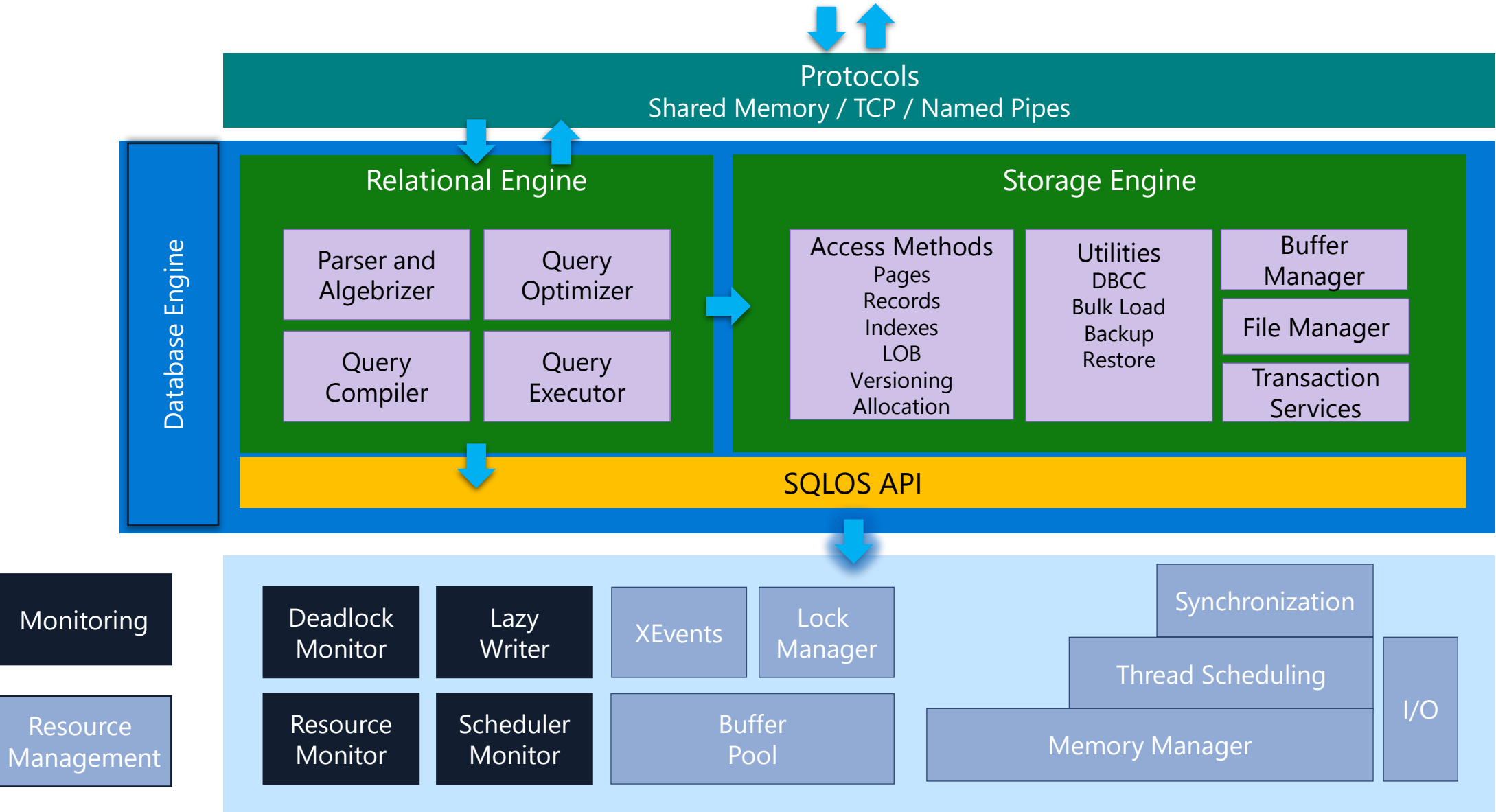


Inside the Database Engine



SQL Server Operating System (SQLOS)

Application layer between Microsoft SQL Server components and the Windows Operating System.

Centralizes resource allocation to provide more efficient management and accounting.

The SQLOS is used by the SQL Server relational database engine for system-level services.

Abstracts the concepts of resource management from components, providing:

- **Scheduling and synchronization support**
- **Memory management and caching**
- **Resource governance**
- **Diagnostics and debug infrastructure**
- **Scalability and performance optimization**

Two Main Functions of SQLOS

Management

- Memory Manager
- Process Scheduler
- Synchronization
- I/O
- Support for Non-Uniform Memory Access (NUMA) and Resource Governor

Monitoring

- Resource Monitor
- Deadlock Monitor
- Scheduler Monitor
- Lazy Writer (Buffer Pool management)
- Dynamic Management Views (DMVs)
- Extended Events
- Dedicated Administrator Connection (DAC)

Dynamic Management Views and Functions

Category	Description
sys.dm_exec_%	Execution and connection information
sys.dm_os_%	Operating system related information
sys.dm_tran_%	Transaction management information
sys.dm_io_%	I/O related information
sys.dm_db_%	Database information

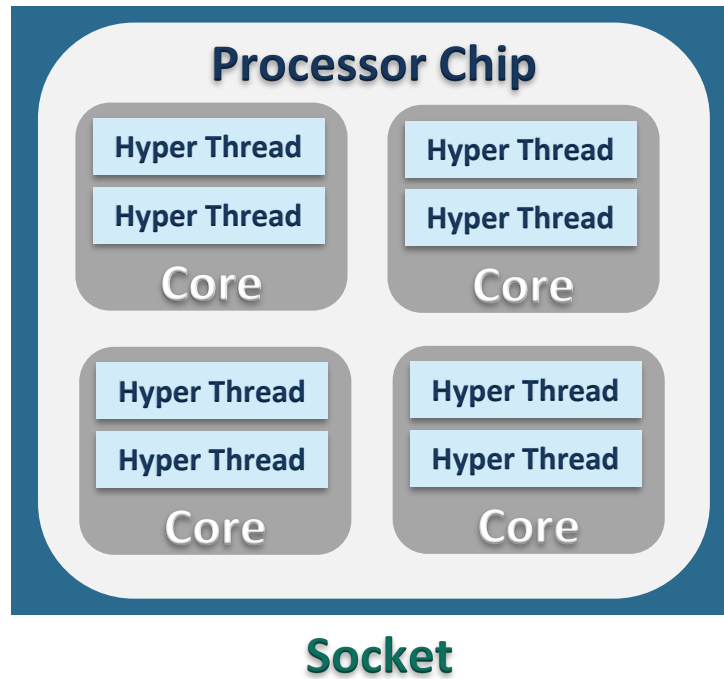
Using Dynamic Management Objects (DMOs)

- Must reference using the sys schema
- Two basic types:
 - Real-time state information
 - Historical information

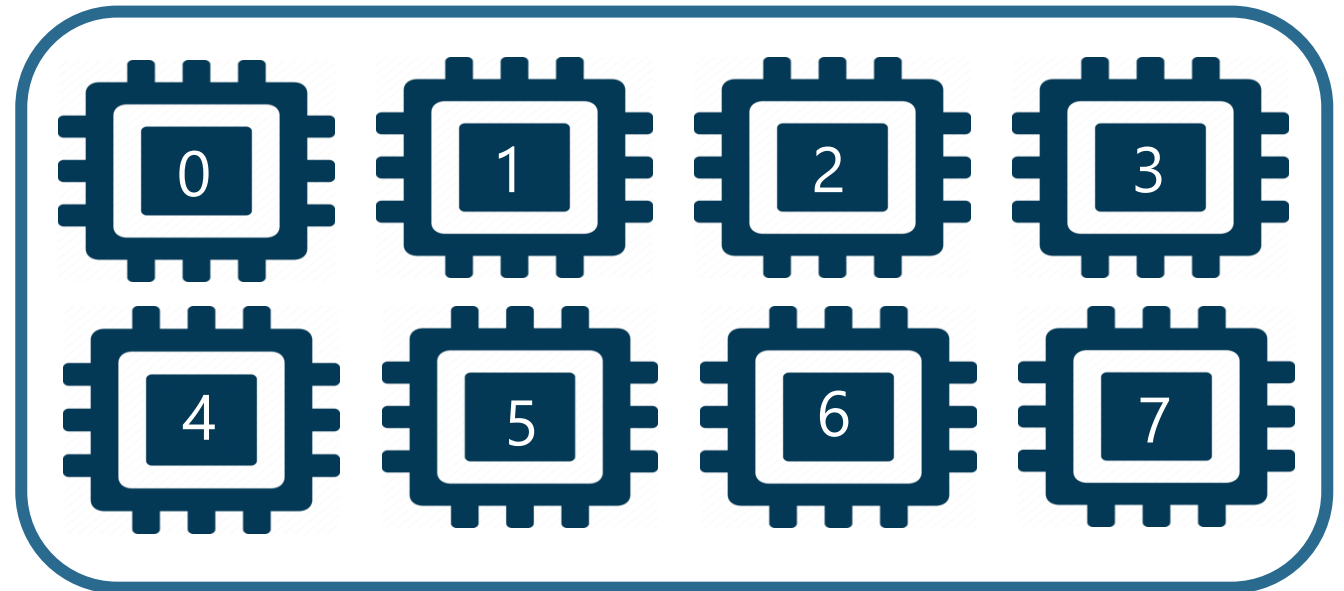
```
SELECT cpu_count, hyperthread_ratio,  
       scheduler_count, scheduler_total_count,  
       affinity_type, affinity_type_desc,  
       softnuma_configuration, softnuma_configuration_desc,  
       socket_count, cores_per_socket, numa_node_count,  
       sql_memory_model, sql_memory_model_desc  
FROM sys.dm_os_sys_info
```

CPU Architecture

Physical Hardware



Logical Processors as seen by the OS



Microsoft SQL Server Scheduling Terminology

Batch

- A statement or set of statements submitted to SQL Server by the user (a query), also referred to as a request
- Monitor with `sys.dm_exec_requests`

Task

- A batch will have one or more tasks (aligns with statements)
- Monitor with `sys.dm_os_tasks`

Worker Thread

- Each task will be assigned to a single worker thread for the life of the task
- Monitor with `sys.dm_os_workers`

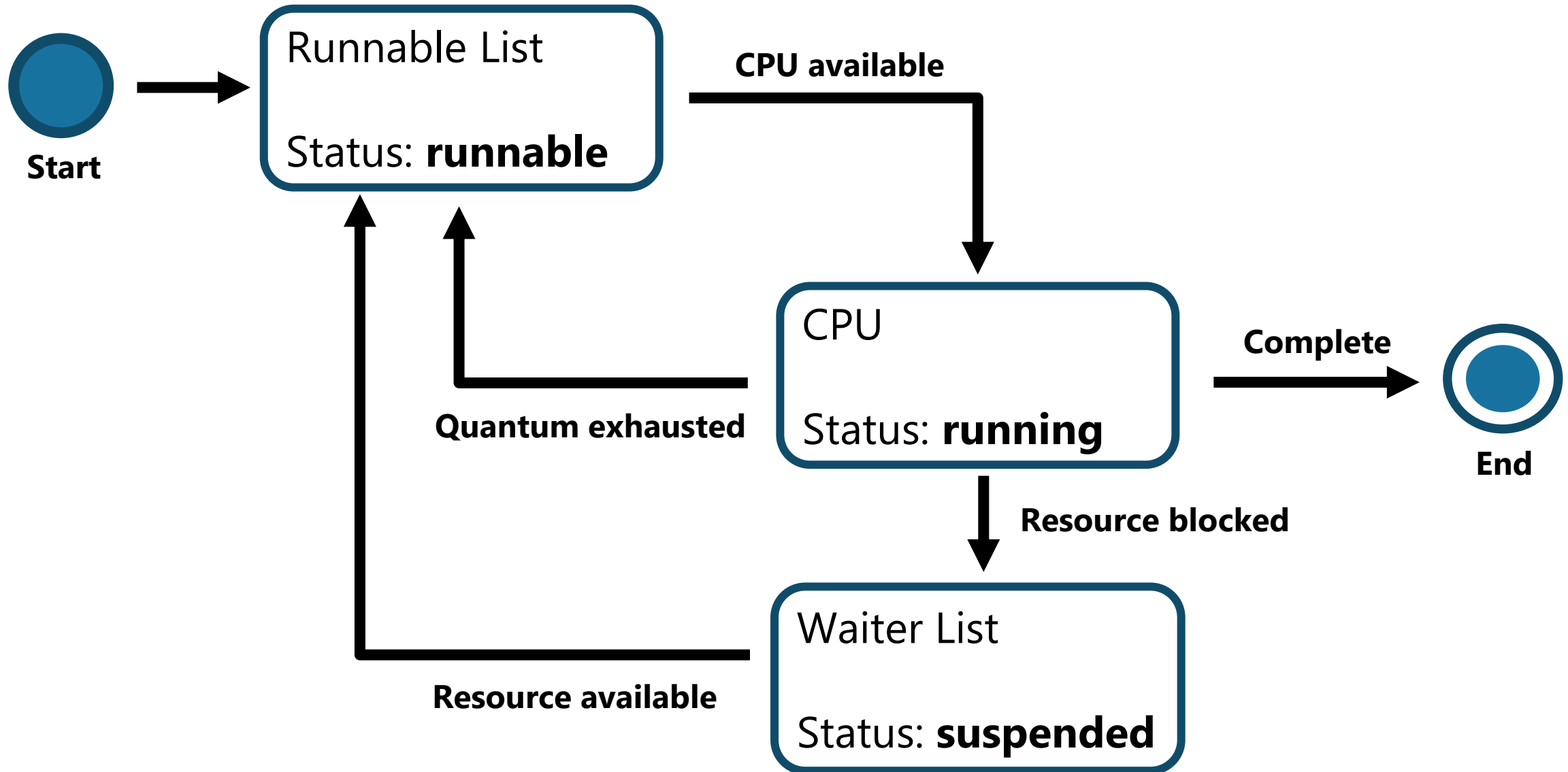
Thread States and Queues

Runnable: The thread is currently in the Runnable Queue waiting to execute. (First In, First Out).

