Recovery II

Checkpoint
ARIES

Alvin Cheung
Aditya Parameswaran
R&G - Chapter 18

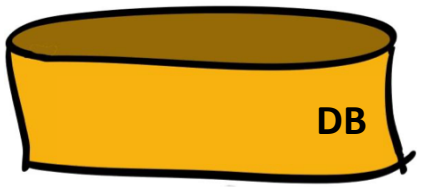
Primitive Operations

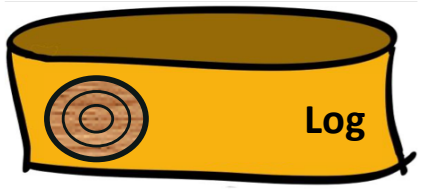
- READ(X,t)
  - copy value of data item X to transaction local variable t
- WRITE(X,t)
  - copy transaction local variable t to data item X

- FETCH(X)
  - read page containing data item X to memory buffer
- FLUSH(X)
  - write page containing data item X to disk

Solution: Write-Ahead Log

- **Log: append-only file containing log records**
  - This is usually on a different disk, separate from the data pages, allowing recovery
- For every update, commit, or abort operation
  - Write a log record
  - Multiple transactions run concurrently, log records are interleaved
- After a system crash, use log to:
  - Redo transactions that did commit
    - Redo ensures Durability
  - Undo transactions that didn't commit
    - Undo ensures Atomicity





Undo Logging

Log records

- <START T>
  - transaction T has begun
- <COMMIT T>
  - T has committed
- <ABORT T>
  - T has aborted
- <T,X,v>
  - T has updated element X, and its *old* value was v

Redo Logging

One minor change to the undo log:

- <T,X,v>= T has updated element X, and its *new* value is v

Problem with Undo / Redo Logging

- We need to recover from the beginning of the log!
  - Computationally expensive
  - Do we always need to start from the very beginning?

The ARIES Recovery Algorithm

ARIES*

- ARIES pieces together several techniques into a comprehensive algorithm
- Developed at IBM Almaden, by C Mohan
- Several variations, e.g. for distributed transactions
- But first, we need to discuss the concept of *checkpoint*

*Algorithms for Recovery and Isolation Exploiting Semantics

Checkpoint

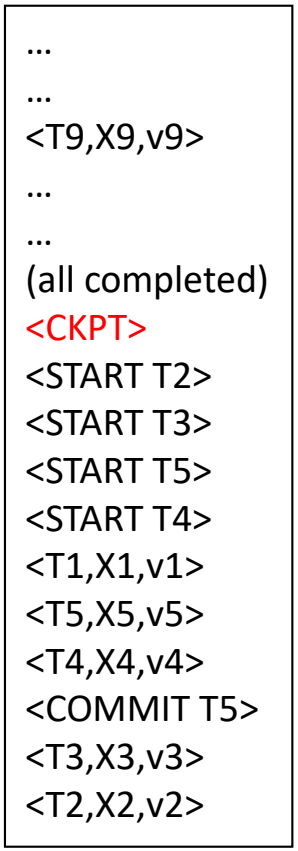
- Idea: save the state the database periodically so that we don't need to always process the entire log

- During a checkpoint:
  - Stop accepting new transactions
  - Wait until **all** current transactions complete (i.e., commit / abort)
  - Flush log to disk
  - Flush all dirty pages to disk
  - Write a <CKPT> log record, flush log again
  - At this point, changes by committed txns have persisted to disk, and aborted txns have rolled back

  - Resume transactions

Undo Recovery with Checkpointing

During recovery,
Stop at first <CKPT>



All txns here are completed
No need to recover
Can truncate this part of the log

Txns T2,T3,T4,T5 need to be recovered

Fuzzy Checkpointing

- Problem with checkpointing: database freezes during checkpoint
  - Not accepting any new transactions!
- Would like to checkpoint while database still processes incoming txns
- Idea: *fuzzy* checkpointing
  - Save state of all txns and page statuses
    - Some txns can be running and dirty pages not flushed yet!
    - Need new data structures to store such info

Fuzzy Checkpointing: Idea

- Keep track of:
  1. txn states (running, committing, etc)
  2. dirty pages and which txn's action first caused page to become dirty

- Save 1 and 2 to disk at checkpoint
- At recovery:
  - Re-create 1 and 2 from the log
  - Re-create running txns and dirty pages in memory
  - Replay rest of the log (will see what this means)

Fuzzy Checkpointing: Data Structures

- Each log record has a **Log Sequence Number** (LSN)
  - A unique integer that's increasing (e.g., line number)

- Each data page has a **Page LSN**
  - The LSN of the most recent log record that updated that page.

