

2 TB SALES

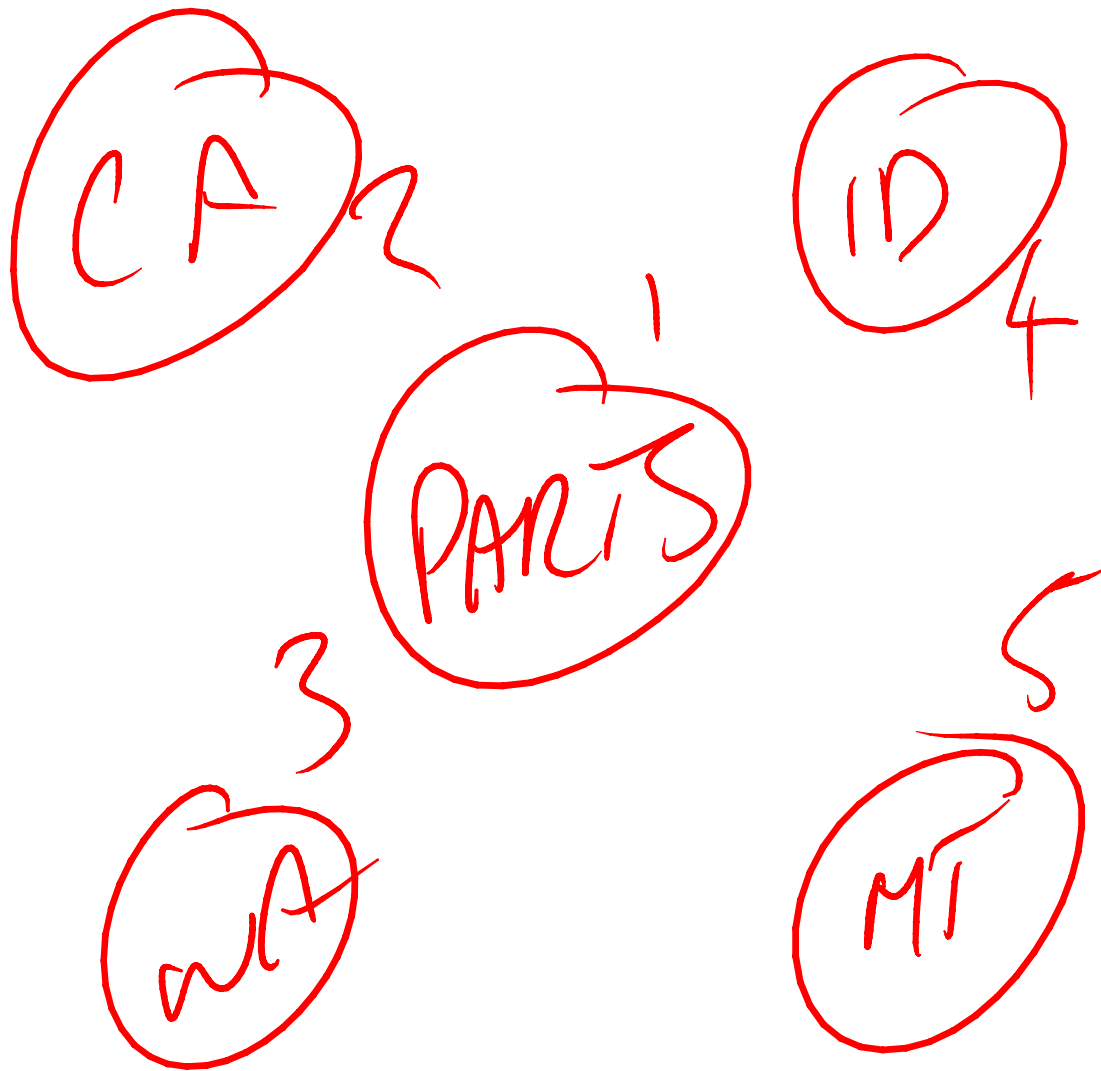
2011 - - - - - 2014

DW

OLTP

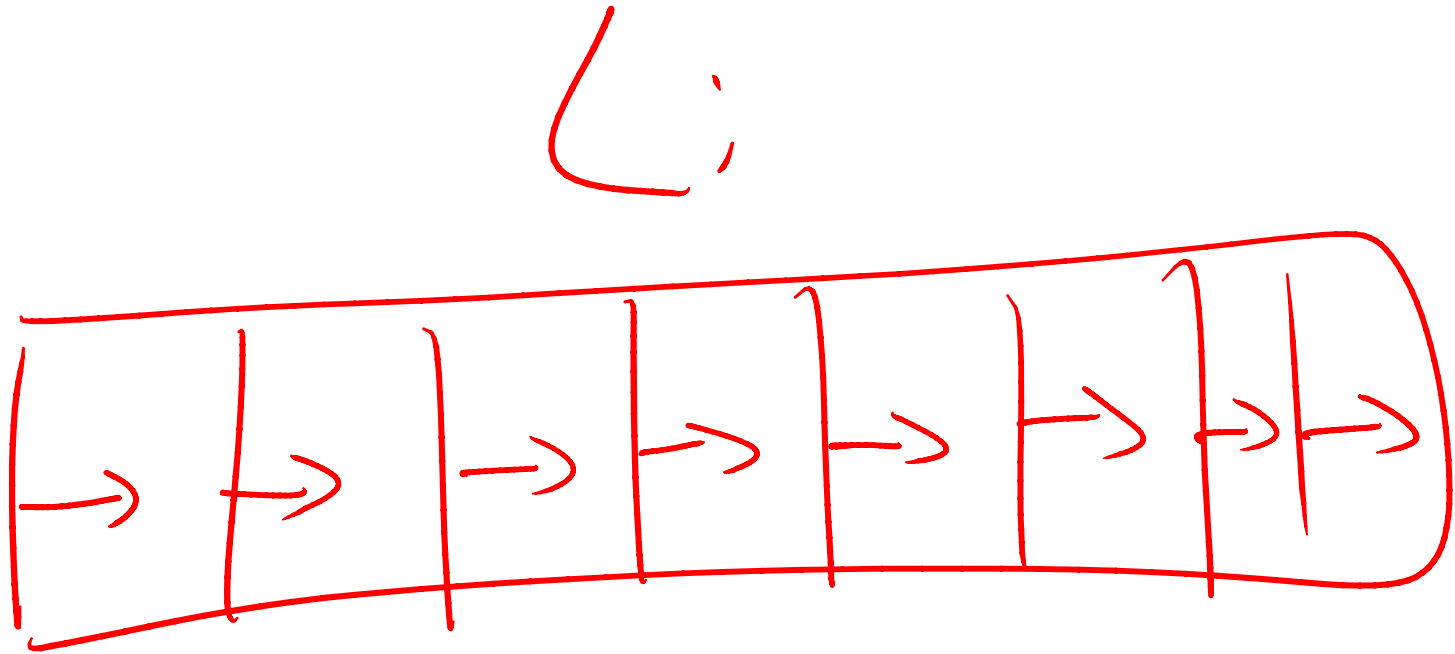
M1S7 Explaining that dividing a large database into filegroups allows targeted restores during disaster recovery, rather than having to wait for the entire single file database to restore.

