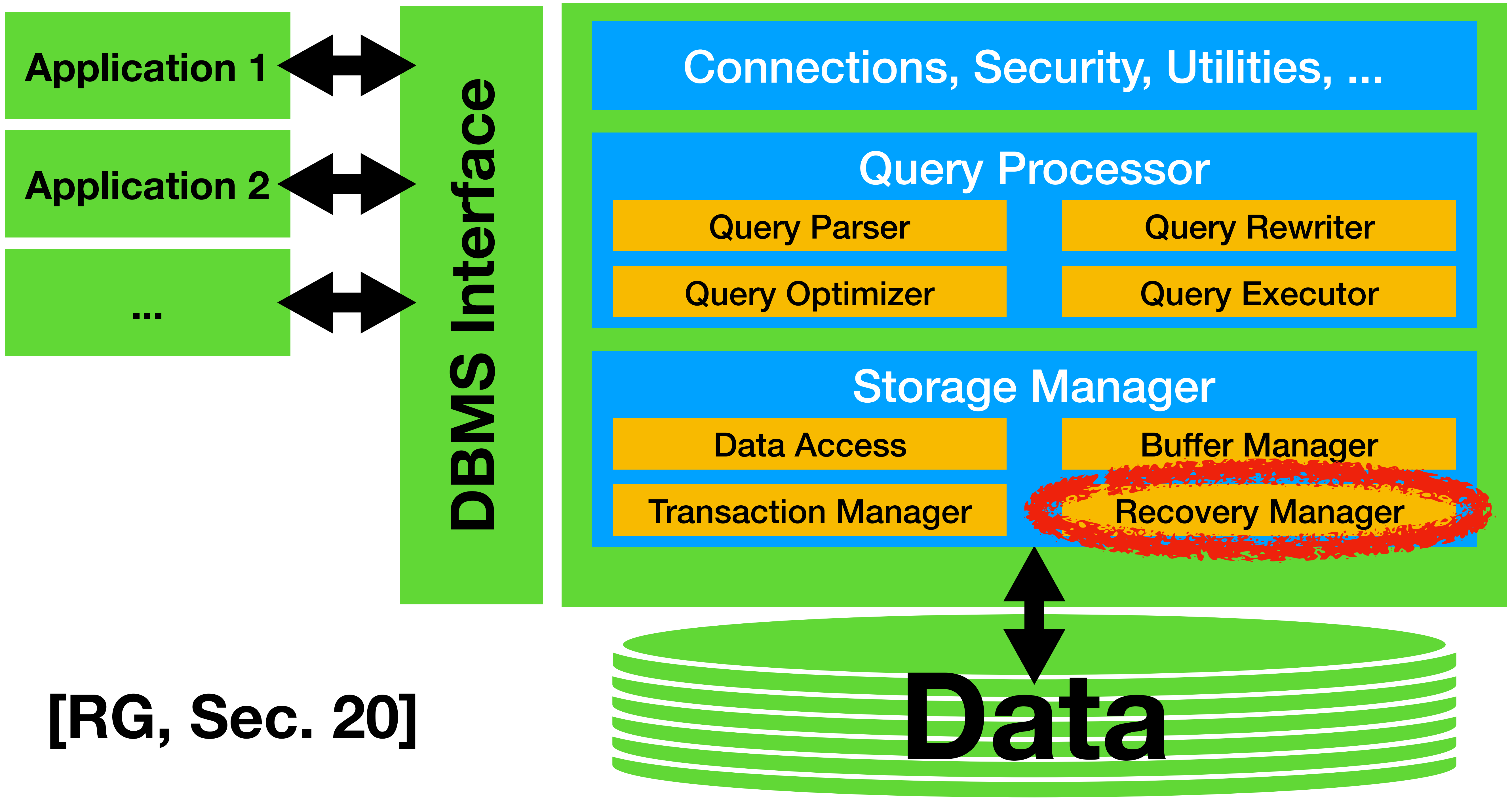
Recovery After System Crashes

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

Database Management Systems (DBMS)



[RG, Sec. 20]

Reminder ACID

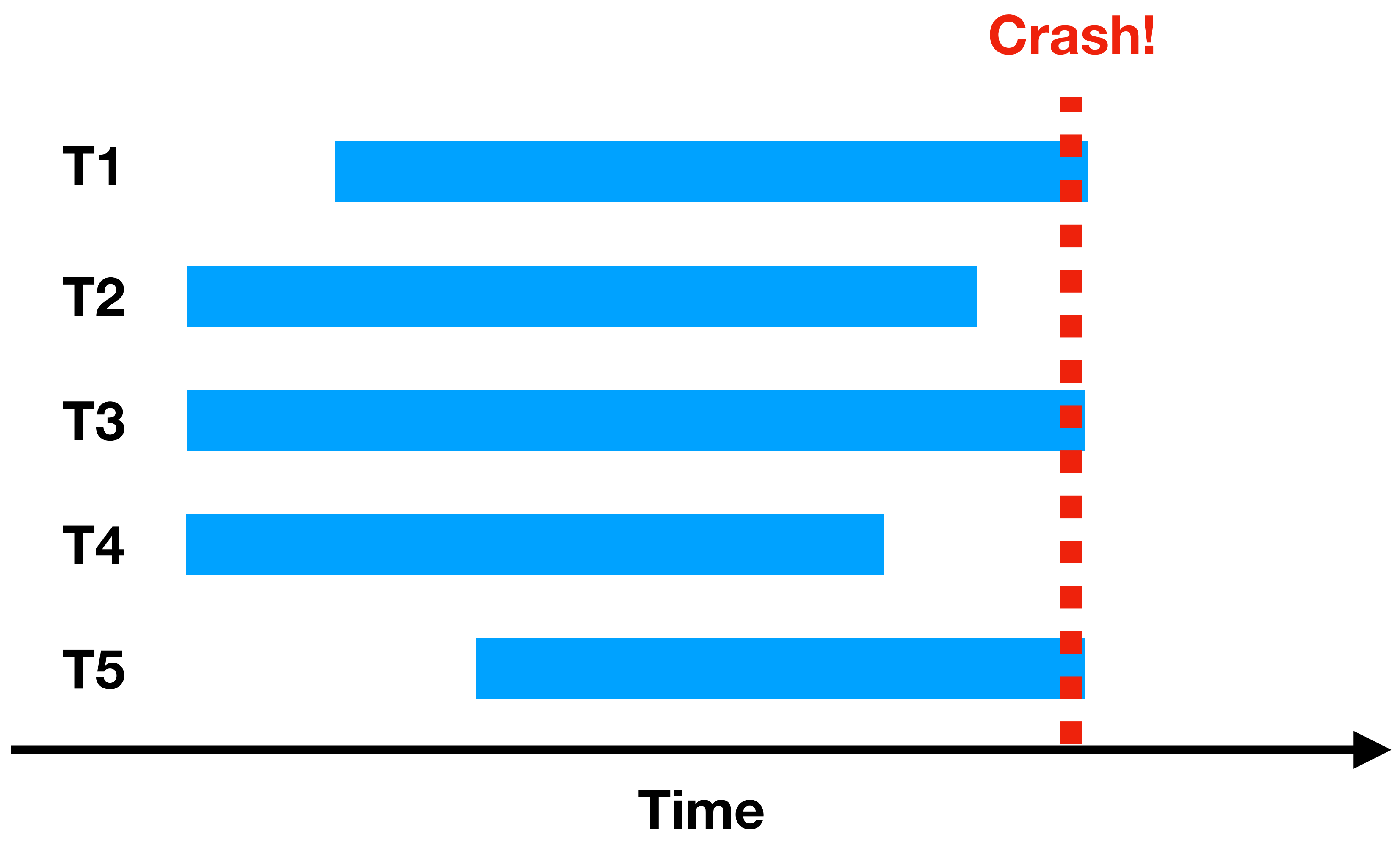
- **Atomicity**: no partial executions
- **Consistency**: data remains consistent
- **Isolation**: simulate serial execution
- **Durability**: no lost data

