Lecture 12

10/23/2024

ARIES!!!!

The Ram, Salvador Dalí, 1928

Recovery Recap

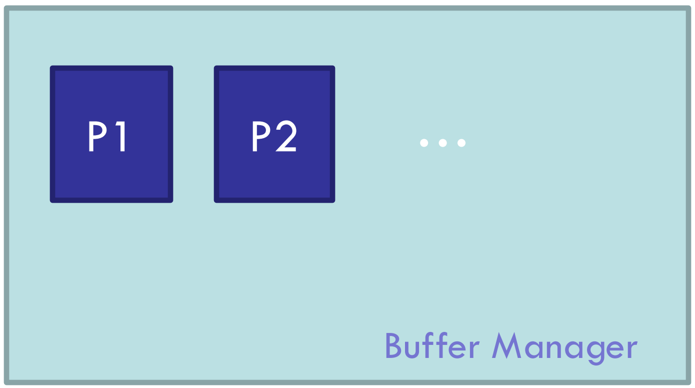
- What happens during crash:
  - Memory is reset
  - State on disk persists

- After a crash, recovery ensures:
  - **Atomicity:** partially finished xactions are rolled back
  - **Durability**: committed xactions are on stable storage (disk)

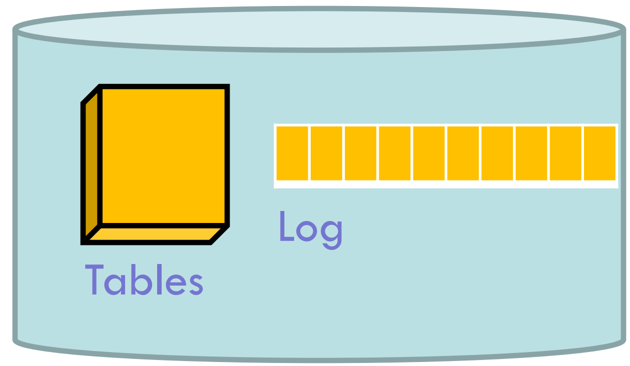
- Brings database into a transaction consistent state, where committed transactions are fully reflected, and uncommitted transactions are completely undone

Database State

Log records start and end of transactions, and contents of writes done to tables so we can solve both problems



Memory



Disk

After crash, memory is gone!

Problem 1: Some transactions may have written their uncommitted state to tables – need to **UNDO**

Problem 2: Some transactions may not have flushed all of their state to tables prior to commit – need to **REDO**

Types of Log Records

- **Start (SOT)** Log Sequence Number (LSN), Transaction ID (TID)
  - LSN is a monotonically increasing log record number
- **End (EOT)** LSN, TID, outcome (commit or abort)
- **UNDO** LSN, TID, before image
- **REDO** LSN, TID, after image

For ARIES:

- **CHECKPOINT** LSN, TID, state to limit how much is logged
- **CLR**(Compensation Log Records) LSN, TID, allows us to restart recovery

Write Ahead Logging

Write what we plan to do, before we do it

Recovery with NO FORCE / STEAL

- After crash, we must:
  - REDO "winner" transactions that had committed
  - UNDO "loser" transactions that had not committed

- Winners are transactions with SOT and COMMIT in log
- Losers are those with SOT and no EOT, or ABORT

- Need to REDO winners from start to end
- Need to UNDO losers in reverse, from end to start
- Also need to UNDO aborted transactions

Hard Problems with Recovery

- B-Tree:
  - Problem 1: Logical Inserts create different Btrees
  - Problem 2: Crash while updating a multi-page B-Tree or inconsistency between B-Tree and data pages
- Cost of checkpoints (do we have to block the system while checkpointing?)
- Recovery time (how long do we have to wait until the system is again available)
- Crash during recovery
- Escrow updates
- ….

ARIES

- Gold standard in logging
  - Specifies **all** the details
- NO FORCE/STEAL
- Recoverable recovery
- “Physiological” Logging
- Low-overhead Checkpoints (will explain)
- Support for escrow operations (will explain)
  - E.g., increment / decrements

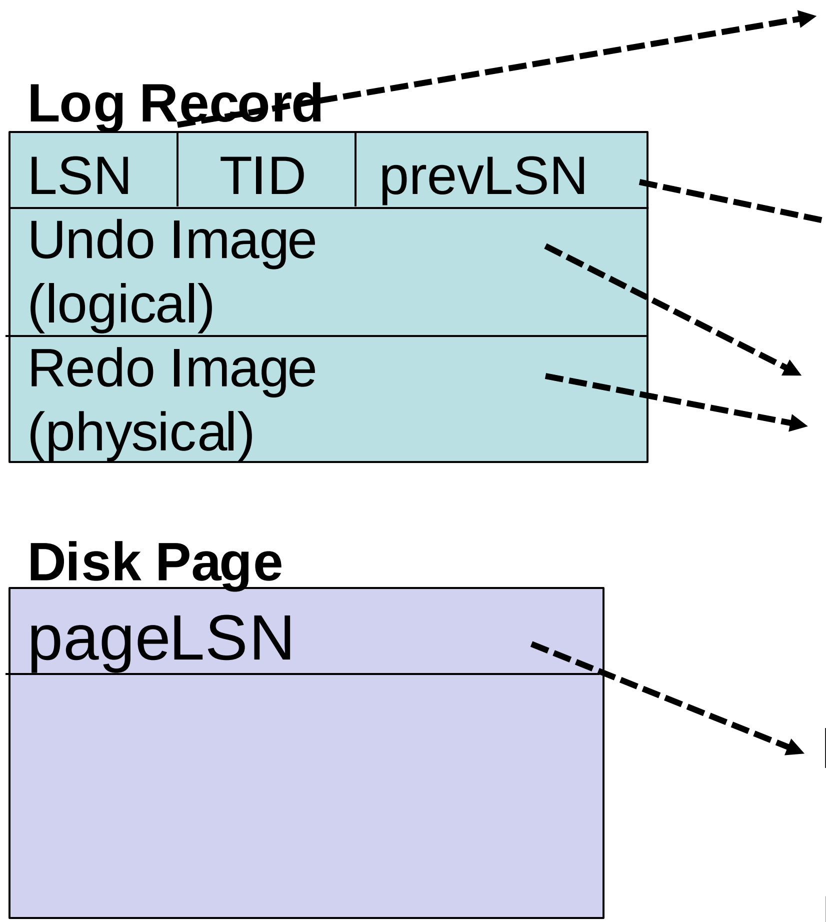
It might be hard to appreciate how “cool” ARIES is

ARIES Approach: 3 Log Passes

FW = Forward Pass; BW = Backward Pass

- **Analysis**, to see what needs to be done (FW)
- **Redo**, to ensure DB reflects updates that are in the log but not in tables (FW)
  - Including those that belong to txns that will eventually be rolled back!
  - Why? Ensures "action consistent" state -- which will allow logical undo.
  - "Repeating History"
- **Undo**, to rollback losers (BW)

Log Record Format



Every log record has an LSN associated with it.

The previous LSN written by this transaction

Update records have both UNDO and REDO information.

Every time a page is written, the latest LSN associated with that log record is included as the pageLSN.

