SQLBits

Session: Thursday, 17:10

Introduction to Performance Troubleshooting Using Wait Statistics (L2-300)

Paul S. Randal
paul@SQLskills.com

SQLskills
immerse yourself in sql server

Paul S. Randal

- Consultant/Trainer/Speaker/Author
- CEO, SQLskills.com
  - Email: Paul@SQLskills.com
  - Blog: https://www.SQLskills.com/blogs/Paul
  - Twitter: @PaulRandal
  - 5 years at DEC responsible for the VMS file-system and chkdsk
  - Almost 9 years as developer/manager in the SQL Storage Engine team through August 2007, ultimately responsible for Core Storage Engine
- Instructor-led training, consulting (anything you need)
- Online training: http://pluralsight.com/
- Get our newsletter: https://www.sqlskills.com/Insider

Overview

- Very common to see ‘knee-jerk’ performance tuning where someone jumps to a conclusion based on superficial analysis of performance data
- Interpreting wait statistics is not hard, but needs practice

- We’re going to cover
  - Introduction
  - Thread lifecycle
  - Waits and wait times
  - DMVs
  - Some common wait types

Interpreting the Data

- Don’t do ‘knee-jerk’ performance troubleshooting
  - Work through the data to see what may be the root cause
  - You’ll end up spending less time overall
- Proficiency in using wait statistics data comes from:
  - Retrieving the data correctly
  - Understanding what common wait types mean
  - Recognizing patterns
  - Avoiding inappropriate Internet advice
  - Practice!
- Better to have a series of snapshots of wait statistics over time

What are Waits?

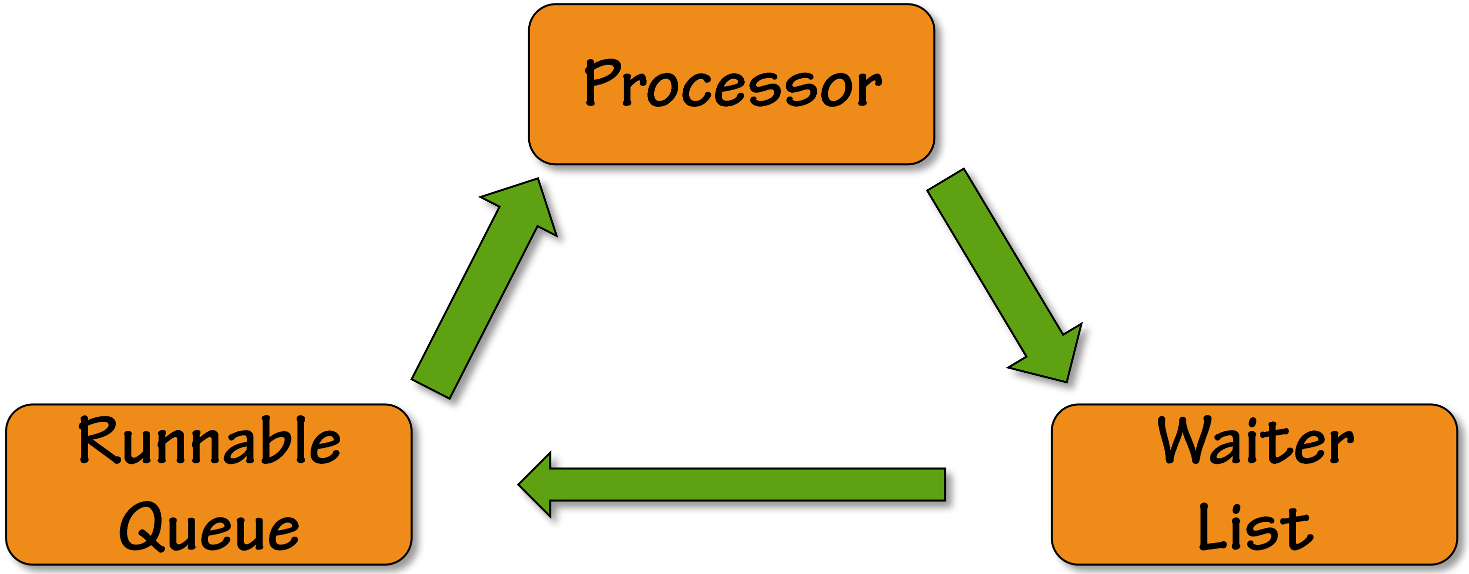
- The term ‘wait’ means that a thread running on a processor cannot proceed because a resource it requires is unavailable
  - It has to wait until the resource is available
- The resource being waited for is tracked by SQL Server
  - Each resource maps to a wait type
- Example resources that may be unavailable:
  - A lock (LCK_M_XX wait type)
  - A data file page in the buffer pool (PAGEIOLATCH_XX wait type)
  - Results from part of a parallel query (CXPACKET wait type)
  - A latch (LATCH_XX wait type)

