Coping With System Failure

Chapter 17 of GUW

Objectives

- To understand how data can be protected in the face of system failure
  - Techniques for supporting the goal of resilience, that is, the integrity of data when the system fails in some way.

- Lecture outline

- Issues and Models for Resilient Operations
- Logging
  - Undo Logging
  - Redo Logging
  - Undo/Redo Logging
- Protecting Against Media Failures

- Issues and Models for Resilient Operations

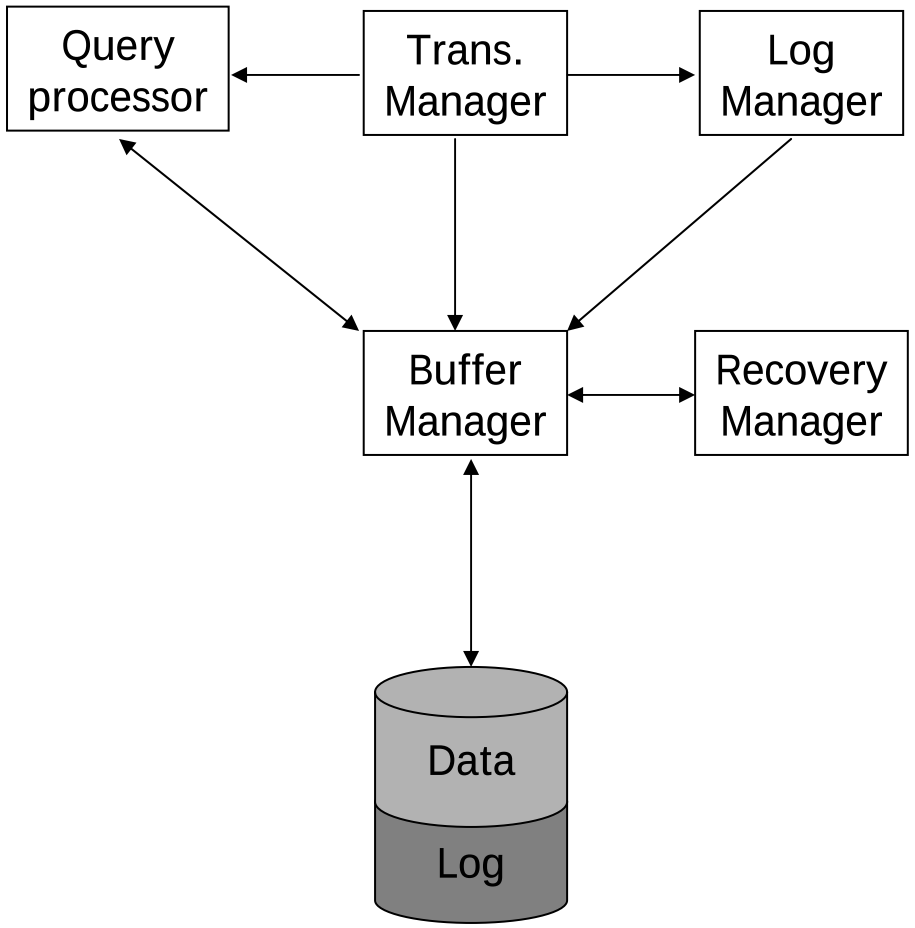
- Failure modes
- Transactions
- Correct execution of transactions
- The primitive operations of transactions

-- Failure Modes

- Erroneous data entry.
  - Example: Mistyping a value
  - Solution: Constraints
- System failures
  - Example: Lost state of transaction. May be due power outrage.
  - Solution: logging
- Media failure
  - Example: Bad sector in a disk or disk head crash
  - Solution: RAID, Archiving, and redundant copies
- Catastrophic failure
  - Fire
  - Solution: Archiving and redundancy

