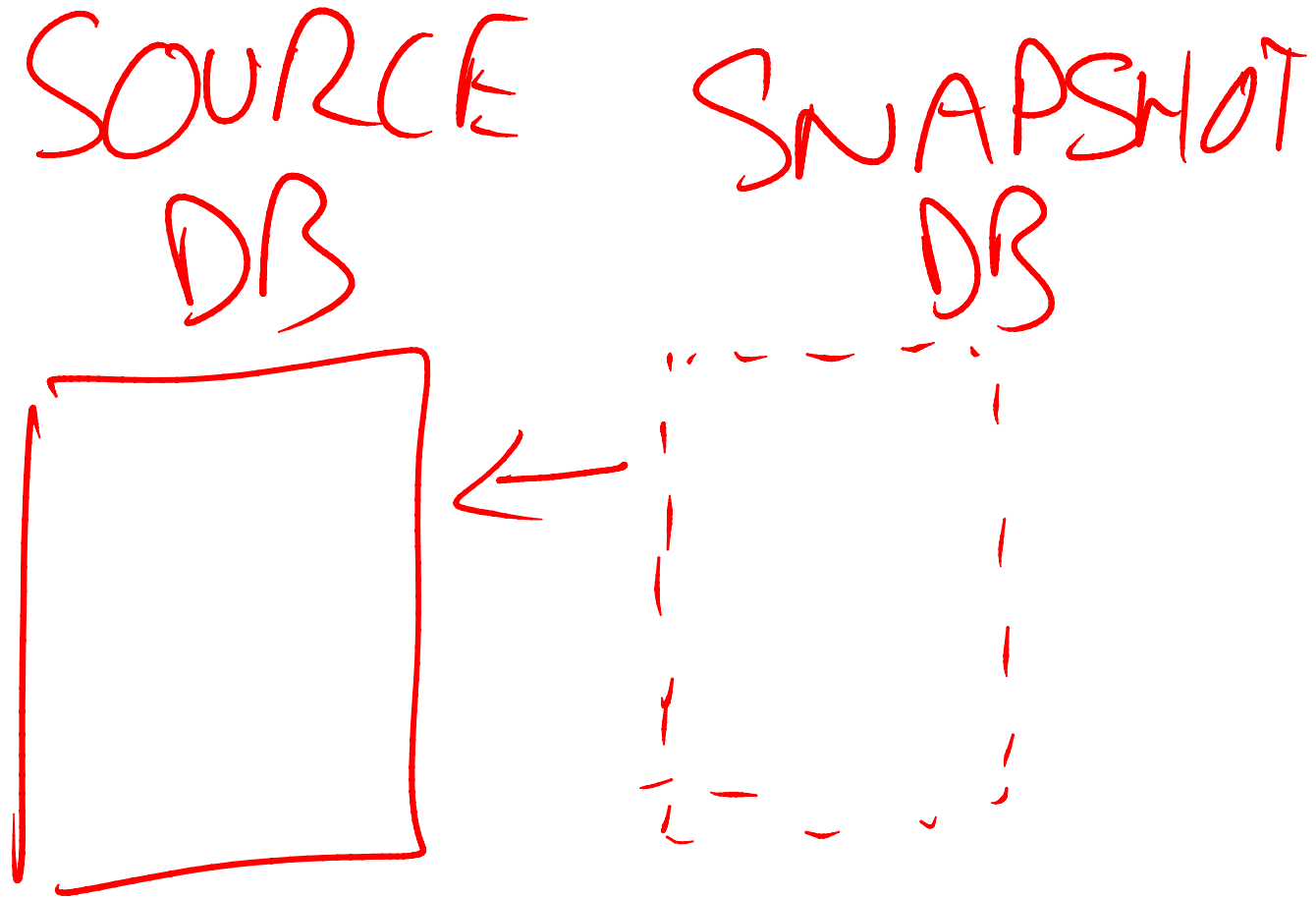
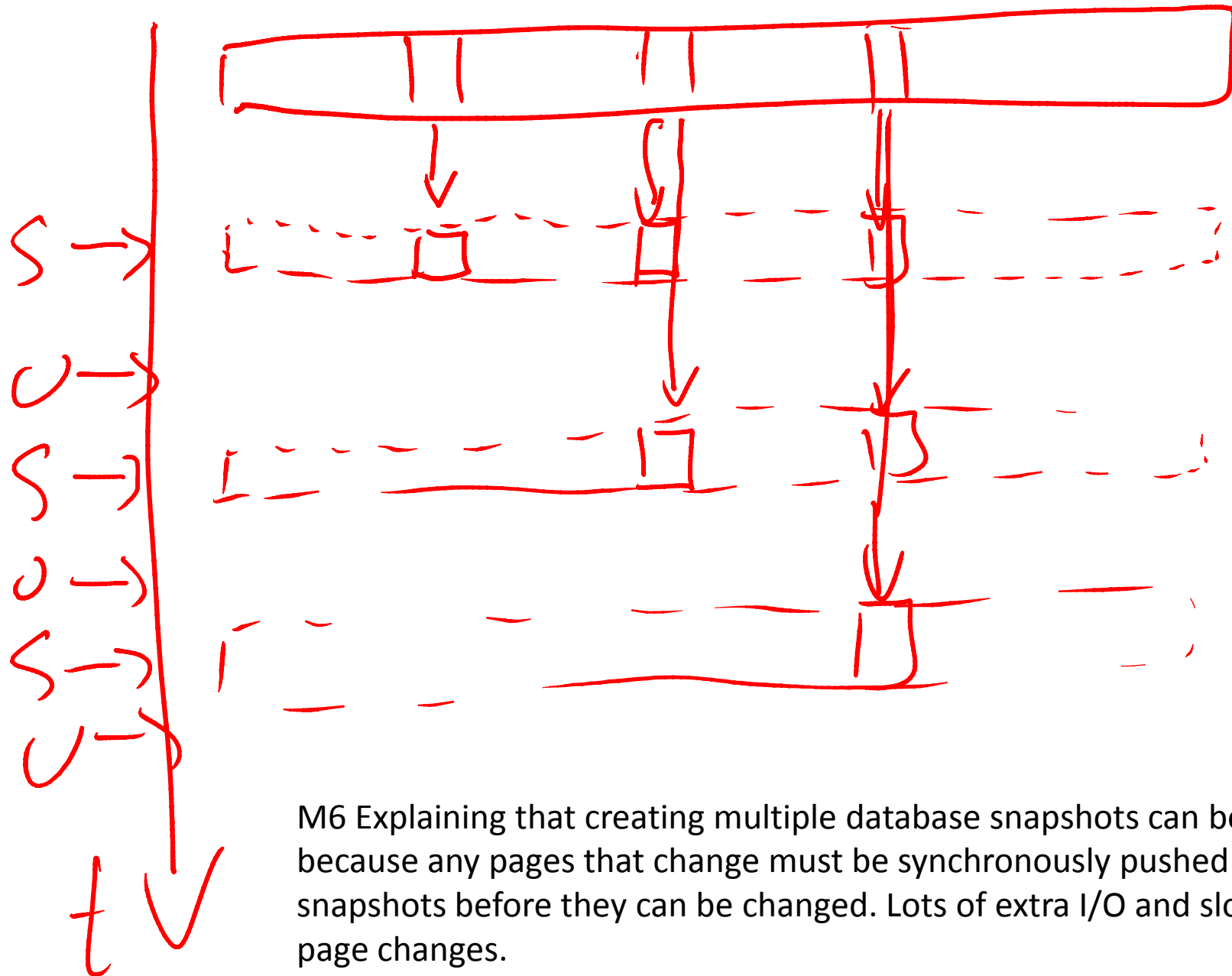




