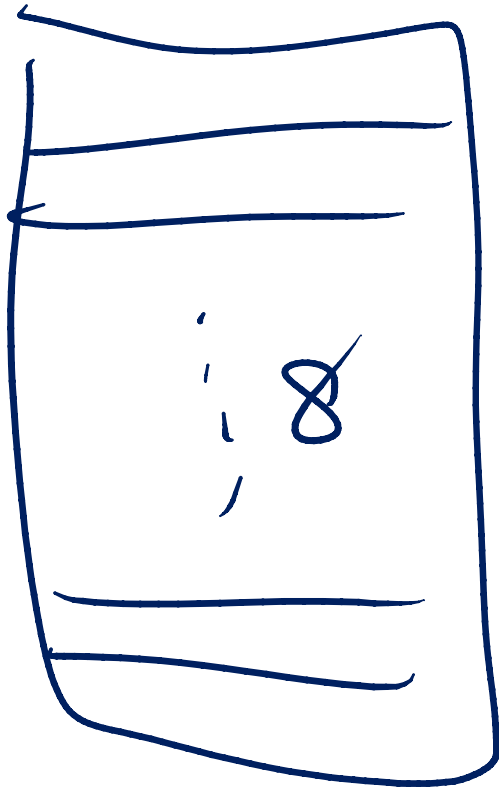
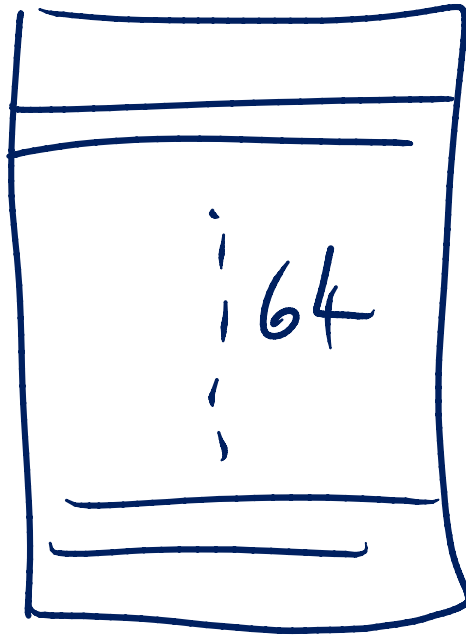
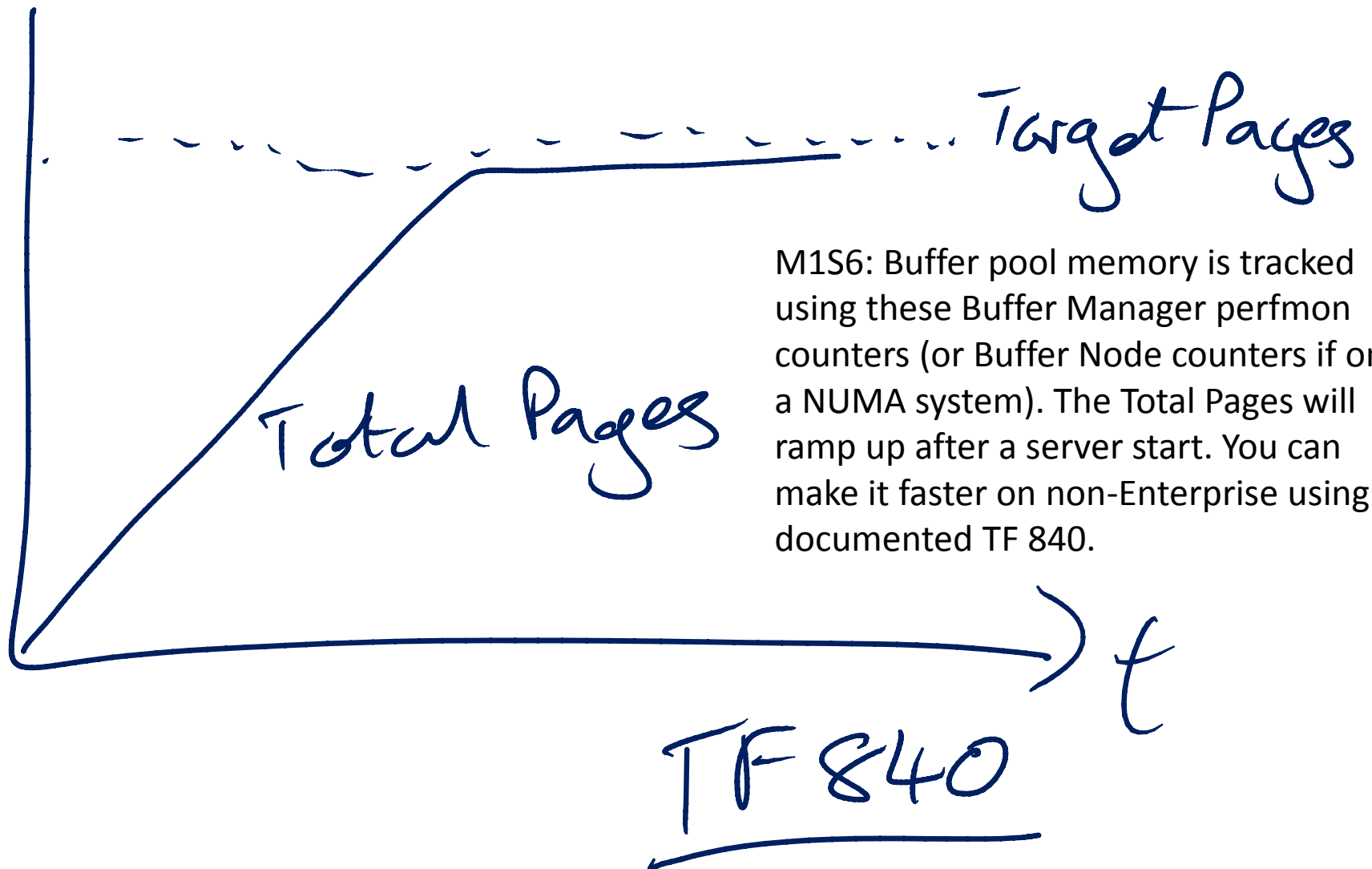




100 24 → OFF Row 0.5%



M1S5 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.