Thread Scheduling

- SQL Server performs its own thread scheduling
  - Called non-preemptive scheduling
  - More efficient for SQL Server than relying on Windows scheduling
  - Performed by the SQLOS layer of the Storage Engine
- Each processor core (whether logical or physical) has a scheduler
  - A scheduler is responsible for managing the execution of work by threads
  - Schedulers exist for user threads and for internal operations
  - Use the sys.dm_os_schedulers DMV to view schedulers
- When SQL Server has to call out to the OS, it must switch the calling thread to preemptive mode so the OS can interrupt it if necessary

Components of a Scheduler

- All schedulers are composed of three ‘parts’



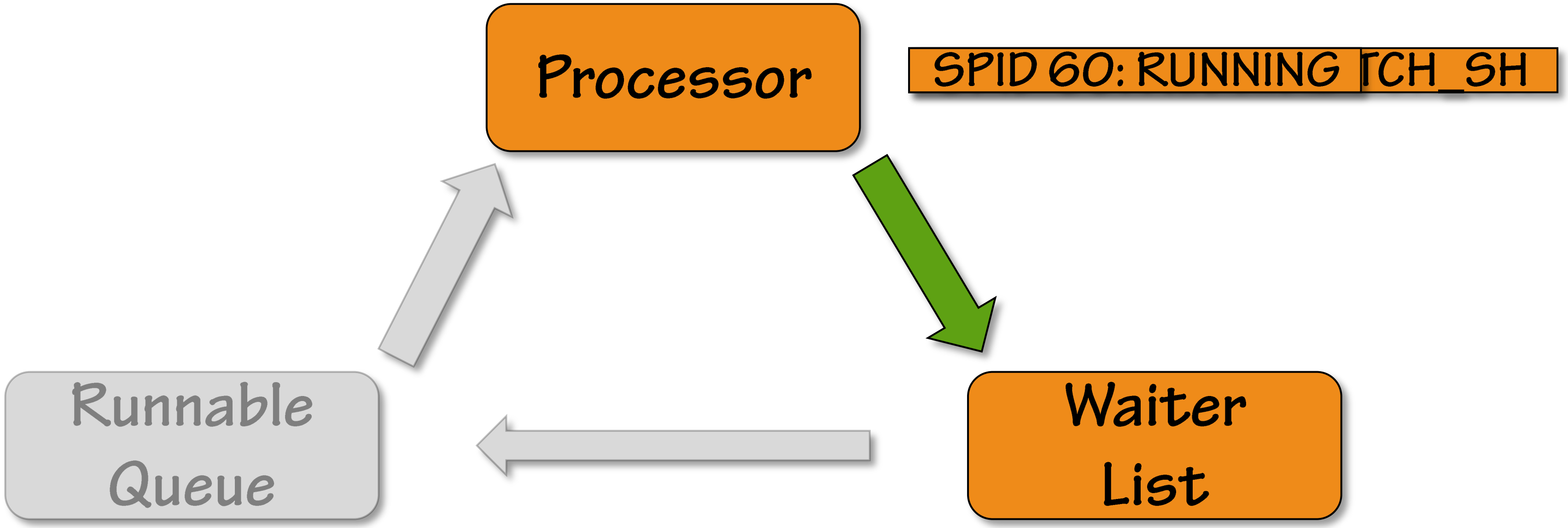
- Threads transition around these until their work is complete

Thread States

- A thread can be in one of three states when being actively used as part of processing a query
- RUNNING
  - The thread is currently executing on the processor
- SUSPENDED
  - The thread is currently on a Waiter List waiting for a resource
- RUNNABLE
  - The thread is currently on the Runnable Queue waiting to execute on the processor
- Threads transition between these states until their work is complete

