Transactional Information Systems:

Theory, Algorithms, and the Practice of Concurrency Control and Recovery

Gerhard Weikum and Gottfried Vossen

© 2002 Morgan Kaufmann
ISBN 1-55860-508-8

“Teamwork is essential. It allows you to blame someone else.” (Anonymous)
Part III: Recovery

• 11 Transaction Recovery
• 12 Crash Recovery: Notion of Correctness
• 13 Page-Model Crash Recovery Algorithms
• 14 Object-Model Crash Recovery Algorithms
• 15 Special Issues of Recovery
• 16 Media Recovery
• 17 Application Recovery
Chapter 13: Page-Model
Crash Recovery Algorithms

• 13.2 Basic Data Structures
  • 13.3 Redo-Winners Paradigm
  • 13.4 Redo-History Paradigm
  • 13.5 Lessons Learned

“History is written by the winners.” (Alex Haley)

“History is a people‘s memory, and without a memory, man is demoted to the lower animals.” (Malcolm X)
Basic Data Structures for Crash Recovery (1)

type Page: record of
    PageNo: identifier;
    PageSeqNo: identifier;
    Status: (clean, dirty) /* only for cached pages*/;
    Contents: array [PageSize] of char;
end;

persistent var StableDatabase: set of Page indexed by PageNo;
var DatabaseCache: set of Page indexed by PageNo;
Basic Data Structures for Crash Recovery (2)

type LogEntry: record of
  LogSeqNo: identifier;
  TransId: identifier;
  PageNo: identifier;
  ActionType:(write, full-write, begin, commit, rollback);
  UndoInfo: array of char;
  RedoInfo: array of char;
  PreviousSeqNo: identifier;
end;
persistent var StableLog:
  ordered set of LogEntry indexed by LogSeqNo;
var LogBuffer:
  ordered set of LogEntry indexed by LogSeqNo;
type TransInfo: record of
  TransId: identifier;
  LastSeqNo: identifier;
end;
var ActiveTrans:
  set of TransInfo indexed by TransId;

Remark: log entries can be physical or physiological
Recall: (Log) Sequence Numbers

Database Cache

- page b: 4215
- page q: 4219
- page p: 3155
- page z: 4217

Log Buffer

- begin(t_{20}): 4220
- write(q, t_{17}): 4219
- write(z, t_{17}): 4217
- commit(t_{19}): 4218
- write(b, t_{17}): 4215
- write(q, t_{19}): 4216
- write(q, t_{19}): 4199
- write(b, t_{17}): 4208

Stable Database

- page b: 4215
- page q: 2788
- page p: 3155
- page z: 4158

Stable Log

- page z: 4217
- page q: 4219
- page z: 4218
- page b: 4215
- page q: 4216
- page q: 4216
- page z: 4215
- page b: 4215
- page b: 4215

Volatile Memory

Stable Storage
Correspondence of Data Structures and Abstract Model

0) action with sequence number \( s \in \text{StableLog} \)
\[ \iff \text{LSN } s \text{ is in StableLog} \]

1) write action with sequence number \( s \) on page \( p \in \text{StableDatabase} \)
\[ \iff \text{StableDatabase}[p].\text{PageSeqNo} \geq s \]

2) write action with sequence number \( s \) on page \( p \in \text{CachedDatabase} \)
\[ \iff \text{DatabaseCache}[p].\text{PageSeqNo} \geq s \lor \text{StableDatabase}[p].\text{PageSeqNo} \geq s \]

Typical implementation for 1) and 2):
\[ \text{DatabaseCache}[p].\text{PageSeqNo} := \max \{ s \mid \text{there is a write action on } p \text{ with sequence number } s \} \]
Chapter 13: Page-Model
Crash Recovery Algorithms

• 13.2 Basic Data Structures
• 13.3 Redo-Winners Paradigm

  • 13.3.1 Actions During Normal Operation
    • 13.3.2 Simple Three-Pass Algorithm
    • 13.3.3 Enhanced Algorithm:
      Log Truncation, Checkpoints, Redo Optimization
    • 13.3.4 Complete Algorithm:
      Handling Transaction Aborts and Undo Completion

• 13.4 Redo-History Paradigm
• 13.5 Lessons Learned
write or full-write (pageno, transid, s):
  DatabaseCache[pageno].Contents := modified contents;
  DatabaseCache[pageno].PageSeqNo := s;
  DatabaseCache[pageno].Status := dirty;
  newlogentry.LogSeqNo := s;
  newlogentry.ActionType := write or full-write;
  newlogentry.TransId := transid;
  newlogentry.PageNo := pageno;
  newlogentry.UndoInfo := information to undo update
    (before-image for full-write);
  newlogentry.RedoInfo := information to redo update
    (after-image for full-write);
  newlogentry.PreviousSeqNo :=
    ActiveTrans[transid].LastSeqNo;
  ActiveTrans[transid].LastSeqNo := s;
  LogBuffer += newlogentry;
Actions During Normal Operation (2)

**fetch** (pageno):

```
DatabaseCache += pageno;
DatabaseCache[pageno].Contents :=
    StableDatabase[pageno].Contents;
DatabaseCache[pageno].PageSeqNo :=
    StableDatabase[pageno].PageSeqNo;
DatabaseCache[pageno].Status := clean;
```

**flush** (pageno):

```
if there is logentry in LogBuffer
    with logentry.PageNo = pageno
then force ( ); end /*if*/;
StableDatabase[pageno].Contents :=
    DatabaseCache[pageno].Contents;
StableDatabase[pageno].PageSeqNo :=
    DatabaseCache[pageno].PageSeqNo;
DatabaseCache[pageno].Status := clean;
```

**force** ( ):

```
StableLog += LogBuffer;
LogBuffer := empty;
```
**Actions During Normal Operation (3)**

begin (transid, s):
   ActiveTrans += transid;
   ActiveTrans[transid].LastSeqNo := s;
   newlogentry.LogSeqNo := s;
   newlogentry.ActionType := begin;
   newlogentry.TransId := transid;
   newlogentry.PreviousSeqNo := nil;
   LogBuffer += newlogentry;

commit (transid, s):
   newlogentry.LogSeqNo := s;
   newlogentry.ActionType := commit;
   newlogentry.TransId := transid;
   newlogentry.PreviousSeqNo :=
       ActiveTrans[transid].LastSeqNo;
   LogBuffer += newlogentry;
   ActiveTrans -= transid;
   force ( ); //potential bottleneck
Correctness and Efficiency Considerations for Actions During Normal Operation

Theorem 13.1:
During normal operation, the redo logging rule, the undo logging rule, and the garbage collection rule are satisfied.

Forced log I/O is potential bottleneck during normal operation
→ group commit for log I/O batching
Chapter 13: Page-Model
Crash Recovery Algorithms

- 13.2 Basic Data Structures
- 13.3 Redo-Winners Paradigm
  - 13.3.1 Actions During Normal Operation
  - 13.3.2 Simple Three-Pass Algorithm
    - 13.3.3 Enhanced Algorithm:
      Log Truncation, Checkpoints, Redo Optimization
    - 13.3.4 Complete Algorithm:
      Handling Transaction Aborts and Undo Completion
- 13.4 Redo-History Paradigm
- 13.5 Lessons Learned
Overview of Simple Three-Pass Algorithm

- **Analysis pass:**
  determine start of stable log from master record
  perform forward scan
  to determine winner and loser transactions

- **Redo pass:**
  perform forward scan
  to redo all winner actions in chronological (LSN) order
  (until end of log is reached)

- **Undo pass:**
  perform backward scan
  to traverse all loser log entries in reverse chronological order
  and undo the corresponding actions
Assumptions for the Simple Three-Pass Algorithm

• Only full writes
• No aborted transactions
• PRED $\Rightarrow$ LRC and CSR: in particular, LRC, upon wi-wj conflict:
  1. ai < wj[x]
  2. Otherwise:
     1. tj commits: ci < cj
     2. tj aborts: aj < ai if ti aborts
Simple Three-Pass Algorithm (1)

restart ():
    analysis pass () returns losers;
    redo pass ();
    undo pass ();

analysis pass () returns losers:
var losers: set of record

    TransId: identifier;
    LastSeqNo: identifier;
    end indexed by TransId;

losers := empty;
min := LogSeqNo of oldest log entry in StableLog;
max := LogSeqNo of most recent log entry in StableLog;
for i := min to max do
    case StableLog[i].ActionType:
        begin: losers += StableLog[i].TransId;
                losers[StableLog[i].TransId].LastSeqNo := nil;
        commit: losers -= StableLog[i].TransId;
        full-write: losers[StableLog[i].TransId].LastSeqNo := i;
    end /*case*/;
end /*for*/;
**Simple Three-Pass Algorithm (2)**

**redo pass ( )**:  
min := LogSeqNo of oldest log entry in StableLog;  
max := LogSeqNo of most recent log entry in StableLog;  
for i := min to max  
do  
    if StableLog[i].ActionType = full-write and  
        StableLog[i].TransId not in losers  
    then  
        pageno = StableLog[i].PageNo;  
        fetch (pageno);  
        full-write (pageno)  
            with contents from StableLog[i].RedoInfo;  
    end /*if*/;  
end /*for*/;
Simple Three-Pass Algorithm (3)

undo pass ( ):
    while there exists t in losers
        such that losers[t].LastSeqNo <> nil
do

    nexttrans = TransNo in losers
        such that losers[nexttrans].LastSeqNo =
            max {losers[x].LastSeqNo | x in losers};
    nextentry = losers[nexttrans].LastSeqNo;
    if StableLog[nextentry].ActionType = full-write
        then
            pageno = StableLog[nextentry].PageNo;
            fetch (pageno);
            full-write (pageno)
                with contents from StableLog[nextentry].UndoInfo;
            losers[nexttrans].LastSeqNo :=
                StableLog[nextentry].PreviousSeqNo;
        end /*if*/;
end /*while*/;
Theorem 13.2:
When restricted to full-writes as data actions, the simple three-pass recovery algorithm performs correct recovery.

