Recovery

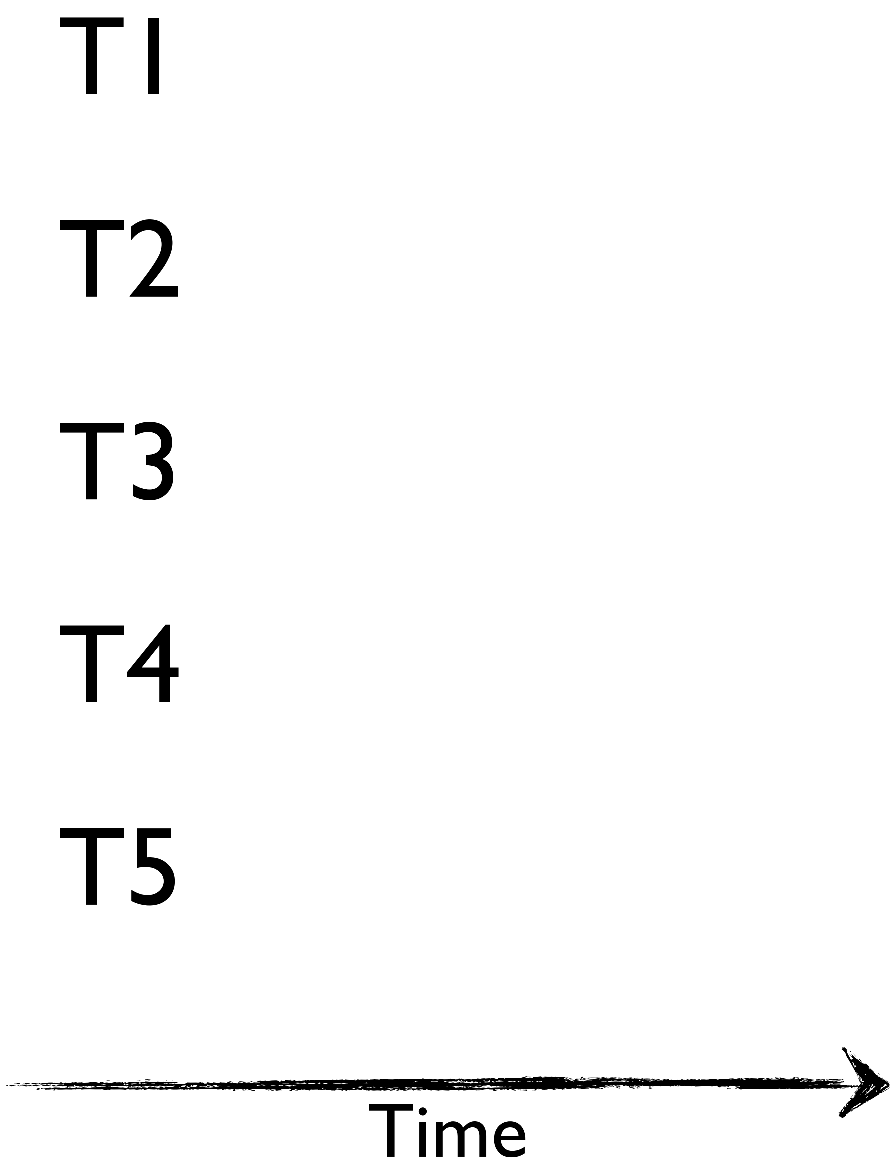
R&G Chapter 18

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

A.C.I.D.

- **Atomicity**: All actions in a transaction happen, or none happen.
- **Consistency**: If the transaction maintains consistency, and the DB starts consistent, then the database ends consistent.
- **Isolation**: The execution of one transaction is isolated from all other transactions.
- **Durability**: If a transaction commits, its effects persist.

