Database Management Systems II

Robert Gottstein

gottstein@dvs.tu-darmstadt.de

TECHNISCHE UNIVERSITÄT DARMSTADT

Buffer Management and Recovery

flashyDB DVS

Exercise 8.1
Buffer Management and Recovery

a) Which of the ACID properties of transactions does the recovery manager (RM) enforce?

Concurrency Control - takes care of Isolation

Recovery - takes care of Atomicity & Durability

Ultimate goal:

- transactions execute **atomically** (isolated from one another, no interference; either all Tx operations executed and made permanent or none).

Exercise 8.1
Buffer Management and Recovery

b) Explain the "steal / no steal" and "force / no force" strategies for buffer management. (RM=Recovery Manager; BM=Buffer Manager)

Steal: RM allows BM to flush pages containing uncommitted data to disk (Note: Flush is possible after unfix, the earliest).

No Steal: RM forces BM to keep modified pages in buffer until transaction commits (simplifies UNDO)

Force: RM forces BM to flush pages written by a transaction when the transaction commits (avoids partial REDO).

No Force: may allow BM to keep modified pages in buffer.

Exercise 8.1
Buffer Management and Recovery

Simplify UNDO??

A "**Steal**" approach allows **uncommitted** data to be written to the stable database.

- *Should a system failure occur before a transaction has committed, all data it has written to the stable database must be **UNDONE** on Restart*.

Avoid Partial REDO??

A "**No Force**" approach allows a transaction to **commit before** all of its **writes have been propagated** to the stable database. *Should a system failure occur at this point, the transaction's writes will have to be **REDONE** on Restart.*

Exercise 8.1
Buffer Management and Recovery

TECHNISCHE UNIVERSITÄT DARMSTADT

c) What actions are required at restart, depending on the chosen buffer management strategy? Discuss the advantages and disadvantages of different strategies.

Strategy	UNDO	REDO
No Steal / Force	NO	NO
Steal / Force	YES	NO
No Steal / No Force	NO	YES
Steal / No Force	YES	YES

'Force' ⇔ **'No REDO'**
'No Force' ⇔ **'REDO'**
'Steal' ⇔ **'UNDO'**
'No Steal' ⇔ **'No UNDO'**

Exercise 8.1
Buffer Management and Recovery

NO STEAL:

(+) **No need for UNDO**, but

(-) **Poor throughput**

If a "No Steal" approach is used the system ***might quickly run out of buffer space***, ***blocking some transactions and leading to thrashing.***

FORCE:

(+) **No need for REDO**, but

(-) **Poor response time**, due to excessive random disk I/O at commit.

Exercise 8.1
Buffer Management and Recovery

- Most commercial systems employ a "**Steal, No-Force**" buffer management policy.

- Efficiency during normal operation is maximized at the expense of less efficient processing of failures.

Exercise 8.1
Buffer Management and Recovery

d) Describe the WAL Protocol.

When must "update log records" (incl. before/after images of changed data items) be written to stable storage?

Exercise 8.1
Buffer Management and Recovery

TECHNISCHE UNIVERSITÄT DARMSTADT

To enforce ***Atomicity*** and ***Durability*** in the presence of failures:

Undo Rule:
If x's location in the stable database presently contains the last ***committed*** value of x, then that value must be saved in stable storage before being overwritten in the stable database by an ***uncommitted*** value.
Enables Undo, on transaction/crash recovery.

Redo Rule:
Before a transaction can commit, all data that it has been written must be stored on stable storage (e.g. in the stable database or a log).
Enables Redo on crash recovery.

Exercise 8.1
Buffer Management and Recovery

*These rules ensure that the **last committed value** of each data item is always available **in stable** storage.*

Thus, in case of a failure, **the recovery system can always transform the stable database to a consistent state** - the last committed state.

Most systems enforce the above rules using the so-called **Write-Ahead Logging Protocol (WAL)**

Exercise 8.1
Buffer Management and Recovery

TECHNISCHE UNIVERSITÄT DARMSTADT

The WAL protocol logs all update operations to stable storage, ensuring that:

1. The **log record for an update is forced to disk before the affected data page** gets to disk.

2. Before **a transaction commits**:
   all log records for its updates are forced to disk (only log records)

Exercise 8.1
Buffer Management and Recovery

Under the WAL protocol when a transaction ***commits, all log records of its writes are forced to disk.*** It is worth to contrast this with the operations taken under the FORCE approach*.*

If a FORCE approach is used ***all the pages modified*** by a transaction, rather than a portion of the log that includes all its records, must be forced to disk on commit.

