

2 TB SALES

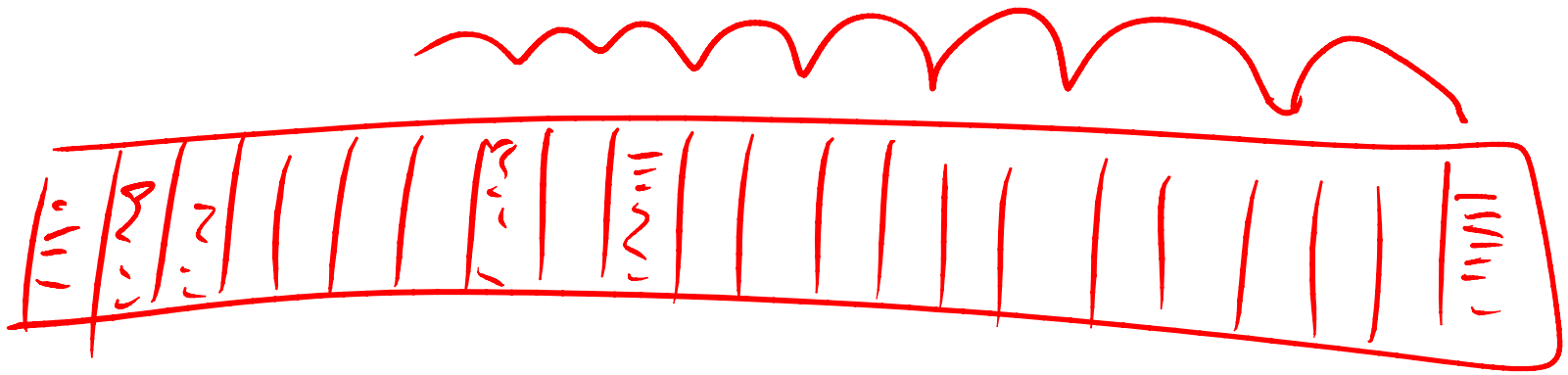
2011 - - - - - 2014

DW

OLTP

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. (older, neater copy of this slide using 2014 values)

