

# Homework Sheet 6 - Persistence

In the following regard an abstract game with its information stored server sided. Assume the following data to be saved by the server in objects  $O_1, \dots, O_3$ . In the beginning every object  $O_i$  has the value  $o_i$ .

Beginning at time  $t_{10}$  game information should be stored persistently on the hard disk every 10 ticks to avoid data loss by the server in case of a system error.

Assume that writing an object onto the hard disk takes two ticks.

The server thereby performs the following changes of the database:

Outline how the following logging algorithms proceed:

- (a) Naive Snapshot, (b) Copy-on-Update
- (c) Wait-Free Zigzag, (d) Wait-Free Ping-Pong
- (e) Discuss advantages and disadvantages of these methods.

| Time     | Object | New Value |
|----------|--------|-----------|
| $t_6$    | $O_1$  | $o'_1$    |
| $t_9$    | $O_2$  | $o'_2$    |
| $t_{12}$ | $O_3$  | $o'_3$    |
| $t_{15}$ | $O_1$  | $o''_1$   |
| $t_{16}$ | $O_3$  | $o''_3$   |
| $t_{22}$ | $O_2$  | $o''_2$   |
| $t_{22}$ | $O_3$  | $o'''_3$  |

# 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

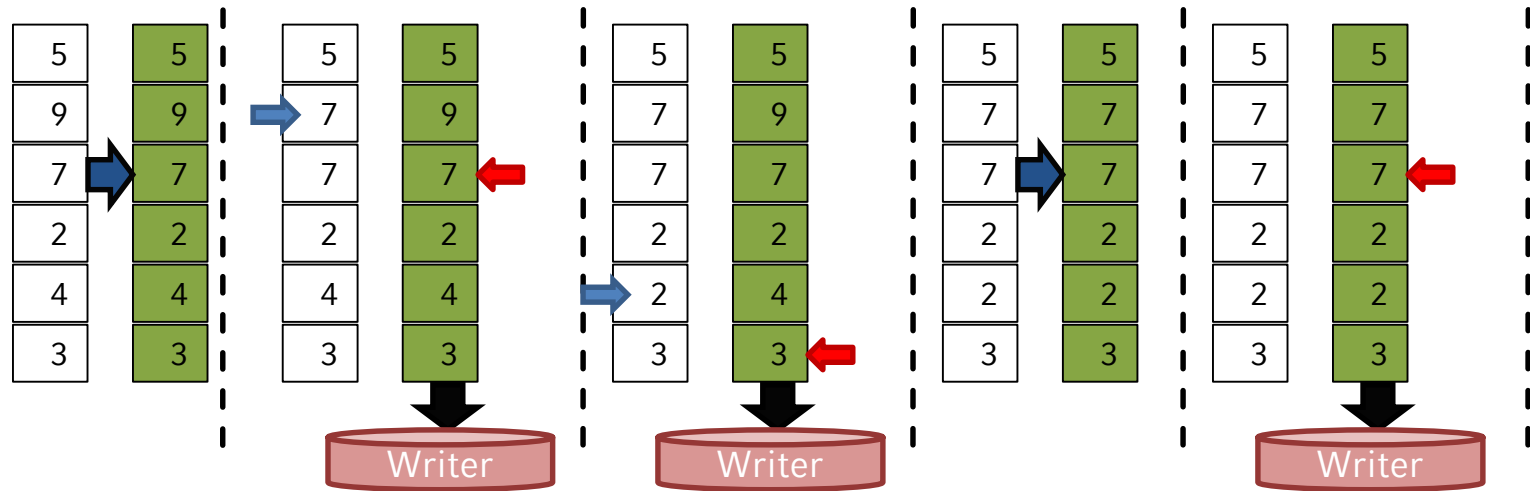
## **Advantages:**

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

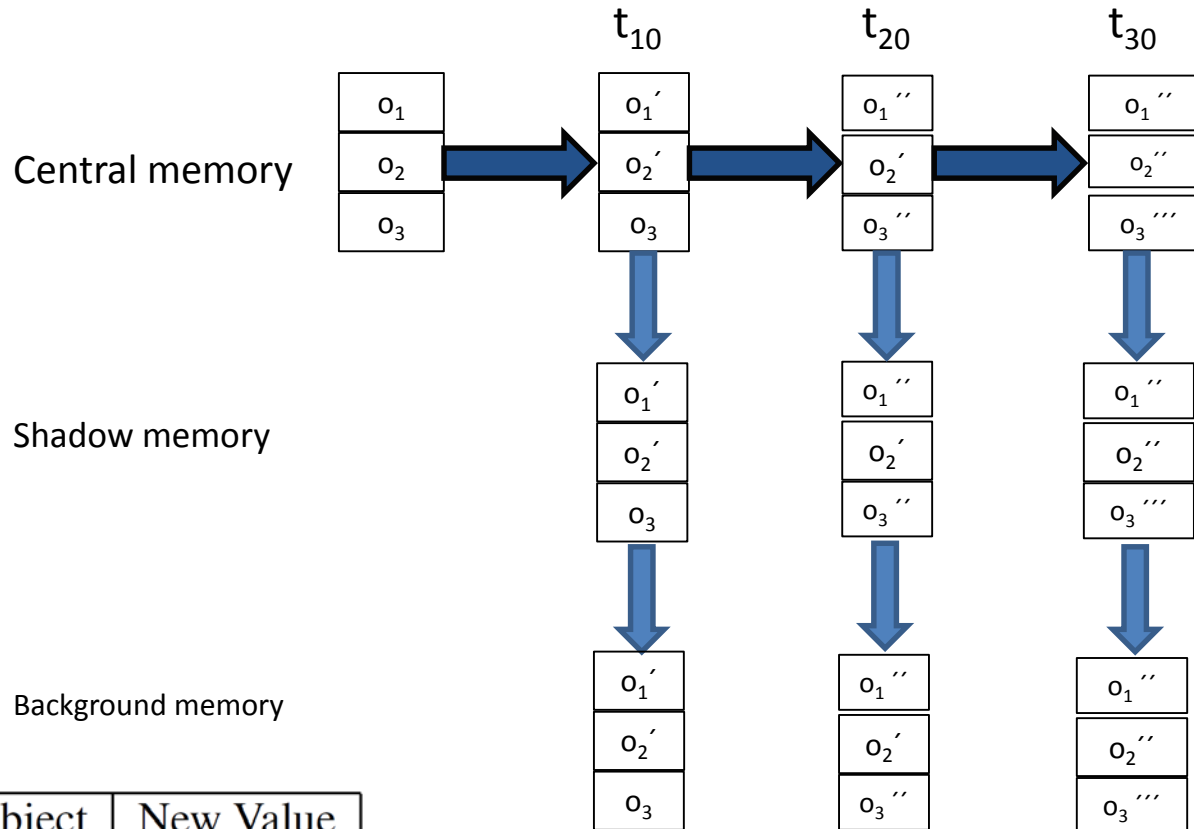
## **Disadvantages:**

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

# Naive-Snapshot



# Naive-Snapshot



| Time     | Object | New Value |
|----------|--------|-----------|
| $t_6$    | $O_1$  | $o_1'$    |
| $t_9$    | $O_2$  | $o_2'$    |
| $t_{12}$ | $O_3$  | $o_3'$    |
| $t_{15}$ | $O_1$  | $o_1''$   |
| $t_{16}$ | $O_3$  | $o_3''$   |
| $t_{22}$ | $O_2$  | $o_2''$   |
| $t_{22}$ | $O_3$  | $o_3'''$  |

Writer needs  $3 * 2s = 6$  seconds to transfer the complete game state from the shadow memory to the background memory (secondary storage)

Writer starts writing out every 10 seconds => Since  $6s < 10s$  he is finished with writing before the next game state is copied to the shadow memory