The set of **all changed pages is typically much larger than the log records**, because the size of an update log record is close to the size of the changed bytes, which is likely to be much smaller than the page size. Further, the log is maintained as a **sequential file**, and thus all writes to it are **sequential** writes. Consequently, the **cost of forcing the log tail is much smaller** than the cost of writing all changed pages to disk.

Exercise 8.2

TECHNISCHE UNIVERSITÄT DARMSTADT

Recovery Algorithms

Exercise 8.2
Recovery Algorithms

a) What **phases does the Restart procedure** after a crash usually go through?

b) Discuss the difference between the **Redo-Winners and Redo-History** approach to developing Recovery algorithms.

c) **Why is checkpointing needed?** Describe the major types of checkpointing and the trade-offs that they achieve between 'performance during normal operation' and 'performance during restart'?

Exercise 8.2
Recovery Algorithms

After a system crash:

- TXs that committed before the crash are called **'Winner TXs'**, i.e. TXs for which **a commit log** entry is found in the stable log.
- TXs active at the time of the crash are called **'Loser TXs'**, i.e. TXs for which **neither commit nor abort log entry exists** in the stable log.

Exercise 8.2
Recovery Algorithms

RESTART usually proceeds in 3 phases:

1. **Analysis Phase**
   Examines the log and collects information about the system state as of the time of the crash - e.g. winner-TXs, loser-TXs, dirty pages, etc.
2. **REDO Phase (REDO-Recovery)**
   Actions of winner TXs (and optionally of loser TXs) are redone.
3. **UNDO Phase (UNDO-Recovery)**
   Actions of loser-TXs are undone.

Note:

some algorithms may perform the 2-nd and 3-rd phase in the opposite order.

Exercise 8.2
Recovery Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Log Sequence Number

LSN	LOG
10	**Update: T1 writes P5**
20	**Update: T2 writes P3**
30	**T2 commit**
40	**T2 end**
50	**Update: T3 writes P1**
60	**Update: T3 writes P3**
X	**Crash, Restart**

1. At restart, **analysis** identifies
   - T1,T3 as transactions active at crash, must be undone
   - T2 committed, all actions must be written to disk
   - P1, P3, P5 potentially dirty pages
2. **Redo phase**
   - all updates are reapplied in order (including those of T1 and T3)
3. **Undo phase**
   - uncommitted updates are undone in reverse order (update T3 to P3, update T3 to P1, update T1 to P5)

Exercise 8.2
Recovery Algorithms

Two general approaches to developing a recovery algorithm:

REDO-Winners Paradigm:
only updates of winner transactions are redone at restart.

REDO-History Paradigm:
all updates are redone at restart (both of winner and loser Txs). Restores state as of the time of the failure. After that updates of loser transactions are undone in the **UNDO** Phase.

Exercise 8.2
Recovery Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

REDO-History

- ***Simplifies*** the overall recovery algorithm by treating winner-, loser- and aborted-transactions more uniformly
- Avoids having to flush any pages during Restart
- Restart may end up first redoing certain loser actions and then undoing them in the UNDO phase. However, overhead is usually not too big.

Exercise 8.2
Recovery Algorithms

c) Why is checkpointing needed? Describe the major types of checkpointing and the trade-offs that they achieve between 'performance during normal operation' and 'performance during restart'?

Checkpointing:

Activity that writes information to stable storage during normal operation in order to reduce the amount of work that would have to be done by RESTART in case of a crash.

Exercise 8.2
Recovery Algorithms

3 types of activities are performed at checkpoint:

1. **Storing TX state information**: active, committed, aborted TXs,...

2. **Storing buffer state information:** dirty pages,...