P 2014 2013 2012 2011



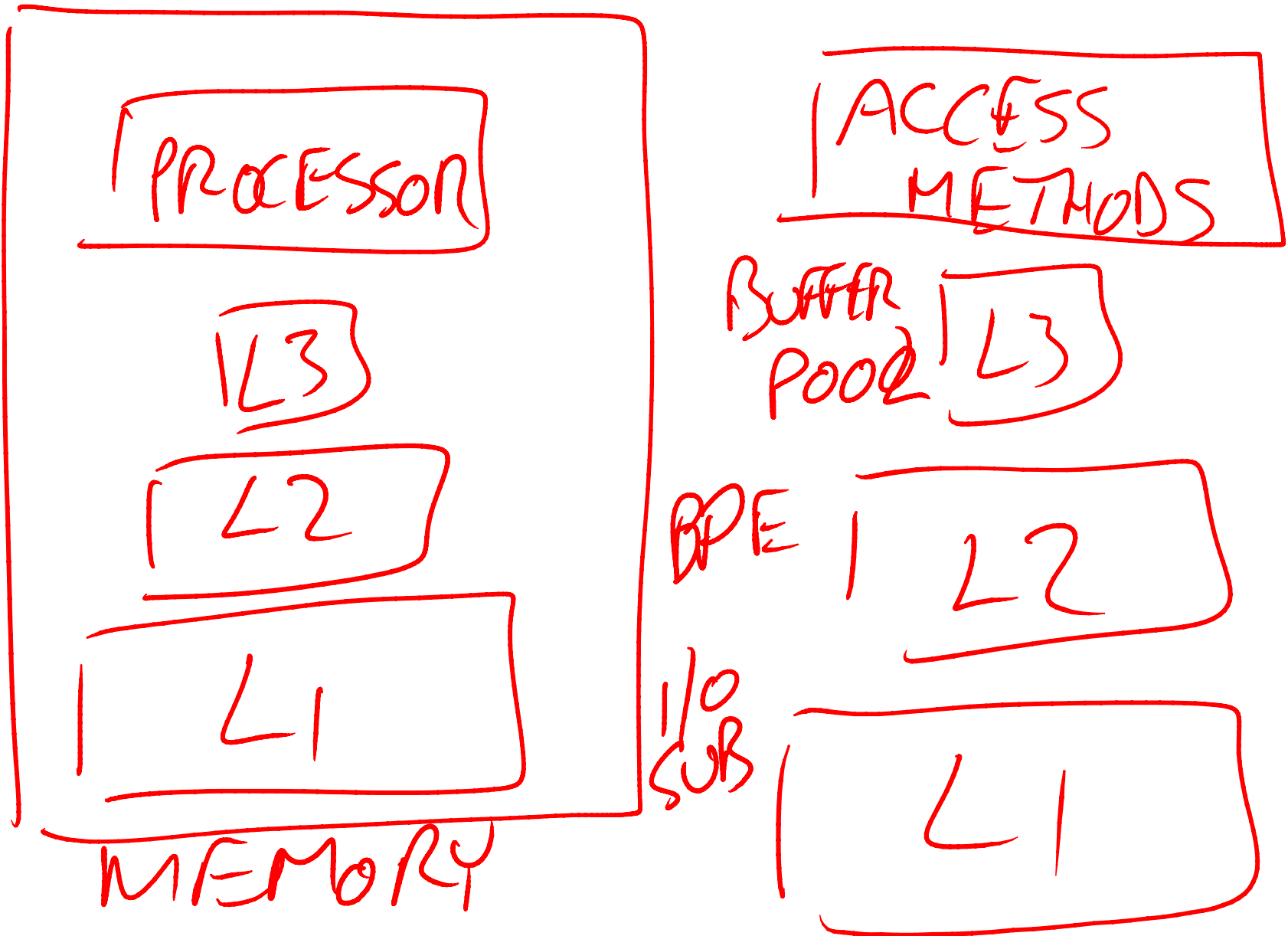
M1S7 Explaining a different partitioning scenario – auto parts sales to WA/CA/ID/MT.

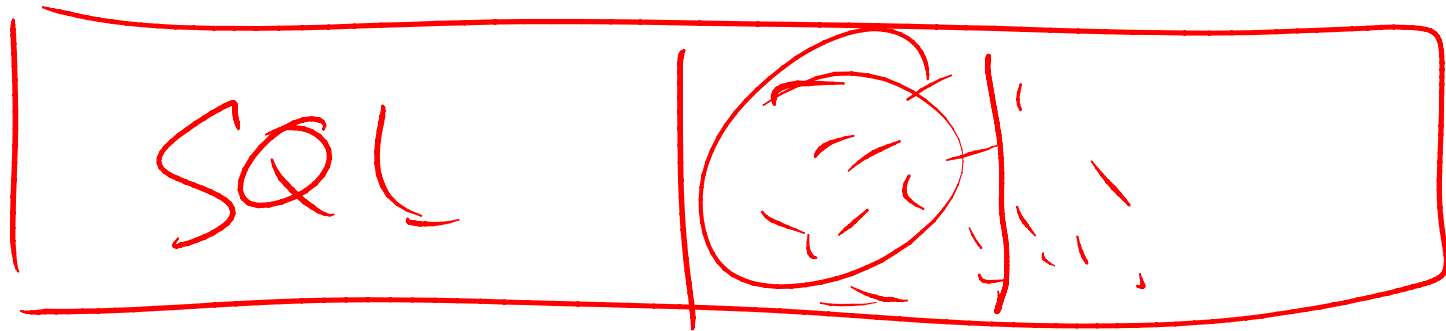
If things go down, it makes sense to restore the portion of the database that will bring in the most sales quickly – CA, as it has the most population.



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.

M1S13 Explaining 2014 Buffer Pool Extension. It's like have L1-L3 caches for your data in memory. BPE sits in between memory and the I/O subsystem.





