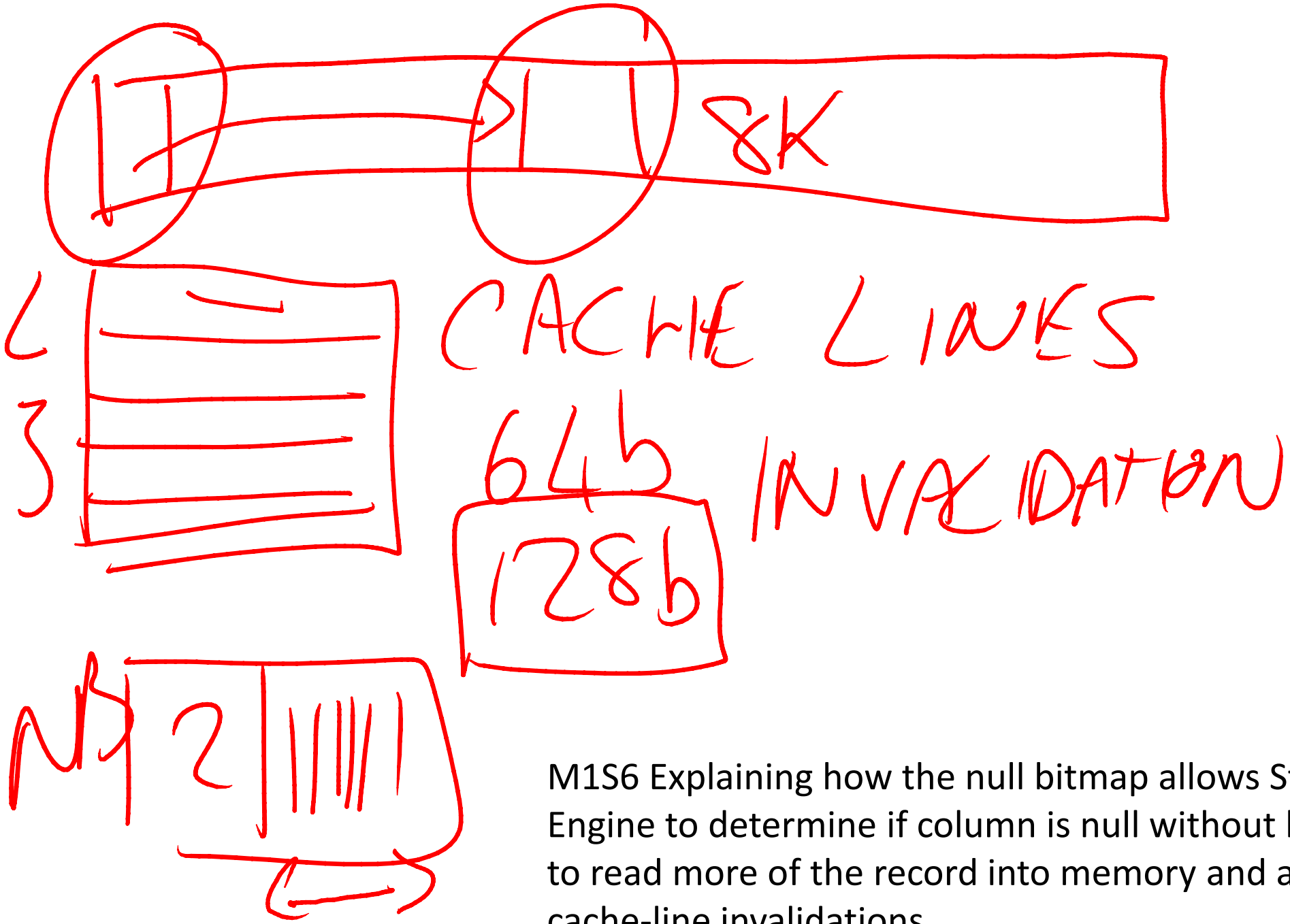
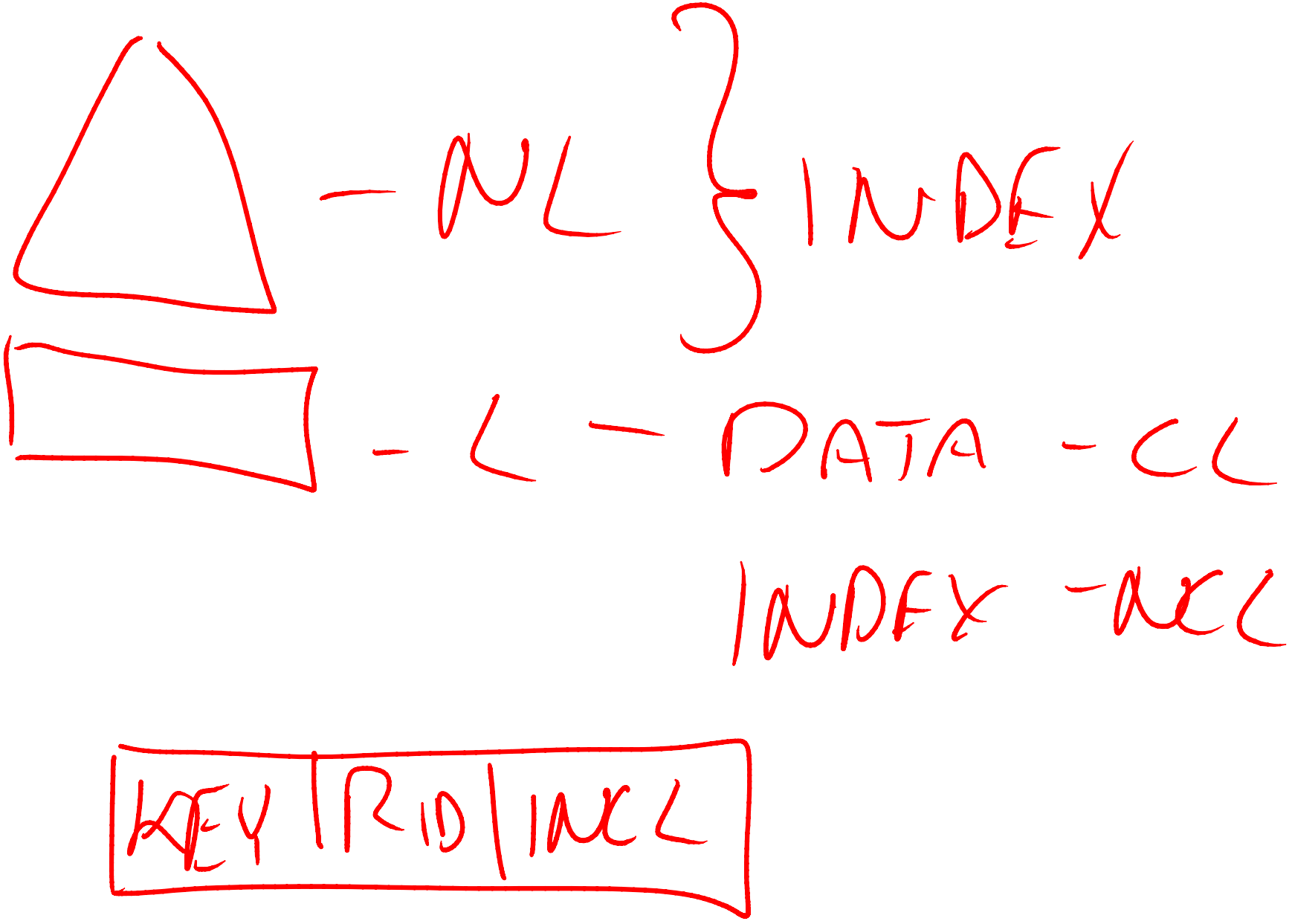
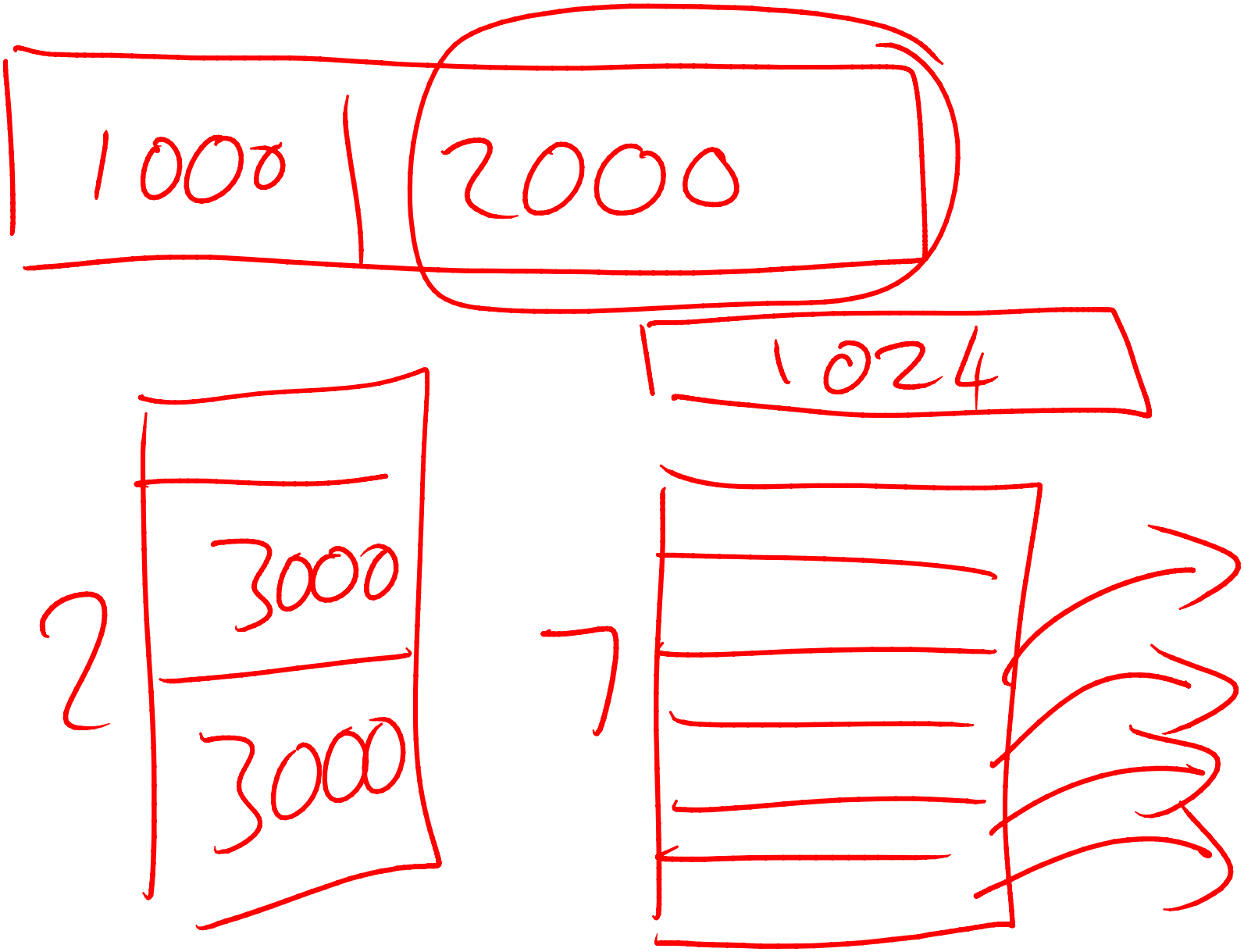


M1S6 Explaining how the null bitmap allows Storage Engine to determine if column is null without having to read more of the record into memory and avoiding cache-line invalidations.

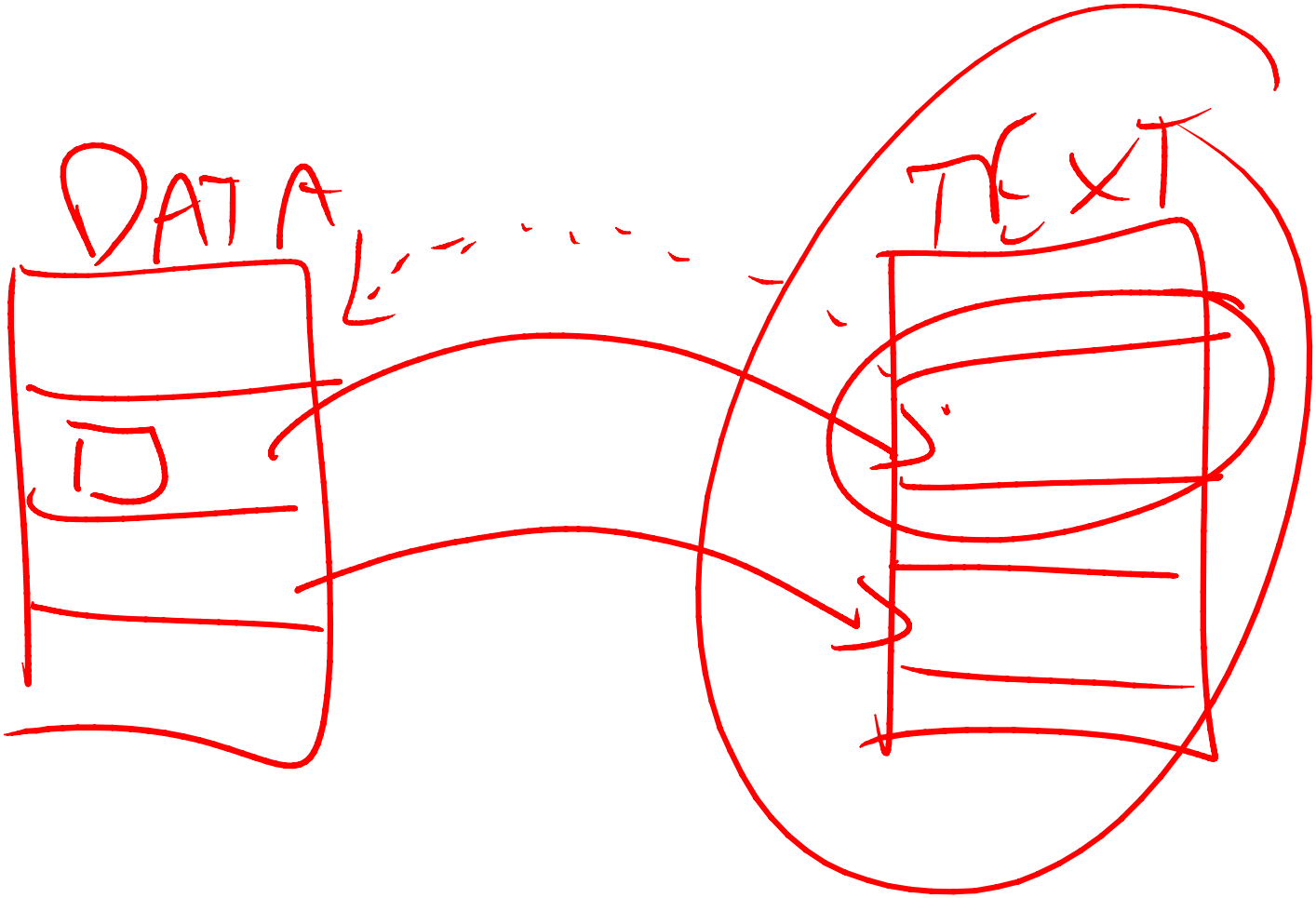
M1S8 Explaining how nonclustered indexes on heaps have a physical RID in each record that links back to the data record



M1S8 Explaining how clustered indexes have data records at the leaf level, nonclustered indexes have index records at the leaf level, and both kinds of indexes have index records at the non-leaf levels. Index record structure has the index keys, the link back to the data record, and any INCLUDEd columns.

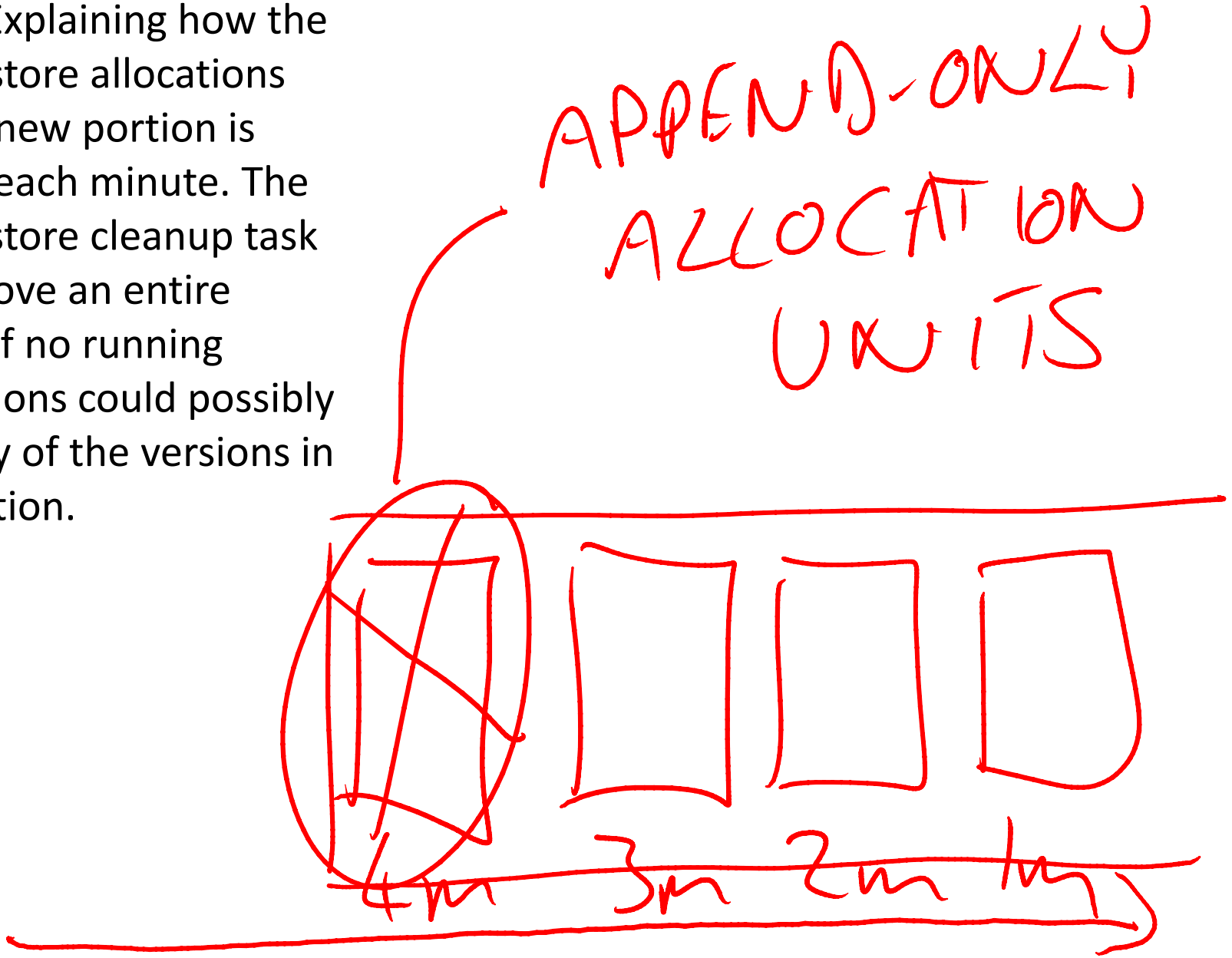


M1S10 Explaining how you decide whether to have large amounts of rarely used LOB data stored in-row or off-row. In-row means bigger records and lower data density on pages.