Innovation Check

Log Rec

LSN	T
Undo Ima (logical)	
Redo Ima (physical)	

Disk Pag

pageLS

N

by this

JNDO

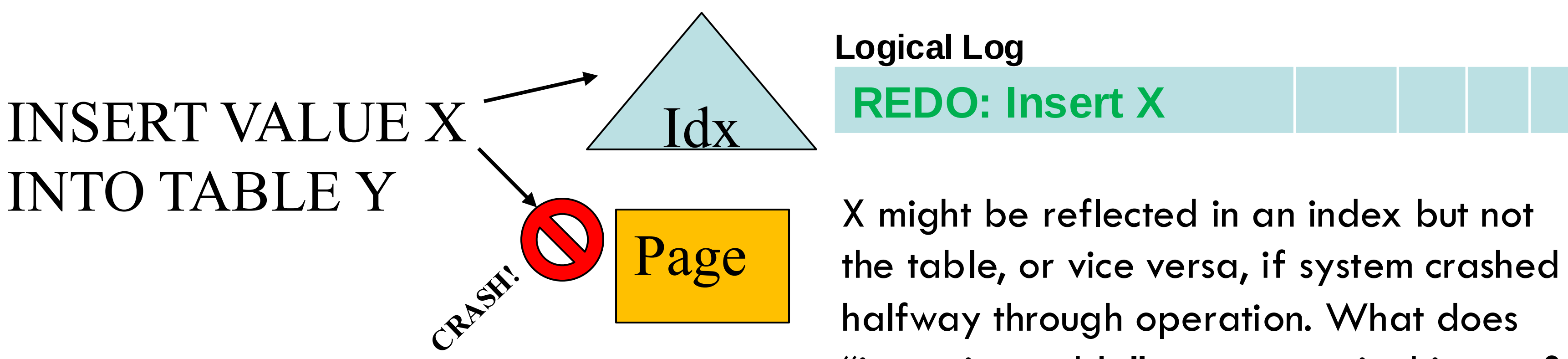
, the
at log
geLSN.

“Physiological” Logging

- REDO is Physical
- UNDO is Logical

REDO must be physical

- At time of crash, database may not be in an "action consistent" state
- Some ops can encompass multiple non-atomic physical operations



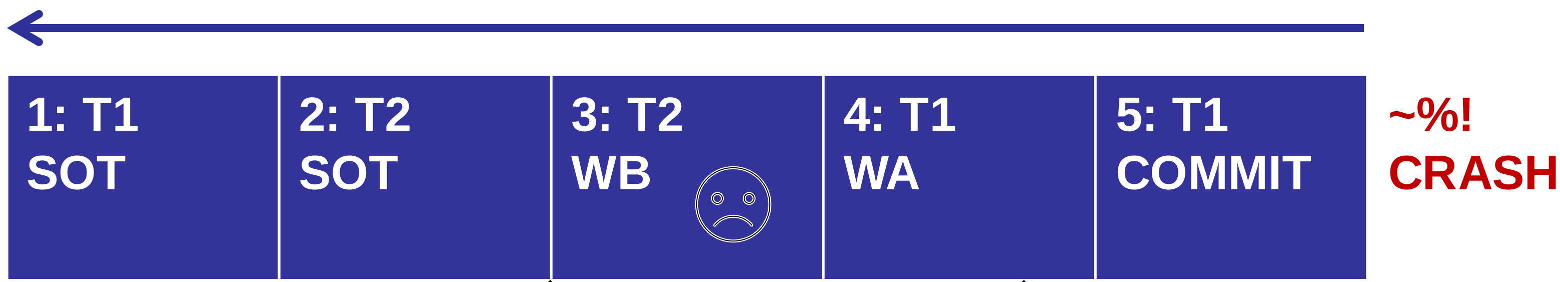
X might be reflected in an index but not the table, or vice versa, if system crashed halfway through operation. What does "insert into table" even mean in this case?

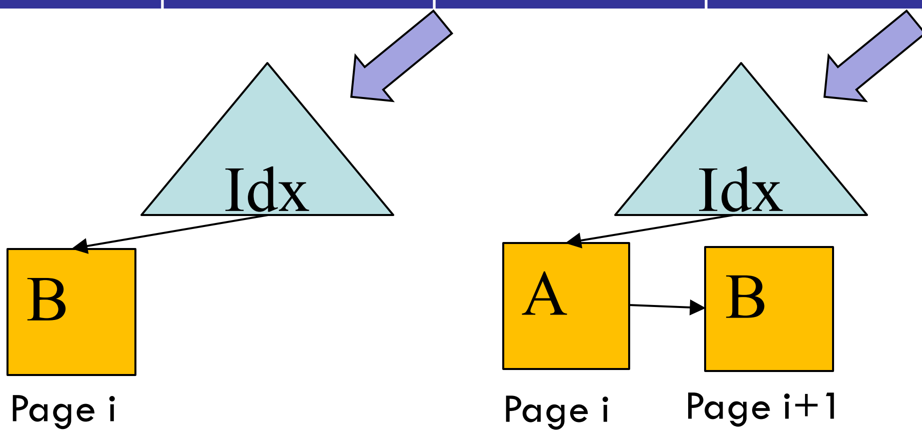
- Much easier to replay with physical logging

UNDO must be logical

- We only UNDO some actions
- Implies state when UNDOing may not be same as when the log was written
- Physical logging (e.g., of the specific before and after images in the page) would fail

UNDO Example





At time LSN 3 is written

At time LSN 4 is written

← This will also be the state when LSN 3 is UNDOne!

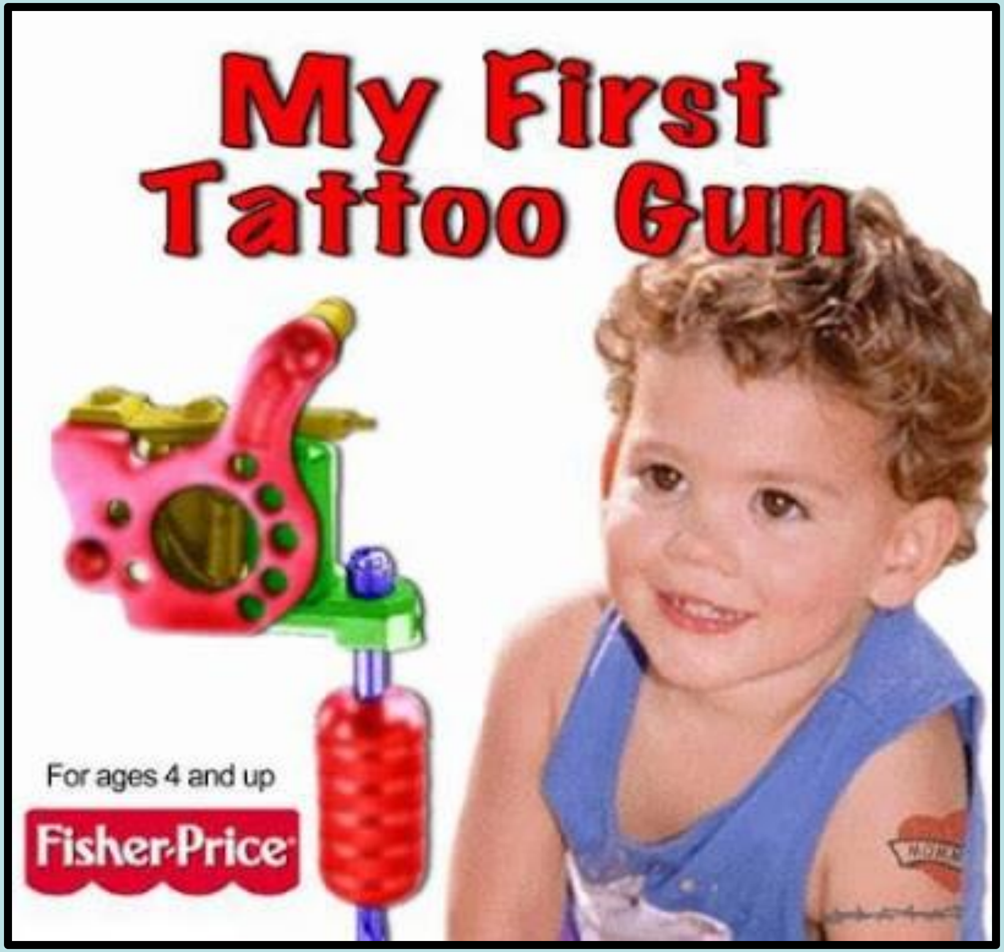
Different from state when record was written

Demands logical undo (B is on a different page!)

