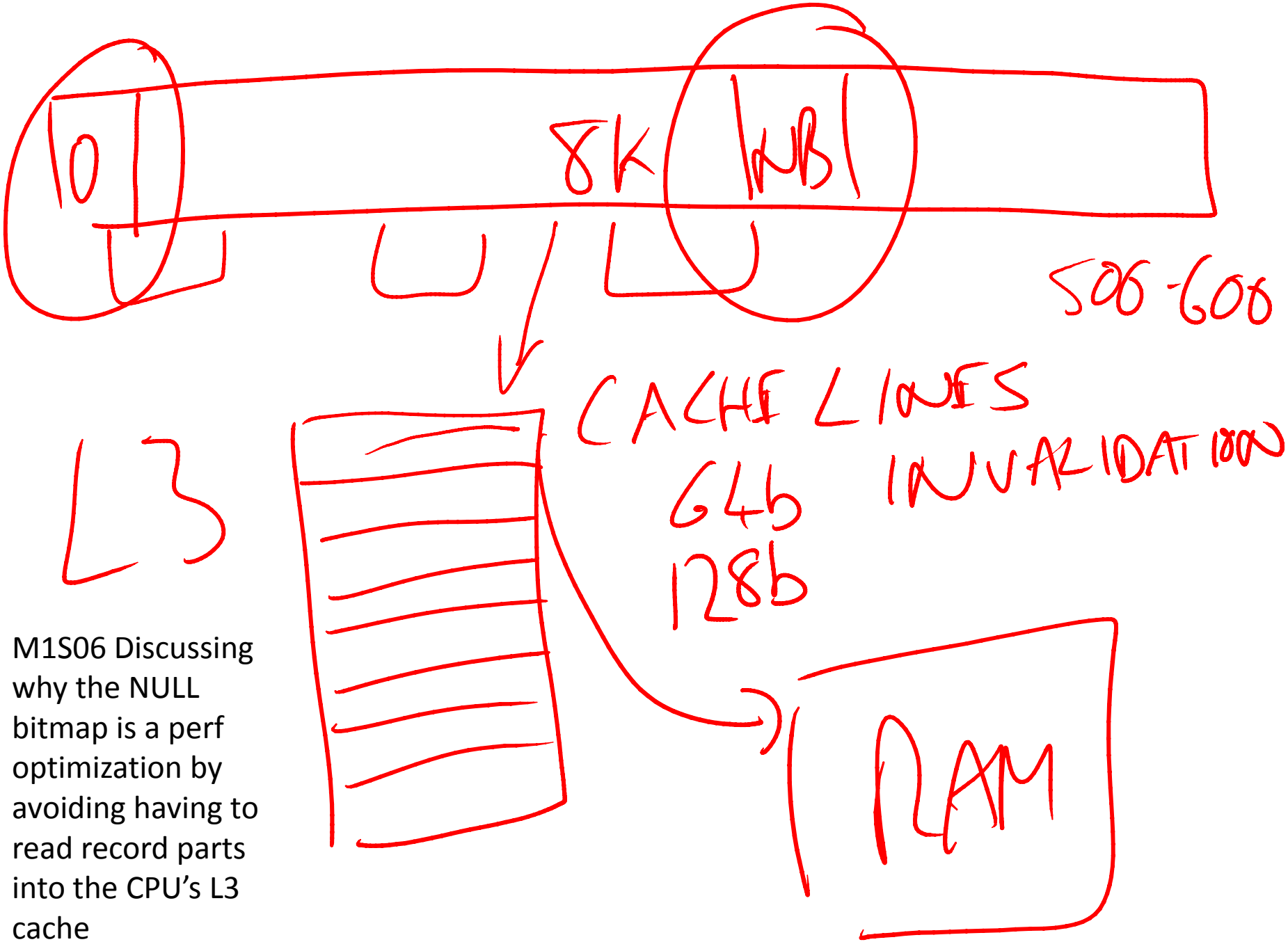
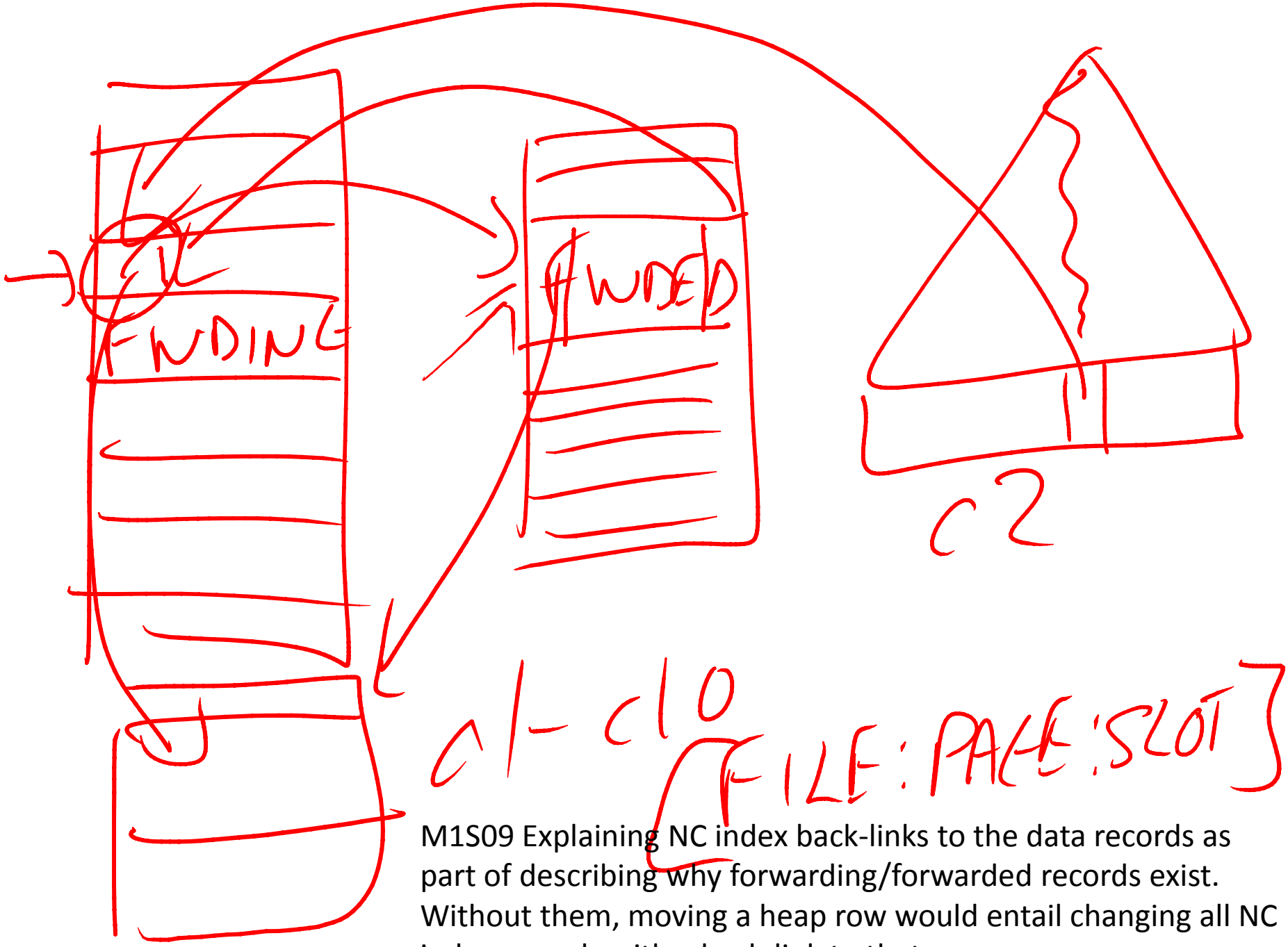
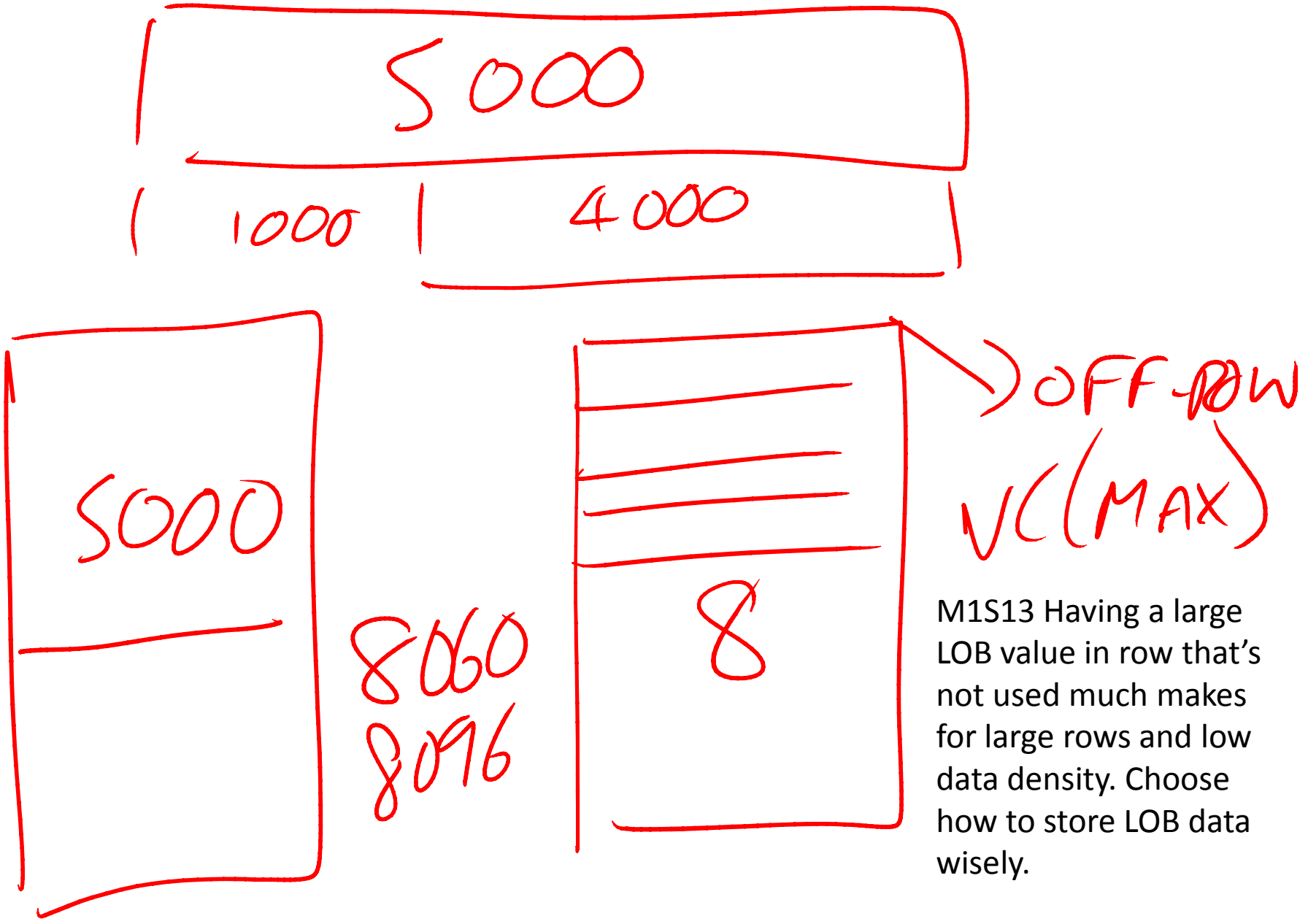


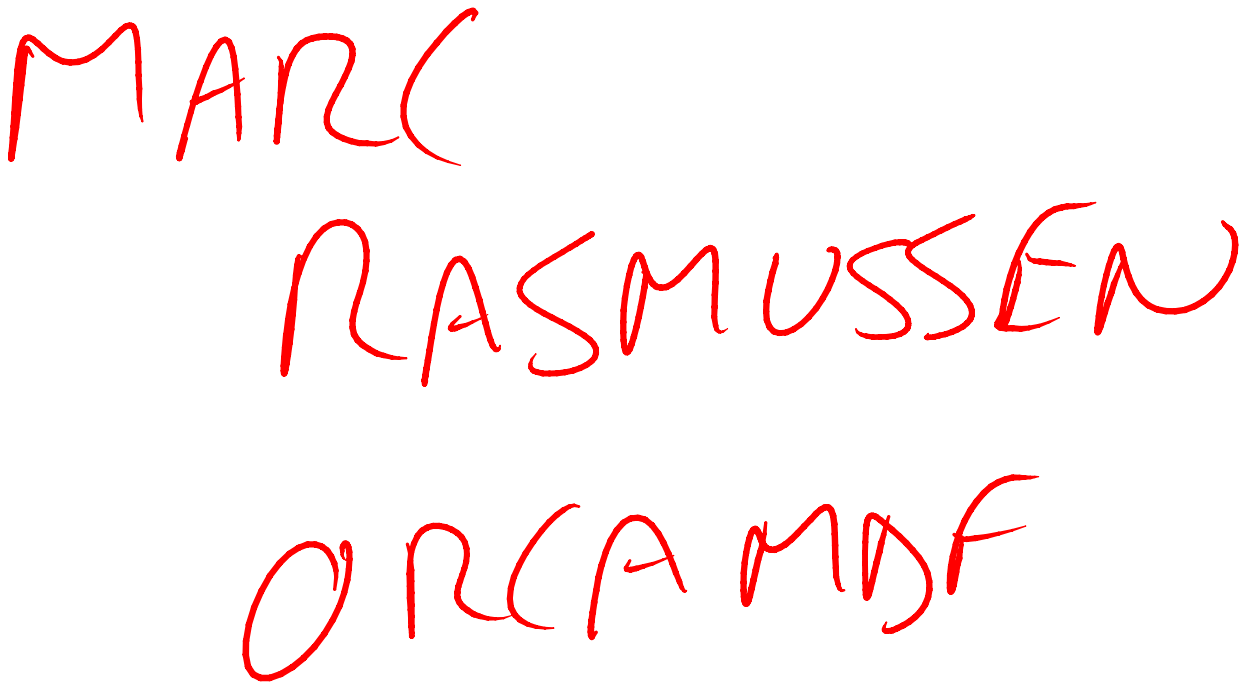
M1S06 Discussing why the NULL bitmap is a perf optimization by avoiding having to read record parts into the CPU's L3 cache



