Lecture Notes

Managing and Mining Multiplayer Online Games

Summer Term 2018

Chapter 4: DataPersistence

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

Overview

- requirements
- save games and replays
    - state logs
    - transition logs
    - action logs
- persistence in MMOs
- check-point-recovery methods
    - Naive Snapshot
    - Copy-on-Update
    - Wait-Free Zigzag
    - Wait-Free Ping-Pong

Need for Persistence

1. **saving a part ($gs \in GS$) of the current game state**
   - allows resuming the game at another time (save game)
   - create a consistent game state, in case of a system crash
   - saving is only possible at certain locations
     (e.g. Resident Evil, Diablo III, …)
   - certain parts are not saved
     (NPC position/Monster/Enemies, Random maps, …)

2. **saving a replay of a game ($gs_1, \ldots, gs_{end}$)**
   - allows for retracing and analyzing the course of the game
   - usually saved on the clients
   - replays can grow quite large, depending on the format of stored data
   - since the last GS is also part of the replay, replays could be used as save games as well

Persistence Layer Requirements

- saving should not slow the game down
  => tick duration must not exceed the time limit
- save games should be as up-to-date as possible
  => if possible GS should be saved every tick

- loading a save game is supposed to create a consistent state
  => all GE contained in the save game should be equally up-to-date
  => minimum requirement: game state must be consistent (every GE appears only once etc.)

Important Impact Factors:

- size of the game state
- requirements to actuality and tick frequency
- impact of loading time on recovery
- part of the game state/ history of the game to be saved

Methods for Replays and Save games

Save-Game/Replay: local file containing the game state/ course of play

State-Log:

every game entity is saved every x ticks

⇒ sequential file containing a series of game states

example: demo-files of Quake/Half-life/Counterstrike
(parsed and transformed into XML) (Bachelor Thesis: J. Rummel 2011)

```
<replay path=" c:\data ncsdemos n dus t210 .dem“ duration=" 3379,459 " noOfRounds="39”
   mapname="dedust2 " maxCl ients="16" serverName="HLTV.org/VeryGames .net">
 <rounds>
<roundnumber="1" roundBegin="0" roundEnd=" 40 ,496184 “  endingReason="Bombing" winner=" Terrorists">
<teamScore ct="2" t="1" />
 <ticks>
<ticktime="1"> . . . </tick>
<ticktime="2"> . . . </tick>
<ticktime="3">
<players>
<playerid=" 765611887383 " localName="q“ 15 team="Terrorist" kills="3" deaths="7“x=“680" y=“819" z="164“
 angle0="2" angle1="60" moveType=“" weaponModel="172„  modelIndex="149“  isHi t="Helmet" outOfAmmo="
Rifle" />
....
```

State-Log Discussion

Advantages:

- documents a genuine series of game states
- random access to every point in time
- loading process is very simple and fast

Disadvantages:

- high redundancy for small change rates
- large data volume, due to high temporal resolution (every Tick)
- maximum writing load => possibly not feasible for large game states

Transition-Log

Log all changes to the game states via:

- time-stamp
- ID of the GameEntity
- Attribute
- New value

Advantage:

- more compact than a snap-shot
- less volume means less computation effort

Disadvantage:

- reconstruction of game state is more complex
- changes must be registered with the persistence layer

Action-Log

- contains the sequence of all user inputs
- the game is needed to “re-play” the game based on the user input
- random events must be saved
  (seeding or random numbers)

example: Starcraft II (*.sc2replays file) after parsing by sc2gears
(http://sites.google.com/site/sc2gears/)

```
0:00  TSLHyuN       Select Hatchery (10230)
0:00  TSLHyuN       Select Larva x3 (1027c,10280,10284), Deselect all
0:00  TSLHyuN       Train Drone
0:01  TSLHyuN       Train Drone
0:01  roxkisSlivko  Select Hatchery (10250)
0:01  roxkisSlivko  Select Larva x3 (10270,10274,10278), Deselect all
0:01  TSLHyuN       Select Drone x6 (10234,10238,1023c,10240,10244,10248), Deselect all
0:01  roxkisSlivko  Train Drone
0:01  TSLHyuN       Right click; target: Mineral Field (10114)
0:01  roxkisSlivko  Select Egg (10270), Deselect 1 unit
0:01  roxkisSlivko  Select Drone x6 (10254,10258,1025c,10260,10264,10268), Deselect all
0:01  roxkisSlivko  Right click; target: Mineral Field (10164)
0:01  TSLHyuN       Deselect 6 units
0:01  roxkisSlivko  Right click; target: Mineral Field (10164)
0:02  TSLHyuN       Right click; target: Mineral Field (10170)
0:02  roxkisSlivko  Deselect 6 units
….
```

