

Fast Databases with Fast Durability and Recovery Through Multicore Parallelism

Wenting Zheng, Stephen Tu (MIT/UC Berkeley) Eddie Kohler
(Harvard), Barbara Liskov (MIT)

Motivation

- In-memory databases
 - popular
 - fast
 - **slow recovery**

Possible solutions

- Replication
 - Not a complete solution
- Traditional, logging and checkpointing based approaches
 - Can they keep up in today's environments with millions of transactions each second?

Logging

- Value/Transaction logging
- Naive logging is very slow
- Production databases can easily produce >50GB of log data each minute
- Logging to SSD disks is fast (sequential writes), but replaying them is slow
- Periodically require compressing into checkpoints
 - Causes data movement and pollutes caches

Silo

- A multi-core in-memory database
- Spawns as many threads as there are cores
- Pins threads to cores
- Stores tables as cache-friendly concurrent B-trees
 - Trees only keep the keys, data is kept in separately allocated records

SiloR

- Adds logger and checkpoint threads
 - Usually one per disk (and bound to it)
 - Used for writing to many disks simultaneously
- Uses value logging
 - Database can be reconstructed in parallel
 - Very I/O heavy

Serializing transactions

- Every transaction is assigned a TID (transaction id)
 - Transaction are grouped into epochs
- Every record maintains a TID of the last modifying transaction
- Uses Optimistic Concurrency Control
 - Compute whole transaction, rollback if any read/written resources have changed in the meantime

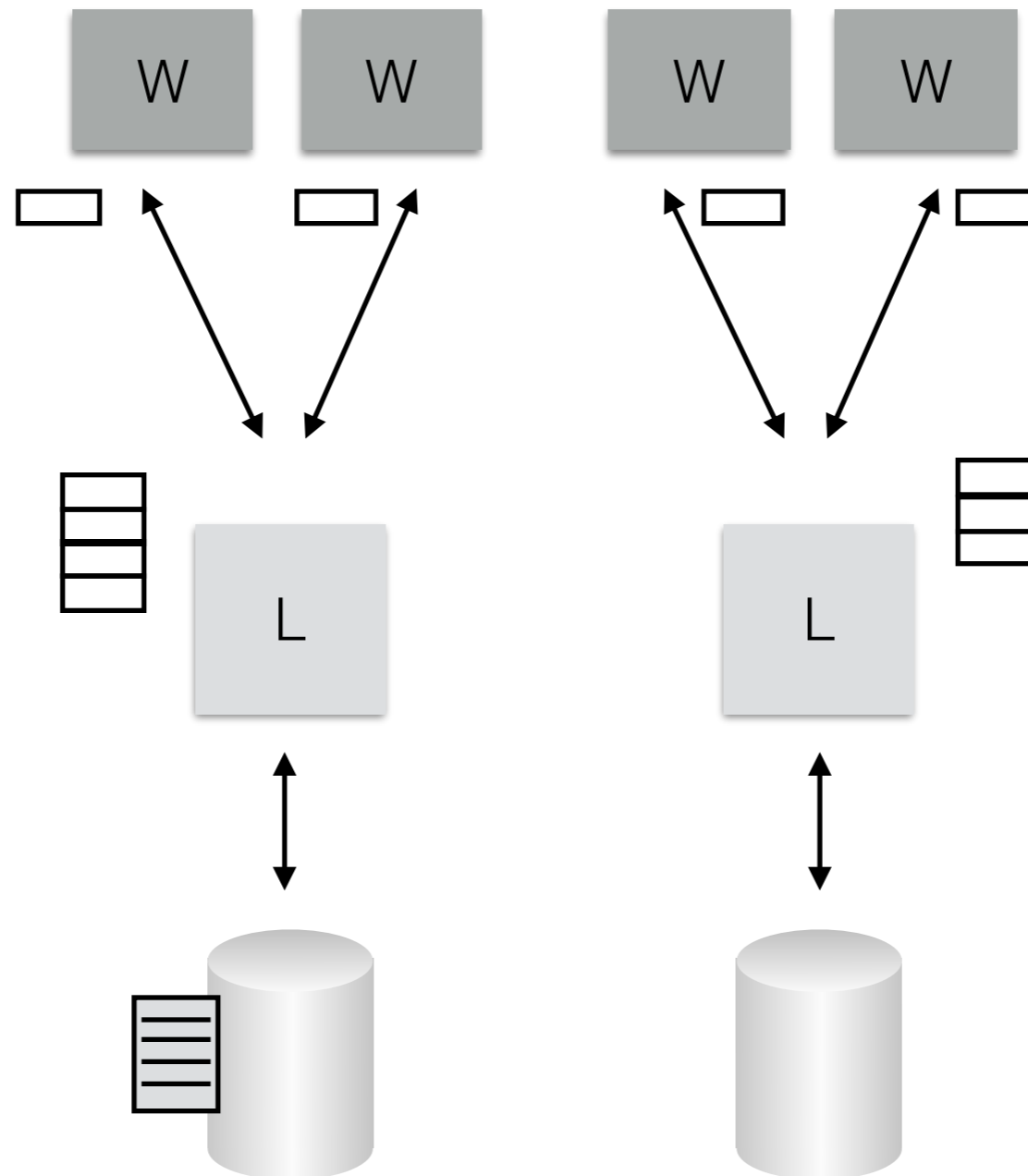
Computing TIDs

- A TID is:
 - greater than TID of any record used in the transaction
 - greater than the last TID committed by this worker
 - in the current epoch

