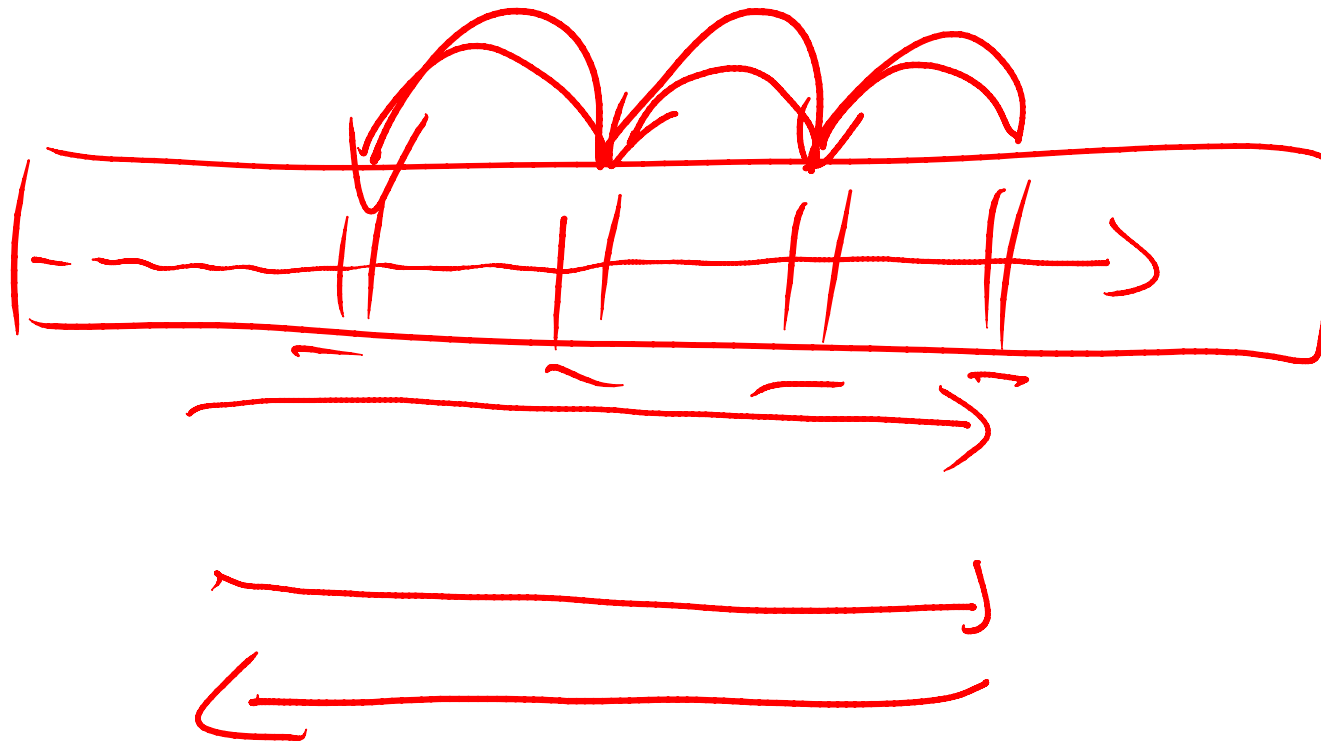
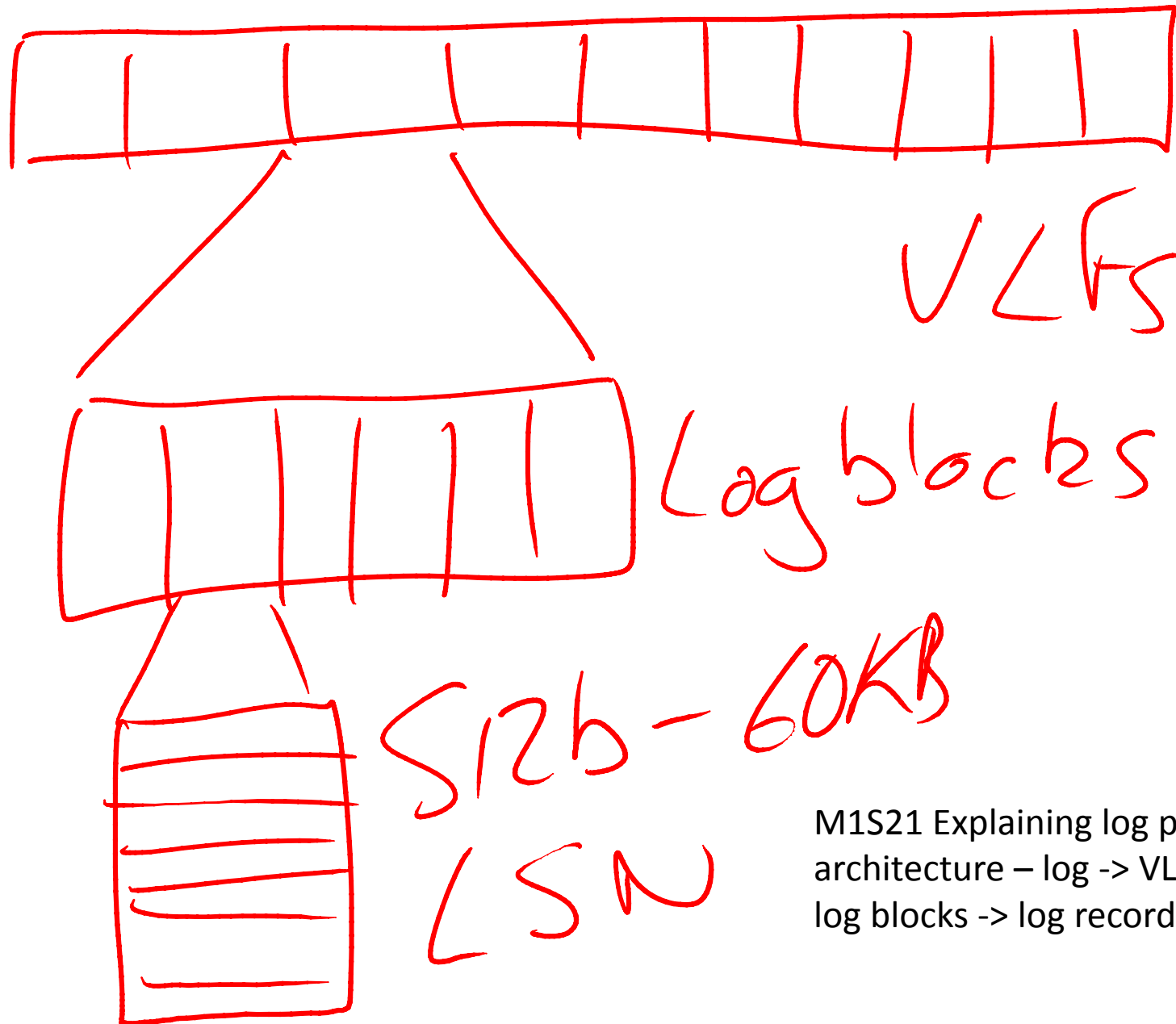




M1S21 Explaining how log IO is sequential write always but can be random or sequential read. Sequential reads are from log readers (e.g. log backup) and random reads are from transactions rolling back (or crash recovery). Log records in a transaction are chained backwards and are undone in reverse order from when they were generated.

