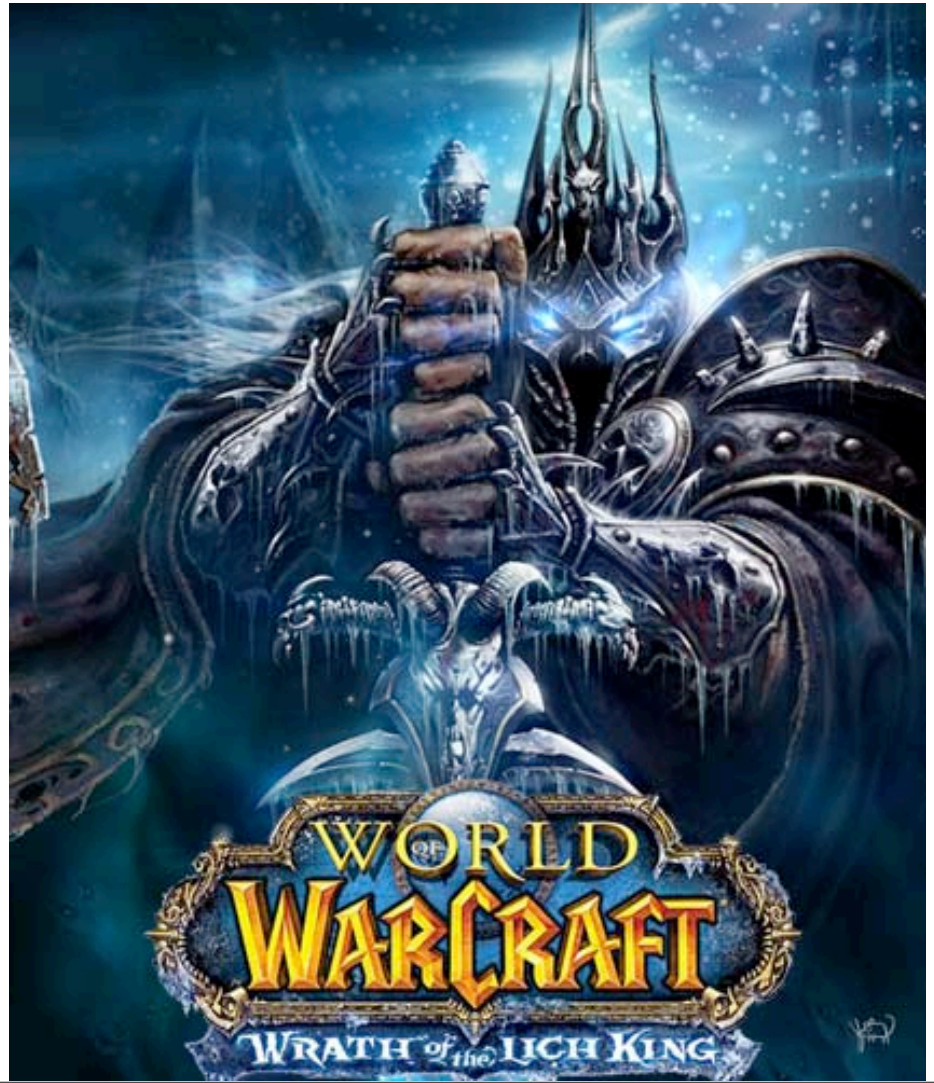
An Evaluation of Checkpoint Recovery For Massively Multiplayer Online Games

Marcos Vaz Salles Tuan Cao Benjamin Sowell
Alan Demers Johannes Gehrke Christoph Koch Walker White

Cornell University

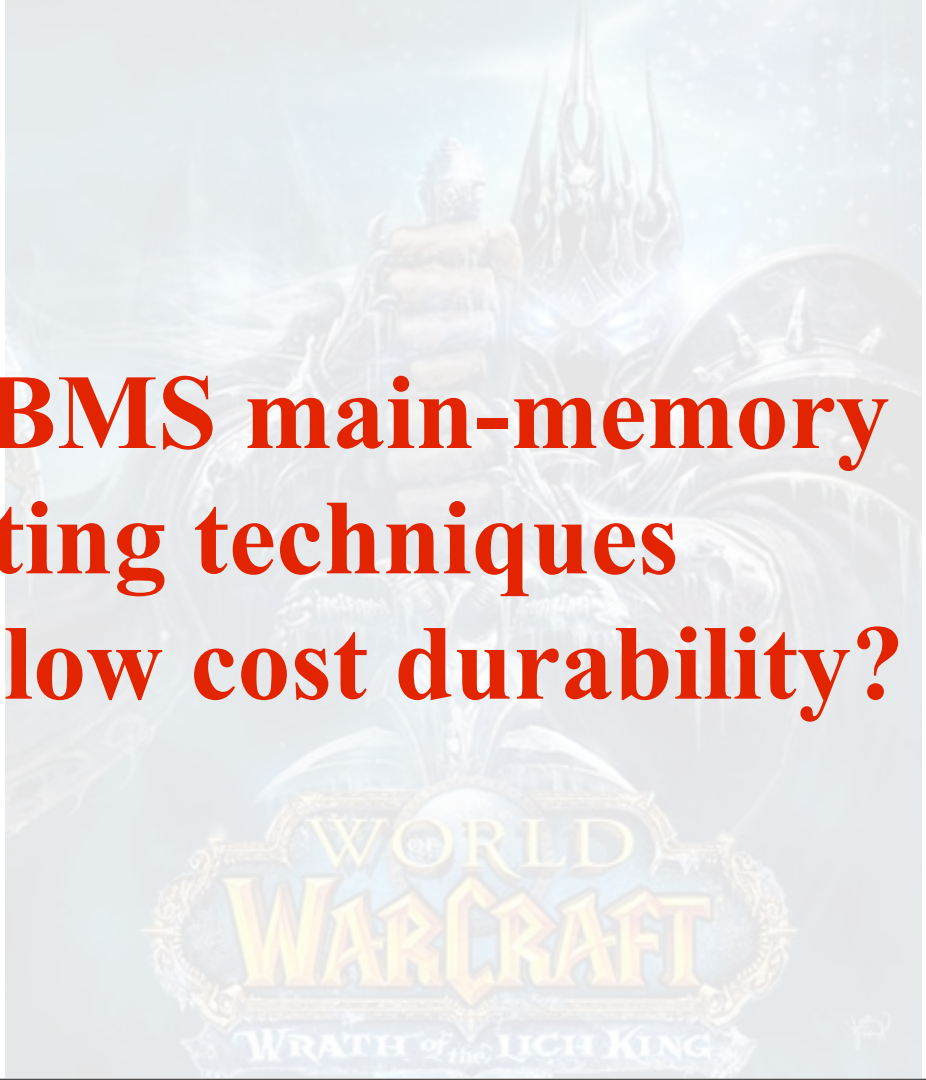
MMO Games

- Simulate **long-lived**, **interactive** virtual worlds
- **Persist** across user sessions



MMO Games

- Simulate **long-lived**, **interactive** virtual worlds
- **Persist** across user sessions



Can existing DBMS main-memory checkpointing techniques help to achieve low cost durability?

Our Contributions

- Show how to adapt consistent checkpointing techniques
- Provide a thorough simulation model and evaluate six different recovery strategies

Our Contributions

- Show how to adapt consistent checkpointing techniques
- Provide a thorough simulation model and evaluate six different recovery strategies

Why simulation?

➡ Ability to evaluate different types of hardware

Talk Outline

- Introduction to MMOs
  - The game state
  - The game loop
- Consistent Checkpointing Techniques
  - Requirements
  - Performance Criteria
  - Performance Model
- Experiments
  - Experimental Setup
  - Model Validation
  - Results
- Conclusion

The Game State

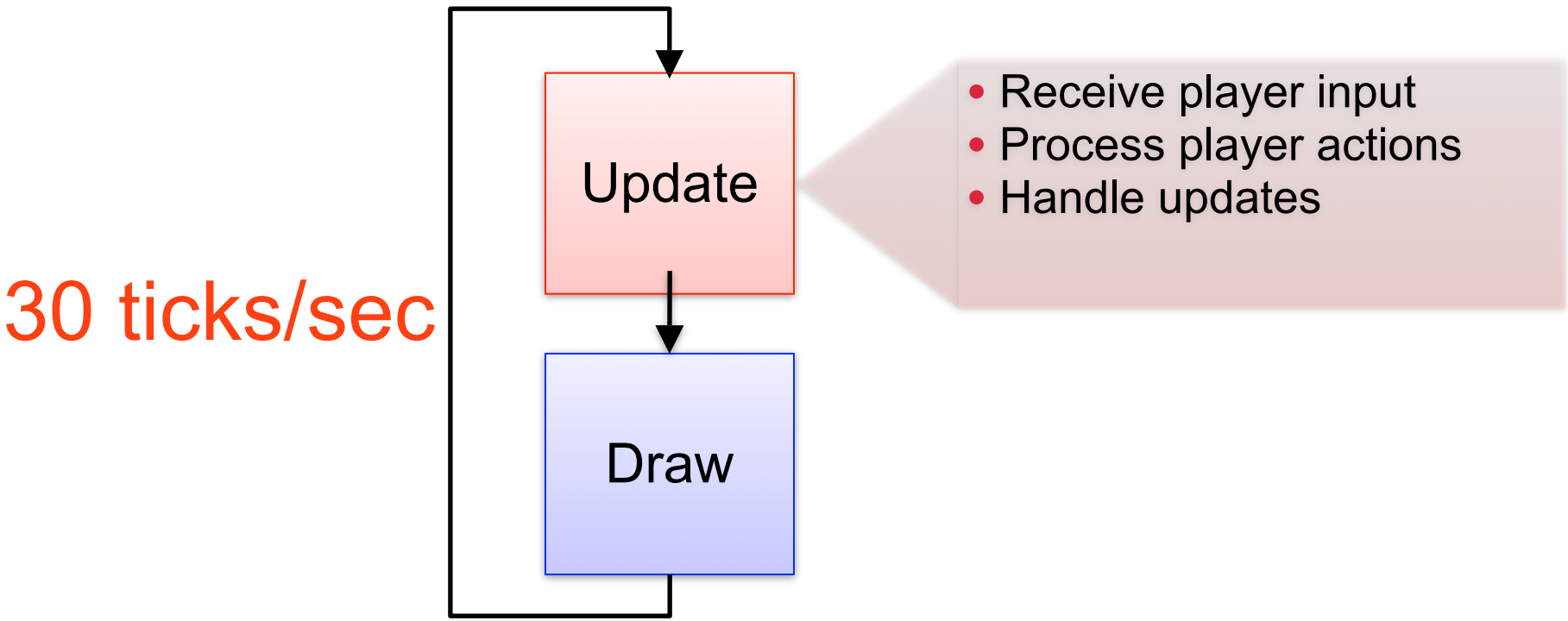
- Tables containing game objects
  (e.g., attributes of game characters)

- Must be kept in main memory

The Game State

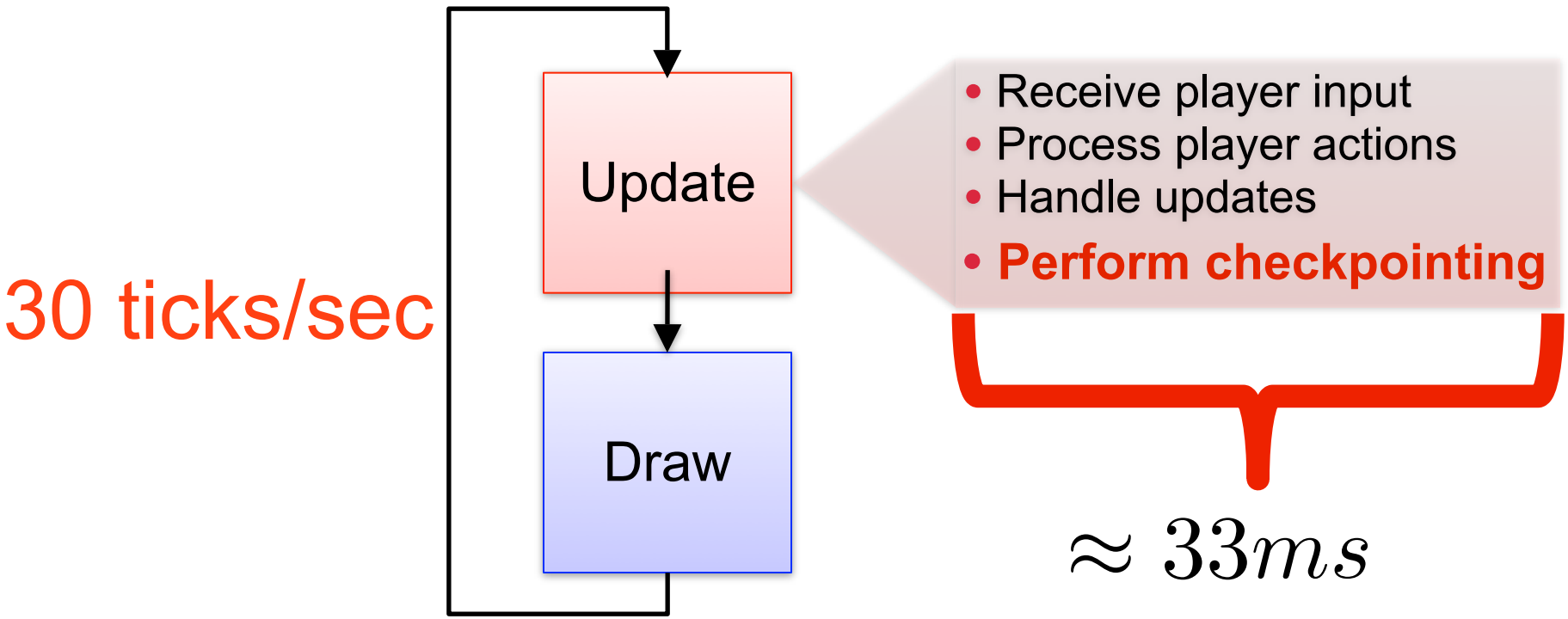
Race	Strength	Agility	Stamina	Intellect	Spirit	Armor	Health
Human	108	73	99	29	46	146	2169
Dwarf	110	69	102	28	41	138	2199
Night Elf	105	78	98	29	42	156	2159
Gnome	103	76	98	33	42	152	2159
Draenei	109	70	98	30	44	140	2159
Orc	111	70	101	26	45	140	2189
Troll	109	75	100	25	43	150	2179
Undead	107	71	100	27	47	142	2179
Blood Elf	105	75	97	33	41	150	2149
Tauren	113	68	101	24	44	136	2298

The Game Loop



The game state is consistent at the end of every tick.

The Game Loop



The game state is consistent at the end of every tick.

Requirements

- Small overhead
  - ✦ Entire checkpointing process must fit into the game simulation
- Uniform overhead
  - ✦ Low latency, no hiccup in the game
- No data loss
  - ✦ Recover to the point of the crash

Requirements

- Small overhead
- Uniform overhead
- No data loss

Requirements

➡ Performance Criteria

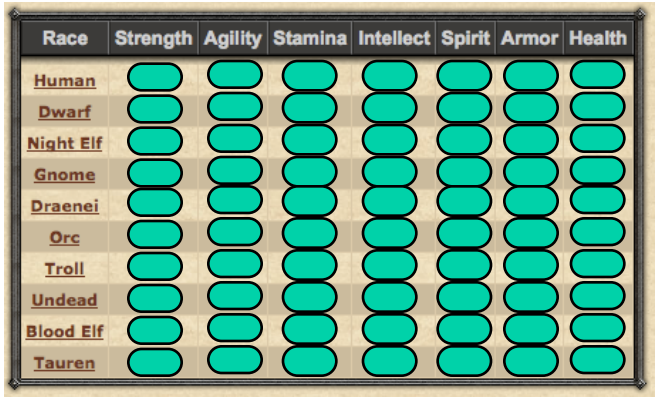
- Small overhead
  - ➡ Average Overhead Time
- Uniform overhead
  - ➡ Overhead Distribution
- No data loss
  - ➡ Checkpointing Time
  - ➡ Recovery Time

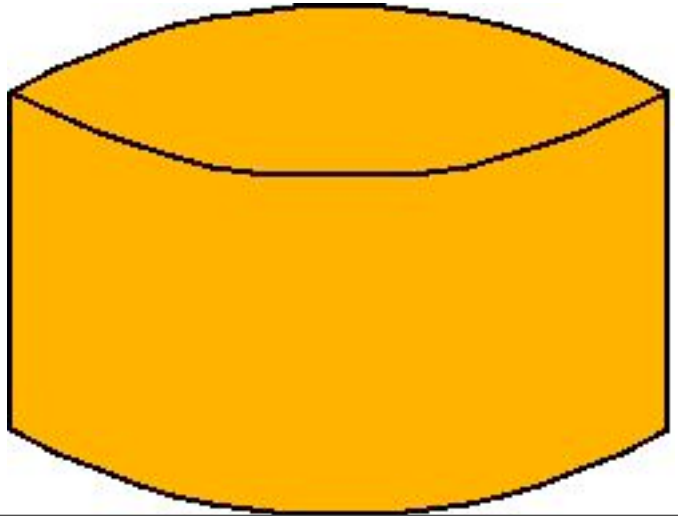
Checkpointing Techniques

All Objects	Dirty Objects
Log Based	Double Backup
Eager Copy	Copy On Update

Checkpointing Techniques

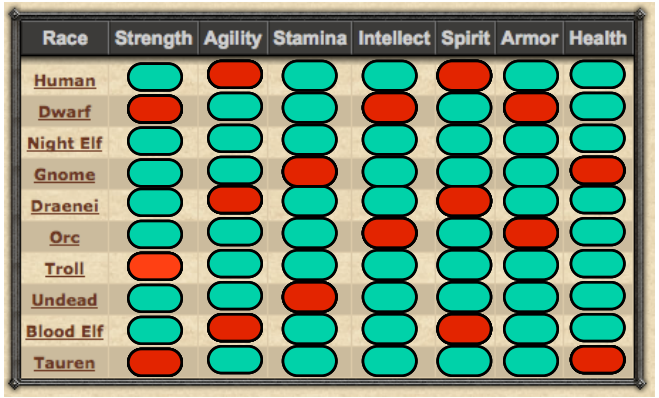
All Objects	Dirty Objects
Log	Double Backups
Eager Copy	Copy On Update

