Lecture Notes

Managing and Mining Multiplayer Online Games

Summer Term 2017

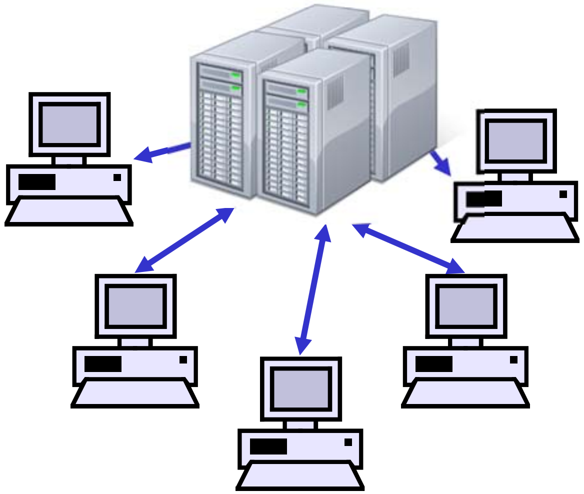
Chapter 3: Distributed Game Architectures

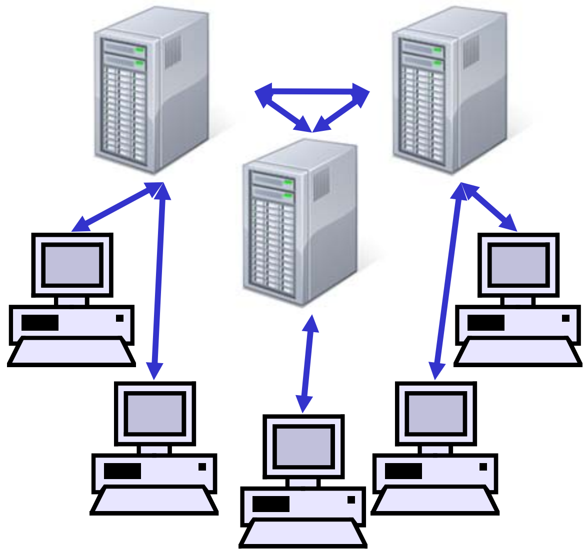
http://www.dbs.ifi.lmu.de/cms/VO_Managing_Massive_Multiplayer_Online_Games

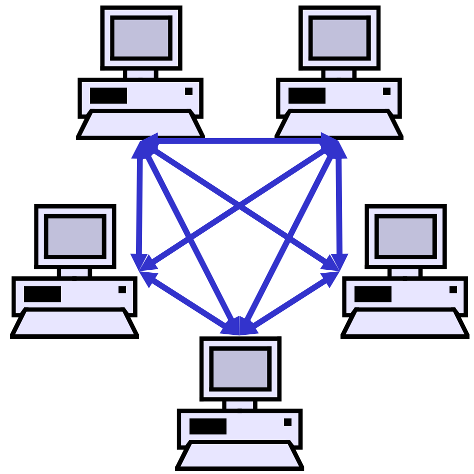
Overview

- Architectures for distributed games
- Distributed of action handling
  - Fat-Client vs. Thin-Client
  - Problems of centralized and decentralized computation
  - Problems with local time stamps
- Spatial movement and dead reckoning
  - Update strategies
  - Movement models
  - Error correction
- Network protocols and games
  - Typical network load for games
  - TCP and games
  - UDP and games

MMOG Architectures







Client-Server:

- Provider hosts game in a computing center
- Game client and server run different software
- Centralized solution for:
  - account-management
  - partitioning of the game world
  - monitoring
  - persistence

Multi-Server:

- Several servers
- Redundant data storage
- Network distance between client and server is generally shorter
- Dynamic solutions:
  - replication
  - proxy-Server