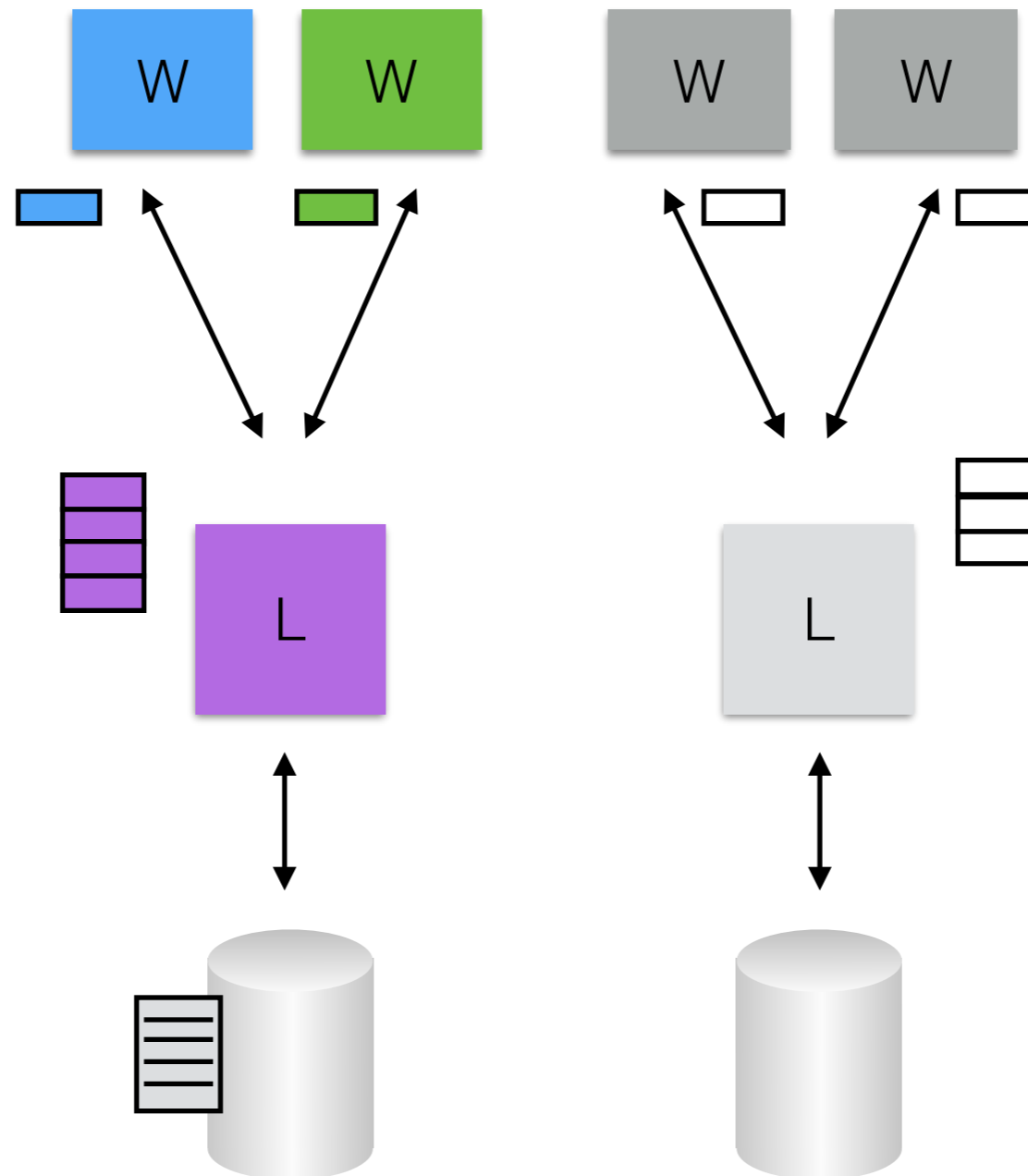
Pros and cons

- Reduces contention (compared to a global counter)
- **Ordering of TIDs doesn't follow the ordering of transactions!**
 - Problems with write-after-read dependencies
 - Replaying the transactions in TID order might recover the wrong database
- Still, the database is easily recoverable from logs
 - Largest TID wins