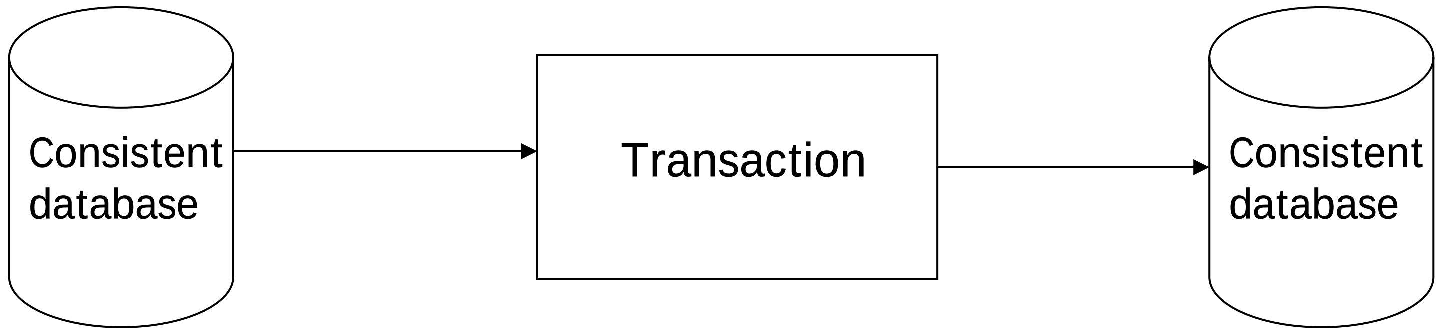
-- Transactions

- Is a unit of execution of DB operations.
- Atomic
- Specified by Programmer
  - Transaction ends with a COMMIT or ROLLBACK commands
- Assuring transactions are executed correltly is the job of transaction manager which:
  - Issues signals to the log manager
  - Concurrently executing transactions do not intefere.



--- Correct execution of transactions

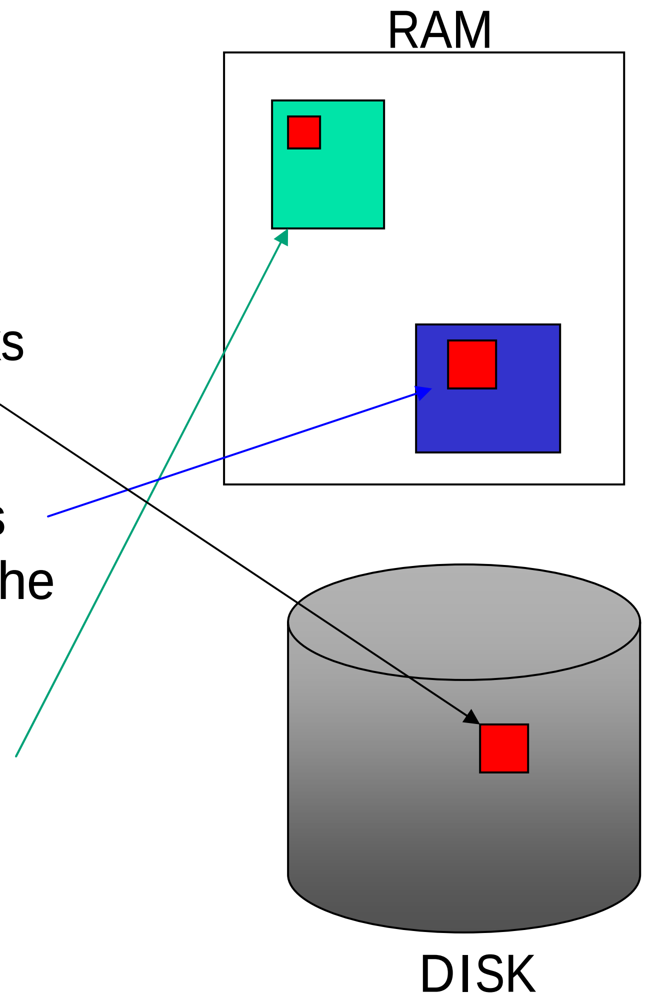
- A fundamental assumption of a transaction is the ***correctness principle***, which is
  - If a transaction executes in the absence of any other transaction or system error, and its starts with the DB in a consistent state, then the DB is in a consistent state when the transaction ends.



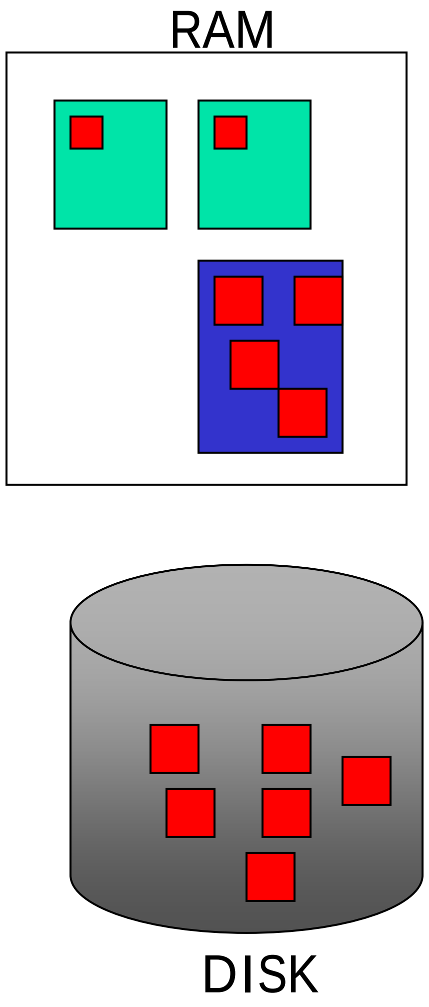
- No other transaction
- No system error

--- The Primitive Operations of Transactions ...

