

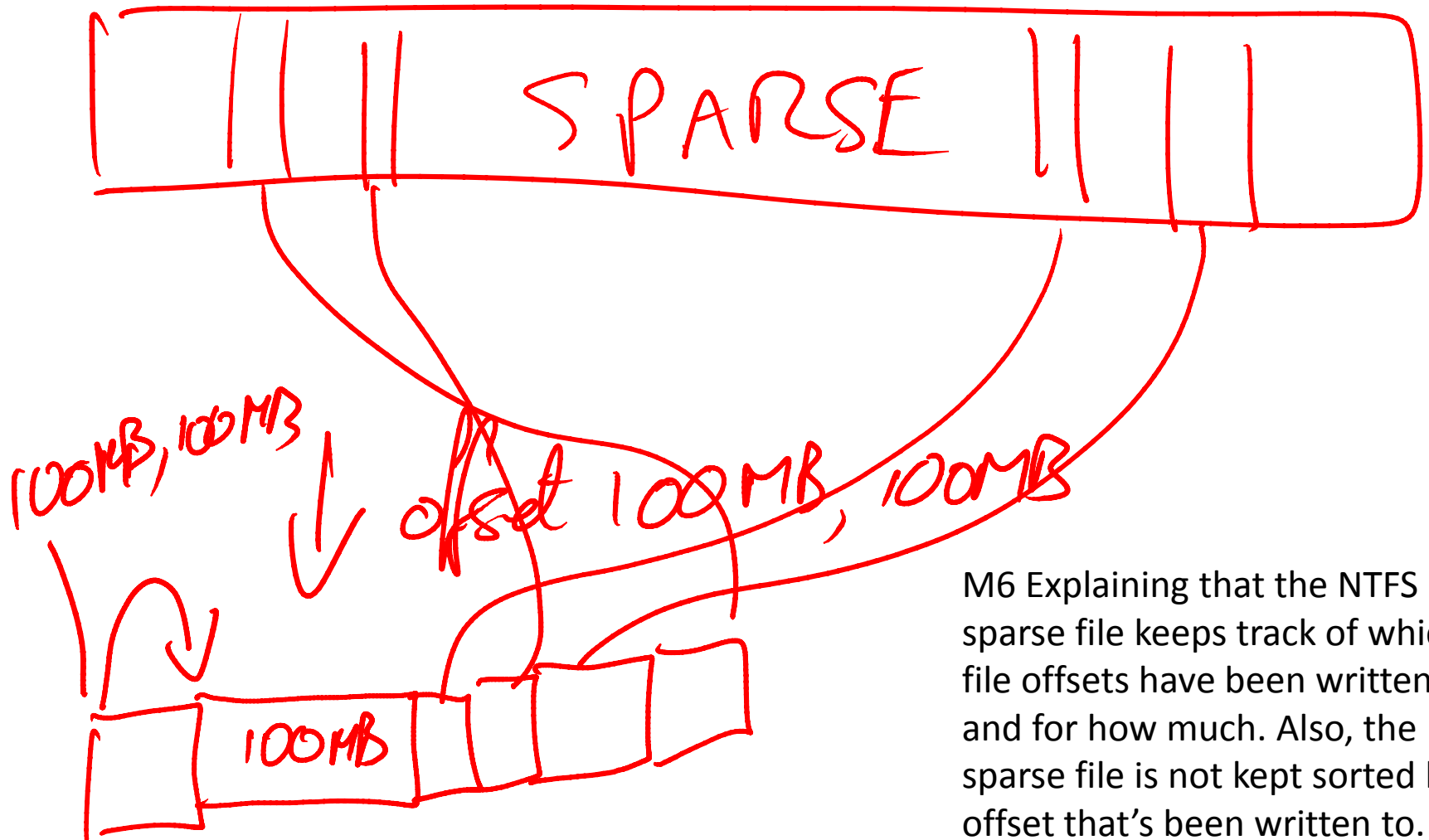
```
int c[100000000] sparse;
```

```
c[9466313] = 12;
```