Running: One active thread executing on a processor.

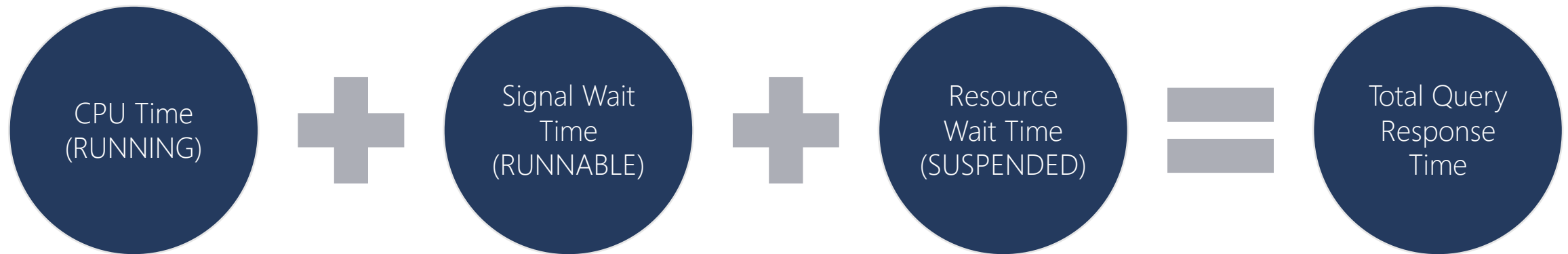
Suspended: Placed on a Waiter List waiting for a resource other than a processor. (No specific order).

Yielding



Task Execution Model

- The full cycle between the several task states, for how many times it needs to cycle, is what we experience as the total query response time.



Relevant Dynamic Management Views (DMVs)

sys.dm_os_wait_stats

- Returns information about all the waits encountered by threads that ran.
- Includes wait type, number of tasks that waited in the specific wait type, total and max wait times, and the amount of signal waits.

sys.dm_os_waiting_tasks

- Returns information about the wait queue of tasks actively waiting on some resource.

sys.dm_exec_requests

- Returns information about each request that is in-flight.
- Includes session owning the request and status of the request, which will reflect the status of one or more tasks assigned to the request.

Waiting Tasks DMV

```
SELECT w.session_id, w.wait_duration_ms, w.wait_type,  
       w.blocking_session_id, w.resource_description,  
       s.program_name, t.text, t.dbid, s.cpu_time, s.memory_usage  
FROM sys.dm_os_waiting_tasks as w  
     INNER JOIN sys.dm_exec_sessions as s  
       ON w.session_id = s.session_id  
     INNER JOIN sys.dm_exec_requests as r  
       ON s.session_id = r.session_id  
     OUTER APPLY sys.dm_exec_sql_text (r.sql_handle) as t  
WHERE s.is_user_process = 1;
```

session_id	wait_duration_ms	wait_type	blocking_session_id	resource_description
58	8563	LCK_M_S	62	keylock hobtid=72057594047365120 dbid=5 id=lock1...

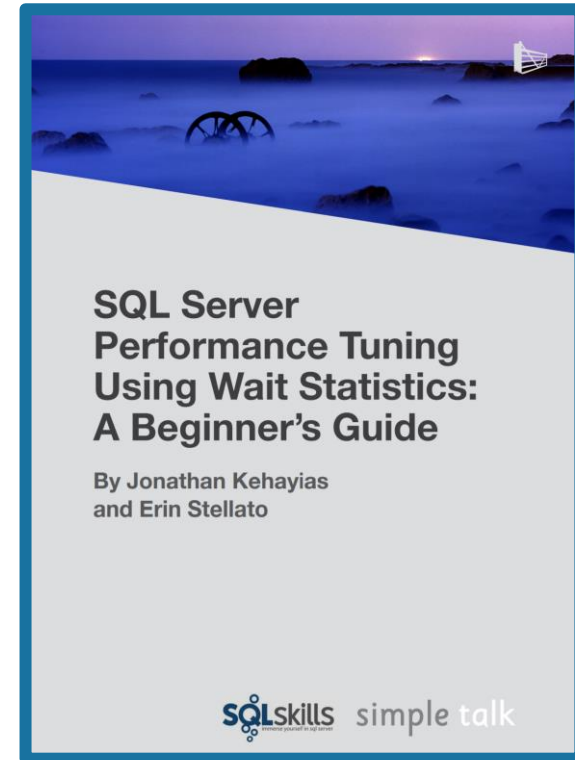
Troubleshooting Wait Types

Aaron Bertrand – Top Wait Types

<https://sqlperformance.com/2018/10/sql-performance/top-wait-stats>

Paul Randal – SQL Skills Wait Types Library

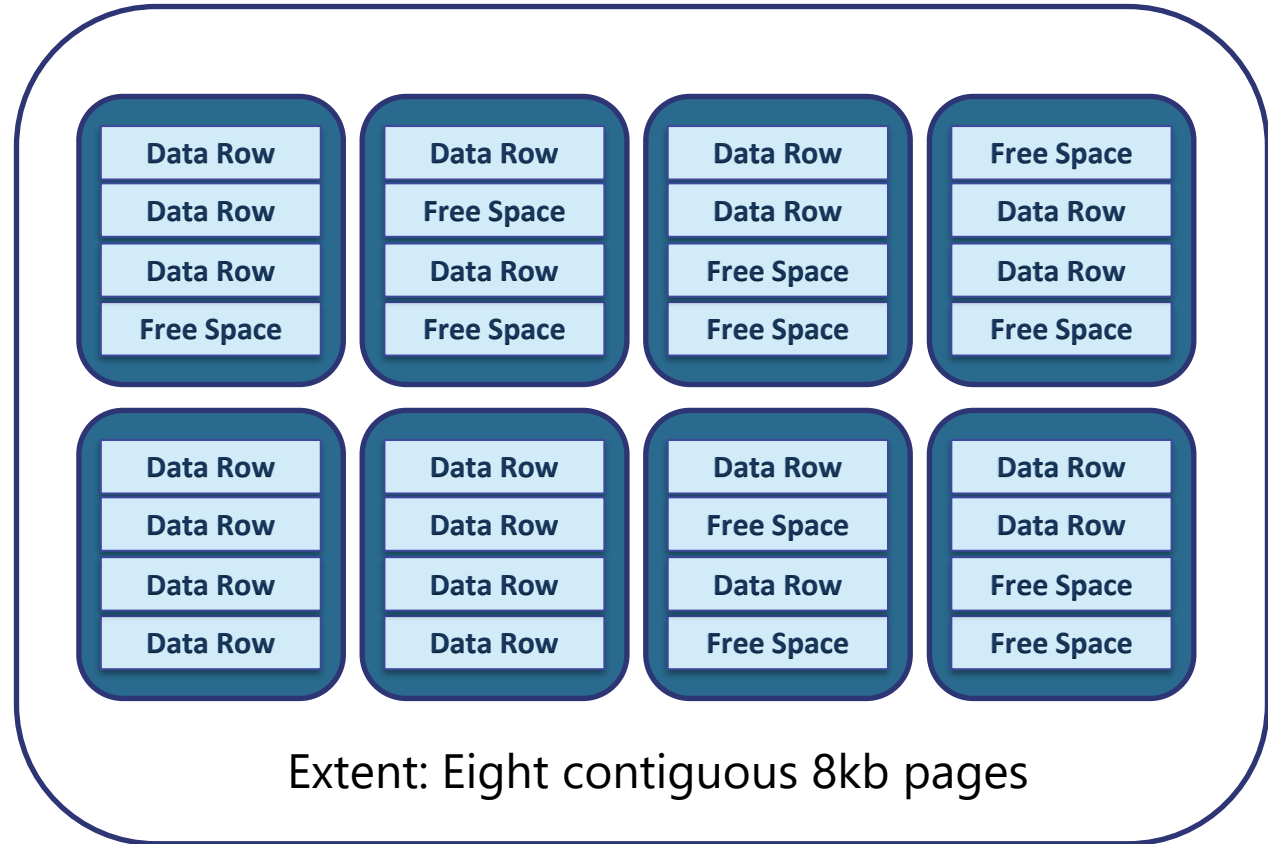
<https://www.sqlskills.com/help/waits/>



SQL Server Object Allocation



Primary Data File (.mdf)
Secondary Data File (.ndf)



Uniform extents: Pages used by a single object.

Mixed extents: Pages used by different objects.

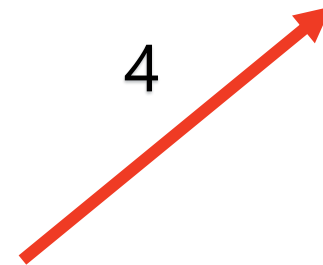
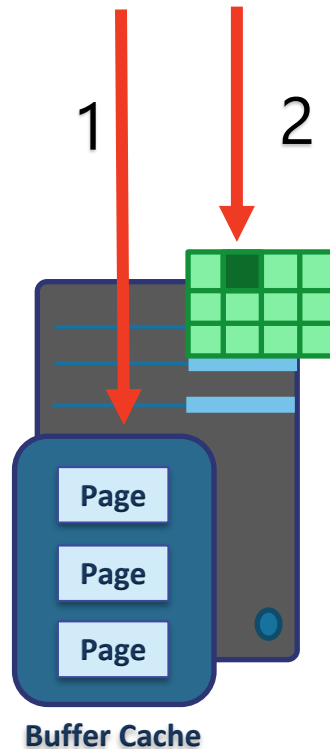
SQL Server Disk I/O (Write-Ahead Logging)

```
UPDATE Accounting.BankAccounts  
SET Balance -= 200  
WHERE AcctID = 1
```

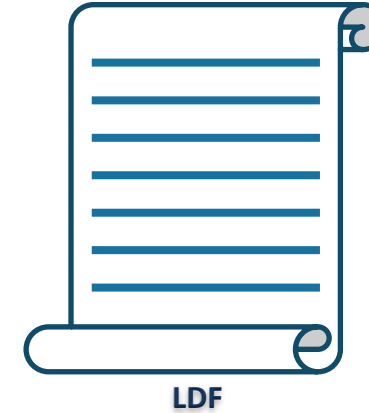
1. Data modification is sent to buffer cache in memory.

2. Modification is recorded in the log cache.

3. Data pages are located or read into the buffer cache and then modified.



4. Log cache record is flushed to the transaction log



5. At checkpoint, dirty data pages are written to the database file.



Log Buffer Flushing

SQL Server will flush the log buffer to the log file

- SQL Server gets a commit request of a transaction that changes data.
- The log buffer fills up. (Max size 60kb.)
- SQL Server needs to harden dirty data pages (checkpoints)
- Manually request a log buffer flush using the `sys.sp_flush_log` procedure

Log buffer flushing results in a WRITELOG wait type.

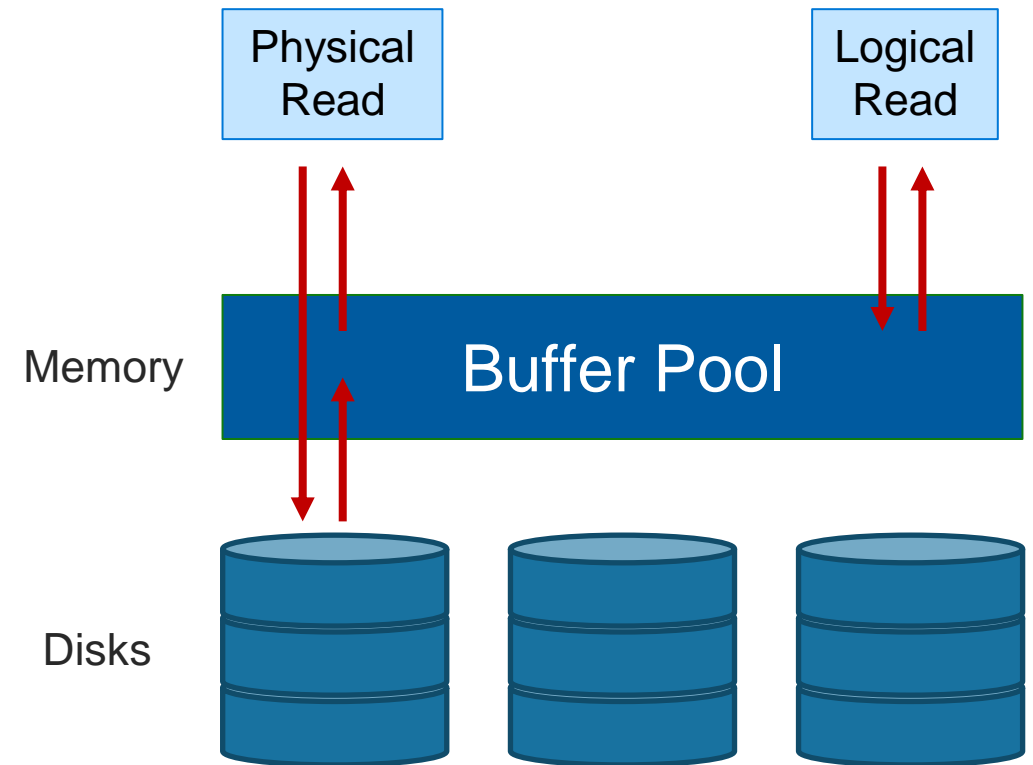
SQL Server Buffer Pool

Stores 8 kilobytes (KB) pages of data to avoid repeated disk I/O.

- Pages held in the buffer until the space is needed by something else.

Lazy Writer searches for eligible buffers.

