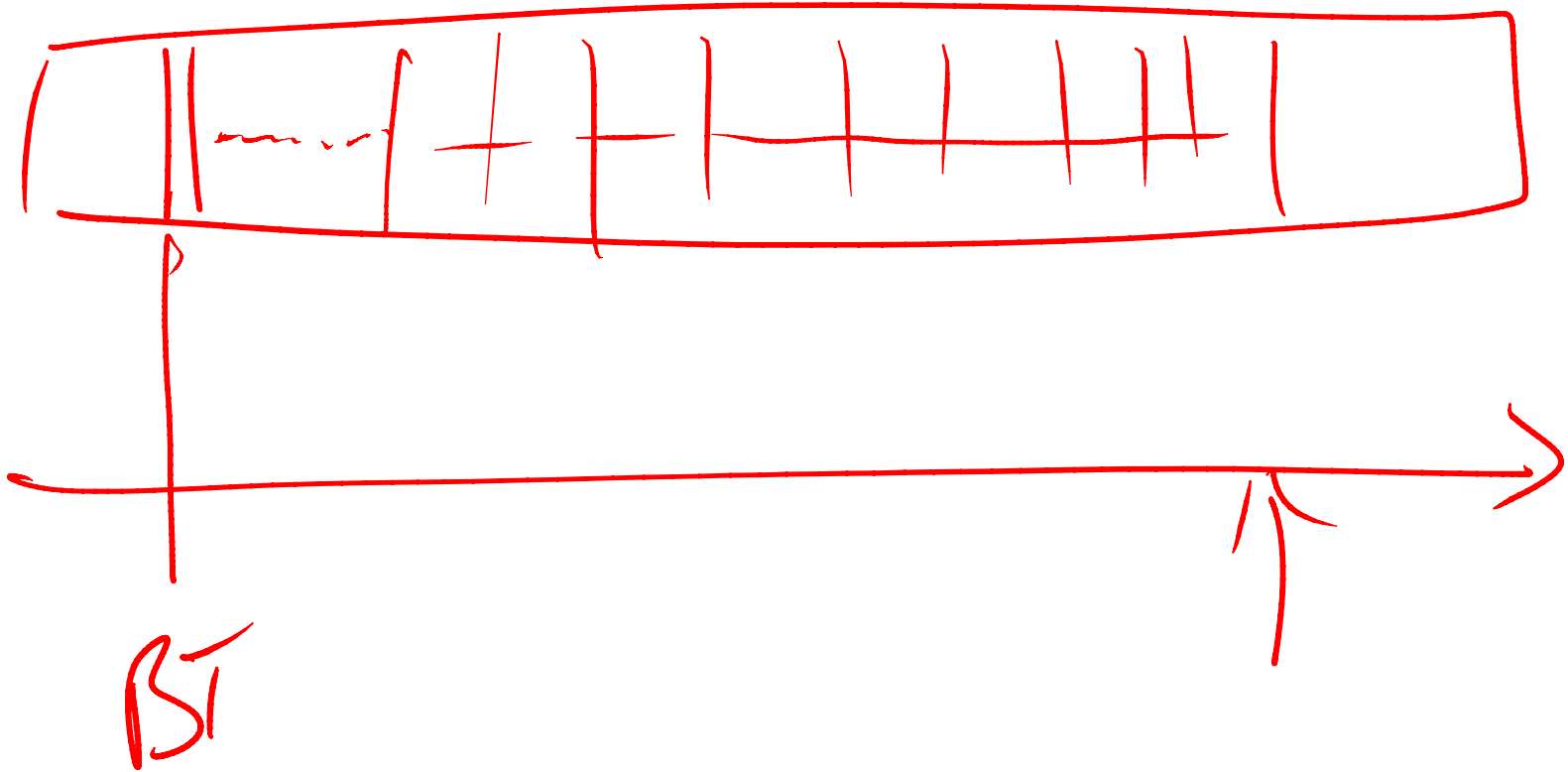




M1S7 Explaining that dividing a large database into filegroups allows targeted restores during disaster recovery, rather than having to wait for the entire single file database to restore.

