



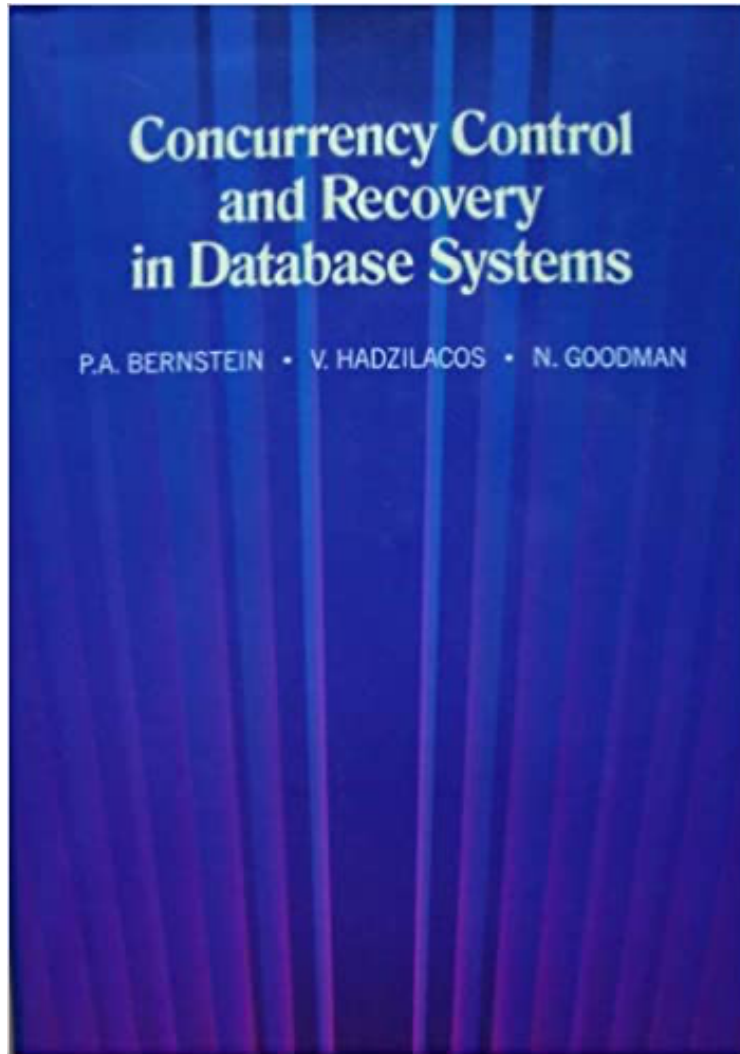
CS 764: Topics in Database Management Systems

Lecture 16: Durability

Xiangyao Yu

11/1/2021

Today's Paper: Durability



Addison-Wesley, 1987

Agenda

Durability

Force vs. No Force and Steal vs. No Steal

Logging schemes

- REDO only
- UNDO only
- REDO + UNDO
- No REDO + No UNDO

Durability

Durability: The database must recover to a valid state no matter when a crash occurs

- Committed transactions should persist
- Uncommitted transactions should roll back

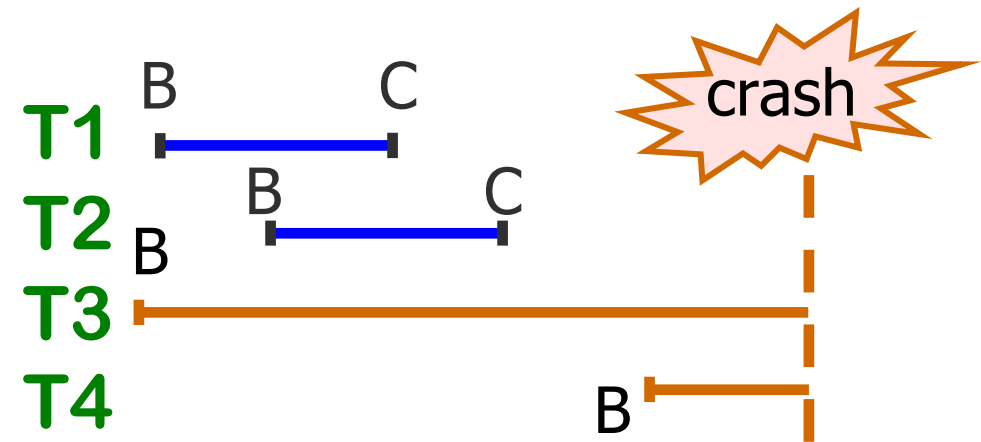
Durability

Durability: The database must recover to a valid state no matter when a crash occurs

- Committed transactions should persist
- Uncommitted transactions should roll back

