Lecture #12

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Recovery Protocols

@Andy_Pavlo // 15-721 // Spring 2019

DATABASE RECOVERY

Recovery algorithms are techniques to ensure database **consistency**, **atomicity** and **durability** despite failures.

Recovery algorithms have two parts:

→ Actions during normal txn processing to ensure that the DBMS can recover from a failure.

→ Actions after a failure to recover the database to a state that ensures atomicity, consistency, and durability.

OBSERVATION

Many of the early papers (1980s) on recovery for in-memory DBMSs assume that there is non-volatile memory.

→ Battery-backed DRAM is large / finnicky
→ Real NVM is coming…

This hardware is still not widely available so we want to use existing SSD/HDDs.

A RECOVERY ALGORITHM FOR A HIGH-PERFORMANCE MEMORY-RESIDENT DATABASE SYSTEM
SIGMOD 1987

IN-MEMORY DATABASE RECOVERY

Slightly easier than in a disk-oriented DBMS because the system has to do less work:

→ Do not need to track dirty pages in case of a crash during recovery.
→ Do not need to store undo records (only need redo).
→ Do not need to log changes to indexes.

But the DBMS is still stymied by the slow sync time of non-volatile storage.

TODAY'S AGENDA

Logging Schemes
Checkpoint Protocols
Restart Protocols

LOGGING SCHEMES

Physical Logging

→ Record the changes made to a specific record in the database.
→ Example: Store the original value and after value for an attribute that is changed by a query.

Logical Logging

→ Record the high-level operations executed by txns.
→ Example: The **UPDATE**, **DELETE**, and **INSERT** queries invoked by a txn.

PHYSICAL VS. LOGICAL LOGGING

Logical logging writes less data in each log record than physical logging.

Difficult to implement recovery with logical logging if you have concurrent txns.

→ Harder to determine which parts of the database may have been modified by a query before crash if running at lower isolation level.
→ Takes longer to recover because you must re-execute every txn all over again.

SILO

In-memory OLTP DBMS from Harvard/MIT.

→ Single-versioned OCC with epoch-based GC.
→ Same authors of the Masstree.
→ Eddie Kohler is unstoppable.

SiloR uses physical logging + checkpoints to ensure durability of txns.

→ It achieves high performance by parallelizing all aspects of logging, checkpointing, and recovery.

FAST DATABASES WITH FAST DURABILITY AND RECOVERY THROUGH MULTICORE PARALLELISM
OSDI 2014

SILOR – LOGGING PROTOCOL

The DBMS assumes that there is one storage device per CPU socket.

→ Assigns one logger thread per device.
→ Worker threads are grouped per CPU socket.

As the worker executes a txn, it creates new log records that contain the values that were written to the database (i.e., REDO).

SILOR – LOGGING PROTOCOL

Each logger thread maintains a pool of log buffers that are given to its worker threads.

When a worker's buffer is full, it gives it back to the logger thread to flush to disk and attempts to acquire a new one.

→ If there are no available buffers, then it stalls.

SILOR – LOG FILES

The logger threads write buffers out to files:

- → After 100 epochs, it creates a new file.
- → The old file is renamed with a marker indicating the max epoch of records that it contains.

Log record format:

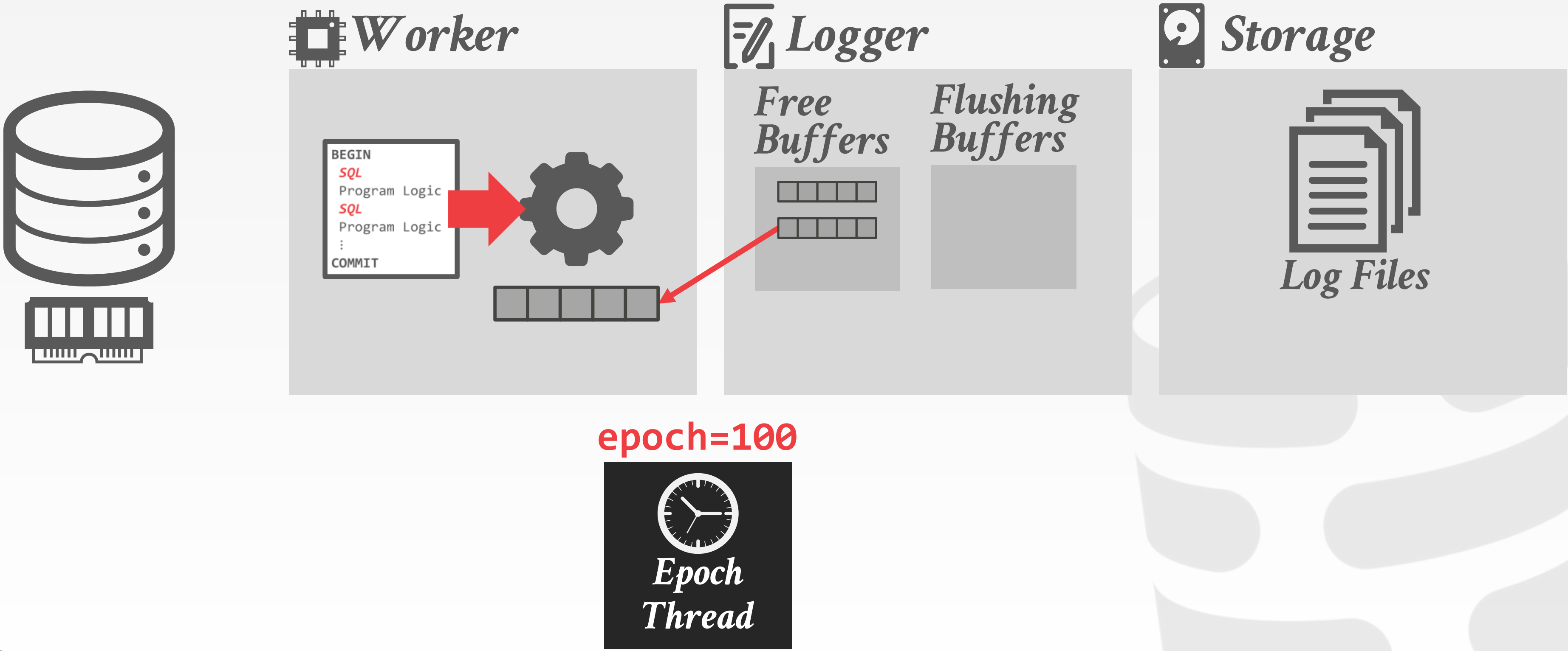
- → Id of the txn that modified the record (TID).
- → A set of value log triplets (Table, Key, Value).
- → The value can be a list of attribute + value pairs.

```
UPDATE people
   SET isLame = true
 WHERE name IN ('Lin','Andy')
```