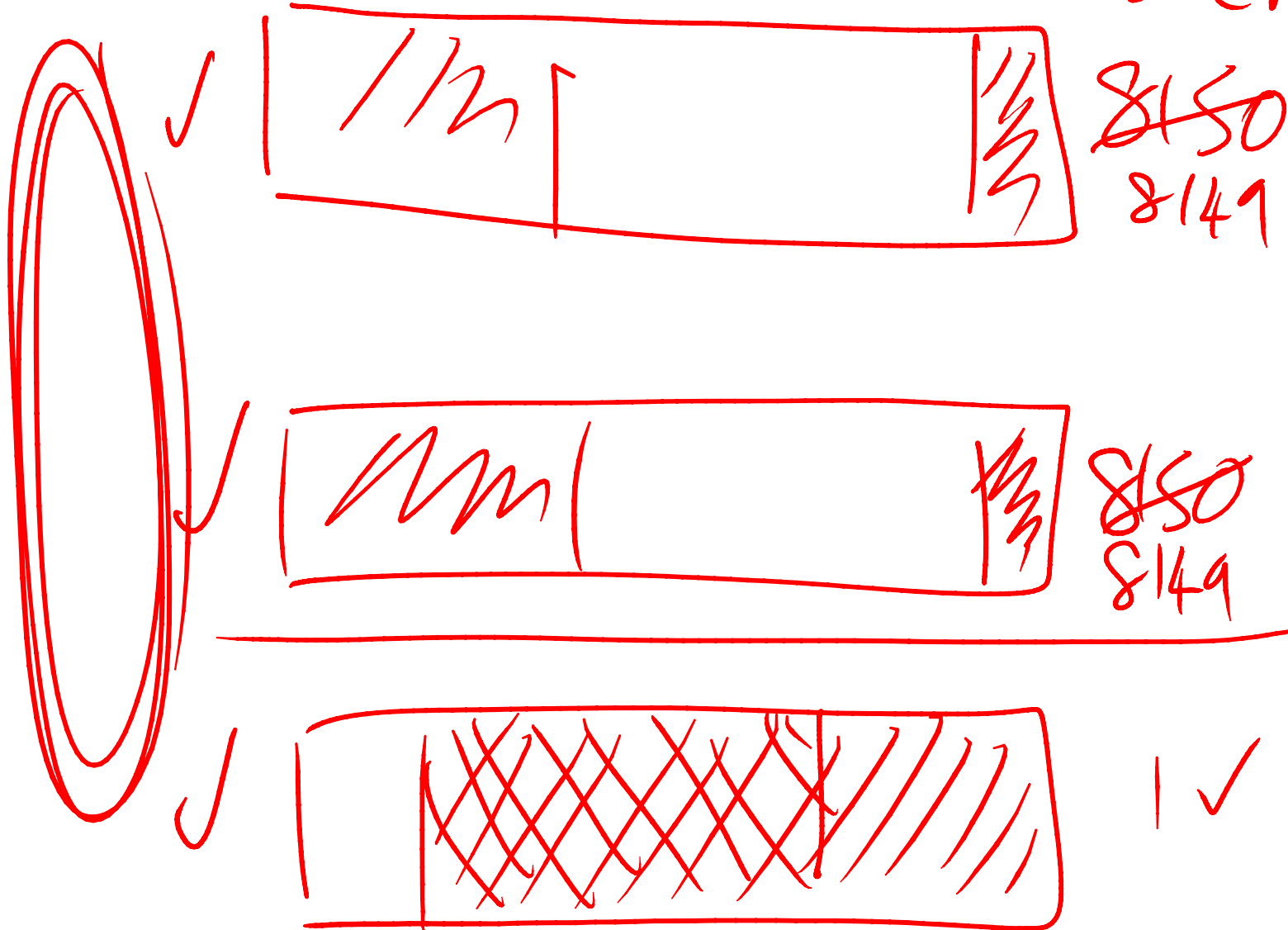
PARTIAL



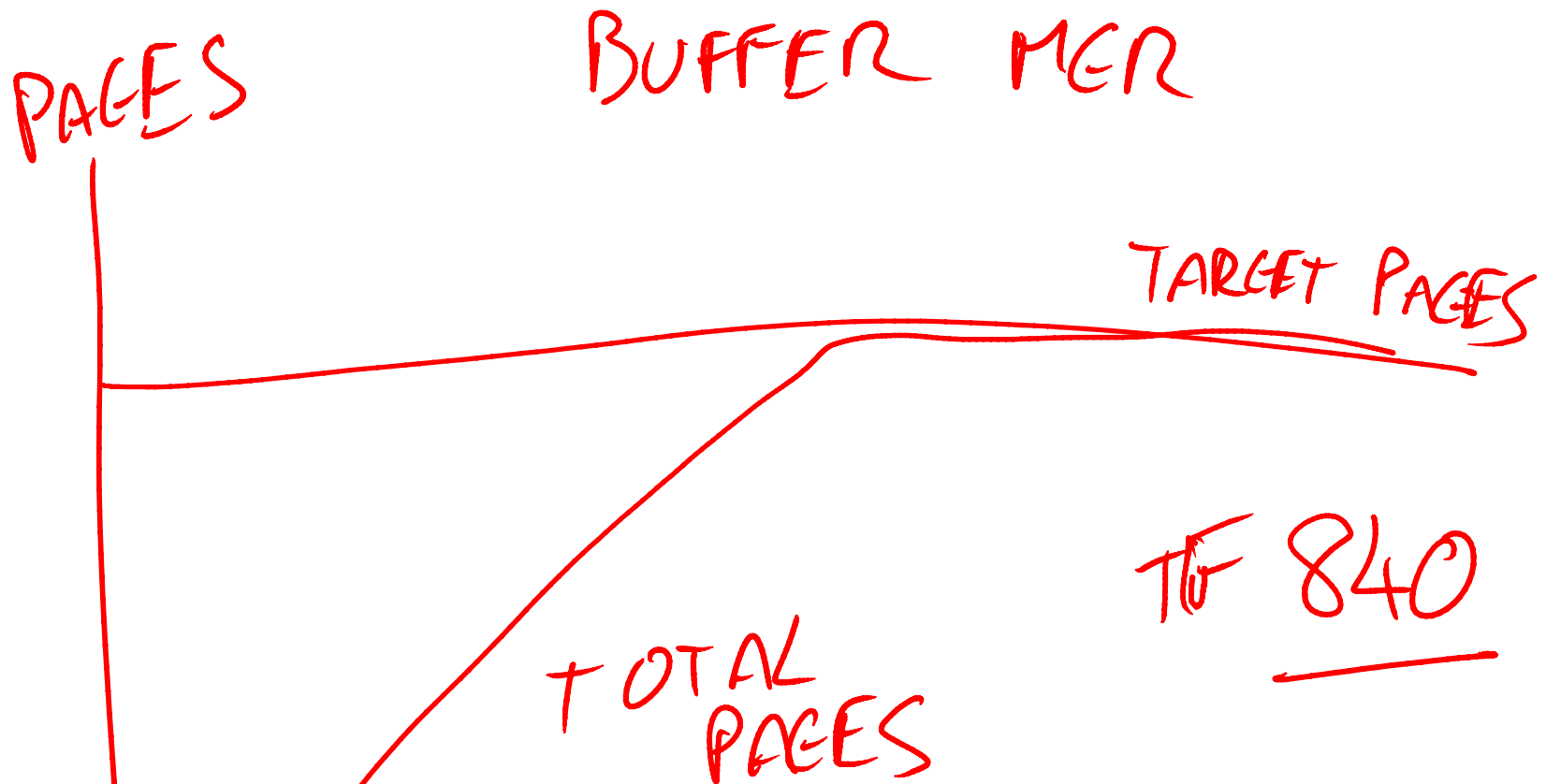


M1S28: Troubleshooting log use in tempdb. A long-running transaction starts, the log grows, but when you look, the long-running transaction hasn't use much log space. It's because a whole bunch of other transactions occurred, and the log can't clear because of the long-running transaction.

WEIGHTING

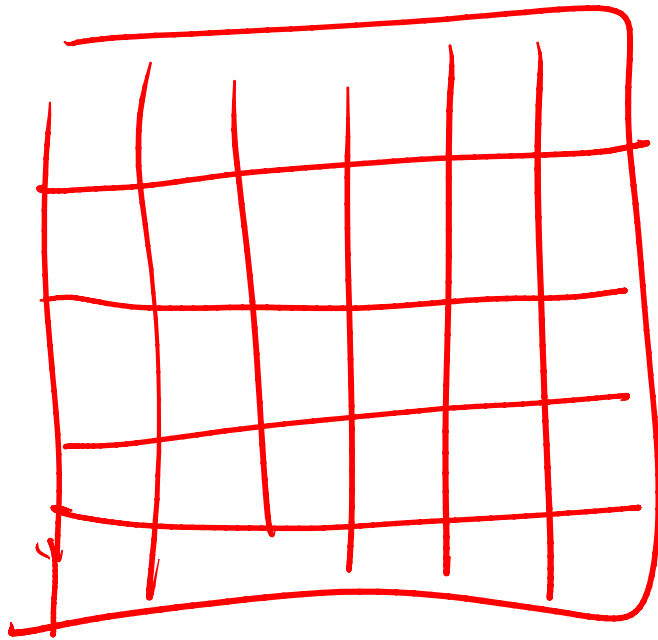


M1S30 Explaining proportional fill and round robin. Adding another file to a full filegroup does not allow for rebalancing of data.