- If the buffer is dirty, an asynchronous write (lazy write) is posted so that the buffer can later be freed.
- If the buffer is not dirty, it is freed.



Checkpoints

Flushes dirty pages from the buffer pool to the disk. Frequency of checkpoints varies based on the database activity and recovery interval.

Automatic (default) – Database engine issues checkpoints automatically based on the server level “recovery interval” configuration option

Indirect (new in SQL Server 2012) – Database engine issues checkpoints automatically based on the database level TARGET_RECOVERY_TIME

```
ALTER DATABASE [AdventureWorksPTO] SET TARGET_RECOVERY_TIME = 60 SECONDS
```

Manual – Issued in the current database for your connection when you execute the T-SQL CHECKPOINT command

Internal – Issued by various server operations

SET STATISTICS IO

```
SET STATISTICS IO ON
GO
SET STATISTICS TIME ON
SELECT SOH.SalesOrderID, SOH.CustomerID,
OrderQty, UnitPrice, P.Name
FROM Sales.SalesOrderHeader AS SOH
JOIN Sales.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID
JOIN Production.Product AS P
ON P.ProductID = SOD.ProductID
SET STATISTICS IO, TIME OFF
```

Used to identify physical reads and logical reads for a query

```
(121317 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server r
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server
Table 'SalesOrderDetail'. Scan count 1, logical reads 428, physical reads 0, pag
Table 'Product'. Scan count 1, logical reads 15, physical reads 0, page server r
Table 'SalesOrderHeader'. Scan count 1, logical reads 57, physical reads 0, page

SQL Server Execution Times:
    CPU time = 94 ms,  elapsed time = 1653 ms.
```

Page types

Types	Page Type (ID)	Description
Data and Index	Data (1)	Data rows with all data, except text, ntext, image, nvarchar(max), varchar(max), varbinary(max), and xml data, when text in row is set to ON
	Index (2)	Index Entries
	Text/Image (3 or 4)	Large Object Data Type, variable length columns when the data row exceeds 8 kilobytes (KB)
Allocation	GAM, SGAM (8 and 9)	Extent Allocation information
	PFS (11)	Information about page allocation and free space available on pages
	IAM (10)	Information about extents used by a table or index per allocation unit
Restore	Bulk Changed Map (17)	Information about extents modified by bulk operations since the last BACKUP LOG statement per allocation unit
	Differential Changed Map (16)	Information about extents that have changed since the last BACKUP DATABASE statement per allocation unit
Metadata	Boot (13)	Information about the database; Each database has only one Boot page
	File Header (15)	Information about the file. It is the first page (page 0) in every file

The Role of Allocation Maps and PFS in Object Allocation

PFS and IAM are used to determine when an object needs a new extent allocated

GAMs and SGAMs are used to allocate the extent

DBCC IND

```
USE AdventureWorks2012
DBCC TRACEON(3604) -- Print to results pane
DBCC IND(0, 'HumanResources.Employee', -1)
-- Parameter 1: Is the DatabaseName, 0 is current database
-- Parameter 2: The table name
-- Parameter 3: Index ID, -1 Shows all indexes, -2 shows only IAM Pages
```

	PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType	IndexLevel	NextPageFID	NextPagePID	PrevPageFID	PrevPagePID
1	1	874	NULL	NULL	1237579447	1	1	72057594045136896	In-row data	10	NULL	0	0	0	0
2	1	875	1	874	1237579447	1	1	72057594045136896	In-row data	2	1	0	0	0	0
3	1	1048	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1049	0	0
4	1	1049	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1050	1	1048
5	1	1050	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1051	1	1049
6	1	1051	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1052	1	1050
7	1	1052	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1053	1	1051
8	1	1053	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	1	1054	1	1052
9	1	1054	1	874	1237579447	1	1	72057594045136896	In-row data	1	0	0	0	1	1053
10	1	9287	NULL	NULL	1237579447	2	1	72057594050510848	In-row data	10	NULL	0	0	0	0
11	1	9286	1	9287	1237579447	2	1	72057594050510848	In-row data	2	0	0	0	0	0
12	1	9289	NULL	NULL	1237579447	3	1	72057594050576384	In-row data	10	NULL	0	0	0	0

Query executed successfully.	STUDENTSERVER (12.0 RTM)	STUDENTSERVER\Student ...	AdventureWorks2012	00:00:0
------------------------------	--------------------------	---------------------------	--------------------	---------

DBCC PAGE

```
DBCC TRACEON(3604) -- Print to results pane
DBCC PAGE (0,1,0,3)
-- Parameter 1: Is the DatabaseName, 0 is current database
-- Parameter 2: The File ID
-- Parameter 3: The Page ID
-- Parameter 4: The print option, 3 is verbose
```

.00 % <

Messages

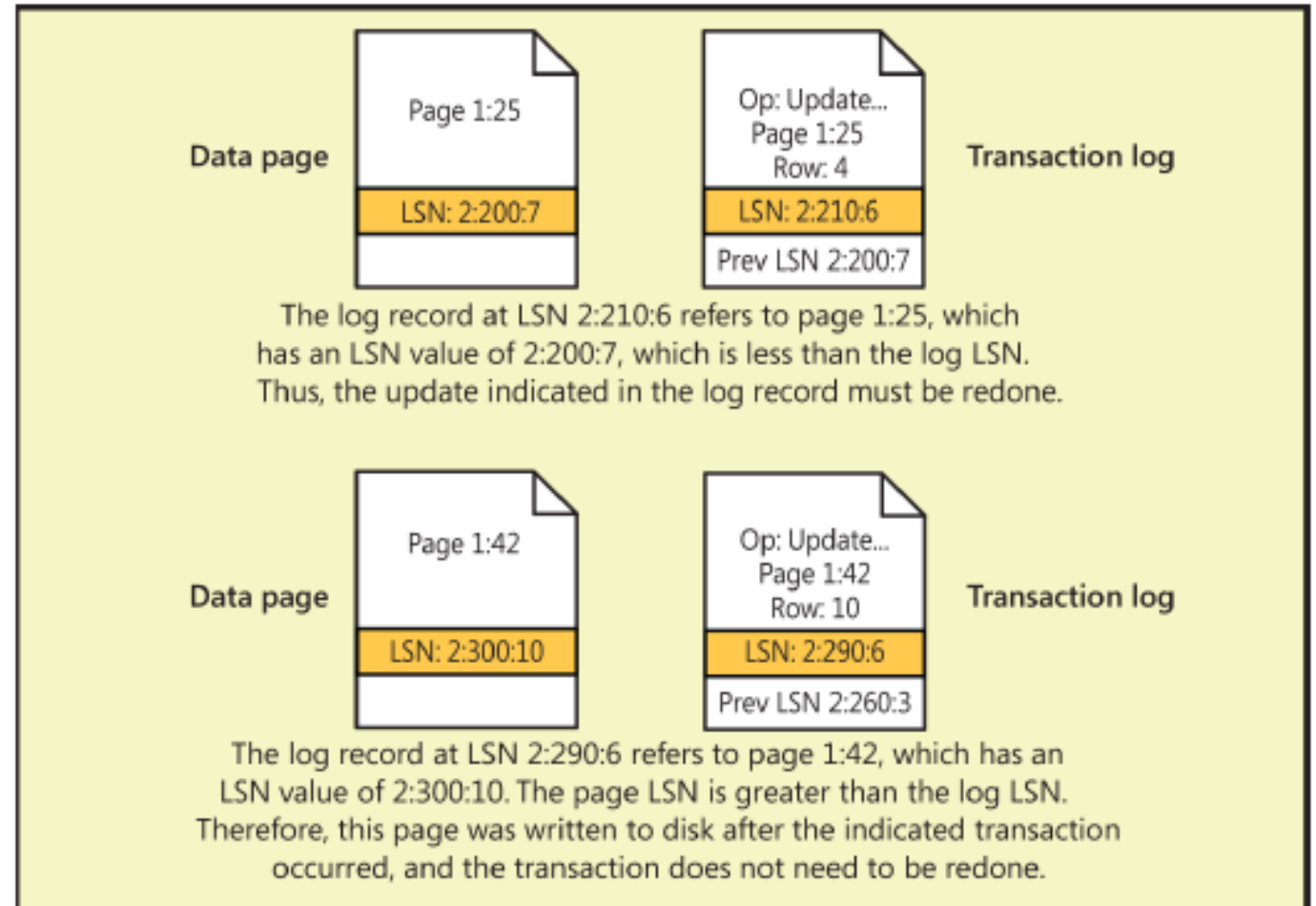
PAGE HEADER:

Page @0x000000027757A000

m_pageId = (1:0)	m_headerVersion = 1	m_type = 15
m_typeFlagBits = 0x0	m_level = 0	m_flagBits = 0x208
m_objId (AllocUnitId.idObj) = 99	m_indexId (AllocUnitId.idInd) = 0	Metadata: AllocUnitId = 6488064
Metadata: PartitionId = 0	Metadata: IndexId = 0	Metadata: ObjectId = 99
m_prevPage = (0:0)	m_nextPage = (0:0)	pminlen = 0
m_slotCnt = 1	m_freeCnt = 6989	m_freeData = 7831
m_reservedCnt = 0	m_lsn = (181:50952:34)	m_xactReserved = 0
m_xdesId = (0:0)	m_ghostRecCnt = 0	m_tornBits = -820886669
DB Frag ID = 1		

Page LSN and recovery

- Last LSN in the page header
- Log Record has two LSNs
 - Previous LSN from the data page
 - Current log record LSN
- All used for recovery