Desired Behavior after system restarts


- T1, T2 should persist
- T3, T4 should be aborted



Failure Types

Transaction failures

- Transaction aborts

System failures  Focus of database research

- All volatile states lost

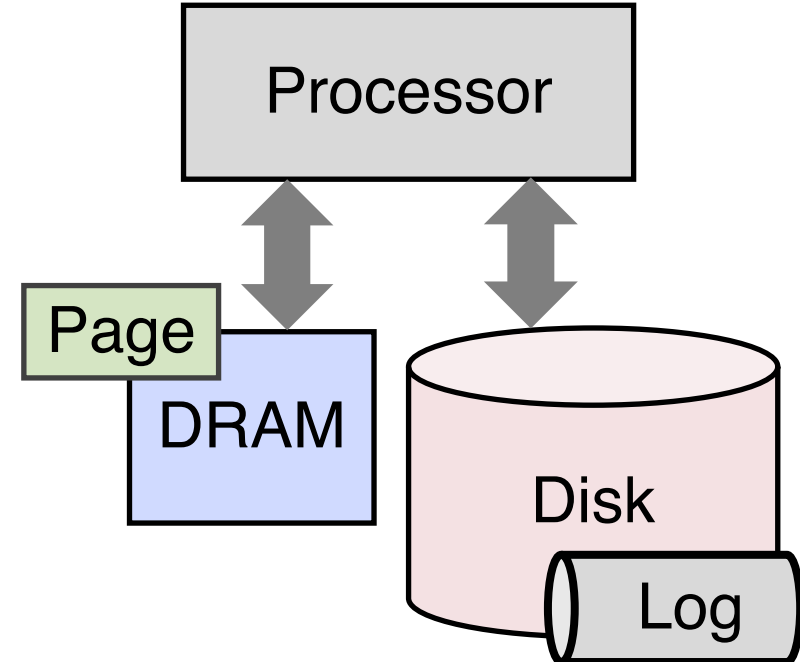
Media failures

- Some persistent states lost

Write-Ahead Logging (WAL)

Before a transaction commits, its modifications must persist

Before writing dirty data to disk, rollback information must persist



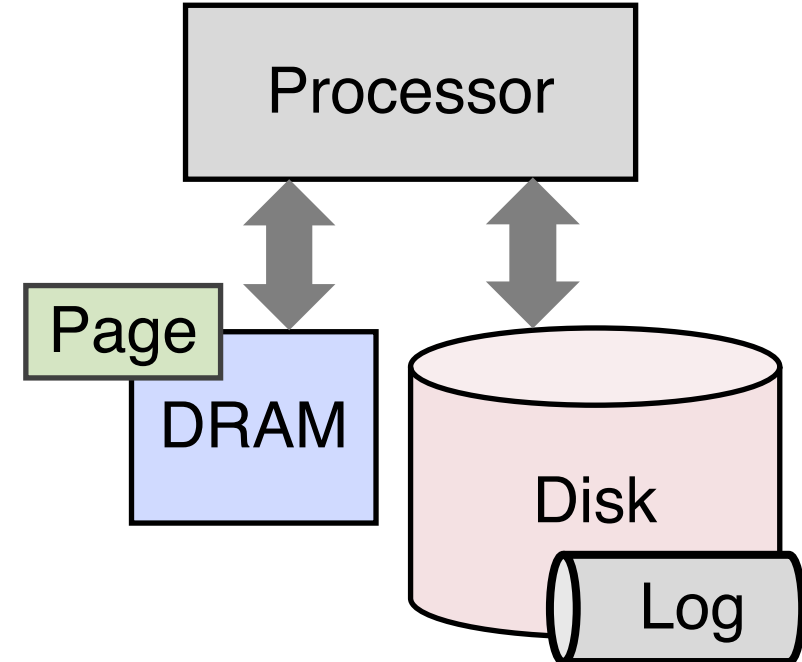
Write-Ahead Logging (WAL)

Before a transaction commits, its modifications must persist

Before writing dirty data to disk, rollback information must persist

Write-ahead logging: changes are written to the log before updating the database tables

- Writing to log incurs sequential IO



Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

Steal: Dirty pages can be flushed to disk before the transaction commits

- Advantage: other transactions can use the buffer slot in DRAM
- Challenge: system crashes after flushing dirty pages but before the transaction commits
=> Dirty data on disk
- Solution: **UNDO logging** before each update

Buffer Management Policy

Force: All dirty pages must be flushed when the transaction commits

Buffer Management Policy

Force: All dirty pages must be flushed when the transaction commits

No Force: Dirty pages may stay in memory after the transaction commits

- Advantage: reduce # random IO
- Challenge: system crashes after the transaction commits but before the dirty pages are flushed
 - => missing updates from committed transactions
- Solution: **REDO logging** before each update

Buffer Management Policy

	Steal	No Steal
Force	UNDO only	No REDO nor UNDO
No Force	REDO and UNDO logging (ARIES)	REDO only

Buffer Management Policy

	Steal	No Steal
Force	UNDO only	No REDO nor UNDO
No Force	REDO and UNDO logging (ARIES)	REDO only