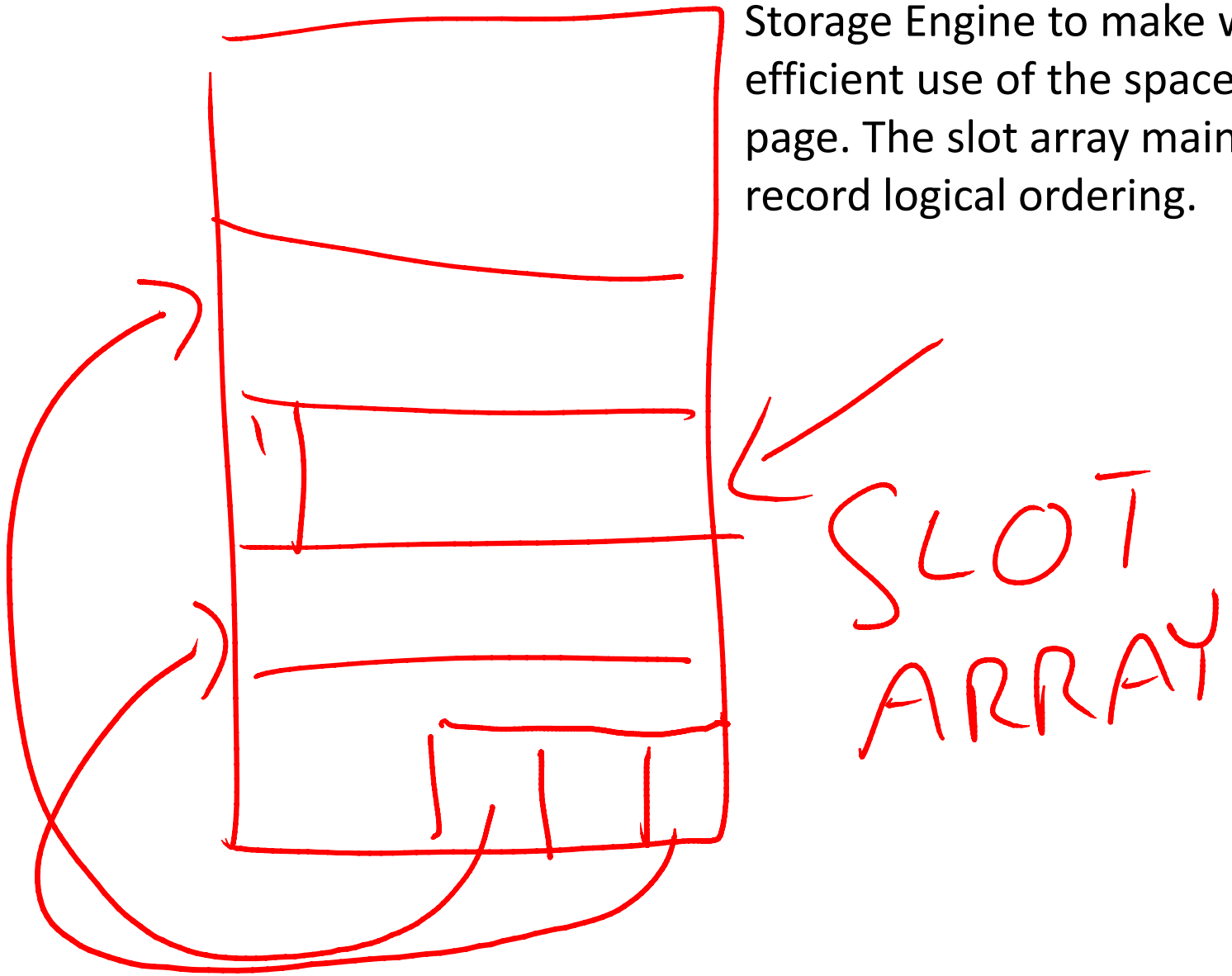


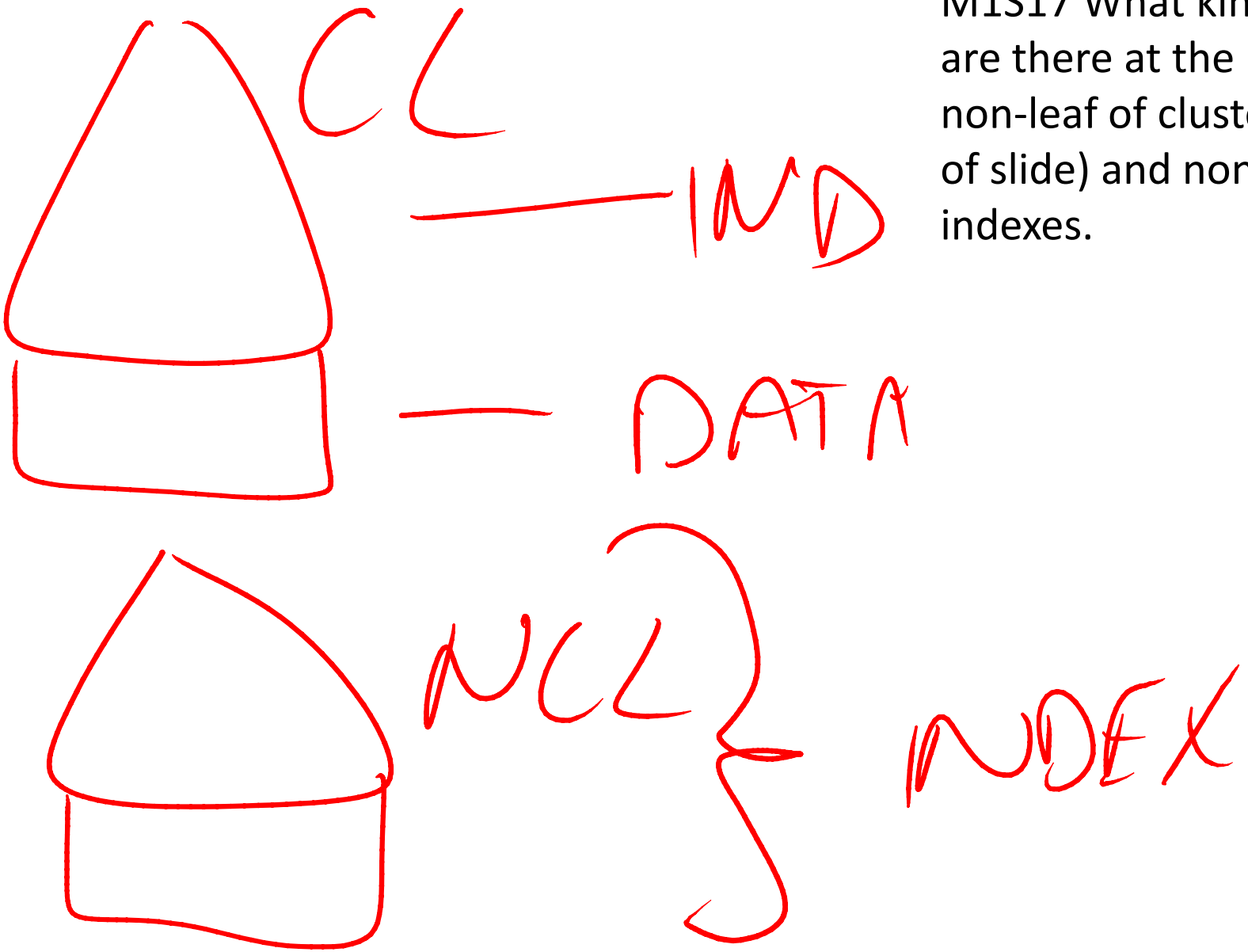
M1S10 Explaining that there are links from data/index records to off-row LOB storage but there are no back-links. This makes things like shrink run slowly when moving LOB data as each record that's moved needs a table scan to find the data record that points to it.

M1S13 Explaining how the version store allocations work. A new portion is created each minute. The version store cleanup task can remove an entire portion if no running transactions could possibly need any of the versions in that portion.



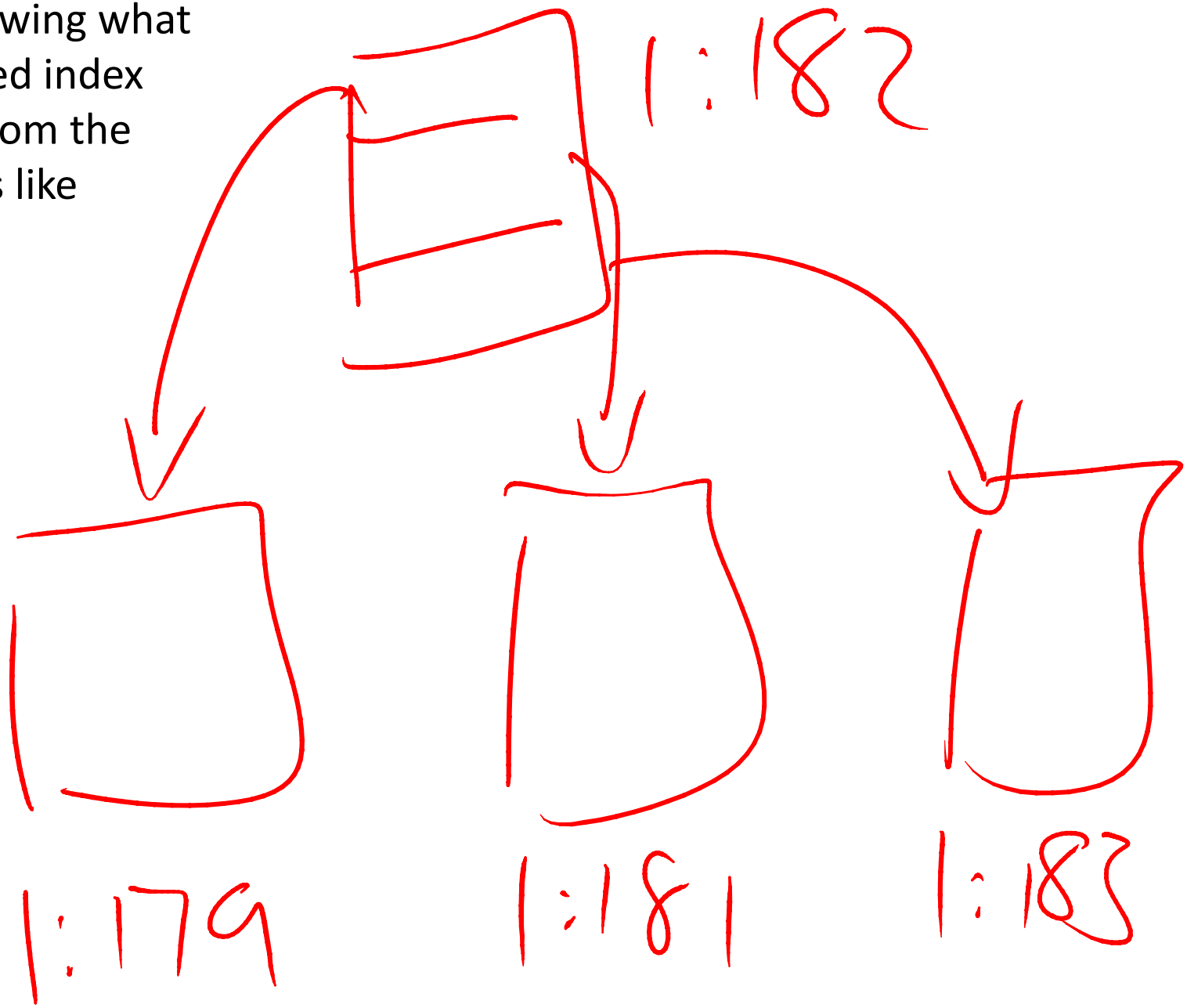
M1S16 Explaining how the slot array at the bottom of pages allows the Storage Engine to make whatever efficient use of the space on the page. The slot array maintains record logical ordering.

