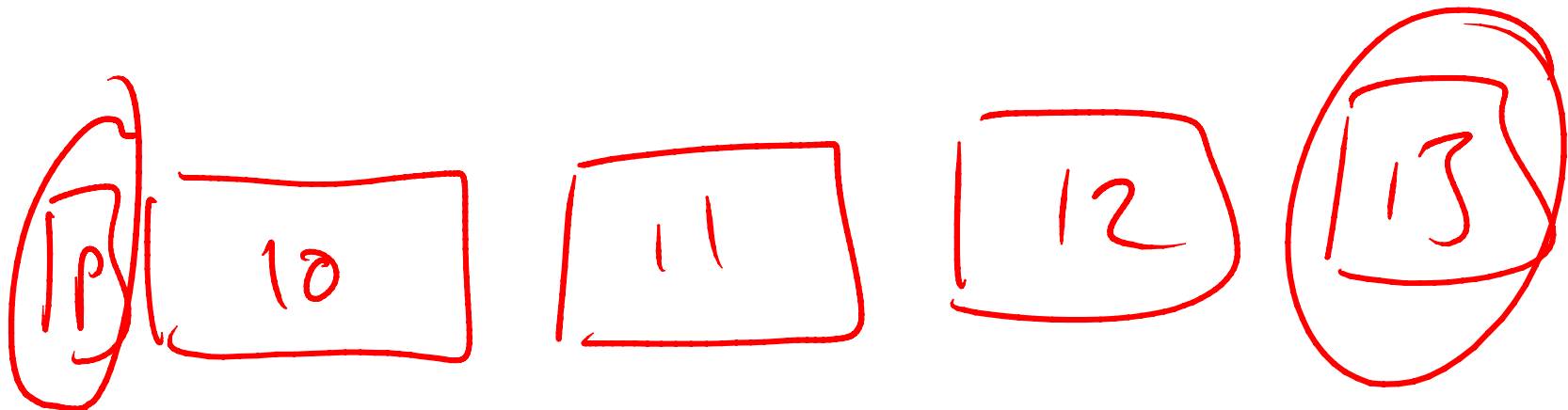
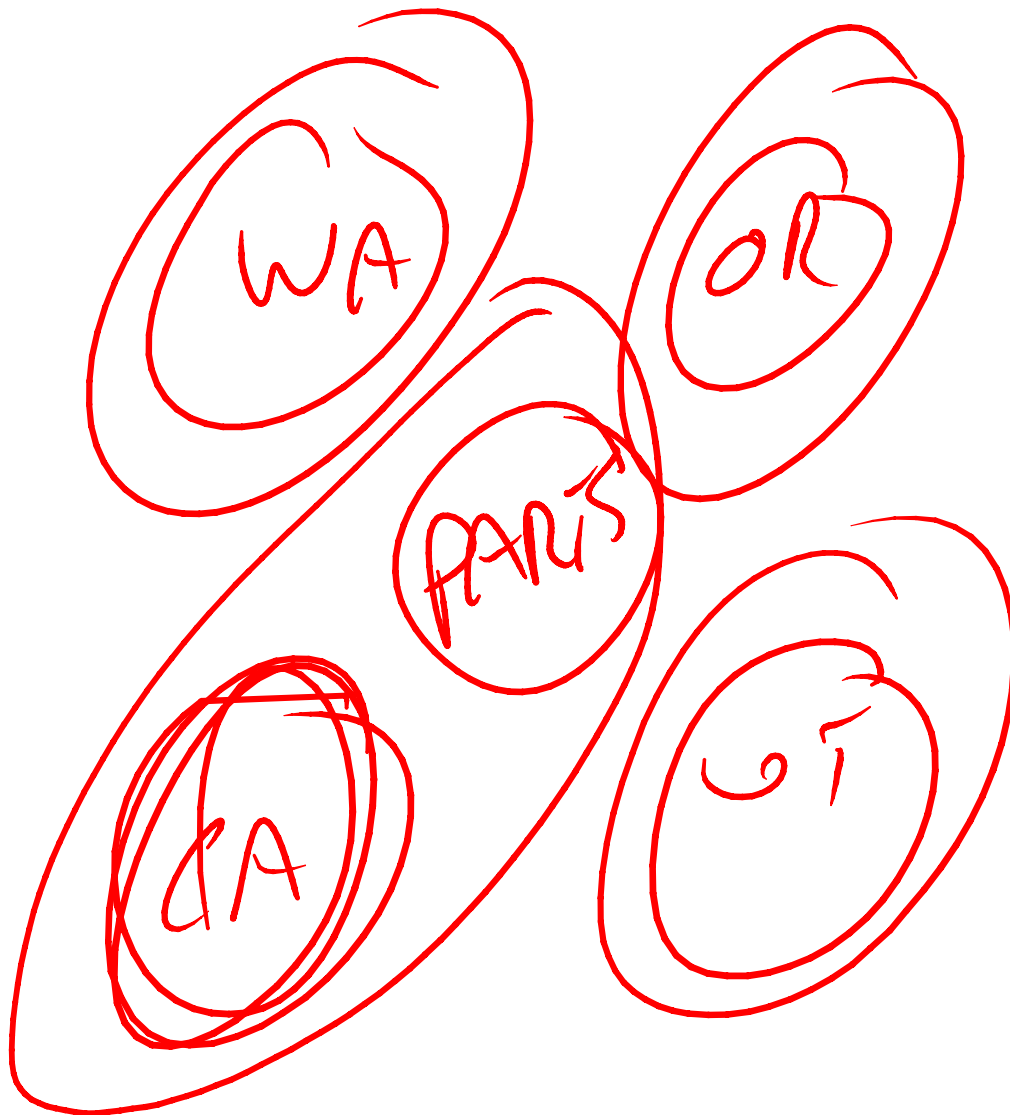