Motivation



T1

T2

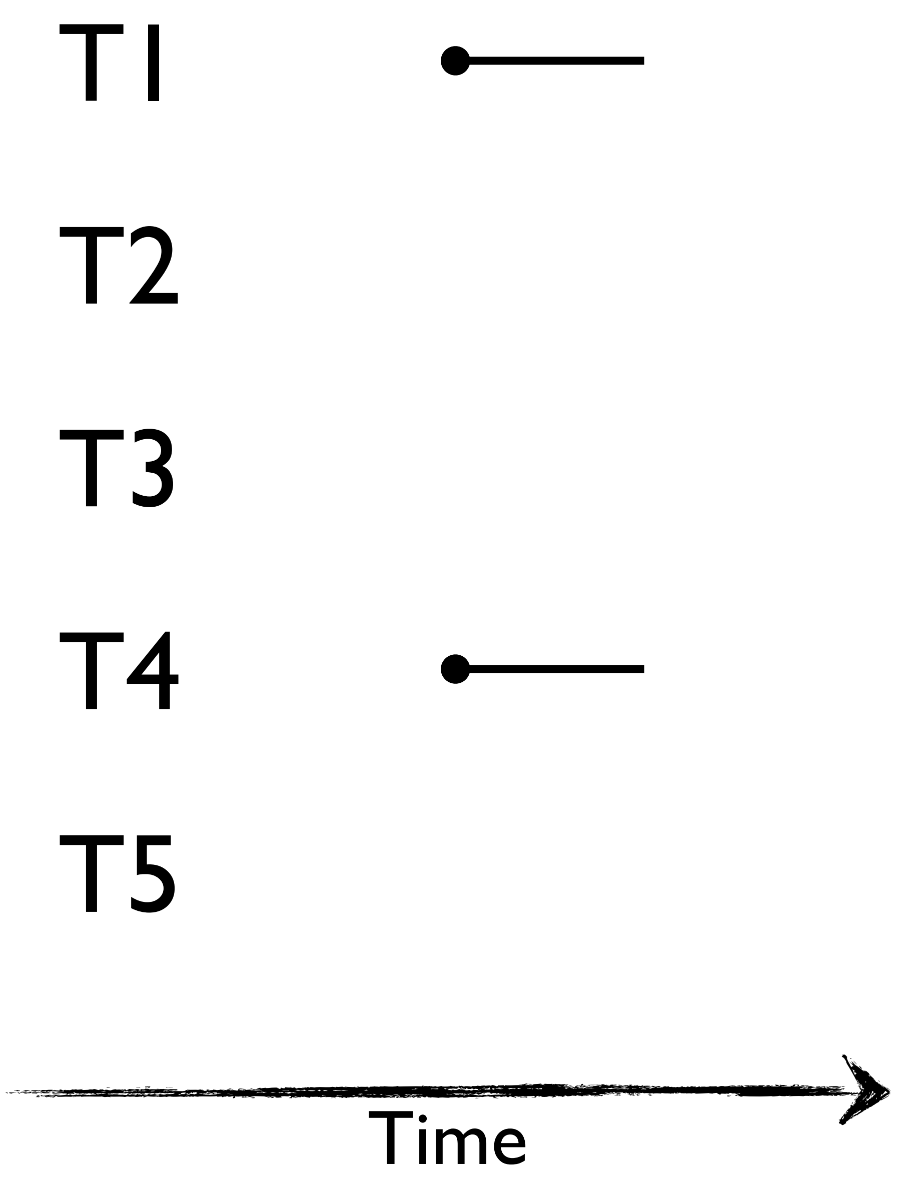
T3

T4

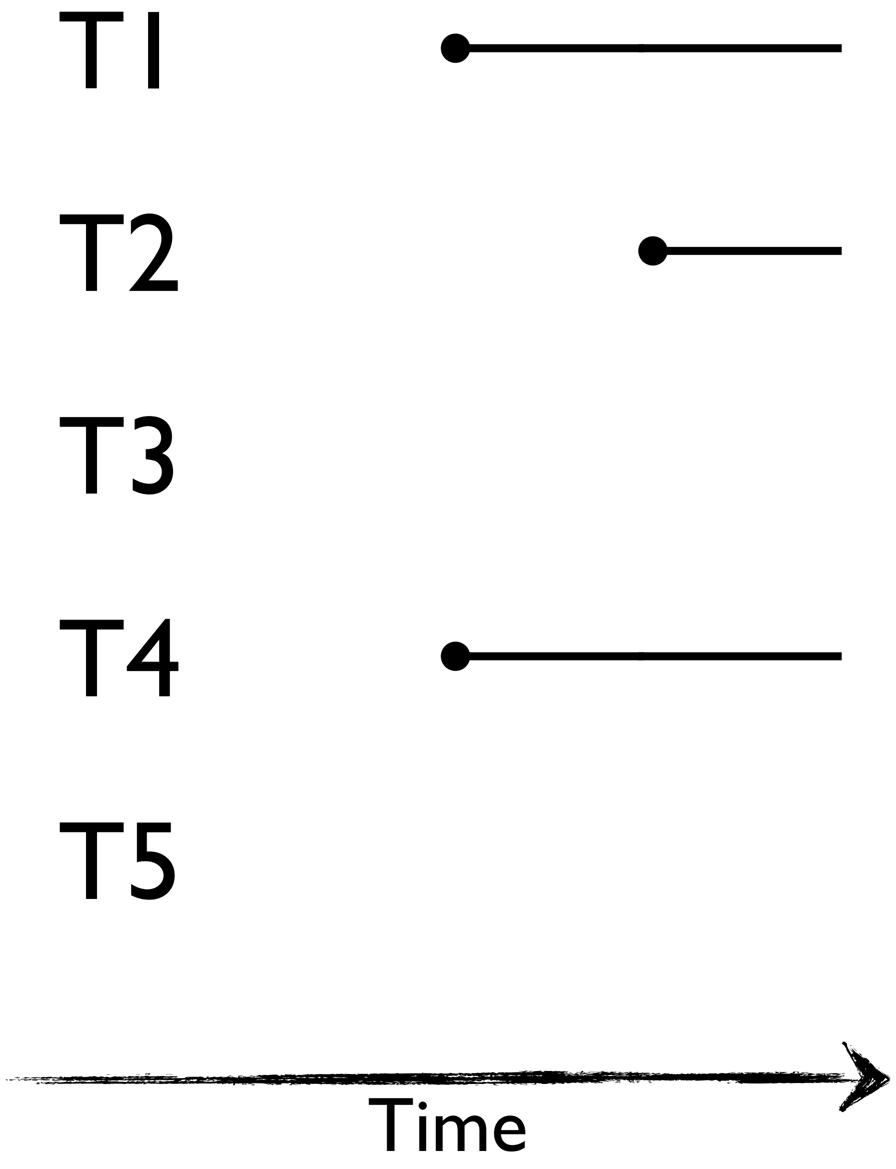
T5

Time

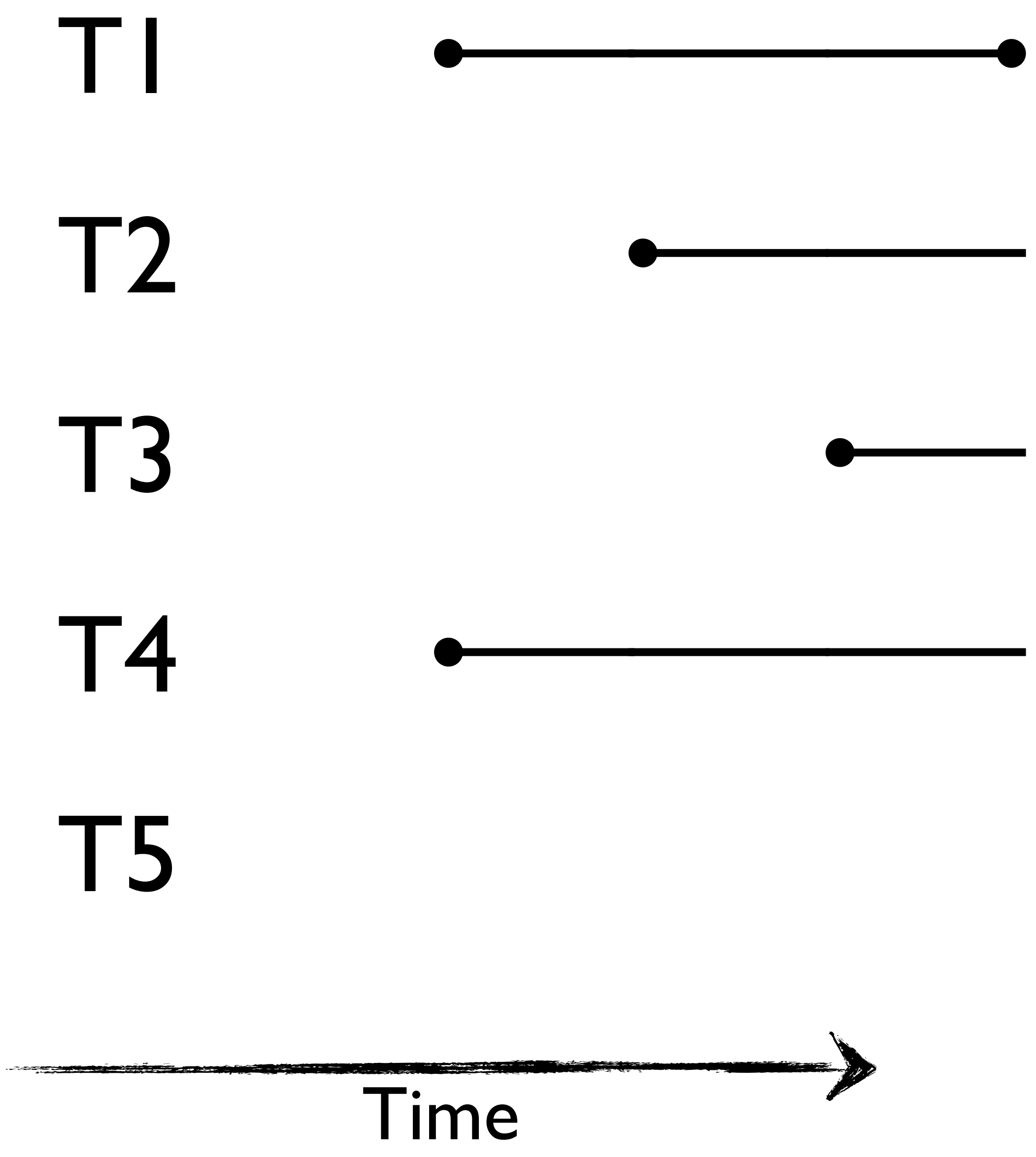