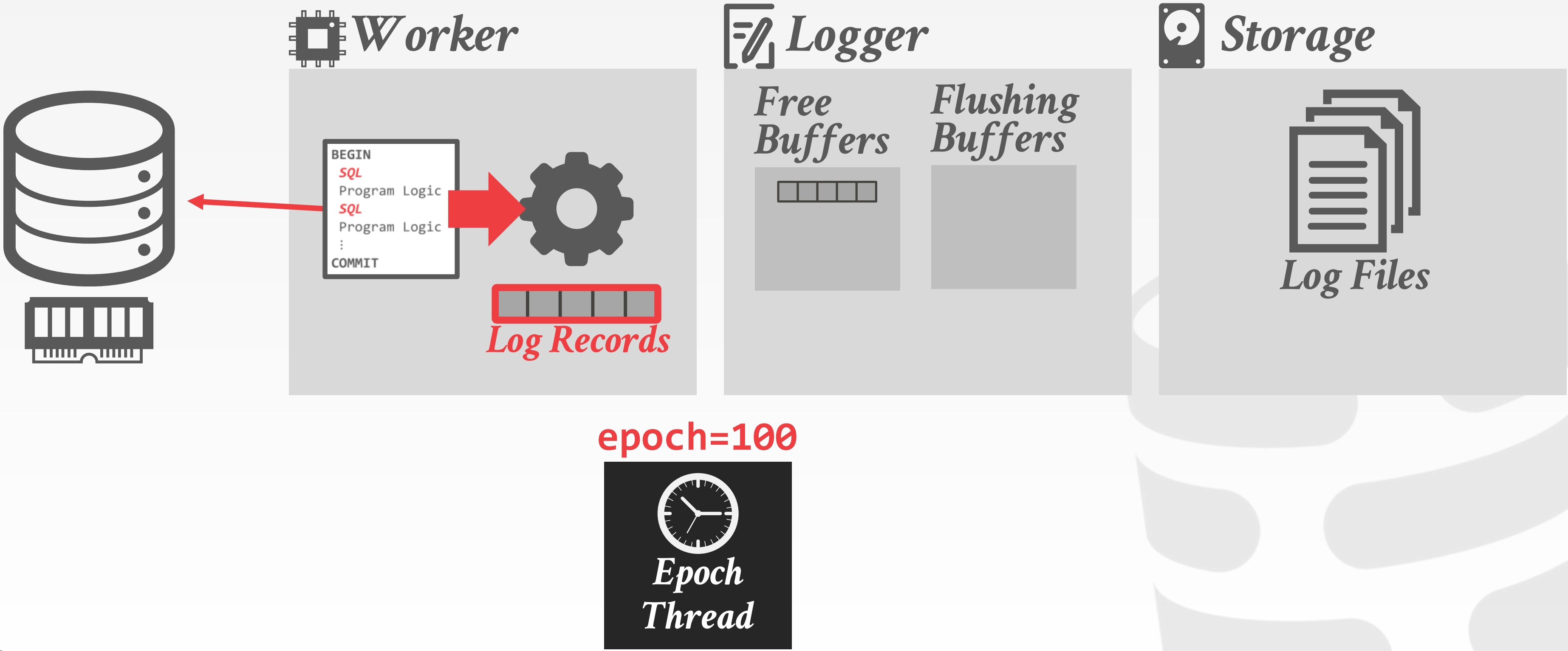
→

```
Txn#1001
[people, 888, (isLame→true)]
[people, 999, (isLame→true)]
```

