CSE 444: Database Internals

Section 6:

Transactions - Recovery

Review in this section

1. UNDO logging

2. REDO logging

3. Updating ARIES Data Structures

Undo Logging

- Two Rules:
  - – 1. If a transaction writes element **X**, then the log record of this update **<T,X,v>** must be written to disk before the new value of **X** is written to disk.
  - – 2. If a transaction commits, then the **COMMIT** must be written to disk only after all elements changed by the transaction have been written to disk.

UNDO LOG RULES

1. <T, X, v> before OUTPUT(X)
2. OUTPUT(X) before <COMMIT>

Act				Disk A	Disk B	Log
						<START T>
INPU				8	8	
READ				8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

When recovering (with UNDO logging)…

- We can not simply ignore the log before a recent commit
  - Many transactions interleave at once. If we truncate before a commit for a transaction, any information about those unfinished transactions would be lost.

- Instead, we can use checkpoint the log periodically…

Review: Checkpointing

- **Checkpointing (naïve)**
  - Write a <START CKPT(T1,…,Tk)>. Flush log to disk
  - **Stop accepting new transactions**
  - Wait until all active transactions abort/commit
  - Write <CKPT>. Flush log to disk.
  - Resume accepting transactions

- **Nonquiescent Checkpointing**
  - Write a <START CKPT(T1,…,Tk)>. Flush log to disk
  - Continue normal operation
  - When all of T1,…,Tk have completed, write <END CKPT>. Flush log to disk
  - **More efficient, system does not seem to be stalled**

Problem 1. UNDO Logging

```
LSN1        <START T1>
LSN2        <T1 X 5>
LSN3        <START T2>
LSN4        <T1 Y 7>
LSN5        <T2 X 9>
LSN6        <START T3>
LSN7        <T3 Z 11>
LSN8        <COMMIT T1>
LSN9        <START CKPT(T2,T3)>
LSN10       <T2 X 13>
LSN11       <T3 Y 15>
            *CRASH*
```

1.
Show how far back in the recovery manager needs to read the log

(which LSN do we need to read up to?)

UNDO: How far to scan log from the end?

- **Case 1**: See **<END CKPT>** first
  - All incomplete transactions began after <START CKPT…>
- **Case 2:** See **<START CKPT(T1..TK)>** first
  - Incomplete transactions began after <START CKPT…> or incomplete ones among T1.. TK
  - Find the earliest <START Ti> among them
  - At most we have to go until the previous <START CKPT>…<END CKPT>

```
LSN1     <START T1>
LSN2     <T1 X 5>
LSN3     <START T2>
LSN4     <T1 Y 7>
LSN5     <T2 X 9>
LSN6     <START T3>
LSN7     <T3 Z 11>
LSN8     <COMMIT T1>
LSN9     <START CKPT(T2,T3)>
LSN10    <T2 X 13>
LSN11    <T3 Y 15>
         *CRASH*
```

Problem 1. UNDO Logging

LSN1	<START T1>
LSN2	<T1 X 5>
LSN3	<START T2>
LSN4	<T1 Y 7>
LSN5	<T2 X 9>
LSN6	<START T3>
LSN7	<T3 Z 11>
LSN8	<COMMIT T1>
LSN9	<START CKPT(T2,T3)>
LSN10	<T2 X 13>
LSN11	<T3 Y 15>
	CRASH

1.
Show how far back in the recovery manager needs to read the log

(write the earliest LSN)

LSN3
(start of the earliest transaction among incomplete transactions)

Problem 1. UNDO Logging

LSN1	<START T1>
LSN2	<T1 X 5>
LSN3	<START T2>
LSN4	<T1 Y 7>
LSN5	<T2 X 9>
LSN6	<START T3>
LSN7	<T3 Z 11>
LSN8	<COMMIT T1>
LSN9	<START CKPT(T2,T3)>
LSN10	<T2 X 13>
LSN11	<T3 Y 15>
	CRASH

2.
Show the actions of the recovery manager during recovery.

Problem 1. UNDO Logging

LSN1	<START T1>
LSN2	<T1 X 5>
LSN3	<START T2>
LSN4	<T1 Y 7>
LSN5	**<T2 X 9>**
LSN6	<START T3>
LSN7	**<T3 Z 11>**
LSN8	<COMMIT T1>
LSN9	<START CKPT(T2,T3)>
LSN10	**<T2 X 13>**
LSN11	**<T3 Y 15>**
	CRASH

2.
Show the actions of the recovery manager during recovery.

Y = 15
X = 13
Z = 11
X = 9

Redo Logging

- One Rule:
  - 1. Before modifying any element X on disk, all log records pertaining to this modification (<T,X,v> and the <COMMIT T>), must appear on disk.

