Rollback and Recovery

So far

- ACID = CI + AD
- 2PL enforces Consistency and Isolation while allowing maximum concurrency
- Consistency is also a responsibility of the transaction (and underlying data model)
- Atomicity = All or nothing
- Durability = Once committed the changes are forever

Atomicity

- A transaction
  - Either completes (commits) with all its changes being permanent
  - Or it aborts (rollbacks) with no change to the state of the database

Durability

- A committed transactions changes are forever in face of system crashes (power failure, media failure, program failure, etc.)

Read-only transactions

- No need to "undo" (there were no changes made)
- Nothing updated = so durability is a non-issue

Buffer pool and Transactions

- A page to be read or updated in brought into pool
  - It’s pinned to the pool
- Changes are written to the page in the buffer pool
  - When done with update, the page is marked dirty and unpinned
  - Page remains in the pool till replaced

Why not keep page pinned till done?

- Keep page pinned till the transaction either commits or aborts? (no steal)
- Pro
  - aborts (rollback) would be easy as we can simply discard the page in the buffer pool
    - (But must lock the whole page for this)
- Con
  - Too many pages may be unnecessarily be in memory reducing performance

“Stealing” pages

- If page unpinned, the changes may or may not be on disk (depending on whether it was replaced or not)
  - Need to undo changes to the (disk-resident) page in case of abort
    - Keep track of the changes in a log, so that the changes can be undone
    - Log is written to the disk before the disk page is written
  - But better usage of the buffer pool

Why not force write at commit?

- At commit time, we can force writing of all dirty pages of this transaction
- Con:
  - A write bottleneck at commit time
  - A frequently accessed pages would be written several times impacting performance
- Pro:
  - All changes are durable - no need to "redo" any updates in face of system failure

No Force

- A dirty page remains in buffer pool at commit
- What happens to enforce durability?
    - Must “log” the changes somewhere safe before committing, so that we can “replay” the changes in case of system failure in case the page is not yet written to disk
    - The log must be written to disk before commit is complete

WAL

- Log is written to disk before a dirty page is written to disk
    - Atomicity
- Log is written to disk before a transaction commits
    - Durability

Log

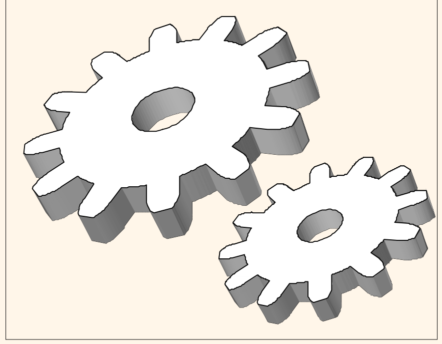
- Ordered list of changes to the data organized by transactions and time
    - LSN - log sequence number (monotonically increasing)
    - type of log record (update, commit, abort, etc.)
    - prevLSN = previous LSN for this transaction
    - TransactionId
    - PageId
    - Where in page (offset)
    - Length (how many bytes)
    - Old data
    - New data

What else?

- System maintains flushedLSN (max LSN of the log records that have been written to stable storage)
- The pages contain a pageLSN - the LSN of the most recent log record that updated the page
  - pageLSN <= flushedLSN before a dirty page is written to disk
- The transactions table maintains lastLSN for the transaction
  - How one would find the prevLSN for a new log record for transaction
- The dirty page table maintains a recLSN (recoveryLSN) which is the LSN of the first update to the page - this would be the earliest log that would be redone if necessary

Normal Execution

- All writes are logged in the log
- Upon commit the log is flushed up-to the transaction's lastLSN
- Upon a dirty page being written the log is flushed to the pageLSN
- 2PL (usually strict or conservative 2PL)
- Once in a while the log and system is checkpointed