Proof sketch:
1) all winners must have a commit log entry on stable log
   losers without any stable log entries are irrelevant
2) redo restores last committed write for each page
   (which absorbs all earlier winner writes)
3) LRC implies that losers follow winners for each page
   ⇒ undo restores page state as of the time
     before the first loser write and after the last winner write
   ⇒ resulting cached database contains exactly the last
     committed write of the original history
Example Scenario: up to Crash

- $t_1$: $w(a)$, $w(d)$
- $t_2$: $w(c)$, $w(e)$
- $t_3$: $w(b)$, $w(d)$
- $t_4$: $w(d)$
- $t_5$: $w(a)$, $w(b)$, $w(f)$

Flushes:
- $d$: $flush(d)$, $flush(d)$
- $b$: $flush(b)$

Crash
Example Scenario: from Crash on

1st crash

2nd crash

1st restart (incomplete)

2nd restart (complete)

1st restart

2nd restart

resume normal operation

analysis pass

redo pass

undo pass

analysis pass

redo pass

undo pass

w(a)
w(b)
w(c)
w(d)
w(e)
w(f)
t1 w(a) w(d)
t2 w(c) w(e)
t3 w(b) w(d)
t4 w(d)
t5 w(a) w(b) w(f)
flush(d) flush(d) flush(b)
1st crash 2nd crash

6/27/2016 Transactional Information Systems
### Example under Simple Three-Pass Algorithm

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t₁)</td>
<td></td>
<td></td>
<td>1: begin (t₁)</td>
<td></td>
</tr>
<tr>
<td>2: begin (t₂)</td>
<td></td>
<td></td>
<td>2: begin (t₂)</td>
<td></td>
</tr>
<tr>
<td>3: write (a, t₁)</td>
<td>a: 3</td>
<td></td>
<td>3: write (a, t₁)</td>
<td></td>
</tr>
<tr>
<td>4: begin (t₃)</td>
<td></td>
<td></td>
<td>4: begin (t₃)</td>
<td></td>
</tr>
<tr>
<td>5: begin (t₄)</td>
<td></td>
<td></td>
<td>5: begin (t₄)</td>
<td></td>
</tr>
<tr>
<td>6: write (b, t₃)</td>
<td>b: 6</td>
<td></td>
<td>6: write (b, t₃)</td>
<td></td>
</tr>
<tr>
<td>7: write (c, t₂)</td>
<td>c: 7</td>
<td></td>
<td>7: write (c, t₂)</td>
<td></td>
</tr>
<tr>
<td>8: write (d, t₁)</td>
<td>d: 8</td>
<td></td>
<td>8: write (d, t₁)</td>
<td></td>
</tr>
<tr>
<td>9: commit (t₁)</td>
<td></td>
<td></td>
<td>9: commit (t₁)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>10: flush (d)</td>
<td></td>
<td></td>
<td>10: flush (d)</td>
<td></td>
</tr>
<tr>
<td>11: write (d, t₃)</td>
<td>d: 11</td>
<td></td>
<td>11: write (d, t₃)</td>
<td></td>
</tr>
<tr>
<td>12: begin (t₅)</td>
<td></td>
<td></td>
<td>12: begin (t₅)</td>
<td></td>
</tr>
<tr>
<td>13: write (a, t₅)</td>
<td>a: 13</td>
<td></td>
<td>13: write (a, t₅)</td>
<td></td>
</tr>
<tr>
<td>14: commit (t₃)</td>
<td></td>
<td></td>
<td>14: commit (t₃)</td>
<td>11, 12, 13, 14</td>
</tr>
<tr>
<td>15: flush (d)</td>
<td></td>
<td></td>
<td>15: flush (d)</td>
<td></td>
</tr>
<tr>
<td>16: write (d, t₄)</td>
<td>d: 16</td>
<td></td>
<td>16: write (d, t₄)</td>
<td></td>
</tr>
<tr>
<td>17: write (e, t₂)</td>
<td>e: 17</td>
<td></td>
<td>17: write (e, t₂)</td>
<td></td>
</tr>
<tr>
<td>18: write (b, t₅)</td>
<td>b: 18</td>
<td></td>
<td>18: write (b, t₅)</td>
<td></td>
</tr>
<tr>
<td>19: flush (b)</td>
<td></td>
<td>b: 18</td>
<td>19: flush (b)</td>
<td>16, 17, 18</td>
</tr>
<tr>
<td>20: commit (t₄)</td>
<td></td>
<td></td>
<td>20: commit (t₄)</td>
<td></td>
</tr>
<tr>
<td>21: write (f, t₅)</td>
<td>f: 21</td>
<td></td>
<td>21: write (f, t₅)</td>
<td></td>
</tr>
</tbody>
</table>

⚠️SYSTEM CRASH ⚠️
analysis pass: losers = \{t_2, t_5\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>redo (3)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (6)</td>
<td>b: 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (a)</td>
<td></td>
<td>a: 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (8)</td>
<td>d: 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (d)</td>
<td></td>
<td>d: 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (11)</td>
<td>d:11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

‡SECOND SYSTEM CRASH‡
SECOND RESTART

analysis pass: losers = \{t_2, t_5\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>redo(3)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo(6)</td>
<td>b: 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo(8)</td>
<td>d: 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo(11)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo(16)</td>
<td>d: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(18)</td>
<td>b: 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(17)</td>
<td>e: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(13)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(7)</td>
<td>c: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SECOND RESTART COMPLETE: RESUME NORMAL OPERATION
Incorporating General Writes
As Physiological Log Entries

**Principle:**

- state testing during the redo pass:
  
  for log entry for page p with log sequence number i,
  redo write only if \( i > p.\text{PageSeqNo} \)
  and subsequently set \( p.\text{PageSeqNo} := i \)

- state testing during the undo pass:
  
  for log entry for page p with log sequence number i,
  undo write only if \( i \leq p.\text{PageSeqNo} \)
  and subsequently set \( p.\text{PageSeqNo} := i-1 \)

If \( i \) is bigger than the effect of \( i \)'s action is not there!
Simple Three-Pass Algorithm with General Writes

redo pass ():

... fetch (pageno);
if DatabaseCache[pageno].PageSeqNo < i then
    read and write (pageno)
        according to StableLog[i].RedoInfo;
    DatabaseCache[pageno].PageSeqNo := i;
end /*if*/;

undo pass ():

... fetch (pageno);
if DatabaseCache[pageno].PageSeqNo >= nextentry.LogSeqNo then
    read and write (pageno)
        according to StableLog[nextentry].UndoInfo;
    DatabaseCache[pageno].PageSeqNo :=
        nextentry.LogSeqNo - 1;
end /*if*/;
...
Correctness of Simple Three-Pass Algorithm for General Writes

Theorem 13.3:
The simple three-pass recovery algorithm with sequence number testing performs correct recovery for general writes.
### Example under Simple Three-Pass Algorithm with General Writes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t₁)</td>
<td></td>
<td></td>
<td>1: begin(t₁)</td>
<td></td>
</tr>
<tr>
<td>2: begin (t₂)</td>
<td></td>
<td></td>
<td>2: begin (t₂)</td>
<td></td>
</tr>
<tr>
<td>3: write (a, t₁)</td>
<td>a: 3</td>
<td></td>
<td>3: write (a, t₁)</td>
<td></td>
</tr>
<tr>
<td>4: begin (t₃)</td>
<td></td>
<td></td>
<td>4: begin (t₃)</td>
<td></td>
</tr>
<tr>
<td>5: begin (t₄)</td>
<td></td>
<td></td>
<td>5: begin (t₄)</td>
<td></td>
</tr>
<tr>
<td>6: write (b, t₃)</td>
<td>b: 6</td>
<td></td>
<td>6: write (b, t₃)</td>
<td></td>
</tr>
<tr>
<td>7: write (c, t₂)</td>
<td>c: 7</td>
<td></td>
<td>7: write (c, t₂)</td>
<td></td>
</tr>
<tr>
<td>8: write (d, t₁)</td>
<td>d: 8</td>
<td></td>
<td>8: write (d, t₁)</td>
<td></td>
</tr>
<tr>
<td>9: commit (t₁)</td>
<td></td>
<td></td>
<td>9: commit (t₁)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>10: flush (d)</td>
<td>d: 8</td>
<td></td>
<td>10: flush (d)</td>
<td></td>
</tr>
<tr>
<td>11: write (d, t₃)</td>
<td>d: 11</td>
<td></td>
<td>11: write (d, t₃)</td>
<td></td>
</tr>
<tr>
<td>12: begin (t₅)</td>
<td></td>
<td></td>
<td>12: begin (t₅)</td>
<td></td>
</tr>
<tr>
<td>13: write (a, t₅)</td>
<td>a: 13</td>
<td></td>
<td>13: write (a, t₅)</td>
<td></td>
</tr>
<tr>
<td>14: commit (t₃)</td>
<td></td>
<td></td>
<td>14: commit (t₃)</td>
<td>11, 12, 13, 14</td>
</tr>
<tr>
<td>15: flush (d)</td>
<td>d: 11</td>
<td></td>
<td>15: flush (d)</td>
<td></td>
</tr>
<tr>
<td>16: write (d, t₄)</td>
<td>d: 16</td>
<td></td>
<td>16: write (d, t₄)</td>
<td></td>
</tr>
<tr>
<td>17: write (e, t₂)</td>
<td>e: 17</td>
<td></td>
<td>17: write (e, t₂)</td>
<td></td>
</tr>
<tr>
<td>18: write (b, t₅)</td>
<td>b: 18</td>
<td></td>
<td>18: write (b, t₅)</td>
<td></td>
</tr>
<tr>
<td>19: flush (b)</td>
<td>b: 18</td>
<td></td>
<td>19: flush (b)</td>
<td>16, 17, 18</td>
</tr>
<tr>
<td>20: commit (t₄)</td>
<td></td>
<td></td>
<td>20: commit (t₄)</td>
<td>20</td>
</tr>
<tr>
<td>21: write (f, t₅)</td>
<td>f: 21</td>
<td></td>
<td>21: write (f, t₅)</td>
<td></td>
</tr>
</tbody>
</table>

**SYSTEM CRASH**
analysis pass: losers = \{t_2, t_5\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>redo (3)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (6)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (a)</td>
<td></td>
<td>a: 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (8)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (11)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Redo steps on d with LSN <= 11 are suppressed

