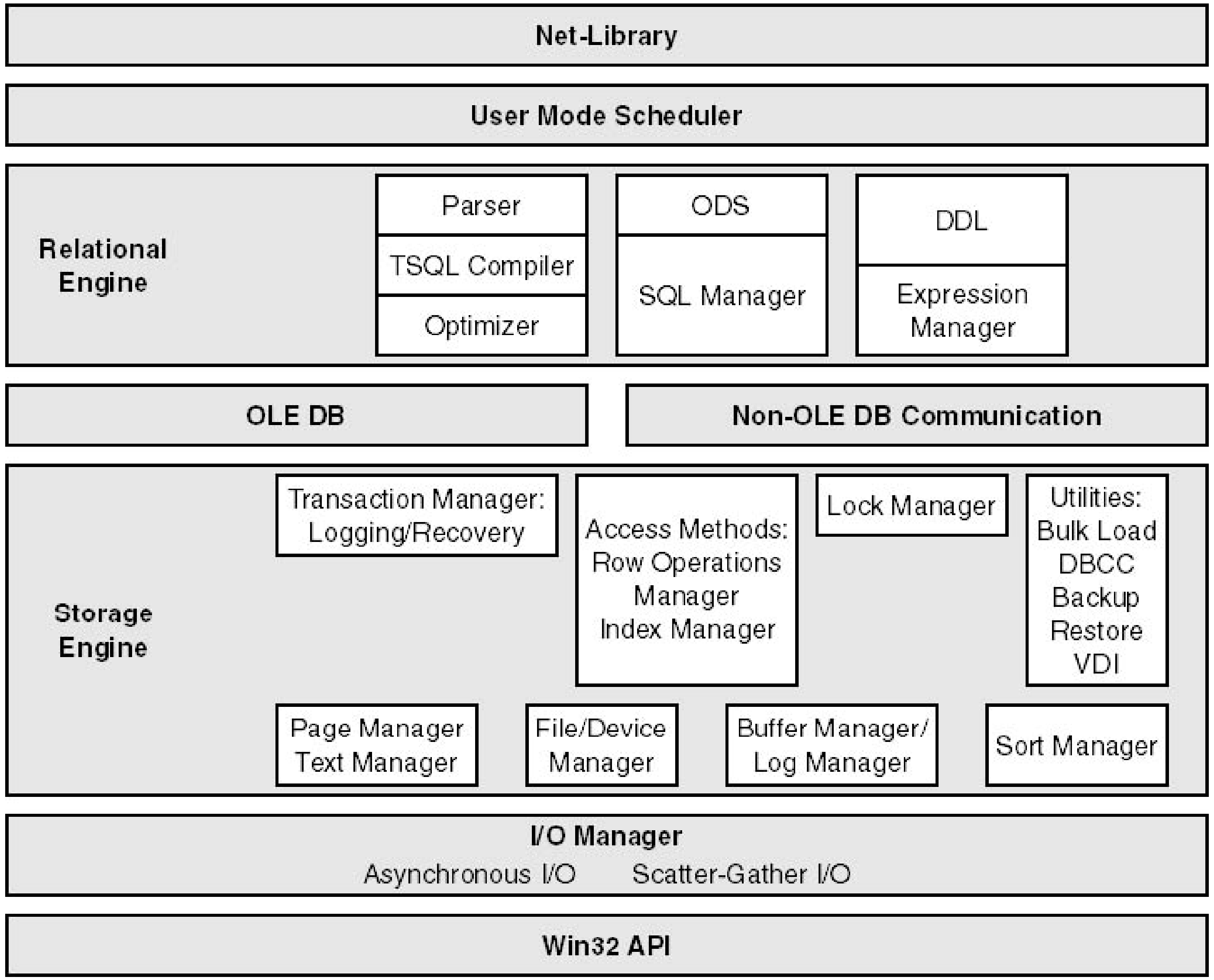
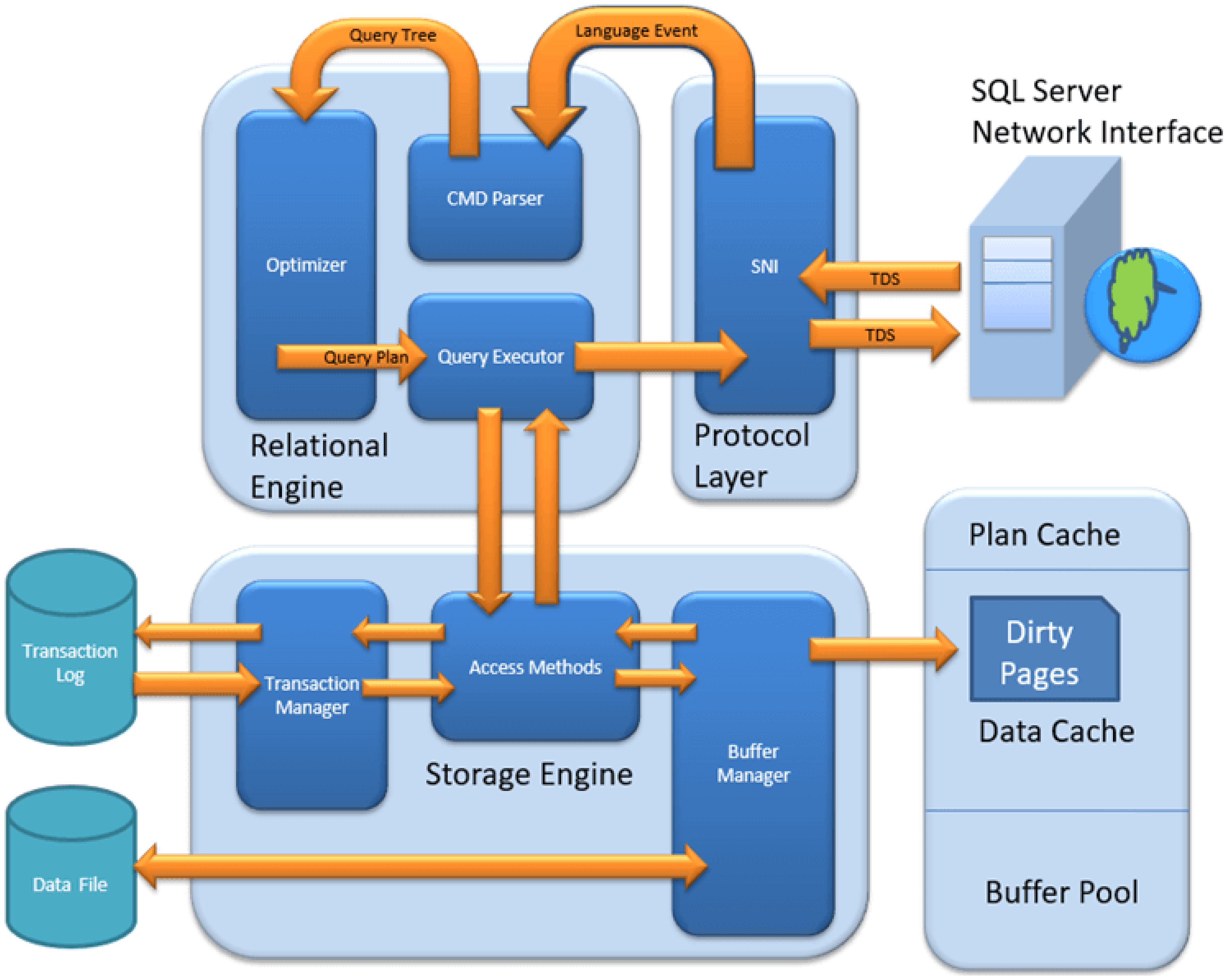
Performance Tuning

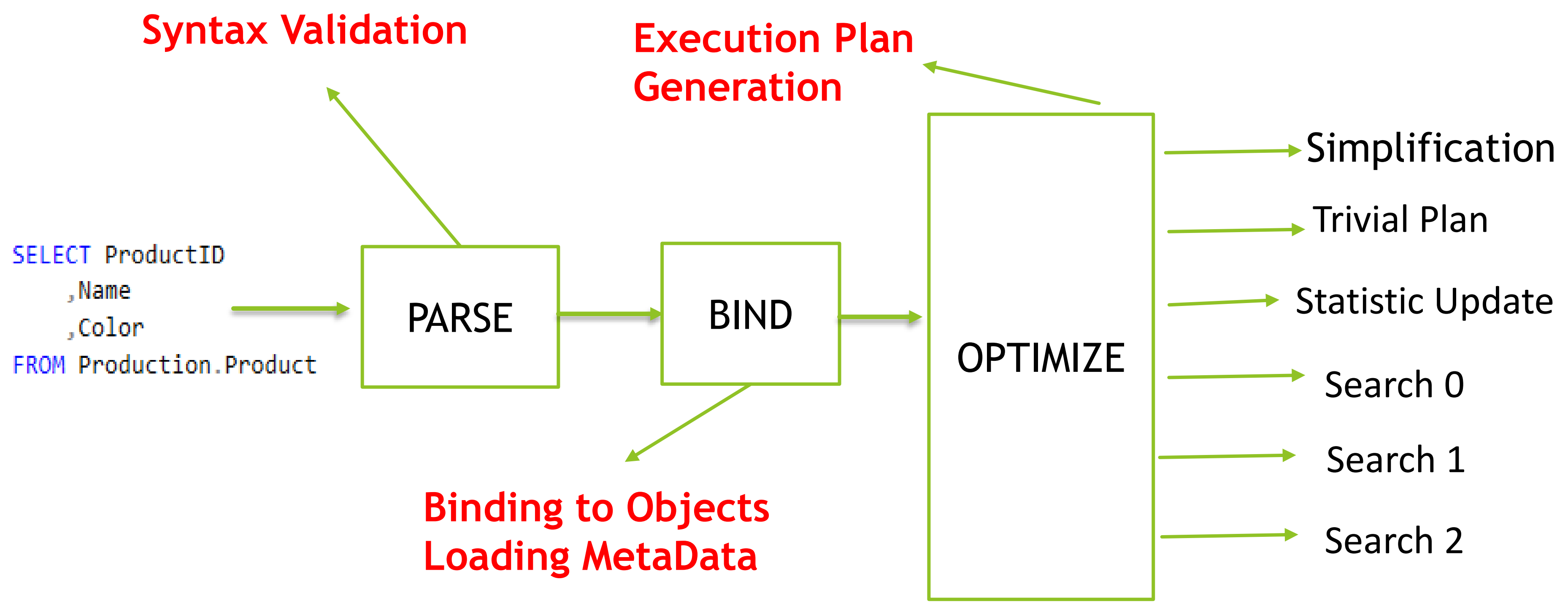
SQL Architecture components



SQL Architecture – How queries are processed



Query Life Cycle



```sql
SELECT ProductID
      ,Name
      ,Color
FROM Production.Product
```

The goal of Query Optimization is not to find BEST Execution plan, to find a *good enough* execution plan.

SQL OS or UMS(User Mode Scheduler)

What You Will Learn

- Scheduling Basics
- Preemptive vs Non-Preemptive scheduling
- UMS basics
- Troubleshooting UMS issues

Preemptive vs Non-Preemptive scheduling

- Windows uses a priority-driven, preemptive scheduling algorithm
  - Threads execute within their quantum
  - Once the quantum expires, threads are "scheduled off" and must wait for NT to schedule them again.
  - Potential for threads to be preempted before quantum expiration by higher-priority threads.
- UMS is a cooperative *user-mode* scheduling sub-system
  - Simply put, UMS is a thread pool.
  - Allows SQL to be smart about when it waits
  - Allows SQL-specific management of CPU time.
  - UMS is *logical not physical:* We do not change NT scheduling behavior, we just manage it.

UMS – Design Goal

- Control which thread should run from SQL server. We try to run one thread per CPU(Scheduler) but there are exceptions.
- Priority is irrelevant as far as scheduling is concerned.
- Not complete scheduling (just some construct to make a better decision when we allow OS to schedule our threads)
- Avoid causing a thread to switch into kernel mode whenever possible
- Support asynchronous I/O

Terms

Quantum

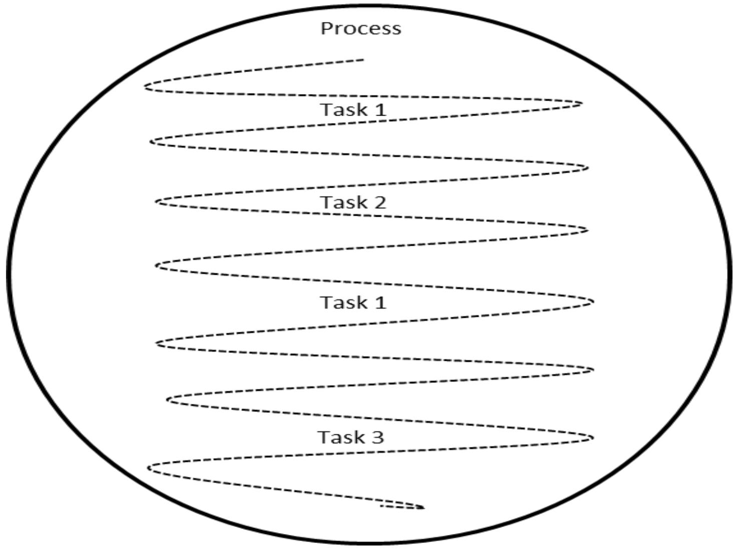
- The time slice a thread is given to run by the Windows scheduler.
- Windows creates the illusion that all threads run concurrently by partitioning the time each thread is allowed to run into time slices called quantum's. It hands quantum to threads in a round-robin fashion.

Context Switch

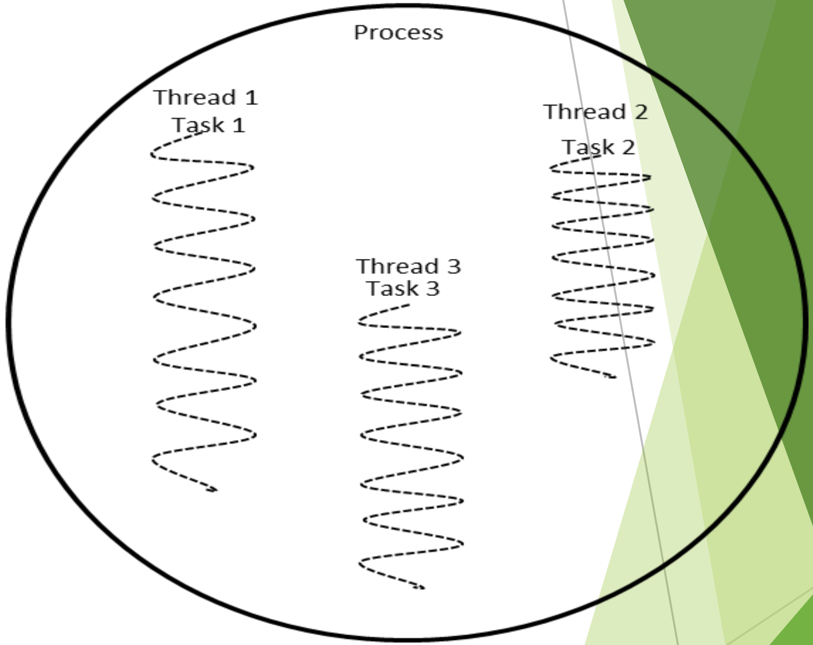
- Saving info for old thread
- Loading info for new thread

Process & Threads

Process as Execution Unit



Threads as Multiple Execution Contexts in a Process



Key Take Away

- Processor is just a container which provide infrastructure for threads to execute.
- Process does not execute or run rather Threads associated with the process runs on processor.
- A process as such contains several individual tasks that may or may not be dependent on each other. This makes us to do a design change.