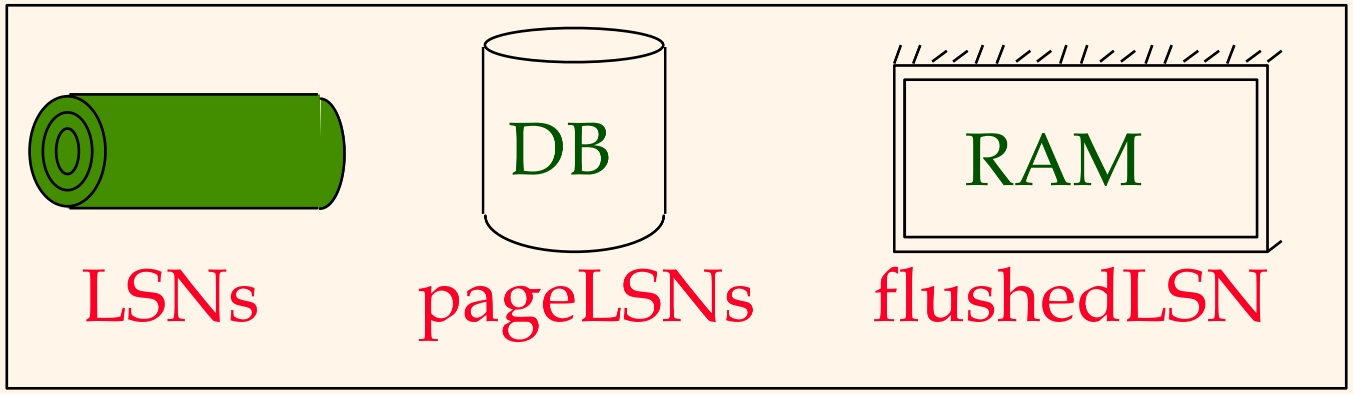


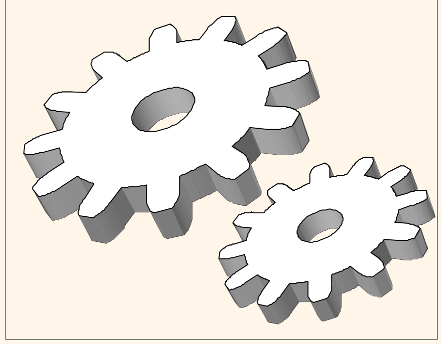
Write-Ahead Logging (WAL)

- The Write-Ahead Logging Protocol:
  ① Must force the log record for an update *before* the corresponding data page gets to disk.
  , Must write all log records for a Xact *before commit*.
- #1 guarantees Atomicity.
- #2 guarantees Durability.

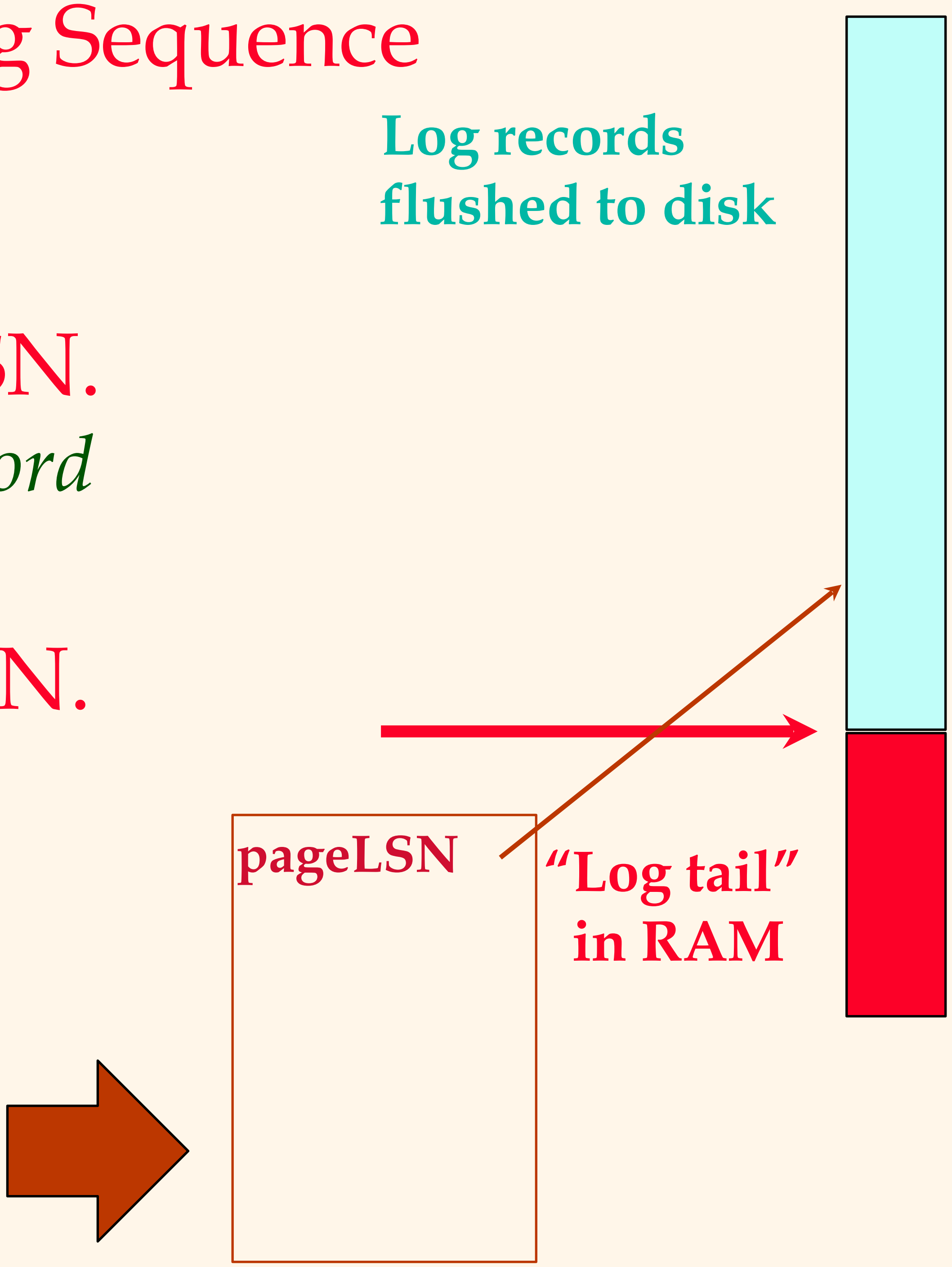
- Exactly how is logging (and recovery!) done?
  - We'll study the ARIES algorithms.