# 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

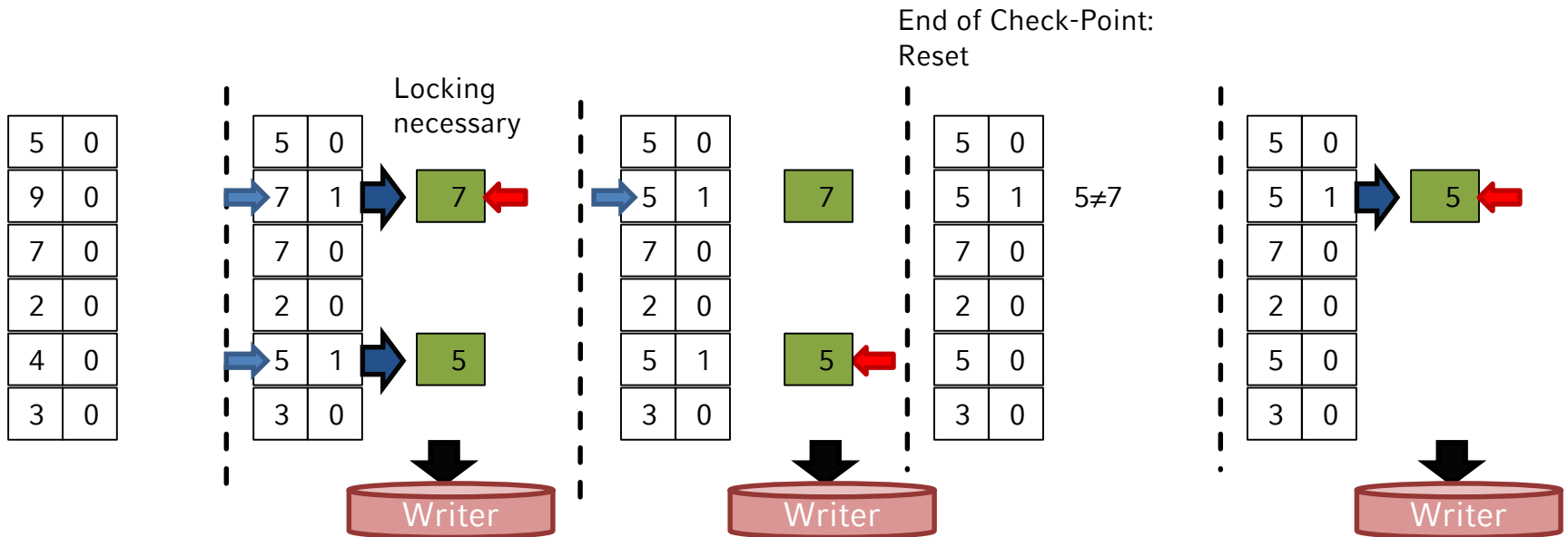
## **Advantages:**

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

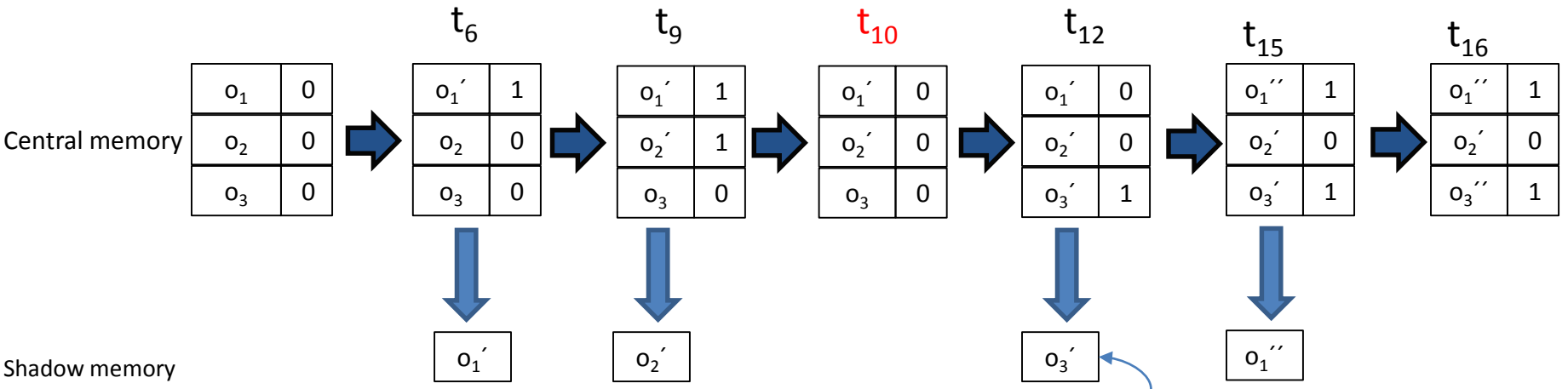
## **Disadvantages:**

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

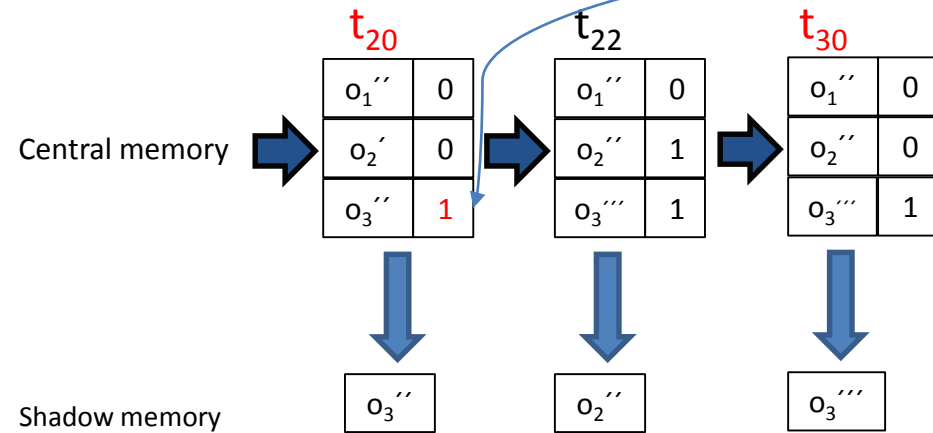
# Copy-On-Update



# Copy-On-Update



$o_3'$  is written out since  $t_{14}$   
 $o_3'' (\neq o_3')$  is the most actual value =>  
 Has to be written into secondary  
 storage still at time  $t_{20}$  (next checkpoint)



| Time     | Object | New Value |
|----------|--------|-----------|
