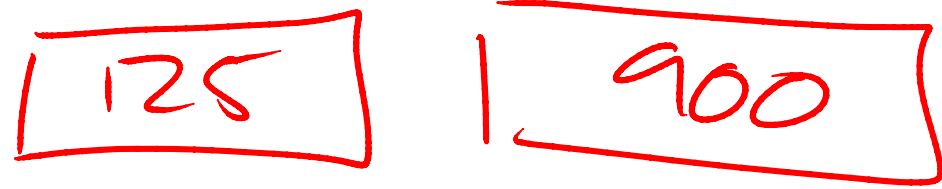
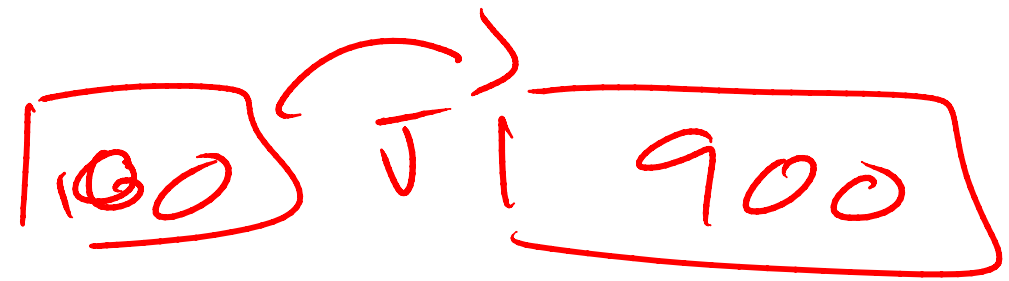


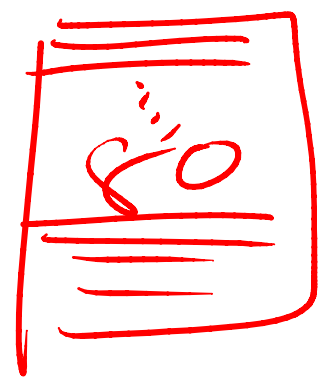
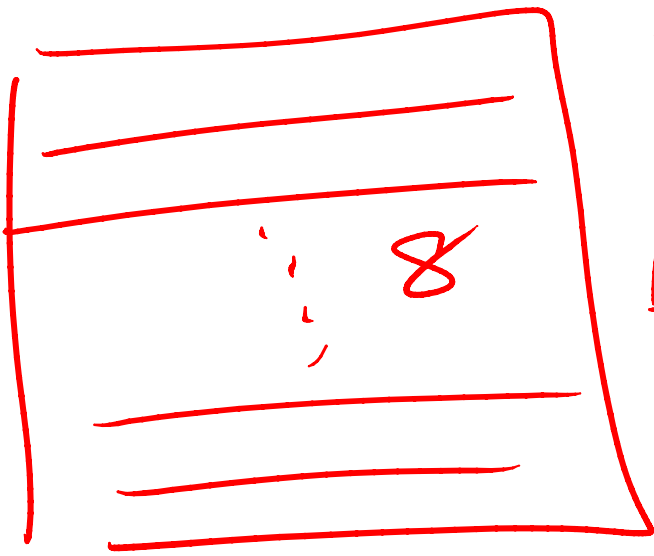
2%

