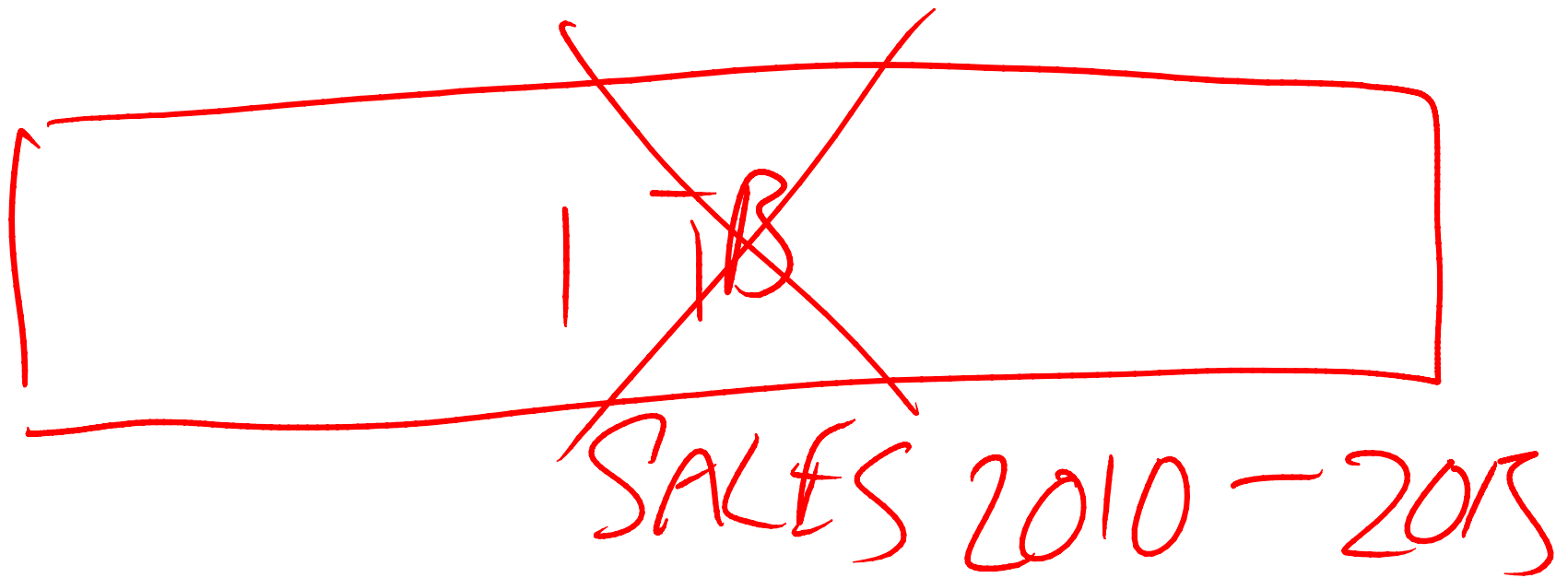
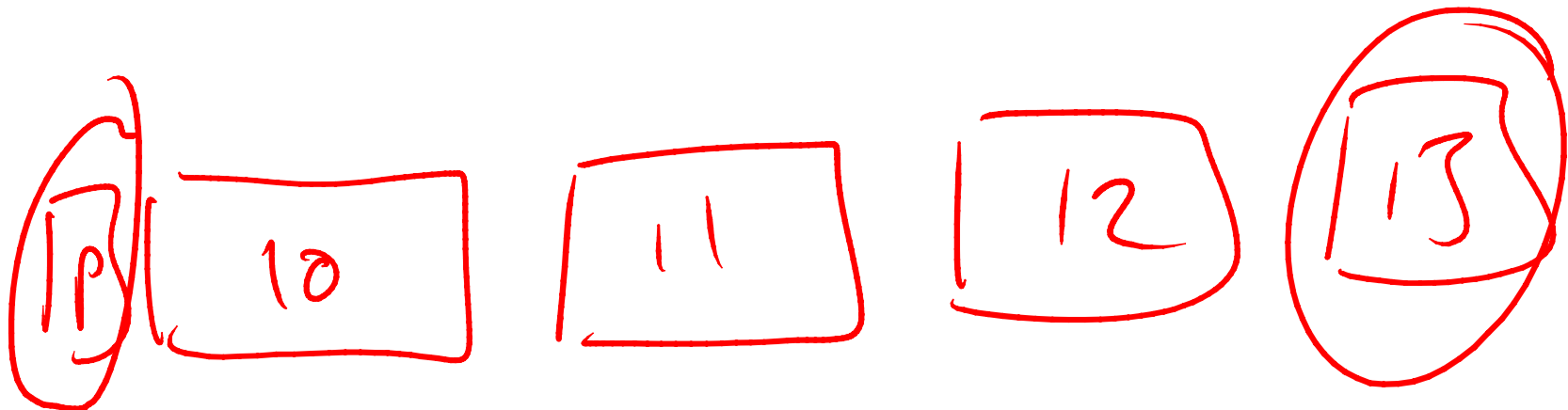
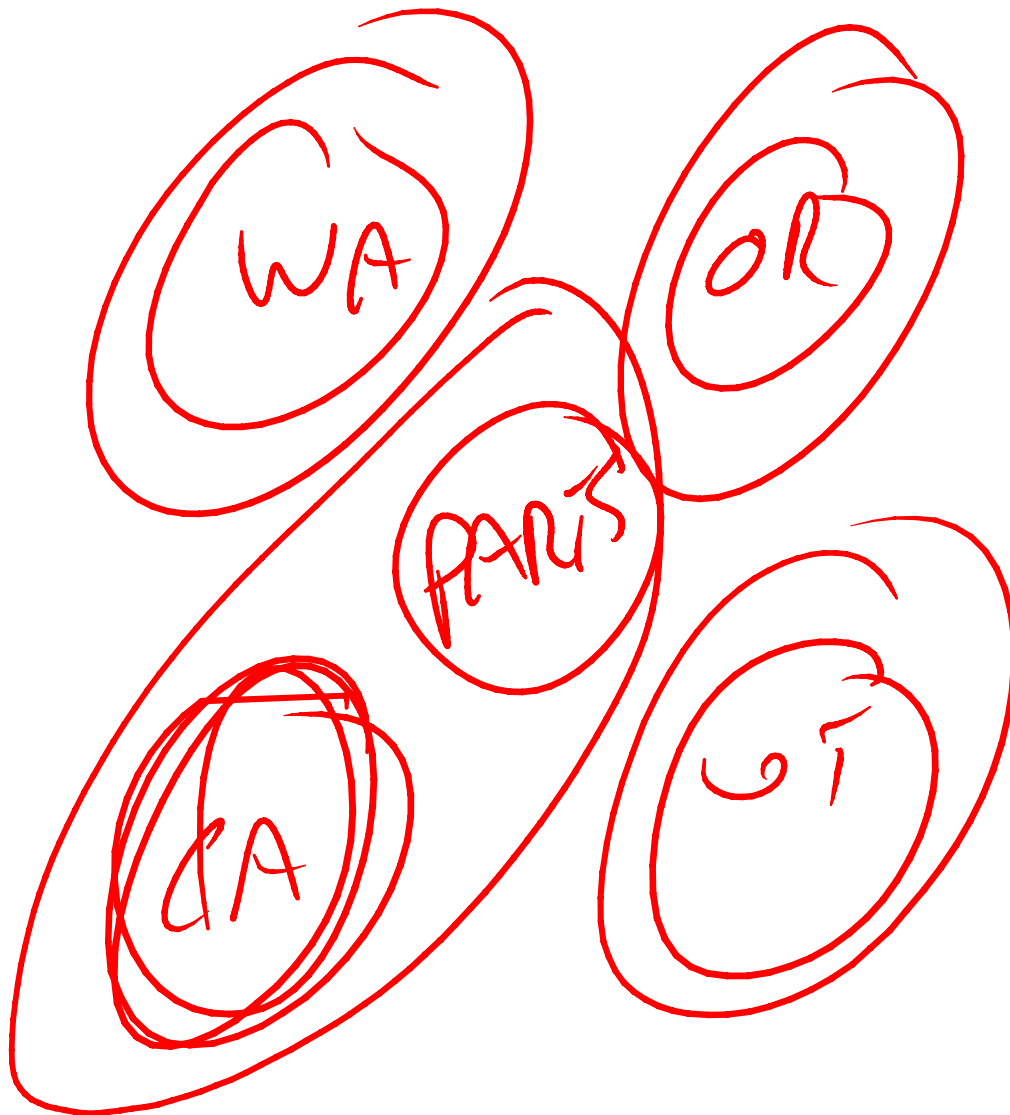