M6S3 Discussing how a database snapshot is useless without the underlying source database being present too.



M6 Explaining that creating multiple database snapshots can be bad because any pages that change must be synchronously pushed into all snapshots before they can be changed. Lots of extra I/O and slows down page changes.

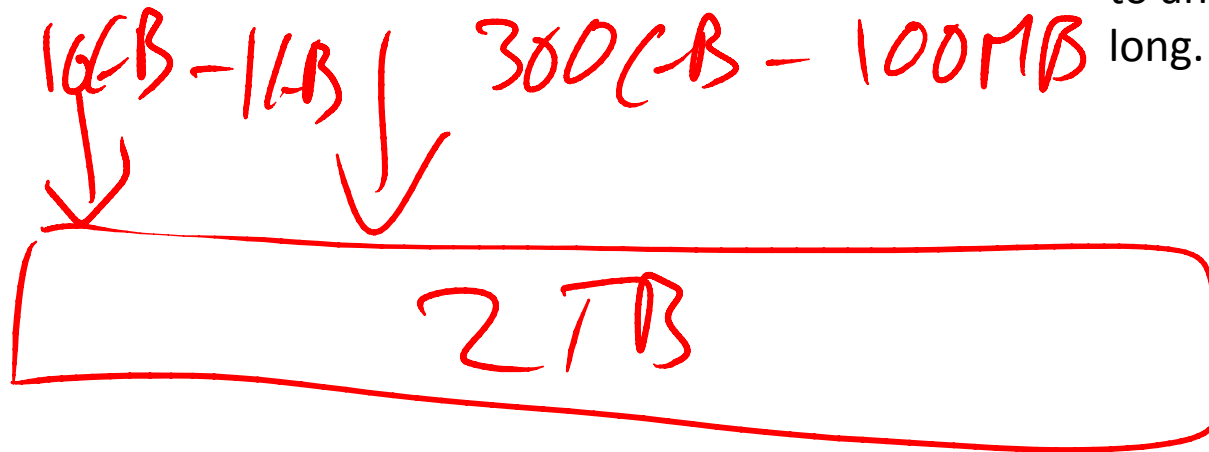
'int c[1000000] sparse';

c[913464] = 10

M6 Drawing an analogy between sparse files in NTFS and sparse arrays in various programming languages. You can define an array of arbitrary size but only the populated elements take up memory. Any elements you ask for that haven't been populated result in an error.

M6 Explaining that the NTFS sparse file keeps track of which file offsets have been written to and for how long.

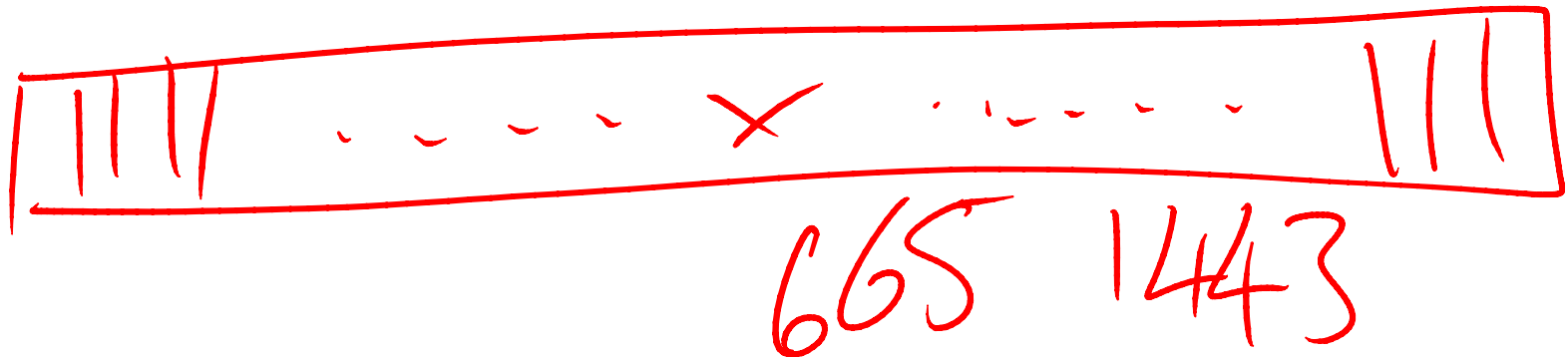
SPARSE FILE



300GB 10GB
(64KB) (100MB) [1GB]

M6 Explaining how there are a limited number of NTFS clusters in a volume (no matter the cluster size – but larger cluster size = larger volume). A volume where the free space is heavily fragmented will mean fragmented clusters making up the sparse file, resulting in error 665 and the sparse file becoming unavailable. NTFS limitation.

