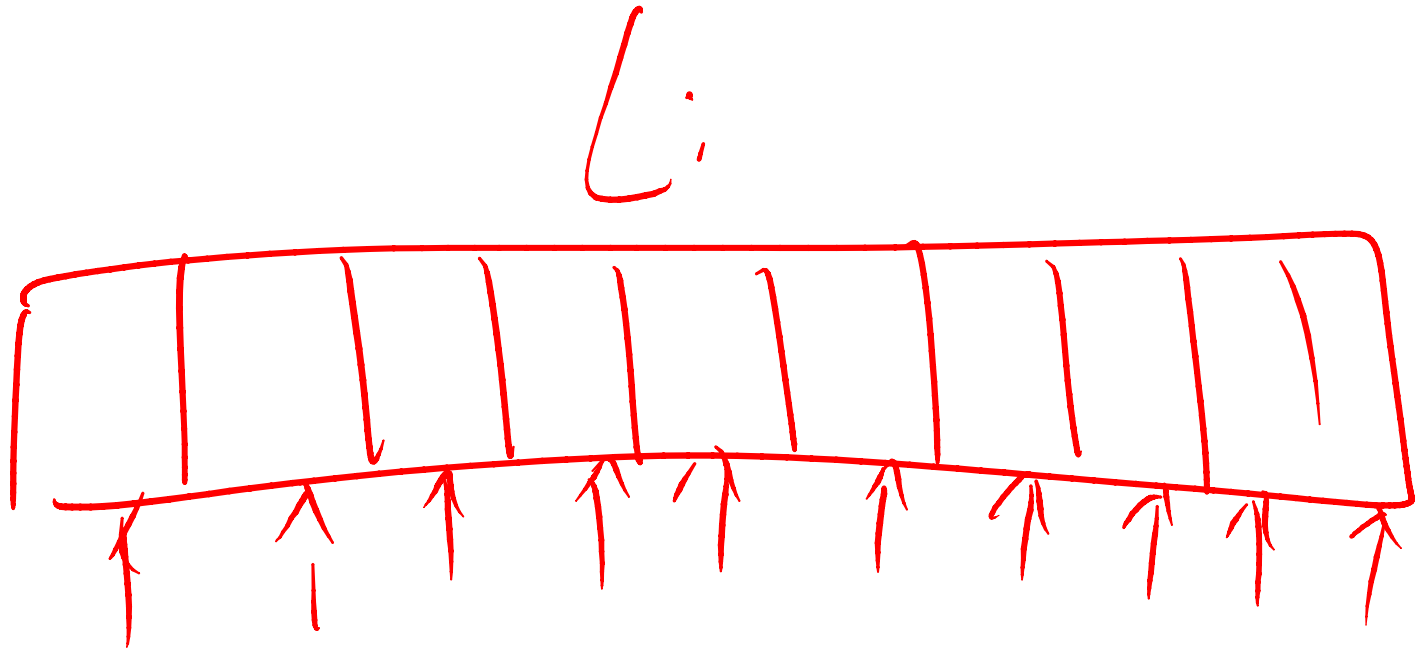


#sqlhelp

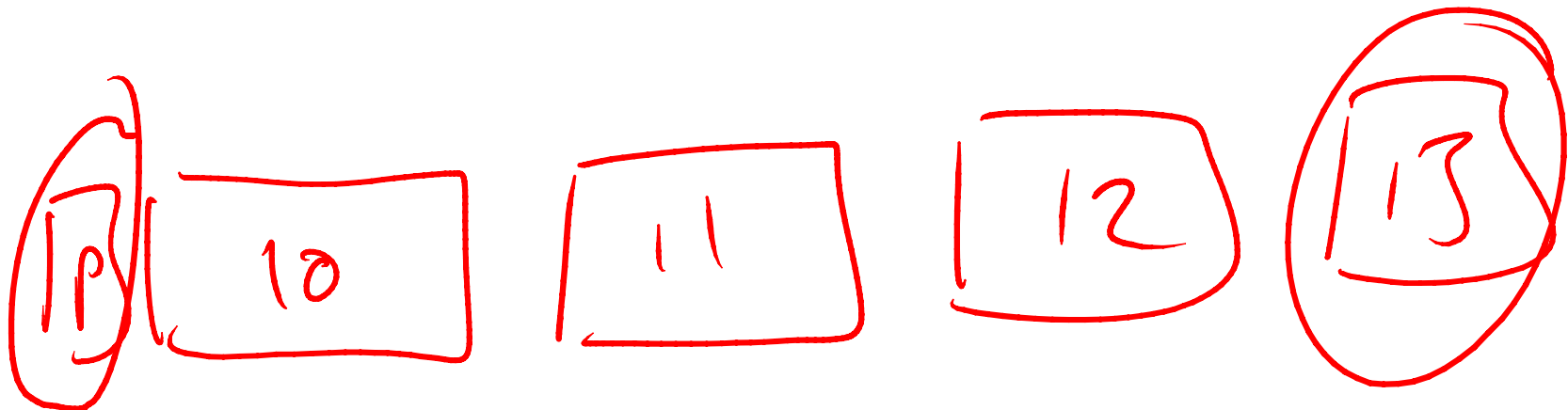
Your friend on twitter...