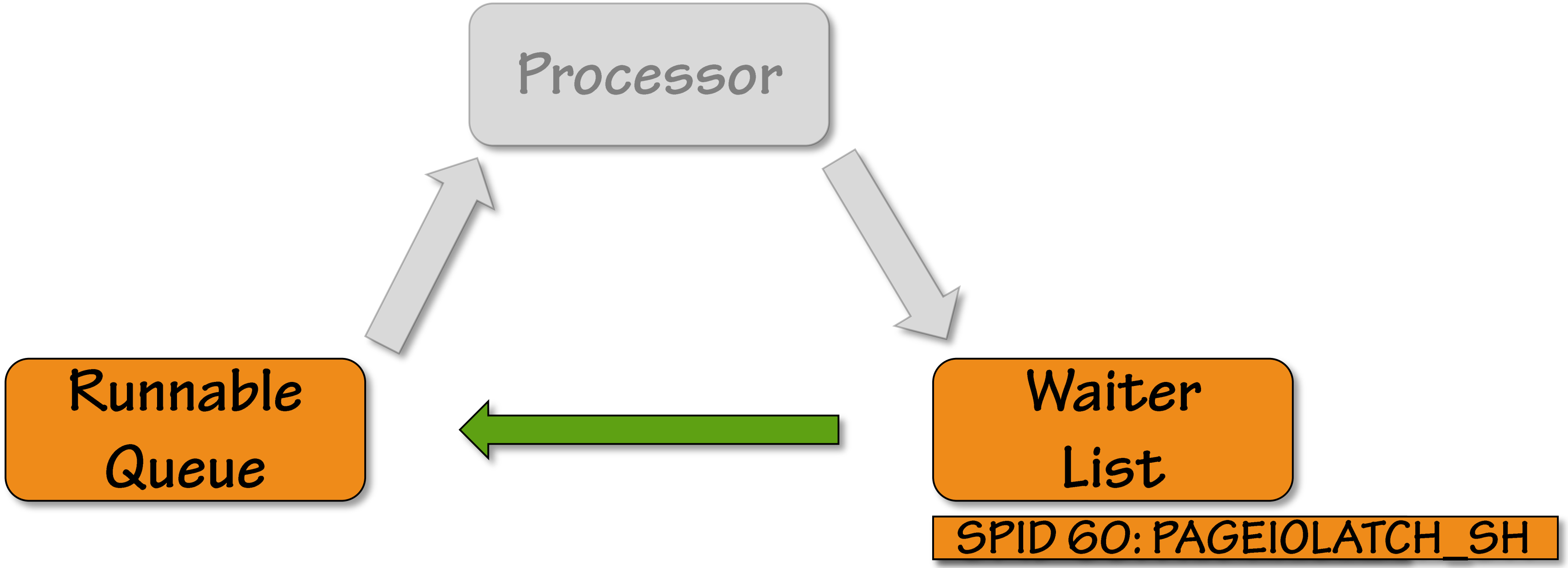
Transition: RUNNING to SUSPENDED

- A thread continues executing on the processor until it must wait for a resource to become available
  - The thread's state changes from RUNNING to SUSPENDED
  - The thread has been 'suspended' and moves to a Waiter List