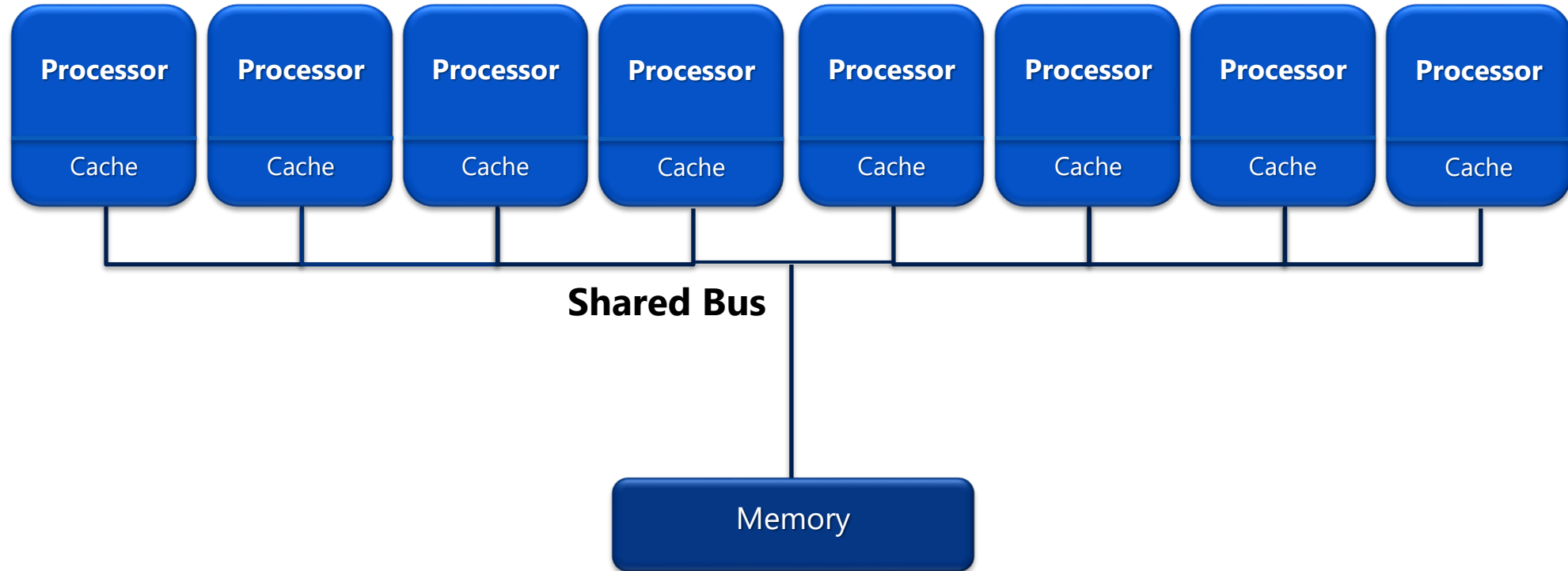


SQL Server 2014 VLF Growth Improvement

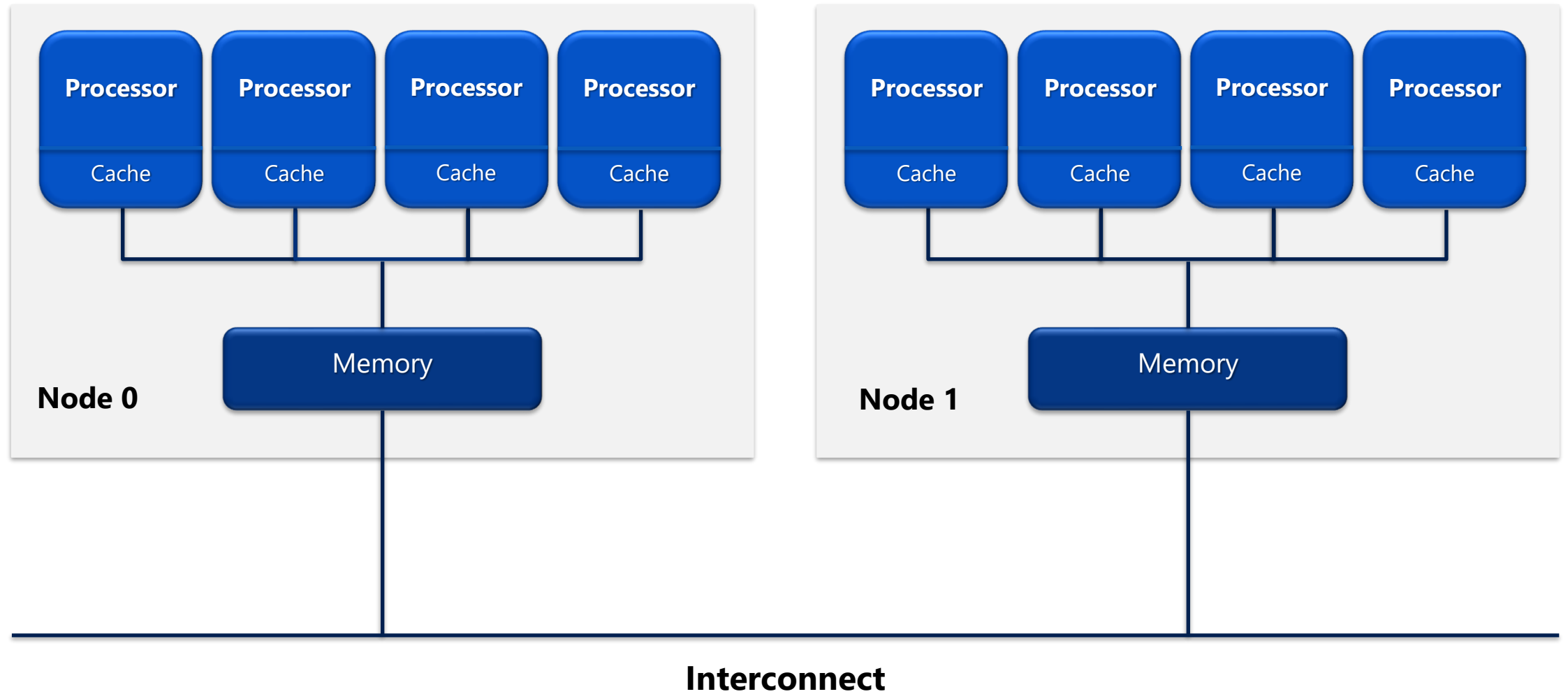
- Is the growth size less than 1/8 the size of the current log size?
 - Yes: create 1 new VLF equal to the growth size
 - No: use the previous formula
- Example of a 256 MB log file with an autogrowth setting of 5 MB
 - 2012 and earlier: 10 auto-grows of 5MB would add 4 VLFs x 10 auto-grows
 - 2014 and later: 10 auto-grows of 5MB each would only create 10 VLFs

Grow Iterations + Log size	Up to SQL Server 2012	From SQL Server 2014
0 (256 MB)	8	8
10 (306 MB)	48	18
20 (356 MB)	88	28
80 (656 MB)	328	88
250 (1.2 GB)	1008	258
3020 (15 GB)	12091	3028

Symmetric Multi-Processing (SMP)



Non-Uniform Memory Access (NUMA)



SQL Server Configuration

Processor Configuration Settings And Best Practices

Affinity Mask

- Assigns CPUs for SQL Server use
- Set via sp_configure or Alter Server Configuration
- Only required in specific scenarios

Max Degree of Parallelism (MAXDOP)

- Maximum number of processors that are used for the execution of a query in a parallel plan. This option determines the number of threads that are used for the query plan operators that perform the work in parallel.

Cost Threshold for Parallelism

- Only queries with a cost that is higher than this value will be considered for parallelism
- Only required when dealing with excessive parallelism

Max Worker Threads

- Number of threads SQL Server can allocate
- Recommended value is 0. SQL Server will dynamically set the Max based on CPUs and CPU architecture

SQL Server Configuration

MAXDOP Setting and Best Practices

Best Practice Recommendations (documented in [KB 2806535](#)):

Server with single NUMA node	Less than or equal to 8 logical processors	Keep MAXDOP at or below # of logical processors
Server with single NUMA node	Greater than 8 logical processors	Keep MAXDOP at 8
Server with multiple NUMA nodes	Less than or equal to 16 logical processors per NUMA node	Keep MAXDOP at or below # of logical processors per NUMA node
Server with multiple NUMA nodes	Greater than 16 logical processors per NUMA node	Keep MAXDOP at half the number of logical processors per NUMA node with a MAX value of 16

How to determine Thread Stack Memory

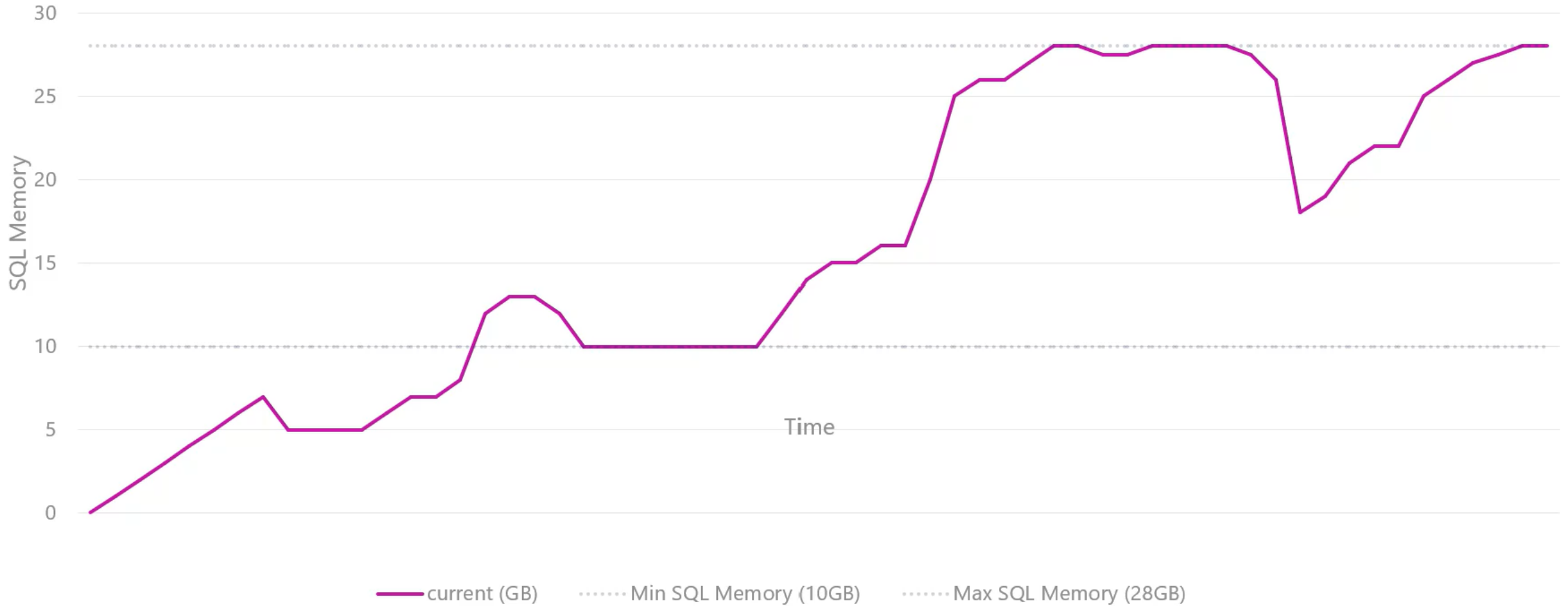
Maximum Worker Threads
 $512 + (\text{Processors} - 4) * 16$

*

2mb per thread

Cores	Threads	Memory (MB)
4	512	1,024
8	576	1,152
16	704	1,408
32	960	1,920
64	1,472	2,944
80	1,728	3,456

Dynamic Memory Management

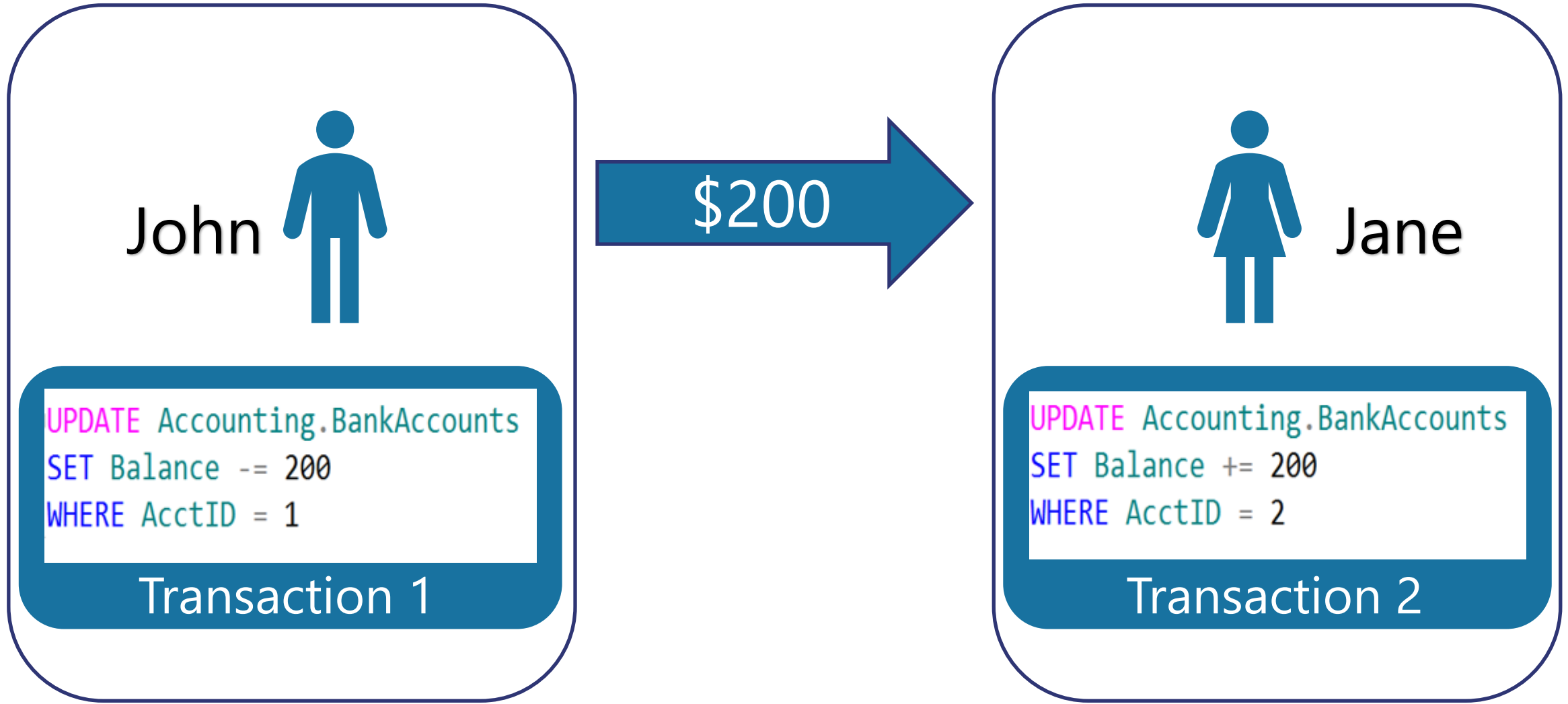


What is a Transaction?

A transaction is a series of one or more statements that need to operate as a single logical unit of work.

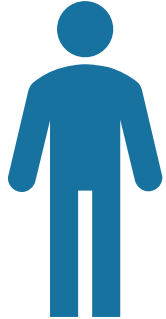
To qualify as a transaction, the logical unit of work must possess all four of the ACID properties.

Logical Units of Work – Auto Commit Transactions



Single Logical Unit of Work – Explicit Transactions

John



```
Begin Transaction BankUpdate  
UPDATE Accounting.BankAccounts  
SET Balance -= 2/0  
WHERE AcctID = 1  
  
UPDATE Accounting.BankAccounts  
SET Balance += 200  
WHERE AcctID = 2  
Commit Transaction
```



Jane

\$200

Transactions must pass the ACID test

Atomicity – All or Nothing

Consistent – Only valid data

Isolated – No interference

Durable – Data is recoverable

Working with Transactions

```
CREATE SCHEMA Accounting Authorization dbo
```

```
CREATE TABLE BankAccounts
```

```
(AcctID int IDENTITY,
```

```
AcctName char(15),
```

```
Balance money,
```

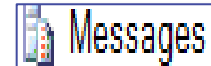
```
ModifiedDate date)
```

```
INSERT INTO Accounting.BankAccounts
```

```
VALUES('John', 500, GETDATE())
```

```
INSERT INTO Accounting.BankAccounts
```

```
VALUE('Jane', 750, GETDATE())
```



Messages

Msg 156, Level 15, State 1, Line 8

Incorrect syntax near the keyword 'INSERT'.

Msg 102, Level 15, State 1, Line 11

Incorrect syntax near 'VALUE'.

