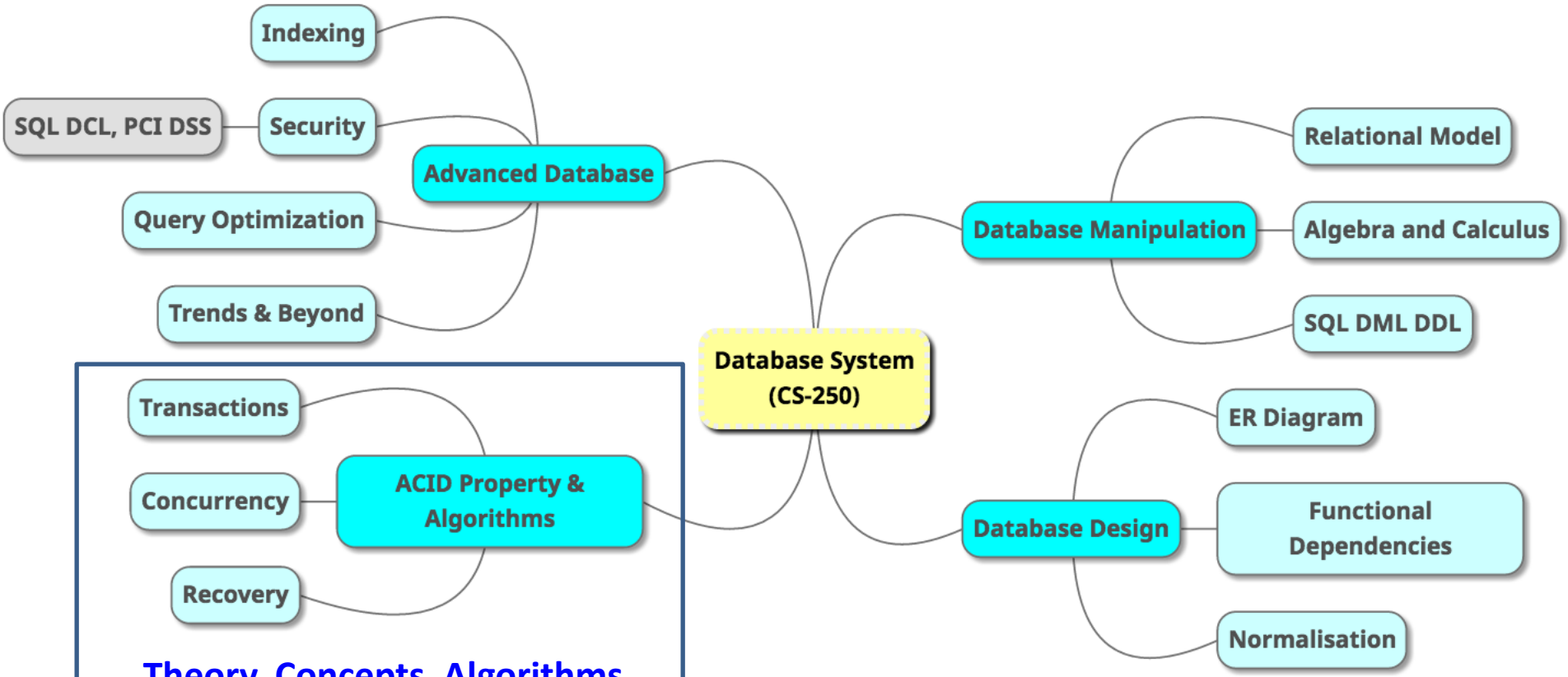
Recovery

Gary KL Tam

Department of Computer Science
Swansea University

Overview


Indexing
SQL DCL, PCI DSS
Security
Advanced Database
Query Optimization
Trends & Beyond
Relational Model
Database Manipulation
Algebra and Calculus
SQL DML DDL
Database System
(CS-250)
Transactions
Concurrency
ACID Property &
Algorithms
Recovery
ER Diagram
Database Design
Functional
Dependencies
Normalisation
Theory, Concepts, Algorithms
3 Inter-related Components
to uphold ACID properties

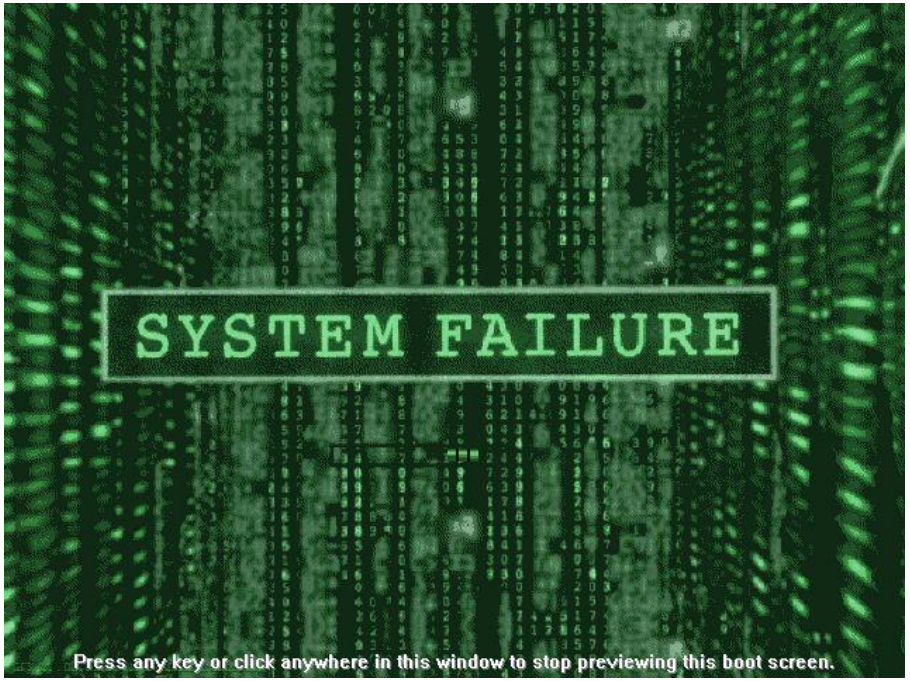
ACID Properties REVIEW

Each transaction must have:

- **Atomicity.** Either committed or aborted.
- **Consistency.** No violation of any user-defined constraint after the transaction finishes.
- **Isolation.** Concurrent transactions are not aware of each other. Each thinks that it was the only running transaction.
- **Durability.** If a transaction is committed, its changes to the database are permanent, even in the presence of system failures.

Failures Classification

- Transactions must be durable, but some failures will be unavoidable
  - Failure of integrity constraints
  - System crashes
  - Power failures
  - User mistakes
  - Sabotage
  - Software glitches
  - Hardware problems
  - Natural disasters
  - …

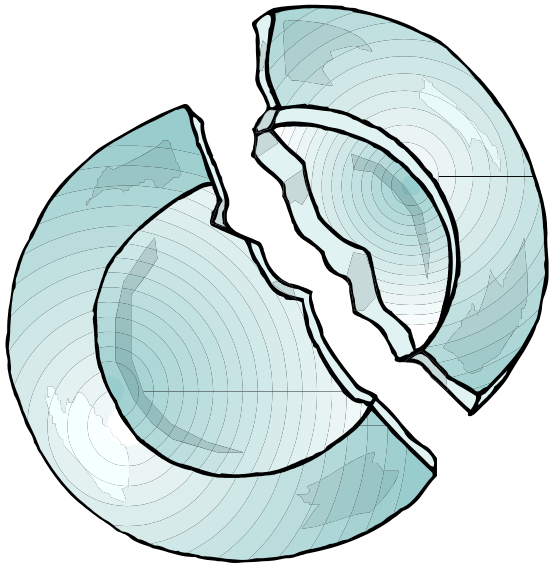


Recovery

- Prevention is better than a cure
  - Reliable OS
  - Security
  - Uninterruptible power supply (UPS)
  - Surge protectors
  - RAID arrays
- Can’t protect against everything, system recovery will be necessary

Recovery Algorithms

- Recovery techniques guarantee database consistency in case of failures.
- There are two parts:
  1. Actions taken during normal transaction processing to prepare the necessary information for recovery;
  2. Actions taken after a failure to recover the database content.



Stable Disks

- Next, we will introduce recovery algorithm assuming that the disk is stable, that is, it never loses any information under any circumstances.

- In practice, stable storage can be approximated by data duplication.
  - Maintain multiple copies of data on separate disks.
  - Store copies at remote sites.

- Data loss occurs only if all copies are destroyed simultaneously, which is very rare.