Not needed in REDO, because we "repeat history" and replay everything

- Physical modifications made to the database since last time will still be correct.

Innovation Check



1: T1 SOT

SH

the state
NDOne!

B

Pag

At t

when record

do (B is on a

Not needed

- Physical r

ct.

ARIES Normal Operation

- Two key data structures:
  - *Transaction table* -- list of active transactions
  - *Dirty page table* -- List of pages that have been modified and not yet written to disk

- Data structures updates as system runs:
  - Pages asynchronously flushed to disk
    - Log forced before flush (but not before write)
    - Flushes are not logged
  - Log forced before COMMIT ack'd

Transaction Table

xactionTable

lastLSN	TID
13	3

- All active transactions in table
- **lastLSN**: most recent log record written by that transaction

Dirty Page Table

dirtyPgTable

pgNo	recLSN
D	8
B	10
A	11
E	13

Recall, dirty pages are periodically flushed to disk by a background process.

On flush, remove from dirtyPageTable

- One entry for each page that has been modified but not flushed to disk
- <u>recLSN</u>: log record that **first** dirtied the page

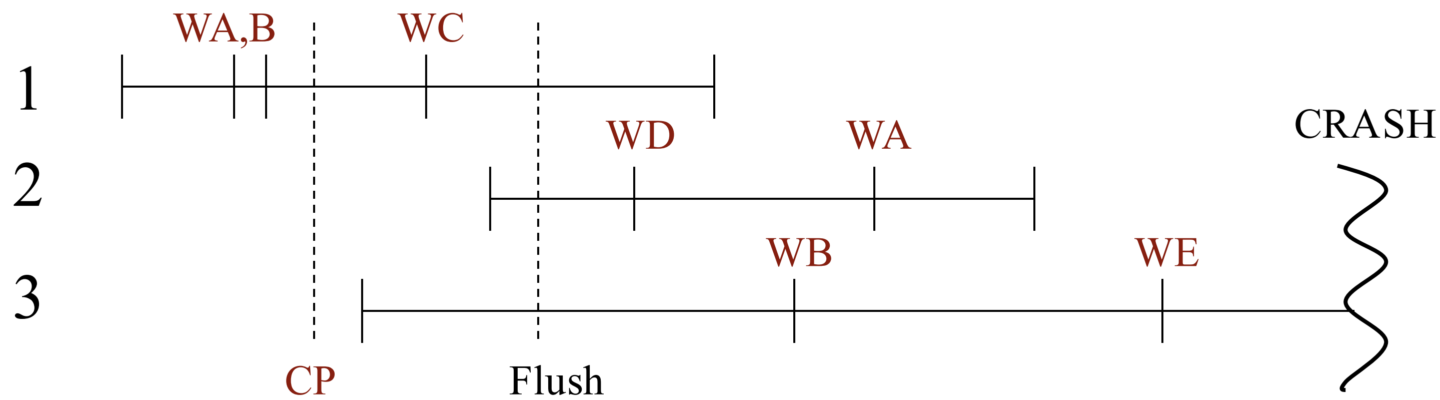
Checkpoints

- Taken periodically
- Log record that contains:
  1. the state of the dirty page table and
  2. the transaction table

  Checkpoints don't require pages themselves to be flushed to disk

- Allow us to limit amount of log we have to keep and replay during crash

ARIES Example



LSN	Type	Tid	PrevLSN	Data (Page)
1	SOT	1		
2	UP	1	1	A
3	UP	1	2	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Flush

ARIES Data Structures

xactionTable

lastLSN	TID
1	1

dirtyPgTable

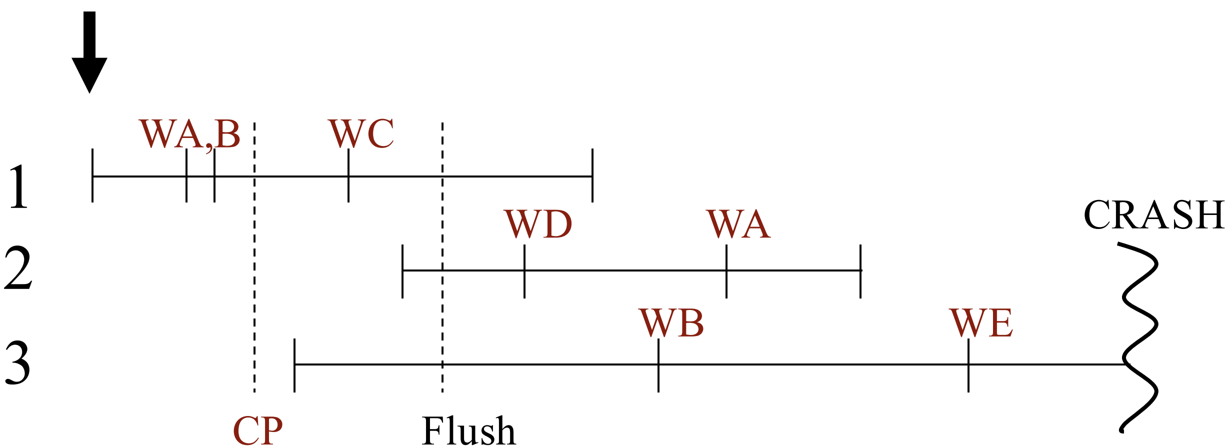
pgNo	recLSN

Checkpoint

xactionTable	
dirtyPgTable	

Disk

Page	pageLSN
A	?
B	?
C	?
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
2	1

dirtyPgTable

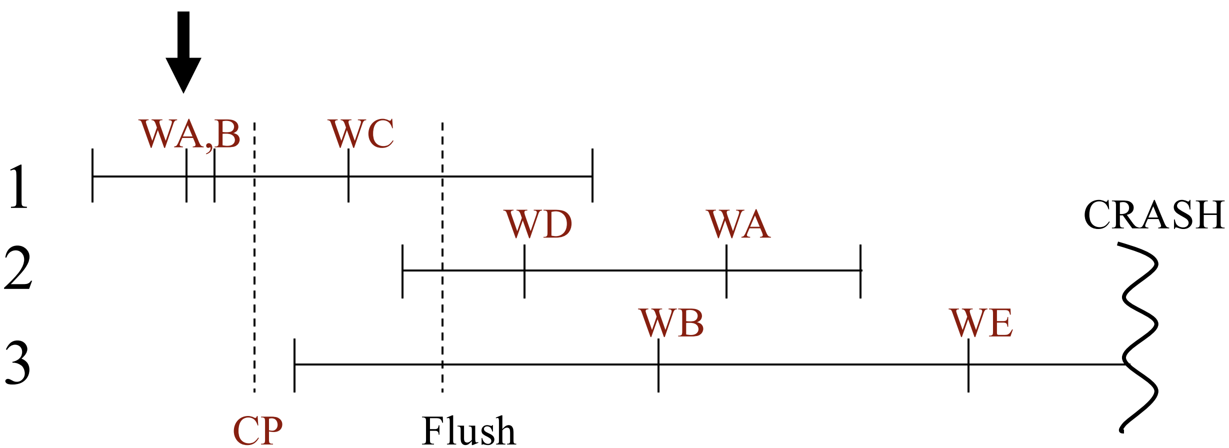
pgNo	recLSN
A	2

Checkpoint

xactionTable	
dirtyPgTable	

Disk

Page	pageLSN
A	?
B	?
C	?
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
3	1