CLUSTERS
ALLOCATION ON ITS = 4K



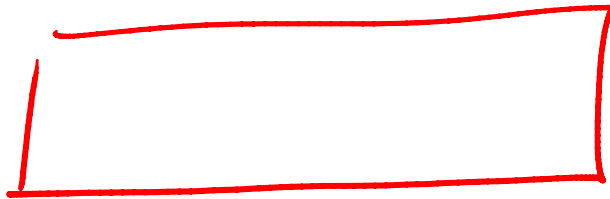
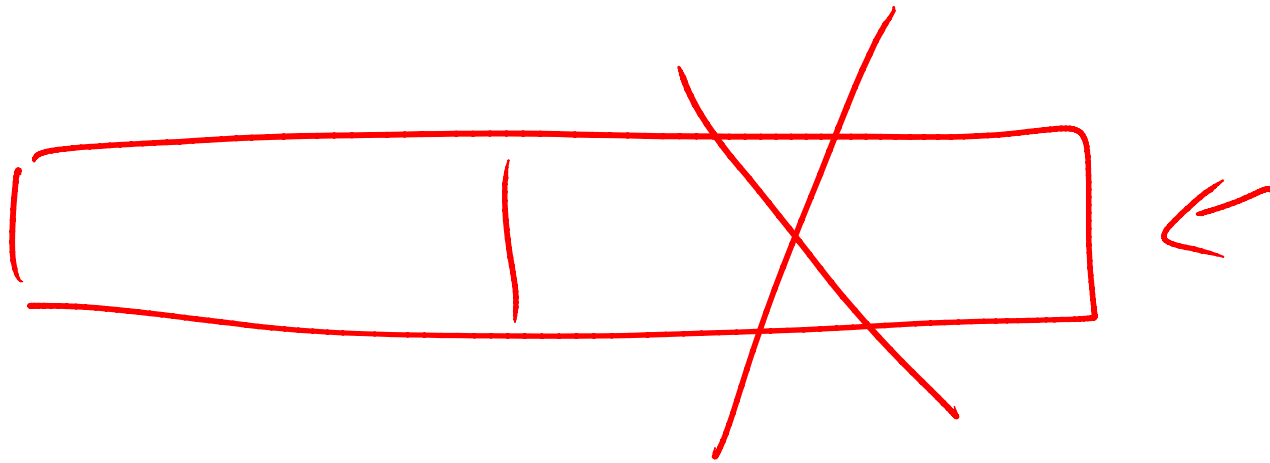
0 < 64MB 4 VLFs

64 < 1GB 8 "

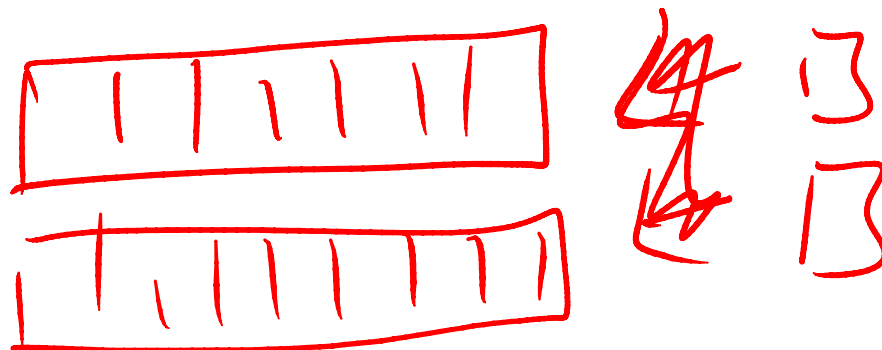
1GB > 16 "