3. **Flushing buffer pages to disk.**

Exercise 8.2
Recovery Algorithms

3 Major Types of Checkpointing:

1. **Transaction-Consistent**
   (also called Commit-Consistent)

2. **Action-Consistent**
   (also called Cache-Consistent)

3. **Fuzzy-Checkpointing (FC)**

Transaction-Consistent

checkpoint():

- Stop processing **new transactions** (!)
- Wait for all active **transactions** to finish
- Flush all dirty buffer pages to disk
- Mark the log with a checkpoint-marker
- Continue processing transactions

Transaction-Consistent

restart():

- Must only consider TXs active **after** the checkpoint
- REDO from checkpoint-marker to end of the log
- UNDO from log end backwards until the checkpoint-marker

Transaction-Consistent

Evaluation:

Pro:

restart() is simplified

Con:

checkpoint() slow
$\Rightarrow$ long downtimes during normal operation

Action-Consistent

checkpoint():

- Stop processing **new operations**
- Wait for all started **operations** to finish
- Flush all dirty buffer pages to disk
- Write a checkpoint-marker in log containing "active_TX_list"
- Continue processing operations

Action-Consistent

restart():

- Must only consider transactions that were active **at and after** the checkpoint
- REDO from checkpoint-marker to the end of the log
- UNDO from log end backwards until the checkpoint-marker, then continue further until all loser transactions are undone.

Action-Consistent

Evaluation:

Pro:

reduces checkpoint delay

Con:

flushing the whole buffer is still quite costly

Fuzzy-Checkpointing (FC)

There are ***different variants*** of Fuzzy-Checkpointing. Will look at the one proposed by Bernstein et. al, which is an optimization of Action-Consistent checkpointing - Generic Fuzzy Checkpointing.

Fuzzy-Checkpointing (FC)

checkpoint():

- Stop processing **new operations**
- Wait for all started **operations** to finish
- Flush **only** the buffer pages that were dirty at the last checkpoint and still haven't been flushed since then
- Create a newdirty_pages_list and store it in memory, so that you can later use it to determine which pages to flush at the next checkpoint.
- Write a checkpoint-marker in log containing "active_TX_list"
- Continue processing operations

Fuzzy-Checkpointing (FC)

restart():

- must only consider transactions that were active **at and after** the penultimate checkpoint
- REDO from penultimate checkpoint-marker to the end of the log
- UNDO from log end backwards until the penultimate checkpoint-marker, then continue further until all loser transactions are undone.

Fuzzy-Checkpointing (FC)

Evaluation:

Pro:

Further **reduces checkpoint delay** - the hope is that the buffer manager's normal replacement activity will flush most pages that were dirty and not flushed at the previous checkpoint.

Con:

Still requires some flushing at checkpoint and delays active transactions.

Fuzzy-Checkpointing (FC)

Note:

The **ARIES-Style Fuzzy-Checkpointing** further improves performance by not requiring any pages to be flushed during checkpointing.

The ARIES Algorithm

Based on: “Chapter 20: Crash Recovery” (Slides)
Database Management Systems. R. Ramakrishnan and J. Gehrke

Exercise 8.3
The ARIES Algorithm

Describe the ARIES Algorithm for Crash Recovery:

- What ***types of log records*** are used and what ***structures*** do they have?
- What ***management information*** is maintained in main memory?
- What is ***done during the Analysis, REDO and UNDO phases*** of the Restart procedure?
- How can ***pageLSRs be used to minimize the amount*** of work to be done at Restart?
- What are ***Compensation Log Records*** used for?
- How is ***checkpointing*** implemented?

Based on: “Chapter 20: Crash Recovery” (Slides)
Database Management Systems. R. Ramakrishnan and J. Gehrke