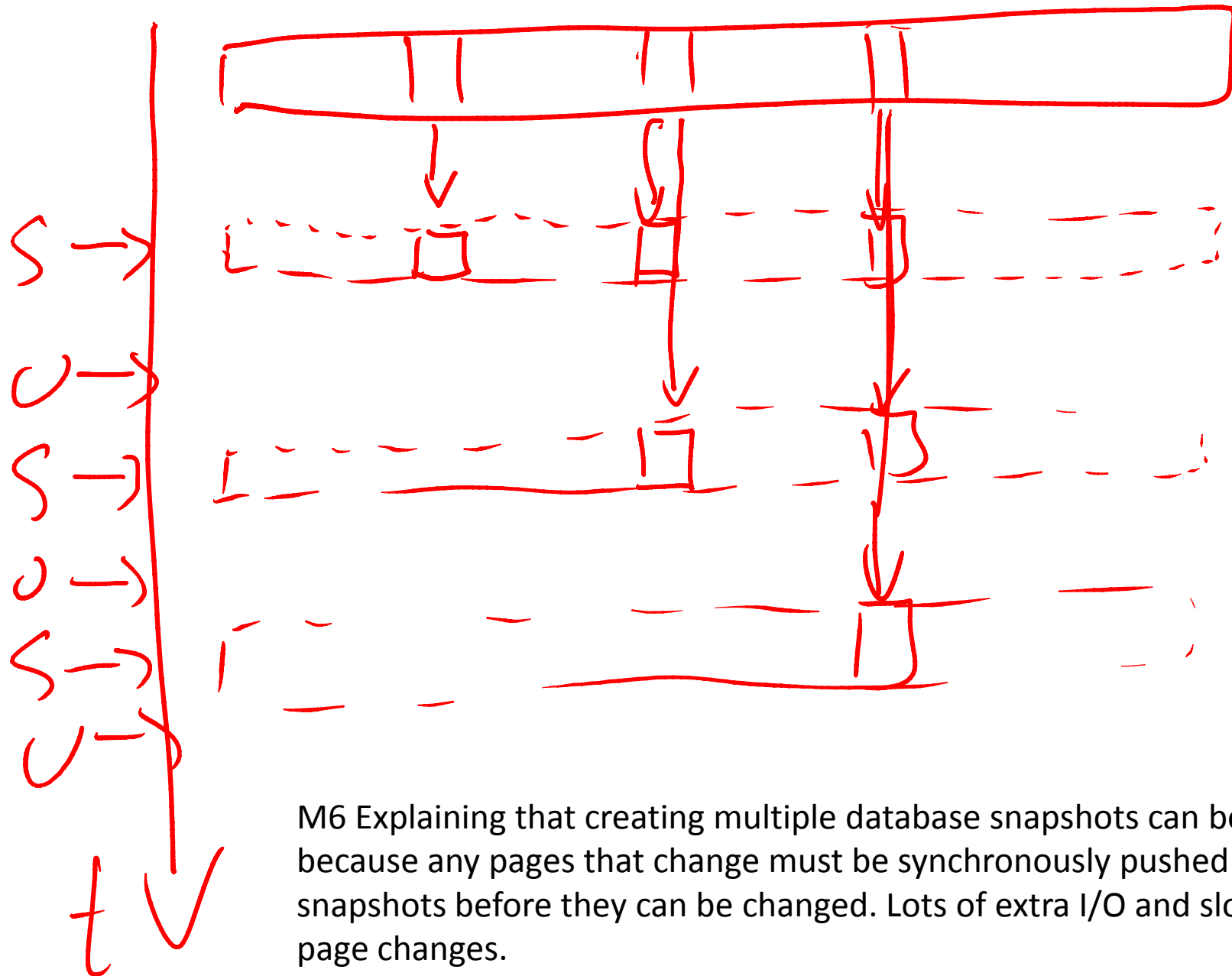
```
a = c[56]
```

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.