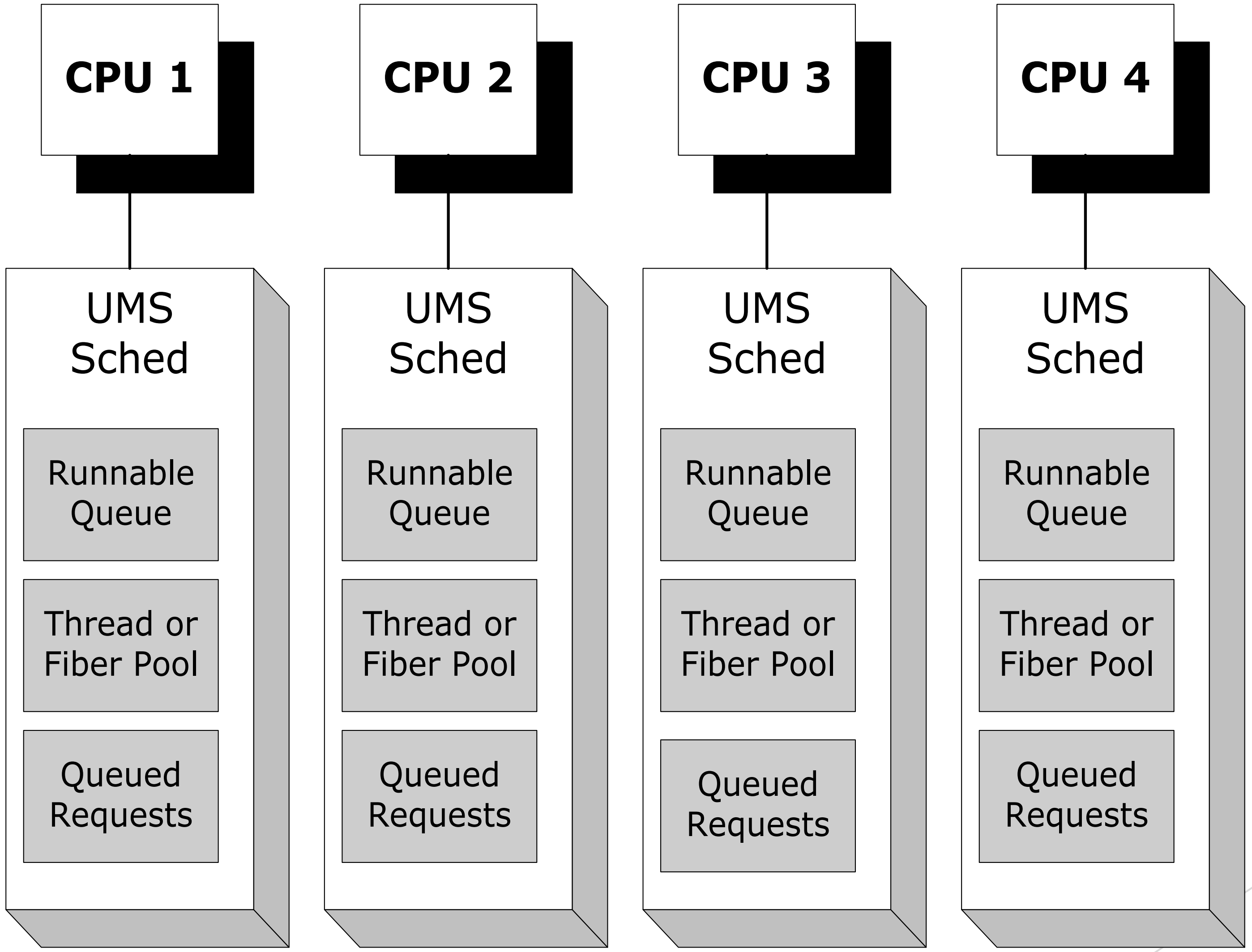
What a thread is

- A thread is unit of execution within a process
- Multiple threads can exist within a process
  - Each is given small amount of processor time
  - Waits while other threads execute (called scheduling)
- SQL Server uses OS threads to perform its work
  - Called worker threads
  - Dedicated threads exists
    - Such as for performing checkpoints, deadlock monitoring, and ghost record cleanup
  - Most are in a pool of threads that SQL Server uses to process user requests (Working Threads)

Thread Scheduling

- SQL Server has its own thread scheduling
  - Called non-preemptive scheduling
  - Is more efficient than Windows scheduling
  - Performed by the SQLOS layer of the Storage Engine
- Each processor core (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

Basic Architecture



Thread States

- RUNNING
  - The thread is currently executing on the processor
- SUSPENDED
  - The thread is currently on the Waiter List waiting for a resource
- RUNNABLE
  - The thread is currently on the Runnable Queue waiting to execute on the processor

Scheduling Infrastructure under UMS

- Runnable Queue
  - A list of processes that are ready to run.
- Waiters List
  - A list of processes that are waiting for some resource to become available.
- Worker Queue
  - A list of processes that are queued to a given scheduler, and are waiting to be serviced by an active worker thread.
- Timer List
  - A list of "timed" processes that are signaled to run on a regular interval.
  - Do not wait on resources like normal "waiters"
  - Examples
    - Lock Monitor is queued to the timer list
    - Voluntary scheduler yields of fixed time
- IO Request List
  - A list of in progress and pending IO requests that need to be "completed"

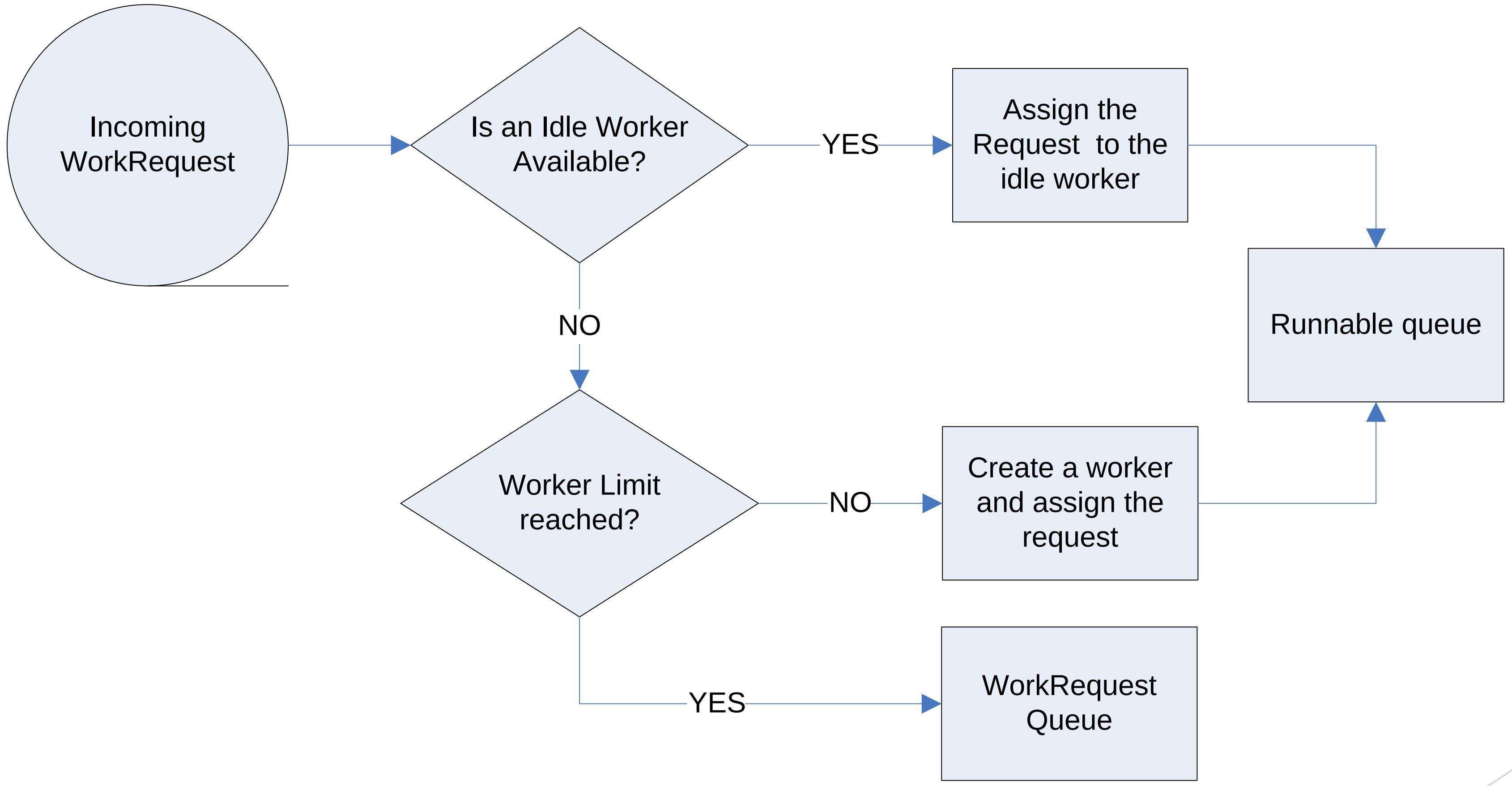
The Waiter List

- Is an unordered list of threads that are suspended
  - Any thread can be notified at any time that the resource it is waiting for is now available
- No time limit for a thread on the waiter list
  - Although execution timeouts or lock timeouts may take effect
- No limit on the number of threads on waiter list
- The sys.dm_os_waiting_tasks DMV
  - Shows which threads are currently waiting and what they are waiting for

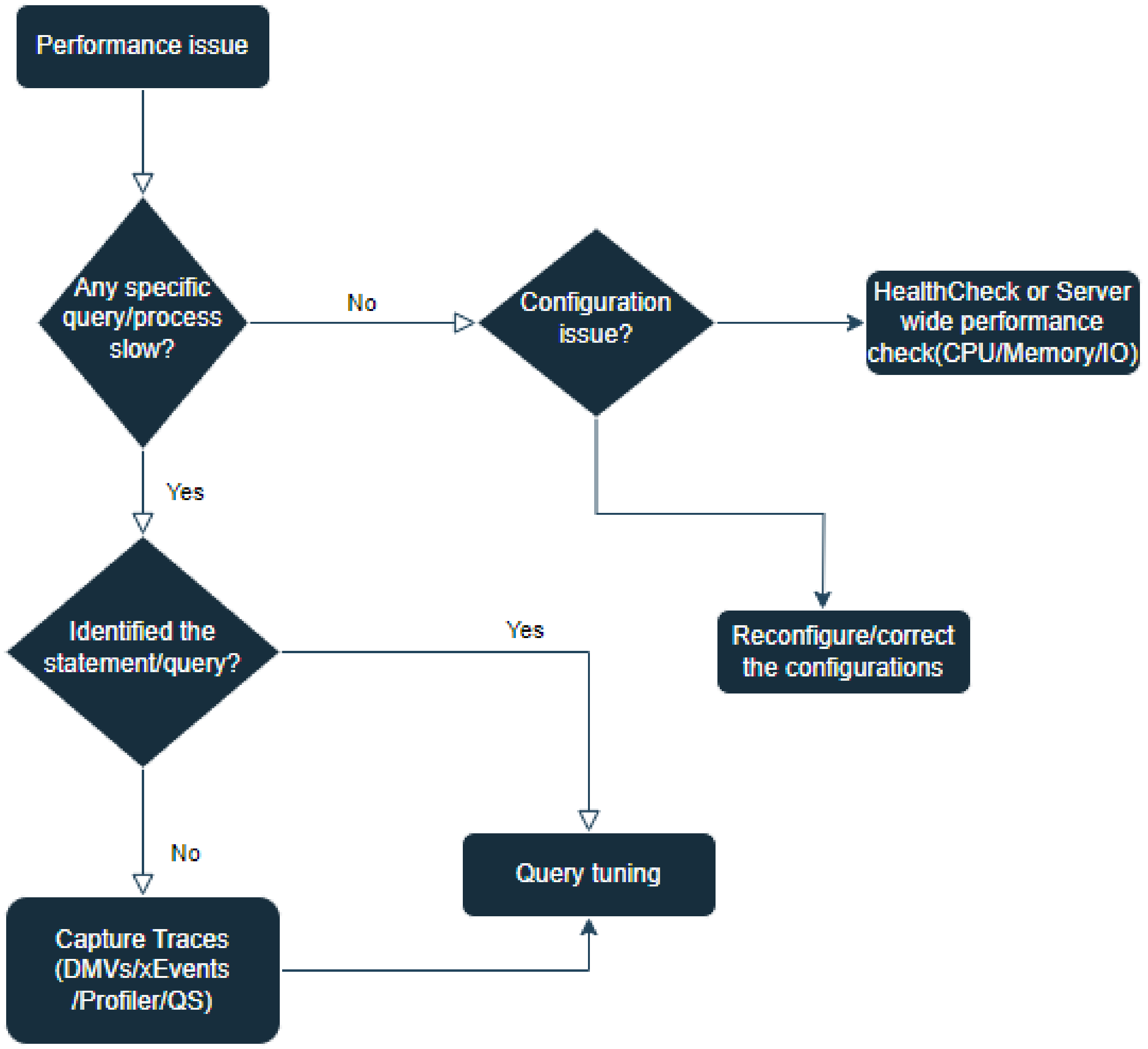
The Runnable Queue

- Is a strict First-In-First-Out (FIFO) queue
  - Resource Governor exception
- The sys.dm_os_schedulers DMV shows the size of the Runnable Queue in Runnable_tasks_count column

Handling WorkRequest Decision Tree


Incoming WorkRequest
Is an Idle Worker Available?
YES
Assign the Request to the idle worker
Runnable queue
NO
Worker Limit reached?
NO
Create a worker and assign the request
YES
WorkRequest Queue

Performance tuning cause map


Performance issue
Any specific query/process slow?
No
Configuration issue?
HealthCheck or Server wide performance check(CPU/Memory/IO)
Yes
Reconfigure/correct the configurations
Identified the statement/query?
Yes
No
Query tuning
Capture Traces (DMVs/xEvents /Profiler/QS)

SQL Server basic configurations

Hardware and Operating System Considerations

- CPU (Number of cores based on the workload)
- Memory configuration
- DISK subsystem
  - Make sure to have a data and log file in a separate disks to have a better performance
- Network

Configure SQL Server

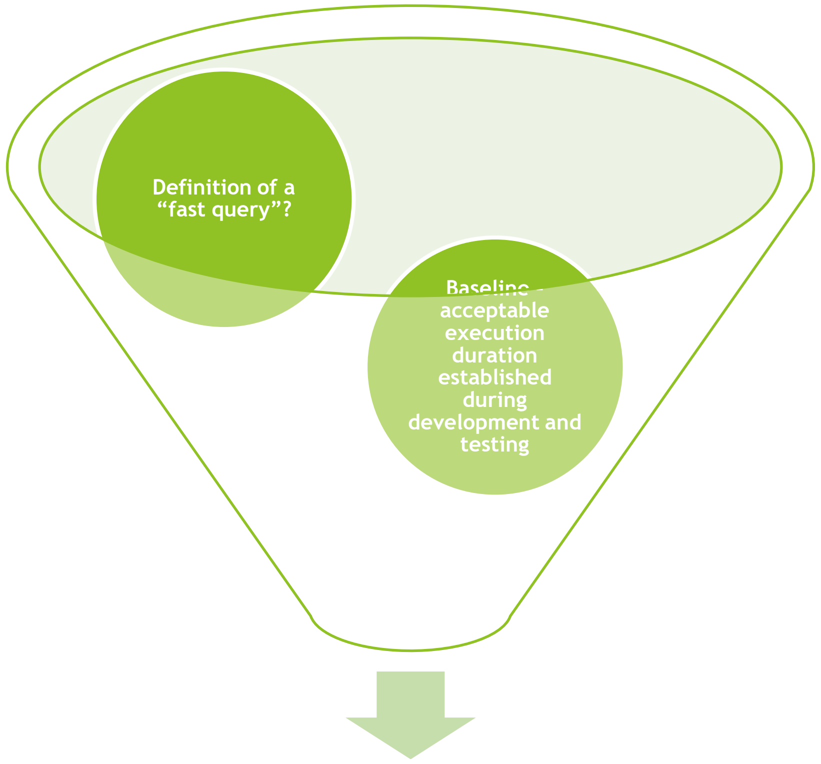
- SQL Server Version and Patching Level to make sure no known issues or bugs (@@version)
- **Instant file initialization**
  - Add SQL account to "Perform Volume maintenance tasks" in local security policies and restart the SQL service
- **LPM(locked pages in memory)**
  - Helps to lock the SQL memory from trimming
- **TempDB configurations**
- **Trace flags**
  - **-T1118**(Prevents mixed extent allocations), not needed starting from 2016 SQL
  - **-T1117**(SQL Server auto-grows all data files in the file group when one of the files is out of space). not needed starting from 2016 SQL
  - **-T4199** (Optimizer fixes)

SQL Server basic configurations – Continue..

- Server wide options
  - **Optimize for Ad-hoc Workloads**: This configuration option controls how SQL Server caches execution plans of ad-hoc. It initially stores the stub of the plan and cache the plan during next execution of the same query
  - Max memory
  - Parallelism
- Database wide options
  - Disable auto shrink
  - Disable auto close(Auto close will remove the pages, plans from memory)
  - Recovery model
  - Database compatibility level
  - VLF(Proper auto growth and initial size for faster recovery)
- Unnecessary monitoring which impacts the performance
  - Like xEvents to capture the statements and the execution plan of the query/statement.

