

2TB

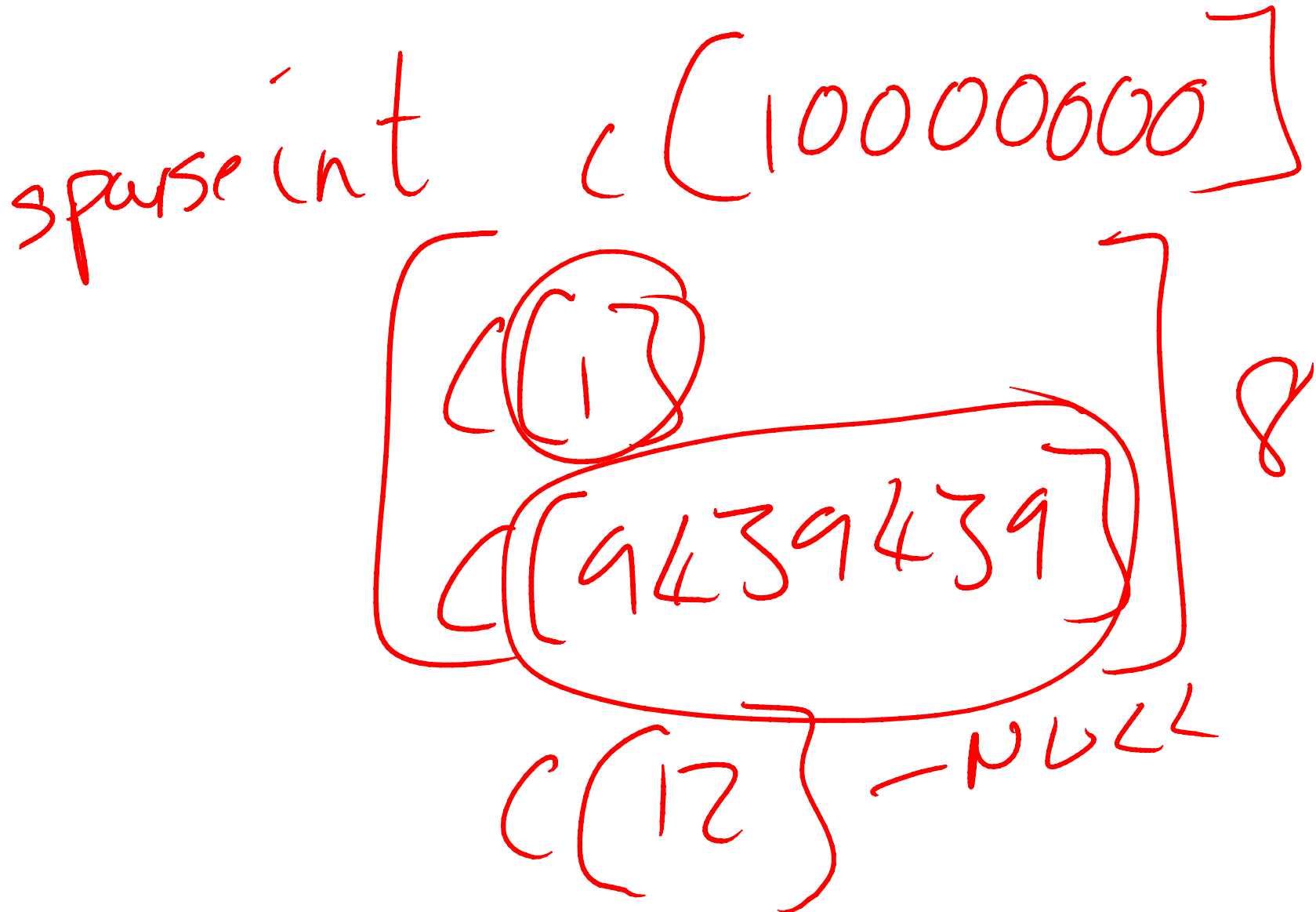
M4 Explaining that a database snapshot sparse files appear to be the same size as the database data files, but they're not. Their size on size is initially very small.

SPARSE FILES

64K

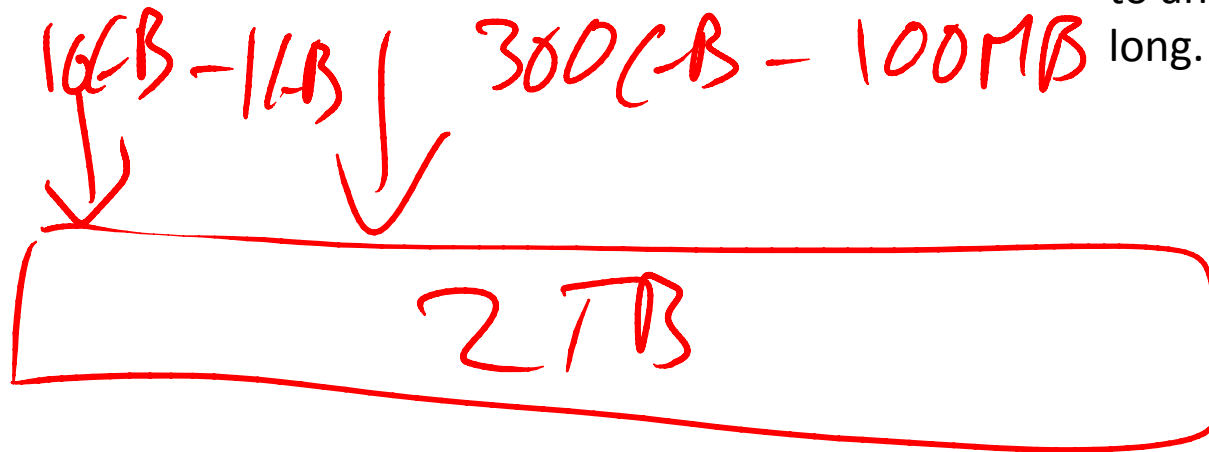
2TB

M4 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.