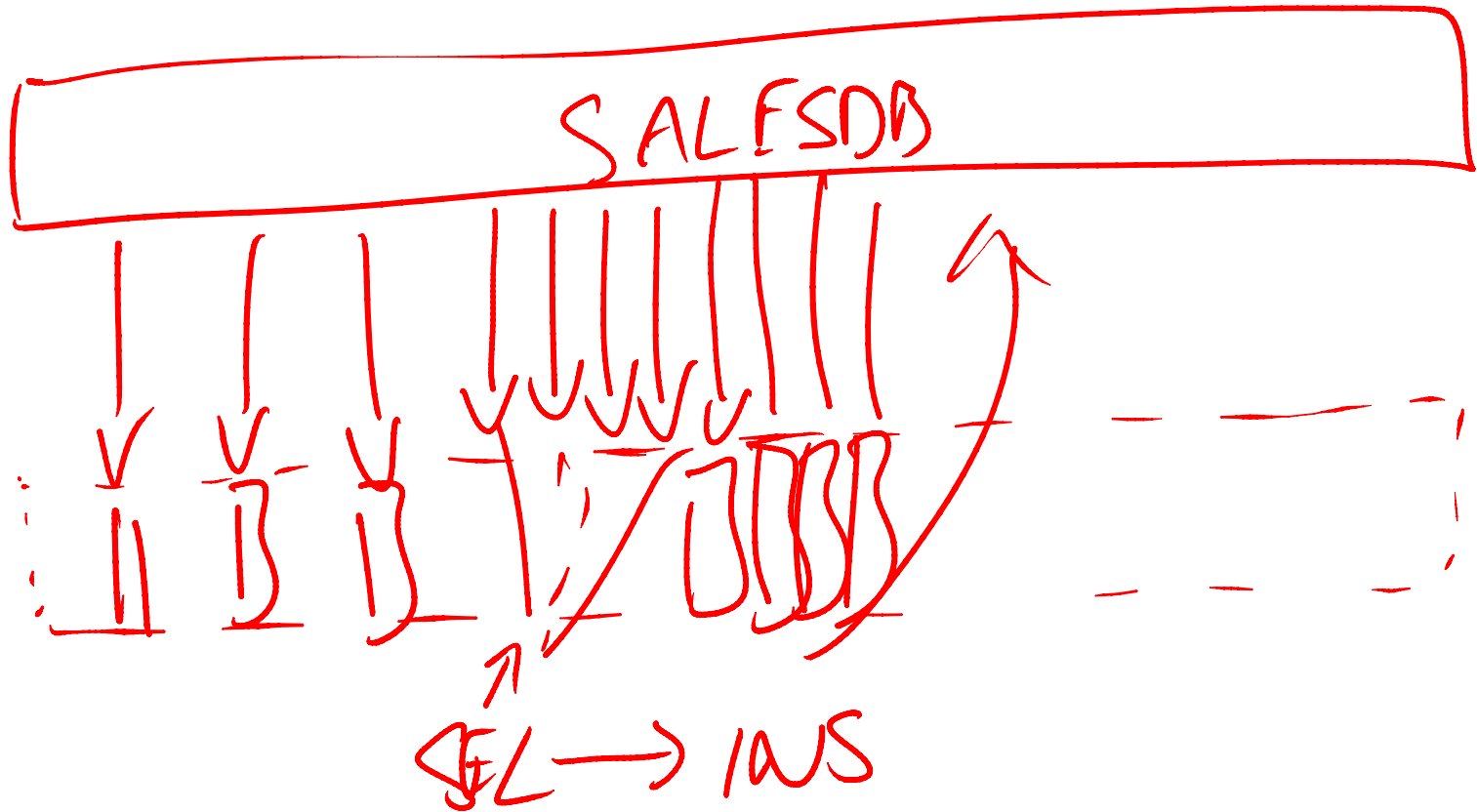
1 TB



M6 Explaining that the NTFS sparse file keeps track of which file offsets have been written to and for how much. Also, the sparse file is not kept sorted by offset that's been written to. The index at the start takes care of that.