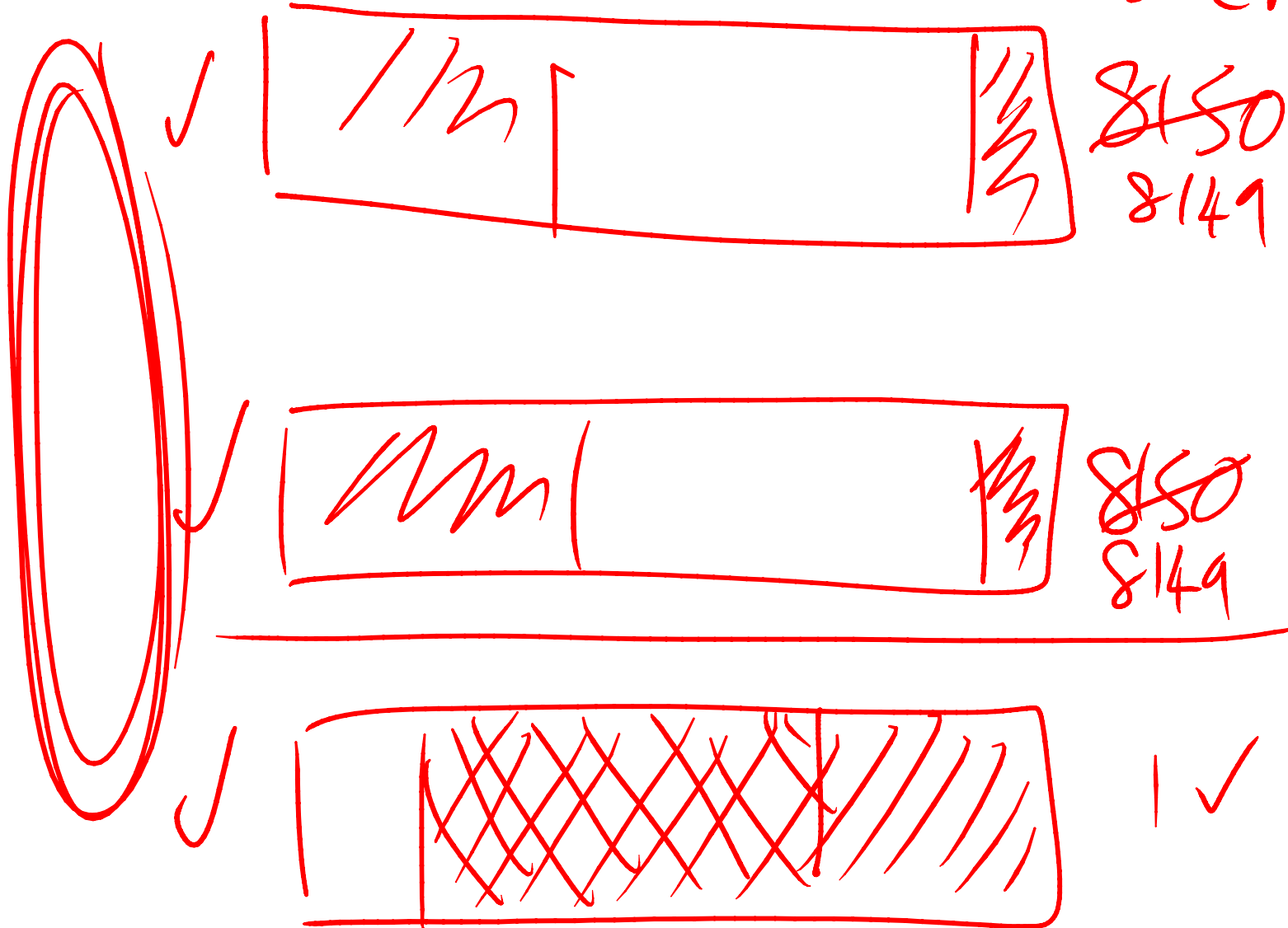
P 2014 2013 2012 2011



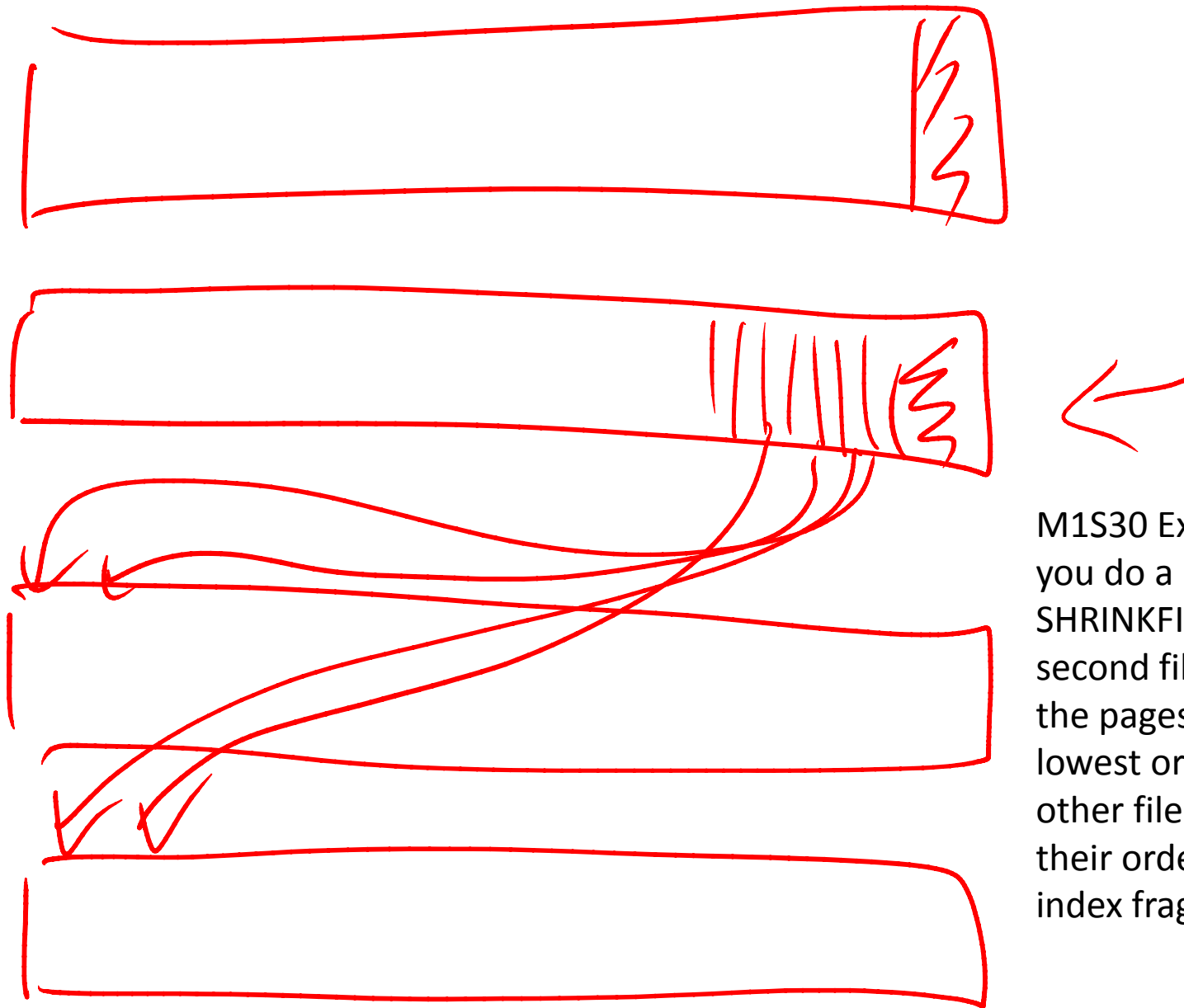
VLFs

M1S23 VLF fragmentation is bad because of having to search through the VLFs for a particular VLF sequence number.

