Microsoft

SQL Server Query Execution and Plans

Module 6

Learning Units covered in this Module

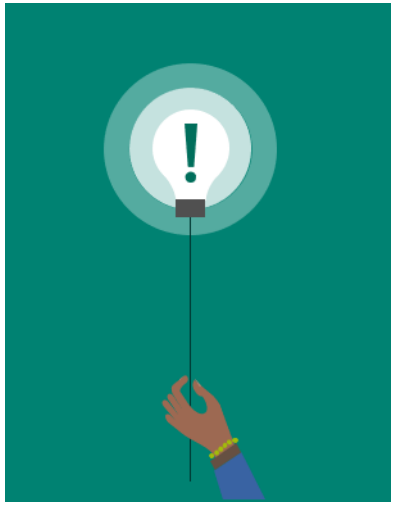
- Lesson 1: SQL Server Query Execution
- Lesson 2: SQL Server Query Plan Analysis

Lesson 1: SQL Server Query Execution

Objectives

After completing this learning, you will be able to:

- Explain Query Compilation and Optimization Process.
- Explain Query Execution Process.
- Explain Recompilation causes.



SQL Server Execution Plan

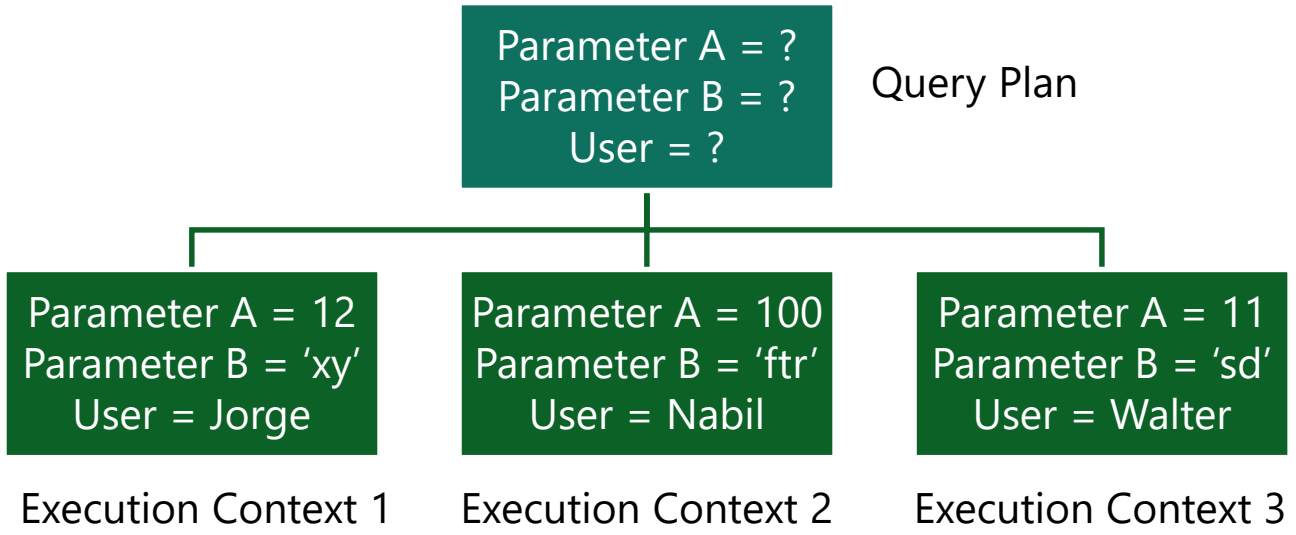
Main components

Compiled Plan (or Query Plan)

Compilation produces a query plan, which is a read-only data structure used by any number of users.

Execution Context

A data structure used to hold information specific to a query execution, such as parameter values.



SQL Server Execution Plan Caching

Overview

Part of the memory pool used to store execution plans – also known as plan cache.

The plan cache has two stores for all compiled plans:

The **Object Plans** cache store (OBJCP)

used for plans related to persisted objects (stored procedures, functions, and triggers).

The **SQL Plans** cache store (SQLCP)

used for plans related to autoparameterized, dynamic, or prepared queries.

SQL Server compilation and execution

Concepts

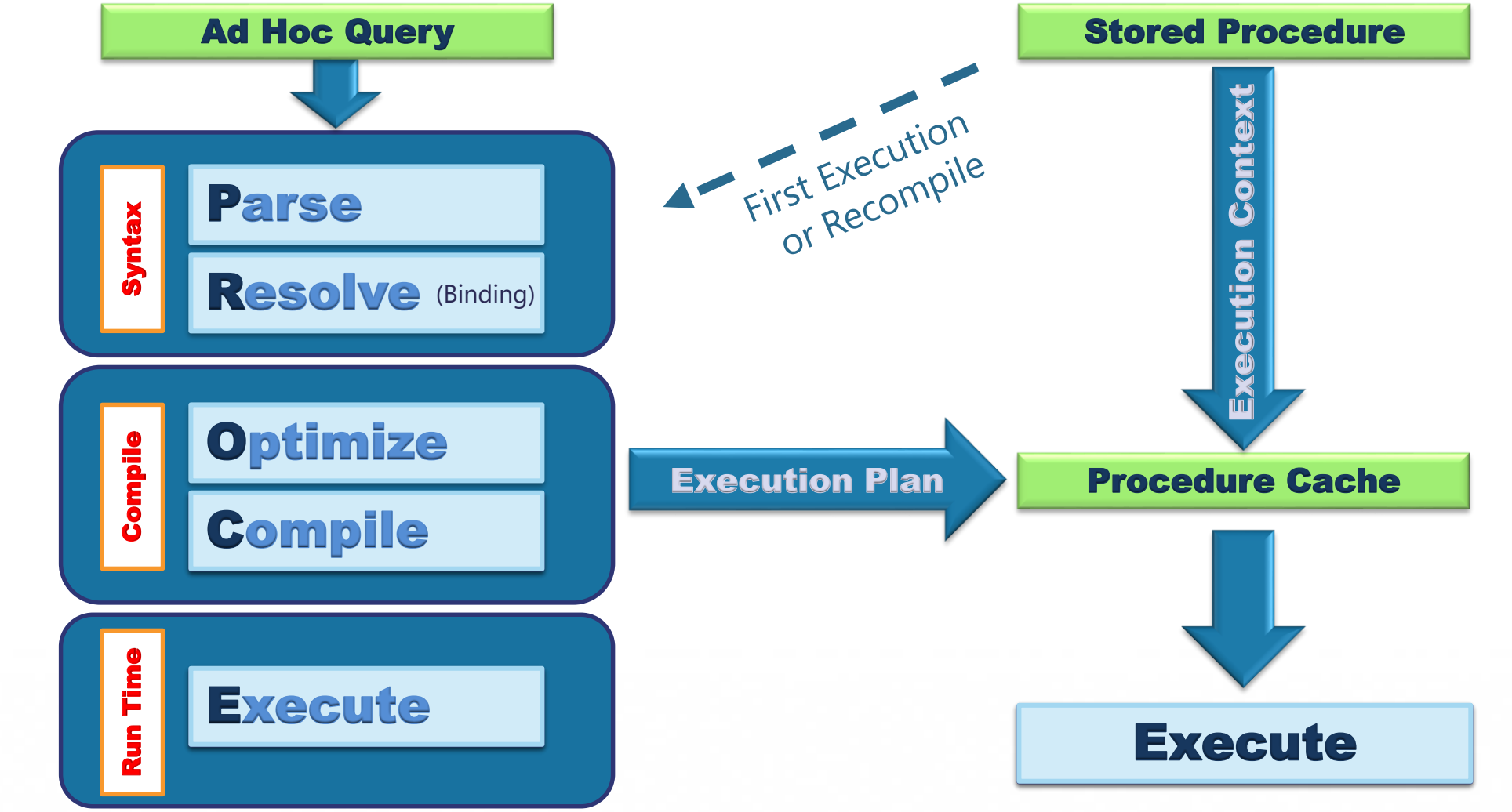
Compilation

Process of creating a good enough query execution plan, as quickly as possible for a query batch.

Refer to both the compilation of non-DML constructs in SQL statements (control flow, DDL, etc.) and the process of Query Optimization.

Query Execution

Process of executing the plan that is created during query compilation and optimization.



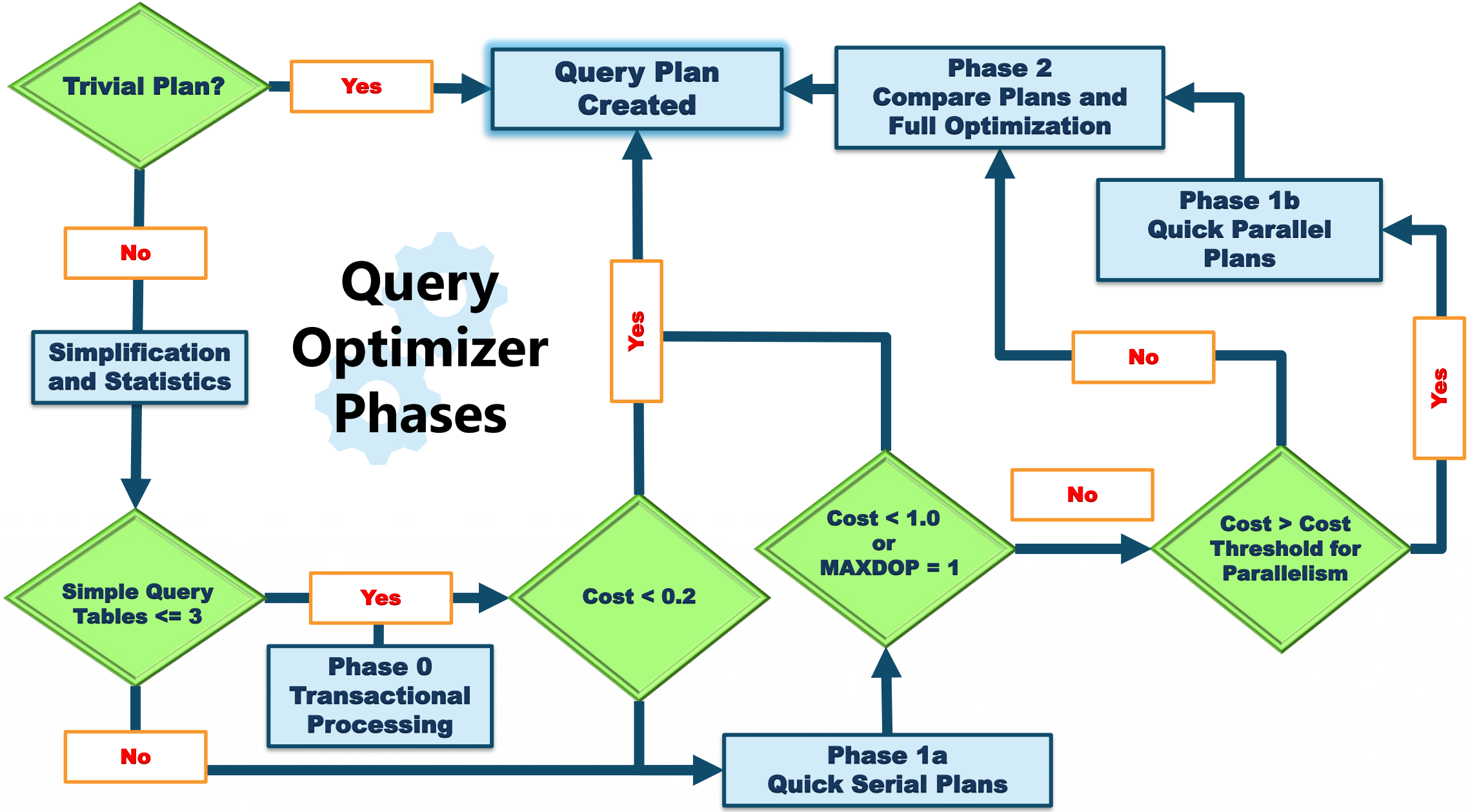
SQL

Sets

empid	lastname	firstna...	title	titleofcourt...	birthdate
1	Davis	Sara	CEO	Ms.	1958-12-08 00:00:00.000
2	Funk	Don	Vice President, Sales	Dr.	1962-02-19 00:00:00.000
3	Lew	Judy	Sales Manager	Ms.	1973-08-30 00:00:00.000
4	Peled	Yael	Sales Representative	Mrs.	1947-09-19 00:00:00.000
5	Buck	Sven	Sales Manager	Mr.	1965-03-04 00:00:00.000

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 Optimizer Phases
Trivial Plan?
Yes
Query Plan Created
Phase 2 Compare Plans and Full Optimization
Phase 1b Quick Parallel Plans
No
Simplification and Statistics
Yes
No
Yes
Simple Query Tables <= 3
Yes
Phase 0 Transactional Processing
Cost < 0.2
Cost < 1.0 or MAXDOP = 1
No
Cost > Cost Threshold for Parallelism
No
Phase 1a Quick Serial Plans

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

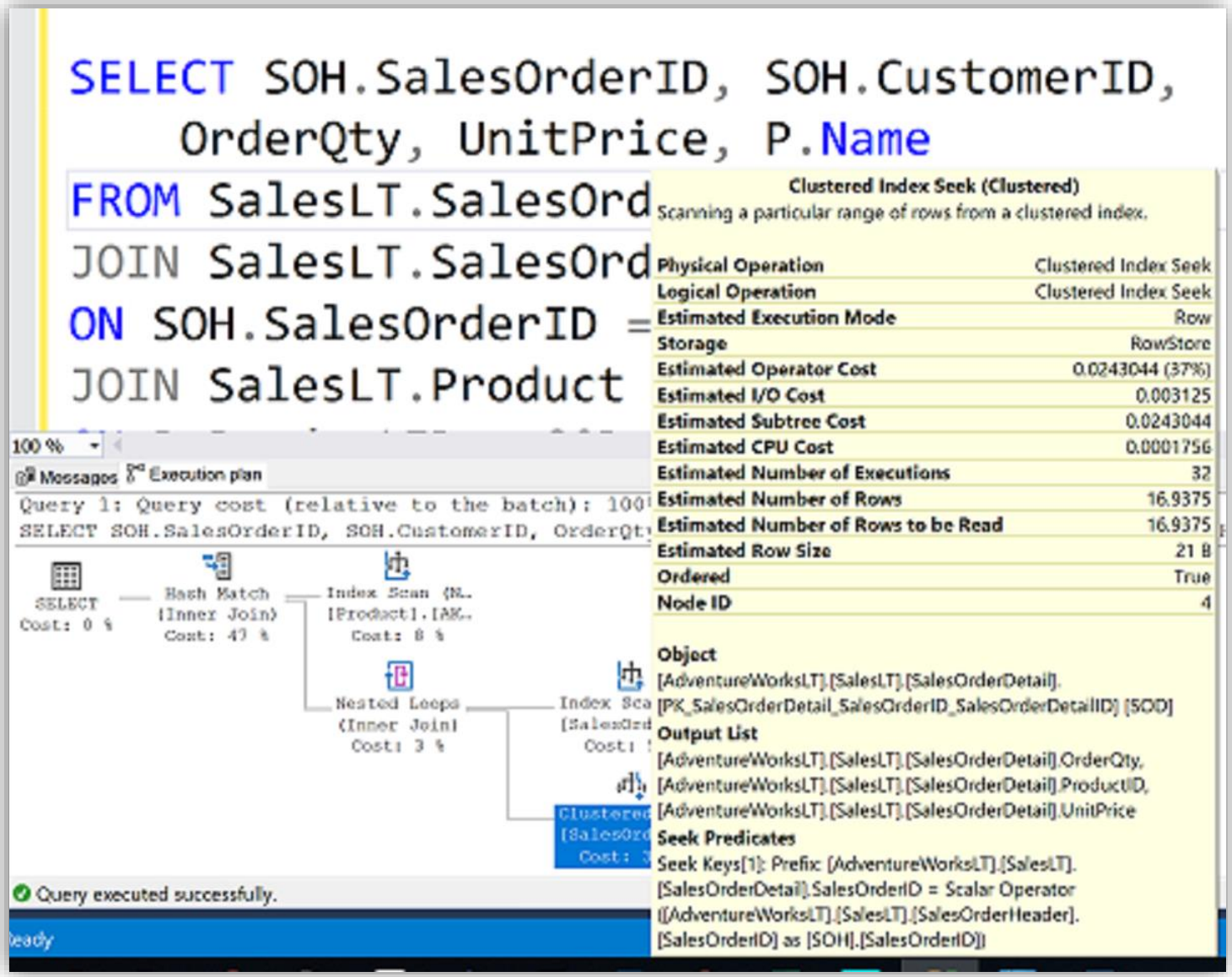
Domain Simplification: Reduces complex ranges to simple ranges

- **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?



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.

SQL Server Execution Plan

Execution modes

Row mode execution

- Efficient for OLTP scenarios
- Used with traditional tables, where data is stored in row-wise format.
- Operators read all columns from qualifying rows based on predicate, and for each row it retrieves columns needed for the result set

Batch mode execution

- Efficient for Data Warehousing scenarios
- Used to process multiple rows together
- Closely integrated with, and optimized around the ColumnStore storage format
- Operators read only the columns required for the result, from group of rows together.
- Starting SQL Server 2019, Batch mode does not require a ColumnStore index as in the previous versions.

SQL Server Execution Plan Recompilations

Overview

Most recompilations are required either for statement correctness or to obtain potentially faster query execution plan.

The engine detects changes that invalidate execution plan(s) and marks those as not valid. New plan must be recompiled for the next query execution.

Starting with SQL Server 2005, whenever a statement within a batch causes recompilation, only the statement inside the batch that triggers recompilation is recompiled.

SQL Server Execution Plan Recompilations

Recompilation reasons

Table / Index Changes

- Changes made to objects referenced by the query (ALTER TABLE and ALTER VIEW).
- Changing or dropping any indexes used by the execution plan.

Stored Procedures

- Changes made to a single procedure, which would drop all plans for that procedure from the cache (ALTER PROCEDURE).
- Explicit call to sp_recompile.
- Executing a stored procedure using the WITH RECOMPILE option.

Data Volume

- Updates on statistics used by the execution plan
- For tables with triggers, if the number of rows in the inserted or deleted tables grows significantly.