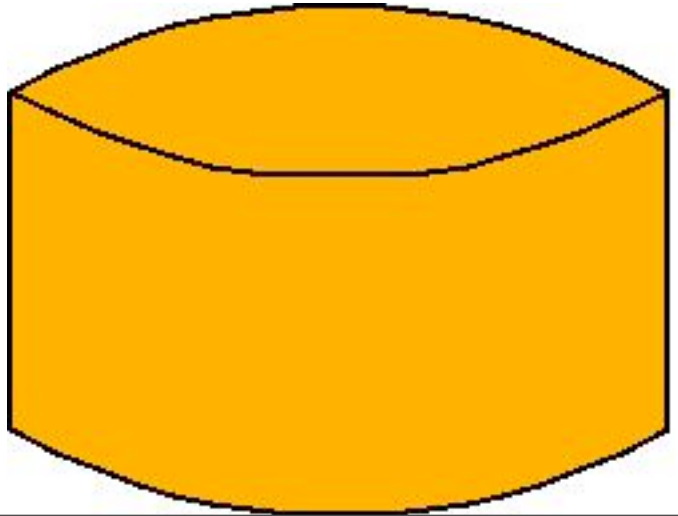




Checkpointing Techniques

All Objects	Dirty Objects
Log	Double Backups
Eager Copy	Copy On Update

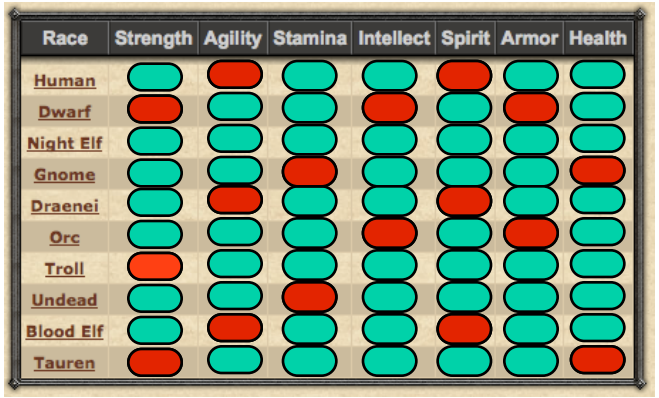


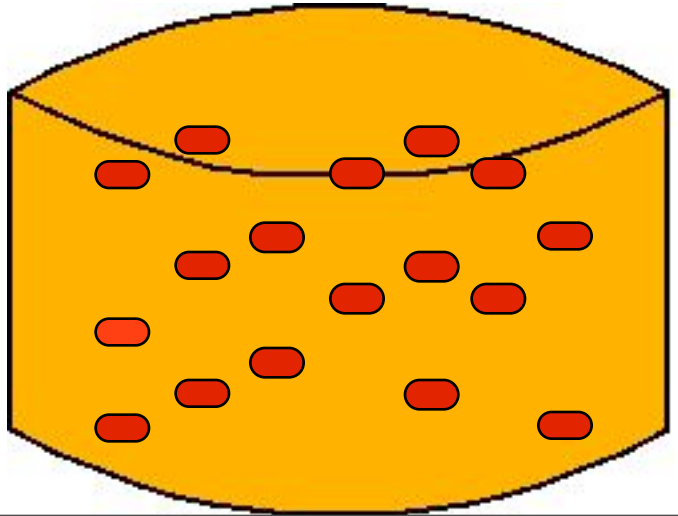


Checkpointing Techniques

All Objects	Dirty Objects
Log	Double Backups
Eager Copy	Copy On Update

Dirty Objects

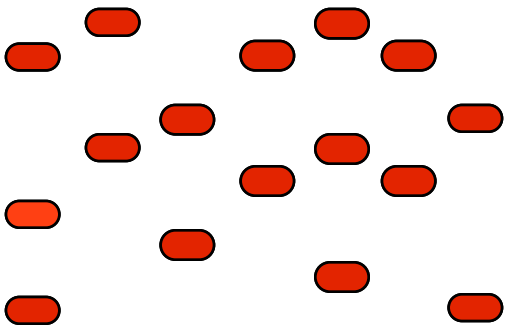


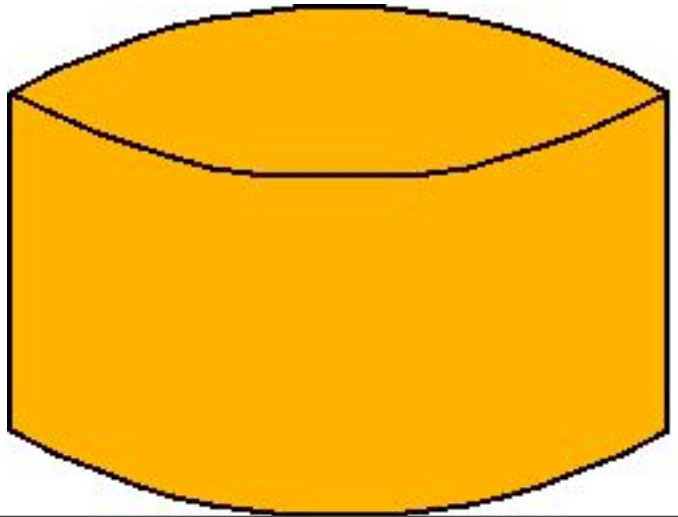


Checkpointing Techniques

All Objects	Dirty Objects
Log Based	Double Backup
Eager Copy	Copy On Update

Log-Based

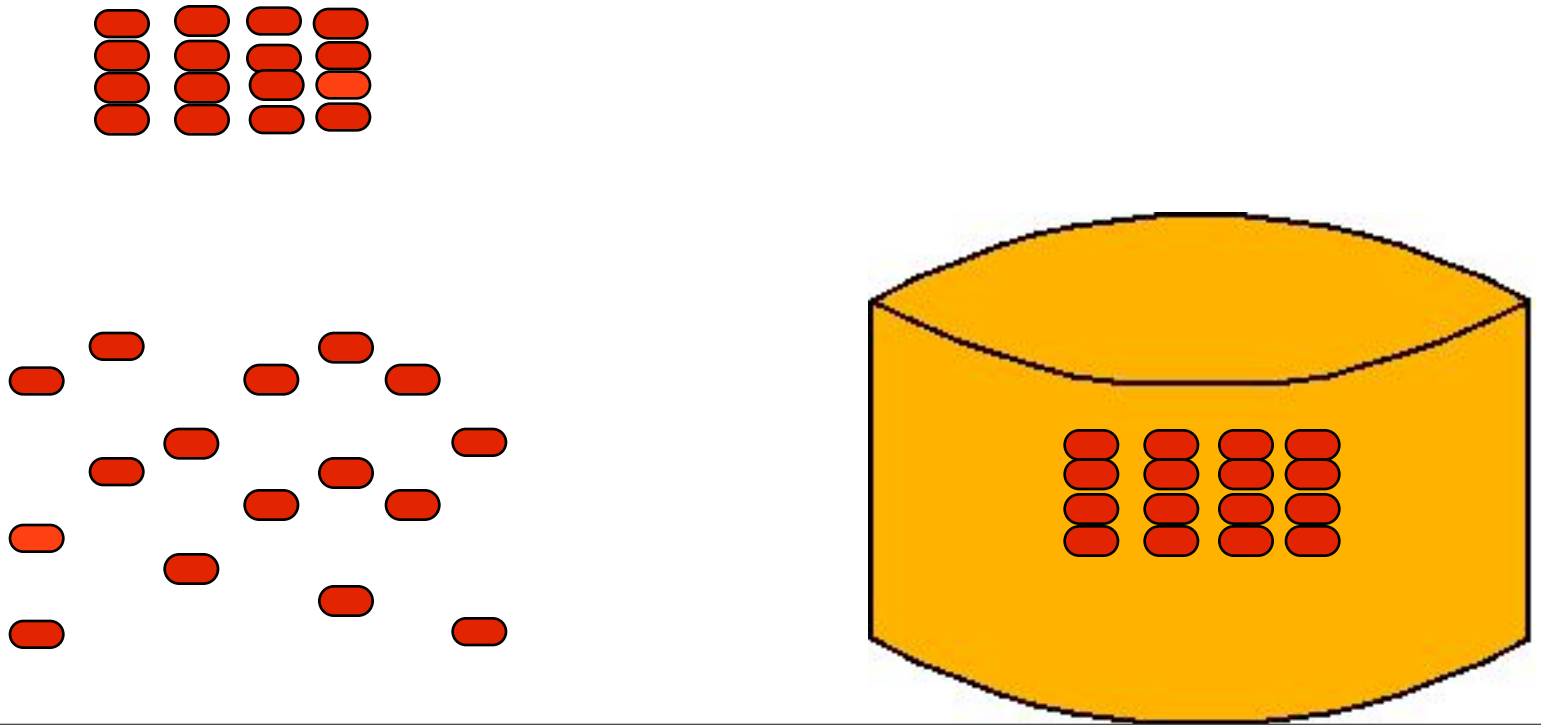




Checkpointing Techniques

All Objects	Dirty Objects
Log Based	Double Backup
Eager Copy	Copy On Update

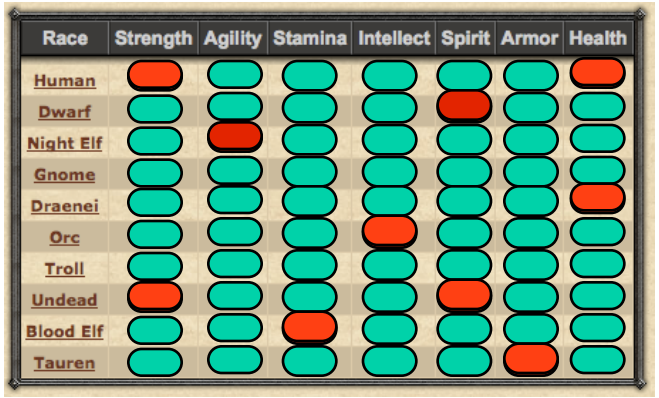
Log-Based

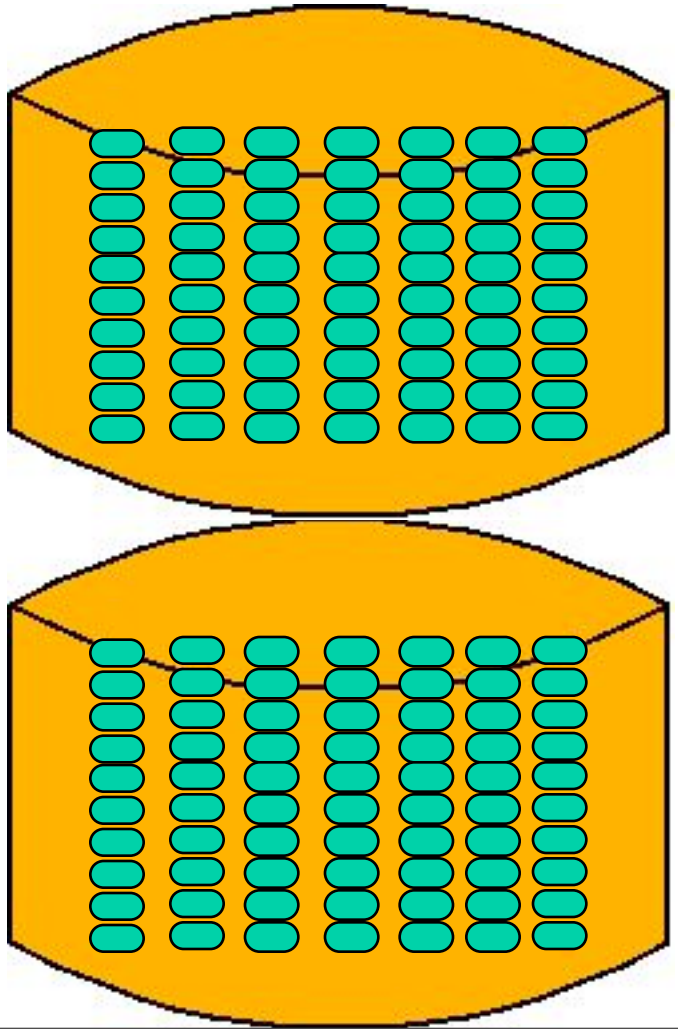


Checkpointing Techniques

All Objects	Dirty Objects
Log Based	Double Backup
Eager Copy	Copy On Update

Double Backup



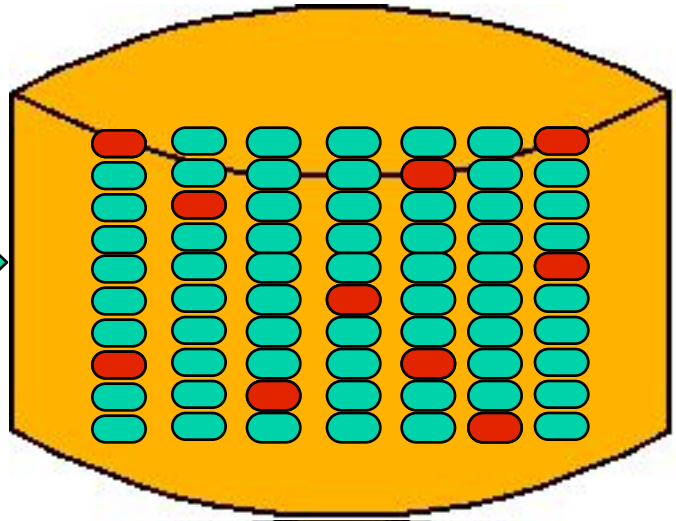


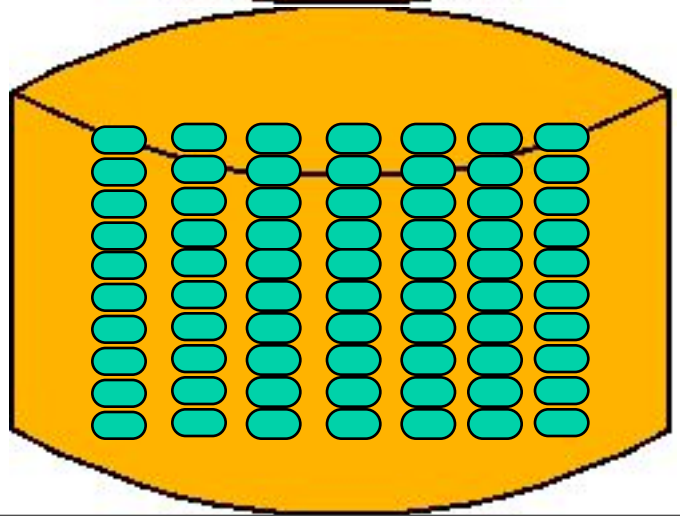
Checkpointing Techniques

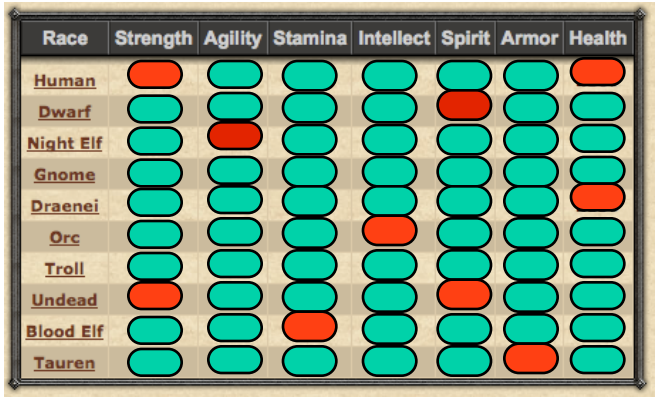
All Objects	Dirty Objects
Log Based	Double Backup
Eager Copy	Copy On Update

Double Backup

Containing a consistent copy of the game state





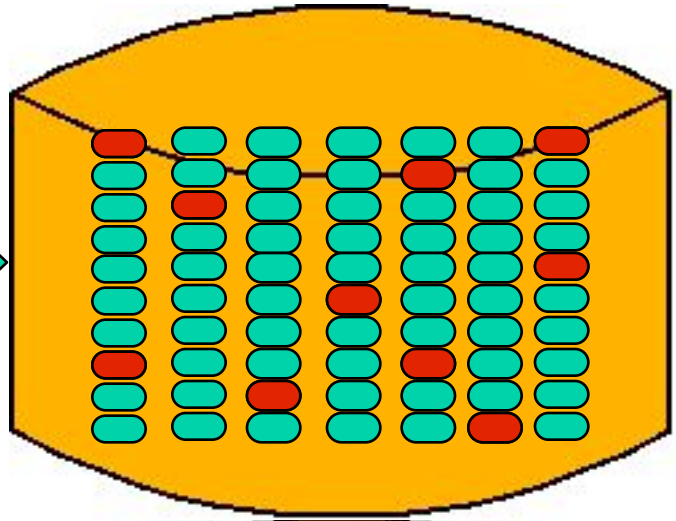


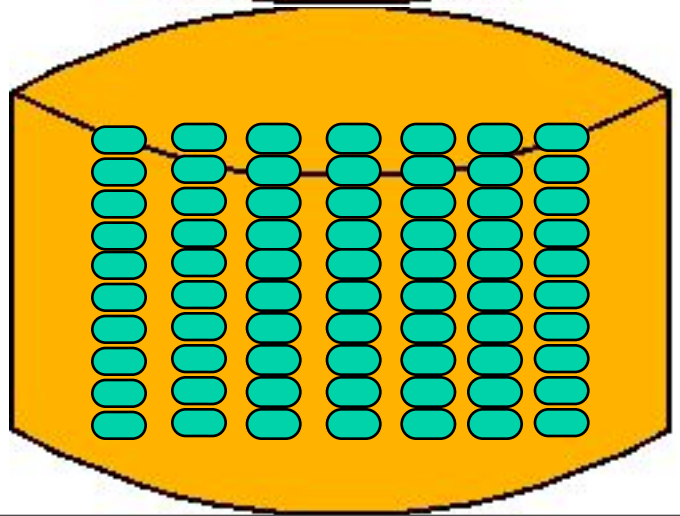
Checkpointing Techniques

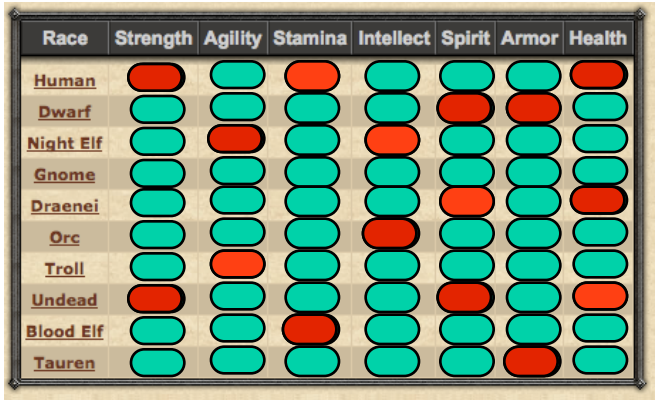
All Objects	Dirty Objects
Log Based	Double Backup
Eager Copy	Copy On Update