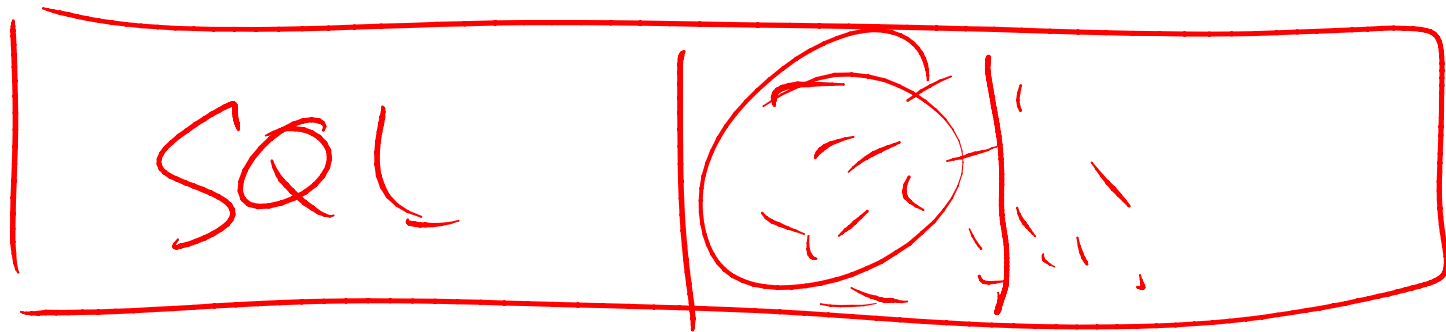
M1S7 Explaining that dividing a large database into filegroups allows targeted restores during disaster recovery, rather than having to wait for the entire single file database to restore.





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.

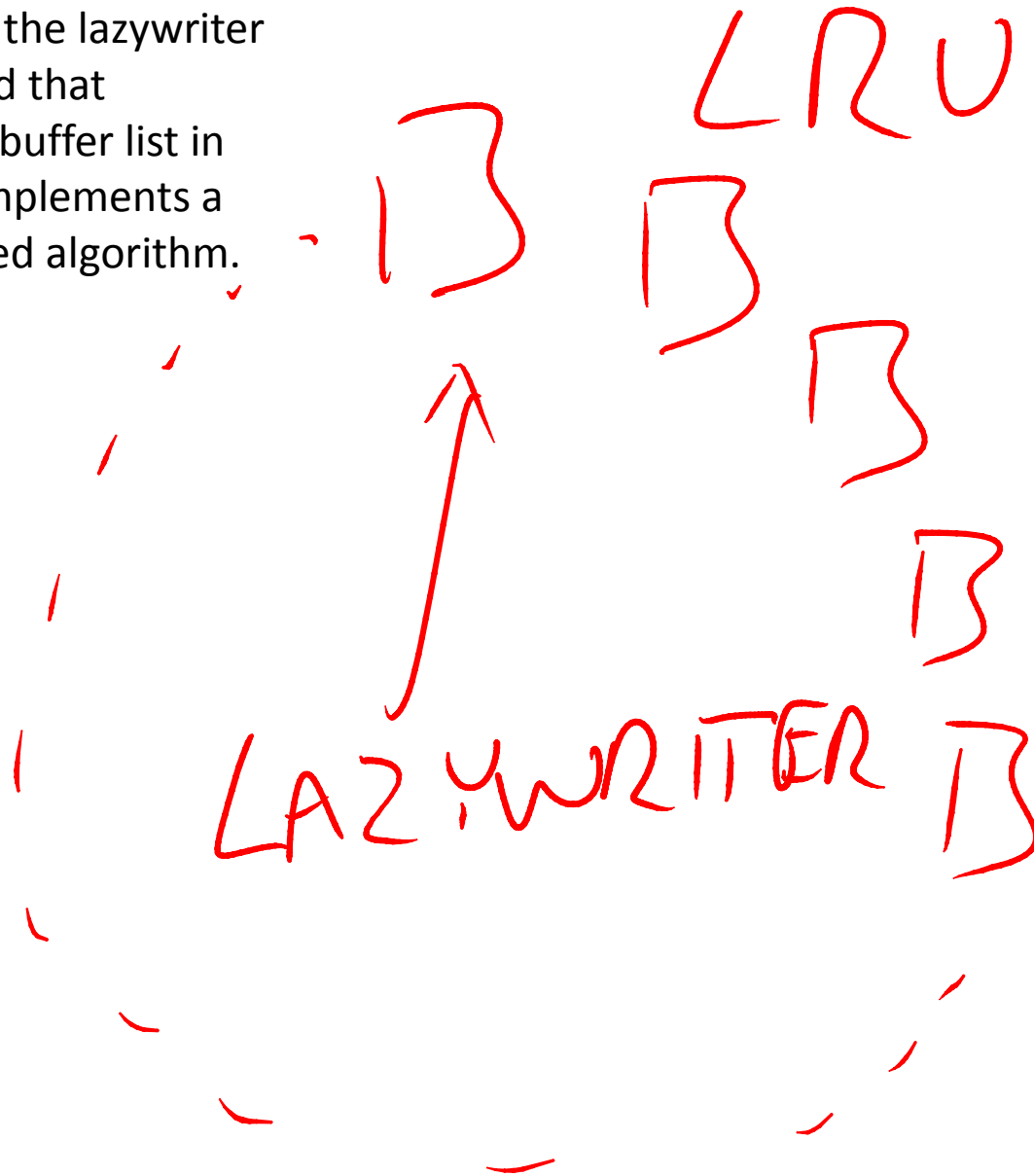


SA

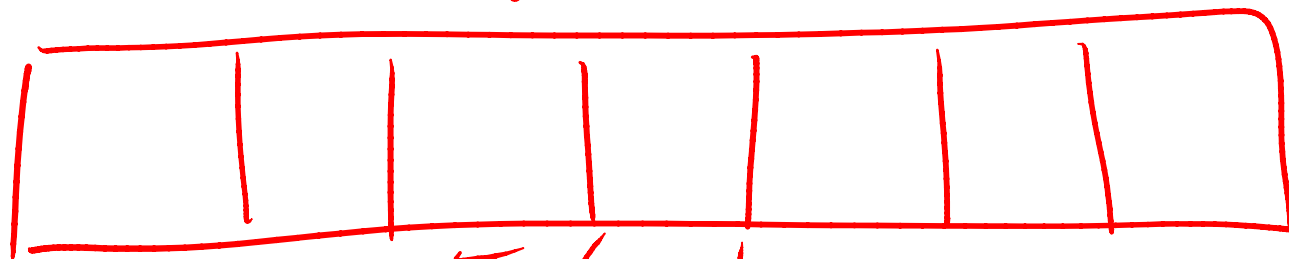
DBCC PAGE

M1S14 Explaining why instant file initialization is off by default. Sharing a volume between SQL Server and 'secure files' can lead to information leakage if the secure files are deleted, then SQL grows a data file, and an SA can use DBCC PAGE to get a hex dump of the bits/bytes that comprised the deleted secure file. If you don't have this situation, turn instant file initialization on.