WEIGHTING



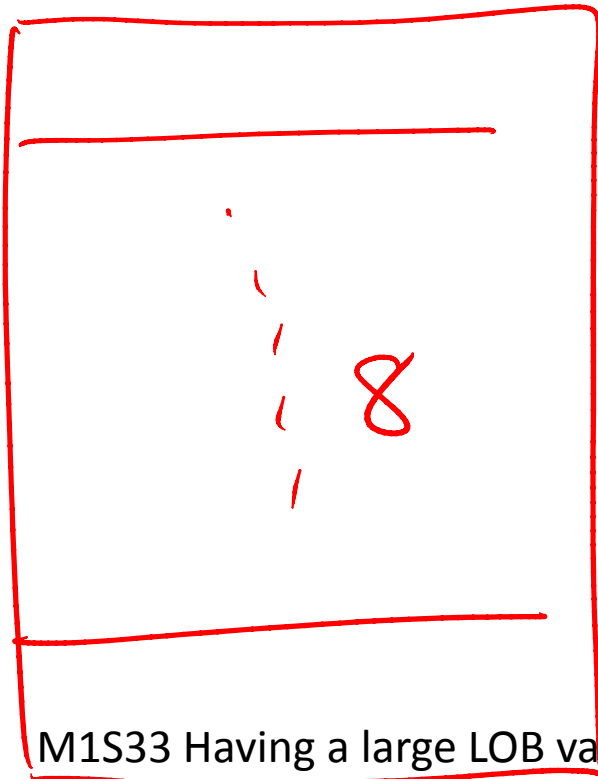
M1S30 Explaining proportional fill and round robin. Adding another file to a full filegroup does not allow for rebalancing of data.



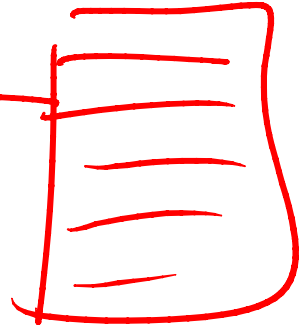
M1S30 Explaining that if you do a DBCC SHRINKFILE on the second file, it will move the pages in highest to lowest order into the other files, reversing their order and causing index fragmentation.



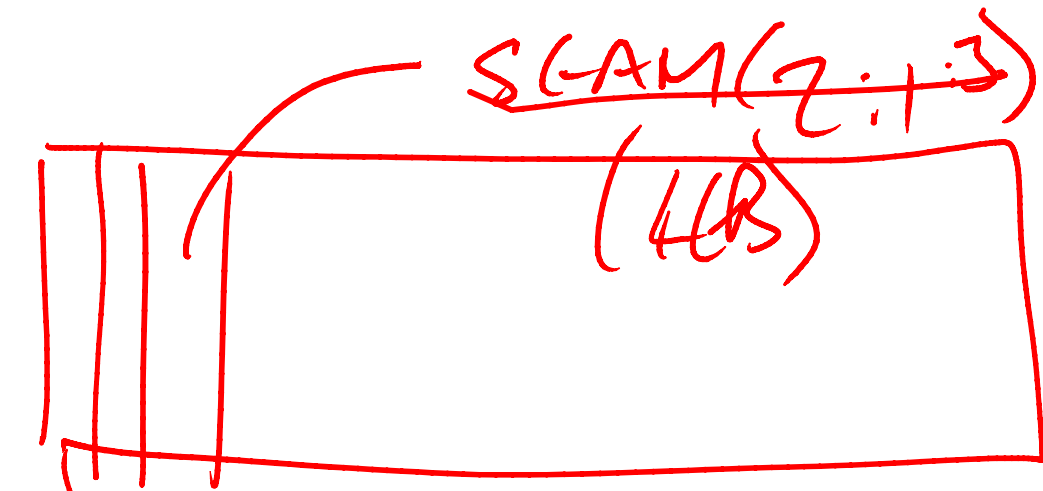
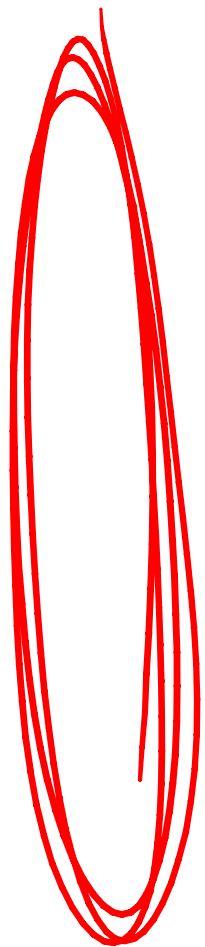
LARGE VALUES OFF ROW