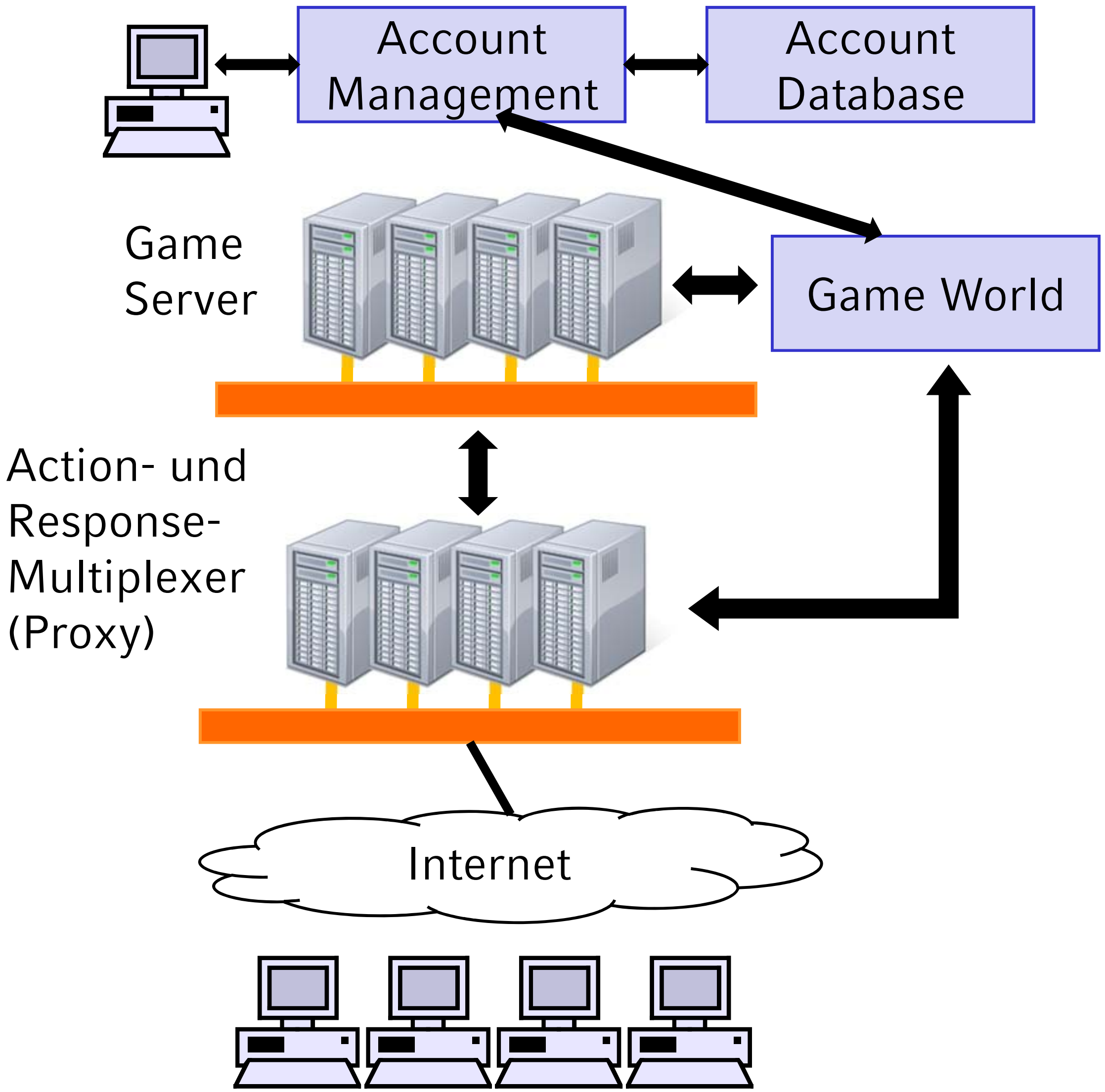
Peer-to-Peer:

- No explicit Servers
- Data exchange between adjacent peers
- Every peer is hosting part of the game world
- Dynamic partition of the game world

Detailed Client Server Architecture

- Hosted in a computer center
- Several game servers share game state of a realm
  - zones shards/realms, instances
  - strict division of zones
  - seamless distribution (communication between servers)
- An authentication / account management service exists
- Action- and Response-Multiplexer

  (Proxy) may ease the load on game servers by taking over particular functionalities



Distributing the Game Core

Design choices:

- What kind of participants (peers) exist?
- What are the peers exchanging?
  (actions, object states, user input, …)
- Who is authorized to read which part and who is also entitled to write?
- How is the load redistributed among existing peers?
- How is time between peers synchronized?

Protocol content

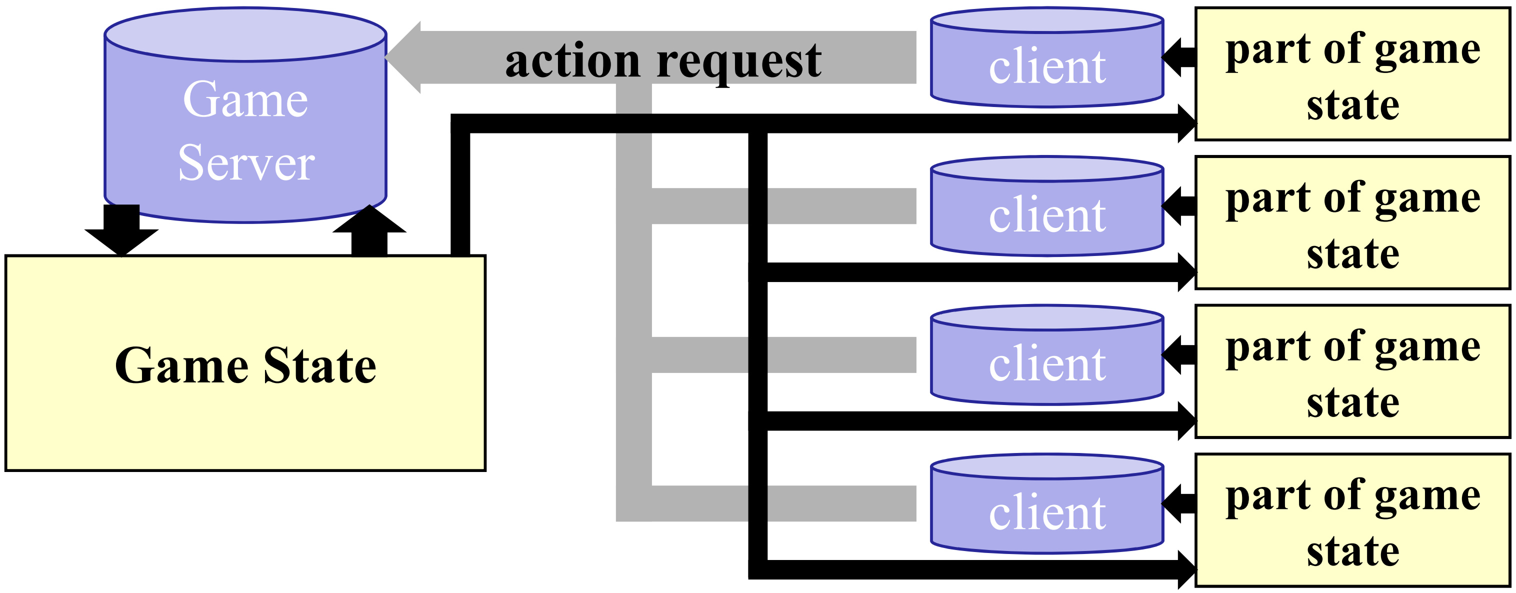
- Object attributes: (Action Result Protocol)
  - protocol sets the current parameter value of a game entity
    (set player „ Facemelt0r“ HP to 96)