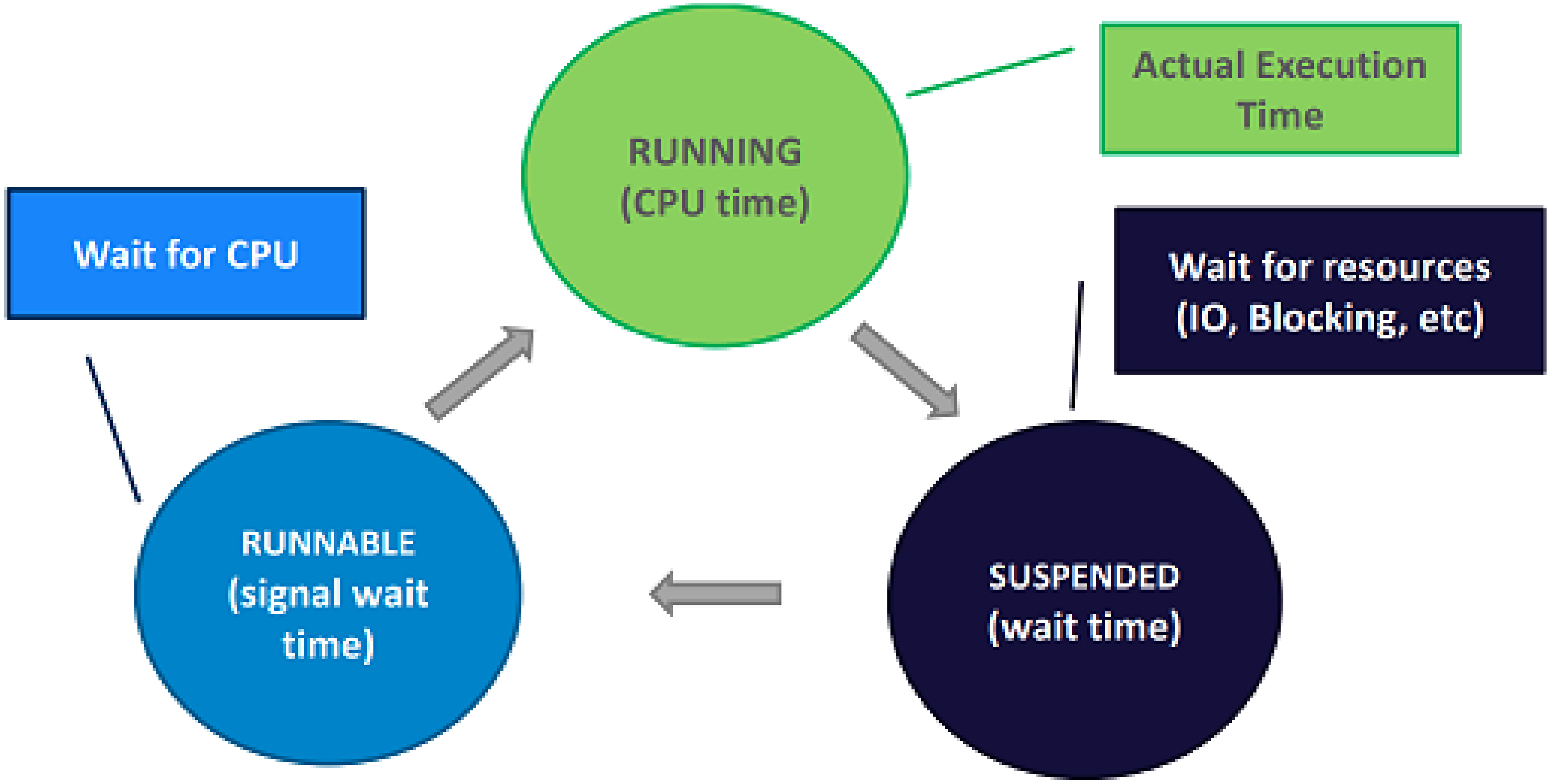
Wait Statistics

What Performance Problem? Show Me Your Baseline

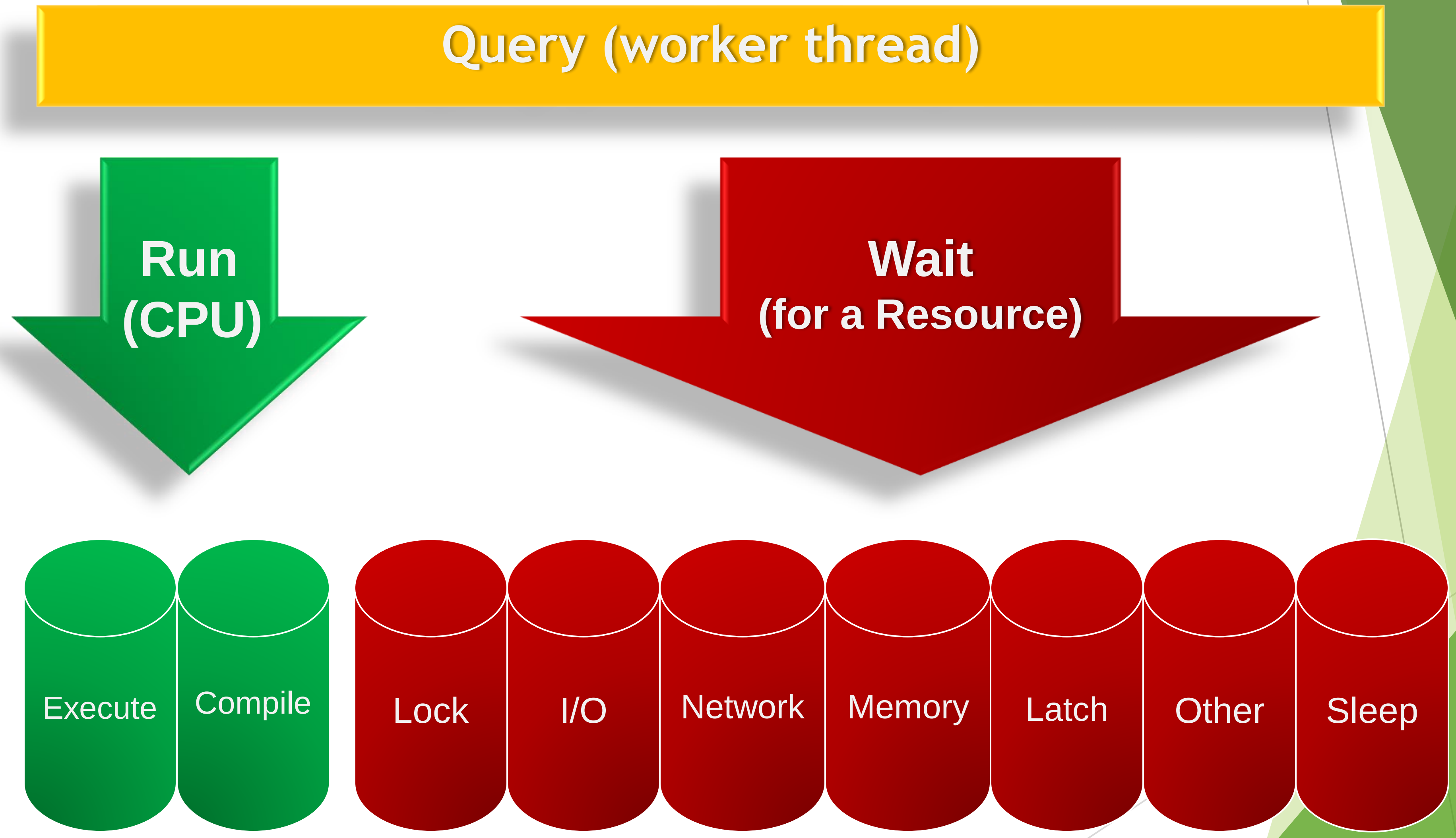

Definition of a "fast query"?
Baseline - acceptable execution duration established during development and testing

No Baseline, No Performance Problem

Query Life Cycle (Simplified)



Run vs. Wait?


Query (worker thread)
Run
(CPU)
Wait
(for a Resource)
Execute
Compile
Lock
I/O
Network
Memory
Latch
Other
Sleep

Wait statistics

Wait time: thread is waiting for a resource

Cumulative wait time: Wait time since the service/service restarted

Signal wait time: A thread in the runnable queue, waiting for CPU cycle to execute

Total wait time: **Waittime (i..e waiting time for resource) + signal wait time(i.e. runnable thread waiting time)**

DMVs to monitor statistics

- sys.dm_os_waiting_tasks
  - Live troubleshooting
- sys.dm_os_wait_stats
  - Historical wait (The time SQL restarts)
  - Track cumulative and total

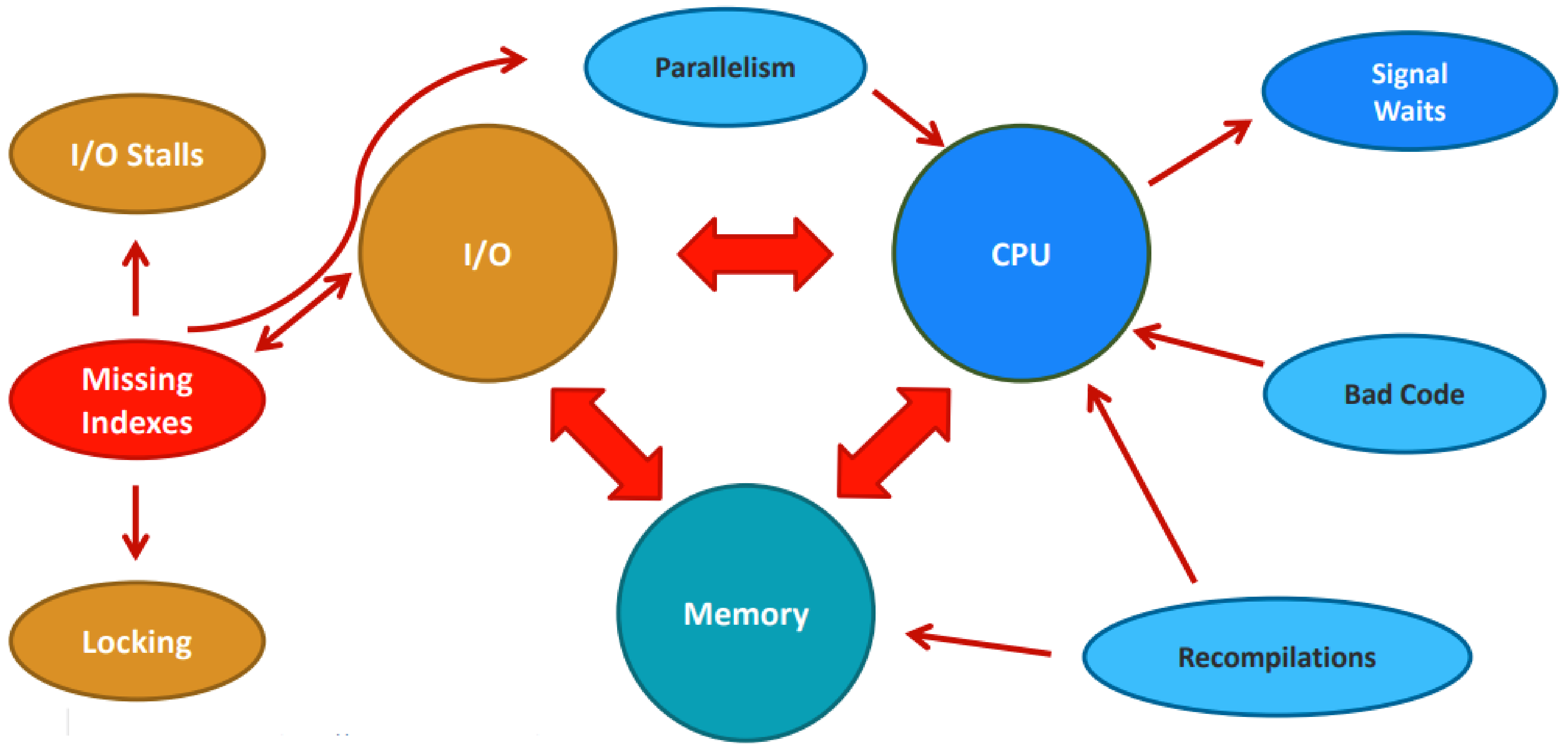
Common wait types

- CXPACKET
- SOS_SCHEDULER_YIELD
- THREADPOOL
- LCK_*
- PAGEIOLATCH_*: waits associated with delays in a thread being able to bring a data file page into cache from the physical data file
- WRITELOG: waits related to issues with writing to the log file
  - SQL Server Delayed durability to reduce the writelog
- **HADR_SYNC_COMMIT**
- **RESOURCE_SEMAPHORE**
- **RESOURCE_SEMAPHORE_QUERY_COMPILE**

Other DMVs

- sys.dm_exec_requests
- sys.dm_exec_session_wait_stats
- sys.dm_os_schedulers

Everything is Related



SQL Server I/O

Fundamentals and Troubleshooting

Asynchronous vs. Synchronous

- Synchronous- make I/O request and wait for completion
- Asynchronous – make I/O request and continue executing code. Check if I/O was completed by the OS and handle completion
  - Apparent performance benefit
  - More complex management (at code level)
  - Any thread can handle completion
- SQL Server uses Mostly Asynchronous I/O
  - Data read/write, Log Writes, Xevents, Backup
- Some functionality uses synchronous I/O
  - Errolog writes, SQL Trace
- Note: see call stacks in notes

Page Reads

- Reads occur in one or multiple of 8 KB pages
- Can be performed by any worker thread servicing queries
- SQL Server does mostly reads – driven by workload
- Use ReadFileScatter() to maximize throughput
- Read Ahead -64 contiguous pages at a time – 512 KB

Page Writes

- Writes typically occur in multiple of 8 KB pages
- Page is NOT immediately written to disk when modified in memory – optimize I/O
  - It is marked as Dirty
  - Multiple modifications can occur to a page in memory
  - Transaction Log record must be written to disk though (write-ahead logging)
- Is performed by System threads (not regular worker threads)
  - Checkpoint, lazy writer
- Much less frequent than reads – it occurs periodically

Troubleshooting SQL Server I/O

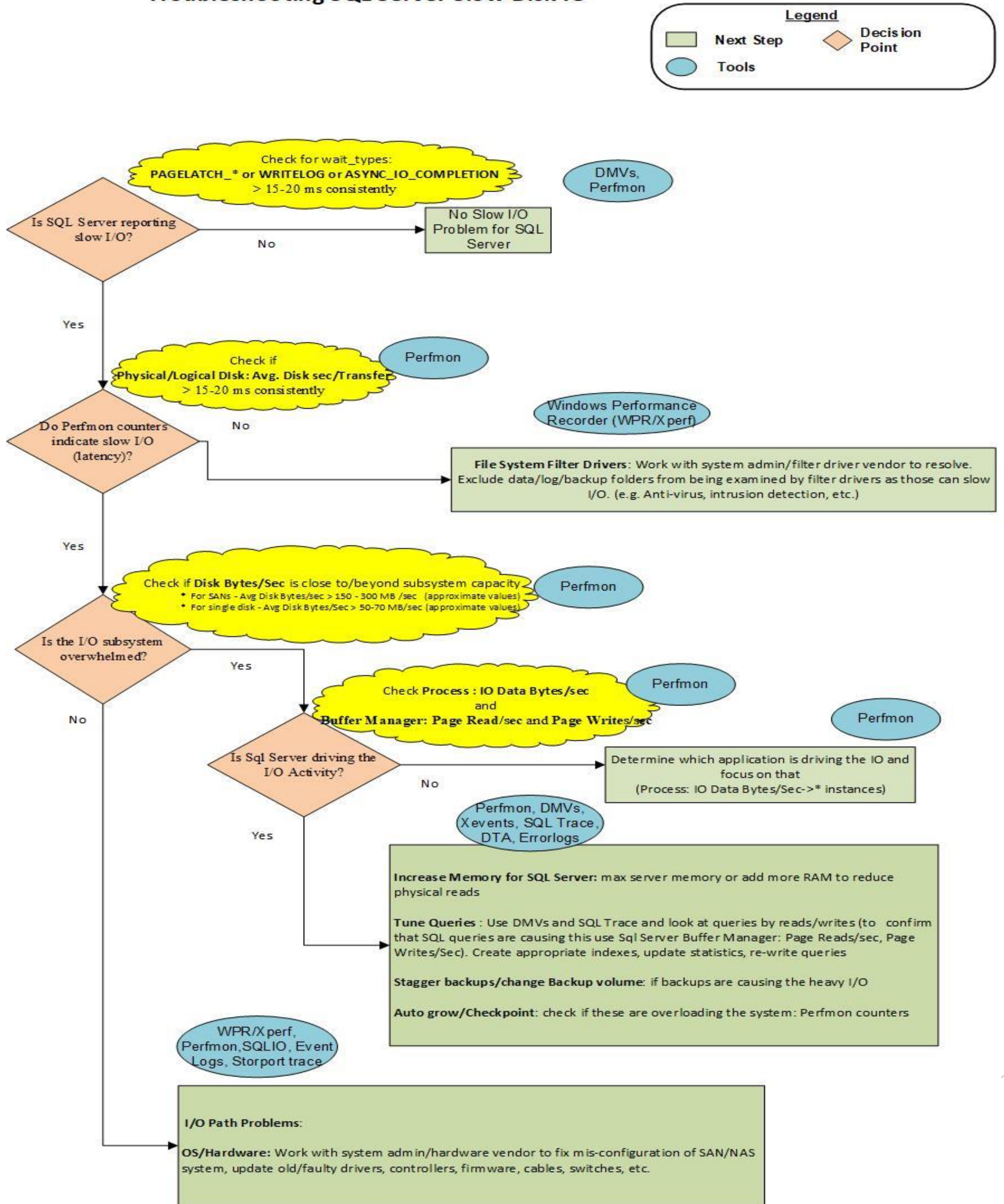
- Performance Monitor has almost all the answers
  - **SQL Server: Buffer Manager**
    - **Page Reads/Sec**: number of physical database page reads that are issued per second
    - **Page Writes/Sec**: number of physical database page writes that are issued per second
  - Physical or Logical Disk
    - **Avg Disk Sec/Read**
    - **Avg Disk Sec/Write**
    - **Disk Write Bytes/Sec**
    - **Disk Read Bytes/Sec**
  - **Process**
    - **I/O Data Bytes/Sec**

Troubleshooting SQL Server I/O

Avg disk sec/transfer

- Less than 10 ms - very good
- Between 10 - 20 ms - okay
- Between 20 - 50 ms - slow, needs attention
- Greater than 50 ms – Serious I/O bottleneck

Troubleshooting SQL Server Slow Disk IO


Legend
Next Step
Decision Point
Tools
Check for wait_types: PAGELATCH_* or WRITELOG or ASYNC_IO_COMPLETION > 15-20 ms consistently
DMVs, Perfmon
Is SQL Server reporting slow I/O?
No
No Slow I/O Problem for SQL Server
Yes
Check if Physical/Logical Disk: Avg. Disk sec/Transfer > 15-20 ms consistently
Perfmon
Do Perfmon counters indicate slow I/O (latency)?
No
Windows Performance Recorder (WPR/Xperf)
Yes
Check if Disk Bytes/Sec is close to/beyond subsystem capacity
* For SANs - Avg Disk Bytes/sec > 150 - 300 MB /sec (approximate values)
* For single disk - Avg Disk Bytes/Sec > 50-70 MB/sec (approximate values)
Perfmon
Is the I/O subsystem overwhelmed?
Yes
No
Check Process : IO Data Bytes/sec and Buffer Manager: Page Read/sec and Page Writes/sec
Perfmon
Perfmon
Is Sql Server driving the I/O Activity?
No
Yes
Perfmon, DMVs, Xevents, SQL Trace, DTA, Errorlogs
WPR/Xperf, Perfmon,SQLIO, Event Logs, Storport trace

File System Filter Drivers: Work with system admin/filter driver vendor to resolve. Exclude data/log/backup folders from being examined by filter drivers as those can slow I/O. (e.g. Anti-virus, intrusion detection, etc.)

Determine which application is driving the IO and focus on that (Process: IO Data Bytes/Sec->* instances)

Increase Memory for SQL Server: max server memory or add more RAM to reduce physical reads

Tune Queries : Use DMVs and SQL Trace and look at queries by reads/writes (to confirm that SQL queries are causing this use Sql Server Buffer Manager: Page Reads/sec, Page Writes/Sec). Create appropriate indexes, update statistics, re-write queries

Stagger backups/change Backup volume: if backups are causing the heavy I/O

Auto grow/Checkpoint: check if these are overloading the system: Perfmon counters

I/O Path Problems:

OS/Hardware: Work with system admin/hardware vendor to fix mis-configuration of SAN/NAS system, update old/faulty drivers, controllers, firmware, cables, switches, etc.

Troubleshooting Tools

- Performance Monitor
- sys.dm_io_pending_io_requests
  - look for **io_pending** column > 0
- sys.dm_exec_requests
  - look for I/O related **wait_type** values (see next slide)
- sys.dm_exec_query_stats
  - look for high values in **total_physical_reads**

OK, so SQL Server is slow.

Start by finding out the predominant waits. We should query both the cumulative wait information for the instance (sys.dm_os_wait_stats) and the per-query wait information (sys.dm_exec_requests/sys.dm_os_waiting_tasks/Xevent wait_completed or wait_info) Do you see these?

- PAGEIOLATCH_*
  - When a SQL Server is going to read a buffer page from disk, an exclusive latch on the buffer is acquired (_EX). This is necessary because of course the content of that page are about to be written to the buffer. This is immediately followed by an attempt to acquire a shared latch (_SH) which is of course blocked by the EX – this blocking is what will show in wait_stats as PAGEIOLATCH_SH.
- IO_COMPLETION
  - Non-data page IO operations.
- ASYNC_IO_COMPLETION
  - File growth
- BACKUPIO
  - Waiting for a backup-specific IO operation
- BACKUPBUFFER
  - This occurs when all of the buffers allocated to hold data waiting to be flushed to a backup are full. This is often accompanied by BACKUPIO which would indicate the buffers may be full due insufficient flush rate caused by IO latency.