Reminder ACID

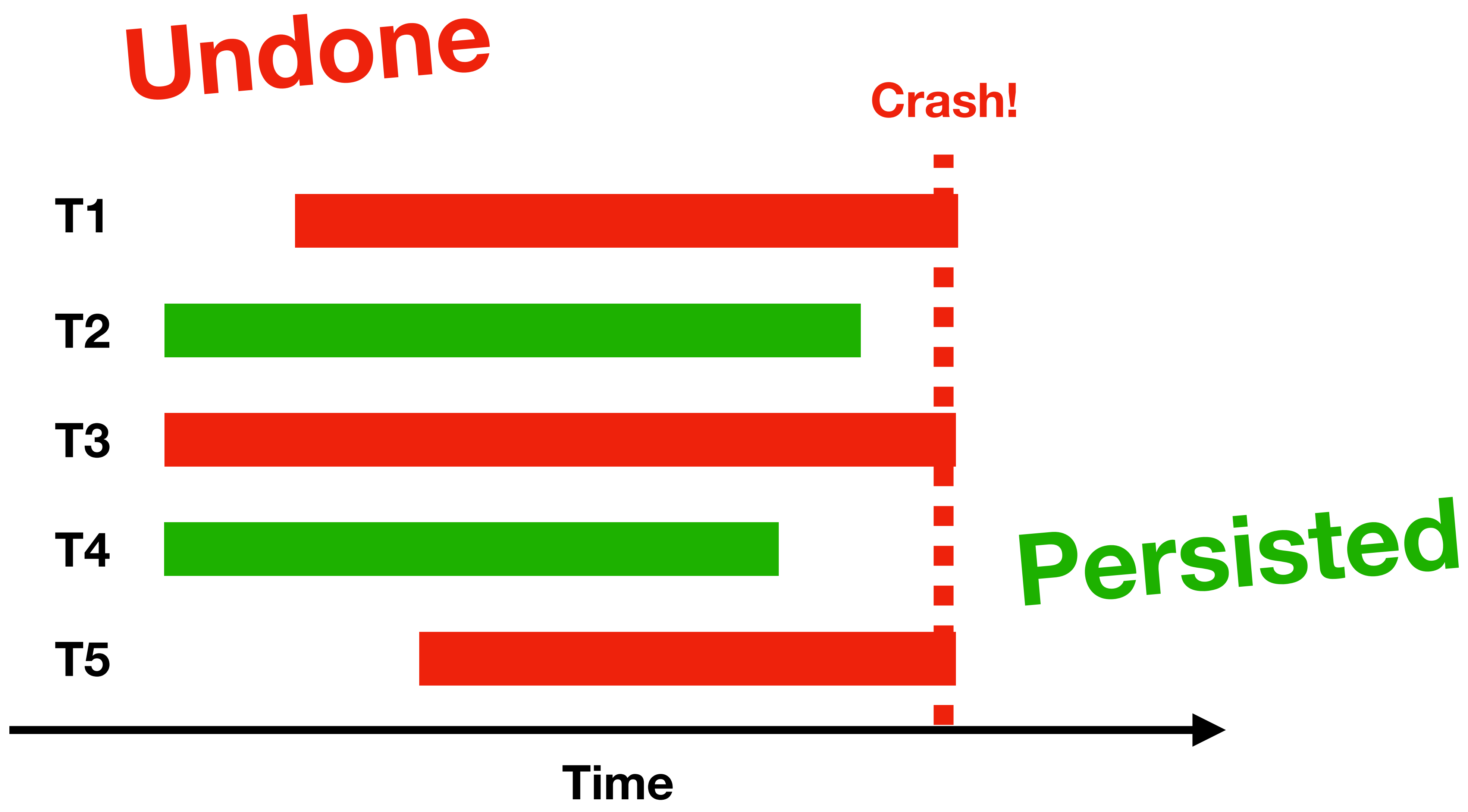
Focus of Recovery Manager

- **Atomicity**: no partial executions
- **Consistency**: data remains consistent
- **Isolation**: simulate serial execution
- **Durability**: no lost data

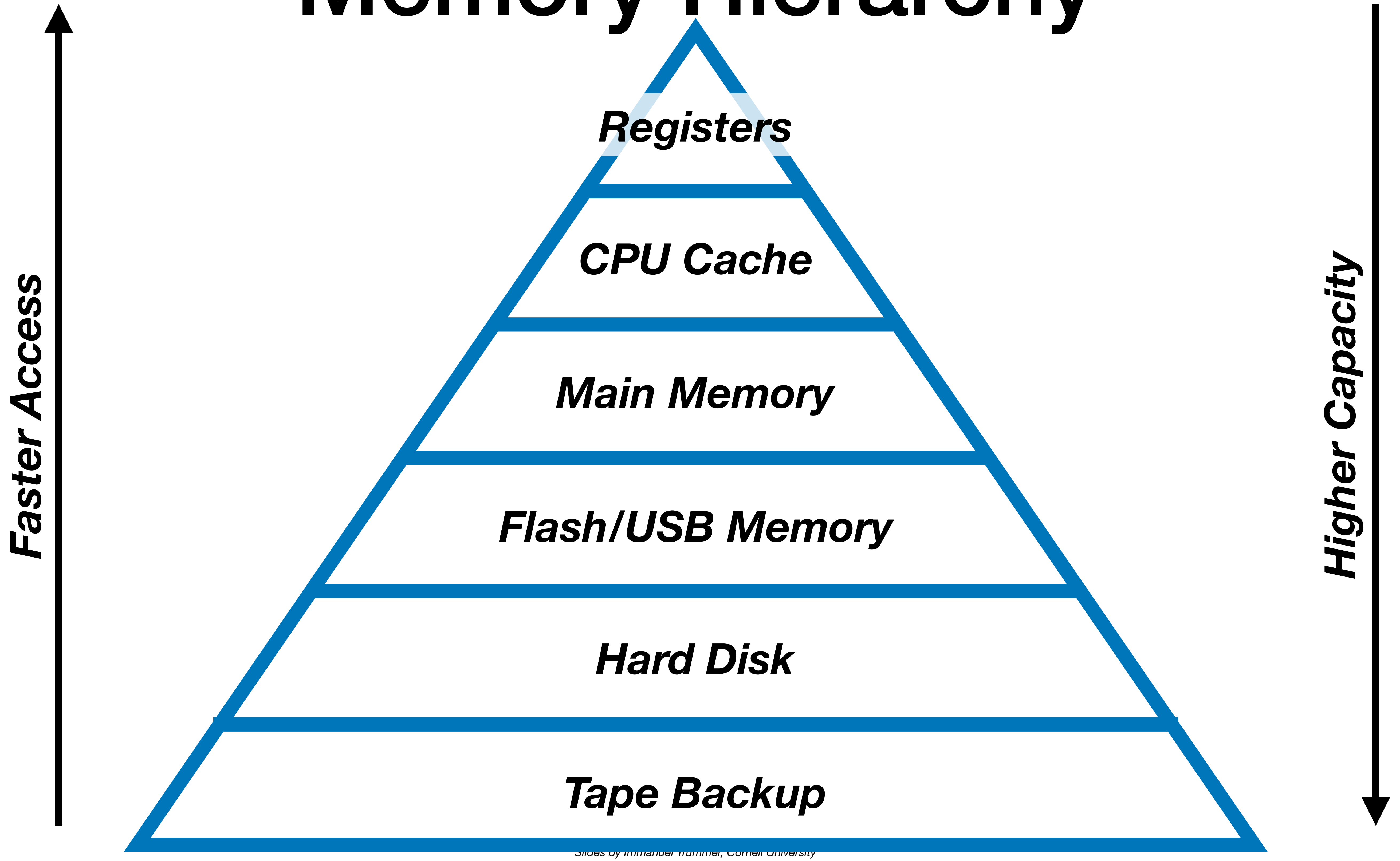
Desired Behavior



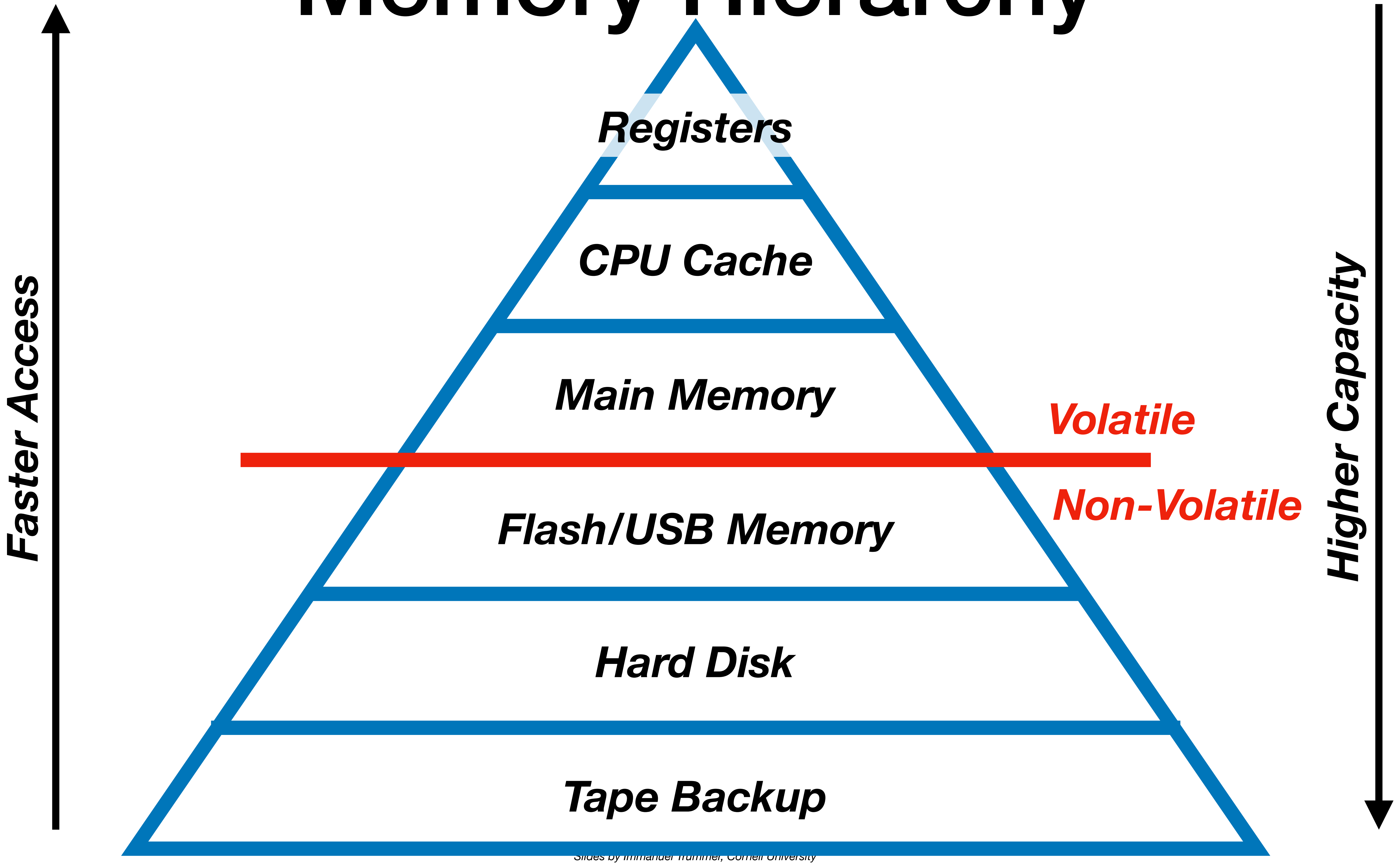
Desired Behavior



Reminder: Memory Hierarchy



Reminder: Memory Hierarchy



Durability Challenges

- Guarantee: effects of committed transactions **persist**
  - E.g., must still hold in case of sudden **power failure**
- Changes to data are initially written to **buffer pool**
  - Buffer pool in main memory, therefore **volatile**!
- **Persist** each change before commit to hard disk?
  - **Bad performance** (many page writes, small changes)

Atomicity Challenges

- Guarantee: no transaction is **partially executed**
- Can leave changes in **main memory** until commit
  - Means each transaction holds **many buffer slots**
  - **Bad throughput**: few transactions execute concurrently
- Alternative: write changes to **disk** to free up memory
  - Means we have **partial results** persistent on hard disk
  - May need to **undo changes** to achieve atomicity

Summary of Options

	No Steal (Buffer Pages from Ongoing Transactions)	Allow Steal
Force (Every Change to Disk)	Poor response time, poor throughput	Poor response time
No Force	Poor throughput	Good time & throughput

Summary of Options

	No Steal (Buffer Pages from Ongoing Transactions)	Allow Steal
Force (Every Change to Disk)	Poor response time, poor throughput	Poor response time
No Force	Poor throughput	Good time & throughput

But What About Durability/ Atomicity??