100	900 LOB
-----	---------

OLTP



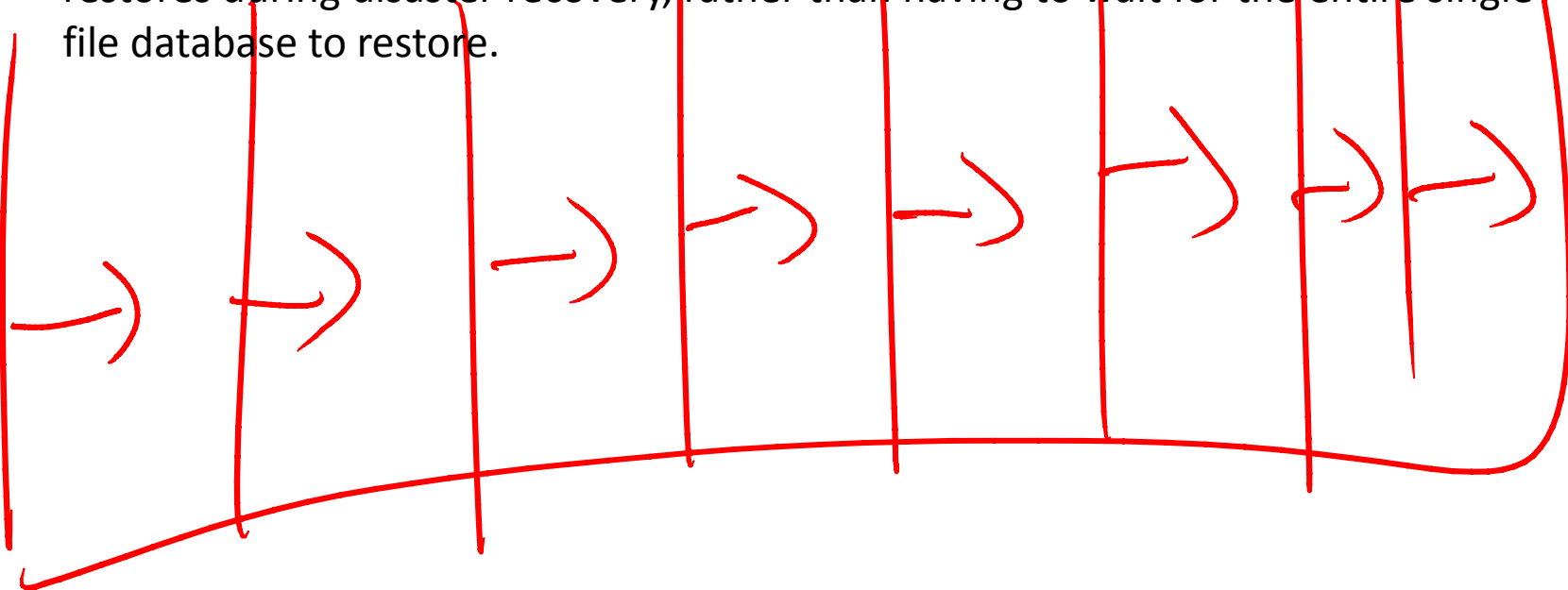
off row

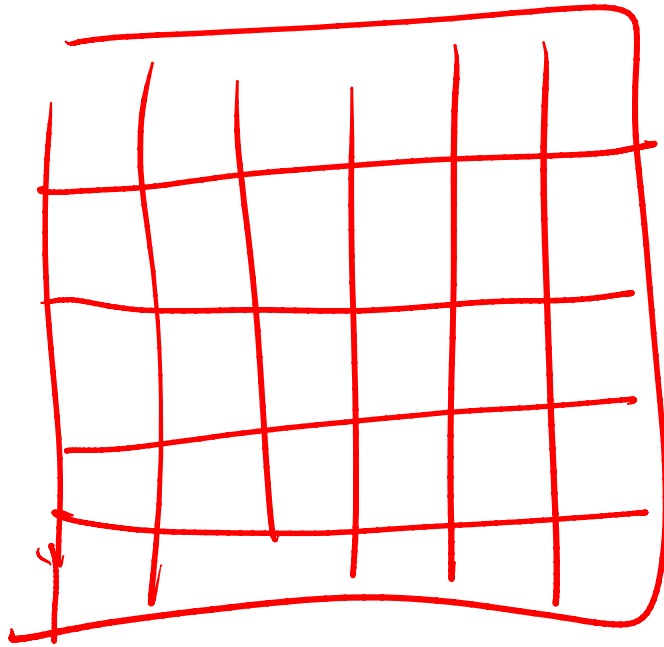


M1S5 Having a large LOB value in row that's not used much makes for large rows and low data density. Choose how to store LOB data wisely.