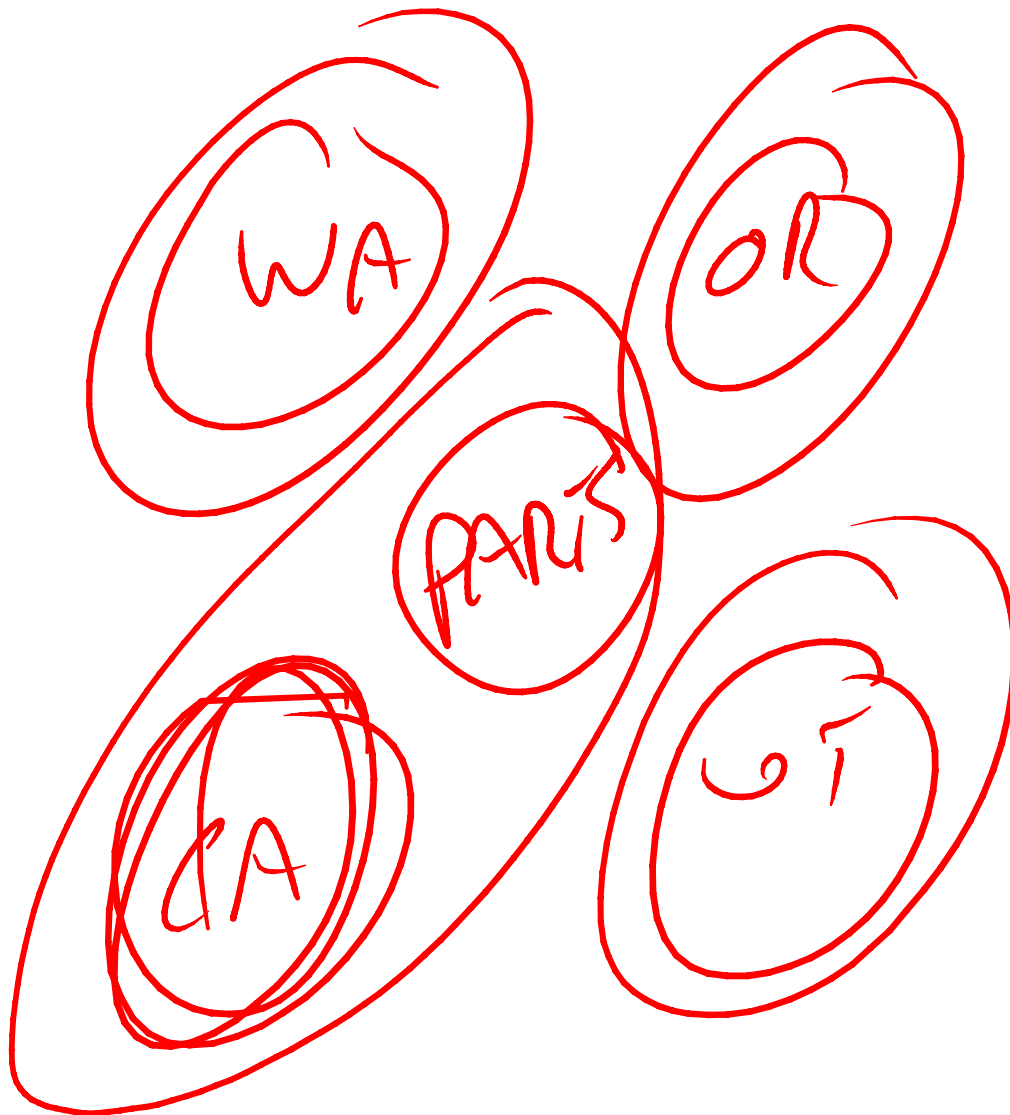
M1S7 Be careful about putting all the log files on a single volume. If they're all having activity, that creates multiple write points on the volume, making it a random I/O workload, even though each log is sequential-write only.





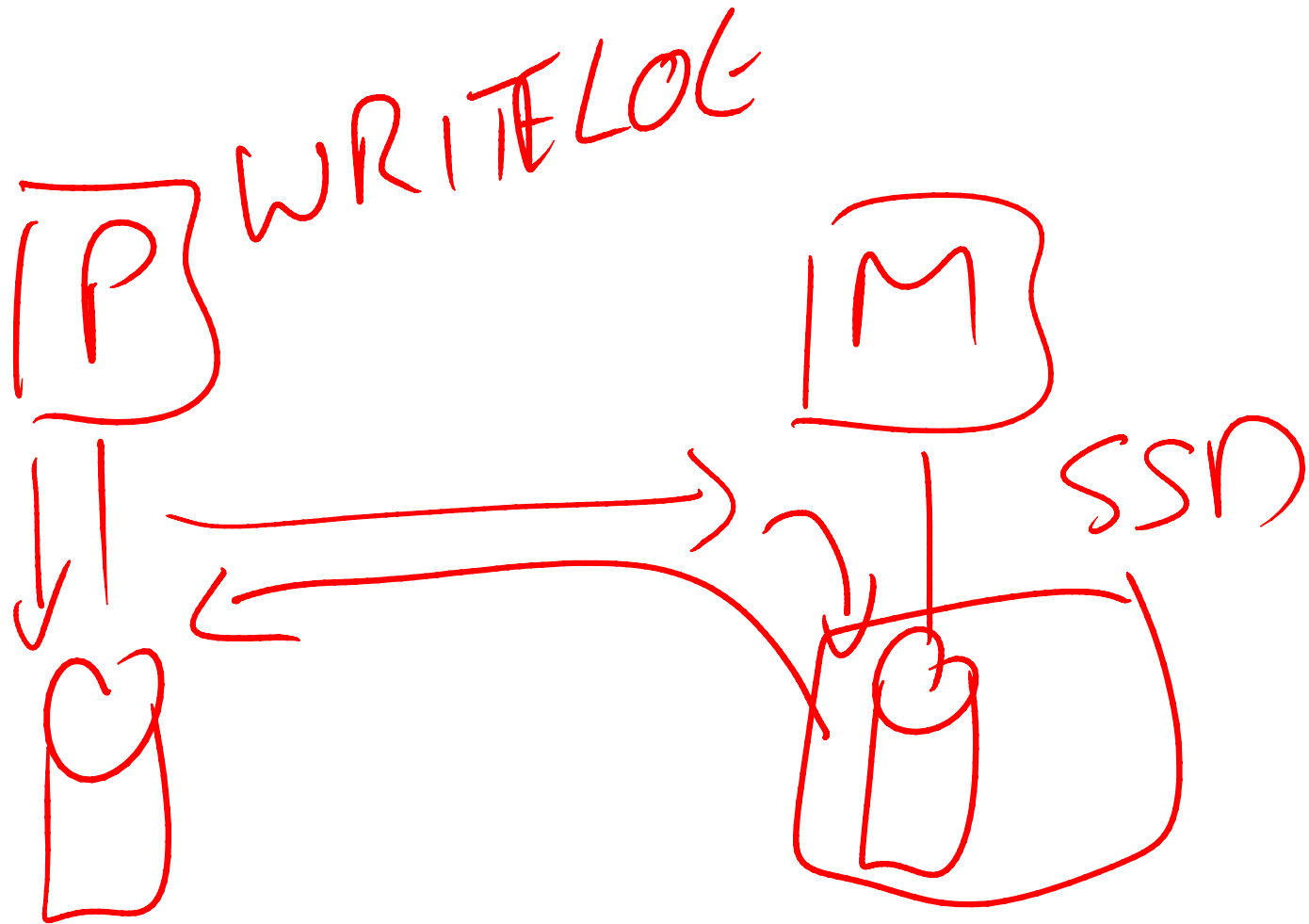
M1S7 Explaining that dividing a large database into filegroups allows targeted restores during disaster recovery, rather than having to wait for the entire single file database to restore.