  - protocol sends relative changes
    (reduce player „Facemelt0r“ HP by 100)

- Actions: (Action Request Protocol)
  - Contains only Player input without direct impact on game state
  - Protocol only transfers user input
    => Results must be calculated on the server
    (Try to hit Player „Facemelt0r“ with „Uppercut“)

Thin Client Solution

- Server holds the complete game state and is solely authorized to change it
- Clients receive a part of the game state upon login
- Server transmits game state changes to clients
- Client transmits actions it wants to execute to the server (Action Requests)
- Server collects all incoming action requests
- Actions are handled in their order of arrival and results are transmitted to affected clients



Exclusive Thin Client Solution

Advantages:

- Game state is centrally managed
  - Consistent game state for calculating action results
  - No conflicts from several contradictory game states
  - Persistence system is able to save consistent game states
- Low potential for cheating/ action handling only on the server

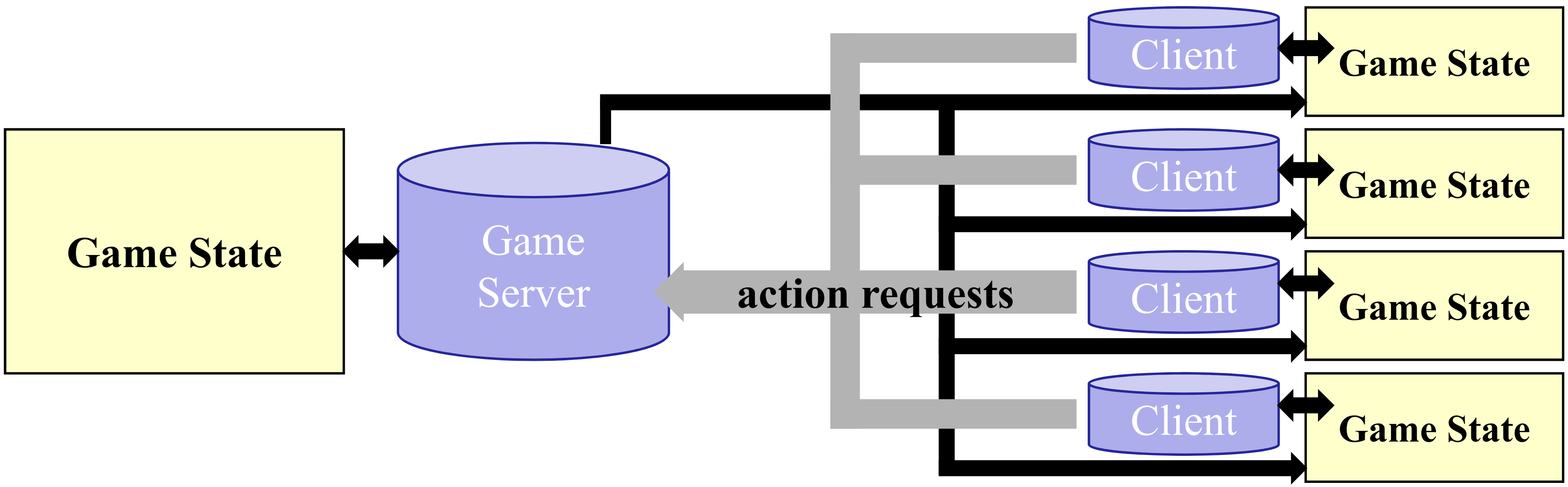
Disadvantages:

- maximum server load because all action handling is server based
- potential for high latency (actions need to be transmitted to the server and back to take effect)
- client sided processing power is largely unused

  (client only displays the game state and transmits user input to the server)