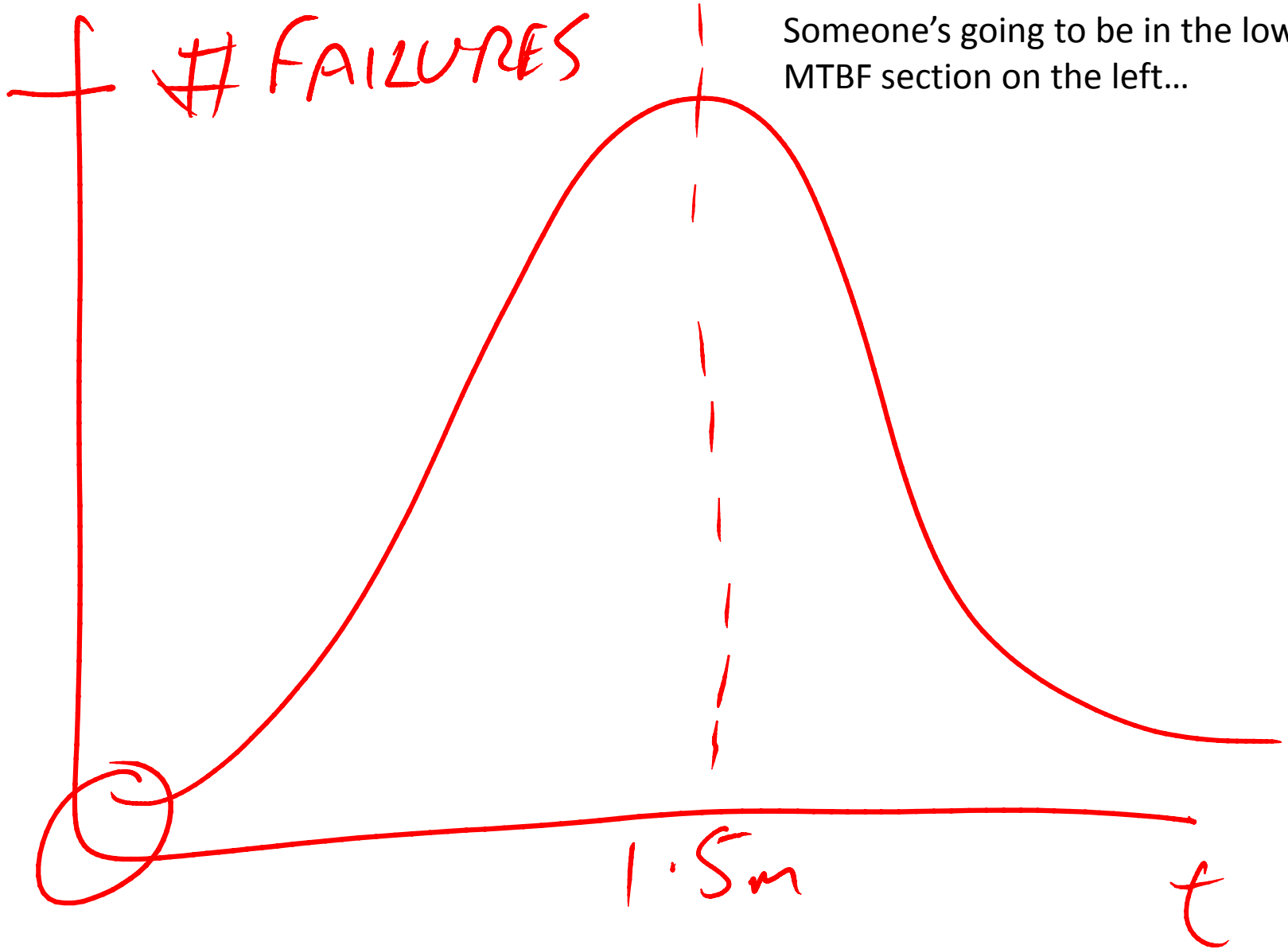


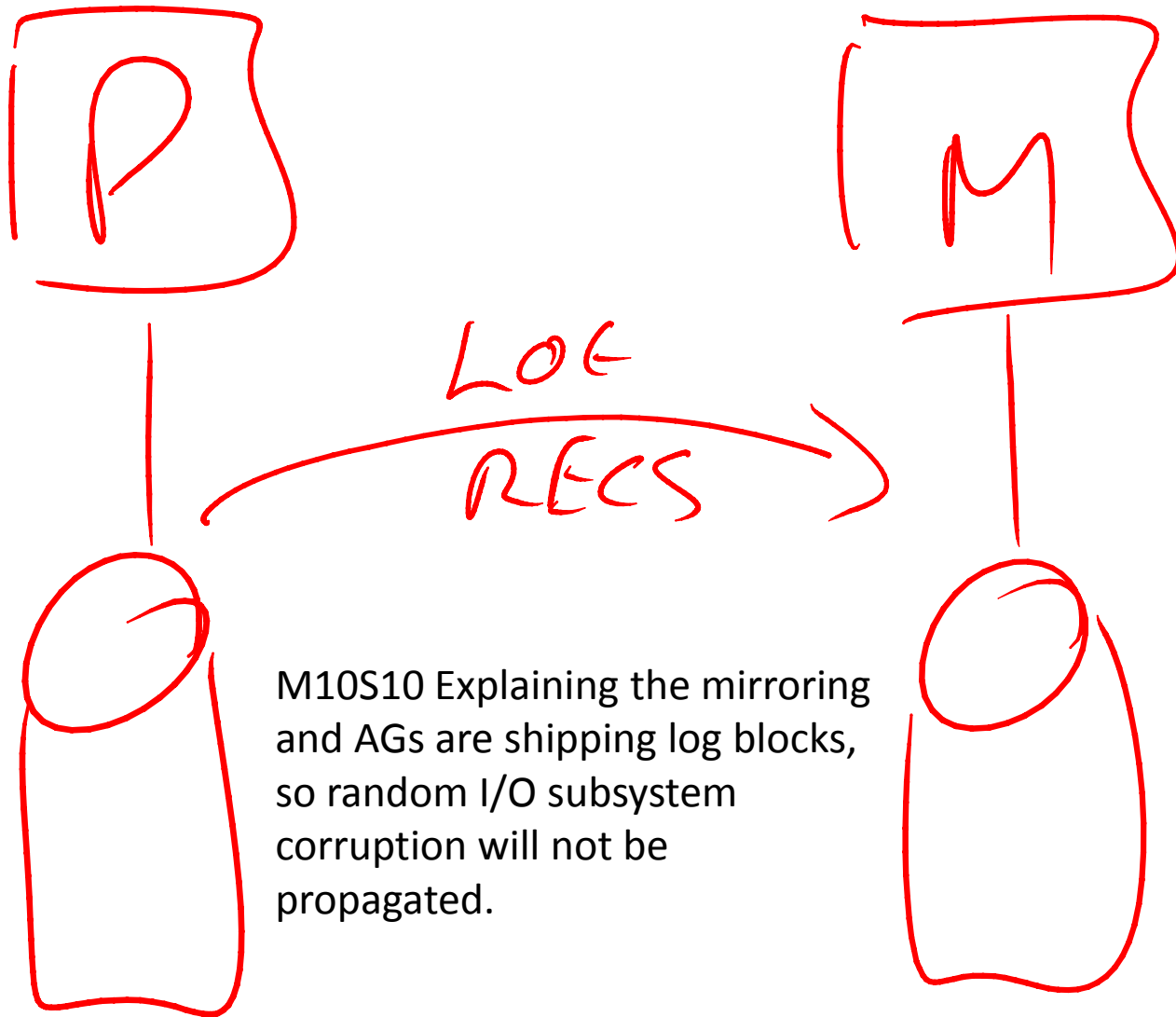
M6 Explaining that creating multiple database snapshots can be bad because any pages that change must be synchronously pushed into all snapshots before they can be changed. Lots of extra I/O and slows down page changes.