| $t_6$    | $O_1$  | $o_1'$    |
| $t_9$    | $O_2$  | $o_2'$    |
| $t_{12}$ | $O_3$  | $o_3'$    |
| $t_{15}$ | $O_1$  | $o_1''$   |
| $t_{16}$ | $O_3$  | $o_3''$   |
| $t_{22}$ | $O_2$  | $o_2''$   |
| $t_{22}$ | $O_3$  | $o_3'''$  |

# Wait-Free Zigzag

- every object contains two flags referring to a game state (GS): MW (Write-State) and MR (Read-State) for handling actions
- entries in GS[MW] are not changed during this period
- for changes MR is set to MW
- writer-thread reads the element from GS[-MW<sub>i</sub>] for object i

## Advantages:

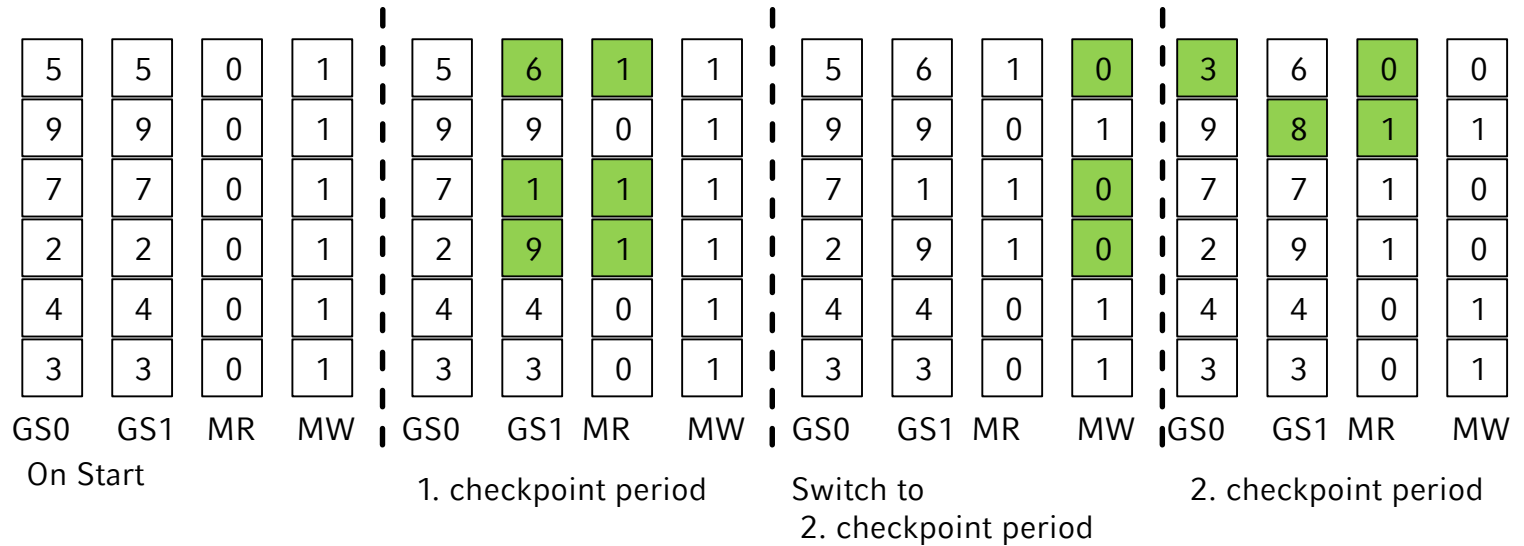
- no locking necessary
- changes can be written over time