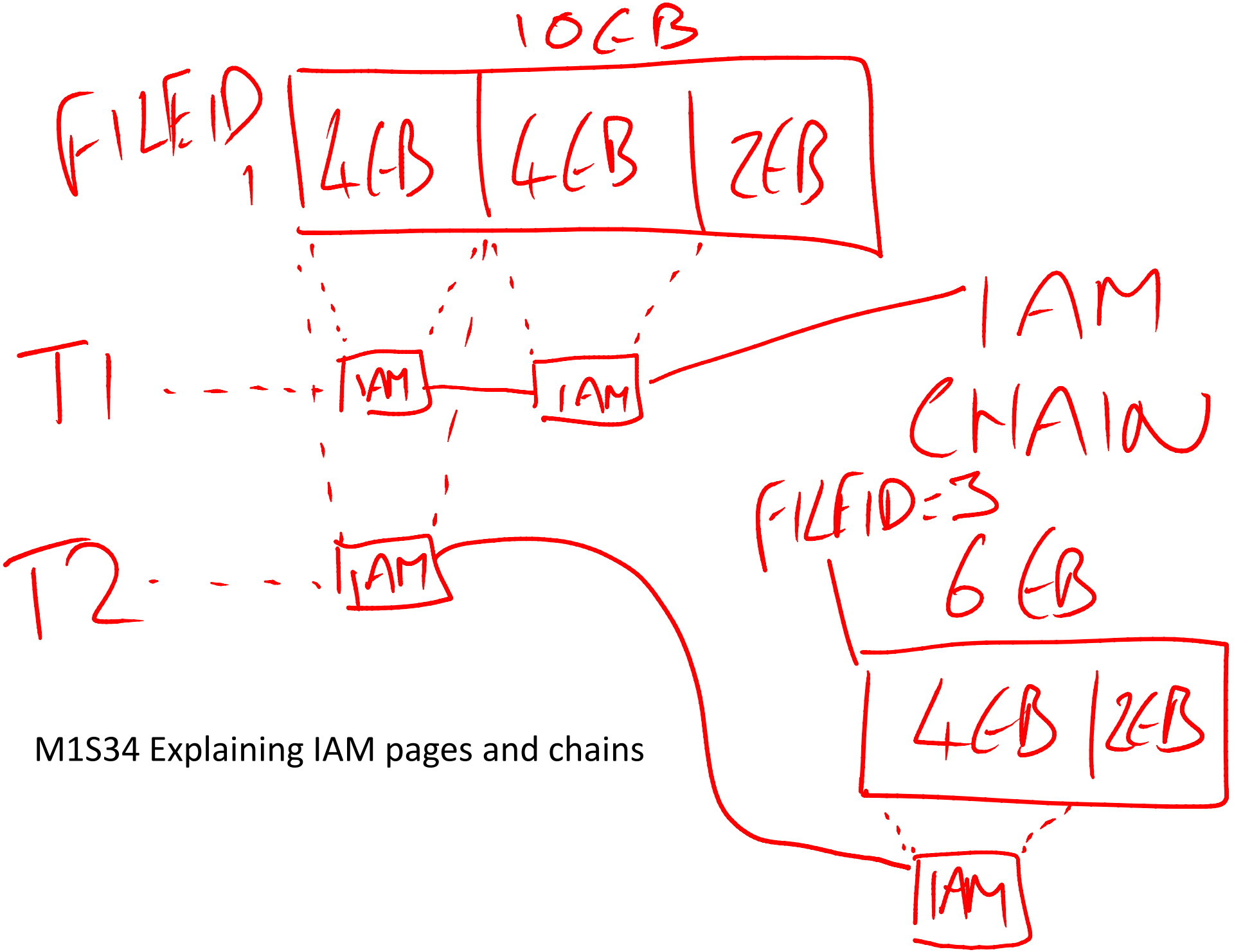




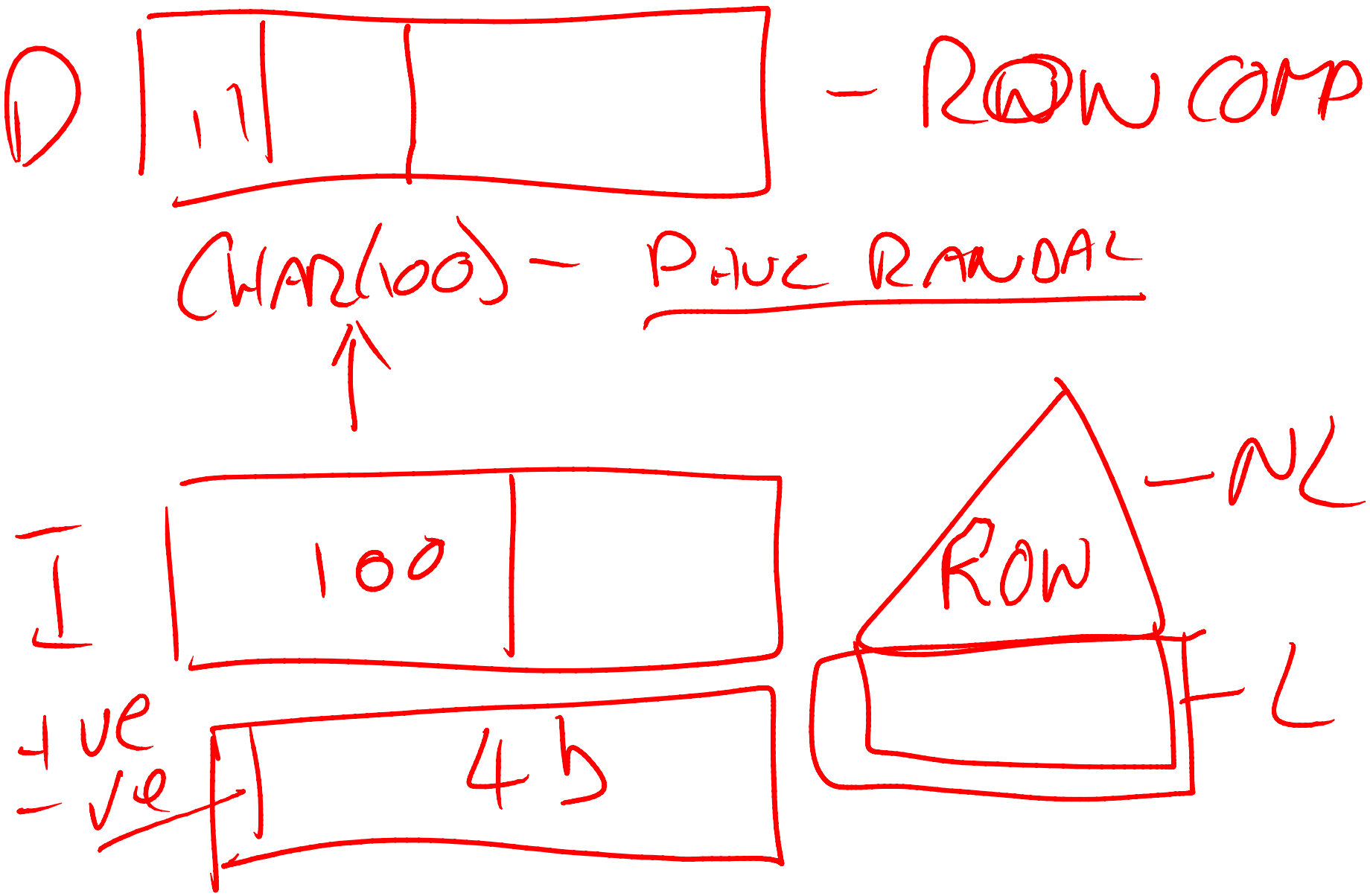
M1S17 What kind of pages are there at the leaf and non-leaf of clustered (at top of slide) and nonclustered indexes.

M1S20 Showing what the clustered index structure from the demo looks like

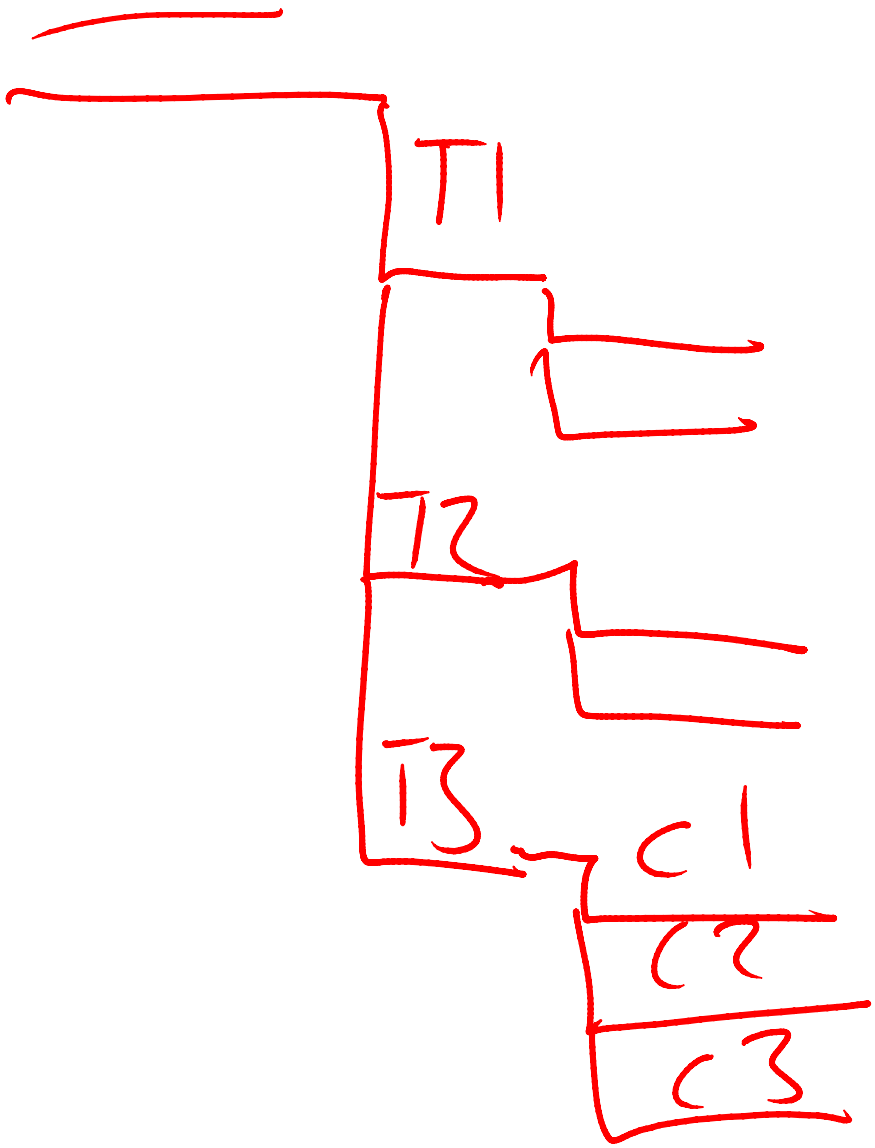




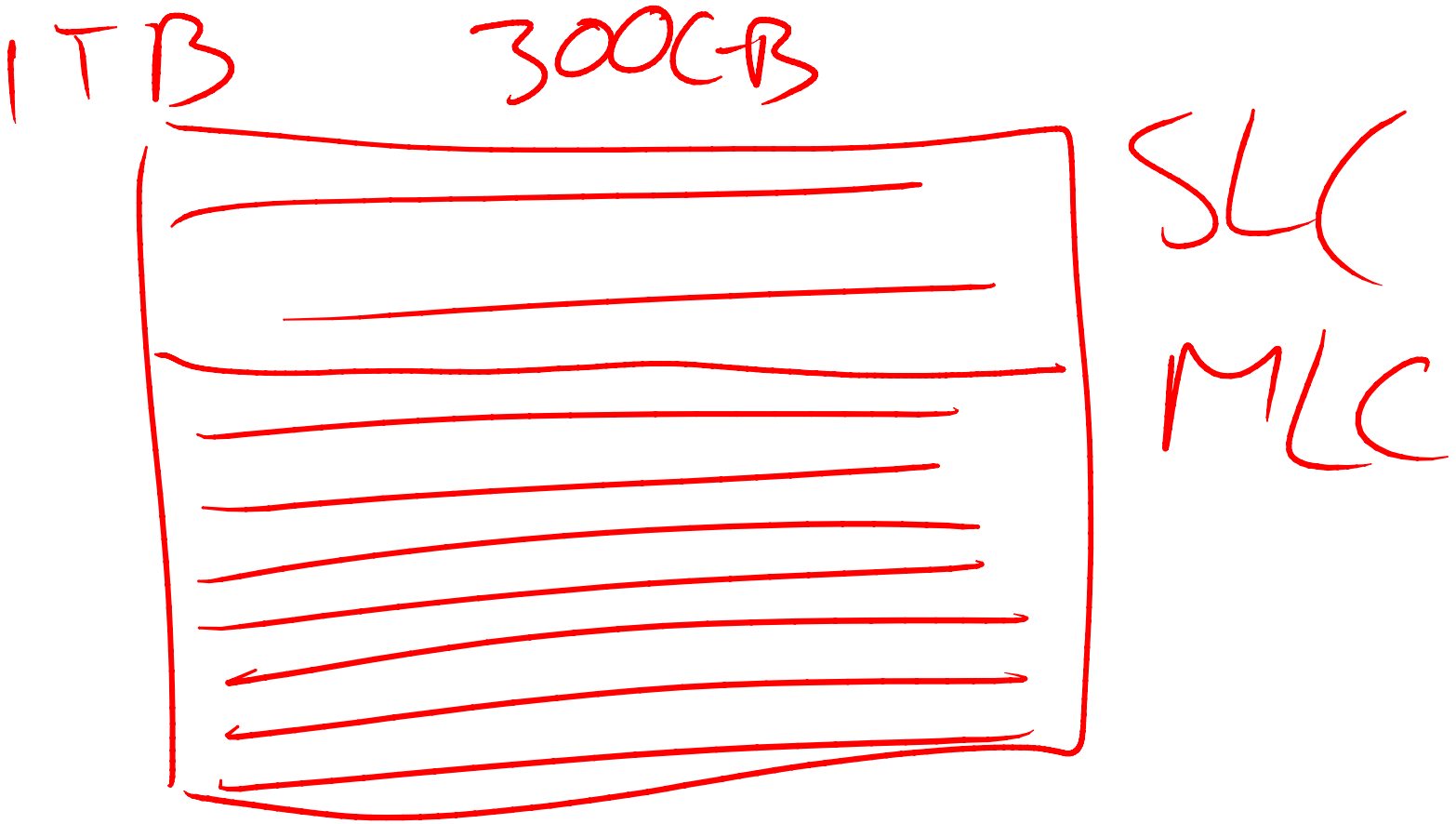
M1S34 Explaining IAM pages and chains



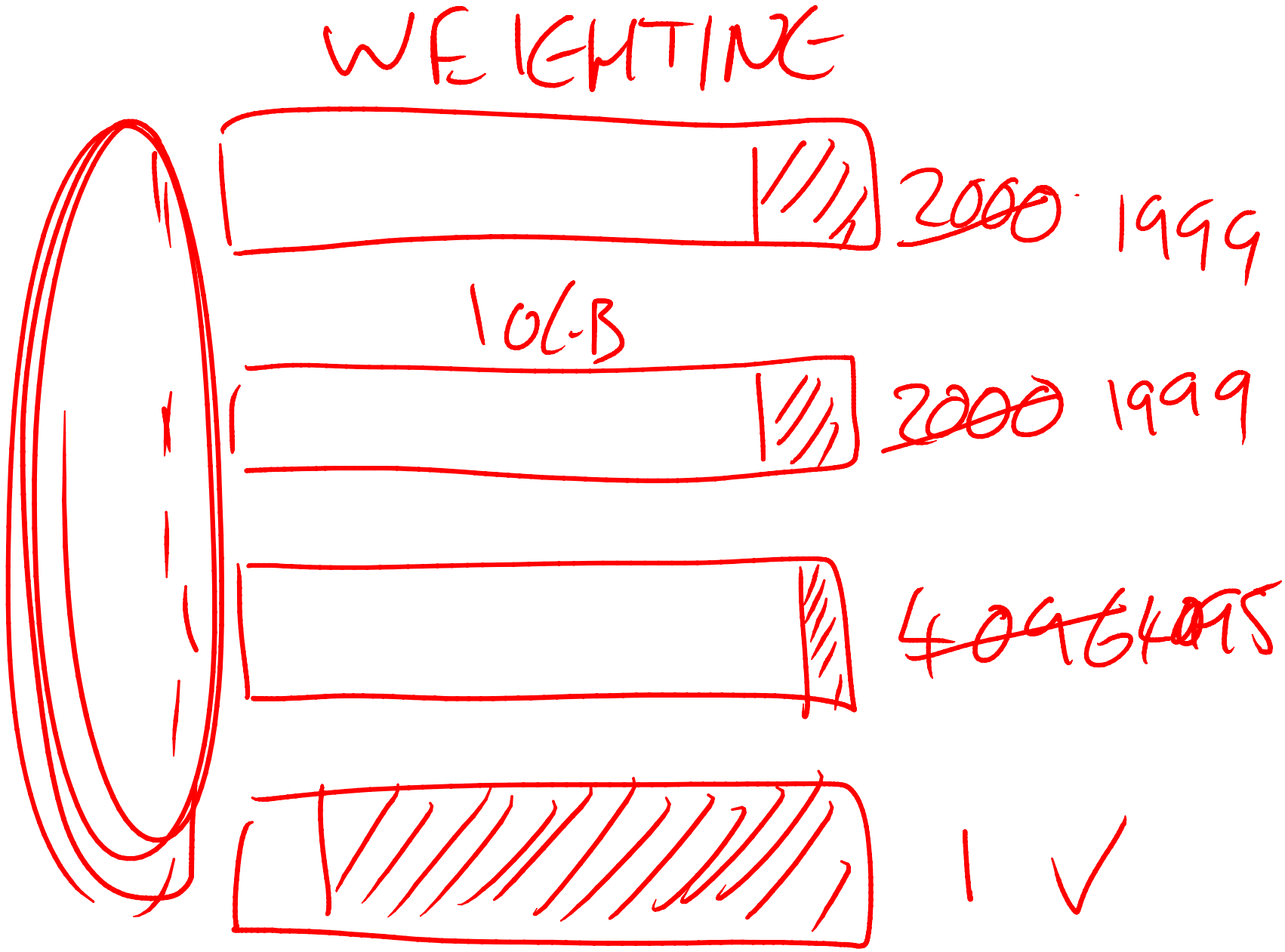
M2S4 Discussing that row compression will squeeze out empty bytes of storage from char or +ve/-ve integers.



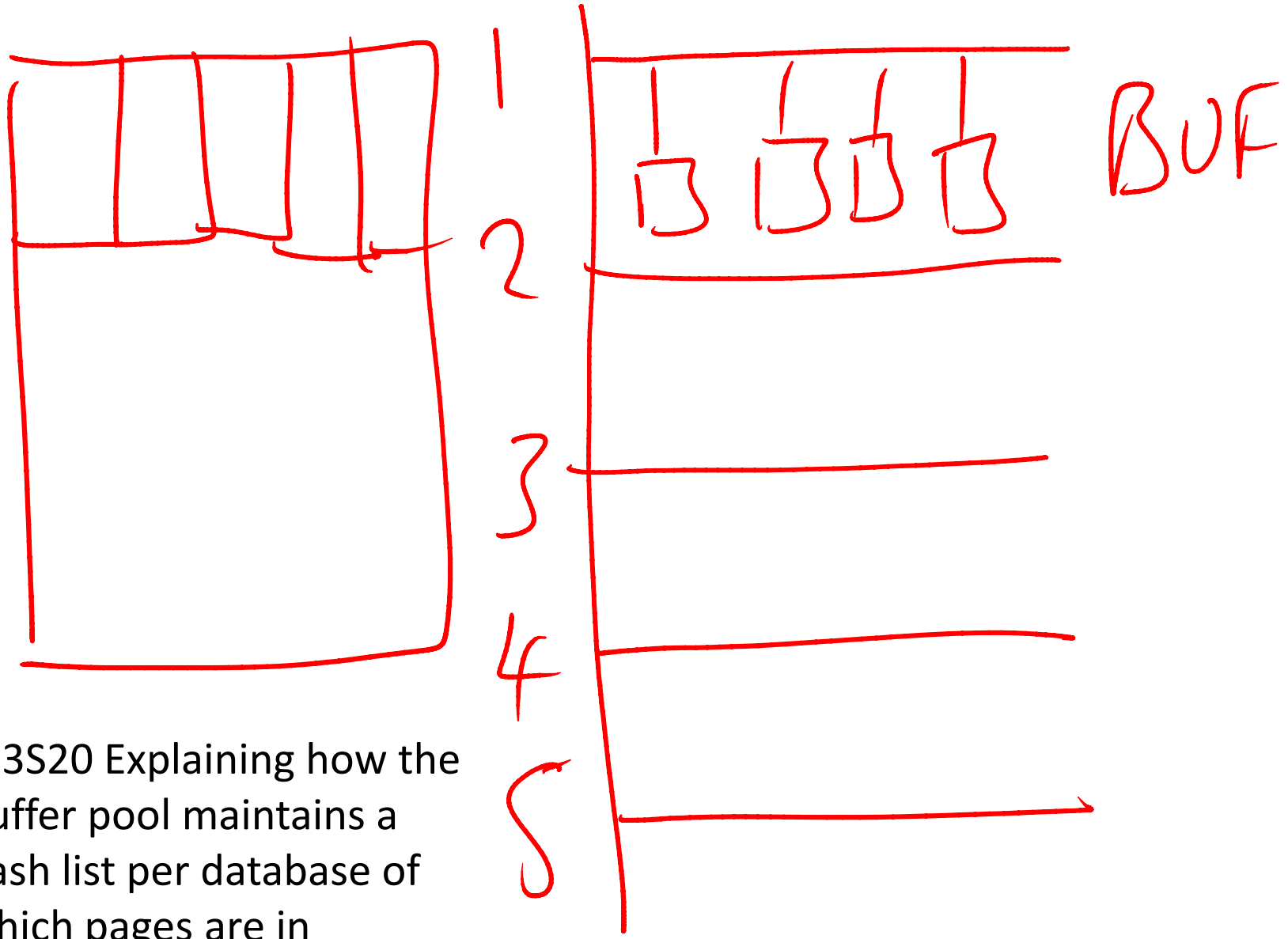
M2S39 Explaining the directory structure created in a FILESTREAM data container. Each table has a directory and then a sub-directory for each FILESTREAM column.