L

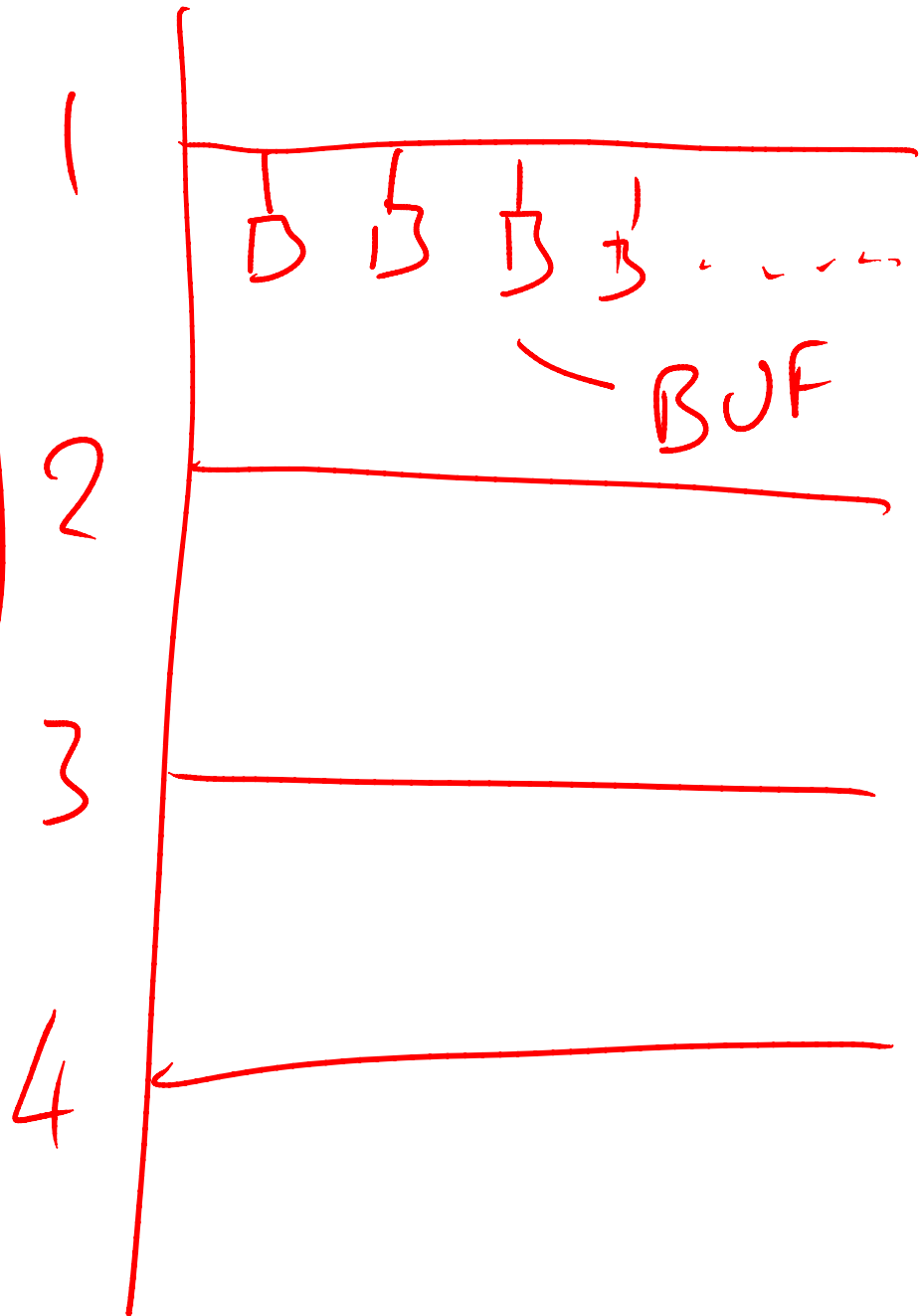
M1S7 Explaining that dividing a large database into filegroups allows targeted restores during disaster recovery, rather than having to wait for the entire single file database to restore.





W/ASH
LIST

M1S34: The buffer pool contents are tracked by BUF structures, arranged per database in a list, for easy access to metadata about a page in memory.