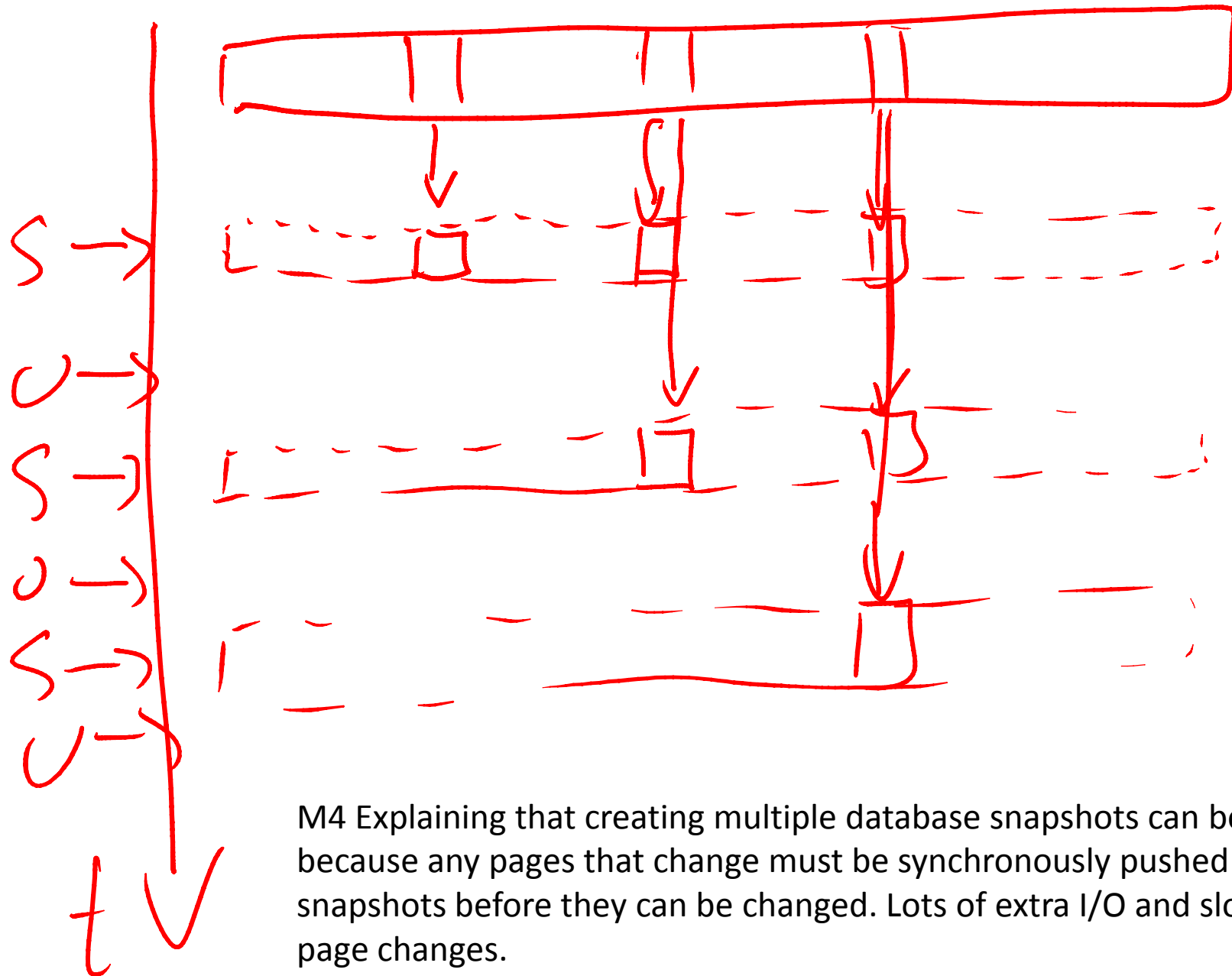


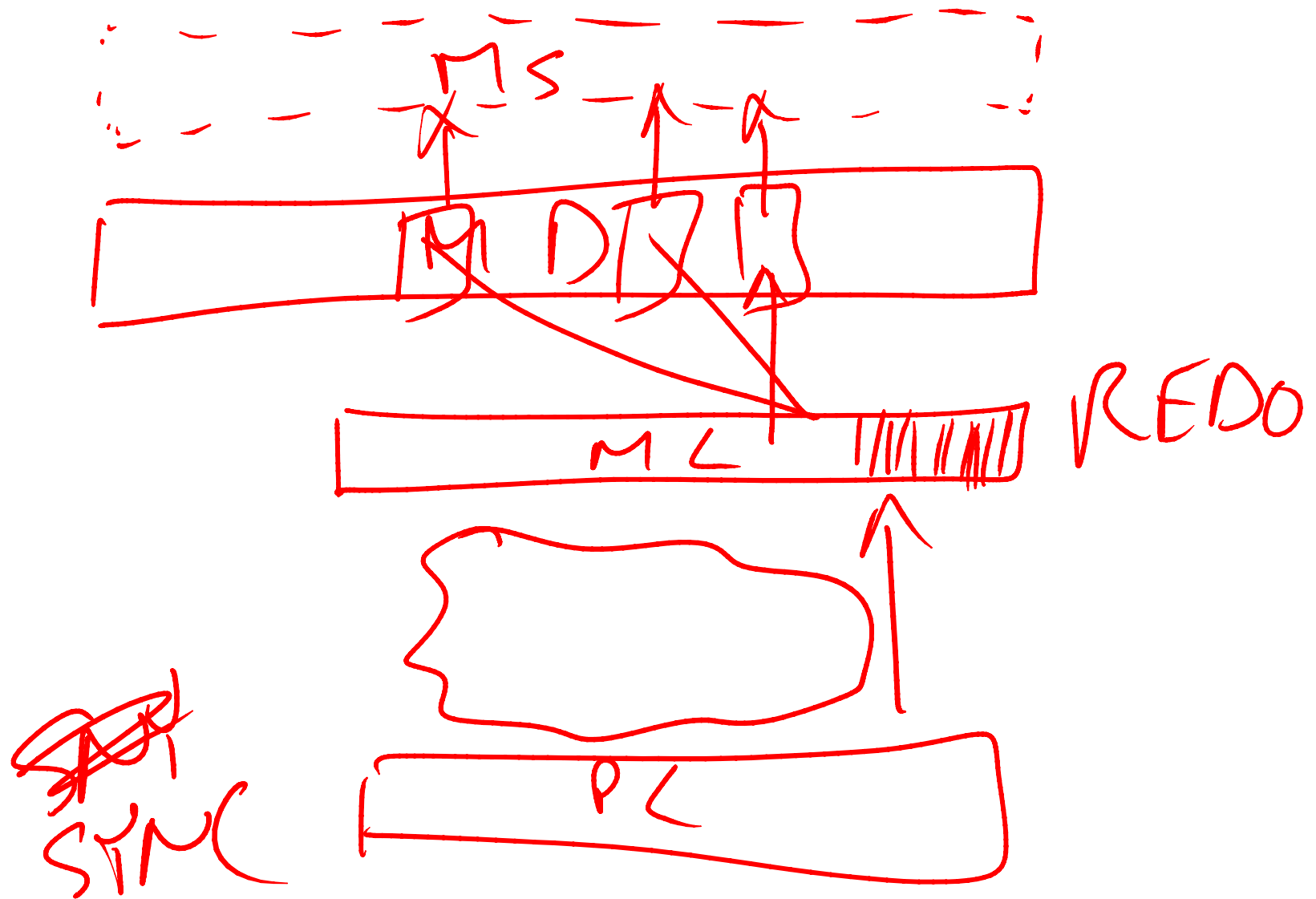
M4 Explaining that the NTFS sparse file keeps track of which file offsets have been written to and for how long.

SPARSE FILE



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





