MARK BROADBENT
Principal Consultant
sqlcloud
SQLCLOUD.CO.UK

Contact…

mark.broadbent@sqlcambs.org.uk

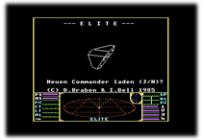

@retracement

tenbulls.co.uk

Guilty pleasures…

SERVERLESS

Likes…



Badges…

Microsoft CERTIFIED
Master: SQL Server

MVP Microsoft® Most Valuable Professional

Microsoft Community Contributor AWARD

Microsoft CERTIFIED Solutions Expert Data Platform

Microsoft CERTIFIED Solutions Expert Data Management and Analytics

PASS

Community…

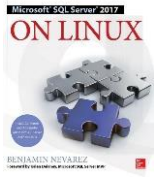
PASS HYBRID VIRTUAL CHAPTER

SQLEA

SQL SATURDAY CAMBRIDGE

SHAREPOINT SATURDAY CAMBRIDGE

Tech editor…



Agenda

SQL Server Mechanics and Configuration

Concurrency models and supporting technologies

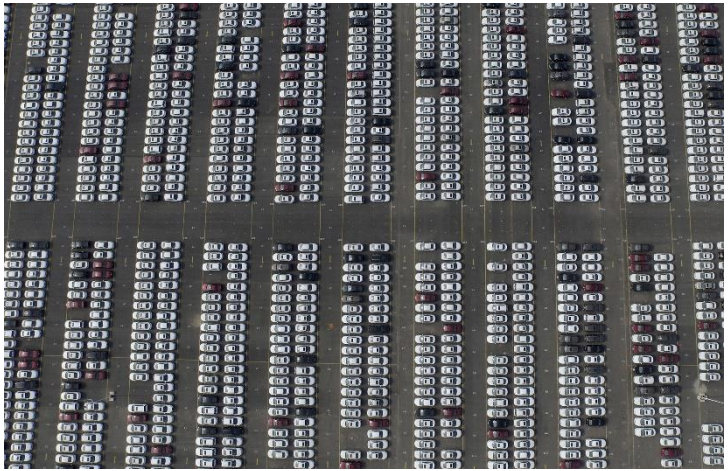
Concurrency and Correctness #fails and solutions

Improvements

Speed, Concurrency, Correctness?