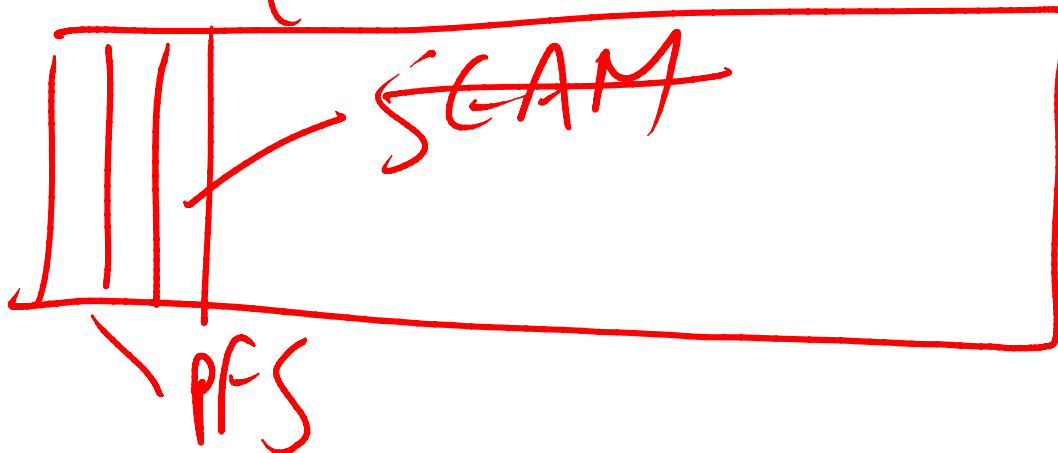
OK



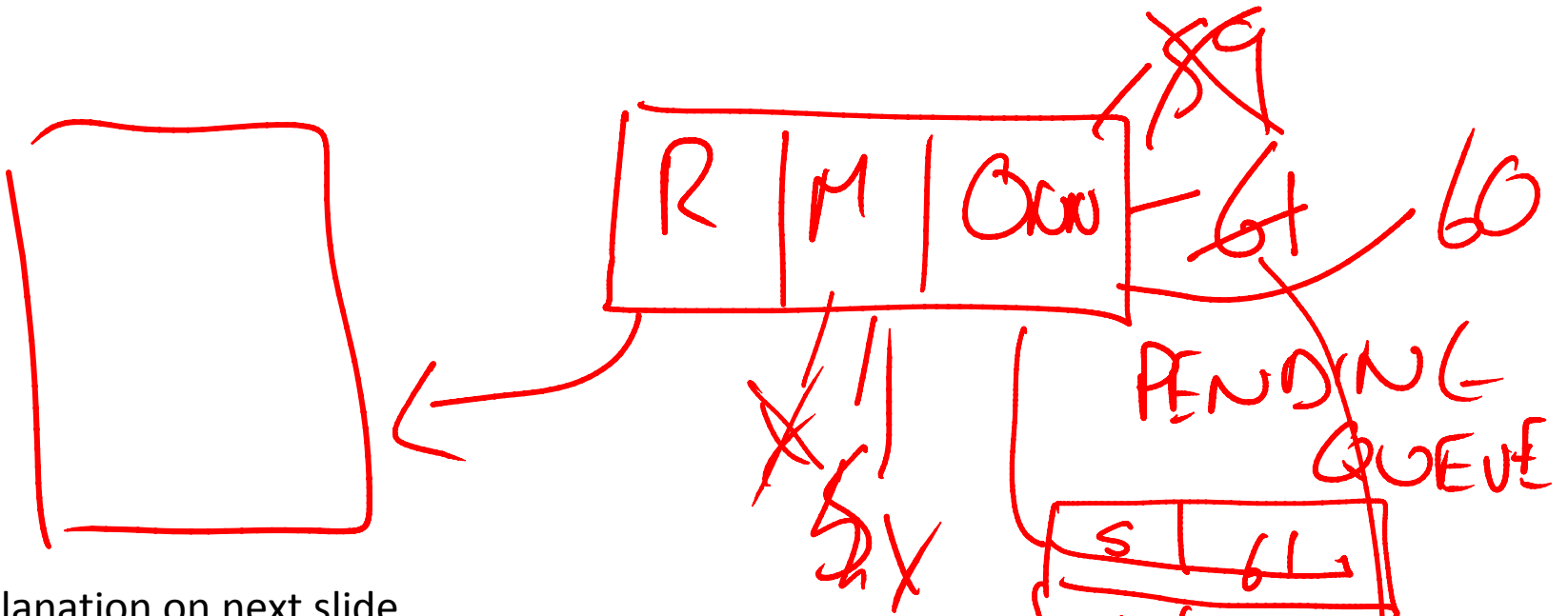
M1S33 Having a large LOB value in row that's not used much makes for large rows and low data density. Choose how to store LOB data wisely.



