Lecture 15-16: Recovery

Wednesday-Friday, May 2-4, 2007
Announcements

Homework 3:

• Attributes v.s. elements: /item v.s. /@item
• Data is not clean
  – OK to return any sensible answer, no need to clean
• See the two examples in the mini-tutorial (e.g. fn:string)
• Check the lecture notes (e.g. for group-by)
• If query doesn’t work, try a simpler one to debug
Outline

• Undo logging 17.2
• Redo logging 17.3
• Redo/undo 17.4
Transaction Management

Two parts:

• Recovery from crashes: ACID
• Concurrency control: ACID

Both operate on the buffer pool
Recovery

From which of the events below can a database actually recover?

- Wrong data entry
- Disk failure
- Fire / earthquake / bankruptcy / ….
- Systems crashes
## Recovery

<table>
<thead>
<tr>
<th>Type of Crash</th>
<th>Prevention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrong data entry</td>
<td>Constraints and Data cleaning</td>
</tr>
<tr>
<td>Disk crashes</td>
<td>Redundancy: e.g. RAID, archive</td>
</tr>
<tr>
<td>Fire, theft, bankruptcy…</td>
<td>Buy insurance, Change jobs…</td>
</tr>
<tr>
<td>System failures: e.g. power</td>
<td>DATABASE RECOVERY</td>
</tr>
</tbody>
</table>

*Most frequent*
System Failures

• Each transaction has *internal state*

• When system crashes, internal state is lost
  – Don’t know which parts executed and which didn’t

• Remedy: use a **log**
  – A file that records every single action of the transaction
Transactions

• Assumption: the database is composed of **elements**
  – Usually 1 element = 1 block
  – Can be smaller (=1 record) or larger (=1 relation)

• Assumption: each transaction reads/writes some elements
Primitive Operations of Transactions

- **READ(X,t)**
  - copy element X to transaction local variable t
- **WRITE(X,t)**
  - copy transaction local variable t to element X

- **INPUT(X)**
  - read element X to memory buffer
- **OUTPUT(X)**
  - write element X to disk
Example

START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t)
COMMIT;

Atomicity:
BOTH A and B are multiplied by 2
<table>
<thead>
<tr>
<th>Action</th>
<th>t</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT(A)</td>
<td></td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>
Crash occurs after OUTPUT(A), before OUTPUT(B)
We lose atomicity
The Log

• An append-only file containing log records
• Note: multiple transactions run concurrently, log records are interleaved
• After a system crash, use log to:
  – Redo some transaction that didn’t commit
  – Undo other transactions that didn’t commit
• Three kinds of logs: undo, redo, undo/redo
Undo Logging

Log records

- **<START T>**
  - transaction T has begun
- **<COMMIT T>**
  - T has committed
- **<ABORT T>**
  - T has aborted
- **<T,X,v>**
  - T has updated element X, and its *old* value was v
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT(A)</td>
<td></td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>Action</td>
<td>T</td>
<td>Mem A</td>
<td>Mem B</td>
<td>Disk A</td>
<td>Disk B</td>
<td>Log</td>
</tr>
<tr>
<td>------------</td>
<td>----</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>INPUT(A)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
</tbody>
</table>

WHAT DO WE DO?
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT(A)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
</tbody>
</table>

WHAT DO WE DO?
After Crash

• In the first example:
  – We UNDO both changes: A=8, B=8
  – The transaction is atomic, since none of its actions has been executed

• In the second example
  – We don’t undo anything
  – The transaction is atomic, since both it’s actions have been executed
Undo-Logging Rules

U1: If T modifies X, then <T,X,v> must be written to disk before OUTPUT(X)

U2: If T commits, then OUTPUT(X) must be written to disk before <COMMIT T>

• Hence: OUTPUTs are done early, before the transaction commits
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT(A)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>&lt;COMMIT T&gt;</td>
</tr>
</tbody>
</table>
Recovery with Undo Log

After system’s crash, run recovery manager

• Idea 1. Decide for each transaction T whether it is completed or not
  – <START T>….<COMMIT T>…. = yes
  – <START T>….<ABORT T>…….. = yes
  – <START T>.......................... = no

• Idea 2. Undo all modifications by incomplete transactions
Recovery with Undo Log

Recovery manager:

• Read log from the end; cases:
  <COMMIT T>: mark T as completed
  <ABORT T>: mark T as completed
  <T,X,v>: if T is not completed
           then write X=v to disk
           else ignore
  <START T>: ignore
Recovery with Undo Log

Question 1 in class: Which updates are undone?