☞SECOND SYSTEM CRASH☜
SECOND RESTART

analysis pass: losers = \{t_2, t_5\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>consider-redo(3)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo(6)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo(8)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo(11)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo(16)</td>
<td>d: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(18)</td>
<td>b: 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(17)</td>
<td>e: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(13)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(7)</td>
<td>c: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SECOND RESTART COMPLETE: RESUME NORMAL OPERATION
Chapter 13: Page-Model
Crash Recovery Algorithms

- 13.2 Basic Data Structures

- 13.3 Redo-Winners Paradigm
  - 13.3.1 Actions During Normal Operation
  - 13.3.2 Simple Three-Pass Algorithm
  - 13.3.3 Enhanced Algorithm:
    Log Truncation, Checkpoints, Redo Optimization
  - 13.3.4 Complete Algorithm:
    Handling Transaction Aborts and Undo Completion

- 13.4 Redo-History Paradigm

- 13.5 Lessons Learned
Need and Opportunity for Log Truncation

Major cost factors and potential availability bottlenecks:
1) analysis pass and redo pass scan entire log
2) redo pass performs many random I/Os on stable database

Improvement:
continuously advance the log start pointer (garbage collection)
• for redo, can drop all log entries for page p that
  precede the last flush action for p =: RedoLSN (p);
  min\{RedoLSN (p) | dirty page p\} =: SystemRedoLSN
• for undo, can drop all log entries that
  precede the oldest log entry of a potential loser =: OldestUndoLSN

Remarks:
for full-writes, all but the most recent after-image can be dropped
log truncation after complete undo pass requires global flush
Log Truncation

log truncation ( ):  
OldestUndoLSN :=  
    min {i | StableLog[i].TransId is in ActiveTrans};  
SystemRedoLSN := min {DatabaseCache[p].RedoLSN};  
OldestRedoPage := page p such that  
    DatabaseCache[p].RedoLSN = SystemRedoLSN;  
NewStartPointer := min{OldestUndoLSN, SystemRedoLSN};  
OldStartPointer := MasterRecord.StartPointer;  
while OldStartPointer - NewStartPointer  
    is not sufficiently large  
and SystemRedoLSN < OldestUndoLSN  
do  
    flush (OldestRedoPage);  
    SystemRedoLSN := min{DatabaseCache[p].RedoLSN};  
    OldestRedoPage := page p such that  
        DatabaseCache[p].RedoLSN = SystemRedoLSN;  
    NewStartPointer := min{OldestUndoLSN, SystemRedoLSN};  
end /*while*/;  
MasterRecord.StartPointer := NewStartPointer;
Heavy-Weight Checkpoints

```
begin (t_i)
begin (t_k)
write (... , t_i)
write (... , t_k)
write (... , t_i)
ActiveTrans: \{t_i, t_k\}
```

**Checkpoint Log Analysis:***

- **Stable Log:** Records transactions in the order they occur.
- **Master Record:** Contains metadata about the checkpoint.
- **LastCP Pointer:** Tracks the last checkpointed transaction.
- **LastSeqNo's:** Sequence numbers for log entries.
- **Analysis Pass:** Reviews log entries for consistency.
- **Redo Pass:** Reapplies committed transactions.
- **Undo Pass:** Reverses transactions that were not committed.
Recovery with Heavy-Weight Checkpoints (1)

```plaintext
checkpoint ():
  for each p in DatabaseCache do
    if DatabaseCache[p].Status = dirty
      then flush (p);
  end /*if*/;
end /*for*/;
logentry.ActionType := checkpoint;
logentry.ActiveTrans :=
  ActiveTrans (as maintained in memory);
logentry.LogSeqNo := new sequence number;
LogBuffer += logentry;
force ();
MasterRecord.LastCP := logentry.LogSeqNo;
```
Recovery with Heavy-Weight Checkpoints (2)

**analysis pass** ( ) returns losers:

cp := MasterRecord.LastCP;
losers := StableLog[cp].ActiveTrans;
max := LogSeqNo of most recent log entry in StableLog;
for i := cp to max do
    case StableLog[i].ActionType:
        ... maintenance of losers
        as in the algorithm without checkpoints
        ...
    end /*case*/;
end /*for*/;

**redo pass** ( ):

cp := MasterRecord.LastCP;
max := LogSeqNo of most recent log entry in StableLog;
for i := cp to max do
    ... page-state-testing and redo steps
    as in the algorithm without checkpoints
    ...
end /*for*/;
## Example with Heavy-Weight Checkpoints

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t₁)</td>
<td></td>
<td></td>
<td>1: begin (t₁)</td>
<td></td>
</tr>
<tr>
<td>2: begin (t₂)</td>
<td></td>
<td></td>
<td>2: begin (t₂)</td>
<td></td>
</tr>
<tr>
<td>3: write (a, t₁)</td>
<td>a: 3</td>
<td></td>
<td>3: write (a, t₁)</td>
<td></td>
</tr>
<tr>
<td>4: begin (t₃)</td>
<td></td>
<td></td>
<td>4: begin (t₃)</td>
<td></td>
</tr>
<tr>
<td>5: begin (t₄)</td>
<td></td>
<td></td>
<td>5: begin (t₄)</td>
<td></td>
</tr>
<tr>
<td>6: write (b, t₃)</td>
<td>b: 6</td>
<td></td>
<td>6: write (b, t₃)</td>
<td></td>
</tr>
<tr>
<td>7: write (c, t₂)</td>
<td>c: 7</td>
<td></td>
<td>7: write (c, t₂)</td>
<td></td>
</tr>
<tr>
<td>8: write (d, t₁)</td>
<td>d: 8</td>
<td></td>
<td>8: write (d, t₁)</td>
<td></td>
</tr>
<tr>
<td>9: commit (t₁)</td>
<td></td>
<td></td>
<td>9: commit (t₁)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>10: flush (d)</td>
<td></td>
<td>d: 8</td>
<td>10: flush (d)</td>
<td>d: 8</td>
</tr>
<tr>
<td>11: write (d, t₃)</td>
<td>d: 11</td>
<td></td>
<td>11: write (d, t₃)</td>
<td></td>
</tr>
<tr>
<td>12: begin (t₅)</td>
<td></td>
<td></td>
<td>12: begin (t₅)</td>
<td></td>
</tr>
<tr>
<td>13: write (a, t₅)</td>
<td>a: 13</td>
<td></td>
<td>13: write (a, t₅)</td>
<td></td>
</tr>
<tr>
<td><strong>14: checkpoint</strong></td>
<td></td>
<td>a: 13, b: 6, c: 7, d: 11</td>
<td><strong>14: CP</strong></td>
<td>11, 12, 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ActiveTrans: {t2, t3, t4, t5}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>
### Example with Heavy-Weight Checkpoints

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14: Checkpoint</td>
<td></td>
<td>a: 13, b: 6, c: 7, d: 11</td>
<td>14: CP</td>
<td>11, 12, 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ActiveTrans: {t2, t3, t4, t5}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15: commit (t₃)</td>
<td></td>
<td></td>
<td>15: commit (t₃)</td>
<td>15</td>
</tr>
<tr>
<td>[16: flush (d)]</td>
<td></td>
<td>d: 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17: write (d, t₄)</td>
<td></td>
<td>d: 17</td>
<td>17: write (d, t₄)</td>
<td></td>
</tr>
<tr>
<td>18: write (e, t₂)</td>
<td></td>
<td>e: 18</td>
<td>18: write (e, t₂)</td>
<td></td>
</tr>
<tr>
<td>19: write (b, t₅)</td>
<td></td>
<td>b: 19</td>
<td>19: write (b, t₅)</td>
<td></td>
</tr>
<tr>
<td>20: flush (b)</td>
<td></td>
<td>b: 19</td>
<td></td>
<td>17, 18, 19</td>
</tr>
<tr>
<td>21: commit (t₄)</td>
<td></td>
<td></td>
<td>21: commit (t₄)</td>
<td>21</td>
</tr>
<tr>
<td>22: write (f, t₅)</td>
<td></td>
<td>f: 22</td>
<td>22: write (f, t₅)</td>
<td></td>
</tr>
</tbody>
</table>

**瘁SYSTEM CRASH瘁**
RESTART

analysis pass: losers = \{t_2, t_5\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>redo(17)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d: 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>e: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a: 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c: 6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESTART COMPLETE: RESUME NORMAL OPERATION
Dirty Page List for Redo Optimization

Keep track of
• the set of dirty cached pages
• for each such page the sequence number of
  the oldest write action that followed the most recent flush action
  (redo sequence numbers)
Avoid very old RedoSeqNo‘s by write-behind demon

```pascal
type DirtyPageListEntry: record of
  PageNo: identifier;
  RedoSeqNo: identifier;
end;
var DirtyPages: set of DirtyPageListEntry indexed by PageNo;
```

Record dirty page list in checkpoint log entry and reconstruct (conservative approximation of) dirty page list during analysis pass
→ exploit knowledge of dirty page list and redo sequence numbers for I/O optimizations during redo
Light-Weight Checkpoints

- **begin (ti)**
- **begin (tk)**
- **write (...,ti)**
- **write (...,tk)**
- **write (...,ti)**

Active Trans: \{ti, tk\}

- **checkpoint**
  - **stable log**
- **LastCP**
- **Start Pointer**
- **master record**
- **Dirty Pages:** \{p, q, x\}
- **RedoSeqNo’s**
- **LastSeqNo’s**

Analysis pass
Redo pass
Undo pass
Recovery with Light-Weight Checkpoints (1)

```plaintext
checkpoint ( ):
    DirtyPages := empty;
    for each p in DatabaseCache do
        if DatabaseCache[p].Status = dirty then
            DirtyPages += p;
            DirtyPages[p].RedoSeqNo :=
                DatabaseCache[p].RedoLSN;
        end /*if*/;
    end /*for*/;
    logentry.ActionType := checkpoint;
    logentry.ActiveTrans :=
        ActiveTrans (as maintained in memory);
    logentry.DirtyPages := DirtyPages;
    logentry.LogSeqNo := new sequence number;
    LogBuffer += logentry;
    force ( );
    MasterRecord.LastCP := logentry.LogSeqNo;
```
Recovery with Light-Weight Checkpoints (2)