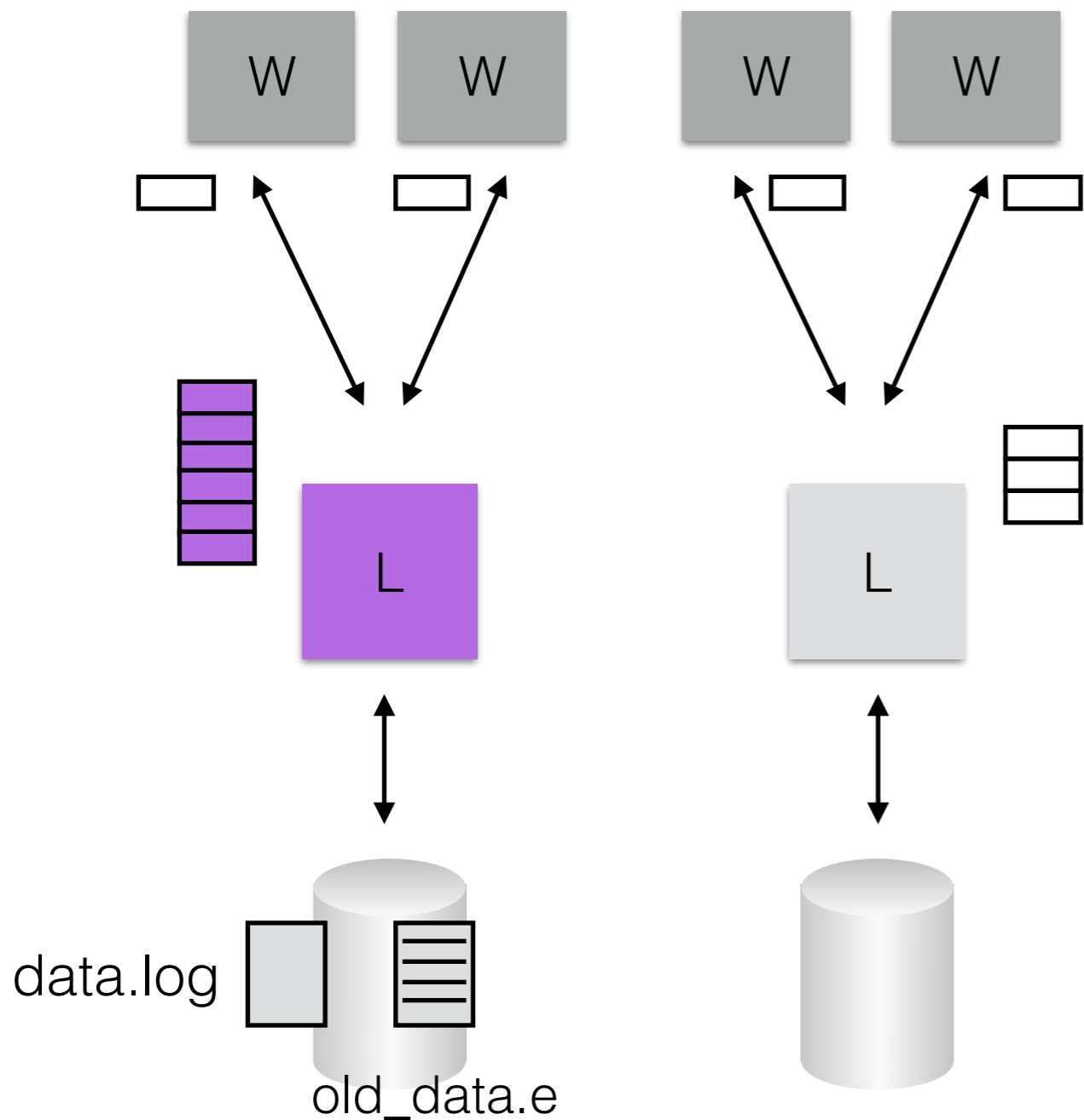
Logging structure



Logging structure



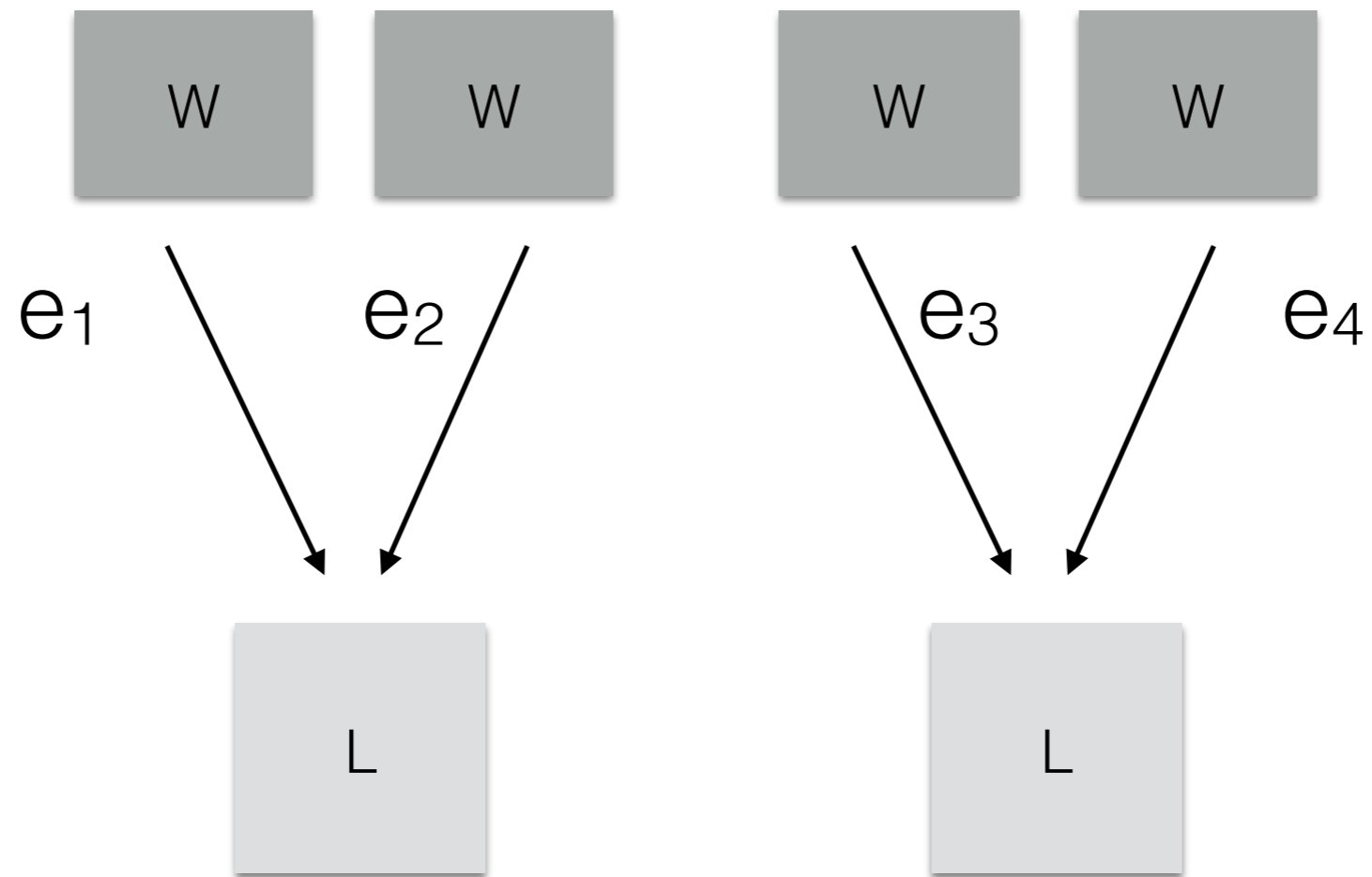
Log rotation



Log file renamed to **old_data.e**, where **e** is the largest epoch seen in that particular file.