TABLE

INDEX A

10,000

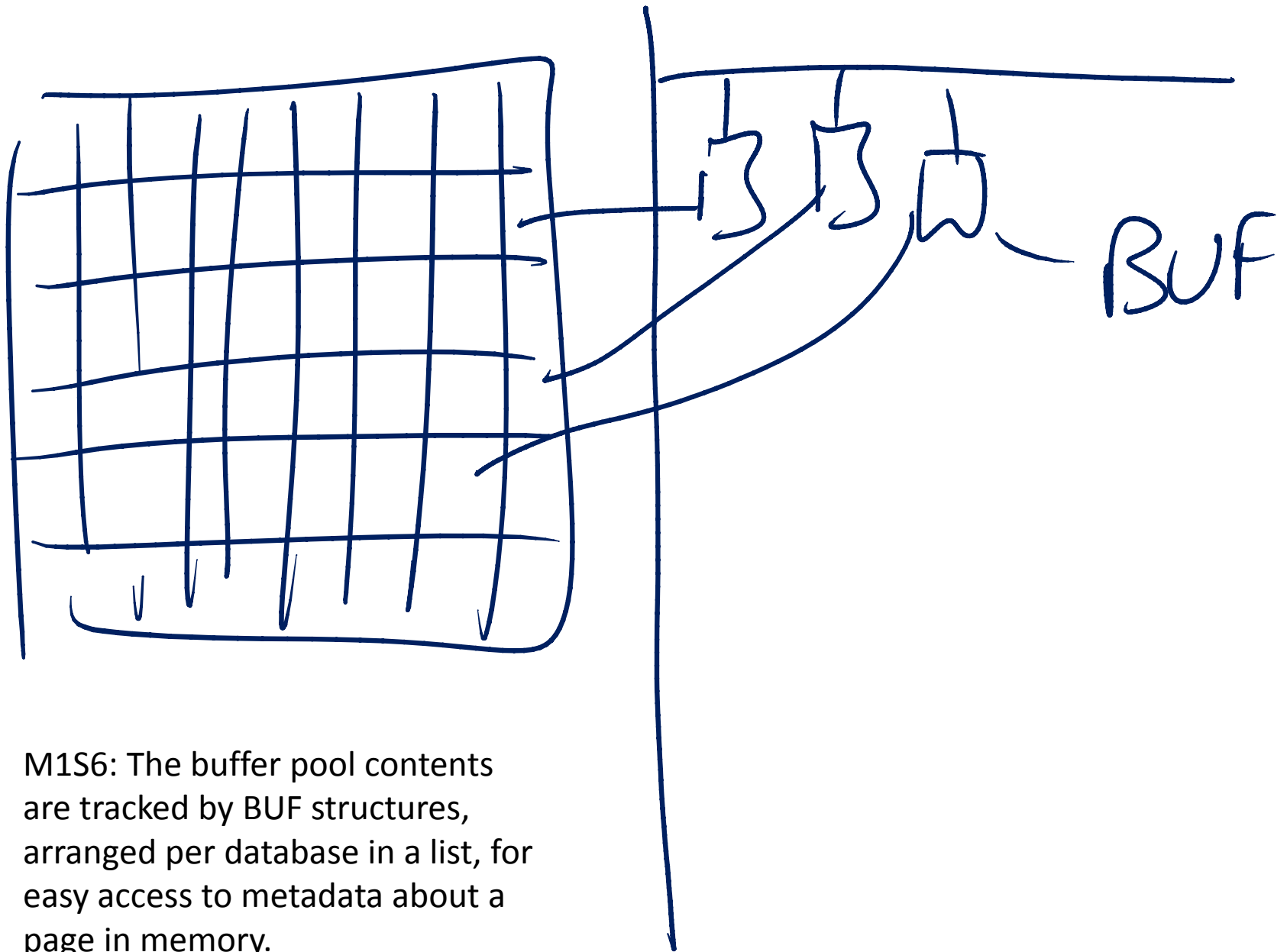
INDEX B

100,000

IN MEM

M1S6 The Query Optimizer does not know what's in memory, and makes plan choices based on index size, not what's in memory. See editorial in

<https://www.sqlskills.com/insidercontent/201609/20160926newsletter.pdf> for example