Logging for Durability

- Need to **persist changes** before commit for durability
- Updating actual data before commit is **inefficient**
    - Need to write lots of pages with **small changes**
- Idea: only store **"deltas"** in compact representation
    - Write **one page** with deltas instead of many data pages
    - Write deltas **before commit** for recovery after restart
    - Those deltas form (part of) the **log**

Logging for Atomicity

- Want to **swap** buffer pages between ongoing transactions
- Need to **persist changes** on swapped buffer pages
    - Otherwise, changes are **lost**, losing durability
- May have **uncommitted changes** persistent on disk
    - What if the corresponding transaction **aborts** at crash?
- Must be able to **undo** changes to guarantee atomicity
- Solution: write **log entries** to enable undoing updates

Write-Ahead Logging

- Write-ahead logging is characterized by **two rules**
  1. Write all log entries of a transaction **before commit**
  2. Write all log entries of a buffer page **before persisting**
- Rule 1 guarantees **durability**
  - Use log entries for **redo** in case of a crash
- Rule 2 guarantees **atomicity**
  - Use log entries for **undo** in case of a crash

ARIES Algorithm Overview

- One of the most popular **recovery algorithms**
- Uses **write-ahead logging** at run time
- Executes **multiple phases** after a crash:
    - **Analysis**: determine transactions to undo/redo via log
    - **Redo**: get back to state directly before the crash
    - **Undo**: undo effects of aborted transactions