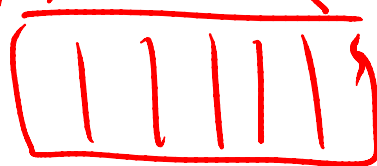
M1S31 Explaining the lazywriter background thread that manages the free buffer list in the buffer pool. Implements a Least Recently Used algorithm.



VLFs



LOG_COMMIT
- TRAN
x ACT



SIZE - 60KB

LOG BUFFER

WRITELOG



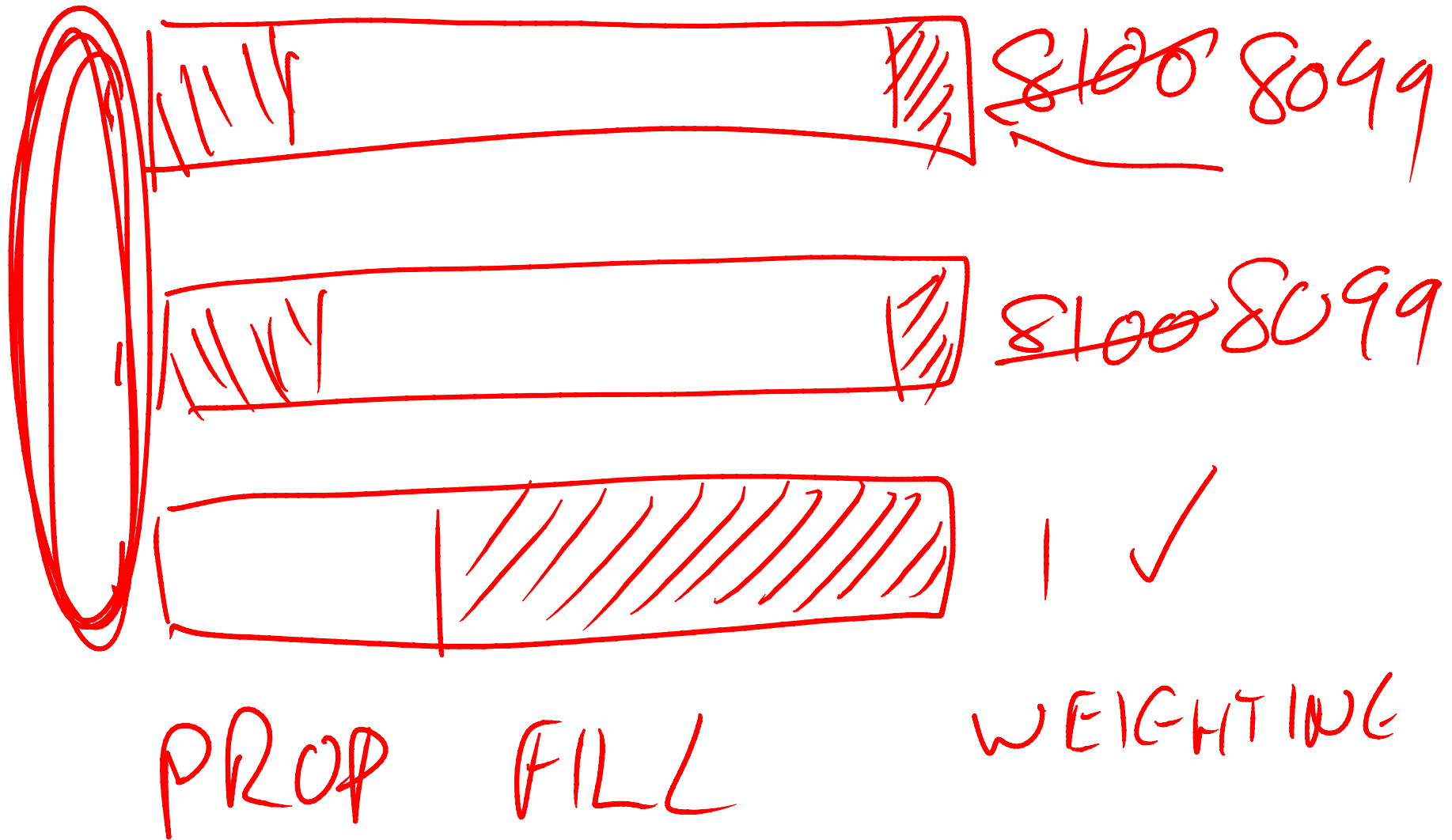
128

3840K

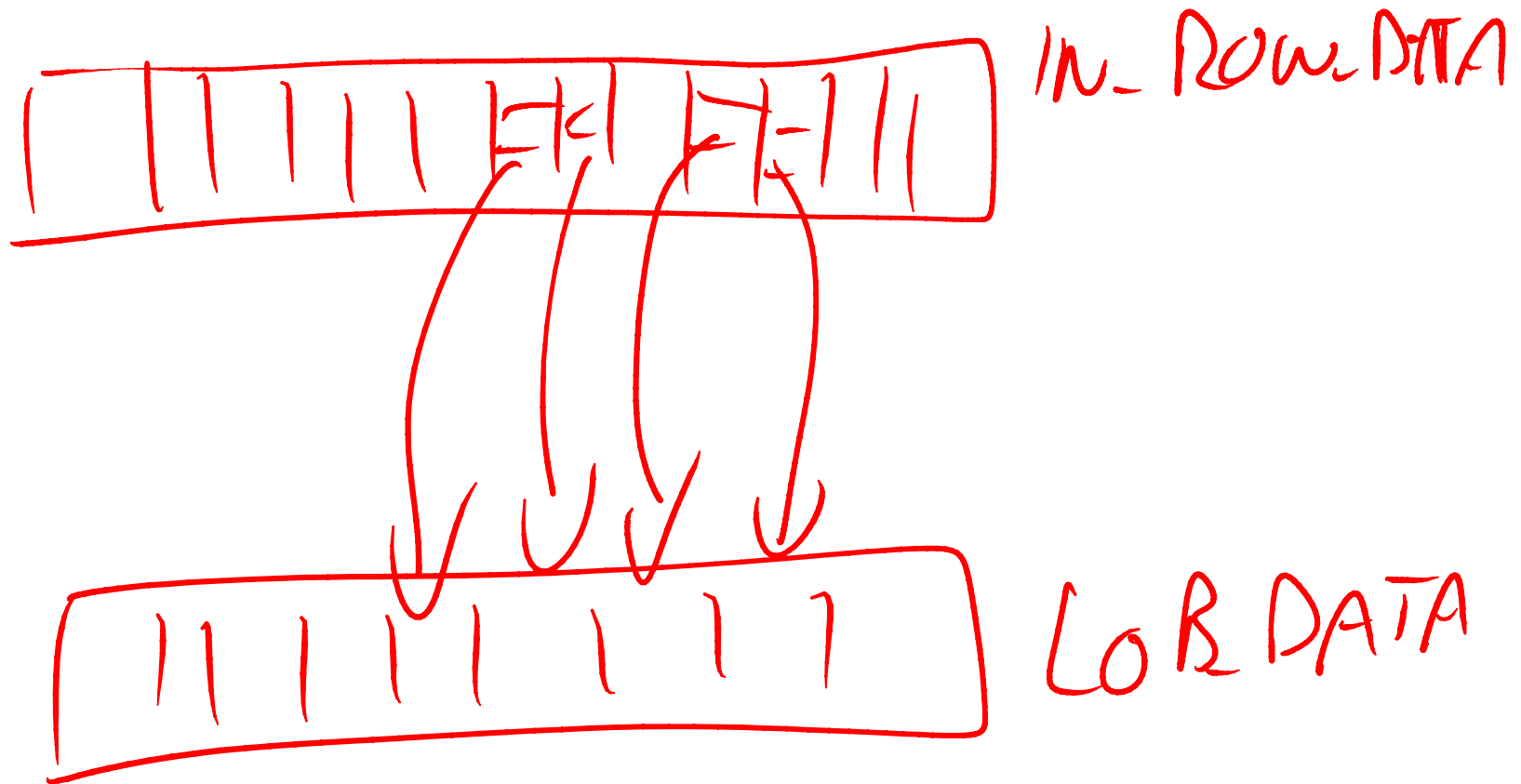
ASYNC I/O
32



M1S18 VLF fragmentation is bad because of having to search through the VLFs for a particular VLF sequence number. See the article link for more info.



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