**analysis pass** ( ) returns losers, DirtyPages:

```plaintext
cp := MasterRecord.LastCP;
losers := StableLog[cp].ActiveTrans;
DirtyPages := StableLog[cp].DirtyPages;
max := LogSeqNo of most recent log entry in StableLog;
for i := cp to max do
  case StableLog[i].ActionType:
    ...
    maintenance of losers as in the algorithm without checkpoints
    ...
  end /*case*/;
  if StableLog[i].ActionType = write or full-write
    and StableLog[i].PageNo not in DirtyPages
  then
    DirtyPages += StableLog[i].PageNo;
    DirtyPages[StableLog[i].PageNo].RedoSeqNo := i;
  end /*if*/;
end /*for*/;
```
redo pass ():

cp := MasterRecord.LastCP;
SystemRedoLSN := min{cp.DirtyPages[p].RedoSeqNo};
max := LogSeqNo of most recent log entry in StableLog;
for i := SystemRedoLSN to max do
  if StableLog[i].ActionType = write or full-write
      and StableLog[i].TransId not in losers
  then
    pageno := StableLog[i].PageNo;
    if pageno in DirtyPages
      and i >= DirtyPages[pageno].RedoSeqNo
    then
      fetch (pageno);
      if DatabaseCache[pageno].PageSeqNo < i
      then
        read and write (pageno)
        according to StableLog[i].RedoInfo;
        DatabaseCache[pageno].PageSeqNo := i;
      else
        DirtyPages[pageno].RedoSeqNo :=
        DatabaseCache[pageno].PageSeqNo + 1;
      end/*if*/; end/*if*/; end/*if*/; end/*for*/;
### Example with Light-Weight Checkpoints

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t_1)</td>
<td></td>
<td></td>
<td>1: begin (t_1)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>2: begin (t_2)</td>
<td></td>
<td></td>
<td>2: begin (t_2)</td>
<td></td>
</tr>
<tr>
<td>3: write ((a, t_1))</td>
<td>a: 3</td>
<td></td>
<td>3: write ((a, t_1))</td>
<td></td>
</tr>
<tr>
<td>4: begin (t_3)</td>
<td></td>
<td></td>
<td>4: begin (t_3)</td>
<td></td>
</tr>
<tr>
<td>5: begin (t_4)</td>
<td></td>
<td></td>
<td>5: begin (t_4)</td>
<td></td>
</tr>
<tr>
<td>6: write ((b, t_3))</td>
<td>b: 6</td>
<td></td>
<td>6: write ((b, t_3))</td>
<td></td>
</tr>
<tr>
<td>7: write ((c, t_2))</td>
<td>c: 7</td>
<td></td>
<td>7: write ((c, t_2))</td>
<td></td>
</tr>
<tr>
<td>8: write ((d, t_1))</td>
<td>d: 8</td>
<td></td>
<td>8: write ((d, t_1))</td>
<td></td>
</tr>
<tr>
<td>9: commit (t_1)</td>
<td></td>
<td></td>
<td>9: commit (t_1)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>10: flush (d)</td>
<td></td>
<td>d: 8</td>
<td>10: flush (d)</td>
<td>11, 12, 13, 14</td>
</tr>
<tr>
<td>11: write ((d, t_3))</td>
<td>d: 11</td>
<td></td>
<td>11: write ((d, t_3))</td>
<td></td>
</tr>
<tr>
<td>12: begin (t_5)</td>
<td></td>
<td></td>
<td>12: begin (t_5)</td>
<td></td>
</tr>
<tr>
<td>13: write ((a, t_5))</td>
<td>a: 13</td>
<td></td>
<td>13: write ((a, t_5))</td>
<td></td>
</tr>
<tr>
<td>14: checkpoint</td>
<td></td>
<td>DirtyPages: {a, b, c, d} RedoLSNs: {a: 3, b: 6, c: 7, d: 11} ActiveTrans: {t_2, t_3, t_4, t_5}</td>
<td>14: CP</td>
<td>11, 12, 13, 14</td>
</tr>
</tbody>
</table>
### Example with Light-Weight Checkpoints

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14: Checkpoint</td>
<td></td>
<td></td>
<td>14: CP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DirtyPages: {a, b, c, d}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RedoLSNs: {a: 3, b: 6, c: 7, d: 11}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ActiveTrans: {t₂, t₃, t₄, t₅}</td>
<td>11, 12, 13, 14</td>
</tr>
<tr>
<td>15: commit (t₃)</td>
<td></td>
<td></td>
<td>15: commit (t₃)</td>
<td></td>
</tr>
<tr>
<td>16: flush (d)</td>
<td></td>
<td></td>
<td>15: commit (t₃)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>d: 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17: write (d, t₄)</td>
<td></td>
<td>d: 17</td>
<td>17: write (d, t₄)</td>
<td></td>
</tr>
<tr>
<td>18: write (e, t₂)</td>
<td></td>
<td>e: 18</td>
<td>18: write (e, t₂)</td>
<td></td>
</tr>
<tr>
<td>19: write (b, t₅)</td>
<td></td>
<td>b: 19</td>
<td>19: write (b, t₅)</td>
<td></td>
</tr>
<tr>
<td>20: flush (b)</td>
<td></td>
<td>b: 19</td>
<td>19: write (b, t₅)</td>
<td>17, 18, 19</td>
</tr>
<tr>
<td>21: commit (t₄)</td>
<td></td>
<td></td>
<td>21: commit (t₄)</td>
<td>21</td>
</tr>
<tr>
<td>22: write (f, t₅)</td>
<td></td>
<td>f: 22</td>
<td>22: write (f, t₅)</td>
<td></td>
</tr>
</tbody>
</table>

⇒ SYSTEM CRASH ⇒
RESTART

analysis pass: losers = \{t_2, t_5\}
DirtyPages = \{a, b, c, d, e\}
RedoLSNs: a: 3, b: 6, c: 7, d: 11, e: 18

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>consider-redo(3)</td>
<td>a: 3</td>
<td>LOOKS LIKE A REAL REDO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b: 19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>skip-redo(8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo(11)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo(17)</td>
<td>d: 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(19)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(18)</td>
<td>e: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(13)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(7)</td>
<td>c: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESTART COMPLETE: RESUME NORMAL OPERATION
Recovery with Flush Log Entries

**analysis pass** ( ) returns losers, DirtyPages:

```
cp := MasterRecord.LastCP;
losers := StableLog[cp].ActiveTrans;
DirtyPages := StableLog[cp].DirtyPages;
max := LogSeqNo of most recent log entry in StableLog;
for i := cp to max do
    case StableLog[i].ActionType:
        ...
        maintenance of losers
        as in the algorithm without checkpoints
        ...
    end /*case*/;
    if StableLog[i].ActionType = write or full-write
        and StableLog[i].PageNo not in DirtyPages
    then
        DirtyPages += StableLog[i].PageNo;
        DirtyPages[StableLog[i].PageNo].RedoSeqNo := i;
    end /*if*/;
    if StableLog[i].ActionType = flush then
        DirtyPages -= StableLog[i].PageNo;
    end /*if*/;
end /*for*/;
```
## Example with Light-Weight Checkpoints

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t_1)</td>
<td></td>
<td></td>
<td>1: begin (t_1)</td>
<td></td>
</tr>
<tr>
<td>2: begin (t_2)</td>
<td></td>
<td></td>
<td>2: begin (t_2)</td>
<td></td>
</tr>
<tr>
<td>3: write (a, t_1)</td>
<td>a: 3</td>
<td></td>
<td>3: write (a, t_1)</td>
<td></td>
</tr>
<tr>
<td>4: begin (t_3)</td>
<td></td>
<td></td>
<td>4: begin (t_3)</td>
<td></td>
</tr>
<tr>
<td>5: begin (t_4)</td>
<td></td>
<td></td>
<td>5: begin (t_4)</td>
<td></td>
</tr>
<tr>
<td>6: write (b, t_3)</td>
<td>b: 6</td>
<td></td>
<td>6: write (b, t_3)</td>
<td></td>
</tr>
<tr>
<td>7: write (c, t_2)</td>
<td>c: 7</td>
<td></td>
<td>7: write (c, t_2)</td>
<td></td>
</tr>
<tr>
<td>8: write (d, t_1)</td>
<td>d: 8</td>
<td></td>
<td>8: write (d, t_1)</td>
<td></td>
</tr>
<tr>
<td>9: commit (t_1)</td>
<td></td>
<td></td>
<td>9: commit (t_1)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>10: flush (d)</td>
<td>d: 8</td>
<td></td>
<td>10: flush (d)</td>
<td>10, 11, 12, 13, 14</td>
</tr>
<tr>
<td>11: write (d, t_3)</td>
<td>d: 11</td>
<td></td>
<td>11: write (d, t_3)</td>
<td></td>
</tr>
<tr>
<td>12: begin (t_5)</td>
<td></td>
<td></td>
<td>12: begin (t_5)</td>
<td></td>
</tr>
<tr>
<td>13: write (a, t_5)</td>
<td>a: 13</td>
<td></td>
<td>13: write (a, t_5)</td>
<td></td>
</tr>
<tr>
<td>14: checkpoint</td>
<td></td>
<td></td>
<td>14: CP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DirtyPages: {a, b, c, d}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RedoLSNs: {a: 3, b: 6, c: 7, d: 11}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ActiveTrans: {t_2, t_3, t_4, t_5}</td>
<td></td>
</tr>
</tbody>
</table>

0/27/2016 Transactional Information Systems 13-49
### Example with Light-Weight Checkpoints

