

XtreemOS



*Enabling Linux
for the Grid*

distributed game state management

Workpackage 3.4 & 4.2

Michael Sonnenfroh, University of Düsseldorf
XtreemOS summer school, 11 September 2009

XtreemOS IP project

is funded by the European Commission under contract IST-FP6-033576



Information Society
Technologies





- **Introduction**
 - Overview
 - Basic Architecture
 - Examples
 - Algorithmic View
- **Object Sharing Service**
- **Wissenheim**
- **Conclusion**





Massively Multiuser Virtual Environments



Second Life



EVE Online



World of Warcraft



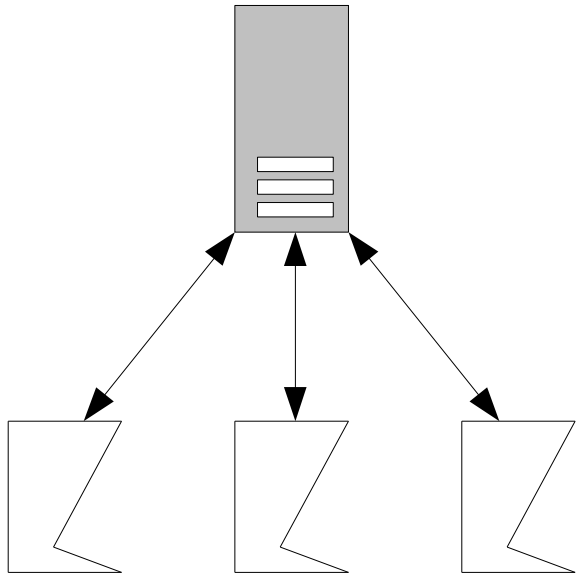
MMVEs business

- **Business Models**
 - pay per month
 - per per item / feature
 - advertisement
- **InGame Business**
 - virtual items
 - land for money
 - virtual factories
 - gold market on ebay



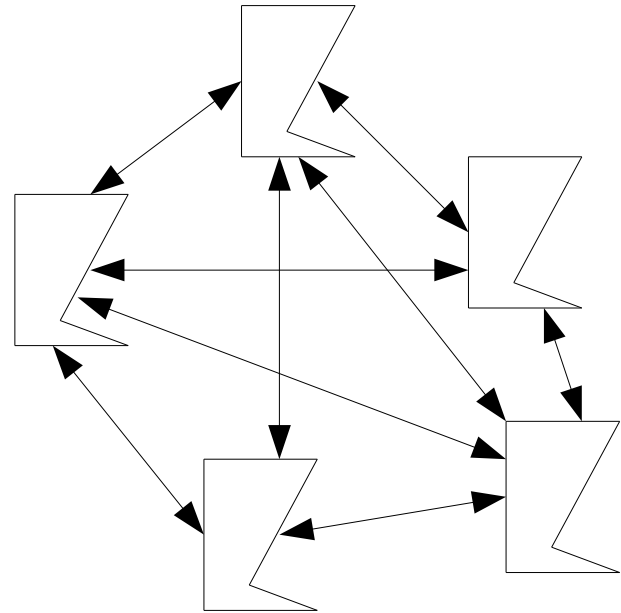


basic network architectures



client / server

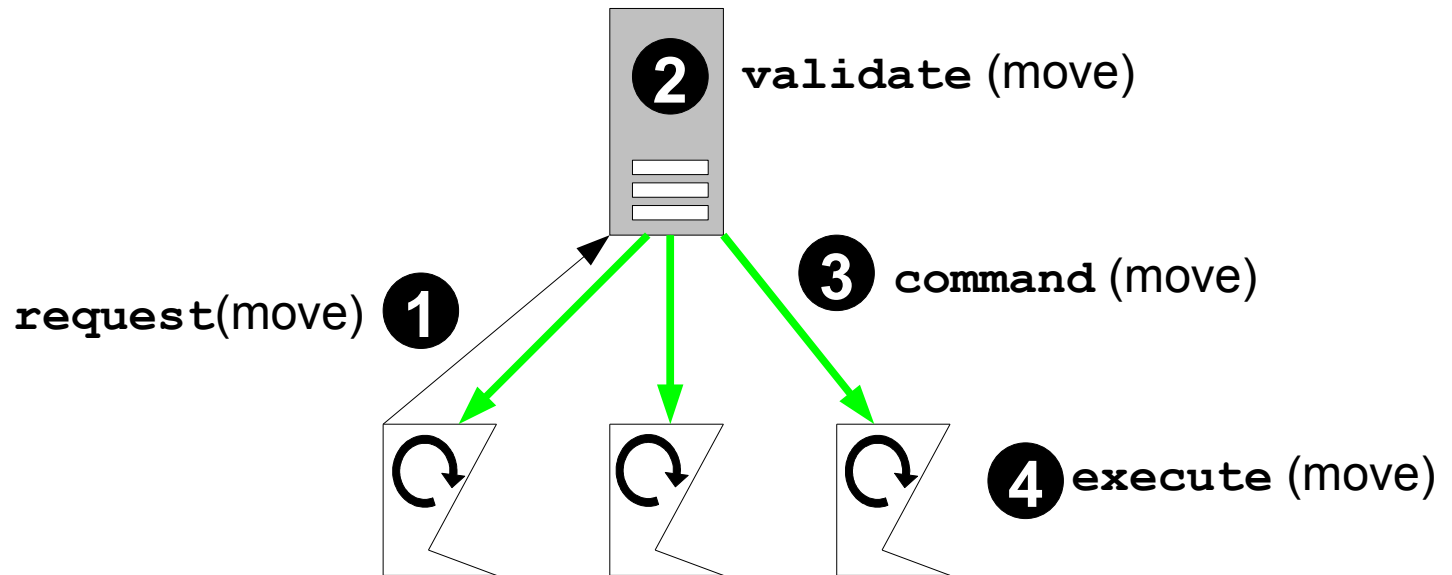
...