M4 Explaining how a snapshot on a mirror can slow down REDO processing on the mirror, and if too much I/O contention is created on the mirror, can also slow down transaction commits on the principal if using synchronous mirroring.

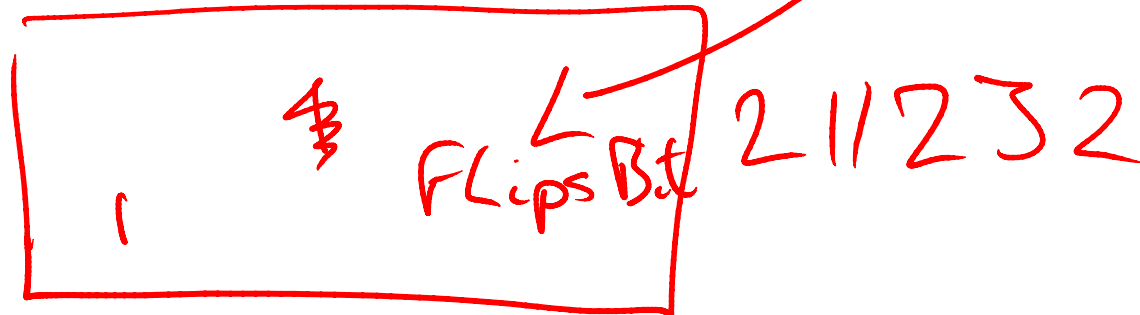
TABLE

[c1, c2, c3]