SA

DBCC PAGE

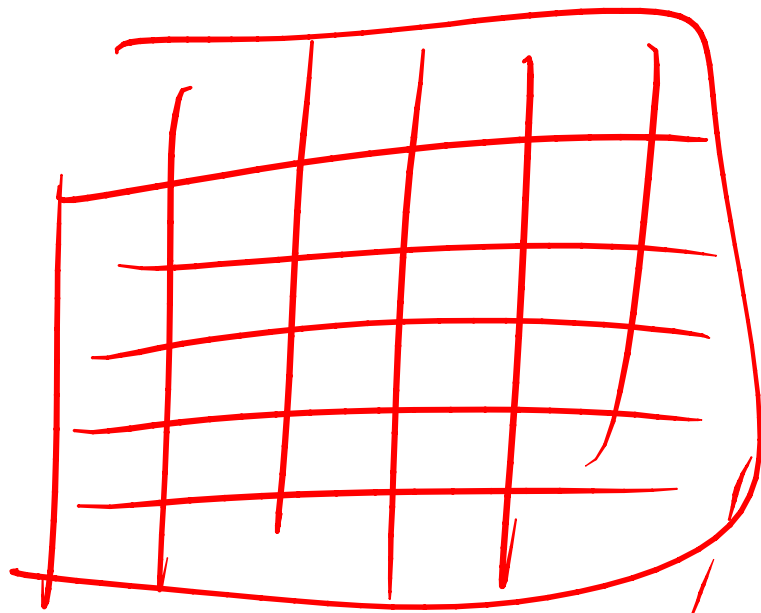
M1S14 Explaining why instant file initialization is off by default. Sharing a volume between SQL Server and 'secure files' can lead to information leakage if the secure files are deleted, then SQL grows a data file, and an SA can use DBCC PAGE to get a hex dump of the bits/bytes that comprised the deleted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

M1S14 How to tell if you have Instant Initialization enabled if you don't have Windows access. Enable these two trace flags and create a dummy database. If you see a line in the error log about zeroing out the data file, you do NOT have Instant Init enabled. Get Windows admin to grant Perform Volume Maintenance Tasks privilege and restart SQL Server.

3004

3605

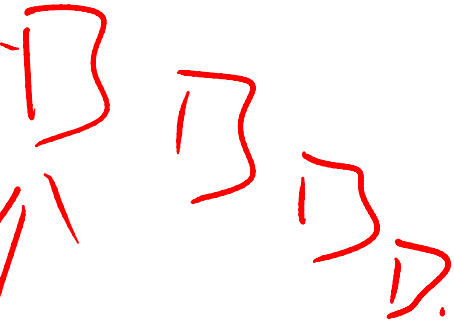
DBCC TRACEON(3004,
3605, -1)