Fat Client Solutions

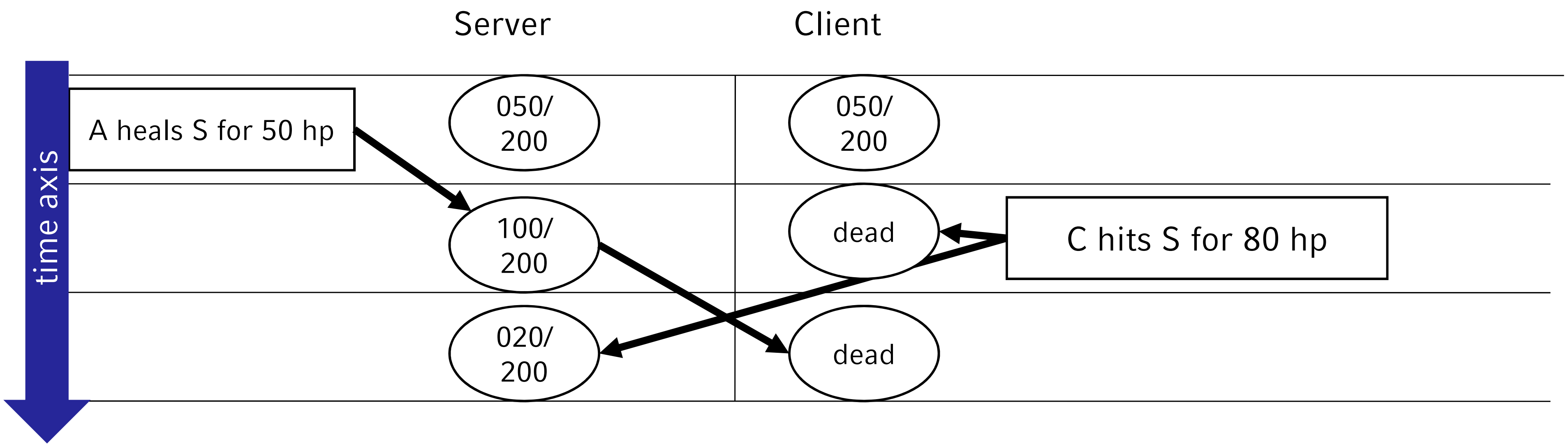
- Every Client has her own objects which only he is authorized to edit
- Server manages chronological sequence with time stamps and transmits changes to the other clients
- Local game states may vary, due to transmission delay
- Chronological sequence may be inconsistent, since local changes may be implemented before global with an earlier time stamp



Conflicts during decentralized computing

- Local changes need time to be transmitted to the network
- Actions are calculated for and executed on local game states

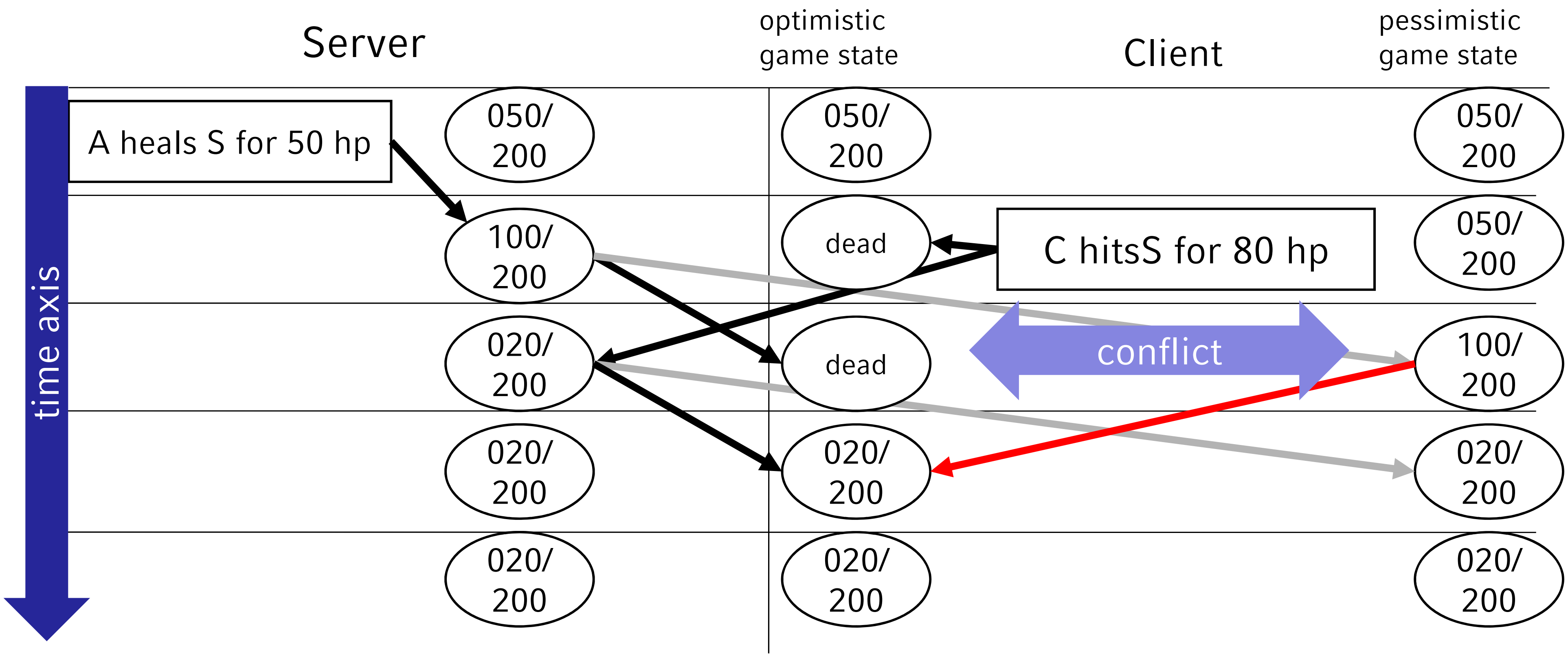
  => changes that predate the action may not be taken into account
- Simple solutions:
  - Client is not allowed to change local data without server acknowledgment
  - Using object protocols the server may send an update of the current game entity state.