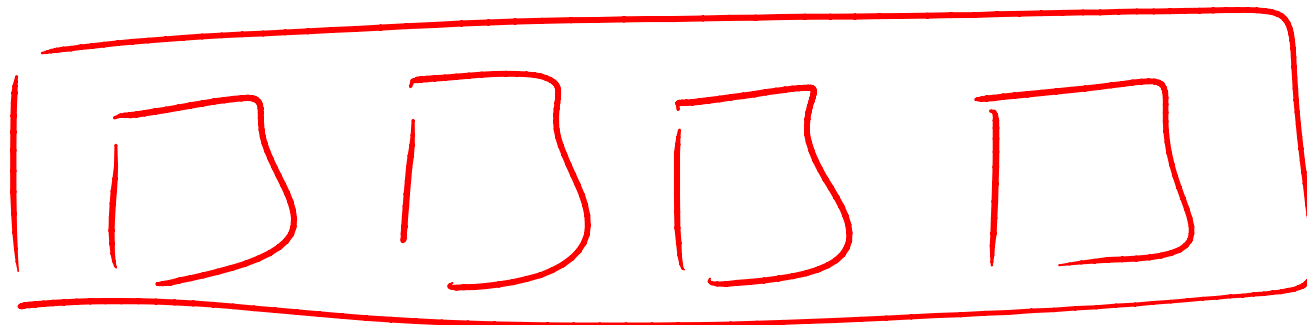
M1S7 Explaining that dividing a large database into filegroups allows targeted restores during disaster recovery, rather than having to wait for the entire single file database to restore.