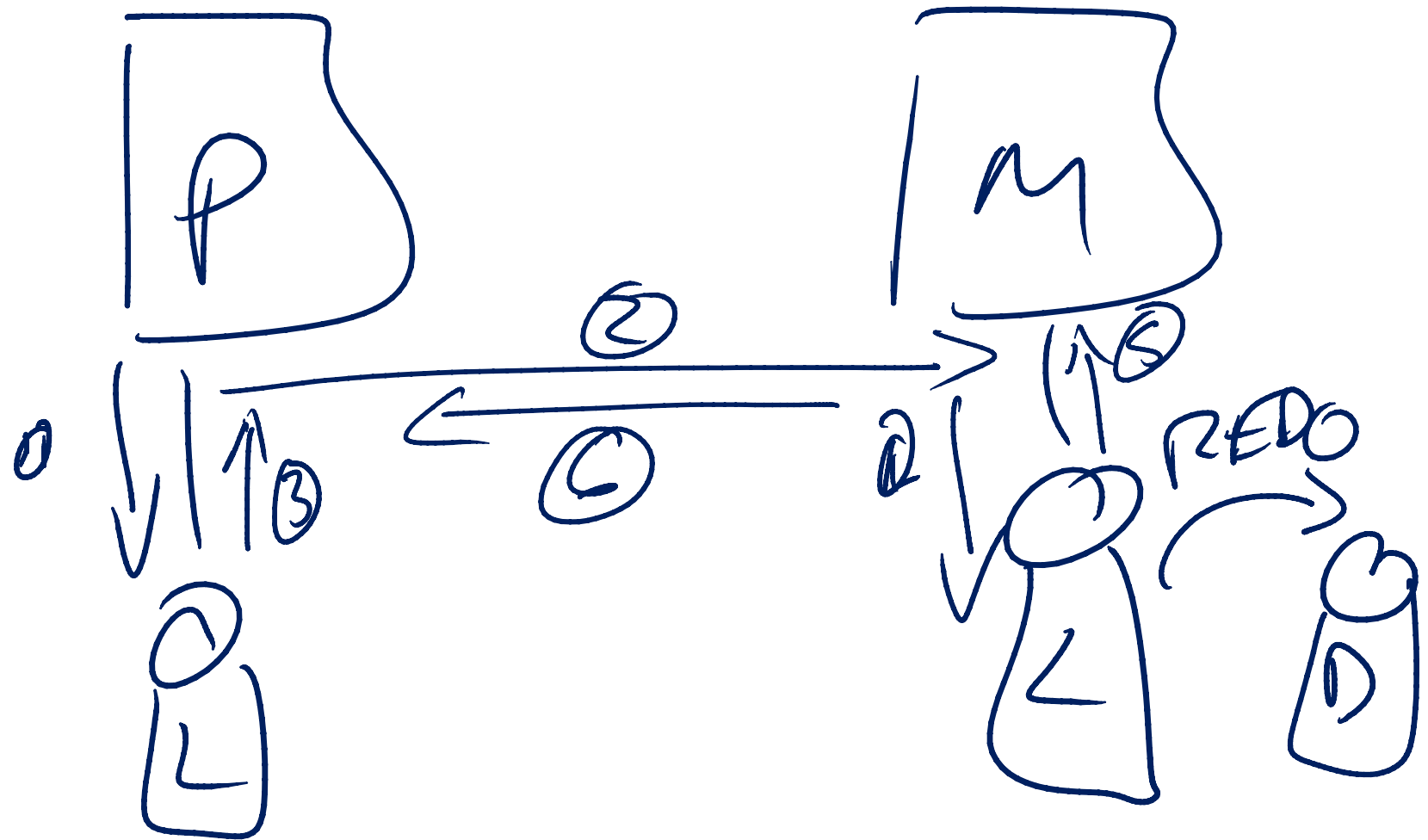
M1S6: The buffer pool contents are tracked by BUF structures, arranged per database in a list, for easy access to metadata about a page in memory.

BUFFER
DISFAVORING

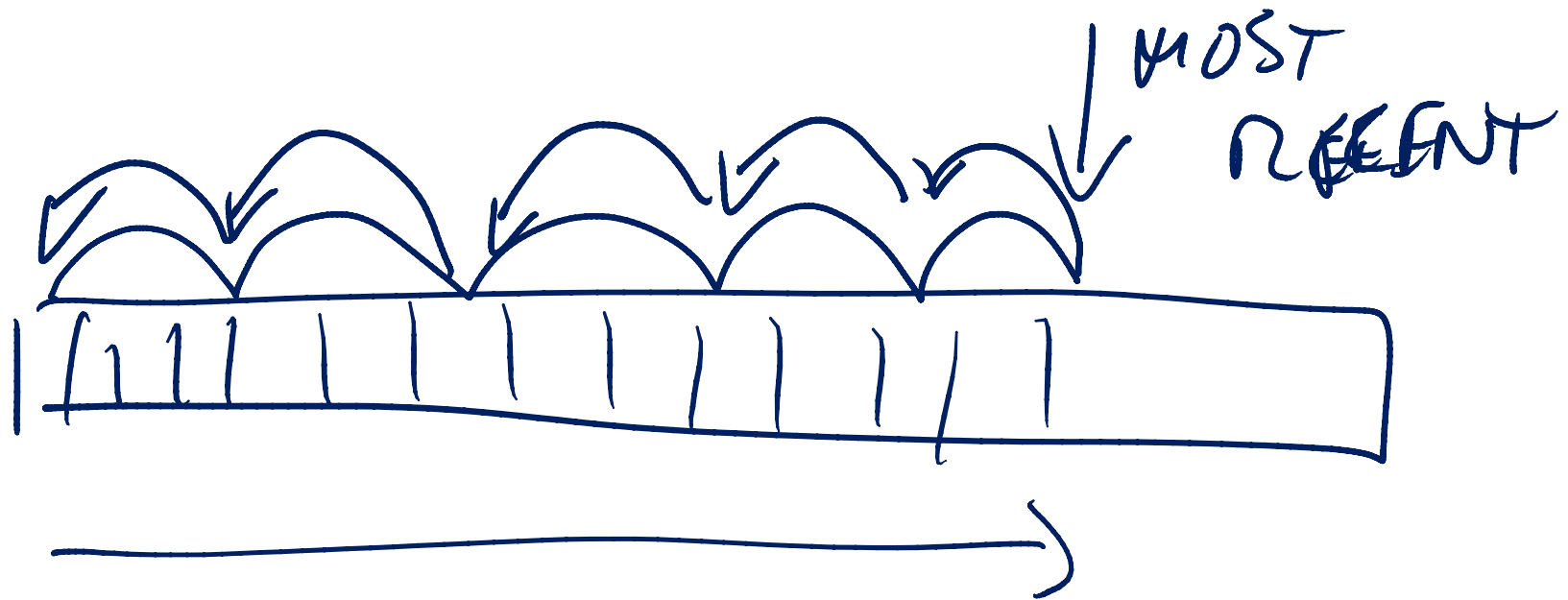
LRU-K, K=2



M1S6: Buffer pool implements a least-recently-used algorithm, to determine which pages are dropped when there's memory pressure. When memory pressure, lazy writer sweeps through buffers to find LRU ones. Disfavoring is discussed [here](#).