peer to peer

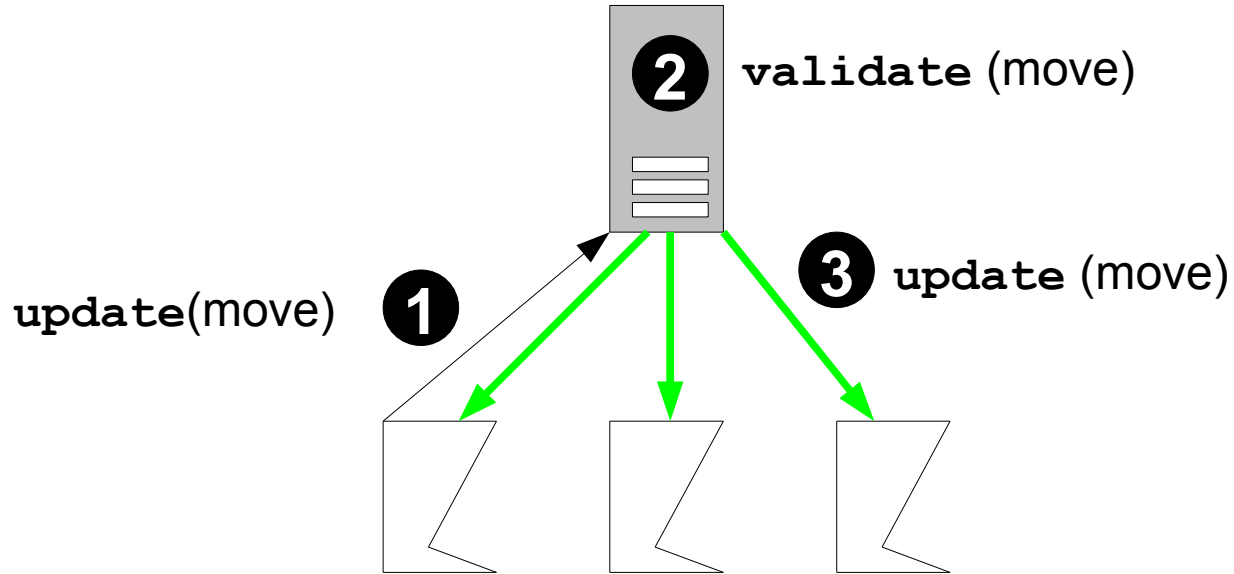


programming models – command based



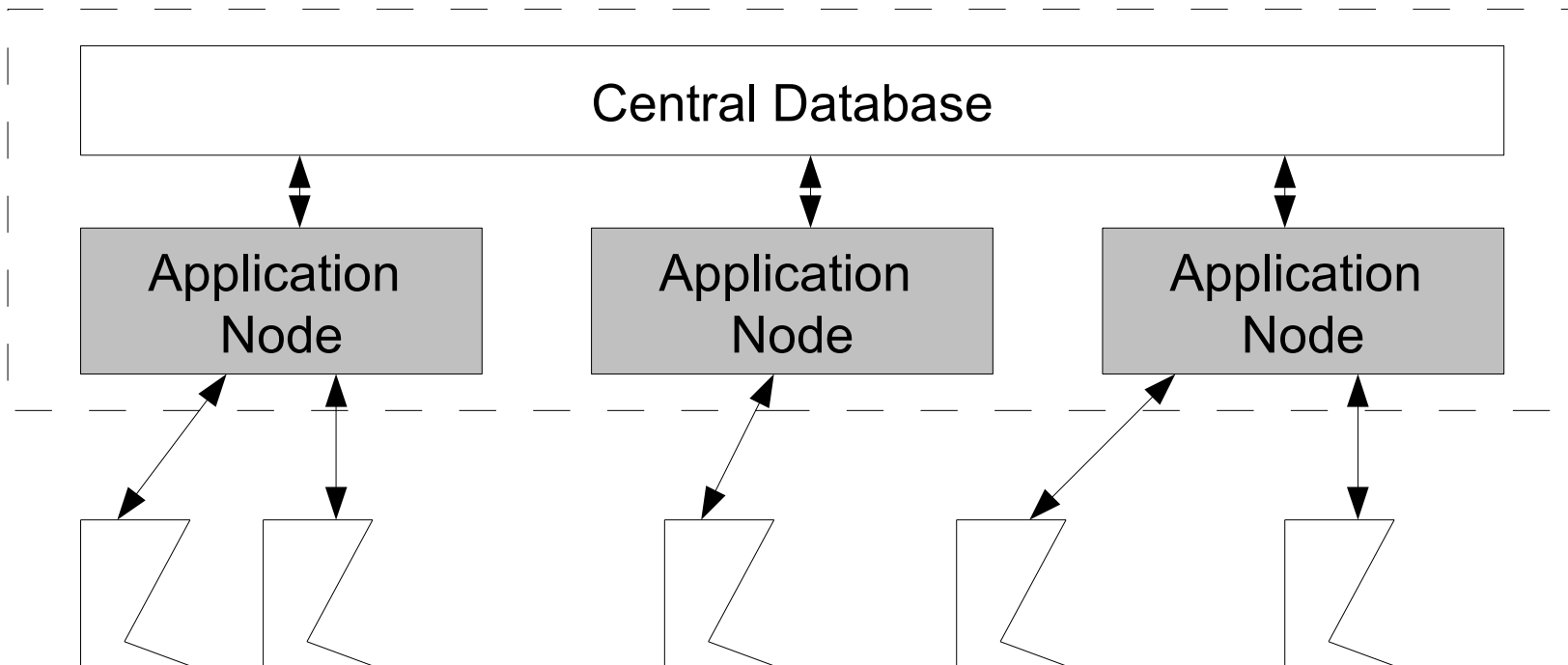


programming models – update based





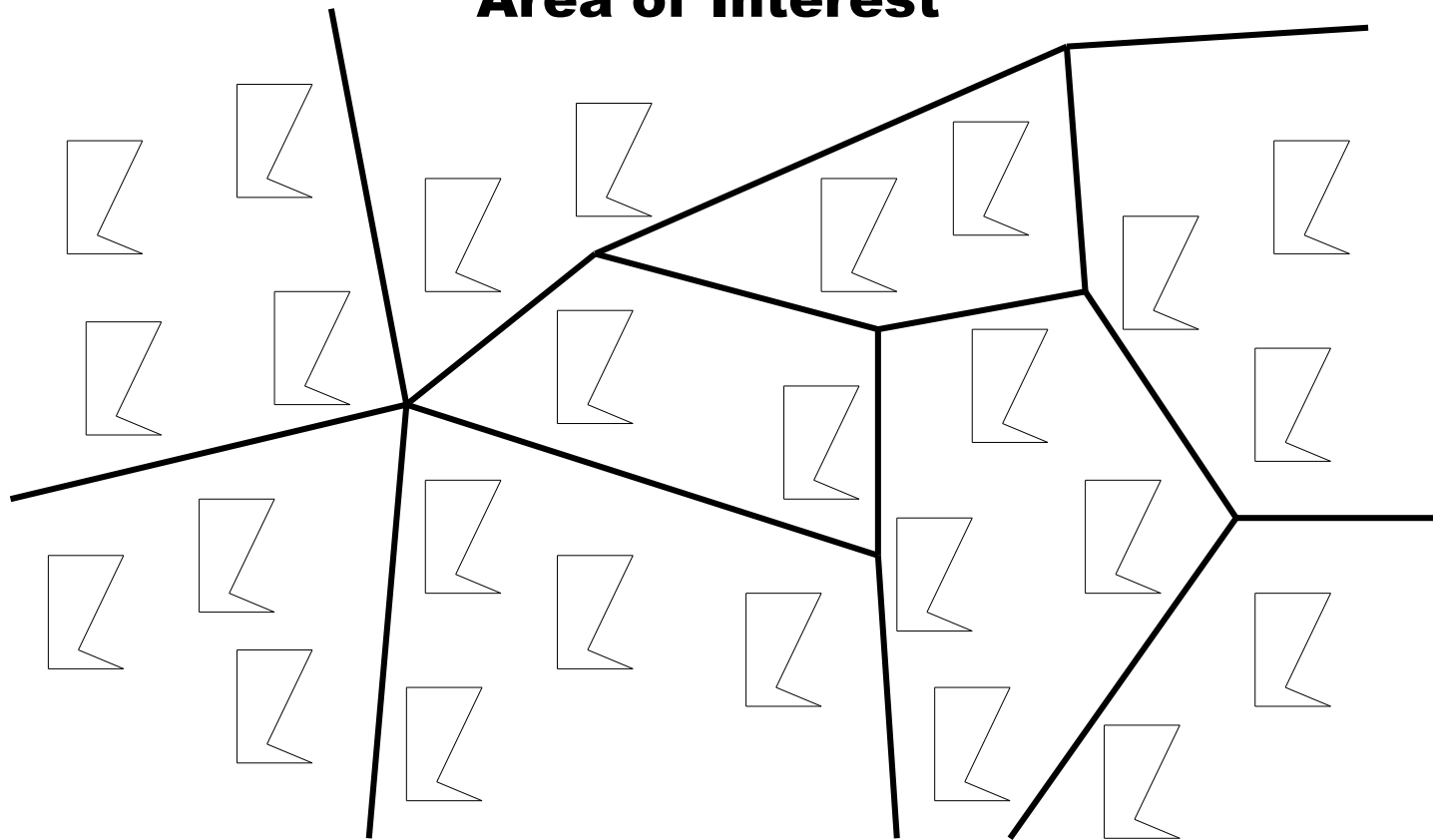
Project Darkstar - network





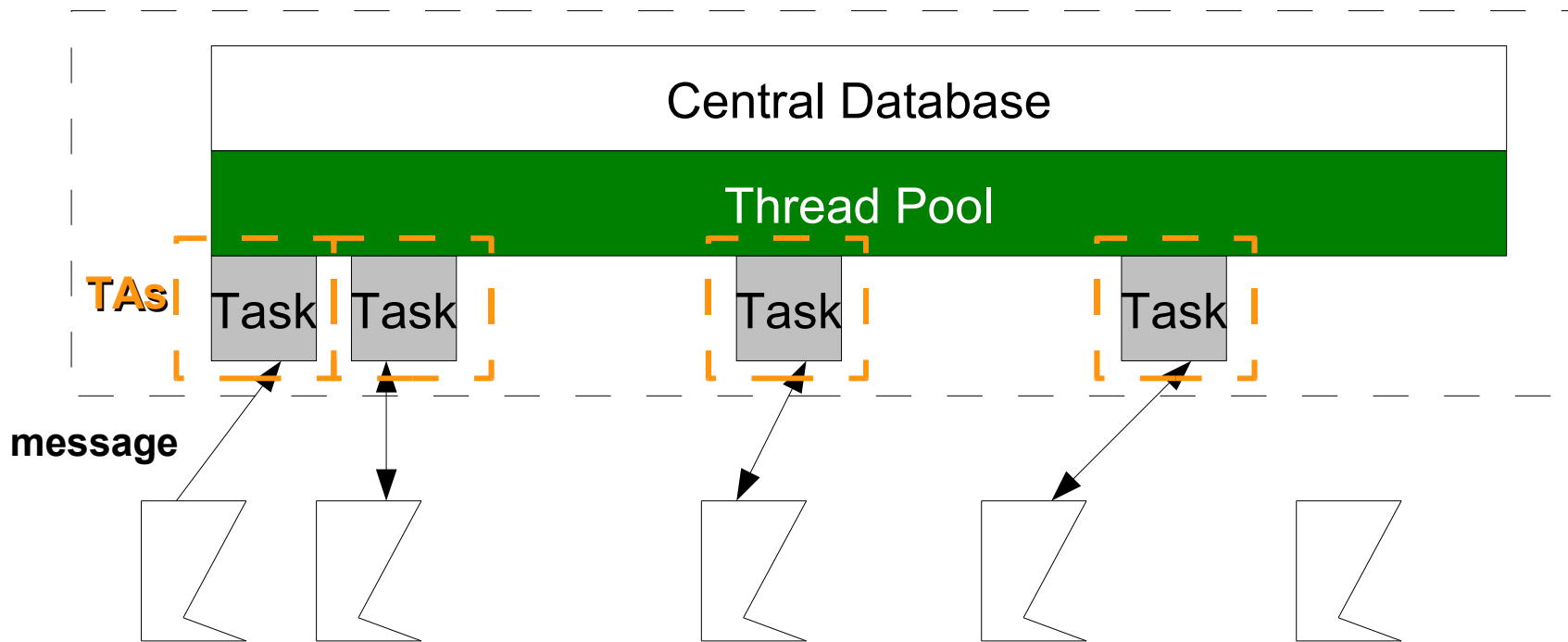
Introduction basic architecture

Area of Interest



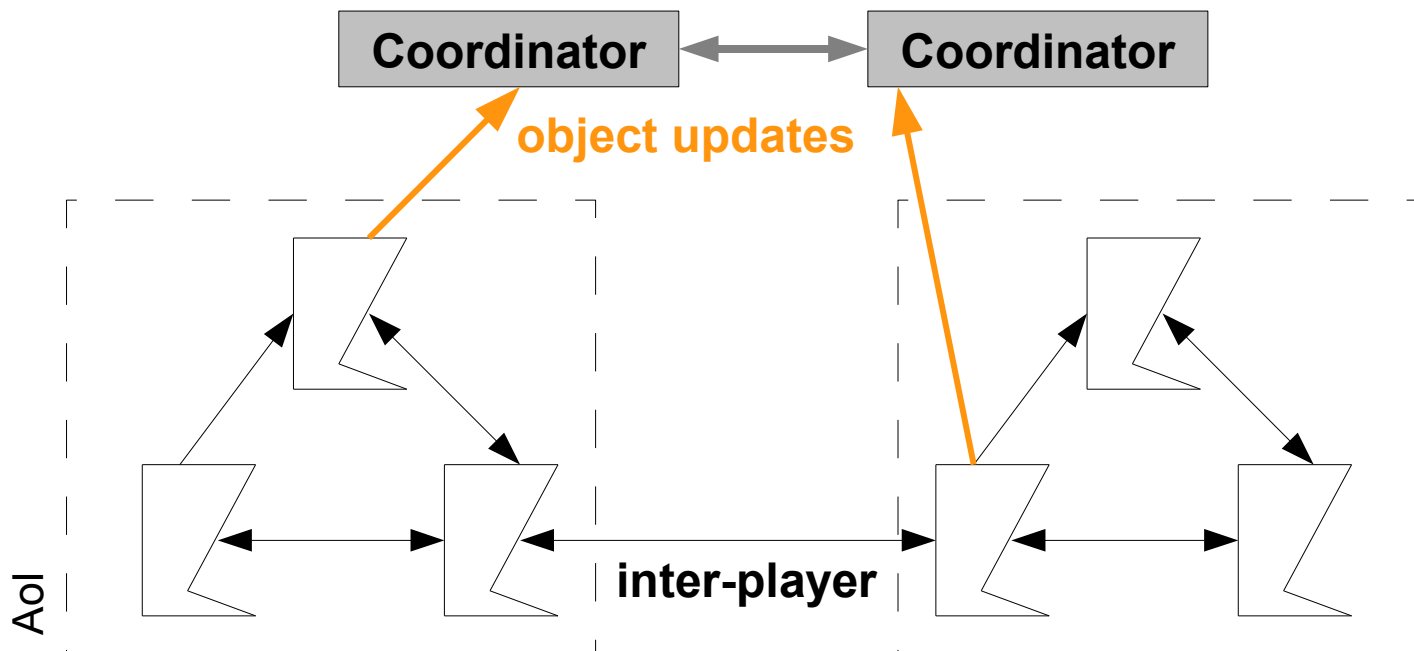


Project Darkstar - threading





SimMud - Overview





Current Approaches

- **Message Based**
 - algorithms must be aware of messages
 - network model is reflected in the algorithms
 - algorithms splitted into receiver / responder
 - consistency based on message automate





Our Approach

- **Object Based**
 - algorithms access distributed & shared objects
 - network model is transparent
 - one algorithm for all participating nodes
 - memory based consistencies





- Introduction
- **Object Sharing Service**
 - Overview
 - Network Architecture
 - Transactions
- Wissenheim
- Conclusion





- **Purpose**

- improve data exchange and consistency maintenance
- complement message-passing
- offer location-transparent, fault-tolerant transactional storage for distributed objects
- different consistency models
- overlay network architecture

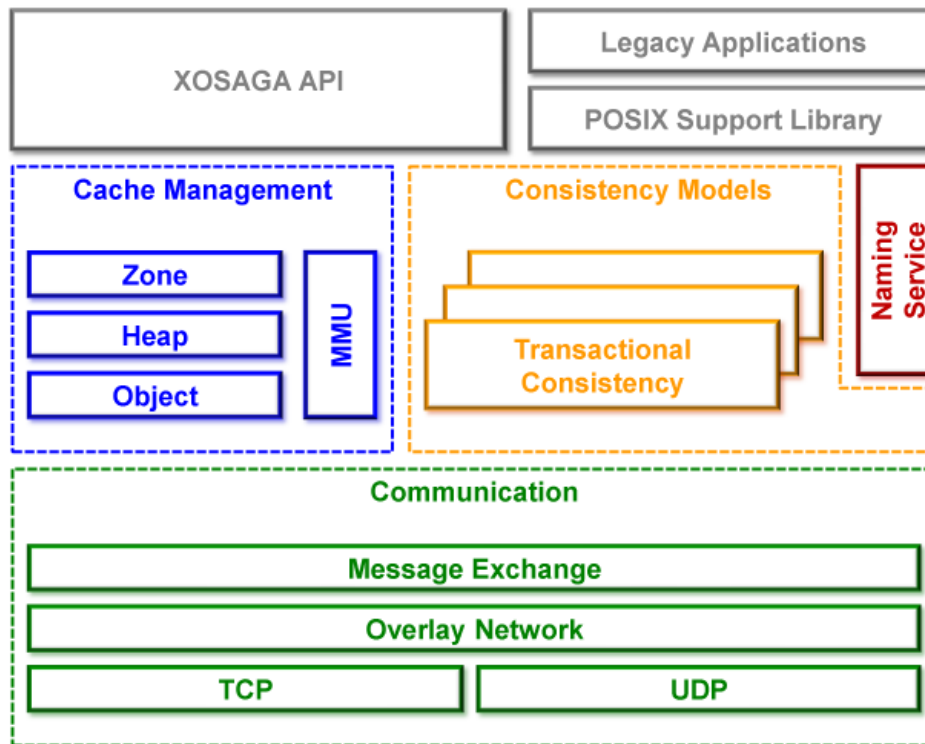
- **Related**

- BlobSeer / JuxMem
- DiSTM
- GigaSpaces



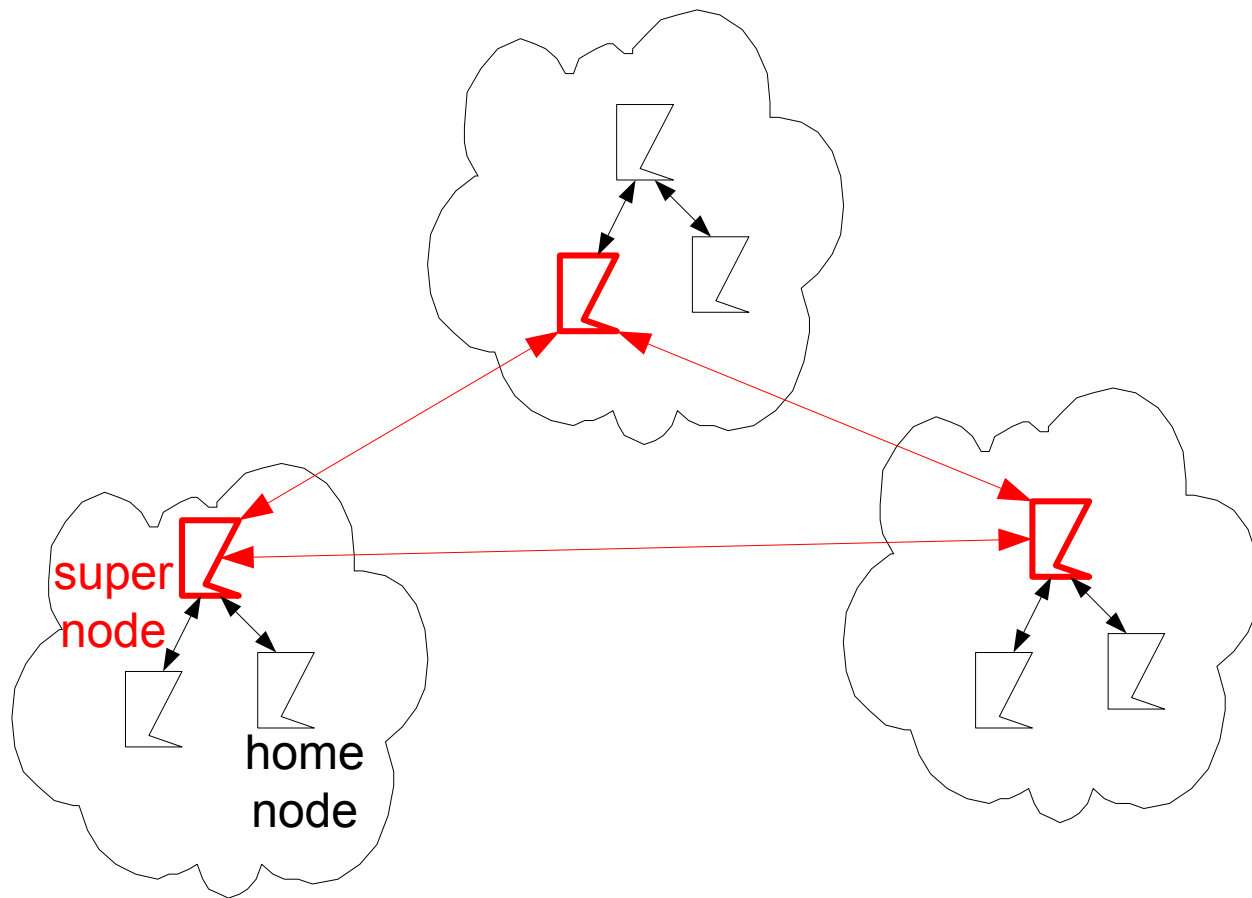


Architecture



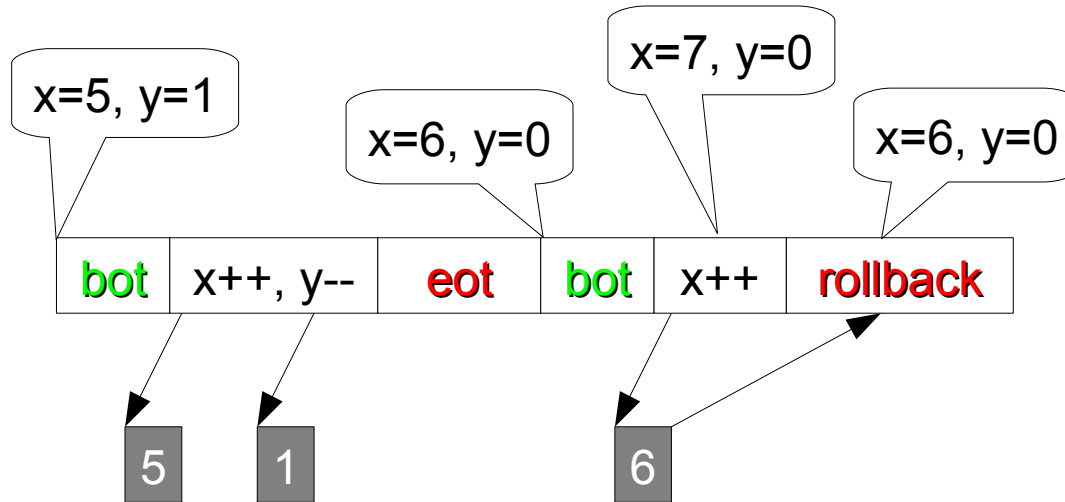


Object Sharing Service network architecture



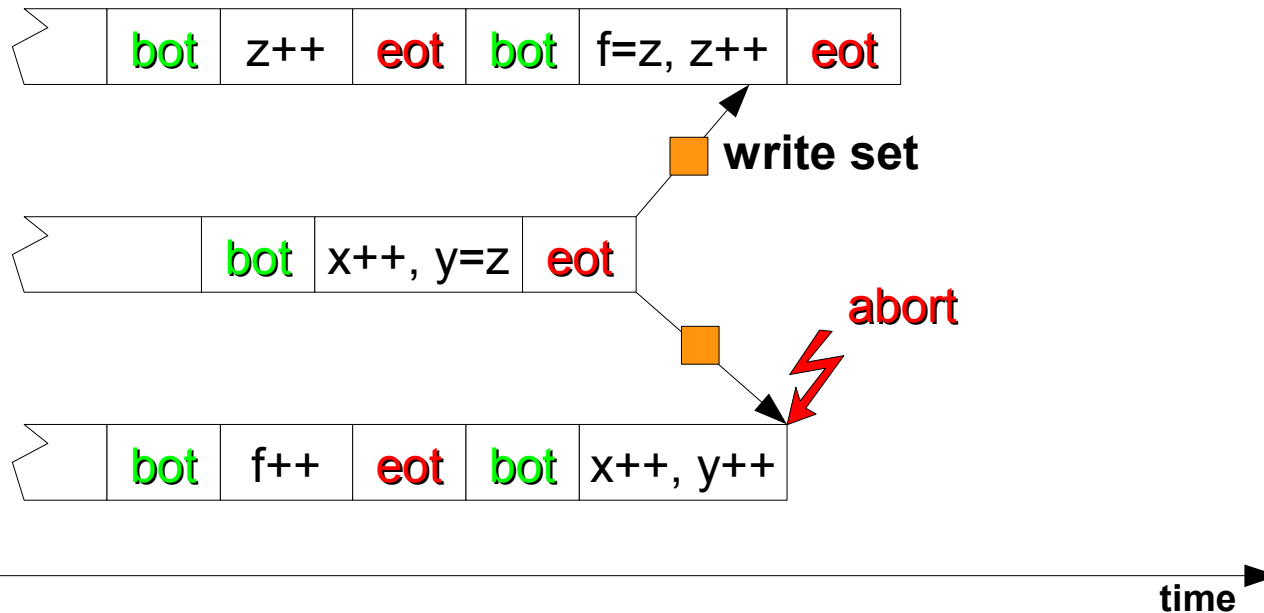


Speculative Transactions





Forward Validation





Usage

```
int i=5;  
bot();  
    printf("Number %d lives\n",i);  
    obj.setPos(12,12);  
    i = 8;  
eot();
```