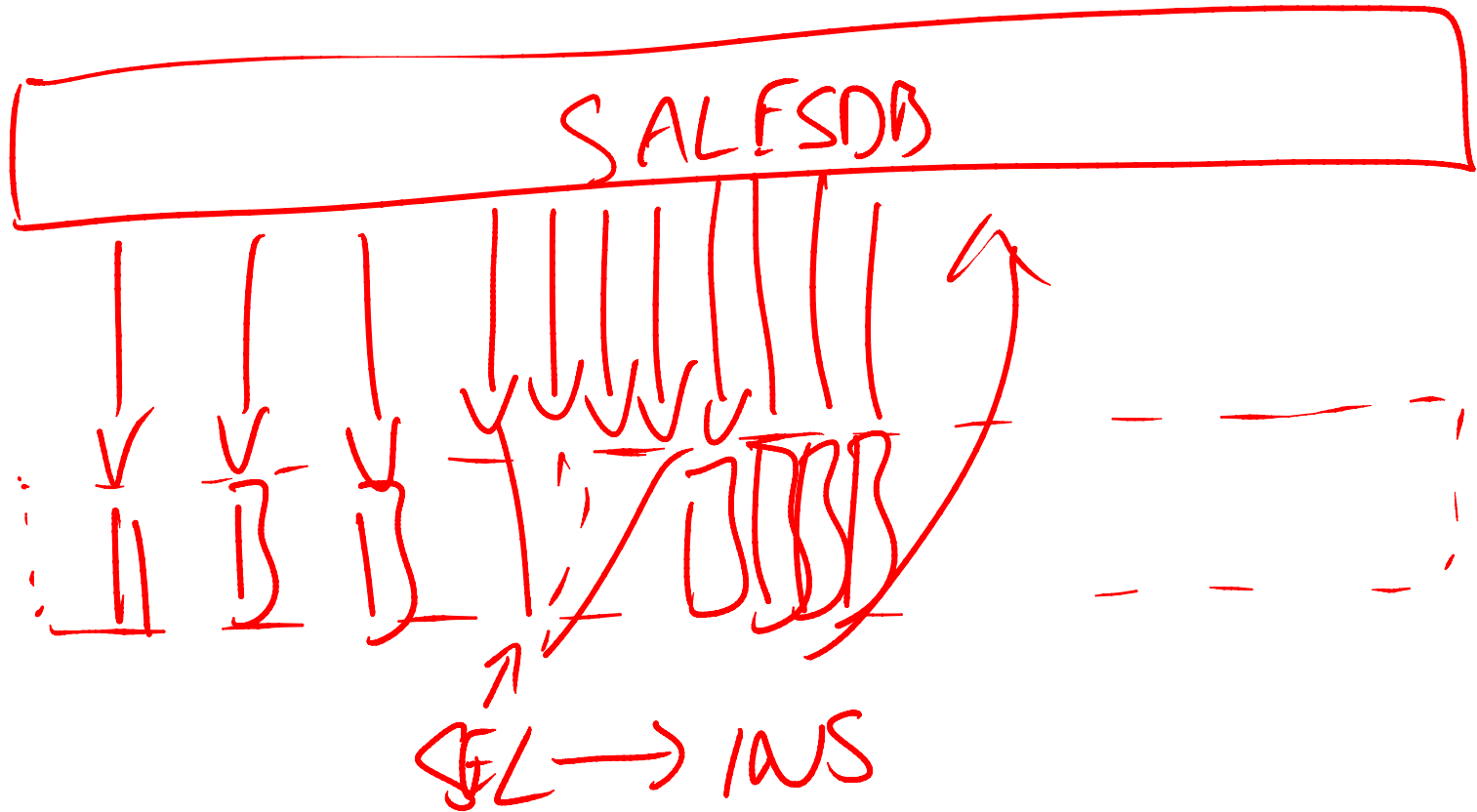
M6 The formula for how many VLFs you get depending on the initial size, growth size, or autogrowth size. See

<http://www.sqlskills.com/blogs/kimberly/transaction-log-vlfs-too-many-or-too-few/>



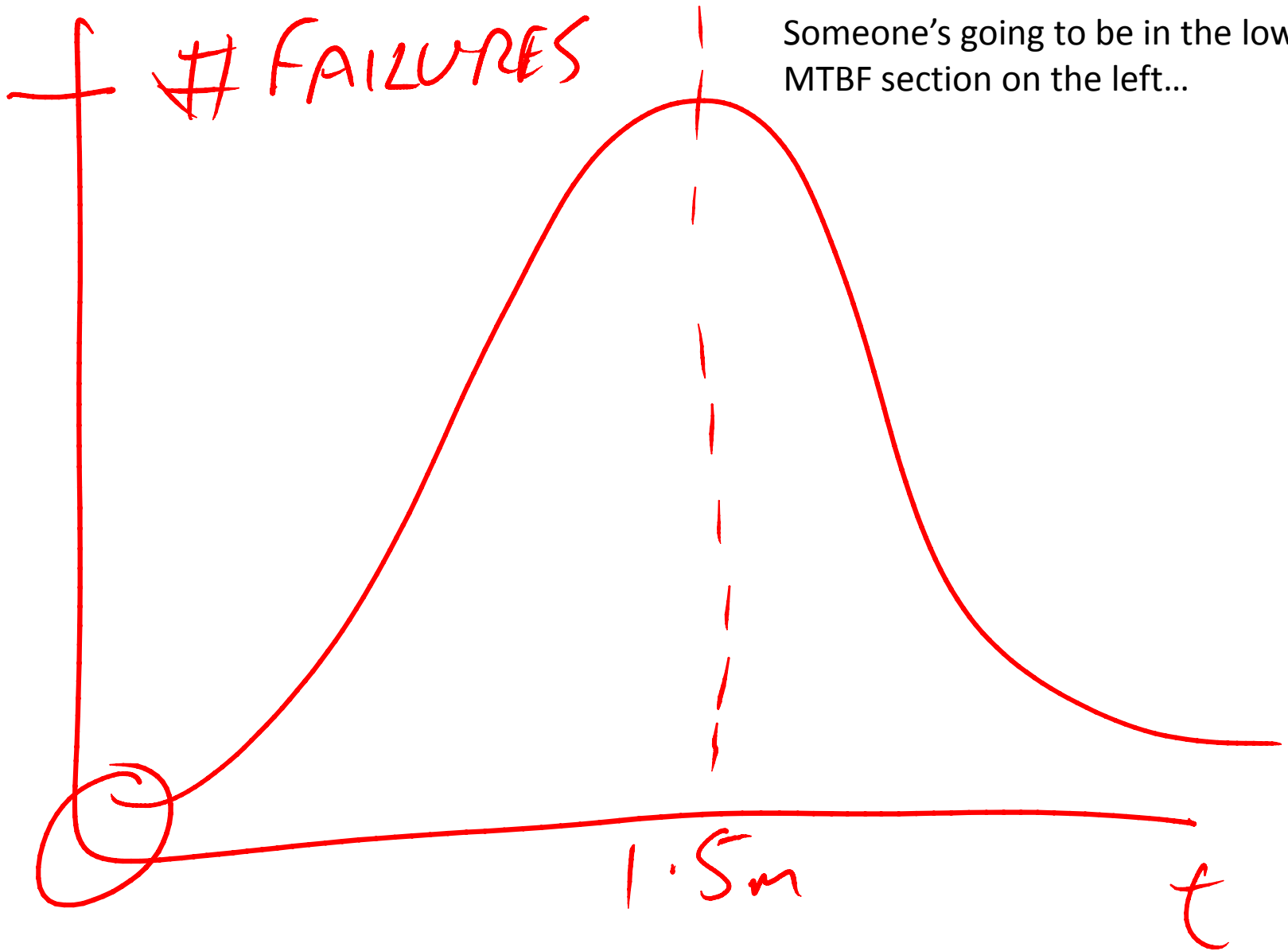
M6S16 Discussing log file shrinking. File ID 2 cannot be removed, only shrunk to min of 2 VLFs. With 2 log files, each can be shrink to one VLF, then the second log file cannot be removed.