M1 Answering question about why shrink is so slow with LOB data. The data records point to the LOB pages, but there's no back-link. So every time a LOB page is moved, a table scan must be performed to find which data record to update. Sloooooooooow.

100	900
-----	-----

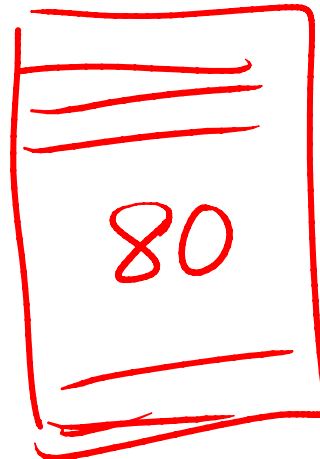
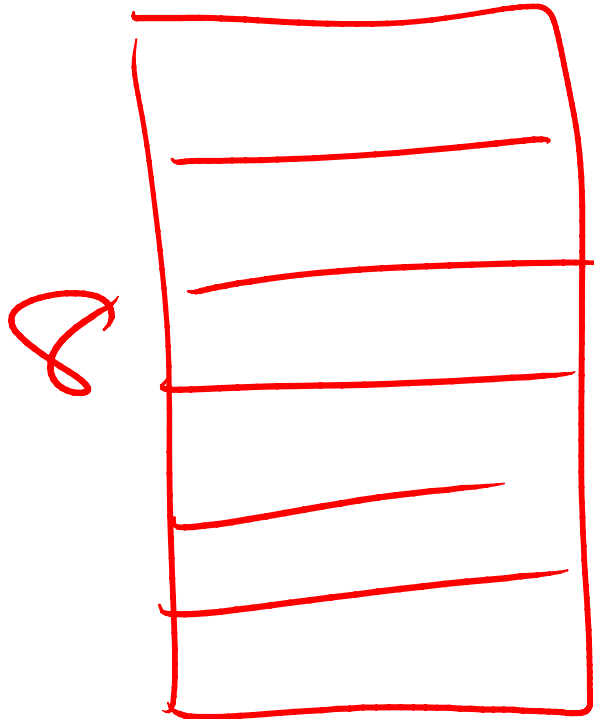
DATA

DENSITY

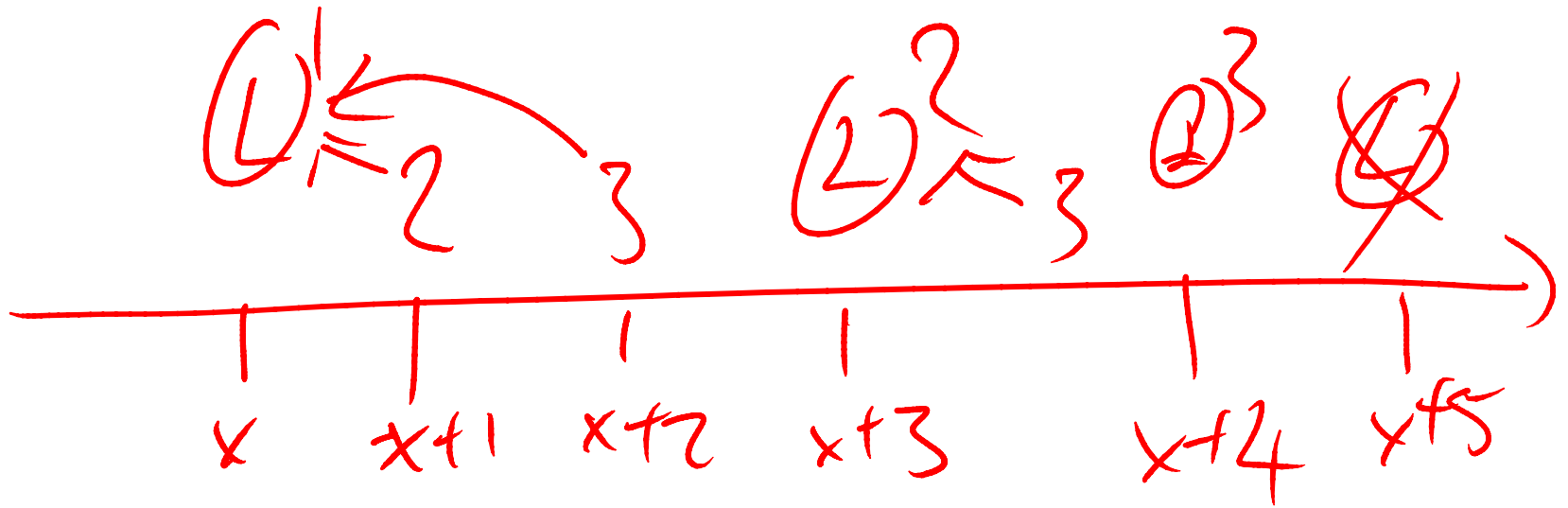
95%

100

900



M1 Explaining how row size affects data density. Putting rarely used data in another table increases the amount of useful data per page, increasing data density and improving performance.