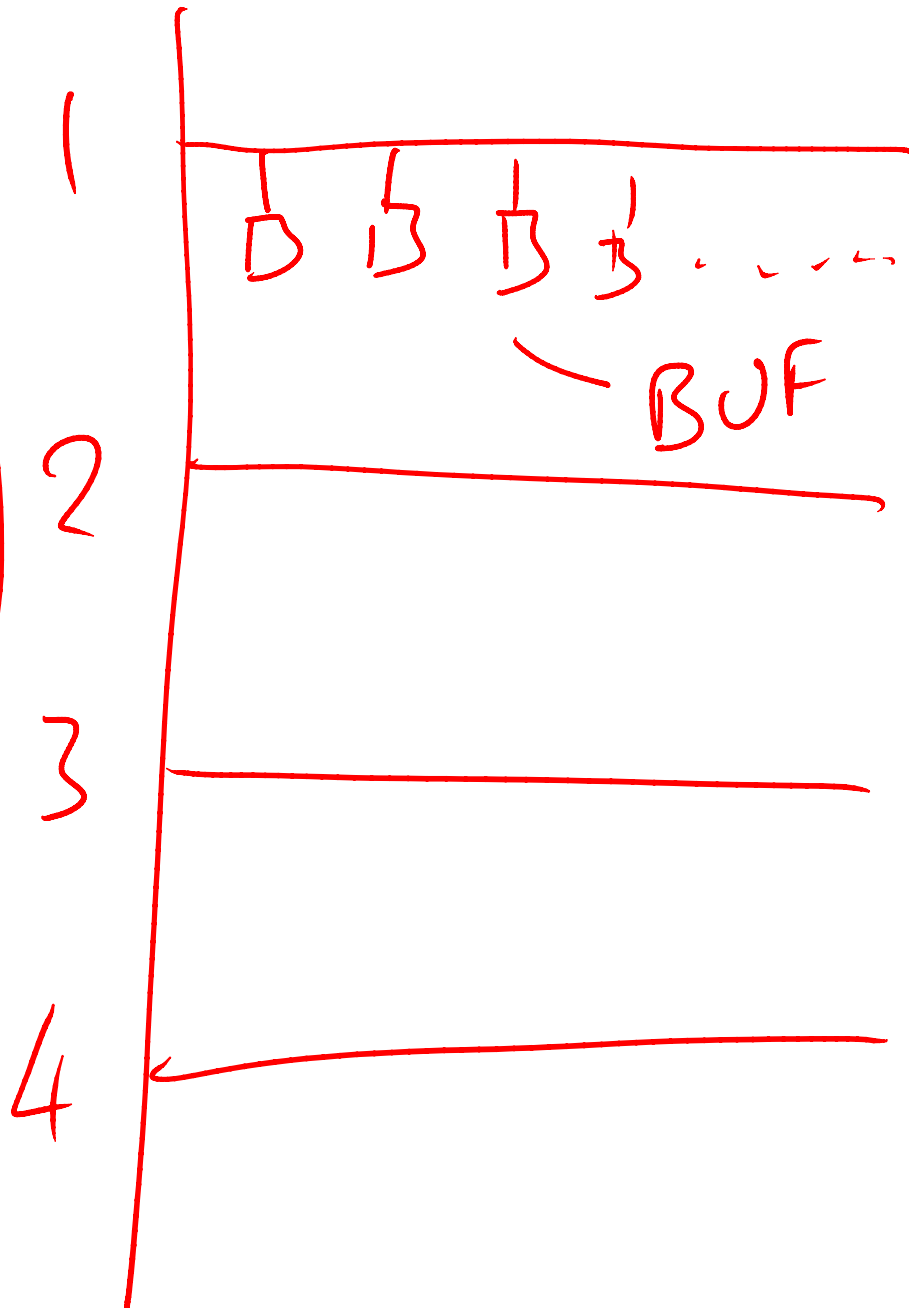
M1S34: Buffer pool memory is tracked using these perfmon counters (or Buffer Node counters if on a NUMA system). The Total Pages will ramp up after a server start. You can make it faster on non-Enterprise using documented TF 840.

7t



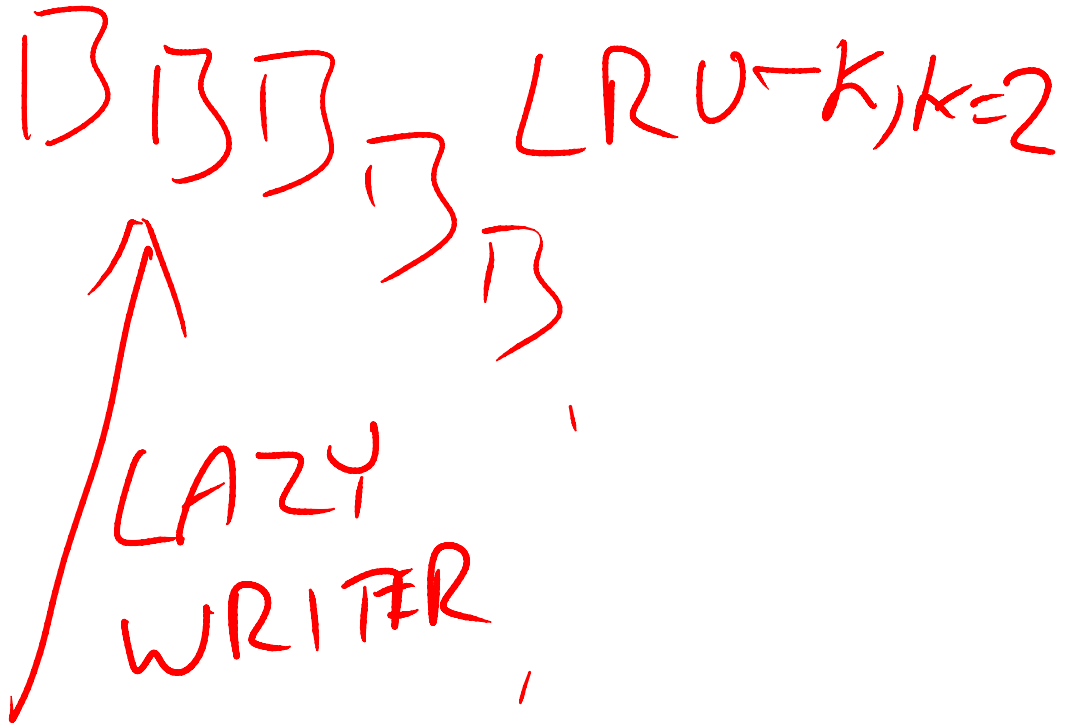
W/ASH
LIST

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.



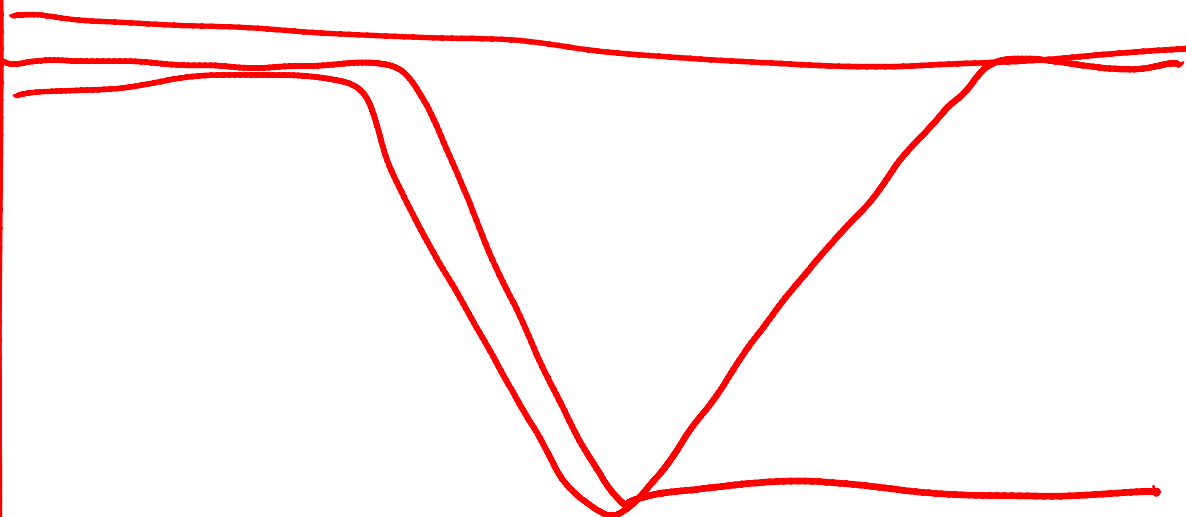
PAGE LIFE EXPECTANCY (PLE)

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.