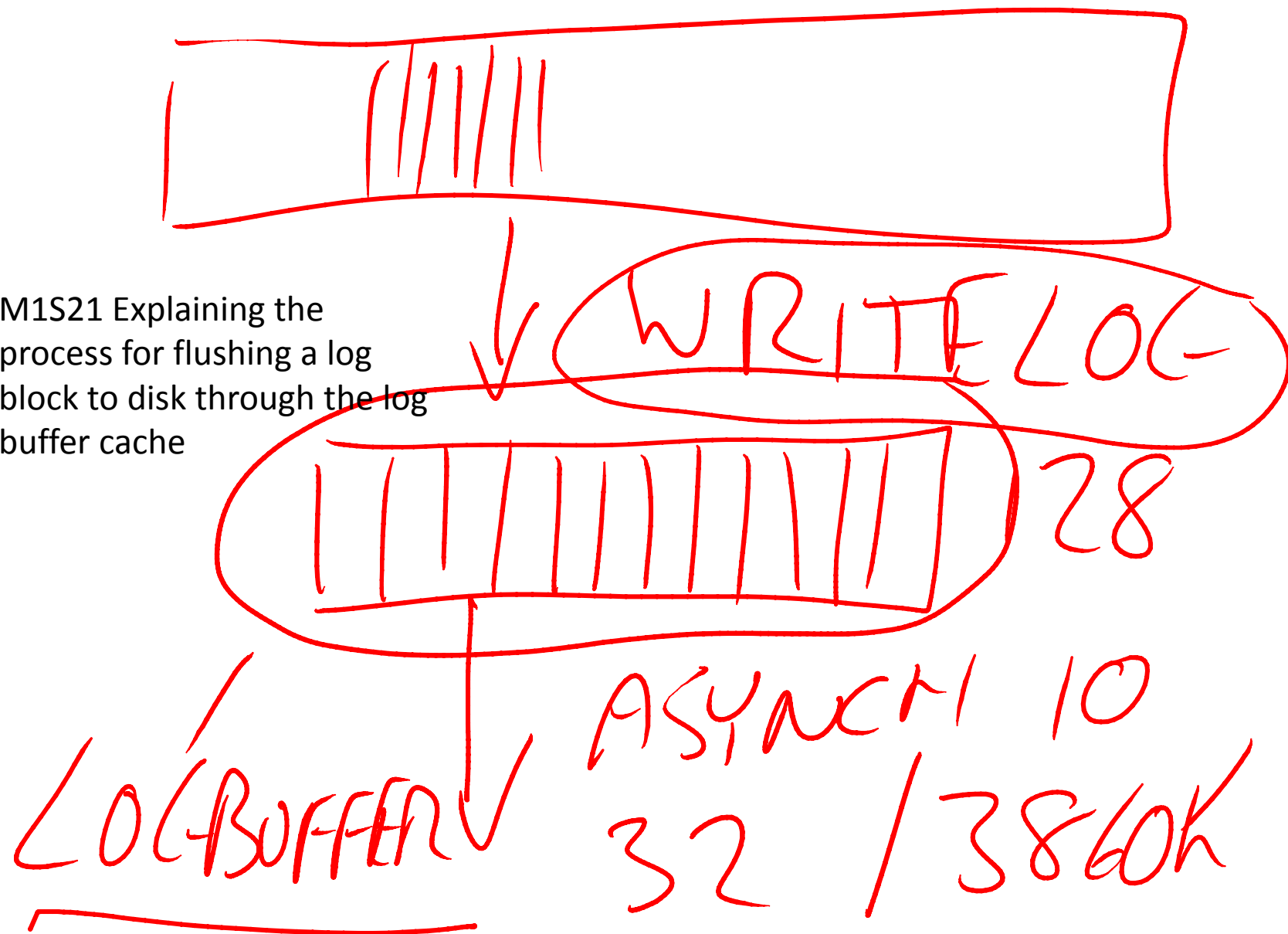
M1S21 Explaining how a page split moves (usually) half the rows from the page that needs space into a new page – causing 50% free space in each page.

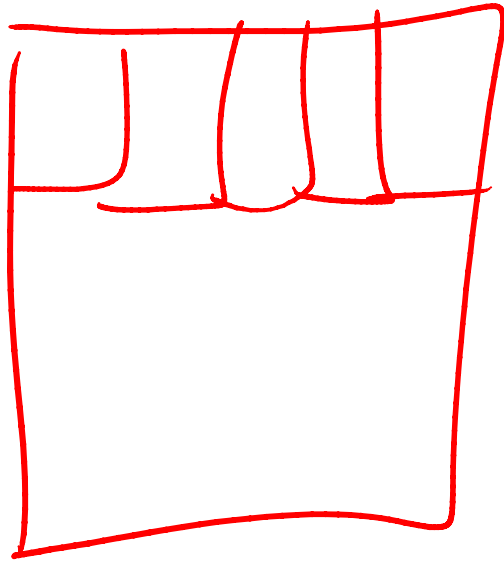




M1S21 Explaining log physical
architecture – log -> VLFs ->
log blocks -> log records

M1S21 Explaining the
process for flushing a log
block to disk through the log
buffer cache



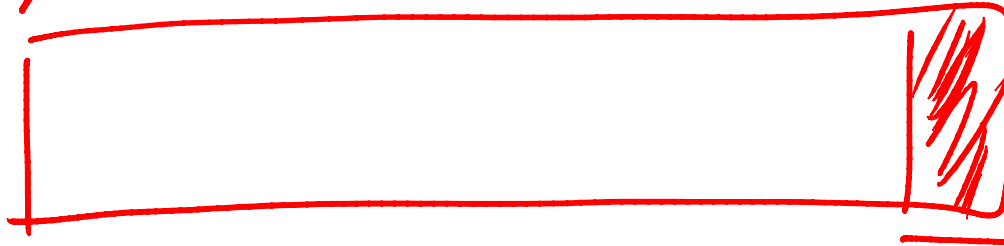
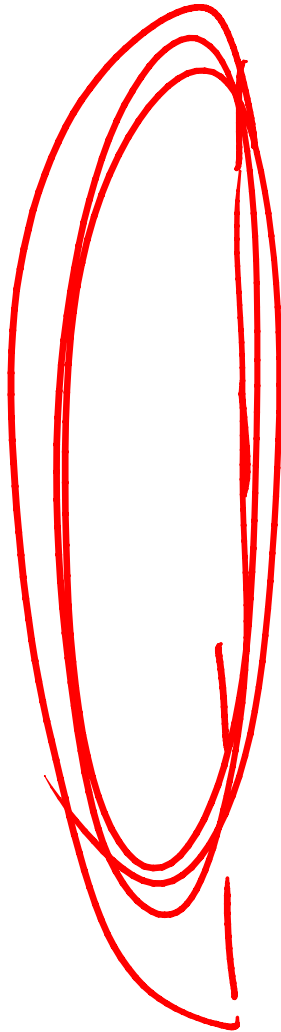


M1S23 Explaining the buffer pool is a big block of memory with a 'clock hand' sweeping through all the buffers implementing a least-recently-used algorithm to determine which pages to drop to make room.

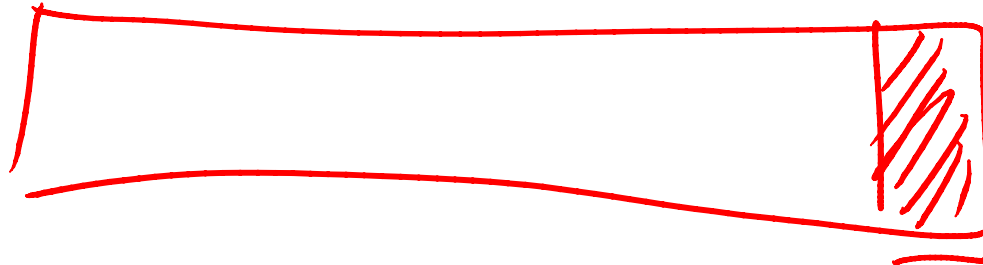


T 1117

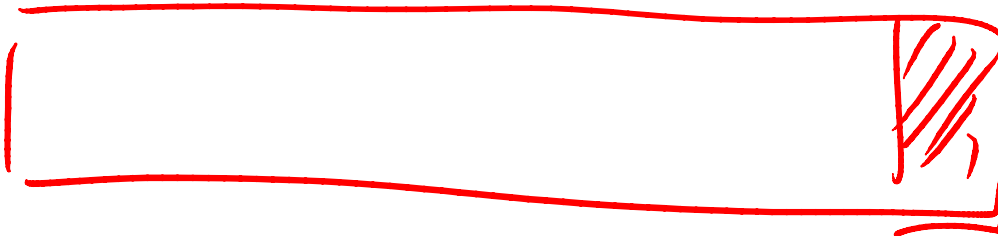
M1S23/4 Round-robin allocation and proportional fill. T1117 makes all files in a filegroup grow at the same time