WAL & the Log

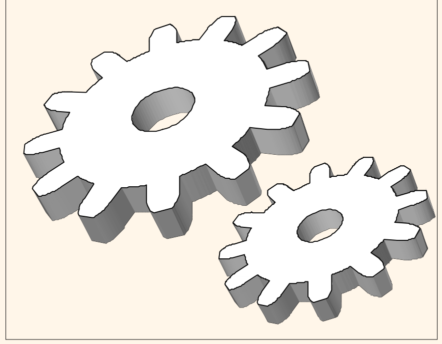




- Each log record has a unique Log Sequence Number (LSN).
  - LSNs always increasing.
- Each *data page* contains a pageLSN.
  - The LSN of the most recent *log record* for an update to that page.
- System keeps track of flushedLSN.
  - The max LSN flushed so far.
- WAL: *Before* a page is written,
  - pageLSN ≤ flushedLSN



Log Records



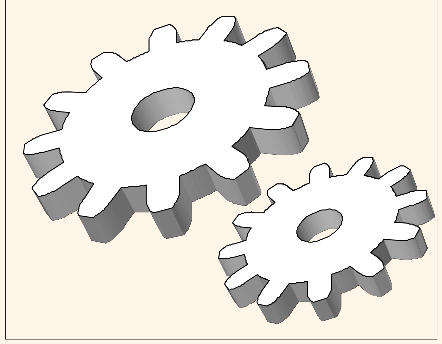
LogRecord fields:

prevLSN
XID
type
pageID
length
offset
before-image
after-image

update records only: pageID, length, offset, before-image, after-image

Possible log record types:

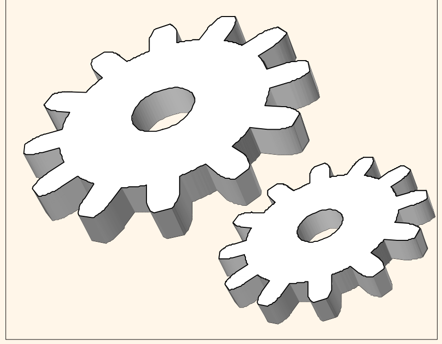
- **Update**
- **Commit**
- **Abort**
- **End** (signifies end of commit or abort)
- Compensation Log Records (CLRs)
  - for UNDO actions



Other Log-Related State

- Transaction Table:
  - One entry per active Xact.
  - Contains XID, status (running/commited/aborted), and lastLSN.
- Dirty Page Table:
  - One entry per dirty page in buffer pool.
  - Contains recLSN -- the LSN of the log record which ***first*** caused the page to be dirty.

