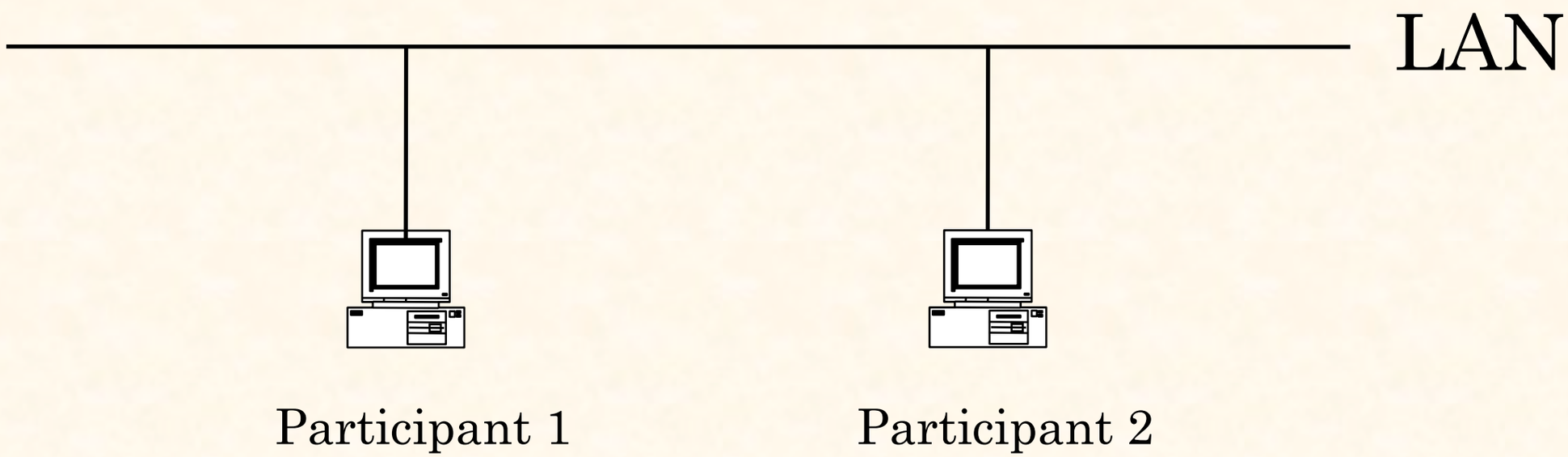
Net VEs architecture

- Distributed architecture
  - Fully distributed (no server)
    - ✓Peer to Peer P2P
    - ✓P2P with multicasting
  - Centralized (Client/Server C/S)
  - Multiple servers architecture
  - Coordinated multi servers architecture

- Architecture choice

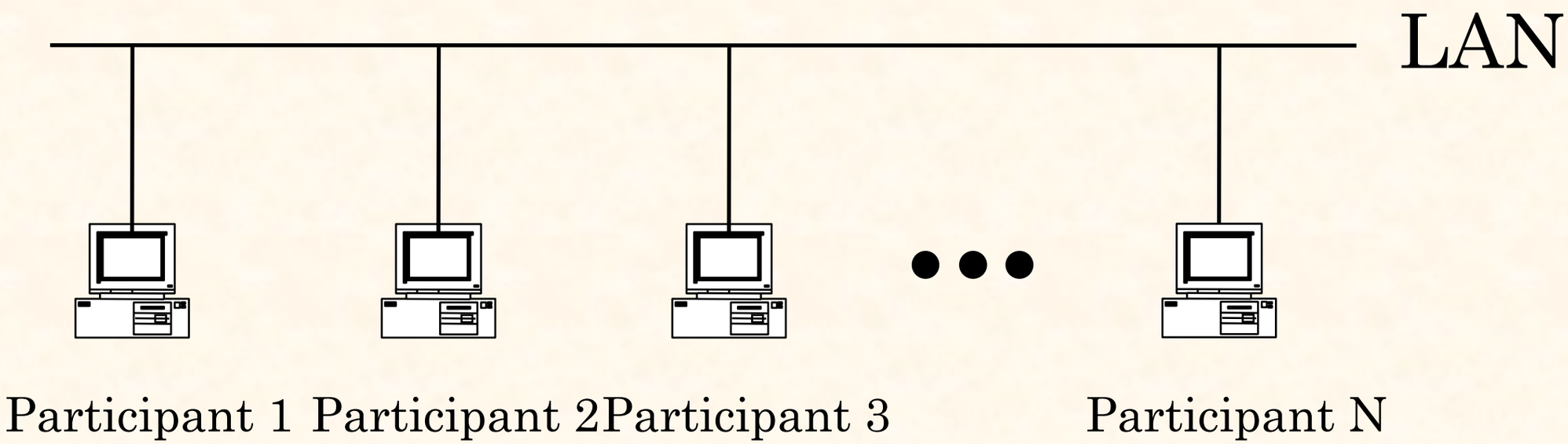
Peer to peer

- Simple case: 2 participants on a LAN
  - Each participant's machine directly sends state changes to the other participant's one



Peer to peer

- General case: each participant can talk to every other participants using broadcast



Peer to peer

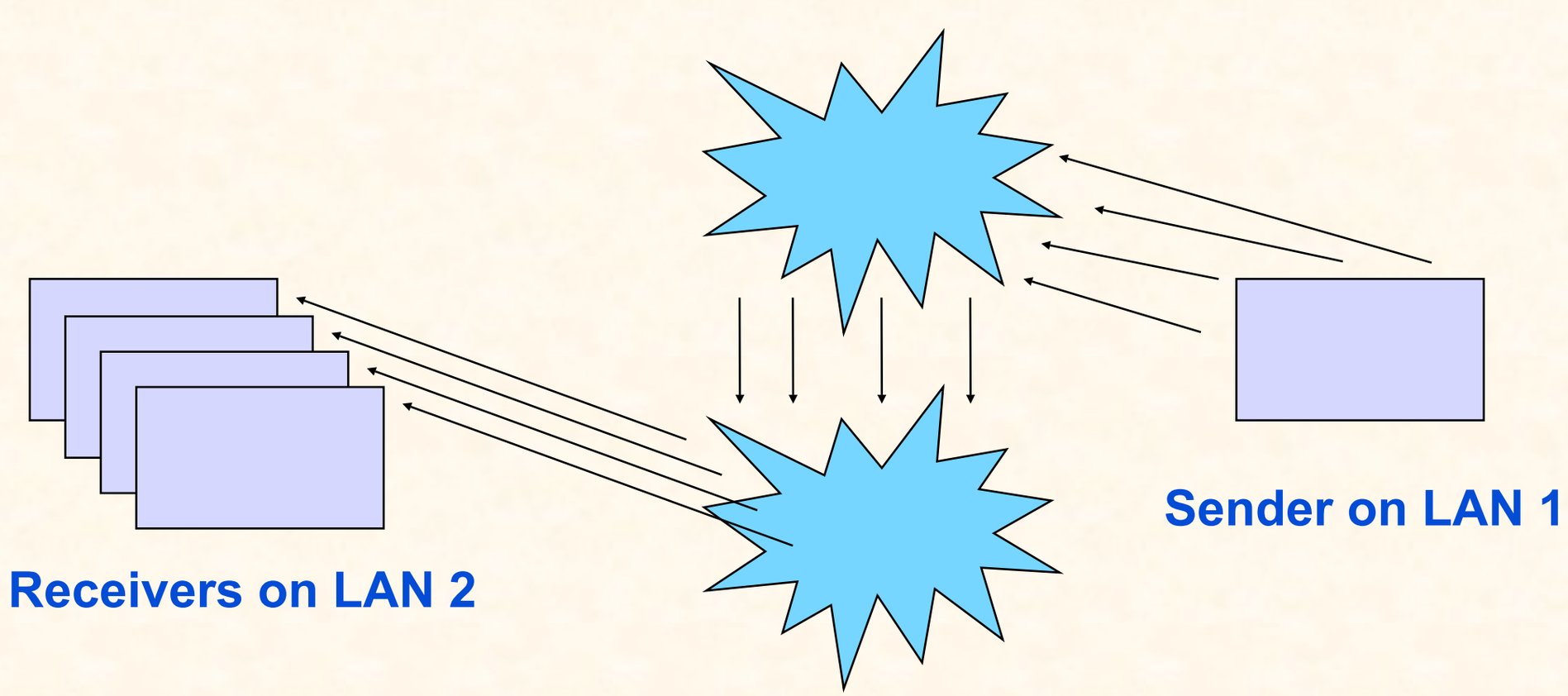
- Example: 10 Mbps Ethernet LAN (100 Mbps)
  - Available bandwidth : ~ 7 Mbps (80 Mbps)
  - Ethernet saturates when bandwidth usage reaches about 70% (switching saves a bit more bandwidth)
- Example of standard packet: DIS's ESPDU
  - 144 bytes long (1152 bits)
- Worst case scenario
  - limit is 6000 PDU / second with 7 Mbps (70000)
  - if each participant generates 30 PDU / second
  - limit is 200 participants on a 10 Mbps Ethernet LAN (2300)

Peer to peer

- But, we don't only want to play on LANs
- WANs cannot use broadcasting. Each packet must be sent to each participant individually.
- Therefore 200 participants on a 10 Mbps WAN becomes:
  - 1 participant sending to 200 participants
  - 200 participants sending to 1 participant
  - 14 participants simultaneously communicating
  - 1 participant managing 14 vehicles sending to 14 participants
  - ...

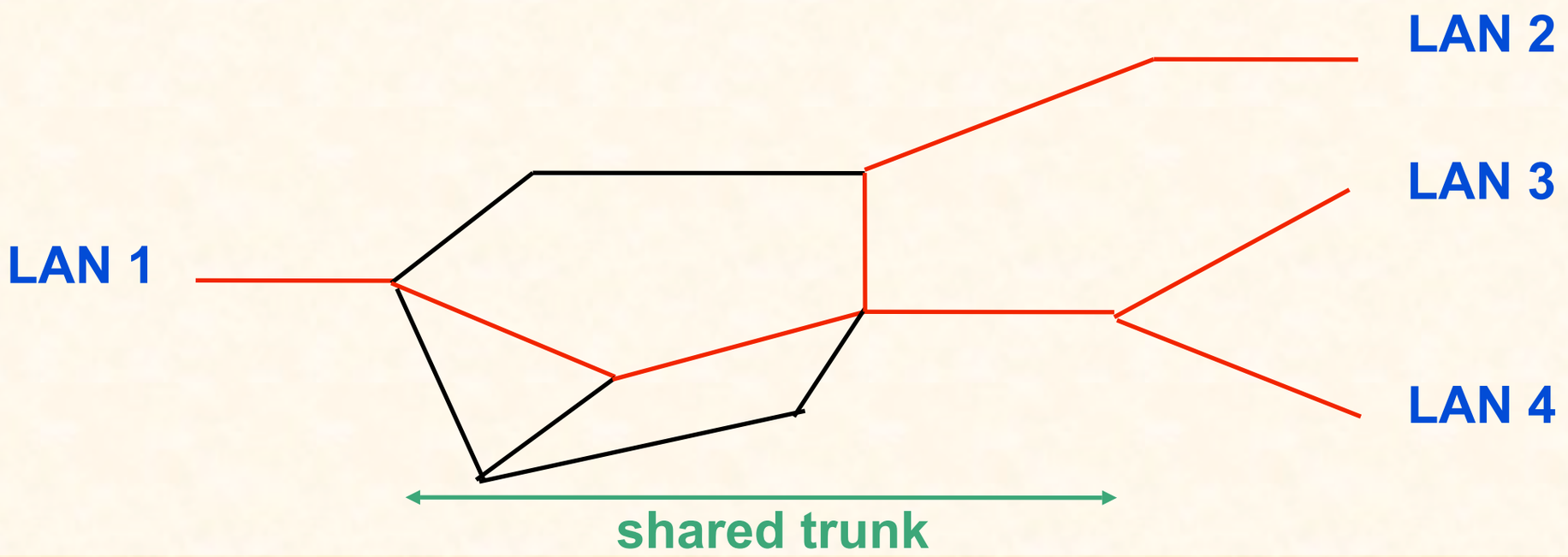
P2P with Multicasting

- Problem : sending messages on a WAN is costly
  - Messages between LANs are sent separately
  - The same message may be sent several times between two LANs



P2P with Multicasting

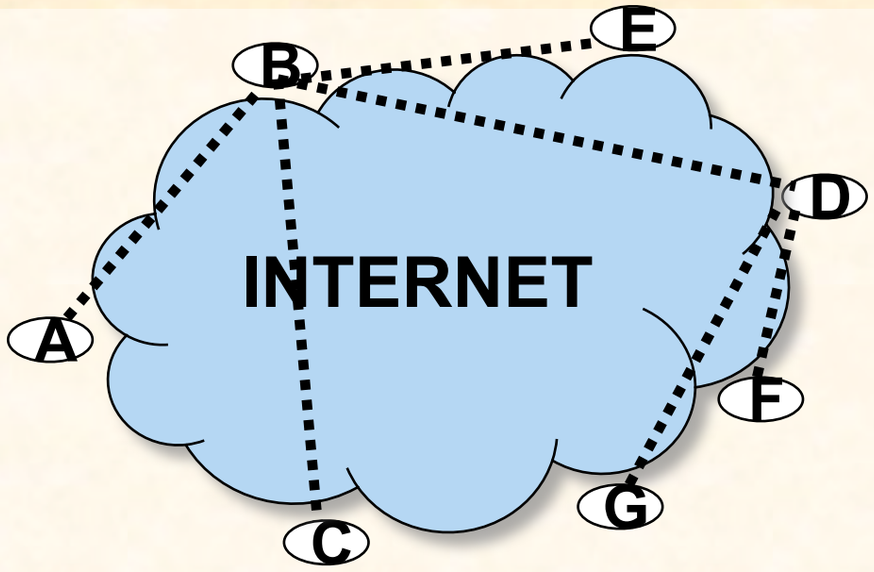
- Solution: multicasting uses « trunk sharing » to save network bandwidth
- After joining a multicast IP address every participant receive all messages sent on this address
- Allows groups of any size to communicate with a single transmission
- Often presented as « The Solution » to scalability problems

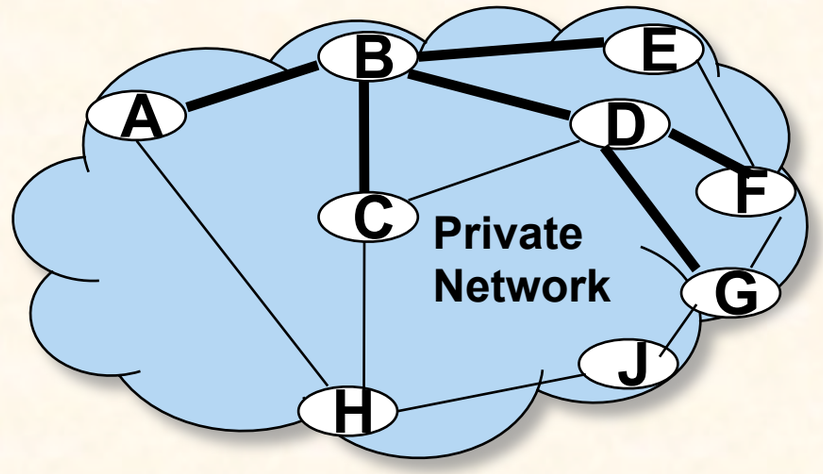


Multicasting nowadays

- Multicasting is quite an old technology but the MBone isn't the Internet
  - Multicast routers are not configured for anyone to use multicasting
  - However it's the best solution ...
  - … therefore if a small percentage of the participants can't multicast we can use proxies
    - ✓Each proxy joins the Multicast address
    - ✓Some clients directly connect through a proxy (usually using UDP)
  - Other solutions: overlay multicast or application level multicast

Overlay Multicast





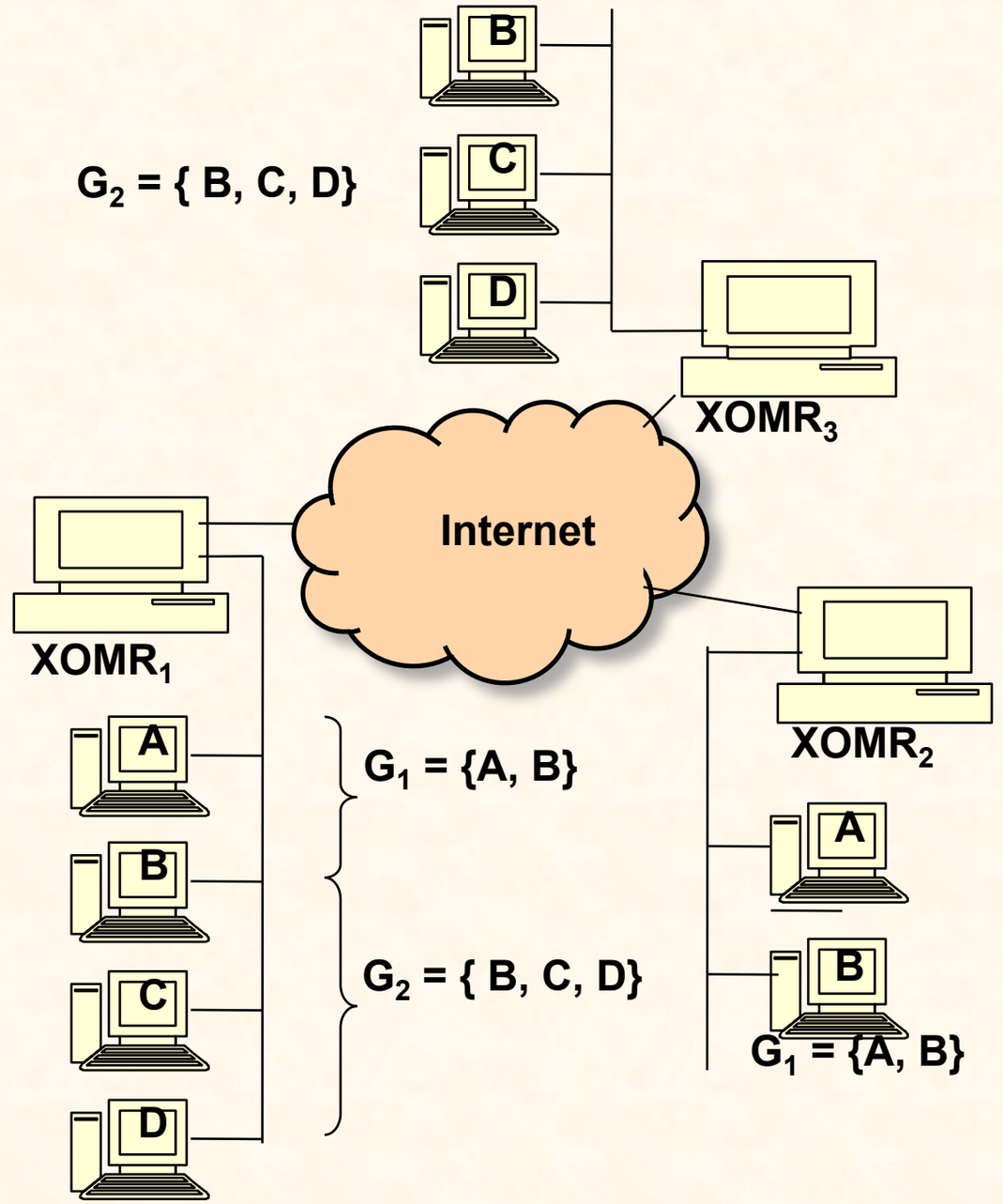
Overlay Multicast :