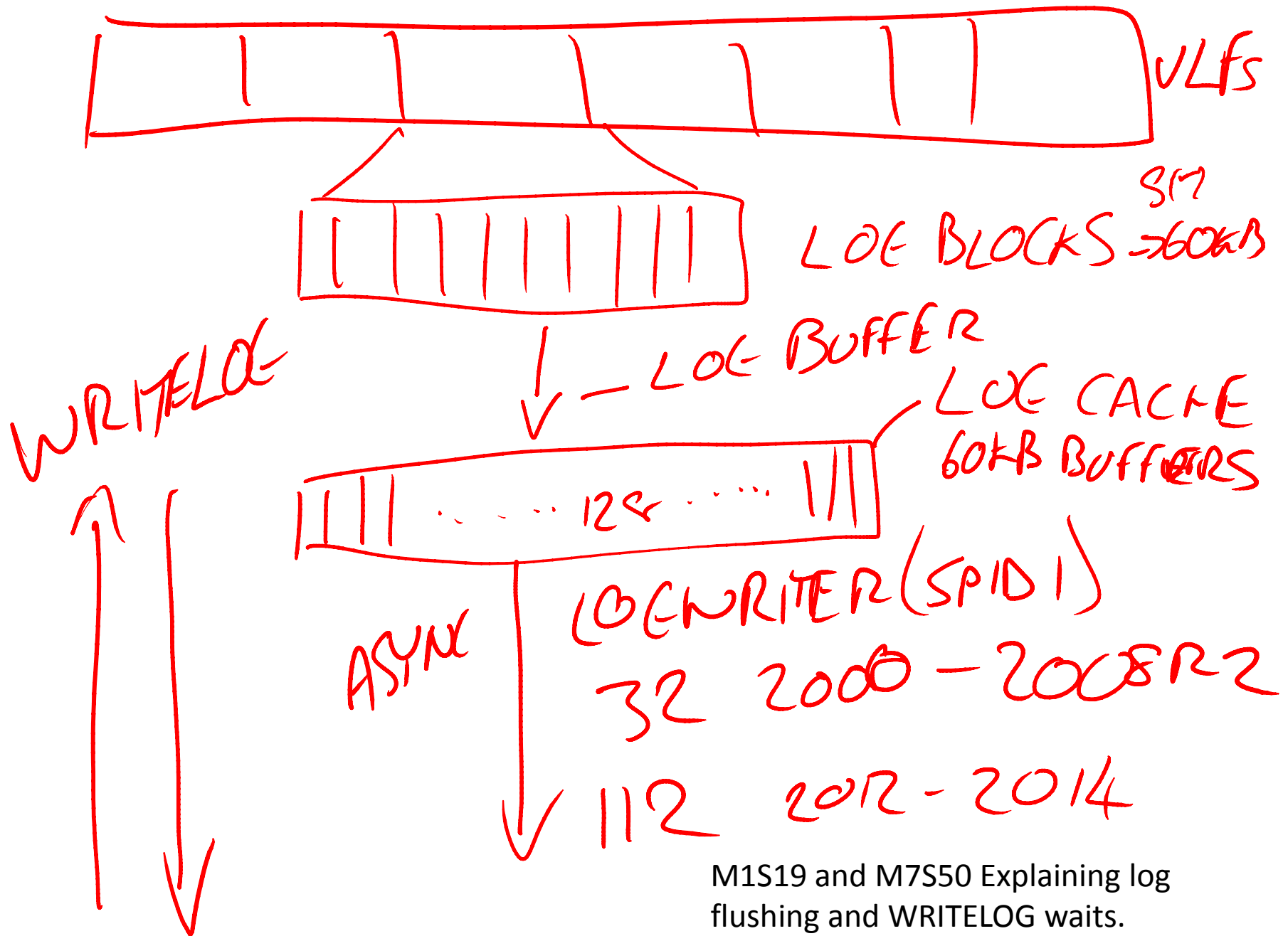
LRU

CHECKPOINT

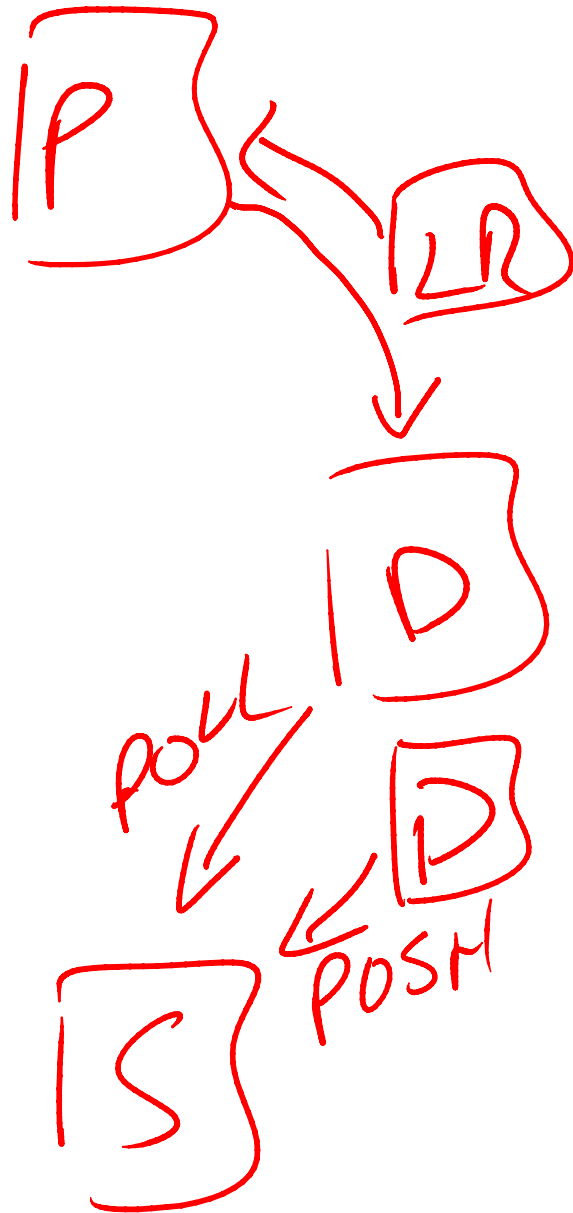
LAZY
WRITER



M1S15 Explaining how usually data file pages are written by periodic checkpoints, but there's also the background lazy writer task in the buffer pool that is responsible for ensuring there is free space. It implements a Least Recently Used algorithm. All the buffers in the buffer pool can be thought of as points on a large clockface, and the lazy writer as a sweeping second hand, evaluating buffers in turn.



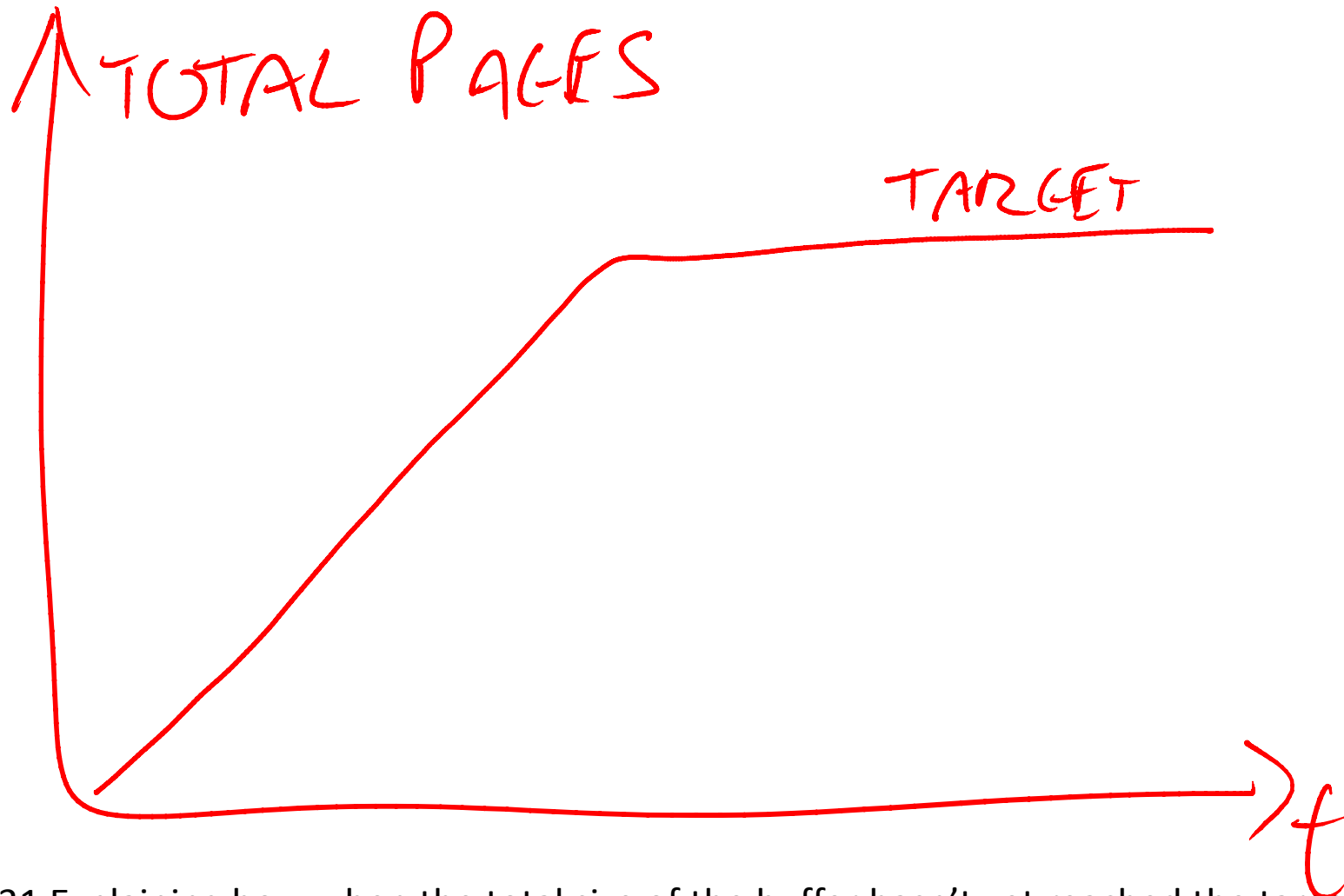
M1S19 and M7S50 Explaining log flushing and WRITELOG waits.
LOGWRITER is the log writing thread.



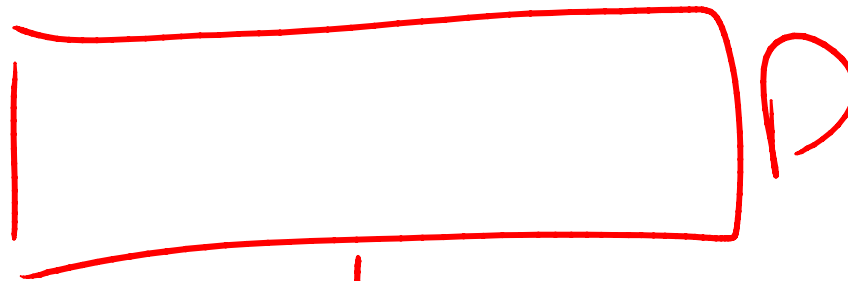
M1S21 Explaining how transactional replication works. The Log Reader Agent reads from the publication log, harvesting committed transactions and putting them into the distribution database. The Distribution Agent either pushes or pulls them to the subscription database. Pull subscriptions are more efficient and less prone to bottlenecks.

```
WHILE @@TRANCOUNT > 0  
    COMMIT TRAN;
```

M1S28 When developers don't know their transaction nesting level (nested transactions don't exist, remember), code like the above can be seen. Yuck.



M1S31 Explaining how when the total size of the buffer hasn't yet reached the target size, each page read is converted to an extent read, in Enterprise Edition. All other Editions can do this using TF 840.