dirtyPgTable

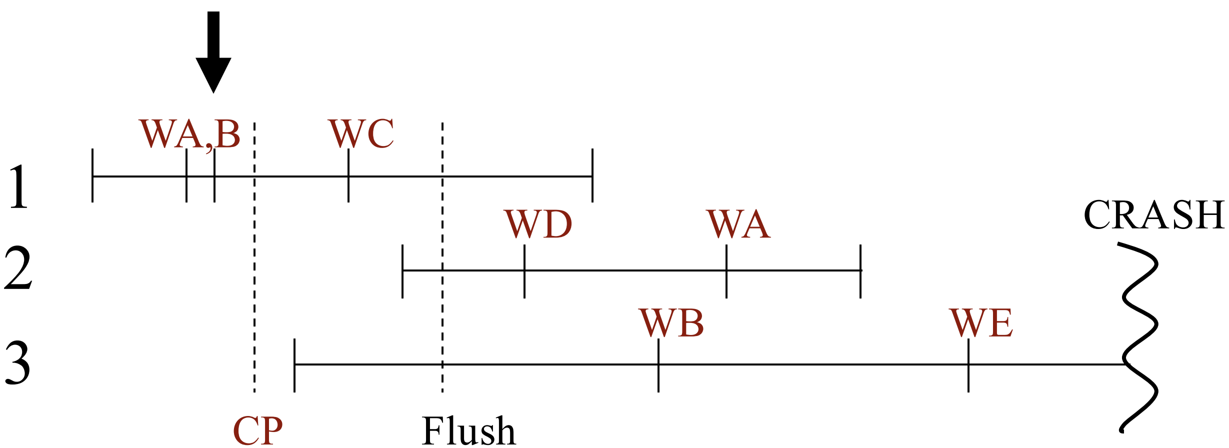
pgNo	recLSN
A	2
B	3

Checkpoint

xactionTable	
dirtyPgTable	

Disk

Page	pageLSN
A	?
B	?
C	?
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
3	1

dirtyPgTable

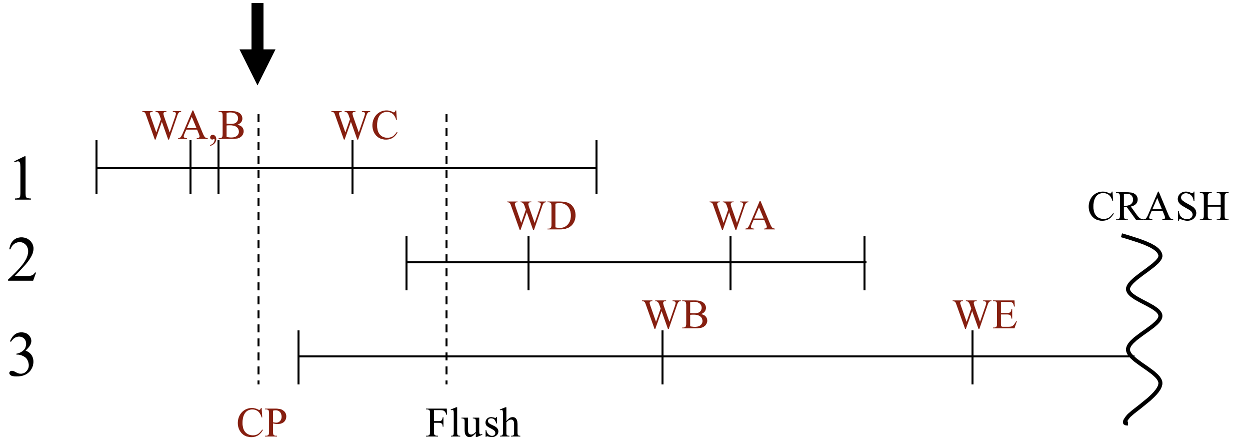
pgNo	recLSN
A	2
B	3

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	?
B	?
C	?
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
3	1
5	3

dirtyPgTable

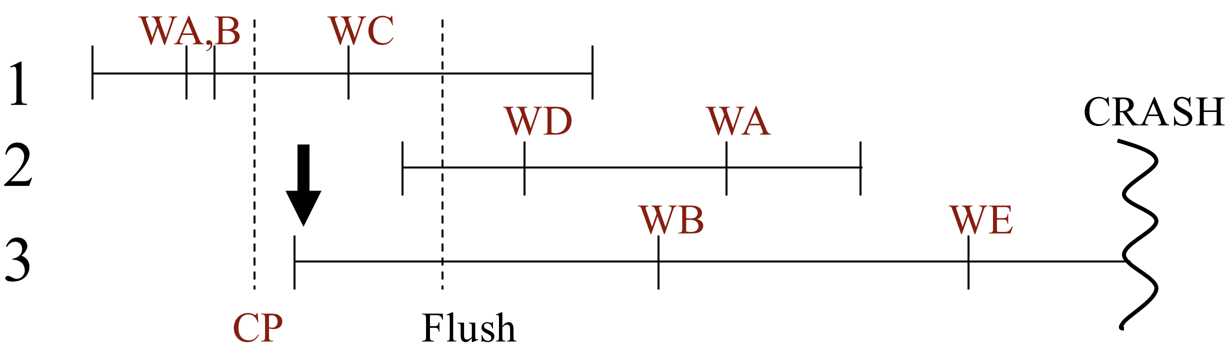
pgNo	recLSN
A	2
B	3

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	?
B	?
C	?
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
6	1
5	3

dirtyPgTable

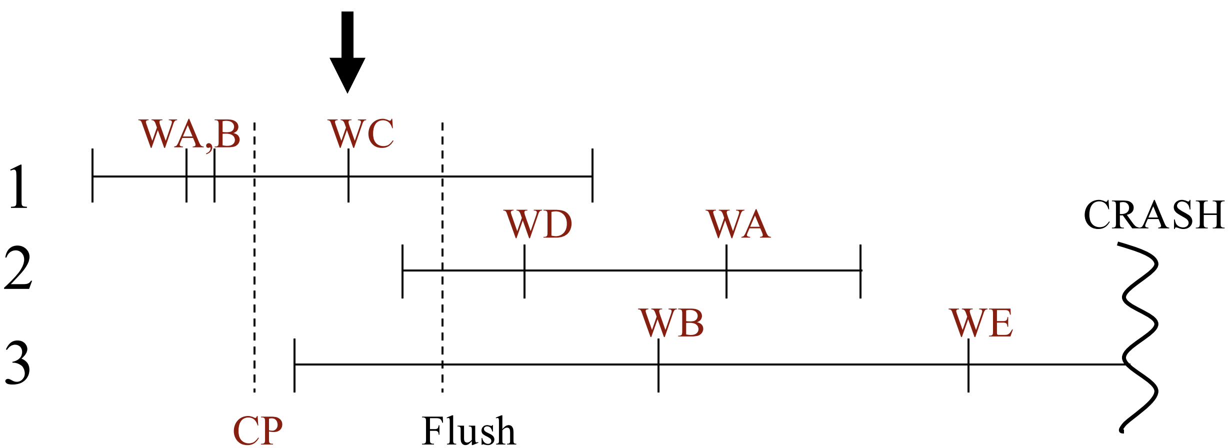
pgNo	recLSN
A	2
B	3
C	6

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	?
B	?
C	?
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
6	1
5	3
7	2

dirtyPgTable

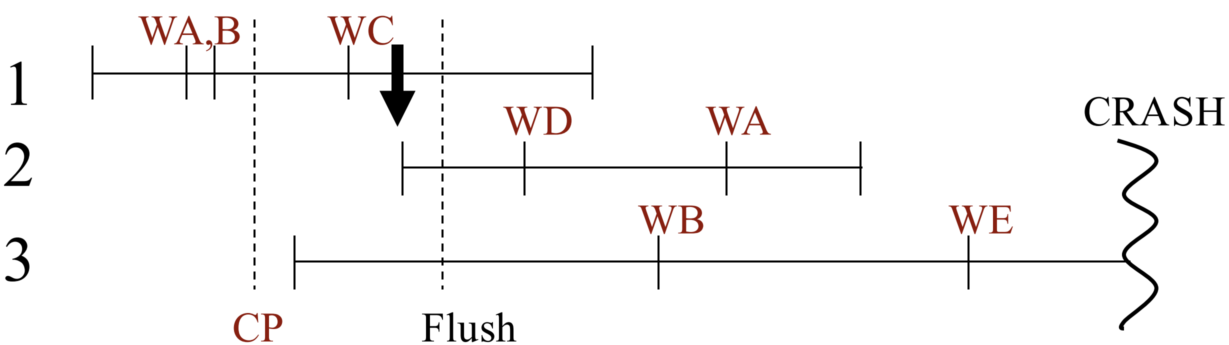
pgNo	recLSN
A	2
B	3
C	6

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	?
B	?
C	?
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
6	1
5	3
7	2

