designing for

speed

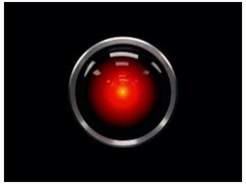
correctness

concurrency

MARK BROADBENT

Agenda

Query Tuning and Indexing



Isolation and Locking

Transaction Processing and overheads

Improvements

Speed, Concurrency, Correctness?

Speed

Correctness

Concurrency

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."

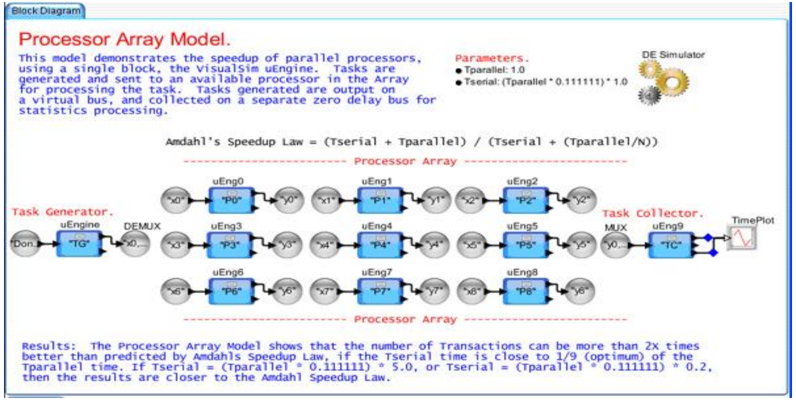
…or the more closely we look at a system in motion, the less we see.

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.

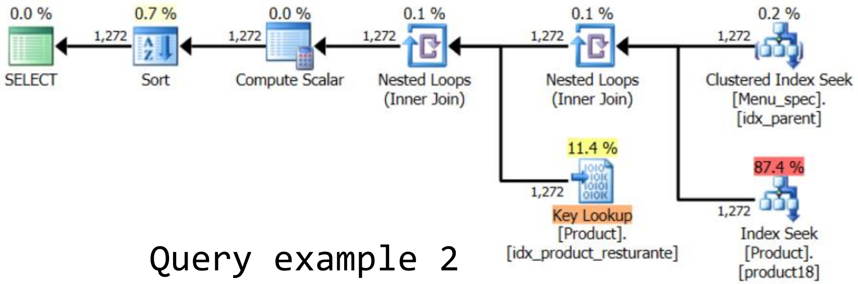
Amdahl's law



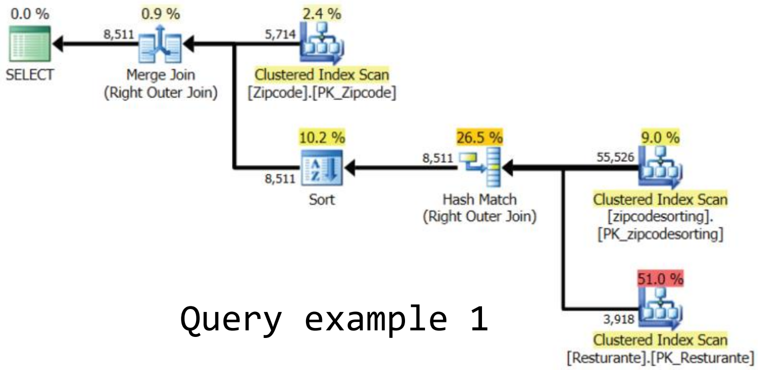
"The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program."

Is my query a problem?

- Review query cost and memory grant
- Compare #rows returned against #rows accessed by the child operators
- Problem operators: scans, lookups, sorts, joins – why are they used?
- Predicate logic (e.g. using OR requires a separate op, 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.