- Introduction
- Object Sharing Service
- **Wissenheim**
 - Basics
 - Shared Game State
 - Workflow
 - Speed Ups
 - tGOS
- Conclusion



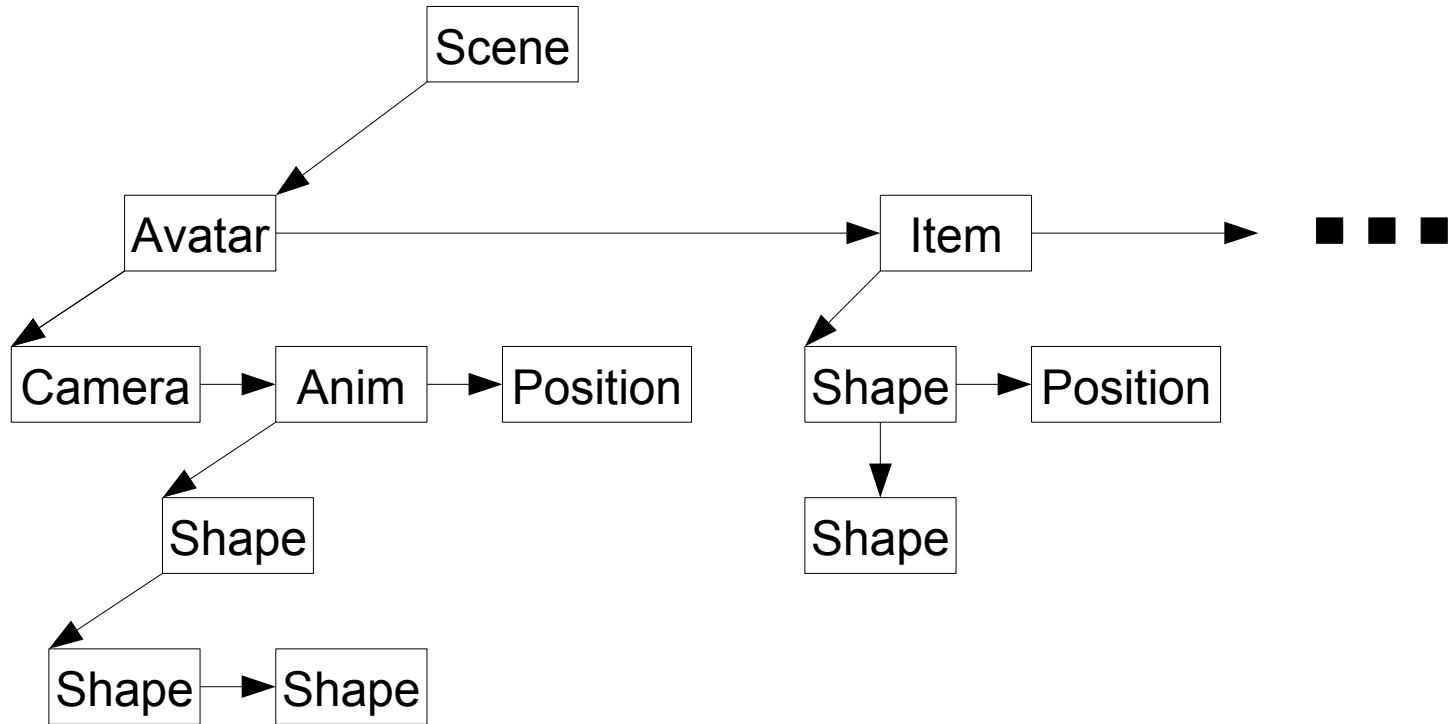


- **Multiuser Virtual Environment**
 - developed at University of Ulm
 - avatar based experience
 - used for teaching content
- **Different Flavours**
 - Sun Java with tGOS
 - native Linux with OSS
 - RainbowOS (in progress)
- **You can visit !**
 - www.wissenheim.de
 - full public access in Oktober



