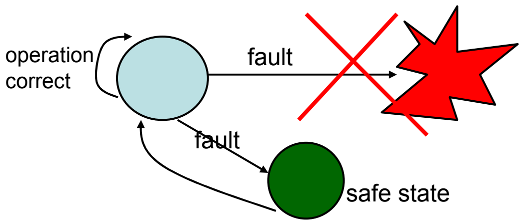
16 Logging and Recovery in Database systems

Lit.: Eickler/ Kemper chap 10, Elmasri /Navathe chap. 17, Garcia-Molina, Ullman, Widom: chap. 21

16.1 Introduction: Fail safe systems

- How to make a DBS fail safe ?
- What is "a fail safe system"?
  - system fault results in a safe state
  - liveness is compromised



- There is no fail safe system...

  ... in this very general sense
- Which types of failures will not end up in catastrophe?

Introduction

- Failure Model
  - What kinds of faults occur?
  - Which fault are (not) to be handled by the system?
  - Frequency of failure types (e.g. Mean time to failure MTTF)
  - Assumptions about what is NOT affected by a failure
  - Mean time to repair (MTTR)

16.1.2 DBS related failures

- Transaction abort
  - Rollback by application program
  - Abort by TA manager (e.g. deadlock, unauthorized access, ...)
    - frequently: e.g. 1 / minute
    - recovery time: < 1 second
  - System failure
  - malfunction of system
    - infrequent: 1 / weak (depends on system)
  - power fail
    - infrequent: 1 / 10 years
      (depends on country, backup power supply, UPS)

Assumptions:

- content of main storage lost or unreliable
- no loss of permanent storage (disk)

DBS related failure model

More failure types (not discussed in detail)

- Media failure (e.g. disk crash)
  - ⇨ Archive
- Catastrophic ("9-11-") failure
  - – loss of system
  - ⇨ Geographically remote standby system

Disks : ~ 500000 h (1996), see diss. on raids http://www.cs.hut.fi/~hhk/phd/phd.html

Fault tolerance

Fault tolerant system

– fail safe system, survives faults of the failure model

- How to achieve a fault tolerant system?
  - – Redundancy
    - Which data should be stored redundantly ?
    - When / how to save / synchronize them
  - – Recovery methods
    - Utilize redundancy to reconstruct a consistent state
      ⇨ "warm start"
  - – Important principle:
    Make frequent operations fast

Terminology

- Log
  - redundantly stored data
  - Short term redundancy
  - Data, operations or both
- Archive storage
  - Long term storage of data
  - Sometimes forced by legal regulations
- Recovery
  - Algorithms for restoring a consistent DB state after system failure using log or archival data

16.2 DBS Logging and Recovery Principles

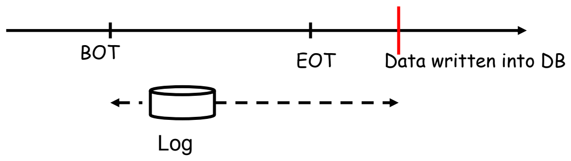
Transaction failures

- Occur most frequently
- Very fast recovery required
- Transactional properties must be guaranteed

Assumption of failure model:
data safe when written into database

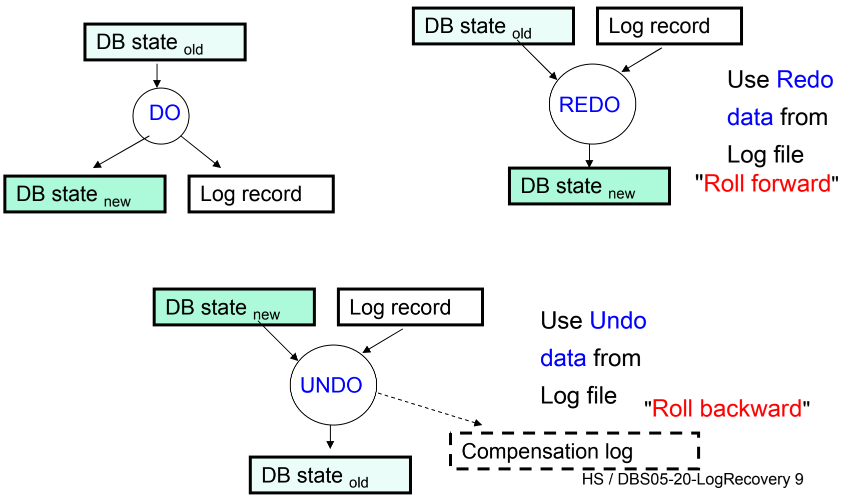
When should data be written into DB / when logged?