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14: Checkpoint</td>
<td></td>
<td></td>
<td>14: CP</td>
<td>10, 11, 12, 13, 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DirtyPages:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{a, b, c, d}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RedoLSNs:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{a: 3, b: 6, c: 7, d: 11}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ActiveTrans:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{t₂, t₃, t₄, t₅}</td>
<td></td>
</tr>
<tr>
<td>15: commit (t₃)</td>
<td></td>
<td></td>
<td>15: commit (t₃)</td>
<td>15</td>
</tr>
<tr>
<td>16: flush (d)</td>
<td></td>
<td>d: 11</td>
<td>16: flush (d)</td>
<td></td>
</tr>
<tr>
<td>17: write (d, t₄)</td>
<td></td>
<td>d: 17</td>
<td>17: write (d, t₄)</td>
<td></td>
</tr>
<tr>
<td>18: write (e, t₂)</td>
<td></td>
<td>e: 18</td>
<td>18: write (e, t₂)</td>
<td></td>
</tr>
<tr>
<td>19: write (b, t₅)</td>
<td></td>
<td>b: 19</td>
<td>19: write (b, t₅)</td>
<td></td>
</tr>
<tr>
<td>20: flush (b)</td>
<td></td>
<td>b: 19</td>
<td>20: flush (b)</td>
<td>16, 17, 18, 19</td>
</tr>
<tr>
<td>21: commit (t₄)</td>
<td></td>
<td></td>
<td>21: commit (t₄)</td>
<td>20, 21</td>
</tr>
<tr>
<td>22: write (f, t₅)</td>
<td></td>
<td>f: 22</td>
<td>22: write (f, t₅)</td>
<td></td>
</tr>
</tbody>
</table>

**SYNC SYSTEM CRASH**
RESTART

- Losers: \{t_2, t_5\}
- DirtyPages: \{a, c, d, e\}
- RedoLSNs: a: 3, c: 7, d: 17, e: 18

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>consider-redo(3)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo(6)</td>
<td>b: 19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>skip-redo(8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>skip-redo(11)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo(17)</td>
<td>d: 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo(19)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(18)</td>
<td>e: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(13)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo(7)</td>
<td>c: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESTART COMPLETE: RESUME NORMAL OPERATION
Correctness of Enhanced Three-Pass Algorithm

Theorem 13.4:
Extending the simple three-pass recovery algorithm with log truncation, heavy-weight or light-weight checkpoints, and flush action logging (or any subset of these features) preserves the correctness of crash recovery.
Chapter 13: Page-Model Crash Recovery Algorithms

• 13.2 Basic Data Structures

• 13.3 Redo-Winners Paradigm
  • 13.3.1 Actions During Normal Operation
  • 13.3.2 Simple Three-Pass Algorithm
  • 13.3.3 Enhanced Algorithm:
    Log Truncation, Checkpoints, Redo Optimization
  • 13.3.4 Complete Algorithm:
    Handling Transaction Aborts and Undo Completion

• 13.4 Redo-History Paradigm

• 13.5 Lessons Learned
Problems with Aborted Transactions as Losers

- identifying losers would require full log scan (without advantage from checkpoints)
- losers would precede winners in serialization order

Example:
### Example Scenario with Aborted Transactions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t₁)</td>
<td></td>
<td></td>
<td>1: begin (t₁)</td>
<td></td>
</tr>
<tr>
<td>2: write (a, t₁)</td>
<td>a: 2</td>
<td></td>
<td>2: write (a, t₁)</td>
<td></td>
</tr>
<tr>
<td>3: commit (t₁)</td>
<td></td>
<td></td>
<td>3: commit (t₁)</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4: begin (t₂)</td>
<td></td>
<td></td>
<td>4: begin (t₂)</td>
<td></td>
</tr>
<tr>
<td>5: write (a, t₂)</td>
<td>a: 5</td>
<td></td>
<td>5: write (a, t₂)</td>
<td></td>
</tr>
<tr>
<td>6: abort (t₂)</td>
<td></td>
<td></td>
<td>6: abort (t₂)</td>
<td>4, 5, 6</td>
</tr>
<tr>
<td>7: begin (t₃)</td>
<td></td>
<td></td>
<td>7: begin (t₃)</td>
<td></td>
</tr>
<tr>
<td>8: write (a, t₃)</td>
<td>a: 8</td>
<td></td>
<td>8: write (a, t₃)</td>
<td></td>
</tr>
<tr>
<td>9: commit (t₃)</td>
<td></td>
<td></td>
<td>9: commit (t₃)</td>
<td>7, 8, 9</td>
</tr>
<tr>
<td>10: begin (t₄)</td>
<td></td>
<td></td>
<td>10: begin (t₄)</td>
<td></td>
</tr>
<tr>
<td>11: write (b, t₄)</td>
<td>b: 11</td>
<td></td>
<td>11: write (b, t₄)</td>
<td></td>
</tr>
<tr>
<td>12: write (a, t₄)</td>
<td>a: 12</td>
<td></td>
<td>12: write (a, t₄)</td>
<td></td>
</tr>
<tr>
<td>13: flush (a)</td>
<td>a: 12</td>
<td></td>
<td>13: flush (a)</td>
<td>10, 11, 12</td>
</tr>
</tbody>
</table>

**SYSTEM CRASH**

**RESTART**

### Analysis pass: “losers” = \{t₂, t₄\}

- consider-redo (2) a: 12
- consider-redo (8) a: 12
- undo (12)
- consider-undo (11)
- undo (5)
Handling Aborted Transactions as Winners

- create compensation log entries for inverse operations of transaction rollback
- complete rollback by creating rollback log entry
- during crash recovery, aborted transactions with complete rollback are winners, incomplete aborted transactions are losers

Theorem 13.5:
The extension for handling transaction rollbacks during normal operation preserves the correctness of the three-pass algorithm.
Completion of Transaction Rollback

abort (transid):
logentry := ActiveTrans[transid].LastSeqNo;
while logentry is not nil and "if"
    logentry.ActionType = write or full-write do
newlogentry.LogSeqNo := new sequence number;
newlogentry.ActionType := compensation;
newlogentry.PreviousSeqNo := ActiveTrans[transid].LastSeqNo;
newlogentry.RedoInfo :=
    inverse action of the action in logentry;
newlogentry.UndoInfo :=
    inverse action of inverse action of action in logentry;
ActiveTrans[transid].LastSeqNo := newlogentry.LogSeqNo;
LogBuffer += newlogentry;
write (logentry.PageNo) according to logentry.UndoInfo;
logentry := logentry.PreviousSeqNo;
end /*while*/
newlogentry.LogSeqNo := new sequence number;
newlogentry.ActionType := rollback;
newlogentry.TransId := transid;
newlogentry.PreviousSeqNo := ActiveTrans[transid].LastSeqNo;
LogBuffer += newlogentry;
ActiveTrans -= transid; force ( );
### Example with Aborted Transactions as Winners

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin ($t_1$)</td>
<td></td>
<td></td>
<td>1: begin ($t_1$)</td>
<td></td>
</tr>
<tr>
<td>2: write (a, $t_1$)</td>
<td>a: 2</td>
<td></td>
<td>2: write (a, $t_1$)</td>
<td></td>
</tr>
<tr>
<td>3: commit ($t_1$)</td>
<td></td>
<td></td>
<td>3: commit ($t_1$)</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4: begin ($t_2$)</td>
<td></td>
<td></td>
<td>4: begin ($t_2$)</td>
<td></td>
</tr>
<tr>
<td>5: write (a, $t_2$)</td>
<td>a: 5</td>
<td></td>
<td>5: write (a, $t_2$)</td>
<td></td>
</tr>
<tr>
<td>6: abort ($t_2$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7: compensate (5: write (a, $t_2$))</td>
<td></td>
<td></td>
<td>7: compensate (a, $t_2$)</td>
<td></td>
</tr>
<tr>
<td>8: rollback ($t_2$)</td>
<td></td>
<td></td>
<td>8: rollback ($t_2$)</td>
<td>4, 5, 7, 8</td>
</tr>
<tr>
<td>9: begin ($t_3$)</td>
<td></td>
<td></td>
<td>9: begin ($t_3$)</td>
<td></td>
</tr>
<tr>
<td>10: write (a, $t_3$)</td>
<td>a: 10</td>
<td></td>
<td>10: write (a, $t_3$)</td>
<td></td>
</tr>
<tr>
<td>11: commit ($t_3$)</td>
<td></td>
<td></td>
<td>11: commit ($t_3$)</td>
<td>9, 10, 11</td>
</tr>
<tr>
<td>12: begin ($t_4$)</td>
<td></td>
<td></td>
<td>12: begin ($t_4$)</td>
<td></td>
</tr>
<tr>
<td>13: write (b, $t_4$)</td>
<td>b: 13</td>
<td></td>
<td>13: write (b, $t_4$)</td>
<td></td>
</tr>
<tr>
<td>14: write (a, $t_4$)</td>
<td>a: 14</td>
<td></td>
<td>14: write (a, $t_4$)</td>
<td></td>
</tr>
<tr>
<td>15: abort ($t_4$)</td>
<td></td>
<td></td>
<td>15: abort ($t_4$)</td>
<td></td>
</tr>
<tr>
<td>16: compensate (14: write (a, $t_4$))</td>
<td></td>
<td></td>
<td>16: compensate (a, $t_4$)</td>
<td></td>
</tr>
<tr>
<td>17: flush (a)</td>
<td>a: 16</td>
<td></td>
<td></td>
<td>12, 13, 14, 16</td>
</tr>
</tbody>
</table>

**SYSTEM CRASH**
analysis pass: “losers” = \{t_4\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>consider-redo (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo (16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a: 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo (14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a: 13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo (13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESTART COMPLETE: RESUME NORMAL OPERATION
הבחינה ביום ב', 30/6 בשעה 0900
تكلים בטאוב 201.
Undo Completion

• create undo-complete log entry for each loser,
• flush pages modified during undo, and
• set OldestUndoLSN to nil (to facilitate log truncation)
• do not treat as winers in subsequent crashes!!
• need a minor change in the analysis pass.
• a new classification 'undone' for which we neither redo nor undo.