M1S12-13 Showing what happens when a transaction commits and the final log block for it is flushed to disk, when sync DBM or AG is involved. The local I/O will complete first but the transaction commit cannot complete until the mirror acknowledges the remote I/O.



M1S12 Discussing how VLF fragmentation (too many VLFs) can cause slow down of rollbacks. This is because of having to search through the VLFs for a particular VLF sequence number when rolling back log records in the reverse order in a transaction. The more VLFs there are, the longer the search for each log records takes.

PFS

2:1:1

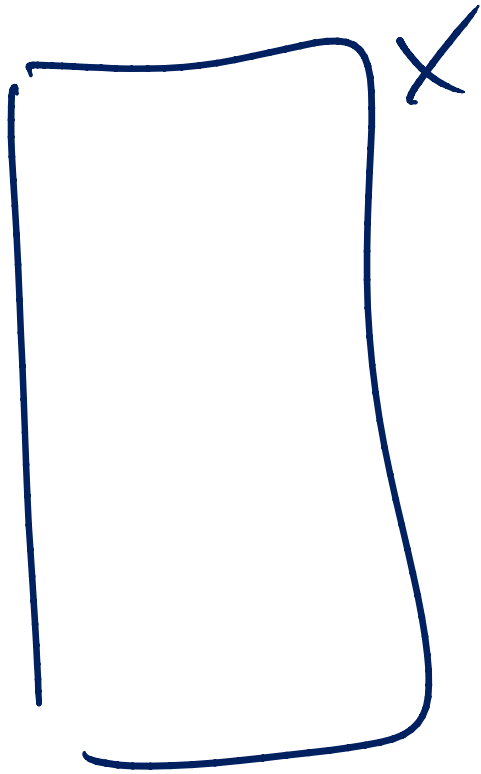
~~SGAM~~

2:1:3

IF 1118

PFS

M1S26 Explaining PFS and SGAM contention in tempdb and how adding multiple files spreads the contention over multiple files, thus reducing it.



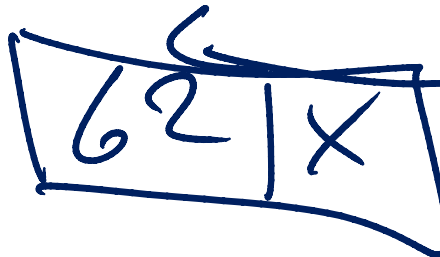
RES.	G.L.	P.Q.
------	------	------

GRANTED
LIST

PENDING
QUEUE



62	X
----	---



M6: see next slide



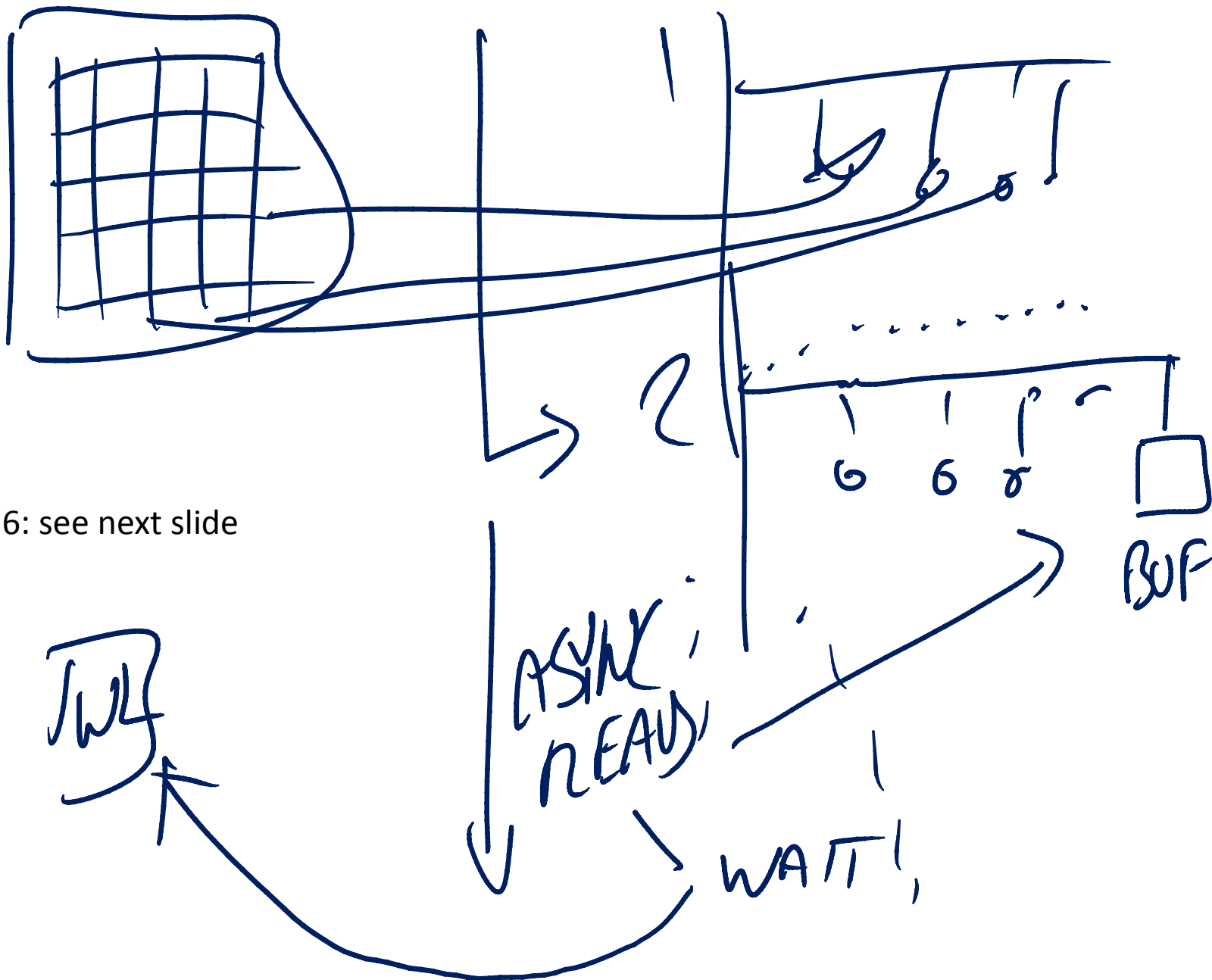
M6 Explaining how the lock pending queue works.