Disk-based DB

Buffer Management Policy

	Steal	No Steal
Force	UNDO only	No REDO nor UNDO
No Force	REDO and UNDO logging (ARIES)	REDO only

Disk-based DB **Main memory DB**

Buffer Management Policy

	Steal	No Steal
Force	UNDO only	No REDO nor UNDO
No Force	REDO and UNDO logging (ARIES)	REDO only

Non-volatile memory DB

Disk-based DB

Main memory DB

REDO Only (no-force + no-steal)

Example: main memory database (e.g., Silo)

- **NO STEAL**: Memory is large enough to hold working set of transactions
- **NO FORCE**: Disk contains only the **checkpoint** and the **log**

REDO Only (no-force + no-steal)

Example: main memory database (e.g., Silo)

- **NO STEAL**: Memory is large enough to hold working set of transactions
- **NO FORCE**: Disk contains only the **checkpoint** and the **log**

Forward processing: Flush REDO log records to disk before commit

Recovery: Replay the log since the last checkpoint

Checkpoint: Write a consistent snapshot to disk

REDO Only — Extension

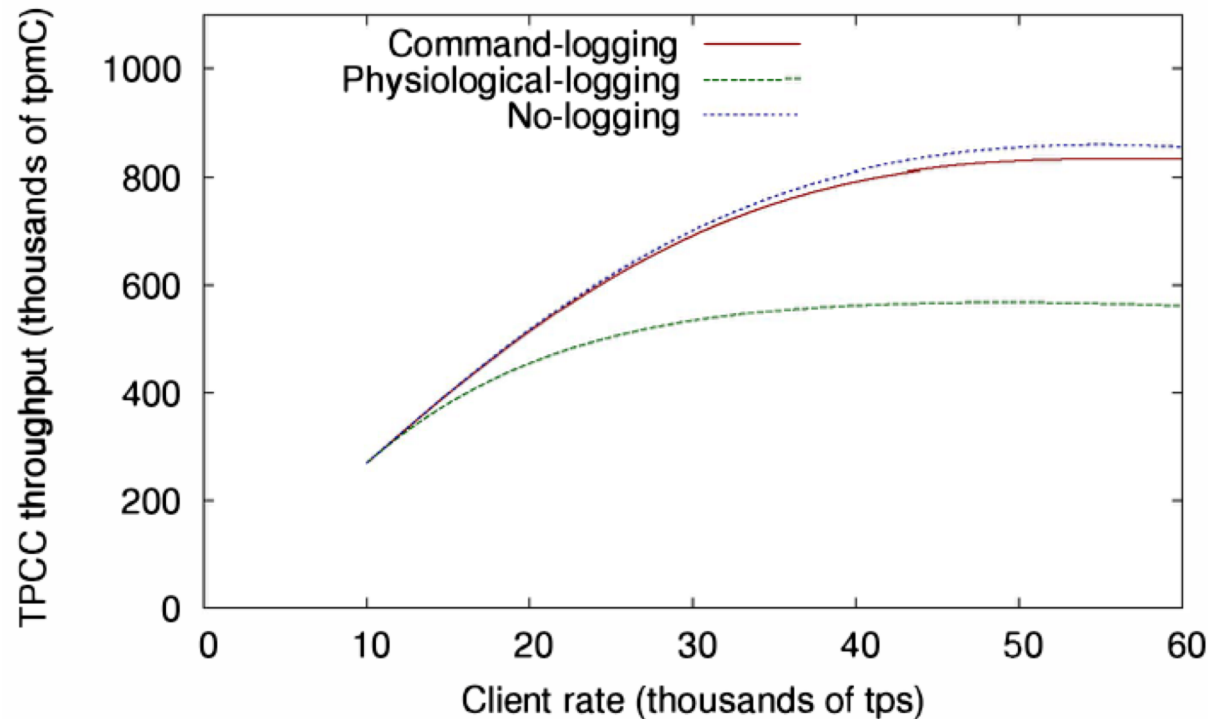
Command logging

- Log commands of transactions (much smaller than the data logging)
- Recovery reruns the transactions in-order

REDO Only — Extension

Command logging

- Log commands of transactions (much smaller than the data logging)
- Recovery reruns the transactions in-order