Action-Log Discussion

Advantages:

- replays can be more compact
  (actions per minute APM vs. ticks per second)
- no redundancy
- may contain more information than the game state
  => User inputs that had no influence on the game state
  (e.g. mouse-movement, points of view, …)

Disadvantage:

- restoring the last state is very expensive, due to rerunning the game
- hard for large numbers of random elements
- computer controlled players/objects:
  - requires deterministic behavior
    (NPC behavior is part of the game and can be simulated as well)
  - AI should be controlled by the same rudimentary commands as human players

Save games in MMOs

Normal games: „small“ game states with decentralized replays/save games

⇒ local clients write peripheral game states into files

For MMO Games:

- complete and consistent game state is only on the server
  => persistence has to be implemented centrally on the server
  => on loss of connection the state on the server counts
- game state is substantially more extensive
  => performance of write operations may slow down the game loop
  => unstructured files are impractical
  (selective loading of players after login)
  => historical information about the course of play would sometimes cause large data volumes

MMOGs and relational databases

Managing large amounts of strictly structured objects

=> Use of a relational database

Advantages of a relational database:

- databases provide certain consistency checks (no duplicates, ...)
- databases support selective requests with efficient indexes
- current game state is immediately available
- databases possess innate recovery mechanisms (protection against system and hardware failures)

Disadvantages:

- structured saving and anomaly avoidance increases processing times of change operations

Persistence via Log-files

all changes in game state are quickly saved

⇒ logging with sequential files

Advantage:

- system has almost no overhead
- writing at the end of a sequential file

  ⇒ minimal waiting time

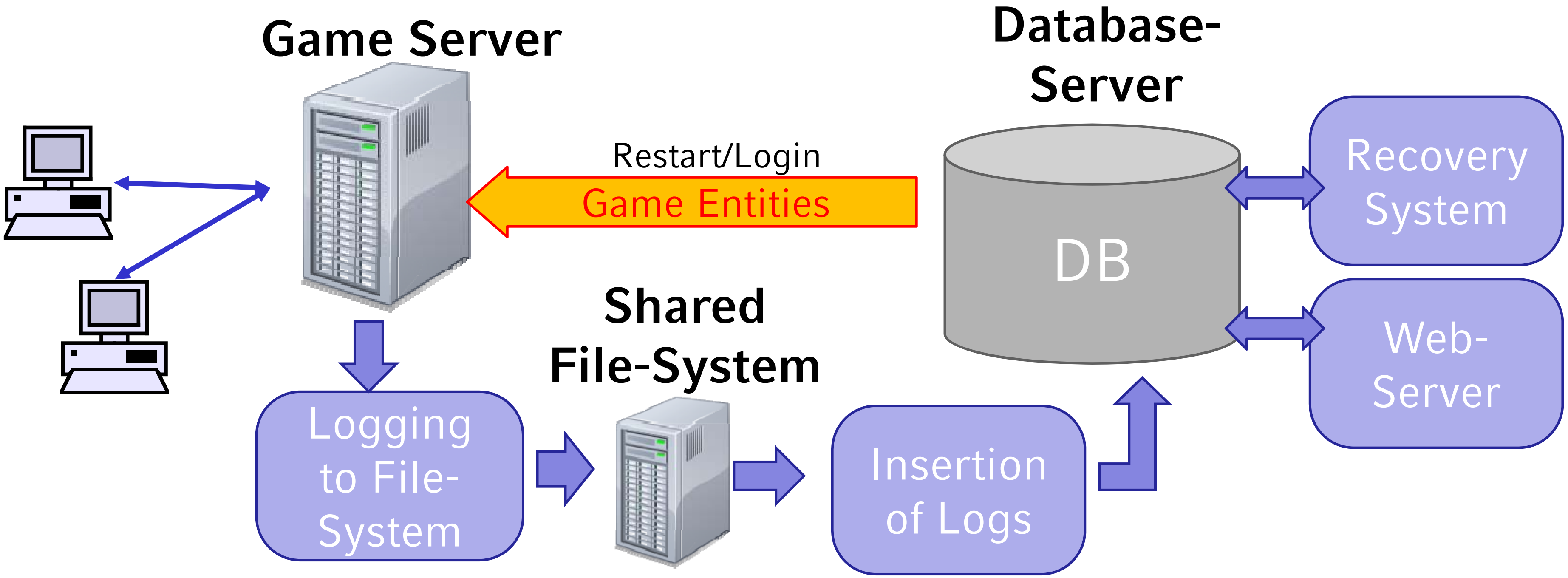
Disadvantages:

- no protection from hard-drive or system errors
- selective requests are not supported
- loading the last consistent game state may require extensive reconstructions by reapplying changes beginning with the last checkpoint

Example for a Hybrid Architecture

- writing data from game server to persistence layer via logging process

  => minimum impact on tick length
- at persistence-server: insertion of log-files into a database server
  - game states are saved durably and secure
    - game state is consistent and redundancy-free
    - includes recovery mechanisms (possible remote storage)
  - information is decoupled from the game for inquiry services (e.g. Armory, …)



Open Issues

- Which logging method is most suitable for volatile systems?
  - change rate for objects
    (How many objects change during a tick?)
  - change complexity
    (are actions more compact than resulting attribute changes?)
  - burstiness of changes
    (Do changes happen periodically in large numbers?)
- Which part of the game state needs to be saved?
  - all moving objects
  - states of all players
  - spatial positions of players and objects
- Concurrency and backlog
  - How fast must different actions be saved?
  - (running vs looting)

Check-Point Recovery Method for Games

- **Check-Point**: consistent image of the game state
- **Check-Point Phase:** time needed to create a check-point.
- **Goal**: Saving the game state with a minimal overhead in the game loop
  => minimal influence on latency
- **Idea**: information is not saved directly, instead all information is copied to a shadow copy
  - data in shadow copy is not affected by actions
  - game loop does not need to wait for the I/O-system
    (uses an asynchronous write-thread)
  - writing may take several ticks, persistence layer lags slightly behind
- **Classification of strategies based on** :
  - bulk-copies vs. selectively copying
  - locking single objects
  - resetting dirty-bits
  - memory usage

Naive-Snapshot

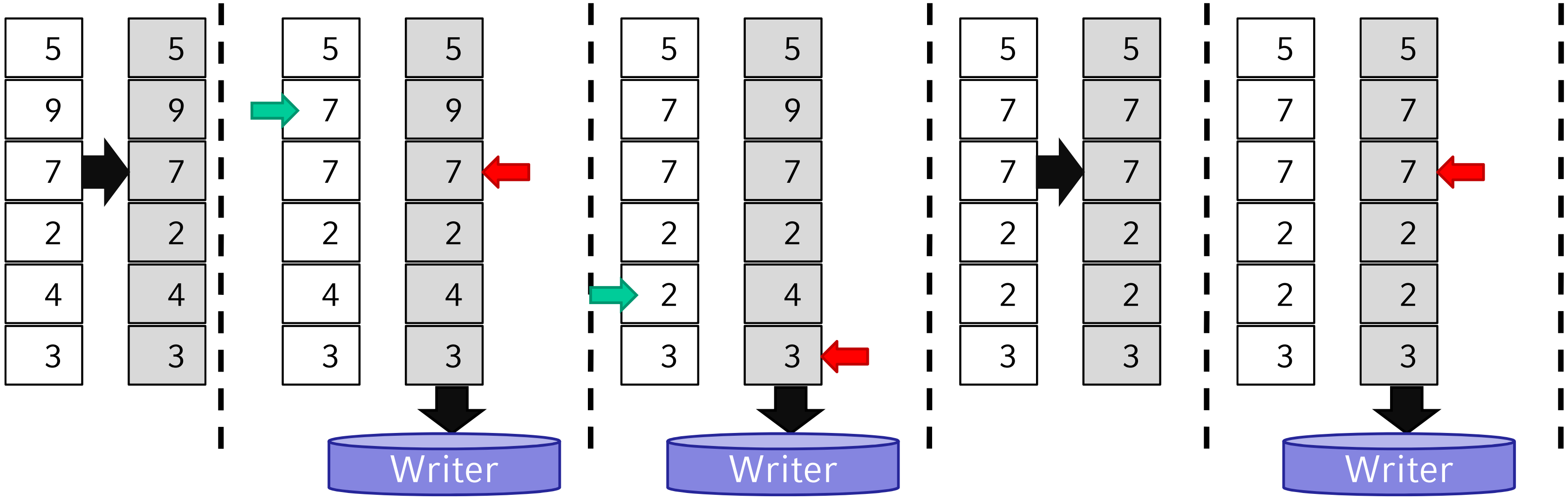
- If write-thread is finished with the last check-point, copy the whole game state into shadow memory.
- After finishing copying and at the start of the next tick, the write-thread writes the copied game state from shadow memory.