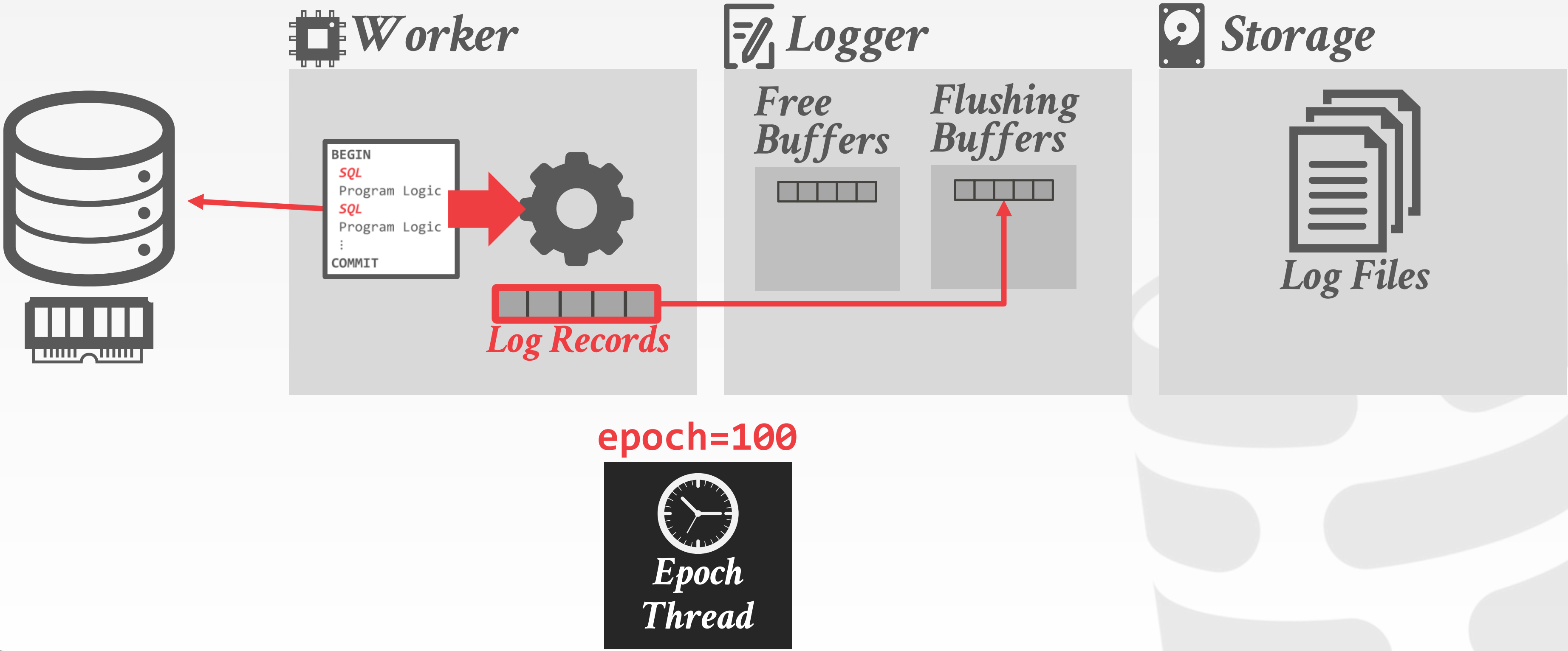
SILOR – ARCHITECTURE



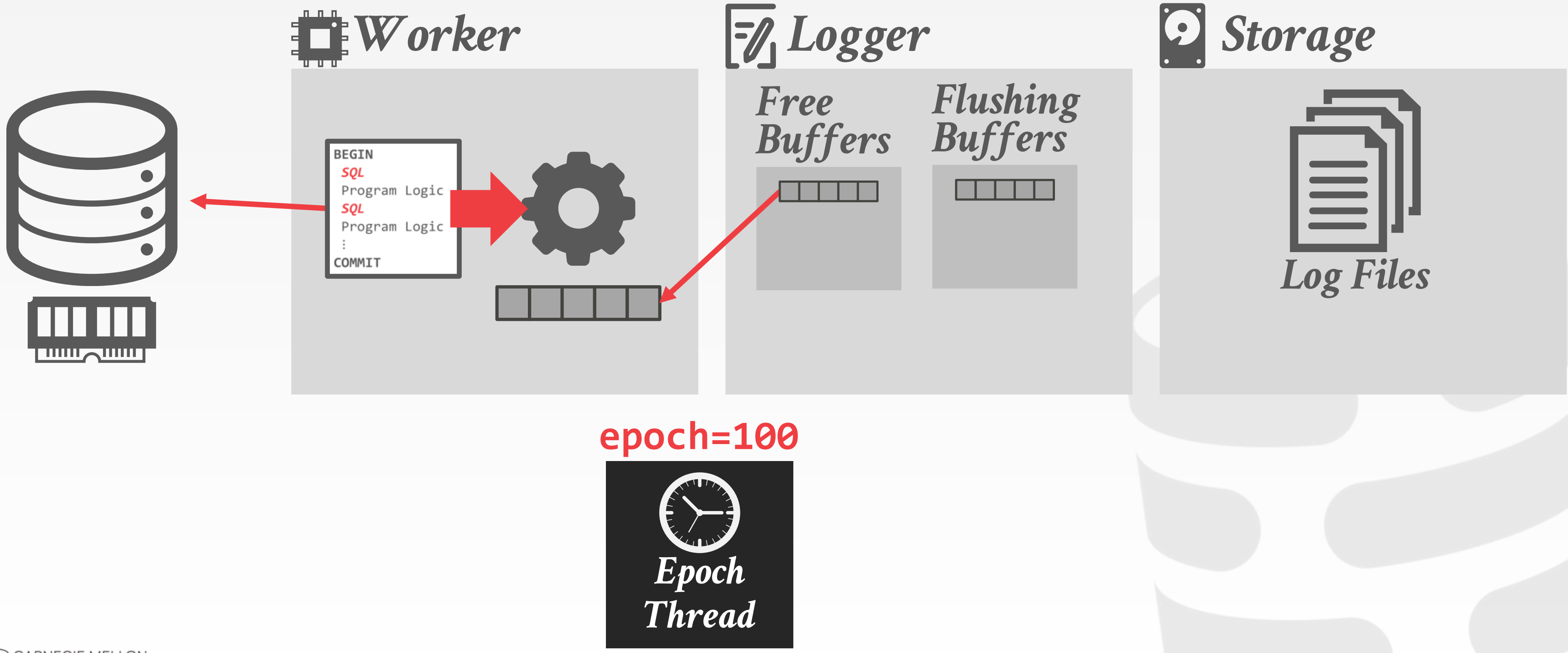
SILOR – ARCHITECTURE



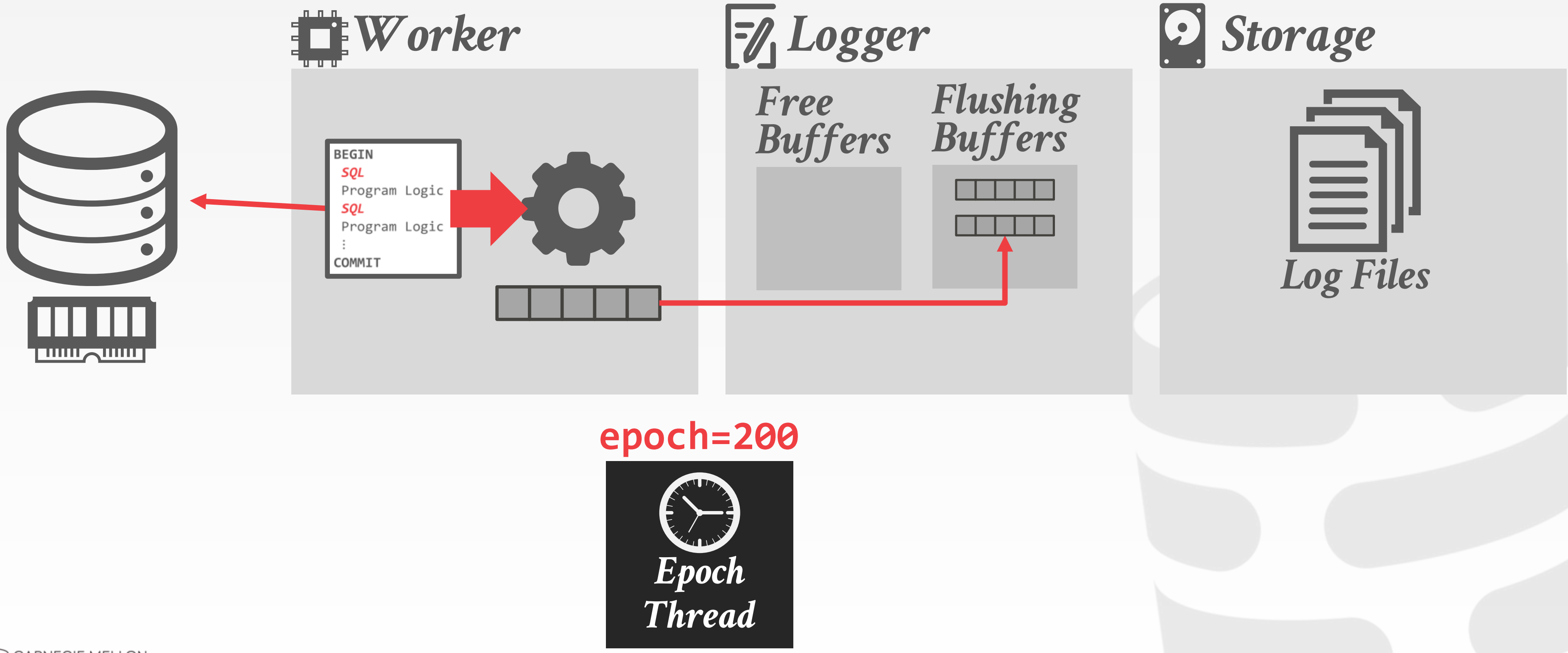
SILOR – ARCHITECTURE



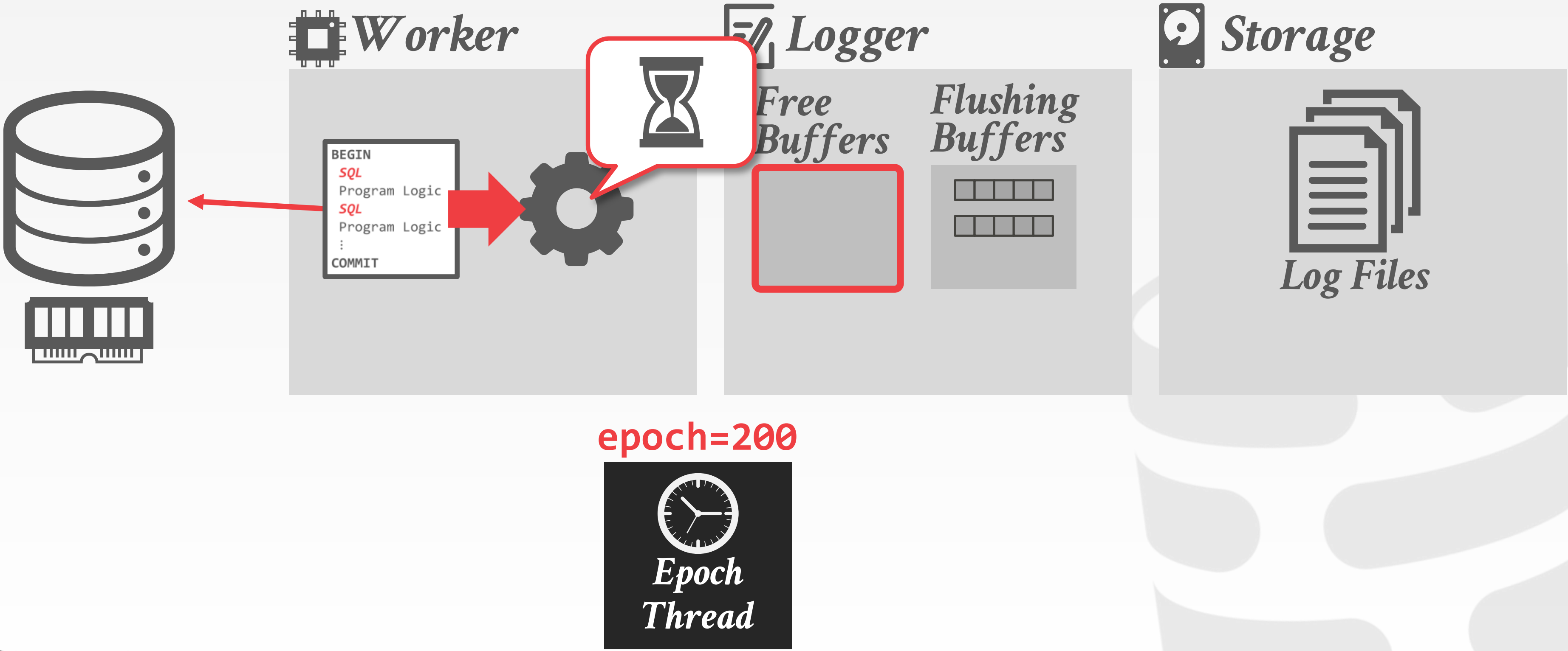
SILOR – ARCHITECTURE



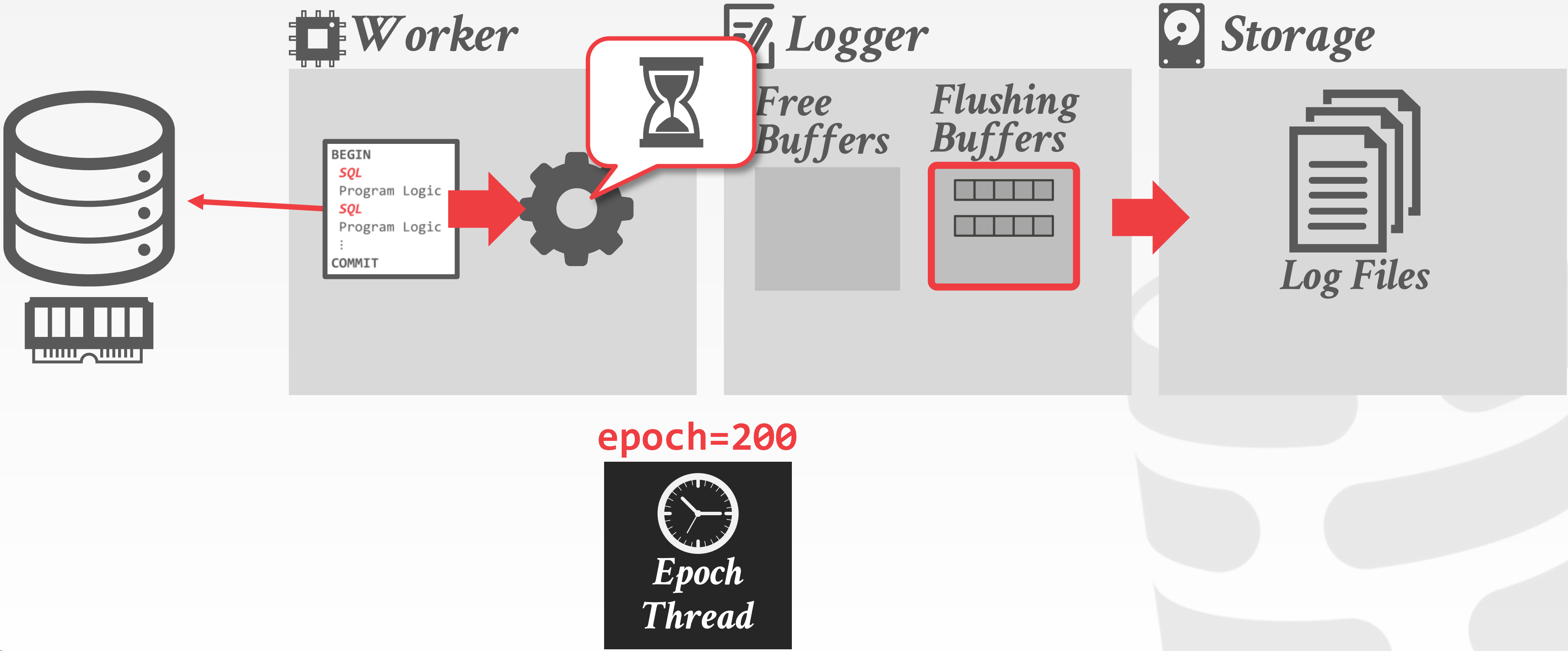