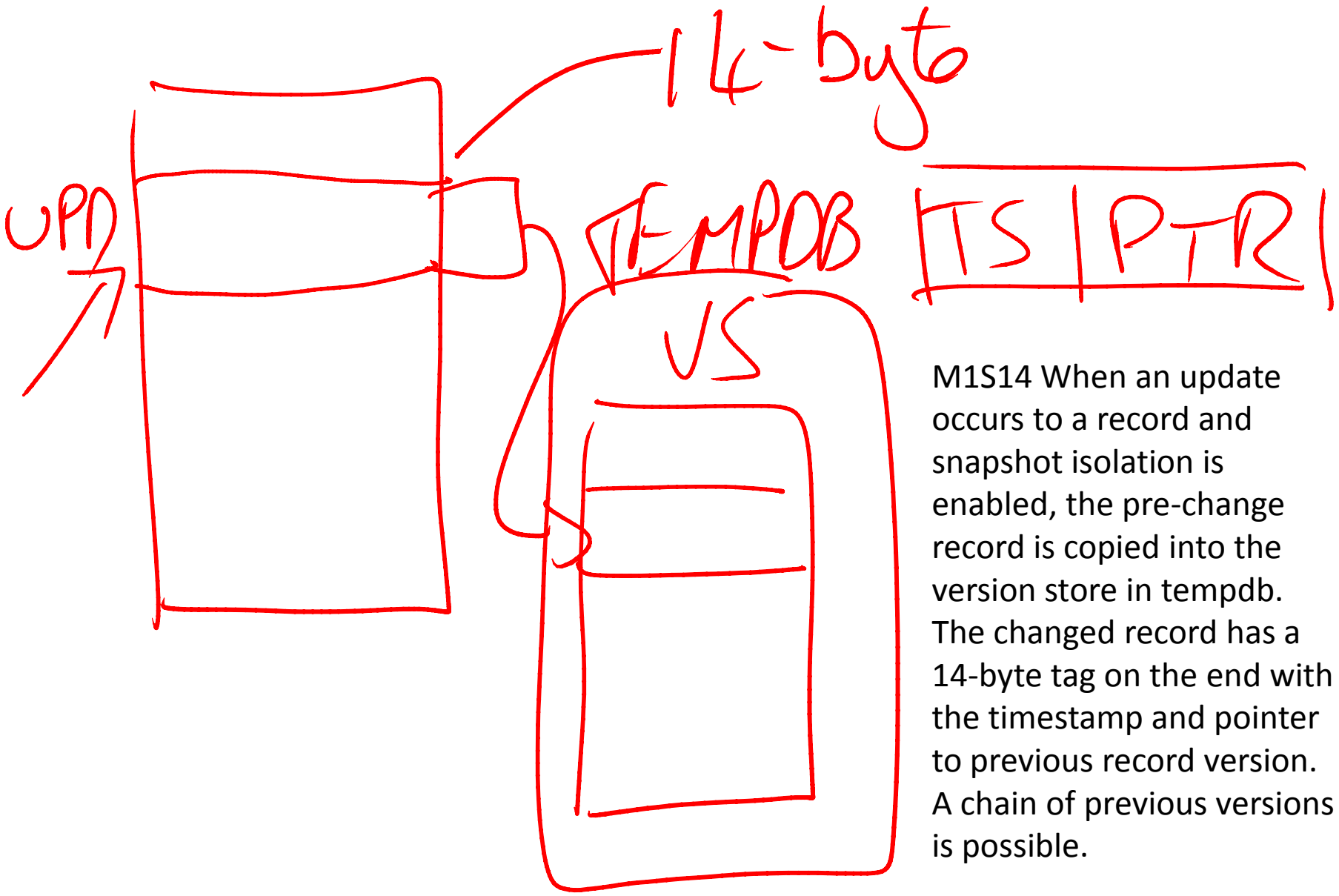
M1S09 Explaining NC index back-links to the data records as part of describing why forwarding/forwarded records exist. Without them, moving a heap row would entail changing all NC index records with a back-link to that row.



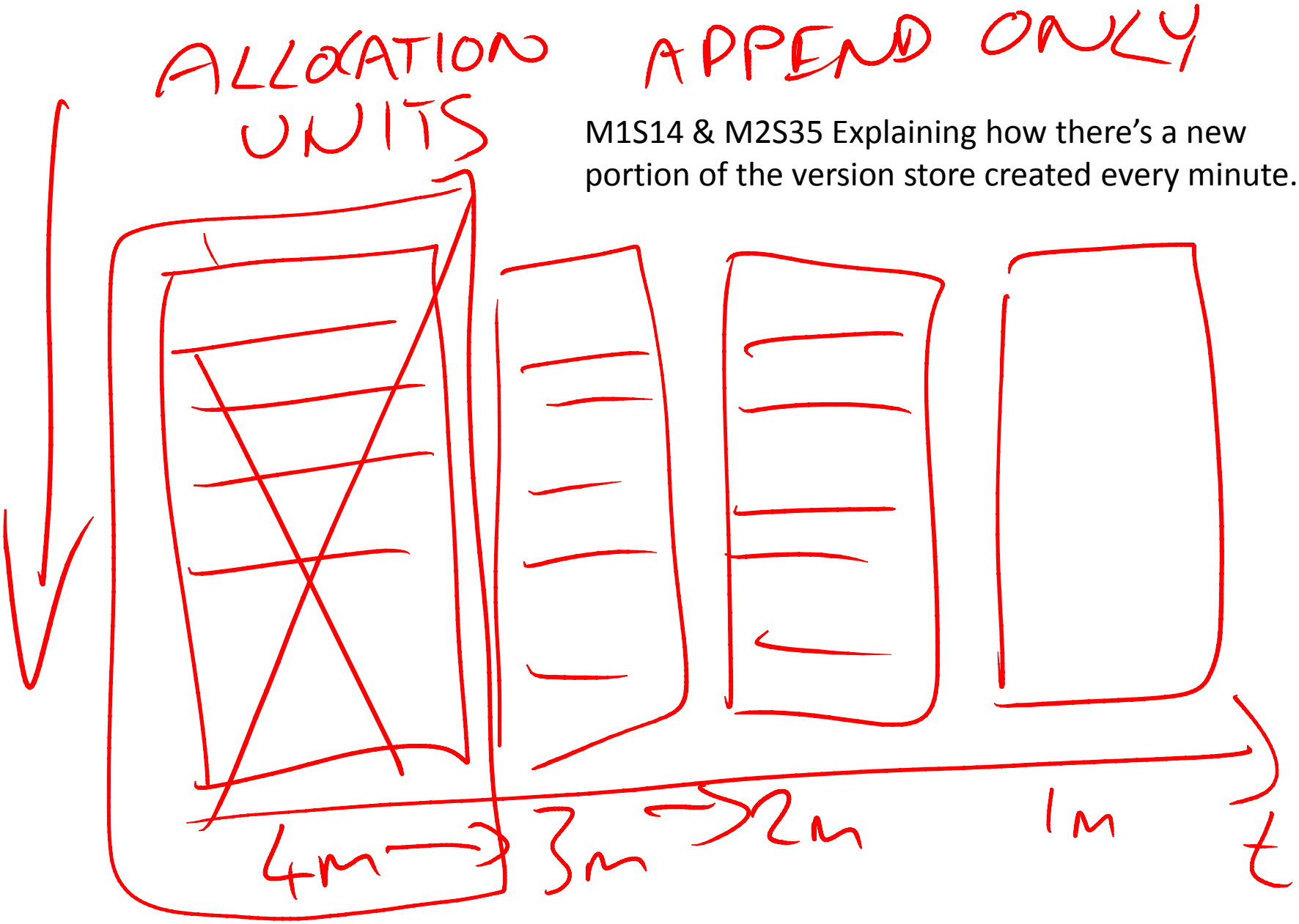
M1S13 Having a large LOB value in row that's not used much makes for large rows and low data density. Choose how to store LOB data wisely.



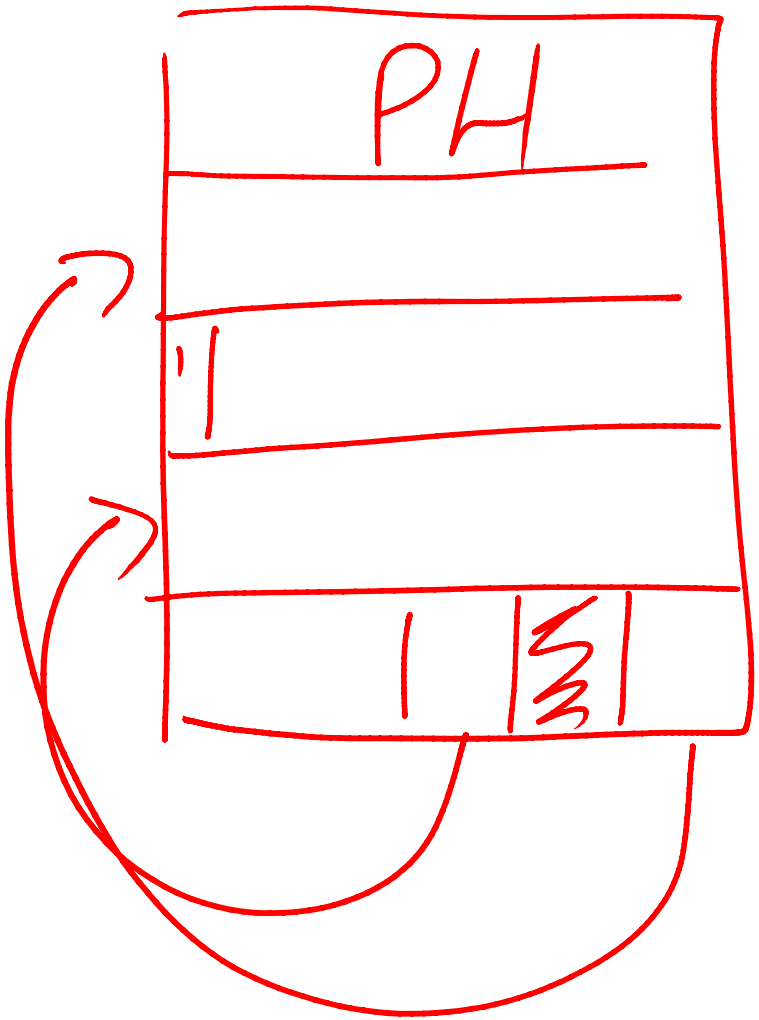
Marc Rasmussen has a cool tool called OrcaMDF that allows you to look around data file structures.



M1S14 When an update occurs to a record and snapshot isolation is enabled, the pre-change record is copied into the version store in tempdb. The changed record has a 14-byte tag on the end with the timestamp and pointer to previous record version. A chain of previous versions is possible.



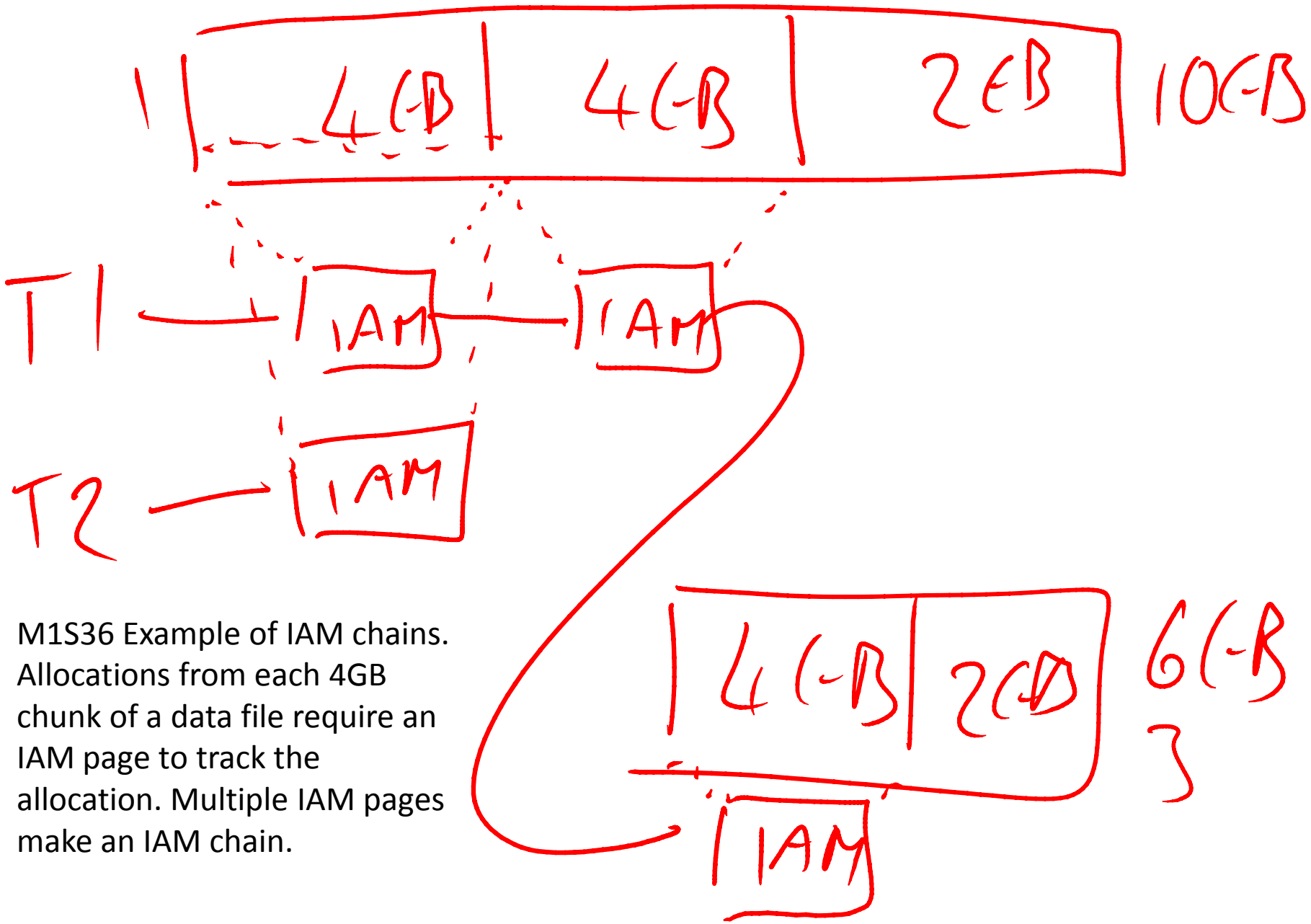
M1S14 & M2S35 Explaining how there's a new portion of the version store created every minute.