dirtyPgTable

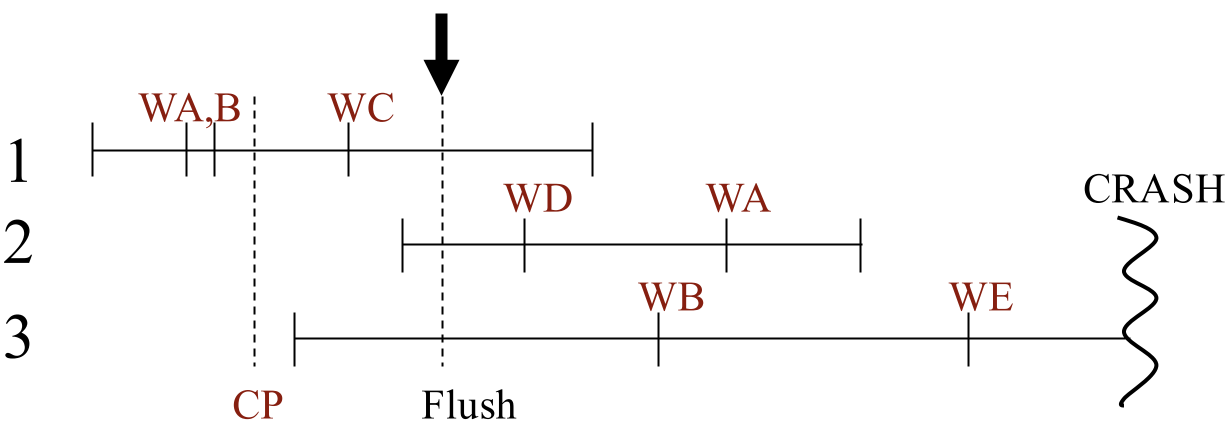
pgNo	recLSN

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
6	1
5	3
8	2

dirtyPgTable

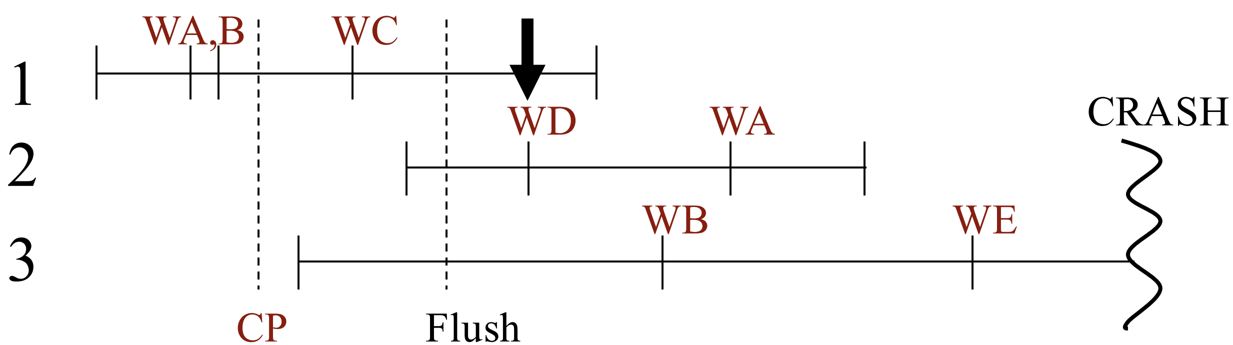
pgNo	recLSN
D	8

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
~~6~~	~~1~~
5	3
8	2

dirtyPgTable

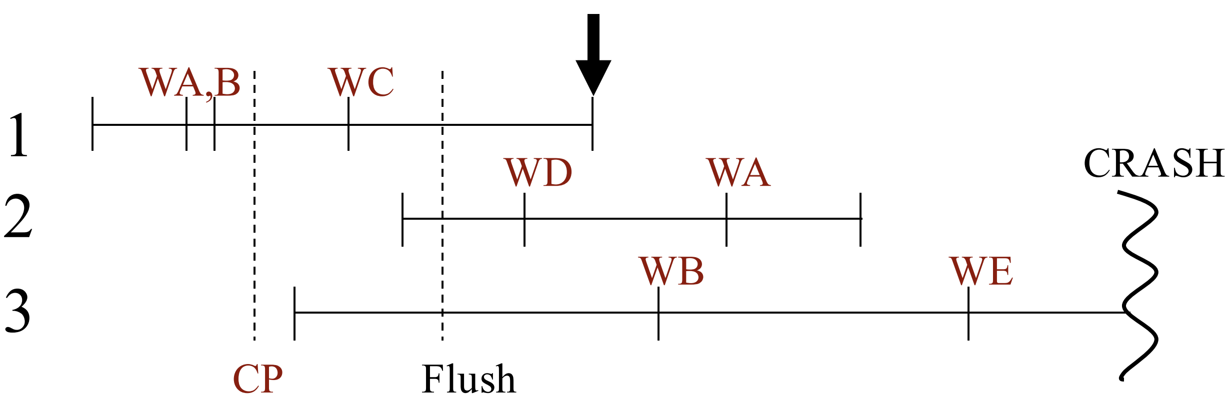
pgNo	recLSN
D	8

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
5	3
8	2

dirtyPgTable

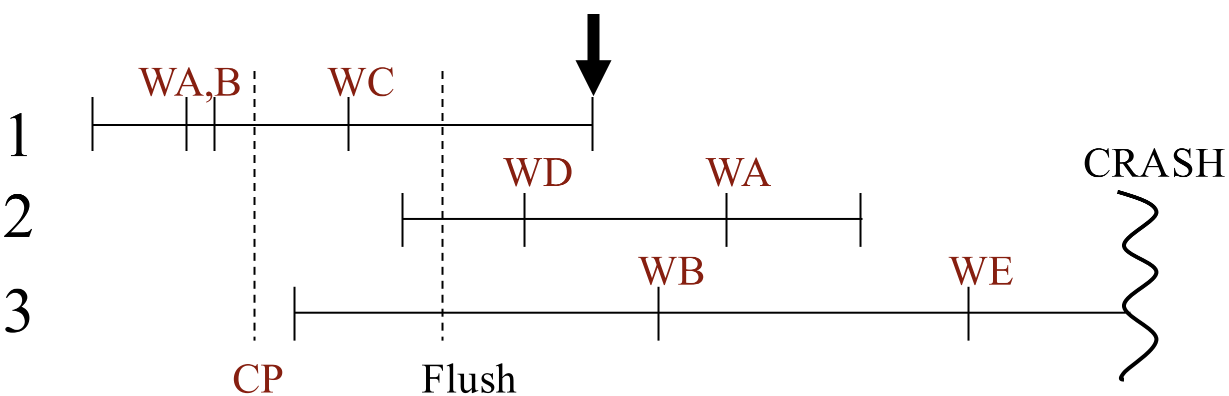
pgNo	recLSN
D	8

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
10	3
8	2

dirtyPgTable

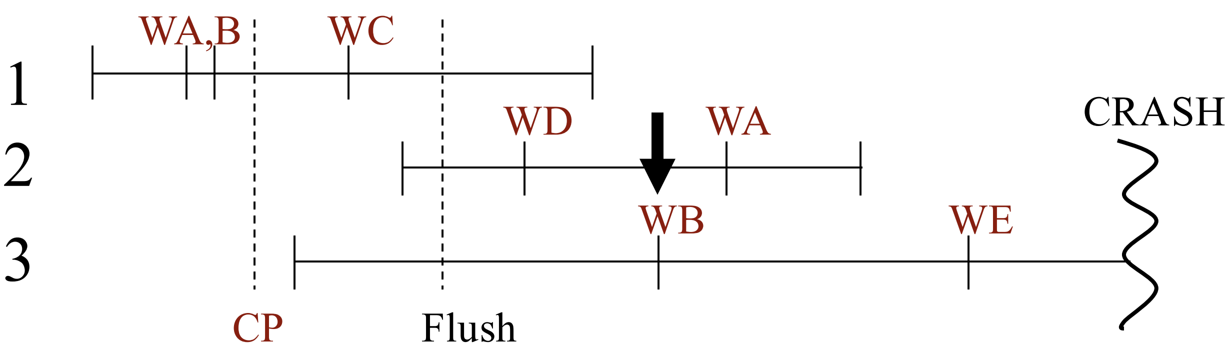
pgNo	recLSN
D	8
B	10

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
10	3