- many to many
  - ➢ multiple senders for the same group
  - ➢ source based trees
- Open network
  (independent of any management domains)
- Can be adapted to an application
  - ➢ End to end QOS
  - ➢ Efficient management of groups

Standard IP Multicast:

- Generally: 1 to many
  - ➢ only one sender
  - ➢ root based tree
- Private network
  (usually only works on one management domain)
- Independent of the application

Overlay Multicast example: XOM

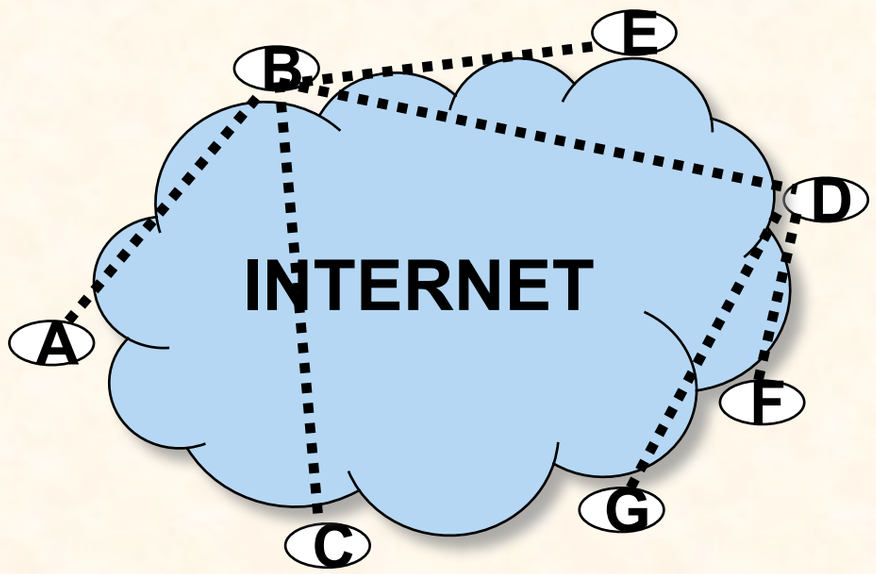


Point to point communications

standard multicast

Application Level Multicast

- Same principle as Overlay Multicast
- But everything is managed in the applications
  - Advantage: we can have a better adaptation to the application
  - Disadvantage: the application becomes more complex



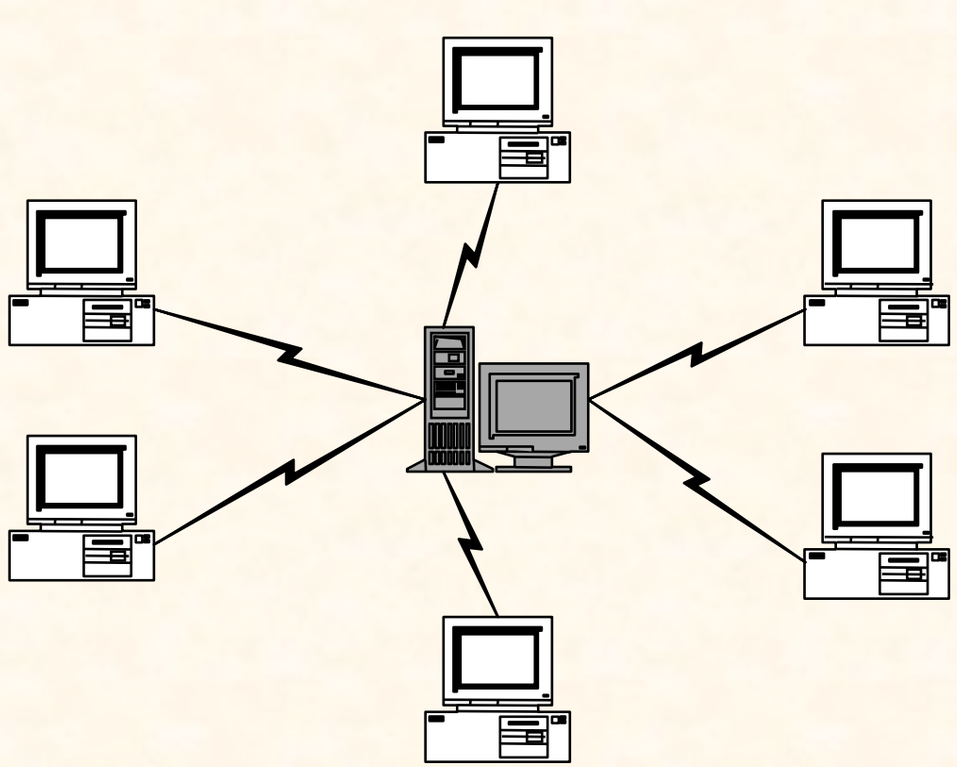
Here A,B…G are applications not Overlay Mcast routers

Fully distributed architectures

- Advantages
  - No central point of failure, no bottleneck
  - Multicasting saves network bandwidth
  - Can use several Multicast addresses for message filtering
- Disadvantages
  - Can be difficult to setup and manage
  - Bandwidth usage (without Mcast) is in O(participants^2)
  - Every packet exchanged on a LAN may need to be examined by every computer (Mcast and Bcast without switching)

Centralized architectures

- Used by most FPS games on the Internet
  - Usually manage a few 100s players simultaneously
  - Limit depends on the complexity and interactivity of the application

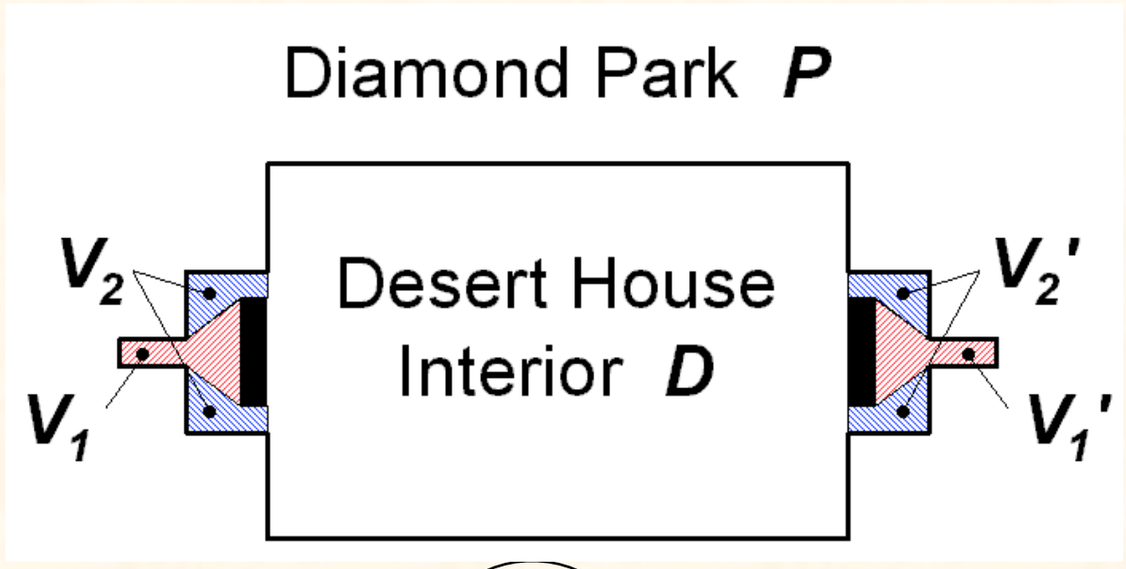


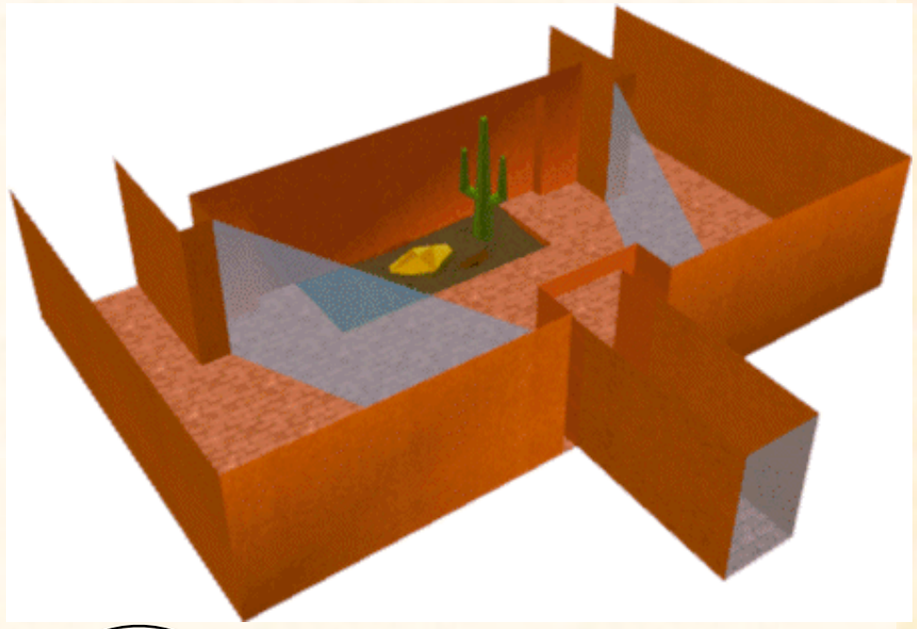
Centralized architectures

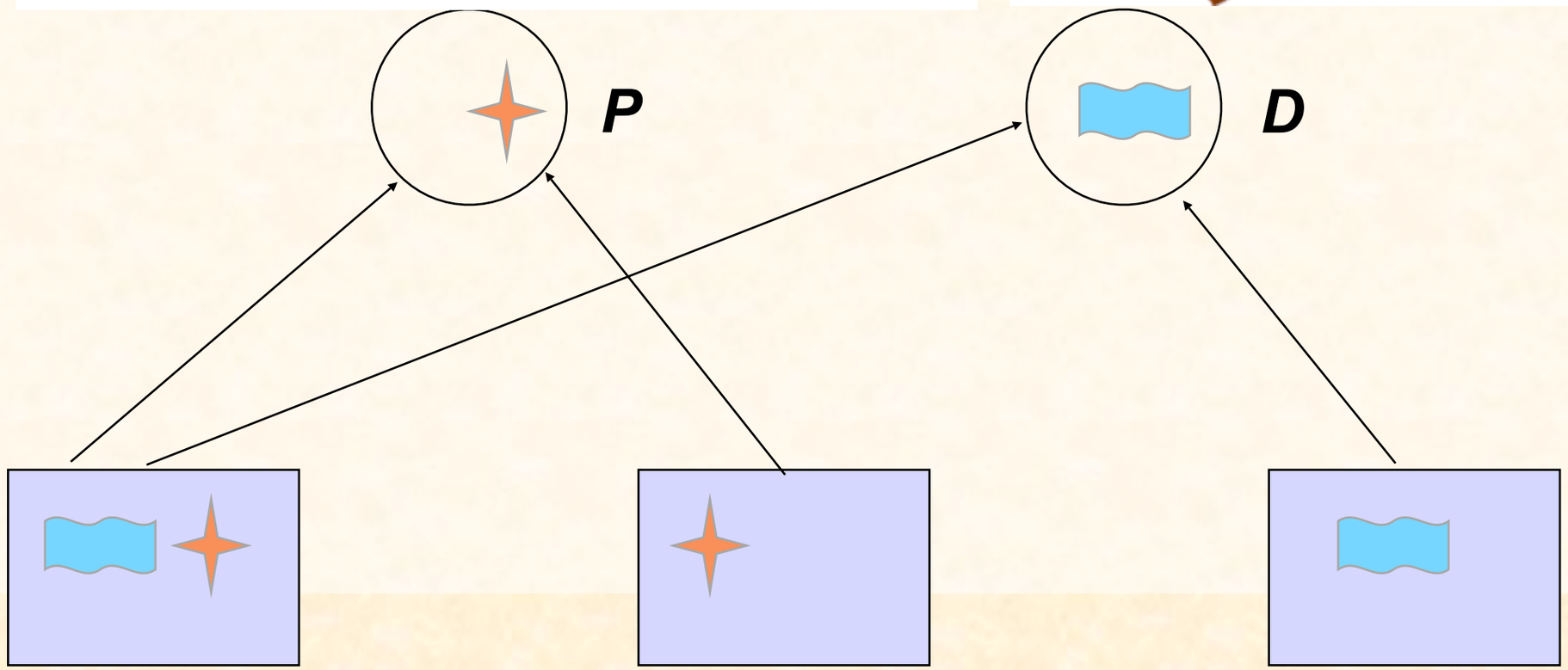
- Disadvantages
  - The server can become a bottleneck
  - Reliability problems (single point of failure)
  - Increases latency
- Advantages
  - Bottlenecks can be useful
  - You can authenticate/bill participants easily
  - Server can filter messages
  - Server can compress several messages in one message

Multi-server architectures

- Servers can manage different parts of the VE





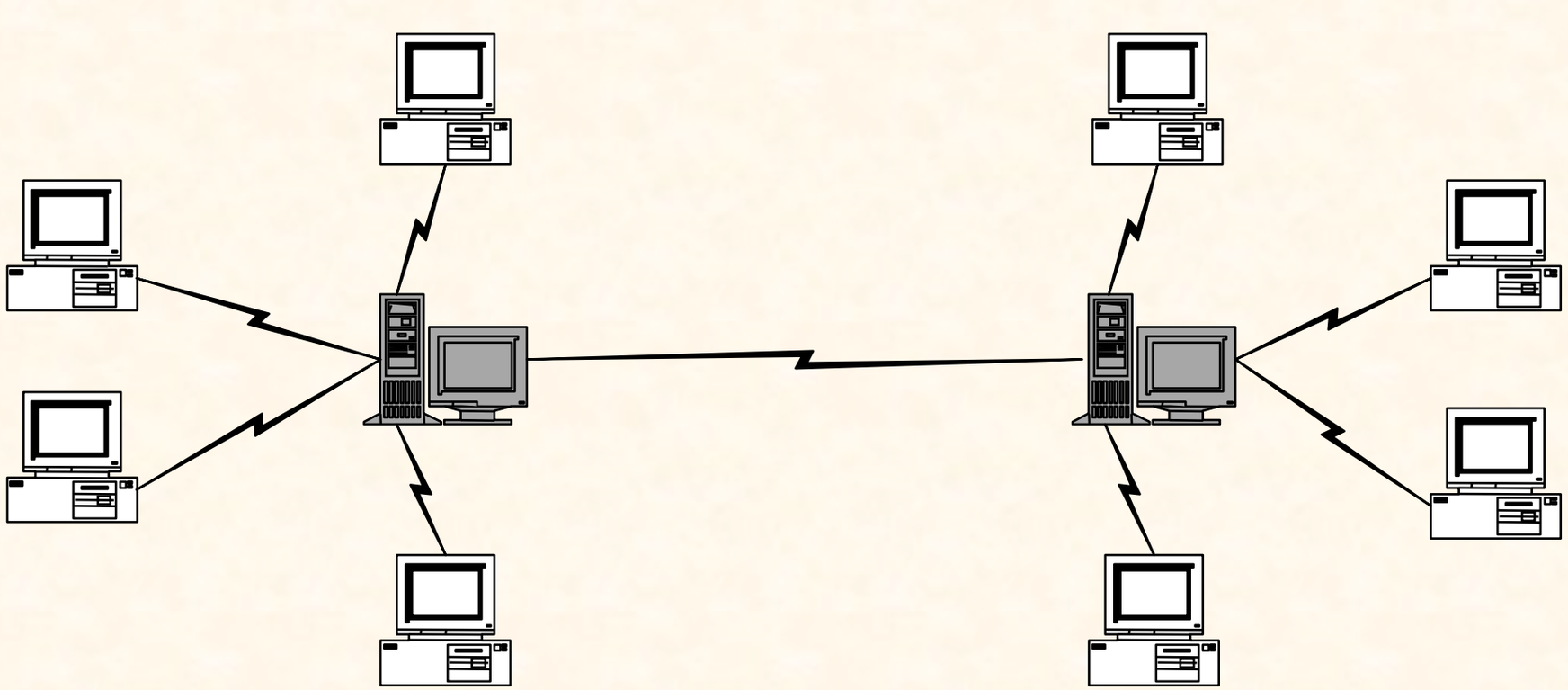


Multi-server architectures

- Advantages
  - Reliability: redundancy through the use of several servers
  - Scalability: divide tasks either by
    - ✓grouping clients (MMO shards)
    - ✓dividing the world
- Disadvantages
  - Multi-server shared objects do not propagate their changes (client grouping)
  - We still can have one single point of failure (world division)
    - ✓in fact several single points of failure
  - That's how the World Wide Web works
    - ✓it works isn't it ?