See next slide

Theorem 13.6:
The method for undo completion preserves the correctness of the three-pass algorithm.
Complete Undo Algorithm (1)

undo pass ():
  FlushList := empty;
  while there exists t in losers such that losers[t].LastSeqNo <> nil do
    nexttrans := TransNo in losers such that
    losers[TransNo].LastSeqNo = max {losers[x].LastSeqNo | x in losers};
    nextentry = losers[nexttrans].LastSeqNo;
    if StableLog[nextentry].ActionType = write then
      pageno := StableLog[nextentry].PageNo; fetch (pageno);
      if DatabaseCache[pageno].PageSeqNo >= nextentry.LogSeqNo;
        read and write (StableLog[nextentry].PageNo)
        according to StableLog[nextentry].UndoInfo;
        DatabaseCache[pageno].PageSeqNo := nextentry.LogSeqNo - 1;
        FlushList += pageno;
      end /*if*/;
      losers[nexttrans].LastSeqNo := StableLog[nextentry].PreviousSeqNo;
    end /*if*/;
  end /*while*/;
for each p in FlushList do
    flush (p);
end /*for*/;
for each t in losers do
    newlogentry.LogSeqNo := new sequence number;
    newlogentry.ActionType := undo-complete;
    newlogentry.TransId := losers[t].TransId;
    LogBuffer += newlogentry;
end /*for*/;
force ( ) ;
## Example with Undo Completion

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t₁)</td>
<td></td>
<td></td>
<td>1: begin (t₁)</td>
<td></td>
</tr>
<tr>
<td>2: write (a, t₁)</td>
<td>a: 2</td>
<td></td>
<td>2: write (a, t₁)</td>
<td></td>
</tr>
<tr>
<td>3: commit (t₁)</td>
<td></td>
<td></td>
<td>3: commit (t₁)</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4: begin (t₂)</td>
<td></td>
<td></td>
<td>4: begin (t₂)</td>
<td></td>
</tr>
<tr>
<td>5: write (a, t₂)</td>
<td>a: 5</td>
<td></td>
<td>5: write (a, t₂)</td>
<td></td>
</tr>
<tr>
<td>6: abort (t₂)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7: compensate (5: write (a, t₂))</td>
<td>a: 7</td>
<td></td>
<td>7: compensate (a, t₂)</td>
<td>4, 5, 7, 8</td>
</tr>
<tr>
<td>8: rollback (t₂)</td>
<td></td>
<td></td>
<td>8: rollback (t₂)</td>
<td>4, 5, 7, 8</td>
</tr>
<tr>
<td>9: begin (t₃)</td>
<td></td>
<td></td>
<td>9: begin (t₃)</td>
<td></td>
</tr>
<tr>
<td>10: write (b, t₃)</td>
<td>b: 10</td>
<td></td>
<td>10: write (b, t₃)</td>
<td></td>
</tr>
<tr>
<td>11: commit (t₃)</td>
<td></td>
<td></td>
<td>11: commit (t₃)</td>
<td>9, 10, 11</td>
</tr>
<tr>
<td>12: begin (t₄)</td>
<td></td>
<td></td>
<td>12: begin (t₄)</td>
<td></td>
</tr>
<tr>
<td>13: write (b, t₄)</td>
<td>b: 13</td>
<td></td>
<td>13: write (b, t₄)</td>
<td></td>
</tr>
<tr>
<td>14: write (a, t₄)</td>
<td>a: 14</td>
<td></td>
<td>14: write (a, t₄)</td>
<td></td>
</tr>
<tr>
<td>15: abort (t₄)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16: compensate (14: write (a, t₄))</td>
<td>a: 16</td>
<td></td>
<td>16: compensate (a, t₄)</td>
<td>12, 13, 14, 16</td>
</tr>
<tr>
<td>17: flush (a)</td>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18: begin (t₅)</td>
<td></td>
<td></td>
<td>18: begin (t₅)</td>
<td></td>
</tr>
<tr>
<td>19: write (c, t₅)</td>
<td>c: 19</td>
<td></td>
<td>19: write (c, t₅)</td>
<td></td>
</tr>
<tr>
<td>20: begin (t₆)</td>
<td></td>
<td></td>
<td>20: begin (t₆)</td>
<td></td>
</tr>
<tr>
<td>21: write (d, t₆)</td>
<td>d: 21</td>
<td></td>
<td>21: write (d, t₆)</td>
<td></td>
</tr>
<tr>
<td>22: flush (c)</td>
<td>c: 19</td>
<td></td>
<td></td>
<td>18, 19, 20, 21</td>
</tr>
</tbody>
</table>

**SYSTEM CRASH**
RESTART

analysis pass: “losers” = \{t_4, t_5, t_6\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>consider-redo (2)</td>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (5)</td>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (7)</td>
<td>a: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (10)</td>
<td>b: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo (21)</td>
<td>d: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo (19)</td>
<td>c: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo (16)</td>
<td>a: 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undo (14)</td>
<td>a: 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-undo (13)</td>
<td>b: 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (a)</td>
<td></td>
<td>a: 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (c)</td>
<td></td>
<td>c: 18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

23: undo-complete (t_4)
24: undo-complete (t_5)
25: undo-complete (t_6)

23, 24, 25

RESTART COMPLETE: RESUME NORMAL OPERATION
Chapter 13: Page-Model
Crash Recovery Algorithms

- 13.2 Basic Data Structures
- 13.3 Redo-Winners Paradigm

- 13.4 Redo-History Paradigm
  - 13.4.1 Actions During Normal Operation
  - 13.4.2 Simple Three-Pass and Two-Pass Algorithms
  - 13.4.3 Enhanced Algorithms
  - 13.4.4 Complete Algorithms
- 13.5 Lessons Learned
Basic Idea

• In **Redo-Winners**, the redo pass considers only winners, at the expense of complicating transaction aborts and log truncation.

• In **Redo-History**, *all* actions are repeated in chronological order, i.e.,

  1. it first reconstructs the cached database,
  2. then undoes losers from there.
ARIES Family of Locking and Recovery Algorithms

This page is devoted to tracking information on the ARIES (Algorithms for Recovery and Isolation
Exploiting Semantics) family of locking, logging and recovery algorithms for persistent data management. I
have included information on the books and university courses which cover ARIES with links to course
materials, teachers and authors. The impact of ARIES on products, prototypes and researchers is also outlined.
A listing of our papers and patents on ARIES is also included.

The impact of ARIES on the research and the commercial worlds was recognized with the "10 Year Best
Impact Paper Award" at VLDB99. The birth and evolution of ARIES is described in my VLDB99 paper.
ARIES is covered in 14 books and more than 80 universities' computer science courses across the world
(Australia, Canada, Denmark, England, Finland, France, Germany, Greece, India, Iran, Israel, Italy, Korea,
New Zealand, Norway, Singapore, Spain, Sweden, Taiwan, USA). Excluding self-citations, so far, the main
ARIES paper (TODS, March 1992) has been cited more than 230 times, the ARIES/TM paper (SIGMOD92)
90 times, and the ARIES/KVL paper (VLDB90) 60 times. The referenced citation lists are much more
complete than the ones at DBLP, ACM and ResearchIndex.

I am very thankful to the professors, authors and systems builders who have made the ARIES algorithms
extremely popular via their books, courses, papers and implementations. Any comments, corrections and
additions to this page's contents would be most welcome!
Chapter 13: Page-Model
Crash Recovery Algorithms

• 13.2 Basic Data Structures
• 13.3 Redo-Winners Paradigm
• 13.4 Redo-History Paradigm
  • 13.4.1 Actions During Normal Operation
  • 13.4.2 Simple Three-Pass and Two-Pass Algorithms
    • 13.4.3 Enhanced Algorithms
    • 13.4.4 Complete Algorithms
• 13.5 Lessons Learned
Key Properties of Redo-History Algorithms

- *Optional* analysis pass
  - determines losers and
  - reconstructs DirtyPages list,
    using the analysis algorithm of the redo-winners paradigm
- Redo pass starts from SystemRedoLSN and
  - redoes *both* winner and loser updates,
    with LSN-based state testing for idempotence,
    to reconstruct the database state as of the time of the crash
- Undo pass initiates *rollback* for all loser transactions,
  *intuitively*, using the code for rollback during normal operation,
  with undo steps (without page state testing)
  - creating *compensation log entries* and
  - *advancing* page sequence numbers
**Redo Pass of Redo-History Algorithms**

**redo pass ( )**:

min := LogSeqNo of oldest log entry in StableLog;
max := LogSeqNo of most recent log entry in StableLog;
for i := min to max do
   pageno = StableLog[i].PageNo;
    fetch (pageno);
    if DatabaseCache[pageno].PageSeqNo < i then
        read and write (pageno)
            according to StableLog[i].RedoInfo;
        DatabaseCache[pageno].PageSeqNo := i;
    end /*if*/;
end /*for*/;
Undo Pass of Redo-History Algorithms (1)

undo pass ( ):
    ActiveTrans := empty;
    for each t in losers do
        ActiveTrans += t;
        ActiveTrans[t].LastSeqNo := losers[t].LastSeqNo;
    end /*for*/;
    while there exists t in losers such that losers[t].LastSeqNo <> nil do
        nexttrans := TransNo in losers such that losers[nexttrans].LastSeqNo =
        max {losers[x].LastSeqNo | x in losers};
        nextentry := losers[nexttrans].LastSeqNo;
Undo Pass of Redo-History Algorithms (2)

if StableLog[nextentry].ActionType in {write, compensation} then
   pageno := StableLog[nextentry].PageNo; fetch (pageno);
    if DatabaseCache[pageno].PageSeqNo >= nextentry.LogSeqNo then
        newlogentry.LogSeqNo := new sequence number;
        newlogentry.ActionType := compensation;
        newlogentry.PreviousSeqNo :=
            ActiveTrans[transid].LastSeqNo;
        newlogentry.RedoInfo :=
            inverse action of the action in nextentry;
        newlogentry.UndoInfo := inverse action of the
            inverse action of the inverse action of action in
            ActiveTrans[transid].LastSeqNo :=
            newlogentry.LogSeqNo;
        LogBuffer += newlogentry;
        read and write (StableLog[nextentry].PageNo)
            according to StableLog[nextentry].UndoInfo;
        DatabaseCache[pageno].PageSeqNo := newlogentry.LogSeqNo;
    end /*if*/;
    losers[nexttrans].LastSeqNo :=
        StableLog[nextentry].PreviousSeqNo;
end /*if*/;
Unto Pass of Redo-History Algorithms (3)

if StableLog[nextentry].ActionType = begin
    then
        newlogentry.LogSeqNo := new sequence number;
        newlogentry.ActionType := rollback;
        newlogentry.TransId := StableLog[nextentry].TransId;
        newlogentry.PreviousSeqNo :=
            ActiveTrans[transid].LastSeqNo;
        LogBuffer += newlogentry;
        ActiveTrans -= transid;
        losers -= transid;
    end /*if*/;
end /*while*/;
force ( );
### Simple Three-Pass Redo-History Algorithm