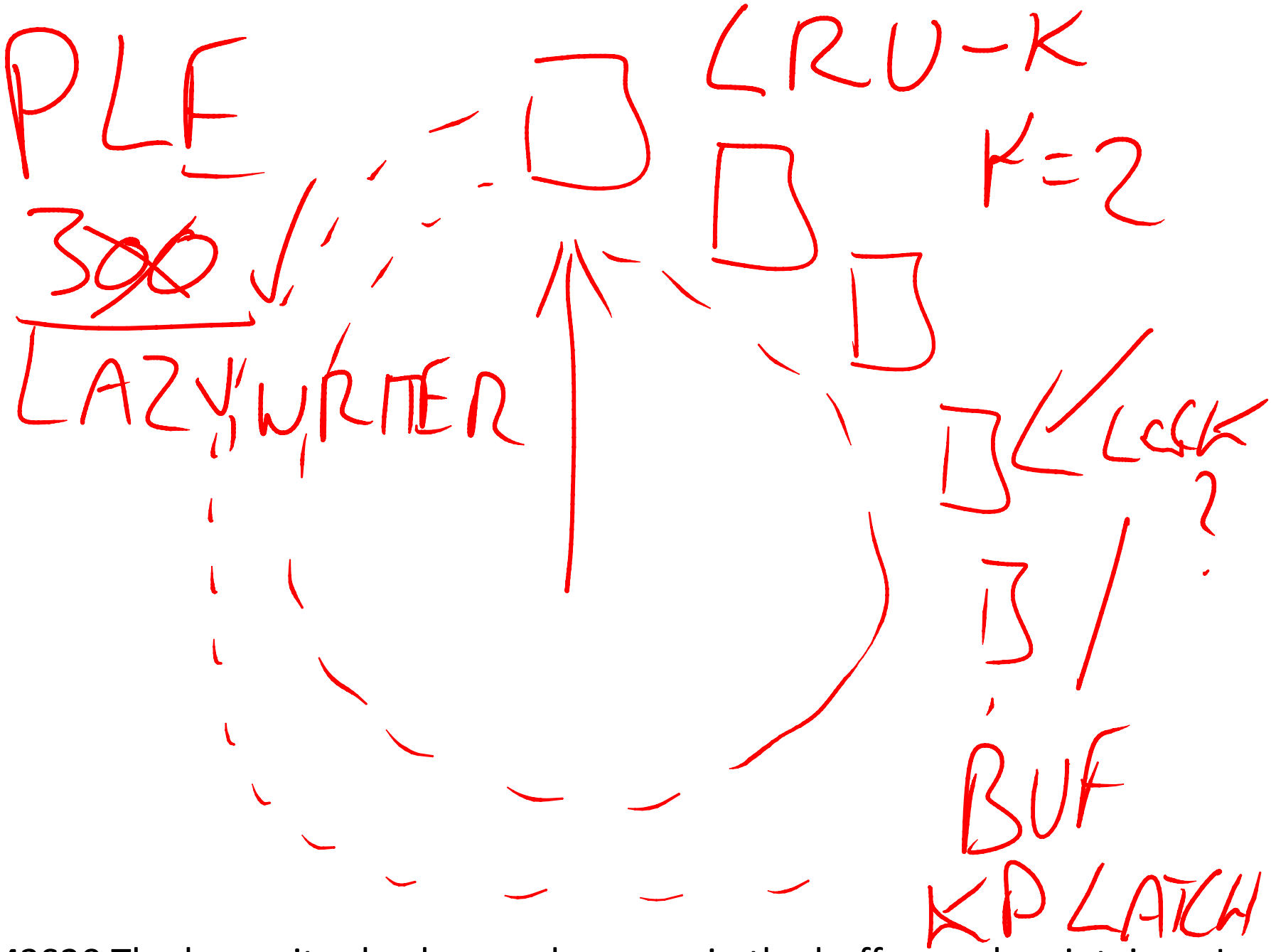
M3S17 Explaining how SSDs spread the writes over the whole ‘surface’ of the drive to amortize the degradation of memory cells across all cells to avoid one portion of the drive wearing out first.



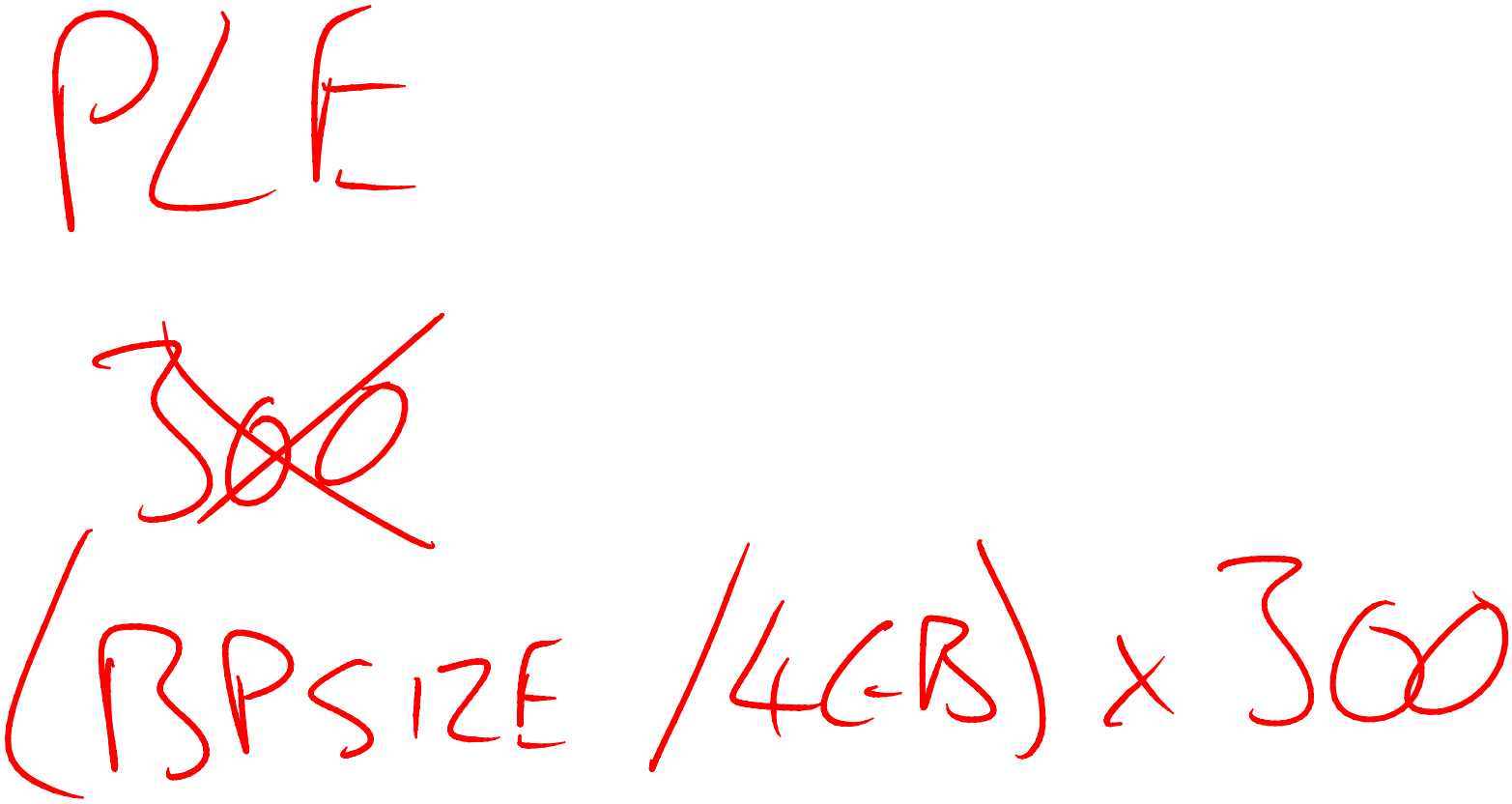
M3S19 Explaining round-robin allocation and proportional fill.



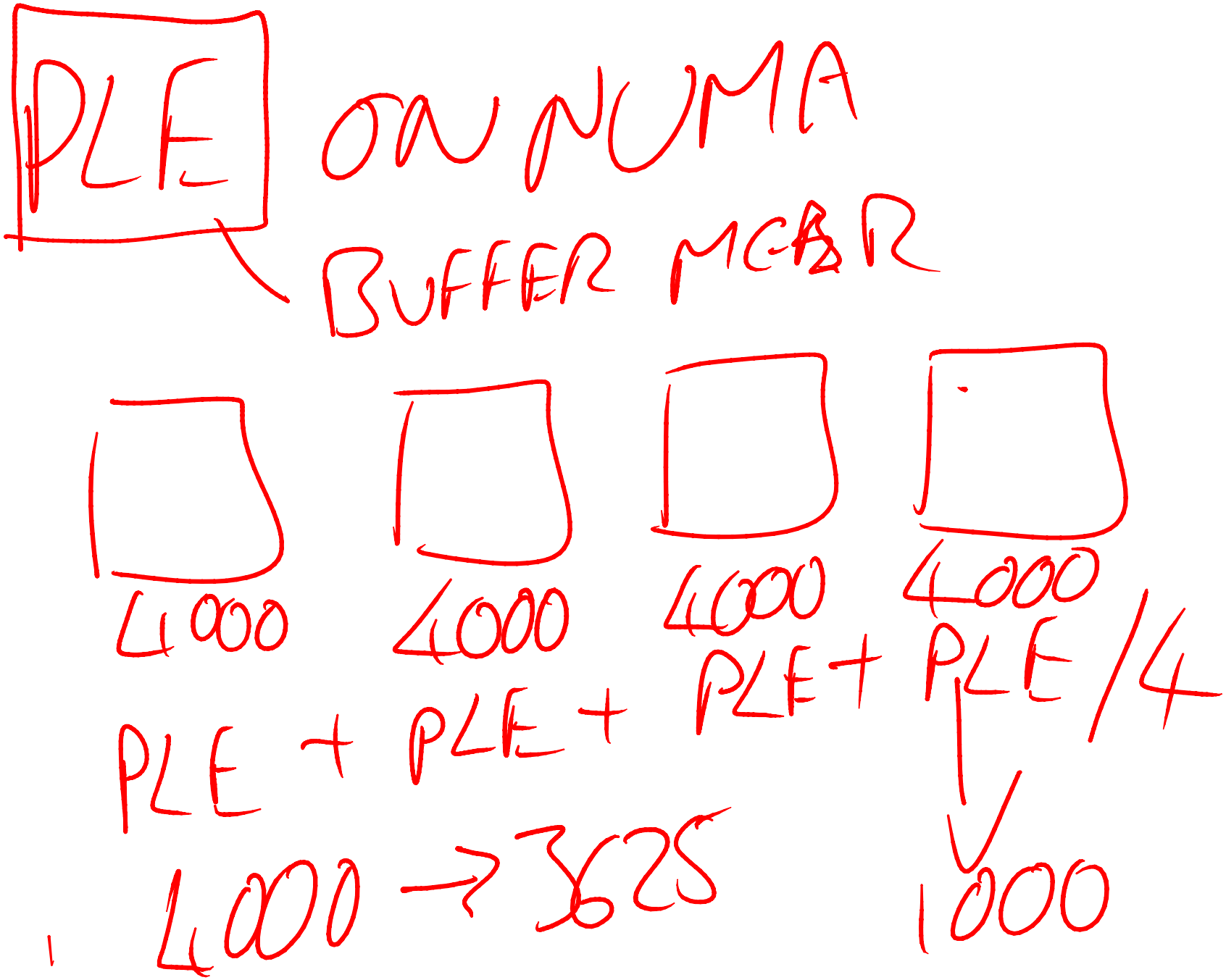
M3S20 Explaining how the buffer pool maintains a hash list per database of which pages are in memory.



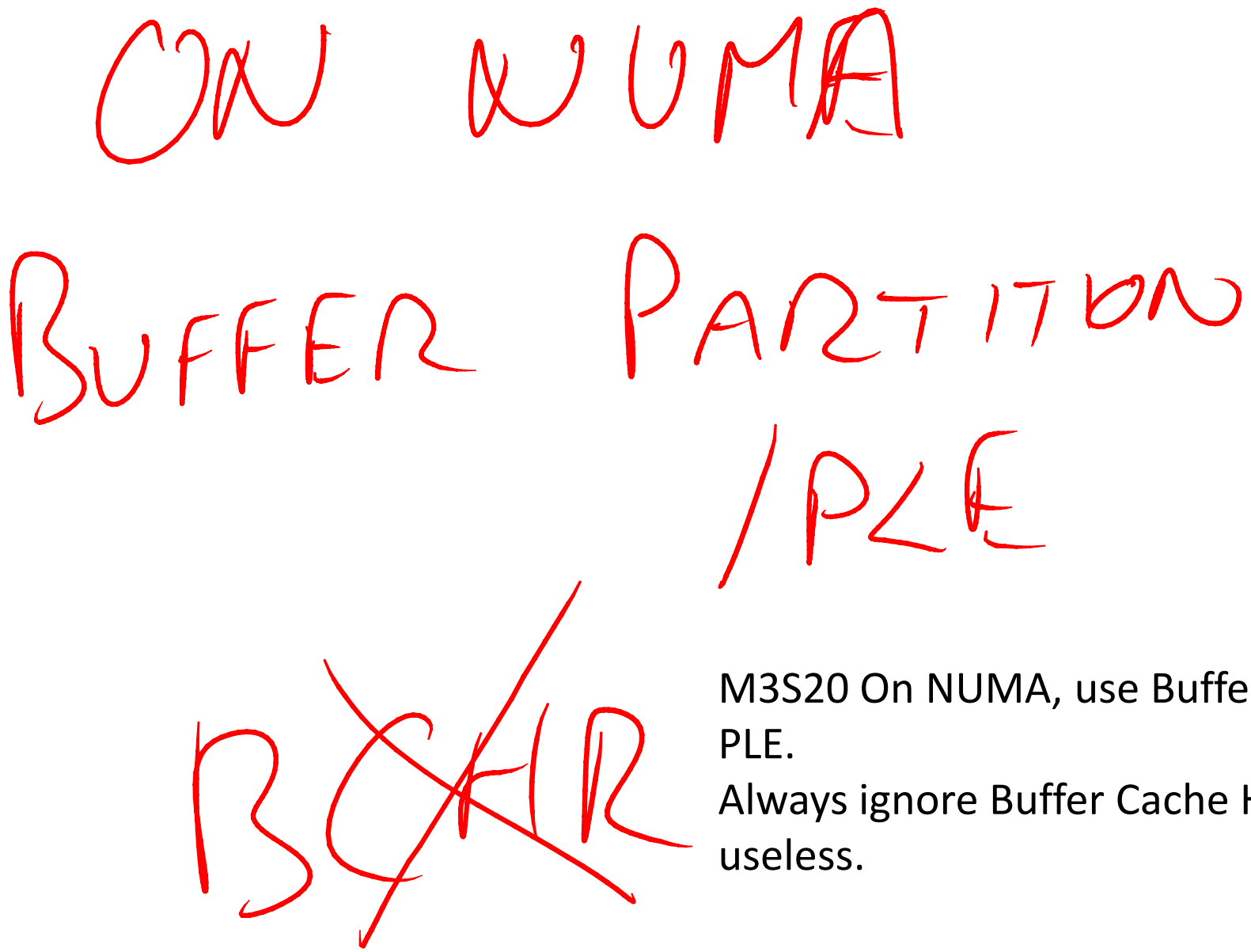
M3S20 The lazywriter background process in the buffer pool maintains a Least Recently Used list of pages to enable getting free space. An unlocked page may have a KP (keep) latch on it to prevent the page being dropped.



M3S20 Page Life Expectancy threshold of 300 is way too low and is often misquoted by people. Use the formula (buffer pool size in GB/4GB)x300 as a better threshold to care about.