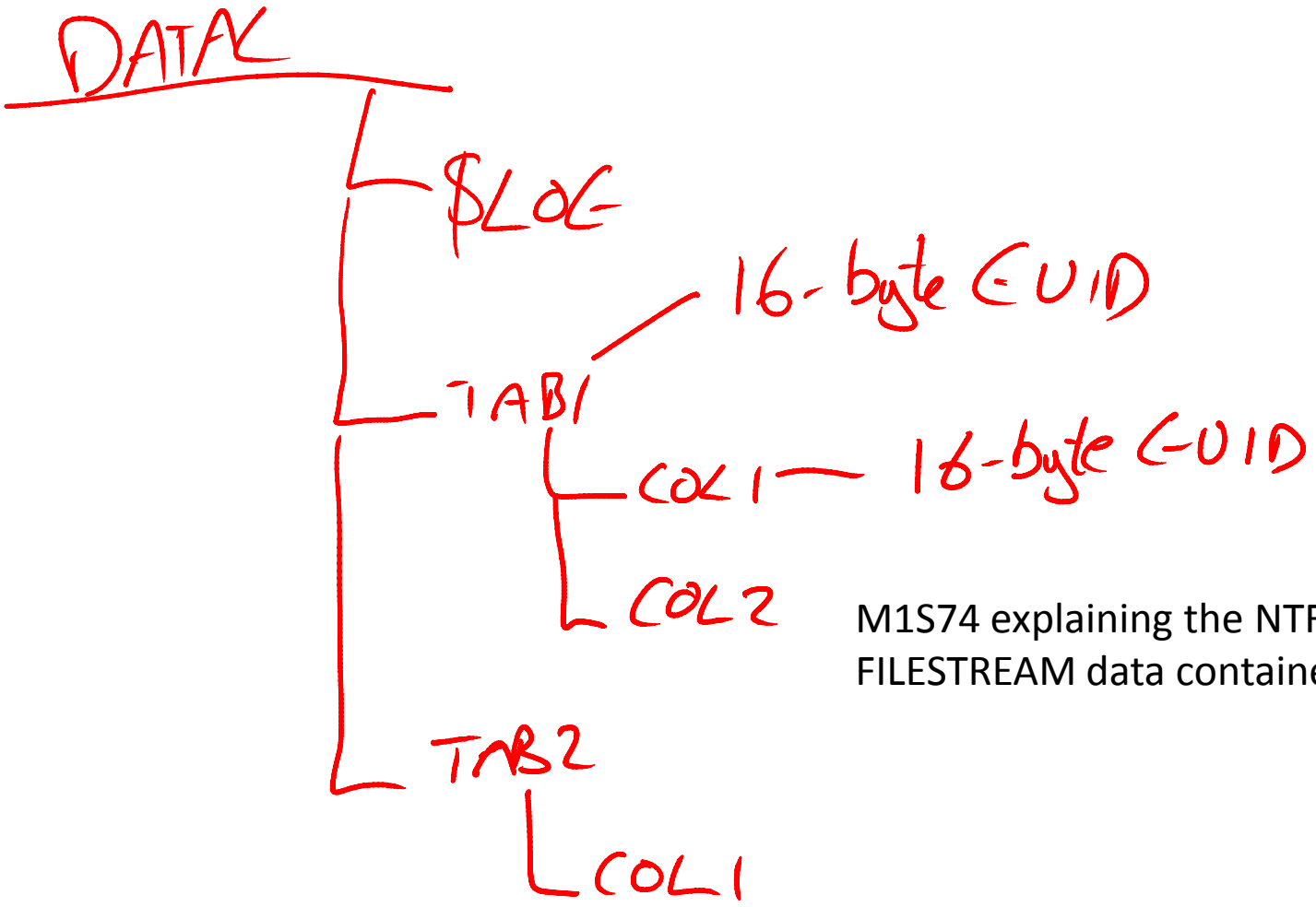
M1S15 When a record is deleted, it's marked as a ghost record. The ghost cleanup task does not physically delete the records – it removes the slot array entry that points to the record on the page. I.e. the record is unmarked as being a record and the area on the page is now free space – that just happens to have bits and bytes that used to be a valid record.

```
Select * from fn_dblog(NULL, NULL)
where [Operation]
= 'LOP_EXPUNGE_GHOST'
```

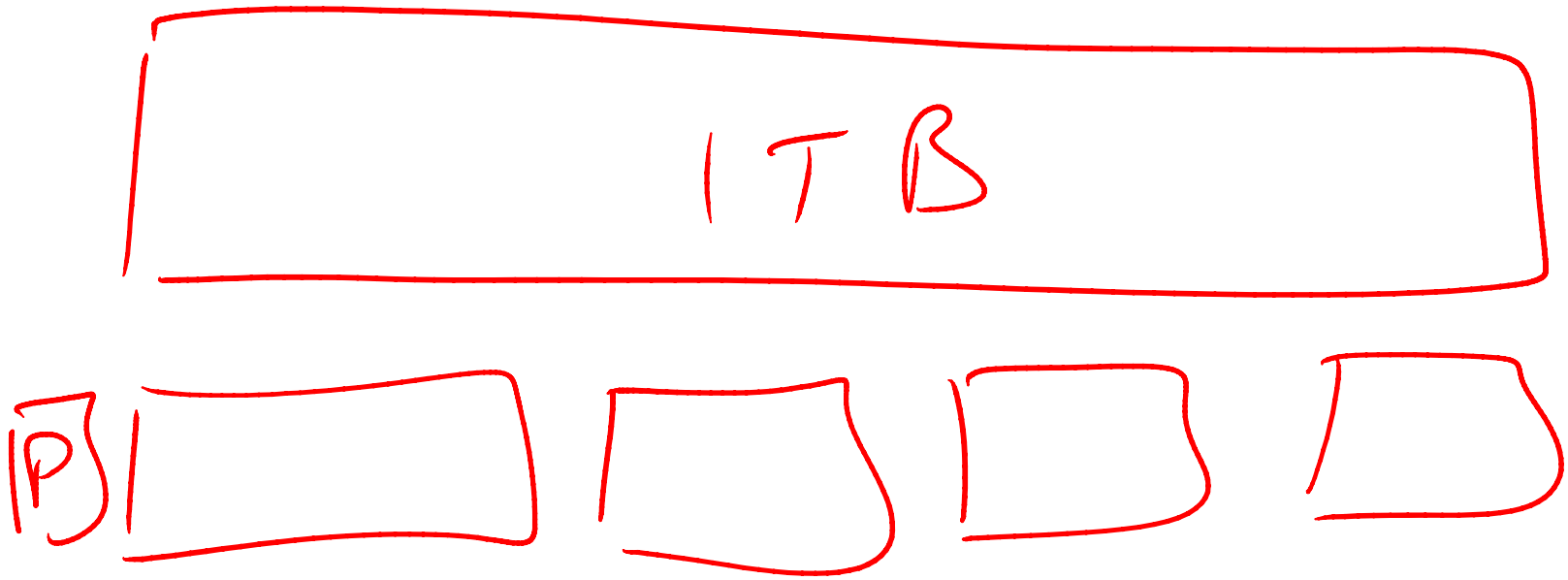
How to see if the ghost cleanup task is doing anything. The log record name is actually LOP_EXPUNGE_ROWS



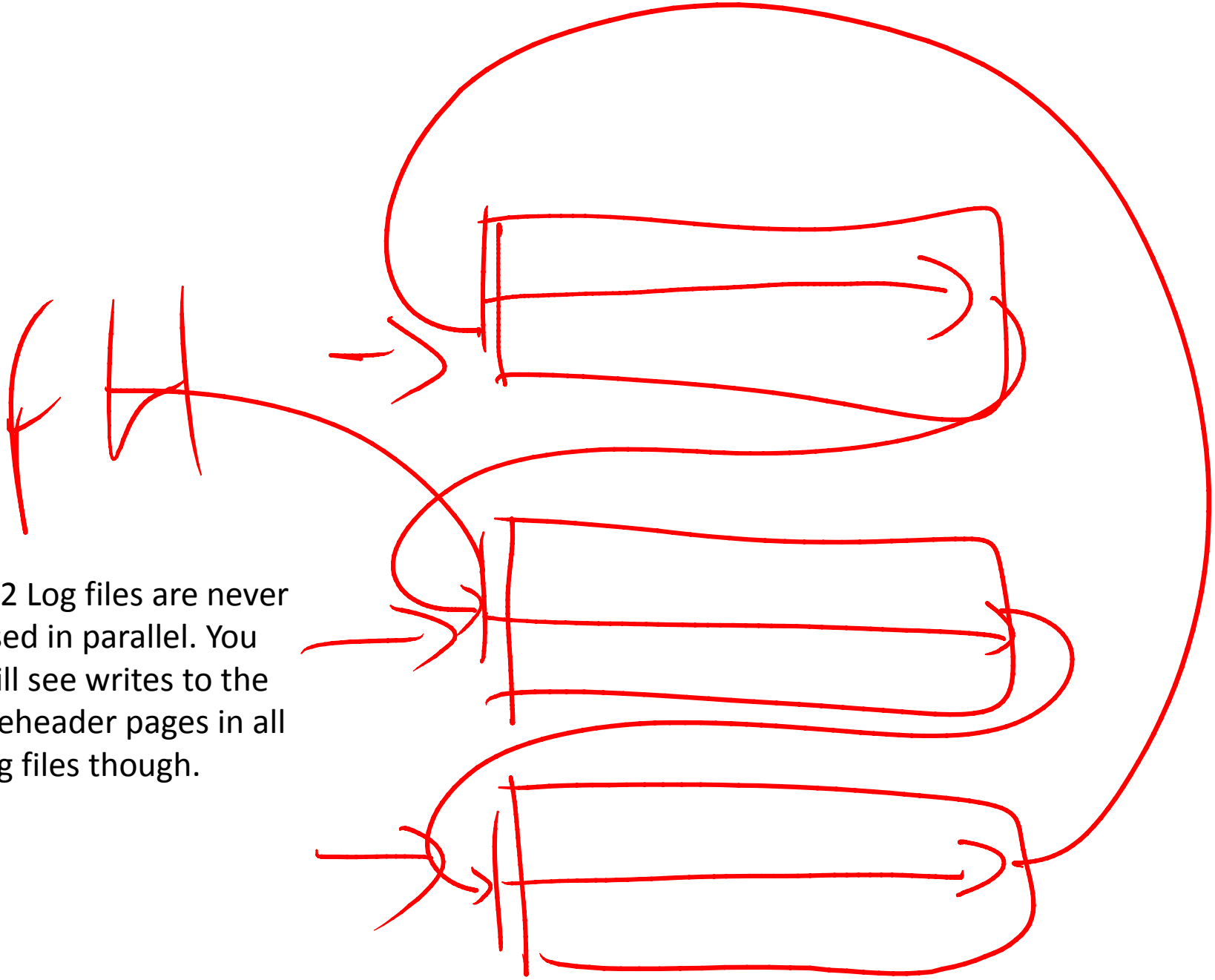
M1S36 Example of IAM chains. Allocations from each 4GB chunk of a data file require an IAM page to track the allocation. Multiple IAM pages make an IAM chain.



M1S74 explaining the NTFS structure of a FILESTREAM data container



M2S7 Splitting a large database up into multiple filegroups can enable you to do small, targeted restores or even a partial restore from scratch – in the event that the database is damaged or destroyed.



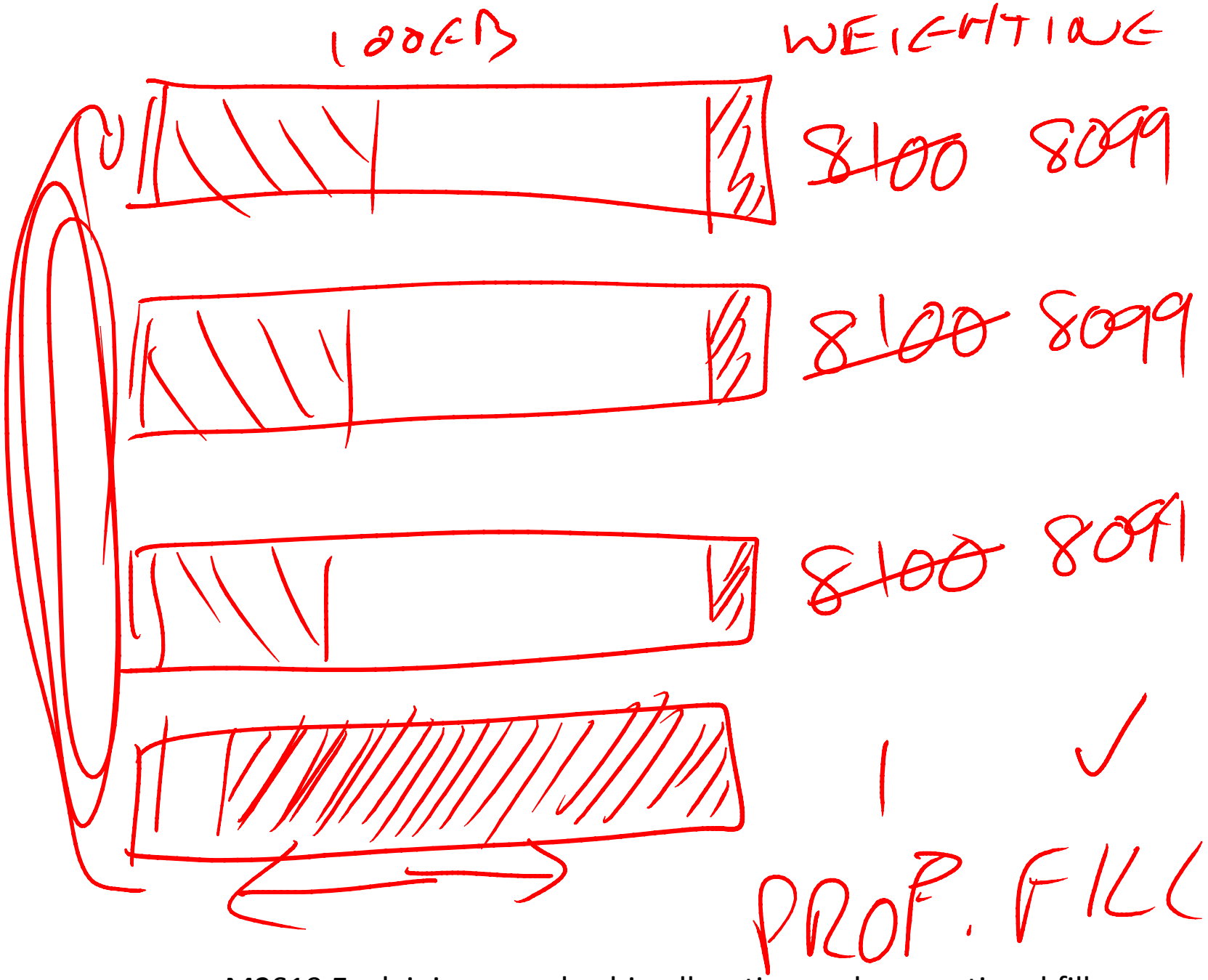
M2 Log files are never used in parallel. You will see writes to the fileheader pages in all log files though.

