Massively multi-player games and Project Darkstar

Who am I?

- Jeff Kesselman, Chief Instigator of Project Darkstar, Sun Microsystems Laboratories
  - 15 years in games and multi-media before coming to Sun:
    - American Interactive Media (Phillips)
    - Crystal Dynamics
    - Total Entertainment Network (TEN)
  - 9 years at Sun
    - Win32 Java 1.3 Performance Tuning
    - Initial leader of the JInput project
    - 2 yrs in Sun “Game Technologies Group”
    - 2.5 years at Sun Labs (Project Darkstar)

Goals For The Week

This week we will cover:

- The History and Structure of Multiplayer games
- The technical game-play challenges going online brings
- How the Project Darkstar server is designed to ease the impact of some of those challenges

What is Project Darkstar?

- Project Darkstar is a network application container designed specifically for mainstream online games.
  - Project Darkstar customers are game developers.
  - Project Darkstar applications are games or game-like applications
- More details to follow...

Lecture Map

Day One: History of Multiplayer	Day 2: MUDs, MMOs and Darkstar	Day 3: Project Darkstar	Day 4: Project Darkstar and Chess
Evolution of the Game	Evolution of the MMO	Comparative architecture: Traditional v. PD	Details of Darkstar Coding Do's and Don'ts
Multi-player Architectures	The Motivation for Project Darkstar	The Project Darkstar Coding Model	Chess: Designing a PD based server

Topics Not Covered

- These lectures are intended to familiarize you with the theory behind writing massively multi-player games and the theory and design behind the Project Darkstar server. They do not cover:
  - > Installation and operations of a Project Darkstar (PD) back-end.
  - > Language syntax and APIs
  - > For these and other specifics of coding PD based games, see the PD tutorials included in the downloads.

Unit One:

History of Multi-player

What this lecture is about

The Evolutionary History of the Architecture of Online Massively Multi-player games

Lecture Overview, Day One

- Day One, Lecture
  - > Evolution of Games
  - > Review: Single-player game structure
  - > Multi-player game structure
  - > MUDs and MMOs

Where game architecture comes from

- Game software has DNA
  - It carries the history of the industry within it
  - In order to understand current games, you need to understand the history

Where game architecture comes from

- Game software has DNA
  - It carries the history of the industry within it
  - In order to understand current games, you need to understand the history
- Game software usually evolves incrementally
  - Game development is generally risk adverse
  - Game development is on tight schedules
  - Games general vary only in minor way from what came before

Where game architecture comes from

- Game software has DNA
  - It carries the history of the industry within it
  - In order to understand current games, you need to understand the history
- Game software usually evolves incrementally
  - Game development is generally risk adverse
  - Game development is on tight schedules
  - Games general vary only in minor way from what came before
- Leaps happen rarely but occasionally
  - Usually by 'cross-breeding' unrelated software

Single Player Game Architecture

The Game Loop, A review

Start at the beginning

- The primordial ooze of games
  - BASIC “guess the number”

```
10 N = INT(RND(1)*100 + 1)
20 PRINT “Guess a number between 1 and 100”
30 INPUT G
40 IF G = N GOTO 100
50 IF G < N GOTO 80
60 PRINT “Too high”
70 GOTO 20
80 PRINT “Too low”
90 GOTO 20
100 PRINT “You got it!”
110 END
```

Contains all the “organs” of a modern game

- “The Game Loop”
  - > Initialization

```
10 N = INT(RND(1)*100 + 1)
```

  - > Update/Render loop

```
20 PRINT “Guess a number between 1 and 100”
30 INPUT G
40 IF G = N GOTO 100
50 IF G < N GOTO 80
60 PRINT “Too high”
70 GOTO 20
80 PRINT “Too low”
90 GOTO 20
100 PRINT “You got it!”
```

  - > Intermingled because simple BASIC isn't structured

All games have a game loop

- Turn Based
  - Stop in Update to collect all input
- Example:
  - Chess:
    - Update:
      - input chess move
      - Run Artifical Intelligence (AI) to calculate response
    - Render:
      - Re-draw or animate chess board

All games have a game loop

- Real Time
  - Poll inputs in Update and go on
- Example:
  - First Person Shooter (FPS)
    - Update:
      - Every N frames (or time ticks)
        - Read input keys
        - Calculate player fire if any
        - Run AI to calculate response
        - Calculate Mobile Object (MOB) fire if any
        - Move Player
        - Move MOBs
    - Render:
      - Animate 1 frame (or N ticks) of gunfire and motion

Differences Btw Turn based and Real time

- Turn based
  - Blocking input
  - One trip around the loop == 1 game turn
- Real Time
  - Polled input
  - One trip around the loop == fraction of game turn

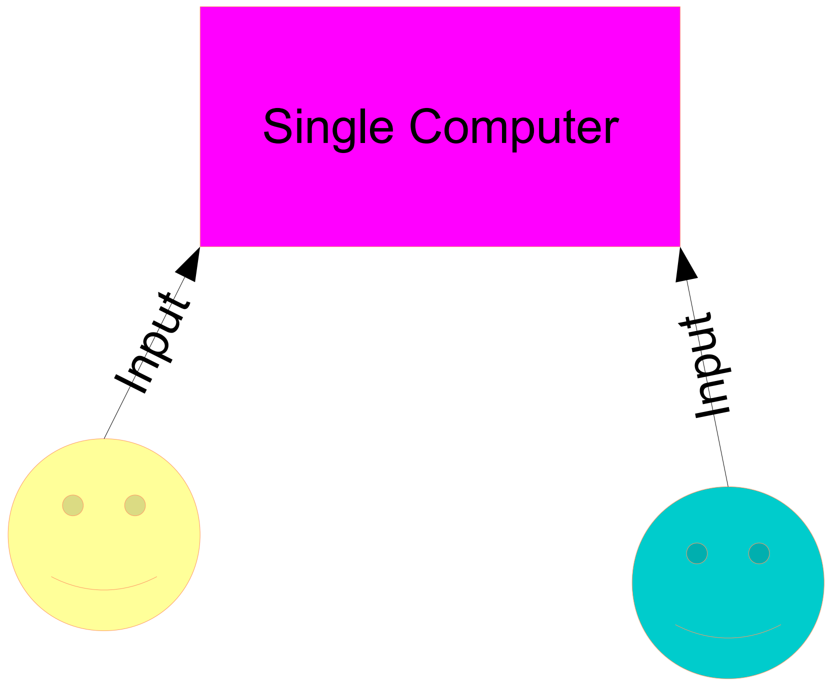
- "Game Turn" above is defined as one read of the controllers and the calculation and animation of the response.

Multi-player games

An evolutionary line

Multi-Player, the next evolution

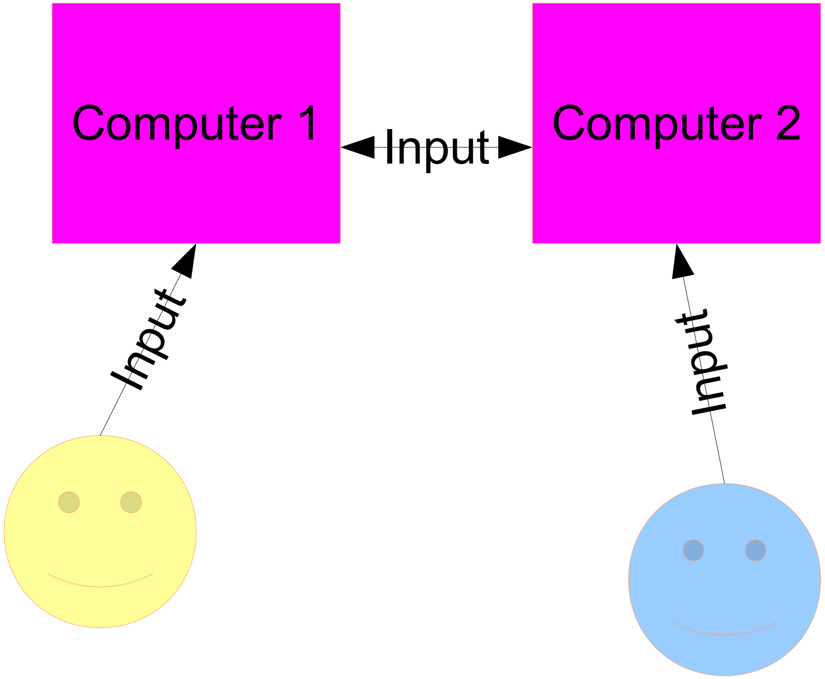
- Multiple Players on one computer
- Turn Based
  - > Players each enter their own move sequentially in Update
- Real Time
  - > Each player has their own set of keys or input device
  - > All players are polled in Update



Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Revision A

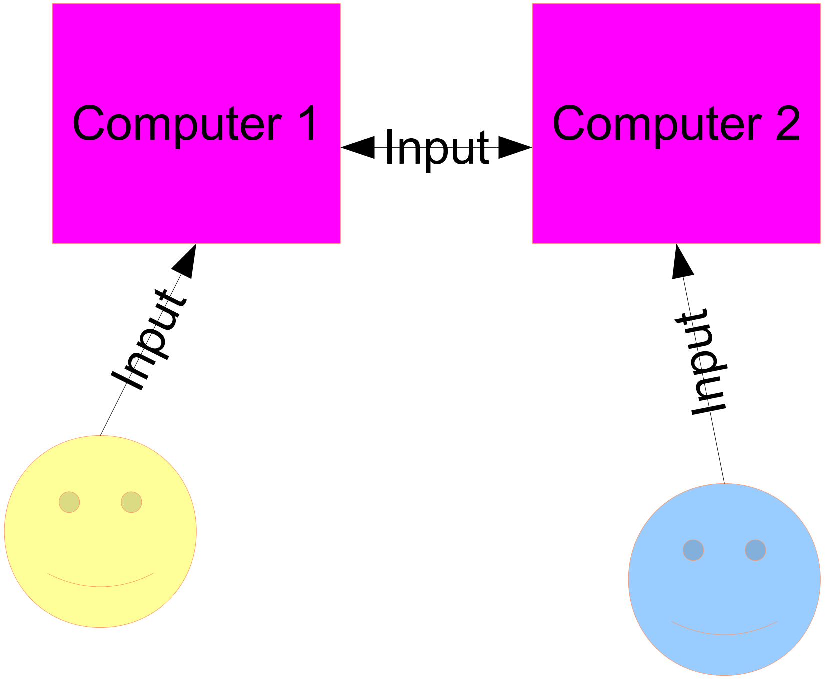
Multi-Station, the first networked games

- Played on LANs
- Non-local players are on virtual devices
  - Other players input happens on foreign machines
  - Is communicated over network
  - Is processed in Update at every machine as if all input was local



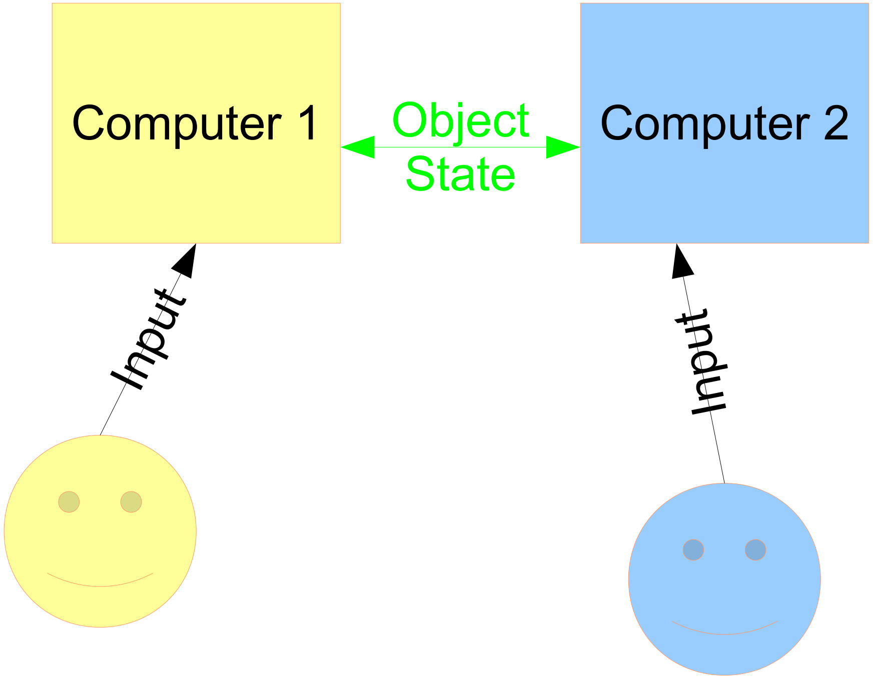
Multi-Station, the first networked games

- The "lock-step" model
  - Every station is running the same game/simulation (sim)
  - Works because on a LAN, latency is infinitesimal



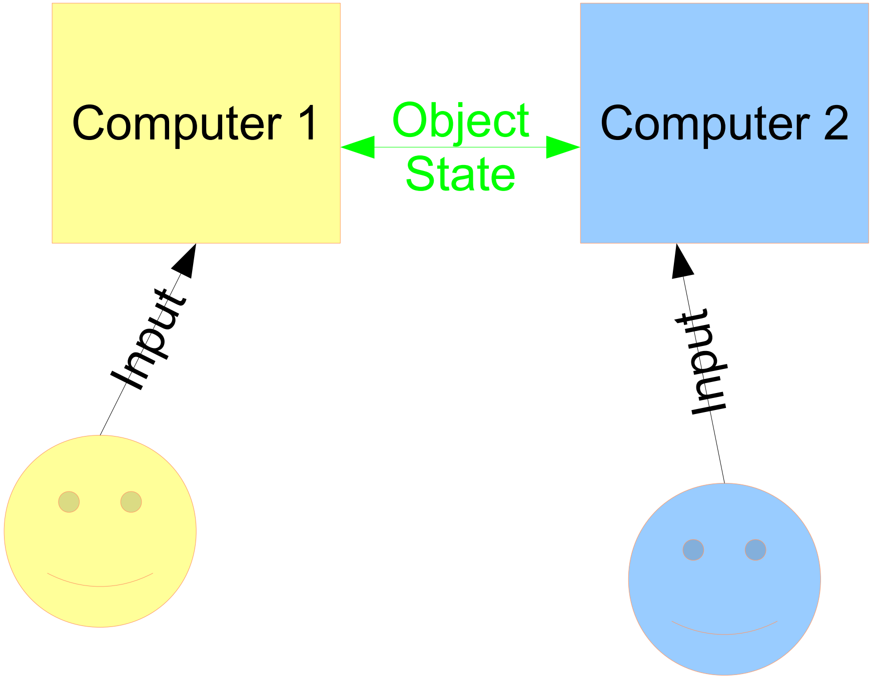
Flight Sims: Open Loop/Asynchronous (Asynch)

- Based on work for SimNet (DIS)
  - Each system has its own variant world state
  - Each vehicle is simulated on one machine
    - Periodic time-stamped state updates sent to others
    - Lower freq then controller input



Flight Sims: Open Loop/Asynch

- Dead Reckoning
  - > Each sim makes "best guess" at non-local positions
    - – Use vehicle model to assist
      - "Tanks don't fly"
  - > Corrects as updates are received
    - > Note: Updates always in past.
  - > Requires conflict resolution mechanism
    - > "shooter decides"



Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Revision A

Stepping into Cyberspace

- First Internet capable games/techniques
- Kali
  - > NBIOS emulator over TCP/IP
  - > Lock step games tended to play badly
    - > Reducing packets per second helped
    - > Latency buffering helped
  - > Open loop/asynch tended to play well
    - > Already designed for limited bandwidth and real net latencies
- TCP/IP support added to games
  - > Pluggable 'net drivers'
  - > More attention paid to latency and bandwidth issues

Internet Play:
Lock Step Pros and Cons

- Pros ?

Internet Play: Lock Step Pros and Cons

- Pros
  - Cheat proof
  - Exact synchronization assured
- Cons ?