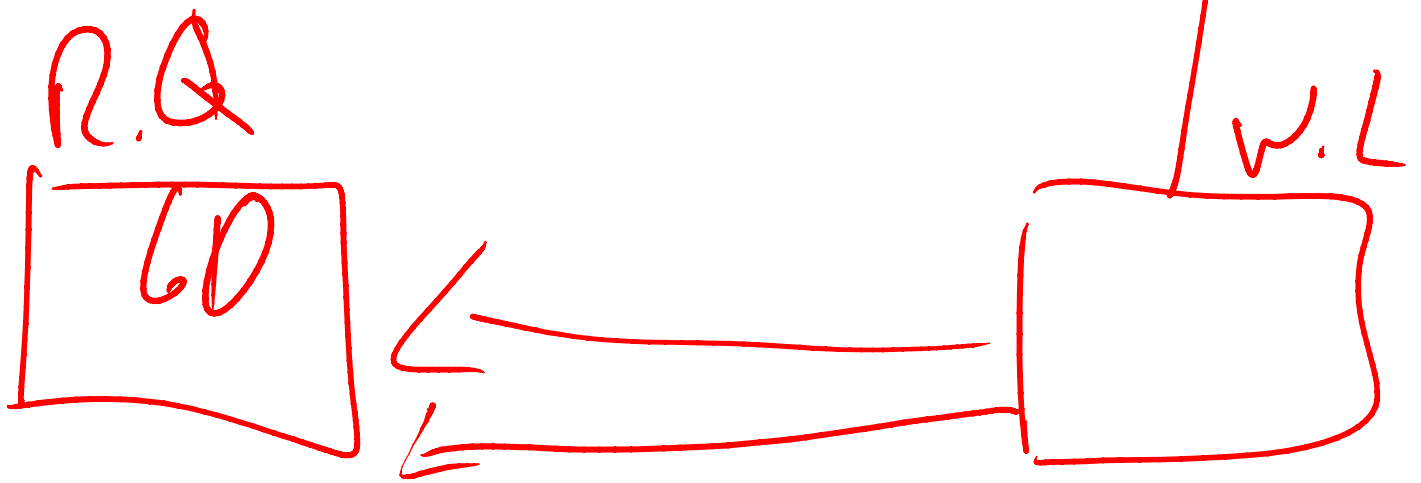
PFS(2:1:1)
(64MB)



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



Explanation on next slide



M7 Explaining how the lock pending queue works.

At time:

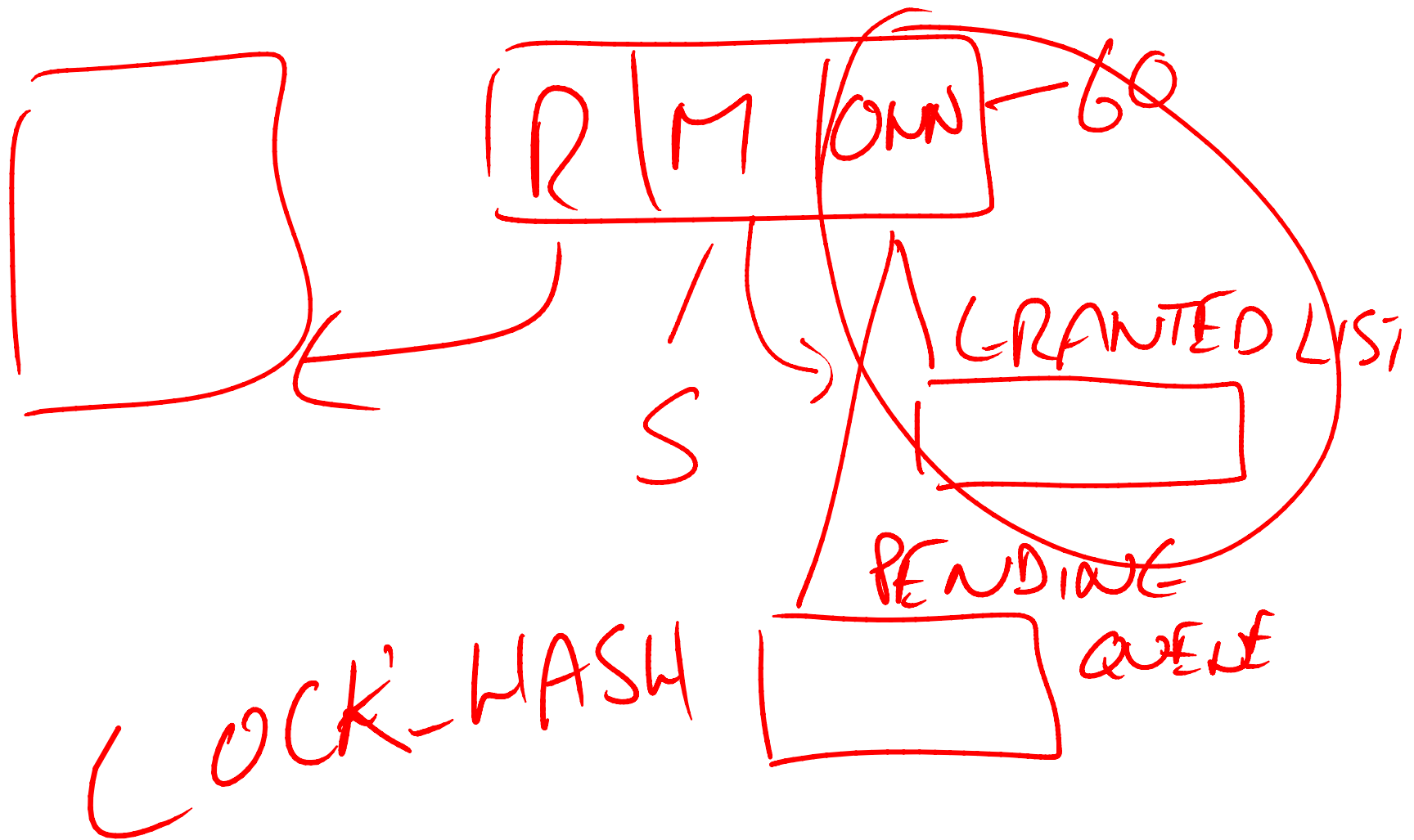
X: Thread 59 holds the lock

X+1: Thread 61 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 60 tries to acquire the lock and can't. It registers a callback routine etc etc, and it behind thread 61 in the pending queue.

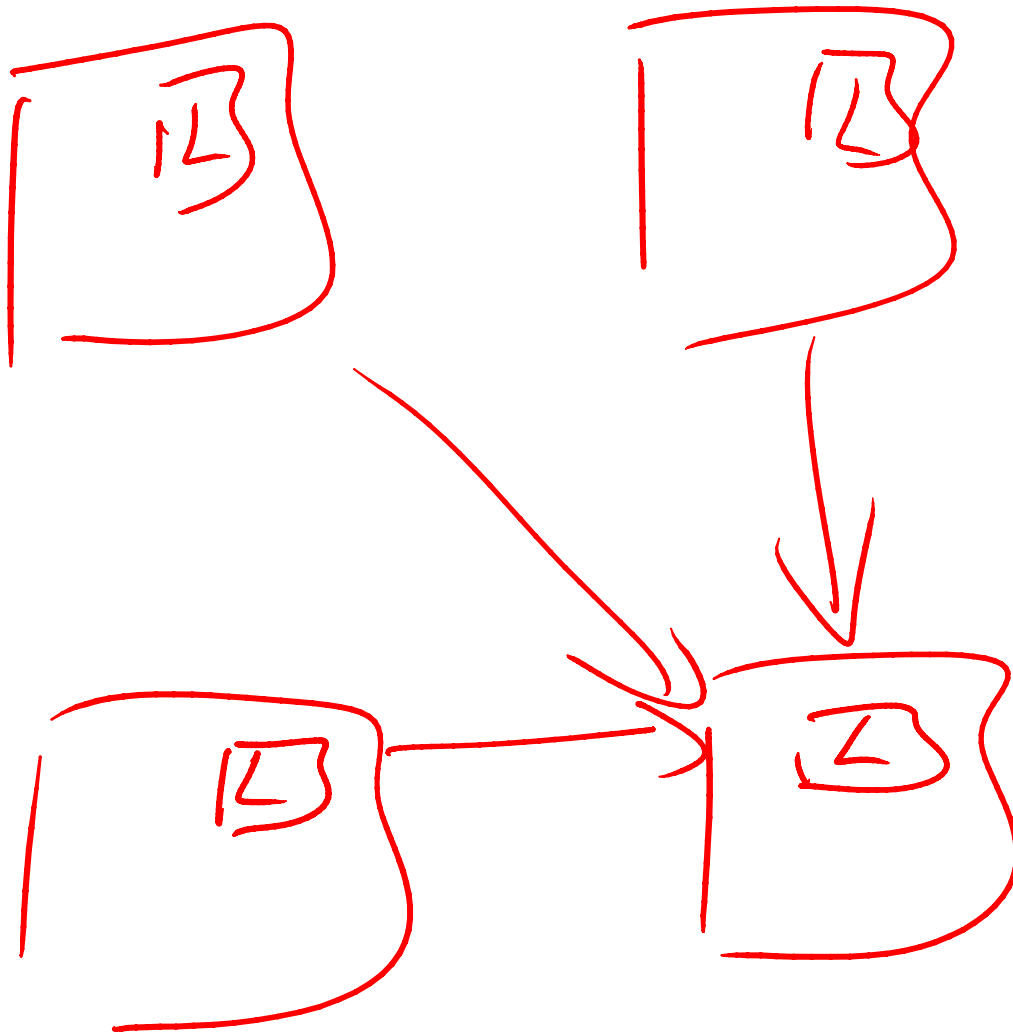
X+3: Thread 59 releases the lock, which grants the lock to thread 61. Thread 61's callback routine is called, signaling it and allowing it to move to the runnable queue on it's scheduler. Thread 60 is now at the head of the pending queue for the lock.