Fuzzy Checkpointing: Data Structures

Dirty pages table

pageID	recLSN
P5	102
P6	103
P7	101

- Dirty Page Table
  - Lists all dirty pages
  - For each dirty page: recoveryLSN (recLSN) = first LSN that caused page to become dirty

Transactions

txnID	lastLSN	Status
T100	104	commit
T200	103	abort

- Transactions Table
  - Lists all txn's and their statuses
  - For each txn: lastLSN = its most recent update LSN (if active)

Fuzzy Checkpointing: Data Structures

Log (WAL)

LSN	prevLSN	txnID	pageID	Log Payload
101	-	T100	-	START
102	-	T200	-	START
103	102	T200	P6	<old val, new val>
104	101	T100	P5	<old val, new val>

- Write ahead log
  - Same as before, but we store both old and new values in update records
  - New field **prevLSN** = LSN of the previous log record written by this txnID
  - Actions of a transaction form a linked list backwards in time

Fuzzy Checkpointing Example

Dirty pages table (DPT)

pageID	recLSN
P5	102
P6	103
P7	101

Transactions

txnID	lastLSN	Status
T100	104	commit
T200	103	abort

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	START
102	-	T200	P5	START
103	102	T200	P6	6, 21
104	101	T100	P5	39, 100

Buffer Pool

P8	P2	. . .
P5 PageLSN=104	**P6** PageLSN=103	**P7** PageLSN=101
…	…	…

Fuzzy Checkpointing: Protocol

Dirty pages table (DPT)

pageID	recLSN
P5	102
P6	103
P7	101

Transactions

txnID	lastLSN	Status
T100	104	commit
T200	103	abort

- Write a <BEGIN CKPT> to log
- Flush log to disk
- Continue normal operation
- When DPT and Transactions tables are written to the disk, write <END CKPT> to log
- Flush log to disk

ARIES Normal Operation

What to do when a transaction:

- Starts
- Updates a page
- Commits
- Aborts

- What to do when buffer manager:
  - FETCH a page
  - FLUSH a page

Transactions

txnID	lastLSN	Status
T100	101	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5

Buffer Pool

P8	P2	. . .
P5 PageLSN=…	P6 PageLSN=…	P7 PageLSN=101

Dirty pages table

pageID	recLSN
P7	101

ARIES Normal Operation

Transactions

txnID	lastLSN	Status
T100	101	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5

Buffer Pool

P8	P2	. . .
P5 PageLSN=…	**P6** PageLSN=…	**P7** PageLSN=101

Dirty pages table

pageID	recLSN
P7	101

Transaction starts

- Write START record in **Log**
- Update Transactions table

ARIES Normal Operation

Transaction starts

- Write START record in **Log**
- Update Transactions table

Ex: T105 starts

- Write **<START,T105>** in **Log**
- Add T105 in Transactions and set lastLSN = null

Transactions

txnID	lastLSN	Status
T100	101	running
T105	-	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5
102	-	T105	-	START

Buffer Pool

P8	**P2**	. . .
P5 PageLSN=…	**P6** PageLSN=…	**P7** PageLSN=101

Dirty pages table

pageID	recLSN
P7	101

ARIES Normal Operation

Transaction updates

- Write update record in **Log**
- Update the following:
  - **prevLSN=lastLSN**
  - **pageLSN=LSN**
  - **lastLSN=LSN**
  - **recLSN**=if null then **LSN**

Transactions

txnID	lastLSN	Status
T100	101	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5

Buffer Pool

P8	**P2**	. . .
P5 PageLSN=…	**P6** PageLSN=…	**P7** PageLSN=101

Dirty pages table

pageID	recLSN
P7	101

ARIES Normal Operation

Transaction updates

- Write update record in **Log**
- Update the following:
  - **prevLSN=lastLSN**
  - **pageLSN=LSN**
  - **lastLSN=LSN**
  - **recLSN**=if null then **LSN**