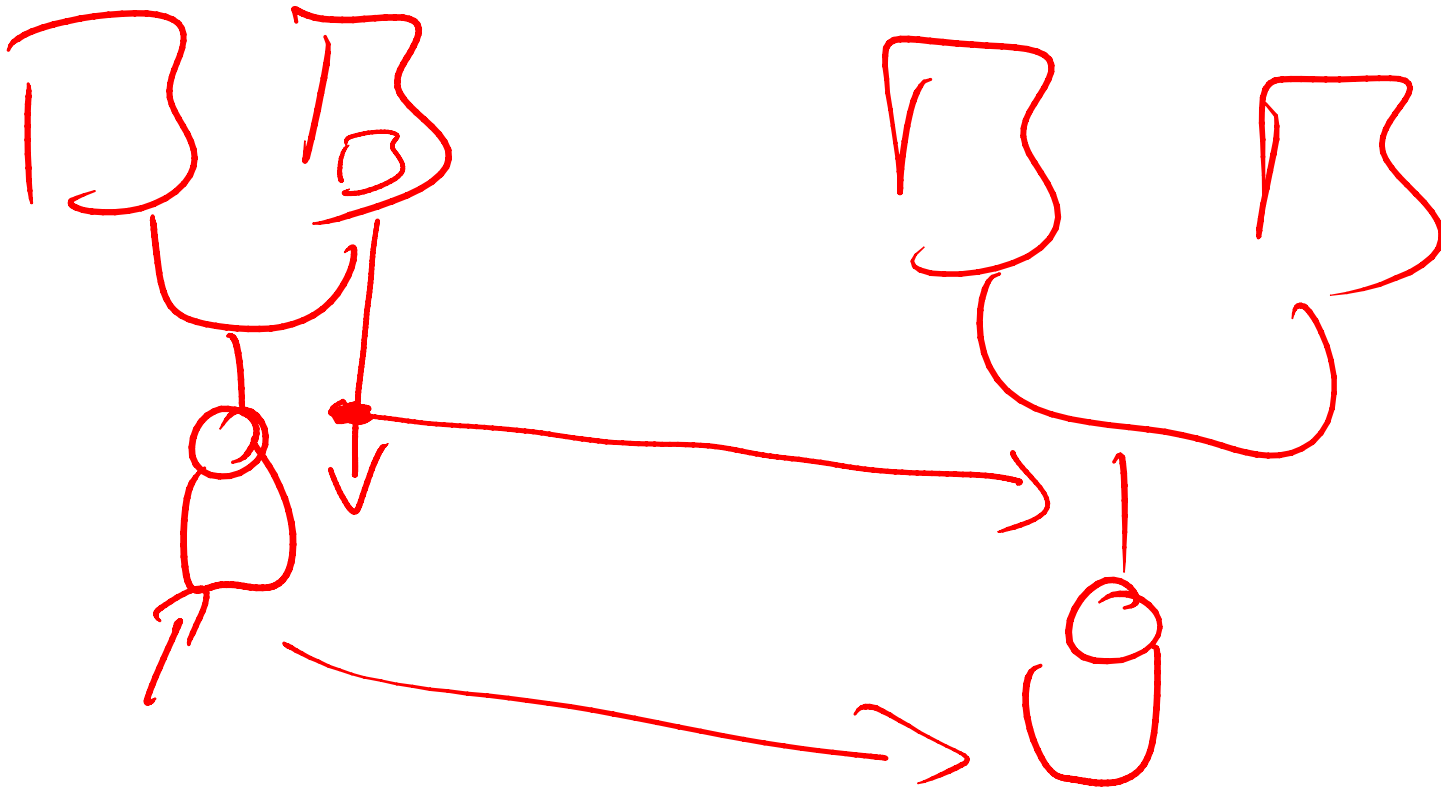
[NC 1 c1
NC 2 c2]