Double Backup

Containing a consistent copy of the game state





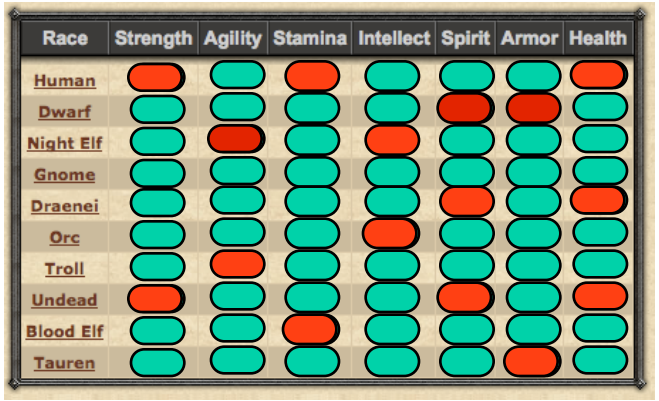


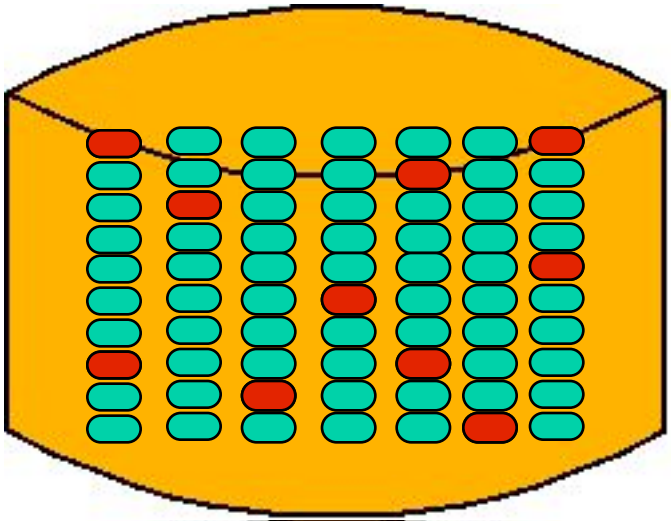
Checkpointing Techniques

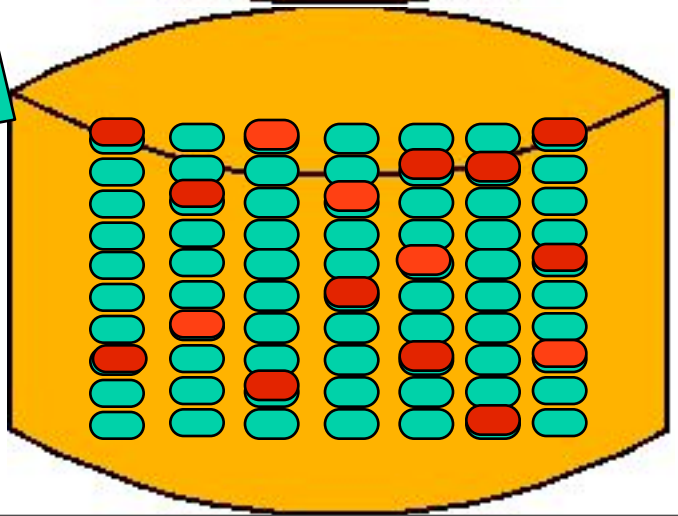
All Objects	Dirty Objects
Log Based	Double Backup
Eager Copy	Copy On Update

Double Backup

Containing a consistent copy of the game state

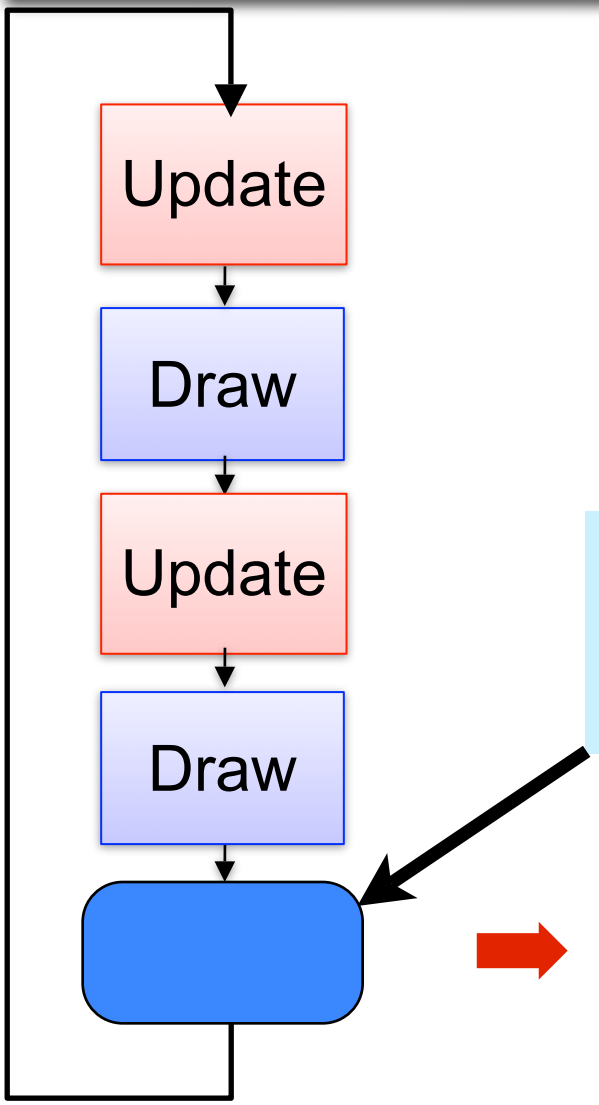






Checkpointing Techniques

All Objects	Dirty Objects
Log	Double Backups
Eager Copy	Copy On Update



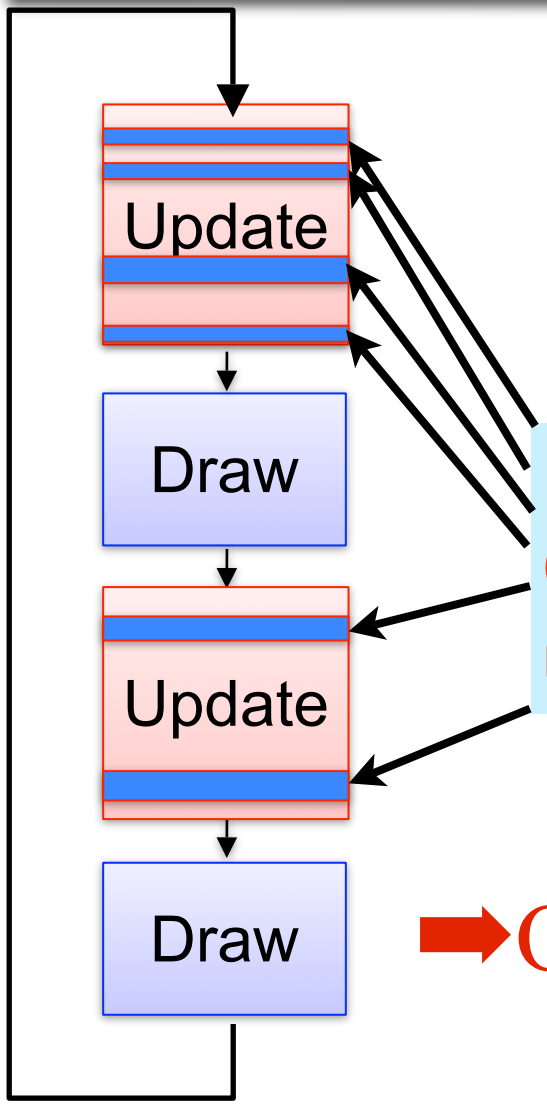
Eager Copy

Pause the simulation loop
Copy objects to memory

➡ Creates a hiccup in the game

Checkpointing Techniques

All Objects	Dirty Objects
Log	Double Backups
Eager Copy	Copy On Update



Copy On Update

Intercept each update
Copy object to memory before updating it

➡Overhead is spread over ticks

Checkpointing Algorithms

	All Objects	Dirty Objects	Eager Copy	Copy On Update	Double Backup	Log
Naive-Snapshot	X		X			X
Dribble-And-Copy-On-Update	X			X		X
Atomic-Copy-Dirty-Objects		X	X		X	
Partial-Redo		X	X			X
Copy-On-Update		X		X	X	
Copy-On-Update-Partial-Redo		X		X		X

Checkpointing Algorithms

	All Objects	Dirty Objects	Eager Copy	Copy On Update	Double Backup	Log
Naive-Snapshot	X		X			X
Dribble-And-Copy-On-Update	X			X		X
Atomic-Copy-Dirty-Objects		X	X		X	
Partial-Redo		X	X			X
Copy-On-Update		X		X	X	
Copy-On-Update-Partial-Redo		X		X		X

Checkpointing Algorithms

Naive-Snapshot

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

Partial-Redo

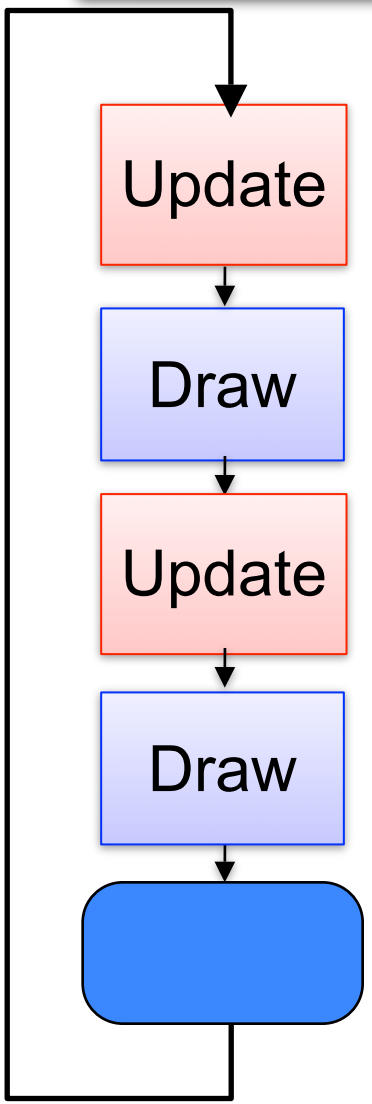
All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

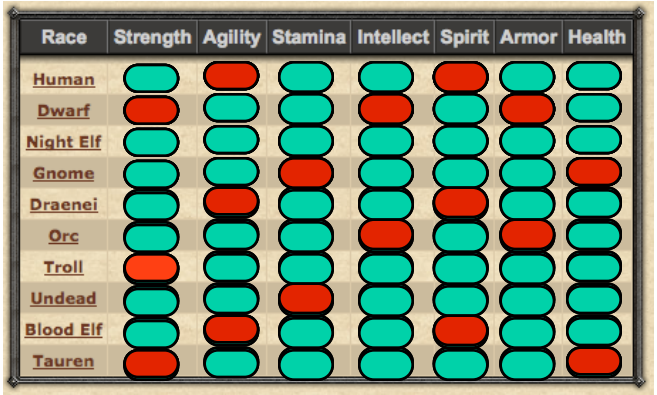
Copy-On-Update

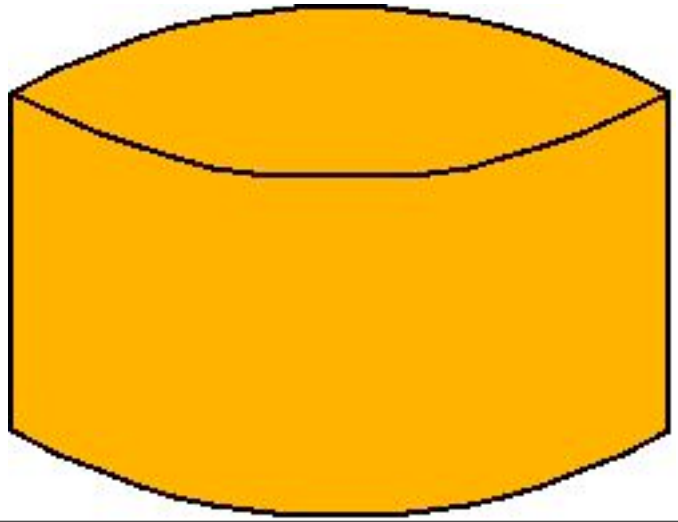
All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