Ex: T100 writes 10 in P7

- Write **<T100,P7,5,10>** in the **Log**
  - New LSN: 102
- Update other tables (see arrows)

Transactions

txnID	lastLSN	Status
T100	102	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5
102	101	T100	P7	5, 10

Buffer Pool

P8	P2	. . .
P5 PageLSN=…	P6 PageLSN=…	P7 PageLSN=102

Dirty pages table

pageID	recLSN
P7	101

ARIES Normal Operation

Page flushes

- Flush log up to (and incl.) **pageLSN**
- Remove page from **Dirty Pages Table** and **Buffer Pool**

Ex: Buffer manager wants to FLUSH(P7)

- Flush **Log** up to (and incl.) **101**
- Remove P7 from **Dirty Pages Table** and **Buffer Pool**

Transactions

txnID	lastLSN	Status
T100	101	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5

Buffer Pool

P8		. . .
P5 PageLSN=…	**P6** PageLSN=…	**P7** PageLSN=101

Dirty pages table

pageID	recLSN
P7	101

ARIES Normal Operation

Page flushes

- Flush log up to (and incl.) **pageLSN**
- Remove page from **Dirty Pages Table** and **Buffer Pool**

Ex: Buffer manager wants to FLUSH(P7)

- Flush **Log** up to (and incl.) **101**
- Remove P7 from **Dirty Pages Table** and **Buffer Pool**

Transactions

txnID	lastLSN	Status
T100	101	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5

Buffer Pool

P8		. . .
P5 PageLSN=…	**P6** PageLSN=…	

Dirty pages table

pageID	recLSN

ARIES Normal Operation

Page fetches

- Create entry in **Dirty Pages Table** and **Buffer Pool**

Ex: Buffer manager wants FETCH(P2)

- Create entry in **Dirty Pages** table set **recLSN** = NULL
- Bring page into **Buffer Pool**

Transactions

txnID	lastLSN	Status
T100	101	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	P7	1, 5

Buffer Pool

P8	**P2**	. . .
P5 PageLSN=…	**P6** PageLSN=…	

Dirty pages table

pageID	recLSN
P2	-

ARIES Normal Operation

Transaction commits

- Write commit record to Log
- Flush Log up to this entry
  - Txn is now considered committed!
- Update Transactions to commit
- (At a later time) flush dirty pages to disk
- Write end record to Log
- Update Transactions to complete

- Recall we are using WAL!

Transactions

txnID	lastLSN	Status
T100	102	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	-	START
102	-	T100	P7	1, 5

Buffer Pool

P8	**P2**	. . .
P5 PageLSN=…	**P6** PageLSN=…	**P7** PageLSN=101

Dirty pages table

pageID	recLSN
P7	101

ARIES Normal Operation

Transaction commits

Ex: T100 commits

- Write **<COMMIT,T100>**
- Flush Log up to this entry
- Update Transactions to commit
- (At a later time) flush dirty pages to disk
- Write **<END,T100>** in the Log
- Update Transactions to complete

Transactions

txnID	lastLSN	Status
T100	102	complete

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	-	START
102	-	T100	P7	1, 5
103	-	T100	-	COMMIT
104	-	T100	-	END

Buffer Pool

P8	P2	. . .

Dirty pages table

pageID	recLSN

ARIES Normal Operation

Transaction aborts

- Write abort record to Log
- (Optional) Flush Log up to this entry
  - Optional as all incomplete txns are considered as aborted anyway
- Find first action to undo from lastLSN
- Go to log and start undo changes
- Write compensation record (CLR) to Log
  - CLR's undoNextLSN points to next record to undo (part of the payload)
- Follow **prevLSN** to retrace more (if any) actions to undo
- Once done, change Transactions to abort
- (At a later time) flush dirty pages to disk
- Write end record to Log
- Update Transactions to complete

Transactions

txnID	lastLSN	Status
T100	103	running

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	-	START
102	101	T100	P7	1, 5
103	102	T100	P6	2, 6