Motivation



Motivation



Motivation



T1

T2

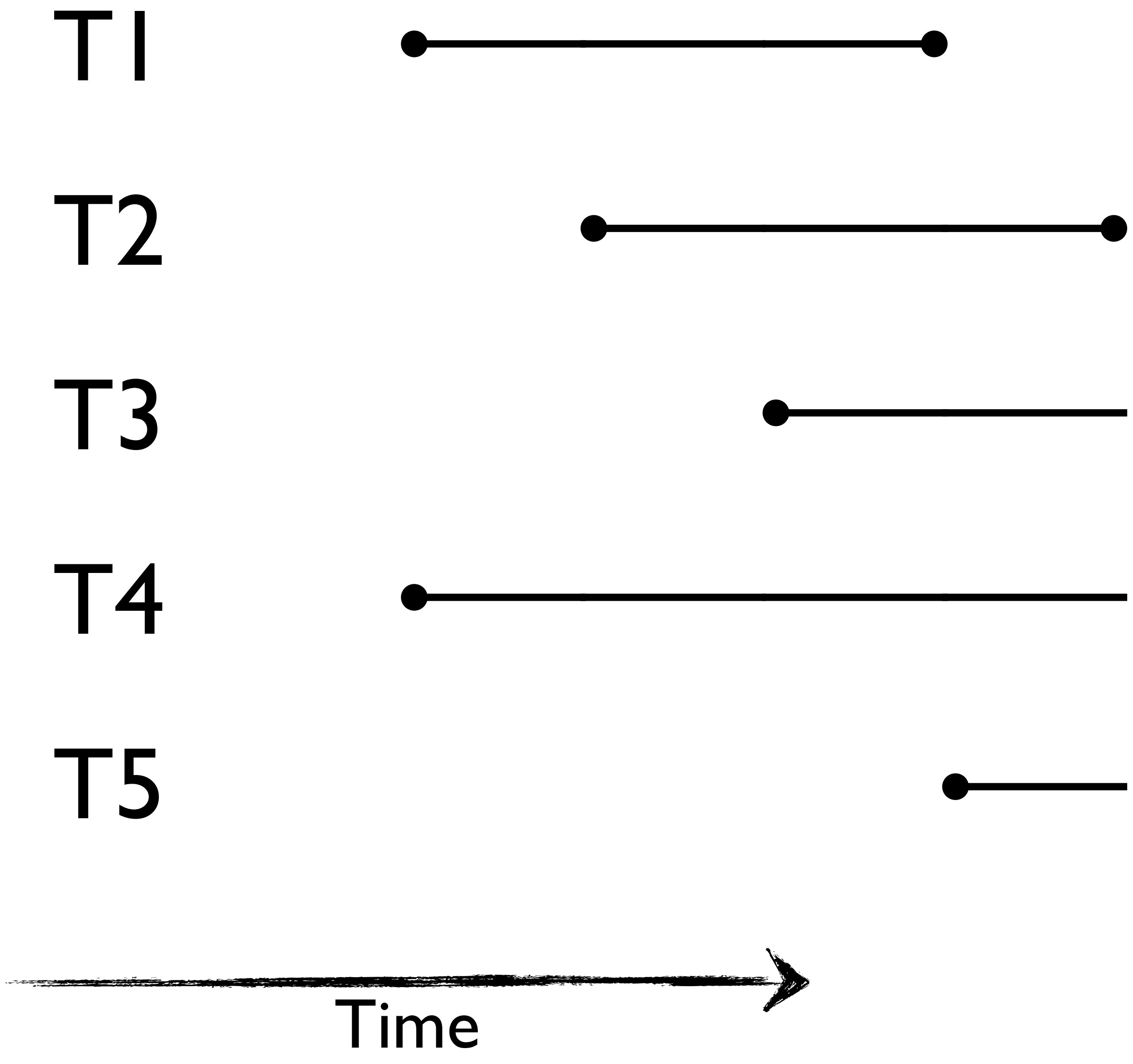
T3

T4

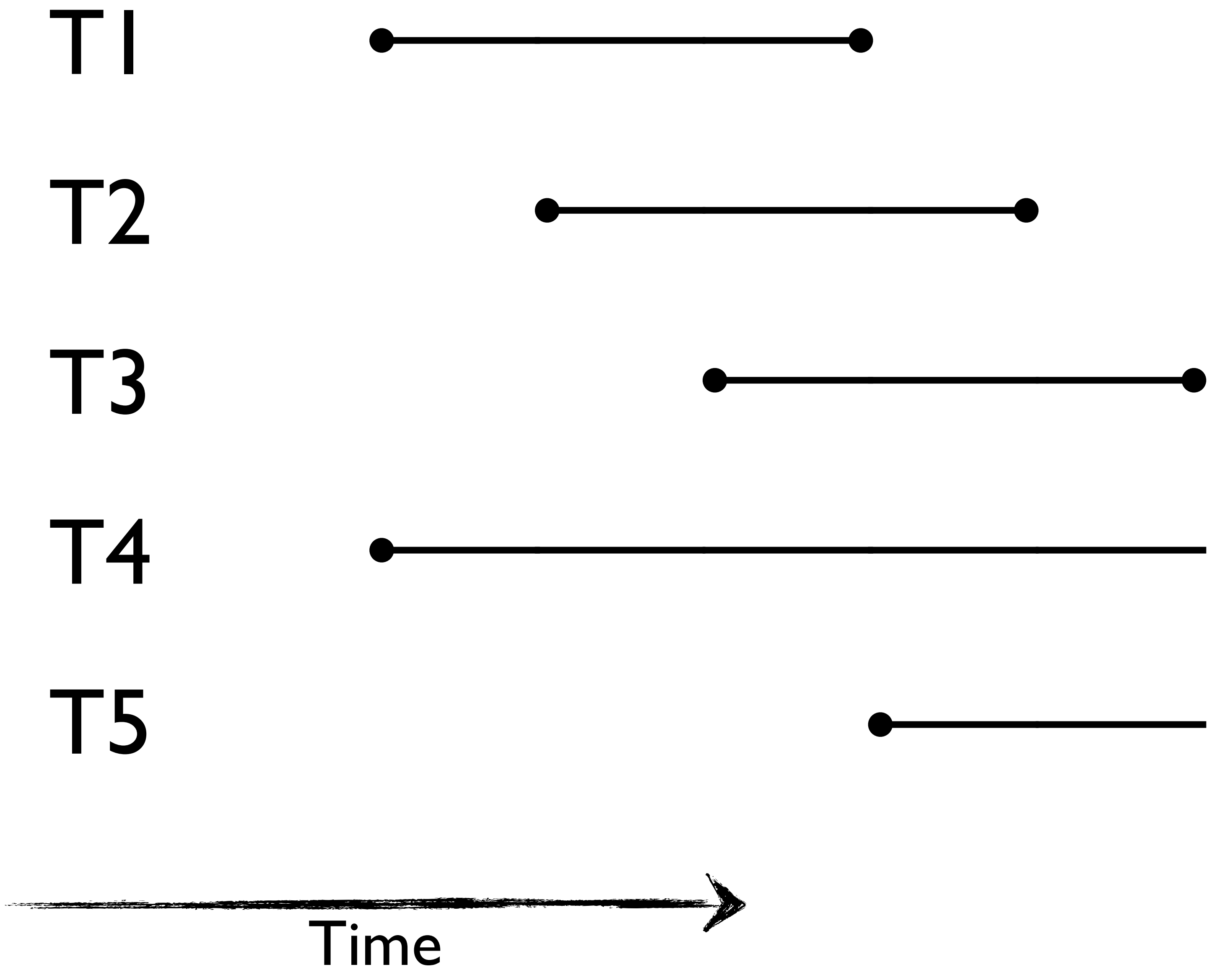
T5

Time

Motivation