At time:

X: Thread 61 holds the lock in S mode

X+1: Thread 62 tries to acquire the lock in IX and can't. It put itself on the pending queue in the lock value block, which is the waiter list for this resource. It then suspends itself, and moves off the CPU.

X+2: Thread 61 releases the lock. It checks the pending queue for locks that can now be granted ownership of the lock (taking into account any other threads still holding the lock). In this case, thread 62 is granted the lock in IX mode. Thread 61 then moves thread 62 to the runnable queue of the scheduler where thread 62 was running.



M6: see next slide

M6 Explaining how a physical I/O works

Thread looks in the hash lists for the BUF structure for the page it wants.

It doesn't find it.

It creates the BUF structure and latches it EX (this prevents any other threads who also want that page from trying to read it from disk).

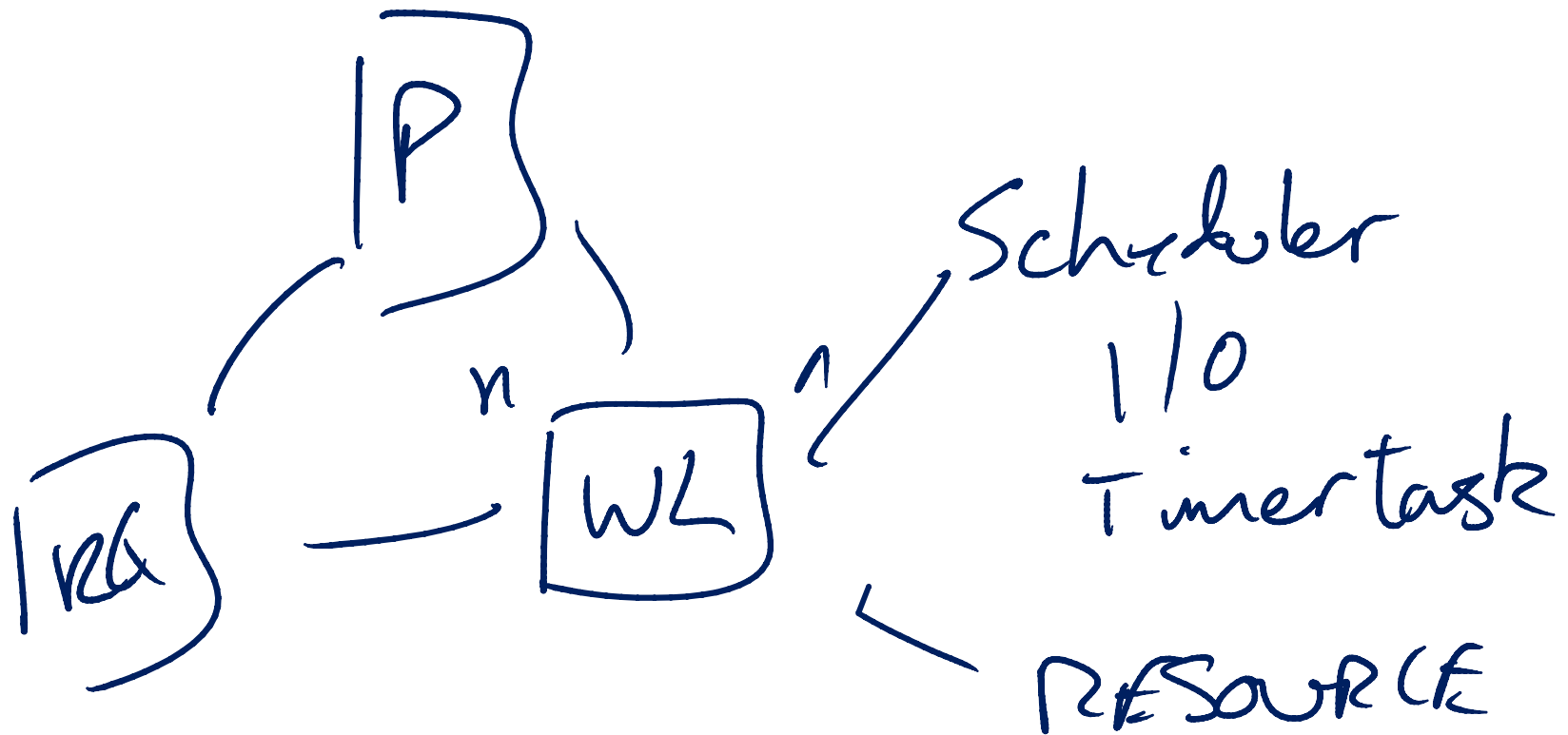
It starts the async physical read.