This lecture

Overview

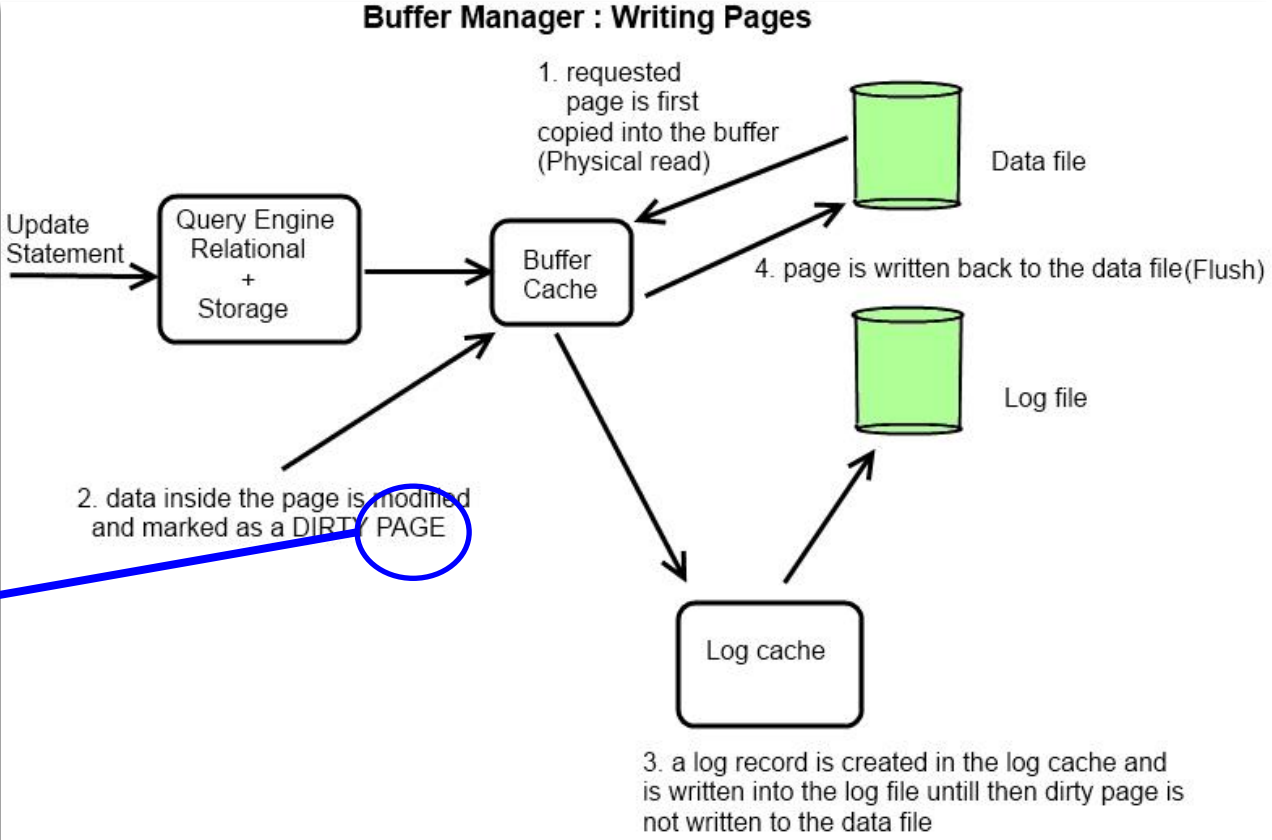
- Log-based recovery
  - Transaction logs
- Recovery for concurrent execution
- Backup

Transaction Log

- The transaction log records details of all transactions
  - Any changes the transaction makes to the database
  - When & how the transactions complete
- The log is stored on disk, not in memory
  - If the system crashes, the log is preserved
  - Write ahead log rule
  - The entry in the log must be made before COMMIT processing can complete

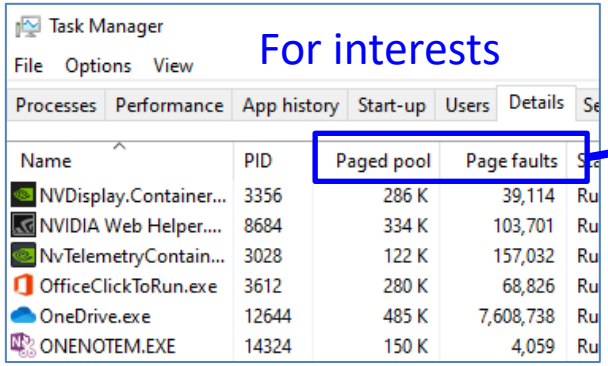
Example - MS SQL server

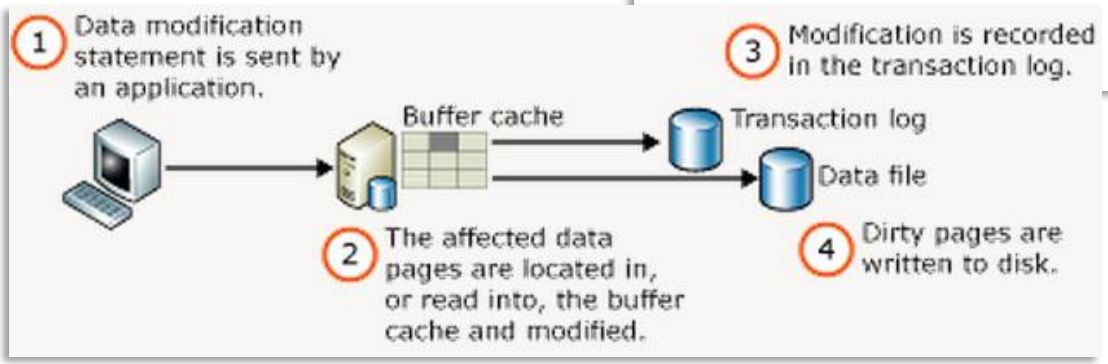
- Write-ahead logging (WAL)
- Log and Data
  - Two different disks



https://sqljunkieshare.com/2012/03/15/how-does-the-buffer-manager-writes-the-data-in-the-sql-server/

@sqljunkieshare

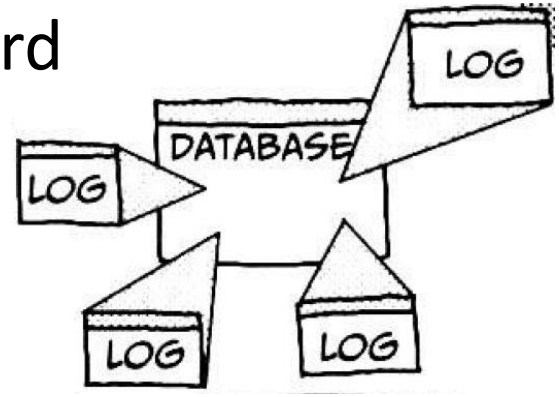




http://ellynguyen.wordpress.com/2011/07/19/ms-sql-server-system-processes/

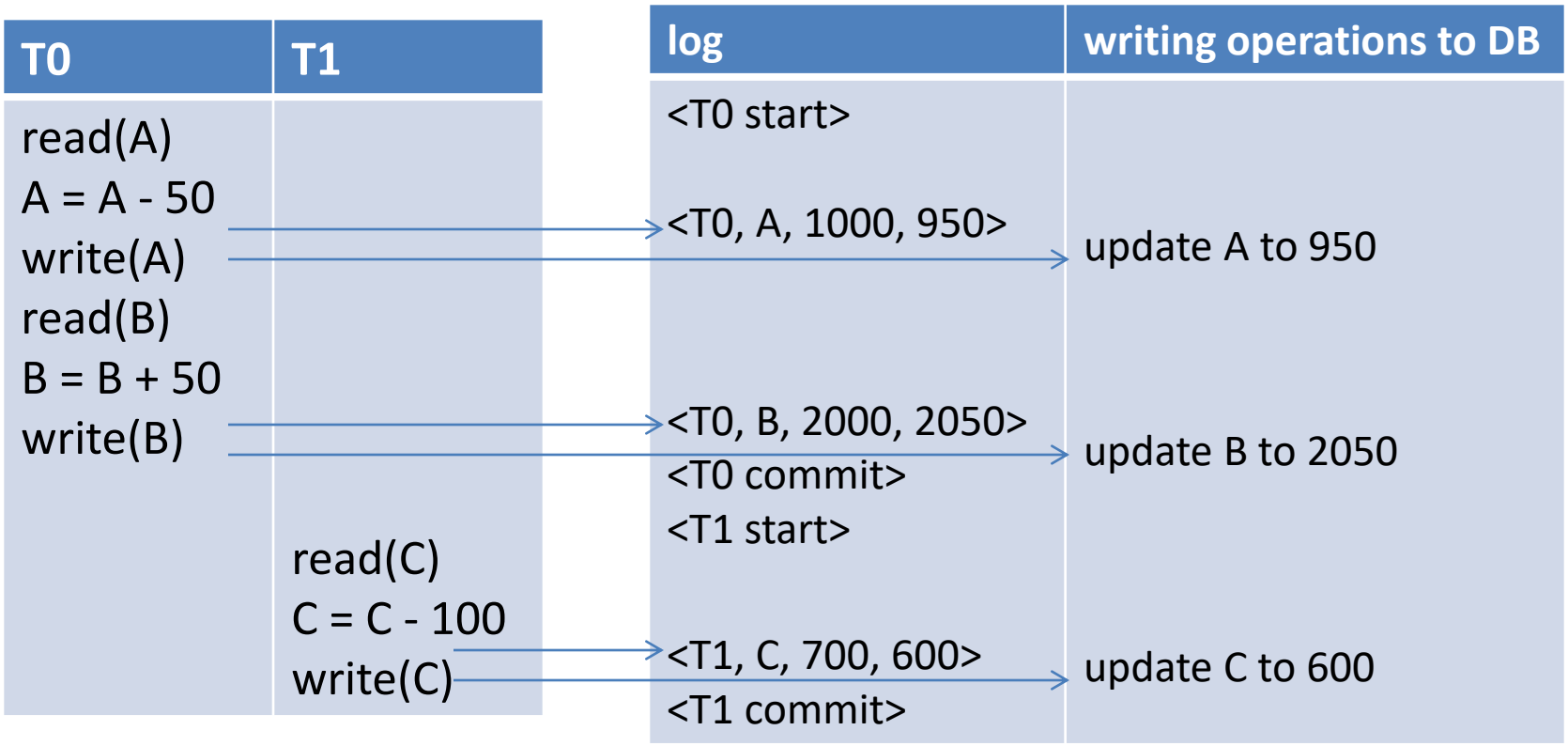
Log-based recovery

- The log records are created as follows.
- Before a transaction *T* starts, it writes a <*T* **start**> record
- Before *T* executes **write**(*X*), a log record <*T*, *X*, V_{old}, V_{new} > is created.
  - V_{old} is the old value of *X*.
  - V_{new} is the new value of *X*.