Tempdb

TEMPDB

- What is it:
  - Global database used by everyone
  - Works in a round robin fashion
  - No durable (Recreate new tempdb files on every restart)
- Configure
  - High performance disks
  - Multiple files 1 per CPU upto 8 files
  - Proper auto growth
- Contention: Metadata (PFS/GAM)
  - Fixing: Trace flag 1117/1118
  - Table variables vs Temp tabes
    - Trace flag 2453 with compile (2019 default)
- SPILLS: Wrong estimates
- Alwayson: Readable secondaries

Tempdb space Usage

- Internal Objects
- Version Store
- User Objects

Internal Objects

- Intermediate runs of sort,hash joins and hash aggregate
- To store XML variables or other large object (LOB) data type variables
- By queries that need a spool to store intermediate results
- By Service Broker to store messages in transit
- By INSTEAD OF triggers to store data for internal processing

Version Store

- Version stores are used to store row versions generated by transactions for features such as snapshot isolation, triggers, MARS (multiple active result sets), and online index build

User Objects

- User objects include both user-defined tables and indexes, and system catalog tables and indexes.

Restrictions

- Adding filegroups.
- Backing up or restoring the database.
- Changing collation. The default collation is the server collation.
- Changing the database owner. tempdb is owned by dbo.
- Creating a database snapshot.
- Dropping the database.
- Dropping the guest user from the database.
- Enabling change data capture.
- Participating in database mirroring.
- Removing the primary filegroup, primary data file, or log file.
- Renaming the database or primary filegroup.
- Running DBCC CHECKALLOC.
- Running DBCC CHECKCATALOG.
- Setting the database to OFFLINE.
- Setting the database or primary filegroup to READ_ONLY.

Tempdb Size and Placement Recommendations

- Allow automatic growth for tempdb
- create one data file for each CPU on the server
  - Make each data file the same size; this allows for optimal proportional-fill performance.
- Put the tempdb database on a fast I/O subsystem.
- Put the tempdb database on disks that differ from those that are used by user databases

Tempdb Space Requirements

- Query
- Triggers
- Snapshot isolation and read committed snapshot (RCSI)
- MARS
- Online index creation
- Temporary tables, table variables, and table-valued functions
- DBCC CHECK
- LOB parameters
- Cursors
- Service Broker and event notification
- XML and LOB variable
- Query notifications
- Database mail
- Index creation
- User-defined functions

Space required by queries

Query operators

- Sort
  - The sort operator needs tempdb space to sort the full rowset
- Hash Match
  - This operator has two inputs—one to build the hash table and other to probe it
- Spool
  - This operator requires that the full input rowset be stored in tempdb.

Space required by features that use version store

- Snapshot isolation
- Read committed snapshot isolation (RCSI)
- Online index build
- Triggers
- MARS

Space required by other features

- Temporary tables, table variables, and table-valued functions all use space in tempdb
- LOB variables (including parameters) and XML variables all consume space in tempdb
- Service Broker uses about 1 MB per dialog.
- Index build has an option to sort in tempdb.
- Two kinds of server-side cursors require tempdb space: static and keyset

Monitoring tempdb

- Dmv’s

  sys.dm_db_file_space_used

  sys.dm_db_session_file_usage

  sys.dm_db_task_space_usage

  sys.dm_tran_version_store_space_usage

  sys.dm_tran_version_store

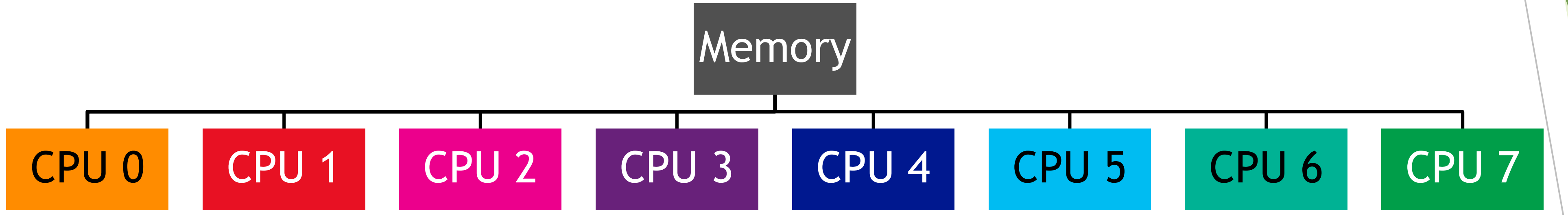
SQL Server Memory Architecture and Troubleshooting

Agenda

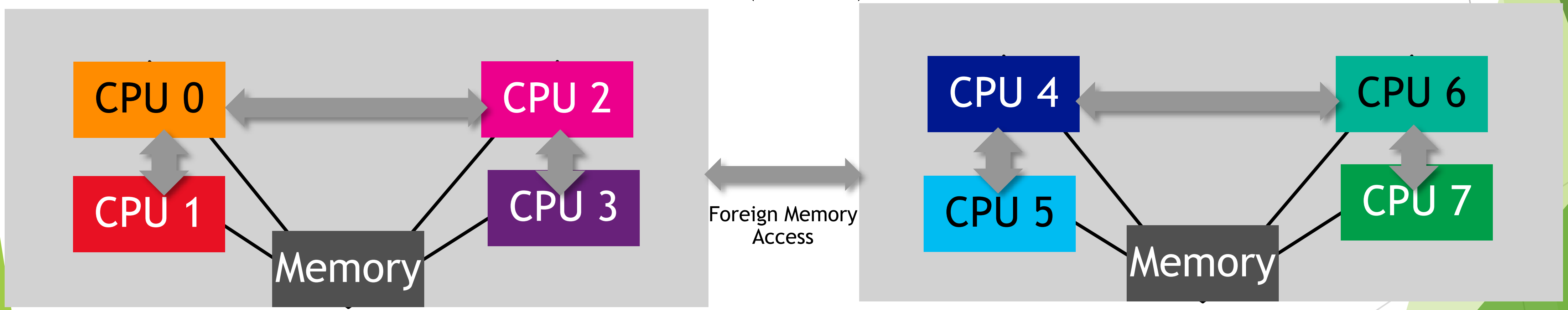
- NUMA and SQL server
- Memory Concepts and Architecture
  - Memory models
  - How SQL allocates memory
  - Memory Clerks
  - Buffer Pool
  - Memory outside the Bpool
  - How SQL frees memory
- Troubleshooting
  - Common Scenarios
  - Tools to troubleshoot (DMVs, Perfmon, DBCC Memorystatus, Error Logs)

NUMA Concepts

Symmetric Multi Processing
(SMP)

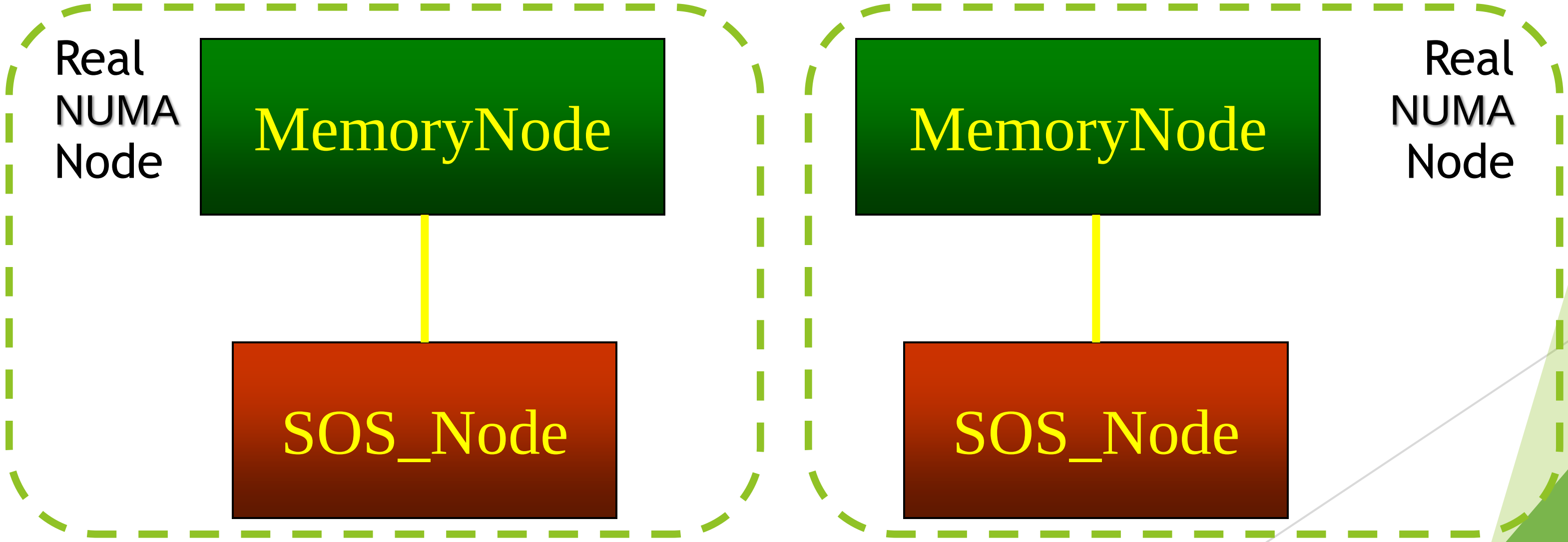


Non-Uniform Memory Access
(NUMA)

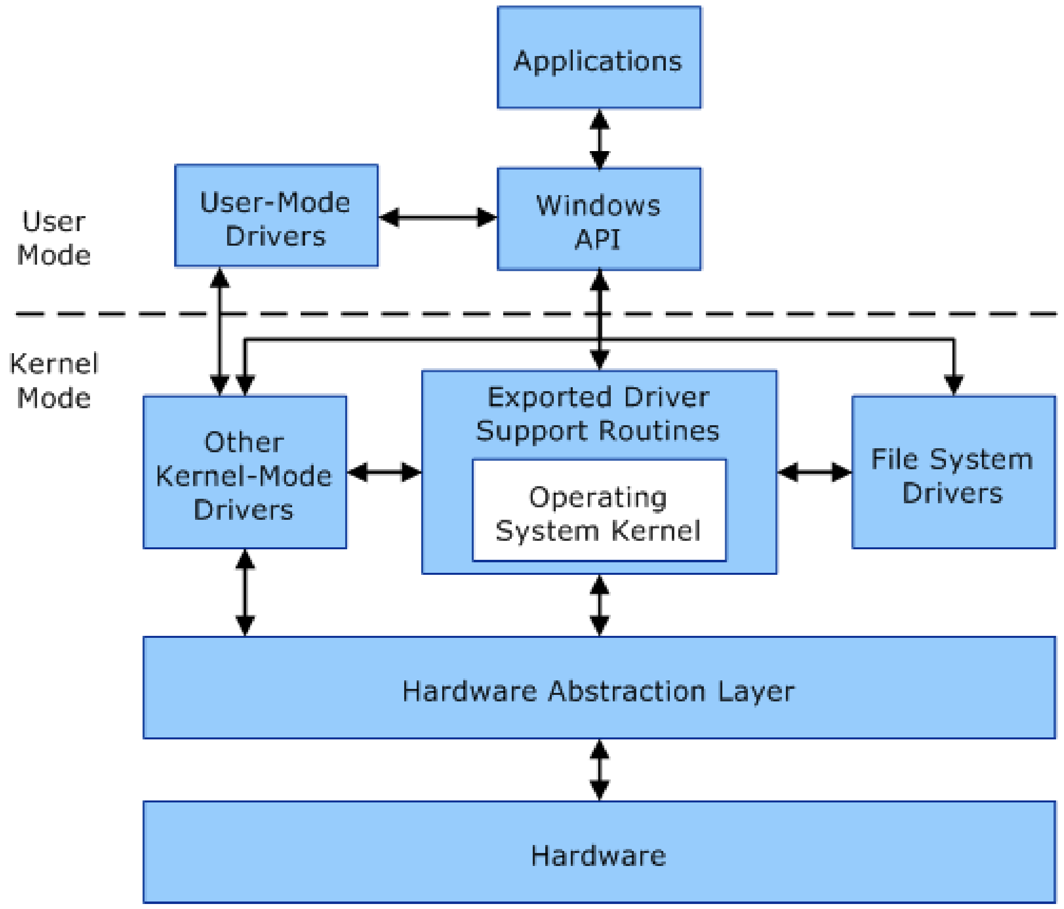


Real NUMA

- Memory Node per physical NUMA node
- SOS_Node for each Memory Node
- T8015 disables NUMA detection – treated as SMP

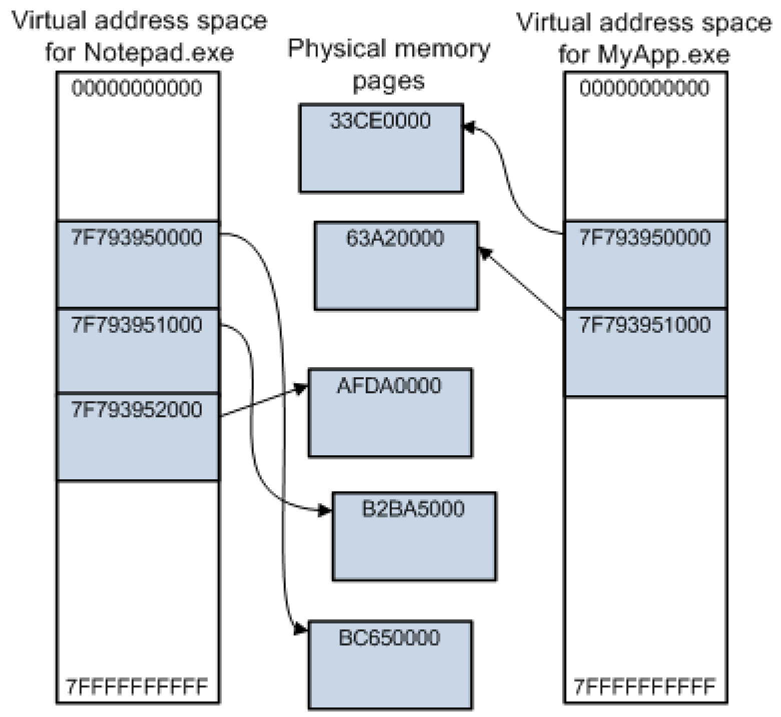


User mode and Kernel mode



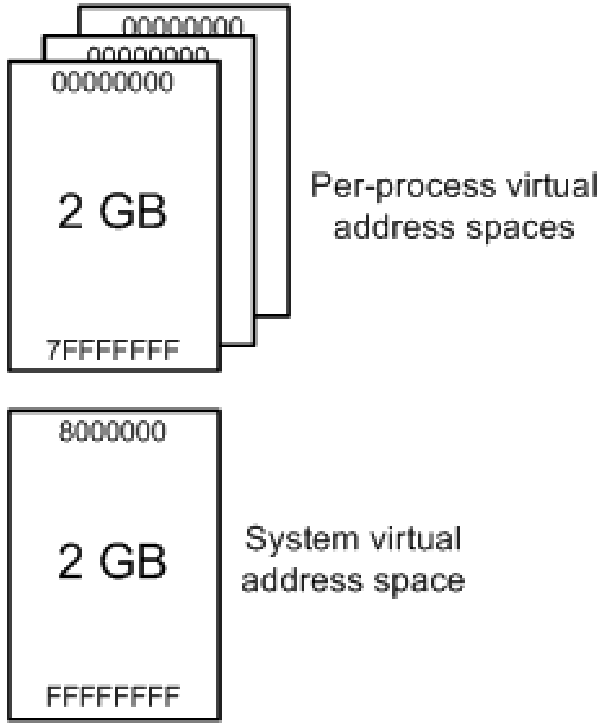
User mode and kernel mode - Windows drivers | Microsoft Docs

Virtual Address Space (VAS)



[Virtual address spaces - Windows drivers | Microsoft Docs]

VAS(User/System)


00000000
00000000
00000000
2 GB
7FFFFFFF
Per-process virtual address spaces
8000000
2 GB
FFFFFFFF
System virtual address space

Working Set

- The working set of a process is the set of pages in the virtual address space of the process that are currently resident in physical memory.
- The working set contains only pageable memory allocations; nonpageable memory allocations such as Address Windowing Extensions (AWE) or large page allocations are not included in the working set.
- When a process references pageable memory that is not currently in its working set, a *page fault* occurs.
- The system page fault handler attempts to resolve the page fault and, if it succeeds, the page is added to the working set. (Accessing AWE or large page allocations never causes a page fault, because these allocations are not pageable

Pages can be removed from a process working set as a result of the following actions:

- The process reduces or empties the working set by calling the **SetProcessWorkingSetSize**, **SetProcessWorkingSetSizeEx** or **EmptyWorkingSet** functio n.
- The memory manager trims pages from the working set to create more available memory.
- The memory manager must remove a page from the working set to make room for a new page (for example, because the working set is at its maximum size).

Memory Models

Conventional

LOCKED PAGES

LARGE PAGES

How SQL Server Allocates Memory

- Any user-mode process in Windows allocates virtual memory – VirtualAlloc() API
  - Reserve – just set aside memory addresses in process
  - Commit – actually allocate the memory in RAM
  - Free – release memory
- In 64-bit Windows, virtual address limit is 8 TB and starting from 2012 R2 it’s 128 TB
- SQL Server
  - <u>Reserves memory</u>
  - <u>Commits</u> memory AS NEEDED
  - <u>Frees</u> memory only under memory pressure
- Uses AWE API (Allocate User Physical Pages) if Locked Pages in Memory set

How SQL Server Allocates Memory

- SQL engine will *not* acquire all requested memory upon startup
  - Grows buffer pool gradually as it needs the memory
- If another application on the server grabs memory first, SQL may not be able to obtain all the memory you have configured for it to use
- If large page memory model enabled, SQL will reserve and commit memory upon startup

Memory Clerks

- A memory clerk is an accountant of memory
  - Keep track of where and how much memory was allocated
- SQL Server contains four types of memory clerks
  - Generic
  - Cache store
  - User store
  - Object store.
- Memory clerks also process notifications from Resource Monitor and directs it to the appropriate destination (e.g. Buffer Pool, Cache Stores, CLR, Lock Manager, etc.).