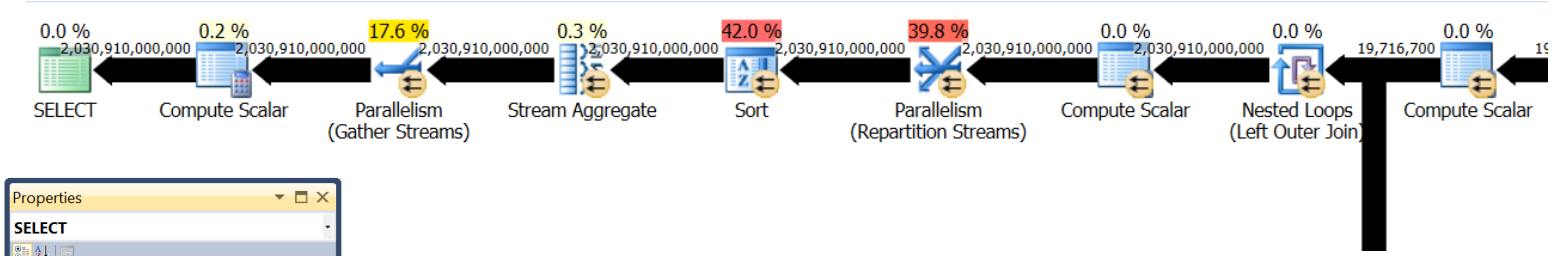
Speed

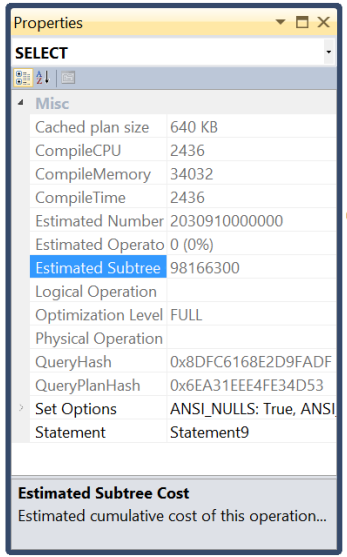
Concurrency

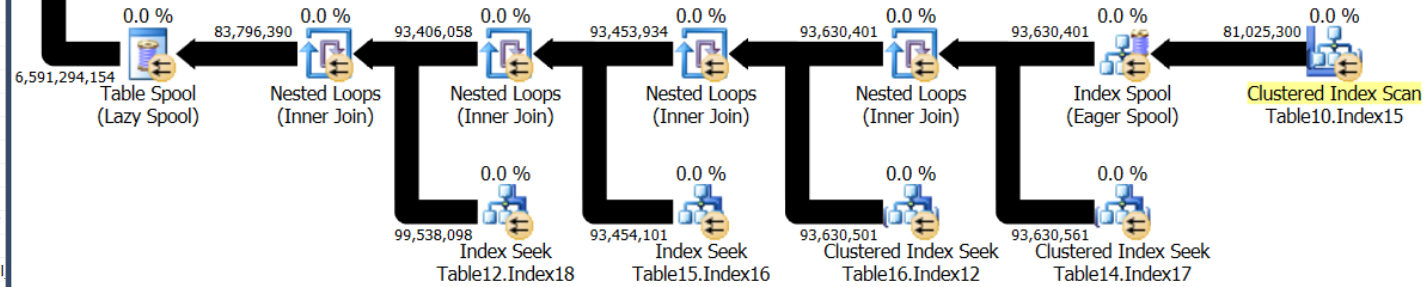


Correctness

The “Benedict” Query







Laws

Heisenberg uncertainty principle

"the more precisely the position of some particle is determined, the less precisely its momentum can be known, and vice versa."

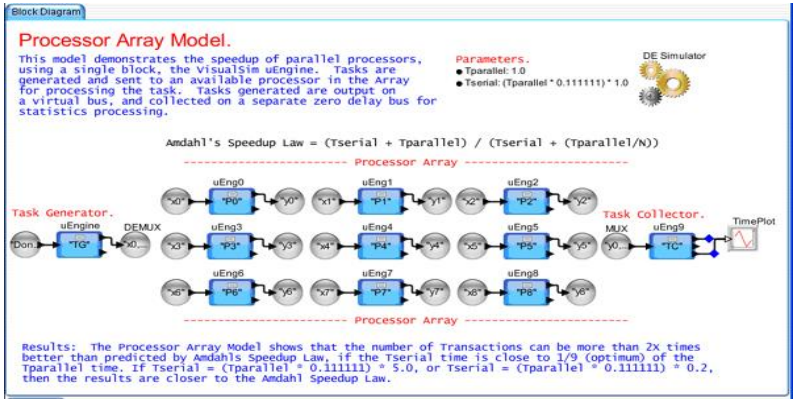
...or the more closely we look at a system in motion, the less we see.

Pareto principle (aka the 80/20 rule)

"80% of Italy's land was owned by 20% of the population."

...or 80% of the performance problems are 20% of the queries.

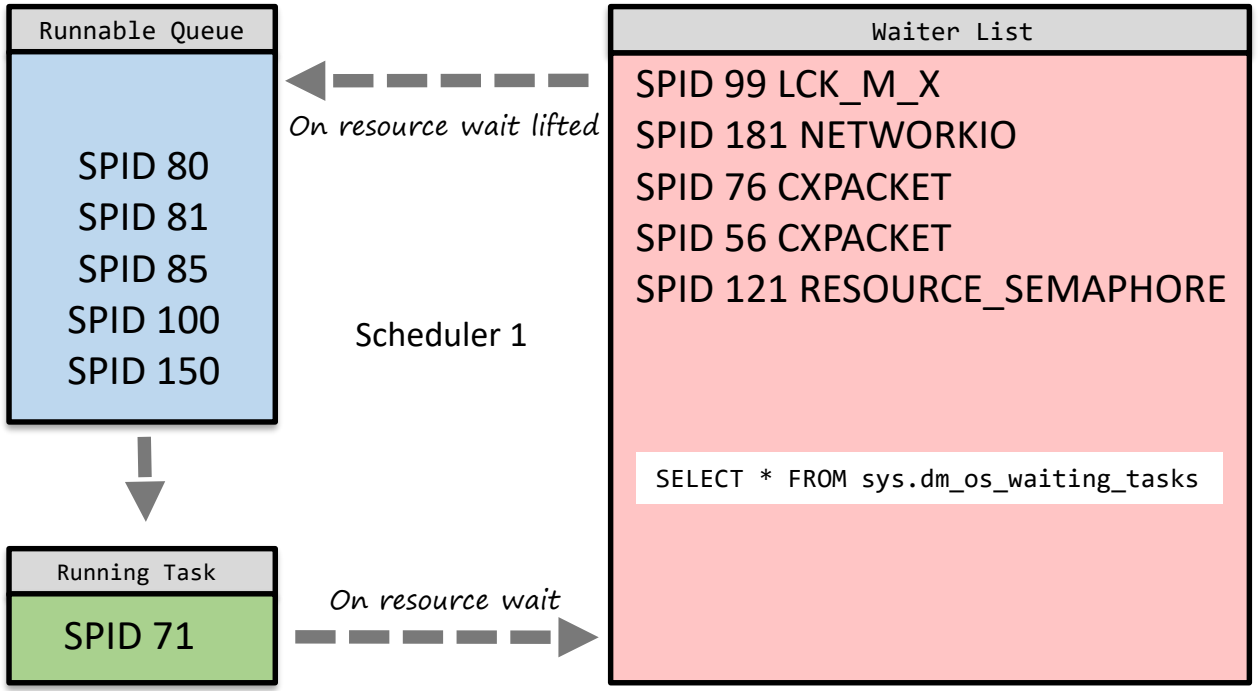
Amdahl's law



"The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program."