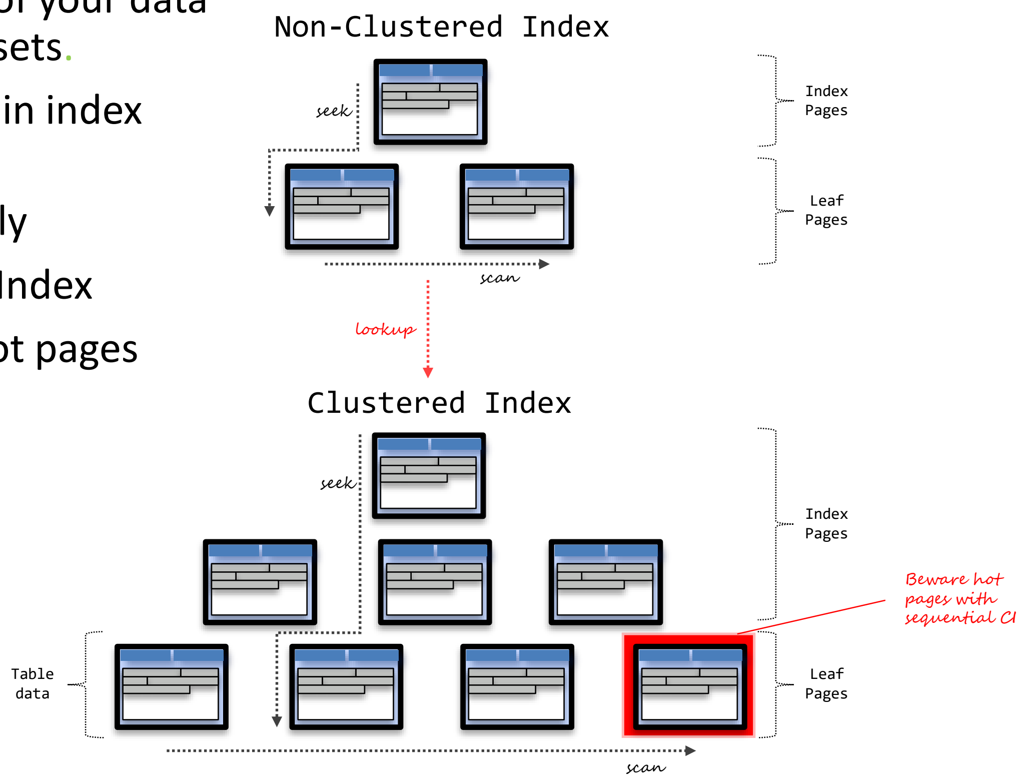
Query example 2



Query example 1

Indexes

- Clustered Index defines 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 pages only
- Cluster key present in Non-Clustered Index
- Sequential Cluster key can result in hot pages



Query Tuning and Indexing review

- Limit query columns (and if all necessary) include in index (for covering)
  - Index key columns may be used for the join or filter predicates for efficient seeks
  - Included columns avoid lookups for remaining columns (at the leaf level).
- Indexes add overhead – HEAPS ONLY really only good for loading.
- Keep index key columns few and small
  - But index on multiple columns to achieve high SARGability
- Index creates statistics object. Temp stats 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).

Demo

Query “Speed”

Isolation and Locking

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 provide balance between consistency and correctness.
- Set at session level, transaction level and statement
- Only RC or RCSI can be set as a default
- Both on-disk SI and RCSI still block on writes – but remove need for NOLOCK
- Use In-Memory OLTP is functionally better than either.