Internet Play:
Lock Step Pros and Cons

- Pros
  - Cheat proof
  - Exact synchronization assured
- Cons
  - Every player's experience limited by worst case
  - Handles latency spikes poorly
  - Handles dropped players poorly
    - Needs to wait for timeout to determine drop v. spike

Internet Play: Open Loop/Asynch Pros and Cons

- Pros ?

Internet Play:
Open Loop/Asynch Pros and Cons

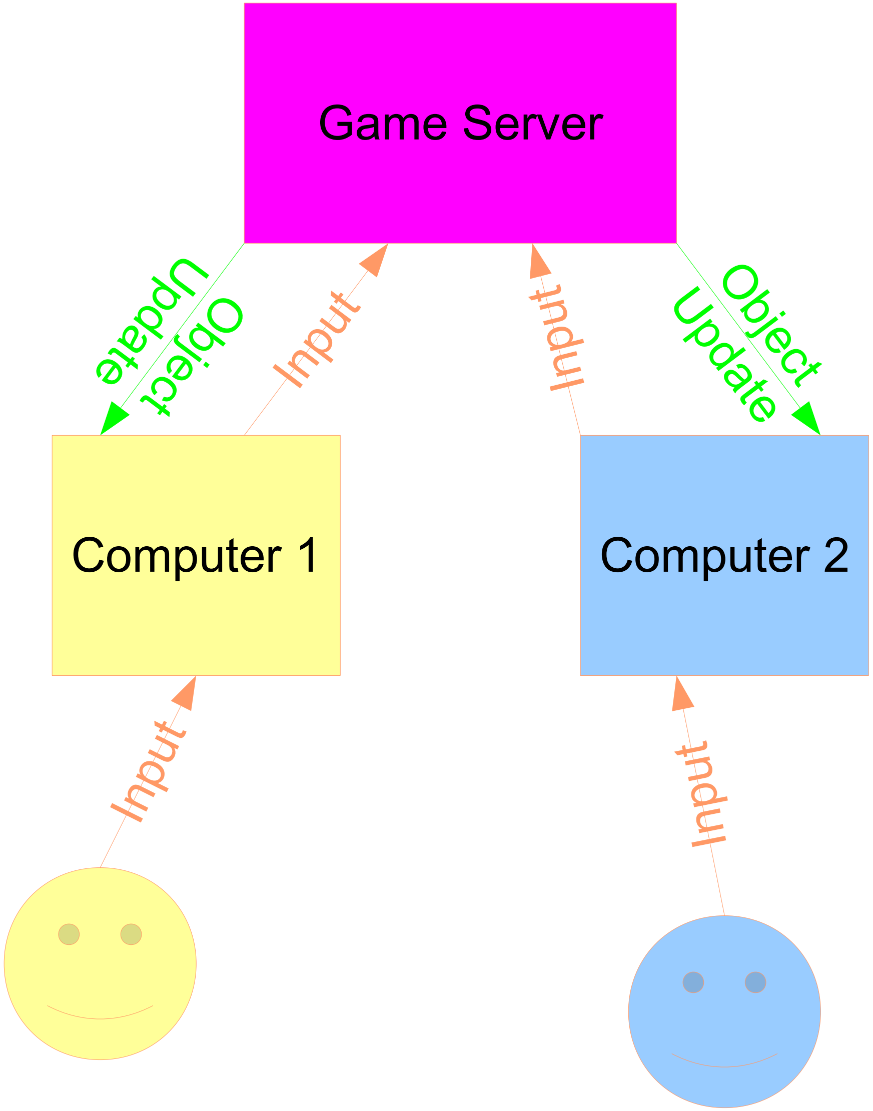
- Pros
  - Good at hiding latency
    - Smooth predict/correct over many frames
  - Better bandwidth control
    - Can communicate less often
      - 'shape' by distance
      - Out of sight, out of mind
- Cons ?

Internet Play: Open Loop/Asynch Pros and Cons

- Pros
  - Good at hiding latency
    - Smooth predict/correct over many frames
  - Better bandwidth control
    - Can communicate less often
      - 'shape' by distance
      - Out of sight, out of mind
- Cons
  - Prone to cheating
    - Need to trust sender as to position
    - Need to trust shooter as to hit/miss
  - Occasional 'warping' or other artifacts
- In general, technique used by all vehicle sims

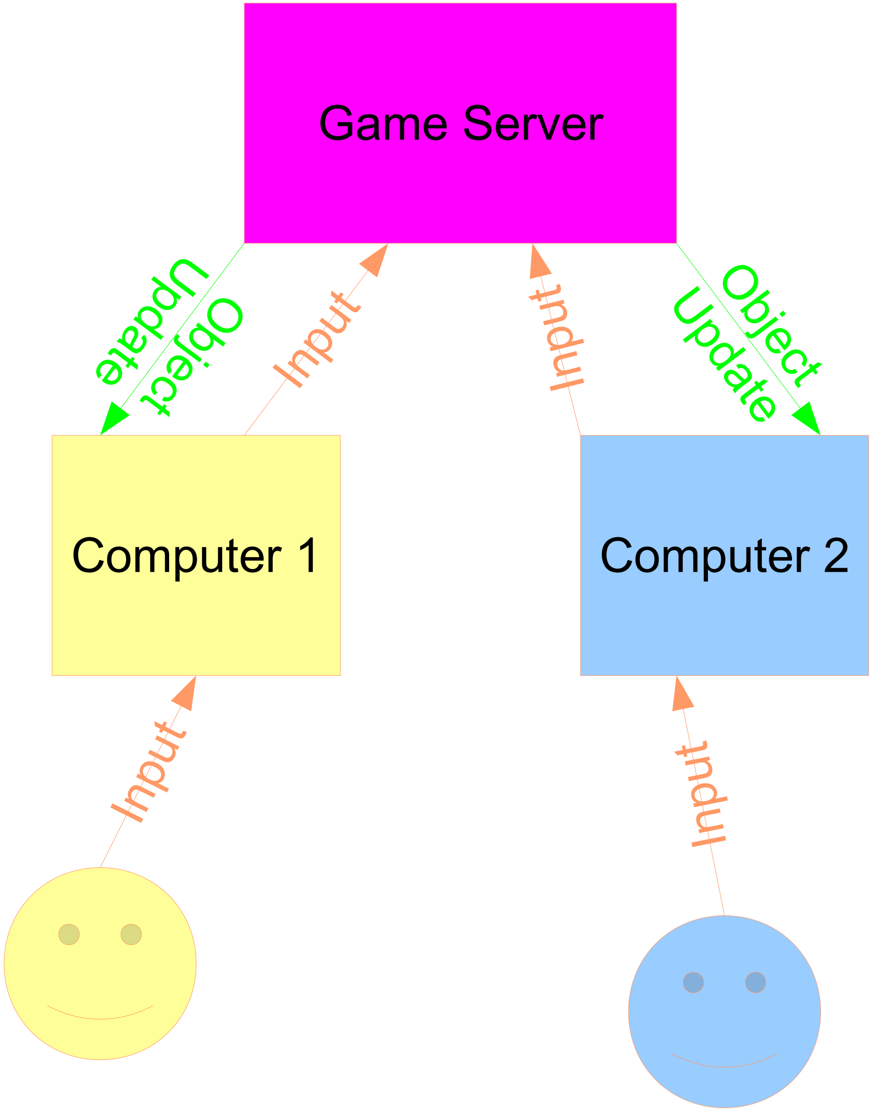
Quake: The first client/server game

- Server runs authoritative simulation
- Clients run open loop/asynch views
  - Really rich "controllers" for server.



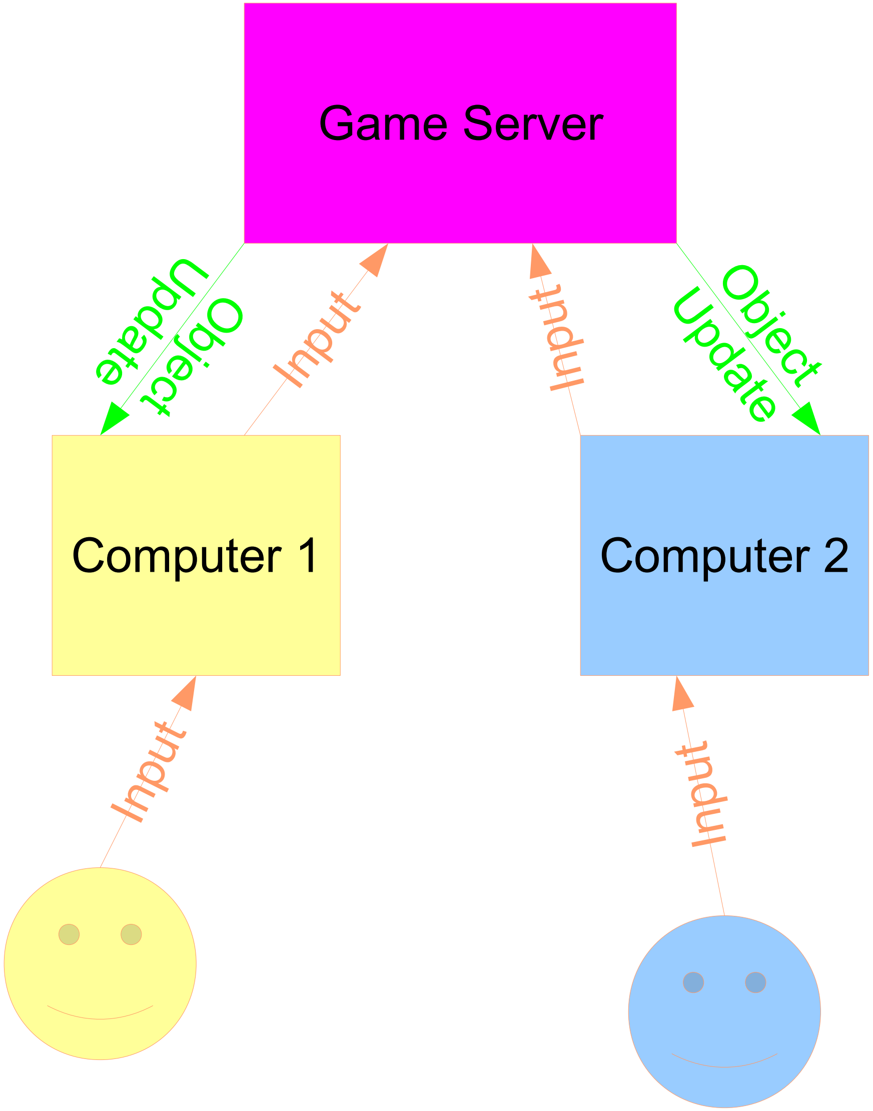
Quake: The first client/server game

- Pros ?



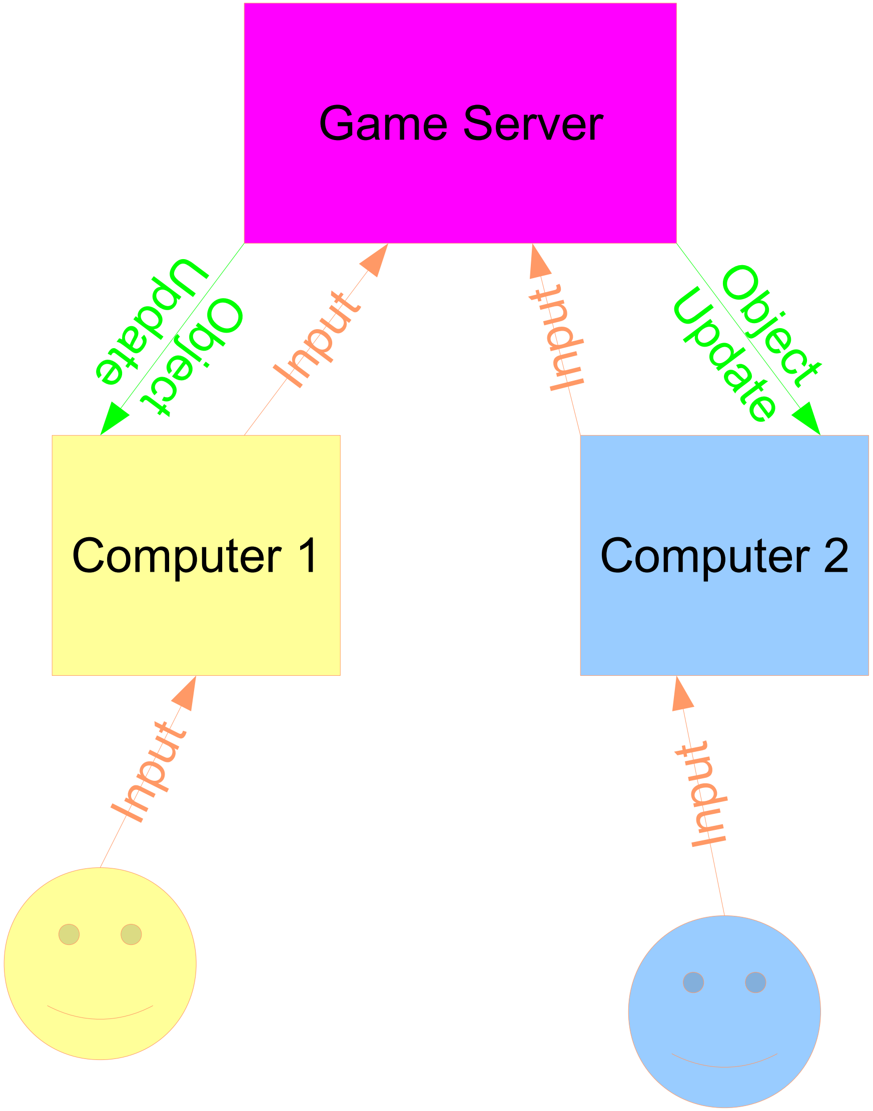
Quake: The first client/server game

- Pros
  - > Cheating is much more difficult
  - > Still not totally impossible
    - > Aimbot
- Cons ?



Quake: The first client/server game

- Pros
  - Cheating is much more difficult
  - Still not totally impossible
    - Aimbot
- Cons
  - What looks like hit to shooter can miss
  - "Low Ping Bastard" (LPB) effect



First Person Shooters Today

- Still fundamentally Quake model
- Player interactivity limited to control LPB effect
- Packet encryption to defeat aimbot
  - Not perfect security, but generally good enough

Game Discovery: LANs

- On LAN, players communicated with broadcast
  - First, broadcast play
    - Only one game session per LAN
  - Later, broadcast discovery, unicast play
    - Multiple sessions per LAN

Game Discover: WANs

- In Cyberspace, no one can hear you broadcast
  - On Internet, players need each others IPs
  - Initially, player entered manually
    - Found each other through IRC
  - GameSpy offers discovery service
    - Programmatic, but still over IRC
    - Simple directory server plus chat
    - Funded by advertising on client
  - TEN and MPath offer complete services
    - Net APIs and star architecture comm servers

Game Discovery Today

- TEN and MPath are gone
- Gamespy
  - Industry standard
  - has expanded data services
  - Now has comm API
    - Thin wrapper over peer to peer TCP/IP and UDP
    - Does UDP socket introduction through IRC
  - Licensed per game, advertising in Gamespy client
    - Most games don't use the Gamespy client
- Xbox Live/PC Live
  - Microsoft's attempt to get into the TEN/MPath space
  - Yearly fee, electronic retailing

Tomorrow... MUDs and MMOs or..

“The British are Coming!”