11	2

dirtyPgTable

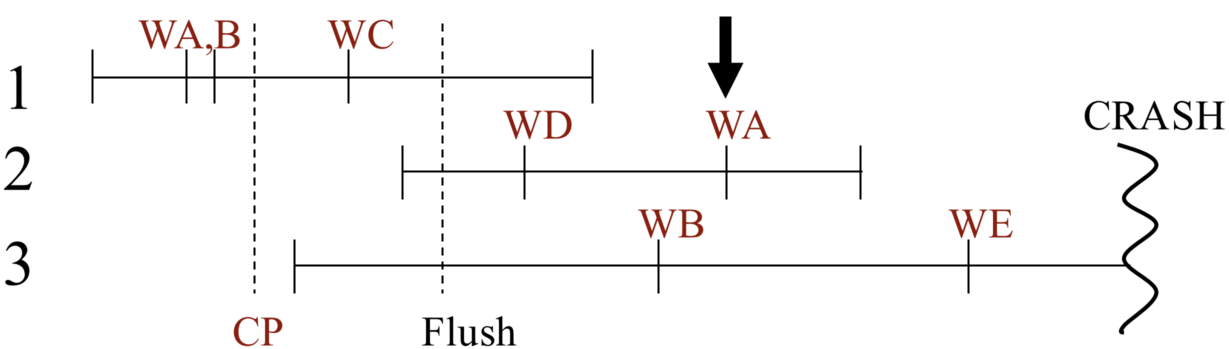
pgNo	recLSN
D	8
B	10
A	11

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
10	3
~~11~~	~~2~~

dirtyPgTable

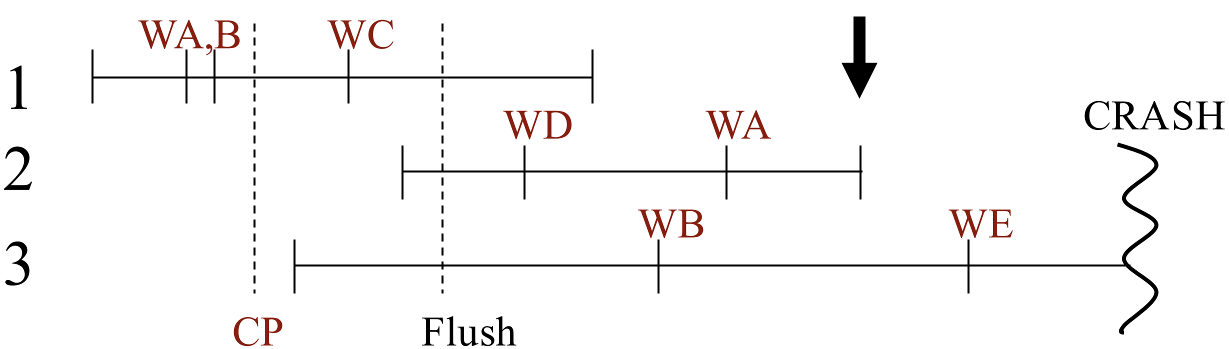
pgNo	recLSN
D	8
B	10
A	11

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
10	3

dirtyPgTable

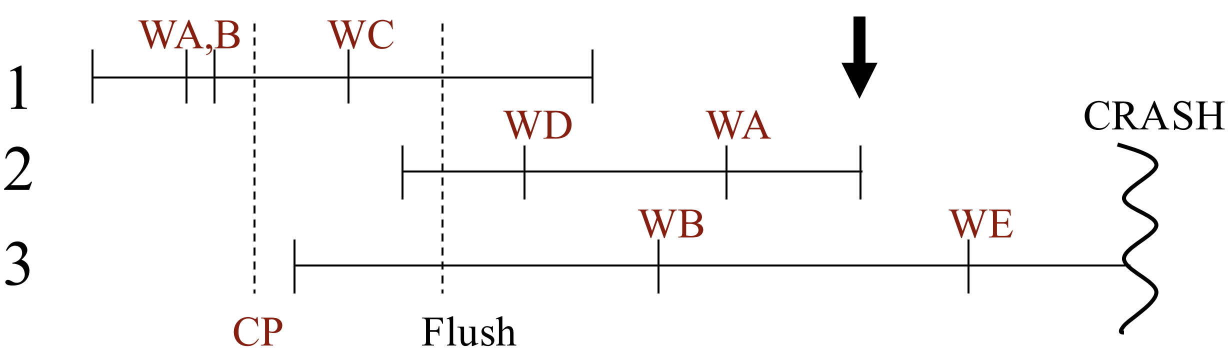
pgNo	recLSN
D	8
B	10
A	11

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



ARIES Data Structures

xactionTable

lastLSN	TID
13	3

dirtyPgTable

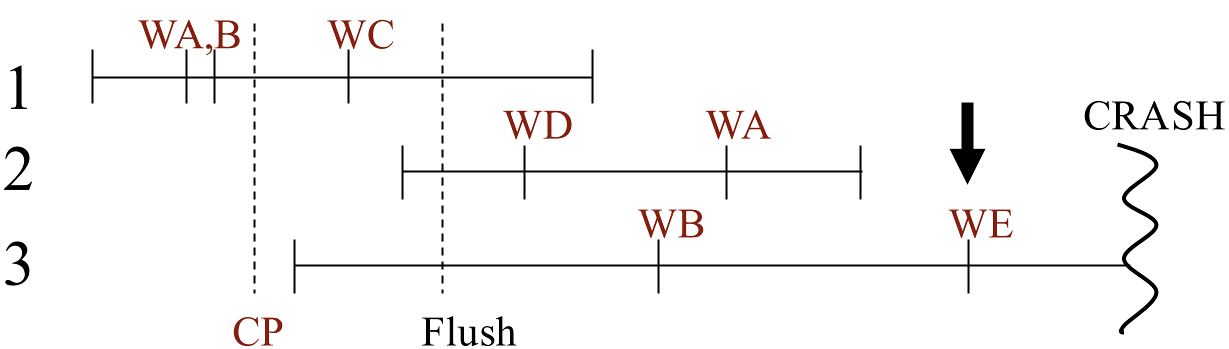
pgNo	recLSN
D	8
B	10
A	11
E	13

Checkpoint

xactionTable	3-1
dirtyPgTable	A-2, B-3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?