Advantages:

- no overhead from locking or bit-resets
- efficient for large numbers of changes

Disadvantages:

- for limited numbers of changes large overhead for copying and writing
- periodically expensive for ticks during where the game state is copied



Copy-On-Update

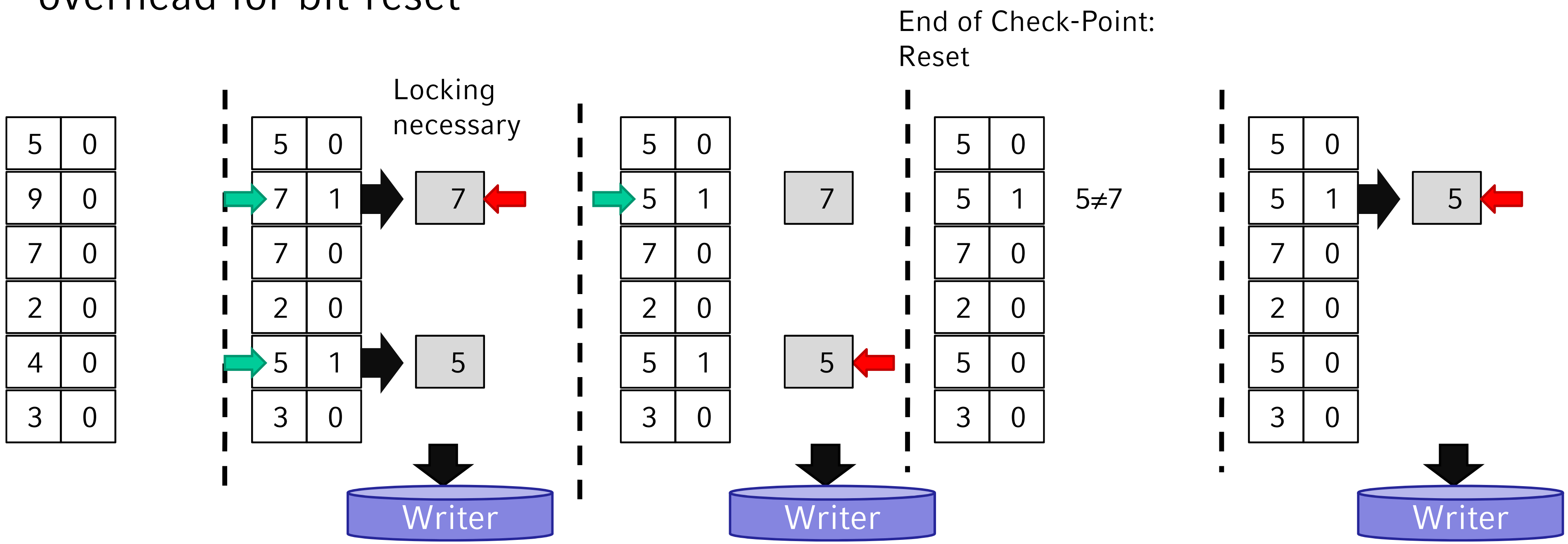
- on change, objects are copied to shadow memory and marked (dirty-bits)
- objects are copied only once per period
- after a check-point has been written markers are reset

Advantages:

- smaller change volume
- better distribution of copies over multiple ticks

Disadvantages:

- requires locking to avoid simultaneous change and copy operations
- overhead for bit-reset