Creating Stored Procedures

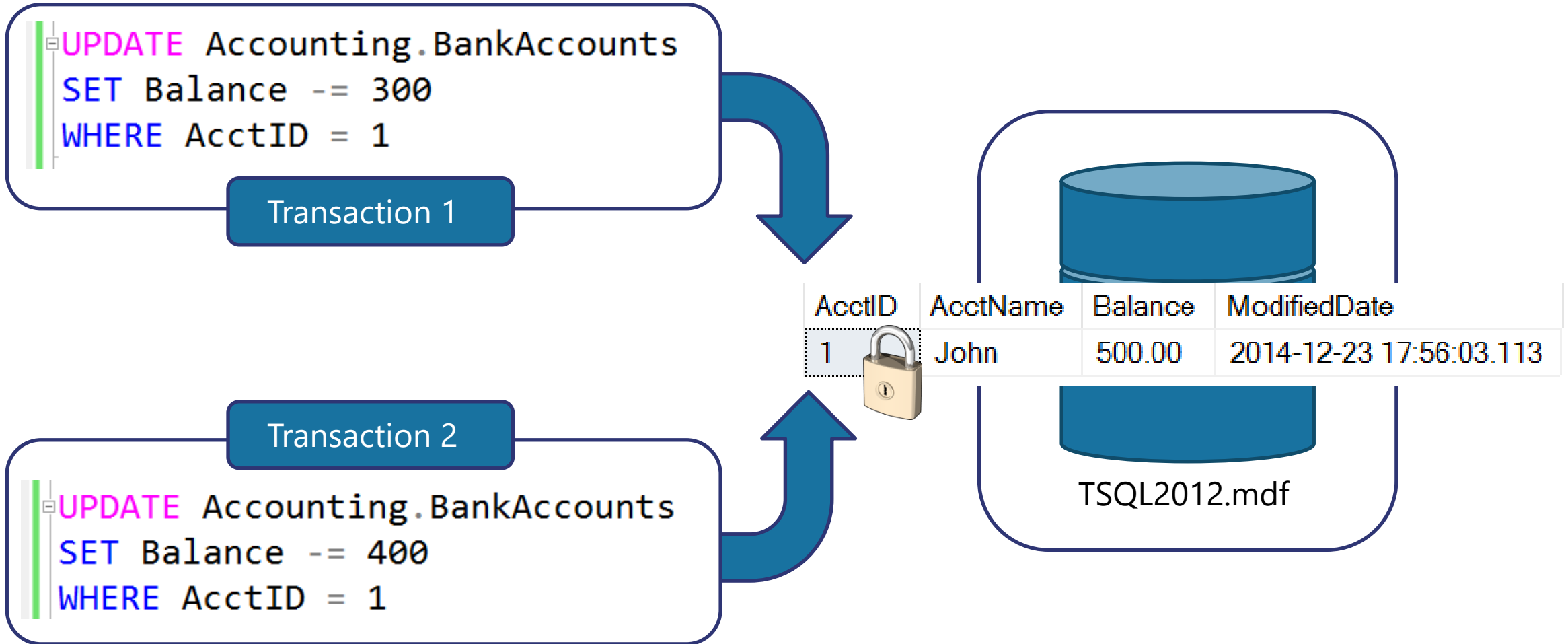
```
≡ ALTER PROCEDURE spAccountTransfer
  (@Amount smallmoney, @a1 tinyint, @a2 tinyint)
AS
  SET NOCOUNT ON

≡ UPDATE Accounting.BankAccounts
  SET Balance -= @Amount
  WHERE AcctID = @a1

≡ UPDATE Accounting.BankAccounts
  SET Balance += @Amount
  WHERE AcctID = @a2

PRINT 'Transfer Complete'
GO
```

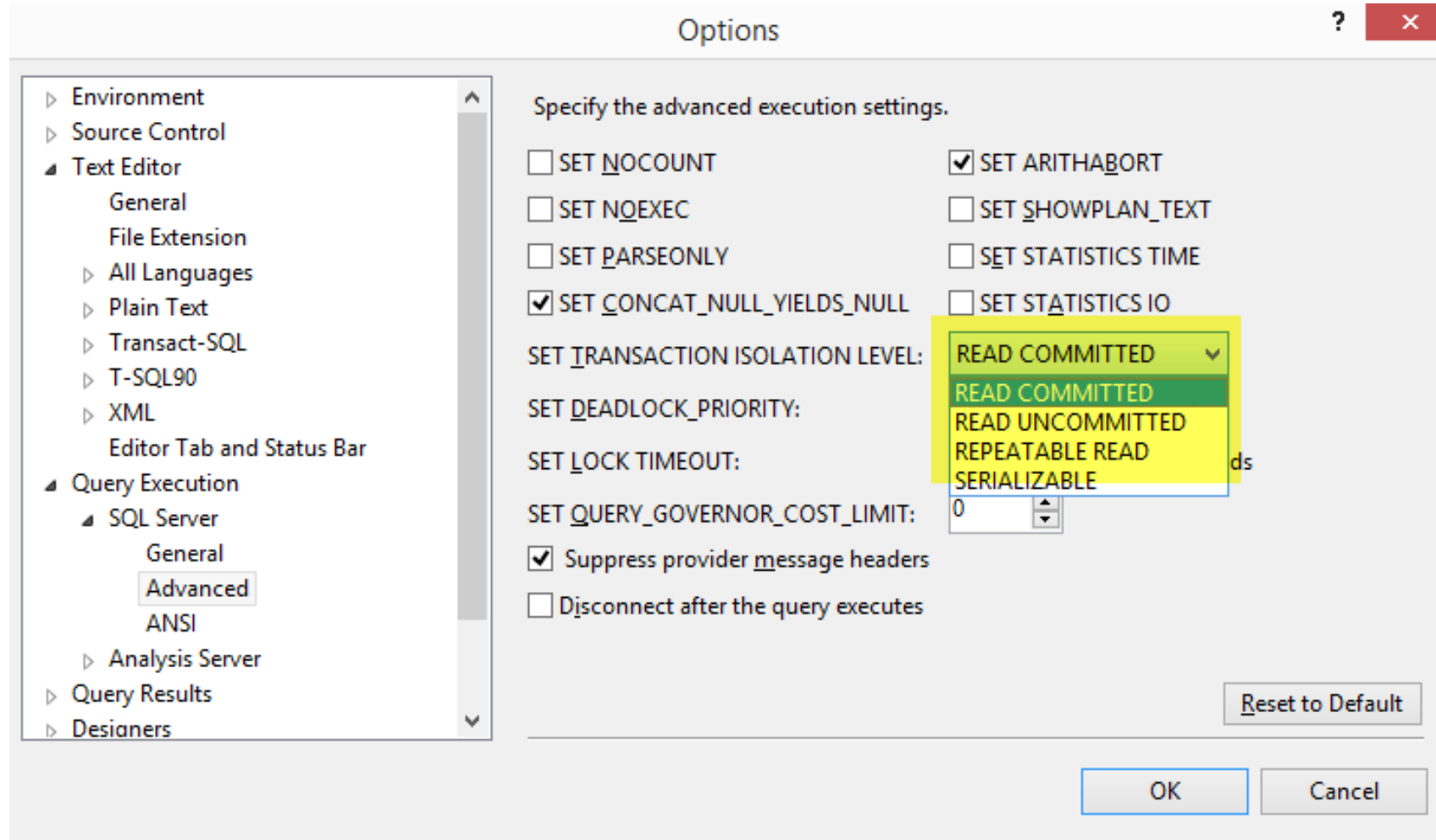
What is a Lock?



Transaction Isolation Levels

Isolation Level	Dirty Read	Lost Update	Nonrepeatable Read	Phantoms
Read uncommitted	Yes	Yes	Yes	Yes
Read committed (default)	No	Yes	Yes	Yes
Repeatable read	No	No	No	Yes
Serializable	No	No	No	No
Snapshot	No	No	No	No

Isolation Levels



Lost Updates

```
1 -- SQL Server Concurrency
2 -- Lost Update - Session 1
3 USE TSQLE2012
4 GO
5 DECLARE @OldBalance int, @NewBalance int
6 BEGIN TRAN
7     SELECT @OldBalance = Balance
8     FROM Accounting.BankAccounts
9     WHERE AcctID = 1
10    SET @NewBalance = @OldBalance - 300
11    WAITFOR DELAY '00:00:30:000'
12    UPDATE Accounting.BankAccounts
13    SET Balance = @NewBalance
14    WHERE AcctID = 1
15
16    SELECT @OldBalance AS OldBalance,
17    AcctID, AcctName, Balance
18    FROM Accounting.BankAccounts
19    WHERE AcctID = 1
20 COMMIT TRAN
```

OldBalance	AcctID	AcctName	Balance
500	1	John	200.00

```
1 -- SQL Server Concurrency
2 -- Lost Update - Session 2
3 USE TSQLE2012
4 GO
5 DECLARE @OldBalance int, @NewBalance int
6 BEGIN TRAN
7     SELECT @OldBalance = Balance
8     FROM Accounting.BankAccounts
9     WHERE AcctID = 1
10    SET @NewBalance = @OldBalance - 400
11
12    UPDATE Accounting.BankAccounts
13    SET Balance = @NewBalance
14    WHERE AcctID = 1
15
16    SELECT @OldBalance AS OldBalance,
17    AcctID, AcctName, Balance
18    FROM Accounting.BankAccounts
19    WHERE AcctID = 1
20 COMMIT TRAN
```

OldBalance	AcctID	AcctName	Balance
500	1	John	100.00

Uncommitted dependency (dirty read)

```
-- SQL Server Concurrency
-- Dirty Read - Session 1
USE TSQ2012
GO
SET TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED
BEGIN TRAN
    UPDATE Accounting.BankAccounts
    SET Balance -= 300
    WHERE AcctID = 1
    WAITFOR DELAY '00:00:10:000'
    ROLLBACK TRAN
    SELECT AcctID, AcctName, Balance
    FROM Accounting.BankAccounts
    WHERE AcctID = 1
```

Clean Read

AcctID	AcctName	Balance	ModifiedDate
1	John	500.00	2013-02-16

```
--SQL Server Concurrency
--Dirty Read - Session 2
USE TSQ2012
SET TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED
SELECT * FROM Accounting.BankAccounts
WHERE AcctID = 1
```

Dirty Read

AcctID	AcctName	Balance	ModifiedDate
1	John	200.00	2015-12-12

Inconsistent analysis (non-repeatable read)

```
1 --SQL Server Concurrency
2 --Repeatable Read - Session 1
3 USE TSQL2012
4 SET TRANSACTION ISOLATION LEVEL
5 READ COMMITTED --REPEATABLE READ
6 BEGIN TRAN
7     SELECT AcctID, ModifiedDate
8     FROM Accounting.BankAccounts
9     WAITFOR DELAY '00:00:30:000'
10    SELECT AcctID, ModifiedDate
11    FROM Accounting.BankAccounts
12 COMMIT TRAN
```

```
1 --SQL Server Concurrency
2 --Repeatable Read - Session 2
3 USE TSQL2012
4 BEGIN TRAN
5     UPDATE Accounting.BankAccounts
6     SET ModifiedDate = '01/05/2013'
7 COMMIT TRAN
```

READ COMMITTED

AcctID	ModifiedDate
1	2015-12-12
2	2015-12-12

AcctID	ModifiedDate
1	2013-01-05
2	2013-01-05

REPEATABLE READ

AcctID	ModifiedDate
1	2015-12-12
2	2015-12-12

AcctID	ModifiedDate
1	2015-12-12
2	2015-12-12

Phantom Reads

```
--SQL Server Concurrency
--Phantom Read - Session 1
USE TSQL2012
SET TRANSACTION ISOLATION LEVEL
READ COMMITTED
BEGIN TRAN
    SELECT AcctID, AcctName,
           Balance, ModifiedDate
    FROM Accounting.BankAccounts
    WAITFOR DELAY '00:00:10:000'
    SELECT AcctID, AcctName,
           Balance, ModifiedDate
    FROM Accounting.BankAccounts
COMMIT TRAN

--Phantom Read - Session 2
USE TSQL2012
BEGIN TRAN
    DELETE FROM Accounting.BankAccounts
    WHERE AcctID IN(3,5,6)
COMMIT TRAN
```

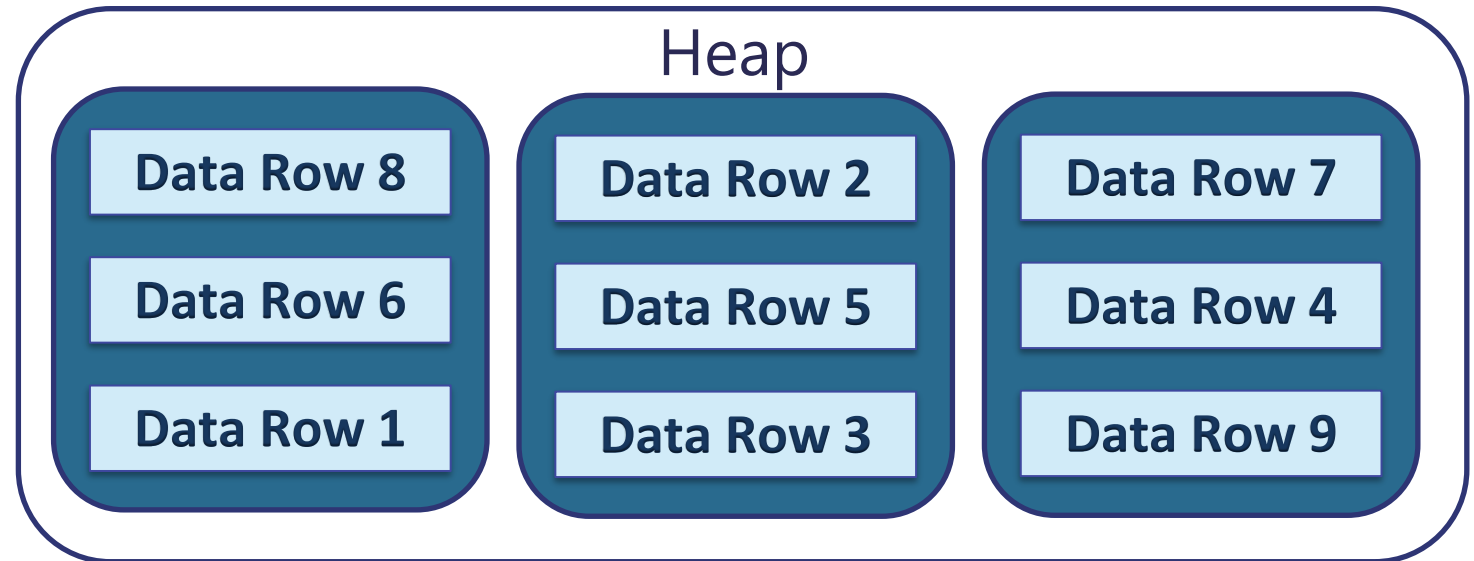
Missing records

AcctID	AcctName	Balance	ModifiedDate
1	John	500.00	2016-01-02
2	Armando	750.00	2016-01-02
3	Kelli	1250.00	2016-01-02
4	Jessica	1005.00	2016-01-02
5	Maddison	745.00	2016-01-02
6	Alicen	555.00	2016-01-02
7	Molly	790.00	2016-01-02
8	Amy	650.00	2016-01-02