Exercise 8.3
The ARIES Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

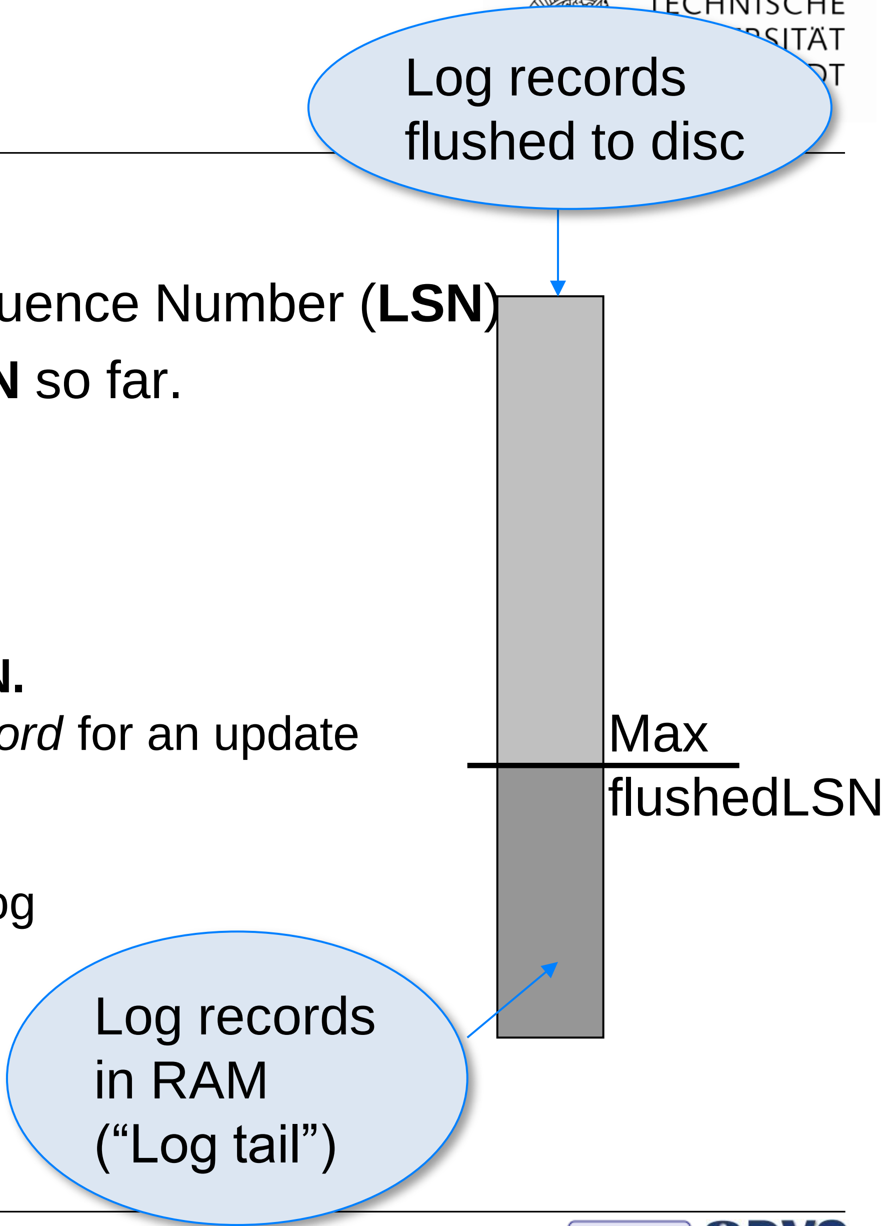
Normal Execution of a Transaction

- "Series of reads & writes operations, followed by commit or abort"
- We will assume that write is atomic on disk
- Strict 2PL
- STEAL, NO-FORCE buffer management, with Write-Ahead Logging

Exercise 8.3
The ARIES Algorithm

WAL

- Each log record has a unique Log Sequence Number (**LSN**)
- System keeps track of last **flushedLSN** so far.



- Each ***data page*** **contains a pageLSN.**
  - The LSN of the most recent *log record* for an update to that page.

- **WAL**: Before a page is written, flush log
  ➔ *pageLSN* $\leq$ *flushedLSN*

flashyDB DVS

Exercise 8.3
The ARIES Algorithm

- What ***types of log records*** are used and what ***structures*** do they have?

