

Transactions and Recovery

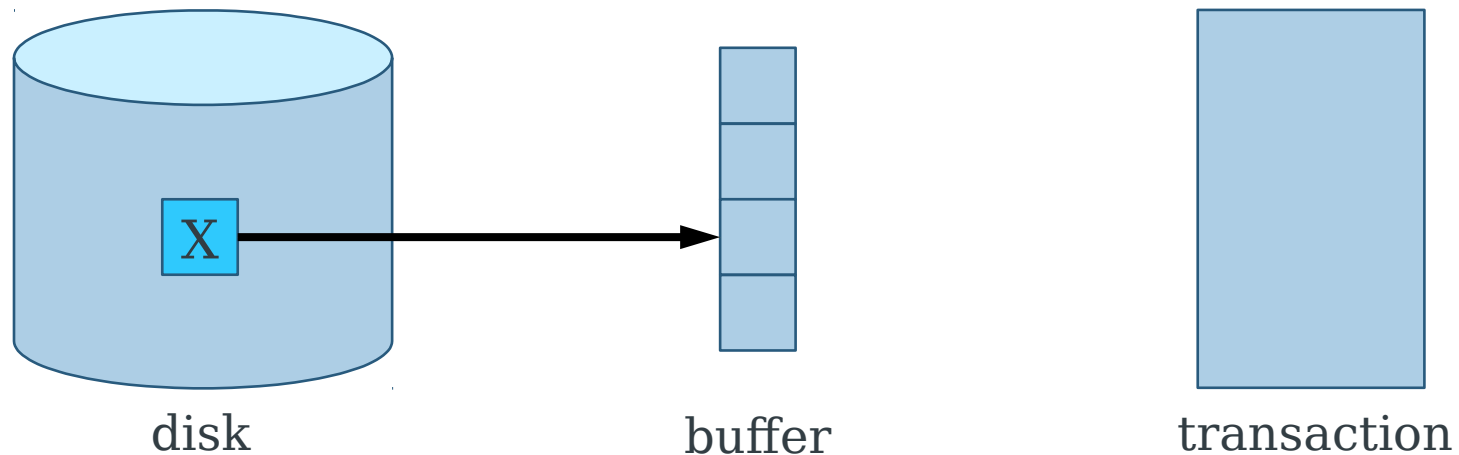
COMP3211 Advanced Databases

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk
2014-2015

Durability

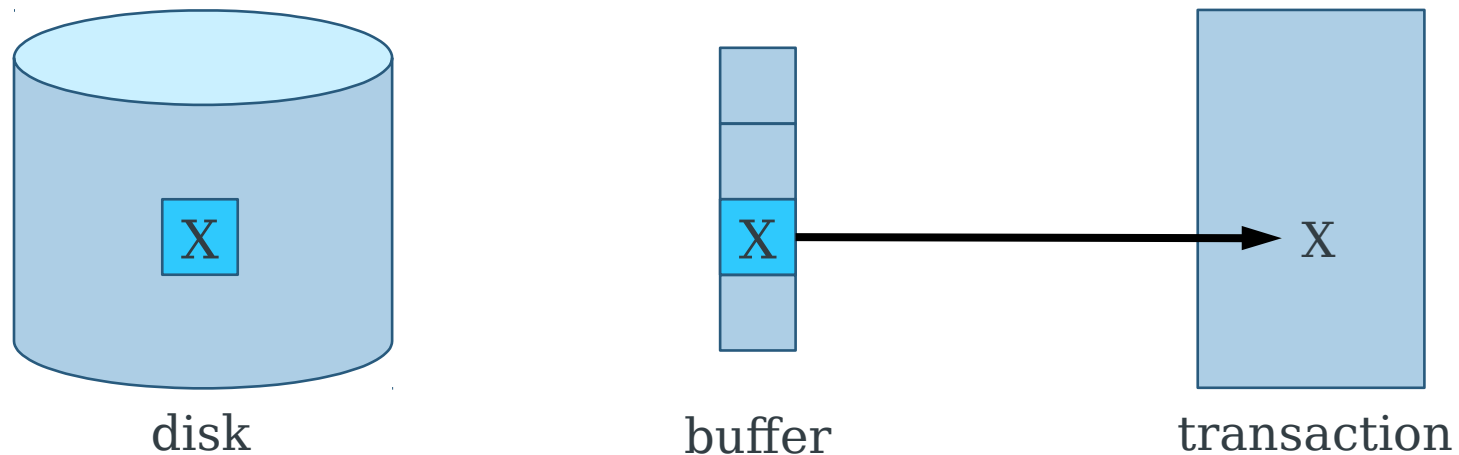
Once a database is changed and committed, changes should not be lost because of failure

input(X)