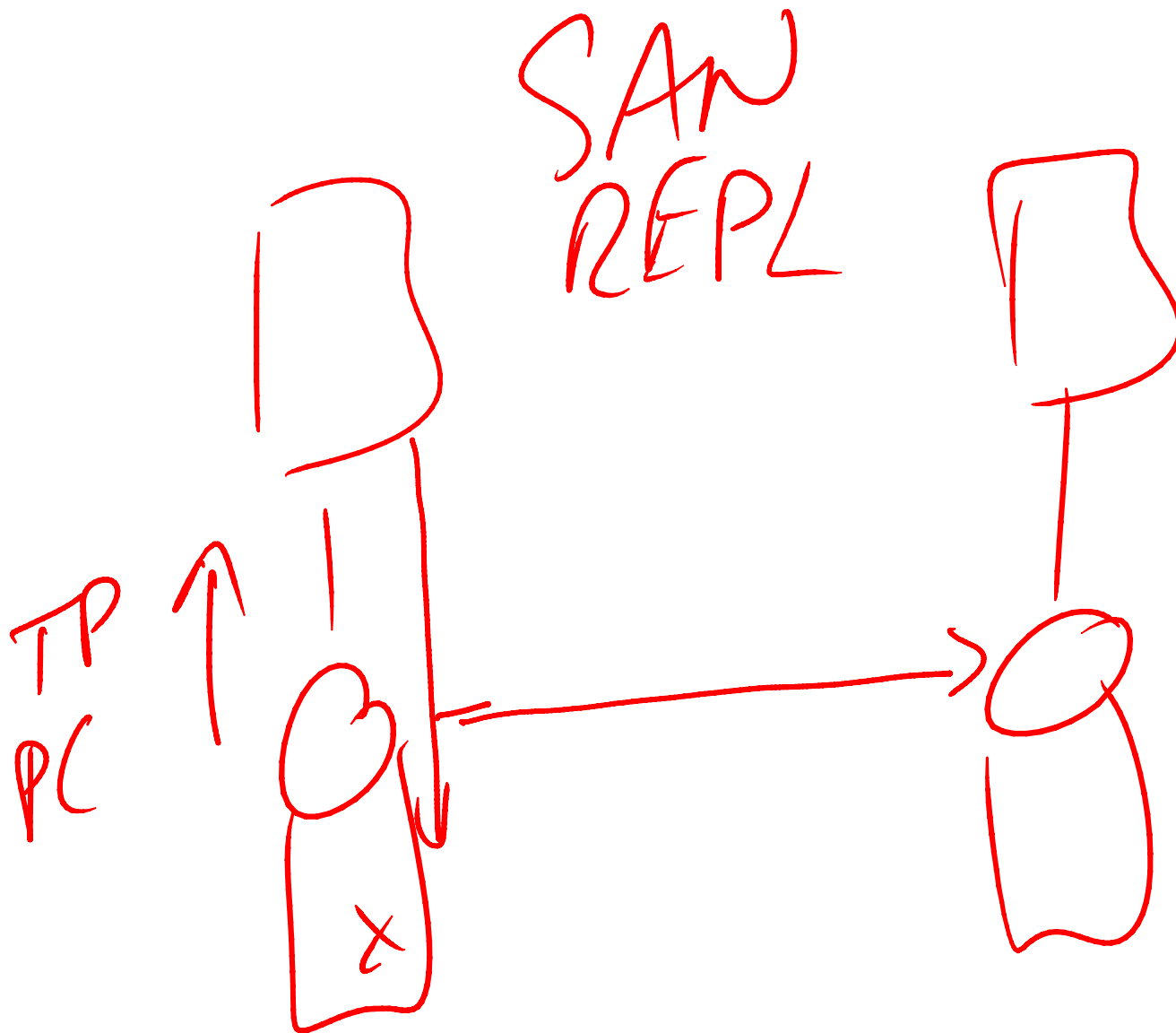
M6S17 Explaining the demo on recovering data from a snapshot. After dropping the table, the snapshot doesn't grow because the pages from the dropped table haven't been changed yet. As we repopulate the newly-created table though, it's pulling unchanged pages from the source database through the context 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...