- Before *T* finishes, it creates a log record <*T* **commit**>.

Example – Creation of Log



<T start>, <T, X, V_{old}, V_{new}>, <T commit>

- writing is performed ***along with*** the creation of the log.

Example 1 with crash

log
<T0 start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>

- Consider the above log found after a recovery.
- T0 ***did not finish***, and must be aborted.
  - However, it already made some changes in the database (by modifying A to 950 and B to 2050).
  - Therefore, aborting it calls for un-doing these changes.
- Un-doing is done in the bottom-up order.
  - First, set B to 2000, according to the last log record.
  - Then, set A to 1000, according to the last-but-one record.

Example 2 with crash

log
<T0 start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0 commit>
<T1 start>
<T1, C, 700, 600>

- T0 committed. We do not need to do anything for it (its changes have been physically applied).
- T1 needs to be aborted.
- We un-do its changes by setting C to 700.

Example 3 with crash

log
<T0 start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0 commit>
<T1 start>
<T1, C, 700, 600>
<T1 commit>

- Both transactions committed.
- We do not need to do anything.
- (In practice, re-doing is normally carried out.)

A problem

- In practice, a transaction may have a large number of statements, which in turn generates a long log list.
- Problems in recovery.
  - Searching inside the log is time-consuming (e.g., for start/commit records)
  - Most transactions that need to be re-done may have already written all the changes to the disk physically. Although re-doing them causes no harm, it makes the recovery process longer.
- Solution: checkpoints

Transaction Timeline

- Recovery with checkpoint
  - Any transaction that was running at the time of failure needs to be undone
  - Any transactions that committed since the last checkpoint need to be redone

- To log a checkpoint, we use the entry:

<checkpoint *L*>
L - the list of transactions active at the time of the checkpoint

Checkpoint Example

log

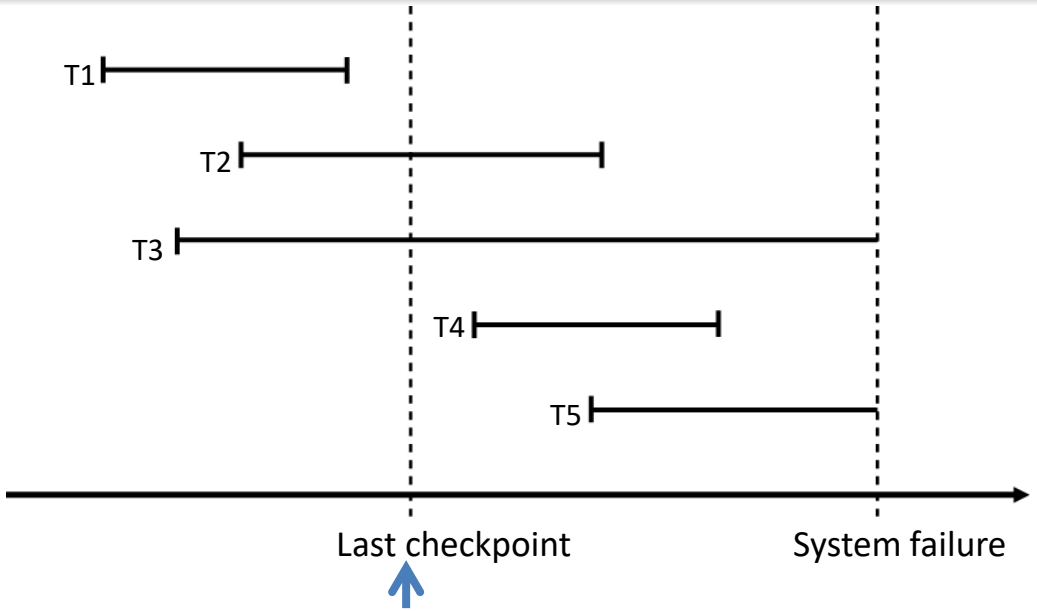
<T1 start>
<T1, A, 0, 10>
<T3 start>
<T3, E, 0, 12>
<T2 start>
<T2, B, 0, 10>
<T1 commit>
<checkpoint T2,T3 >
<T4 start>
<T4, A, 10, 20>
<T4, D, 0, 10>
<T5 start>
<T5, A, 20, 30>
<T2, C, 0, 10>
<T2 commit>
<T4 commit>
Failure

← T2, T3 is the transactions active at the time of the checkpoint

Checkpoint Example

log

<T1 start>
<T1, A, 0, 10>
<T3 start>
<T3, E, 0, 12>
<T2 start>
<T2, B, 0, 10>
<T1 commit>
<checkpoint T2,T3 >
<T4 start>
<T4, A, 10, 20>
<T4, D, 0, 10>
<T5 start>
<T5, A, 20, 30>
<T2, C, 0, 10>
<T2 commit>
<T4 commit>
Failure

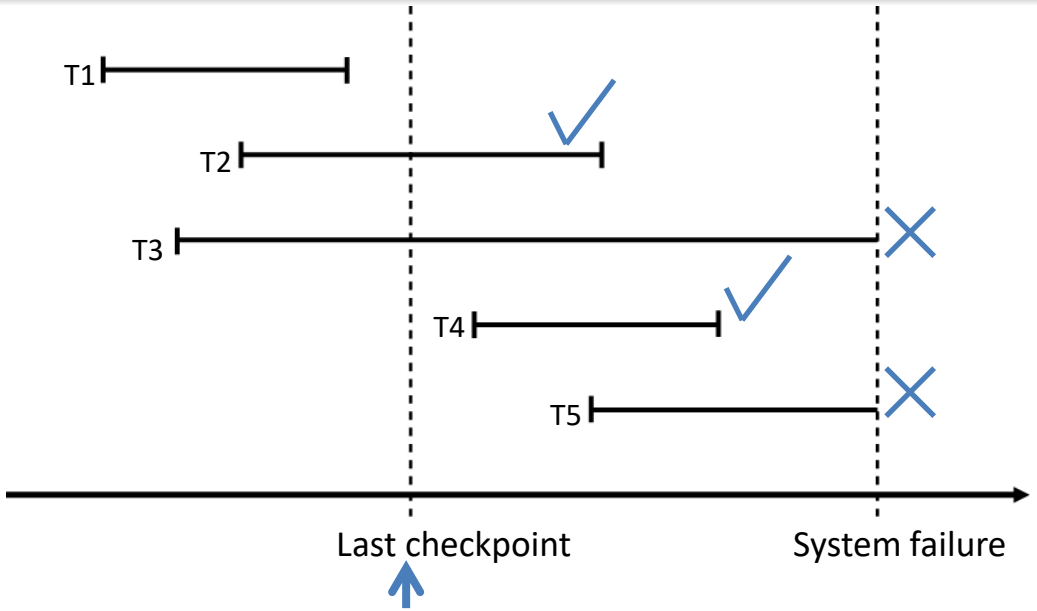


T2, T3 is the transactions active at the time of the checkpoint

Checkpoint Example

log

<T1 start>
<T1, A, 0, 10>
<T3 start>
<T3, E, 0, 12>
<T2 start>
<T2, B, 0, 10>
<T1 commit>
<checkpoint T2,T3 >
<T4 start>
<T4, A, 10, 20>
<T4, D, 0, 10>
<T5 start>
<T5, A, 20, 30>
<T2, C, 0, 10>
<T2 commit>
<T4 commit>
Failure