JIMMY MAY

MSDN

sys.dm_io_virtual_file_stats

M1S8 Google for Jimmy May to find his blog – he wrote the disk partition alignment whitepaper. Look in sys.dm_io_virtual_file_stats to see the average read/write latencies for files (google for it and me for a script to do it for you)

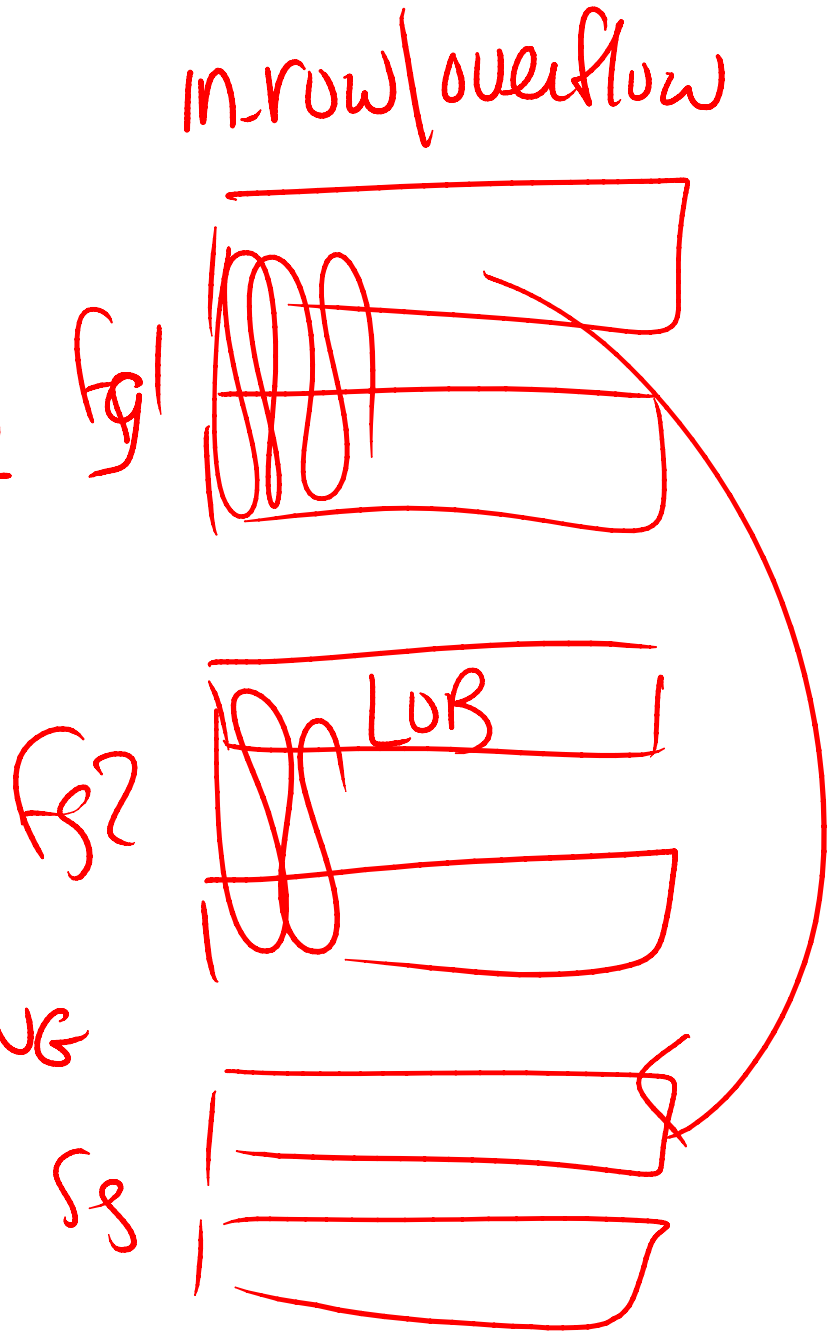


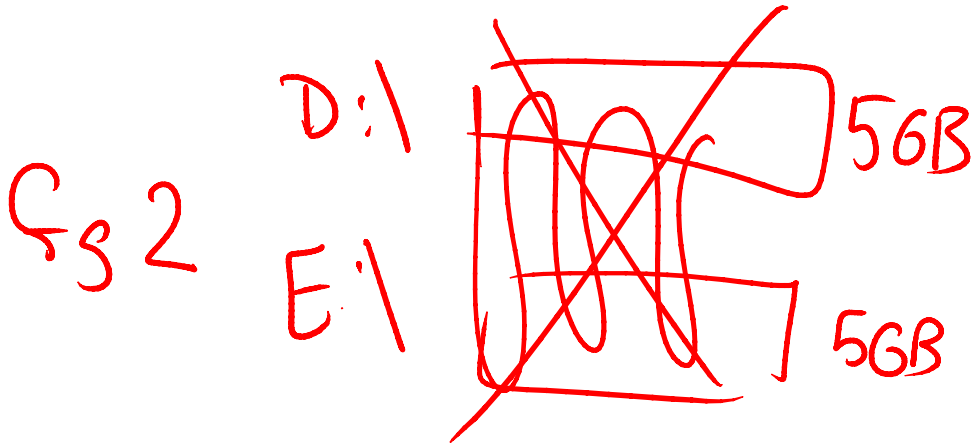
M2S19 Explaining round-robin allocation and proportional fill

CR T
on fg1
text image_on fg2

M2 Kimberly explaining the method she blogged about (on our SQL Mag blog) to allow LOB to be moved.

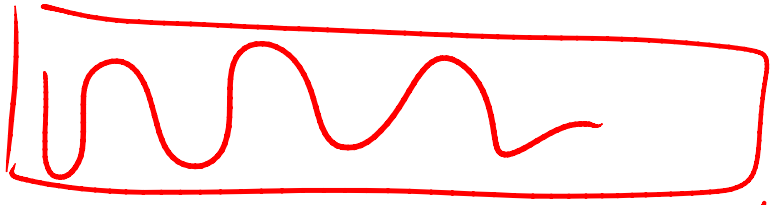
CR w/DROP_EXISTING
of CL on fg3



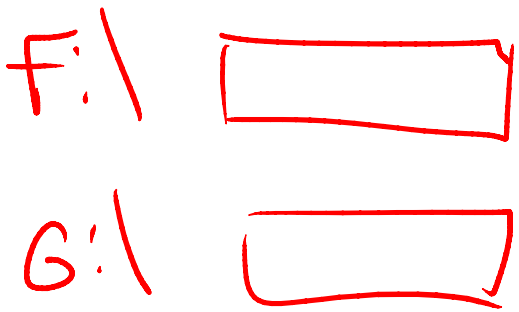


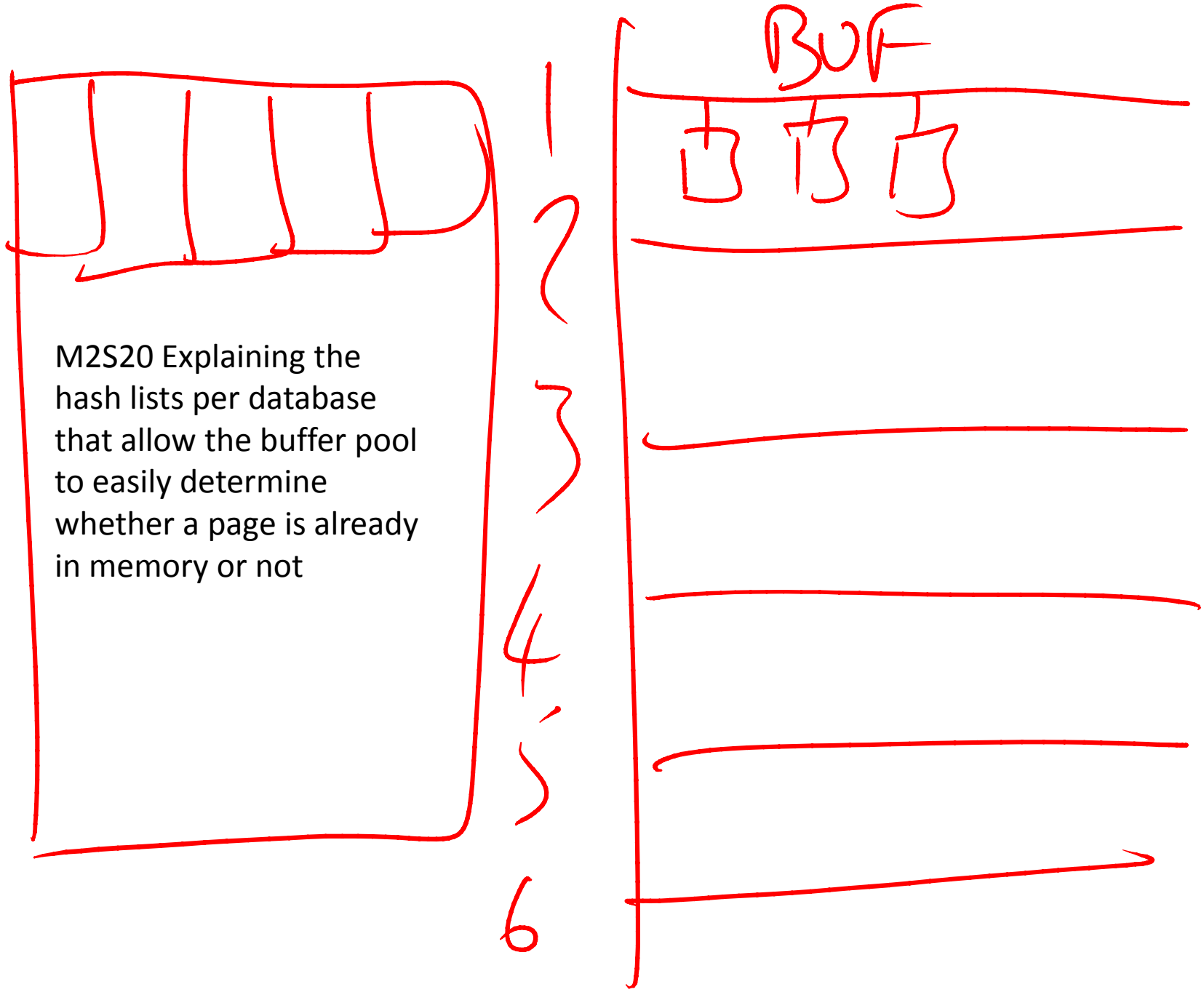
M2 Kimberly explaining the method she blogged about (on our SQL Mag blog) to allow LOB to be moved.

Want LOB ① Temp ADD file 10GB on T:\temp
② Shrink file on Both D:\ & E:\

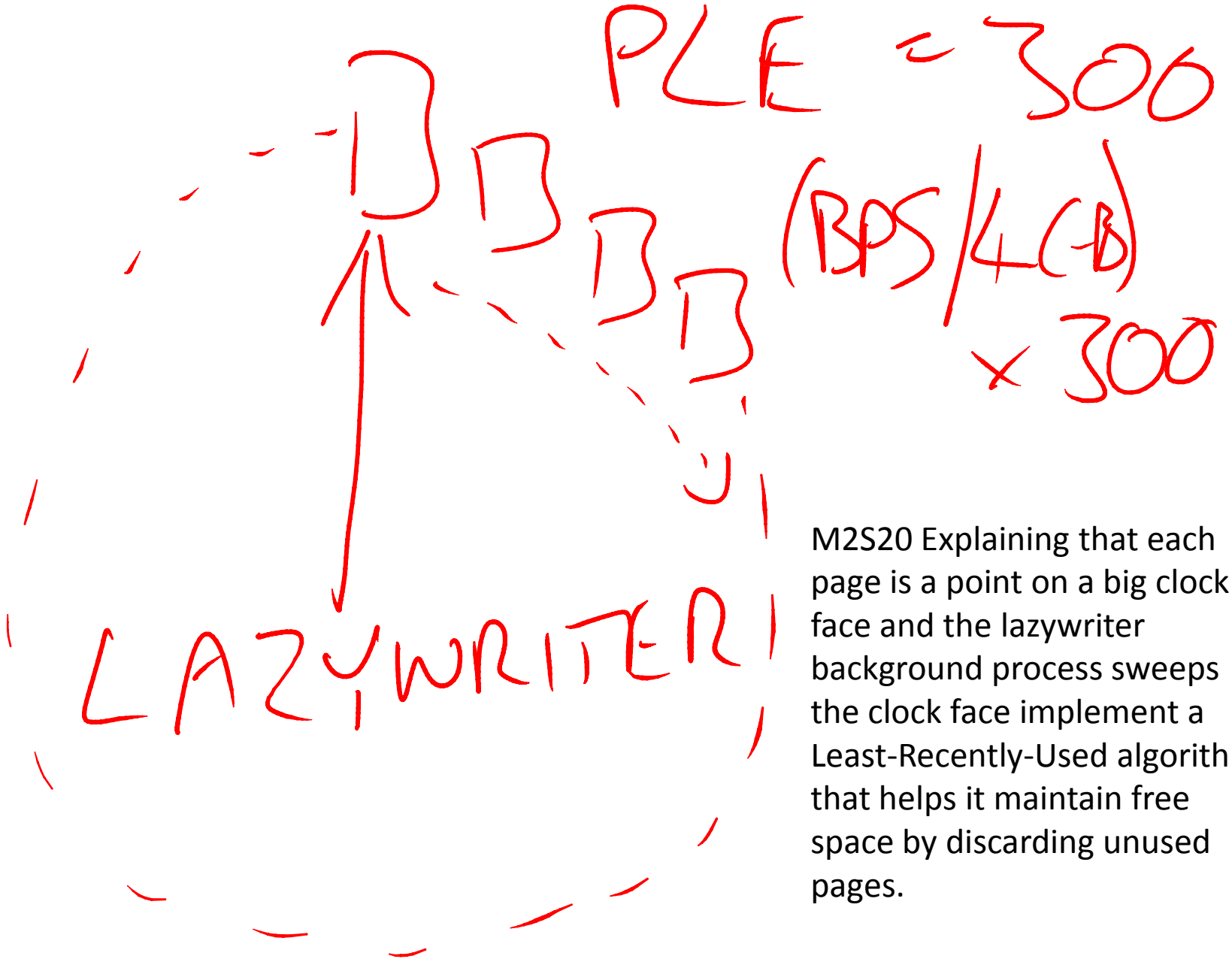


Shrink again



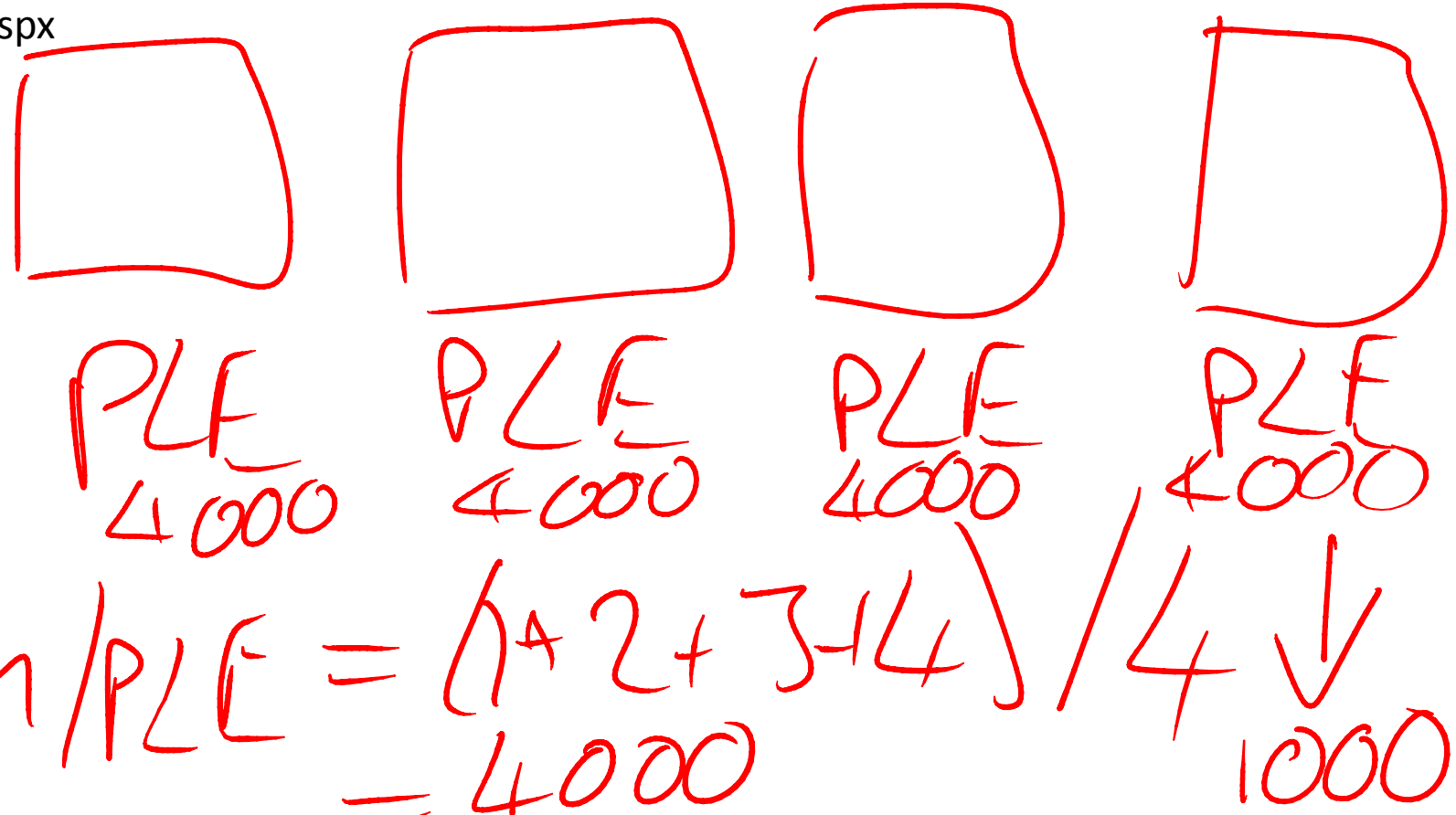


M2S20 Explaining the hash lists per database that allow the buffer pool to easily determine whether a page is already in memory or not