How should data be logged?



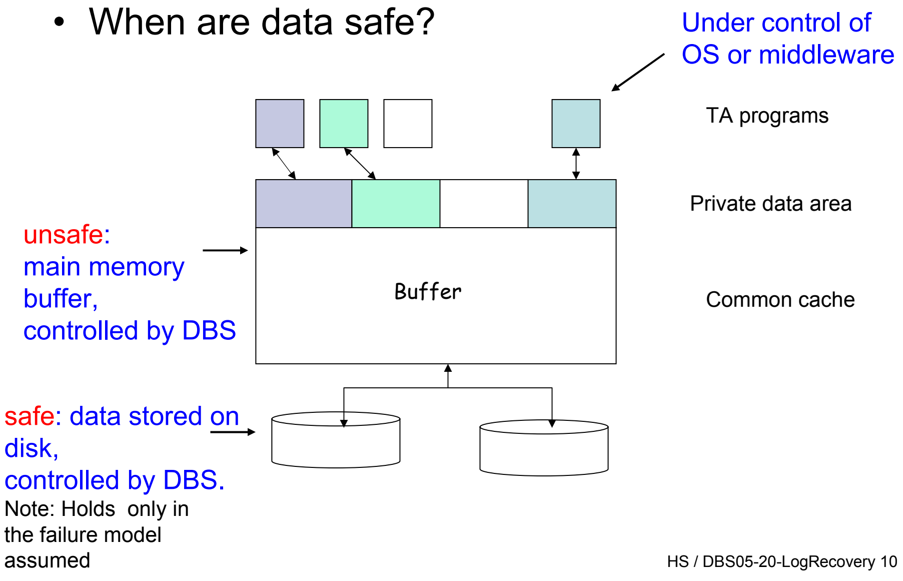
16.2.1 The UNDO / REDO Principle

- Do-Undo-Redo



DBS Architecture

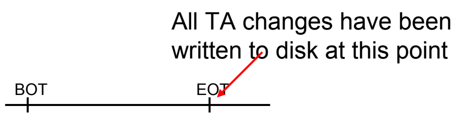
- When are data safe?


Under control of OS or middleware
TA programs
Private data area
Common cache
Buffer
unsafe: main memory buffer, controlled by DBS
safe: data stored on disk, controlled by DBS.

Note: Holds only in the failure model assumed

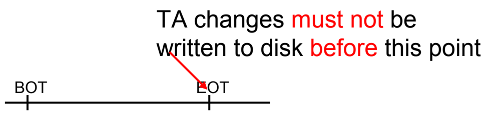
Redo / Undo

- Why REDO ?
  - Changed data into database after each commit
    ⇨ no redo
  - In general too slow to force data to disk at commit time



Redo / Undo

- Why UNDO ?
  - no dirty data written into DB before commit:
    ⇨ no undo

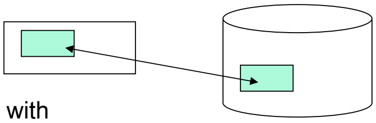


- Logging and Recovery dependent from other system components
  - Buffer management
  - Locking (granularity)
  - Implementation of writes into DB