LOCK

LOCK VALUE BLOCK

↳ WHAT

↳ MODE

↳ PENDINE Q

TH. ADDR
MODE
T. ADDR
MODE

Explanation on the next slide

M5 Explaining how the lock pending queue works.

At time:

X: Thread 1 holds the lock

X+1: Thread 2 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.

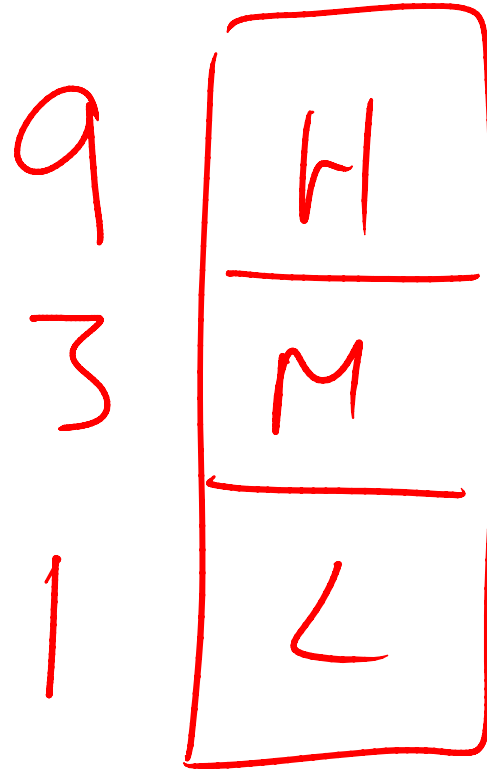
X+2: Thread 3 tries to acquire the lock and can't. It registers a callback routine etc etc, and it behind thread 2 in the pending queue.

X+3: Thread 1 releases the lock, which grants the lock to thread 2. Thread 2's callback routine is called, signaling it and allowing it to move to the runnable queue on it's scheduler. Thread 3 is now at the head of the pending queue for the lock.

X+4: Thread 2 releases the lock, which grants the lock to thread 3. Etc etc. There is now no pending queue for the lock.

X+5: Thread 3 releases the lock, now there is no lock for that resource.

M5S9 Explaining how the Runnable Queue is a true FIFO (First-In, First-Out) queue unless Resource Governor workloads groups and group priorities are being used. High-Medium-Low priority equals 9-3-1 threads through the Runnable Queue.