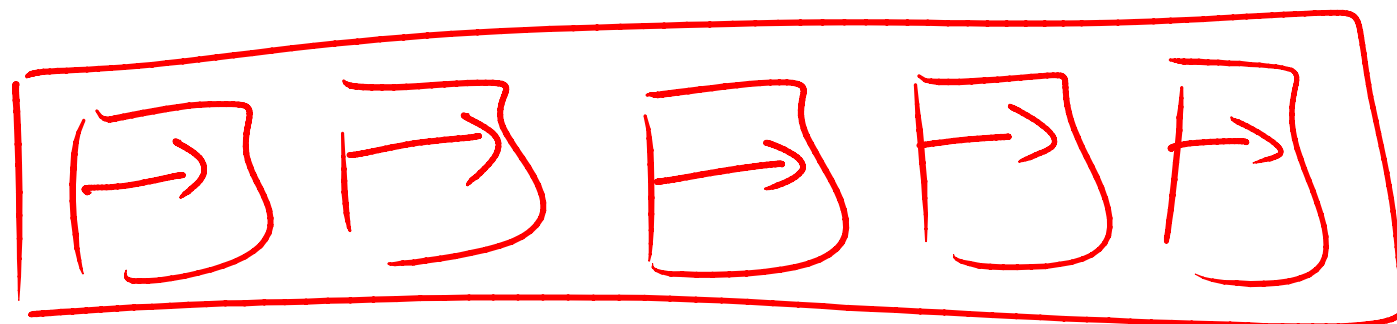


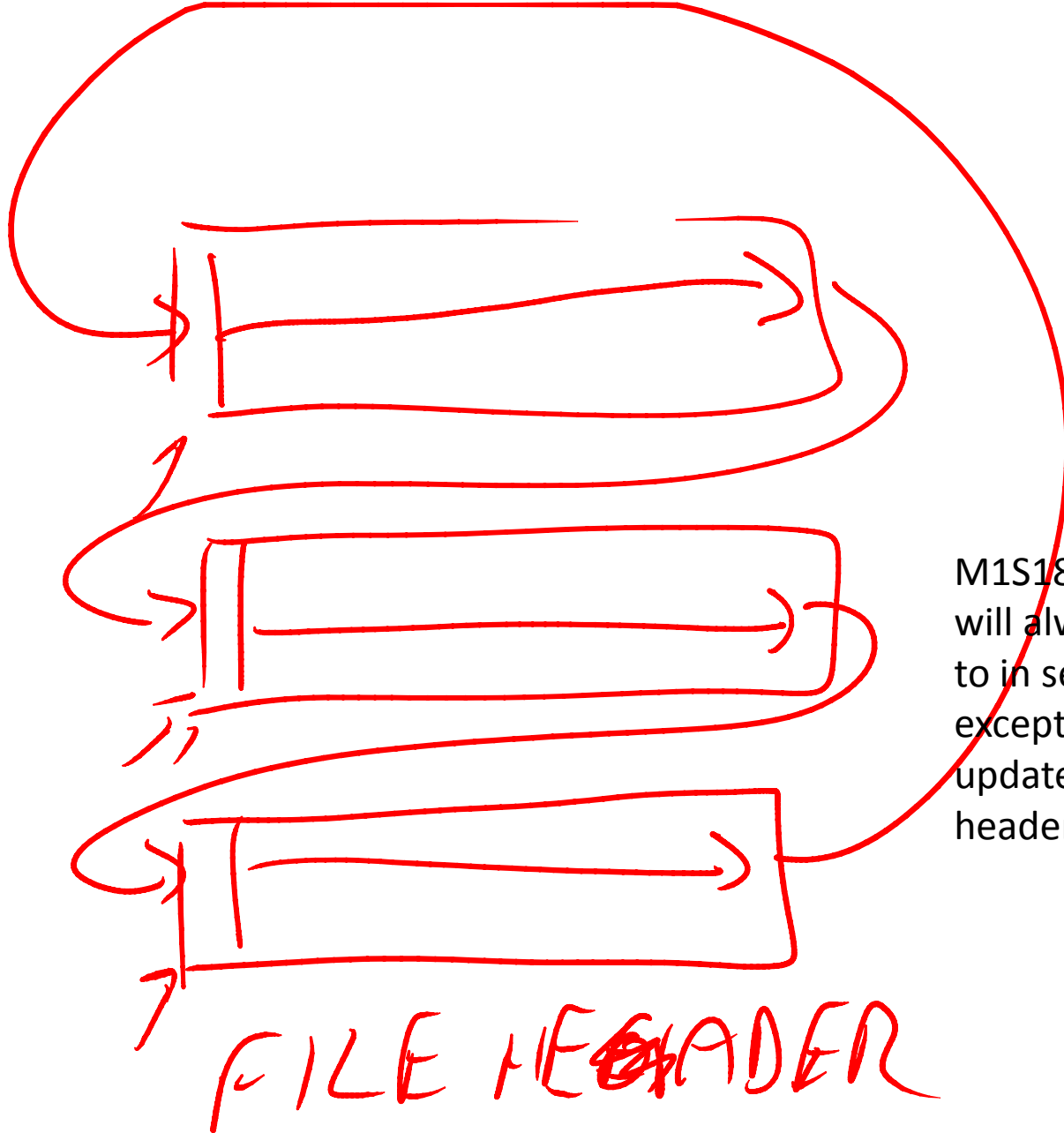
M1S7 Explaining a different partitioning scenario – auto parts sales to WA/CA/OR/UT.

If things go down, it makes sense to restore the portion of the database that will bring in the most sales quickly – CA, as it has the most population.

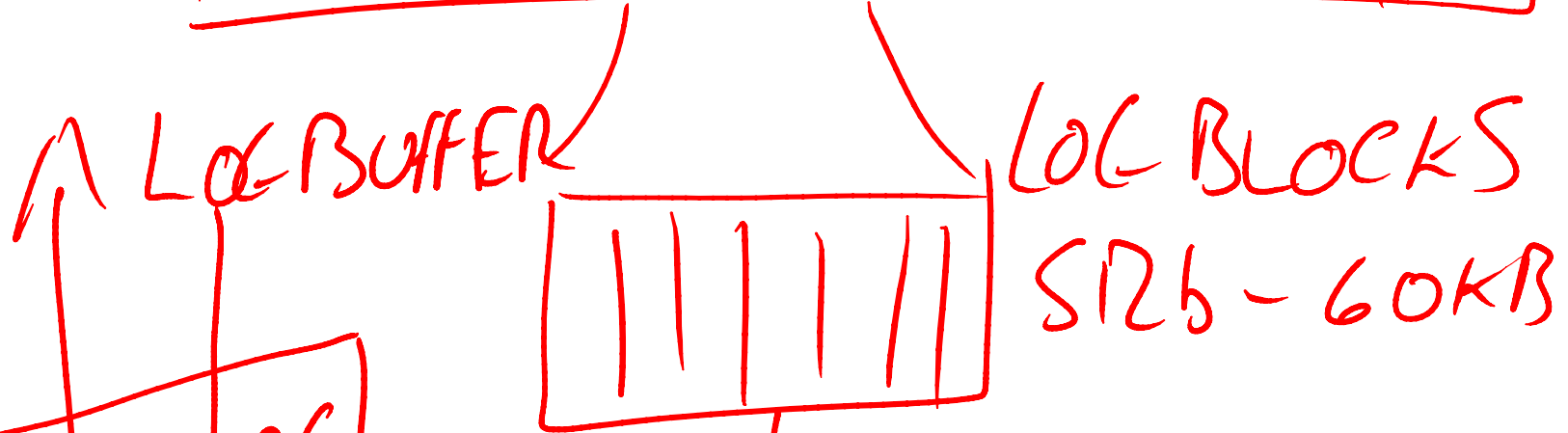
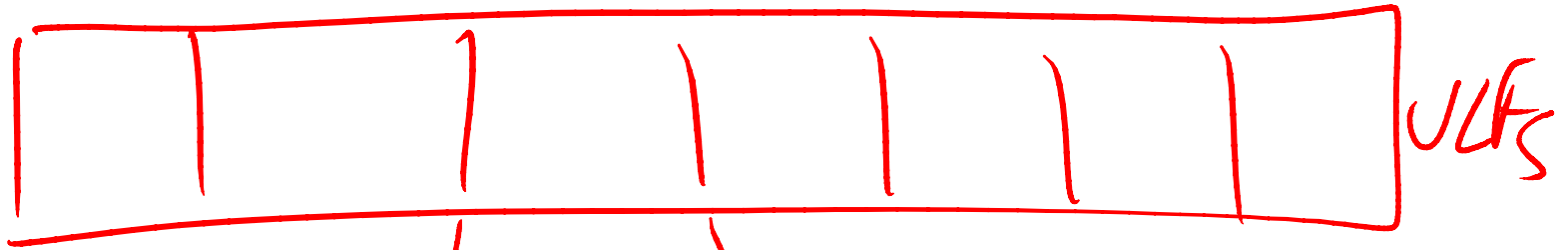


M1S7 Be careful about putting all the log files on a single volume. If they're all having activity, that creates multiple write points on the volume, making it a random I/O workload, even though each log is sequential-write only.