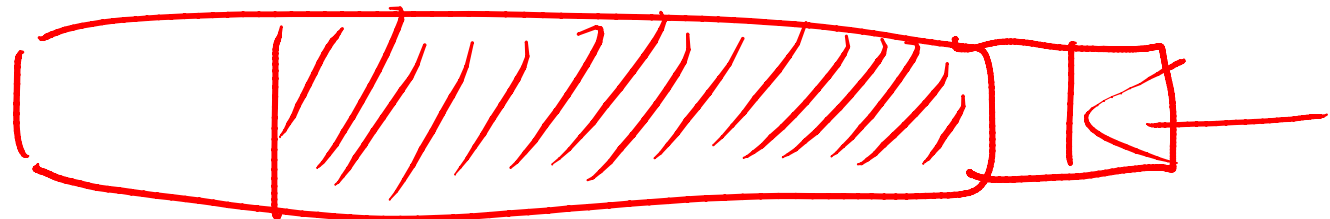
2040



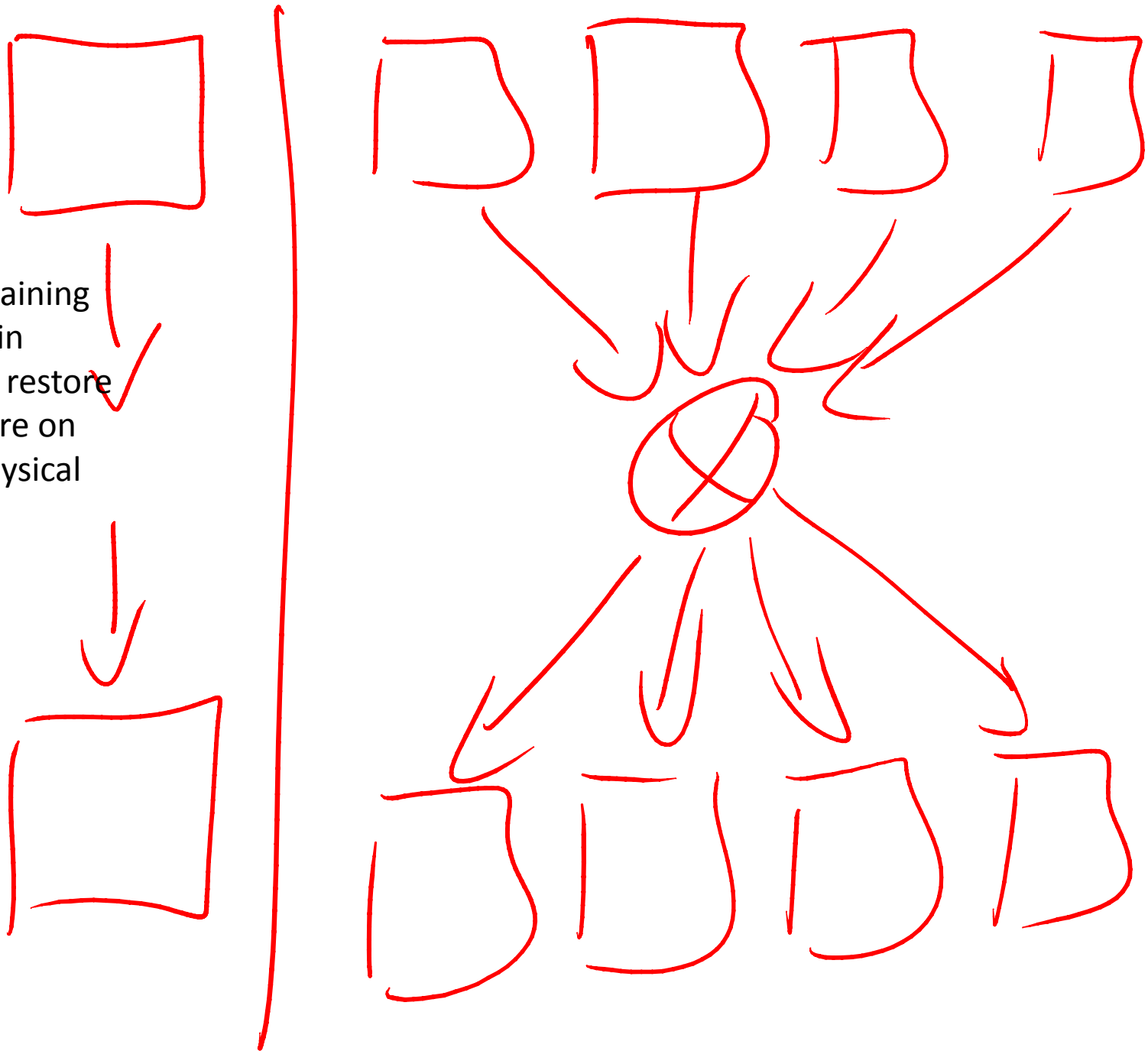
2040

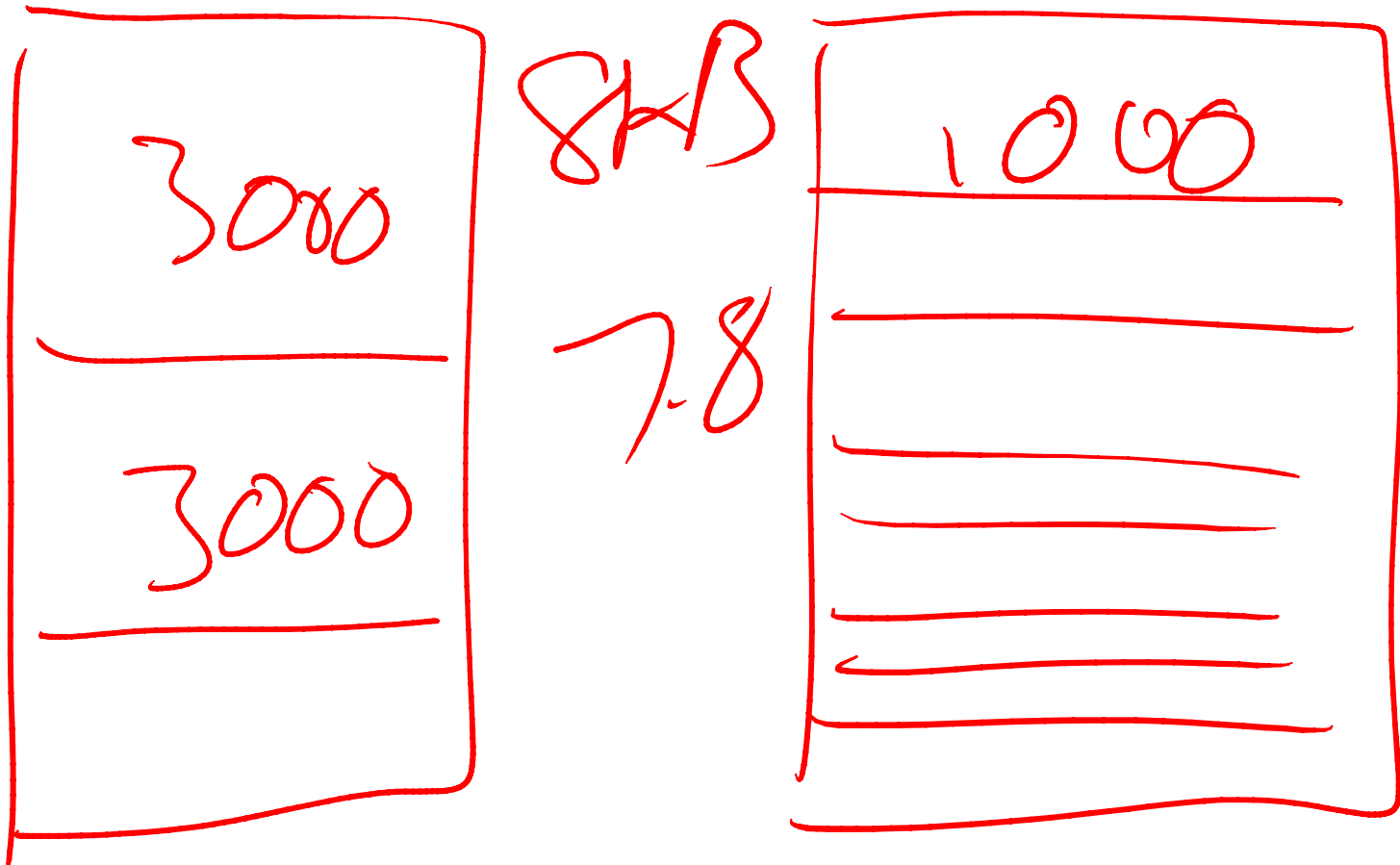


2040



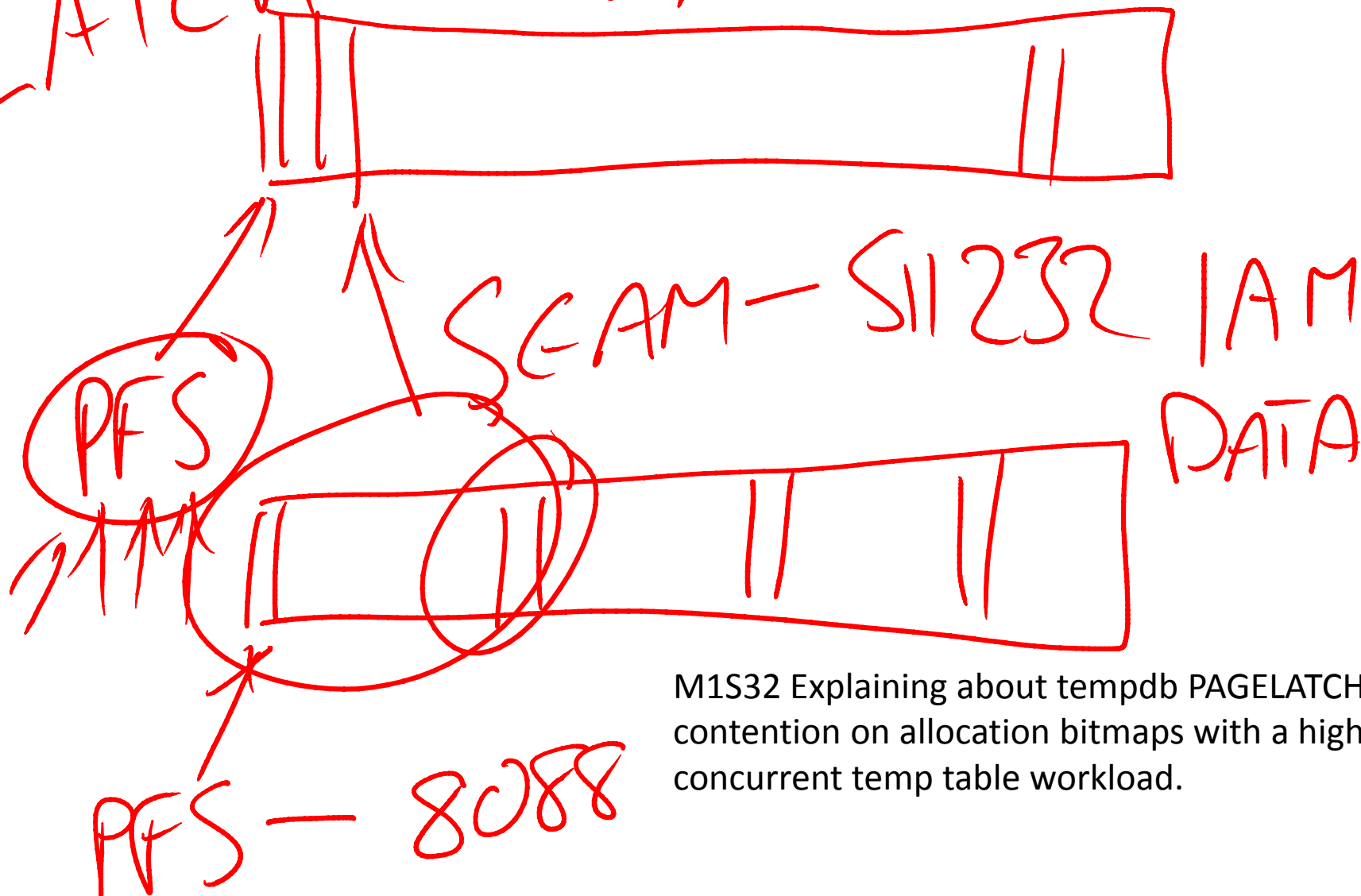
M1S25 Explaining
parallelism in
backup and restore
if the files are on
separate physical
paths





M1S27 Explaining about having LOB data inline with the table row vs somewhere else can lead to low data density for commonly used columns. Sometimes better to have the LOB data stored separately from the commonly used columns.