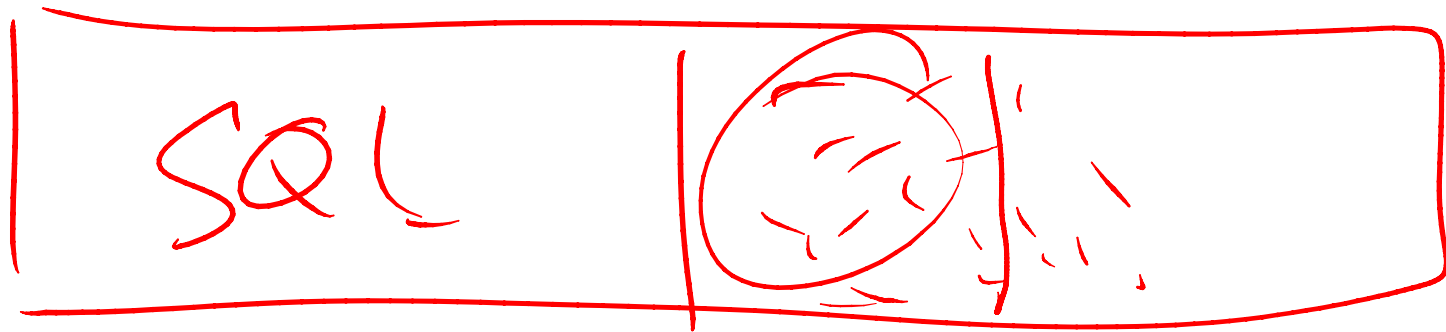


M1S7 Explaining a different partitioning scenario – auto parts sales to WA/CA/OR/UT.

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.



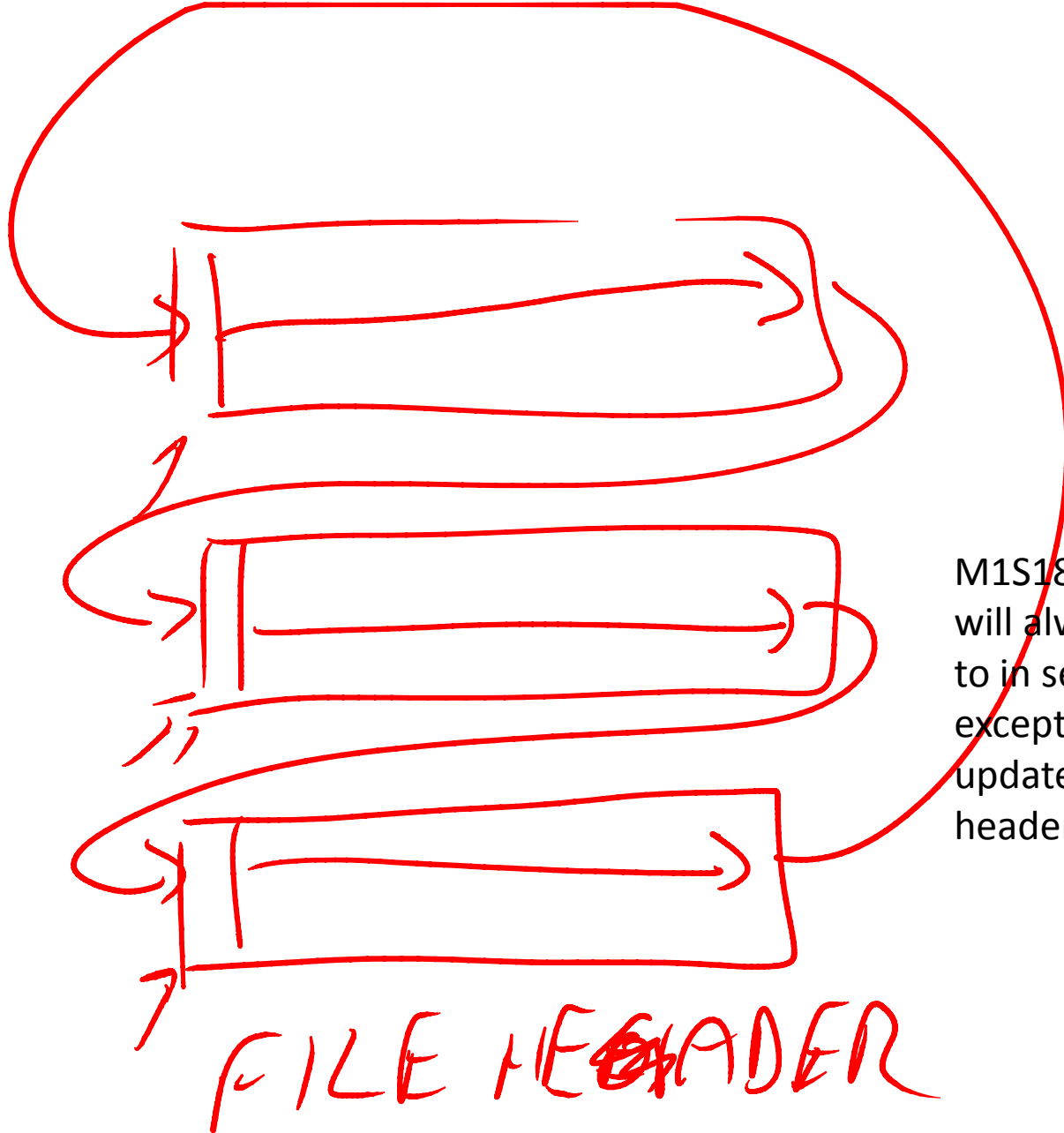
M1S13 A potential use of SSDs is to speed up the I/O subsystem on a mirror. If there is a perf issue with synchronous mirroring because of the time waiting for the hardening on the mirror log drive, stick it on an SSD.