SQLOS Scheduling

- 1 scheduler per logical CPU
- ***wait_time_ms*** is sum of time on waiter list + time on runnable queue
- ***signal_wait_time_ms*** is time spent on runnable queue
- High signal waits → high CPU pressure
- Large time on waiter list → high resource bottlenecks




```
SELECT * FROM sys.dm_os_wait_stats
```

Wait_type	waiting_tasks_count	wait_time_ms	Max_wait_time_ms	signal_wait_time_ms
RESOURCE_SEMAPHORE	6	50	20	8
CXPACKET	120	50	200	18
LCK_M_S	19199	5000	780	80
LCK_M_X	3	21	7	3

SQLOS Scheduling review

- Parallel queries will utilize all schedulers for their runnable quantum (and most by default will go parallel!)
  - Set ***max degree of parallelism*** to avoid single heavy parallel query from overwhelming cores (consider NUMA configuration too)
  - Set ***cost threshold for parallelism*** to sensible level.
- Monitor waits and queues performance impact over time (rather than a snapshot)
- Consider implementing Resource Governor.

SQLOS Scheduling review (common waits)

- LCK_M_* (wait for lock grant for read/write/update) → locking/ concurrency bottleneck.
- RESOURCE_SEMAPHORE (wait for query memory grant) → memory bottleneck
- CXPACKET waits always exist. Excessive CXPACKET waits → imbalance of parallelism
- SOS_SCHEDULER_YIELD (exhausted runnable queue quantum) → CPU pressure
- PAGEIOLATCH_* (latch wait for IO read/write to memory) → disk IO bottleneck
- PAGELATCH_* (latch wait for read/write to a page in memory) → system overloaded
- LATCH_* (latch wait for non-buffer resource) → resource overloaded.