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t₁)</td>
<td></td>
<td>1: begin(t₁)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: begin (t₂)</td>
<td></td>
<td>2: begin (t₂)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: write (a, t₁)</td>
<td>a: 3</td>
<td>3: write (a, t₁)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: begin (t₃)</td>
<td></td>
<td>4: begin (t₃)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5: begin (t₄)</td>
<td></td>
<td>5: begin (t₄)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6: write (b, t₃)</td>
<td>b: 6</td>
<td>6: write (b, t₃)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7: write (c, t₂)</td>
<td>c: 7</td>
<td>7: write (c, t₂)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8: write (d, t₁)</td>
<td>d: 8</td>
<td>8: write (d, t₁)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9: commit (t₁)</td>
<td></td>
<td>9: commit (t₁)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
<td></td>
</tr>
<tr>
<td>10: flush (d)</td>
<td></td>
<td>10: flush (d)</td>
<td></td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>11: write (d, t₃)</td>
<td>d: 11</td>
<td>11: write (d, t₃)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12: begin (t₅)</td>
<td></td>
<td>12: begin (t₅)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13: write (a, t₅)</td>
<td>a: 13</td>
<td>13: write (a, t₅)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14: commit (t₃)</td>
<td></td>
<td>14: commit (t₃)</td>
<td></td>
<td>11, 12, 13, 14</td>
</tr>
<tr>
<td>15: flush (d)</td>
<td></td>
<td>15: flush (d)</td>
<td></td>
<td>11, 12, 13, 14</td>
</tr>
<tr>
<td>16: write (d, t₄)</td>
<td>d: 16</td>
<td>16: write (d, t₄)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17: write (e, t₂)</td>
<td>e: 17</td>
<td>17: write (e, t₂)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18: write (b, t₅)</td>
<td>b: 18</td>
<td>18: write (b, t₅)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19: flush (b)</td>
<td></td>
<td>19: flush (b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20: commit (t₄)</td>
<td></td>
<td>20: commit (t₄)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21: write (f, t₅)</td>
<td>f: 21</td>
<td>21: write (f, t₅)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SYSTEM CRASH AND RESTART**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>redo (3)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (6)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (a)</td>
<td></td>
<td>a: 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (7)</td>
<td>c: 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (8)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (11)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (13)</td>
<td>a: 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (16)</td>
<td>d: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (17)</td>
<td>e: 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (18)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (a)</td>
<td></td>
<td>a: 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22: compensate (18)</td>
<td>b: 22</td>
<td></td>
<td>22: compensate (18: b, t₅)</td>
<td></td>
</tr>
<tr>
<td>23: compensate (17)</td>
<td>e: 23</td>
<td></td>
<td>23: compensate (17: e, t₂)</td>
<td></td>
</tr>
<tr>
<td>flush (b)</td>
<td>b: 22</td>
<td></td>
<td></td>
<td>22, 23</td>
</tr>
<tr>
<td>24: compensate (13)</td>
<td>a: 24</td>
<td></td>
<td>24: compensate (13: a, t₅)</td>
<td></td>
</tr>
<tr>
<td>25: rollback (t₅)</td>
<td></td>
<td></td>
<td>25: rollback (t₅)</td>
<td></td>
</tr>
</tbody>
</table>

**SECOND SYSTEM CRASH AND SECOND RESTART**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis pass: losers = {t_2, t_5}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (17)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (22)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (23)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26: compensate (23)</td>
<td></td>
<td></td>
<td>26: compensate (23, e: t_2)</td>
<td></td>
</tr>
<tr>
<td>27: compensate (22)</td>
<td></td>
<td></td>
<td>27: compensate (22, e: t_5)</td>
<td></td>
</tr>
<tr>
<td>28: compensate (18)</td>
<td></td>
<td></td>
<td>28: compensate (18, b: t_5)</td>
<td></td>
</tr>
<tr>
<td>29: compensate (17)</td>
<td></td>
<td></td>
<td>29: compensate (17, e: t_5)</td>
<td></td>
</tr>
<tr>
<td>30: compensate (13)</td>
<td></td>
<td></td>
<td>30: compensate (13, a: t_5)</td>
<td></td>
</tr>
<tr>
<td>31: rollback (t_5)</td>
<td></td>
<td></td>
<td>31: rollback (t_5)</td>
<td></td>
</tr>
<tr>
<td>32: compensate (7)</td>
<td></td>
<td></td>
<td>32: compensate (7, c, t_2)</td>
<td></td>
</tr>
<tr>
<td>33: rollback (t_2)</td>
<td></td>
<td></td>
<td>31: rollback (t_2)</td>
<td></td>
</tr>
<tr>
<td>force</td>
<td></td>
<td></td>
<td></td>
<td>26, 27, 28, 29, 30, 31, 32, 33</td>
</tr>
</tbody>
</table>

SECOND RESTART COMPLETE: RESUME NORMAL OPERATION
Correctness of Simple Redo-History Algorithm

Theorem 13.7:
The simple three-pass redo-history recovery algorithm performs correct recovery.

Proof sketch:
• redo pass establishes the postcondition
  \( \forall p \ \forall t \ \forall o \in \text{stable log}: (o \text{ belongs to } t \text{ and refers to } p) \Rightarrow o \in \text{cached db} \)
• undo pass performs rollback like during normal operation and establishes the postcondition
  \( \forall p \ \forall t \ \forall o \in \text{stable log}: (o \text{ belongs to } t \text{ and refers to } p \text{ and } t \in \text{losers}) \Rightarrow o \notin \text{cached db} \)
• as losers follow winners in the serialization order, the final postcondition of the entire restart is
  \( \forall p \ \forall t \ \forall o \in \text{stable log}: (o \text{ belongs to } t \text{ and refers to } p \text{ and } t \in \text{winners}) \Rightarrow o \in \text{cached db} \)
• a second crash during redo does not affect the second restart
• a second crash during undo could leave losers prolonged with some (but not all) inverse actions; the second restart will treat them as if the inverse actions were forward actions, and thus is no different from the first restart
Chapter 13: Page-Model
Crash Recovery Algorithms

- 13.2 Basic Data Structures
- 13.3 Redo-Winners Paradigm
- 13.4 Redo-History Paradigm
  - 13.4.1 Actions During Normal Operation
  - 13.4.2 Simple Three-Pass and Two-Pass Algorithms
  - 13.4.3 Enhanced Algorithms
- 13.4.4 Complete Algorithms
- 13.5 Lessons Learned
Undo Completion for Redo-History Algorithms

By completing losers, creating CLEs, and advancing page sequence numbers during undo, upon completed restart the log can be truncated at the SystemRedoLSN (without need for flushing)

(Minor) problem:
repeated crashes during undo lead to
multiple-times inverse actions that could make successive restarts longer

Example:

10: write(t_i,a)
20: write(t_i,b)
30: write(t_i,c)
first crash: redo 10, 20, 30
need to undo 30, 20, 10
40: write(t_i,c)^{-1}
50: write(t_i, b)^{-1}
second crash: redo 10, 20, 30, 40, 50
need to undo 50, 40, 30, 20, 10
60: (write(t_i, b)^{-1})^{-1}
70: (write(t_i, c)^{-1})^{-1}
80: write(t_i,c)^{-1}
90: write(t_i, b)^{-1}
100: write(t_i, a)^{-1}
second restart complete
Next Undo Sequence Number Backward Chaining

Multiple-times inverse actions can be avoided by backward chaining a CLE to the predecessor of its corresponding forward action and following this NextUndoSeqNo backward chain during undo

Example:  
10: write(t_1, a), NextUndoSeqNo=nil  
20: write(t_i, b), NextUndoSeqNo=10  
30: write(t_i, c), NextUndoSeqNo=20  
first crash: redo 10, 20, 30; need to undo 30, 20, 10  
40: write(t_i, c)^{-1}, NextUndoSeqNo=20  
50: write(t_i, b)^{-1}, NextUndoSeqNo=10  
second crash: redo 10, 20, 30, 40, 50; need to undo 10  
60: write(t_i, a)^{-1}, NextUndoSeqNo=nil  
second restart complete
Illustration of Next Undo Sequence Number Backward Chaining

- **NextUndoSeqNo´s**
- **PreviousSeqNo´s**

Log:

- \( \begin{align*}
\text{begin (} t_i \text{)} \\
\text{10: write (} t_i, a \text{)} \\
\text{20: write (} t_i, b \text{)} \\
\text{30: write (} t_i, c \text{)} \\
\text{40: write (} t_i, c \text{)} \\
\text{50: write (} t_i, b \text{)} \\
\text{60: write (} t_i, a \text{)} \\
\end{align*} \)
Undo Pass with CLEs and NextUndoSeqNo Backward Chaining (1)

undo pass ():
    ActiveTrans := empty;
    for each t in losers do
        ActiveTrans += t;
        ActiveTrans[t].LastSeqNo := losers[t].LastSeqNo;
    end /*for*/;
    while there exists t in losers
        such that losers[t].LastSeqNo <> nil
        do
            nexttrans = TransNo in losers
                such that losers[nexttrans].LastSeqNo =
                max {losers[x].LastSeqNo | x in losers};
            nextentry := losers[nexttrans].LastSeqNo;
            if StableLog[nextentry].ActionType = compensation
                then
                    losers[nexttrans].LastSeqNo :=
                        StableLog[nextentry].NextUndoSeqNo;
                end /*if*/;
    end while;