Motivation



T1

T2

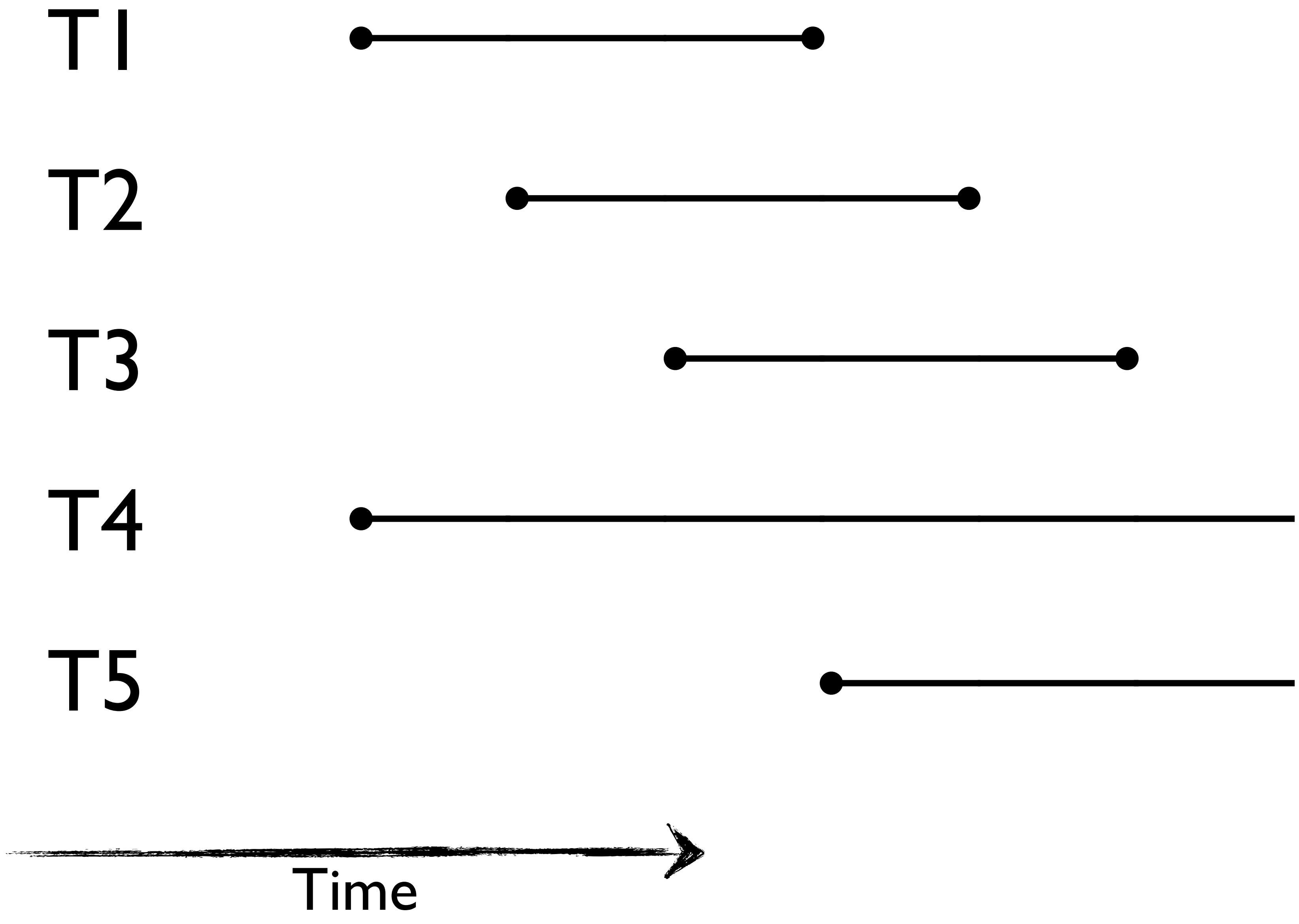
T3

T4

T5

Time

Motivation


T1

T2

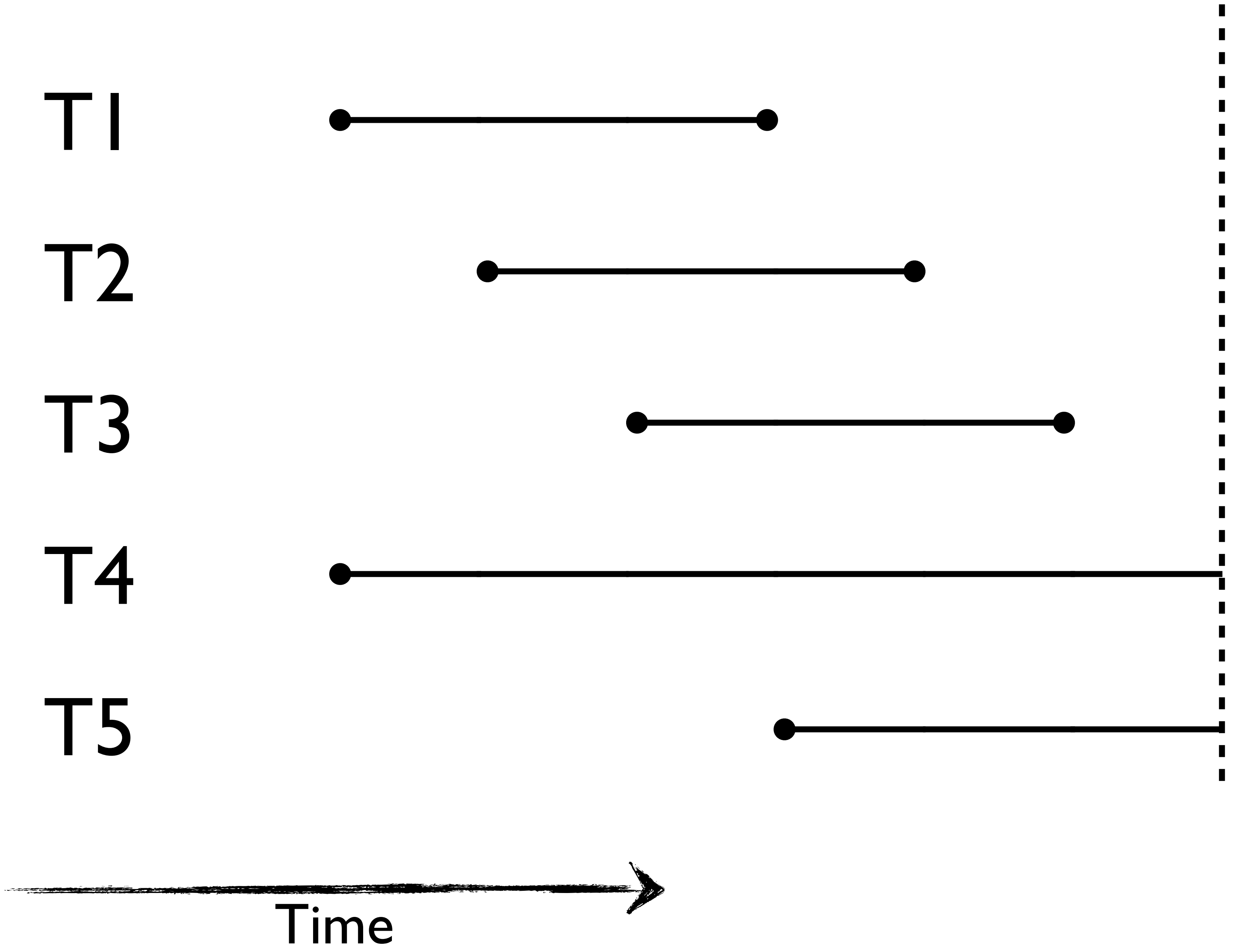
T3

T4

T5

Time

Motivation



T1

T2

T3

T4

T5

Time

Image copyright: Wikimedia Commons

Motivation

Committed Transactions.
These should be present when the DB restarts.