PLE

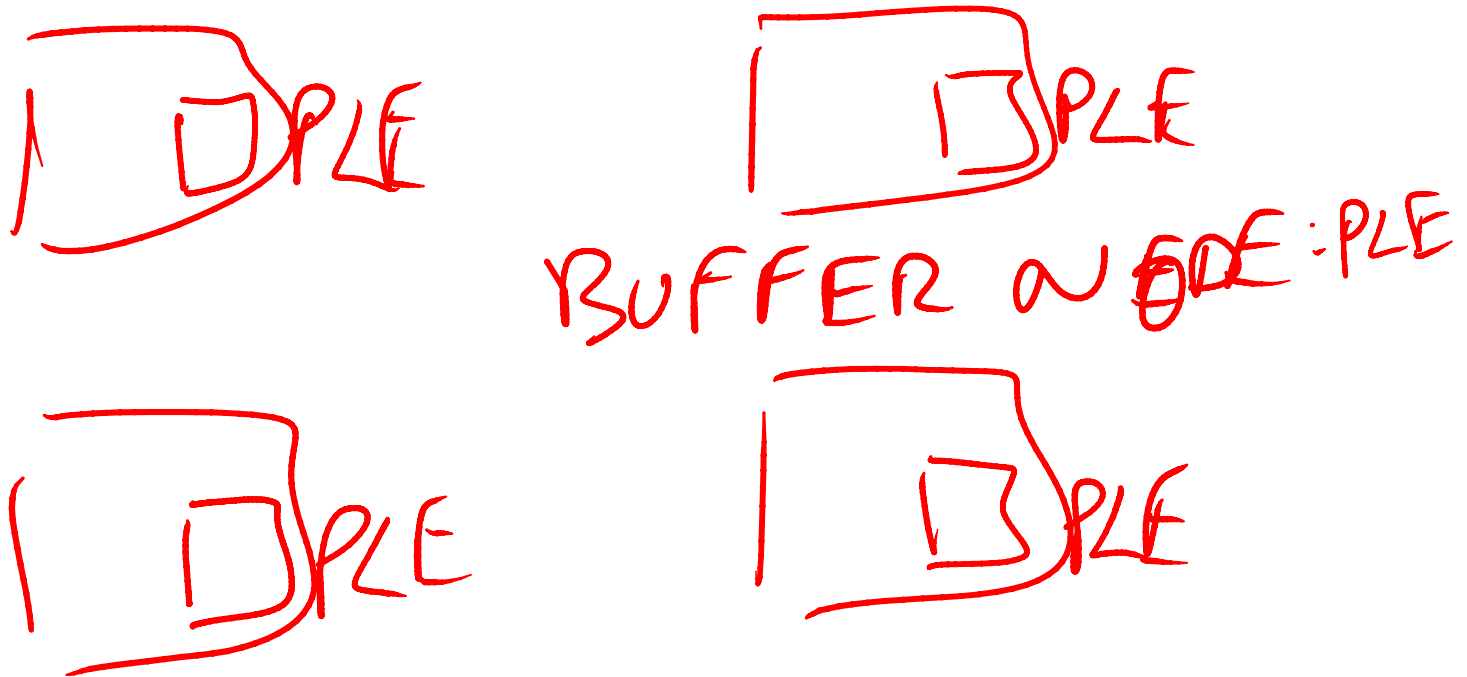
M1S34: The best PLE measurement is the one from the steady-state of your server. 300 is nonsense. If your PLE drops and comes back up, that's normal. If it drops and stays down, that's memory pressure (or lots of reads) you should investigate.



t

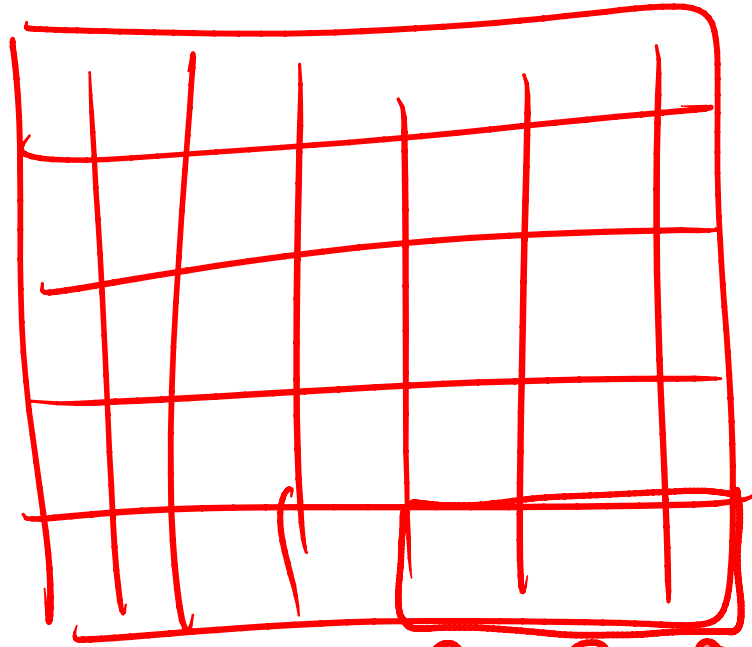
BUFFER MGR : PLE

NUMA



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/>

M1S34: (We had a lot of discussions here!) Some operations will do buffer pool disfavoring – see <http://www.sqlskills.com/blogs/paul/buffer-pool-disfavoring/>

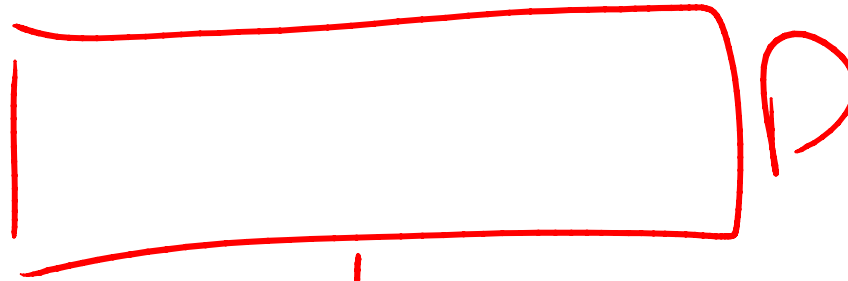


~~R R R~~
L L L
~~R R R~~
L L L

LRU

BUFFER
POOL

DISFAVORING

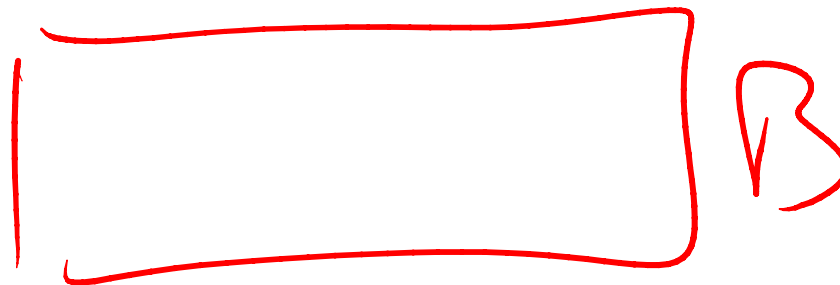


M1S36 Explaining that backup striping makes restores and backups faster. One reader thread per distinct drive letter/mount point, and one writer thread similarly.