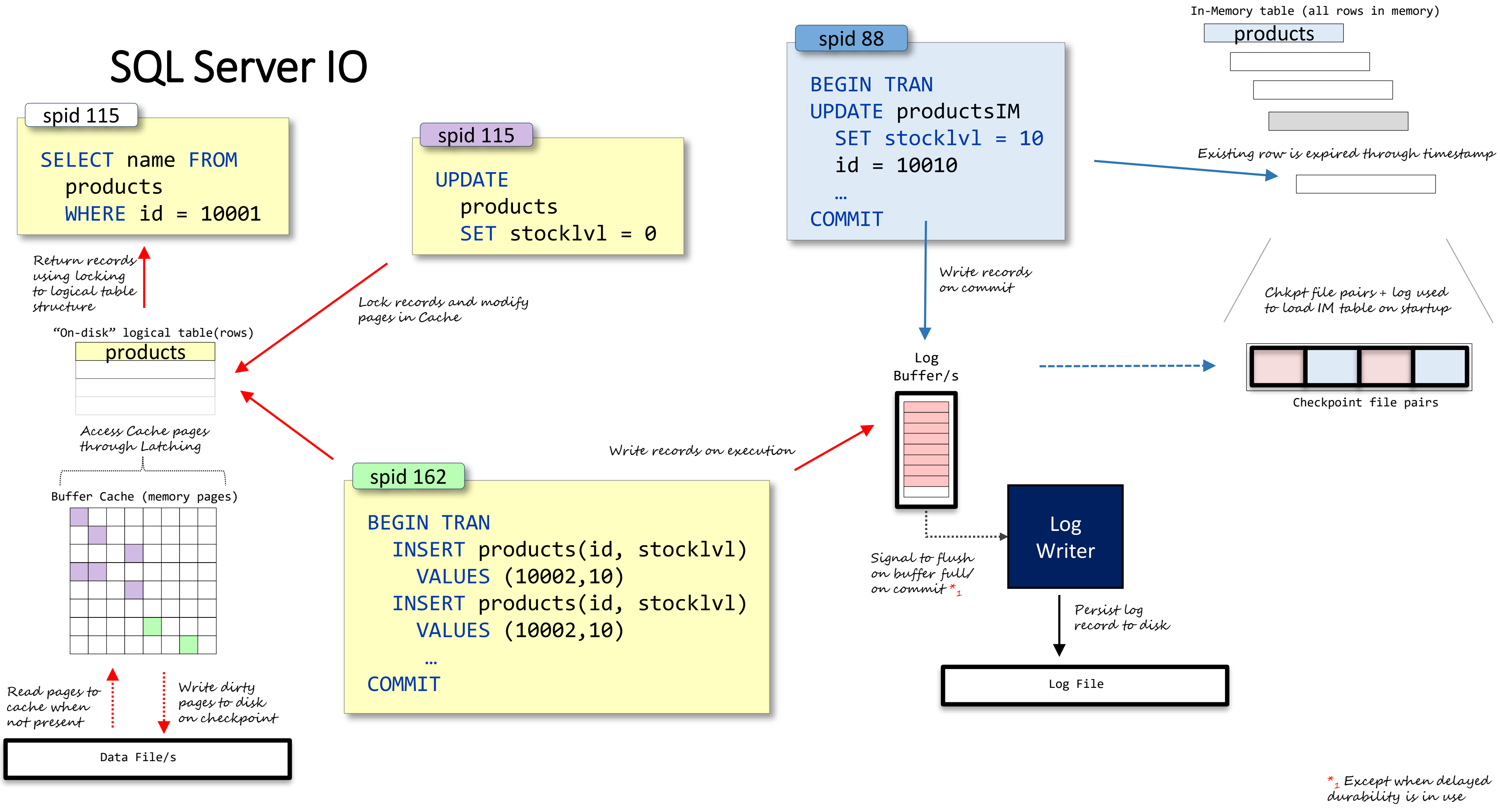
Locking review

- 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)
- 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)
- 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