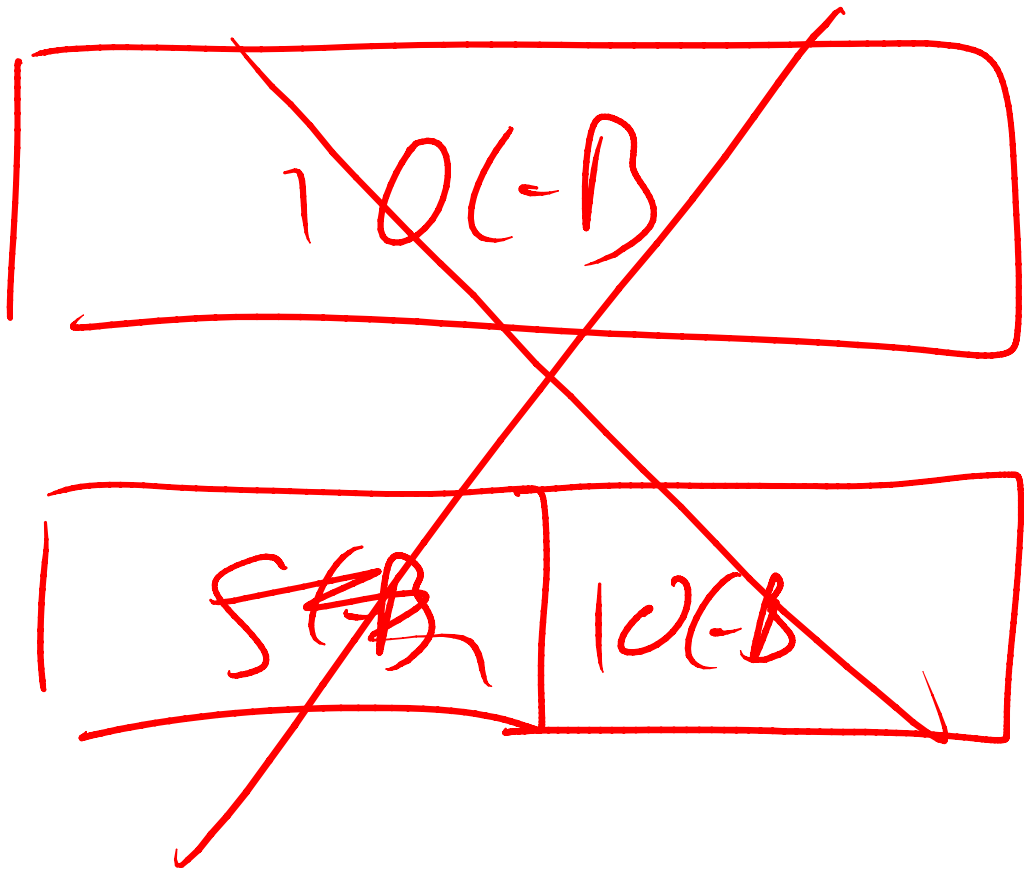
M3S20 On NUMA, Buffer Manager PLE is an average of all PLEs in each Buffer Partition. Drastic drop in one NUMA node won't reflect accurately.



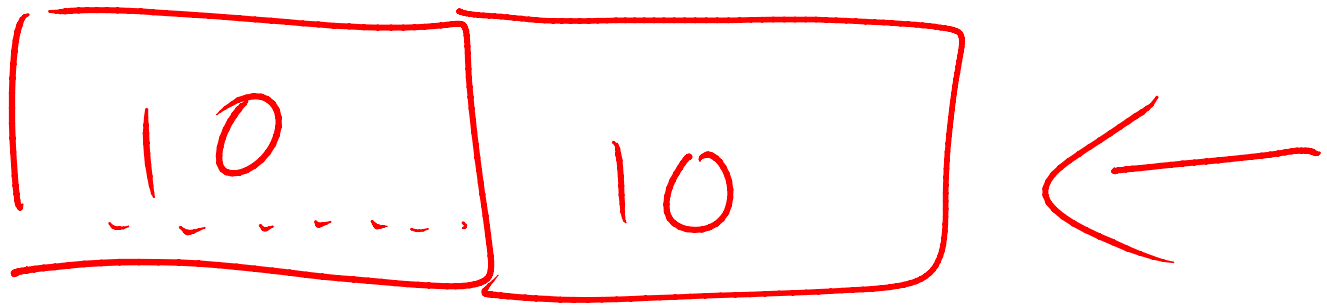
M3S20 On NUMA, use Buffer Partition PLE.
Always ignore Buffer Cache Hit Ratio. It's useless.

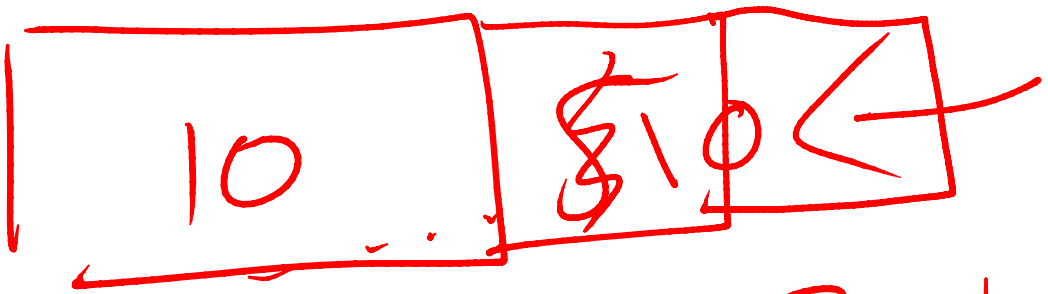
M3S22 Explaining how after creating a file, the file system trusts only up to the highwatermark – the highest point in the file written to. Instant file initialization removes the need to zero out the file.

M3S24 Explaining the slight security risk of using IFI on a volume shared with other data uses.



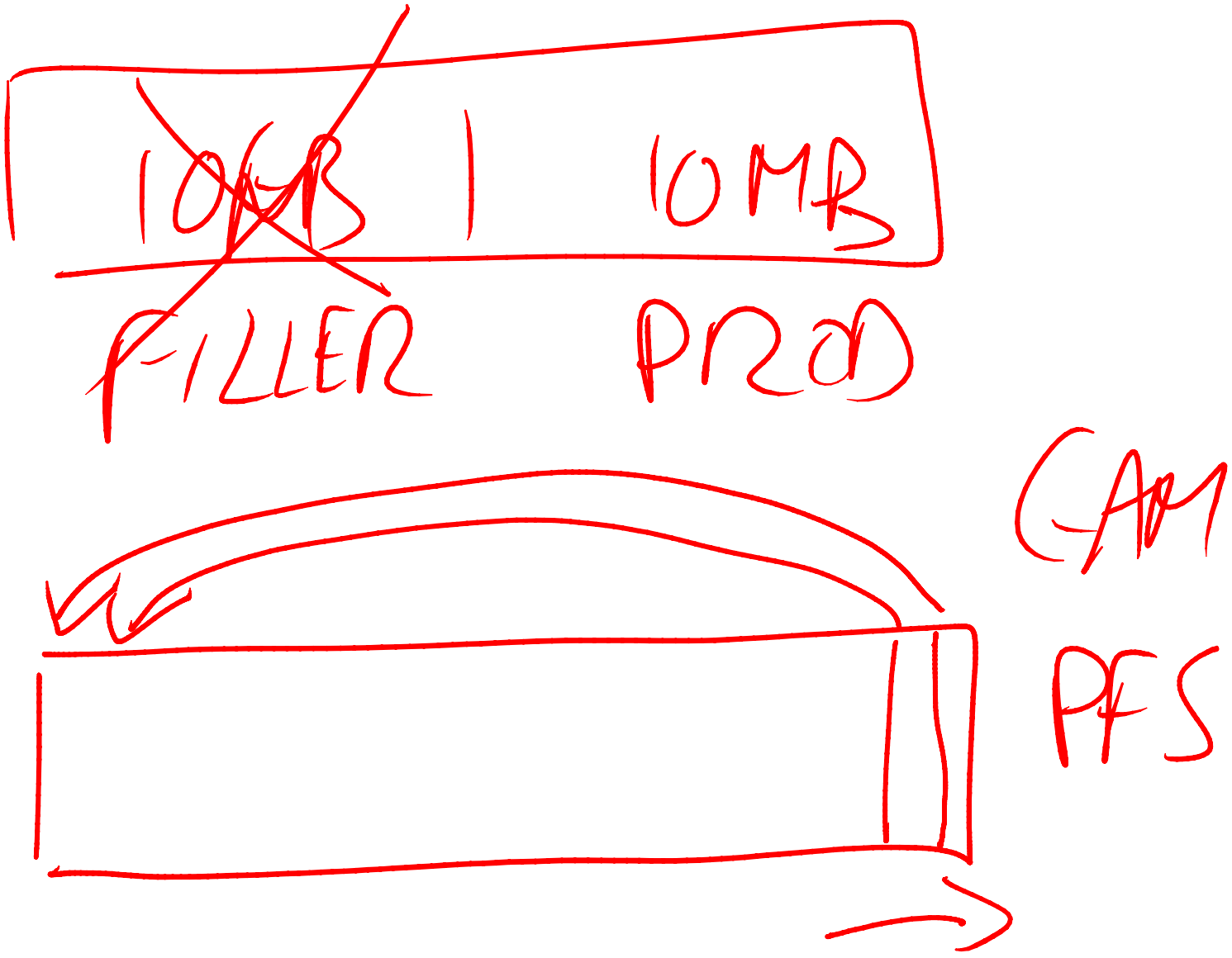
M3S26 Answering a question about whether having different sized files on a non-tempdb database will screw up autogrow and proportional fill. It won't.





TF 1106 2:1

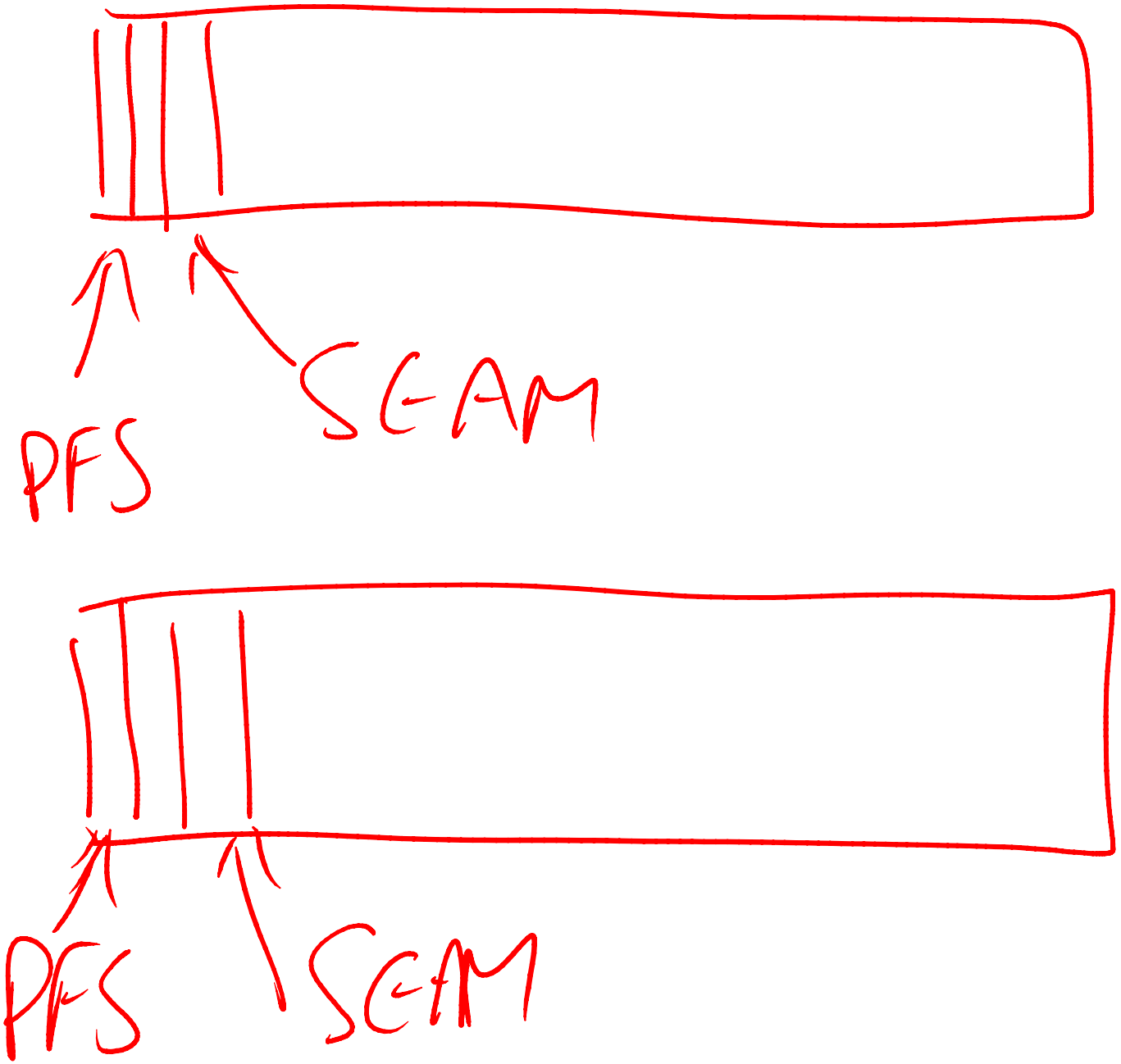
M3S26 Continuing to answer the question… undocumented trace flag 1106 will dump the proportional fill weightings. I haven't played with it yet.



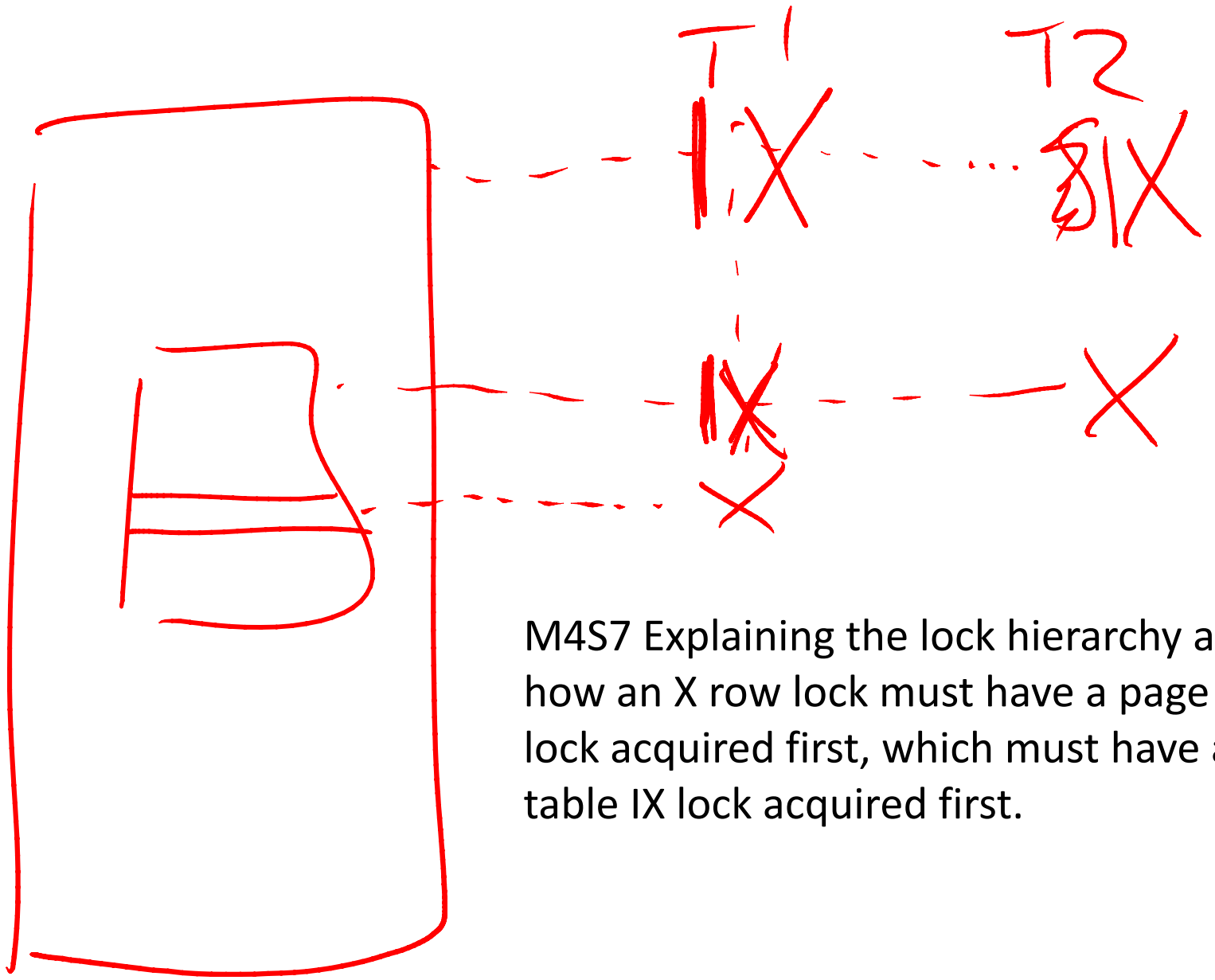
M3S28 Explaining the shrink demo setup.