Buffer Pool

P8	P2	. . .
P5 PageLSN=…	P6 PageLSN=103	P7 PageLSN=102

Dirty pages table

pageID	recLSN
P7	101

ARIES Normal Operation

Ex: T100 aborts

- Write **<ABORT,T100>** to Log
- From lastLSN, LSN 103 is first to undo
- Undo 103 and write <CLR, 102, P6, 2>
  - 102 is the next record to undo
- Follow **prevLSN**, LSN 102 is next to undo
- Undo 102 and write <CLR, -, P7, 1>
  - No more actions to undo!
- Change Transactions to abort
- (At a later time) flush dirty pages to disk
- Write **<END,T100>** to Log
- Update Transactions to complete

Transactions

txnID	lastLSN	Status
T100	107	complete

Log (WAL)

LSN	prevLSN	txnID	pageID	Payload
101	-	T100	-	START
102	101	T100	P7	1, 5
103	102	T100	P6	2, 6
104	103	T100	-	ABORT
105	104	T100	P6	CLR, 2, 102
106	105	T100	P7	CLR, 1, -
107	106	T100	-	END

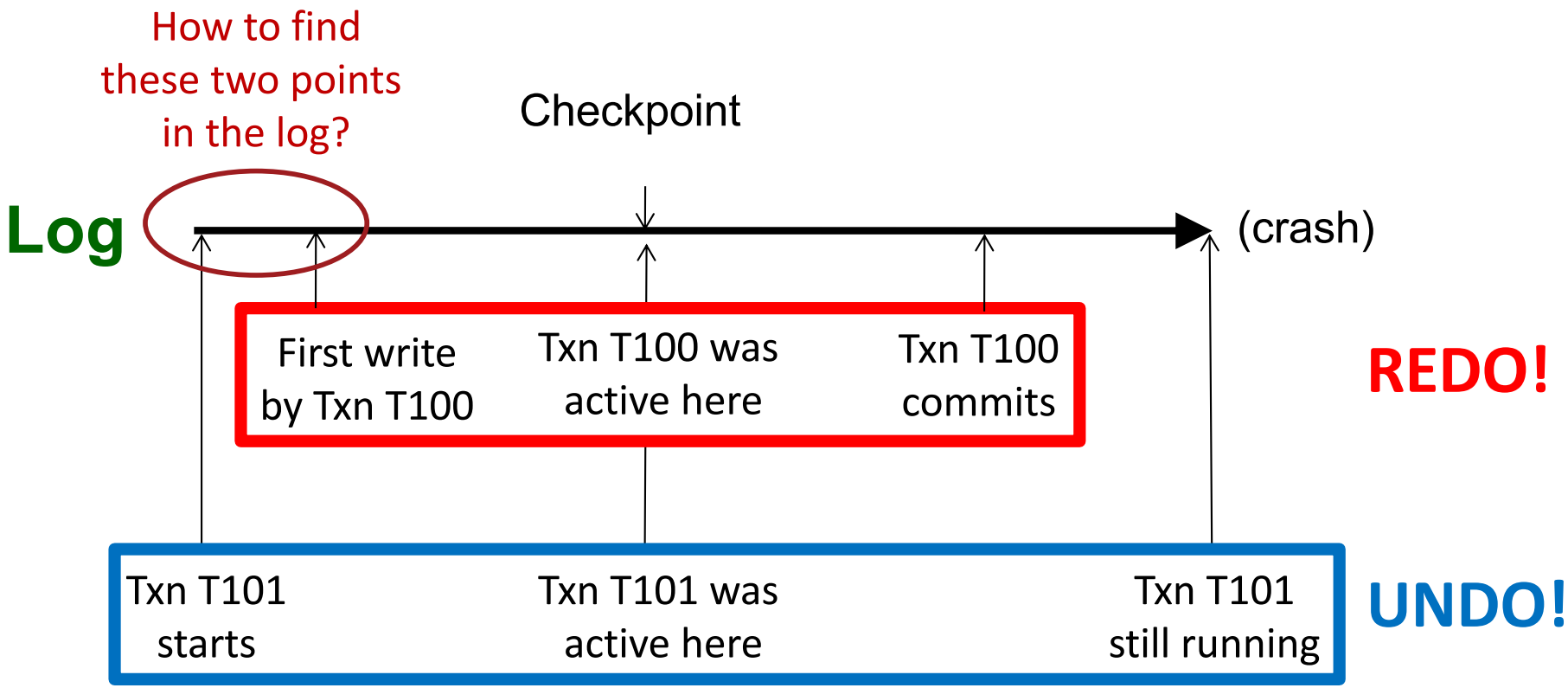
Will see a better algorithm in a few slides!