REDO LOG RULE

Both <T, X, v> and <COMMIT >
before OUTPUT(X)
v = new value

Action					Disk B	Log
						<START T>
READ(A,t)					8	
t:=t*2					8	
WRITE(A,t)	16	16		8	8	<T,A,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,16>
						<COMMIT T>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

Problem 2: REDO Logging

1. < START T1 >
2. < T1, A, 10 >
3. < START T2 >
4. < T2, B, 5 >
5. < T1, C, 7 >
6. < START T3 >
7. < T3, D, 12 >
8. < COMMIT T1 >
9. < START CKPT ???? >
10. < START T4 >

11.< T2, E, 5 >
12.< COMMIT T2 >
13.< T3, F, 1 >
14.< T4, G, 15 >
15.< END CKPT >
16.< COMMIT T3 >
17.< START T5 >
18.< T5, H, 3 >
19.< START CKPT ???? >
20.< COMMIT T5 >

*** CRASH ***

1.
What are the correct values of the two <START CKPT ????> records?

Problem 2: REDO Logging

1. < START T1 >
2. < T1, A, 10 >
3. < START T2 >
4. < T2, B, 5 >
5. < T1, C, 7 >
6. < START T3 >
7. < T3, D, 12 >
8. < COMMIT T1 >
9. < START CKPT ???? >
10. < START T4 >
11. < T2, E, 5 >
12. < COMMIT T2 >
13. < T3, F, 1 >
14. < T4, G, 15 >
15. < END CKPT >
16. < COMMIT T3 >
17. < START T5 >
18. < T5, H, 3 >
19. < START CKPT ???? >
20. < COMMIT T5 >

1\.
What are the correct values of the two <START CKPT ????> records?

First START CKPT:
< START CKPT (T2, T3) >

Second START CKPT:
< START CKPT (T4, T5) >

Problem 2: REDO Logging **(Checkpoint)**

1. < START T1 >
2. < T1, A, 10 >
3. < START T2 >
4. < T2, B, 5 >
5. < T1, C, 7 >
6. < START T3 >
7. < T3, D, 12 >
8. < COMMIT T1 >
9. **< START CKPT T2,T3 >**
10. < START T4 >
11. < T2, E, 5 >
12. < COMMIT T2 >
13. < T3, F, 1 >
14. < T4, G, 15 >
15. **< END CKPT >**
16. < COMMIT T3 >
17. < START T5 >
18. < T5, H, 3 >
19. < START CKPT T4,T5 >
20. < COMMIT T5 >

NOTE:

<Commit T3> after <END CKPT>

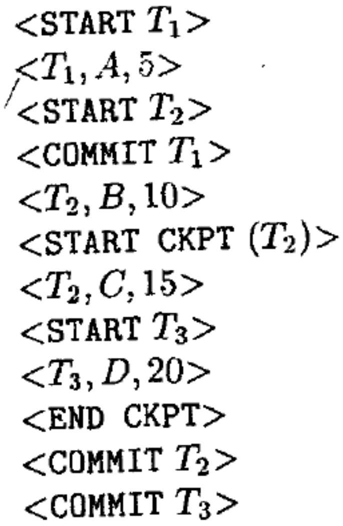
What are we CKPTing?

The transactions that committed before **<START CKPT>**

REDO: How far to scan log from the start?