TF 1117

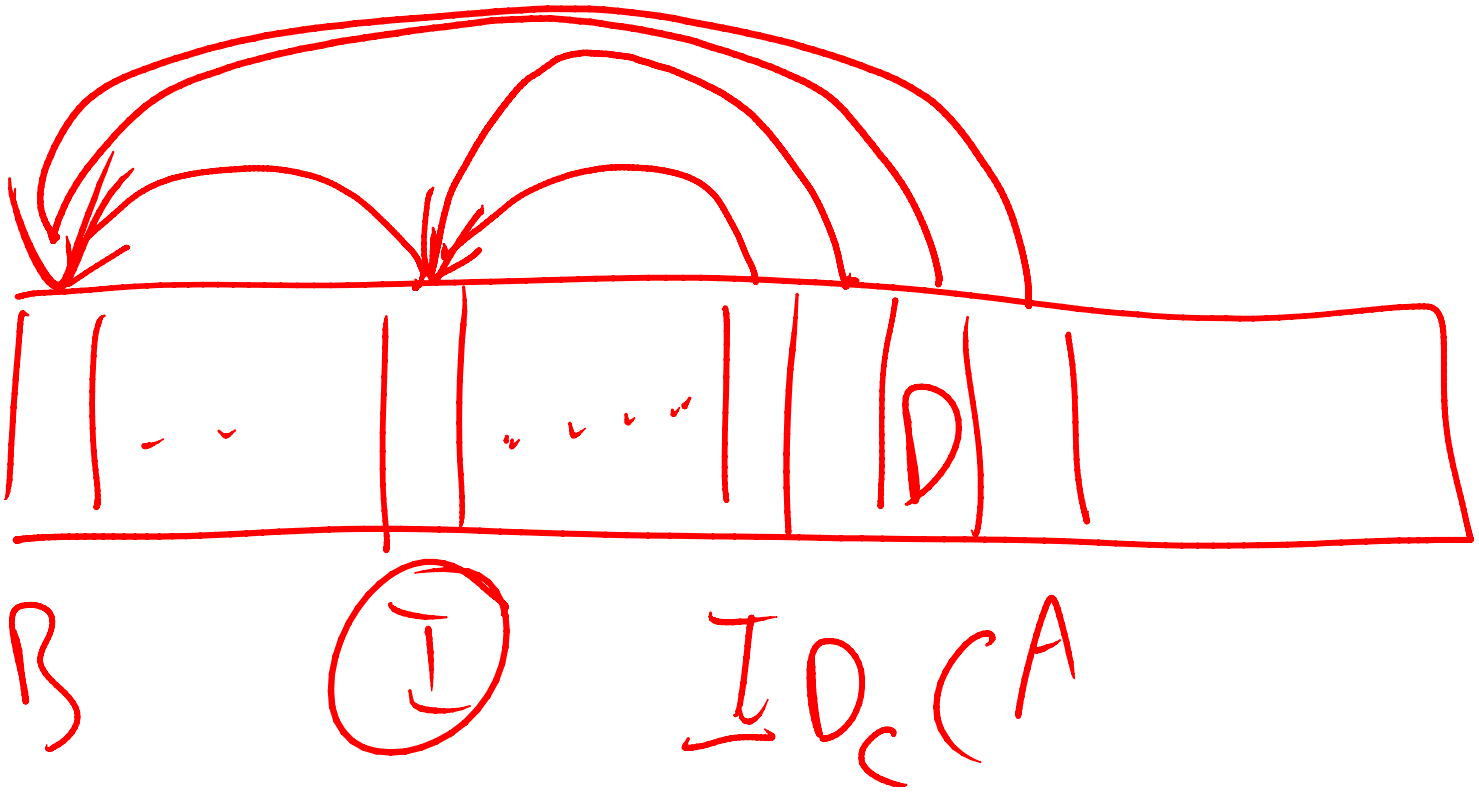
M3S37 This trace flag means that when a file in a filegroup has to grow, all files in the filegroup will grow at the same time. It's a workaround for the allocation bug in tempdb that if one file is created larger than the others, it will be the only one that grows.
Be careful – trace flag affects all databases, not just tempdb.



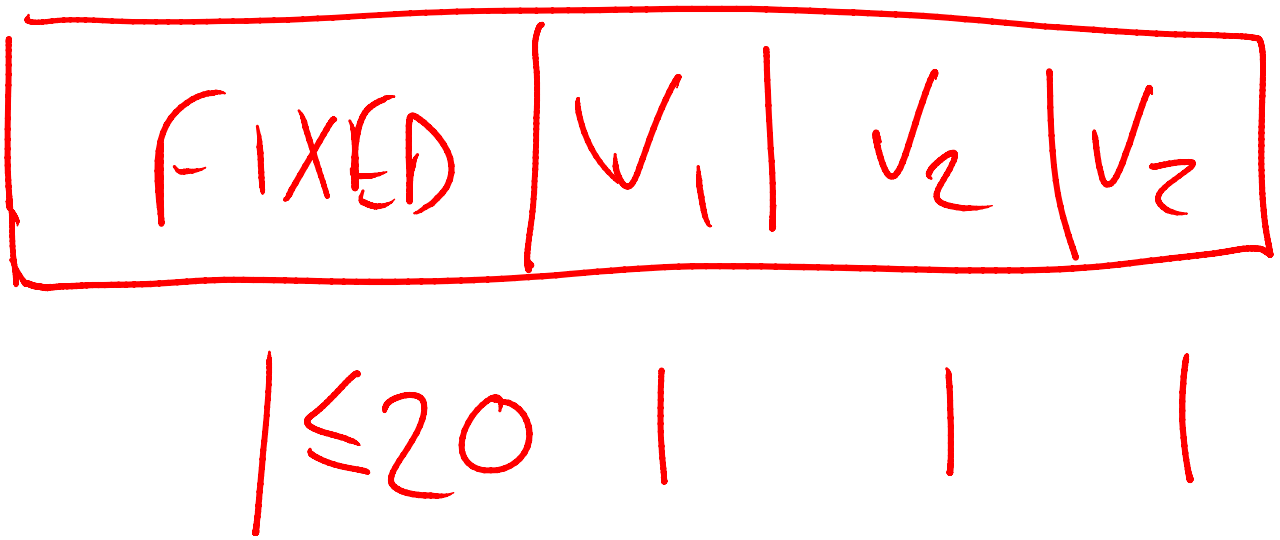
M3S41 Explaining PFS and SGAM contention in tempdb



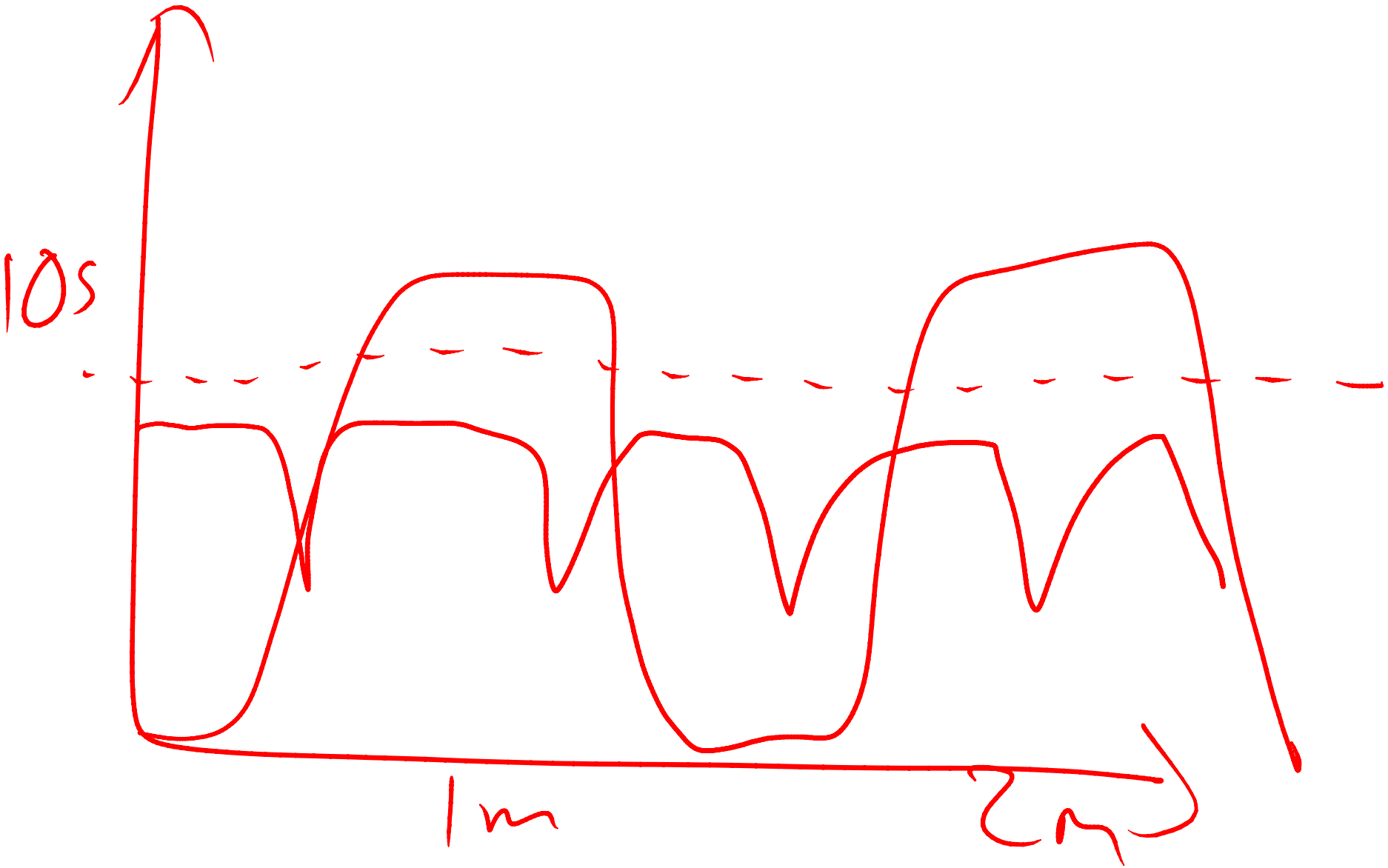
M4S7 Explaining the lock hierarchy and how an X row lock must have a page IX lock acquired first, which must have a table IX lock acquired first.



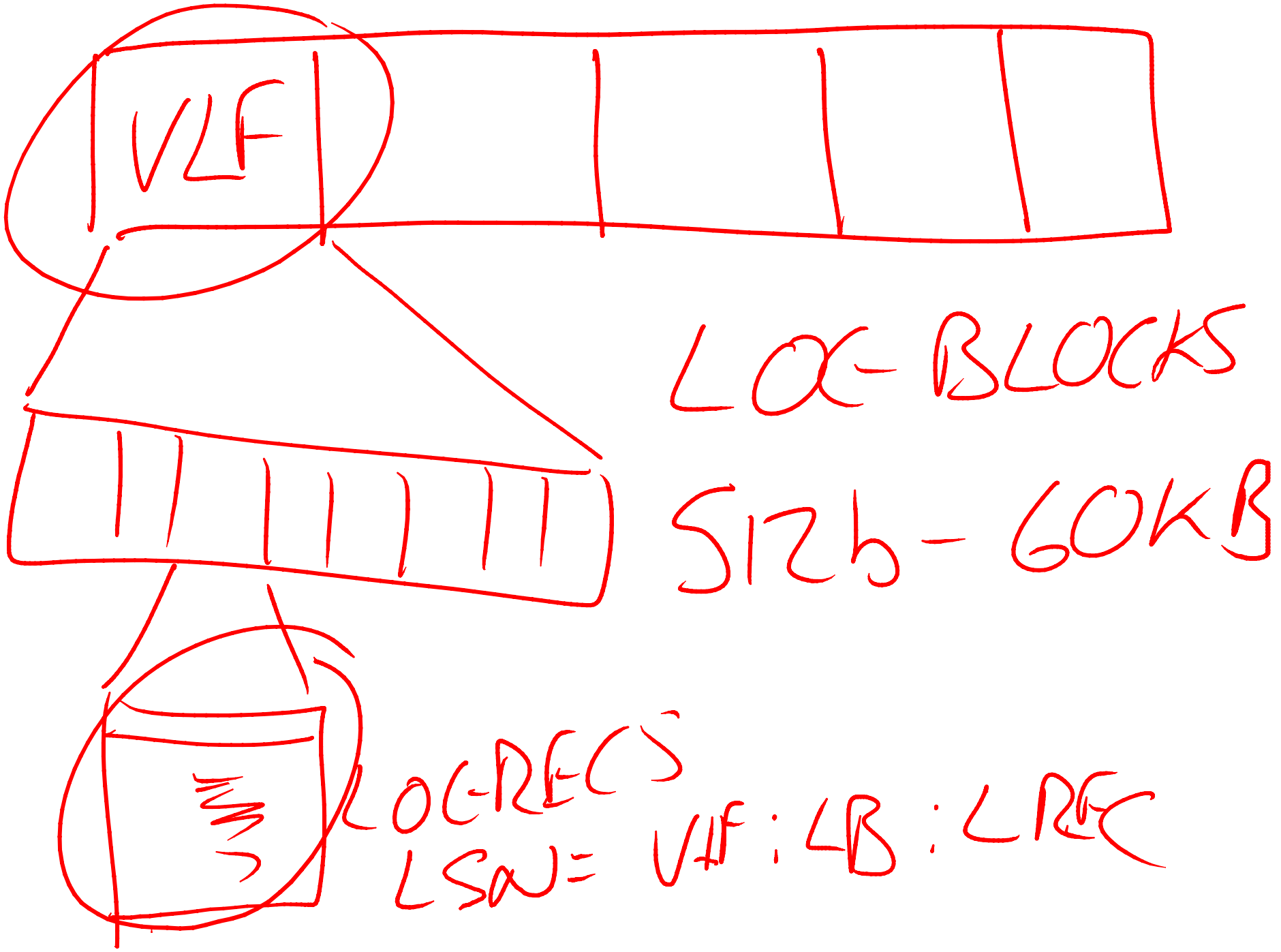
M4S12 Explaining how compensation log records work when a transaction rolls back. Log records are linked backwards in LSN order. Compensation log records link back over the record they're compensating for, to the prior log record. This avoids crash recovery having to compensate again if a subsequent crash occurs.

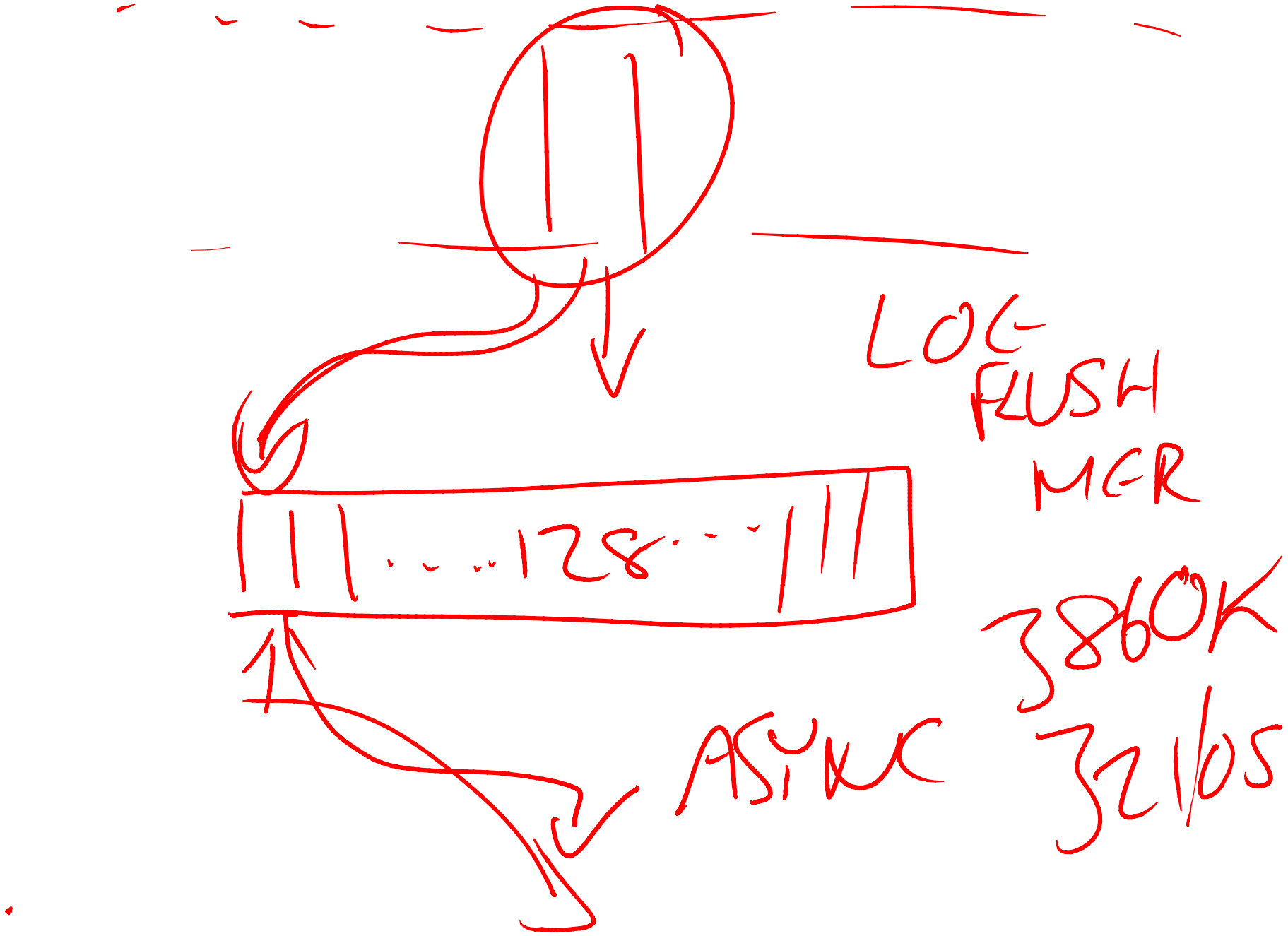


M4S12 Showing how a record is broken up into portions. Each variable length column is a portion and so is the entire fixed-width portion of the record. See the comment at the bottom of demo 1 in module 4 for more explanation.