SA

DBCC PAGE

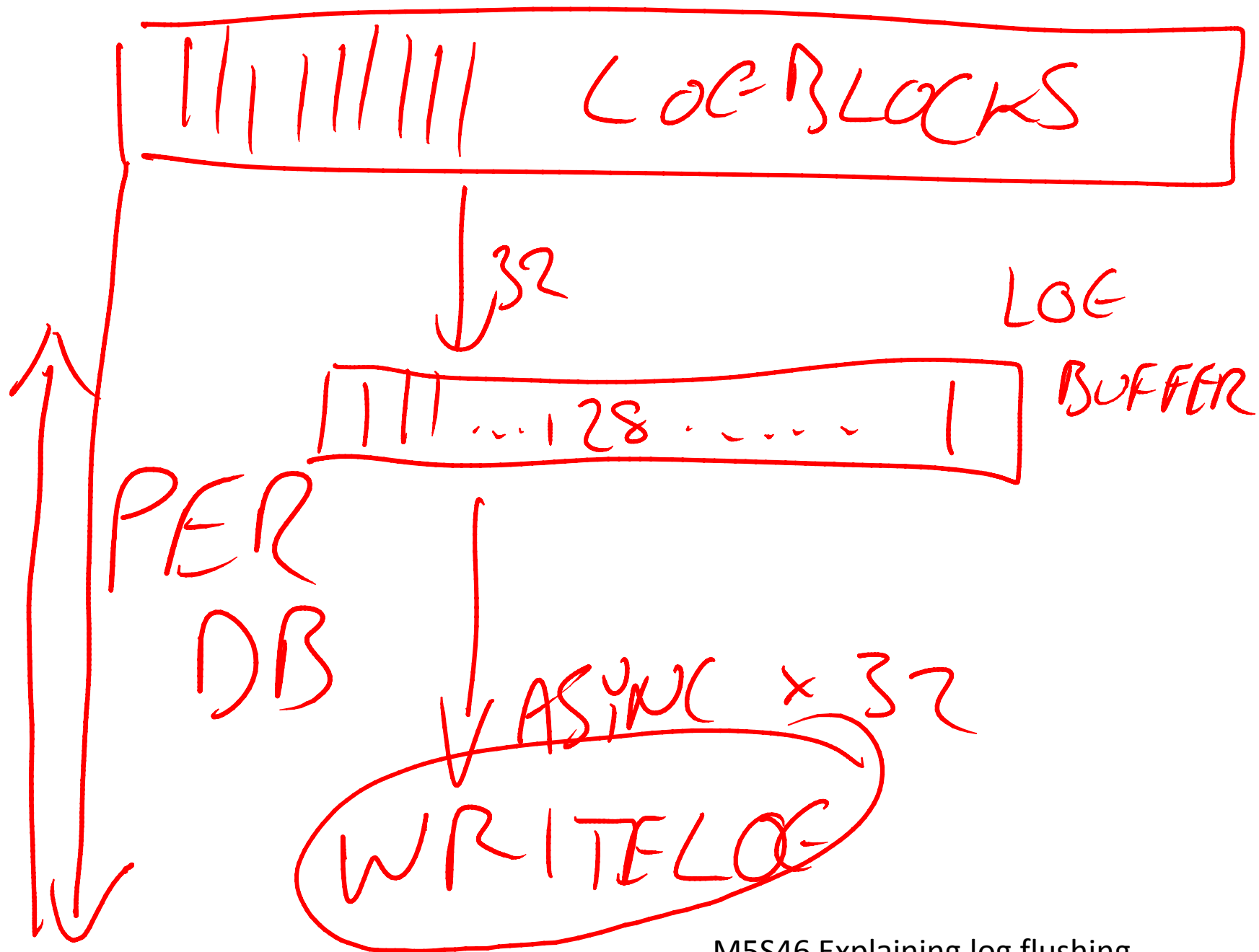
M1S14 Explaining why instant file initialization is off by default. Sharing a volume between SQL Server and 'secure files' can lead to information leakage if the secure files are deleted, then SQL grows a data file, and an SA can use DBCC PAGE to get a hex dump of the bits/bytes that comprised the deleted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