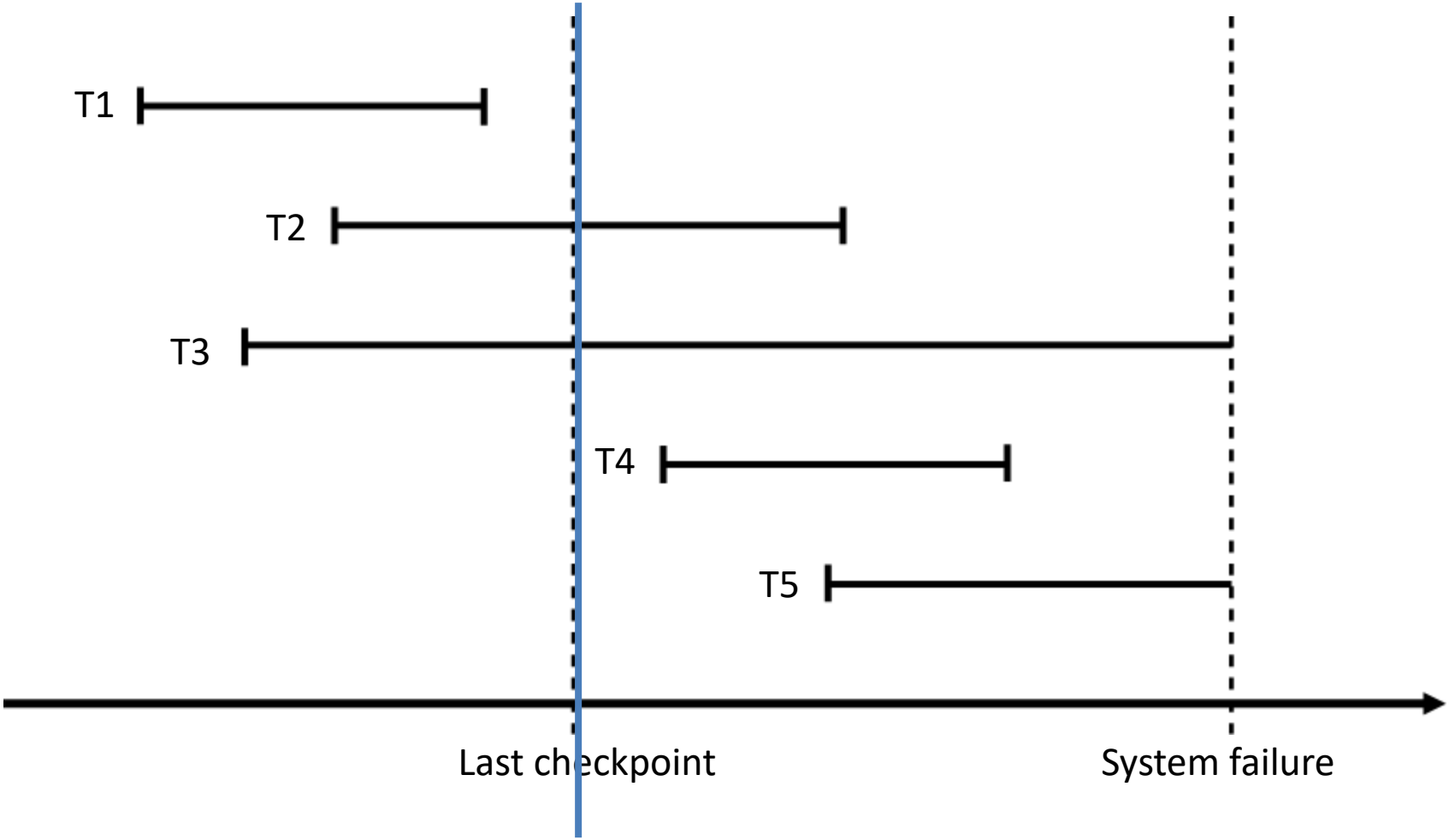
← T2, T3 is the transactions active at the time of the checkpoint

- T1 can be ignored (updates already output to disk).
- T2: the part of it after the checkpoint is re-done. ✓
- T4: completely redone. ✓
- T3, T5: completely undone. ✗

Transaction Recovery – Step1

- Computer needs an algorithm
- Initialise two lists of transactions: UNDO and REDO
  - UNDO – all transactions running at the last checkpoint
  - REDO – empty
- For every entry in the log since the last checkpoint, until the failure:
  1. If a <start> entry is found for T, Add T to UNDO
  2. If a <commit> entry is found for T, Move T From UNDO to REDO

Transaction Recovery – Step1 Example



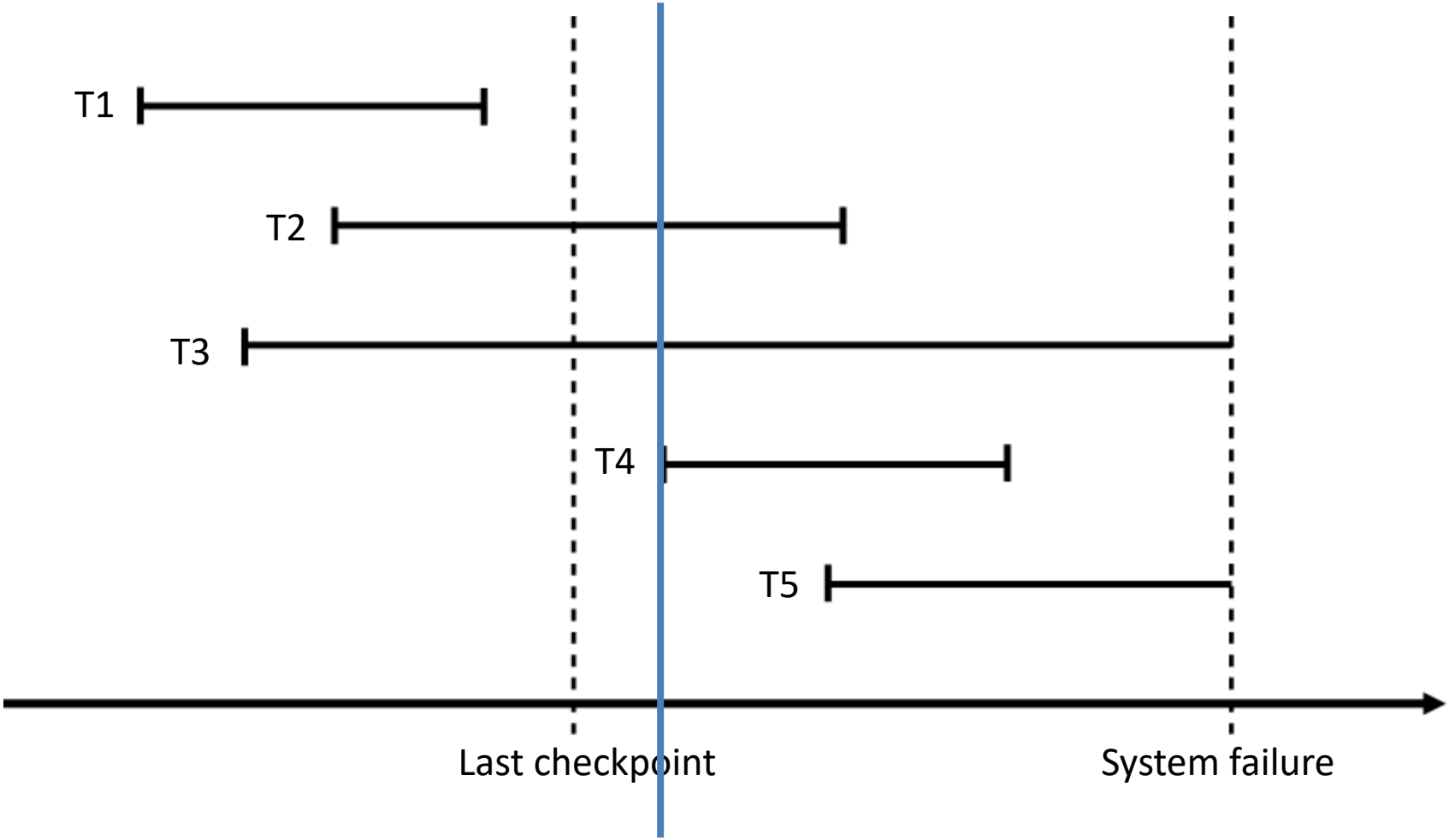
UNDO: T2, T3

REDO:

Last Checkpoint

Active transactions: T2, T3

Transaction Recovery – Step1 Example



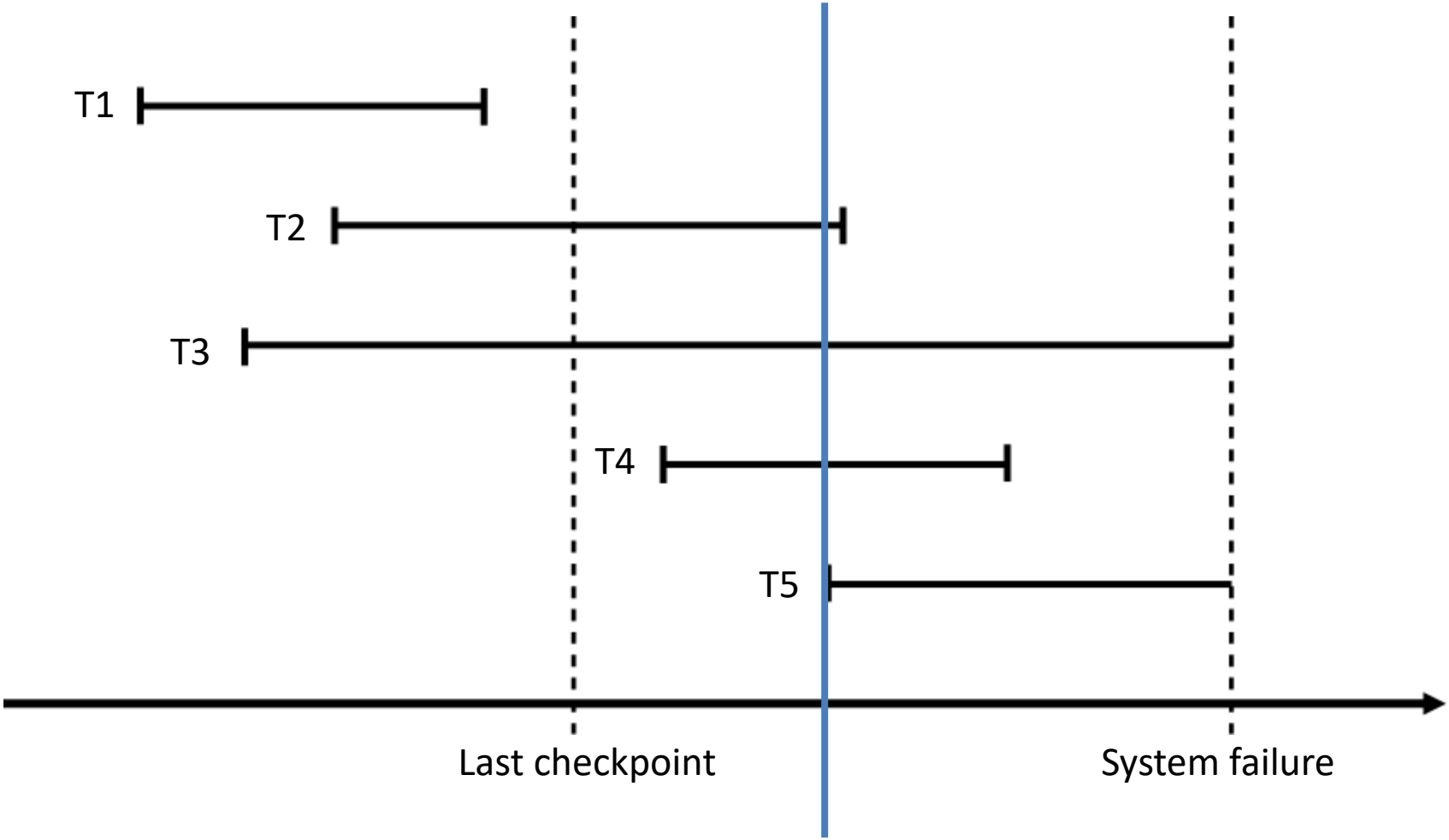
UNDO: T2, T3, T4

REDO:

T4 starts

Add T4 to UNDO

Transaction Recovery – Step1 Example



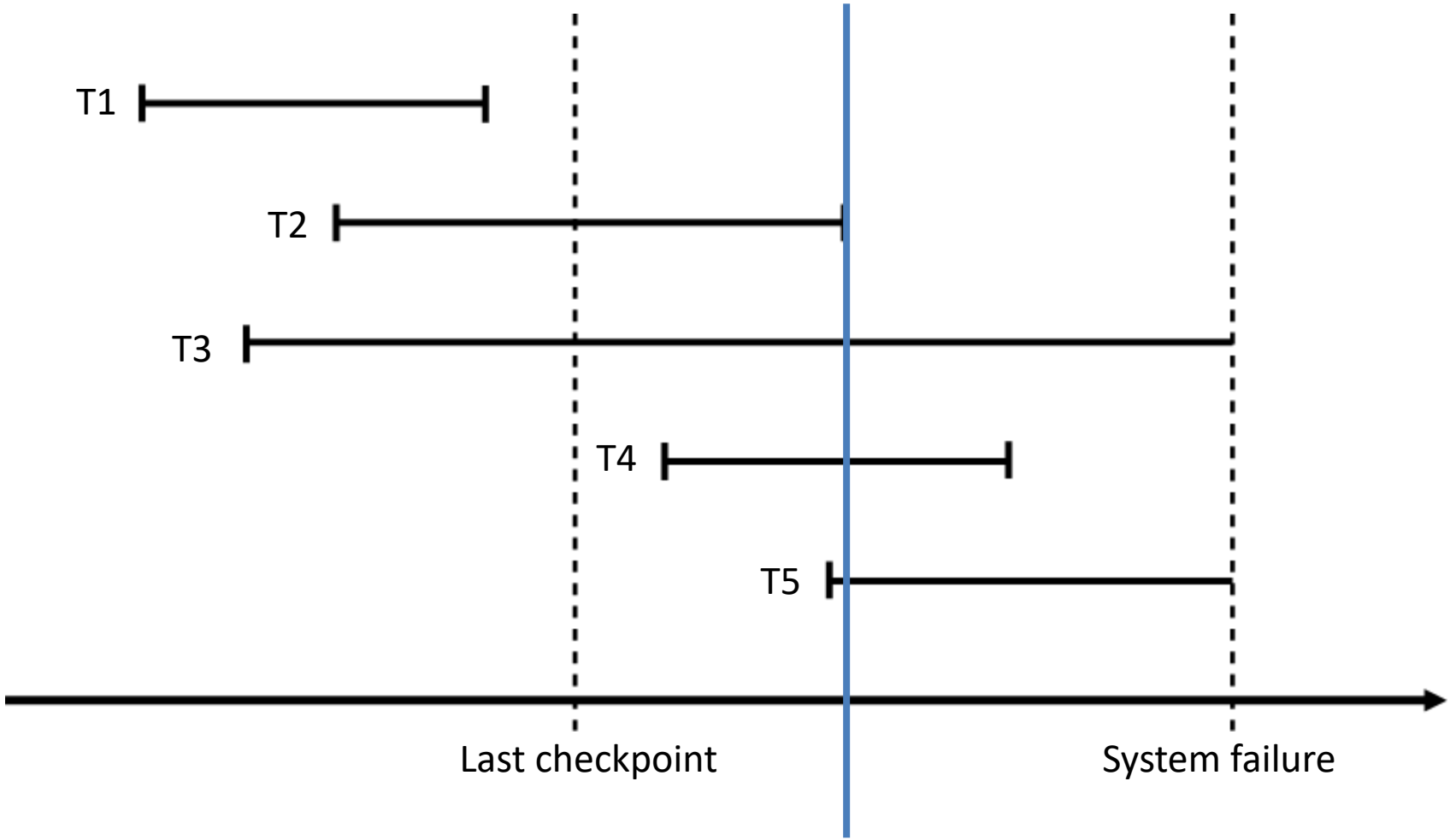
UNDO: T2, T3, T4, T5

REDO:

T5 starts

Add T5 to UNDO

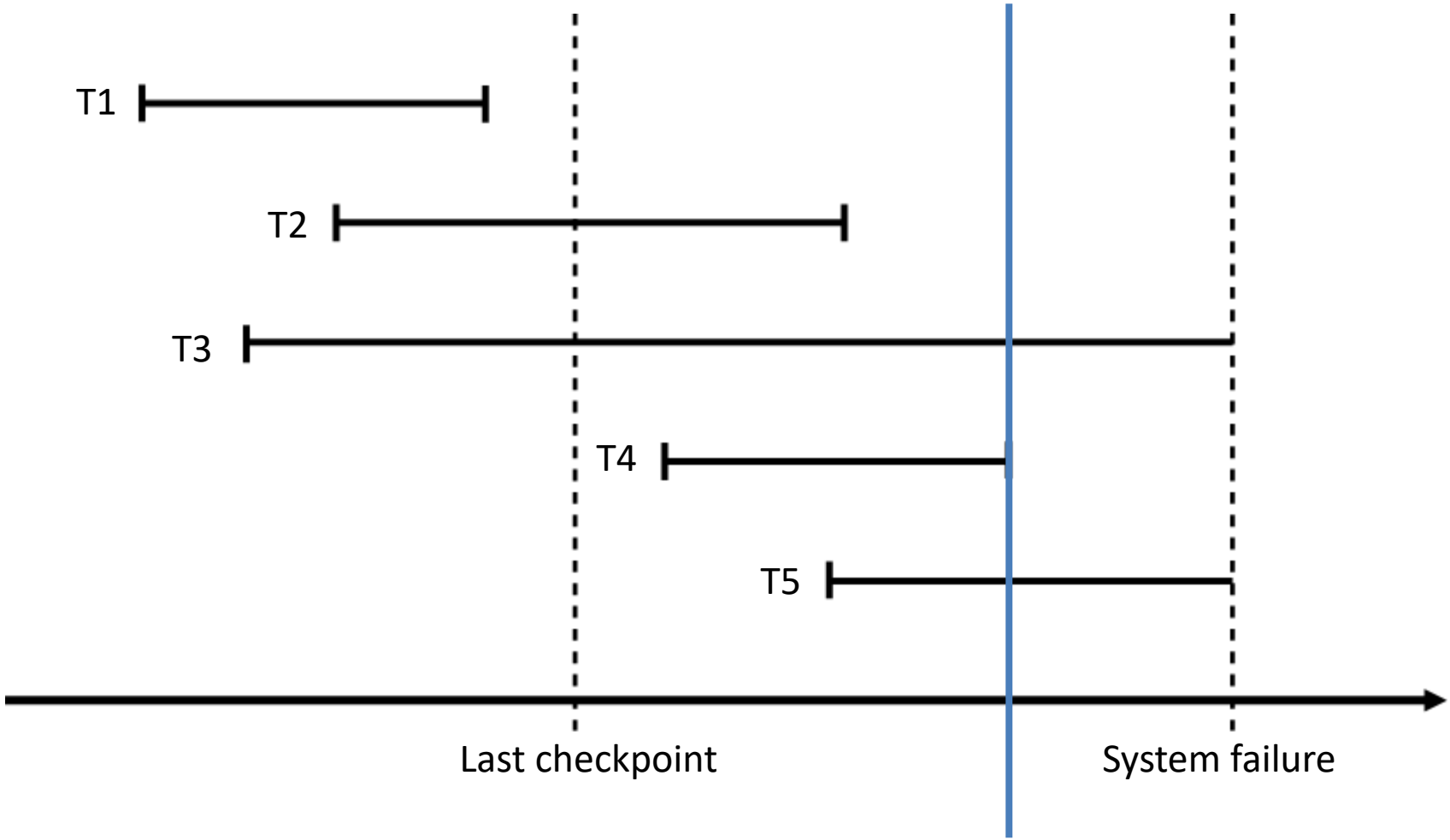
Transaction Recovery – Step1 Example



UNDO: T3, T4, T5
REDO: T2

T2 commit
Move T2 to REDO

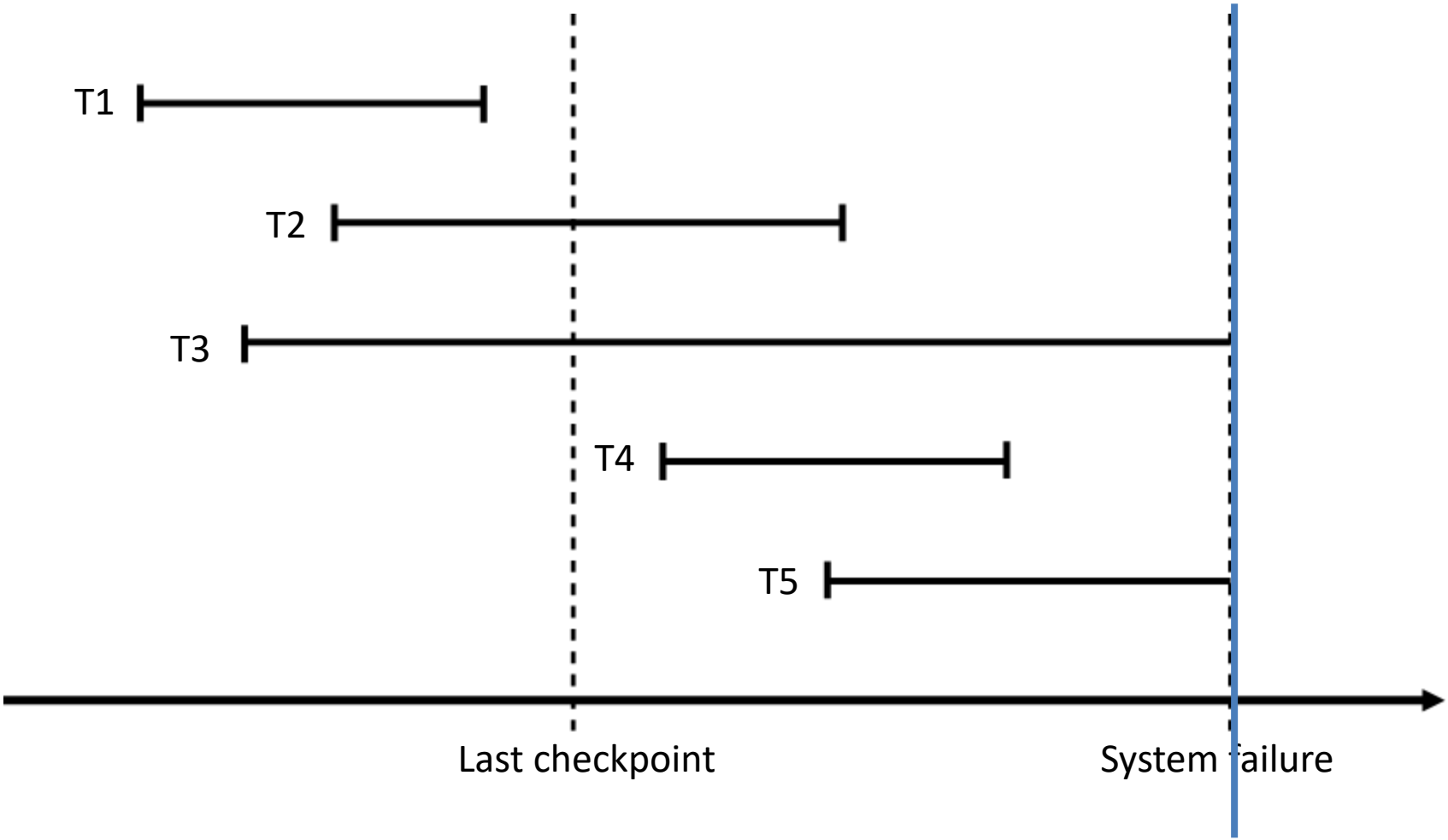
Transaction Recovery – Step1 Example



UNDO: T3, T5
REDO: T2, T4

T4 commit
Move T4 to REDO

Transaction Recovery – Step1 Example

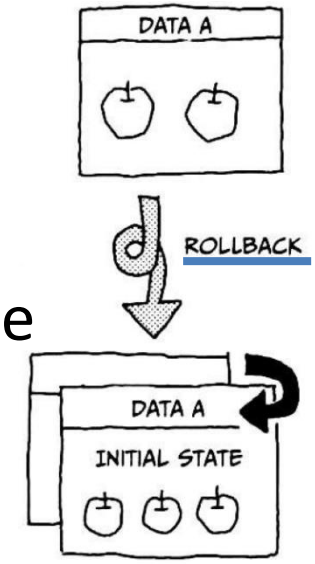


UNDO: T3, T5
REDO: T2, T4

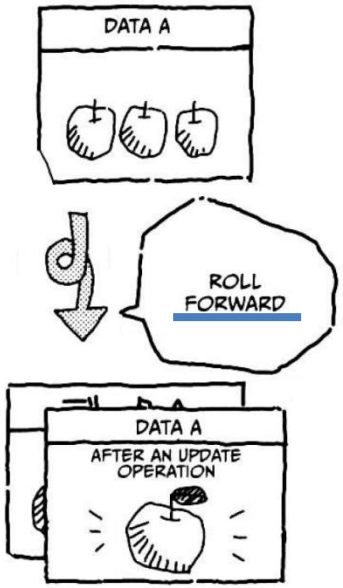
Okay, now begins step 2.

Recovery with Checkpoint – Step 2

- Backwards recovery
  - We need to undo some transactions
  - Working backwards through the log we undo every record by any transaction on the UNDO list
  - This returns the database to a consistent state



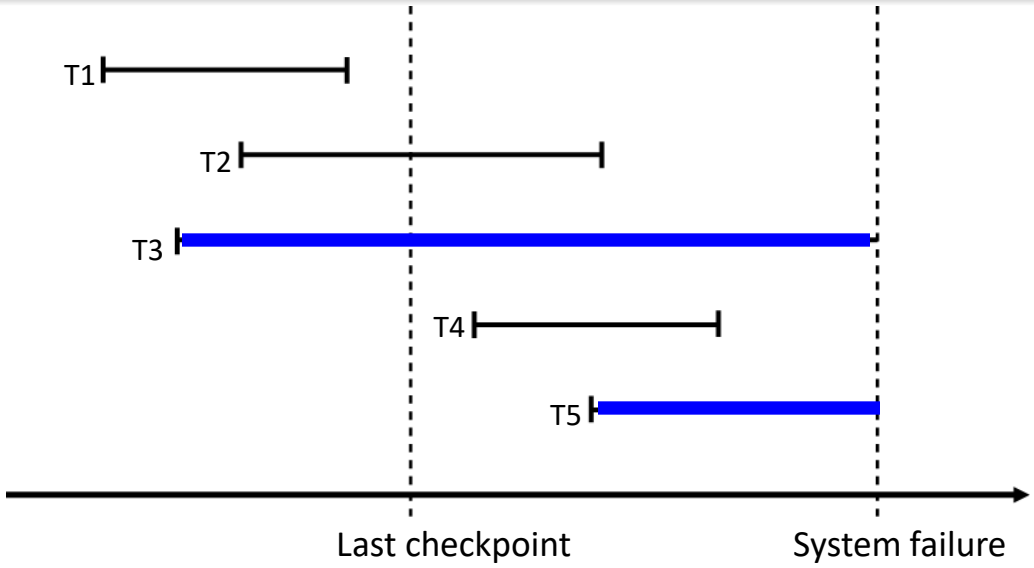
- Forwards recovery
  - Some transactions need to be redone
  - Working forwards through the log we redo any record by a transaction on the REDO list
  - This brings the database up to date