Other

- Large numbers of changes to keys (generated by statements from other users that modify a table referenced by the query).
- Temporary table changes

Questions?

Knowledge Check

What is meant by SQL Server's query optimizer being **cost-based**?

When is a query considered for a parallel execution plan?

Will SQL Server evaluate **every** possible query plan in the process of optimization? Why?

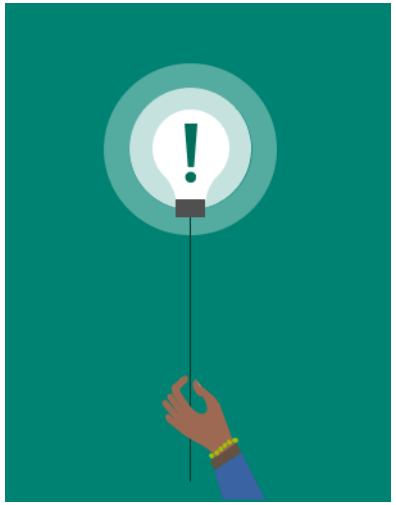
Name two recompilation causes.

Lesson 2: SQL Server Query Plan Analysis

Objectives

After completing this learning, you will be able to:

- Read execution plans.
- Understand logical and physical join operators.
- Describe data access.



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.

Contents of an Execution Plan

- Sequence in which the source tables are accessed.
- Methods used to extract data from each table.
- How data is joined
- Use of temporary worktables and sorts
- Estimated rowcount, iterations, and costs from each operator
- Actual rowcount and iterations

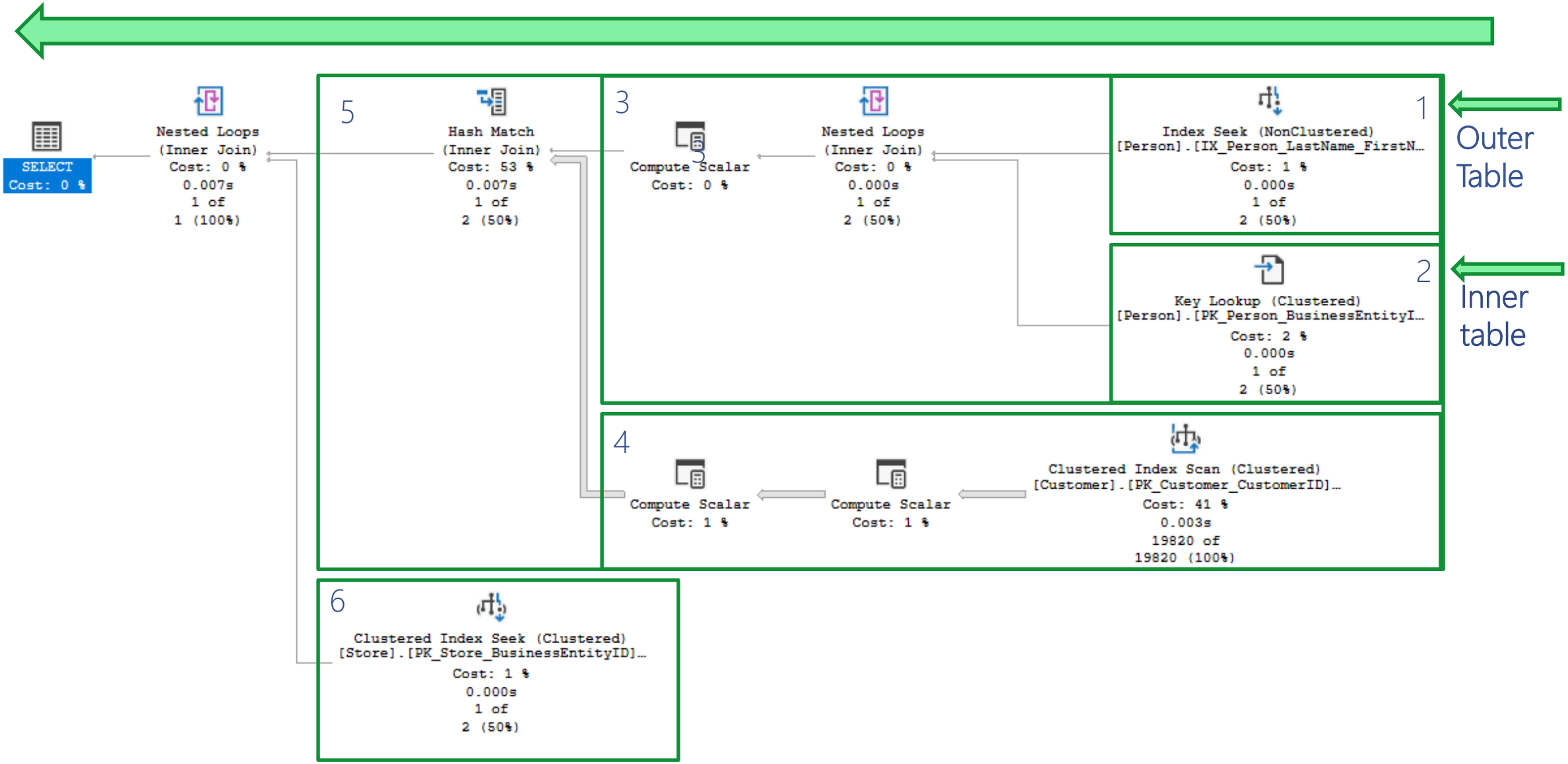
How to see the query plan

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

SSMS Graphical Plan

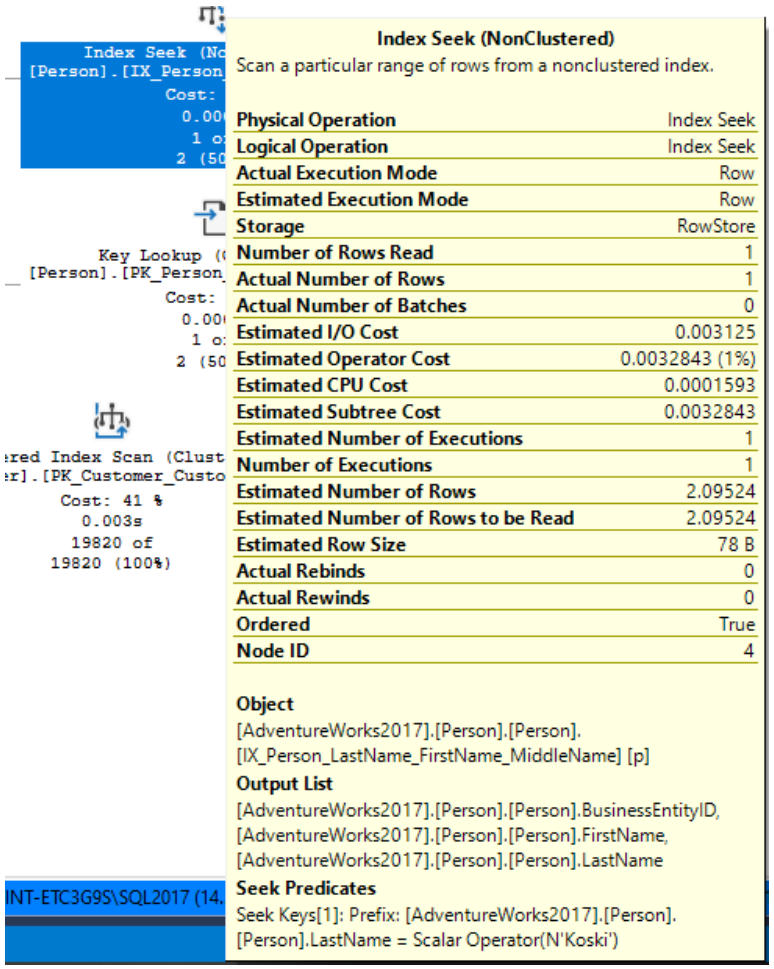
Execution Flow



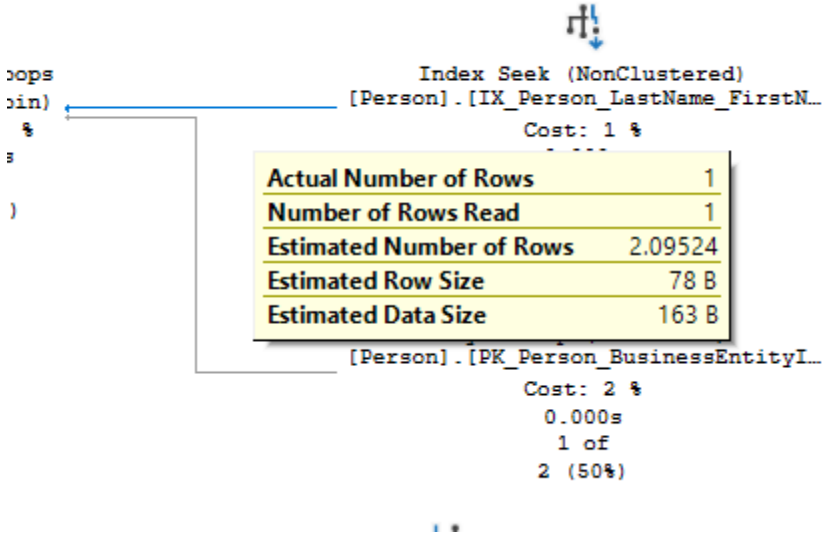
SSMS Graphical Plan

Data flow between the operators and statistical data of each operator

Statistical data for the operator

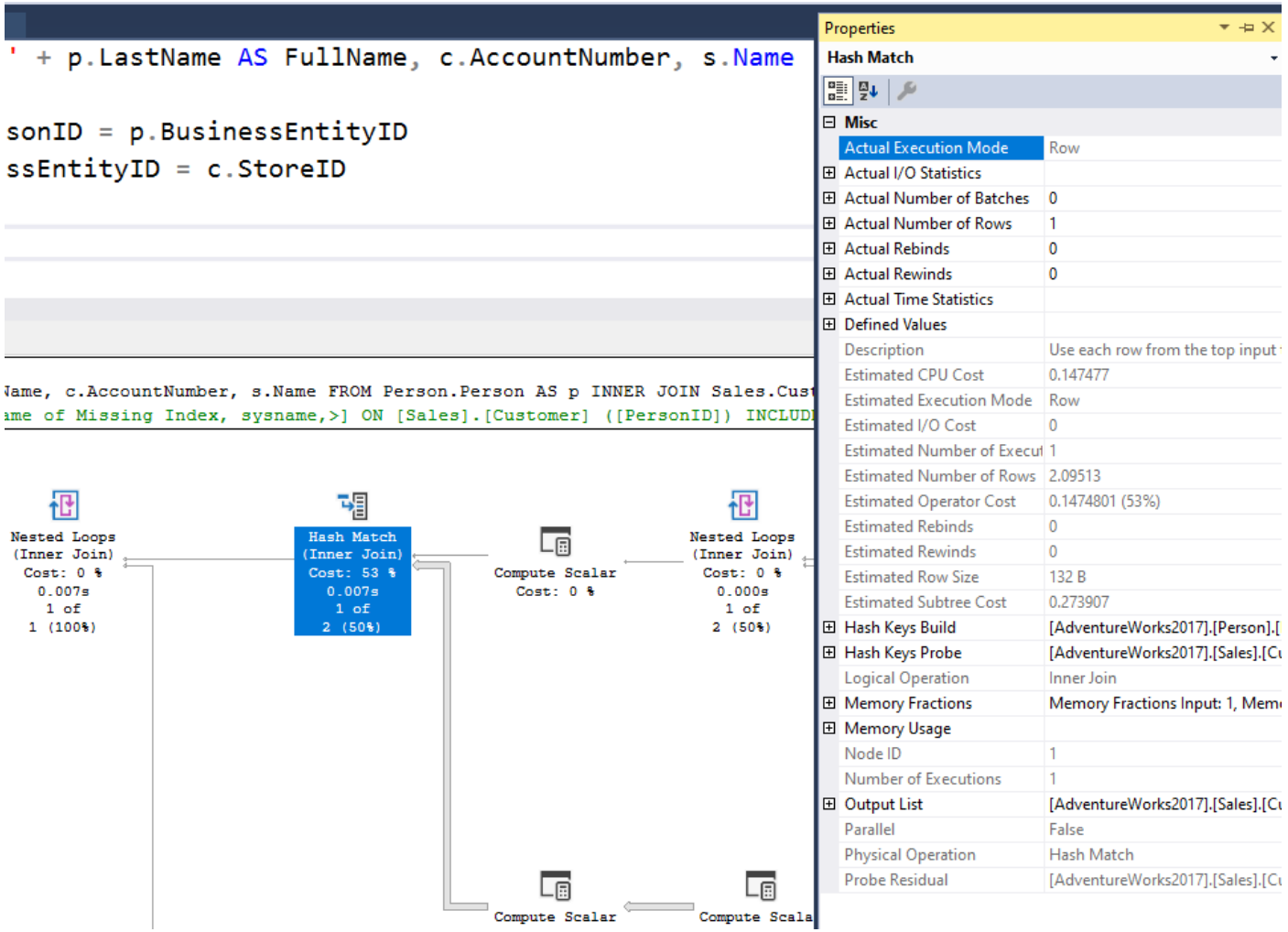


Data flow statistics



SSMS Graphical Plan

Properties sheet



Management Studio Properties sheet includes even more detailed information about each operator and about the overall query plan.

Use the most recent version of Management Studio as every new version display more detailed information about the Query Plan when examining the plan in graphical mode.

SSMS Graphical Plan

Live Query Statistics

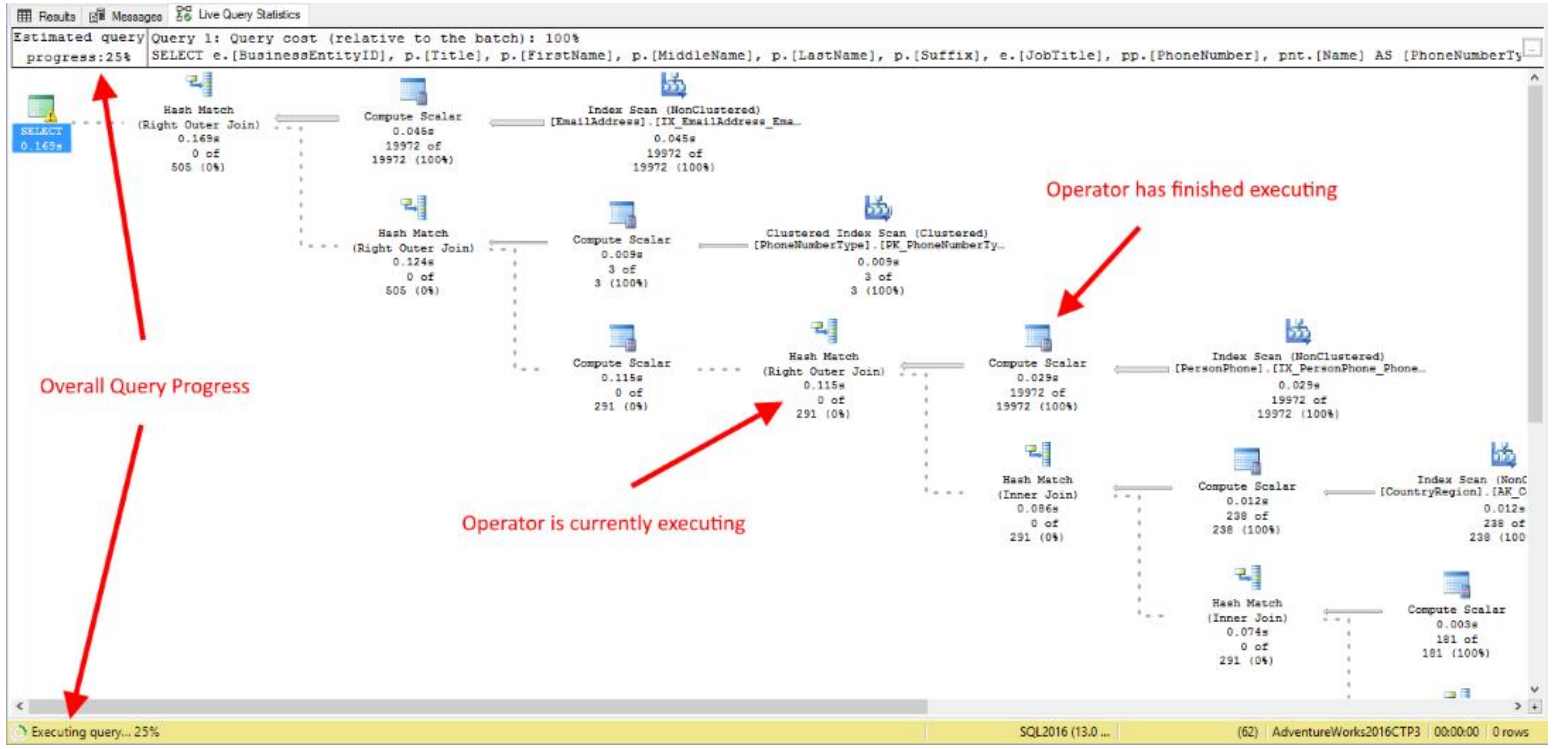
View CPU/memory usage, execution time, and query progress

Allows drill down to live operator-level statistics, such as:

- Number of generated rows
- Elapsed time
- Operator progress
- Live warnings

This feature is primarily intended for troubleshooting purposes.

Using this feature can moderately slow the overall query performance.



Execution Plan

Notable operators

Operators describe how SQL Server executes a query. The query optimizer uses operators to build a query plan to create the result specified in the query.