LATCH SH EX



M1532 Explaining about tempdb PAGELATCH contention on allocation bitmaps with a highly concurrent temp table workload.

<8cores: 1:1

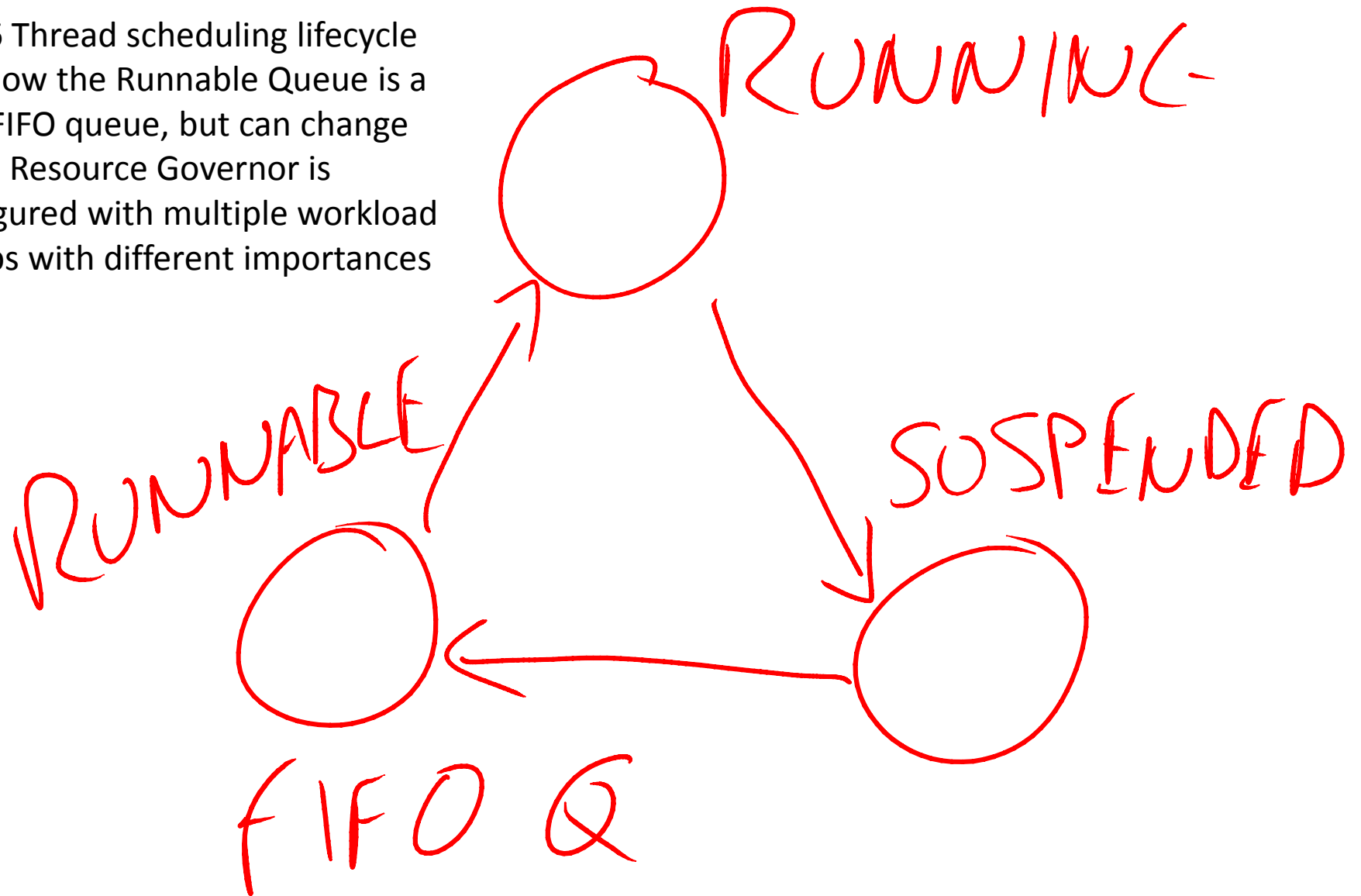
>8cores: 8, +4

M1S33 Explaining Bob Ward's tempdb data file algorithm (which I agree with).

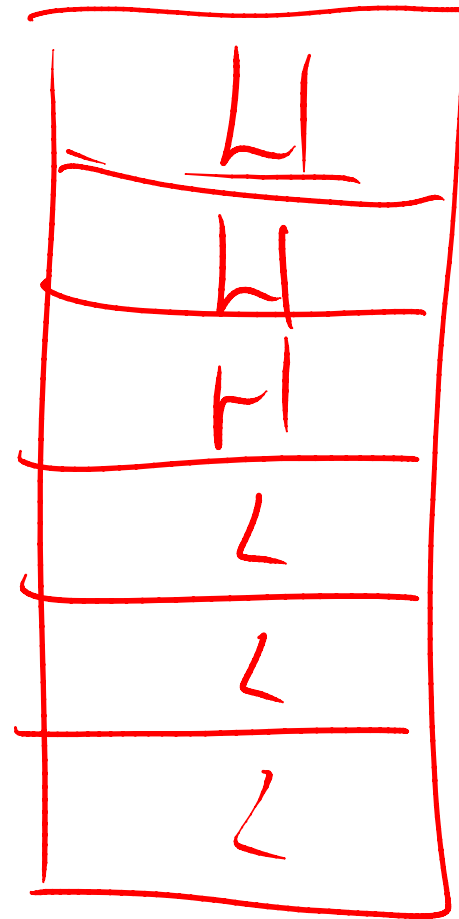
Less than 8 cores: #data files = #cores

More than 8 cores: #data files = 8. Increase in chunks of 4 if contention continues.