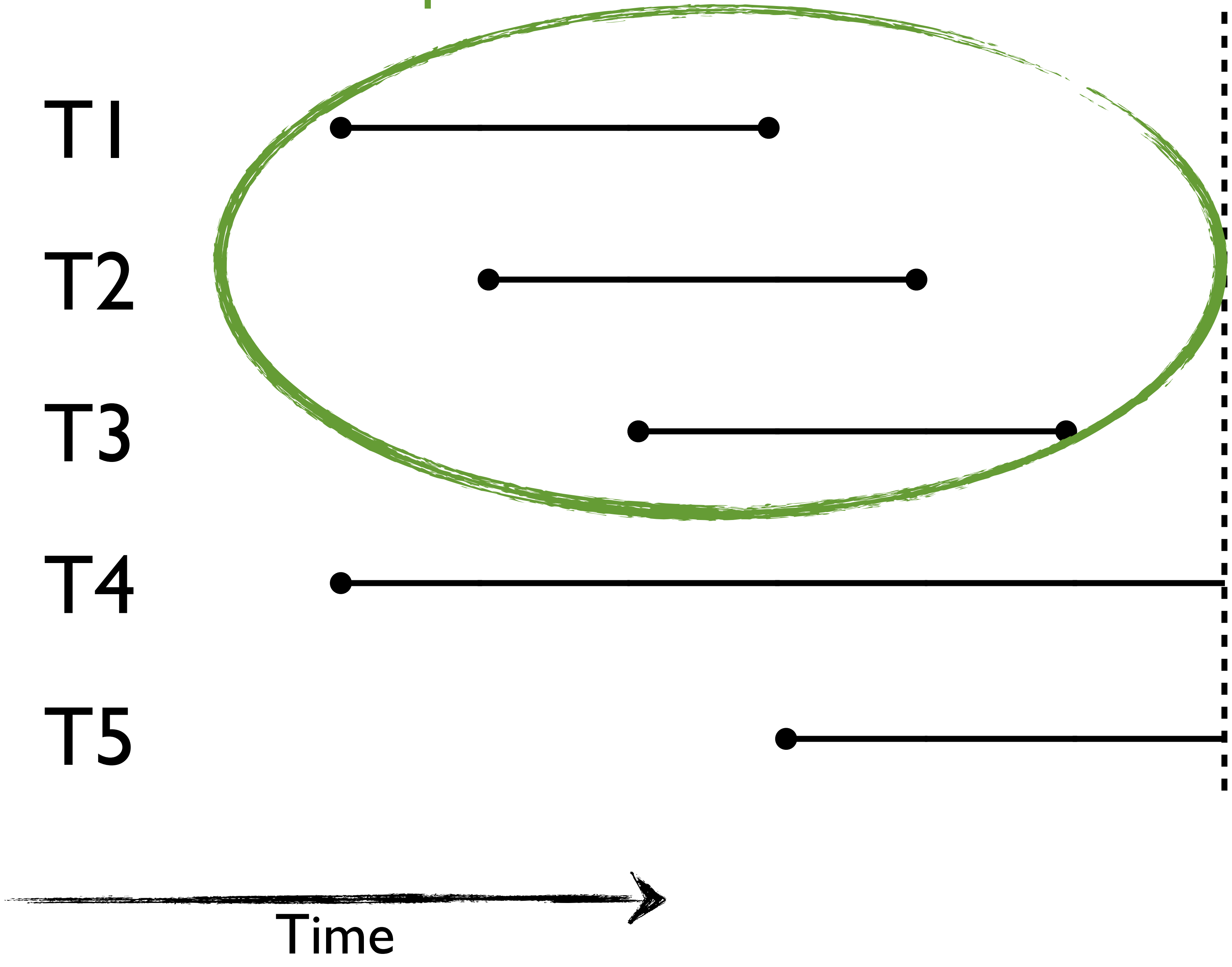
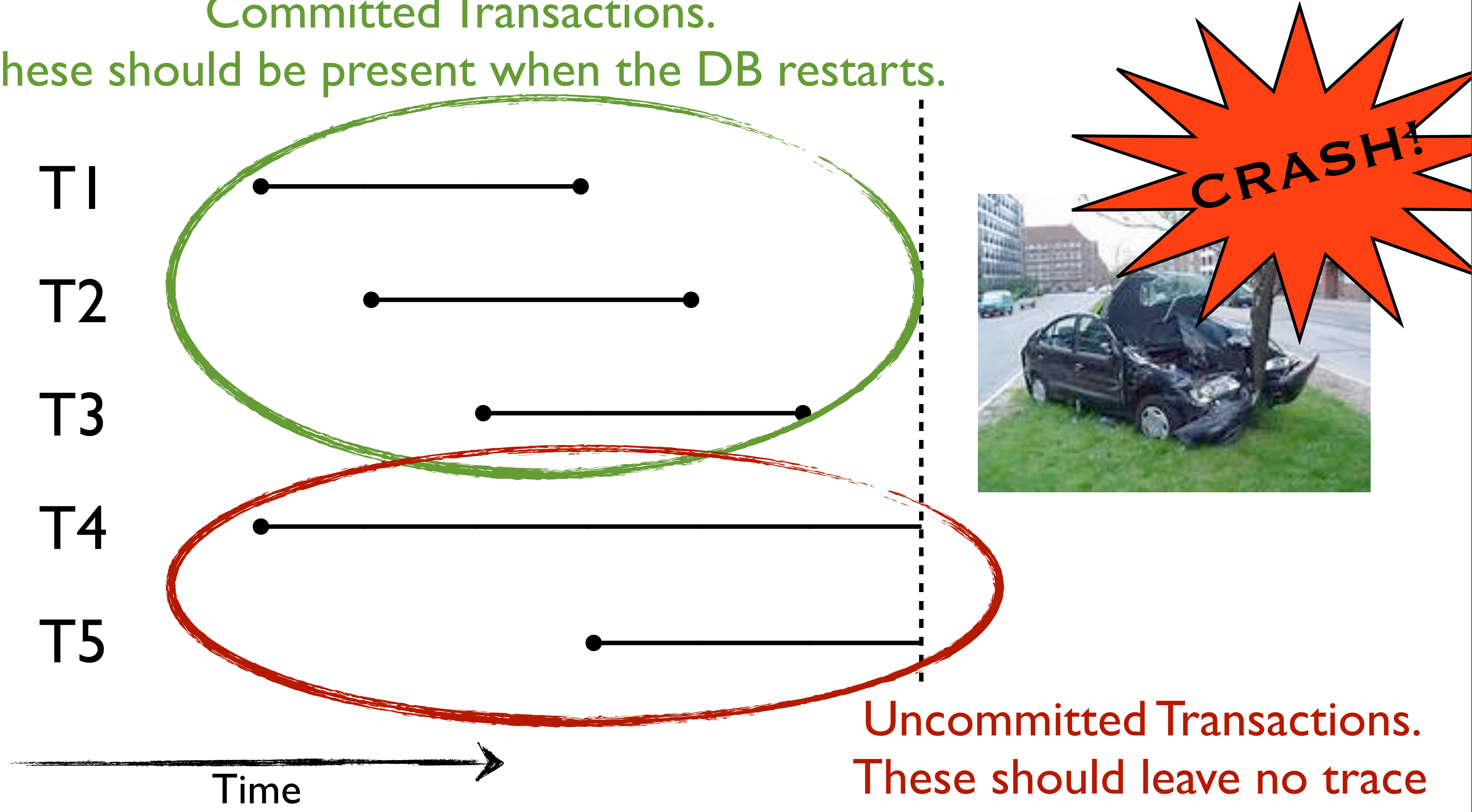


Image copyright: Wikimedia Commons

Motivation



Committed Transactions.
These should be present when the DB restarts.

Uncommitted Transactions.
These should leave no trace

Image copyright: Wikimedia Commons

Goals

- Support recovery from crashes
- Ensure that the effects of committed transactions are recovered fully.
- Ensure that the effects of uncommitted transactions are discarded fully.
- We want a simple scheme to guarantee atomicity and durability.

Assumptions

- Concurrency control is in effect
  - Strict 2-Phase Locking Specifically
- Updates are happening "in place"
  - Updates are applied directly to the (single) on-disk copy of the data.

The Buffer Pool

- An intermediary between memory and disk.
- Challenges:
    - **Force**: Every write goes directly to disk?
        - Provides durability.
        - … but results in poor response time.

The Buffer Pool

- An intermediary between memory and disk.
- Challenges:
    - **Steal**: Allow buffer-pool frames to be stolen from uncommitted xacts?
        - If stealing not allowed, poor throughput.
        - If stealing allowed, atomicity not preserved.

The Buffer Pool

	No Steal	Steal
Force	Trivial!	
No Force		Desired

Steal

- Enforcing **atomicity** is hard
  - To steal frame F, the current page P in F is written to disk.
  - Some transaction T has written to P.
  - What if T aborts?
  - Need to be able to UNDO T's writes to P.

No Force

- Enforcing **durability** is hard
  - What if the system crashes before a modified page is written to disk?
  - What if the system crashes while writing several pages to disk?
  - Need to be able to REDO writes that haven’t been finalized yet.

Basic Idea: Logging

- Record all information needed to perform REDO and UNDO operations in a log.
  - Log accesses are all sequential writes.
  - Can use a separate disk for logs.
  - Minimal info (delta) is written to log, so can fit many updates in one page.

Example Log Record

- Transaction ID
- Page ID
- Offset
- Length
- Old Data
- New Data
- (Some extra control info that we'll see shortly)

Write-Ahead Logging

- The write-ahead logging protocol
  - Force the log record for an update before the corresponding data page is written.
    - Guarantees Atomicity (REDO).
  - Force all log records for a transaction before the transaction commits.
    - Guarantees Durability (UNDO).