Table scan


Clustered index scan


Clustered index seek


RID lookup


Key lookup


ColumnStore Index Scan

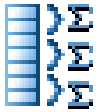

Nonclustered index scan


Nonclustered index seek


Sort


Table spool


Index spool


Stream Aggregation

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 SalesLT.SalesOrderHeader as SOH
JOIN SalesLT.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID
JOIN SalesLT.Product AS P
ON P.ProductID = SOD.ProductID
```

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

Nested Loops
(Inner Join)
Cost: 3 %

What to look for in the query plan

Warnings	• Information about possible issues with the plan
Left Most Operator	• Overall properties of the plan to establish baseline
Right to Left	• Solve issues early in the plan
Expensive Operators	• Look from most expensive to least expensive
Sort Operators	• Locate why there is a sort operation and is it needed
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

Demonstration

Query Plan Analysis

- Use the graphical execution plan and IO statistics to tune a query.
- Explore Live Query Statistics.

Query Plan Analysis

- Identifying and tuning high cost operators in the query plan

Questions?

Knowledge Check

What are the physical join operators?

What is a method to eliminate a lookup?

Is it recommended to eliminate all lookups operators?

Under what circumstances would a table scan be more efficient than an index seek on a non-clustered, non-covering index?

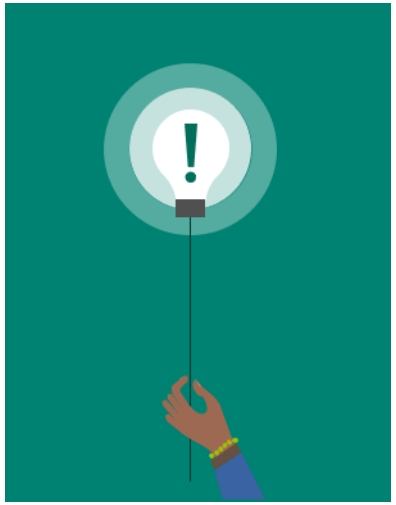
Is a Clustered Index Scan more efficient than a Table Scan?

Lesson 3: SQL Server Intelligent Query Processing

Objectives

After completing this learning, you will be able to:

- Understand the Intelligent query processing features.
- Enable/disable Intelligent query processing features.



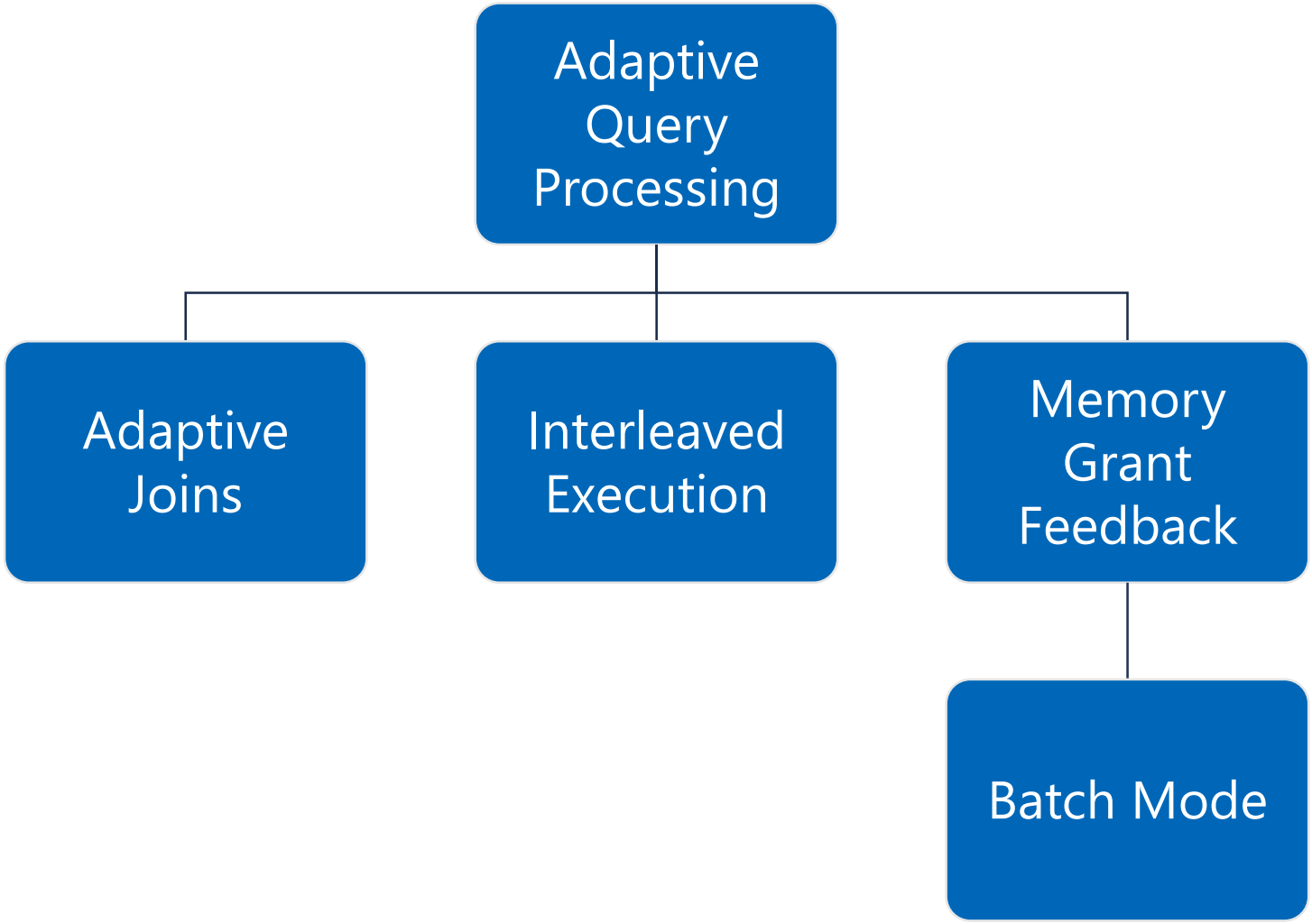
A History of Intelligent Query Processing

Adaptive Query Processing (2017)

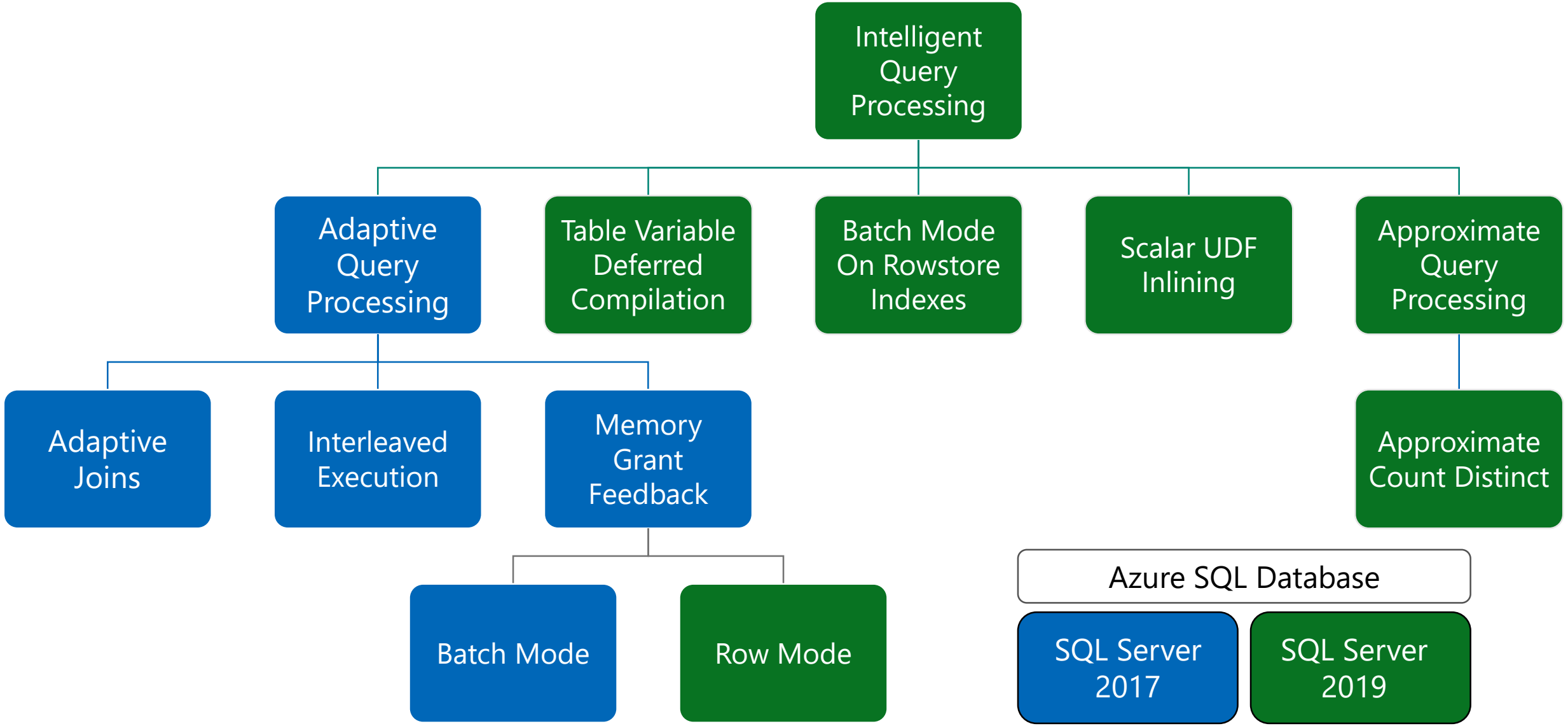
Intelligent Query Processing (2019)

New Features of IQP (2022)

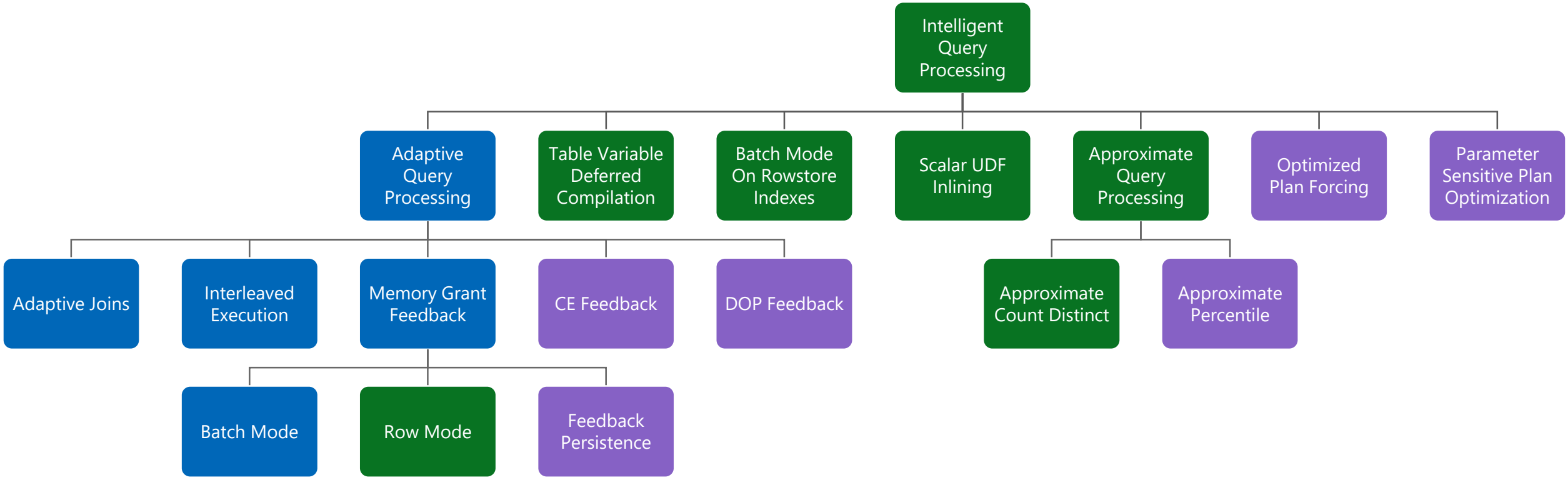
Adaptive Query Processing (2017)



Intelligent Query Processing (2019)



Intelligent Query Processing (2022)



http://aka.ms/IQP

Azure SQL Database

2017 2019 2022

Enabling and Disabling – Instance Level

For SQL Server 2017 Features	• Enabled by default in Compatibility level 140 or higher • To disable change compatibility level to 130 or lower
For SQL Server 2019 Features	• Enabled by default in Compatibility level 150 or higher • To disable change compatibility level to 140 or lower
For SQL Server 2022 Features	• Enabled by default in Compatibility level 160 or higher • To disable change compatibility level to 150 or lower

Enabling and Disabling – Database Level

Different settings for 2017 vs Azure SQL, SQL Server 2019 and higher

```
ALTER DATABASE SCOPED CONFIGURATION SET DISABLE_BATCH_MODE_ADAPTIVE_JOINS = ON|OFF;
```

```
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_ADAPTIVE_JOINS = ON|OFF;
```

To get a list of Database Scoped Configuration settings

```
SELECT * From sys.database_scoped_configurations;
```

configuration_id	name	value
7	INTERLEAVED_EXECUTION_TVF	1
8	BATCH_MODE_MEMORY_GRANT_FEEDBACK	1
9	BATCH_MODE_ADAPTIVE_JOINS	1
10	TSQL_SCALAR_UDF_INLINING	1
16	ROW_MODE_MEMORY_GRANT_FEEDBACK	1
18	BATCH_MODE_ON_ROWSTORE	1
19	DEFERRED_COMPILATION_TV	1
28	PARAMETER_SENSITIVE_PLAN_OPTIMIZATION	1
31	CE_FEEDBACK	1
33	MEMORY_GRANT_FEEDBACK_PERSISTENCE	1
34	MEMORY_GRANT_FEEDBACK_PERCENTILE_GRANT	1
35	OPTIMIZED_PLAN_FORCING	0

Enabling and Disabling – Statement Level

You can disable features at the statement scope if necessary.

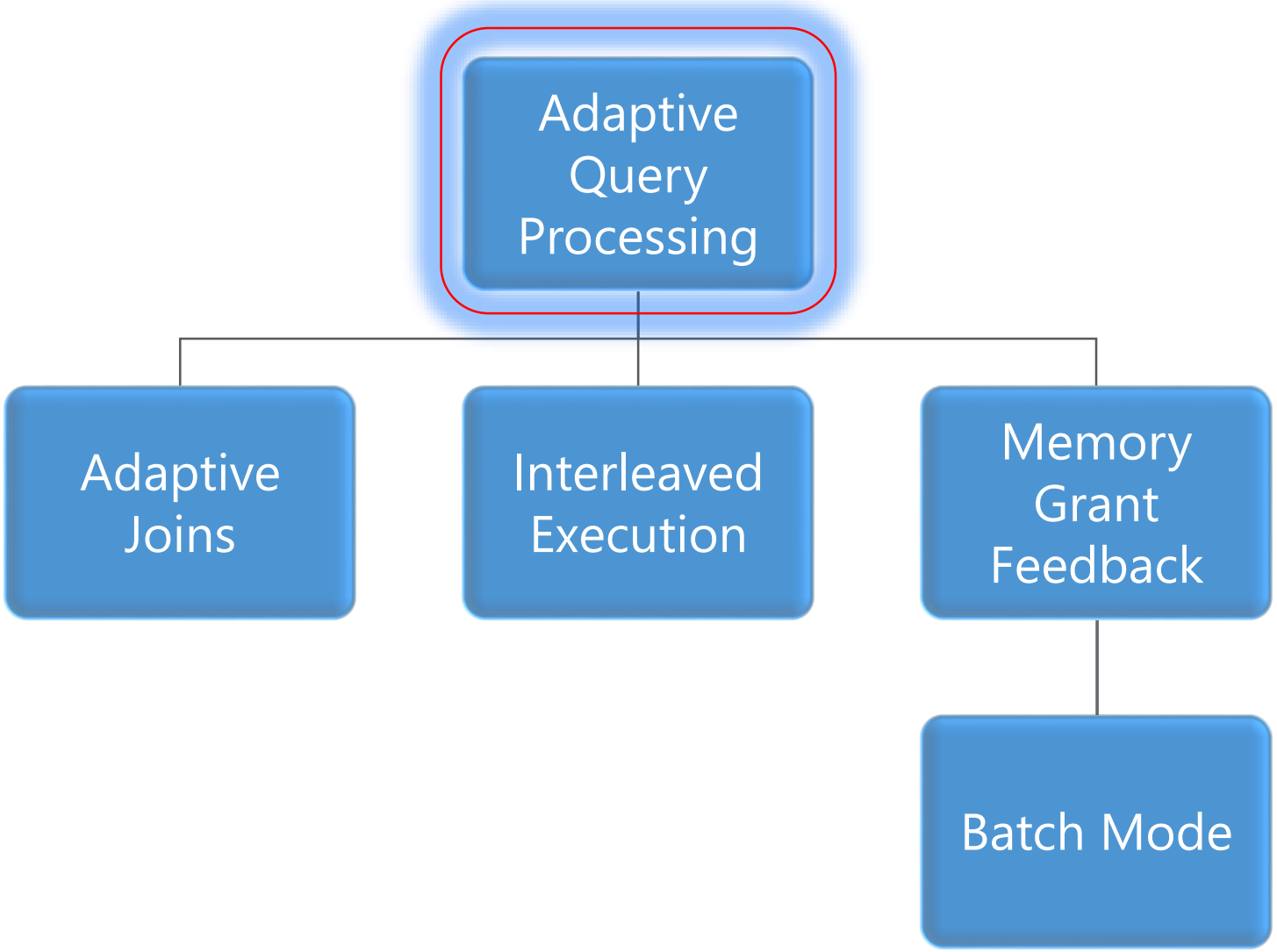
```
<statement>
OPTION (USE HINT('DISABLE_BATCH_MODE_ADAPTIVE_JOINS'));
```

To get a list of valid query use hints