BT

NESTED SUBXACT

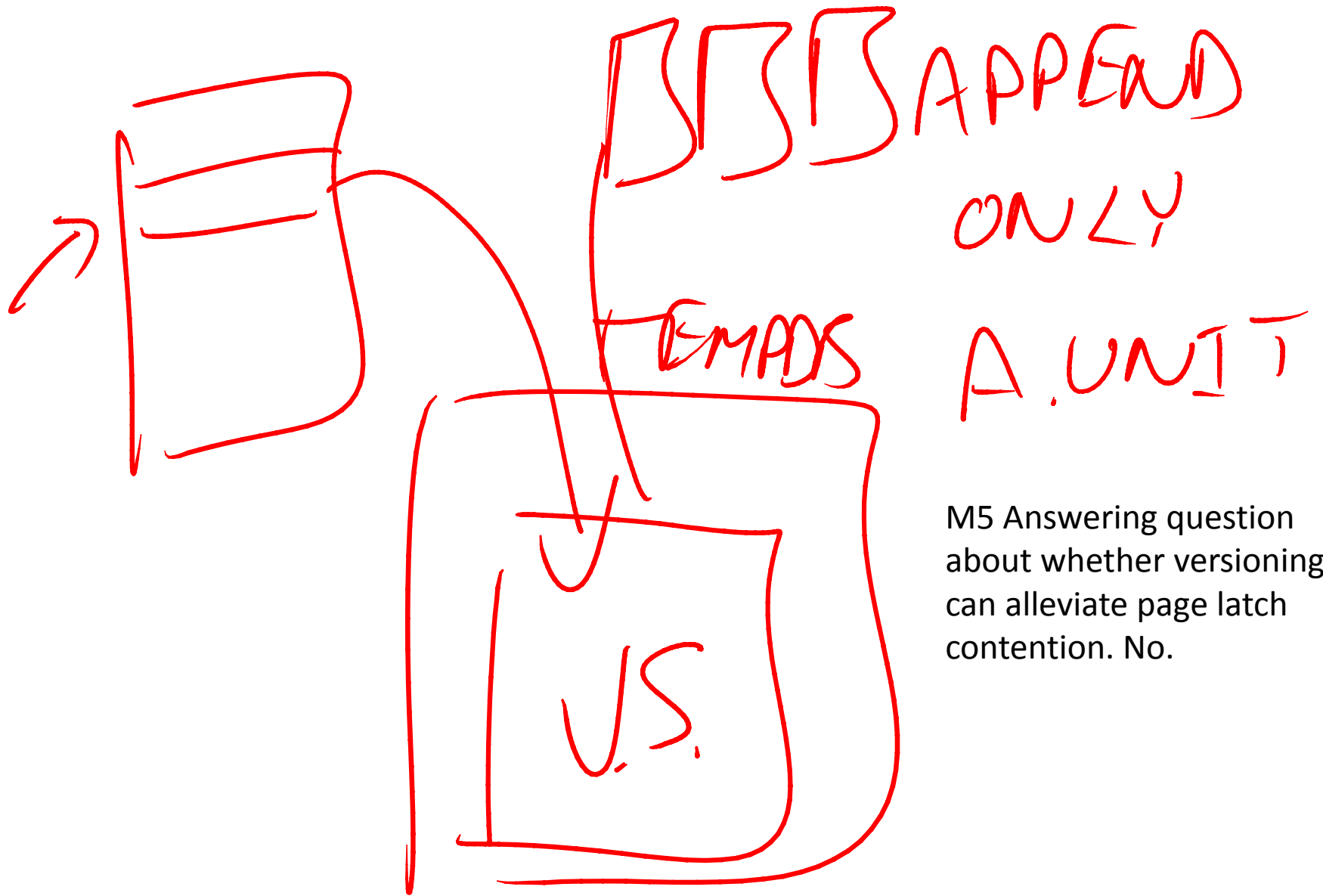
INSERT ✓

BST

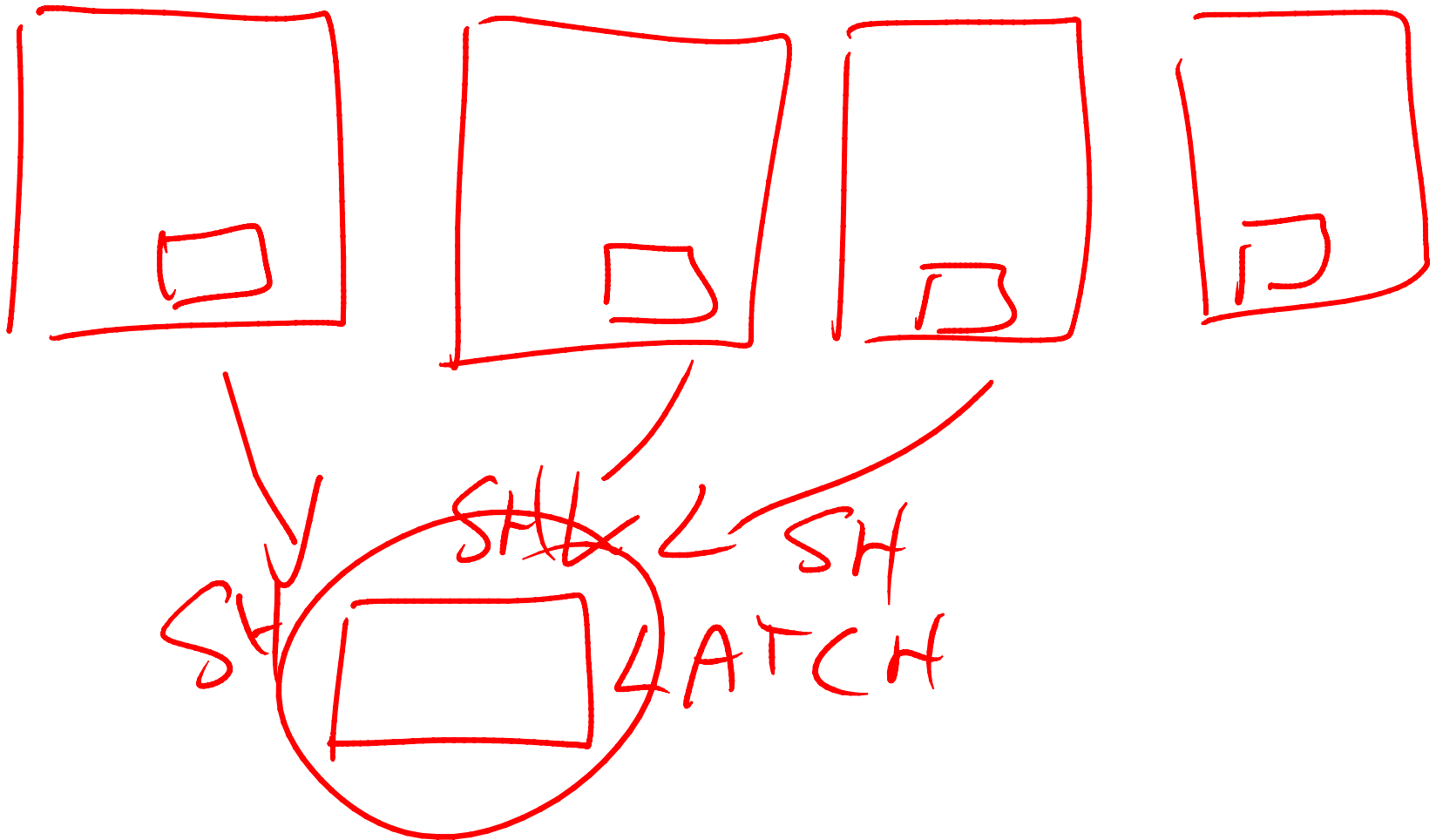
SPLIT

LST

M5S30 Explaining how a page split is performed by a system transaction (a sub transaction of your transaction) that commits and does not roll back if the user transaction rolls back