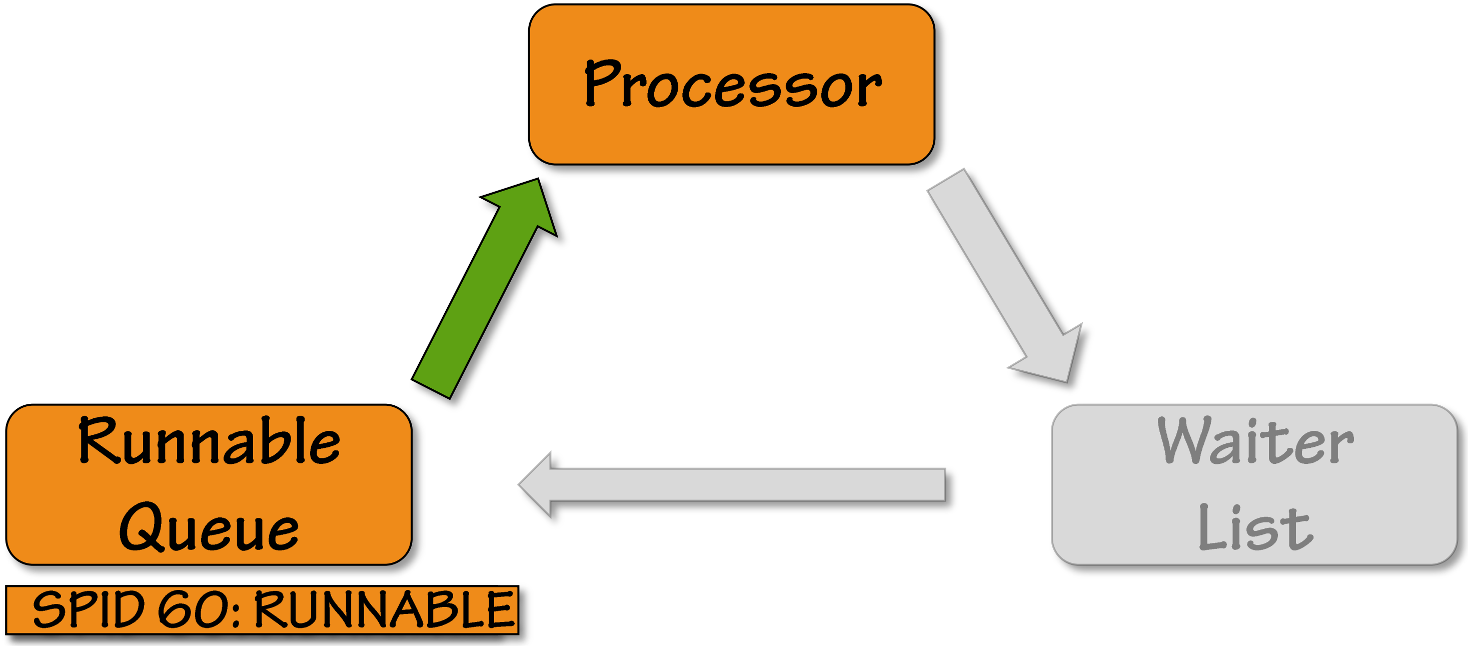
Transition: SUSPENDED to RUNNABLE

- A thread continues to wait until it is told that the resource is available
  - The thread’s state changes from SUSPENDED to RUNNABLE
  - The thread moves to the Runnable Queue
  - This is called being ‘signaled’

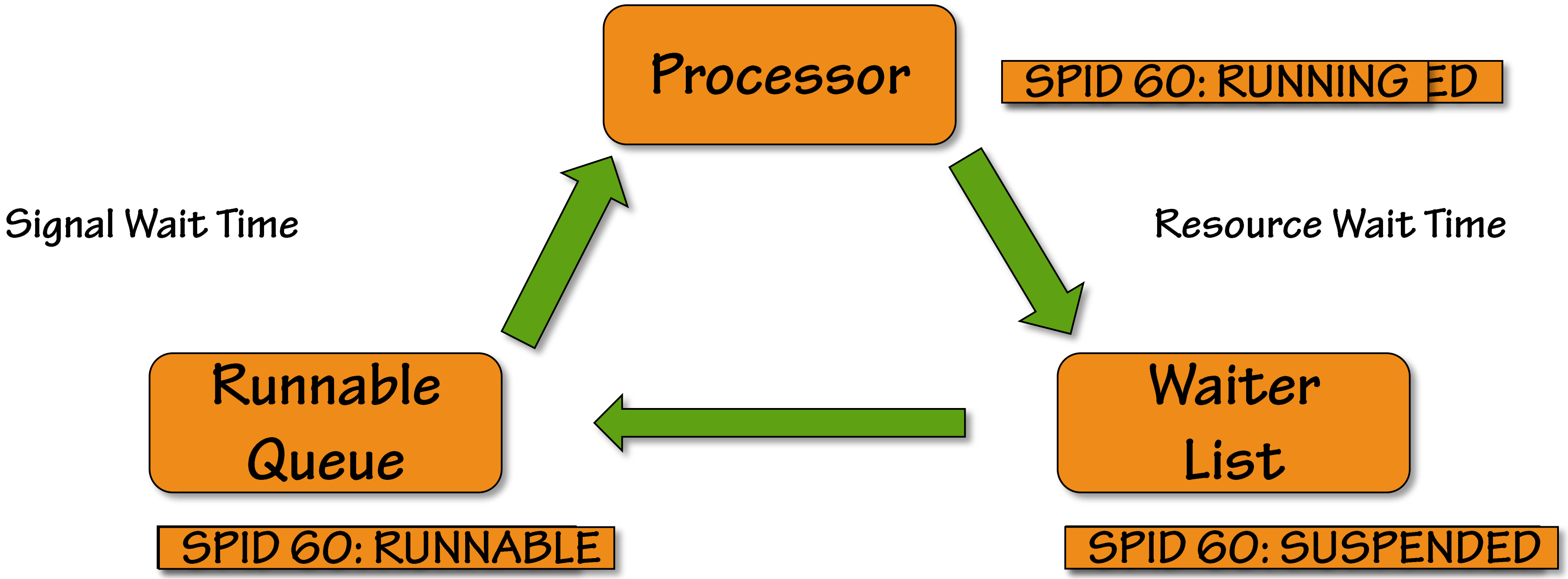


Transition: RUNNABLE to RUNNING

- The thread waits on the Runnable Queue until it is picked as the next thread when the processor becomes available
  - The thread's state changes from RUNNABLE to RUNNING
  - 2019+: it might move to a different scheduler in the same NUMA node