End of Unit One

Unit Two: MMO Architecture in Depth

What this lecture is about

The Evolution of MUDs and MMOs

Lecture Overview, Day Two

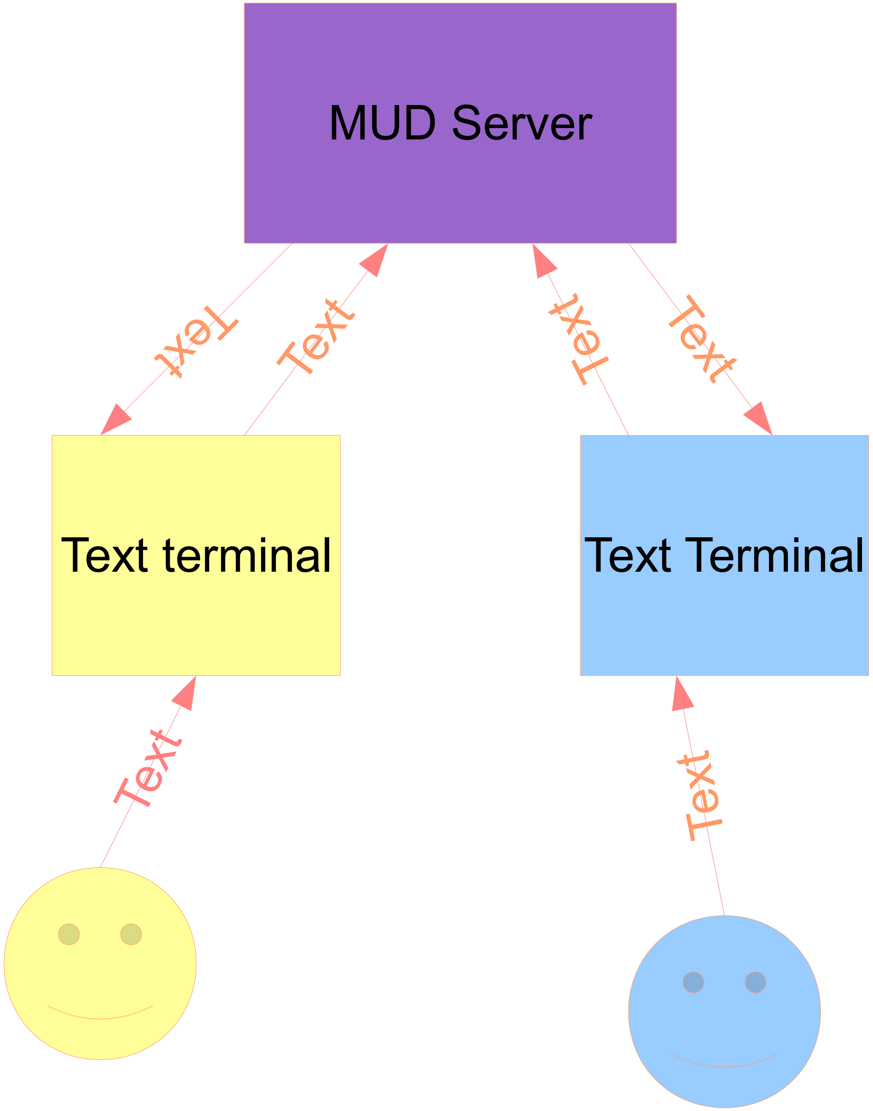
- The evolution of the MMO
  - > From MUD to WOW in 30 minutes
- The Difficulties facing today's MMO developers
  - > The motivations for Project Darkstar

MUD's and MMOs

Foreign DNA

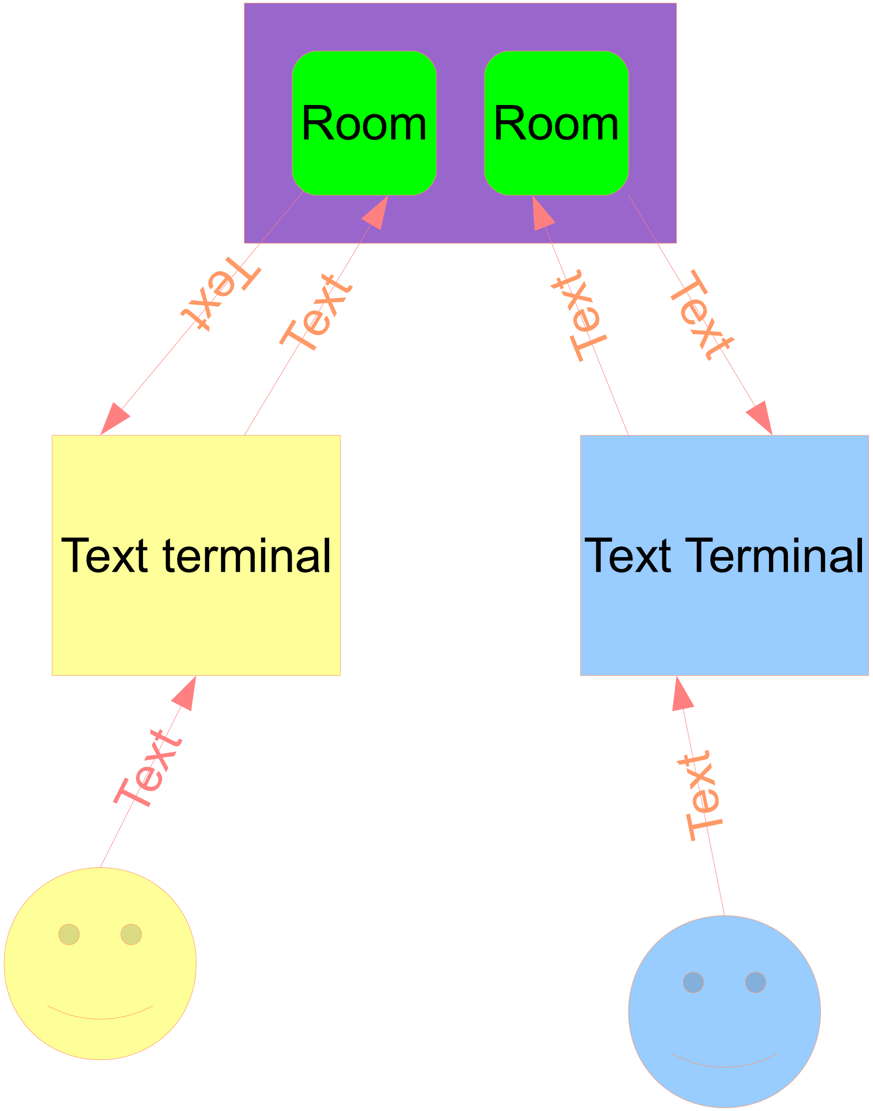
Meanwhile, in merrie olde England

- The Birth of the MUD
  - > Multi-user text adventures
  - > Event driven servers
  - > Textual command based world simulation
    - > User submits text, eg “take sword”
    - > Server updates world state and sends textual reply
      - – Others also see text for world state change



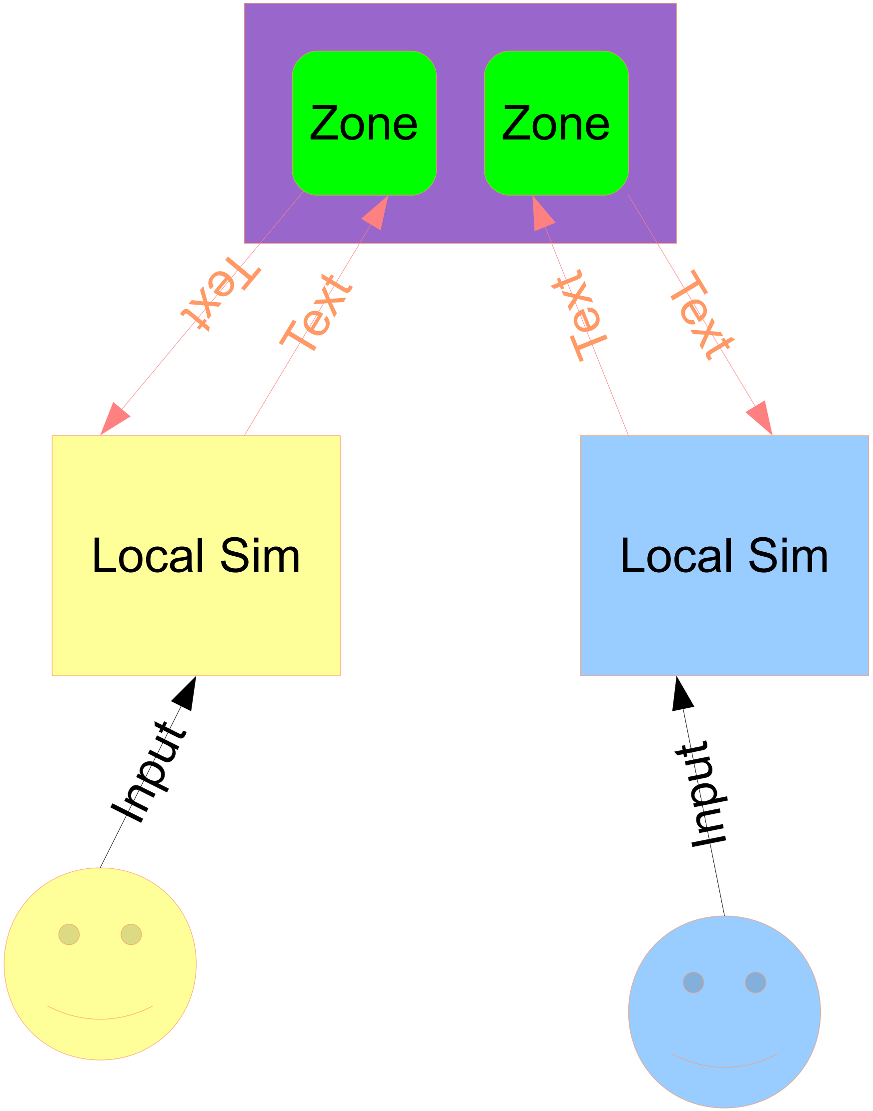
Meanwhile, in merrie olde England

- Used concept of “room” to break down n-squared communication problem
  - > Only those in room 'see' changes to room state
  - > Only those in room can act on others in room
  - > What if you run out of rooms?
    - – Virtual /instanced' rooms



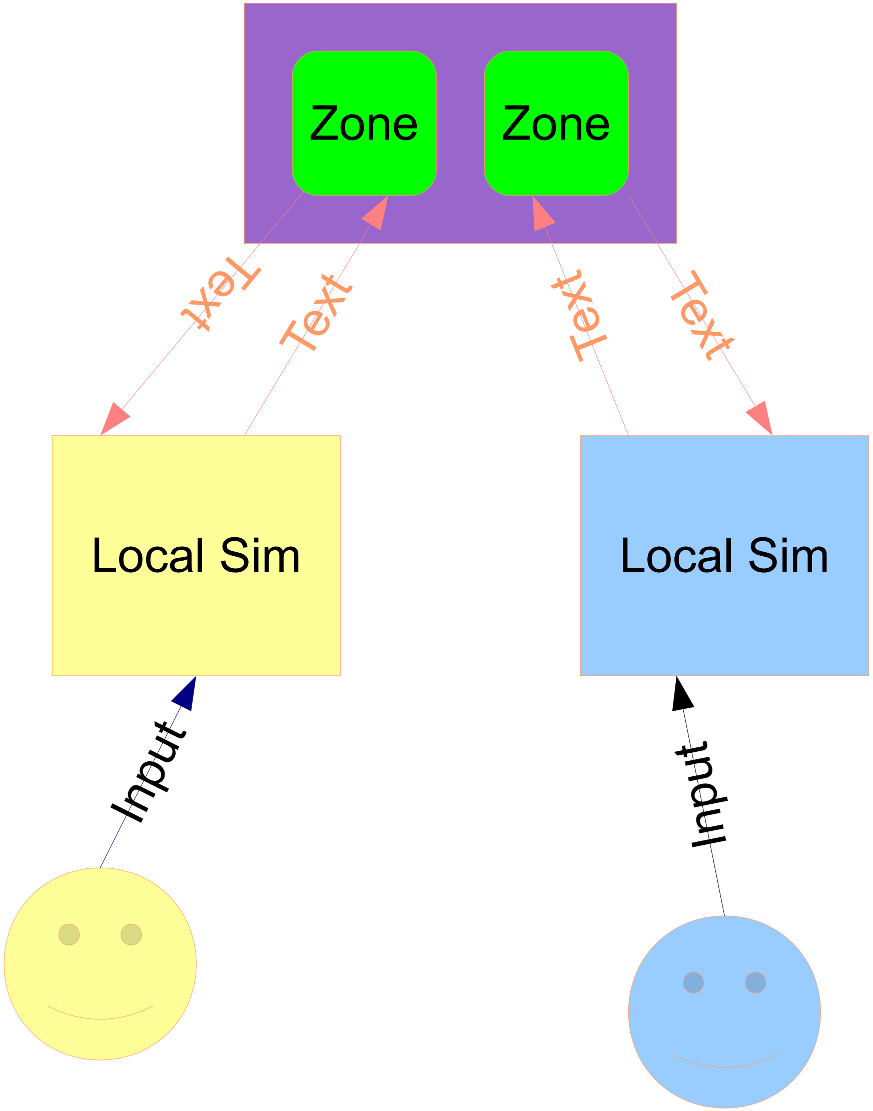
Ultima Online: The Visual MUD

- 2D game for client
  - Levels or “maps” as in previous 2D games
  - Each player on map has a position
- MUD for server
  - Map becomes feature of room (Zone is born)
  - Position on map becomes feature of player object



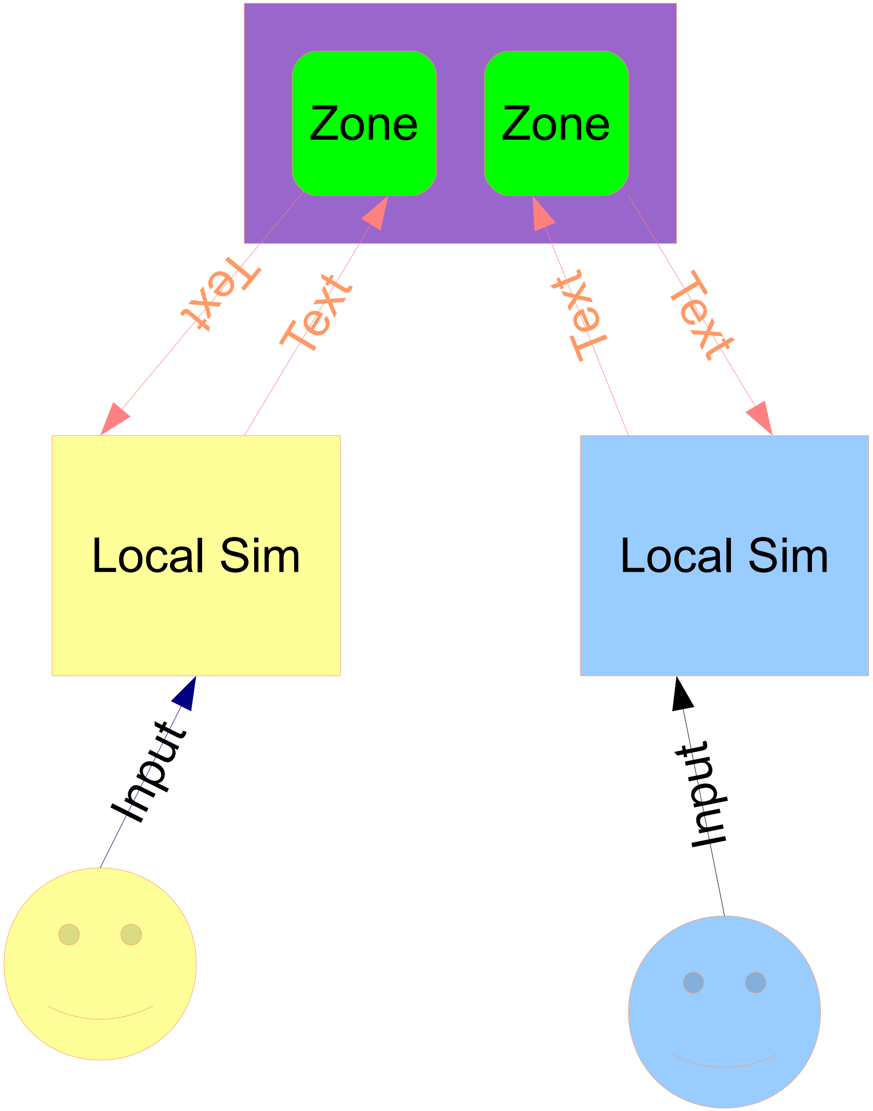
Ultima Online: The Visual MUD

- Hybrid of vehicle sim and text mud
  - > Motion == Open Loop/Asynch game
    - > Higher frequency then vehicle sim
    - > Gen. more players at once
    - > Loose combat model compensates
  - > World interaction == event driven MUD
    - > Still text & event driven