SILOR – ARCHITECTURE



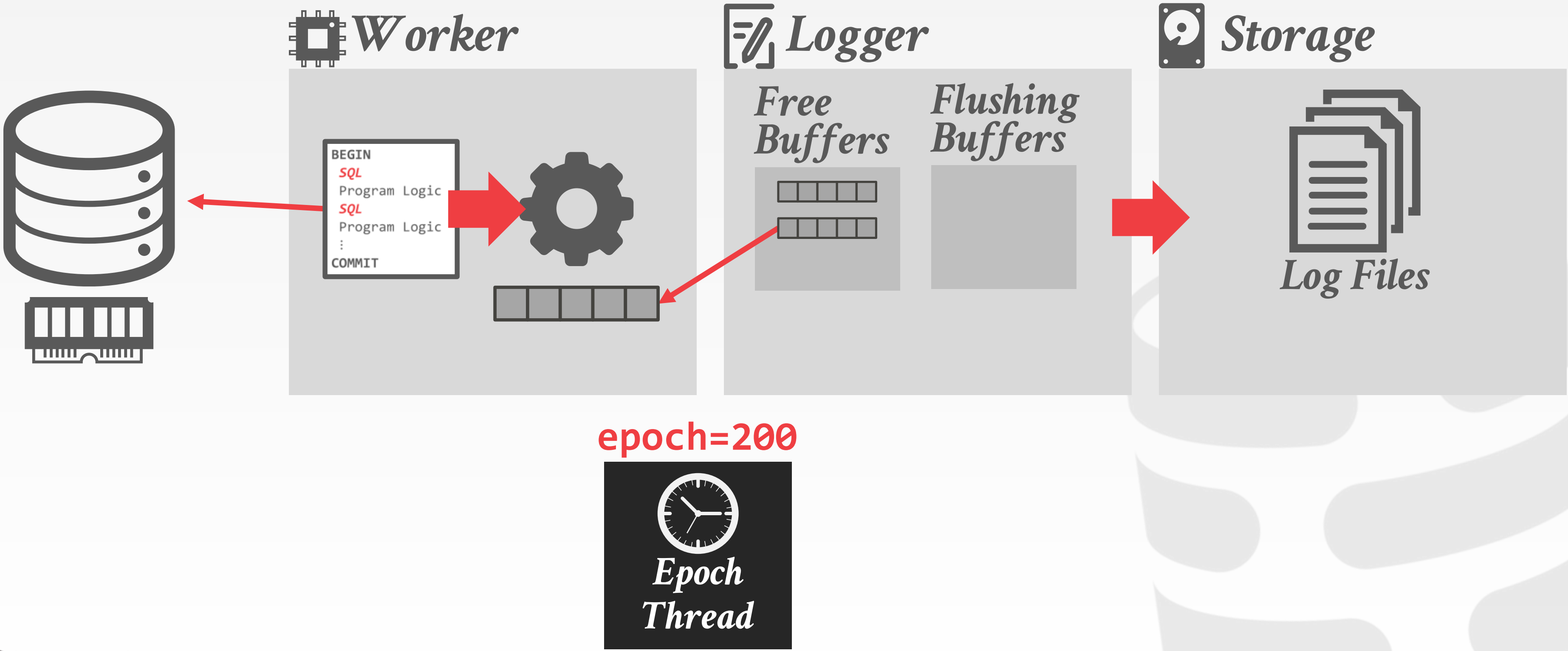
SILOR – ARCHITECTURE



SILOR – ARCHITECTURE



SILOR – ARCHITECTURE



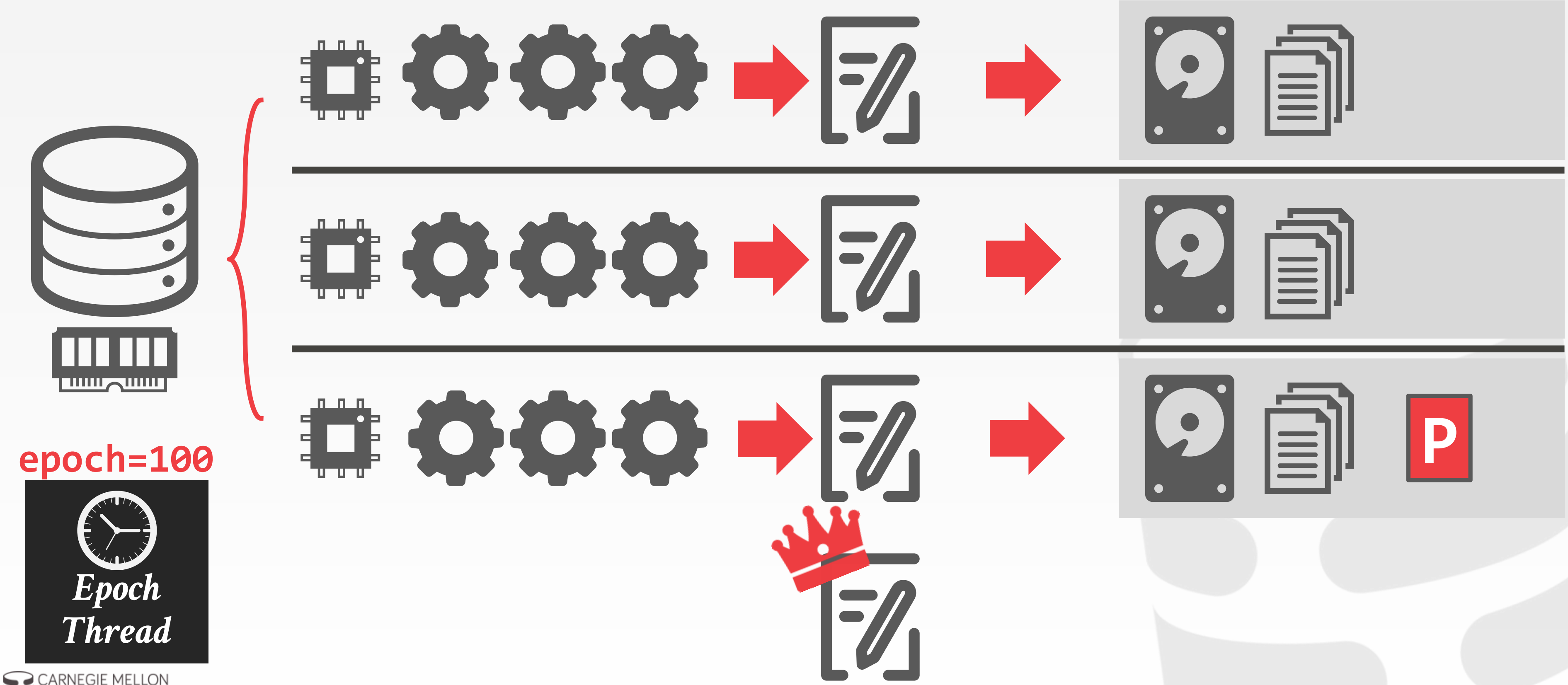
SILOR – PERSISTENT EPOCH

A special logger thread keeps track of the current persistent epoch (***pepoch***)