Checkpointing

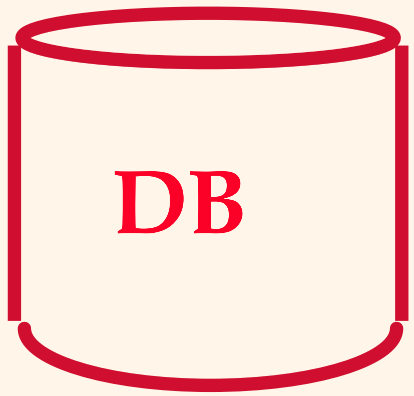


- Periodically, the DBMS creates a checkpoint, in order to minimize the time taken to recover in the event of a system crash. Write to log:
  - begin_checkpoint record: Indicates when chkpt began.
  - end_checkpoint record: Contains current *Xact table* and *dirty page table*. This is a `fuzzy checkpoint':
    - Other Xacts continue to run; so these tables accurate only as of the time of the begin_checkpoint record.
    - No attempt to force dirty pages to disk; effectiveness of checkpoint limited by oldest unwritten change to a dirty page. (So it's a good idea to periodically flush dirty pages to disk!)
    - The log is flushed!
  - Store LSN of chkpt record in a safe place (*master* record).

The Big Picture: What's Stored Where

LogRecords

prevLSN
XID
type
pageID
length
offset
before-image
after-image



Data pages

each
with a
pageLSN

master record

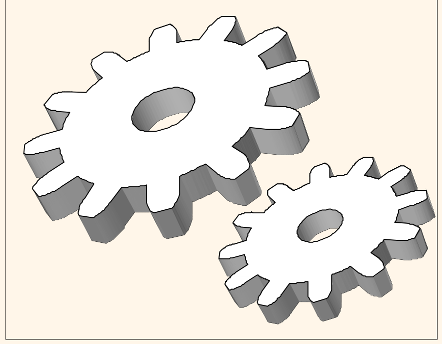
Xact Table

lastLSN
status