BUFFER MER

TARGET PAGES

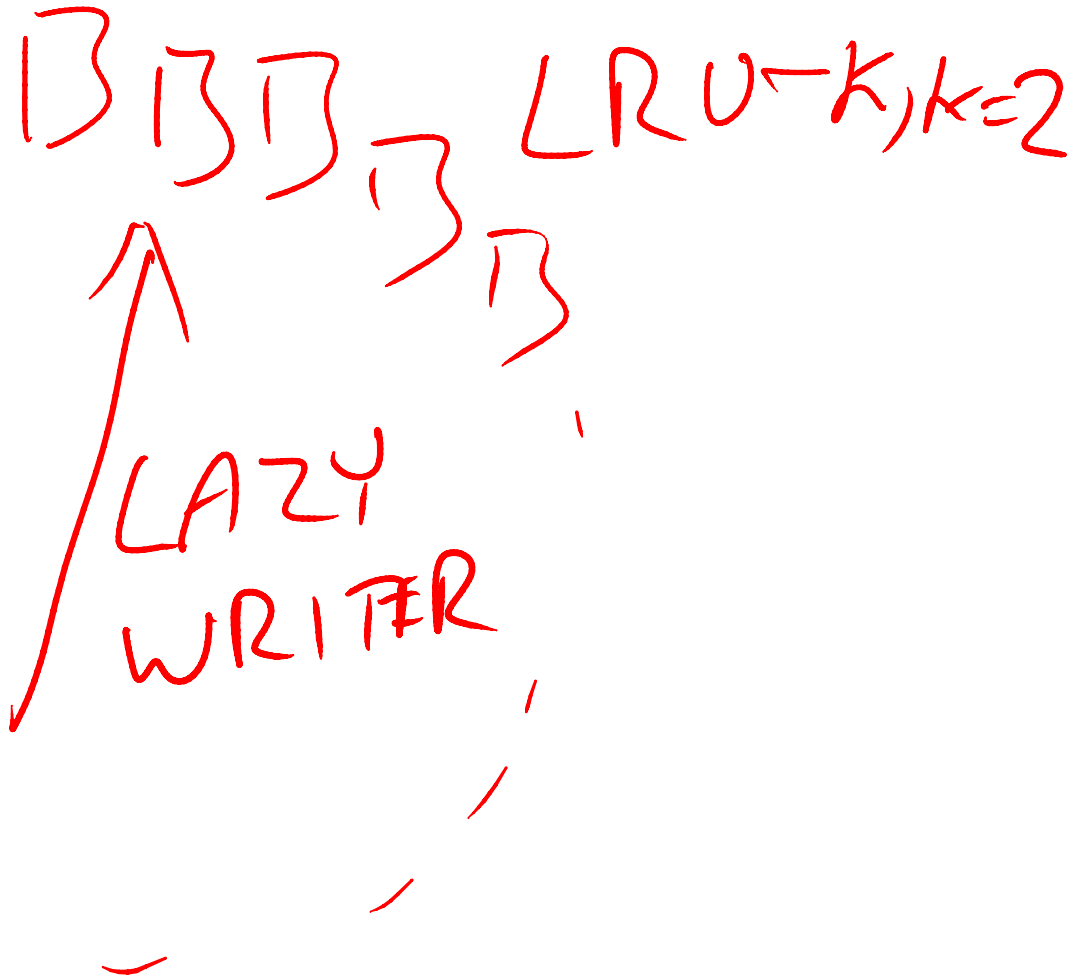
TOTAL PAGES

M1S34: Buffer pool memory is tracked using these perfmon counters (or Buffer Node counters if on a NUMA system). The Total Pages will ramp up after a server start. You can make it faster on non-Enterprise using documented TF 840.

t

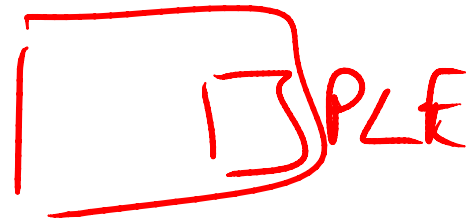
PAGE LIFE EXPECTANCY (PLE)

M1S34: Buffer pool implements a least-recently-used algorithm, to determine which pages are dropped when there's memory pressure. When memory pressure, lazy writer sweeps through buffers to find LRU ones. PLE can be thought of as a measure of pressure on the buffer pool free list. Also how long a page will remain in memory in seconds.