## Disadvantage:

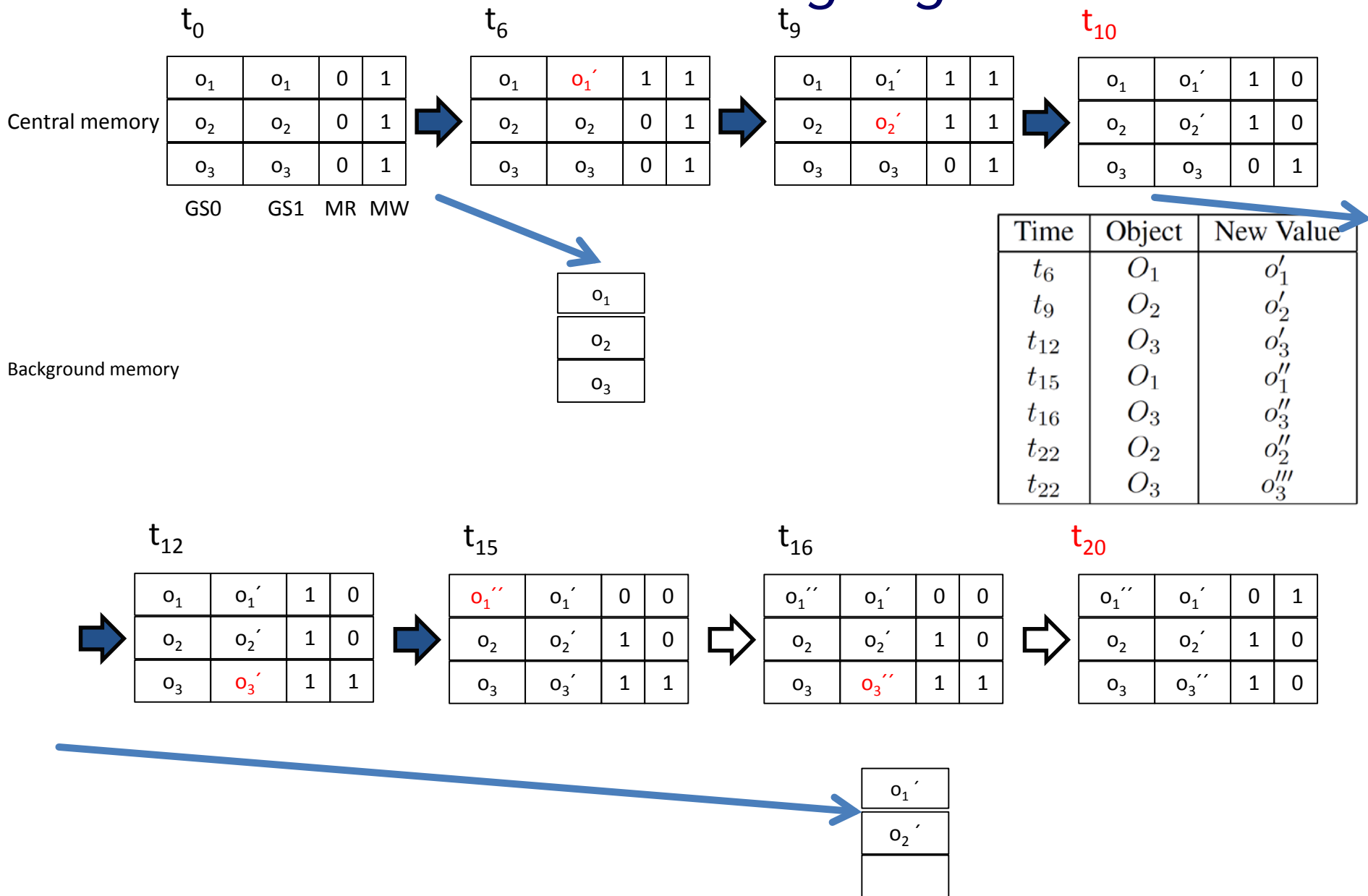
- still requires bit-reset at the end of each period