Exercise 8.3
The ARIES Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

Log Records

LSN	PrevLSN	TransID	type	pageID	length	offset	Before image	After-image	Undonext LSN

Common to all log records | **Specific for update log records** | **Specific for CLRs**

Possible log record types:

- **Update**
- **Commit** (Force write causes log tail to be flushed to disk, remove Transaction from Transaction Table)
- **Abort** (➔Initiat Undo)
- **End** (signifies end of commit or abort (cleanup complete)
- **Compensation Log Records (CLRs)**
  – for UNDO actions

Exercise 8.3
The ARIES Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

Transaction and Dirty Page Tables

Transaction Table:

- contains **one entry for each active transaction**
- entries contain **transaction ID**, **status** (in progress, committed, aborted), **lastLSN** (points to last log entry for TX)
- if *status is aborted or committed*, ***entry is removed after clean-up***

Dirty Page Table:

- contains an entry for each modified page in buffer pool that has not been written to disk
- **recLSN** is the LSN *for first log record that caused page to become dirty* (earliest log record that might have to be redone for this page during restart)

Exercise 8.3
The ARIES Algorithm

Checkpointing

Write to log:

- **begin_checkpoint record:**
  Indicates when chkpt began
- **end_checkpoint record:**
  Contains current ***Transaction table (TT)*** and ***dirty page table (DPT)*.**

This is a 'fuzzy checkpoint':

- Other Transactions continue to run ➔ TT and DPT 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**

Background process flushes dirty pages to disk

Store LSN of chkpt record in a safe place (*master* record)

Exercise 8.3
The ARIES Algorithm

- What ***management information*** is maintained in main memory?

  (What is stored where?)

Exercise 8.3
The ARIES Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

What's Stored Where

LogRecords	➔	Log (Stable Storage / **RAM**)
Data Pages	➔	DB
Master Record	➔	Stable Storage (e.g. DB)
Transaction Table	➔	**RAM**
Dirty Page Table	➔	**RAM**
Flushed LSN	➔	**RAM**

Exercise 8.3
The ARIES Algorithm

Simple Transaction Abort ***(No Crash)***

1. **Get lastLSN** of Transaction from TT.
   Can follow chain of log records backward via the prevLSN field.

2. **Write Abort** log record (Before starting UNDO)

To perform UNDO, must have a lock on data
➔No Problem (Why?)

3. Before restoring old value of a page ➔**write CLR**
   CLR has a pointer to the next LSN to undo
   CLRs never Undone (but might be Redone (Why?))

4. At end of UNDO, **write an “end” log record**

Exercise 8.3
The ARIES Algorithm

Transaction Commit

1. **Write "commit" record to log**

2. **Flush all log records** up to Transaction's lastLSN (➔TT)

   ➔Guarantees that flushedLSN ≥ lastLSN

3. **Write "end" record to log.**

Exercise 8.3
The ARIES Algorithm

Crash Recovery

Analysis

Figure out which Transactions committed since checkpoint, which failed

Reconstruct DPT

REDO

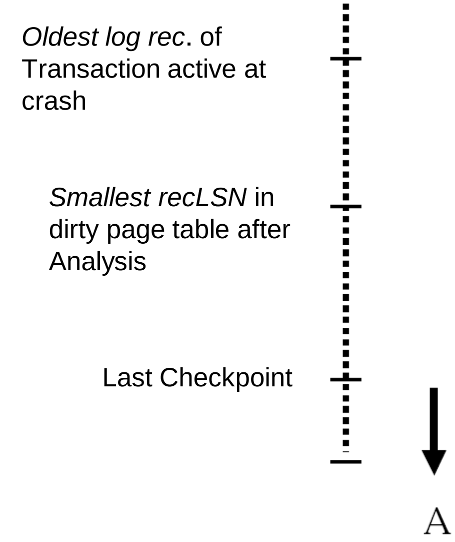
REDO **all** actions (Winners & Losers)

UNDO

UNDO effects of failed transactions

Exercise 8.3
The ARIES Algorithm

The Analysis Phase



1. Reconstruct state at checkpoint

2. Scan log forward from checkpoint

   End record:
   Remove Transaction from Transaction table

   Other records: Add Transaction to Transaction table (set lastLSN=LSN)

   Update record:
   If P not in Dirty Page Table,
   ➔ Add P to DPT, set its recLSN=LSN

Exercise 8.3
The ARIES Algorithm

The REDO Phase

Repeat *History* to reconstruct state at crash:

Reapply ***all* updates** (even of aborted Transaction!), redo CLRs

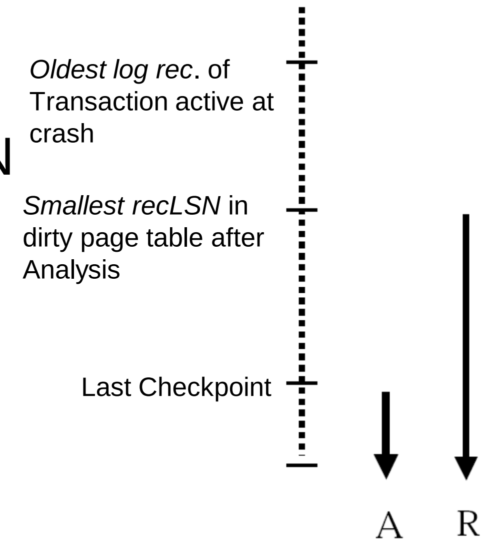
Scan forward from log rec containing smallest recLSN in DPT (earliest change)

For each CLR or update log rec, REDO action unless:

- Affected page is not in DPT
- Affected page is in DPT, but has recLSN > LSN
- pageLSN ≥ LSN

To REDO an action:

- Reapply logged action
- Set pageLSN to LSN. No additional logging!



Exercise 8.3
The ARIES Algorithm

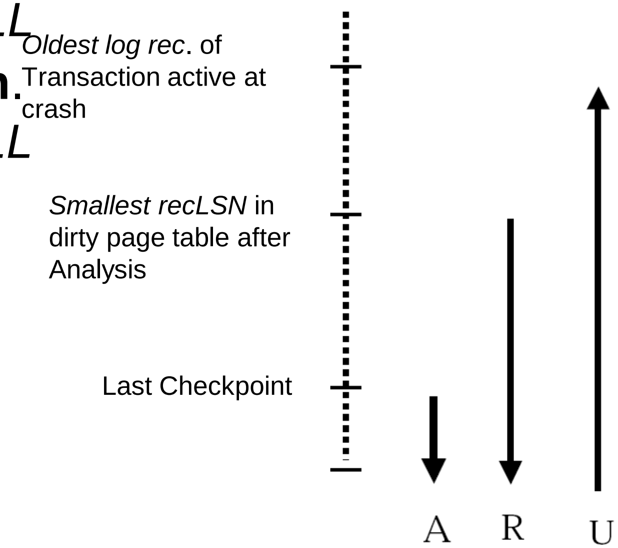
The UNDO Phase

ToUndo={ *l* | *l* a lastLSN of a “loser” Transaction}

Repeat

1. ***Choose largest LSN*** among ToUndo.
2. If this *LSN is a CLR and undonextLSN == NULL*
   ➔ **Write an End record for this Transaction.**
3. If this *LSN is a CLR, and undonextLSN != NULL*
   ➔ **undonextLSN to ToUndo**
4. Else *this LSN is an update*. **Undo the update,**
   ➔ **Write a CLR, add prevLSN to ToUndo**.

Until ToUndo is empty. ToUndo: 50, 60

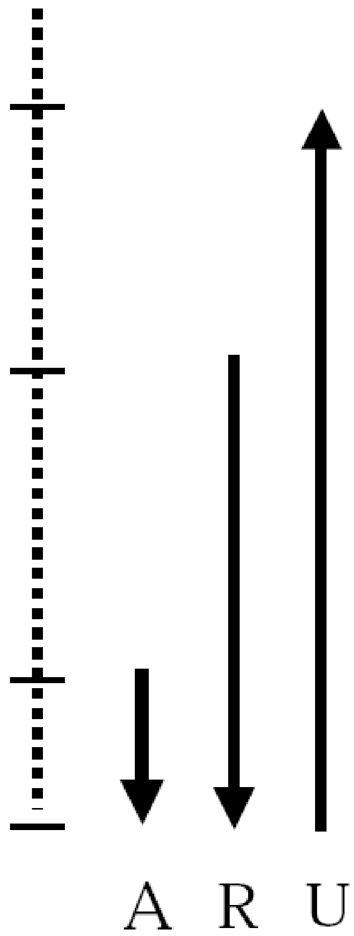


Exercise 8.3
The ARIES Algorithm

Oldest log rec. of Transaction active at crash

Smallest recLSN in dirty page table after Analysis

Last Checkpoint



EOE End of Exercise

TECHNISCHE UNIVERSITÄT DARMSTADT

Written exam will be on

March 31st 09:00 - 11:00 in S101/A1

flashyDB DVS