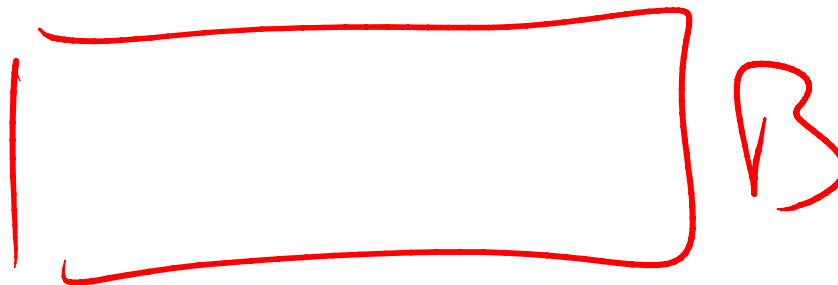


M1S33 Explaining that backup striping makes restores and backups faster. One reader thread per distinct drive letter/mount point, and one writer thread similarly.

↓ 1 READER

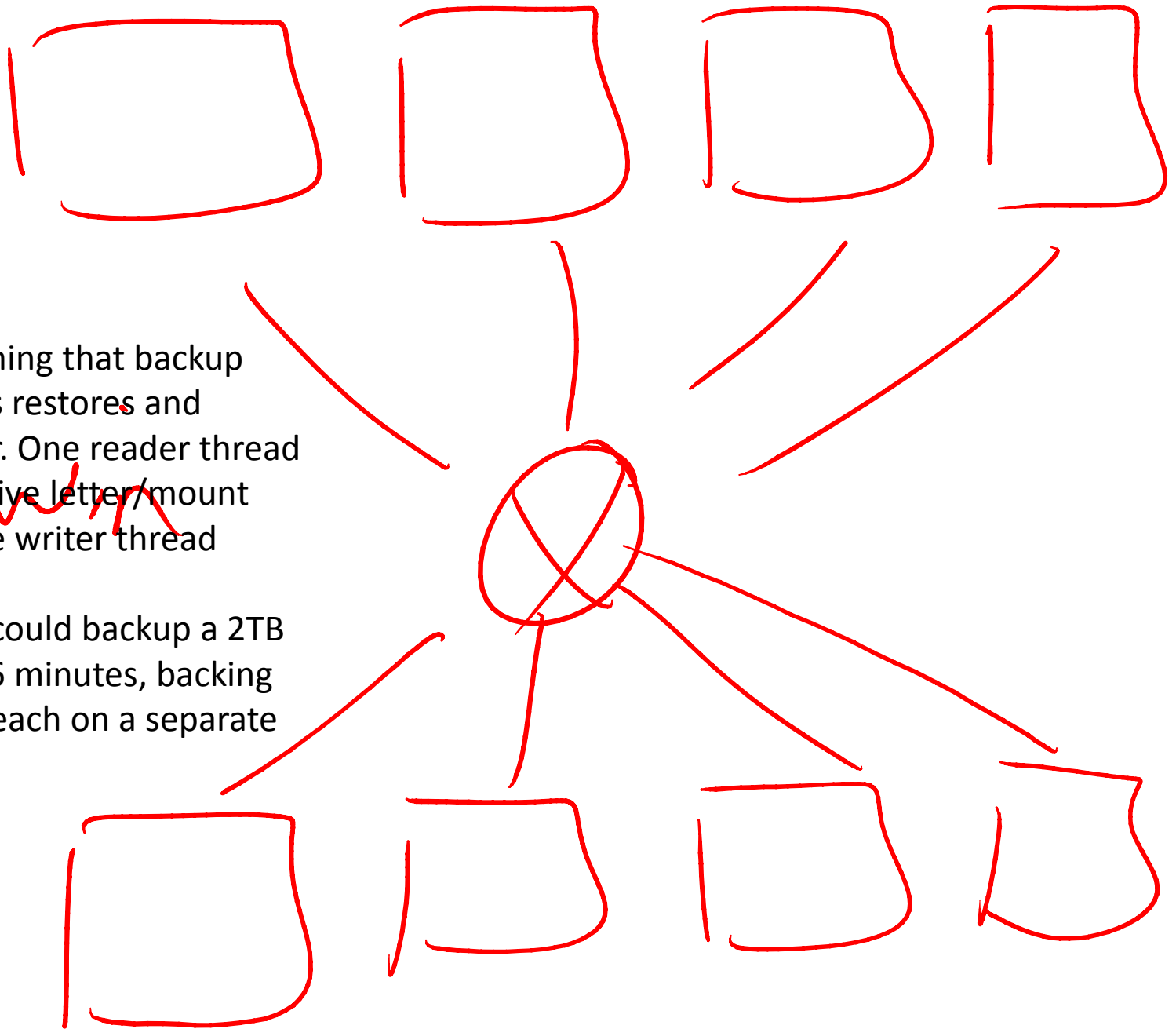
Bwin in 2005 could backup a 2TB database in 36 minutes, backing up to 12 files each on a separate drive.

↓ 1 WRITER

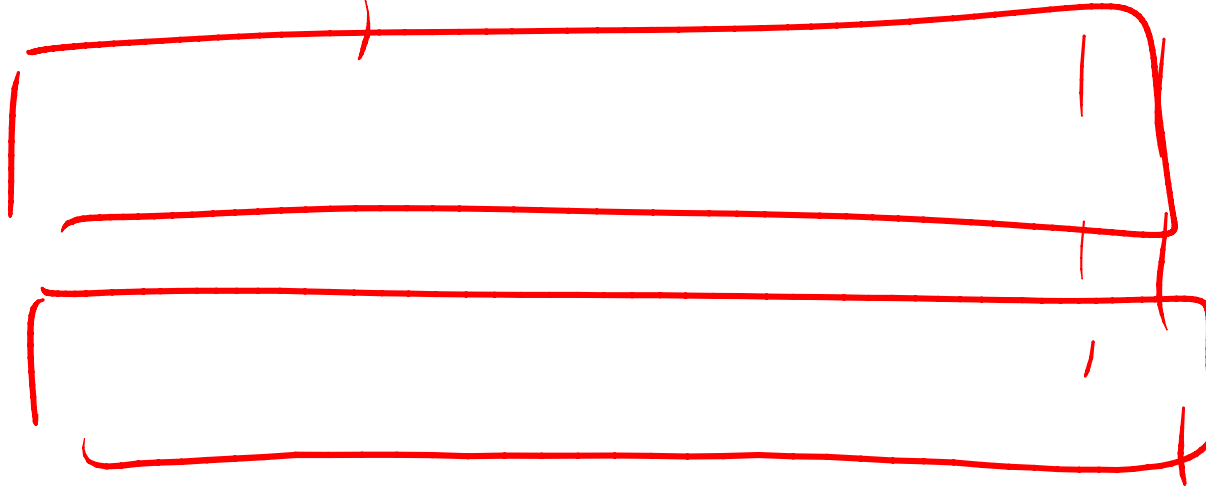
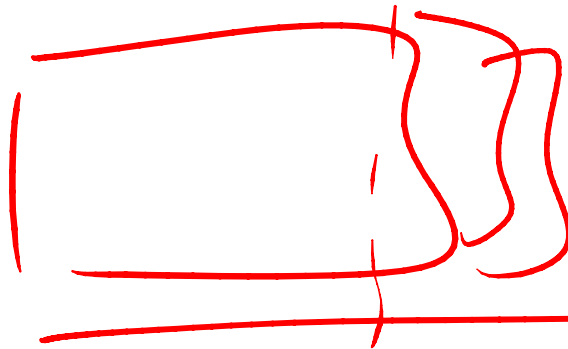
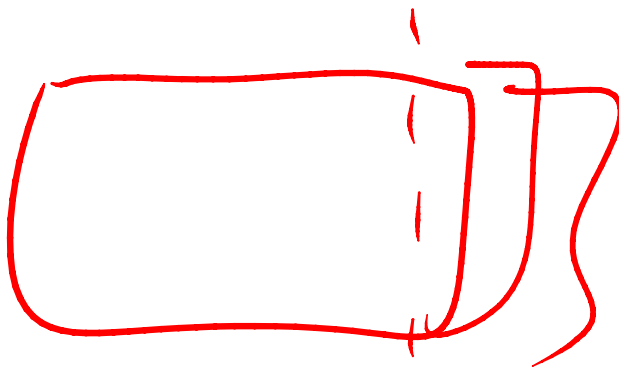