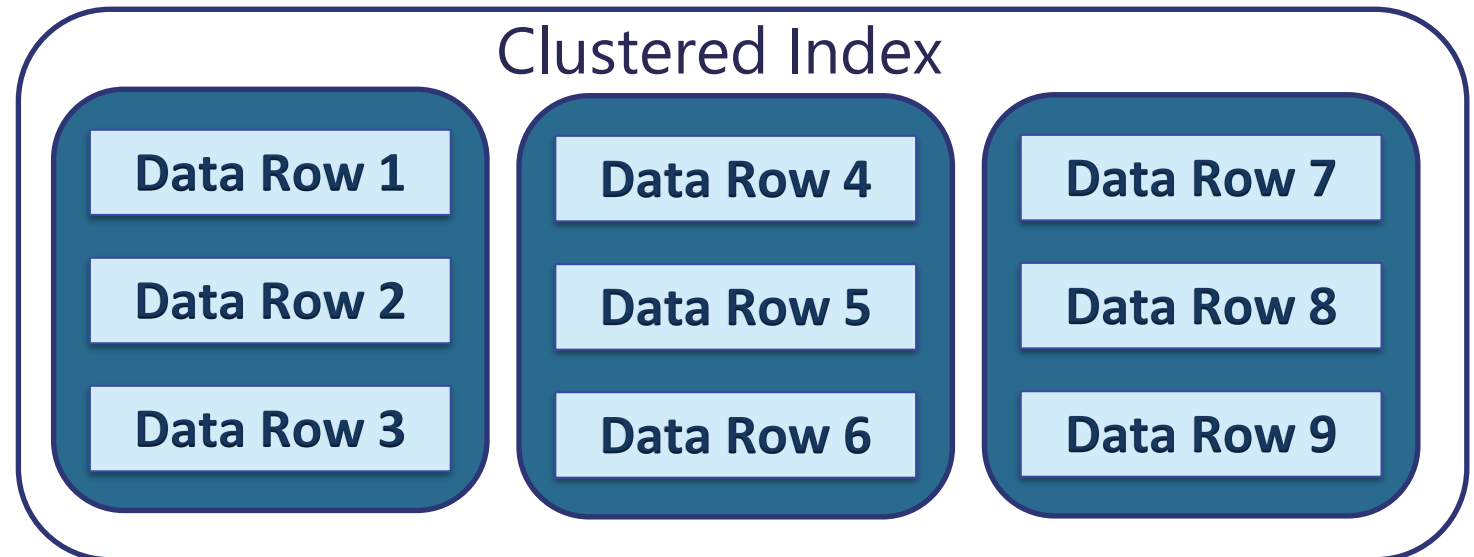
AcctID	AcctName	Balance	ModifiedDate
1	John	500.00	2016-01-02
2	Armando	750.00	2016-01-02
4	Jessica	1005.00	2016-01-02
7	Molly	790.00	2016-01-02
8	Amy	650.00	2016-01-02
9	Logan	1050.00	2016-01-02

How Data is Stored in Data Pages

Data stored in a Heap is not stored in any order and normally does not have a Primary Key.



Clustered Index data is stored in sorted order by the Clustering key. In many cases, this is the same value as the Primary Key.



Characteristics of a Good Clustering Key

Narrow

- Use a data type with a small number of bytes to conserve space in tables and indexes

Unique

- To avoid SQL adding a 4-byte uniquifier

Static

- Allows data to stay constant without constant changes which could lead to page splits

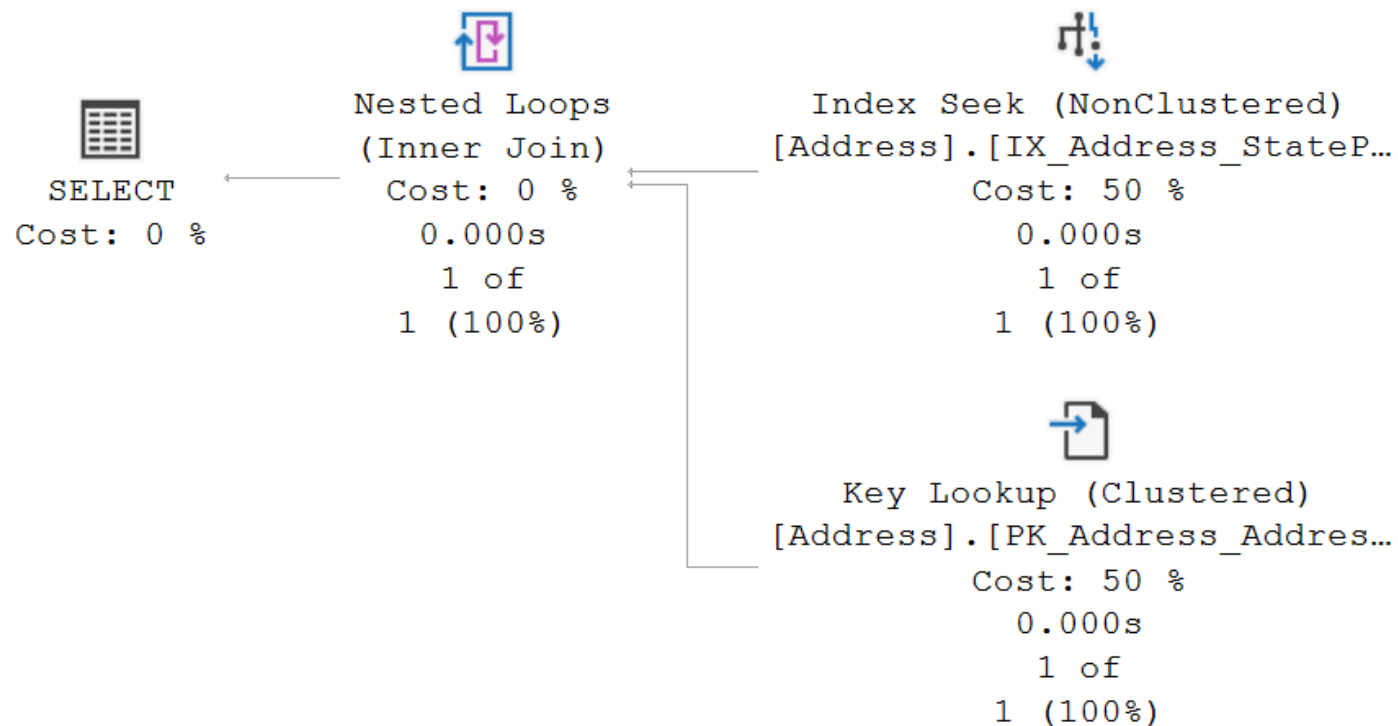
Increasing

- Allows better write performance and reduces fragmentation issues

Key Lookup

Query 1: Query cost (relative to the batch): 100%

```
SELECT [AddressID],[StateProvinceID],[City] FROM [Person].[Address] WHERE [StateProvinceID]=@1
```



Object

[AdventureWorks2016].[Person].[Address].
[IX_Address_StateProvinceID]

Output List

[AdventureWorks2016].[Person].[Address].AddressID,
[AdventureWorks2016].[Person].[Address].StateProvinceID

Object

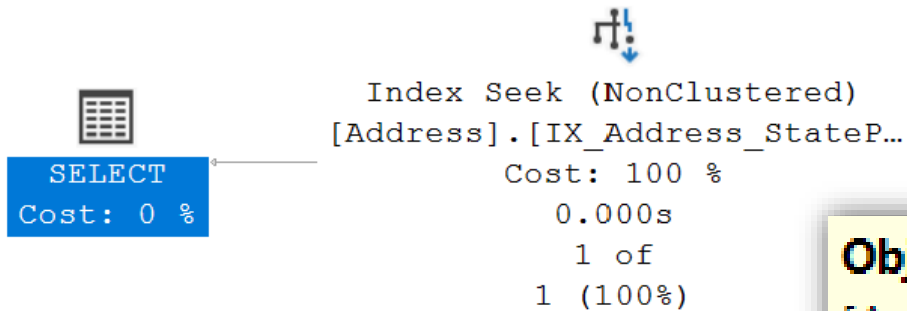
[AdventureWorks2016].[Person].[Address].
[PK_Address_AddressID]

Output List

[AdventureWorks2016].[Person].[Address].City

Non-Clustered Index with Included Column

Query 1: Query cost (relative to the batch): 100%
SELECT [AddressID],[StateProvinceID],[City] FROM [Person].[Address] WHERE [StateProvinceID]=@1



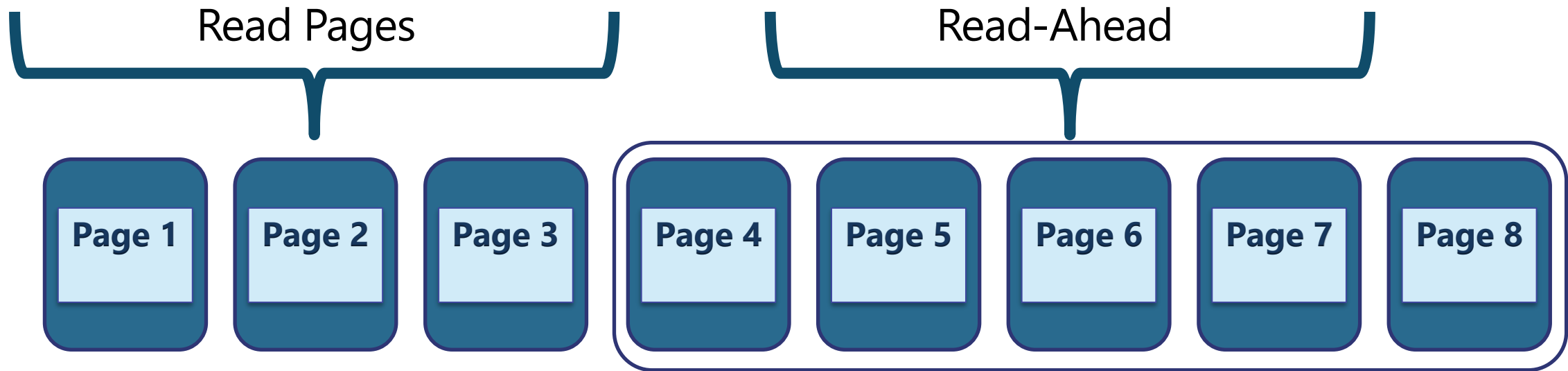
Object

[AdventureWorks2016].[Person].[Address].
[IX_Address_StateProvinceID]

Output List

[AdventureWorks2016].[Person].[Address].AddressID,
[AdventureWorks2016].[Person].[Address].City,
[AdventureWorks2016].[Person].[Address].StateProvinceID

Read-Ahead Scans



- Read-ahead anticipates the data and index pages needed to fulfill a query execution plan and brings the pages into the buffer cache before they are used by the query.
- The read-ahead mechanism allows the Database Engine to read up to 64 contiguous pages (512KB) from one file.

Columnstore Index Types

SQL Server 2012

- Only Non-Clustered, Non-Updatable Columnstore Indexes.
- Only available in Enterprise Edition.

SQL Server 2014

- Introduced Updatable, Clustered Columnstore Indexes
- Only available in Enterprise Edition.

SQL Server 2016

- Introduced Updatable, Non-Clustered Columnstore Indexes
- Available on Standard Edition. (Service Pack 1)

SQL Server 2019

- Online rebuilds for Clustered Columnstore Indexes.

Row Groups & Segments

Segment

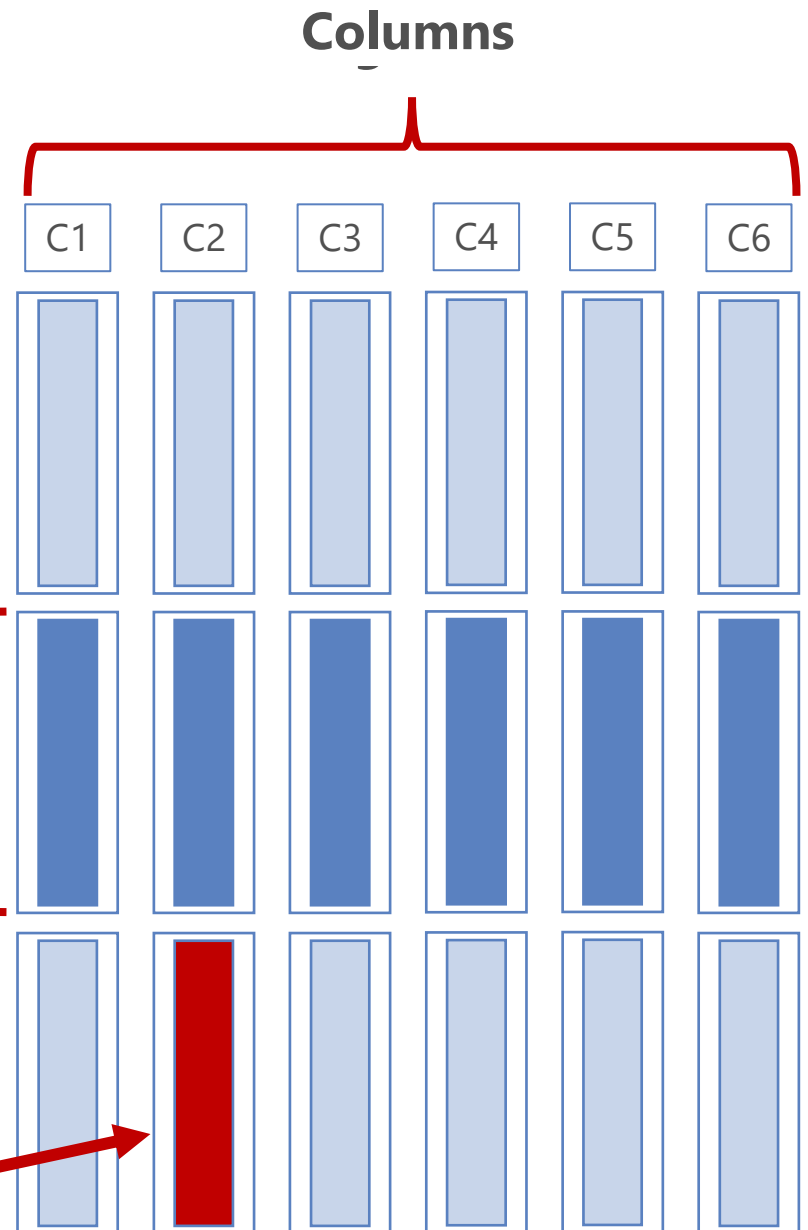
- Contains values for one column for a set of rows.
- Segments are compressed.
- Each segment is stored in a separate LOB.
- It is a unit of transfer between disk and memory.

