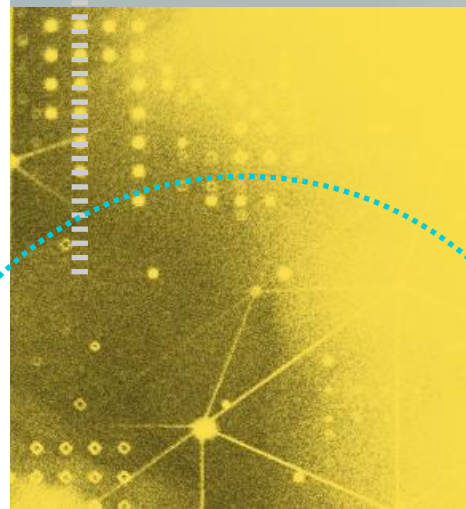
solarwinds

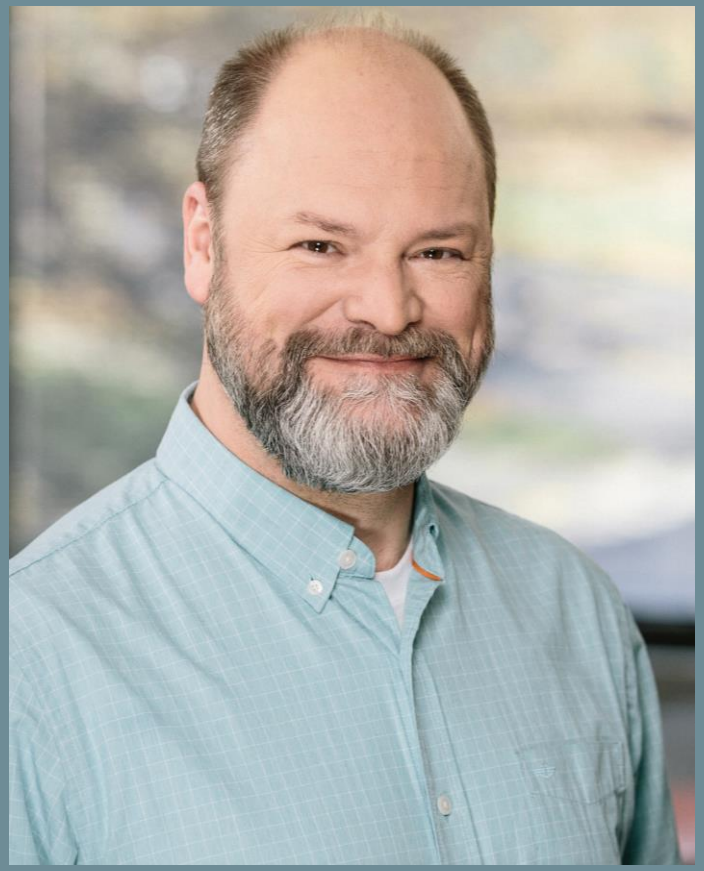
# How to Troubleshoot SQL Server CPU Issues

For SQL Server and Azure SQL database systems



# Kevin Kline

Technology Evangelist - Databases



Kevin has over 35+ years' experience as a developer, data scientist, DBA, enterprise architect, and IT manager with over a dozen books to his credit. He was also the founder and former president of PASS.org. When not working, he enjoys gardening, traveling, writing, teaching, and indie rock.


twitter.com/kekline

bsky.com/kekline

facebook.com/kekline

linkedin.com/in/kekline/

# Agenda

For SQL Server and Azure SQL

Let’s talk about SQL Server® CPU issues

Most common CPU issues?