Naive-Snapshot

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

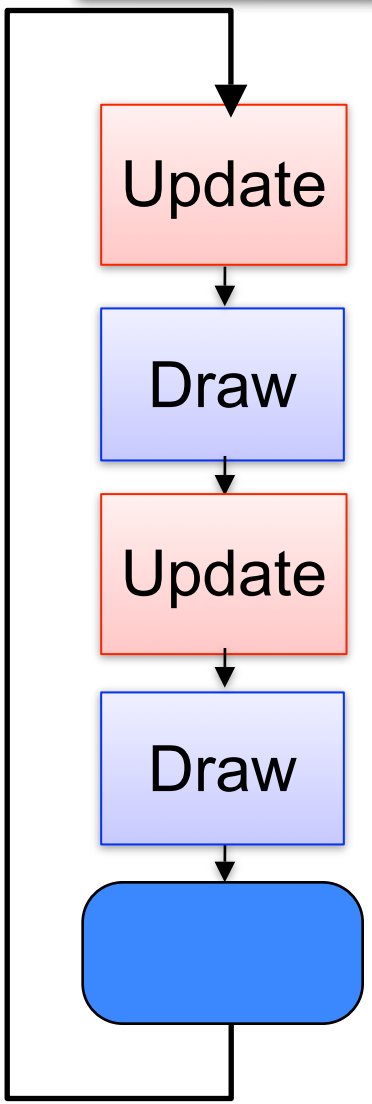


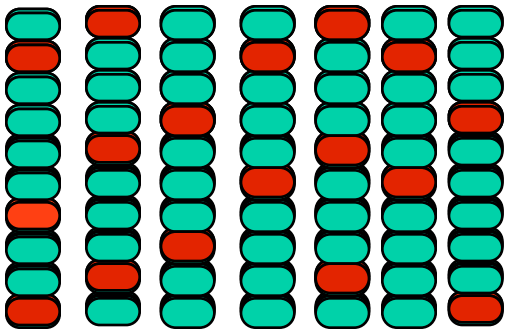


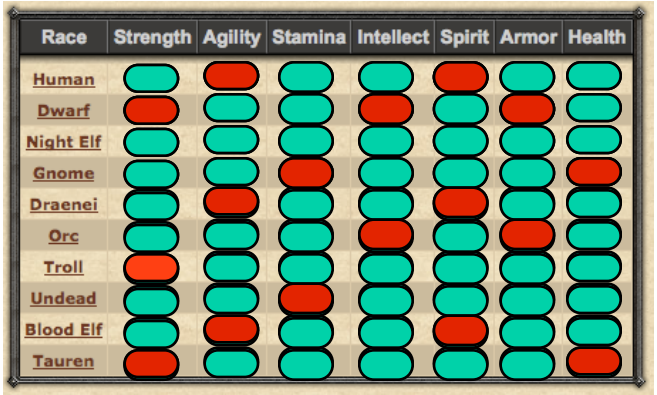


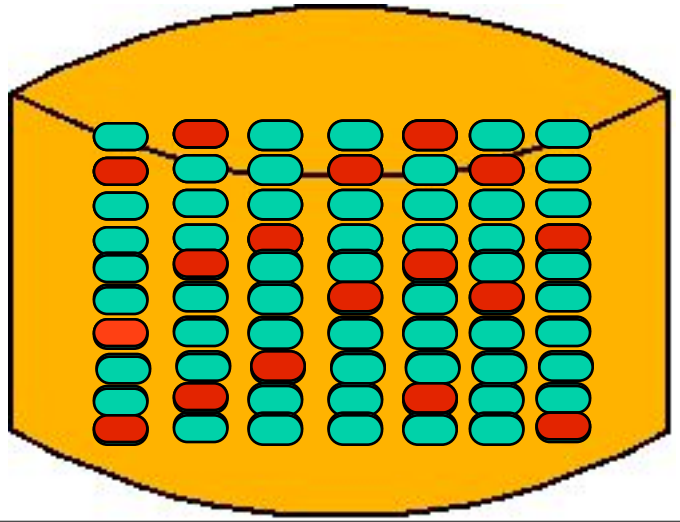
Naive-Snapshot

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU



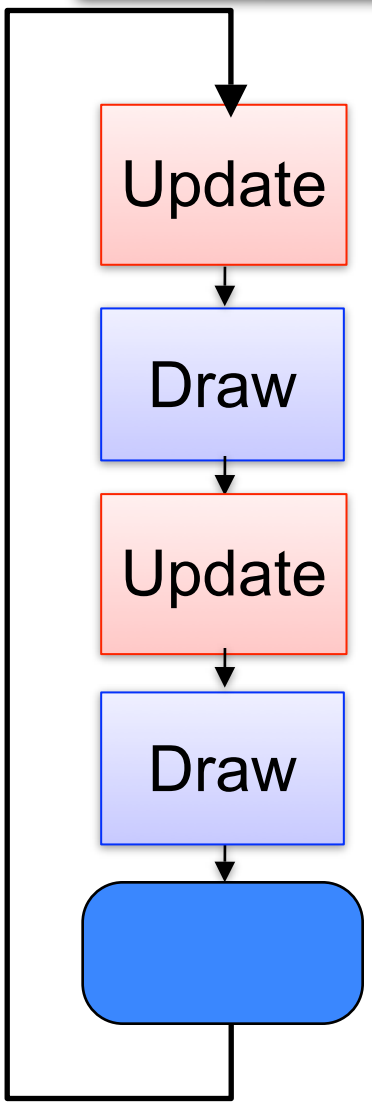


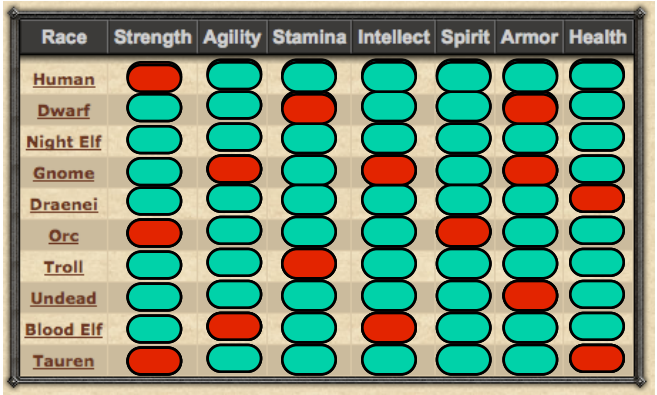


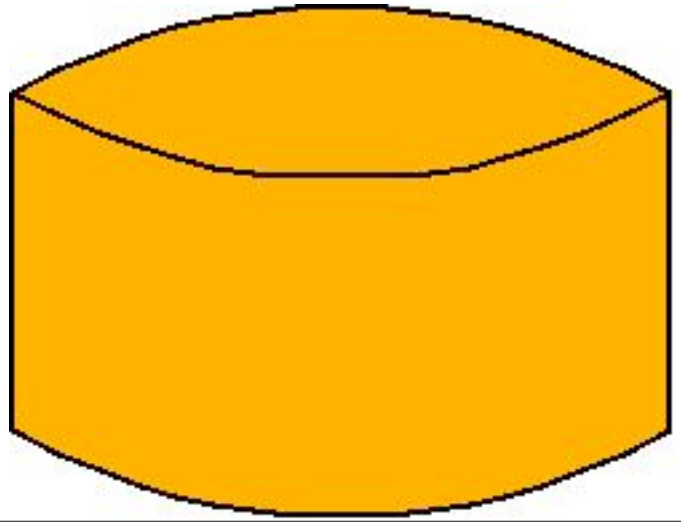


Partial-Redo

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

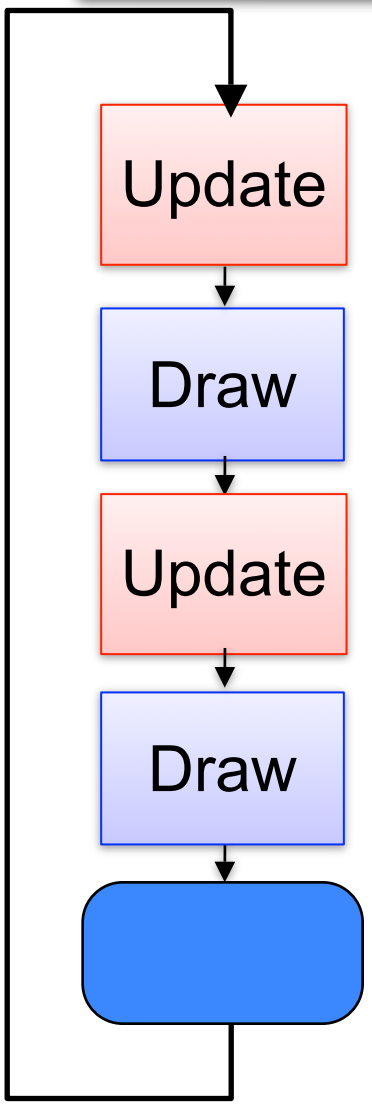


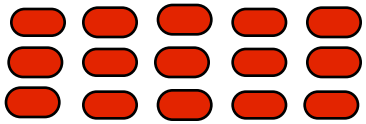


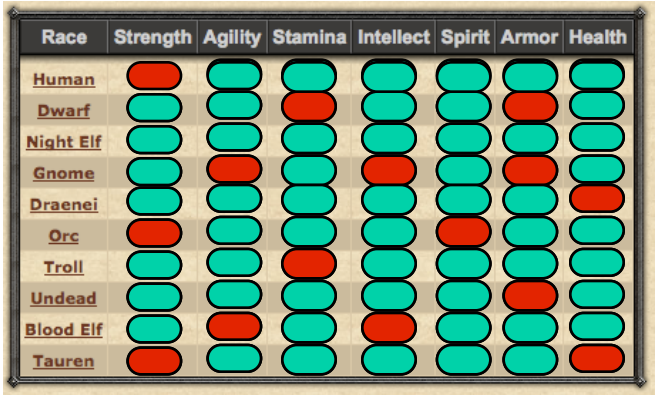


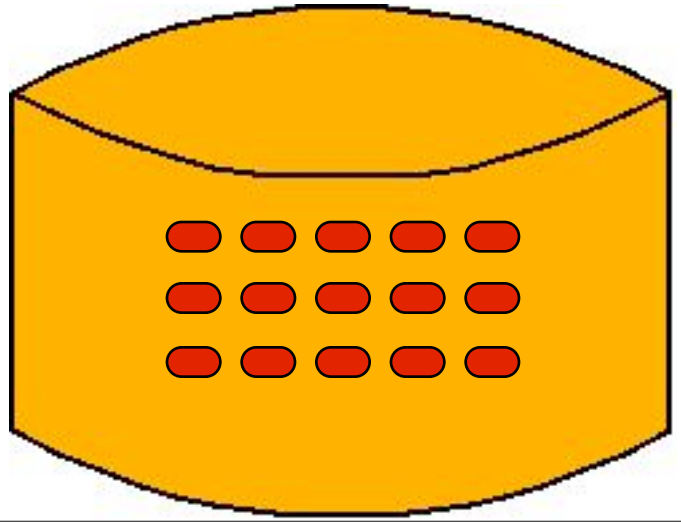
Partial-Redo

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU



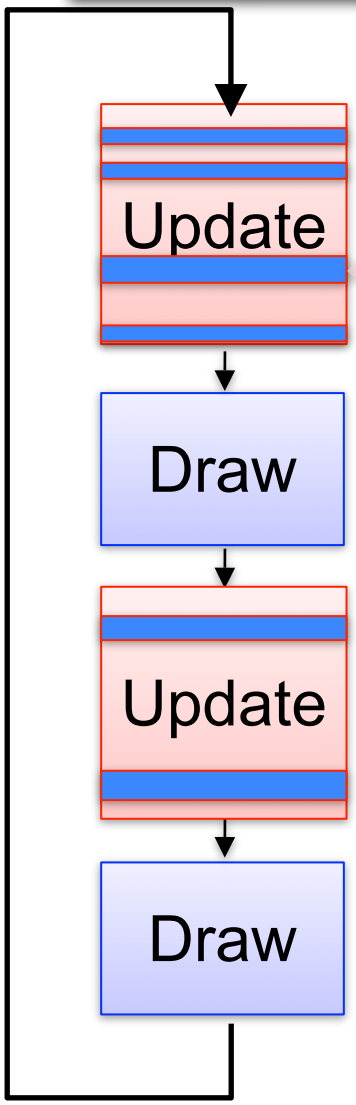




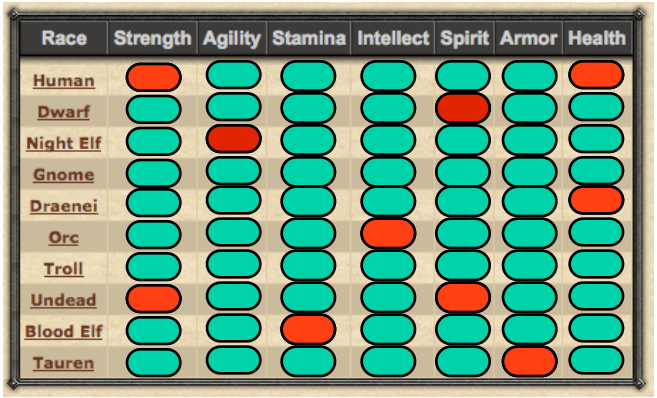


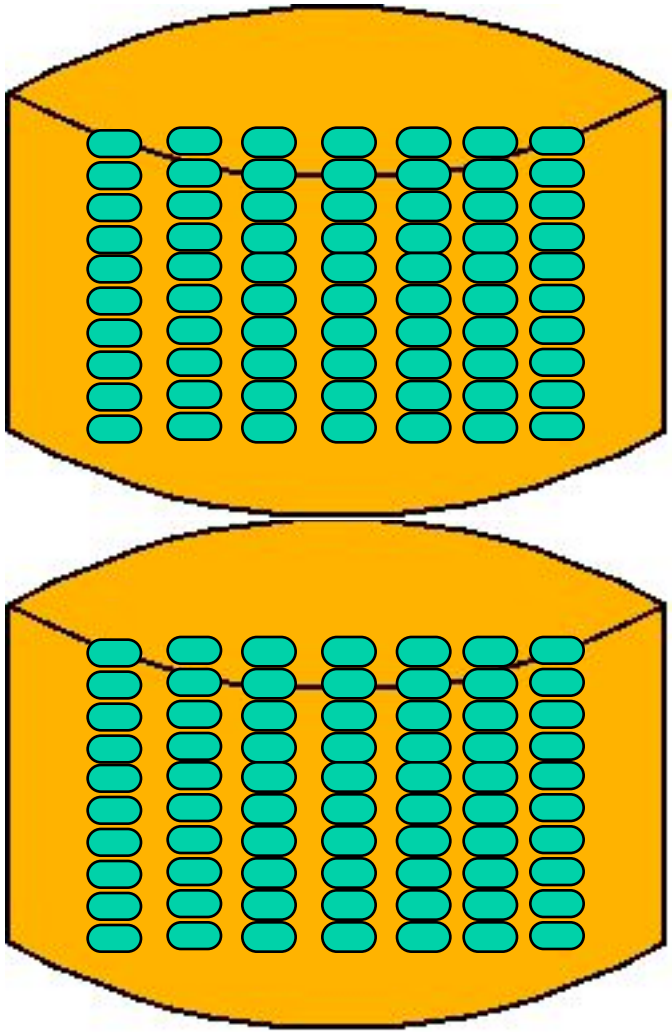
Copy-On-Update

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU



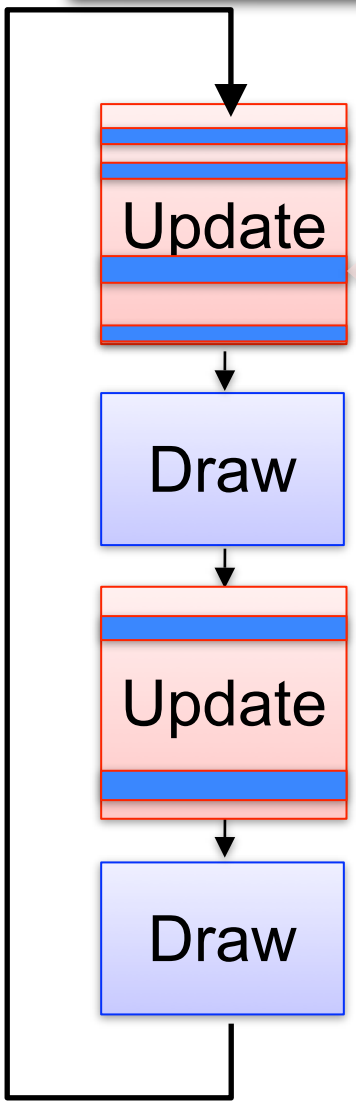
Copy object to main memory before updating it



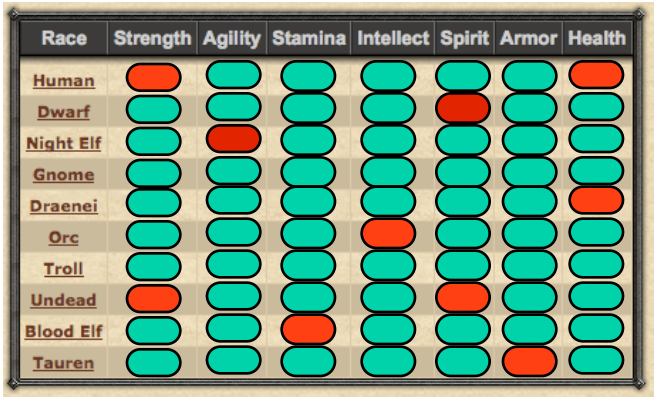


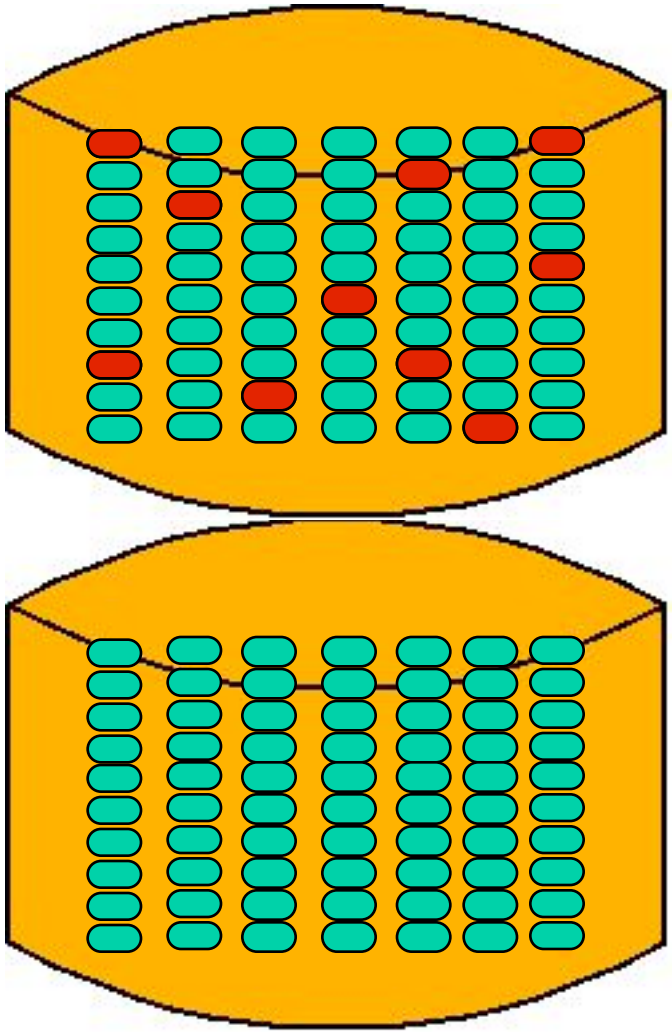
Copy-On-Update

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU



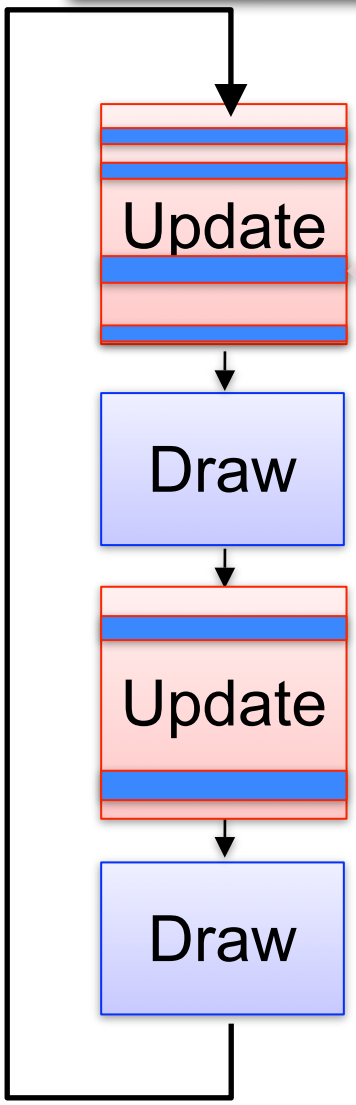
Copy object to main memory before updating it



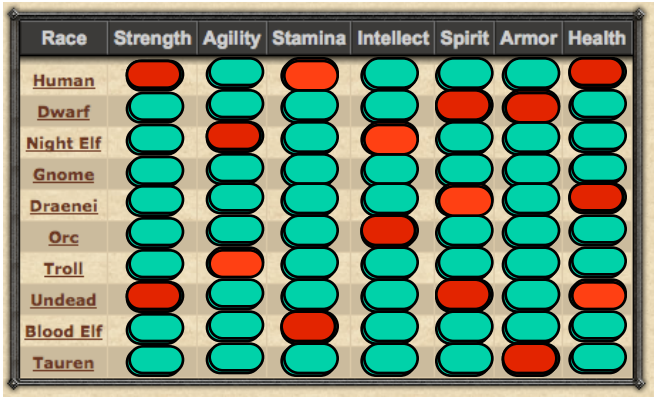


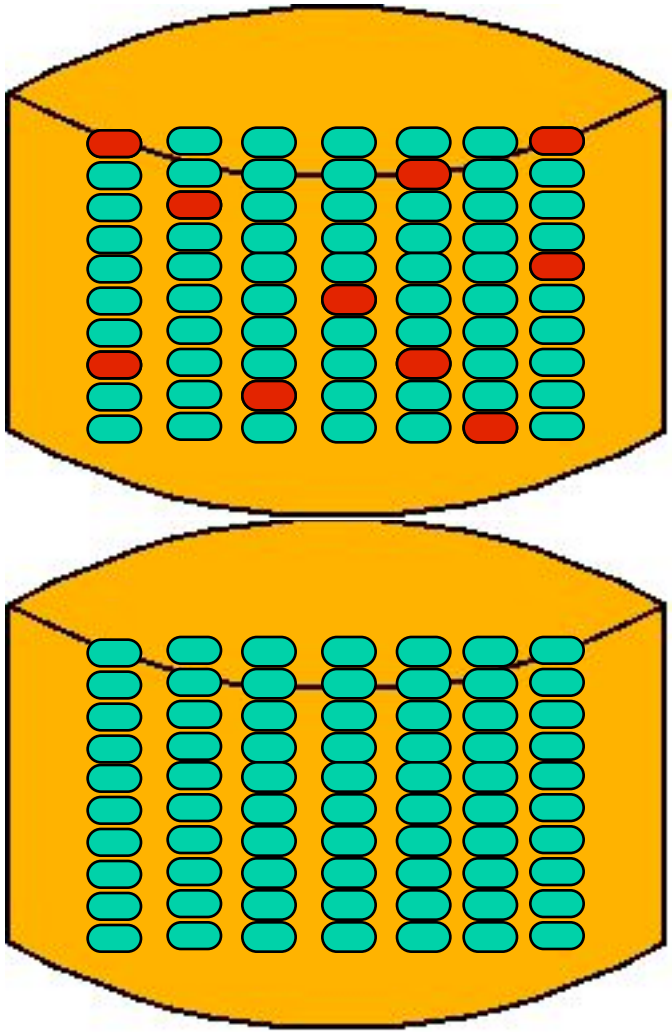
Copy-On-Update

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU



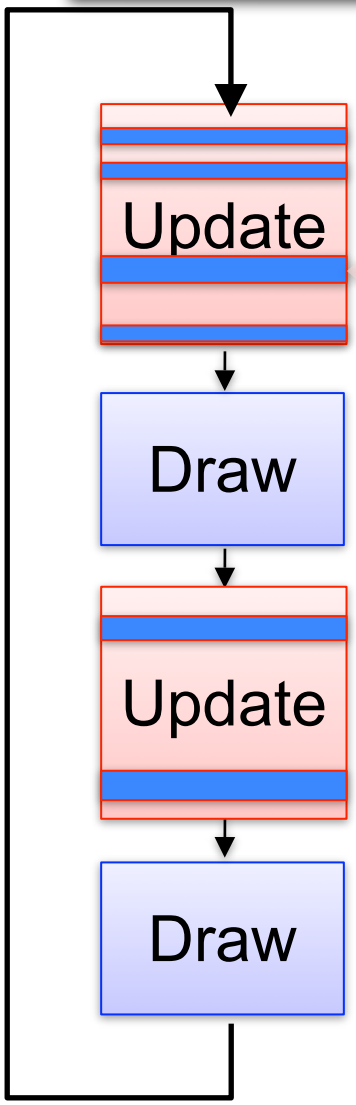
Copy object to main memory before updating it