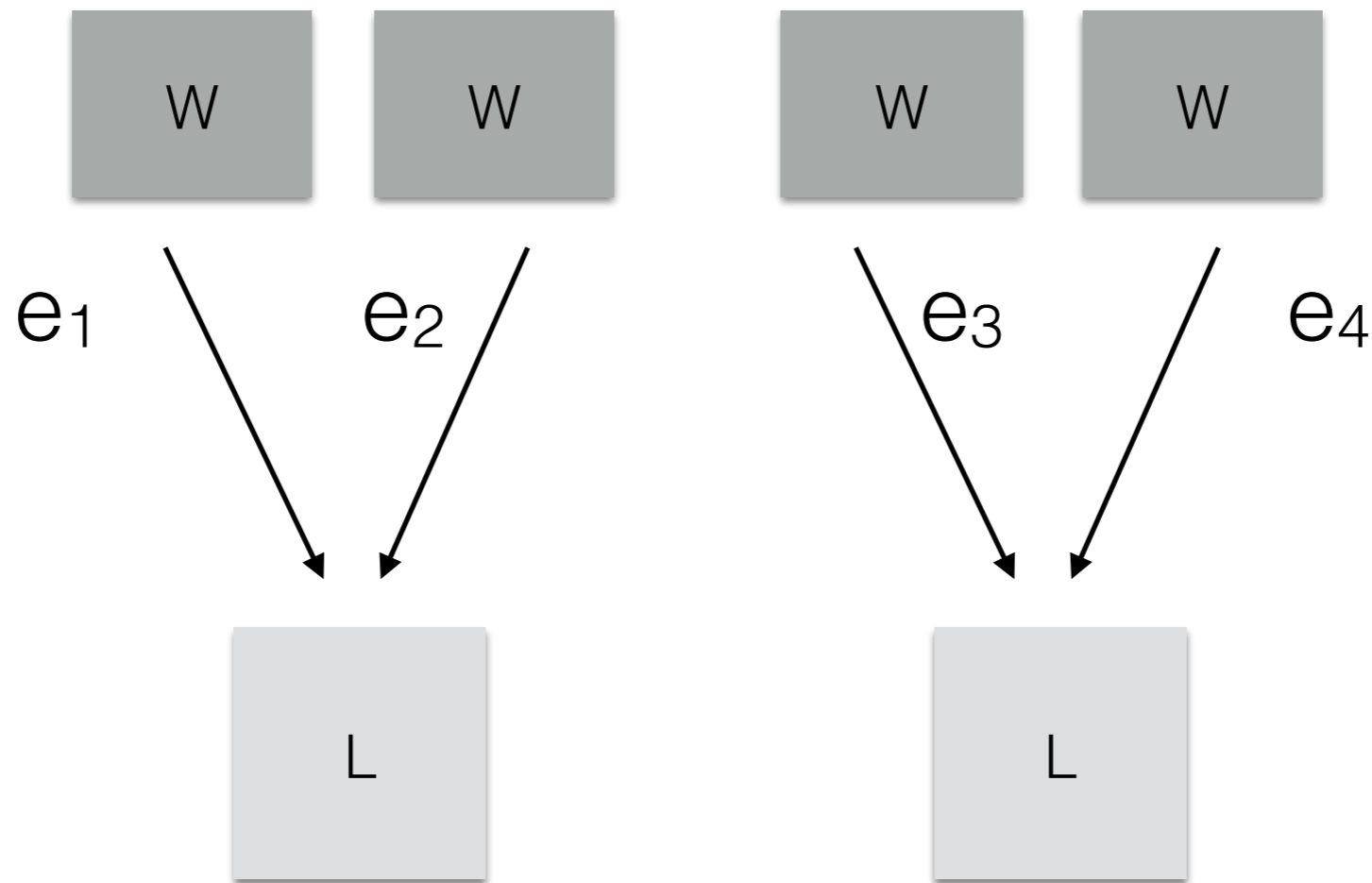
Persistence epoch



$$\mathbf{e_P} = \min \{e_1, e_2, e_3, e_4\} - 1$$



Persistence epoch

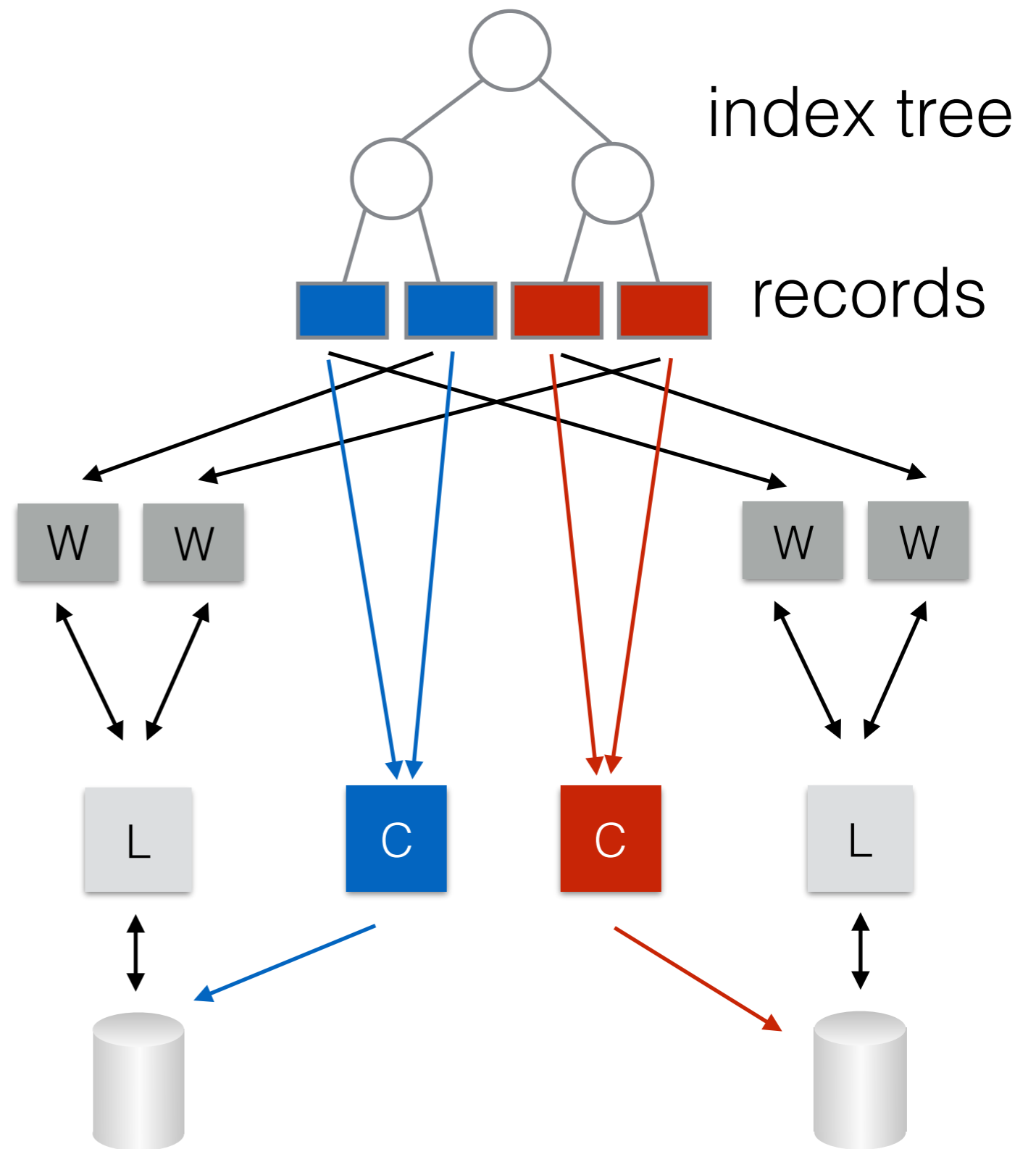


$$\mathbf{e_P} = \min \{e_1, e_2, e_3, e_4\} - 1$$

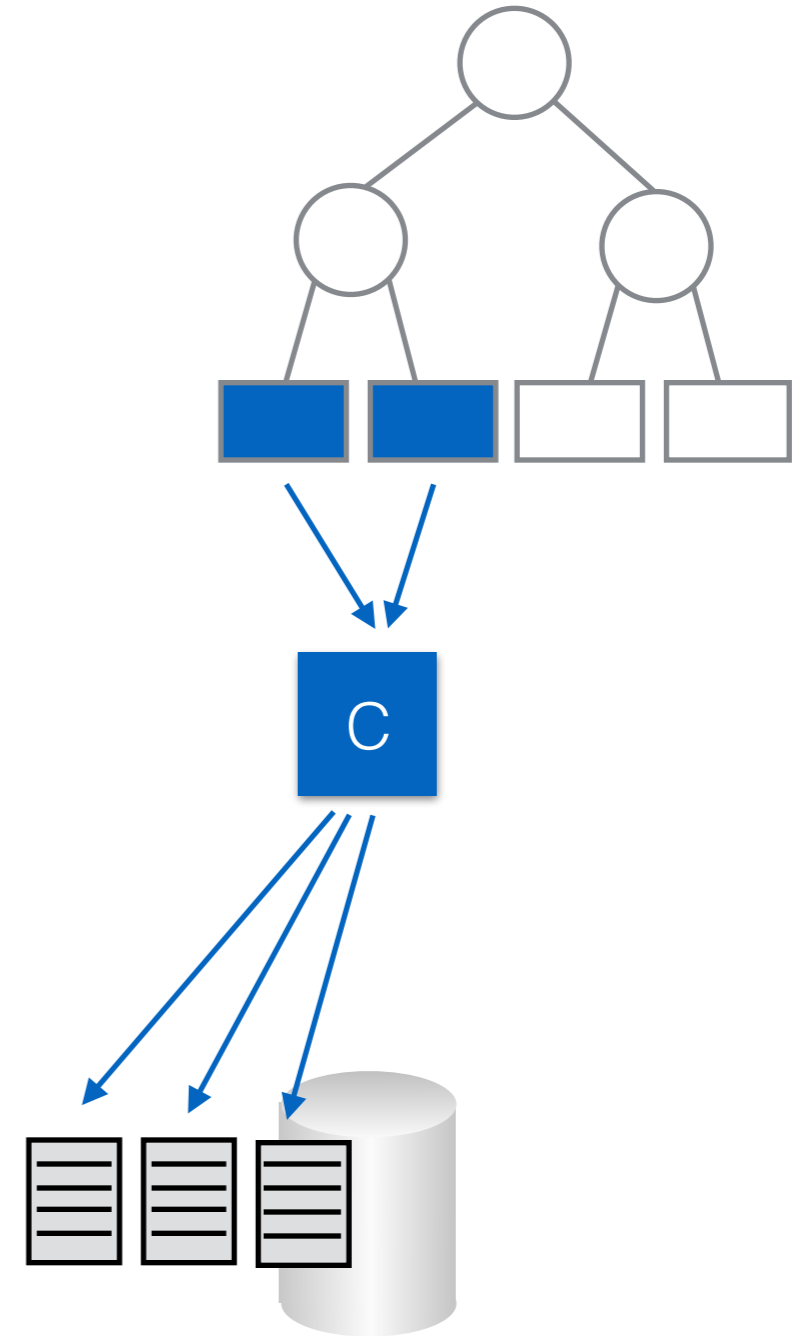
all transactions in epochs $\leq \mathbf{e_P}$
are persistent



- Parallel checkpointing
- Checkpoint happens regularly



- Tree walk over a range of each table - inconsistent checkpoint
- Only committed records in checkpoint
- Writes out to multiple files, enabling easy recovery parallelism



Checkpointing process

- Each checkpoint has an associated interval $[a,b]$
 - a - epoch when all checkpointers started
 - b - epoch when the checkpoint has been completed
- Checkpoints save only records with epochs $\leq a$
 - 20% reduction in space
- After the checkpoint, old log files can be deleted
 - Not all of them though

Other checkpointing techniques

- Snapshots
 - Guarantee consistency
 - Greatly increase memory usage
- Saving diffs
 - Much more complicated
 - Requires specialised file format design
 - Not necessary

Checkpoint recovery

- Load the most recent checkpoint and complete it with latest logs
- Each table is spread across multiple disks and multiple files on each of them
- Data is organised so that the recovery threads benefit from data locality and have little interference

Log recovery

- Data is unstructured and out of order
 - Not an issue - largest TID wins
 - Still very little contention
- Exposes a lot of parallelism thanks to value logging
- Replays state up to the last persistent epoch
- Uses ALL CORES
- Processes files in the opposite order they were written

Evaluation

- Experiment setup
 - single machine with four 8 core Intel Xeon E7-4830 processors (32 physical cores)
 - machine has 256 GB of DRAM, 64 GB of DRAM attached to each socket
 - 4 disks: 3 Fusion IO drives, 1 RAID-5 disk array