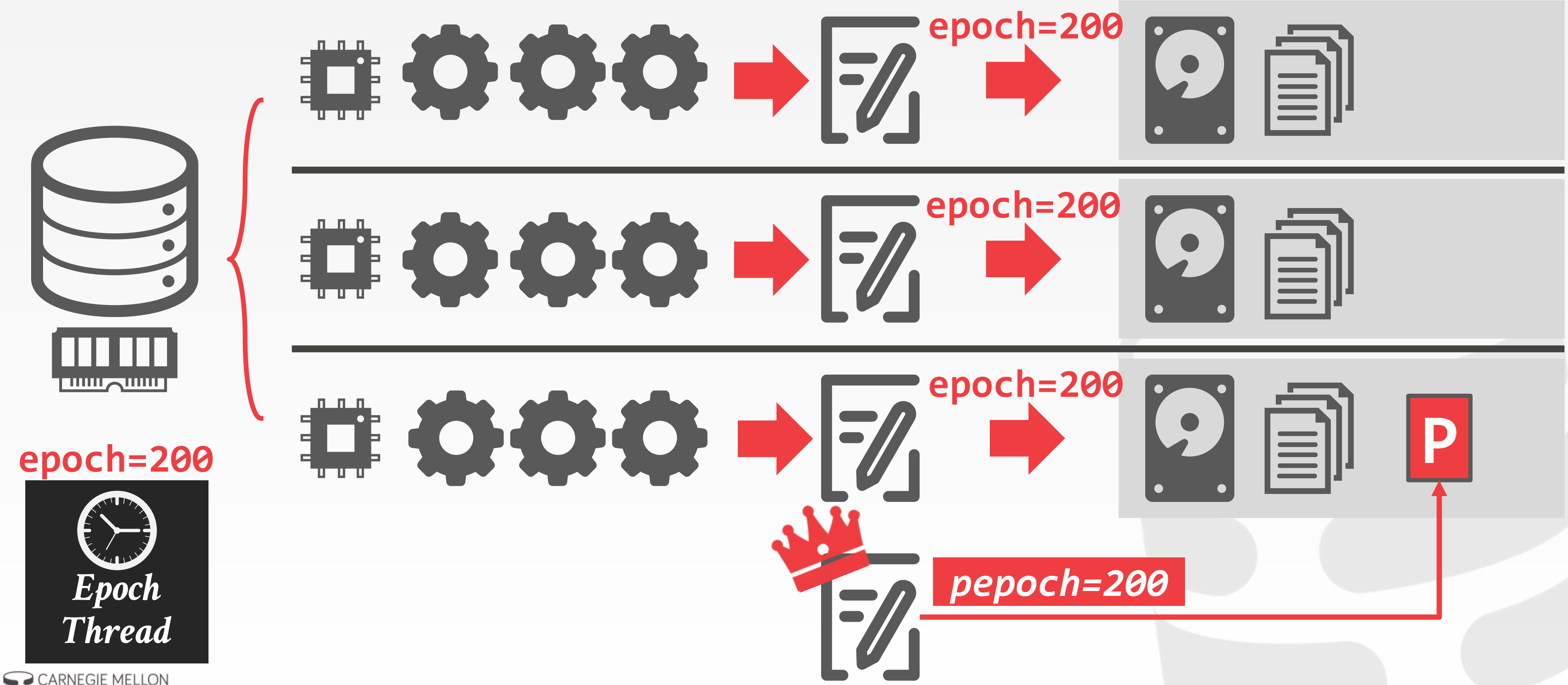
→ Special log file that maintains the highest epoch that is durable across all loggers.

Txns that executed in epoch ***e*** can only release their results when the ***pepoch*** is durable to non-volatile storage.

SILOR – ARCHITECTURE



SILOR – ARCHITECTURE



SILOR – RECOVERY PROTOCOL

Phase #1: Load Last Checkpoint

→ Install the contents of the last checkpoint that was saved into the database.
→ All indexes have to be rebuilt.

Phase #2: Log Replay

→ Process logs in reverse order to reconcile the latest version of each tuple.
→ The txn ids generated at runtime are enough to determine the serial order on recovery.

SILOR – LOG REPLAY

First check the ***pepoch*** file to determine the most recent persistent epoch.

→ Any log record from after the ***pepoch*** is ignored.

Log files are processed from newest to oldest.

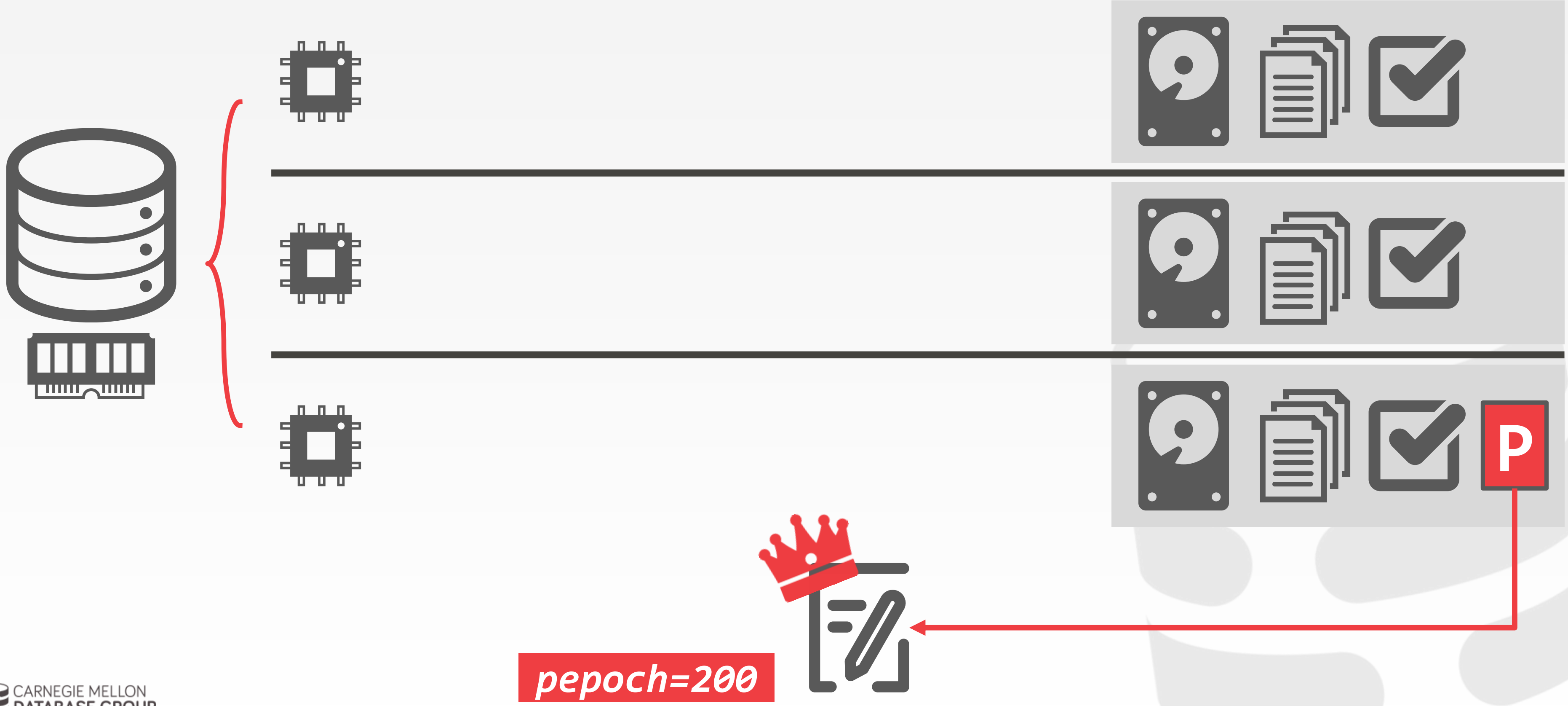
→ Value logging is able to be replayed in any order.

→ For each log record, the thread checks to see whether the tuple already exists.