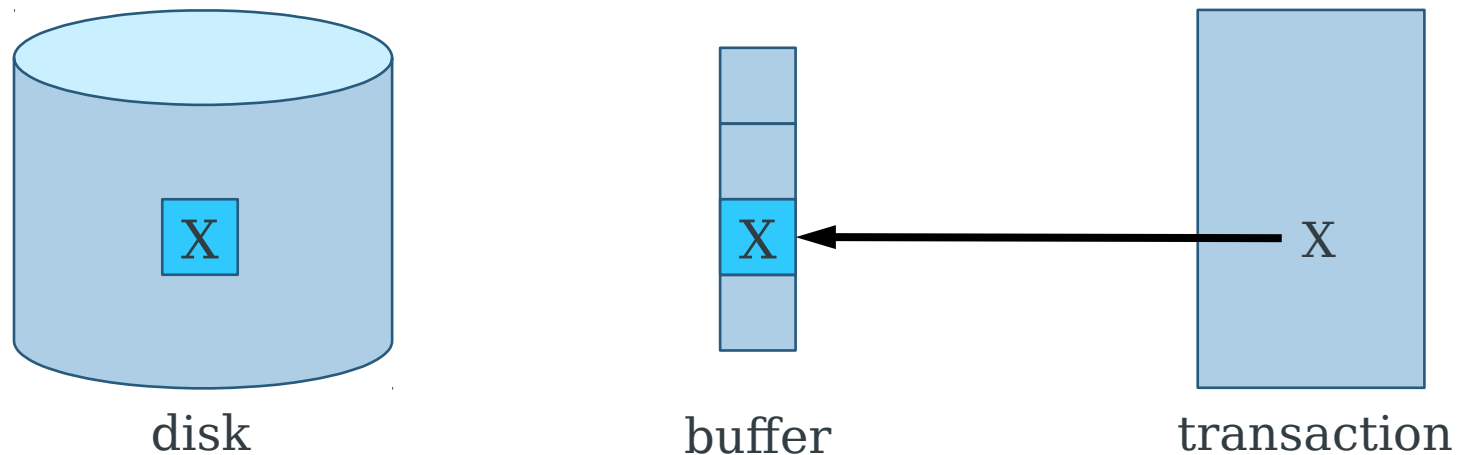
Copy the disk block containing
database item X into a memory
buffer

read(X)



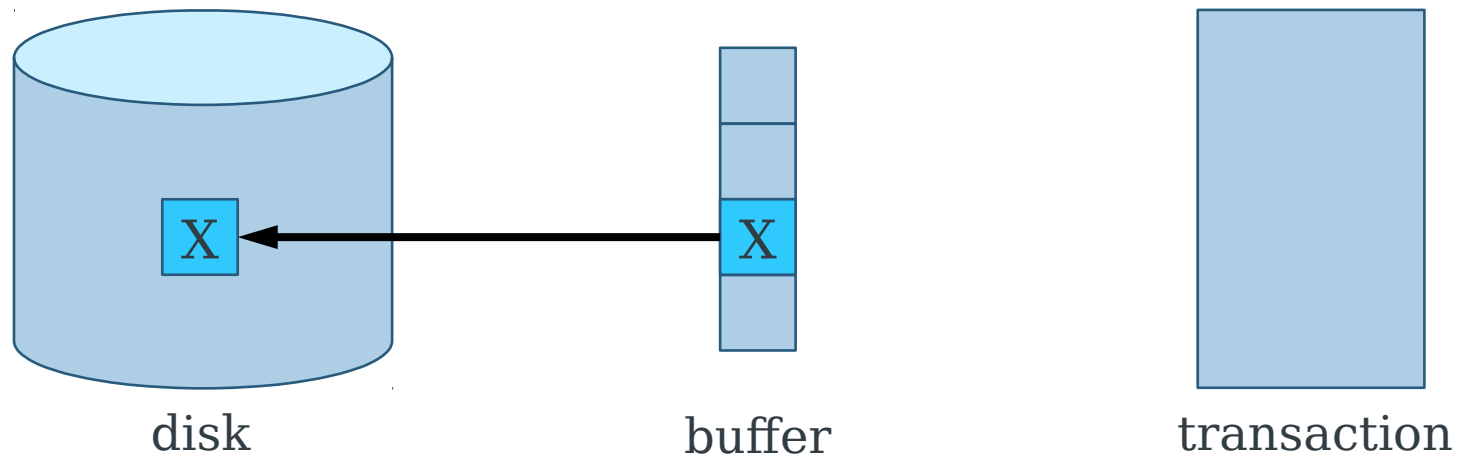
Read a database item X into a local variable. If the block containing X is not already in a memory buffer, first $\text{input}(X)$

write(X)