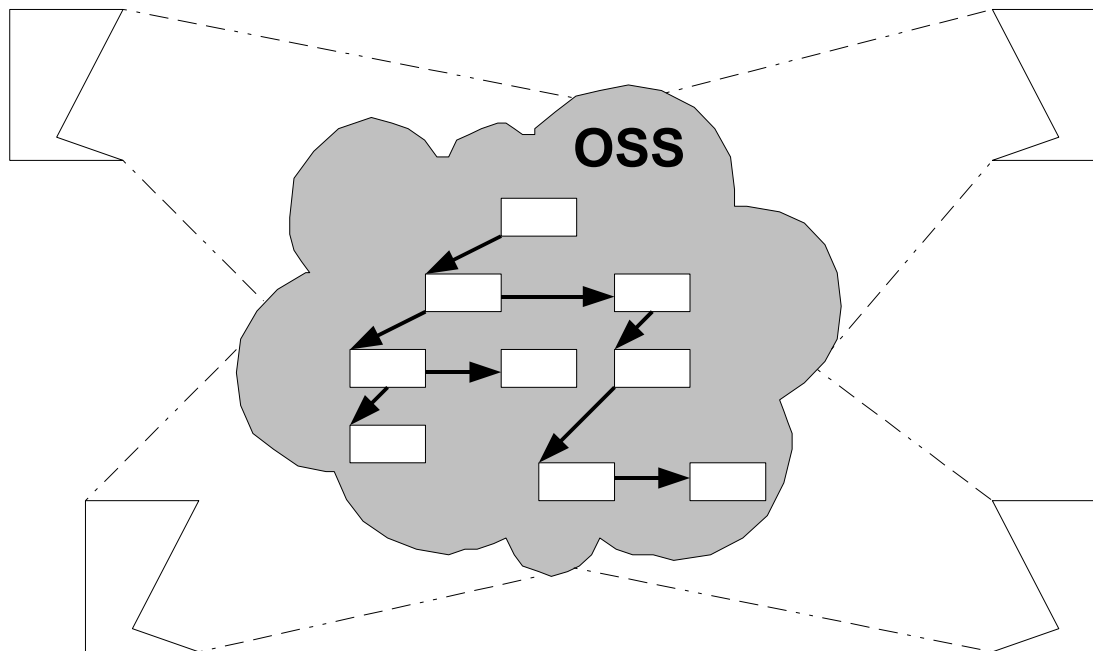


Scene Graph



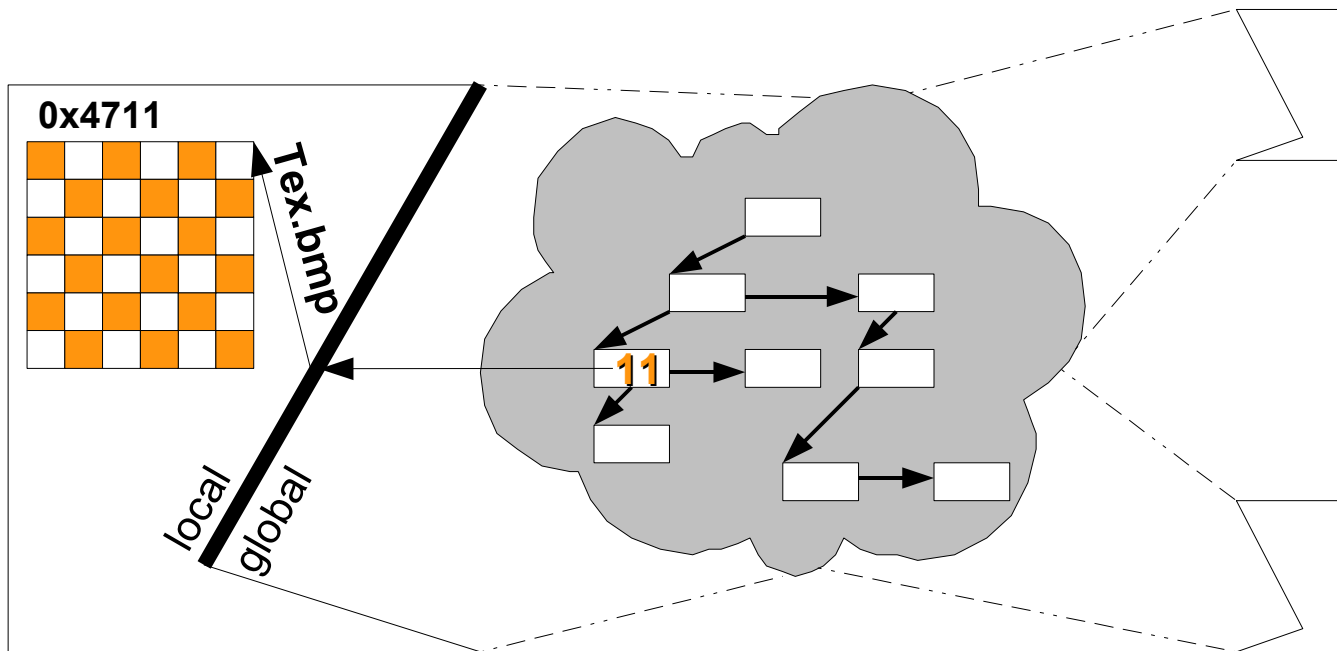


Scene Graph Sharing



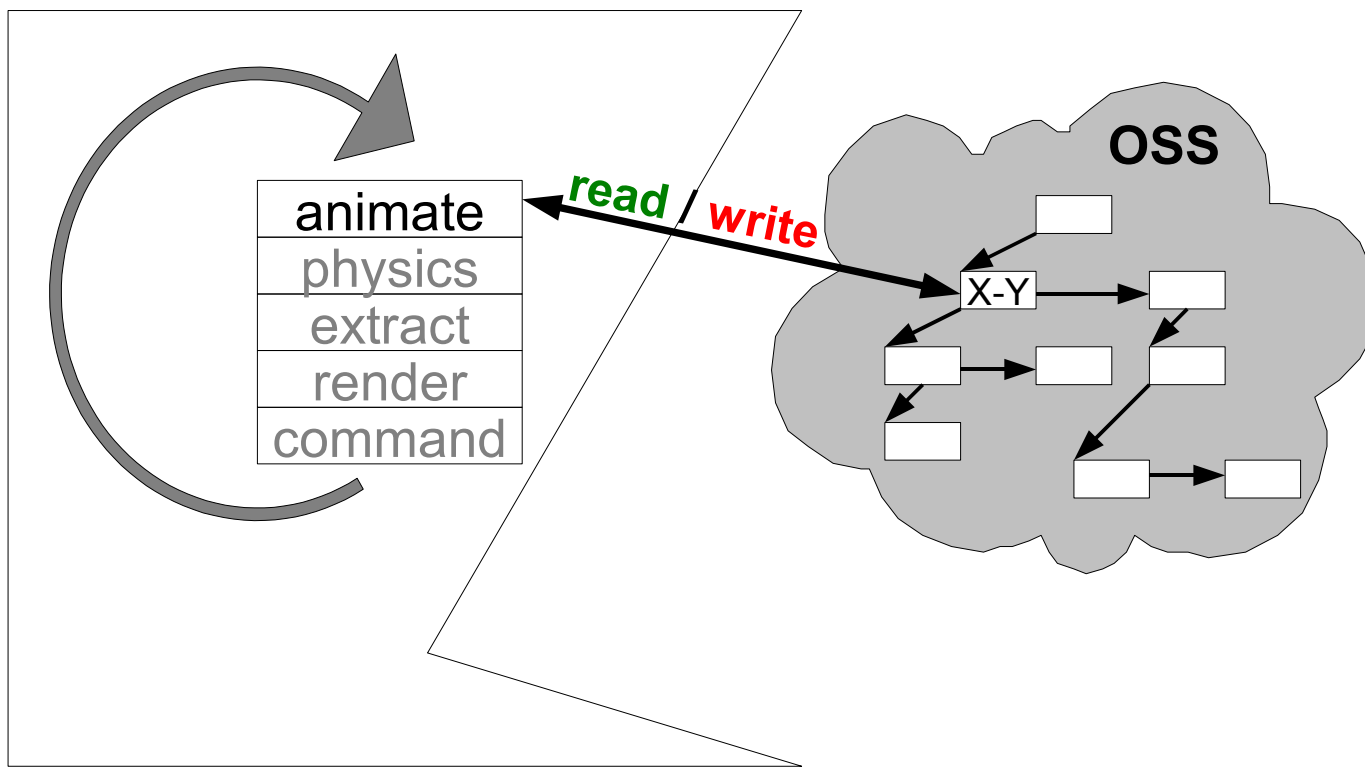


Resources



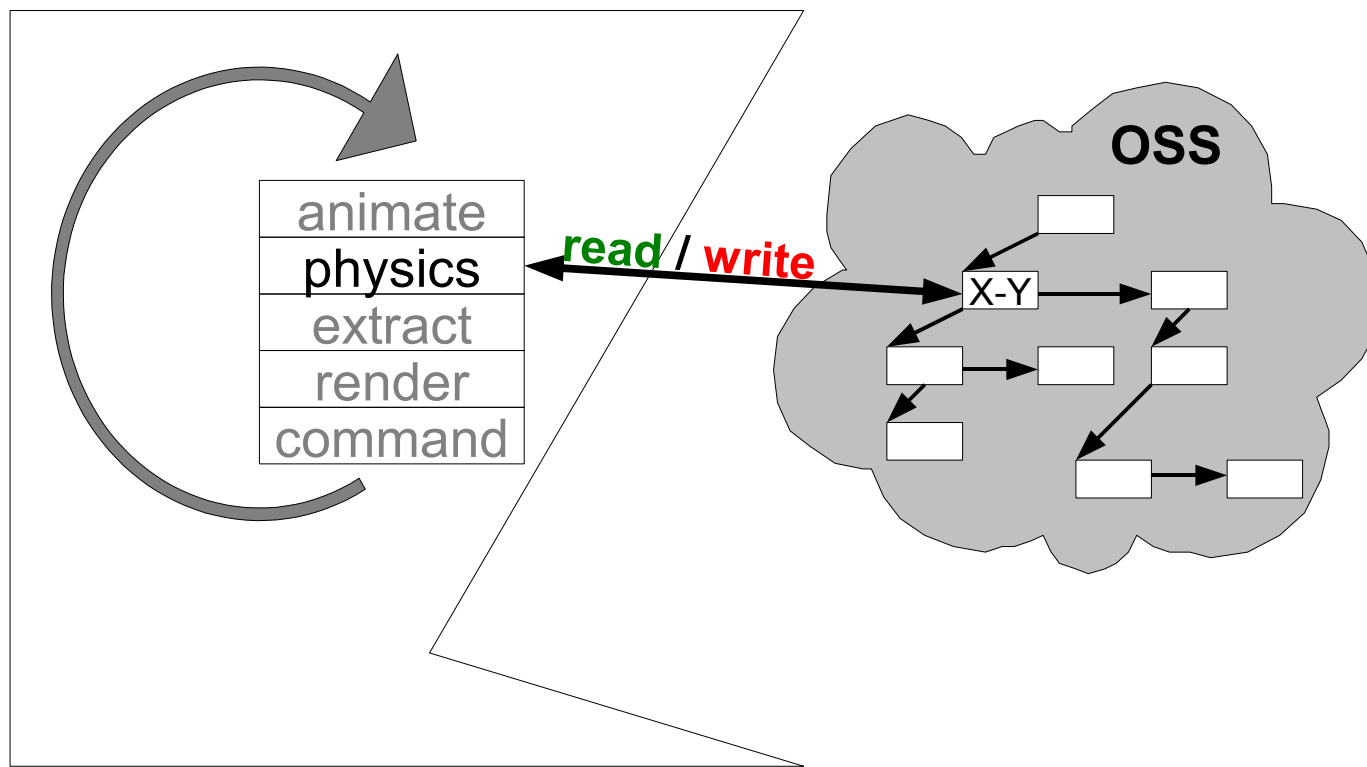


The Loop



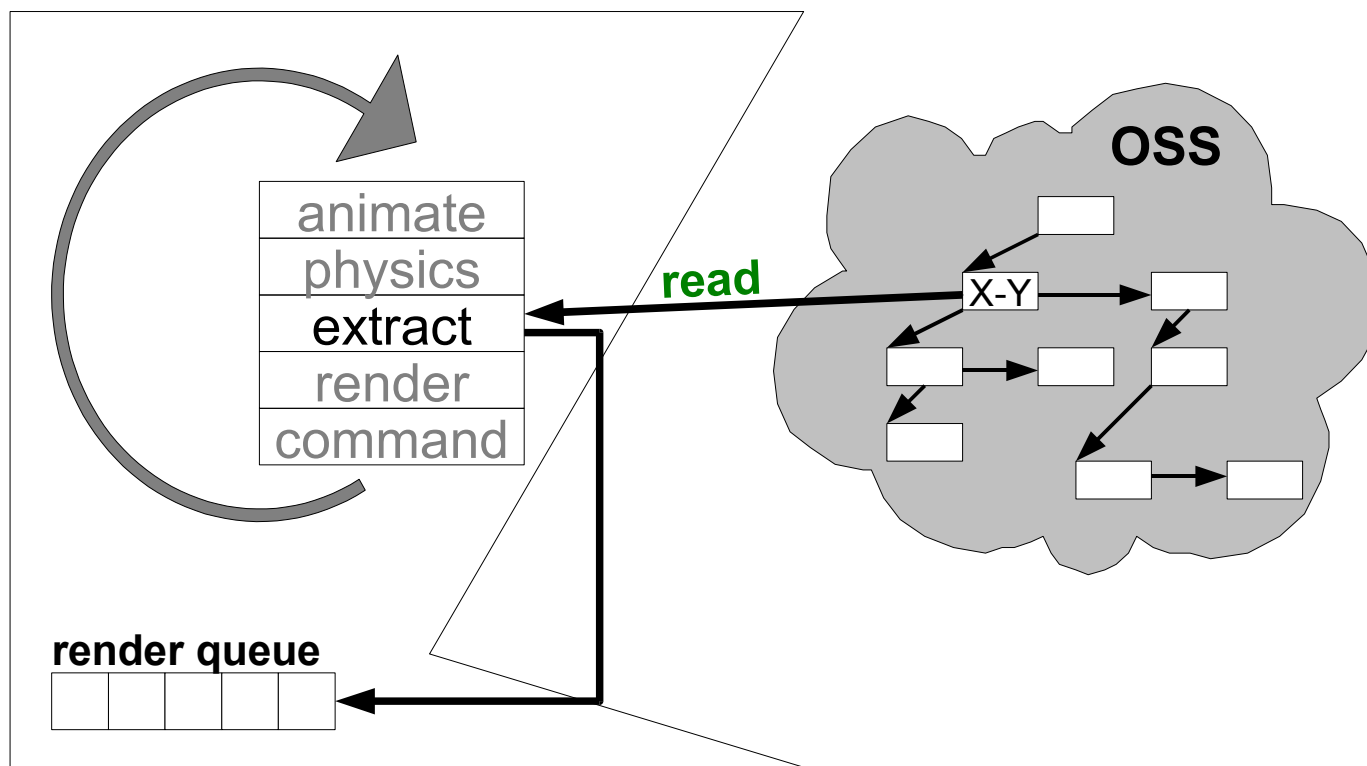


The Loop



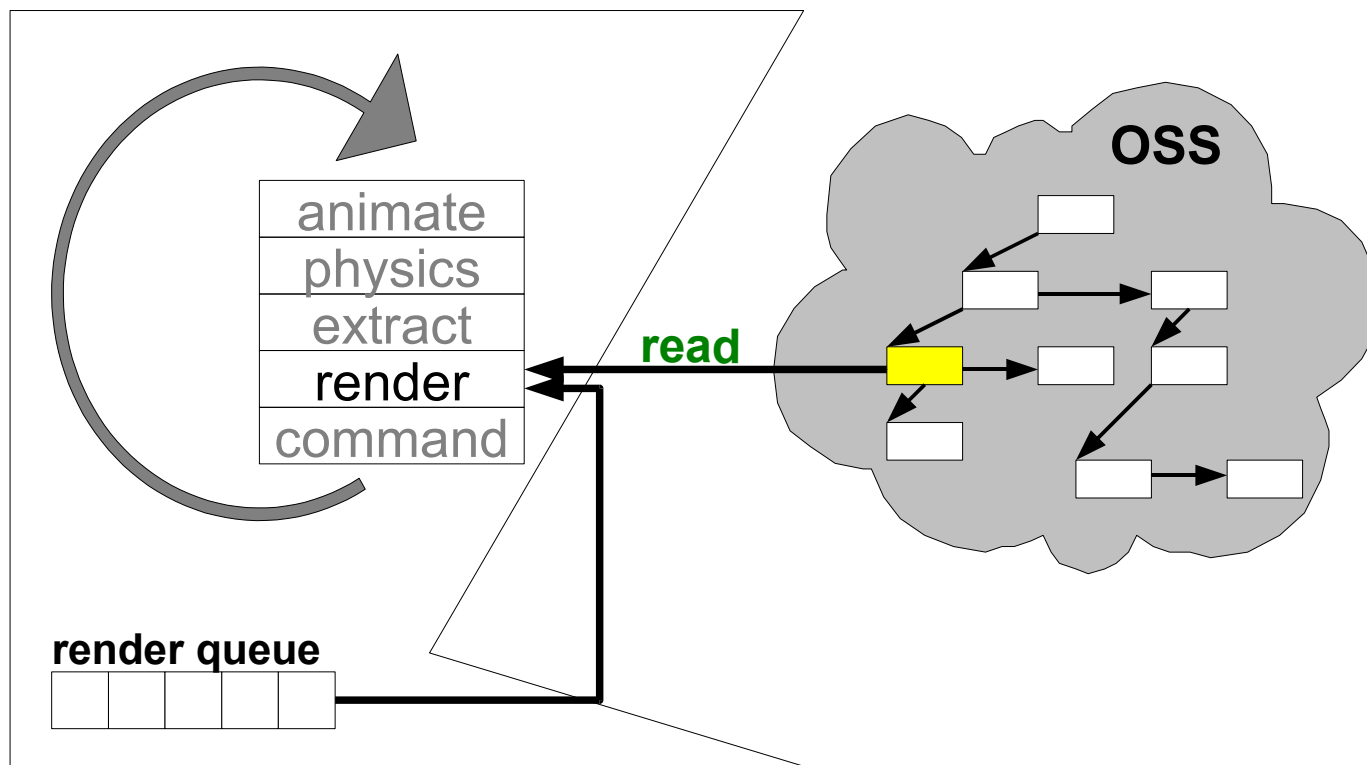


The Loop



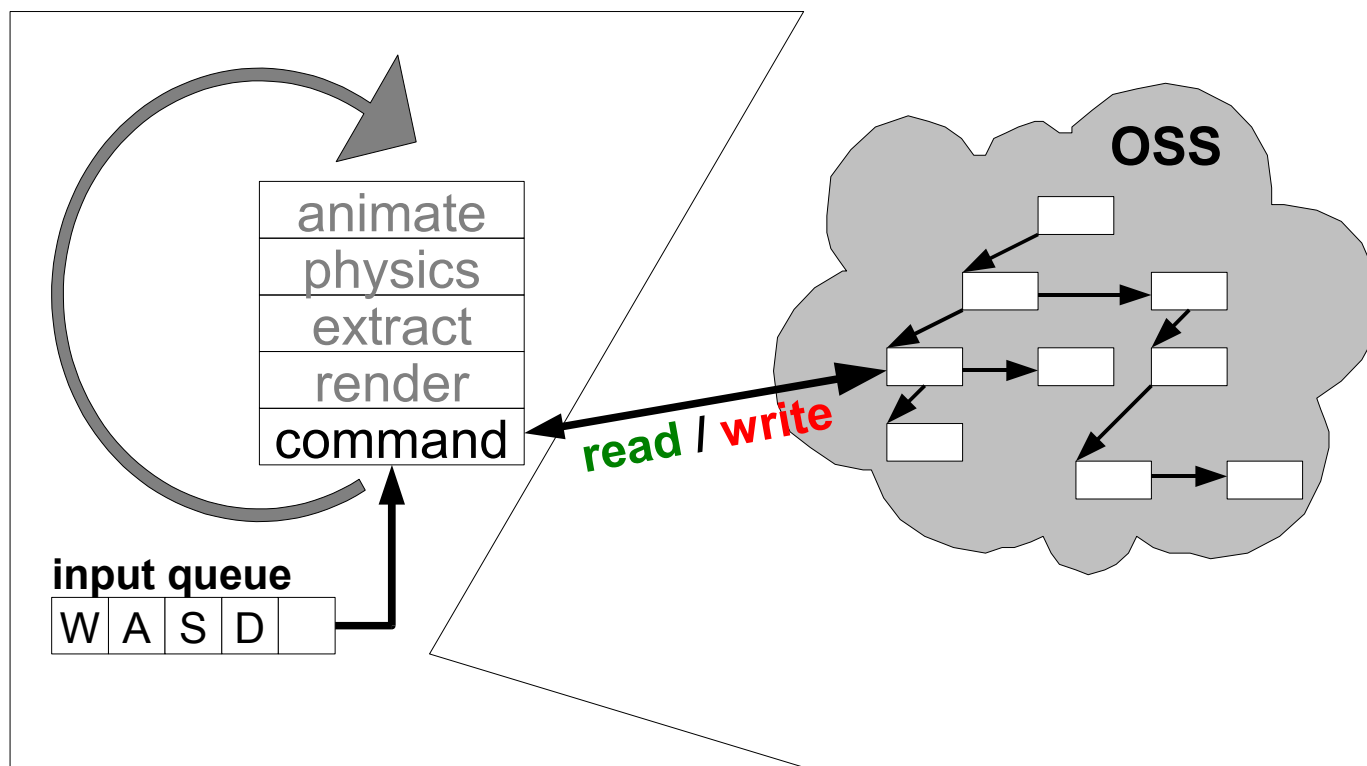


The Loop