Solution Approach

Reset local actions

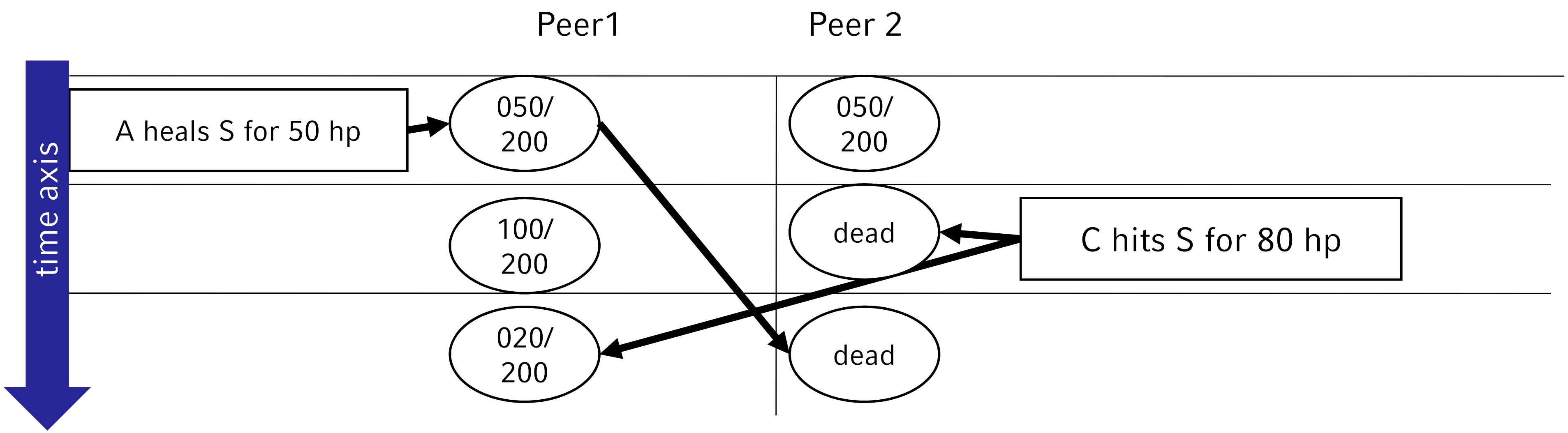
- Client has 2 game states:
  - optimistic GS (contains local changes)
  - pessimistic GS (contains actions transmitted by the server)
- On mismatch: Resetting the optimistic GS to the state of the pessimistic GS



Local time

up until now: One server handles processing sequence

- Impossible for P2P games and multi server architecture
  => sequence is inconclusive after arrival at server
  => organization by local time stamps on creation
- During processing both, own and foreign changes may appear in incorrect sequence
- In case of inconsistencies game entities can be synchronized