Wait Times Definition



Wait Time = Resource Wait Time + Signal Wait Time

sys.dm_os_waiting_tasks DMV

- This DMV shows all threads that are currently suspended
- Think of it as the ‘what is happening right now?’ view of a server
  - Usually very first thing to run when approaching a ‘slow’ server
- Most useful information this DMV provides:
  - Session ID and execution context ID of each thread
  - Wait type for each suspended thread
  - Description of the resource for some wait types
    - E.g. for locking wait types, the lock level and resource is described
  - Wait time for each suspended thread
  - If the thread is blocked by another thread, the ID of the blocking thread
    - Show what’s the head of a blocking chain and can show non-intuitive patterns

sys.dm_os_wait_stats DMV

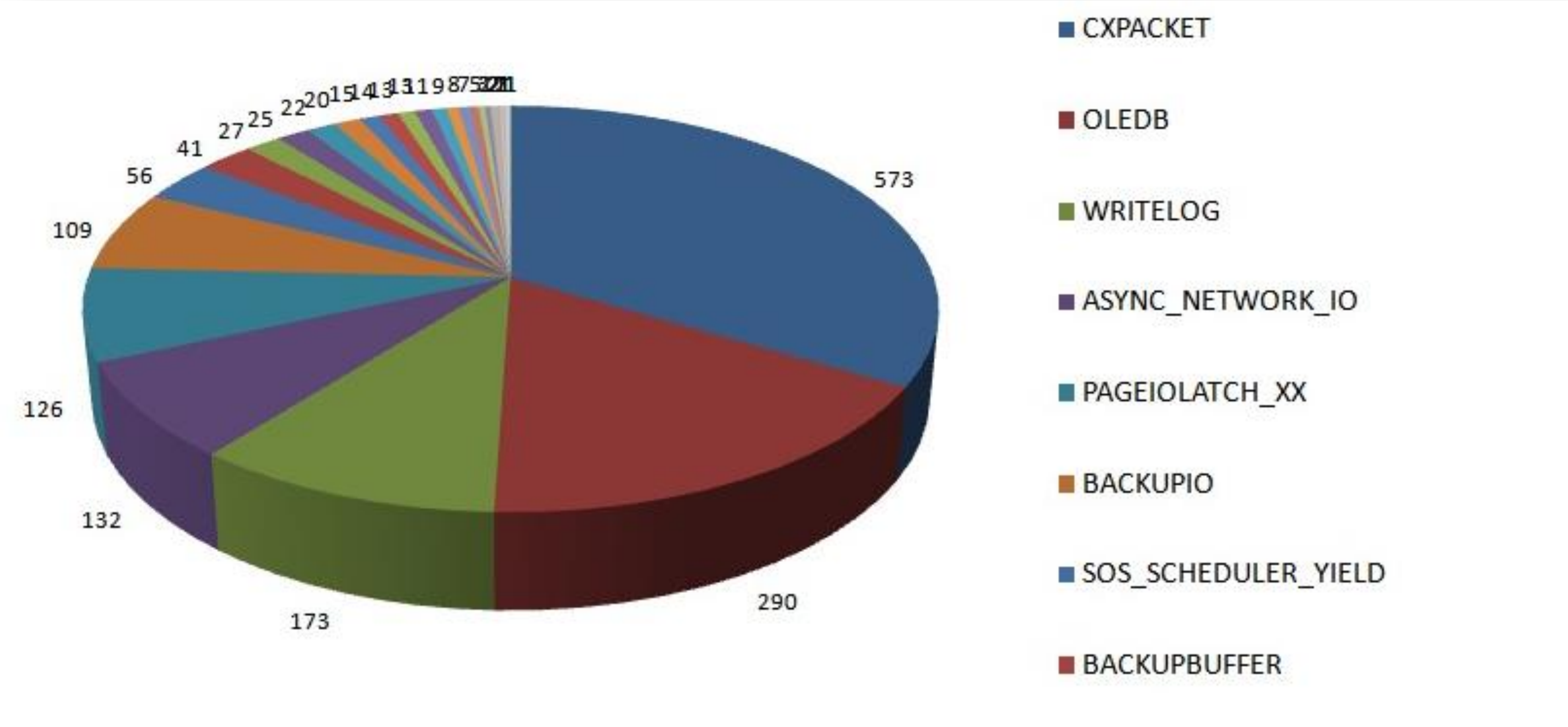
- This DMV shows aggregated wait statistics for all wait types
  - Aggregated since the server started or the wait statistics were cleared
- Think of this as the 'what has happened in the past?' view of a server
- Most useful information this DMV provides :
  - The name of each wait type
  - The number of times a wait has been for this wait type
  - The aggregate signal and overall wait times for all waits for this wait type
- Some math is required to make the results useful
  - Calculating the resource wait time
  - Calculating the average times rather than the total times

What’s Relevant?