↓ 1 READER

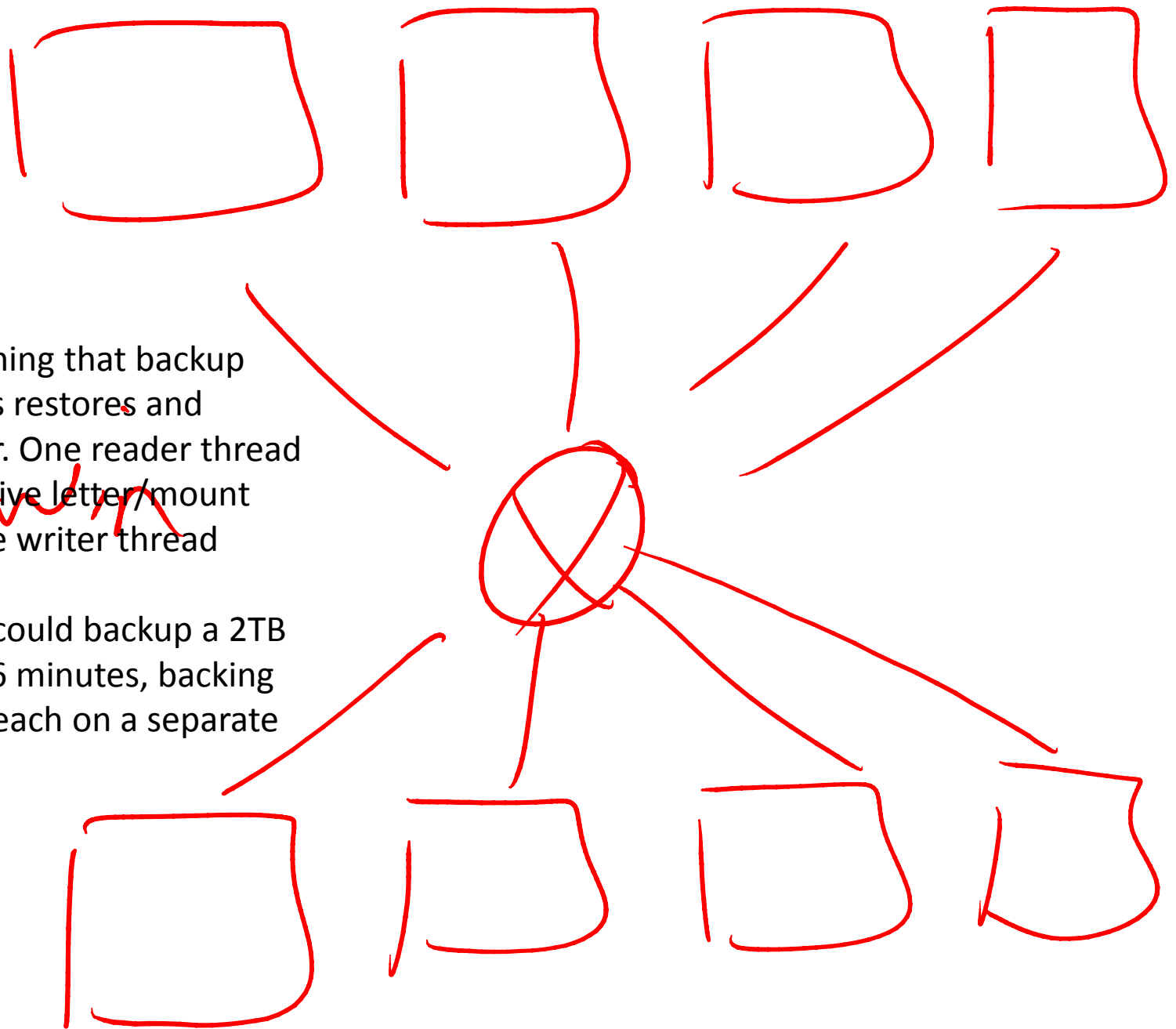
Bwin in 2005 could backup a 2TB database in 36 minutes, backing up to 12 files each on a separate drive.

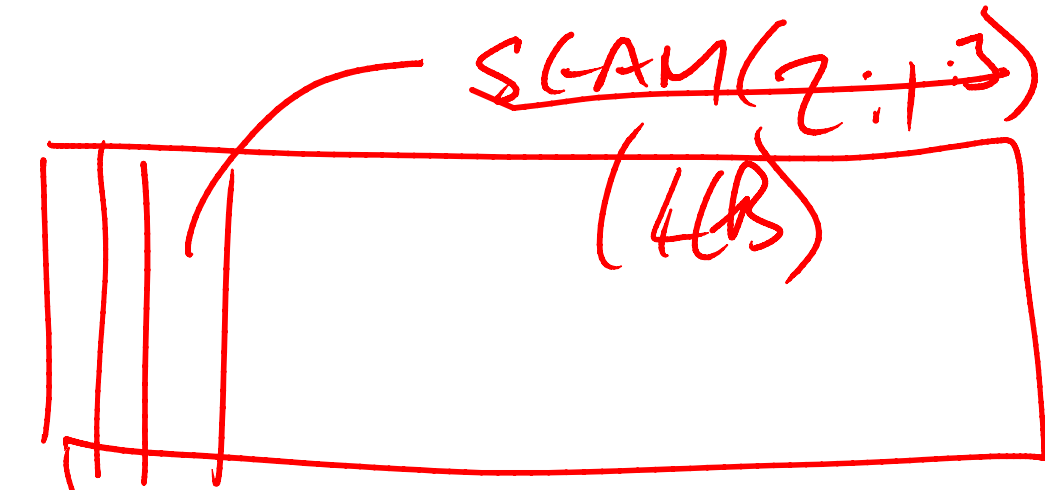
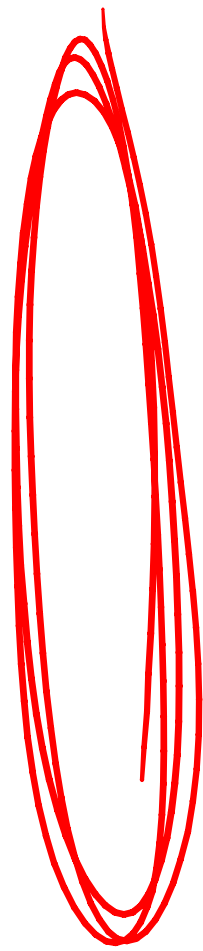
↓ 1 WRITER



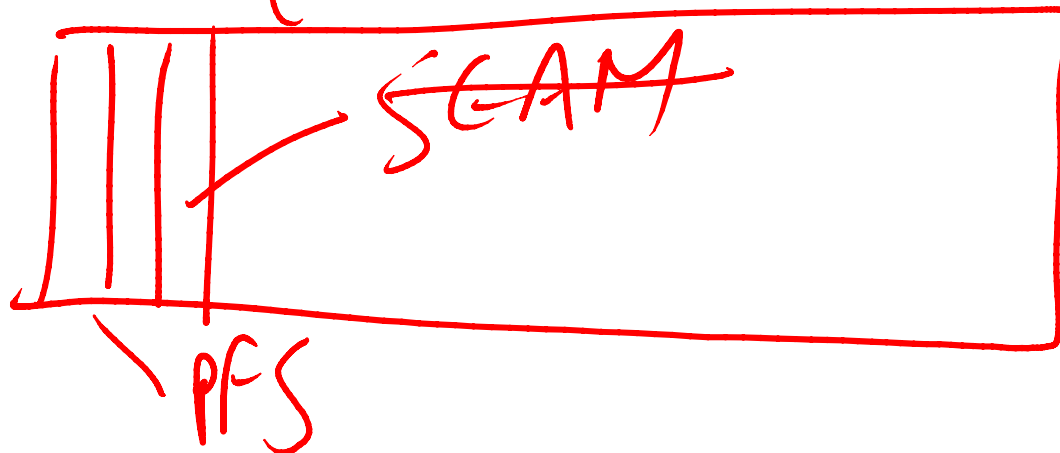
M1S36 Explaining that backup striping makes restores and backups faster. One reader thread per distinct drive letter/mount point, and one writer thread similarly.