```
SELECT * FROM sys.dm_exec_valid_use_hints;
```

name
DISABLE_INTERLEAVED_EXECUTION_TVF
DISABLE_BATCH_MODE_MEMORY_GRANT_FEEDBACK
DISABLE_BATCH_MODE_ADAPTIVE_JOINS
DISABLE_ROW_MODE_MEMORY_GRANT_FEEDBACK
DISABLE_DEFERRED_COMPILATION_TV
DISABLE_TSQL_SCALAR_UDF_INLINING
ASSUME_FULL_INDEPENDENCE_FOR_FILTER_ESTIMATES
ASSUME_PARTIAL_CORRELATION_FOR_FILTER_ESTIMATES
DISABLE_CE_FEEDBACK
DISABLE_MEMORY_GRANT_FEEDBACK_PERSISTENCE
DISABLE_DOP_FEEDBACK
DISABLE_OPTIMIZED_PLAN_FORCING

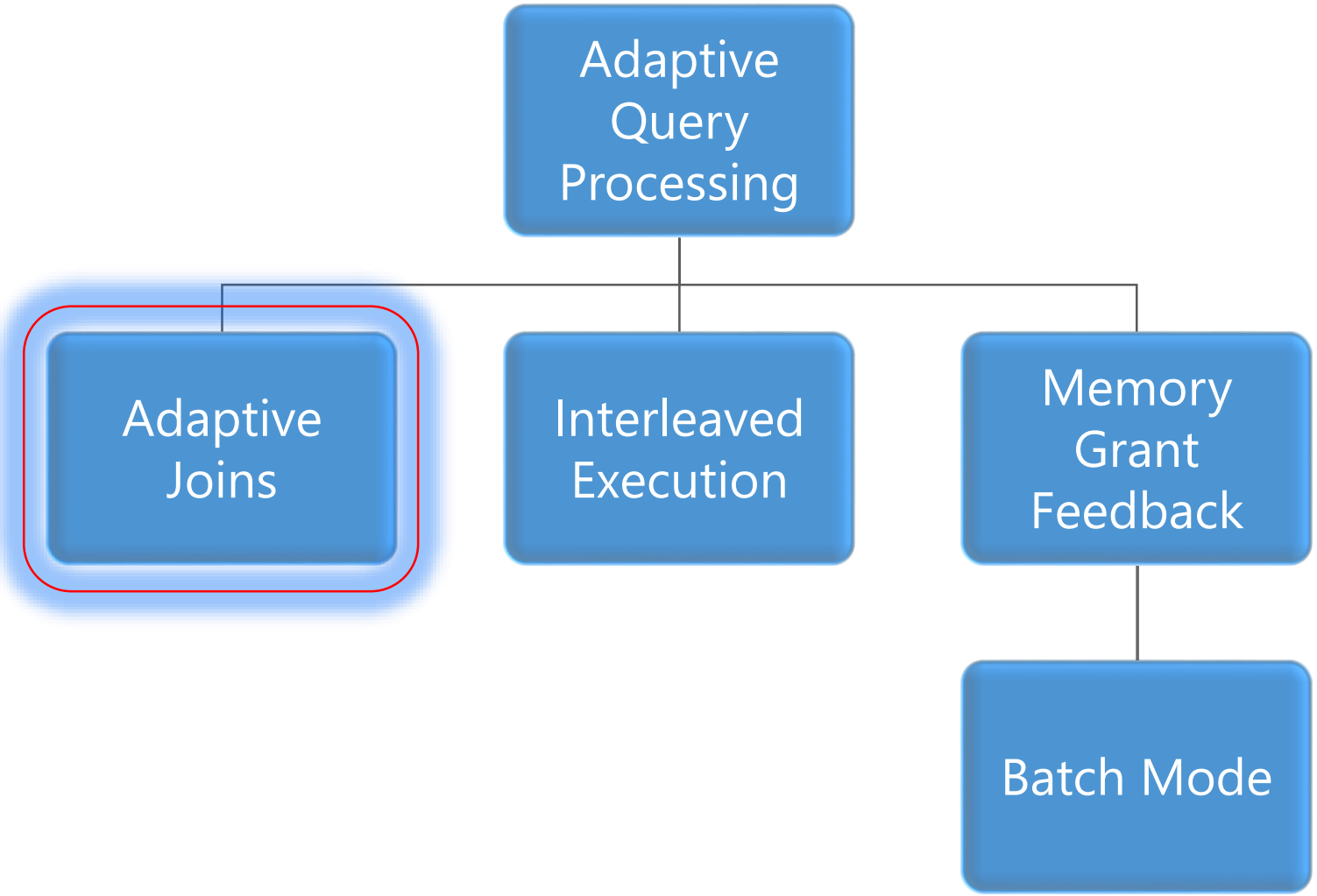
Adaptive Query Processing (2017)



Addresses performance issues related to the cardinality estimation of an execution plan.

These options can provide improved join type selection, row-calculations for Multi-Statement Table-Valued Functions, and memory allocation of row storage.

Batch Mode Adaptive Joins (2017)



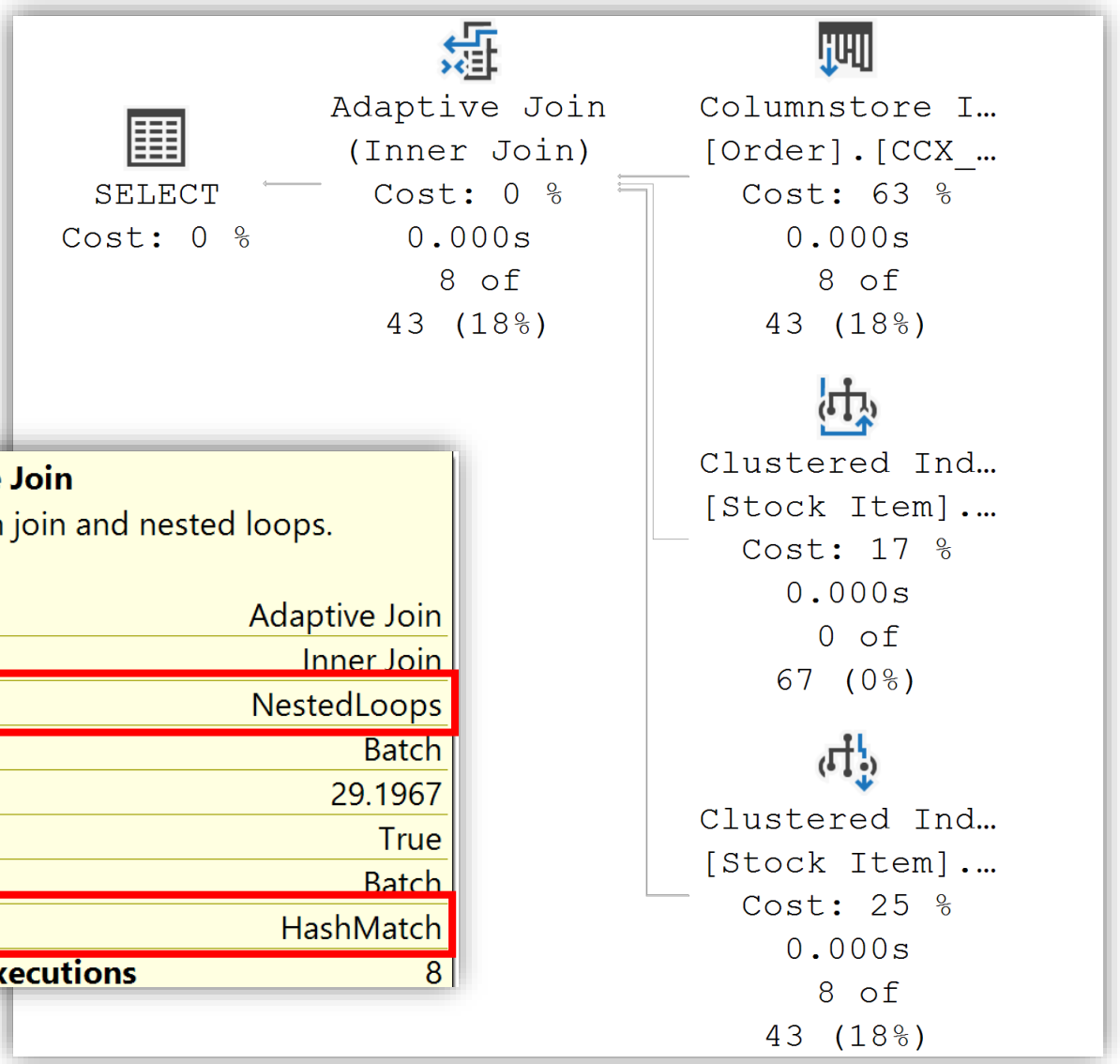
This feature enables the choice of either the Hash or the Nested Loop join type.

Decision is deferred until statement execution.

No need to use join hints in queries.

Batch Mode Adaptive Joins (2017)

Adaptive Joins

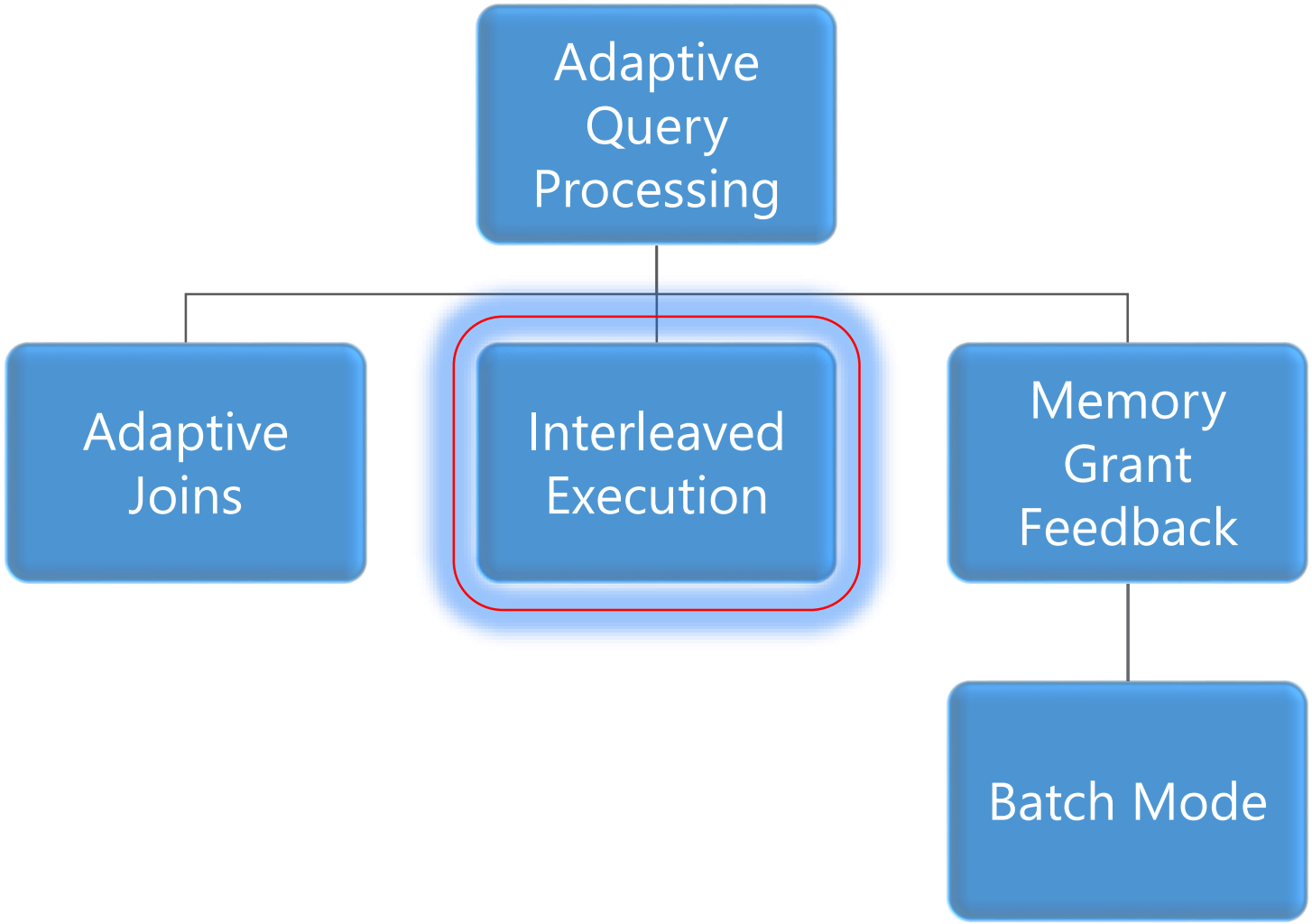


Adaptive Join

Chooses dynamically between hash join and nested loops.

Physical Operation	Adaptive Join
Logical Operation	Inner Join
Actual Join Type	NestedLoops
Actual Execution Mode	Batch
Adaptive Threshold Rows	29.1967
Is Adaptive	True
Estimated Execution Mode	Batch
Estimated Join Type	HashMatch
Actual Number of Rows for All Executions	8

Interleaved Execution (2017)



Previously, when a Multi-Statement Table-Valued Function was executed, it used a fixed row estimate of 100 rows.

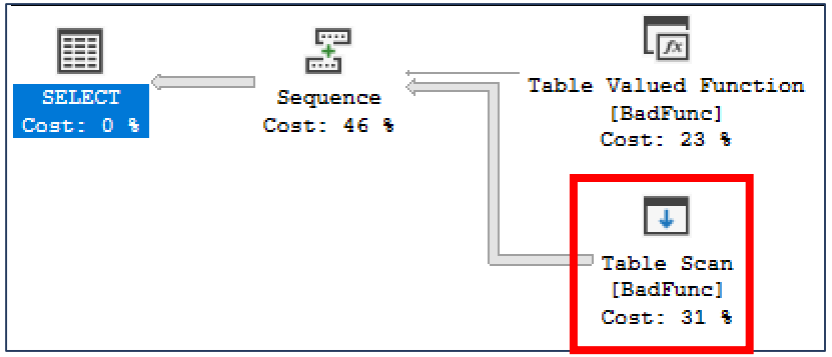
Now execution is paused so a better cardinality estimate can be captured.

Interleaved Execution (2017)

Interleaved Execution

Compatibility Level 120/130

Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	12345
Actual Number of Rows	12345
Actual Number of Batches	0
Estimated Operator Cost	0.003392 (92%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.000267
Estimated Subtree Cost	0.003392
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	100
Estimated Number of Rows	100
Estimated Row Size	67 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

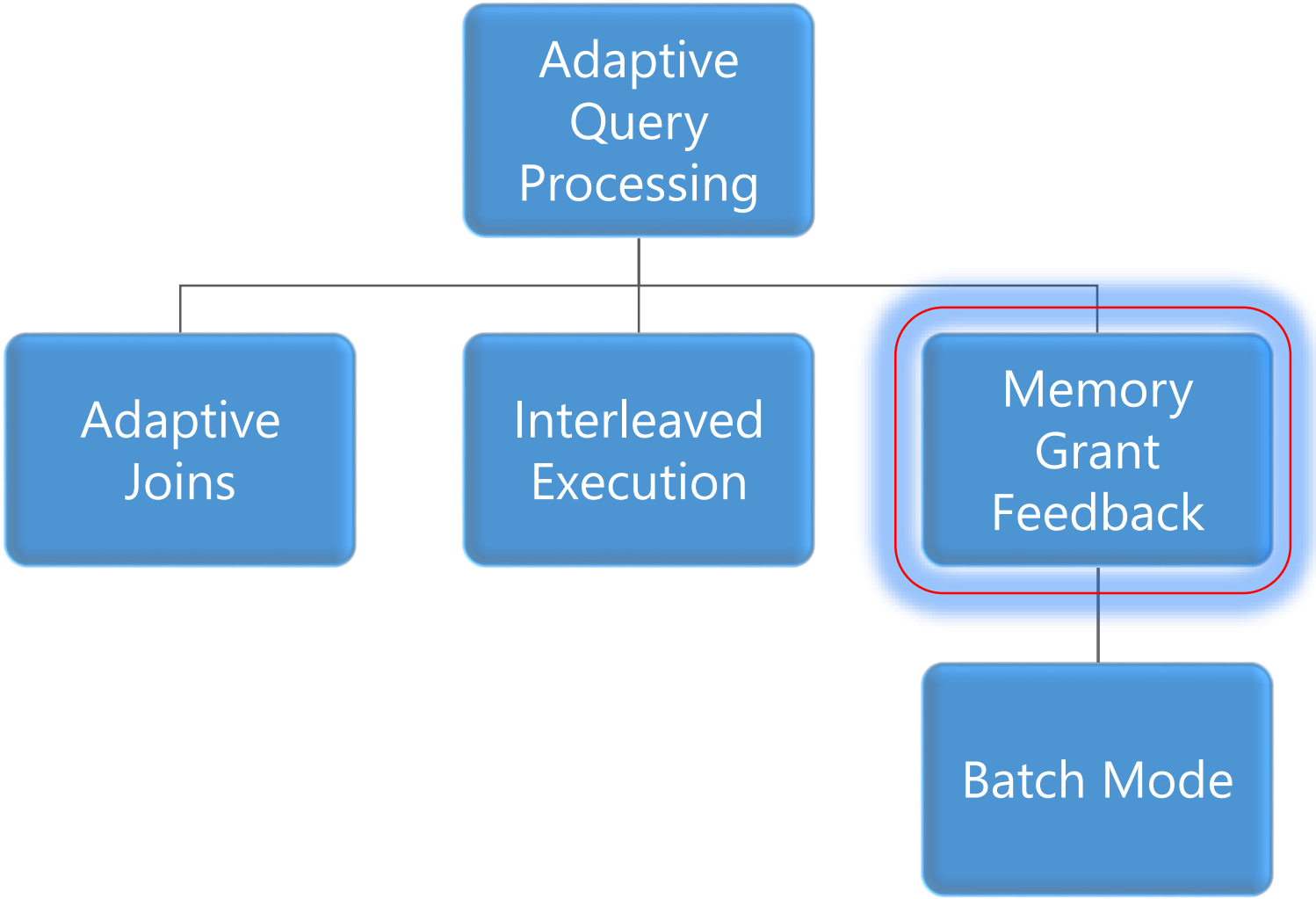


During optimization if SQL Server encounter a read-only multi-statement table-valued function (MSTVF), it will pause optimization, execute the applicable subtree, capture accurate cardinality estimates, and then resume optimization for downstream operations.