- **Identify committed transactions**

- Case 1: See <END CKPT> first
  - All committed transactions before <START CKPT (T1.. TK)> are written
  - Consider T1.. Tk, or transactions that started after <START CKPT…>, trace back until earliest <START Ti>



<START T_1>
<$T_1, A, 5$>
<START T_2>
<COMMIT T_1>
<$T_2, B, 10$>
<START CKPT (T_2)>
<$T_2, C, 15$>
<START T_3>
<$T_3, D, 20$>
<END CKPT>
<COMMIT T_2>
<COMMIT T_3>

REDO: How far to scan log from the start?

- **Identify committed transactions**

- Case 1: See <END CKPT> first
  - All committed transactions before <START CKPT (T1.. TK)> are written
  - Consider T1.. Tk, or transactions that started after <START CKPT…>, trace back until earliest <START Ti>

- Case 2: See <START CKPT(T1..TK)> first
  - Committed transactions before START CKPT might not have been written
  - Find previous <END CKPT>, its matching <START CKPT(S1, … Sm)>
  - Redo committed transactions that started after <START CKPT T1..Tk> or S1.. Sm

Problem 2:   REDO Logging

1. < START T1 >
2. < T1, A, 10 >
3. < START T2 >
4. < T2, B, 5 >
5. < T1, C, 7 >
6. < START T3 >
7. < T3, D, 12 >
8. < COMMIT T1 >
9. < START CKPT T2,T3 >
10.< START T4 >
11.< T2, E, 5 >
12.< COMMIT T2 >
13.< T3, F, 1 >
14.< T4, G, 15 >
15.< END CKPT >
16.< COMMIT T3 >
17.< START T5 >
18.< T5, H, 3 >
19.< START CKPT T4,T5 >
20.< COMMIT T5 >

2.
What fragment of the log does the recovery manager need to read?

Problem 2: REDO Logging

1. < START T1 >
2. < T1, A, 10 >
3. **< START T2 >**
4. < T2, B, 5 >
5. < T1, C, 7 >
6. **< START T3 >**
7. < T3, D, 12 >
8. < COMMIT T1 >
9. **< START CKPT T2,T3>**
10. < START T4 >
11. < T2, E, 5 >
12. **< COMMIT T2 >**
13. < T3, F, 1 >
14. < T4, G, 15 >
15. < END CKPT >
16. **< COMMIT T3 >**
17. < START T5 >
18. < T5, H, 3 >
19. **< START CKPT T4,T5>**
20. < COMMIT T5 >

2.
What fragment of the log does the recovery manager need to read?

- We know there was a commit for T5.
- In the previous START CKPT, T2 and T3 were the two active transactions. Both transactions committed and must thus be redone.
- T2 was the earliest one

Problem 2: REDO Logging

1. < START T1 >
2. < T1, A, 10 >
3. < START T2 >
4. < T2, B, 5 >
5. < T1, C, 7 >
6. < START T3 >
7. < T3, D, 12 >
8. < COMMIT T1 >
9. < START CKPT T2,T3 >
10. < START T4 >
11. < T2, E, 5 >
12. < COMMIT T2 >
13. < T3, F, 1 >
14. < T4, G, 15 >
15. < END CKPT >
16. < COMMIT T3 >
17. < START T5 >
18. < T5, H, 3 >
19. < START CKPT T4,T5 >
20. < COMMIT T5 >

3\.
Which elements are recovered by the redo recovery manager? compute their values after recovery.

Problem 2: REDO Logging

1. < START T1 >
2. < T1, A, 10 >
3. < START T2 >
4. **< T2, B, 5 >**
5. < T1, C, 7 >
6. < START T3 >
7. **< T3, D, 12 >**
8. < COMMIT T1 >
9. < START CKPT T2,T3 >
10. < START T4 >
11. **< T2, E, 5 >**
12. < COMMIT T2 >
13. **< T3, F, 1 >**
14. < T4, G, 15 >
15. < END CKPT >
16. < COMMIT T3 >
17. < START T5 >
18. **< T5, H, 3 >**
19. < START CKPT T4,T5 >
20. < COMMIT T5 >

3.

Which elements are recovered by the redo recovery manager? compute their values after recovery.

All changes by T2, T3, T5 (committed)

B=5
D=12
E=5
F=1
H=3

Review: ARIES Data Structures (UNDO/REDO Logging)

Example.