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

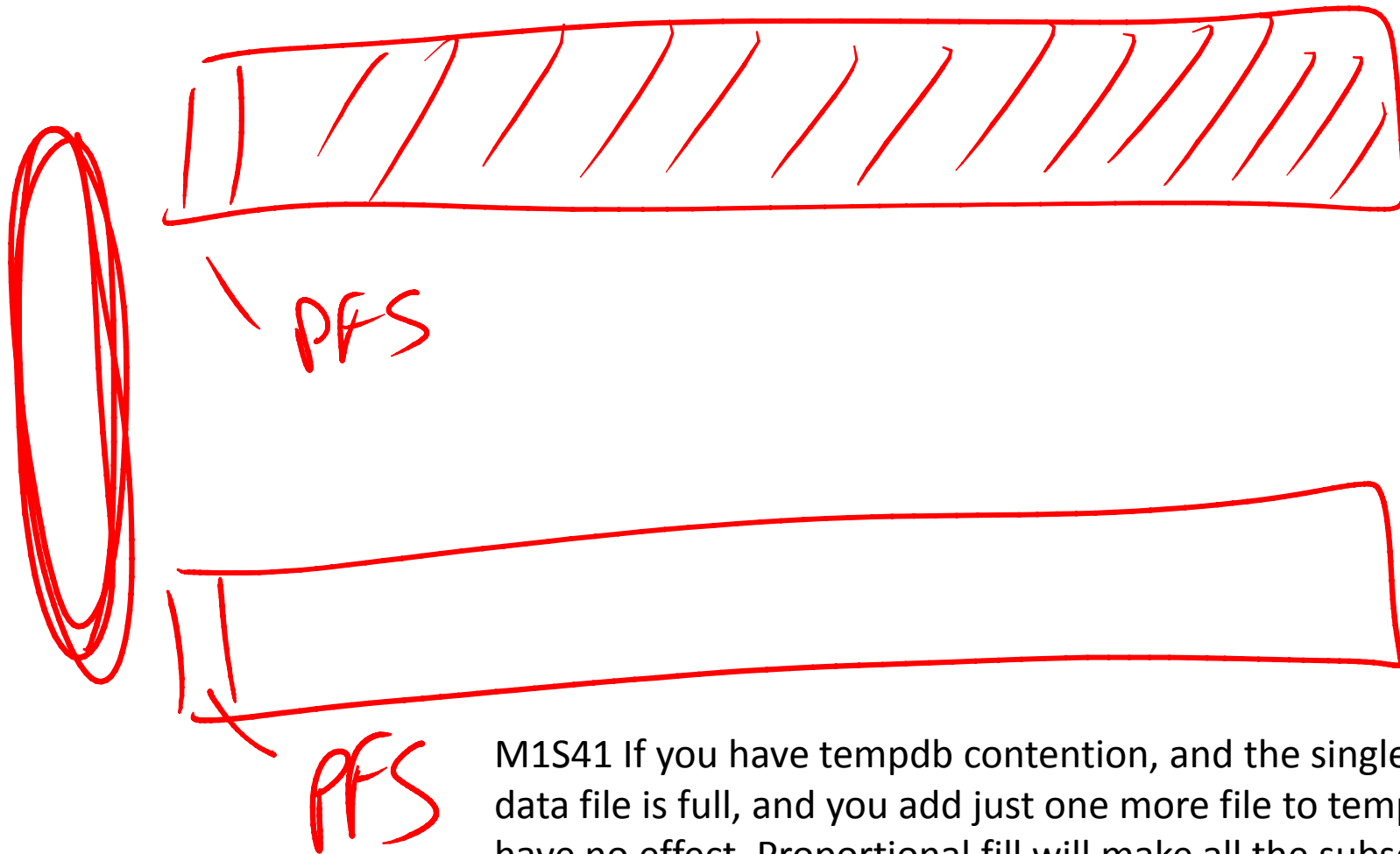




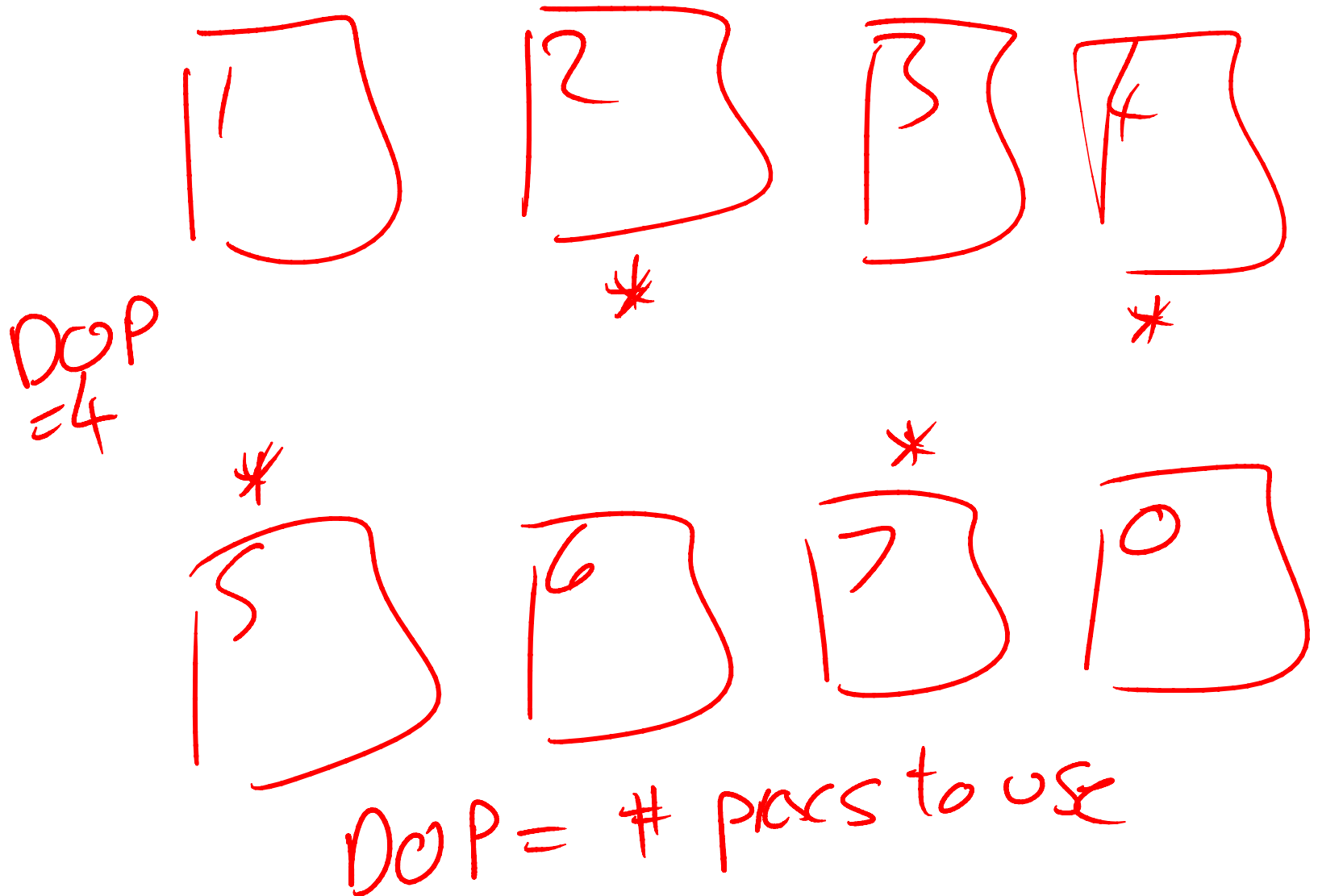
PFS(2:1:1)
(64MB)



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



M1S41 If you have tempdb contention, and the single tempdb data file is full, and you add just one more file to tempdb, it will have no effect. Proportional fill will make all the subsequent allocations come from the new file, so all the contention will be transferred to the PFS page at the start of the new file. See <http://www.sqlskills.com/blogs/paul/correctly-adding-data-files-tempdb/>



M7S10 Distribution of threads in a parallel query. DOP = number of processors to use. No matter how many parallel threads a query uses, they will be distributed among the same DOP schedulers.