- An extremely important point to bear in mind is that waits ALWAYS occur inside SQL Server
  - Look for actionable items and filter out things like background tasks
  - Filter out benign waits such as WAITFOR, LAZYWRITER_SLEEP
    - Look at the demo code to see what I mean
- Need to identify the top, relevant waits and then drill in
- Example:
  - 1000 waits for LCK_M_S: Is it a problem?
  - No, if that was over 8 hours, there were 10 million locks acquired, and total wait time for the LCK_M_S locks was only 50s altogether
  - Yes, if each wait was for 50s

Top Wait Types

- Survey results from 1700+ SQL Server instances across Internet



Source: my blog at https://sqlskills.com/p/083

PAGEIOLATCH_XX Wait and Solutions

- Waiting for a data file page to be read from disk into memory
  - Common modes to see are SH and EX
- Do not assume the I/O subsystem or I/O path is the problem
- Further analysis:
  - Determine which tables/indexes are being read
  - Analyze I/O subsystem latencies with sys.dm_io_virtual_file_stats
    - Move the affected data files to faster I/O subsystem?
  - Correlate with CXPACKET waits, suggesting parallel scans
    - Create appropriate nonclustered indexes and/or update statistics
  - Examine query plans for parallel scans and implicit conversions
  - Investigate buffer pool memory pressure and Page Life Expectancy
    - If data volume has increased, consider increasing memory

PAGELATCH_XX Wait and Solutions

- Waiting for access to an in-memory data file page
  - Common modes to see are SH and EX
- Do not confuse these with PAGEIOLATCH_XX waits
- Does not mean add more memory or I/O capacity
- Further analysis:
  - Determine the page(s) that the thread is waiting for access to
  - Classic tempdb contention?
    - Add tempdb data files, enable trace flag 1118, reduce temp table usage
    - 2019 helps with this, including system tables in memory
  - Analyze the table and index structures involved
    - Excessive page splits occurring in indexes
    - Insert-point hotspot in a clustered index with ever-increasing key

LCK_M_XX Wait and Solutions

- A thread is waiting for a lock that cannot be granted because another thread is holding an incompatible lock
- Do not assume that locking is the root cause
- Further analysis:
  - Follow blocking chain to see what the lead blocking thread is waiting for
  - Use blocked process report to capture info on queries waiting for locks
    - Michael Swart's blog post (https://sqlskills.com/p/090)
  - Lock escalation from a large update or table scan?
    - Consider a different indexes, snapshot isolation, a different isolation level, or locking hints
  - Something preventing a transaction from releasing its locks quickly?
      - E.g. synchronous DBM/AG, DTC, or log throughput problems

Demo: Insert hotspot and using the DMVs

WRITELOG Wait

- What does it mean:
  - Waiting for a transaction log block buffer to flush to disk
- Avoid knee-jerk response:
  - Do not assume that log file I/O system has a problem (can be the case)
  - Do not create additional transaction log files
- Further analysis:
  - Correlate WRITELOG wait time with I/O subsystem latency using sys.dm_io_virtual_file_stats
  - Look at average size of transactions and for extra log being generated
  - Look at average disk write queue length for log drive
    - If constantly 31/32 (111/112 on SQL 2012+) then the internal limit has been reached for outstanding transaction log writes for a single database