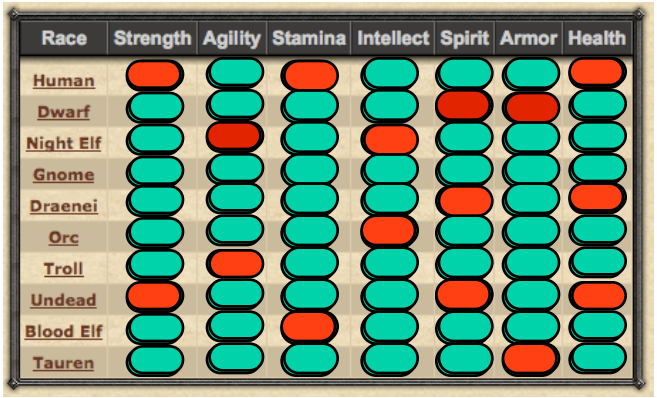


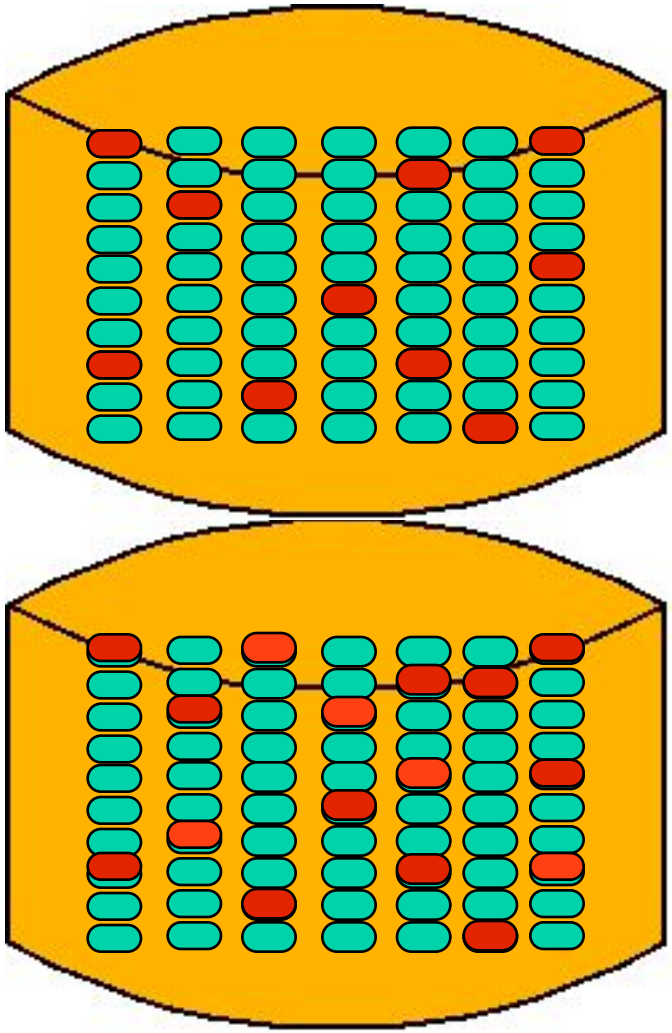
Copy-On-Update

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU



Copy object to main memory before updating it





Performance Model Components

- In main-memory copy time
- Disk write time
- Update handler overhead
- Recovery Time

Performance Model Components

- In main-memory copy time

$$\Delta T_{memcopy}(k) = \frac{k \cdot S_{obj}}{B_{mem}}$$

- Disk write time

$$\Delta T_{disk\text{-}write}(k) = \frac{k \cdot S_{obj}}{B_{disk}}$$

- Update handler overhead

$$\Delta T_{overhead} = O_{bit} + O_{lock} + \Delta T_{memcopy}$$

- Recovery Time

$$\Delta T_{recovery} = \Delta T_{restore} + \Delta T_{replay}$$

$$\Delta T_{restore} = \frac{(k \cdot C + n) \cdot S_{obj}}{B_{disk}}$$

(Partial-Redos)

$$\Delta T_{restore} = \frac{n \cdot S_{obj}}{B_{disk}}$$

(The other algorithms)

Update Handler Overhead

$$\Delta T_{overhead} = O_{bit} + O_{lock} + \Delta T_{memcopy}$$

O_{bit}	Overhead to test a dirty bit
O_{lock}	The cost to lock an object
$\Delta T_{memcopy}$	Cost of a memory copy of an atomic object

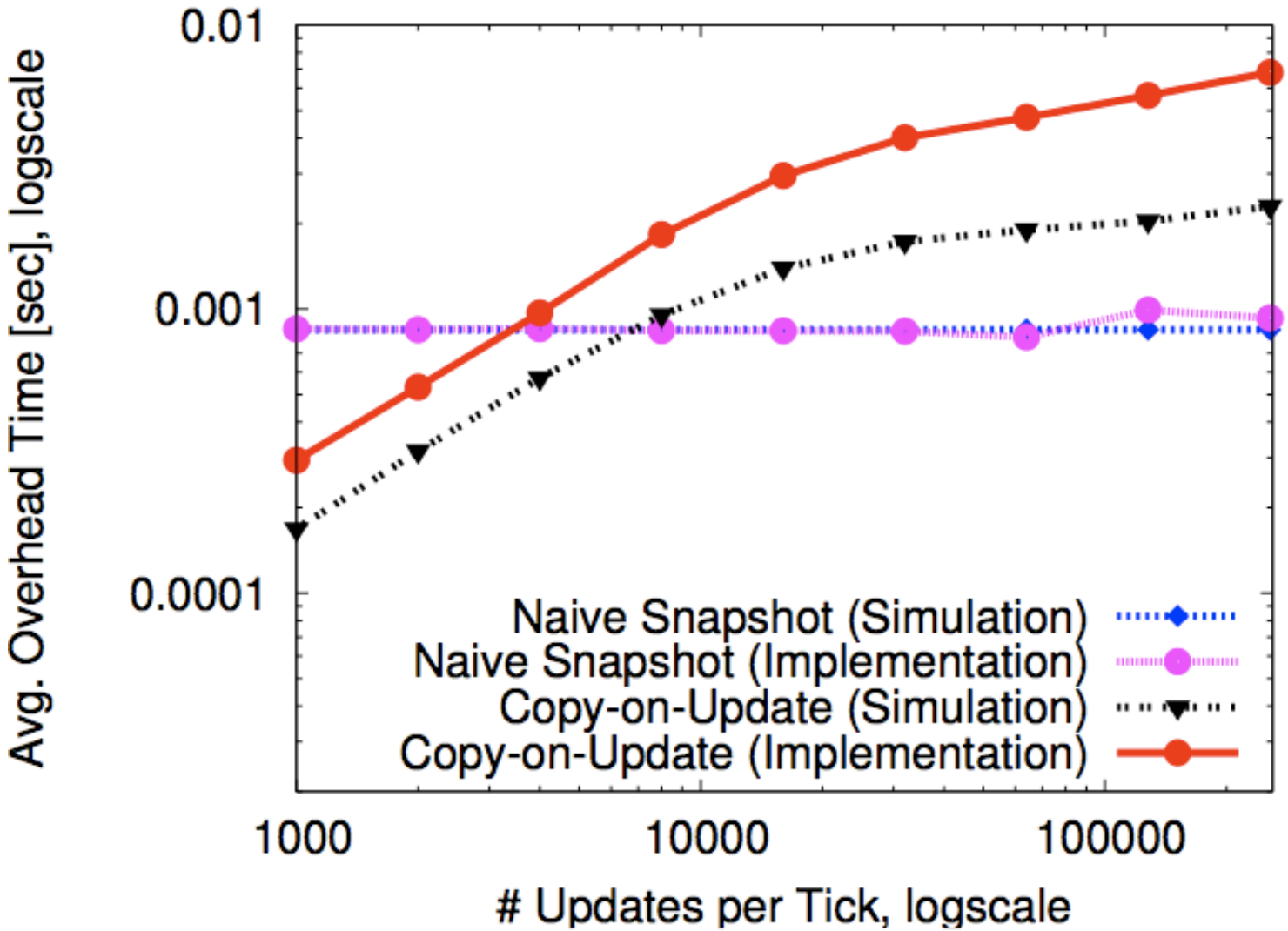
$$\Delta T_{memcopy} = \frac{S_{obj}}{B_{mem}}$$

S_{obj}	Size of an object
B_{mem}	Memory bandwidth

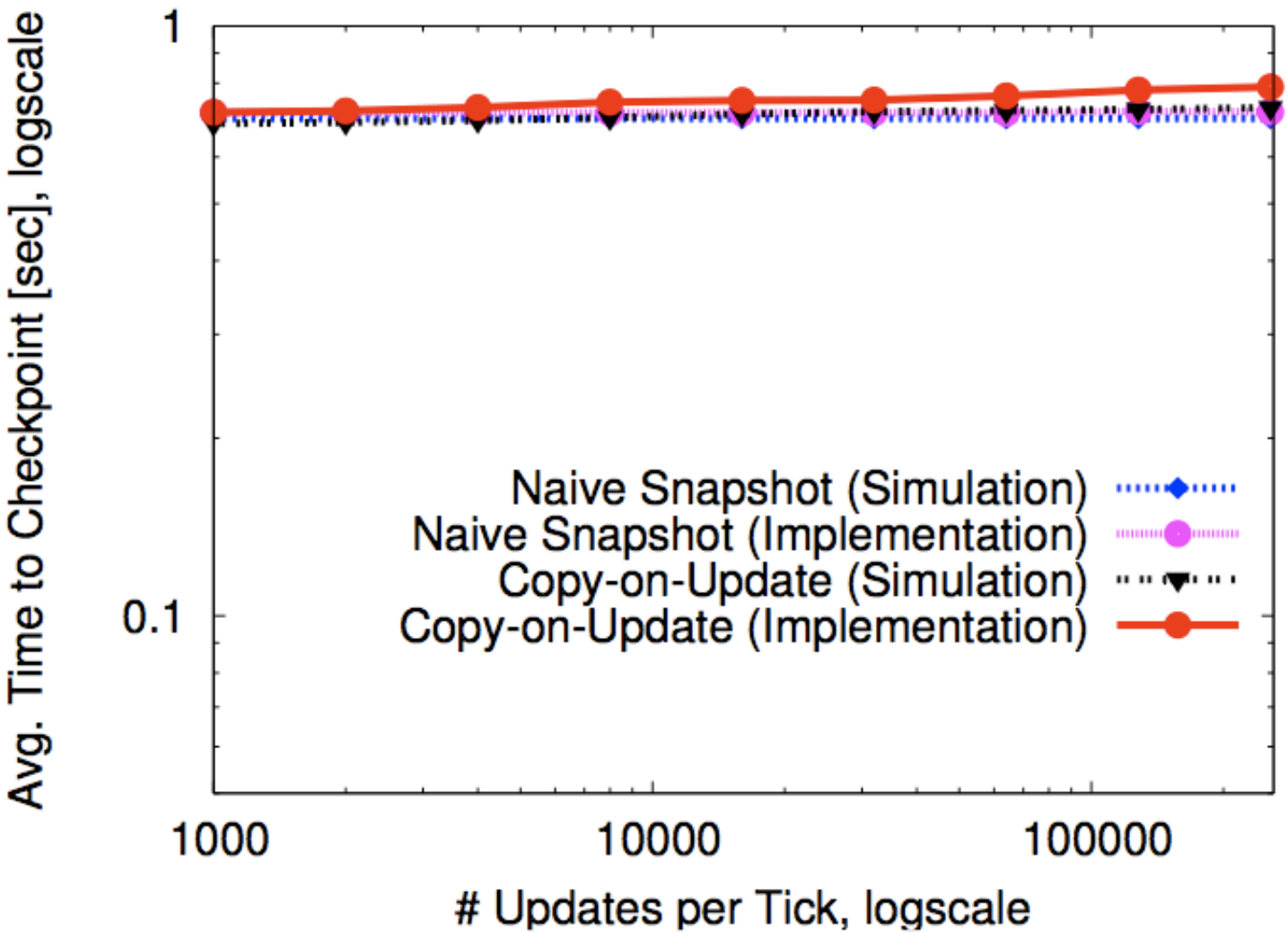
Experimental Validation

- Implemented Naive-Snapshot and Copy-On-Update in C++
- Running on our high-end server
  - Intel Core 2 Quad 2.4 Ghz
  - 4MB cache
  - 4GB RAM
  - Ubuntu (kernel 2.6.27-14)
  - 80GB 7200rpm SATA drive with 8MB cache
  - Game ticks operate at 30Hz

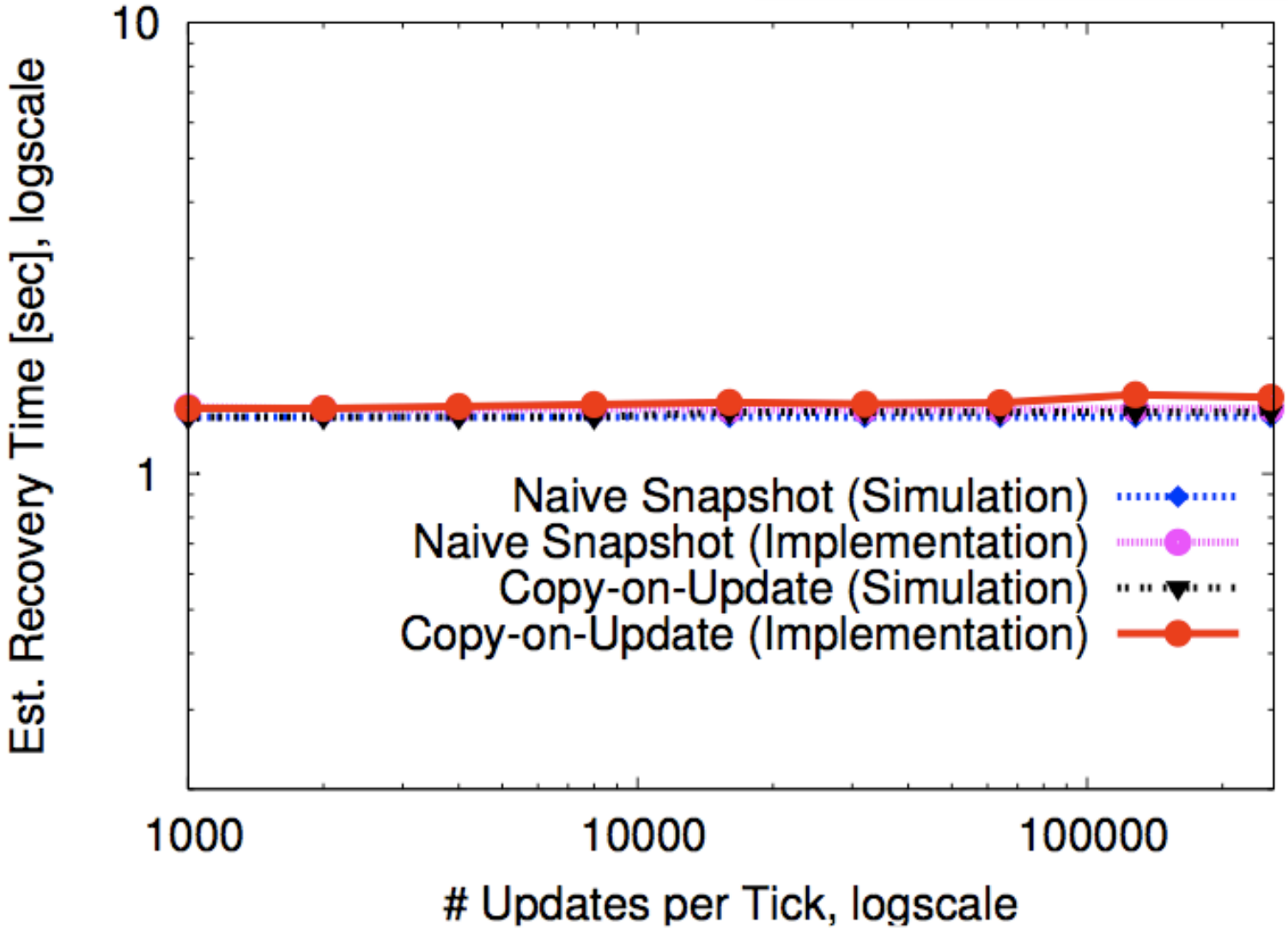
Overhead Time



Checkpointing Time



Recovery Time



Experiments

- Scaling with update rates
- Prototype game

Data Set 1: Synthetic Trace

- Generate updates according to a Zipf distribution
- Vary the number of updates per tick from 1k to 256k updates per tick

Data Set 2: Prototype Game Trace

- We simulated a medieval battle of the type common in many MMOs
- Knights, archers and healers, divided into two teams
- The objective is to defeat as many enemies as possible

Refer to http://www.cs.cornell.edu/~wmwhite/papers/2007-SIGMOD-Games.pdf

Simulation Parameters

Memory Bandwidth	B_{mem}
Memory Latency	O_{mem}
Lock Overhead	O_{lock}
Bit test/set Overhead	O_{bit}
Disk Bandwidth	B_{disk}

From our micro-benchmarks running on

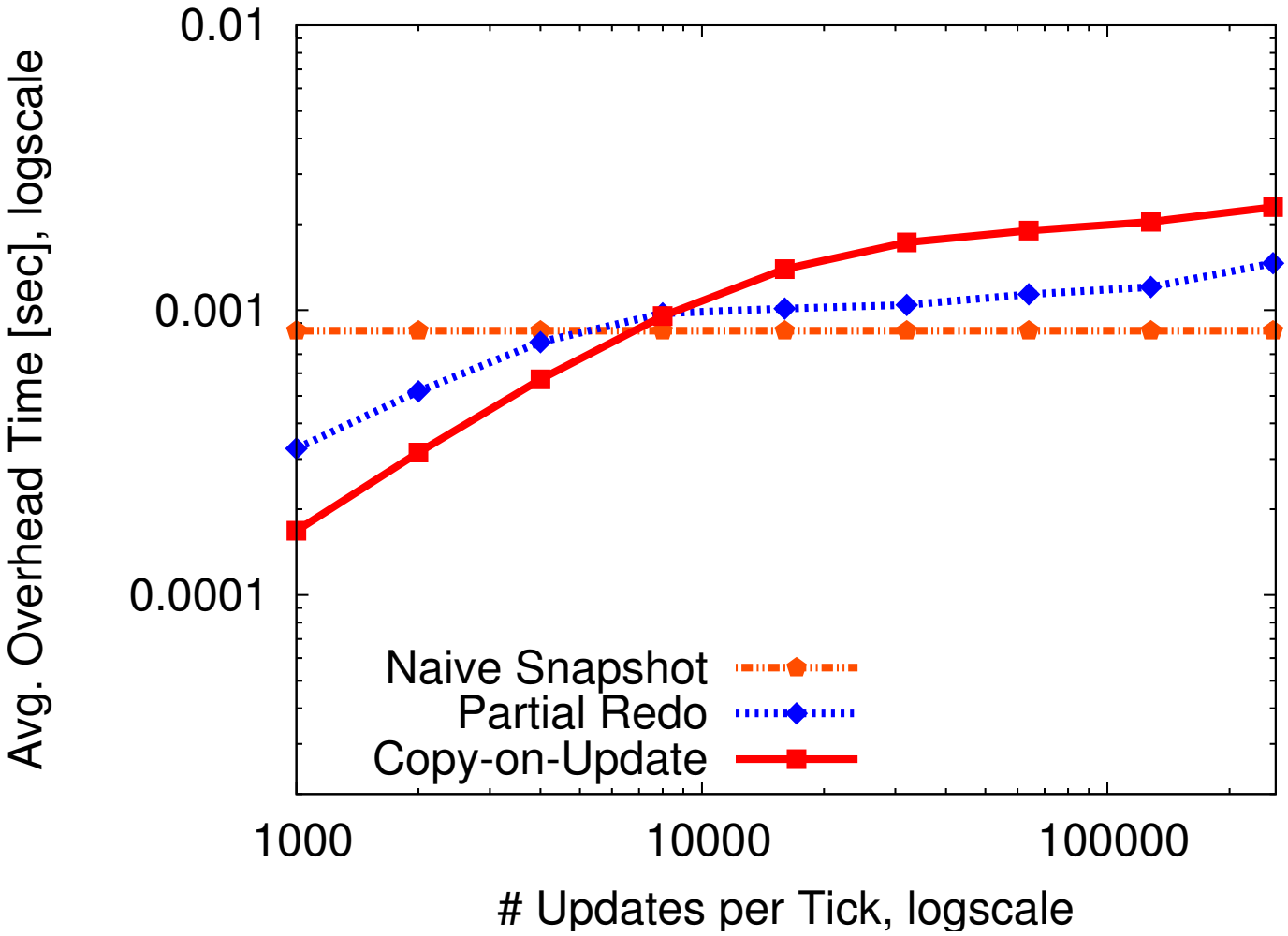
- Intel Core 2 Quad 2.4 Ghz
- 4MB cache
- 4GB RAM
- Ubuntu (kernel 2.6.27-14)
- 80GB 7200rpm SATA drive with 8MB cache