ARIES Recovery

- Start recovery from the last checkpoint
  - Easy to recover with non-fuzzy checkpoints. Just roll forward!
- With fuzzy checkpoints, we now need to handle:
  - Active transactions when checkpoint was taken
  - Dirty pages that were not flushed to disk yet

- Main principles:
  - Redo all actions before crash and bring DBMS to the exact state right when it crashed
  - Unroll changes from incomplete txns when crash occurred
  - Log all undo changes to ensure changes are not undone

ARIES Recovery



ARIES Recovery

Recovery from a system crash is done in 3 passes:

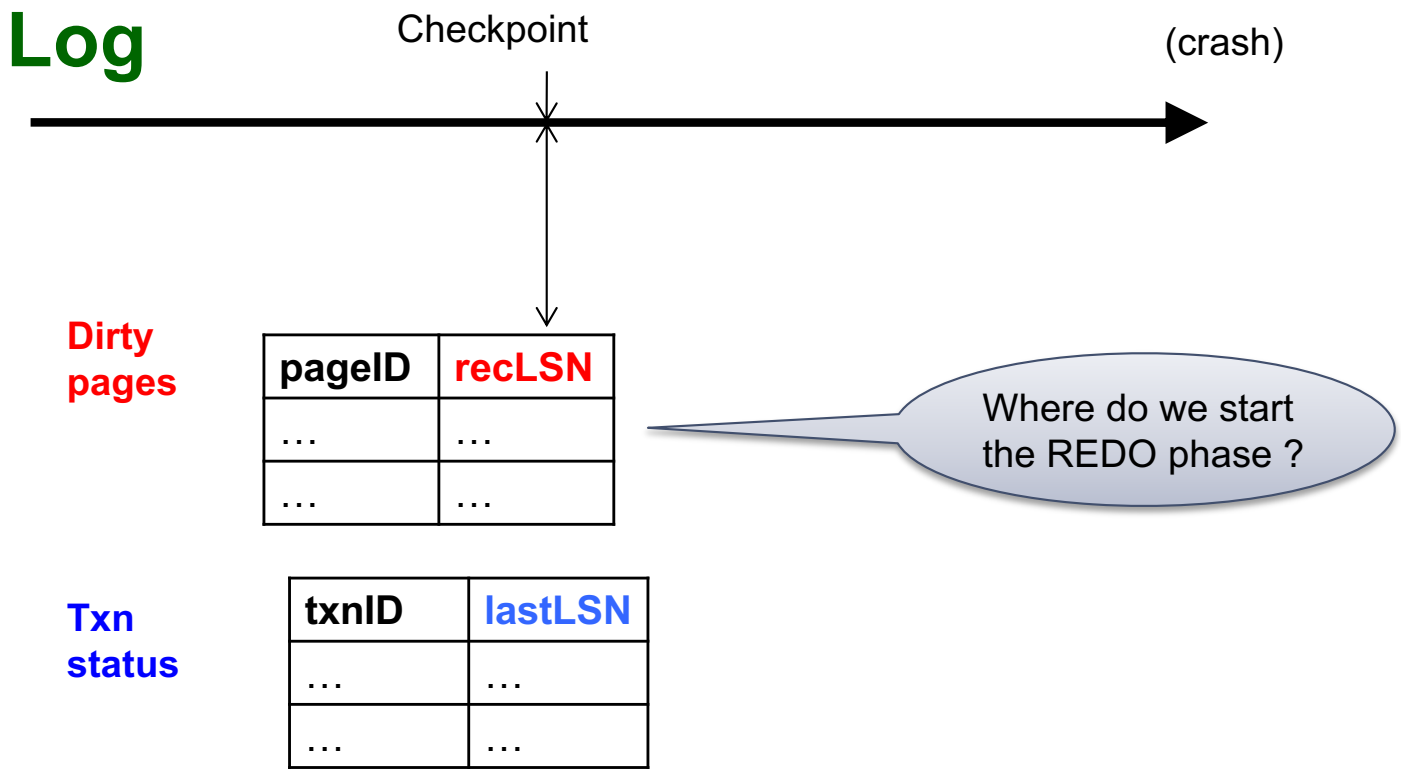
1. **Analysis pass**
   - Recreate list of dirty pages and active transactions
2. **Redo pass**
   - Redo all operations, even for those that were incomplete before crash
   - Goal is to replay DB to the state at the moment of the crash
3. **Undo pass**
   - Unroll effects of all incomplete transactions at time of crash
   - Log changes during undo in case of another crash during undo

