Recovery

CSC443H1 Database System Technology

Niv Dayan

Last stretch for the project!

(Deadline on Dec 1st)

TA & Normal Office this Thursday
In both BA5233 and BA5230

3-4 pm

6–7 pm

Last lecture today

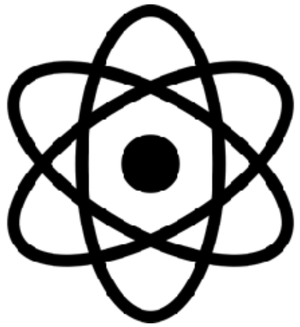
Last lecture today

One hour lecture followed by tutorial questions

Please do the course evaluation!

ACID Transactions

Atomicity



All or nothing

Consistency

Transition across consistent states

Isolation

Not corrupted by concurrency

Durability

Recover from failure

What Endangers the ACID properties?

Concurrency

System Failure

What Endangers the ACID properties?

Concurrency

Last week

System Failure

Today

System Failure

Media failure

Power failure

Data center failure

System Failure

Media failure

RAID (covered)

Power failure

Data center failure

System Failure

Media failure

RAID (covered)

Power failure

Logging

Data center failure

System Failure

Media failure

RAID (covered)

Power failure

Logging

Data center failure

Replication

Example:
Bank transfer

A = A - 100
B = B + 100

Example:
Bank transfer

A = A - 100

Power fails

B = B + 100

If the update to account A is saved to disk, money disappears from the system

Example:
Bank transfer

A = A - 100

Power fails

B = B + 100

If the update to account A is saved to disk,
money disappears from the system

Example:
Bank transfer

A = A - 100

Power fails

B = B + 100

What's a high level solution?

Solution outline: (1) wrap this up as a transaction

Start transaction

A = A - 100

B = B + 100

End transaction

Solution outline: (1) wrap this up as a transaction

(2) write changes to by each transaction in a log

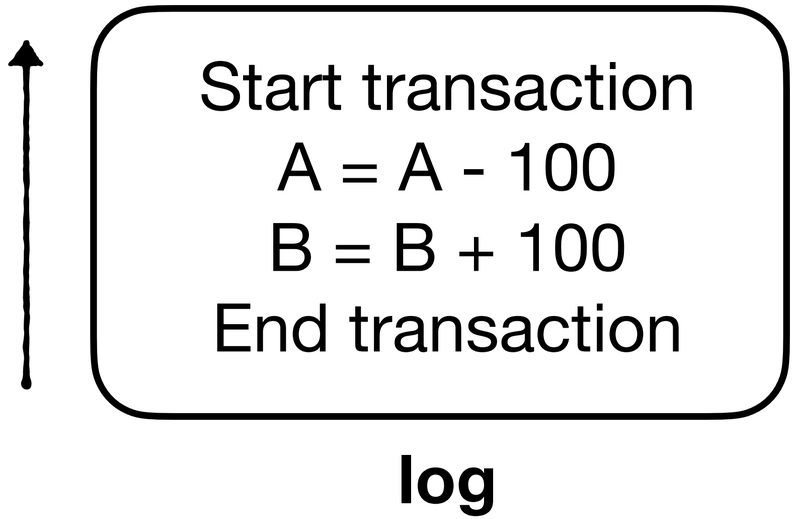
Start transaction
A = A - 100
B = B + 100
End transaction

log: an append-only file in storage

Solution outline: (1) wrap this up as a transaction

(2) write changes to by each transaction in a log

(3) after power fails, scan log and cancel effects of uncommitted transactions

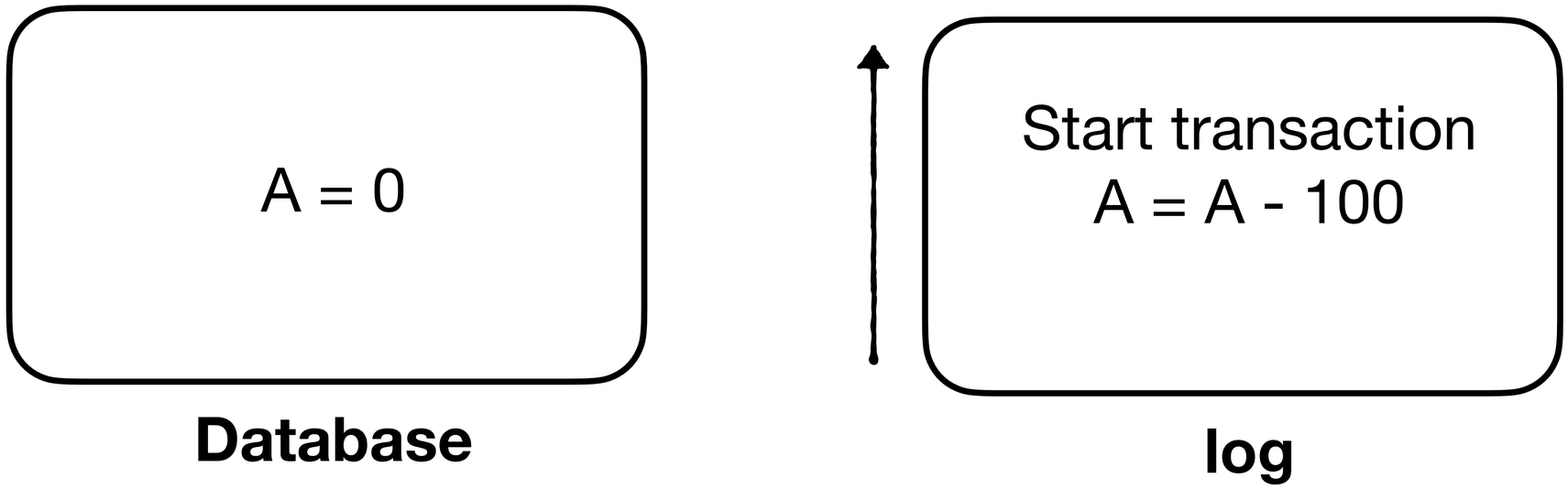


log

Solution outline: (1) wrap this up as a transaction

(2) write changes to by each transaction in a log

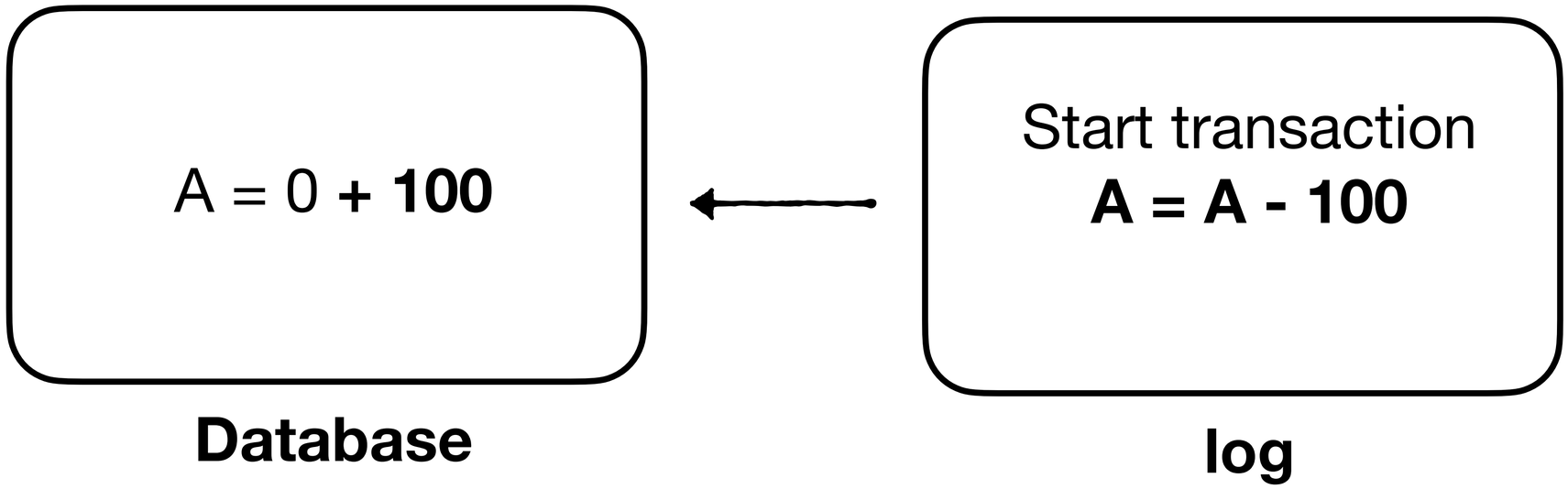
(3) after power fails, scan log and cancel effects of uncommitted transactions



Solution outline: (1) wrap this up as a transaction

(2) write changes to by each transaction in a log

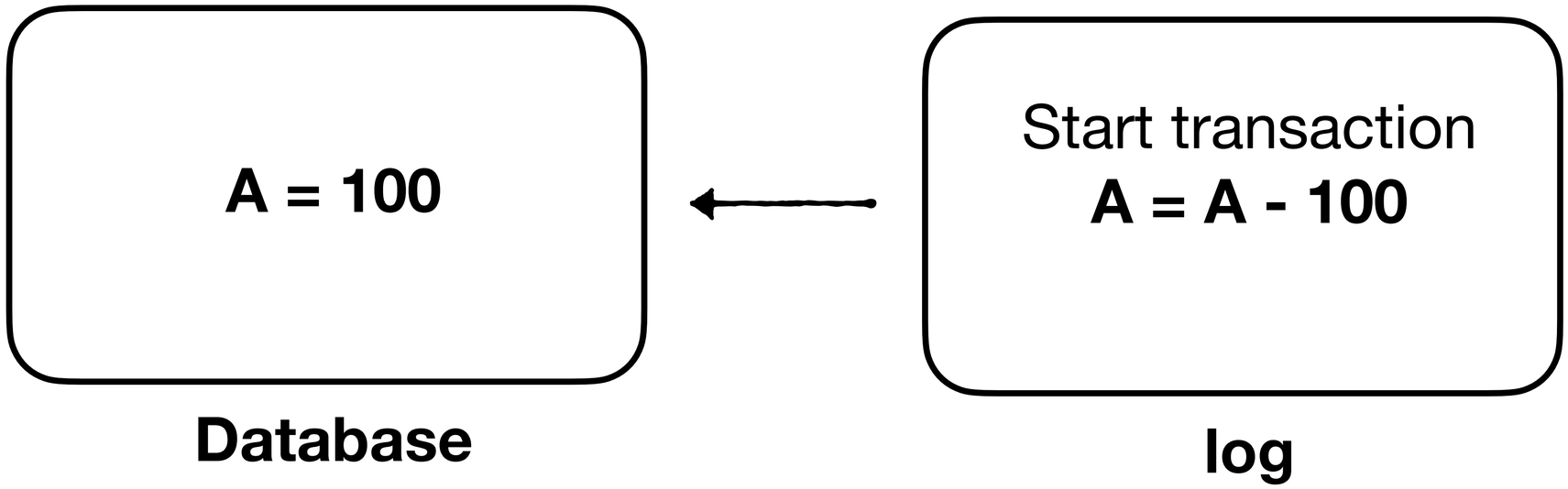
(3) after power fails, scan log and cancel effects of uncommitted transactions



Solution outline: (1) wrap this up as a transaction

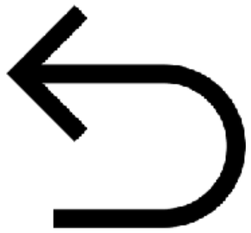
(2) write changes to by each transaction in a log

(3) after power fails, scan log and cancel effects of uncommitted transactions



Two Types of logging with different trade-offs

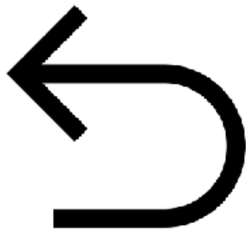
Undo



Redo

Two Types of logging with different trade-offs

Undo

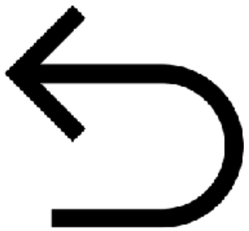


random I/Os

Redo

holds memory

Undo



random I/Os

Redo

holds memory

Redo/Redo

Addresses both problems!

Undo logging:

Invariant: If transactions is marked as committed in the log, all of its changes are persisted in storage

Undo logging:

Invariant: If transactions is marked as committed in the log, all of its changes are persisted in storage

Consequence: to recover, undo effects of uncommitted transactions

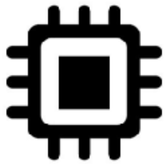
Undo logging:

Invariant: If transactions is marked as committed in the log, all of its changes are persisted in storage

Consequence: to recover, undo effects of uncommitted transactions

Corollary: no constraints on buffer pool eviction policy

Undo logging:



Buffer pool

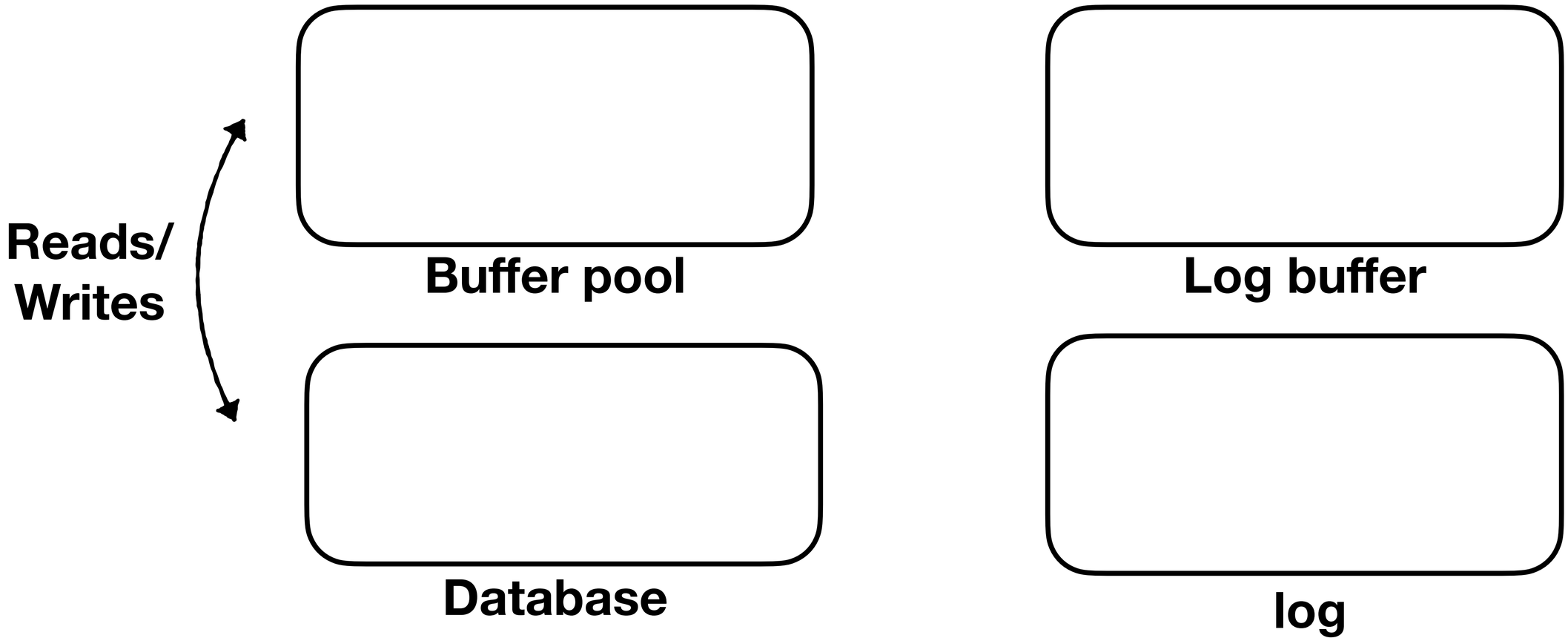
Log buffer

Database

log

Undo logging:

Assume relational database

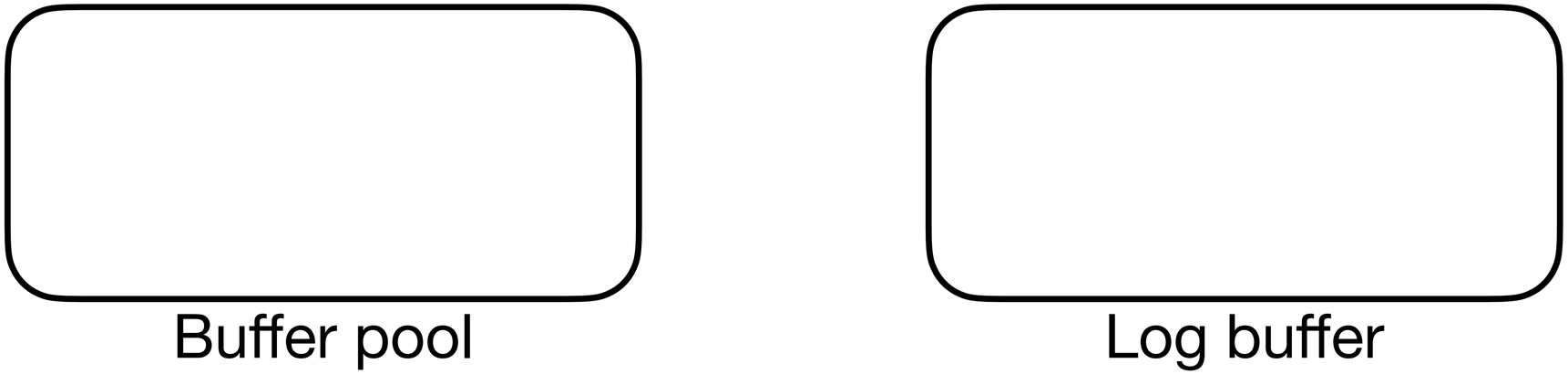


Undo logging:

Transaction T1

A = A - 100

B = B + 100

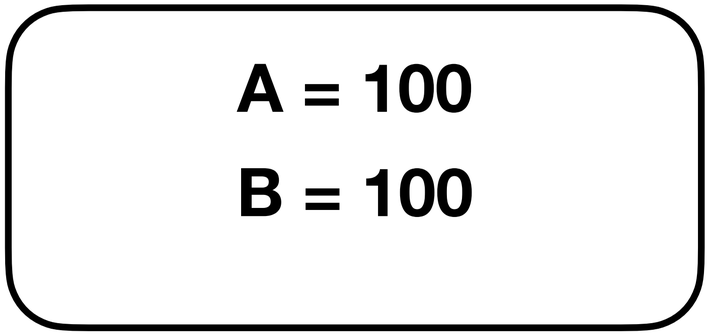


Undo logging:

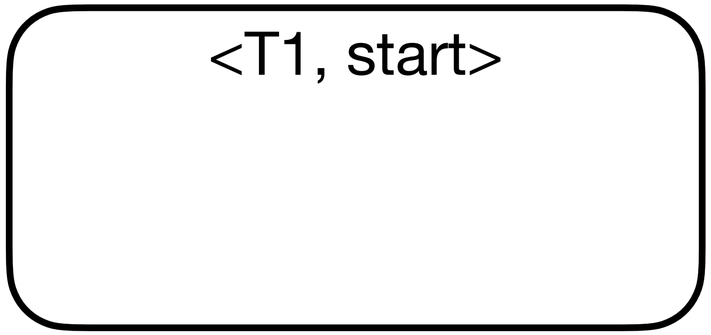
Transaction T1

A = A - 100

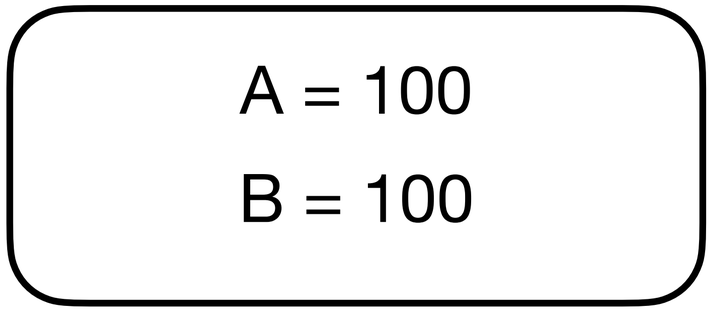
B = B + 100



Buffer pool



Log buffer



Database

log

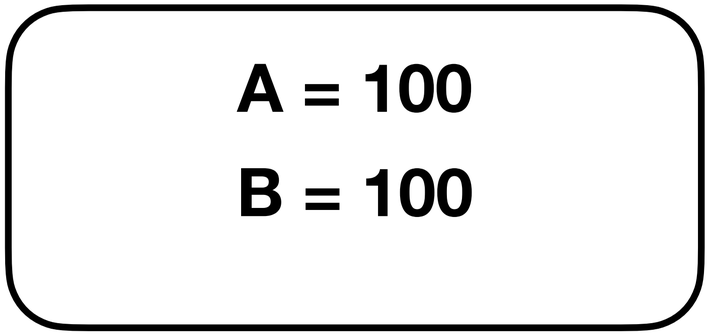
Undo logging:

Transaction T1

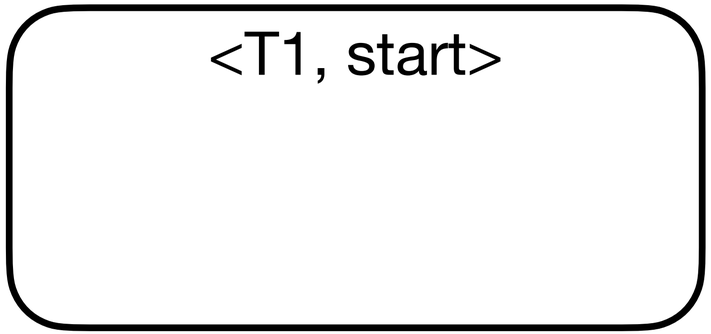
A = A - 100

B = B + 100

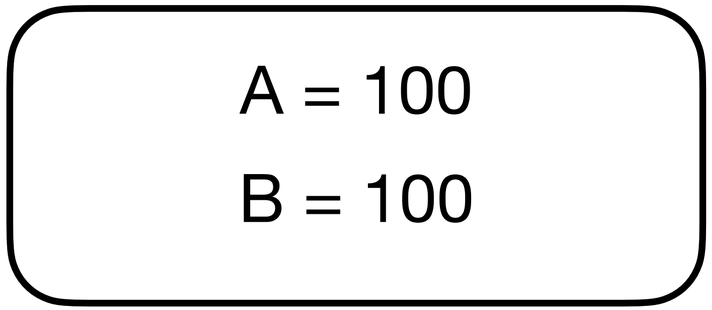
Lock both items



Buffer pool



Log buffer



Database

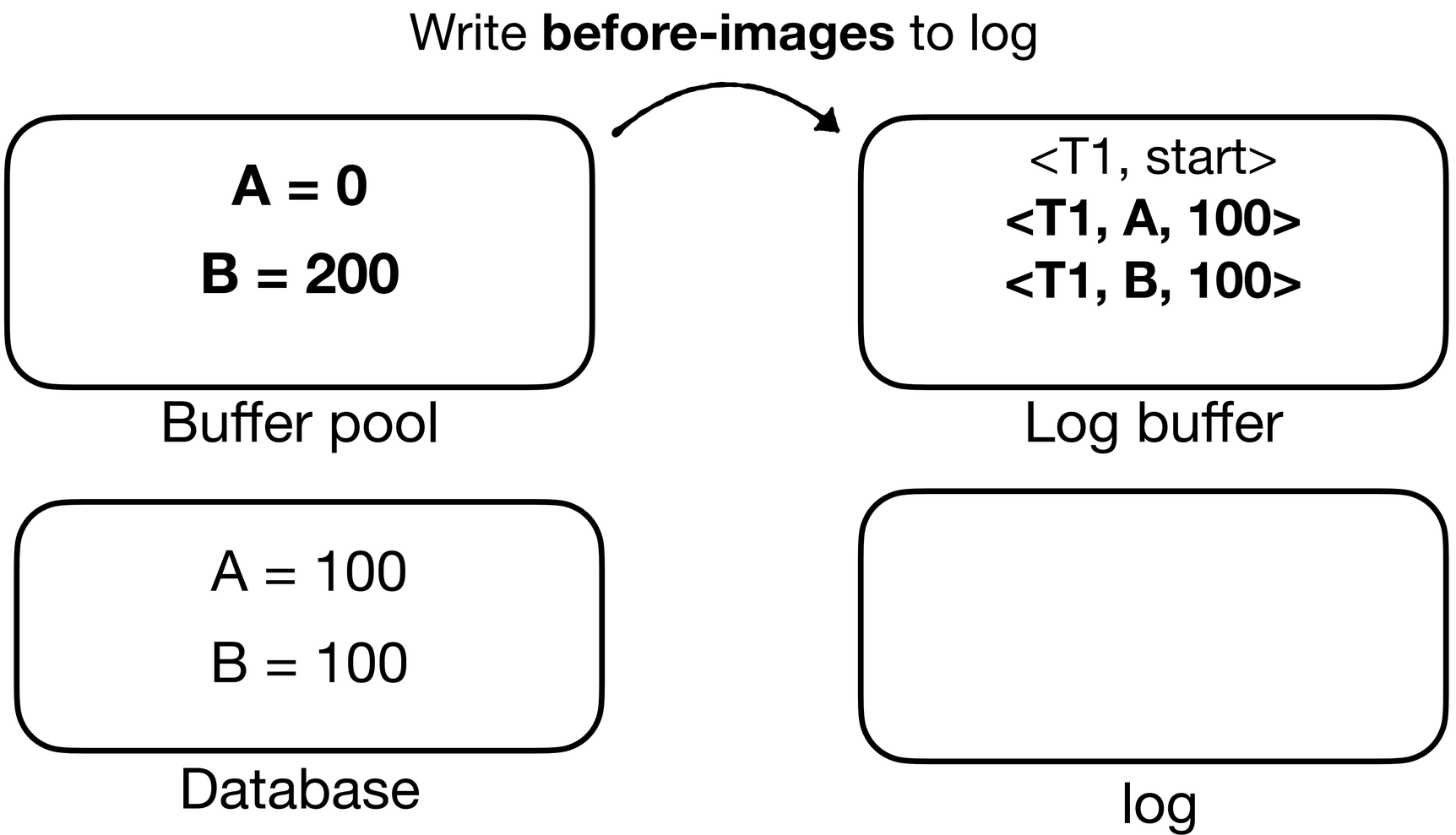
log

Undo logging:

Transaction T1

A = A - 100

B = B + 100

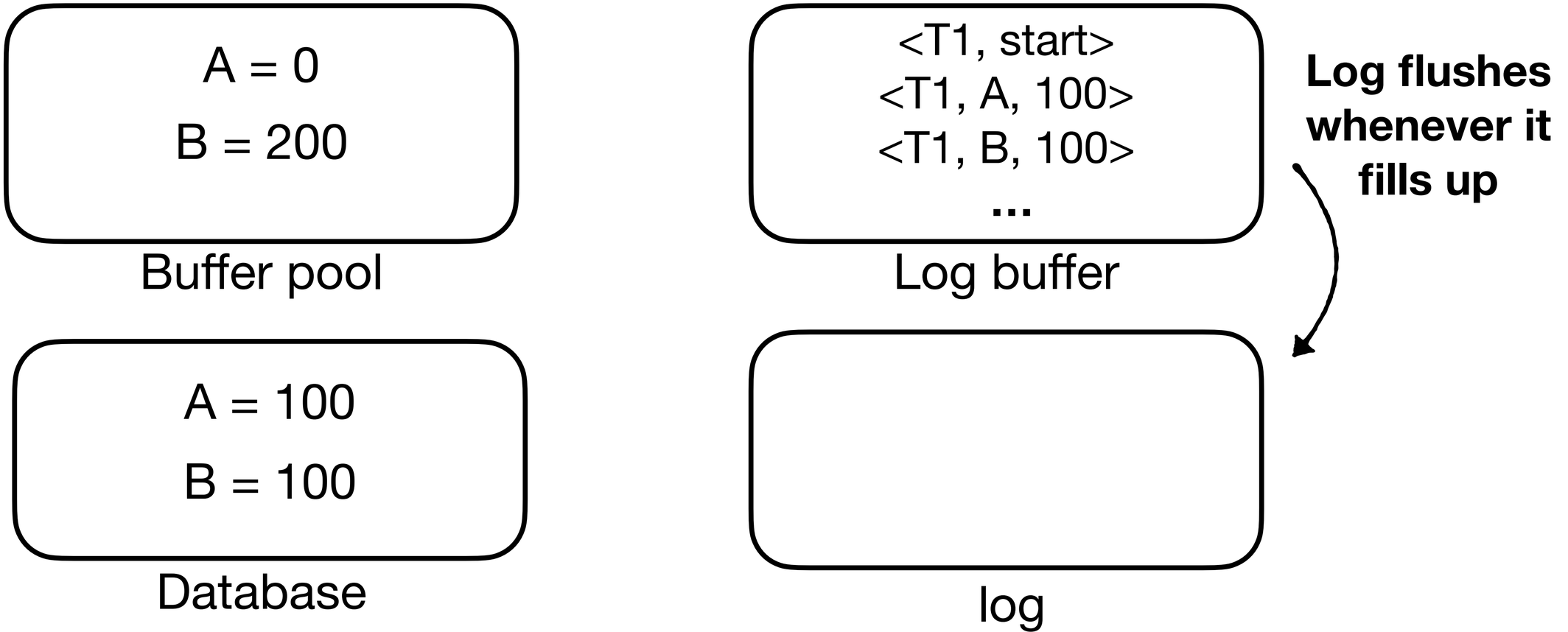


Undo logging:

Transaction T1

A = A - 100

B = B + 100



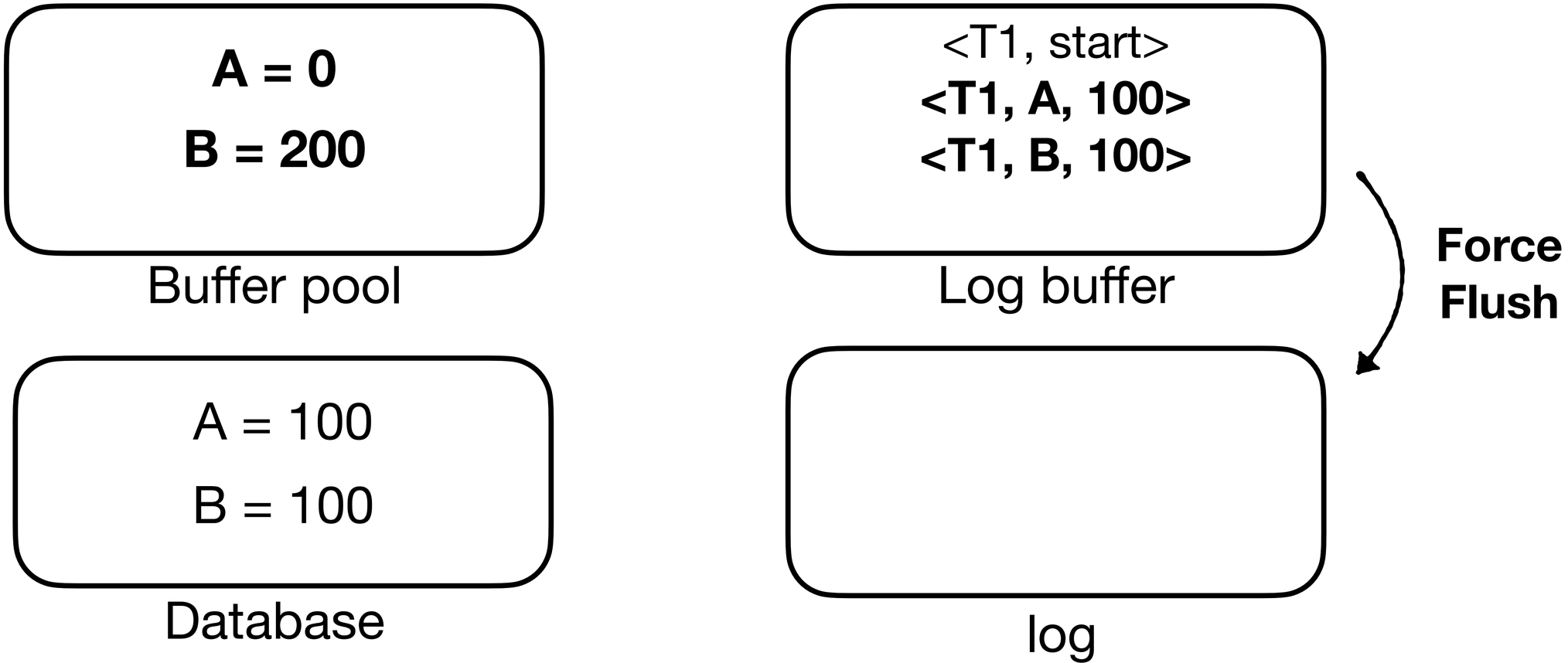
Undo logging:

Transaction T1

A = A - 100

B = B + 100

However, to commit any transaction, we force flush the log.

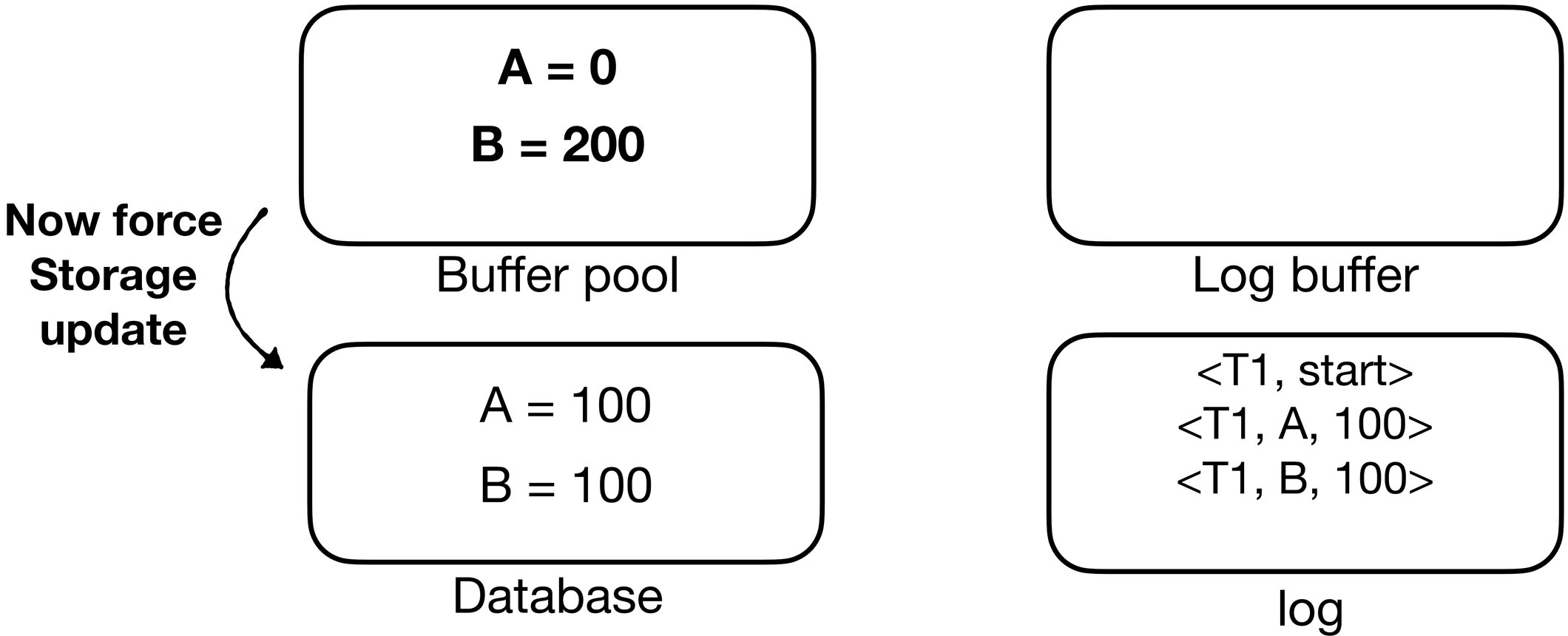


Undo logging:

Transaction T1

A = A - 100

B = B + 100

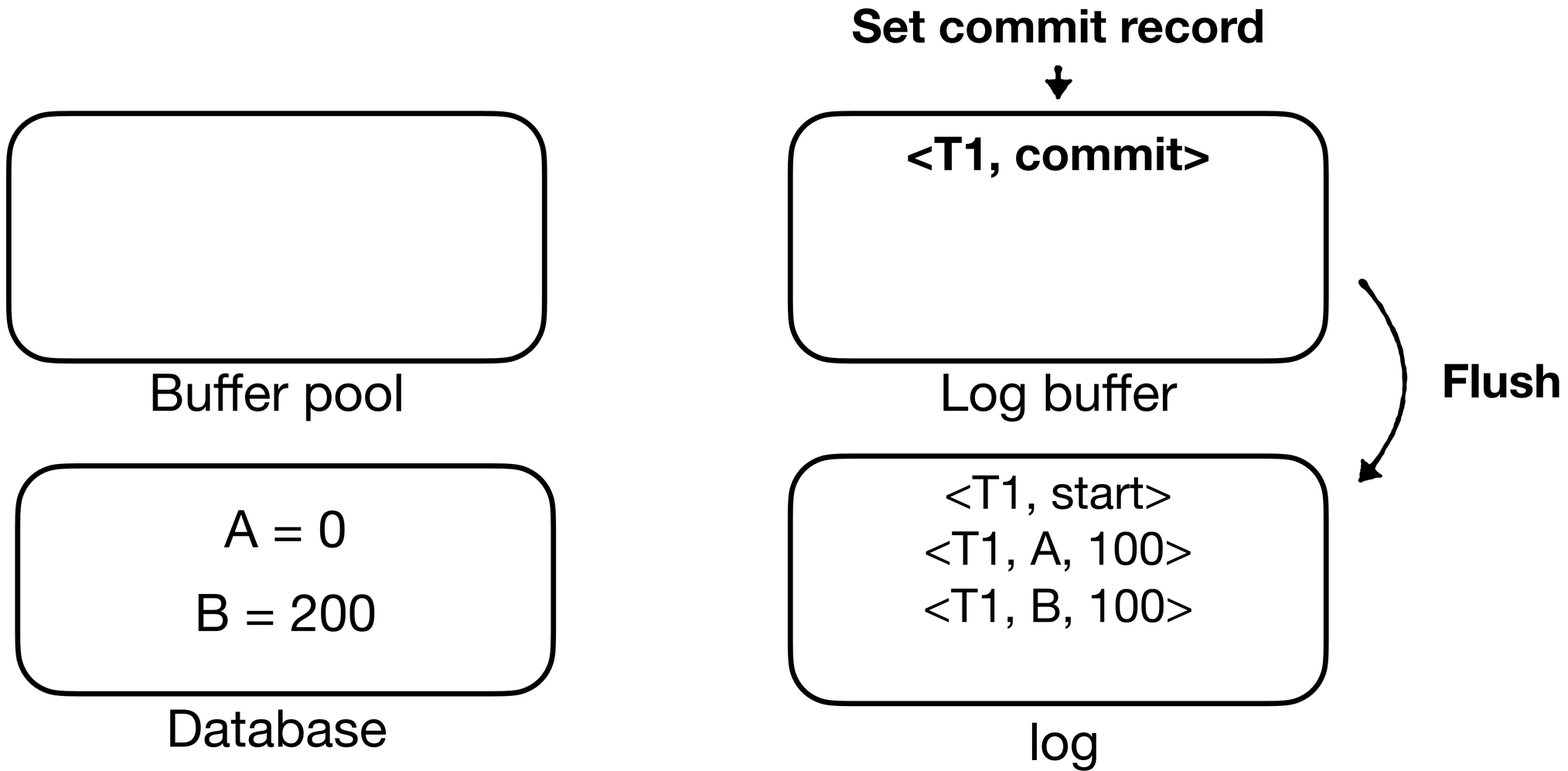


Undo logging:

Transaction T1

A = A - 100

B = B + 100



Undo logging:

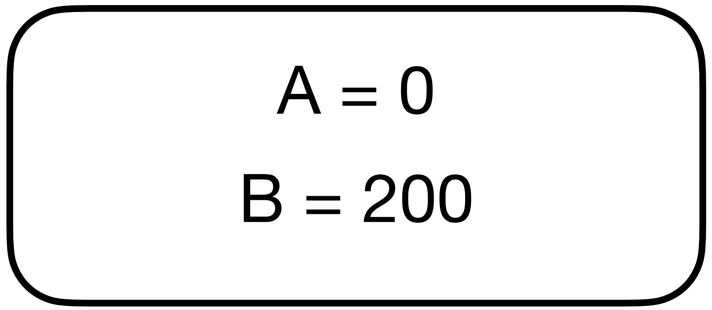
Transaction T1

A = A - 100

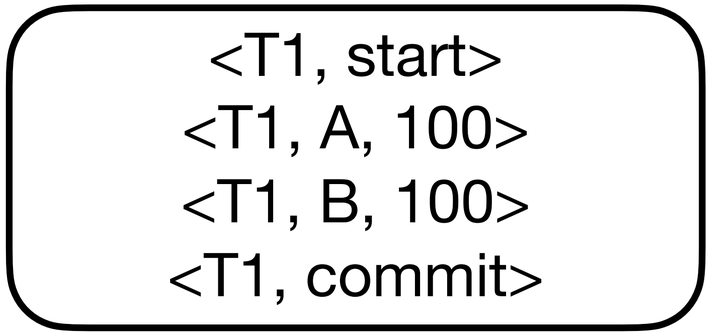
B = B + 100

Buffer pool

Log buffer



Database



log

Undo logging:

(1) write before-image for any changed value to log buffer

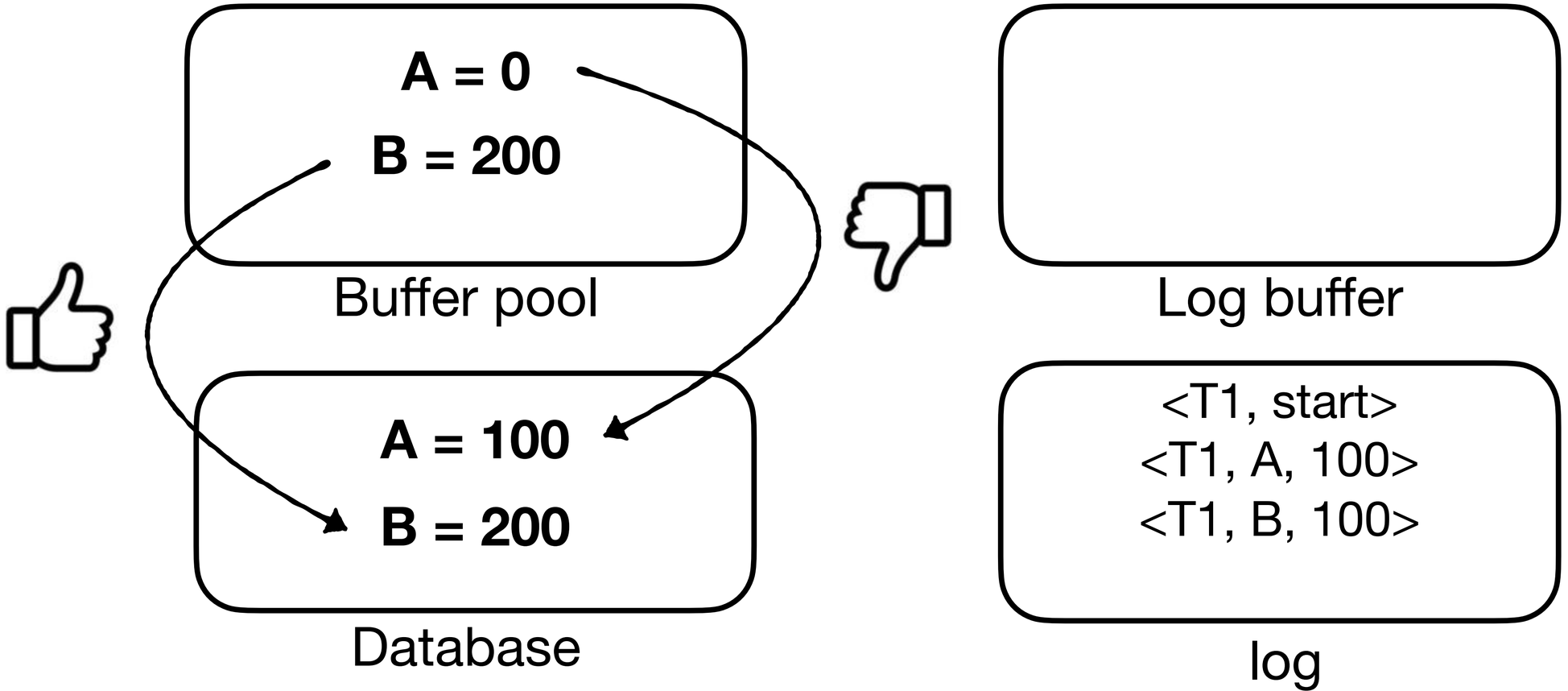
(2) force flush log

(3) force the changed data into storage

(4) add commit record to log buffer & force flush again

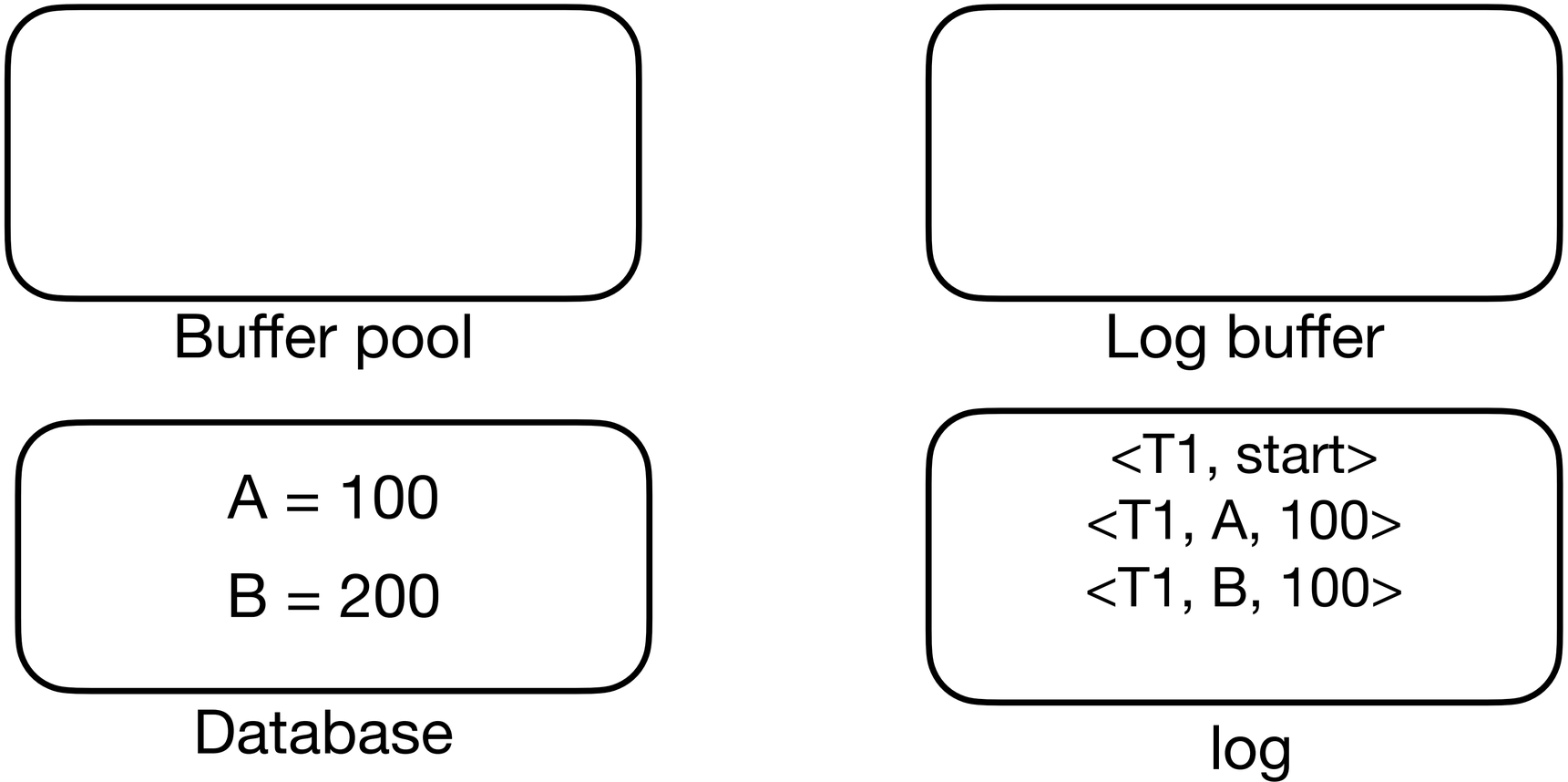
Recovery Undo logging:

Suppose power fails after we save the value of B to storage but before we save the value of A



Recovery Undo logging:

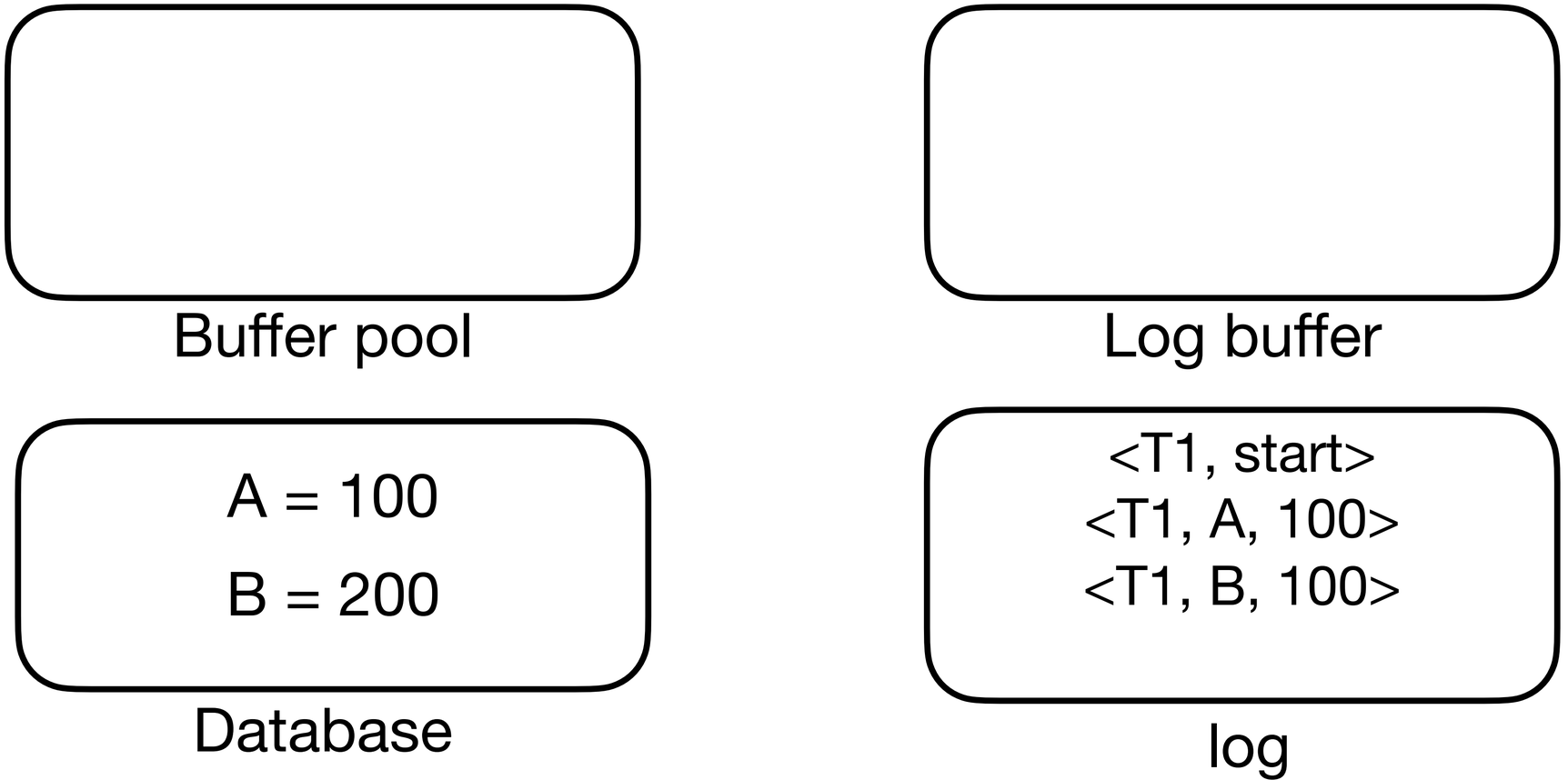
We lose all contents in memory



Recovery Undo logging:

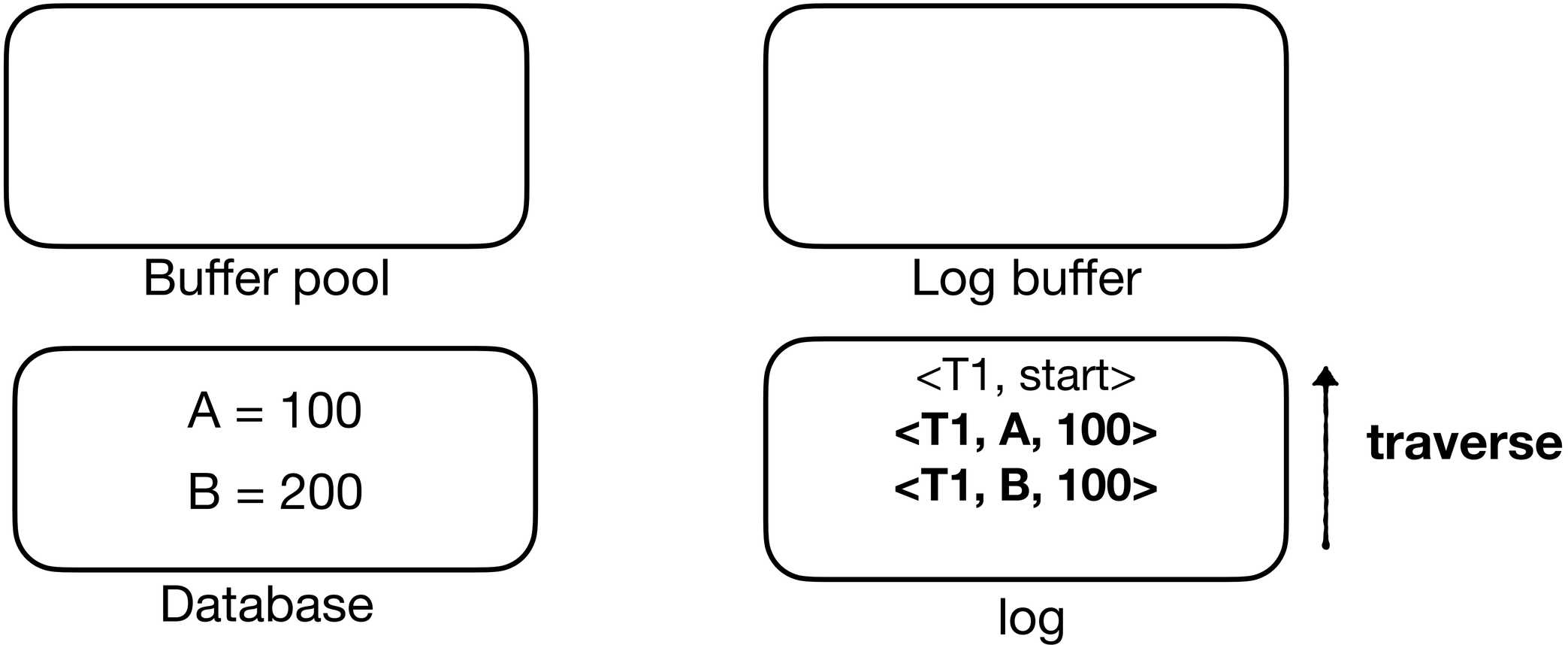
We lose all contents in memory

How do we recover?



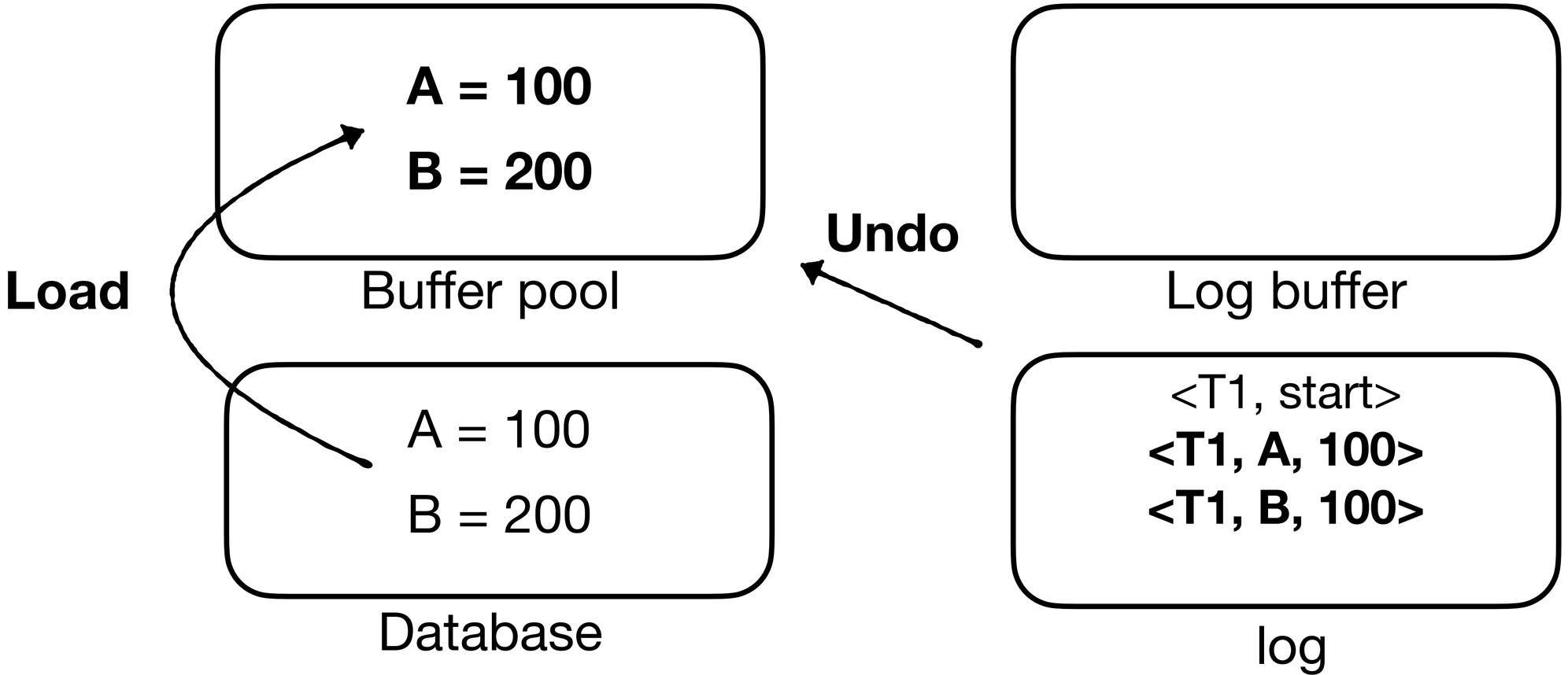
Recovery Undo logging:

During recovery, we traverse the log backwards



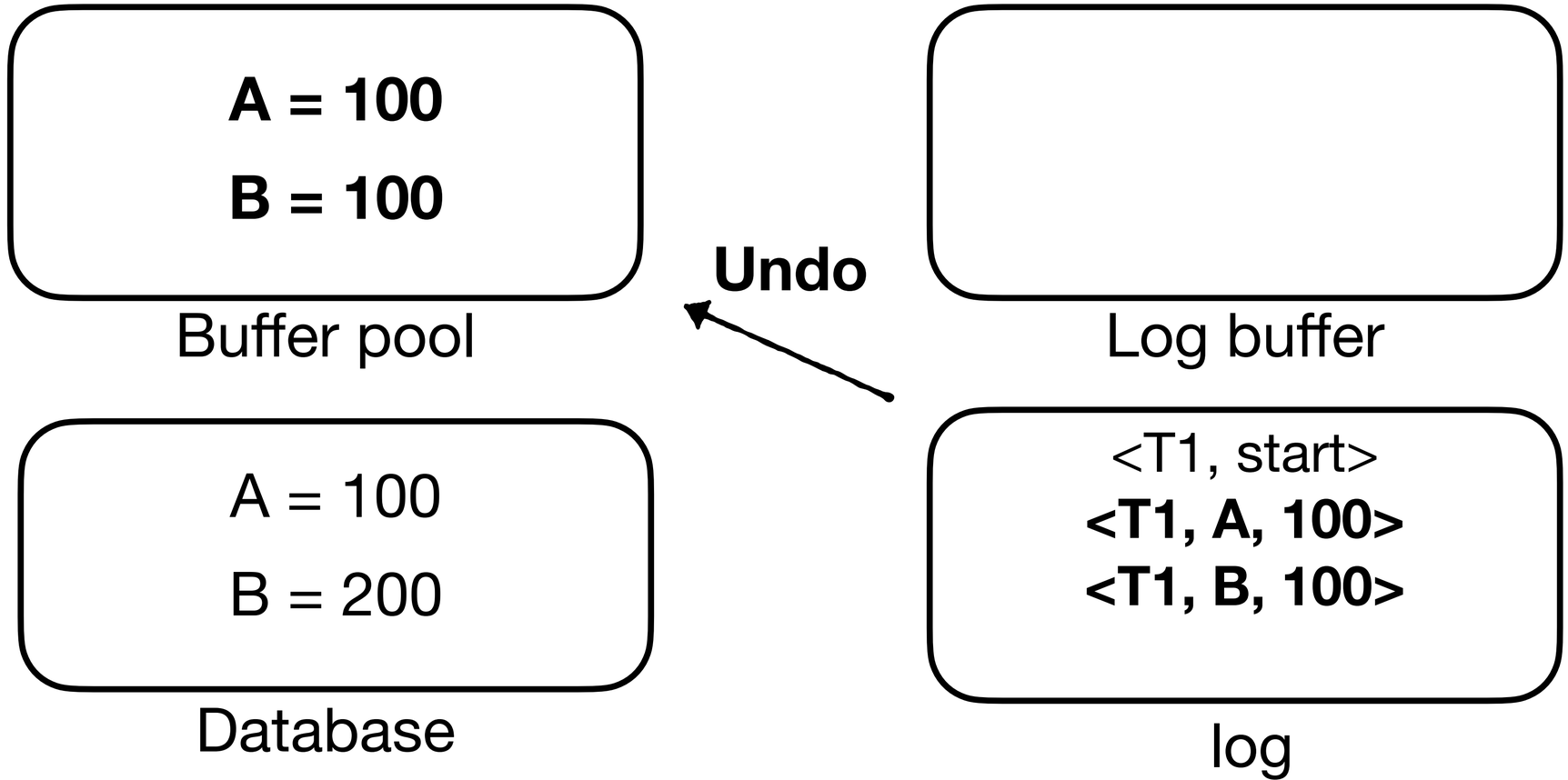
Recovery Undo logging:

Undo any operation of an uncommitted transaction



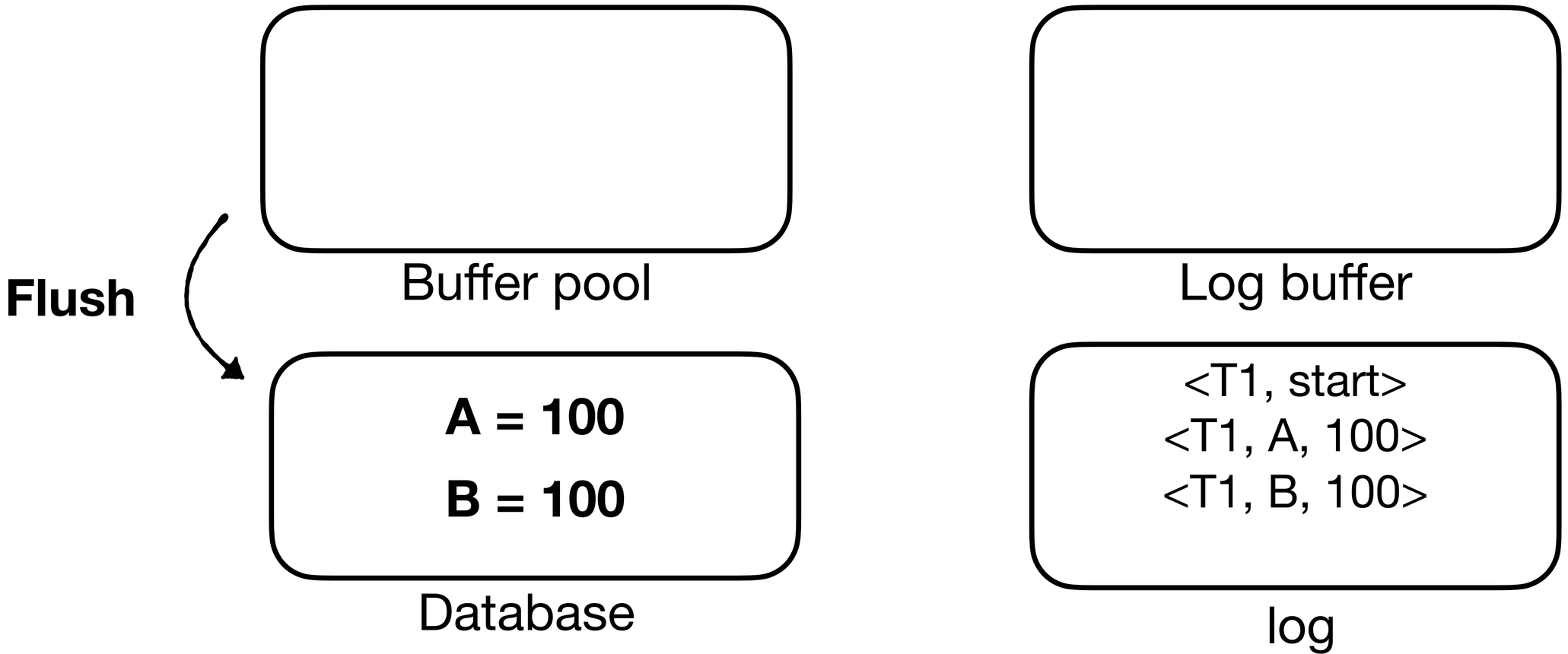
Recovery Undo logging:

Undo any operation of an uncommitted transaction

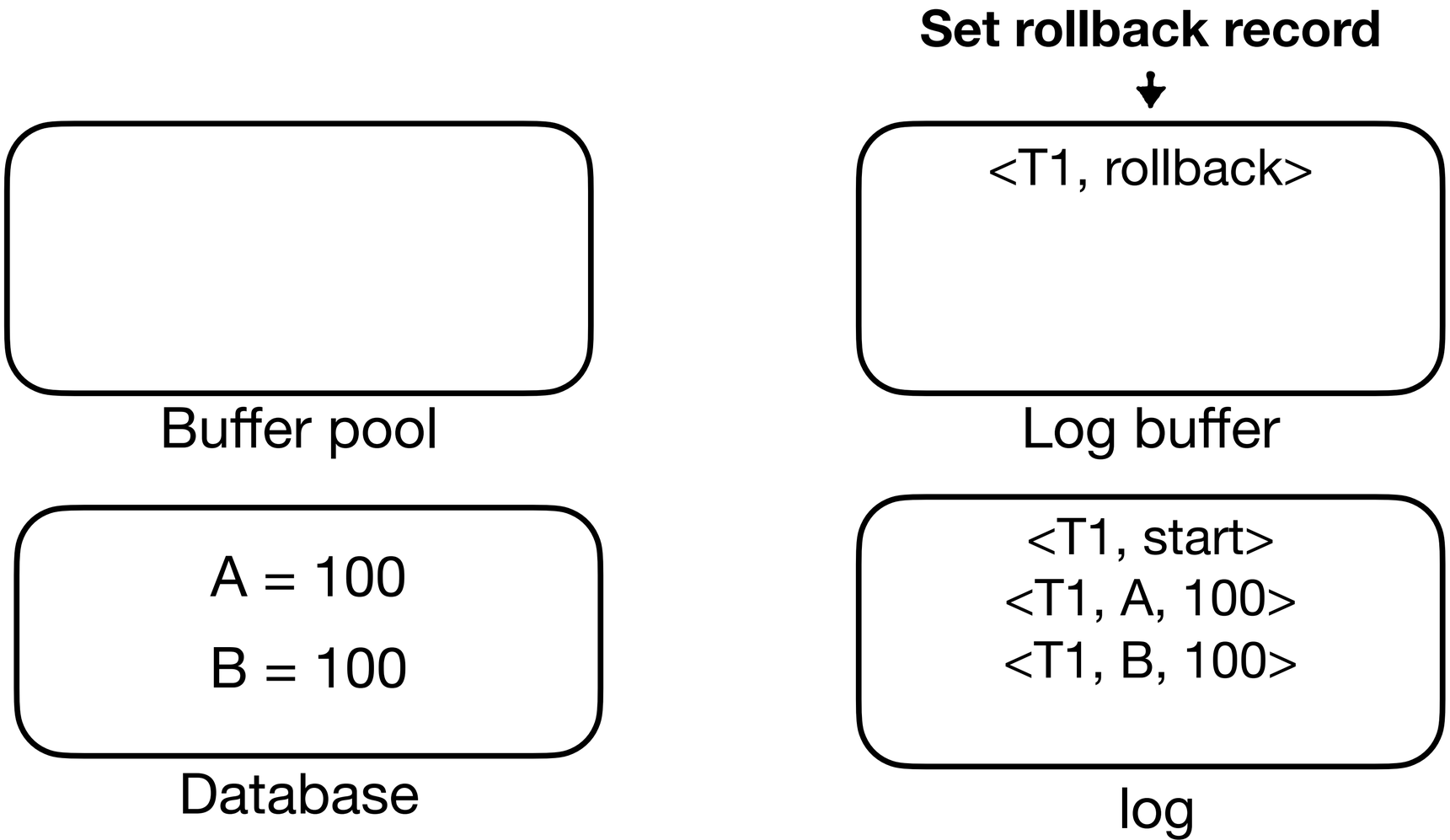


Recovery Undo logging:

Undo any operation of an uncommitted transaction

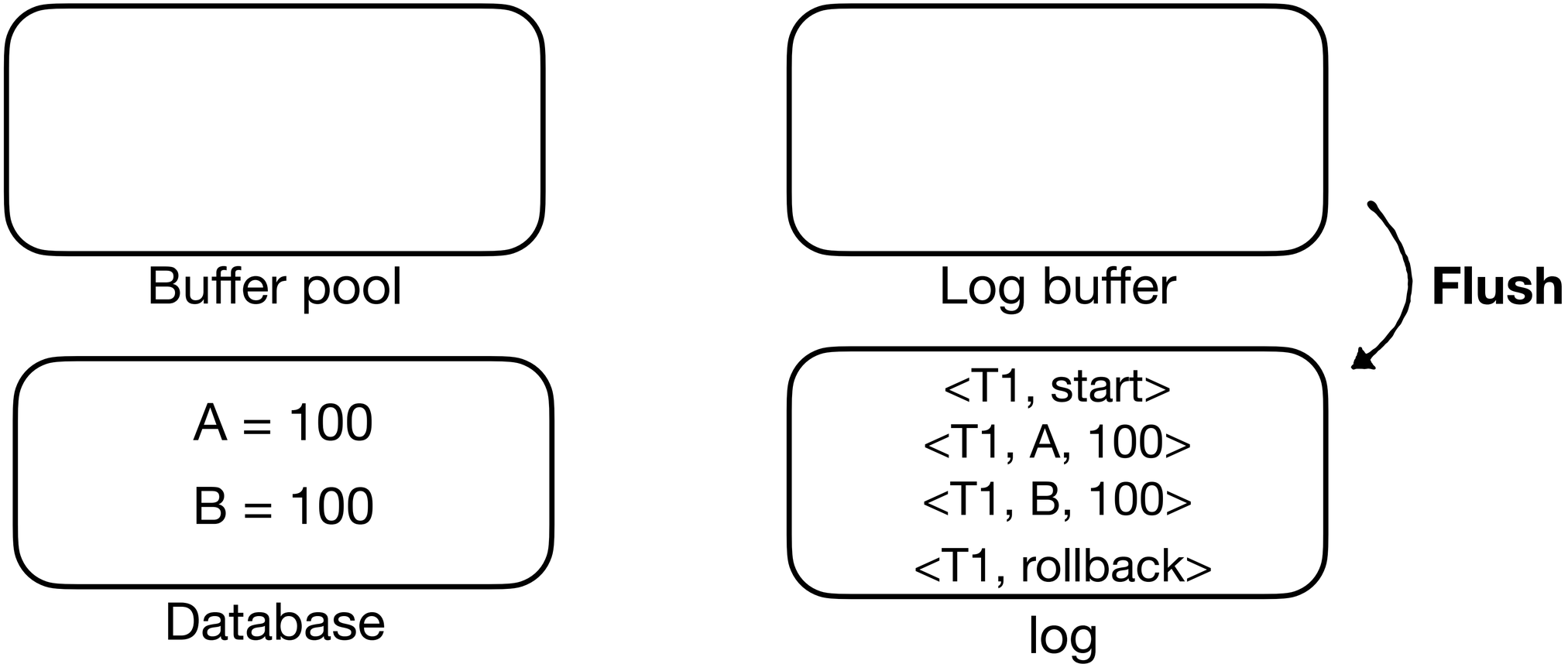


Recovery Undo logging:



Recovery Undo logging:

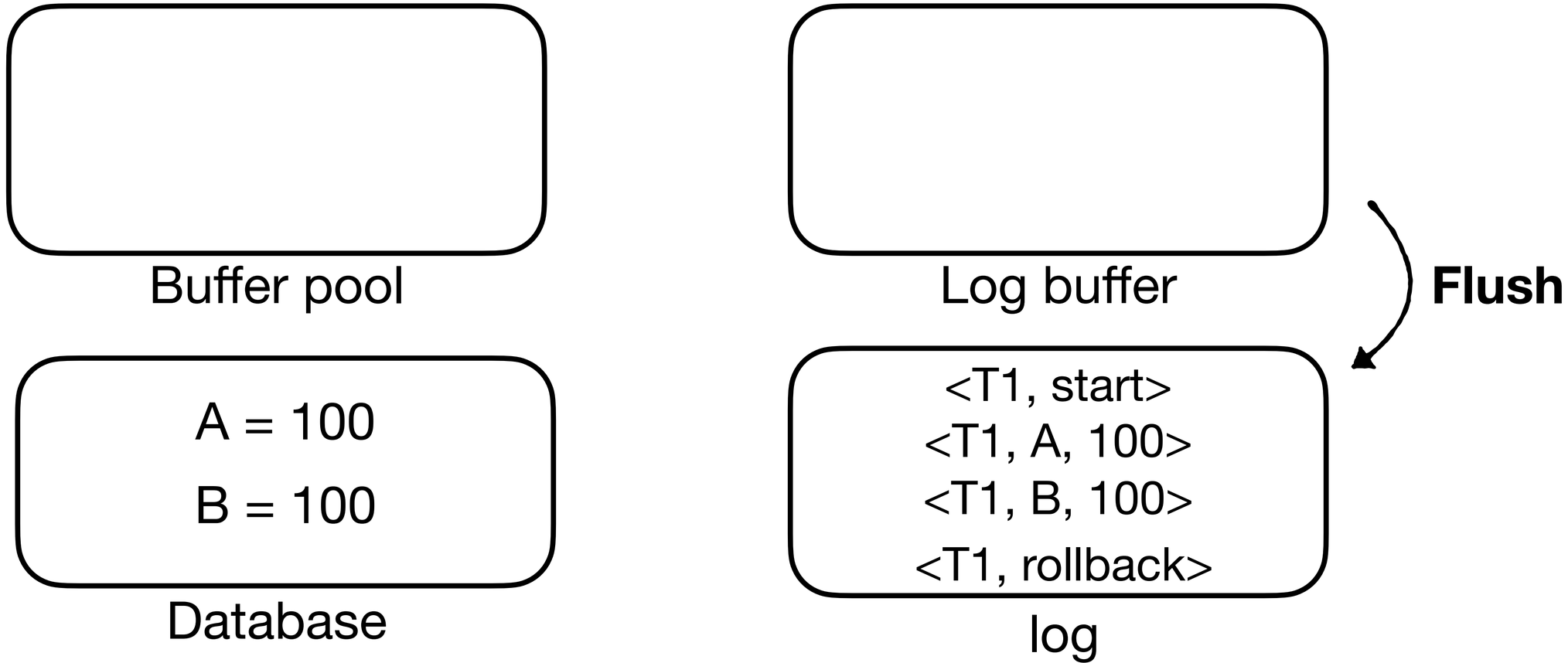
Why do we need this rollback record?