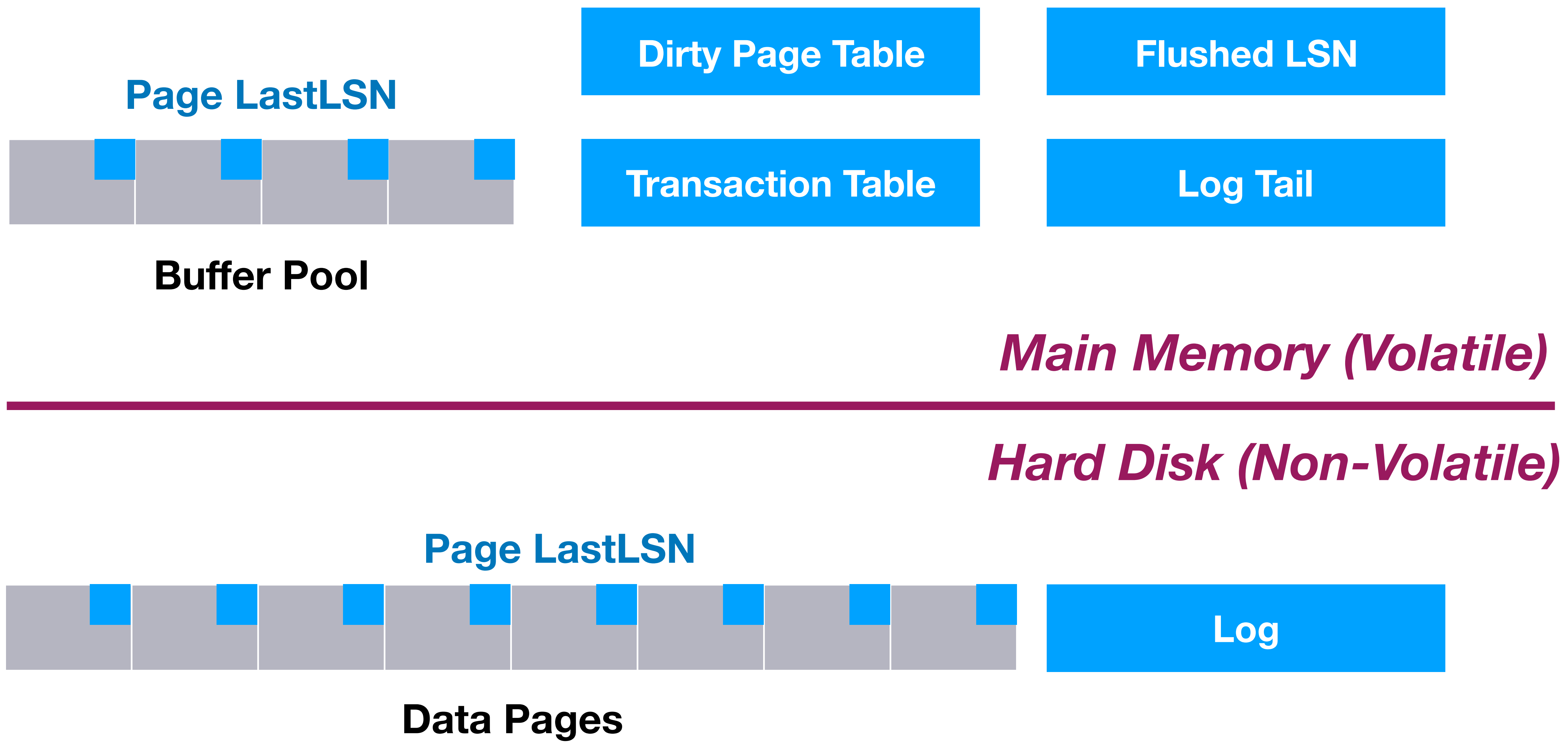
Outlook

- ARIES data structures
- ARIES run time behavior
- ARIES recovery algorithm

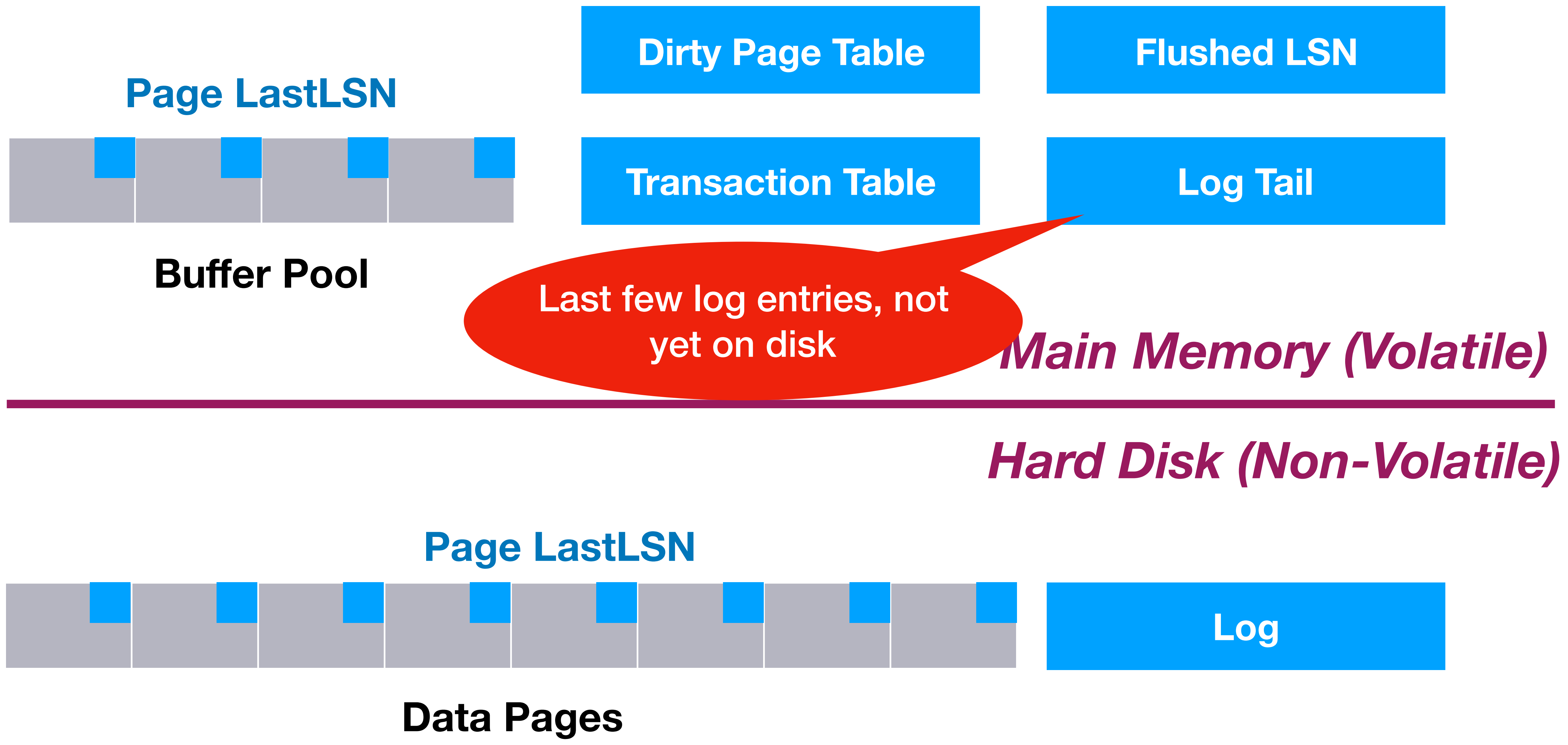
Outlook

- **ARIES data structures**
- ARIES run time behavior
- ARIES recovery algorithm

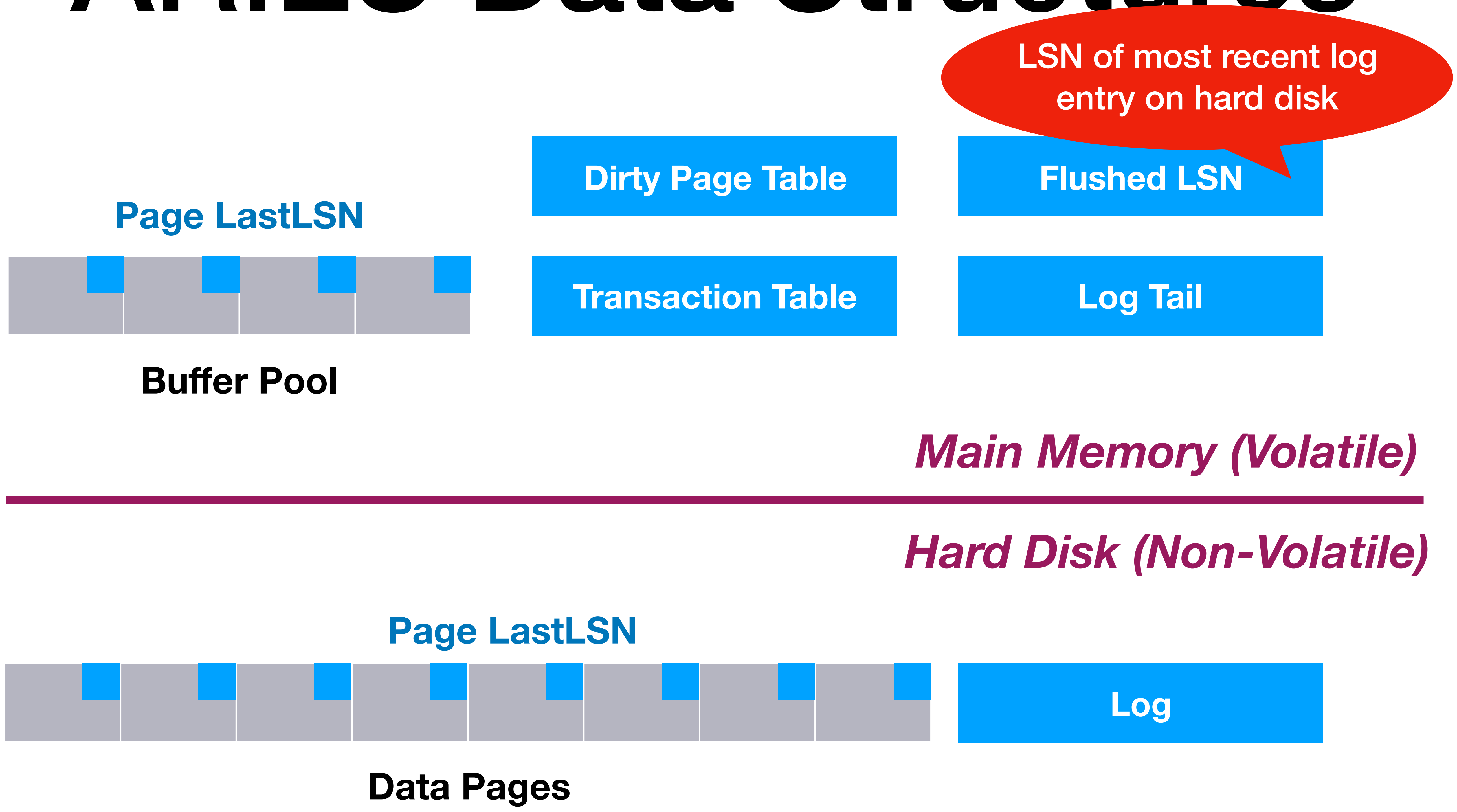
ARIES Data Structures



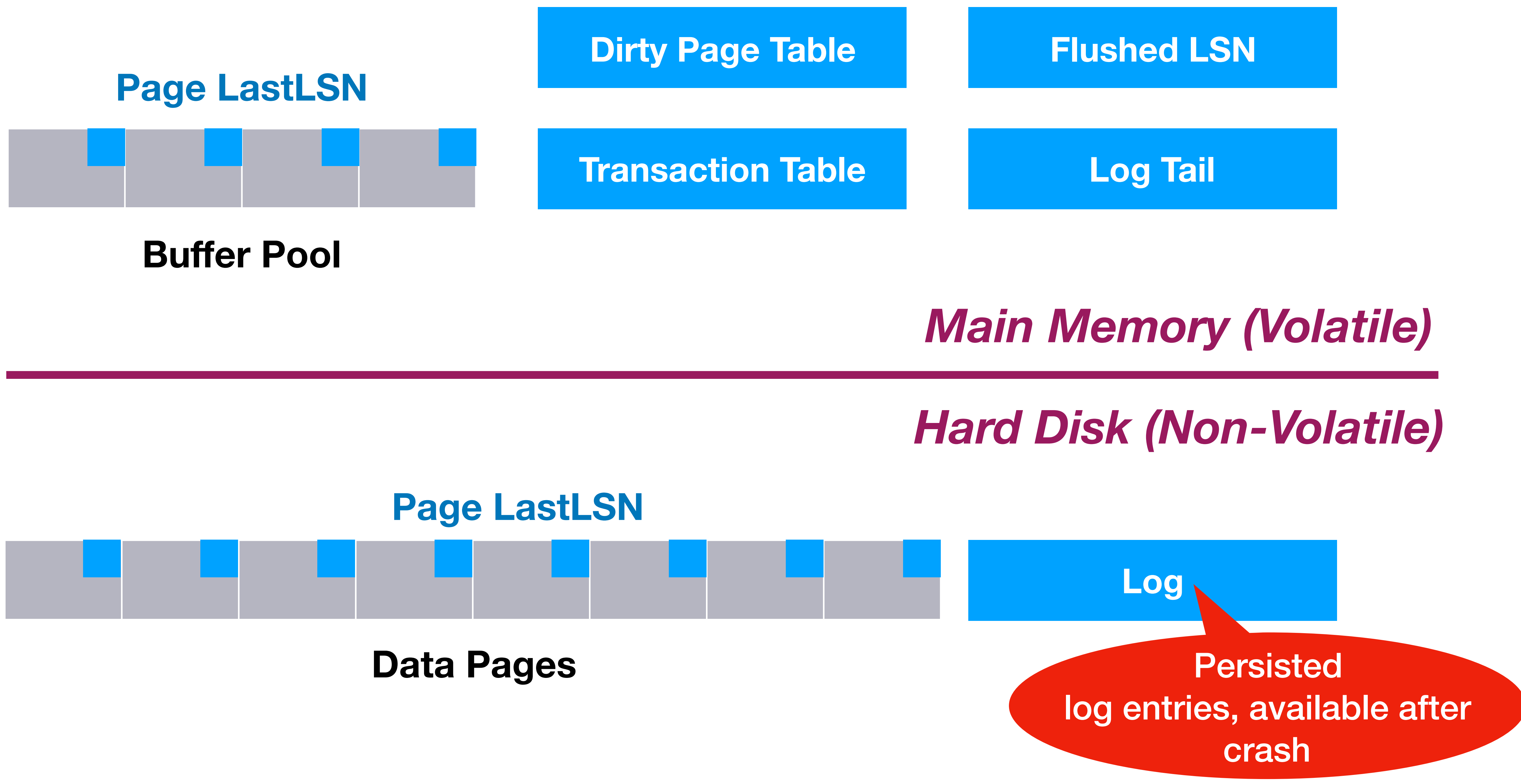
ARIES Data Structures



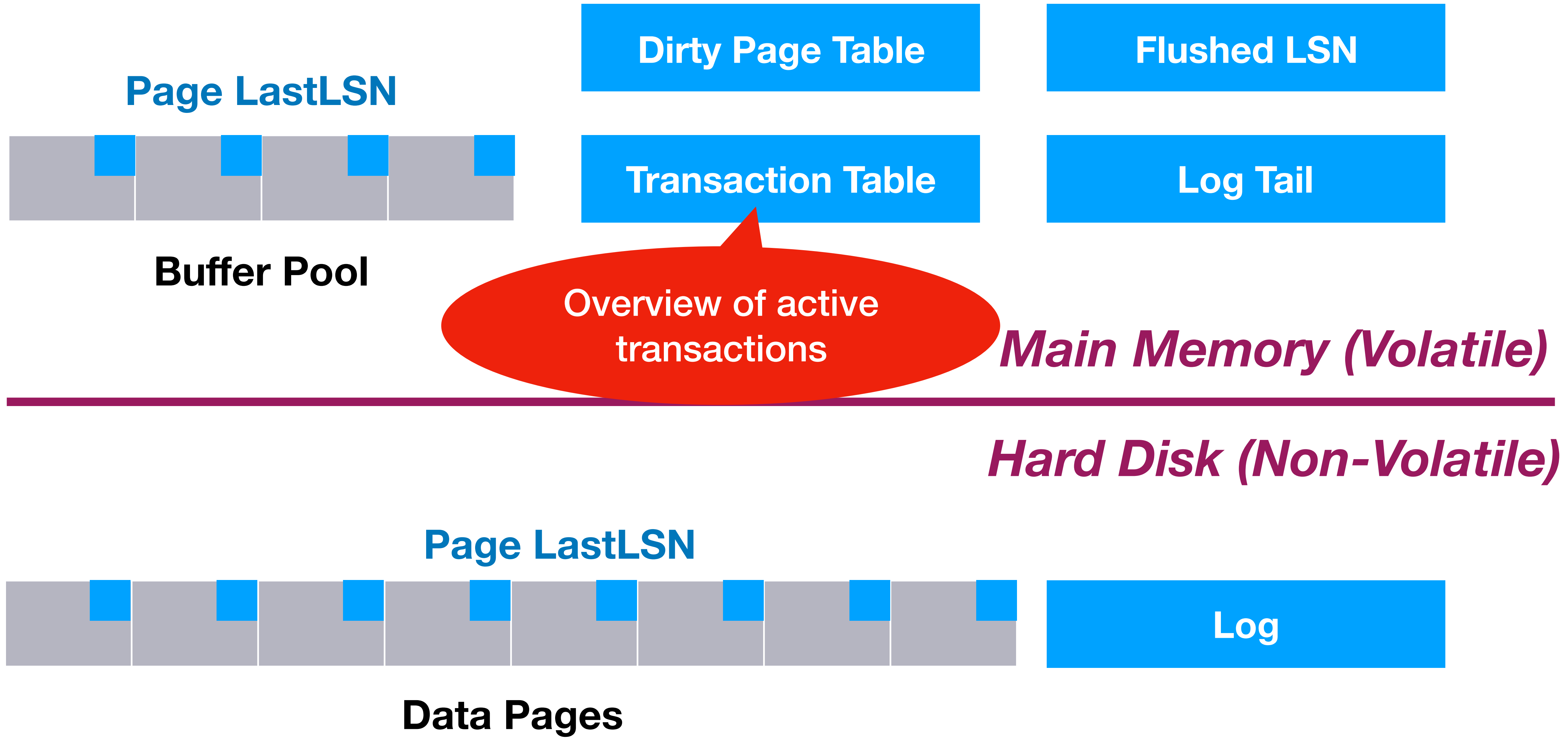
ARIES Data Structures



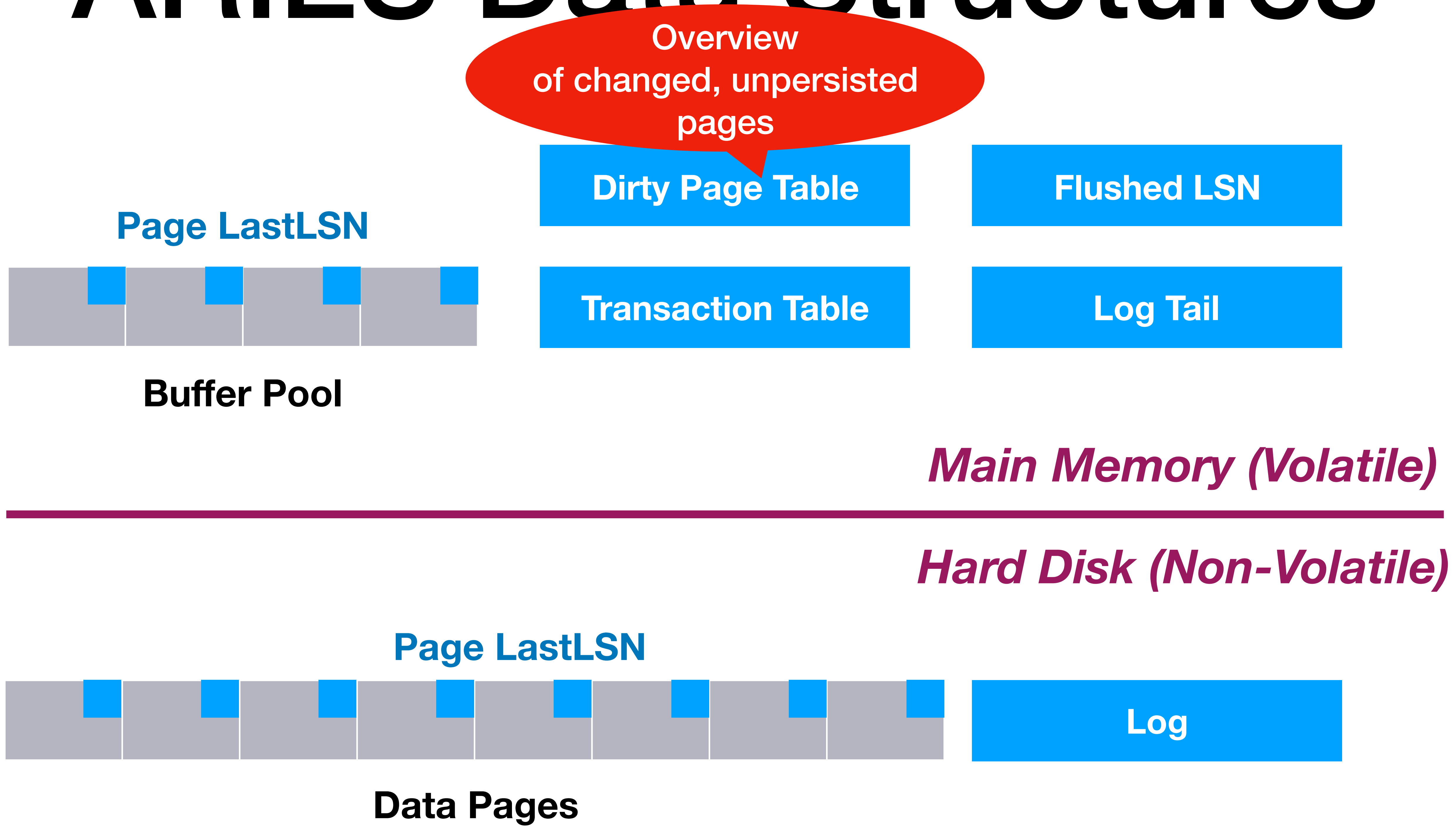
ARIES Data Structures



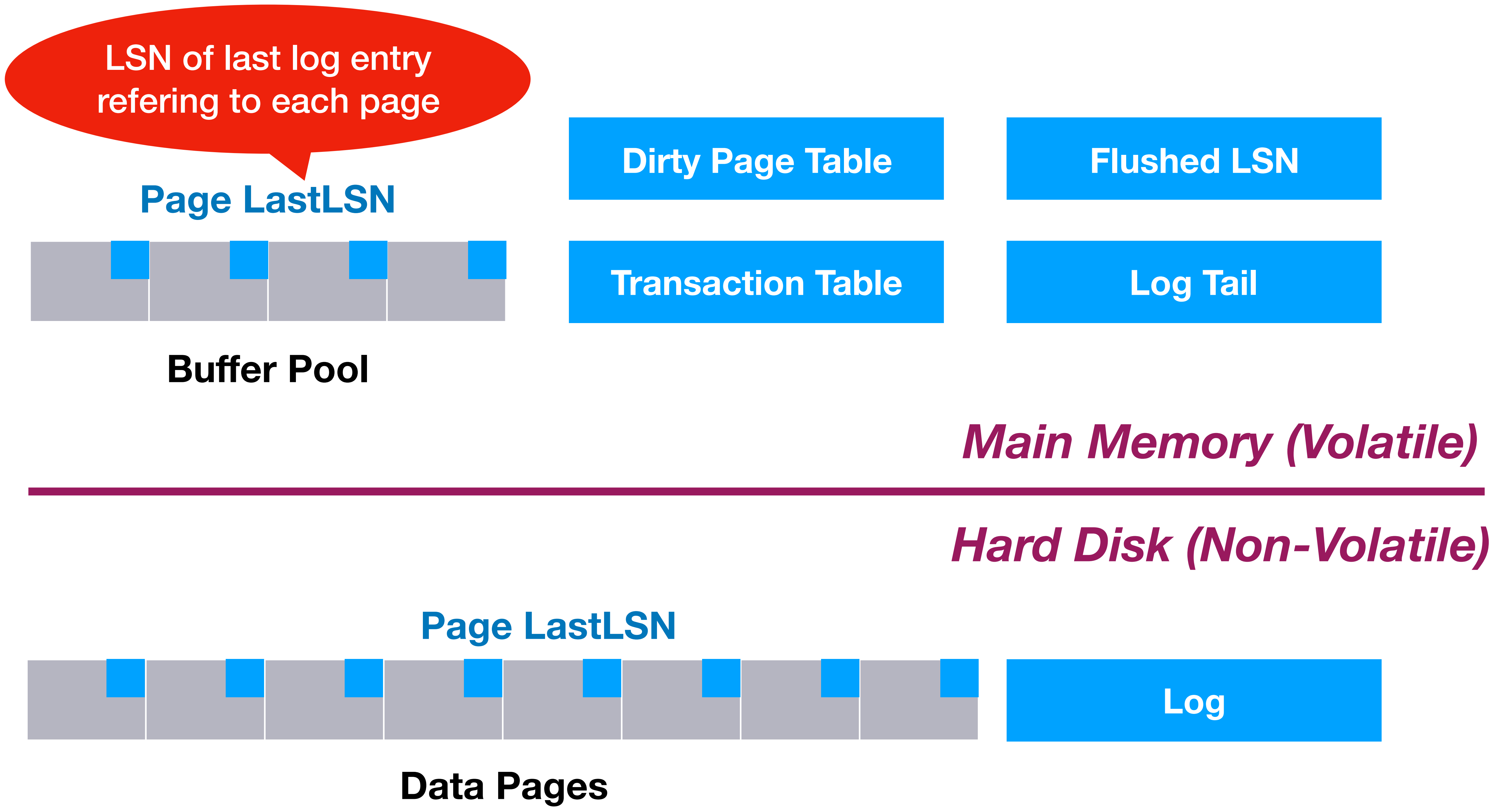