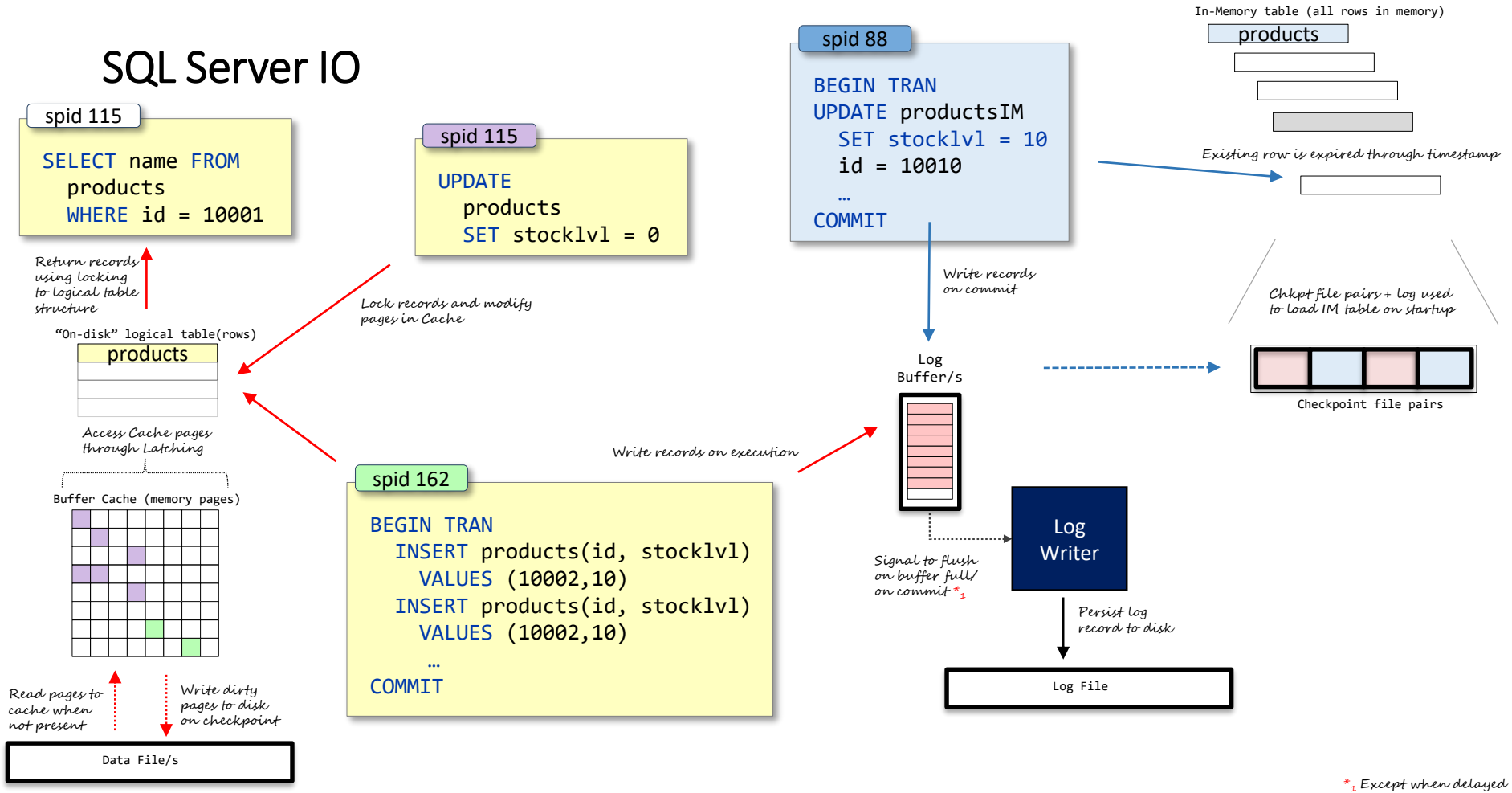
https://www.sqlskills.com/help/waits/

https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2005/administrator/cc966413(v=technet.10)

https://www.sqlskills.com/blogs/paul/capturing-wait-statistics-period-time/

Wait stats are expected and relative to your workload

SQL Server IO

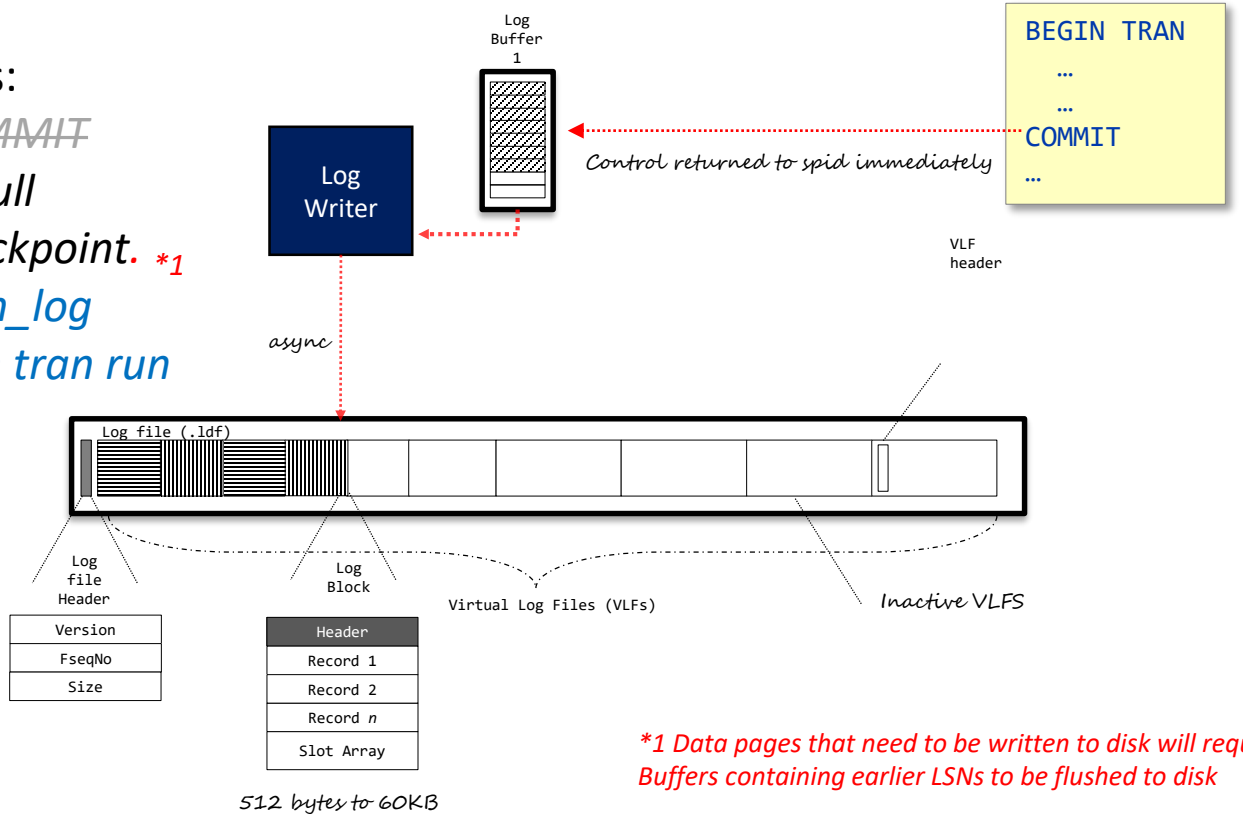


*1 Except when delayed durability is in use

Delayed Durable Transactions

Log flushes:

- ~~On COMMIT~~
- *When full*
- *On Checkpoint.* *1
- *sp_flush_log*
- *Durable tran run*



*1 Data pages that need to be written to disk will require Buffers containing earlier LSNs to be flushed to disk

SQL IO review

- Large reads or writes will reduce your Buffer cache page life expectancy
  - Use page compression on a table and index pages
  - Set ***max server memory*** so that OS has a reservation –Approx 2GB, set to limit buffer cache
  - Implement indexing strategy and tune queries.
- Transaction log will always be a key bottleneck, improve by
  - Reduce IO overhead (reduce data change)/ Consider In-Memory OLTP
  - Consider Delayed Durability
  - Improve hardware
  - Avoid logfile log growths.