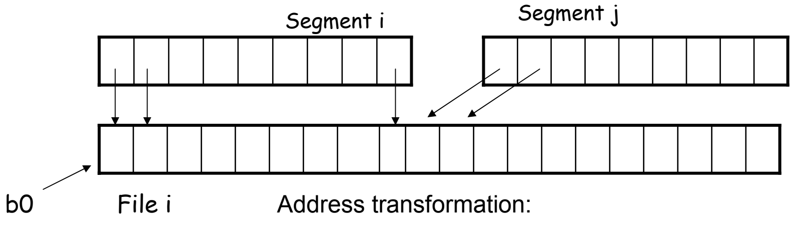
16.2.2 Writing into the DB

Update-in-place

A data page is written back to its physical address



Simple implementation with
Direct page adressing

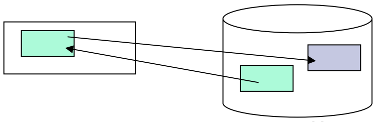


Address transformation:

blockAdr_File = b0 + d*pageAdr_Seg

Writing data

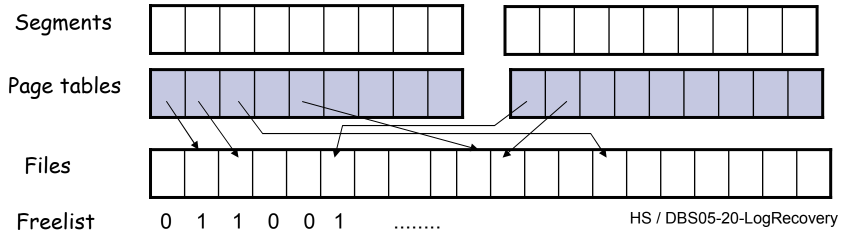
- Indirect write to DB



Advantage: simple undo

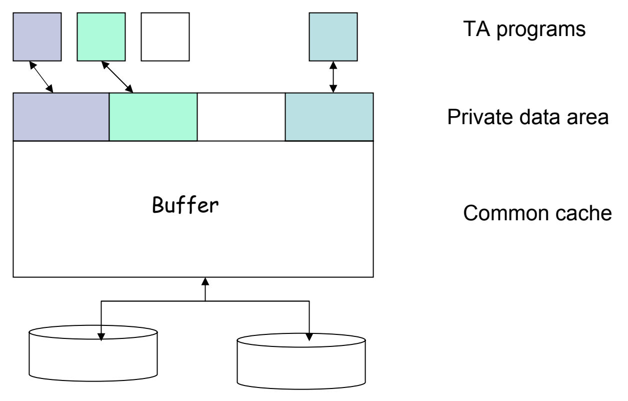
Issue: how can multiple writes be made atomic

  – Implementation by look-aside buffer
  – Simple implementation by indirect page adressing
    - Block address of a segment page is looked up in page table



16.2.3 Buffer Management

- Influence of buffering
  - Database buffer (cache) has very high influence on performance



DBS Buffer

- Buffer management
  - Interface:

fetch(P)	load Page P into buffer (if not there)
pin(P)	don't allow to write or deallocate P
unpin(P)	
flush(P)	write page if dirty
deallocate(P)	release block in buffer

  - No transaction oriented operations
- Influence on logging and recovery
  - When are dirty data written back?
  - Update-in-place or update elsewhere?
- Interference with transaction management
  - When are committed data in the DB, when still in buffer?
  - May uncommitted data be written into the DB?

Logging and Recovery Buffering

- Influence on recovery
  - Force: Flush buffer before EOT (commit processing)
  - NoForce: Buffer manager decides on writes, not TA-mgr
  - NoSteal : Do not write dirty pages before EOT
  - Steal: Write dirty pages at any time

	Steal	NoSteal
Force	Undo recovery no Redo	No recovery (!) impossible with update-in-place /immediate
NoForce	Undo recovery and Redo recovery	No Undo but Redo recovery

16.2.4 Write ahead log

Rules for writing log records

- Write-ahead-log principle (WAL)
  - before writing dirty data into the DB write the corresponding (before image) log entries
    WAL guarantees undo recovery in case of steal buffer management
- Commit-rule ("Force-Log-at-Commit")
  - Write log entries for all data changed by a transaction into stable storage before transaction commits
    This guarantees sufficient redo information