Dirty Page Table

recLSN

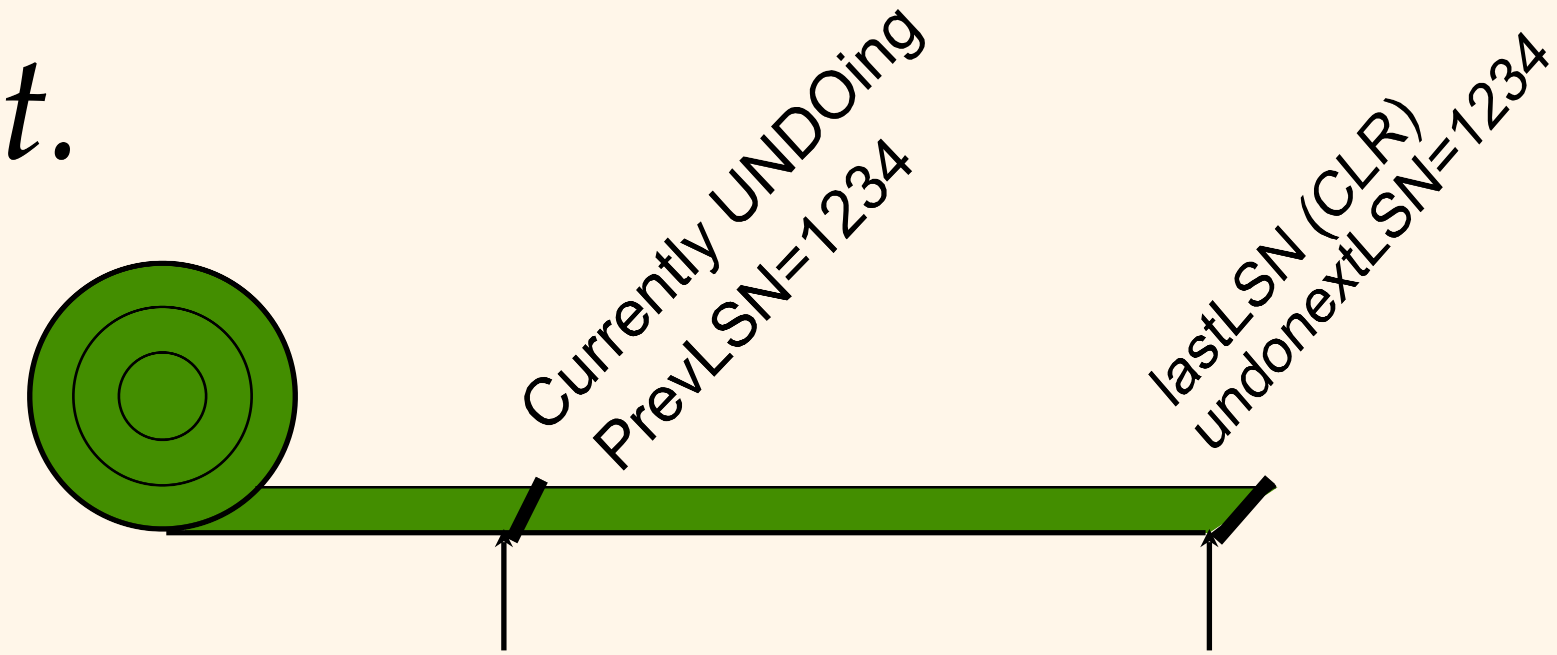
flushedLSN



Simple Transaction Abort

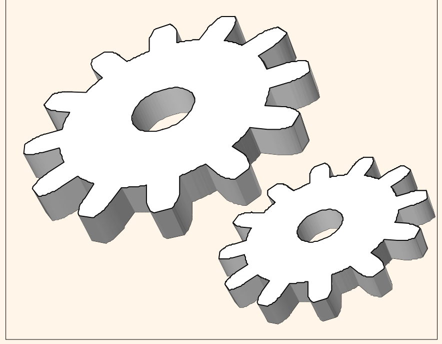
- For now, consider an explicit abort of a Xact.
  - No crash involved.
- We want to "play back" the log in reverse order, UNDOing updates.
  - Get lastLSN of Xact from Xact table.
  - Can follow chain of log records backward via the prevLSN field.
  - Before starting UNDO, write an *Abort* log record.
    - For recovering from crash during UNDO!

Abort, cont.



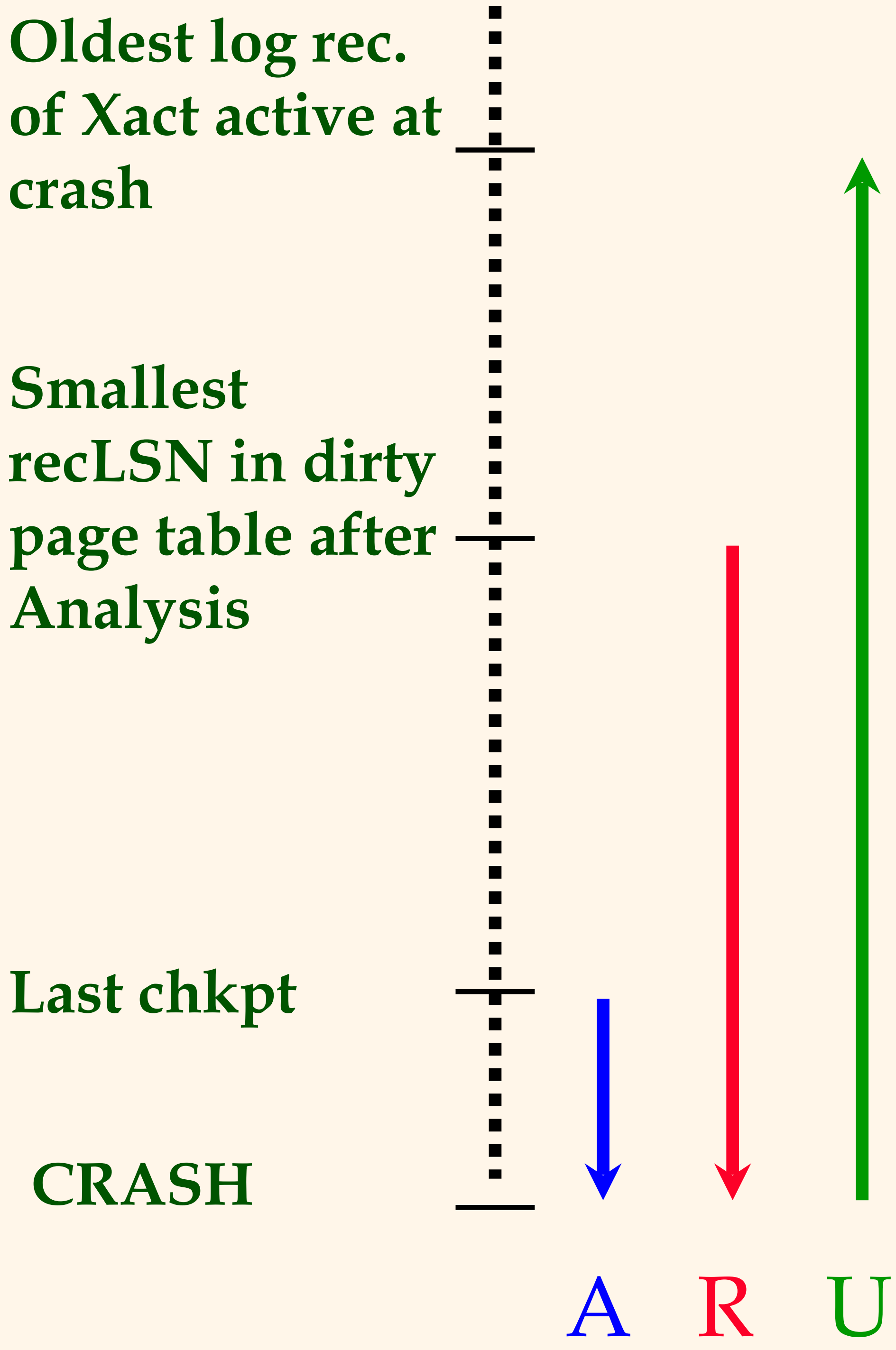
- To perform UNDO, must have a lock on data!
  - No problem!
- Before restoring old value of a page, write a CLR:
  - You continue logging while you UNDO!!
  - CLR has one extra field: undonextLSN
    - Points to the next LSN to undo (i.e. the prevLSN of the record we're currently undoing).
  - CLRs *never* Undone (but they might be Redone when repeating history: guarantees Atomicity!)
- At end of UNDO, write an "end" log record.