PAGE

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

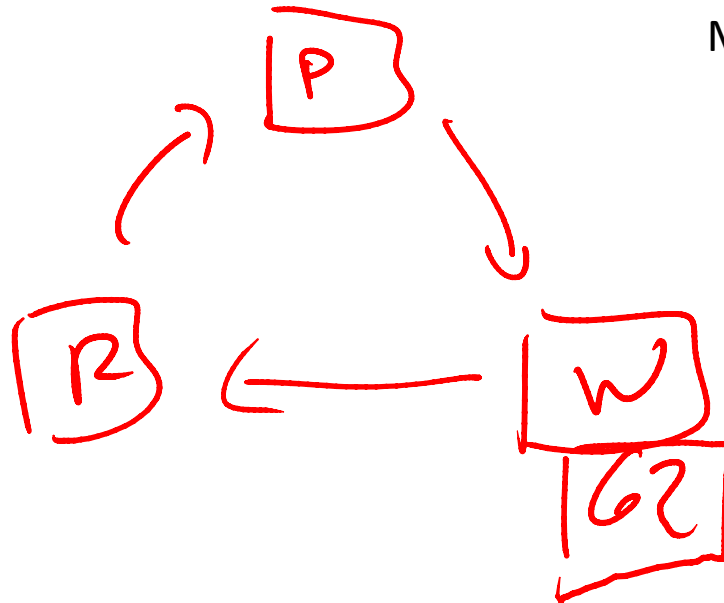
GRANTED
LIST

X	61
---	----

PENDING QUEUE

5	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 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 its 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.

PAGE

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

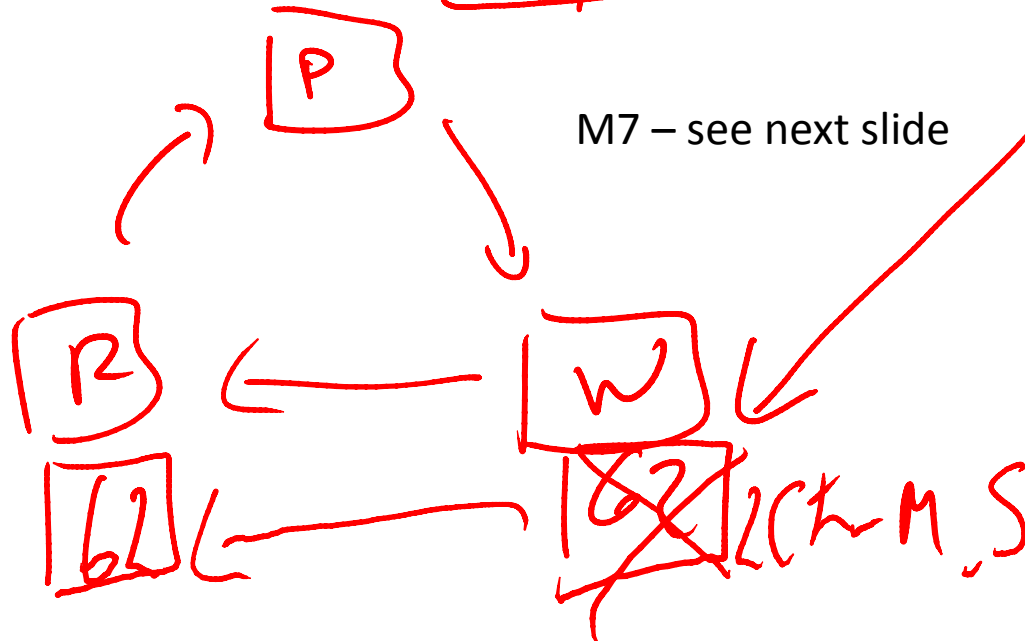
GRANTED
LIST

PENDING QUEUE

X	61
S	62

S	62	ADDR
--------------	---------------	------

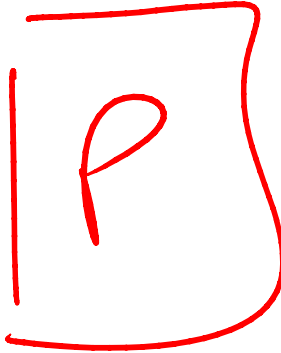
M7 – see next slide



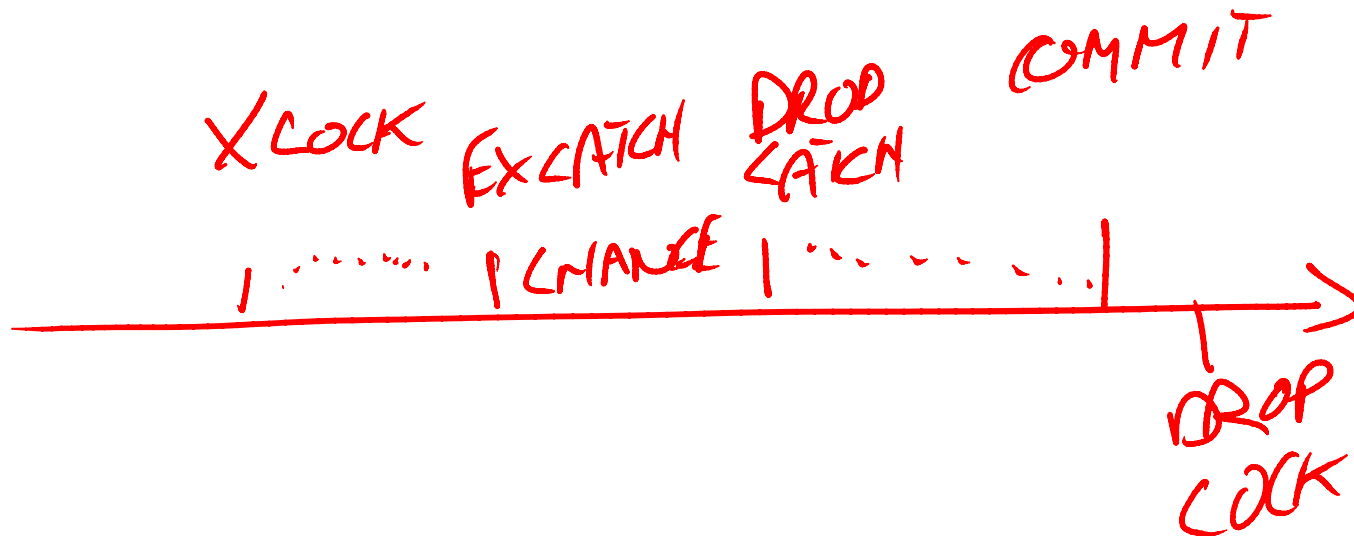
M7 Explaining how the lock pending queue works.