Innovation Check

1
2
3

WB WE

CP Flush

Crash Recovery

- 3 Phases
  - Analysis
    - Rebuild data structures
    - Determine winners & losers
  - Redo
    - “Repeat history”
    - Why?
  - Undo
    - Undo Losers

Analysis Pass

- Goal: reconstruct the state of the transaction table and the dirty page table at the time the crash occurred.
- Play log forward
  - Add and remove xactions to/from the transaction table on SOT and COMMIT/ABORT
  - Update the lastLSN on writes
  - Update the dirty page table as writes happen

State After Analysis

- After analysis, what can we say about dirty page table and transaction table?
- Txn table tells us what to UNDO
- Dirty pages is a conservative list of pages that need to be REDOne

- Why is it “conservative”?
  - Because we don’t actually know what is on disk; some pages may already have updates applied

Where to Begin Analysis

- Beginning of log?
  - Ok, but may require us to scan a lot of log

- Last checkpoint!
- How do we find it?
  - Keep a pointer to the checkpoint at a well-known place on disk

Analysis

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Analysis

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
3	1

dirtyPgTable

pgNo	recLSN
A	2
B	3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Analysis

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
3	1
5	3

dirtyPgTable

pgNo	recLSN
A	2
B	3

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Analysis

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
6	1
5	3

dirtyPgTable

pgNo	recLSN
A	2
B	3
C	6

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Analysis

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
6	1
5	3
7	2

dirtyPgTable

pgNo	recLSN
A	2
B	3
C	6

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Analysis

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
6	1
5	3
8	2

dirtyPgTable

pgNo	recLSN
A	2
B	3
C	6
D	8

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Analysis

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3		
11	UP	2		
12	EOT	2		
13	UP	3		

Dirty page table doesn’t reflect true state on disk.
Conservative: *at least* all previous LSNs are on disk

Maybe some pages were flushed, then there was an immediate CRASH before the dirtyPgTable was saved to the log in a CP

xactionTable

lastLSN	TID
13	3

Losers

dirtyPg

pgNo	
A	
B	
C	6
D	8
E	13

B	
C	6
D	?
E	?

Redo

- Where to begin?
  - Checkpoint?
  - Min(recLSN)! – earliest unflushed update
- What to REDO
  - Everything?
    - Slow
    - Problematic if using logical (escrow) logging
  - Redo an update UNLESS:
    - Page is not in dirtyPgTable
      - Page flushed prior to checkpoint, didn't redirty
    - LSN < recLSN
      - Page flushed & redirtied prior to checkpoint (that is, the LSN is already reflected on disk)
    - LSN <= pageLSN
      - Page flushed after checkpoint

Only step that requires going to disk

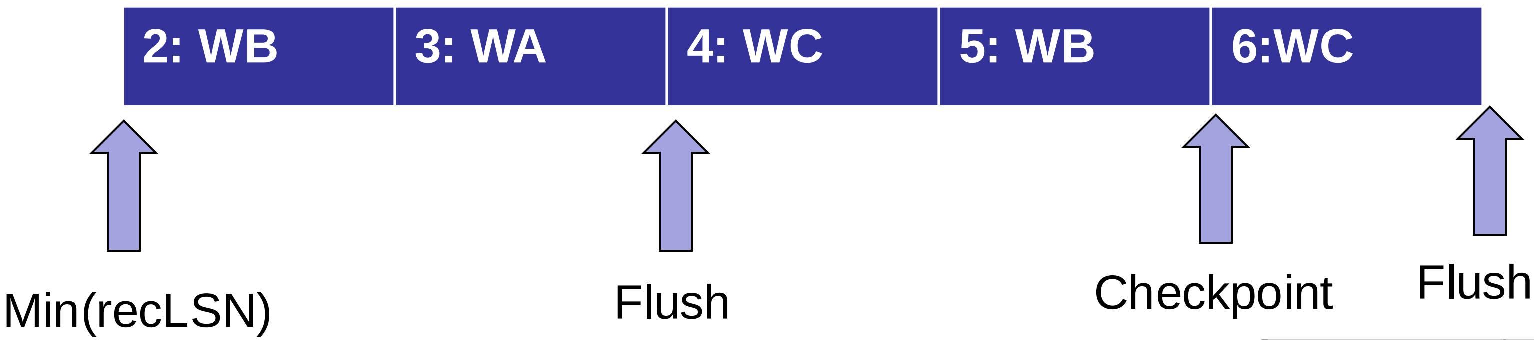
dirtyPgTable

pgNo	recLSN
A	$\boxed{2}$
B	3
C	6
D	8
E	13

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

REDO Conditions Example



dirtyPgTable @ CP

pgNo	recLSN
B	5
C	4

Redo an update UNLESS:

- Page is not in dirtyPgTable
  - Page flushed prior to checkpoint, didn't redirty
- LSN < recLSN
  - Page flushed & redirtied prior to checkpoint
- LSN <= pageLSN
  - Page flushed after checkpoint

A/LSN 3

B/LSN 2

C/LSN 6

Disk

Page	pageLSN
A	3
B	5
C	6

Redo Example

Redo UNLESS

- Page is not in dirtyPgTable
- LSN < recLSN
- LSN <= pageLSN

DirtyPgTable

pgNo	recLSN
B	3
C	6
D	8
E	13

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Flush

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Redo Example

Redo UNLESS

- Page is not in dirtyPgTable
- LSN < recLSN
- LSN <= pageLSN

DirtyPgTable

pgNo	recLSN
C	6
D	8
E	13

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Flush

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Redo Example

Redo UNLESS

•Page is not in dirtyPgTable

•LSN < recLSN

•LSN <= pageLSN

DirtyPgTable

pgNo	recLSN
D	8
E	13

Flush

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	~~A~~
3	UP	1	3	~~B~~
4	CP			
5	SOT	3		
6	UP	1	3	~~C~~
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Redo Example

Redo UNLESS

- Page is not in dirtyPgTable
- LSN < recLSN
- LSN <= pageLSN

Flush

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

DirtyPgTable

pgNo	recLSN
D	8
E	13

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Redo Example

Redo UNLESS

- Page is not in dirtyPgTable
- LSN < recLSN
- LSN <= pageLSN

DirtyPgTable

pgNo	recLSN
B	10
D	8
E	13

Flush

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	~~A~~
3	UP	1	3	~~B~~
4	CP			
5	SOT	3		
6	UP	1	3	~~C~~
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Redo Example

Redo UNLESS

- Page is not in dirtyPgTable
- LSN < recLSN
- LSN <= pageLSN

DirtyPgTable

pgNo	recLSN
A	11
B	10
D	8
E	13

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Flush

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

Redo Example

Redo UNLESS

- Page is not in dirtyPgTable
- LSN < recLSN
- LSN <= pageLSN

DirtyPgTable

pgNo	recLSN
A	11
B	10
D	8
E	13

Flush

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	~~A~~
3	UP	1	3	~~B~~
4	CP			
5	SOT	3		
6	UP	1	3	~~C~~
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Disk

Page	pageLSN
A	2
B	3
C	6
D	?
E	?

State identical to pre-crash state

Undo

- Walk backwards, following prevLSNs to UNDO losers

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
13	3

Undo

- Walk backwards, following prevLSNs to UNDO losers

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
13	3

Undo

- Walk backwards, following prevLSNs to UNDO losers

LSN	Type	Tid	PrevLSN	Data
1	SOT	1		
2	UP	1	2	A
3	UP	1	3	B
4	CP			
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

xactionTable

lastLSN	TID
13	3

•Why can we just blindly apply UNDOs? Repeated history!

Innovation Check

F

13	UP	3	10	E

state

Study Break

LSN	Txn ID	Type	Page ID / Object
10	Checkpoint		
11	T1	SOT	
12	T1	UP	P1/A
13	T2	SOT	
14	T3	SOT	
15	T2	UP	P5/B
16	T2	Commit	
17	T3	UP	P3/C
18	Checkpoint		
19	T3	UP	P3/C
20	T3	Commit	

No flushes occur during the execution of these transactions. At the time of the first checkpoint (LSN 10), the dirty page table and the transaction table are both empty.

recLSN = first LSN that dirtied the page

lastLSN = most recent log record written by the transaction

Study Break

LSN	Txn ID	Type	Page ID / Object
10	Checkpoint		
11	T1	SOT	
12	T1	UP	P1/A
13	T2	SOT	
14	T3	SOT	
15	T2	UP	P5/B
16	T2	Commit	
17	T3	UP	P3/C
18	Checkpoint		
19	T3	UP	P3/C
20	T3	Commit	

PageID	recLSN
P1	12
P3	17
P5	15

Dirty Page Table

Transaction ID	lastLSN
T1	12

Transaction Table

What must the status of the tables have been at the time of the crash?