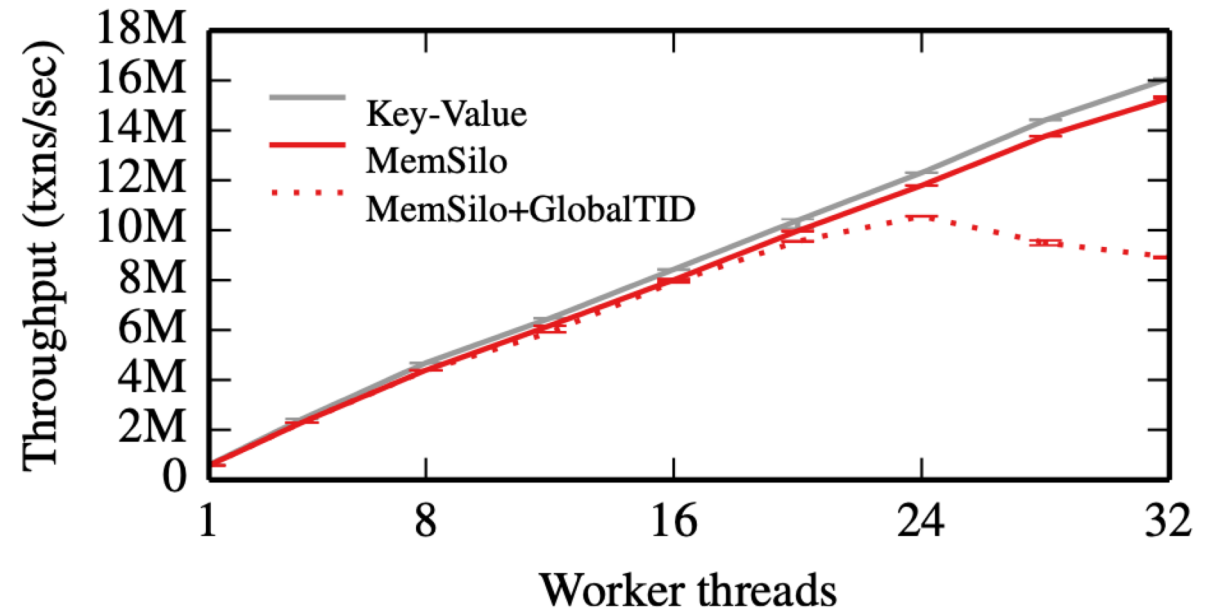


REDO Only — Extension

Command logging

Parallel logging (Silo)

- Support multiple log streams
- Epoch-based commit
- Write versioned records to log



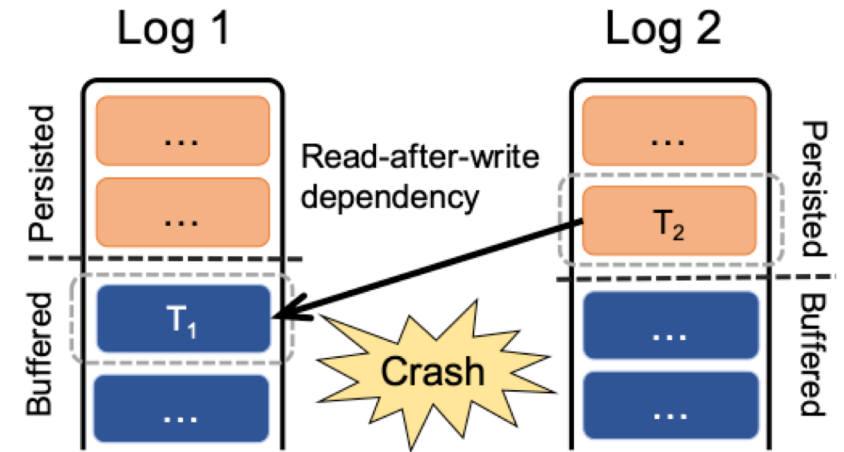
REDO Only — Extension

Command logging

Parallel logging (Silo)

Generalized parallel logging (Taurus)

- **Challenge 1:** When to commit?
(cannot commit after being persistent)



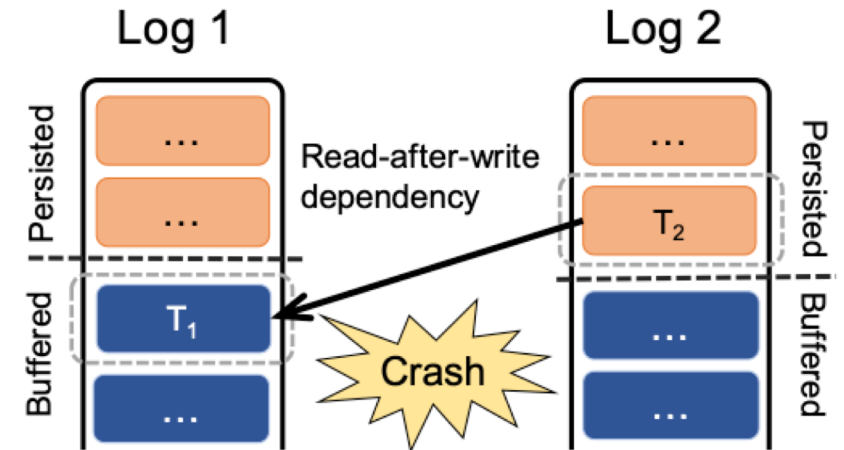
REDO Only — Extension

Command logging

Parallel logging (Silo)