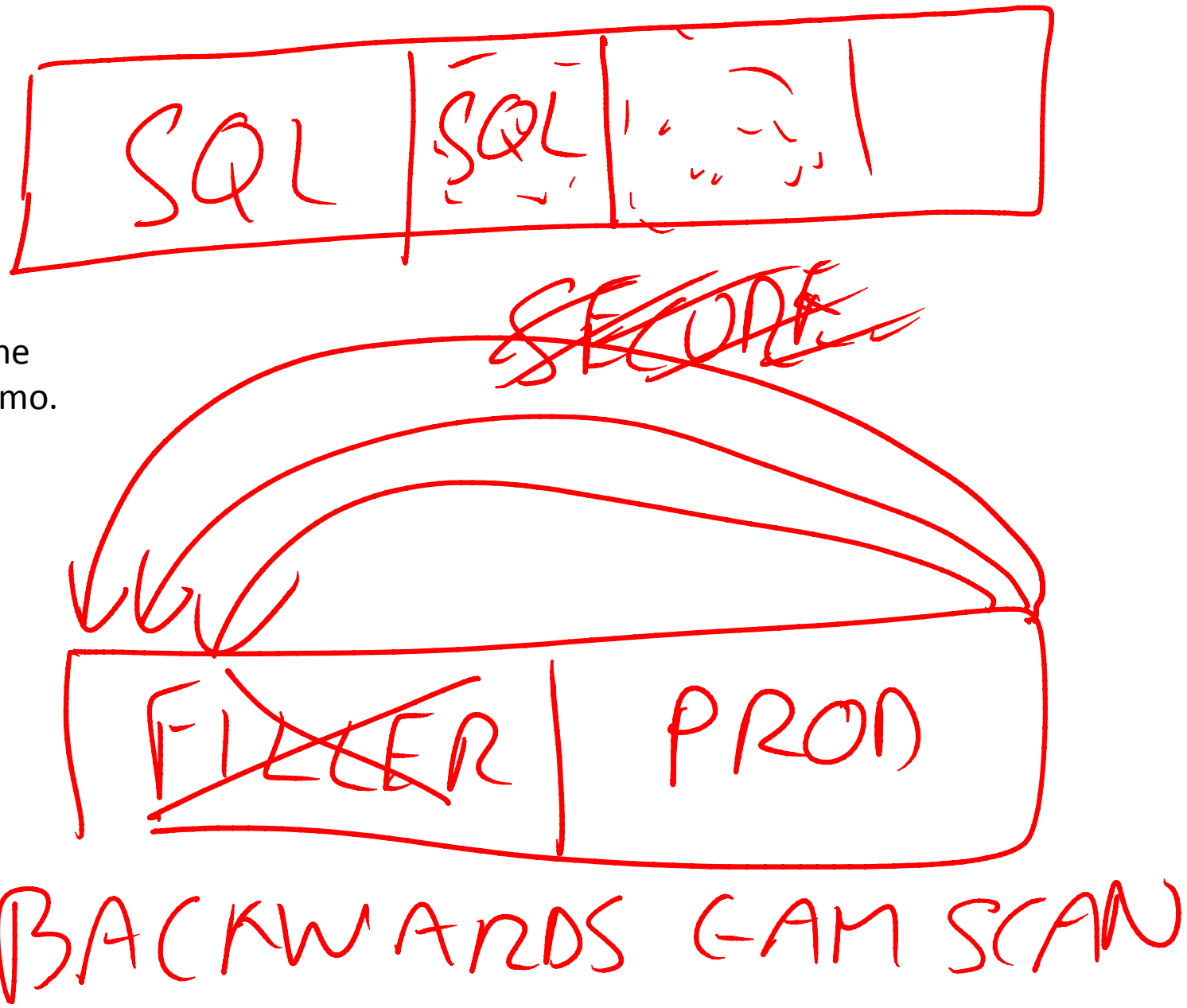
M2S20 Explaining that each page is a point on a big clock face and the lazywriter background process sweeps the clock face implement a Least-Recently-Used algorithm that helps it maintain free space by discarding unused pages.

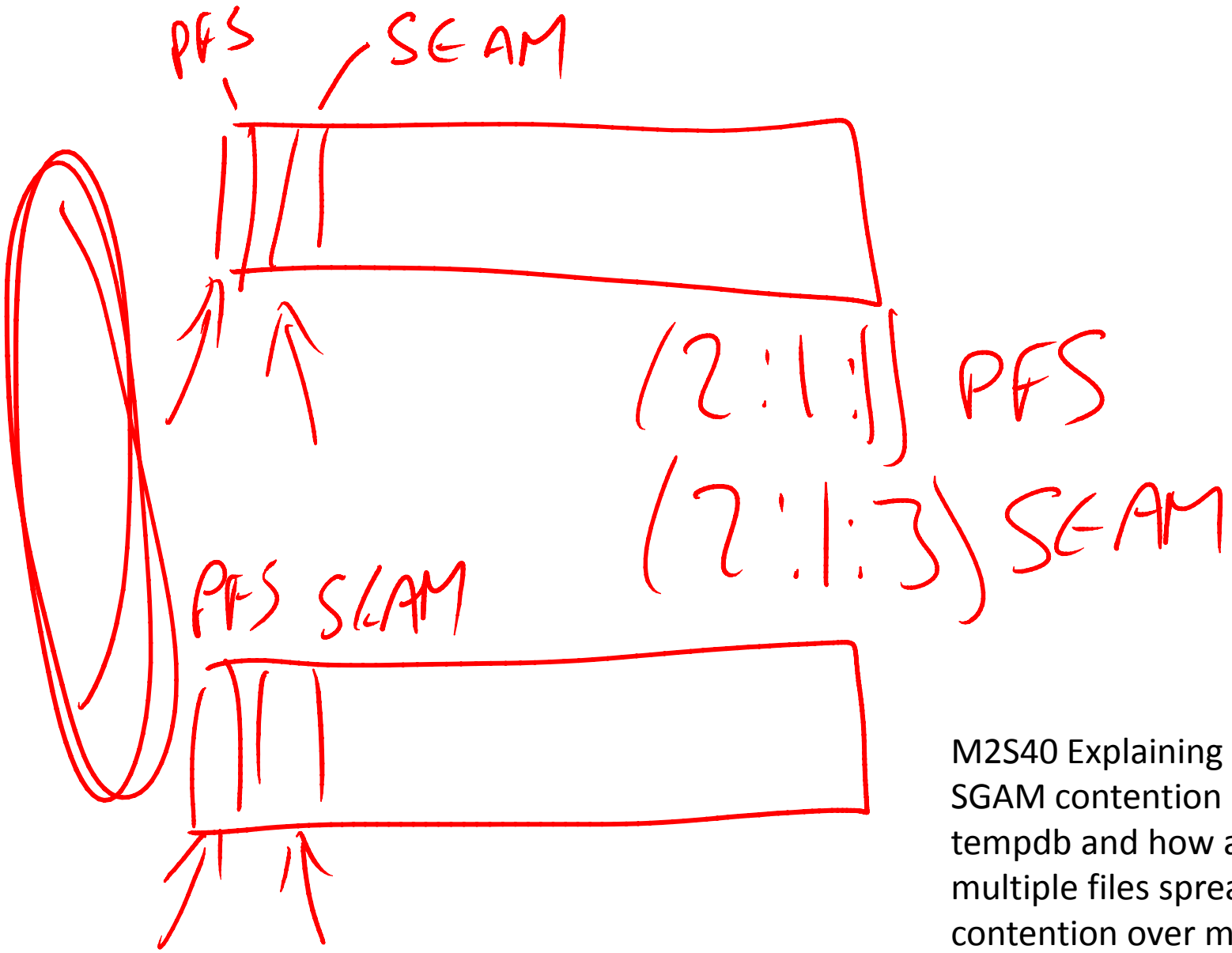
See http://www.sqlskills.com/BLOGS/PAUL/post/Page-Life-Expectancy-isnt-what-you-think.aspx



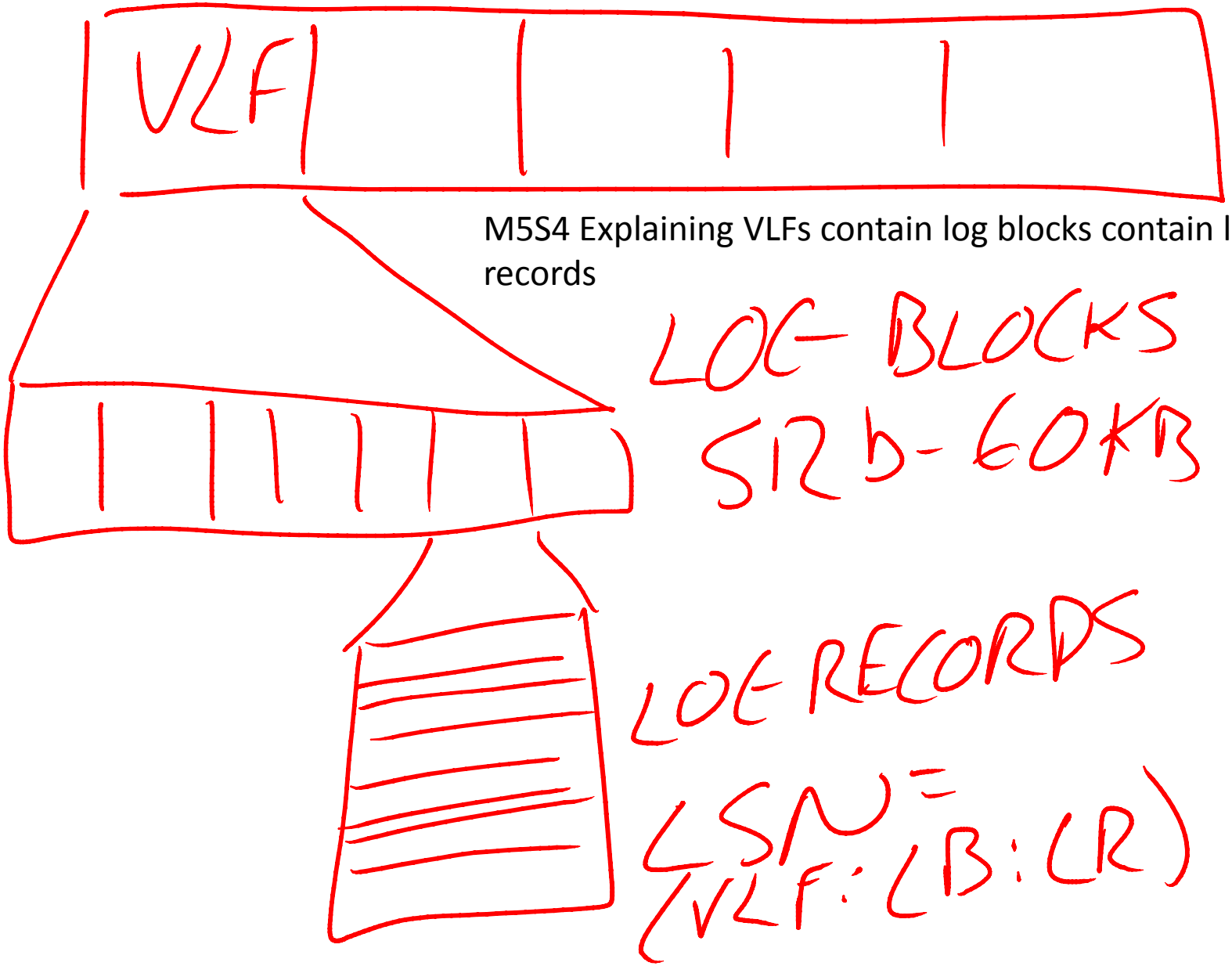
This should be Buffer Node instead of Buffer Partition – my mistake.

M2S28 The shrink demo.