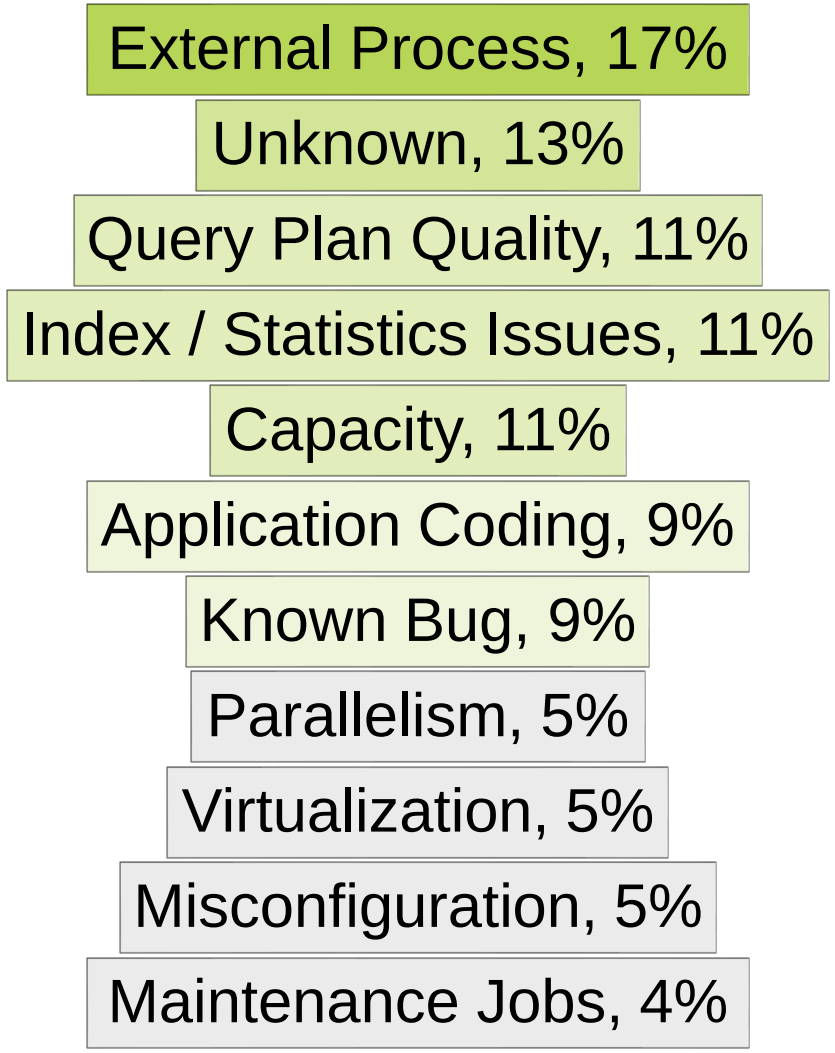
Checklist for SQL Server® CPU troubleshooting

Basic CPU troubleshooting strategies

- Fix the right problem
- Find the root cause of memory issues
- Remediate the root cause

# Most Common CPU Issues on SQL Server

From Microsoft CSS



# SQL Server CPU Configuration Settings

Makes a big difference when misconfigured:

- CPU power option setting, very commonly ignored. Don't!
- vCPU power settings also apply

Makes a noticeable difference:

- Max degrees of parallelism – ymmv, recommend 0, 4, or 8 depending OR number of cores per NUMA node
- Cost threshold for parallelism – ymmv, recommend 30 or more

Rarely makes a positive difference:

- Max worker threads - do not adjust unless told to by CSS
- Priority boost - obsolete from Win2k era
- Processor affinity - use Resource Governor instead
- Lightweight pooling – obsolete, from Win2k era

# CPU Troubleshooting Checklist

1. Shortcut - Has anything changed?
2. Determine whether the issue is *inside* or *outside* of SQL Server?
3. Is the issue caused by a SQL Server error?
4. What do the wait statistics say?
5. Correlate wait statistics against other metrics, Ring Buffers, or QS
6. Follow-up and post-mortem

# Your Shortcuts – Track Changes and Know Baselines

Remember, change = risk

Your best shortcut is to know what has changed:

- Instance-level, use sp_configure or sys.configurations
- Database-level, use sp_dboption or sys.databases
- Changes to database objects, use DDL triggers for meta-data changes:
  - Developers with SA access to production databases?
  - Other options exist, like Change Data Capture and Audit

Know your baseline performance metrics. Compare accordingly.

# Error Logs – Steps 2 and 3

Is the issue inside or outside of SQL Server? Is the issue an error?

- Windows Application Log helps eliminate non-SQL Server problems
- SQL Server Error Log and SQL Server Agent Log
  - Available both as TXT and through SSMS
  - SQL Server keeps the six most recent, cycling with each reboot

WARNING!