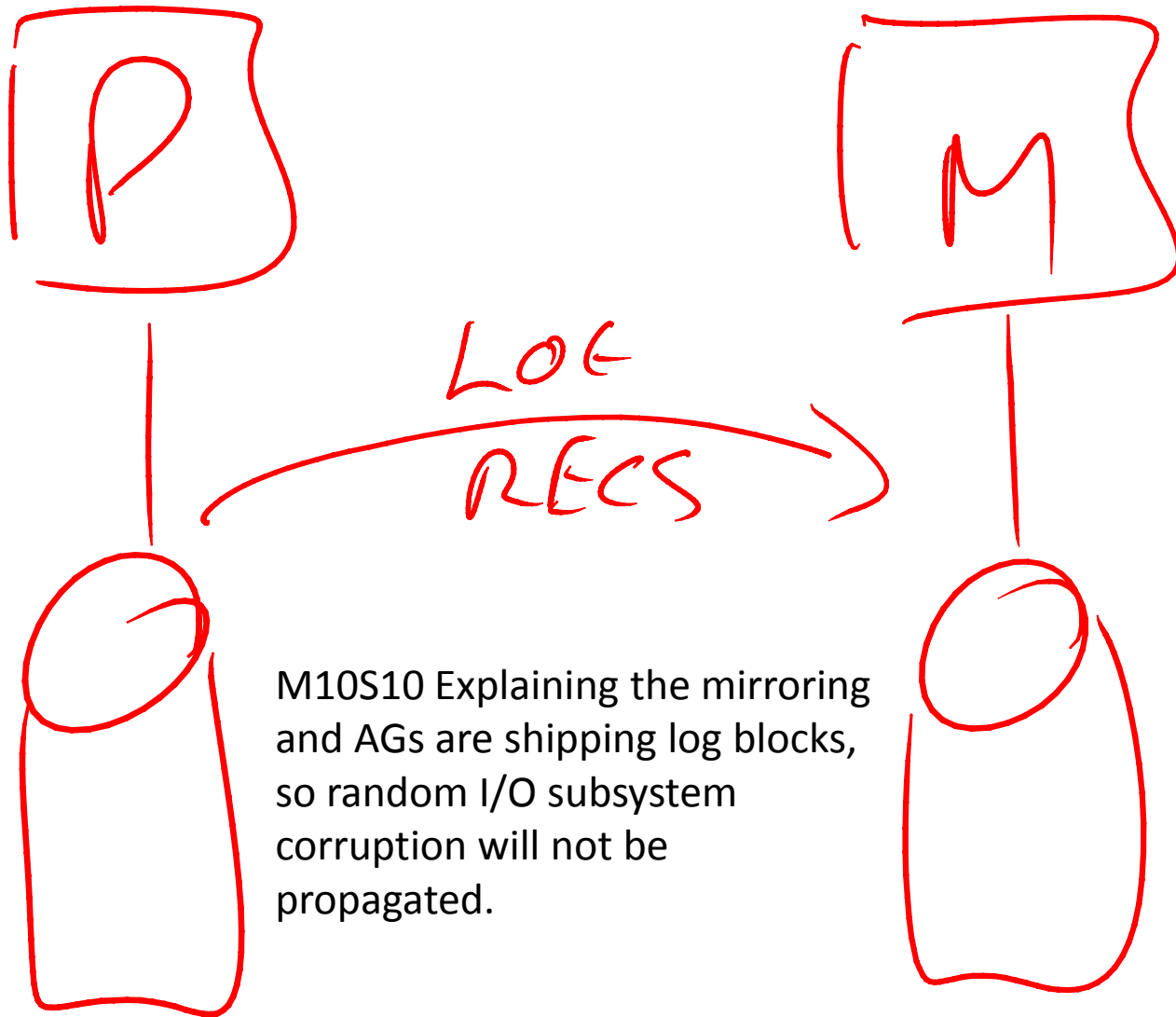




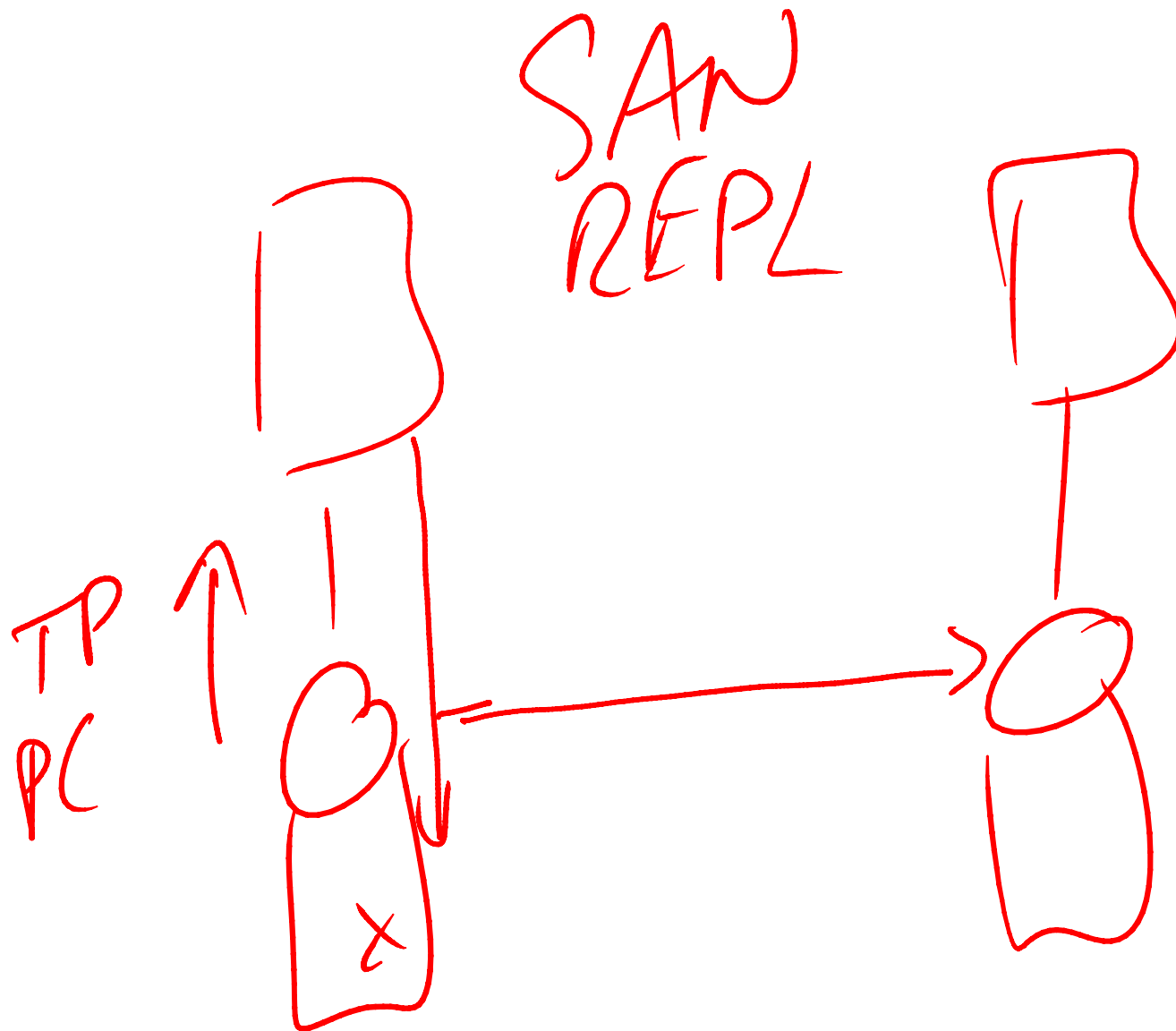
M6 Explaining the demo on recovering data from a snapshot. After dropping the table, the snapshot doesn't grow because the pages from the dropped table haven't been changed yet. As we repopulate the newly-created table though, it's pulling unchanged pages from the source database through the context of the snapshot to populate the new table, thus changing pages and pushing them into the snapshot. A bit of a mind-bender.

M10S8 The MTBF curve of drives.
Someone's going to be in the low
MTBF section on the left...