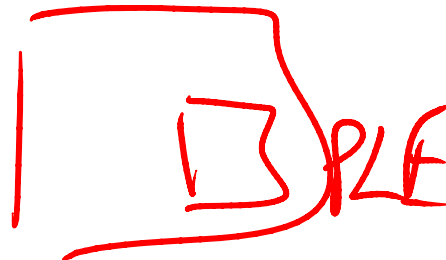
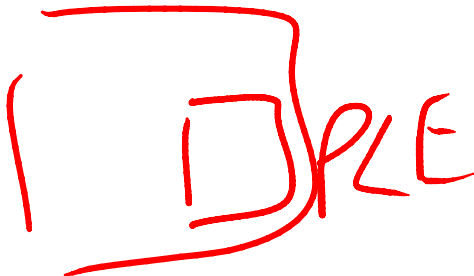


BUFFER MGR : PLE

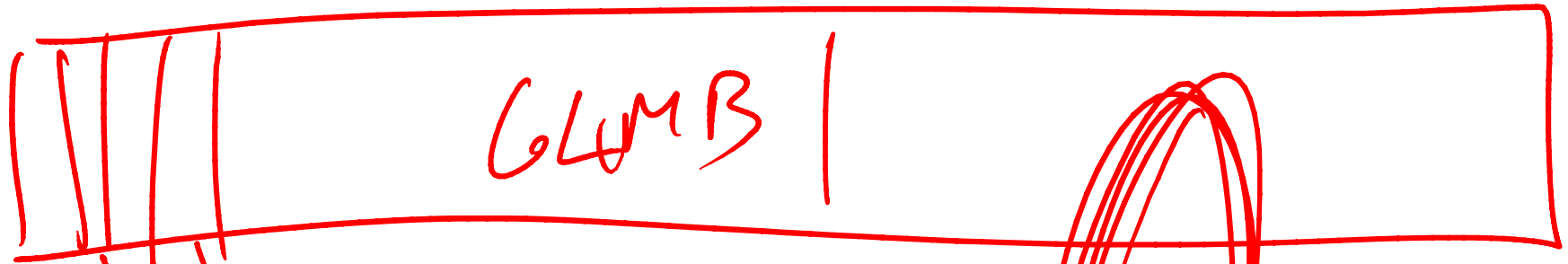
NUMA



BUFFER NODE: PLE



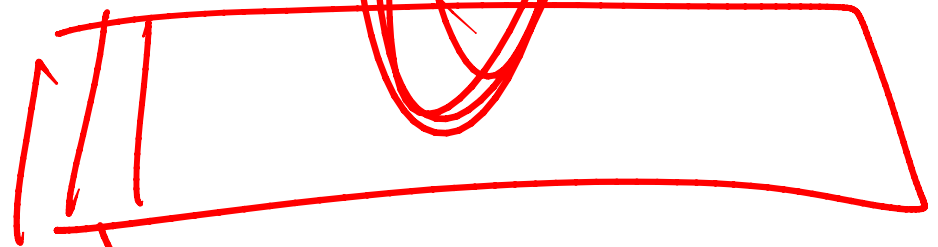
M1S34: On NUMA, you need to watch each Buffer Node PLE, not the Buffer Manager PLE. See <http://www.sqlskills.com/blogs/paul/page-life-expectancy-isnt-what-you-think/>



PFS SGAM

EX

1 101
0



PFS

M1S41 Explaining PFS and SGAM contention in tempdb and how adding multiple files spreads the contention over multiple files, thus reducing it.