Transaction Commit



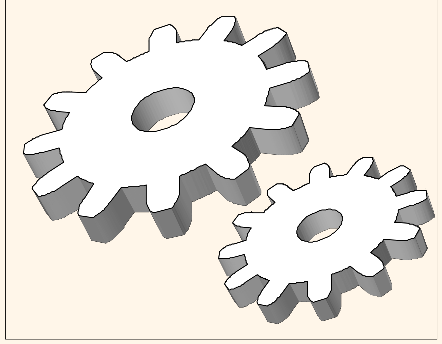
- Write commit record to log.
- All log records up to Xact's lastLSN are flushed.
  - Guarantees that flushedLSN ≥ lastLSN.
  - Note that log flushes are sequential, synchronous writes to disk.
  - Many log records per log page.
- Commit() returns.
- Write end record to log.

Crash Recovery: Big Picture


Oldest log rec. of Xact active at crash
Smallest recLSN in dirty page table after Analysis
Last chkpt
CRASH
A
R
U

- Start from a checkpoint (found via master record).
- Three phases. Need to:
  - Figure out which Xacts committed since checkpoint, which failed (Analysis).
  - REDO ***all*** actions.
    - (repeat history)
  - UNDO effects of failed Xacts.

Recovery: The Analysis Phase



- Reconstruct state at checkpoint.
  - via end_checkpoint record.
- Scan log forward from checkpoint.
  - End record: Remove Xact from Xact table.
  - Other records: Add Xact to Xact table, set lastLSN=LSN, change Xact status on commit.
  - Update record: If P not in Dirty Page Table,
    - Add P to D.P.T., set its recLSN=LSN.

Analysis Phase

Dirty Page Table

PageId	recLSN

Transaction Table

xactId	lastLSN

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700

Analysis Phase

Dirty Page Table

PageId	recLSN
P500	10

Transaction Table

xactId	lastLSN
T1000	10

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700

Analysis Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20

Transaction Table

xactId	lastLSN
T1000	10
T2000	20

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700

Analysis Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20

Transaction Table

xactId	lastLSN
T1000	10
T2000	30

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700

Analysis Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40

Transaction Table

xactId	lastLSN
T1000	40
T2000	30

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700

Analysis Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40

Transaction Table

xactId	lastLSN
T1000	40

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700

Analysis Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

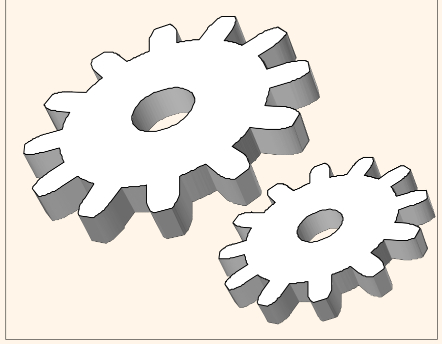
Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700

Recovery: The REDO Phase



- We *repeat History* to reconstruct state at crash:
  - Reapply *all* updates (even of aborted Xacts!), redo CLRs.
- Scan forward from log rec containing smallest recLSN in D.P.T. For each CLR or update log rec LSN, REDO the action unless:
  - Affected page is not in the Dirty Page Table, or
  - Affected page is in D.P.T., but has recLSN > LSN, or
  - pageLSN (in DB) ≥ LSN.
- To REDO an action:
  - Reapply logged action.
  - Set pageLSN to LSN. No additional logging!