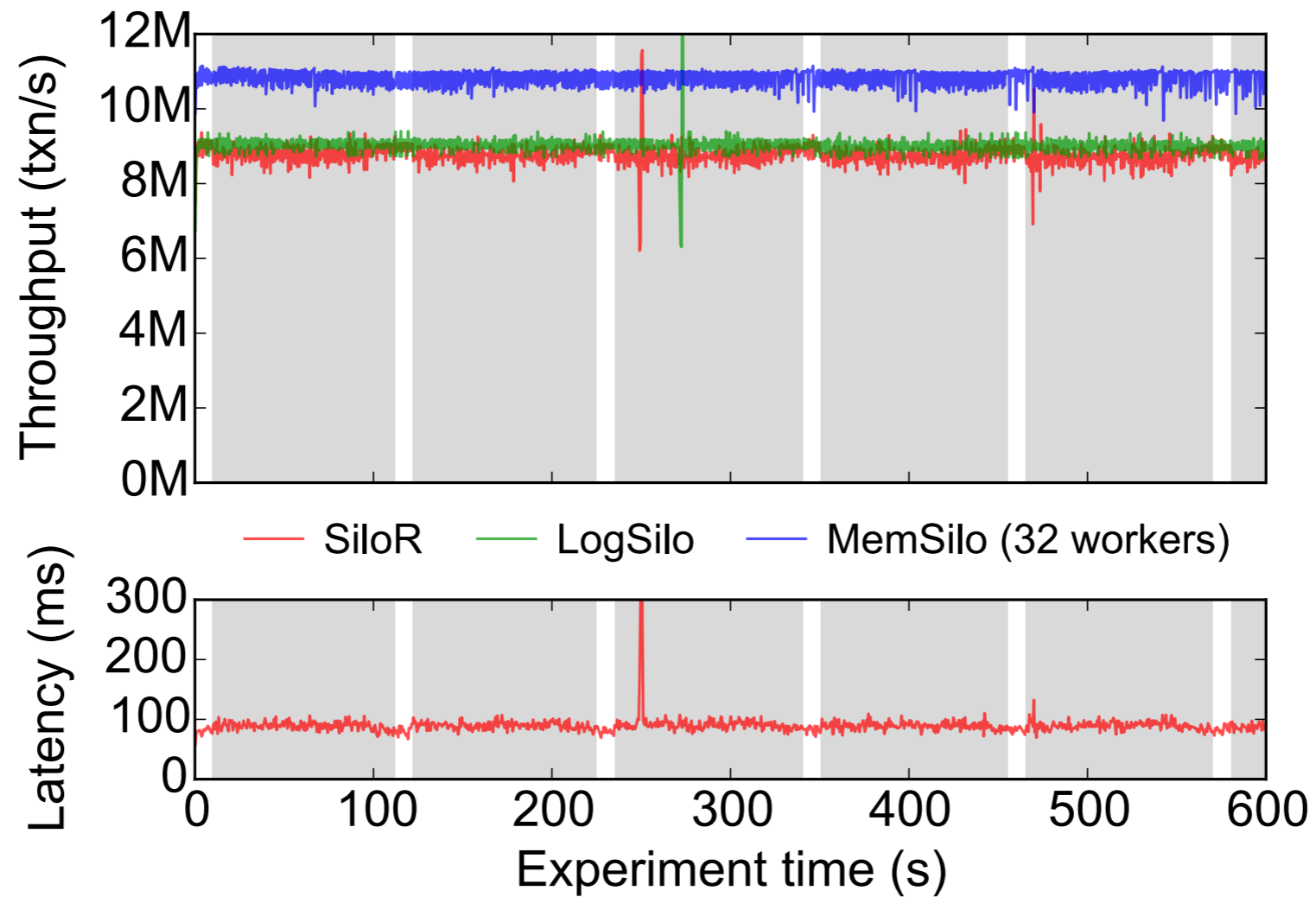


Evaluation - YCSB-A

- Key-value benchmark
- 400 million keys, 100 byte records
- 70% read, 30% write
- 28 workers, 4 loggers, 4 checkpoint threads
- Database does not grow



Evaluation - YCSB-A



Avg throughput: 8.76 Mtxns/s, 9.01 Mtxns/s, 10.83 Mtxns/s



Recovery for YCSB-A

Simulates crash right before the *second* checkpoint completes

	Recovered database	Checkpoint	Log	Total
Size	43.2 GB	36 GB	64 GB	100 GB
Recovery time		33 s	73 s	106 s