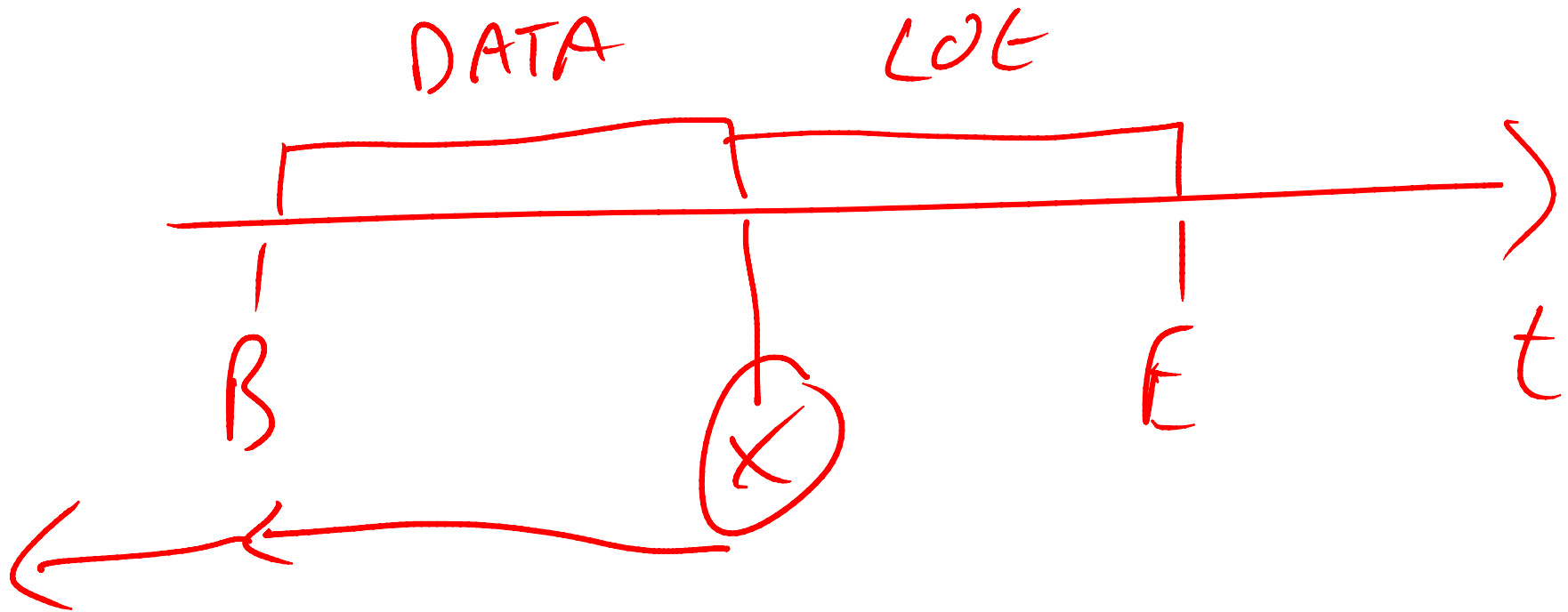
M1S18 Multiple log files will always be written to in sequential order, except for periodic updates to the file header pages.