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

- Start with the smallest recLSN in the Dirty Page Table, 10 in our example
- Redo unless
  - Page is not in dirty page table
  - Page is in dirty page table but its recLSN > LSN of the log record
  - pageLSN of the page is >= LSN of the log record

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

```
P500:xxxxABCGDExxxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

```
P500:xxxxxABCGDExxxxx     P500:xxxxxDEFGDExxxxx, pageLSN:10
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

```
P500:xxxxABCGDExxxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

```
P500:xxxxxDEFGDExxxxx, pageLSN:10
P600:xxxxxxxxxKLMxxxx, pageLSN:20
```

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

```
P500:xxxxxABCGDExxxxx      P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx      P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

```
P500:xxxxxABCGDExxxxx     P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx     P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx     P505:xxxxWXYxxxxxxxxx, pageLSN:40
P700:xxxACExxxxxxxxxx
```

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

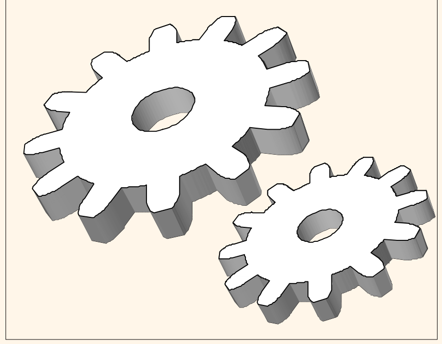
xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

```
P500:xxxxxABCGDExxxxx    P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx    P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx    P505:xxxxWXYxxxxxxxxx, pageLSN:40
P700:xxxACEXxxxxxxxxx    P700:xxxBDFXxxxxxxxxx, pageLSN:60
```

Recovery: The UNDO Phase



ToUndo={ *l* | *l* a lastLSN of a "loser" Xact}

Repeat:

- Choose largest LSN among ToUndo.
- If this LSN is a CLR and undonextLSN==NULL
  - Write an End record for this Xact.
- If this LSN is a CLR, and undonextLSN != NULL
  - Add undonextLSN to ToUndo
- Else this LSN is an update. Undo the update, write a CLR, add prevLSN to ToUndo.

Until ToUndo is empty.

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	60

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after
10	-	T1000	update	P500	5	ABC	DEF
20	-	T2000	update	P600	9	HIJ	KLM
30	20	T2000	update	P500	8	GDE	QRS
40	10	T1000	update	P505	5	TUV	WXY
50	30	T2000	commit				
60	40	T1000	update	P700	3	ACE	BDF

```
P500:xxxxxABCGDExxxxx       P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx       P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx       P505:xxxxWXYxxxxxxxxx, pageLSN:40
P700:xxxACEXxxxxxxxxx       P700:xxxBDFXxxxxxxxxx, pageLSN:60
```

ToUndo: LSN:60

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	70

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40

```
P500:xxxxxABCGDExxxxx     P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx     P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx     P505:xxxxWXYxxxxxxxxx, pageLSN:40
P700:xxxACExxxxxxxxxx     P700:xxxACExxxxxxxxxx, pageLSN:70
```

ToUndo: LSN:40

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10

```
P500:xxxxxABCGDExxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

```
P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx, pageLSN:70
```

ToUndo: LSN:10

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	90

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10
90	80	T1000	CLR	P500	5		ABC	-

```
P500:xxxxxABCGDExxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

```
P500:xxxxxABCQRSxxxxx, pageLSN:90
P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx, pageLSN:70
```

ToUndo:{}

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10
90	80	T1000	CLR	P500	5		ABC	-
100	90	T1000	end					

```
P500:xxxxxABCGDExxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

```
P500:xxxxxABCQRSxxxxx, pageLSN:90
P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx, pageLSN:70
```

Crash while recovery

- Crash doing analysis phase
  - no impact, no changes were made
- Crash doing redo phase
  - no impact, we are just recreating state of affairs at time of crash
- Crash during undo phase
  - how do we deal with CLRs?

Undo Phase Crash

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10

```
P500:xxxxxABCGDExxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

```
P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx, pageLSN:70
```