1. T_{1000} changes the value of **A from “abc” to “def”** on **page P500**
2. T_{2000} changes the value of **B from “hij” to “klm”** on **page P600**
3. T_{2000} changes the value of **D from “mnp” to “qrs”** on **page P500**
4. T_{1000} changes the value of **C from “tuv” to “wxy”** on **page P505**
5. T_{2000} commits and the end log record is written
6. T_{1000} changes the value of **E from “pq” to “rs”** on **page P700**
7. **P600** is flushed to disk
8. **Crash!!**

See Section 7

ARIES Data Structures

Dirty page table

pageID	recLSN

Transaction table

transID	lastLSN	status

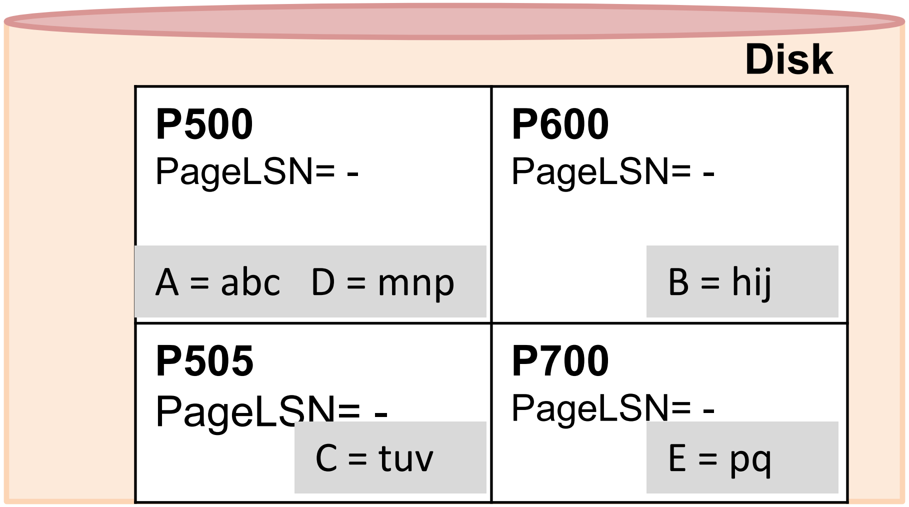
Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101						

Buffer Pool

P500
PageLSN= -
A = abc D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq



Disk

P500
PageLSN= -
A = abc D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq

First operation:

1. T_{1000} changes the value of **A from “abc” to “def”** on **page P500?**

Dirty page table

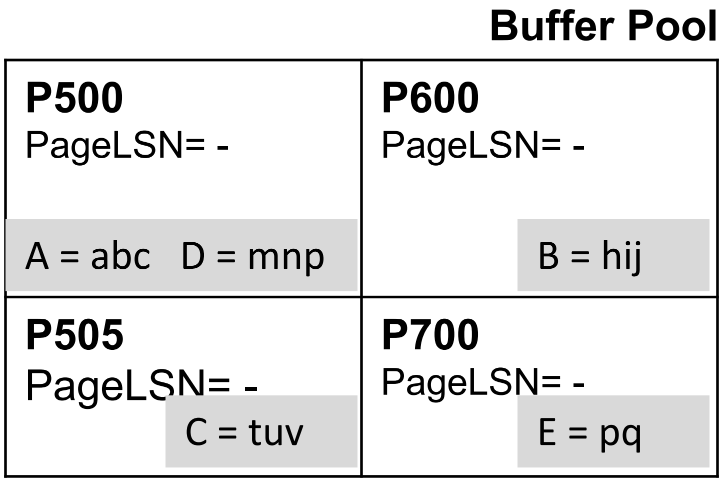
pageID	recLSN

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101						

Transaction table

transID	lastLSN	status



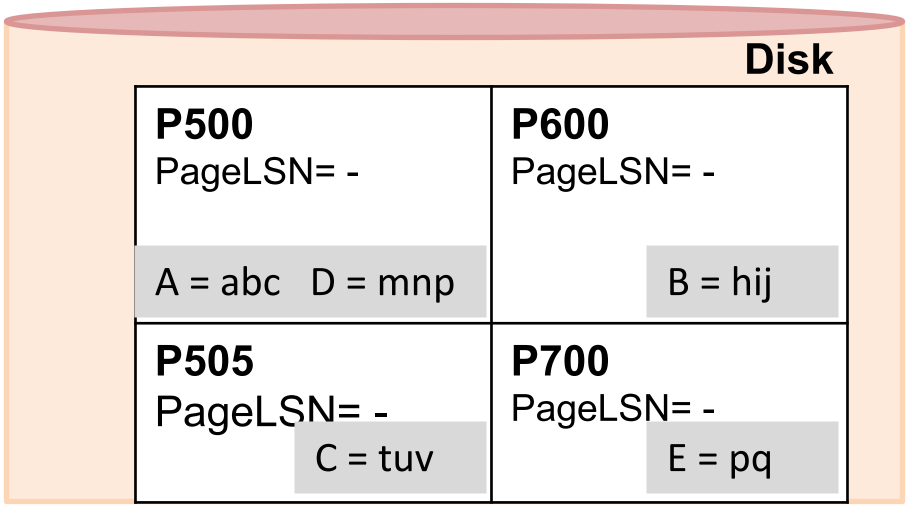
Buffer Pool

P500
PageLSN= -
A = abc D = mnp

P600
PageLSN= -
B = hij

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq



Disk

P500
PageLSN= -
A = abc D = mnp

P600
PageLSN= -
B = hij

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq

Changes

1. T_{1000} changes the value of **A from “abc” to “def”** on **page P500**

Dirty page table

pageID	recLSN
P500	101

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	101	Running

Buffer Pool

P500
PageLSN= 101
A = def D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq

Disk

P500
PageLSN= -
A = abc D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq

Next:

2. T_{2000} changes the value of **B from “hij” to “klm”** on **page P600 ?**

Dirty page table

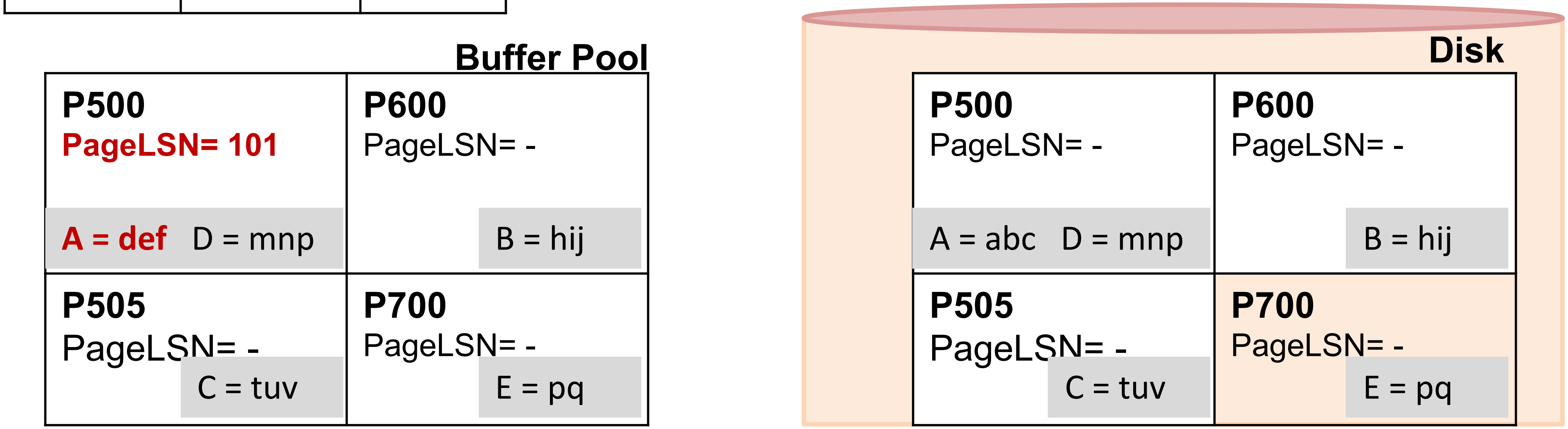
pageID	recLSN
P500	101

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	101	Running



Buffer Pool

P500
PageLSN= 101
A = def D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq

Disk

P500
PageLSN= -
A = abc D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq

Changes:

2. T_{2000} changes the value of B from “hij” to “klm” on page P600 ?

Dirty page table

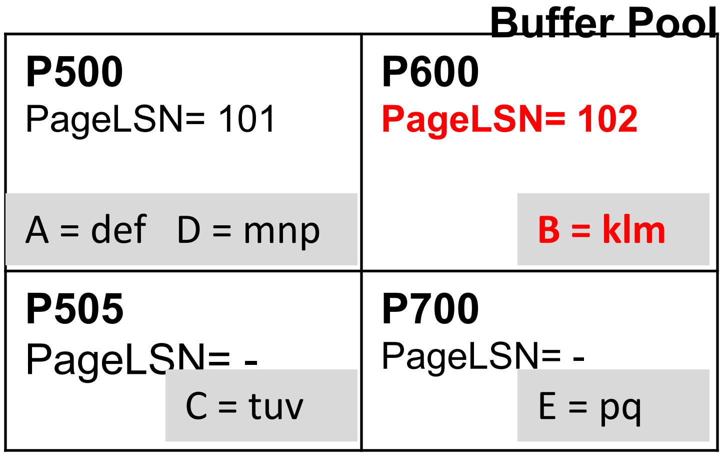
pageID	recLSN
P500	101
P600	**102**

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	**102**	**Running**



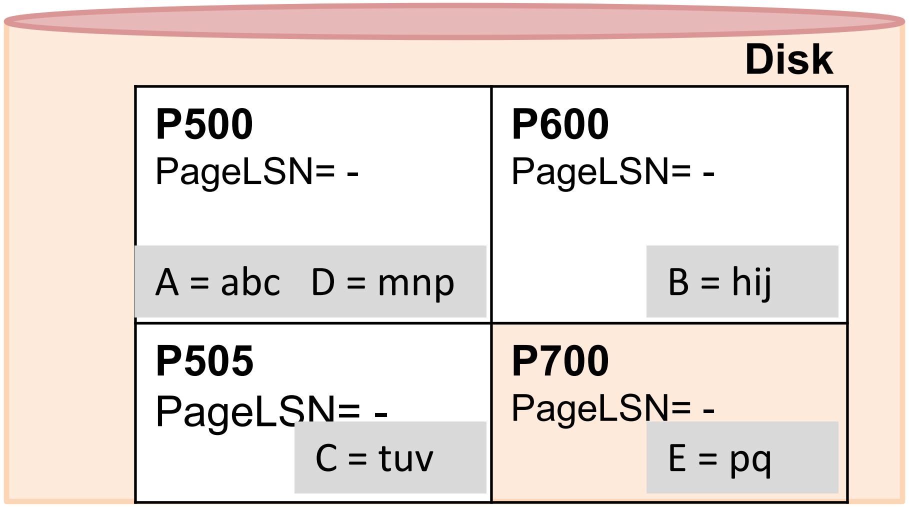
Buffer Pool

P500
PageLSN= 101
A = def D = mnp

P600
PageLSN= 102
B = klm

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq



Disk

P500
PageLSN= -
A = abc D = mnp

P600
PageLSN= -
B = hij

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq

Next:

3. T_{2000} changes the value of **D from “mnp” to “qrs”** on **page P500?**

Dirty page table

pageID	recLSN
P500	101
P600	**102**

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	**102**	**Running**

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	

Buffer Pool

P500
PageLSN= 101
A = def D = mnp
	P600
	PageLSN= 102
	B = klm

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq

Disk

P500
PageLSN= -
A = abc D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq

Changes:

3. T_{2000} changes the value of D from “mnp” to “qrs” on page P500

Dirty page table

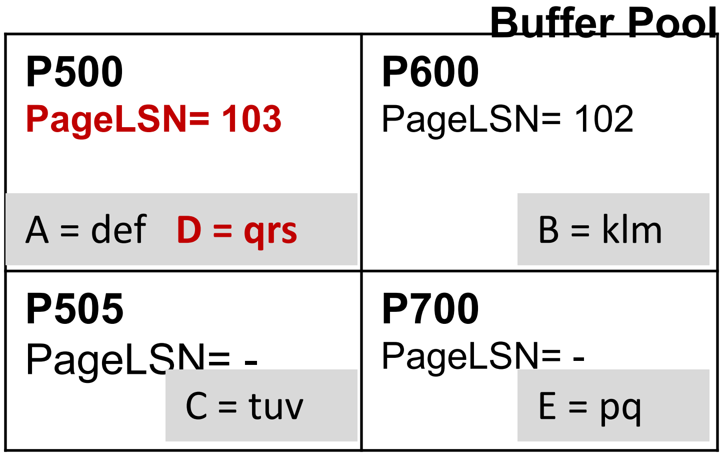
pageID	recLSN
P500	101
P600	102

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	**103**	**Running**



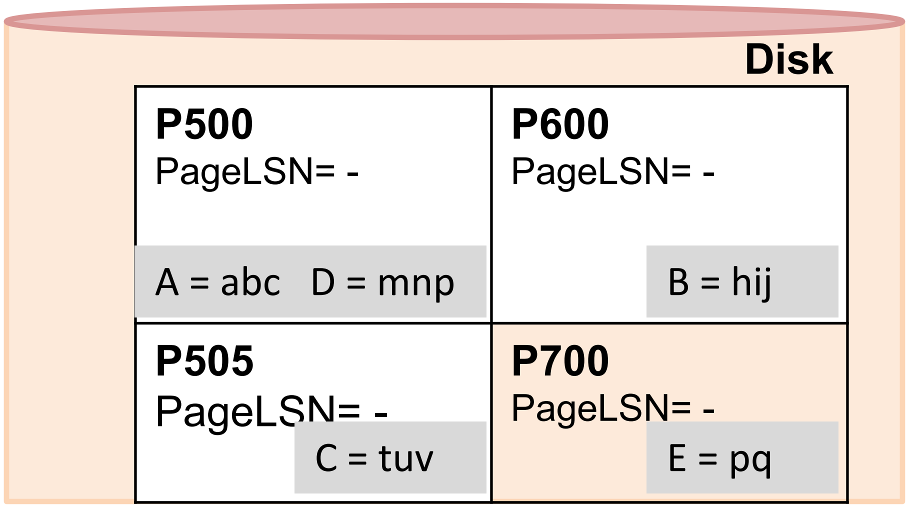
Buffer Pool

P500
PageLSN= 103
A = def D = qrs

P600
PageLSN= 102
B = klm

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq



Disk

P500
PageLSN= -
A = abc D = mnp

P600
PageLSN= -
B = hij

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq

Next:

4. T_{1000} changes the value of **C from “tuv” to “wxy”** on **page P505?**

Dirty page table

pageID	recLSN
P500	101
P600	102

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	**103**	**Running**

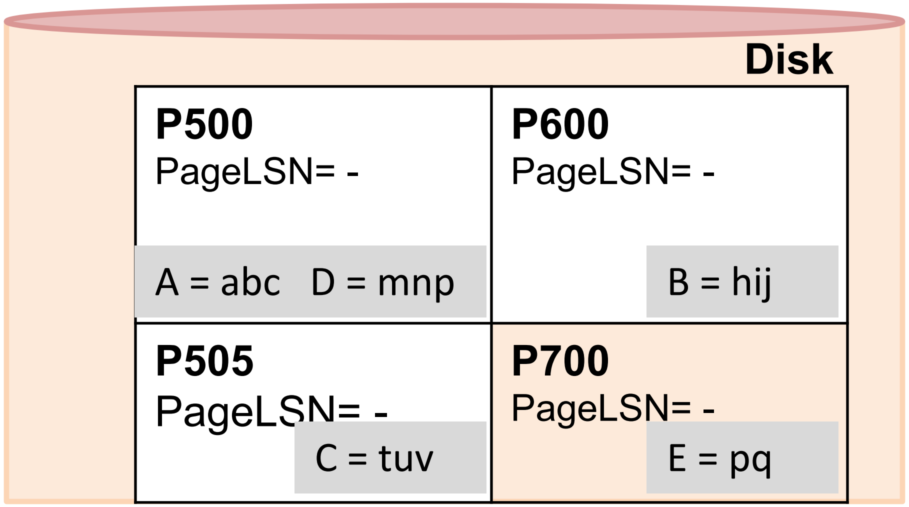
Buffer Pool

P500
PageLSN= 103
A = def **D = qrs**

P600
PageLSN= 102
B = klm

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq


Disk

P500
PageLSN= -
A = abc D = mnp

P600
PageLSN= -
B = hij

P505
PageLSN= -
C = tuv

P700
PageLSN= -
E = pq

Changes:

4. T_{1000} changes the value of C from “tuv” to “wxy” on page P505?

Dirty page table

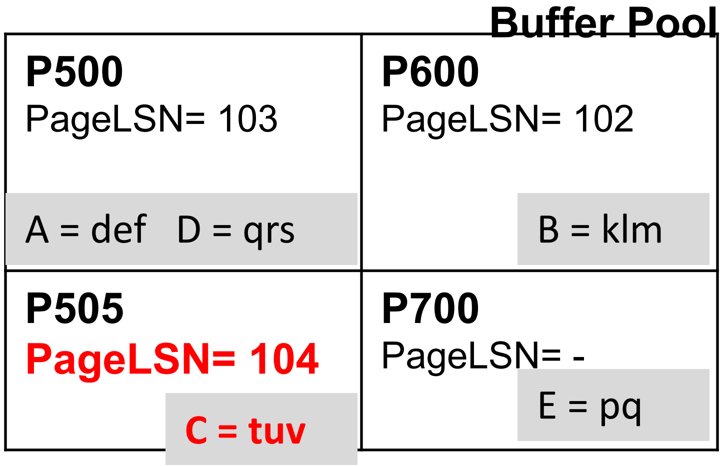
pageID	recLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
T_{2000}	103	Running

Log

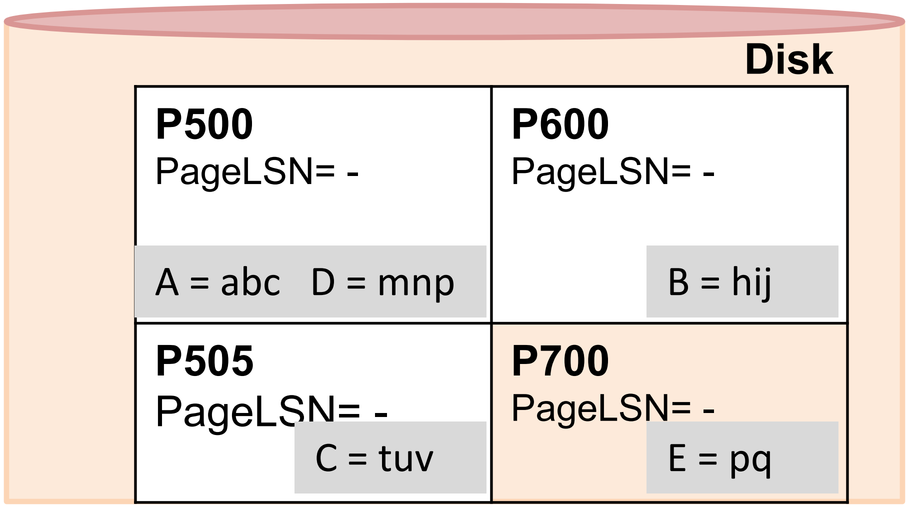
LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T_{1000}	P505	Write C “tuv” -> “wxy”	Update	-



Buffer Pool

P500
PageLSN= 103
A = def D = qrs
	P600
	PageLSN= 102
	B = klm

P505
PageLSN= 104
C = tuv
	P700
	PageLSN= -
	E = pq



Disk

P500
PageLSN= -
A = abc D = mnp
	P600
	PageLSN= -
	B = hij

P505
PageLSN= -
C = tuv
	P700
	PageLSN= -
	E = pq