WRITE LOG



ASYNC
WRITE LOGWRITER

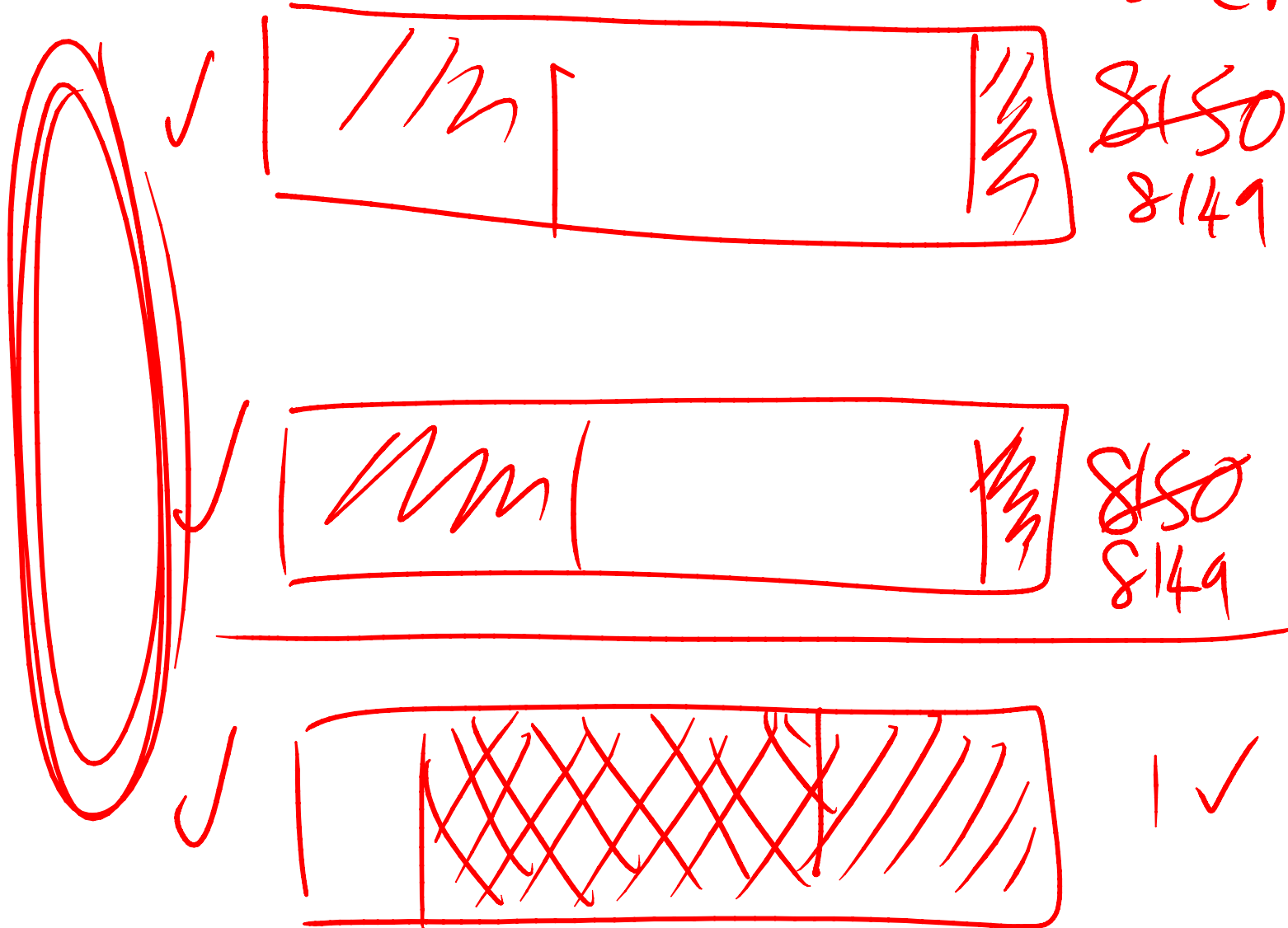


M1 Explaining how a data backup occurs. It starts at time B by doing a checkpoint, then reads the data in the backup, finishing at time X. From time X to time E, it's reading all the transaction log from time X back to at least time B, and possibly to a point in time earlier if there was an active transaction at time B. When you restore the backup and allow recovery to run, you get the database as of time X.

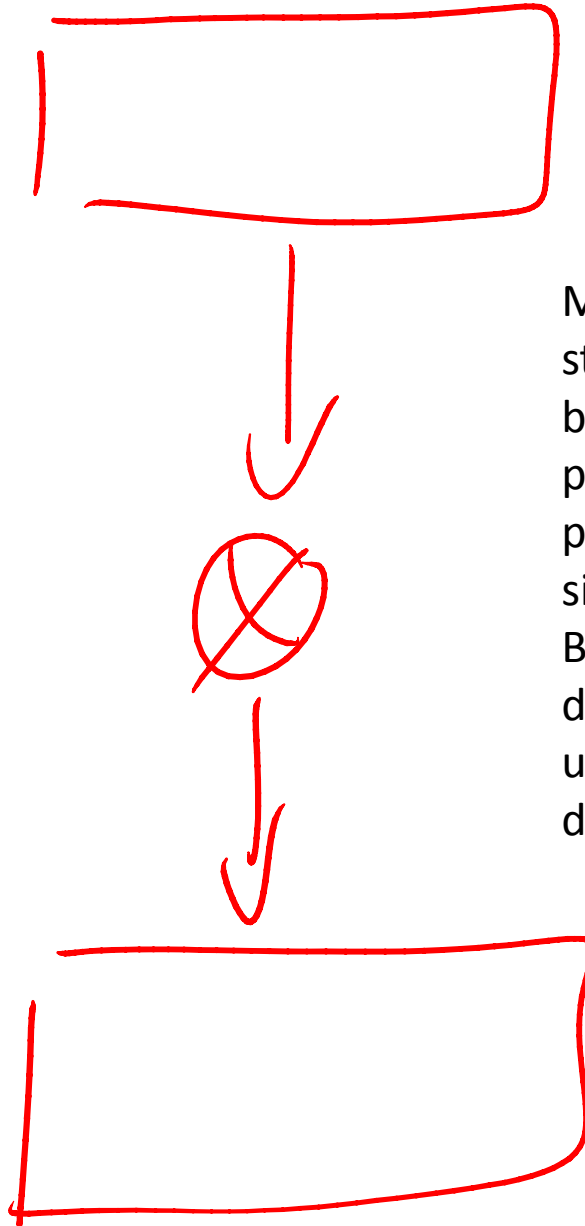
LOP_DELETE_SPLIT

M1 This is the log record type to watch for to show page splits occurring. See the blog post at http://www.sqlskills.com/blogs/paul/using-fn_dblog-fn_dump_dblog-and-restoring-with-stopbeforemark-to-an-lsn/

WEIGHTING



M1S22 Explaining proportional fill and round robin. Adding another file to a full filegroup does not allow for rebalancing of data.