1118

PAGE

RES.	GL	PQ
------	----	----

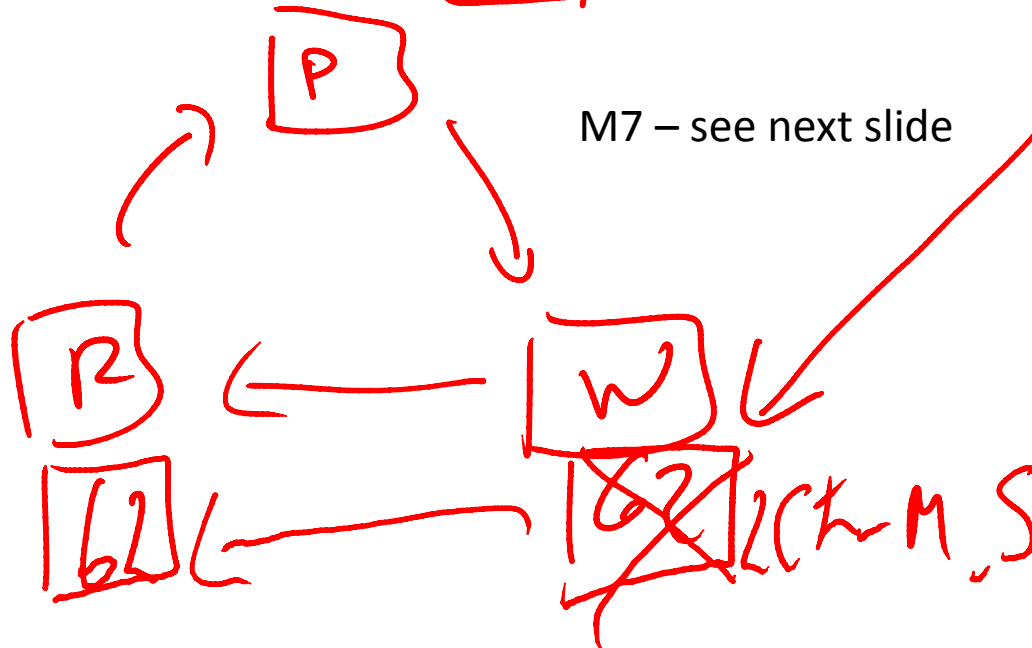
GRANTED
LIST

PENDING QUEUE

X	61
S	62

~~S | 62 | ADDR.~~

M7 – see next slide



M7 Explaining how the lock pending queue works.

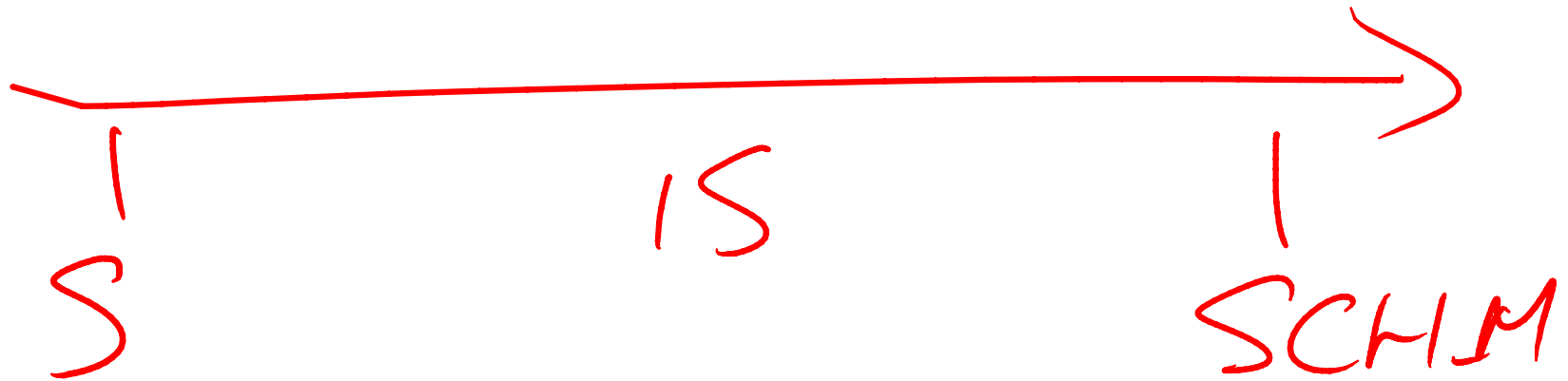
At time:

X: Thread 61 holds the lock in X mode

X+1: Thread 62 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 61 releases the lock. It checks the pending queue for locks that can now be granted ownership of the lock (taking into account any other threads still holding the lock). In this case, thread 62 is granted the lock in S mode. Thread 62's callback routine is called, signaling it and moving it to the runnable queue on it's scheduler.

WAIT_AT_LOW_PRIORITY



TAD IX

M7S48 Online index operations in 2014 have a new feature `WAIT_AT_LOW_PRIORITY`, that allow the two blocking locks (S and SCH_M) to not cause blocking, and after a set amount of time, either kill the operation or kill the blockers.

See next slide...