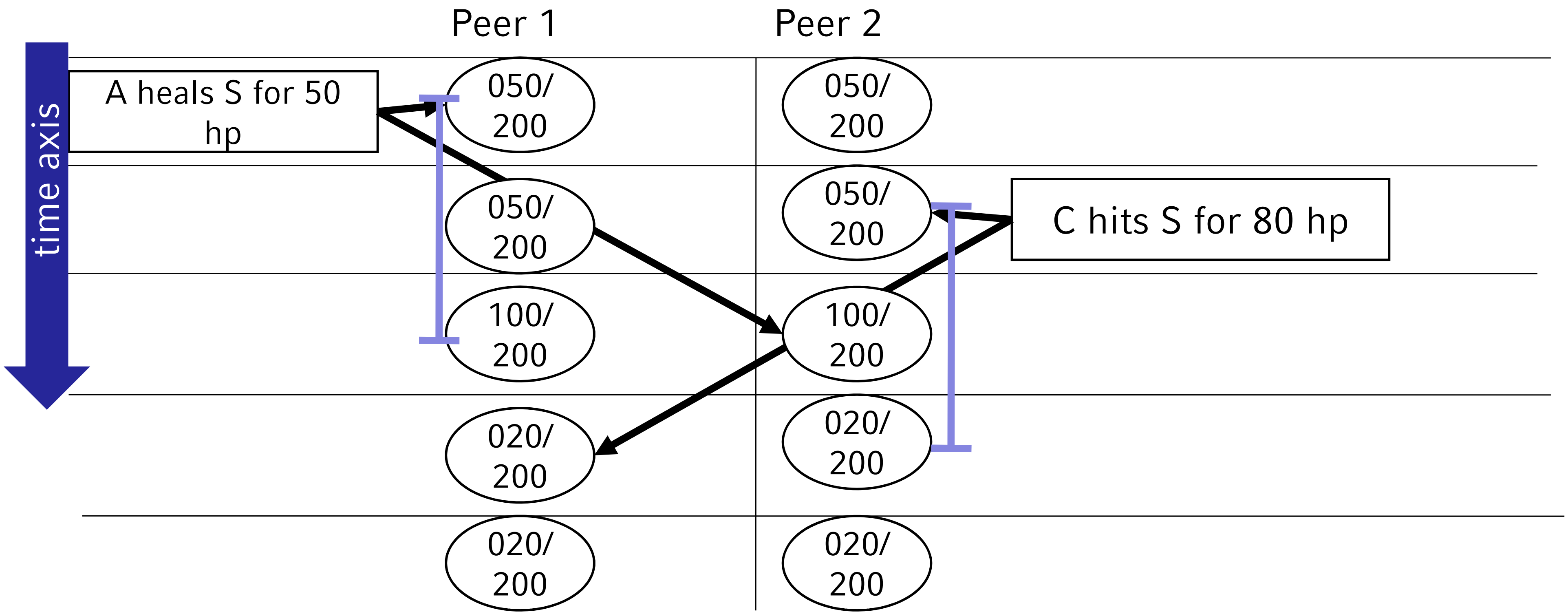


Solutions by Local Lag Mechanism

Problem is causes by the lack of knowledge about previous actions

Solution: Lag-Mechanism

- Processing updates is delayed to allow for other actions to arrive in time
- If this time frame is exceeded, conflict detection and reset become necessary



Application in Games

Games can combine several approaches by processing actions differently.

Server side processing	**Client side processing**
• content accuracy is important • response time less important • chronological order is important	• response time is crucial • synchronization and Sequence are less important
• damage and healing • item pick up	• position- and movement- data • animations and other display effects

Conclusion:

- Generally speaking, there is a trade off between latency (here: response time) and consistency of the game world.
- Another issue is reducing remote updates to reduce the needed bandwidth.

Movement Information

Movement-updates play a special role in distributed virtual environments

- Fluid movement
  - ⇒ Position may change several times per second (24-60 FPS)
  - ⇒ Calculation should be closely tied to rendering
  - ⇒ Handling movement and other actions in the same way might disturb animation
- Precise positions are mostly irrelevant for game play:
  - ⇒ Due to the fast update rate the loss of several position updates is often negligible

Consequences:

- Real-time movement for games is predominantly calculated locally on the client
- Sequences of precise positions are not relayed to other peers to save bandwidth
- Movement is extrapolated locally and positions are synchronized only at certain times

=> **Dead Reckoning**: Simulating movement between two updates to allow for fluid movement with limited bandwidth.

Dead Reckoning

components for games:

- Update-Strategy on the server side (owner of changed) game entity:
  When is position information transmitted and with what frequency?
  (influences bandwidth and error rate on the client)

- Movement model on remote peer:
  How is movement extrapolated between two updates?
  (influences error rate and perception of movement on the client)

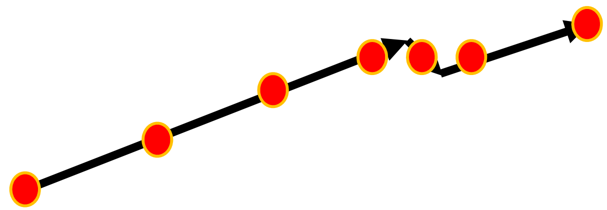
- Error correction on remote peer:
  How are estimated and received position merged?
  (influences perception on the client)

=> there is a trade-off between:

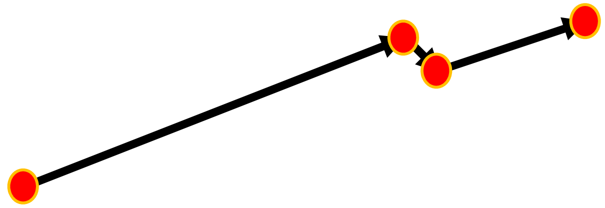
  - bandwidth and error rate
  - perception and processing time

Update-Strategies for Dead Reckoning

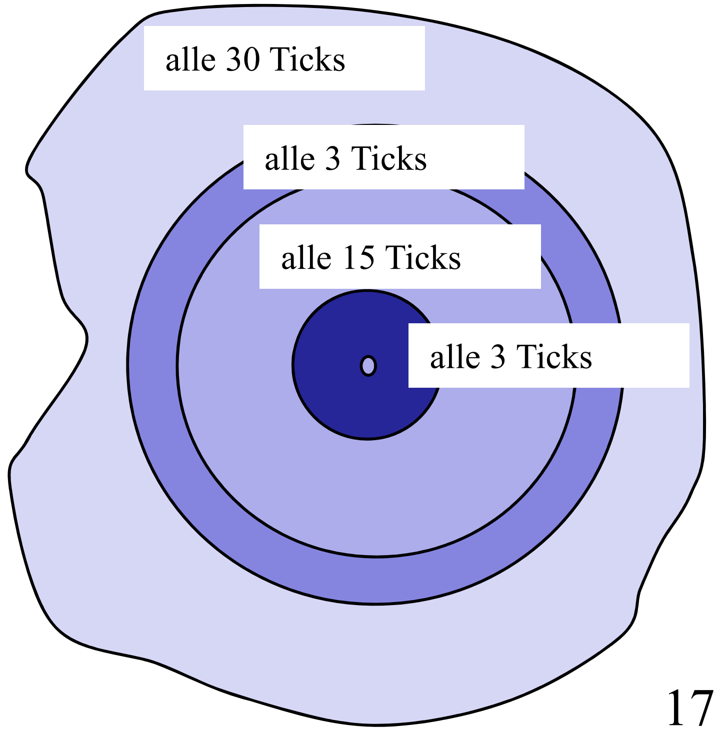
- regular updates:
  - send updates in regular intervals



- event based updates:
  - send updates on changing direction or movement type



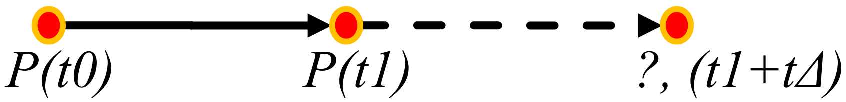
- distance-based-updates:
  - precise positions are more important the closer an object is
  - the closer an object is to a critical range (e.g. weapon range)
  - transmits regular updates, but with different rates, depending on distance.