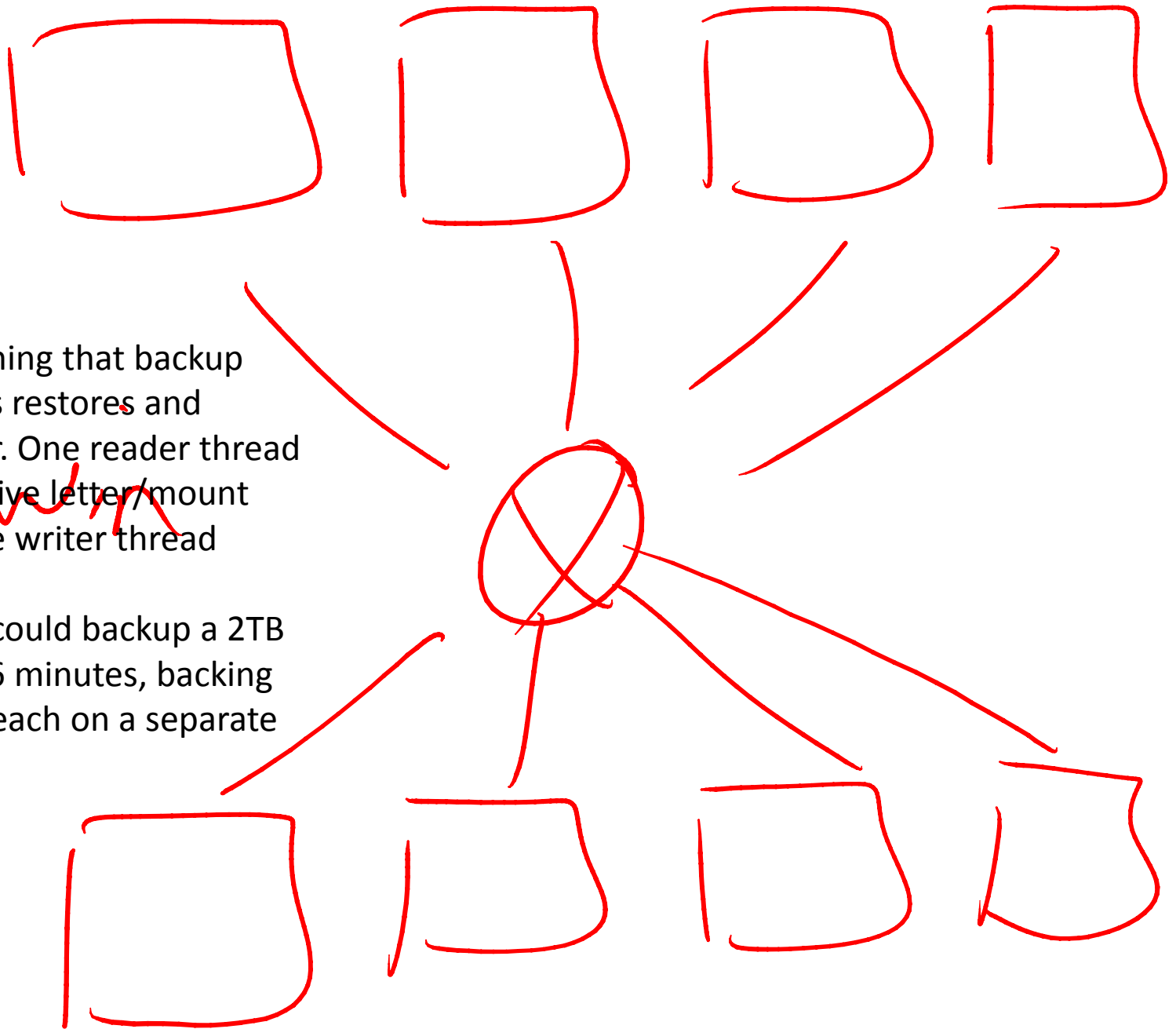


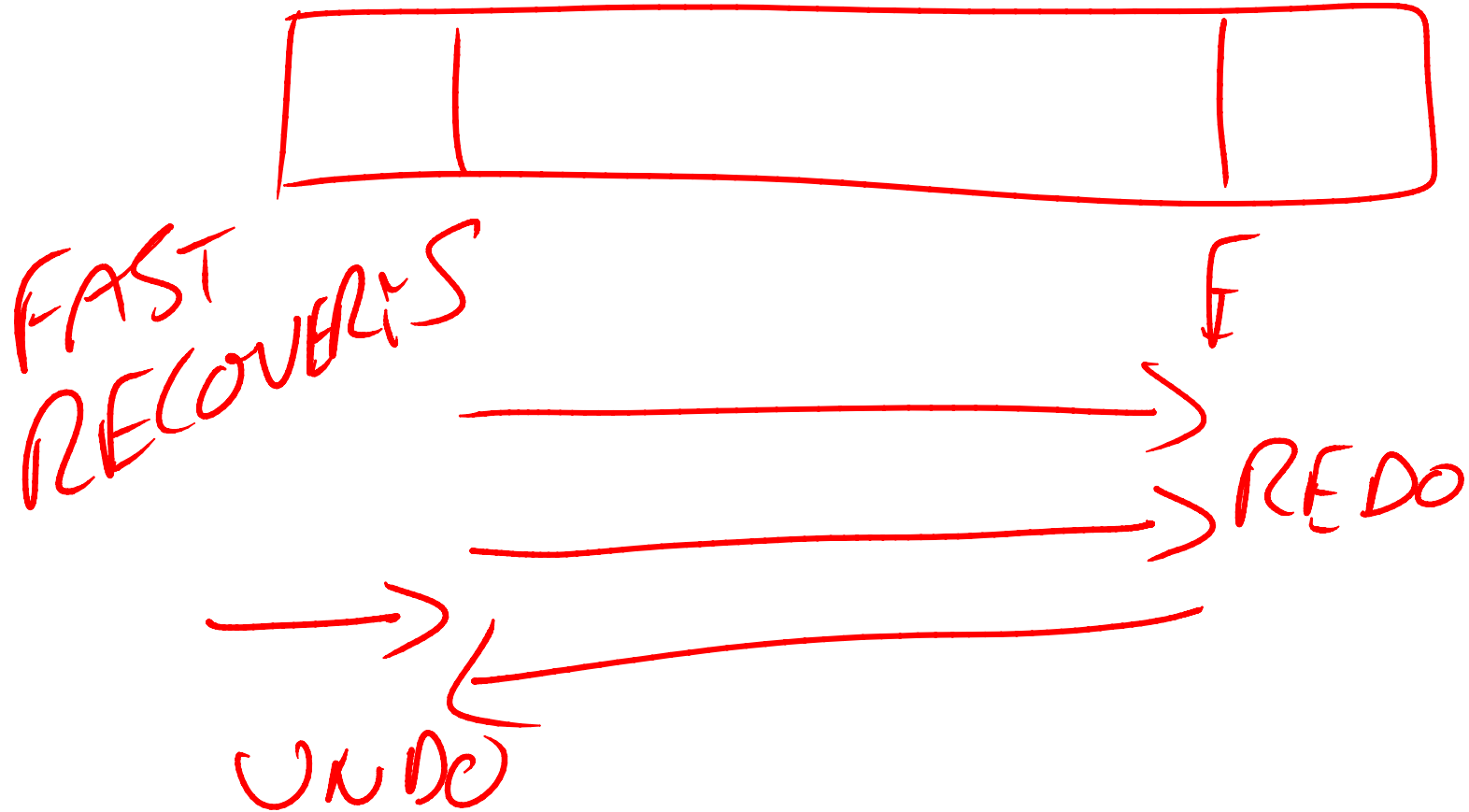
M1S33 Explaining that backup striping makes restores and backups faster. One reader thread per distinct drive letter/mount point, and one writer thread similarly.

Bwin in 2005 could backup a 2TB database in 36 minutes, backing up to 12 files each on a separate drive.