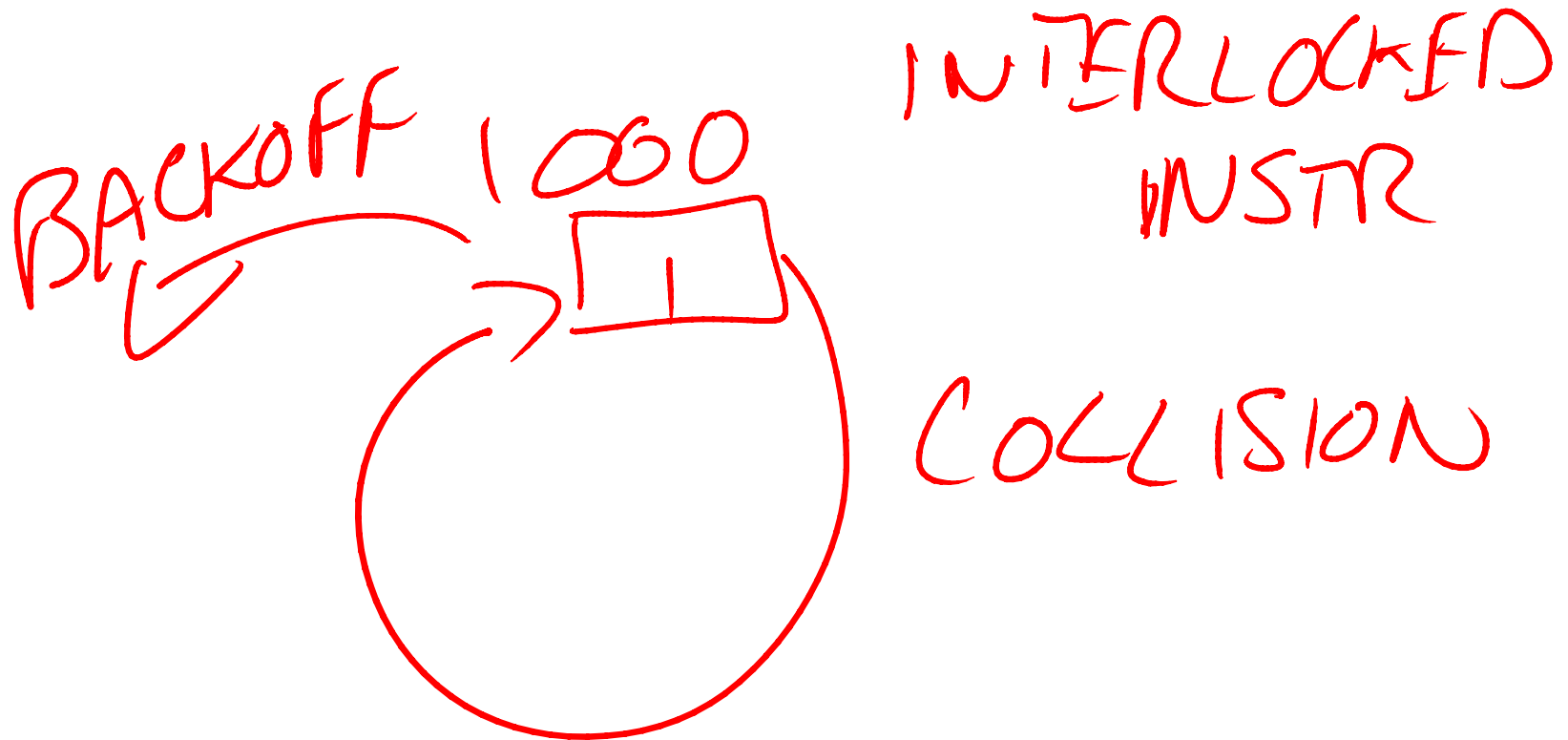


M5 Answering question
about whether versioning
can alleviate page latch
contention. No.



M5S32 Explaining super-latches, where a SH latch is made per NUMA node to speed things up.

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.