Movement model for Dead Reckoning

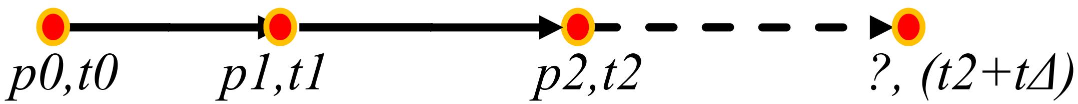
Point in time: t_i *Position:* $p(t_i)=(x_i,y_i)$ *Average speed:* $v(t_i)$ *acceleration:* $a(t_i)$

Linear movement with constant speed:



$$p(t_1+t_\Delta)=p(t_1)+\underbrace{\frac{p(t_1)-p(t_0)}{\|p(t_1)-p(t_0)\|}}_{\text{direction}}\cdot t_\Delta\cdot\underbrace{\frac{\|p(t_1)-p(t_0)\|}{(t_1-t_0)}}_{\text{speed}}=\boxed{p(t_1)+t_\Delta\cdot\frac{p(t_1)-p(t_0)}{(t_1-t_0)}}$$

Linear movement with constant acceleration:



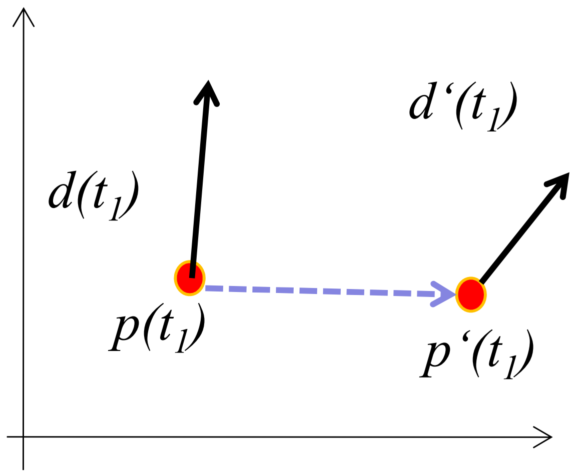
$$v(t_2+t_\Delta)=v(t_2)+t_\Delta\cdot\frac{v(t_2)-v(t_1)}{(t_2-t_1)}=v(t_2)+t_\Delta\cdot\frac{\frac{\|p(t_2)-p(t_1)\|}{(t_2-t_1)}-\frac{\|p(t_1)-p(t_0)\|}{(t_1-t_0)}}{(t_2-t_1)}$$

$$p(t_2+t_\Delta)=p(t_2)+t_\Delta\cdot v(t_2+t_\Delta)\cdot\frac{p(t_2)-p(t_1)}{\|p(t_2)-p(t_1)\|}$$

Error Correction for Dead Reckoning

Problem: prediction and update do not correspond.

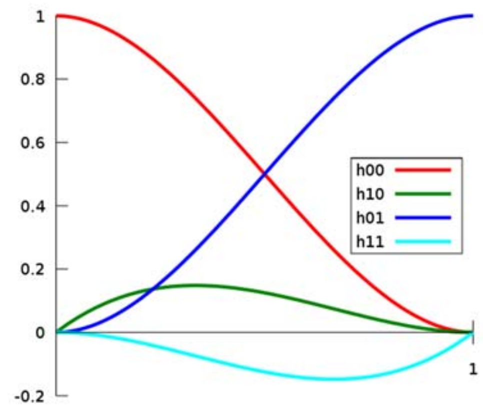
- Object on a remote peer is overwritten by an update for high error rates:
  - objects jump
  - objects disappear and reappear elsewhere



- Both Positions are merged with a accelerated fluid movement:
  - e.g. cubic polynomials: Bezier, B-Splines, Hermite
  - Must allow for a certain correction time Δt

Hermite graphs for polynomial smoothing

Four base polynomials:

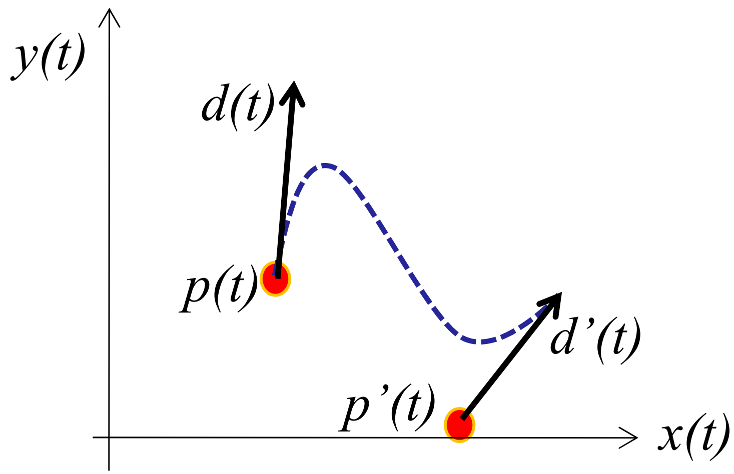


- $h1(x) = 2x^3 – 3x^2 + 1$
- $h2(x) = –2x^3 + 3x^2$
- $h3(x) = x^3 – 2x^2 + x$
- $h4(x) = x^3 – x^2$

Connecting points p and p′ + d′ using the following linear combination

$p(x) = p(t)\, h1(x) + p'(t+\Delta t) h2(x) + d(t)\, h3(x) + d'(t)\, h4(x)$ $(0 \leq x \leq 1)$

- Position: *p(t)* via Dead Reckoning
- movement direction: *d(t)* via Dead Reckoning
- target: $p'(t+\Delta t)$ via server-update
- whereby $p'(t+\Delta t) = p'(t) + d'(t)$ is the position
- at the point $t+\Delta t$ in time, expected from the update
- Δt: Time for corrections (compensation by faster speed)



Thoughts on Client-Server Communication

Important influential factors

- **Latency**: Time until the system reacts
  - round trip time (RTT)
  - package size
  - system load aside from the network
- **Bandwidth**: How large is the transfered volume?
- **Burstiness**: How is the data volume distributed over time?
- **Connection-oriented/package oriented protocols**
  - Connection oriented: Routing happens once
  - Package oriented: Routing happens for every package
- **Security**: Is loss of data possible?

Requirements of computer games

application/platform	payload size (bytes)			avg. bandwidth requirement	
	avg.	min	max	pps	bps
Anarchy Online(PC)‡	98	8	1333	1.582	2168
World of Warcraft (PC)	26	6	1228	3.185	2046
Counter Strike (PC)	36	25	1342	8.064	19604
Halo 3	247	32	1264	27.778	60223
Gears of War (XBOX 360)	66	32	705	2.188	10264
Tony Hawk's Project 8 (XBOX 360)	90	32	576	3.247	5812
Test Unlimited (XBOX 360)	80	34	104	25	22912

from: Harcsik, Petlund, Griwodz, Halvorsen: Latency Evaluation of Networking Mechanisms for Game Traffic, NetGames'07, 2007

- small package sizes
- little bandwidth is used
- latency by genres:
  - RTS-games: <1000 ms
  - RPG: < 500 ms
  - FPS: < 100 ms

  (Estimated latency for observing an impairment of gaming experience)

Protocols and Communication solutions

TCP/IP:

- Safe protocol: by re-transmission
- Flow control and congestion control
- Optimized for bandwidth usage and data transfer
  (sending big packages to reduce the transmitted TCP-Headers)

Disadvantages:

- Packages may arrive with significant delay(re-transmission)
  => increased latency
  => package may not be needed anymore since newer information have already been transmitted
- Optimizing bandwidth artificially increases latency
  - Waiting for payload for under filled packages
  - Confirmation packages confirm several packages or are embedded within returning traffic

Optimization by tuning and turning features off.

Protocols and Communication solutions

UDP

- Minimal datagram service
- No explicit connection to the remote station
- Unsafe transmission, Correct sequence is not guaranteed
- No congestion control mechanisms

Advantages:

- No re-transmission of lost packages

  => outdated information will not be resend
- less Header-Overhead

Use:

- Servers as a base for middle ware solutions, which implement missing service features in the following protocol layer:
  - maintain update sequence
  - security for certain update operations (e.g. picking up items, ...)

Conclusion network protocols

- TCP/IP is still the most used protocol since routers and infrastructure handle it well
- UDP offers a cost-efficient solution for just-in-time services (Voice, Movement data, ...)
- Secure services are still imperative for most games and must be implemented in the application layer of the protocol
- Studies for other protocols (e.g. SCTP), show no significant increase in performance
- MMORPGs (e.g. World of Warcraft) use TCP for communication

Learning Goal

- Client-Server and P2P architecture in games
- Distribution of Action handling:
  - global processing
  - lokal processing with centralized chronology
  - lokal processing with local chronology

- Dead Reckoning
  - update-Strategies
  - movement models
  - error correction

- Requirements for network protocols for games
  - TCP and games
  - UDP, middle ware and games

List of references

- N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, W. White
  Scalability for virtual worlds
  In Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on, 2009.

- Jens Müller, Andreas Gössling, Sergei Gorlatch
  On correctness of scalable multi-server state replication in online games
  In Network and System Support for Games (NETGAMES'06), 2006.

- Jouni Smed, Timo Kaukoranta, Harri Hakonen
  A Review on Networking and Multiplayer Computer Games
  In IN MULTIPLAYER COMPUTER GAMES, PROC. INT. CONF. ON APPLICATION AND DEVELOPMENT OF COMPUTER GAMES IN THE 21ST CENTURY, 2002.

- S. Harcsik, A. Petlund, C. Griwodz, P. Halvorsen
  Latency evaluation of networking mechanisms for game traffic
  In Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, 2007.