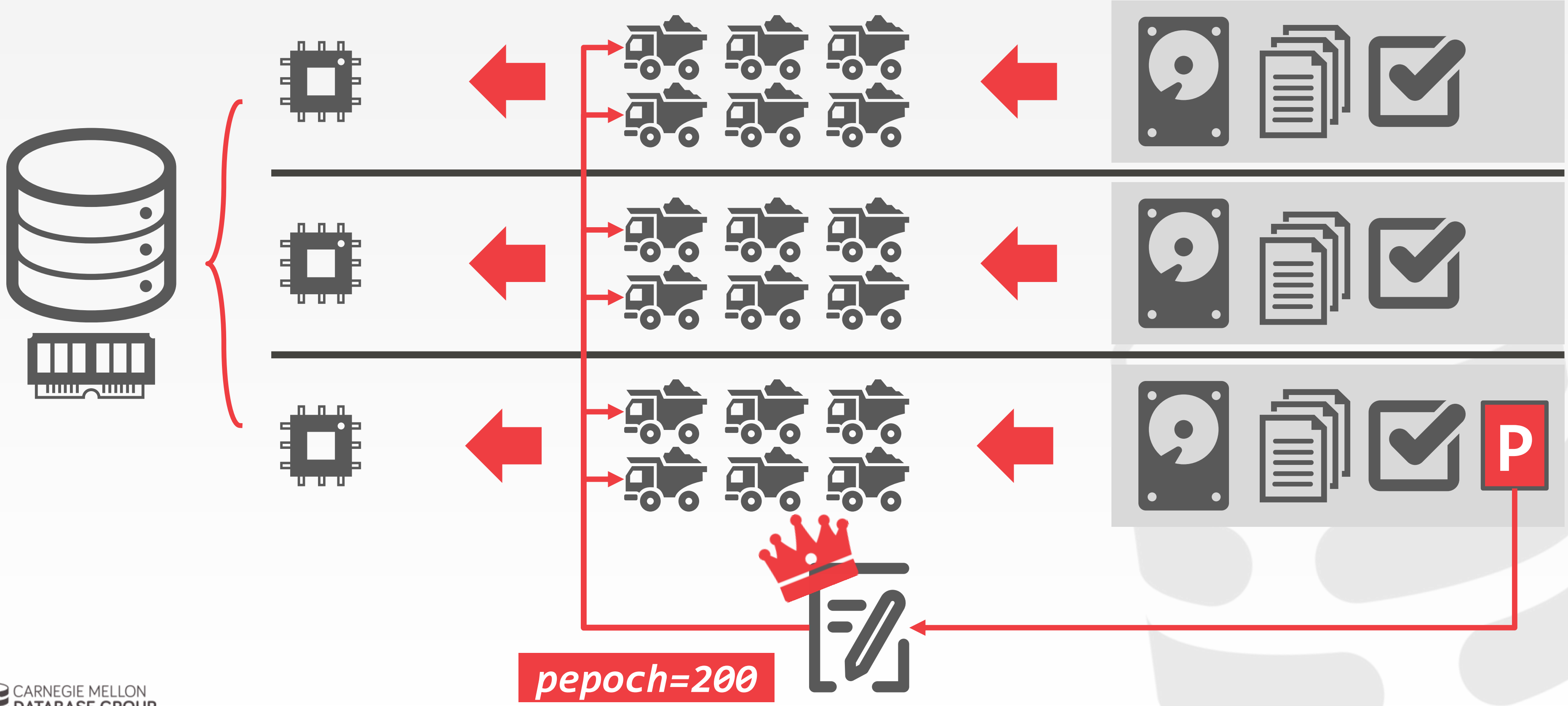
→ If it does not, then it is created with the value.

→ If it does, then the tuple's value is overwritten only if the log TID is newer than tuple's TID.