Transaction Recovery – Step2 Example

log

<T1 start>
<T1, A, 0, 10>
<T3 start>
<T3, E, 0, 12>
<T2 start>
<T2, B, 0, 10>
<T1 commit>
<checkpoint T2,T3 >
<T4 start>
<T4, A, 10, 20>
<T4, D, 0, 10>
<T5 start>
<T5, A, 20, 30>
<T2, C, 0, 10>
<T2 commit>
<T4 commit>
Failure



A=20
E=0

UNDO: T3, T5
REDO: T2, T4

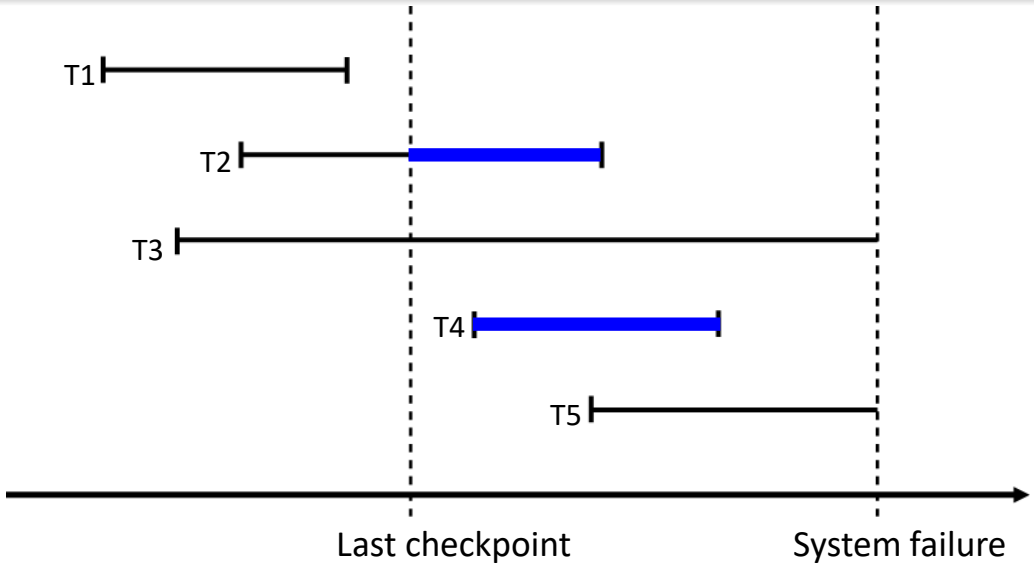
Undo: for a record <T, X, V_{old}, V_{new}>
Reverse value of X to V_{old}

- From the bottom of the log, scan backwards.
- Check each record in the log
- Undo each record of a transaction in undo-list, the blue records (e.g, T5 in UNDO)
- Until the <start> records have been seen for all transactions in undo-list.

Transaction Recovery – Step2 Example

log

<T1 start>
<T1, A, 0, 10>
<T3 start>
<T3, E, 0, 12>
<T2 start>
<T2, B, 0, 10>
<T1 commit>
<checkpoint T2,T3 >
<T4 start>
<T4, A, 10, 20>
<T4, D, 0, 10>
<T5 start>
<T5, A, 20, 30>
<T2, C, 0, 10>
<T2 commit>
<T4 commit>
Failure



A=20
D=10
C=10

UNDO: T3, T5
REDO: T2, T4

Redo: for a record <T, X, V_{old}, V_{new}>
Reassign the value V_{new} to X

- From the checkpoint, scan forwards.
- Check each record in the log
- Redo each record of a transaction in redo-list, the blue records
- Until the end of the log.

Transaction Recovery

- The above is a simplified discussion
  - Recalls data write to buffer
  - And it takes time to flush to disk
- This is not catered in the above scheme

- (Not require in syllabus) Our log scheme follows the “immediate DB modification recovery scheme”. There are many other recovery schemes (e.g. Deferred DB modification).

Media Failures

- System failures are not too severe
  - Only information since the last checkpoint is affected
  - This can be recovered from the transaction log
- Media failures (e.g. Disk failure) are more serious
  - The stored data is damaged
  - The transaction log itself may be damaged

Backups

- Backups are necessary to recover from media failure
  - The transaction log and entire database is written to secondary storage
  - Very time consuming, often requires downtime
- Backup frequency
  - Frequent enough that little information is lost
  - Not so frequent as to cause problems
  - Every night is a common compromise

Recovery from Media Failure

1. Restore the database from the last backup
2. Use the transaction log to redo any changes made since the last backup

- If the transaction log is damaged you can't do step 2
  - Store the log on a separate physical device to the database harddrive (e.g. secondary server)
  - This reduces the risk of losing both together

Side note (Out Of Syllabus)

- This is not Advanced Database System.

- We discuss one version of log-based recoveries.
- There are many more schemes out there.
- Different vendors provide different utilities to recover database from logs.

Database Recovery

theguardian

News | Sport | Comment | Culture | Business | Money | Life & style

News › Technology › Software

How NatWest's IT meltdown developed

Guardian's investigations suggest bank's problems began on Tuesday night when it updated key piece of software called CA-7

The Telegraph

Home News World Sport Finance Comment Blogs Culture Travel Life Women

Companies | Comment | Personal Finance | Economics | Markets | Festival of Business

Investing | Saving Tips | Savings | Interest Rates | Funds | Gold | Mortgages | Credit Card

HOME » FINANCE » PERSONAL FINANCE » CONSUMER TIPS » BANKING

NatWest crisis could see knock-on effects drag on for weeks

People may suffer problems with NatWest bank accounts for weeks despite the company's claims to have fixed its computer problems, the Financial Ombudsman Service has warned.

- Two servers, one for backup
  - Main, Secondary
- Tue, 19/06/2012
  - Problem identified – software upgrade (secondary server)
- Wed, Thur, 20-21/06/2012
  - Recovery – overnight long process
  - Failed!
  - Whole process re-do!

- Recovery succeeded on Fri, 22/06/2012
  - http://www.theguardian.com/technology/2012/jun/25/how-natwest-it-meltdown

Transaction, Concurrency, Recovery

theguardian

News | Sport | Comment | Culture | Business | Money | Life & style

News › Technology › Software

How NatWest's IT meltdown developed

Guardian's investigations suggest bank's problems began on Tuesday night when it updated key piece of software called CA-7

- Richard Price, a Norwich-based systems developer who has worked on banking systems that linked into NatWest's, explains: "**Banking systems are like a huge game of Jenga [the tower game played with interlaced blocks of wood]**. ***Two unrelated transactions*** might not look related now, but 500,000 transactions from now they might have a huge relation. So everything needs to be processed ***in order***." Thus Tuesday's batch must run before Wednesday's or Thursday's to avoid, for example, penalising someone who has a large sum of money leave their account on Thursday that might put them in debt but which would be covered by money arriving on Wednesday.

Concurrency

Transaction

Recovery

Summary Report 2013

Recovery

Sophisticated Batch Scheduler

… banking systems need to be completed ***in sequence***, primarily in order to implement the previous day's banking business and bring all account balances and ***transaction*** information up-to-date …

Roll back of package (note, not transaction)

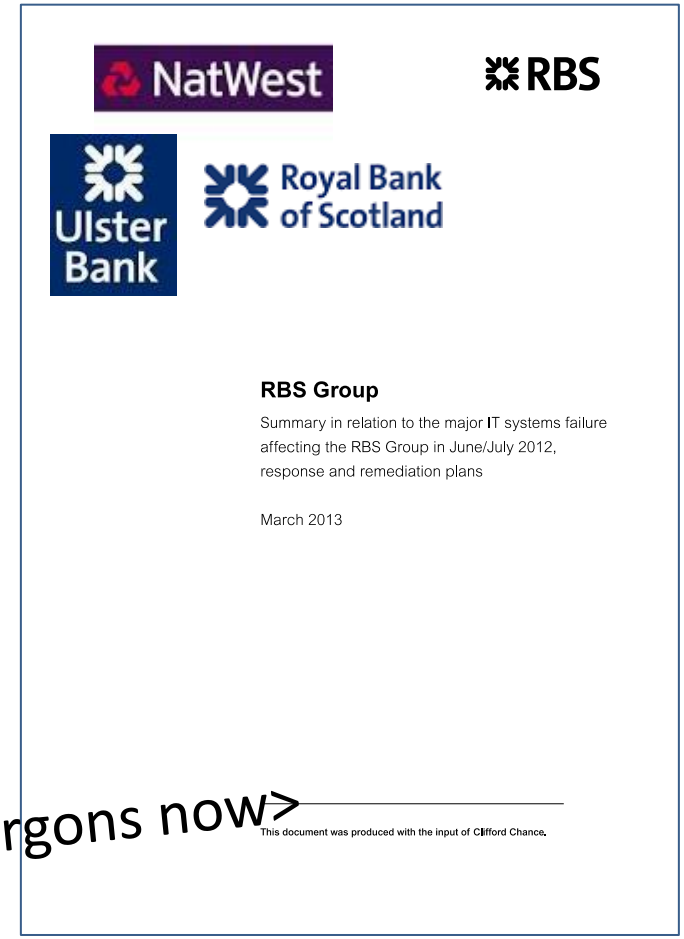
… on Tuesday 19 June 2012, it made a technical decision to reverse the upgrade… In more technical language, the decision was to ***roll back*** the batch scheduler software package to a "***stable known state***" by "backing out" the upgraded software to revert to the previous version…

<you know these jargons now>

Missing batch! ***(these are transactions)***

… on Tuesday 19 June 2012 … 11pm… some ***batch jobs were missing*** and others were ***out of sequence***.