Compatibility Level 140 or higher

Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	12345
Actual Number of Rows	12345
Actual Number of Batches	0
Estimated Operator Cost	0.0168615 (31%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0137365
Estimated Subtree Cost	0.0168615
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	12345
Estimated Number of Rows	12345
Estimated Row Size	67 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

Batch Mode Memory Grant Feedback (2017)



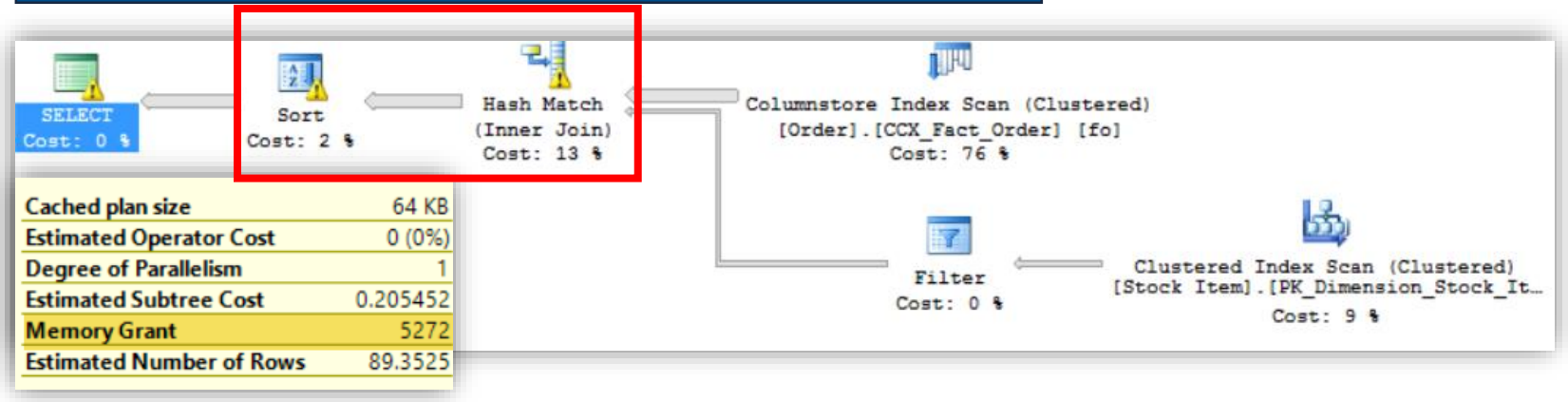
When compiling an execution plan, the query engine estimates how much memory is needed to store rows during join and sort operations.

Too much memory allocation may impact performance of other operations. Not enough will cause a spill over to disk.

This feature recalculates memory on first execution and updates the cached plan.

Batch Mode Memory Grant Feedback (2017)

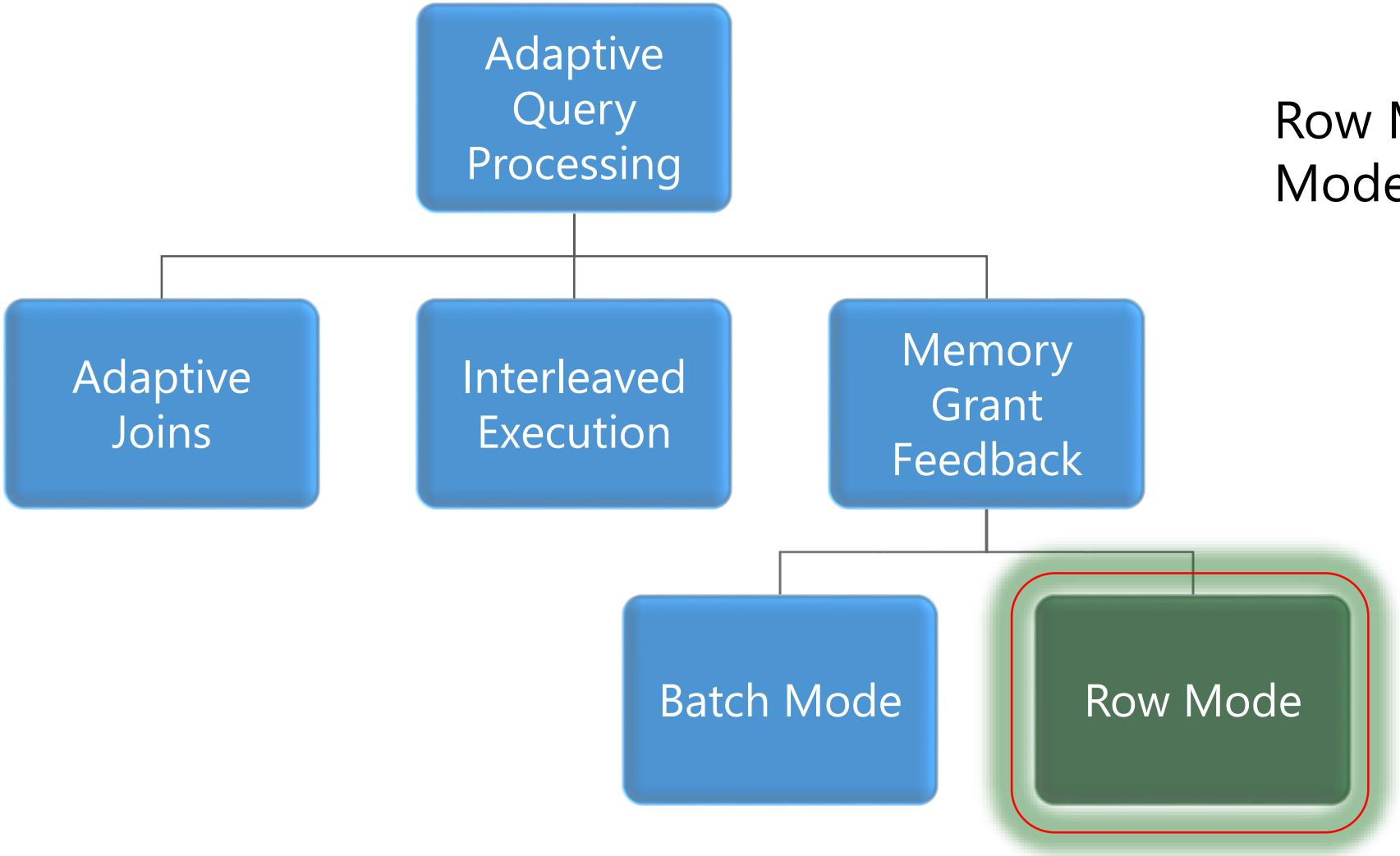
First Execution (Spills detected; feedback generated)



Second Execution (Memory grant adjusted)

Memory Grant Feedback (Batch Mode)

Row Mode Memory Grant Feedback (2019)



Row Mode is just like Batch Mode, but different.

Row Mode Memory Grant Feedback (2019)

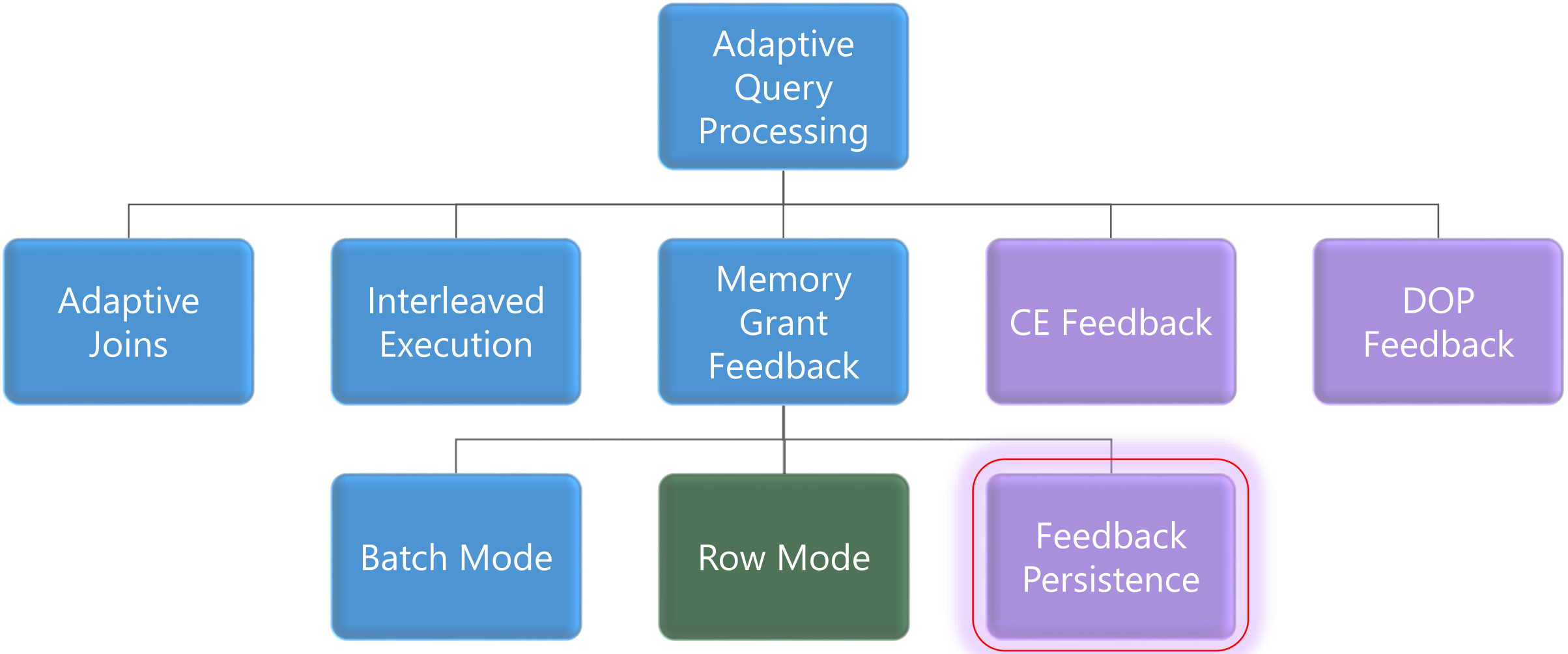
Expands on the batch mode memory grant feedback feature by also adjusting memory grant sizes for row mode operators.

MemoryGrantInfo	
DesiredMemory	13992
GrantedMemory	13992
GrantWaitTime	0
IsMemoryGrantFeedbackAdjusted	YesStable
LastRequestedMemory	13992
MaxQueryMemory	1497128
MaxUsedMemory	3744

Memory Grant Feedback (Row Mode)

Two new query plan attributes will be shown for actual post-execution plans.

Feedback Persistence (2022)



Feedback Persistence and Percentile (2022)

Problem: Cache Eviction

- Feedback is not persisted if the plan is evicted from cache or failover
- Record of how to adjust memory is lost and must re-learn

Solution: Persist the feedback

- Persist the memory grant feedback in the Query Store

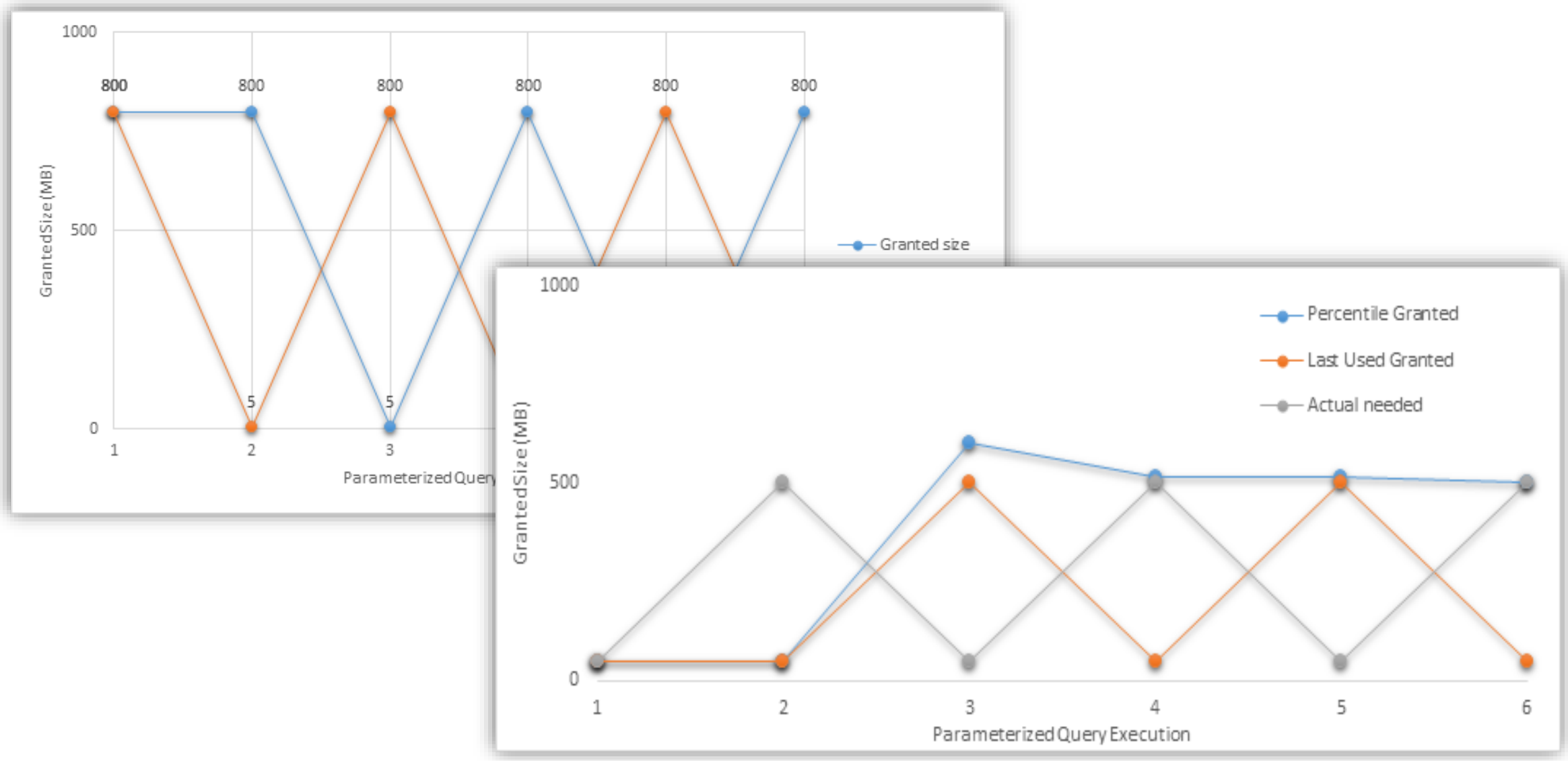
Problem: Oscillating Feedback

- Memory grants adjusted based on last feedback
- Parameter Sensitive Plans could change feedback

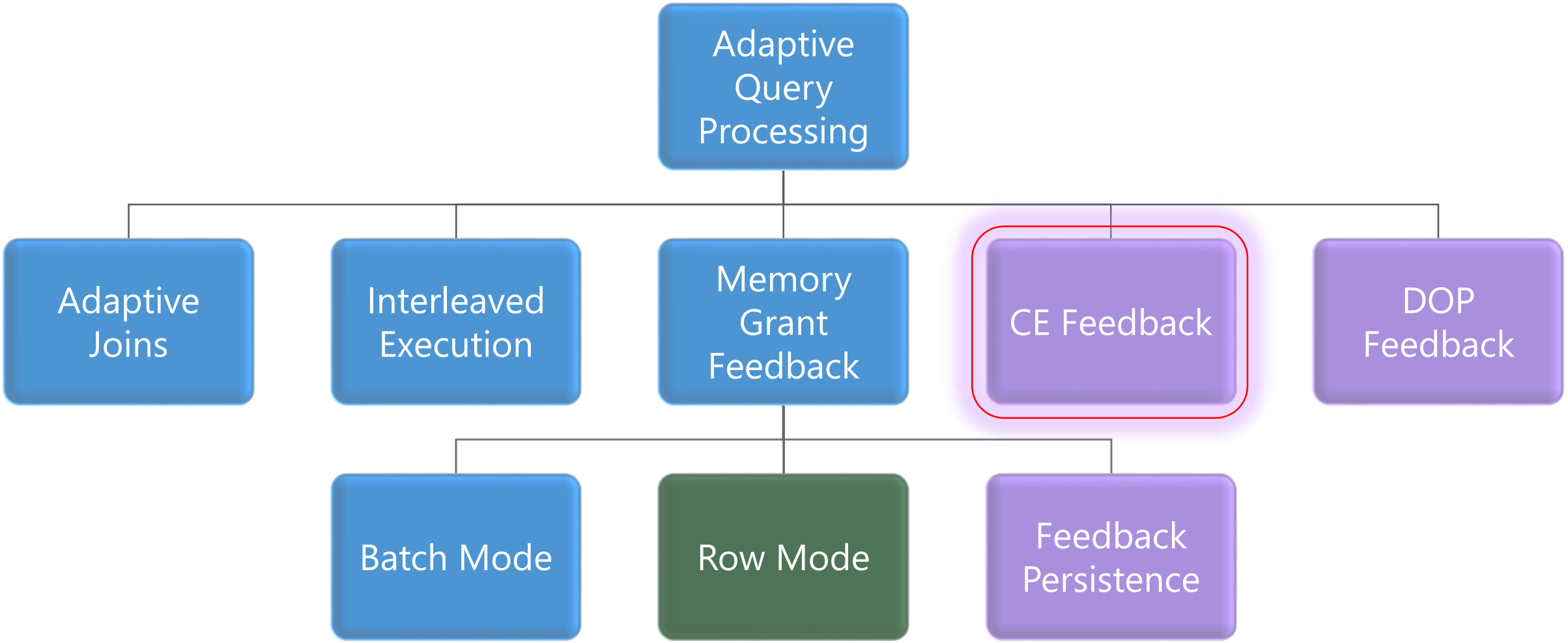
Solution: Percentile-based calculation

- Smooths the grant size values based on execution usage history

Feedback Persistence and Percentile (2022)



Cardinality Estimator Feedback (2022)



Cardinality Estimator Feedback (2022)

Cardinality Estimation Today

- CE determines the estimated number of rows for a query plan