1. Analysis Phase

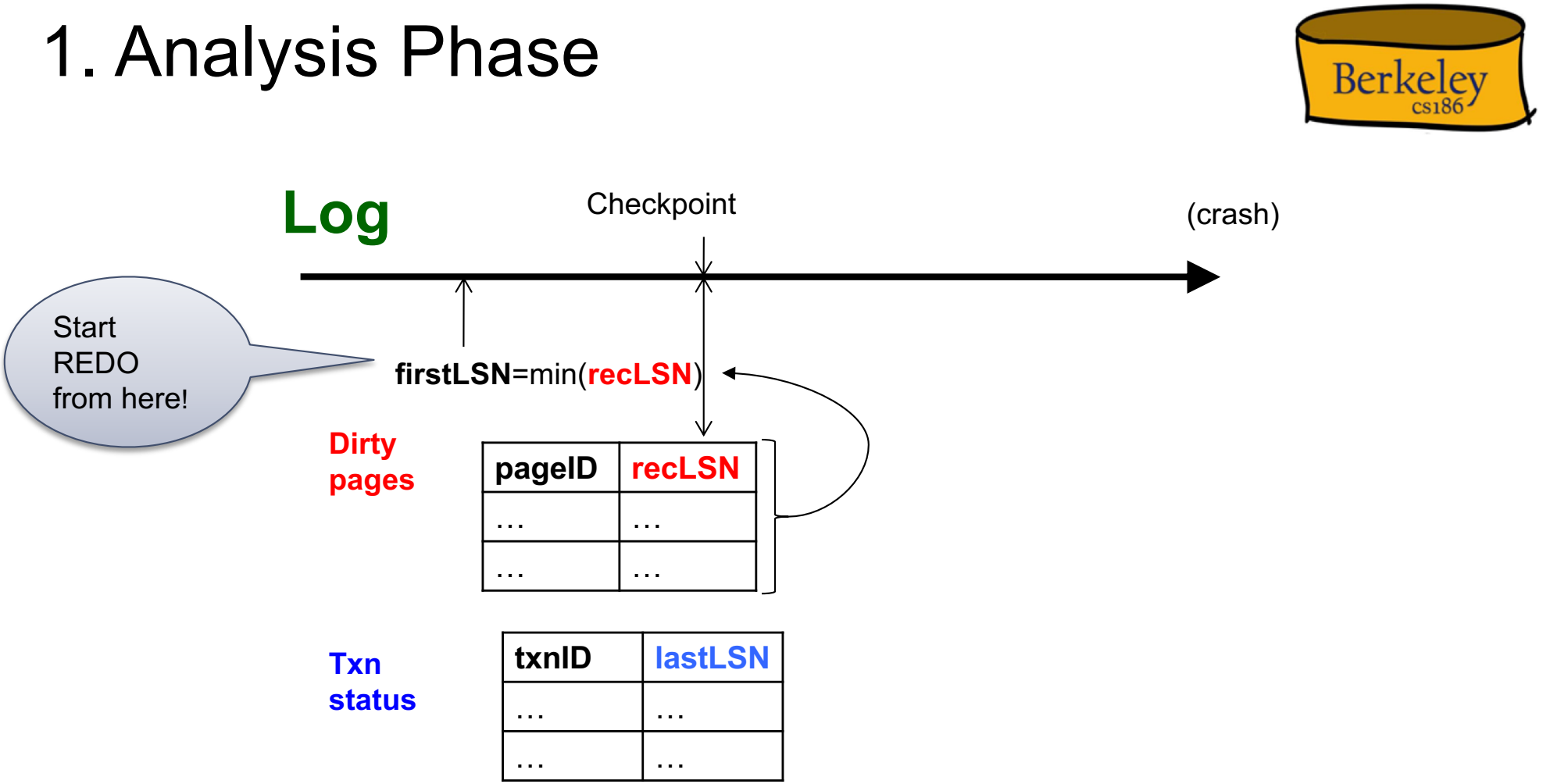
- Goal
  - Determine point in log (**firstLSN**) where to start REDO
  - Determine set of dirty pages when crashed
  - Identify active transactions when crashed