<what? are't these logs properly stored?>



http://www.rbs.com/content/dam/rbs/Documents/News/2014/11/RBS_IT_Systems_Failure_Final.pdf

Database Recovery

RBS fined £56m over 'unacceptable' computer failure

20 November 2014 | Business

Share

Huge economical and financial impacts!

Muddiest Points (Out Of Syllabus)

- Are we going to implement all these transaction stuffs?
- No, you ***need not*** implement transactions in CS-250.
  You only ***need*** to know the theories and algorithms.
- RDBMS natively support them:

  e.g. MySQL: `START TRANSACTION`
  e.g. Oracle: `Set / Start Transaction (Read Write / Read Only) Commit / Rollback`

  Varies from vendors to vendors. Not required in CS-250

- More often in PHP:

```
try {
  $dbh->beginTransaction();
  ...
  $dbh->commit();
} catch (Exception $e) {
  $dbh->rollBack();
}
```

NOT REQUIRE IN CW2!
Don't waste time.

Server Side Programming level
(e.g. use **PDO in PHP**)

http://php.net/manual/en/pdo.begintransaction.php

Muddiest Points (Out Of Syllabus)

- Are we going to implement DFS in SQL database?

- No, you ***need not*** implement DFS in CS-250 / RDBMS.
- For ***R***DMBS, these facilities ***are already available***.
  - For example, MySQL innodb (engine) only store REDO logs,
  - but it handles all recovery ***by default*** with two files: `ib_logfile0` and `ib_logfile1`. If MySQL crashes, it ***automatically*** recover.
  - Oracle keeps separate REDO, UNDO logs however.
- *The exact setup differs between vendors.* ***We don't touch***.

- I teach and exam the ***knowledge*** on ***algorithms*** only.
- It helps you understand those concepts when ***read docs***.
- In the future, you may need to use them (e.g. MongoDB)

Out of Syllabus

Recovery-ExtraReading(OutOfSyllabus).pdf

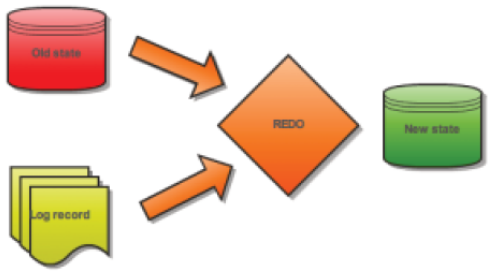
Fun MySQL fact of the day: redo logs

Over the last few weeks, we've considered how useful the MySQL binary log can be, but you may be thinking, *"if the binary log doesn't get written until commit, how does MySQL undo a transaction if it crashes?"*. And, well, if you remember back through March and April, you'll recall that transactions are handled by a table's storage engine, not MySQL. For exampl
said, let's get one thing out of the way: while the binary log c
recovery, it is not actively used in crash recovery.

Instead, InnoDB has its own write-ahead log called the "redo changes that have been made to a page and/or record. By d
files, `ib_logfile0` and `ib_logfile1`, which InnoDB writ
the changes in the redo log are forward-only and contain no
the database.

Why log?

Since InnoDB tries to keep the working set in memory (InnoDB Buffer Pool), ther
be flushed to disk. So in the event of volatile memory failure or during a system re
in the database in (D)urable memory and that each transaction is (A)tomic.



What is actually logged?

<space id, page no, operation code, data>

Note: Gary is NOT an expert in DBMS.

Industry is using these concepts and knowledge!

Fun MySQL fact of the day: everything is two-phase

Yesterday, we started thinking about InnoDB's redo log, and I left you with a thought to consider: how can MySQL keep multiple storage engines' redo logs in sync with the binary log. Today, we'll consider how MySQL does it.

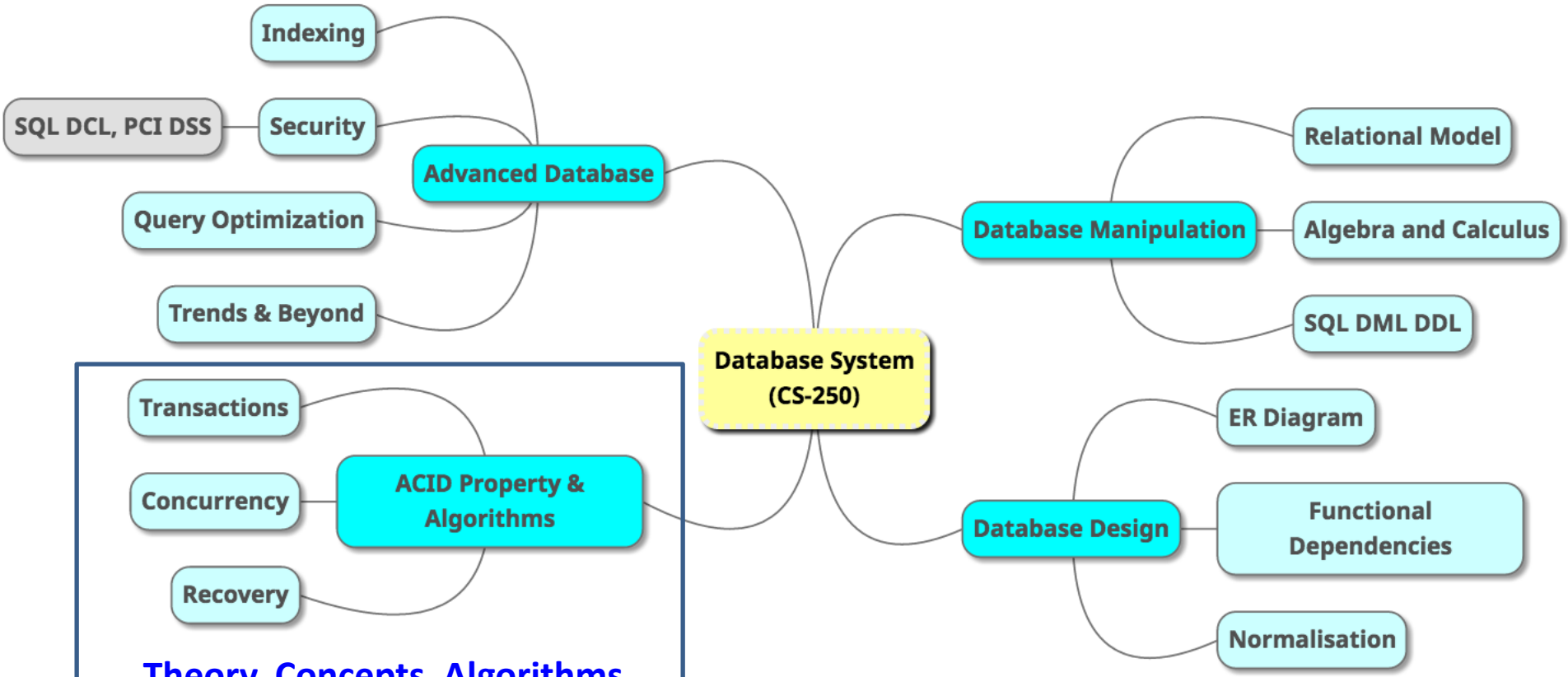
You may have heard the term, "two-phase commit" (or 2PC) before, but if not, it is an algorithm used to coordinate multiple systems participating in a distributed, or global, transaction. The most common standard for two-phase commits is the X/Open XA specification, often just called "XA" (for
rchitecture). Today, we will consider the XA standard only at the highest level, but if you're
person that likes to maximise fun, you can read the full technical standard. If you *have*
erm "two-phase commit" before, you probably heard it used with distaste. Maybe
along the lines of "two-phase commit is [bad]" or "two-phase commit doesn't scale". And,
'm not going to argue too strongly about that, you may be surprised to learn that MySQL
ernally for all transactions to ensure that all storage engines participating in a transaction
e properties of ACID. Let's see how

14.6 InnoDB On-Disk Structures

- 14.6.1 Tables
- 14.6.2 Indexes
- 14.6.3 Tablespaces
- 14.6.4 InnoDB Data Dictionary
- 14.6.5 Doublewrite Buffer
- 14.6.6 Redo Log
- 14.6.7 Undo Logs

Logs are inside the InnoDB engine

Overview


Indexing
SQL DCL, PCI DSS
Security
Advanced Database
Query Optimization
Trends & Beyond
Relational Model
Database Manipulation
Algebra and Calculus
SQL DML DDL
Database System
(CS-250)
Transactions
Concurrency
ACID Property &
Algorithms
Recovery
ER Diagram
Database Design
Functional
Dependencies
Normalisation
Theory, Concepts, Algorithms
3 Inter-related Components
to uphold ACID properties

Past student

- MongoDB is part of NOSQL movement (more in wk10).
- MongoDB is not “fully natively ACID-compliant”.

- Bassam (2019): “When I self-learn MongoDB, all these concepts:
  - transactions
  - concurrency
  - recovery
  keep popping up.“
- “These lectures are very important! Tell your students.”