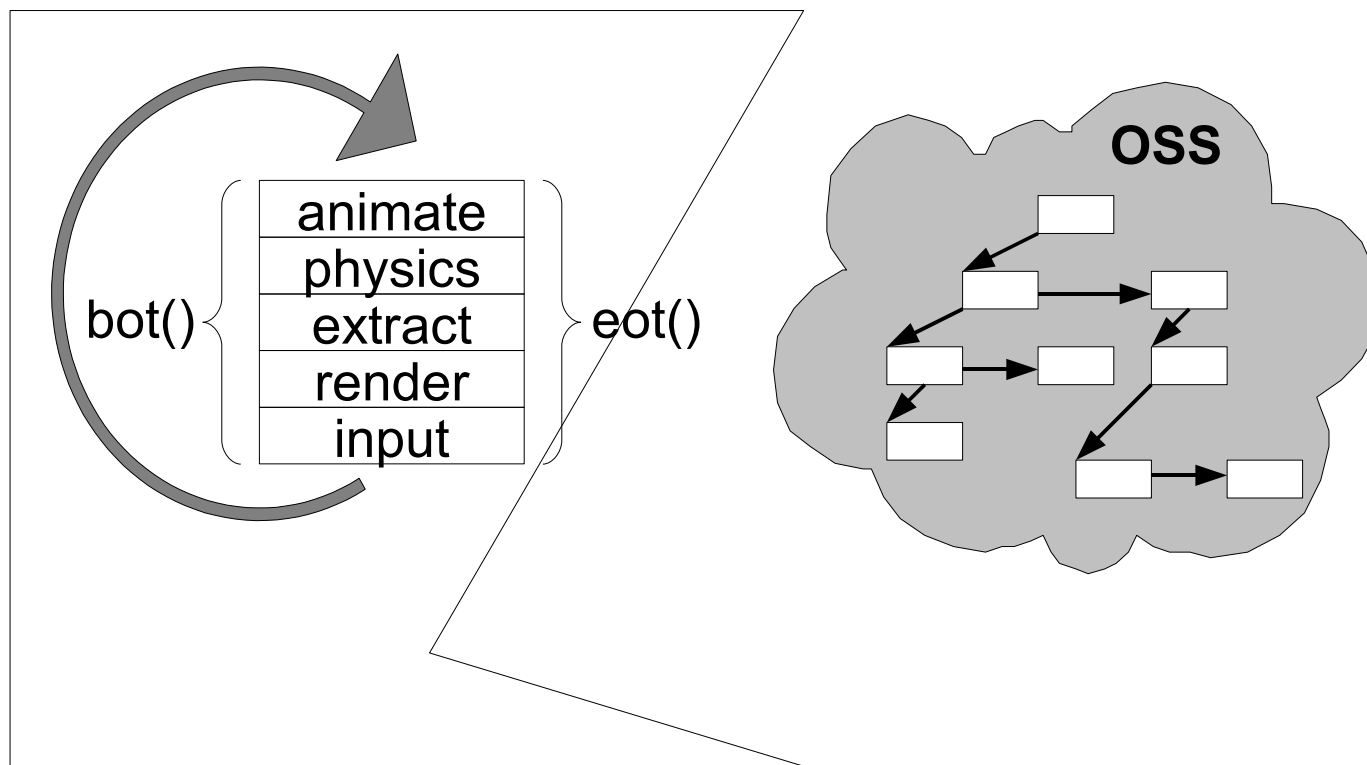


The Loop



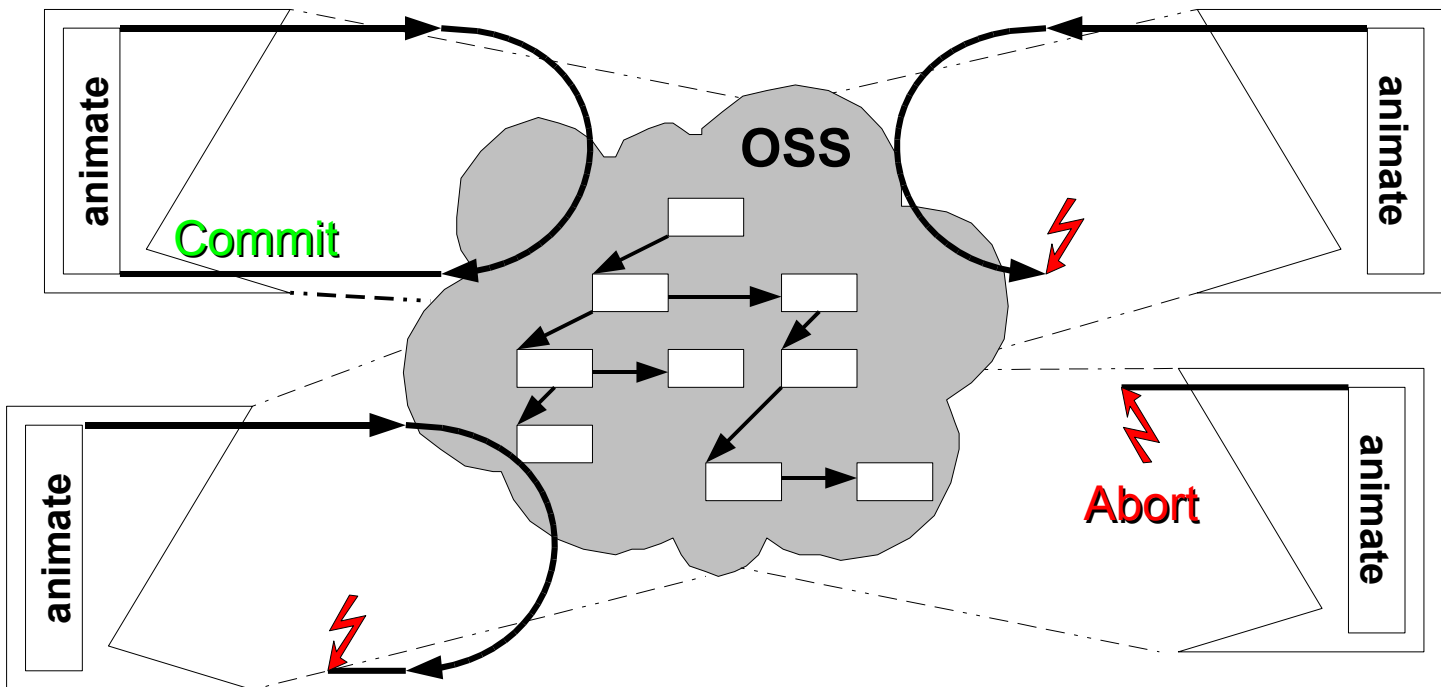


Synchronization



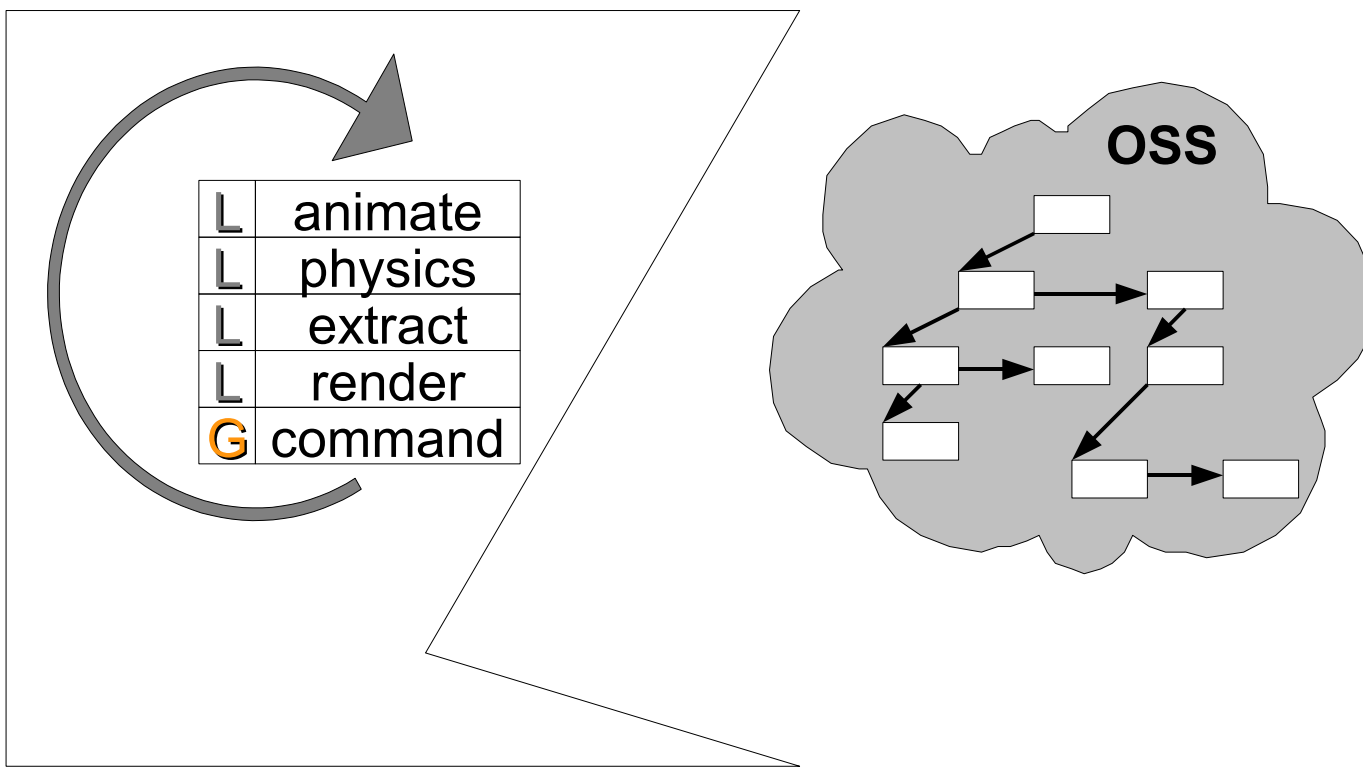


The Problem



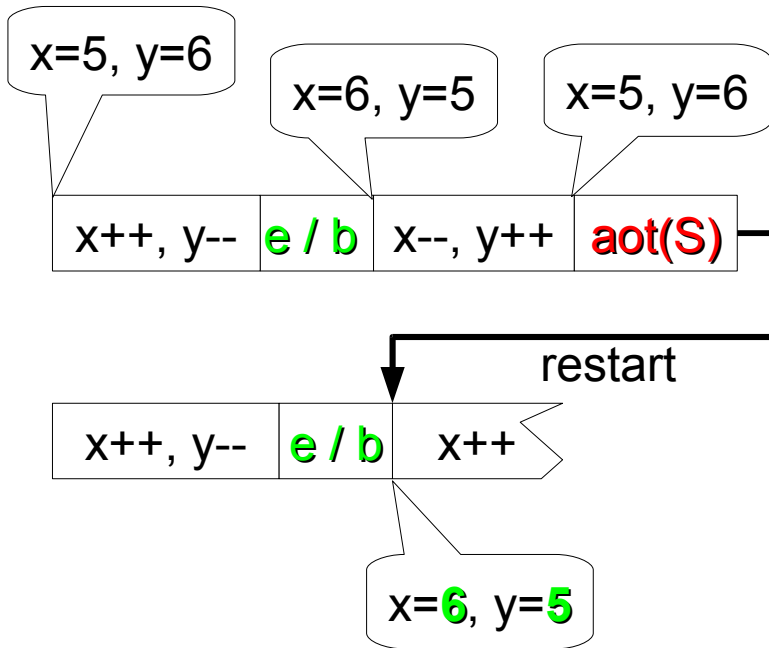


The Loop Revisited

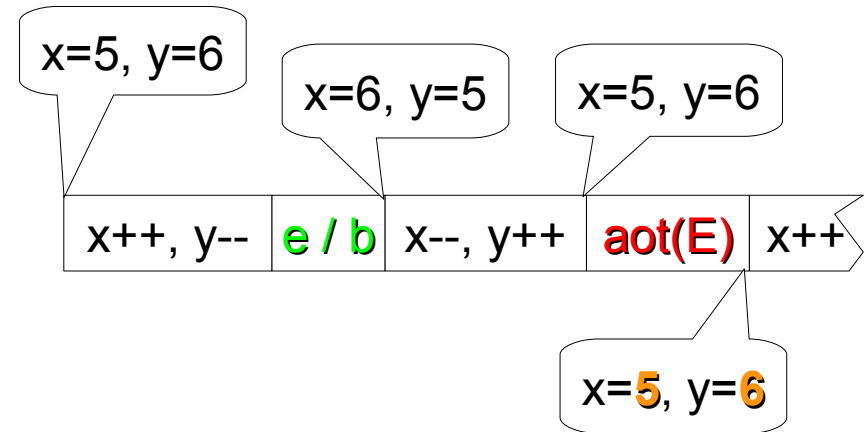




Special Abort Transaction

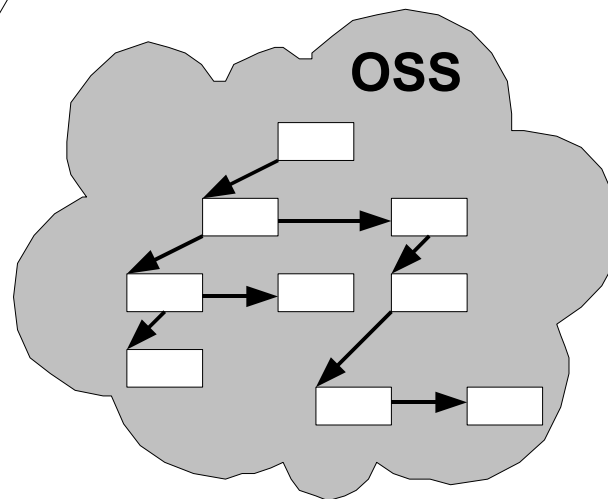
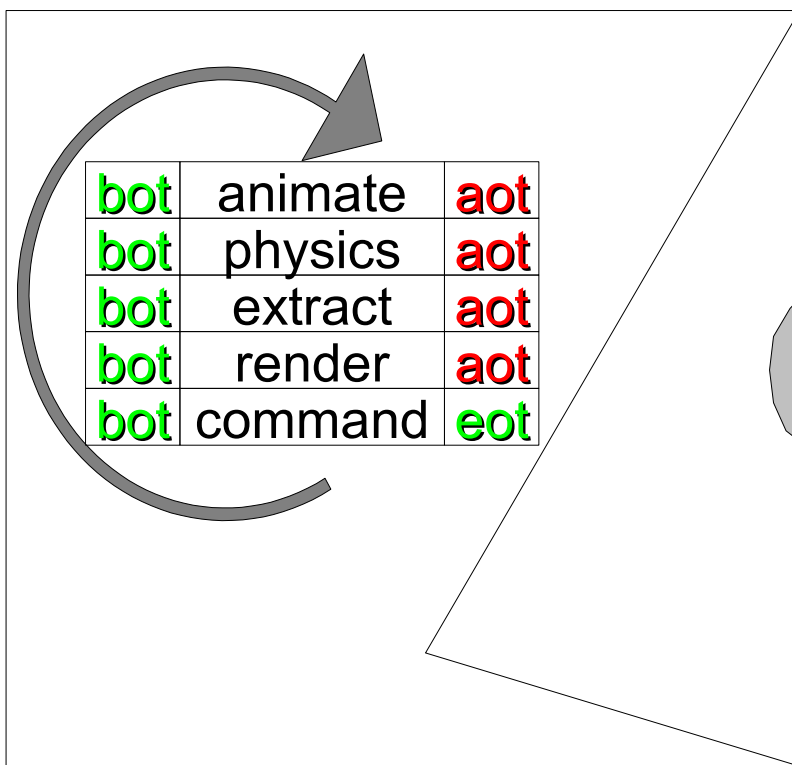