Wait-Free Zigzag

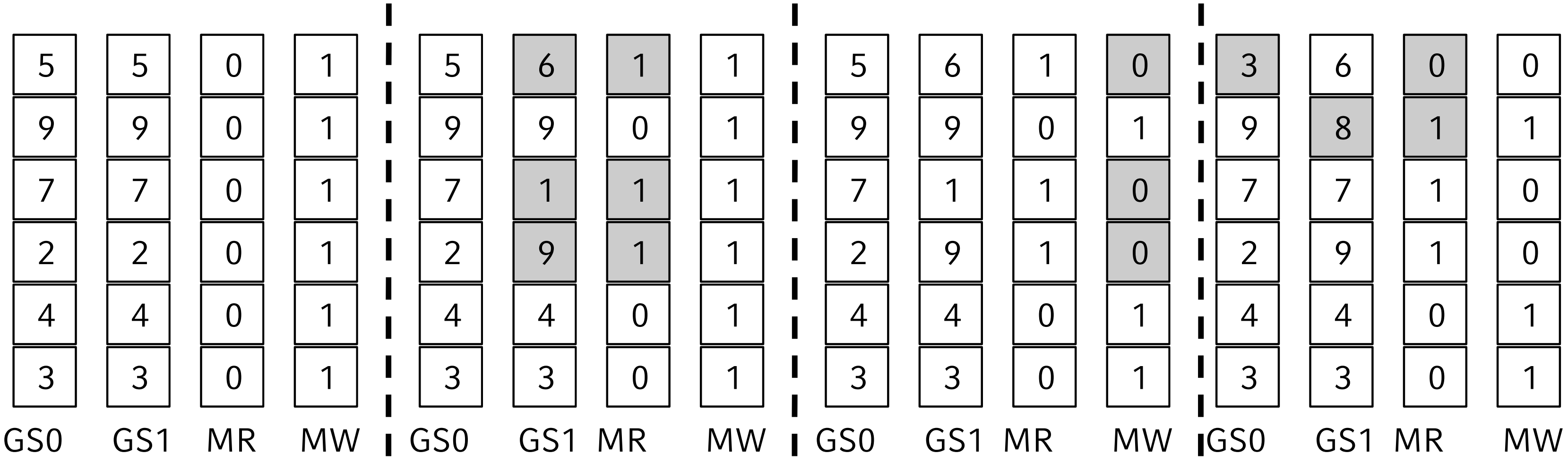
- every object contains two flags referring to a game state: MW (Write-State) and MR (Read-State) for handling actions
- entries in MW are not changed during the checkpoint period
- update: new value is set in $GS[MW_i]$ and MR_i is set to MW_i
- writer-thread reads the element from $GS[\neg MW_i]$ for object i
- end of checkpoint period: if MW_i equals MR_i, flip MW_i

Advantages:

- no locking necessary
- changes can be written over time

Disadvantage:

- still requires bit-reset at the end of each period



GS0	GS1	MR	MW
5	5	0	1
9	9	0	1
7	7	0	1
2	2	0	1
4	4	0	1
3	3	0	1

On Start

GS0	GS1	MR	MW
5	6	1	1
9	9	0	1
7	1	1	1
2	9	1	1
4	4	0	1
3	3	0	1

1. checkpoint period

GS0	GS1	MR	MW
5	6	1	0
9	9	0	1
7	1	1	0
2	9	1	0
4	4	0	1
3	3	0	1

Switch to
2. checkpoint period

GS0	GS1	MR	MW
3	6	0	0
9	8	1	1
7	7	1	0
2	9	1	0
4	4	0	1
3	3	0	1

2. checkpoint period

Wait-Free Ping-Pong

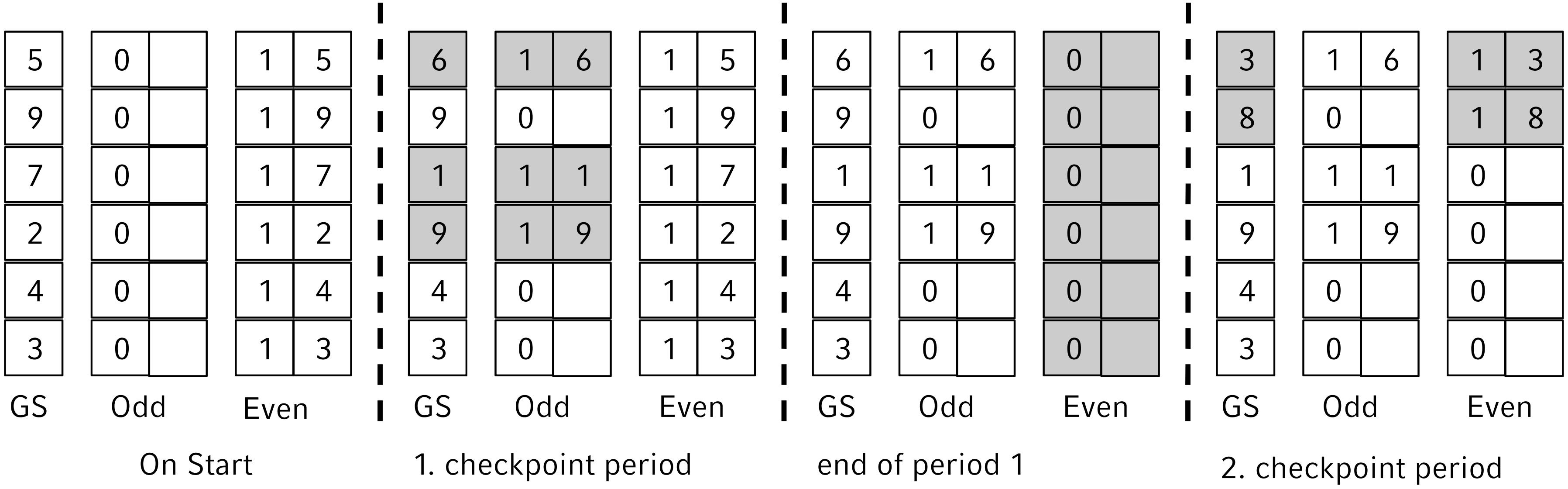
- uses 3 game states:
  action handling (GS), persistence-system (read), persistence-system (write) (odd or even)
- updates always take place in GS and persistence-system (write)
- writer-thread reads persistence-system (read)
- for a new period swap persistence-system(write) and persistence-system(read)

Advantage:

- neither locking nor bit-reset at the end of a period

Disadvantage:

- triple memory requirements instead of double



GS	Odd		Even	
5	0		1	5
9	0		1	9
7	0		1	7
2	0		1	2
4	0		1	4
3	0		1	3

On Start

GS	Odd		Even	
6	1	6	1	5
9	0		1	9
1	1	1	1	7
9	1	9	1	2
4	0		1	4
3	0		1	3

1. checkpoint period

GS	Odd		Even	
6	1	6	0	
9	0		0	
1	1	1	0	
9	1	9	0	
4	0		0	
3	0		0	

end of period 1

GS	Odd		Even	
3	1	6	1	3
8	0		1	8
1	1	1	0	
9	1	9	0	
4	0		0	
3	0		0	

2. checkpoint period

Discussion

- Naive-Snapshot is easiest to implement for very volatile systems with several changes

- the less changes happen, the more advantageous the other methods become

- Wait-Free Ping-Pong and Wait-Free Zig-Zag prevent locking the game entity by the persistence-system

- Wait-Free Ping-Pong also reduces overhead for phase-shifts, but uses a great deal of memory

Learning Goals

- functionality of the persistence system
  - saving a game state
  - saving a sequence of game states (Replay)

- types of save games and replays:
- state-log, transition-log, action-log

- persistency in MMOs:

  Databases, Logging and Hybrid Architectures

- check-point recovery methods for MMOs
  - Naive-Snap Shot
  - Copy-on-update
  - Wait-Free Zigzag
  - Wait-Free Ping-Pong

Literature

- Tuan Cao, Marcos Vaz Salles, Benjamin Sowell, Yao Yue, Alan Demers, Johannes Gehrke, Walker White
  Fast checkpoint recovery algorithms for frequently consistent applications
  In Proceedings of the 2011 International Conference on Management of Data, 2011.

- Marcos Antonio Vaz Salles, Tuan Cao, Benjamin Sowell, Alan J. Demers, Johannes Gehrke, Christoph Koch, Walker M. White
  An Evaluation of Checkpoint Recovery for Massively Multiplayer Online Games
  PVLDB, 2(1): 1258-1269, 2009.

- Kaiwen Zhang, Bettina Kemme, and Alexandre Denault. 2008. **Persistence in massively multiplayer online games**. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games* (NetGames '08), ACM, New York, NY, USA, 53-58.