Coordinated multi server architectures

- Servers communicate with each other
  - in order to manage shared objects
  - for load balancing



Coordinated multi server architectures

- Advantages
  - A hierarchy of servers can be used to filter efficiently
  - Can manage dynamic load balancing
  - Sharing the same virtual world
  - Servers can communicate using Mcast (hybrid P2P and C/S)
- Disadvantages
  - Coordination is a difficult task
  - If you use a hierarchy you can increase latency

Summary

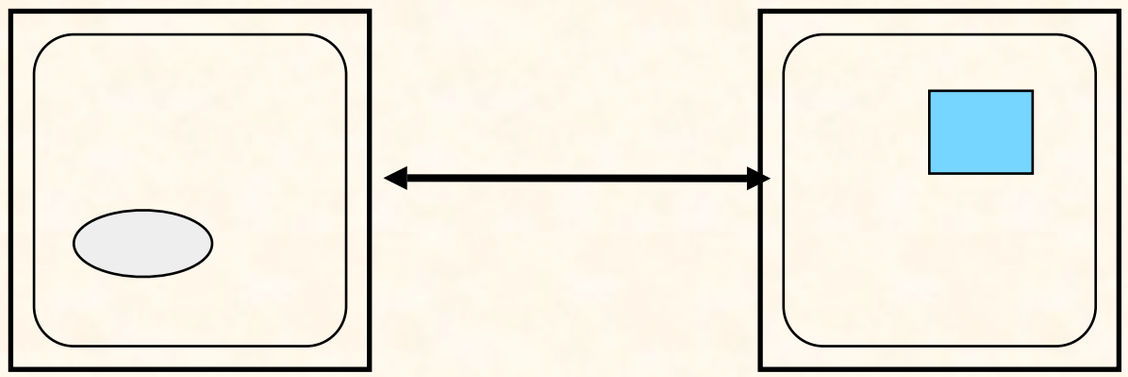
- Choice isn't easy
- If the most important thing for you is
- scalability: P2P / Multi-servers
- reliability: coordinated multi-servers
- simplicity: centralized
- interactivity: P2P

Dynamic Shared State Management

- Definition
- Where is the problem ?
- Coherency/throughput tradeoff
- Different solutions
  - Centralized repositories
  - Frequent state regeneration
  - Dead-reckoning

Dynamic Shared State Management

- A successful networked VE is used by several users
- … distributed everywhere in the real world
- … constantly modifying their avatars (position, etc.)
- … and trying to see each other in realtime



- Problem: how can we be certain that they all see the same world ?

What is the problem ?

- Goal: transmit information to every computer in realtime
  - But, network crossing takes time (latency)
  - Each participant may see a different latency
  - Throughput is limited by the network bandwidth
  - Packets may be lost

- The problem: trying to make sure that everyone has received some data takes time

Coherency/throughput tradeoff

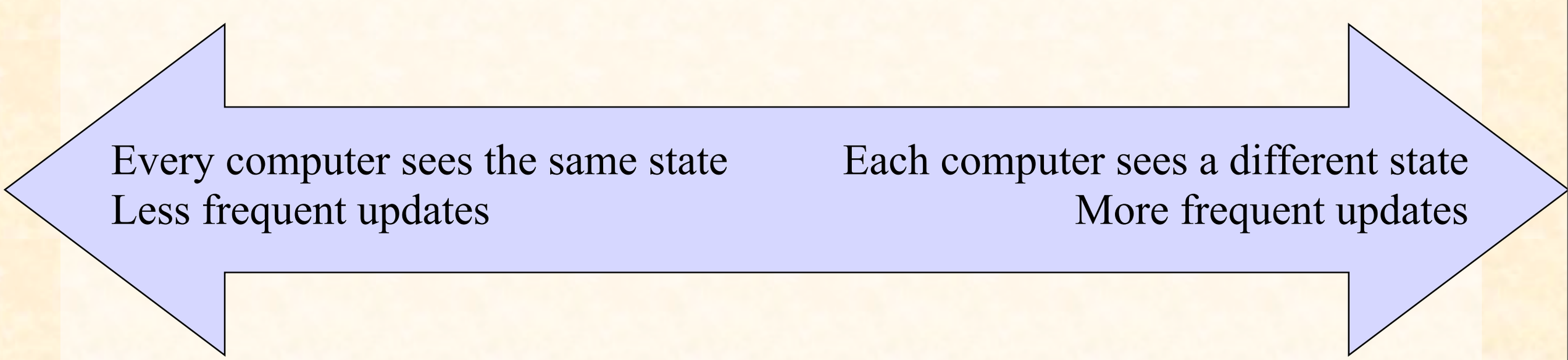
- It is impossible to allow very frequent changes to be made to the shared state and, at the same time, guarantee that every participant sees the same shared state

- We have to choose between
  - a very dynamic world (with lots of frequent changes)
    … and do not wait transmission based incoherency problem resolutions
  - and a less dynamic world
    … and take the time needed to solve all incoherency problems

Design implications

- You need to choose a technique while taking into account the specificities of the simulated world

Coherency ***Throughput***

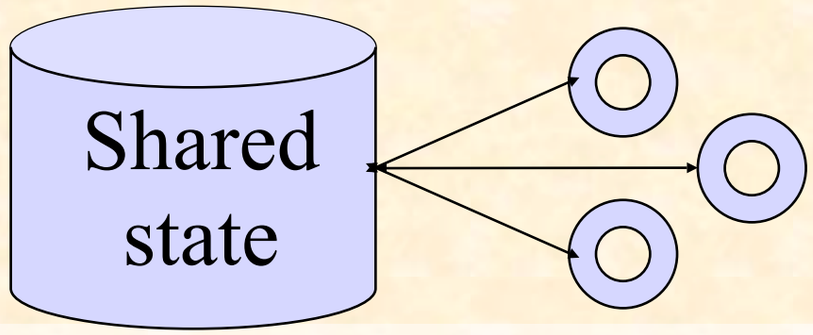


Central Repositories

Frequent Regeneration of the shared state

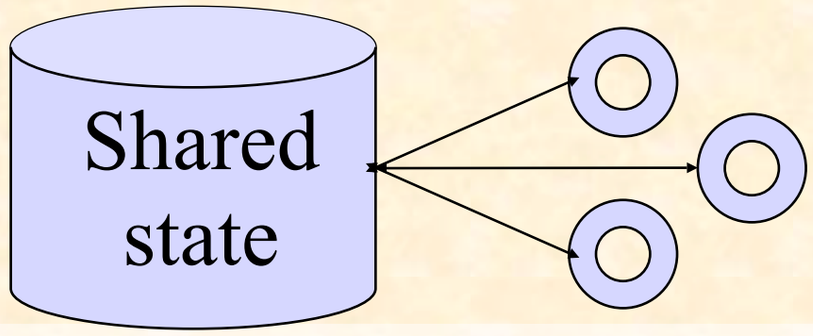
Dead Reckoning

Central Repositories



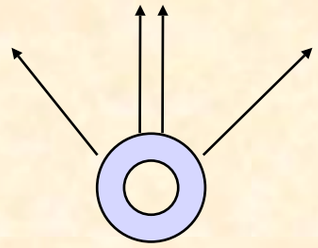
- Store the shared state in a central place
  - Updates are made to this central repository
  - Every shared state reads are done from this central place (no cache)
  - Example: place everything on a networked file system (NFS or AFS)
  - Better solutions: virtual repositories
    - ✓A server sends updates synchronously to each client cache
    - ✓Shared coherency protocols (ex. ISIS)
  - Used by the first versions of DIVE, BrickNet, Shastra, and some video-conferencing tools...

Central Repositories



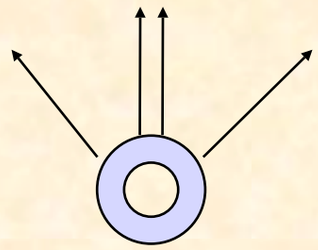
- Advantages
  - Absolute coherency of the shared state
  - Data is owned by everyone -- You just need to add locks or semaphores if it is needed by the application
- Disadvantages
  - Often creates a central point of failure
  - Often creates a bottleneck
  - Not very efficient
  - Big communication overhead
- Only use this for small systems on LANs or for applications that require a guaranteed shared state

Frequent regeneration



- Frequently send your current state to everyone else
  - Often use multicast with filtering to reduce bandwidth consumption
  - You can easily add that in your even loop or on a timer
  - Ignores network losses: updates are frequent enough so that incoherencies don't last for long
  - Can be integrated with a server which manages object ownership or with a distributed locks system
  - Used by SGI Dogfight, RING, Doom (and the first FPSes - up to Quake 1)

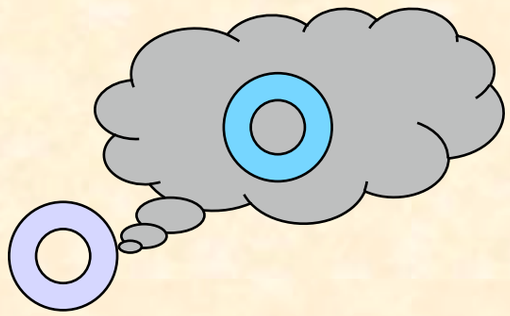
Frequent regeneration



- Advantages
  - Easy to implement
  - No need for servers or other infrastructure
  - Better throughput than central repositories
- Disadvantages
  - May use a huge bandwidth (only use this on LANs please)
  - Users perceive the network lag and its variation directly
  - Not transparent to the user because of this

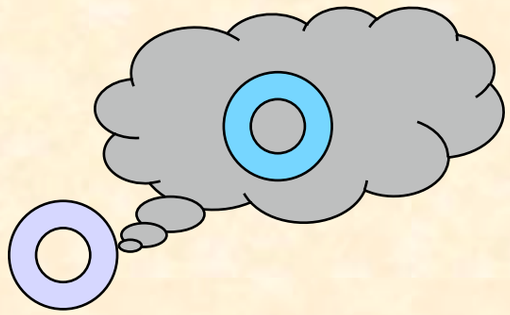
- Most used technique, good for average systems on LANs, useful if you need to adapt a monolithic system

Dead-reckoning



- Predict the state of distant objects
  - Computers try to predict the behaviors of distant objects using their past update messages
  - When an update is received, we use convergence algorithms to modify the (badly) predicted state and update prediction algorithms
  - Updates are more infrequent (only when prediction is too bad)
  - Object ownership must be explicit
  - Used by all military environments (SIMNET, DIS, NPSNET...) and most modern FPS games and MMORPGs

Dead-reckoning



- Advantages
  - Each object may generate its update messages autonomously
  - Hides the network latency
  - Reduces update message numbers and therefore the bandwidth usage
- Disadvantages
  - More complex to implement
  - Prediction model accuracy depends on the object type
  - Prediction errors can be big if you have a bad network (lots of losses or high bandwidth)

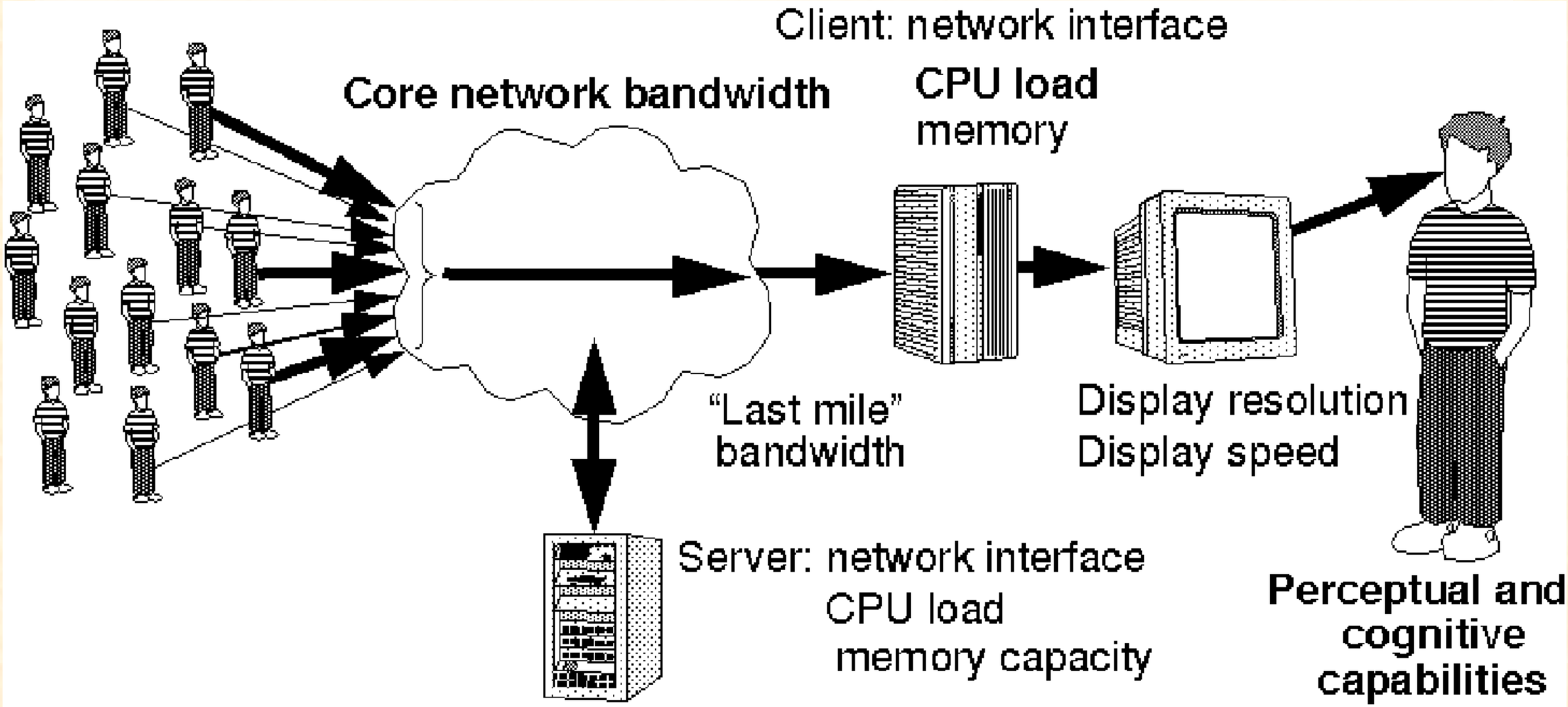
- Interesting for big environments on WANs that can cope with small incoherencies

Area of Interest Filtering

- Problem overview
- Defining Interest
- Examples
  - Spatial Model of Interaction
  - Predictive Interest Management
  - Expanding Spheres
  - HLA routing spaces
  - NPSNET spatial filtering
  - RING/Spline spatial filtering
  - Space Scale Structure
  - Three-tiered interest management

Problem Overview

- Potential bottlenecks



Problem Overview

- Goal : see and hear « enough », but not more...
  - Scalability
  - Efficiency
- « enough » is defined by:
  - Interest, perception, visibility...
  - System constraints (bottlenecks)

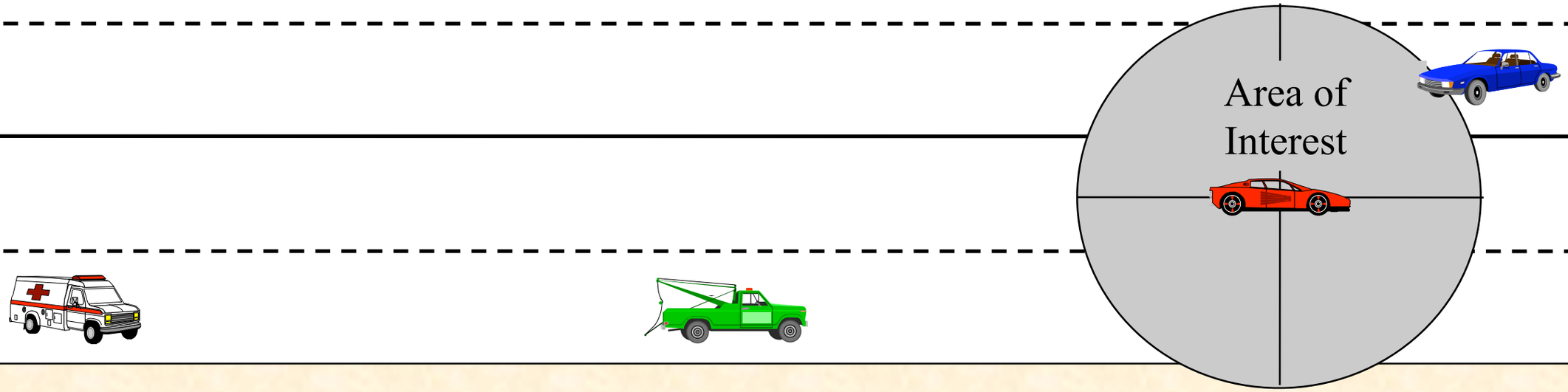
Problem Overview

- Goal : see and hear « enough », but not more...
  - Scalability
  - Efficiency
- « enough » is defined by:
  - Interest, perception, visibility...
  - System constraints (bottlenecks)

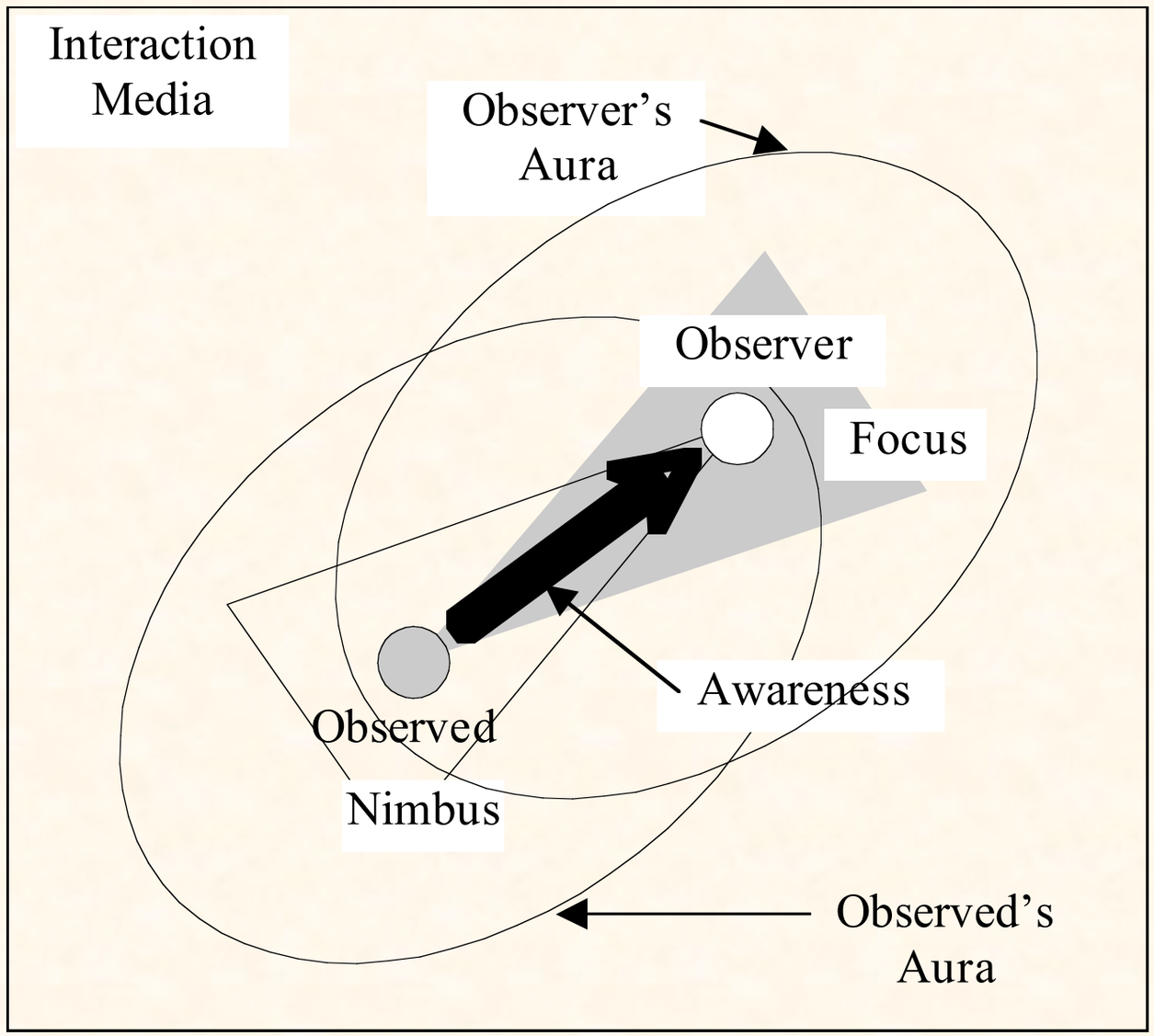


Problem Overview

- Goal : see and hear « enough », but not more...
  - Scalability
  - Efficiency
- « enough » is defined by:
  - Interest, perception, visibility...
  - System constraints (bottlenecks)