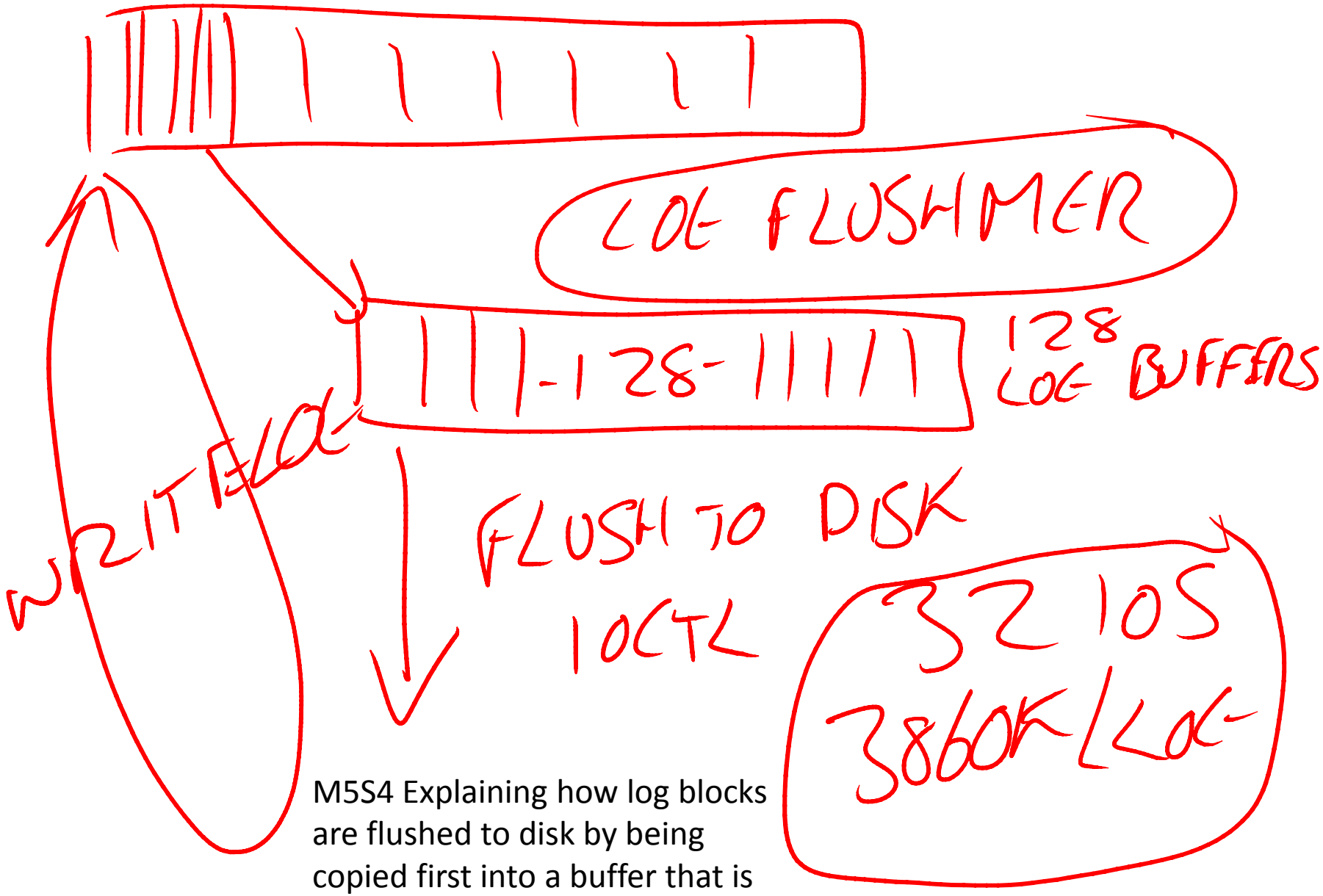




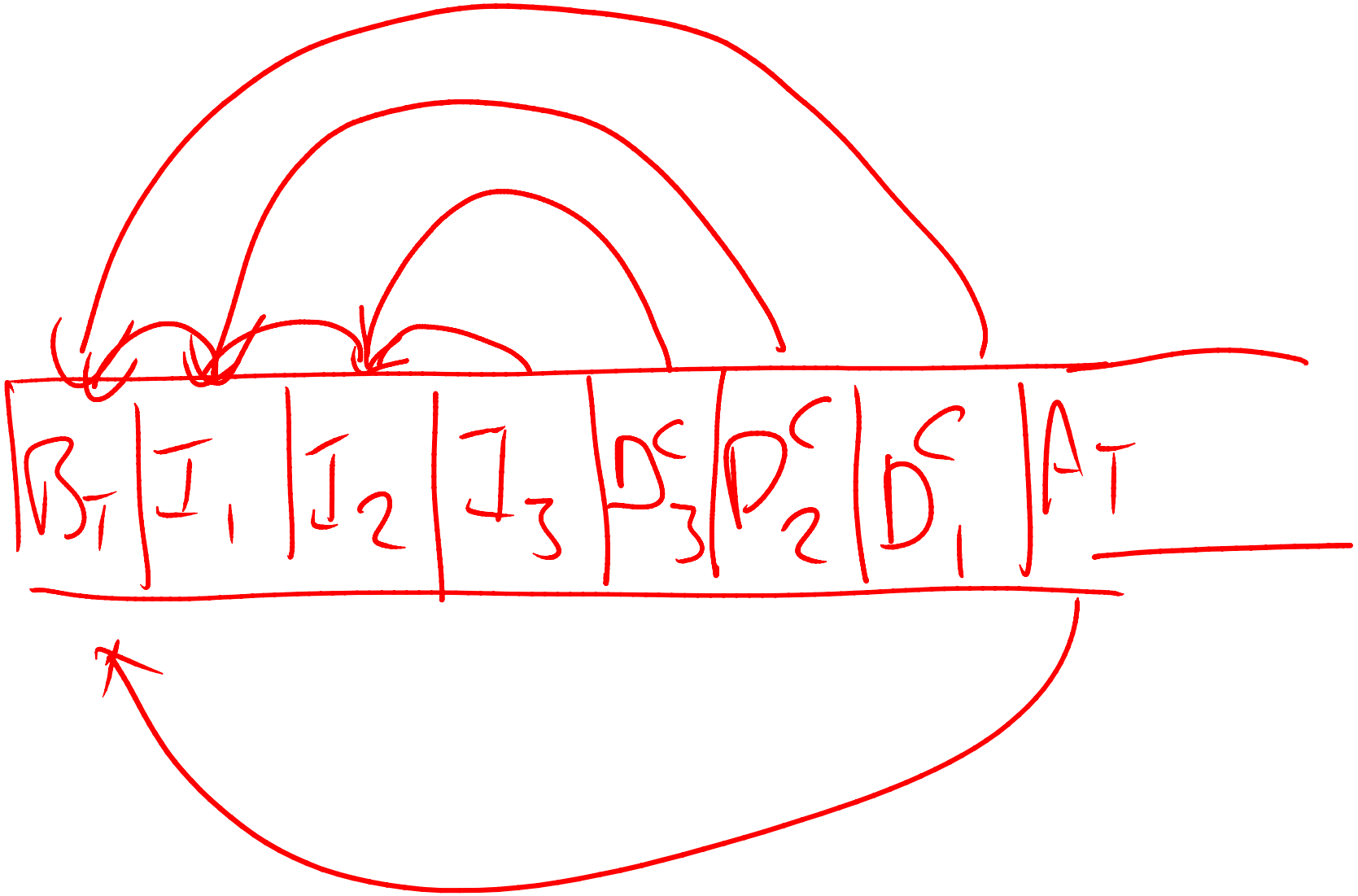
M2S40 Explaining PFS and SGAM contention in tempdb and how adding multiple files spreads the contention over multiple files, thus reducing it.



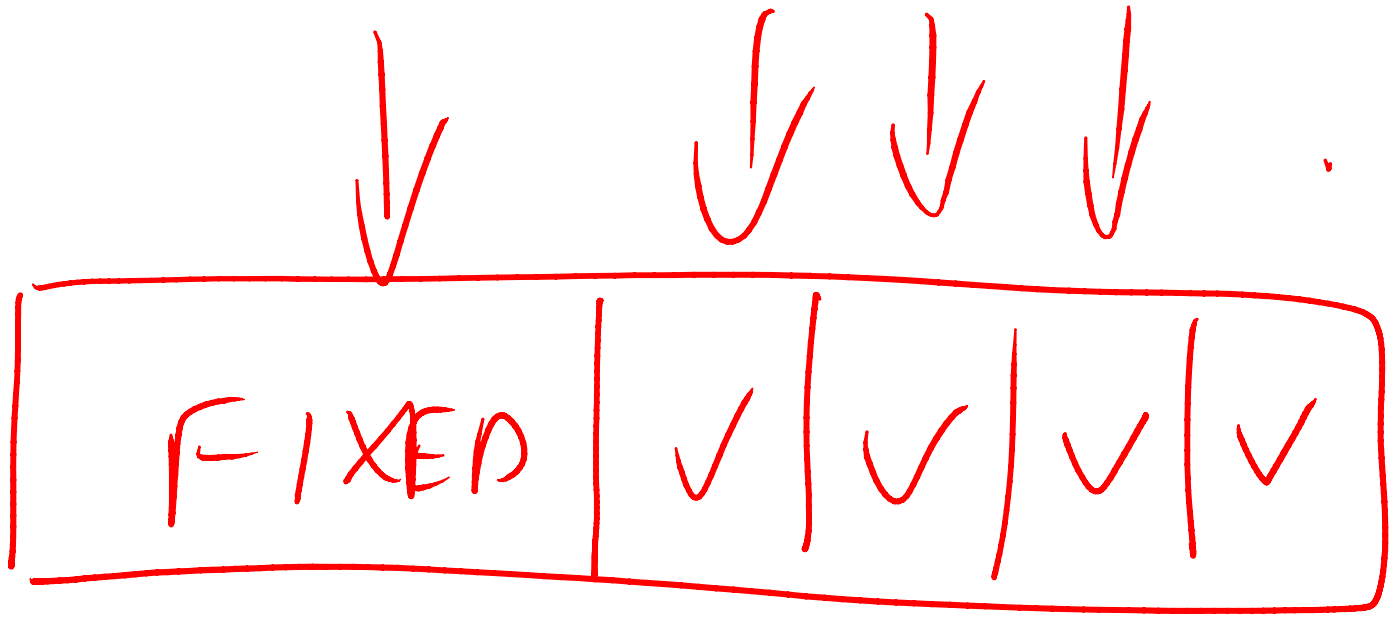
M5S4 Explaining VLFs contain log blocks contain log records



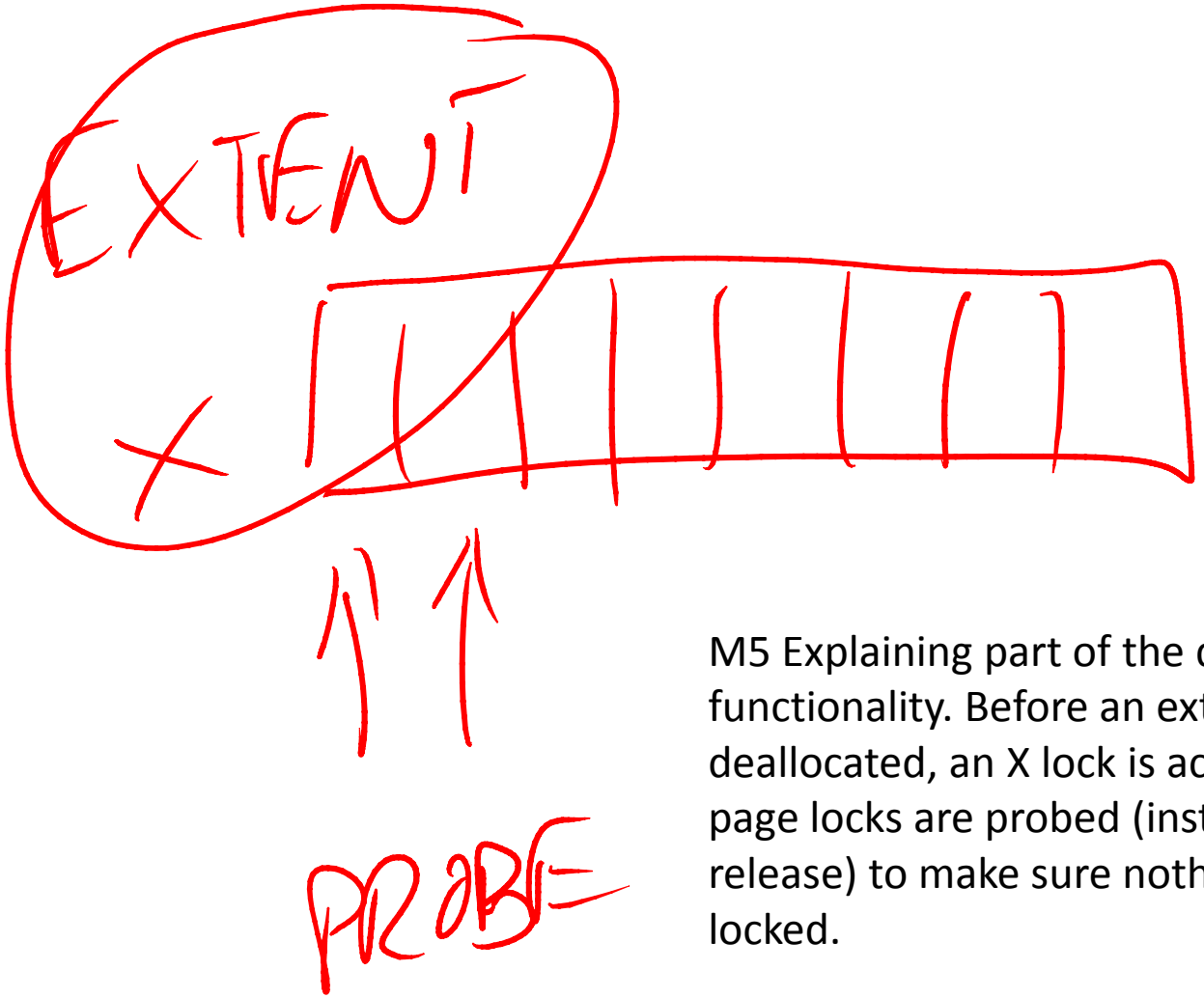
M5S4 Explaining how log blocks are flushed to disk by being copied first into a buffer that is then async written to disk.



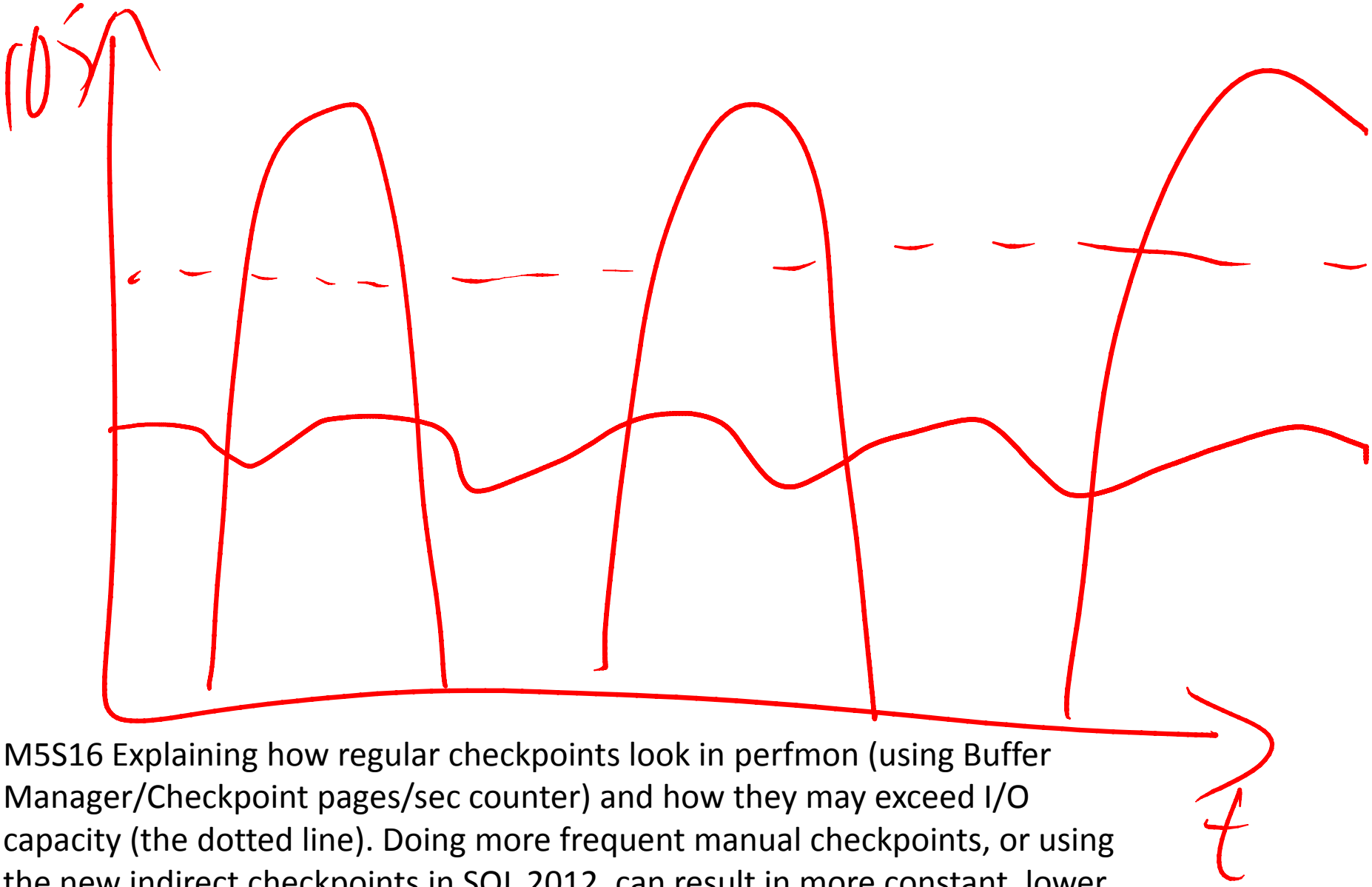
M5S11 Explaining how log records link back to the previous log record in the transaction to allow the transaction to be rolled back by generating compensating actions in reverse order. A compensation log record will link over the log record it is compensating for.