ARIES Data Structures



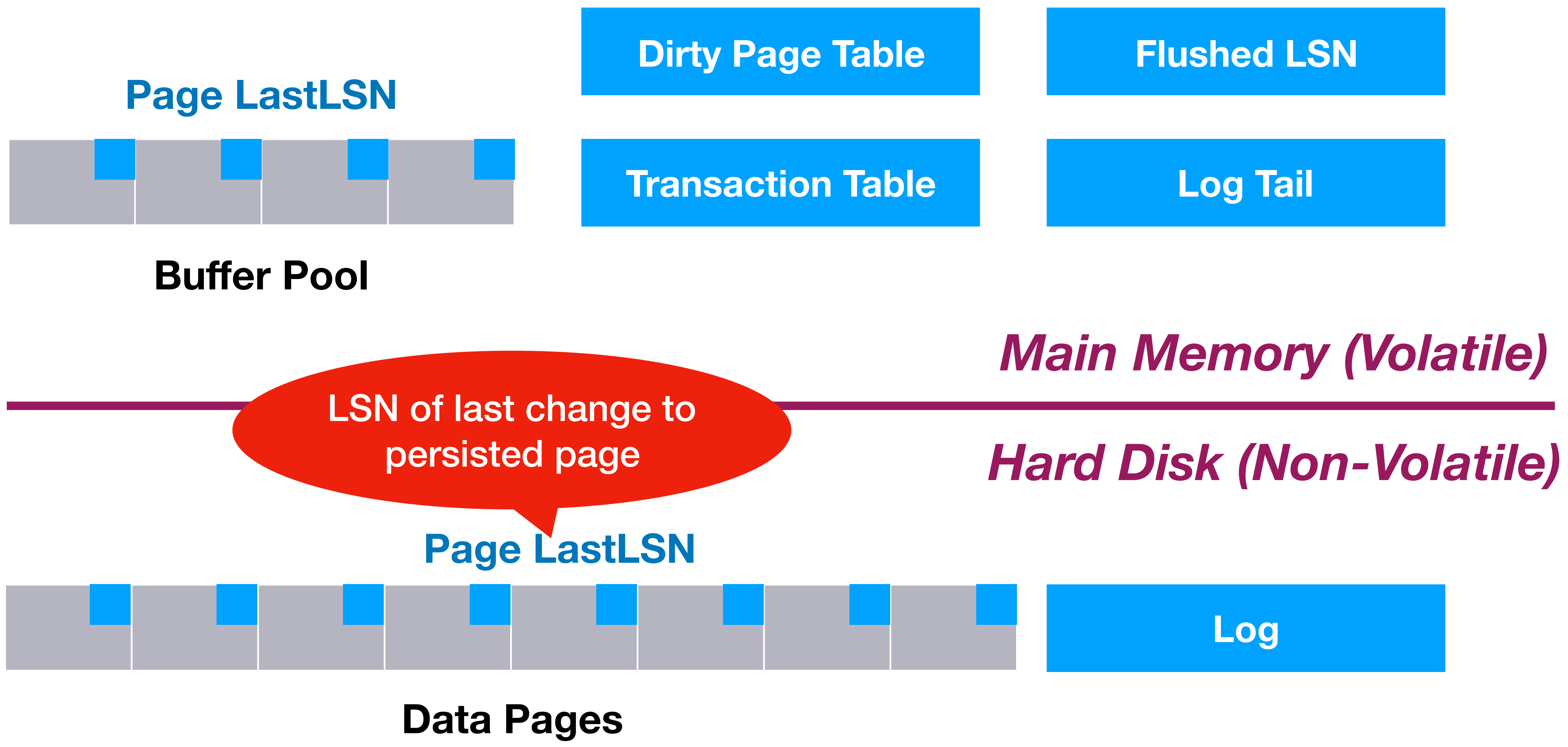
ARIES Data Structures



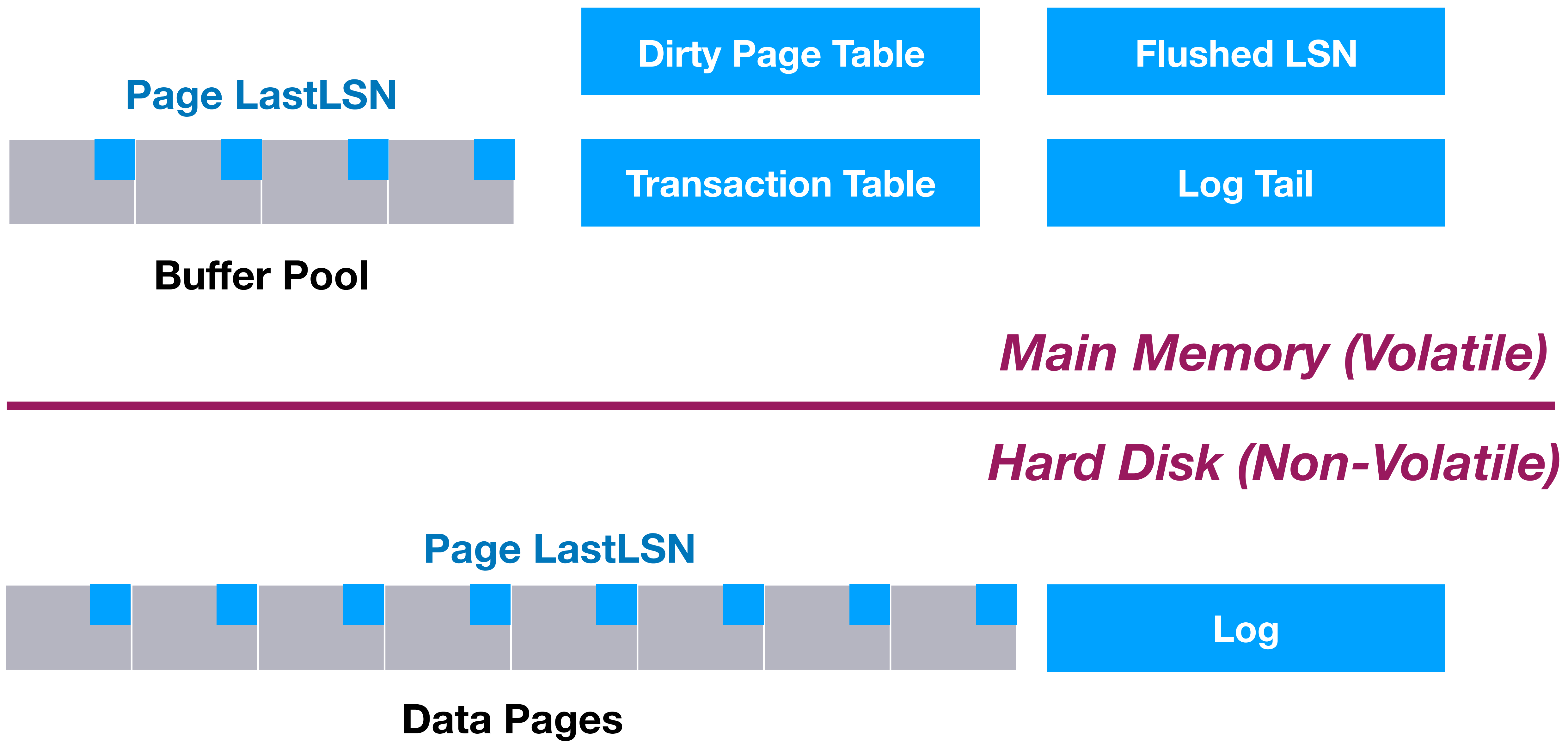
ARIES Data Structures



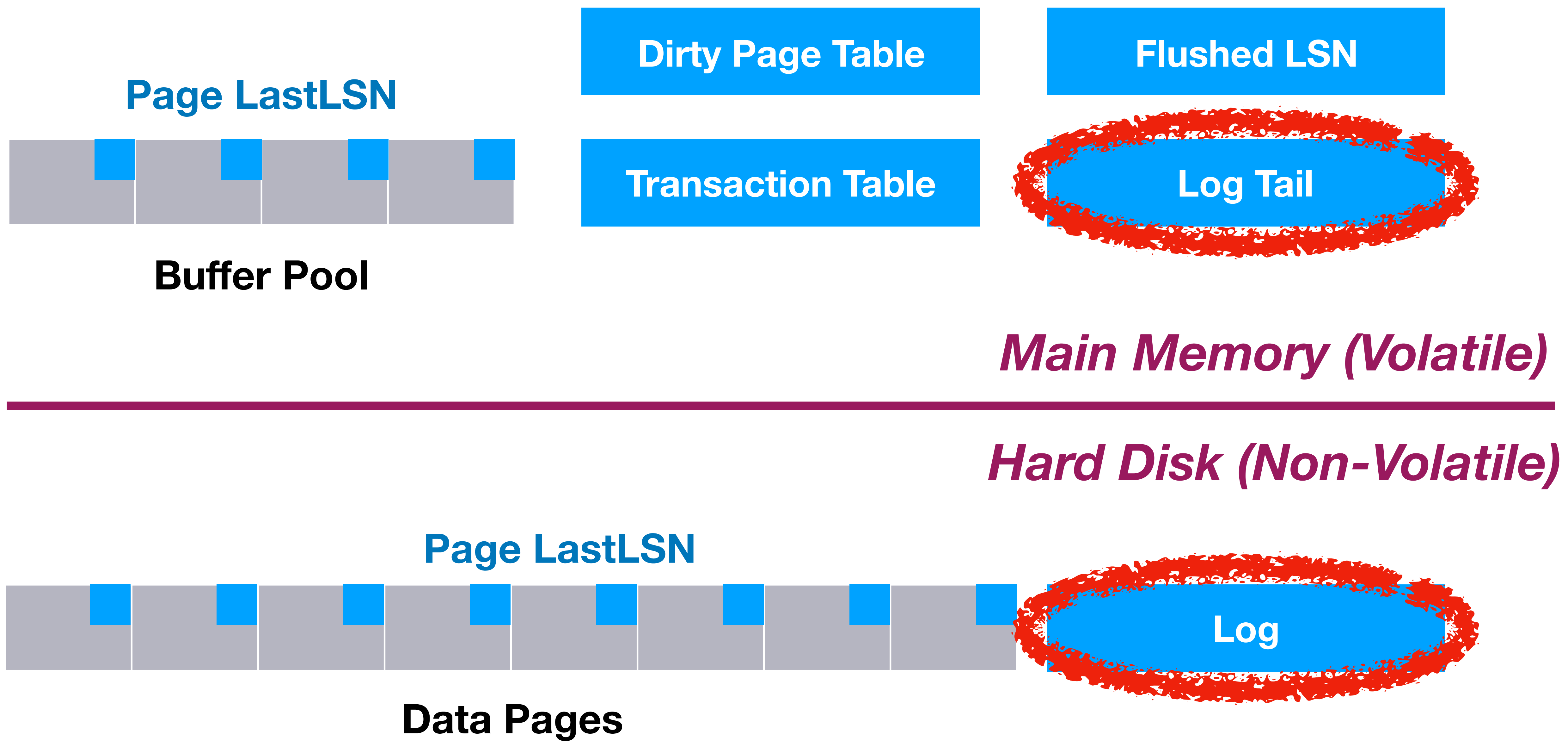
ARIES Data Structures



ARIES Data Structures



ARIES Data Structures



Types of Log Entries

- **Update**: states new and prior value after data update
  - New value for **redo**, old value for **undo**
- **Commit**: indicates that a transaction committed
- **Abort**: indicates that a transaction aborted
- **End**: indicates cleanup for transaction finished