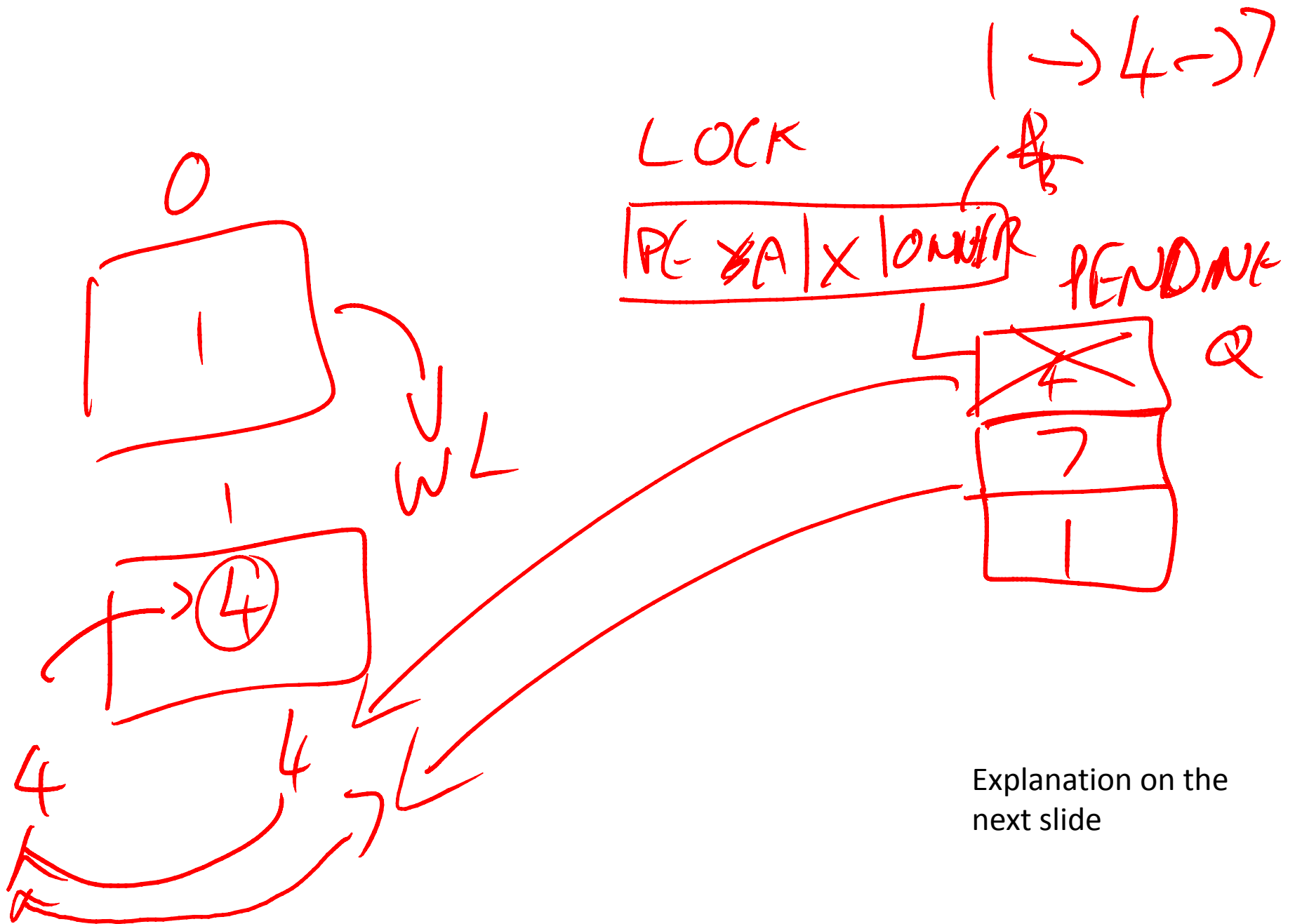
M1S33 Explaining that backup striping makes restores and backups faster. One reader thread per distinct drive letter/mount point, and one writer thread similarly.

Bwin in 2005 could backup a 2TB database in 36 minutes, backing up to 12 files each on a separate drive.





M1 Explaining fast recovery. Three passes through the log, analysis, REDO, and UNDO. Fast recovery lets you in after REDO (when crash recovery and failover occur, not during restore).



Explanation on the next slide

M5 Explaining how the lock pending queue works.

At time:

X: Thread 1 holds the lock

X+1: Thread 4 tries to acquire the lock and can't. It registers a callback routine for itself on the pending queue in the lock value block. Then it suspends itself, moves off the CPU and onto the waiter list.