Write-Ahead Logging

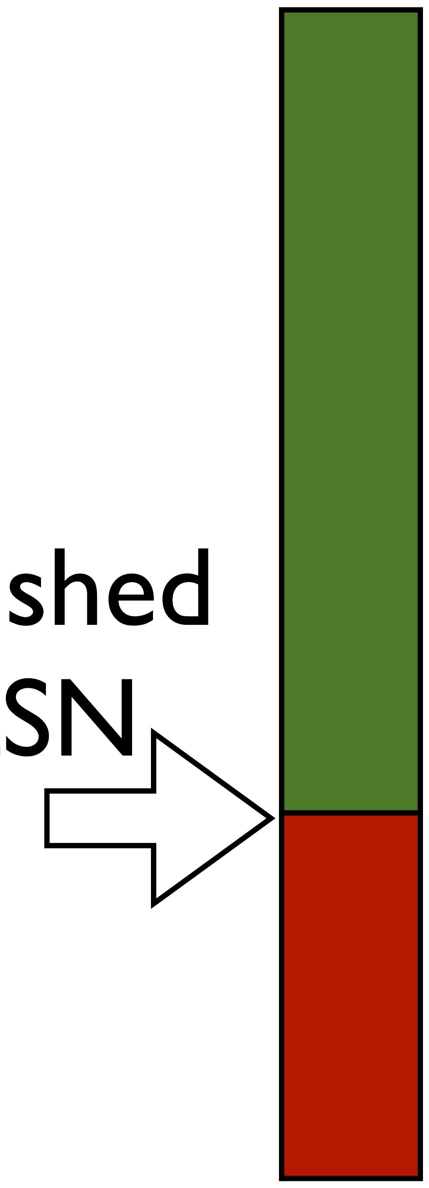
- The write-ahead logging protocol
  - Force the log record for an update before the corresponding data page is written.
    - Guarantees Atomicity (REDO).
  - Force all log records for a transaction before the transaction commits.
    - Guarantees Durability (UNDO).

How would recovery be implemented?

WAL & The Log

- Each log record has a unique Log Sequence Number (LSN)
- Each data page contains a Page LSN.
- System keeps (in memory) the max Flushed LSN (max LSN forced to disk)
- Before a page is written, ensure that **Page LSN ≤ Flushed LSN**

Flushed Log Records



Flushed LSN

Tail (Ram Only)

Log Records

- Record Types
  - **Update**
  - **Commit**
  - **Abort**
  - **End** (of commit/abort)
  - CLRs

Log Record Fields

```
Prev LSN
XID
Record Type
Page ID
Length
Offset
Before-Image
After-Image
```

Only needed for **update**: Page ID, Length, Offset, Before-Image, After-Image

Other Log-Related State

- **Transaction Table**
  - One entry per active transaction
  - Contains XID, Transaction Status, LastLSN
- **Dirty Page Table**
  - One entry per dirty page in the buffer pool.
  - Contains **RecLSN**: LSN of the log record that first caused the page to be dirty.

Transaction Execution

- Normal execution:
  - A Series of reads and writes followed by a commit or an abort.
  - Strict 2-Phase Locking
  - Steal, No-Force buffer management with write-ahead logging.

The Big Picture

The Log

Log Records

- `Prev LSN`
- `XID`
- `Record Type`
- `Page ID`
- `Length`
- `Offset`
- `Before-Image`
- `After-Image`

The DB (Disk)

Data Pages

(each with a PageLSN)

Master Record

The DB (Ram)

Transaction Table

- `XID`
- `LastLSN`
- `Status`

Dirty Page Table

- `RecLSN`

Flushed LSN

Simple Transaction Abort (no crash involved)

“Play Back” the log in reverse order, UNDOing records

Transaction Table

Log

image credit: openclipart.org

Simple Transaction Abort (no crash involved)

“Play Back” the log in reverse order, UNDOing records

Transaction Table

ABORT [XID]

Log

(necessary for crash recovery)

image credit: openclipart.org

Simple Transaction Abort (no crash involved)

"Play Back" the log in reverse order, UNDOing records

Transaction Table

XID, LastLSN

ABORT [XID]

Log

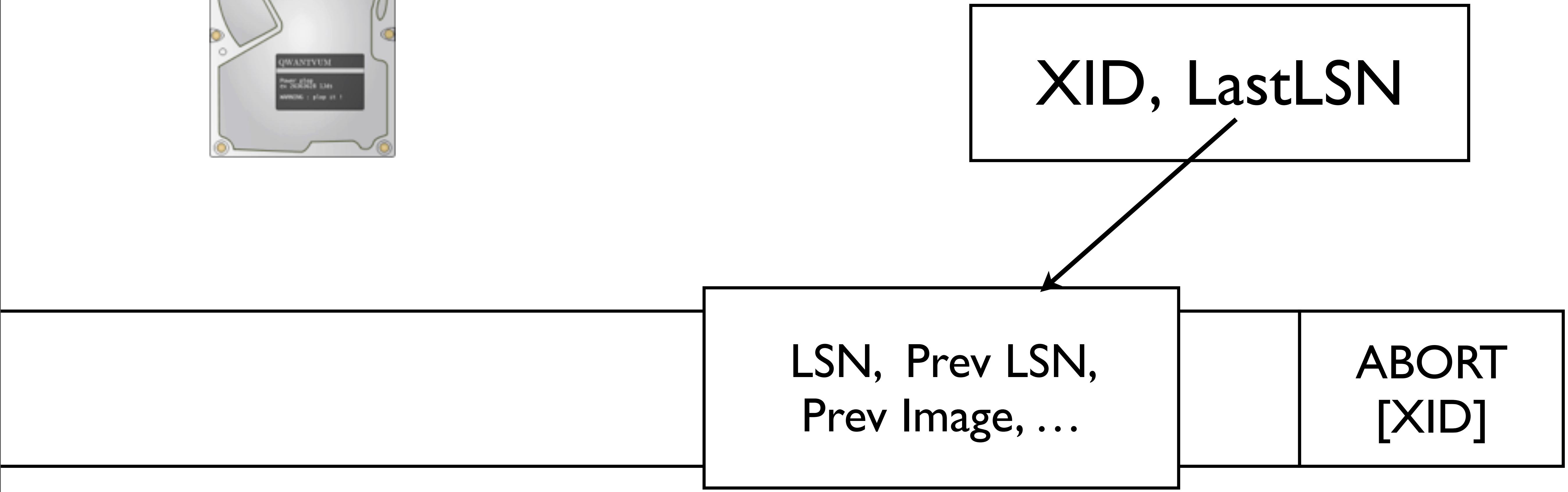
(necessary for crash recovery)

image credit: openclipart.org

Simple Transaction Abort (no crash involved)

“Play Back” the log in reverse order, UNDOing records

Transaction Table



XID, LastLSN

LSN, Prev LSN, Prev Image, …

ABORT [XID]

