# ARIES (& Logging)

*April 2-4, 2018*

What does it mean for a transaction to be committed?

If commit returns successfully, the transaction…

- … is recorded completely (atomicity)
- … left the database in a stable state (consistency)
- …'s effects are independent of other xacts (isolation)
- … will survive failures (durability)

commit
returns
successfully

=

the xact's
effects
are visible
<u>forever</u>

| | | |
|---|---|---|
| commit returns successfully | = | the xact's effects are visible <u>forever</u> |
| commit called but doesn't return | = | the xact's effects <u>may</u> be visible |

# Motivation

T1

T2

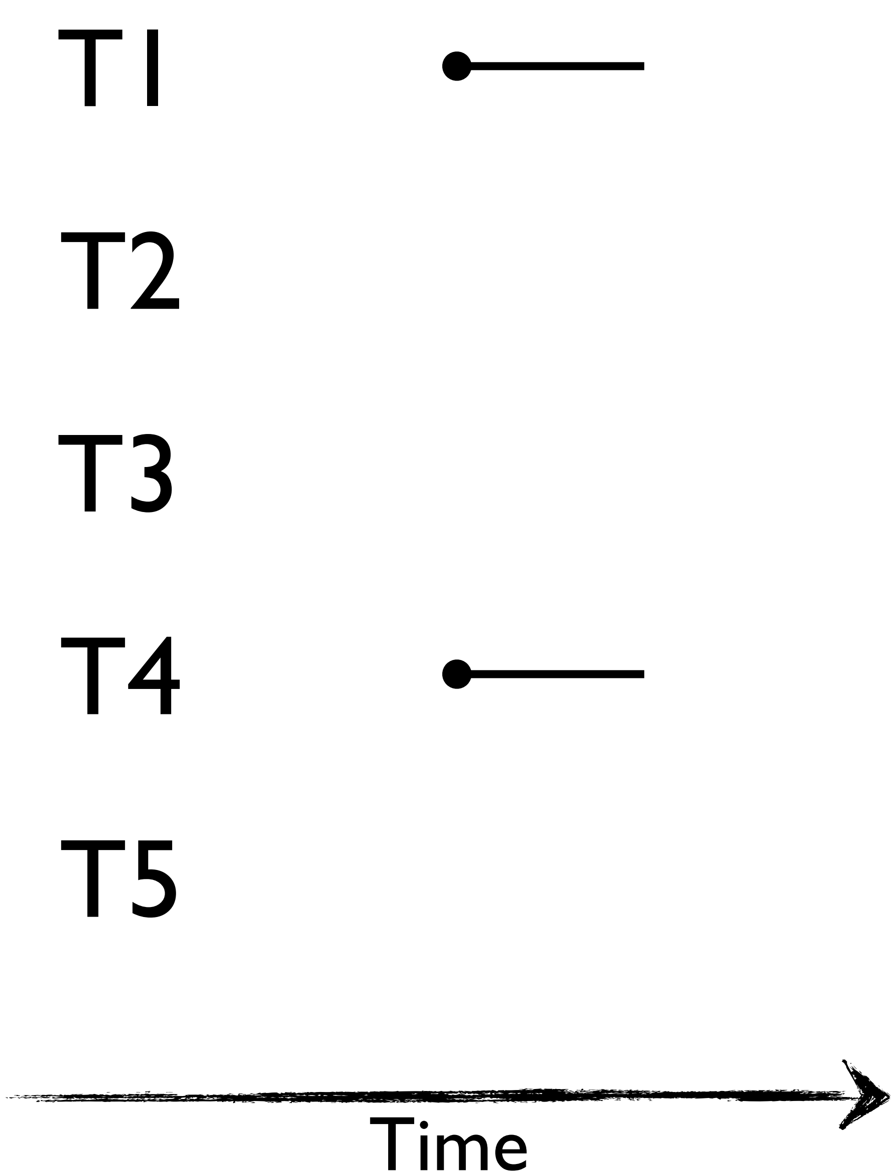
T3

T4

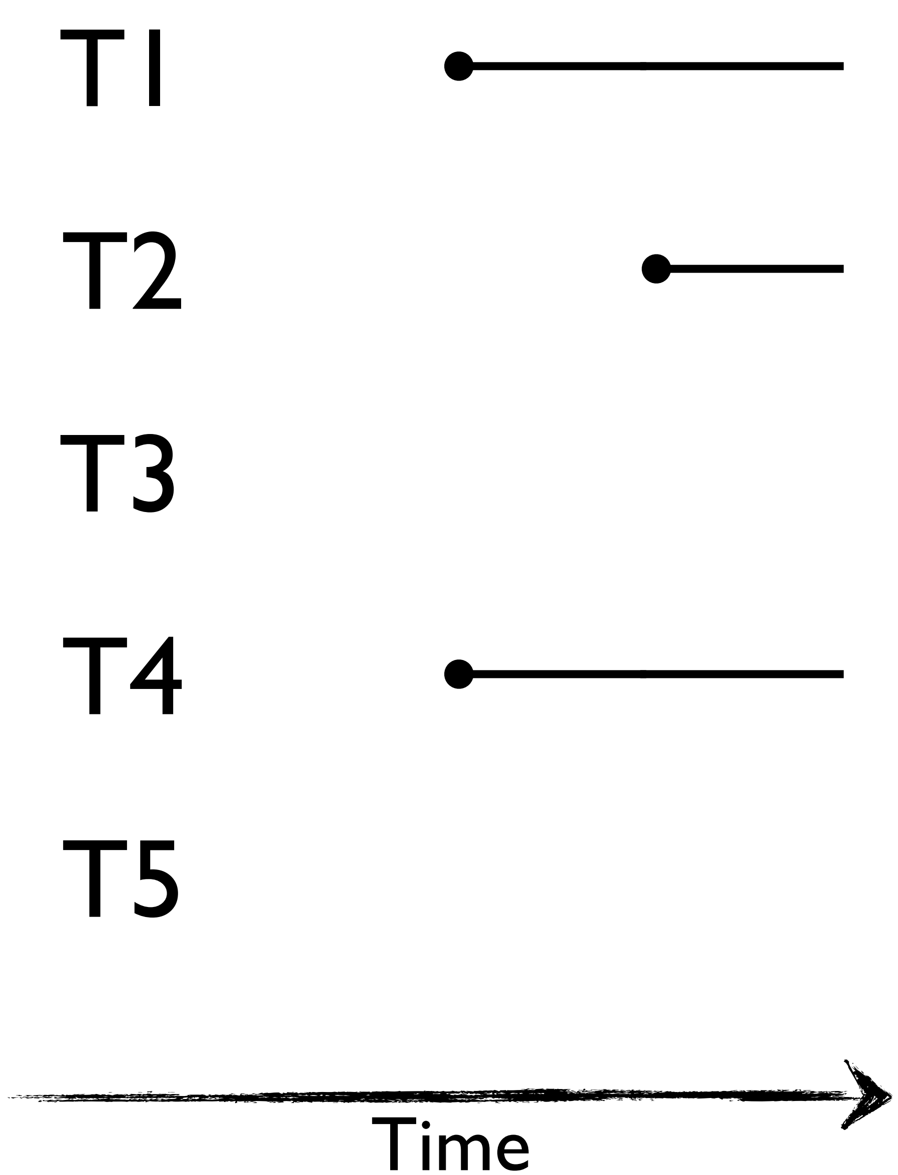
T5

Time