- **Compensation**: indicates we undid prior operation
  - Must keep track in case of **crashes during recovery**

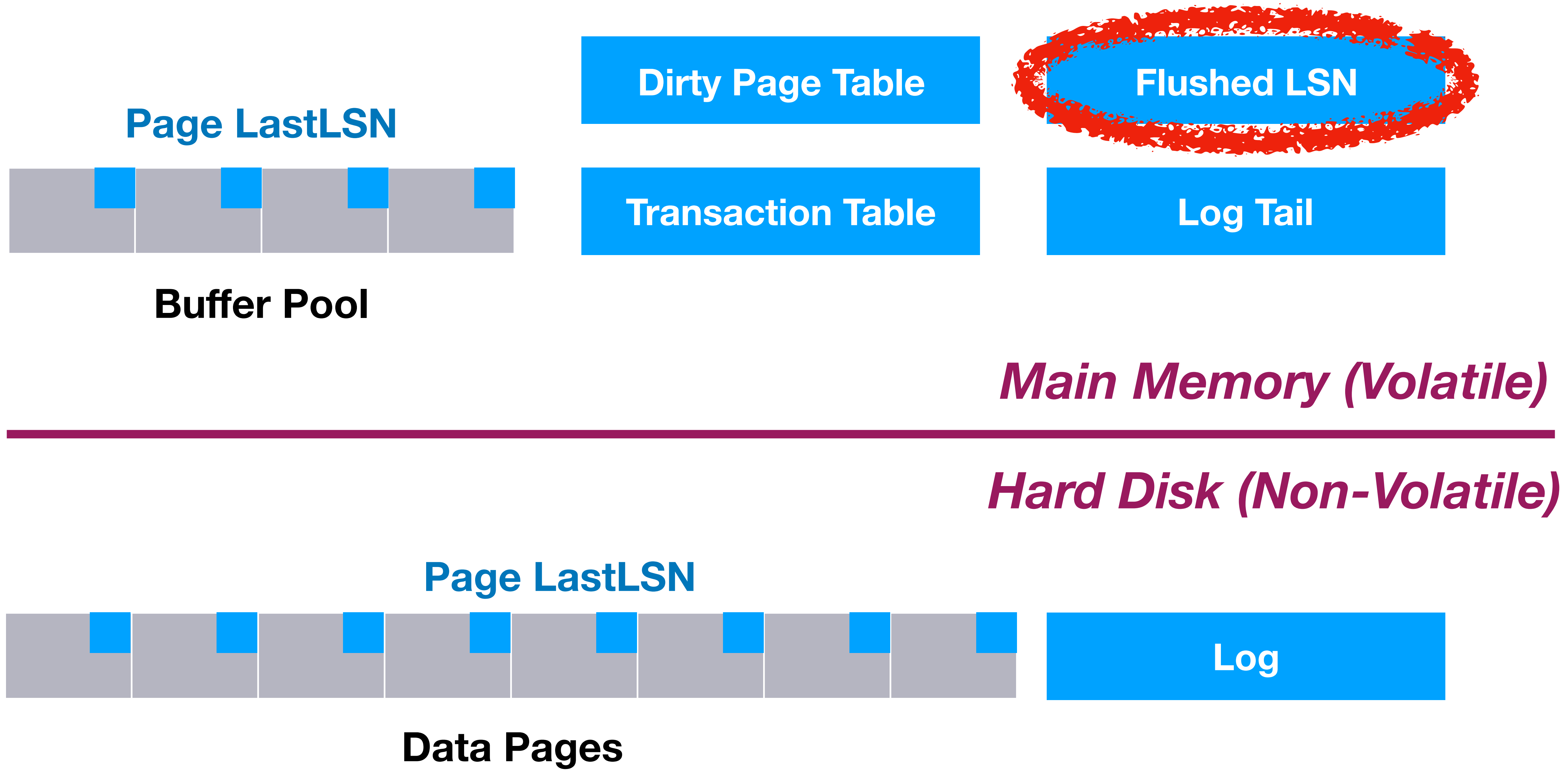
Generic Log Entry Fields

- Each log entry has an ID, the log sequence number (**LSN**)
- **TransID**: this transaction generated the log entry
- **PrevLSN**: LSN of previous entry for same transaction
- **Type**: type of log entry (see previous slide)

Added Fields for Updates

- **PageID**: logging update that refers to this page
- **Length**: so many bytes were changed by update
