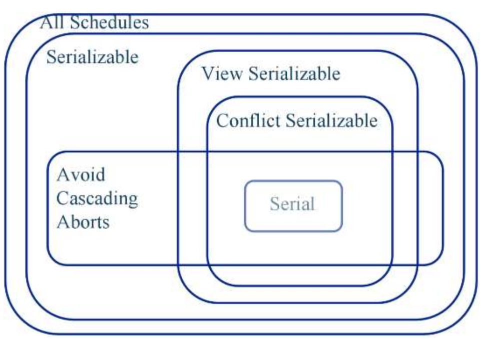
CS186 Final Cheat Sheet

<u>Transactions & Currency Control:</u>

- **DBMS:** database management system manages the queries
- **Concurrency control** provides correct and fast data access in the presence of concurrent work by many users
- **Recovery** ensures database is fault tolerant
- **Concurrent execution arguments:**
  - **Throughput** argument: increases TPS (transactions/second)
  - **Latency** argument: latency not dependent on another unrelated transaction
- Inconsistent/dirty reads and inconsistent updates could happen
- A **transaction** is a collection of operations that form a single logical unit (one atomic unit of work) (begin, SQL statements, end)
- **ACID:** *Atomicity*: all or none, *consistency*, starts and ends consistent, *isolation*: execute each Xact from others, *durability*: commit persists
- **Serial schedules:** each transaction runs from start to finish without any intervening actions from other transactions
- 2 schedules are **equivalent** if they involve same xact, each individual xact actions are ordered the same and leave the db in same final state
- **Serializable** if s is equivalent to any serial schedule
- **Conflict equivalent**: same actions, conflicts ordered same way
- **Conflict serializable:** can be equal to serial schedules. Steps: draw dependency graph. If cycle, not conflict serializable.
- **View serializable** = conflict serial + blind writes. Hard to enforce

<u>Locking</u>

- **2PL** two phase locking is pessimistic, get all locks, then release all
  - Guarantees conflict serializability, doesn't prevent cascading aborts
  - **Strict 2PL** releases all locks at the same time



- **Lock manager** maintains a hashtable of locked objects, mode and wait queue
- Three ways of dealing with **deadlock:** prevention, avoidance, detection & resolution.
  - Prevention: use resource ordering (screen < network < print)
  - Avoidance: Priorities based on age (now – start_time)
  - **Wait-Die**: if Ti has higher priority, Ti waits for Tj; else Ti aborts
  - **Wound-Wait**: if Ti has higher priority, Tj aborts; else Ti waits
  - Detection: generates "waits for" graphs, and checks if cycles

<u>Recovery</u>

- Assume strict 2PL for recovery & in place updates
- **FORCE:** every update is on DB before commit
  - Durability without REDO, but poor perf
- **NO STEAL:** don't allow buffer-pool frames with <u>uncommitted</u> updates to be replaced
  - Atomicity without UNDO logging, poor perf
- **STEAL, NO-FORCE** is preferred policy

LogRecord fields:

- LSN
- prevLSN
- XID
- type
- update records only:
  - pageID
  - length
  - offset
  - before-image
  - after-image

	No Steal	Steal
No Force		Fastest
Force	Slowest	

Performance Implications

	No Steal	Steal
No Force	No UNDO REDO	UNDO REDO
Force	No UNDO No REDO	UNDO No REDO

Logging/Recovery Implications

- **WAL (Write Ahead Logging)** is the protocol that we use
  - Force <u>log record</u> before data page (Atomicity)
  - Force all log records for an X before commit (durability)
- WAL Info: **LSN**: unique and increasing, **flushedLSN** (last logged LSN), **pageLSN**: points to an LSN from the DB. pageLSNi <= flushedLSN
- Possible log records: **update**, **commit**, **abort**, **checkpoint** (log maintenance), **CLRs** (compensation log records – never undone), **end** (end of commit or abort)
- Two in-memory tables for recovery:

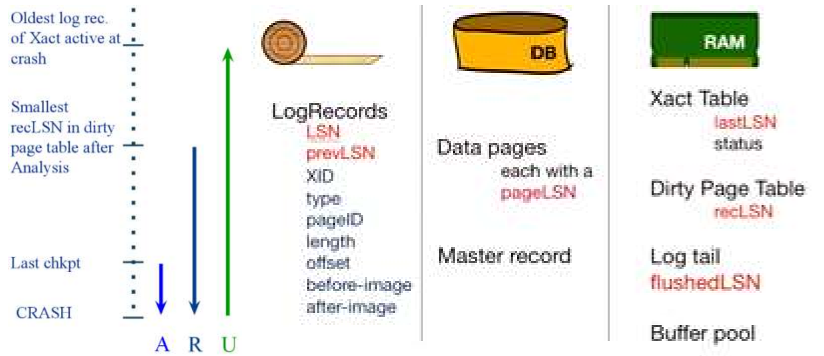
Transaction Table

<u>XID</u>	Status	lastLSN
1	R	33
2	C	42

Dirty Page Table

<u>PageID</u>	recLSN
46	11
63	24

  - Xact table: **lastLSN** (most recent LSN written)
  - Dirty Page table: **recLSN** LSN that <u>*first*</u> caused the dirty page



- Assume disk write is atomic. Strict 2PL WAL.
- **Checkpoints** have Xact and dirty page table
- 3 phases for crash recovery:
  - **Analysis** figure out which Xacts committed since checkpoint, which failed.
  - **REDO** all actions (repeat history)
  - **UNDO** effects of failed Xacts
- **Analysis:**
  - Reestablish knowledge of checkpoint state
  - **End** recs: remove Xact from Xact table
  - **Update** recs: if page P not in Dirty Page Table, Add P to DPT, set its recLSN=LSN
  - **!End** recs: Add Xact to Xact table, set lastLST=LSN, change Xact status on commit
  - At end, for any Xacts in the Xact table in Committing state, generate a corresponding END log record, and remove Xact from Xact table
  - Xact table says which xacts were active at last log flush before crash
  - DPT says which dirty pages <u>might not</u> have made it to disk
- **REDO:**
  - Reapply all updates (even aborted Xacts), redo CLRs.
  - Scan forward from log rec containing smallest <u>recLSN</u> in DPT
  - For each update log record or CLR with LSN, REDO action unless:
    - Affected page not in DPT
    - Affected page in DPT but has recLSN>LSN
    - pageLSN (in DB) >= LSN (IO required)
  - Reapply action, set pageLSN to LSN (no log, no force)
- **UNDO:**

ToUndo={lastLSNs of all Xacts in the Xact Table} // a.k.a. "losers"

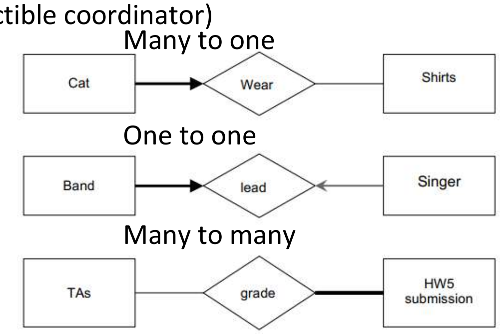
Repeat until ToUndo is empty:
- Choose (and remove) largest LSN among ToUndo.
- If this LSN is a CLR and undonextLSN==NULL
  - Write an End record for this Xact.
- If this LSN is a CLR, and undonextLSN != NULL
  - Add undonextLSN to ToUndo
- Else this LSN is an update. Undo the update, write a CLR, add prevLSN to ToUndo.

- **Recovery Manager** guarantees Atomicity & Durability

<u>Replication Consistency and NoSQL</u>

- Why replicate? Doesn't help with data scaling (this is what sharding is for), does help with workload scaling (load balancing)
- Replication increases availability (reduced latency)
- **Single master replication**: both sides handle full update volume
- **Single master log-shipping**: Log is shipped to secondary nodes
- **Multi-master**: 2PC can cause high latency, no 2PC write conflicts
- Replication granularity can be at DB, table, partition or tuple level
- If coordinator fails: participants hold locks indefinitely (not good)
- **Paxos** a consensus protocol like 2PC to get a group of nodes to agree on a proposal. Resilient to node failures (majority)
  - Replicate coordinator 2F + 1 replicas to tolerate F faults (virtually indestructible coordinator)

<u>Pre-Final Review</u>



- **Closure of attribute** is denoted with A+, says everything that can be determined by A.
- **BCNF** (3NF & x->y, x is super key). Ex. A -> BCD, BC -> AD, D -> B. D->B isn't a super key (separate table to ADC and DB)
- **Dependency preserving**: After decomposition, are the FD the same or not (do they fit with the new tables?)
- **Lossless**: Perform alpha on tables.