End of Analysis Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLSN	xactID	type	PageId
10	-	T1000	update	P500
20	-	T2000	update	P600
30	20	T2000	update	P500
40	10	T1000	update	P505
50	30	T2000	commit	
60	40	T1000	update	P700
70	60	T1000	CLR	P700
80	70	T1000	CLR	P505

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10

```
P500:xxxxxABCGDExxxxx     P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx     P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx     P505:xxxxWXYxxxxxxxxx, pageLSN:40
P700:xxxACEXxxxxxxxxx     P700:xxxBDFXxxxxxxxxx, pageLSN:60
```

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10

```
P500:xxxxxABCGDExxxxx     P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx     P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx     P505:xxxxWXYxxxxxxxxx, pageLSN:40
P700:xxxACExxxxxxxxxx     P700:xxxACExxxxxxxxxx, pageLSN:70
```

Redo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10

```
P500:xxxxxABCGDExxxxx     P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx     P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx     P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx     P700:xxxACEXxxxxxxxxx, pageLSN:70
```

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10

```
P500:xxxxxABCGDExxxxx      P500:xxxxxDEFQRSxxxxx, pageLSN:30
P600:xxxxxxxxxHIJxxxx      P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx      P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx      P700:xxxACEXxxxxxxxxx, pageLSN:70
```

ToUndo: LSN:10

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN
T1000	80

Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10
90	80	T1000	CLR	P500	5		ABC	-

```
P500:xxxxxABCGDExxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

```
P500:xxxxxABCQRSxxxxx, pageLSN:90
P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx, pageLSN:70
```

ToUndo: {}

Undo Phase

Dirty Page Table

PageId	recLSN
P500	10
P600	20
P505	40
P700	60

Transaction Table

xactId	lastLSN

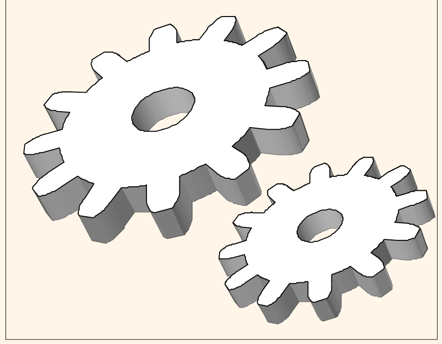
Log

LSN	prevLS	xactID	type	PageId	Offset	before	after	next
10	-	T1000	update	P500	5	ABC	DEF	
20	-	T2000	update	P600	9	HIJ	KLM	
30	20	T2000	update	P500	8	GDE	QRS	
40	10	T1000	update	P505	5	TUV	WXY	
50	30	T2000	commit					
60	40	T1000	update	P700	3	ACE	BDF	
70	60	T1000	CLR	P700	3		ACE	40
80	70	T1000	CLR	P505	5		TUV	10
90	80	T1000	CLR	P500	5		ABC	-
100	90	T1000	end					

```
P500:xxxxxABCGDExxxxx
P600:xxxxxxxxxHIJxxxx
P505:xxxxTUVxxxxxxxxx
P700:xxxACEXxxxxxxxxx
```

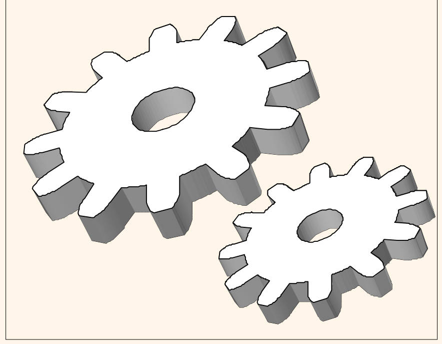
```
P500:xxxxxABCQRSxxxxx, pageLSN:90
P600:xxxxxxxxxKLMxxxx, pageLSN:20
P505:xxxxTUVxxxxxxxxx, pageLSN:80
P700:xxxACEXxxxxxxxxx, pageLSN:70
```

Summary of Logging/Recovery



- Recovery Manager guarantees Atomicity & Durability.
- Use WAL to allow STEAL/NO-FORCE w/o sacrificing correctness.
- LSNs identify log records; linked into backwards chains per transaction (via prevLSN).
- pageLSN allows comparison of data page and log records.

Summary, Cont.



- Checkpointing: A quick way to limit the amount of log to scan on recovery.
- Recovery works in 3 phases:
  - Analysis: Forward from checkpoint.
  - Redo: Forward from oldest recLSN.
  - Undo: Backward from end to first LSN of oldest Xact alive at crash.
- Upon Undo, write CLRs.
- Redo "repeats history": Simplifies the logic!