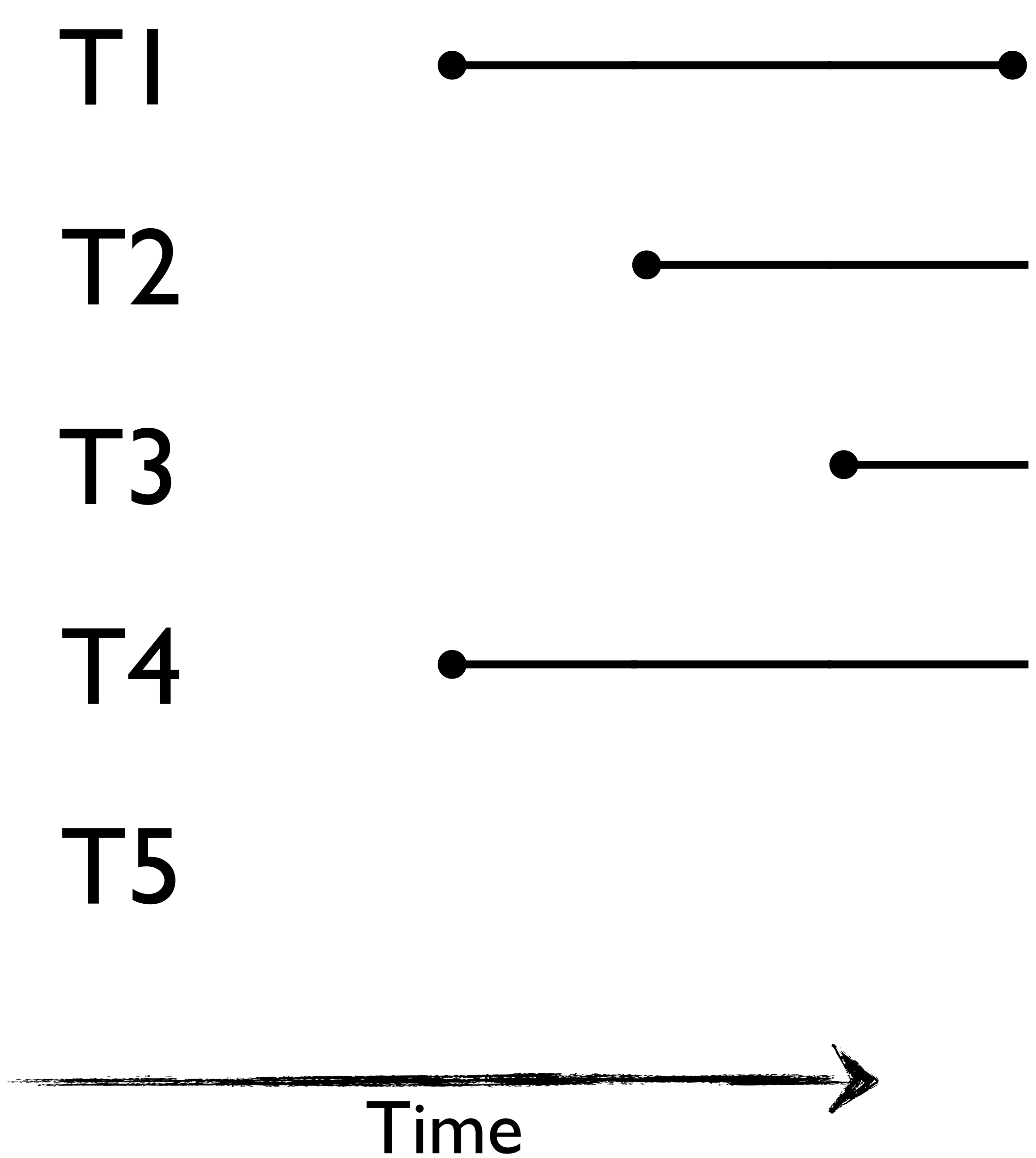
# Motivation



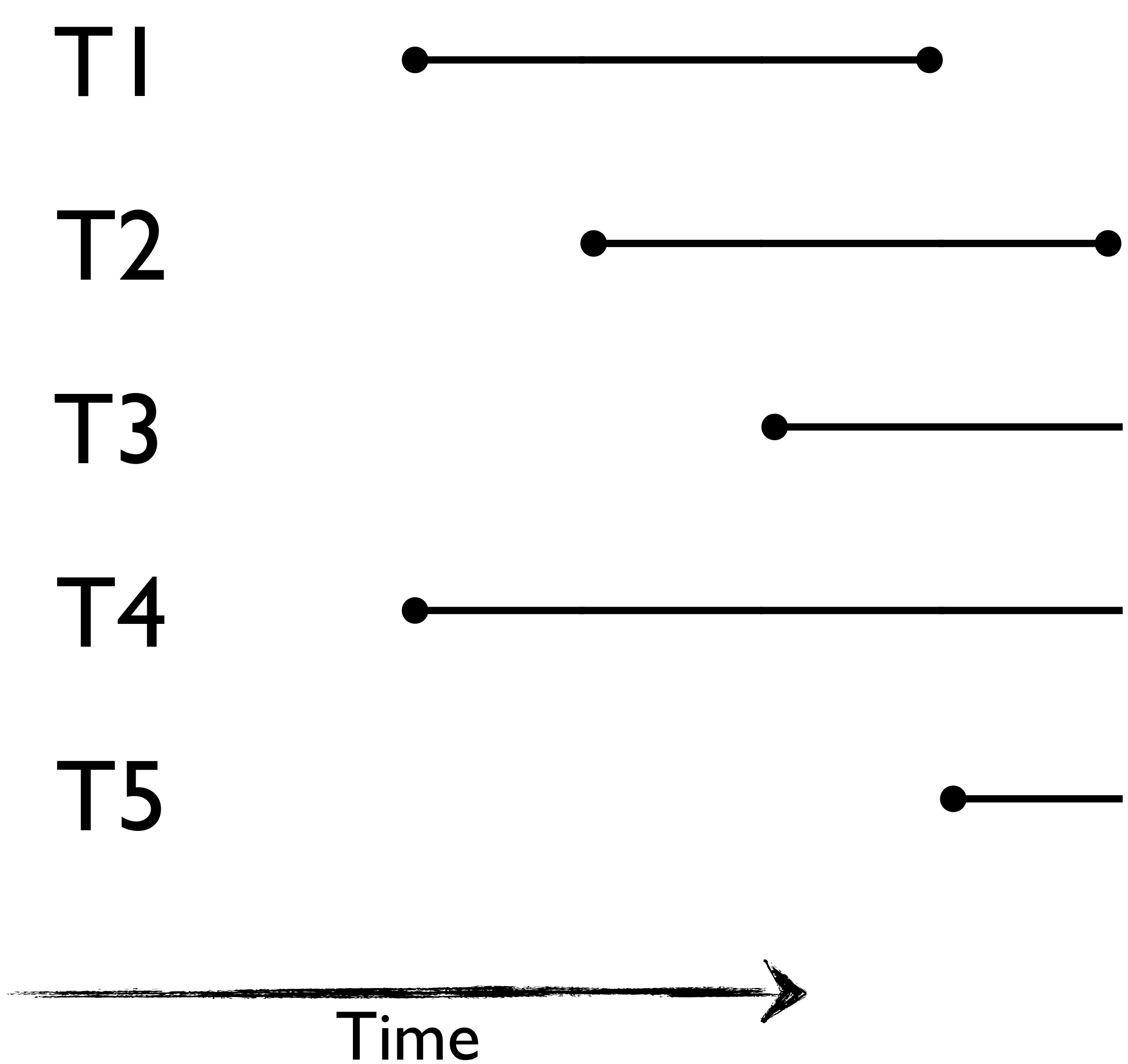
# Motivation



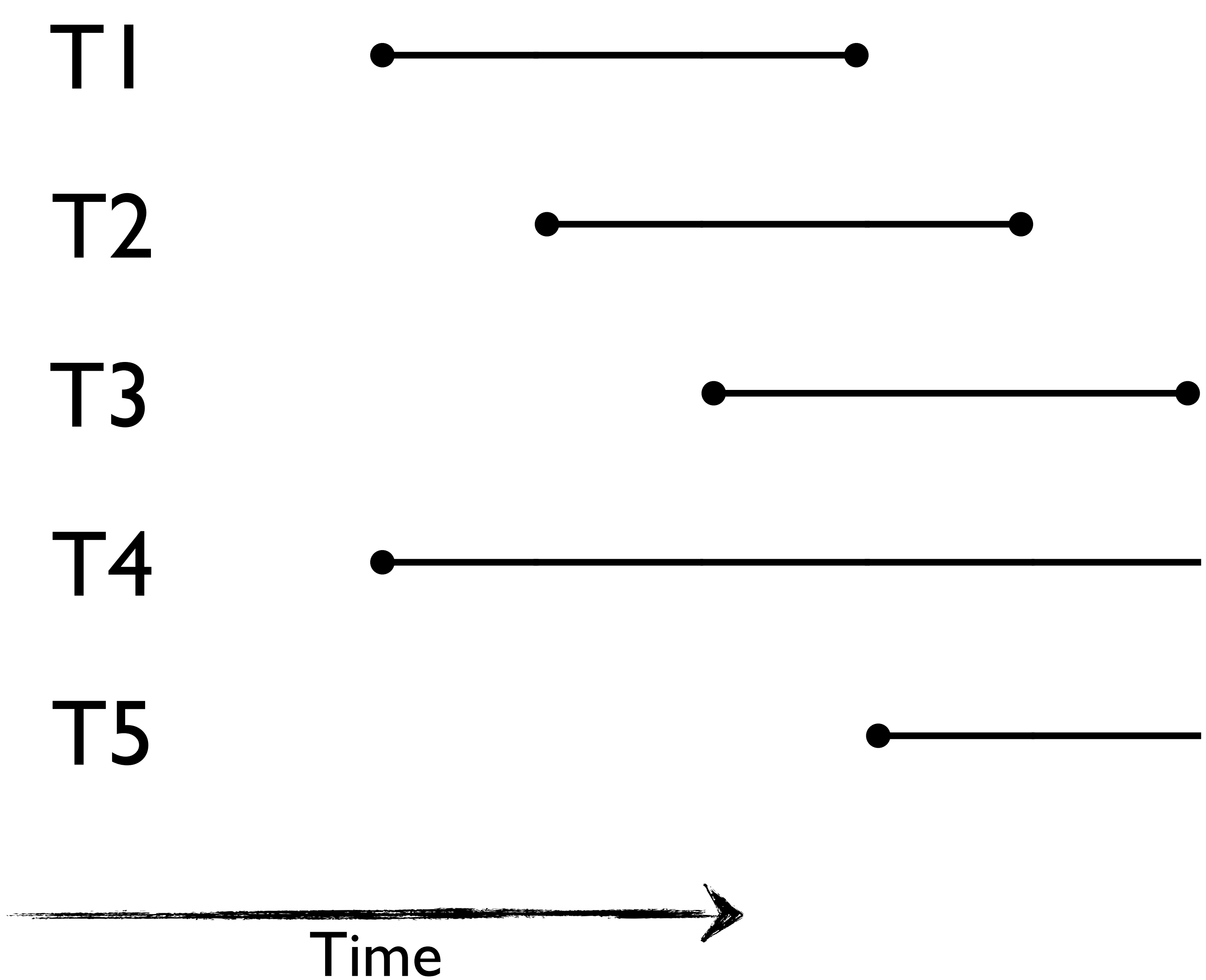
# Motivation



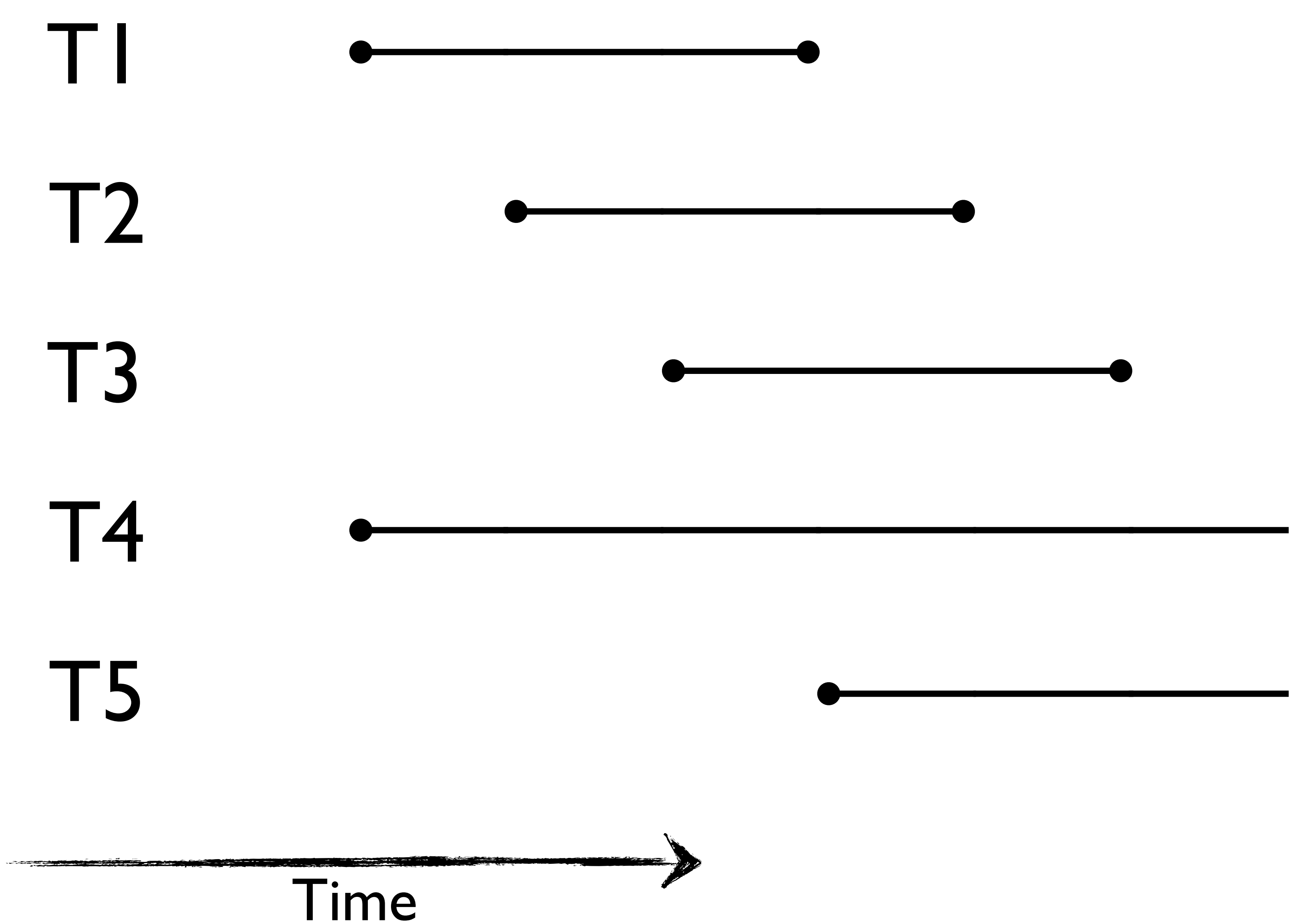
# Motivation



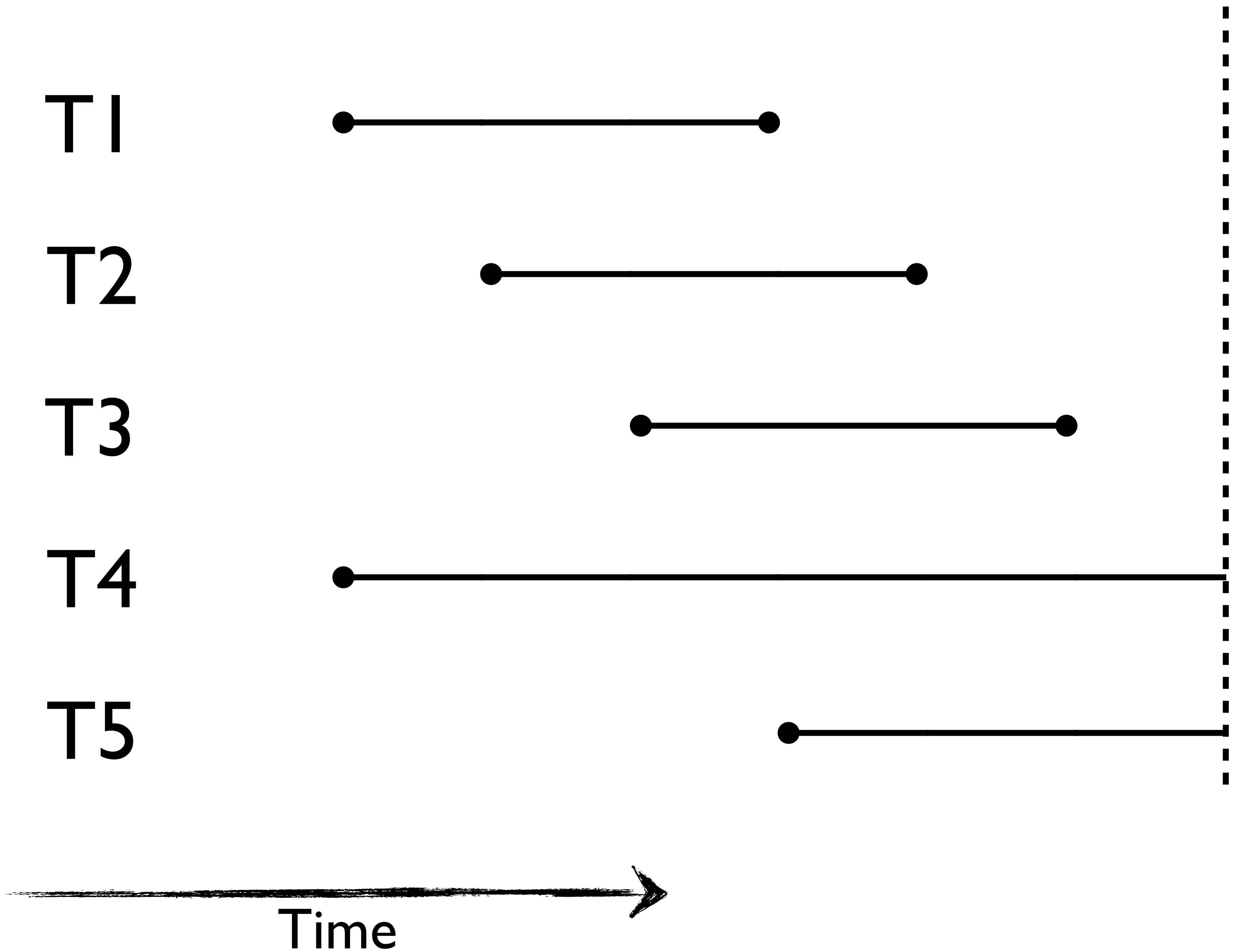
# Motivation



# Motivation

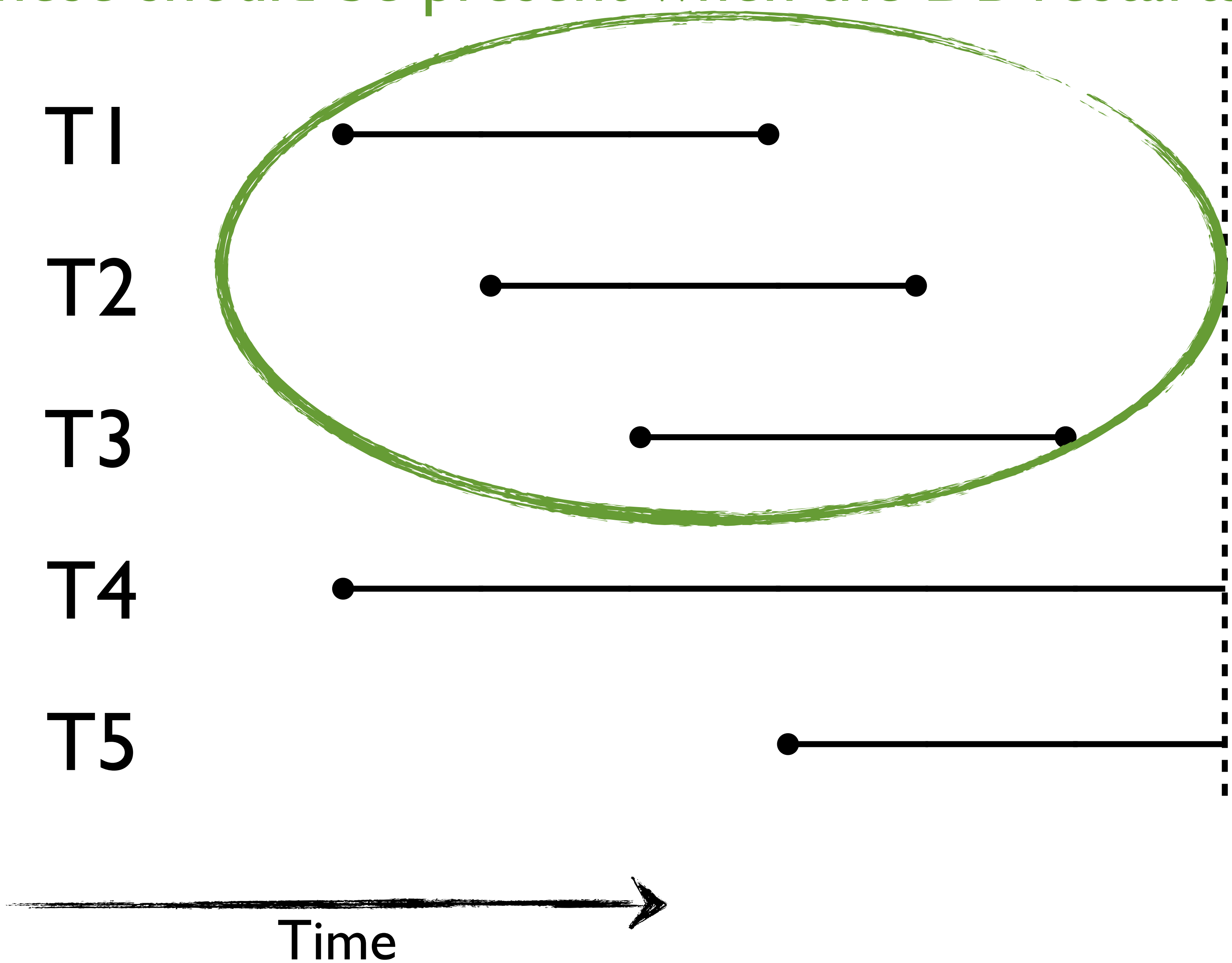


# Motivation



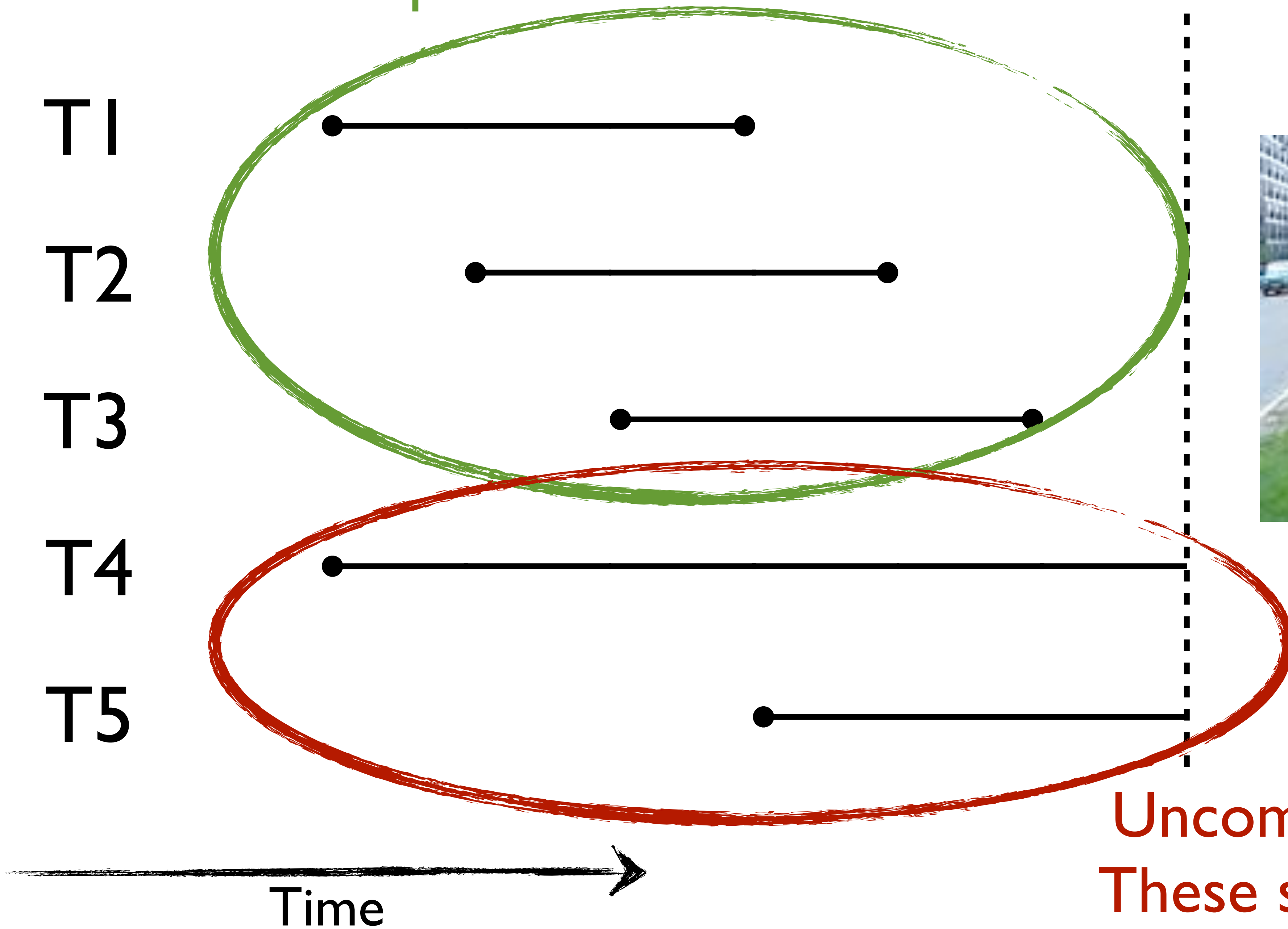

# Motivation

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




# Motivation

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





Uncommitted Transactions.
These should leave no trace

- How do we guarantee durability under failures?

- How do aborted transactions get rolled back?

- How do we guarantee atomicity under failures?