Performance Criteria

- Average Overhead Time
- Overhead Distribution
- Recovery Time
- Checkpointing Time

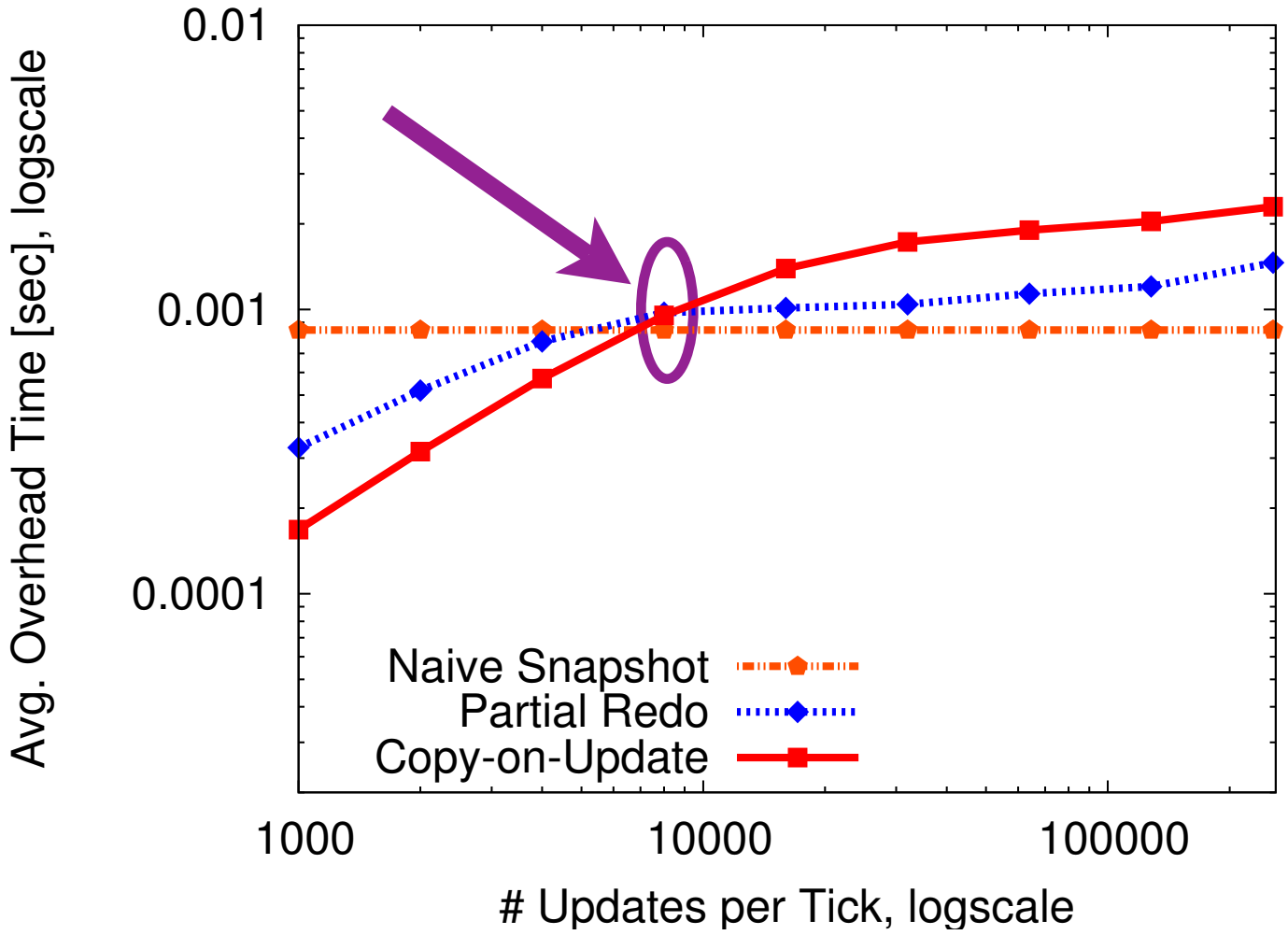
Scaling on Number Of Updates Per Tick

(Overhead Time)

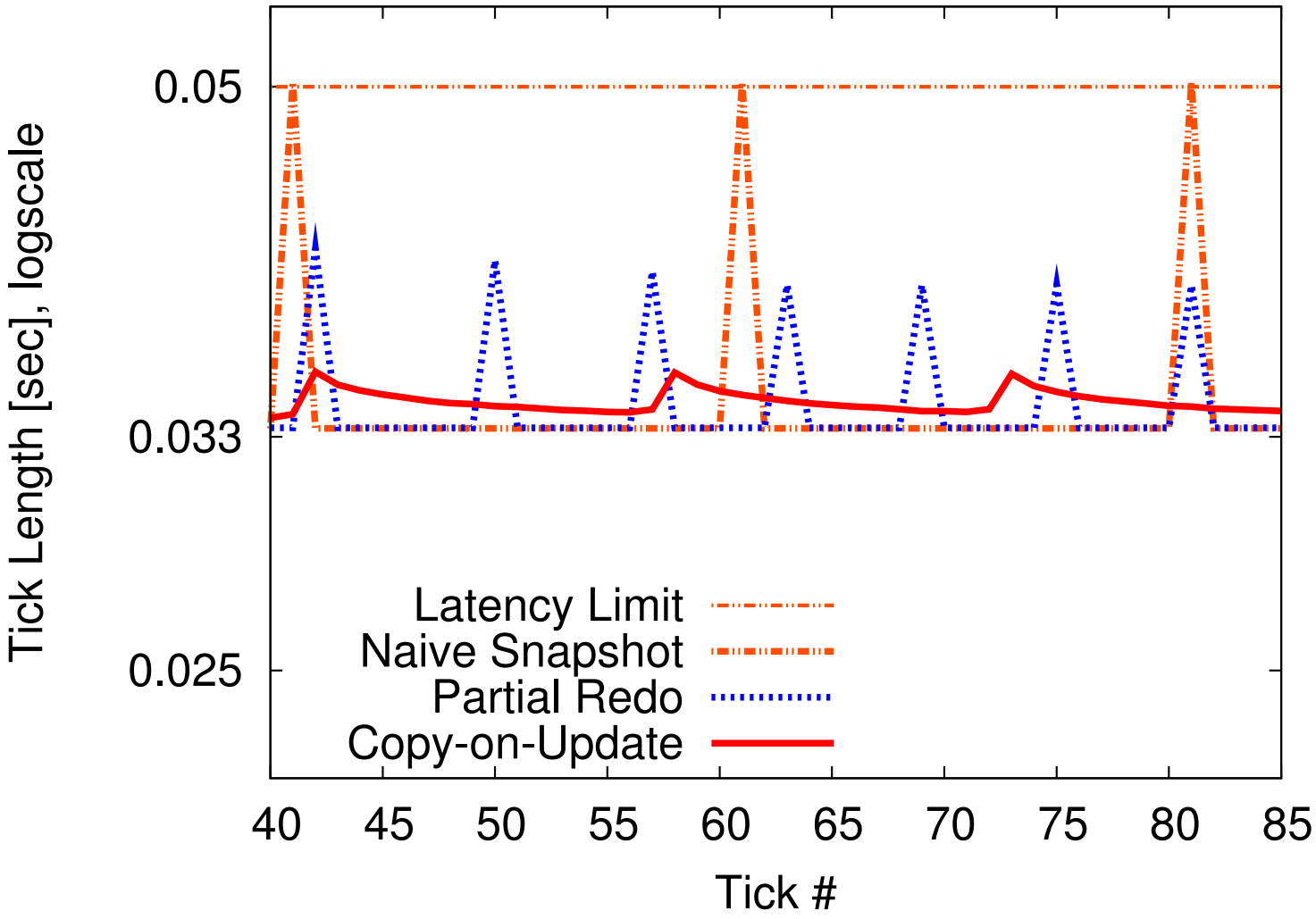


Scaling on Number Of Updates Per Tick

(Overhead Time)

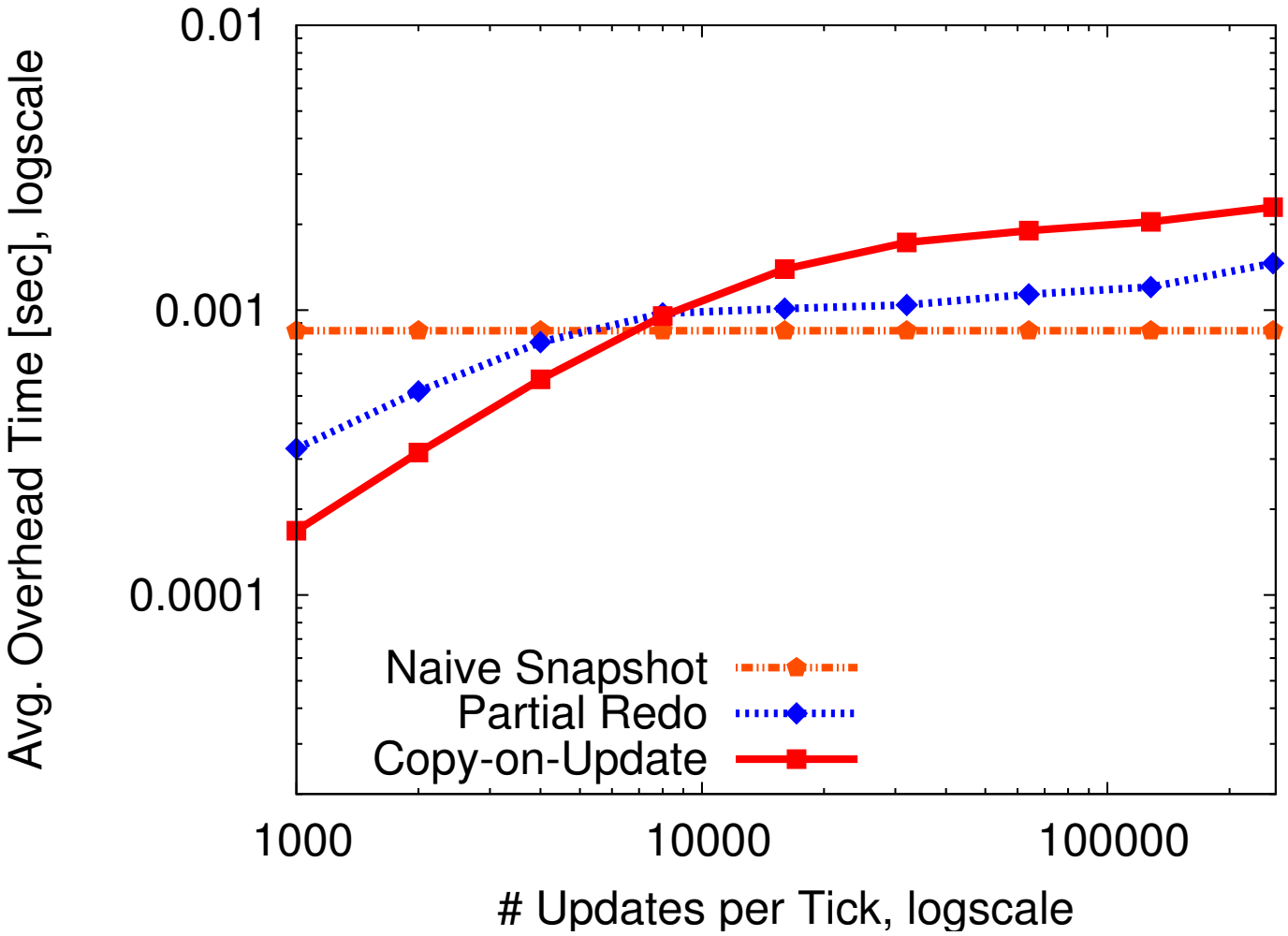


Latency Analysis
(8k updates per tick)



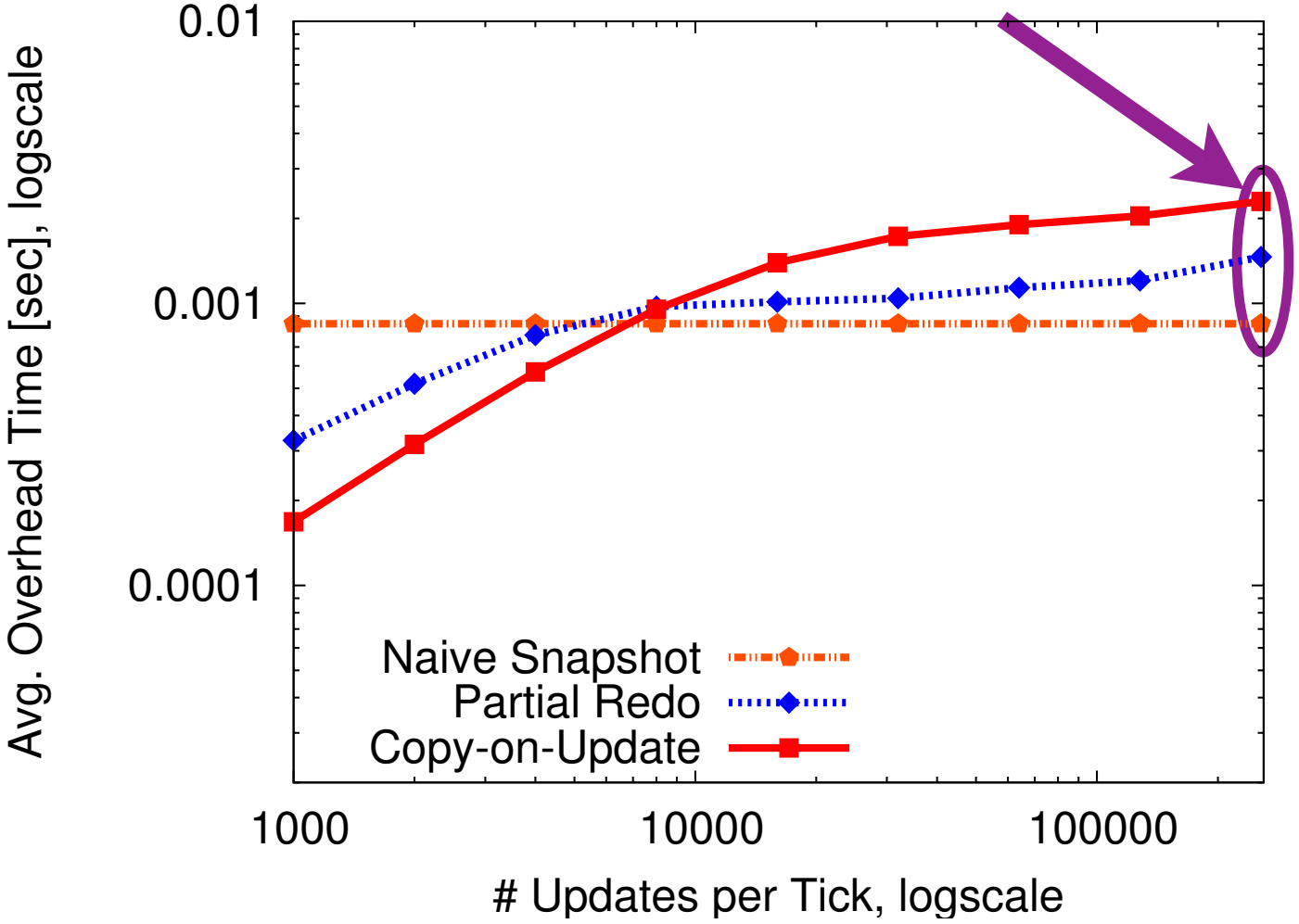
Scaling on Number Of Updates Per Tick

(Overhead Time)