M2S6 Thread scheduling lifecycle
and how the Runnable Queue is a
true FIFO queue, but can change
when Resource Governor is
configured with multiple workload
groups with different importances



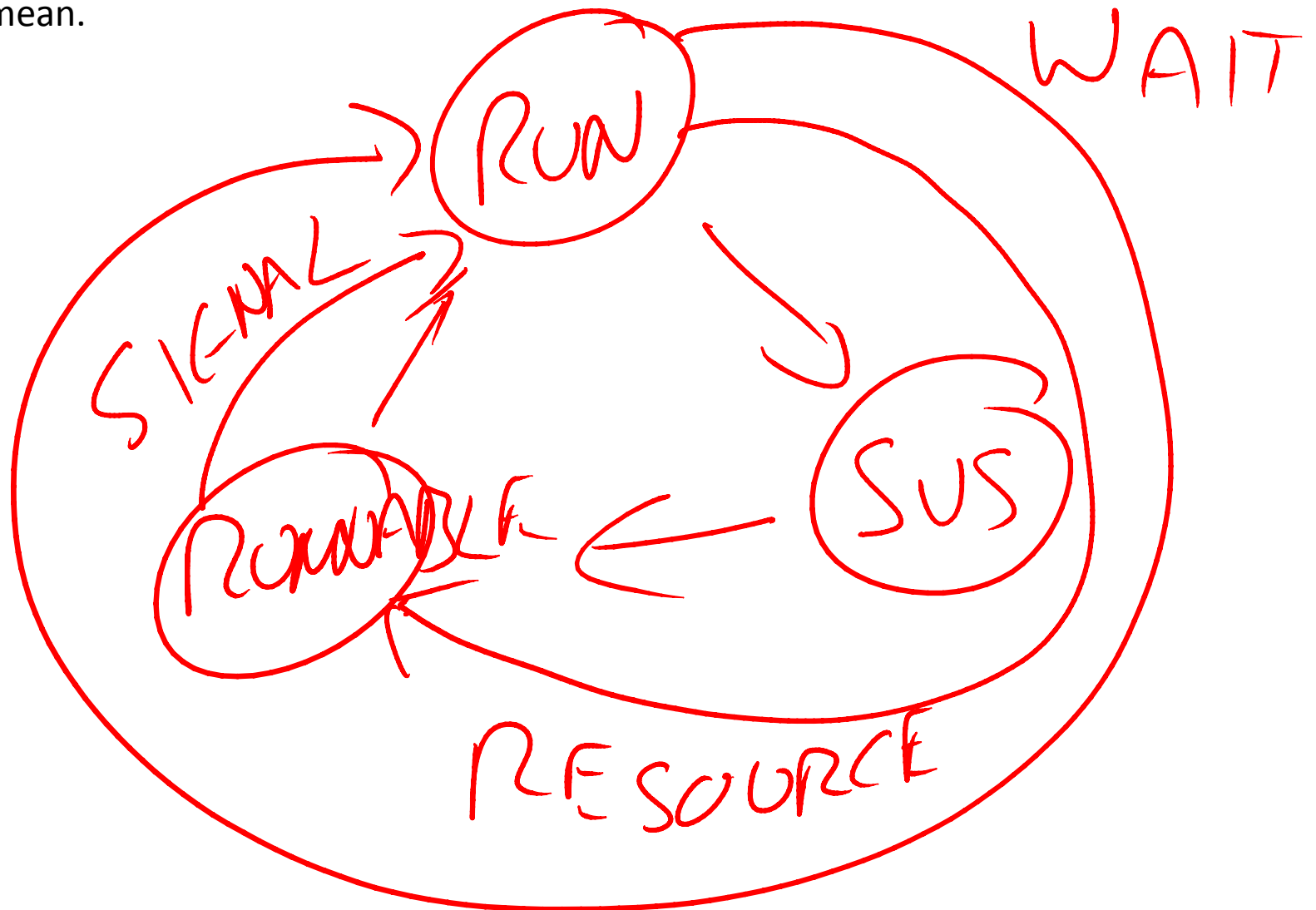
M2S6 Thread
scheduling lifecycle
and how the
Runnable Queue is
a true FIFO queue,
but can change
when Resource
Governor is
configured with
multiple workload
groups with
different
importances

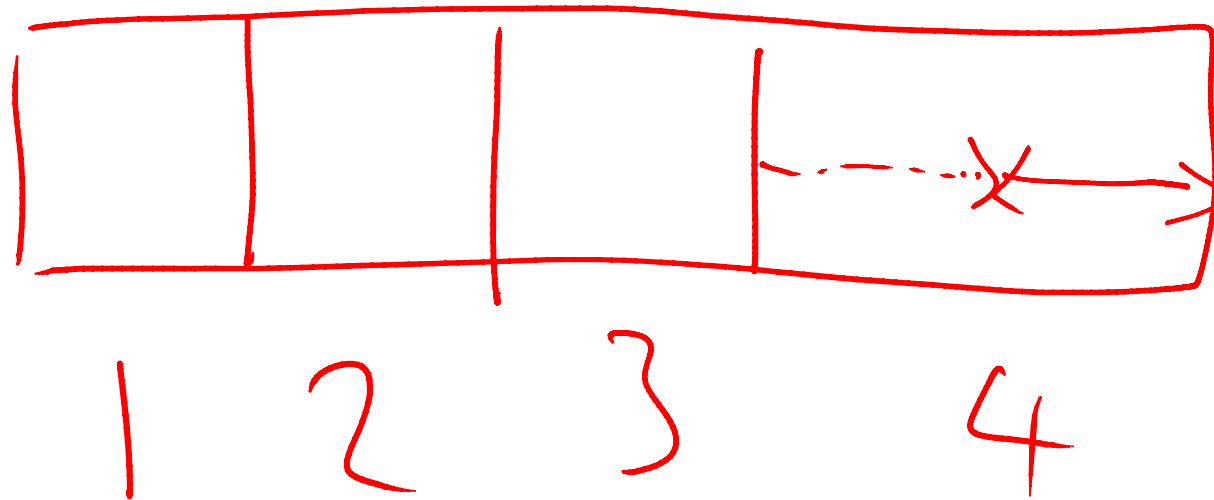


H:M:L
9:3:1

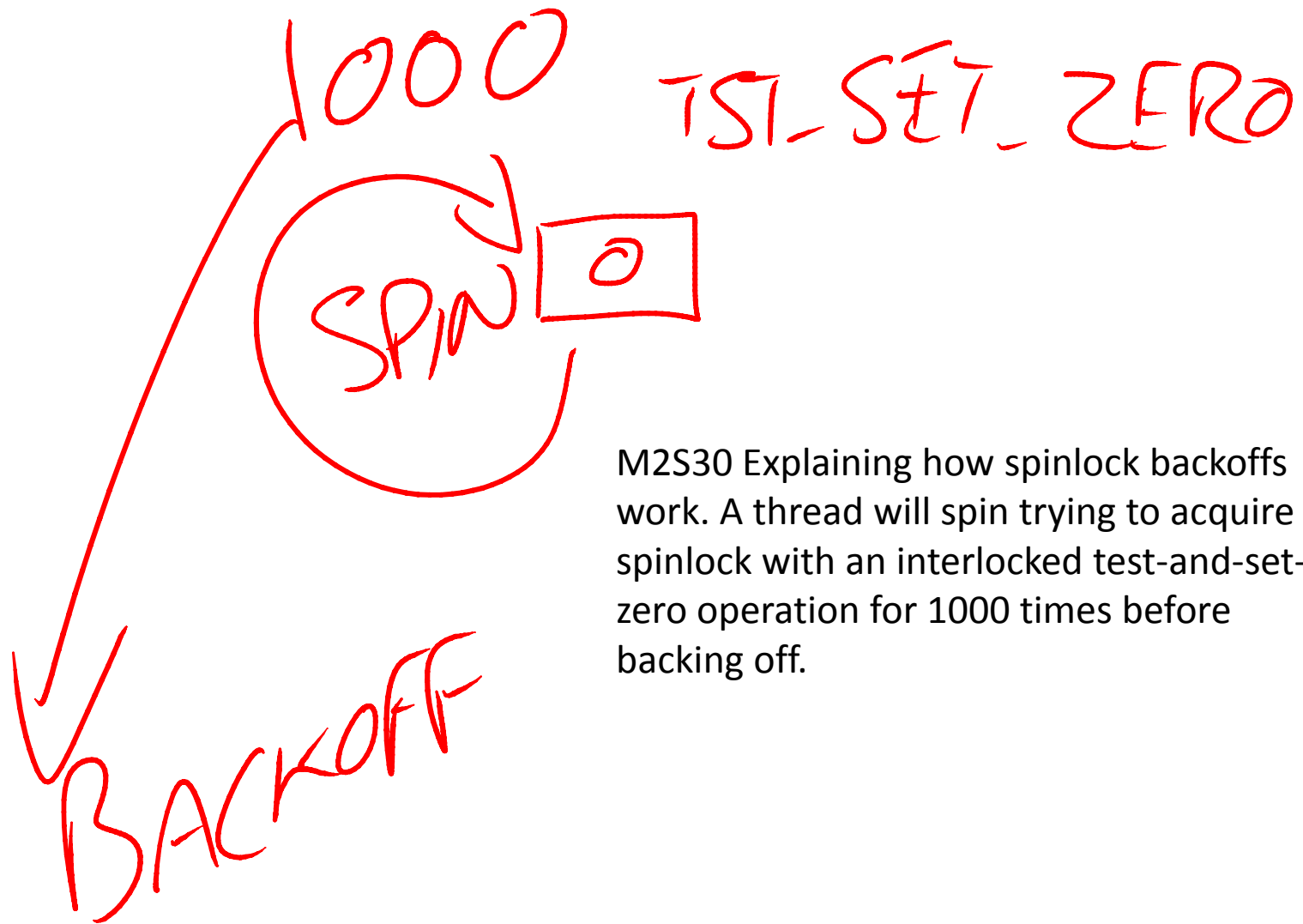
H

M2S11 Showing what the three wait times mean.