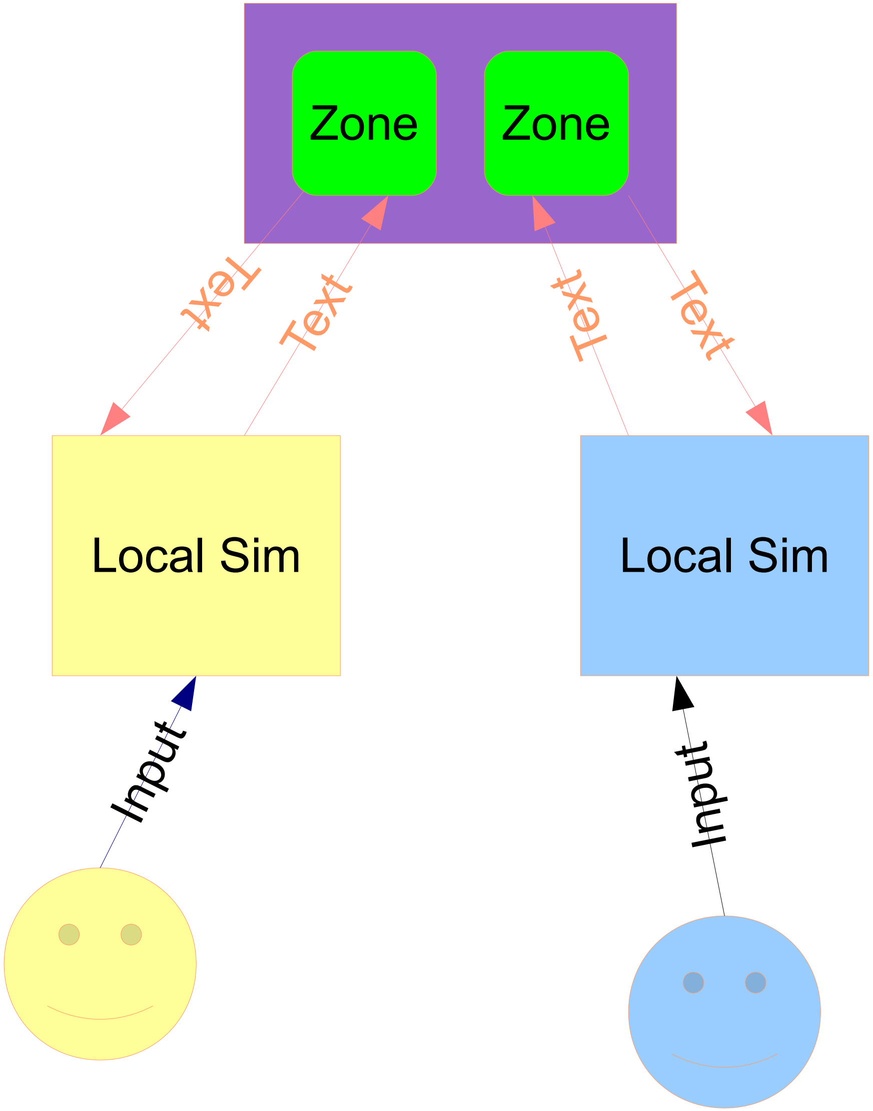
Ultima Online: The Visual MUD

- Issues?



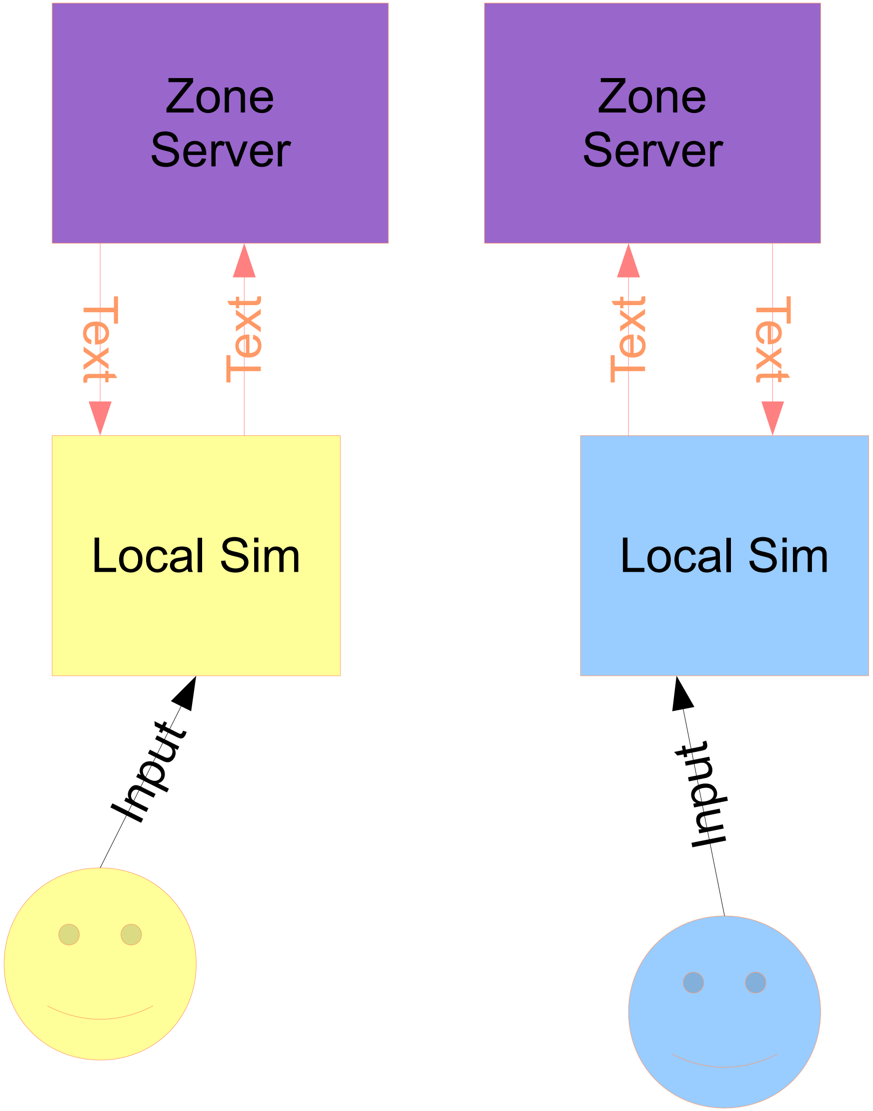
Ultima Online: The Visual MUD

- Issues?
  - Over-crowding of "popular rooms"
    - "fire marshal limit"
  - Scalability limited by power of server
    - Replicate server
  - Server crash loses state of whole world
    - Static worlds
    - Persistence of users
      - Inventory
      - Experience
      - Quest flags



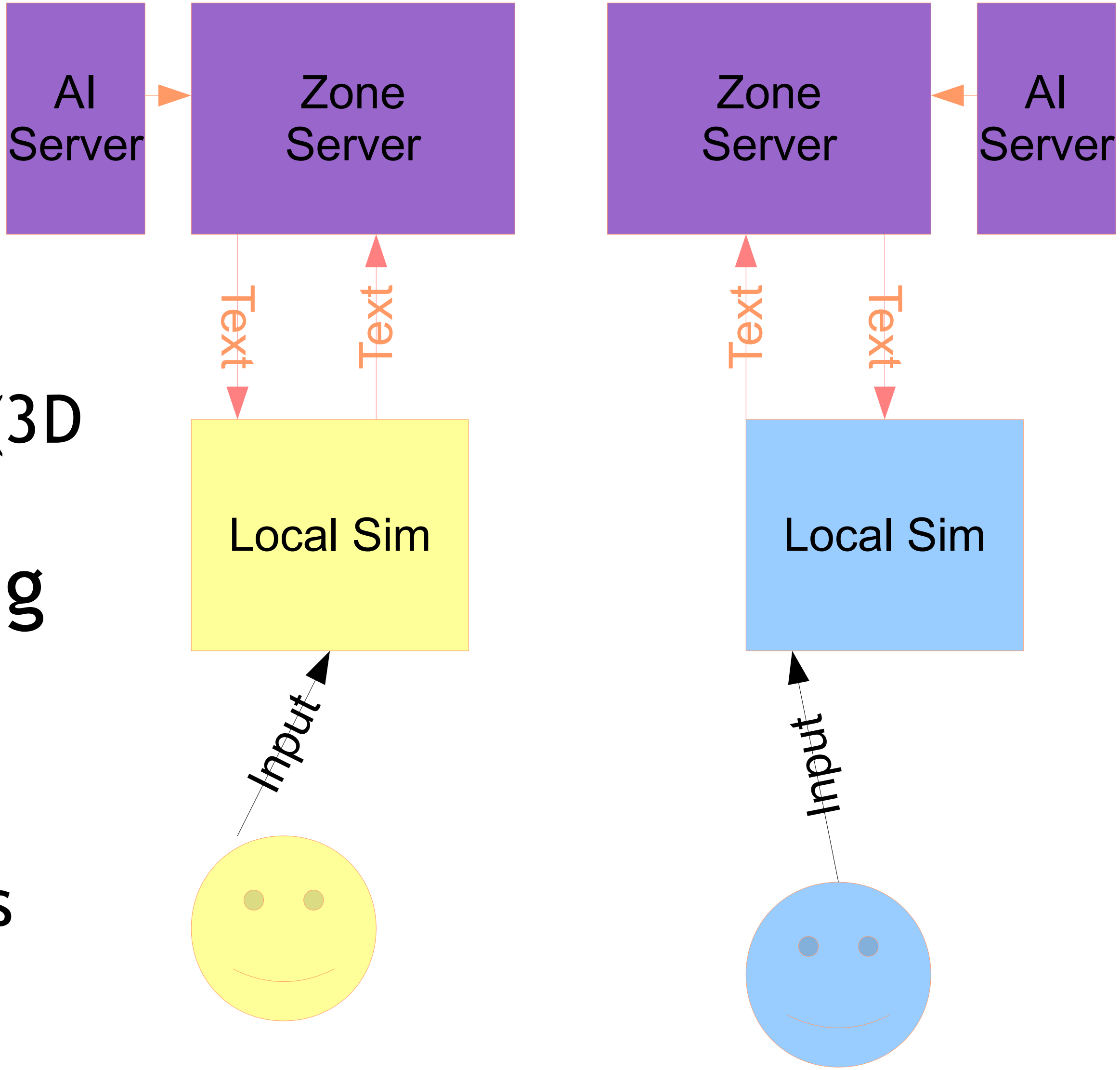
Everquest (EQ): The birth of the Shard

- EQ needed more power
  - More users
  - More work per user (3D world)
- Solved by clustering
  - Server per Zone
  - One cluster is called a 'shard'
    - Shard is represented to user as one 'server'
    - Terminology left over from UOL



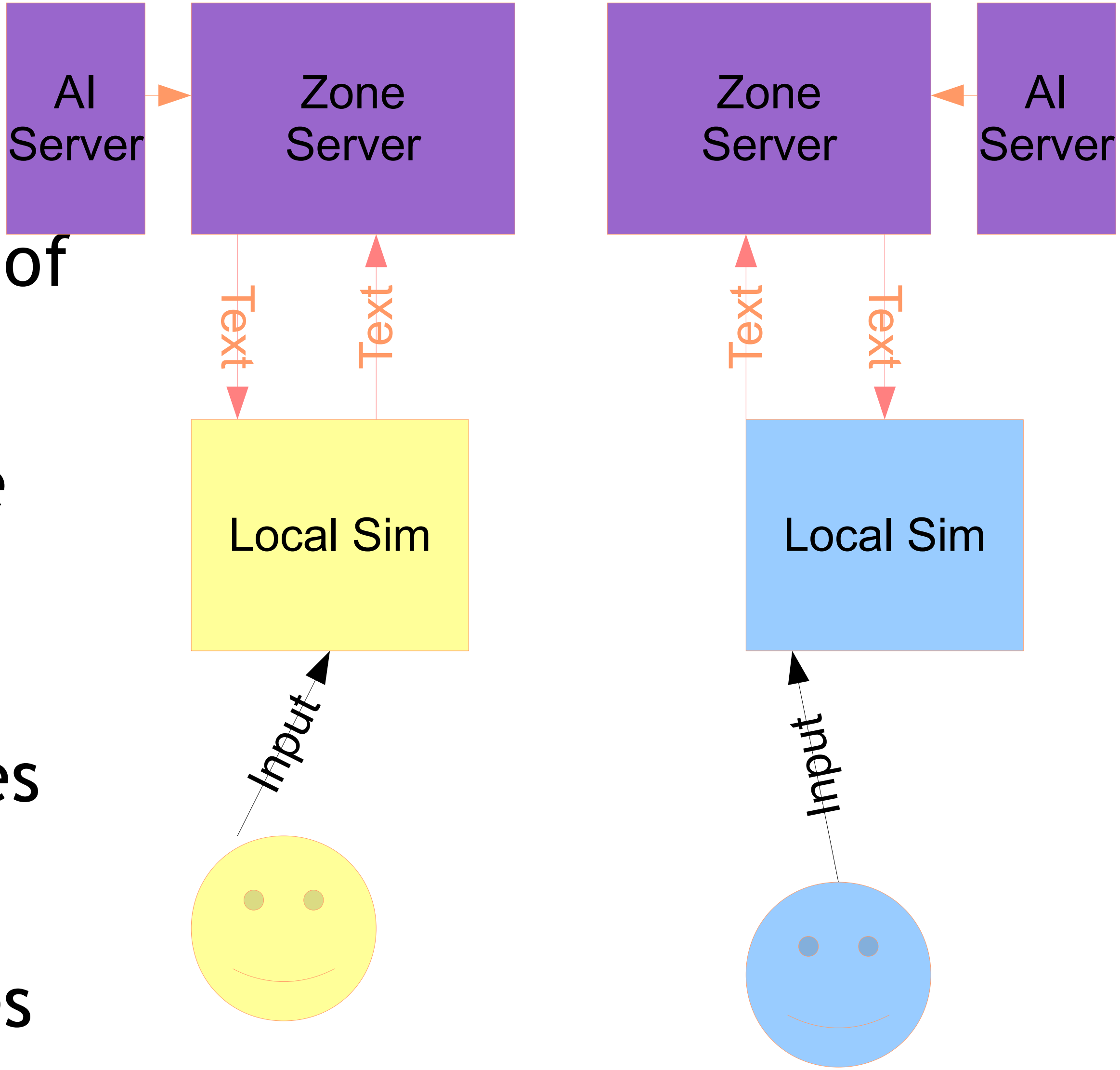
Everquest (EQ): Further load reduction

- EQ needed more power
  - More users
  - More work per user (3D world)
- Solved by clustering
  - Moved MOB AI to separate server
    - A system "player"
  - Other special servers
    - Commerce
    - Chat
    - Physics (CoX)



Everquest (EQ): Further load reduction

- Issues?
  - Many single points of partial failure
  - Zone server failure means loss of zone state
    - Like UO but only partial loss of world
  - Over crowded zones
    - Return of the fire marshall
  - Under utilized zones
    - Wasted CPU resources



Phantasy Star Online: The rebirth of the Virtual Room

- Question: Can we do better scaling then shards?
- PSO Answer: Mission Instancing
  - One standard zone as a “hub”
    - Chat
    - Create parties
    - Get a 'mission'
  - Mission is a virtual zone
    - Created when party enters
    - Destroyed when party leaves
    - Limits n-squared to max party size
    - Only has state while occupied
      - Can be run on a random machine from a pool

Modern MMOs

- Generally some mix of persistent and instanced Zones
  - Guild Wars
    - Towns persistent, all else instanced
    - Like PSO with multiple hubs
  - CoH/CoV
    - Persistent outdoors divided into Zones
      - Outdoors 'street sweep' missions
    - Instanced 'indoors'
      - Indoor instanced missions
    - Late addition: Instanced outdoors
      - Duplicates for over-flow
      - Breaks immersion some
        - “Are you in Atlas 1 or Atlas 2?”