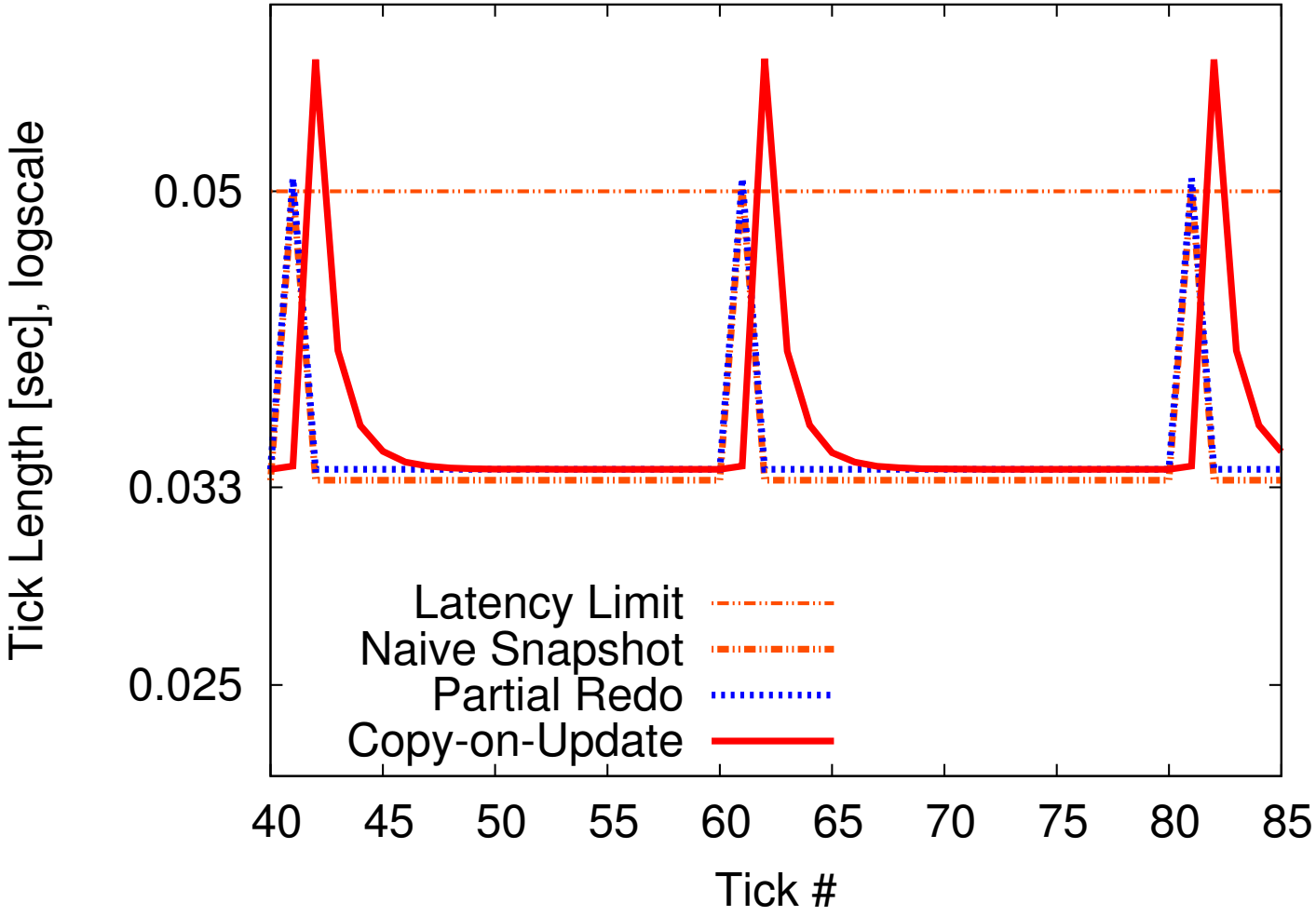


Scaling on Number Of Updates Per Tick

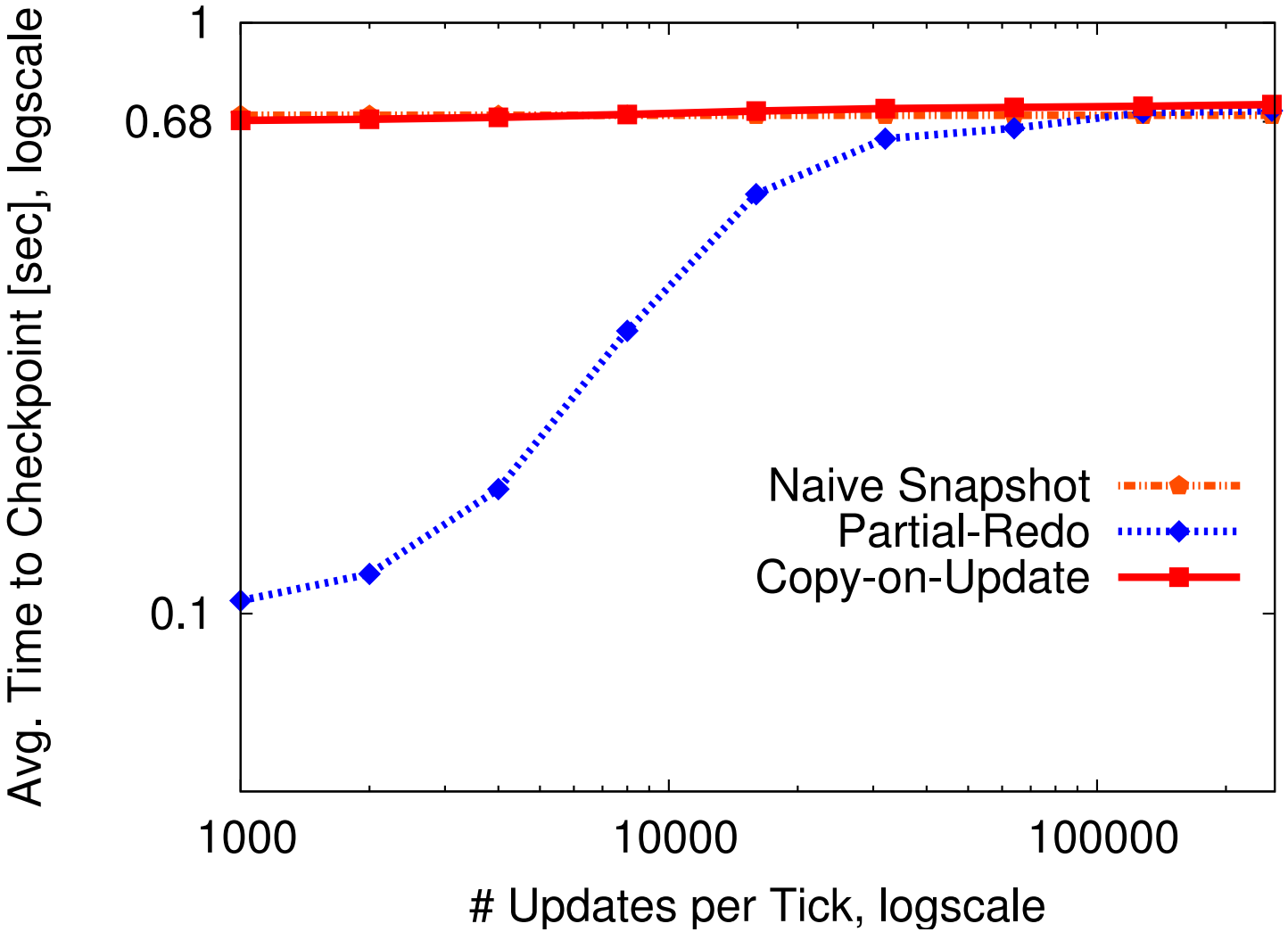
(Overhead Time)



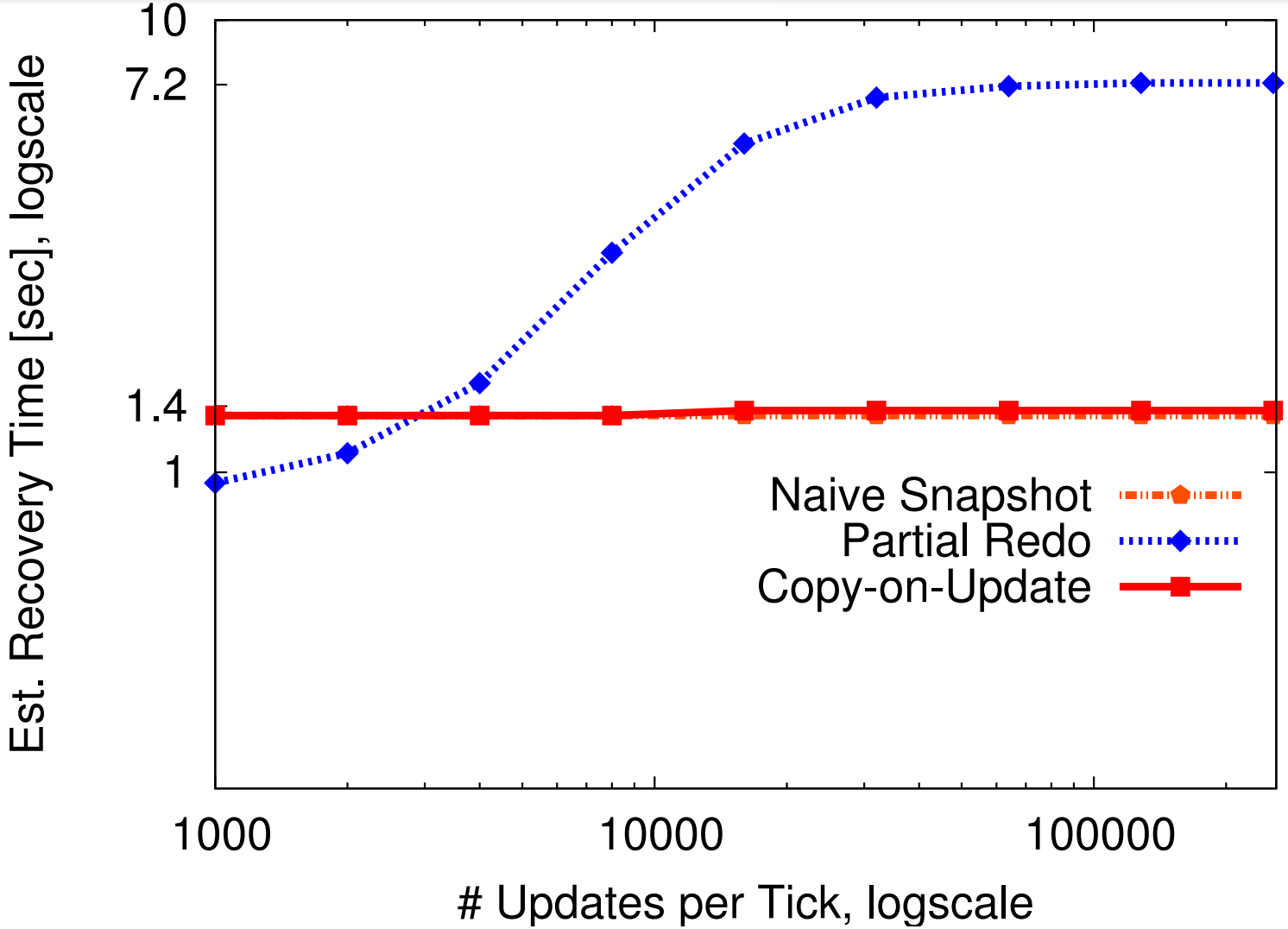
Latency Analysis
(256k updates per tick)



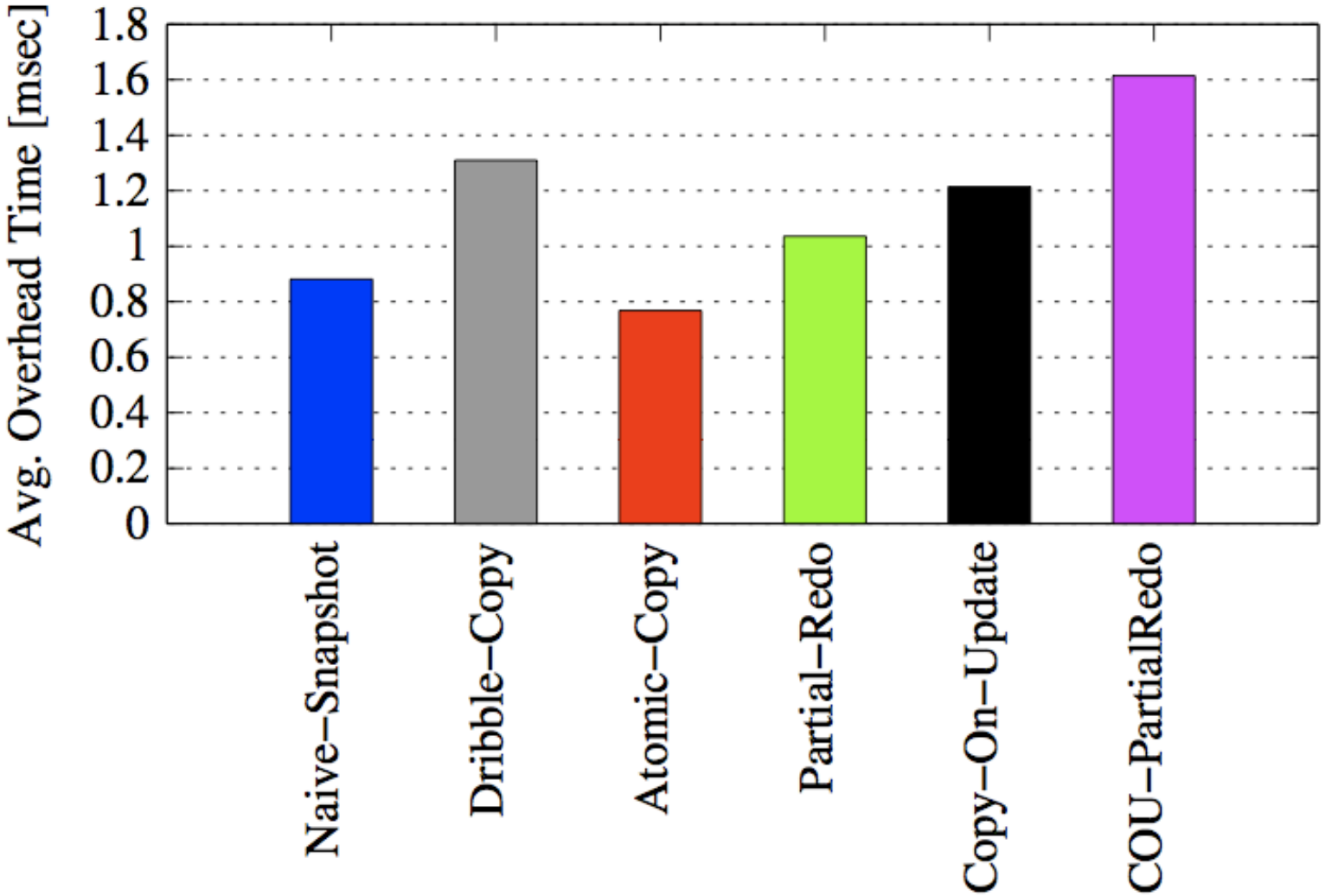
Scaling on Number Of Updates Per Tick (Checkpointing Time)



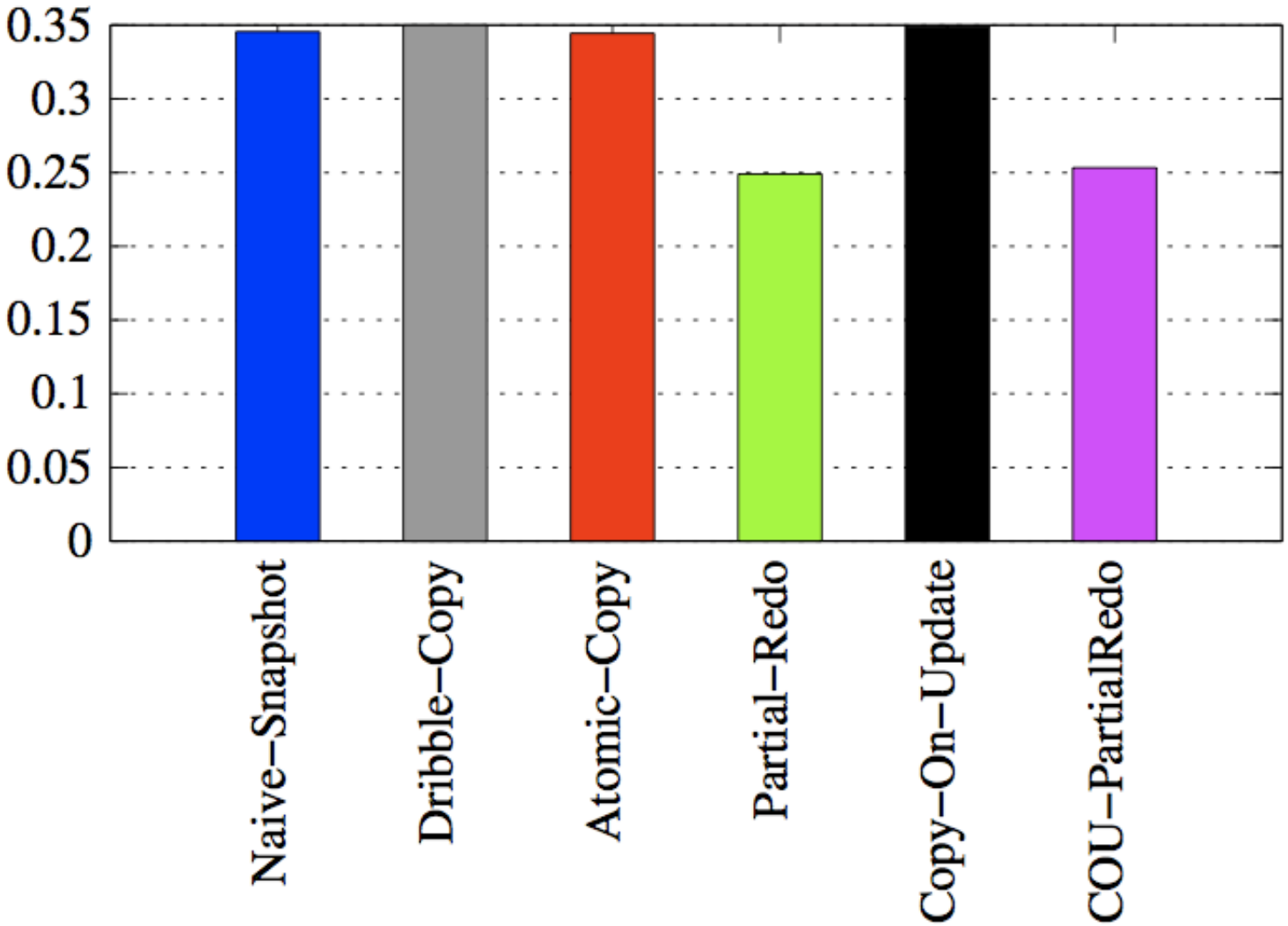
Scaling on Number Of Updates Per Tick (Recovery Time)



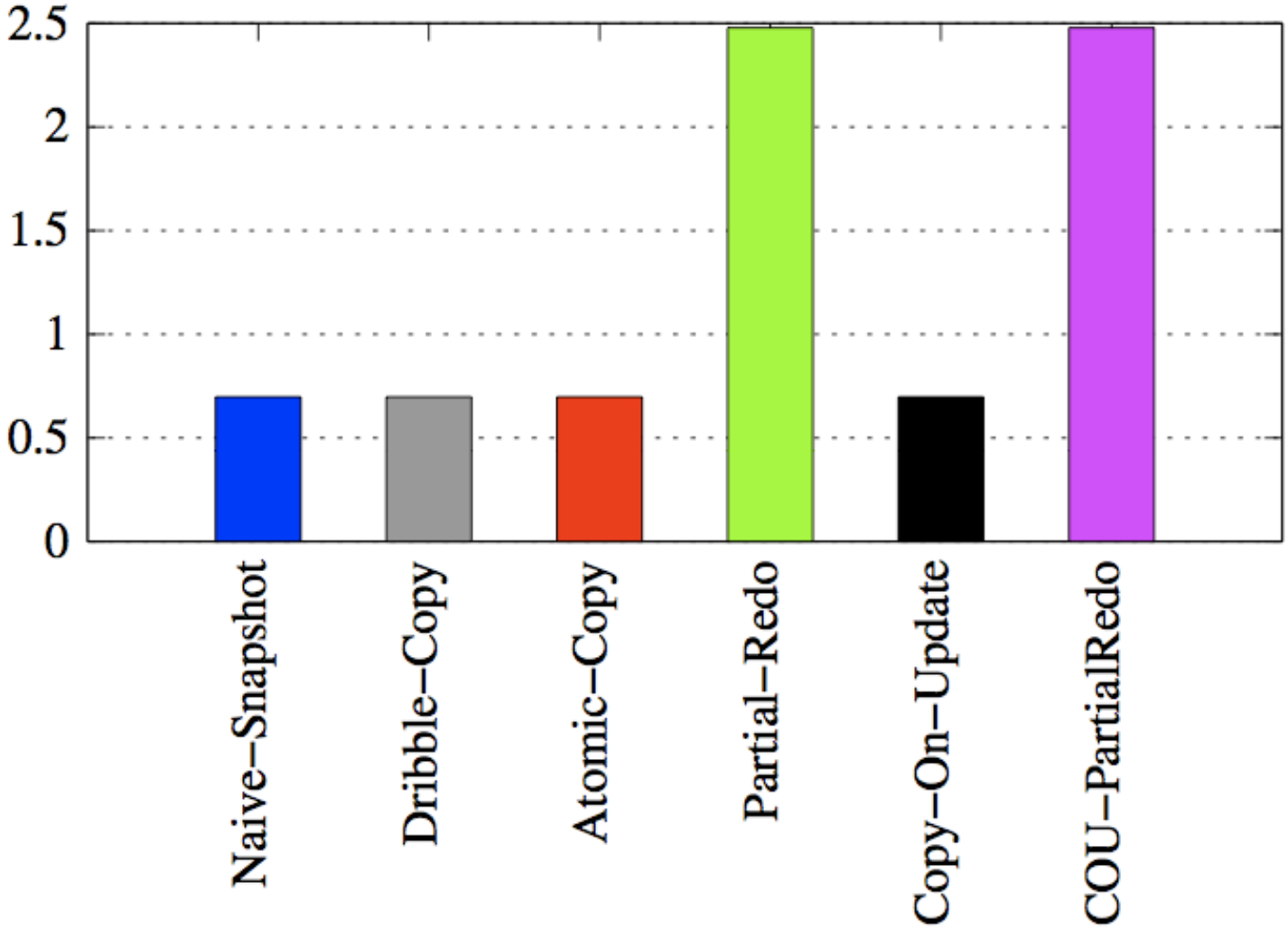
Experiments with Prototype Game Server (Overhead Time)



Experiments with Prototype Game Server (Checkpointing Time)



Experiments with Prototype Game Server (Recovery Time)



Conclusion

- Methods based on double backup recover much faster than methods checkpointing to logs
- For moderate update rates, the best method in terms of both latency and recovery time is Copy-On-Update
- In case of extremely high update rates, we recommend Naive-Snapshot
- Available to download at :

  http://www.cs.cornell.edu/bigreddata/games

References

K. Salem and H. Garcia-Molina.
"Checkpointing Memory-Resident Databases". In Proc. ICDE, 1989.

W. White, A. Demers, C. Koch, J. Gehrke, and R. Rajagopalan.
"Scaling Games to Epic Proportions". In Proc. SIGMOD, 2007.

G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali, and P. Stodghill.
"Recent Advances in Checkpoint/Recovery Systems". In Proc. IPDPS, 2006.