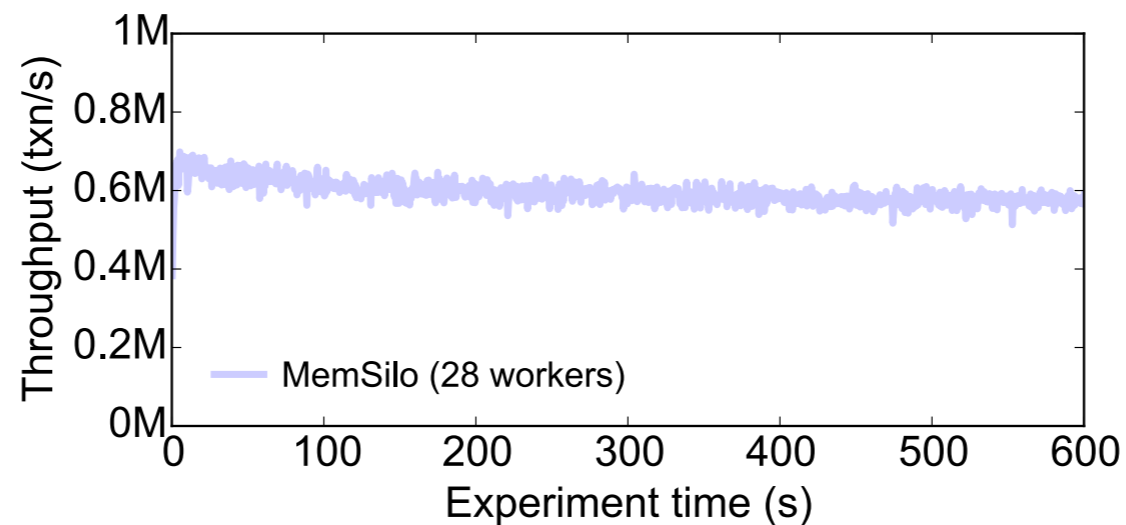
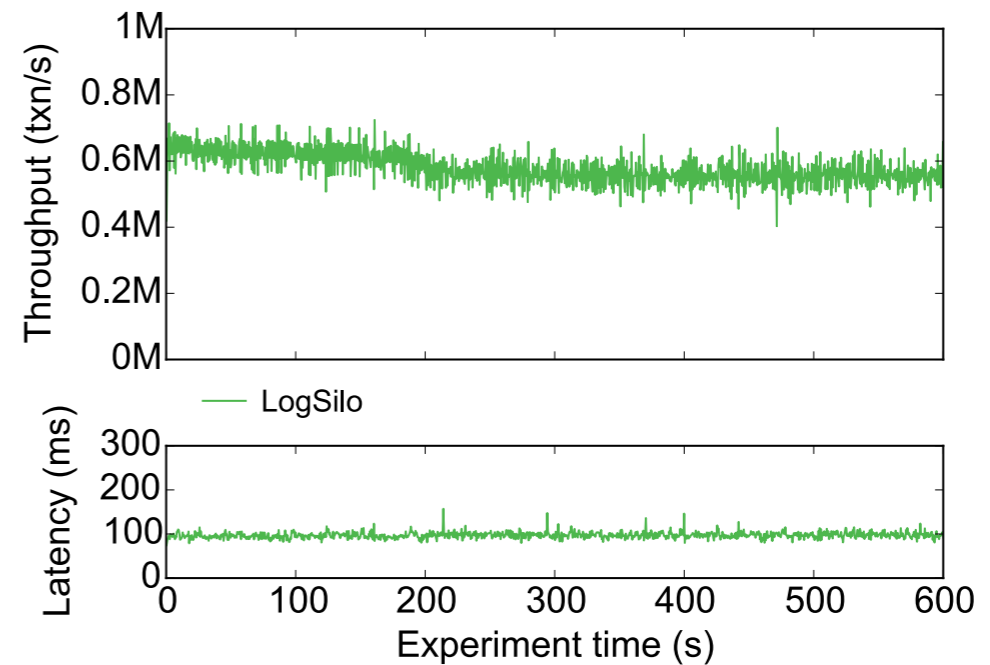
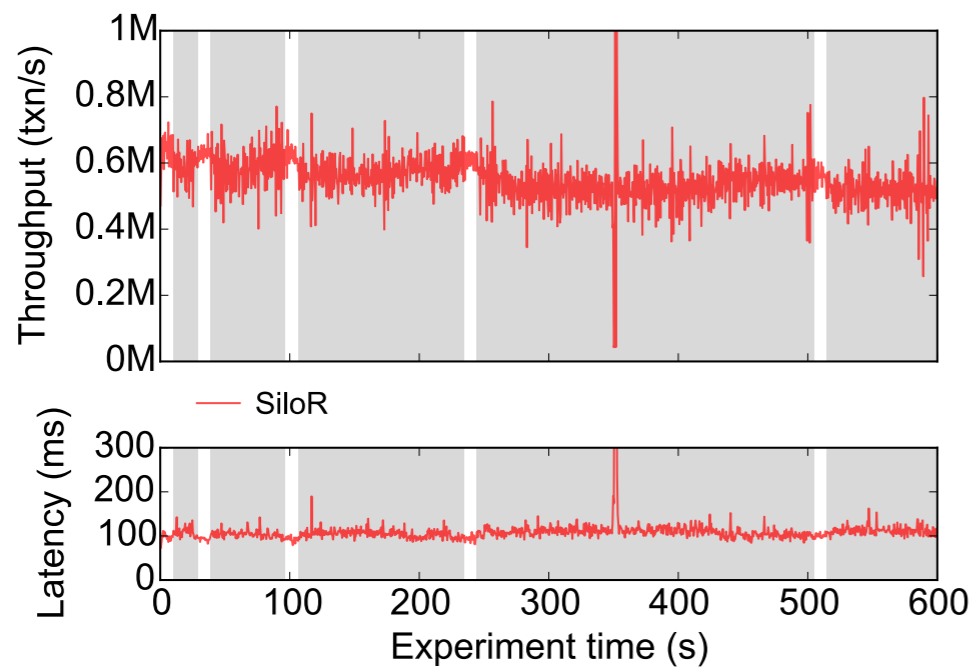


Evaluation - TPC-C

- TPC-C is a popular OLTP benchmark
- 28 workers, 4 loggers, 4 checkpoint threads
- Database size grows very fast
- Checkpoint period also grows



Evaluation - TPC-C



Avg throughput: 548 Ktxns/s, 575 Ktxns/s, 592 Ktxns/s



Recovery for TPC-C

Simulates crash right before the *fourth* checkpoint completes

	Recovered tuples	Checkpoint	Log	Total
Size	72.2 GB	15.7 GB	180 GB	195.7 GB
Recovery time		17 s	194 s	211 s



Conclusion

- Built a persistence system for a very fast multicore in-memory database
- Used parallelism in all parts of the system to enable
 - small degradation in runtime performance
 - recovery of large database in a few minutes



Thank you!

Questions?