- CE models are based on statistics and assumptions about the distribution of data
- Learn more about CE models and assumptions https://aka.ms/sqlCE

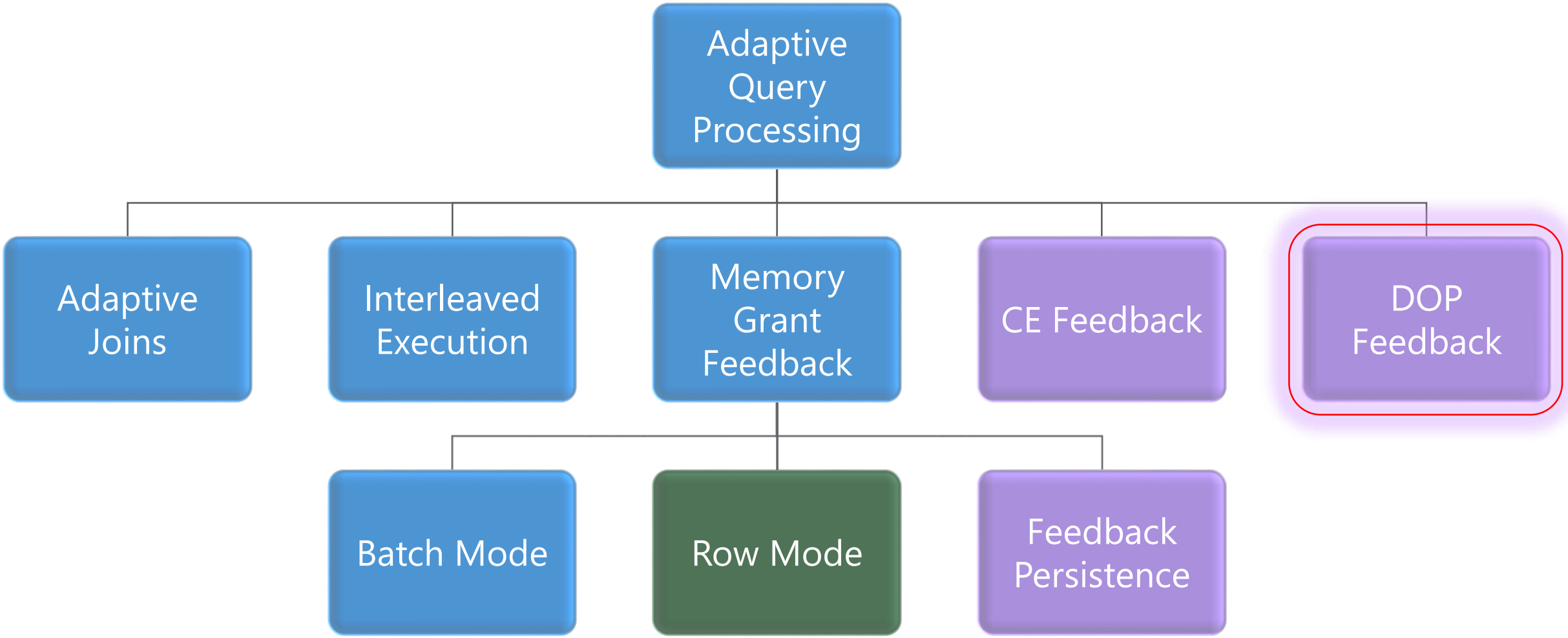
Problem: Incorrect Assumptions for Cardinality Estimates

- The cardinality estimator sometimes makes incorrect assumptions
- Poor assumptions leads to poor query plans.
- One CE models doesn't fit all scenarios

Solution: Learn from historical CE model assumptions

- CE Feedback will evaluate accuracy for repeated queries
- If assumption looks incorrect, test a different CE model assumption and verify if it helps
- If a CE model assumption does help, it will replace the current plan in cache.

Degree of Parallelism Feedback (2022)



Degree of Parallelism Feedback (2022)

Parallelism Today

- Parallelism is often beneficial for querying large amounts of data, but transactional queries could suffer when time spent coordinating threads outweighs the advantages of using a parallel plan

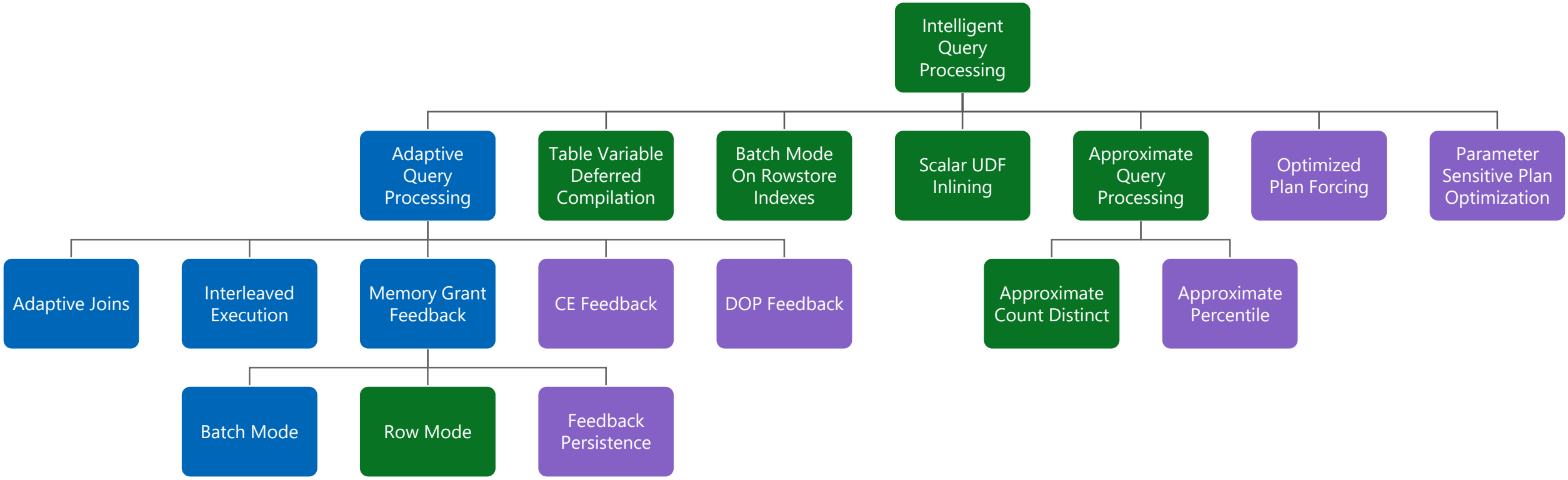
Current Settings

- Before SQL Server 2019, default value for MAXDOP = 0
- With SQL Server 2019, default is calculated at setup based on available processors
- Azure SQL Database the default MAXDOP is 8

DOP Feedback

- DOP Feedback will **identify** parallelism inefficiencies for repeating queries, based on CPU time, elapsed time, and waits
- If parallelism usage is inefficient, the DOP will be **lowered** for next execution (min DOP = 2) and then **verify** if it helps
- Only verified feedback is persisted (Query Store).
  - If next execution regresses, back to last good known DOP

Intelligent Query Processing (2022)



Azure SQL Database

2017 2019 2022

http://aka.ms/IQP

Intelligent Query Processing (2019 Features)

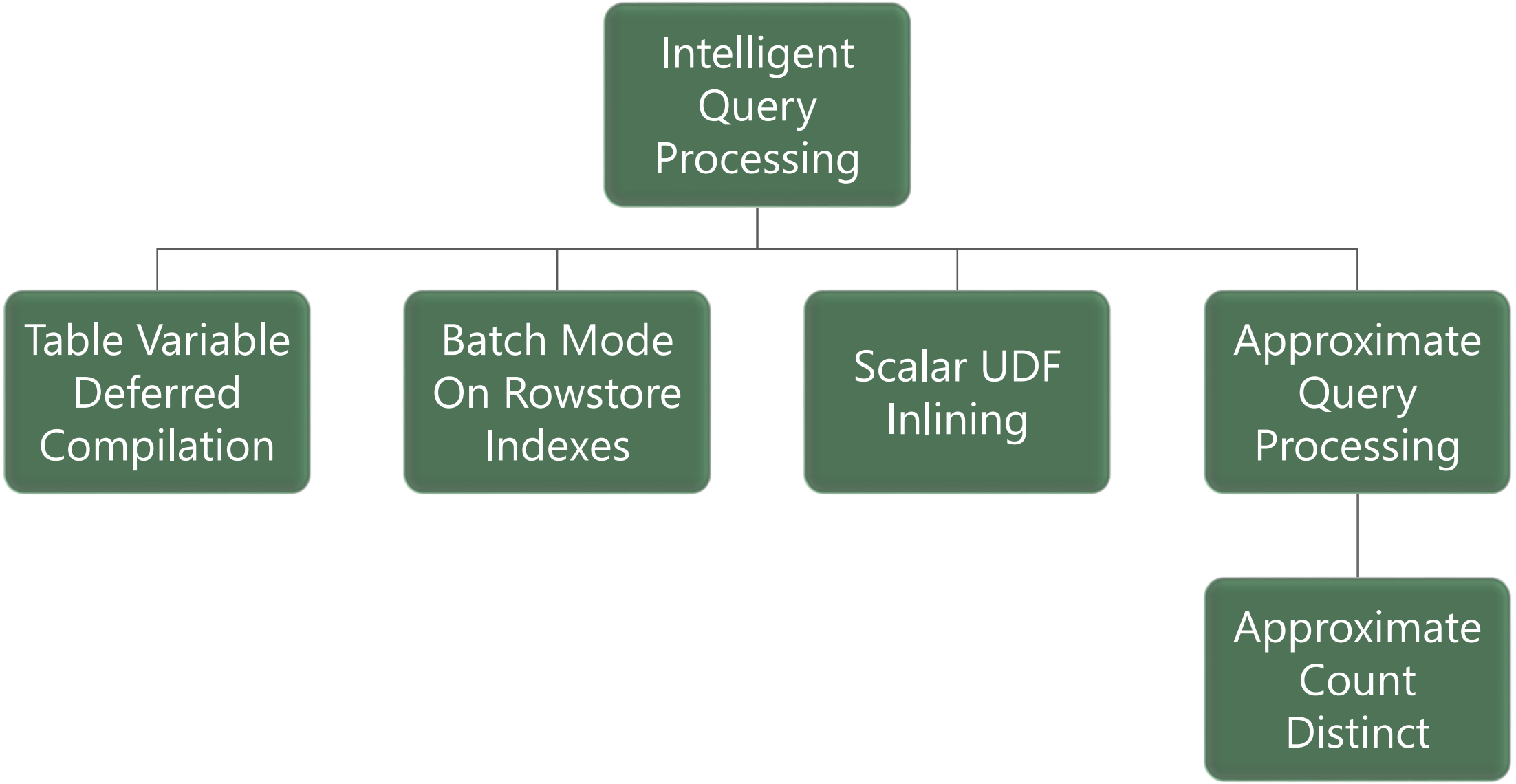
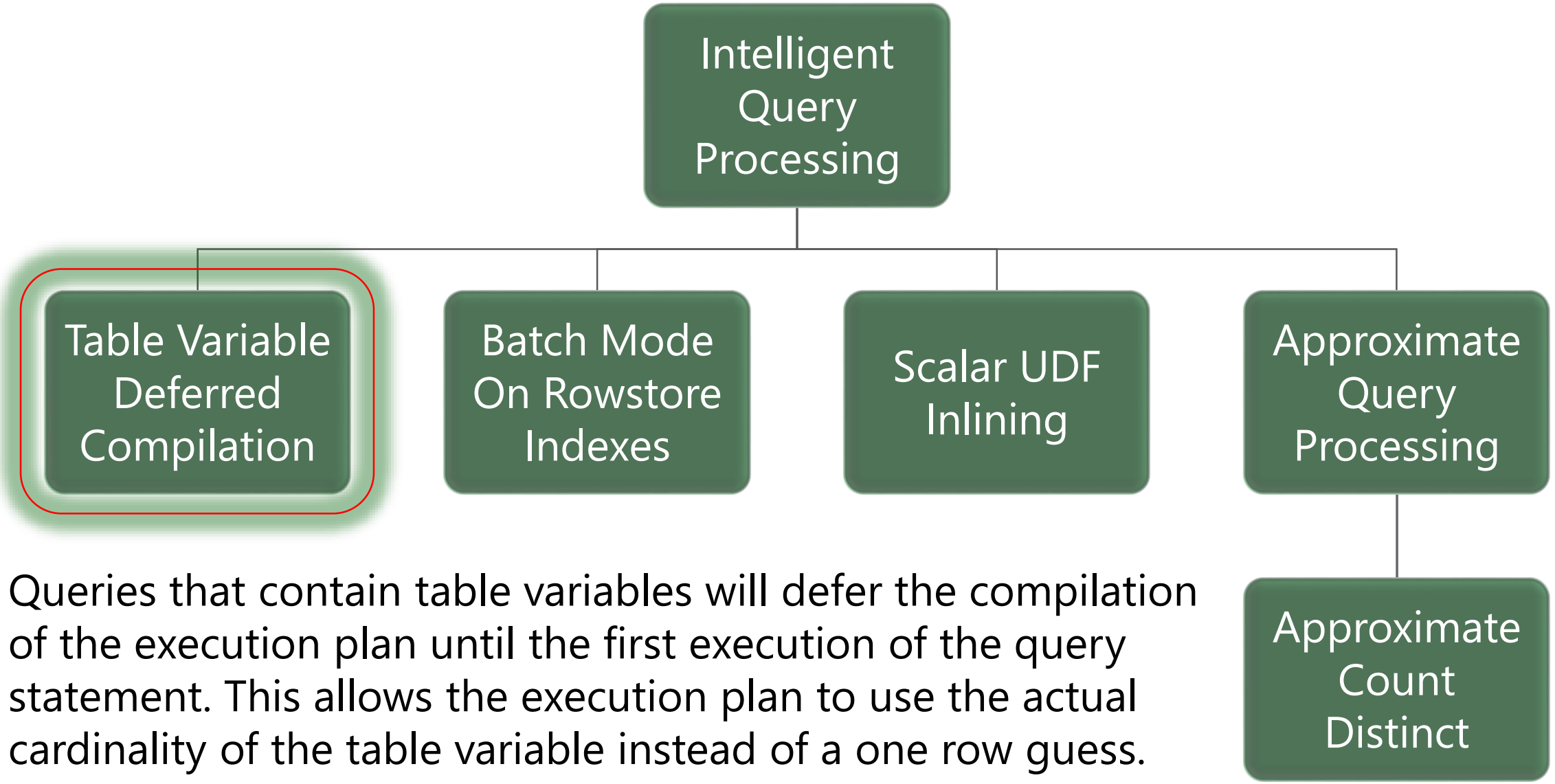
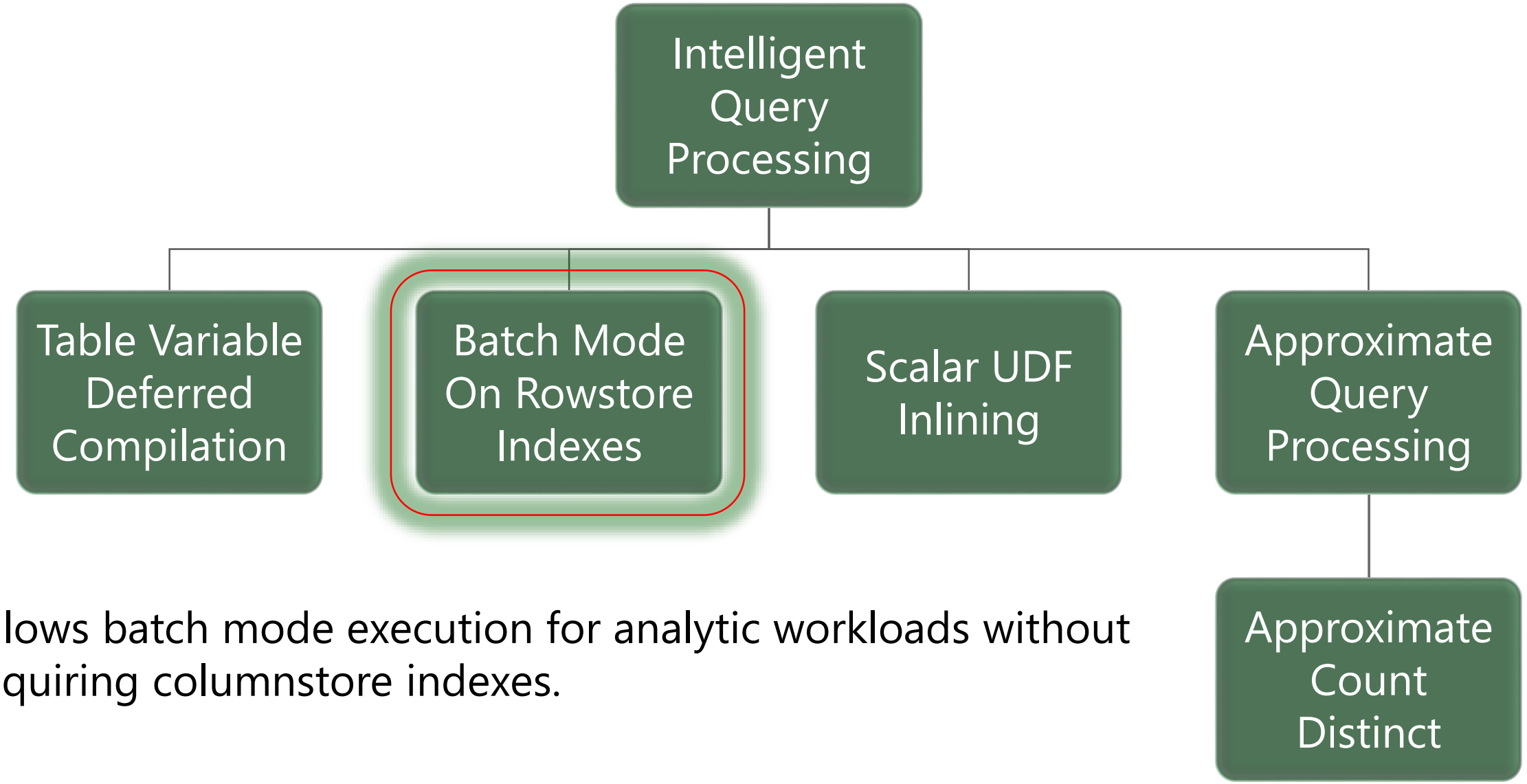


Table Variable Deferred Compilation



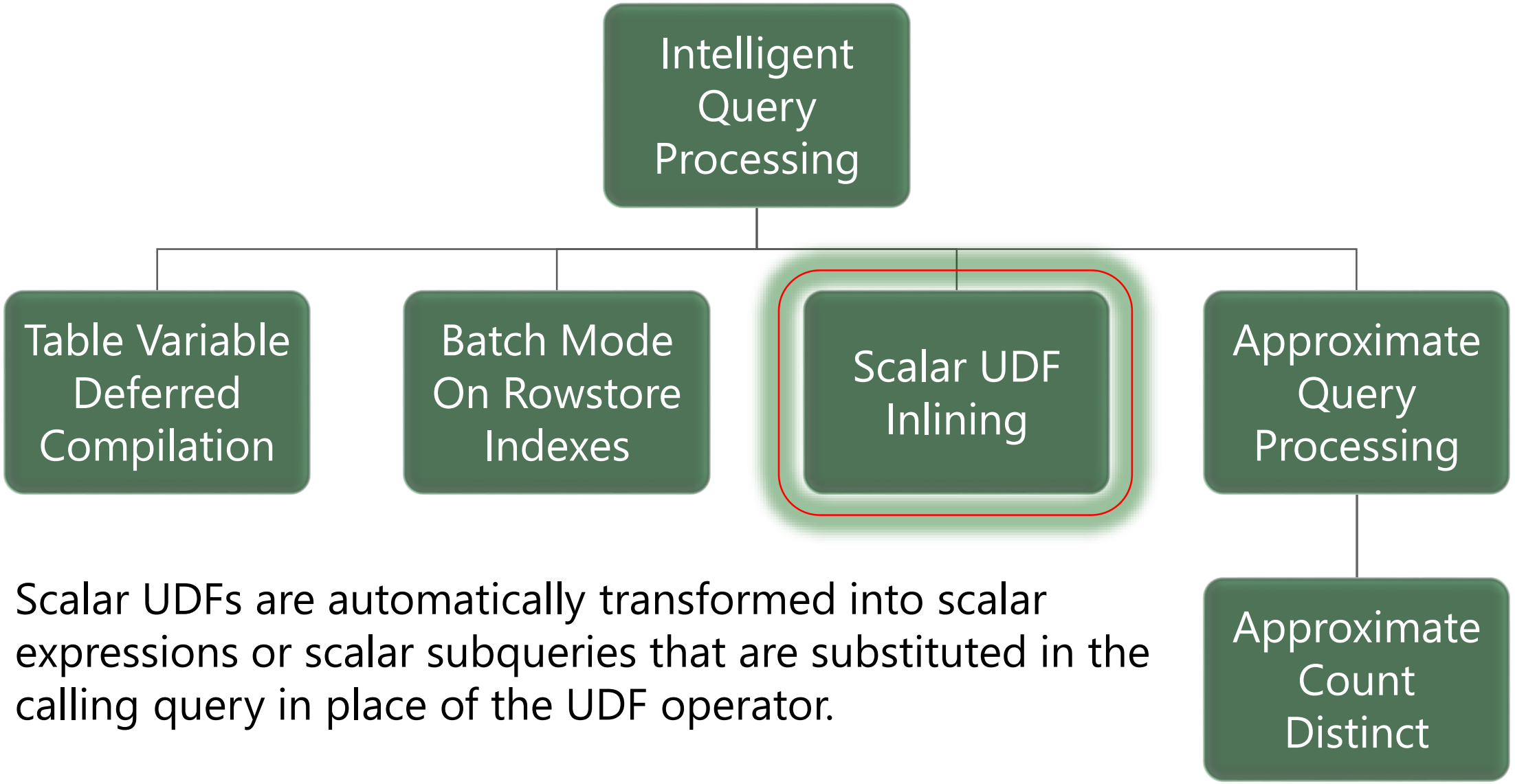
Queries that contain table variables will defer the compilation of the execution plan until the first execution of the query statement. This allows the execution plan to use the actual cardinality of the table variable instead of a one row guess.

Batch Mode on Rowstore Indexes



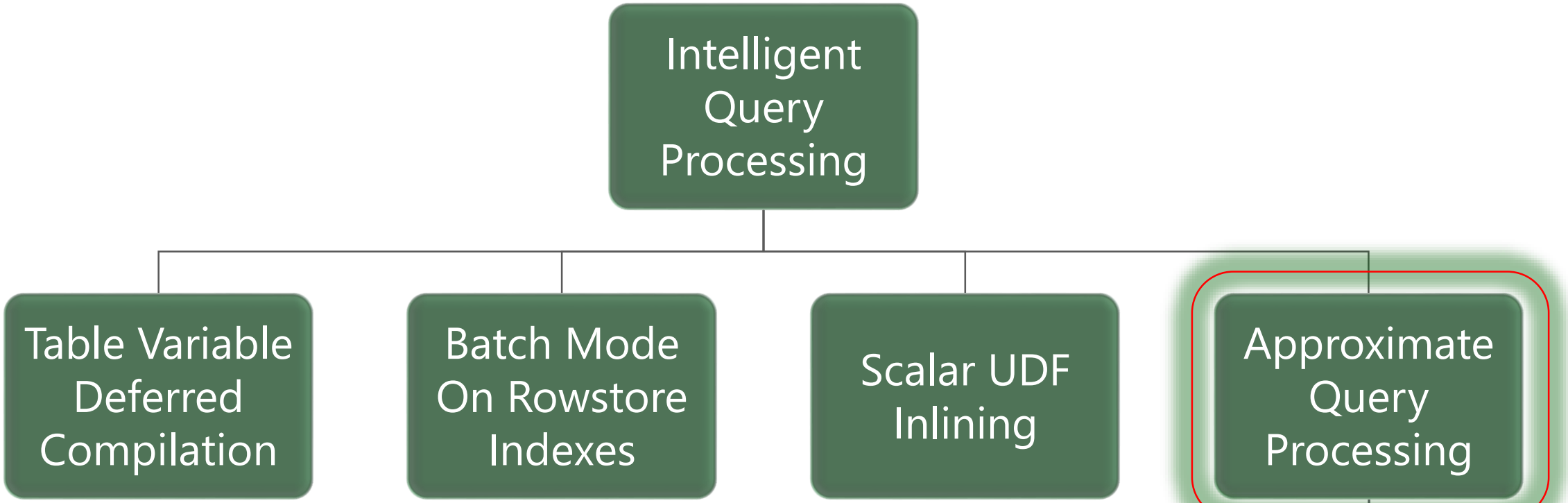
Allows batch mode execution for analytic workloads without requiring columnstore indexes.

Scalar User-Defined Function Inlining



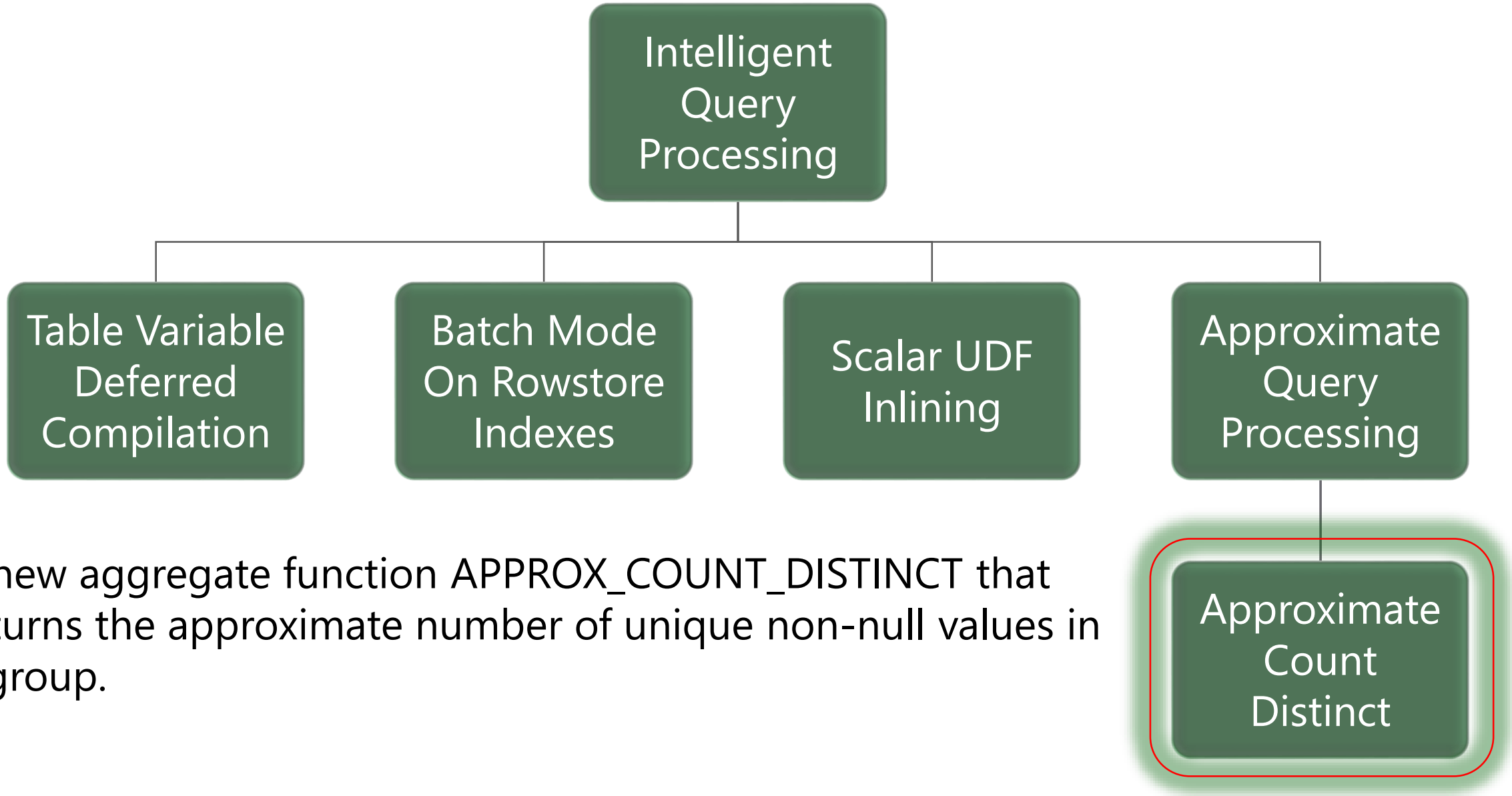
Scalar UDFs are automatically transformed into scalar expressions or scalar subqueries that are substituted in the calling query in place of the UDF operator.