Isolation Levels

Isolation Level	"Bad Dependencies"	TX Dependencies	Concurrency
READ UNCOMMITTED	Dirty Read, Non-Repeatable Read, Phantoms, Lost Updates	WRITE → WRITE	GOOD: prone to allocation order scan failures
READ COMMITTED	Non-Repeatable Read, Phantoms , Lost Updates	WRITE → WRITE READ → WRITE WRITE → READ	OK: wait on both only writers held to EOT
READ COMMITTED (with Snapshot)	Non-Repeatable Read, Phantoms , Lost Updates	WRITE → WRITE	GOOD: no wait on readers only writers held to EOT
REPEATABLE READ	Phantoms	WRITE → WRITE READ → WRITE WRITE → READ	LOW: read and write locks held to EOT
SERIALIZABLE	None	WRITE → WRITE WRITE → READ READ → WRITE	LOWEST: read and write locks held to EOT
SNAPSHOT	*None (though Causal consistency concerns, lost update prevention and other behaviours)*	WRITE → WRITE	GOOD: only wait on writes held to EOT

Isolation levels with In-Memory OLTP are either restricted and/ or non-blocking

Isolation review

- Isolation Levels attempt to ~~solve~~ reduce interleaving dependency problems.
- Set at session level, transaction level and statement
- NOLOCK is prone to failures for long running queries (and dirty reads) instead use:
  - Use SNAPSHOT ISOLATION for consistency
  - Use READ COMMITTED SNAPSHOT for concurrency (and as new default)
- RCSI can be set as a default, but has all consistency challenges as READ COMMITTED
- Both SI and RCSI still block on writes
- In-Memory OLTP is functionally better than either.

Demo

Consistency

Transactions

- Transactions either explicitly defined or automatically implied (auto-commit)
  - Auto-commit transactions = statements outside a transaction block.
- Long running transactions effect the concurrency of the system
- Isolation level effects a transaction's bad behaviours and dependencies
- Error handling (+ rollback) reduces bad behaviours
- Multi-statement operations are prone to lost-updates –under the ~~right~~ wrong circumstances.

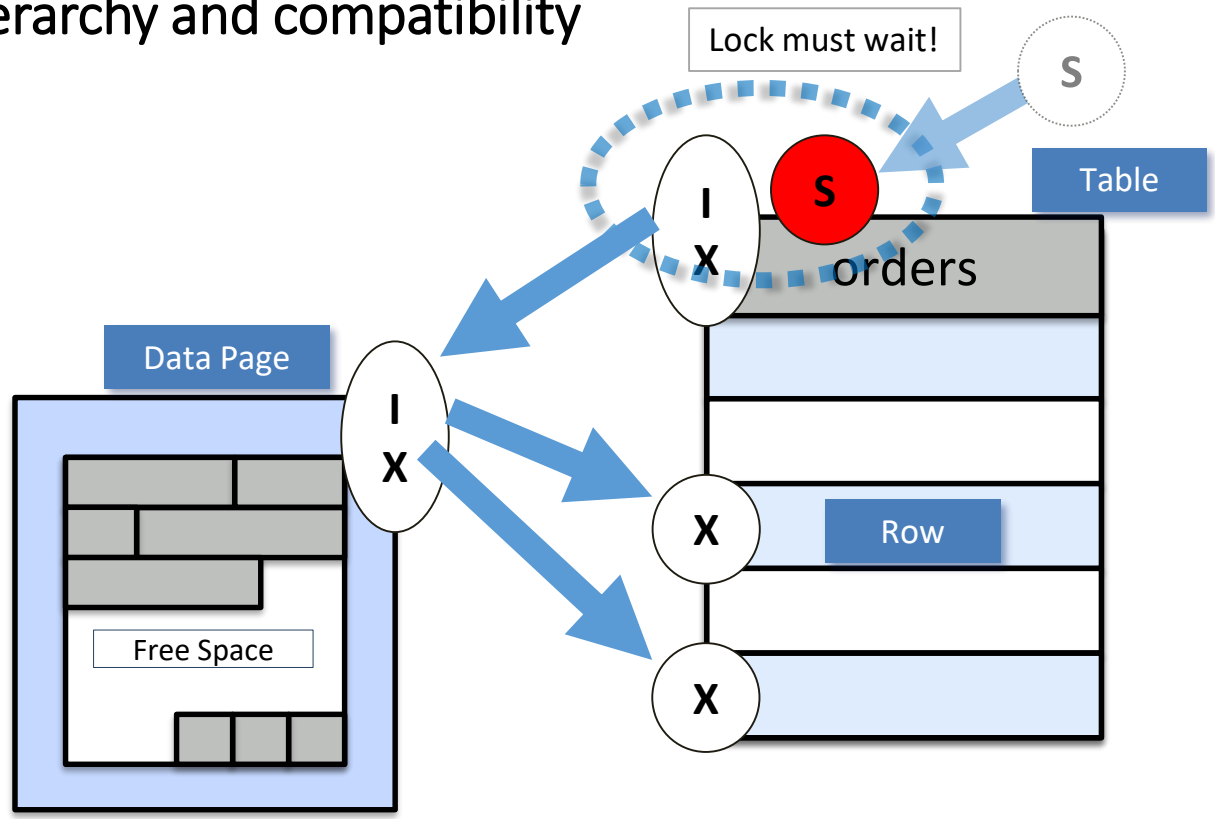
SQLQuery1.sql

```
…
BEGIN TRAN
  INSERT INTO t1 VALUES (1)
  SELECT @id=id FROM t2
  DELETE FROM t2 WHERE id = @id
COMMIT
```