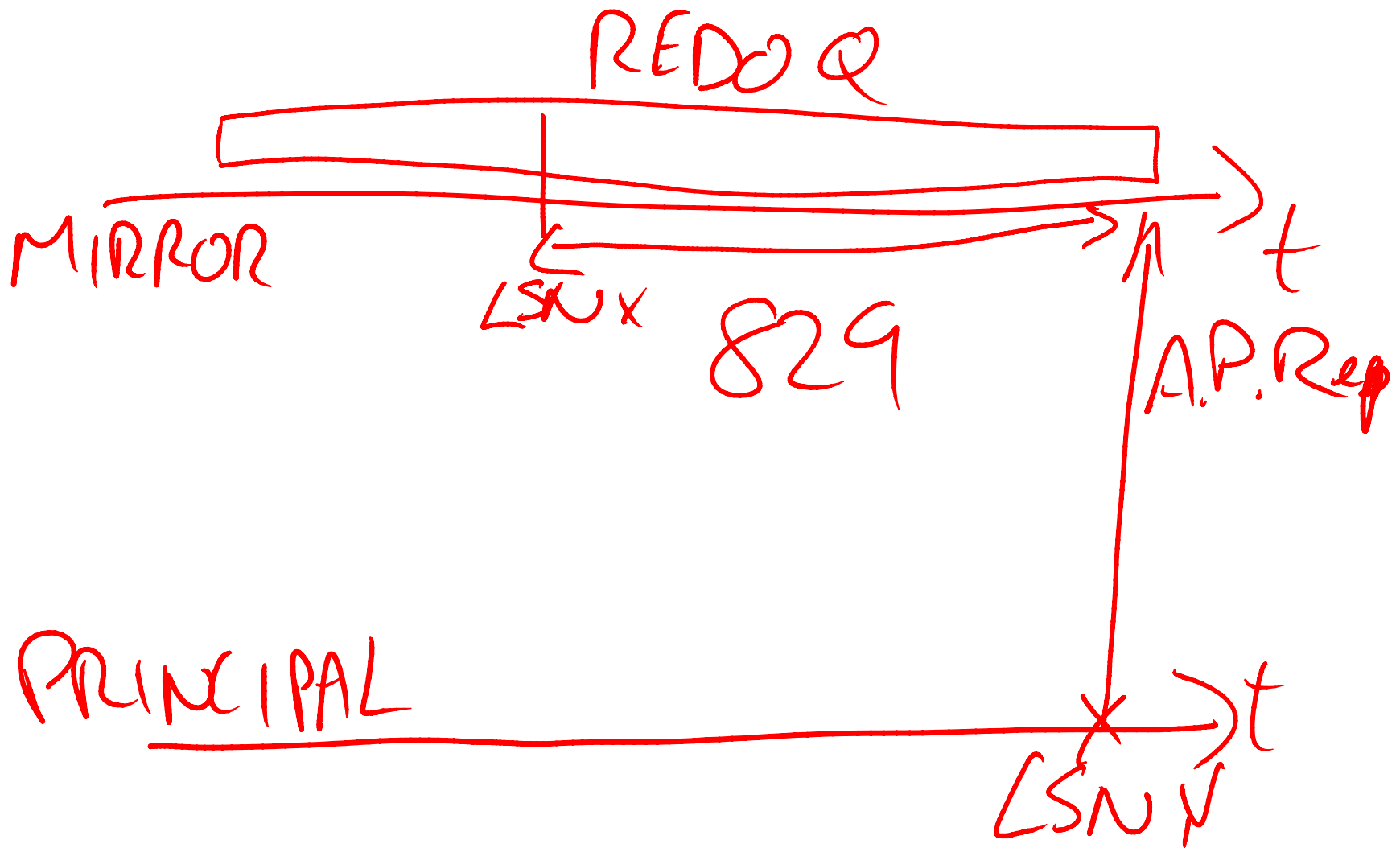




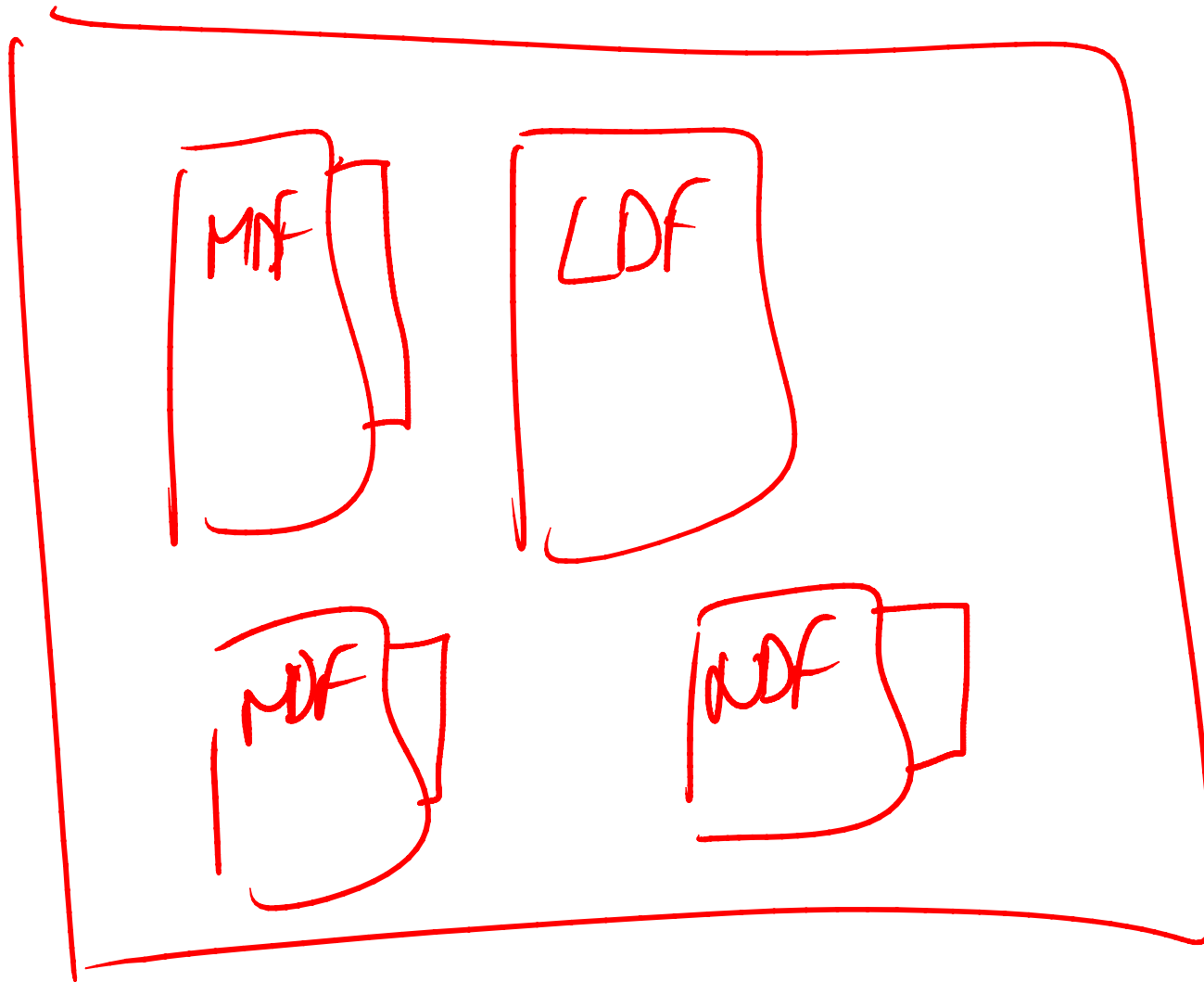
M10S10 Explaining the mirroring and AGs are shipping log blocks, so random I/O subsystem corruption will not be propagated.