Approximate Query Processing



Approximate Query Processing is a new feature family that will be used to aggregate large datasets where performance is more critical than absolute precision.

Approximate Count Distinct (2019)



A new aggregate function APPROX_COUNT_DISTINCT that returns the approximate number of unique non-null values in a group.

Approximate Count Distinct

It returns the approximate number of unique non-null values in a group.

It is designed to provide aggregations across large data sets where responsiveness is more critical than absolute precision.

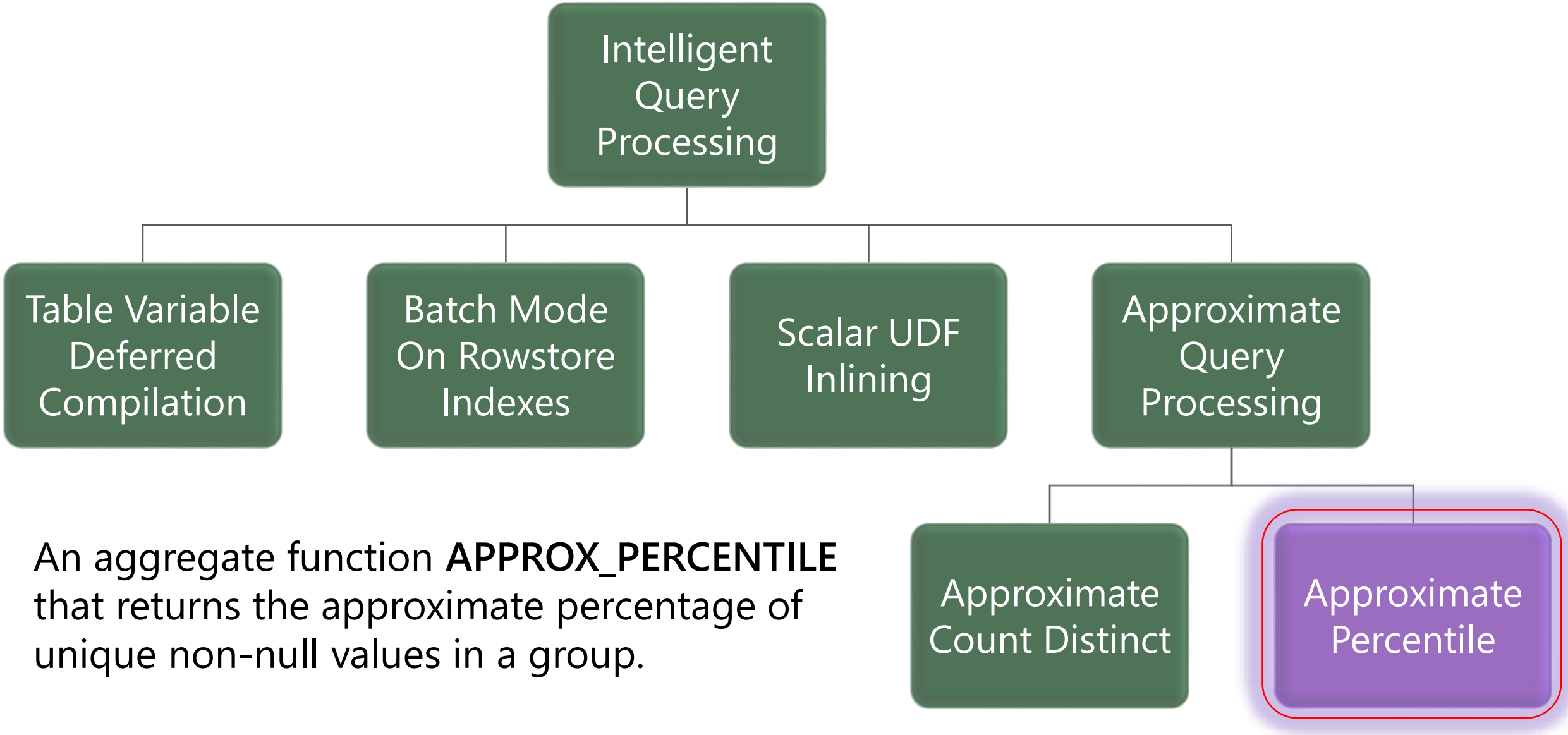
Guarantees up to a 2% error rate within a 97% probability.

Requires less memory than an exhaustive COUNT DISTINCT operation so it is less likely to spill memory to disk compared to COUNT DISTINCT.

Approximate Count Distinct

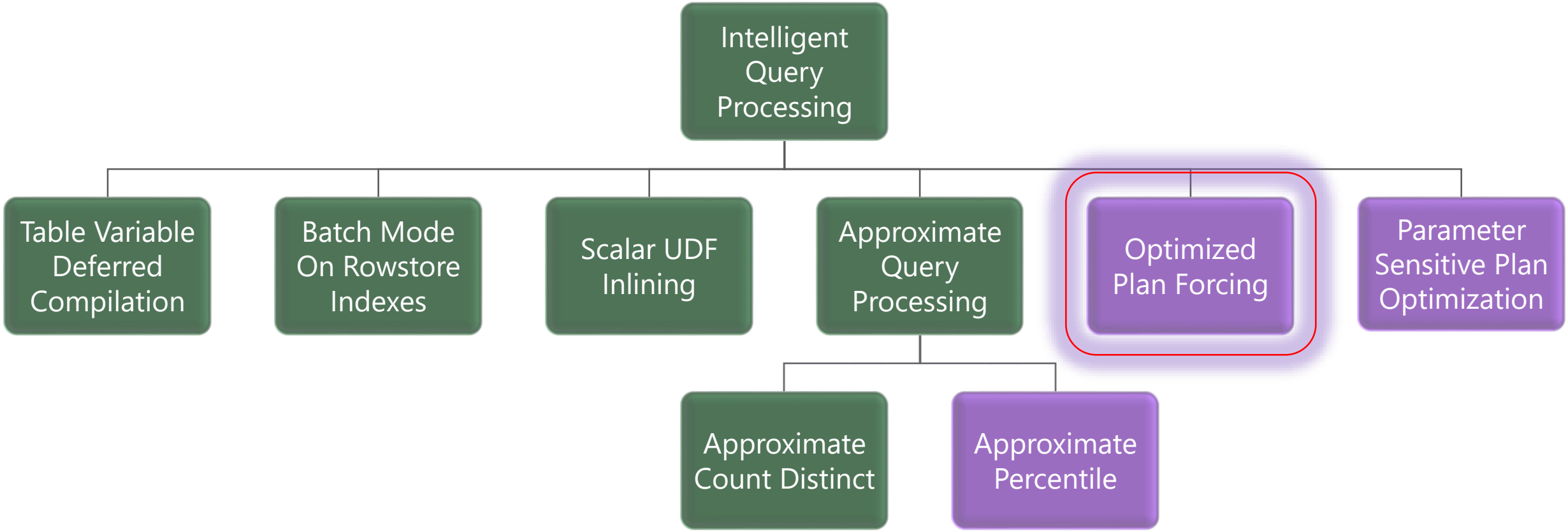
```
SELECT APPROX_COUNT_DISTINCT(O_OrderKey) AS Approx_Distinct_OrderKey
FROM dbo.Orders;
```

Approximate Percentile (2022)



An aggregate function **APPROX_PERCENTILE** that returns the approximate percentage of unique non-null values in a group.

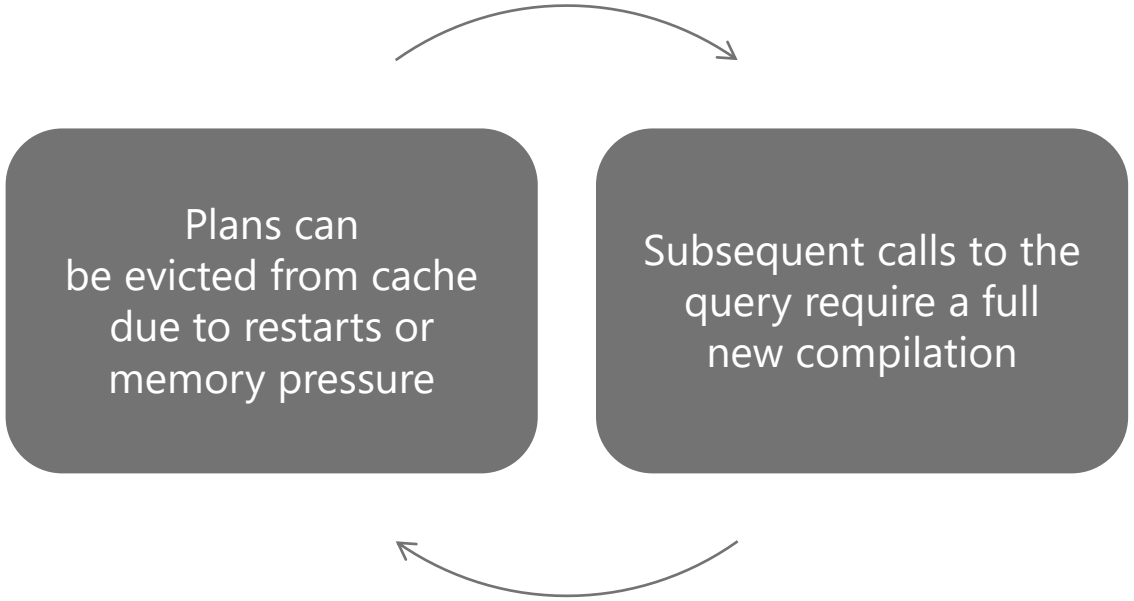
Optimized Plan Forcing (2022)



Optimized Plan Forcing (2022)

Query Compilation Today

- Query optimization and compilation is a multi-phased process of quickly generating a “good-enough” query execution plan
- Query execution time includes compilation. Can be time and resource consuming
- To reduce compilation overhead for repeating queries, SQL caches query plans for re-use

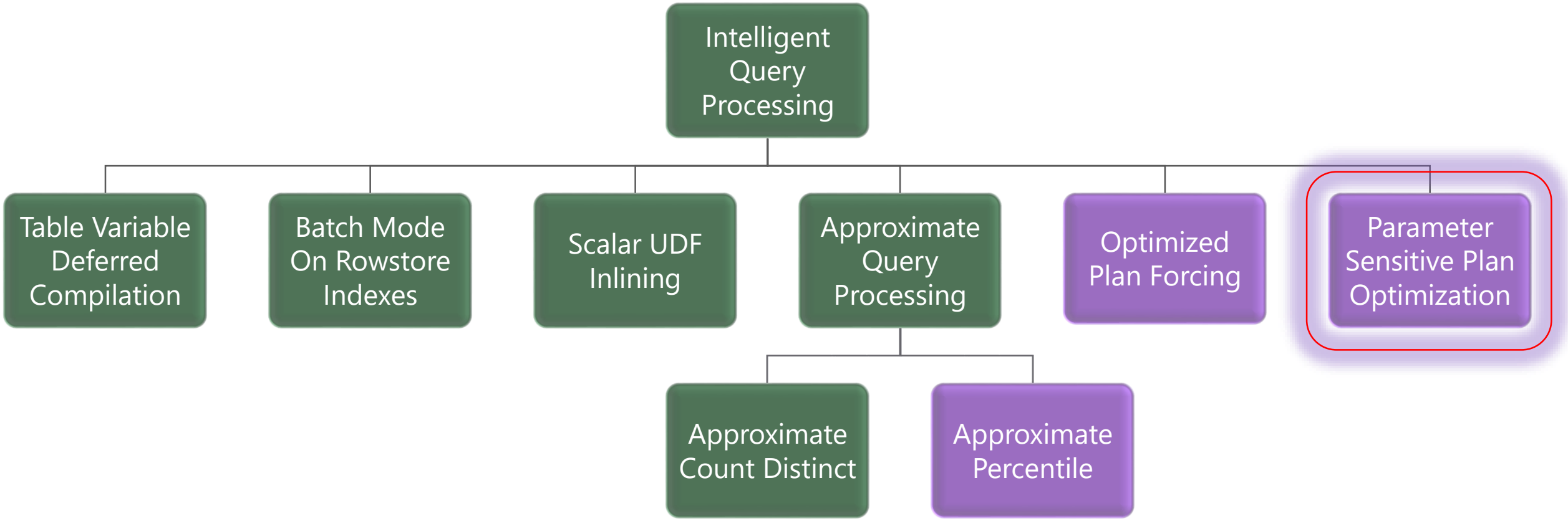


Optimized Plan Forcing (2022)

Query Compilation Replay

- Stores a *compilation replay script* (CRS) that persists key compilation steps in Query Store (not user visible)
- Version 1 targets previously forced plans through Query Store and Automatic Plan Correction
- Uses those previously-recorded CRS to quickly reproduce and cache the original forced plan **at a fraction of the original compilation cost**
- Compatible with Query Store hints and secondary replica support

Parameter Sensitive Plan Optimization (2022)



Parameter Sensitive Plans (2022)

Parameter Sensitive Plans Today

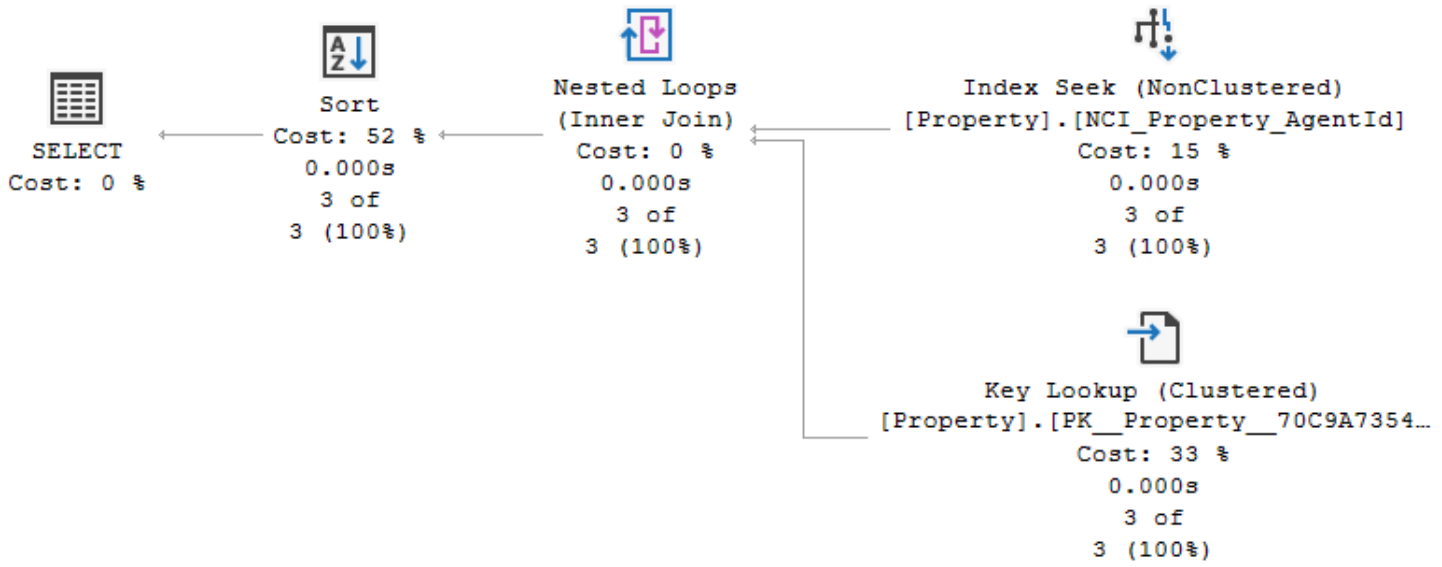
- Parameter-sniffing problem refers to a scenario where a **single** cached plan for a parameterized query is **not optimal for all** possible input parameter values
- If plan is not representative of most executions, you have a perceived “bad plan”