How SQL Server Frees Memory

- SQL Server will free memory under external or internal memory pressure
  - External – OS notifies all applications of pressure
  - Internal – SQL Bpool is exhausted, and memory is needed
- Memory is freed immediately after usage
  - Optimizer memory
  - Memory grants – sort and hash operations
  - Lock memory
- Resource Monitor thread (RM)
  - RM monitors state of the external and internal memory indicators.
  - Once one of them changes, it broadcasts a notification requesting consumers to cut back/shrink.
  - Each Memory Clerk responds by freeing as much as it can to stay productive
  - Common first target – procedure cache
- Lazy Writer thread
  - Responsible for freeing Bpool data/index pages only

Troubleshooting Memory Issues

Common Scenarios

1. Out of Memory Errors – internal pressure
   - No memory available - 701 errors
   - Waiting for Compile or Sort/Hash Memory
2. Working Set Trimming/SQL is paged out – external memory pressure
3. Low memory for data/index pages

No Memory to Run Query -701

- Error 701 is most common OOM indicator
  - 701 - There is insufficient system memory to run this query.
  - 701 - There is insufficient system memory in resource pool 'default' to run this query.
- Typically indicates that Bpool has been exhausted
- Have to find the culprit consumer(s)
  - DBCC MEMORYSTATUS
  - Sys.dm_os_memory_clerks
- Common culprits –
  - Large procedure cache
    - CACHESTORE_SQLCP
    - CACHESTORE_OBJCP
    - Large sort/hash operations - MEMORYCLERK_SQLQUERYEXEC, MEMORYCLERK_SQLQERESERVATIONS

No Memory to Run Query -701

  - Large Compilation consumers -MEMORYCLERK_SQLOPTIMIZER
  - Memory pressure from outside has reduced SQL usage
  - Sp_configure max server memory is low
- Once you locate the large consumer take action to prevent
  - For large procedure cache – parameterize queries
  - For large sort/hash operations – tune query, re-write, use indexes, etc.
  - For large compilation – tune query, re-write, simplify
- For external pressure – check what other applications are using the system and set max server memory accordingly
- Add more RAM on system

Waiting for Compile or Sort/Hash Memory

- Two types of memory allocations where threads will wait before they timeout
  - Memory for compilation
  - Memory grant (sort and hash operations)
- Resource_Semaphore* Wait Types
  - RESOURCE_SEMAPHORE_QUERY_COMPILE - compilation
  - RESOURCE_SEMAPHORE – sort and hash memory
- Eventually, the waits will time out
  - 8628 - A time out occurred while waiting to optimize the query. Rerun the query.
  - 8645 - A time out occurred while waiting for memory resources to execute the query. Rerun the query.

Waiting for Compile or Sort/Hash Memory … cont.

- Examine DBCC MEMORYSTATUS
  - Sort and hash - MEMORYCLERK_SQLQUERYEXEC, MEMORYCLERK_SQLQERESERVATIONS
  - Compilation - MEMORYCLERK_SQLOPTIMIZER, Gateways
- Ultimately have to identify query that is causing the issue and tune it
  - Identify using SQL trace or DMVs
- Could be caused by another consumer so identify largest consumer
  - DBCC MEMORYSTATUS
  - Sys.dm_os_memory_clerks

Working Set Trimming

- SQL Server memory could be paged out by Windows
  - Due to low memory on system
  - Due to a kernel driver making an incorrect memory allocation
- Errorlog/Event log may contain:
  - A significant part of sql server process memory has been paged out. This may result in a performance degradation.
- SQL Server will be very slow or unresponsive

Working Set Trimming

- Common Culprits
  - SQL Server Max Server Memory is not set appropriately causing overconsumption of RAM
  - Other applications on system consume unexpected amounts of memory – again ‘max server memory’
  - A kernel driver calling a memory allocation API can trigger
- Use Perfmon counters to identify
  - Working set of all process reduced significantly
  - Spot a new application’s working set jumping sharply
- Kernel driver issues or Vmware
- Setting Locked Pages in Memory keeps Bpool from being paged out – can help

Low Memory for Data Cache

- Bpool is primarily a data cache
- If lots of memory is stolen, then insufficient resources to cache data/index pages
- This will cause slower performance because data pages need to be read from disk
- Harder to spot
- Indicators Buffer Manager object
    - Buffer Cache Hit Ratio < 90%
    - Page Life Expectancy < 300 sec
    - Very high Page Reads/Sec

Low Memory for Data Cache

- Reduce Stolen Memory
- Identify largest consumers via DBCC MEMORYSTATUS and resolve
  - For large procedure cache – parameterize queries
  - For large sort/hash operations – tune query, re-write, use indexes, etc.
  - For large compilation – tune query, re-write, simplify
- For external pressure – check what other applications are using the system and set max server memory accordingly
- Add more RAM – may be a valid step here
  - Determined by Database sizes

Locking, Blocking and Deadlock

Lock Resources

Resource	Description
RID	A row identifier used to lock a single row within a heap.
KEY	A row lock within an index used to protect key ranges in serializable transactions.
PAGE	An 8-kilobyte (KB) page in a database, such as data or index pages.
EXTENT	A contiguous group of eight pages, such as data or index pages.
HoBT	A heap or B-tree. A lock protecting a B-tree (index) or the heap data pages in a table that does not have a clustered index.
TABLE	The entire table, including all data and indexes.
FILE	A database file.
APPLICATION	An application-specified resource.
METADATA	Metadata locks.
ALLOCATION_UNIT	An allocation unit.
DATABASE	The entire database.

KEY

- KEY is a row lock taken on a table that has clustered index and/or a row lock taken on a non-clustered index.
- Can be retrieved using *%%lockres%%* function.
- KEY-RANGE lock means a range of rows have been locked depending on the predicate.
  - Implemented in serializable isolation level to avoid phantoms.
  - There are nine types of KEY-RANGE locks.

KEY-RANGE

- KEY-RANGE lock is another type of KEY lock.
- A range of rows have been locked depending on the predicate.
- KEY-RANGE locks are implemented in serializable isolation level to avoid phantoms.
- There are nine types of KEY-RANGE locks:
  - RangeS-S
  - RangeS-U
  - RangeIn-Null
  - RangeX-X
  - RangeIn-S
  - RangeIn-U

KEY-RANGE (Continued)

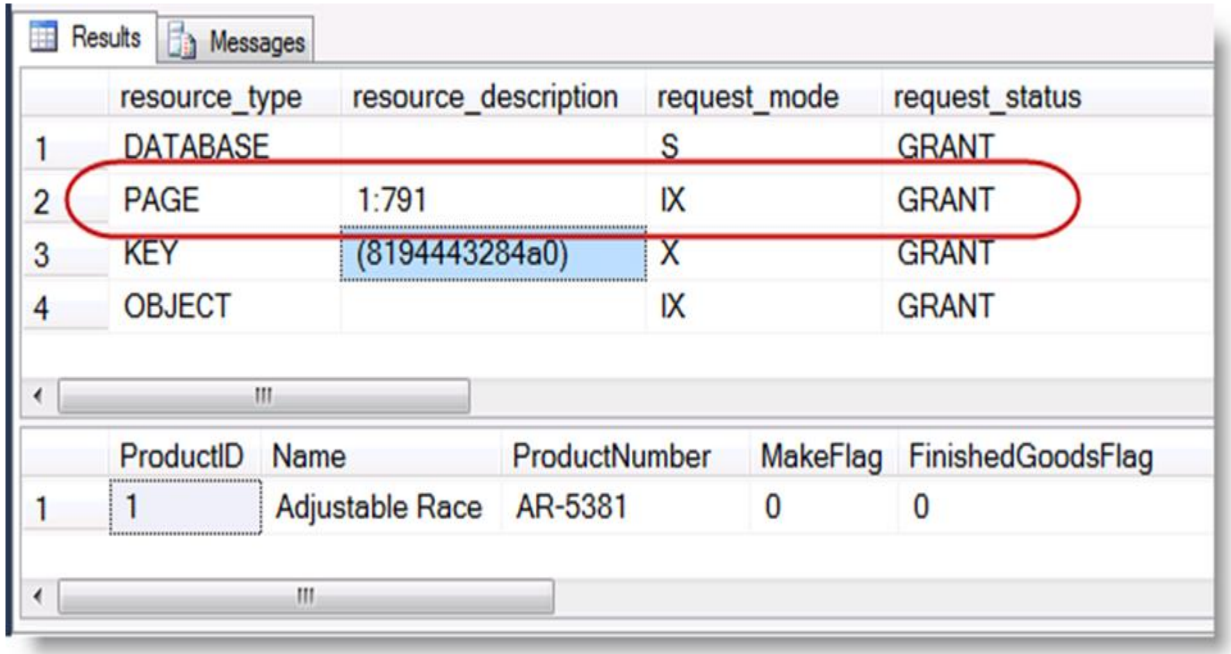
- RangeIn-X
- RangeX-S
- RangeX-U

RID

- RID is a row lock on a table, which is a heap.
- The RID format is *FILE:PAGE:SLOTID*.
- The resource can be retrieved using *%%lockres%%* function.

PAGE

- PAGE is a lock on an index page or a data page.
- The format is *FILE ID: PAGE NUMBER*.



	resource_type	resource_description	request_mode	request_status
1	DATABASE		S	GRANT
2	PAGE	1:791	IX	GRANT
3	KEY	(8194443284a0)	X	GRANT
4	OBJECT		IX	GRANT

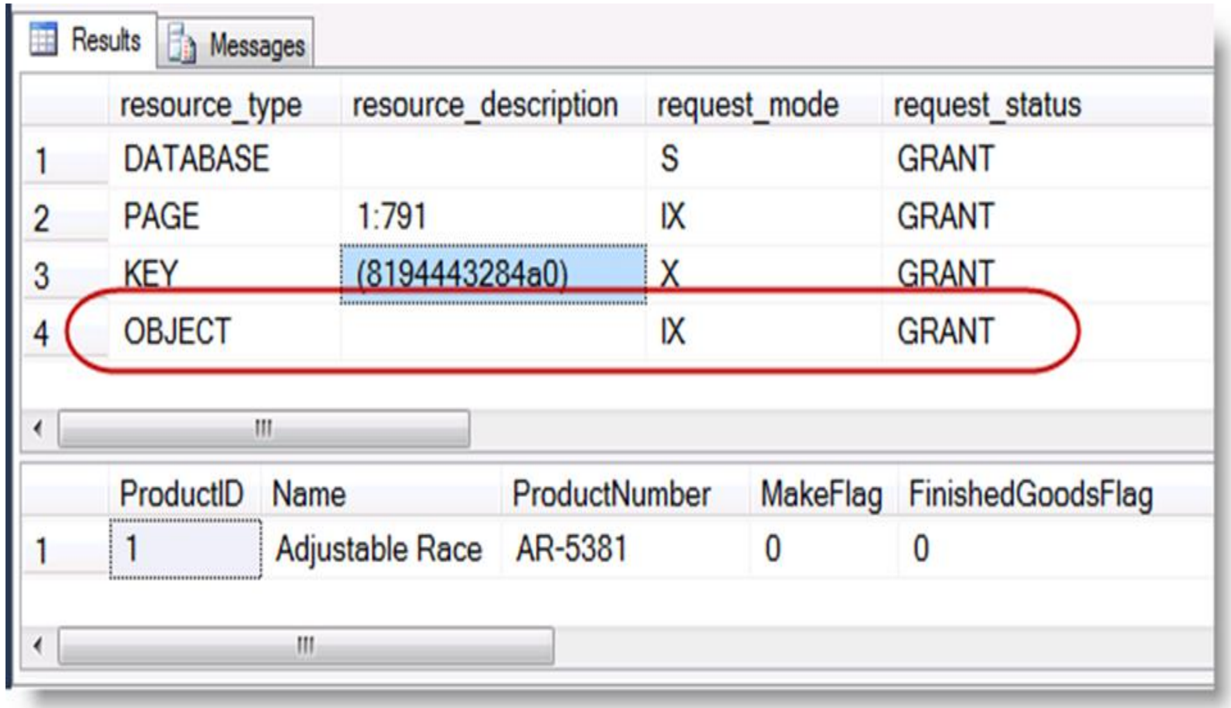
	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag
1	1	Adjustable Race	AR-5381	0	0

EXTENT

- EXTENT is a contiguous set of eight pages.
- The format is *FILE ID: FIRST PAGE NUMBER*.
- Pages in EXTENTS are allocated to Tables.

OBJECT

- Generally, OBJECT is a table lock, but could be anything with an OBJECT_ID.
- If the table is locked, all the data pages and associated indexes will be locked.

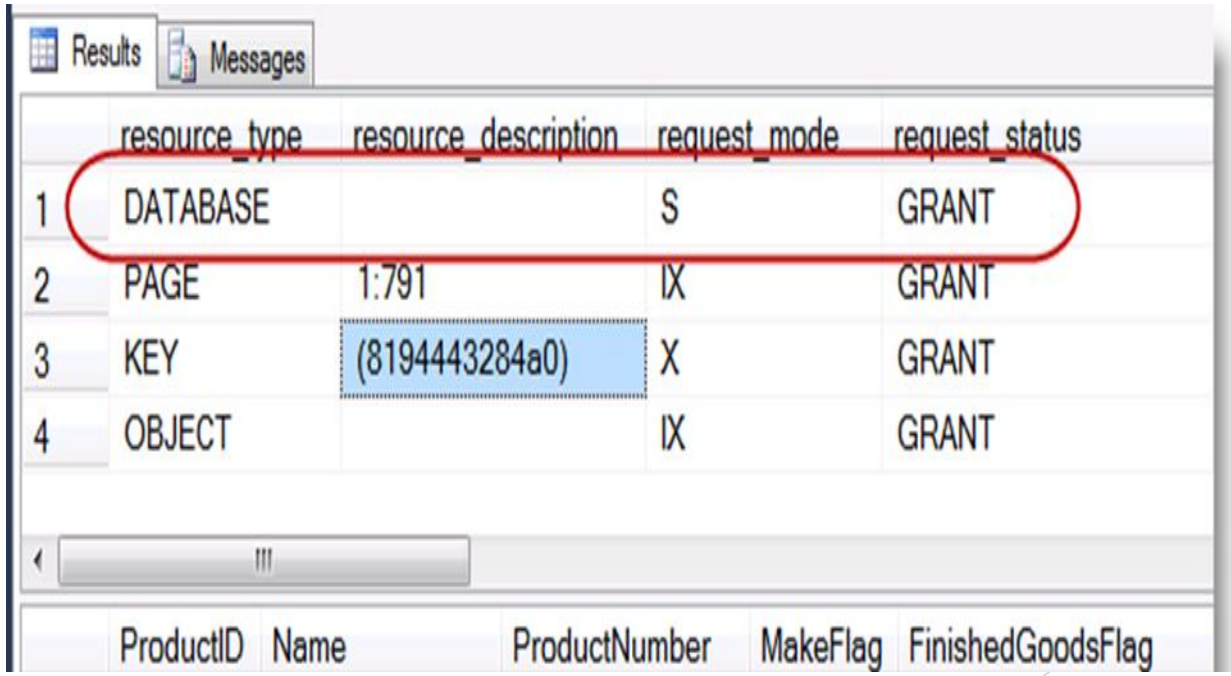


	resource_type	resource_description	request_mode	request_status
1	DATABASE		S	GRANT
2	PAGE	1:791	IX	GRANT
3	KEY	(8194443284a0)	X	GRANT
4	OBJECT		IX	GRANT

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag
1	1	Adjustable Race	AR-5381	0	0

DATABASE

- DATABASE lock is a shared lock denoting that the process is using the database.
- SQL Server checks for these locks when it needs exclusive lock on the database to restore, alter, drop, or close the database.
- If a process has a shared lock on the database, SQL Server’s attempt to exclusively lock the database is blocked.



	resource_type	resource_description	request_mode	request_status
1	DATABASE		S	GRANT
2	PAGE	1:791	IX	GRANT
3	KEY	(8194443284a0)	X	GRANT
4	OBJECT		IX	GRANT

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag

Summary

- Describe lock resources.
- Describe the various types of locks including:
  - KEY locks
  - KEY-RANGE locks
  - RID locks
  - PAGE locks
  - EXTENT locks
  - OBJECT locks
  - HoBT locks
  - DATABASE locks

Lock Compatibility

- Lock compatibility controls whether multiple transactions can acquire locks on the same resource at the same time.
- If a resource is already locked by another transaction, a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock.
- If the mode of the requested lock is not compatible with the existing lock, the transaction requesting the new lock waits for the existing lock to be released or for the lock timeout interval to expire.
- Example: No lock modes are compatible with exclusive locks.

Lock Compatibility

Existing granted mode	IS	S	U	IX	SIX	X
Requested mode						
Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No	No
Update (U)	Yes	Yes	No	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No	No
Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
Exclusive (X)	No	No	No	No	No	No

Lock Escalation

- Lock Escalation is the process of converting many fine-grain locks into fewer coarse-grain locks reducing the system overhead and increasing the probability of concurrency contention.
- Does not escalate row or key-range locks to page locks, but escalates them directly to table locks.
- In SQL Server 2008, locking of partitioned tables can escalate to the HoBT level for the associated partition instead of to the table lock.

Lock Escalation Thresholds

- Lock escalation is triggered when it is not disabled on the table by using the ALTER TABLE SET LOCK_ESCALATION option, and when either of the following conditions exist:
  - A single Transact-SQL (T-SQL) statement acquires at least 5,000 locks on a single non-partitioned table or index.
  - A single T-SQL statement acquires at least 5,000 locks on a single partition of a partitioned table and the ALTER TABLE SET LOCK_ESCALATION option is set to **AUTO**.
  - The number of locks in an instance of the Database Engine exceeds memory or configuration thresholds.

Disabling Lock Escalation

- For partitioned tables, use the LOCK_ESCALATION option of ALTER TABLE to escalate locks to the HoBT level instead of the table or to disable lock escalation.
- You can also use trace flags 1211 and 1224 to disable all or some lock escalations.

Identifying Blocking

This lesson explains the various techniques to identify blocking.