Undo Pass with CLEs and NextUndoSeqNo Backward Chaining (2)

if StableLog[nextentry].ActionType = write then
    pageno := StableLog[nextentry].PageNo; fetch (pageno);
    if DatabaseCache[pageno].PageSeqNo
        >= nextentry.LogSeqNo then
        newlogentry.LogSeqNo := new sequence number;
        newlogentry.ActionType := compensation;
        newlogentry.PreviousSeqNo :=
            ActiveTrans[transid].LastSeqNo;
        newlogentry.NextUndoSeqNo :=
            nextentry.PreviousSeqNo;
        newlogentry.RedoInfo :=
            inverse action of the action in nextentry;
        ActiveTrans[transid].LastSeqNo :=
            newlogentry.LogSeqNo;
        LogBuffer += newlogentry;
        read and write (StableLog[nextentry].PageNo)
            according to StableLog[nextentry].UndoInfo;
        DatabaseCache[pageno].PageSeqNo :=
            newlogentry.LogSeqNo;
    end /*if*/;
Undo Pass with CLEs and NextUndoSeqNo Backward Chaining (3)

losers[nexttrans].LastSeqNo =
    StableLog[nextentry].PreviousSeqNo;
end /*if*/;

if StableLog[nextentry].ActionType = begin then
    newlogentry.LogSeqNo := new sequence number;
    newlogentry.ActionType := rollback;
    newlogentry.TransId :=
        StableLog[nextentry].TransId;
    newlogentry.PreviousSeqNo :=
        ActiveTrans[transid].LastSeqNo;
    LogBuffer += newlogentry;
    ActiveTrans -= transid;
    losers -= transid;
end /*if*/;

end /*while*/;

force ( );
Transaction Abort During Normal Operation with CLEs and NextUndoSeqNo Backward Chaining

(1)

\textbf{abort} (transid):

\begin{verbatim}
logentry := ActiveTrans[transid].LastSeqNo;
while logentry is not nil and
    logentry.ActionType = write or full-write do
newlogentry.LogSeqNo := new sequence number;
newlogentry.ActionType := compensation;
newlogentry.PreviousSeqNo :=
    ActiveTrans[transid].LastSeqNo;
newlogentry.RedoInfo :=
    inverse action of the action in logentry;
newlogentry.NextUndoSeqNo :=
    logentry.PreviousSeqNo;
ActiveTrans[transid].LastSeqNo :=
    newlogentry.LogSeqNo;
LogBuffer += newlogentry;
write (logentry.PageNo)
    according to logentry.UndoInfo;
logentry := logentry.PreviousSeqNo;
end /*while*/
\end{verbatim}
Transaction Abort During Normal Operation with CLEs and NextUndoSeqNo Backward Chaining (2)

```plaintext
newlogentry.LogSeqNo := new sequence number;
newlogentry.ActionType := rollback;
newlogentry.TransId := transid;
newlogentry.PreviousSeqNo :=
    ActiveTrans[transid].LastSeqNo;
newlogentry.NextUndoSeqNo := nil;
LogBuffer += newlogentry;
ActiveTrans -= transid;
force ( );
```
### Example with Undo Completion of Three-Pass Redo-History Recovery

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: begin (t₁)</td>
<td></td>
<td></td>
<td>1: begin(t₁)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>2: begin (t₂)</td>
<td></td>
<td></td>
<td>2: begin (t₂)</td>
<td></td>
</tr>
<tr>
<td>3: write (a, t₁)</td>
<td>a: 3</td>
<td></td>
<td>3: write (a, t₁)</td>
<td></td>
</tr>
<tr>
<td>4: begin (t₃)</td>
<td></td>
<td></td>
<td>4: begin (t₃)</td>
<td></td>
</tr>
<tr>
<td>5: begin (t₄)</td>
<td></td>
<td></td>
<td>5: begin (t₄)</td>
<td></td>
</tr>
<tr>
<td>6: write (b, t₃)</td>
<td>b: 6</td>
<td></td>
<td>6: write (b, t₃)</td>
<td></td>
</tr>
<tr>
<td>7: write (c, t₂)</td>
<td>c: 7</td>
<td></td>
<td>7: write (c, t₂)</td>
<td></td>
</tr>
<tr>
<td>8: write (d, t₁)</td>
<td>d: 8</td>
<td></td>
<td>8: write (d, t₁)</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>9: commit (t₁)</td>
<td></td>
<td></td>
<td>9: commit (t₁)</td>
<td></td>
</tr>
<tr>
<td>10: flush (d)</td>
<td>d: 8</td>
<td></td>
<td>10: flush (d)</td>
<td></td>
</tr>
<tr>
<td>11: write (d, t₃)</td>
<td>d: 11</td>
<td></td>
<td>11: write (d, t₃)</td>
<td></td>
</tr>
<tr>
<td>12: begin (t₅)</td>
<td></td>
<td></td>
<td>12: begin (t₅)</td>
<td></td>
</tr>
<tr>
<td>13: write (a, t₅)</td>
<td>a: 13</td>
<td></td>
<td>13: write (a, t₅)</td>
<td></td>
</tr>
<tr>
<td>14: commit (t₃)</td>
<td></td>
<td></td>
<td>14: commit (t₃)</td>
<td>11, 12, 13, 14</td>
</tr>
<tr>
<td>15: flush (d)</td>
<td>d: 11</td>
<td></td>
<td>15: flush (d)</td>
<td></td>
</tr>
<tr>
<td>16: write (d, t₄)</td>
<td>d: 16</td>
<td></td>
<td>16: write (d, t₄)</td>
<td></td>
</tr>
<tr>
<td>17: write (e, t₂)</td>
<td>e: 17</td>
<td></td>
<td>17: write (e, t₂)</td>
<td></td>
</tr>
<tr>
<td>18: write (b, t₅)</td>
<td>b: 18</td>
<td></td>
<td>18: write (b, t₅)</td>
<td></td>
</tr>
<tr>
<td>19: flush (b)</td>
<td>b: 18</td>
<td></td>
<td>19: flush (b)</td>
<td>16, 17, 18</td>
</tr>
<tr>
<td>20: commit (t₄)</td>
<td></td>
<td></td>
<td>20: commit (t₄)</td>
<td>20</td>
</tr>
<tr>
<td>21: write (f, t₅)</td>
<td>f: 21</td>
<td></td>
<td>21: write (f, t₅)</td>
<td></td>
</tr>
</tbody>
</table>

**시스템 캐러지 및 재시작**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis pass: losers = {t_2, t_5}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (3)</td>
<td>a: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (6)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (a)</td>
<td></td>
<td>a: 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (7)</td>
<td>c: 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (8)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (11)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (13)</td>
<td>a: 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (16)</td>
<td>d: 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (17)</td>
<td>e: 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (18)</td>
<td>b: 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush (a)</td>
<td></td>
<td>a: 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23: compensate (17)</td>
<td>e: 23</td>
<td></td>
<td></td>
<td>23: compensate (17: e, t_2) NextUndoSeqNo: 7</td>
</tr>
<tr>
<td>flush (b)</td>
<td></td>
<td>b: 22</td>
<td></td>
<td>22, 23</td>
</tr>
<tr>
<td>24: compensate (13)</td>
<td>a: 24</td>
<td></td>
<td></td>
<td>24: compensate (13: a, t_5) NextUndoSeqNo: nil</td>
</tr>
<tr>
<td>25: rollback (t_5)</td>
<td></td>
<td></td>
<td></td>
<td>25: rollback (t_5)</td>
</tr>
</tbody>
</table>

†SECOND SYSTEM CRASH AND SECOND RESTART †
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis pass: losers = {t_2, t_5}</td>
<td>consider-redo (3)</td>
<td>a: 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>consider-redo (6)</td>
<td>b: 22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>redo (7)</td>
<td>c: 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>consider-redo (8)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>consider-redo (11)</td>
<td>d: 11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>consider-redo (13)</td>
<td>a: 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (16)</td>
<td>d: 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (17)</td>
<td>e: 17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (18)</td>
<td>b: 22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consider-redo (22)</td>
<td>b: 22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>redo (23)</td>
<td>e: 23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26: compensate (13)</td>
<td>a: 26</td>
<td></td>
<td></td>
<td>26: compensate (13, e: t_2)</td>
<td>NextUndoSeqNo: nil</td>
</tr>
<tr>
<td>27: rollback (t_3)</td>
<td></td>
<td></td>
<td></td>
<td>27: rollback (t_3)</td>
<td></td>
</tr>
<tr>
<td>28: compensate (7)</td>
<td>c: 28</td>
<td></td>
<td></td>
<td>32: compensate (7: c, t_2)</td>
<td>NextUndoSeqNo: nil</td>
</tr>
<tr>
<td>33: rollback (t_2)</td>
<td></td>
<td></td>
<td></td>
<td>31: rollback (t_2)</td>
<td></td>
</tr>
<tr>
<td>force</td>
<td></td>
<td></td>
<td></td>
<td>26, 27, 28, 29</td>
<td></td>
</tr>
</tbody>
</table>

SECOND RESTART COMPLETE: RESUME NORMAL OPERATION
Correctness of Undo Completion with CLEs and NextUndoSeqNo Backward Chaining

**Theorem 13.8:**
The method for undo completion, based on executing inverse actions and creating CLEs that are backward-chained to reflect the next undo log sequence numbers, preserves the correctness of the three-pass or two-pass redo-history recovery algorithms.

**Proof sketch:**
The following invariant holds:
\[
\forall \text{log sequence numbers } s \in \text{stable log such that all more recent log entries of losers, including } s, \text{ have been processed by the undo pass: }
\forall u \in \text{stable log with } u.\text{LogSeqNo} \geq s.\text{LogSeqNo}: \forall o \in \text{stable log: (u.\text{TransId} \in \text{losers and } o.\text{TransId} = u.\text{TransId and o.LogSeqNo} > u.\text{NextUndoSeqNo}) } \Rightarrow o \notin \text{cached db}
\]
Chapter 13: Page-Model
Crash Recovery Algorithms

• 13.2 Basic Data Structures
• 13.3 Redo-Winners Paradigm
• 13.4 Redo-History Paradigm
• 13.5 Lessons Learned
Lessons Learned

- **Redo-history algorithm preferable**
  because of uniformity, no need for page flush during restart, simplicity, and robustness
  (and extensibility towards object model, see Chapter 14)
- **Main ingredients are:**
  - three passes for log analysis, redo, undo
  - light-weight checkpoints for log truncation
  - additional flush log entries for further savings of redo cost
  - compensation log entries
    for transaction rollback and undo completion