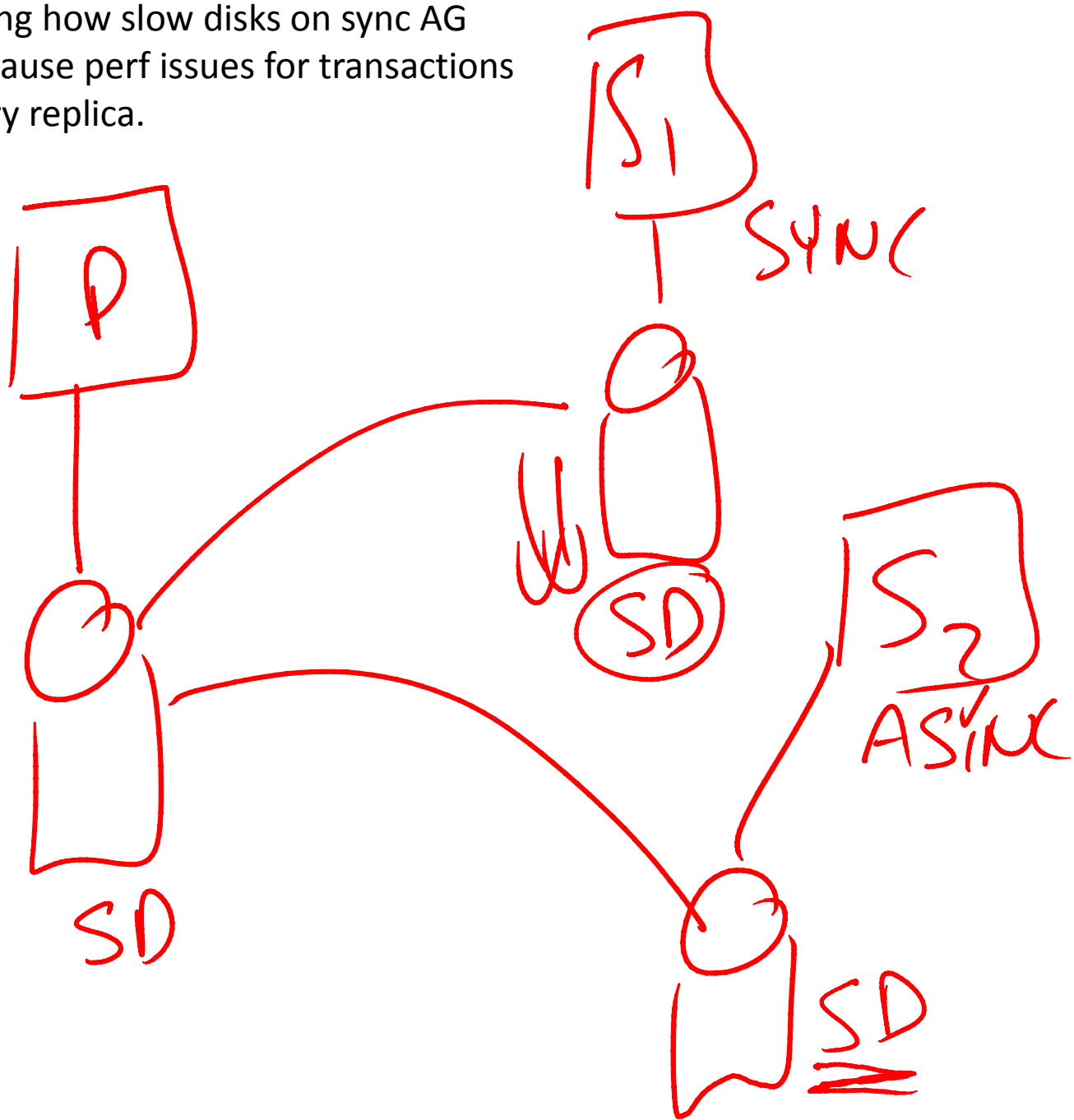


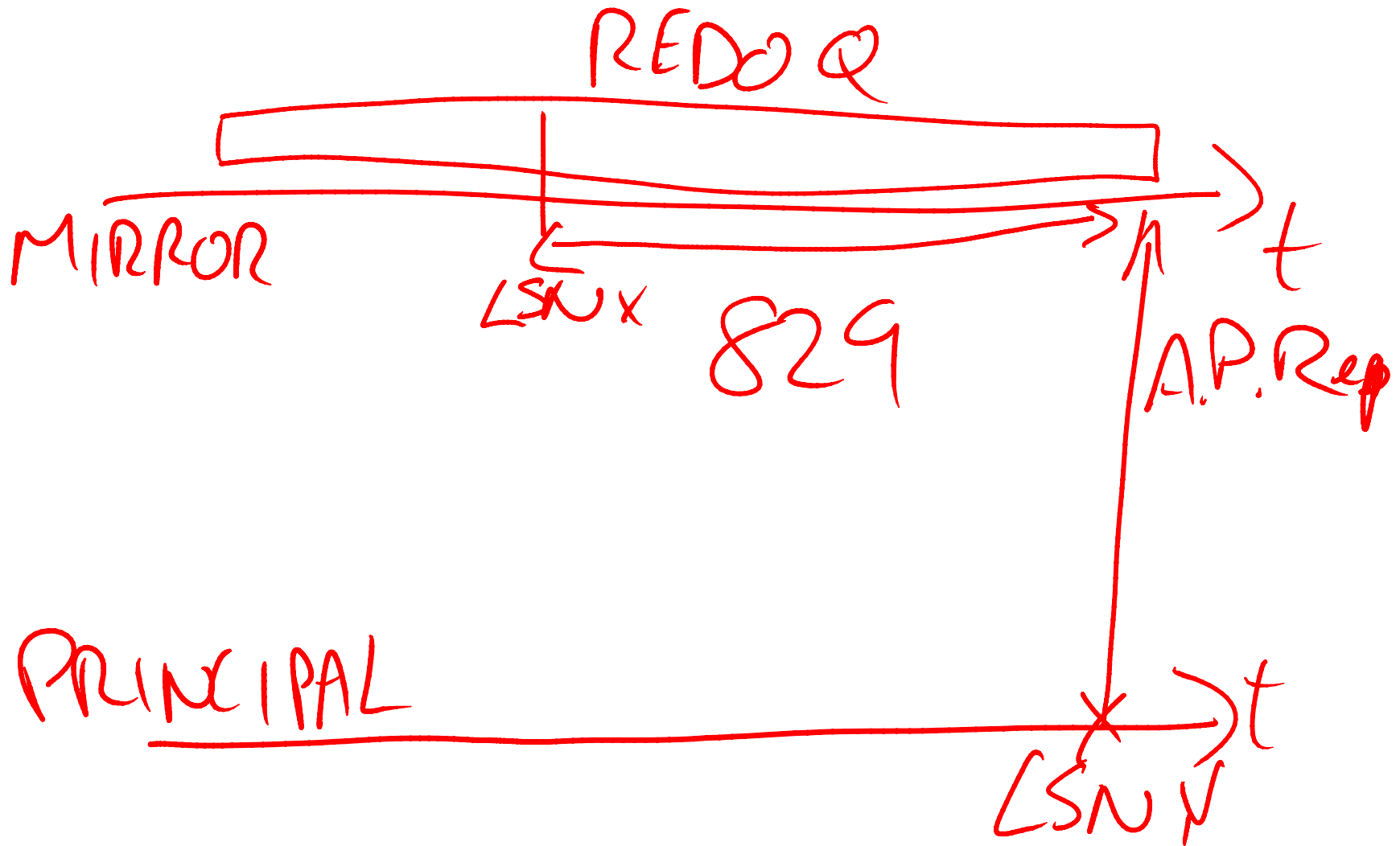
M10S10 Explaining the mirroring and AGs are shipping log blocks, so random I/O subsystem corruption will not be propagated.