**Problem 1**: Providing durability under failures.

Simplified Model

When a write succeeds, the data is completely written

# Problems

- A crash occurs part-way through the write.

- A crash occurs before buffered data is written.

# Write-Ahead Logging

Before writing to the database, first write what you plan to write to a log file…

| | |
|---|---|
| **A** | 8 |
| **B** | 12 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

**Log**

`W(A:10)`

# Write-Ahead Logging

Once the log is safely on disk
you can write the database

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

**<u>Log</u>**

`W(A:10)`

# Write-Ahead Logging

Log is append-only,
so writes are always
efficient

| | |
|---|---|
| A | ~~8~~ 10 |
| B | 12 |
| C | 5 |
| D | 18 |
| E | 16 |

**Log**

```
W(A:10)
W(C:8)
W(E:9)
```

# Write-Ahead Logging

…allowing random writes
to be safely batched

**<u>Log</u>**

```
W(A:10)
W(C:8)
W(E:9)
```

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | ~~5~~ 8 |
| **D** | 18 |
| **E** | ~~16~~ 9 |

**Problem 2**: Providing rollback.

# Single DB Model

**<u>Txn 1</u>**

```
A = 20
B = 14
COMMIT
```

**<u>Txn 2</u>**

```
E = 19
B = 15
ABORT
```

| | |
|---|---|
| **A** | 8 |
| **B** | 12 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

# Single DB Model

**Txn 1**

```
A = 20
B = 14
COMMIT
```

**Txn 2**

```
E = 19
B = 15
ABORT
```

| | |
|---|---|
| **A** | ~~8~~ 20 |