It puts itself on the I/O waiter list for the scheduler.

It comes off the processor and is now suspended.

Some other thread on that scheduler, when it's about to get suspended, will check to see whether any of the I/Os for threads on that scheduler have completed. If so, whatever thread is waiting for the completed I/O is moved to the runnable queue.

Remember: threads schedule themselves cooperatively.

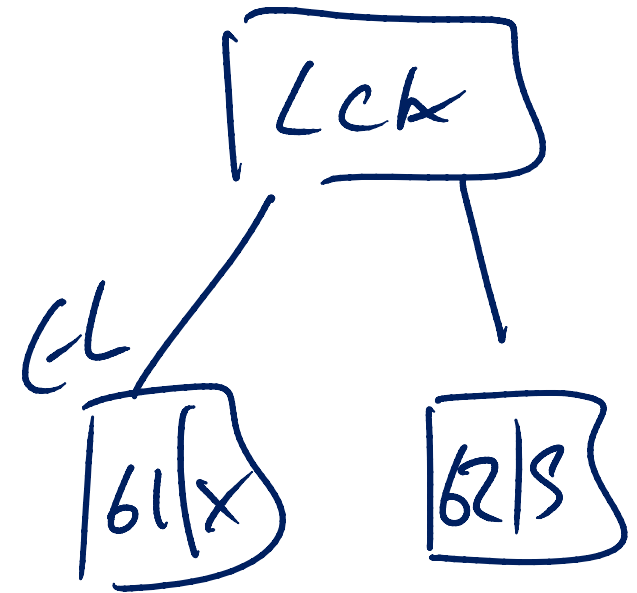
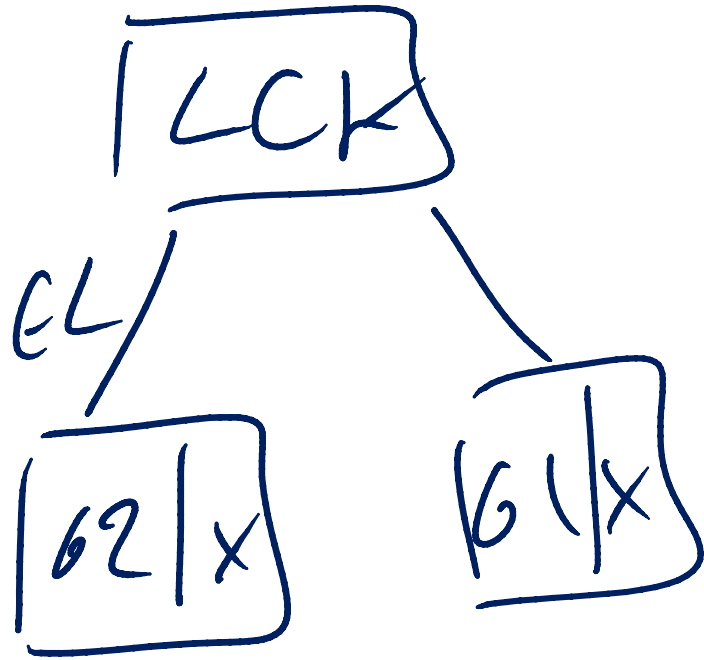


M6S9 The 'waiter list' is often described as being part of the scheduler. It's actually composed of several things:

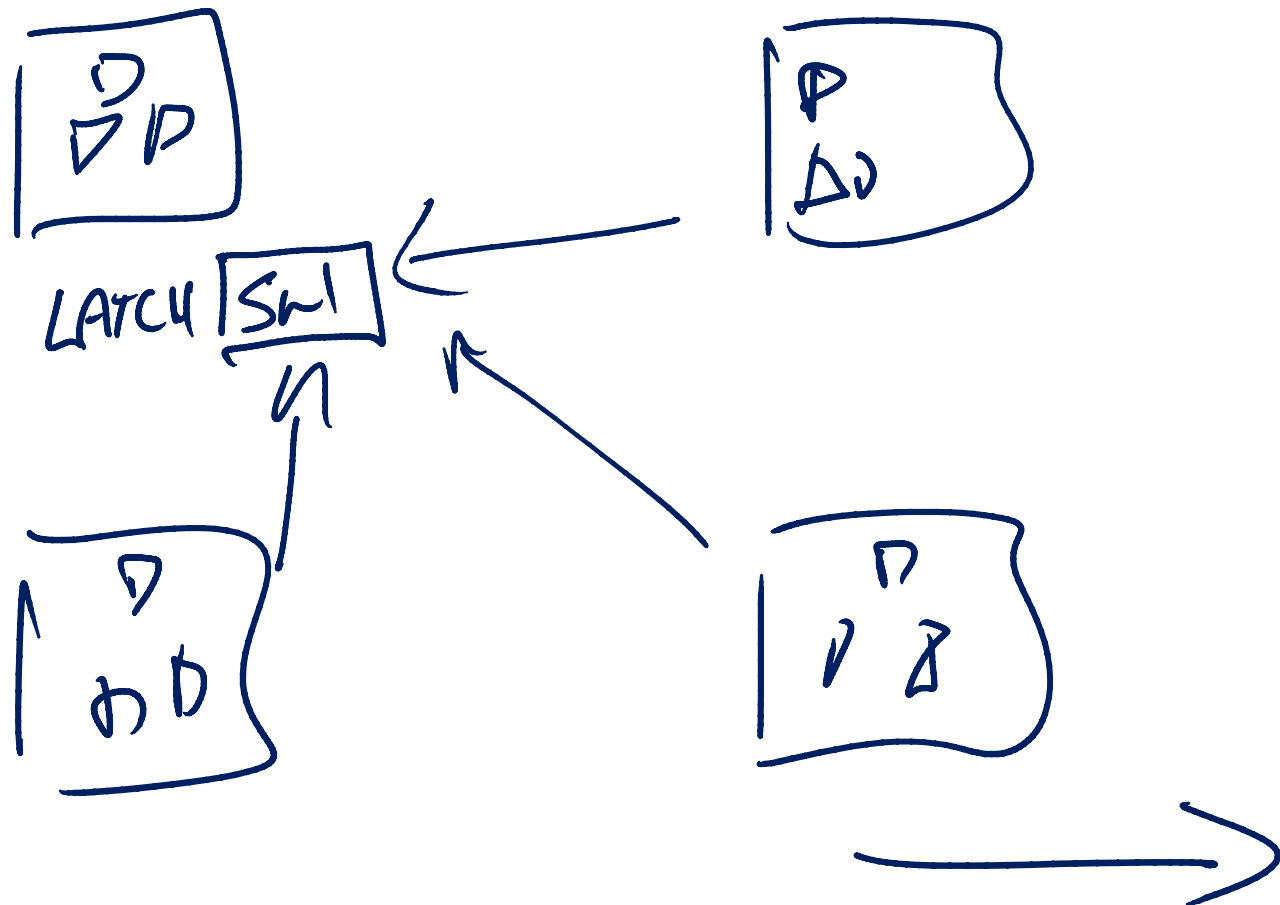
- 1) The I/O waiter list for the scheduler
- 2) The timer task waiter list for the scheduler
- 3) Threads from this scheduler waiting on other resources on the various resource pending queues (e.g. a lock)