Row Group

- Segments for the same set of rows comprise a row group.
- Position of a value in a column indicates to which row it belongs to.

Segment

Row
group



Showing Statistics

```
DBCC SHOW_STATISTICS ('Sales.SalesOrderDetail', 'IX_SalesOrderDetail_ProductID')  
WITH STAT_HEADER, HISTOGRAM
```

Results								
Messages								
Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	File
IX_SalesOrderDetail_ProductID	Nov 7 2012 6:44PM	121317	121317	200	0.0078125	12	NO	N
RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS				
730	0	288	0	1				
732	0	130	0	1				
738	154	600	2	77				
741	167	94	1	167				
742	0	288	0	1				

```
SELECT  
    ProductID, RecordCount = COUNT(*)  
FROM Sales.SalesOrderDetail  
WHERE  
    ProductID >= 732 AND  
    ProductID <= 738  
GROUP BY ProductID
```

Results	
Messages	
ProductID	RecordCount
732	130
733	44
736	110
738	600

Cardinality Estimator and Statistics

Statistics Properties - IX_SalesOrderDetail_ProductID

Select a page

- General
- Details
- Filter

Script Help

Table Name: SalesLT.SalesOrderDetail

Statistics Name: IX_SalesOrderDetail_ProductID

Statistics for INDEX 'IX_SalesOrderDetail_ProductID'.

Name	Updated
IX_SalesOrderDetail_ProductID	Aug 20 2019 1:09PM

All Density	Average Length
0.007042253	4
0.001845018	8
0.001845018	12

Histogram Steps	RANGE_HI_KEY	RANGE_ROWS
707		0
708		0
711		0
712		0
714		0
715		0
716		0
718		2
722		0
738		0
739		0
742		0
743		0
747		0
748		0

Server: jdsql-one.database.windows.net

Connection: johndeardurff

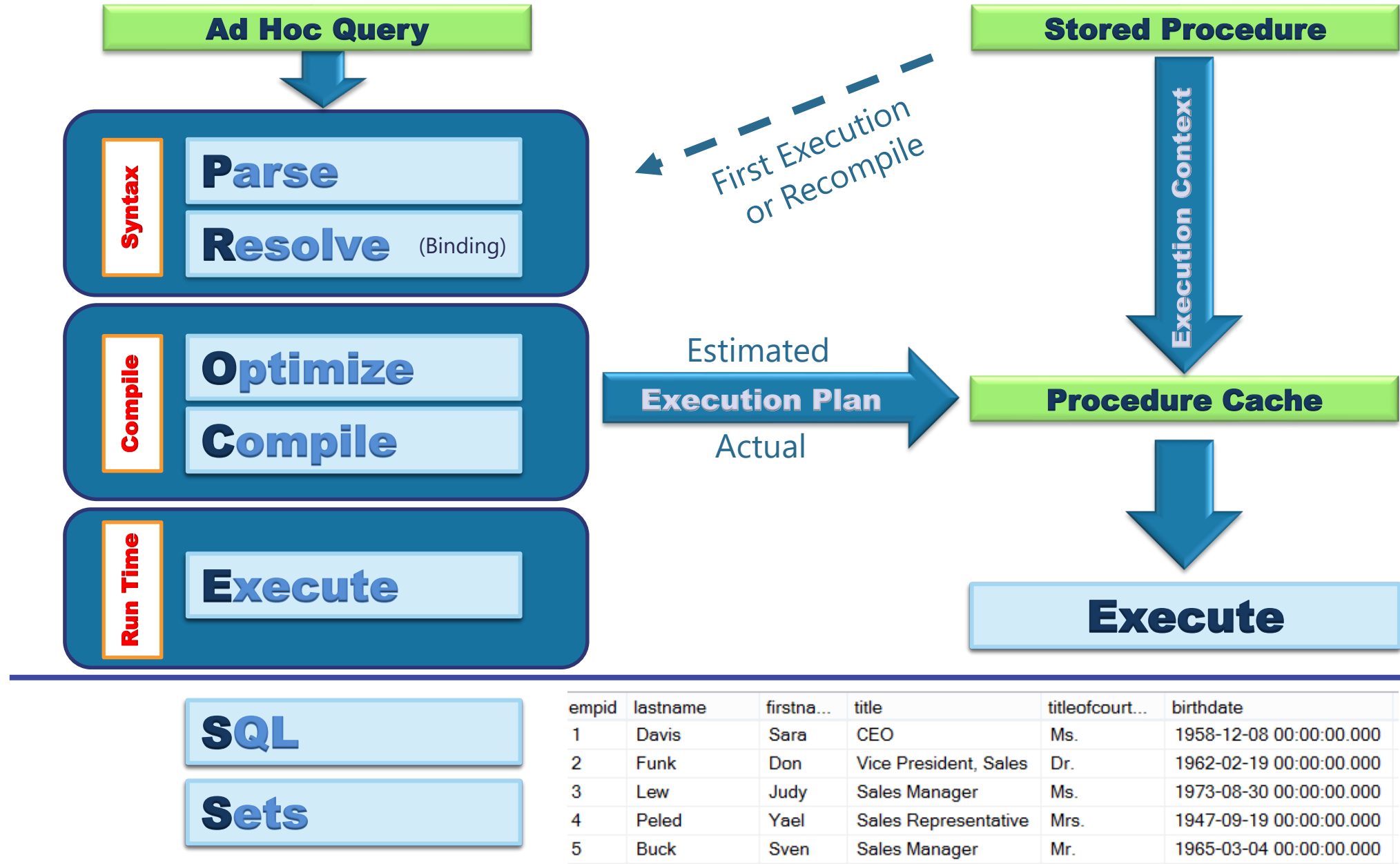
View connection properties

Progress

Ready

OK Cancel

How Queries are Processed



What does the binding step resolve?

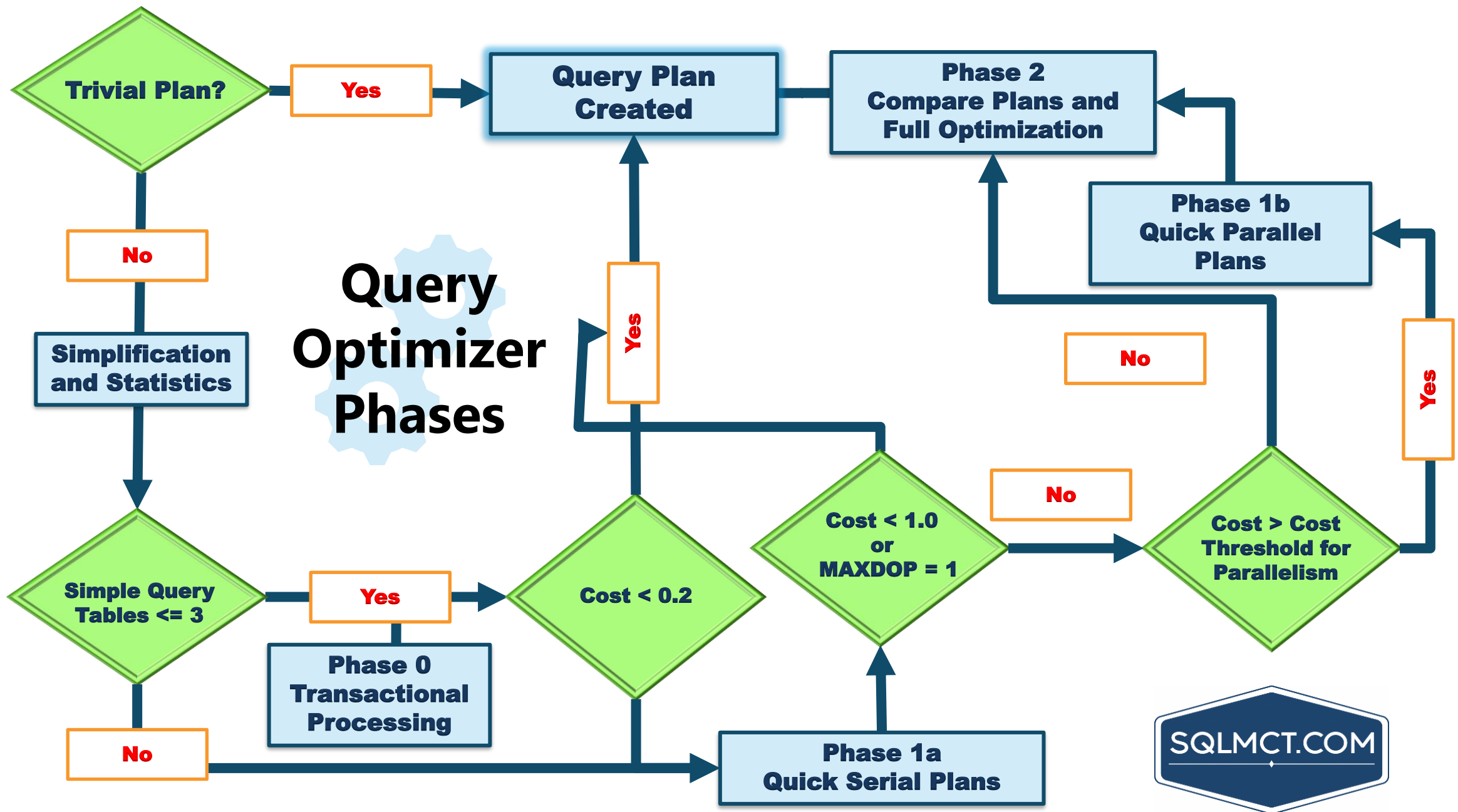
User permissions are checked.

Does a cached plan exist?

Object names (Tables, Views, Columns, etc.) to see if they exist.

Resolve aliases of columns and tables

Data types and if implicit data type conversions are needed.



Query Simplification phases

Constant Folding: Expressions with constant values are reduced

- **Quantity** = $2 + 3$ becomes **Quantity** = **5**
- **10** < **20** becomes **True**

Contradiction Detection: Removes criteria that doesn't match table constraints

- **Constraint:** Age > 18
- **Contradiction:** WHERE Age < 18

Domain Simplification: Reduces complex ranges to simple ranges

- **Complex range:** ID > 10 and ID < 20 or ID > 30 and < 50
- **Simplified range:** ID > 10 and < 50

Join Simplification: Removes redundant joins that are not necessary

Predicate Pushdown: Perform calculations only on rows returned

What is an Execution Plan?

```
SELECT SOH.SalesOrderID, SOH.CustomerID,
       OrderQty, UnitPrice, P.Name
FROM SalesLT.SalesOrderDetail
JOIN SalesLT.SalesOrderHeader
ON SOH.SalesOrderID =
JOIN SalesLT.Product
```

100 % ▾

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT SOH.SalesOrderID, SOH.CustomerID, OrderQty,

Query executed successfully.

ready

Clustered Index Seek (Clustered)	
Scanning a particular range of rows from a clustered index.	
Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.0243044 (37%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0243044
Estimated CPU Cost	0.0001756
Estimated Number of Executions	32
Estimated Number of Rows	16,9375
Estimated Number of Rows to be Read	16,9375
Estimated Row Size	21 B
Ordered	True
Node ID	4

Object
[AdventureWorksLT].[SalesLT].[SalesOrderDetail].

Output List
[AdventureWorksLT].[SalesLT].[SalesOrderDetail].OrderQty,
[AdventureWorksLT].[SalesLT].[SalesOrderDetail].ProductID,
[AdventureWorksLT].[SalesLT].[SalesOrderDetail].UnitPrice

Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorksLT].[SalesLT].
[SalesOrderDetail].SalesOrderID = Scalar Operator
([AdventureWorksLT].[SalesLT].[SalesOrderHeader].
[SalesOrderID] as [SOH].[SalesOrderID])

How to see the query plan

Text and XML

Command		Execute query?	Include estimated row counts & stats (Estimated Query Plan)	Include actual row counts & stats (Actual Query Plan)
Text Plan	SET SHOWPLAN_TEXT ON	No	No	No
	SET SHOWPLAN_ALL ON	No	Yes	No
	SET STATISTICS PROFILE ON	Yes	Yes	Yes
XML Plan	SET SHOWPLAN_XML ON	No	Yes	No
	SET STATISTICS PROFILE XML	Yes	Yes	Yes

How to see the query plan

Graphical execution plan

Estimated Execution Plan (Before Execution)

- The compiled plan.

Actual Execution Plan (After Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information available after the execution completes, such as execution warnings, or in newer versions of the Database Engine, the elapsed and CPU time used during execution.