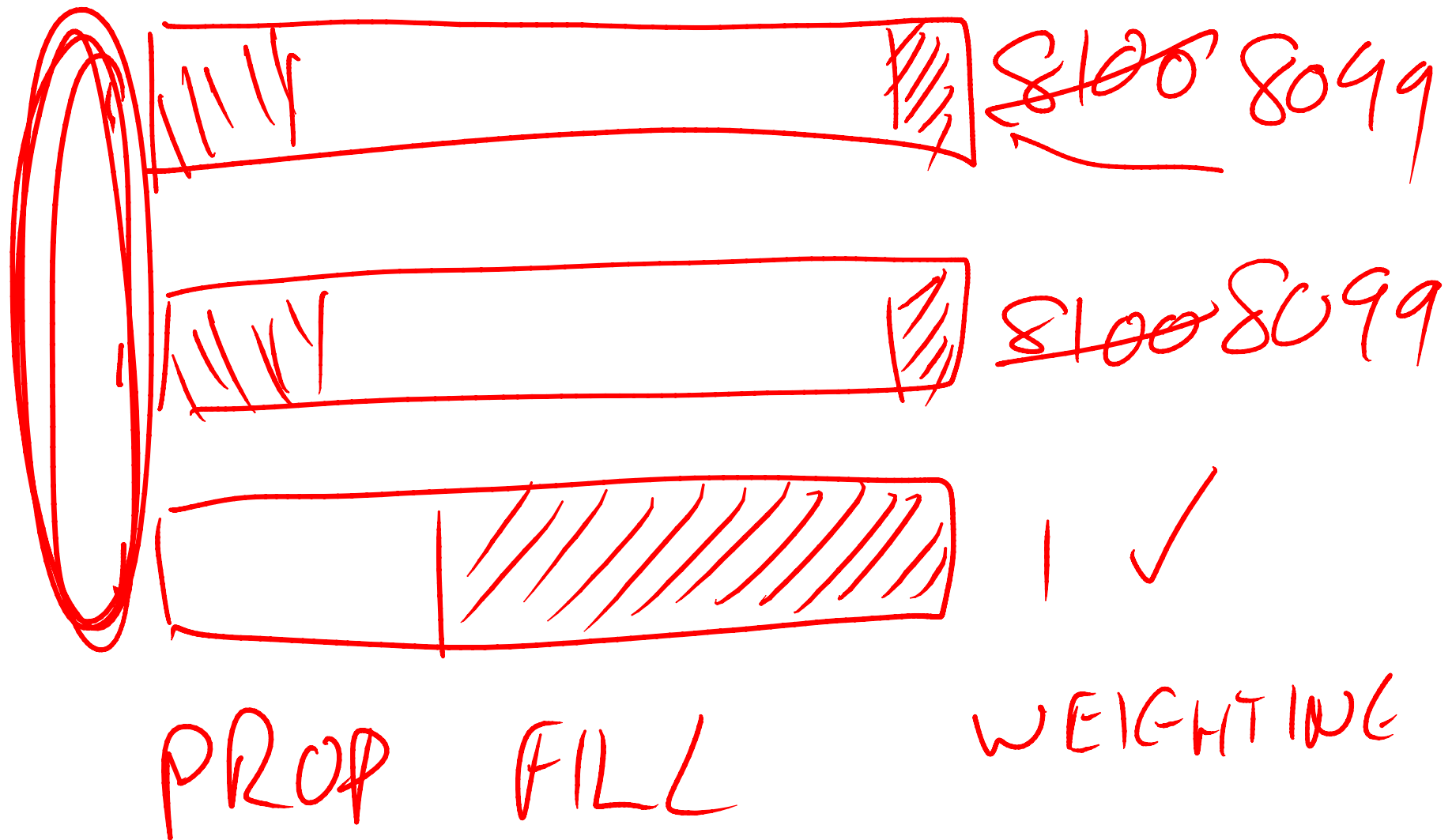


M1S18 Multiple log files will always be written to in sequential order, except for periodic updates to the file header pages.

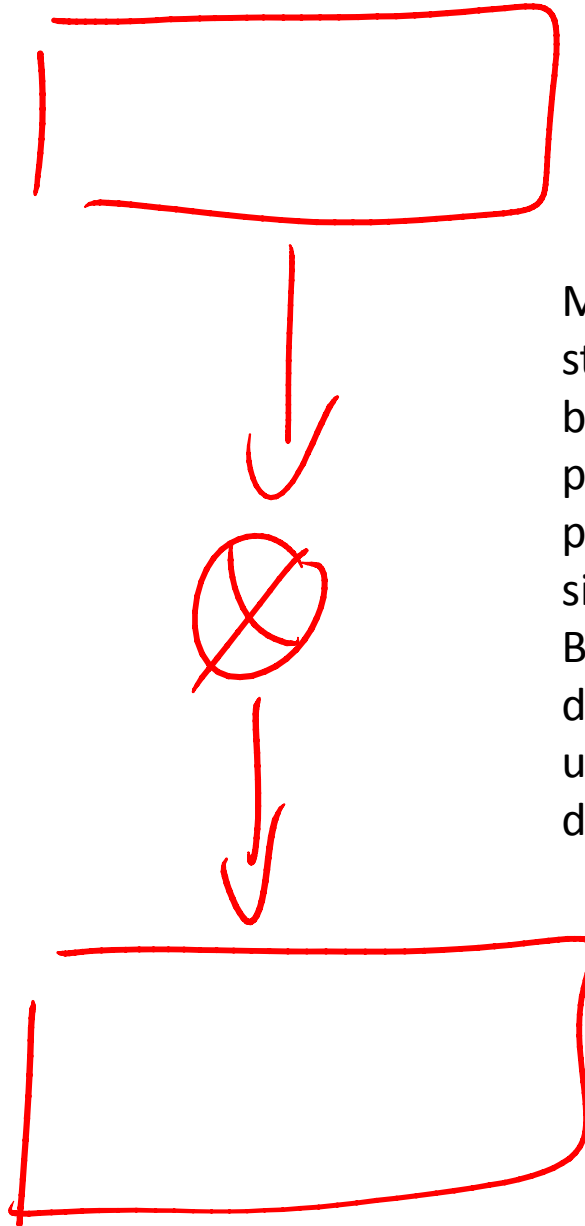


M1S18 VLF fragmentation is bad because of having to search through the VLFs for a particular VLF sequence number. See the [article link](#) for more info.