- **Offset**: first byte on page affected by update
- **Before-Image**: original value before the update
- **After-Image**: new value after the update

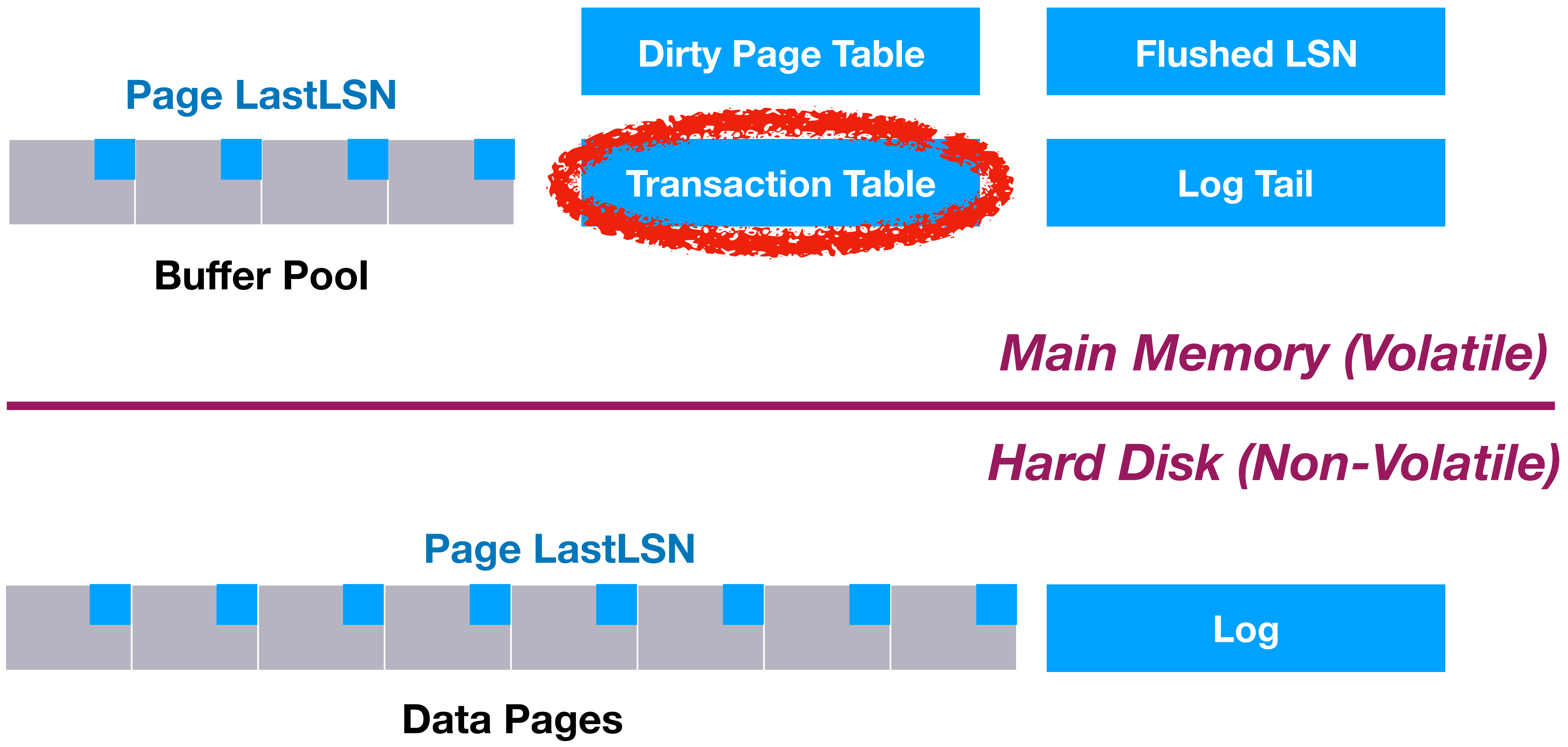
ARIES Data Structures



Flushed LSN

- FlushedLSN: log entries persistent **up to** this entry
- Can exploit to verify rules of **write-ahead** logging
- Must persist transaction log entries **before commit**
  - Must have **flushedLSN ≥ transaction lastLSN**
- Must persist log entries about page **before disk write**
  - Must have **flushedLSN ≥ page's pageLSN**