That's the state of the art today

- Various minor tweaks
  - Incremental improvements
  - Different mixes of techniques
- Things to remember
  - Game development is a me-too business
    - Technical evolution happens slowly due to risk
    - Mostly focused on client experience
  - Architectural innovation happens elsewhere
    - Biggest leaps are usually the adoption of techniques already proven elsewhere

Issues Facing Today's Game Developer

- Single player games expanding user expectations
  - > Physics
  - > Advanced AI
  - > Interactive Environments
- Online user base growing non-linearly
  - > Great for business, bad for engineering
- All this == greater hunger for CPU and communication bandwidth

Game development hit the wall

- The game loop is a mono-threaded view of the world
  - “near-realtime” coding is what game developers know how to do
- Past growth was fueled by Moore's law CPU speed ups
  - CPUs suddenly stopped getting faster
  - Moore's law is now multiplying cores instead
    - Taking advantage of it is hard
      - Outside game developers' skill sets
    - Most business oriented solutions too slow and limiting
      - Business app servers optimized for avg throughput
        - Games care more about worst case latency
      - Wrong model-- still need to know about locks and databases

The answer.... Project Darkstar

- Research Question:
  - Observation: Multi-threaded, multi-machine code is vital to enable future online games
  - Observation: Multi-threaded, multi-machine coding is very hard to get right
  - Observation: Game coders know nothing about multi-threaded programming
  - The Question: Can we make multi-threaded, multi-machine game code automatically out of mono-threaded programs in a way that optimizes for worst case latency?

Is this possible?

- Can we make multi-threaded, multi-machine code automatically out of mono-threaded programs?
  - No. Pretty much proved impossible
- Can we make multi-threaded, multi-machine **online game** code automatically out of mono-threaded programs?
  - A special case
  - With a few constraints we believe this **is** possible

How?

Tune in Thursday ... same bat time... same bat channel

End of Unit Two

Unit Three:
Project Darkstar

What this lecture is about

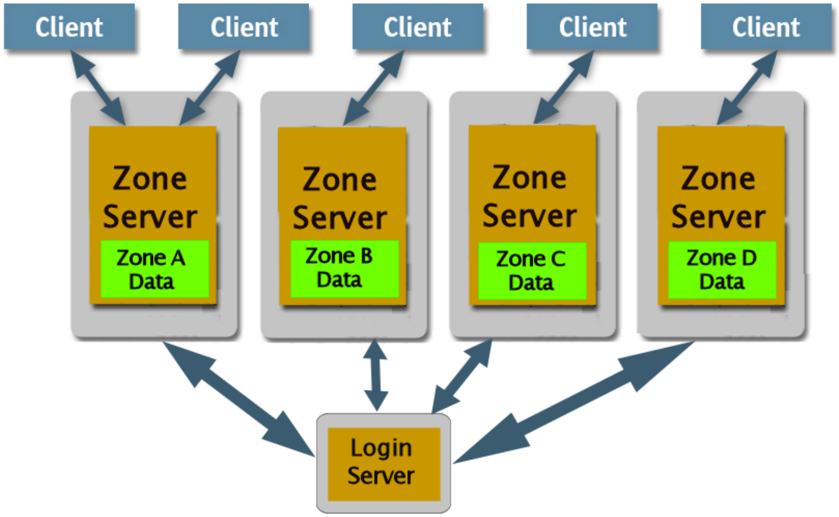
The motivation and architecture of Project Darkstar

Lecture Overview, Day Three

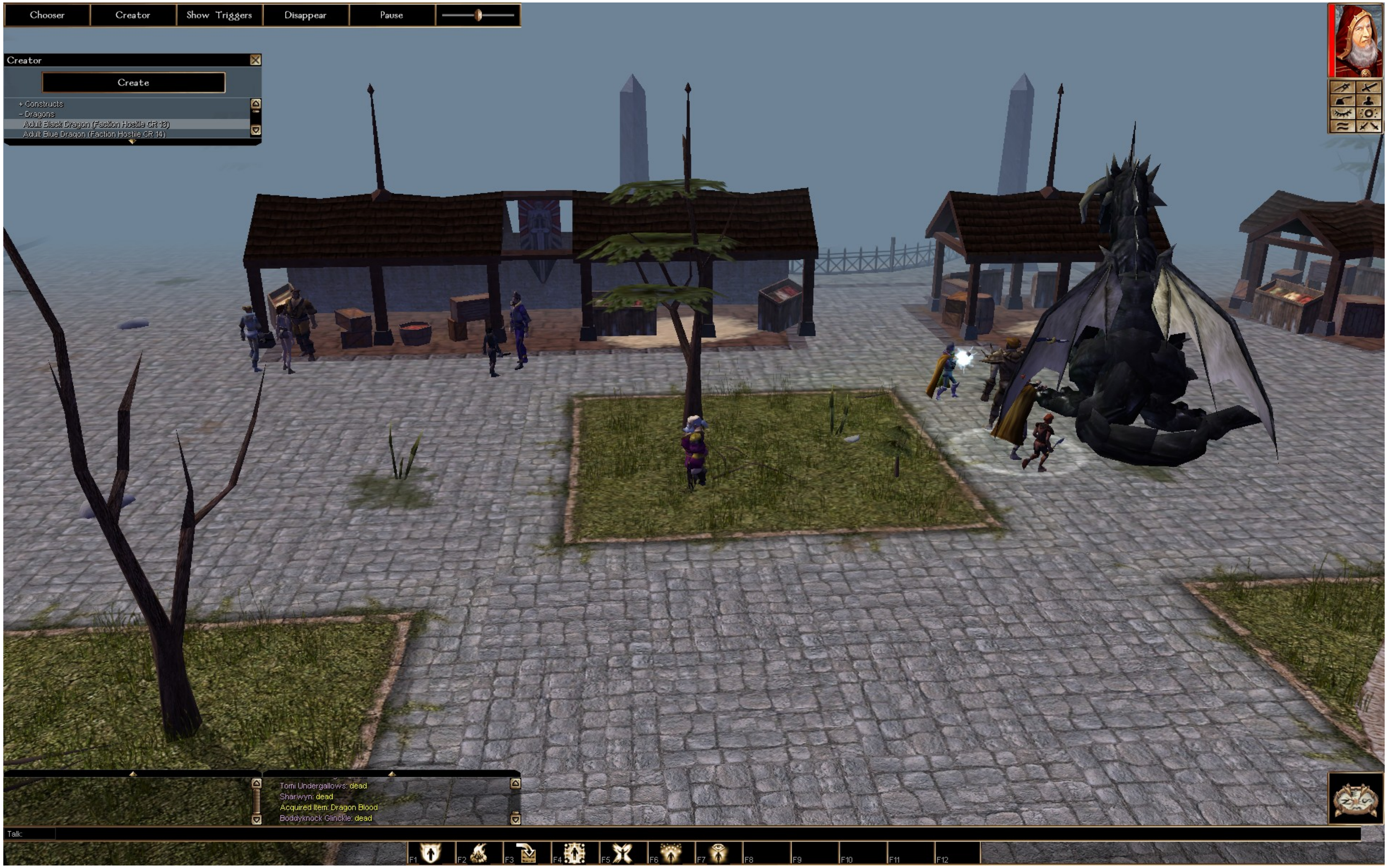
- Review: MMOs today
  - > Today's MMO architecture
  - > Issues facing today's developers
- Project Darkstar
  - > The motivations for Project Darkstar

Traditional MMO Architecture

- World broken up geographically into “Zones”
- Each Zone is on a Zone Server
- All state for that Zone in Zone Server's memory
- User state check pointed to Login Server

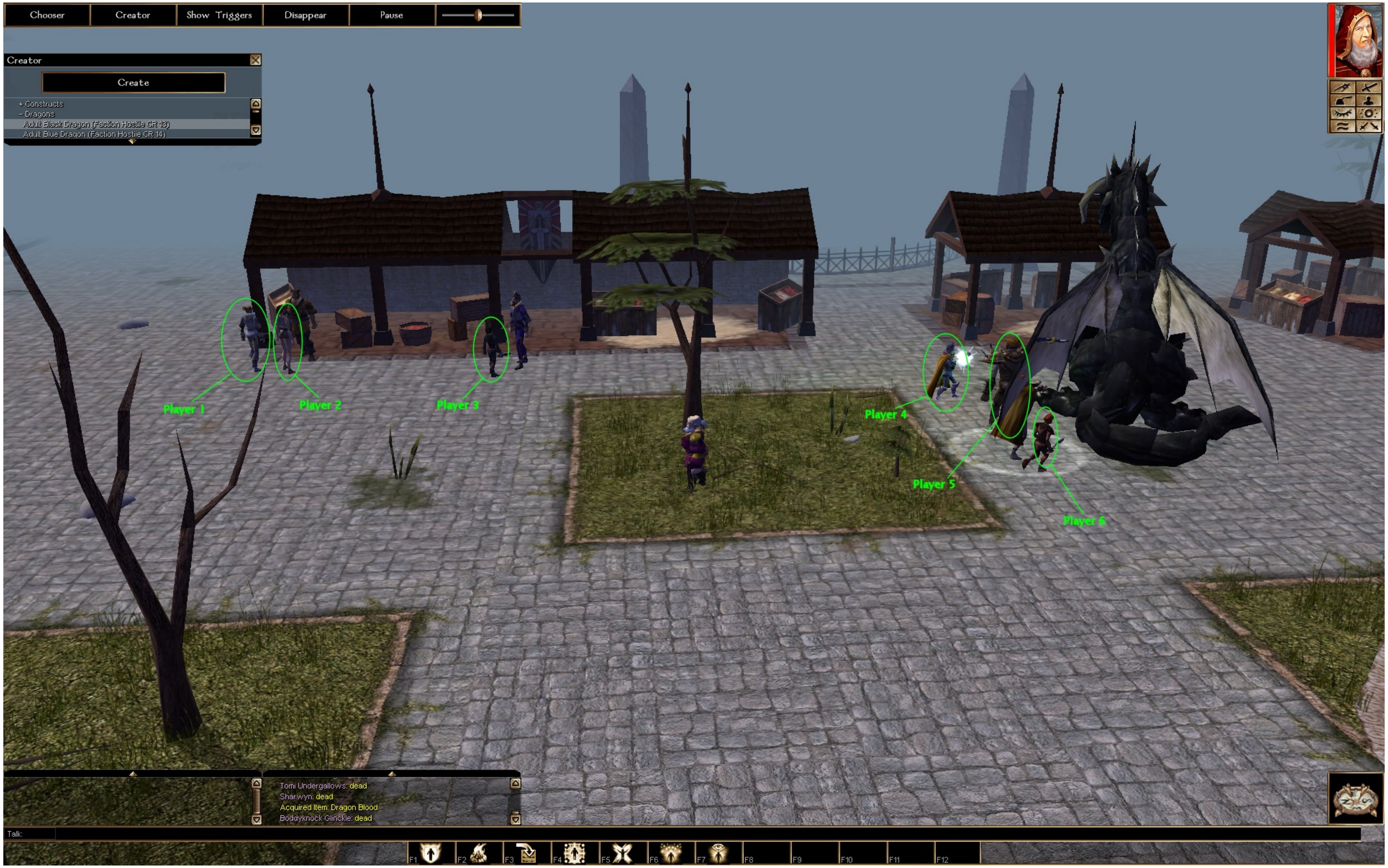


Typical MMO Scene



Tomi Undergallows: dead
Sharwyn: dead
Acquired Item: Dragon Blood
Boddyknock Glinckle: dead

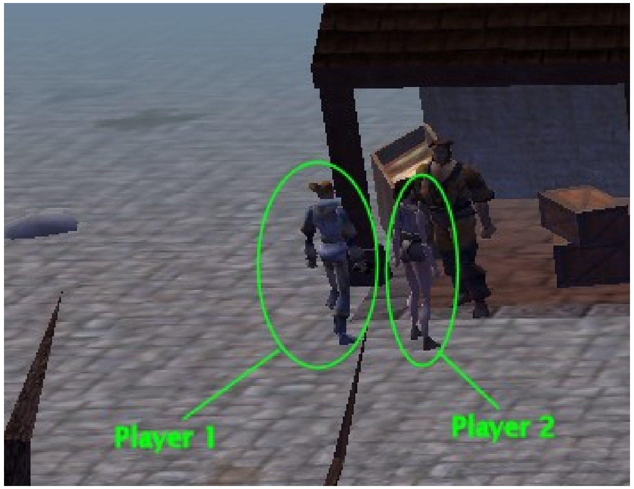
Whats going on here?



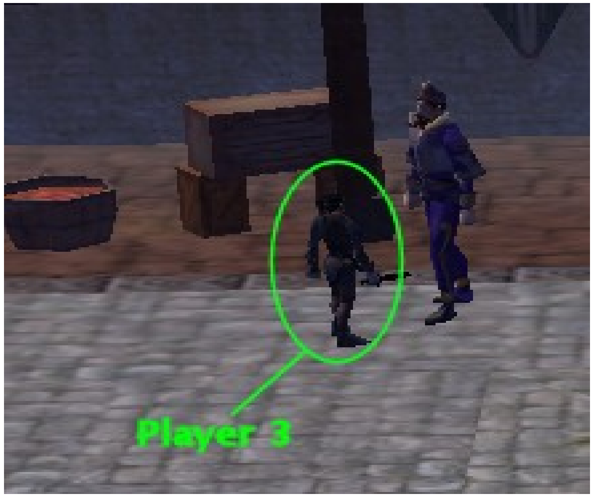
```
Tomi Undergallows: dead
Sharwyn: dead
Acquired Item: Dragon Blood
Boddyknock Glinckle: dead
```

Whats going on here?

- These players are dealing with a merchant

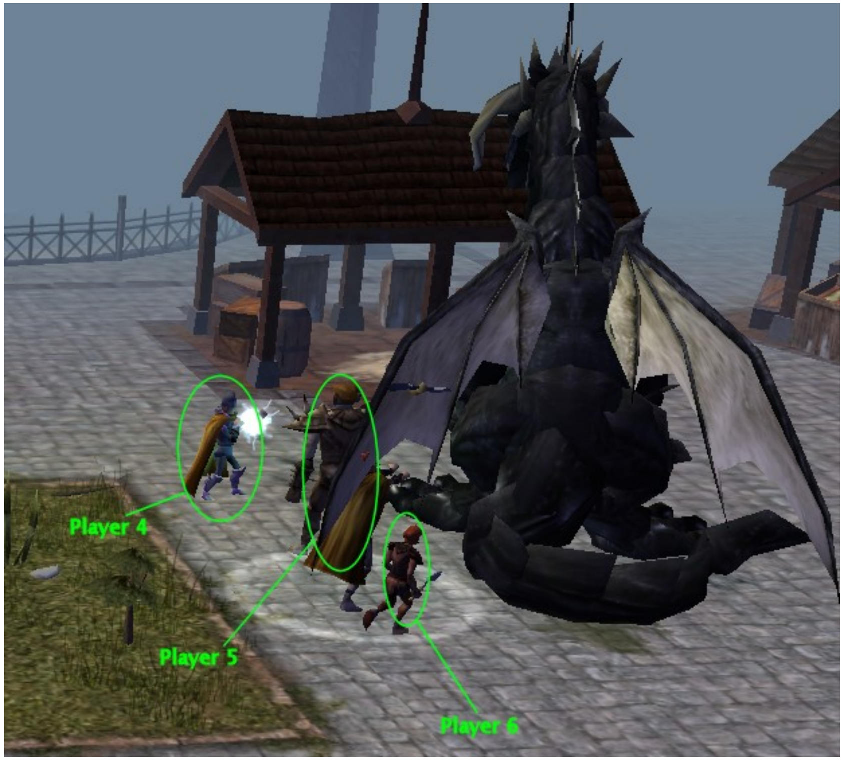


- This player is talking with an NPC



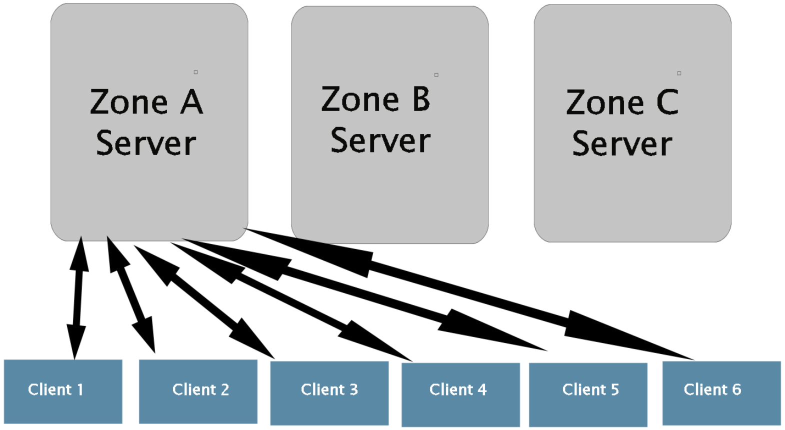
Whats going on here?