M2S23 Explaining CXPACKET waits. If a non-even distribution of work is given to the threads, the control thread and all non-busy thread accumulate CXPACKET wait times.



M2S30 Explaining how spinlock backoffs work. A thread will spin trying to acquire a spinlock with an interlocked test-and-set-if-zero operation for 1000 times before backing off.

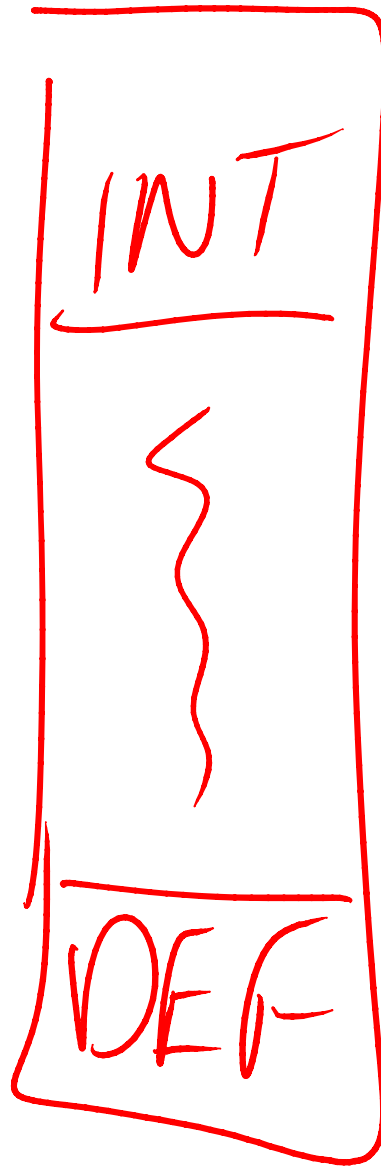
32

M8S9 Explaining the priority of MAXDOP settings.
Server-wide is weakest as it can be over-ridden with a query hint. Resource Governor workload group setting cannot be overridden.