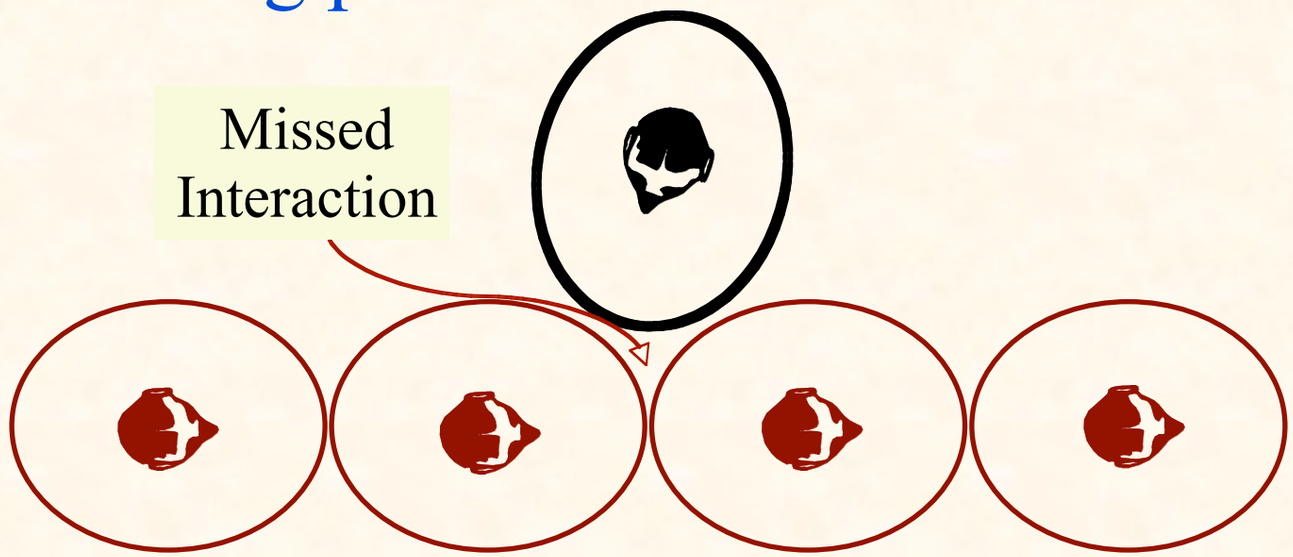


MASSIVE's Spatial Model of Interaction

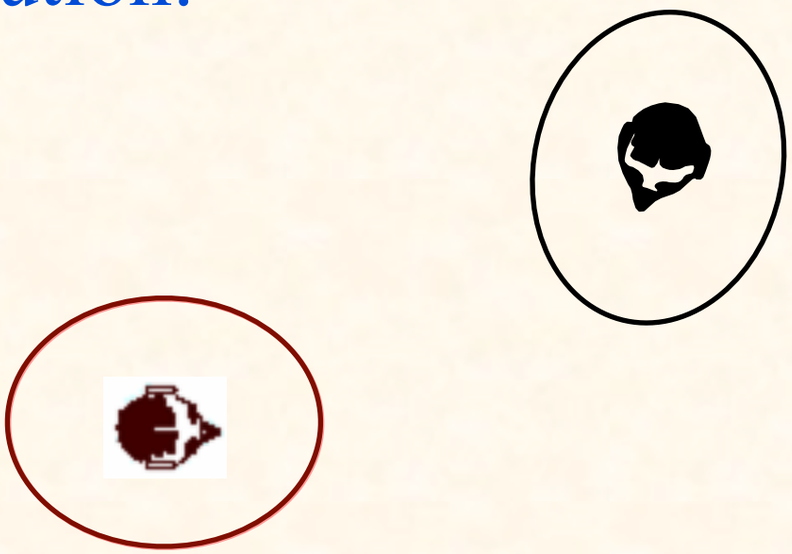


Predictive Interest Management

- Based on the spatial model of interaction
- Solves the following problem:

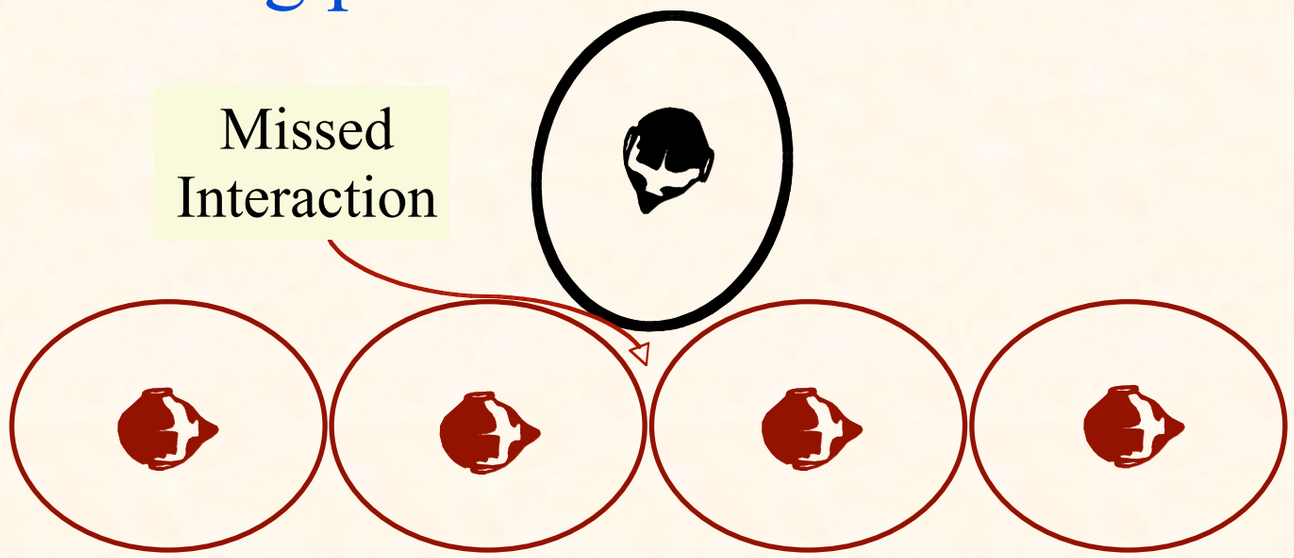


- Solution:



Predictive Interest Management

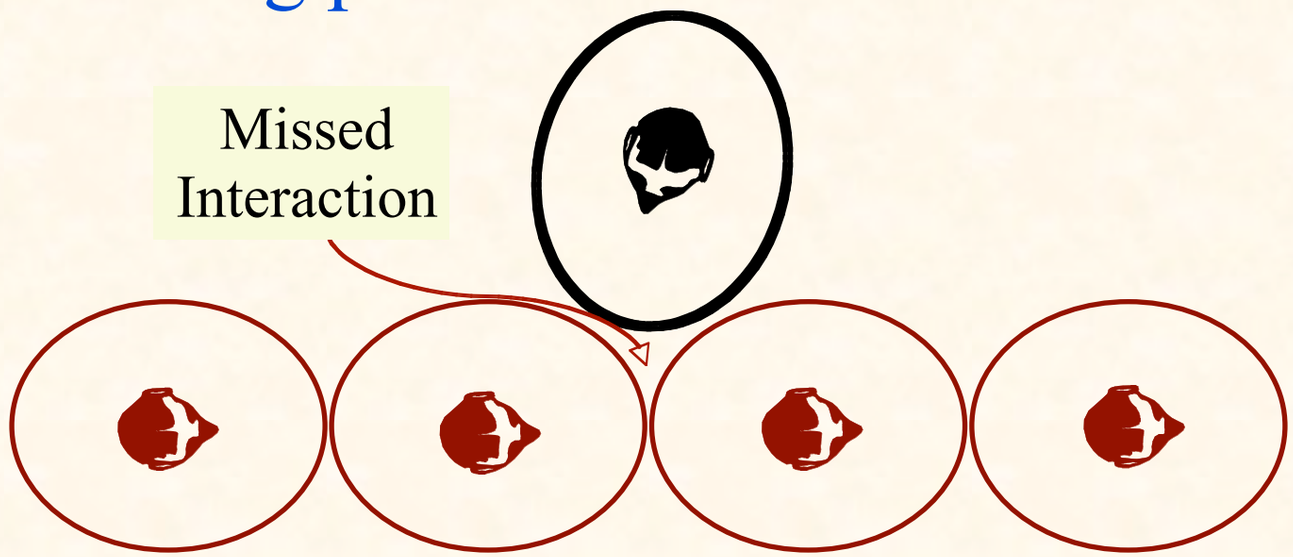
- Based on the spatial model of interaction
- Solves the following problem:



- Solution:

Predictive Interest Management

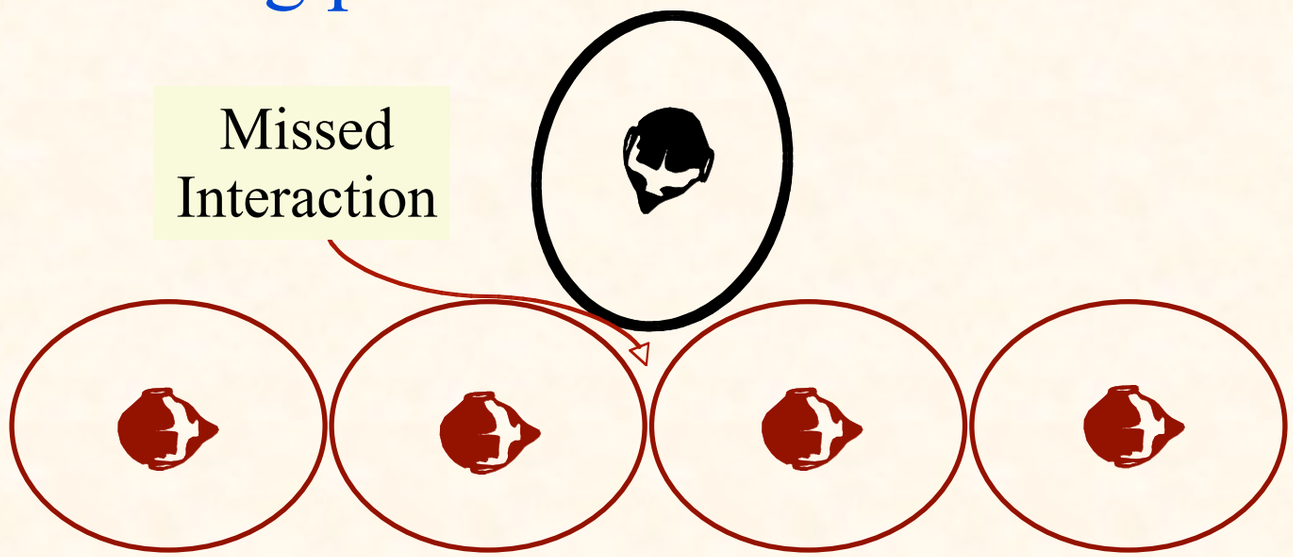
- Based on the spatial model of interaction
- Solves the following problem:



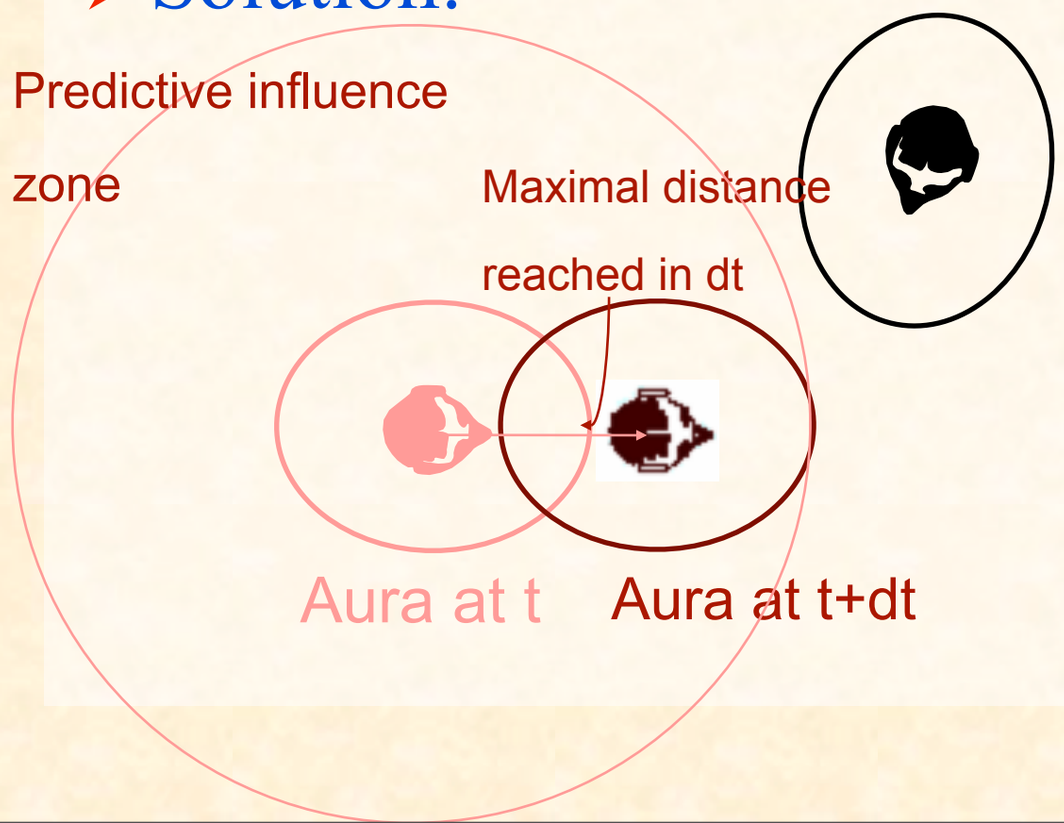
- Solution:

Predictive Interest Management

- Based on the spatial model of interaction
- Solves the following problem:

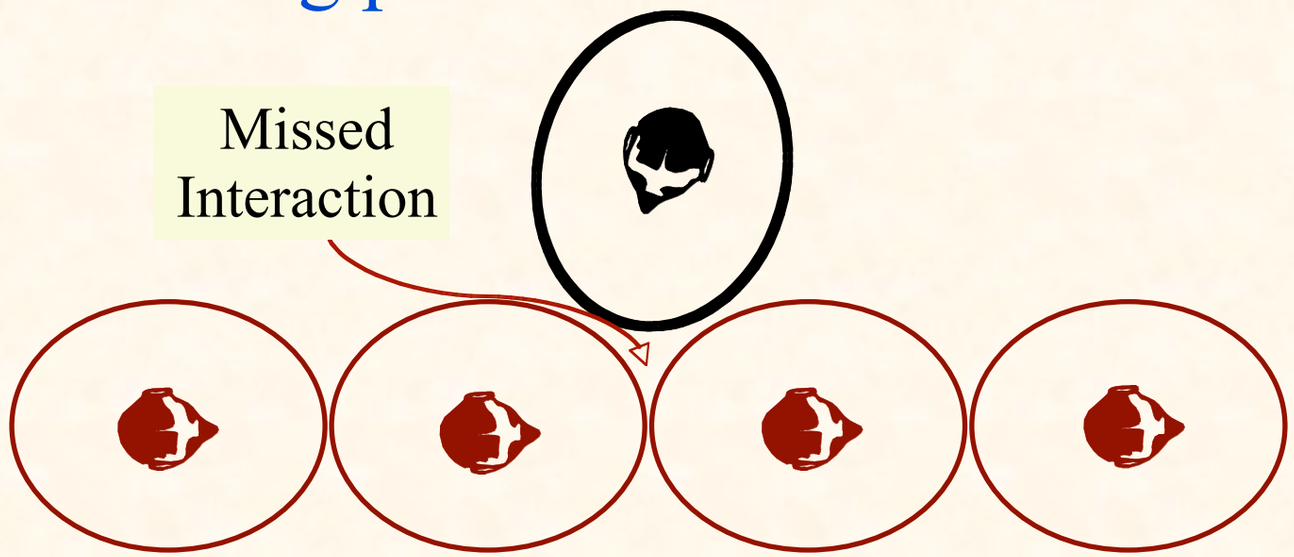


- Solution:

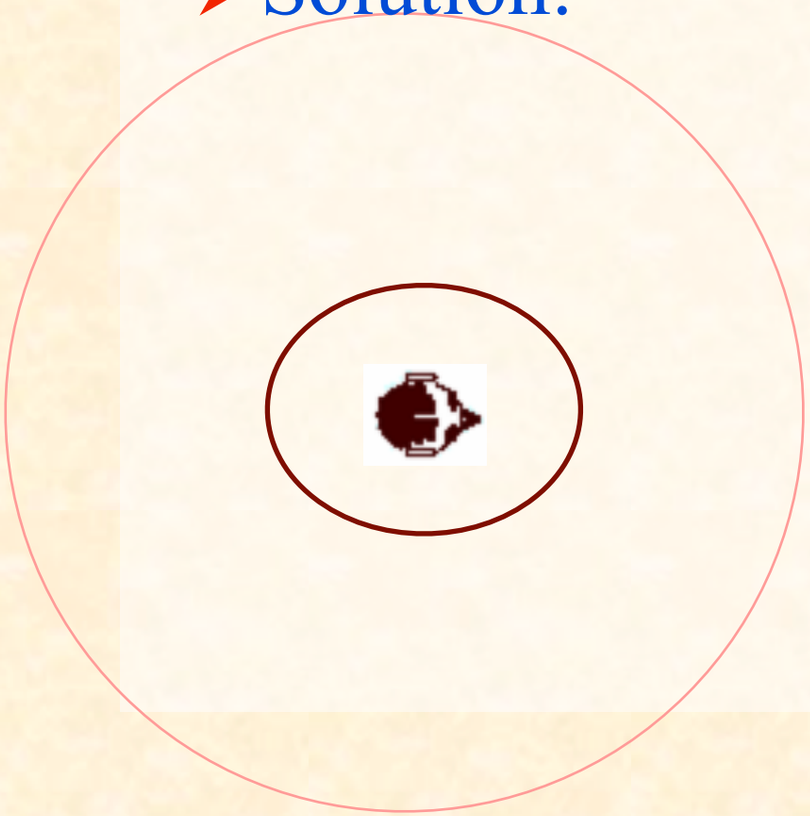


Predictive Interest Management

- Based on the spatial model of interaction
- Solves the following problem:

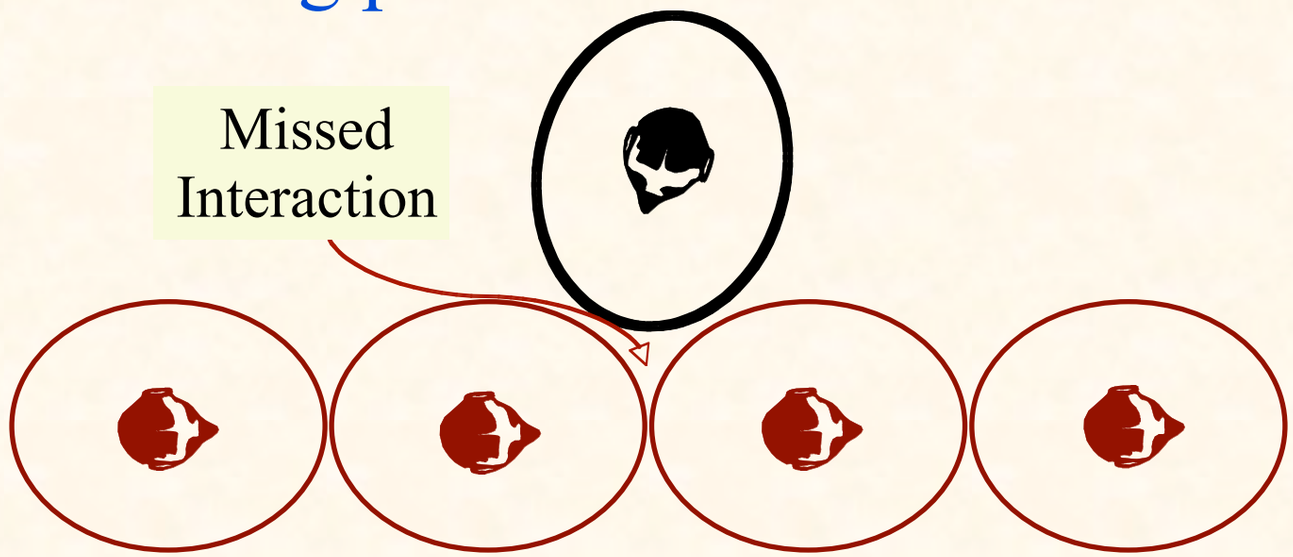


- Solution:

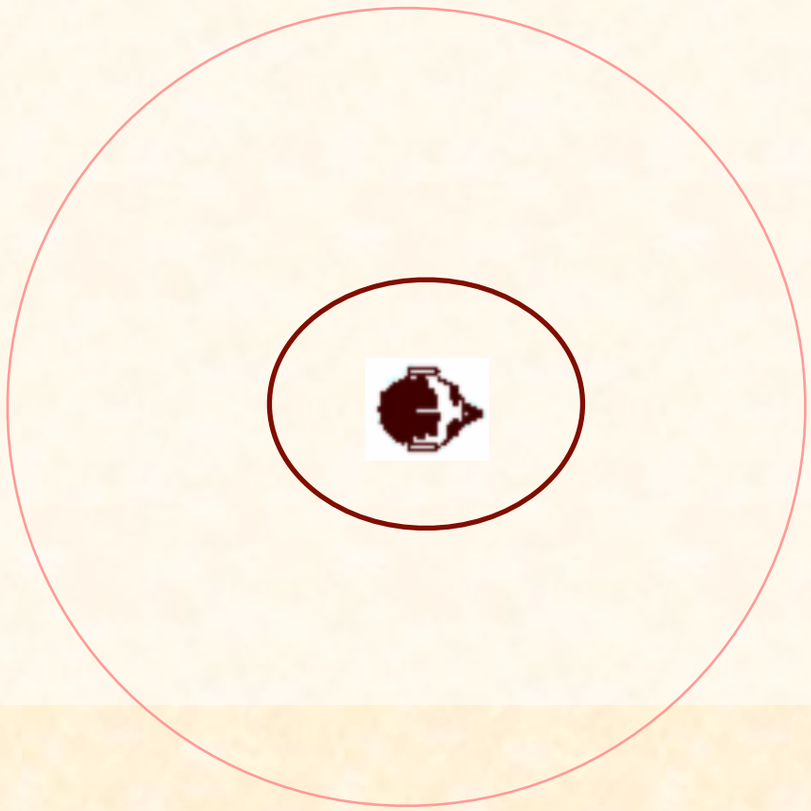


Predictive Interest Management

- Based on the spatial model of interaction
- Solves the following problem:

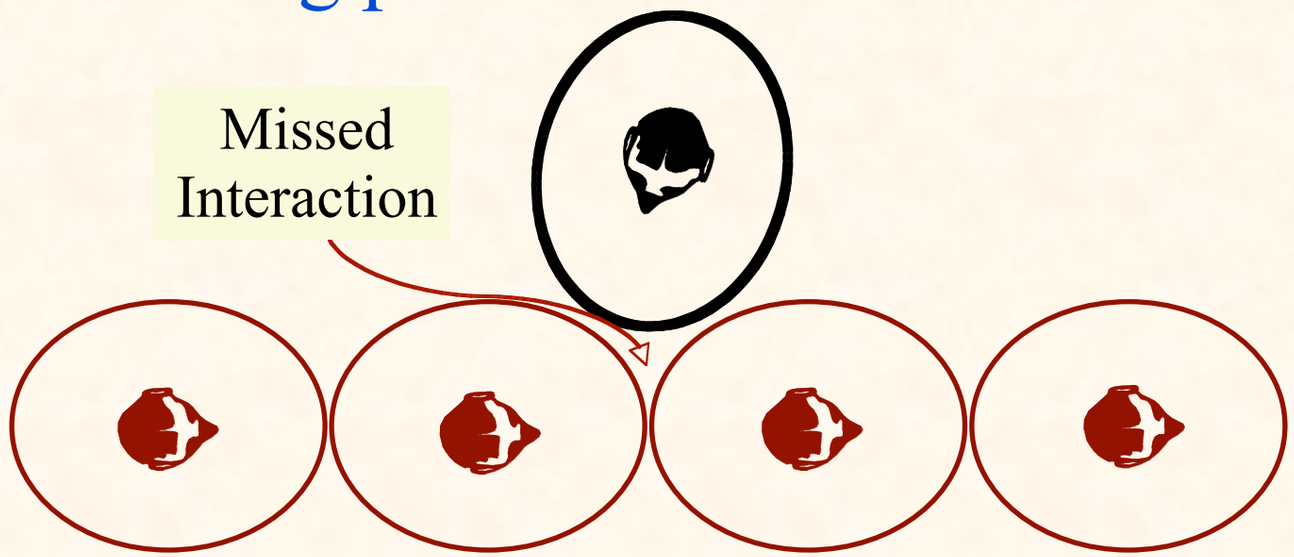


- Solution:

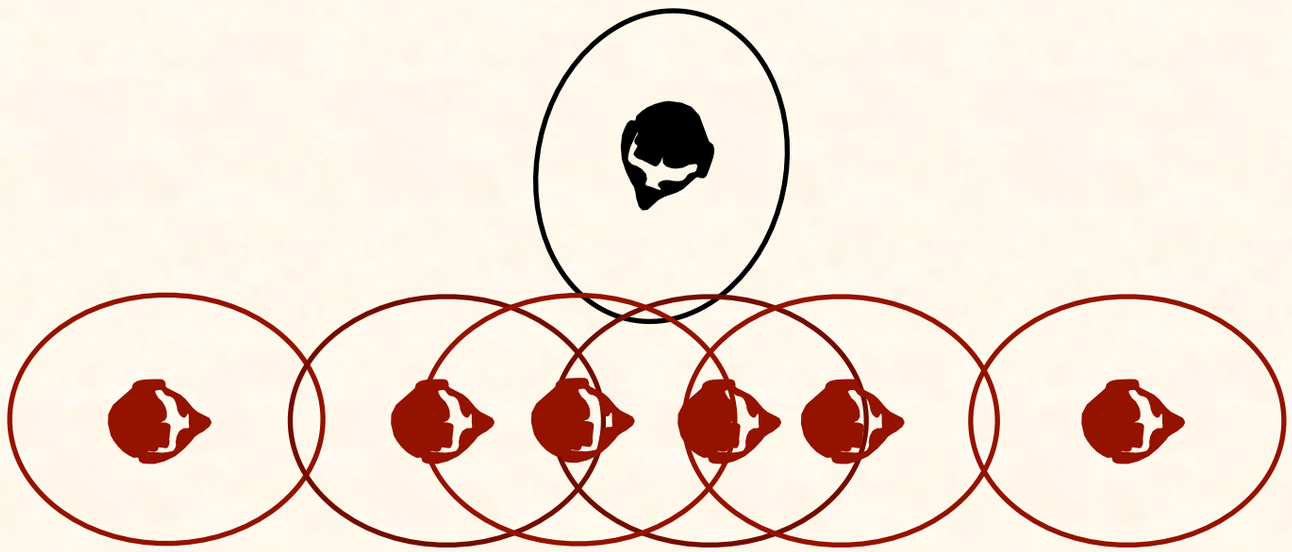


Predictive Interest Management

- Based on the spatial model of interaction
- Solves the following problem:



- Solution:

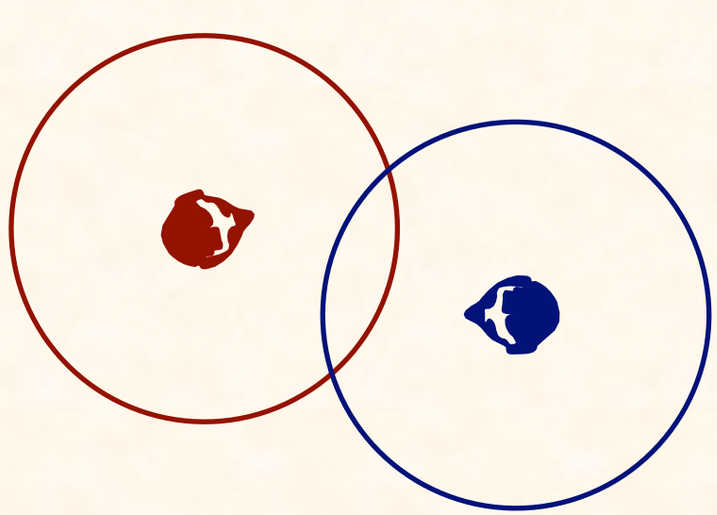


Extending spheres

- Based on the spatial model of interaction
- Adds a hierarchy to limit the number of collision detections

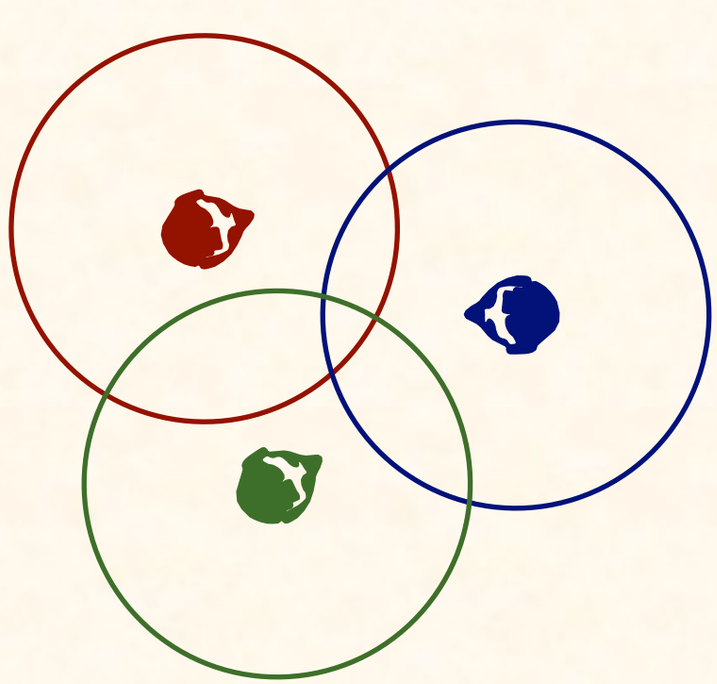
Extending spheres

- Based on the spatial model of interaction
- Adds a hierarchy to limit the number of collision detections



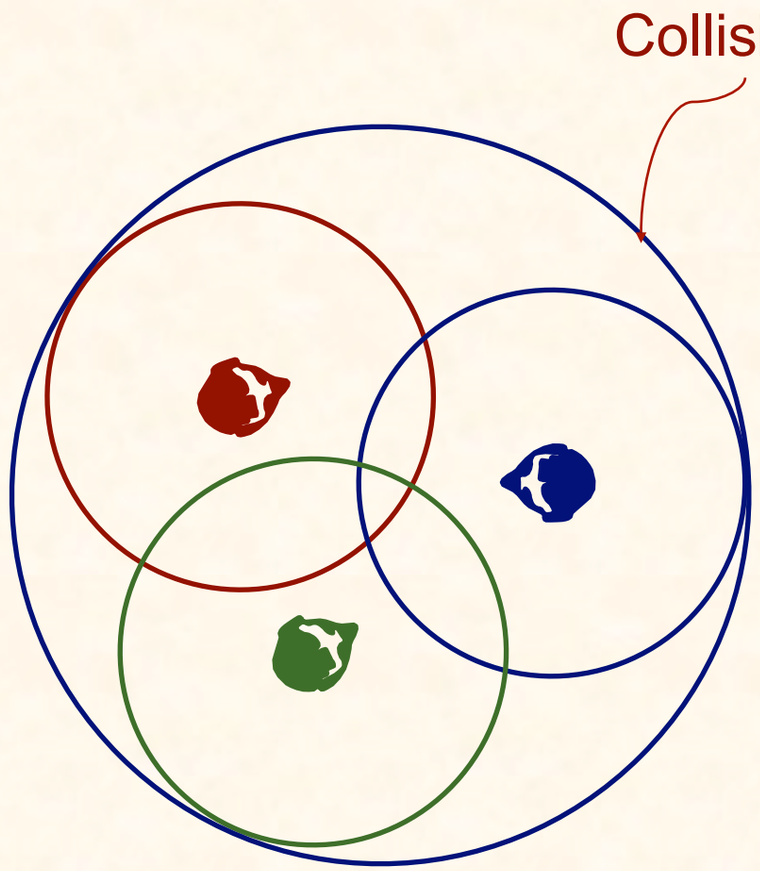
Extending spheres

- Based on the spatial model of interaction
- Adds a hierarchy to limit the number of collision detections



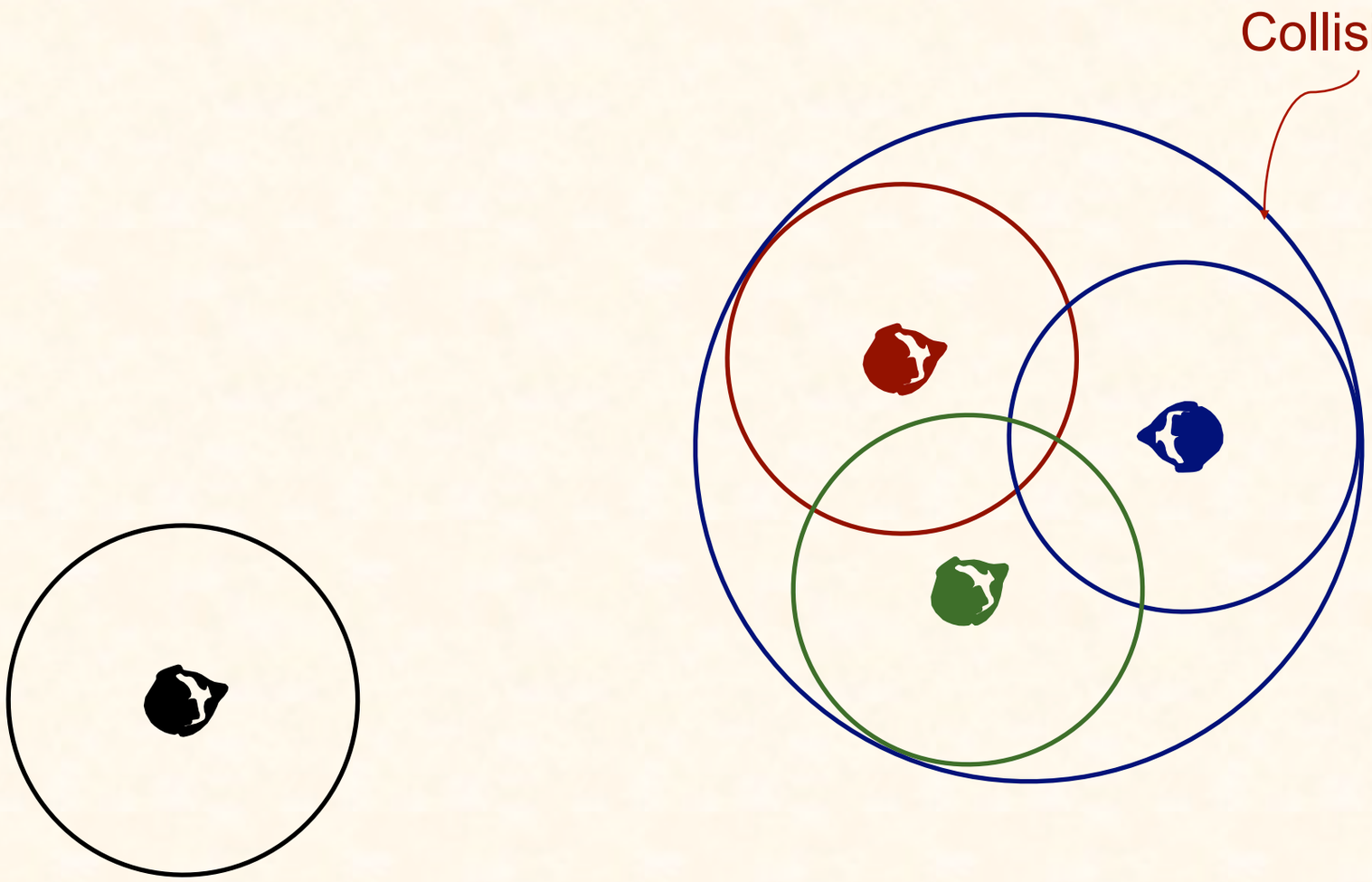
Extending spheres

- Based on the spatial model of interaction
- Adds a hierarchy to limit the number of collision detections