Generalized parallel logging (Taurus)

- **Challenge 1:** When to commit? (cannot commit after being persistent)
- **Challenge 2:** Whether to recover? (Not all persistent transactions have committed)



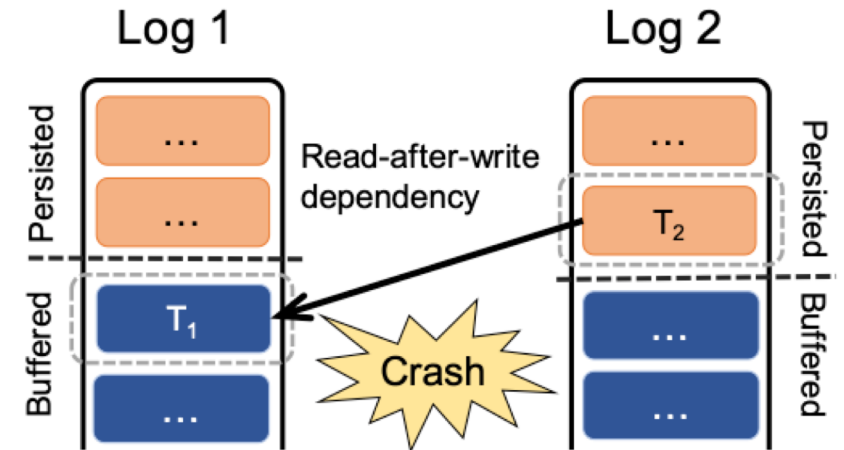
REDO Only — Extension

Command logging

Parallel logging (Silo)

Generalized parallel logging (Taurus)

- **Challenge 1:** When to commit?
(cannot commit after being persistent)
- **Challenge 2:** Whether to recover? (Not all persistent transactions have committed)
- **Challenge 3:** How to determine the right recovery order?



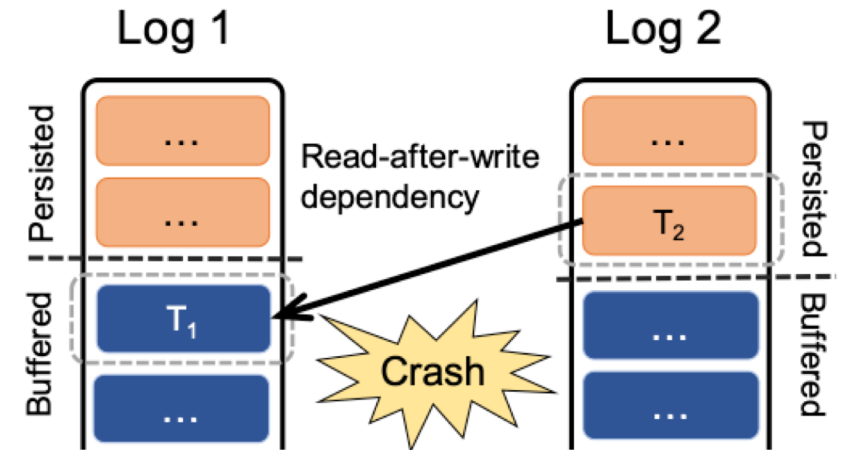
REDO Only — Extension

Command logging

Parallel logging (Silo)

Generalized parallel logging (Taurus)

- **Challenge 1:** When to commit?
(cannot commit after being persistent)
- **Challenge 2:** Whether to recover? (Not all persistent transactions have committed)
- **Challenge 3:** How to determine the right recovery order?
- **Key idea:** maintain ordering using vector clock



REDO Only — Extension

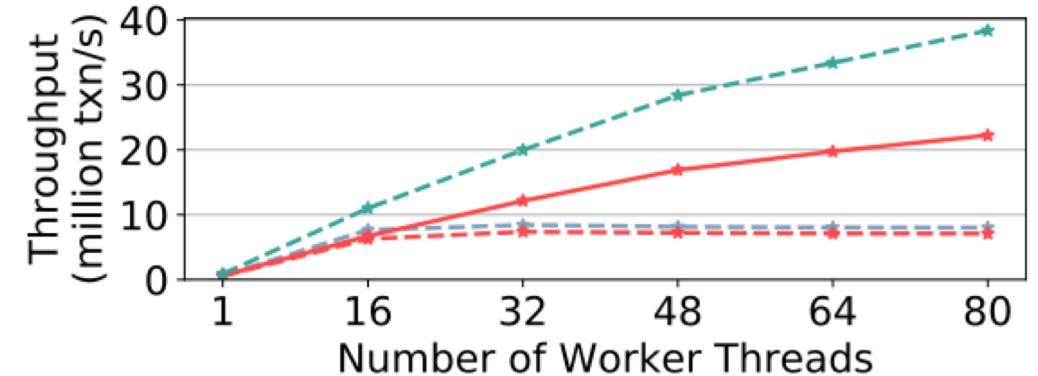
Command logging

Parallel logging (Silo)