Transaction Review

- Keep transactions short (especially in OLTP).
- Readers and writers are the root of consistency problems avoid mixing loads and understand the locking model driving them.
- SNAPSHOT isolation is better suited for consistency, but RCSI is better suited to concurrency.
- Ensure you ALWAYS handle errors
  - Replay deadlocks where applicable
  - Detect and replay on lost-update detection errors where applicable.

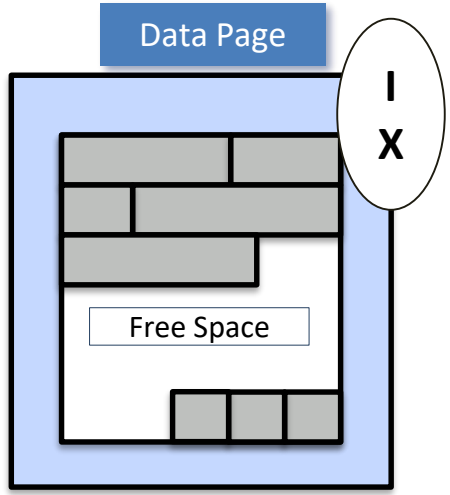
Lock hierarchy and compatibility

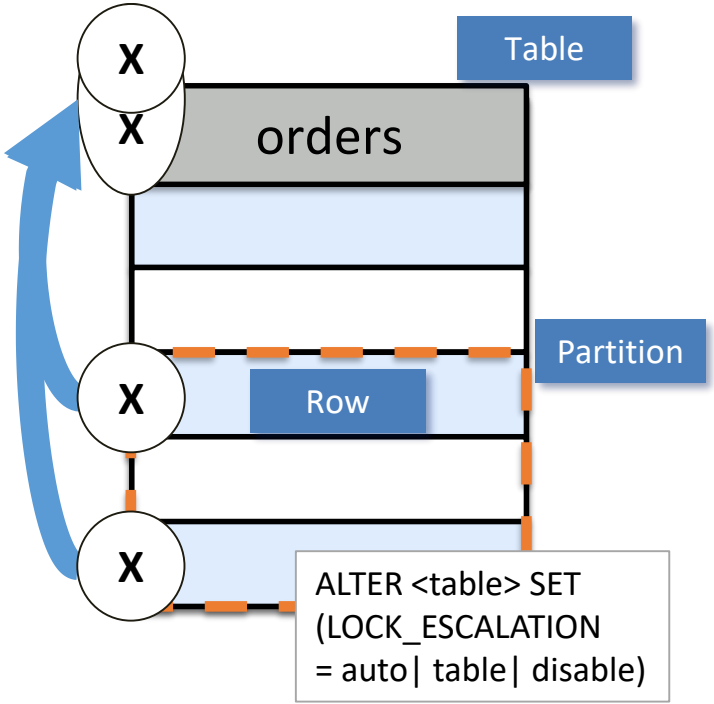


Locks either compatible or in-compatible. Locks are only compared against others on the resource level only

Lock Escalation

Don’t escalate by **TF1211**
or take artificial IS on table
Ignore # locks by **TF1224**