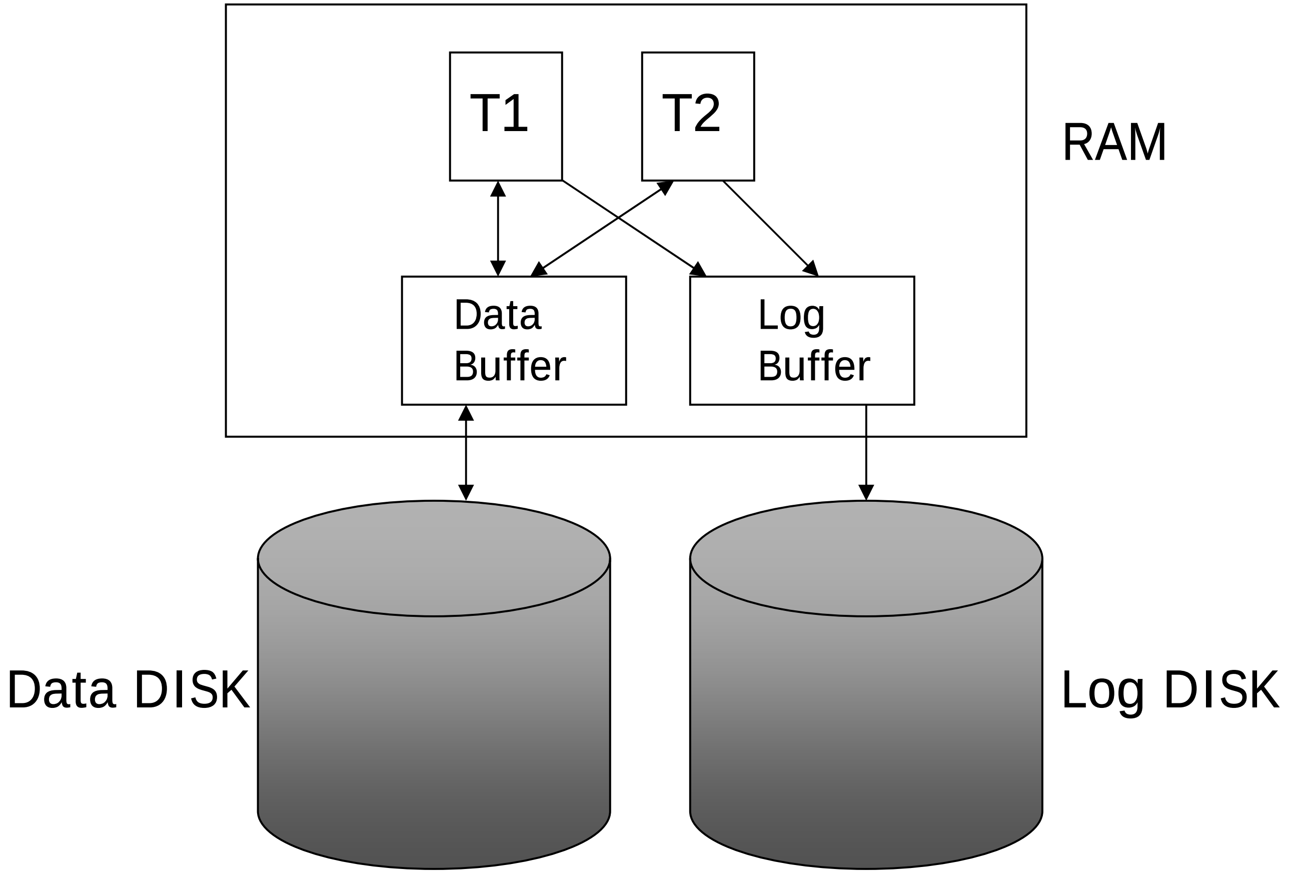
- The three address spaces that interact during a transaction are:
  - The space of the disk blocks holding the DB elements.
  - The memory space address space that is managed by the buffer manager
  - The local address space of the transaction



... --- The Primitive Operations of Transactions ...