Current Workarounds

- RECOMPILE
- OPTION (OPTIMIZE FOR…)
- OPTION (OPTIMIZE FOR UNKNOWN)
- Disable parameter sniffing entirely
- KEEPFIXEDPLAN
- Force a known plan
- Nested procedures
- Dynamic string execution

PSP today (Example of Real Estate agent's portfolio)

New compile on Agent 4

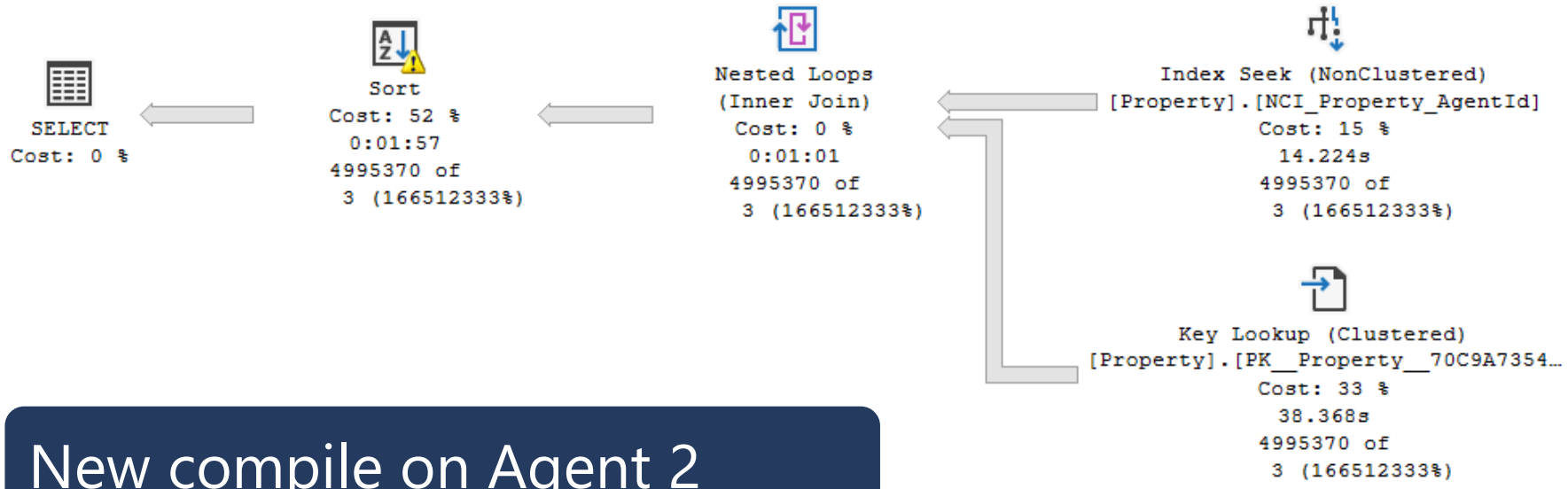


QueryTimeStats	
CpuTime	0
ElapsedTime	0

This example was borrowed from Pedro Lopes @SQLPedro

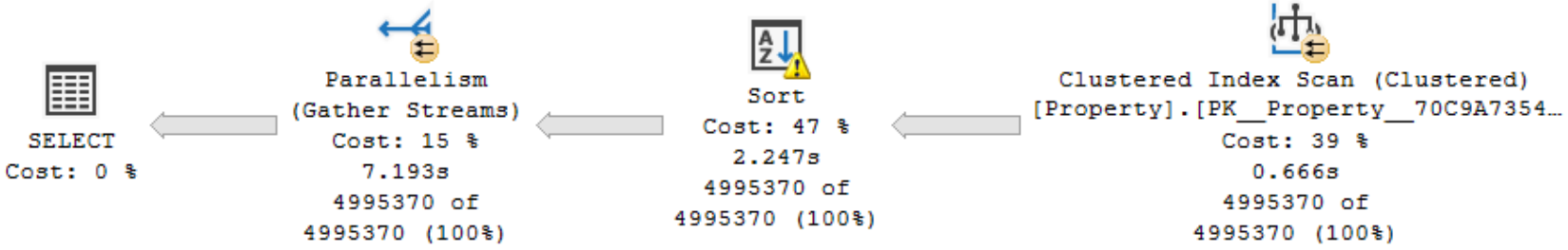
PSP today (Example of Real Estate agent's portfolio)

Using cached plan for Agent 2



QueryTimeStats	
CpuTime	88667
ElapsedTime	214222

New compile on Agent 2



QueryTimeStats	
CpuTime	46620
ElapsedTime	105288

PSP Optimization (2022)

Automatically enables multiple, active cached plans for a single parameterized statement

Cached execution plans will accommodate different data sizes based on the customer-provided runtime parameter value(s)

Design considerations

- Too many plans generated could create cache bloat, so limit # of plans in cache
- Overhead of PSP optimization must not outweigh downstream benefit
- Compatible with Query Store plan forcing

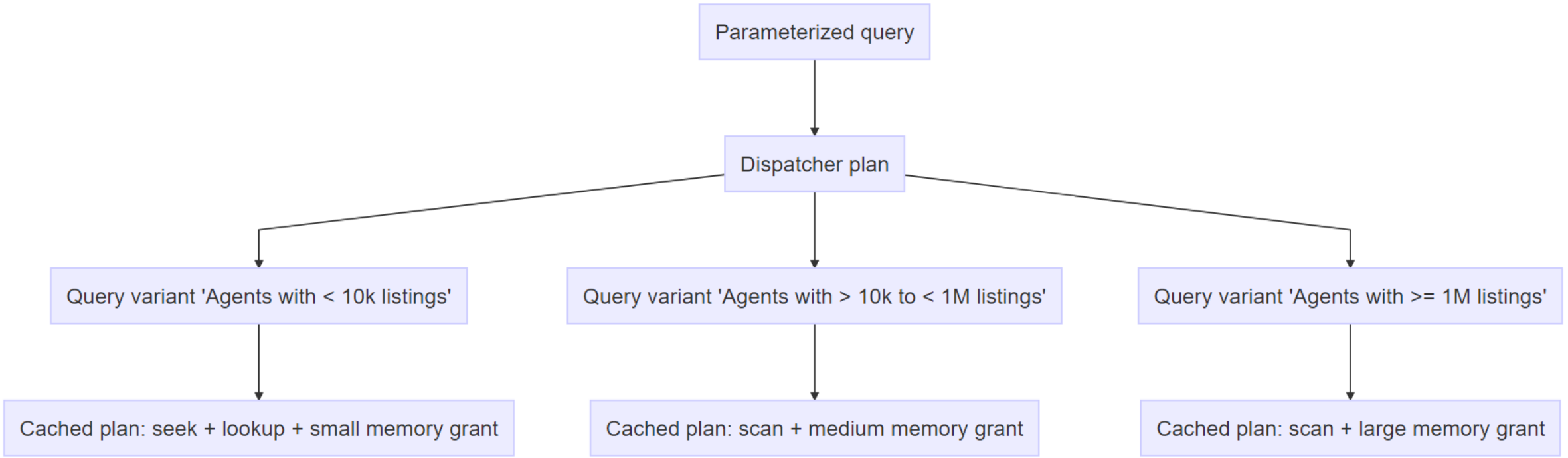
PSP Predicate Selection (2022)

During initial compilation PSP optimization will evaluate the most "at risk" parameterized predicates (up to three out of all available)

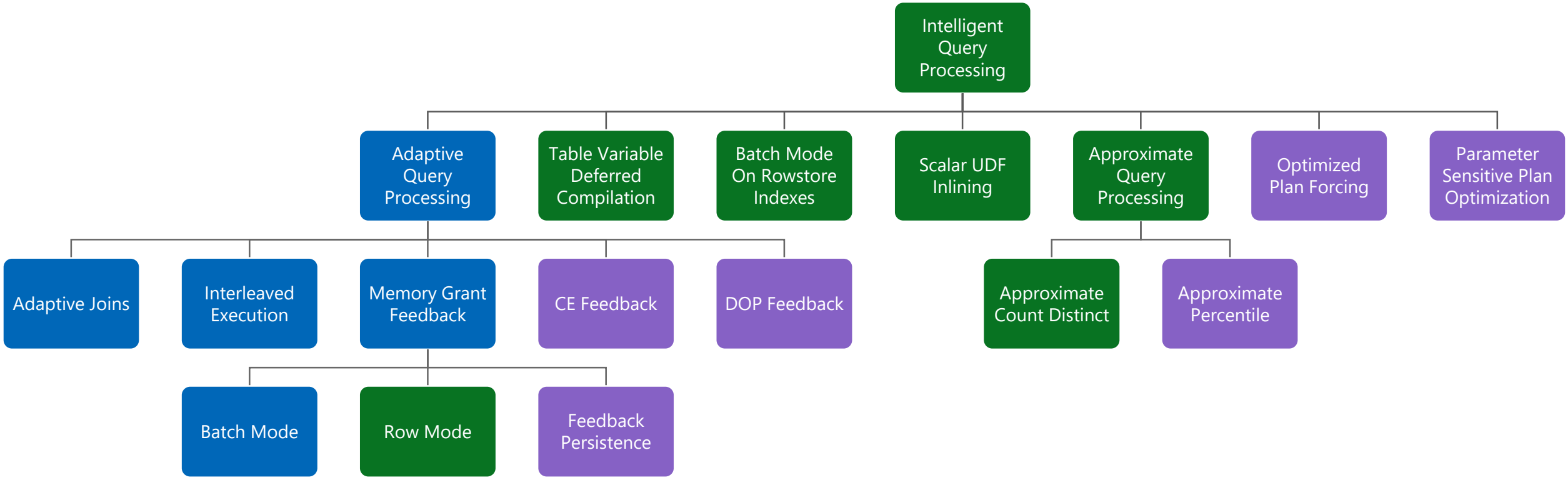
First version is scoped to equality predicates referencing statistics-covered columns; `WHERE AgentId = @AgentId`

Uses the statistics histogram to identify non-uniform distributions

Boundary Value Selection (Dispatcher Plan)



Intelligent Query Processing (2022)



http://aka.ms/IQP

Azure SQL Database

2017 2019 2022

Demonstration

Intelligent query processing

- Interleaved Execution
- Batch Mode on RowStore
- Memory Grant Feedback (Row Mode)

Intelligent query processing

- Observing Batch-Mode Memory Grant Feedback
- Using APPROX_COUNT_DISTINCT to improve performance
- Observing Table Variable Deferred Compilation
- Observing Scalar UDF Inlining

Questions?

Knowledge Check

Is it possible to disable Intelligent Query Processing features?

On queries not using Interleaved execution for MSTVFs. How many rows are estimated for a MSTVFs?

Does table variable deferred compilation increase the recompilation frequency?

On queries not using table variable deferred compilation. How many rows are estimated for a table variable?

What is the minimum compatibility level that supports Batch mode on rowstore?