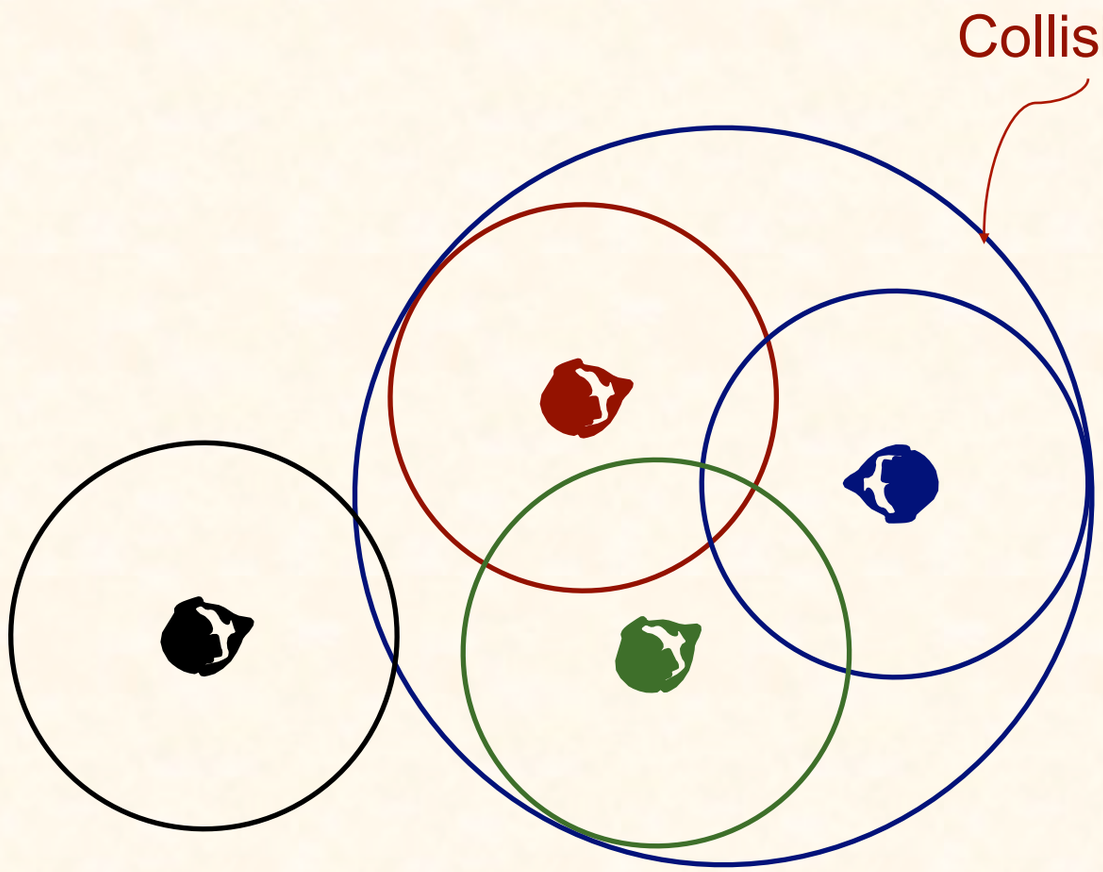
Extending spheres

- Based on the spatial model of interaction
- Adds a hierarchy to limit the number of collision detections



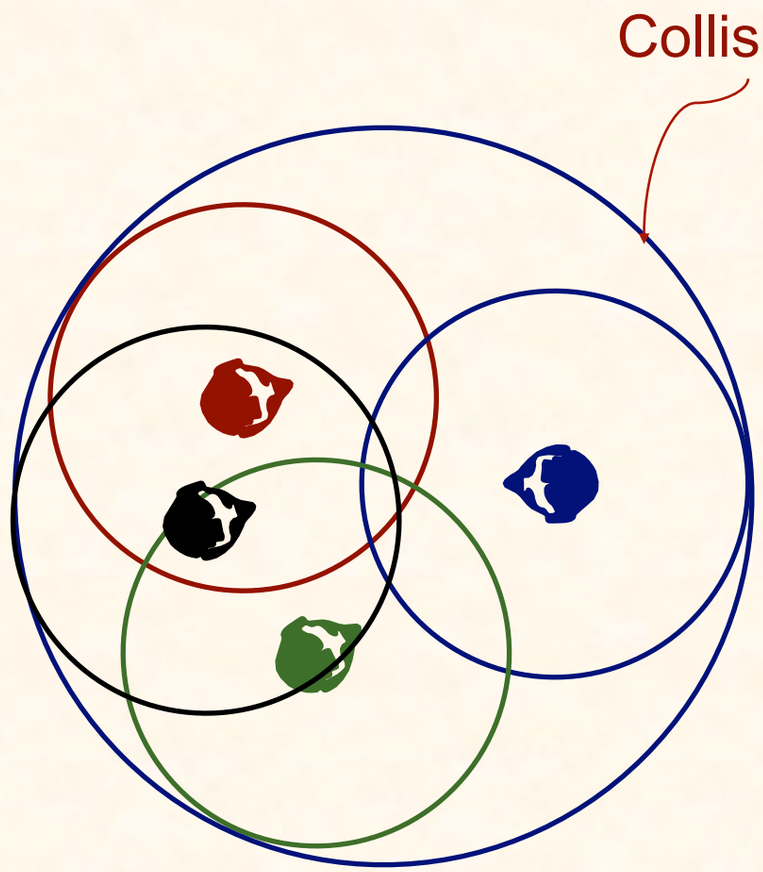
Extending spheres

- Based on the spatial model of interaction
- Adds a hierarchy to limit the number of collision detections

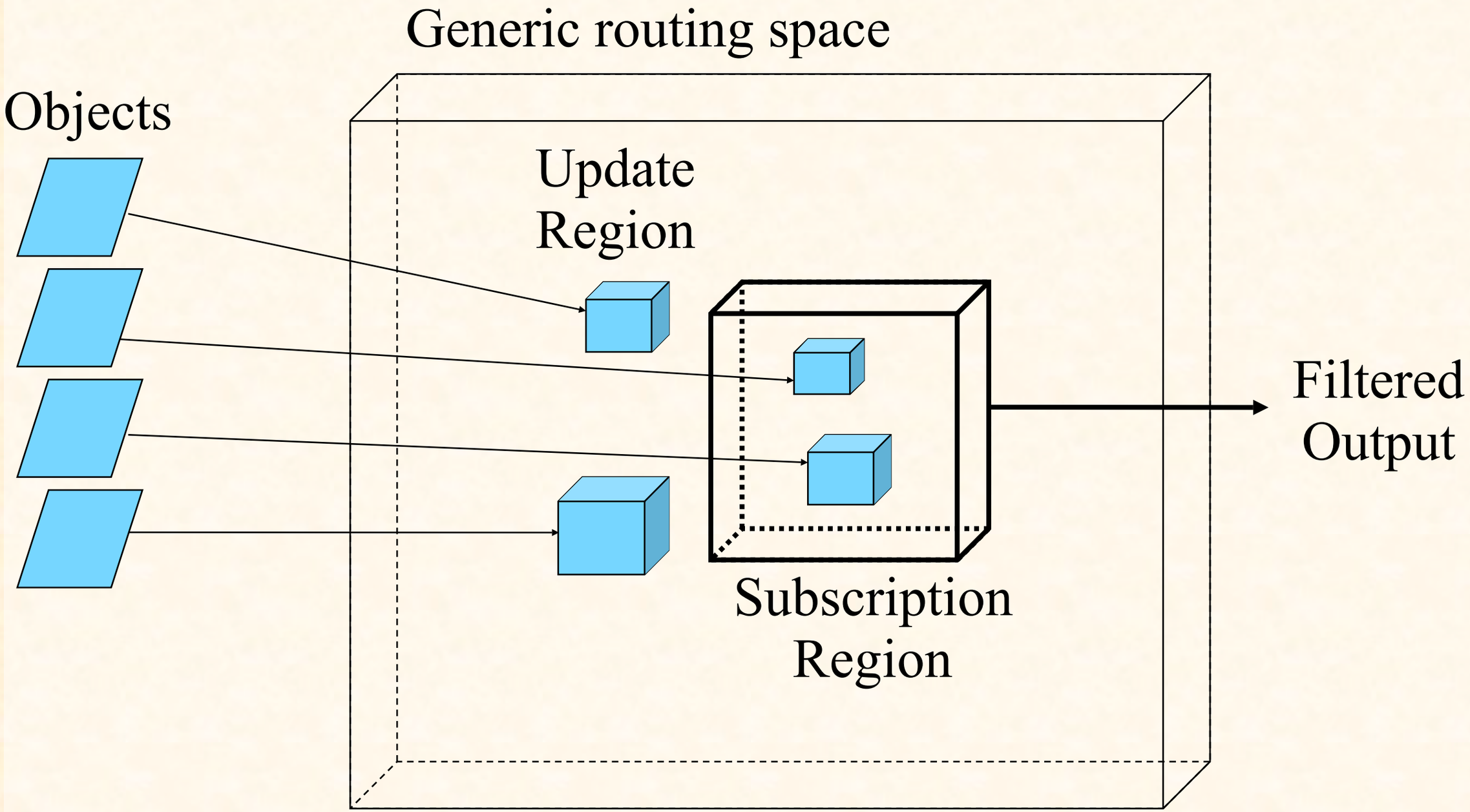


Extending spheres

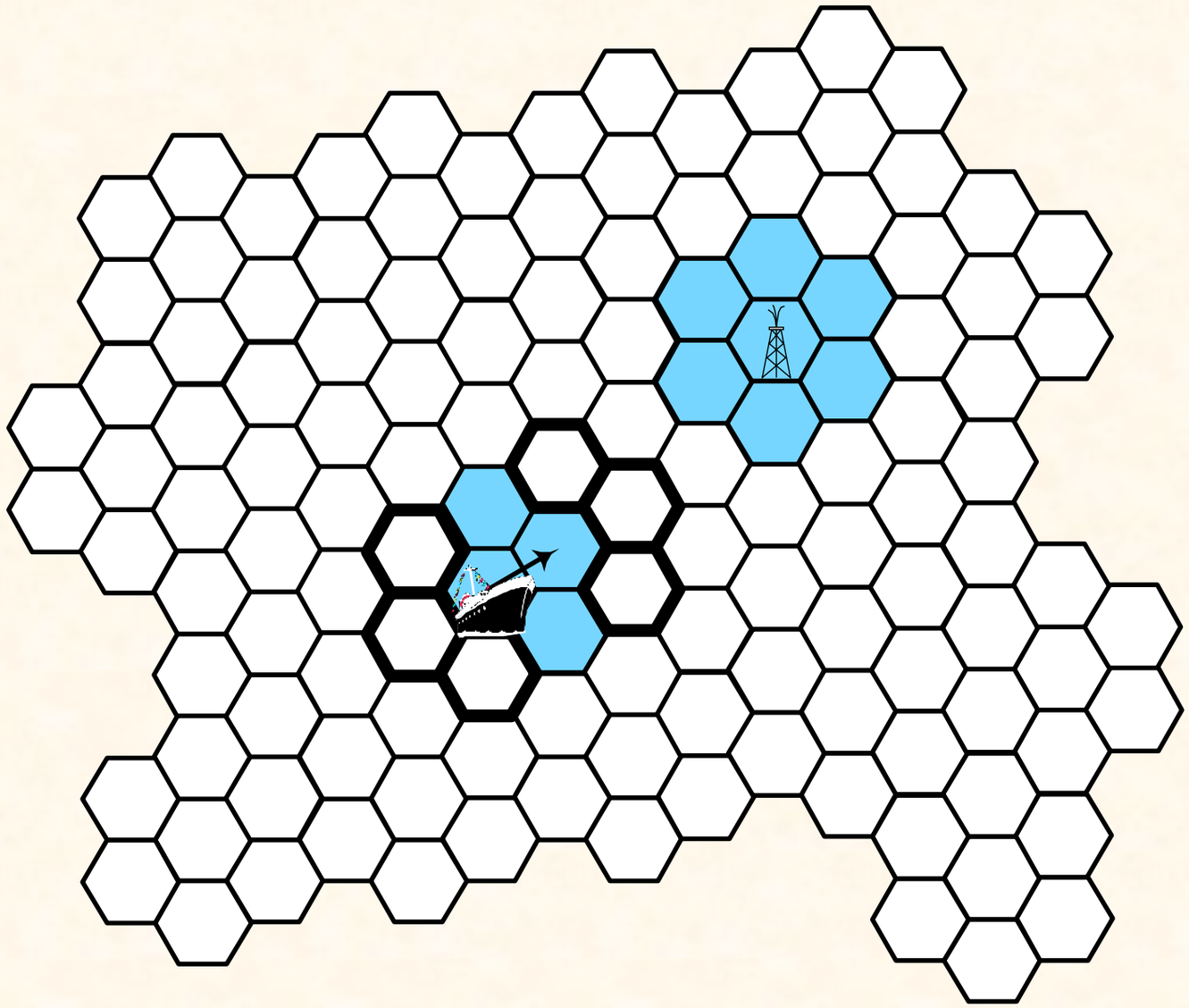
- Based on the spatial model of interaction
- Adds a hierarchy to limit the number of collision detections