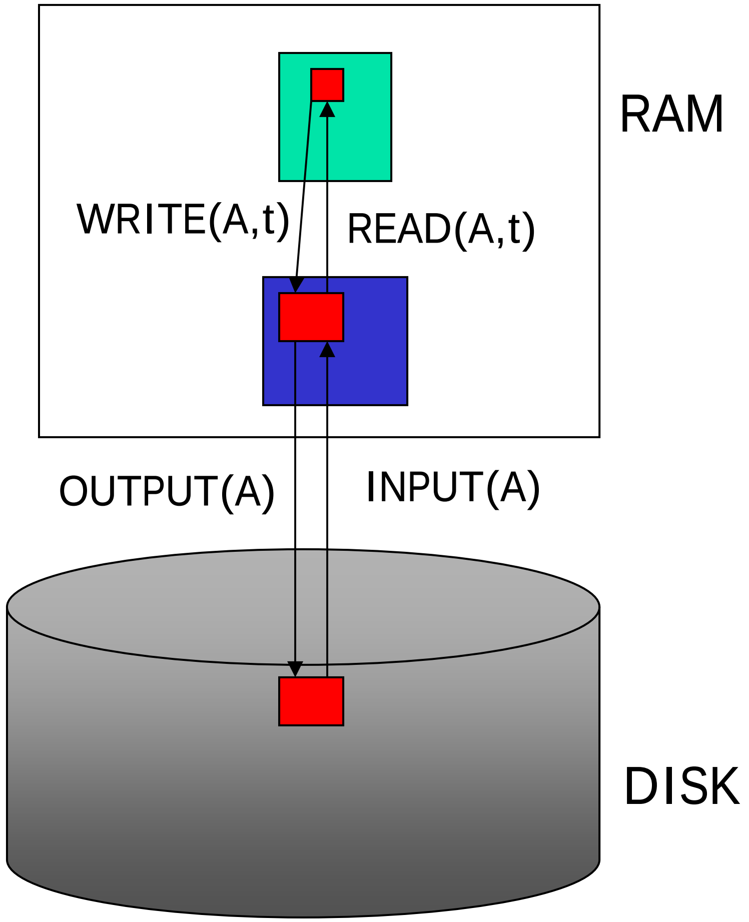
... --- The Primitive Operations of Transactions ...



--- The Primitive Operations of Transactions ...

- During a transaction, the operations that move data between the above address spaces are:
  - **INPUT(X):** Copy the disk block containing X to memory buffer.
  - **READ(X,t):** Copies element X to the transaction local variable t.
  - **WRITE(X,t):** Copy the transaction variable t to DB element X in a memory buffer.
  - **OUTPUT(X):** Copy the block containing X from its buffer to disk.

--- The Primitive Operations of Transactions ...



-- Logging ...

- As transaction executes, the log manager has the job of recording in the log each important event.
- Each event is represented as log record and saved into a log file.
- Log records from different transactions are interleaved in the log file.
- New log records are only appended at the end of the log file.

... -- Logging

- There are several forms of log record that are used with each type of logging. These are:
  - <START T>: Transaction T has began.
  - <COMMIT T>: Transaction T has completed successfully.
  - <ABORT T>: Transaction T could not complete succesfully.
    - The changes of B must be rolled back.

--- Undo Logging

- Used for recovery from failure
- Beside the previously mentioned log records, an update log record <T, X, v> is needed, where
    - T is transaction
    - X is the database element that was changed by T
    - v is a former value of X.

---- Undo Logging Rules

1. If transaction T modifies DB element X, then the log record of the form <T, X, v> must be written to the disk before the new value of X is written to the disk.
2. It a transaction commits, then the COMMIT log record must be written disk only after all the database elements changed by the transaction have being written to disk, but as soon there after as possible.

---- Undo Logging: Example

Step	Action	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A,t)	8	8		8	8	
3	t := t * 2	16	8		8	8	
4	WRITE(A,t)	16	16		8	8	<T, A, 8>
5	READ(B,t)	8	16	8	8	8	
6	t := t * 2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 8>
8	FLUSH_LOG						
9	OUTPUT(A)	16	16	16	16	8	
10	OUTPUT(B)	16	16	16	16	16	
11							<COMMIT T>
12	FLUSH_LOG						

---- Recovery Using Undo Logging

- Start the log from the end
- If T is a transaction whose COMMIT record is available then do nothing for T.
- Otherwise, T is an incomplete transaction, or an aborted transaction. The recovery manager must change the values of X into v, incase X has been altered before the crash.
- After making these changes, the recovery manager must right a log record <ABORT T> for each incomplete transaction T, that was not previously aborted.