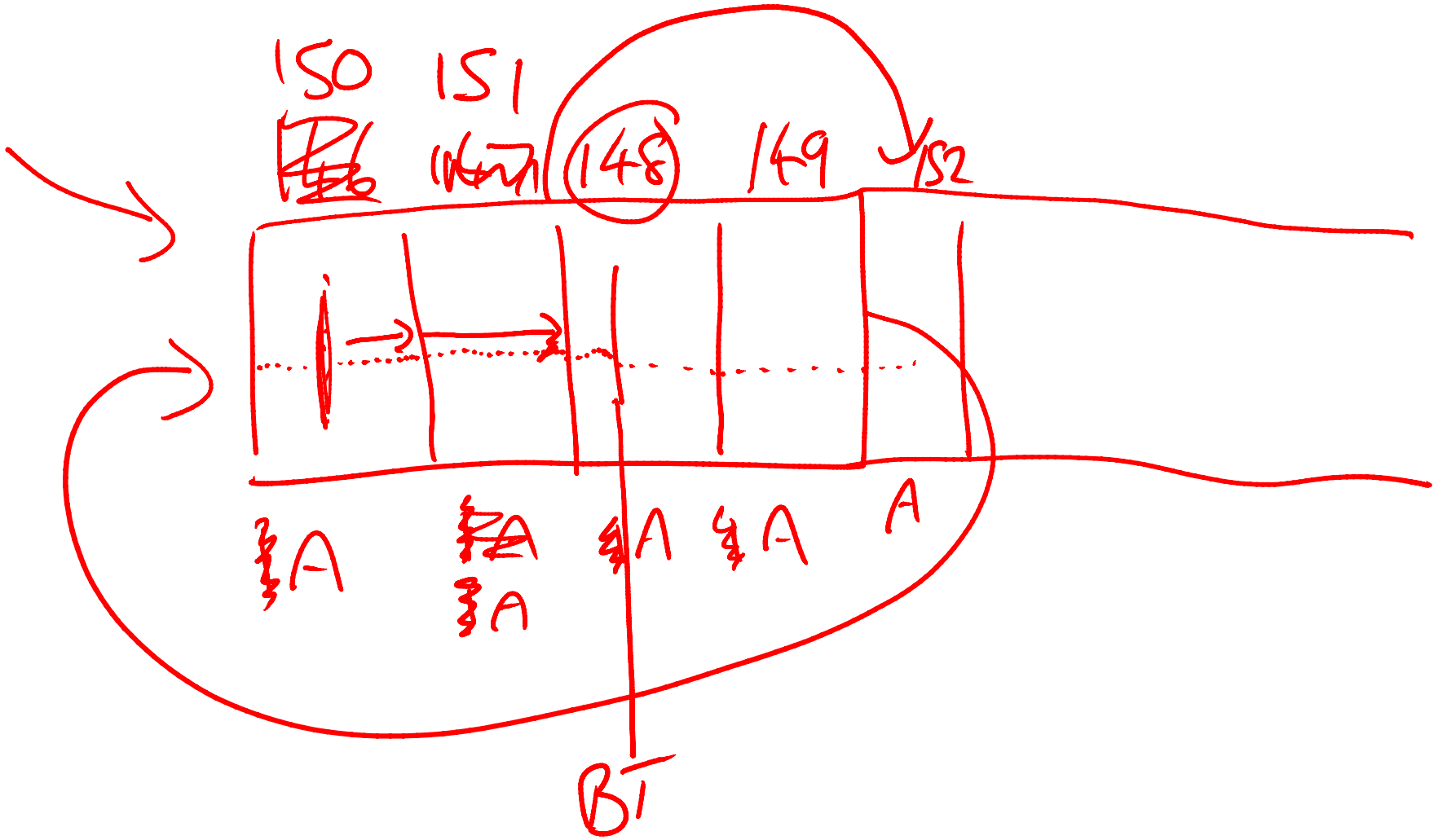


M4S17 Explaining how regular checkpoints may overload the IO subsystem and doing more frequent, manual checkpoints may keep the overall IO load below the threshold at which the I/O subsystem is overloaded.

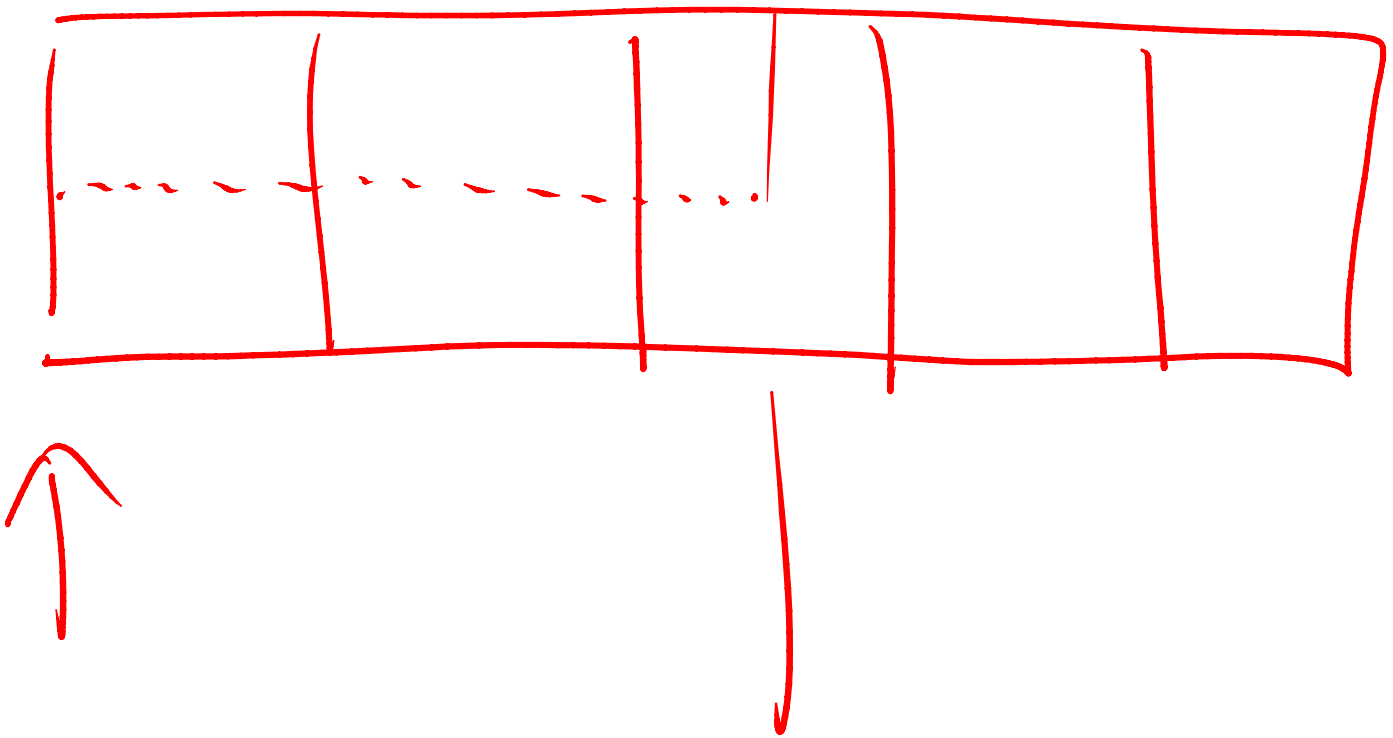

VLF
LOG BLOCKS
512b - 60KB
LOG RECS
LSN = VLF : LB : LREC



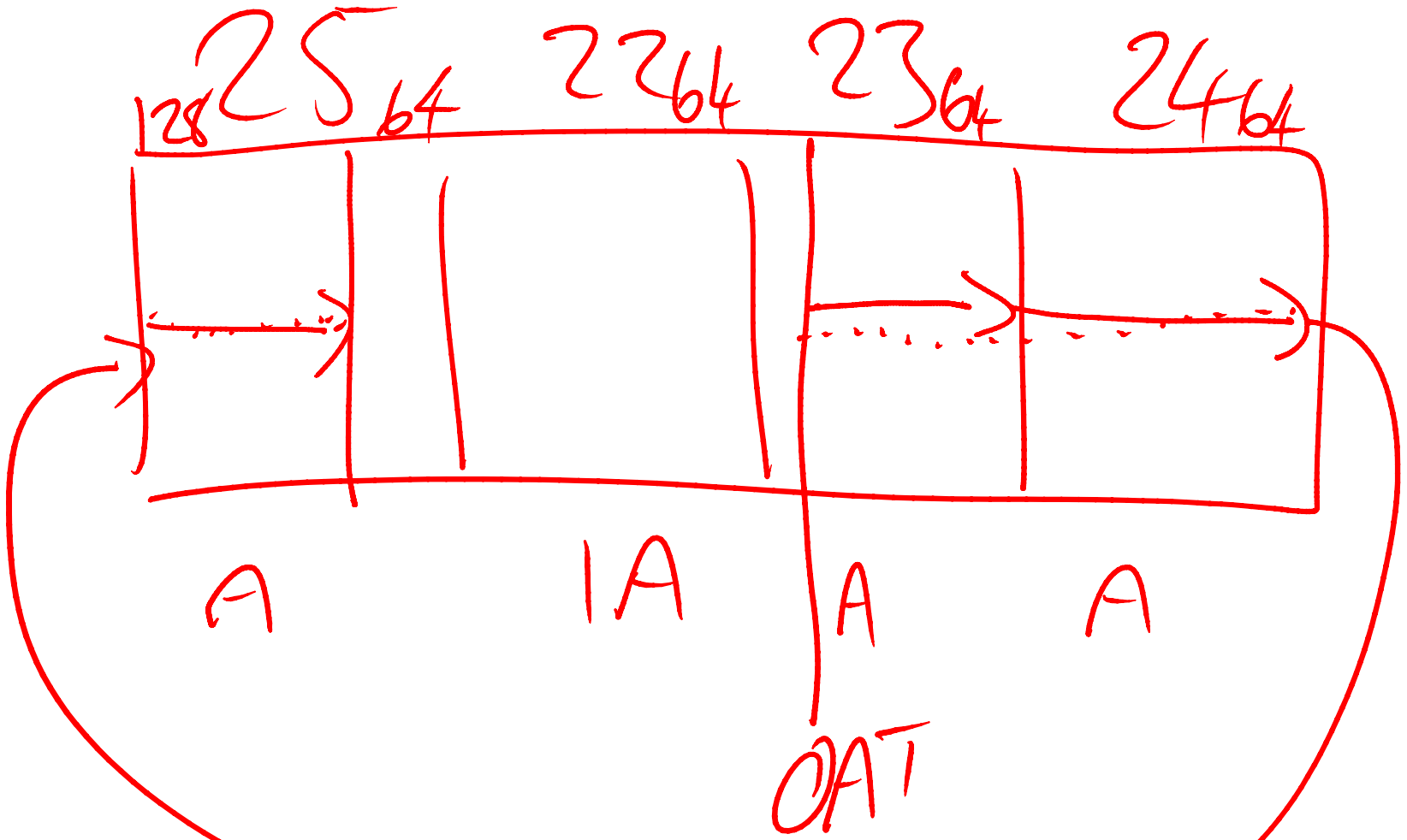
M4 Explaining how log blocks are written to disk.



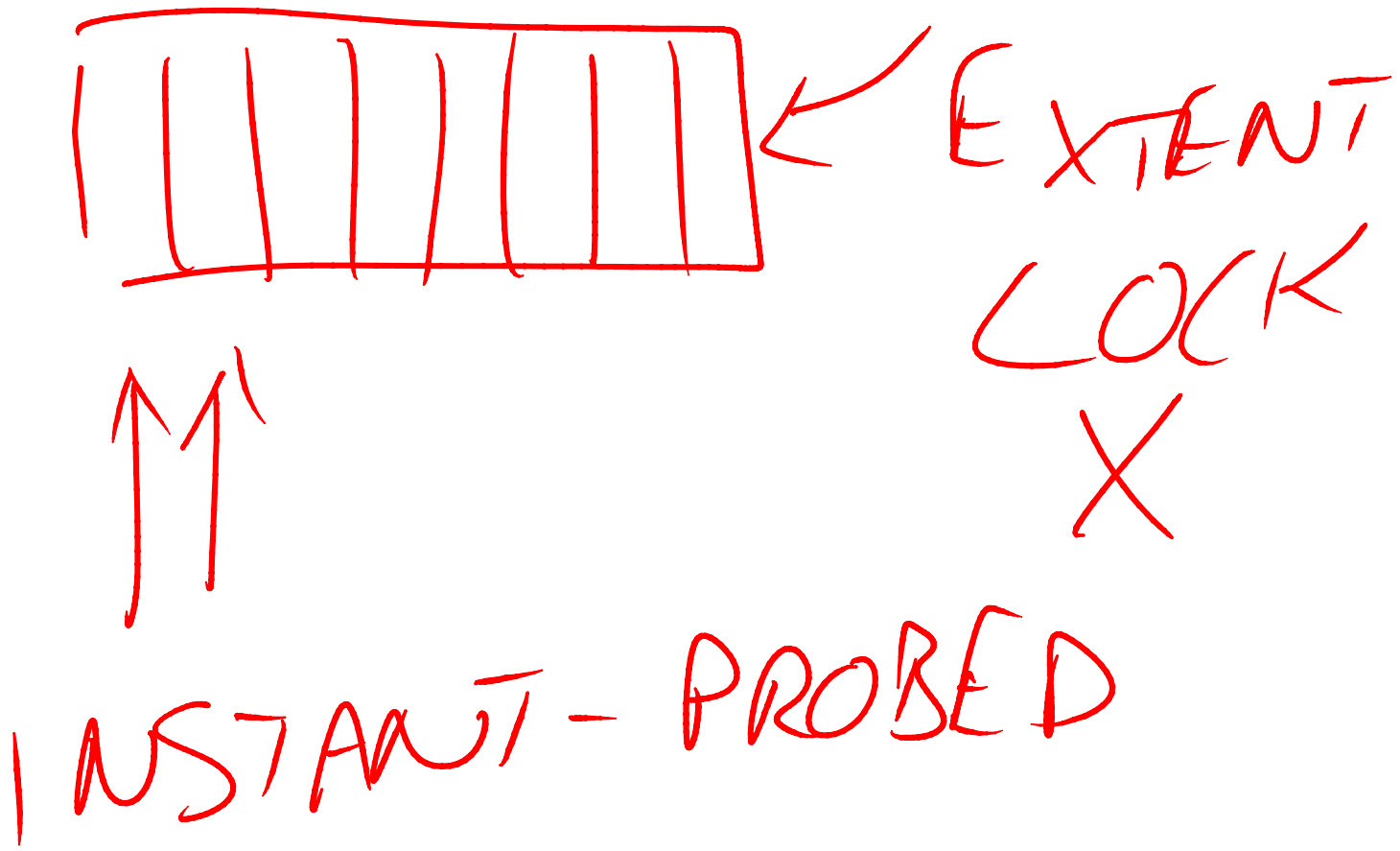
M4S29 working through the circular log demo…



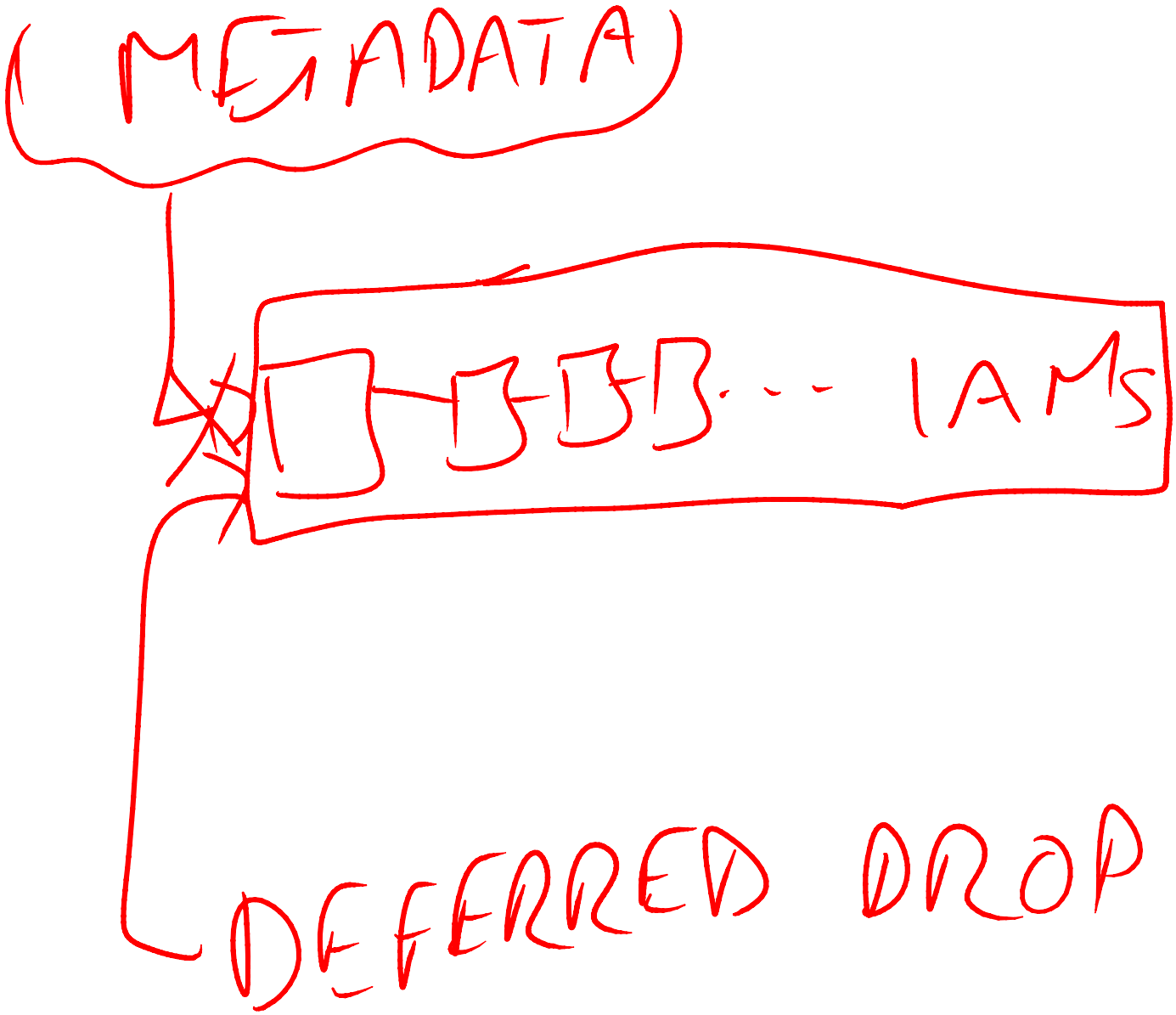
M4S30 When the log hasn’t yet wrapped, crash recovery reads until it finds a zero’d out log block. This is why new portions of the log must be zero initialized.



M4S30 If the log has wrapped, crash recovery reads through VLFs until it comes across a log block with the old parity bits set. Parity bits flip from 64 to 128 and back each time a VLF is re-used.