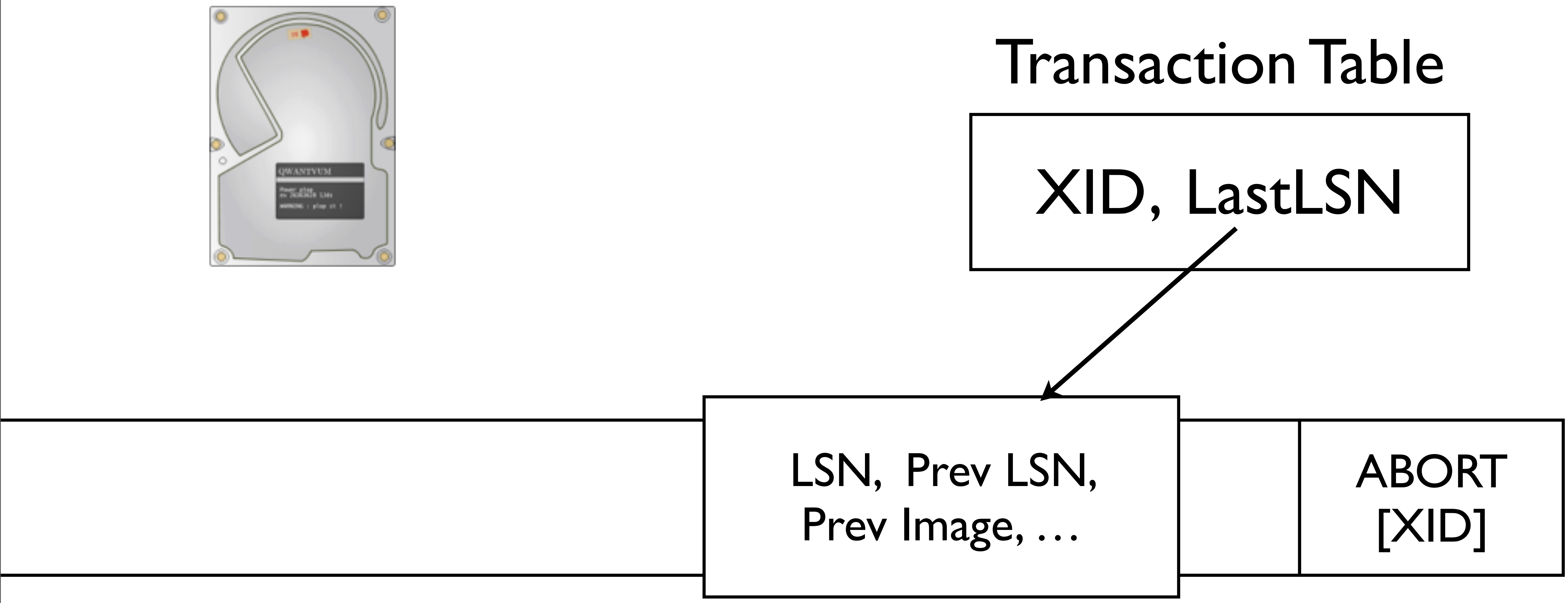
Log

(necessary for crash recovery)

image credit: openclipart.org

Simple Transaction Abort (no crash involved)

"Play Back" the log in reverse order, UNDOing records



Transaction Table

XID, LastLSN

LSN, Prev LSN, Prev Image, …

ABORT [XID]

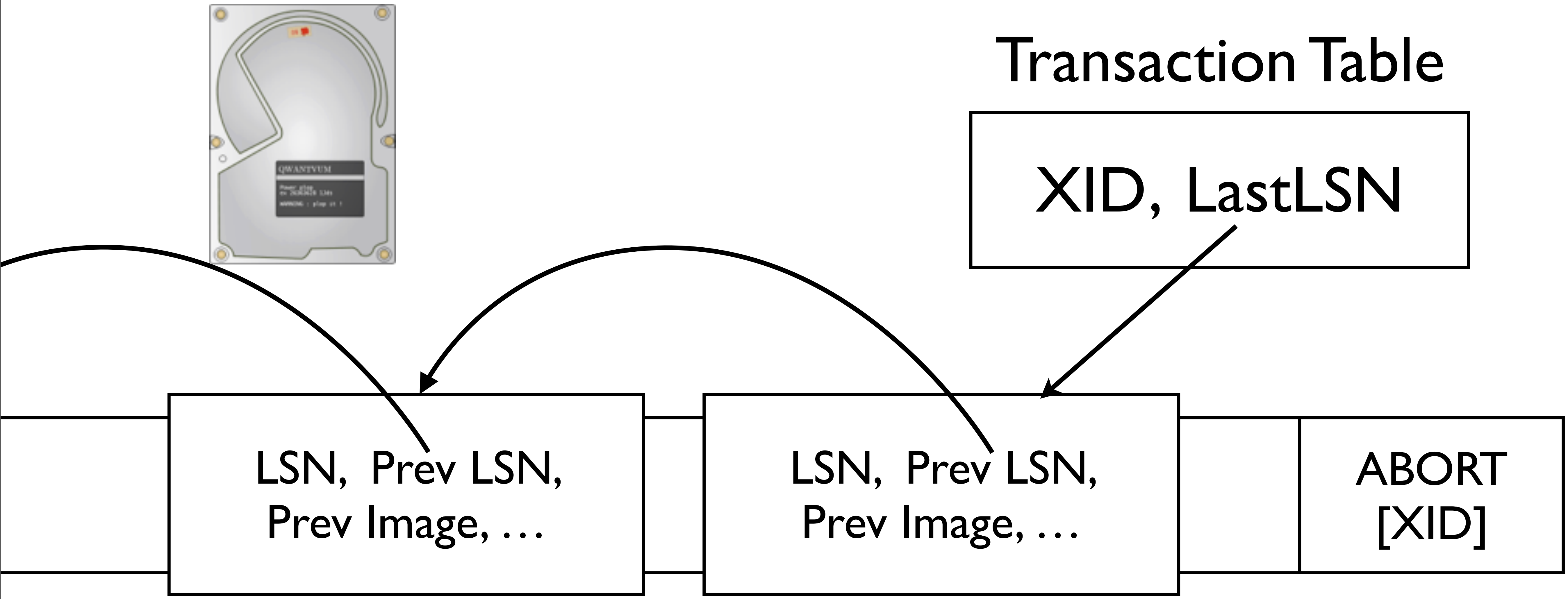
(necessary for crash recovery)

Log

image credit: openclipart.org

Simple Transaction Abort (no crash involved)

"Play Back" the log in reverse order, UNDOing records



Transaction Table

XID, LastLSN

LSN, Prev LSN, Prev Image, …

LSN, Prev LSN, Prev Image, …

ABORT [XID]

Log

(necessary for crash recovery)

image credit: openclipart.org

Simple Transaction Abort (supporting crash recovery)

- Before restoring the old value of a page, write a Compensation Log Record (CLR).
  - Logging continues <u>during</u> UNDO processing.
  - CLR has an extra field: UndoNextLSN
    - Points to the next LSN to undo (the PrevLSN of the record currently being undone)
  - CLRs are never UNDOne.
    - But might be REDOne when repeating history.
    - (Why?)

Transaction Commit

- Write **Commit** Record to Log
- All Log records up to the transaction's LastLSN are flushed.
  - Guarantees that FlushedLSN ≥ LastLSN
  - Note that Log Flushes are Sequential, Synchronous Writes to Disk
- Commit() returns.
- Write **End** record to log.

Next week: Crash Recovery