→ gen NC 1 record

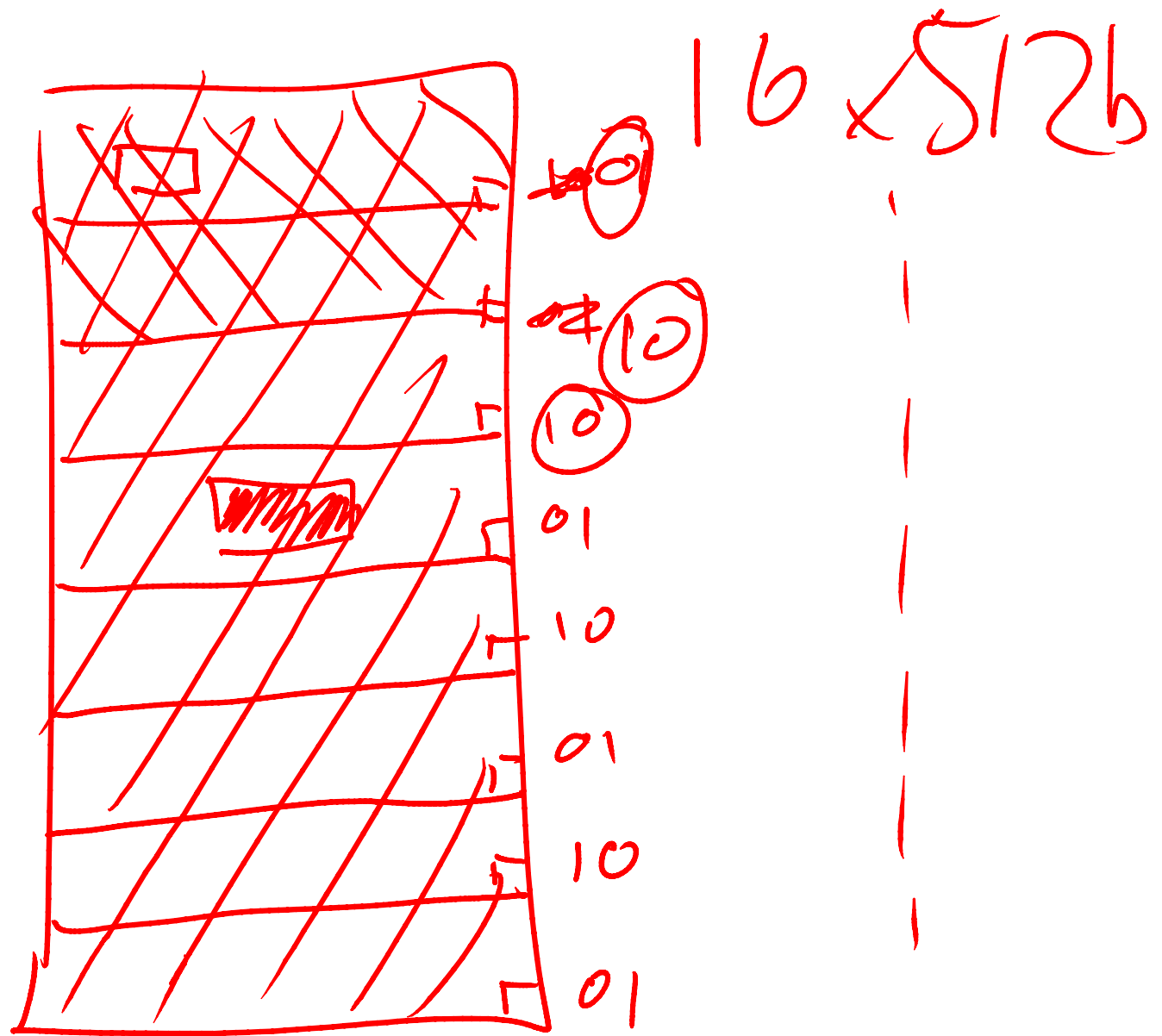
→ HASH → 4 byte #



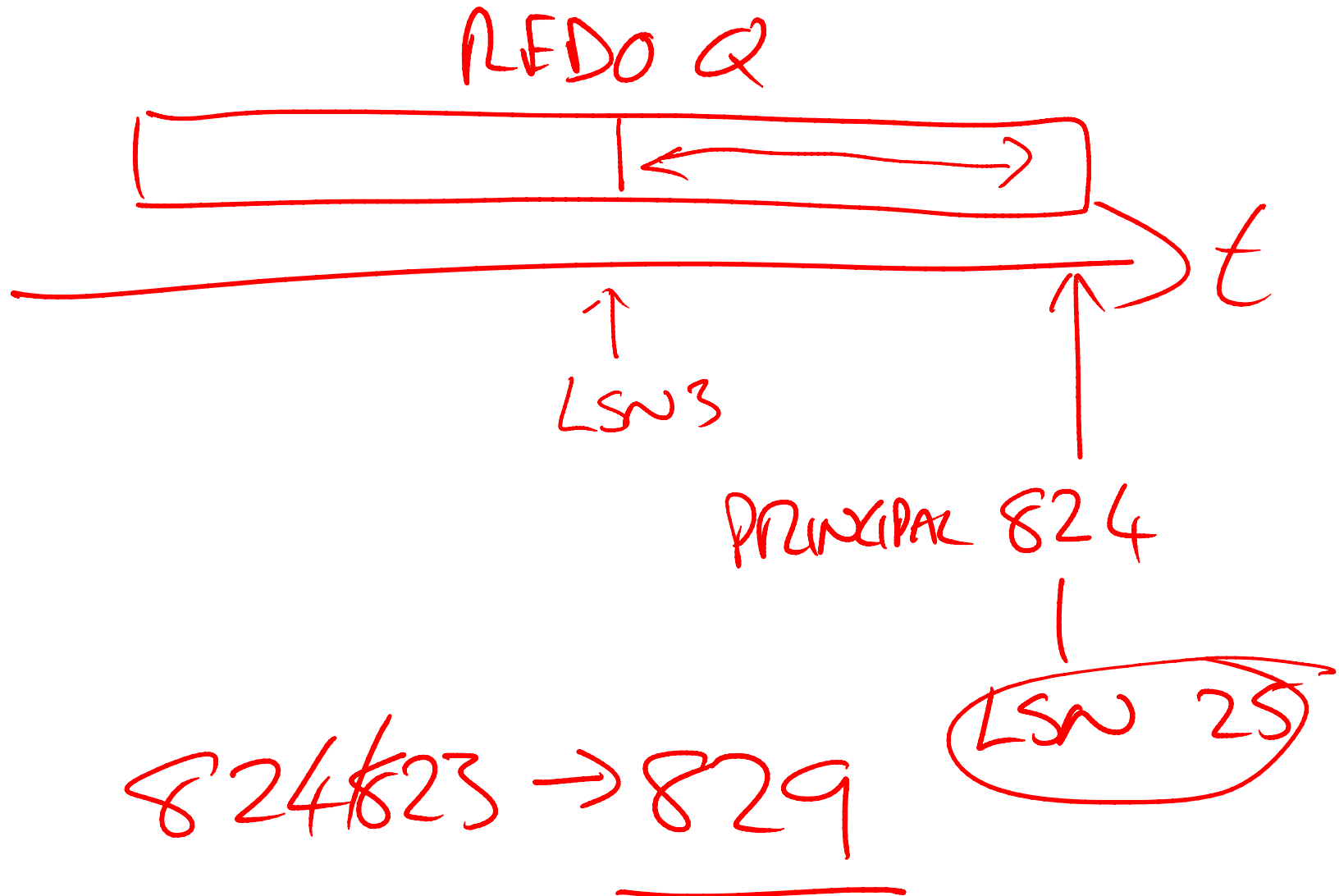
M10 Explaining how the nonclustered index checking algorithm works. Each data record is used to construct all NC index records, then each is hashed to a value and a bit is flipped in a bitmap. Hashing the actual NC index records should map to the same bits, flipping the bits back again. Lossy algorithm,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