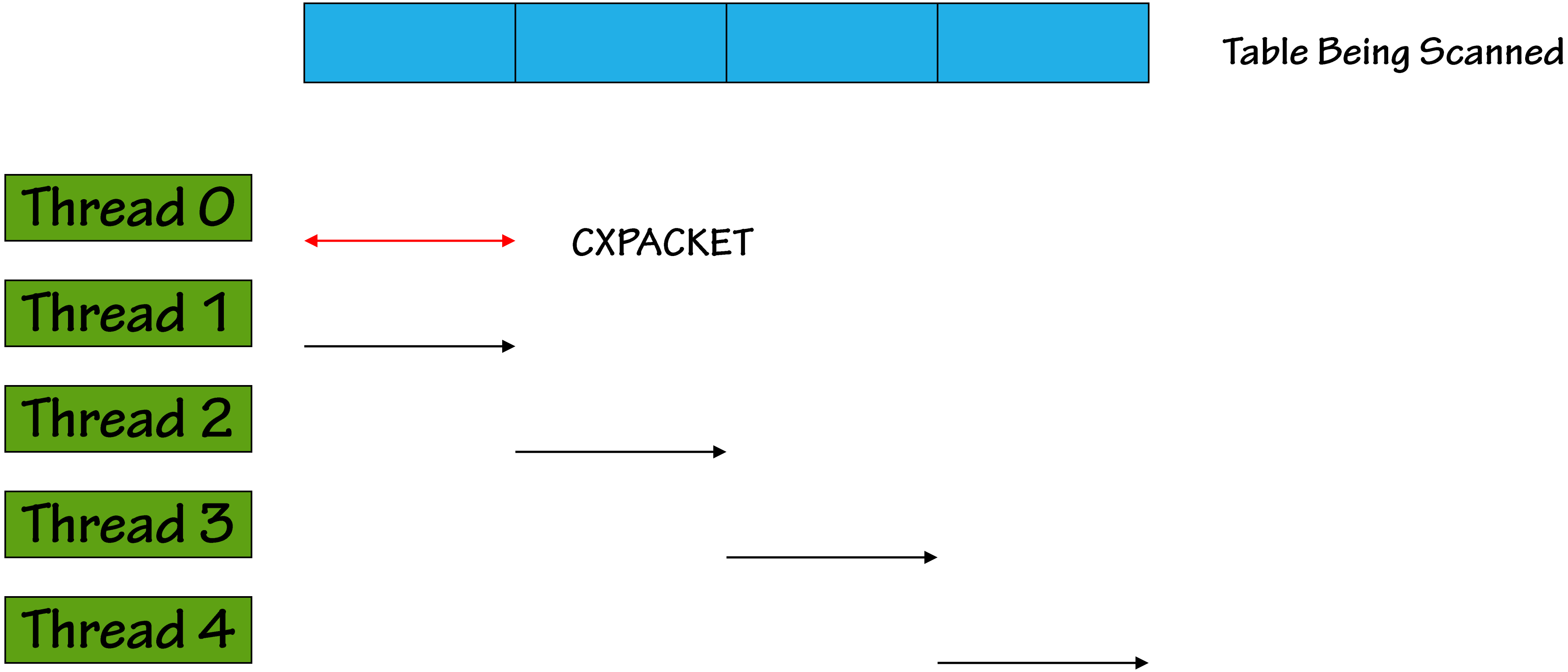
WRITELOG Wait Solutions

- Move the log to a faster I/O subsystem
- Increase size of transactions to prevent many tiny log block flushes
- Remove unused nonclustered indexes to reduce logging overhead from maintaining unused indexes during DML operations
- Check for incorrect CACHE size of SEQUENCE objects
- Change index keys or introduce FILLFACTORs to reduce page splits
- Investigate whether synchronous database mirroring/AGs/SAN replication is introducing delays
- Potentially split the workload over multiple databases or servers
- SQL Server 2014: delayed durability and In-memory OLTP