Recovery Undo logging:

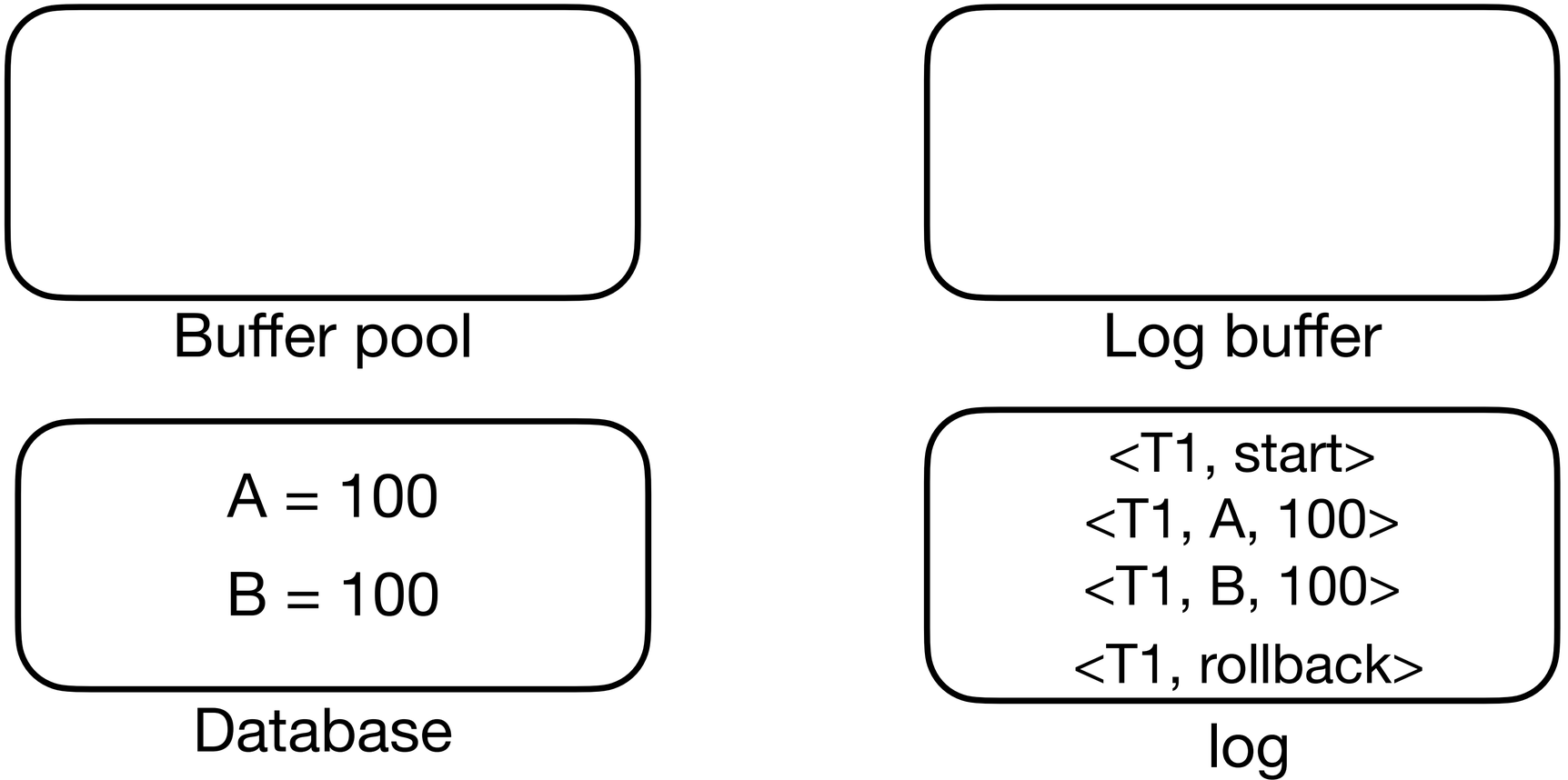
Why do we need this rollback record?

It saves us work if power fails again so we don't undo B again

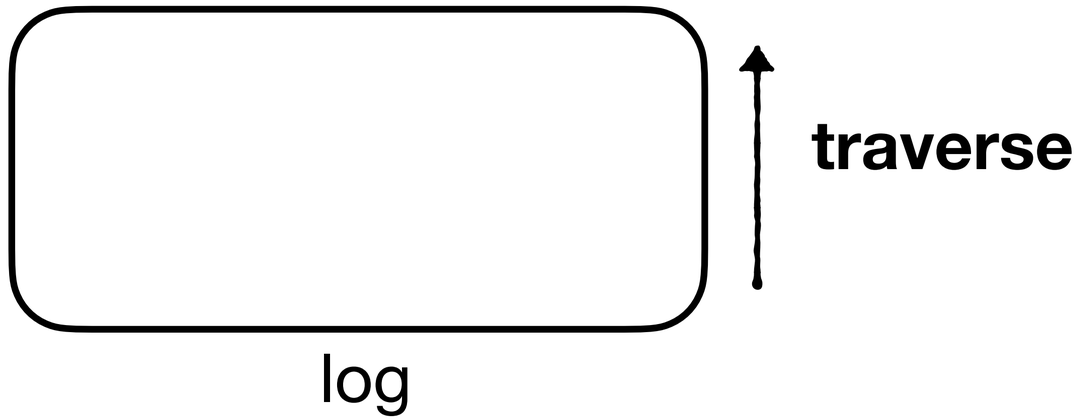


Recovery Undo logging:

We are now done. It's as if the original transaction never executed.

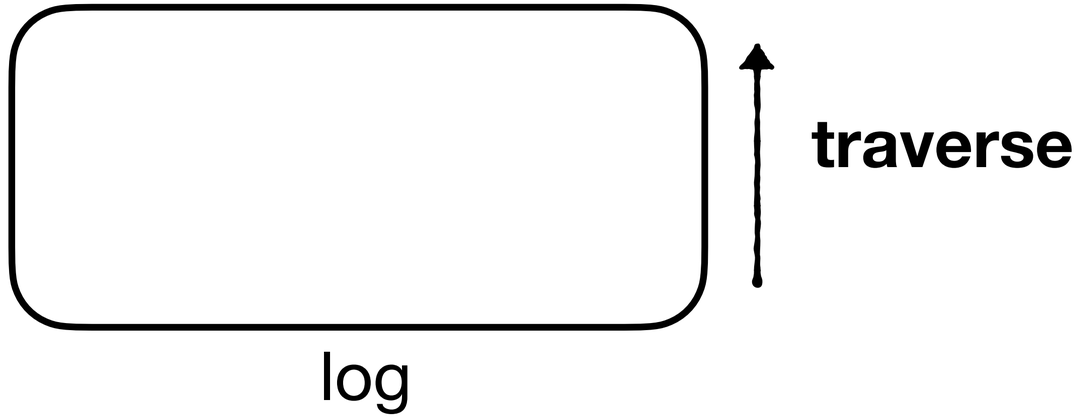


How much of the log must we traverse during recovery?



How much of the log must we traverse during recovery?

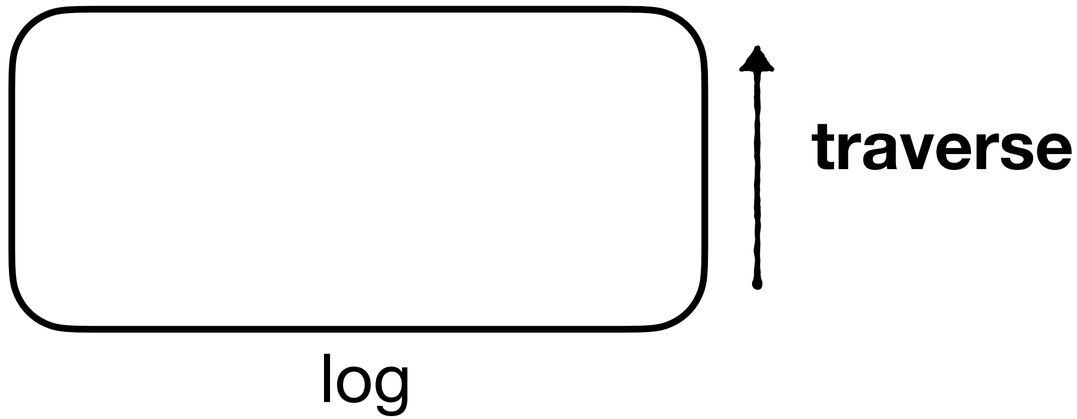
All of it, there may be a transaction from the very beginning that hasn't committed.



How much of the log must we traverse during recovery?

All of it, there may be a transaction from the very beginning that hasn't committed.

How can we bound recovery time without having to traverse the whole log?



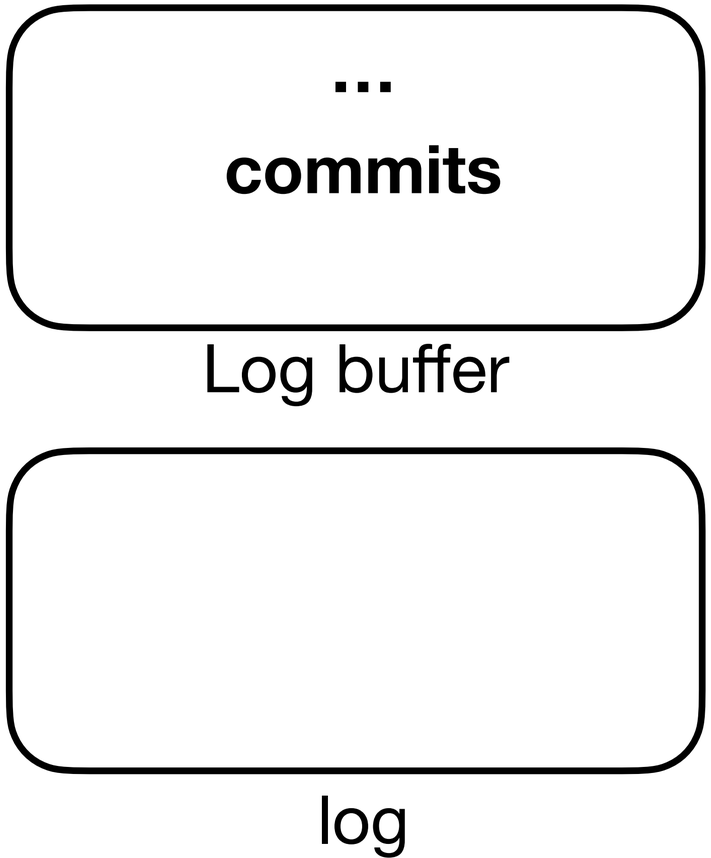
Checkpointing

Log buffer

log

Checkpointing

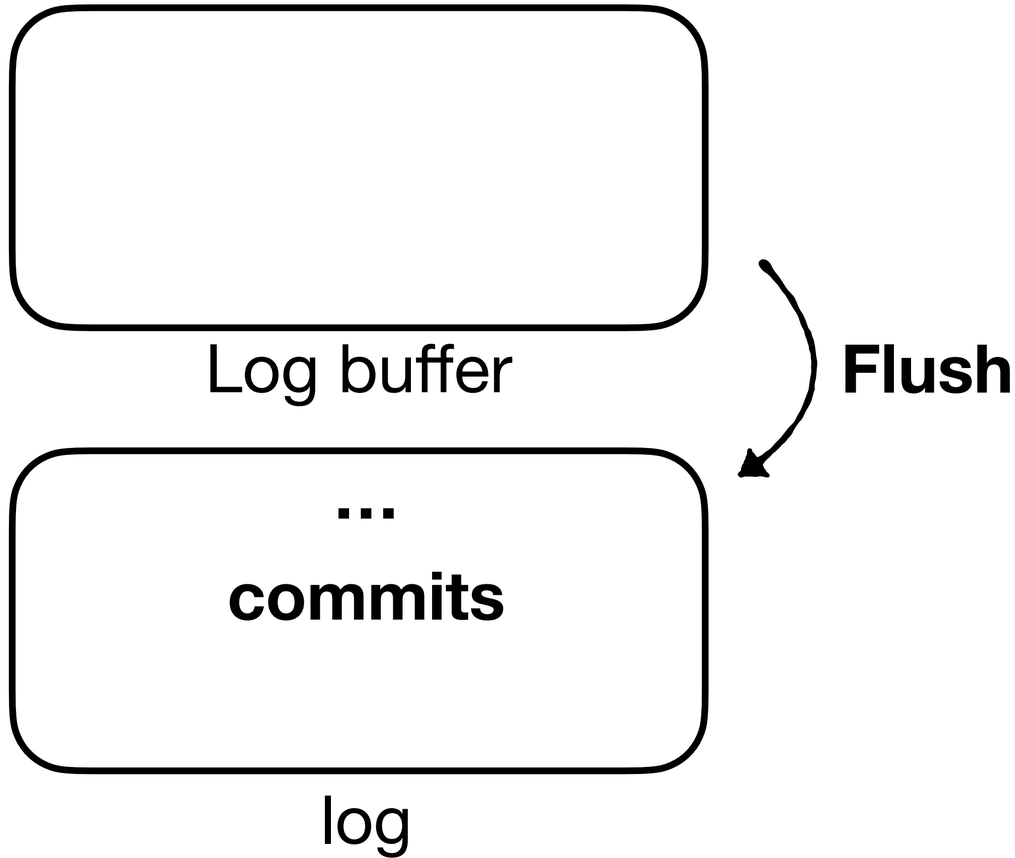
(1) Stop accepting new transactions & wait until all existing ones commit



Checkpointing

(1) Stop accepting new transactions & wait until all existing ones commit

(2) flush log to storage

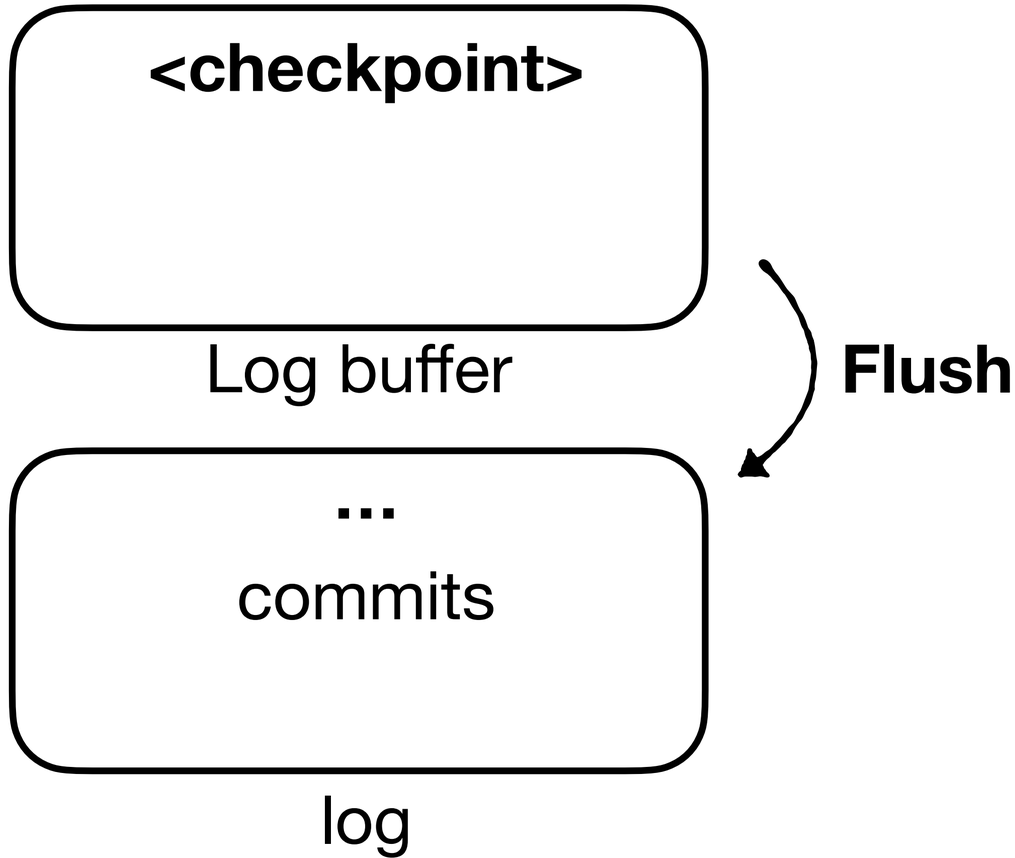


Checkpointing

(1) Stop accepting new transactions & wait until all existing ones commit

(2) flush log to storage

(3) add checkpoint record to log buffer and flush again



Checkpointing

(1) Stop accepting new transactions & wait until all existing ones commit
(2) flush log to storage
(3) add checkpoint record to log buffer and flush again

Log buffer

During next recovery, only traverse log up to first checkpoint record we encounter

log

UNDO Logging steps

(1) write before-image for any changed value to log buffer

(2) force flush log

(3) force the changed data into storage

(4) add commit record to log buffer & force flush again

UNDO Logging steps

(1) write before-image for any changed value to log buffer

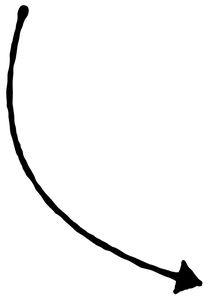
(2) force flush log

(3) force the changed data into storage

(4) add commit record to log buffer & force flush again

Any performance issues?

Many random I/Os



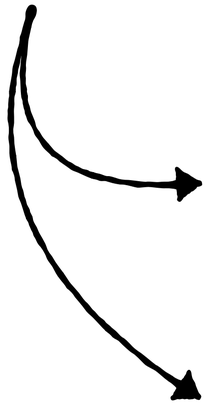
(1) write before-image for any changed value to log buffer

(2) force flush log

(3) force the changed data into storage

(4) add commit record to log buffer & force flush again

Two sync I/Os



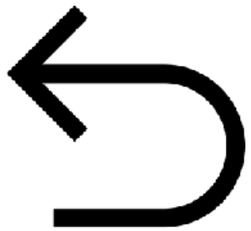
(1) write before-image for any changed value to log buffer

(2) force flush log

(3) force the changed data into storage

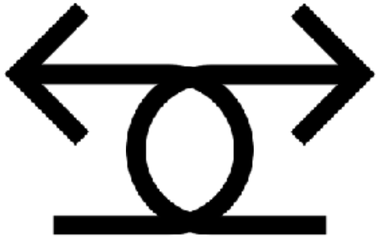
(4) add commit record to log buffer & **force flush again**

Undo



Redo

Redo/Redo



random I/Os

Redo logging:

Invariant: If transactions **is not** marked as committed in the log in storage, none of its changes have been persisted to the database

Redo logging:

Invariant: If transactions **is not** marked as committed in the log in storage, none of its changes have been persisted to the database

Consequences: (1) buffer pool cannot evict uncommitted (dirty) data

Redo logging:

Invariant: If transactions **is not** marked as committed in the log in storage, none of its changes have been persisted to the database

Consequences: (1) buffer pool cannot evict uncommitted (dirty) data

(2) while recovering, we can ignore uncommitted transactions

Redo logging:

Invariant: If transactions **is not** marked as committed in the log in storage, none of its changes have been persisted to the database

Consequences: (1) buffer pool cannot evict uncommitted (dirty) data

(2) while recovering, we can ignore uncommitted transactions

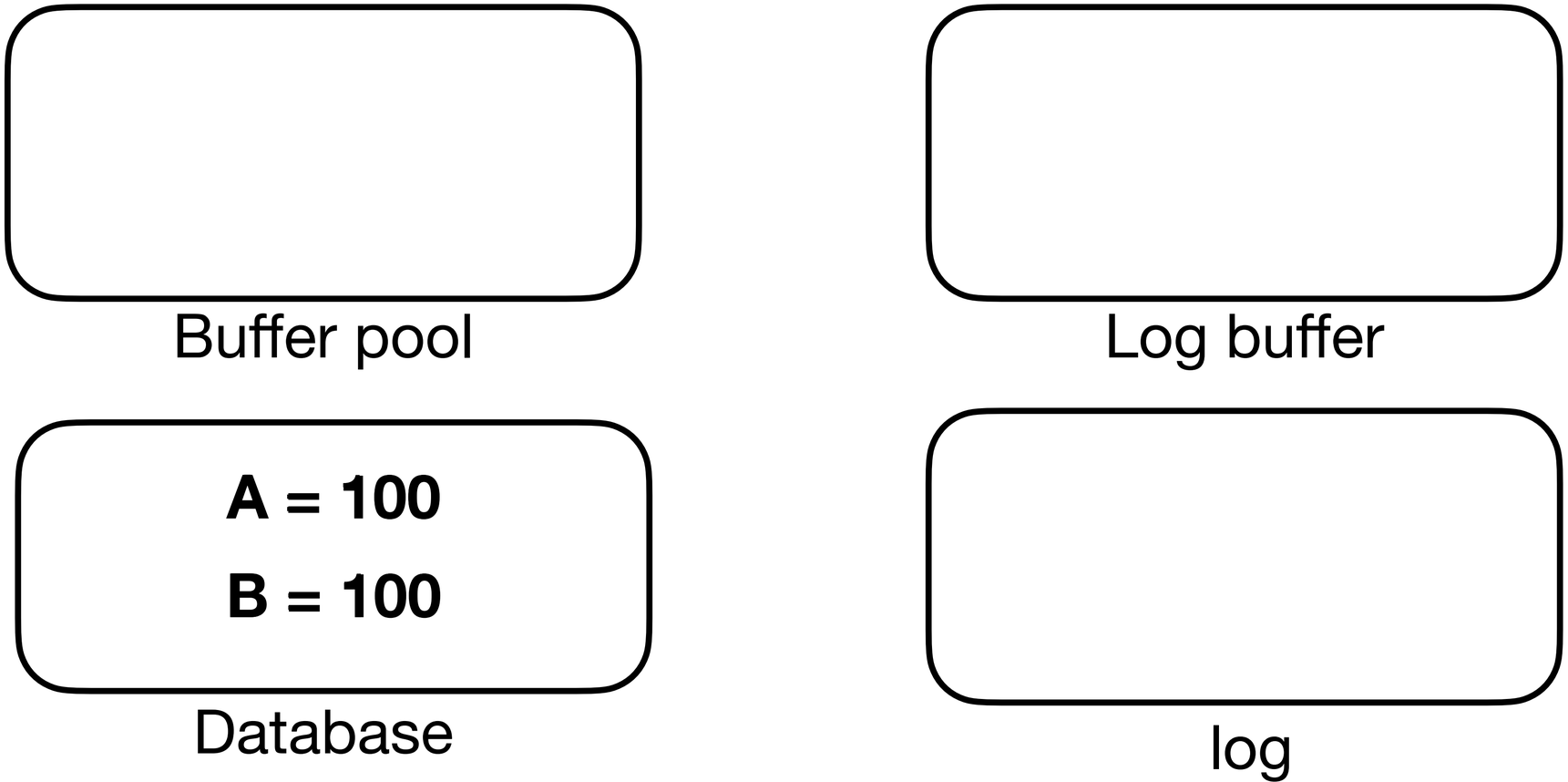
(3) Must redo committed transactions if their changes have not been flushed

Redo logging:

Transaction

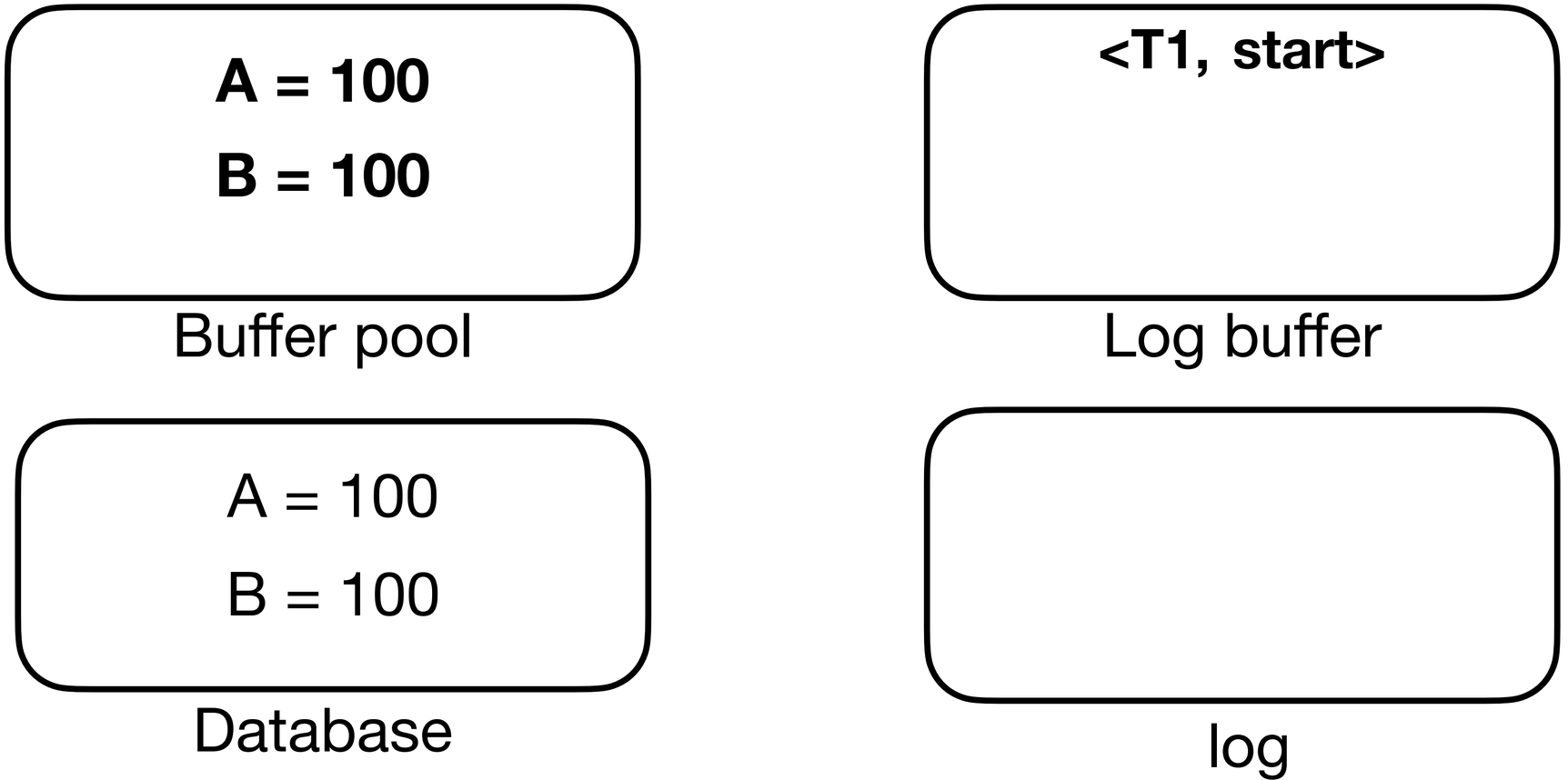
A = A - 100

B = B + 100



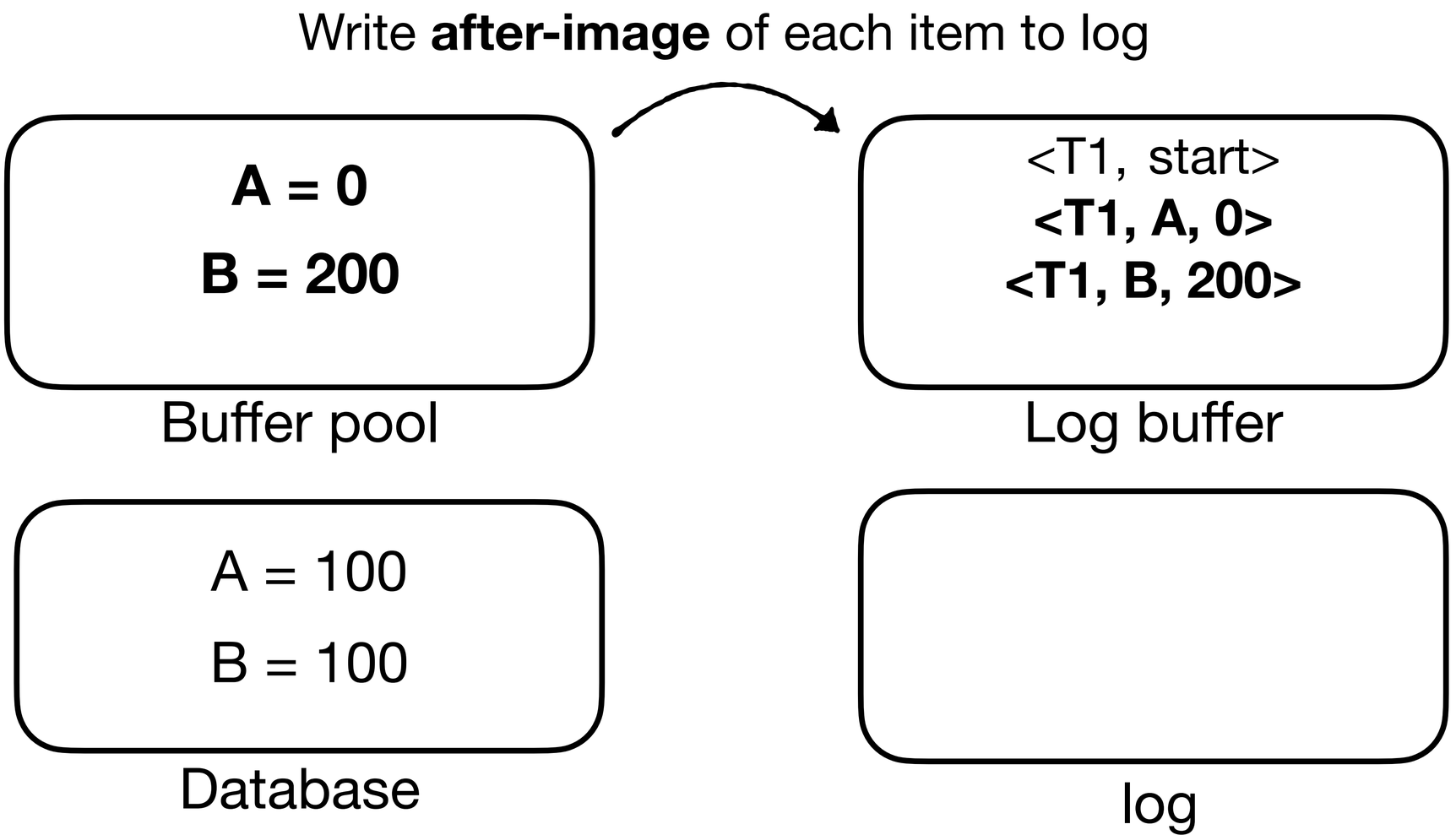
Redo logging:

Transaction
A = A - 100
B = B + 100



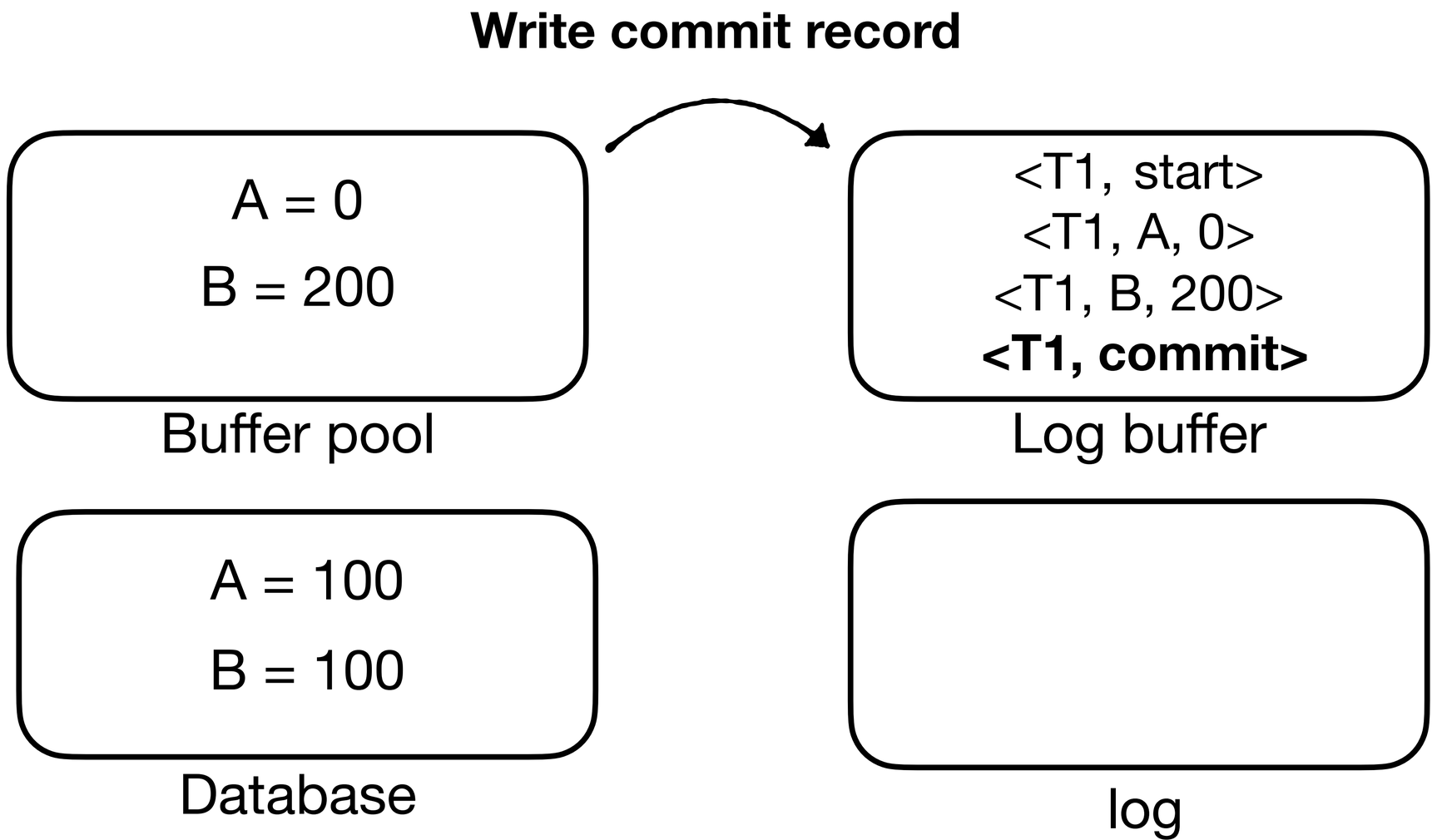
Redo logging:

Transaction
A = A - 100
B = B + 100



Redo logging:

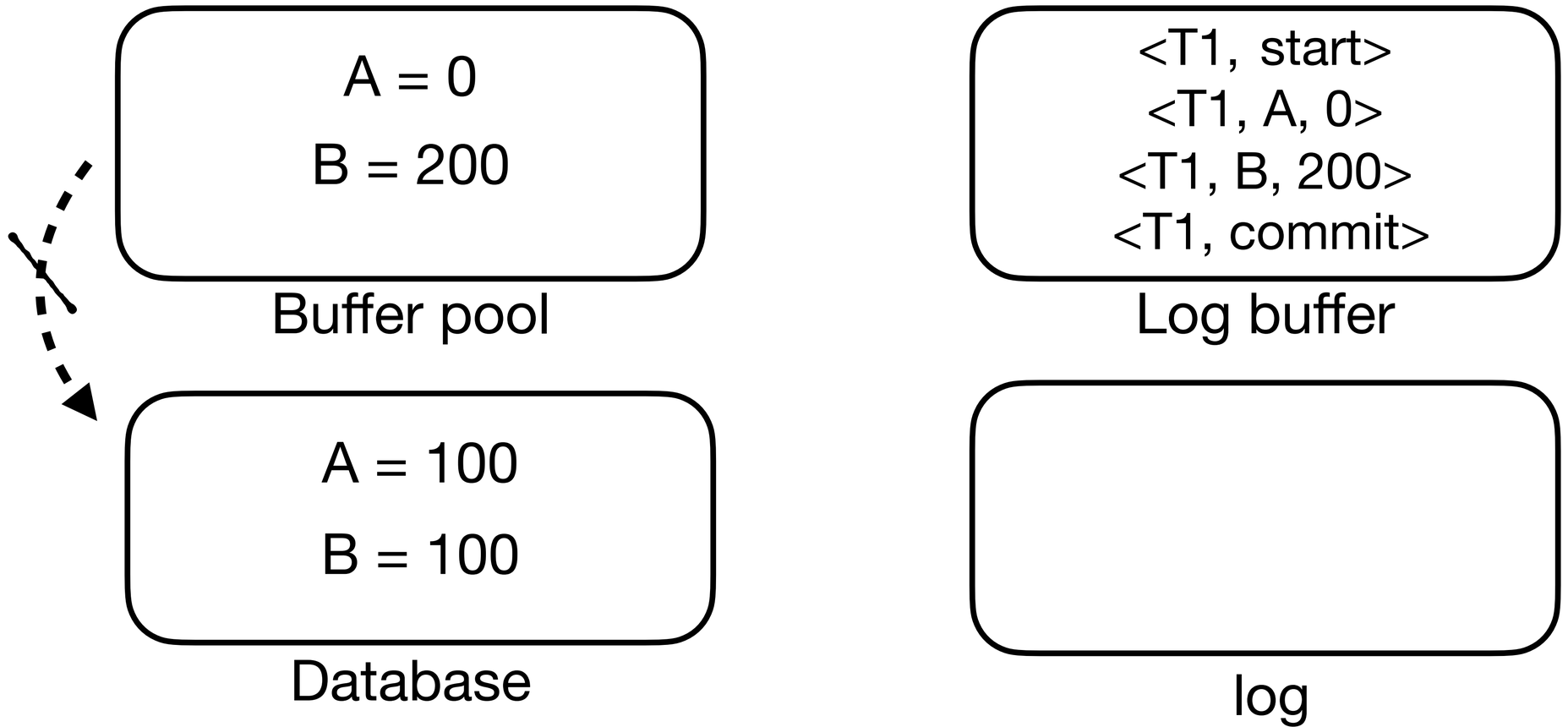
Transaction
A = A - 100
B = B + 100



Redo logging:

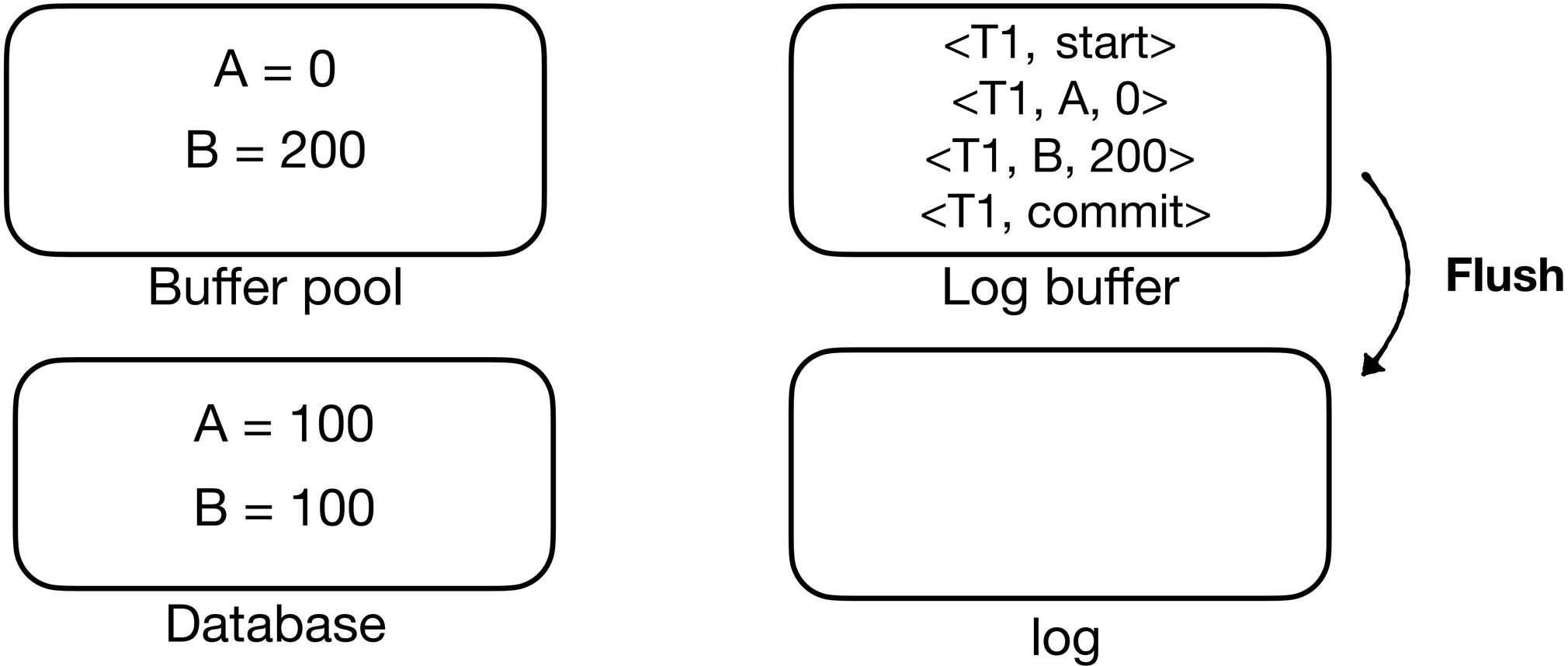
Transaction
A = A - 100
B = B + 100

Not allowed to evict until log buffer flushes



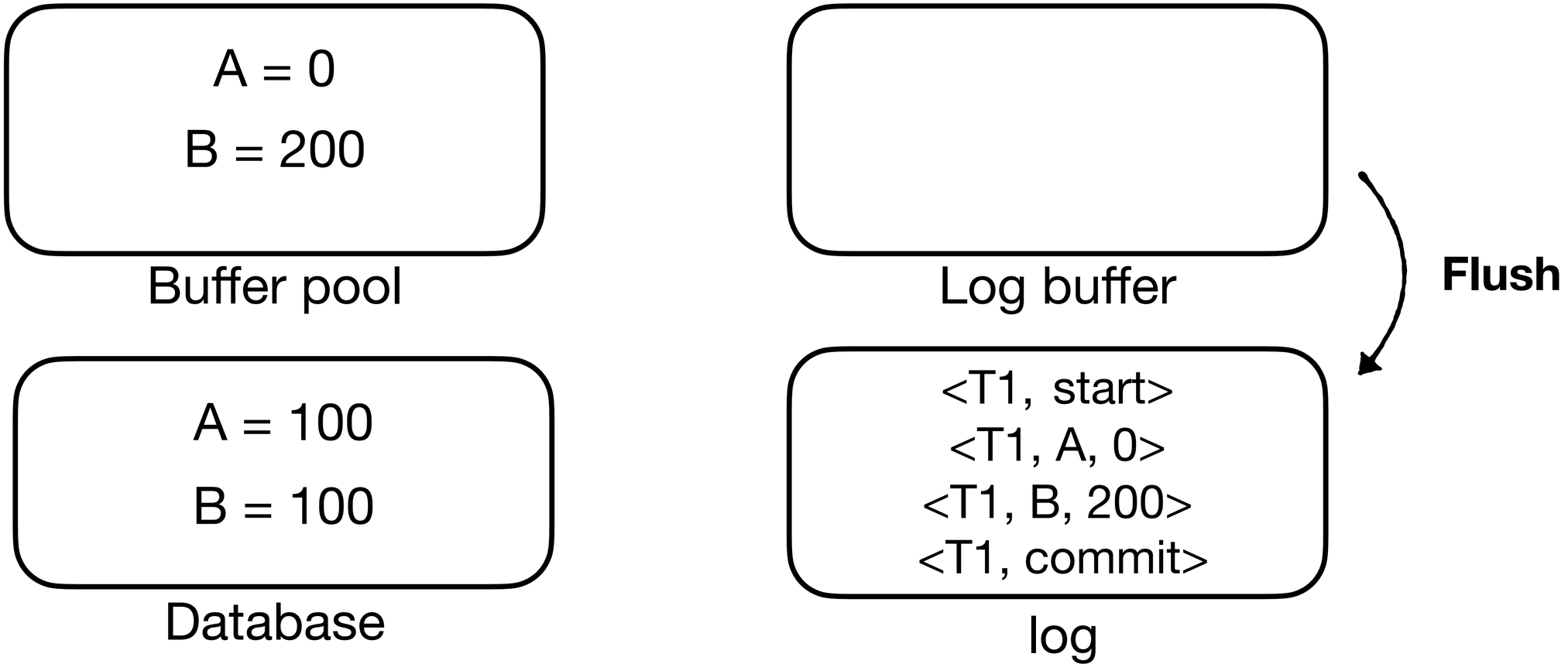
Redo logging:

Transaction
A = A - 100
B = B + 100



Redo logging:

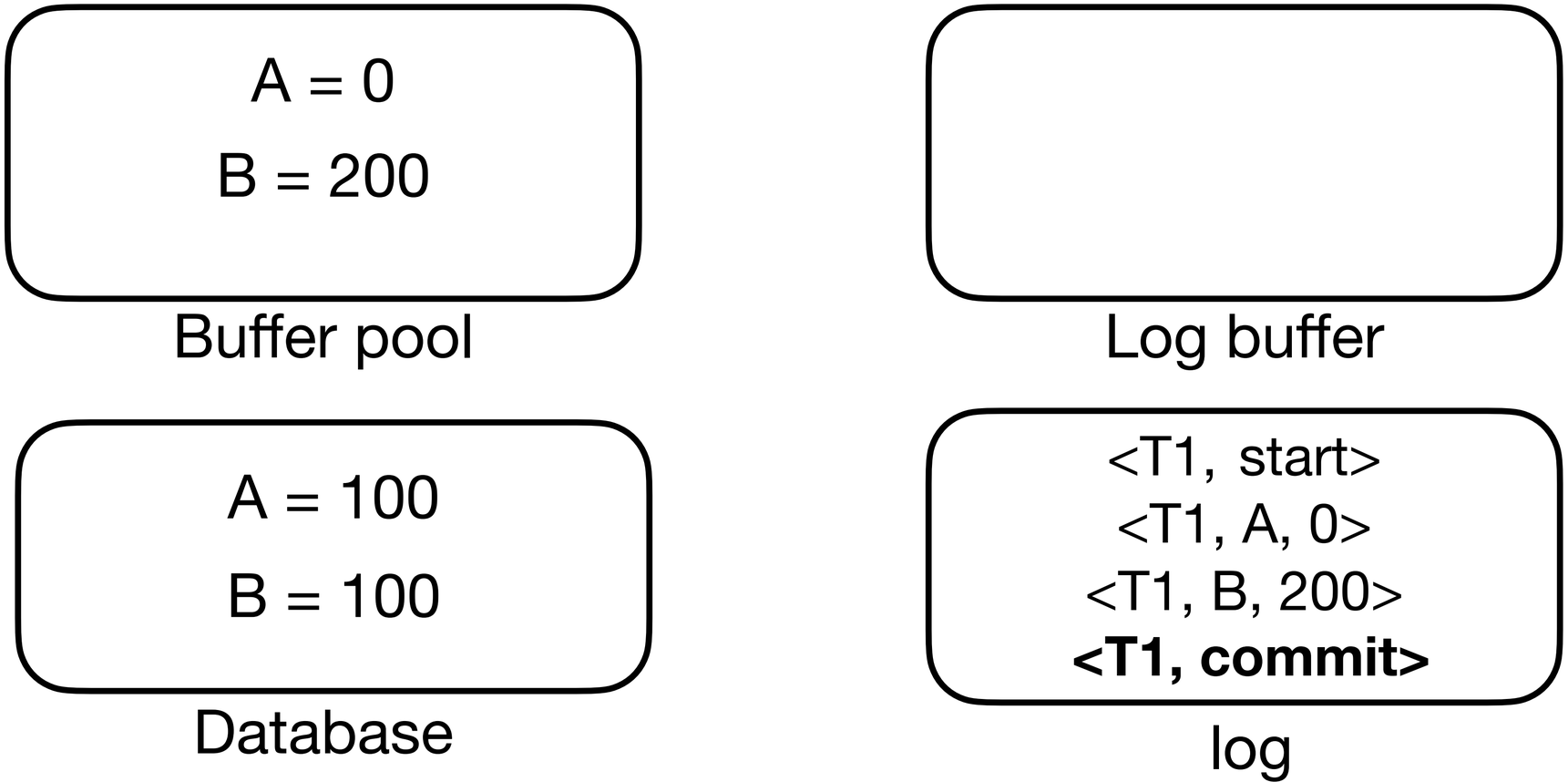
Transaction
A = A - 100
B = B + 100



Redo logging:

Transaction
A = A - 100
B = B + 100

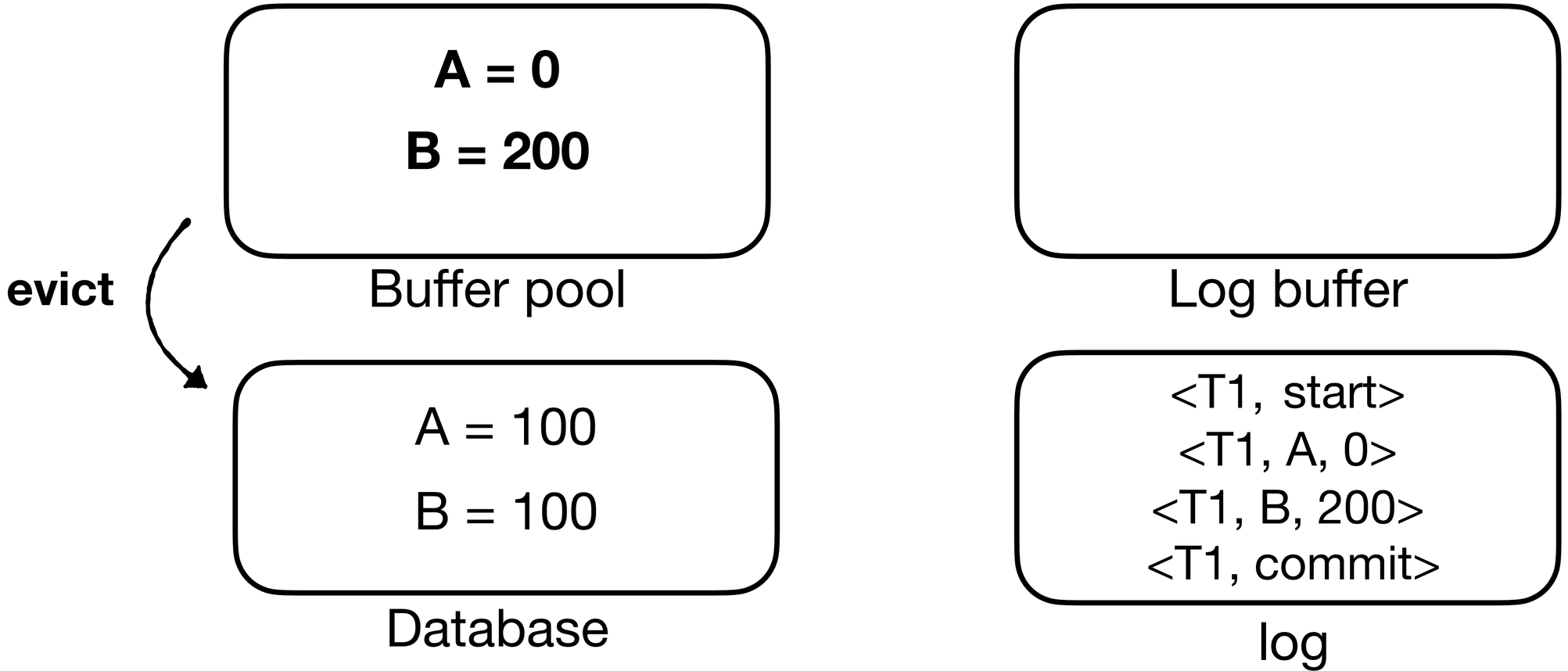
Transaction is now committed



Redo logging:

Transaction
A = A - 100
B = B + 100

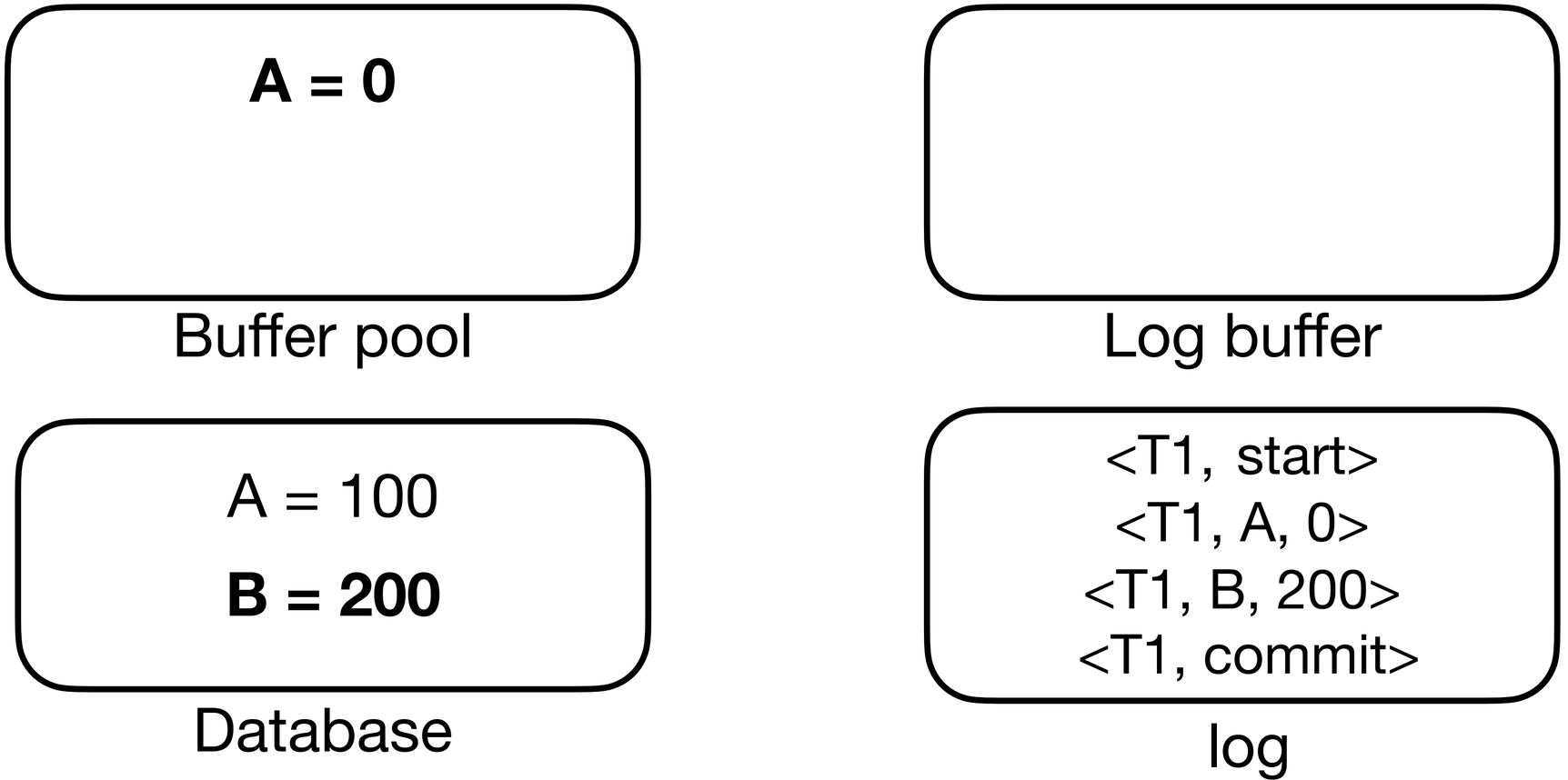
Now allowed to evict at leisure



Redo logging:

Transaction
A = A - 100
B = B + 100

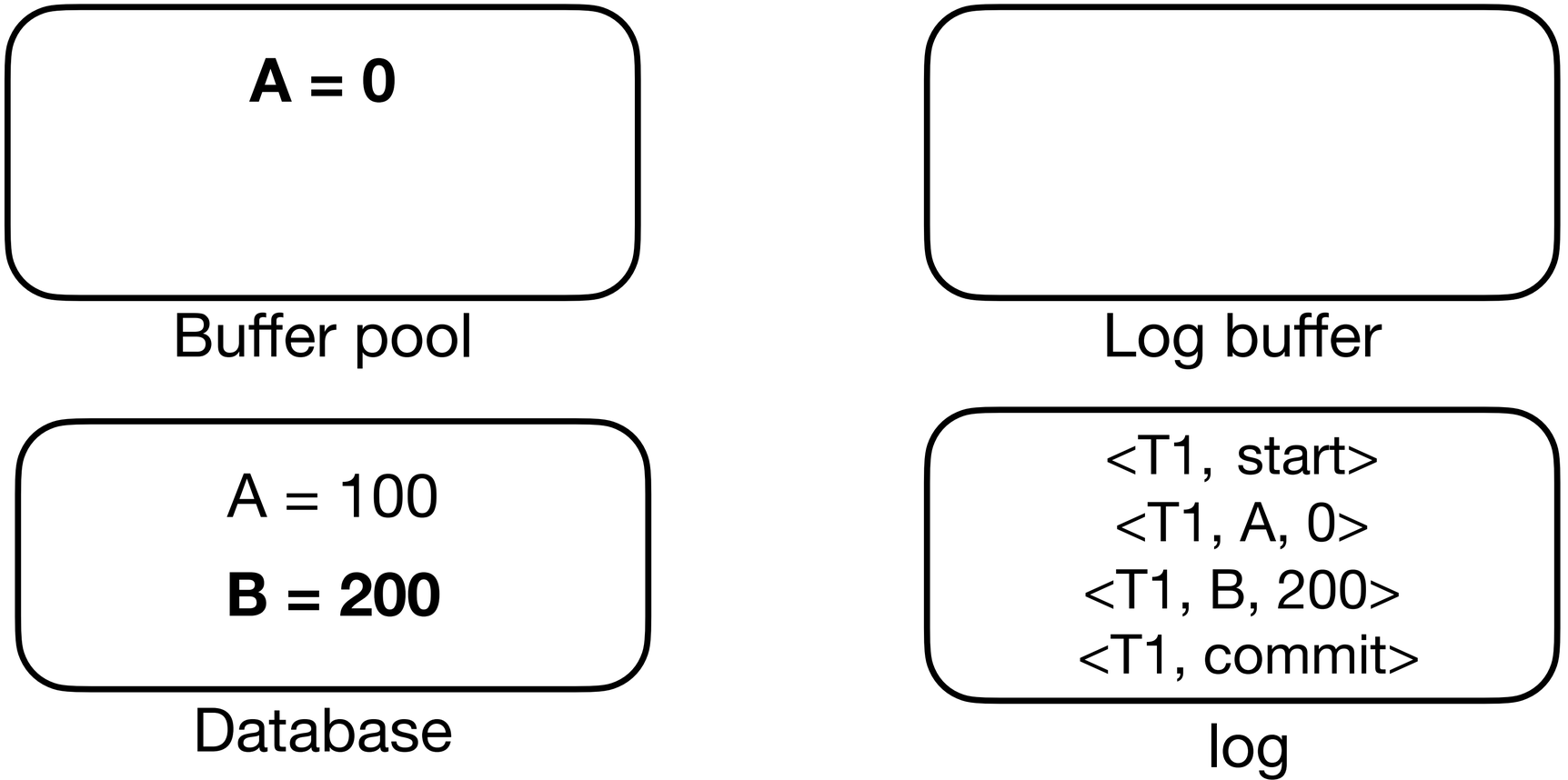
Note that we don't have to force all changes to storage at once
(A stays in memory in this example)



Redo logging:

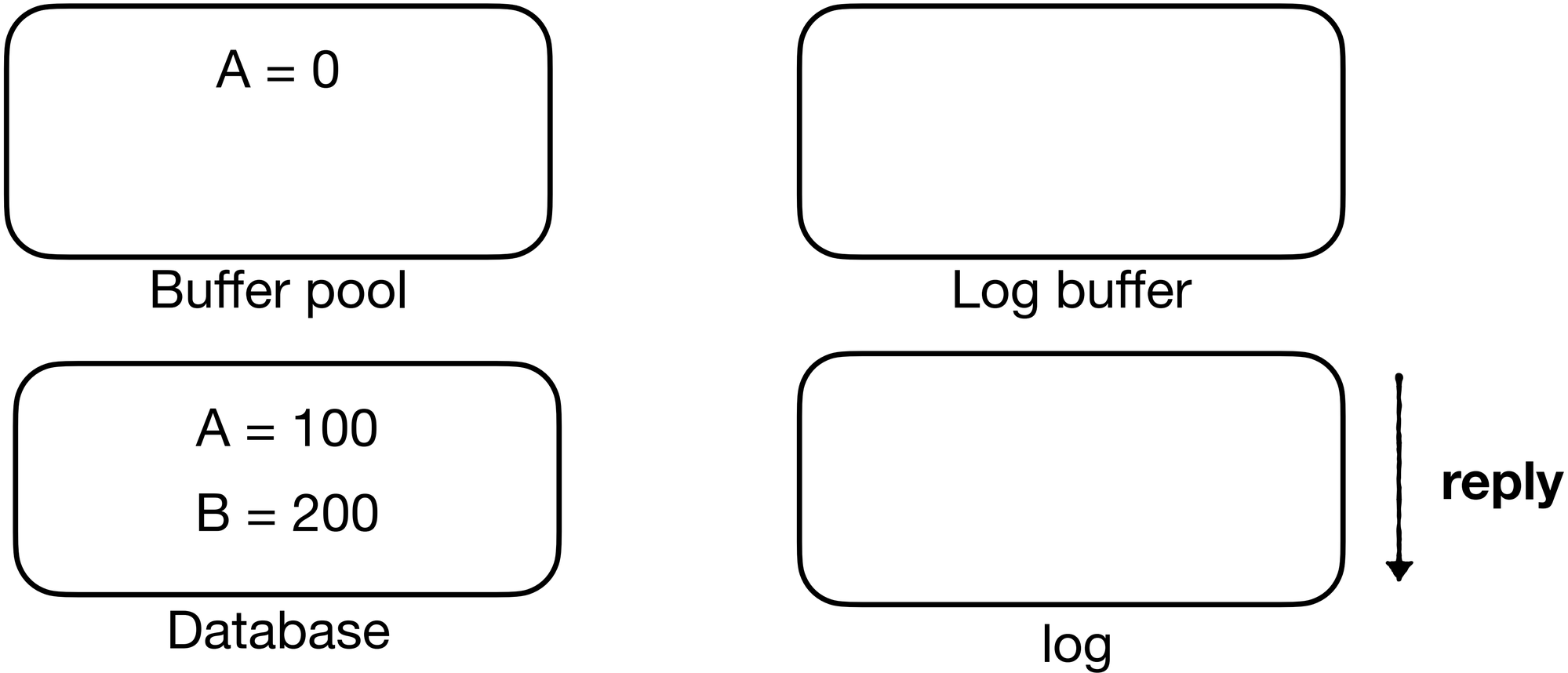
Transaction
A = A - 100
B = B + 100

The buffer evicts committed data autonomously based on clock or LRU



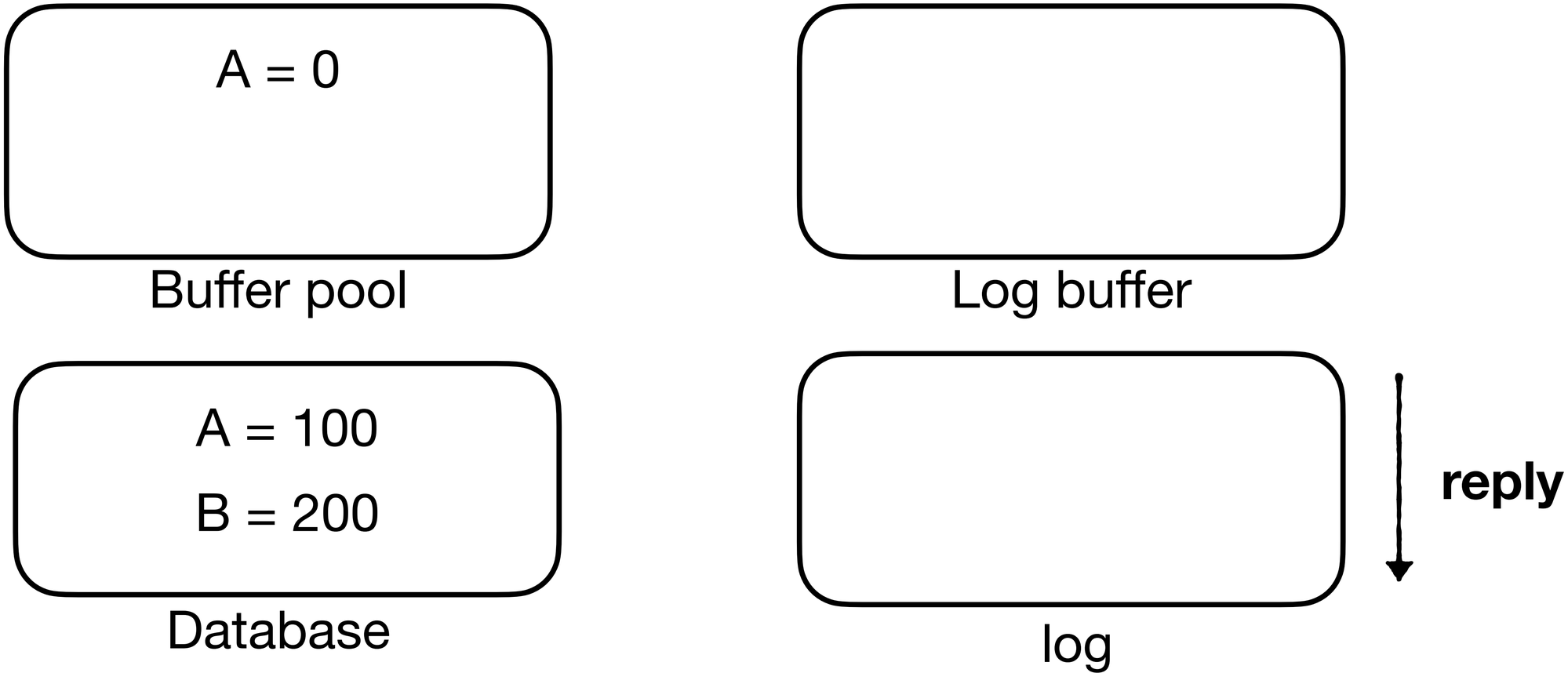
Recovery with Redo logging:

To recover, traverse log forward, replying effects of all committed transactions



Recovery with Redo logging:

To recover, traverse log forward, replying effects of all committed transactions



Recovery with Redo logging:

Suppose the transaction was not committed and power failed

A = 0

Buffer pool

Log buffer

A = 100

B = 100

Database

<T1, start>
<T1, A, 0>
<T1, B, 200>

log

Recovery with Redo logging:

Suppose the transaction was not committed and power failed

If there is no commit, nothing to do as we know modified versions didn't reach storage

Buffer pool

Log buffer

A = 100

B = 100

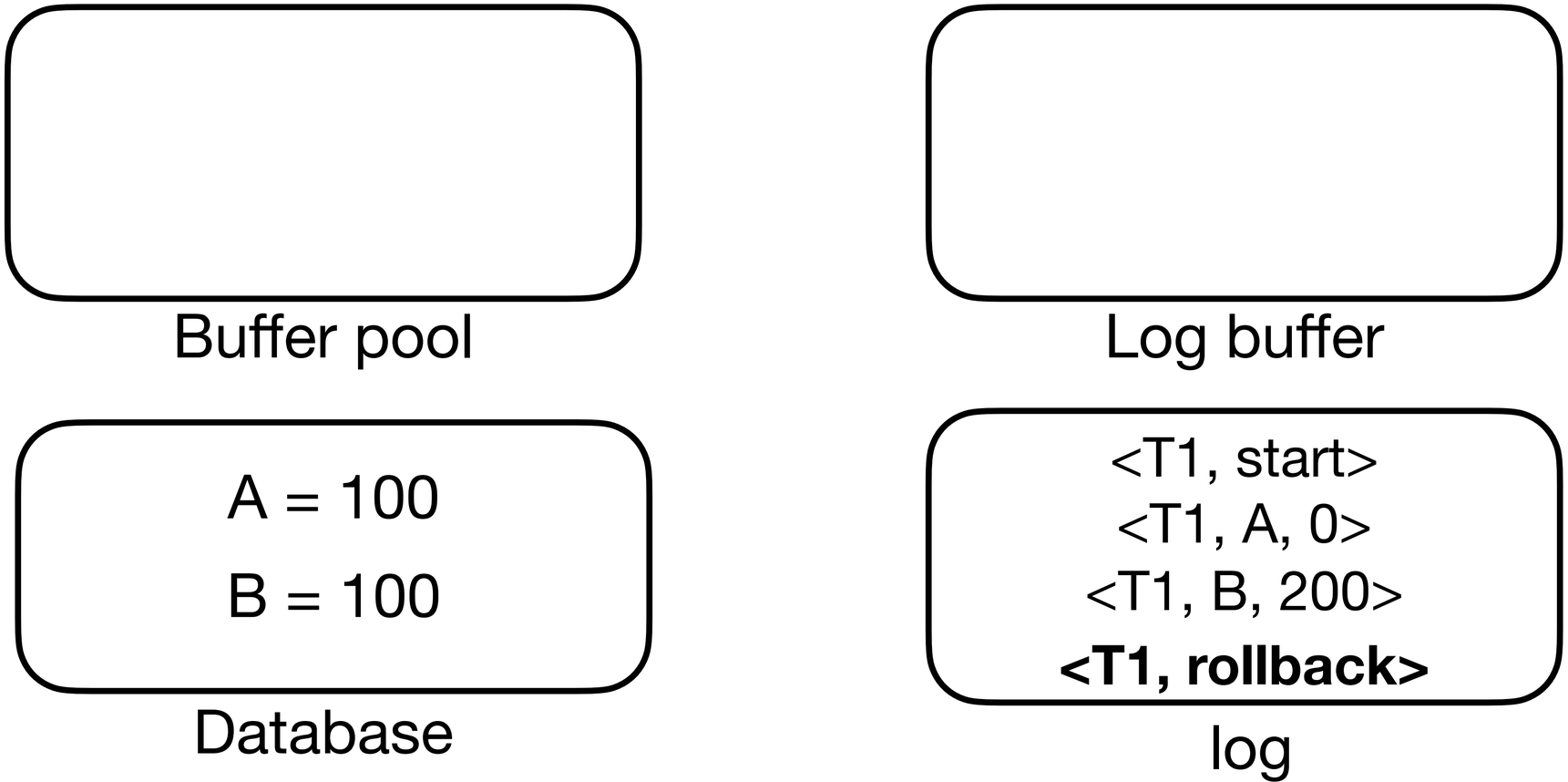
Database

<T1, start>
<T1, A, 0>
<T1, B, 200>

log

Recovery with Redo logging:

add rollback record



Recovery with Redo logging:

Now, suppose the transaction committed but power failed before all changes reached storage

A = 0

Buffer pool

Log buffer

A = 100

B = 200

Database

<T1, start>
<T1, A, 0>
<T1, B, 200>
<T1, commit>

log

Recovery with Redo logging:

Now, suppose the transaction committed but power failed before all changes reached storage

How to recover?