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

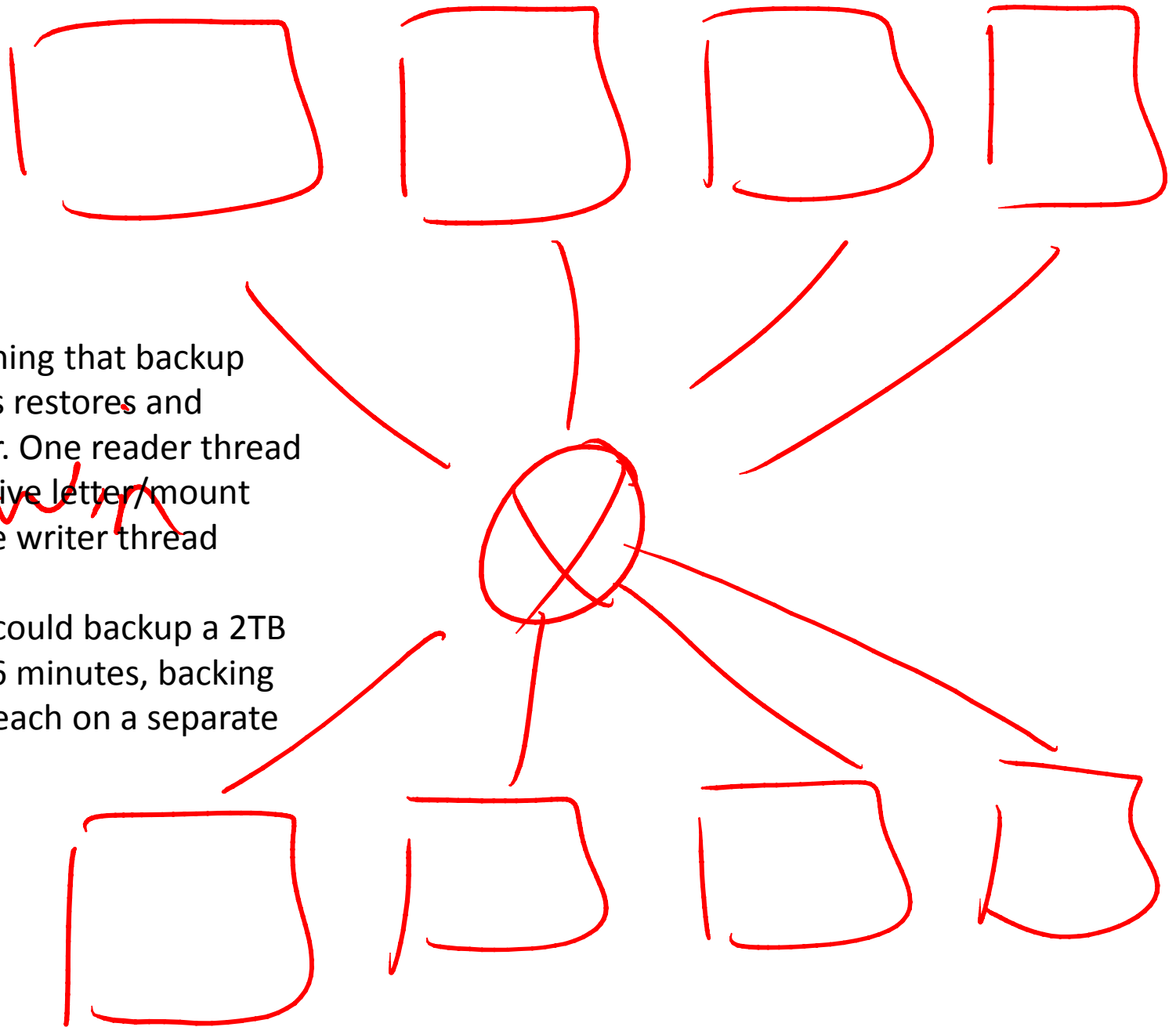


M1S33 Explaining that backup striping makes restores and backups faster. One reader thread per distinct drive letter/mount point, and one writer thread similarly.

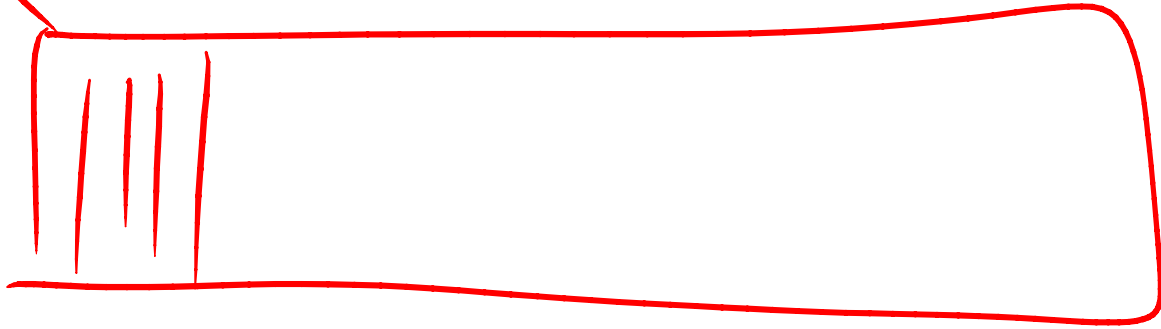
Bwin in 2005 could backup a 2TB database in 36 minutes, backing up to 12 files each on a separate drive.

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

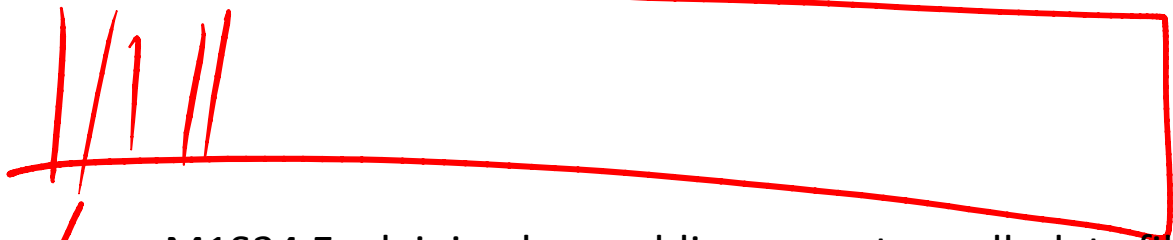
Bwin in 2005 could backup a 2TB database in 36 minutes, backing up to 12 files each on a separate drive.



PFS



TF 1118 - 328551

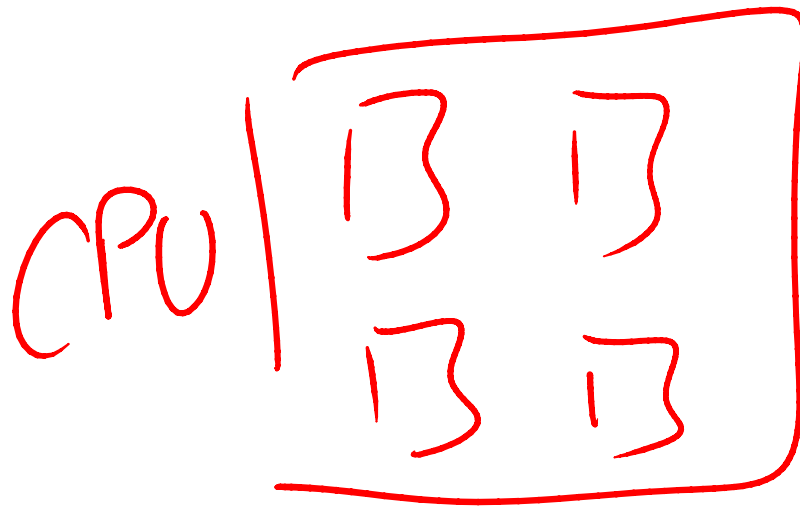


M1S34 Explaining how adding more tempdb data files leads to round-robin allocation and a spread of the contention to multiple files, leading to lower overall contention.