| **B** | 12 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

# Single DB Model

**Txn 1**

```
A = 20
B = 14
COMMIT
```

**Txn 2**

```
E = 19
B = 15
ABORT
```

| | |
|---|---|
| **A** | ~~8~~ 20 |
| **B** | ~~12~~ |
| **C** | 5 |
| **D** | 18 |
| **E** | ~~16~~ 19 |

# Single DB Model

**Txn 1**

```
A = 20
B = 14
COMMIT
```

**Txn 2**

```
E = 19
B = 15
ABORT
```

| | |
|---|---|
| **A** | ~~8~~ 20 |
| **B** | ~~12~~ 14 |
| **C** | 5 |
| **D** | 18 |
| **E** | ~~16~~ 19 |

# Single DB Model

| | |
|---|---|
| A | ~~8~~ 20 |
| B | ~~12~~ ~~14~~ 15 |
| C | 5 |
| D | 18 |
| E | ~~16~~ 19 |

**Txn 1**

```
A = 20
B = 14
COMMIT
```

**Txn 2**

```
E = 19
B = 15
ABORT
```

# Staged DB Model

**Txn 1**

```
A = 20
B = 14
COMMIT
```

**Txn 2**

```
E = 19
B = 15
ABORT
```

| | |
|---|---|
| A | 8 |
| B | 12 |
| C | 5 |
| D | 18 |
| E | 16 |

| | |
|---|---|
| A | 8 |
| B | 12 |
| C | 5 |
| D | 18 |
| E | 16 |

# Staged DB Model

**Txn 1**

```
A = 20
B = 14
COMMIT
```

**Txn 2**

```
E = 19
B = 15
ABORT
```

| | |
|---|---|
| **A** | ~~8~~ 20 |
| **B** | ~~12~~ 14 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

| | |
|---|---|
| **A** | 8 |
| **B** | ~~12~~ 15 |
| **C** | 5 |
| **D** | 18 |
| **E** | ~~16~~ 19 |

# Staged DB Model

**Txn 1**

```
A = 20
B = 14
COMMIT
```

**Txn 2**

```
E = 19
B = 15
ABORT
```

| | |
|---|---|
| **A** | ~~8~~ 20 |
| **B** | ~~12~~ 14 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

Is staging always possible?