RESOURCE

- └ Locks
- └ latches
- └ mon grants
- └ threads

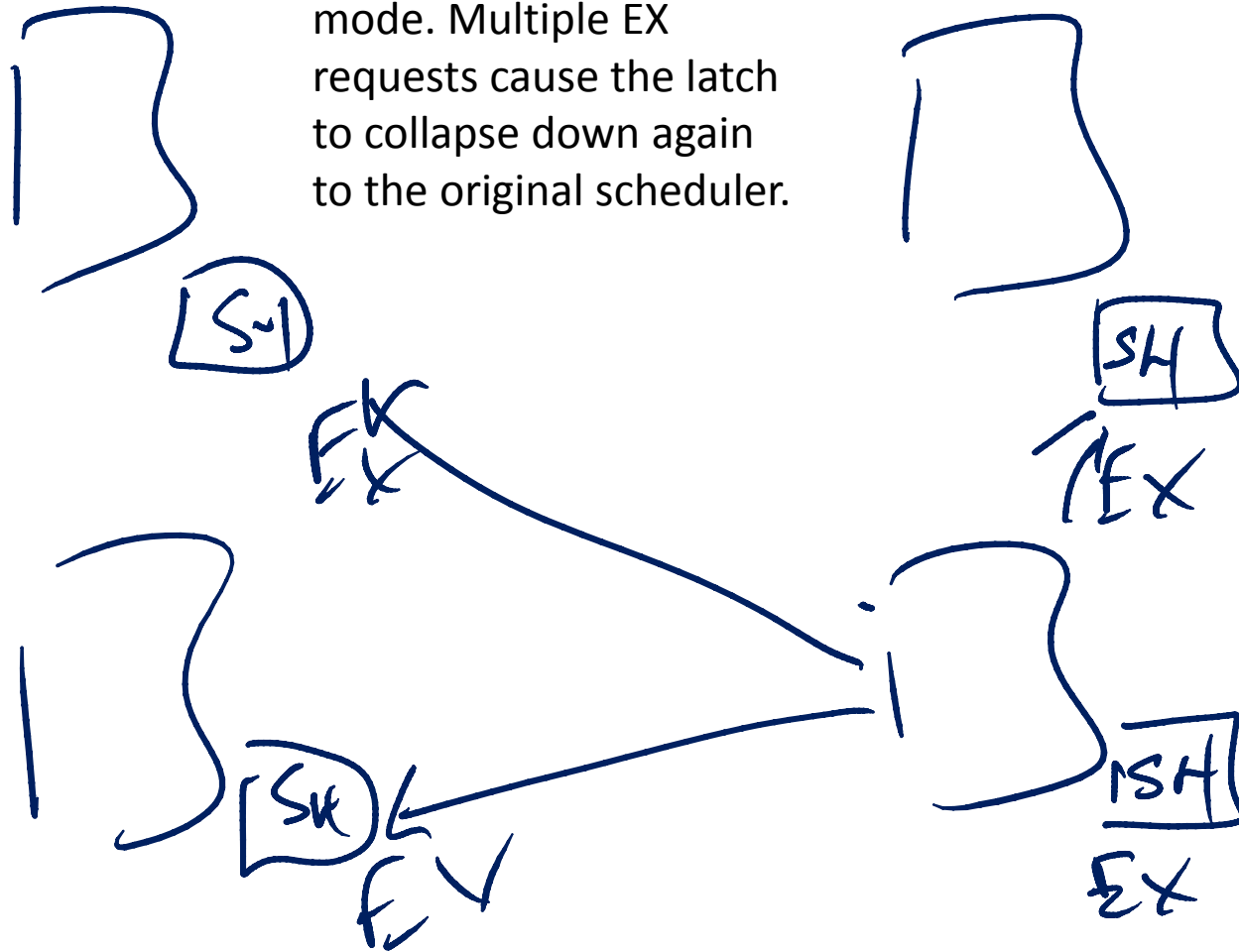


Deadlock scenario: Spid 62 X locks the first page. Spid 61 X locks the second page. (GL = Granted List of locks for that resource.) Spid 61 then tries to X lock the first page. Spid 62 then tries to S lock the second page. That's a deadlock. The deadlock monitor notices and picks a victim, based on who's done the least amount of work. Let's say it's 62. Thread 62 (waiting for the S lock) is signaled (moves to the runnable queue) and the deadlock monitor has set a bit in Thread 62's memory that indicates it's been chosen as a deadlock victim. Thread 62 then initiates rolling back the transaction that it's part of.

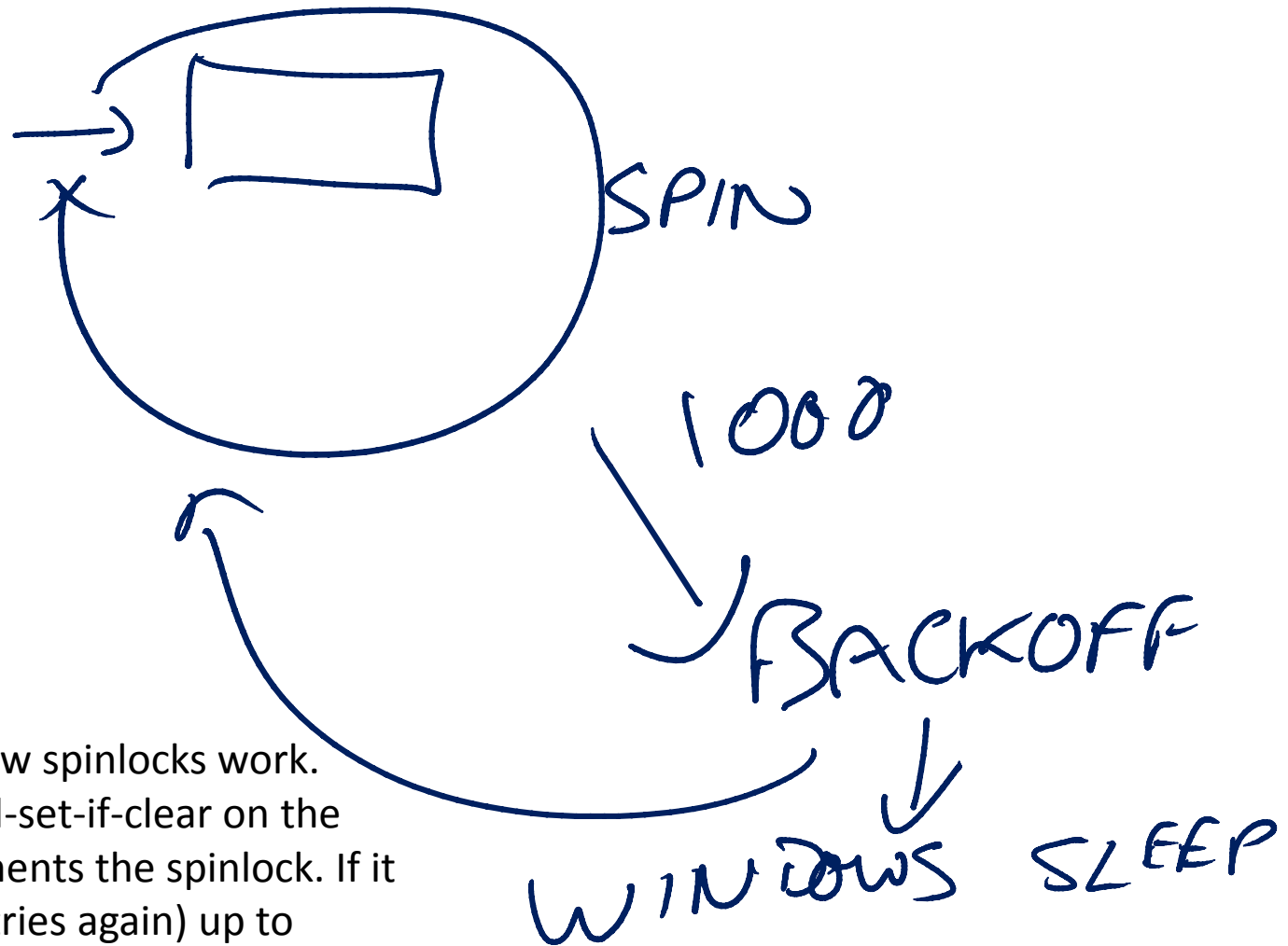


M6S35 Explaining about superlatches. This is when there's only one latch and it's owned by one scheduler (big boxes are scheduler)...

M6S35 Explaining about superlatches. This is when there's one latch per scheduler, in SH mode. Multiple EX requests cause the latch to collapse down again to the original scheduler.



INTERLOCKED COLLISIONS



M6S40 Explaining how spinlocks work.
Code does a test-and-set-if-clear on the
memory that implements the spinlock. If it
can't get it, it spins (tries again) up to
(generally) 1000 times and then backs off.

M6S61 Keeping
MAXDOP set to size
of a NUMA node
helps to avoid
foreign memory
accesses, which are
expensive. Check in
Buffer Node
perfmon object. Big
boxes are NUMA
nodes. Small boxes
are buffer pool
partitions.