standard
eventual



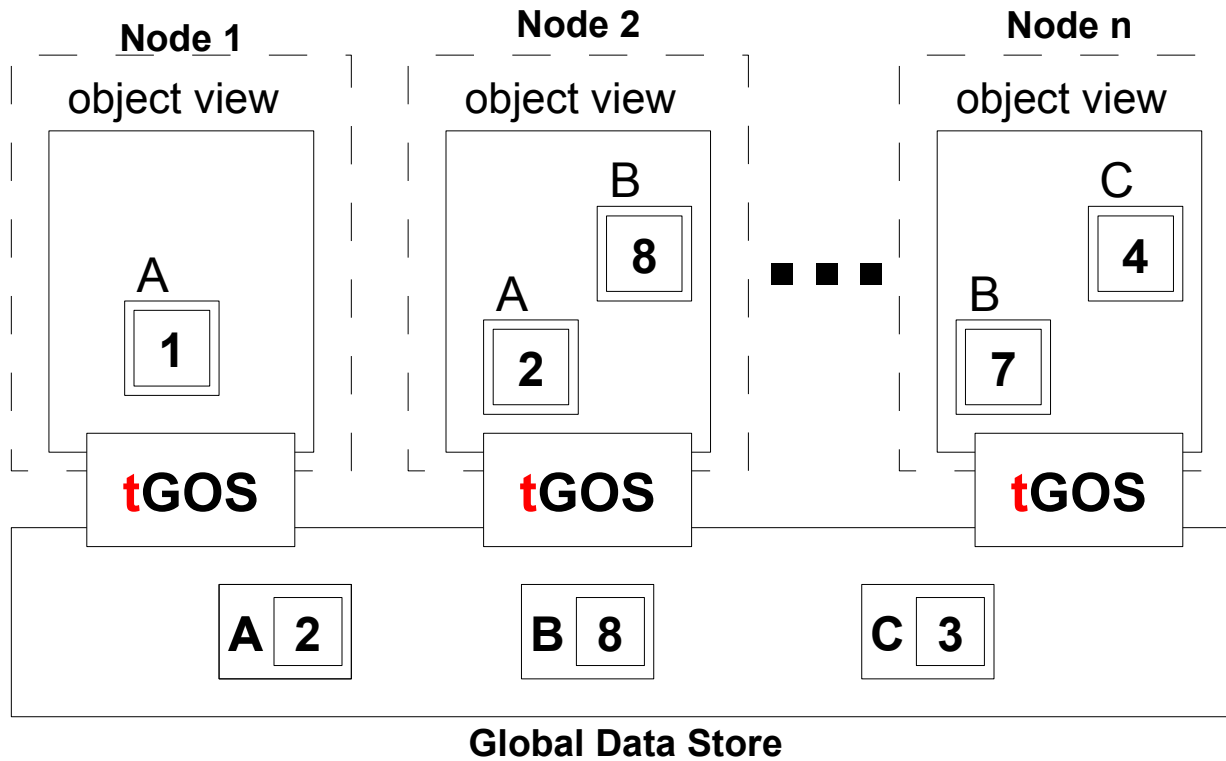


The Eventual Loop





Basic Concept





Basic Operations

Push

Pull

Sync

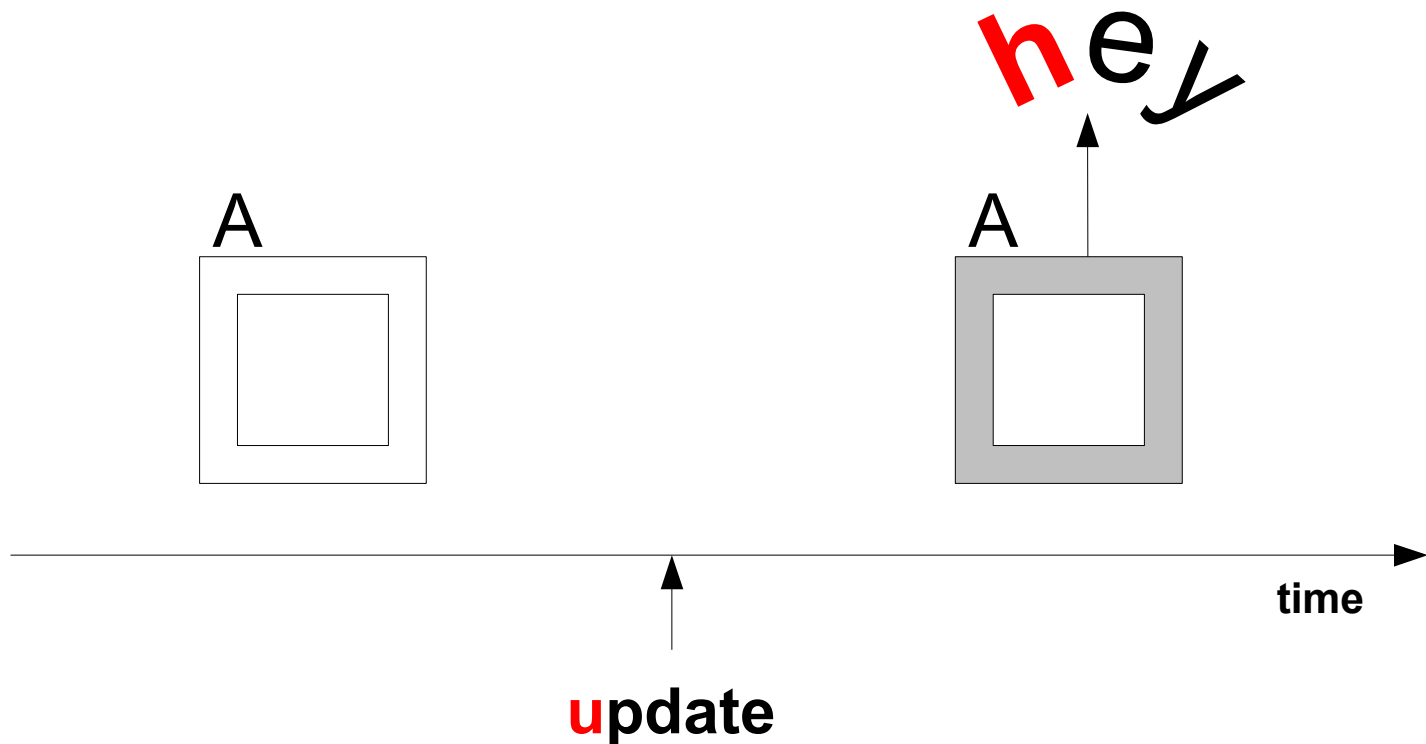
Invalidate

Order



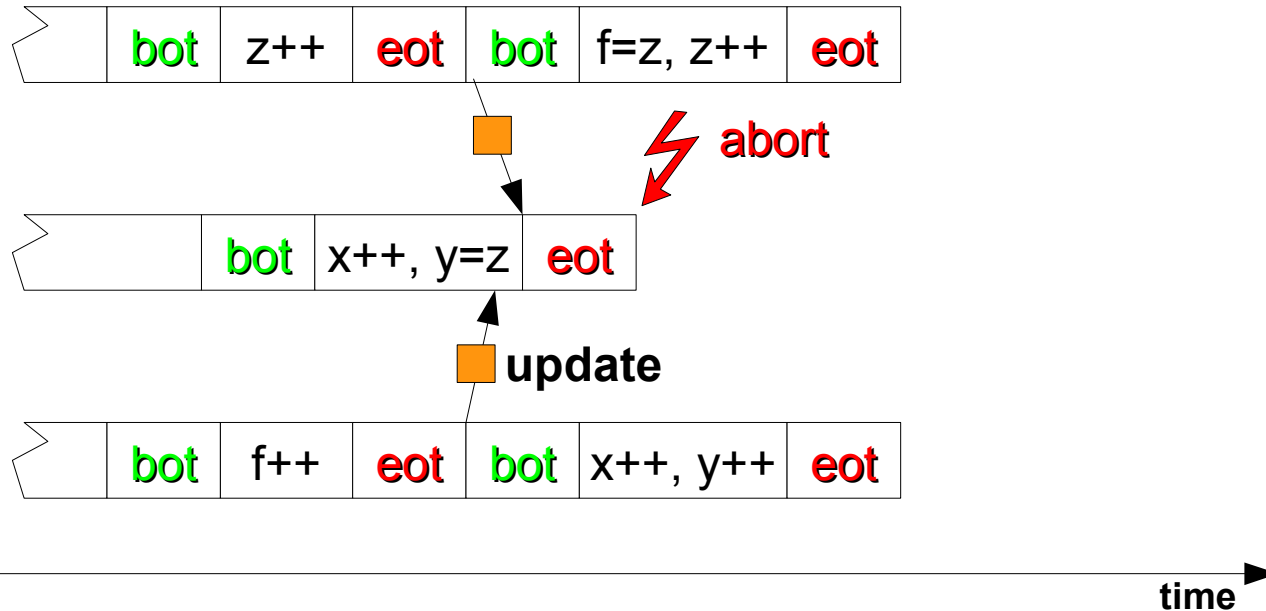


Events





Backward Validation





Transactions

```
int i=5;  
do {  
    begin()  
    add2TA(obj)  
    obj.setPos(12,12);  
    i = 8;  
} while (!commit());
```





- Introduction
- Object Sharing Service
- Wissenheim
- **Conclusion**
 - Transactions
 - Object Based Approach





■ General

- easy to use
- feasible for consistency management of multiple objects
- guarantee high level of consistency
- fast in low conflict environment
- optimistic approach is deadlock free
- fine grain usage possible

■ OSS

- automatic read / write set generation
- scalability through Super Nodes

■ tGOS

- complementary to OSS
- multiple consistency domains (Aol)
- update mechanism





- **Object Based Approach**
 - frees the algorithm from the network
 - network model is exchangeable
 - one-node-view
 - more focus on consistency
 - message based programming also possible





XtreemOS



*Enabling Linux
for the Grid*



distributed game state management

Workpackage 3.4 & 4.2

Michael Sonnenfroh, University of Düsseldorf
XtreemOS summer school, 11 September 2009



*XtreemOS IP project
is funded by the European Commission under contract IST-FP6-033576*





- **Introduction**
 - Overview
 - Basic Architecture
 - Examples
 - Algorithmic View
- **Object Sharing Service**
- **Wissenheim**
- **Conclusion**





Massively Multiuser Virtual Environments



Second Life



EVE Online



World of Warcraft



- Massively Multiuser Virtual Environments are game scenarios or virtual worlds build to provided interactions between a huge amount of human players
- Second Life (SL) is a well known, free to use Getaway
- World of Warcraft (WoW) is the leading MMORPG with about eleven million paying users, in contrast to SL, WoW is a classical game design which has been developed to support large player numbers
- EVE Online is a extreme MMVE with concurrent user numbers in the thousands, currently a combination with a console based ego shooter is developed



MMVEs business

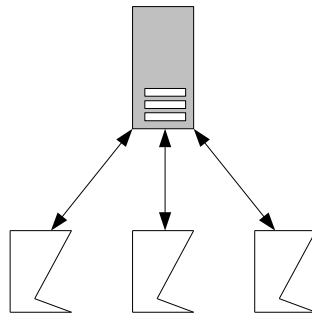
- **Business Models**
 - pay per month
 - per per item / feature
 - advertisement
- **InGame Business**
 - virtual items
 - land for money
 - virtual factories
 - gold market on ebay



- monthly based payment is a model used by many high end MMVEs like WoW, Lord of the Rings, EVE Online
- Second Life has a flourishing virtual business where people are buying virtually crafted items, Linden Labs is selling virtual land
- second market for virtual items like gold trading via eBay for games like WoW
- complex business models inside MMVEs (with problems like inflation)

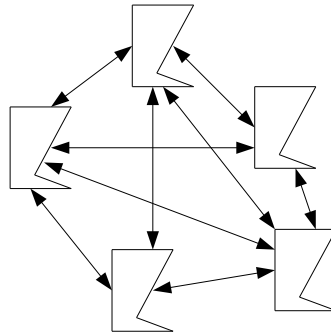


basic network architectures



client / server

...



peer to peer



- in principal we have two different network architectural types, client /server and peer to peer (and many combinations in between)
- client / server is the dominating type in conjunction with MMVEs (e.g. WoW and SL are using this type)
- client / server very useful for security and account management (central control)
- large server clusters are required, therefore high costs
- peer to peer uses no central component, better resilience against failure but security is hard to achieve, scalability problems if fully meshed => hybrid models

XtremOS
Enabling Linux for the Grid

Introduction
basic architecture

programming models – command based

request(move) ①

② validate (move)

③ command (move)

④ execute (move)

Information Society Technologies

11/09/2009
Oxford

XtremOS IP project
is funded by the European Commission under contract IST-FP6-033576

6

Command Based Architecture

- 1) client sends move request to server
- 2) server validates the request (collision detection, etc.)
- 3) server sends a new command to all clients
- 4) every client is executing the command

- Used for example in realtime strategy (RTS) games, where one command can effect many units
- deterministic command execution necessary on all nodes, nondeterministics can lead to butterfly effects
- e.g. used in Age of Empires (1-3), Command & Conquer etc.

XtremOS
Enabling Linux for the Grid

Introduction
basic architecture

programming models – update based

update(move) **1**

2 validate (move)

3 update (move)

Information Society Technologies

11/09/2009
Oxford

XtremOS IP project
is funded by the European Commission under contract IST-FP6-033576

7

Update Based Architecture

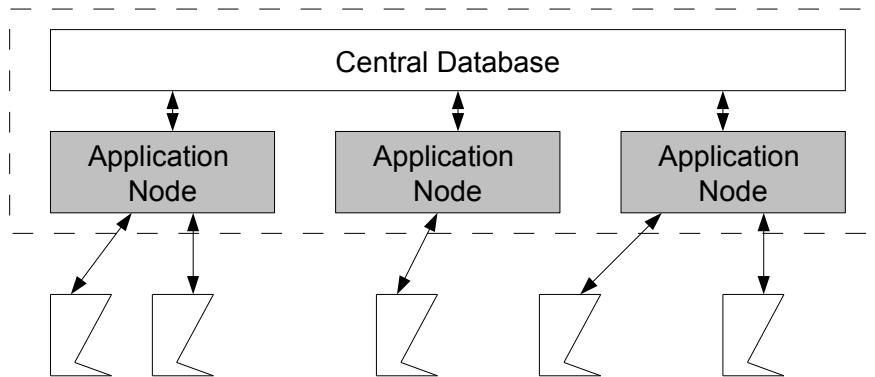
- 1) client sends new new position of an avatar (or any other updated information)
- 2) server validates the request (collision detection, etc.)
- 3) server sends the position to all clients

- Used for example in ego shooters to update the position of an avatar, mixed forms are very common, were a new position along with a moving vector is send to allow dead reckoning

- validation is difficult


- higher bandwidth needed, server my be used as relay only

- e.g. used in Quake,

**Project Darkstar - network**

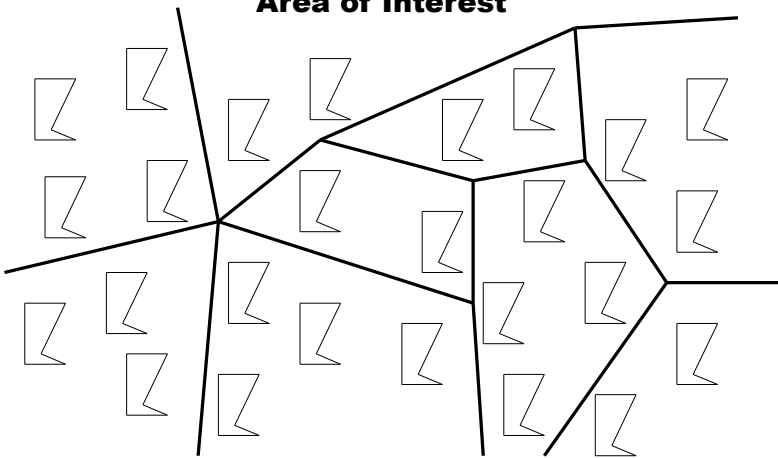
- Open Source MMVE frame work by Sun Microsystems
- allows the classical client / server model
- provides a simple interface for clients and for the server application (login, logout, messages)
- provides group communications via channels
- automatic persistence via Berkley database

XtremOS
Enabling Linux
for the Grid



Introduction basic architecture

Area of Interest




Information Society
Technologies

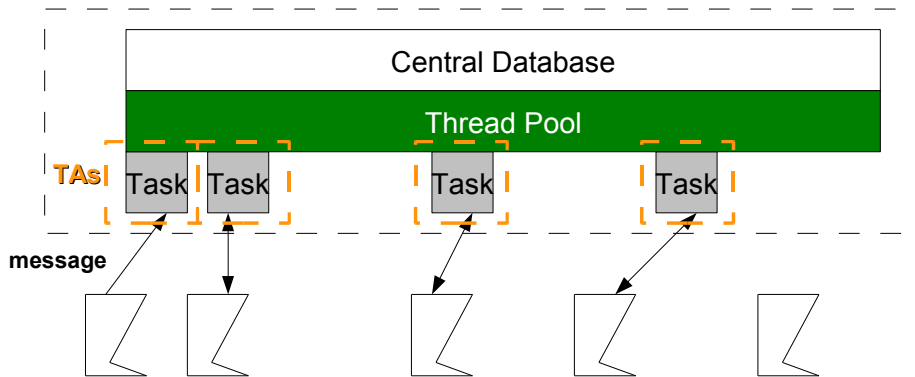
11/09/2009
Oxford

XtremOS IP project
is funded by the European Commission under contract IST-FP6-033576

9



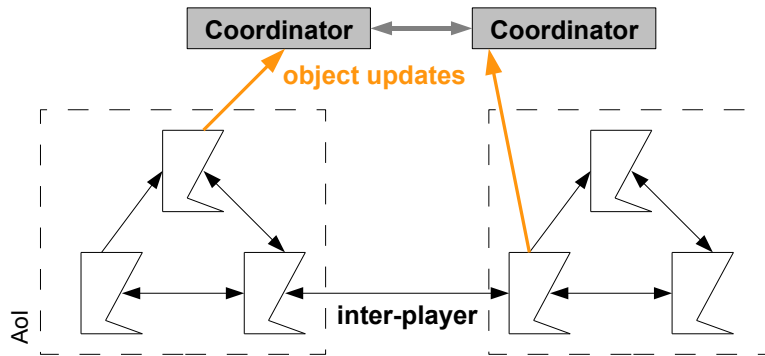
- if all nodes would require updates from all other nodes the bandwidth would be exceeded quite fast or limit the amount of users
- Area of Interests (AoI) are used to partition users into disjunct groups to keep communication locally, traditional computer science approach: “divide and conquer”
- Inter-AoI communication should be reduced to a minimum

**Project Darkstar - threading**

- every incoming message is run within a single task
- the tasks are run by threads from a thread pool (one thread per task)
- for data consistency every task is run using the ACID paradigm known from transactions



SimMud - Overview



- SimMUD is a peer to peer architecture based on Pastry and Scribe
- fully peer to peer for player interactions like movement
- access to shared object like items is controlled by controllers in a client / server manor
- aims to reduce load on servers and use them only for security and consistency of items
- message based architecture
- Area of Interest management to form group of nodes



Current Approaches

- **Message Based**
 - algorithms must be aware of messages
 - network model is reflected in the algorithms
 - algorithms splitted into receiver / responder
 - consistency based on message automate



- every algorithm must be aware of the message based architecture beneath, because this is the only way to communicate
- the network model is reflected in the algorithm, a algorithm written for a client / server network model cannot be used transparently for a peer to peer model
- it is often necessary to distinguish in the algorithm between client and server part thus increasing complexity
- consistency is defined by the automata defined by the message model



Our Approach

- **Object Based**
 - algorithms access distributed & shared objects
 - network model is transparent
 - one algorithm for all participating nodes
 - memory based consistencies



- algorithms are based on access to objects, which are shared transparently
- the underlying network model, if client /server, p2p or any mixture is hidden
- the algorithm is executed alike on all nodes, there is no predefined role like in client / server models
- consistency is maintained on a memory basis, distinct from any messages that might be used



- Introduction
- **Object Sharing Service**
 - Overview
 - Network Architecture
 - Transactions
- Wissenheim
- Conclusion



**▪ Purpose**

- improve data exchange and consistency maintenance
- complement message-passing
- offer location-transparent, fault-tolerant transactional storage for distributed objects
- different consistency models
- overlay network architecture

▪ Related

- BlobSeer / JuxMem
- DiSTM
- GigaSpaces



- work of WP 3.4 at the University of Düsseldorf
- add on for existing systems
- easy access to parallel applications
- the OSS architecture is designed to be extendible
- BlobSeer support for handling large binary blocks (TB size) on a Grid level
- JuxMem provides globally unique address space on a Grid level
- DiSTM is a software transactional framework for clusters
- GigaSpaces is an application server providing shared memory for Java VMs

XtremOS
Enabling Linux for the Grid

Object Sharing Service
overview

Architecture

The diagram illustrates the architecture of the Object Sharing Service. It is divided into two main sections: Client Management and Consistency Model. The Client Management section includes a CLIENT box and a Cache Management block containing Cache, Heap, and Stack components. The Consistency Model section includes a SERVER box and a Consistency Model block containing Transactional Consistency and Strict Consistency components. Below these sections are Memory Extension and Overlay Network blocks, which are connected to the client and server components.