- Staging takes up more memory.
- Merging after-the-fact can be harder.
- Merging after-the-fact introduces more latency!

for the single database model

**Problem 2**: Providing rollback.

^

# UNDO Logging

Store both the “old” and the “new” values of the record being replaced

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | ~~5~~ 8 |
| **D** | 18 |
| **E** | ~~16~~ 9 |

**Log**

```
W(A:8→10)
W(C:5→8)
W(E:16→9)
```

# UNDO Logging

| | |
|---|---|
| A | ~~8~~ 10 |
| B | 12 |
| C | ~~5~~ 8 |
| D | 18 |
| E | ~~16~~ 9 |

**Active Xacts**

Xact:1, Log: 45

Xact:2, Log: 32

**Log**

```
43: W(A:8→10)
44: W(C:5→8)
45: W(E:16→9)
```

# UNDO Logging

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | ~~5~~ 8 |
| **D** | 18 |
| **E** | ~~16~~ 9 |

**Active Xacts**

Xact:3, Log: 45

**ABORT**

Xact:2, Log: 32

**Log**

```
43: W(A:8→10)
44: W(C:5→8)
45: W(E:16→9)
```

# UNDO Logging

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | ~~5~~ 8 |
| **D** | 18 |
| **E** | 16 |

**Active Xacts**

Xact:1, Log: 45

**ABORT**

Xact:2, Log: 32

**Log**

```
43: W(A:8→10)
44: W(C:5→8)
45: W(E:16→9)
```

# UNDO Logging

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

**Active Xacts**

Xact:1, Log: 45 **ABORT**

Xact:2, Log: 32

**Log**

```
43: W(A:8→10)
44: W(C:5→8)
45: W(E:16→9)
```

# UNDO Logging

| | |
|---|---|
| **A** | 8 |
| **B** | 12 |
| **C** | 5 |
| **D** | 18 |
| **E** | 16 |

**Active Xacts**

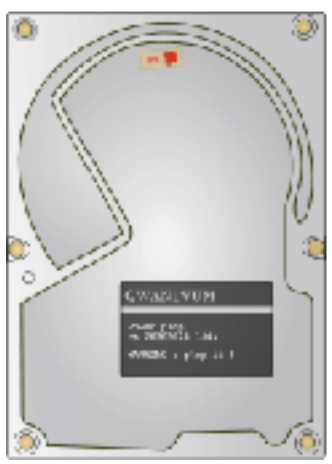
Xact:1, Log: 45 **ABORT**

Xact:2, Log: 32

**Log**

```
43: W(A:8→10)
44: W(C:5→8)
45: W(E:16→9)
```

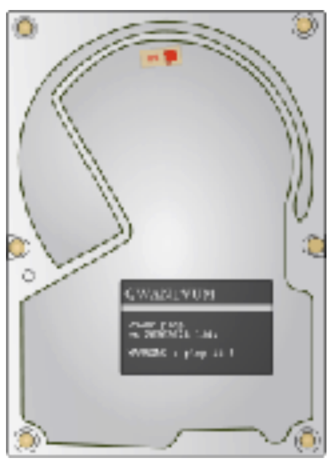
# Log Sequence Number
# Linked Lists



Transaction Table

Log

# Log Sequence Number Linked Lists



Transaction Table

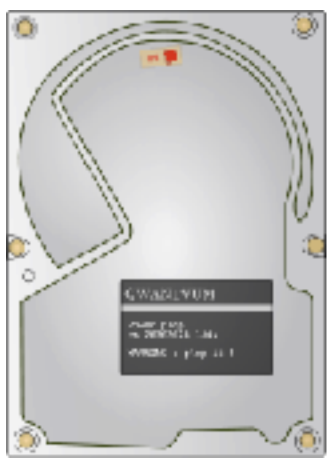
| |
|---|
| |

| | |
|---|---|
| | ABORT [XID] |

Log

(necessary for crash recovery)

# Log Sequence Number Linked Lists



Transaction Table

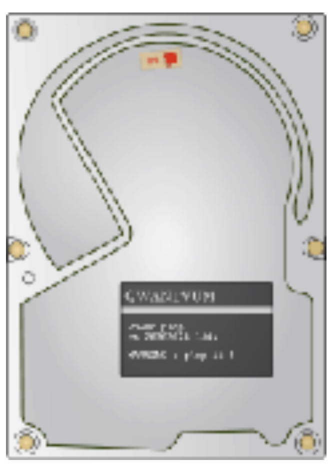
| XID, LastLSN |
|---|

| | |
|---|---|
| | ABORT [XID] |

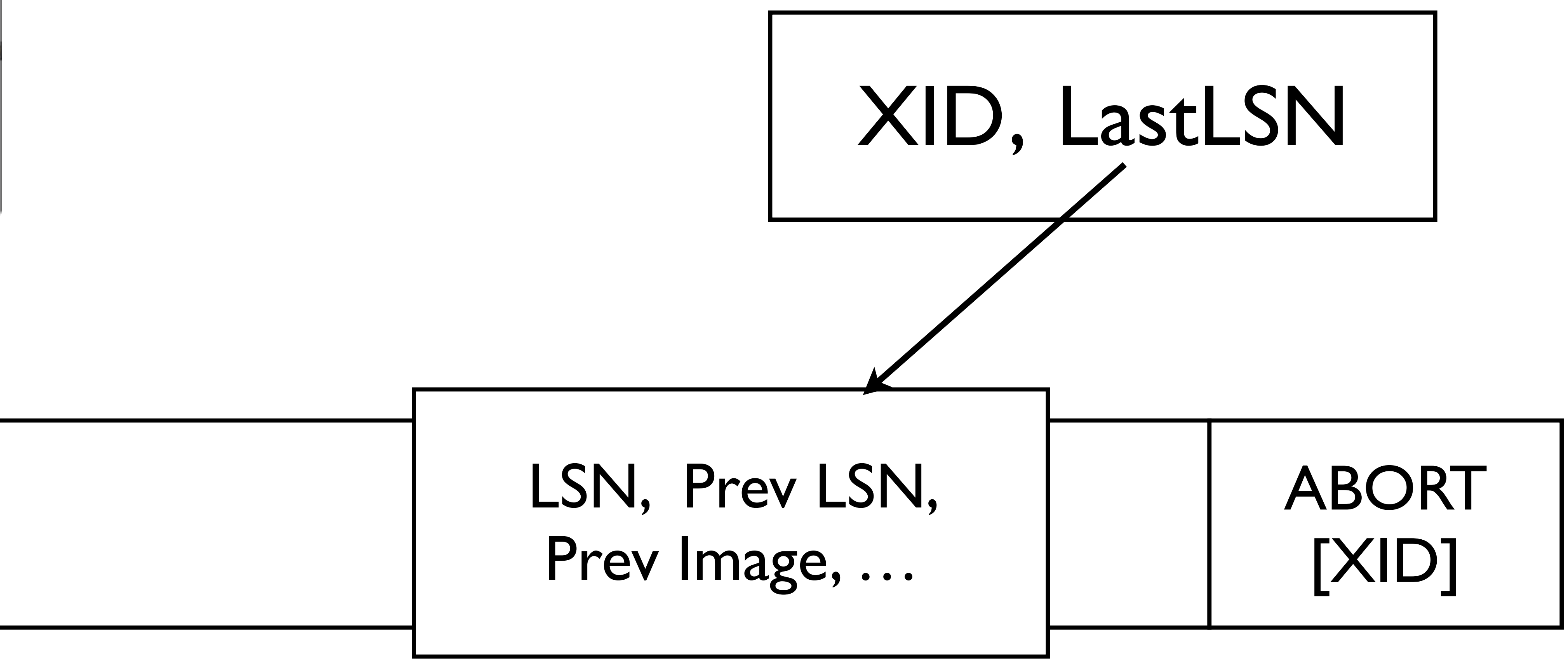
Log

(necessary for crash recovery)