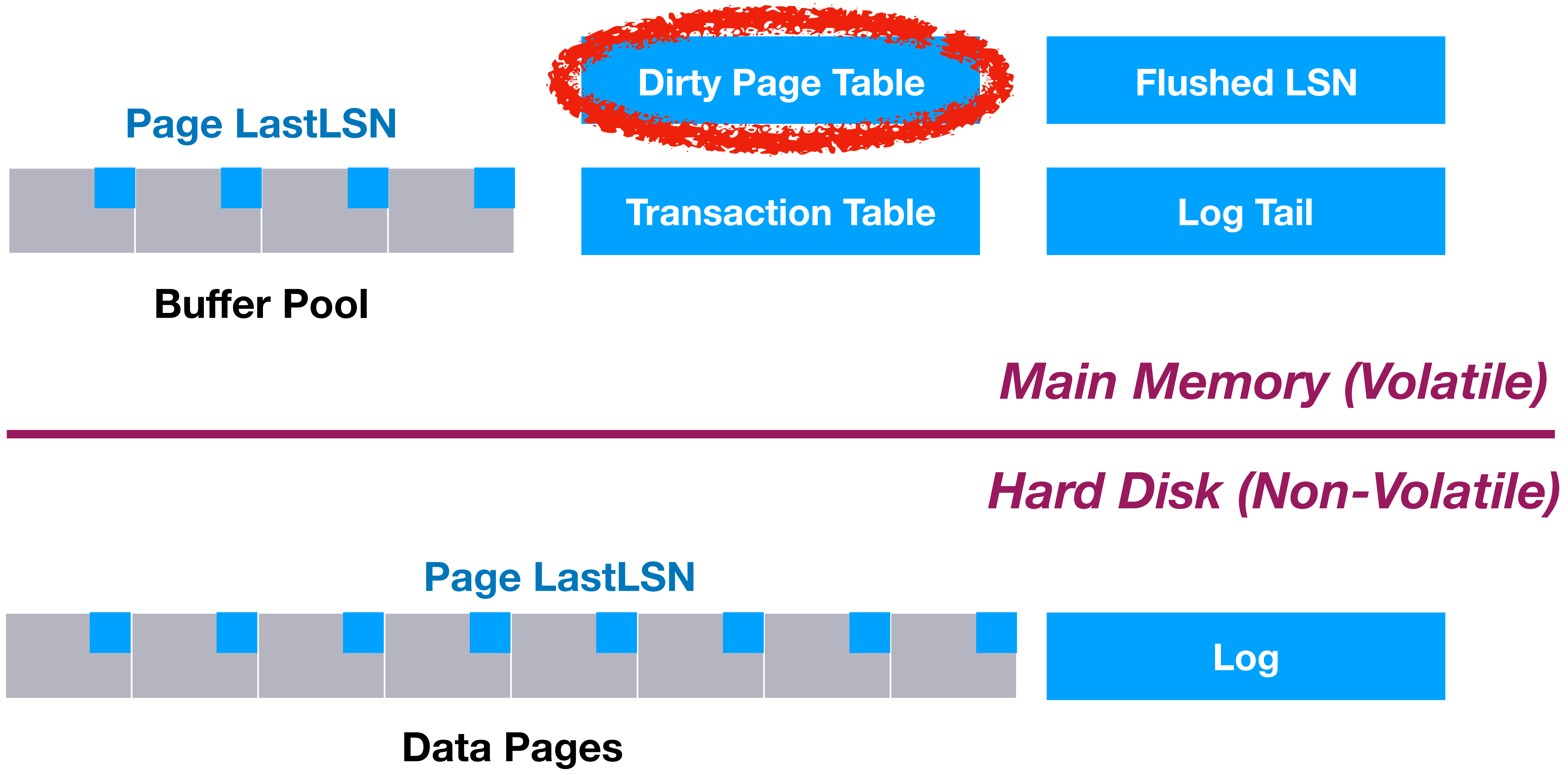
ARIES Data Structures



Transaction Table

- Contains one entry for each **active transaction**
- Stores for each transaction **three fields**:
  - **TransID**: transaction ID
  - **Status**: running/committed/aborted
  - **LastLSN**: ID of last log entry by that transaction

ARIES Data Structures



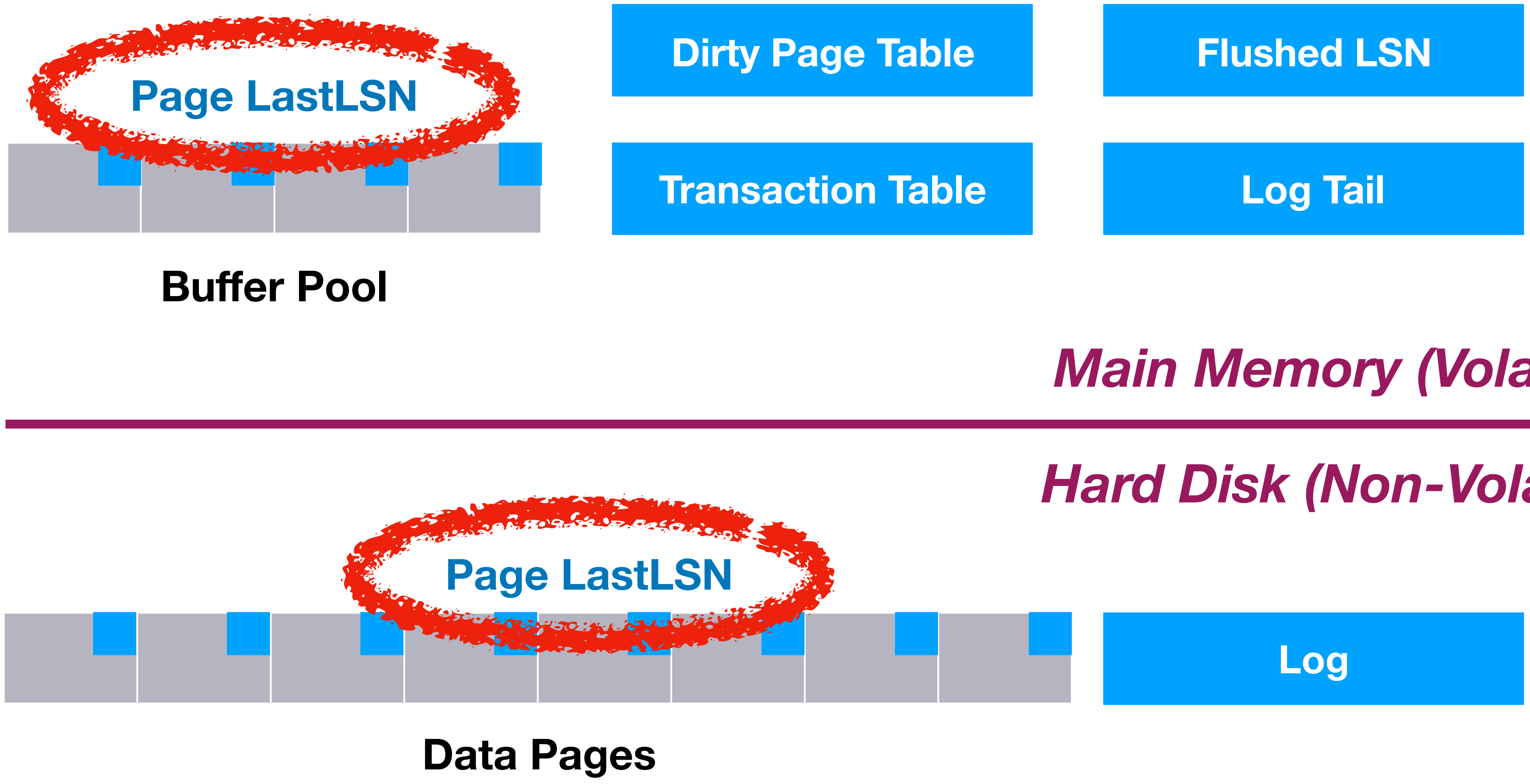
Dirty Page Table

- **Dirty page**: in-memory version differs from disk version
  - This means **changes would be** lost by crash
- **Dirty page table** stores one entry per dirty page, storing
  - **PageID**: ID of dirty page
  - **RecLSN**: LSN of first log entry making page dirty

Dirty Page Table

- **Dirty page**: in-memory version differs from disk version
  - This means **changes would be** lost by crash
- **Dirty page table** stores one entry per dirty page, storing
  - **PageID**: ID of dirty page
  - **RecLSN**: LSN of first log entry making page dirty

ARIES Data Structures



Page LastLSN

- The LSN of the **last operation** changing that page
- Stored for **each page** in memory and each page on disk
- LasLSN of disk and memory version of page may **differ**

How Does the PageLSN of In-Memory and Disk Version Relate for Dirty Pages?