Overview

- SQL Server provides several ways to identify locking:
  - Locks Event Category
  - SQL Server, Locks Object
  - Activity Monitor
  - DMVs that can give you comprehensive locking information includes:
    - sys.dm_exec_requests
    - sys.dm_tran_locks
    - sys.dm_os_waiting_tasks

sys.dm_tran_locks

- Returns the information about each request that is executing within the SQL Server.
- Contains column blocking_session_id.
- Filter on blocking_session_id > 0.

sys.dm_exec_requests

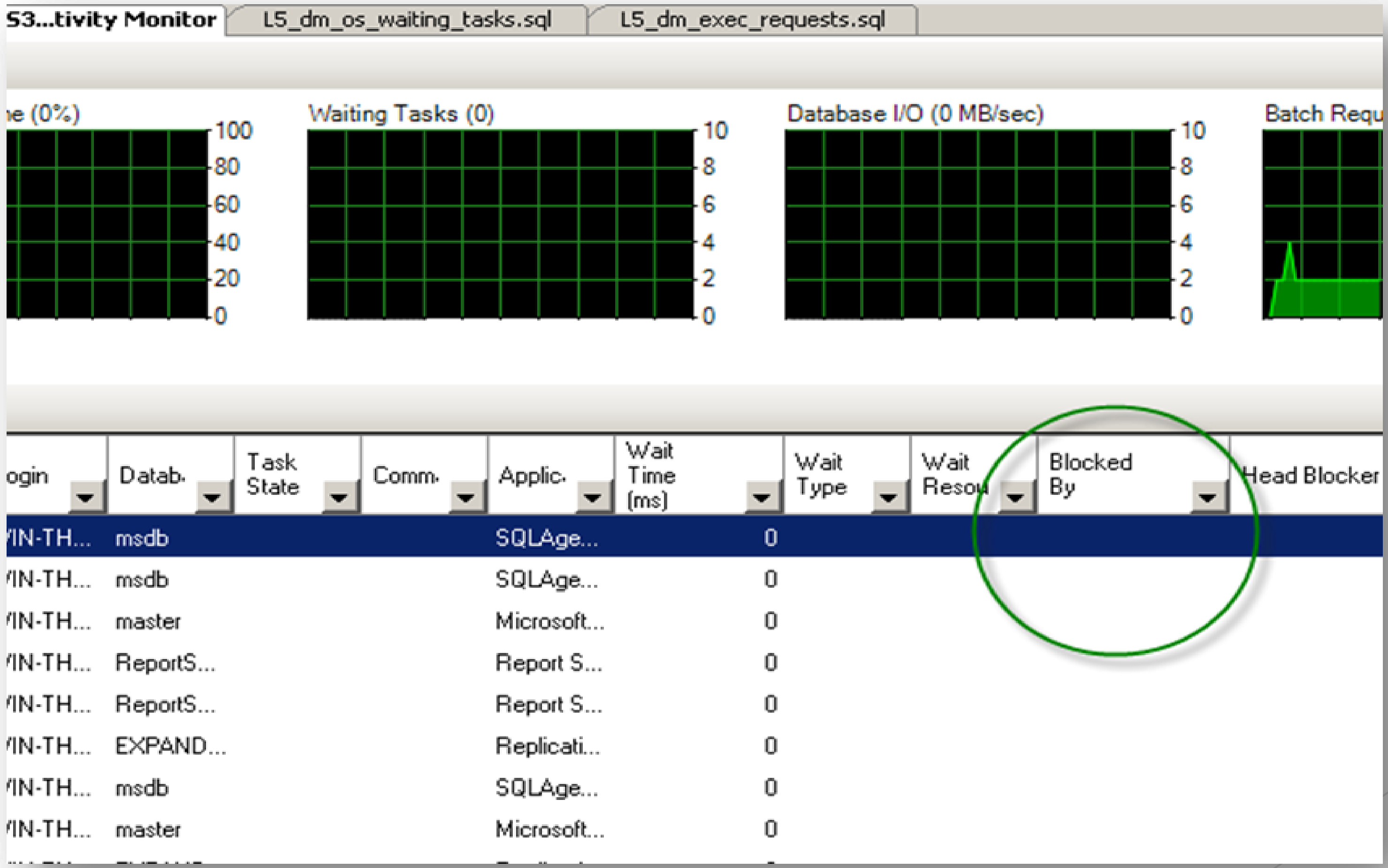
- Returns the information about each request that is executing within the SQL Server.
- Contains column blocking_session_id.
- Filter on blocking_session_id > 0.

sys.dm_os_waiting_tasks

- sys.dm_os_waiting_tasks is the waiter list.
- Contains currently suspended sessions and reasons for the suspension.
- If the wait is due to blocking, the blocker and blocked resource are shown in the columns blocking_session_id and resource.

Activity Monitor

- Process/Activity User Task Pane contains **Blocked By** column (if there are blocking sessions, the ID of the session that is blocking the task).



ogin	Datab.	Task State	Comm.	Applic.	Wait Time (ms)	Wait Type	Wait Resou	Blocked By	Head Blocker
IN-TH...	msdb			SQLAge...	0				
IN-TH...	msdb			SQLAge...	0				
IN-TH...	master			Microsoft...	0				
IN-TH...	ReportS...			Report S...	0				
IN-TH...	ReportS...			Report S...	0				
IN-TH...	EXPAND...			Replicati...	0				
IN-TH...	msdb			SQLAge...	0				
IN-TH...	master			Microsoft...	0				

Common Causes of Blocking

This lesson explains some common causes of blocking.

Overview

- Blocking happens when one connection from an application holds a lock and a second connection requires a conflicting lock type.
- This forces the second connection to wait, blocked on the first. One connection can block another connection, regardless of whether they emanate from the same application or separate applications on different client computers.
- Most blocking problems happen because a single process holds the locks for an extended period of time, causing a chain of blocked processes, all waiting on other processes for locks.

Common Blocking Scenarios

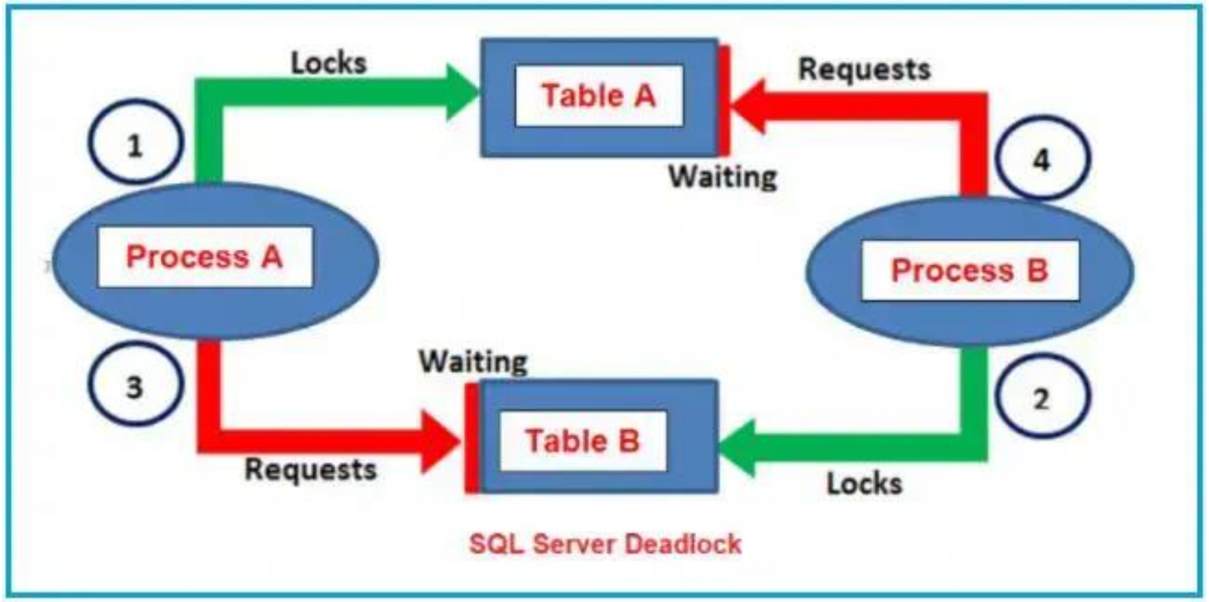
- Submitting queries with long execution time.
- Canceling queries that are not committed or rolled back.
- Applications that are not processing all the results to completion.
- Distributed client/server deadlocks.

Guidelines for Designing Applications to Avoid Blocking

- Do not use or design an application that allows users to fill in edit boxes that generate a long-running query.
- Do not use or design an application that allows user input within a transaction.
- Allow query cancellation.
- Use a query or Lock Time-out to prevent a runaway query and avoid distributed deadlocks.
- Immediately fetch all result rows to completion.
- Keep transactions as short as possible.
- Explicitly control connection management.
- Stress test the application at the full projected concurrent user load.

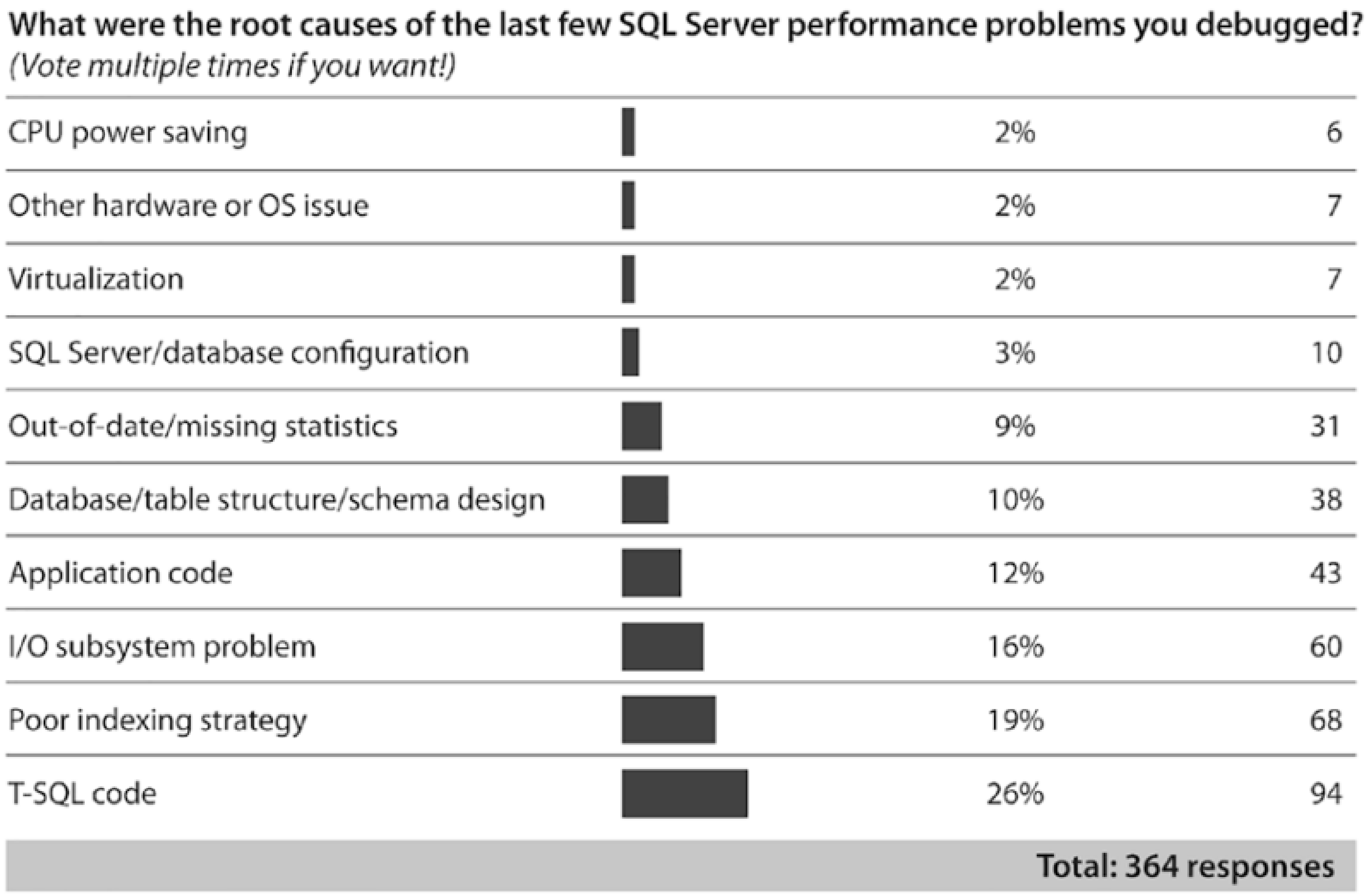
Deadlocks

- Transaction A acquires a shared lock on row 1.
- Transaction B acquires a shared lock on row 2.
- Transaction A now requests an exclusive lock on row 2, and is blocked until transaction B finishes and releases the shared lock it has on row 2.
- Transaction B now requests an exclusive lock on row 1, and is blocked until transaction A finishes and releases the shared lock it has on row 1.



Query Performance

Root causes of poor performance



What were the root causes of the last few SQL Server performance problems you debugged?
(Vote multiple times if you want!)

CPU power saving	▮	2%	6
Other hardware or OS issue	▮	2%	7
Virtualization	▮	2%	7
SQL Server/database configuration	▮	3%	10
Out-of-date/missing statistics	▮	9%	31
Database/table structure/schema design	▮	10%	38
Application code	▮	12%	43
I/O subsystem problem	▮	16%	60
Poor indexing strategy	▮	19%	68
T-SQL code	▮	26%	94

Total: 364 responses

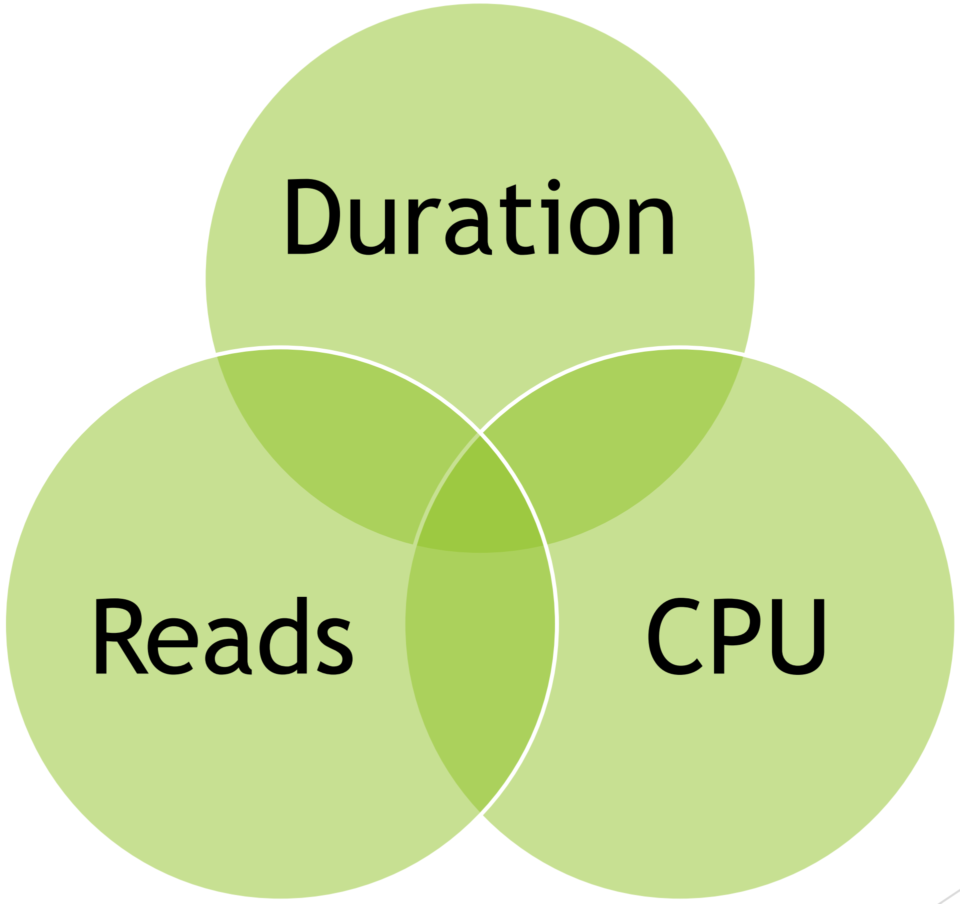
Common Performance Issues

- Insufficient or poor indexes
- Inaccurate or missing statistics
- Bad T-SQL • Problematic execution plans
- Excessive blocking
- Deadlocks
- Implicit convertion
- Incorrect database design
- Poor execution plan reuse
- Frequent recompilation of queries
- Network issue between SQL and the application

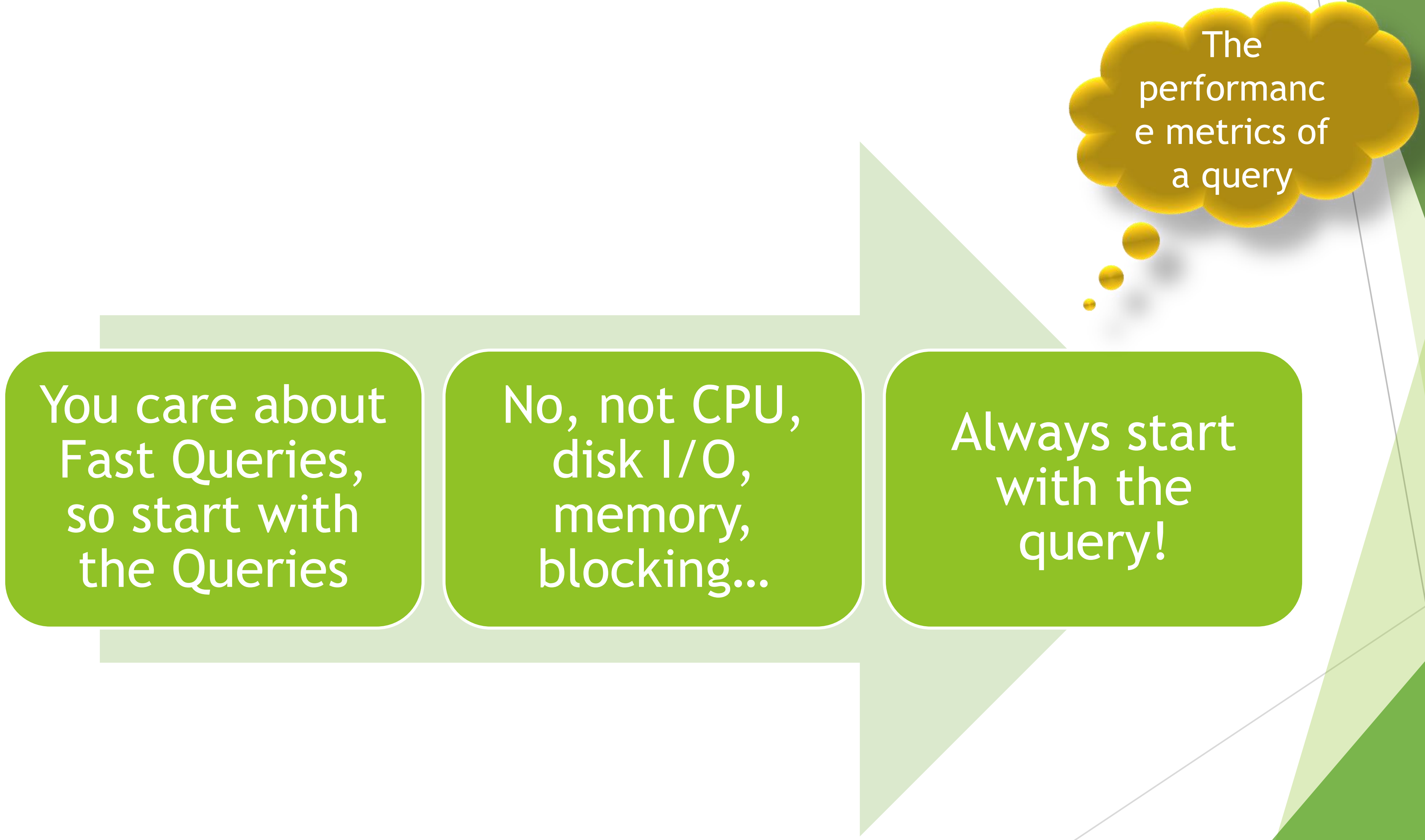
Methods for Capturing Query Performance Metrics

- Using Dynamic Management Views
- Capturing detailed metrics using Extended Events/profiler
- Query performance metrics in the Query Store
- SET STATISTICS TIME/IO
- Activity monitor
- Time Statistics in the Execution Plan

SQL Server Performance



SQL Server Performance: Where do I start?