Write the value of local variable into
database item X in a memory buffer

output(X)



Copy the block containing X from
memory buffer to disk

Expanded Transaction

```
read(X)
X := X - 10
write(X)
read(Y)
Y := Y + 10
write(Y)
output(X)
output(Y)
```

Expanded Transaction

Action	X	Y	X _m	Y _m	X _d	Y _d	Log
			20	50			
read(X)	20		20		20	50	
X := X - 10		10		20		20	50
write(X)	10		10		20	50	
read(Y)	10	50	10	50	20	50	
Y := Y+10	10	60	10	50	20	50	
write(Y)	10	60	10	60	20	50	
output(X)	10	60	10	60	10	50	
output(Y)	10	60	10	60	10	60	

Logging

Logging

Main approach to recovering from a system crash relies on a persistent record of changes made during a transaction

Append-only files used by log manager to record events

Three main approaches to logging:

- Undo Logging
- Redo Logging

Undo/Redo Logging

Log Records

<start T >

Transaction T has started execution

<commit T >

Transaction T has completed successfully and will make no further changes to database items

<abort T >

Transaction T could not complete successfully. No changes made by T will be copied to disk.

Undo Logging

Undo Logging

Repair a database following a system crash by undoing the effects of transactions that were incomplete at the time of the crash

Introduces a new record type to record changes:

$\langle T, X, old \rangle$

Transaction T has changed database item X from its old value

Undo Logging Rules

U1: If transaction T modifies database item X, then a log record of the form $\langle T, X, \text{old} \rangle$ must be written to disk **before** the new value of X is written to disk

U2: If a transaction T commits, then its $\langle \text{commit } T \rangle$ log record must be written to disk only **after** all database items changed by T have been written to disk (but then as soon as possible)

Transaction with Undo Logging

Action	X	Y	X _m	Y _m	X _d	Y _d	Log
			20	50			<start T>
read(X)	20		20		20	50	
X := X - 10		10		20		20	50
write(X)	10		10		20	50	<T, X, 10>
read(Y)	10	50	10	50	20	50	
Y := Y+10	10	60	10	50	20	50	
write(Y)	10	60	10	60	20	50	<T, Y, 60>
flush log							
output(X)	10	60	10	60	10	50	
output(Y)	10	60	10	60	10	60	
							<commit T>
flush log							

Recovery with Undo Logging

```
for each log entry <T, X, old>, scanning backwards {  
    if <commit T> has been seen {  
        do nothing  
    } else {  
        change the value of X in the database back to  
old  
    }  
}  
for each incomplete transaction T (that was not  
aborted) {  
    write <abort T> to log  
}  
flush log
```


Checkpointing with Undo Logging

Disadvantage of this approach: we must scan the entire log

Introduce a periodic checkpoint in the log

- Before checkpoint, all transactions have committed or aborted
- Only need search backwards through the log to the most recent checkpoint

New log record type:

<ckpt>

The database has been checkpointed

Checkpointing

1. Stop accepting new transactions
2. Wait until all active transactions commit/abort and write `<commit T>/<abort T>` to the log
3. flush log
4. write `<ckpt>` to log
5. flush log
6. Resume accepting transactions

Nonquiescent Checkpointing

Need to stop transaction processing while checkpointing

- System may appear to stall
- Allow new transactions to enter the system during the checkpoint.

New log record types:

<start ckpt (T1...Tn)>

Checkpoint starts. T1...Tn are active transactions that have not yet committed

<end ckpt>

Nonquiescent Checkpointing

1. Write `<start ckpt (T1...Tn)>` to log and flush log
2. Wait until T1..Tn have all committed or aborted
3. Write `<end ckpt>` to log and flush log

Note that new transactions may be started during step 2

Recovery with Checkpointed Undo Logging

Two cases, depending on latest checkpoint log record:

<end ckpt>

- All incomplete transactions began after the previous <start ckpt ()>
- Disregard the log before the previous <start ckpt ()>

<start ckpt (T1...Tn)>

- System crash occurred during checkpoint
- Incomplete transactions are those encountered before the <start ckpt ()> and those of T1...Tn that were not committed before the crash
- Disregard the log before the start of the earliest incomplete

Redo Logging

Issues with Undo Logging

U2: If a transaction T commits, then its <commit T> log record must be written to disk only **after** all database items changed by T have been written to disk (but then as soon as possible)

- Potentially causes more disk i/o operations
- Can we let changes reside in buffer memory for longer?