PFS



PFS

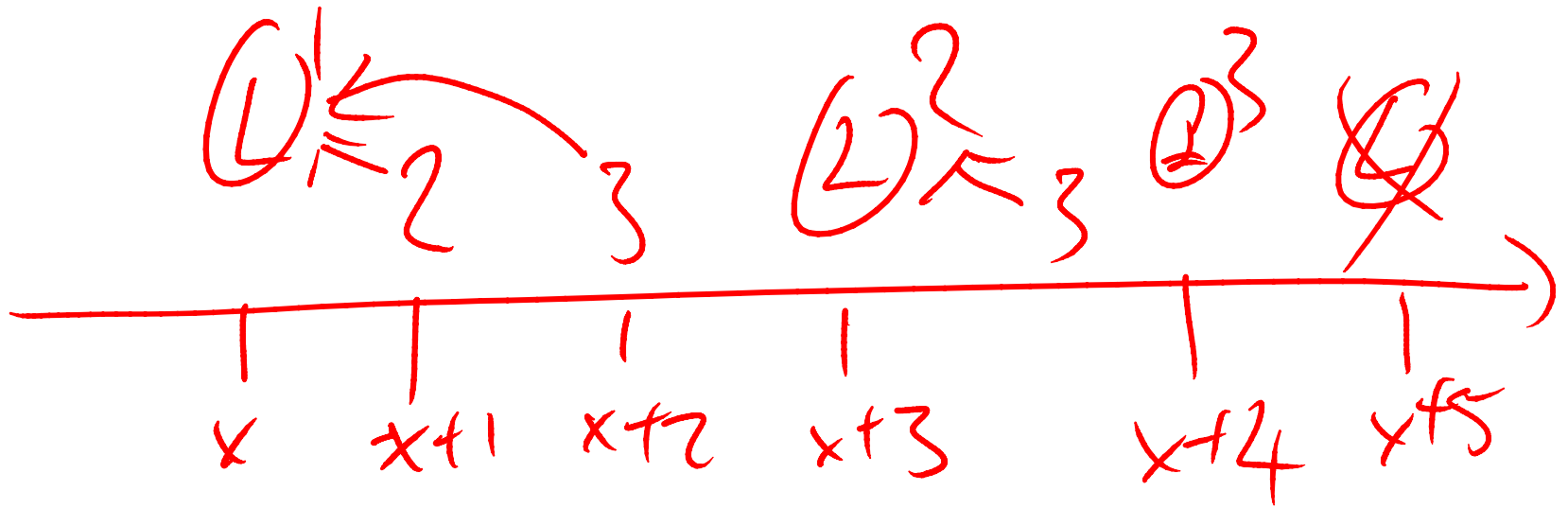


8 PHYSICAL
CORES



HT x 2
16 LOGICAL
CORES

Explaining how a CPU contains physical cores. A server with two CPUs, each with 4 cores, will have 8 physical cores. If hyperthreading is enabled, that becomes 16 logical cores.



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

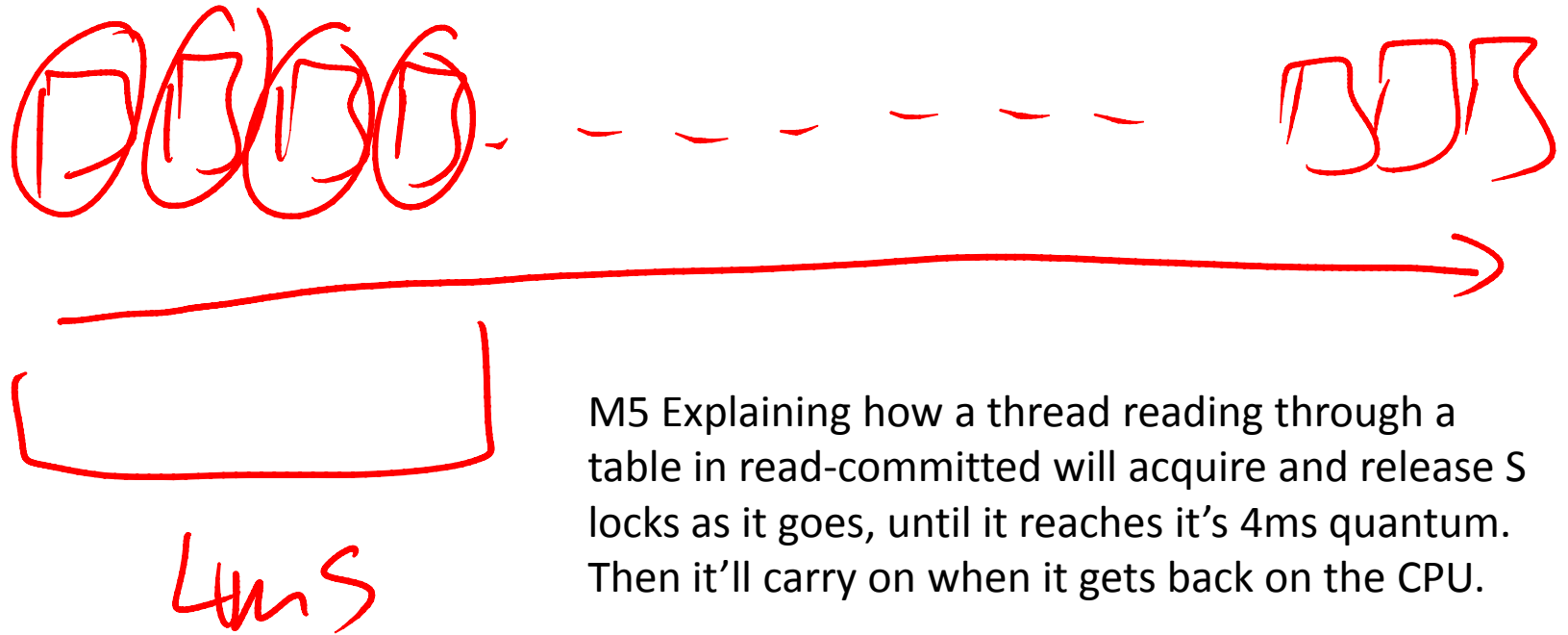
X+1: Thread 2 tries to acquire the lock and can't. It registers a callback routine for itself on the pending queue in the lock value block. Then it suspends itself, moves off the CPU and onto the waiter list.

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.



M5 Explaining how a thread reading through a table in read-committed will acquire and release S locks as it goes, until it reaches it's 4ms quantum. Then it'll carry on when it gets back on the CPU.

Additionally, there was a discussion whether table locking equaled serialization isolation level, or exhibits the same behavior. It absolutely does NOT – they're quite orthogonal.