Locking review

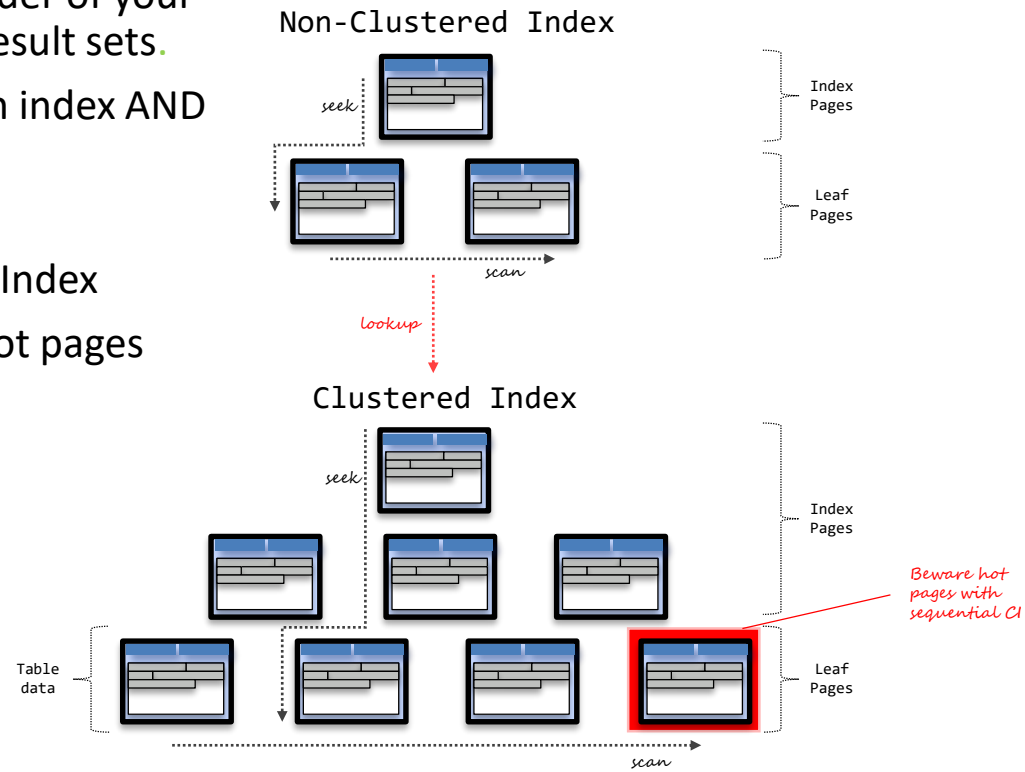
- Are ONLY memory structures (lock blocks) and can consume a lot of memory
  - Try to avoid the need for escalation!.
- Explicitly use table locks ONLY when it makes sense (bulk loading/ reporting)
- Lock duration depends upon ISOLATION LEVEL
  - Write locks last for the entire transaction!
  - Read locks (by default) are taken and released for each row read.

Demo

On-disk concurrency

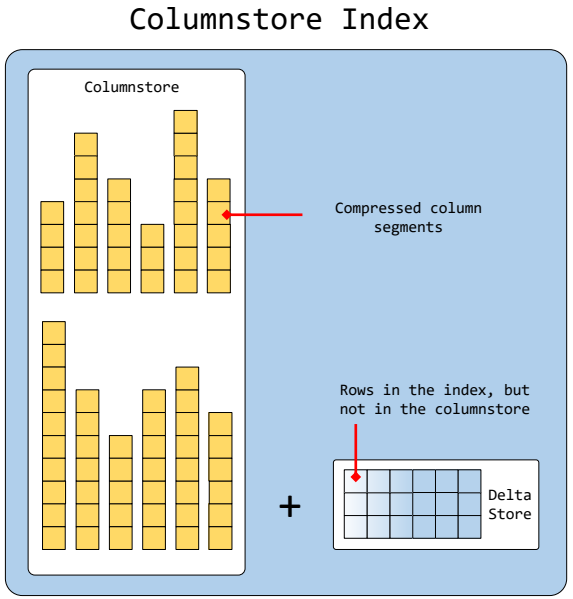
Indexing

- Clustered Index defines the logical order of your data – good for range scans, sorted result sets.
- Key columns determine space used in index AND leaf pages
- Included columns in NC leaf only
- Cluster key present in Non-Clustered Index
- Sequential Cluster key can result in hot pages



Columnstore indexes

- High volume column based storage
  - Ideal for data warehouse fact tables
  - Also use for reporting /archival
- Deltastore used for incremental changes
  - Avoids fragmentation
  - Bypassed during bulk loads
- Results are columnstore + deltastore
- Implement as Clustered Columnstore
- Or create Non-Clustered Columnstore on rowstore or In-Memory OLTP!
- Or create rowstore nonclustered on columnstore!!!
- Row mode or Batch mode execution