Correctness and on-disk concurrency

SQL Server IO



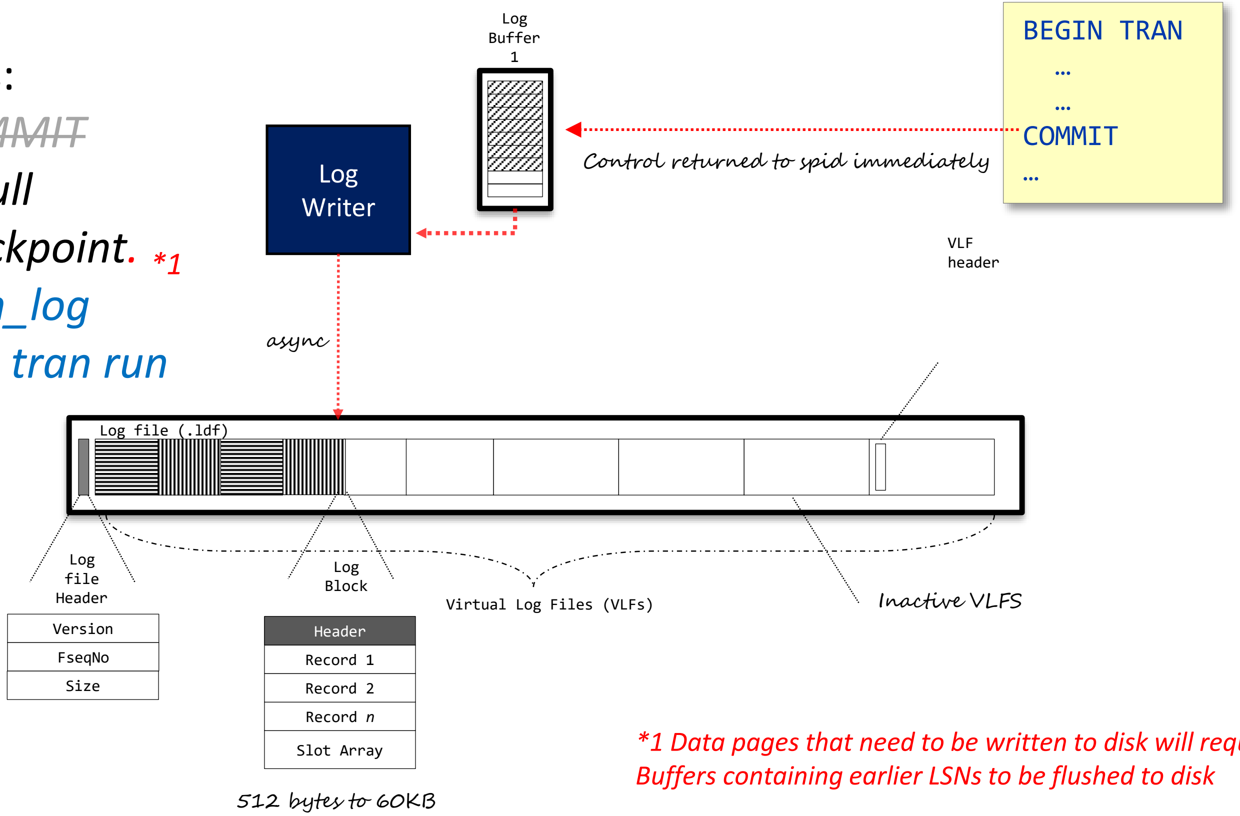
SQL IO review

- 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.

Using Delayed Durability

Log flushes:

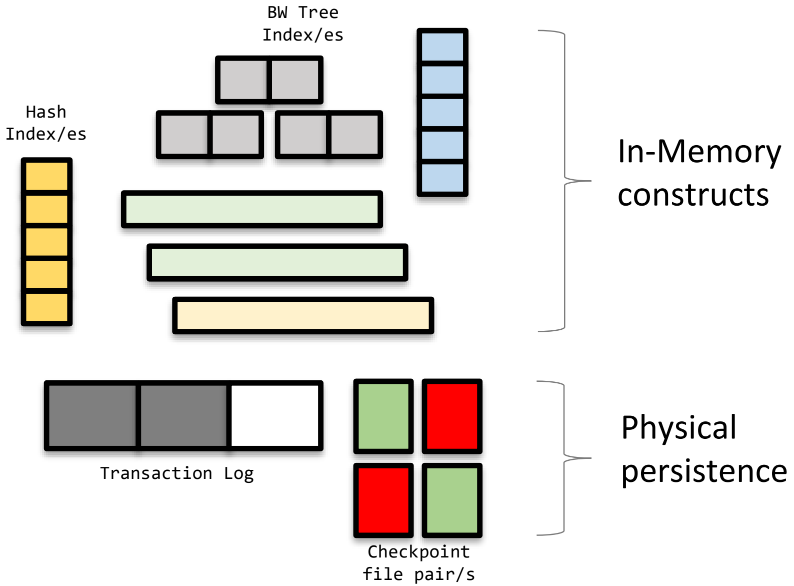
- ~~On COMMIT~~
- *When full*
- *On Checkpoint.* *1
- *sp_flush_log*
- *Durable tran run*