Redo Logging

Ignore incomplete transactions, repeat changes made by committed transactions

Write $\langle \text{commit } T \rangle$ log record to disk **before** changed values are written to disk

- If no $\langle \text{commit } T \rangle$ records has been written, no changes by T have been written to disk

Introduces a different record type to record changes:

$\langle T, X, \text{new} \rangle$

Transaction T has changed database item X to a new value

Redo-Logging Rule

R1: Before modifying a database item X on disk, all log records related to the modification ($\langle T, X, \text{new} \rangle$, $\langle \text{commit } T \rangle$) must be written to disk

Transaction with Redo Logging

Action	X	Y	X _m	Y _m	X _d	Y _d	Log
			20	50			<start T>
read(X)	20		20		20	50	
X := X - 10		10		20		20	50
write(X)	10		10		20	50	<T, X, 10>
read(Y)	10	50	10	50	20	50	
Y := Y+10	10	60	10	50	20	50	
write(Y)	10	60	10	60	20	50	<T, Y, 60>
							<commit T>
flush log							
output(X)	10	60	10	60	10	50	
output(Y)	10	60	10	60	10	60	

Recovery with Redo Logging

identify the committed transactions

```
for each log entry  $\langle T, X, new \rangle$ , scanning forwards {  
    if  $T$  is not committed {  
        do nothing  
    } else {  
        write value  $new$  for  $X$  to the database  
    }  
}  
for each incomplete transaction  $T$  {  
    write  $\langle \text{abort } T \rangle$  to log  
}  
flush log
```

Checkpointing with Redo Logging

1. Write log record $\langle \text{start ckpt } (T1..Tn) \rangle$, where $T1...Tn$ are uncommitted, and flush log
2. Write to disk all database items that have been written to buffers but not yet to disk, by transactions that have already committed
3. Write log record $\langle \text{end ckpt} \rangle$ and flush log

Recovery with Checkpointed Redo Logging

As with checkpointed undo logging, two cases:

<end ckpt>

- Every value written by transactions that committed before the corresponding <start ckpt ()> has been written to disk - ignore
- Any transaction named in the checkpoint start, or which has started since, may have changes that have not been written to disk (even if the transaction has committed)

Recovery with Checkpointed Redo Logging

As with checkpointed undo logging, two cases:

<start ckpt (T1...Tn)>

- Can't tell whether committed transactions prior to this checkpoint had their changes written to disk
- Search back to the previous <end ckpt>, find its corresponding <start ckpt ()> and treat as before

Undo/Redo Logging

Undo/Redo Logging

Aims to address issues in both undo and redo logging

- Undo logging may increase number of disk i/o operations
- Redo logging requires that all modified blocks be kept in buffers until the transaction commits and the logs flushed

Introduces a different record type to record changes:

$\langle T, X, old, new \rangle$

Transaction T has changed database item X from an old to a new value

Undo/Redo Logging Rules

UR1: Before transaction T modifies any database item X on disk, the update record $\langle T, X, \text{old}, \text{new} \rangle$ must be written to disk

UR2: A $\langle \text{commit } T \rangle$ record must be flushed to disk as soon as it is written to the log

Note: the $\langle \text{commit } T \rangle$ log record may come before or after any of the changes on disk

Transaction with Undo/Redo Logging

Action	X	Y	X _m	Y _m	X _d	Y _d	Log
			20	50			<start T>
read(X)	20		20		20	50	
X := X - 10		10		20		20	50
write(X)	10		10		20	50	<T, X, 20, 10>
read(Y)	10	50	10	50	20	50	
Y := Y+10	10	60	10	50	20	50	
write(Y)	10	60	10	60	20	50	<T, Y, 50, 60>
flush log							
output(X)	10	60	10	60	10	50	
							<commit T>
flush log							
output(Y)	10	60	10	60	10	60	

Recovery with Undo/Redo Logging

1. Redo all committed transactions from oldest to newest
2. Undo all incomplete transactions from newest to oldest

Checkpointing with Undo/Redo Logging

1. Write <start ckpt (T1...Tn)> to log and flush log
2. Write to disk all dirty buffers (i.e. those with one or more changed database items, not just those from committed transactions)
3. Write <end ckpt> to log and flush log