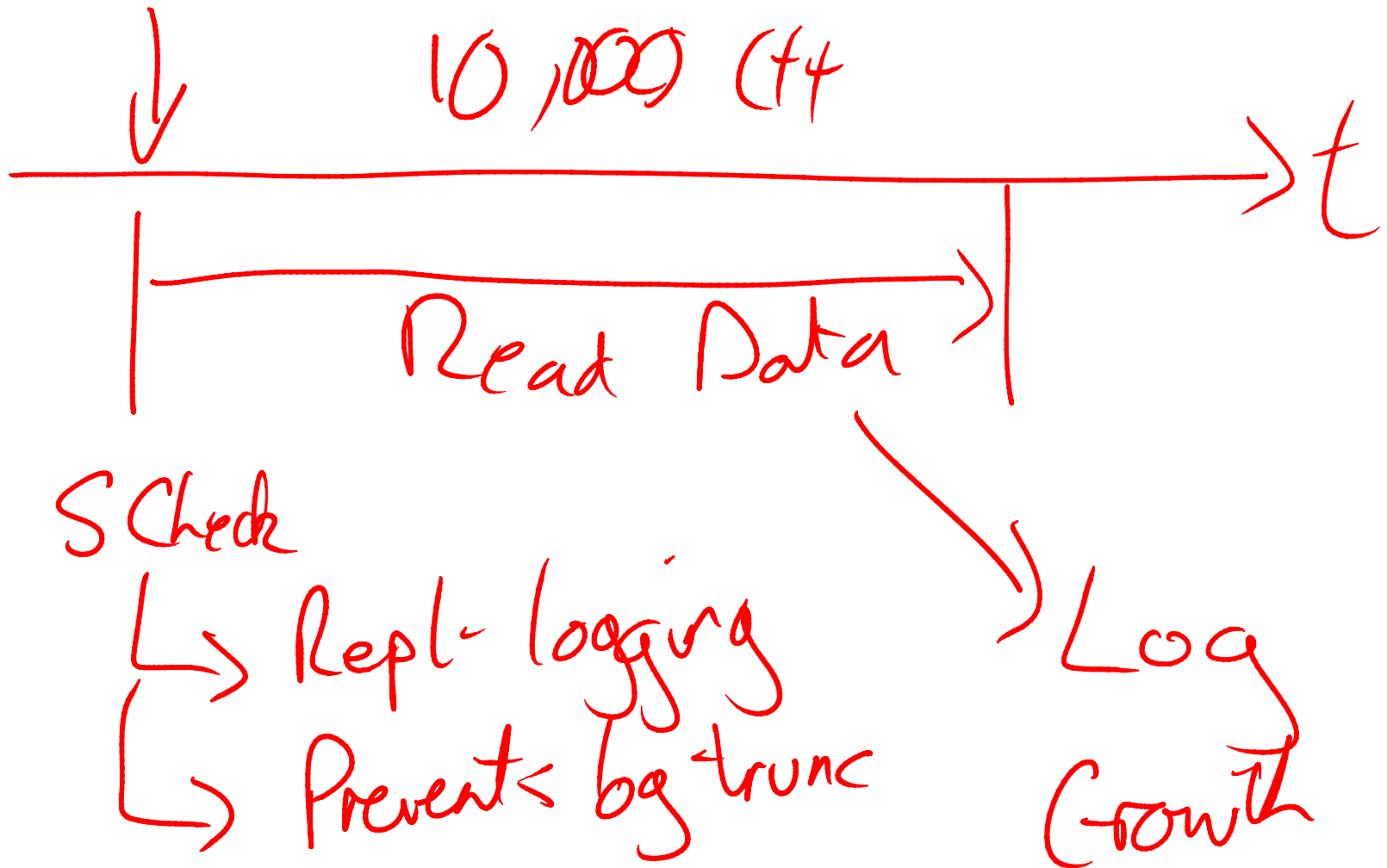
M10 Explaining that I/O corruption doesn't necessarily get sent to the secondary side of a mirrored SAN if the corruption occurs after the point at which the I/O is sent to the mirror.