X+4: Thread 61 releases the lock, which grants the lock to thread 60. Etc etc. There is now no pending queue for the lock.

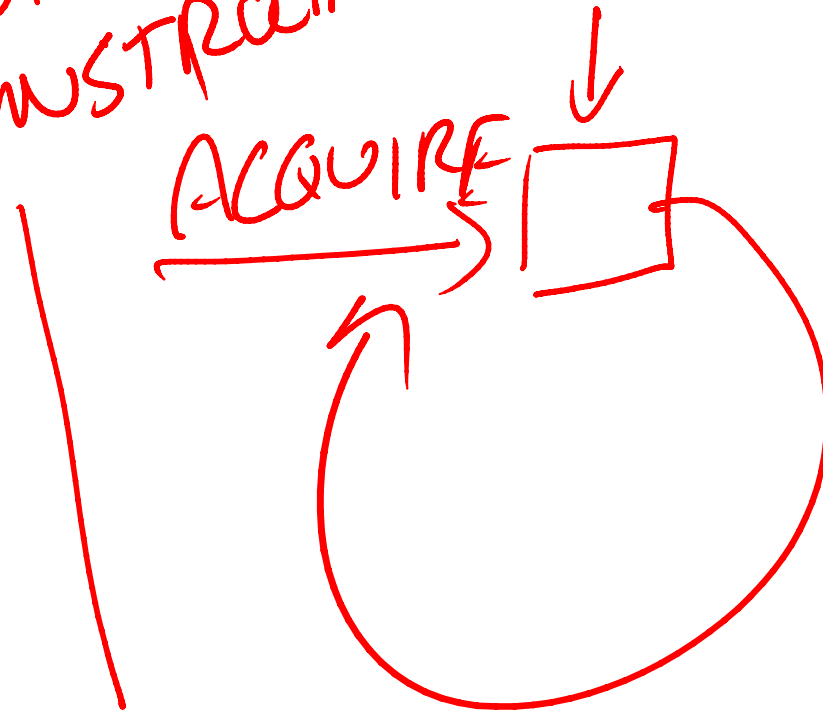


M7 We enhanced our picture of operations to show that the lock owners are all in the granted list for the lock. For instance, there may be 4 IS owners and one S owner, plus 2 IX thread in the pending queue. If the S owner drops the lock, both IX threads are moved to the granted list (because IX is compatible with IS).

M7S35 Explaining about
superlatches.