D. J. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker, and D. Wood.
"Implementation Techniques for Main Memory DatabaseSystems". In Proc. SIGMOD, 1984.

D. Rosenkrantz.
"Dynamic Database Dumping". In Proc. SIGMOD, 1978.

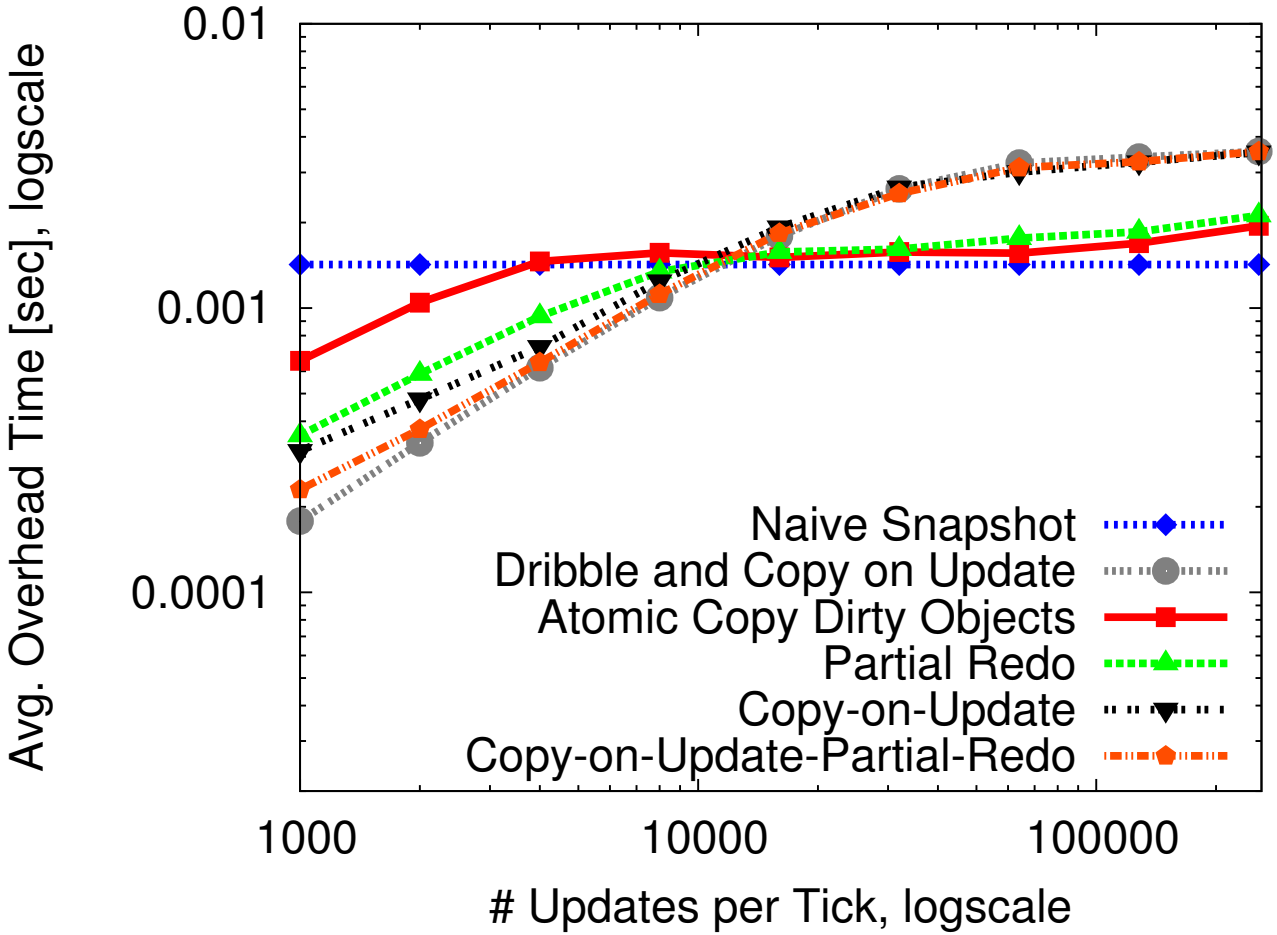
Richard Fujimoto.
"Parallel and Distributed Simulation Systems". John Wiley & Sons, 2000.

Conclusion

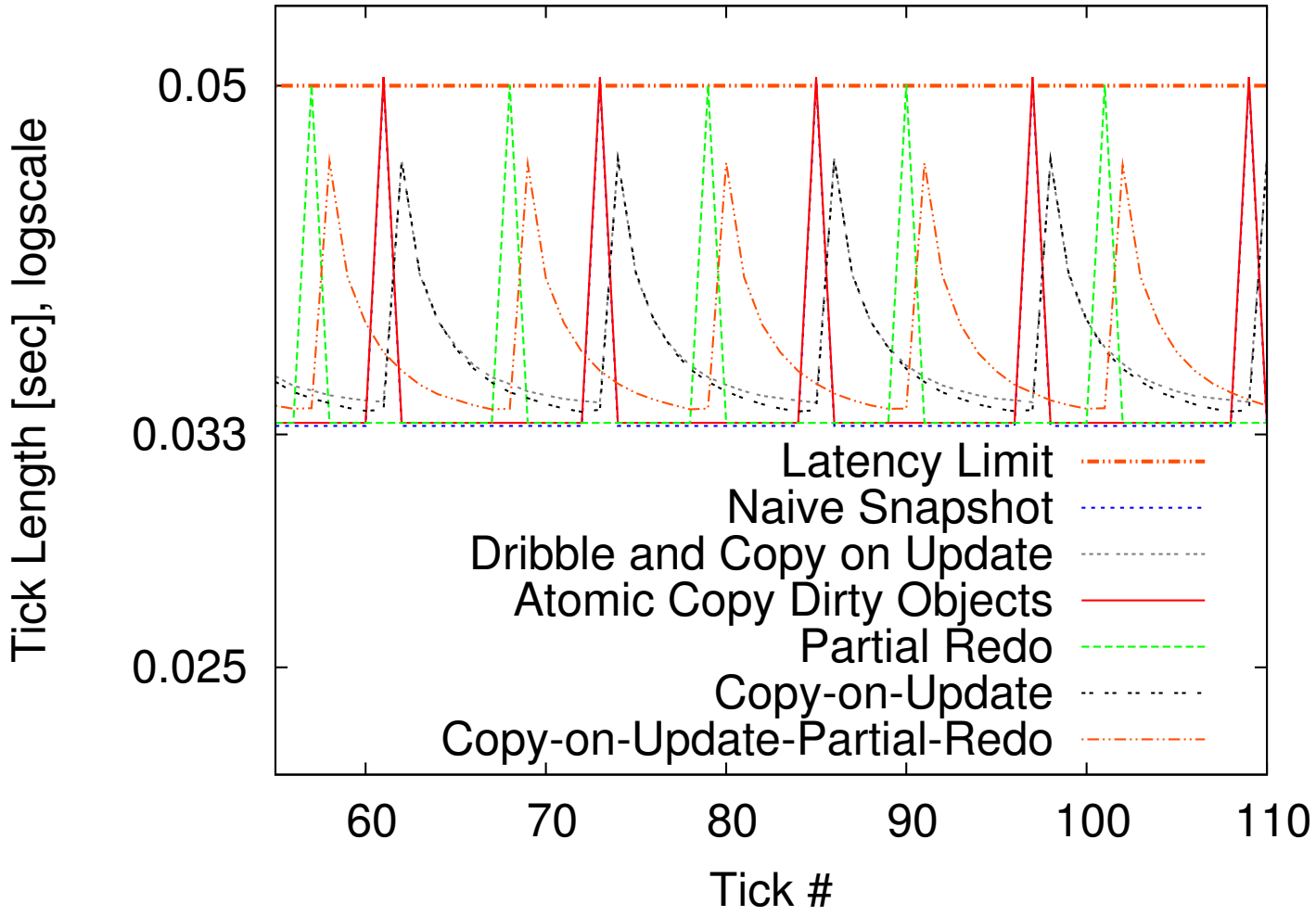
- Methods based on double backup recover much faster than methods checkpointing to logs
- For moderate update rates, the best method in terms of both latency and recovery time is Copy-On-Update
- In case of extremely high update rates, we recommend Naive-Snapshot
- Available to download at :

  http://www.cs.cornell.edu/bigreddata/games

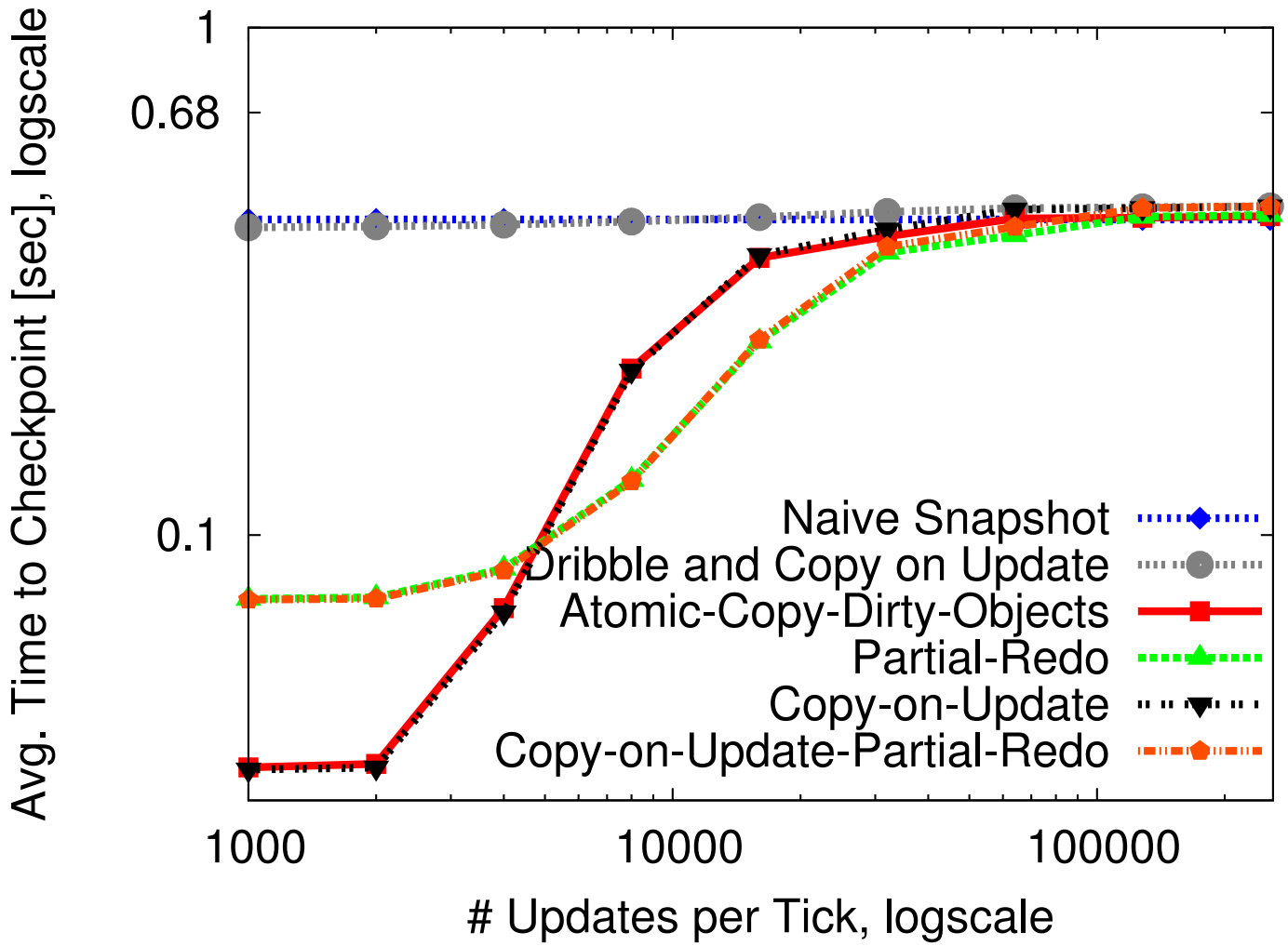
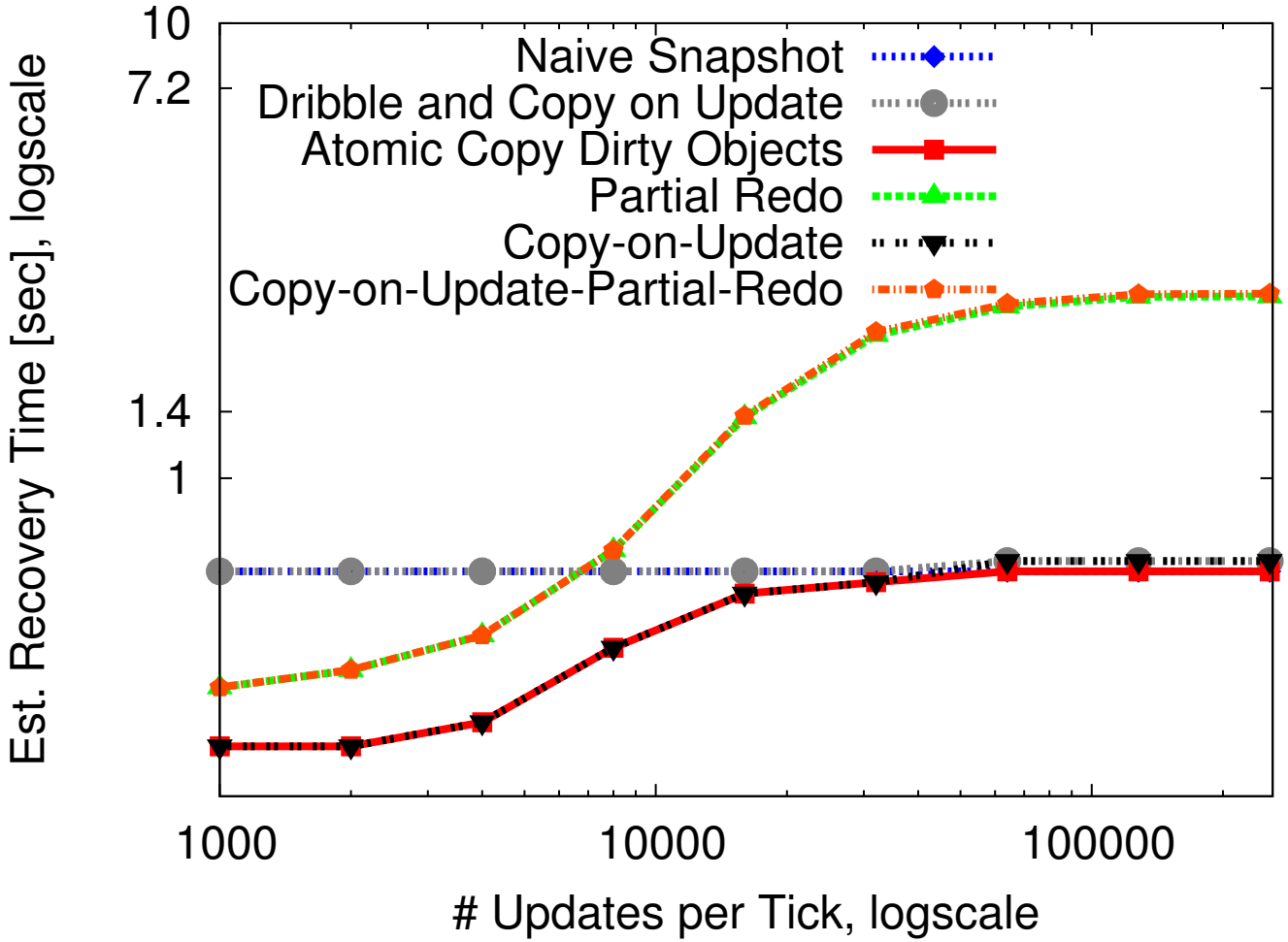
SSD - Scaling on Number Of Updates Per Tick
(Overhead Time)



SSD - Latency Analysis
(64k updates per tick)



SSD - Scaling on Number Of Updates Per Tick
(Checkpointing Time)



SSD - Scaling on Number Of Updates Per Tick
(Recovery Time)