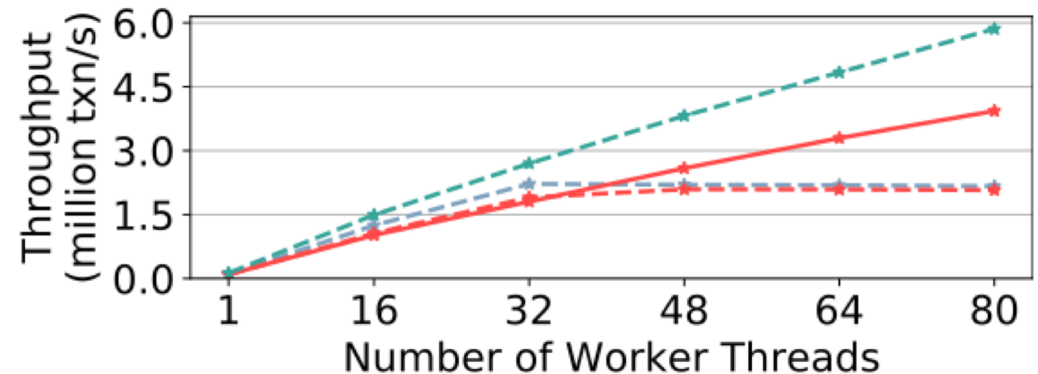
Generalized parallel logging (Taurus)

- **Challenge 1:** When to commit? (cannot commit after being persistent)
- **Challenge 2:** Whether to recover? (Not all persistent transactions have committed)
- **Challenge 3:** How to determine the right recovery order?
- **Key idea:** maintain ordering using vector clock

—+— No Logging —+— SiloR Data —+— Taurus Command —+— Taurus Data



(b) TPC-C Payment



(c) TPC-C New-Order

UNDO Only (force + steal)

Example: NVM database, data replication to another node

- **STEAL**: In-place updates to NVM or backup node cannot be executed atomically
- **FORCE**: NVM or backup DRAM is fast enough for random writes

UNDO Only (force + steal)

Example: NVM database, data replication to another node

- **STEAL**: In-place updates to NVM or backup node cannot be executed atomically
- **FORCE**: NVM or backup DRAM is fast enough for random writes

Forward processing: Flush UNDO log records before updating records in the tables; commit after all records are updated

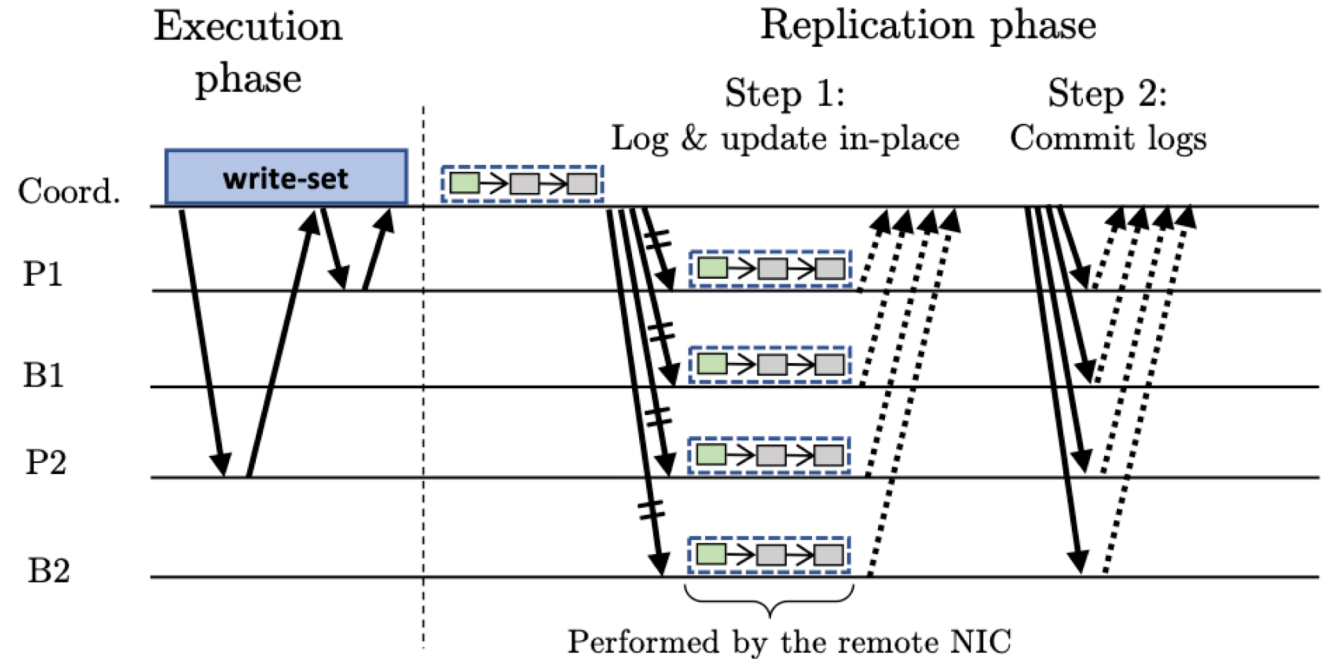
- The UNDO log size can be bounded

Recovery: Rollback uncommitted transactions

UNDO Only Example

Update states in backup node

- Use one-sided RDMA to avoid CPU computation in backup node
- Primary sends **undo records** and **in-place updates** which are applied in-order