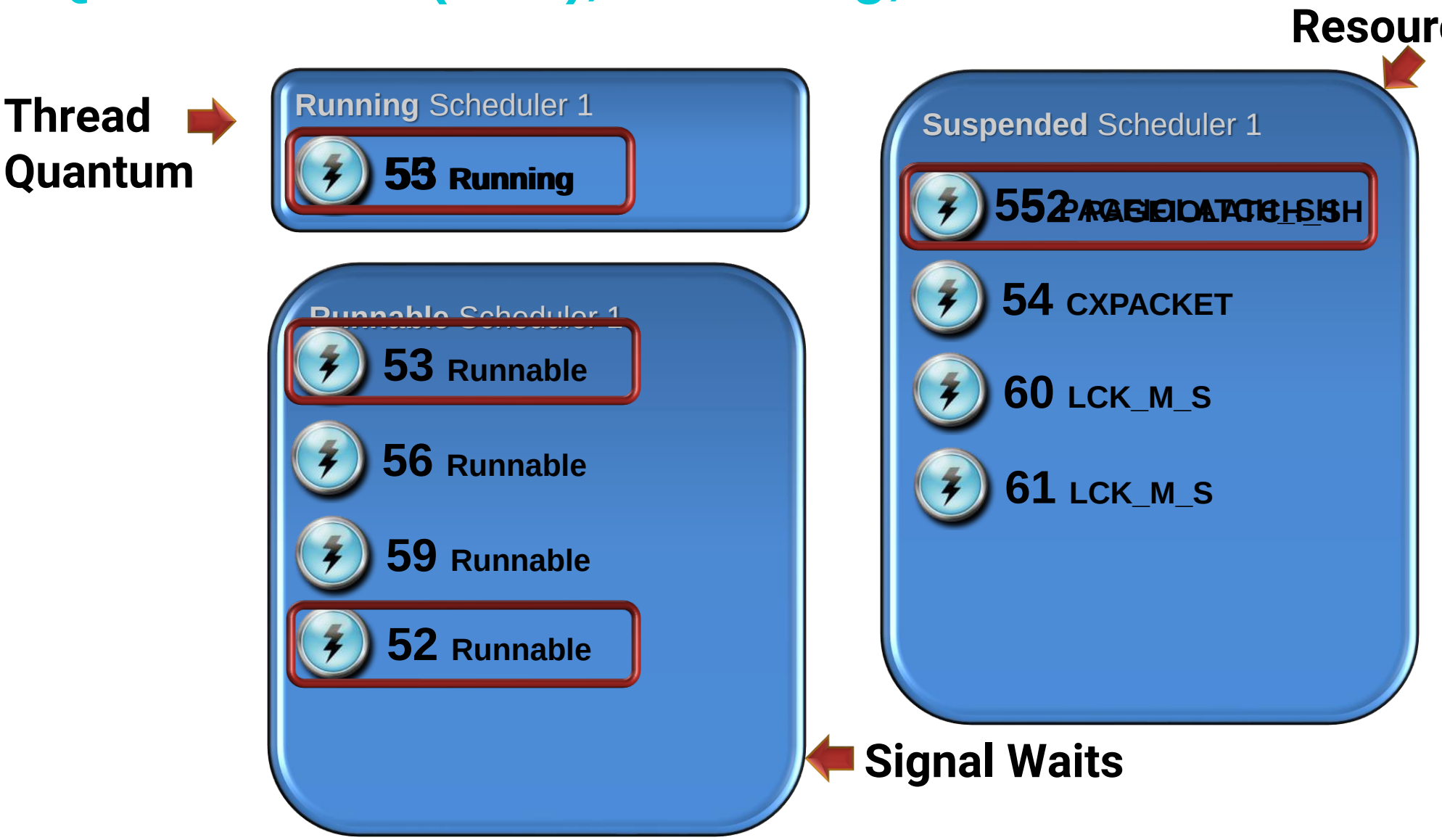
*Always make sure to enable SQL Server Agent notifications for high severity errors (either severity level 16 or 18)*

# Wait Statistics – Step 4

Make sure to fix the correct problem

- 1. Anytime a process in SQL Server waits for something:
  - It is reported as a wait type, a cumulative value since list reboot
  - Reveals where to find the bottlenecks, but not the root cause
- 2. Two broad categories:
  - *Signal wait* is the total time spent waiting for CPU
  - *Resource wait* is the total time spent waiting for system resources, such as memory, IO, locks, etc. There are *lots* of subcategories.

# SQL Server OS (SOS), Threading, and Waits




Bookmark the amazing Wait Stats Library by Paul Randal at https://www.sqlskills.com/help/waits/

*Credit to Joe Sack at Microsoft for this animation*

# Top 10 Wait Statistics from the Field

**CPU PRESSURE**

**CPU pressure**: SOS_SCHEDULER_YIELD

**Parallelism**: CXPACKET

**LOCKING**

**Long term blocking**: LCK_X, LCK_M_U, & LCK_M_X

**MEMORY**

**Data buffer latch**: PAGELATCH_X

**Non-data buffer latch**: LATCH_X

**Memory grants**: RESOURCE_SEMAPHORE

**I/O**

- **Buffer I/O latch**: PAGEIOLATCH_X
- **Tran log disk subsystem**: WRITELOG & LOGBUFFER
- **General I/O issues**: ASYNC_IO_COMPLETION & IO_COMPLETION

**NETWORK PRESSURE**

- **Network I/O**: ASYNC_NETWORK_IO

# DEMO

*CPU Diagnostics and Root-Cause Analysis*

SQL Server can tell you exactly which databases, workloads, and transactions need to be tuned.