You care about Fast Queries, so start with the Queries

No, not CPU, disk I/O, memory, blocking…

Always start with the query!

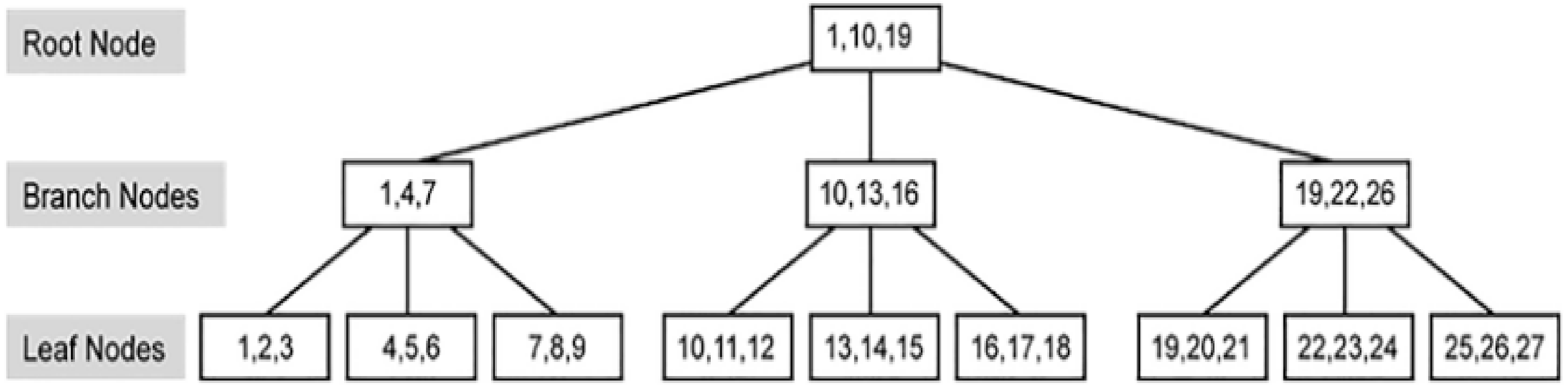
The performance metrics of a query

Query Performance Metrics: CPU, Duration, Reads

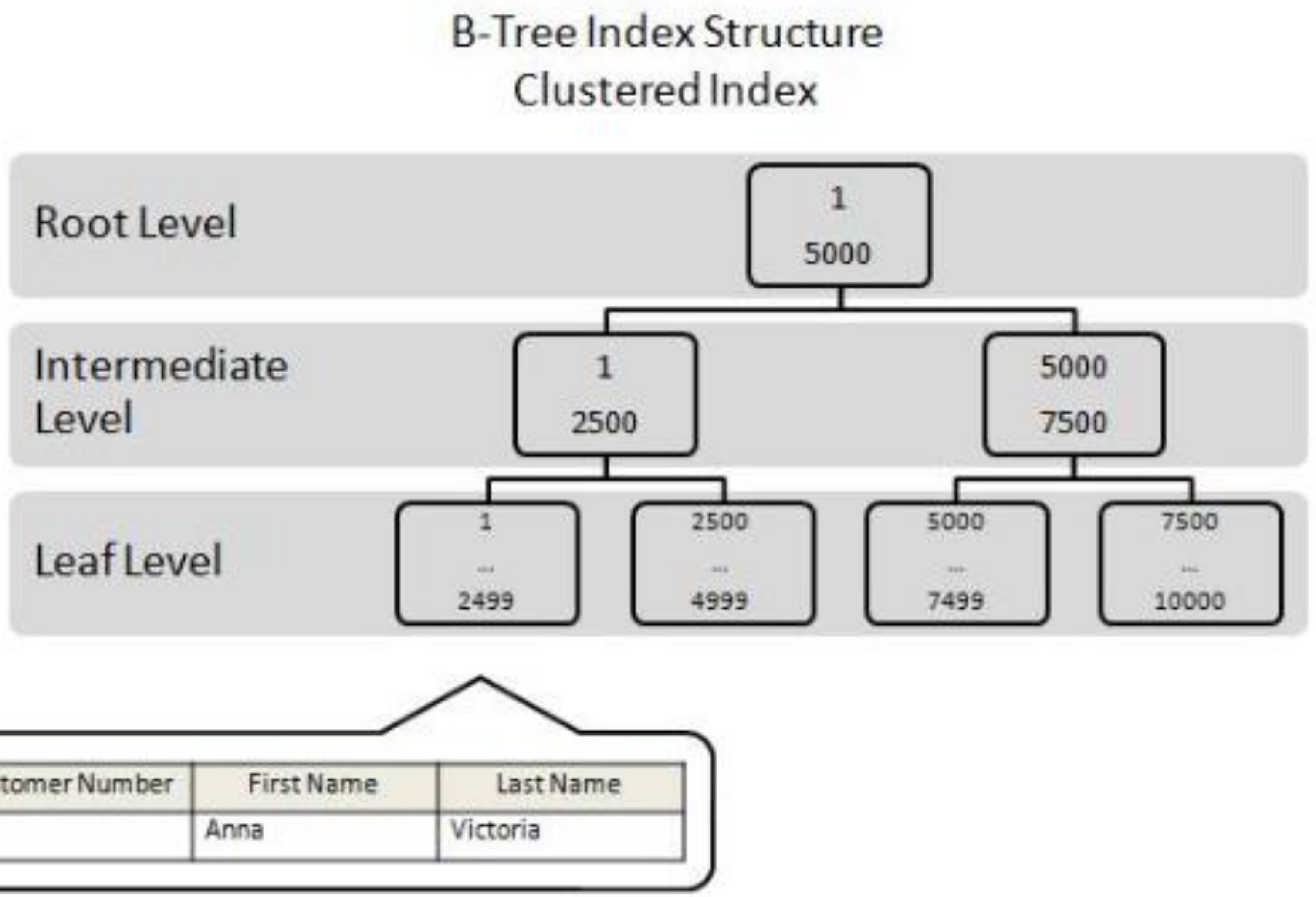
- CPU (worker time) – the time spent by a worker thread executing a query on the CPU
- Duration (elapsed time) – overall time a worker thread executed a query
  - Includes: running on the CPU and waiting for resources
  - **Duration - CPU = Wait Time!**
- Reads (logical reads) – number of 8-KB data/index pages read by a query

Indexes in SQL Server

B-Tree Strucure

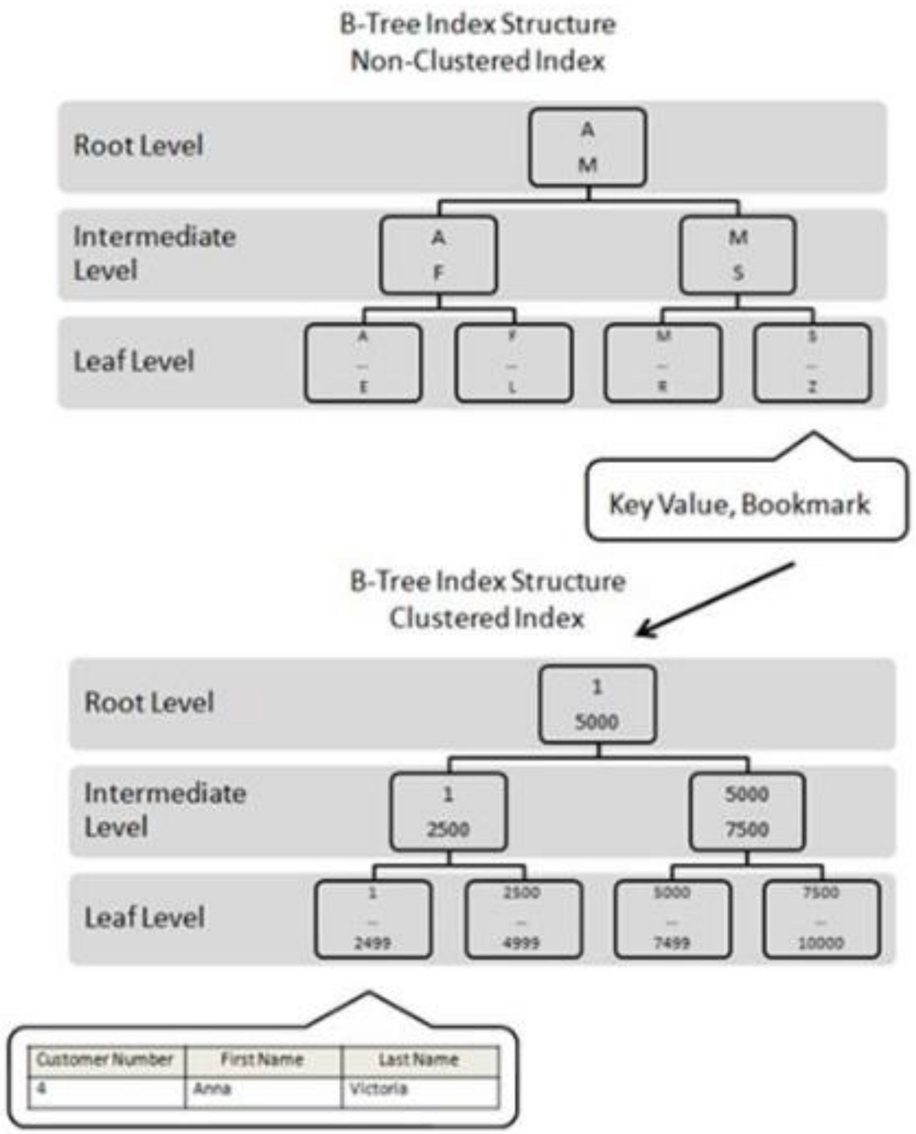


Clustered index



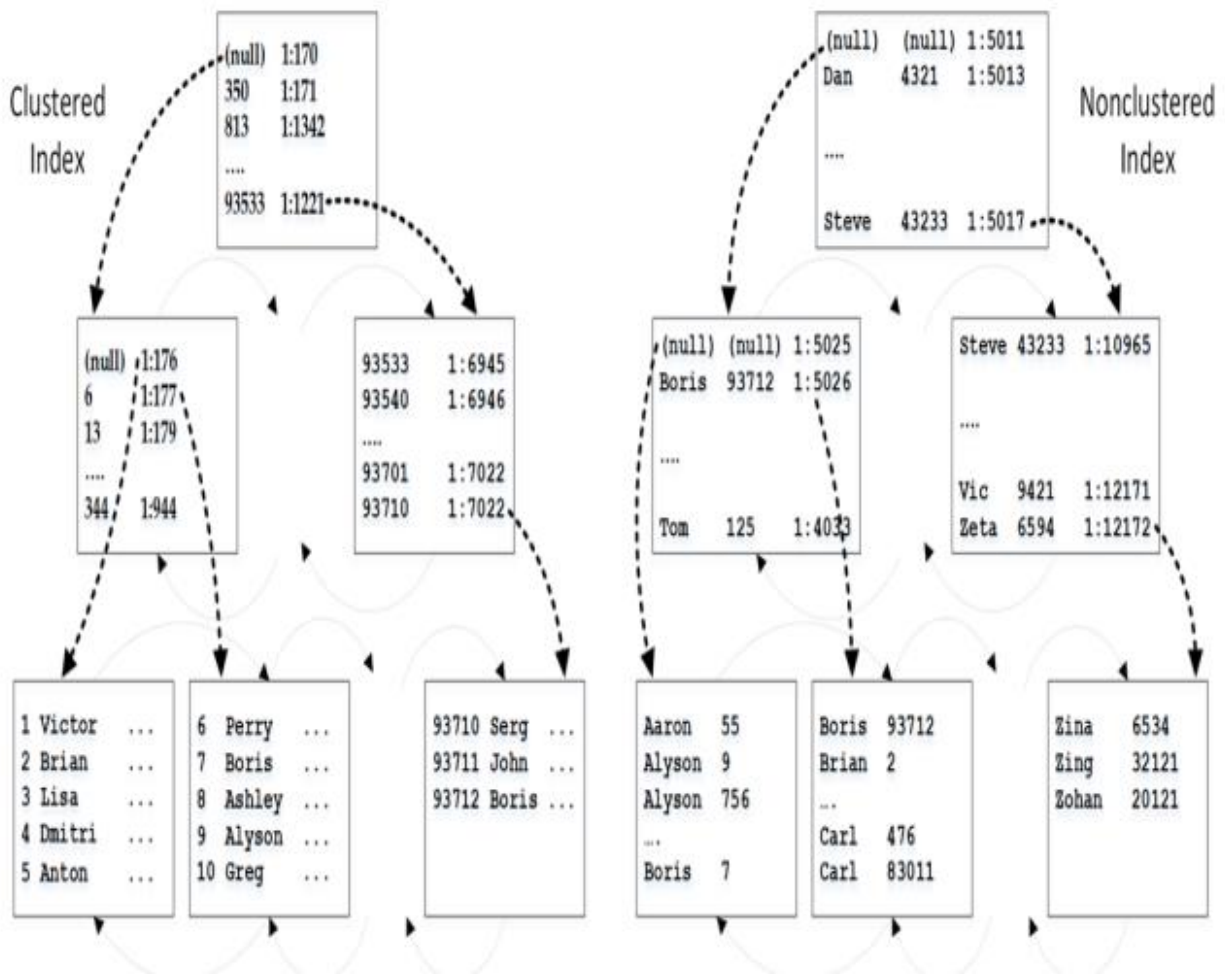
Customer Number	First Name	Last Name
4	Anna	Victoria