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

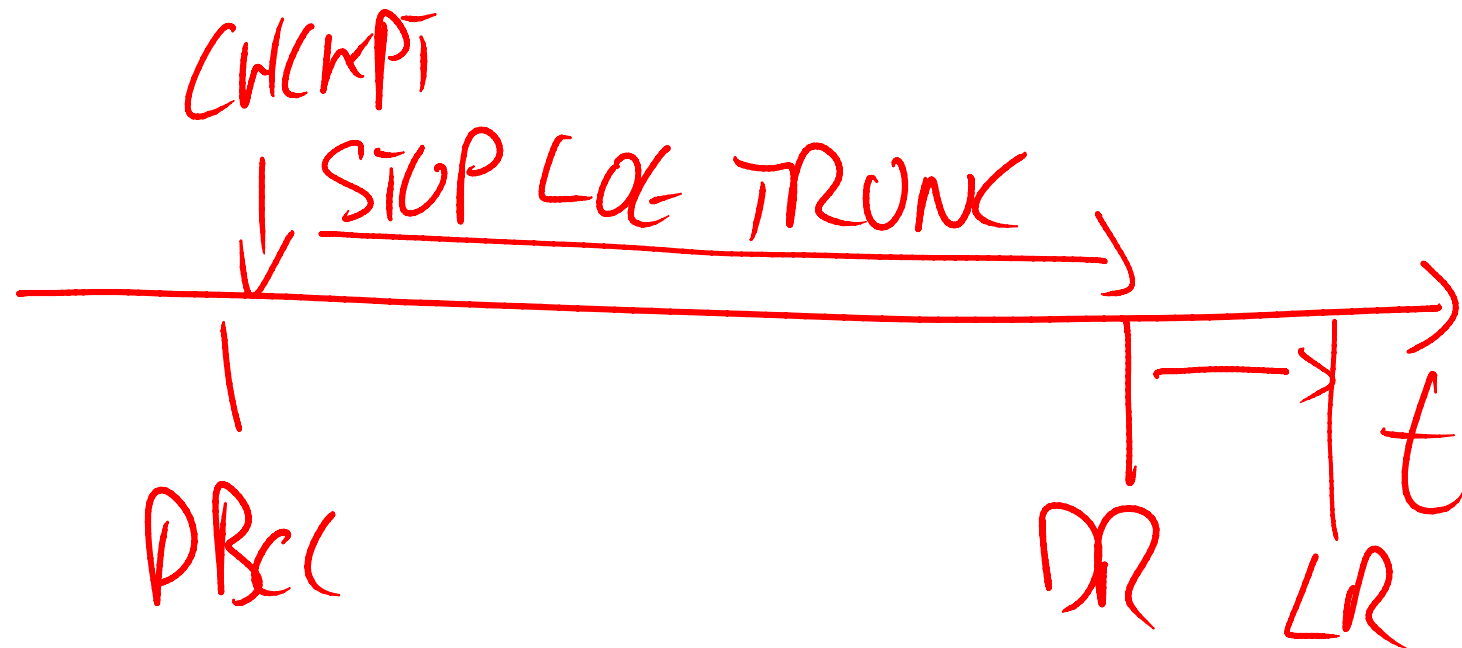


M10S21 Explaining how automatic page repair in DBM may not be instant, because the mirror REDO queue needs to get to the LSN at which the principal asked for the damaged page, to avoid sending an older page image to the principal



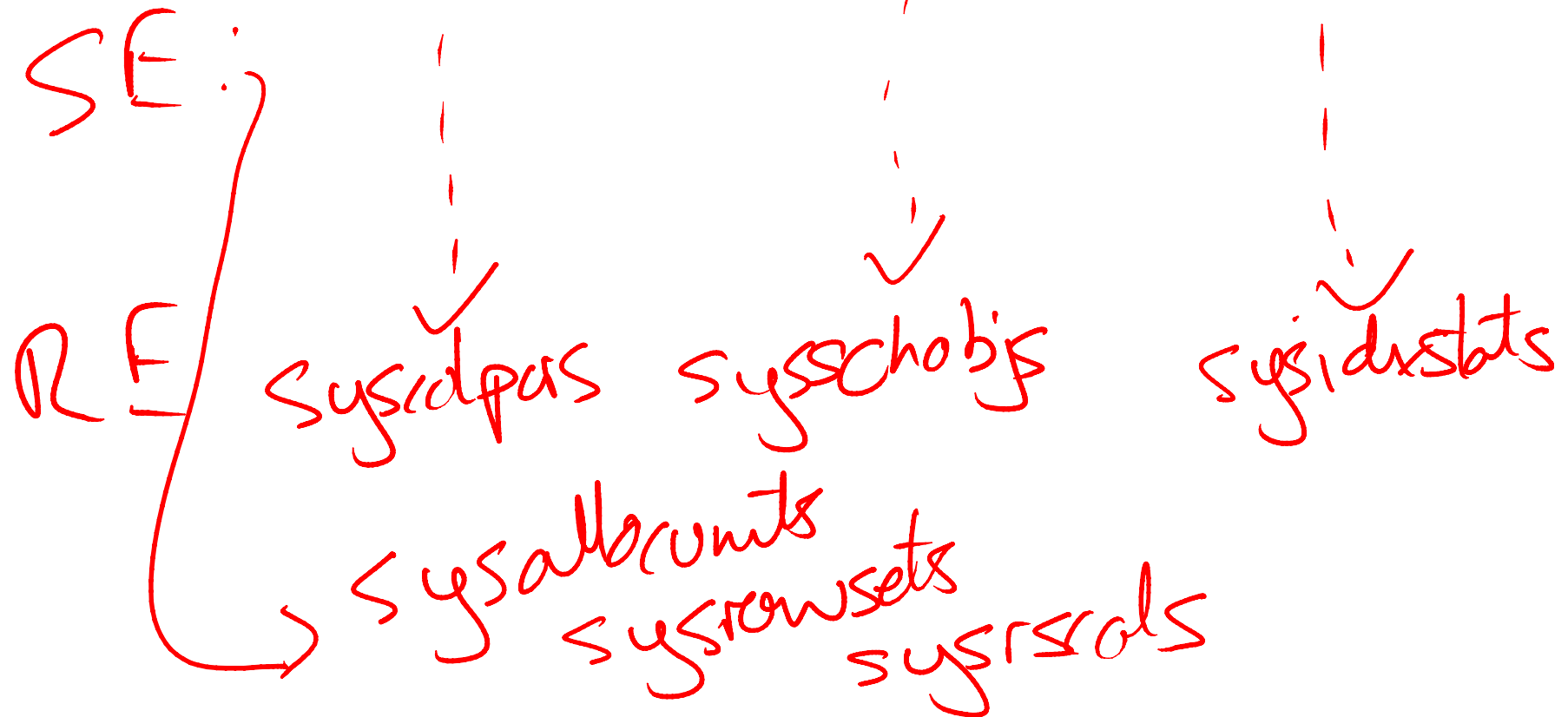
M10

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. You can see alternate streams using fsutil.