Study Break # 2

LSN	Txn ID	Type	Page ID / Object
10	Checkpoint		
11	T1	SOT	
12	T1	UP	P1/A
13	T2	SOT	
14	T3	SOT	
15	T2	UP	P5/B
16	T2	Commit	
17	T3	UP	P3/C
18	Checkpoint		
19	T3	UP	P3/C
20	T3	Commit	

No flushes occur during the execution of these transactions. At the time of checkpoint at LSN 10, the dirty page table and the transaction table are both empty.

1. At what LSN does the analysis phase begin? **18**

2. At what LSN does the REDO phase begin? **Min(recLSN) = 12**

3. What is the first LSN that is undone? **12**

Truncating Log

- Do we have to keep log forever?
- What is the earliest point in the log we will ever look at?

min(last checkpoint,min(recLSN))

➔we can safely truncate anything earlier

Are we done?

Compensation Log Records (CLRs)

- CLR record written after each UNDO
- Avoid repeating UNDO work
- Why?
  - Because UNDO Is logical, and we don't check if records have already been UNDONE. Could get into trouble if re-undid some logical operation.

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

Losers: 3

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E
14	CLR	3	13	E, 10

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E
14	CLR	3	13	E, 10

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E
14	CLR	3	13	E, 10
15	CLR	3	14	B, 5

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E
14	CLR	3	13	E, 10
15	CLR	3	14	B, 5

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E
14	CLR	3	13	E, 10
15	CLR	3	14	B, 5

UNDO with CLR

LSN	Type	Tid	PrevLSN	Data
5	SOT	3		
6	UP	1	3	C
7	SOT	2		
8	UP	2	7	D
9	EOT	1	6	
10	UP	3	5	B
11	UP	2	8	A
12	EOT	2	11	
13	UP	3	10	E
14	CLR	3	13	E, 10
15	CLR	3	14	B, 5
16	EOT	3	15	

REDO with CLR

- REDO CLRs on crash recovery
  - Use REDO rules to check if updates in CLRs have already been done
    - Avoids repeating operational (escrow) operations
  - After processing CLR, update recLSN field in dirtyPgTable
    - Allows UNDO to start from the right place, should we checkpoint while UNDOing

ARIES/Logging Recap

- NO FORCE, STEAL logging
- Use write ahead logging protocol
- Must FORCE log on COMMIT
- Periodically take (lightweight) checkpoints
- Asynchronously flush disk pages (without logging)

Disaster Recovery

- What if:
  - Disk on machine fails
  - Computer won't restart
  - Data center loses power
  - …
- Solution: replication

Replication

- Typical approach: dedicated "hot standby"
  - Kept up to date via "log shipping" – it executes operations in the log in identical order to the primary
- May have several replicas, one nearby in local data center, one further away
  - "Half-width of a hurricane"
- Replicas often used for read-only queries
  - Have excess capacity because they are not processing xactions, just replaying log

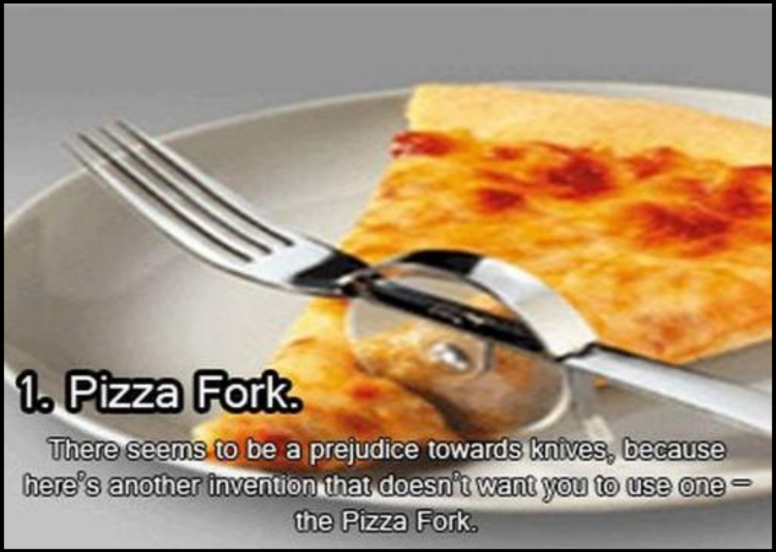
Replica Fail Over

- On failure, start directing queries to replica
- Bring up new replica
  - Using, e.g., nightly backup + log
- Complex in practice:
  - Have to be really sure the database failed
    - Many organizations rely on manual failover
  - Failover needs to be tested frequently
  - Replication load can be significant

Transactions Summary

- Transactions provide a powerful way to isolate concurrent operations on the DB
- Studied two-phase locking
- Saw how write-ahead logging can be used to provide recoverability and roll-back
- Next time: Optimistic Concurrency Control, distributed DBs, and distributed txns

Thank You

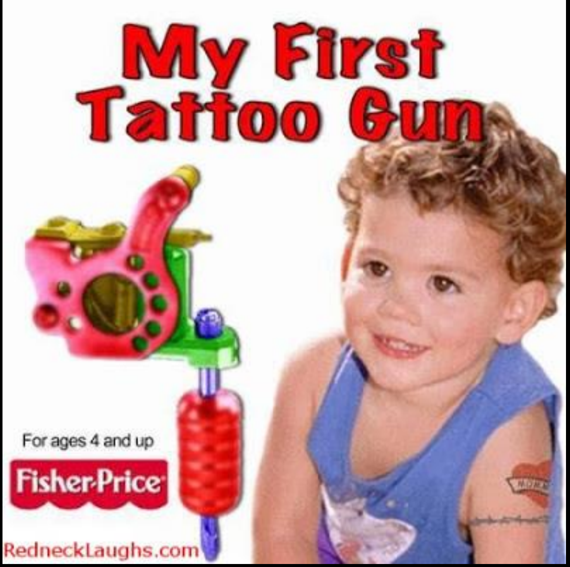

1. Pizza Fork.
There seems to be a prejudice towards knives, because here's another invention that doesn't want you to use one – the Pizza Fork.


5 Doramad-Zahnpfleger stellen sich vor
Ich bin die radioaktive Substanz. Meine Strahlen massieren das Zahnfleisch. Gesundes Zahnfleisch - gesunde Zähne.
Ich bin die medizinische Seife - mein Schaum reinigt die ganze Mundhöhle bis in alle Winkel.
Ich - der Emulgator - sorge dafür, daß „DORAMAD" immer sahnig und frisch bleibt!
Ich bin das Aroma - durch mich erfrischt „DORAMAD" köstlich die gesamte Mundhöhle!
Ich - der ganz feine Putzkörper - mache die Zähne blendend weiß, schone den Schmelz!
Das ist die radioaktive biologisch wirksame Zahncreme
Doramad
Radioaktive Zahncreme
KLEINE TUBE 45 ₰
GROSSE TUBE 75 ₰
EIN ERZEUGNIS DER AUERGESELLSCHAFT · A · G · BERLIN · N · 65


Control Toys
LOVELY
Strait Jacket
5+ YEARS


TOP 10 MOST AWKWARD INVENTIONS
10. Chopstick Fan
The Chopstick Fan, because hot noodles is just asking for trouble. The chopstick fan cools down those noodles on your chopsticks like nothing else. I mean, who wants to wait for natural airflow?


5. Toilet roll hat
I doubt the toilet roll hat will ever catch on. If you seriously need that much tissue, you shouldn't be out in the first place. Either way, the inventor clearly wasn't a big fan of those things we call 'pockets'.


My First Tattoo Gun
For ages 4 and up
Fisher-Price
RedneckLaughs.com