Non clustered index

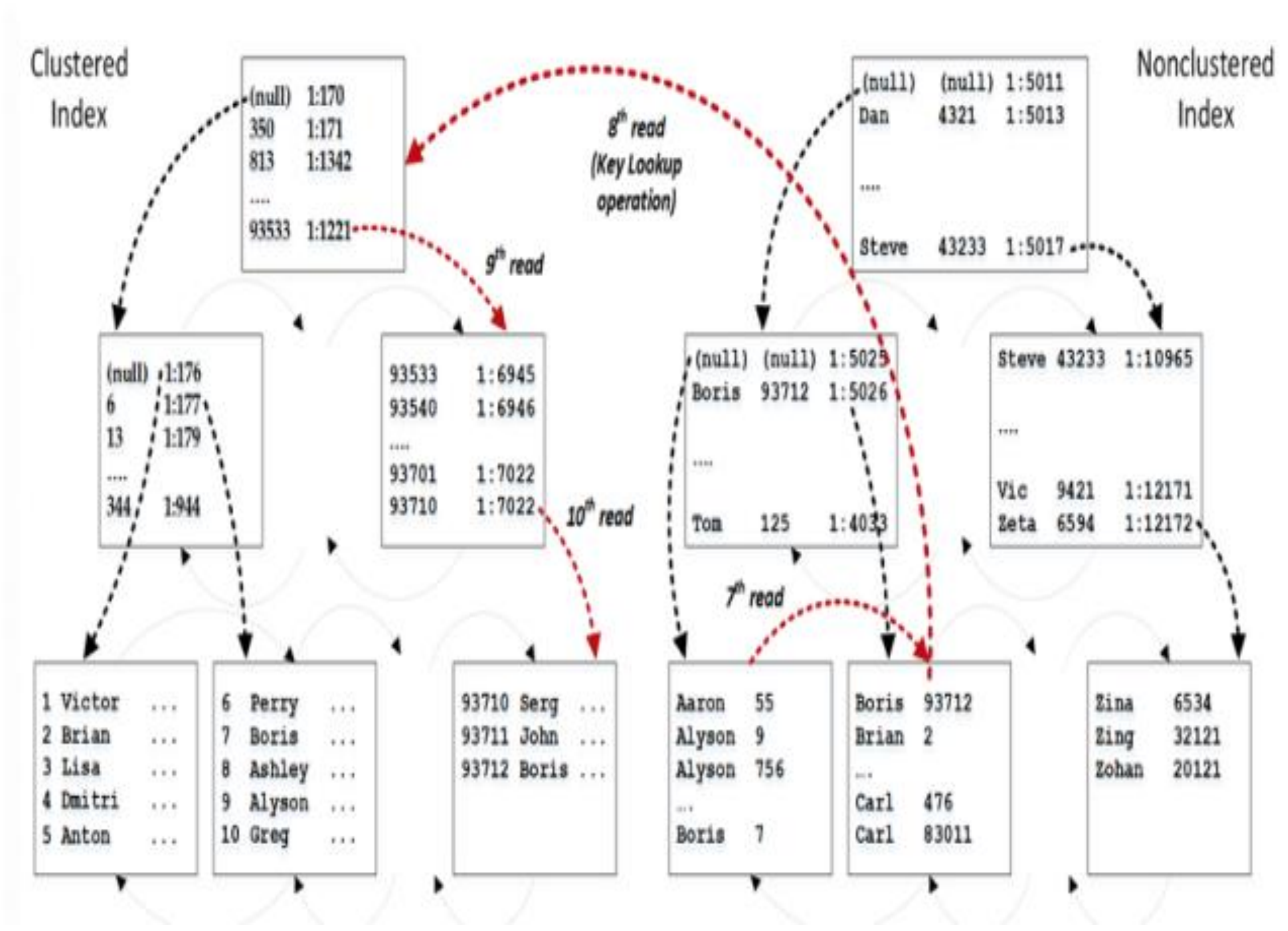


Customer Number	First Name	Last Name
4	Anna	Victoria

Clustered and Non-Clustered

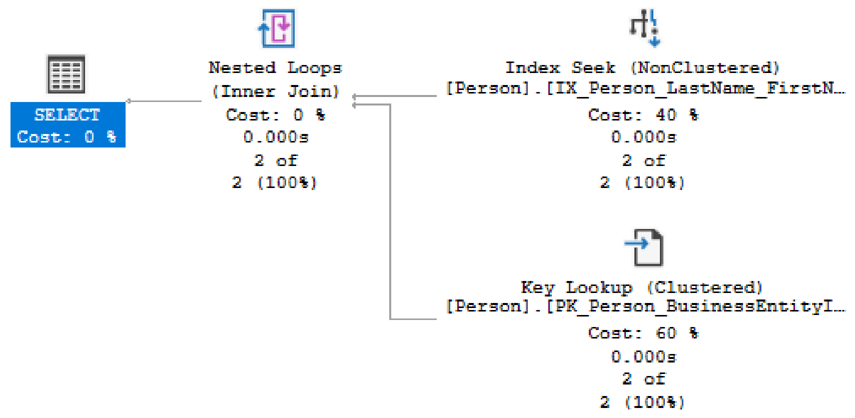


Non clustered index and lookup's

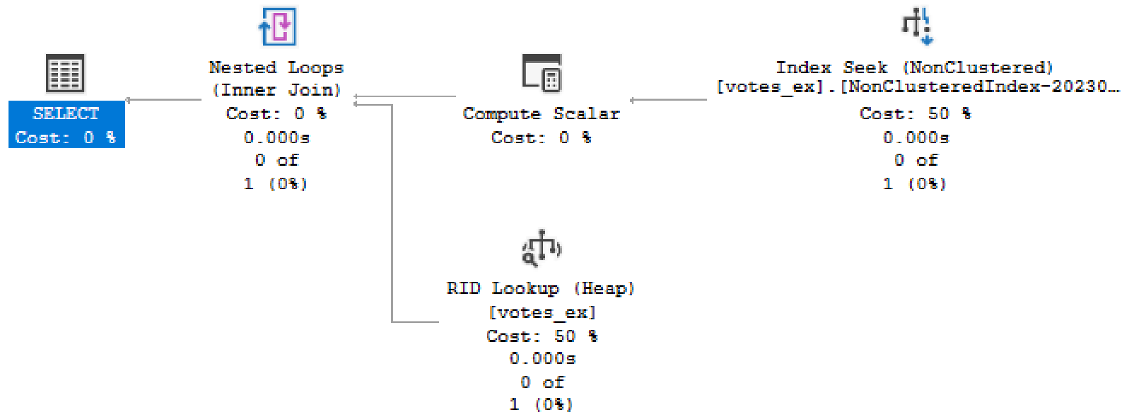


Lookups

- Key Lookup



- Row Lookup



SARGable vs Non-SARGable

What is a "SARGable" query?: Search Argument-able

- SARGable query will
- use index effectively (e.g. index seek instead of scan)
- save system resources (e.g. CPU time, DISK I/O)
- Improve the query performance
- SARGable operators:

  (e.g. <, >, =, <=, >=, conditional LIKE, and BETWEEN), pass the correct datatype

Non-SARGable query pattern

- SUBSTRING
- LEFT
- LTRIM
- RTRIM
- User defined functions

Statistics and Cardinality Estimation

- SQL Server stores information about data distribution in the index in internal objects called statistics.
- The Query Optimizer uses statistics to create query plans that improve query performance.
- The Query Optimizer uses these statistics to estimate the *cardinality*, or number of rows, in the query result.

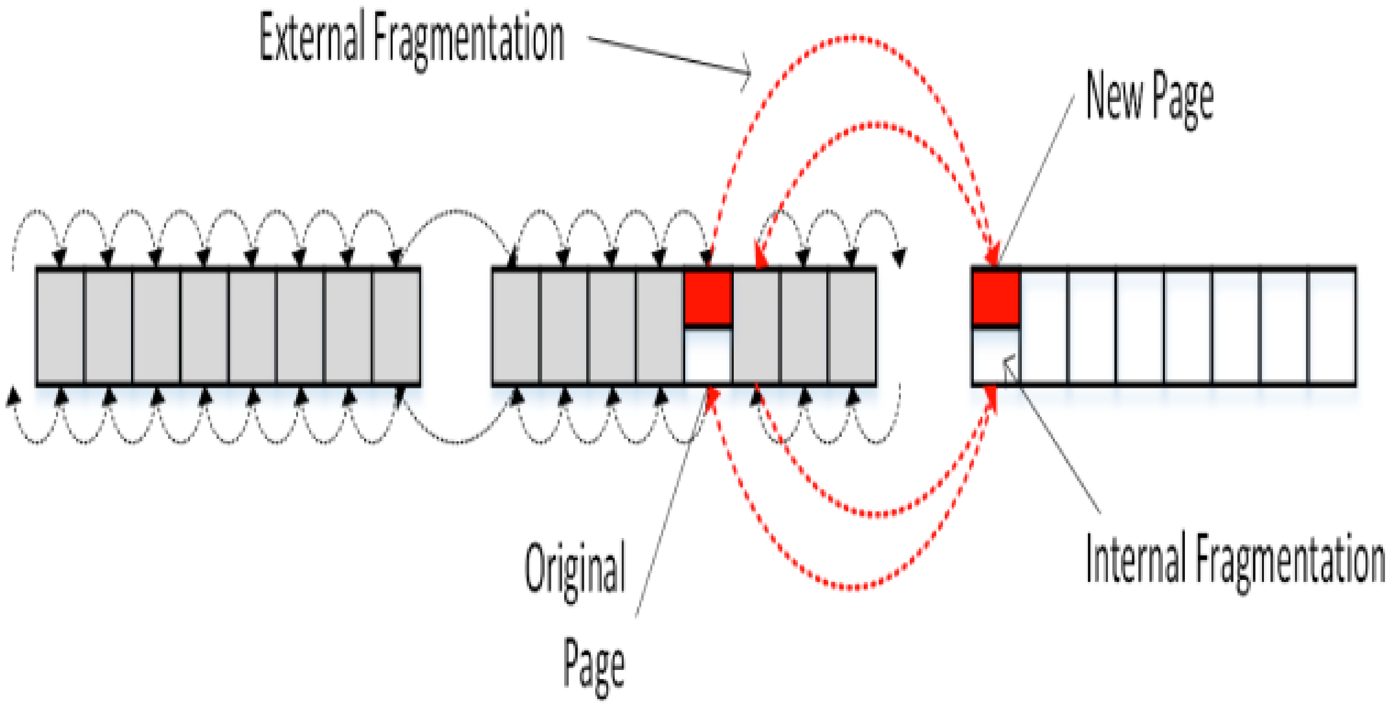
Check statistics

- **DBCC SHOW_STATISTICS**

  Three result sets

  - Header/metadata
  - Density
  - Histogram
- **sys.dm_db_stats_properties**

Fragmentation



Reduce Fragmentation

- Rebuild the indexes
  - If Fragmentation percentage is > 30% , go for rebuild
- Reorganize the indexes
  - If Fragmentation is between 5% - 30% , go with Index Reorganize

DMV to track fragmentation

sys.dm_db_index_physical_stats

- Columns to note:
  - avg_page_space_used_in_percent
  - avg_fragmentation_in_percent
  - fragment_count

Execution Plans in SQL Server

- Actual Execution Plan
  - Once the query gets completed
- Estimated Execution Plan
  - Compiled plan
- Live query statistics
  - Runtime statistics during query execution

How to capture the execution plans

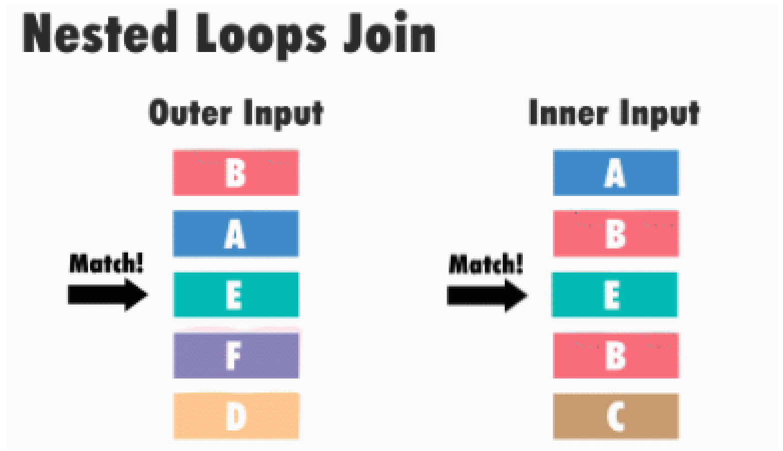
- SSMS (Ctrl + L for Estimated, Ctrl + M for actual)
- Profiler
- xEvents
- DMVs to capture the cached compile plans
  - dm_exec_cached_plans
  - dm_exec_query_plan_stats
- Query Store

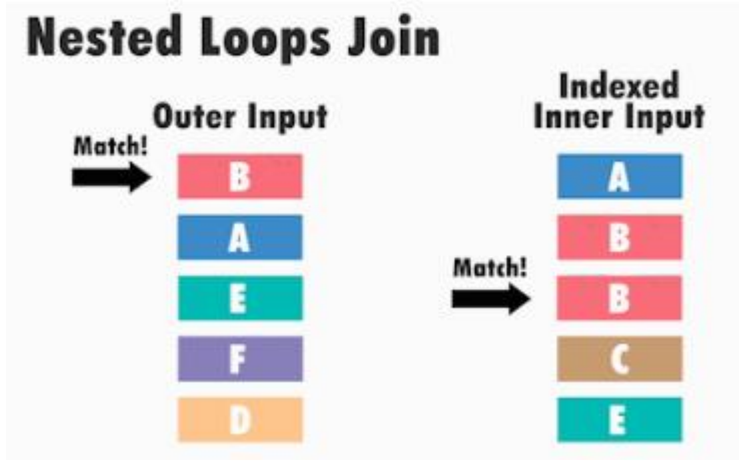
Physical Joins in SQL server

- **Nested Loops Join**
- **Merge joins**
- **Hash Match joins**
- **Adaptive Join (SQL 2017 and above)**

Physical Joins in SQL server – Continue..

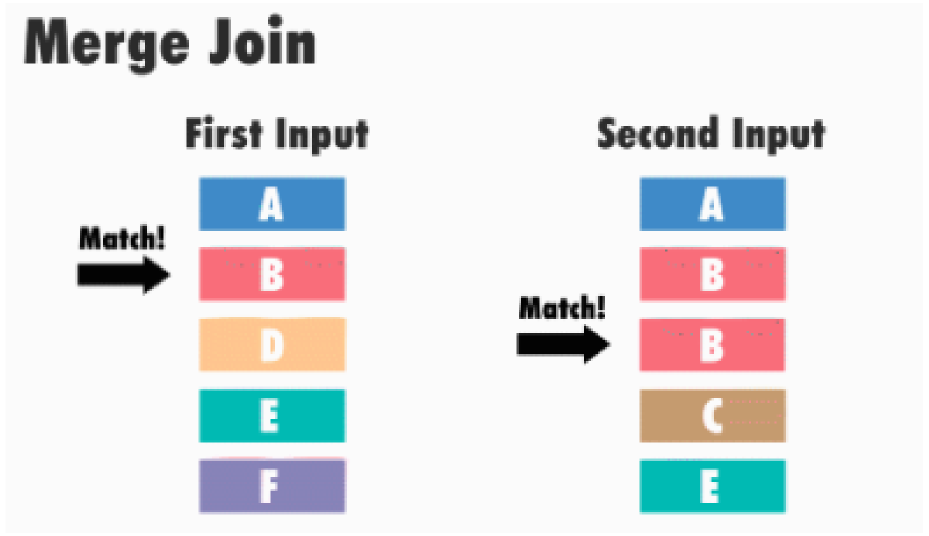
Nested Loops Join

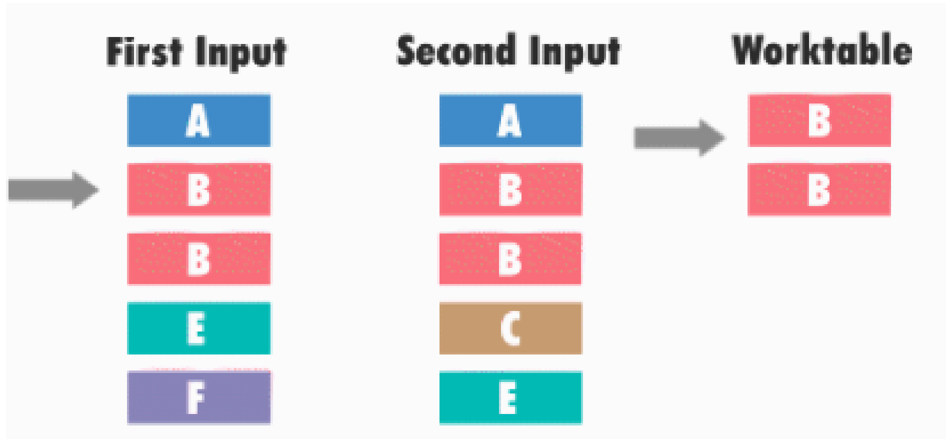




Physical Joins in SQL server – Continue..

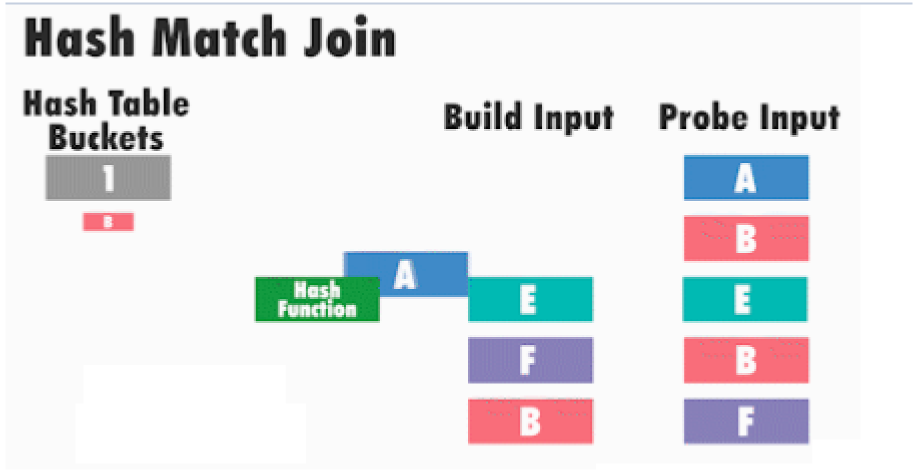
Merge Join (Fastest join if the data is pre sorted)

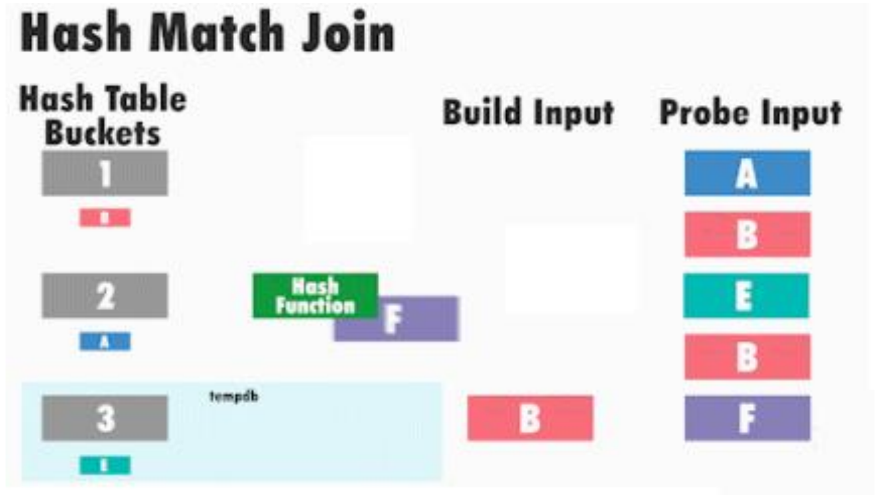




Physical Joins in SQL server – Continue..

Hash Match





Comparing Join Types

	Nested Loop Join	Merge Join	Hash Join
Best use case	At least input is small; index on the join column(s) in another input	Medium to large inputs, sorted on index key	Medium to large inputs
Requires sorted input	No	Yes	No
Blocking operator	No	No	Yes (Build phase only)
Uses memory	No	No	Yes
Uses tempdb	No	No (Sort may spill to tempdb)	Yes, in case of spills

Operators in SQL Server execution plan

Operator	Purpose
Table/Index Scan	Reading data
Index Seek	Reading data
Lookup	Reading data
Nested Loops	Combining data
Merge Join	Combining data
Hash Match	Combining data (Blocking operator)
Sort	Grouping and Ordering (blocking operator)
Table/Index Insert Table/Index Update Table/Index Delete	Modifying data
Parallelism	Performance

What to Look for in an Execution Plan

- First operator
  - SELECT/INSERT/ UPDATE/DELETE
- Missing Indexes
- Warnings
- Estimated versus actual number of rows
- Operator cost
- Cached plan size
- CardinalityEstimationModelVersion
- CompileCPU, CompileMemory, CompileTime
- RetrievedFromCache
- Query Time Statistics

Common Query Plan Issues

- Statistics
- Missing Indexes
- Non-SARGable Predicates
- Multi-statement Table Valued Function (TVF)
- Parameter Sniffing
- Data Type Mismatch