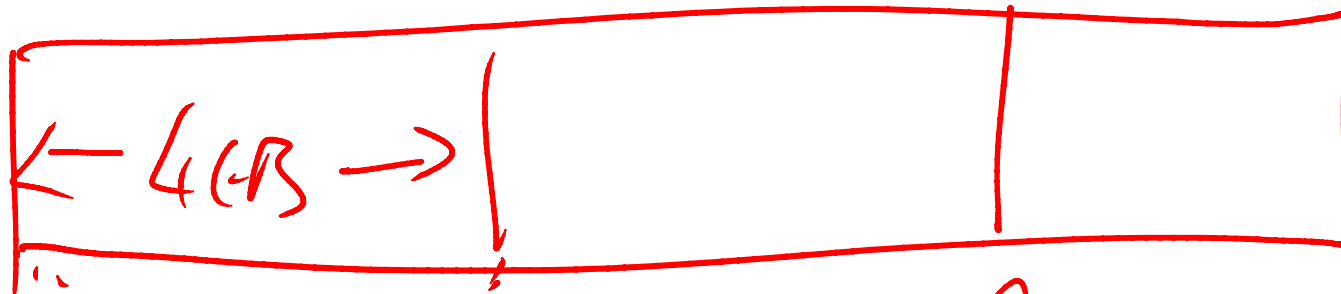


M10S25 Explaining how CHECKDB in 2000 was online, using log analysis to perform crash recovery inside CHECKDB. Log truncation was prevented, leading to log growth during long CHECKDB runs. I had fun ripping out the 10,000 lines of code it took for this during 2005 dev when I switched to using a database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

2000: syscolumns, sysobjects, sysindexes



M10 Explaining the names of the hidden system tables that store column metadata.



T1

IAM

T2

IAM

GAM INTERVAL
4GB

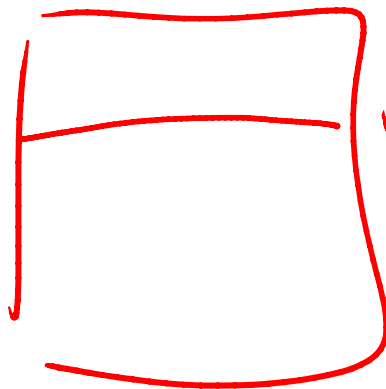
M10S26 Allocation checks will XOR all the IAM pages for a single 4GB GAM interval together to make sure no two tables have the same extents allocated to them.

XOR = $\emptyset$

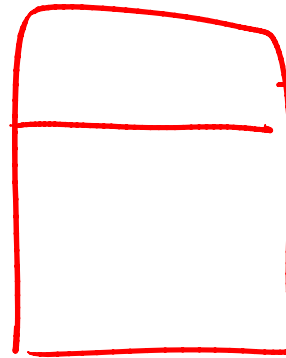
HEAP

NC1

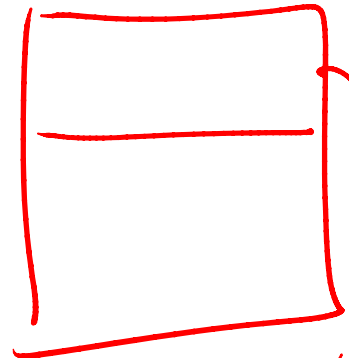
NC2



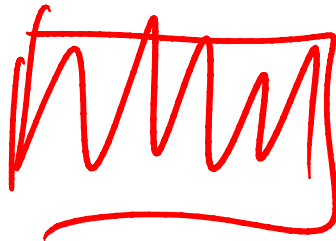
c1, c2, c3



c1



c2

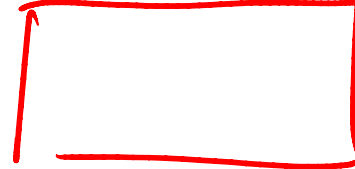


CHECKSUM

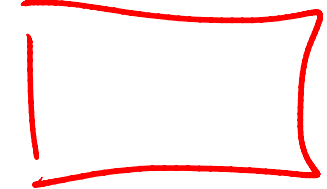
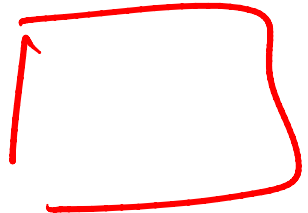
HASH

HASH

HASH



HASH



See next page...

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 the value is folded into a checksum for the nonclustered index. Each actual NC index records is hashed and folded into a second checksum for the nonclustered index. At the end of the checks, for each nonclustered index, the checksum created from the data records should match the checksum created from the index records. Lossy algorithm, so any inconsistencies mean a deep-dive to figure out which records are incorrect.

M10 When hacking-
attaching a SUSPECT
database, the file IDs
of the dummy
database you create
must match those of
the database you're
trying to hack-attach.