- These players are fighting a Dragon



Traditional Architecture: Load

- All this action occurs in Zone A
  - > Must be processed by Zone Server A
  - > Other Zone Servers can be idle
- Geographic Distribution
  - > Industry standard architecture
  - > Would be perfect if people were Gaussian



Zone A Server

Zone B Server

Zone C Server

Client 1

Client 2

Client 3

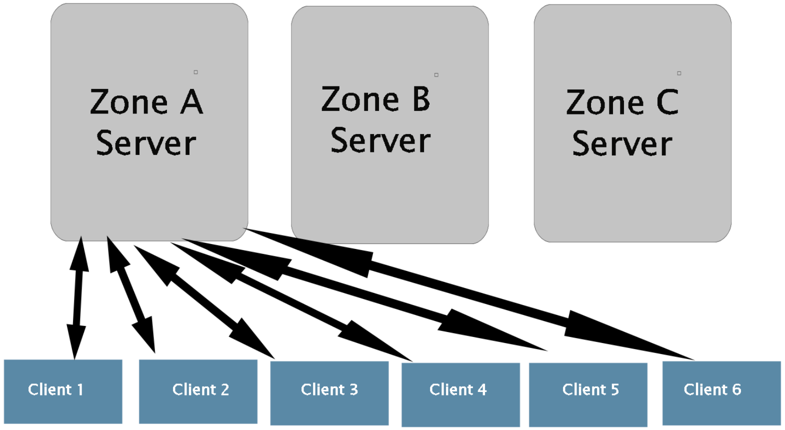
Client 4

Client 5

Client 6

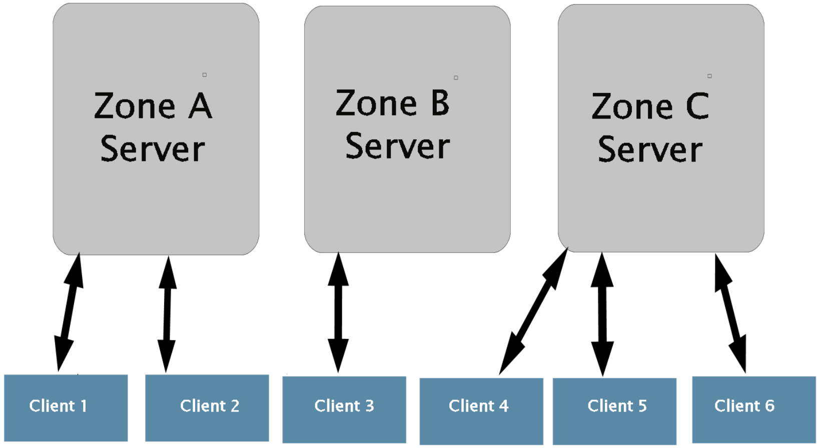
Traditional Architecture: Failure

- If Zone A server fails
  - Zone's game state is lost
  - Players states are lost back to last checkpoint
  - Players cannot get back in until server is restored
    - Just happened to me on CoH
    - Required CSR action



MMOs are inherently parallel

- Wouldn't it be great if the action could be split up?
  - Merchant being processed by one server
  - NPC chat by another
  - Fight by another
- Problems:
  - Interactions are many, varied and dynamic
  - Parallel programing is hard



Zone A Server

Zone B Server

Zone C Server

Client 1

Client 2

Client 3

Client 4

Client 5

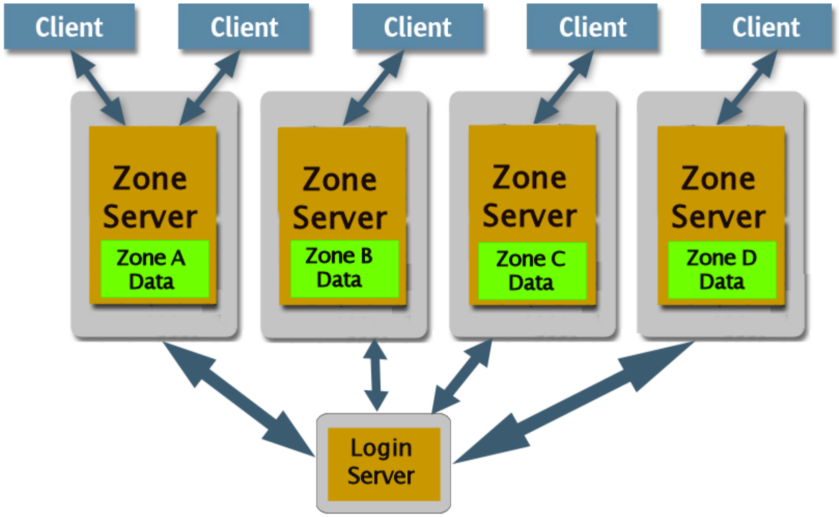
Client 6

What we really want is...

- A way to dynamically allocate interactions to a pool of servers
- A way to get whatever data is needed to that server
- A way to recover state in the case of failure
- A coding model that is comfortable and intuitive for people who think mono-threaded
  - ENTER PROJECT DARKSTAR

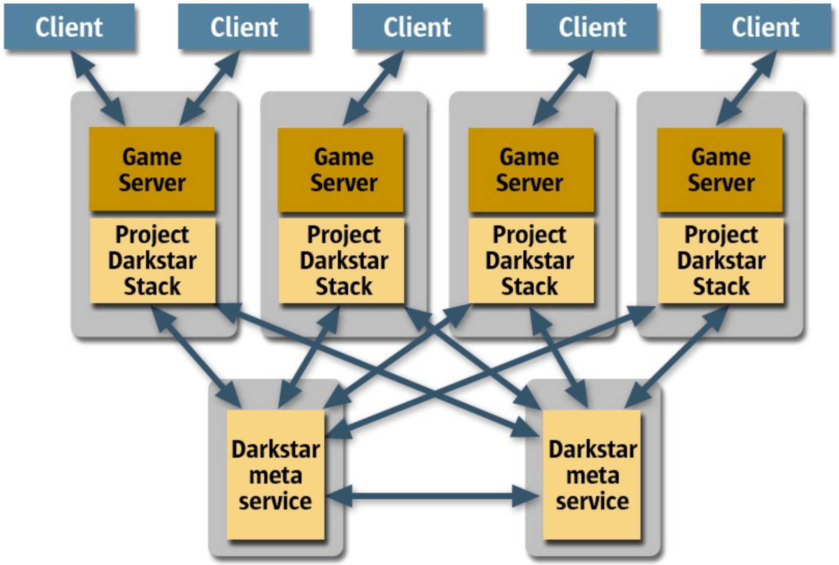
Recall...

- Scales badly
- Wastes resources
- Limits persistence
- Has problematic failure modes



Project Darkstar Architecture

- Stateless processing nodes
- Identical code on each processing node
- Data is stored in a meta service (Data Manager)
- Data flow to processing nodes as needed



Darkstar application model

- Event-driven Programs
  - > Event generates a task
  - > Task code is *apparently* mono-threaded
  - > Tasks are independent
  - > Code that does not meet this model must be deployed in a Darkstar "service"
- Tasks **must**
  - > Be short-lived
  - > Access data through Darkstar services
  - > Communicate through Darkstar services

Making it multi-threaded

- All tasks are transactional
  - Either everything is done, or nothing is
  - Commit or abort determined by data access and contention
- Data access
  - Data store detects conflicts, changes
  - If two tasks conflict
    - One will abort and be re-scheduled
    - One will complete
- Transactional communication
  - Actual communication only happens on commit

Project Darkstar Data Store

- Not a relational database
  - Is an enterprise class database
    - Reliable, Scalable, Fault Tolerant
  - No SQL
  - Optimized for 50% read/50% write
- Keeps all game state
  - Stores everything persisting longer than a single task
  - Shared by all copies of the stack
- No explicit locking protocols
  - Detects changes automatically
  - Programmer can provide hints for optimization

Project Darkstar Communication

- Listeners hear client communication
  - Simple client protocol
  - Listeners established on connection
- Client-to-client through the server
  - Very fast data path
  - Allows server to listen if needed
    - Can slow down communication
- Mediation virtualizes end points
  - Indirection abstracts actual channels
  - Any processing node can talk to any user

Distributing the load

- Darkstar tasks can run anywhere
  - > Data comes from the data store
  - > Communications is mediated
  - > Where a task runs doesn't matter
- Tasks can be allocated on different machines
  - > Players on different machines can interact
  - > The programmer doesn't need to chose
- Tasks can be moved
  - > Meta-services can track loads and move tasks
  - > New stacks can be added at runtime

The End Result

- Game programmer friendly programming model
  - A single thread
  - A single machine
- Multiple threads
  - Task scheduling part of the infrastructure
  - Concurrency control through the data store, transactions
- Multiple machines
  - Darkstar manages data and communication references
  - Computation can occur on any machine
  - Machines can be added (or subtracted) at any time

Some additional advantages

- Entire world is persistent
  - Not just user data
  - World can evolve
  - Durability guaranteed within a few seconds
- Major sources of error eliminated
  - Race conditions
  - Breaks in referential integrity
    - "dupe" bug
- Fails over and tolerates failure
  - Loss of individual node just increases load on others
  - Enterprise class Data Store recovers from complete failure

Does *not apply to many problems*

- **NOT A GENERAL SOLUTION TO MULTI-THREADED PROGRAMMING**
  - Impossible, remember?
  - The system works because of the assumptions we make that happen to match how games work
    - System tuned for worst-case latency
      - J2EE tuned for transactional throughput
    - System tuned for lots of little packets
      - Not a distribution server
      - For distribution of large static data blocks there are existent solutions
        - Web servers
        - Streaming servers

However...

- Can apply to other kinds of games
  - > Great platform for MMO casual games
  - > Good platform for Matchmaking and social services
- Can apply to “game-like” applications
  - > Car Auctions
  - > Military simulation
  - > Who knows??

Tomorrow

Coding for Project Darkstar

Unit Four: Implementing a Project Darkstar based game server

What this lecture is about

The nitty gritty details of coding using Project Darkstar

Part One

Client/Server design for Chess

Chess as a casual massively multiplayer game

- What belongs on client?
  - Game session management
    - Keep it simple – Every two players is a game
    - Need a login interface
  - Game interface
    - Game board display and animation
    - Move entry
    - Other game displays (timer? In-game chat?)

Chess as a casual massively multiplayer game

- What belongs on Server?
  - Game Session Management
    - Collect pairs of users
    - Create a game session for each pair
  - Game logic
    - Game state storage
    - Rules engine
    - AI for single player games

Part Two

Fundamental Project Darkstar “Moving Parts”

Tasks

- Darkstar application code is executed in Tasks
  - A task is a thread of control plus a transactional context.
  - Are time limited (default is 100ms)
  - Can be one-shot or repeating
  - Can be delayed or ASAP

Task Execution

- Execution is event driven
  - Event is translated to a task
  - User events
    - Result of client action (login,send,logoff,etc)
    - Are ordered in relation to user
    - Are unordered in relation to other users or system events.
  - System Events
    - Generated by Services
    - Queued by other tasks

System Events and Event Listeners

- Two system event listener interfaces
  - AppListener
    - Two event methods on AppListener
      - initialize()
      - loggedIn(...)
  - ClientSessionListener
    - receivedMessage(...)
    - disconnected(...)

Managed Objects

- Tasks execute methods on Managed Objects
  - Actually, this is an over-simplification but good enough for now
- Managed Objects are..
  - Stored in DataStore automatically
  - Can be bound to a name
  - Referenced through ManagedReference
  - *Almost* POJO

Life Cycle of a Managed Object

- MO is implicitly created in database the first time it is "seen" by the Data Manager.
  - > Ie DataManager.createReference(...) or DataManager.setBinding(...)
- MO state is saved at end of task
- MO must be explicitly destroyed
  - > DataManager.remove(...)
  - > There is NO gc of the database
- MO methods get executed by tasks or other MOs

Making Managed Objects

- Managed Object is a POJO that implements Serializable and ManagedObject
  - Executed by events
  - Persistence managed by Project Darkstar server

```
public class Counter implements
      Serializable,ManagedObject {
   int count=0;

   public int incrCount() {
      return count++
   }
}
```

Managed Objects

- ManagedObject do not require explicit locking
  - > However hinting helps the system optimize
    - > Call into system using managers
    - > Get managers using AppContext

```
public class Counter implements
      Serializable,ManagedObject {
   int count=0;

   public int incrCount() {
      DataManager dmgr=AppContext.getDataManager();
      dmgr.markForUpdate(this);
      return count++
   }
}
```

Managed Reference

- Managed Objectss must reference other Managed Objects through ManagedReference fields
  - Java objects referenced through Java reference fields are part of the private state of the containing Managed Object
    - Eg the int in Counter is part of the Counter instance's state
  - Managed References break the serialization graph and allow reference between Managed Objects
    - The reference is part of the containing MO, but the MO referenced has its own state

Managed Reference Example

- Wrong (will exception at runtime):

```
public class MyObj implements Serialzable, ManagedObject{
    Counter myCounter= new Counter;

    public class incr(){
        return counterIncr;
    }
}
```

- Right

```
public class MyObj implements Serialzable, ManagedObject{
    ManagedReference myCounterRef=
        AppContext.getDataManager().createReference(
            new Counter);

    public class incr(){
        return myCounterRef.(Counter.class).incr();
    }
}
```

Services and Managers

- Managed Obejct code calls Services through Managers
  - A service is..
    - A non-transactional piece of code
      - Not time limited
    - Not distributed (local to the VM)
      - May implement its own distribution scheme
    - Can talk to other services
    - Extensible
      - New services many be plugged into the system
    - The “driver level” of the system
  - A manager is..
    - A Task facing facade for a Service
    - Not required for all Services

Std Services with Managers

- Used by Tasks, System or other Services
  - Channel Manager
    - Provides efficient data transfer to groups of users spread across many nodes
  - Data Manager
    - Provides access to the Managed Objects
  - Task Manager
      - Provides ability to queue new tasks
  - Future services under consideration
    - Long running task manager
      - Provides easy way to do non-transactional time unbounded tasks
    - RDBMS manager
      - Access to external JDBC database

Std Services without Managers

- Used by other services and/or system
- Can also generate events
- Watchdog Service
  - Watches health of nodes
- Node mapping service
  - Maintains knowledge each node's workload
  - Redistributes work in case of node failure
- Client Session SeSrvice
  - Handles client logon/logoff
  - Maintains knowledge of client connection point

System Bootstrap

- How do initial listeners get registered?
- AppListener is “bootstrap” MO
  - AppListener class defined in app properties file
  - Iff there are no MOs in data store when server starts
    - Server creates bootstrap MO of specified class
    - Server registers that MO as the system AppListener for the two system events
    - System generates an initialize() event
      - Initialize() method sets up game MOs
- ClientSessionListener returned from AppListener.loggedIn(...)
  - Failure to return a ClientSessionListener results in immediate session termination