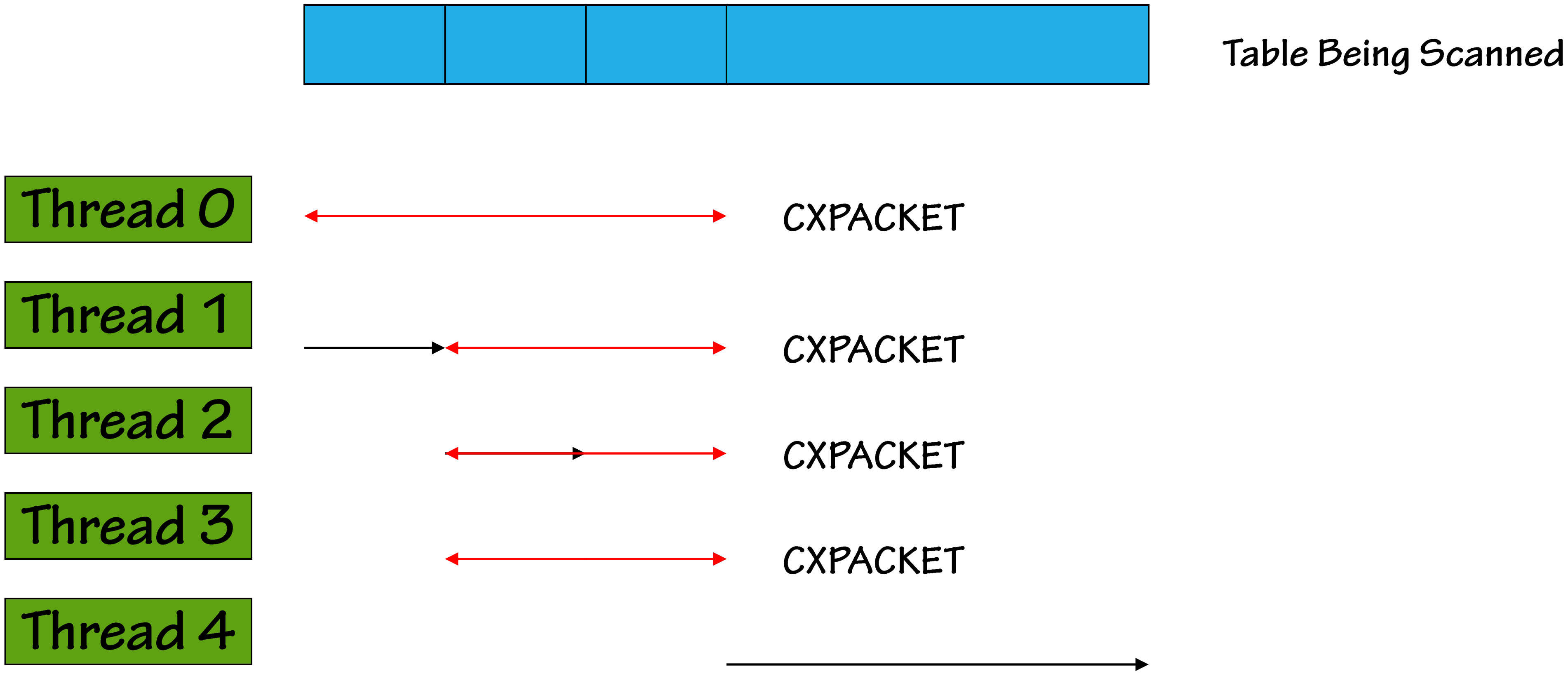
CXPACKET Wait Explanation

- What does it mean:
  - Parallel operations are taking place
  - Accumulating very fast implies skewed work distribution amongst threads or one of the workers is being blocked by something
- Avoid knee-jerk response:
  - Do not set server-wide MAXDOP to 1, disabling parallelism
- Further analysis:
  - Correlation with PAGEIOLATCH_SH waits? Implies large, parallel scans
  - Examine CXPACKET query plan to see if the query plans make sense
  - Are there non-zero ID threads showing CXPACKET wait?

CXPACKET Wait Example (1)



CXPACKET Wait Example (2)



CXPACKET Wait Solutions

- Possible root-causes:
  - Just parallelism occurring
  - Table scans because of missing nonclustered indexes or incorrect query plan
  - Out-of-date statistics or cardinality issue causing skewed work distribution
- If there is actually a problem:
  - Make sure statistics are up-to-date and appropriate indexes exist
  - MAXDOP for a query? Or just a database (in 2016+)? Or Resource Governor?
  - MAXDOP for the instance? Test to figure out best value for *you*:
    - No NUMA then = # logical cores, up to max of 8
    - NUMA = # logical cores per NUMA node, up to 16 (2016+) or 8 (< 2016)
    - General guidance, soft-NUMA complicates this
  - Set ‘cost threshold for parallelism’ higher to avoid parallel plans
    - Jon’s blog post at https://sqlskills.com/p/094 provides a guestimate

Demo: Parallelism

ASYNC_NETWORK_IO Wait

- What does it mean:
  - SQL Server is waiting for a client to acknowledge receipt of sent data
- Avoid knee-jerk response:
  - Do not assume that the problem is network latency
- Further analysis:
  - Analyze client application code, client app server, network latencies
- Possible root-causes and solutions:
  - Usually poorly-coded application that is doing RBAR (Row-By-Agonizing-Row)
    - Very easy to show using a large query and SSMS on same machine as SQL Server
  - Could be from using MARS with large result sets or BCP inbound
    - Also look for network issues, incorrect duplex settings, or TCP chimney offload problems (see https://sqlskills.com/p/102)

OLEDB Wait

- What does it mean:
  - The OLE DB mechanism is being used
- Avoid knee-jerk response that problem is linked servers
- Further analysis:
  - What are the queries doing that are waiting for OLEDB?
  - If linked servers are being used, what is causing delay on linked server?
- Possible root-causes:
  - DBCC CHECKDB and related commands use OLE DB internally
  - Many DMVs use OLE DB internally so it could be a third-party monitoring tool that is repeatedly calling DMVs (especially if they're very short waits)
  - Poor performance of a linked server

Summary: Methodology

- Gather information about exactly when the performance problem arose and the user-visible characteristics of the problem
- Gather information about what changed before the problem arose
- Examine the output from sys.dm_os_waiting_tasks
  - What is happening on the server right now?
- Examine the output from sys.dm_os_wait_stats
  - What has happened in the past?
- Look at the top 3-4 relevant waits
- Avoid temptation to knee-jerk and equate symptoms with root-cause
- Gather further information from relevant sources
    - DMVs, query plans, performance counters, code analysis

Resources

- Comprehensive waits/latches library: https://www.SQLskills.com/helps/waits
- Whitepapers:
  - SQL Server Performance Tuning Using Wait Statistics: A Beginners Guide
    - https://sqlskills.com/p/103
  - Diagnosing and Resolving Latch Contention on SQL Server
  - Diagnosing and Resolving Spinlock Contention on SQL Server
    - Gnarly links – see our whitepapers page at https://sqlskills.com/p/104
- Blog post categories
  - https://www.sqlskills.com/blogs/paul/category/wait-stats/ /latches/ /spinlocks/
- Pluralsight: SQL Server: Performance Tuning Using Wait Statistics

Thank you!

Questions? Paul@SQLskills.com

SQLskills

immerse yourself in sql server