No UNDO and No REDO (force + no-steal)

Example: NVM database

- **NO STEAL**: Main memory large enough to hold working set of transactions
- **FORCE**: NVM or backup DRAM is fast enough for random writes

Forward processing: Must ensure that all updates of a transaction are performed using an atomic operation

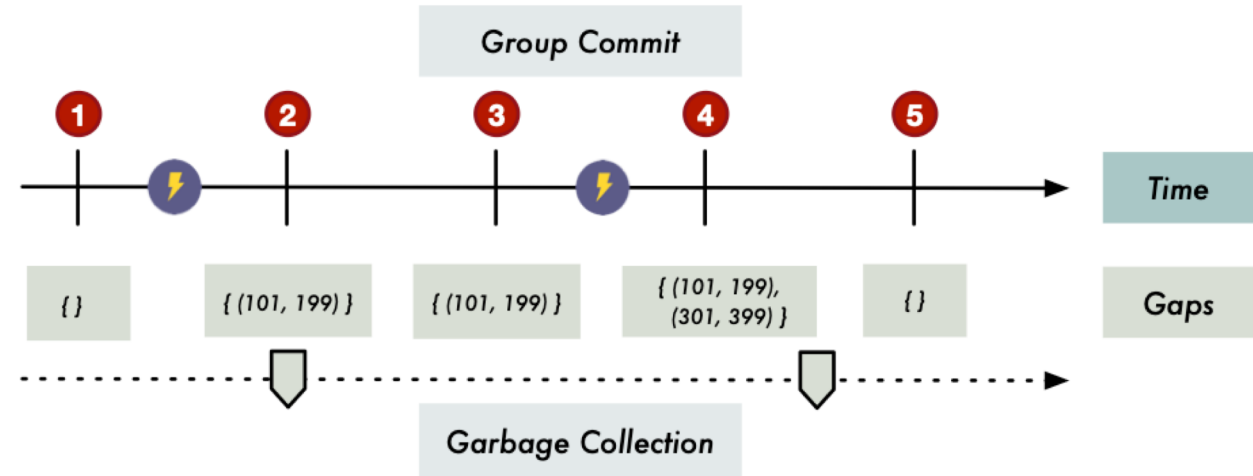
No UNDO and No REDO Example

Multi-version database

- No in-place update
- Each version has a timestamp

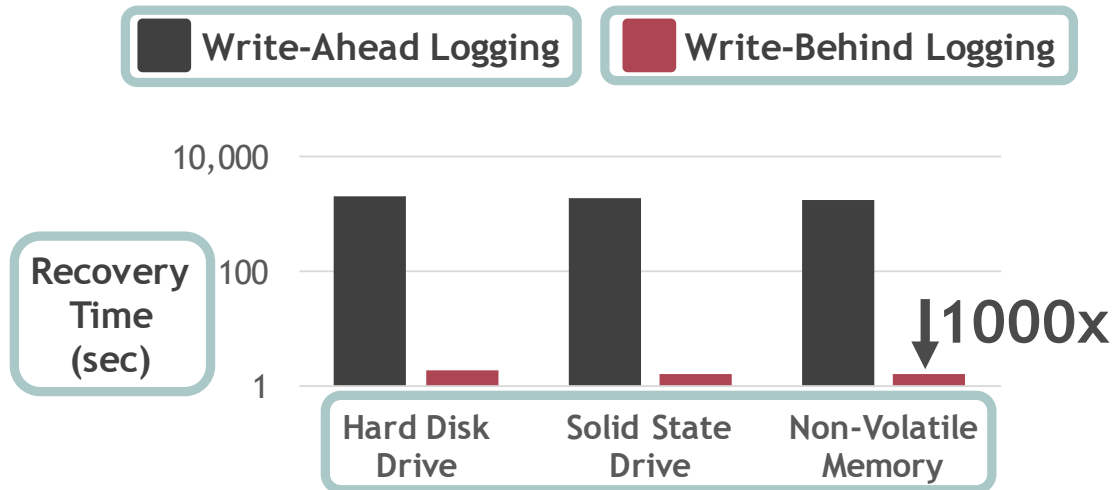
Group commit

- A single log record of a timestamp range (c_p , c_d)
- Transactions before c_p commit
- DBMS does not assign timestamps larger than c_d before next group commit