8

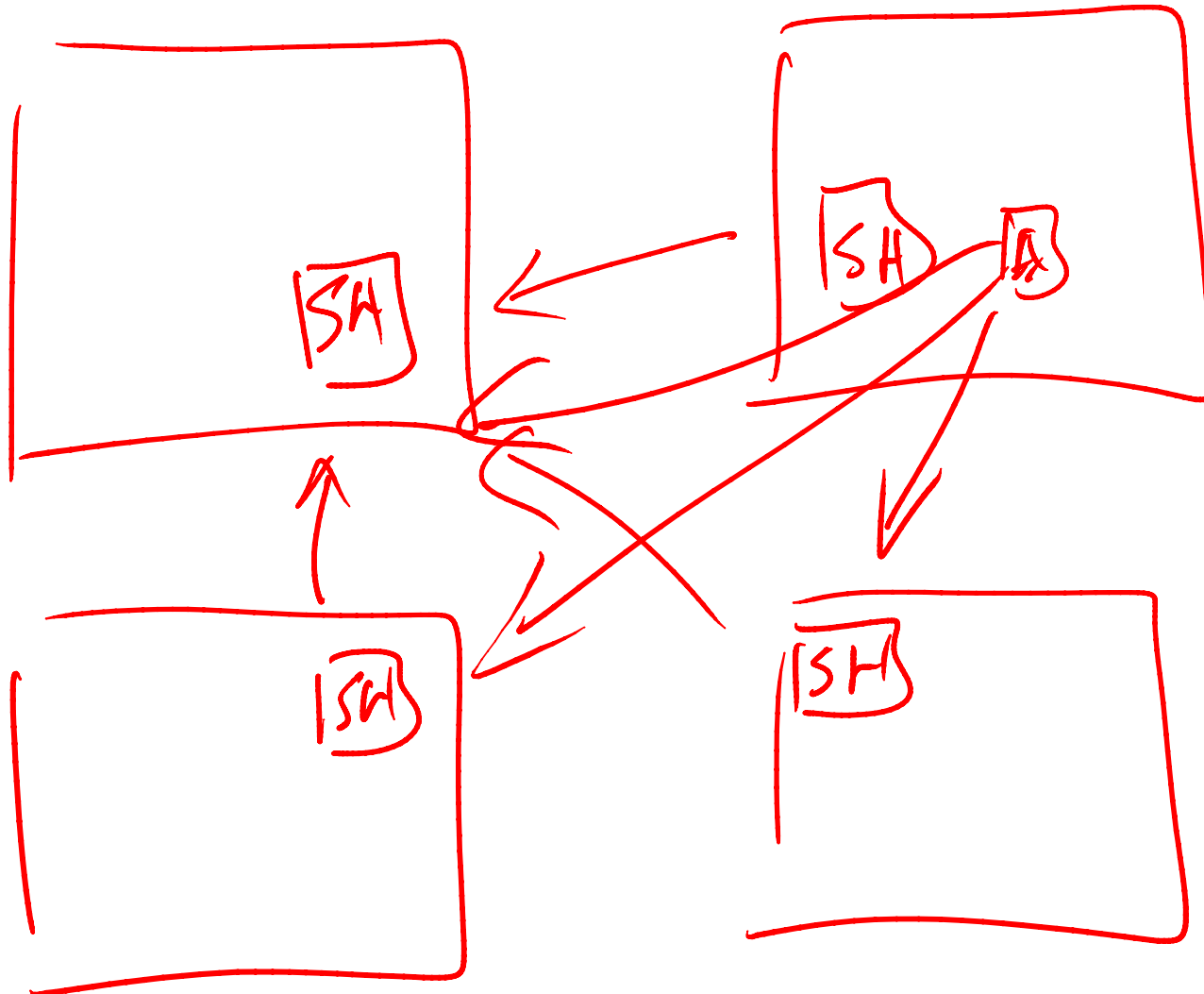
IX

X	←
X	←
X	←
X	←
X	←
X	←

IDENTITY
TABLE IX

EX LATCH

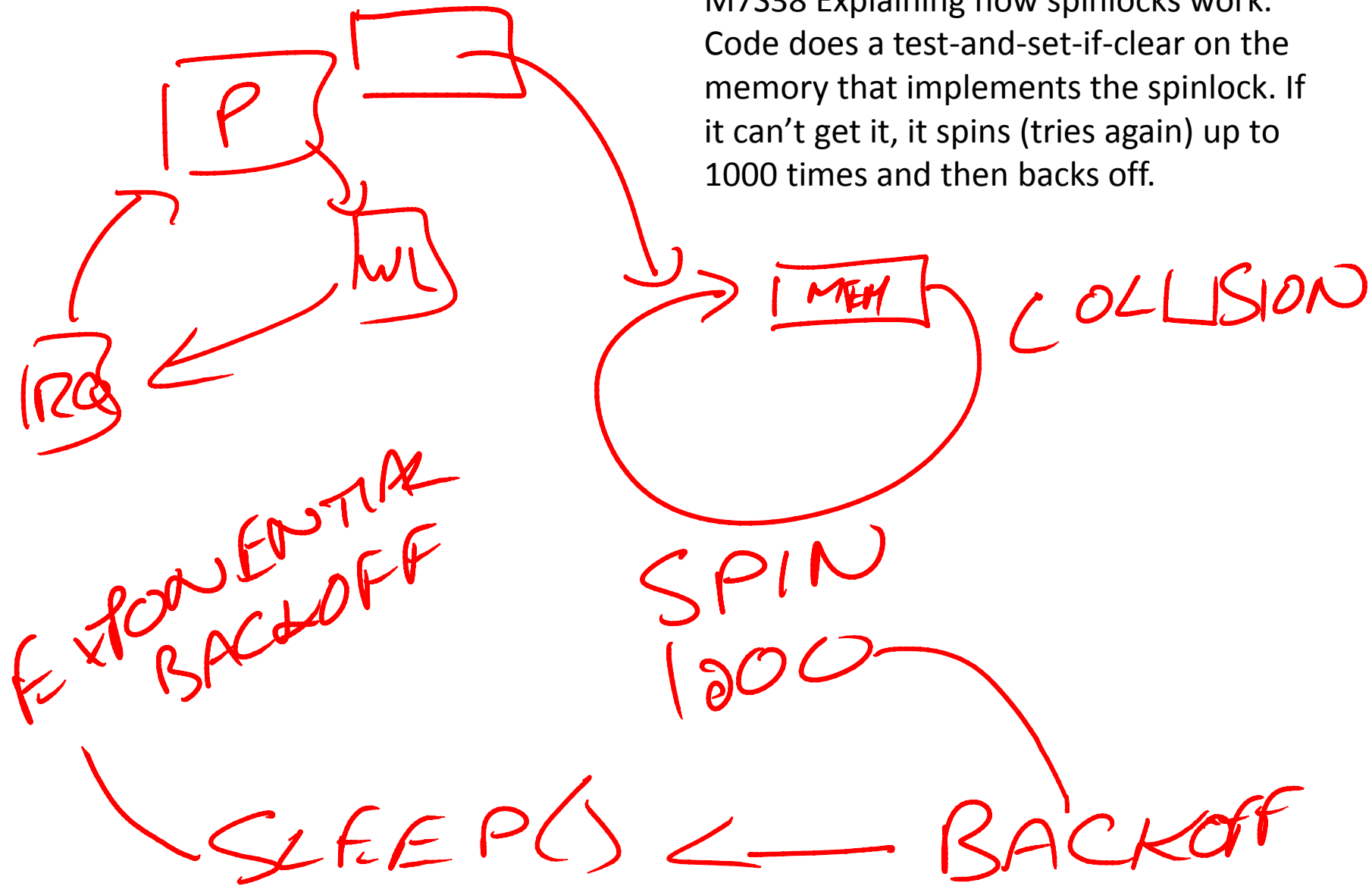
M7S45 Explaining that to access a page to do an insert, multiple concurrent threads can hold non-conflicting row X locks, but they all need to acquire the EX latch on the page, which becomes a bottleneck. If the clustered index has an int/bigint identity, and the row size is small enough, it could result in an insert hotspot.



NUMA

M7S35 Explaining about
superlatches.

M7S38 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.



10

12
048

13
159

13

14
24

15

16
37

17

M7S54 For DOP=4, there could be up to $DOP \times 2 + 1$ threads per parallel zone. The operator can only use DOP schedulers, so the threads for the parallel operator are distributed amongst the DOP schedulers that SQLOS chooses to use.