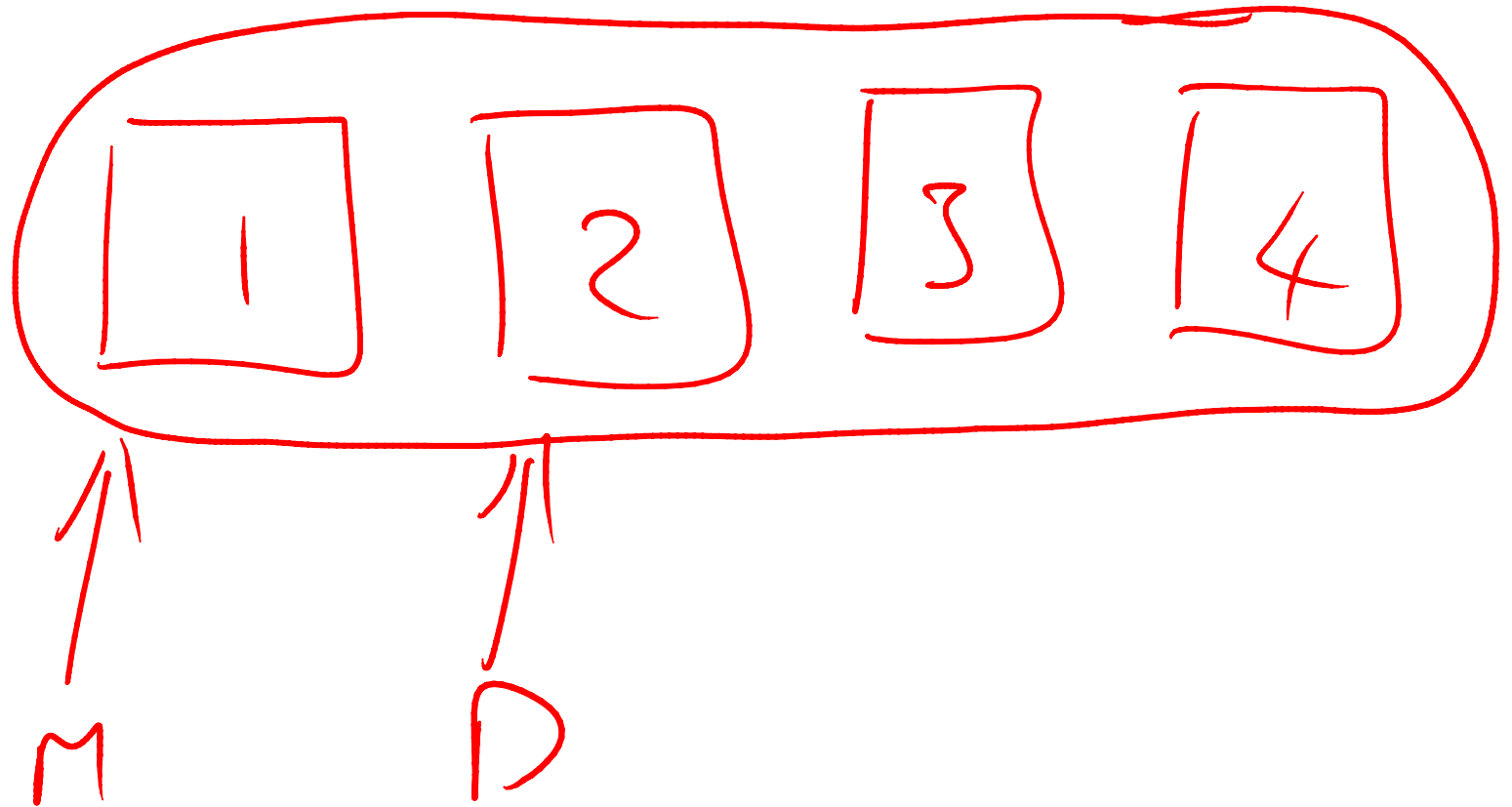
4 > MAXDOP RG

8 Q. HINT

1 INSTANCE

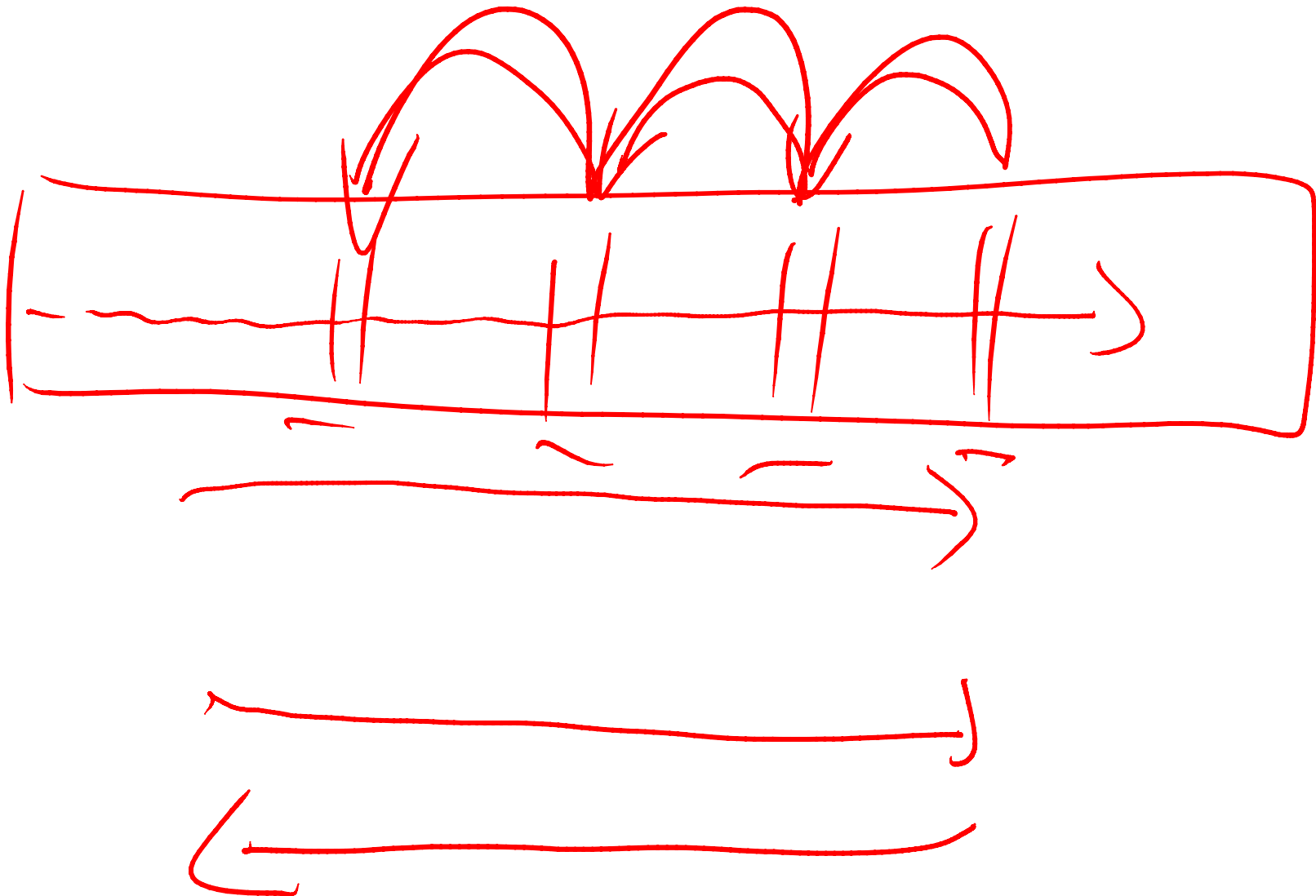


M8S10: Explaining the relative priority of the resource pools. Internal is at the top, default is at the bottom.



M8S11 Explaining how CPU governing is per-scheduler rather than across all CPUs which can lead to non-intuitive results.

M11 Explaining how online index operations work with momentary S and Sch-Modification locks, plus antimatter records.
See <http://www.microsoft.com/technet/prodtechnol/sql/2005/onlineindex.mspx>

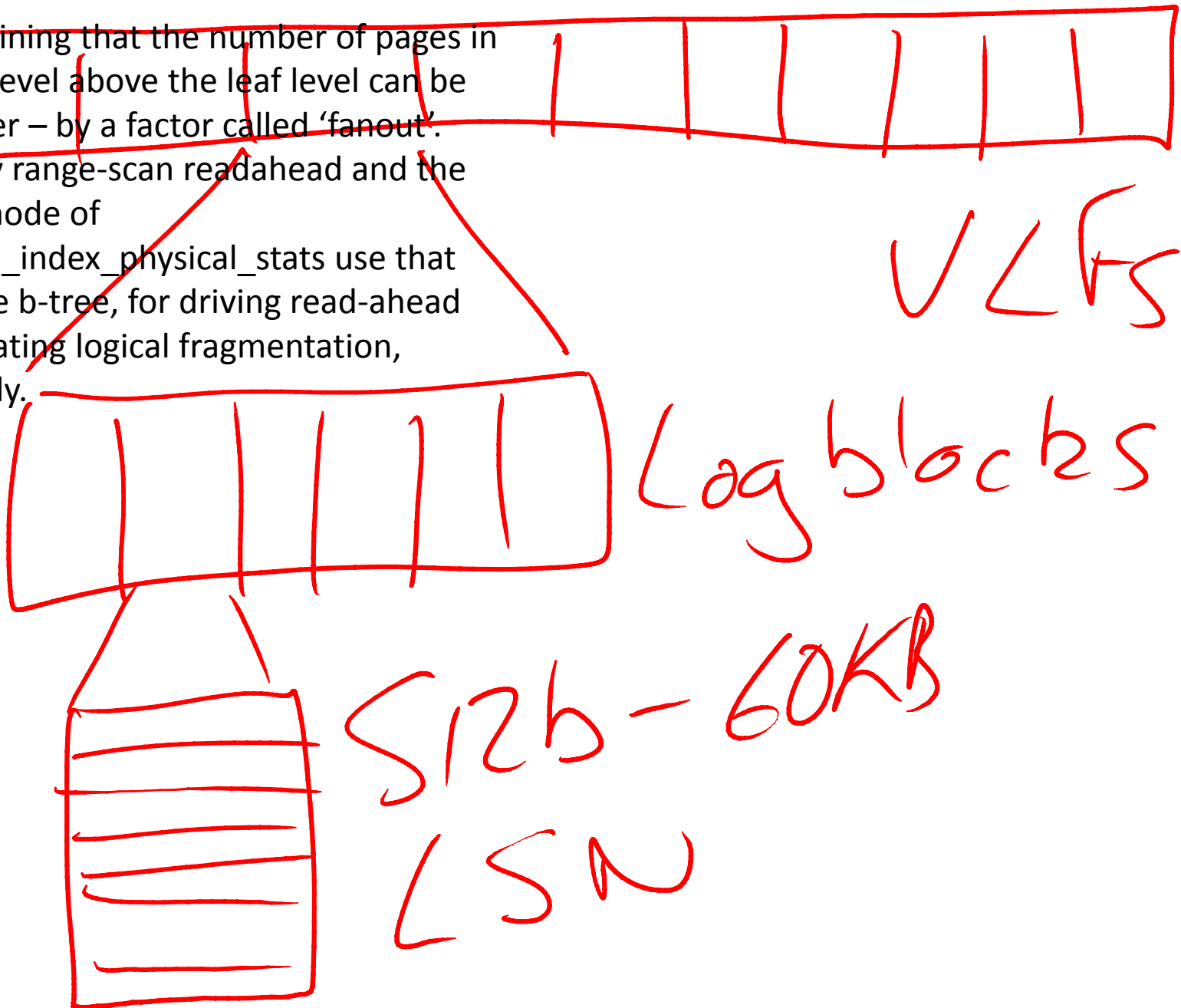


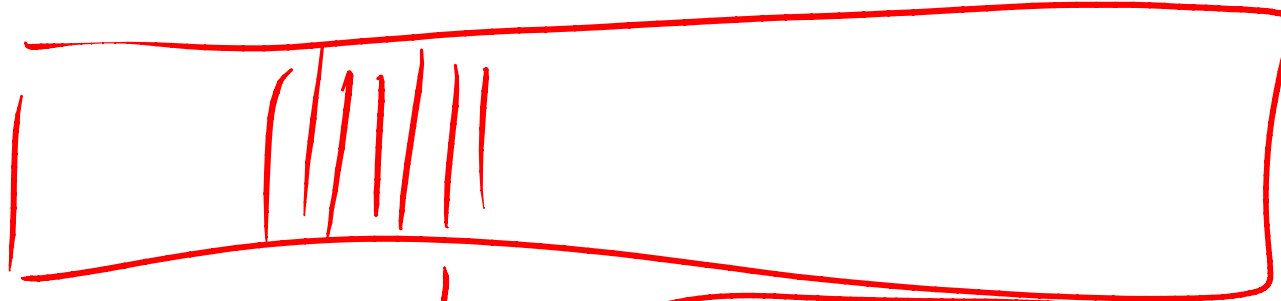


M11 Explaining how the extent locks taken during a truncate or drop used to lead to potential out-of-memory issues for large tables. Since SQL 2000 SP3 there has been a deferred-drop background task that drops/truncates large sets of allocations in chunks to avoid out-of-memory problems.

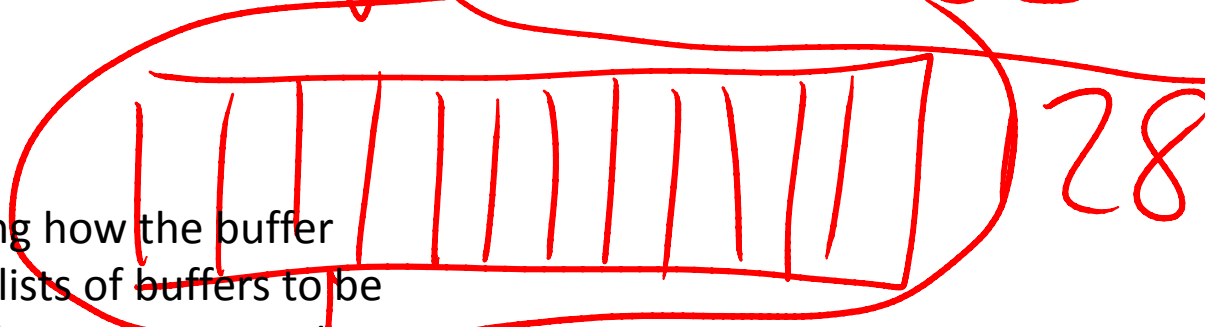
M11 Explaining that the number of pages in the index level above the leaf level can be much fewer – by a factor called ‘fanout’.