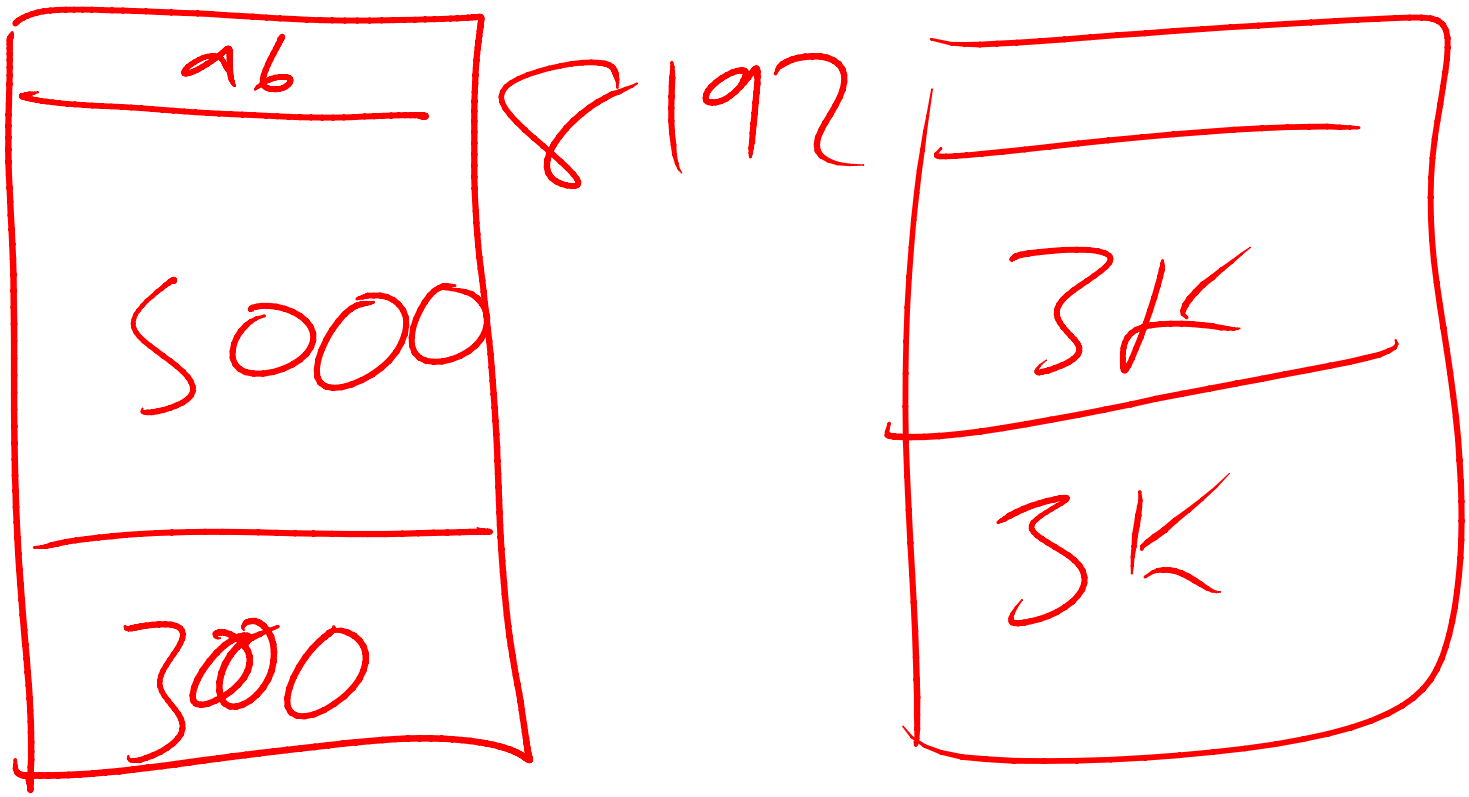


M9 Answering a question about deferred-drop and why it's there. Because extent locks have to be held during drop/truncate operations and it's possible to run out of lock memory.
Deallocating an extent means taking an X allocation lock and then instant-probing all page locks to ensure no-one's holding a page lock.

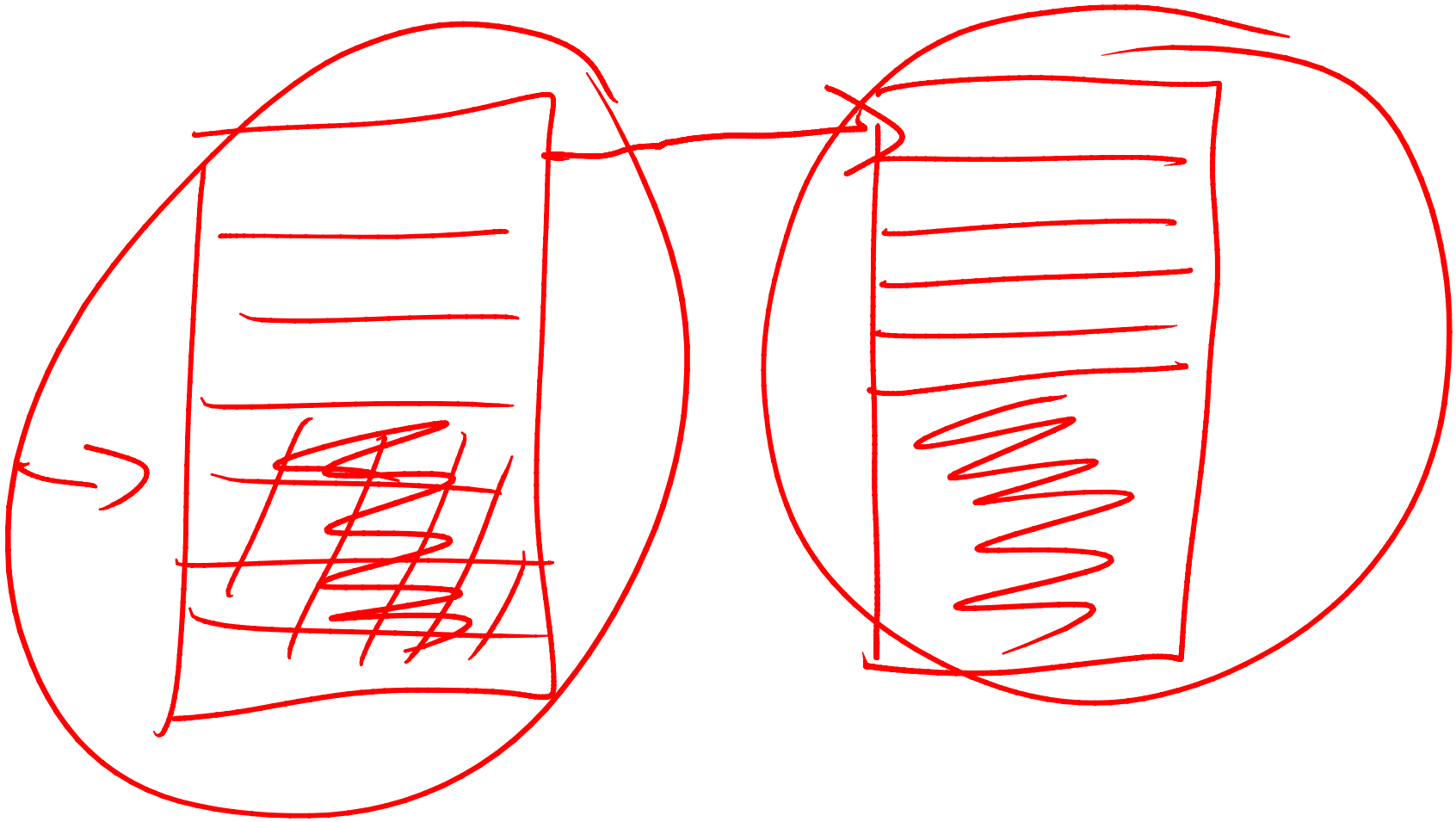


M9 Showing how deferred drop unhooks allocations from metadata and puts them on the deferred-drop queue.

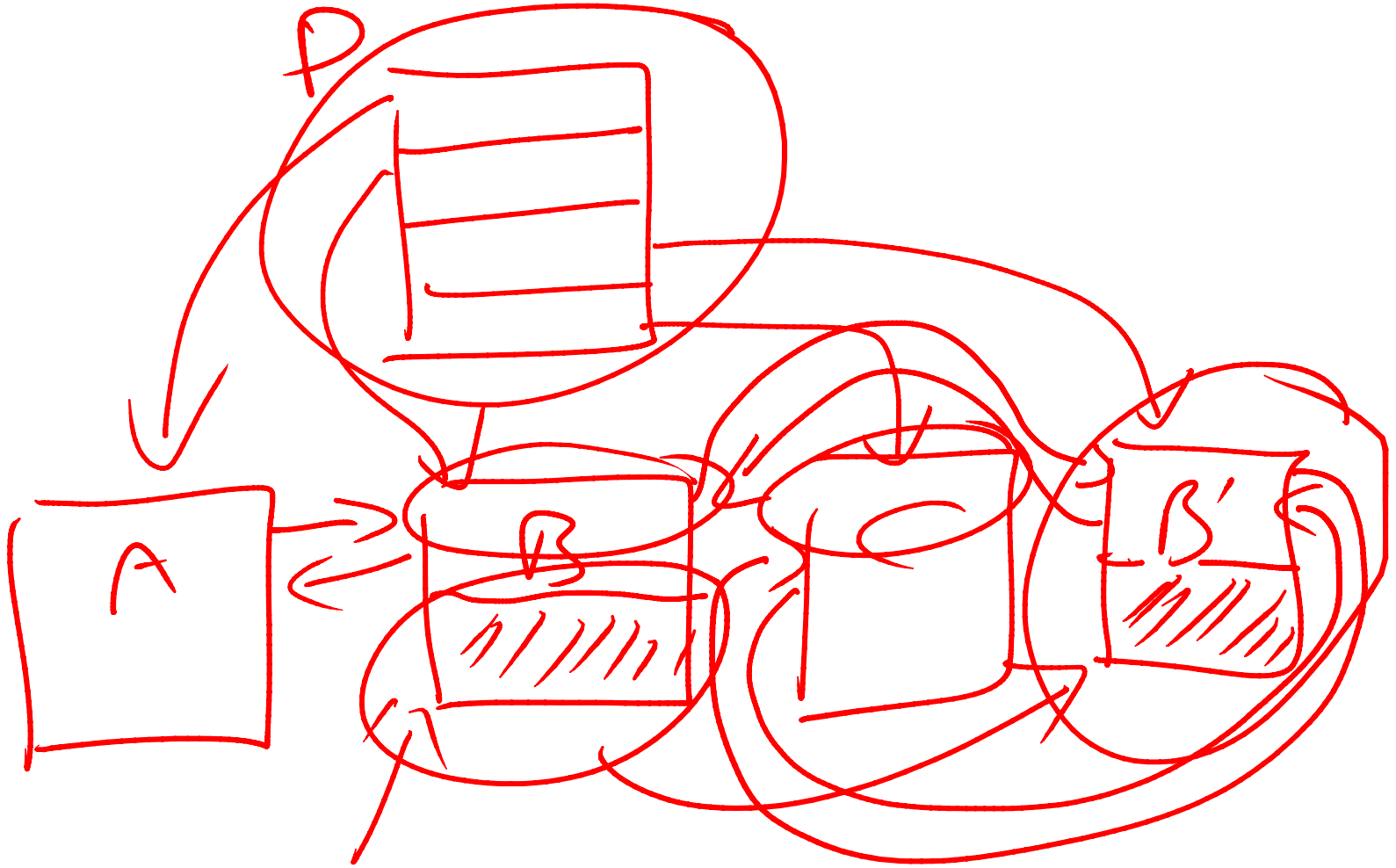
M9S5 Explaining how the Storage Engine finds a record on a page using binary search instead of brute-force cracking open each record in sequence.



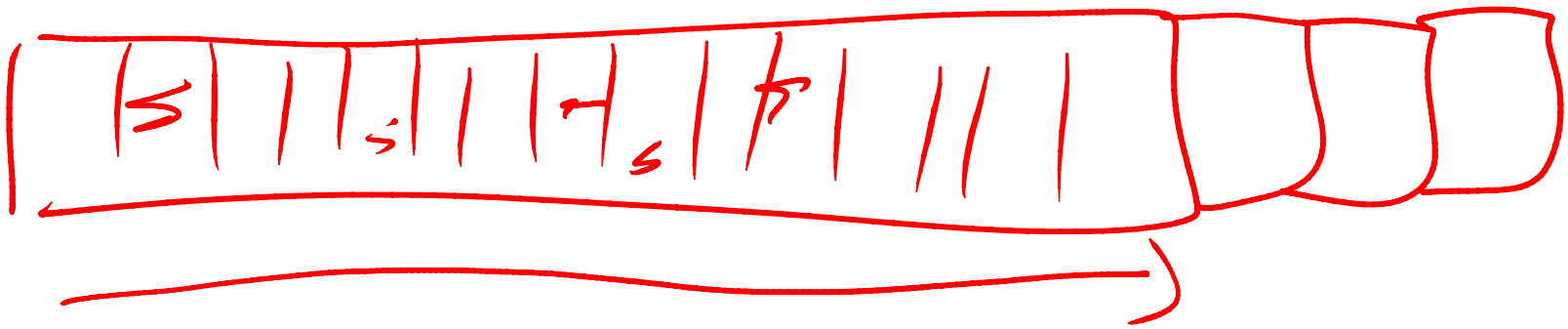
M9S14 Explaining wasted space from fixed-size large records.



M9S14 Explaining wasted space after a page split.



M9S17

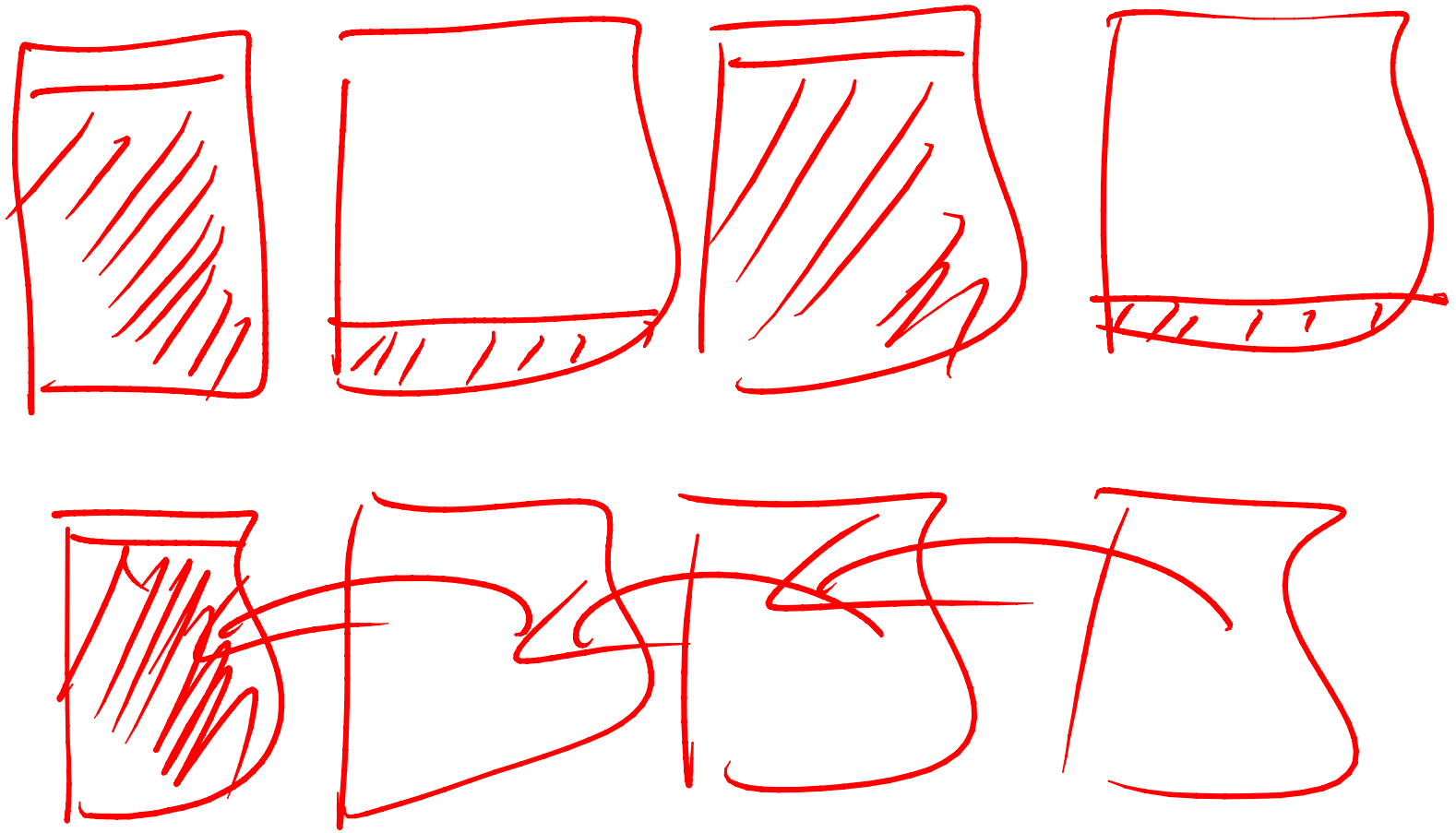


M9 Explaining how in an append-only insert pattern, deleted records earlier in the index won't have their space reused.

RFC SIZE	NO SPLIT	SPLIT	X
1000	1244	6772	5x
100	344	5964	18x
10	256	10364	45x

M9S21 demo results

M9S22 Example of MySpace homepage comments table fragmentation.



M9S42 Explaining how the SAP databases would have a no-space, lots-of-space, no-space pattern in their indexes (top of slide) and my pathological case avoidance (bottom of slide) would mean nothing happened.