M1S33 Explaining that backup striping makes restores and backups faster. One reader thread per distinct drive letter/mount point, and one writer thread similarly.

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



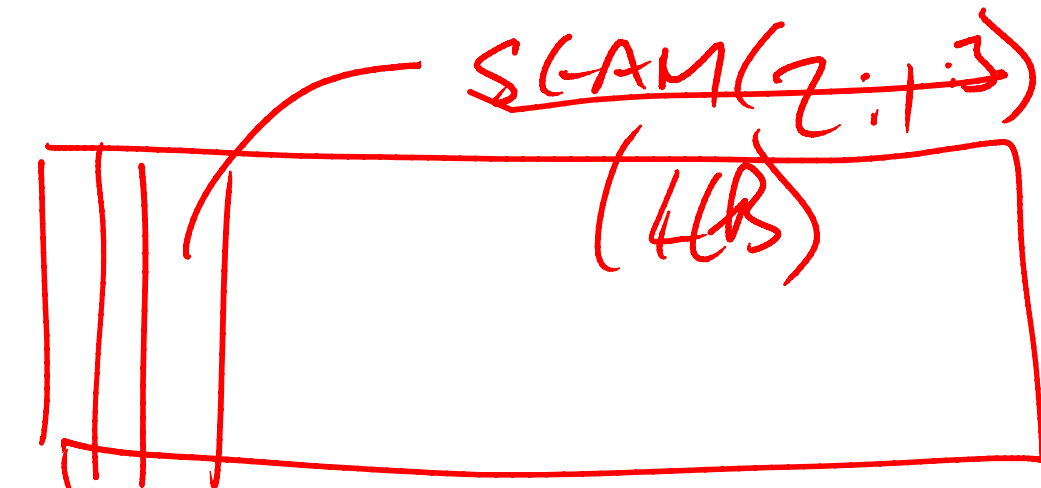
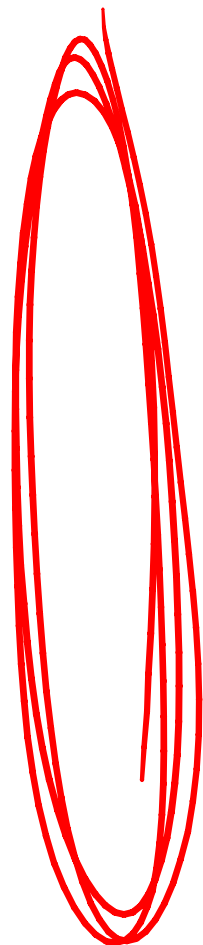
4GB