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

Using In-Memory OLTP

- 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 thru interop or Natively Compiled Stored Procedures
- No fragmentation concerns.



Demo

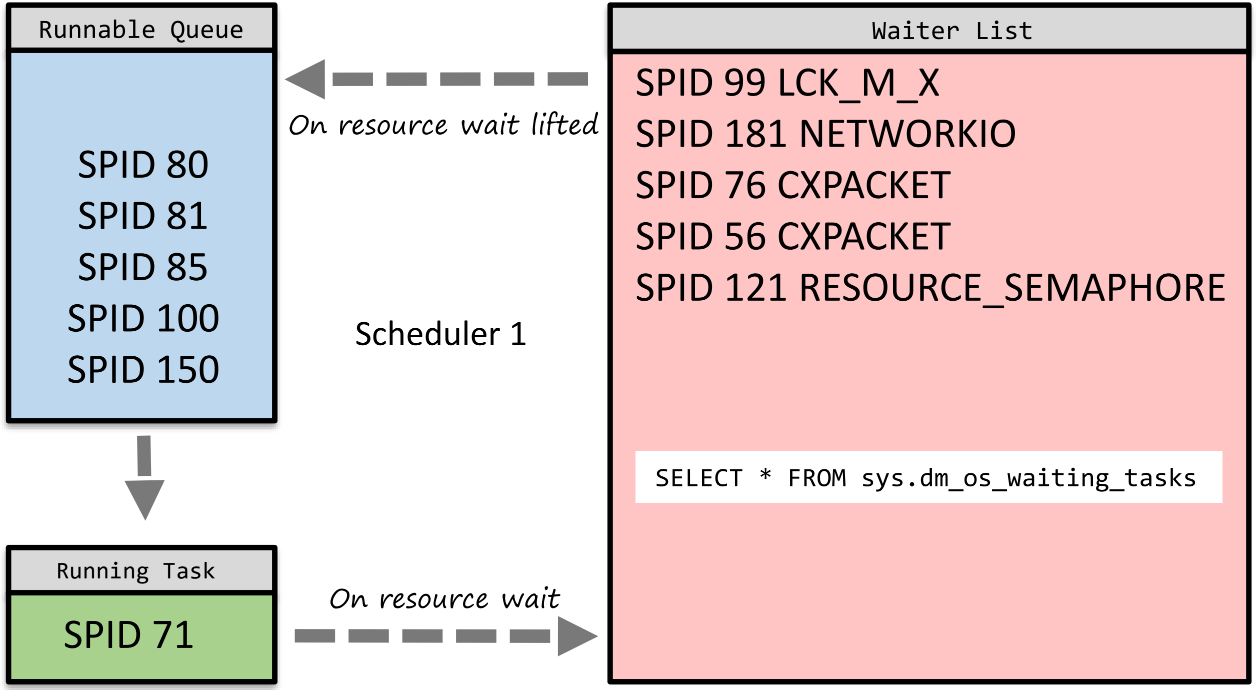
Making Improvements and utilizing other technologies

CPU (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

THEREFORE

- 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 (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

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.

In summary

- Keep transactions short and complex logic
- Use RCSI or SI isolation – especially on disk
- Capture, review, and optimise all query plans
- Implement new technologies to improve concurrency
- Test your results in parallel to prove consistency.