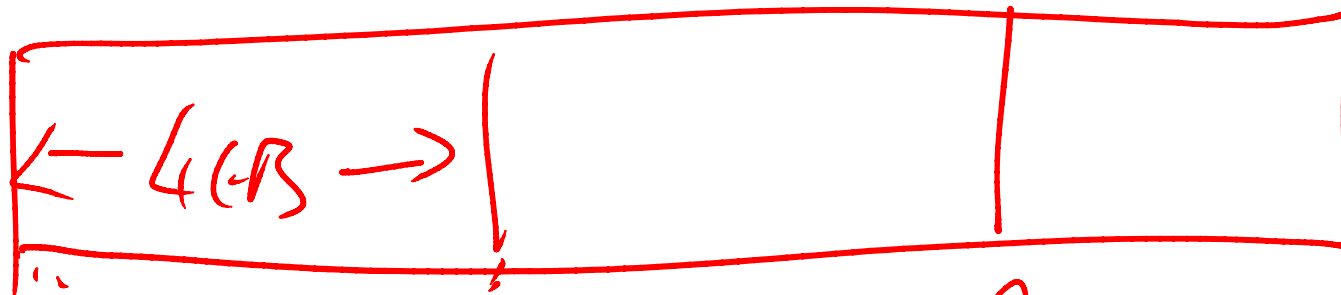
M10S11 Explaining torn-page detection



M10S15 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



M10S23 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

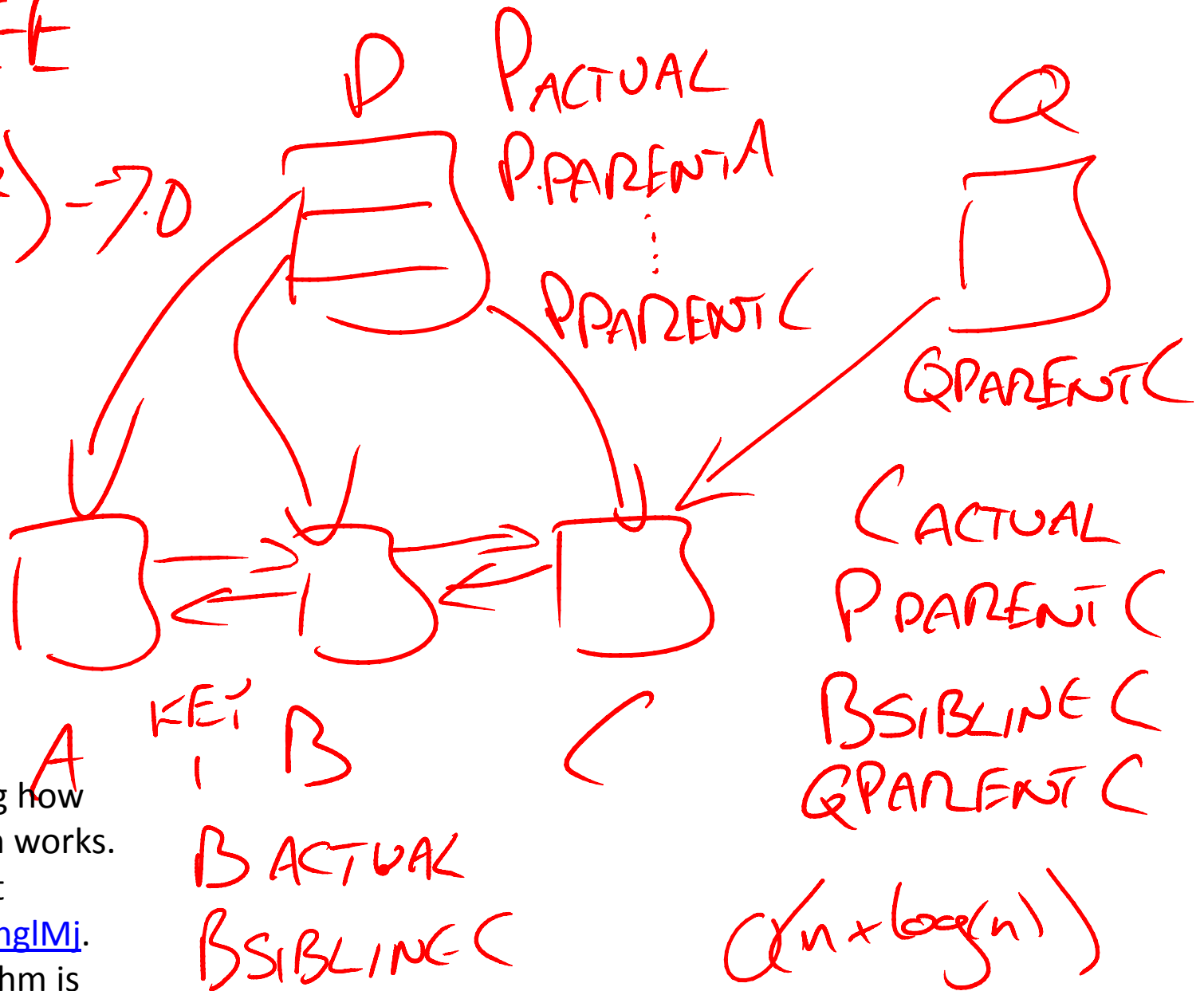
T2 ... IAM

M10 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 = $\emptyset$

BSTREE

$O(n^2) \rightarrow 0$



M10 Explaining how
fact generation works.
More details at
<http://bit.ly/QnglMj>.
Current algorithm is
 $\text{Order}(n * \log(n))$.