11/09/2009
Oxford

XtremOS IP project
is funded by the European Commission under contract IST-FP6-033576

16

Information Society Technologies

European Union

- OSS is using a page based memory model to simulate a shared memory
- user access to the share memory regions is transparently logged by the OSS library via mprotect
- network model is interchangeable
- multiple consistency models are available, strict and transactional are available

XtremOS
Enabling Linux for the Grid

**Object Sharing Service
network architecture**

The diagram illustrates a network architecture for an Object Sharing Service. It features three cloud-like regions. The leftmost cloud contains a 'super node' (a red square with a white 'Z' shape) and a 'home node' (a white square with a white 'Z' shape). The top and right clouds each contain a red square with a white 'Z' shape, representing super nodes. Red arrows indicate connections: one from the super node in the left cloud to the super node in the top cloud, one from the super node in the left cloud to the super node in the right cloud, and one from the super node in the right cloud to the super node in the top cloud. Each cloud also contains several smaller white squares with white 'Z' shapes, representing other nodes in the network.

Information Society Technologies

11/09/2009
Oxford

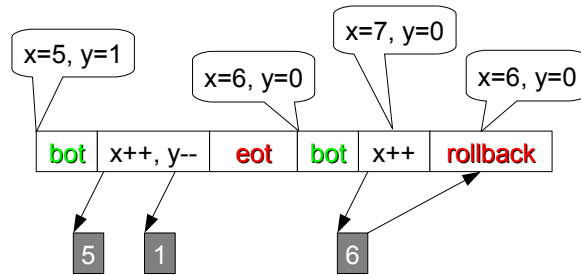
XtremOS IP project
is funded by the European Commission under contract IST-FP6-033576

17

- OSS uses a topology-aware overlay network
- routes messages
- uses XtremOS directory service for group membership (WP 3.2)
- Super Nodes partition network traffic into global and local to reduce overall traffic (comparable to AoI)



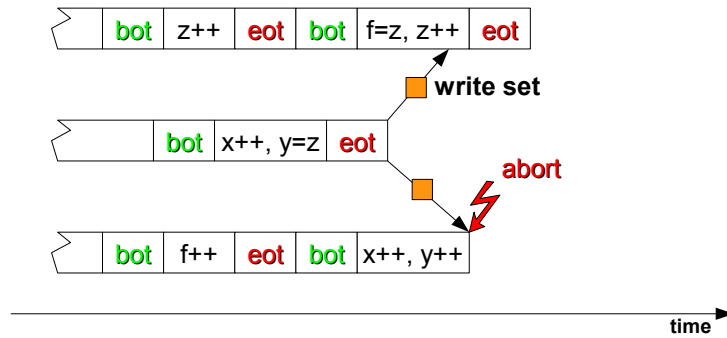
Speculative Transactions



- optimistic approach for transactions, transaction is making copies of all accessed objects / pages (shadow copies) prior to access
- in case of rollback, altered objects / pages are restored using the shadow copies made
- in case of successful commit, shadow copies are discarded
- validation takes place at commit time
- pessimistic approach uses locks to synchronize access to objects / pages



Forward Validation



- forward validation
- token is used to serialise access (first-wins strategy for token retrieval)
- write set contains information which pages have been altered by the transaction
- running transactions will now check if they have any conflicts (read/write, write/write)
- if they detect a conflict, transaction will be aborted and all changes undone (shadow copies)



Usage

```
int i=5;
bot();
    printf("Number %d lives\n",i);
    obj.setPos(12,12);
    i = 8;
eot();
```



- bot() starts the transaction, OSS will save the stack position
- read / write set and shadow pages are automatically generated using the mprotect mechanism
- aborts can happen in between, OSS will then restore shadow pages, cut stack back to bot() and restart
- aborts in between can be dangerous if system functions are used inside the transactions (e.g. locks used by system functions), therefore a deferred abort can be used at eot() time
- eot() will validate the transaction and abort if necessary
- stack variables written within transactions are not secured by transaction!



- Introduction
- Object Sharing Service
- **Wissenheim**
 - Basics
 - Shared Game State
 - Workflow
 - Speed Ups
 - tGOS
- Conclusion





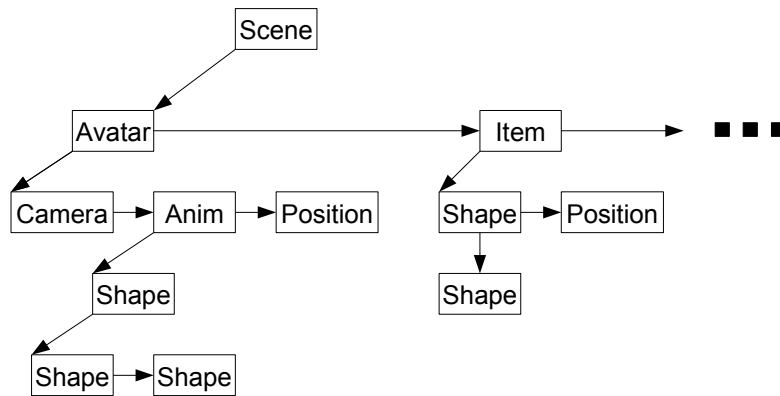
- **Multiuser Virtual Environment**
 - developed at University of Ulm
 - avatar based experience
 - used for teaching content
- **Different Flavours**
 - Sun Java with tGOS
 - native Linux with OSS
 - RainbowOS (in progress)
- **You can visit !**
 - www.wissenheim.de
 - full public access in Oktober



- the Wissenheim project has been started in 2004 as a demonstrator for the transactional memory operating system Plurix
- Wissenheim concentrates mainly in conveying educational topics, e.g. lecture topics of the University of Ulm
- it is now available in three different flavours:
 - Sun Java Version accessible via a standard Java enabled Webrowsers
 - native XtreemOS demo application
 - Rainbow DSM (upcoming)



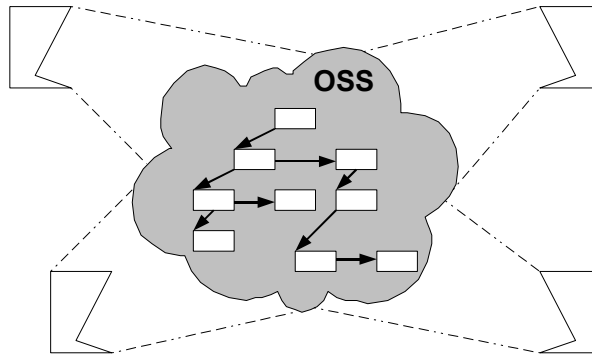
Scene Graph



- Wissenheim is using an acyclic graph structure as the main building block for the distributed game state
- the scene graph is structured as a transformation hierarchy and also incorporates information about meshes, materials and textures used
- new avatars are added to a scene by adding them to the graph
- shape objects are simple meshes with one material and multiple textures, they are the basic graphical objects available within Wissenheim
- avatar objects are grouping objects with special meaning
- animation objects (Anim) are used to mark the start point of a key frame animation in the scene graph
- camera objects are used to generate the rendering view



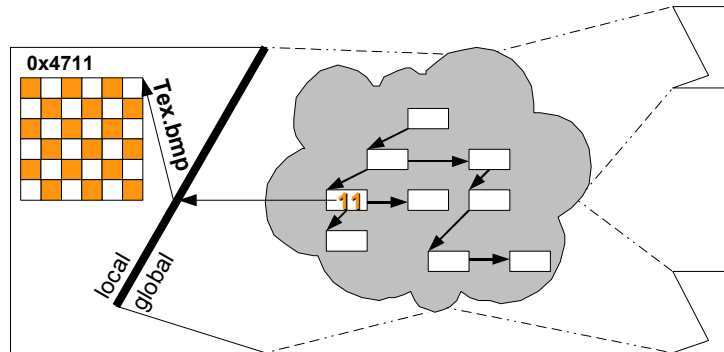
Scene Graph Sharing



- every node is accessing the scene graph directly, all virtual addresses are globally unique!
- a reference from the scene graph to a local reference is therefore forbidden because it cannot be guaranteed that every node will have a valid object at the corresponding local address



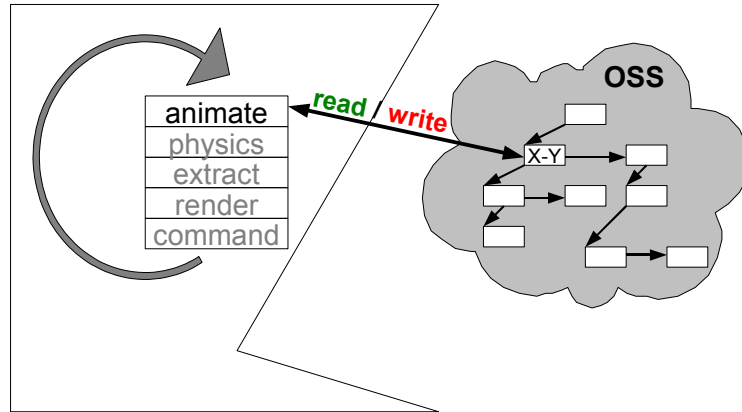
Resources



- resource files like meshes, texture and fonts are not distributed via OSS due to their read only nature and huge size
- these resources are fetched via the XtremFS file systems which allows a reliable and highly distributed access
- because there cannot be any reference from the scene graph to local memory, resources are given a globally unique ID which can be translated to a file name
- resources already accessed are cached by the local node



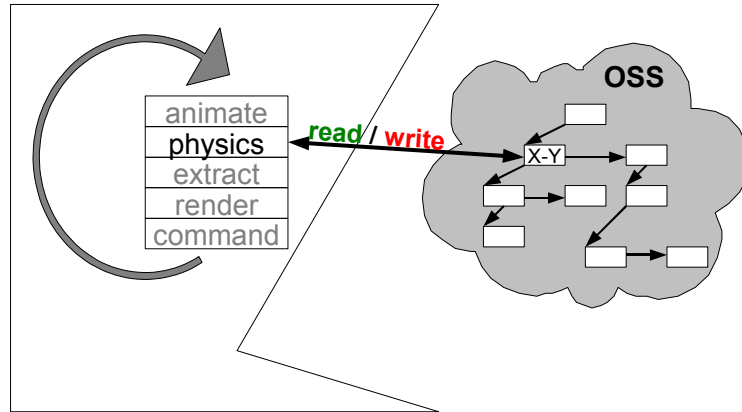
The Loop



- Wissenheim is using the classic single threaded loop for running the game
- external events are stored and handled via the input phase
- the animation phase is altering the transformation of objects to achieve key frame animations, the global scene graph is access by read and write operations
- it also stores information about the animation progress



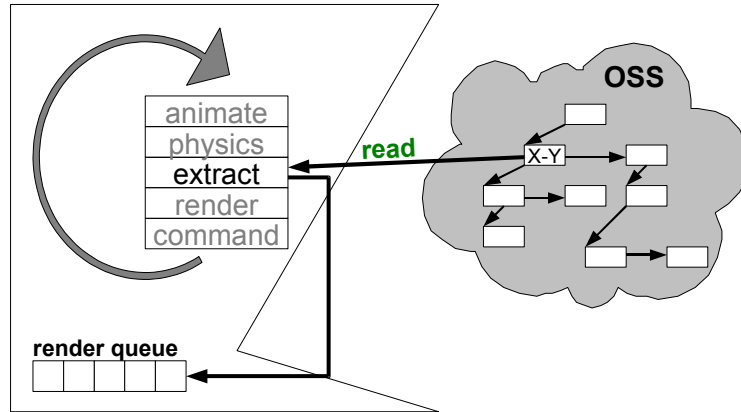
The Loop



- Wissenheim is using the classic single threaded loop for running the game
- external events are stored and handled via the input phase
- the physics is making collision detection and moving the physics-aware objects by altering their transformation
- the scene graph is accessed by read and write operations



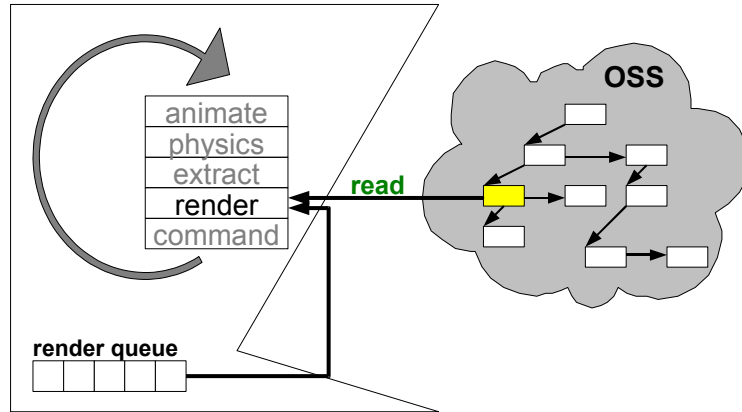
The Loop



- the extraction phase is reading hot spot data like the transformation of shapes from the scene graphs and creates copies in the render queue
- copies are made to make it less likely that the render phase might get aborted
- the render queue holds every shape which should be painted
- the extractor will also sort the shapes according to render rules (solid, transparent, etc.)
- reference to materials are copied by reference