Standard Managers and Events

- Data Manager
  - Interface to data store
  - Generates no events
- Task Manager
  - Interface to task queue
  - Generates no events
    - Can do repeating tasks, sort of like heartbeat event
- Channel Manager
  - Interface to channel system
  - Can generate events
- Other managers may be plugged in
  - Can generate events if needed

Coding for Darkstar

Some best and worst practices

Designing Managed Objects

- Avoid Object Contention
  - Code is *apparently* mono-threaded
  - Darkstar takes locks underneath
  - Ergo: Must design app to avoid object contention
- Balance contention with overhead
  - Fetching each object has some fixed overhead
  - Loading object has variable overhead according to size
  - Ergo: Managed Object should encapsulate all data that is used together but as little other data as possible, bounded by a trivial size

Avoid unscalable algorithms

- Exponential growth will kill you
  - Object access has a cost
  - Touching n-squared objects is death
    - Example: polling all objects to see who is close
  - Communication has a cost
    - Sending n-squared packets is death
    - Example: everyone in a single chat
- Divide and Conqour
  - Create "awareness groups"
    - Remember the MUD rooms?
  - Proactive objects
    - Put themselves in/out of groups

Implementing Managed Objects

- A few constraints
  - No inner classes (except static ones)
    - Hold invisible references that can mess up serialization
  - No static fields (except final static ones)
    - Static field values specific to a VM
    - ManagedObjects float between many VMs
  - No references to shared Java objects
    - Every primitive and object referenced by a ManagedObject is part of its own state
  - No Java references to other ManagedObjects
    - Use ManagedReference
    - Breaks the serialization graph

Using Managed Objects

- Locking behavior
  - Working copy is fetched from ManagedReference:
    - Get() is a read lock
    - GetForUpdate() is a write lock
    - MarkForUpdate() is a promotion from read to write
    - Managed Objects that are only read locked but are changed will be promoted to write locked at task commit time
  - Multiple locks are harmless
  - Write locks cannot be de-promoted
  - All locks are held til task commit
  - Task aborts in deadlock, commits on exit

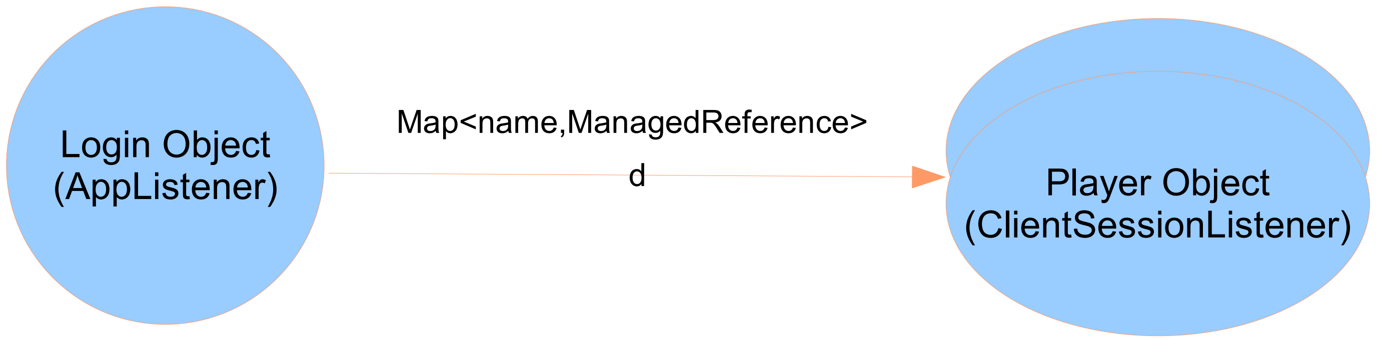
Locking Strategy

- In general....
  - Use get() if you do not know if an object will be updated
  - Use getForUpdate() or markForUpdate() when you know it will get updated
- Unless you are an expert in multi-processing, this will produce the best results

Thinking Project Darkstar

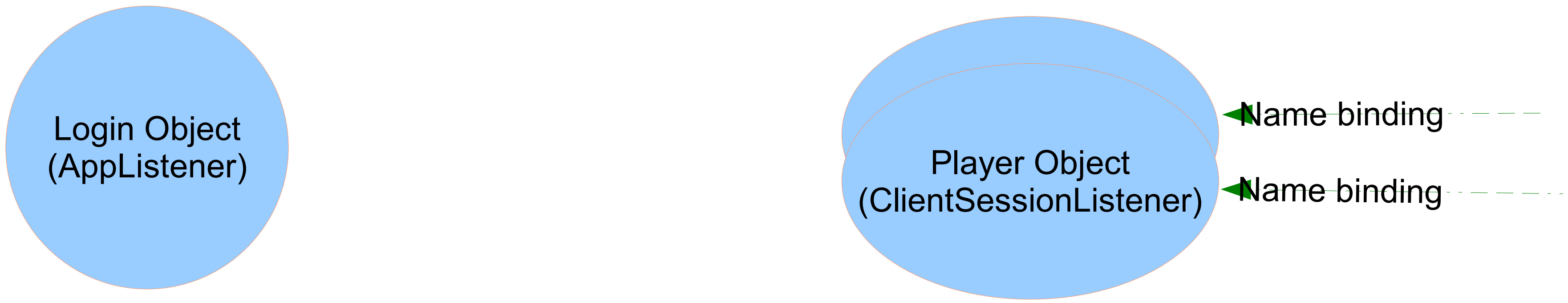
A few common anti-patterns..... see if you can spot the problem!

Anti-pattern One



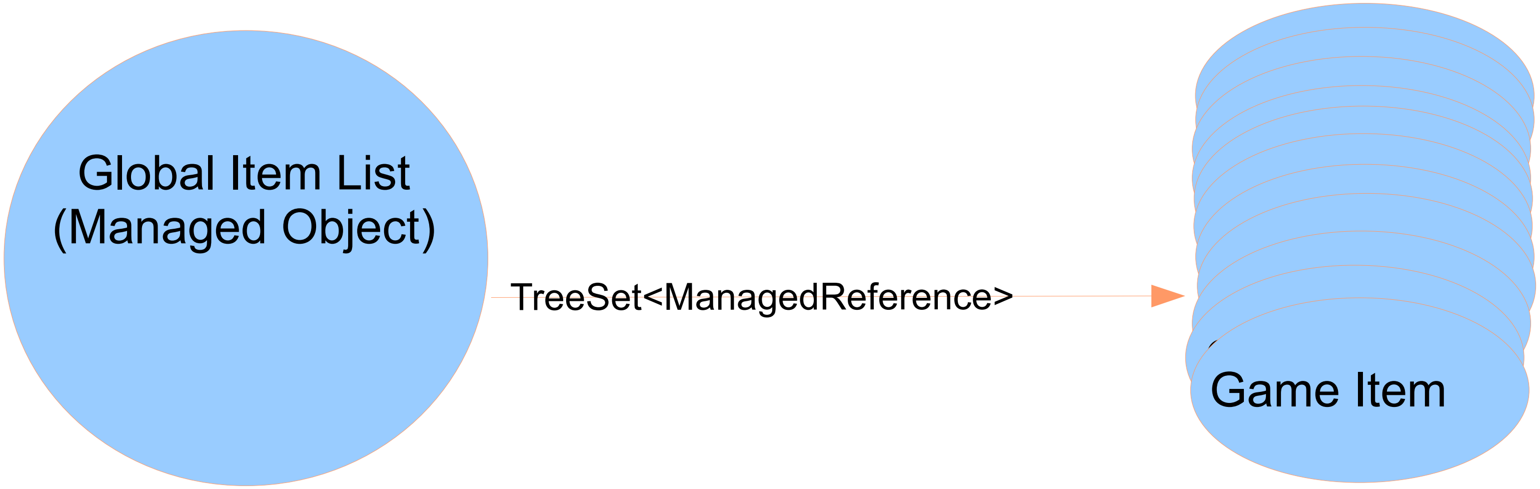
AppListener maintains a Map of all registered users to their login names

Anti-pattern One: Serialization of common occurence



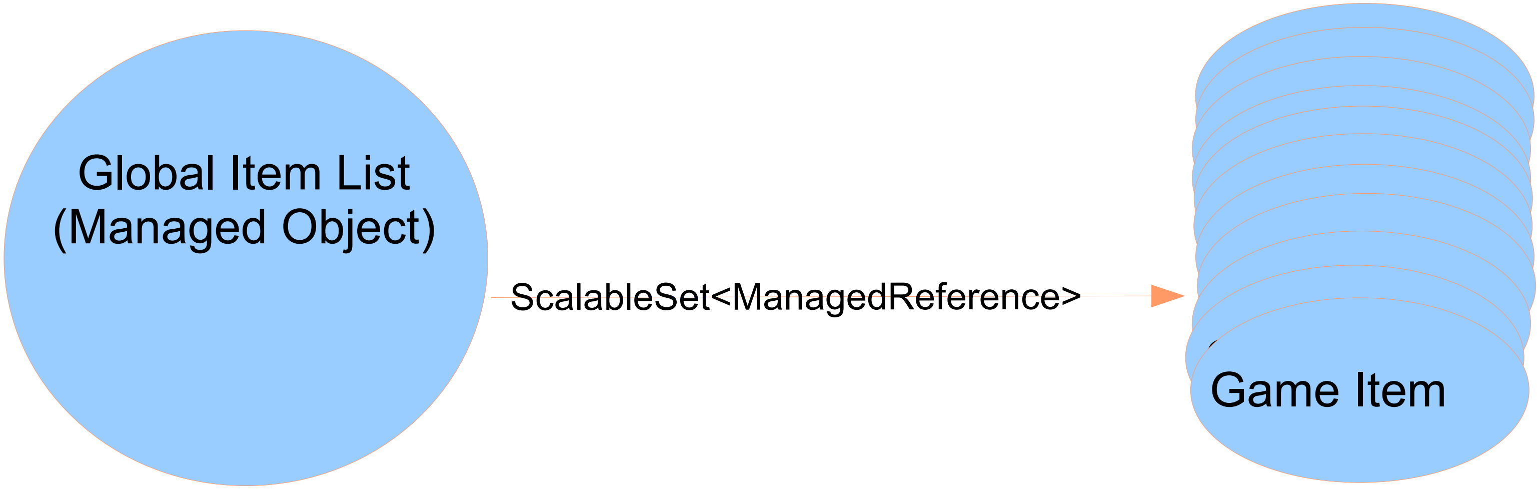
- Every new user must lock Login Object
  - > Serializes new user creation
  - > Use name bindings instead to find user object

Anti-pattern Two:



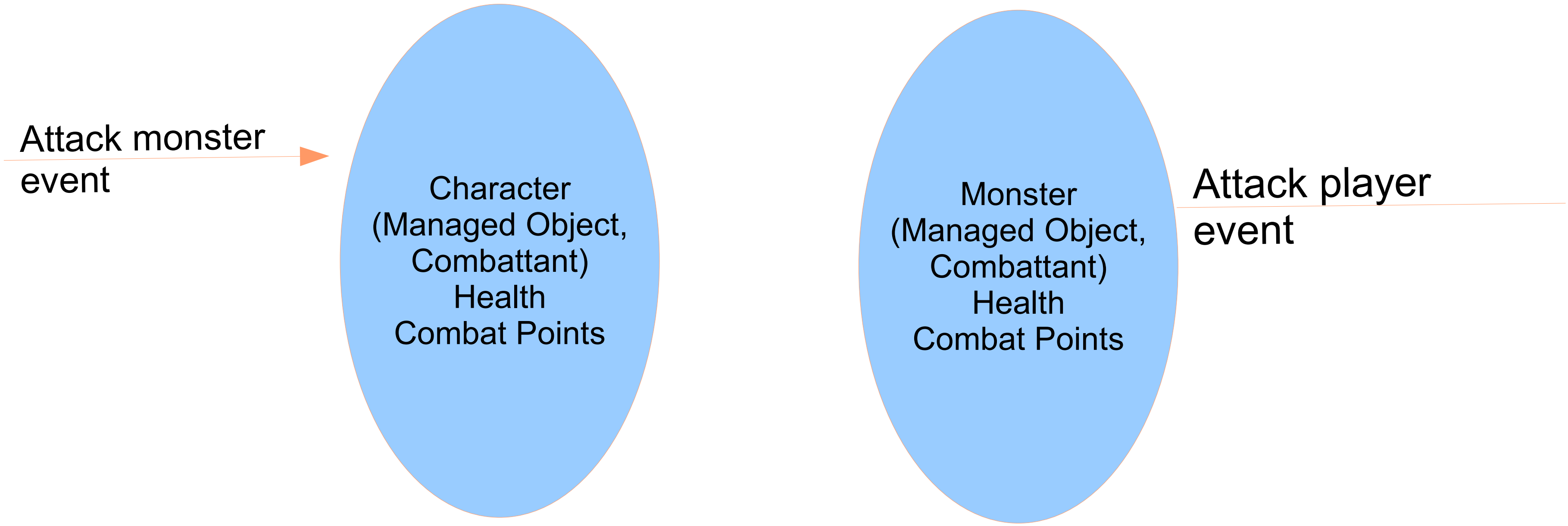
A managed object keeps a TreeSet of all games items currently in the world.

Anti-pattern Two: Large Java Collection or Array



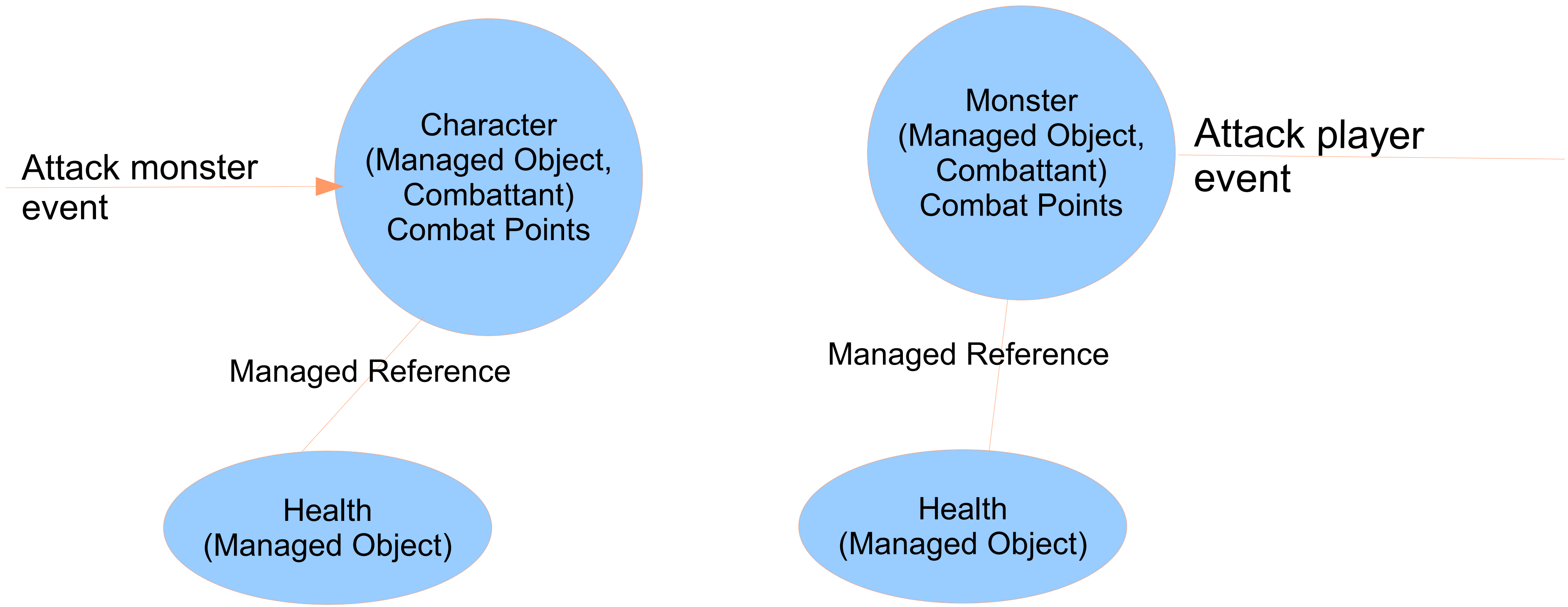
- Java Collection types do not scale
  - User proper sparse data structures
  - Where a large collection is truly required, use ProjectDarkstar collection types
    - ScalableSet
    - ScalableHashMap