Question 2 in class: How far back do we need to read in the log?
Recovery with Undo Log

• Note: all undo commands are *idempotent*
  – If we perform them a second time, no harm is done
  – E.g. if there is a system crash during recovery, simply restart recovery from scratch
Recovery with Undo Log

When do we stop reading the log?

- We cannot stop until we reach the beginning of the log file
- This is impractical

Instead: use checkpointing
Checkpointing

Checkpoint the database periodically
- Stop accepting new transactions
- Wait until all current transactions complete
- Flush log to disk
- Write a `<CKPT>` log record, flush
- Resume transactions
Undo Recovery with Checkpointing

During recovery, Can stop at first <CKPT>

- ... <T9,X9,v9>
- ... <T1,X1,v1>
- ... <T5,X5,v5>
- (all completed) 
- <CKPT>
- <START T2>
- <START T3>
- <START T5>
- <START T4>
- <T4,X4,v4>
- <T5,X5,v5>
- <T1,X1,v1>
- <T3,X3,v3>
- <T2,X2,v2>

transactions T2,T3,T4,T5

other transactions
Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- Idea: nonquiescent checkpointing

Quiescent = being quiet, still, or at rest; inactive
Non-quiescent = allowing transactions to be active
Nonquiescent Checkpointing

- Write a $\text{<START CKPT}(T_1,\ldots,T_k)\text{>}$ where $T_1,\ldots,T_k$ are all active transactions
- Continue normal operation
- When all of $T_1,\ldots,T_k$ have completed, write $\text{<END CKPT>}$
Undo Recovery with Nonquiescent Checkpointing

During recovery, Can stop at first \texttt{<CKPT>}

\texttt{<START CKPT T4, T5, T6>}

... T4, T5, T6, plus later transactions

\texttt{<END CKPT>}

earlier transactions plus T4, T5, T6

T4, T5, T6, plus later transactions

later transactions

Q: why do we need \texttt{<END CKPT>} ?
Redo Logging

Log records

- \(<\text{START T}> = \text{transaction T has begun}\)
- \(<\text{COMMIT T}> = \text{T has committed}\)
- \(<\text{ABORT T}> = \text{T has aborted}\)
- \(<\text{T,X,v}> = \text{T has updated element X, and its new value is v}\)
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,16&gt;</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,16&gt;</td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Redo-Logging Rules

R1: If T modifies X, then both <T,X,v> and <COMMIT T> must be written to disk before OUTPUT(X)

• Hence: OUTPUTs are done \textit{late}
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,16&gt;</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,16&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Recovery with Redo Log

After system’s crash, run recovery manager

• Step 1. Decide for each transaction T whether it is completed or not
  – <START T>….<COMMIT T>…. = yes
  – <START T>….<ABORT T>…….. = yes
  – <START T>………………………… = no

• Step 2. Read log from the beginning, redo all updates of committed transactions
Recovery with Redo Log

<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
...
...
Nonquiescent Checkpointing

- Write a `<START CKPT(T1,…,Tk)>`
  where T1,…,Tk are all active transactions
- Flush to disk all blocks of committed transactions *(dirty blocks)*, while continuing normal operation
- When all blocks have been written, write `<END CKPT>`
Redo Recovery with Nonquiescent Checkpointing

Step 1: look for
The last
<END CKPT>

All OUTPUTs
of T1 are
known to be on disk

Step 2: redo from the earliest start of T4, T5, T6 ignoring transactions committed earlier

All OUTPUTs of T1 are known to be on disk
Comparison Undo/Redo

• Undo logging:
  – OUTPUT must be done early
  – If <COMMIT T> is seen, T definitely has written all its data to
    disk (hence, don’t need to redo) – inefficient

• Redo logging
  – OUTPUT must be done late
  – If <COMMIT T> is not seen, T definitely has not written any of its
    data to disk (hence there is not dirty data on disk, no need to undo)
    – inflexible

• Would like more flexibility on when to OUTPUT: undo/redo logging (next)
Undo/Redo Logging

Log records, only one change

• \(<T,X,u,v> = T \text{ has updated element } X, \text{ its old value was } u, \text{ and its new value is } v>\)
Undo/Redo-Logging Rule

UR1: If T modifies X, then <T,X,u,v> must be written to disk before OUTPUT(X)

Note: we are free to OUTPUT early or late relative to <COMMIT T>
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8,16&gt;</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8,16&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Can OUTPUT whenever we want: before/after COMMIT
Recovery with Undo/Redo Log

After system’s crash, run recovery manager

• Redo all committed transaction, top-down
• Undo all uncommitted transactions, bottom-up
Recovery with Undo/Redo Log

<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
...