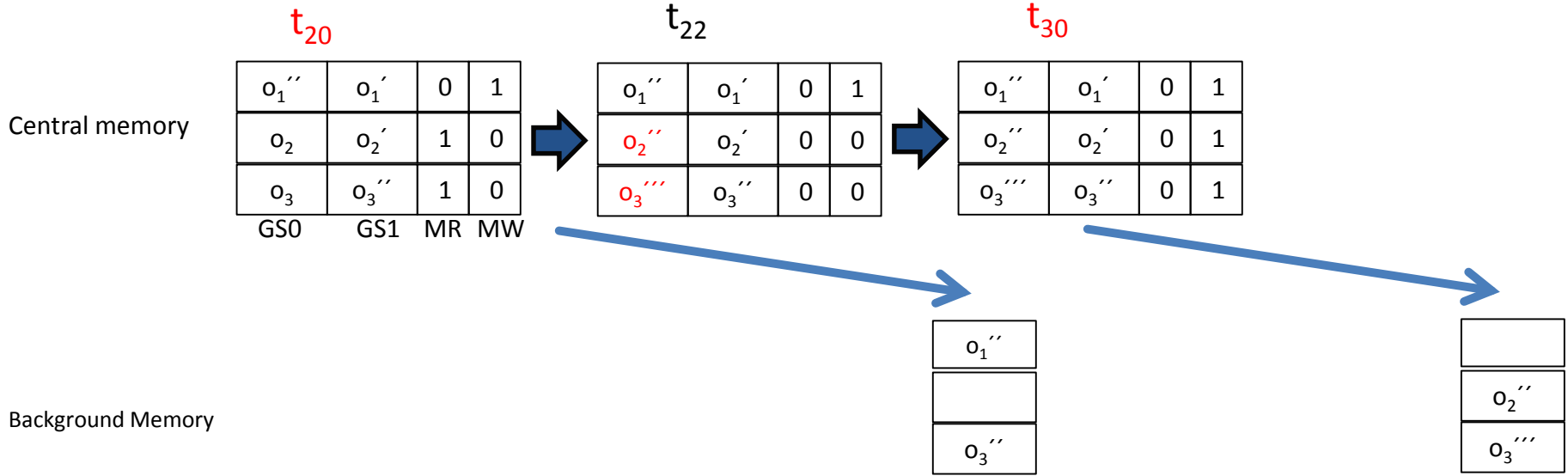
# Wait-Free Zigzag



# Wait-Free Zigzag



# Wait-Free Zigzag (2)



| Time     | Object | New Value |
|----------|--------|-----------|
| $t_6$    | $O_1$  | $o_1'$    |
| $t_9$    | $O_2$  | $o_2'$    |
| $t_{12}$ | $O_3$  | $o_3'$    |
| $t_{15}$ | $O_1$  | $o_1''$   |
| $t_{16}$ | $O_3$  | $o_3''$   |
| $t_{22}$ | $O_2$  | $o_2''$   |
| $t_{22}$ | $O_3$  | $o_3'''$  |