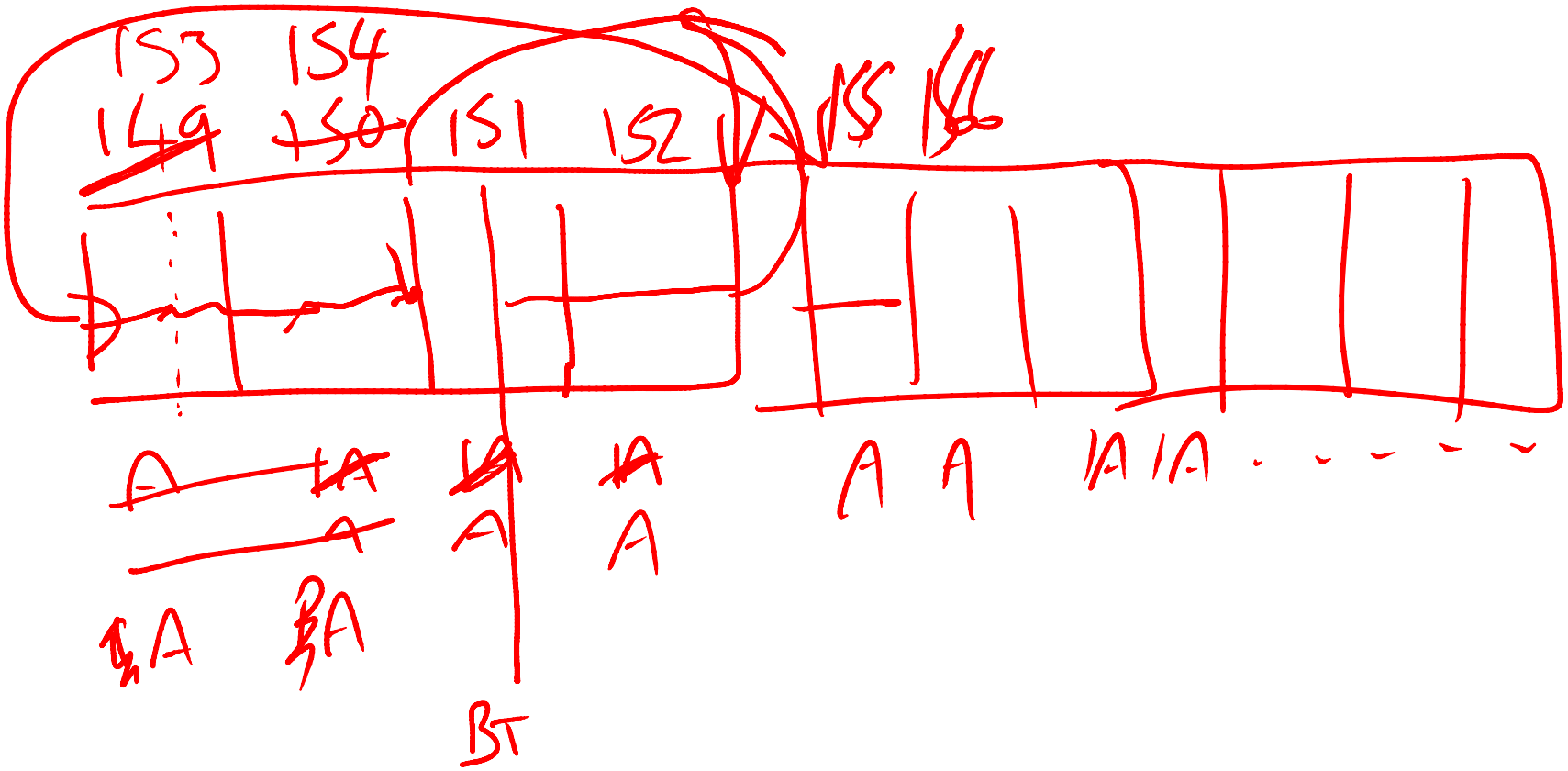
M5S11 Explaining how the Access Methods determines how to log UPDATEs efficiently by splitting the logging by portion of the record. Each variable length column is a portion, as is the fixed-width portion (up to 16 bytes in each log record)



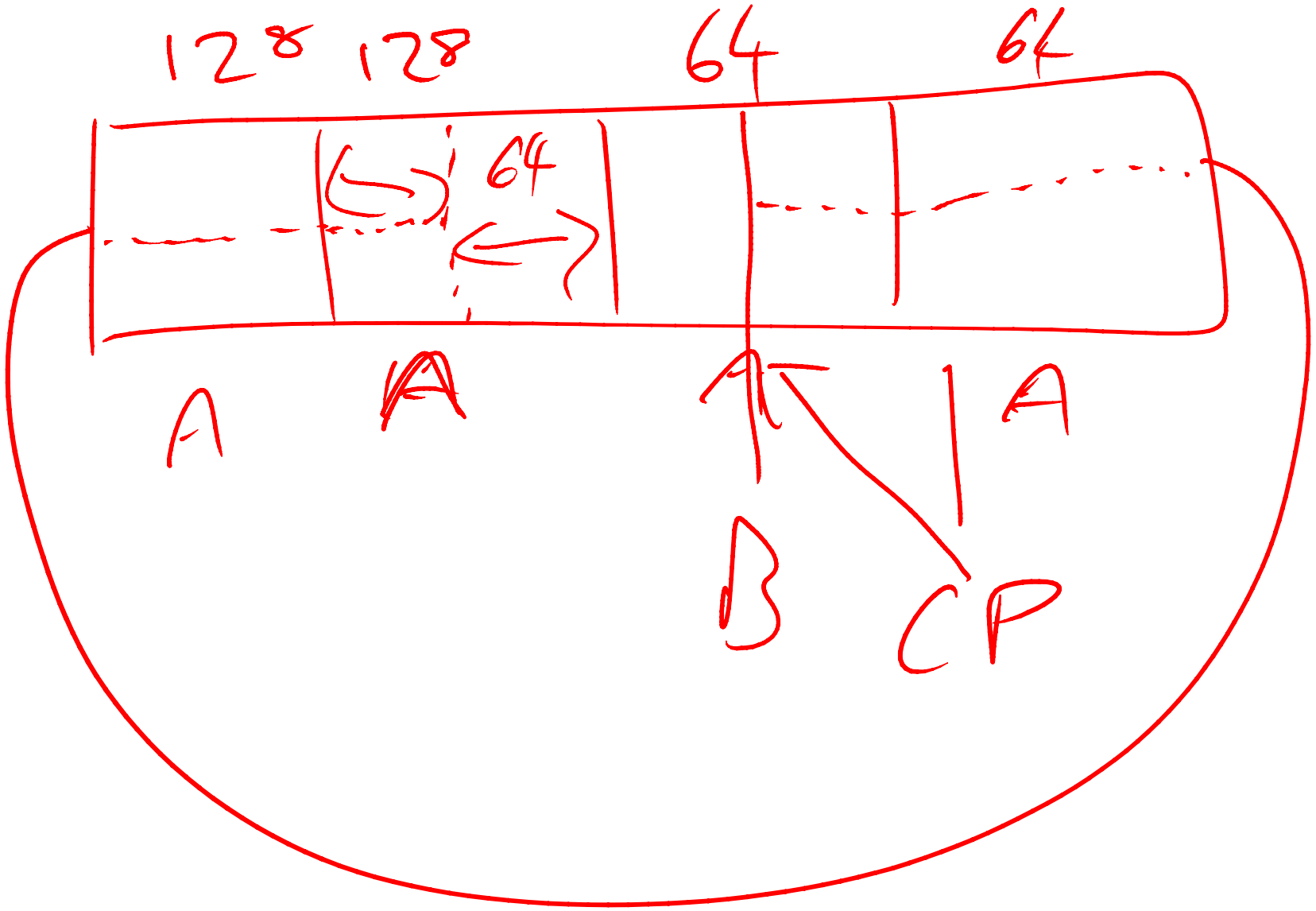
M5 Explaining part of the deferred-drop functionality. Before an extent can be deallocated, an X lock is acquired on it and all 8 page locks are probed (instant acquire X lock and release) to make sure nothing else has them locked.



M5S16 Explaining how regular checkpoints look in perfmon (using Buffer Manager/Checkpoint pages/sec counter) and how they may exceed I/O capacity (the dotted line). Doing more frequent manual checkpoints, or using the new indirect checkpoints in SQL 2012, can result in more constant, lower IO load.



M5S28 Explaining the circular log demo. We discussed log space reservation that happens to ensure the active transactions can roll back without having to grow the log – this accounts for extra log growths. Space reserved in the log does not make a VLF become active – as there are no log records written out, just empty space reserved.



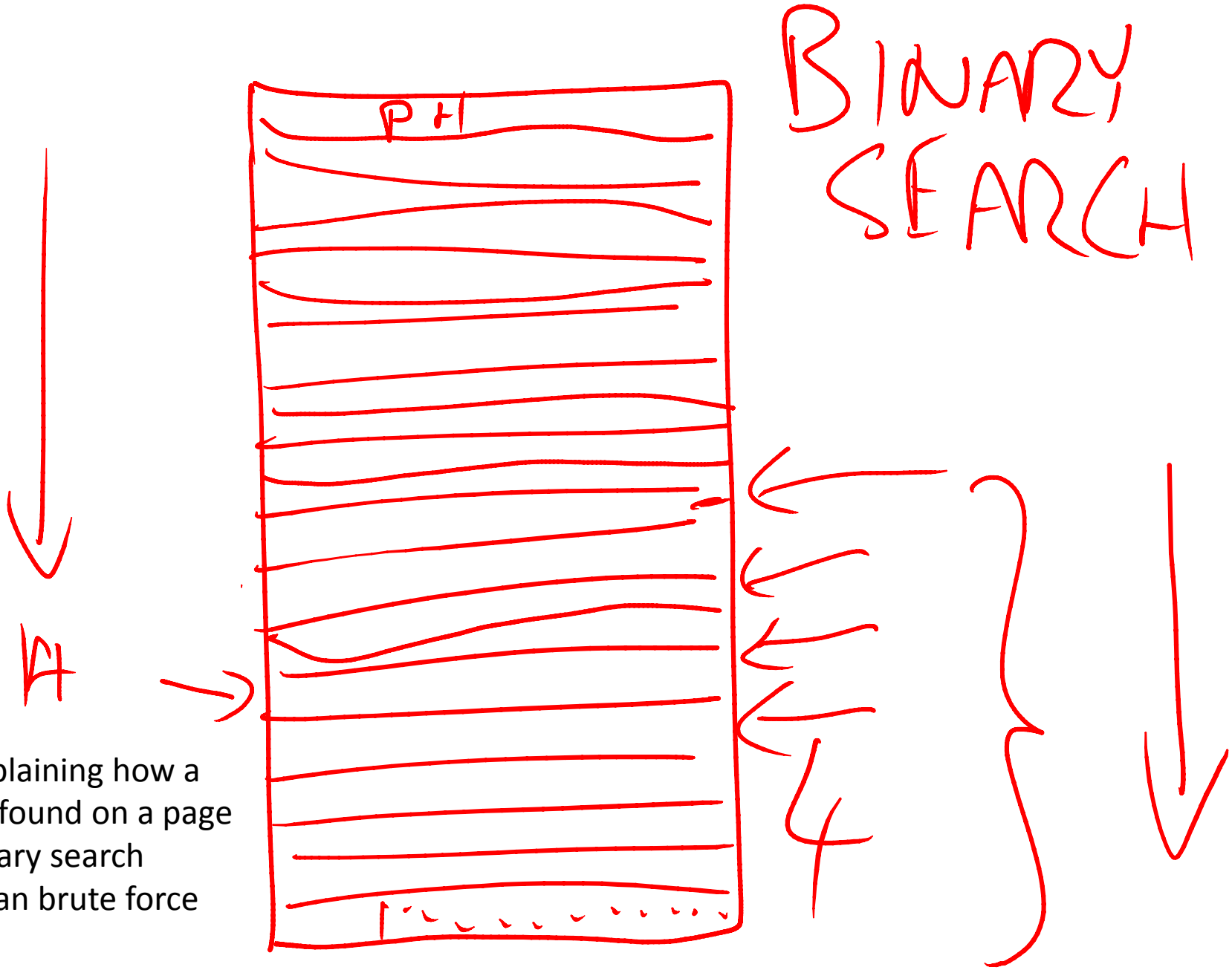
M5S29 Explaining the use of parity bits to help SQL Server determine where the end of the log is during crash recovery.

This slide was missing from your printout. M8S40

Setting FILLFACTOR

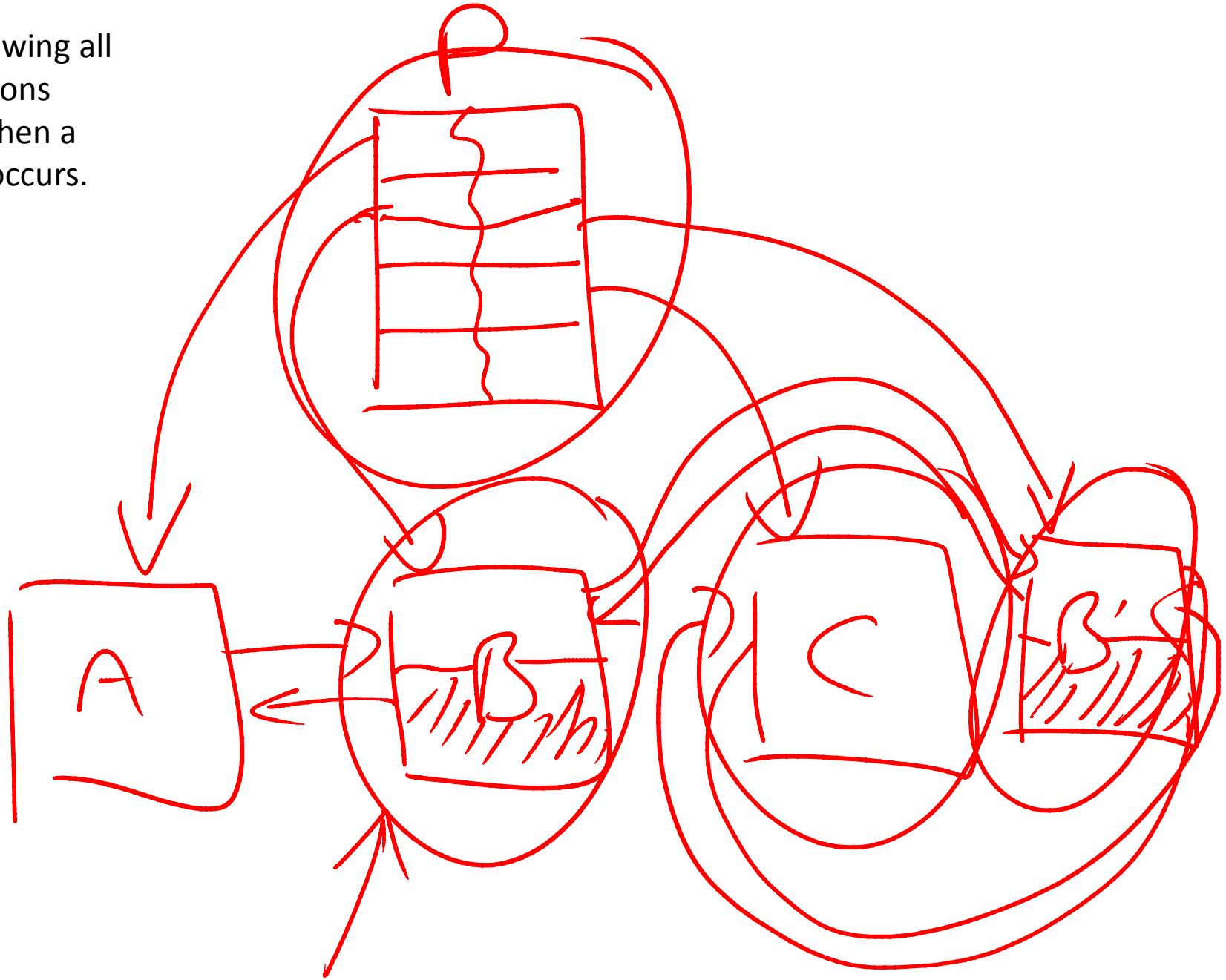
- Can be set when creating or rebuilding an index
  - Stores the fillfactor in the index metadata
- Cannot be set directly with REORGANIZE