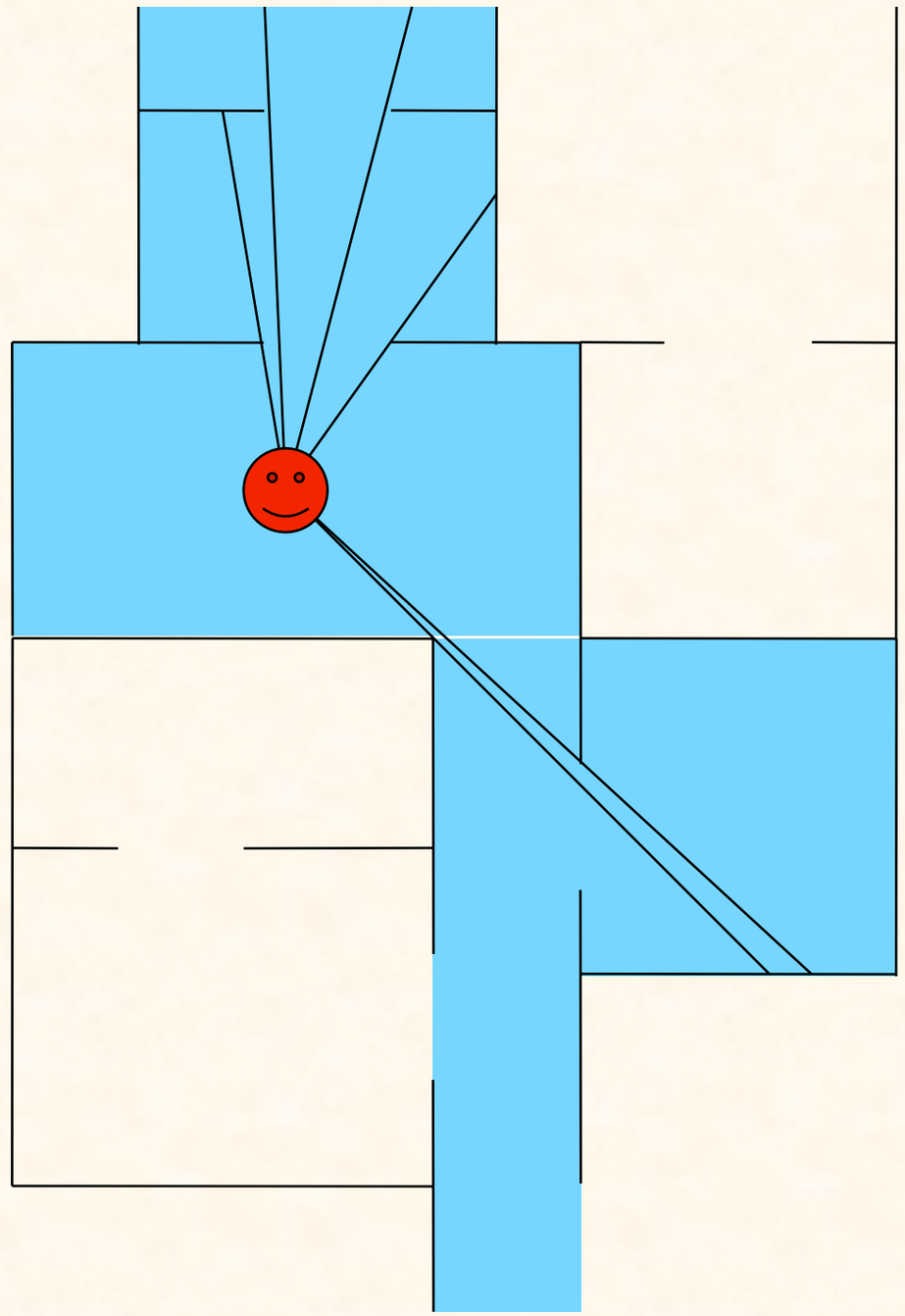
HLA routing spaces



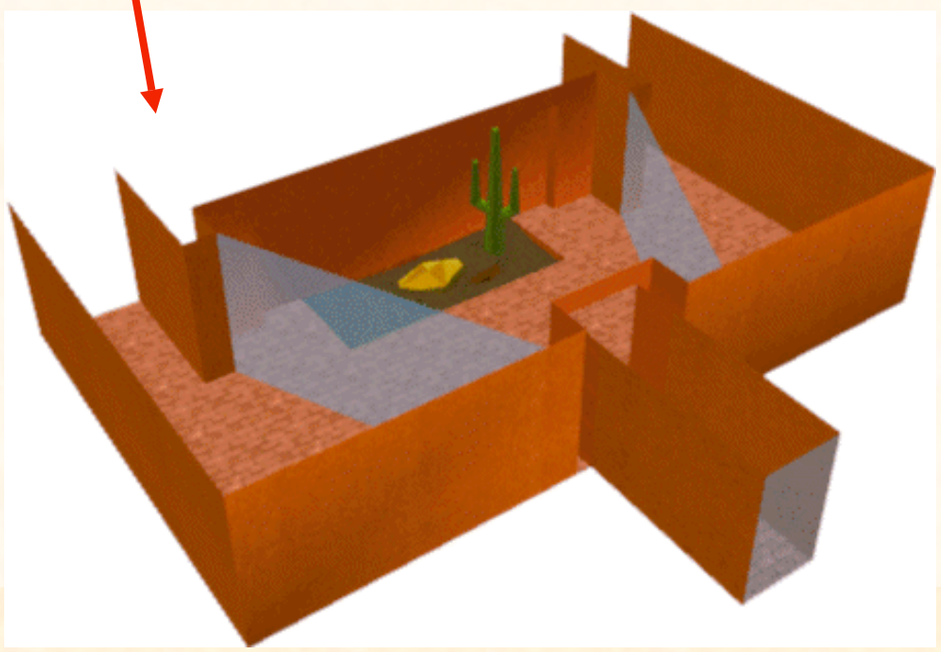
NPSNET's spatial filtering

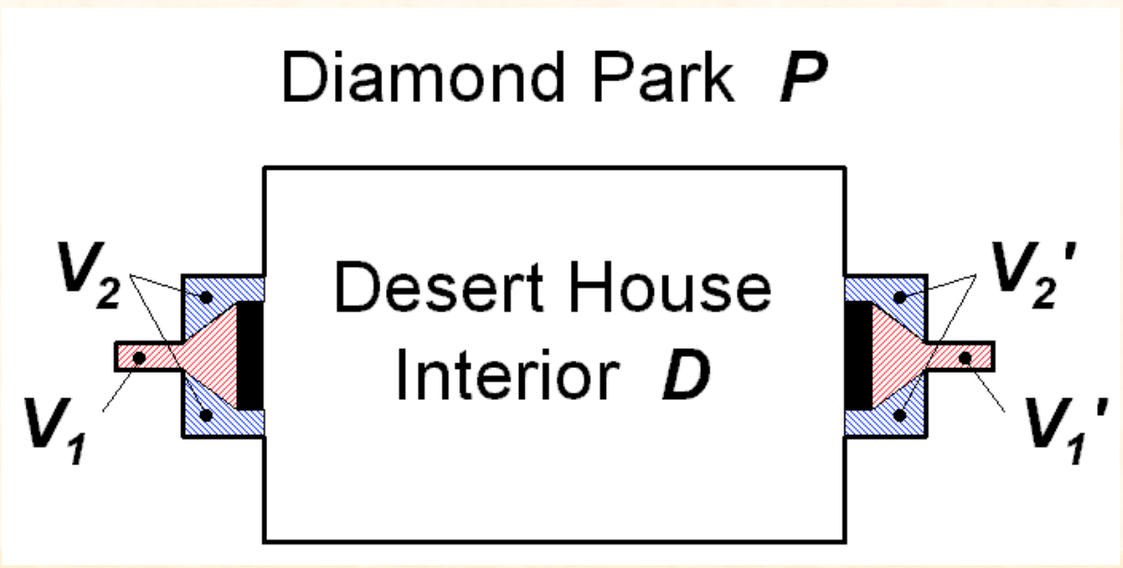


RING's spatial filtering

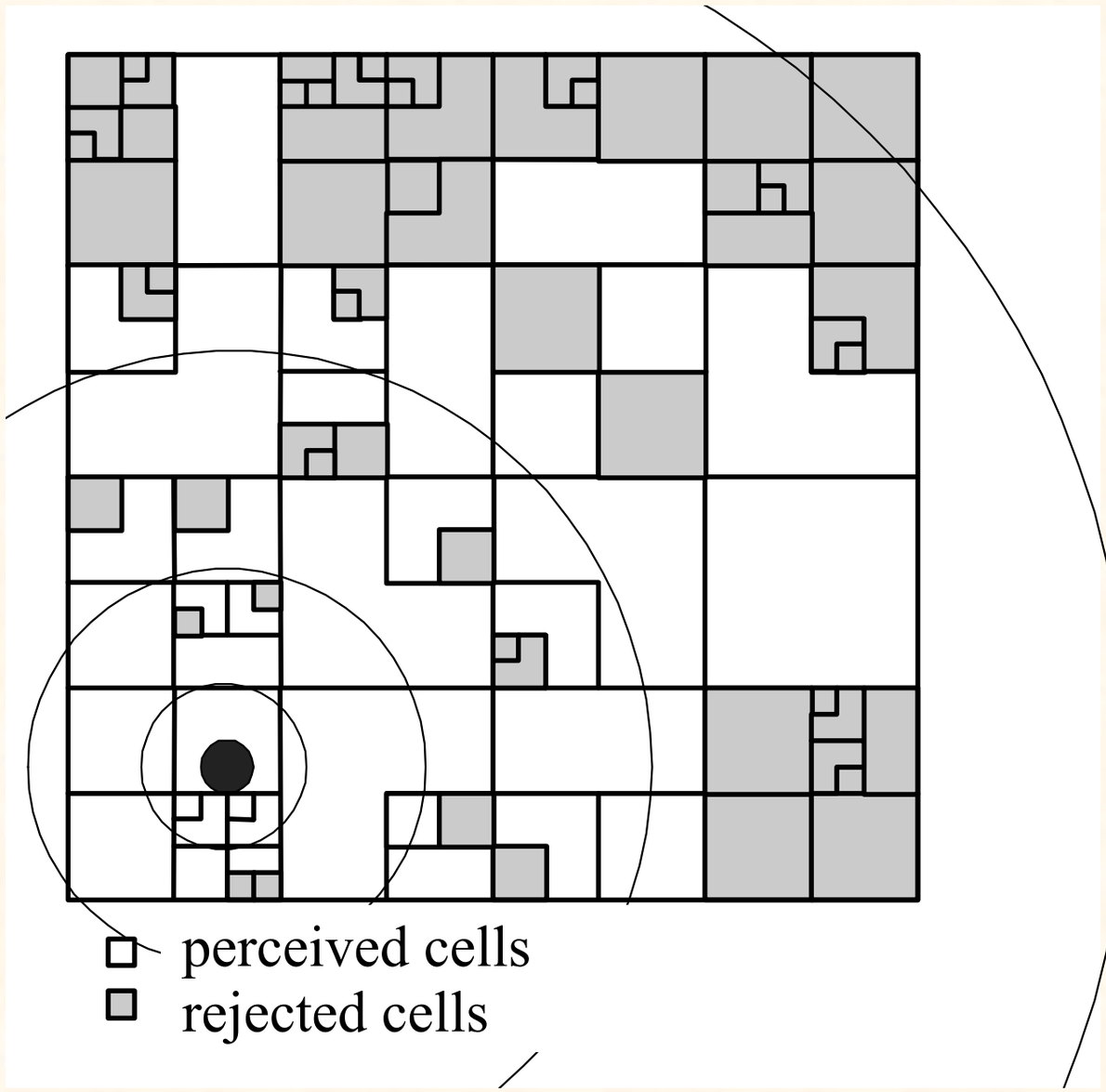


Spline's spatial filtering



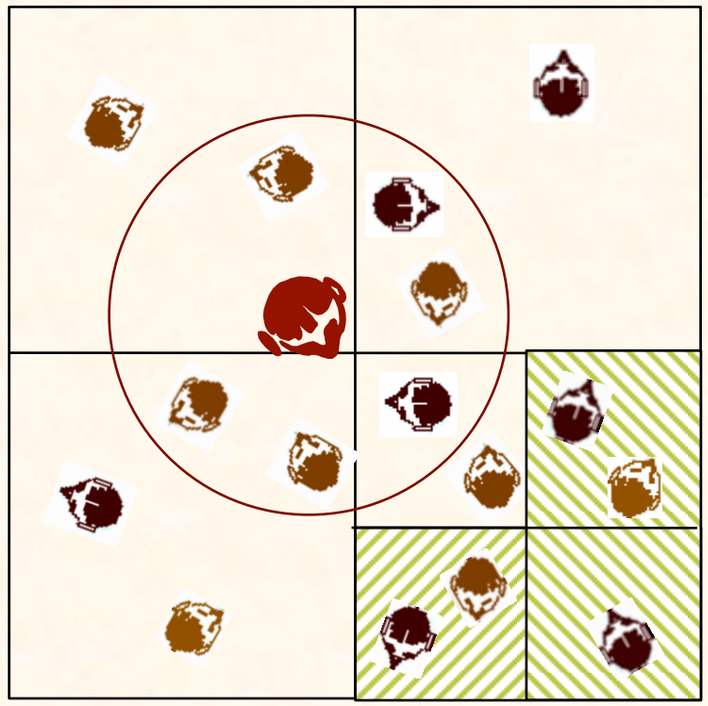


Space Scale Structure



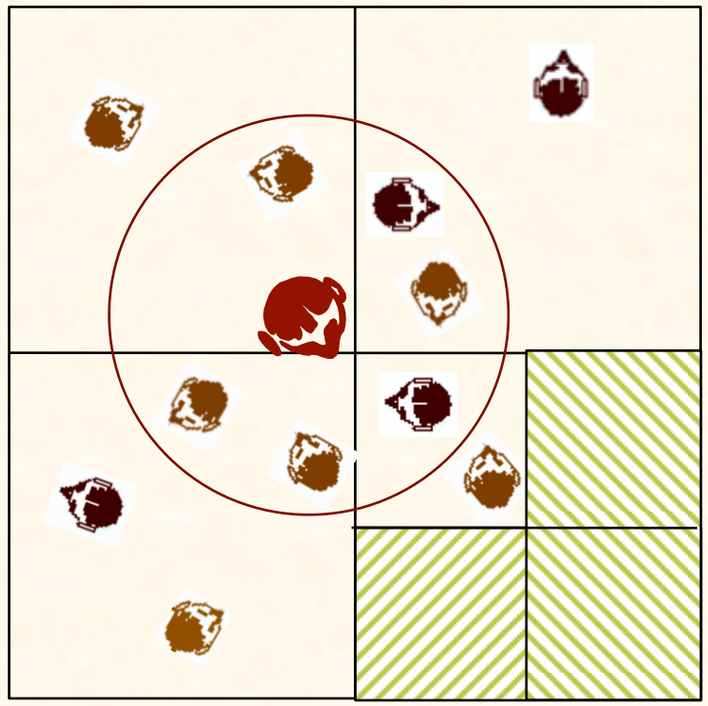
Three-tiered interest management

- First tier
  - Quadtree filtering



Three-tiered interest management

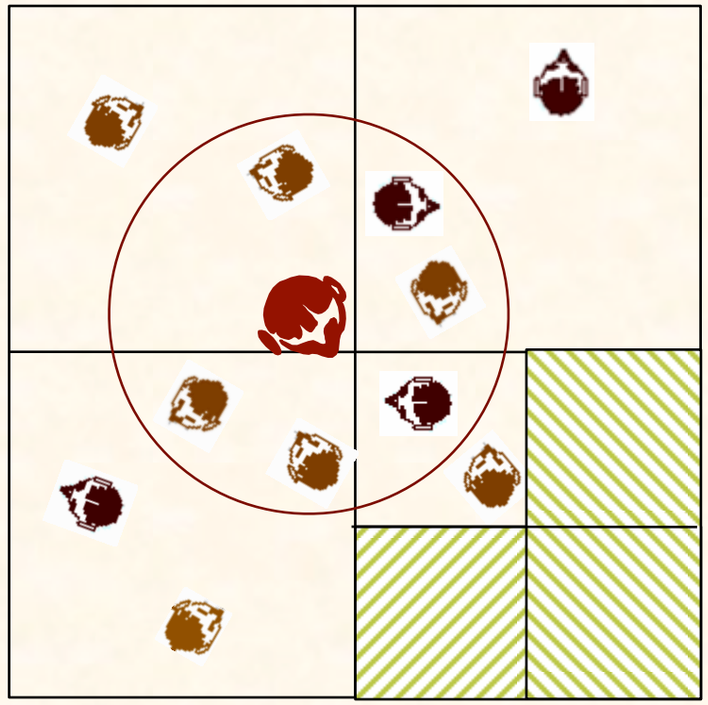
- First tier
  - Quadtree filtering



Three-tiered interest management

- First tier
  - Quadtree filtering

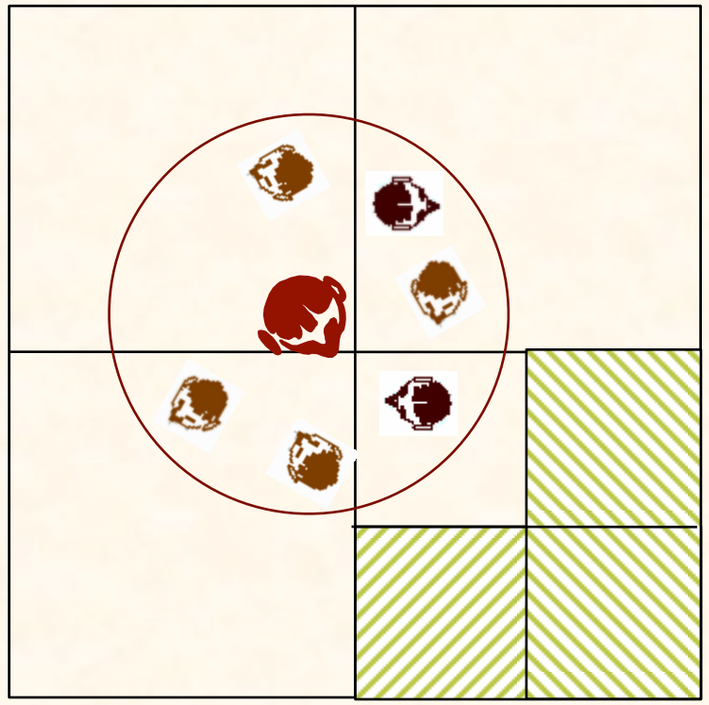
- Second tier
  - Auras filtering



Three-tiered interest management

- First tier
  - Quadtree filtering

- Second tier
  - Auras filtering

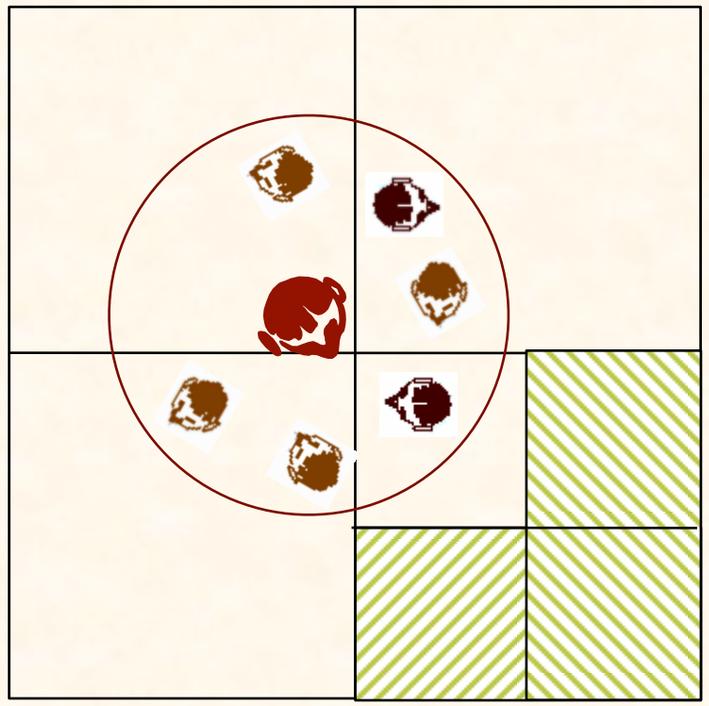


Three-tiered interest management

- First tier
  - Quadtree filtering

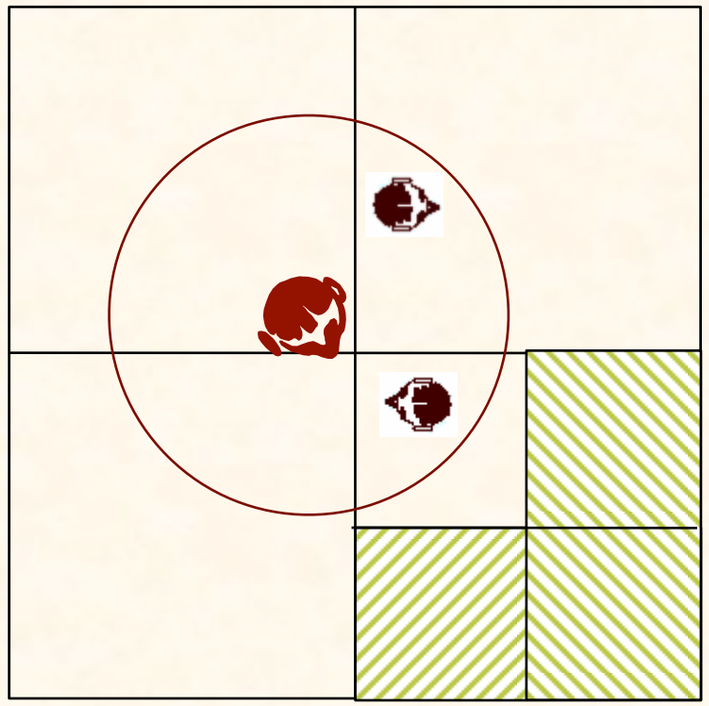
- Second tier
  - Auras filtering

- Third tier
  - Functional filtering



Three-tiered interest management

- First tier
  - Quadtree filtering
- Second tier
  - Auras filtering
- Third tier
  - Functional filtering



Networked games specifics

- RTS: Real Time Strategy
  - Synchronous model
  - Asynchronous model
- FPS: First Person Shooters
  - Standard Architecture
- MMORPG
  - “Shards”
  - Zones
  - Instances
  - Proxies
  - Seamless Architecture

RTS

- Player manages an army and civilians
- Creates buildings and units
- Gives orders to units
- Visibility is very important (“Fog of war”)
- Often limited interactivity
- Typical phases of play:
  - Building
  - Exploration
  - Combat

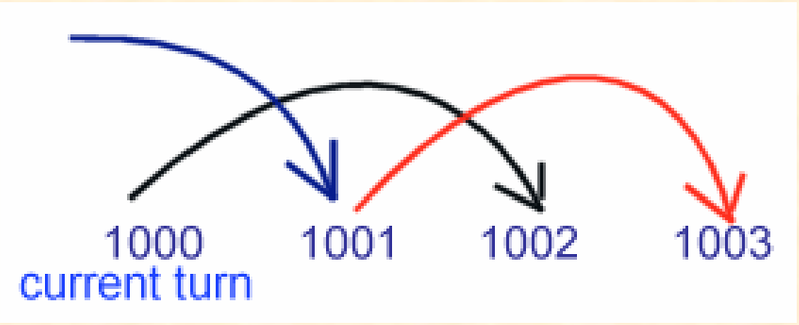
Synchronous RTS

- Problem: how can you send updates for thousands of units with a 56K modem connection
  - E.g. a 2D position (x,y -16 bits), an orientation (8 bits) and a unit id (16 bits) => 5*8 = 40 bits
  - Limit: 56000 / 40 = 1400 units moving simultaneously once per second
- Solution: identical world simulation running on each computer. Network is only used to send player orders (e.g. mouse events, keyboard events, graphical menu selections or orders given to a group of units...)