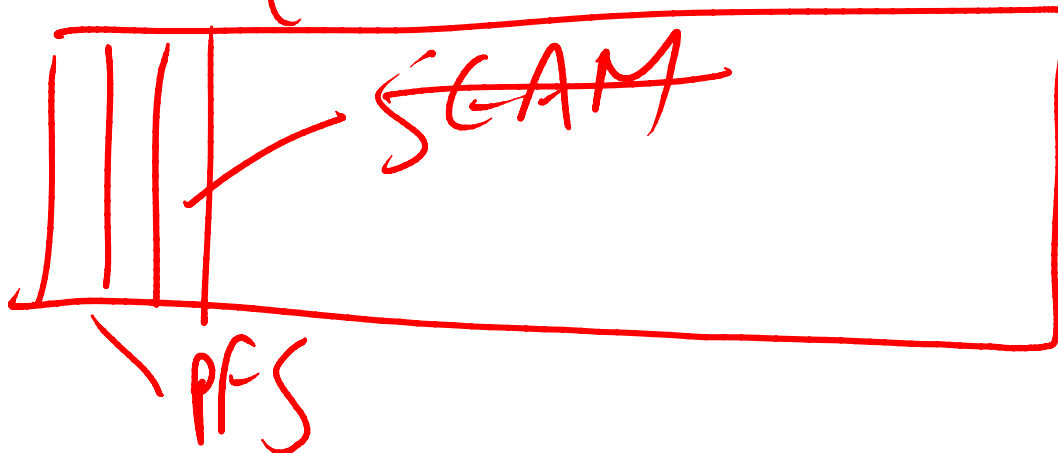
8GB



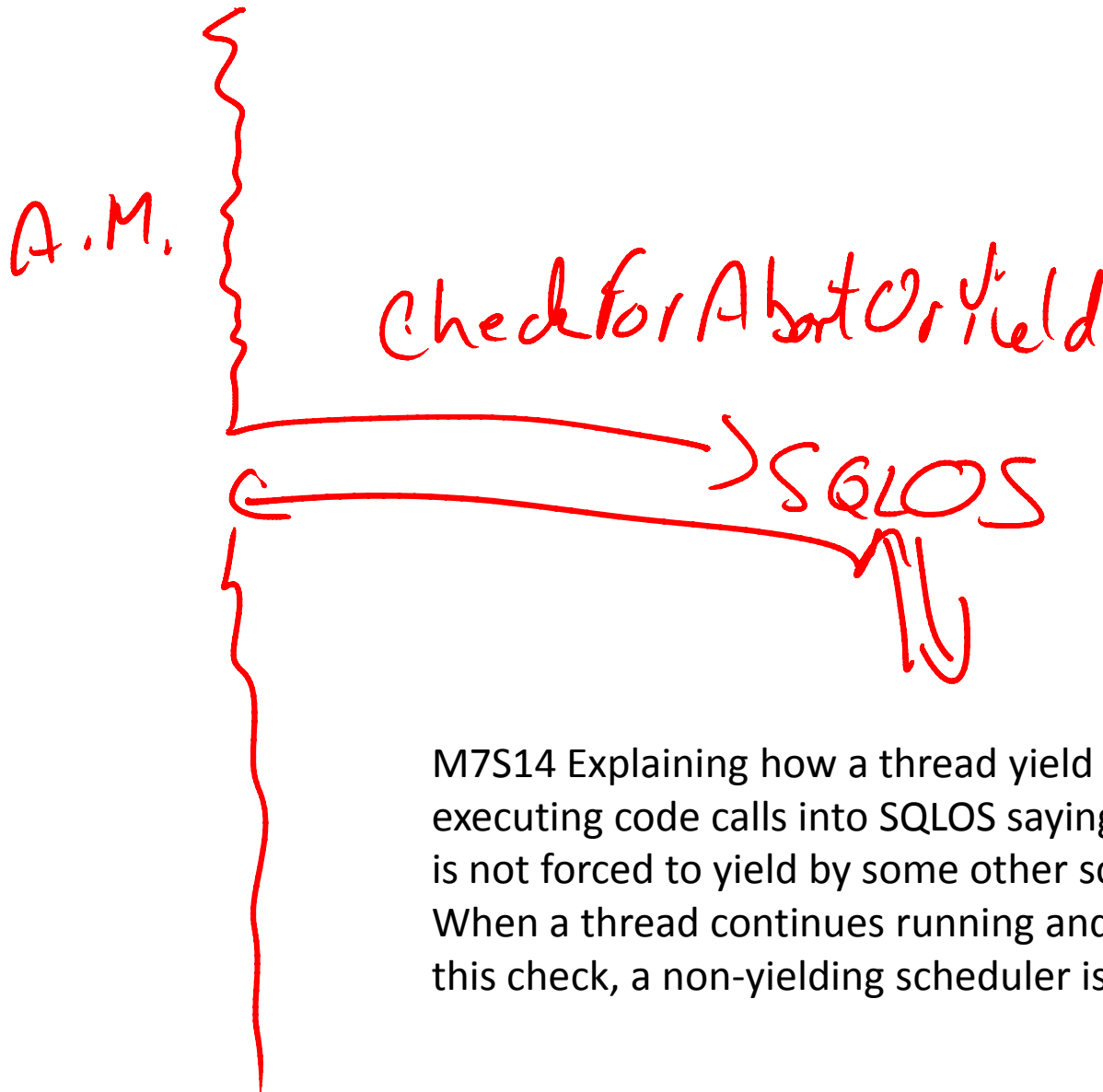
M1S36 Adding new files to tempdb. They all need to be the same size. If existing files are growing, add new files that are larger, and then resize the existing files to be the same as the new files. This is better than trying to get new files to be exactly the same size as the existing, changing files.



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



M1S39 Explaining PFS and SGAM contention in tempdb and how adding multiple files spreads the contention over multiple files, thus reducing it.



M7S14 Explaining how a thread yield works. The thread that's executing code calls into SQLLOS saying 'do I need to yield'? It is not forced to yield by some other scheduling mechanism. When a thread continues running and running without doing this check, a non-yielding scheduler is the result (a bug).