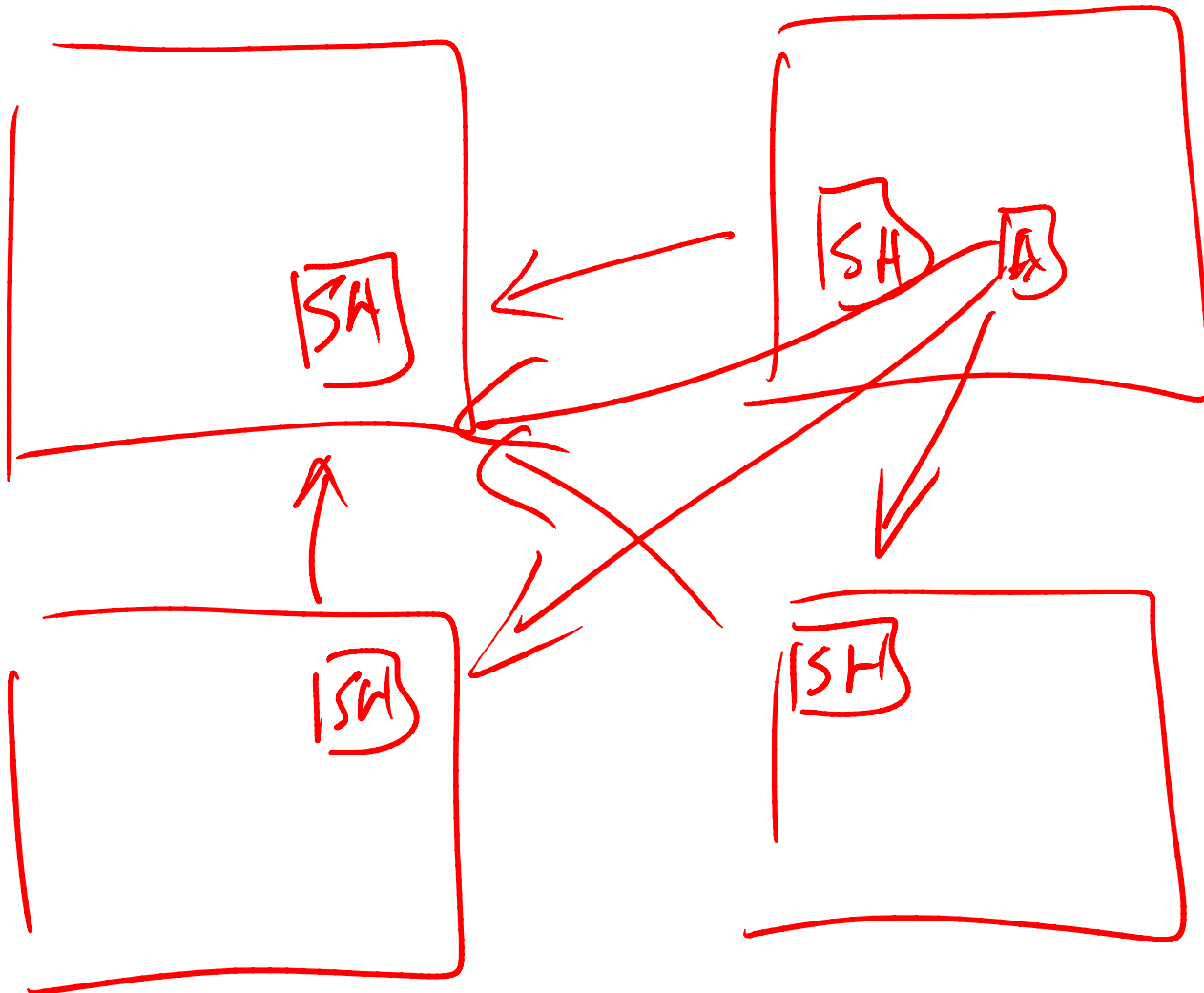
At time:

X+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.



M7S29 Explaining how long a latch is held – only as long as the data structure access is needed. If a transaction is inserting a record on a page, the row is X locked and the lock is held until transaction commit. To actually change the page, an EX latch is required on the page, which is only held until Access Methods has finished changing the page.

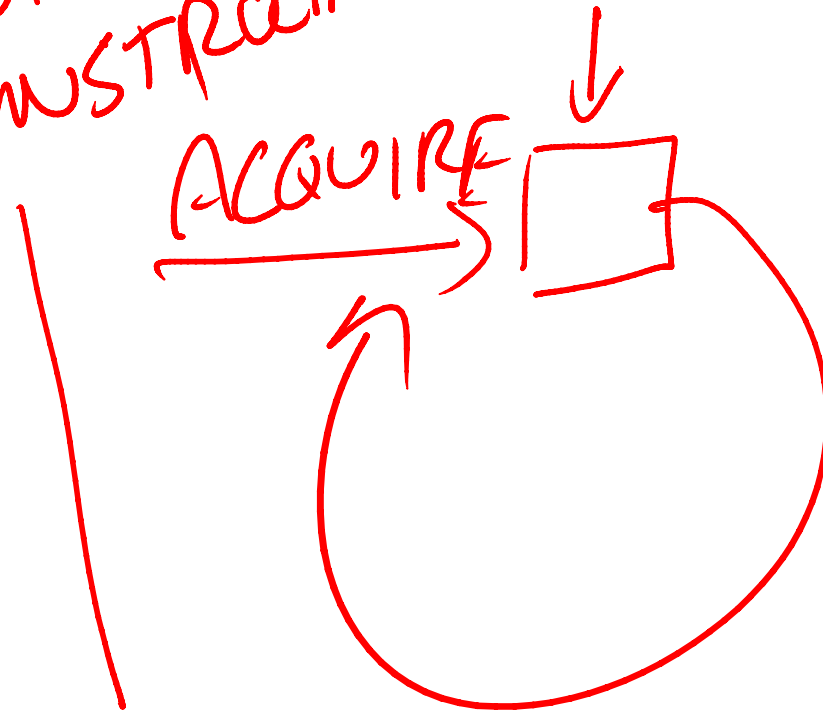




NUMA

M7S33 Explaining about
superlatches.

INTERLOCKED
INSTRUCTION SPINLOCK



SPINNING

1000

COLLISION

TST_SET_IF_CLEAR

BACK OFF

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.

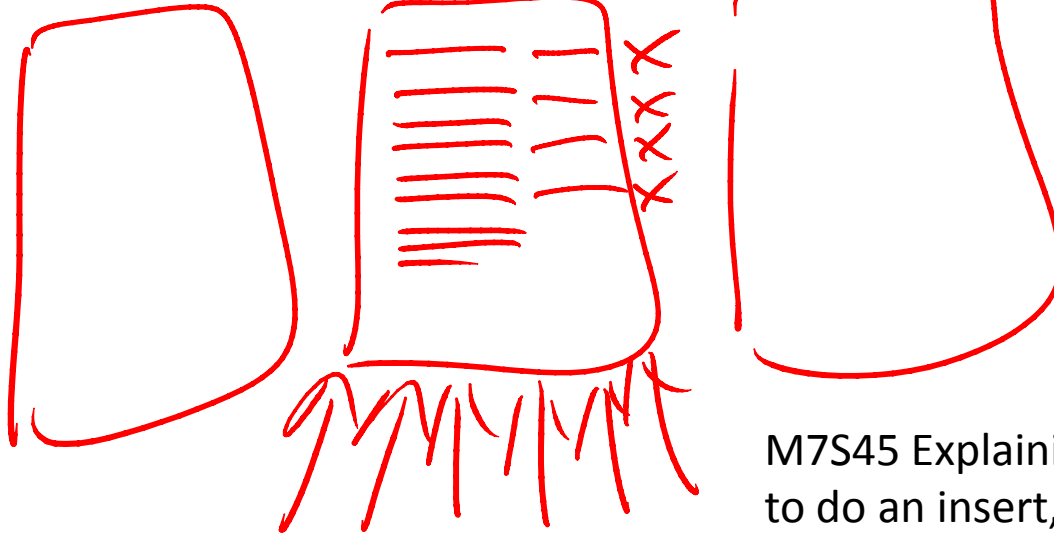
TIX

TIX = table IX lock

PIX = page IX lock

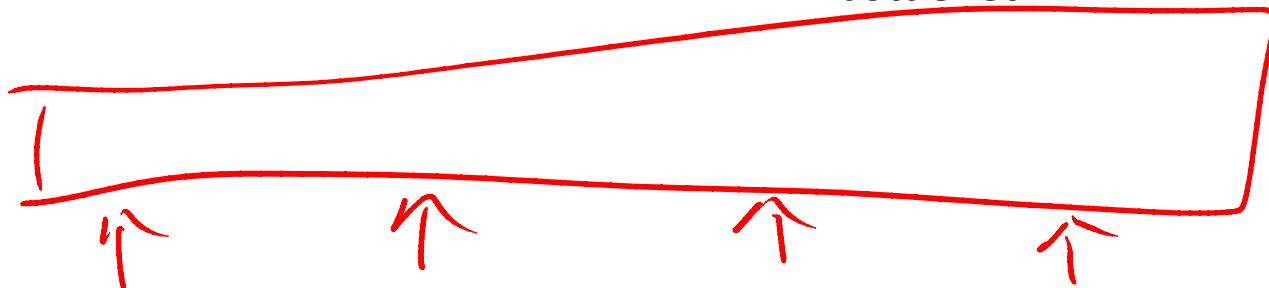
And there are row X locks

SOPIX



X X X
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 X latch on the page, which becomes a bottleneck.



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