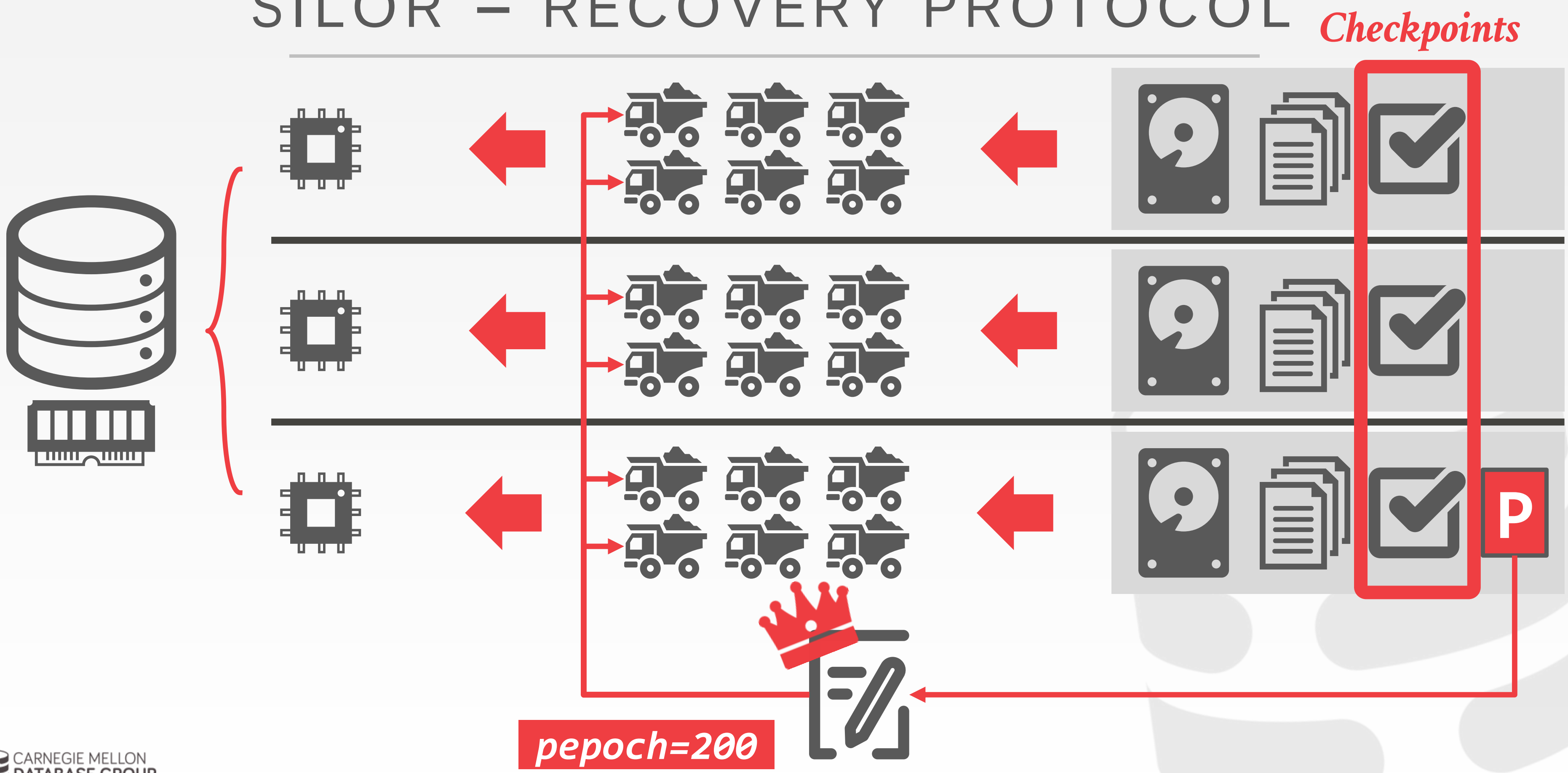
SILOR – RECOVERY PROTOCOL



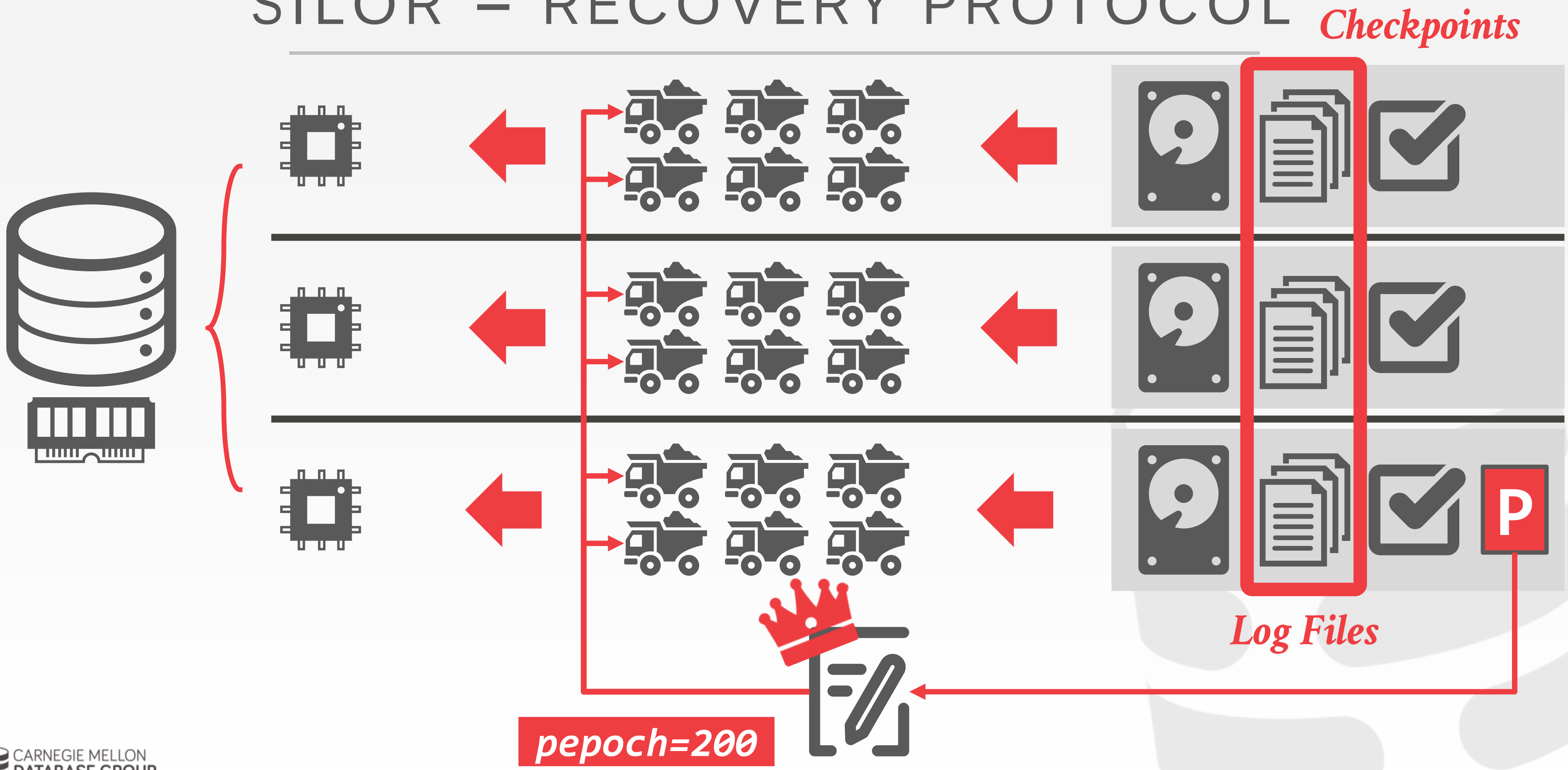
SILOR – RECOVERY PROTOCOL



SILOR – RECOVERY PROTOCOL



SILOR – RECOVERY PROTOCOL



OBSERVATION

Often the slowest part of the txn is waiting for the DBMS to flush the log records to disk.

Have to wait until the records are safely written before the DBMS can return the acknowledgement to the client.

GROUP COMMIT

Batch together log records from multiple txns and flush them together with a single **fsync**.

→ Logs are flushed either after a timeout or when the buffer gets full.
→ Originally developed in IBM IMS FastPath in the 1980s

This amortizes the cost of I/O over several txns.

EARLY LOCK RELEASE

A txn's locks can be released before its commit record is written to disk as long as it does not return results to the client before becoming durable.

Other txns that read data updated by a **pre-committed** txn become dependent on it and also have to wait for their predecessor's log records to reach disk.

OBSERVATION

Logging allows the DBMS to recover the database after a crash/restart. But this system will have to replay the entire log each time.

Checkpoints allows the systems to ignore large segments of the log to reduce recovery time.

IN-MEMORY CHECKPOINTS

There are different approaches for how the DBMS can create a new checkpoint for an in-memory database.

The choice of approach in a DBMS is tightly coupled with its concurrency control scheme.

The checkpoint thread(s) scans each table and writes out data asynchronously to disk.

IDEAL CHECKPOINT PROPERTIES

Do **not** slow down regular txn processing.

Do **not** introduce unacceptable latency spikes.

Do **not** require excessive memory overhead.

CONSISTENT VS. FUZZY CHECKPOINTS

Approach #1: Consistent Checkpoints

→ Represents a consistent snapshot of the database at some point in time. No uncommitted changes.
→ No additional processing during recovery.

Approach #2: Fuzzy Checkpoints

→ The snapshot could contain records updated from transactions that have not finished yet.
→ Must do additional processing to remove those changes.

CHECKPOINT MECHANISM

Approach #1: Do It Yourself

→ The DBMS is responsible for creating a snapshot of the database in memory.
→ Can leverage on multi-versioned storage.

Approach #2: OS Fork Snapshots

→ Fork the process and have the child process write out the contents of the database to disk.
→ This copies everything in memory.
→ Requires extra work to remove uncommitted changes.

HYPER – OS FORK SNAPSHOTS

Create a snapshot of the database by forking the DBMS process.

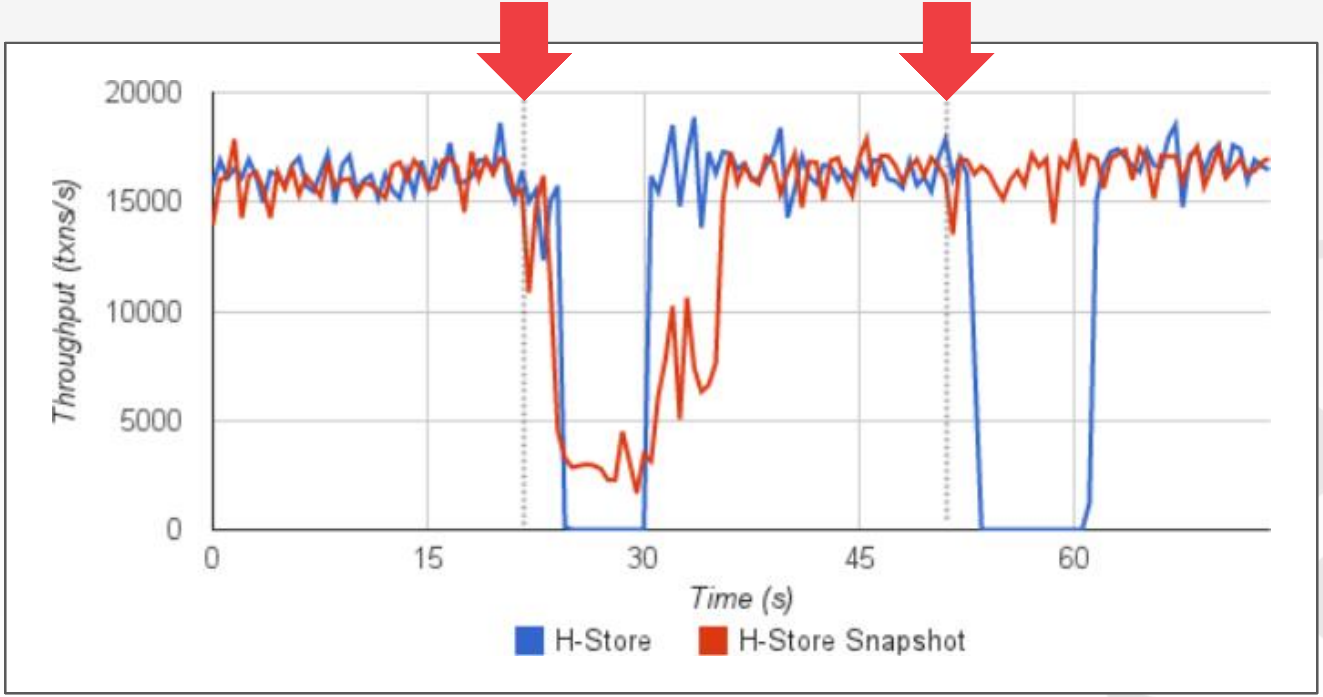
→ Child process contains a consistent checkpoint if there are not active txns.
→ Otherwise, use the in-memory undo log to roll back txns in the child process.