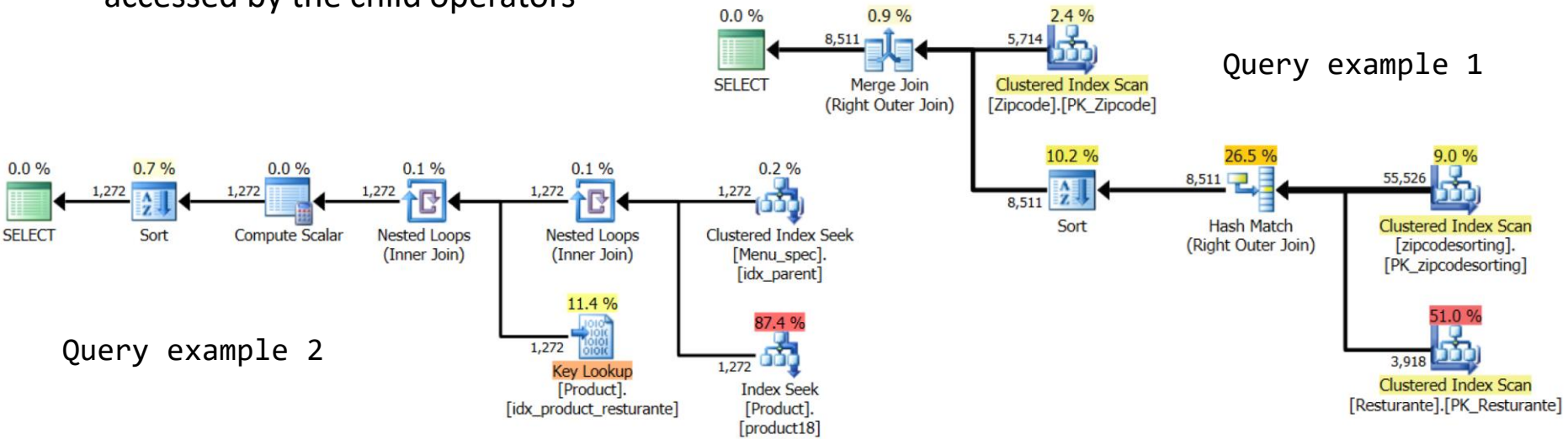


Indexing review

- Indexes add overhead – HEAPS really only good for loading.
- Keep key columns few and small
  - But index on multiple columns to achieve high SARGability
- An index creates statistics object. If you have temp statistics but no indexes could indicate missing indexes
- Leading wildcards on predicates will not use index, use ‘M%’ rather than ‘%ark’
- Calculated predicates will scan (such as Sales * Qty < 100)
- All tools potentially duplicate or create excessive indexes rather than consolidate.

Is my query a problem?

- Look at number of estimated rows returned, query cost, memory grant
- Identify problem operators: sorts, joins, scans, lookups – why are they used?
- Compare rows returned from query against rows accessed by the child operators

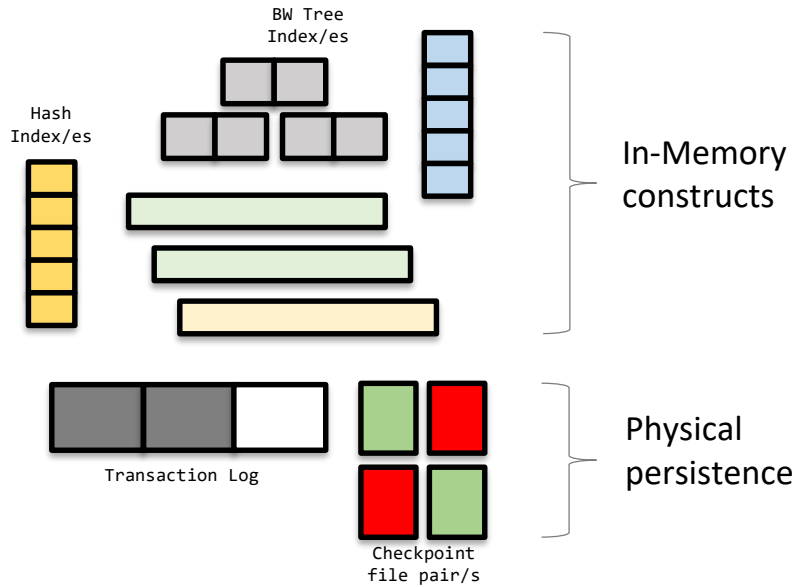


Query tuning review

- Limit columns used in query (and if all are necessary) include in index
  - Index key columns may be used for the join or filter predicates for efficient seeks
  - Included columns will avoid lookups for remaining columns (at the leaf level).
  - ALL cluster key columns are present in nonclustered index so avoid duplicating their use in your non-clustered index (unless required!)
- Consider your predicate logic (e.g. OR will essentially require a separate operation, but cannot return duplicates so will often require a sort and temp table to eliminate)
- Beware of joins or filters on different predicate types –implicit conversion

In-Memory OLTP to the rescue…

- In-Memory data structures (optimised for memory).
- Persistence through Transaction Log and checkpoint file pair/s
- No TempDB overhead – all versioning In-Memory
- Logging optimizations and improvements
- Lockless and latchless operation
- Query through interop or Natively Compiled Stored Procedures
- No fragmentation concerns.



Demo

Improvements and In-Memory Concurrency

In summary

- Keep transactions short
- Avoid complex logic
- Use RCSI or SI isolation
- Implement new technologies to improve concurrency
  - Use In-Memory OLTP if you can
  - Use Columstore where required
- Review (and optimise) all query plans
- Test code and results in parallel.

Thank you for listening!

Email: mark.broadbent@sqlcambs.org.uk

Twitter: @retracement

Blog: http://tenbulls.co.uk

Slideshare: http://www.slideshare.net/retracement

Demo: https://github.com/retracement/2_fast_2_furious