Anti-pattern Three


Attack monster event
Character (Managed Object, Combattant) Health Combat Points
Monster (Managed Object, Combattant) Health Combat Points
Attack player event

- Each attack subtracts a combat point and subtracts health from opponent.
  - Common code

Anti-pattern Three: Deadlocks



- Each combattant locks self, then opponent
  - > Almost gauranteed deadlock
- Right solution is to split health and combat points on separate Managed Objects

Part Four

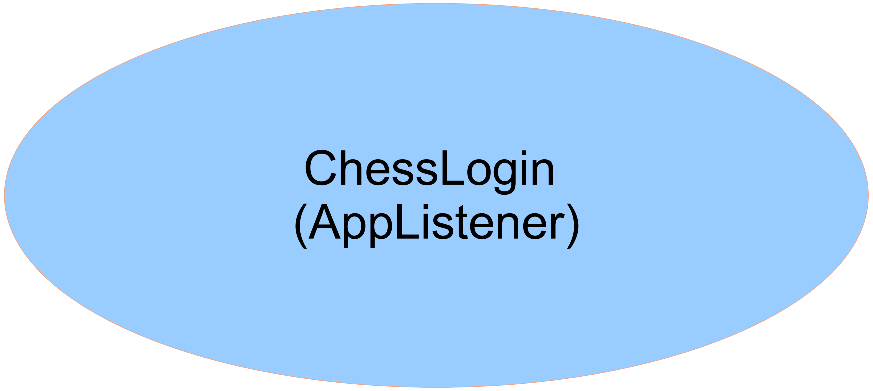
Chess Server Object Design

Session Management

- Features
  - > Logon
  - > Find or create a UserObject for this user
  - > Group every two users to a new board

Project Darkstar ChessLogin AppListener

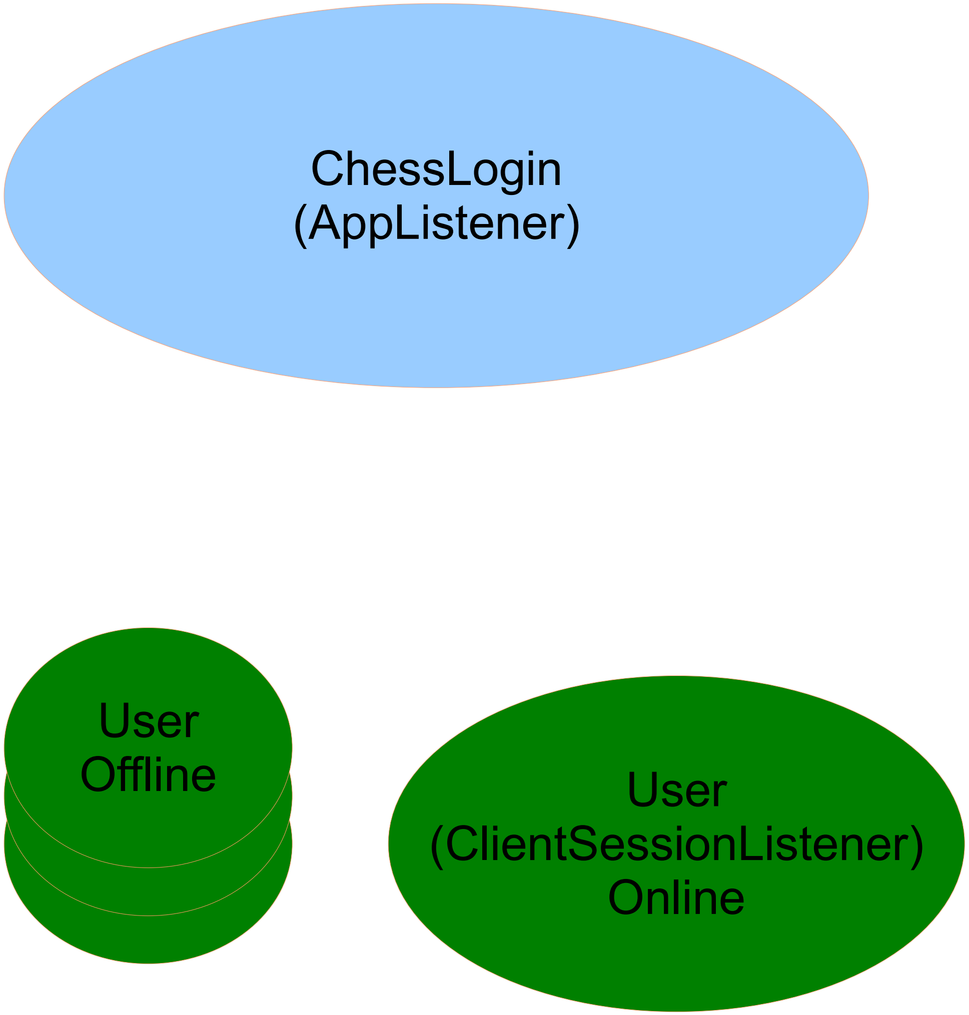
- On initialize()
  - Nothing to do



ChessLogin
(AppListener)

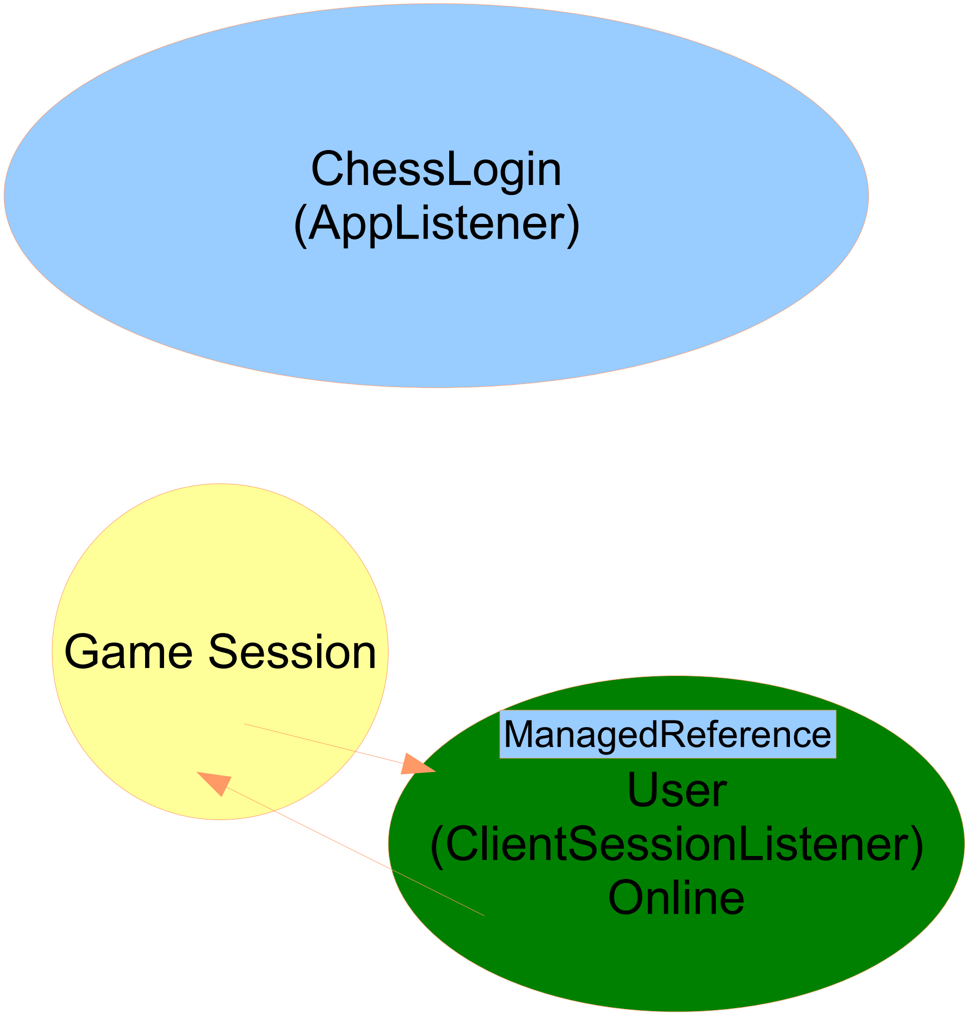
Project Darkstar ChessLogin AppListener

- On loggedIn(...)
  - Lookup User object by bound name
  - Iff User object does not exist
    - Create and bind to name



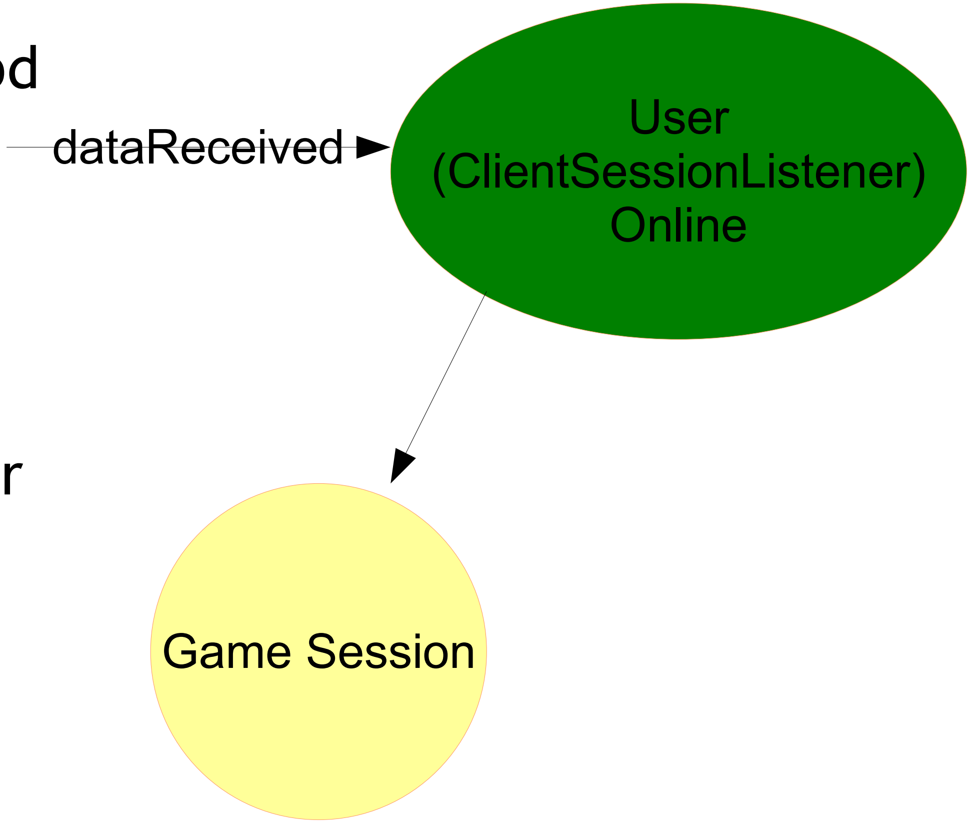
Project Darkstar ChessLogin AppListener

- On loggedIn(...)
  - ...
  - Create game session
  - Add ptr to game session to use
  - Return User object



Project Darkstar ChessLogin AppListener

- On dataReceived(...)
  - User parses message
  - Calls appropriate method on Game Session
- On disconnected()
  - Call playerLeft on Game Session
  - Game Session declares remaining player the winner
  - Game session cleans up

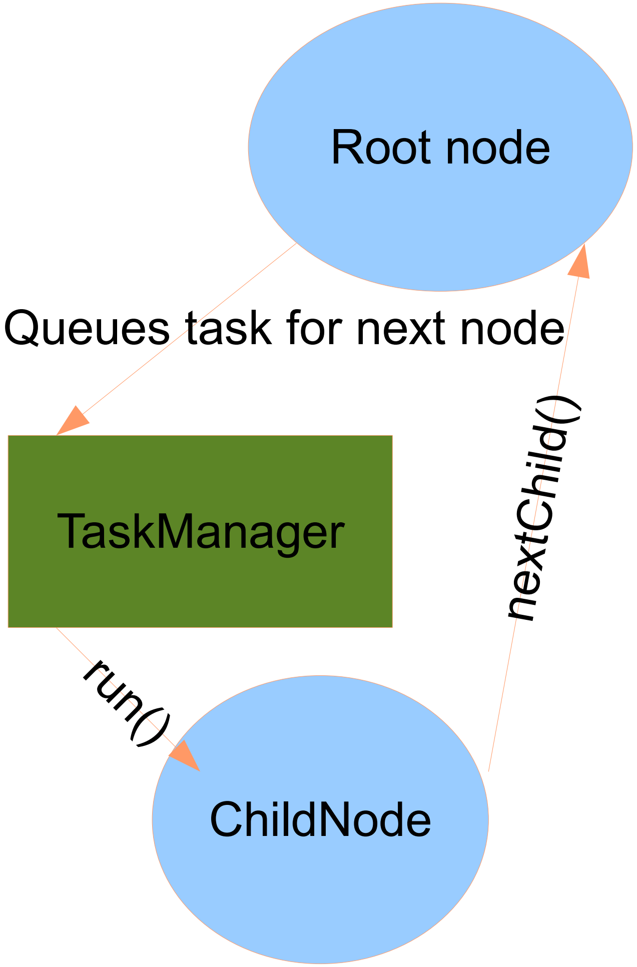


Game Session

- Handles no events
- Has entry points for user objects to call
- Maintains board state
- Runs AI for server's moves

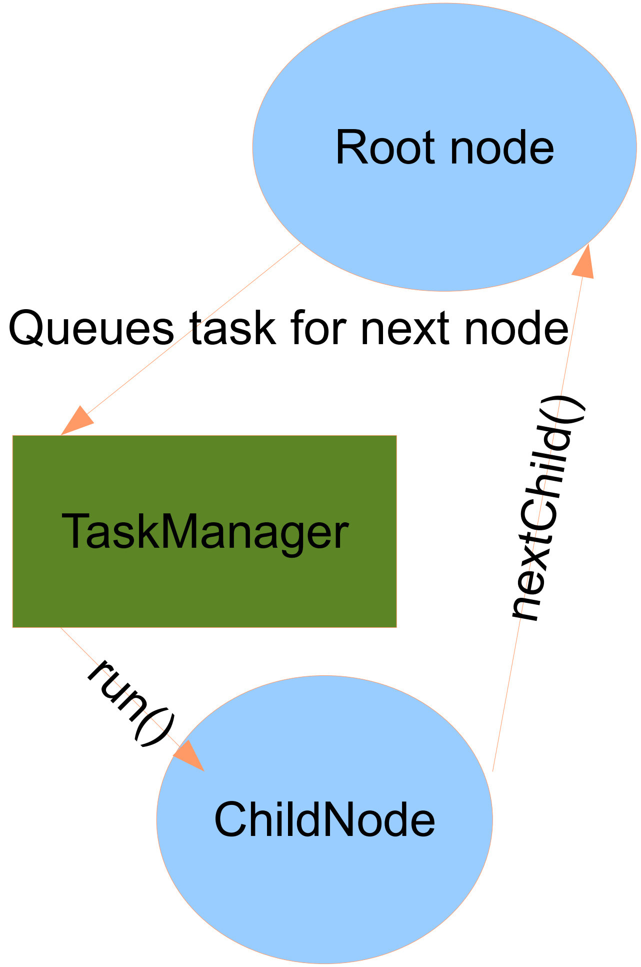
Doing the Chess AI

- Walking the entire move tree will take more then 100ms
- Soln is to break each node's evaluation into a separate task
- Tasks chain using the TaskManager
- Nodes recurse scheduling tasks
- nextChild() calls parent.nextChild() when no children are left to process



Tree Walk code provided to you

- Abstract base class InOrderTreeNode
  - Implement abstract methods
  - Instantiate root node
  - Call root.evaluate()
- Example app included: InOrderWordJumbler


Root node
Queues task for next node
TaskManager
nextChild()
run()
ChildNode