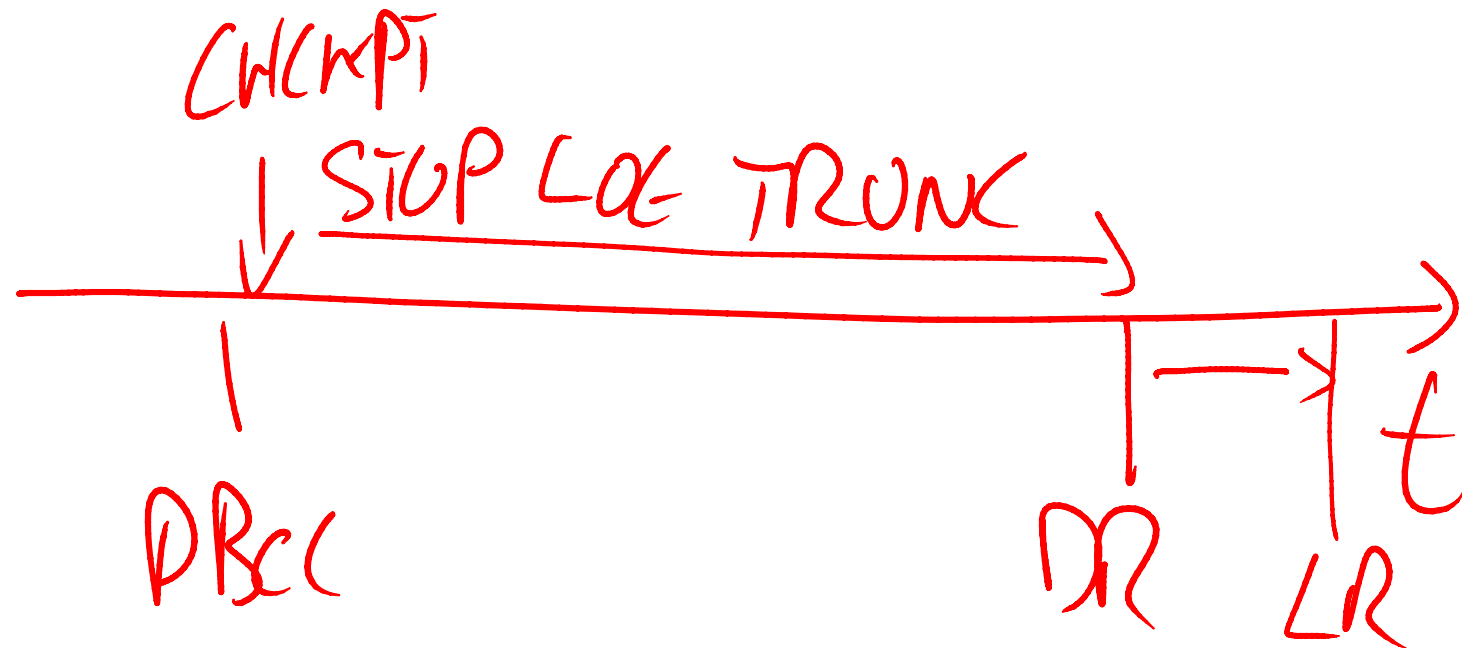
M10S10 Explaining how I/O subsystem-based replication on ships disk blocks changed by the host.

M10 Explaining how slow disks on sync AG replicas can cause perf issues for transactions on AG primary replica.

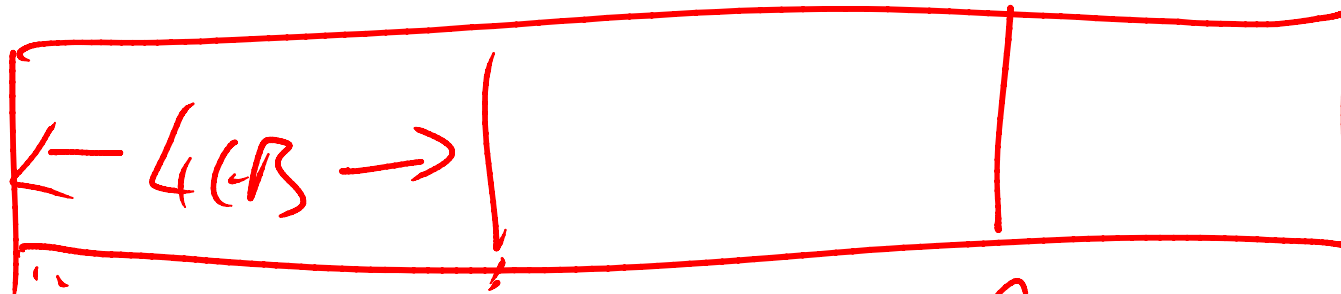




M10S21 Explaining how automatic page repair in DBM may not be instant, because the mirror REDO queue needs to get to the LSN at which the principal asked for the damaged page, to avoid sending an older page image to the principal



M10S25 Explaining how CHECKDB in 2000 was online, using log analysis to perform crash recovery inside CHECKDB. Log truncation was prevented, leading to log growth during long CHECKDB runs. I had fun ripping out the 10,000 lines of code it took for this during 2005 dev when I switched to using a database snapshot.



GAM INTERVAL
4GB

T1 ... IAM

T2 ... IAM

M10S26 Allocation checks will XOR all the IAM pages for a single 4GB GAM interval together to make sure no two tables have the same extents allocated to them.

XOR = $\emptyset$

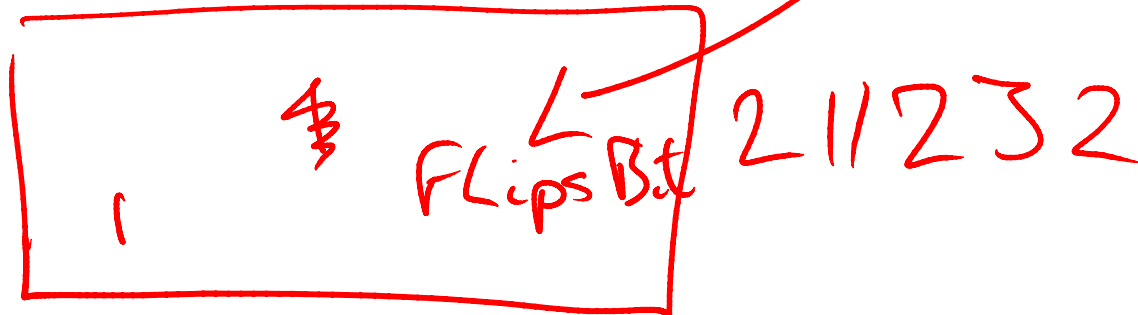
TABLE

[c1, c2, c3]

[NC 1 c1
NC 2 c2]

→ gen NC 1 record

→ HASH → 4 byte #



M10 Explaining how the nonclustered index checking algorithm works. Each data record is used to construct all NC index records, then each is hashed to a value and a bit is flipped in a bitmap. Hashing the actual NC index records should map to the same bits, flipping the bits back again. Lossy algorithm, so any bits set at end mean a deep-dive to figure out which records are incorrect.