Buffer pool

Log buffer

A = 100

B = 200

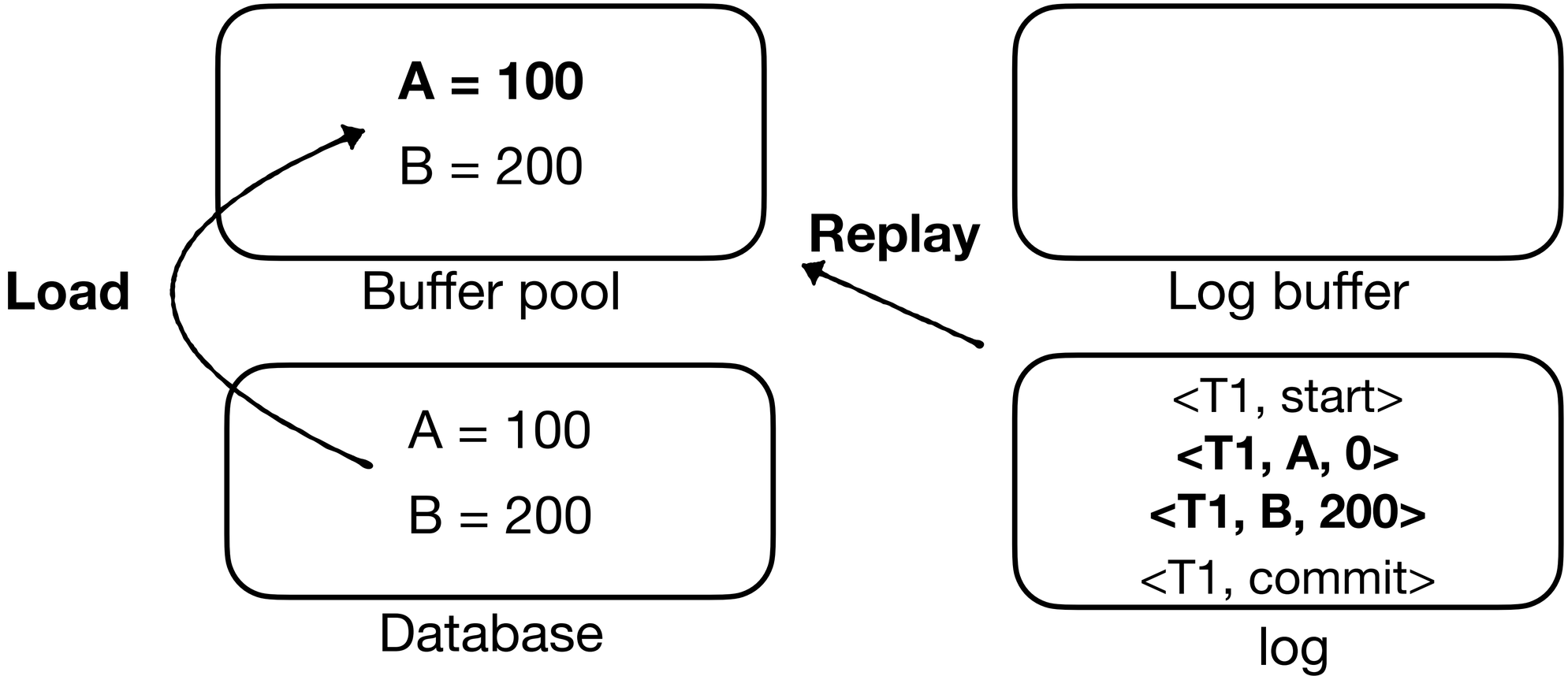
Database

<T1, start>
<T1, A, 0>
<T1, B, 200>
<T1, commit>

log

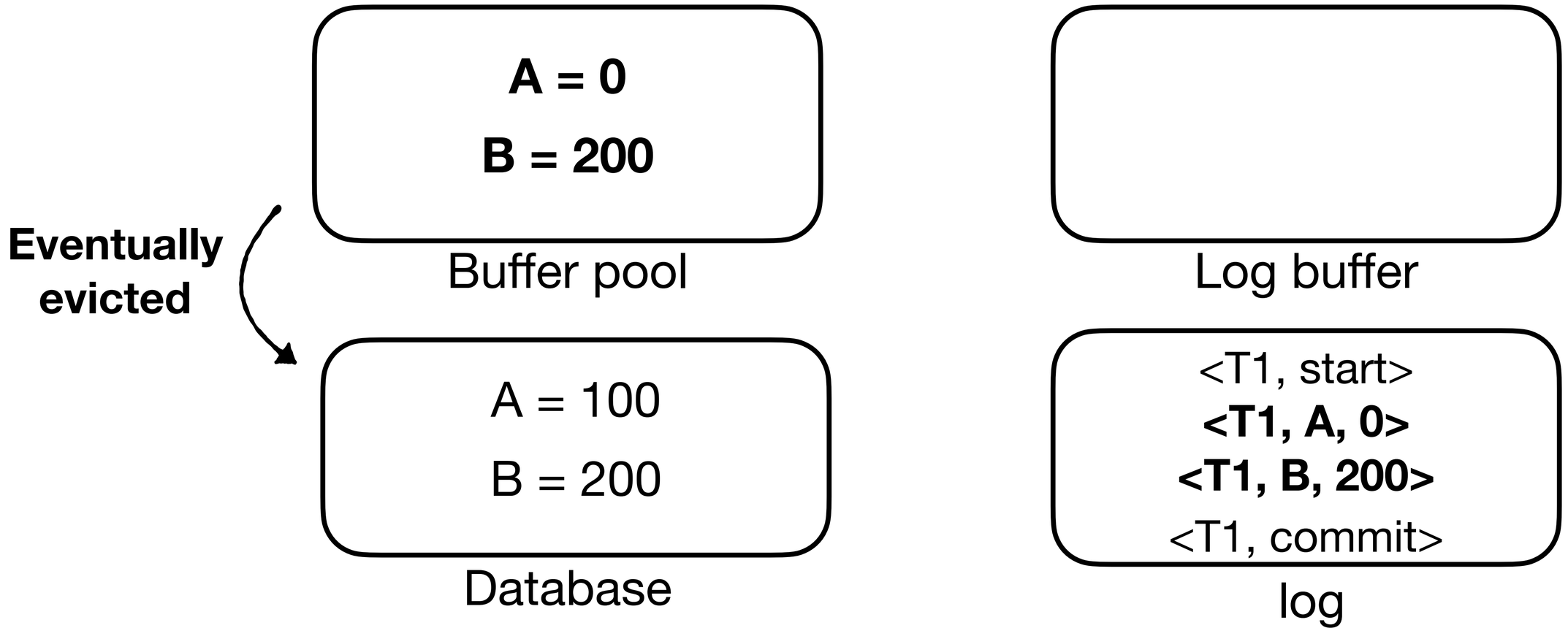
Recovery with Redo logging:

Now, suppose the transaction committed but power failed before all changes reached storage



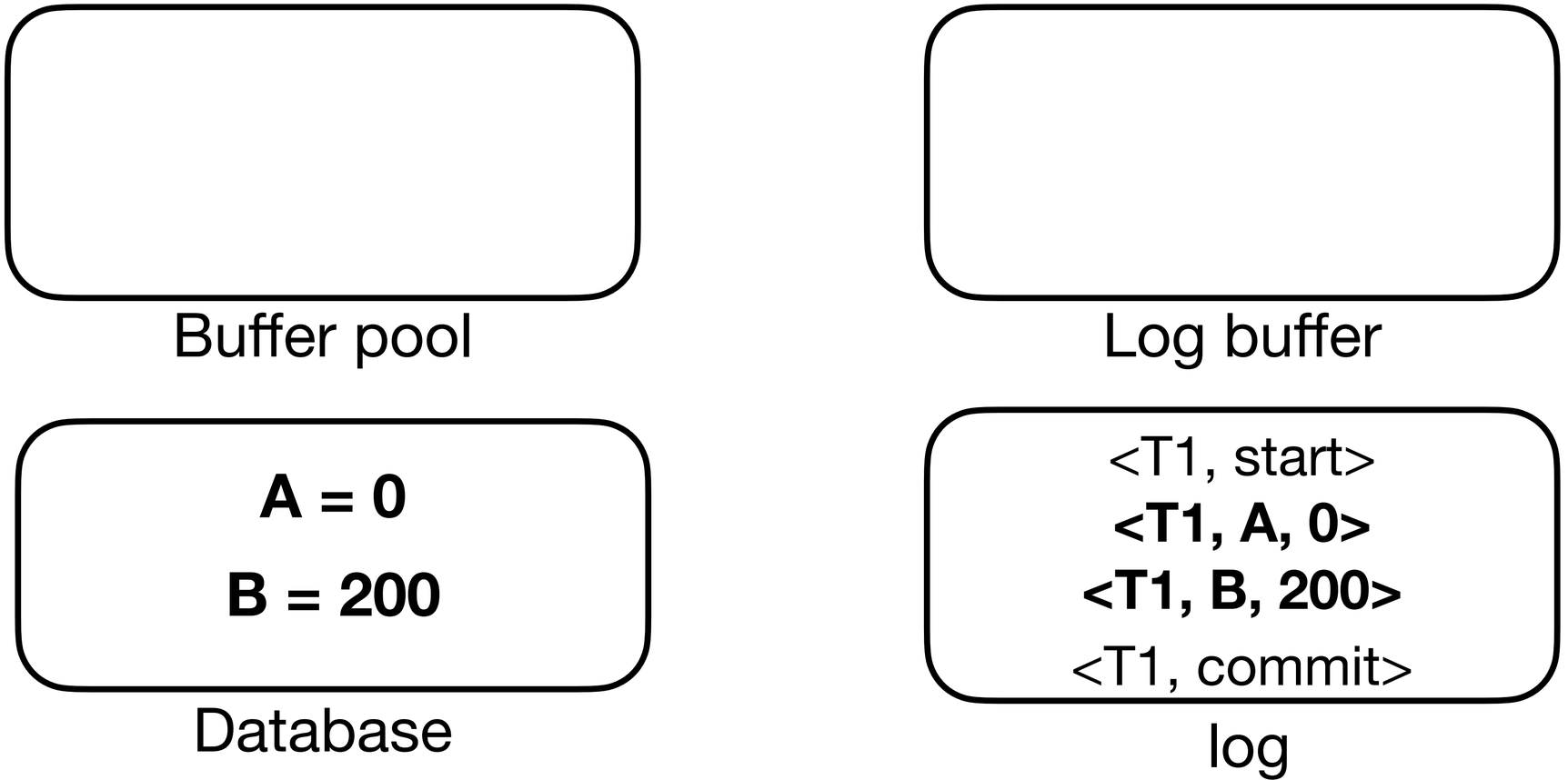
Recovery with Redo logging:

Now, suppose the transaction committed but power failed before all changes reached storage



Recovery with Redo logging:

Now, suppose the transaction committed but power failed before all changes reached storage



Does not force data into storage

(1) write after-image for any changed value to log buffer
(2) keep changed data in buffer pool
(3) add commit record to log buffer & flush
(4) flush changed data to storage at leisure

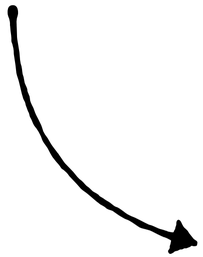
Does not force data into storage

(1) write after-image for any changed value to log buffer
(2) keep changed data in buffer pool
(3) add commit record to log buffer & flush
(4) flush changed data to storage at leisure

Problem?

Any problems with Redo logging?

Holds up memory



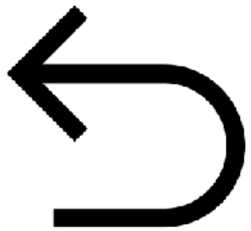
(1) write after-image for any changed value to log buffer

(2) keep changed data in buffer pool

(3) add commit record to log buffer & flush

(4) flush changed data to storage at leisure

Undo

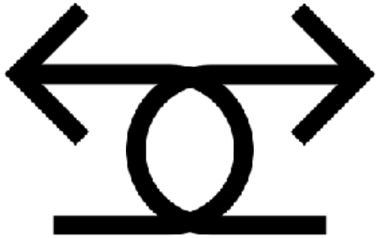


random I/Os

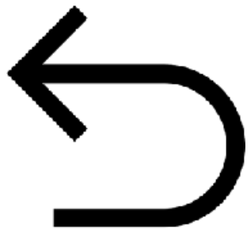
Redo

holds memory

Undo/Redo



Undo

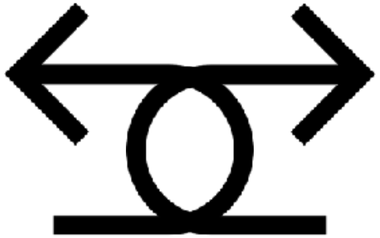


random I/Os

Redo

holds memory

Redo/Redo



Addresses both problems!

Undo/Redo logging:

Invariant: for any data persisted to storage, its before and after images must first be persisted in the log

Undo/Redo logging:

Invariant: for any data persisted to storage, its before and after images must first be persisted in the log

data item granularity rather than transaction-level granularity :)

Undo/Redo logging:

Invariant: for any data persisted to storage, its before and after images must first be persisted in the log

Consequence: To recover, undo uncommitted transactions

& redo committed transactions

Undo/Redo logging:

Invariant: for any data persisted to storage, its before and after images must first be persisted in the log

Consequence: To recover, undo uncommitted transactions

& redo committed transactions

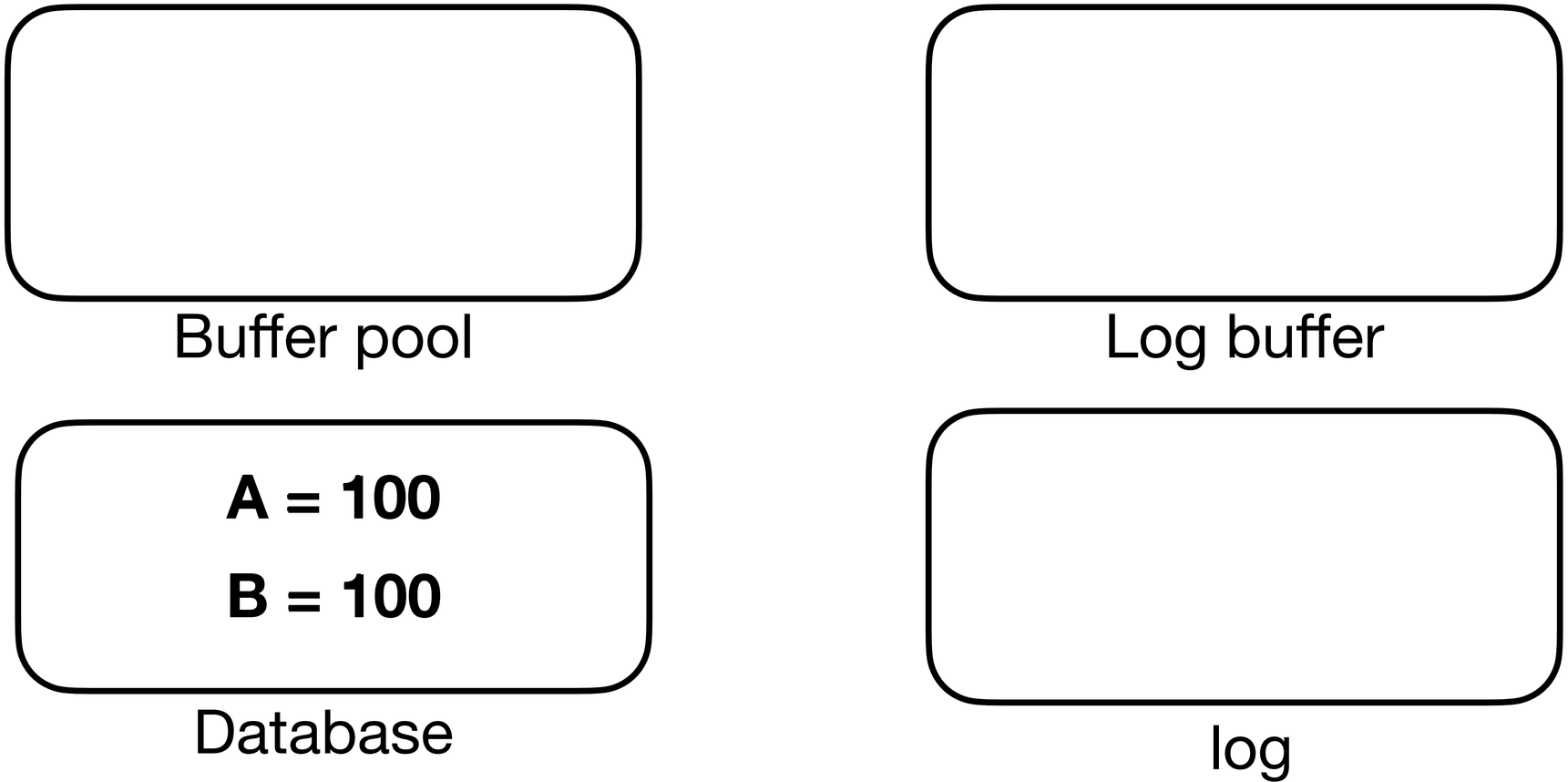
Corollary: cannot evict dirty data item to storage for which log entry was not already written

Undo/redo logging:

Transaction T1

A = A - 100

B = B + 100

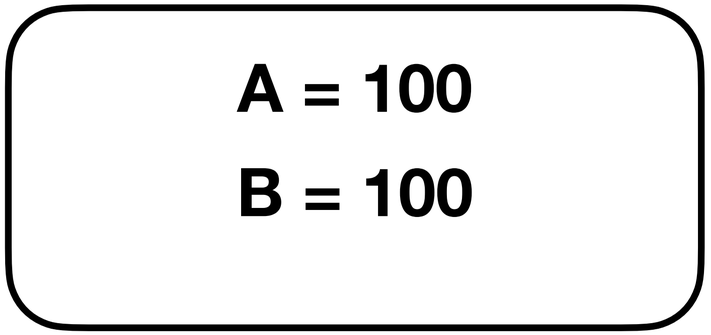


Undo logging:

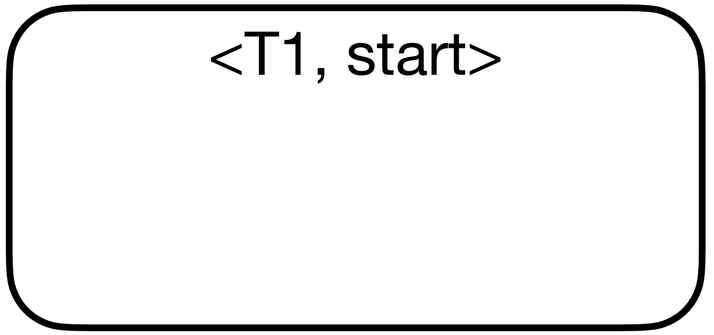
Transaction T1

A = A - 100

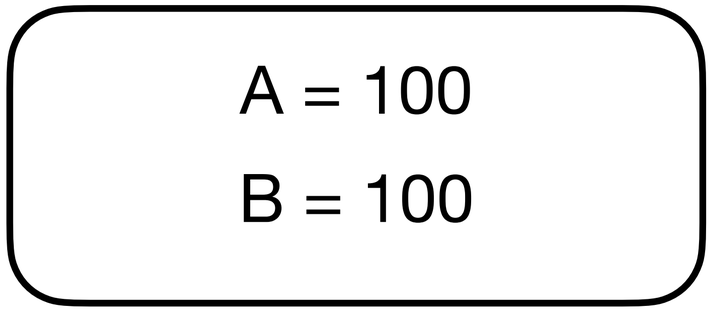
B = B + 100



Buffer pool



Log buffer



Database

log

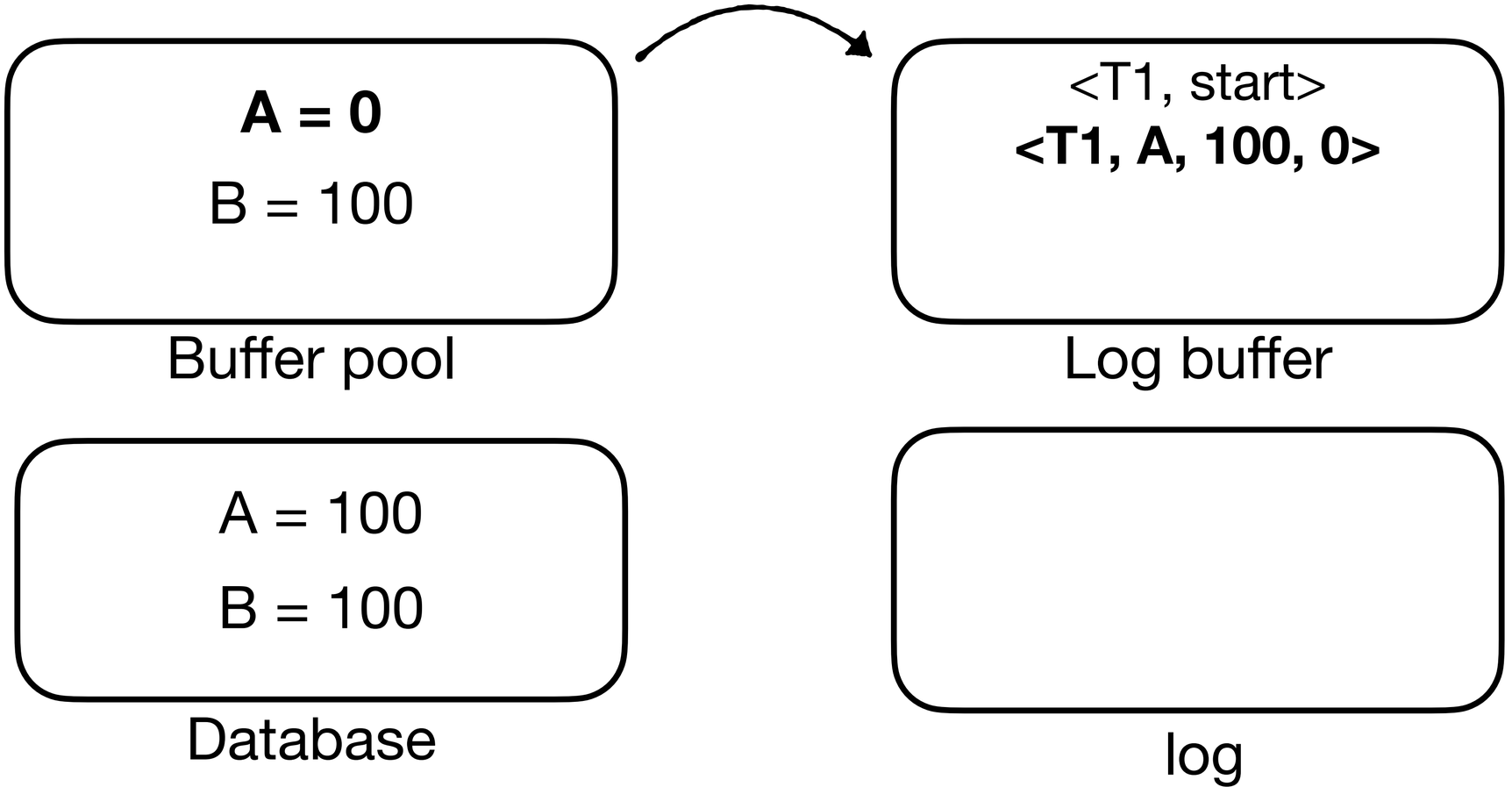
Undo logging:

Transaction T1

A = A - 100

B = B + 100

Write before & after image to log



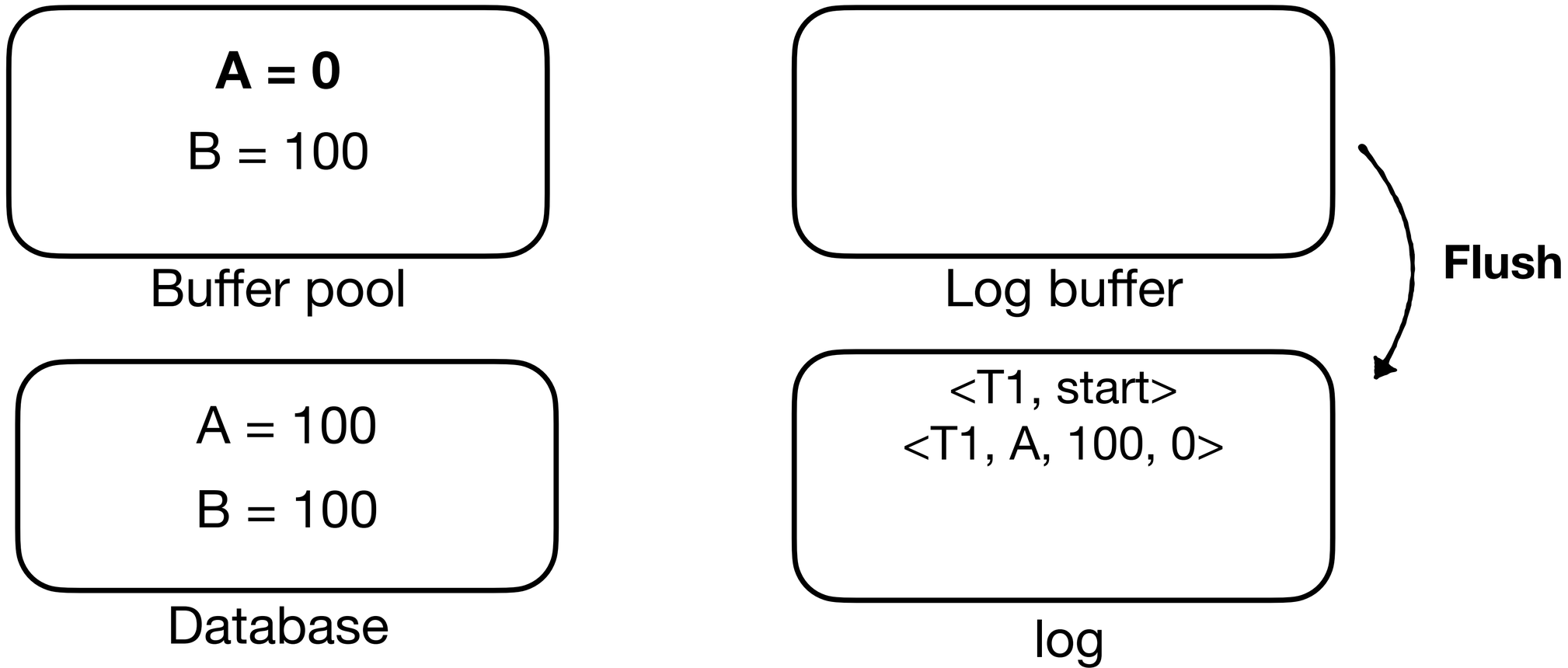
Undo logging:

Transaction T1

A = A - 100

B = B + 100

Suppose log flushes due to other transaction



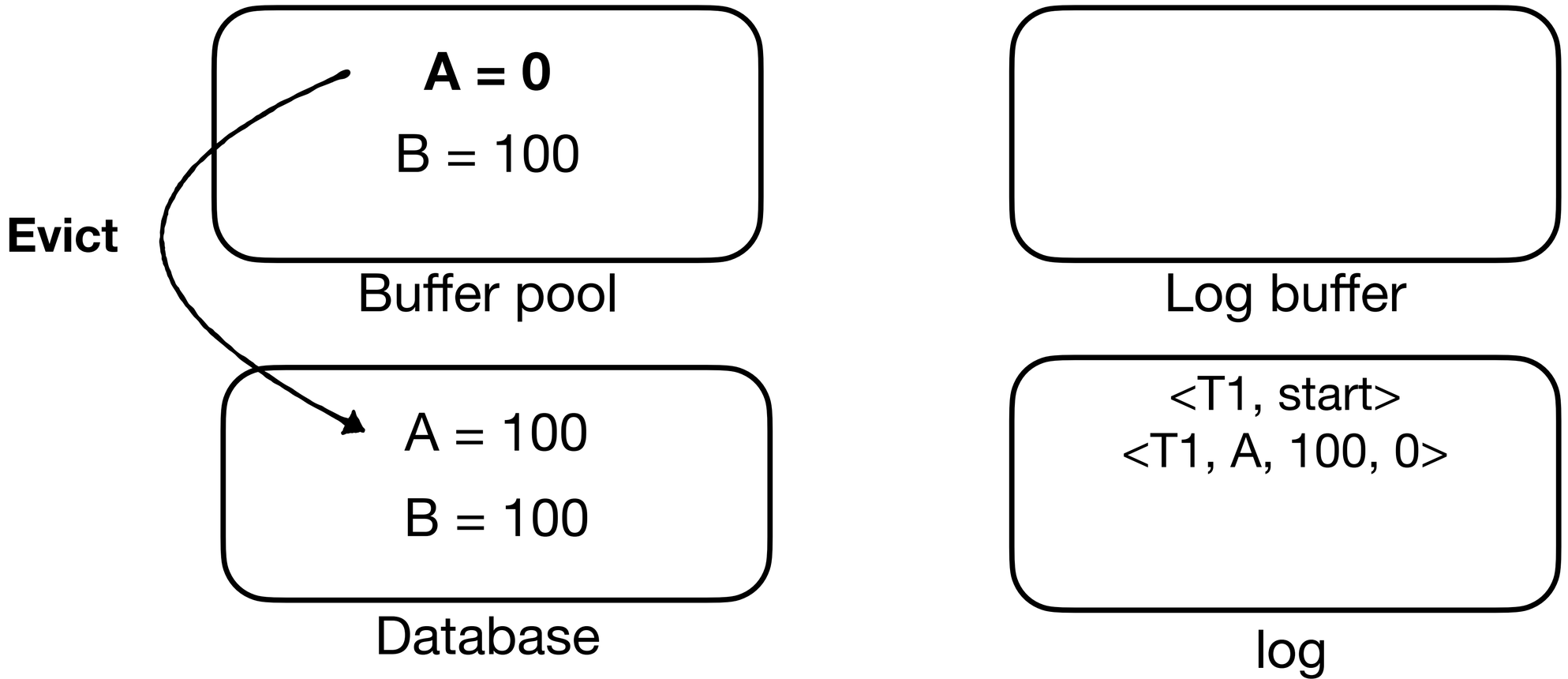
Undo logging:

Transaction T1

A = A - 100

B = B + 100

We are now allowed to evict A if buffer pool needs to



Undo logging:

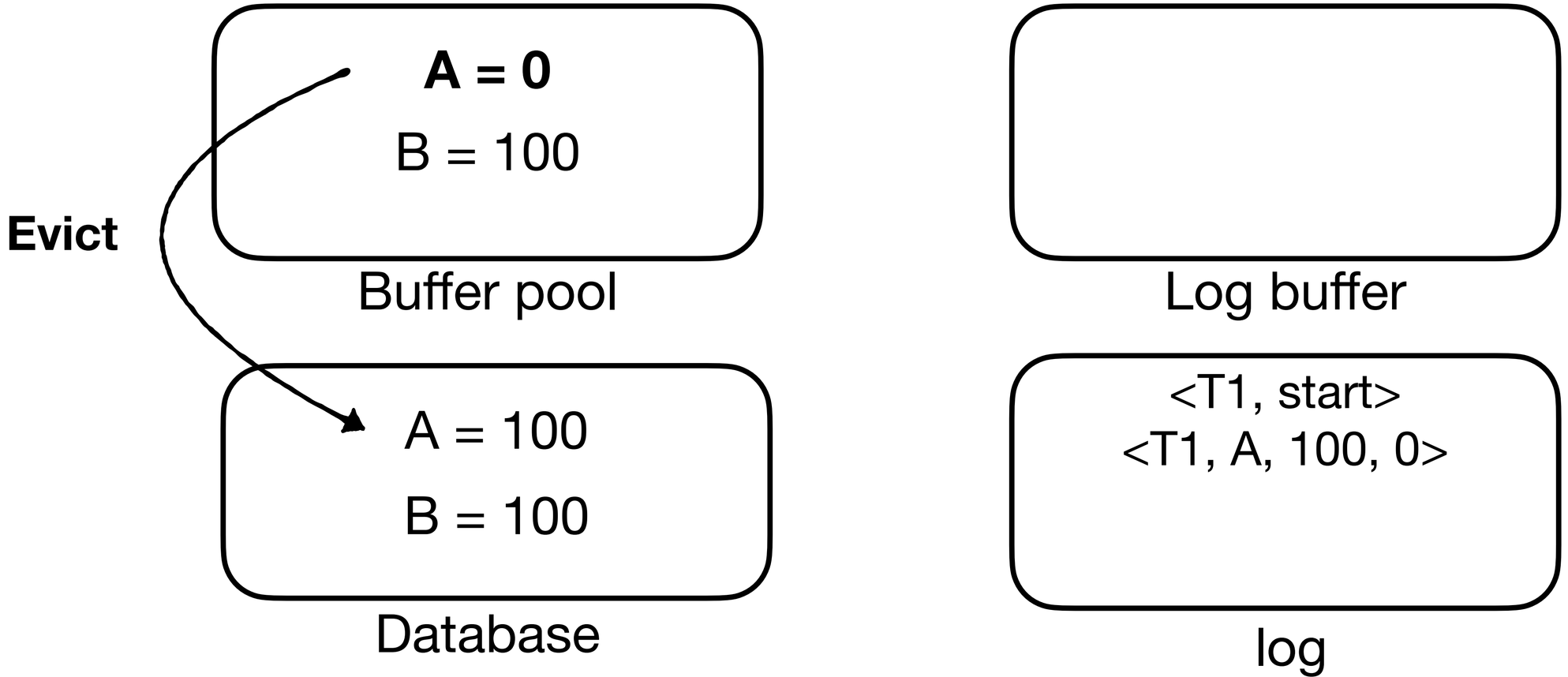
Transaction T1

A = A - 100

B = B + 100

We are now allowed to evict A if buffer pool needs to

we do not need to keep changed data in memory like undo logging :)



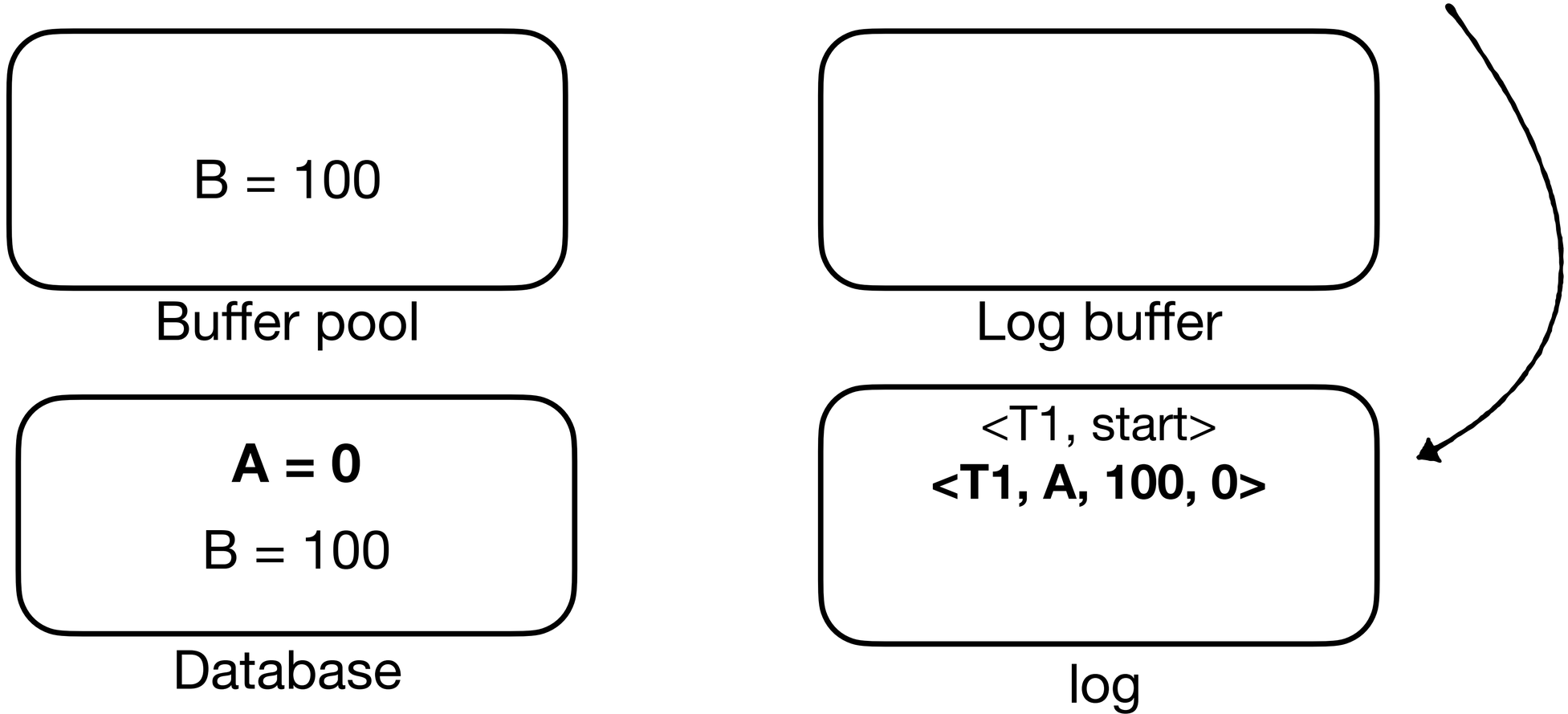
Undo logging:

Transaction T1

A = A - 100

B = B + 100

If power fails at this point, we could undo change to A using log



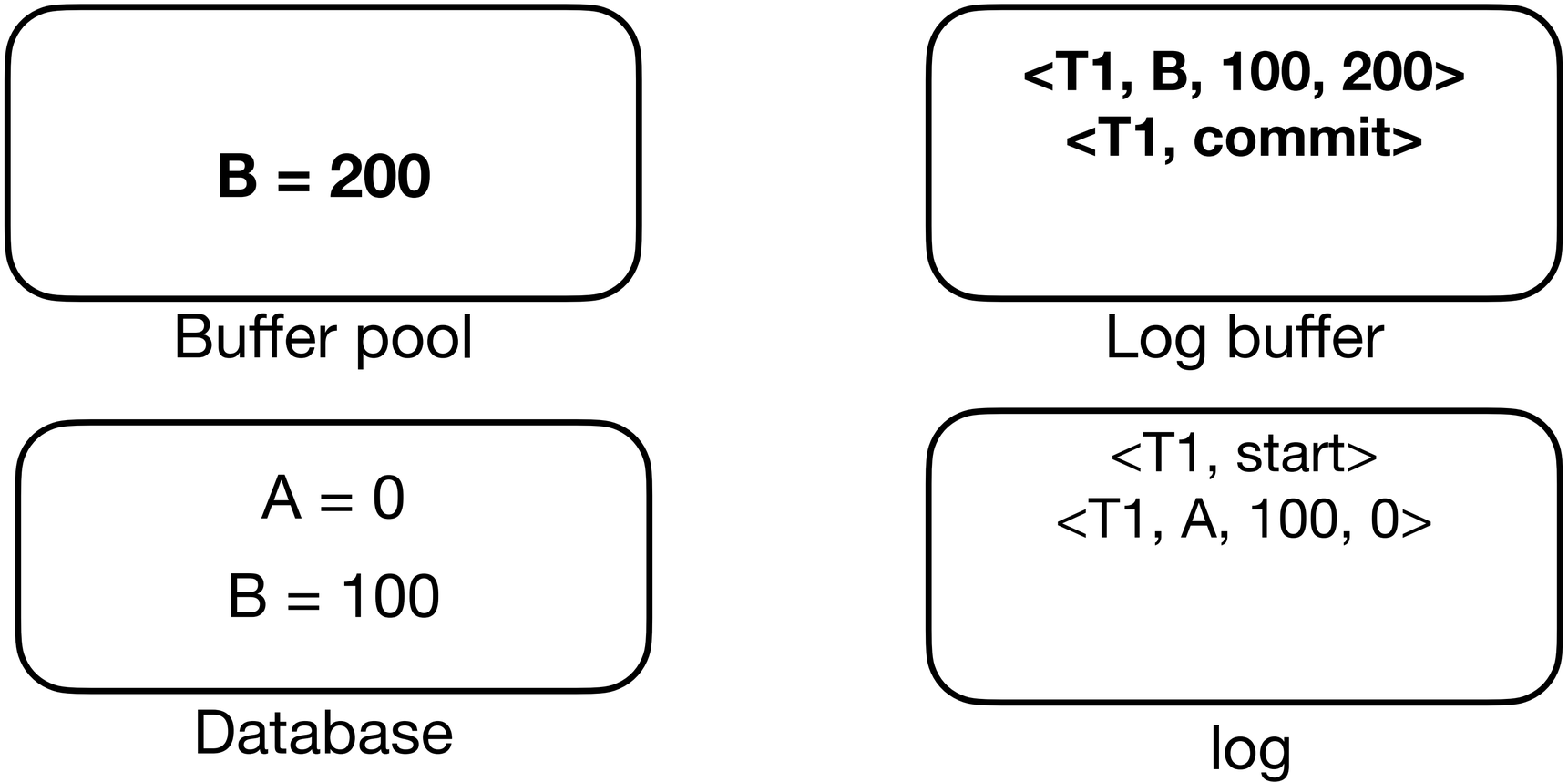
Undo logging:

Transaction T1

A = A - 100

B = B + 100

Otherwise, continue with the transaction

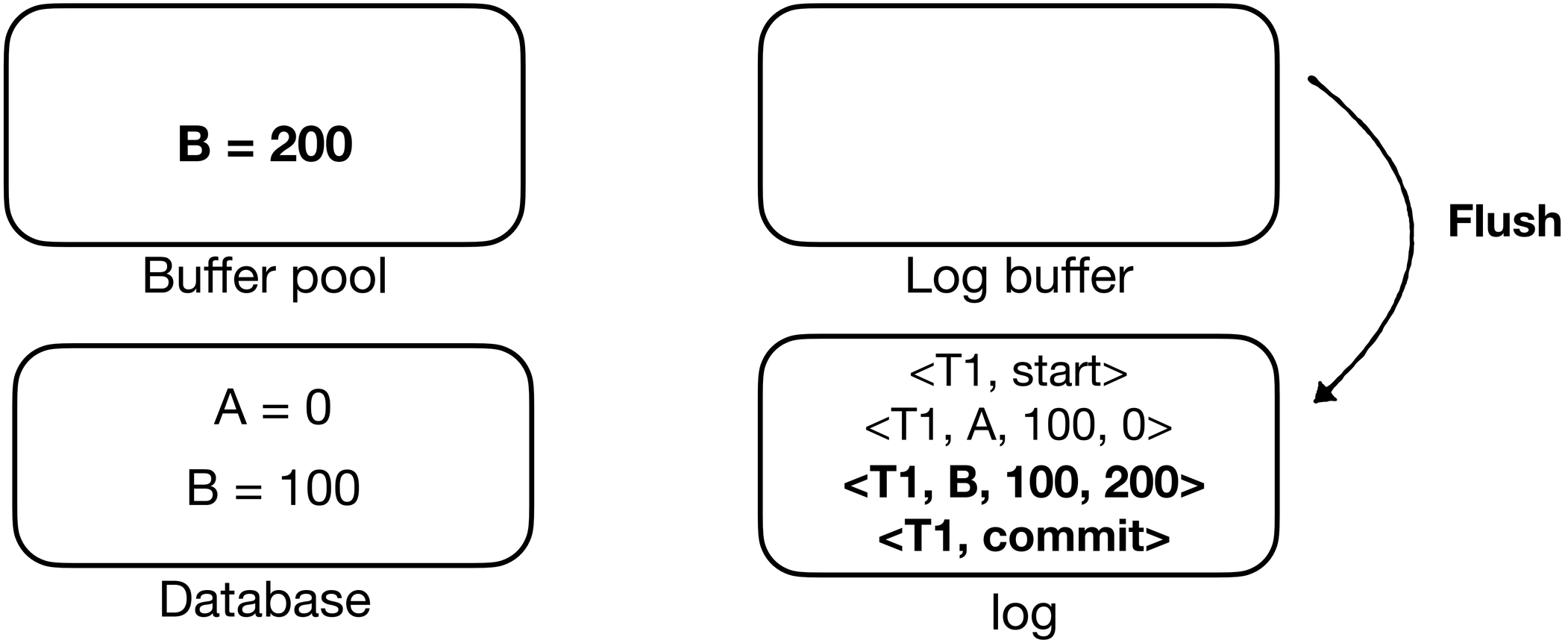


Undo logging:

Transaction T1

A = A - 100

B = B + 100



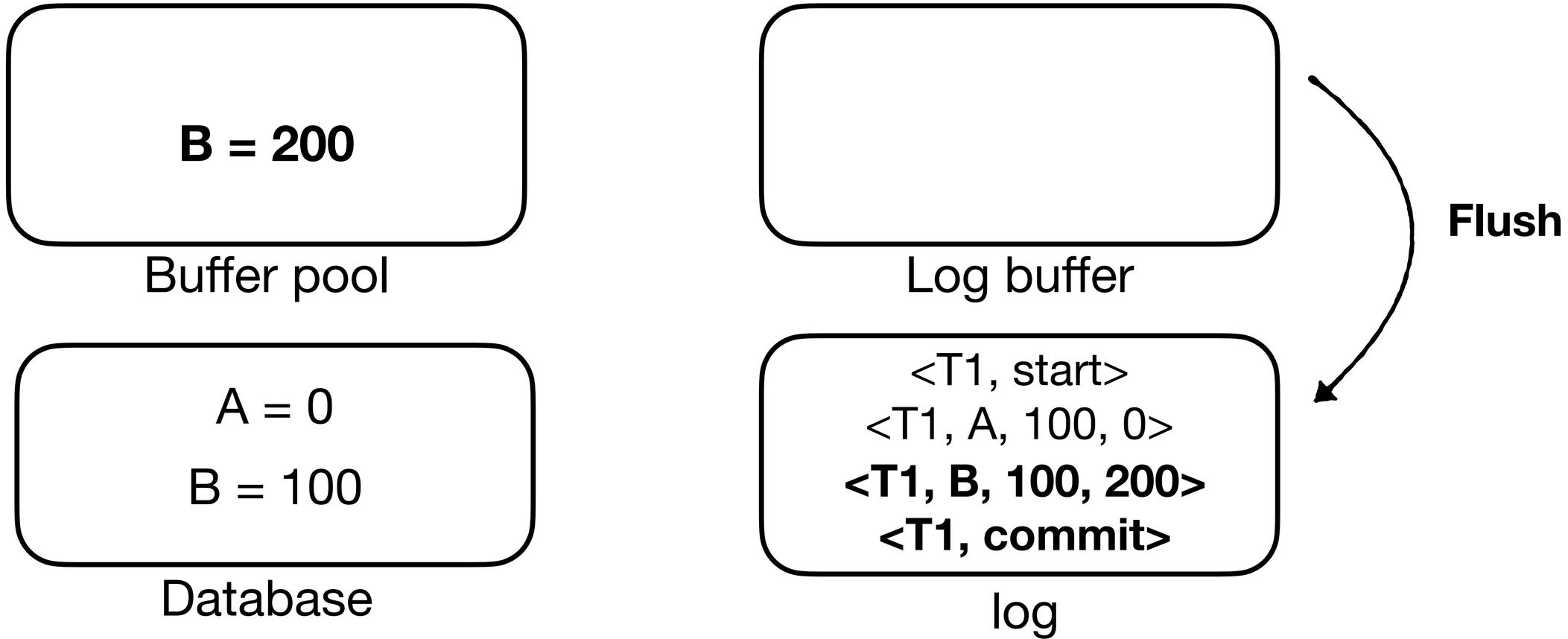
Undo logging:

Transaction T1

A = A - 100

B = B + 100

Do do not now need to force changed data into storage like undo logging :)



Undo logging:

Transaction T1

A = A - 100

B = B + 100

Suppose power now fails before we save B to storage

B = 200

Buffer pool

Log buffer

A = 0

B = 100

Database

<T1, start>
<T1, A, 100, 0>
<T1, B, 100, 200>
<T1, commit>

log

Undo logging:

Transaction T1

A = A - 100

B = B + 100

Suppose power now fails before we save B to storage

How to recover B?

Buffer pool

Log buffer

A = 0

B = 100

Database

<T1, start>
<T1, A, 100, 0>
<T1, B, 100, 200>
<T1, commit>

log

Undo logging:

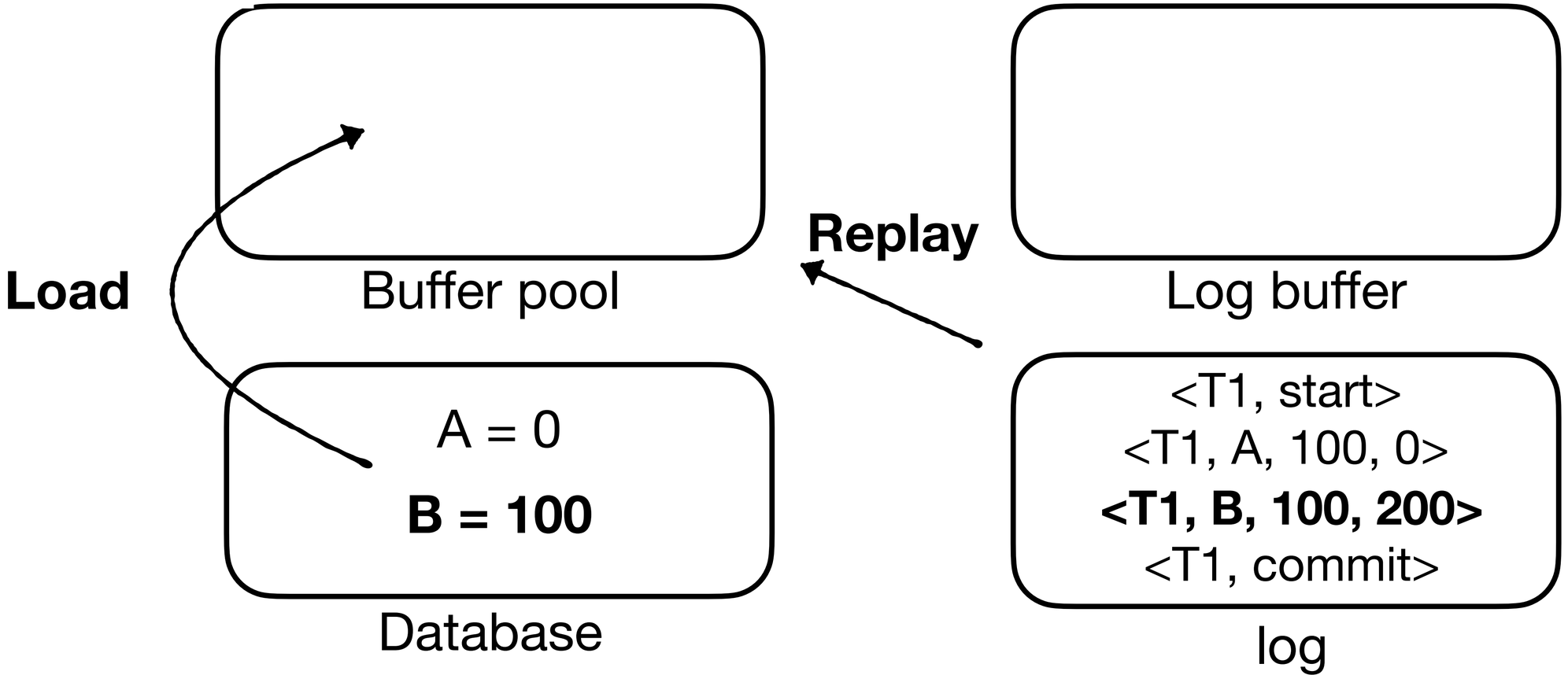
Transaction T1

A = A - 100

B = B + 100

Suppose power now fails before we save B to storage

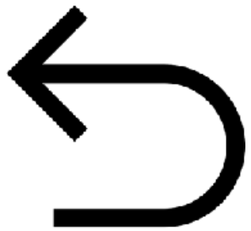
How to recover B?



Undo/Redo logging rules

(1) write before and after-image for any changed value to log buffer
(2) before modifying item on disk, must flush log record
(3) when transaction is finished, flush commit record

Undo

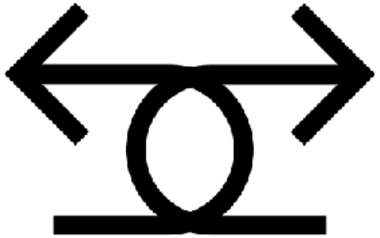


random I/Os

Redo

holds memory

Redo/Redo



Address both problems!

Tutorial on Recovery
beings now :)