---- Checkpointing with Undo Logging

- As observed recovery requests the entire log be examined, which is bad.
- The solution: checkpoint
  1. Stop accepting new transactions
  2. Wait until all currently active transactions commit or abort and have written a COMMIT or ABORT record on the log
  3. Flush the log to disk
  4. Write a log record <CKPT>, and flash the log again
  5. Resume accepting transactions
- With such Checkpointing, no need to recover transactions executed prior to checkpoint.

---- Nonquiescent Checkpointing with Undo Logging

- With nonquiescent no need to stop new transactions during checkpointing, hence preferred.
- Steps of nonquiescent checkpointing:

1. Write a log record <START CKPT (T1, T2, ..., Tn)> and flush the log. T1, T2, ..., Tn are the active transactions
2. Wait until all of T1, T2, ..., Tn commit or abort, but do not prohibit other transactions from starting
3. When all of T1, T2, ..., Tn have completed, write a log record <END CKPT> and flush the log.

---- Recovery with Nonquiescent Checkpointing

- Go back words starting from the end of the log
  - If <END CKPT> is encountered first go back until the last <START CKPT>.
  - But if <START CKPT> is encountered first go until the one before the last <START CKPT>.

--- Redo Logging

- In redo logging the meaning of <T, X, w> is, Transaction T wrote the new value w for DB element X.
- Rule
  - Before modifying any DB element X on disk, it is necessary that all log records pertaining to this modification of X, including both the <T, X, w> and the <COMMIT T> log records, must appear on disk.

---- Redo Logging: Example

Step	Action	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A,t)	8	8		8	8	
3	t := t * 2	16	8		8	8	
4	WRITE(A,t)	16	16		8	8	<T, A, 16>
5	READ(B,t)	8	16	8	8	8	
6	t := t * 2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 16>
8							<COMMIT T>
9	FLUSH_LOG	16	16	16	16	8	
10	OUTPUT(A)	16	16	16	16	16	
11	OUTPUT(B)						

---- Recovery with Redo Logging

1. Identify the committed transactions.
2. Scan the log forward from the beginning. For each log record <T, X, w> encountered.
   a. If T is not a committed transaction, do nothing.
   b. If T is committed, write v for DB element X.
3. For each incomplete transaction T, write an <ABORT T> record to the log and flush the log.

---- Checkpoint with Redo Logging

- Steps:
  - Write a log record <START CKPT (T1, T2, ..., Tn)>, where T1, T2, ..., Tn are the currently active transactions, and flush the log
  - Write to disk all DB elements that were written to buffers but yet not to disk by transactions that had already committed when the SATRT CKPT record was written to the log.
  - Write an <END CKPT> record to the log and flush the log.

--- Undo/Redo Logging

- Except for the update record, the other log records are the same as that of redo and undo loggings.
- The update log record has 4 components:<T, X, v, w> where:
  - T is the transaction
  - X is the DB element
  - v is the old value of X
  - w is the new value of X
- Rule:
  - Before modifying any DB element X on disk because of change made by some transaction T, it is necessary that the update record <T, X, v, w> appear on disk.

---- Undo/Redo Logging: Example

Step	Action	t	M-A	M-B	D-A	D-B	LOG
1							<START T>
2	READ(A,t)	8	8		8	8	
3	t := t * 2	16	8		8	8	
4	WRITE(A,t)	16	16		8	8	<T, A, 16>
5	READ(B,t)	8	16	8	8	8	
6	t := t * 2	16	16	8	8	8	
7	WRITE(B,t)	16	16	16	8	8	<T, B, 16>
8	FLUSH_LOG	16	16	16	16	8	
9	OUTPUT(A)	16	16	16	16	16	
10							<COMMIT T>
11	OUTPUT(B)						

---- Recovery with Undo/Redo Logging

1. Redo all committed transactions in the order earliest-first
2. Undo all the incomplete transactions in the order latest-first.

---- Checkpointing with Undo/Redo Logging

- Steps:
  - Write a log record <START CKPT (T1, T2, ..., Tn)>, where T1, T2, ..., Tn are the currently active transactions, and flush the log.
  - Write to buffer all dirty buffers. Unlike redo logging we flush all buffers not just those written by the committed transactions.
  - Write an <END CKPT> record to the log, and flush the log.

- Protecting Against Media Failure

- To recover from media failure
  - Archive all the logs.
  - Make periodic backup (dump) of the whole database.
- Recovery steps:
  - Restore the DB from the archive
    - Find the most recent full dump and reconstruct the DB from.
    - If there are later incremental dumps, modify the DB accordingly to each, earliest first
  - Modify the DB using the surviving logs. Use the method of recovery appropriate to the log method being used.

- Reference

- Chapter 17 of GUW

END