# 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)

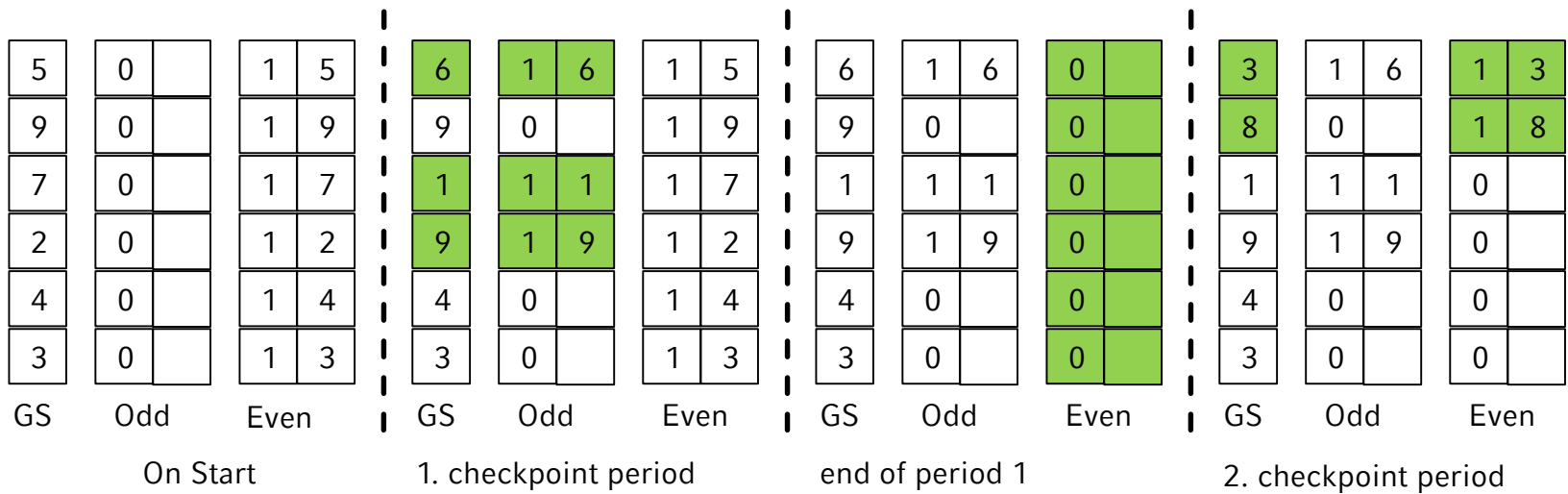
## **Advantage:**

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

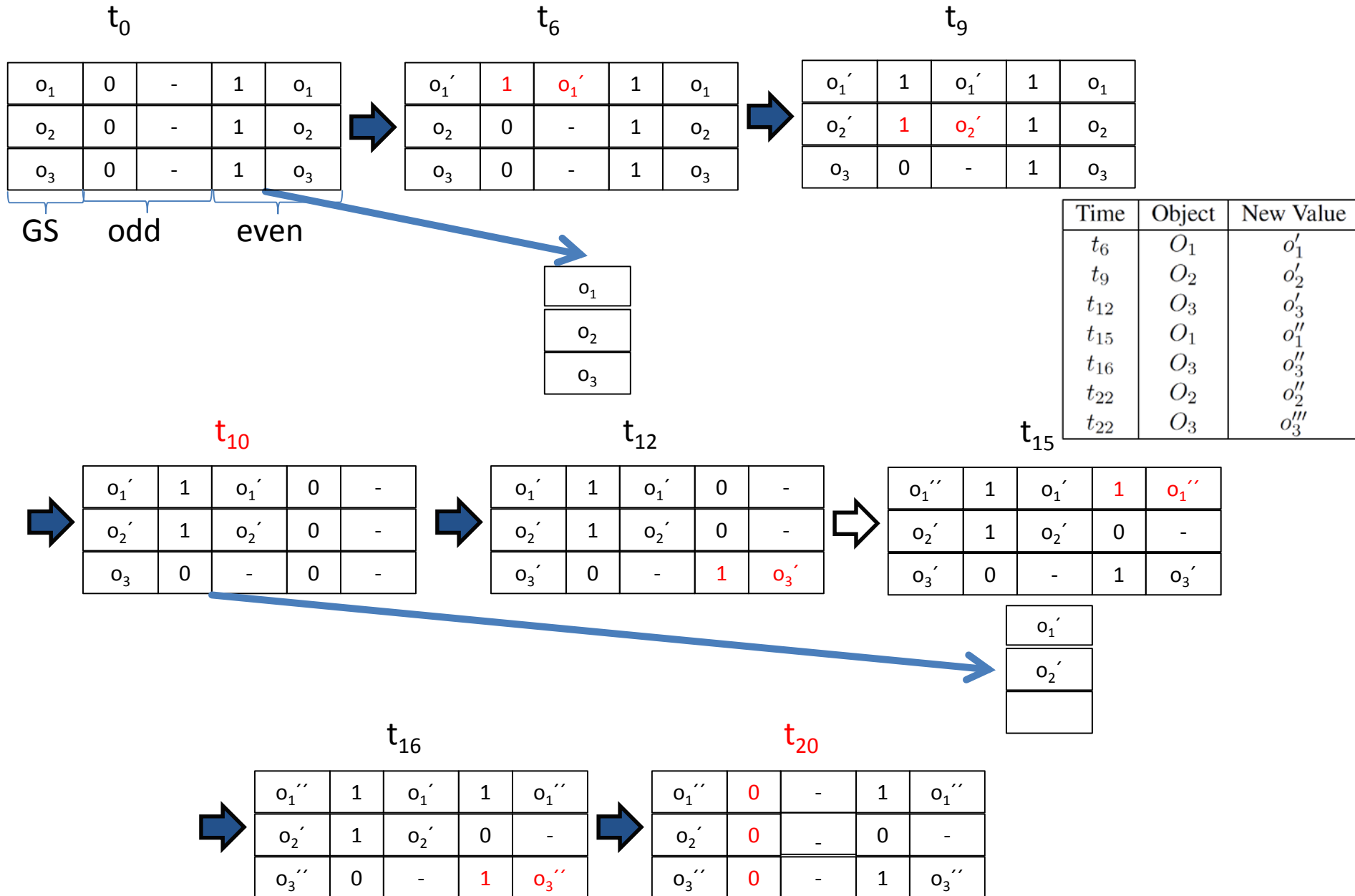
## **Disadvantage:**

- triple memory requirements instead of double

# Wait-Free Ping-Pong



# Wait-Free Ping-Pong



# Wait-Free Ping-Pong (2)

$t_{20}$

|         |   |   |   |         |
|---------|---|---|---|---------|
| $o_1''$ | 0 | - | 1 | $o_1''$ |
| $o_2'$  | 0 | - | 0 | -       |
| $o_3''$ | 0 | - | 1 | $o_3''$ |

$t_{22}$

|          |   |          |   |         |
|----------|---|----------|---|---------|
| $o_1''$  | 0 | -        | 1 | $o_1''$ |
| $o_2''$  | 1 | $o_2''$  | 0 | -       |
| $o_3'''$ | 1 | $o_3'''$ | 1 | $o_3''$ |

GS    odd    even

|         |
|---------|
| $o_1''$ |
|         |
| $o_3''$ |

$t_{30}$

|          |   |          |   |   |
|----------|---|----------|---|---|
| $o_1''$  | 0 | -        | 0 | - |
| $o_2''$  | 1 | $o_2''$  | 0 | - |
| $o_3'''$ | 1 | $o_3'''$ | 0 | - |

|          |
|----------|
|          |
| $o_2''$  |
| $o_3'''$ |

| Time     | Object | New Value |
|----------|--------|-----------|
| $t_6$    | $O_1$  | $o_1'$    |
| $t_9$    | $O_2$  | $o_2'$    |
| $t_{12}$ | $O_3$  | $o_3'$    |
| $t_{15}$ | $O_1$  | $o_1''$   |
| $t_{16}$ | $O_3$  | $o_3''$   |
| $t_{22}$ | $O_2$  | $o_2''$   |
| $t_{22}$ | $O_3$  | $o_3'''$  |

# 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