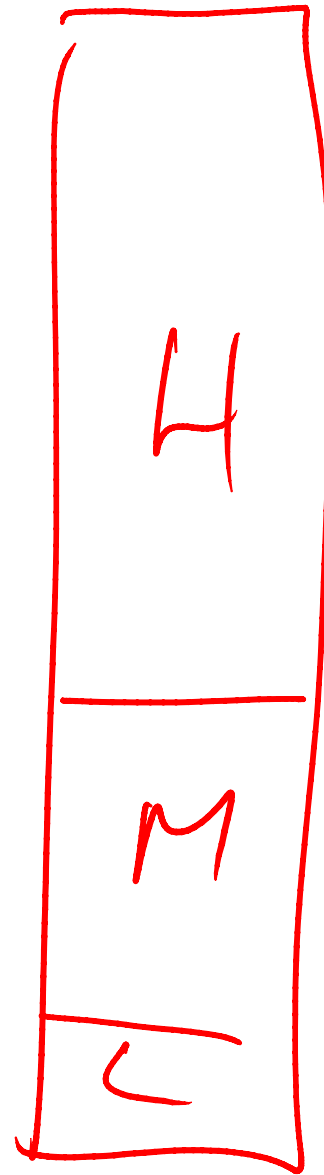
RESOURCE POOL

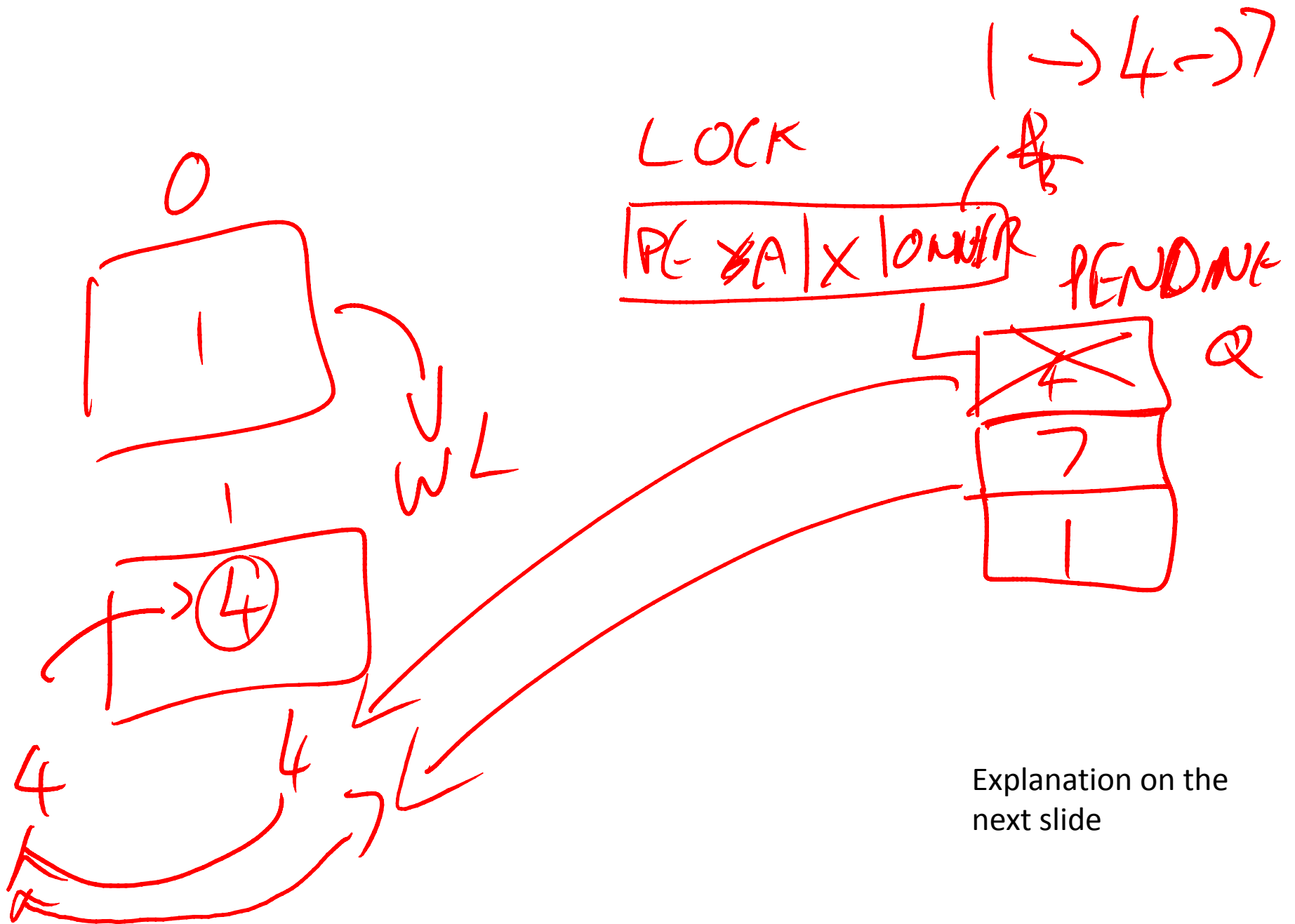
> / WORKLOAD GROUPS

H / M / L

9:3:1

M7S17 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.





Explanation on the next slide

M7 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.

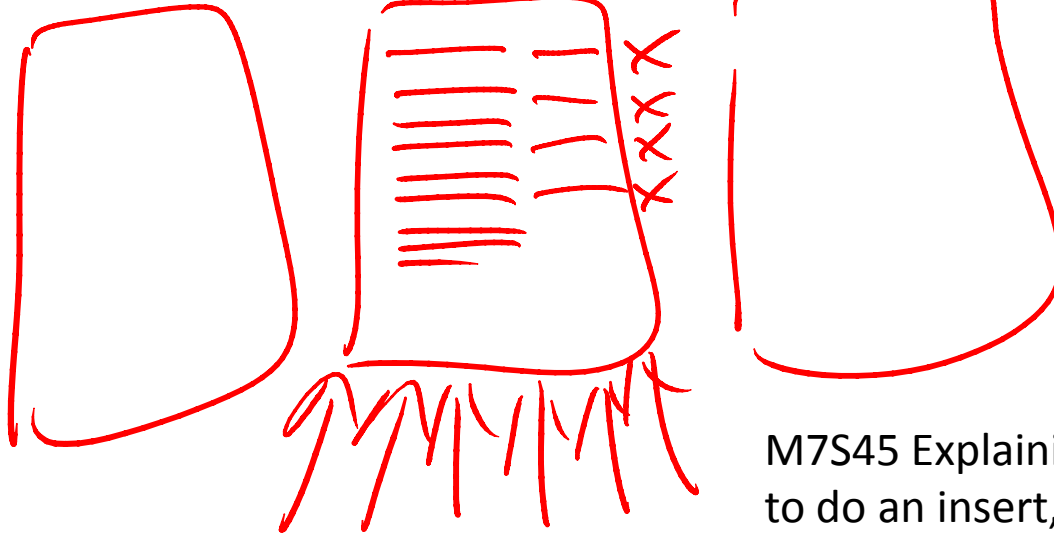
TIX

TIX = table IX lock

PIX = page IX lock

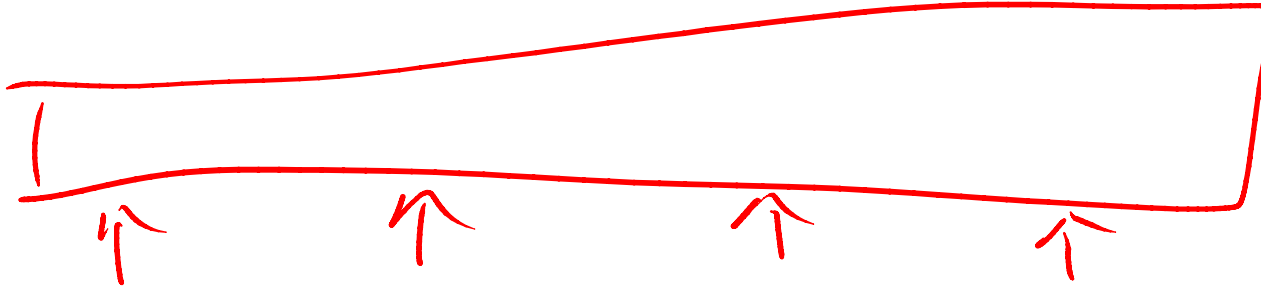
And there are row X locks

SOPIX



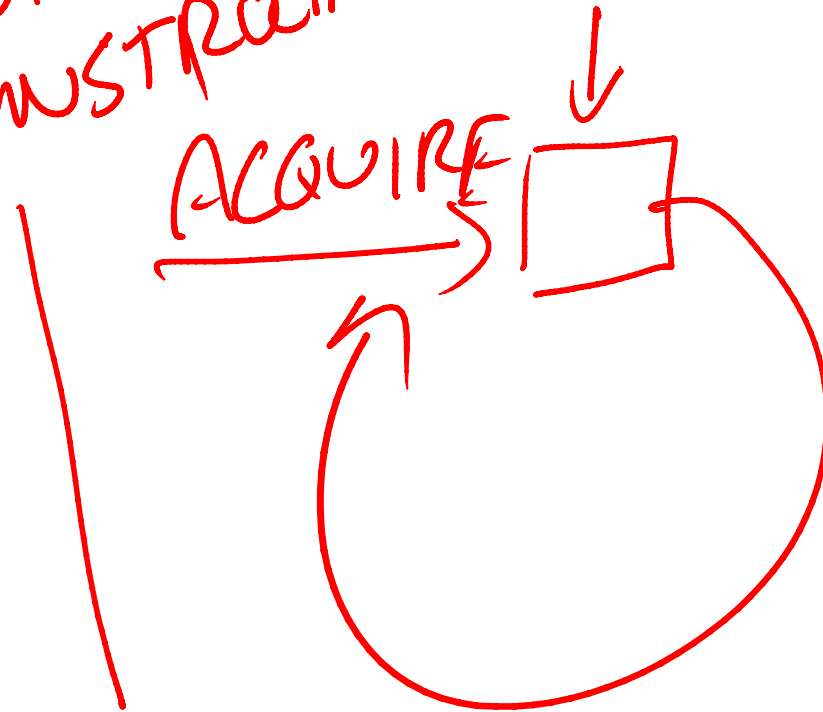
XXX
LATCH

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.

INTERLOCKED
INSTRUCTION SPINLOCK



SPINNING

1000

COLLISION

TST_SET_IF_CLEAR

BACKOFF

M7S37 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.

M1S39 Explaining how in 2008 and before, lock hash collisions could occur where two sets of keys hash to the same 4-byte value. Solve by upgrading or adding another table column.

KEY