Live Query Statistics (During Execution)

- The same as the compiled plan plus its execution context.
- This includes runtime information during execution progress and is updated every second. Runtime information includes for example the actual number of rows flowing through the operators.
- Enables rapid identification of potential bottlenecks.

What to look for in a query plan

Warnings

- Information about possible issues with the plan

Top Left Operator

- Overall properties of the plan

Expensive Operators

- Look from most expensive to least expensive

Data Flow Statistics

- Thicker arrows mean more data is being passed

Nested Loop Operator

- Possible to create index that covers query

Scans vs Seeks

- Not necessarily bad, but could indicate I/O issues

Skewed Estimates

- Statistics could be stale or invalid

Execution Plan Table Operators

Data stored in a Heap is not stored in any order and normally does not have a Primary Key.



Table Scan
[BankAccounts]
Cost: 100 %

Clustered Index data is stored in sorted order by the Clustering key. In many cases, this is the same value as the Primary Key.



Clustered Index Scan (Cluste...
[BankAccounts].[pk_acctID]
Cost: 100 %

Using a WHERE statement on an Index could possibly have the Execution Plan seek the Index instead of scan.

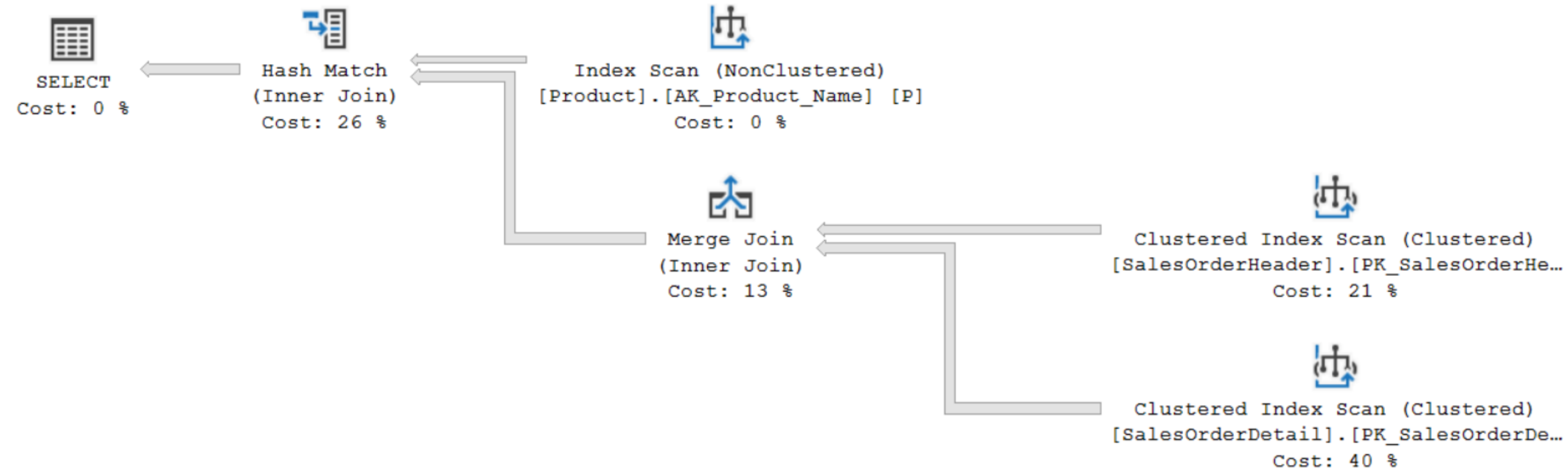


Clustered Index Seek (Cluste...
[BankAccounts].[pk_acctID]
Cost: 100 %

Execution Plan Join Operators (Code)

```
SELECT SOH.SalesOrderID, SOH.CustomerID,  
       OrderQty, UnitPrice, P.Name  
FROM Sales.SalesOrderHeader AS SOH  
     JOIN Sales.SalesOrderDetail AS SOD  
         ON SOH.SalesOrderID = SOD.SalesOrderID  
     JOIN Production.Product AS P  
         ON P.ProductID = SOD.ProductID
```

Execution Plan Join Operators (Plan)



Execution Plan Join Operators

A Merge Join is useful if both table inputs are in the same sorted order on the same value.



Merge Join
(Inner Join)
Cost: 39 %

A Hash Match is used when the tables being joined are not in the same sorted order.



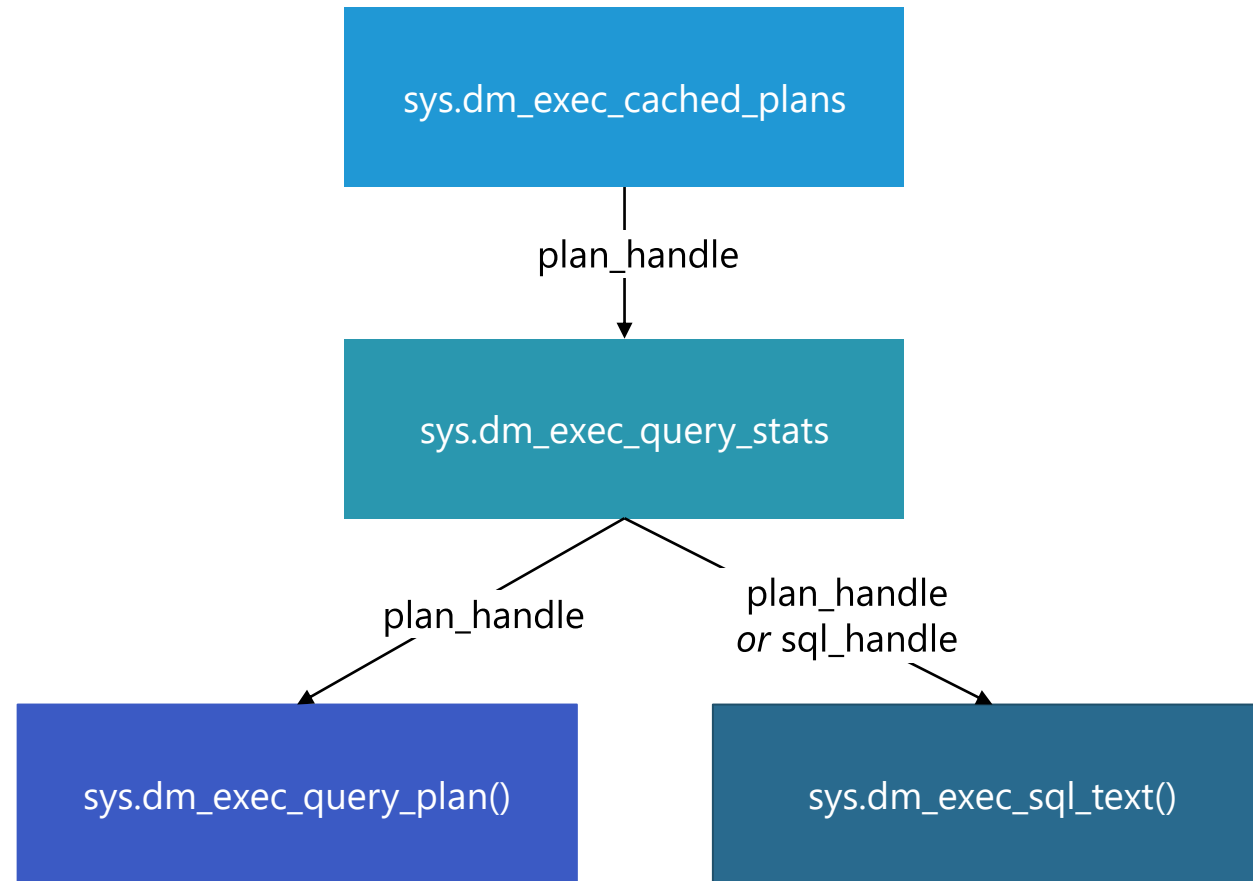
Hash Match
(Inner Join)
Cost: 47 %

A Nested Loop is used when a small (outer) table is used to lookup a value in a larger (inner) table.

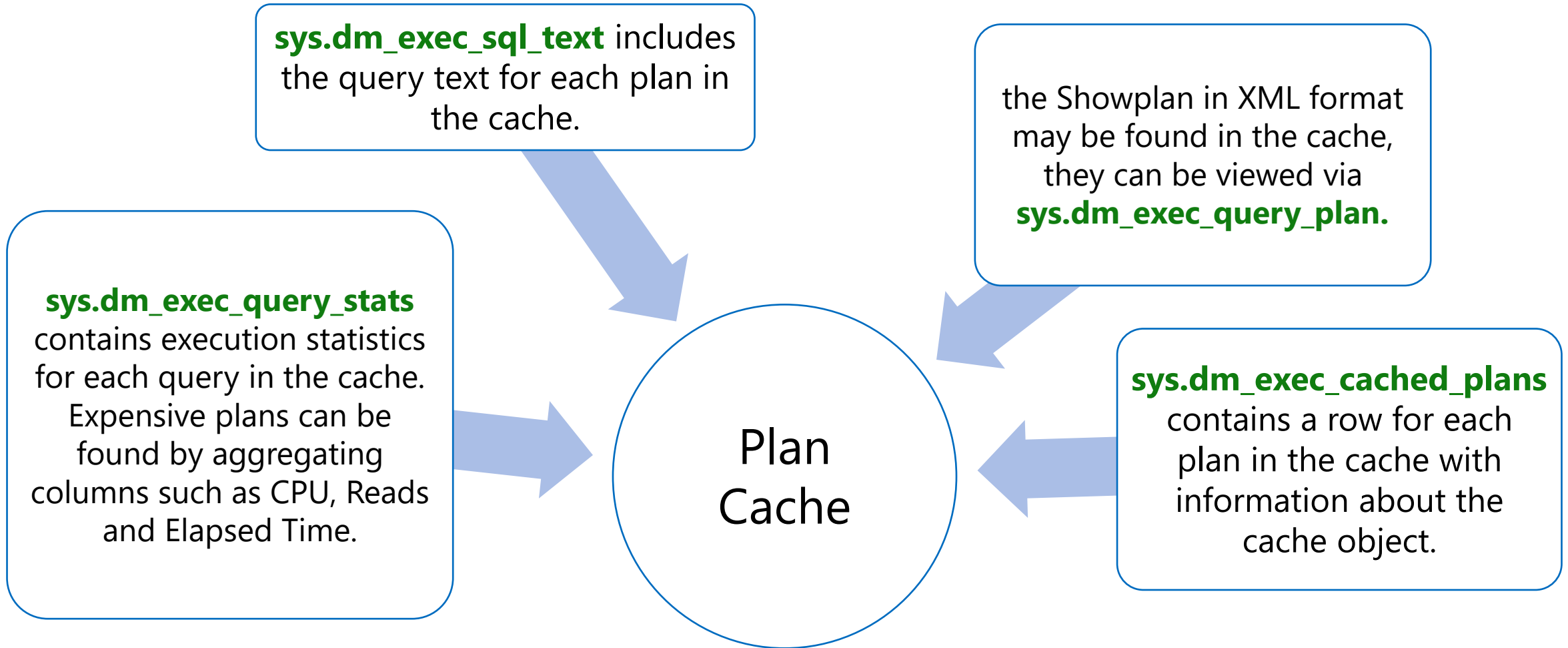


Nested Loops
(Inner Join)
Cost: 3 %

Relationships between DMOs



Queries in the Plan Cache

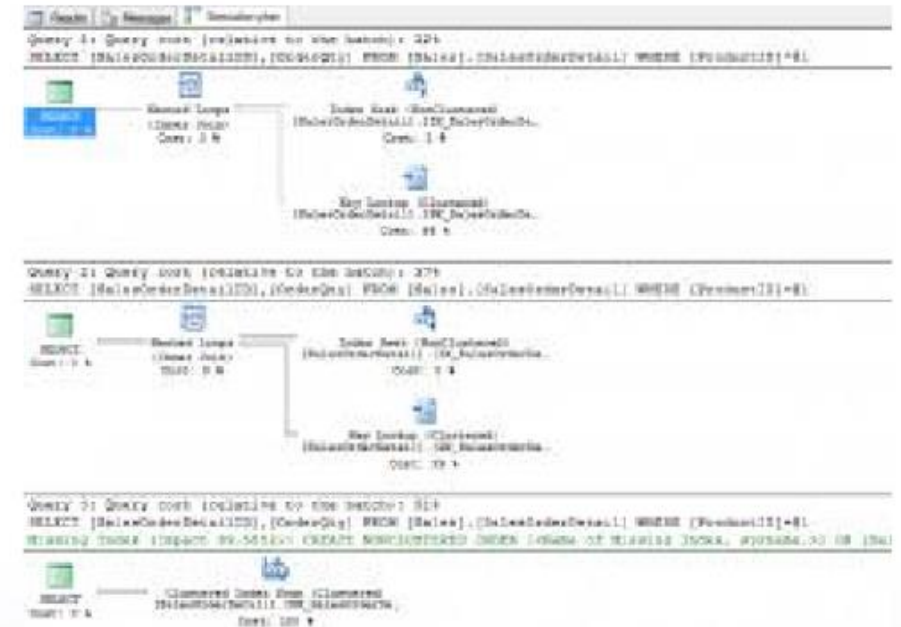


Parameter Sniffing

```

1 SELECT SalesOrderDetailID, OrderQty
2 FROM Sales.SalesOrderDetail
3 WHERE ProductID = 897
4
5 SELECT SalesOrderDetailID, OrderQty
6 FROM Sales.SalesOrderDetail
7 WHERE ProductID = 945
8
9 SELECT SalesOrderDetailID, OrderQty
10 FROM Sales.SalesOrderDetail
11 WHERE ProductID = 870

```



```

1 CREATE PROCEDURE Get_OrderQuantity
2 (@ProductID int)
3 AS
4 SELECT SalesOrderDetailID, OrderQty
5 FROM Sales.SalesOrderDetail
6 WHERE ProductID = @ProductID

```