Continue processing txns in the parent process.

HYPER: A HYBRID OLTP&OLAP MAIN MEMORY DATABASE SYSTEM BASED ON VIRTUAL MEMORY SNAPSHOTS
ICDE 2011

H-STORE – OS FORK SNAPSHOTS

Workload: TPC-C (8 Warehouses) + OLAP Query



CHECKPOINT CONTENTS

Approach #1: Complete Checkpoint

→ Write out every tuple in every table regardless of whether were modified since the last checkpoint.

Approach #2: Delta Checkpoint

→ Write out only the tuples that were modified since the last checkpoint.
→ Can merge checkpoints together in the background.

FREQUENCY

Approach #1: Time-based

→ Wait for a fixed period of time after the last checkpoint has completed before starting a new one.

Approach #2: Log File Size Threshold

→ Begin checkpoint after a certain amount of data has been written to the log file.

Approach #3: On Shutdown (Mandatory)

→ Perform a checkpoint when the DBA instructs the system to shut itself down. Every DBMS (hopefully) does this.

CHECKPOINT IMPLEMENTATIONS

	Type	*Contents*	*Frequency*
MemSQL	Consistent	Complete	Log Size
VoltDB	Consistent	Complete	Time-Based
Altibase	Fuzzy	Complete	Manual?
TimesTen	Consistent (*Blocking*) Fuzzy (*Non-Blocking*)	Complete Complete	On Shutdown Time-Based
Hekaton	Consistent	Delta	Log Size
SAP HANA	Fuzzy	Complete	Time-Based

OBSERVATION

Not all DBMS restarts are due to crashes.
- → Updating OS libraries
- → Hardware upgrades/fixes
- → Updating DBMS software

Need a way to be able to quickly restart the DBMS without having to re-read the entire database from disk again.

FACEBOOK SCUBA – FAST RESTARTS

Decouple the in-memory database lifetime from the process lifetime.

By storing the database shared memory, the DBMS process can restart and the memory contents will survive.

FAST DATABASE RESTARTS AT FACEBOOK
SIGMOD 2014

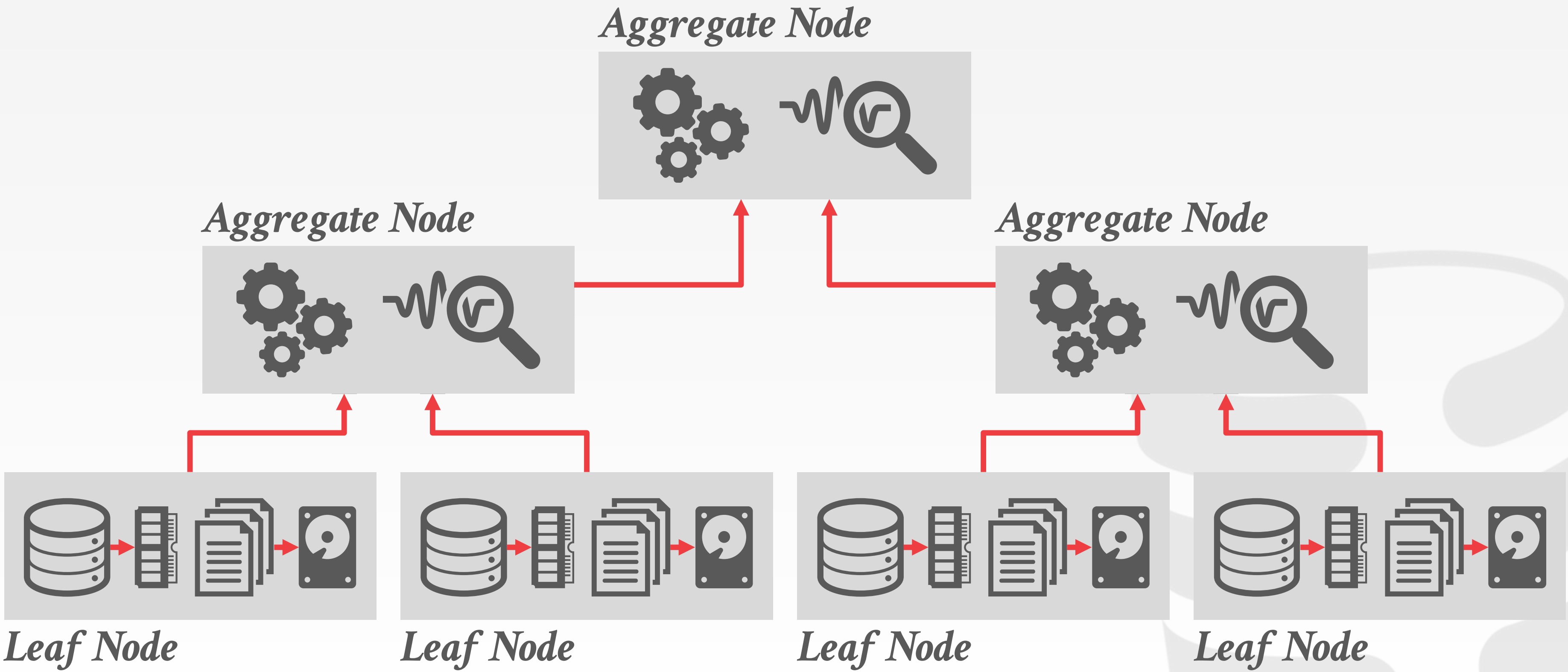
FACEBOOK SCUBA

Distributed, in-memory DBMS for time-series event analysis and anomaly detection.

Heterogeneous architecture

→ **Leaf Nodes**: Execute scans/filters on in-memory data
→ **Aggregator Nodes:** Combine results from leaf nodes

FACEBOOK SCUBA – ARCHITECTURE



SHARED MEMORY RESTARTS

Approach #1: Shared Memory Heaps

→ All data is allocated in SM during normal operations.
→ Have to use a custom allocator to subdivide memory segments for thread safety and scalability.
→ Cannot use lazy allocation of backing pages with SM.

Approach #2: Copy on Shutdown

→ All data is allocated in local memory during normal operations.
→ On shutdown, copy data from heap to SM.

Y RESTARTS

Roman Leventov @leventov · 24 Mar 2018
Replying to @andy_pavlo @CSDatCMU
In this lecture, you say that #jemalloc couldn't be ported to shared memory, because shared memory couldn't be allocated lazily. I think it could. Probably something different was meant.

1

Andy Pavlo @andy_pavlo · 24 Mar 2018
See Section 3 in the Scuba fast restart paper (SIGMOD'14). I think that it is a implementation issue that is specific to jemalloc 15721.courses.cs.cmu.edu /spring2018/pap...

> At Facebook, our default heap memory allocator is jemalloc [8]. Jason Evans, the author of jemalloc, discussed writing a new shared memory allocator with us. jemalloc uses lazy allocation of backing pages for virtual memory to avoid fragmentation. Since Scuba is entirely memory-bound (rather than CPU-bound), using memory efficiently is very important. In shared memory, lazy allocation of backing pages is not possible. We worried that an allocator in shared memory would lead to increased fragmentation over time.

1 1

Roman Leventov @leventov · 24 Mar 2018
I just manually checked. mmap with PROT_READ | PROT_WRITE, MAP_SHARED. File in /dev/shm. 4.4.0-1052-aws #61-Ubuntu Linux. It does allocate lazily according to `free` command.

1

Andy Pavlo @andy_pavlo · 25 Mar 2018
Nice! Good to know.

ory Heaps

g normal operations.
o subdivide memory
calability.
cking pages with SM.

down

ory during normal

eap to SM.

FACEBOOK SCUBA – FAST RESTARTS

When the admin initiates restart command, the node halts ingesting updates.

DBMS starts copying data from heap memory to shared memory.
→ Delete blocks in heap once they are in SM.

Once snapshot finishes, the DBMS restarts.
→ On start up, check to see whether the there is a valid database in SM to copy into its heap.
→ Otherwise, the DBMS restarts from disk.

PARTING THOUGHTS

Physical logging is a general purpose approach that supports all concurrency control schemes.

→ Logical logging is faster but not universal.

Copy-on-update checkpoints are the way to go especially if you are using MVCC

Non-volatile memory is coming…

NEXT CLASS

Networking Protocols

Project #2 Announcement + Potential Topics