- REBUILD and REORGANIZE use the metadata-stored fillfactor, if there is one, otherwise they will use the instance-wide fillfactor
  - Unless a fillfactor is specified on the REBUILD
  - I.e. REBUILD specified fillfactor overrides metadata-stored fillfactor, which overrides instance-wide fillfactor
- Fillfactor is only set in index metadata by a CREATE or REBUILD



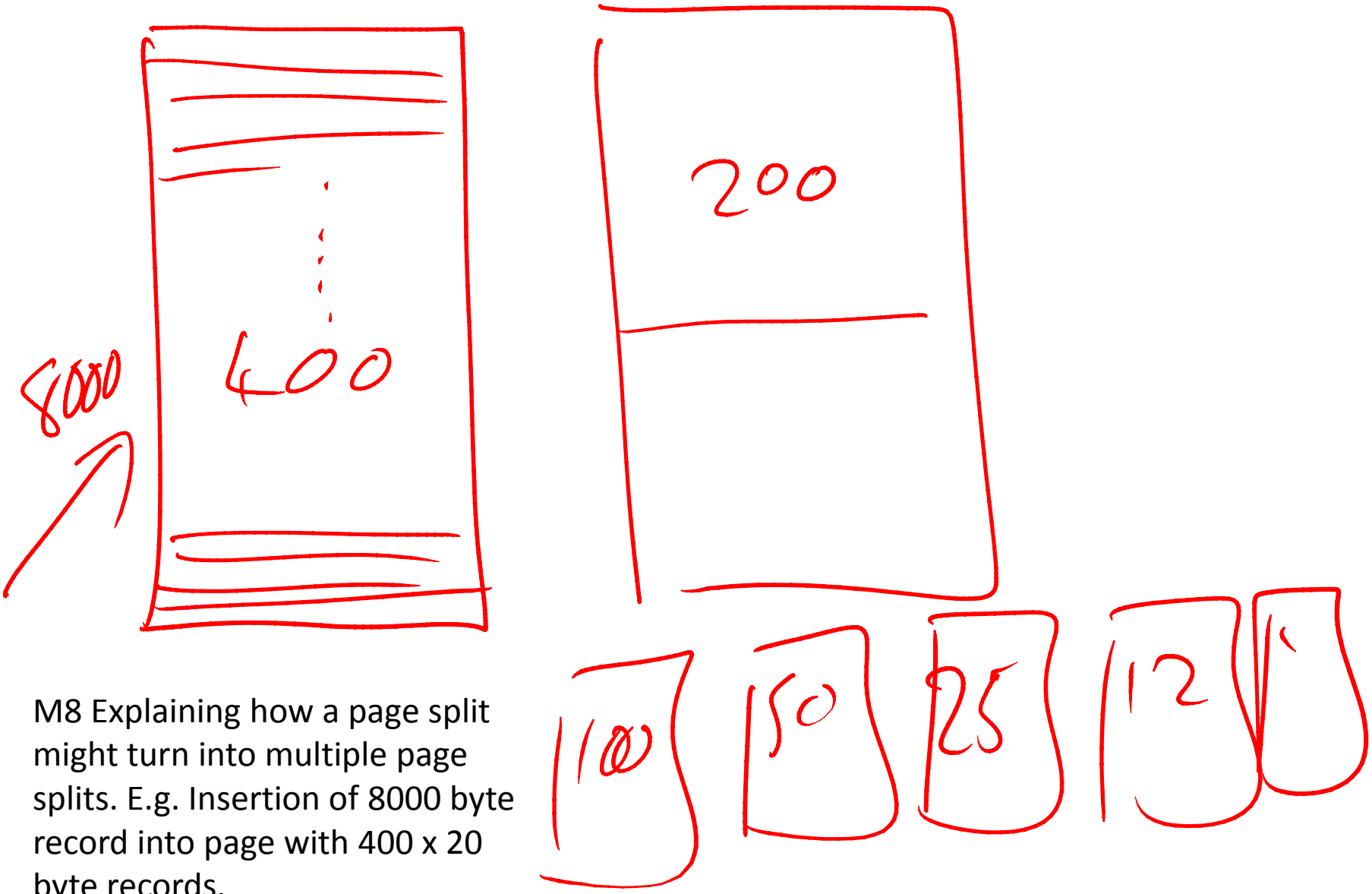
M8S5 Explaining how a record is found on a page using binary search rather than brute force search.

M8S15 Showing all the operations required when a page split occurs.

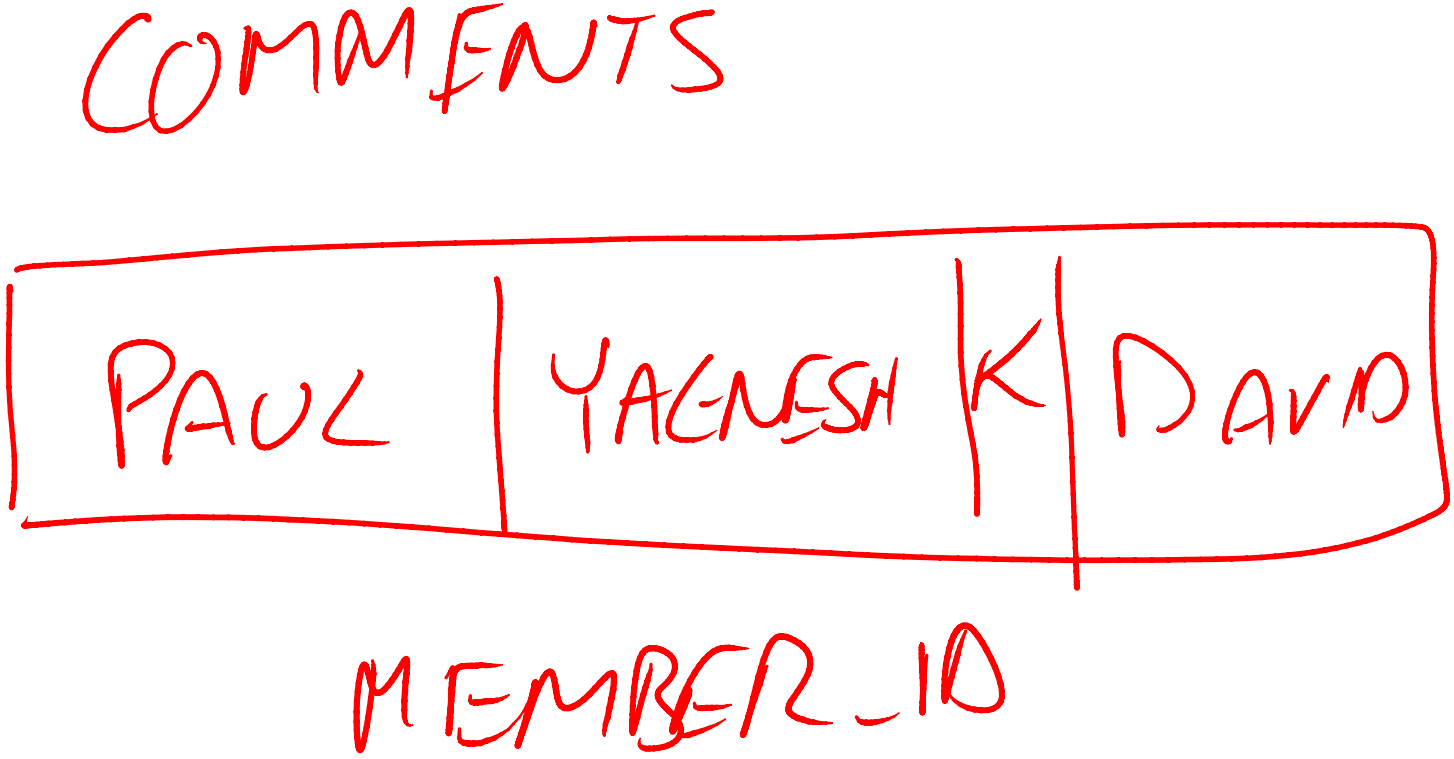


Row Size	No Split	Split	How Bad
1060	1244	6772	x5
100	344	5964	x18
10	256	10364	x45

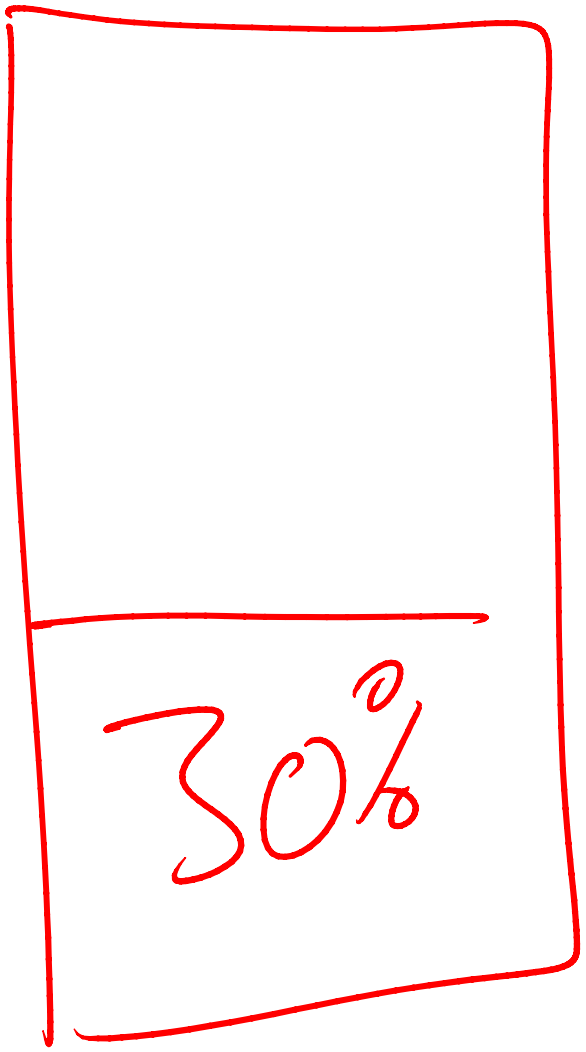
M8S22 The numbers from the page split logging cost demo.



M8 Explaining how a page split might turn into multiple page splits. E.g. Insertion of 8000 byte record into page with 400 x 20 byte records.

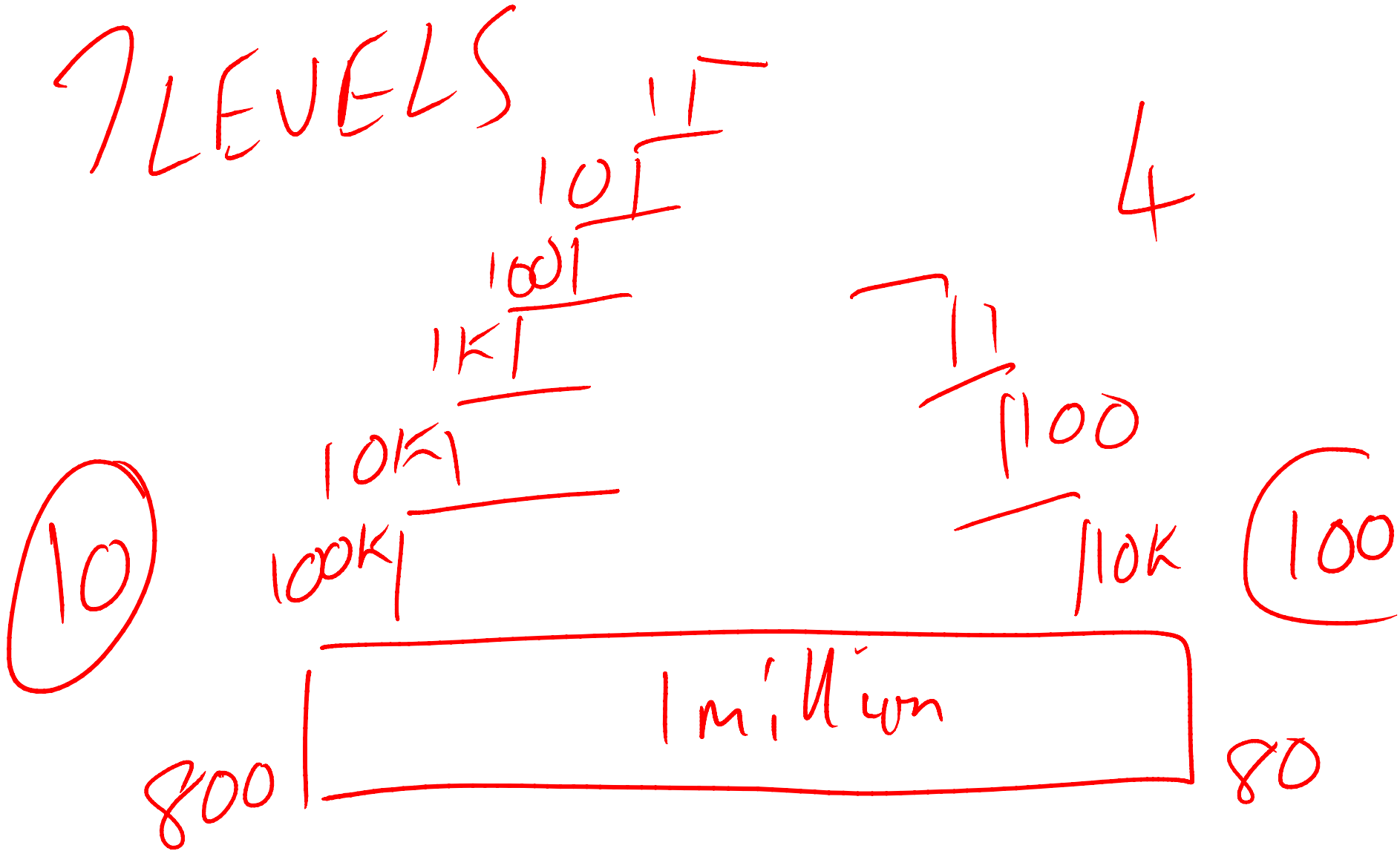


M8S23 The MySpace story. The ‘K’ is the tiny portion of comments on Kimberly’s page because she wasn’t very popular back then… ☺



FF = 70

M8S25 To leave 30% free space, set the fillfactor to 70.



M8 Explaining ‘fanout’. Larger index key size means less rows per page on each index level, meaning more index levels and more CPU for index traversal from root to leaf.

M8 Explaining the hierarchy of MAXDOP settings.

M8S37 Explaining staggered index maintenance with database mirroring. See http://bit.ly/IS04W5 for more explanation