This is why range-scan readahead and the LIMITED mode of sys.dm_db_index_physical_stats use that level of the b-tree, for driving read-ahead and calculating logical fragmentation, respectively.





↓ WRITE LOG



M12S6: Explaining how the buffer pool keeps hash lists of buffers to be able to tell what's in memory and what's not. The BUF structures are dumped by DBCC PAGE and also include the amount of free space on the page, maintained in real time.

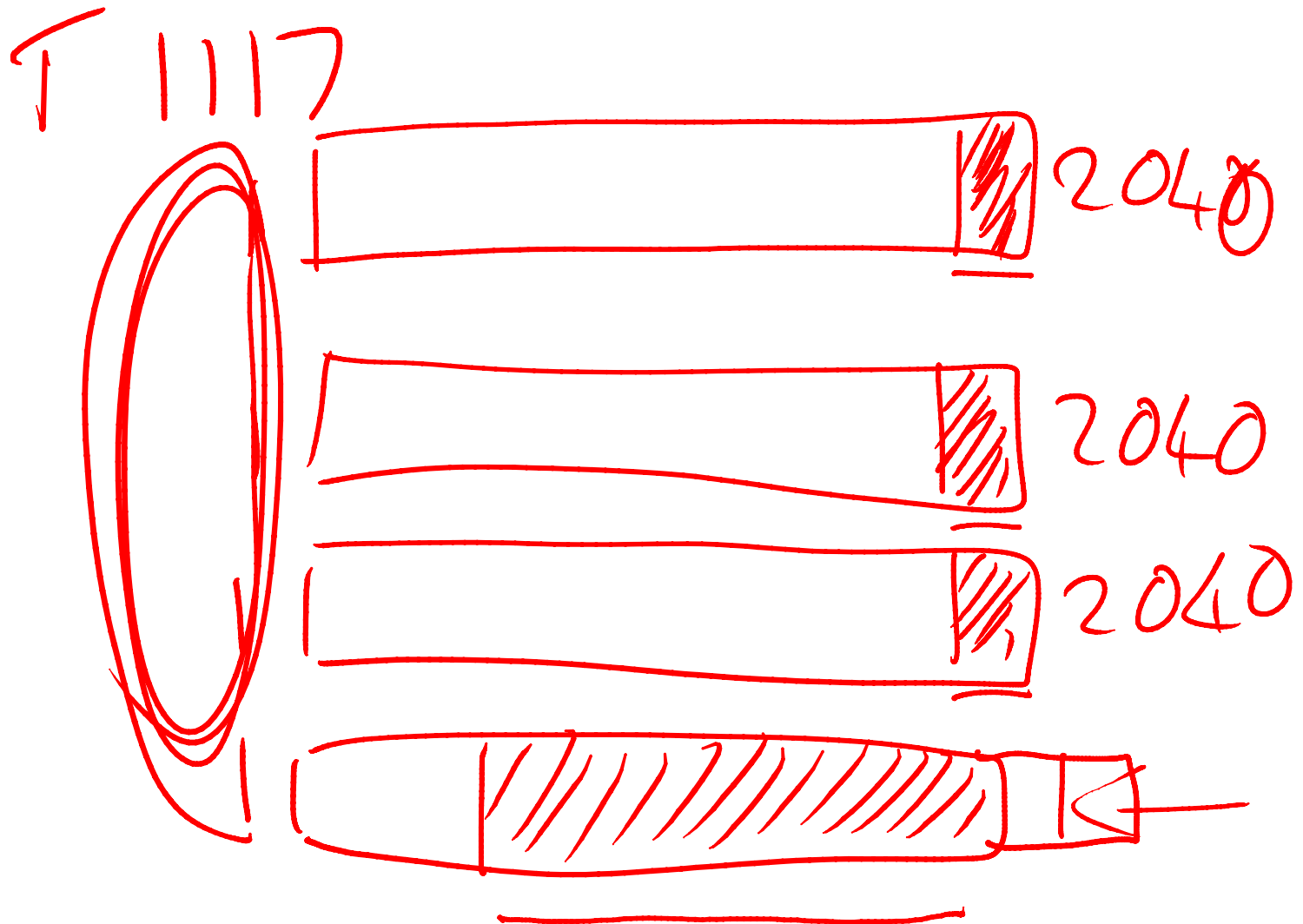
LOG BUFFER

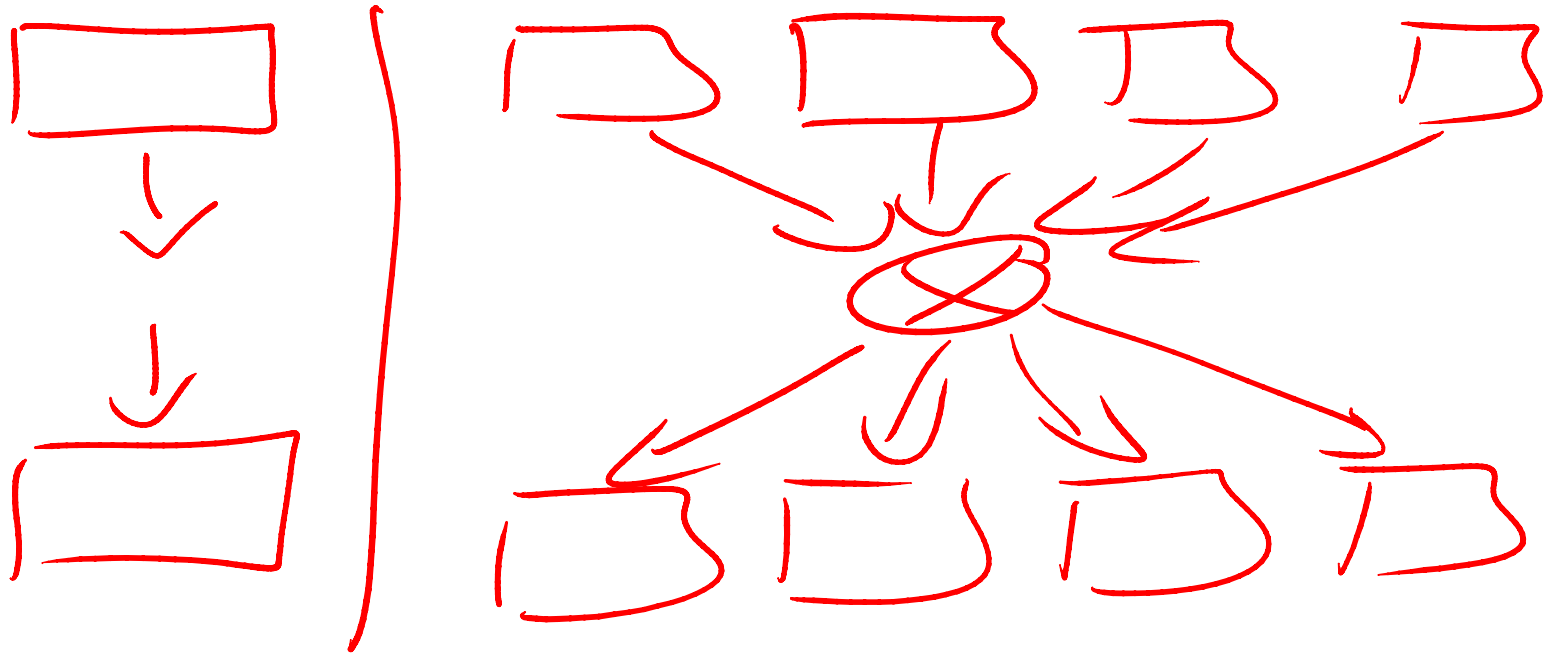
ASYNCH 10
32 / 3860K



M12: Track checkpoints using these three trace flags

M12S12 The version store is split into chunks every minute. The background cleanup task runs every minute and can remove chunks if no transactions exist that could possibly need to see those versions. Long-running transactions can lead to tempdb growth if the version store can't be truncated.





M12 demo: Explaining that a change to an index key column in a page with 4000 byte rows will result in a page split. Non-intuitive.

3000	8MB	1000
3000	78	

M12: Explaining that nested transactions don't exist.