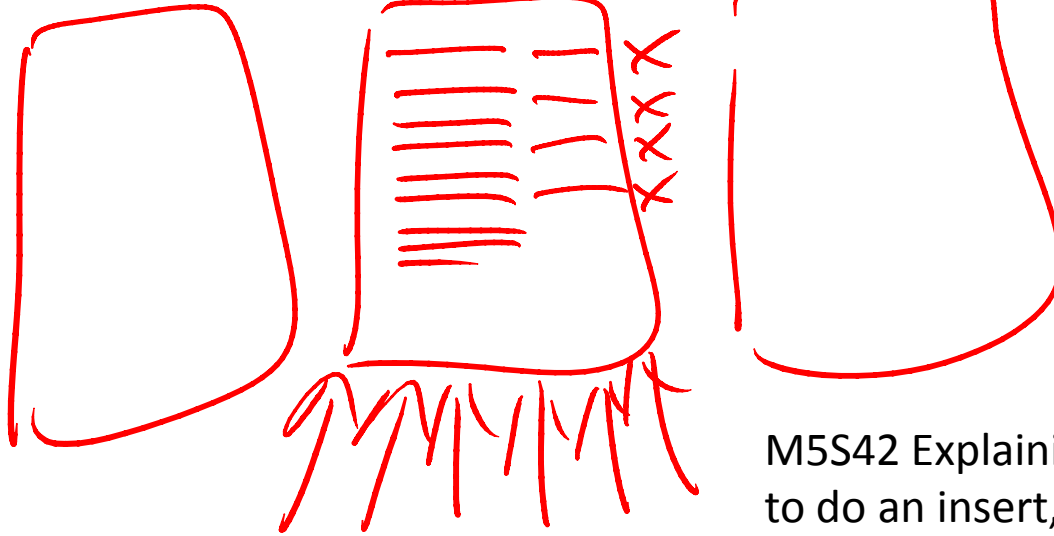
TIX

TIX = table IX lock

PIX = page IX lock

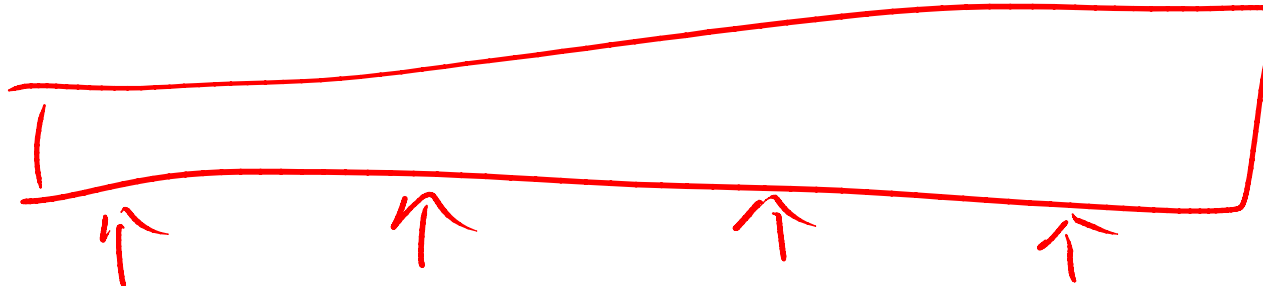
And there are row X locks

SOPIX



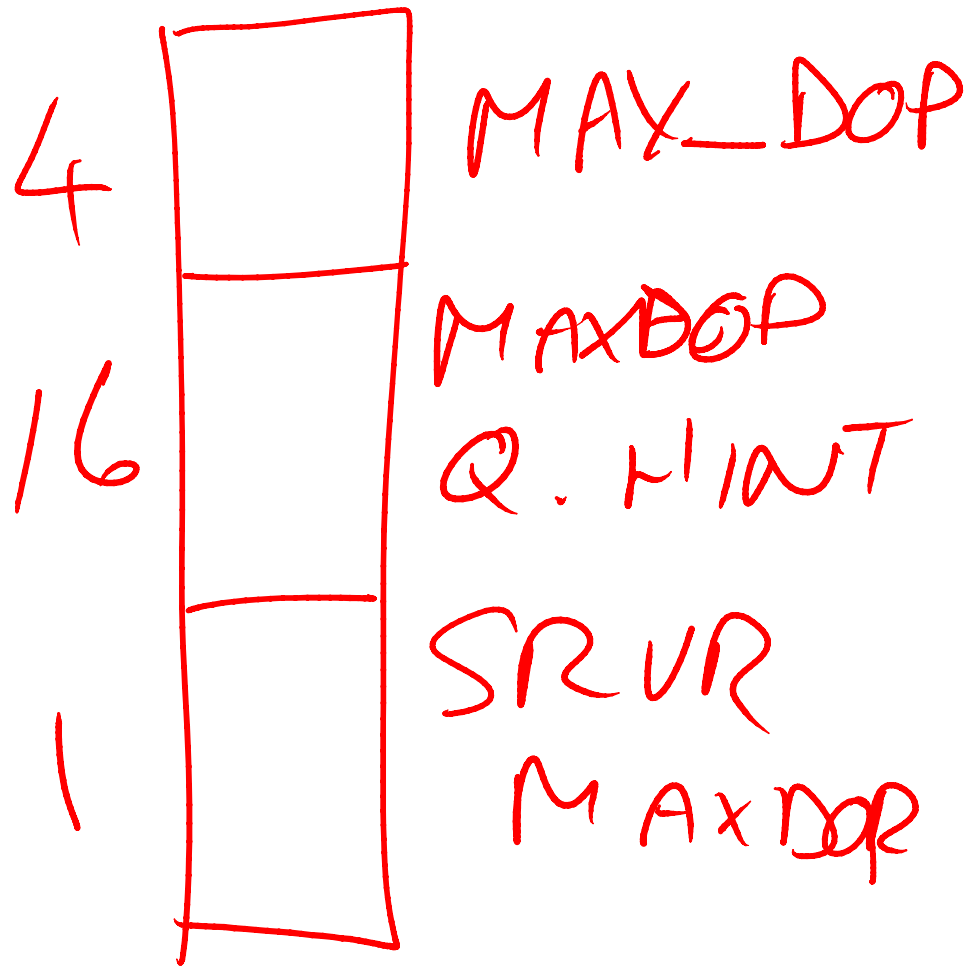
X X X
LATCH

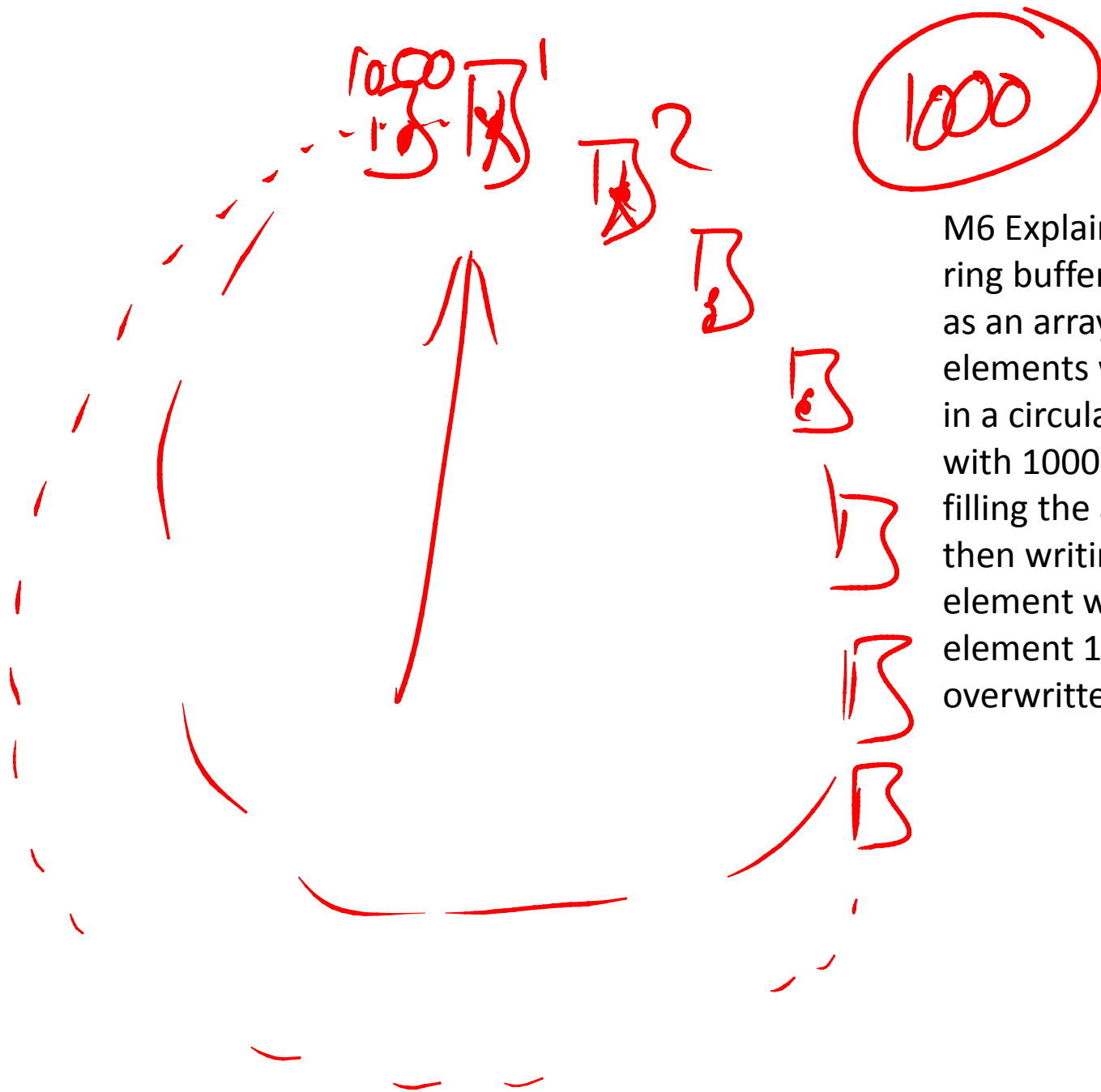
M5S42 Explaining that to access a page to do an insert, multiple concurrent threads can hold non-conflicting row X locks, but they all need to acquire the X latch on the page, which becomes a bottleneck.



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

M5S54 Explaining the hierarchy of DOP control. sp_configure can be overridden with a query hint by anyone, but Resource Governor workload group MAX_DOP cannot be overridden.





M6 Explaining what a ring buffer is. Think of it as an array of data elements which is filled in a circular fashion. E.g. with 1000 elements, filling the array and then writing the 1001st element will cause element 1 to be overwritten.

M6S21 Explaining what the bucketizer/histogram target does – creates a histogram.