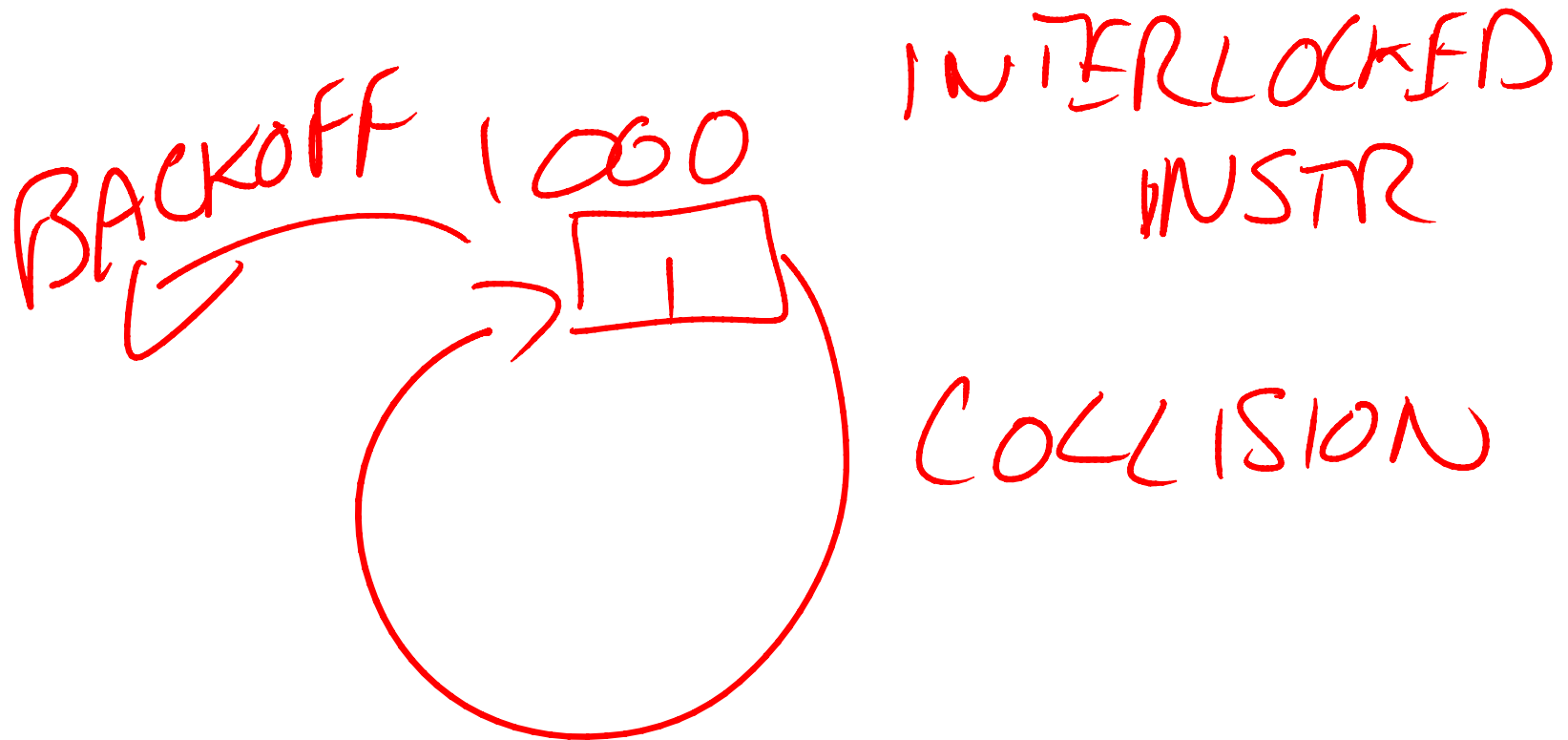
X+2: Thread 7 tries to acquire the lock and can't. It registers a callback routine etc etc, and it behind thread 4 in the pending queue.

X+3: Thread 1 releases the lock, which grants the lock to thread 4. Thread 4's callback routine is called, signaling it and allowing it to move to the runnable queue on it's scheduler. Thread 7 is now at the head of the pending queue for the lock.

X+4: Thread 4 releases the lock, which grants the lock to thread 7. Etc etc. There is now no pending queue for the lock.

X+5: Thread 1 tries to acquire the lock again and can't, so registers the callback routine, goes on the pending queue, and suspends onto the waiter list.

SPINLOCKS



M5 Explaining how spinlocks work. Code does a test-and-set-if-clear on the memory that implements the spinlock. If it can't get it, it spins (tries again) up to 1000 times and then backs off. `SOS_SCHEDULER_YIELD` is the wait type when a backoff occurs.

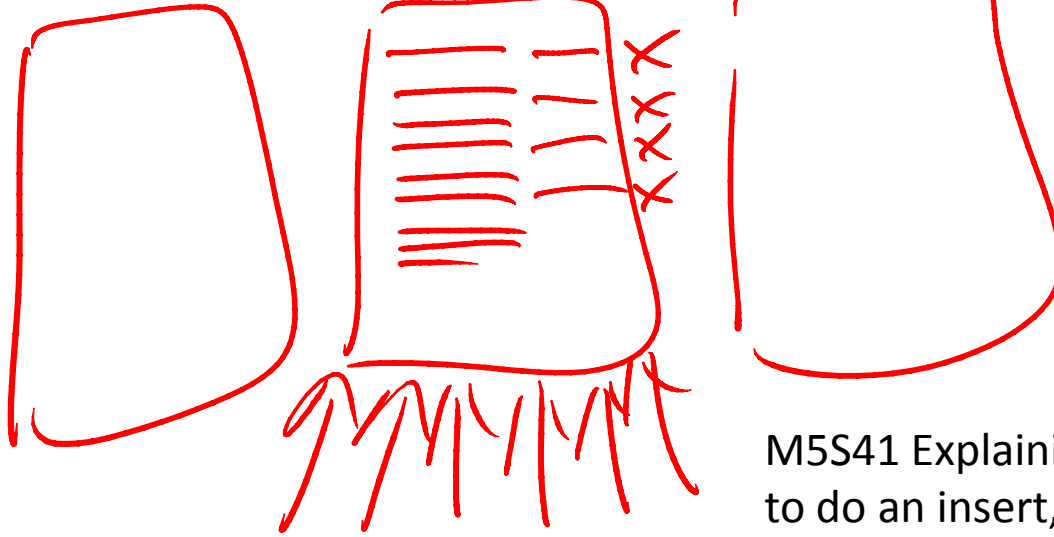
TIX

TIX = table IX lock

PIX = page IX lock

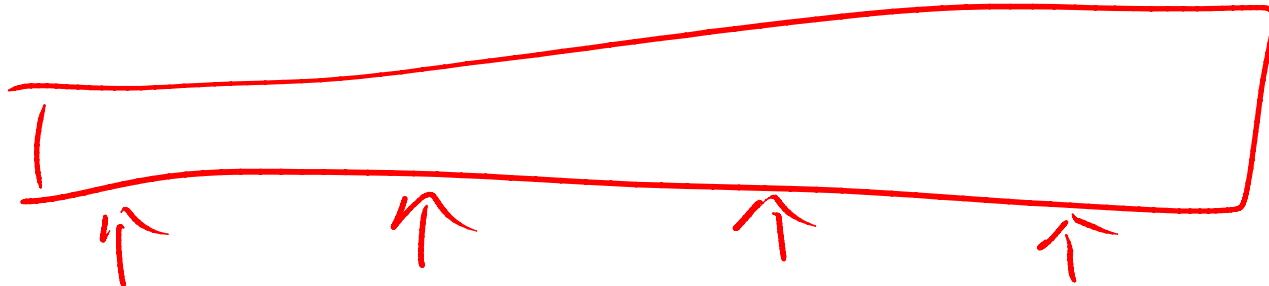
And there are row X locks

SOPIX



XXX
LATCH

M5S41 Explaining that to access a page to do an insert, multiple concurrent threads can hold non-conflicting row X locks, but they all need to acquire the X latch on the page, which becomes a bottleneck.



Solve that problem by creating multiple insert points, with a GUID or hash partitioning.