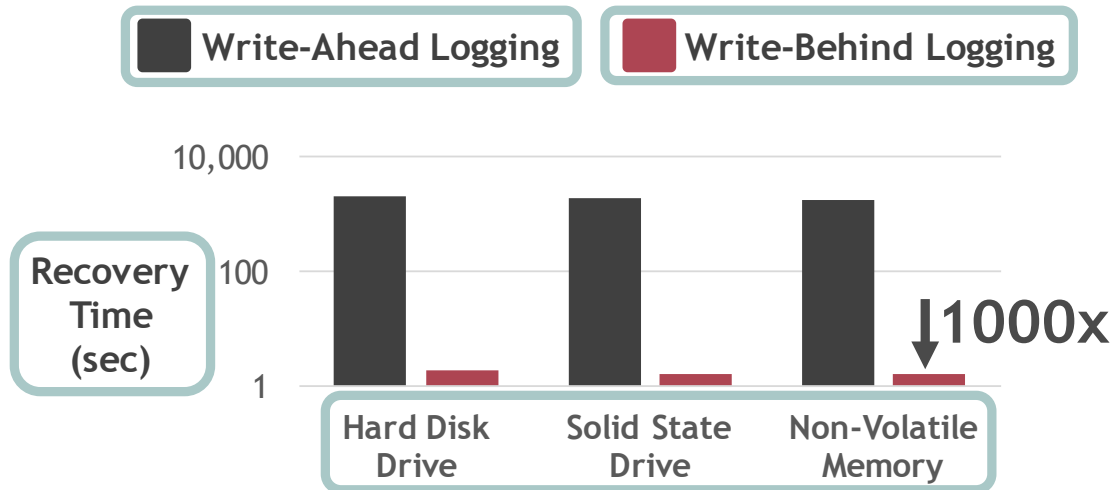
No UNDO and No REDO Example

APPLICATION AVAILABILITY

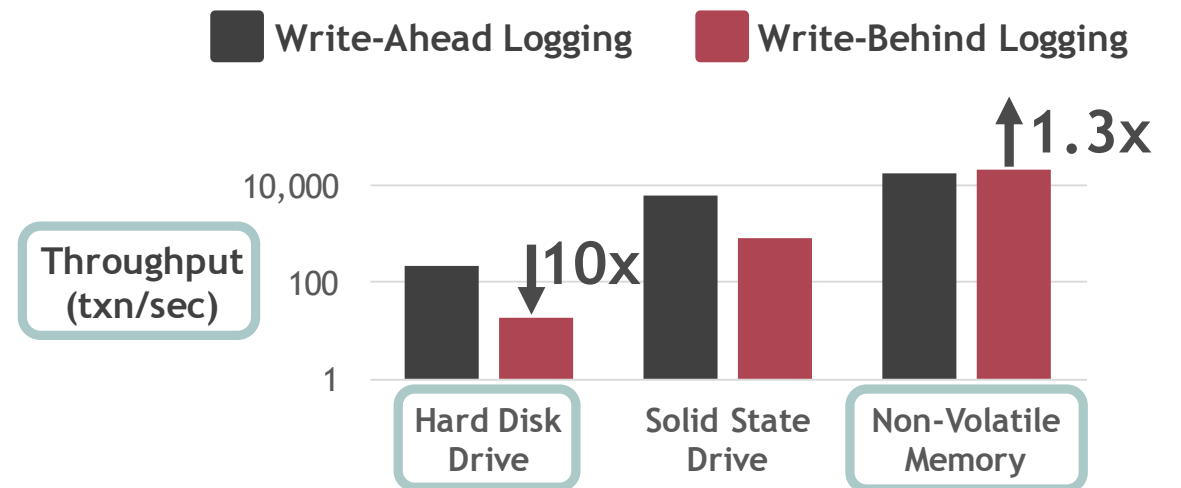


No UNDO and No REDO Example

APPLICATION AVAILABILITY



PERFORMANCE



REDO and UNDO

Example: Disk-based database

- **STEAL**: Memory not large enough to hold working set of transactions (e.g., long running transactions)
- **NO FORCE**: Random writes to disk are slow

Forward processing: Flush UNDO and REDO records before writing to data pages

Recovery: ARIES (next lecture)

Checkpoint: Fuzzy checkpoint

Q/A – Aries Recovery

Which (redo/no-redo vs. undo/no-undo) is used most commonly?

Logging at transaction level instead of operation level?

How does PM change the design space?

Hybrid undo/redo and no-undo/redo?

Major improvement since 1980s?

How do these algorithms work in distributed systems?

Before Next Lecture

Submit review before next lecture

- C. Mohan, et al. [ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging](#). ACM Transactions on Database Systems, 1992
- Can skip Section 1 and 2 and everything after (including) Section 8
- About 25–30 pages to read