Synchronous RTS

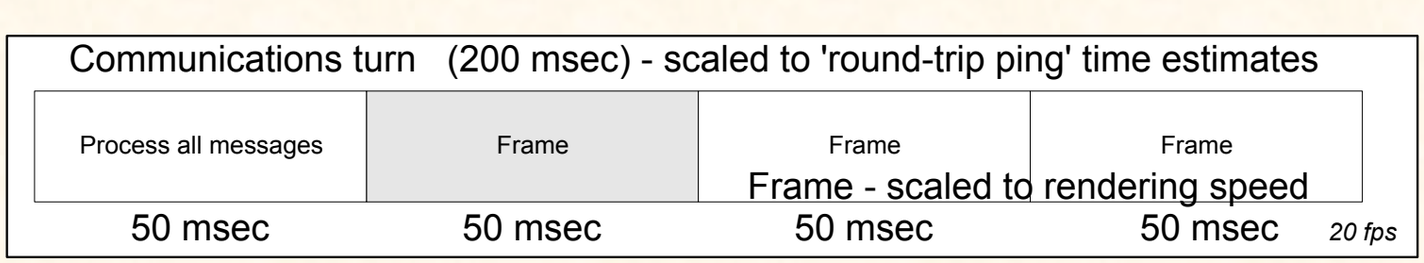
- Example: Age of Empire 1 and 2
  - Computes the length of a game turn taking into account of the perceived latency (round trip time / 2)
  - A game turn is composed of several smaller simulation frames
  - In each game turn
    - ✓For each player action => send a network message
    - ✓Receive other players actions
      - Stored in a queue
    - ✓At the end of turn, send an “end of turn” message including time data in order to regulate the turn length if necessary
    - ✓Then execute actions in the same order on each computer (action1 of player 1, action 1 of player 2, action 2 of player 1...)
    - ✓Then execute N simulation frames (N depends on the turn duration)

Synchronous RTS

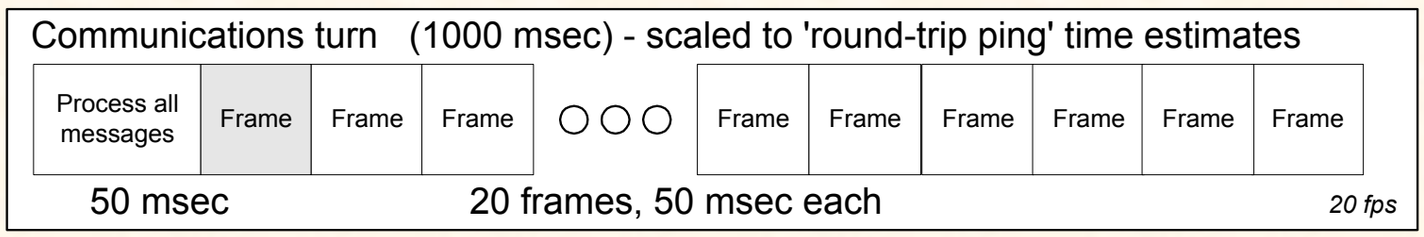


- Example: Age of Empire 1 and 2
  - When latency is too big, send actions that will be executed in 1 or 2 game turns
  - Game turn duration depends on latency

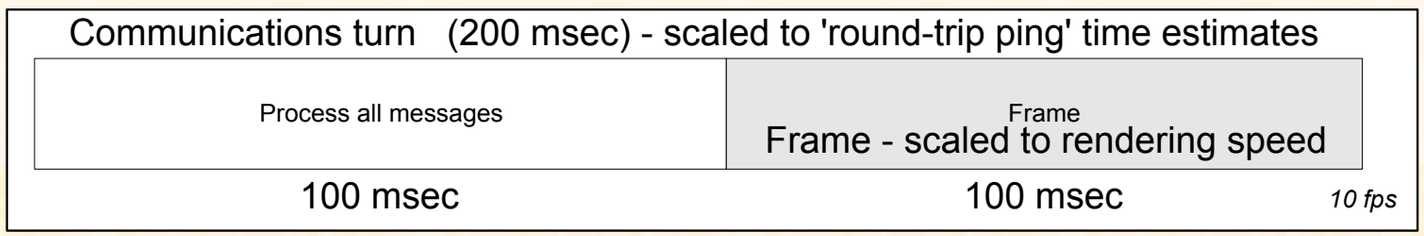
A Single Communication Turn



High Internet Latency with normal machine performance



Poor machine performance with normal latency



Synchronous RTS

- Advantages
  - Even with a very small bandwidth you can manage a huge number of units
  - Example of bandwidth usage: Warcraft III uses 7 kbps max
- Disadvantages
  - Need to wait for all players messages before executing their actions. With huge latencies, the game can stall. It's not good for scalability (in term of number of players)
  - Need to have a totally deterministic simulation. Random generators need to be synchronized and you need to make exactly the same number of calls on each computer. Very hard to code.
  - It is impossible to join the game after it is started
  - There are lots of cheats: map revelation...

Asynchronous RTS

- Idea: a server manages the complete simulation and only sends updates for the part of the map that is visible to each player
- Advantages:
  - A lot easier to code (uses filtering techniques)
  - No more “map revelation” cheats
  - Players can be added after game start
- Disadvantages:
  - Used bandwidth increases a lot during the game. The more you explore with a lot of units the more space you see (and in some games you can even use radars...)
  - Game turns can be very long in order to wait for all updates.

FPS

- Player controls one character
- Collects weapons, armors and bonus objects
- Kills other player characters (or “bots”)
- Very interactive game: fast paced action

FPS: standard architecture

- A server simulates the world
- It receives moves/shoots orders from each client (player or bot)
- It computes actions on the world
- It sends back results to all clients
- In order to manage big latencies it often uses “dead reckoning” techniques
- Advantage:
  - Very interactive (using dead-reckoning)
- Disadvantage:
  - Scalability is limited (a MMOFPS requires a lot of servers with a small latency with each client)

MMORPG

- Each player controls one character
- Explores the world
- Chats with other players (near ones or distant ones - guilds)
- During combats, the player activate skills/spells/special attacks which all have a limit on the number of time you can use them in a given period. The rest of the time, the character auto-attacks.
- Characters gain new skills over time.
- Interactivity is smaller than for a FPS but bigger than for a RTS
- Biggest interest: several thousands of players share the same world

MMORPG: "shards"

- It is often impossible to allow every subscribers to connect to the same server.
- To solve this problem, Ultima Online introduced the "shard" concept.
- Each "shard" is an identical copy of the game world (at game start). Each "shard" then evolves independently.
- A player must choose in which "shard" he will play when he creates his character (some MMOGs have character transfer paying services)
- The term comes from the fact that in one of the Ultima games, the player discovered that the universe was in fact trapped in several "shards" of a magical crystal.

MMORPG: "shards"

- Advantages
  - You can add "shards" when the number of subscribers increases and remove them (transferring remaining characters) when it decreases.
  - You can setup "shards" in several real world locations in order to have a smaller latency with clients.
- Disadvantage
  - Players are not in the same shared world, which limits interactions with other players

- Note: some MMOs like EVE Online do not use "shards" but limit interactivity to allow this.

MMORPG: zones

- In order to manage more users in the same “shard” Everquest introduced the zone concept in 1999.
- Each zone is connected to one or several other zones through “teleporters” which allow a character to change zones.
- A player only sees and interacts with players in the same zone.
- Generally, each zone is managed by a different process.
- The whole process set is then distributed statically or dynamically on a set of server computers (gathered in a cluster).
- Optimizations allow all zones on the same computer to be managed by the same process (in order to avoid context switchings).

MMORPG: zones

- Advantages
  - The virtual world CPU usage is distributed on several machines.
  - You can manage more players in the same “shard”.
- Disadvantages
  - Zones create artificial divisions of the virtual world.
  - One zone can become very popular and saturate the computer that manages it.
  - It is often difficult to divide the world in zones “a priori” in order to insure a good load balancing.

MMORPG: instances

- The instance (or instanced zone) concept has been introduced by Everquest in 2003.
- It consists in reserving a zone to a small group of players and duplicate it for each group (there are therefore several instances of the same zone).
- Instances are managed in the same way as zones but require far less computer resources.
- A single computer can manage a lot of instances.
- Moreover, as instances have a limited duration (at most a few hours), the cluster can load balance using instances. New instances will be created on the computer that has the lowest CPU load.

MMORPG: instances

- Advantages
  - Instances allow a fine balancing of the virtual world CPU load.
  - It is easy to predict the CPU load needed for an instance because it only manages a small number of players.
- Disadvantage
  - There are no real disadvantages except that duplication isn't very realistic.

- Note: instances can have long durations (they are then called raids) but this is more of a database issue.

MMORPG: proxies

- In order to avoid a client to interact with all the servers in a MMO cluster, games use proxy servers.
- When a client first connects to the game (and after an authentication step managed by specific servers) a proxy is chosen for this client. This proxy will stay the same during all the playing session (until client disconnects).

- Advantage
  - Client doesn't need to disconnect/connect to several computers when the character change zones, instances...
  - Proxies hide other servers that are thus less exposed to hacking attempts (proxies are in the DMZ)
- Disadvantage
  - Proxy crossing adds a little latency

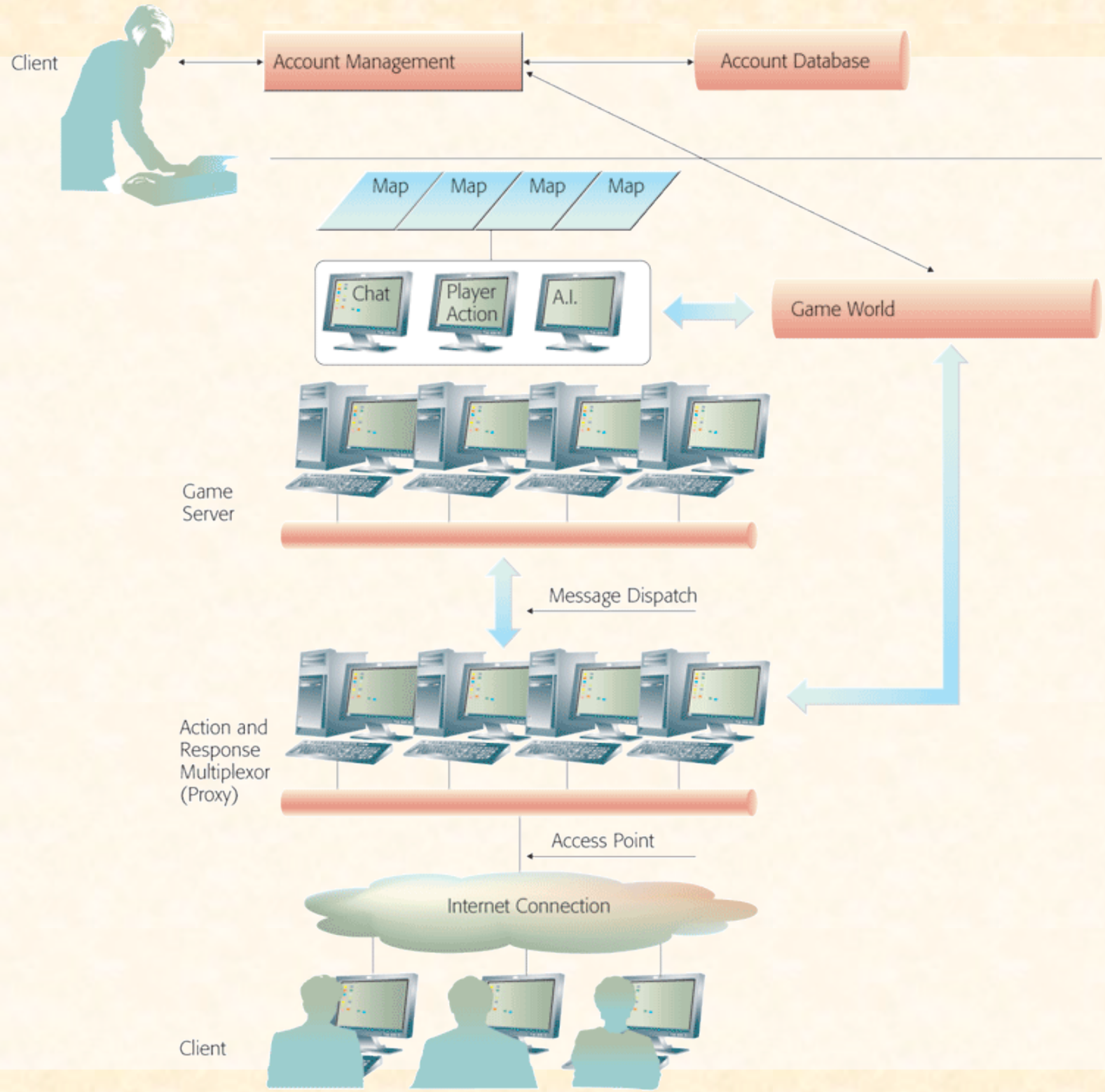
MMORPG: seamless architecture

- The seamless architecture (i.e. with no apparent border between zones) has been introduced by World of Warcraft in 2004.
- It consists in managing all the virtual world (or a big part of it) in only one big zone that is managed by several computers.
- The zone is divided in regions, dynamically allocated to several computers.
- If a region becomes too popular it is divided again and some sub-regions are allocated to other computers.

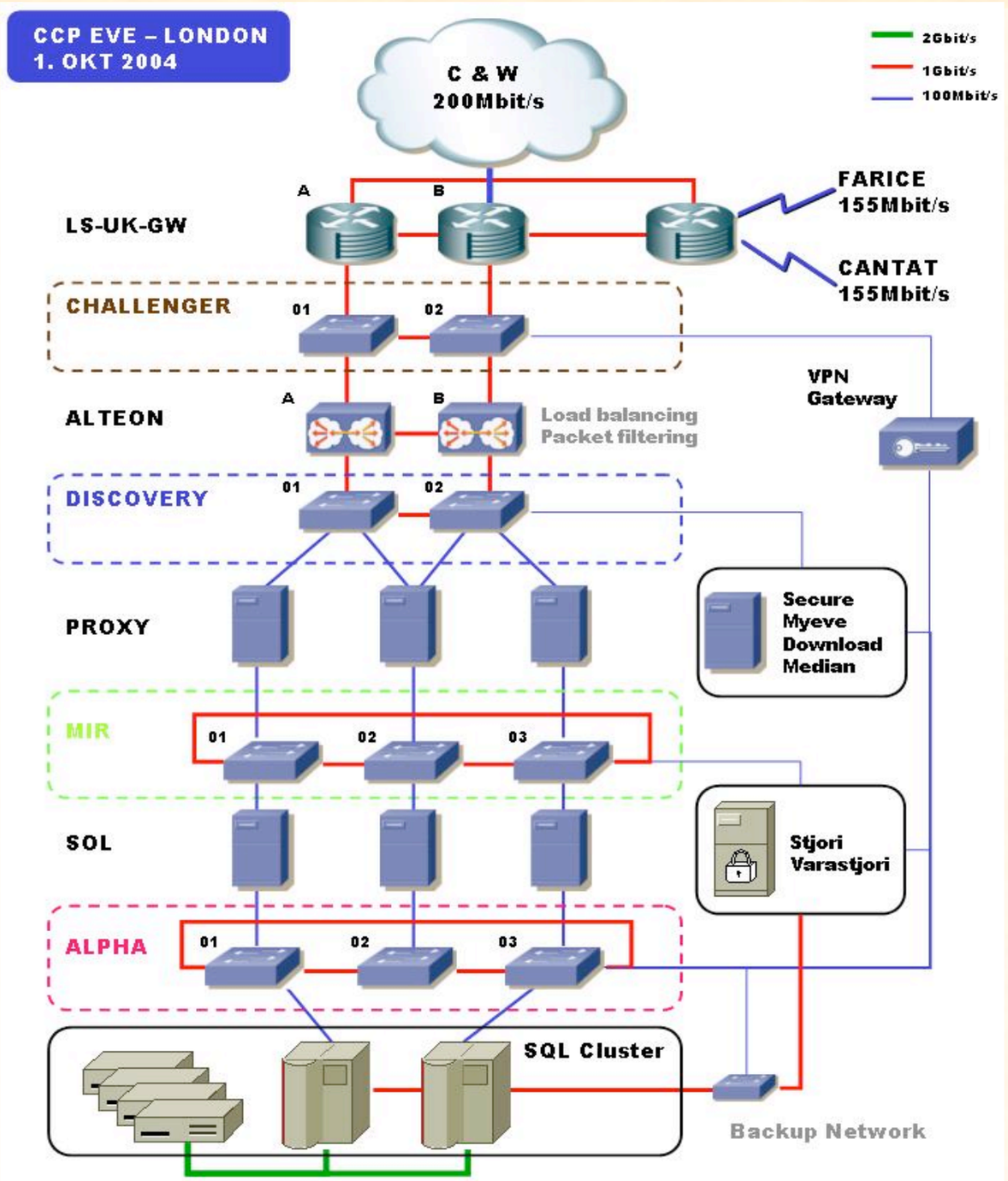
MMORPG: seamless architecture

- Advantage
  - The world is more realistic as there are no artificial divisions.
- Disadvantages
  - This kind of architecture is way more difficult to manage and code.
  - Object exchange between characters in different regions is very complex to code in order to avoid duplications if one of the computers (managing one region) crashes.

MMO architecture example



MMO architecture example: EVE Online



- 400 GHz CPU / 200 Gb RAM
- 2 Routers (CISCO Alteon)
- 14 Proxy servers (IBM Blade)
- 55 Sol (zones) servers (IBM x335)
- 2 database servers (in a cluster, IBM Brick x445)
- Windows 2000, MS SQL Server
- Upgrade:
  - AMD x64 IBM Blade

Thanks

- This keynote is based in part on:
  - Michael Zyda courses (Gamepipe laboratory)
  - "Networked Virtual Environments - Design and Implementation" book, Sandeep Singhal & Michael Zyda, ACM Press 1999.
  - Networked virtual environments course at SIGGRAPH'99
  - Nicolas Farcet's PhD defense keynote
  - Several research papers...