- Approach
  - Rebuild transactions table and dirty pages table
  - Recover these from the last checkpoint in the log
  - Compute: **firstLSN** = smallest of all pages' recoveryLSN
    - This is the earliest point that a write was made to buffer pool that hasn't persisted yet

1. Analysis Phase



1. Analysis Phase



2. Redo Phase

Main principle: replay history

- Process Log forward, starting from **firstLSN**
- Read every log record sequentially
- Redo actions are not recorded in the log

2. Redo Phase: Details

For each Log entry record LSN: **<T,P,old,new>**

- Write new value to page P
- Only redo those that need to be redone
  - How to determine that?

2. Redo Phase: Details

For each Log entry record LSN: **<T,P,old,new>**

- If P is not in Dirty Page then **don't redo.** How did this happen?
  - P was flushed to DB, removed from DPT before checkpoint
  - *Then* DPT flushed at checkpoint

- P is in DPT, but recLSN > LSN, then **don't redo.** How did this happen?
  - P was flushed to DB, removed from DPT before checkpoint
  - *Then* P was read in *again* and reinserted in DPT with larger recLSN

- P's **pageLSN** on disk > LSN, then **don't redo.** How did this happen?
  - P was updated again and flushed to DB after this log record

- Otherwise redo!

2. Redo Phase: Details

What happens if system crashes during REDO ?

We REDO again! Each REDO operation is *idempotent*: doing it twice is the as doing it once.

3. Undo Phase

- A simple solution:
  - All active txns in the Transactions Table are “losers.”
  - Just abort each loser transaction
  - Problem?
    - Lots of random I/O in the log following the chain of prevLSNs (see earlier slide)
    - Can we do this in one backwards pass of log?

3. Undo Phase

- Define **ToUndo** = set of lastLSN from the loser txns
  - Get them from the transactions table

Transactions

txnID	lastLSN	Status
T100	103	complete
T101	**104**	**running**

3. Undo Phase: Details

While **ToUndo** not empty:

- Choose most recent (largest) LSN in **ToUndo**
- If LSN is a regular log record **<T,P,old,new,prevLSN>**:
  - Undo action
  - Write a CLR where CLR.undoNextLSN = LSN.prevLSN
  - If prevLSN is not null then insert prevLSN into **ToUndo**
    - otherwise, write end record in log (we have fully aborted the txn)
- If LSN is a CLR record:
  - Don't undo!
  - But if CLR.undoNextLSN not null, insert in **ToUndo**
  - otherwise, write end record in log (we have fully aborted the txn)

We're done when there are no more transactions to undo

We can use this algorithm to undo a single txn as well during normal operation

3. Undo Phase: Details

What happens if system crashes during UNDO ?

We do not UNDO again! Instead, each CLR is a REDO record: we simply redo the undo

Additional Crash FAQs to Understand

Q: What happens if system crashes during Analysis?
A: Nothing serious. RAM state lost, need to start over next time.

Q: What happens if the system crashes during REDO?
A: Nothing bad. Some REDOs done, and we'll detect that next time.

Q: How do you limit the amount of work in REDO?
A: Flush asynchronously in the background. Even"hot" pages!

Q: What happens if the system crashes during UNDO?
A: Nothing bad. We'll start with redo again.

Q: How do you limit the amount of work in UNDO?
A: Avoid long-running Xacts.

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**: 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 (running, aborting) after Redo.
- Upon Undo, write CLRs.
- Redo “repeats history”: Simplifies the logic!