These Transact-SQL scripts identify SQL Server CPU issues and show you which databases, files, and elements of SQL Server are most relevant.

# Metrics to Remember From the Demo

| Tool | Monitors | Granularity |
|---|---|---|
| SOS_SCHEDULER_YIELD | Yielding processor time due to thread / context switching | SQL Server instance level *(cumulative since last start)* |
| System Health Session (Xevents) | Wait stat for data retrieving data from the buffer pool | SQL Server instance level *(cumulative since last start)* |
| SELECT serverproperty('processed') to PID | Useful for differentiating against multiple named instances | SQL Server instance level *(cumulative since last start)* |
| Processor: % Processor Time *(look for 80% or less)* Processor: % Privilege Time *(look for 30% or less of % Processor Time)* | PerfMon counters showing % of CPU used by Windows vs % of CPU used by SQL Server kernel actions, like IO requests. | Per second |
| Process (sqlservr): % Processor Time Process (msdmdsrv): % Processor Time | PerfMon counters showing % elapsed processor time spent on SQL Server and SSAS process threads. Watch for a rise above 80%. | Per second |
| System: Processor Queue Length *(look for less than 12)* System: Context Switches/sec *(track for baseline value, beware of sustained overages)* | PerfMon counters showing the number of PIDs waiting in queue for CPU and number of execution context switched in the last second. | Per second |

# DMVs to Remember From the Demo

| Tool | Monitors | Granularity |
|---|---|---|
| sys.dm_os_wait_stats<br>Sys.dm_os_waiting_tasks | Memory wait stats in total and by SPID | SQL Server Instance level<br>*(cumulative since last start)* |
| sys.dm_os_ring_buffers | Server level CPU info | SQL Server Instance level<br>*(cumulative since last start)* |
| sys.dm_exec_requests<br>sys.dm_exec_sql_text<br>sys.dm_exec_sessions<br>sys.dm_exec_connections | See running SPIDS, where they come from, and what SQL they are running. | SQL Server Instance level<br>*(cumulative since last start)* |
| sys.dm_exec_query_plan<br>sys.dm_exec_query_plan | Shows parallelism. Look at plan operators for the former,look for total_elapsed_time < total_worker_time for latter (not useful in heavy blocking scenarios) | SQL Server Instance level<br>*(cumulative since last start)* |
| sys.dm_os_performance_counters | Batch Requests/sec, SQL Compile/sec, then SQL Re-compile/sec, measured in ratio of no more than ~30% per counter | SQL Server Instance level<br>*(per second)* |

# Issues at the Root

Most common findings from a troubleshooting exercise

- Sucktacular SQL (see slide notes for many additional SQL problems), for example:
  - Bad parallelism
  - Implicit conversion
  - UDFs
- Missing or obsolete firmware drivers and updates
- Drivers injected into the Windows driver stack, e.g., anti-virus, auditing and change tracking, encryption, etc.
- Misconfigured or inadequate hardware

# Summary - Important External Troubleshooting Steps

## Ensure there are no CPU Errors in the Windows log

- CPU issues can be a server-crashing event

## Ensure there are no CPU errors in the SQL Server log

- Many SQL Servers will tell you their CPU problem, such as a low memory warning

## Make sure there are no recent, unexplained configuration changes or changes to database objects

- Sometimes, a person causes havoc by sneaking in an unexpected change to a setting, an or application,

# Summary

Remember that CPU problems on SQL Server and Azure SQL are most often due to code and/or design reasons (e.g., sucktacular SQL code, poor database design, poor indexing) rather than hardware (e.g., misconfigured, overloaded, or underperforming hardware).

Properly segregating your IOPs according to purpose, workload, and available resources will help a lot.

The biggest return on investment is frequently better SQL code. Reconfiguring can help, but often only by a small percentile. Data compression and partitioning can also help.

Remember the important DMVs, wait stats, PerfMon counters, and system information from the demo!

1. Always start with wait types. CPU problems often appear with in combination with high I/O wait types.
2. Use DMVs to determine the source of CPU consumption linked to the specific database and query.
3. Cross-reference PerfMon metrics and DMVs for CPU to identify the exact issue and processes or queries
4. Tune queries and possibly lower-level configuration settings or, in rarer cases, change hardware.

# POLL

Would you like to learn more?

# Q&A

# THANK YOU

Slides and scripts for this webcast and others at:

https://thwack.solarwinds.com/members/kekline/blogs

Contact me at:
kevin.kline@solarwinds.com

Social media at @kekline on LinkedIn, Twitter, and Facebook.

solarwinds.com/DPA

solarwinds.com/DPM

sentryone.com/SQLSentry