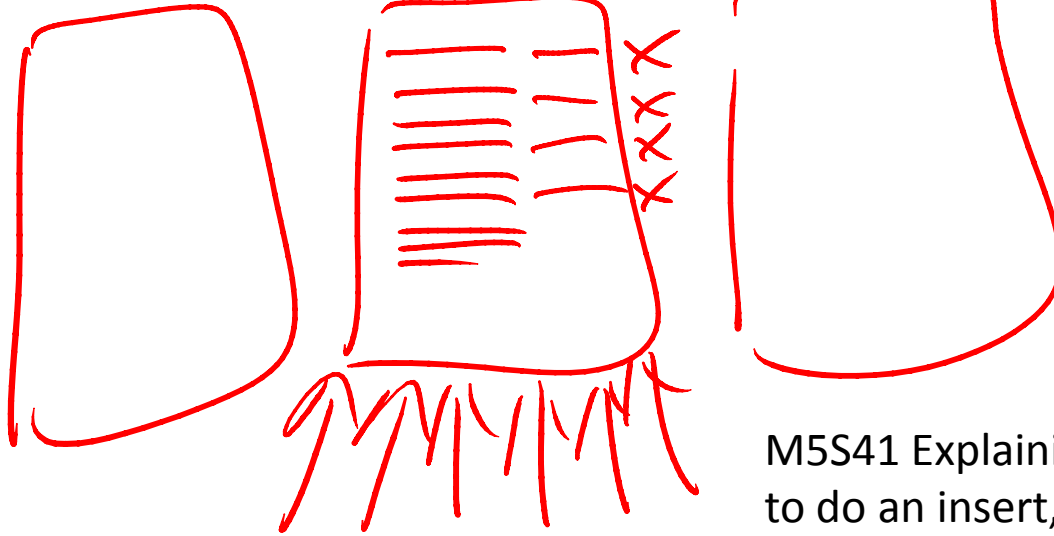
TIX

TIX = table IX lock

PIX = page IX lock

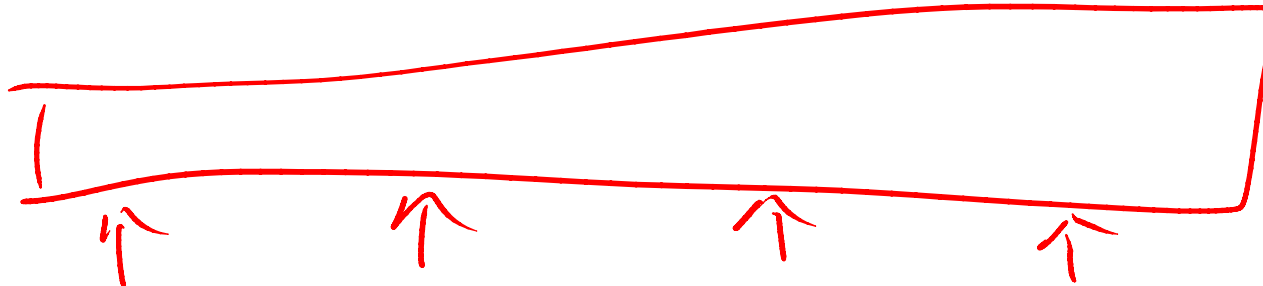
And there are row X locks

SOPIX



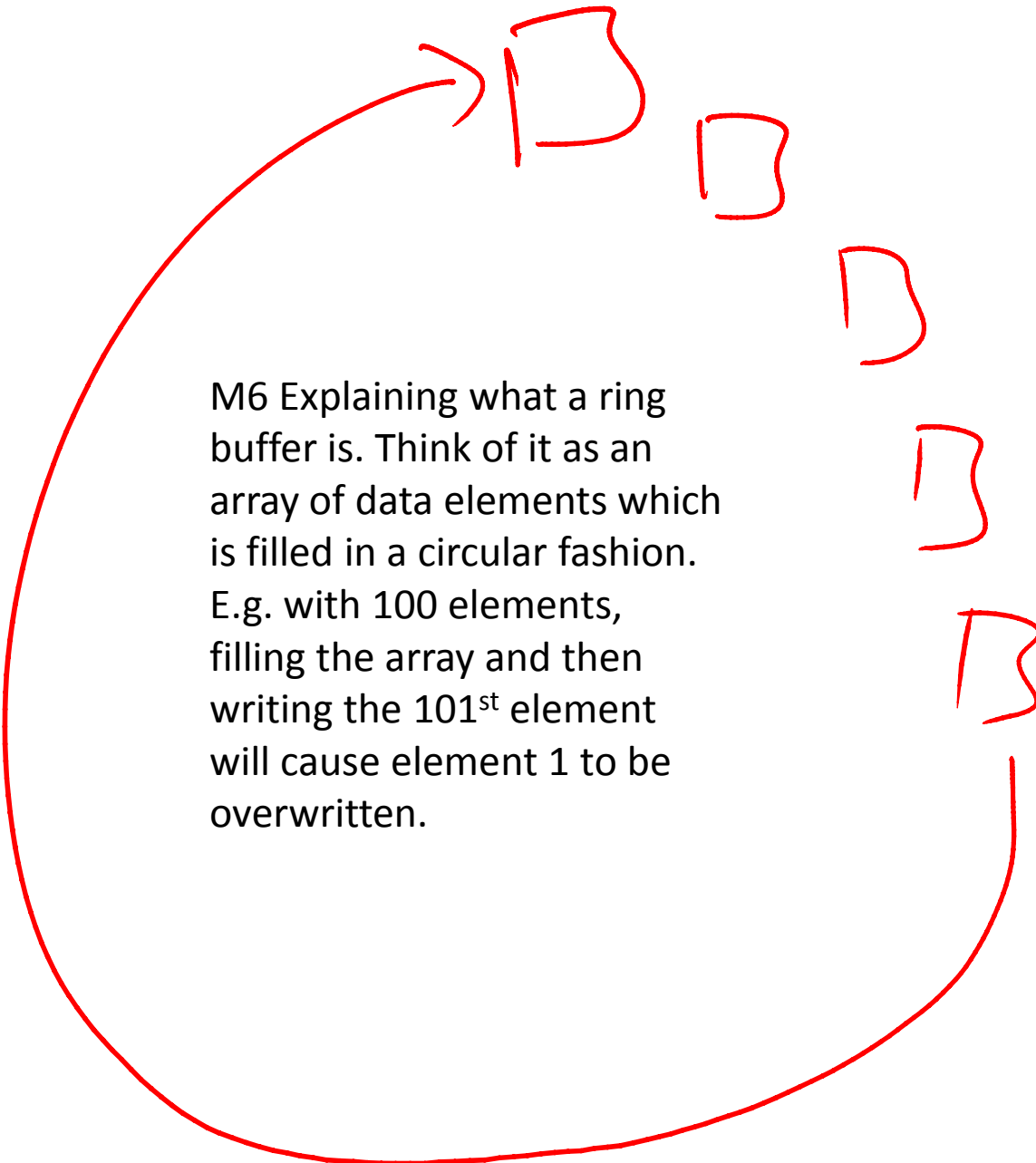
X X X
LATCH

M5S41 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.

100



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 100 elements, filling the array and then writing the 101st element will cause element 1 to be overwritten.