# Log Sequence Number Linked Lists



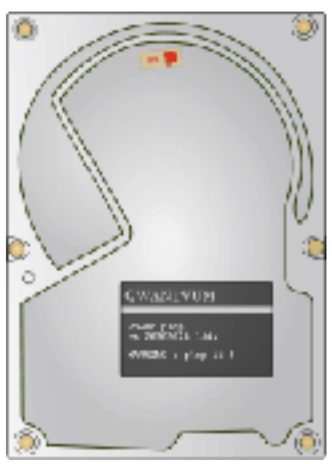
Transaction Table




Log

(necessary for crash recovery)

# Log Sequence Number Linked Lists



Transaction Table

XID, LastLSN

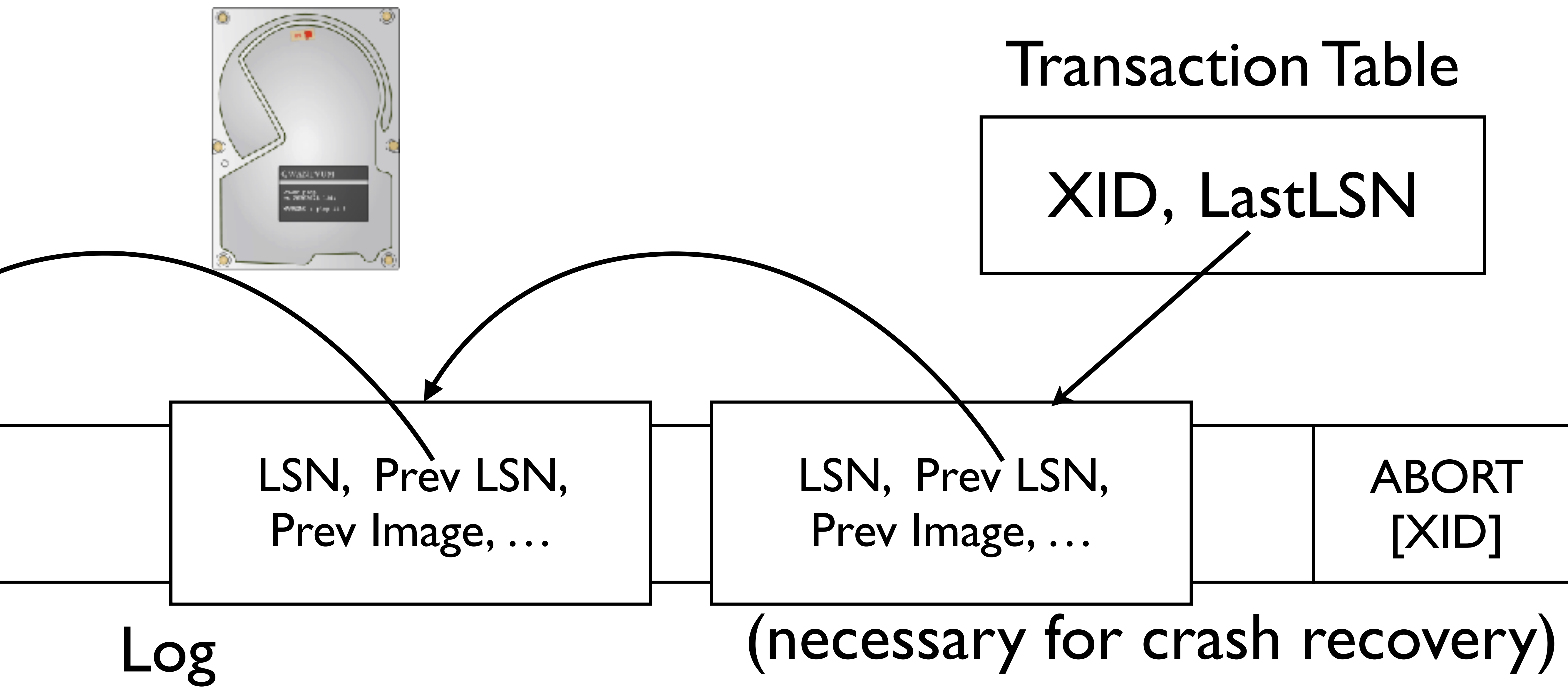
LSN, Prev LSN, Prev Image, …

ABORT [XID]

Log

(necessary for crash recovery)

# Log Sequence Number Linked Lists



**Problem 3**: Providing atomicity.

**Goal**: Be able to reconstruct all state at the time of the DB's crash (minus all running xacts)

What state is relevant?

# DB State

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | ~~5~~ 8 |
| **D** | 18 |
| **E** | ~~16~~ 9 |

**Active Xacts**

Xact:1, Log: 45

Xact:2, Log: 32

**Log**

```
43: W(A:8→10)
44: W(C:5→8)
45: W(E:16→9)
```

# DB State

**On-Disk**
**(or rebuildable)**

| | |
|---|---|
| **A** | ~~8~~ 10 |
| **B** | 12 |
| **C** | ~~5~~ 8 |
| **D** | 18 |
| **E** | ~~16~~ 9 |

**In-Memory Only!**

**<u>Active Xacts</u>**

Xact:1, Log: 45

Xact:2, Log: 32

**On-Disk**

**<u>Log</u>**

```
43: W(A:8→10)
44: W(C:5→8)
45: W(E:16→9)
```

# Rebuilding the Xact Table

Log every COMMIT
(replay triggers commit process)

Log every ABORT
(replay triggers abort process)

New message: END
(replay removes Xact from Xact Table)

# Rebuilding the Xact Table

Log every COMMIT
(replay triggers commit process)

Log every ABORT
(replay triggers abort process)

New message: END
(replay removes Xact from Xact Table)

What about BEGIN?
(when does an Xact get added to the Table?)

# Transaction Commit

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

# 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?)

# Rebuilding the Xact Table

**Optimization**: Write the Xact Table to the log periodically.
(checkpointing)

# ARIES Crash Recovery

- Start from checkpoint stored in master record.
- Analysis: Rebuild the Xact Table
- Redo: Replay operations from all live Xacts (even uncommitted ones).
- Undo: Revert operations from all uncommitted/aborted Xacts.