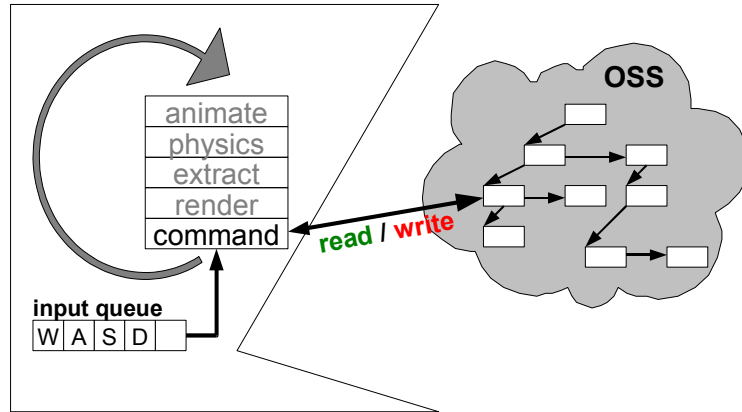
The Loop



- the rendering phase is passing the mesh, texture and material data along with the transformation to the OpenGL interface to render a new picture
- this phase reads the material data stored in the global scene graph along with the copies of the transformation from the render queue



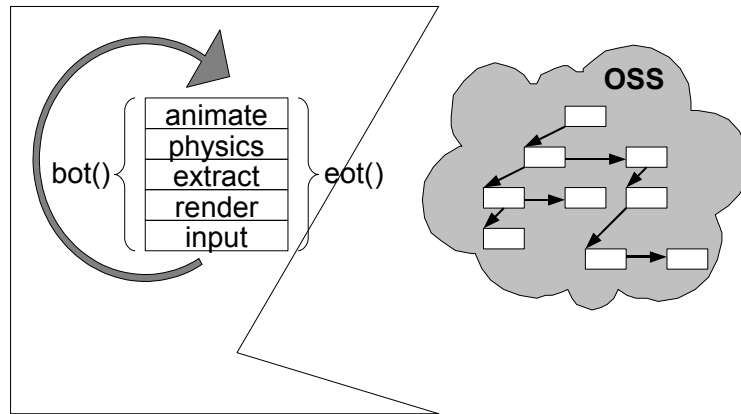
The Loop



- the input phase is executing any stored input event (like key strokes, mouse moves, etc.) by reading the events from the event queue and altering data in the scene graph



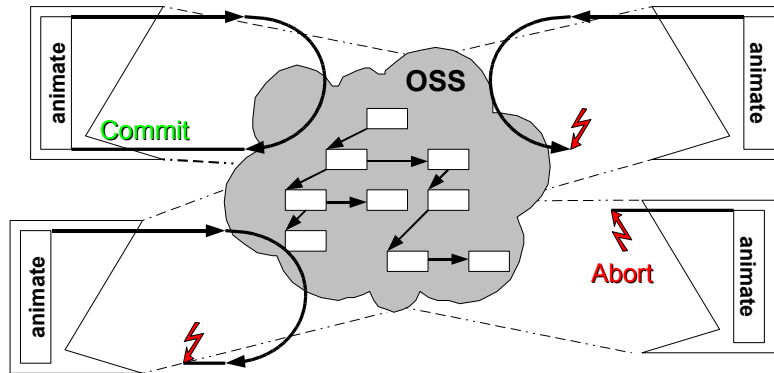
Synchronization



- read and write access to the shared data must be synchronized, the easiest way to do this is by using the transactions provided by OSS
- every phase is here surrounded by an begin-of-transaction (`bot`) and an end-of-transaction (`eot`)
- every commit (`eot`) will result in a token request and a writeset if the commit is successful



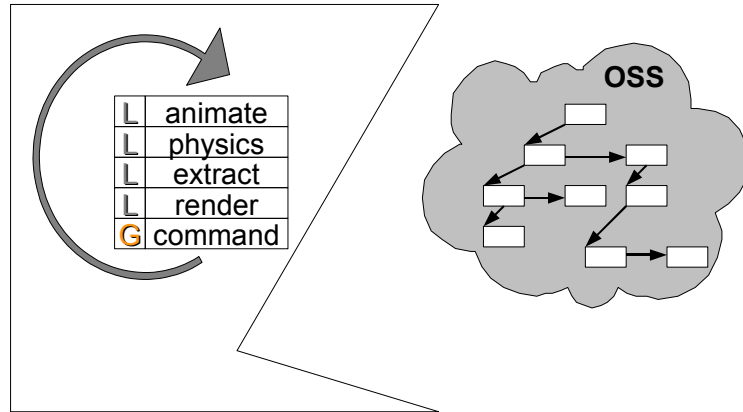
The Problem



- validation phase will induce latency
- maximum amount of transactions limited by the overall latency due time needed for validation
- concurrent access to shared data especially write access will lead to high conflict probability and therefore many aborted transactions
- aborts will results in redo of aborted code pieces and therefore decreasing frame rate
- no “local” write in shared data possible



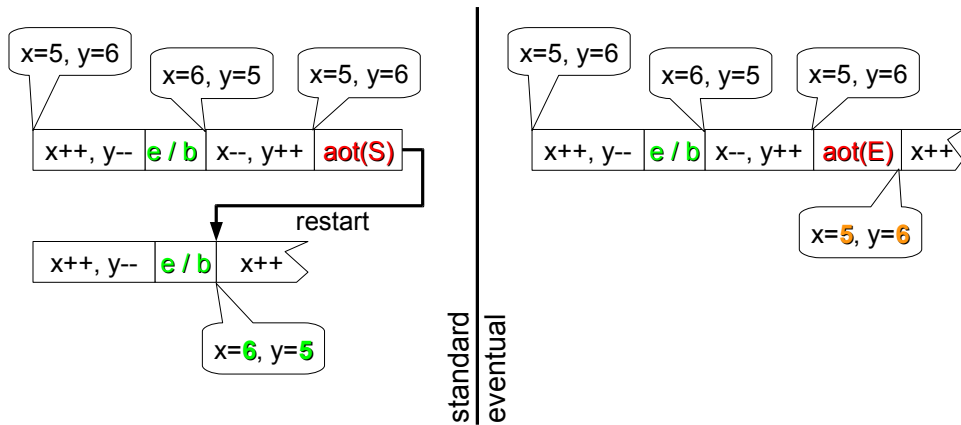
The Loop Revisited



- read and write access to the shared data must be synchronized, the easiest way to do this is by using the transactions provided by OSS
- every phase is here surrounded by an begin-of-transaction (bot) and an end-of-transaction (eot)
- every commit (eot) will result in a token request and a writeset if the commit is successful



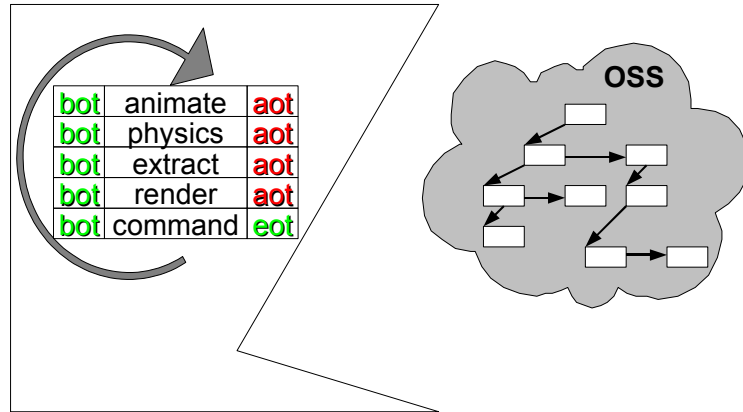
Special Abort Transaction



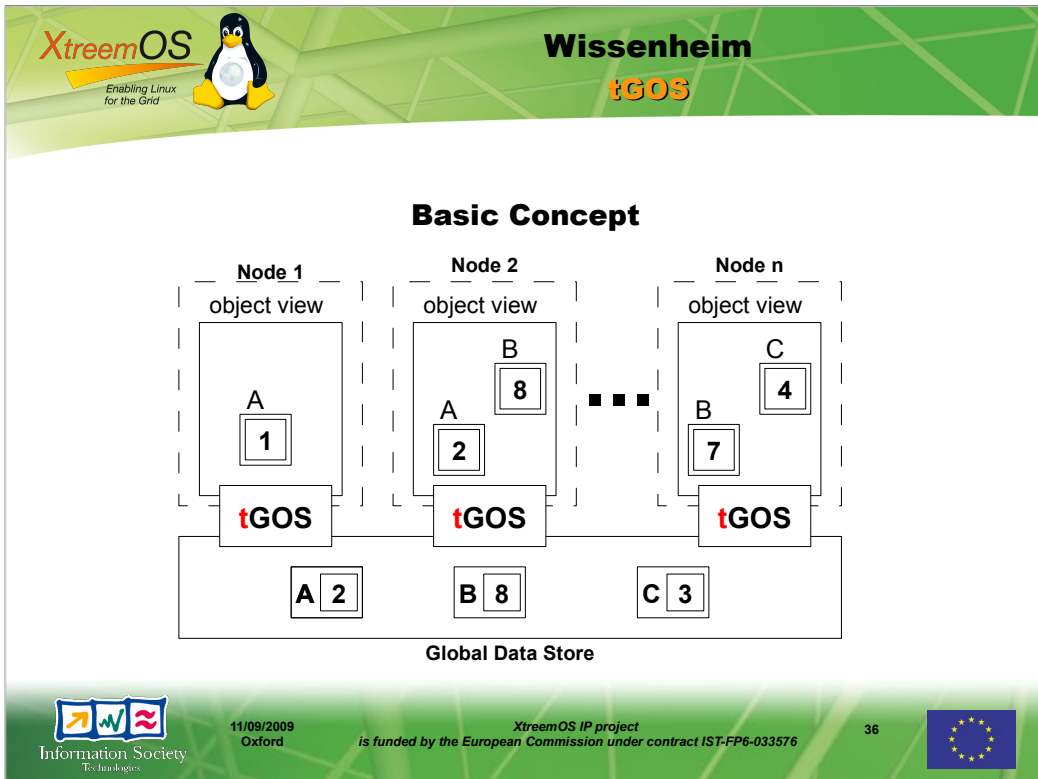
- e /b stands for eot() followed by bot(), means finishing the first TA and starting a new one
- standard abort operation will undo any changes made and restart execution of aborted part
- the eventual abort will end the transaction without undoing changes and without restart
- consistency is now breached, full transactional consistency cannot be guaranteed any more
- no global commit, we have now a local view inside the globally shared data set



The Eventual Loop



- only command loop is now secured by a “real” transaction all other phases are running with eventual consistency
- updates from other committed command phases are integrated into the local view at every bot()
- overall transaction rate greatly reduced, conflict probability for most phases is down to zero
- false sharing can be a huge problem, handling of consistent and inconsistent data delicate
- with this trick a WAN setup from Rennes to Düsseldorf was playable with only minor latency noticeable
- sacrificed ease of use and consistency for speed



- Typed Grid Object Sharing (tGOS)
- complementary to OSS
- designed for pure Java in mind
- object based granularity
- every object view can have a completely different set of objects with different versions
- global data store is the definition of an replication layer which has to incorporate at least partial event ordering and persistence
- only the global data store has a consistent view of all objects
- tGOS provides basic mechanism to distribute objects
- no default consistency
- tGOS is a concept, implementation can vary
- powers Wissenheim Worlds



Basic Operations

Push

Pull

Sync

Invalidate

Order



- tGOS defines only five basic operations on objects
- all operations are synchronous, except order
- push will actualize the version of an object in the global data store (GDS)
- pull will retrieve the most actual version of an object from the GDS
- sync will set an advisory lock on an object
- invalidate will actualize the version of an object in the global data store (GDS)
- order will request the actualized version of an object
- higher consistency models are implemented within the tGOS object model using the basic operations

XtremOS
Enabling Linux for the Grid

Wissenheim
tGOS

Events

update

time

Information Society Technologies

11/09/2009
Oxford

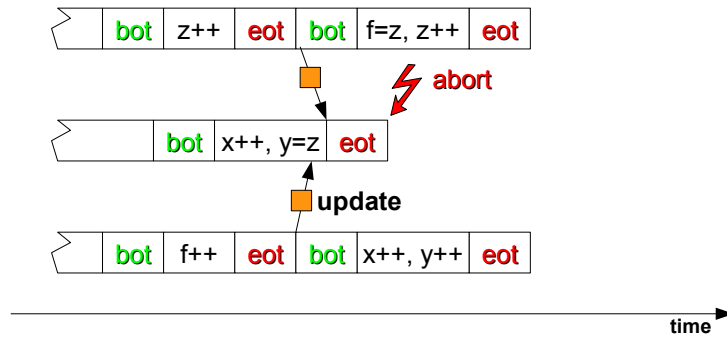
XtremOS IP project
is funded by the European Commission under contract IST-FP6-033576

38

- tGOS introduces events which are called when an object gets updated
- two event types
 - update - called in case of a push operation
 - invalidate - called in case of a invalidate operation



Backward Validation



- forward validation
- token is used to serialise access (first-wins strategy for token retrieval)
- write set contains information which pages have been altered by the transaction
- running transactions will now check if they have any conflicts (read/write, write/write)
- if they detect a conflict, transaction will be aborted and all changes undone (shadow copies)



Transactions

```
int i=5;
do {
    begin()
    add2TA(obj)
    obj.setPos(12,12);
    i = 8;
} while (!commit());
```



- transactions are implemented using the basic operations provided
- due to Java limitations we have to use a loop instead of cutting back the stack like OSS does
- write and read set generation must be done by hand, automation is possible using aspect oriented concepts (e.g. AspectJ)
- at *begin* the object view will be brought to a consistent state (all received updates from commits are integrated)
- *add2TA* will add an object to the read/write set, a shadow copy is then created
- *commit* will try to validate the changes by getting a lock on the token and checking if any transaction with a conflicting read/write set has committed prior, if so the shadow copies are restored and false is returned
- update based mechanism due to backward validation, penalty for committing task
- latency optimized
- empty transaction will bring object view to consistent state without lock request
- reader free, writer waits



- Introduction
- Object Sharing Service
- Wissenheim
- **Conclusion**
 - Transactions
 - Object Based Approach






- **General**
 - easy to use
 - feasible for consistency management of multiple objects
 - guarantee high level of consistency
 - fast in low conflict environment
 - optimistic approach is deadlock free
 - fine grain usage possible
- **OSS**
 - automatic read / write set generation
 - scalability through Super Nodes
- **tGOS**
 - complementary to OSS
 - multiple consistency domains (Aol)
 - update mechanism




- transactions are very useful for creating an intuitive programming model in an highly parallel environment (a lot of research is currently focusing at this topic), especially if used with multiple objects which are connected with each other
- guarantees strict consistency without the need for fine grained locks
- due to optimistic approach high level of parallelism possible
- high conflict rate will destroy parallelism and might lead to starvation problems
- the optimistic approach will not produce deadlocks like the pessimistic approach
- explicit transactions can be used very fine grain
- the current implementation of transactions in tGOS are using an update model, which results in a writer penalty (out-dated-data check are done in case of commit)



Conclusion

object base approach


- **Object Based Approach**
 - frees the algorithm from the network
 - network model is exchangeable
 - one-node-view
 - more focus on consistency
 - message based programming also possible



11/09/2009
Oxford

XtremOS IP project
is funded by the European Commission under contract IST-FP6-033576

43



- the Wissenheim access to the scene graph is done completely without knowledge of the underlying network architecture, the network model
- the algorithms are design with a one-node-view meaning that we don't have to think about splitting into a client / server model or about a special role forced by the network model
- the consistency can be handled like multi-thread consistency on a memory basis instead of thinking about message ordering etc.
- the network model with respect to client / server or p2p used by OSS is of no concern to the application