INTERLOCKED
INSTRUCTION SPINLOCK



SPINNING

1000

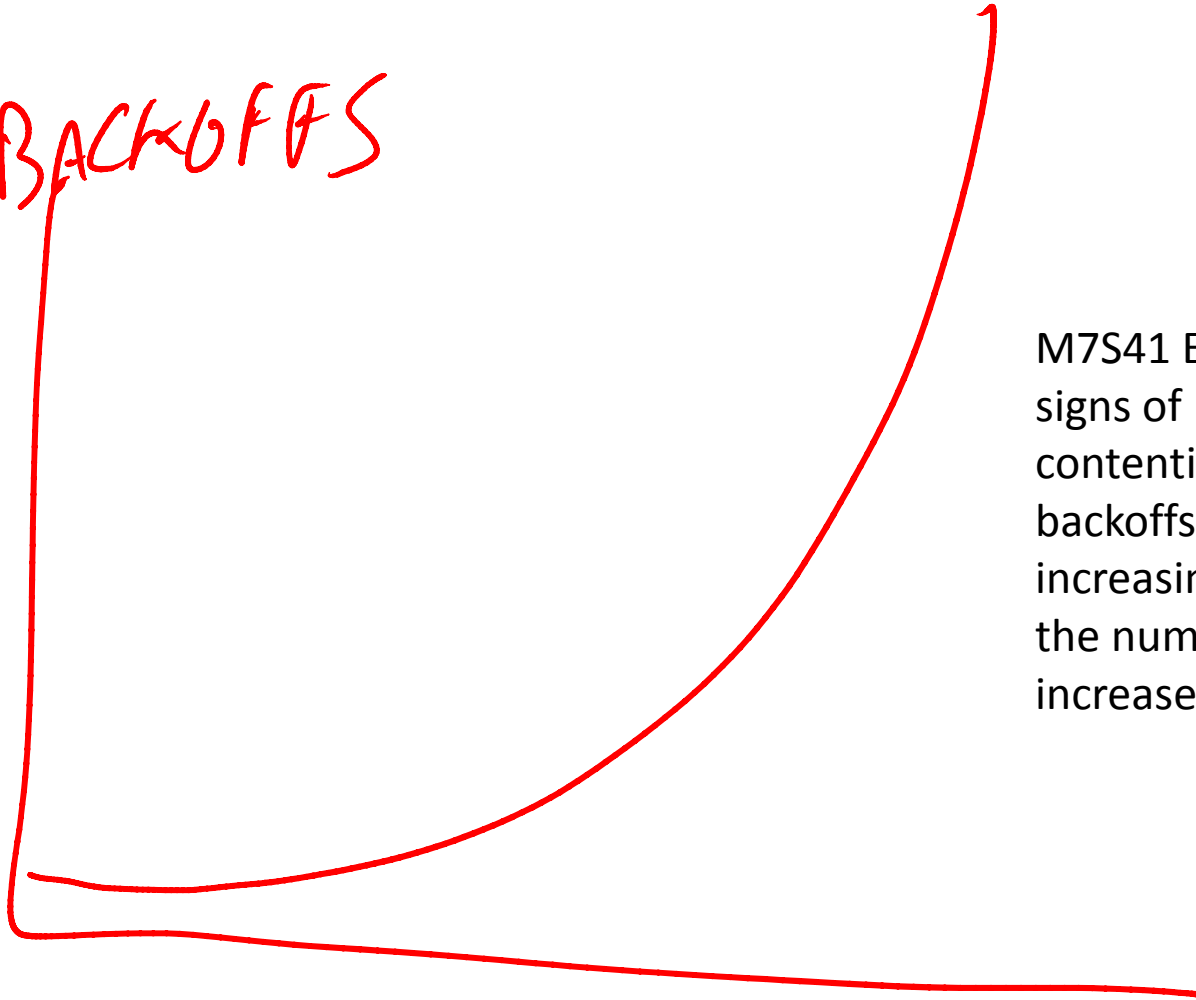
COLLISION

TST_SET_IF_CLEAR

BACKOFF

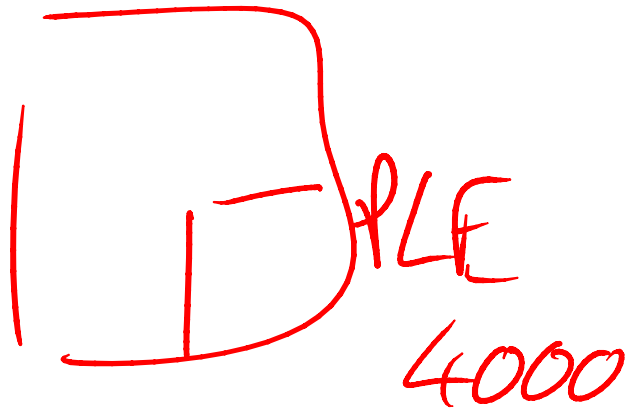
M7S40 Explaining how spinlocks work. Code does a test-and-set-if-clear on the memory that implements the spinlock. If it can't get it, it spins (tries again) up to 1000 times and then backs off. SOS_SCHEDULER_YIELD is the wait type when a backoff occurs.

BACKOFFS

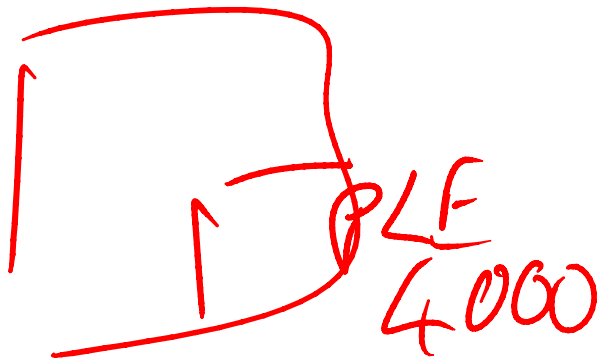


M7S41 Explaining one of the signs of potential spinlock contention – the number of backoffs and CPU usage increasing exponentially as the number of connections increases

CONNECTIONS



See <http://www.sqlskills.com/blogs/paul/page-life-expectancy-isnt-what-you-think/>



$$\frac{4000}{\quad} \Rightarrow 3625$$