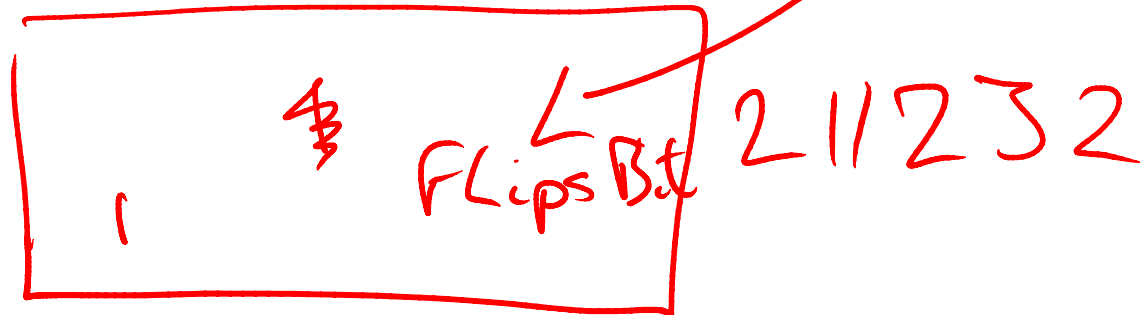
TABLE

[c1, c2, c3]

[NC 1 c1
NC 2 c2]

→ gen NC 1 record

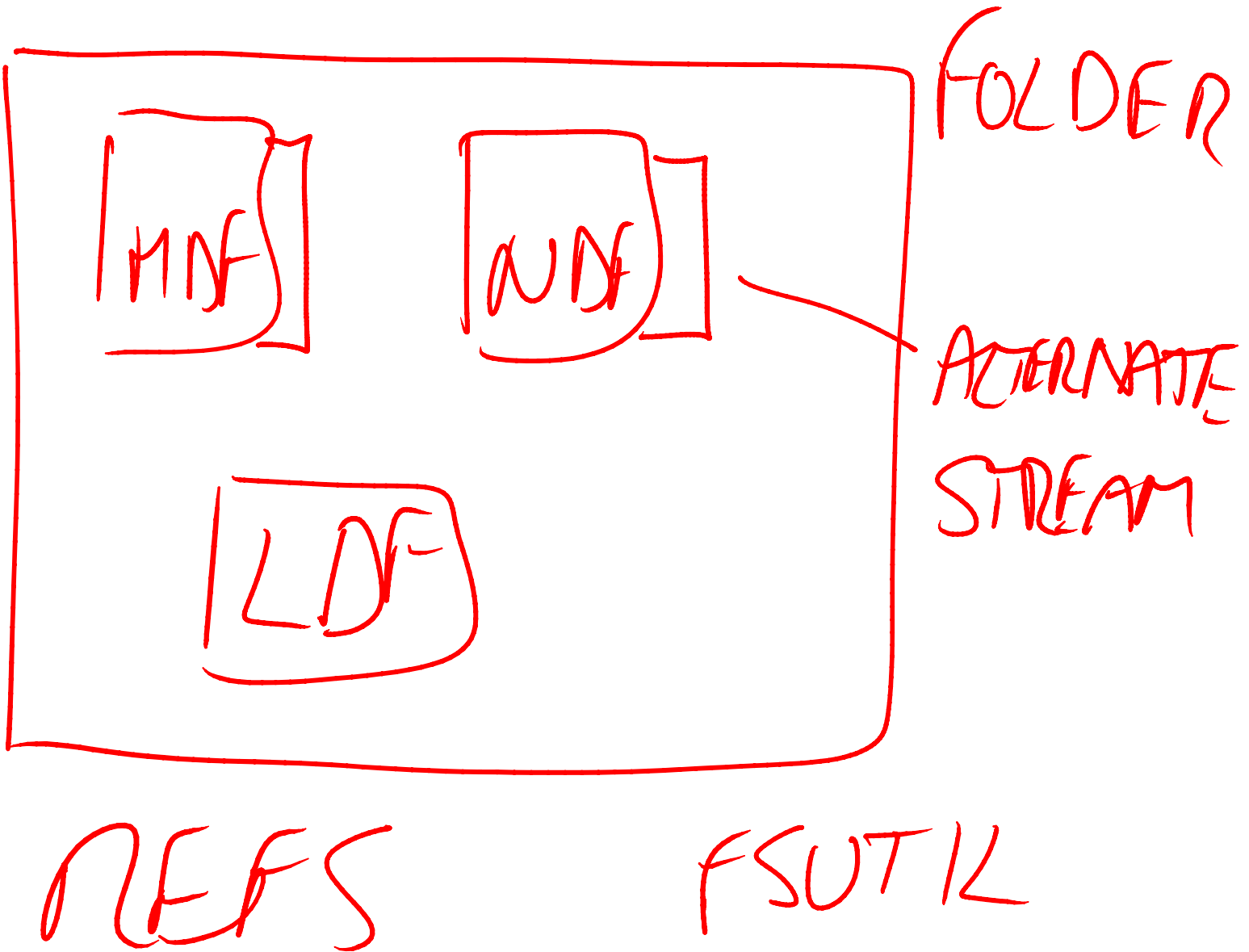
→ HASH → 4 byte #



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.

M10

Explaining that NTFS alternate streams that make up the hidden snapshot are stored in the same NTFS location as the data files of the database. REFS does not support sparse files (yet). You can see alternate streams using fsutil.



CLASSIFIER fn

M10S26 Explaining how Resource Governor works – a way of throttling CHECKDB CPU and memory.

WORKLOAD GROUPS

MAX_DOP
REMEME

Query Execution Memory Grant

CPU
MEM

TC/M

TC/M

RESOURCE POOLS

@@ERROR $\neq$ 0

M10S34 When CHECKDB completes, if it finds errors, the SQL Agent job or SSMS will reflect that. Programmatically, it will complete successfully (unless a nasty error forced it to stop) and so TRY ... CATCH will not work. If it found no errors, @@ERROR will be 0. If it found errors, @@ERROR will be non-zero.